Vzájemné vyloučení v distribuovaném prostředí

PA 150 ◇ Principy operačních systémů

Jan Staudek
http://www.fi.muni.cz/usr/staudek/vyuka/

Verze : podzim 2020

Distribuované vzájemné vyloučení

- Distributed Mutual Exclusion (DME)
- Dva a/nebo více procesů běžících v DS obsahují kritické sekce, KS, sdružené s jistým sdíleným objektem,
- KS v těchto procesech se musí při běhu procesů vzájemně vyloučit, tj. musí se provést sériově
- Neexistují sdílené proměnné typu semafor ani nelze pro implementaci použít některé lokální jádro OS
- Vzájemné vyloučení musí být založeno výhradně na výměně zpráv, a to pomocí asynchronní výměny zpráv a bez znalosti stavu systému jako celku
- Proces provádějící kritickou sekci (nebo do ní vstupující) označíme jako privilegovaný proces

Distribuované vzájemné vyloučení

- Podmínka bezpečnosti – dosáhne se vzájemného vyloučení
  - ✓ V každé konfiguraci DS je privilegovaný nejvýše jeden proces
- Podmínka živosti – zamezuje se stárnutí procesů
  - ✓ Předpoklad – žádný proces nezůstane privilegovaný trvale
  - ✓ Jestliže proces p zkouší vstoupit do KS,
    pak se stane privilegovaný v konečném čase
- podmínka spravedlivosti, podpodmínka živosti
  - ✓ Buďto –

    pořadí vstupů do KS = pořadí vydání žádostí o vstup do KS
  - ✓ Nebo obecnější omezení –

    každý ostatní proces z DS smí žádající proces předběhnout nejvýše 1x

Distribuované vzájemné vyloučení, způsoby řešení

- Řízení přístupu ke zdroji v DS v rámci kritické sekce může být zajišťované servery spravujícími dané zdroje – model klient-server, zámky (*locks*), konstantní složitost, typické řešení pro databázové systémy s transakčním zpracováním, pro žádající procesy je řešení vzájemné výlučnosti transparentní
  (viz transakční zpracování na závěr cyklu přednášek)
- Distribuovaná řešení mohou být založena
  - na předávání příznaku (*token*) práva vstupu do kritické sekce mezi procesy nebo
  - na centralizované správě jediným serverem v systému
  - nebo na distribuované dohodě žádajících procesů

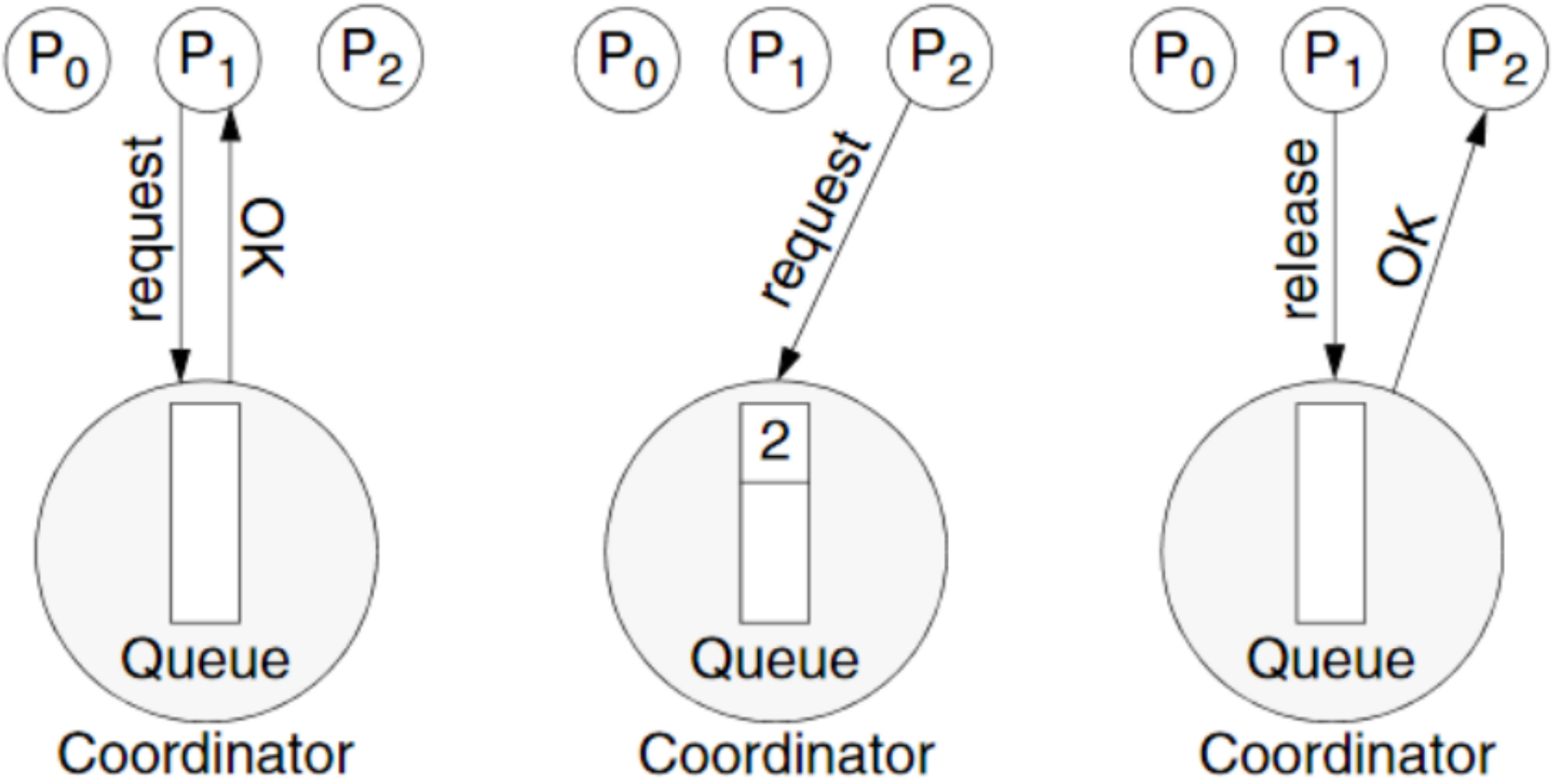
Distribuované vzájemné vyloučení, způsoby řešení

- Distribuovaná řešení mohou regulárnosti komunikačních cest
  (v závorkách se uvádí složitost)
  - ✓ token ring, kruh – předávání práva vstupu pro kruhu (n)
    proces držící právo vstupu je privilegovaný
  - ✓ tree, strom – algoritmus *Raymond* ($\log n$)
  - ✓ založené na získání souhlasu partnerských procesů
  - ✓ token-passing – algoritmus *Suzuki-Kasami* (n)
  - ✓ časová razítka – algoritmus *Ricart-Agrawala* ($2n-1$),
    distribuovaná fronta
  - ✓ hlasovací kvóra – algoritmus *Maekawa* ($\sqrt{n}$),
    aby se proces stal privilegovaný, musí získat souhlas
    od jistého kvora procesů, každý pár kvór musí mít neprázdný průnik

DME – centralizované řešení

- Jeden z procesů v systému – koordinátor vstupu do KS (server)
- Proces, který chce vstoupit do KS (klient), zasílá koordinátorovi zprávu se žádostí o vstup do KS – request
- O pořadí vstupu procesů do KS rozhoduje koordinátor, procesu žádajícímu o vstup do KS posílá koordinátor zprávu – reply – povolující vstup do KS
- Jakmile proces (klient) přijme zprávu reply, vstupuje do KS
- Jakmile proces (klient) opouští KS, posílá koordinátorovi zprávu release

DME – centralizované řešení



- ✓ OK je zpráva s významem reply

DME – centralizované řešení

- Každý vstup do KS požaduje zaslání 3 zpráv
  - ✓ request, reply, release
- Vlastnosti
  - ✓ za splnění podmínky bezpečnosti a živosti algoritmus odpovídá server, logický čas procesů se přitom nemusí sledovat
  - ✓ Počet procesů soupeřících o vstup do KS není limitovaný
  - ✓ Algoritmus nesplňuje podmínku spravedlivosti, protože nerespektuje logický čas v DS
  - ✓ Výkon serveru a komunikační výkon cest k serveru mohou představovat úzké místo
  - ✓ Výpadek serveru způsobí výpadek celého DS, výpadek klienta ostatní klienty neovlivňuje
  - ✓ Koordinátorem může být jeden z procesů, které soutěží o přístup; aby byl vybrán jediný nový koordinátor, musí procesy provést volební algoritmus (viz později)

DME – *Ricart, Agrawala* (1981)

- vstup do KS se řeší distribuovanou dohodou procesů
- procesy udržují (Lamportovy) logické hodiny, při shodě časových razítek žádostí o prioritě žádosti rozhoduje (jedinečné) id procesu
- proces žádá o vstup do KS zprávou reguest rozeslanou N procesům ve skupině
- zprávou reply oslovený proces dává žádajícímu procesu souhlas ke vstupu do KS
- zprávy request jsou časově razítkované (logickým časem), časové razítko určuje prioritu konfliktních požadavků
- Když proces P_i chce vstoupit do KS, vygeneruje nové $TS_i = TS_i + 1$, a zašle všem procesům request (P_i, TS_i)

DME – *Ricart, Agrawala* (1981)

- Když proces P_j přijme zprávu request (P_i, TS_i),
  odpovídá reply procesu P_i buď to
  okamžitě (ani není v KS, ani nežádá o vstup do KS) nebo
  opožděně (je v KS nebo požádal o vstup do KS dříve)
  - ✓ P_j požádal o vstup do KS, ale ten mu dosud nebyl povolen,
    tj. rozeslal request (P_j, TS_j), ale nezískal od všech procesů reply,
    pak porovnává své TS_j s TS_i přijatým v request (P_i, TS_i)
    - je-li jeho TS_j větší než TS_i ze zprávy, posílá reply P_i okamžitě,
      protože P_i žádal o vstup do KS dříve
    - jinak zaslání reply procesu P_i odkládá do výstupu z KS
- Když proces P_i přijme zprávu reply od všech $N-1$ procesů,
  může vstoupit do KS
- Po opuštění KS posílá proces zprávy reply všem procesům,
  kterým dosud na zprávu request neodpověděl reply

DME – *Ricart, Agrawala* (1981), idea programu

On initialization

state := RELEASED;

To enter the section

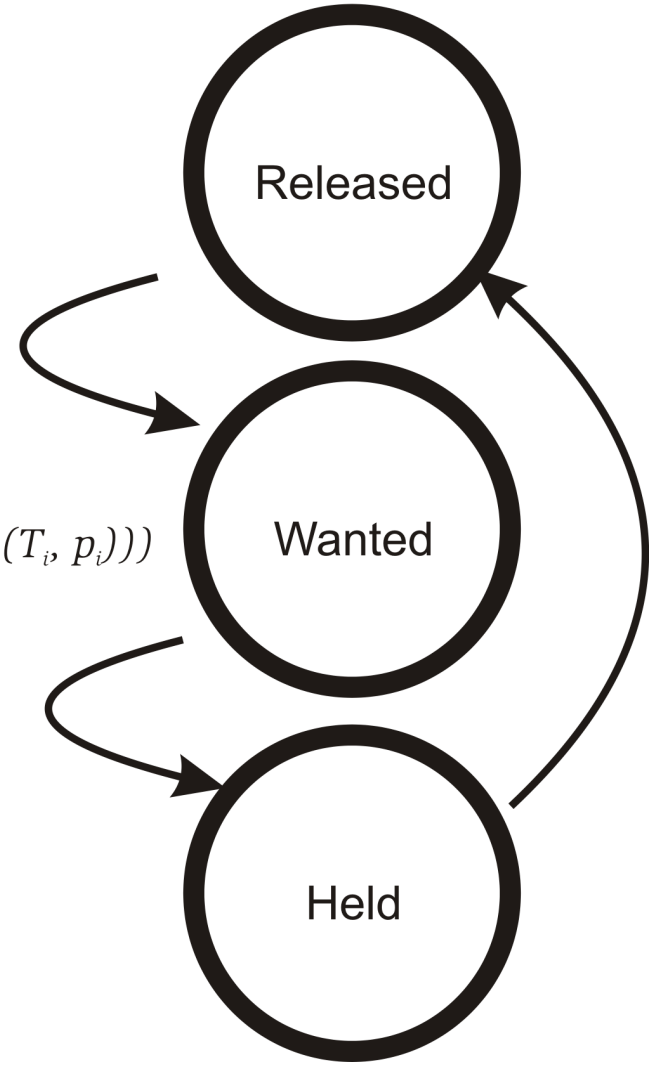
state := WANTED;
Multicast request to all processes;
T := request's timestamp;
Wait until (number of replies received = (N - 1));
state := HELD;

On receipt of a request $<T_i, p_i>$ at p_j ($i \neq j$)

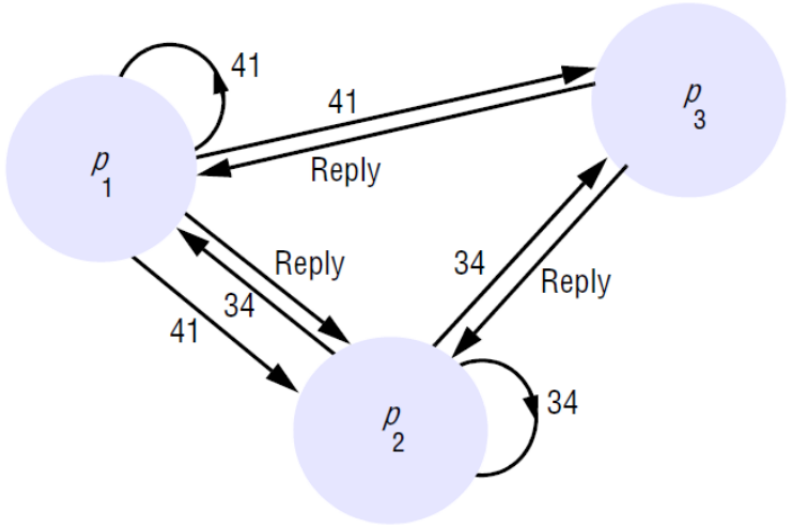
if (state = HELD or (state = WANTED and $(T, p_j) < (T_i, p_i)$))
then
queue request from p_i without replying;
else
reply immediately to p_i;
end if

To exit the critical section

state := RELEASED;
reply to any queued requests;

