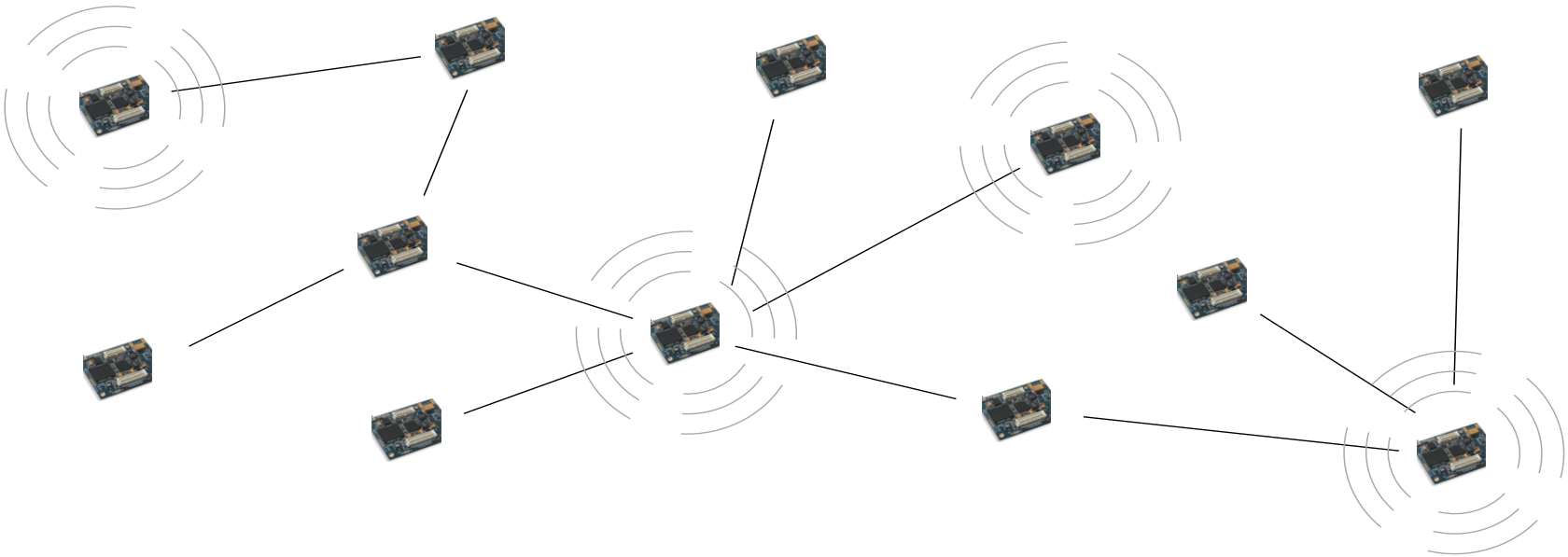
Karlsruher Institut für Technologie

# Algorithmen für Ad-hoc- und Sensornetze

## VL 10 – Kapazität, Scheduling und Färbungen

Fabian Fuchs | 10. Dez. 2015 (Version 1)

INSTITUT FÜR THEORETISCHE INFORMATIK - LEHRSTUHL FÜR ALGORITHMIK (PROF. WAGNER)



# Motivation

Wenn man Übertragungen optimal zeitlich plant, kann man den Kanal besser nutzen! Was kann man noch erreichen, wenn man beachtet, dass man nicht beliebige Übertragungen parallel ausführen kann?

Weitere Fragen:

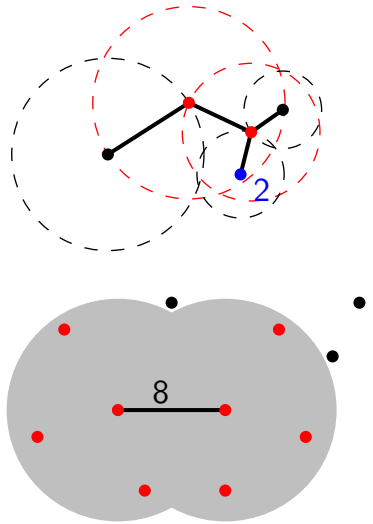
- Wie schnell kann man eine Aufgabe bestenfalls lösen?
- Was für Datenraten kann man maximal erreichen?
- Was sind realistischere Modelle für Interferenz?

Wir werden nur einen ganz, ganz kurzen Blick auf einen riesigen Komplex von Fragen werfen...

# Bisherige Sichtweisen auf Interferenz

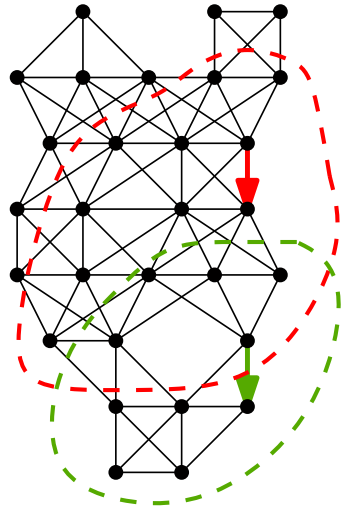
Wir haben Interferenz schon völlig unterschiedlich modelliert...

- Bei der Topologiekontrolle:
  - Knoten stören alle Knoten, die ihnen näher sind als der am weitesten entfernte Kommunikationspartner
  - Übertragungen stören “Kanten”



# Bisherige Sichtweisen auf Interferenz

Wir haben Interferenz schon völlig unterschiedlich modelliert...



- Bei der Topologiekontrolle:
  - Knoten stören alle Knoten, die ihnen näher sind als der am weitesten entfernte Kommunikationspartner
  - Übertragungen stören “Kanten”
- Data Gathering: Empfänger dürfen keinen störenden Sender in $k$-Hop-Nachbarschaft haben

# Bisherige Sichtweisen auf Interferenz

Wir haben Interferenz schon völlig unterschiedlich modelliert...

- Bei der Topologiekontrolle:
  - Knoten stören alle Knoten, die ihnen näher sind als der am weitesten entfernte Kommunikationspartner
  - Übertragungen stören “Kanten”
- Data Gathering: Empfänger dürfen keinen störenden Sender in $k$-Hop-Nachbarschaft haben

Einfache Modelle helfen, verteilte Algorithmen zu betrachten, aber wo wollen wir eigentlich hin?

# Kommunikationsmodelle

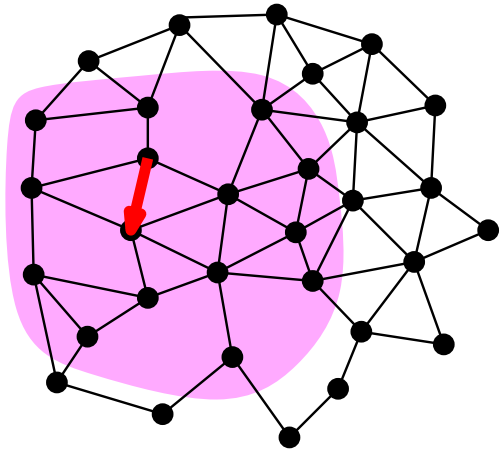
- Verbindungsmodelle: UDG, QUDG, BIG
  - Annahmen darüber, welche Struktur der Verbindungsgraph hat
  - setzt voraus, dass Knoten kommunizieren können oder eben nicht (keine Empfangswahrscheinlichkeiten)
  - Grundlage für angepasste verteilte Algorithmen
  - viele Lösungen ignorieren Interferenz völlig!

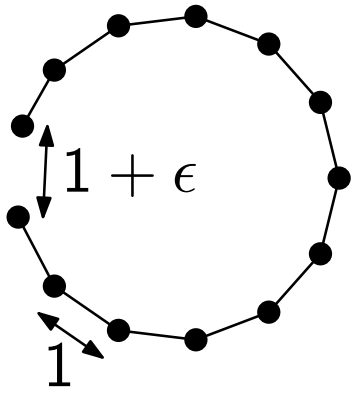
- Interferenzmodelle: ... (ein paar kommen gleich!)
  - formalisieren, unter welchen Voraussetzungen parallele Übertragungen erfolgreich sind
  - oft sehr stark vereinfacht & binär (?)
  - Kombination von komplexen Modellen und komplexen Problemen immer noch selten

# Erinnerung: Hop-Interferenz

## Hop-Interferenzmodell

Im *Hop-Interferenzmodell* kann ein Empfänger $r$ eine Nachricht eines Senders $s$ empfangen, wenn er im Verbindungsgraphen mit $s$ verbunden ist und es keinen anderen Sender $s'$ in der $k$-Hop-Nachbarschaft von $r$ gibt.





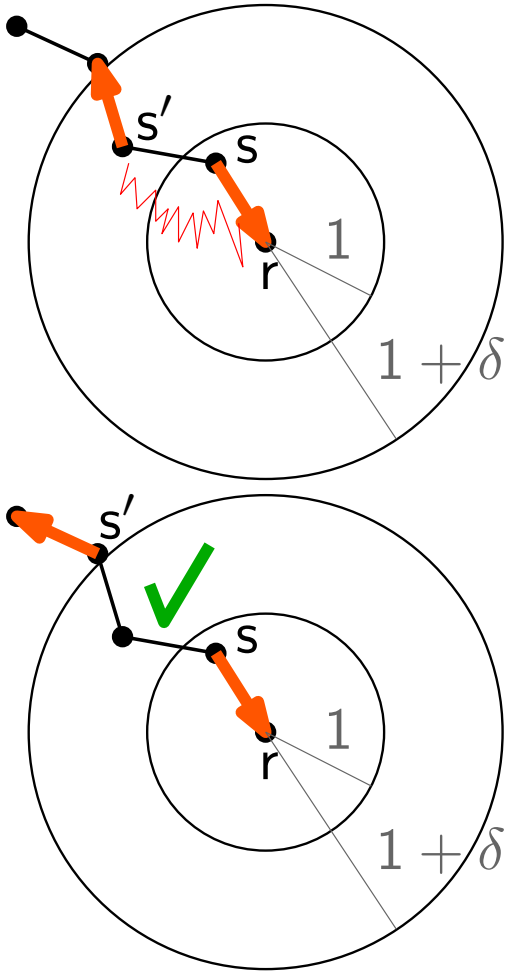

- Typischerweise kombiniert mit UDG
- \+ schön einfach
- \+ verteilte Algorithmen möglich
- \- dafür ist nicht alle Interferenz zwischen dichten Knoten abgedeckt!

# Protokollmodell (einfach)

## Protokollmodell (einfach)

Im vereinfachten *Protokollmodell* kann ein Empfänger $r$ eine Nachricht eines Senders $s$ empfangen, wenn $|r - s| \leq 1$ und es keinen anderen Sender $s'$ mit $|r - s'| \leq (1 + \delta)$ gibt.

- oft einfach $\delta = 1$

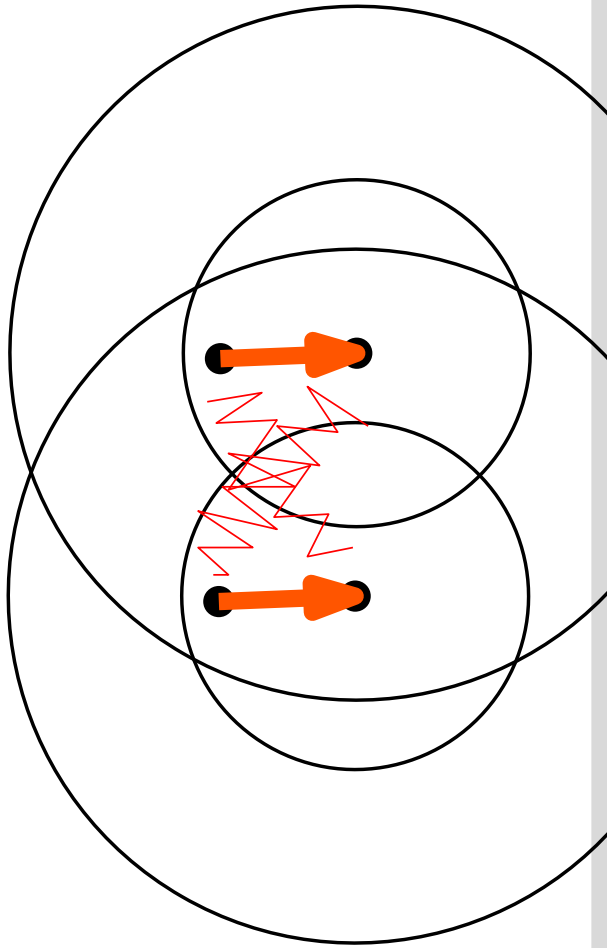


# Protokollmodell (einfach)

## Protokollmodell (einfach)

Im vereinfachten *Protokollmodell* kann ein Empfänger $r$ eine Nachricht eines Senders $s$ empfangen, wenn $|r - s| \leq 1$ und es keinen anderen Sender $s'$ mit $|r - s'| \leq (1 + \delta)$ gibt.

- oft einfach $\delta = 1$
- auch Knoten, die nicht dicht (oder sogar unverbunden) im Kommunikationsgraphen sind, interferieren miteinander
  - verteilte Algorithmen unmöglich?

