Qualität des Chang/Roberts-Algorithmus

- *Satz* (o. Bew.):

Für unidirektionale Ringe der Grösse n und n Initiatoren ist n H_n die optimale mittlere Nachrichtenkomplexität beim *Election-Problem*

- mit anderen Worten: es gibt keinen anderen Algorithmus, der "besser" ist!

--> Chang/Roberts-Algorithmus ist "average case optimal"!

- *Satz* (Rotem et al., o. Bew.):

Für die Nachrichtenzahl m des Chang/Roberts-Algorithmus gilt für alle ε: $\lim_{n \to \infty}$ prob $(m < (1+\varepsilon)\, n\, H_n) = 1$

Interpretation: Bei grösseren Ringen fast nie mehr als n H_n Nachrichten...

Tschebyscheff- / Chernoff-Ungleichungen

- Die Nachrichtenzahl des Chang/Roberts-Algorithmus lässt sich auch genauer abschätzen:

- aus der *Tschebyscheff-Ungleichung* (engl.: "Chebyshev") folgt:

$$\text{prob}\,(m \geq (1+\varepsilon)\, n\, H_n) \leq \frac{\pi^2/6}{\varepsilon^2\, H_n^{\,2}}$$

(vgl. dazu Lehrbücher zur Wahrscheinlichkeitstheorie)

- noch besser lässt sich dies mit der *Chernoff-Ungleichung* abschätzen:

$$\text{prob}\,(m \geq (1+\varepsilon)\, n\, H_n) \leq \exp(-\varepsilon^2\, n H_n/3)$$

- Einige Beispiele (Wahrscheinlichkeit, dass die Nachrichtenzahl unter dem angegebenen Wert liegt):

	Tschebyscheff	*Chernoff*
ε=1, n=10	0.31	0.00046
ε=1, n=100	0.078	10^{-67}
ε=0.5, n=100	0.31	10^{-17}
ε=0.1, n=100	--	0.2

- die Tschebyscheff-Ungleichung ist also recht konservativ!
- auch die Chernoff-Ungleichung ist nur eine Abschätzung; *die Nachrichtenzahl liegt also fast immer sehr nahe am Mittelwert!*

Zur Chernoff-Ungleichung vergleiche man z.B.:

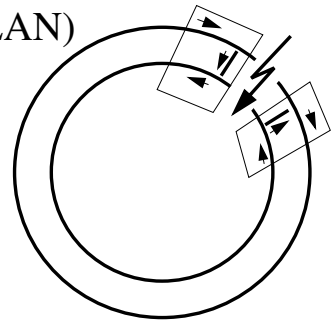
- C. Lavault: Evaluation des algorithmes distribués. Hermes, Paris 1995
- T. Hagerup. C. Rüb: A Guided Tour of Chernoff Bounds. Inform. Proc. Lett. 33, 305-308, 1990

Fehlertoleranz?

- Was geschieht bei Fehlern? ← z.B. beim Election-Algorithmus
  - was für Fehler können auftreten?
  - welche Fehlerarten betrachtet man? --> *Fehlermodell*
- Es werden typischerweise bei verteilten Systemen verschiedene *Fehlerarten* betrachtet (die unterschiedlich schwer zu behandeln sind), z.B.:
  - *crash* oder *failstop*: Prozess stoppt und bleibt gestoppt
  - *omission* (oder *lossy channel*): Nachricht geht unterwegs verloren
  - *link failure*: Kommunikationsverbindung geht (dauerhaft) kaputt
  - *byzantine*: Prozess verhält sich beliebig falsch (generiert illegale Nachrichten,...)
  - *timing* oder *performance*: bei synchronen bzw. Realzeitsystemen
- *Fehlertolerante* Algorithmen sollen robust bezüglich des betrachteten Fehlermodells sein
- Ob man überhaupt Fehler in Betracht zieht und welches Fehlemodell adäquat ist, ist ein *pragmatischer* Aspekt
  - abhängig von den Anforderungen, der Systemumgebung, Einsatzgebiet...
- Problem der *Fehlerentdeckung*: wie unterscheidet man:
  - Ausfall eines benachbarten Prozesses,
  - Ausfall der Kommunikationsverbindung,
  - extreme Nachrichtenverzögerung (länger als der timeout)?
- Fehlerentdeckung in der Praxis i.a. über *timeouts*
  - z.B. kein Acknowledge innerhalb eines bestimmten Zeitintervalls
  - dann *vermutet* man einen Fehler bzw. *verdächtigt* einen Prozess, ausgefallen zu sein (damit man sich da nicht täuscht, ist es manchmal besser, einen so verdächtigen Prozess nochmals explizit auszuschalten!)