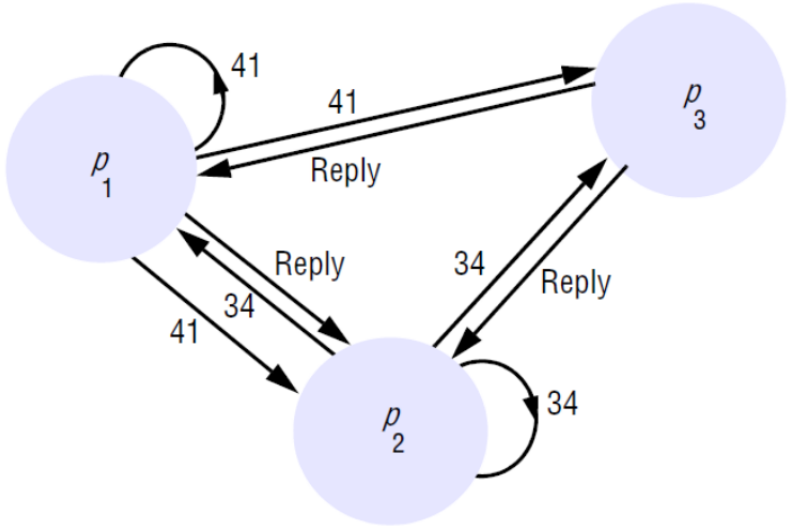


DME – *Ricart, Agrawala, příklad* (1981)



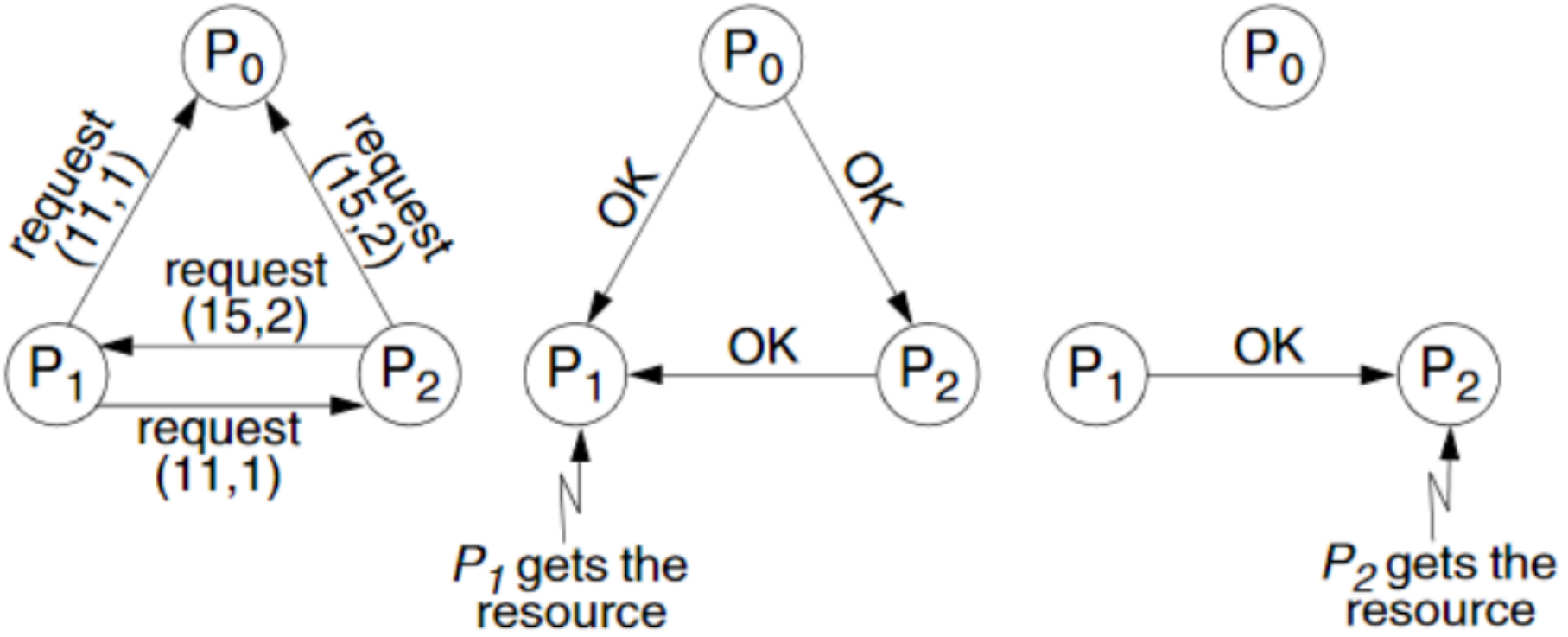
- ✓ Nechť p_3 nemá zájem o vstup do kritické sekce
- ✓ Nechť p_1 a p_2 požadují vstup do kritické sekce současně
- ✓ Časové razítko žádosti p_1 je 41 a žádosti p_2 je 34.
- ✓ p_3 na žádosti odpovídá bez prodlení
- ✓ Když p_2 obdrží žádost p_1, zjistí, že jeho vlastní žádost má nižší časové razítko, tak neodpovídá, drží p_1 mimo hru.

DME – *Ricart, Agrawala,* (1981), příklad



- ✓ Když p_1 zjistí, že požadavek p_2 má nižší časové razítko než jeho žádost, odpovídá okamžitě.
- ✓ p_2 vstoupí do kritické sekce, získal souhlas od $N - 1$ procesů
- ✓ Když p_2 opustí kritickou sekci, odpoví na žádost p_1 a p_1 může vstoupi do kritické sekce

DME – *Ricart, Agrawala,* (1981), příklad 2



DME – *Ricart, Agrawala* (1981)

- Pozitivní vlastnosti
  - ✓ je splněna podmínka bezpečnosti

    N–1 procesů potvrzuje žádajícímu procesu,
    že do KS nevstoupí dokud žádající proces KS neopustí
  - ✓ je splněna podmínka živosti,

    – konfliktní žádosti jsou řešeny v pořadí dle běhu logického času
  - ✓ Jestliže proces opustivší KS nedostane žádnou žádost o vstup do KS,
    může do KS vstoupit bez získání souhlasu ostatních procesů

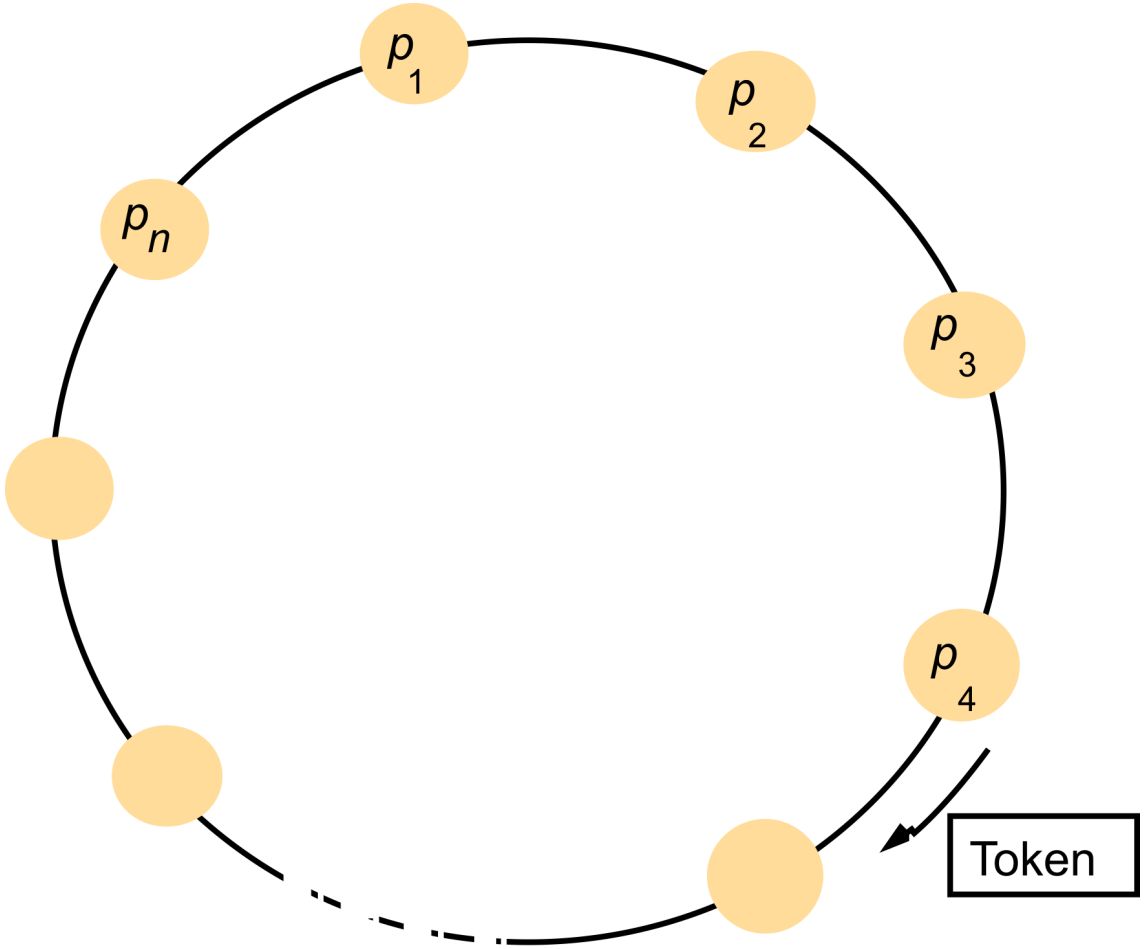
DME – *Ricart, Agrawala* (1981)

- Nežádoucí vlastnosti
  - ✓ Minimální počet zpráv na jeden vstup do KS procesem je $2\times(n-1)$, kde n je počet procesů, což je hodně
  - ✓ proces musí znát identitu všech ostatních procesů v systému, dynamické doplňování a rušení procesů je netriviální
  - ✓ výpadek jednoho procesu způsobí kolaps celého systému (stav všech procesů je potřeba trvale monitorovat)
  - ✓ protokol je vhodný pro malé, stabilní množiny kooperujících procesů

DME – předáváním příznaku po kruhu

- Procesy jsou v distribuovaném systému uspořádané do kruhu
  - ✓ ať již logicky nebo fyzicky
  - ✓ nechť kruh je jednosměrný, každý proces může poslat zprávu pouze svému (např.) pravému sousedovi
- Příznak – speciální zpráva – pešek
- Příznak v DS existuje v jediném exempláři
- Proces, který chce vstoupit do KS,
  může tak učinit, až když obdrží příznak
- Proces, který obdrží příznak
  - buďto vstoupí do KS, pokud chce vstoupit do KS
    a po výstupu z KS příznak pošle následníkovi na kruhu
  - nebo, nechce-li vstupovat do KS,
    příznak pošle následníkovi bez prodlení

DME – předáváním příznaku po kruhu



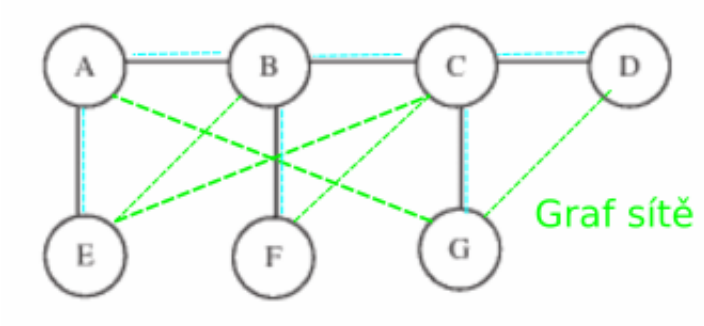
DME – předáváním příznaku po kruhu

- Přednosti řešení
  - ✓ je splněna podmínka bezpečnosti
  - ✓ je splněna podmínka živosti
    nedojde ani k uváznutí ani ke stárnutí
  - ✓ pokud se zná maximální doba řešení KS a počet procesů v kruhu
    je známá i maximální prodleva procesu před vstupem do KS
  - ✓ podmínka spravedlivosti splněna je, avšak ne v pořadí logického času,
    ale se zajištěním, že žadatele může každý proces předběhnout pouze 1x
- Nedostatky řešení (nejsou řešitelné triviálně)
  - ✓ Ztráta příznaku se musí řešit distribuovanou volbou procesu,
    který bude nový příznak generovat
  - ✓ Výpadek jednoho uzlu v kruhové síti se musí řešit rekonstrukcí kruhu

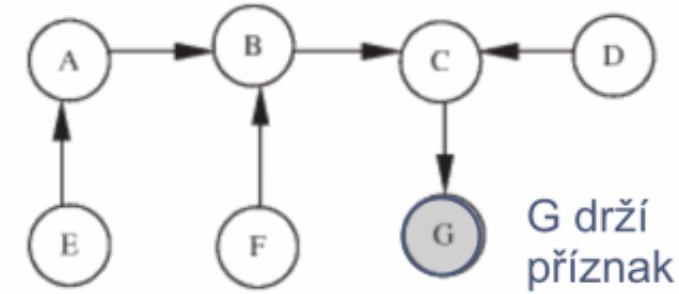
DME, *Raymond*, předávání příznaku po stromu

- Na uzly distribuovaného systému se superponuje kostra
- Po kostře se předává příznak povolující vstup do KS
- Příznak existuje v jediném exempláři,
  drží ho uzel v této konfiguraci tvořící kořen stromu pokrývající kostru grafu, je tudíž privilegovaný
- Pokud příznak povolující vstup do KS drží uzel i,
  pak je tento uzel v tomto okamžiku kořenem stromu pokrývajícím kostru
  a všechny uzly obsahují ukazatel na svého rodiče v tomto stromu
- Při předání příznaku jinému uzlu se konfigurace stromu (orientace hran) aktualizuje

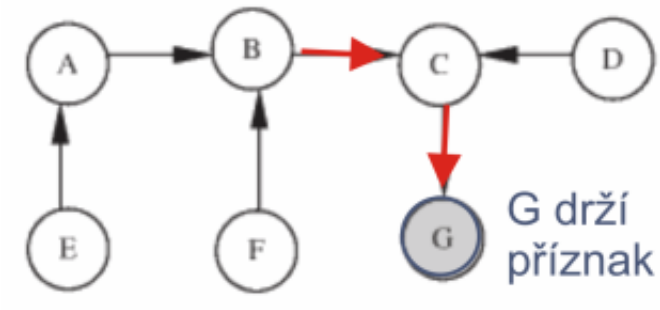
DME, *Raymond*, předávání příznaku po stromu



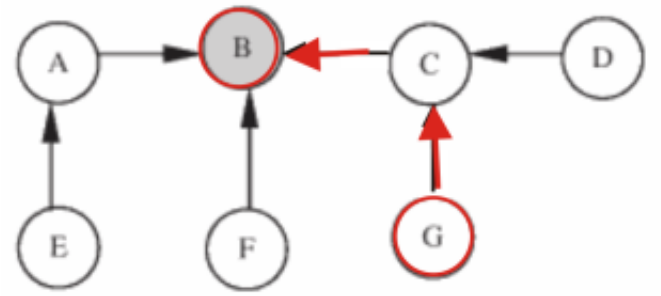
Superponovaná kostra



Iniciální kořenový strom



B požádal svého rodiče o vstup do KS,
rodič B předává žádost po cestě ke kořenu



Dosavadní kořen G se nenachází v KS
a proto předává právo vstupu do KS
po reverzní cestě procesu B,
B se stává novým kořenem stromu

