PDV 12 2018/2019

Volba lídra (koordinátora)

Michal Jakob
michal.jakob@fel.cvut.cz
Centrum umělé inteligence, katedra počítačů, FEL ČVUT

Příklad

Detaily bankovního účtu jsou replikovány na několika serverech.

Ale pouze jeden by měl být zodpovědný za příjem všech požadavků na čtení a zápis – tím je tzv. **lídr** mezi replikami.

Co kdyby:

- byli dva lídři pro jeden účet?
- se servery neshodly na tom, kdo je lídr?
- kdyby lídr havaroval?

Všechny výše uvedené situace by mohl vést k nekonzistencím.

Problém volby lídra

Ze skupiny procesů **vybrat lídra** (který bude řešit specifické úkoly) a **dát vědět všem procesům** ve skupině, kdo je lídrem.

Co se stane, když lídr selže?

- nějaký proces detekuje pomocí detektoru selhání a spustí nové volby

Algoritmus pro volbu lídra musí zajistit:

1. zvolí právě jednoho lídra z bezvadných procesů
2. všechny bezvadné procesy ve skupině se shodnou na tom, kdo je lídr

Systémový model

Skupina N procesů s unikátními identifikátory.

- známe všechny procesy, ale nevíme, které jsou aktivní (bezvadné)

Procesy **mohou havarovat**.

FIFO perfektní komunikační kanál mezi každým párem procesů, tj. zprávy se neduplikují, nevznikají, neztrácejí a jsou doručovány v pořadí odeslání.

Asynchronní systém: neznáma, ale **konečná latence.**

Další požadavky

Každý proces může **vyvolat** volby.

Jeden proces může vyvolat v jeden okamžik pouze **jedny volby**.

Více procesů může vyvolat volby **současně** - pak požadujeme, aby se nakonec shodly na jednom lídrovi.

Výsledek volby lídra by **neměl záviset** na tom, který proces volby **vyvolal**.

Po skončení běhu algoritmu volby lídra má každý proces ve své proměnné $ELECTED$ identifikátor lídra s nejvyšší hodnotou volebního kritéria.

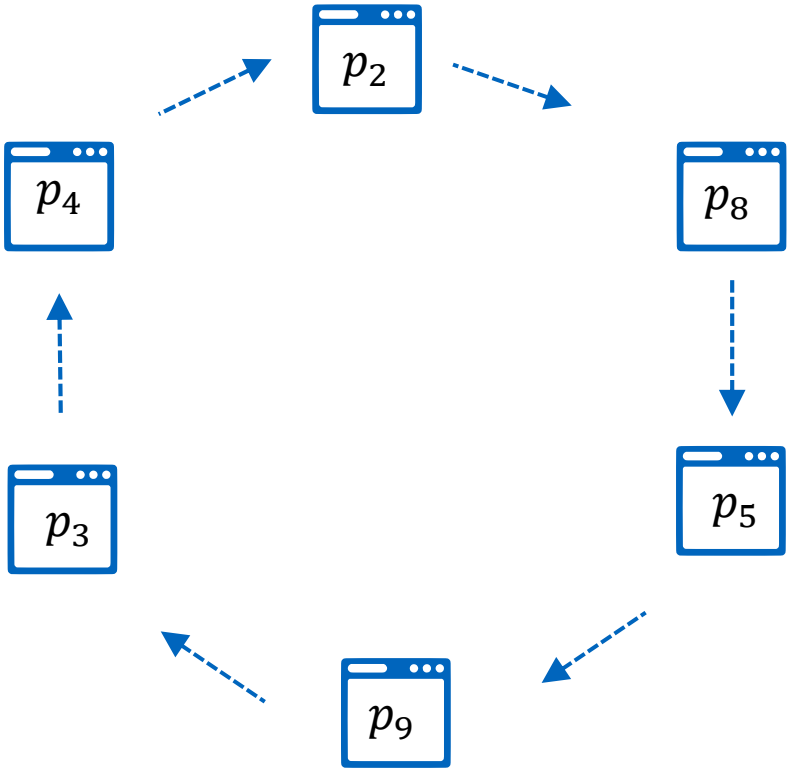
Volební kritérium:

- typicky nejvyšší identifikátor, tj. IP adresa
- ale taky např: nejvíce RAM, diskového prostoru, nebo např. nejvíce souborů v případě P2P sítí.
- musí být fixní a známe všem procesům při zahájení volby

Kruhový algoritmus

Procesy jsou uspořádané do logického kruhu.

Zprávy jsou posílány dokola kruhu **v jednom směru.**



Kruhový algoritmus

P_i: zahájení voleb (po detekci selhání)

P_i pošle po kruhu zprávu ELECTION(i) obsahující jeho identifikátor

P_i: zpracování zprávy ELECTION(j)

if $j > i$ **then** přepošle zprávu ELECTION(j)

