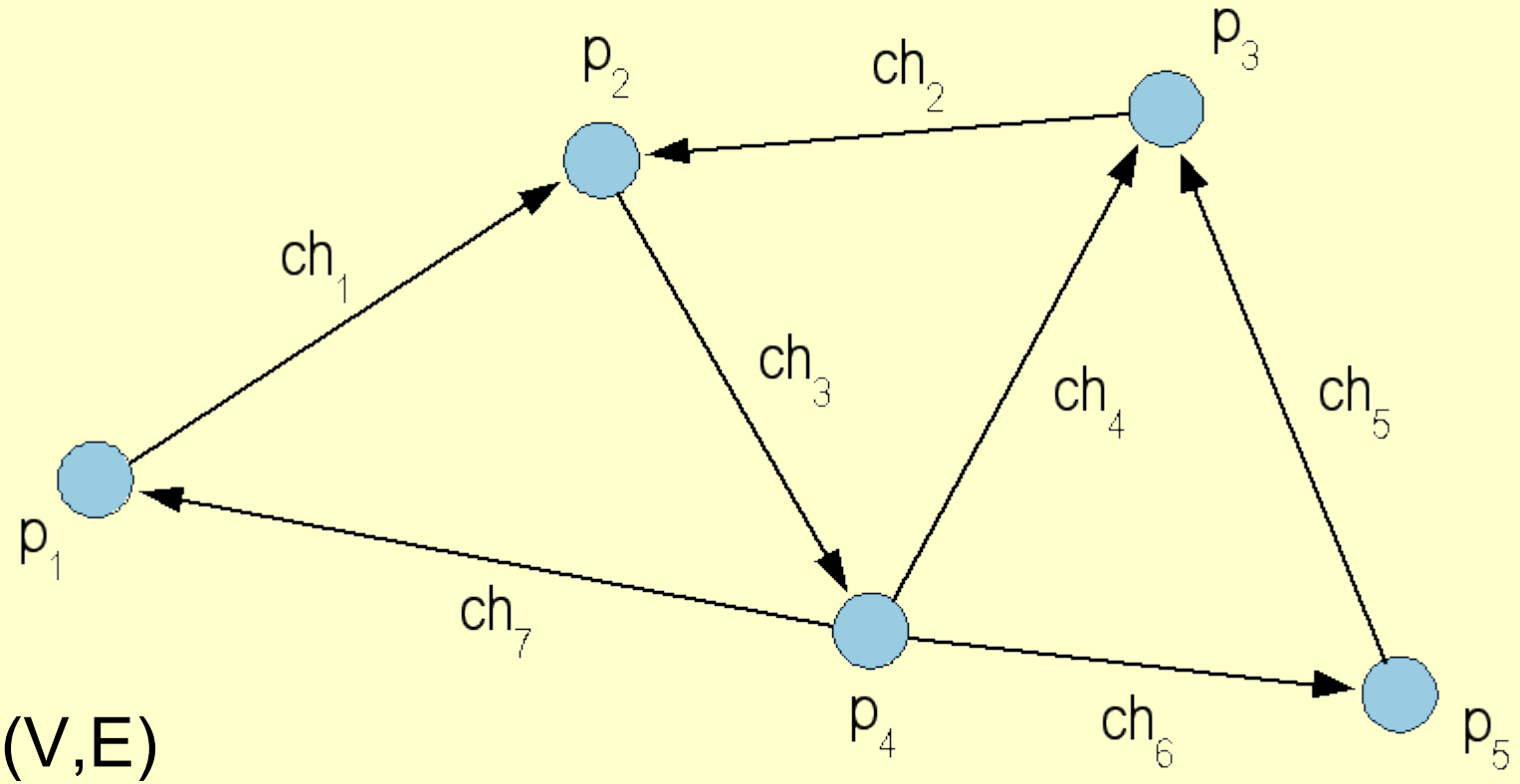
Distribuované systémy a výpočty

X36DSV

Jan Janeček
(zastupuje Peter Macejko)

Model distribuovaného výpočtu

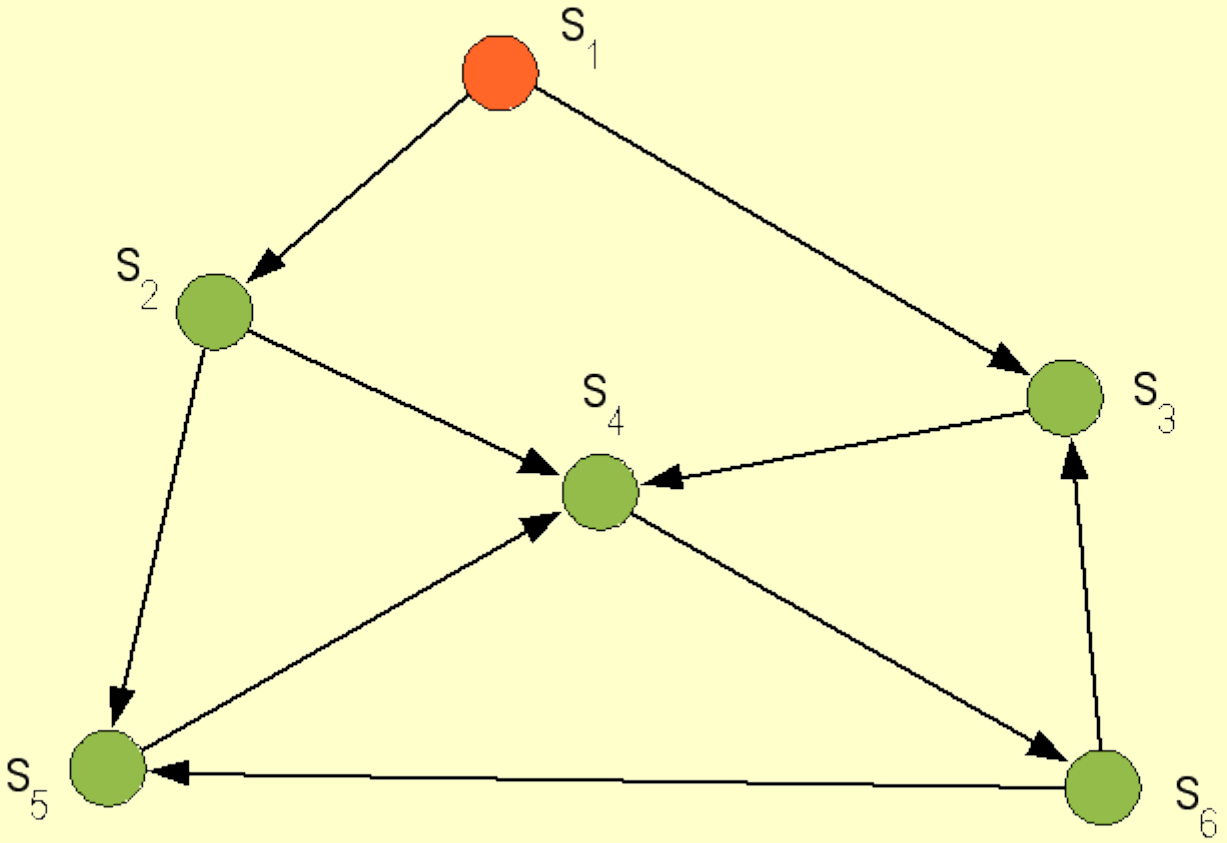


G = (V,E)

V = { p_1, p_2, . . . } - processes

E = { ch_1, ch_2, . . . } - channels

Model procesu



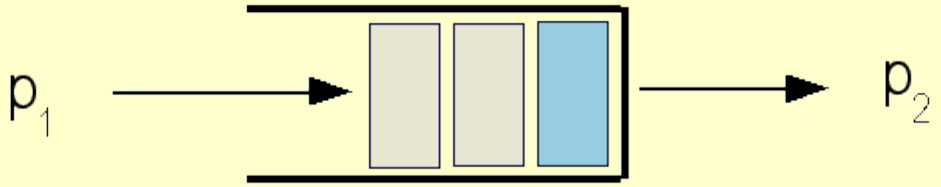
FSM = (S,T,S_1)