Fehlerverhalten des Election-Algorithmus?

- Bei Ausfall einer Kommunikationsverbindung im Ring kann der Algorithmus nicht mehr funktionieren
- In der Praxis (z.B. Token-Ring-LAN) verwendet man daher meistens "Doppelringe"
  - Nachbarstationen schliessen den Ring kurz
  - zweite Unterbrechung wird so allerdings nicht toleriert!



- Konsequenz von Prozessausfällen?
  - in der Praxis hat das Ringinterface einer Station ein "Bypass-Relais" (damit wird die ausgefallene Station einfach kurzgeschlossen)
- Aber: Was geschieht bei folgendem Szenario?
  - Prozess mit der höchsten Identität sendet eine Nachricht aus
  - crasht (und wird kurzgeschlossen), bevor er "seine" Antwort vom Ring nehmen kann
  - diese kreist nun ständig weiter --> Algorithmus terminiert nicht!
- Denkübungen:
  - wie kann man obigen Fall verhindern (den Algorithmus also tolerant gegenüber diesem Fall machen)?
  - was geschieht, wenn ein anderer Prozess entsprechend ausfällt?
  - wie macht man den Algorithmus fehlertolerant gegenüber sporadisch verlorenen Nachrichten?

Praxisrelevanz

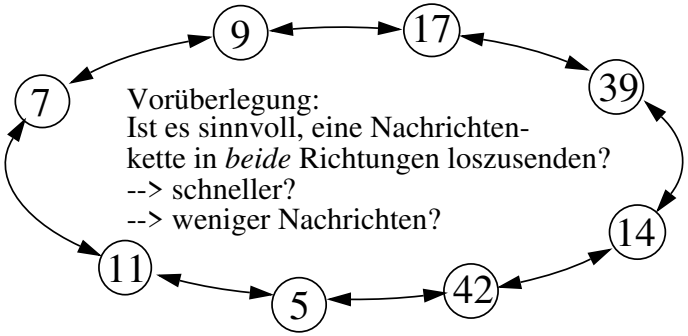
- Für die Praxis sind diverse Aspekte von Algorithmen relevant
  - z.B. Fehlertoleranz bzgl. Verlust von Nachrichten
  - oder: Crash eines Prozesses, der mitten im Senden ist (was geschieht mit Nachrichtenbruchstücken?)
  - oder: Erkennen verfälschter Nachrichten

- Denkübung: Kann beim Election-Verfahren der Fall behandelt werden, dass zwei verschiedene Stationen (versehentlich) die gleiche Identität haben?

- Der Election-Algorithmus muss wiederholt ausführbar sein
  - was geschieht, wenn in einer Ausführung des Algorithmus alte (sehr langsame) Nachrichten einer früheren Ausführung "dazwischenfunken"?
  - wie lässt sich dieses Problem behandeln?

==> Einfache algorithmische Ideen werden in der Praxis oft in komplexe Algorithmen eingebettet, um den ganzen pragmatischen Ansprüchen gerecht zu werden!

Probabilistisches Election-Verfahren für bidirektionale Ringe



- Message-extinction-Prinzip in naheliegender Weise verallgemeinern:

I_p: {M = 0}
M := p;
Wähle mit Wahrscheinlichkeit 1/2 eine Richtung;
send <M> **to** Nachbar in diese Richtung ;