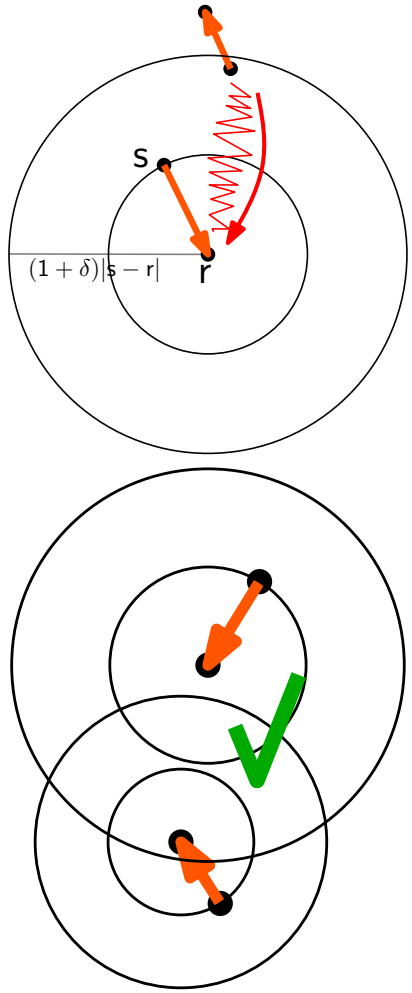


# Protokollmodell (erweitert)

## Protokollmodell

Im *Protokollmodell* kann ein Empfänger $r$ eine Nachricht eines Senders $s$ empfangen, wenn es keinen anderen Sender $s'$ gibt mit $|r - s'| \leq (1 + \delta)|r - s|$.

- „berücksichtigt“ Sender-Empfänger-Distanz
- ähnliche Vor- und Nachteile wie vereinfachte Variante



# Durchsatzkapazität und Transportkapazität

## Durchsatzkapazität

Die Durchsatzkapazität bezeichnet die Anzahl erfolgreich abgelieferter (Multi-Hop-)Pakete pro Zeit.

## Transportkapazität

Die Transportkapazität ist die Anzahl der Bit-Meter, die pro Zeiteinheit transportiert werden kann.

- Abhängig von Knotenverteilung und Kommunikationsmuster
- Frage etwa: „Was ist, unabhängig vom Protokoll, die maximale X-kapazität, wenn zufällig verteilte Knoten jeweils zufällig untereinander Pakete austauschen?“

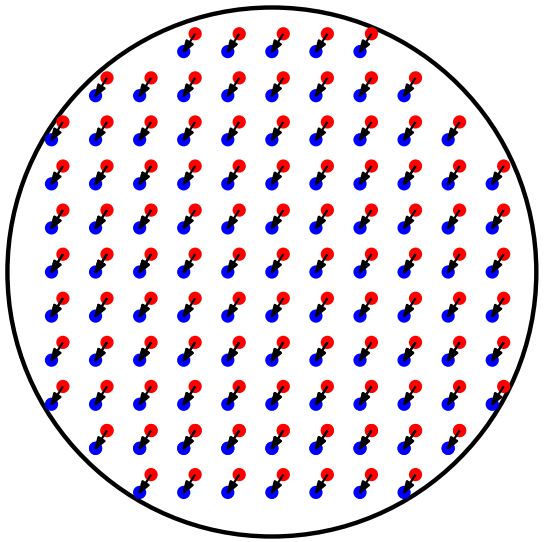
# Eine einfache Schranke für das Protokollmodell

## Satz

Sind $n$ Knoten in einer Scheibe mit Radius 1 plaziert, ist die maximal erreichbare Transportkapazität $\Theta(W\sqrt{n})$ [bit-meter/s] bei einer Datenrate von $W$ [bits/s].

Teil 1: Das ist erreichbar:

- Platziere $n/2$ Sender auf Gitter mit entsprechender Breite; Empfänger so dicht wie möglich verschoben!



# Eine einfache Schranke für das Protokollmodell

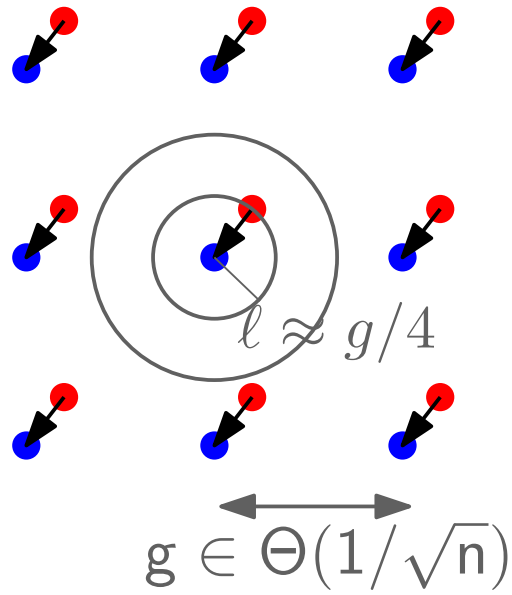
## Satz

Sind $n$ Knoten in einer Scheibe mit Radius 1 plaziert, ist die maximal erreichbare Transportkapazität $\Theta(W\sqrt{n})$ [bit-meter/s] bei einer Datenrate von $W$ [bits/s].

Teil 1: Das ist erreichbar:

- Platziere $n/2$ Sender auf Gitter mit entsprechender Breite; Empfänger so dicht wie möglich verschoben!
- Jede Übertragung überwindet Entfernung $\Omega(1/\sqrt{n})$
- es gibt $\Theta(n)$ Übertragungen

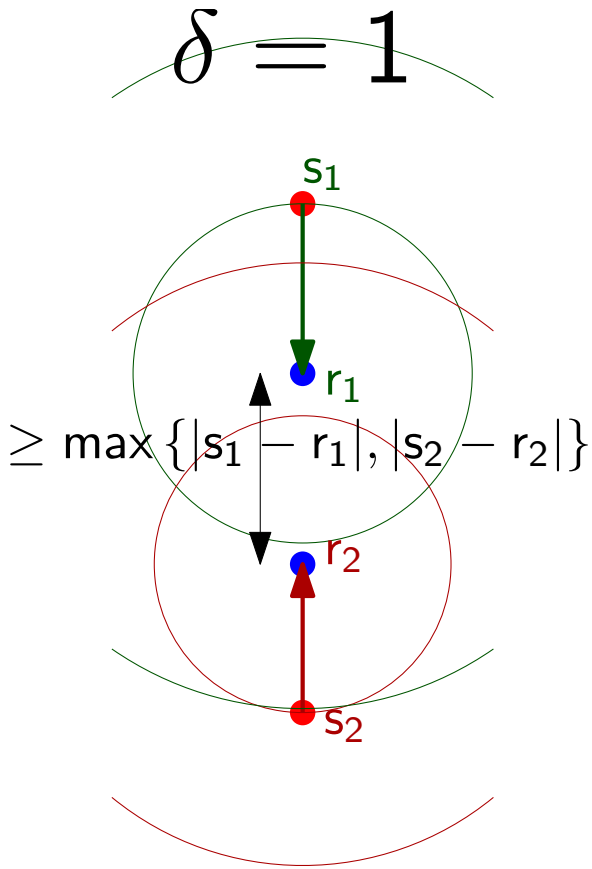
$\Rightarrow$ Gesamtlänge gleichzeitiger Übertragungen $\Omega(\sqrt{n})$



# Das ist optimal!

- zwei Übertragungen $(s_1, r_1)$ und $(s_2, r_2)$ können nur gleichzeitig ausgeführt werden, wenn (o. B.)

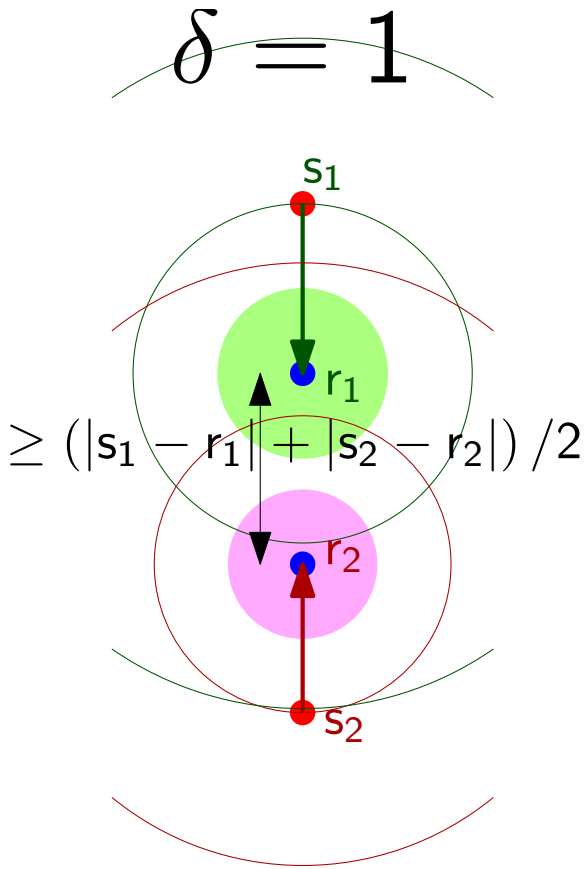
$$|r_1 - r_2| \geq \frac{\delta}{2}(|s_1 - r_1| + |s_2 - r_2|)$$



# Das ist optimal!

- zwei Übertragungen $(s_1, r_1)$ und $(s_2, r_2)$ können nur gleichzeitig ausgeführt werden, wenn (o. B.)

$$|r_1 - r_2| \geq \frac{\delta}{2}(|s_1 - r_1| + |s_2 - r_2|)$$



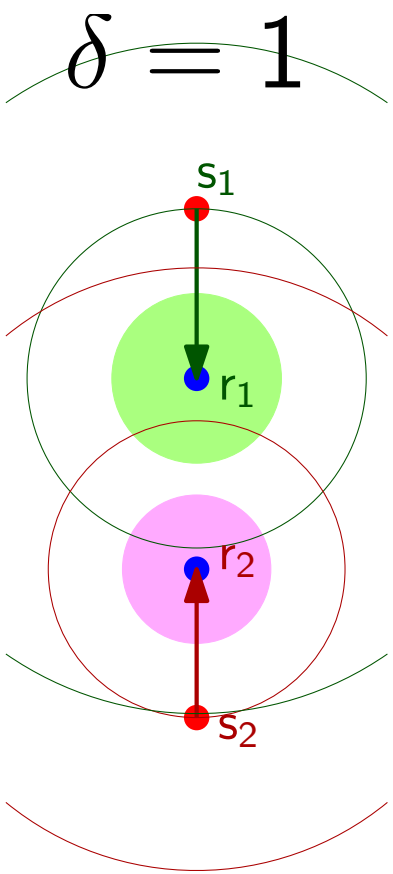
# Das ist optimal!

- zwei Übertragungen $(s_1, r_1)$ und $(s_2, r_2)$ können nur gleichzeitig ausgeführt werden, wenn (o. B.)

$$|r_1 - r_2| \geq \frac{\delta}{2}(|s_1 - r_1| + |s_2 - r_2|)$$

- Wir können um jedem Empfänger $r_i$ einen exklusiven Kreis von $\frac{\delta}{2}|s_i - r_i|$ ziehen
  - von so einem Kreis liegt mindestens ein Drittel in unserem Einheitskreis
  - eine Übertragung mit Länge $\ell$ deckt auf diese Art einen Kreis mit Durchmesser $\ell$ exklusiv ab.
  - bei gleicher Summe der Durchmesser haben gleich große Kreise die kleinste Gesamtfläche!
  - bei $n/2$ gleich großen Kreisen kann jeder Kreis nur Fläche $O(1/n)$ haben, und jeder Durchmesser $O(1/\sqrt{n})$

  $\Rightarrow$ Gesamtlänge der Übertragungen $O(n \cdot 1/\sqrt{n}) = O(\sqrt{n})$



# Physikalisches Modell: SINR

Bisher war Interferenz immer nur Ausschluss zwischen zwei Übertragungen. Keines der betrachteten Modelle hat berücksichtigt, dass sich störende Signale aufsummieren.