DME, *Raymond*, předávání příznaku po stromu

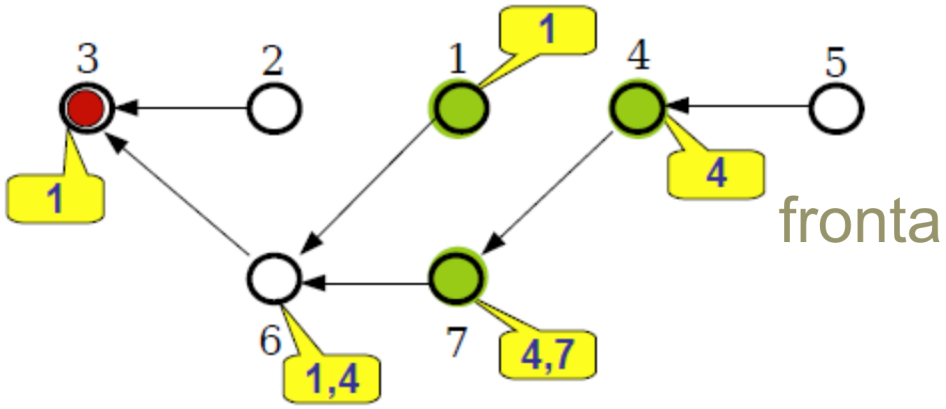
- Chování uzlu
  - ✓ každý uzel s výjimkou kořene stromu má v každém okamžiku pouze jednoho rodiče, kterému předává svůj požadavek na získání příznaku a požadavky svých potomků
  - ✓ každý uzel udržuje FIFO frontu žádostí o vstup do KS
  - ✓ každý uzel přeposílá svému rodiči pouze jeden požadavek na získání příznaku, povolení vstupu do KS,
  - ✓ pokud uzel j sám požaduje získat příznak a jeho fronta není prázdná, pak se staví do své vlastní fronty
  - ✓ uzel j použije získaný příznak pro vstup do své kritické sekce je-li v okamžiku, kdy příznak obdržel, v čele své fronty a je novým kořenem stromu

DME, *Raymond*, předávání příznaku po stromu

- Algoritmus
  - ✓ uzel i posílá žádost o příznak svému rodiči, j
    - je-li FIFO fronta v j prázdná, j zapíše i do své fronty FIFO a pošle žádost svému rodiči, k
    - není-li FIFO fronta v j prázdná, j zapíše i do své fronty FIFO
  - ✓ uzel j získává příznak od svého rodiče, od uzlu k,
    - pokud je na začátku své fronty, vstupuje do KS
    - pokud není na začátku své fronty, přepošle příznak i na počátku fronty a i z FIFO fronty v j odstraní
    - pokud fronta v j po předání příznaku i není prázdná, musí j poslat poslat i žádost, aby příznak získal zpět
  - ✓ jestliže i požaduje příznak a jeho fronta není prázdná, umístí sebe do své fronty.
- Složitost algoritmu je $O(\log\ n)$, uváznutí a stárnutí nehrozí

DME, *Raymond*, předávání příznaku po stromu, příklad

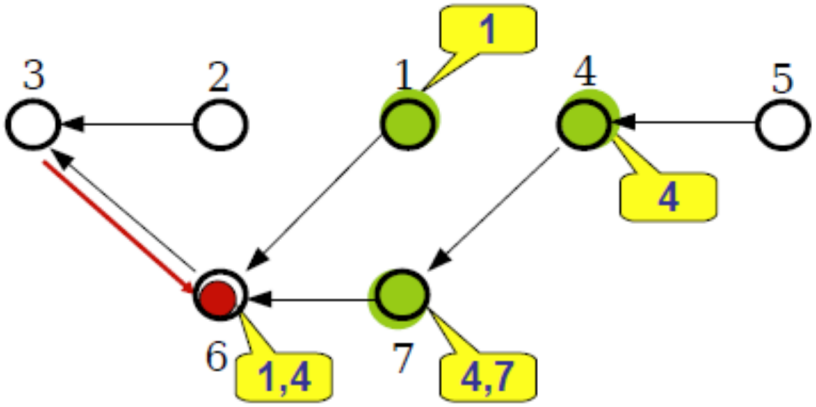
1, 4 a 7 chtějí vstoupit do KS, v tomto časovém sledu



3 drží token a dozvěděl se od 6,
že má token předat via 6 uzlu 1,
učiní tak, sám nežádá o vstup do KS

DME, *Raymond*, předávání příznaku po stromu, příklad

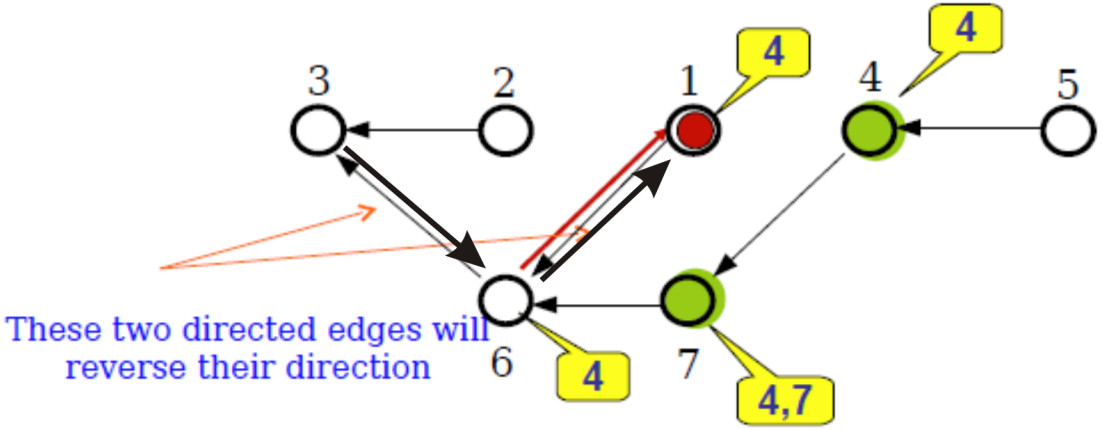
1, 4 a 7 chtějí vstoupit do KS, v tomto časovém sledu



3 pošle token 6,
6 ví, že ho má předat 1,
učiní tak, 1 se stává kořenem
a 6 pošle kořenu žádost 4

DME, *Raymond*, předávání příznaku po stromu, příklad

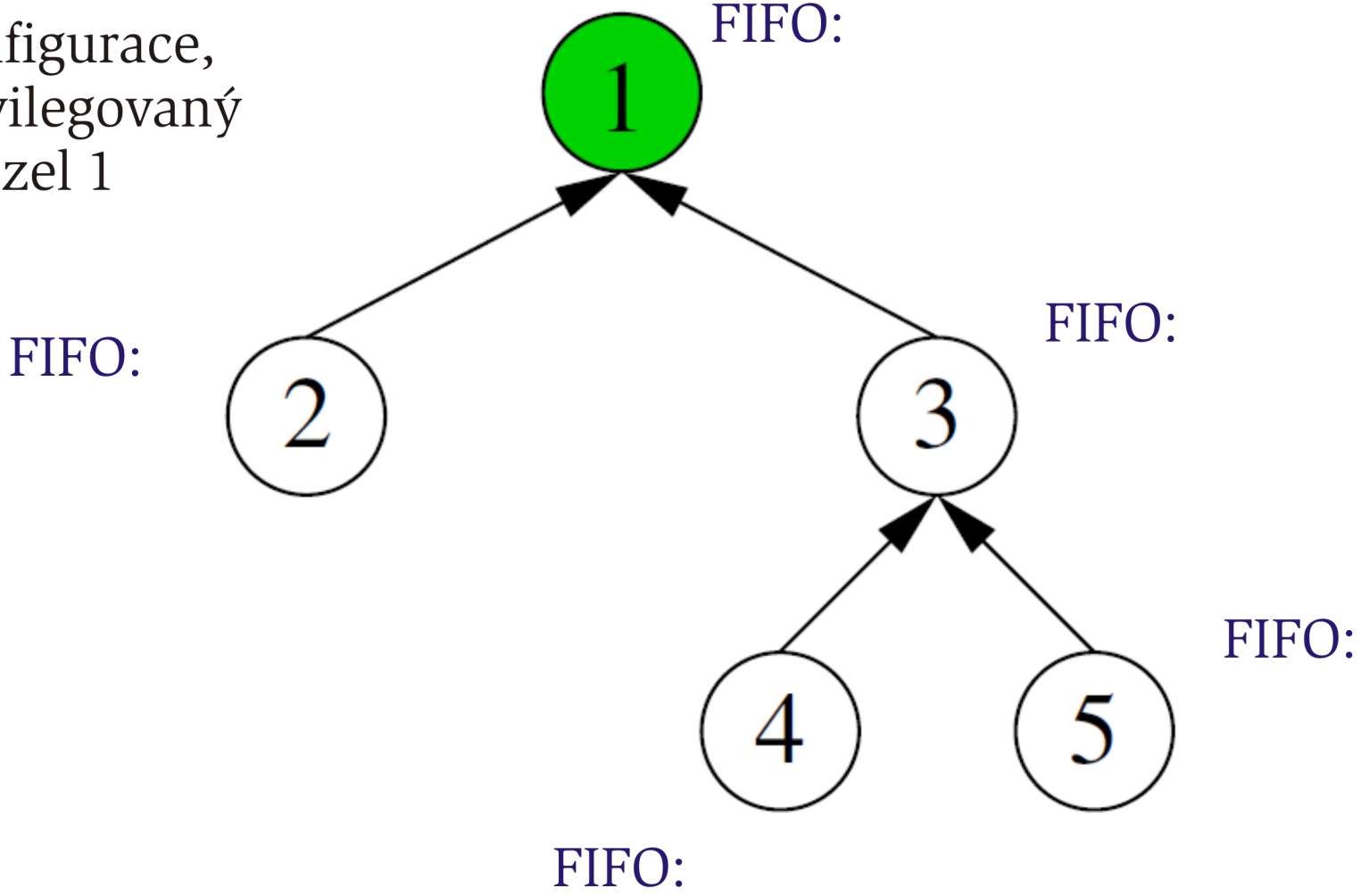
1, 4 a 7 chtějí vstoupit do KS, v tomto časovém sledu



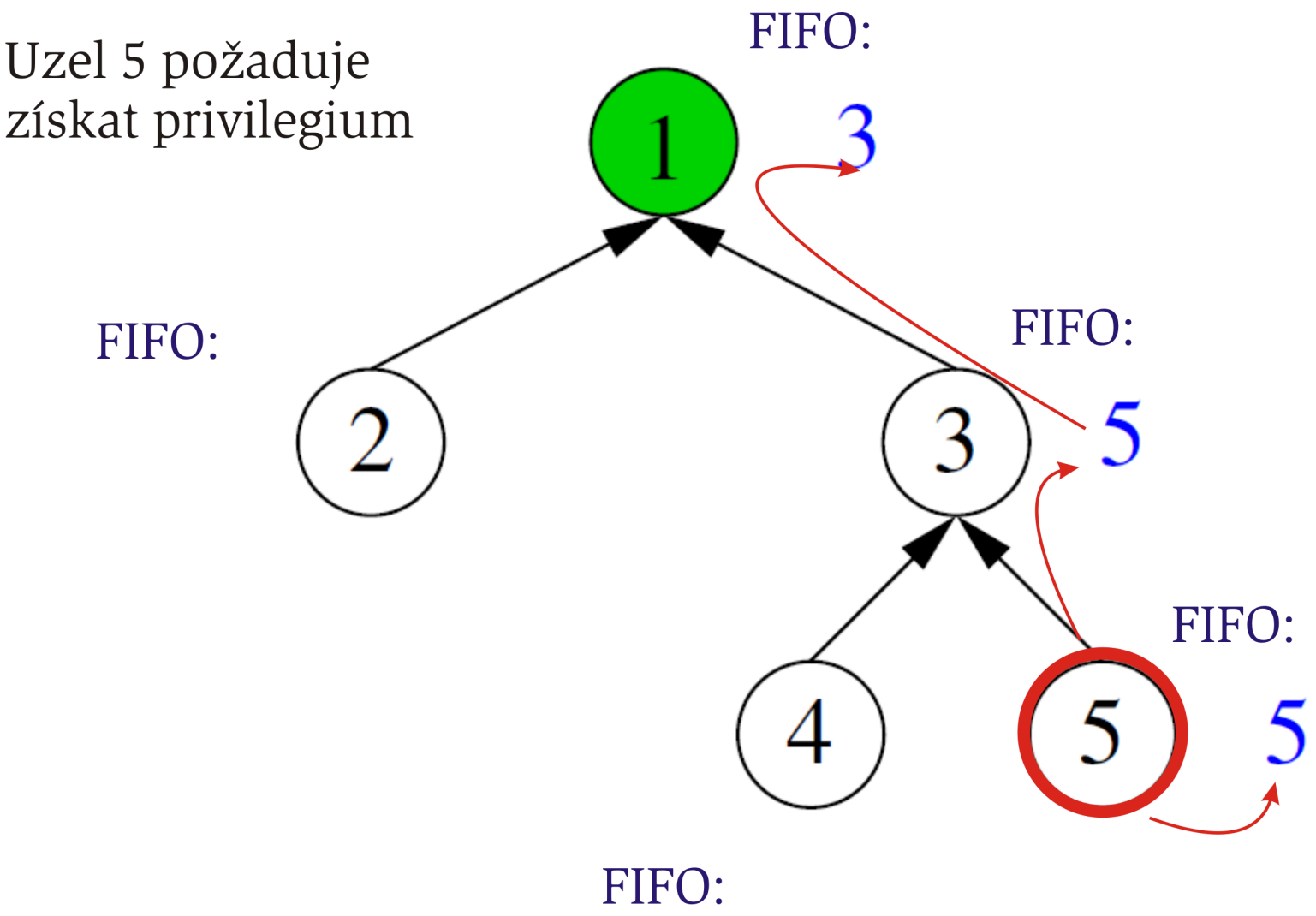
až 1 opustí KS,
předá token via 6 (a 7)
uzlu 4, ten se stane kořenem, ...

