Model distribuovaného výpočtu

Distribuovaný systém

prechod od paralelného k distribuovanému systému

silne spriahnuté systémy (tightly coupled)

- SIMD – Single Instruction, Multiple Data
- STMD – Single Thread, Multiple Data
- SPMD – Single Program, Multiple Data

slabo spriahnuté systémy (loosely coupled)

Distribuovaný výpočet

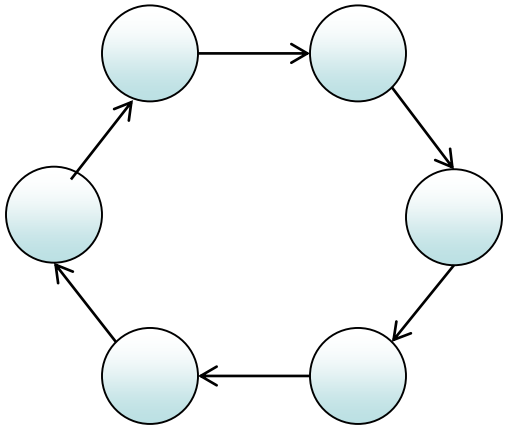
každý procesor má vlastnú pamäť a komunikuje s ostatnými len pomocou správ (message passing)

v SPMD modeli má každý procesor kópiu spoločného (sekvenčného) kódu.

od ostatných sa líši

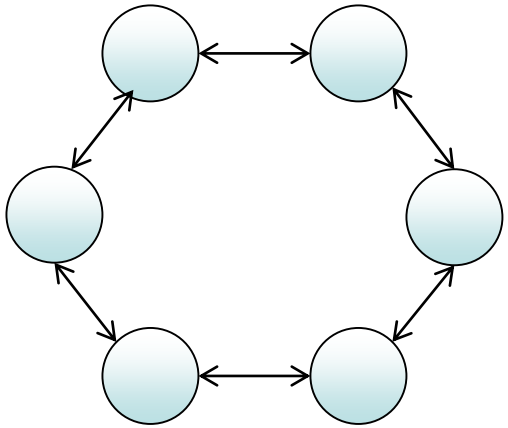
- jednoznačným identifikátorom (**myid**) ... rank
- informáciou o komunikačných (orientovaných) kanáloch, cez ktoré možno prenášať správy (podľa topológie)

Topológia distribuovanej siete



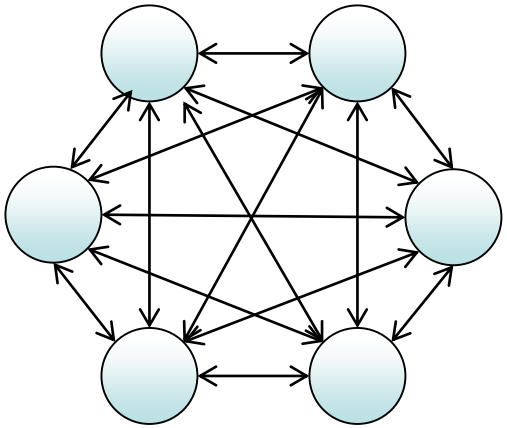
usmernený kruh

(directed ring)



obojsmerný kruh

(bidirected ring)



úplné prepojenie

(full connection)

hyperkocka, strom (acyklická sieť),

všeobecná sieť s jedno resp. obojsmernými prepojeniami ...

Používané distribuované systémy a algoritmy

- smerovanie IP paketov v sieti Internet
- prepínanie rámcov v lokálnej sieti
- odstránenie zacyklenia v lokálnej sieti protokolom STP
- distribuovaná správa doménových mien DNS
- distribuovaná správa zdieľaných súborov v peer-to-peer sieťach
- ...