$S = \{ s_1, s_2, \ldots \}$ - states

$T = \{ t_1, t_2, \ldots \}$ - transitions

Model komunikačního kanálu

- dvoubodové spoje
- FIFO strategie



paralelní výpočet

- synchronní kanál = blokující vysílač
  konečná / jednotková kapacita

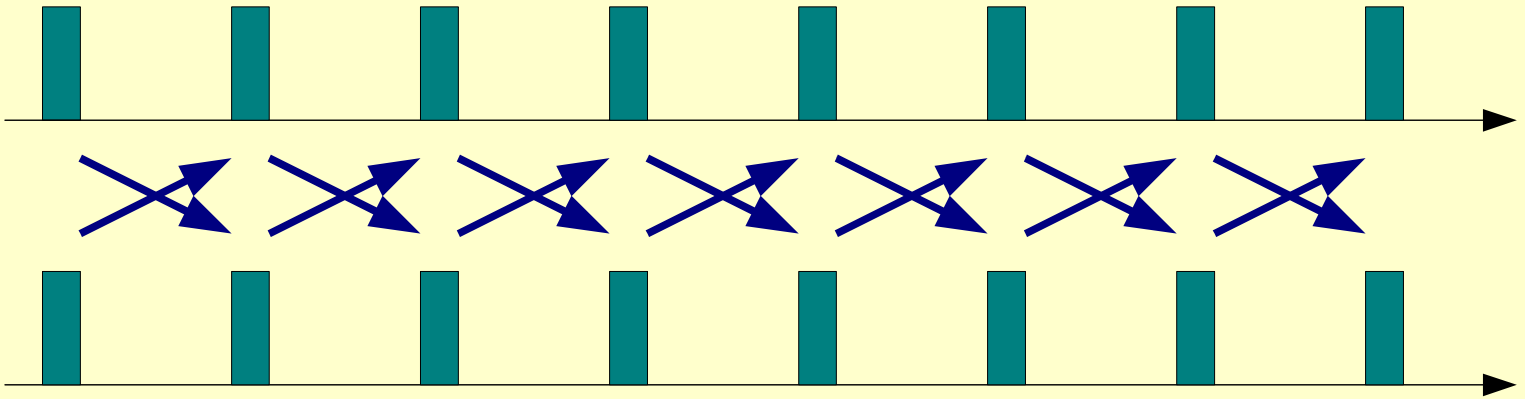
distribuovaný výpočet

- asynchronní kanál = neblokující vysílač
  neomezená kapacita

Model výpočtu

paralelní výpočet

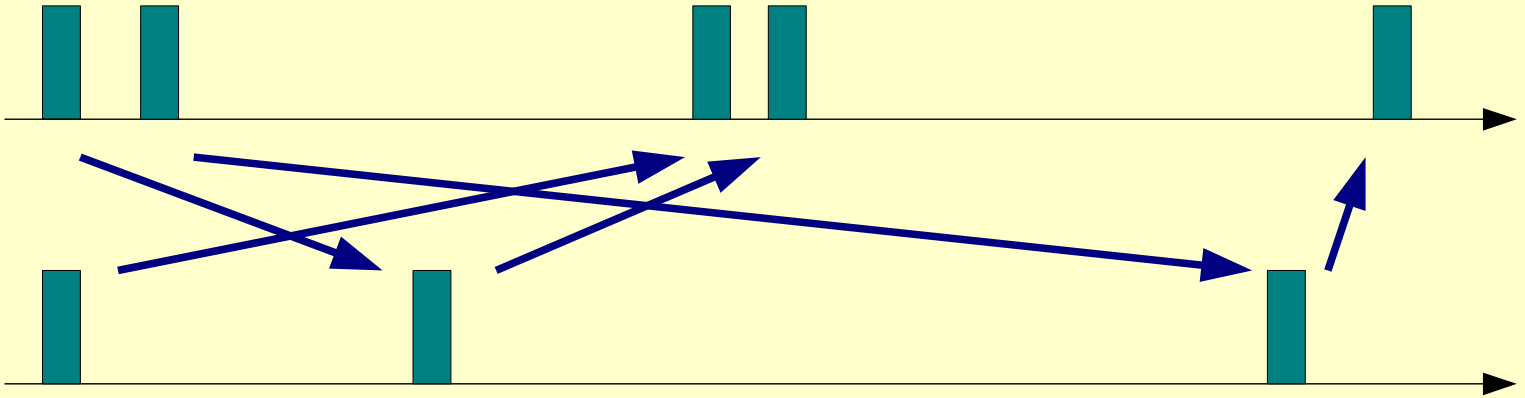
- synchronní kanál = blokující vysílač
- pravidelné střídání fází výpočtu a komunikace u všech procesů
- dovoluje to limitovaná doba komunikace (a výpočtu)
- méně rozsáhlý stavový prostor
- výpočet zůstává deterministický



Model výpočtu

distribuovaný výpočet

- asynchronní kanál = neblokující vysílač
- neomezená doba komunikace (a výpočtu)
- nezávislý postup jednotlivých procesů
- zavádí do popisu nedeterministické chování
- a výrazně rozšiřuje stavový prostor (fronty zpráv)



Symetrie výpočtu

silně symetrický výpočet

- shodný kód procesů (peer-to-peer)
- shodný prováděný výpočet

slabě symetrický výpočet

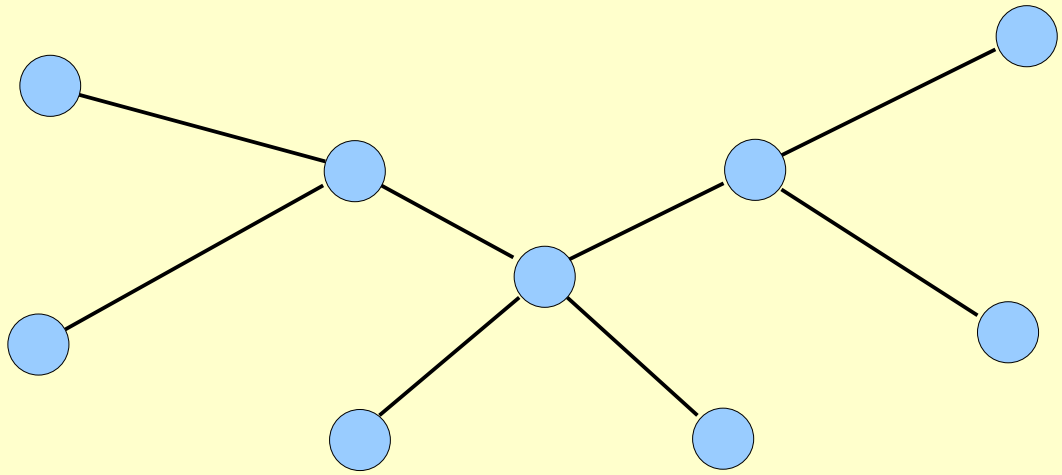
- shodný kód procesů (peer-to-peer)

asymetrický výpočet

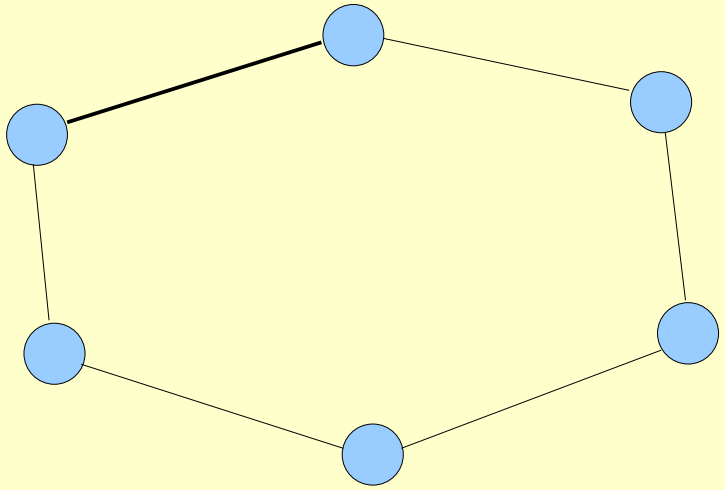
- odlišný kód procesů (client-server)