DME, *Raymond*, předávání příznaku po stromu, další příklad

Iniciální konfigurace, privilegovaný je uzel 1

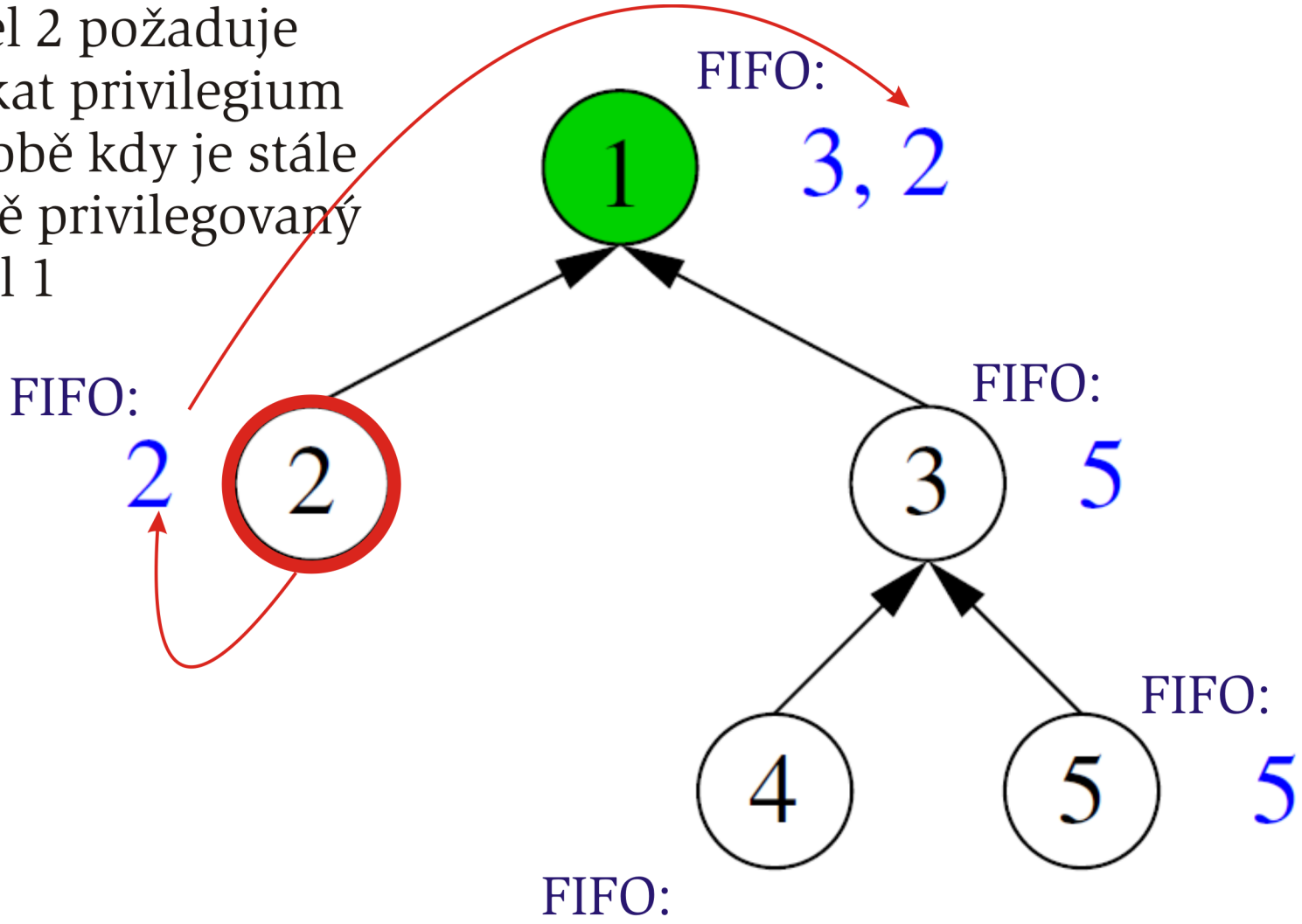


DME, *Raymond*, předávání příznaku po stromu, další příklad



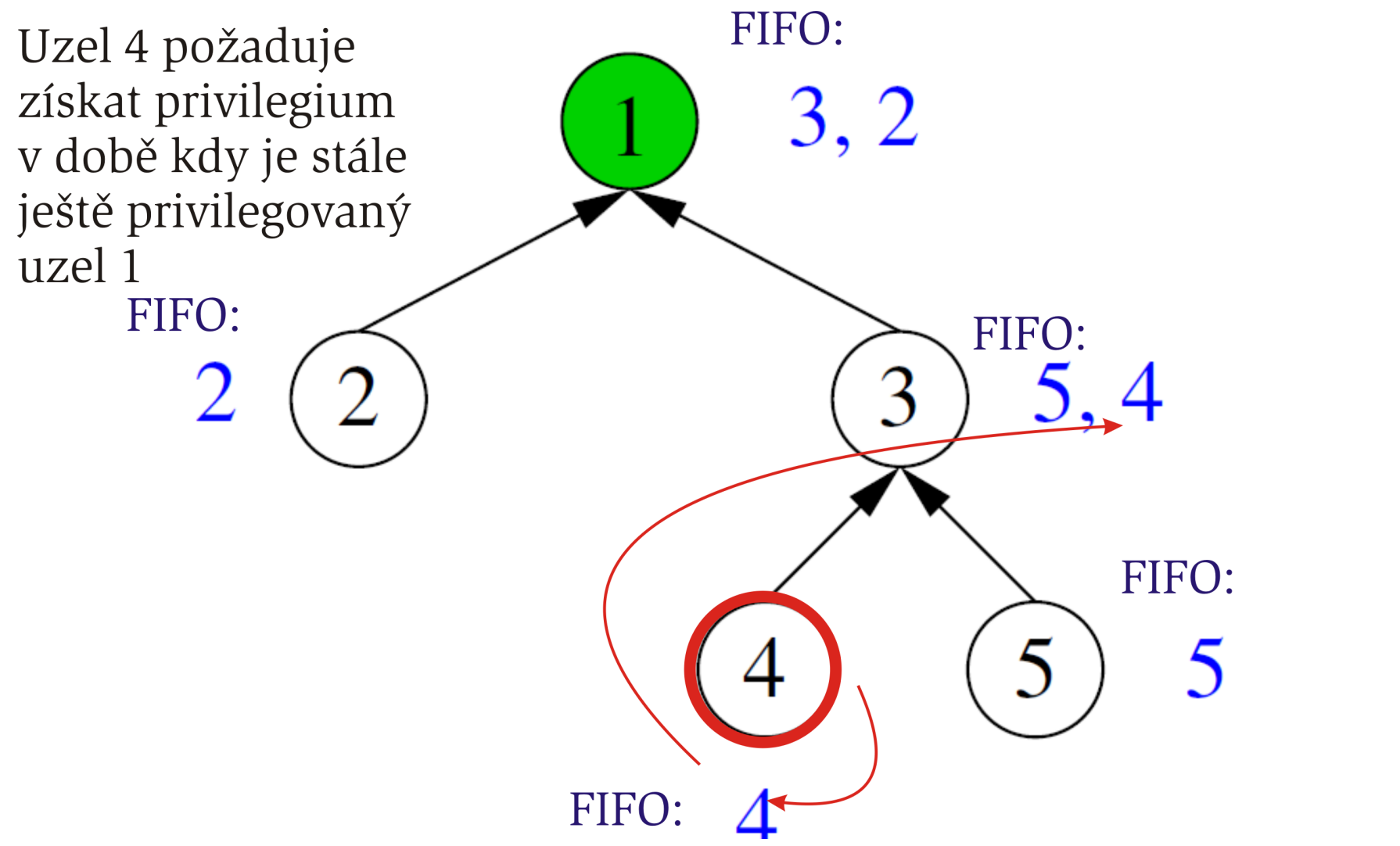
DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 2 požaduje získat privilegium v době kdy je stále ještě privilegovaný uzel 1



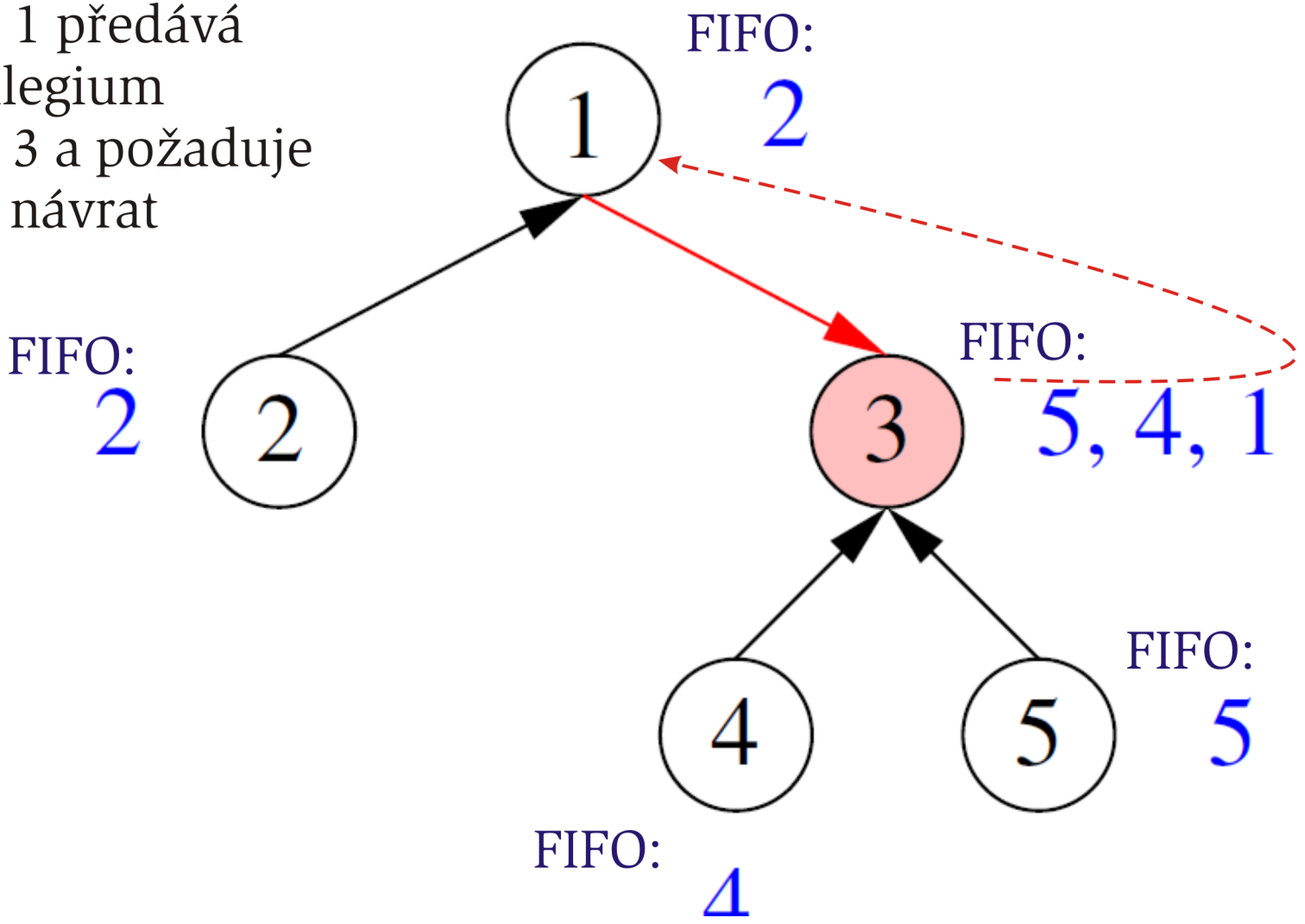
DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 4 požaduje získat privilegium v době kdy je stále ještě privilegovaný uzel 1



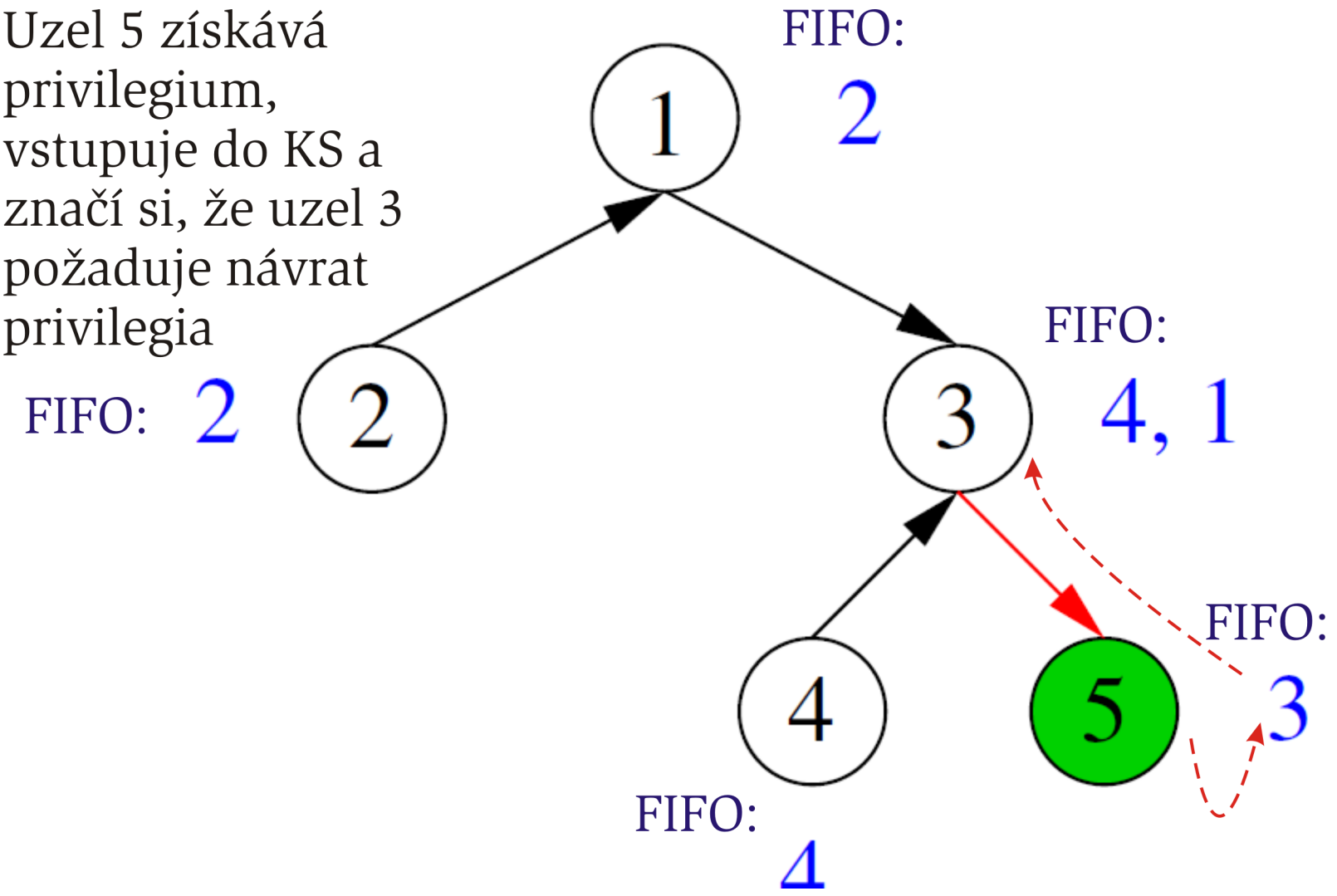
DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 1 předává privilegium uzlu 3 a požaduje jeho návrat



