SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Diplomski studij

Informacijska i komunikacijska tehnologija:

Telekomunikacije i informatika

Računarstvo:

Programsko inženjerstvo i informacijski sustavi

Računarska znanost

Raspodijeljeni sustavi

8. Otpornost na neispravnosti u raspodijeljenom okružju

Ak. god. 2020./2021.

Creative Commons

- slobodno smijete:
  - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo
- pod sljedećim uvjetima:
  - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.
Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava. *Tekst licence preuzet je s http://creativecommons.org/*

Sadržaj predavanja

- Uvod, definicije pojmova
- Otpornost procesa na ispade
- Sporazum skupine procesa
- Pouzdana komunikacija skupine procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak nakon ispada

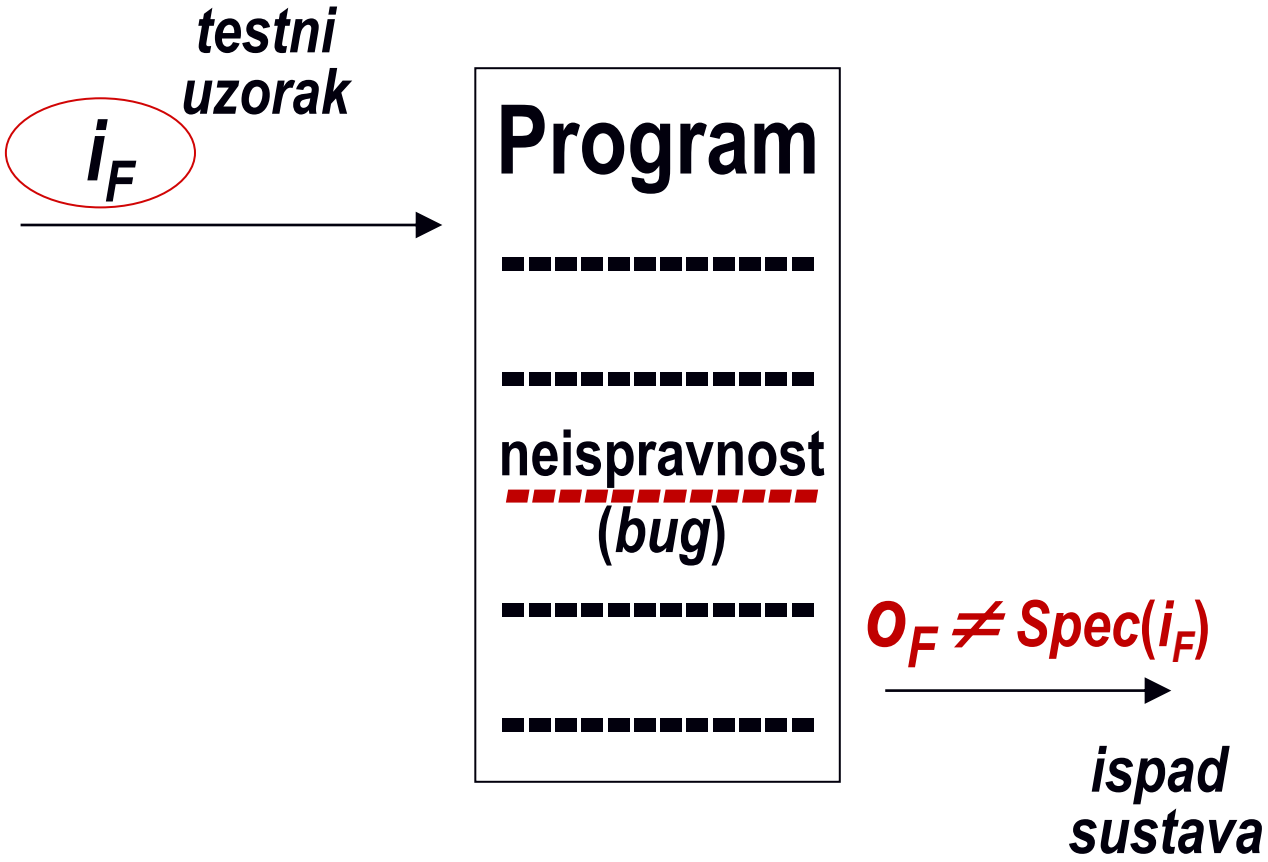
Otpornost na neispravnosti

Fault tolerance (*provide service even in the presence of faults*)

- sposobnost sustava za obavljanje definirane usluge bez obzira na postojeće **neispravnosti** koje izazivaju **ispad** nekih komponenti sustava
- osobina sustava, može prikriti ispad komponenti raspodijeljenog sustava, definiraju se procedure za oporavak sustava
- funkcionalnost sustava može biti ograničena uz negativan utjecaj na njegove performance

> Lamport: "*A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable*."

Podsjetimo se izvođenja programskog koda…



Osnovni pojmovi

- Neispravnost (*fault*)
  - npr. dio programskog koda (*bug*), neispravan komunikacijski kanal, pogreške prilikom oblikovanja sustava
  - uzrok ispada sustava, pronalaženje neispravnosti je težak i važan problem
  - prolazne, isprekidane i trajne neispravnosti
- Ispad sustava (*failure*)
  - stanje sustava koje se detektira kroz nemogućnost korištenja jedne ili više njegovih usluga
  - posljedica neispravnosti, signalizira postojanje neispravnosti u raspodijeljenom sustavu

Vrste ispada u raspodijeljenom okruženju

- Ispad kanala
  - proces *p* je poslao poruku procesu *q*, ali *q* poruku ne prima, jer npr. kanal gubi poruke
- Ispad procesa
  - ispad zaustavljanja (*stoping failure*): proces ne mijenja stanja (ne izvode se prijelazi) premda se ne nalazi u završnom stanju
  - bizantinski ispad (*Byzantine failure*): proces generira proizvoljne izlaze

Vrste ispada procesa

Vrsta ispada	**Opis**
Ispad procesa *ispad zaustavljanja*	Proces neočekivano ulazi u stanje zaustavljanja i ne odgovara na nove zahtjeve.
Pogreška u komunikaciji *pogreška primanja* *pogreška slanja*	Proces ne odgovara na primljeni zahtjev. Proces ne prima zahtjev. Proces ne šalje odgovor.
Vremenska pogreška	Proces šalje odgovor nakon isteka vremenskog roka.
Pogrešan odgovor *sadržaj* *pogrešna promjena stanja poslužitelja*	Generirani odgovor je neispravan. Sadržaj odgovora je neispravan. Poslužitelj ulazi u pogrešno stanje nakon primljenog zahtjeva.
"Bizantska pogreška"	Proces proizvodi proizvoljan odgovor u proizvoljnom trenutku.

Redundancija

Ključna tehnika za prikrivanje neispravnosti raspodijeljenog sustava.

Primjeri redundancije:

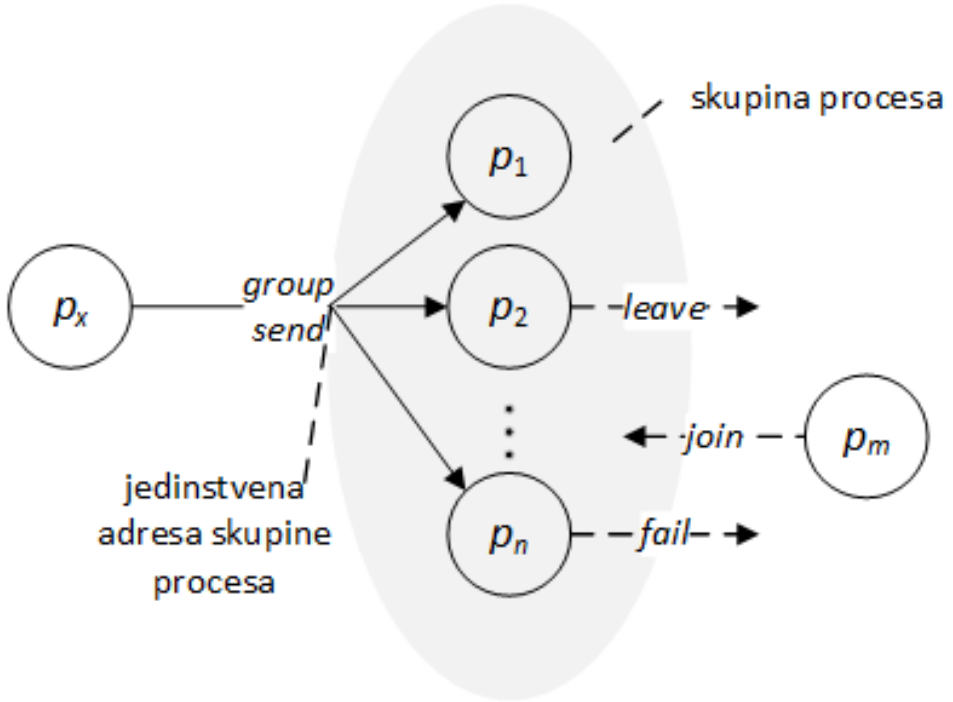
- redundancija informacija
  - npr. Hammingov kod
- vremenska redundancija
  - ponavljanje neke operacije u vremenu
- fizička redundancija
  - dodavanje dodatne opreme (npr. vruća rezerva) ili procesa u sustav (repliciranje procesa)