if $i < j$ **then** nahradí zprávu za ELECTION(i) a pošle dál

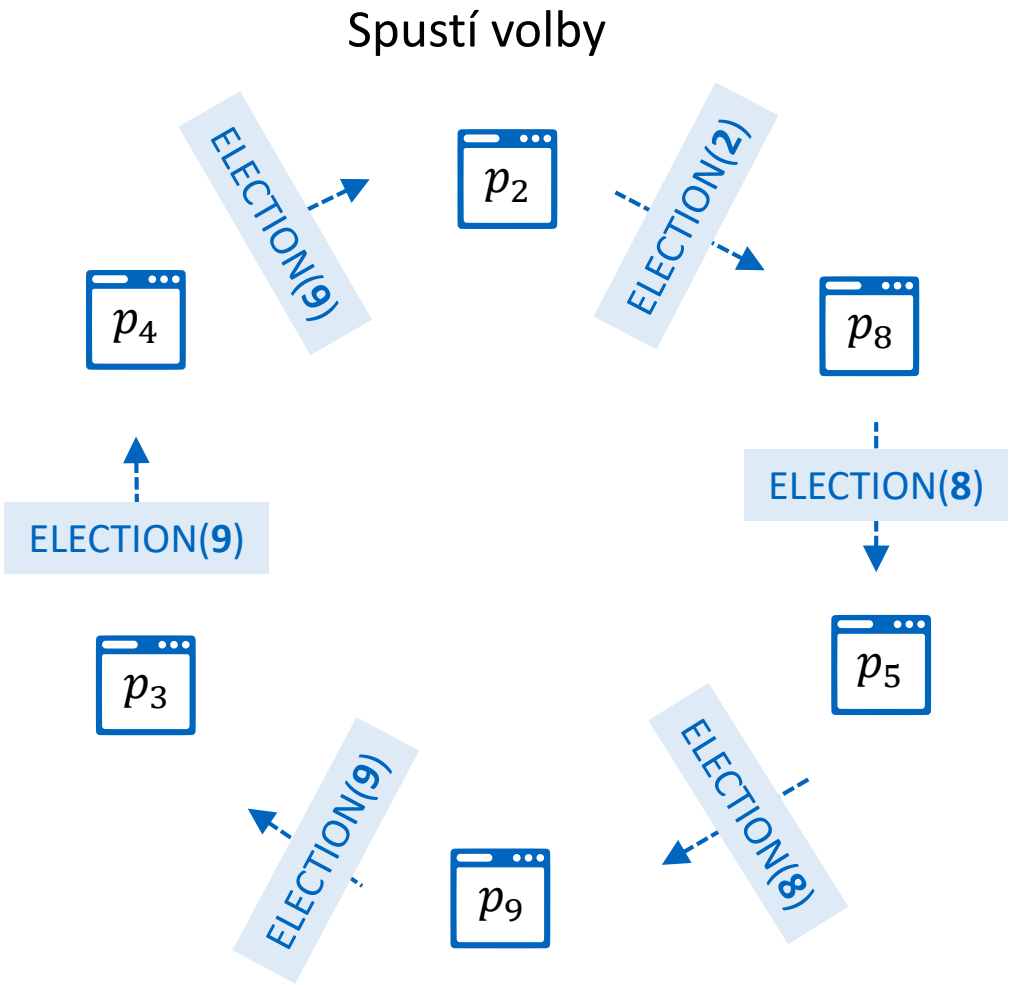
if $i = j$ **then** P_i je nový koordinátor; odešle zprávu ELECTED(i)

P_i: přijetí zprávy ELECTED(j)

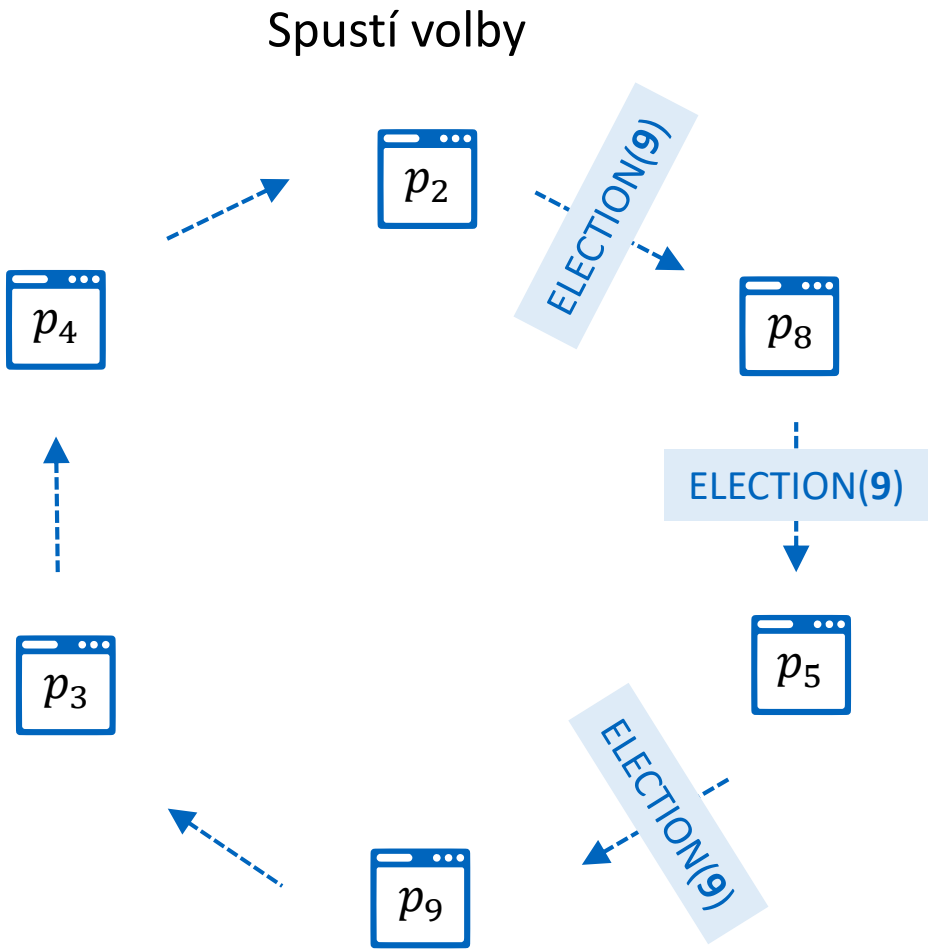
Nastaví svou proměnou $\texttt{elected}_i := j$

