

Proseminar Verteilte Algorithmen, WS 2006/2007  
**Das "Cheating Husbands"-Puzzle**  
eine Einführung

David Rasmus Piegdon

Betreut durch Daniel Willems

Lehrstuhl für Informatik II, RWTH Aachen

<http://www-i2.informatik.rwth-aachen.de>

6. Februar 2006

# Gliederung

- 1 Einführung
- 2 Cheating Husbands
- 3 Synchronisationsprobleme
- 4 Quick Elimination
- 5 Ergebnisse

# Gliederung

- 1 Einführung
- 2 Cheating Husbands
- 3 Synchronisationsprobleme
- 4 Quick Elimination
- 5 Ergebnisse

# Verteilte Systeme

## Zentrale Eigenschaften:

- gut skalierbar
- beliebig parallelisierbar
- ausfallsicherer
- vielversprechendes Anwendungsgebiet
- indeterministisch

# Verteilte Systeme

Zentrale Eigenschaften:

- gut skalierbar
- beliebig parallelisierbar
- ausfallsicherer
- vielversprechendes Anwendungsgebiet
- **indeterministisch**

# Verteilte Systeme

Indeterministisch durch

- Nebenläufigkeit
- unvollständiges Wissen über Gesamtzustand
- fehlende Synchronisation von Komponenten

# Modellierung

- Nicht „klassisch“ Modellierbar
- Neue Modelle gesucht (ca. 1980-1985)
- Zum Beispiel:
  - Wissen einzelner Komponenten
  - Aktionen einzelner Komponenten
  - Kommunikation von Komponenten untereinander

( → „Transitionssysteme“ )

# Modellierung

- Nicht „klassisch“ Modellierbar
  - Neue Modelle gesucht (ca. 1980-1985)
  - Zum Beispiel:
    - Wissen einzelner Komponenten
    - Aktionen einzelner Komponenten
    - Kommunikation von Komponenten untereinander
- ( → „Transitionssysteme“ )

# Gliederung

- 1 Einführung
- 2 Cheating Husbands**
- 3 Synchronisationsprobleme
- 4 Quick Elimination
- 5 Ergebnisse

# Das „Cheating Husbands“-Problem

Ein philosophisches Rätsel

- Dorf mit  $f$  verheirateten Frauen
- Frauen sind der perfekten Logik fähig
- $m$  Frauen werden vom Ehemann betrogen
- Frauen wissen **nur** vom **eigenen Mann nicht**, ob er treu ist

# Die Königin befiehlt

- versammelt alle Frauen auf dem Marktplatz
- „Mindestens ein Ehemann betrügt seine Frau.“
- „Jede von euch weiß, welche Männer untreu sind, ausgenommen euren eigenen.“
- „Kommunikation über die Treue der Ehemänner ist untersagt.“
- „An dem Tag, an dem ihr herausfindet, dass euer Mann untreu ist, erschießt ihr ihn um Mitternacht.“
- Alle diese Punkte sind nun Allgemeinwissen

# Synchronisationsproblem

Synchronisation zwischen allen Ehefrauen:

- $m$  untreuen Männer
- betrogene Frauen kennen  $\lfloor m \rfloor := m - 1$
- nicht betrogene Frauen kennen  $\lceil m \rceil := m$

→ Synchronisation einer Int.Variable  $m$  mit  
max. Differenz 1 zwischen allen Komponenten

# Etwas abstrakter...

- Verteiltes System (Frauen  $\simeq$  Komponenten)
- Integer-Variable  $m$
- $d$  „ $\lfloor m \rfloor$ -Komponenten“ kennen  $\lfloor m \rfloor$
- die anderen („ $\lceil m \rceil$ -Komponenten“) kennen  $\lceil m \rceil$
- Ein **geteiltes** Speicherbit ( $\simeq$  hörbarer Schuss)

# Behauptung

## Behauptung

*$d$   $\lfloor m \rfloor$ -Komponenten können in der  $d$ -ten Runde erkennen, dass sie  $\lfloor m \rfloor$ -Komponenten sind*

Beweis per vollständiger Induktion. . .