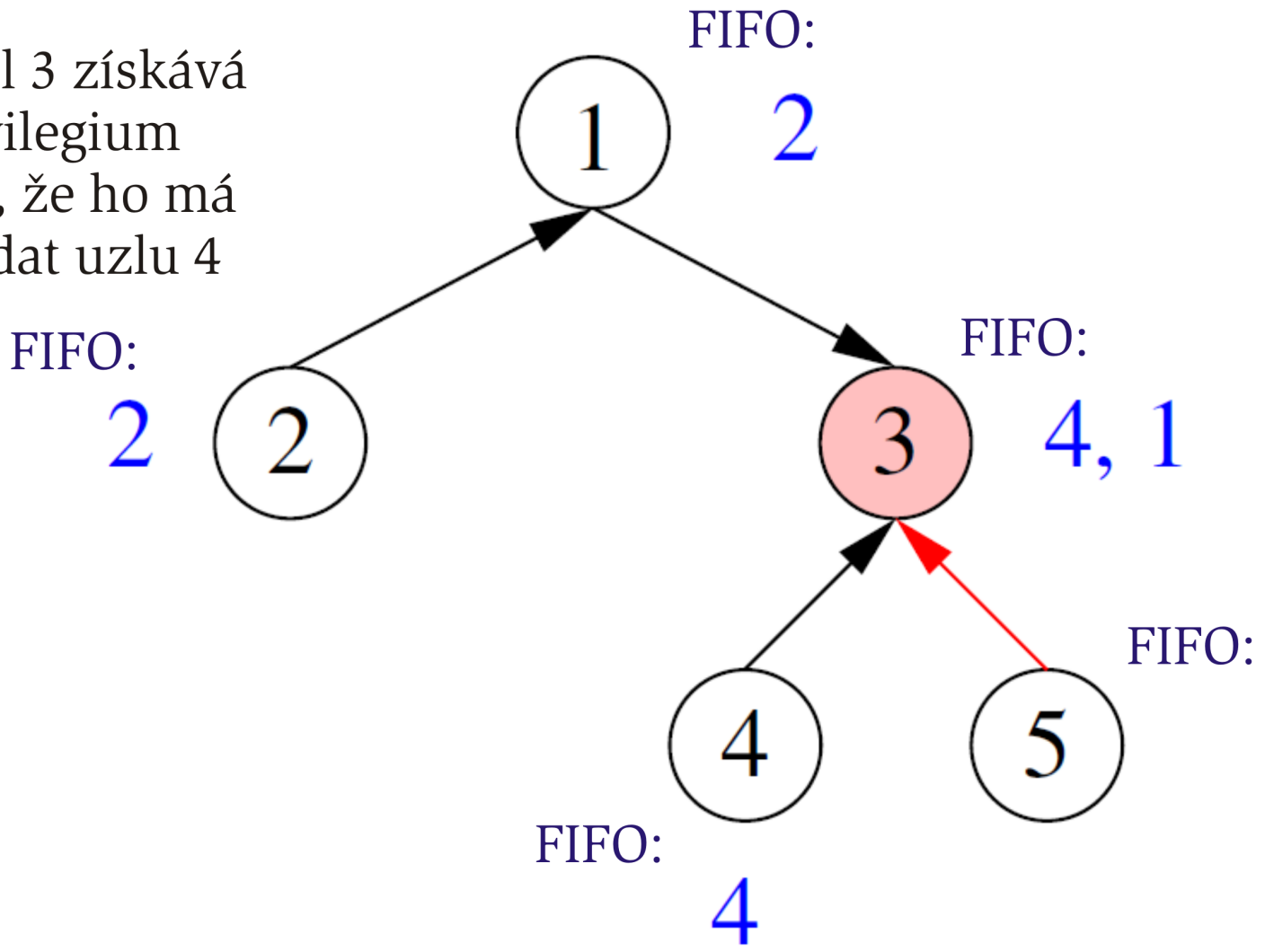
DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 5 získává privilegium, vstupuje do KS a značí si, že uzel 3 požaduje návrat privilegia



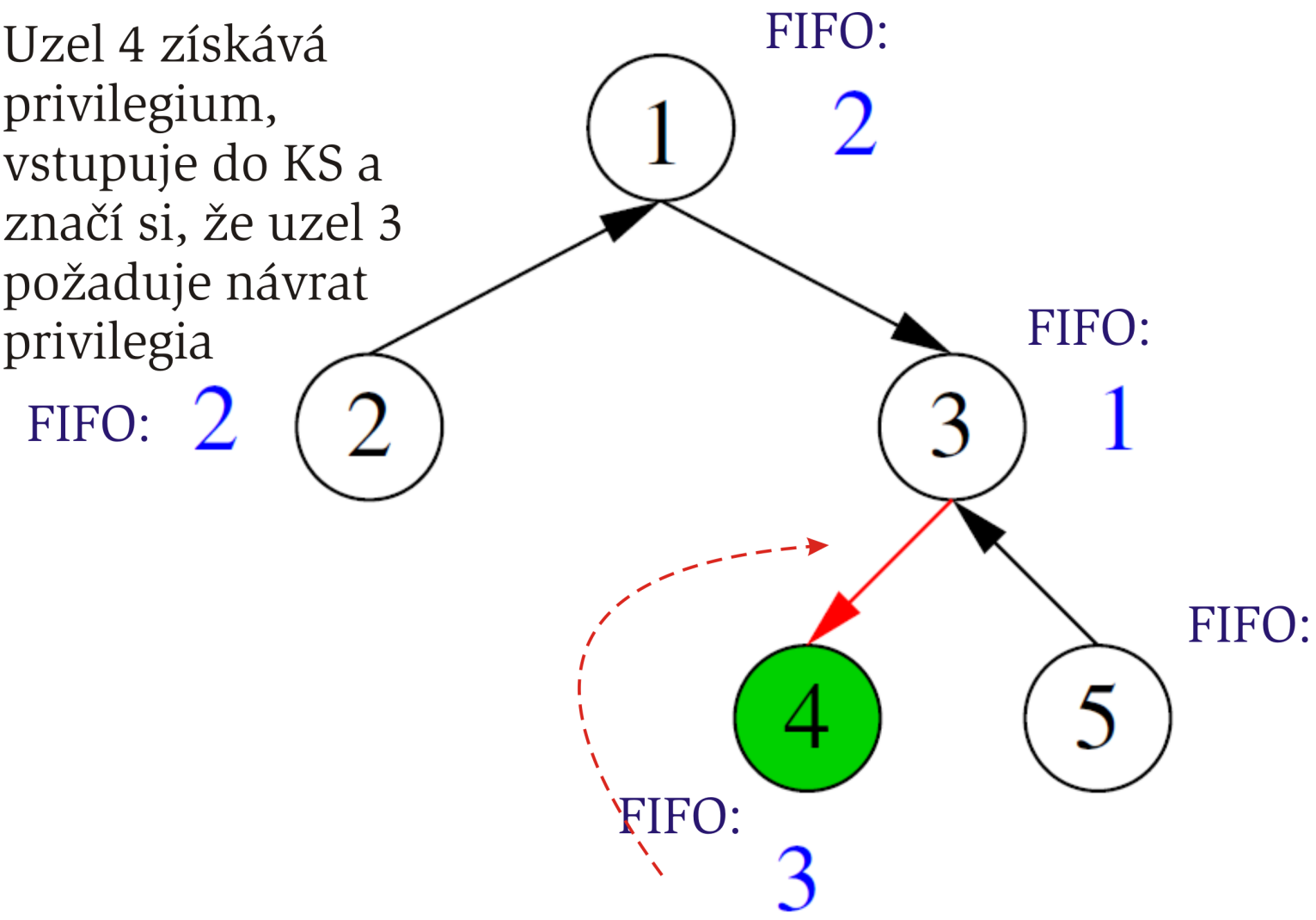
DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 3 získává privilegium a ví, že ho má předat uzlu 4



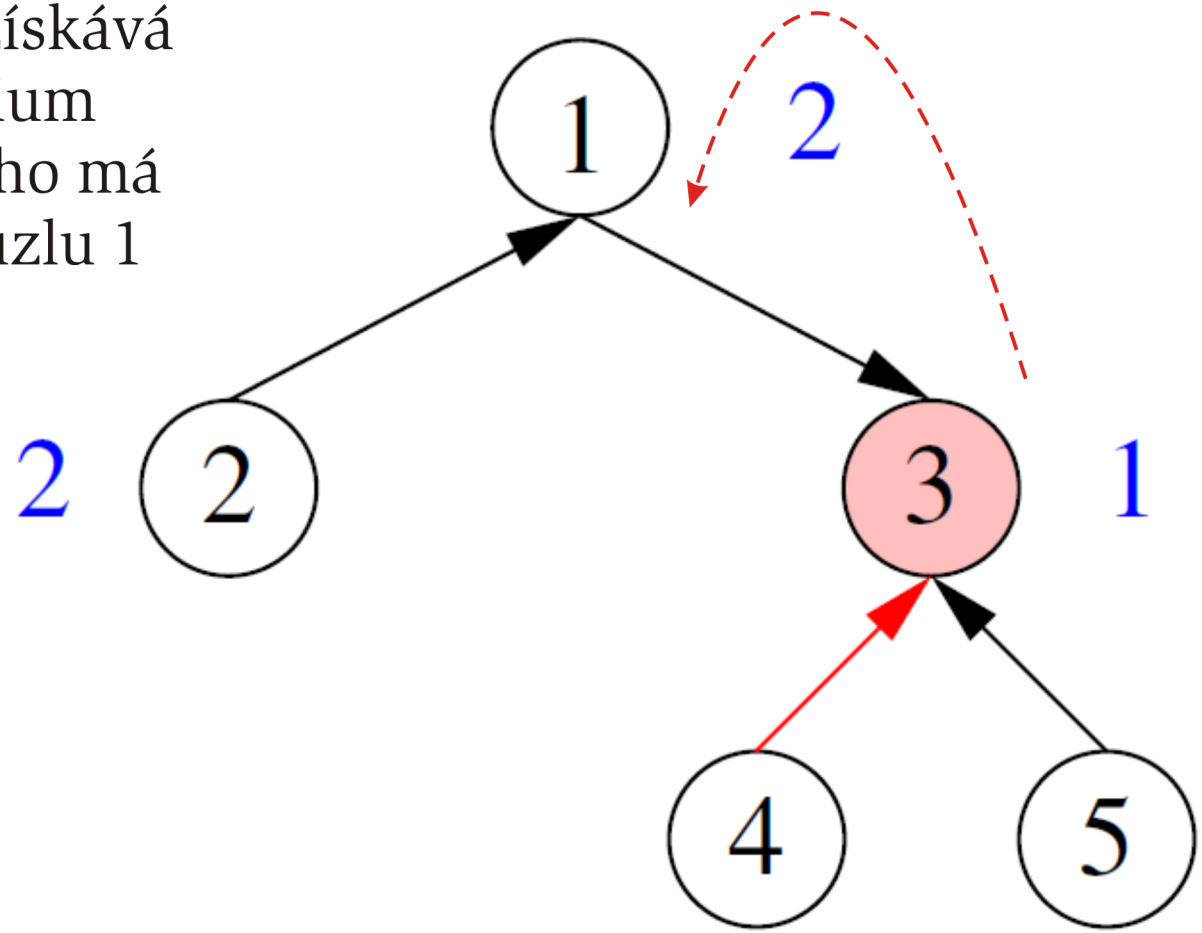
DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 4 získává privilegium, vstupuje do KS a značí si, že uzel 3 požaduje návrat privilegia



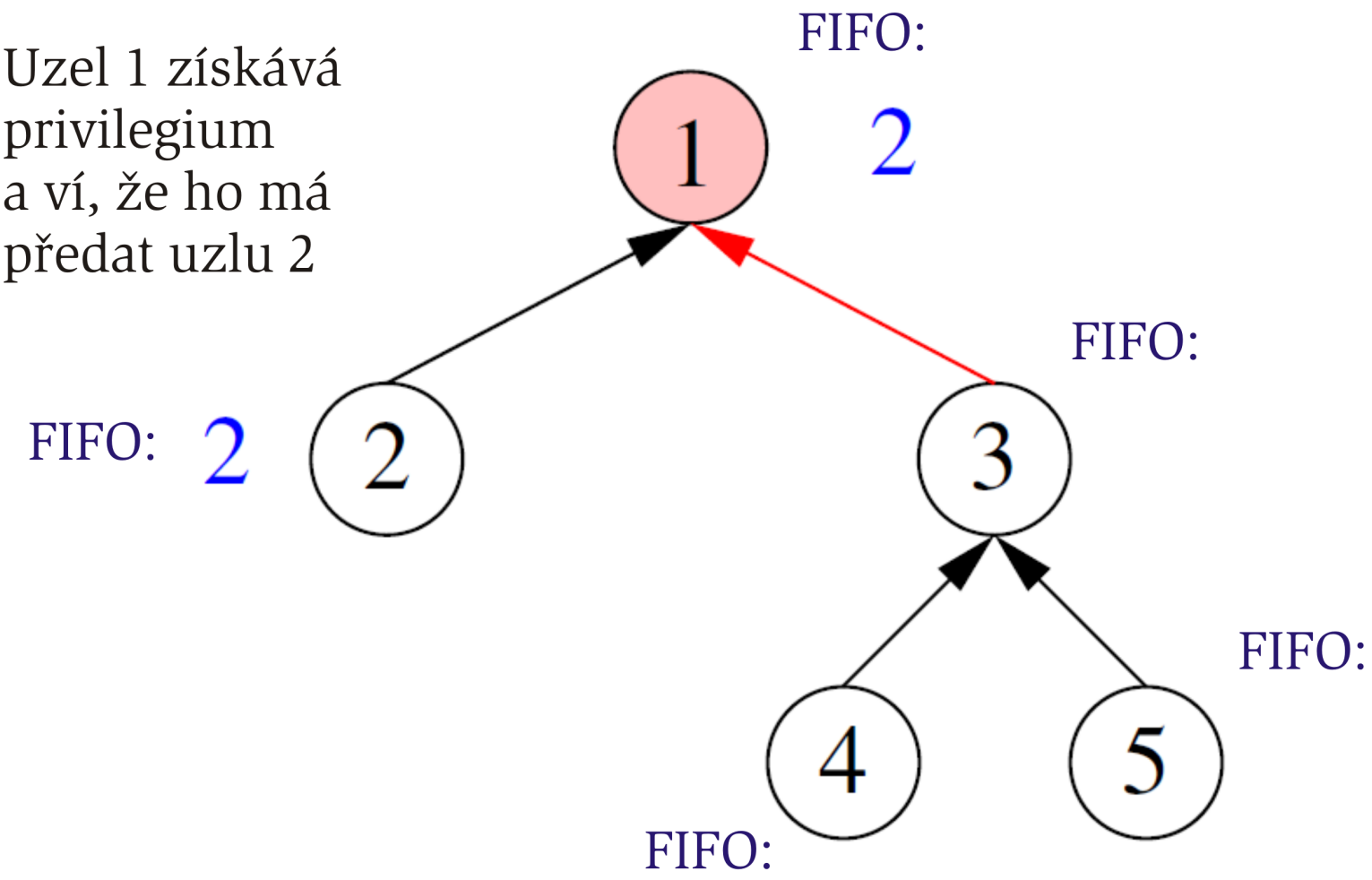
DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 3 získává privilegium a ví, že ho má předat uzlu 1