if $i \neq j$ **then** přepošle zprávu **ELECTED(j)**

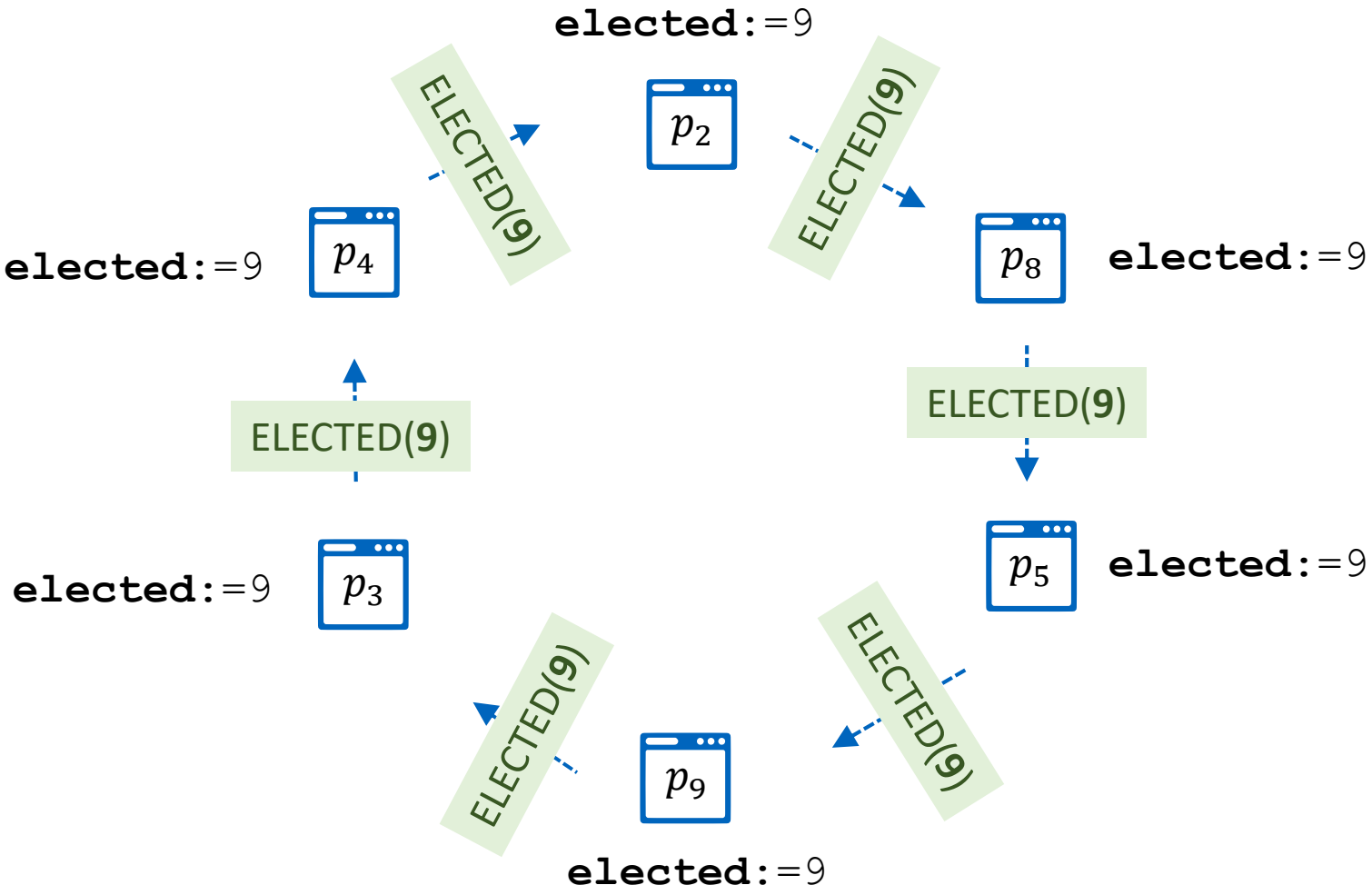
Příklad


Spustí volby
p_2
ELECTION(2)
p_8
ELECTION(8)
p_5
ELECTION(8)
p_9
ELECTION(9)
p_3
ELECTION(9)
p_4
ELECTION(9)

Příklad


Spustí volby
p_2
ELECTION(9)
p_8
ELECTION(9)
p_5
ELECTION(9)
p_9
p_3
p_4

Příklad


elected:=9
p_2
ELECTED(9)
ELECTED(9)
p_8
elected:=9
elected:=9
p_4
ELECTED(9)
ELECTED(9)
p_5
elected:=9
elected:=9
p_3
ELECTED(9)
ELECTED(9)
p_9
elected:=9

Analýza

N procesů; předpokládáme, že v průběhu volby nedochází k selháním

Nejhorší případ

- iniciátor je následník budoucího lídra
- $N - 1$ zpráv pro zprávu ELECTION než se dostane od iniciátora P_i k budoucímu lídrovi P_j
- N přeposlání nezměněné zprávy ELECTION(j) okolo kruhu
- N přeposlání zprávy ELECTED(j) okolo kruhu
- Celkem $\boldsymbol{3N - 1}$ zpráv
- Čas běhu: $\boldsymbol{3N - 1}$ komunikačních latencí

Nejlepší případ

- iniciátor je budoucí lídr
- Celkem $\boldsymbol{2N}$ zpráv
- Čas běhu: $\boldsymbol{2N}$ komunikačních latencí

Pokud nejsou selhání, tak volba skončí **v konečném čase** (živost) a všechny procesy znají **stejného lídra** (bezpečnost)