Sadržaj predavanja

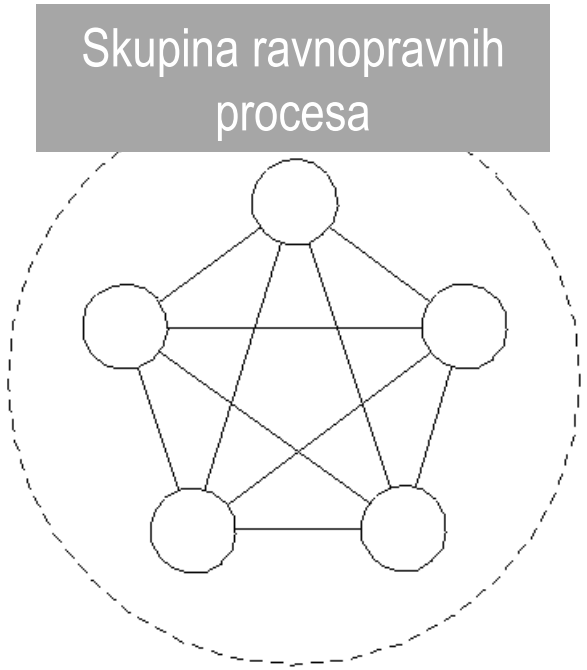
- Uvod, definicije pojmova
- Otpornost procesa na ispade
- Sporazum skupine procesa
- Pouzdana komunikacija skupine procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak nakon ispada

Otpornost procesa na ispade: skupina procesa

- Zamjena jednog procesa skupinom identičnih procesa
  - replikacija procesa radi prikrivanja ispada jednog ili *k* procesa
  - poruka se šalje skupini procesa, svi procesi uspješno primaju ili ne primaju poruku
  - skupine procesa su dinamične
  - jedan proces može biti član više skupina
  - administracija skupine procesa

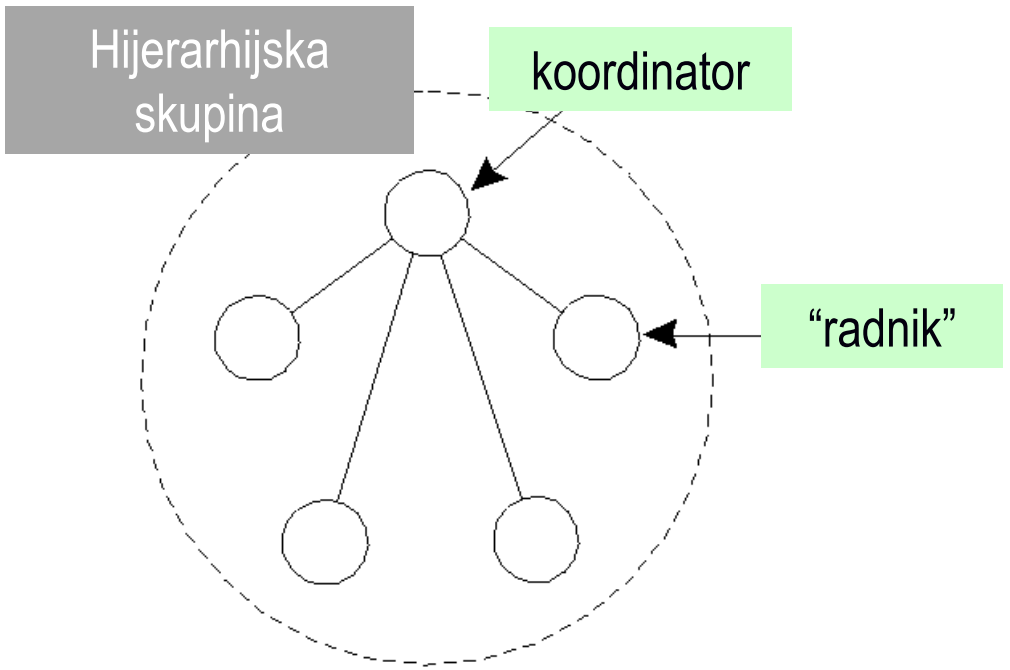


Organizacija skupine procesa



(a)

Prednost?
Nedostatak?



(b)

Prednost?
Nedostatak?

Otkrivanje ispada procesa iz skupine

U slučaju ravnopravnih procesa u skupini

- svaki proces periodički šalje upit ostalim procesima i provjerava njihovo stanje (“*are u alive?*”)
- proces pasivno čeka i prati poruke koje prima od ostalih članova skupine

U hijerarhijskoj skupini procesa koordinator provjerava stanje ostalih članova skupine.

Koliko procesa u skupini?

Tolerancija *k* ispada

- skupina može “preživjeti” ispad najviše *k* procesa

- dovoljan je ***k* + 1** proces da se osigura tolerancija na *k* ispada (jedan proces može preuzeti poslove skupine)
- potrebno je **2*k* + 1** procesa u skupini ako se pretpostavi *k* bizantskih ispada (mehanizam glasovanja: *k* + 1 ispravan proces će “nadglasati” *k* neispravnih)

Postizanje suglasnosti ili sporazuma

- ako se pretpostavi *k* bizantskih ispada, koliko je ukupno procesa potrebno da bi se postigla suglasnost (npr. otkrilo koji su ispali)?
- *sporazum skupine procesa*

Sadržaj predavanja

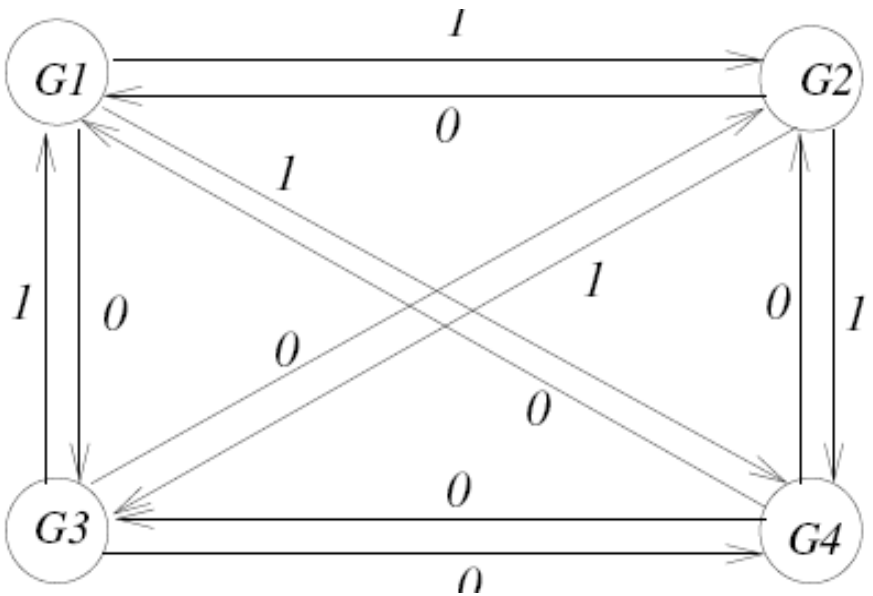
- Uvod, definicija pojmova
- Otpornost procesa na ispade
- Sporazum skupine procesa
- Pouzdana komunikacija skupine procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak

Sporazum skupine procesa

- Procesi trebaju postići sporazum o npr. vrijednosti neke varijable ili o tome hoće li ili neće izvršiti transakciju

Primjer:

- 4 generala trebaju započeti napad koordinirano u isto vrijeme
- za komunikaciju koriste glasnike
- komunikacija je nepouzdana i može trajati prilično dugo
- general izdajnik bi mogao slati pogrešne informacije ostalima



Mogu li generali postići sporazum o vremenu napada?