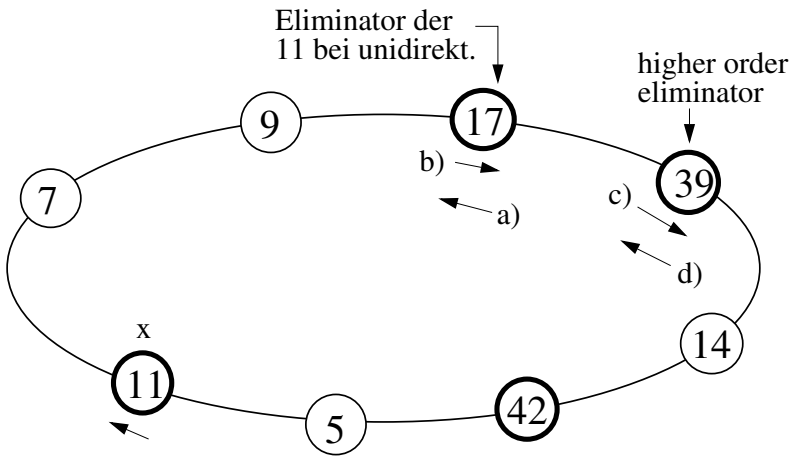
Identität des Prozesses

Uhrzeigersinn oder Gegen-Uhrzeigersinn

R_p: {Eine Nachricht <j> ist eingetroffen}
if M < j **then**
  M := j;
  send <M> **to** ... /* an anderen Nachbarn */
fi /* *weiter*senden */
if j = p **then** *"I am the master"* **fi**

- ist Wahrscheinlichkeit 1/2 eine gute Wahl?
- geht es nicht besser deterministisch als probabilistisch?
- wie hoch ist die mittlere Nachrichtenkomplexität?

Mittlere Nachrichtenkomplexität



- Annahme: Jede Einzelnachricht braucht gleich lang

- In der Hälfte aller Fälle kommt der Eliminator der Nachricht "x" auf halbem Weg entgegen
  - z.B. Fall a) bei den Knoten 11 und 17
  --> sollte 1/4 aller Nachrichten (gegenüber unidirekt. Fall) sparen (wieso?)

- Aber: höchste läuft immer ganz durch
  (jedoch: spielt für n --> ∞ eine "asymptotisch geringe" Rolle)

- Asymptotische mittlere Nachrichtenkomplexität ist *geringer* als 0.75 n ln n (Grund: "Higher order eliminators")
  - z.B. 39 verkürzt den Weg der 11 "etwas" im Fall b), d)
  - 42 als higher order eliminator würde hier aber nichts nützen!

Resultat von Lavault (Beweis schwierig!):
Asymptot. Nachrichtenkomplexität n --> ∞ ist

$$0.5\sqrt{2}\ n \ln n \approx 0.7071\ n \ln n$$

Als *untere Schranke* für das bidir. Election-Problem kennt man 0.5 n ln n (--> "Lücke")

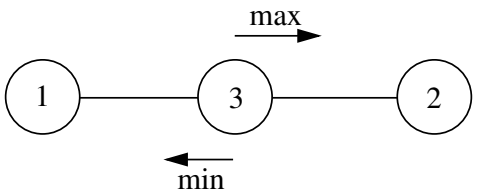
Deterministische bidirektionale Verfahren

(1) "Gerader" Prozess startet im Uhrzeigersinn, andere gegen der Uhrzeigersinn

- "common sense of orientation" vorhanden?
- aber: wenn Identifikationen keine ganze Zahlen (sondern z.B. rationale)?

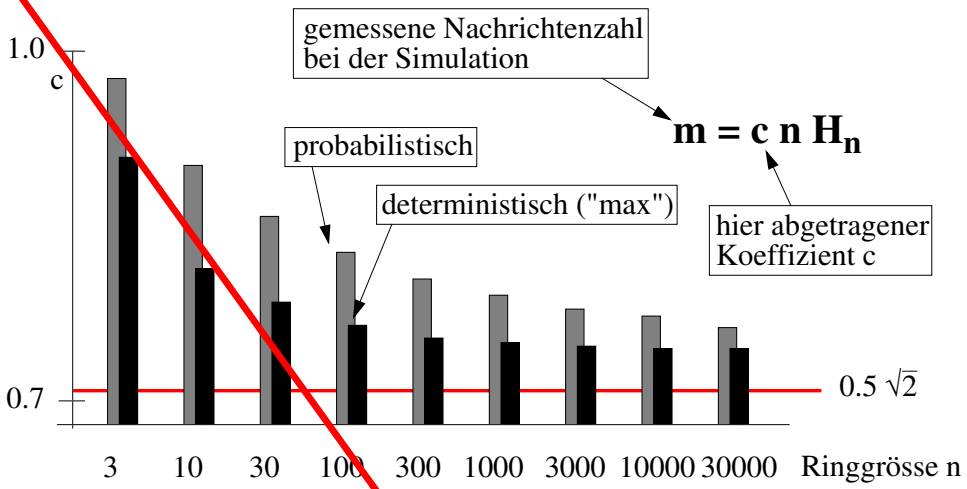
(2) Starten in Richtung des kleineren Nachbarn ("min")

- kennt man Identität der Nachbarn? (wenn nicht, was dann?)
- wieso nicht in Richtung des *grösseren* Nachbarn ("max")?
- was bringen die deterministischen Verfahren "min", "max" im Vergleich zum probabilistischen Verfahren?