# Induktionsstart

$d := 1$

- $\lfloor m \rfloor$ -Komponente kennt  $d - 1 = 0$
- aber mindestens eine  $\lfloor m \rfloor$ -Komponente existiert ( $d \geq 1$ )

→ sie selber ist  $\lfloor m \rfloor$ -Komponente

→ setzt das geteilte Bit und lokal  $m := m + 1$

# Induktionsvoraussetzung

- $d$   $[m]$ -Komponenten erkennen sich selbst nach  $d$  Runden

# Induktionsschritt

- gibt es  $d + 1$   $\lfloor m \rfloor$ -Komponenten, so wissen diese Komponenten von  $d$  Komponenten
  - Erwartung (IV): in Runde  $d$  wird das Speicherbit gesetzt
  - nichts passiert, daher muss es  $d + 1$   $\lfloor m \rfloor$ -Komponenten geben
  - sie setzen das Bit in Runde  $d + 1$  und lokal:  $m := m + 1$
- Behauptung folgt per Vollst. Induktion

# Induktionsschritt

- gibt es  $d + 1$   $\lfloor m \rfloor$ -Komponenten, so wissen diese Komponenten von  $d$  Komponenten
- Erwartung (IV): in Runde  $d$  wird das Speicherbit gesetzt
- nichts passiert, daher muss es  $d + 1$   $\lfloor m \rfloor$ -Komponenten geben
- sie setzen das Bit in Runde  $d + 1$  und lokal:  $m := m + 1$

→ Behauptung folgt per Vollst. Induktion

# Induktionsschritt

- gibt es  $d + 1$   $\lfloor m \rfloor$ -Komponenten, so wissen diese Komponenten von  $d$  Komponenten
- Erwartung (IV): in Runde  $d$  wird das Speicherbit gesetzt
- nichts passiert, daher muss es  $d + 1$   $\lfloor m \rfloor$ -Komponenten geben
- sie setzen das Bit in Runde  $d + 1$  und lokal:  $m := m + 1$

→ Behauptung folgt per Vollst. Induktion

# Induktionsschritt

- gibt es  $d + 1$   $\lfloor m \rfloor$ -Komponenten, so wissen diese Komponenten von  $d$  Komponenten
- Erwartung (IV): in Runde  $d$  wird das Speicherbit gesetzt
- nichts passiert, daher muss es  $d + 1$   $\lfloor m \rfloor$ -Komponenten geben
- sie setzen das Bit in Runde  $d + 1$  und lokal:  $m := m + 1$

→ Behauptung folgt per Vollst. Induktion

# Induktionsschritt

- gibt es  $d + 1$   $\lfloor m \rfloor$ -Komponenten, so wissen diese Komponenten von  $d$  Komponenten
  - Erwartung (IV): in Runde  $d$  wird das Speicherbit gesetzt
  - nichts passiert, daher muss es  $d + 1$   $\lfloor m \rfloor$ -Komponenten geben
  - sie setzen das Bit in Runde  $d + 1$  und lokal:  $m := m + 1$
- Behauptung folgt per Vollst. Induktion

# betrügerische Ehemänner

Bezogen auf das „Cheating-Husbands“-Problem:

- hier:  $d = m$
- Gibt es  $m$  untreue Männer, werden sie in der  $m$ -ten Nacht erschossen

# Gliederung

- 1 Einführung
- 2 Cheating Husbands
- 3 Synchronisationsprobleme**
- 4 Quick Elimination
- 5 Ergebnisse