Problem bizantskog sporazuma (1/2)

Byzantine agreement problem [Lamport et al. 1982]

- **Problem**: svaki proces *i* definira inicijalnu vrijednost v_i, a skupina procesa treba postići sporazum o nizu vrijednosti za svaki proces iz skupine
- **Sporazum**: Svi ispravni procesi prihvaćaju isti niz vrijednosti

  $[v_1, v_2, \ldots v_n]$.
- **Ispravnost**: Ako je proces *i* ispravan i definira vrijednost v_i, svi ostali ispravni procesi prihvaćaju vrijednost v_i kao i-ti element niza. Ako je proces neispravan, ostali ispravni procesi mogu prihvatiti bilo koju vrijednost za taj proces.
- **Završetak**: Svaki ispravan proces će u konačnici prihvatiti vrijednosti niza za ispravne procese.

Problem bizantskog sporazuma (2/2)

Pretpostavke:

- *n* procesa
- *k* neispravnih procesa (**bizantski ispad**)
- proces *i* šalje vrijednost v_i ostalim procesima u skupini
- svaki proces treba kreirati niz **V** duljine *n* takav da vrijedi

  V[i] = v_i

Ponašanje procesa: **sinkrono** ili asinkrono

Slanje poruka: **jednoodredišno**, višeodredišno

Kašnjenje poruka: **ograničeno**, neograničeno

Isporuka poruka: **uređena**, neuređena

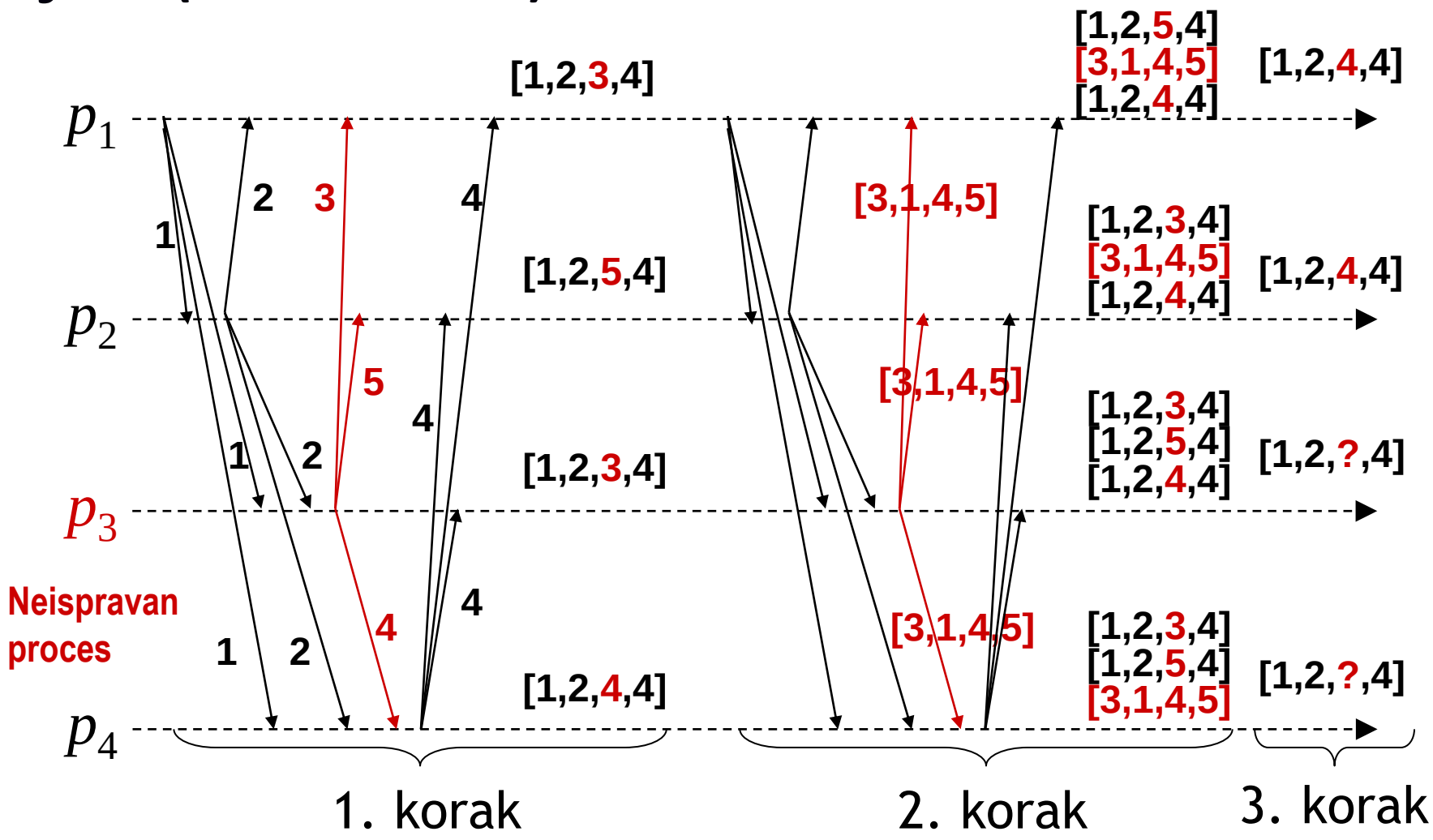
Skica algoritma za k =1

- **1. korak:**
  Ispravni procesi šalju svoju vrijednost v_i ostalim procesima, dok neispravni proces šalje proizvoljne vrijednosti.

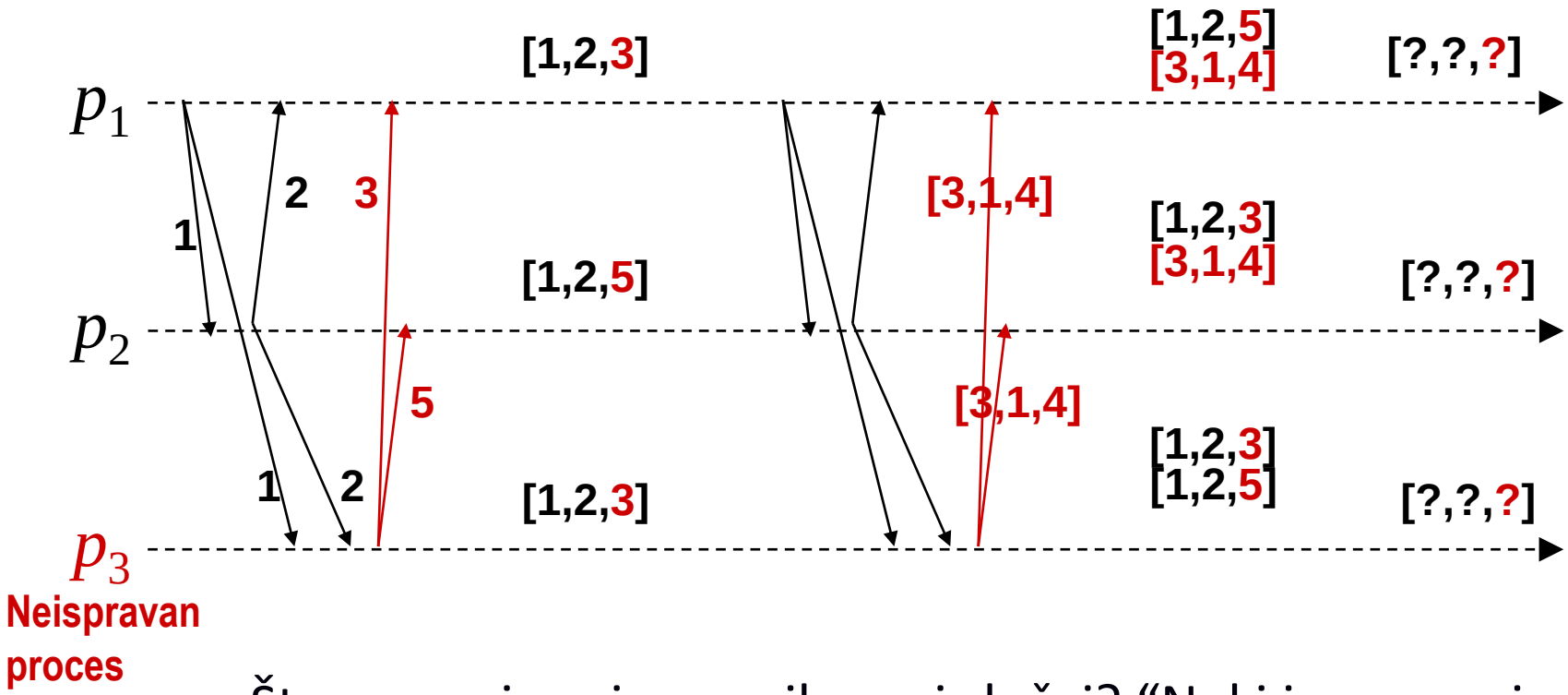
  Svaki proces prikuplja vrijednosti i sprema ih u **V**.
- **2. korak:**
  Svaki proces šalje ostalima svoj **V**, dok neispravni proces šalje proizvoljni **V**.
- **3. korak:**
  Konačno, svaki proces uspoređuje sve primljene nizove i odlučuje se za većinske vrijednosti.