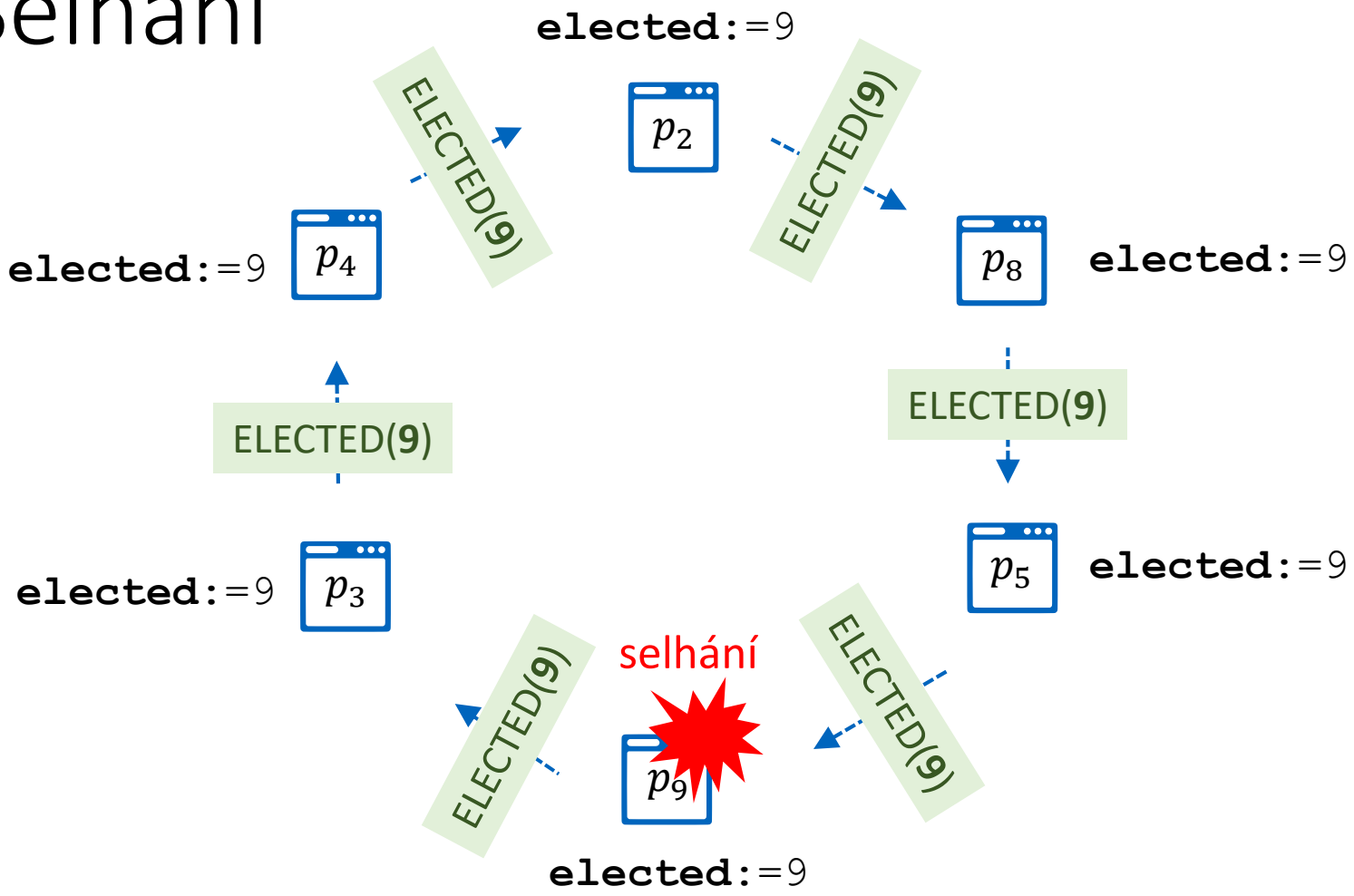
Více iniciátorů

Proces si pamatují nejvyšší `maxID` ze všech dosud obdržených ELECTION/ELECTED zpráv (od poslední úspěšné volby lídra).

Procesy ignorují ELECTION/ELECTED zpráv s nižším ID než `maxID`.

Pouze zprávy od iniciátora s nejvyšším ID dokončí volební cyklus.

Selhání



Selže-li P_9 po odeslání zprávy, tak ELECTION(9) bude cirkulovat do nekonečna i pokud dojde k obnovení topologie kruhu ➔ **porušení živosti**.

Jak řešit selhání?

Možnost 1: Předchůdce nebo následovník rádoby lídra P_k detekuje selhání a spustí nové volby

- Pokud obdrží zprávu ELECTION(k), ale neobdrží odpovídající zprávu ELECTED(k)
- Pokud obdrží podruhé zprávu ELECTED(k)

Ale co když selže předchůdce? A předchůdce předchůdce? …

Možnost 2: Využití detektoru selhání.

- Kterýkoliv proces po přijetí zprávy ELECTION(k) může detekovat selhání P_k pomocí svého lokálního detektoru selhání
- Jakmile detekuje selhání, iniciuje nové volby

Ale: Detektory selhání nemohou být zároveň úplné a přesné:

- neúplný detektor ⇒ selhání nemusí být detekované ⇒ systém zůstane bez lídra
- nepřesný detektor ⇒ lídr může být mylně detekován jako havarovaný ⇒ zbytečně opakované volby, potenciálně neomezený počet krát ⇒ narušení živosti

Bully Algoritmus

Bully algoritmus

Klasický algoritmus pro volbu lídra.

Základní princip: proces, který **detekoval selhání** dosavadního lídra, **vyzve** procesy s **vyšším ID** ve volbách.

Bully algoritmus

P_i: zahájení voleb (po detekci selhání nebo jako reakce na volby)

if P_i má nejvyšší ID
    Pošli COORDINATOR zprávu všem procesům s nižšími identifikátory (volby končí).
else // zahájí volby
    Pošli ELECTION zprávu všem procesům s vyšším ID
    // následně čekání na odpověď

P_i: reakce na ELECTION

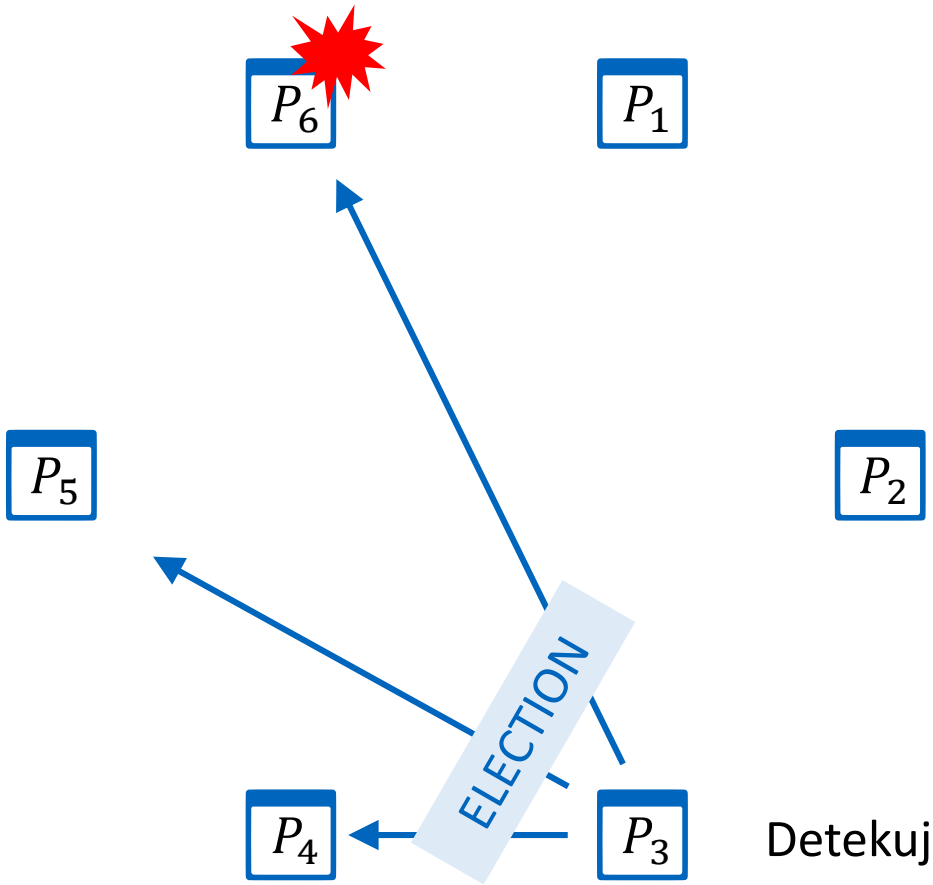
Odpověz OK
If pokud P_i dosud nezahájil volby
    zahaj volby