## SINR-Modell

Im *Signal-to-Interference-plus-Noise-Ratio Modell* kann ein Empfänger $r$ eine Nachricht eines Senders $s$ dekodieren, wenn

$$\frac{S}{I+N} \geq \beta$$

- $S$: Signalstärke des Signals von $s$ bei $r$
- $I$: Interferenz, Summe der fremden Signalstärken bei $r$
- $N$: Noise, Hintergrundrauschen
- $\beta$: zum Dekodieren notwendiges Verhältnis

# Physikalisches Modell: SINR

## SINR-Modell, allgemein

Im *Signal-to-Interference-plus-Noise-Ratio Modell* kann ein Empfänger $r$ eine Nachricht eines Senders $s$ dekodieren, wenn

$$\frac{P_s G_{sr}}{\sum_{s' \neq s} P_{s'} G_{s'r} + N} \geq \beta$$

- $P_s$: Signalstärke, mit der $s$ sendet
- $G_{sr}$: Leistungsabfall zwischen $s$ und $r$

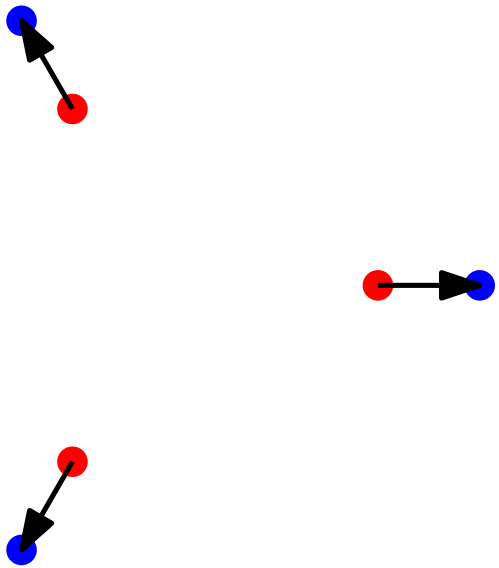
- manchmal nur mit festen Sendeleistungen $P_s \equiv P$
- oft für *geometrischen* Signalabfall definiert: $G_{sr} := d(s,r)^{-\alpha}$ ($\alpha$: path loss exponent)

# Zwei Beispiele (1)

## Beobachtung 1

Im (geometrischen) SINR-Modell kann es drei Übertragungen geben, die paarweise ausgeführt werden können, aber nicht alle gleichzeitig!

- rücke Übertragungen so dicht, dass sie gerade noch paarweise ausgeführt werden können
- werden alle gleichzeitig ausgeführt, empfängt jeder die doppelte Interferenz!
- so etwas modelliert *keines* der anderen Modelle (binärer Ausschluss!)



# Zwei Beispiele (2)

## Beobachtung 1

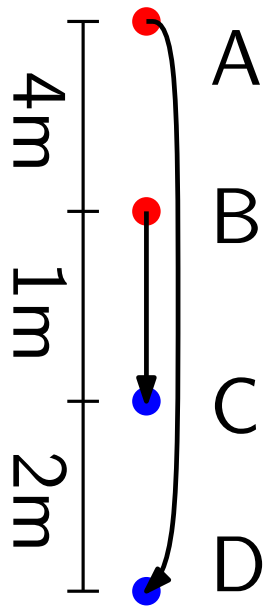
Im (geometrischen) SINR-Modell kann eine Übertragung *über eine andere hinwegsenden!*

- Bsp: $\alpha = 3$, $\beta = 3$, $N = 10\text{nW}$,
  Signalstärken: $P'_A = 1.26\text{mW}$, $P'_B = 31.6\mu\text{W}$
- SINR von $A$ bei $D$:

$$\frac{1.26\text{mW} \cdot 7^{-3}}{0.01\mu\text{W} + 31.6\mu\text{W} \cdot 3^{-3}} \approx 3.11 \geq \beta$$

- SINR von $B$ bei $C$:

$$\frac{31.6\mu\text{W} \cdot 1^{-3}}{0.01\mu\text{W} + 1.26\text{mW} \cdot 5^{-3}} \approx 3.13 \geq \beta$$




Beispiel: Roger Wattenhofer

# Diskussion SINR

SINR-Modell ist sehr viel realistischer, aber gleichzeitig viel komplexer zu analysieren. Unterscheiden sich die Ergebnisse?

- Für zufällige und bestmögliche Knotenpositionen gibt es bei den Schranken keinen nennenswerten Unterschied zum Protokollmodell!
- aber im worst-case!
- Dafür gibt es ein Beispiel, das eng mit *Data Gathering* und *Topology Control* zusammenhängt

# Scheduling+Aggregation

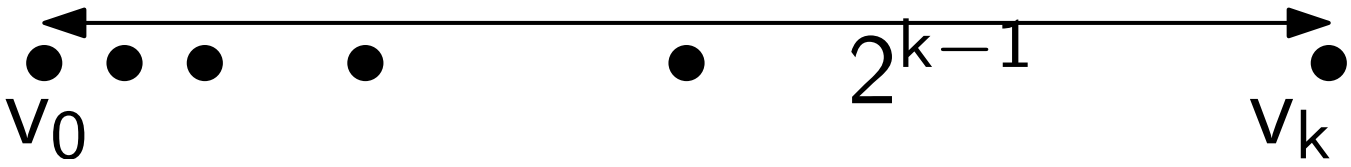
## Problem: Maximale Datenrate bei Aggregation

Gegeben *n* Sensorknoten und eine Senke in der Ebene. Was ist die beste Datenrate für eine vollständige Aggregation, die man auch bei schlechten Knotenpositionen erreichen kann?

- Knotenpositionen beliebig (worst-case, nicht zufällig)!
- Zwischenknoten können empfangene Daten vollständig aggregieren
- Übertragungen können frei gewählt werden
- Wie viele Abfragen (z.B. Durchschnittstemperatur) können pro Zeit bedient werden?

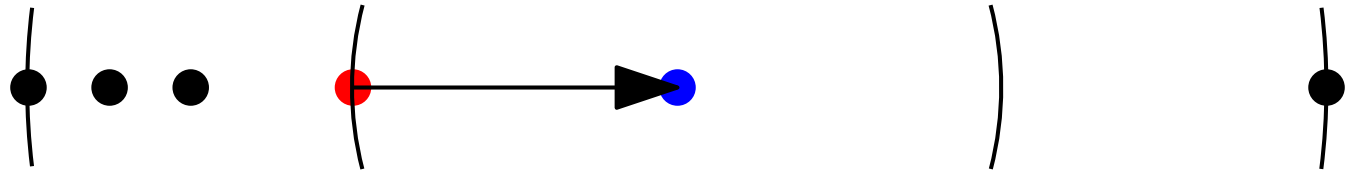
# Datenrate für Aggregation

- Wir betrachten exponentielle Knotenkette



# Datenrate für Aggregation

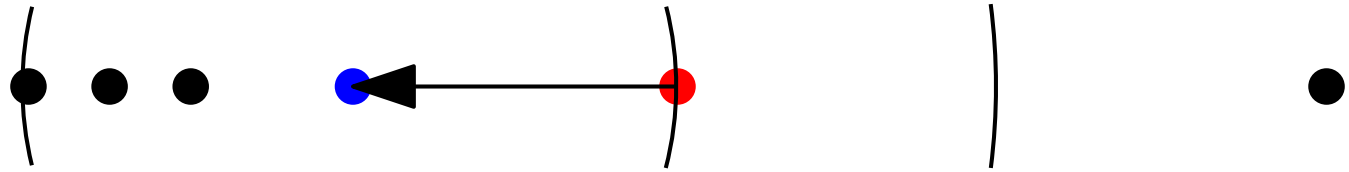
- Wir betrachten exponentielle Knotenkette



- Protokollmodell
  - wenn ein Knoten sendet (egal wohin), kann kein Knoten links davon senden!

# Datenrate für Aggregation

- Wir betrachten exponentielle Knotenkette



- Protokollmodell
  - wenn ein Knoten sendet (egal wohin), kann kein Knoten links davon senden!
  - immer nur ein Knoten sendet!

  $\Rightarrow$ beste Lösung: jeder sendet direkt zur Senke!

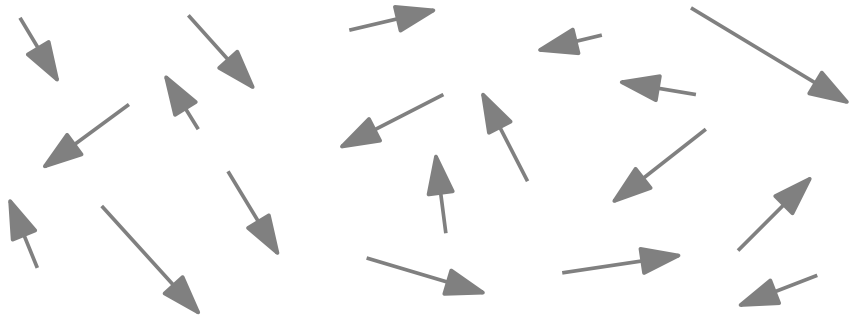
  $\Rightarrow$ Durchsatz immer in $O(1/n)$!
- SINR-Modell
  - genauso schlecht für Sendeleistung $\sim d^{\alpha}$!
  - aber: Durchsatz $\Omega(1/\log^2 n)$ möglich (ohne Beweis)
  - zentral, sehr kompliziert, aber überraschend!
    - *bei Interesse:* [T. Moscribroda, The Worst-Case Capacity of Wireless Sensor Networks, 2007]

# Scheduling belieb. Übertragungen

## Problem: Scheduling

**Gegeben:** Menge von Sender-Empfänger-Paaren $(s_1, r_1), \ldots, (s_n, r_n) \in \mathbb{R}^2 \times \mathbb{R}^2$ und Parameter $\alpha, \beta, N$.

**Gesucht:** Aufteilung der Paare auf möglichst wenige Zeitslots, so dass alle Übertragungen eines Zeitslots im entsprechenden SINR-Modell gleichzeitig erfolgreich stattfinden können.



# Scheduling belieb. Übertragungen

## Problem: Scheduling

**Gegeben:** Menge von Sender-Empfänger-Paaren $(s_1, r_1), \ldots, (s_n, r_n) \in \mathbb{R}^2 \times \mathbb{R}^2$ und Parameter $\alpha, \beta, N$.

**Gesucht:** Aufteilung der Paare auf möglichst wenige Zeitslots, so dass alle Übertragungen eines Zeitslots im entsprechenden SINR-Modell gleichzeitig erfolgreich stattfinden können.

- bereits für einheitliche Sendeleistungen NP-schwer
- $O(\log n)$ Approximationsalgorithmus bekannt
- bessere Algorithmen möglich?

# Empfehlung weiterführende Literatur

Olga Goussevskaia, Yvonne Anne Oswald, Roger Wattenhofer:
*Complexity in Geometric SINR*.
In: Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '07), 2007.

- Behandelt das Scheduling im geometrischen SINR Modell
  - NP-Schwere Beweis
  - Erste Approximationsalgorithmen

# Empfehlung weiterführende Literatur

Olga Goussevskaia, Yvonne Anne Oswald, Roger Wattenhofer:
*Complexity in Geometric SINR*.
In: Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '07), 2007.

- Behandelt das Scheduling im geometrischen SINR Modell
  - NP-Schwere Beweis
  - Erste Approximationsalgorithmen

Olga Goussevskaia, Magnús M. Halldórsson, Roger Wattenhofer:
*Algorithms for Wireless Capacity*.
In: ACM Transactions on Networking (Vol 22, Issue 3, 2014)

- Neuere Resultate
  - $O(1)$-Faktor Approximation für One-Shot Scheduling
  - $O(\log n)$-Faktor Approximation Scheduling
  - Robustheit des geometrischen SINR Modells

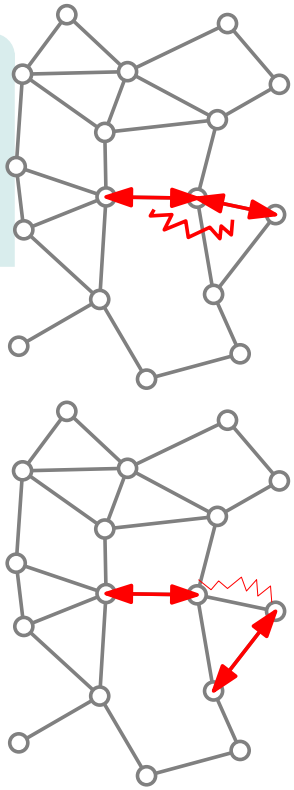
# Zwischenstand: Kapazität und Interferenz

- Kapazität
  - wie viel Durchsatz kann man in Paketen / Bitmetern erreichen?
  - hängt stark ab vom Interferenzmodell und der Knotenverteilung

- Interferenzmodelle
  - klassische Interferenzmodelle sind binäre Ausschlüsse
  - physikalisches SINR Interferenzmodell sehr viel genauer
    - bei worst-case-Betrachtung macht das einen *exponentiellen* Unterschied!
    - leider auch sehr viel komplexer
    - viele offene Fragen!

# Motivation

Kommunikation im drahtlosen Kanal sind selbst bei einfachsten Interferenzmodellen nicht beliebig gleichzeitig möglich



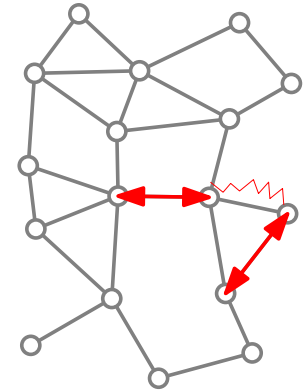
Interferenzmodell:

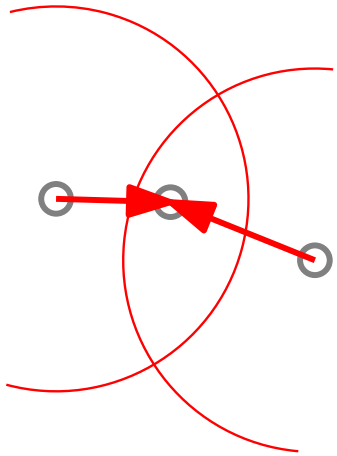
- kein Knoten kann gleichzeitig an zwei Übertragungen teilnehmen
- benachbarte Knoten können nicht gleichzeitig übertragen/empfangen

$\Rightarrow$ Einfach Drauflos-senden ist keine Lösung!