- Procesi mogu donijeti odluku samo o vrijednostima za ispravne procese!

Primjer (n=4, k=1)



Primjer (n=3, k=1)



Što procesi znaju za prikazani slučaj? “Neki je proces ispao!”

Mogu li procesi donijeti odluku za prikazani slučaj?

Rasprava

- Sporazum u slučaju *n* procesa i *k* neispravnih procesa u bizantskom ispadu može se postići samo u slučaju **sinkronog sustava** i kada vrijedi

$$k \leq \left\lfloor \frac{n-1}{3} \right\rfloor$$

(za k =1 i n = 4 vrijedi, za k =1 i n = 3 ne vrijedi)

- Za sporazum u slučaju *k* neispravnih procesa, potrebno je $2k + 1$ ispravnih, tj. ukupno $n = 3k + 1$ procesa (više od 2/3 procesa treba biti ispravno!)
- Dokazano je da u slučaju **asinkronog sustava** u kojem se ne može jamčiti isporuka poruka procesi ne mogu postići sporazum niti za k=1

Sadržaj predavanja

- Uvod, definicija pojmova
- Otpornost procesa na ispade
- Sporazum skupine procesa
- Pouzdana komunikacija skupine procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak nakon ispada

Pouzdana komunikacija skupine procesa (1)

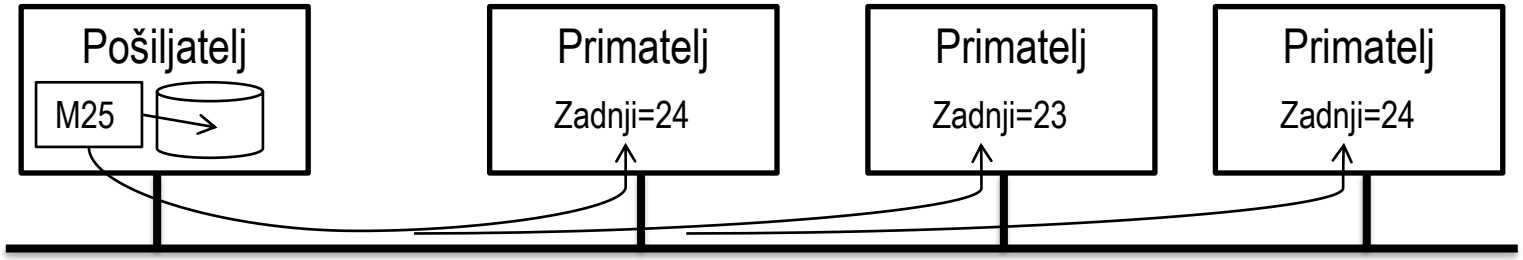
- jamstvo isporuke poruke svim procesima u skupini
- problemi
  - Koji procesi čine skupinu u trenutku slanja poruke?
  - Što se događa ako novi proces ulazi u skupinu procesa dok je isporuka poruke skupini u tijeku?
  - Što se događa ako dođe do ispada pošiljatelja poruke tijekom isporuke poruke ostalim procesima?
  - Što se događa ako jedan od primatelja ispadne tijekom isporuke poruke?

Pouzdana komunikacija skupine procesa (2)

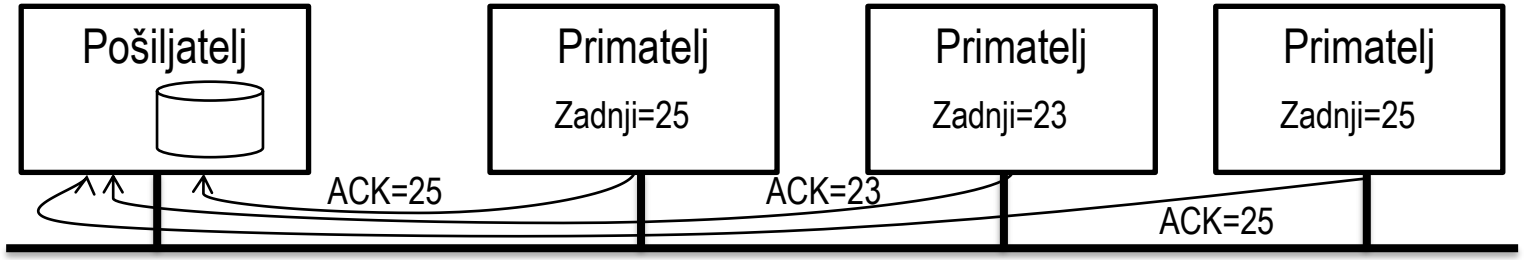
- najjednostavnija praktična implementacija
  - pouzdana komunikacija između svakog para procesa iz skupine, od točke do točke (npr. TCP)
- učinkovita praktična implementacija
  - pouzdano višeodredišno razašiljanje od jednog prema svim procesima u skupini (*multicast*)
  - skalabilnost

Pouzdana komunikacija bez mogućih ispada procesa (1/2)

Višeodredišna komunikacija s potvrdom ("nepouzdani kanal")



Isporuka poruke

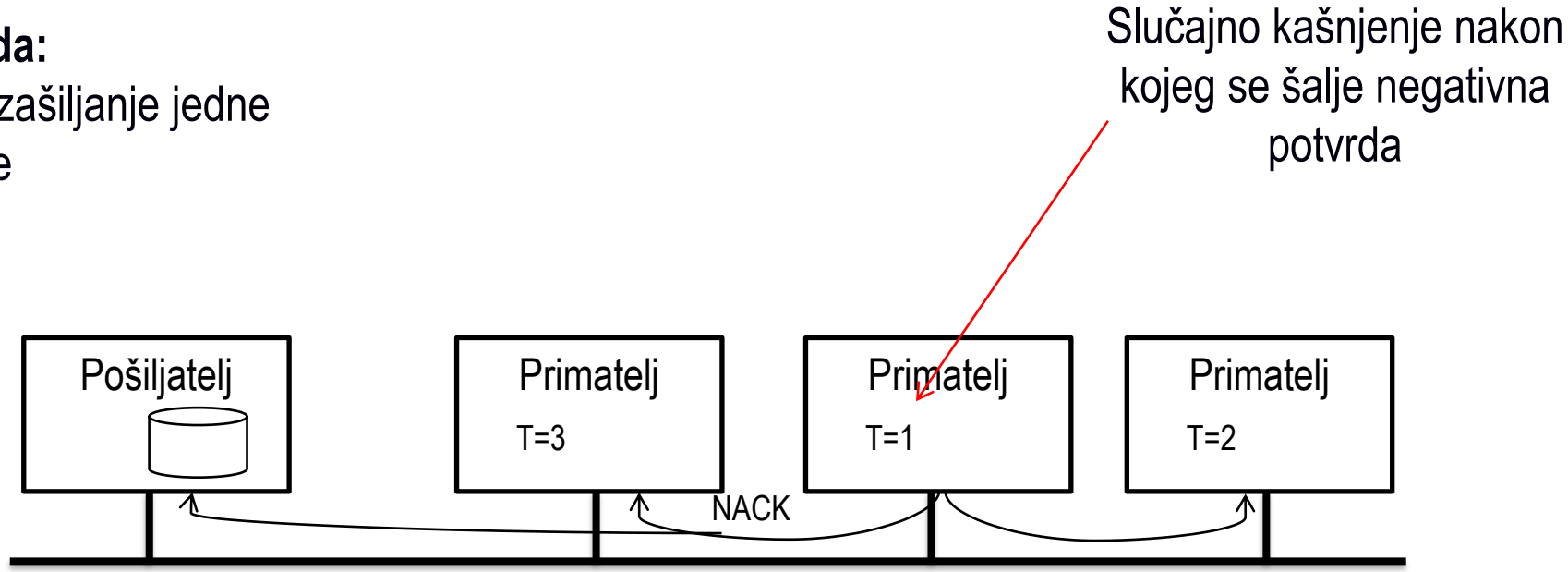


Slanje potvrde

Prema; Tanenbaum, Van Steen

Pouzdana komunikacija bez mogućih ispada procesa (2/2)

