

# Leader Election

CPU-netzwerk:

mehrere gleiche CPUs mit direkter Kommunikation

mappt direkt zu undirected graphs

diff ger/unger. Graphen:  $(u, v) \leftrightarrow \{u, v\}$

↗  
darin wollen wir LE lösen:

LE = es soll ein Knoten als Leader gewählt werden,  
alle anderen <sup>Motivation</sup> Followers  $\Rightarrow$  Leader kann koordinieren

jeder Knoten eignet sich dabei als Leader (alle sind gleich)

$\Rightarrow$  Symmetry Breaking: das ist die Schwierigkeit hier, sich  
konsistent für eine Lösung zu entscheiden

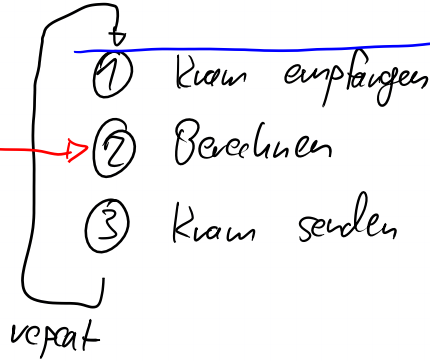
Kommunikationsmodell dabei:

jeder Knoten hat  $k$  Nachbarn,  $k$  nummerierte "Ports", kann  
so an jeden Nachbarn senden + empfangen.  
Dabei gibt es 2 Optionen:

# ① Synchron

"rundenbasiert"

hier  
ist der  
Runde 1  
Entrypoint



Vereinfachungen:  
eine globale Uhr startet  
dies, Berechnungen  
werden immer rechtzeitig  
fertig

# ② Asynchron

"eventbasiert"

- Initialisierungsevent
- Nachrichten-erhalten-Event

Vereinfachungen: für Korrektheit  
Nachrichtübermittlung  
endlich schnell  
für Laufzeit  
 $NÜ \leq \tau_{\text{timestep}}$

# Komplexitätsmaße:

# Runden (synchron)

# ges. Nachrichten (async)

interne Berechnungen werden vernachlässigt! meist eh nur einfache Per.

---

weitere Varianten:

- anonyme Knoten / ID'ed Knoten

- uniform / bekannte Größe

→ Bitstrings in  $O(\log n)$  Länge, als Zahl interpretierbar

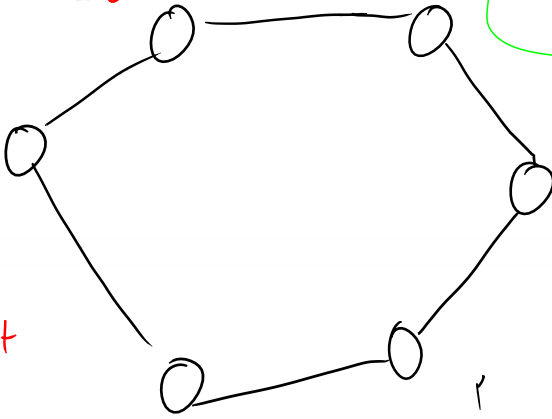
→ nach Leader Elec ist IDs verteilen leicht

(Quantität  $n$  ist globales Wissen)

/ Approximation mit konstanter Abweichung reicht oft

Beispiel Leader Election im Ring <sup>ohne IDs</sup>  $\rightarrow$  Ports sind clockwise oder counter

dies ist unmöglich mit det. Algos



geht nicht

was ist ein det. Algo?

eine Funktion  $f$ , die auf jeden Knoten  $v$  angew. werden kann.

$f$  bekommt alles Wissen, was  $v$  hat:

- History der Nachb.
- "
- ...

aber nicht die ID - existiert nicht

und gibt alle zu sendenden Nachrichten in der kommenden Runde + Entscheidung, ob  $v$  Leader wird

$f$  löst LE, wenn es nach endlich vielen Runden einen Leader gibt.