## Verhalten des Algorithmus in verschiedenen verteilten Systemen?

- $K_{\lfloor m \rfloor} :=$  Menge aller  $\lfloor m \rfloor$ -Komponenten
- $K_{\lceil m \rceil} :=$  Menge aller  $\lceil m \rceil$ -Komponenten
- Ist  $\kappa$  Komponente, dann ist  $d_\kappa$  die  $\kappa$  bekannte Anzahl der  $\lfloor m \rfloor$ -Komponenten ( $d$  oder  $d - 1$ )

# Asynchrone Kommunikation

- Nachrichten kommen **auf jeden Fall** an
- es gibt keine obere Zeitschranke für die Lieferung

# Asynchrone Kommunikation

- Nachrichten kommen **auf jeden Fall** an
- es gibt **keine obere Zeitschranke** für die Lieferung

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : hat  $\beta$  schon Nachricht erhalten?
- $\beta$ : hat  $\alpha$  schon Nachricht erhalten?
- wann kommt Nachricht beim Gegenüber an?
- kann niemals herausgefunden werden

→  $\alpha$  und  $\beta$  müssen **unbestimmt lange** aufeinander warten!

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : hat  $\beta$  schon Nachricht erhalten?
- $\beta$ : hat  $\alpha$  schon Nachricht erhalten?
- wann kommt Nachricht beim Gegenüber an?
- kann niemals herausgefunden werden

→  $\alpha$  und  $\beta$  müssen **unbestimmt lange** aufeinander warten!

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : hat  $\beta$  schon Nachricht erhalten?
- $\beta$ : hat  $\alpha$  schon Nachricht erhalten?
- wann kommt Nachricht beim Gegenüber an?
- kann niemals herausgefunden werden

→  $\alpha$  und  $\beta$  müssen **unbestimmt lange** aufeinander warten!

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : hat  $\beta$  schon Nachricht erhalten?
- $\beta$ : hat  $\alpha$  schon Nachricht erhalten?
- wann kommt Nachricht beim Gegenüber an?
- kann niemals herausgefunden werden

→  $\alpha$  und  $\beta$  müssen **unbestimmt lange** aufeinander warten!

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : hat  $\beta$  schon Nachricht erhalten?
- $\beta$ : hat  $\alpha$  schon Nachricht erhalten?
- wann kommt Nachricht beim Gegenüber an?
- kann niemals herausgefunden werden

→  $\alpha$  und  $\beta$  müssen **unbestimmt lange** aufeinander warten!

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : hat  $\beta$  schon Nachricht erhalten?
- $\beta$ : hat  $\alpha$  schon Nachricht erhalten?
- wann kommt Nachricht beim Gegenüber an?
- kann niemals herausgefunden werden

→  $\alpha$  und  $\beta$  müssen **unbestimmt lange** aufeinander warten!

# Schwach-b-synchrone Kommunikation

- Nachrichten kommen **auf jeden Fall** an
- Obere Zeitschranke für Lieferung:  $(b - 1)$  Zeiteinheiten

## Theorem

*nach  $b \cdot (d - 1)$  Runden: es gibt  $d \lfloor m \rfloor$ -Komponenten*

- aber werden **alle**  $\lfloor m \rfloor$ -Komponenten erkannt?

# Schwach-b-synchrone Kommunikation

- Nachrichten kommen **auf jeden Fall** an
- Obere Zeitschranke für Lieferung:  $(b - 1)$  Zeiteinheiten

## Theorem

*nach  $b \cdot (d - 1)$  Runden: es gibt  $d \lfloor m \rfloor$ -Komponenten*

- aber werden **alle**  $\lfloor m \rfloor$ -Komponenten erkannt?