Problémy distribuovaného systému

- proces nemá informáciu o (aktuálnom) globálnom stave systému (-> uviaznutie)
- neexistuje globálny čas – udalosti nie sú globálne usporiadané podľa času ich vzniku (-> vzájomné vylúčenie)
- vykonávanie procesov je nedeterministické (rozdiely v rýchlosti vykonávania)
- príjem správ nemusí byť v tom poradí, v akom boli odoslané
- výpočtové a komunikačné chyby

Asynchrónny a synchrónny model

Asynchrónny model – výpočty prebiehajú oddelene, v ľubovoľnom poradí a rôznymi rýchlosťami, posielanie a prijímanie správ sú nezávislé udalosti a môžu stať kedykoľvek – poradie nemusí byť stále rovnaké.
zložité programovanie <-> prenositeľnosť

Synchrónny model – vykonávanie simultánnych krokov synchrónne – v dohodnutých časových intervaloch.
jednoduchšie programovanie <-> nároky na prostredie

-> simulácie synchrónneho prostredia v asynchrónnom systéme (čiastočne synchronizované modely)

Synchrónny distribuovaný výpočet

Krok synchrónneho výpočtu obsahuje:

- prijatie všetkých správ zo vstupných komunikačných kanálov (ak sa tam nejaké nachádzajú)
- sekvenčný výpočet (na základe prijatých správ), príprava ďalších správ a ich pripojenie ku príslušným komunikačným kanálom
- odoslanie správ do zvolených kanálov (ak je to potrebné)
- serializované bezpečné prenosy správ (FIFO)

Časová zložitosť – počet týchto krokov, potrebných na dosiahnutie koncového stavu

Komunikačná zložitosť – počet odoslaných správ (presnejšie - počet bitov v správach)

Odosielanie správ

- **send** (data, dest, tag) ... podľa MPI

dest – identifikácia komunikačného kanála (rozhrania), ktorým chceme správu poslať

tag – značka správy (typ dát)

Ssend – synchrónne (blokované) odoslanie – výpočet sa pozastaví až do chvíle, keď správu prijme cieľový proces (čakať môže odosielateľ alebo príjemca)

Bsend – asynchrónne (neblokované) odoslanie do zásobníka správ, z ktorého si môže správu vybrať cieľový proces (**probe** – kontrola zásobníka)

Prijímanie správ

- **receive** (data, src, tag, status) ... podľa MPI

src - identifikácia komunikačného kanála, z ktorého chceme prijať správu

tag – akú značku má mať prijatá správa

ANY_SOURCE – možno použiť na prijatie správy z ľubovoľného vstupu

ANY_TAG – ľubovoľná značka

status – ukazovateľ na prijaté hodnoty src a tag (ak boli volané parametrami ANY)

príklad problému s uviaznutím:

P1:

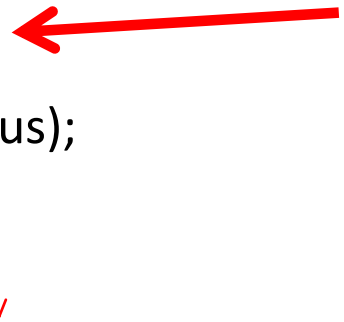
Ssend(x, 2, tag);

receive(z, 2, tag, status);

P2:

Ssend(y, 1 tag);

receive(w, 1, tag, status);



blokovaný send,
čaká na potvrdenie príjmu

dá sa riešiť blokovanými verziami odoslania s upraveným poradím

v synchrónnom modeli predpokladáme neblokované odosielania a blokované príjmy správ (vykonávanie sa pozastaví do času, keď príde správa)

Compare – Exchange

procedure CompareExchange (i,j) {pomocou blokovaných odosielaní s potvrdením}

{ vstup: x = a na P_i , x = b na P_j výstup: x = min{a,b} na P_i , x = max{a,b} na P_j }

if myid = i **then**

    Ssend(x, j, tag)

    receive(temp, j, tag, status)

    if x > temp **then** x := temp **endif**

endif;

if myid = j **then**

    receive(temp, i, tag, status)

    Ssend(x, i, tag)

    if temp > x **then** x:= temp **endif**

endif

end CompareExchange

Pre komunikáciu s viacerými uzlami:

broadcast (data, src)

jeho volanie musí byť synchronizované vo všetkých procesoch – slúži pre odosielanie i pre príjem

scatter (data, datapart, src)

podobne ako broadcast, ale každý proces dostane len svoj kúsok dát

gather (datapart, data, dst)

opačne – údaje od všetkých procesorov dostane cieľový proces

Distribuovaný algoritmus

Korektnosť distribuovaného algoritmu:

- dôkaz správnosti výsledku
- dôkaz ukončenia všetkých procesov
- dôkaz vyprázdnenia všetkých komunikačných kanálov

Voľba koordinátora (leadera) v kruhovej sieti (ring)

- každý procesor je spojený v kruhu s ľavým a pravým susedom (vie ich rozoznať ?)
- výsledkom je dosiahnutie špeciálneho stavu *leader* pre jediný procesor (proces)

známa orientácia ? obojsmerný prenos ?

známy počet procesorov ?

Ak sú všetky procesy identické, problém výberu koordinátora nie je možné riešiť Dôkaz!

Leader election

Každý proces má jednoznačný identifikátor **myid**

Vyhráva ten, ktorého identifikátor je najvyšší (riešenie porušením symetrie – symmetry-breaking)

LCR algoritmus (LeLann, Chang, Roberts) voľby koordinátora v jednosmernom kruhu

- každý proces pošle svoj identifikátor do kruhu
- z opačnej strany prijíma id, ktoré porovnáva so svojím ak je vyššie – prepošle po kruhu ďalej
- koordinátorom sa stane ten, kto dostane svoje id po obehnutí celého kruhu

Ring leader election (LCR)

```
procedure RingLeaderElection
{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}
status := not leader; send(myid, out, 0); receive(i, in, tag);
while tag = 0 do
    if i > myid then send(i, out, 0)
                else if i = myid then send(i, out, 1); status := leader
                     endif
    endif;
    receive(i, in, tag)
endwhile;
if status <> leader then send(i, out, 1)
endif;
end RingLeaderElection
```

Ring leader election (LCR)

procedure RingLeaderElection

{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}

status := **not** leader; **send**(**myid**, out, 0); **receive**(i, in, tag);

while tag = 0 **do**

    if i > **myid then send**(i, out, 0)

            else if i = **myid then send**(i, out, 1); status := leader **endif**

    endif

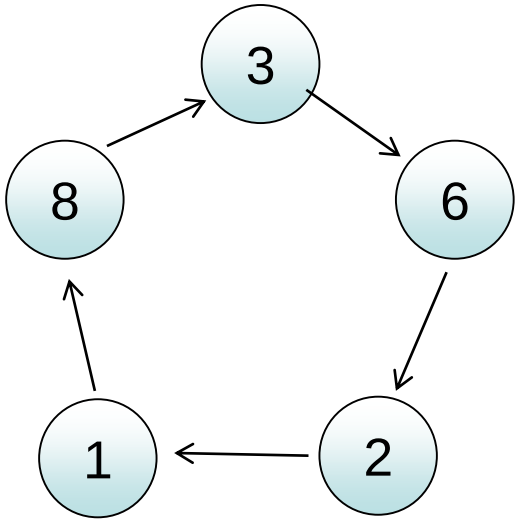
    receive(i, in, tag)

endwhile

if status <> leader **then send**(i, out, 1)

endif

end RingLeaderElection



Ring leader election (LCR)

procedure RingLeaderElection

{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}

status := **not** leader; **send**(**myid**, out, 0); **receive**(i, in, tag);

while tag = 0 **do**

    if i > **myid then send**(i, out, 0)

            else if i = **myid then send**(i, out, 1); status := leader **endif**

    endif

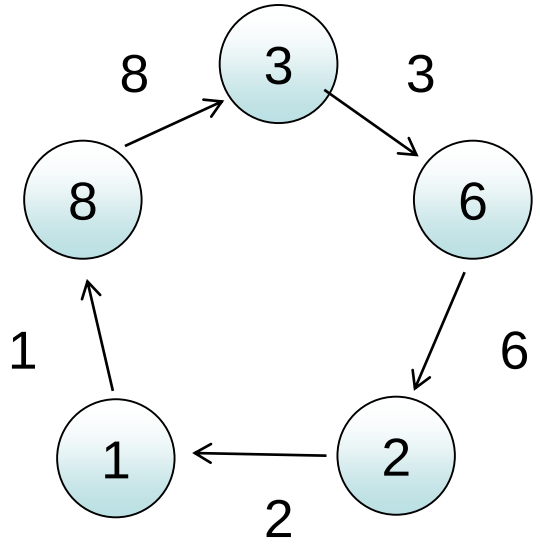
    receive(i, in, tag)