P_i: čekání na odpovědi (po vyvolání voleb)

if nedorazí žádná odpověď v časovém limitu
    prohlaš se jako P_i za lídra;
    pošli COORDINATOR zprávu všem procesům s nižším ID;
    // volby skončily

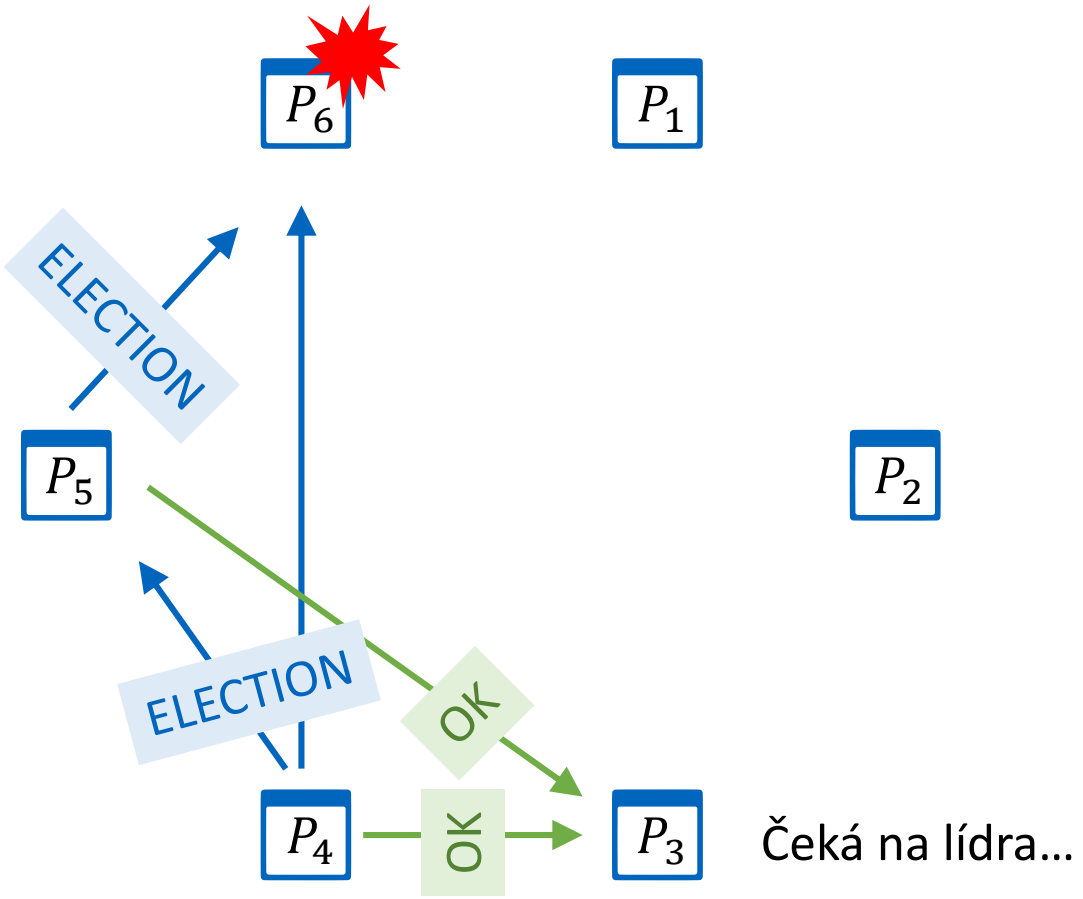
else // existuje aktivní proces s vyšším ID než P_i
    čekej na zprávu COORDINATOR;
    pokud nedorazí v časovém intervalu, iniciuj nové volby