- $t_0$  := Runde, in der Startsignal geschickt wurde
- $t_S$  := erste Runde, in der eine  $[m]$ -Komponente Startsignal erhält
- $t_{r_\kappa}$  := Runde, in der  $\kappa$  das Startsignal erhält
- $t_{e_\kappa}$  := letzte Runde aus Sicht von  $\kappa$
- klar:  $t_0 \in [t_{r_\kappa} - (b - 1), t_{r_\kappa}]$
- also:  $t_S \in [t_{r_\kappa} - (b - 1), t_{r_\kappa} + (b - 1)]$
- Intervall ist von  $t_{r_\kappa}$  in beide Richtungen um  $(b - 1)$  Runden ausgedehnt

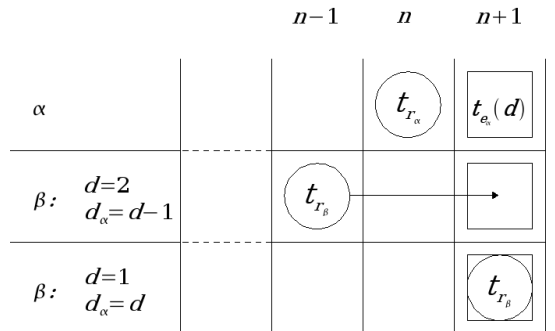
- $t_0$  := Runde, in der Startsignal geschickt wurde
- $t_s$  := erste Runde, in der eine  $[m]$ -Komponente Startsignal erhält
- $t_{r_\kappa}$  := Runde, in der  $\kappa$  das Startsignal erhält
- $t_{e_\kappa}$  := letzte Runde aus Sicht von  $\kappa$
- klar:  $t_0 \in [t_{r_\kappa} - (b - 1), t_{r_\kappa}]$
- also:  $t_s \in [t_{r_\kappa} - (b - 1), t_{r_\kappa} + (b - 1)]$
- Intervall ist von  $t_{r_\kappa}$  in beide Richtungen um  $(b - 1)$  Runden ausgedehnt

- $t_0$  := Runde, in der Startsignal geschickt wurde
- $t_s$  := erste Runde, in der eine  $[m]$ -Komponente Startsignal erhält
- $t_{r_\kappa}$  := Runde, in der  $\kappa$  das Startsignal erhält
- $t_{e_\kappa}$  := letzte Runde aus Sicht von  $\kappa$
- klar:  $t_0 \in [t_{r_\kappa} - (b - 1), t_{r_\kappa}]$
- also:  $t_s \in [t_{r_\kappa} - (b - 1), t_{r_\kappa} + (b - 1)]$
- Intervall ist von  $t_{r_\kappa}$  in **beide** Richtungen um  $(b - 1)$  Runden ausgedehnt

# Beispiel

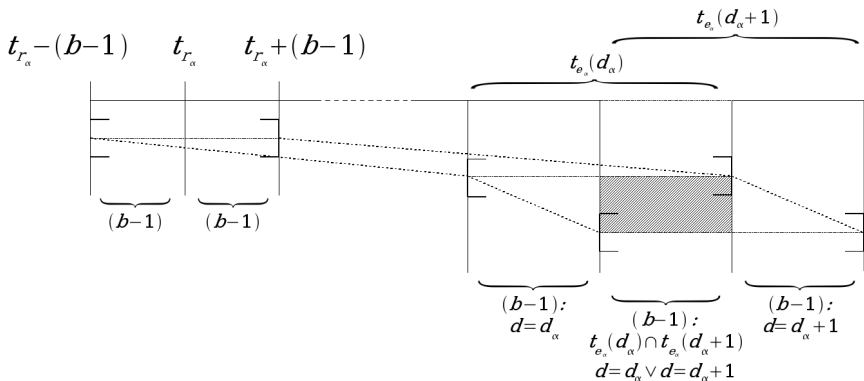
- $b := 2$
- Komponenten  $\alpha$  und  $\beta$
- $\alpha$  weiß, dass  $\beta \in K_{[m]}$
- $t_{e_\alpha} := t_{r_\alpha} + 1$

## Beispiel



- Hat  $\beta$  2 Runden gewartet, so ist  $\alpha$   $\lfloor m \rfloor$ -Komponente
- Hat  $\beta$  0 Runden gewartet, so ist  $\alpha$   $\lceil m \rceil$ -Komponente

