Čas a stav v distribuovaném prostředí

PA 150 ◇ Principy operačních systémů

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

Verze : podzim 2020

Obsah přednášky

- Čas a jeho projevy v distribuovaném prostředí
- Řazení událostí v distribuovaném prostředí
- Globální stav v distribuovaném prostředí
- Distribuované algoritmy, DA
  - ✓ definice kroků prováděných procesy v distribuovaném prostředí vč. vysílaných zpráv
  - ✓ V PA 150 se nejedná o formální kurs distribuovaných algoritmů, jde o intuitivní seznámení s problematikou z hlediska potřebné podpory ze strany operačních systémů
  - ✓ důkazy správnosti a odhady složitosti probíraných distribuovaných algoritmů jsou prezentovány neformálním způsobem.

Počítač, hodiny, čas, trvání, řazení, . . .

- K čemu slouží hodiny (měření času), obecně ?
  - ✓ Stanovení času, ve kterém se událost vyskytla
  - ✓ Stanovení trvání události nebo intervalu mezi 2 událostmi
  - ✓ Stanovení pořadí (časové posloupnosti) série událostí, ve kterém se tyto vyskytly
- Kde všude při zpracování informací hraje čas důležitou roli ?
  - ✓ Konzistentnost časových razítek v e-komerci
  - ✓ Bezpečnostní algoritmy založené na časových razítkách (Kerberos)
  - ✓ Určování pořadí řešení souběžných transakcí
  - ✓ Detekce uplynutí časových limitů, detekce uváznutí a stárnutí
  - ✓ Řízení zámků souborů a zámků záznamů
  - ✓ Plánování událostí a transakcí v čase
  - ✓ Identikace posledních verzí souborů
  - ✓ . . . , . . . , . . . , desítky ?, stovky ? tisíce ? dalších důvodů

Časové poměry v počítači

Table 2.2 Example Time Scale of System Latencies

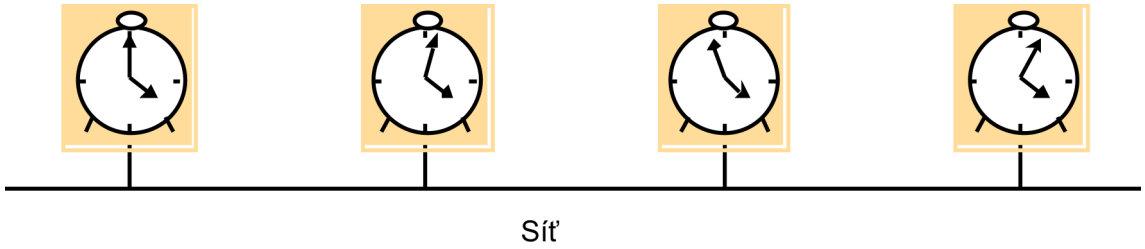
Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 µs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Čas v počítači, čas v síti (v DS)

- Čas v počítači
  - ✓ V každém počítači v GHz třídě je rychlost světla (elmag signálu) (299 792 458 m/s) omezujícím faktorem
  - ✓ V 3GHz počítači se deset tiků hodin odehraje v čase, za který se světlo rozšíří na vzdálenost 1 mm
  - ✓ Jestliže se v jednom tiku se provede 1 operace, musí se do příštího tiku získat data pro příští operaci
  - ✓ Registr CPU nemůže být dále od operační jednotky než 1/20 mm
- Čas v síti
  - ✓ typický time-out síťových operací je 255 s
  - ✓ to odpovídá 3 bilionům (3×10^{12}) operací soudobého CPU
  - ✓ pro srovnání 3 bilony sekund je cca 100 000 let

Problém řazení v čase, skluz, drift

- ✓ Pro řazení událostí (nejen v DS) do časové posloupnosti musí platit pro všechny lokality univerzální standardní čas
- ✓ Lokální hodiny uzlů v DS jsou ale téměř jistě ve vzájemném skluzu



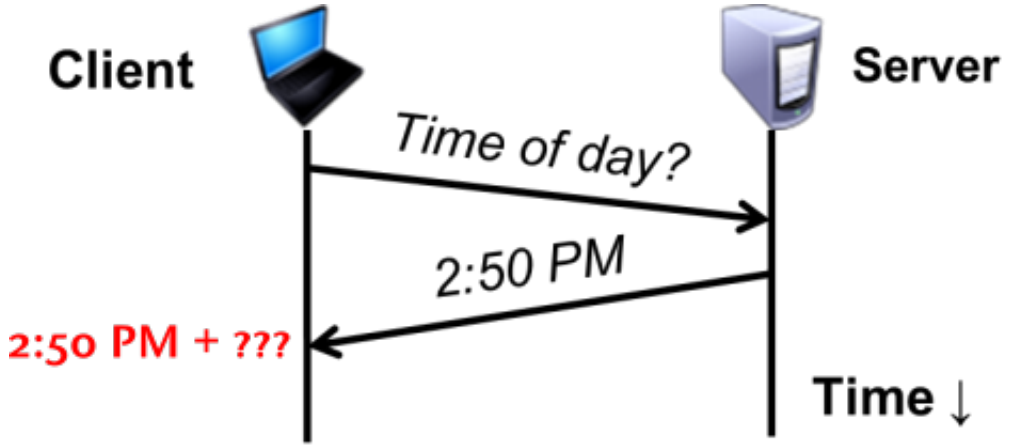
- ✓ skluz, diference je okamžitá hodnota, v čase se mění – drift
- ✓ důvodem odlišnosti rychlostí běhu jednotlivých hodin od ,,skutečného''času je citlivost krystalového oscilátoru radiaci, teplotu, vibrace, věk, . . .
- ✓ povolená diference běžných krystalem řízených hodin 10^{-6} s dává posun času o 1 s za 10^{6} s (11 a půl dne)
- ✓ Vysoce přesné hodiny s povolenou diferencí 10^{-7} až 10^{-8} problém univerzálně neřeší

IAT, *International Atomic Time*, UTC, *Universal Time, Coordinated*

- IAT – atomický čas, extrémně přesný zdroj času
  International Atomic Time – kmity v atomu Cesia 133,
  standardizovaná sekunda podle IAT =
  9 192 631 770 kmitů v atomu Cesia 133, drift 10^{-13}
- Sekundy, hodiny, roky . . . (astronomický čas) se odvozují od rotace země a rotace kolem slunce, tyto periody kolísají, astronomický a atomický čas mají tendenci se rozcházet
- Universal Time, Coordinated, UTC
  - ✓ Korigovaný atomický čas na astronomický čas (občas se přidává 1s)
  - ✓ Rozesílá se rádiově pozemními vysílači a satelity, komerční záležitost
  - ✓ Horší přesnost u koncových uživatelů času,
    od zdroje UTC se rozesílaný čas pozemními stanicemi může lišit až o 10 ms, satelitem (GPS) o cca 1 μs

Synchronizace lokálních hodin reálného času

- Nechť je t čas etalonových hodin (např. získaný UTC)
- Doba přenosu od etalonu do lokálních hodin, t_{trans}
- Čas synchronizovaných lokálních hodin, $t_{recv} = t + t_{trans}$
- t_{trans} bohužel neznáme, silně variuje



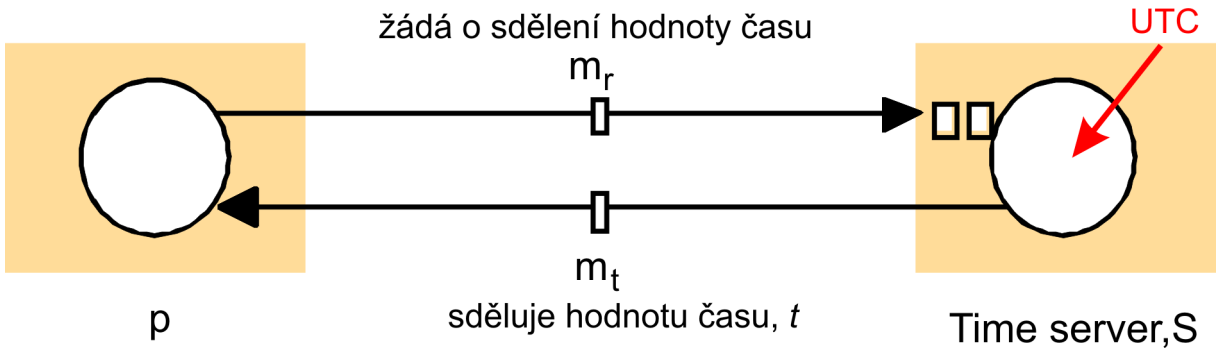
- Procesy potřebují, aby byl t_{recv} trvale udržovaný
  s přednastaveným stupněm přesnosti,
  tj. s definovanou tolerancí shody s t

Algoritmy synchronizace lokálních hodin s etalonem času

- Cristianův algoritmus
- Berkeley algoritmus, Berkeley Unix
- NTP (Network Time Protocol), Internet

Cristianův algoritmus synchronizace hodin

- DS obsahuje jistý počet důvěryhodných časových autorit, časových serverů
- Klient K se periodicky dotazuje časového serveru S na hodnotu času zprávou m_r
- Server např. získává signály ze zdroje UTC, na požádání sděluje hodnotu času podle svých hodin, t, zprávou m_t
- časový interval mezi vysláním požadavku a získáním hodnoty času je doba obrátky, $t_{round-trip}$



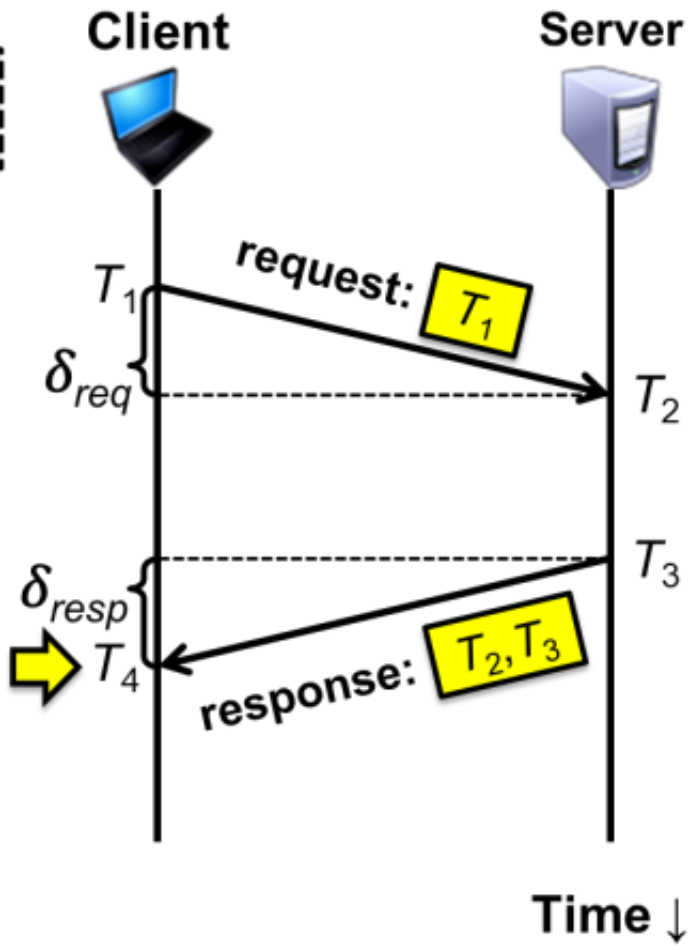
Cristianův algoritmus synchronizace hodin

Cíl: Klient nastavuje čas $T_3 + \delta_{resp}$

- Klient naměří ***round trip time*** δ = $\delta_{req} + \delta_{resp} = (T_4 - T_1) - (T_3 - T_2)$
- **Klient zná δ, nikoli δ_{resp}**

Předpokládá se : $\delta_{req} \approx \delta_{resp}$

Klient nastavuje čas $T_3 + \frac{1}{2}\delta$



Cristianův algoritmus synchronizace hodin

- pokud je $t_{round-trip} \leq \Delta$,
  pak dobrá aproximace času u žadatele je $t_{recv} = t + \frac{\Delta}{2}$
- pokud předpoklad neplatí, aproximace není věrohodná
- jde o pravděpodobnostní algoritmus:
  - $t_{round-trip}$ musí být dostatečně krátký,
    čím je menší Δ, tím je odhad přesnější
  - čím větší přesnost se požaduje,
    tím s menší pravděpodobností se dosahuje,
    protože často neplatí předpoklad dobré aproximace
- výpadek serveru –
  krach synchronizace, důsledek centralismu

Příklad

- Klient se synchronizuje s časovým serverem zasíláním dotazů, dosažené doby obrátek zpráv a získané informace o čase jsou:

Round-trip (ms)	*Time (hr:min:sec)*
22	*10:54:23.674*
25	*10:54:25.450*
20	*10:54:28.342*

- Na jaký čas by klient měl nastavit svoje hodiny ?
  - ✓ minimální naměřená doba obrátky je 20 ms = 0,02 s
  - ✓ klient by měl tudíž zvolit čas získaný s dobou obrátky 20 ms, tudíž bude 10:54:28.342 + 0.02/2 = 10:54:28.352 s přesností ± 10 ms

Příklad, pokrač.