Startsituation bzgl. Knoten 3

Gemessene Nachrichtenkomplexität



- Simulationsergebnisse:

  - Es stellt sich heraus, dass "min" und "max" bzgl. der mittleren Nachrichtenkomplexität etwa identisch sind (!), dass diese Varianten jedoch etwas besser sind als die probabilistische Version (und diese, wie gezeigt, besser als die unidirektionale)

  - Simulationen zeigen auch, dass die asymptotische Nachrichtenkomplexität des probabilistischen Verfahrens lediglich ein theoretisches Ergebnis ist und der Faktor 0.7071... sehr langsam (mit steigendem n) approximiert wird

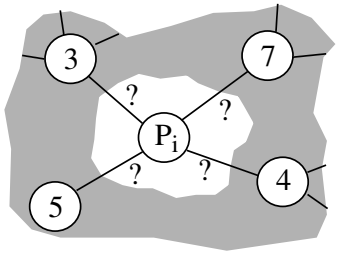
  - Ferner zeigen die Simulationen, dass die *Abweichungen vom Mittelwert* $n H_n$ bzgl. der Nachrichtenkomplexität i.a. nur sehr gering sind; 100000 Simulationen bei einer Ringgrösse von 20 lieferten z.B. stets Nachrichtenzahlen unter $2 n H_n$

  - Beachte bei *Simulationsexperimenten*: sehr viele Einzelexperimente (Varianz, statistisch relevante Ergebnisse) sowie guter Zufallszahlen-generator notwendig

Nachbarschaftswissen

- Zweck: Lerne Identitäten Deiner Nachbarn kennen
  (Idee: "Ich heisse i, wie heisst Du?")

- Hier nachrichtengesteuert für P_i:
  (vgl. verteiltes Approximationsschema)



Netz zusammenhängend
mit bidirektionalen Kanten

```
{Nachricht <j> kommt an}
if not vorgestellt then
    send <i> to all neighbors;
fi;
Nachbarn := Nachbarn ∪ {j};
vorgestellt := true;
```

mehrere?
Einer muss "spontan" mit
der Vorstellung beginnen

- *global terminiert*, wenn
  - vorgstellt = true bei allen Prozessen und keine Nachricht unterwegs
  - *oder*: alle kennen alle

- *lokal terminiert* (genügt meistens!), wenn zu jeder inzidenten Kante die zugehörigen Nachbarknoten bekannt sind

- 2e Nachrichten (bei Ring also 2n); Zeitkomplexität?

Verwendung als:

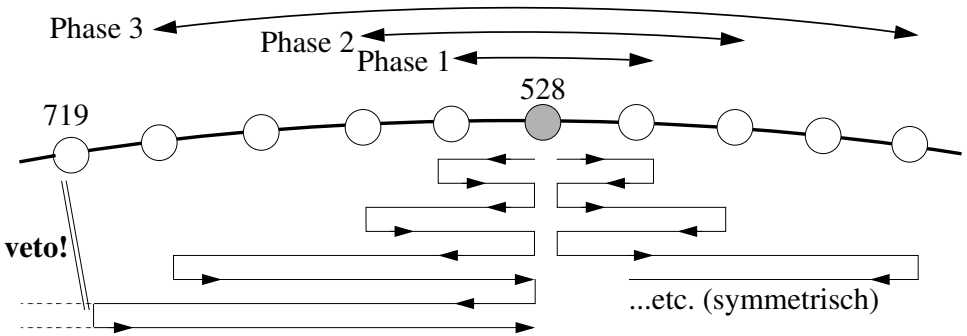
- vorgeschalteter eigener Algorithmus
- "piggybacking" der Vorstellungsnachrichten mit eigentlichen (jew. "ersten") Anwendungsnachrichten

- lohnt sich das bei bidirektionaler Election, um so den als "min" oder "max" bezeichneten Algorithmus einsetzen zu können?