endwhile

if status <> leader **then send**(i, out, 1)

endif

end RingLeaderElection



Ring leader election (LCR)

procedure RingLeaderElection

{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}

status := **not** leader; **send**(**myid**, out, 0); **receive**(i, in, tag);

while tag = 0 **do**

    if i > **myid then send**(i, out, 0)

            else if i = **myid then send**(i, out, 1); status := leader **endif**

    endif

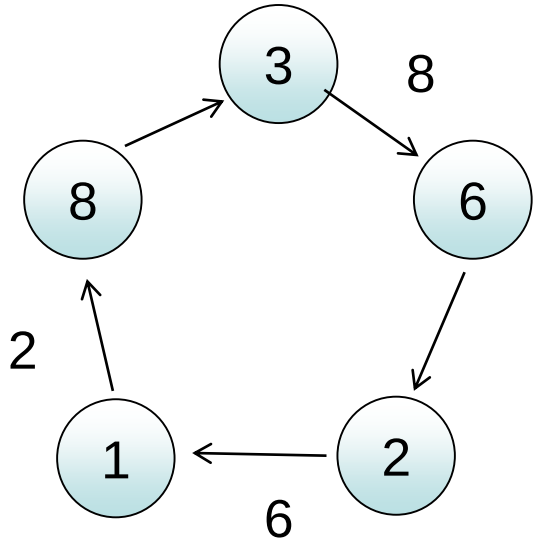
    receive(i, in, tag)

endwhile

if status <> leader **then send**(i, out, 1)

endif

end RingLeaderElection



Ring leader election (LCR)

procedure RingLeaderElection

{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}

status := **not** leader; **send**(**myid**, out, 0); **receive**(i, in, tag);

while tag = 0 **do**

    if i > **myid then send**(i, out, 0)

            else if i = **myid then send**(i, out, 1); status := leader **endif**

    endif

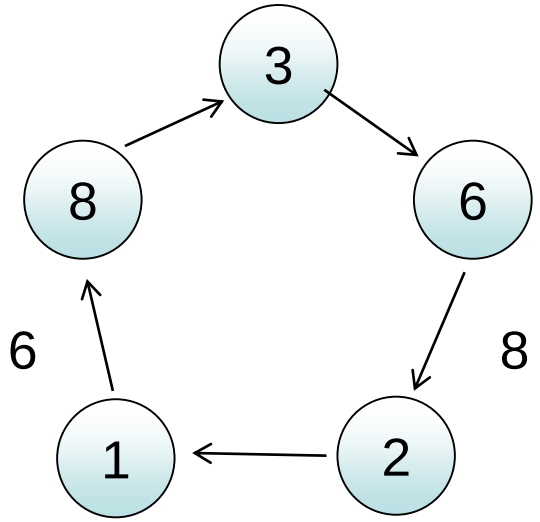
    receive(i, in, tag)

endwhile

if status <> leader **then send**(i, out, 1)