- S jakou přesností je tento odhad správný je-li známo, že minimální čas mezi odesláním a příjmem zprávy v daném systému je 8 ms?
  - ✓ Min, minimální doba přenosu zprávy = 8 ms
  - ✓ Když klient poslal dotaz v čase t, server mohl odpovědět nejdříve za $t + Min$, a nejpozději za $t + t_{round-trip} - Min$, tj. udaný čas je z rozpětí $t_{round-trip} - 2 \times Min$, takže přesnost je $\pm(t_{round-trip}/2 - Min)$
  - ✓ Pokud se neznala doba Min, přesnost byla ± 10 ms
  - ✓ Když platí $Min = 8$ ms, bude přesnost za stejných podmínek ± 2 ms $(20/2 - 8)$
  - ✓ Pokud se při $Min = 8$ ms požaduje přesnost nejvýše ± 1 ms musí být nejvýše $t_{round-trip} = 18$ ms $(18/2 - 8)$

Příklad, pokrač.

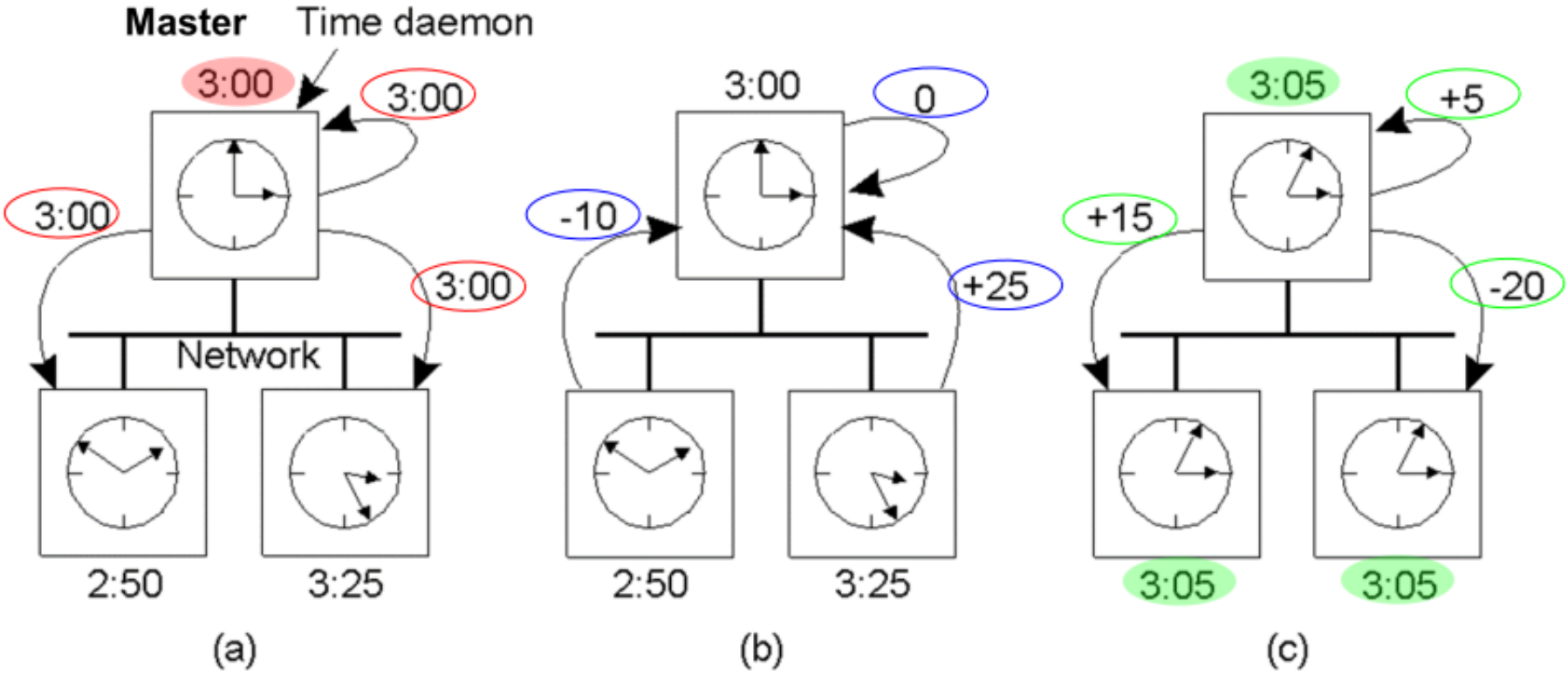
- Čím větší přesnost se požaduje,
  tím s menší pravděpodobností se dosáhne
  - ✓ tj. ideálně by mělo platit $t_{round-trip} = 2 \times Min$
    což je ve běžné síti málo pravděpodobné

- Cristianův algoritmus je vhodný pro LANy
  s dobře odhadnutelnou minimální dobou přenosu zpráv

Berkeley algoritmus synchronizace hodin

- Jeden z počítačů DS je master, ostatní jsou slave:
  - ✓ Master uzel periodicky vyzývá každý slave uzel k zaslání diference jeho času a změří příslušný $t_{round-trip}$ a
  - ✓ ze zjištěných hodnot eliminuje (min, max) hodnoty, zbytek průměruje
  - ✓ každému slave uzlu zašle interval, o který se čas slave uzlu liší od vypočteného průměru – udává de facto nový přesný čas
  - ✓ Slave uzly si zkorigují své lokální hodiny na nový přesný čas
- Výpadek master uzlu lze ošetřit distribuovanou volbou nového master uzlu
  - ✓ konkrétní algoritmus volby bude vysvětlený později
- Konkrétní příklad (reálné měření)
  - ✓ LAN 15 uzlů, max $t_{round-trip}$ 10 ms, interval korekcí 25 ms, drift hodin slave uzlů byl menší než 2×10^{-5} ms

Berkeley algoritmus synchronizace hodin

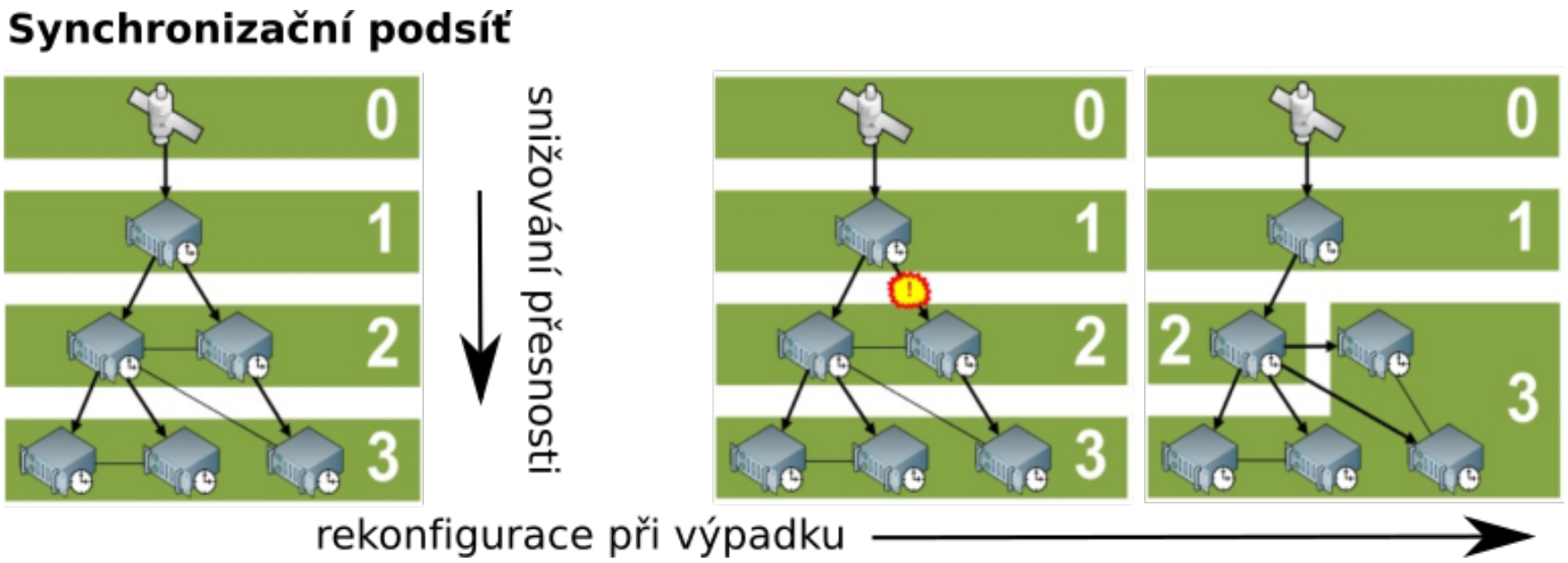


Algoritmus NTP, Network Time Protocol

- Cristianův alg., Berkeley alg. jsou vhodné pro intranety
- NTP, Network Time Protocol používá Internet, veřejná WAN
  - ✓ Služba poskytující klientům v Internetu možnost se přesně synchronizovat s UTC (přesně = v intervalu řádově ms)
  - ✓ PVhodné po zajištění spolehlivých služeb, které mohou přežít dlouhé ztráty konektivity – rekonfigurací po uplynutí time-outu
  - ✓ Pro zajištění ochrany proti interferenci se zlomyslnou nebo náhodně chybující časovou službou

Algoritmus NTP, Network Time Protocol

- Hierarchie NTP serverů
  - ✓ hodiny serveru ve vrstvě 1 řídí signál UTC z vrstvy 0
  - ✓ hodiny serverů ve vrstvě 2 se synchronizují s uzly vrstvy 1, . . .
  - ✓ na úrovni listů jsou klientské stanice



Algoritmus NTP, Network Time Protocol

- Způsoby synchronizace
  - ✓ zprávy protokolů se zasílají protokolem UDP
  - ✓ NTP Multicast Mode, jeden server rozesílá info o čase skupině serverů (multicast, vhodné pro LAN), málo přesná metoda
  - ✓ NTP Procedure-Call Mode, časový server vrací časové razítko, na žádost, přesnější než NTP Multicast Mode, de facto Cristianův algoritmus
  - ✓ NTP Symmetric Mode mezi 2 servery v různých úrovních, servery si opakovaně vyměňují zprávy s časovými razítky, opravují chyby
- Dosahovaná minimalizace skluzu –
  řádově desítky ms ve WAN
  řádově jednotky ms v LAN

Problém použití hodin pro synchronizaci v DS

- Síťové hodiny lze synchronizovat s přesností nejvýše na milisekundy
- CPU provádějí bilióny operací / s,
  pro 3 GHZ CPU 1 ms = 3 000 000 operací
- Mezi 2 hodinami, které mají být synchronizované,
  je typicky prostor, ve kterém jedna CPU může provést milióny operaci, než druhá rozpozná stejné časové razítko
- Časová razítka reálného času sama o sobě nejsou dostatečným nástrojem pro řazení distribuovaných událostí
- pro kooperující procesy důležité je pořadí, nikoliv přesný čas
- nekooperující procesy nemusí být synchronizovány vůbec