Potisnuta potvrda:
Višeodredišno razašiljanje jedne negativne potvrde



Višeodredišno slanje negativne potvrde

Prema: Tanenbaum, Van Steen

Pouzdana komunikacija s ispadima procesa

- jamči isporuku poruke svim ispravnim i dostupnim procesima u skupini ili niti jednom
- potrebno je osigurati i isporuku poruka u određenom redoslijedu

Notacija

- *p* - proces
- *G* – skupina, skup procesa – skupni pogled (*group view*)
- *m* – generirana poruka
- *vc* – poruka koja prenosi informaciju o dolasku ili odlasku procesa iz skupine – promjena pogleda (*view change*)

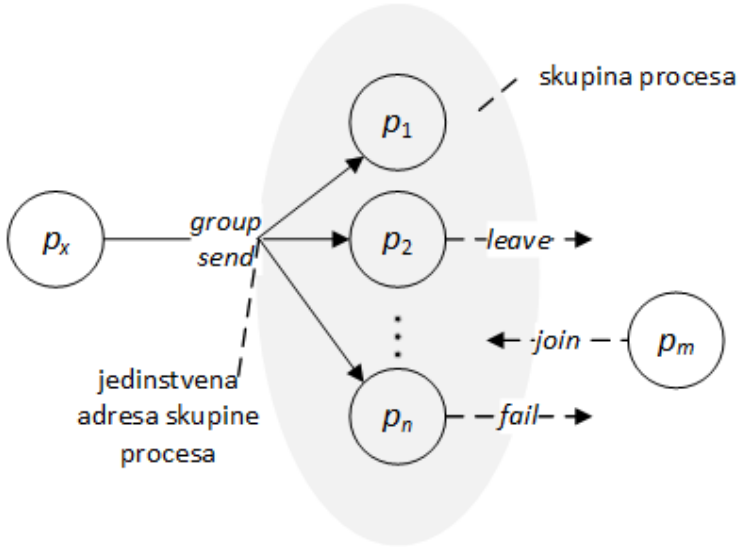
Virtualna sinkronost

Scenarij 1

- Proces *p* šalje poruku *m*, u tome trenutku postoji skupina procesa *G*
- Tijekom isporuke poruke *m* novi proces se uključuje u skupinu i generira se poruka *vc* (*view change*) koja se opet šalje svim članovima iz *G*
- Posljedica: poruke *m* i *vc* su istovremeno u tranzitu
- 2 moguća rješenja:
  - *m* isporučen svim članovima *G* prije isporuke *vc*
  - *m* nije isporučen niti jednom procesu iz *G*

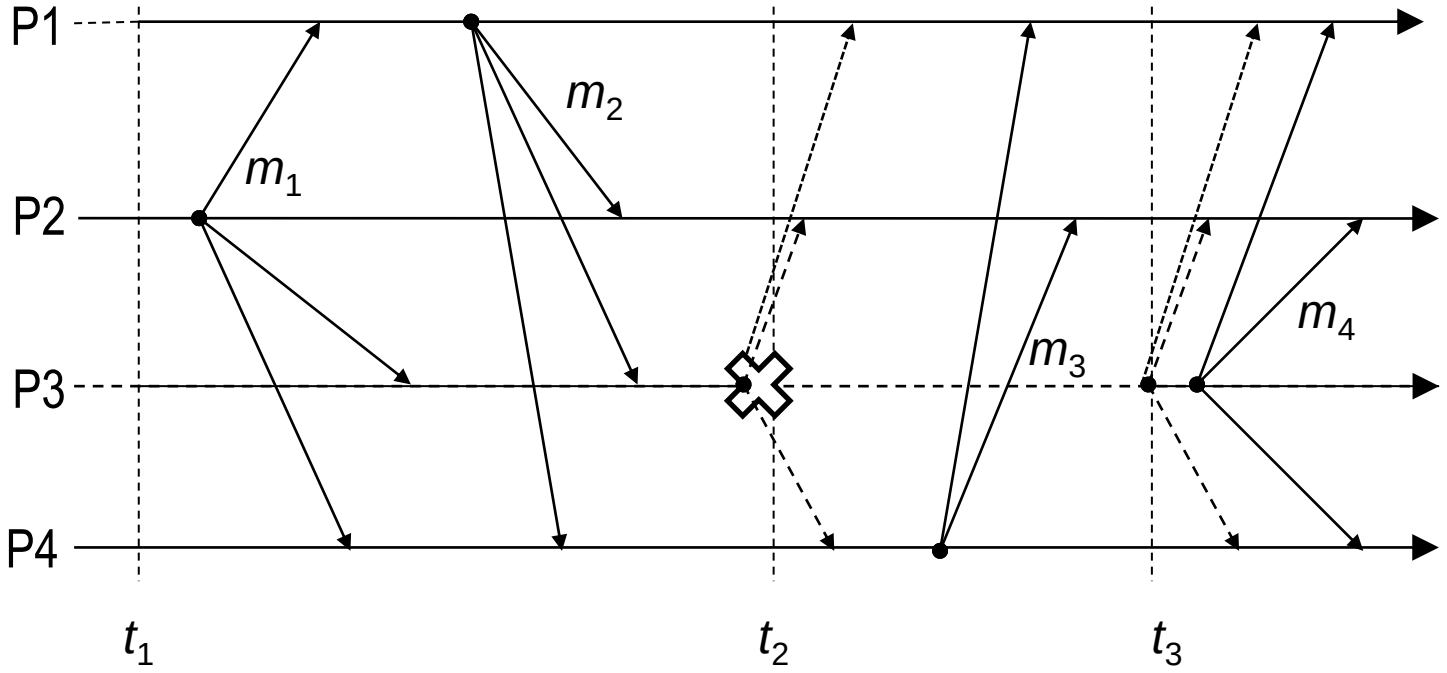
Scenarij 2

- Ako *p* pošalje *m* i ispadne prije isporuke *m* članovima G, ostali procesi ignoriraju *m* (ne treba osigurati isporuku ostalim ispravnim procesima, kao da je *p* ispao prije slanja *m*)



Primjer virtualne sinkronosti

t_2: ispad P3, generira se poruka *vc*



t_3: oporavak P3, poruka *vc*

Poruka se može isporučiti članovima iz G samo ako ne postoji poruka *vc* koja je istovremeno u tranzitu. Implementacija nije trivijalna.