# Ad-hoc-Lösung: CSMA/CD

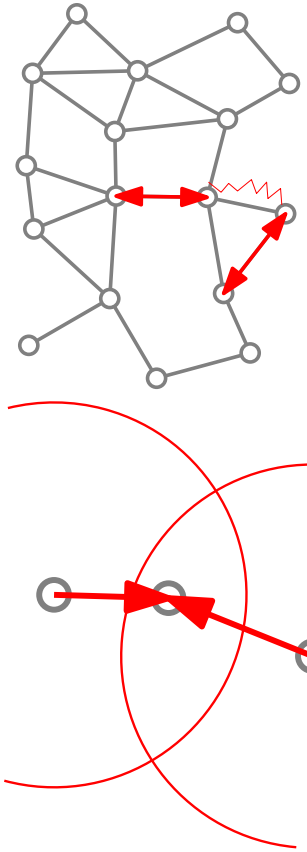
- Ad-hoc-Lösung: CSMA/CD (wie Ethernet)
  - *Carrier Sense Multiple Access/Collision Detection*
  - Carrier Sense: Mithören, ob der Kanal frei ist
  - Collision Detection: Bei Kollision später wiederholen
  - Achtung: Anders als im Ethernet kann der Sender Kollisionen nicht immer erkennen!
  - Was, wenn der Empfänger von jemandem gestört wird, den der Sender nicht sieht? (Hidden Terminal Problem)





# Ad-hoc-Lösung: CSMA/CA

- Ad-hoc-Lösung: CSMA/CA (ähnlich Ethernet)
  - *Carrier Sense Multiple Access/Collision Avoidance*
  - Carrier Sense: Mithören, ob der Kanal frei ist
  - Collision Avoidance: Kollisionen verhindern
  - Achtung: Anders als im Ethernet kann der Sender Kollisionen nicht immer erkennen!
  - Was, wenn der Empfänger von jemandem gestört wird, den der Sender nicht sieht? (Hidden Terminal Problem)
- CSMA/CA-Protokolle (gaaaaanz grob)
  - Sender wartet, bis er freien Kanal sieht
  - Sender schickt „Request To Send“
  - Empfänger bestätigt „Clear To Send“
    - (das hören hoffentlich auch alle Betroffenen!)
  - Sender schickt Daten
  - Empfänger bestätigt



# Die koordinierte Lösung: TDMA

- Übertragungen bekommen Slots in Zeitraster zugewiesen
  - *Time Division Multiple Access*
  - entweder kurzfristig bei Bedarf oder sogar langfristig/periodisch
- Übertragungen sind auch *gleichzeitig* möglich, wenn mehrere Frequenzen/orthogonale Kodierungen zur Verfügung stehen
  - CDMA: Code Division ..
  - FDMA: Frequency Division ..

Übertragungen können sogar gleichzeitig dieselbe Ressource belegen, wenn sie weit genug auseinander liegen!

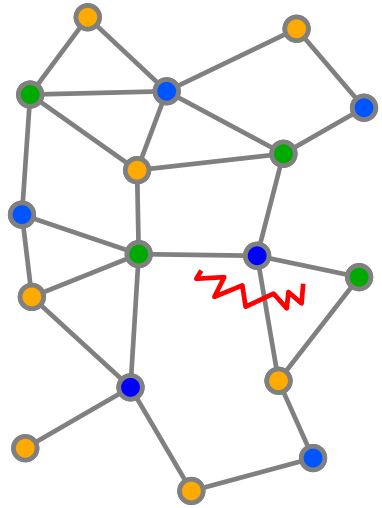
# Knotenfärbung

## Definition

Eine Knotenfärbung eines Graphen $G = (V, E)$ ist eine Abbildung $c : V \to \mathbb{N}$ so, dass für jede Kante die beiden Endpunkte unterschiedliche Farben haben, also

$$\{u, v\} \in E \Rightarrow c(u) \neq c(v)$$

- Eine Färbung $c$ hat die Größe $|c| = \max_{v \in V} c(v)$



- Jeder darf nur senden, wenn „seine Farbe“ dran ist
- Das bringt nicht direkt etwas

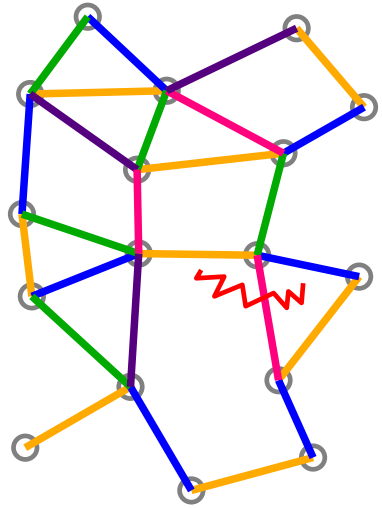
# Kantenfärbung

## Definition

Eine Kantenfärbung eines Graphen $G = (V, E)$ ist eine Abbildung $c : E \to \mathbb{N}$ so, dass für jeden Knoten alle inzidenten Kanten unterschiedliche Farben haben, also

$$\{u, v\}, \{u, w\} \in E \Rightarrow c(\{u, v\}) \neq c(\{u, w\})$$

- Eine Färbung $c$ hat die Größe $|c| = \max_{e \in E} c(e)$



- Jeder darf nur über Kante kommunizieren, „deren Farbe“ dran ist
- schließt zumindest direkte Kollisionen aus

# Knoten-, Kanten-, Abstand-$d$-Färbungen

## Beobachtung

Eine Kantenfärbung ist eine Knotenfärbung im Graphen $G' = (E, E')$ mit $E' := \{\{e_1, e_2\} : e_1 \cap e_2 \neq \emptyset\}$

- Wir können Kantenfärbungen auf Knotenfärbungen zurückführen!
- das lässt sich erweitern auf andere Färbungen
  - Abstand-$d$-Färbungen: Knoten mit Abstand $\leq d$ müssen unterschiedlich gefärbt werden
  - jede Bedingung, in der Ausschlüsse innerhalb von konstanter Entfernung liegen!

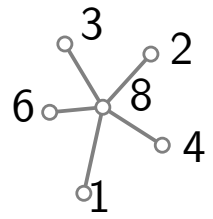
Deshalb kümmern wir uns nur um Knotenfärbungen!

# Warm-up: Zentrale $\Delta + 1$-Färbung

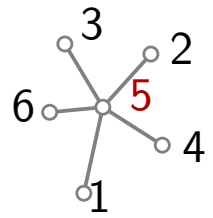
Karlsruher Institut für Technologie

## Satz

Jeder Graph lässt sich mit $\Delta + 1$ Farben färben.



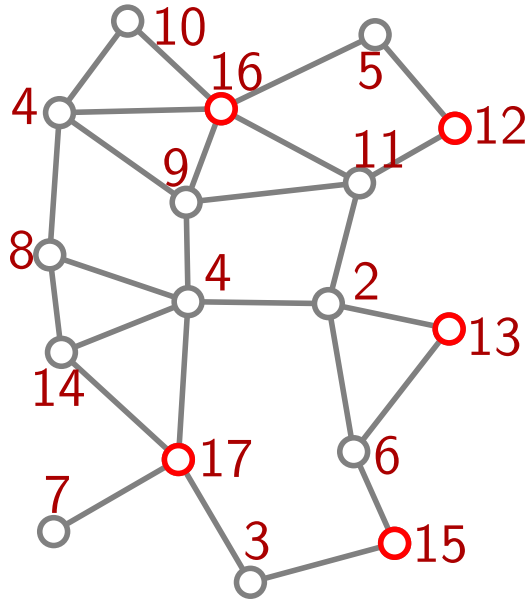

- Beweis: wenn es einen Knoten $u$ gibt, der eine höhere Farbe hat als $\deg(u) + 1$, dann ist eine der „Farben“ $1, \ldots, \deg(u) + 1$ in der Nachbarschaft nicht verwendet worden
- besser wollen wir gar nicht sein, aber wie bekommen wir das *verteilt* hin?



# Warm-up: Verteilte $\Delta + 1$-Färbung

## Einfache Lösung

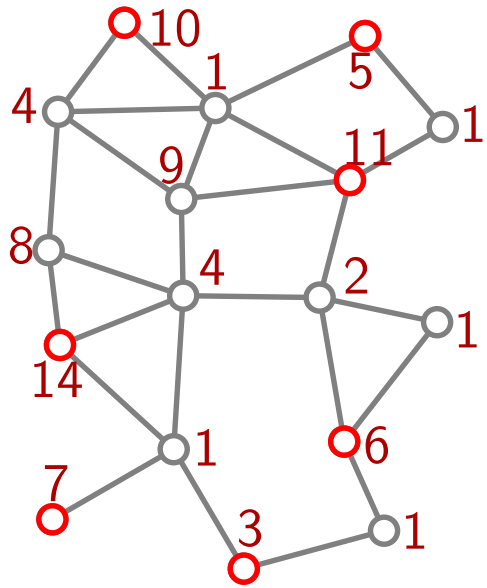
1. jeder färbt sich mit seiner ID
2. jeder Knoten $u$ mit Farbe $c(u) \geq \deg(u) + 1$, der die höchste Farbe in seiner Nachbarschaft hat, wählt die kleinste Farbe aus $1, \ldots, \deg(u)$ aus, die keiner der Nachbarn hat



# Warm-up: Verteilte $\Delta + 1$-Färbung

## Einfache Lösung

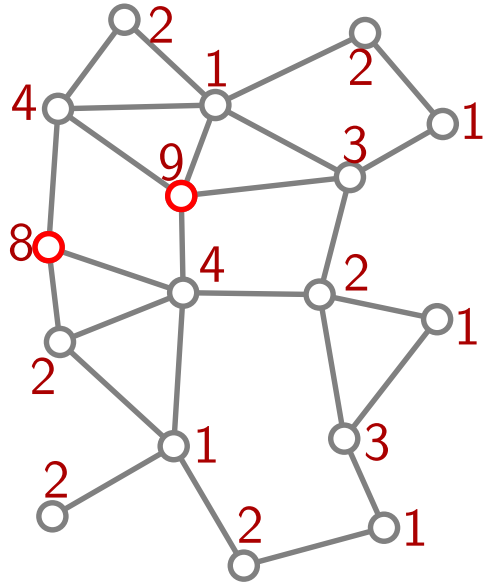
1. jeder färbt sich mit seiner ID
2. jeder Knoten $u$ mit Farbe $c(u) \geq \deg(u) + 1$, der die höchste Farbe in seiner Nachbarschaft hat, wählt die kleinste Farbe aus $1, \ldots, \deg(u)$ aus, die keiner der Nachbarn hat



# Warm-up: Verteilte $\Delta + 1$-Färbung

## Einfache Lösung

1. jeder färbt sich mit seiner ID
2. jeder Knoten $u$ mit Farbe $c(u) \geq \deg(u) + 1$, der die höchste Farbe in seiner Nachbarschaft hat, wählt die kleinste Farbe aus $1, \ldots, \deg(u)$ aus, die keiner der Nachbarn hat



# Satz

## Satz

Es gibt einen verteilten Algorithmus, der eine $\Delta + 1$-Färbung in $O(\Delta^2 + \log_2^*(n))$ Schritten berechnet.