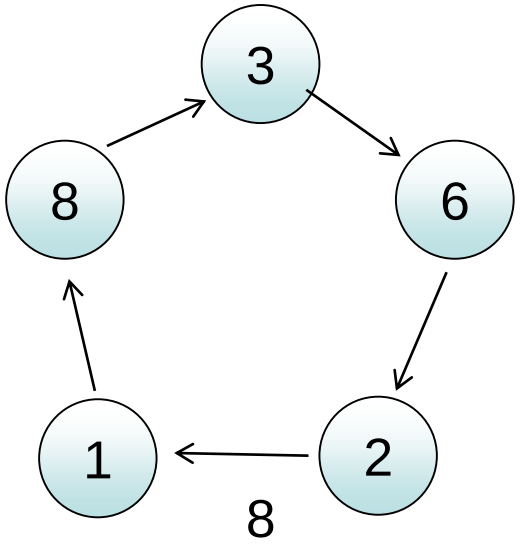
endif

end RingLeaderElection



Ring leader election (LCR)

```
procedure RingLeaderElection
{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}
status := not leader; send(myid, out, 0); receive(i, in, tag);
while tag = 0 do
    if i > myid then send(i, out, 0)
                else if i = myid then send(i, out, 1); status := leader endif
    endif
    receive(i, in, tag)
endwhile
if status <> leader then send(i, out, 1)
endif
end RingLeaderElection
```



Ring leader election (LCR)

procedure RingLeaderElection

{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}

status := **not** leader; **send**(**myid**, out, 0); **receive**(i, in, tag);

while tag = 0 **do**

    if i > **myid then send**(i, out, 0)

            else if i = **myid then send**(i, out, 1); status := leader **endif**

    endif

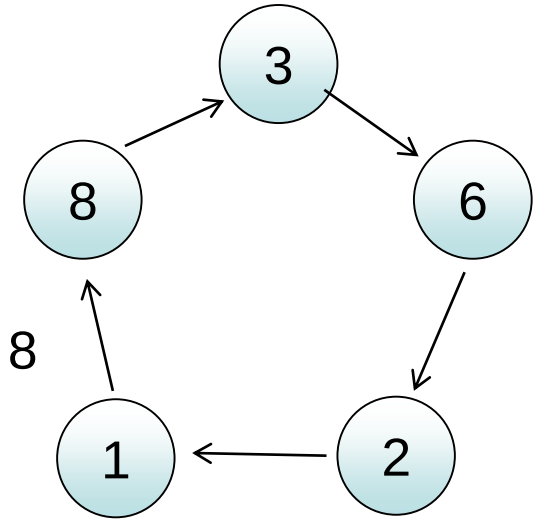
    receive(i, in, tag)

endwhile

if status <> leader **then send**(i, out, 1)

endif

end RingLeaderElection



Ring leader election (LCR)

procedure RingLeaderElection

{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}

status := **not** leader; **send**(**myid**, out, 0); **receive**(i, in, tag);

while tag = 0 **do**

    if i > **myid then send**(i, out, 0)

            else if i = **myid then send**(i, out, 1); status := leader **endif**

    endif

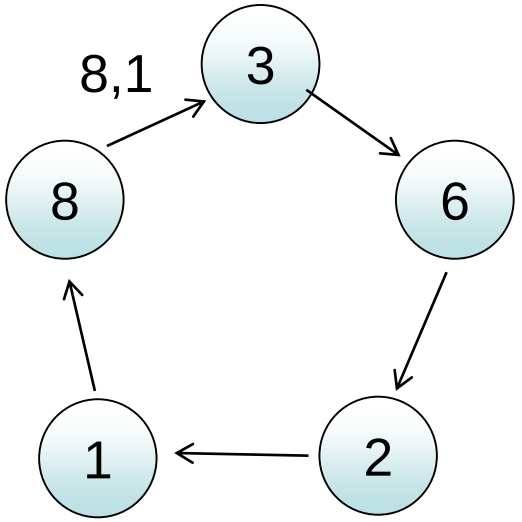
    receive(i, in, tag)

endwhile

if status <> leader **then send**(i, out, 1)

endif

end RingLeaderElection



Ring leader election (LCR)

- proces s max myid nastaví v n-tom kroku stav leader
- žiaden iný proces neprejde do tohto stavu
- algoritmus sa zastaví po 2n krokoch (kvôli ukončeniu všetkých procesov)
- nie je potrebné vedieť počet procesov
- časová zložitosť – $O(n)$
- komunikačná zložitosť – $O(n^2)$