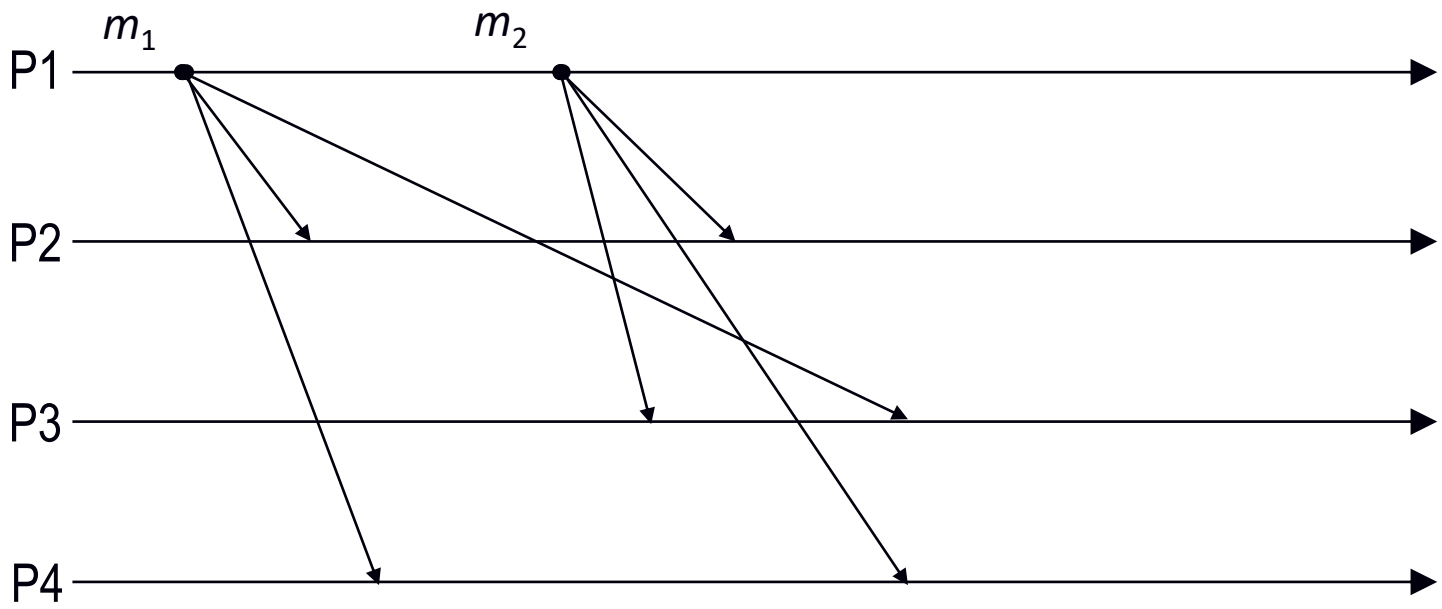
Prema: Tanenbaum, Van Steen

Pouzdana komunikacija: slijed poruka

- slijed kojim procesi primaju poruke od velike je važnosti, jer utječe na promjene stanja tih procesa
- slijed primljenih poruka može biti:
  - neuređen (*unordered multicast*)
  - FIFO (FIFO-*ordered multicast*)
  - potpuno uređen (*totally-ordered multicast*)

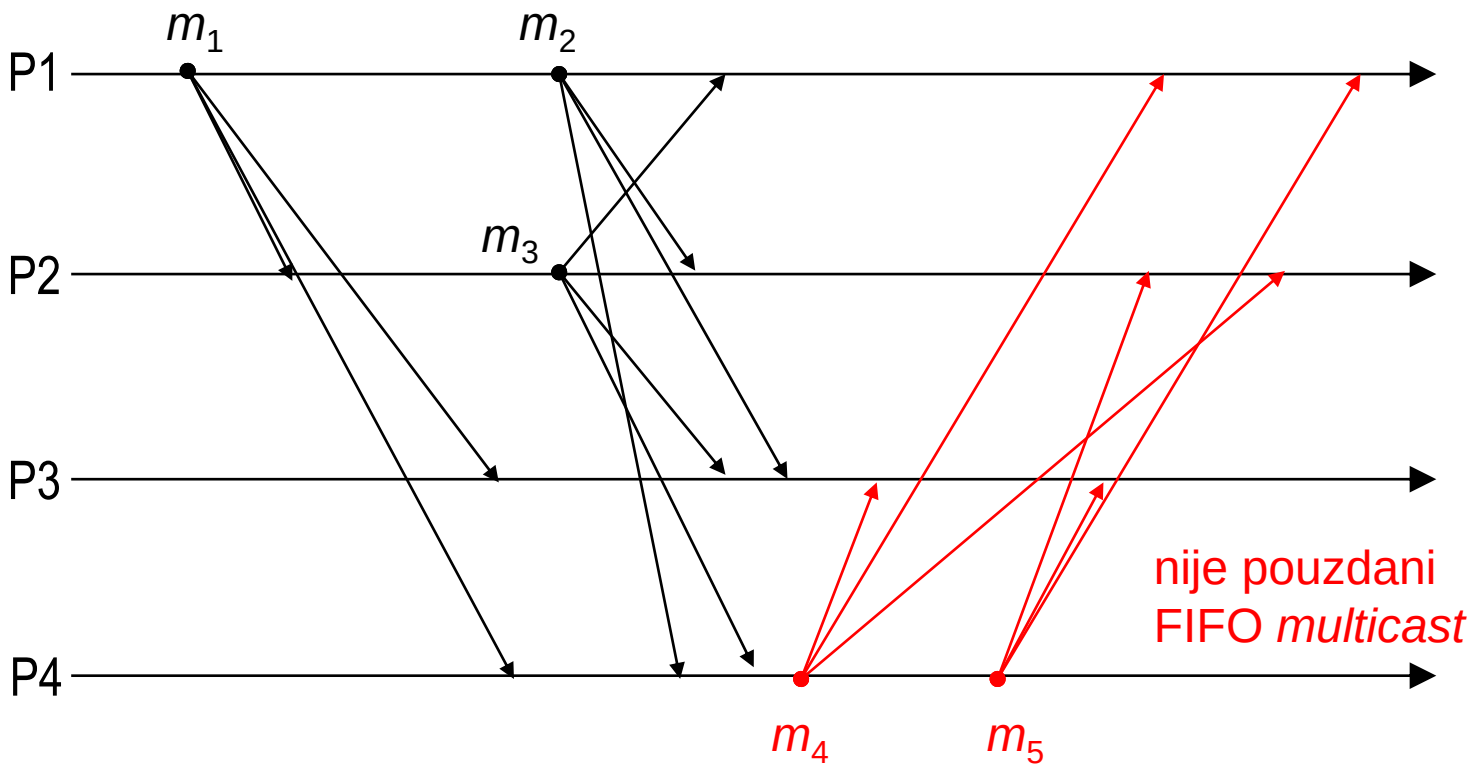
Neuređeni slijed poruka

- Nije važan redoslijed kojim procesi primaju poruke m_1 i m_2.
  - **Pouzdani neuređeni *multicast*** – to je pouzdani *multicast* koji je istovremeno i **virtualno sinkron**.



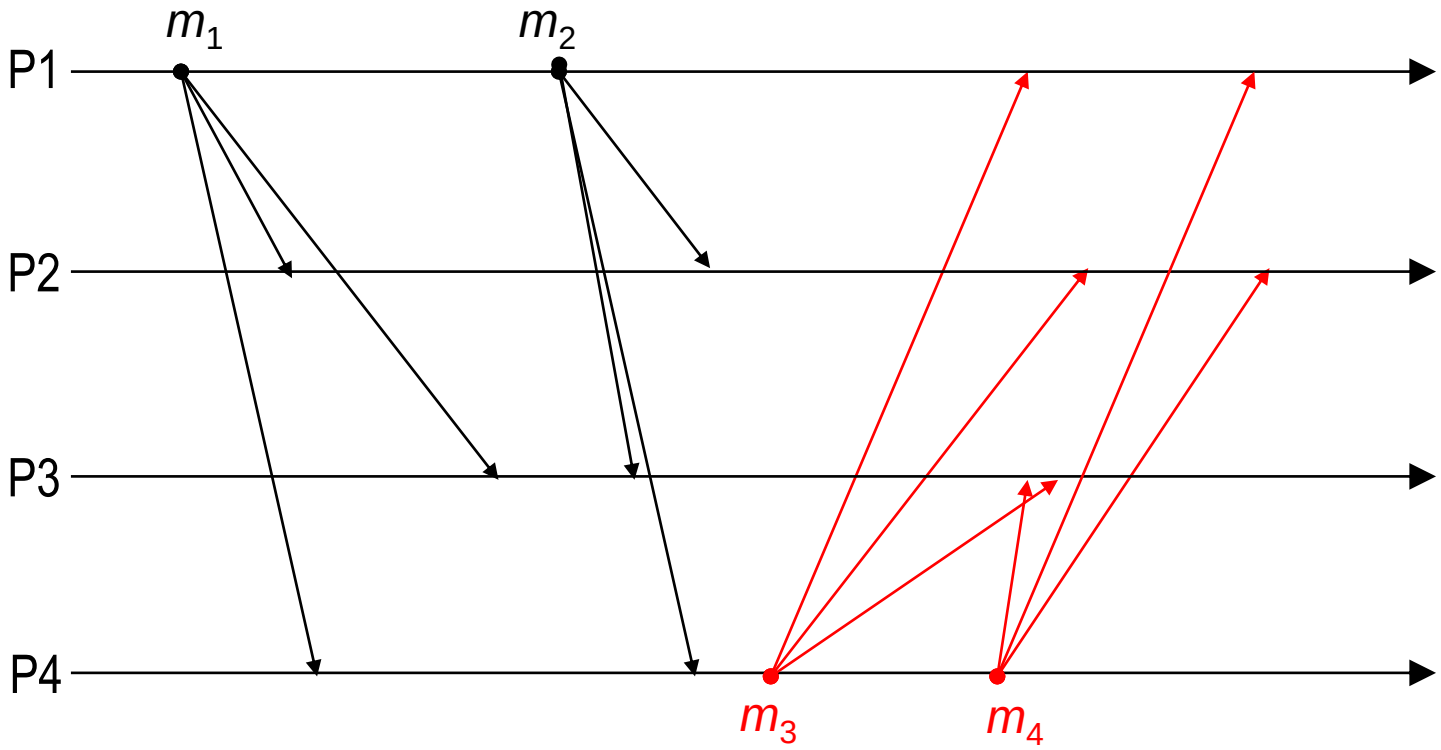
Slijed poruka FIFO

- Poruke *m* koje dolaze **od istog procesa** moraju se isporučiti u redoslijedu kojim su poslane
  - **Pouzdani FIFO *multicast***