Hirschberg / Sinclair-Election-Algorithmus

- Idee: Jeder Knoten versucht, sukzessive Gebiete der Grösse 2^i (i=1,...) zu erobern

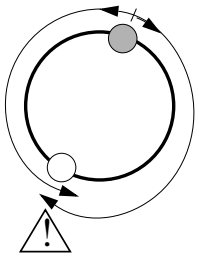
bidirektionaler Ring!



- Ein unterwegs angetroffener grösserer Knoten legt Veto ein
  --> Initiator über Rückmeldung informieren
  --> Initiator wechselt von *aktiv* nach *passiv*

nur noch Nachrichten weiterleiten ("relay")

Gewinnsituation:



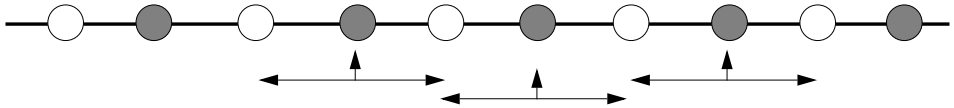
- Nachricht läuft in bereits selbst erobertes Gebiet
- *oder*: Nachricht trifft bei Initiator selbst wieder ein
- es bleibt genau ein Gewinner! (wieso?)

Komplexitätsanalyse

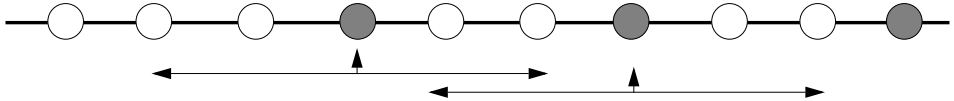
- Ein Prozess kann nur dann eine Kette der Länge 2^i starten, wenn er im Abstand 2^{i-1} in beiden Richtungen überlebt hat
  - Dichte überlebender Prozesse nimmt also exponentiell ab
- Innerhalb eines Bereiches von $1+ 2^{i-1}$ benachbarter Prozesse kann also höchstens einer eine Kette der Länge 2^i starten

---

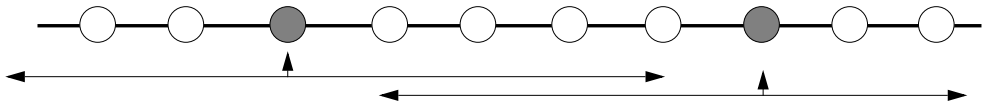
- Jeweils 1 dazwischenliegender Prozess nach Phase 1:



- Jeweils 2 dazwischenliegende Prozesse nach Phase 2:



- Jeweils 4 dazwischenliegende Prozesse nach Phase 3:



- n/2 Prozesse können Ketten der Länge 2 initiieren
- n/3 Prozesse können Ketten der Länge 4 initiieren
- n/5 Prozesse können Ketten der Länge 8 initiieren
  ...
- $n/(1+2^{i-1})$ Prozesse können Ketten der Länge 2^i initiieren

Maximal 8 n $\log_2$ n Nachrichten

- Also: höchstens $n/(1+2^{i-1})$ Prozesse initiieren eine Nachrichtenkette der Länge 2^i in Phase i

- Bei jeder solchen Kette wird jede Kante max. 4 Mal durchlaufen

- In Phase i gibt es also höchstens $4 \times 2^i \times n / (1+2^{i-1}) < 8n$ Nachrichten

- Es gibt höchstens $1+\lceil \log_2 n \rceil$ Phasen

==> ca. 8 n $\log_2$ n $\approx$ 5.55 n ln n Nachrichten *maximal*

(Worst-case-Komplexität!)

- Aber: Wie hoch ist die *mittlere* Nachrichtenkomplexität?

- *Zeitkomplexität*: $2 + 4 + 8 + 16 + ... + 2^i < 2^{i+1} \approx 4n$

Vergleiche dies alles mit dem Chang/Roberts-Algorithmus

- welcher Algorithmus ist "in der Praxis" besser?

Synchrone <--> asynchrone Phasen

- Die Phasen der einzelnen Initiatoren müssen nicht unbedingt synchron laufen!