Konfigurace, přechod, provedení

- Globální stav DS daný stavem jeho procesů a zprávami obsaženými v jeho kanálech je konfigurací DS
- Konfigurace vzniká postupně, po krocích zvaných přechody
- Systém přechodů sestává z
  - ✓ množiny konfiguraci C
  - ✓ binární relace přechodu $\rightarrow$ na C
  - ✓ množiny iniciálních konfigurací $I \subseteq C$
- Konfigurace $\gamma \in C$ je terminální, pokud neexistuje $\gamma \rightarrow \delta$ pro žádnou z $\delta \in C$
- Provedení algoritmu je posloupností konfigurací $\gamma_0\gamma_1\gamma_2\ldots$, kde $\gamma_0 \in I$ a $\gamma_i \rightarrow \gamma_{i+1}$ pro všechna $i \geq 0$
- Konfigurace δ je dosažitelná pokud $\gamma_0\gamma_1\gamma_2\ldots\gamma_k = \delta$, kde $\gamma_0 \in I$ a $\gamma_i \rightarrow \gamma_{i+1}$ pro všechna $0 \leq i < k$

Události

- Každý přechod v DS je vázaný na jistou událost
  v některém z procesů DS
  - ✓ V případě synchronních DS na dvě události ve dvou procesech DS
- Proces může generovat
  vnitřní událost,
  událost vyslání zprávy a
  událost příjmu zprávy
- Proces je iniciátor, pokud jeho první událostí je vnitřní
  událost nebo vyslání zprávy
  - ✓ DA je centralizovaný, pokud existuje právě jeden iniciátor
  - ✓ Decentralizovaný DA může mít více iniciátorů

Příčinné pořadí „stalo-se-před"

- Příčinné pořadí $a \prec b$ výskytů událostí a a b v provedení DA je nejmenší tranzitivní relace taková, že platí
  - ✓ a a b události ve stejném procesu a a se vyskytla dříve než b nebo
  - ✓ a je událost vyslání zprávy a b je událost přijetí této zprávy
- Pokud neplatí ani $a \preceq b$ ani $b \preceq a$ jsou události a a b souběžné
- Permutace událostí, která respektuje příčinné pořadí, výsledek provedení DA neovlivní
  - ✓ Takovou permutaci nazýváme výpočtem
  - ✓ Všechna konečná provedení výpočtu startující ve stejné konfiguraci končí v téže terminální konfiguraci

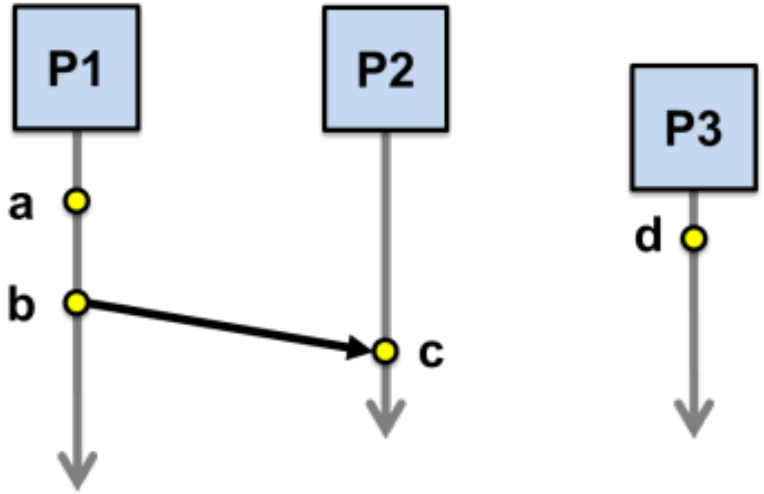
Fyzické hodiny synchronizaci procesů neumožňují

- Odlišnost driftů lokálních hodin v uzlech DS znemožňuje použít pro řazení událostí fyzický čas
- Idea řešení problému – (Lamport 1978)
  sledovat vztah *stalo-se-před* místo fyzického času

Řazení událostí v distribusledovatovaném prostředí

- Logický čas je budován na bázi relace stalo-se-před, značené $\rightarrow$
  - ✓ Jsou-li A a B (vnitřní) události ve stejném procesu a A se stala dříve než B, pak platí $A \rightarrow B$
  - ✓ Je-li A událost *zaslání zprávy* jedním procesem a B je událost *přijetí této zprávy* v jiném procesu, pak platí $A \rightarrow B$
  - ✓ Jestliže platí $A \rightarrow B$ a $B \rightarrow C$, pak platí $A \rightarrow C$

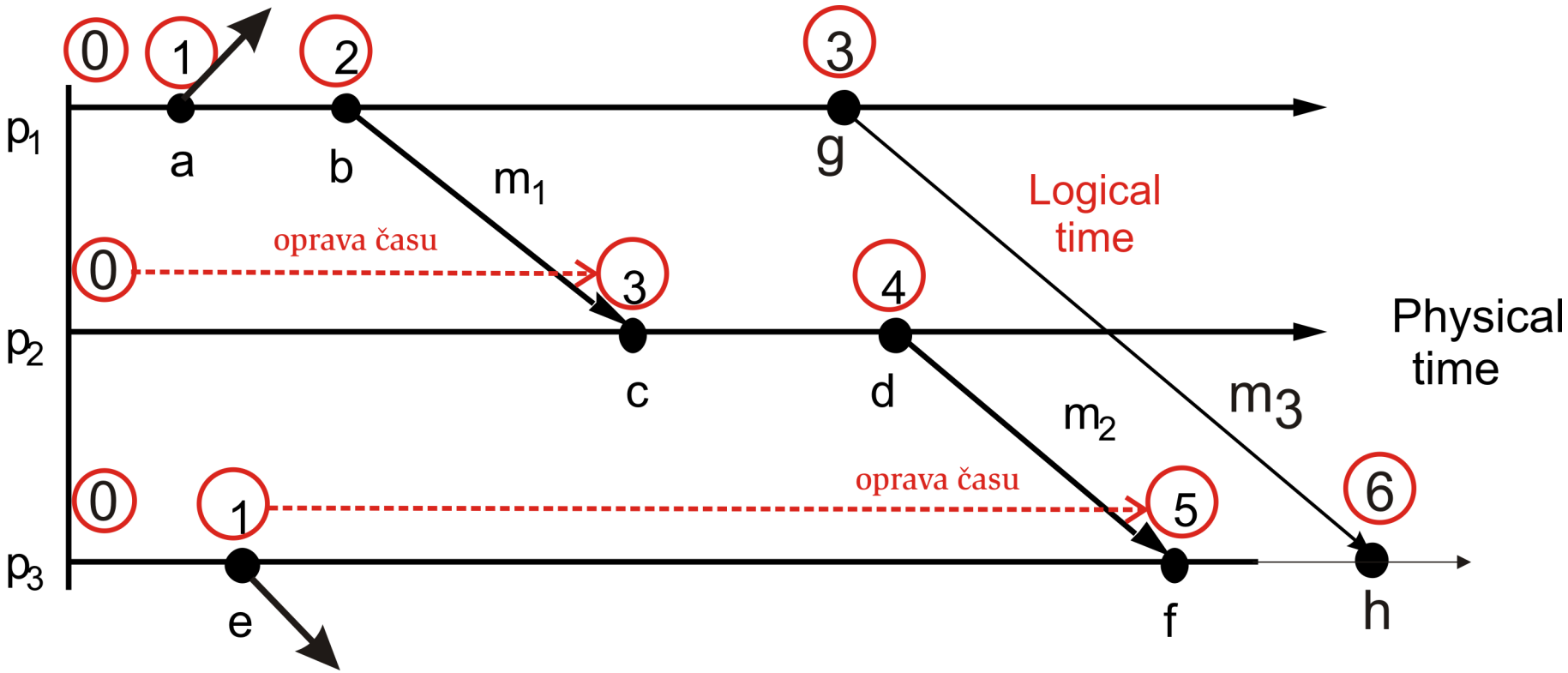
1. If **same process** and **a** occurs before **b**, then **a** → **b**
2. If **c** is a message receipt of **b**, then **b** → **c**
3. If **a** → **b** and **b** → **c**, then **a** → **c**
4. **a, d** not related by → so ***concurrent,*** written as **a** || **d**



Implementace relace $\rightarrow$, běh logického času

- S každou událostí v systému se sváže časové razítko, TS (*time stamp*)
- V každém procesu P_i udržují běh logického času logické (Lamportovy) hodiny C_i
  - ✓ mapují výskyt událostí ve výpočtu do částečné uspořádané množiny, ve které platí $a \rightarrow b \Rightarrow C(a) < C(b)$
  - ✓ logické hodiny lze implementovat např. jako čítač inkrementovaný před každou událostí v procesu, $C_i = C_i + 1$,
  - ✓ vysílanou zprávu proces doplní čas. razítkem, $TS_{zpravy} = C_i$
  - ✓ při příjmu zprávy přijímající proces nejprve nastaví na $C_i = \max(C_i, TS_{zpravy})$ a poté čítač času inkrementuje a poté se mu zpráva zpřístupní

Ilustrace běhu logického času

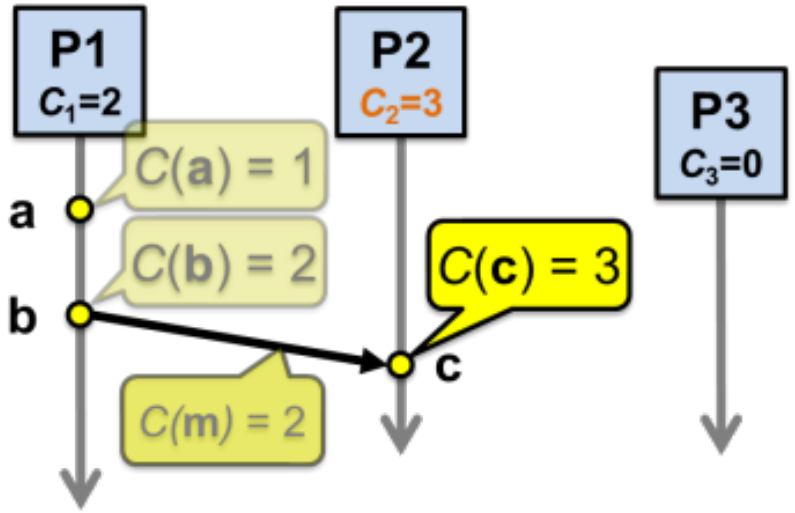


- $a \rightarrow b$, $b \rightarrow c$, $c \rightarrow d$, $d \rightarrow f$, $b \rightarrow g$, $g \rightarrow h$, $f \rightarrow h$
- $a \nrightarrow e$ a $e \nrightarrow a$, píšeme $a \parallel e$, a a e jsou souběžné události
- $e \parallel b$, logický čas $e <$ logický čas b, ale neplatí $e \rightarrow b$

Implementace relace $\rightarrow$, běh logického času

✓ Iniciálně má každý proces nulový logický čas

- Each process $\mathbf{P}_i$ maintains a local clock C_i
- Before executing an event, $C_i \leftarrow C_i + 1$
- On process $\mathbf{P}_j$ receiving a message **m**:
  - Set C_j and receive event time $C(\mathbf{c}) \leftarrow 1 + \max\{ C_j, C(\mathbf{m}) \}$