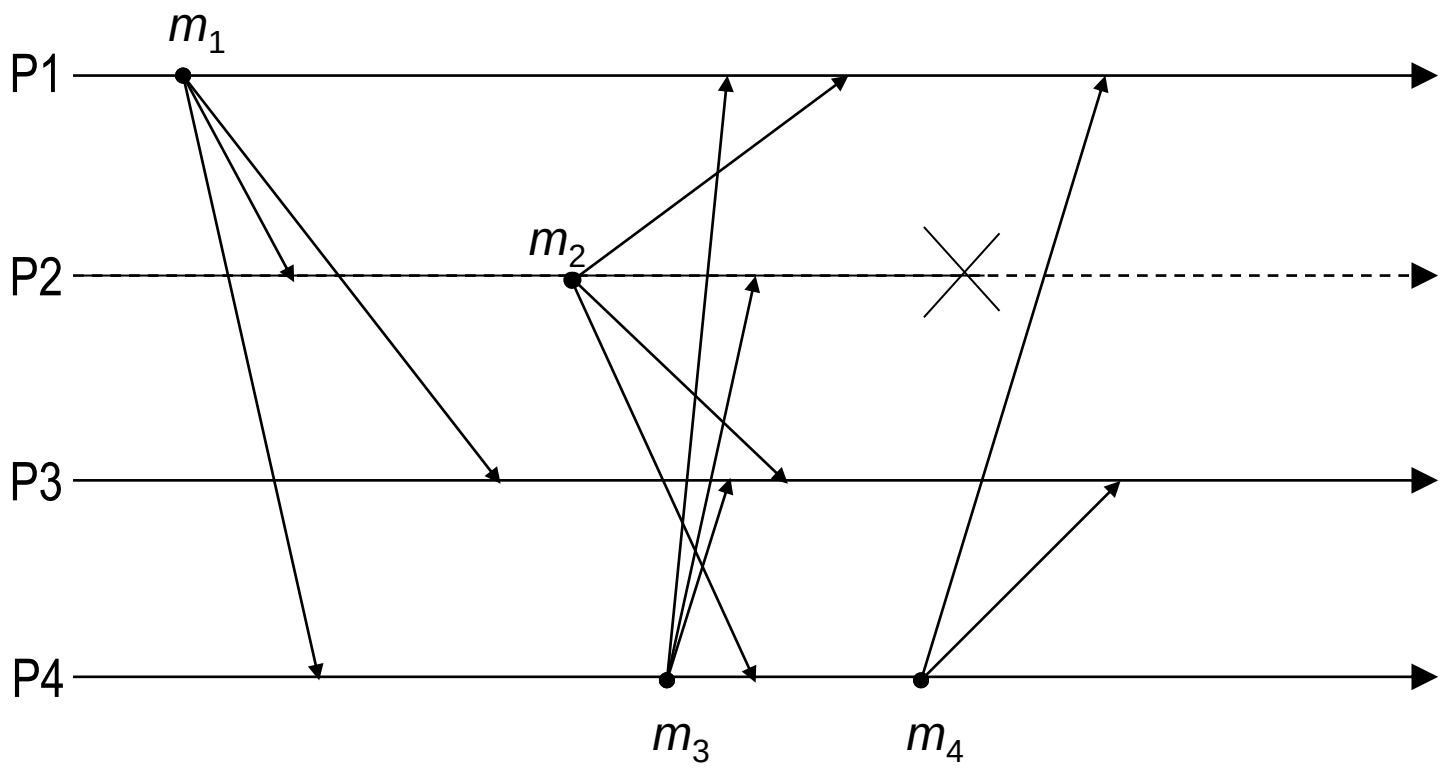
Potpuno uređen slijed poruka

- Poruke se isporučuju u istom redoslijedu svim procesima u skupini (uz FIFO redoslijed poruka od istog procesa)
  - ***Atomic multicast –*** pouzdani virtualno sinkroni multicast s potpuno uređenim slijedom poruka



Potpuno uređen slijed poruka

- Možemo li za sljedeći primjer reći da je ***atomic multicast***?
- Vide li svi ispravni procesi slijed poruka u istom slijedu?



Sadržaj predavanja

- Uvod, definicija pojmova
- Otpornost procesa na ispade
- Sporazum skupine procesa
- Pouzdana komunikacija skupine procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak nakon ispada

Transakcije

- Primjeri: operacije nad bazom podataka, slijed klijentskih zahtjeva za izvođenje operacija nad datotekom, bankarske transakcije
- Svojstva ACID
  - *Atomicity*: izvode se sve operacije unutar jedne transakcije ili niti jedna
  - *Consistency*: izvođenje transakcije dovodi sustav u konzistentno stanje
  - *Isolation*: konkurentne transakcije nemaju utjecaja jedna na drugu
  - *Durability*: nakon završetka transakcije sve su promjene trajne

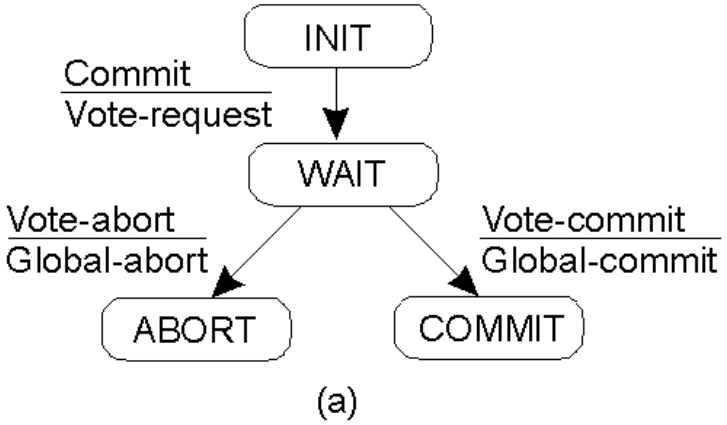
Raspodijeljeno izvršavanje operacije

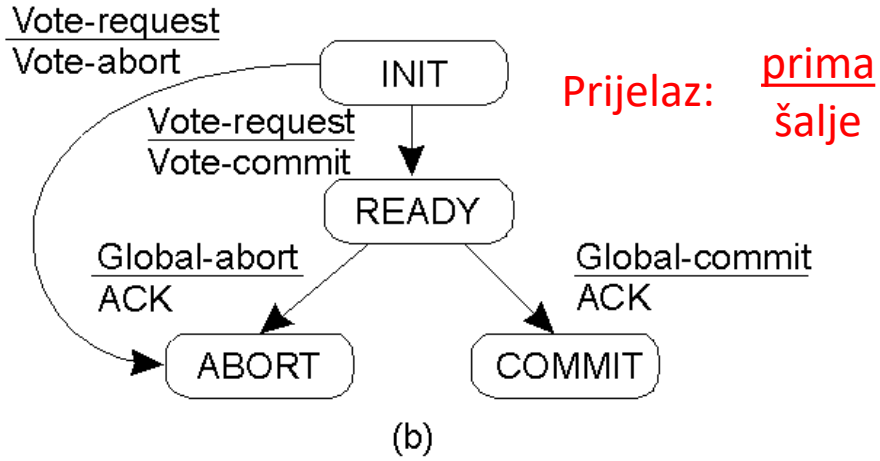
Distributed commit

- operaciju izvodi svaki proces u skupini ili niti jedan
- ako je operacija isporuka poruke, riječ je o *atomic multicast*
- česta primjena za izvođenje raspodijeljenih transakcija
- ideja rješenja - jednofazno izvršavanje (*one-phase commit*)
  - postoji koordinator u skupini procesa
  - koordinator šalje zahtjev ostalim procesima za (lokalno) izvršavanje operacije
  - nedostaci: procesi ne mogu obavijestiti koordinatora u slučaju nemogućnosti izvršavanja operacije, ispad koordinatora

Protokol dvofaznog izvršavanja (1/3)

Two-phase commit protocol (2PC)





Prijelaz: prima / šalje

Automat stanja koordinatora

Automat stanja procesa

U kojim stanjima može doći do blokiranja procesa i koordinatora?