- Damit der Algorithmus dann noch gut funktioniert, vereinbare folgendes:

  - anstelle von Knotenidentitäten betrachtet man das Paar (Phasennummer, Knotenidentität)

  - diese Paare werden lexikographisch geordnet (eine höhere Phasennummer hat also Priorität!)

- Konsequenz: Ein "schneller" Initiator gewinnt gegenüber einem "langsamen" Initiator mit höherer Identität

- Es gewinnt also zwar ein eindeutiger Knoten die Election, das muss aber nicht derjenige mit der grössten Identität sein!

- Unterscheide also:

  - *leader election problem*
  - *maximum finding problem*

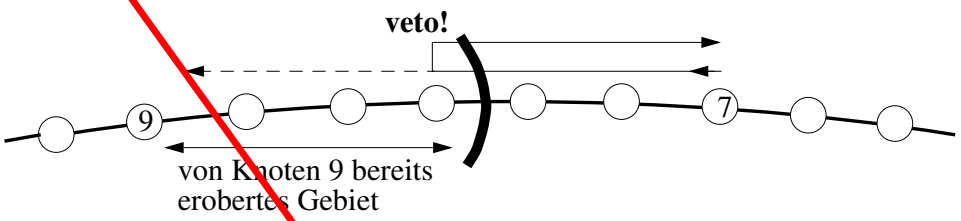
- eine Lösung des maximum finding problems ist immer auch eine Lösung des leader election problems (sofern die Knoten eindeutig numeriert sind)

- Umkehrung?

Optimierungen und Varianten

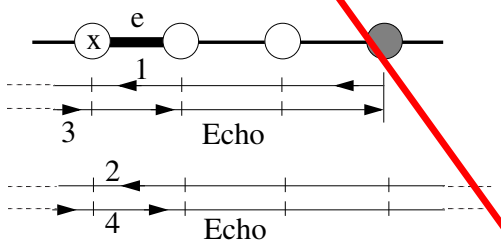
1.) *Auch passive Knoten legen Veto ein:*

(sofern ein grösserer Eroberer zuerst da war) --> verkürzt Nachrichtenketten



- hierzu muss sich jeder Knoten merken, von wem er (zuletzt) erobert wurde (inkl. der Phasennummer)

2.) *Zwei "Echos" zusammenfassen, wenn möglich:*



- Optimierung: Wenn x erst Nachricht 1 erhält, dann Nachricht 2 (vor dem "Echo" 3), dann kann x die beiden Nachrichten, die über e zurückgeschickt werden, zusammenfassen zu einer einzigen physischen Nachricht

---

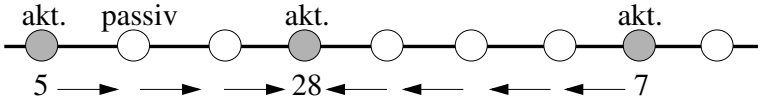
Wie wirken sich diese Optimierungen auf die Nachrichtenkomplexität aus?
(Wer simuliert den Algorithmus und ermittelt die Nachrichtenkomplexität?)

Peterson's Election-Algorithmus (1. Variante)

- Prinzip analog zum Hirschberg / Sinclair - Verfahren: Anzahl aktiver Knoten pro "Phase" mindestens halbieren
  - bidirektionaler Ring
  - anfangs sind alle *aktiv*
  - *passive* Knoten reichen nur noch Nachrichten weiter ("relay")

- Idee: Pro Phase bekommt ein Knoten die Identitäten seiner rechten und linken *noch aktiven* Nachbarn...

Vgl. dies mit iterierter Anwendung des Algorithmus für Nachbarschaftswissen!



...und überlebt nur, wenn er der grösste *aller drei* ist!

Im Unterschied zu Hirschberg / Sinclair gibt es keine Echos / Vetos!

- Ein Überlebender bleibt aktiv und startet eine neue Phase: Sendet seine Identität in beide Richtungen
  (Initial tun das alle Initiatoren)

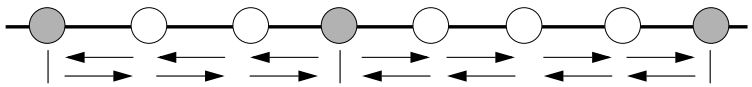
- *Gewonnen*, wenn die eigene Identität empfangen wird

Beachte: In obigem Beispiel wird die 5 von der 28 "passiviert". Bald darauf (in der nächsten Phase) erhält die 5 erneut eine Nachricht "28", um diese weiterzuleiten. Hätte die 5 nicht gleich beim ersten Mal die "28" einfach weiterleiten sollen, so dass Knoten 28 die Nachricht nicht erneut über die Strecke 28 --> 5 senden muss? (Vgl. Chang/Roberts!) Nein! Knoten 5 weiss nicht, ob die 28 die gegenwärtige Phase tatsächlich überlebt - dann wäre die Nachricht "28" fälschlicherweise weitergeleitet worden!