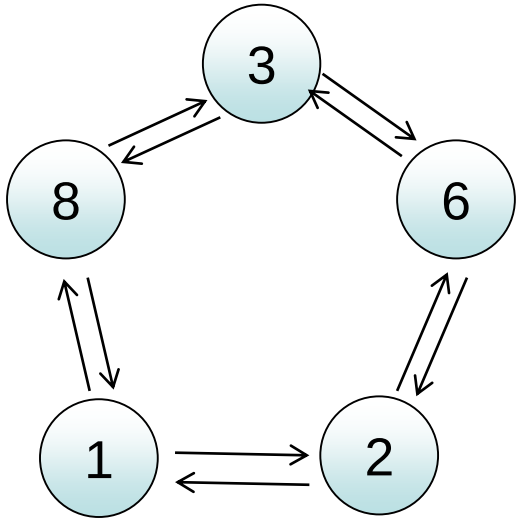
  ... priemerne $O(n \cdot \log n)$

Riešenie porušením symetrie (symmetry-breaking)

Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- najskôr pošle proces svoje id len susedom, ak sa z niektorej strany nevráti (sused má vyššie id), koordinátorom sa nestane
- v ďalšej fáze posielajú procesy, ktorým sa vrátili obidve výzvy, svoje id do vzdialenosti dva – ak sa vráti odpoveď z oboch strán, proces pokračuje
- v k-tej fáze posiela proces do oboch strán svoje id do vzdialenosti 2^k do oboch strán
- proces s vyšším id správu nepreposiela
- do ďalšej fázy pokračujú len procesy, ktorým sa vráti výzva z oboch strán
- ak sa vráti procesu id z opačnej strany, stáva sa koordinátorom

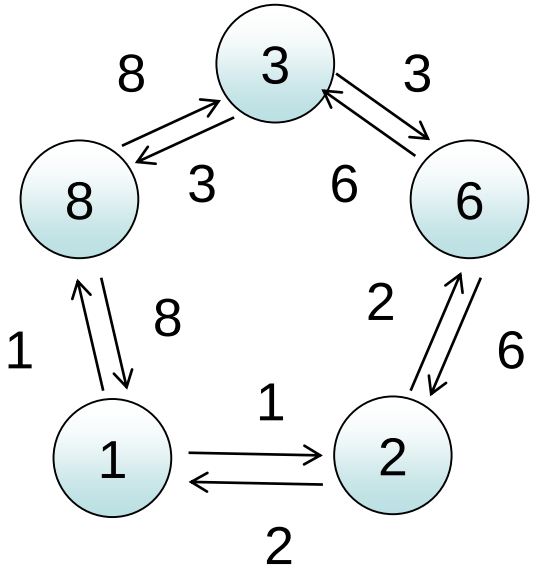


Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- v k-tej fáze posiela proces svoje id len do vzdialenosti 2^k do oboch strán, správu zruší proces s vyšším číslom. Ak sa vráti id z opačnej strany – je leader

- 0-tá fáza – posielajú všetci do oboch strán

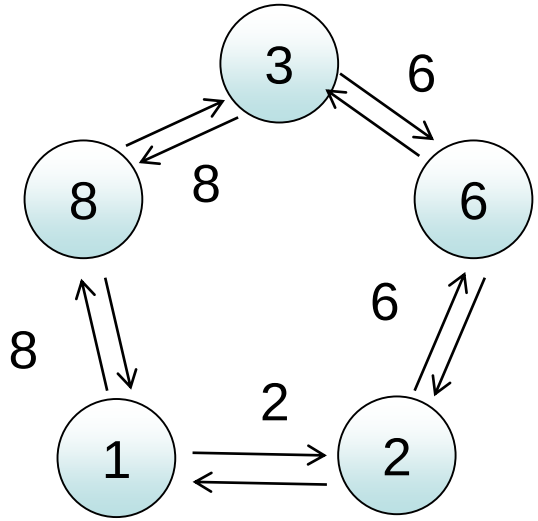


Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- v k-tej fáze posiela proces svoje id len do vzdialenosti 2^k do oboch strán, správu zruší proces s vyšším číslom. Ak sa vráti id z opačnej strany – je leader