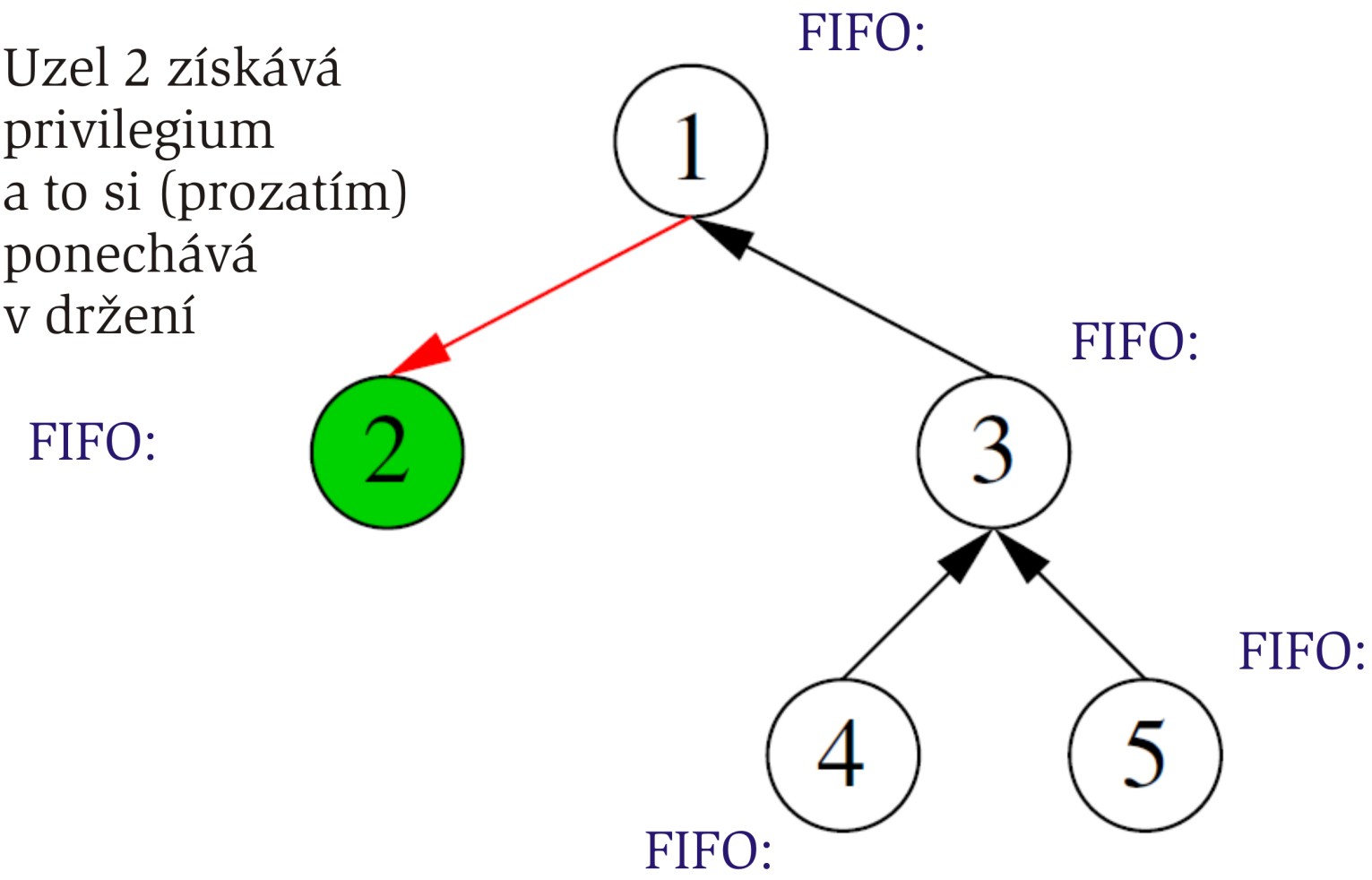
DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 1 získává privilegium a ví, že ho má předat uzlu 2



DME, *Raymond*, předávání příznaku po stromu, další příklad

Uzel 2 získává privilegium a to si (prozatím) ponechává v držení



DME, *Raymond*, předávání příznaku po stromu

- Vzájemná výlučnost je plněna trvale
  - ✓ v síti je vždy pouze jeden privilegovaný uzel, kořen stromu
- Nemůže dojít ke stárnutí
  - ✓ každý požadavek se nakonec přesune na počátek fronty
  - ✓ v řetězci požadavků se nikdy neobjevuje cyklus

DME – *Suzuki-Kasami*, předávání příznaku

- silně souvislá síť procesů (každý proces může komunikovat s každým procesem v síti)
- Proces smí vstoupit do KS jen když drží oprávnění ke vstupu do KS – příznak (*token*)
- Příznak předávaný mezi procesy je v jediném exempláři
- Pokud proces požadující vstoupit do KS nedrží příznak, rozešle všem procesům zprávu request požadující zaslání příznaku
  - ✓ velmi silné omezení – procesy se musí vzájemně znát
- Proces, který drží příznak a není v KS, zašle příznak procesu vybranému z procesů žádajících o příznak
- Procesy se vybírají tak, aby se zabránilo stárnutí, bázi výběru je cykličnost

DME – *Suzuki-Kasami*, předávání příznaku

- Díky asynchronnosti sítě proces i právě držící příznak může dostat žádost o jeho předání od procesu j až když žádost procesu j už byla vyřízena. Řešení:
  - ✓ Takové nadbytečné předání příznaku nenarušuje podmínku bezpečnosti, pouze zatěžuje komunikační systém zbytečnými zprávami
  - ✓ Procesy svoje žádosti o příznak pořadově číslují, request (P_i, SN_i)
  - ✓ Každý proces si udržuje N-prvkové pole R (*Requests*), ve kterém R_i udává pořadové číslo poslední přijaté žádosti od P_i, request (P_i, SN_i)
  - ✓ Platí $R_i = max(R_i, SN_i)$, pokud $R_i > SN_i$, pak se jedná o už zastaralou, neplatnou žádost

DME – *Suzuki-Kasami*, předávání příznaku

- Pokud se žádosti od více procesů u držitele příznaku kumulují, příznak se předává na bázi cyklického pořadí podle jejich identit:
  - ✓ Příznak obsahuje N-prvkové pole T ($Token$), ve kterém T_i udává pořadové číslo poslední žádosti o příznak z procesu P_i
  - ✓ Když proces P_i držící příznak opouští KS, nastaví $T_i = R_i$
  - ✓ Proces P_i držící příznak prohlíží pole T cyklicky, kdykoliv opouští KS nebo když drží příznak a dostal žádost o příznak

    cyklicky = počínaje indexem $i + 1 \ mod \ N$, pak po kroku 1 $mod \ N$,
  - ✓ Když nalezne $R_j = T_j + 1$, pak P_j žádá o příznak a P_i mu ho pošle

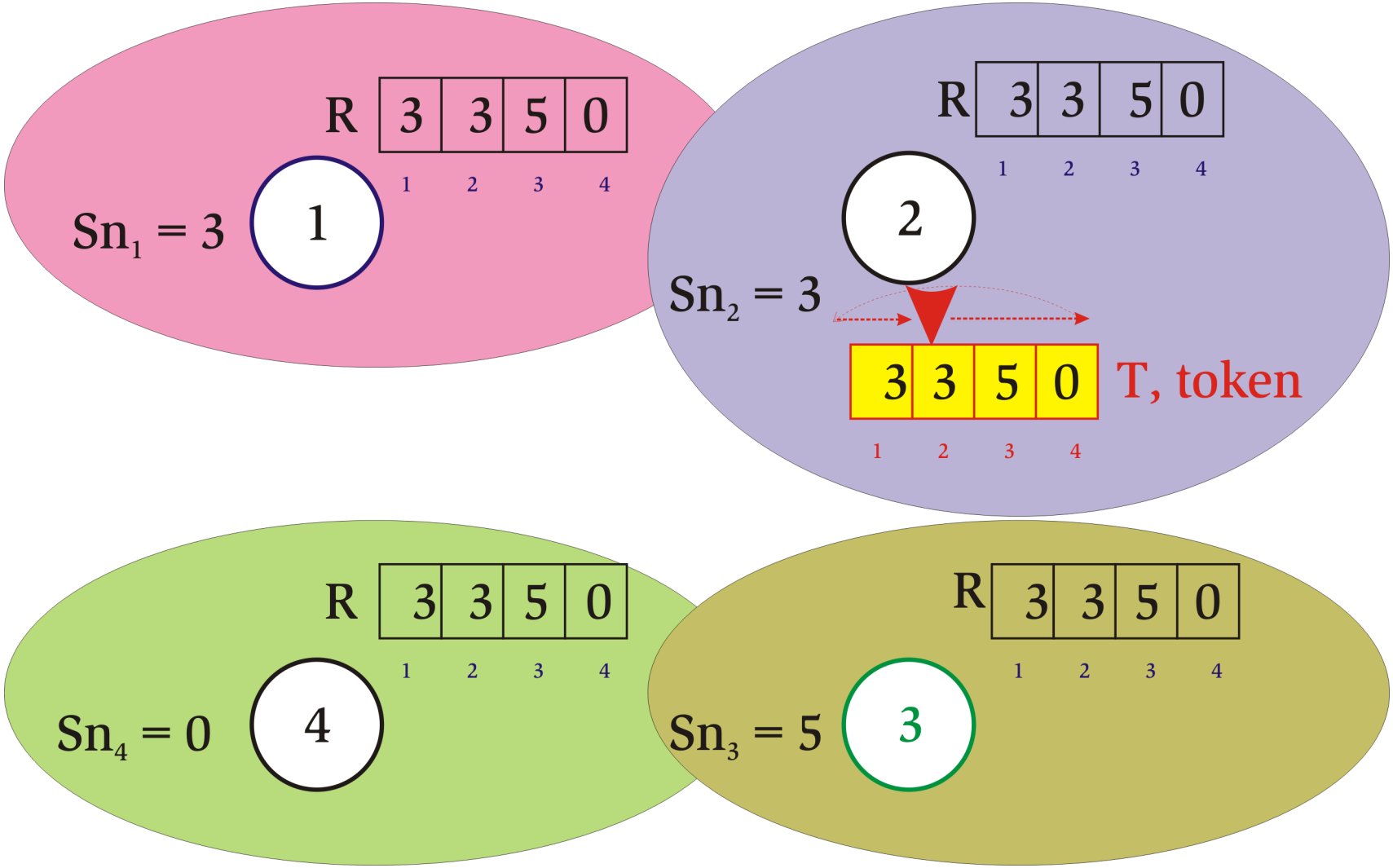
DME – *Suzuki-Kasami*, předávání příznaku

- Počátečně drží příznak libovolný proces
- Proces P_i CHCE vstoupit do KS
  - ✓ NEDRŽÍ příznak: inkrementuje R_i a všem ostatním procesům pošle request (P_i, SN_i), kde $SN_i = R_i$ a čeká až příznak získá
  - ✓ DRŽÍ příznak: vstoupí do KS a po opuštění KS provede $T_i = R_i$ a TEST, zjištění, kterému procesu má předat příznak
- Proces P_i obdrží žádost P_j o povolení vstoupit do KS
  - ✓ NEDRŽÍ příznak: v poli R žádost registruje, $R_j = max(R_j, SN_j)$
  - ✓ DRŽÍ příznak: provede TEST kterému procesu má příznak předat
- TEST
  - ✓ P_i prohledává příznak T v pořadí $i+1, i+2, \ldots, 1, 2, i-1$ a předává příznak prvnímu procesu P_k, pro který platí $R_k = T_k + 1$
  - ✓ i je pozice v T odpovídající procesu, který právě drží příznak

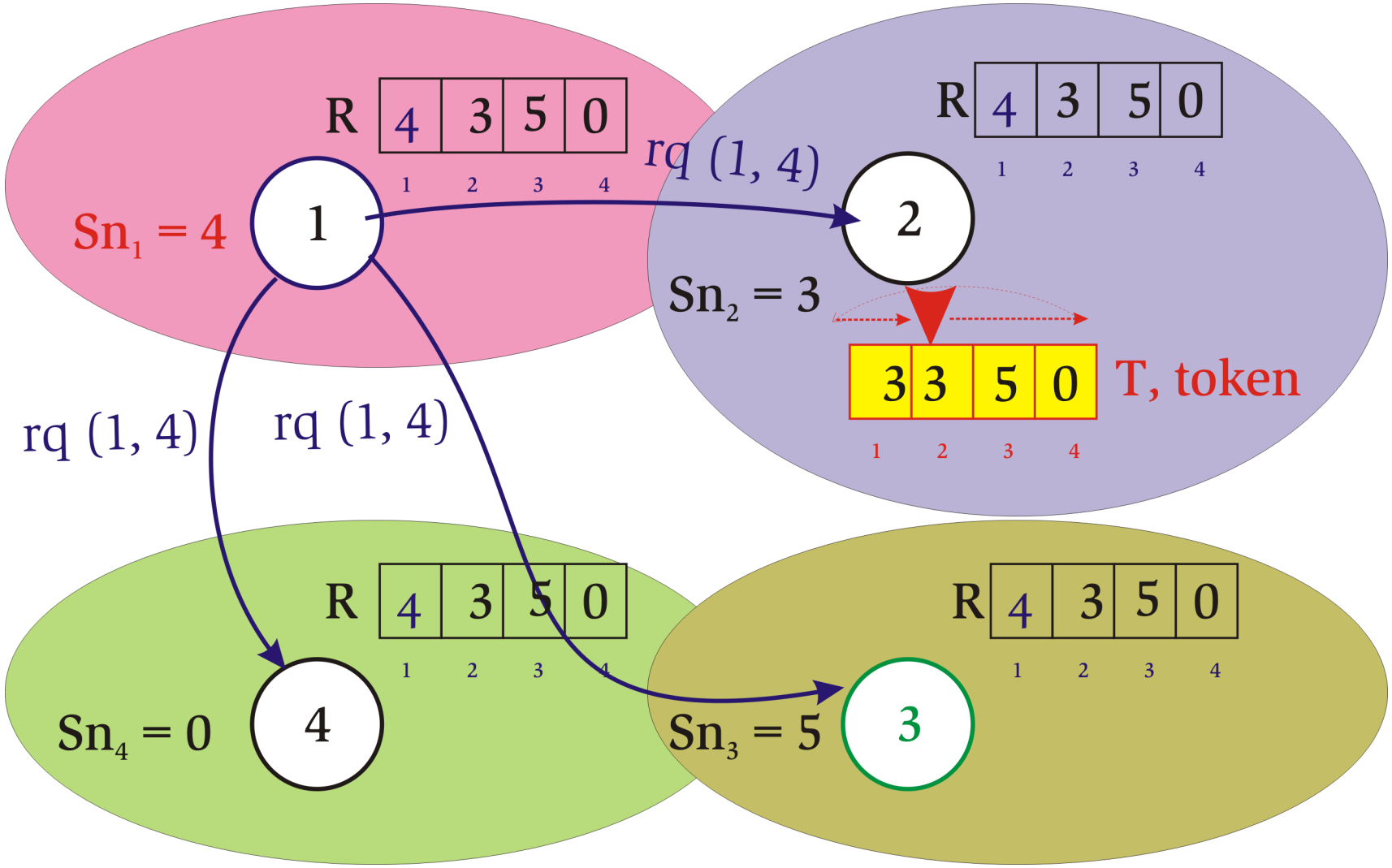
DME – *Suzuki-Kasami*, předávání příznaku