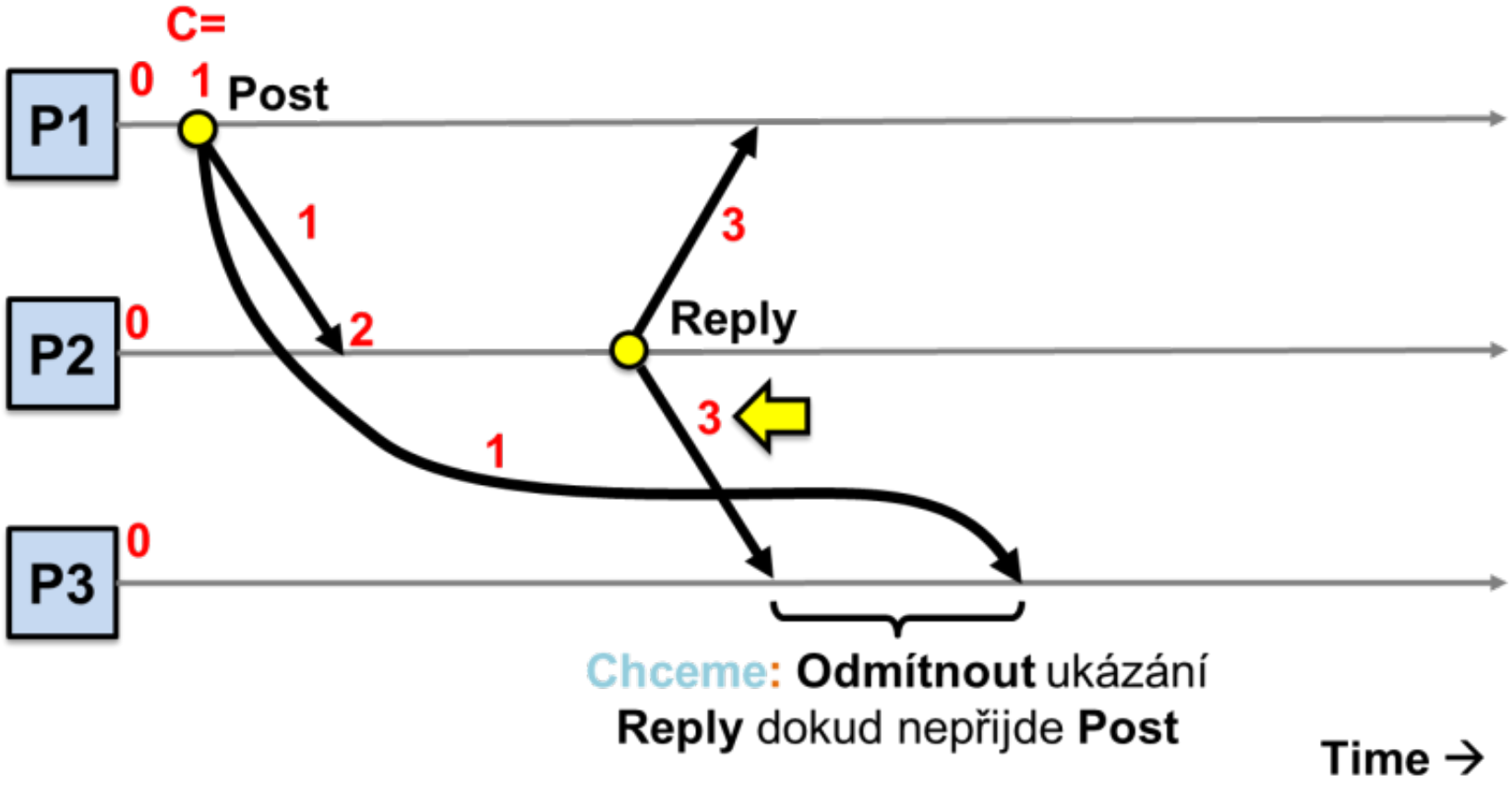


✓ Pro každý pár událostí A a B propojených posloupností událostí (události uskutečněné v jednom procesu nebo vyslání a příjem zprávy), pro který platí $A \rightarrow B$, platí $TS(A) < TS(B)$

✓ Pozor, z $TS(A) < TS(B)$ neplyne $A \rightarrow B$

Lamportova časová razítka nezachycují kauzalitu

Problém kauzality Lamportovy hodiny



- **Lze odmítnout ukázání zprávy s C(message) > local clock + 1?**
- **Nikoli, mezery mohou být způsobeny nezávislými vysíláními, …**

Příklad

- V procesech p_1, p_2, p_3 došlo k následujícím událostem
  - ✓ $p_1 : a, s_1, r_3, b$ (událost a, vyslání zprávy 1, příjem zprávy 3, ud. b)
    $p_2 : c, r_2, s_3$
    $p_3 : r_1, d, s_2, e$
- Tyto události se vyskytly v logických časech
  - ✓ $p_1 : \mathbf{1}(a), \mathbf{2}(s_1), \quad \mathbf{8}(r_3), \mathbf{9}(b)$
    $p_2 : \mathbf{1}(c), \quad \mathbf{6}(r_2), \mathbf{7}(s_3)$
    $p_3 : \quad \mathbf{3}(r_1), \mathbf{4}(d), \mathbf{5}(s_2), \mathbf{6}(e)$
- C přiřazují každé události a délku k nejdelšího příčinného řetězce $a_1 \rightarrow \ldots \rightarrow a_k = a$

Totální uspořádání logického času

- Dvě různé události generované ve dvou různých procesech mohou mít identické Lamportovo časové razítko, C
- např. žádosti o vstup do kritické sekce z více procesů, všechny mají stejnou hodnotou C
- spravedlivost při řešení vstupu do kritické sekce může požadovat totální uspořádání hodnot C v celém DS
- pak lze do časového razítka C doplnit např. id procesu a časová razítka se shodnou hodnotou času uspořádat dle pořadí id procesů
  - ✓ id procesů musí být jedinečné a řadové, např. *integer*, hodnoty

Vektorové časové razítko

- Nedostatkem (skalárního) Lamportova TS je, že
  z $TS(A) < TS(B)$ neplyne $A \rightarrow B$
  Lamportova časová razítka nezachycují kauzalitu
- Toto omezení lze řešit ve skupině N procesů tím, že místo skalárního (Lamportova) časového razítka TS, použijeme vektorové časové razítko V
  - ✓ Každý proces p_i si udržuje svoje vlastní vektorové razítko V_i
  - ✓ Vektorové časové razítko procesu p_i, tj. V_i, má tolik prvků kolik je procesů ve skupině
  - ✓ Iniciálně jsou v p_i všechny prvky V_i nulové, $V_i = (0, 0, \ldots)$
  - ✓ $V_i[i]$: logické hodiny p_i, počet událostí, které se dosud nastaly v p_i
  - ✓ Před tím než p_i vyšle zprávu m procesu p_j nastaví $V_i[i] := V_i[i] + 1$ (událost vyslání zprávy časově orazítkuje)

Vektorové časové razítko

- ✓ $V_i[j]$ reprezentuje znalost logického času p_j v p_i, tu p_i získává z vektorového razítka připojovaného vysílačem p_j ke zprávě přijaté p_i
- ✓ když p_j získá ve zprávě od p_i časové razítko V_i, ve svém V_j nastaví $V_j[i] := max(V_j[i], V_i[i])$ pro $i = 1, 2, \ldots N$
- ✓ Poněvadž p_i před vysláním zprávy inkrementoval $V_i[i]$ a p_j inkrementuje $V_j[i]$ pouze když dostane od p_i časové razítko s větší hodnotou pro p_i, platí $V_j[i] \leq V_i[i]$
- ✓ Ve vektorovém časovém razítku V_j je $V_j[j]$ počet událostí orazítkovaných p_j a $V_j[i]$ pro $i \neq j$ počet událostí v p_i, které příp. ovlivnily p_j,
- ✓ Když událost a označíme razítkem V, každý prvek V je čítačem událostí v jednom procesu, které kauzálně předcházejí a

Vektorové časové razítko

- ✓ Pro porovnání vektorových časových razítek platí pravidla

  $V = V'$ iff $V[j] = V'[j]\ for\ j = 1, 2, , \ldots N$

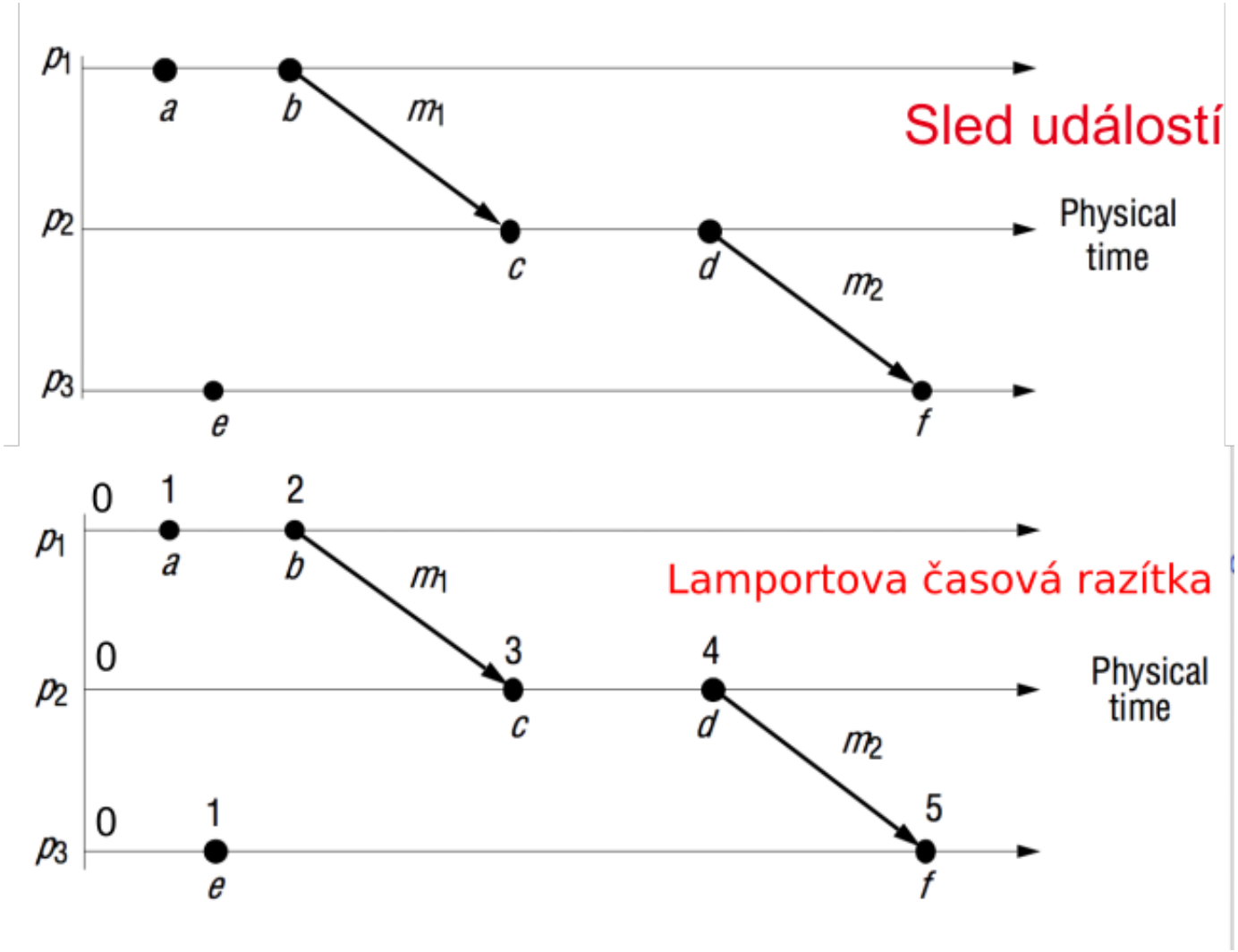
  $V \leq V'$ iff $V[j] \leq V'[j]\ for\ j = 1, 2, , \ldots N$