Komunikační grafy

Tree



Ring



- unidirectional
- bidirectional

Problémy

Broadcasting and synchronization

- distribuce informace všem, čekání na podmínku, …

Election

- výběr uzlu pro speciální úlohu

Termination detection

- rozpoznání konce výpočtu na všech uzlech

Resource allocation

- nalezení zdroje v síti

Problémy

Mutual exclusion

- problémy přístupu ke sdíleným zdrojům

Deadlock detection

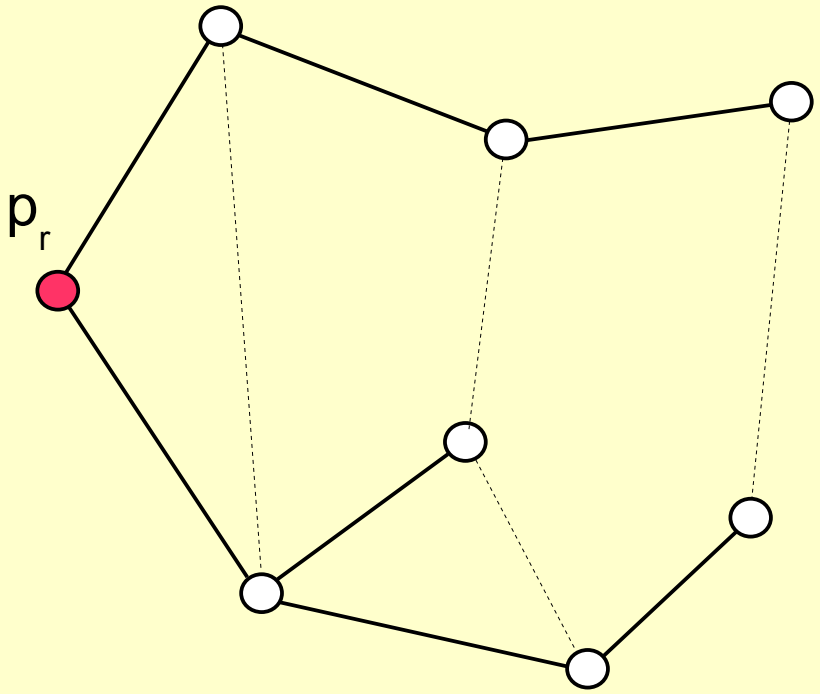
- detekce a případné řešení vzniklých problémů

Distributed file maintenance

- dodržování pořadí žádostí, konzistentní stav systému

Broadcast

synchronní varianta



p_r

no message received:

send M to all children
terminate

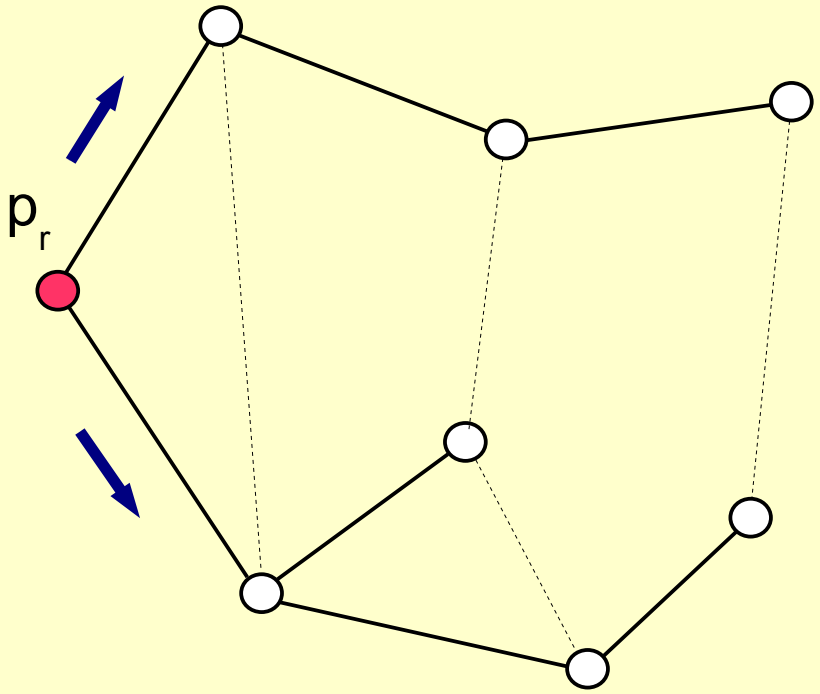
p_i, $0 \le i \le n-1$, $i \ne r$

message M received from the parent:

send M to all children
terminate

Broadcast

synchronní varianta



p_r

no message received:

send M to all children
terminate

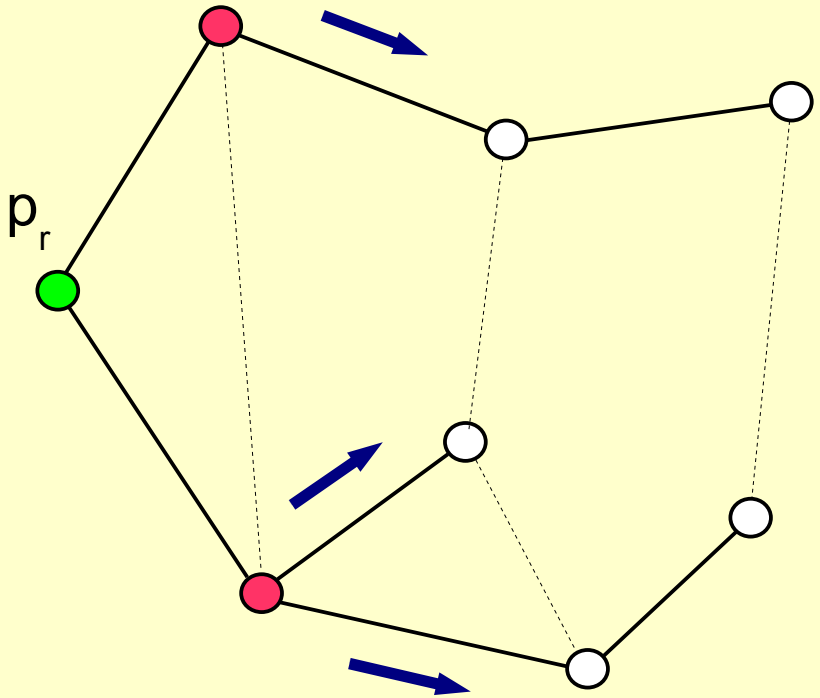
p_i, $0 \le i \le n-1$, $i \ne r$

message M received from the parent:

send M to all children
terminate

Broadcast

synchronní varianta



p_r

no message received:

send M to all children

terminate

p_i, $0 \le i \le n-1$, $i \ne r$

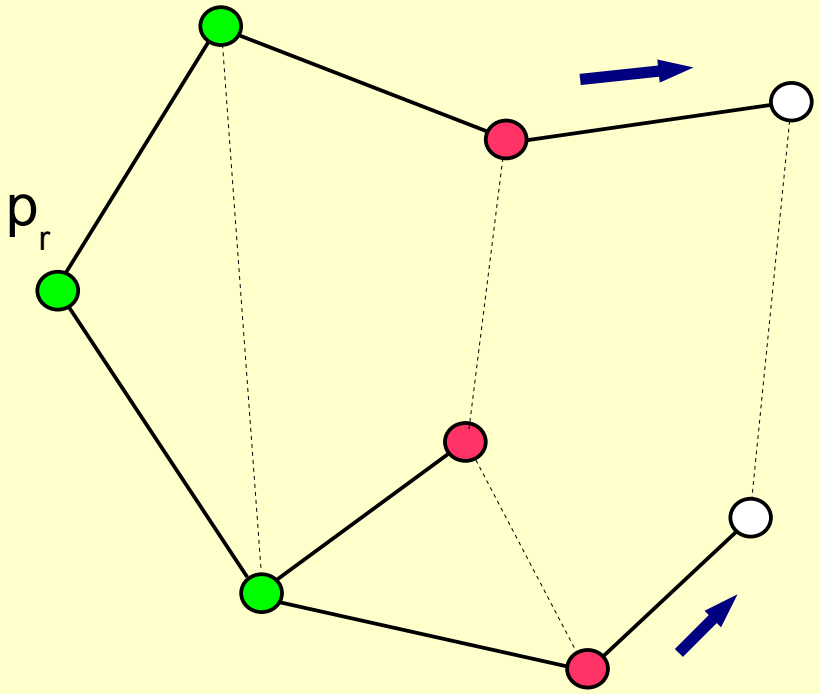
message M received from the parent:

send M to all children

terminate

Broadcast

synchronní varianta



p_r

no message received:

send M to all children
terminate

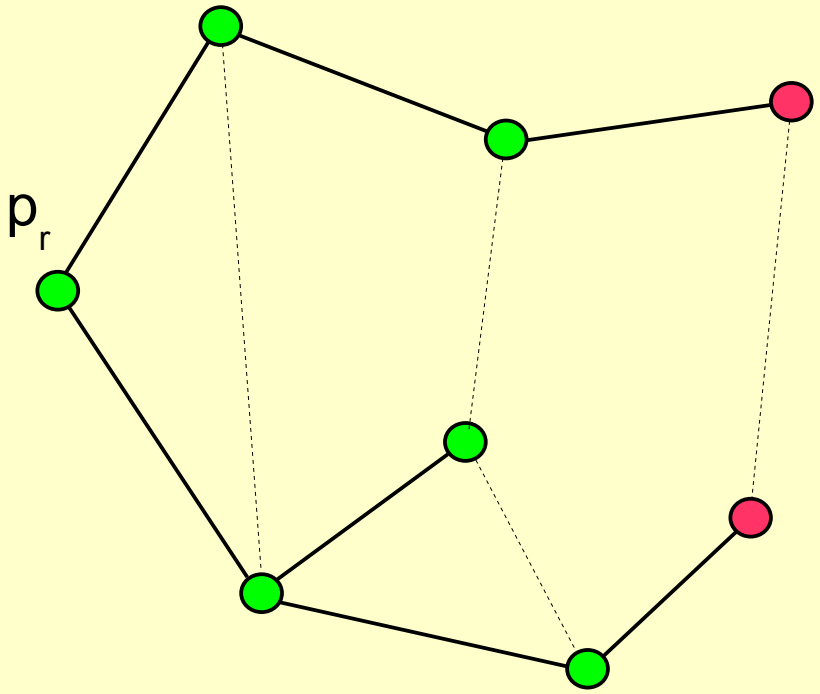
p_i, $0 \le i \le n-1$, $i \ne r$

message M received from the parent:

send M to all children
terminate

Broadcast

synchronní varianta



p_r

no message received:

send M to all children
terminate

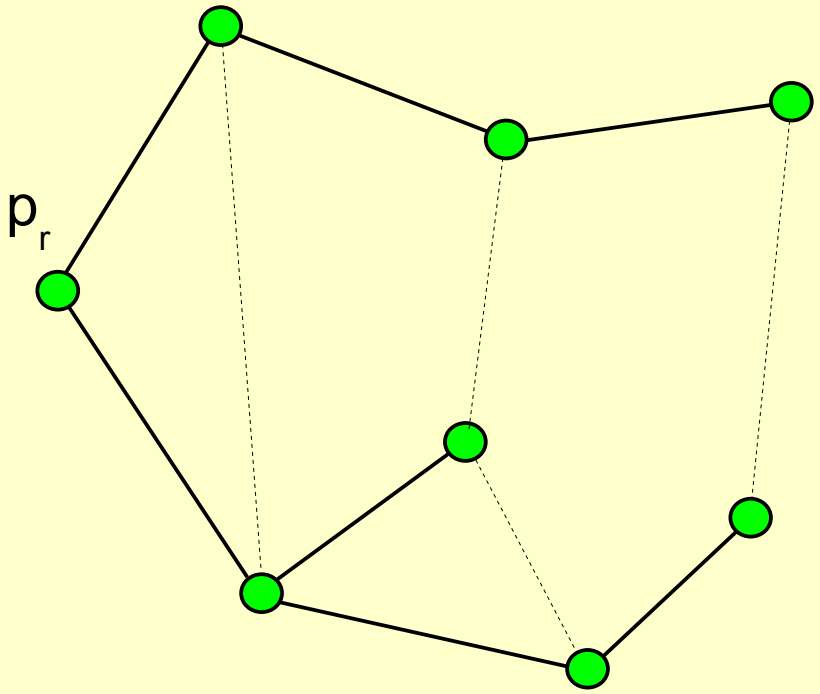
p_i, $0 \le i \le n-1$, $i \ne r$

message M received from the parent:

send M to all children
terminate

Broadcast

synchronní varianta



p_r

no message received:

send M to all children
terminate

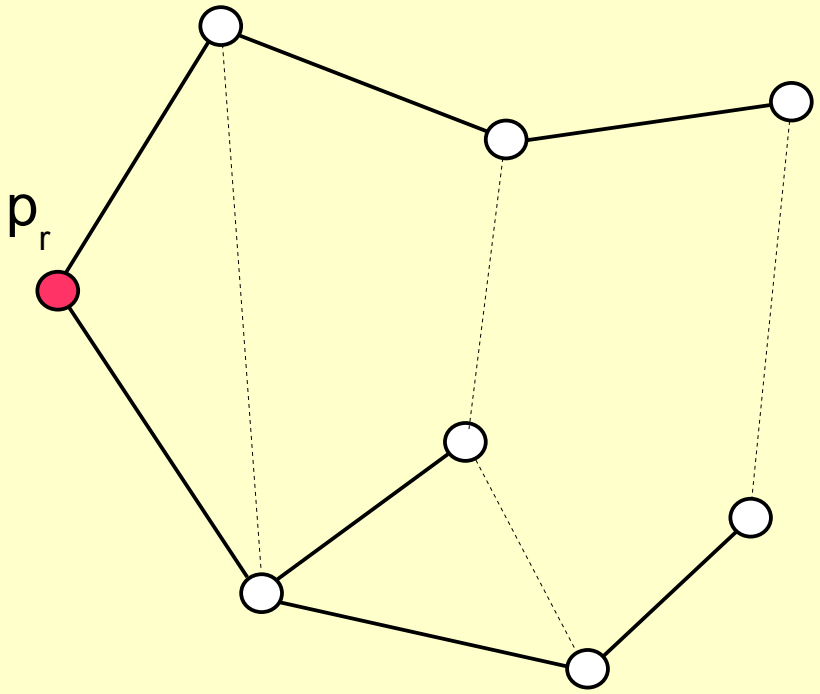
p_i, $0 \le i \le n-1$, $i \ne r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

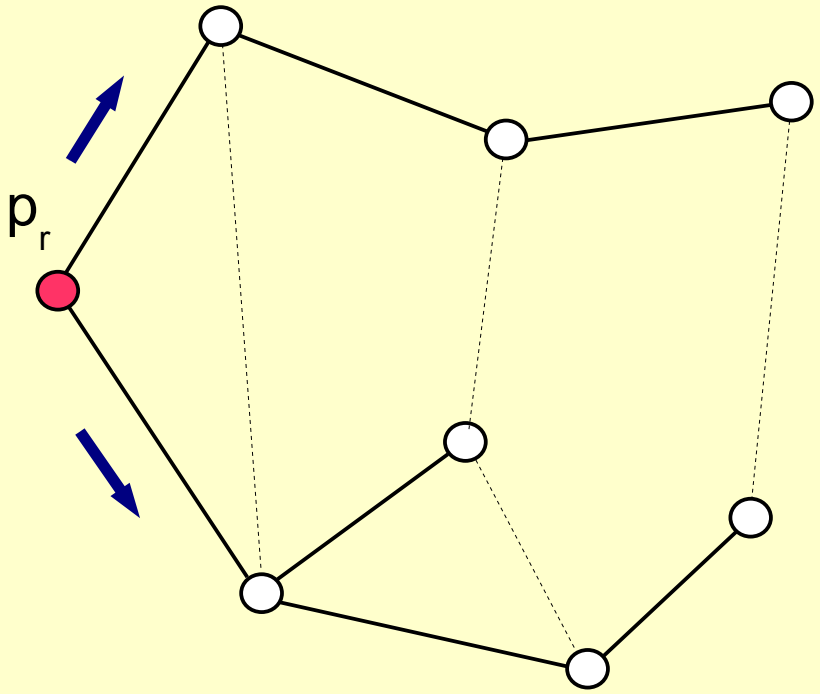
p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