- $\log^*(n)$: Anzahl der Anwendungen von log, bevor das Argument unter 1 fällt. (Bsp: $\log_2^*(10^{50}) = 5$)
- einfacher zu merken:

$$\log_2^*(x) = 5 \Leftrightarrow 2^{2^{2^2}} < x \leq 2^{2^{2^{2^2}}}$$

# Beweisstruktur

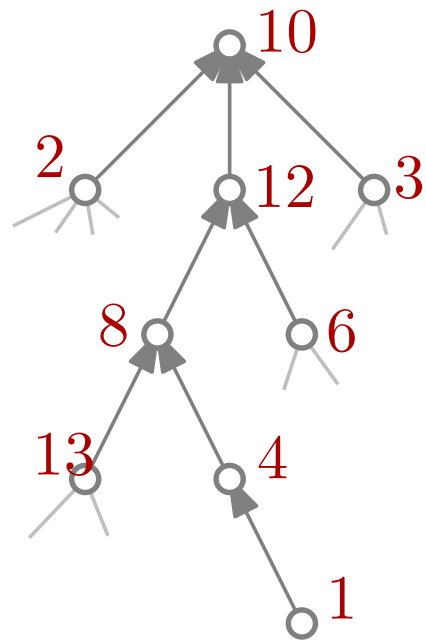
## Beweisstruktur

- es gibt einen Algorithmus, der in einem Baum in $\log^*(n)$ Schritten eine 6-Färbung berechnet
- es gibt einen Algorithmus, der in einem Baum in $k$ Runden eine Färbung mit $3 + k$ Farben auf 3 Farben reduziert
- beide Verfahren funktionieren auch auf „Pseudo-Bäumen“
- jeder Graph läßt sich in $\Delta$ Schritten in $\Delta$ Pseudo-Bäume zerlegen
- $\Delta$ Färbungen mit je 3 Farben lassen sich in $\Delta^2$ Runden auf $\Delta + 1$ Farben reduzieren

# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

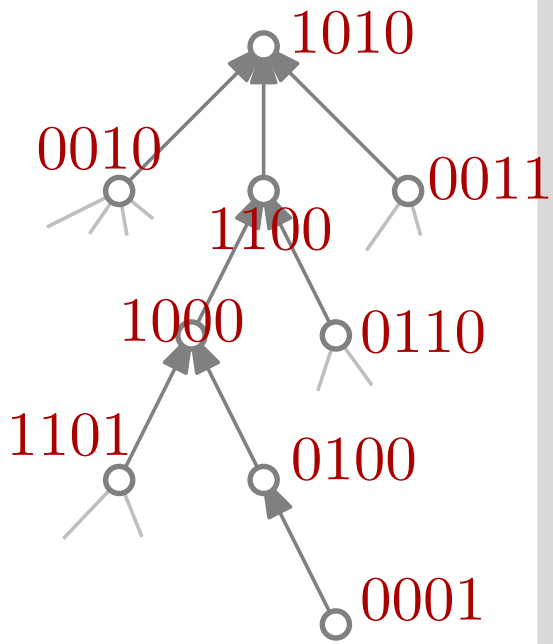
1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)



# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

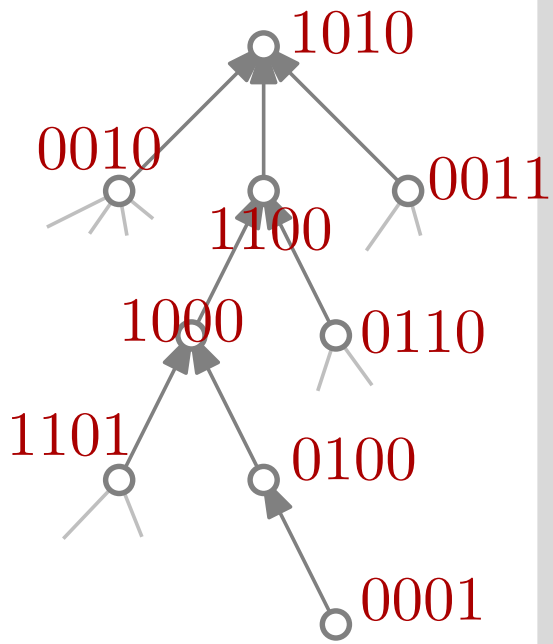
1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)



# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

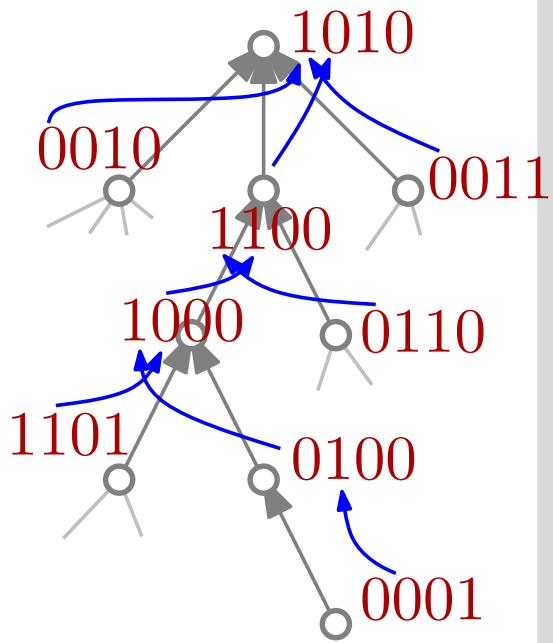
1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)



# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)

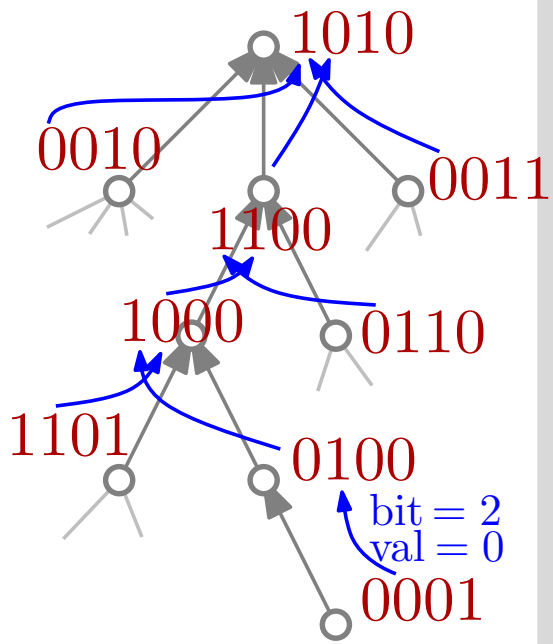



Bit von links

# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)



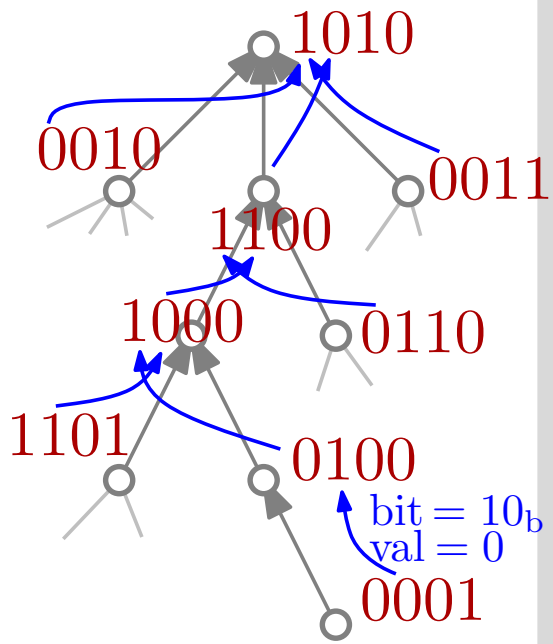

Kodierung von rechts:

Positionen 0,1,2,3

# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)



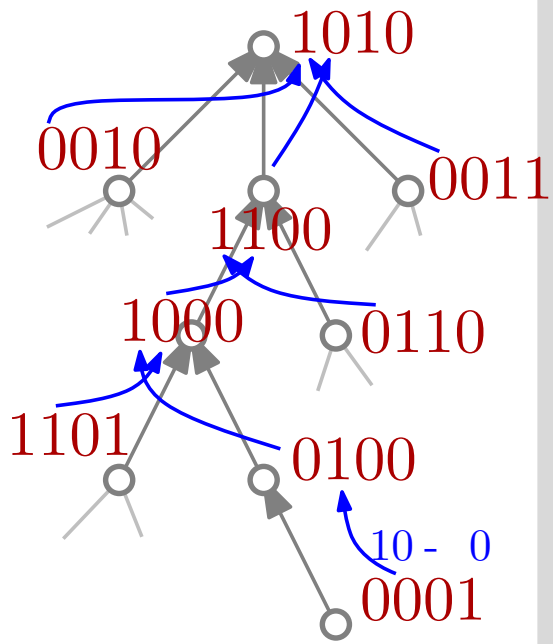

Kodierung von rechts:

Positionen 0,1,2,3

# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

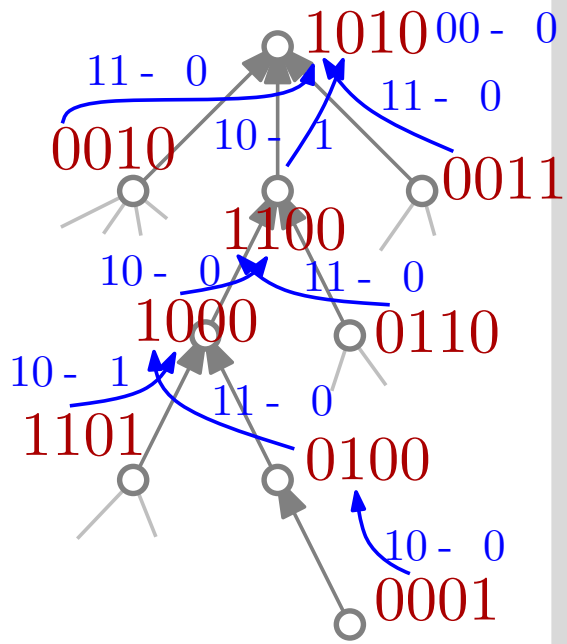
1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)



# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

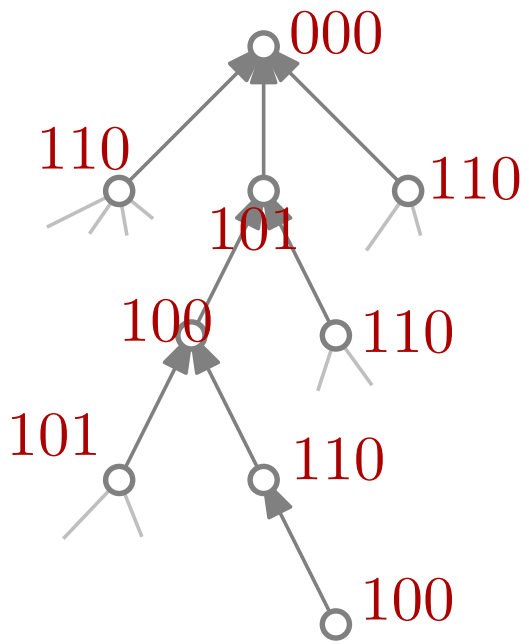
1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)



# 6-Färbung eines Baumes

## 6-Färbung eines Baumes

1. schreibe Label binär in $\log_2 n$ Bits
   - (dafür muss a priori zumindest obere Schranke für $n$ bekannt sein)
2. Setze $L = \log_2 n$ (aktuelle Labellänge)
3. Solange $L > 3$ und dann noch einmal:
   - suche höchstes Bit, in dem sich das eigene Label und das des Vorgängers unterscheiden
   - kodiere dessen Position in $\log_2(L)$ Bits
   - verwende als neues Label die kodierte Position und den Wert des eigenen Bits
   - die Wurzel wählt dafür beliebiges Bit aus
   - setze $L = \log_2(L) + 1$ (neue Labellänge)



# Korrektheit Baum-6-Färbung

## Satz

Der beschriebene Algorithmus berechnet eine 6-Färbung.

- Nach jeder Runde liegt eine Färbung des Baumes vor
  - Annahme: zwei Knoten haben nach einer Runde dasselbe Label
  - dann haben sie beide dasselbe Bit ausgewählt und hatten dort vorher denselben Wert stehen
  - aber das Kind sollte Bit wählen, in dem es sich vom Vorgänger unterscheidet! Widerspruch!
- Die Färbung enthält zum Schluss maximal 6 Farben
  - $L$ ist immer aktuelle Labellänge
  - wenn $L \leq 3$, wird noch eine Runde ausgeführt.
  - dann reichen die drei Beschriebungen $00, 01, 10$ für die Position
  - dann gibt es nur noch die Labels $000, 001, 010, 011, 100, 101$

# Laufzeit Baum-6-Färbung

## Satz

Der beschriebene Algorithmus terminiert nach $O(\log^*(n))$ Schritten.

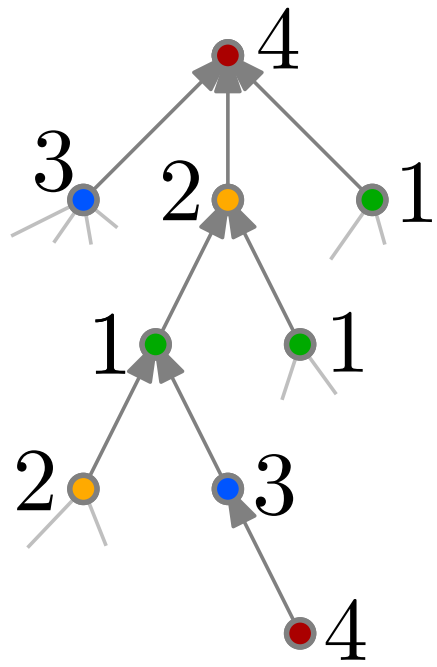
- Das werden wir nicht beweisen.
- Grobe Vorstellung:
  - in jeder Runde wird die Labellänge von $L$ auf $\log_2(L) + 1$ gedrückt
  - ohne das $+1$ wäre die Aussage klar
  - ein bißchen Formelschieberei löst das Problem
  - (wer nicht ohne Leben kann – siehe Buch)

Das war schon Teil 1: Wir können einen Baum in $O(\log^*(n))$ Schritten mit 6 Farben färben!