- Algoritmus je korektní: existuje jediný příznak
- Řešení je úplné, splňuje podmínku živost: požadující proces může předběhnout pouze $N - 1$ procesů díky cyklickému prohlížení příznaku
- Počet zpráv žádostí o vstup do KS, pokud proces nedrží příznak, N
  - ✓ $N - 1$ zpráv request (P_i, TS_i)
  - ✓ 1 zpráva zasílající příznak
- Počet zpráv na vstup do KS, pokud proces drží příznak, $= 0$

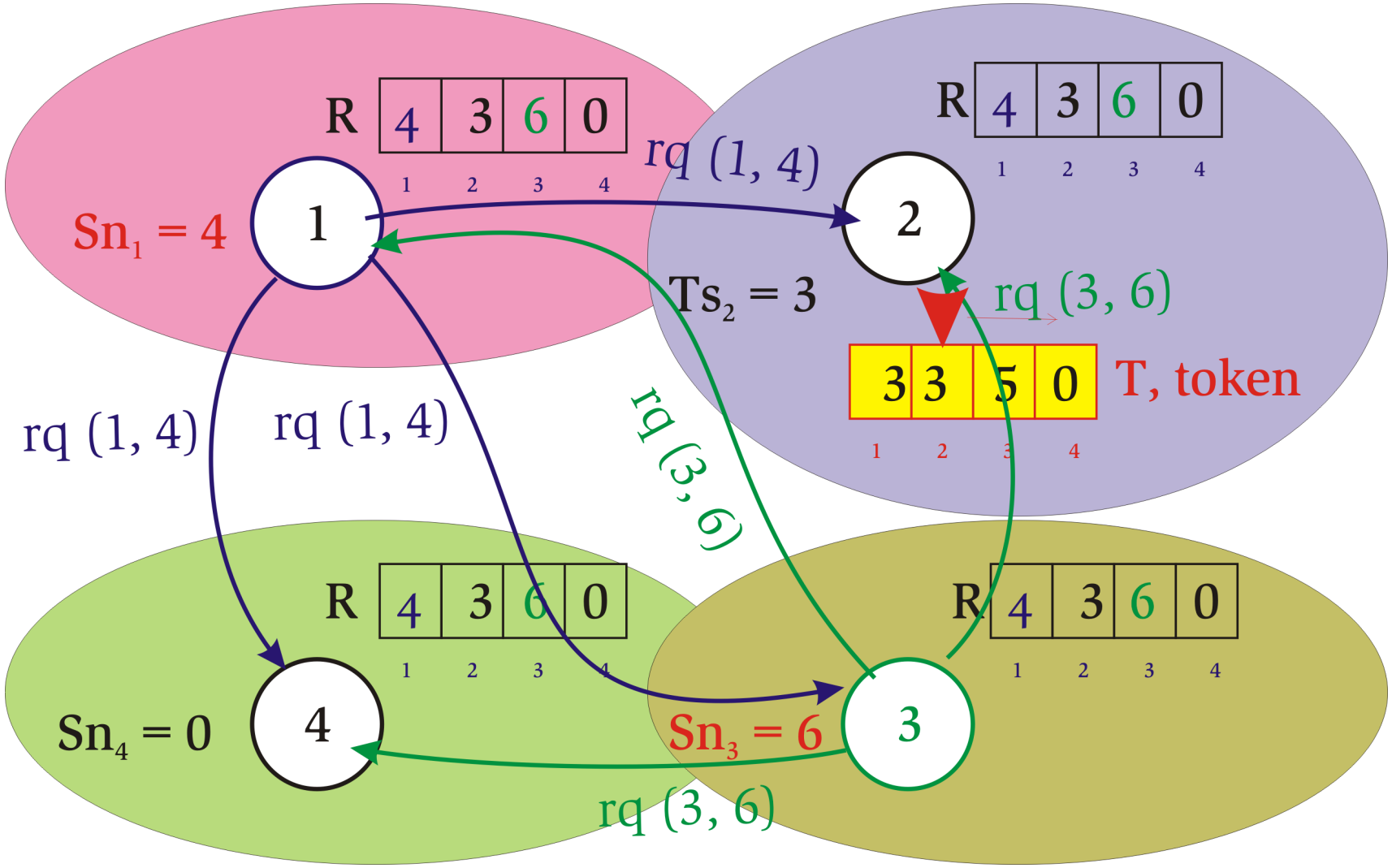
DME – *Suzuki-Kasami*, počáteční stav



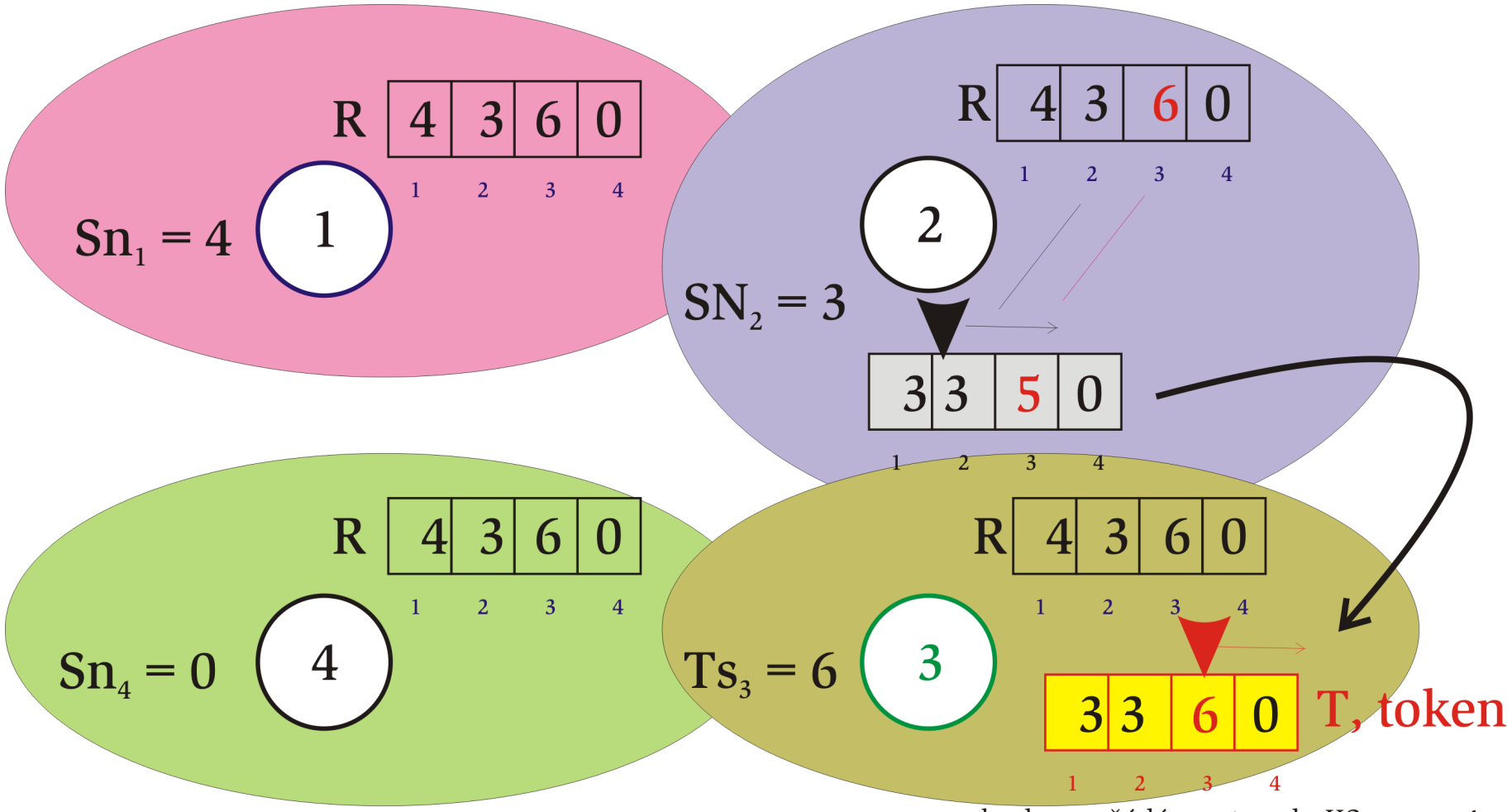
DME – *Suzuki-Kasami*, 1 žádá o token



DME – *Suzuki-Kasami*, 3 žádá o token

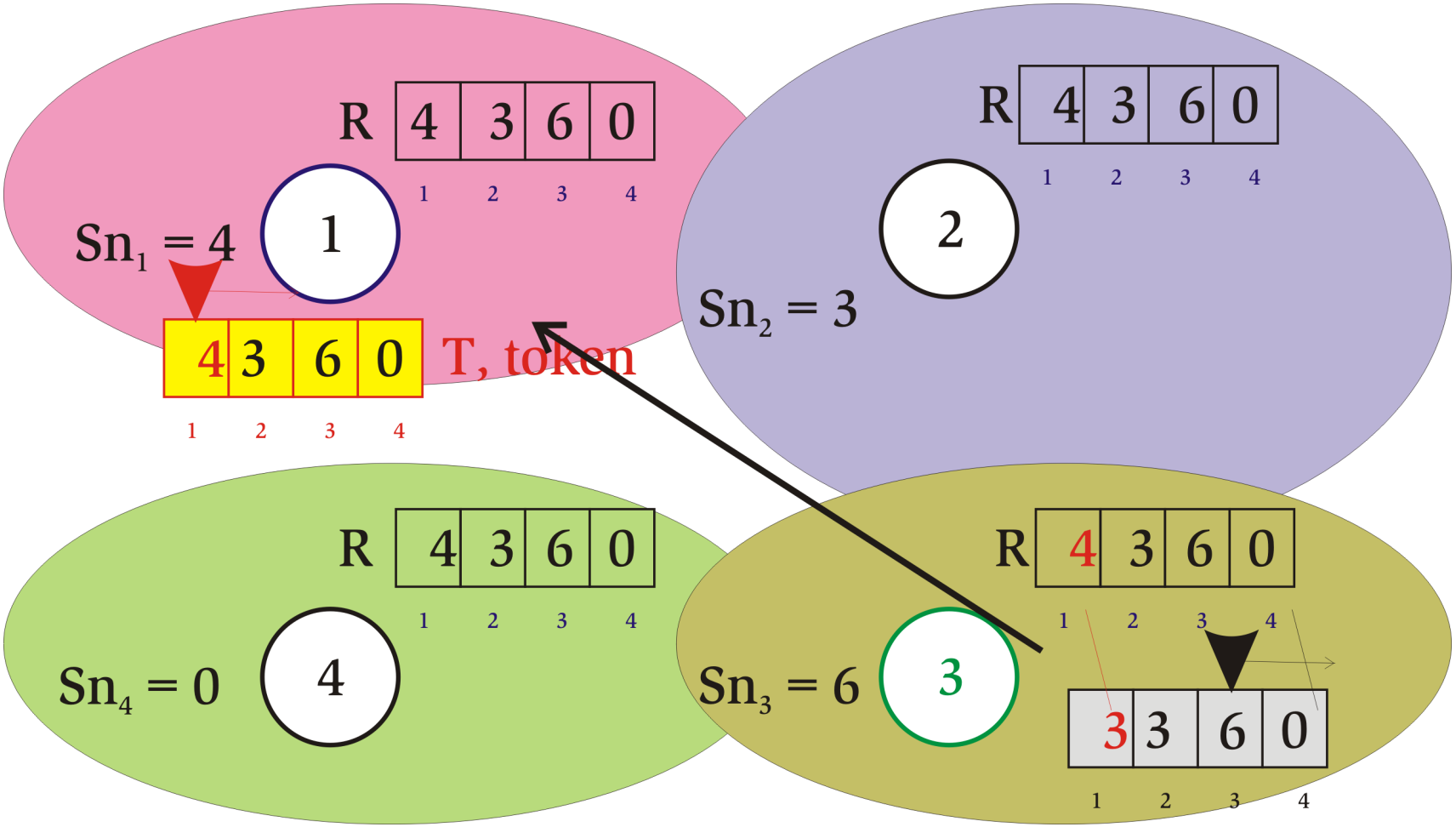


DME – *Suzuki-Kasami*,

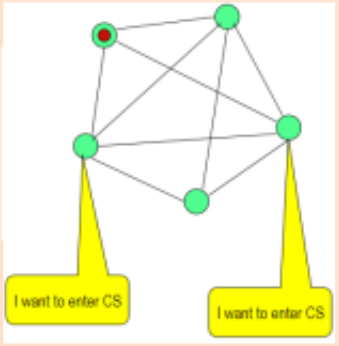


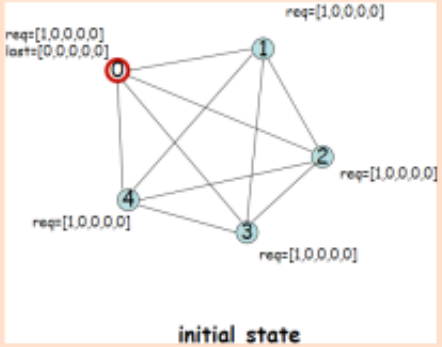
a pokud nepožádá o vstup do KS proces 4,
bude příště uspokojený proces 1