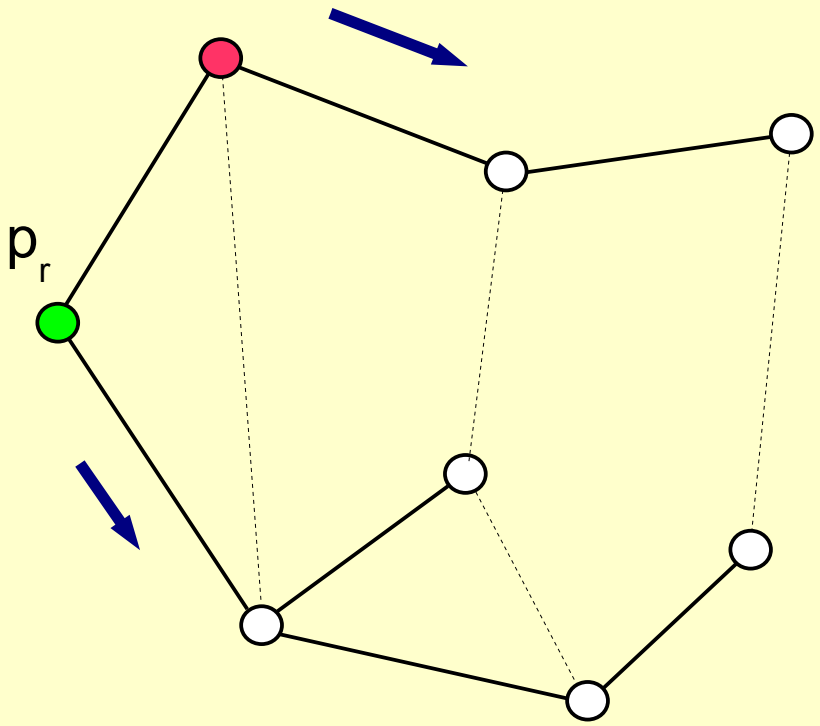
p_i, $0 \le i \le n-1$, $i \ne r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

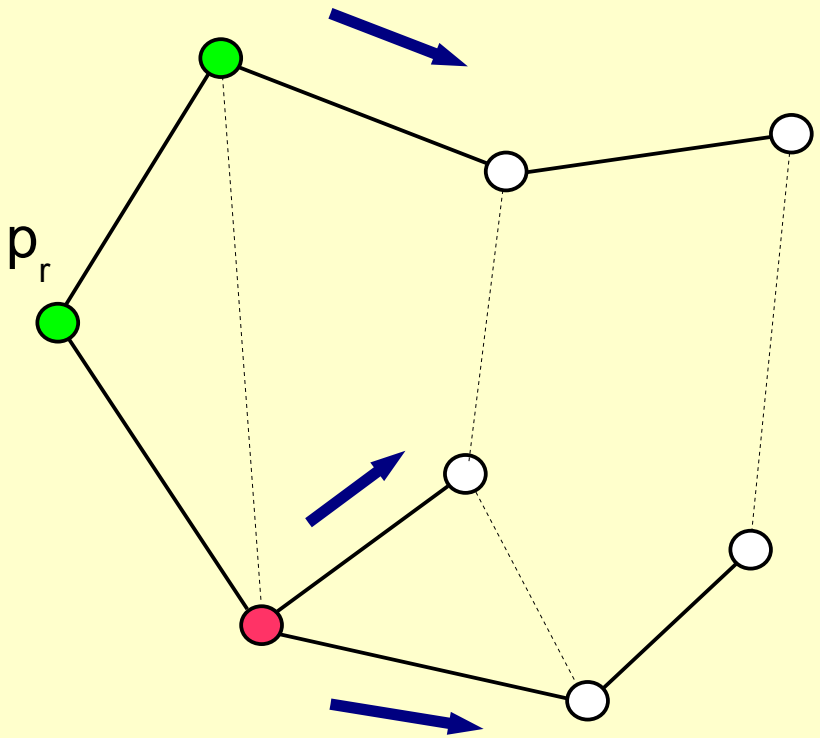
p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

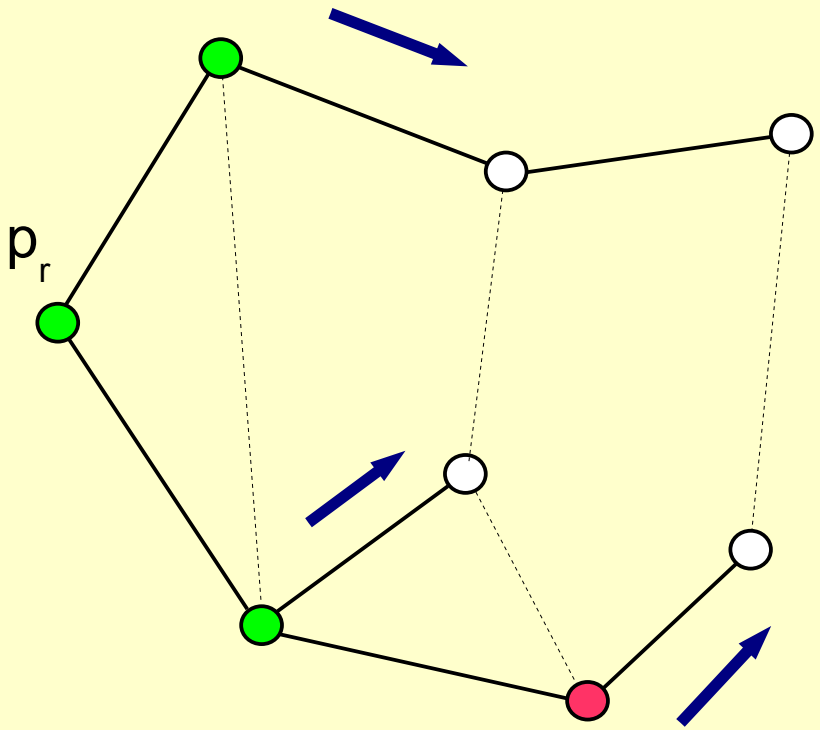
p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

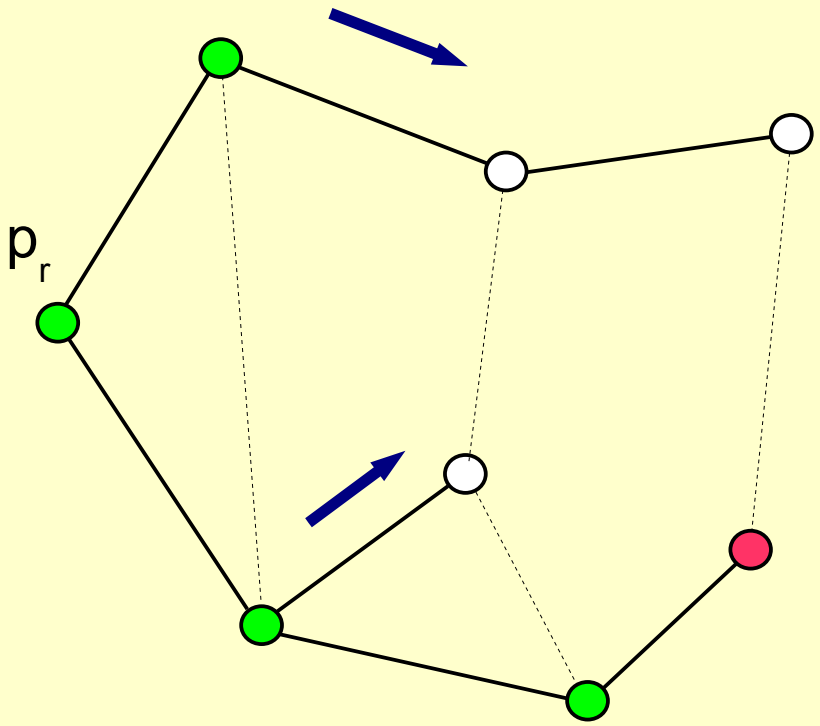
p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