# Kompression von 6 auf 3 Farben im Baum

## Baum-(3+k)-zu-3-Färbung

Für $i = (3 + k), \ldots, 4$ tue folgendes:

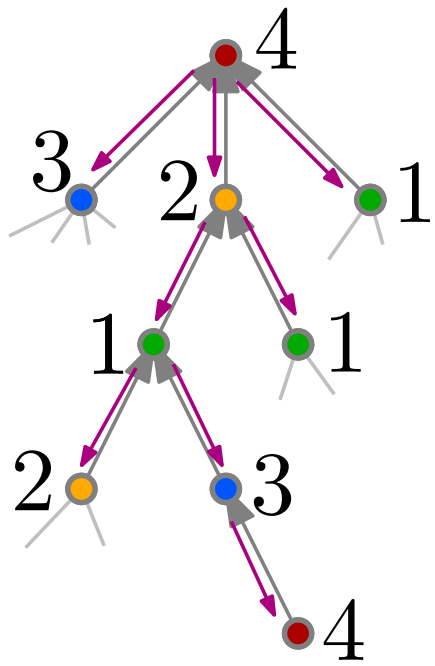


# Kompression von 6 auf 3 Farben im Baum

## Baum-(3+k)-zu-3-Färbung

Für $i = (3 + k), \ldots, 4$ tue folgendes:

1. jeder Knoten nimmt die Farbe seines Vorgängers an („shift down“)
   - Wurzel nimmt *neue Farbe* aus $\{1, 2, 3\}$

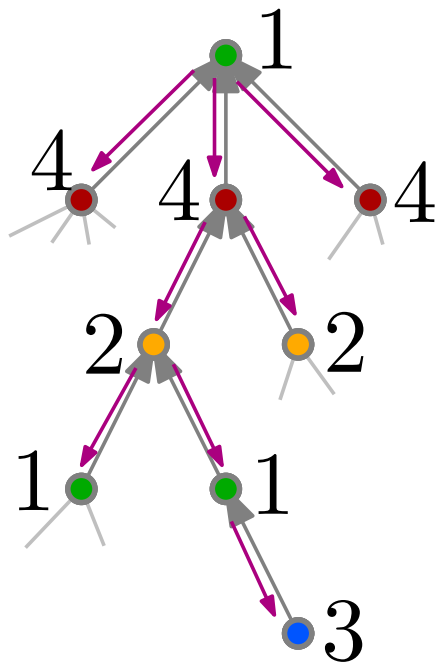


# Kompression von 6 auf 3 Farben im Baum

## Baum-(3+k)-zu-3-Färbung

Für $i = (3 + k), \ldots, 4$ tue folgendes:

1. jeder Knoten nimmt die Farbe seines Vorgängers an („shift down“)
   - Wurzel nimmt *neue Farbe* aus $\{1, 2, 3\}$

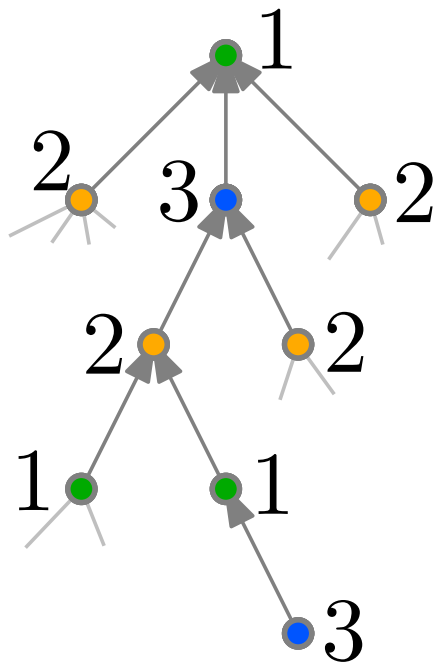


# Kompression von 6 auf 3 Farben im Baum

## Baum-(3+k)-zu-3-Färbung

Für $i = (3 + k), \ldots, 4$ tue folgendes:

1. jeder Knoten nimmt die Farbe seines Vorgängers an („shift down“)
   - Wurzel nimmt *neue Farbe* aus $\{1, 2, 3\}$
2. jeder Knoten mit Farbe $i$ wählt Farbe aus $\{1, 2, 3\}$, die kein Nachbar hat



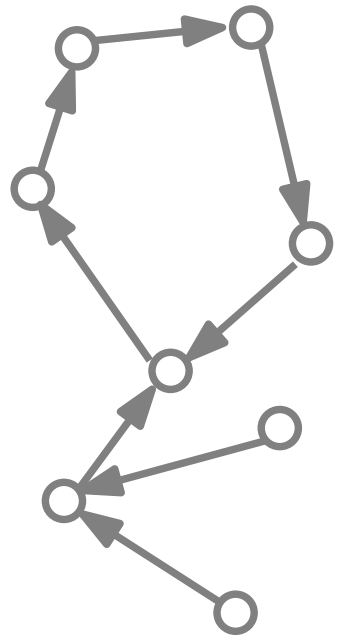

- Korrektheit:
  - nach einem Shift hat jeder Knoten die Farbe seines Vorgängers, und seine Kinder seine alte Farbe
    - das ist eine Färbung!
    - jeder Knoten „sieht“ nur 2 benachbarte Farben
  - in jeder Runde werden wir Farbe $i$ los!
- nach $k$ Runden je 2 Schritte haben wir noch 3 Farben!

# Pseudo-Wälder (klingt schlimmer als es ist)

## Definition Pseudo-Wald

Ein gerichteter Graph heißt *Pseudo-Wald*, wenn jeder Knoten höchstens eine ausgehende Kante hat.

- jeder Knoten zeigt auf maximal einen „Vorgänger“
  - Vorsicht: Graph kann gerichtete Kreise enthalten



## Satz

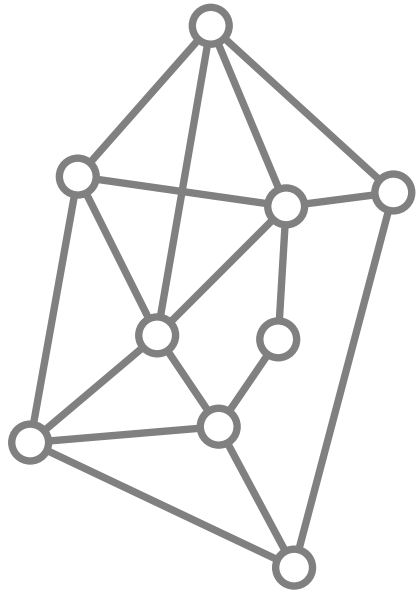
Die Dreifärbung von Pseudo-Wäldern funktioniert exakt wie bei Bäumen.

- Beweise gehen *exakt* wie vorher, nur gibt es ggf. mehrere Knoten ohne Vorgänger

# Pseudo-Wald-Dekomposition

## Definition

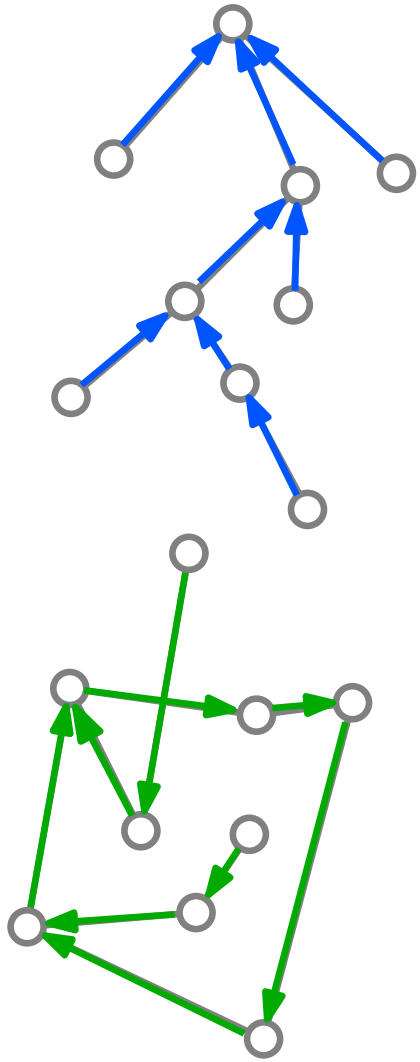
Eine *Pseudo-Wald-Dekomposition* eines Graphen $G$ ist eine Menge von Pseudo-Wäldern $F_1, \ldots, F_\Delta$, so dass jede Kante von $G$ in mindestens einem $F_i$ vorkommt (in beliebiger Richtung).



# Pseudo-Wald-Dekomposition

## Definition

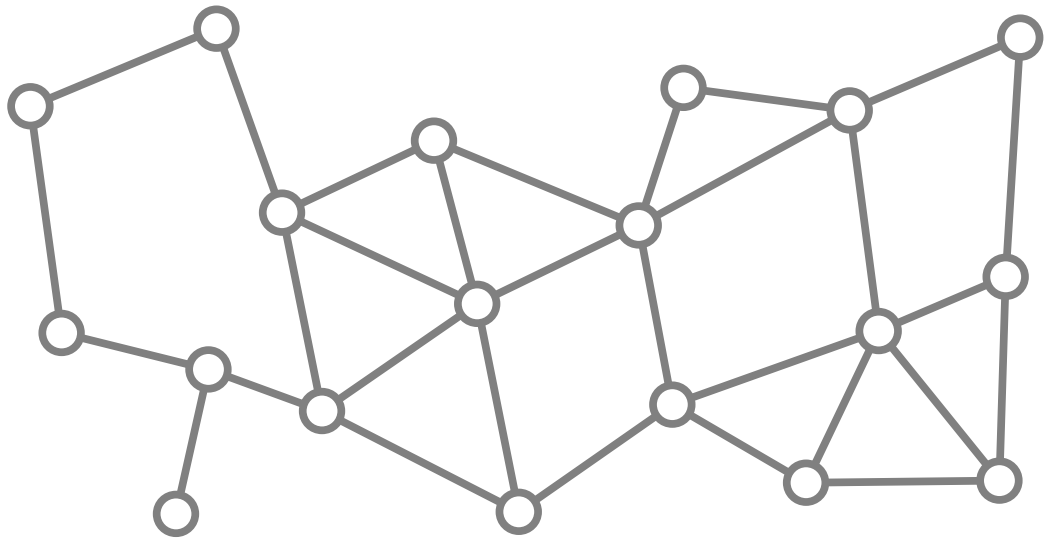
Eine *Pseudo-Wald-Dekomposition* eines Graphen $G$ ist eine Menge von Pseudo-Wäldern $F_1, \ldots, F_\Delta$, so dass jede Kante von $G$ in mindestens einem $F_i$ vorkommt (in beliebiger Richtung).



# Verteilte Pseudo-Wald-Dekomposition

## Verteilte Dekomposition, Runde $i = 1, \ldots, \Delta$

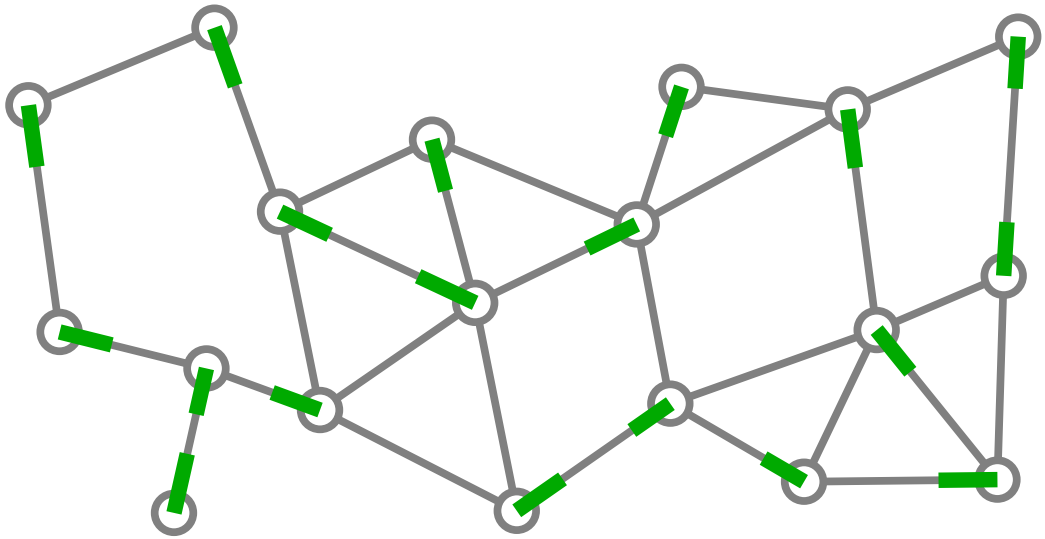
1. jeder Knoten wählt eine inzidente unmarkierte Kante aus, wenn noch möglich
2. eine gewählte Kante $\{u, v\}$ wird markiert und zu $F_i$ hinzugefügt, und zwar von einem Knoten weg, der sie wählte.