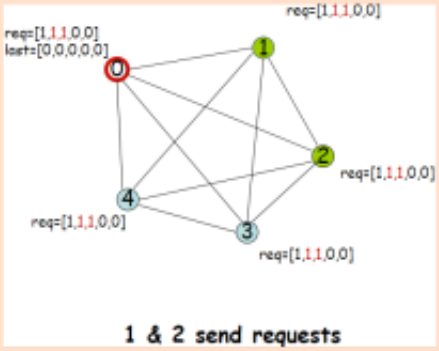
DME – *Suzuki-Kasami*

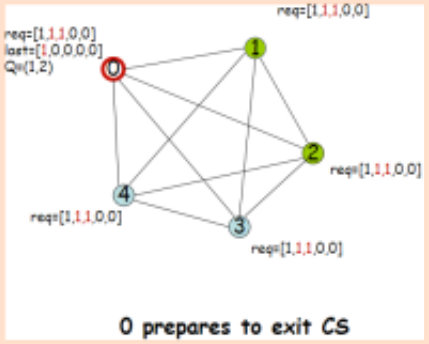


DME – *Suzuki-Kasami*, jiný příklad









silná souvislost

1

2

3

4

5

6

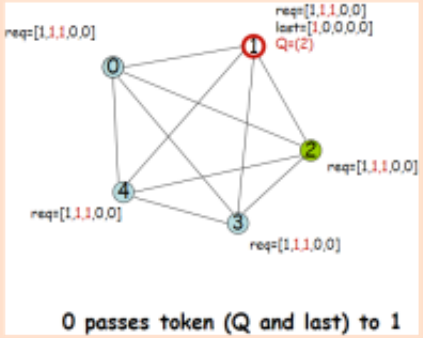
reg odpovídá **R**

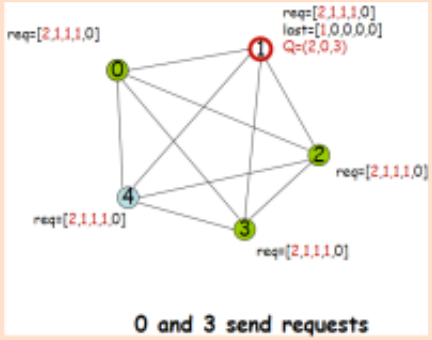
last odpovídá **T**

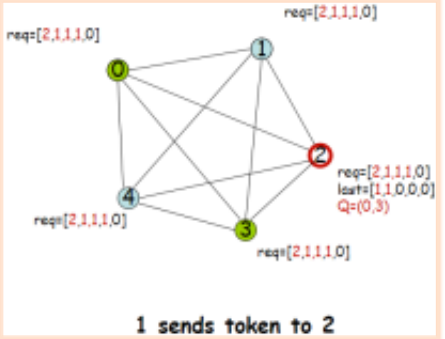
Q je fronta procesů pro které je **req**[k]=1+**last**[k]

Příznak je {**last, Q**}

Proces i, který opouští KS nastaví v **last[i]** své pořadové číslo a zkoriguje **Q** podle **last** a **req**







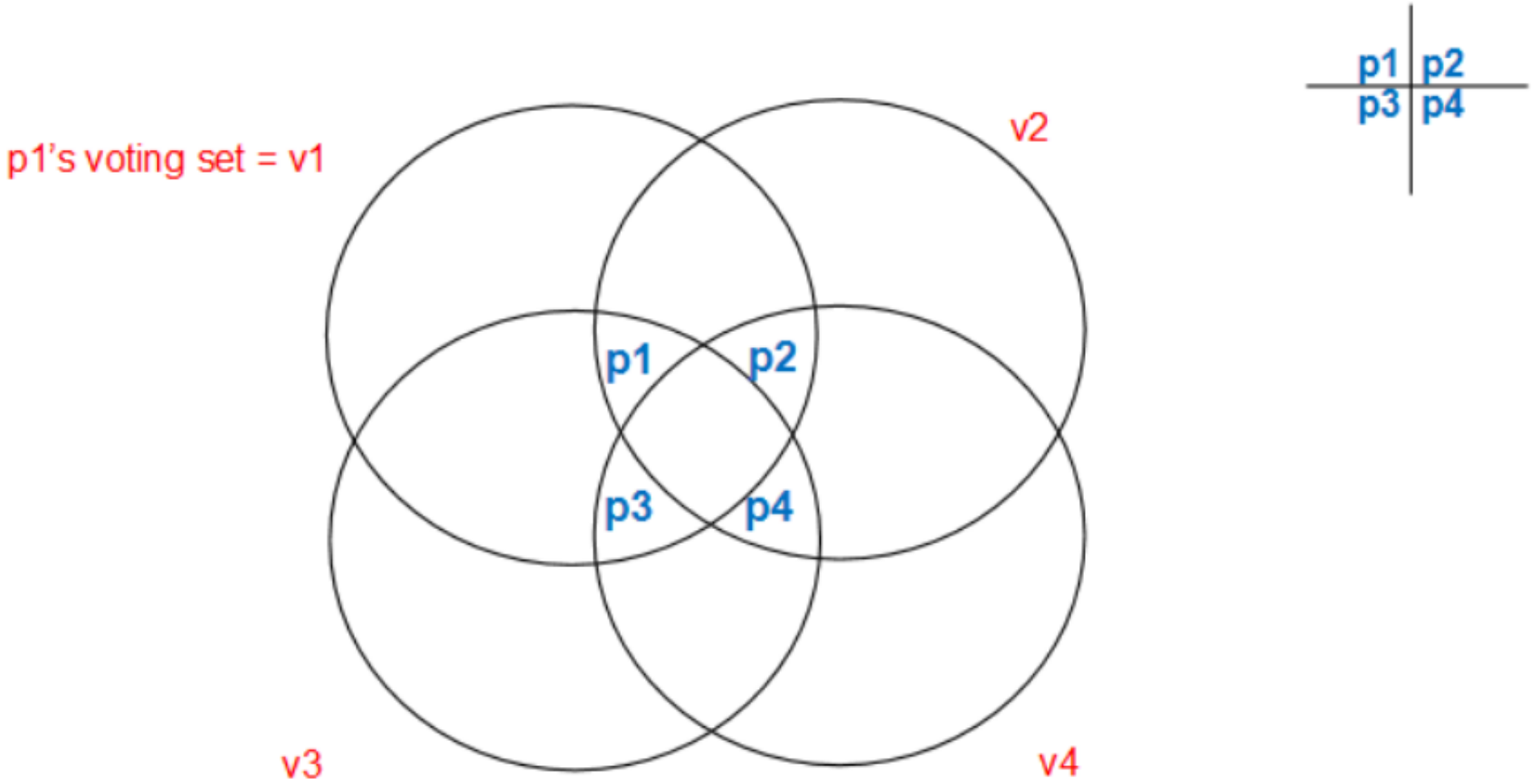
DME – jednoduchý kvórový algoritmus

- proces požadující vstup do kritické sekce pošle žádost o povolení vstupu všem ostatním procesům
- procesy tvoří kvórum, v kvóru má každý proces právě 1 hlas
- každý oslovený proces, který dosud svůj hlas nedal jinému procesu a nenachází se v kritické sekci, povolení udělí
- jakmile proces získá povolení od alespoň $\lceil (N+1)/2 \rceil$ (např. 3 z 4 nebo 5), může vstoupit do kritické sekce
- Po opuštění kritické sekce proces všechny ostatní procesy informuje o uvolnění kritické sekce – vrátí jim jejich hlas
- Nedostatek – hrozí uváznutí
  - ✓ např. každý ze tří souběžně žádajících procesů v množině šesti procesů získá po dvou hlasech

DME – kvórový algoritmus Maekawa

- *Mamoru Maekawa* (1985)
- organizace procesů pro optimalizaci komunikační složitosti
  - ✓ každému procesu p z množiny P, ve které kooperuje N procesů, je přiřazen volební okrsek V_p (*voting set*), kvórum, tvořený jistou podmnožinou procesů z množiny P procesů
  - ✓ p musí pro povolení vstupu do kritické sekce získat všechny hlasy ze svého okrsku, V_p
  - ✓ každý volitel má právě jeden hlas, po výstupu z kritické sekce proces hlasy svým volitelům vrací
  - ✓ kvóra se volí tak, aby byla splněna podmínka bezpečnosti, všechny mají alespoň jednoho společného volitele

DME – kvórový algoritmus Maekawa, princip kvór



DME – kvórový algoritmus Maekawa

- Podmínky pro kvóra, volební okrsky

  ✓ bezpečnost –

  každá dvojice volebních okrsků má alespoň 1 společného člena – proces, $\forall p, q : V_p \bigcap V_q \neq \emptyset$,

  každý proces má pouze jeden hlas $\Rightarrow$
  nemohou být současně zvoleny dva procesy

  ✓ spravedlnost –

  velikost volebních okrsků je konstantní

  $$\forall p, q : |V_p| = |V_q| = K,$$

  všechny procesy potřebují pro vstup do KS získat stejný počet hlasů,

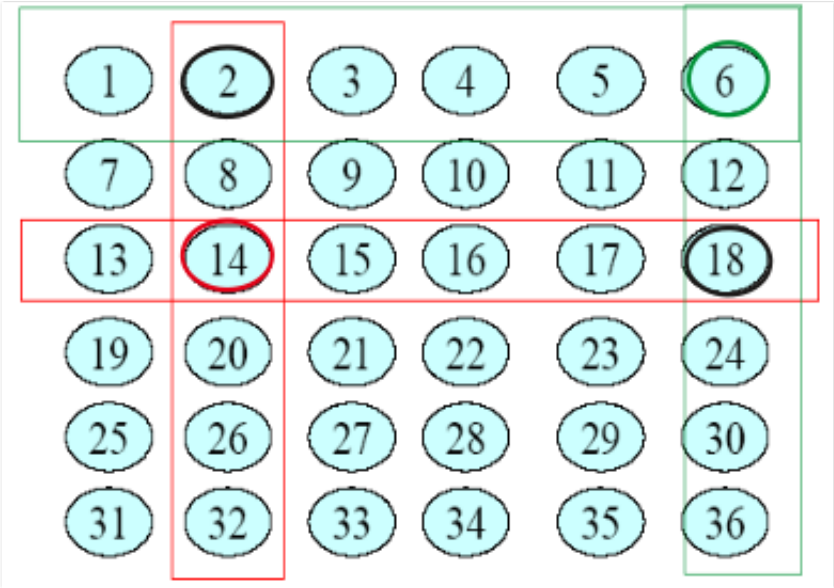
  každý proces má stejnou zodpovědnost,
  je obsažen ve stejném počtu M volebních okrsků,

  $$\forall p, q : |V_i : p \in V_i| = |V_j : q \in V_j| = M$$

DME – kvórový algoritmus Maekawa

- komunikační složitost odpovídá $O(|V_p|)$
  - ✓ cílem implementace je minimalizace velikosti volebních okrsků
- Konstrukce optimálních volebních okrsků
  - ✓ N, počet procesů, nechť platí $N = n^2$
  - ✓ $N = n^2$ je nepodstatné omezení, skutečný počet lze snadno doplnit služebně-formálními procesy pouze řádně povolujícími vstup
  - ✓ procesy označíme (i, j) pro $1 \leq i, j \leq n$,
    procesy se uspořádají do označovací matice $n \times n$
  - ✓ volební okrsek procesu p_{ij} tvoří procesy v i-tém řádku a j-tém sloupci označovací matice
  - ✓ velikost volebního okrsku $K = O(2\sqrt{N})$ je dobrý výsledek

DME – kvórový algoritmus Maekawa



Volební okrsek procesu 14 je vyznačený orámováním relevantního řádku a sloupce.

Vzájemné vyloučení -- jestliže procesu 14 povolil jehovolební okrsek vstup do kritické sekce, pak např. proces 6 nedostane hlas od procesů 2 a 18, ktere jsou v jeho okrsku

DME – kvórový algoritmus Maekawa, idea programu

On initialization

state := RELEASED;
voted := FALSE;

For p_i to enter the critical section

state := WANTED;
Multicast request to all processes in V_i;
Wait until (number of replies received = K);
state := HELD;

On receipt of a request from p_i at p_j

if (state = HELD or voted = TRUE)
then
queue request from p_i without replying;
else
send reply to p_i;
voted := TRUE;
end if

For p_i to exit the critical section

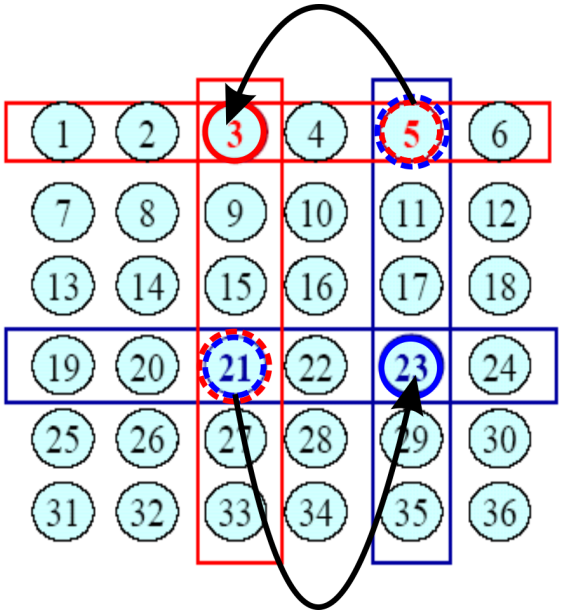
state := RELEASED;
Multicast release to all processes in V_i;

On receipt of a release from p_i at p_j

if (queue of requests is non-empty)
then
remove head of queue – from p_k, say;
send reply to p_k;
voted := TRUE;
else
voted := FALSE;
end if

DME – kvórový algoritmus Maekawa

- V dosud prezentované základní verzi Maekawovův algoritmus neošetřuje uváznutí, není splněná podmínka živosti



Proces 5, který je člen volebních okrsků procesů 3 a 23, dal hlas procesu 3

Proces 21, který je člen volebních okrsků procesů 3 a 23, dal hlas procesu 23

DME – kvórový algoritmus Maekawa

- Prevence uváznutí – musí se respektovat logický čas
  - ✓ procesy udržují nevyřízené požadavky v pořadí logického času
  - ✓ proces p přijme žádost od procesu r s TS_r
  - ✓ p má volný hlas – povolí vstup procesu r
  - ✓ p dal již hlas jinému procesu q s $TS_q < TS_r$ –
    novější požadavek procesu r si zařadí do své fronty požadavků
  - ✓ p dal již hlas jinému procesu q s $TS_q > TS_r$ –
    r je strší proces, p pošle mladšímu procesu q zprávu REJECT
    - pokud q je již v kritické sekci,
      tj. už dostal všechny hlasy ze svého kvóra,
      odpoví (vrátí hlas p) až po opuštění kritické sekce
    - pokud q ještě nezískal všechny hlasy ze svého kvóra,
      není tedy ještě v kritické sekci,
      vrátí hlas procesu p a ten ho předá procesu r

      proces p si ve své frontě požadavků obnoví žádost z q