Zeitkomplexität des Algorithmus als Übung
(dominiert auch hier die letzte Phase?)

Nachrichtenkomplexität

- Pro Phase laufen 2 Nachrichten über *jede* Kante
  - für global *synchrone Phasen* leicht einsichtig
  - aber auch für nicht synchronisierte Phasen richtig!



- Wenn ein Knoten Phase i überlebt, überlebt sein "linker" aktiver Nachbar diese Phase nicht!

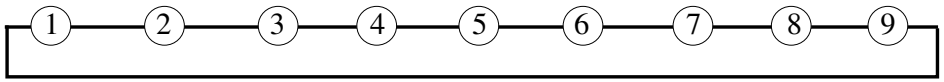
  --> max. $\log_2 n$ Phasen

  --> max. $2 n \log_2 n$ Nachrichten

"in jeder Phase überlebt einer von *dreien*" ist *falsch* - wieso?

  - wie sieht eine Anordnung aus, bei der *maximal viele* Nachrichten entstehen?

- *Sortierte Anordnung*: jeweils durch "rechten" Nachbarn eliminiert, ausgenommen grösster Knoten im Ring



--> in diesem Fall nur 2 Phasen ==> 4n Nachrichten!
(beachte Terminierungserkennung!)

- Mittlere Nachrichtenkomplexität:
  - für jeden Knoten Wahrscheinlichkeit 1/3, Phase zu überleben (wieso?)
  - also im Mittel $\log_3 n$ Phasen

  --> $2 n \log_3 n \approx 1.26 n \log_2 n \approx 1.82 n \ln n$ Nachrichten

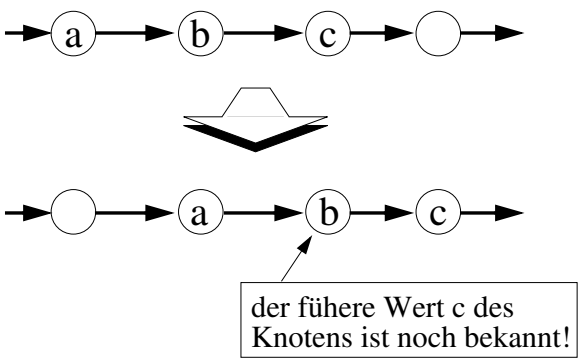
Variante für unidirektionale Ringe

- Man glaubte zunächst, dass ein Election-Algorithmus auf *unidirektionalen* Ringen mindestens $O(n^2)$ Nachrichten im Worst-case-Fall benötigt

- Das stimmt nicht: Der Peterson-Algorithmus lässt sich auf unidirektionalen Ringen *simulieren*!

---

1. "Shift" in Ringrichtung um eine Position bzgl. aktiver Knoten:



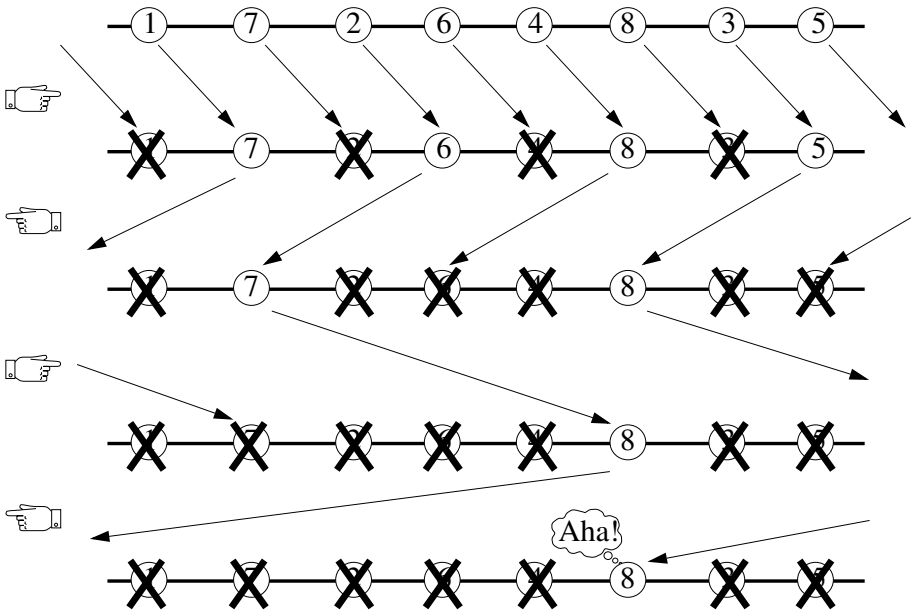
2. Nun kann (der neue) Knoten a an seinen Nachbarn b seinen Wert senden - damit kennt b sowohl a als auch c
(als hätte b Nachrichten von a und c erhalten!)

Damit kostet eine solche Phase global auch nur 2n Nachrichten!

Peterson's Election-Algorithmus (2. Variante)

- Idee einer Optimierung:
  - anstatt sich mit beiden Nachbarn "gleichzeitig" zu vergleichen, sollte ein Knoten sich nur dann mit seinem anderen Nachbarn vergleichen, wenn er den ersten Vergleich gewonnen hat

- Phasen im / gegen den Uhrzeigersinn wechseln sich ab:



  - lässt sich auch wieder unidirektional simulieren!
  - in jeder Phase werden n Nachrichten gesendet (passive Knoten: "relay")

- Denkübungen:
  1) Wie kann man auch bei asynchronen Nachrichten und nicht gleichzeitigem Start der Knoten "eine Art" global getakteter Phasen erreichen?
  2) Man formuliere den Algorithmus aus "Sicht eines Knotens": Wie reagiert ein Knoten auf das Eintreffen einer bestimmten Nachricht?
  3) Man mache sich Gedanken zur Abschätzung der worst-case und der average-case Nachrichtenkomplexität!

Nachrichtenkomplexität

- **Behauptung:**
  Für die *Anzahl der Phasen* r gilt: $r \leq \log_{\varphi} n + O(1)$ (Basis $\varphi = (1+\sqrt{5})/2$)
  ==> Anzahl der Nachrichten $\leq 1.44\ n \log_2 n + c$

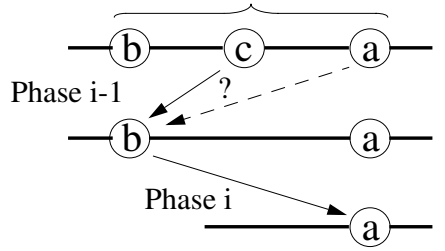
- **Lemma:** $a_i \leq a_{i-2} - a_{i-1}$ (für $i > 1$)

Def: Anzahl Überlebender von Phase i

Anzahl der "Opfer" von Phase i-1

Bew.: Betrachte zwei benachbarte Knoten a, b in Phase i

Gab es einen Knoten zwischen a und b in Phase i-1?



(1) a überlebe Phase i
==> a > b

(2) b hat Phase i-1 überlebt
==> b > c

(1) und (2) ==> $a \neq c$

Also muss es ein c geben, das in Phase i-1 Opfer wurde
Hier: a hat seinen linken Nachbarn in der vorherigen Phase verloren

==> Für jeden Überlebenden in Phase i (hier: a) gibt es mindestens ein Opfer (hier: c) in Phase i-1 □

- Aus $a_i \leq a_{i-2} - a_{i-1}$ folgt $a_{i-2} \geq a_{i-1} + a_i$, also $a_i \geq a_{i+1} + a_{i+2}$
- Ferner gilt $a_{r-1} = 1 = Fib(2)$, $a_{r-2} \geq 2 = Fib(3)$ } $a_{r-3} \geq a_{r-2} + a_{r-1} \geq Fib(2)+Fib(3) = Fib(4)$
- Also: $n = a_0 \geq Fib(r+1)$

--> Ungleichung nach r auflösen!
Weil Fib exponentiell zur Basis φ wächst ($Fib(k) \approx \varphi^k / \sqrt{5}$), folgt die Behauptung