## Allgemeiner...



- nebeneinander liegende Intervalle überschneiden sich um  $(b - 1)$  Runden

# aber...

Intervalle lassen sich „auseinanderziehen“

- Komponente findet heraus, dass sie  $\lfloor m \rfloor$ -Komponente ist
- wartet  $n \geq (b - 1)$  Runden
- setzt dann erst das geteilte Bit

→ Intervalle überlappen sich nicht mehr

Theorem

*nach  $d(b + n)$  Runden werden alle  $d$   $\lfloor m \rfloor$ -Komponenten erkannt*

# aber...

Intervalle lassen sich „auseinanderziehen“

- Komponente findet heraus, dass sie  $\lfloor m \rfloor$ -Komponente ist
- wartet  $n \geq (b - 1)$  Runden
- setzt dann erst das geteilte Bit

→ Intervalle überlappen sich nicht mehr

## Theorem

*nach  $d(b + n)$  Runden werden alle  $d$   $\lfloor m \rfloor$ -Komponenten erkannt*

## aber...

Intervalle lassen sich „auseinanderziehen“

- Komponente findet heraus, dass sie  $\lfloor m \rfloor$ -Komponente ist
- wartet  $n \geq (b - 1)$  Runden
- setzt dann erst das geteilte Bit

→ Intervalle überlappen sich nicht mehr

## Theorem

*nach  $d(b + n)$  Runden werden alle  $d$   $\lfloor m \rfloor$ -Komponenten erkannt*

# aber...

Intervalle lassen sich „auseinanderziehen“

- Komponente findet heraus, dass sie  $\lfloor m \rfloor$ -Komponente ist
- wartet  $n \geq (b - 1)$  Runden
- setzt dann erst das geteilte Bit

→ Intervalle überlappen sich nicht mehr

## Theorem

*nach  $d(b + n)$  Runden werden alle  $d$   $\lfloor m \rfloor$ -Komponenten erkannt*

## aber...

Intervalle lassen sich „auseinanderziehen“

- Komponente findet heraus, dass sie  $\lfloor m \rfloor$ -Komponente ist
- wartet  $n \geq (b - 1)$  Runden
- setzt dann erst das geteilte Bit

→ Intervalle überlappen sich nicht mehr

**Theorem**

*nach  $d(b + n)$  Runden werden alle  $d$   $\lfloor m \rfloor$ -Komponenten erkannt*

# Systeme mit Ausfällen

- hier: synchrone Kommunikation
- Komponenten können **Ausfallen**

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : ist  $\beta$  ausgefallen?
- $\beta$ : ist  $\alpha$  ausgefallen?
- kann niemals herausgefunden werden

aber...

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : ist  $\beta$  ausgefallen?
- $\beta$ : ist  $\alpha$  ausgefallen?
- kann niemals herausgefunden werden

aber...

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : ist  $\beta$  ausgefallen?
- $\beta$ : ist  $\alpha$  ausgefallen?
- kann niemals herausgefunden werden

aber...

- $d := 2$
- $K_{[m]} := \{\alpha, \beta\}$

$$\Rightarrow d_\alpha = d_\beta = d - 1 = 1$$

- $\alpha$ : ist  $\beta$  ausgefallen?
- $\beta$ : ist  $\alpha$  ausgefallen?
- kann niemals herausgefunden werden

aber...

aber...

Zusätzliche Nachricht an alle Komponenten:

- mindestens eine funktionierende Komponente ist  $[m]$ -Komponente

→ synchrones, ausfallloses System modelliert  
(falls garantiert keine Komponente während Synchronisation ausfällt)

# aber...

Zusätzliche Nachricht an alle Komponenten:

- mindestens eine funktionierende Komponente ist  $[m]$ -Komponente

→ synchrones, ausfallloses System modelliert  
(falls garantiert keine Komponente während Synchronisation ausfällt)

# Gliederung

- 1 Einführung
- 2 Cheating Husbands
- 3 Synchronisationsprobleme
- 4 Quick Elimination**
- 5 Ergebnisse

# Komplexitätsklasse

- Bisher: Linearzeit-Algorithmen
- $d$   $[m]$ -Komponenten wurden in  $O(d)$  Runden erkannt
- geht das nicht schneller?