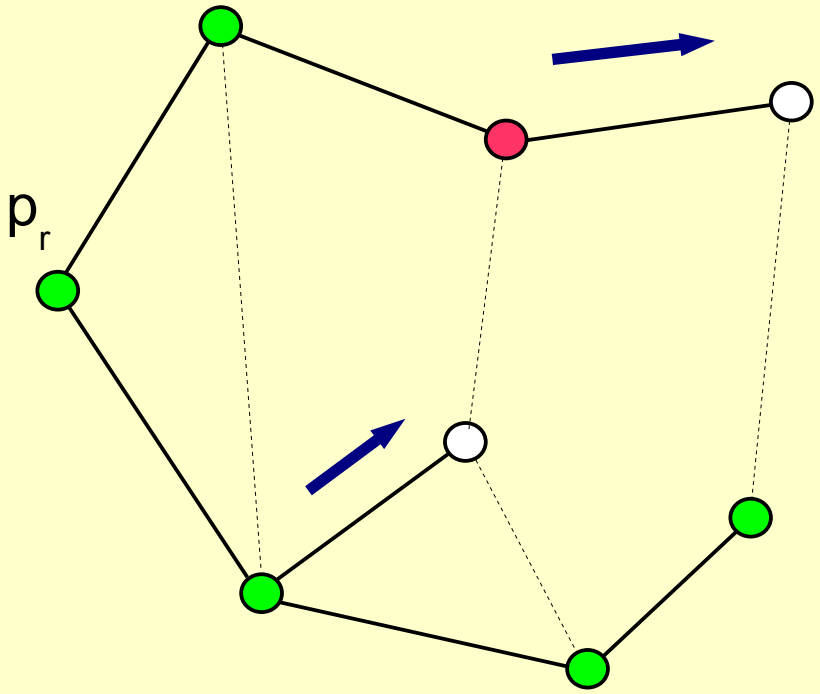
p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

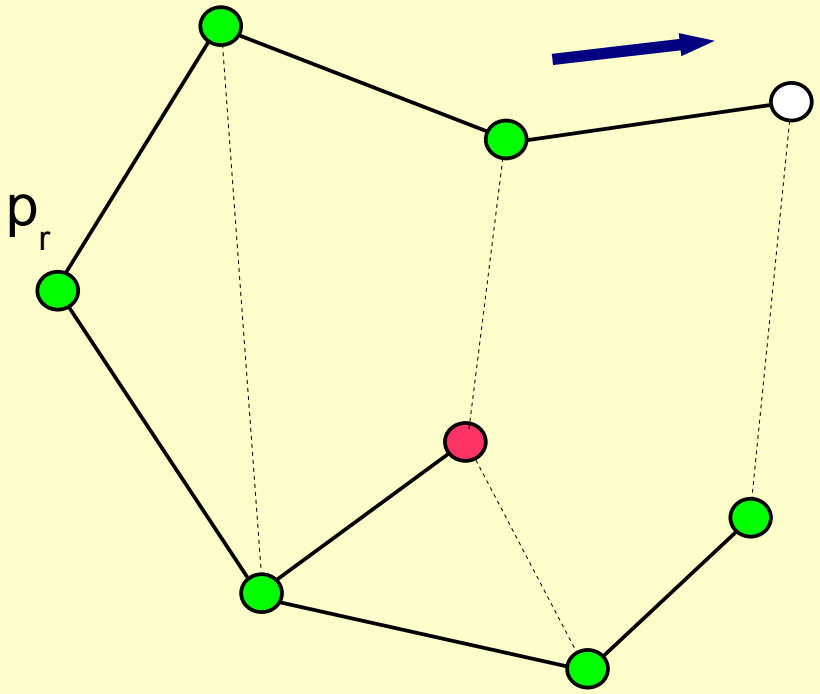
p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

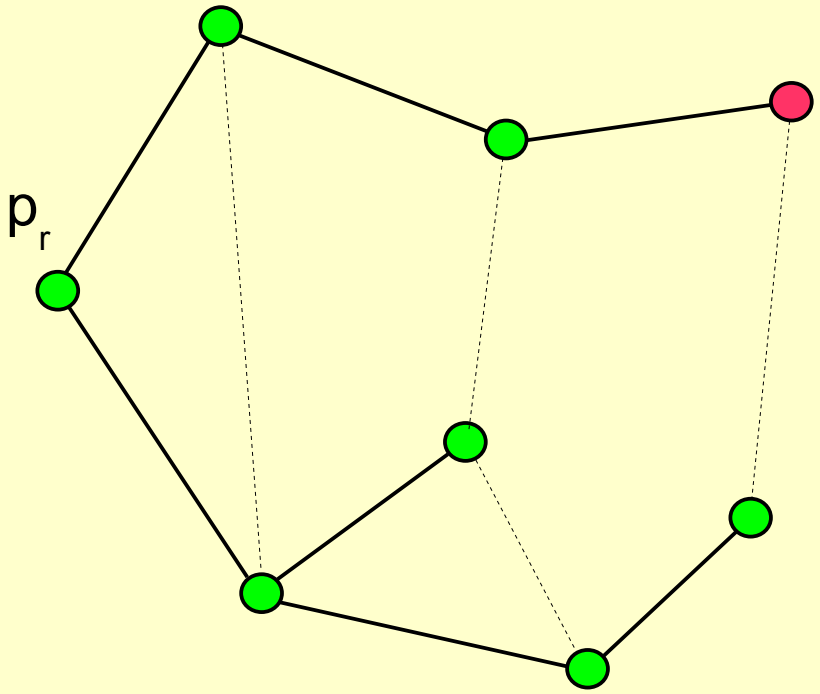
p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

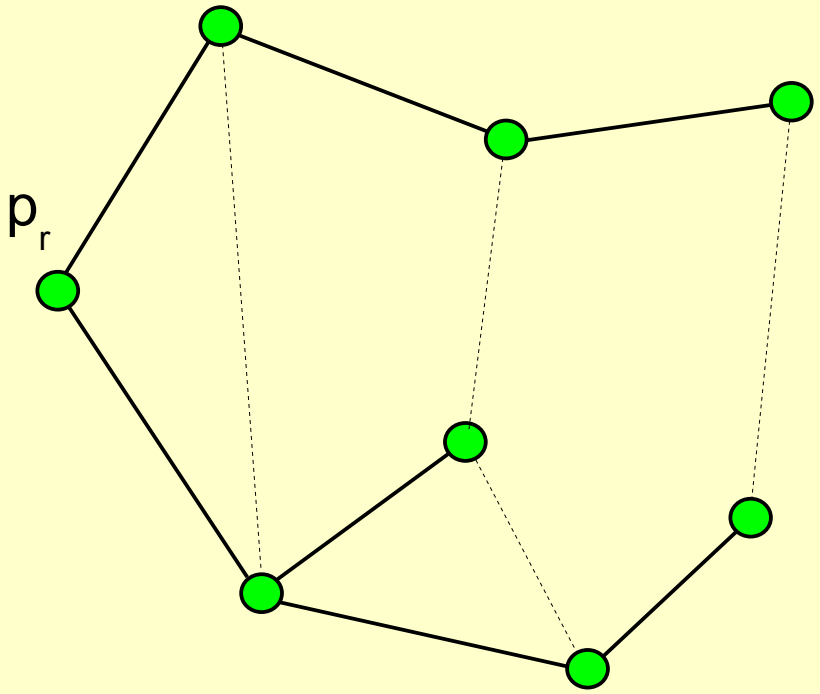
p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

asynchronní varianta



p_r

no message received:

send M to all children
terminate

p_i, $0 \leq i \leq n-1$, $i \neq r$

message M received from the parent:

send M to all children
terminate

Broadcast

Safety condition :

- tvrzení (assertion) platné po celou dobu výpočtu

  vytvářený graf je acyklický

Live condition :