  $V < V'$ iff $V \leq V' \wedge \exists j : V[j] < V'[j]$

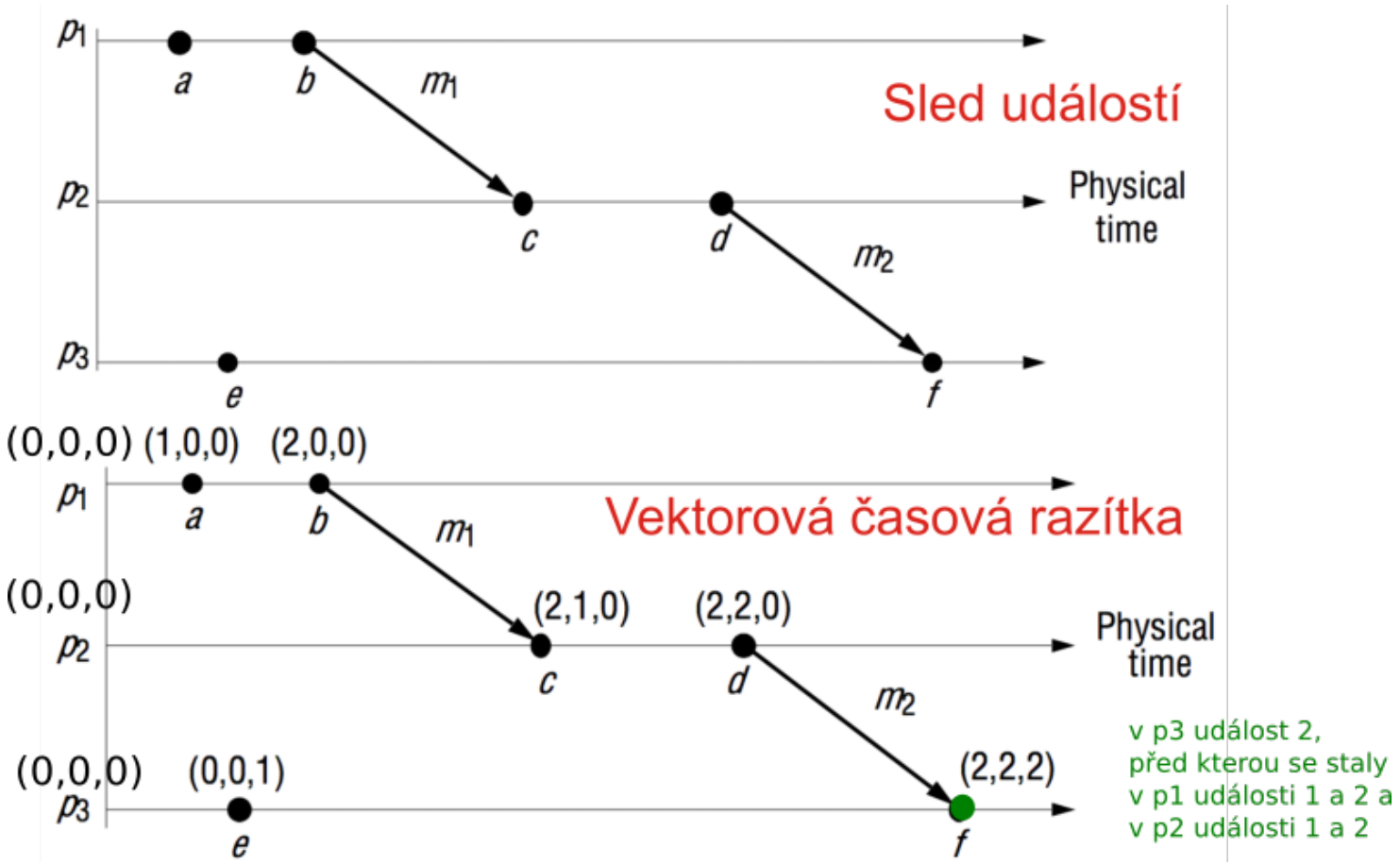
Vektorové časové razítko

- Lze ukázat, že pokud o událostech a, b platí $a \rightarrow b$,
  např. a je vyslání zprávy a b je přijetí této zprávy
  pak platí $V(a) < V(b)$
  - ✓ přijimač inkrementuje svůj čas ve V a všechny ostatní položky ve V zůstanou přinejmenším stejně velké jako ty v časovém razítku odesílatele, tedy $V(a) < V(b)$
  - ✓ Pokud platí $V(a) < V(z)$, pak mezi a a z existuje řetěz událostí svázaných relací ,,stalo se před''
  - ✓ Ze znalosti $C(a) < C(z)$ nelze odvodit žádný závěr
  - ✓ Jsou-li události a a b souběžné, pak neplatí ani $V(a) \leq V(b)$ ani $V(b) \leq V(a)$ a tudíž pokud události a a b nejsou souběžné, pak platí
    iff $V(a) < V(b) \Rightarrow a \rightarrow b$
    iff $V(b) < V(a) \Rightarrow b \rightarrow a$
  - ✓ Vektorové razítko vypovídá o kauzalitě vztahu událostí

Lamportovo časové razítko



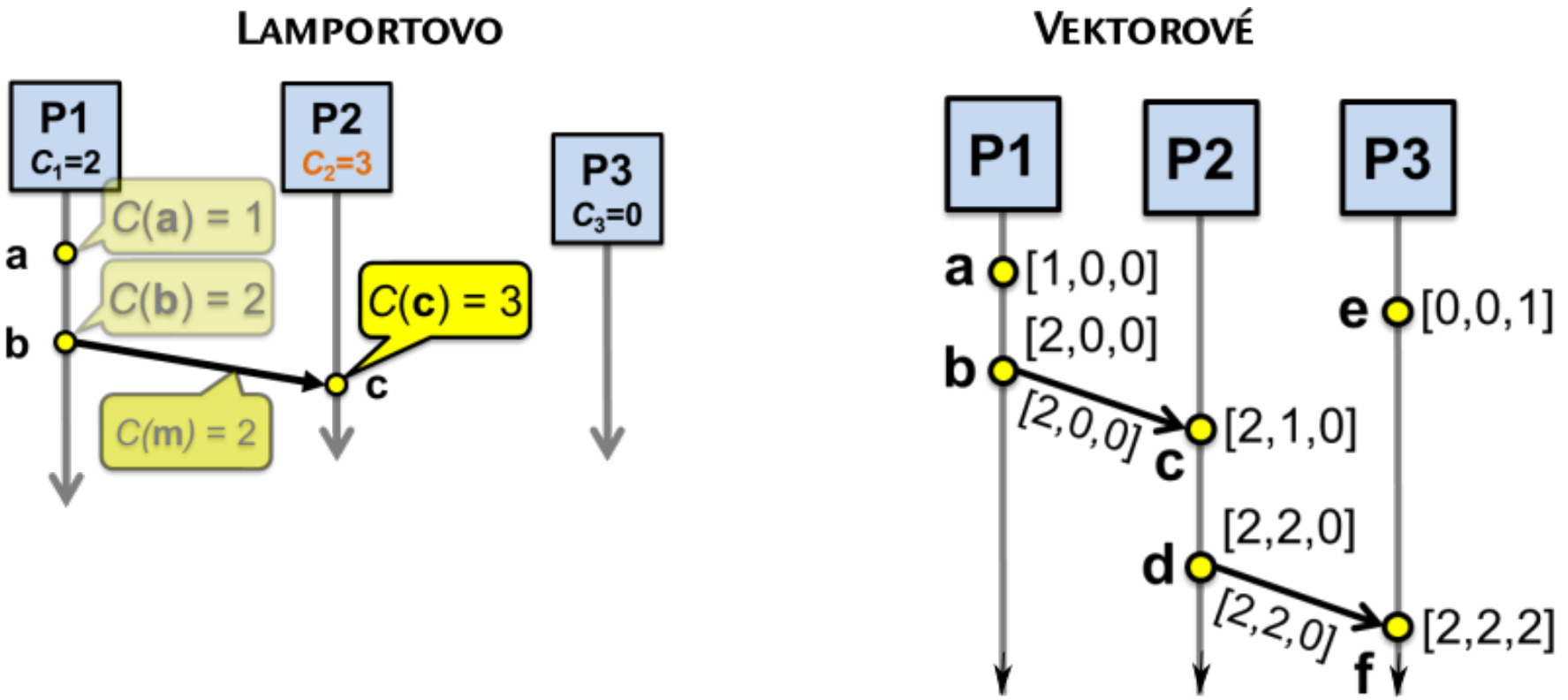
Vektorové časové razítko



✓ $V(a) < V(f) \Rightarrow a \rightarrow f$

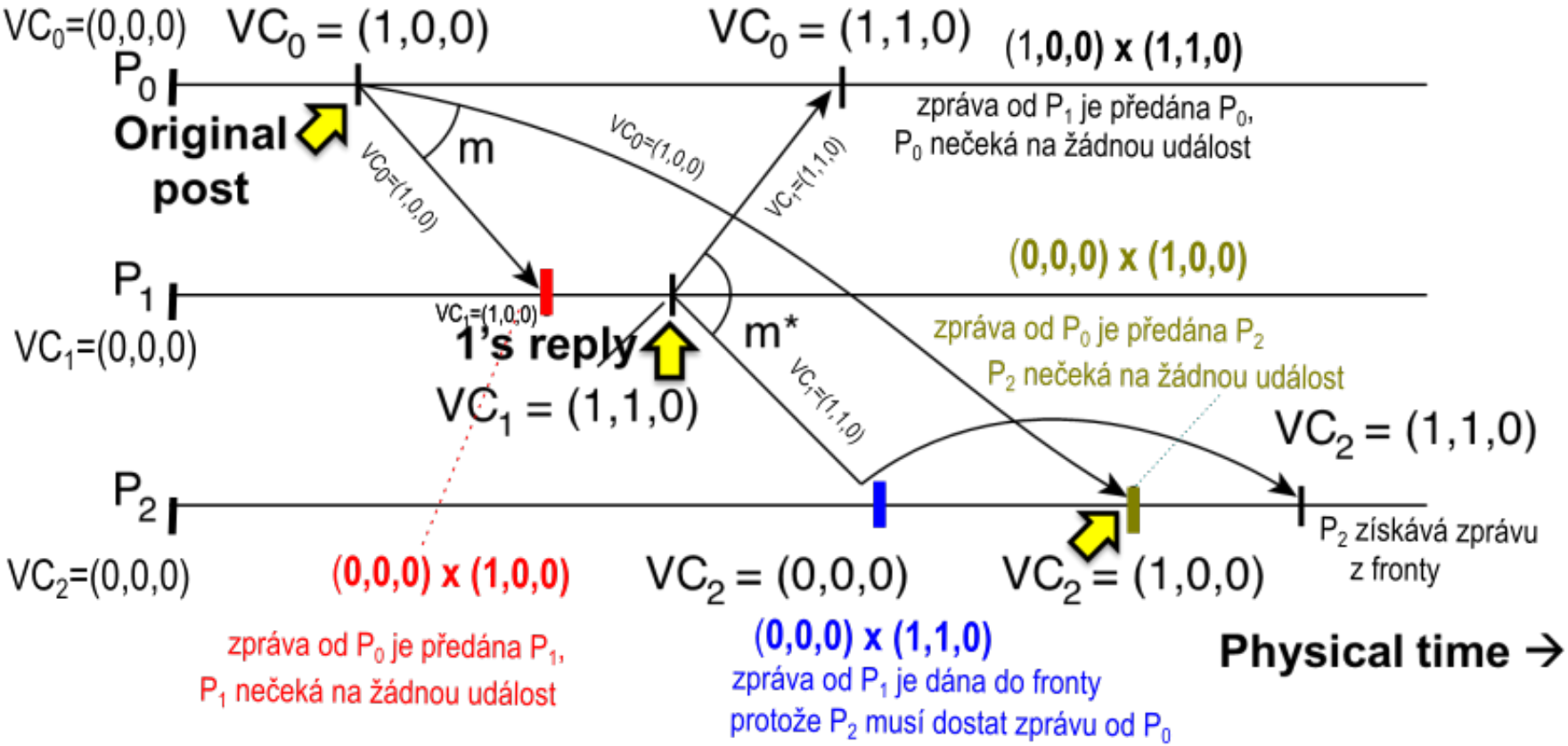
✓ $c \parallel e$, neplatí ani $V(c) \leq V(e)$ ani $V(e) \leq V(c)$

lamportovo a vektorové razítko



- ✓ je-li $V(a) < V(d)$ pak existuje řetěz událostí vázaných relací stalo-se-před mezi a a d
- ✓ mezi nezávislými událostmi neexistuje řetěz událostí vázaných relací stalo-se-před