- 5 odpovedí - do ďalšej fázy postupuje 8 a 6 (dostali späť svoje id z obidvoch strán)

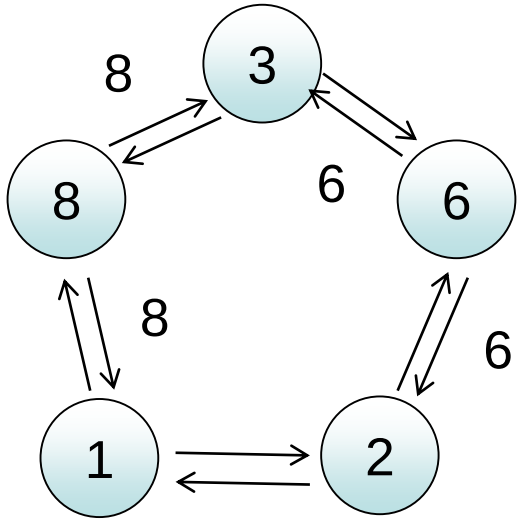


Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- v k-tej fáze posiela proces svoje id len do vzdialenosti 2^k do oboch strán, správu zruší proces s vyšším číslom. Ak sa vráti id z opačnej strany – je leader

- ďalšiu fázu začína 6 a 8

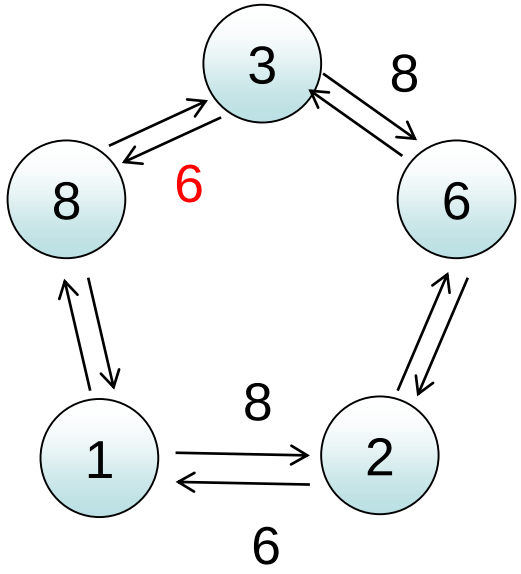


Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- v k-tej fáze posiela proces svoje id len do vzdialenosti 2^k do oboch strán, správu zruší proces s vyšším číslom. Ak sa vráti id z opačnej strany – je leader

- vo vzdialenosti 2 proces 8 neprepošle naspäť výzvu od procesu 6

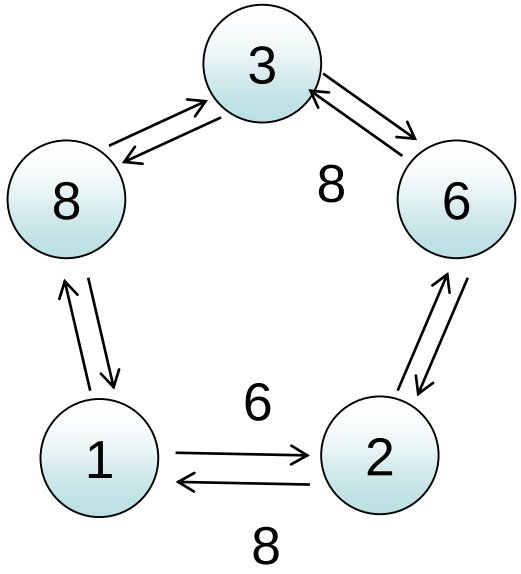


Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- v k-tej fáze posiela proces svoje id len do vzdialenosti 2^k do oboch strán, správu zruší proces s vyšším číslom. Ak sa vráti id z opačnej strany – je leader

- zo vzdialenosti 2 sa vrátia obidve odpovede pre proces 8 (6 len z jednej strany)