# Verteilte Pseudo-Wald-Dekomposition

## Verteilte Dekomposition, Runde $i = 1, \ldots, \Delta$

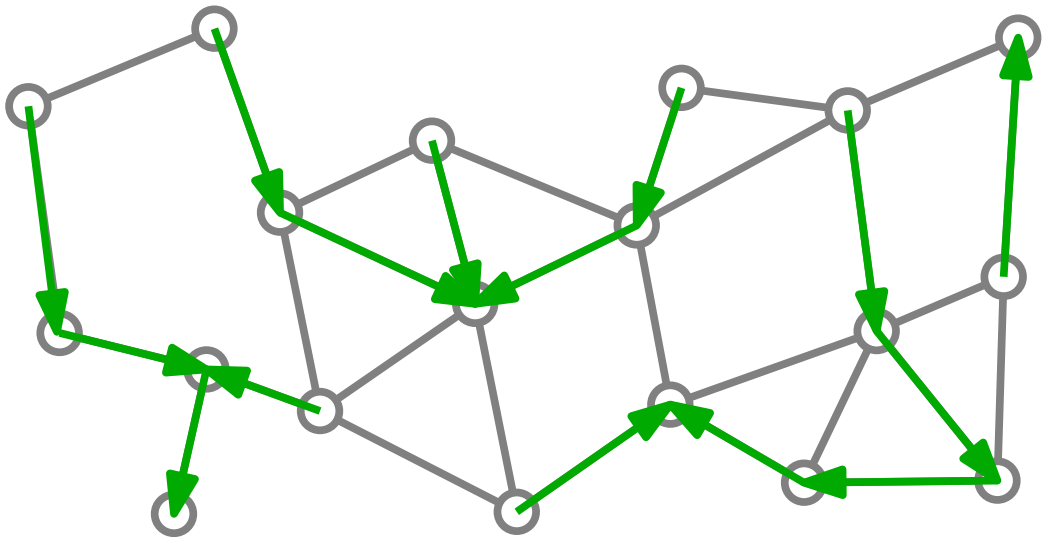
1. jeder Knoten wählt eine inzidente unmarkierte Kante aus, wenn noch möglich
2. eine gewählte Kante $\{u, v\}$ wird markiert und zu $F_i$ hinzugefügt, und zwar von einem Knoten weg, der sie wählte.



# Verteilte Pseudo-Wald-Dekomposition

## Verteilte Dekomposition, Runde $i = 1, \ldots, \Delta$

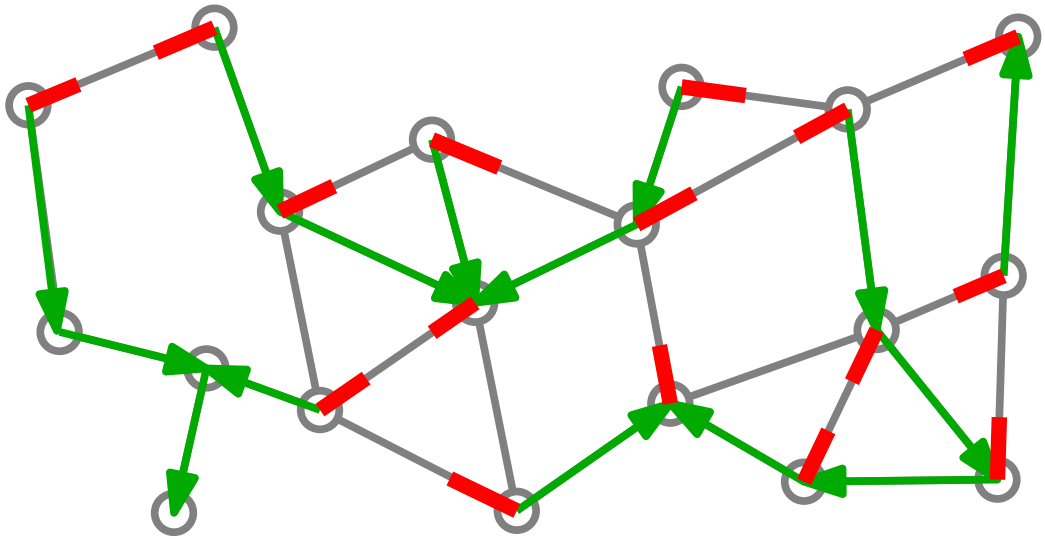
1. jeder Knoten wählt eine inzidente unmarkierte Kante aus, wenn noch möglich
2. eine gewählte Kante $\{u, v\}$ wird markiert und zu $F_i$ hinzugefügt, und zwar von einem Knoten weg, der sie wählte.



# Verteilte Pseudo-Wald-Dekomposition

## Verteilte Dekomposition, Runde $i = 1, \ldots, \Delta$

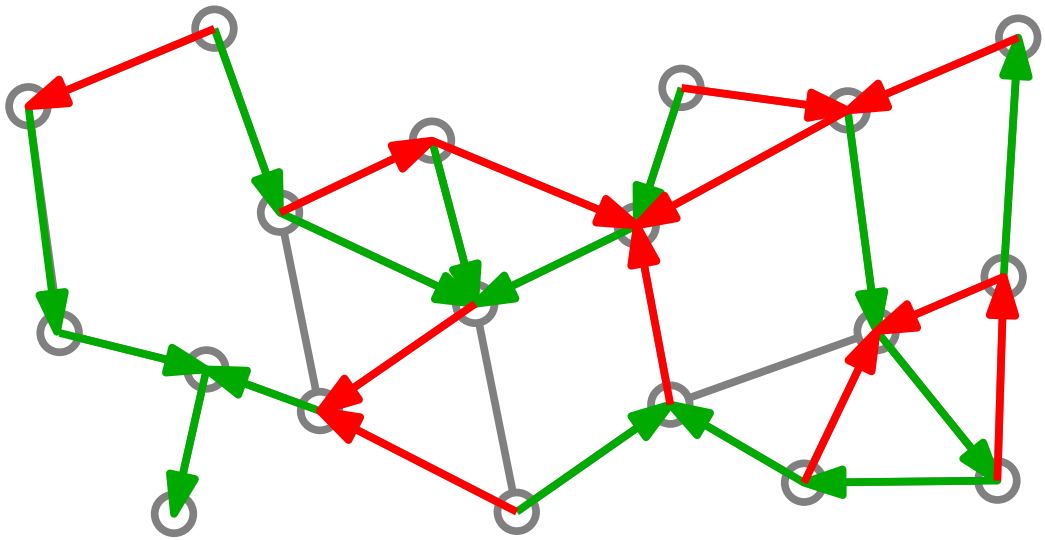
1. jeder Knoten wählt eine inzidente unmarkierte Kante aus, wenn noch möglich
2. eine gewählte Kante $\{u, v\}$ wird markiert und zu $F_i$ hinzugefügt, und zwar von einem Knoten weg, der sie wählte.



# Verteilte Pseudo-Wald-Dekomposition

## Verteilte Dekomposition, Runde $i = 1, \ldots, \Delta$

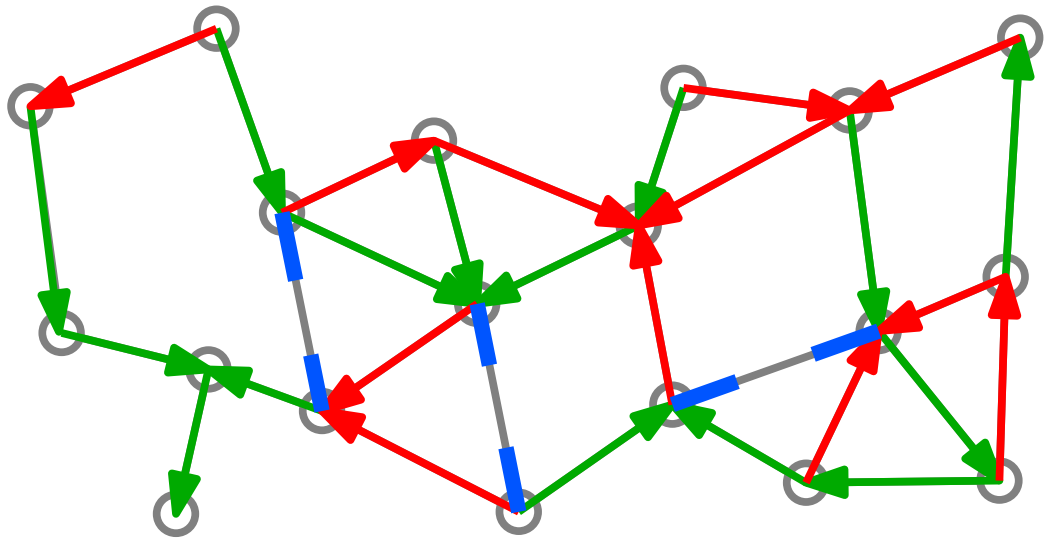
1. jeder Knoten wählt eine inzidente unmarkierte Kante aus, wenn noch möglich
2. eine gewählte Kante $\{u, v\}$ wird markiert und zu $F_i$ hinzugefügt, und zwar von einem Knoten weg, der sie wählte.



# Verteilte Pseudo-Wald-Dekomposition

## Verteilte Dekomposition, Runde $i = 1, \ldots, \Delta$

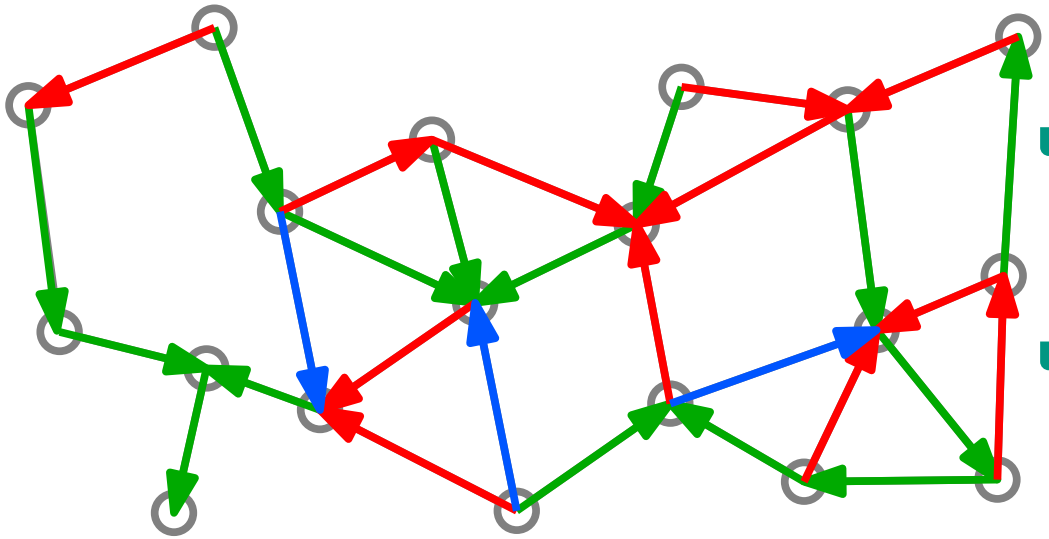
1. jeder Knoten wählt eine inzidente unmarkierte Kante aus, wenn noch möglich
2. eine gewählte Kante $\{u, v\}$ wird markiert und zu $F_i$ hinzugefügt, und zwar von einem Knoten weg, der sie wählte.



# Verteilte Pseudo-Wald-Dekomposition

## Verteilte Dekomposition, Runde $i = 1, \ldots, \Delta$

1. jeder Knoten wählt eine inzidente unmarkierte Kante aus, wenn noch möglich
2. eine gewählte Kante $\{u, v\}$ wird markiert und zu $F_i$ hinzugefügt, und zwar von einem Knoten weg, der sie wählte.



- nach $\Delta$ Runden (oder früher) sind alle Kanten markiert
- in jeder Runde wird ein Pseudo-Wald gewählt

# Wo stehen wir?

- Wir können den Graphen in $\Delta$ Schritten in $\Delta$ Pseudo-Wälder dekomponieren
- Wir können Bäume und Pseudo-Wälder verteilt mit 3 Farben einfärben
  - erst färben wir in $\log^*(n)$ Runden mit konstant vielen Schritten 6-farbig,
  - dann reduzieren wir die Farben in 3 Runden mit konstant vielen Schritten auf 3

$\Rightarrow$ Wir können in $O(\log^*(n))$ Schritten jeden der Pseudo-Wälder 3 dreifärben

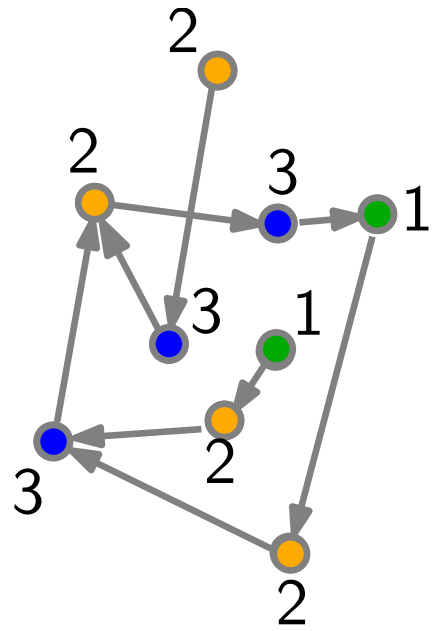
  - das kann ja in allen Pseudowäldern parallel geschehen

was fangen wir mit $\Delta$ 3-Färbungen für die Pseudo-Wälder an? Wir wollen eine $\Delta + 1$-Färbung des Graphen!

# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

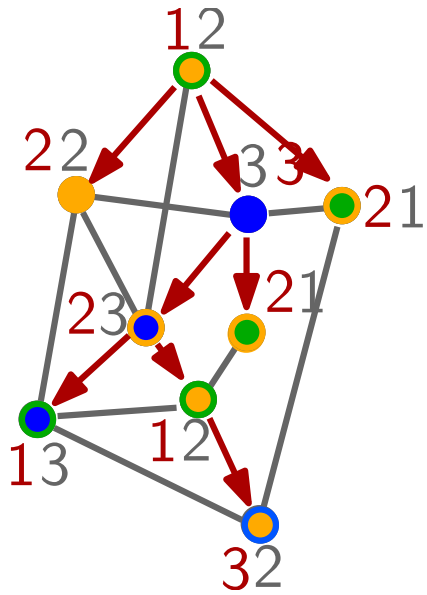
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

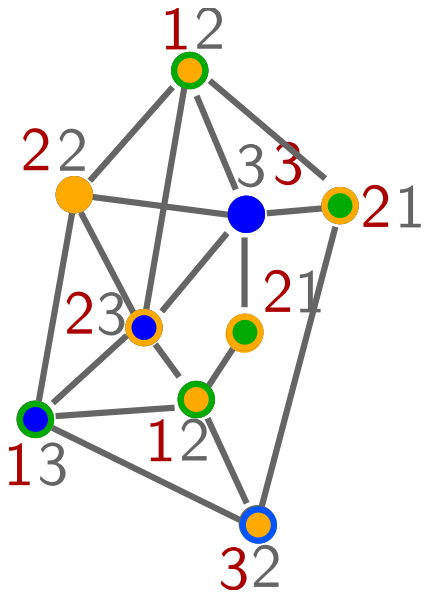
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

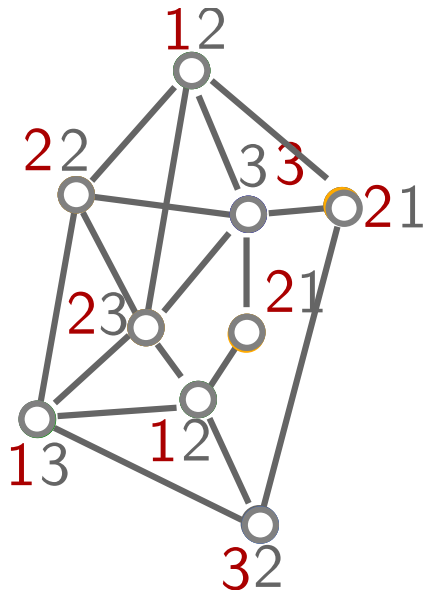
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

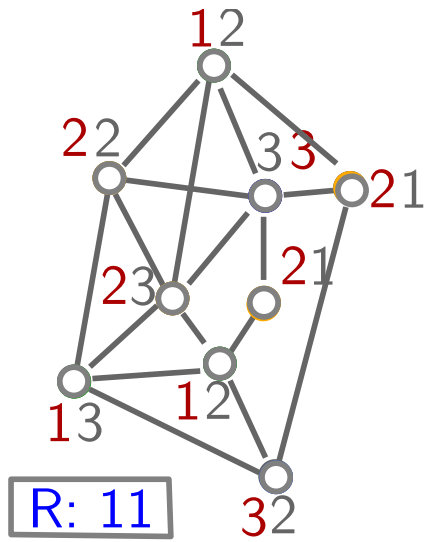
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

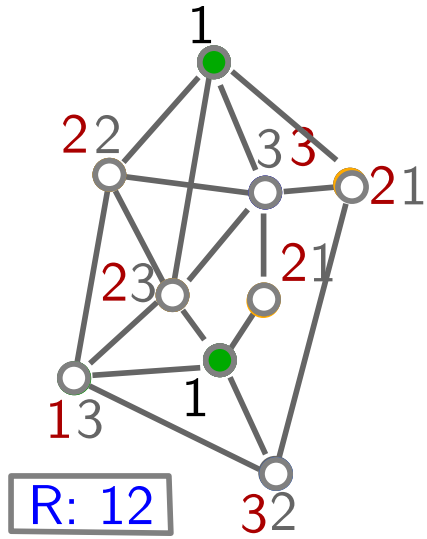
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

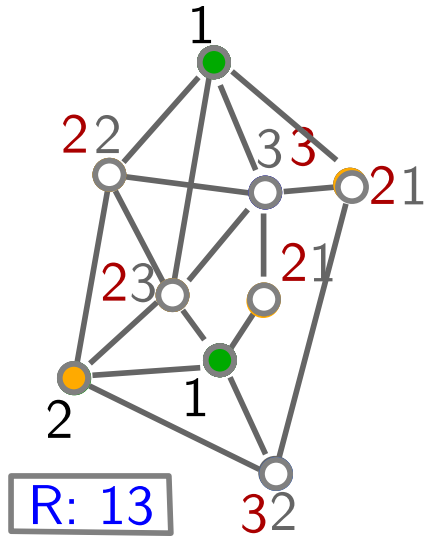
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

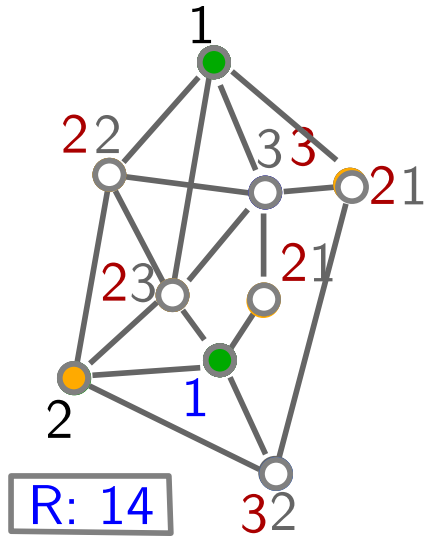
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

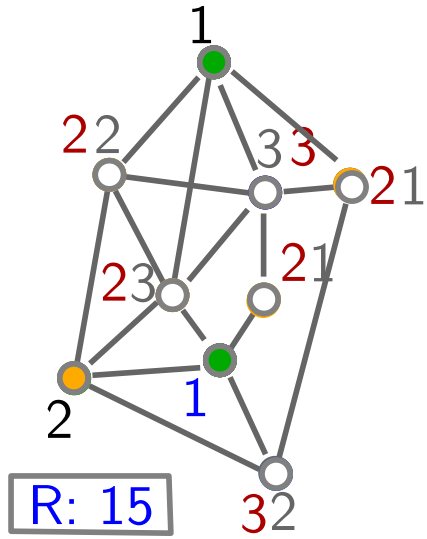
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

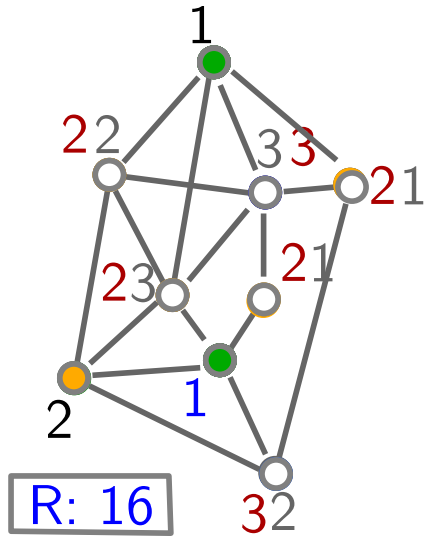
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

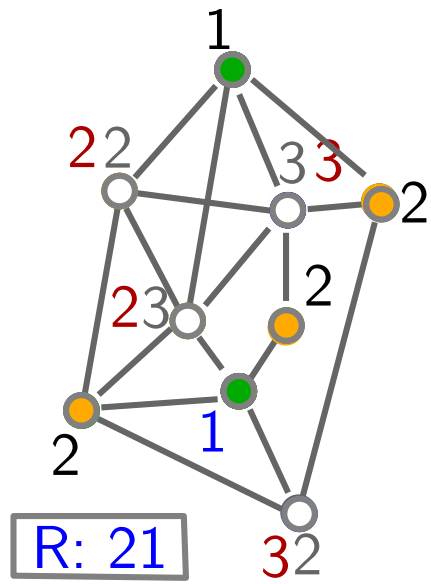
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

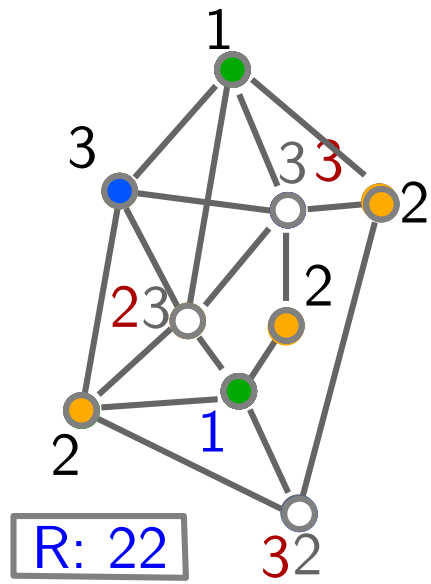
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

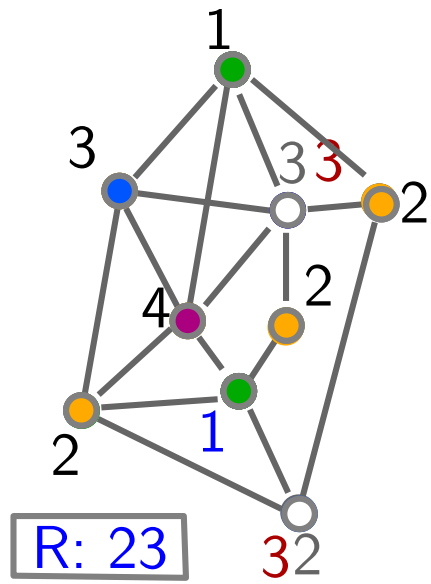
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

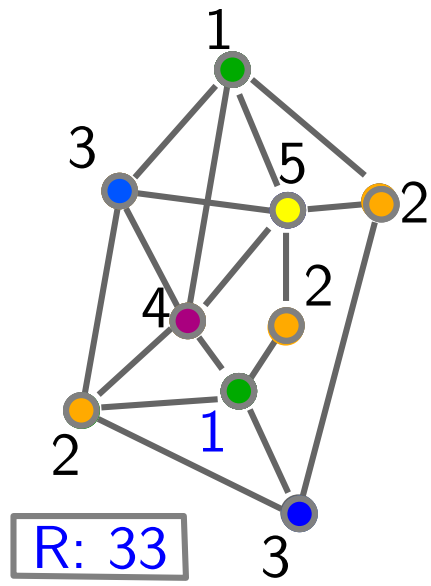
- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat



# Färbungen zusammenführen

Idee: Wir blenden nacheinander die Kanten der Pseudo-Wälder ein und *warten* eine Färbung mit $\Delta + 1$ Farben!

- Wir beginnen mit $F_1$ und dessen 3 Farben
  - das ist eine $\Delta + 1$-Färbung von $F_i$
- die 3-Färbung von $F_j$ und die $\Delta + 1$-Färbung von $F_1 \cup \cdots \cup F_{j-1}$ ergeben $3(\Delta + 1)$ „Mischfarben“ $(1, 0), \ldots, (3, \Delta + 1)$
- in $3(\Delta + 1)$ Runden können immer Knoten mit einer bestimmten Farbe eine einfache Farbe aus $\{1, \ldots, \Delta + 1\}$ wählen, die keiner der Nachbarn hat
- das sind $\Delta \cdot 3\Delta = 3\Delta^2$ Runden zum Zusammenführen



# Wo stehen wir jetzt?

- Wir können den Graphen in $\Delta$ Schritten in $\Delta$ Pseudo-Wälder dekomponieren
- Wir können Bäume und Pseudo-Wälder verteilt mit 3 Farben einfärben
  - erst färben wir in $\log^*(n)$ Runden mit konstant vielen Schritten 6-farbig,
  - dann reduzieren wir die Farben in 3 Runden mit konstant vielen Schritten auf 3

$\Rightarrow$ Wir können in $O(\log^*(n))$ Schritten jeden der Pseudo-Wälder 3 dreifärben

  - das kann ja in allen Pseudowäldern parallel geschehen

- wir können aus den 3-Färbungen für die $\Delta$ Pseudowälder in $O(\Delta^2)$ Schritten eine $\Delta + 1$-Färbung des eigentlichen Graphen gewonnen!
- wir sind fertig, Laufzeit insgesamt: $O(\Delta^2 + \log^* n)$

# Zusammenfassung

- Färbungen sind ein leichtes Mittel, um gegenseitige Ausschlüsse zu modellieren
  - Knoten dürfen nicht gleichzeitig senden oder
  - Kanten dürfen nicht gleichtzeitig aktiv sein
  - das lässt sich immer auf Knotenfärbungen reduzieren
- wir haben einen schnellen, verteilten und überhaupt nicht trivialen Algorithmus für $\Delta + 1$-Färbungen in $O(\Delta^2 + \log^* n)$ Zeit kennengelernt!

# Literatur


1. S. Schmid, R. Wattenhofer: *Algorithmic Models for Sensor Networks*. In: *14th International Workshop on Parallel adn Distributed Real-Time Systems (WPDRTS), 2006*

2. P. Gupta, P. R. Kumar: *The Capacity of Wireless Networks*, Technical report, University of Illinois, Urbana-Champaign, 1999

3. T. Moscibroda: *The Worst-Case Capacity of Wireless Sensor Networks*. In: *International Conference on Information Processing in Sensor Networks*, 2007

4. A. Goldberg, S. Plotkin, G. Shannon: *Parallel symmetry-breaking in sparse graphs*. In: Proceedings of the nineteenth annual ACM symposium on Theory of computing, 1987.