Bully algoritmus

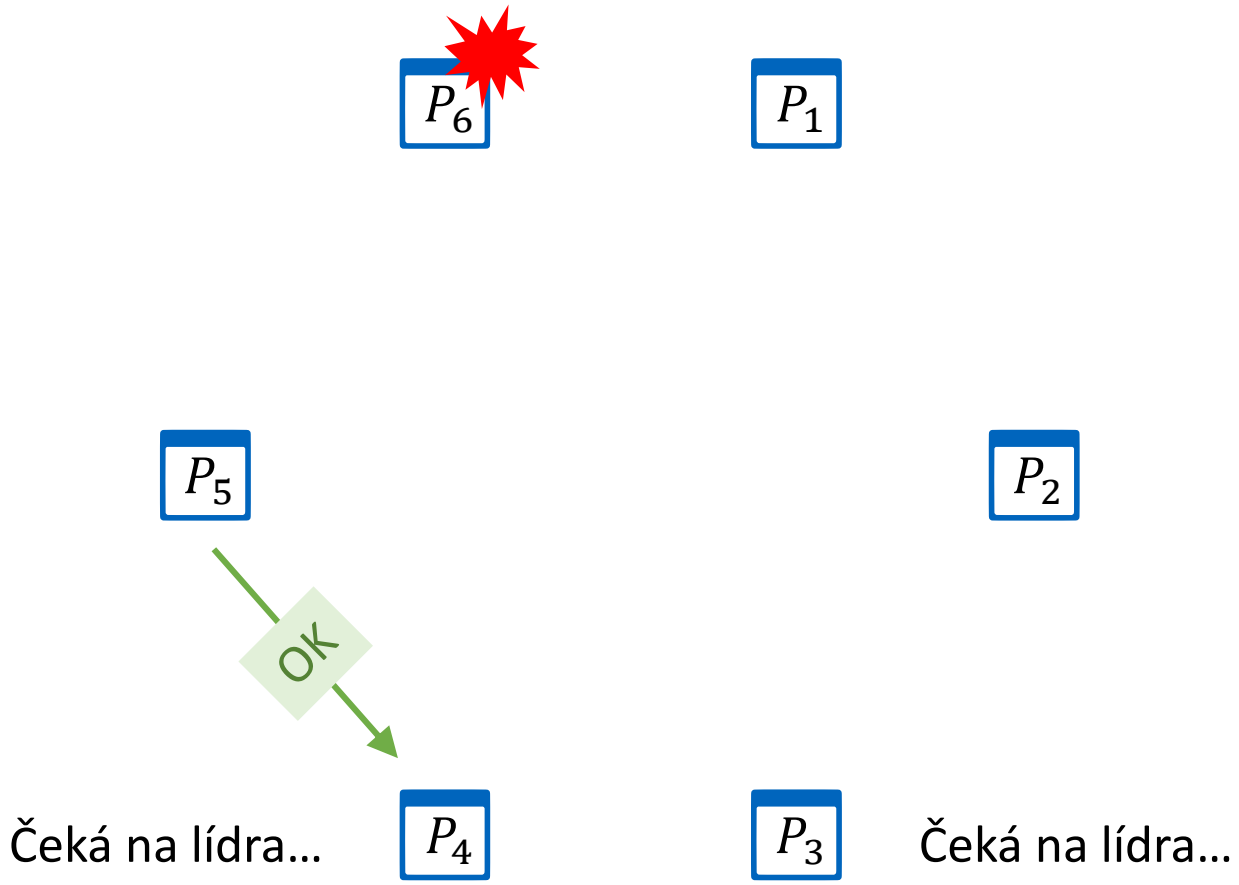


Detekuje selhání P_6

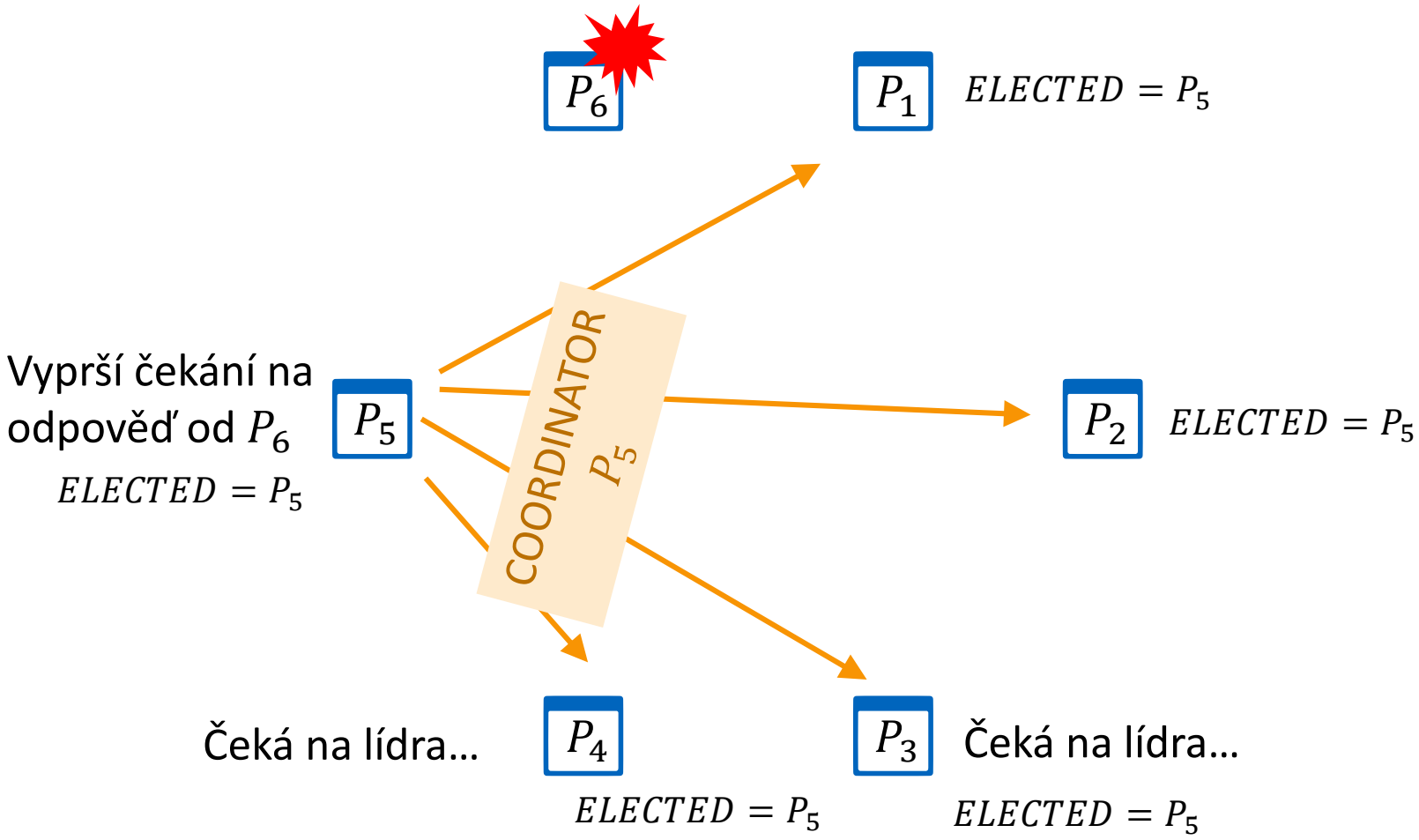
Bully algoritmus



Bully algoritmus

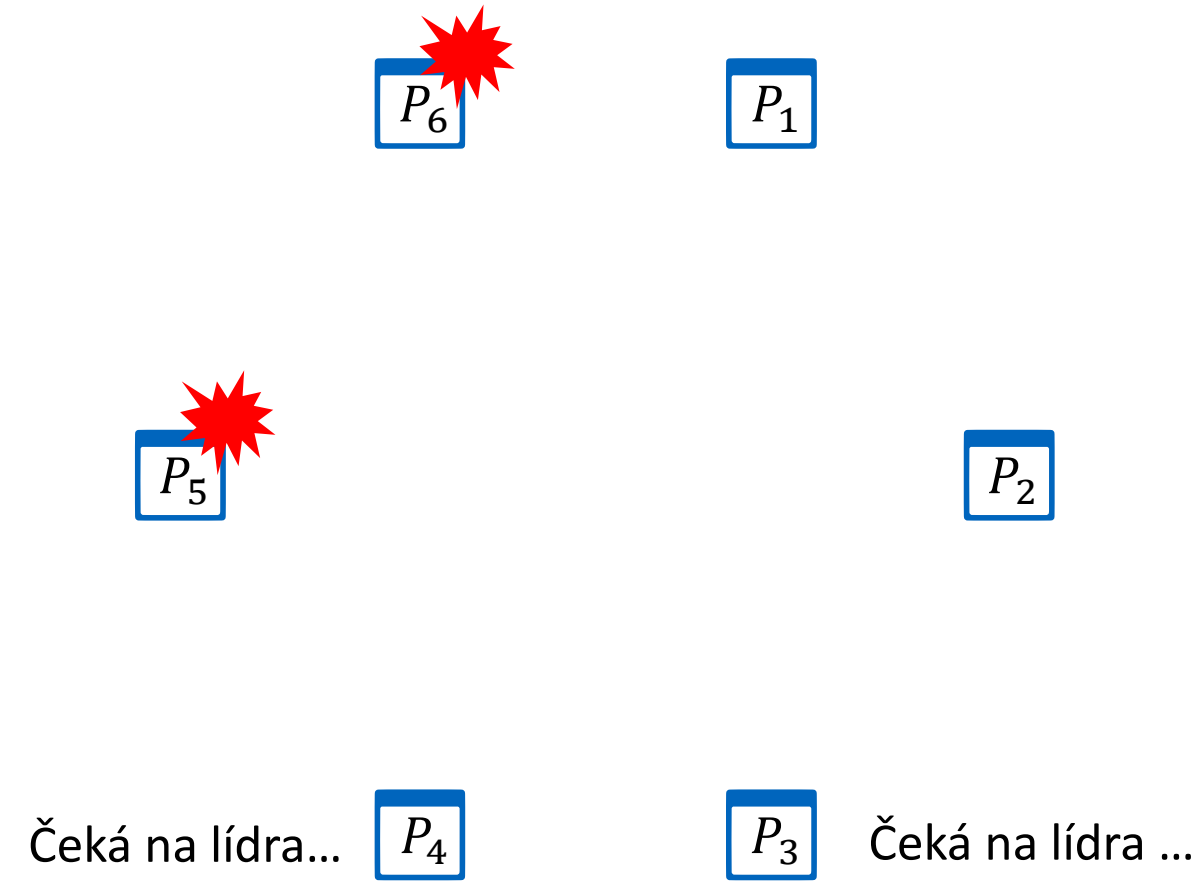


Bully algoritmus

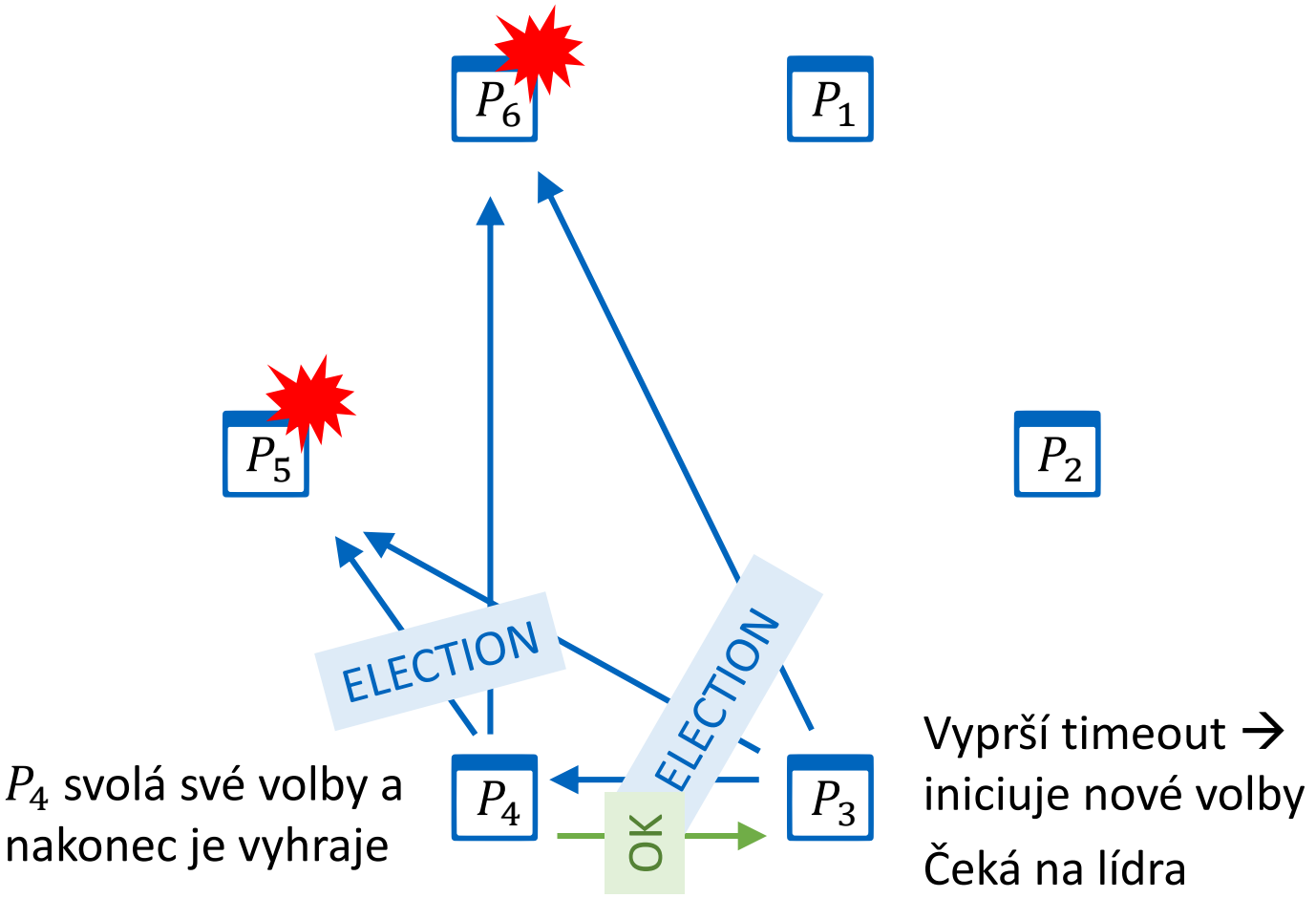


Volby jsou skončeny

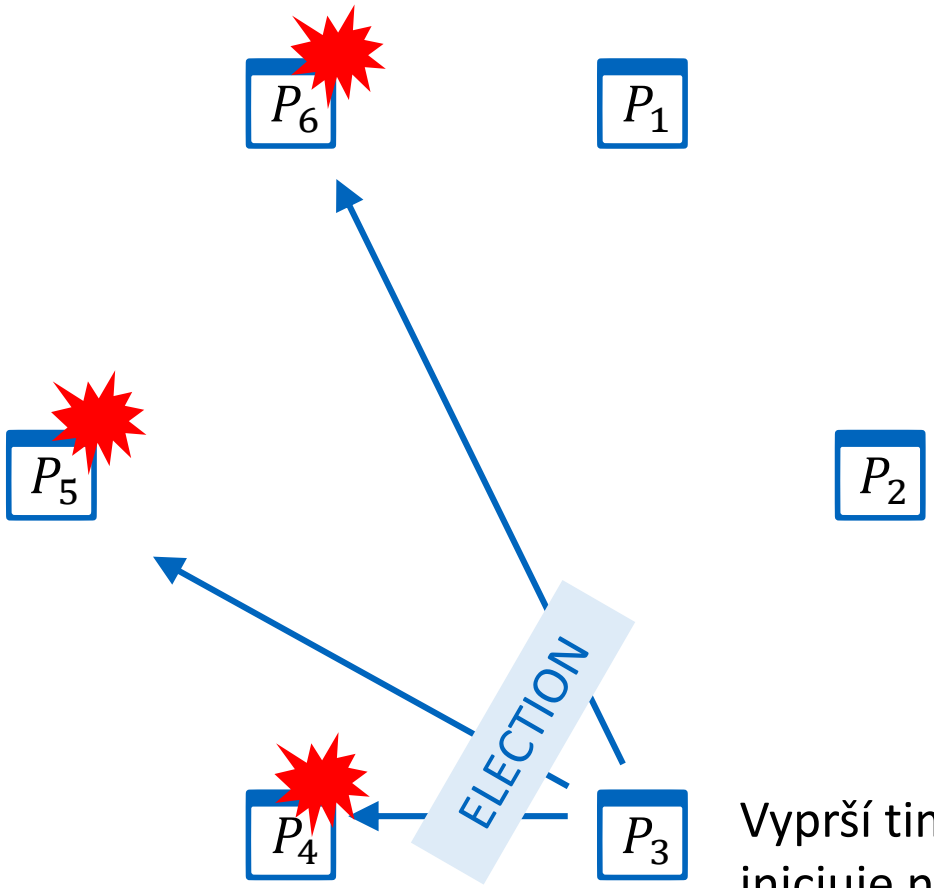
Selhání během volby



Selhání během volby



Selhání během volby



Vyprší timeout → iniciuje nové volby a nakonec je vyhraje

Analýza

Nejhorší případ – selhání lídra detekováno procesem s nejnižším ID.

Pro dokončení volby lídra je v nejhorším případě potřeba **čtyři komunikační latence:**

1. Proces s nejnižším ID pošle ELECTION zprávu ostatním procesům
2. Proces s druhým nejvyšším ID:
   pošle OK procesům s nižšími ID
   pošle ELECTION procesu s nejvyšším ID