Beweis induktiv

(\*) :  $\forall f$  sind in jeder Runde alle empf. Nachr für  $\forall$  Knoten gleich

(\*)  $\Rightarrow$   $\forall$  Knoten haben gleiche Historie

<sup>keine IDs</sup>  $\Rightarrow$  Input zu  $f$  ist immer gleich

$f$  ist Fkt  $\Rightarrow$  Output von  $f$  ist immer gleich

$\Rightarrow f$  ist BS

also zZ: (\*)

(13) in der ersten Runde sind die  $cu/cav$  empf. NR gleich  
 $\Rightarrow$  es wurde  $mix$  gesendet

(14) (\*) gilt bis jetzt, z.Z. für kommende Runde

(15) per 14 haben alle Knoten gleiche Historie, gleich viele Parts etc.  
 $\Rightarrow f$  gibt gleichen Output für alle Knoten  
 $\Rightarrow$  alle Knoten senden dasselbe  
 $\Rightarrow$  alle Knoten empfangen dasselbe

Wir betrachten gleiches Problem mit IDs

Idee: kleinste ID wird Leader (diese ist nicht bekannt!) → sonst trivial

## Clockwise Algorithmus

Sei  $ID(v)$  ID des Knotens  $v$

Jeder Knoten bekommt eine lokale Variable  $T_v$

Runde 1:  $T_v := ID(v)$

sende  $T_v$  an clockwise Nachbarn

Runde  $\geq 2$ : erhalte Nachricht  $N$ :

ist  $N < T_v$ :

$T_v = N$

$v$  wird Follower

$v$  sendet  $N$  weiter

ist  $N = ID(v)$ :

$v$  wird Leader


# Analyse

$O(n)$  Runden:  $n+1$  Runden

Proof

Sei  $z$  Knoten mit niedrigster ID

Sei  $v_i$  der  $i$ -te Knoten nach  $z$  im Uhrzeigersinn

d.h.  $v_0 = z$ ,  $v_1 =$    $= v_2$

Induktionshypothese: In Runde  $i+1$  empfängt  $v_i$   $ID(z)$   
wird Follower und schickt weiter

$\Rightarrow$  Nach  $n+1$  Runden erhält  $z$  die eigene ID  
 $z$  wird Leader

$\Rightarrow$  Nach  $i+1$  Runden wird Knoten  $v_i$  Follower



Es werden höchstens  $O(n^2)$  Nachrichten versendet:

- Nachrichten werden nur gesendet, wenn  $T_v$  geändert wird
- $T_v$  wird nur geändert, wenn die empf. ID kleiner ist
- Es gibt  $n$  IDs

$\Rightarrow T_v$  wird max.  $n$ -mal geändert

- Wir haben  $n$  Knoten

$\Rightarrow O(n^2)$

# CW- Algo (async)

Async hat 2 events: init oder Nachricht...

sync ist  $\approx$  Spezialfall von Async: Knoten werden gleichzeitig init'd bei async  
einfach wie immer

Algo ist ähnlich:

init  $v$ :  $T_v := ID(v)$

sende  $T_v$  clockwise

receive  $M$  at  $v$ :

if  $M < T_v$  or  $T_v$  unset ( $v$  uninitialized):

$T_v = M$

send  $T_v$  clockwise  
become follower

if  $M = ID(v)$

become leader

Analyse

Async CW Algo:  $\leq 2n-1$  ticks to solve LE

da: Zeitmessung beginnt mit einem init'alem Knoten  
der zuerst  $z$  nach max.  $n-1$  Ticks  
 $z$  erhält dann eigene ID nach max  $n$  Ticks  
 $\Rightarrow 2n-1$

Nachrichtenzusatzkomplexität ist  $O(n^2)$  (Proof wie gerade)

Gibt es Improvement bei der Nachrichtenkomplexität?

(Laufzeit ist optimal: weniger als  $n$  geht nicht, Leader muss ja allen sagen dass er Leader ist...)

CW Algo ist manchmal  $\Theta(n^2)$ , das suckt.

Gedanke: Keine Nachricht senden ist auch eine Nachricht (in sync)

Neuer Algo:

receive M at  $v$ :

become follower

send M clockwise

Runden beginnen bei 1

reach round  $ID(v) \cdot n + 1$  at  $v$  and  $v$  is not a follower:

send leader message M clockwise

become leader

# Analyse

- alle  $n+1$  Runden wird jemand Leader
  - dies propagiert in  $n$  Runden
  - in Runde  $n \cdot \min_v 10C_v$  wird jemand Leader
  - $O(n)$  Nachrichten
  - $O(n \cdot \min_v 10C_v + 1)$  Runden
- evtl. ineffizient.

# Conclusion

- LE ist leicht zu formulieren, aber v.V. unmöglich!
- Clockwise Algo: A/sync,  $O(n)/O(n^2)$
- $O(n \cdot \min_v |D(v)|)/O(n)$  Algo

next:  $O(n)/O(n \log n)$  Algo!