Aplikace vektorového razítka



- User 0 posts, user 1 replies to 0's post; user 2 observes

Složitost výpočtu distribuovaných algoritmů

Mirou složitosti může být

- Počet vyměňovaných zpráv, budeme používat nejvíce
- Bitová složitost
  - ✓ počet vyměňovaných bitů zprávami
  - ✓ ma smysl pouze v případech velmi dlouhých zpráv
- Časová složitost – předpoklady:
  - ✓ doba zpracování zprávy v komponentě DS je obv. zanedbatelná
  - ✓ zaslání zprávy spotřebuje alespoň 1 časovou jednotku
- Vesměs nás zajímá nejhorší případ a průměrný případ výpočtu
- O-notace: Pokud ve výpočtu participuje n procesů a nejhorší provedení výpočtu má kvadratickou složitost počtu zpráv, $O(n^2)$, pak se při tomto provedení vymění řádově n^2 zpráv

Model pro studium distribuovaných algoritmů

- Distribuovaný algoritmus DA realizuje v DS,
  ve kterém participuje $n > 1$ procesů
- Každý proces P_i běží na obecně na jiném uzlu sítě (procesoru)
  - ✓ běží $\equiv$ provádí posloupnosti událostí, např.
    lokální výpočet $\rightarrow$ zaslání zprávy $\rightarrow$ přijetí zprávy
- Pro jednoduchost vyjádření algoritmů v celé přednášce platí
  1 uzel = 1 proces, uzel a proces jsou synonyma
- Pokud se neřekne jinak, pak procesy jsou z hlediska logiky
  řízení aplikace DS vzájemně rovnocenné, platí symetrie
- V některých variantách DA procesy mohou mít asymetrické
  postavení z hlediska logiky řízení aplikace
  (např. model klienti-server)

Model pro studium distribuovaných algoritmů

- Zprávy vyslané jedním procesem jinému procesu jsou

  a) přijímané v pořadí jejich vysílání,

  b) doručované silně souvislou komunikační sítí,
  tj. každý proces může komunikovat s každým procesem

  (platnost podmínek a) + b) odpovídá protokolu TCP)

  c) doručené v konečném čase

  o rychlosti jednotlivých komunikačních kanálů
  nelze vyslovit žádný jiný předpoklad

- Pokud neřekneme jinak, nedochází k výpadkům ani komunikačních kanálů ani uzlů

  ✓ Varianty distribuovaných algoritmů v prostředí s poruchami (výpadky) budeme studovat samostatně

Požadované vlastnosti distribuovaných algoritmů

- Bezpečnost, *Safety*, Nothing bad happened yet
  - ✓ Sledovaná podmínka, cíl:
    Z globálního stavu DS (stav všech procesů tvořících DS) je normálními (validními) stavovými přechody nedosažitelný jistý nežádoucí stav
  - ✓ Např. dosahuje se vzájemné vyloučení kritických sekcí procesů, zabraňuje se uváznutí procesů, ...
  - ✓ Typicky se dokazuje indukcí: *jestliže* X *platí pro* $n = 1$ *a jestliže* X *platí pro* $n = m$ *a pro* $n = m + 1$, *pak* X *platí pro všechna* n
  - ✓ Narušení bezpečnosti (tj. narušení dosažitelnosti cíle algoritmu) se prokazuje v konečném počtu kroků řešení
  - ✓ Řešení problému nenarušující bezpečnost je korektní řešení

    Podmínka bezpečnosti musí být splněná v každé konfiguraci každého provedení algoritmu, jedná se o invariant

    Předpoklad P je invariantem, pokud platí $P(\gamma)$ pro všechny $\gamma \in I$ a jestliže $\gamma \rightarrow \delta$, pak platí i $P(\delta)$.

Požadované vlastnosti distribuovaných algoritmů

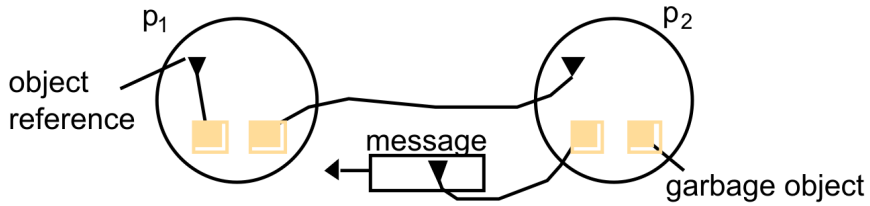
- Živost, *Liveness*, Something good eventually happens
  - ✓ Vlastnost globálního stavu DS zajišťující, že

    jistou posloupností normálních (validních) stavových přechodů
    je dosažitelný jistý, konkrétní žádoucí stav
  - ✓ Např. v konečném počtu kroků algoritmu se zvolí vedoucí uzel v síti nebo proces žádají o vstup do kritické cesty získá právo vstoupit do kritické sekce v konečném čase
  - ✓ Narušení podmínky živosti se prokazuje pouze v nekonečném počtu kroků řešení
  - ✓ Korektní řešení problému (splňující podmínku bezpečnosti) nenarušující živost je úplné, kompletní řešení
  - ✓ Podmínka živosti musí být splněná alespoň v jedné konfiguraci každého provedení algoritmu

Problém znalosti globálního stavu v DS

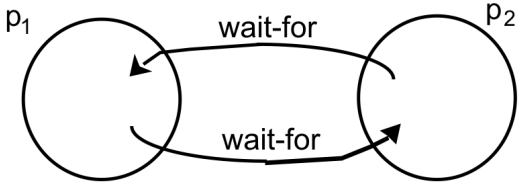
- Platí jistá vlastnost v DS ? Detekce globální vlastnosti, např.
  - ✓ Je jistý objekt dále už nepoužívaný ?
    (lze na něj aplikovat *garbage collection*, nikdo na něj neodkazuje)
  - ✓ Došlo k uváznutí ?
  - ✓ Došlo k ukončení distribuovaného algoritmu ?
  - ✓ Nestačí znát stav procesů, musí se znát i stav komunikačních kanálů

a. Garbage collection



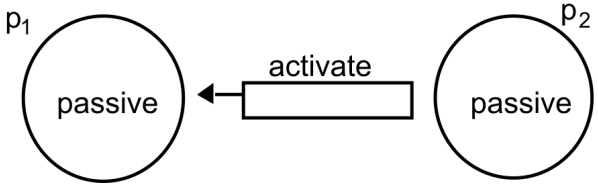
Nestačí znalost stavu uzlů, je nutné znát i stav komunikačních kanálů

b. Deadlock



procesy vzájemně čekají na zprávu jeden od druhého

c. Termination



Všechny procesy sdělí na dotaz, že jsou pasivní. P1 po přijetí aktivační zprávy se stane znovu aktivní, ač již sdělil, že je pasivní

Budování globálního stavu DS, momentka

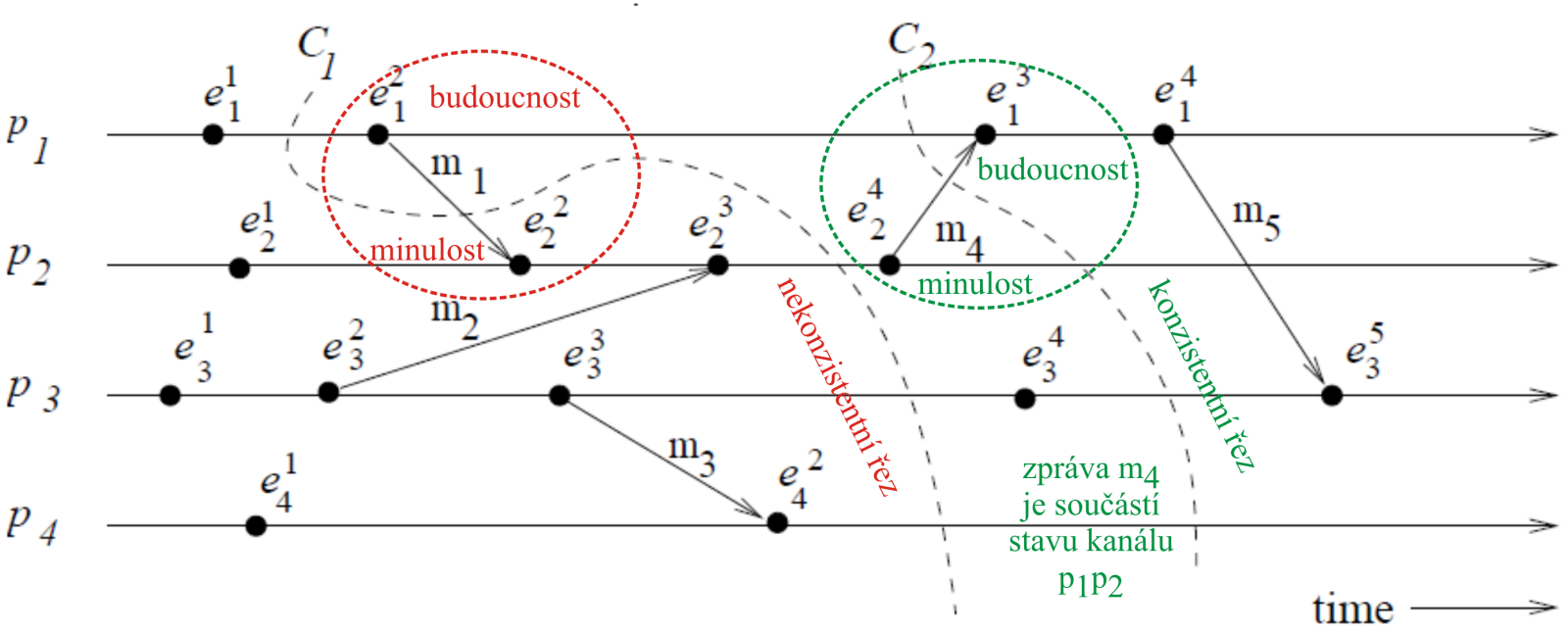
- Jak odvodit globální stav kolekce procesů v asynchronním DS ze znalosti lokálních stavů uzlů pořízených výměnou zpráv v různých okamžicích běhu času ?
- Momentka (*snapshot*) provádění jistého DA poskytuje informaci o některé konfiguraci provádění výpočtu v DS
  - ✓ Momentku potřebujeme pro restart výpočtu po výpadku
  - ✓ Momentka umožní provést detekci uváznutí
  - ✓ Momentka usnadňuje ladění výpočtu
- Přirozeně je žádoucí získat momentku bez zastavení výpočtu
  - ✓ Při výpočtu pak rozlišujeme
    - základní zprávy řešící vlastní DA a
    - řídicí zprávy získávající momentku

Budování globálního stavu DS, momentka, konzistentní řez

- Momentka výpočtu podle DA v DS sestává
  - ✓ z lokálních momentek stavu každého procesu a
  - ✓ ze stavů kanálů všech kanálů v DS (výčet zpráv obsažených v kanálech)
- Momentka výpočtu dle DA v DS je smysluplná, konzistentní, pokud úplně popisuje některou konfigurací provádění DA
- V časovém diagramu běhu procesů v DS můžeme vytvořit řez rozdělující běh výpočtu na minulost a budoucnost vůči vztažnému bodu definovanému řezem (na události vyskytnuvší se v minulosti a na události, které teprve nastanou)
- Konzistentní řez odpovídá konzistentnímu stavu, ve kterém každá zpráva přijatá v minulosti dané konzistentním řezem byla rovněž v minulosti vyslaná

Problém znalosti globálního stavu v DS

- ✓ Momentky výpočtu podle DA v DS se musí vytvářet v konzistentních řezech časových běhů procesů v DS
- ✓ Jestliže proces p_i poslal zprávu m_{ij} procesu p_j po zaznamenáním momentky svého lokálního stavu, musí proces p_j vytvářet momentku svého stavu před zpracováním m_{ij}