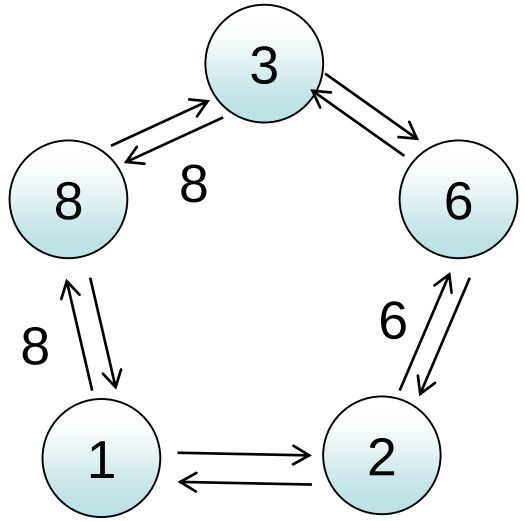


Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- v k-tej fáze posiela proces svoje id len do vzdialenosti 2^k do oboch strán, správu zruší proces s vyšším číslom. Ak sa vráti id z opačnej strany – je leader

- fázu vyhráva len 8
- v ďalších fázach bude len 8 a dostane svoje správy z opačných strán



Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- v k-tej fáze posiela proces svoje id len do vzdialenosti 2^k do oboch strán, správu zruší proces s vyšším číslom
- ak sa mu vráti id z opačnej strany – je víťazom

v k-tej fáze začína najviac $\left\lfloor \frac{n}{2^{k-1}+1} \right\rfloor$ procesov

v k-tej fáze každý proces spôsobí najviac 2^{k+2} správ
- spolu v k-tej fáze bude najviac 4n správ
- vykoná sa najviac 2^{k+1} krokov

fáz je najviac log n, teda celkovo O(n . log n) správ a najviac $2^{\log n+3} = 8n$ krokov –> čas výpočtu bude v O(n)

Bidirectional ring leader election (HS)

procedure HSRingLeaderElection

{vstup myid > 0, in, out; výstup: status – leader alebo not leader}

status := **not** leader; decide:=false; phase:=0; echo:=0; **send**(**myid**,in,1); **send**(**myid**,out,1);

while not decide **do receive**(i, dir, tag); {dir – direction in = 0; out = 1}

if i > **myid then**
if tag > 1 **then send**(i, 1-dir, tag-1) {1-dir je opačné rozhranie k dir}
else if tag = 1 **then send**(i, dir, 0) **else send**(i, 1-dir, 0) **endif endif**

else if i = **myid then if** echo = 0 **then** echo:=1 {očakáva potvrdenie z oboch strán}
else if tag > 0 **then** status := leader; **send**(0, out, **myid**) {terminácia}
else phase := phase+1; **send**(**myid**, in, 2^{phase}); **send**(**myid**, out, 2^{phase});
echo := 0 **endif endif**

else if i = 0 **then** decide := true; **if** status <> leader **then send**(0, out, tag) **endif endif**

endif endif

endwhile

end HSRingLeaderElection {posielanie správ send je neblokované ... Bsend}

Bidirectional ring leader election (HS)

```
HSRingLeaderElection:   #  vstup myid > 0, in, out; výstup: status – leader alebo not leader

status = not leader; decide = false; range = 1; echo = 0
send(myid,in,range); send(myid,out,range)

while not decide :
    receive(i, dir, tag)      #  dir .. direction  in = 0    out = 1

    if  i > myid :
            if  tag > 1 :  send(i, 1-dir, tag-1)    # 1-dir je opačné rozhranie k dir
            if  tag == 1 : send(i, dir, 0)
            if  tag == 0 : send(i, 1-dir, 0)
    if i == myid :
            if echo == 0 : echo = 1          #  očakáva potvrdenie z oboch strán
            if (echo == 1) and (tag > 0) :  status = leader; send(0, out, myid)    # terminácia
            if (echo == 1) and (tag == 0) :         # prechod do novej fázy
                    echo = 0; range *= 2; send(myid, in, range); send(myid, out, range)
    if i == 0 :
            decide = true
            if status <> leader : send(0, out, tag)
```

Ďakujem za pozornosť !