- Komponente  $\alpha$  kennt  $d_\alpha$   $[m]$ -Komponenten

→  $d_\alpha = d - 1$  oder  $d_\alpha = d$  !

$$d_\alpha = d - 1 \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha + 1$$

$$d_\alpha = d \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha - 1$$

- Komponente muss  $\lambda$  suchen mit  $d_\lambda = d_\alpha + 1$  oder  $d_\lambda = d_\alpha - 1$
- dann kann sich  $\alpha$  eindeutig zuordnen

- Komponente  $\alpha$  kennt  $d_\alpha$   $[m]$ -Komponenten

→  $d_\alpha = d - 1$  oder  $d_\alpha = d$  !

$$d_\alpha = d - 1 \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha + 1$$

$$d_\alpha = d \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha - 1$$

- Komponente muss  $\lambda$  suchen mit  $d_\lambda = d_\alpha + 1$  oder  $d_\lambda = d_\alpha - 1$
- dann kann sich  $\alpha$  eindeutig zuordnen

- Komponente  $\alpha$  kennt  $d_\alpha$   $[m]$ -Komponenten

→  $d_\alpha = d - 1$  oder  $d_\alpha = d$  !

$$d_\alpha = d - 1 \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha + 1$$

$$d_\alpha = d \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha - 1$$

- Komponente muss  $\lambda$  suchen mit  $d_\lambda = d_\alpha + 1$  oder  $d_\lambda = d_\alpha - 1$
- dann kann sich  $\alpha$  eindeutig zuordnen

- Komponente  $\alpha$  kennt  $d_\alpha$   $[m]$ -Komponenten

→  $d_\alpha = d - 1$  oder  $d_\alpha = d$  !

$$d_\alpha = d - 1 \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha + 1$$

$$d_\alpha = d \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha - 1$$

- Komponente muss  $\lambda$  suchen mit  $d_\lambda = d_\alpha + 1$  oder  $d_\lambda = d_\alpha - 1$
- dann kann sich  $\alpha$  eindeutig zuordnen

- Komponente  $\alpha$  kennt  $d_\alpha$   $[m]$ -Komponenten

→  $d_\alpha = d - 1$  oder  $d_\alpha = d$  !

$$d_\alpha = d - 1 \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha + 1$$

$$d_\alpha = d \Leftrightarrow \exists \lambda : d_\lambda = d_\alpha - 1$$

- Komponente muss  $\lambda$  suchen mit  $d_\lambda = d_\alpha + 1$  oder  $d_\lambda = d_\alpha - 1$
- dann kann sich  $\alpha$  eindeutig zuordnen

## Konstantzeit-Unterscheidung

	$d \bmod 3 = 0$	$d \bmod 3 = 1$	$d \bmod 3 = 2$	
$d_{[m]} \bmod 3$	2	0	1	$[m]$ -Komponenten
$d_{[m]} \bmod 3$	0	1	2	$[m]$ -Komponenten

- genau 3 Runden benötigt
- Mehrfachsetzung des geteilten Bit erlaubt

	$d \bmod 3 = 0$	$d \bmod 3 = 1$	$d \bmod 3 = 2$	
$d_{[m]} \bmod 3$	2	0	1	$[m]$ -Komponenten
$d_{[m]} \bmod 3$	0	1	2	$[m]$ -Komponenten