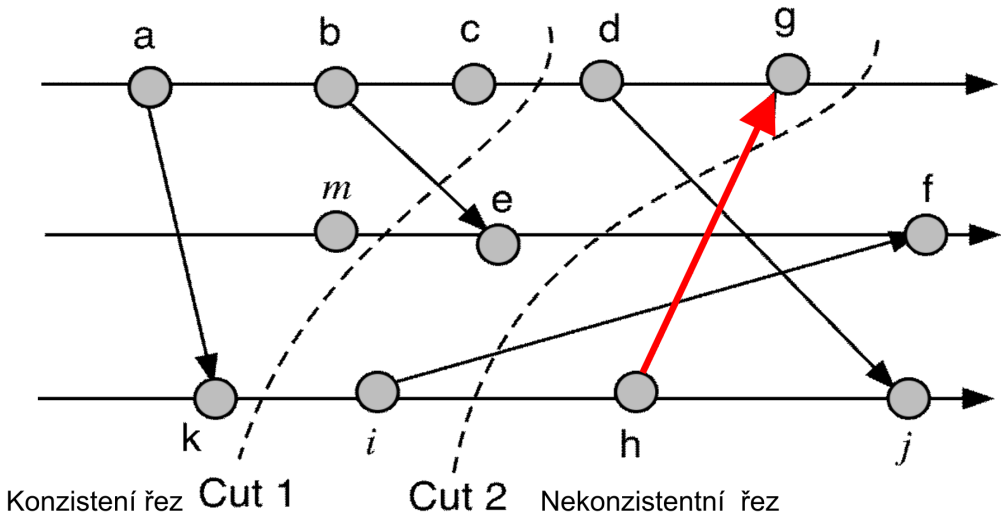


Global State Recording Algorithm, GSRA

- V jakém stavu se se nachází systém N procesů v DS ?
- Dovedeme to zjistit ?
  - ✓ V distribuovaném systému, ve kterém neexistuje ani sdílená paměť ani systémové hodiny,
    je určování okamžitého globálního stavu obtížně řešitelné.
  - ✓ Jestli bude finanční systém tvořený milionem bank, které si sítí předávají 1 CZK, pak prostý dotaz postupně posílaný bankám může zjistit, že ve finančním systému není žádná CZK i že je jich tam milion
- Proč potřebujeme znát globální stav systému N procesů v DS ?
  - ✓ Pro detekci konce činnosti systému procesů v DS
    řešících distribuovanou aplikaci,
    pro detekci uváznutí procesů v DS sdílejících zdroje,
    při vytváření kontrolního bodu pro návrat při obnově, . . .

GSRA, konzistentní řez stavu, konzistentní stav

- Řez C je konzistentní, když platí $(a \in C) \cap (b \prec a) \Rightarrow b \in C$



- Zjištený stav (*snapshot*) je konzistentní, pokud ho formují události náležející konzistentnímu řezu
- Pokud $C1 \prec C2$, pak je $C2$ novější stav
  - ✓ Nás prakticky zajímá nejnovější stav
  - ✓ Stav musí být zjistitelný ,,za pochodu'', neinvazivně

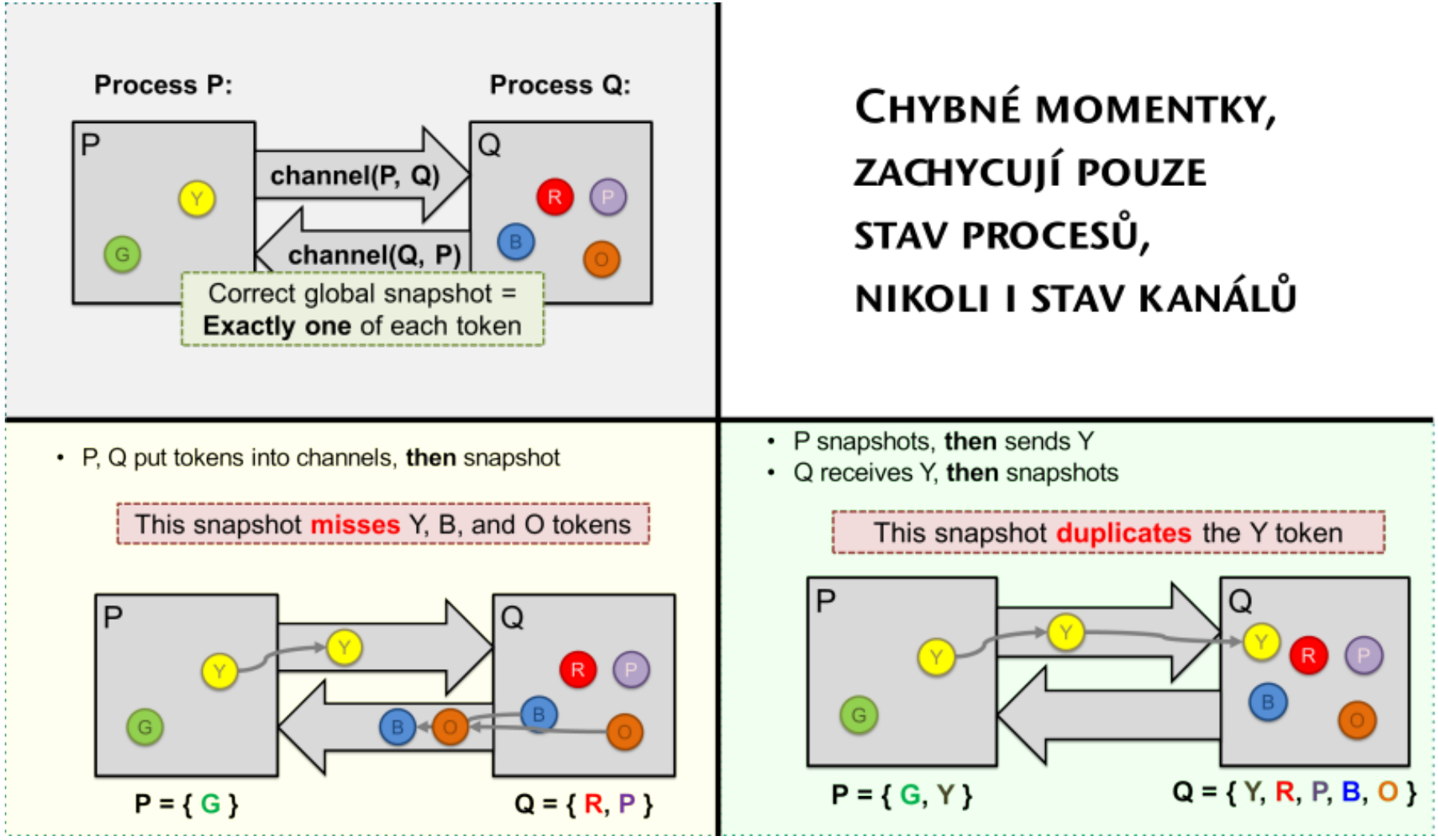
Algoritmus zjištění globálního stavu DS, Chandy & Lamport

- Model DS tvořeného N procesy
  - ✓ Procesy a komunikační kanály nevypadávají,
    každá vyslaná zpráva dorazí k příjemci neporušená a právě jednou
  - ✓ Komunikační kanály jsou jednosměrné, pracují v režimu FIFO,
    každý proces může komunikovat s každým procesem,
    graf sítě je silně souvislý, mezi každými dvěma procesy PQ existují
    dva takové kanály, *channel (P,Q)* a *channel (Q,P)*
  - ✓ Zjišťování globálního stavu může spustit kterýkoliv proces kdykoliv
  - ✓ Zjišťování globálního stavu nenarušuje běh procesů z pohledu aplikace
  - ✓ Každý proces je schopný zaznamenat svůj stav a
    stav každého svého vstupního kanálu (co mu přišlo a dosud zpracoval)

    pokud proces p_i poslal zprávu m_{ij} procesu p_j a p_j ji dosud nepřevzal,
    pak m_{ij} náleží stavu vstupního kanálu $p_i \longrightarrow p_j$

    (taková zpráva je např. přijata službou OS, ale dosud nebyla doručena
    službou middleware aplikačnímu procesu)

Konstrukce momentky stavu DS, příklad 1

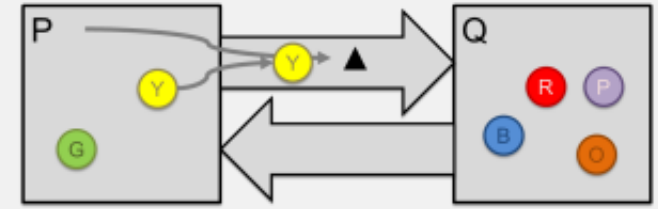


Idea validního řešení momentky

- Musí se zachytávat rovněž stav kanálů
- Pro sledování stavu kanálů se bude vysílat
  řídicí zpráva *marker*
- Proces zaznamená svůj lokální stav
  A
  do každého svého výstupního kanálu vyšle marker
- Každý proces si pamatuje,
  zda už momentku svého lokálního stavu udělal
- Pokud proces do přijetí markeru momentku dosud nevytvořil,
  udělá ji a zaznamená svůj lokální stav A do každého svého výstupního kanálu vyšle marker

Konstrukce momentky stavu DS, příklad 2

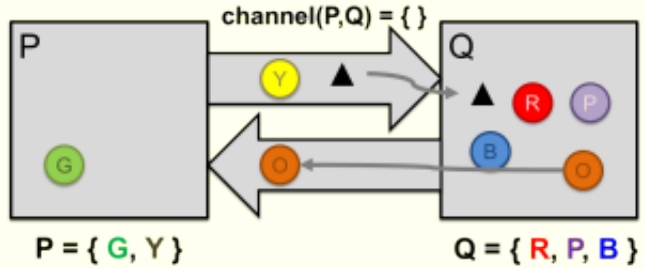
- P **snapshots and sends marker, then** sends Y
- **Send Rule:** Send marker on all outgoing channels
  - **Immediately after snapshot**
  - **Before** sending any **further messages**



snap: P = { G, Y }

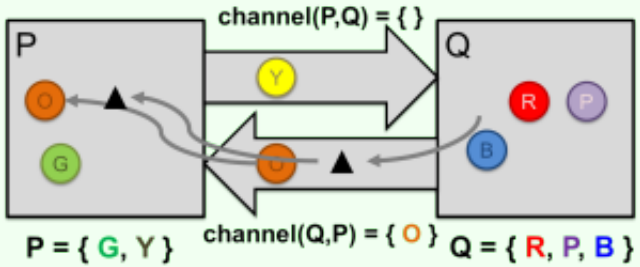
VALIDNÍ MOMENTKY, ZACHYCUJÍ STAV PROCESŮ, I STAV KANÁLŮ

- At the same time, Q sends orange token O
- Then, Q receives marker ▲
- **Receive Rule (if not yet snapshotted)**
  - On receiving marker on channel ***c*** record *c*'s state as **empty**



P = { G, Y } **Q = { R, P, B }**

- Q sends marker to P
- P receives orange token O, then marker ▲
- **Receive Rule (if already snapshotted):**
  - On receiving marker on ***c*** record *c*'s state: **all msgs from *c* since snapshot**



P = { G, Y } **Q = { R, P, B }**

Algoritmus určení globálního stavu DS, Chandy & Lamport

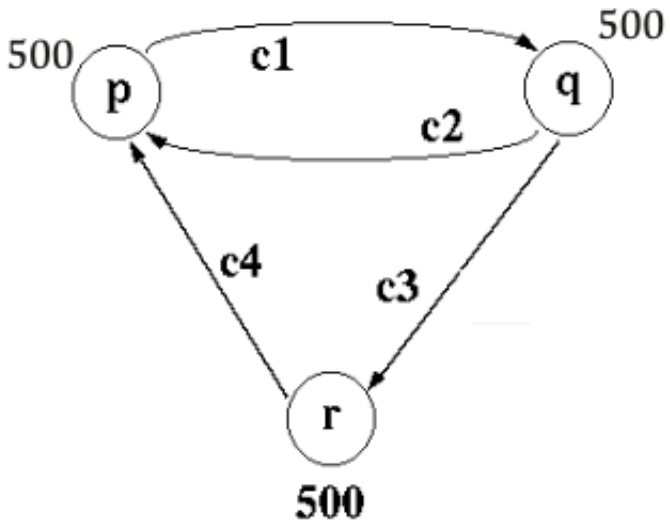
- Proces iniciující zjištění globálního stavu – iniciátor
  - ✓ zaznamená momentku svého lokálního stavu a pošle řídicí zprávu, marker, po svých výstupních kanálech všem svým sousedům v DS
- Když marker získá proces, který dosud nevypracoval momentku svého lokální stavu
  - ✓ zaznamená momentku svého lokálního stavu a pošle ji iniciátorovi
  - ✓ vstupní kanál ze kterého získal marker označí jako prázdný a pošle marker svým sousedům svými výstupními kanály
- Když marker získá proces, který už vypracoval momentku svého lokální stavu
  - ✓ zaznamená stav vstupního kanálu, ze kterého dříve získal marker
    stav = všechny zprávy od posledního záznamu svého stavu do přijetí markeru
    a pošle ho iniciátorovi

Algoritmus určení globálního stavu DS, Chandy & Lamport

- Iniciátor sestaví globální momentku stavu DS jakmile
  zná lokální momentky stavů všech procesů a zprávy,
  které byly uloženy „*v etéru*", nezpracované, v kanálech
- Kanály jsou FIFO, globální momentka je smysluplná
- Marker v kanálech odděluje zprávy na ty,
  které jsou zahrnuté do momentky lokálního stavu od těch,
  které do něj zahrnované nejsou
- Složitost algoritmu odpovídá
  zaslání $O(e)$ zpráv a $O(d)$ času,
  kde e je počet hran v grafu sítě a d je průměr sítě
- Algoritmus je užitečný pro detekci platnosti
  stabilní podmínky (ukončení, uváznutí, . . .)