- tvrzení dosažitelné výpočtem
  (například zaručí, že nedojde k uvázutí)

  výsledný podgraf G'<G je kostrou G

Broadcast

Komunikační složitost :

- počet zpráv potřebných k výpočtu

  n-1 v našem broadcastu

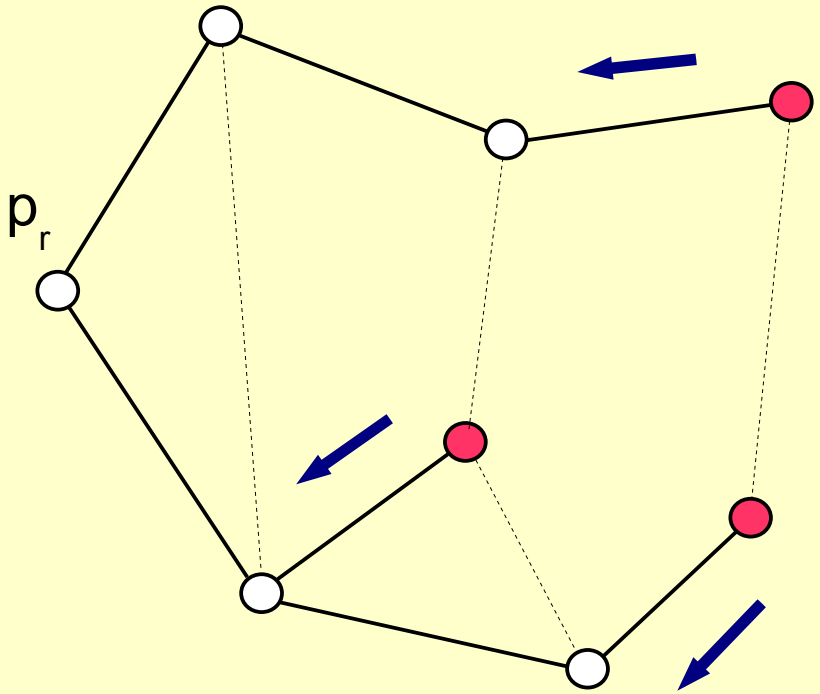
Časová složitost :

- počet kroků potřebných k ukončení výpočtu

  d (hloubka grafu G') v našem broadcastu

Convergecast

synchronní varianta



p_i - having no child

no message received:

send x_i to the parent

terminate

p_i - having child(ren)

message x_j received from the child:

if x_j received from all children

send max(x_j) to the parent

terminate

p_r

message x_j received from the child:

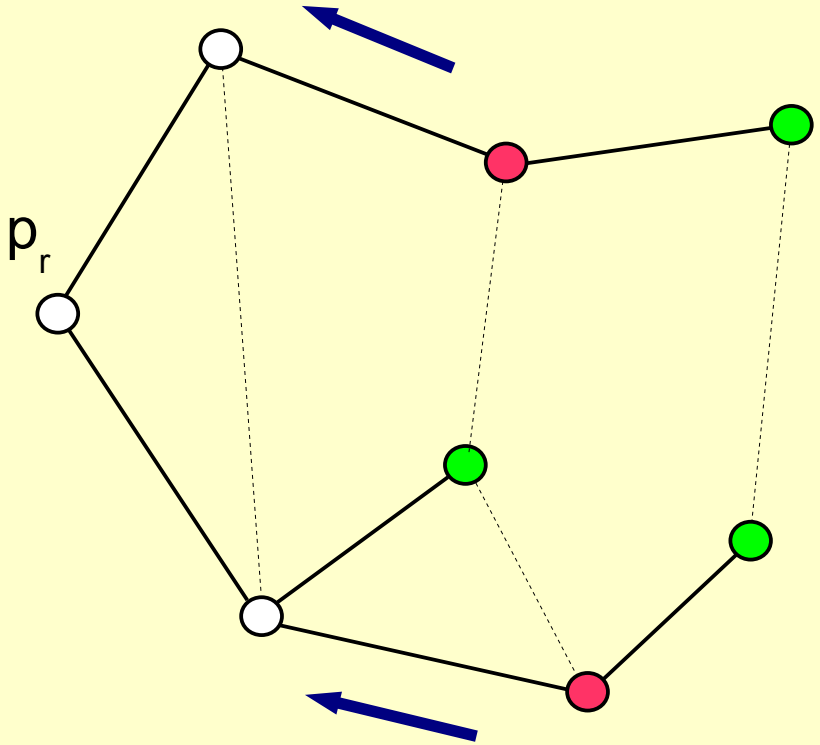
if x_j received from all children

evaluate max(x_j)

terminate

Convergecast

synchronní varianta



p_i - having no child

no message received:

send x_i to the parent

terminate

p_i - having child(ren)

message x_j received from the child:

if x_j received from all children

send max(x_j) to the parent

terminate

p_r

message x_j received from the child:

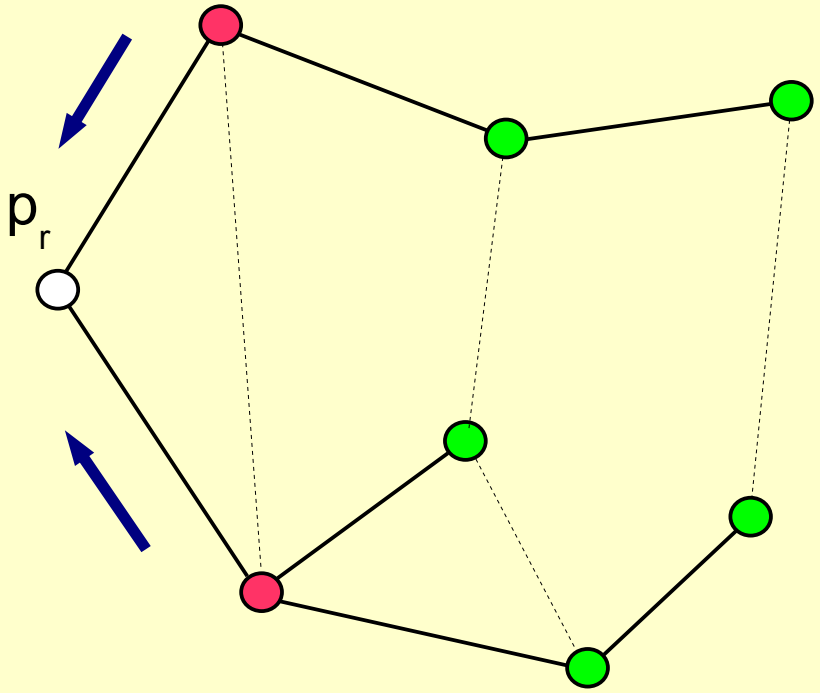
if x_j received from all children

evaluate max(x_j)

terminate

Convergecast

synchronní varianta



p_i - having no child

no message received:

send x_i to the parent

terminate

p_i - having child(ren)

message x_j received from the child:

if x_j received from all children

send max(x_j) to the parent

terminate

p_r

message x_j received from the child:

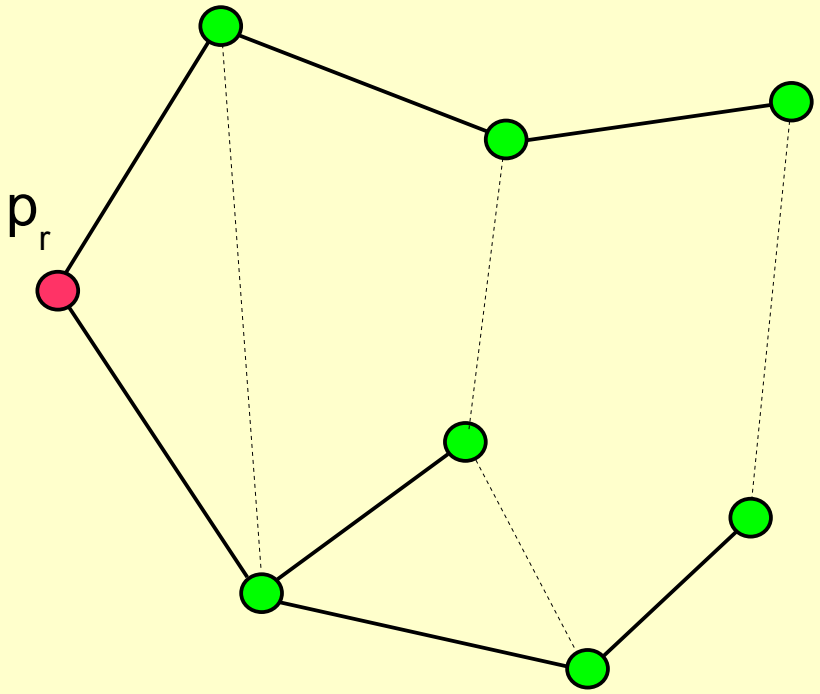
if x_j received from all children

evaluate max(x_j)

terminate

Convergecast

synchronní varianta



p_i - having no child

no message received:

send x_i to the parent

terminate

p_i - having child(ren)

message x_j received from the child:

if x_j received from all children

send max(x_j) to the parent

terminate

p_r

message x_j received from the child:

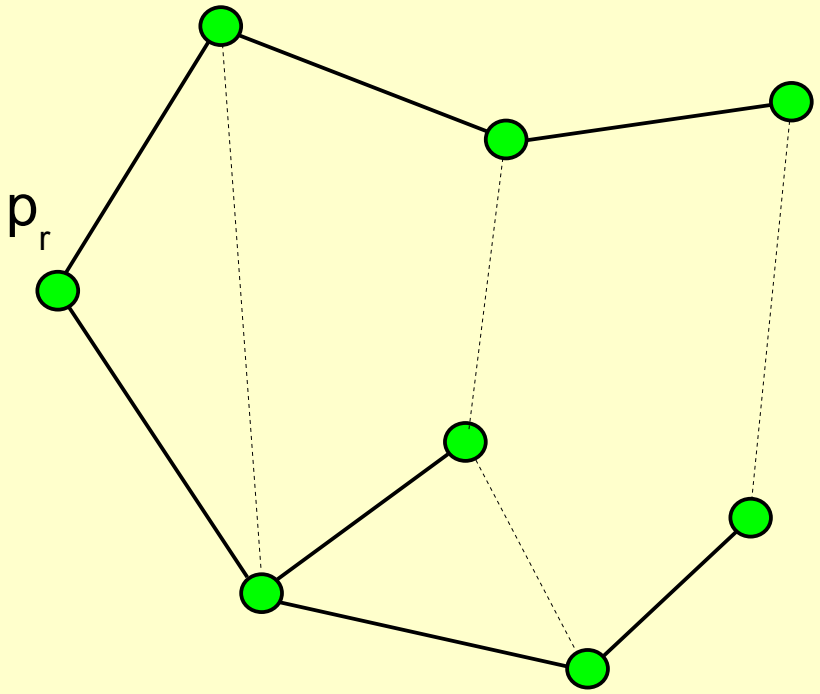
if x_j received from all children

evaluate max(x_j)

terminate

Convergecast

synchronní varianta



p_i - having no child

no message received:

send x_i to the parent

terminate

p_i - having child(ren)

message x_j received from the child:

if x_j received from all children

send max(x_j) to the parent

terminate

p_r

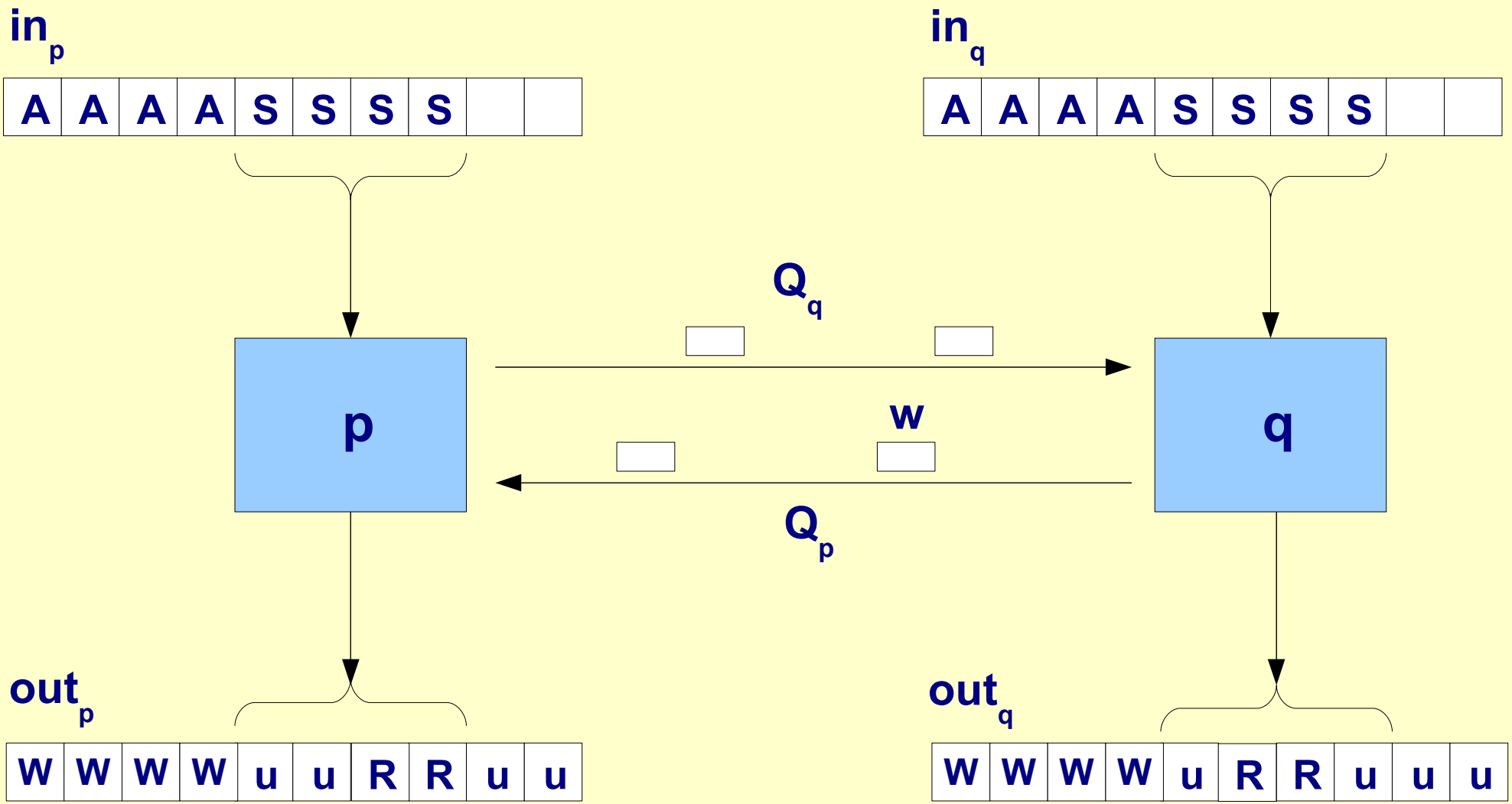
message x_j received from the child:

if x_j received from all children

evaluate max(x_j)

terminate

Balanced sliding window



Balanced sliding window

```
var s_p, a_p : integer                          init 0, 0;
    in_p : array of word                        {* data to be sent *}
    out_p : array of word                   init udef, udef, . . . ;

S_p: {a_p <= i < s_p + l_p}
    begin send(pack,in_p[i],i) to Q_q end

R_p: { <pack,w,i> in Q_p }
    begin receive (pack,w,i);
        if out_p[i] = udef  then
            begin  out_p[i] := w;
                   a_p := max(a_p,i – l_q+1) ;
                   s_p := min{ j | out_p[j] = udef }
            end
        {* else ignore, packet retransmition *}
    end
```

Balanced sliding window

Safety condition - Safe delivery :

In every reachable configuration of the protocol

$out_p[0,...s_p-1] = in_q[0,...s_p-1]$
$out_p[0,...s_p-1] = in_q[0,...s_p-1]$

Live condition - Eventual delivery :

For each integer k>=0, a configuration with

$s_p >= k$ and $s_q ->= k$

is eventually reached;