- genau 3 Runden benötigt
- Mehrfachsetzung des geteilten Bit erlaubt

	$d \bmod 3 = 0$	$d \bmod 3 = 1$	$d \bmod 3 = 2$	
$d_{[m]} \bmod 3$	2	0	1	$[m]$ -Komponenten
$d_{[m]} \bmod 3$	0	1	2	$[m]$ -Komponenten

- genau 3 Runden benötigt
- Mehrfachsetzung des geteilten Bit erlaubt

## Runde 1:

- $\kappa$  mit  $d_{\kappa} \bmod 3 = 0$  setzen das Bit.
- $\kappa$  mit  $d_{\kappa} = 0$  ist  $\lfloor m \rfloor$ -Komponente.

	$d \bmod 3 = 0$	$d \bmod 3 = 1$	$d \bmod 3 = 2$
$\lfloor m \rfloor$	2	0	1
$\lfloor m \rfloor$	0	1	2

## Runde 2:

- Bit in Runde 1 nicht gesetzt:
  - $\kappa$  mit  $d_{\kappa} \bmod 3 = 1$  setzen das Bit.
  - sie sind  $\lfloor m \rfloor$ -Komponenten
- Bit in Runde 1 gesetzt:
  - $\kappa$  mit  $d_{\kappa} \bmod 3 = 2$  setzen das Bit.
  - sie sind  $\lfloor m \rfloor$ -Komponenten

	$d \bmod 3 = 0$	$d \bmod 3 = 1$	$d \bmod 3 = 2$
$\lfloor m \rfloor$	2	0	1
$\lfloor m \rfloor$	0	1	2

## Runde 3:

- Bit weder in Runde 1 noch in Runde 2 gesetzt:
  - alle Komponenten sind  $\lfloor m \rfloor$ -Komponenten
- Bit in Runde 1, aber nicht in Runde 2 gesetzt:
  - Komponenten, die in **Runde 1** Bit setzten, sind  $\lfloor m \rfloor$ -Komponenten

	$d \bmod 3 = 0$	$d \bmod 3 = 1$	$d \bmod 3 = 2$
$\lfloor m \rfloor$	2	0	1
$\lceil m \rceil$	0	1	2

# Gliederung

- 1 Einführung
- 2 Cheating Husbands
- 3 Synchronisationsprobleme
- 4 Quick Elimination
- 5 Ergebnisse**

- Versagen eines klassischen Algorithmus, wenn
  - keine obere Schranke für Lieferzeit von Nachrichten
  - *b*-synchrone Lieferung von Nachrichten
  - System mit ausfallenden Komponenten
- Algorithmus kann teilweise angepasst werden
  - durch vorteilhafte Modellierung

- Versagen eines klassischen Algorithmus, wenn
  - keine obere Schranke für Lieferzeit von Nachrichten
  - $b$ -synchrone Lieferung von Nachrichten
  - System mit ausfallenden Komponenten
- Algorithmus kann teilweise angepasst werden
  - durch vorteilhafte Modellierung

- Versagen eines klassischen Algorithmus, wenn
  - keine obere Schranke für Lieferzeit von Nachrichten
  - $b$ -synchrone Lieferung von Nachrichten
  - System mit ausfallenden Komponenten
- Algorithmus kann teilweise angepasst werden
  - durch **vorteilhafte Modellierung**

# Modellierung

- Modellierung des Systems durch
  - **Wissen** einzelner Komponenten
  - **Aktionen** einzelner Komponenten
  - **Kommunikation** zwischen Komponenten

# Gliederung

- 1 Einführung
- 2 Cheating Husbands
- 3 Synchronisationsprobleme
- 4 Quick Elimination
- 5 Ergebnisse

Fragen?

# Literatur

- *Moses, Dolev, Halpern*: Cheating husbands and other stories: a case study of knowledge, action, and communication.
- *Halpern, Moses*: Knowledge and common knowledge in a distributed environment
- *El Gamal, Orlitsky*: Interactive data compression
- *Tel*: Introduction to Distributed Algorithms