GSRA, příklad aplikace algoritmu Chandy–Lamport

Snapshot/State Recording Example



Finanční systém tvoří 3 banky p, q a r.

Na počátku je v každé bance 500 $,
lokální stav každého uzlu je 500 $.

V bankovním systému je 1 500 $.

Žádná banka není iniciátorem GSRA,
nikdo nepotřebuje znát globální stav,
tj., zda je zachováno integritní omezení,
že ve finančním systému se nachází 1500 $

GS

Node	Recorded State				
	state	c1	c2	c3	c4
p			{}		{}
q		{}			
r				{}	

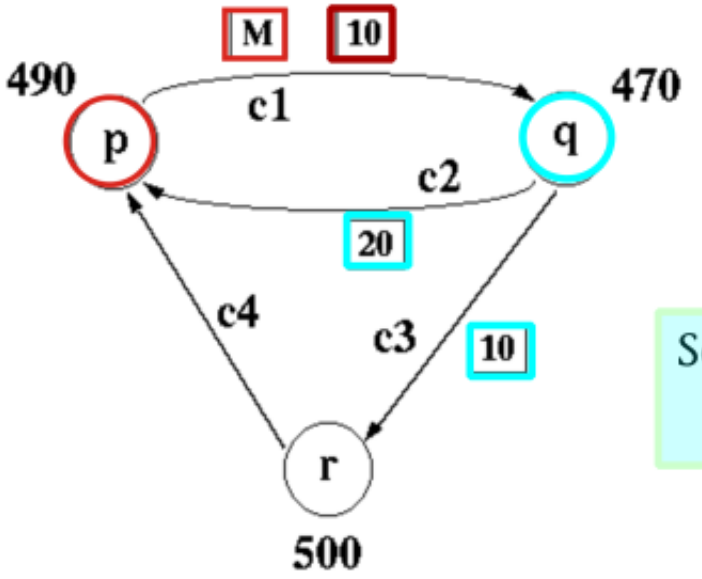
Jednotlivé řádky tabulky GS udržuje banka -- iniciátor GSRA,

Ostatní (uzly) banky zasílají iniciátorovi lokální momentky svých stavů a stavy svých vstupních kanálů.

Jakmile iniciátor bude znát všechny momentky, rozpozná ukončení GSRA.

GSRA, příklad aplikace algoritmu Chandy–Lamport

Snapshot/State Recording Example (Step 1)



Banka *p* posílá kanálem *c1* 10 $ bance *q* a spouští GSRA:

Zaznamenává svou lokální momentku okamžitého stavu (500 - 10 = 490 $) a do kanálu *c1* vysílá *značku* (M, *Marker*).

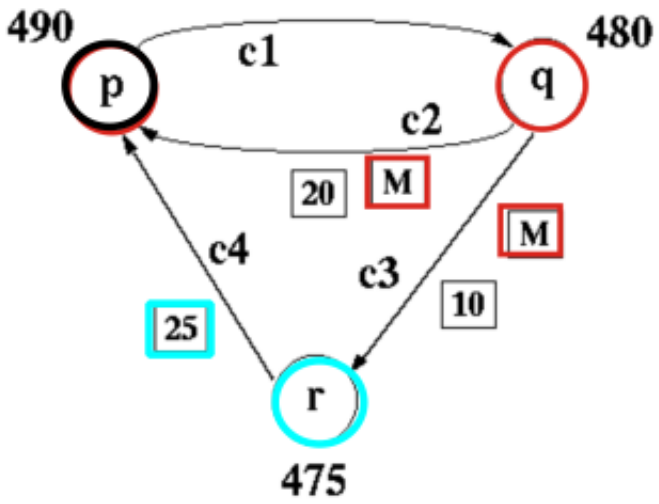
Současně banka *q* posílá
20 $ bance *p* kanálem *c2* a
10 $ bance *r* kanálem *c3*
lokální stav banky *q* je 500 - 20 - 10 = 470 $

Tyto převody dosud do *p* a *r* nedorazily, jsou součástí stavu kanálů *c2* a *c3*,

Node	Recorded State				
	state	c1	c2	c3	c4
p	490		{}		{}
q		{}			
r				{}	

GSRA, příklad aplikace algoritmu Chandy–Lamport

Snapshot/State Recording Example (Step 2)



Banka *q* získala převod 10 $,
zvýšila svůj stav z 470 $ na 480 $,
a poté získala z kanálu *c1* značku.

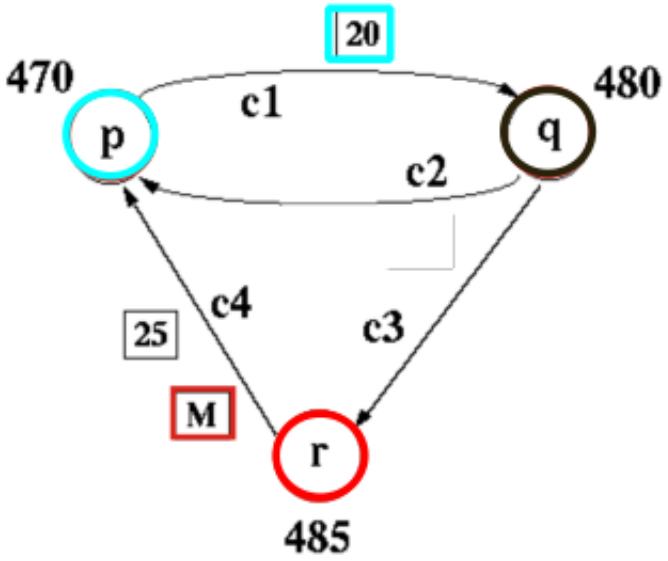
Protože banka *q* získala značku,
zaznamenala lokální momentku svého stavu 480 $
a vyslala značky na své použité
výstupní kanály *c2* a *c3*.

Současně banka *r* převádí bance *p* 25 $ po kanálu *c4*.
Její lokální stav je 500 - 25 = 475 $.
Nezáleží na tom, jestli *r* udělala převod před tím
nebo poté co *q* získala značku
a vytvořila lokální momentku svého stavu.

Node	Recorded State				
	state	c1	c2	c3	c4
p	490		{}		{}
q	480	{}			
r				{}	

GSRA, příklad aplikace algoritmu Chandy–Lamport

Snapshot/State Recording Example (Step 3)



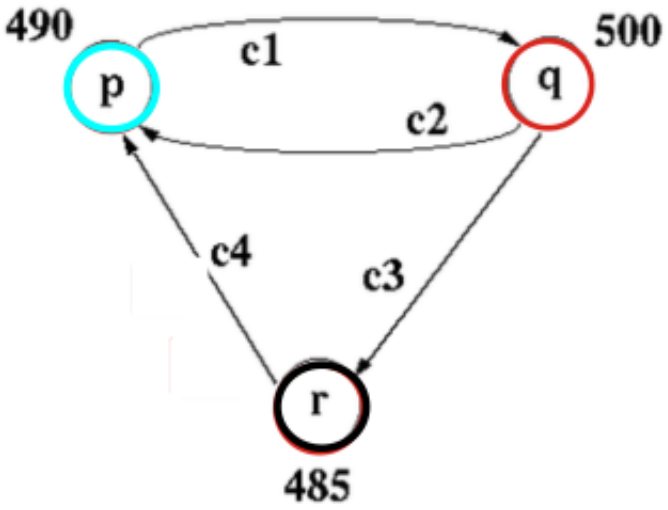
Banka *r* získává kanálem *c3* převod 10 $
a značku,
zvyšuje stav z 475 $ na 485 $
a zaznamenává lokální momentku tohoto stavu.
Poté vysílá značku na svůj
použitý výstupní kanál *c4*.

Mezitím banka *p* posílá dalších 20 $
kanálem *c1* bance *q*.
Stav banky *p* klesá na 490 - 20 = 470 $.

Node	Recorded State				
	state	c1	c2	c3	c4
p	490		{}		{}
q	480	{}			
r	485			{}	

GSRA, příklad aplikace algoritmu Chandy–Lamport

Snapshot/State Recording Example (Step 4)



Node	Recorded State				
	state	c1	c2	c3	c4
p	490		{20}		{}
q	480	{}			
r	485			{}	

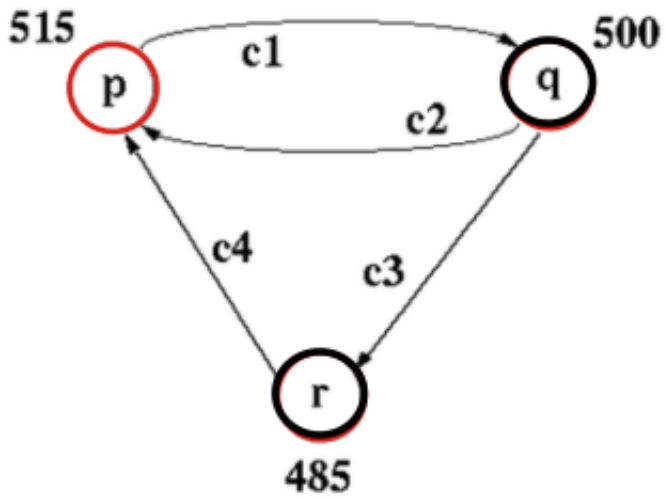
Banka *q* získává kanálem *c1* převod 20 $ zvyšuje svůj stav na 500 $.

Banka *q* nemění svoji lokální momentku stavu, změna stavu se stala po příjmu značky.

Také banka *p* získává převod 20 $ kanálem *c2*. Banka *p* zaznamená převod 20 $ pouze jako část svého zaznamenaného stavu kanálu *c2*.
jakmile získá z tohoto kanálu marker

GSRA, příklad aplikace algoritmu Chandy–Lamport

Snapshot/State Recording Example (Step 5)



Node	Recorded State				
	state	c1	c2	c3	c4
p	490		{20}		{25}
q	480	{}			
r	485			{}	

Banka *p* získává kanálem *c4* převod 25 $ zvyšuje svoji hodnotu na 515 $.

Když banka *p* získá značku z kanálu *c4*, je zaznamenávání lokálních momentek hotovo, všechny banky získaly značku na všech svých vstupních kanálech.

Tabulka ilustruje v jednotlivých řádcích finálně zaznamenané lokální momentky. Řádky tabulky mohou jednotlivé procesy posílat iniciátorovi např. na vyžádání iniciátorem.

Globální stav systému zjišťovaný bankou *p* 490 + 20 +25 + 480 + 0 + 485 + 0 = 1500 $, lokální momentky stavu uzlů říkají 1455 $, stavy kanálů říkají, že 45 $ je v „etéru".