3. Procesu s druhým nejvyšším ID vyprší timeout při čekání na odpověd procesu s nejvyšším ID
4. Proces s druhým nejvyšším ID rozešle zprávu COORDINATOR

Nejhorší případ – komunikační zátěž

- celkem $N - 1$ procesů pošle zprávu ELECTION procesům s vyšším ID
- tj. celkem: $N - 1 + N - 2 + \cdots + 1 = \frac{N(N-1)}{2} = \boldsymbol{O(N^2)}$ **zpráv**

Analýza

Nejlepší případ: proces s druhým nejvyšším ID detekuje selhání lídra

Nejlepší případ - komunikační zátěž:

- $(\boldsymbol{N-2})$ zpráv COORDINATOR
- čas do ukončení: **1 komunikační latence**

Živost

Pokud přestanou selhávat další procesy, dojde časem ke zvolení lídra.

Bully algoritmus pracuje s timeouts → **v asynchronním systému** jeho běh nemusí nikdy skončit → **živost není garantována**.

V **synchronním systému:**

- lze spočítat nejhorší jednosměrnou latence = doba přenosu zprávy + doba reakce na zprávu.
- pokud timeout nastavíme na násobek nejhorší jednosměrné latence, je **živost garantována**.

Proč je volba lídra tak těžké?

Protože souvisí s problémem konsensu!

Pokud bychom byli schopni vyřešit volbu lídra, tak umíme řešit konsensus.

Stačí zvolit lídra a poslední bit jeho ID interpretovat jako výsledek konsensu.

Ale protože konsensus není řešitelný v asynchronních DS se selháními, není řešitelná ani volba lídra.

(řešitelnost = existence algoritmu garantujícího bezpečnost i živost současně)

Souhrn

Volba lídra důležitý problém v DS.

Bully algoritmus klasický algoritmus předpokládající selhání procesů, ale perfektní FIFO kanál.

V asynchronním systému **negarantuje živost**; v synchronním nebo částečně synchronním systému lze (konečnou) živost (tzv. eventual liveness) dosáhnout vhodným **nastavením timeout**.