Prema: Tanenbaum, Van Steen

Protokol dvofaznog izvršavanja (2/3)

Problemi

- Postoje stanja blokiranja na strani koordinatora i procesa
  - Proces je blokiran u stanju INIT kada čeka VOTE_REQUEST – ako ne primi poruku nakon određenog vremena, proces može lokalno odustati od izvršavanja operacije
  - Koordinator je blokiran u stanju WAIT kada čeka odgovore svih procesa – ako nakon nekog perioda ne primi odgovor od svih procesa, koordinator može zaključiti da treba odustati od izvršavanja operacije i poslati svim procesima GLOBAL_ABORT
  - Proces je blokiran u stanju READY čekajući konačnu odluku koordinatora – proces treba saznati koju je poruku koordinator poslao i pitati druge procese što se događa!

Protokol dvofaznog izvršavanja (3/3)

Akcije procesa *p* kada se nalazi blokiran u stanju READY i kada kontaktira drugi proces *q* iz skupine procesa G

Stanje procesa *q*	**Akcije procesa *p***
COMMIT	Obavi prijelaz u COMMIT (jer je *q* primio GLOBAL_COMMIT)
ABORT	Obavi prijelaz u ABORT (jer je *q* primio GLOBAL_ABORT)
INIT	Obavi prijelaz u ABORT (jer *q* nije primio VOTE_REQUEST)
READY	Kontaktiraj drugi proces (jer *q* nije kao ni *p* dobio odgovor od koordinatora)

2PC je blokirajući protokol, jer u slučaju ispada koordinatora nakon slanja VOTE_REQUEST, procesi ne mogu zaključiti o sljedećoj operaciji koju trebaju provesti (svi su u stanju READY)!

Algoritam za koordinatora

```
while START _2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
    wait for any incoming vote;
    if timeout {
        while GLOBAL_ABORT to local log;
        multicast  GLOBAL_ABORT to all participants;
        exit;
    }
    record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT{
    write GLOBAL_COMMIT to local log;
    multicast GLOBAL_COMMIT to all participants;
} else {
    write GLOBAL_ABORT  to local log;
    multicast GLOBAL_ABORT to all participants;
}
```

Algoritam za proces

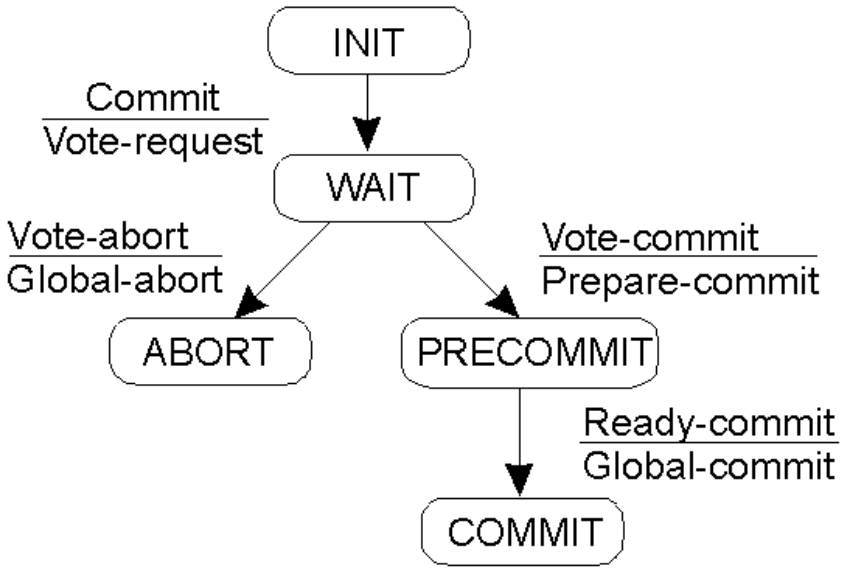
```
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
    write VOTE_ABORT to local log;
    exit;
}
if participant votes COMMIT {
    write VOTE_COMMIT to local log;
    send VOTE_COMMIT to coordinator;
    wait for DECISION from coordinator;
    if timeout {
        multicast DECISION_REQUEST to other participants;
        wait until DECISION is received; /* remain blocked */
        write DECISION to local log;
    }
    if DECISION == GLOBAL_COMMIT
        write GLOBAL_COMMIT to local log;
    else if DECISION == GLOBAL_ABORT
        write GLOBAL_ABORT to local log;
} else {
    write VOTE_ABORT to local log;
    send  VOTE ABORT to coordinator;
}
```

Protokol trofaznog izvršavanja (1/3)

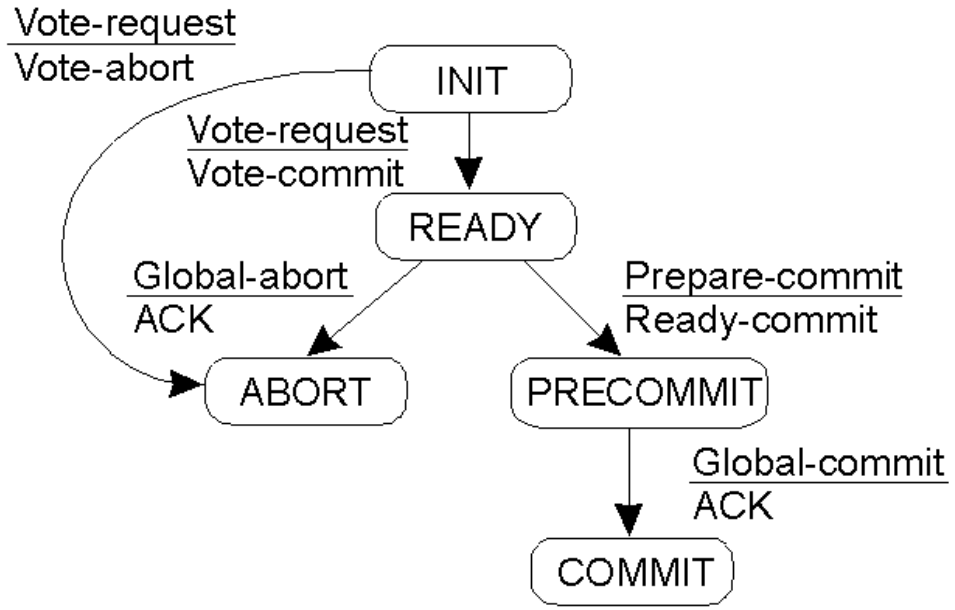
Three-phase commit protocol (3PC)

- rješava problema blokiranja procesa u slučaju ispada koordinatora
- relativno se slabo koristi u praksi, jer se situacija koja dovodi do stanja blokiranja protokola 2PC događa iznimno rijetko

Protokol trofaznog izvršavanja (2/3)



Automat stanja koordinatora



Automat stanja procesa

Nove poruke:
PREPARE_COMMIT
READY_COMMIT

Novo stanje:
PRECOMMIT

Protokol trofaznog izvršavanja (3/3)

- Koordinator može biti blokiran u stanju PRECOMMIT zbog ispada jednog procesa, ali može ostalim procesima poslati GLOBAL_COMMIT
- Proces može biti blokiran u stanjima READY i PRECOMMIT
  - nakon isteka vremenske kontrole zaključuje da je došlo do ispada koordinatora i kontaktira ostale procese
  - ako je *q* u stanju COMMIT i *p* prelazi u COMMIT
  - ako je *q* u stanju ABORT i *p* prelazi u ABORT
  - ako su svi procesi (ili većina) u stanju PRECOMMIT, mogu svi preći u COMMIT
  - ako je *q* u INIT, *p* prelazi u ABORT
  - ako su svi procesi u stanju READY, mogu svi preći u ABORT

Sadržaj predavanja

- Uvod, definicija pojmova
- Otpornost procesa na ispade
- Sporazum skupine procesa
- Pouzdana komunikacija skupine procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak nakon ispada

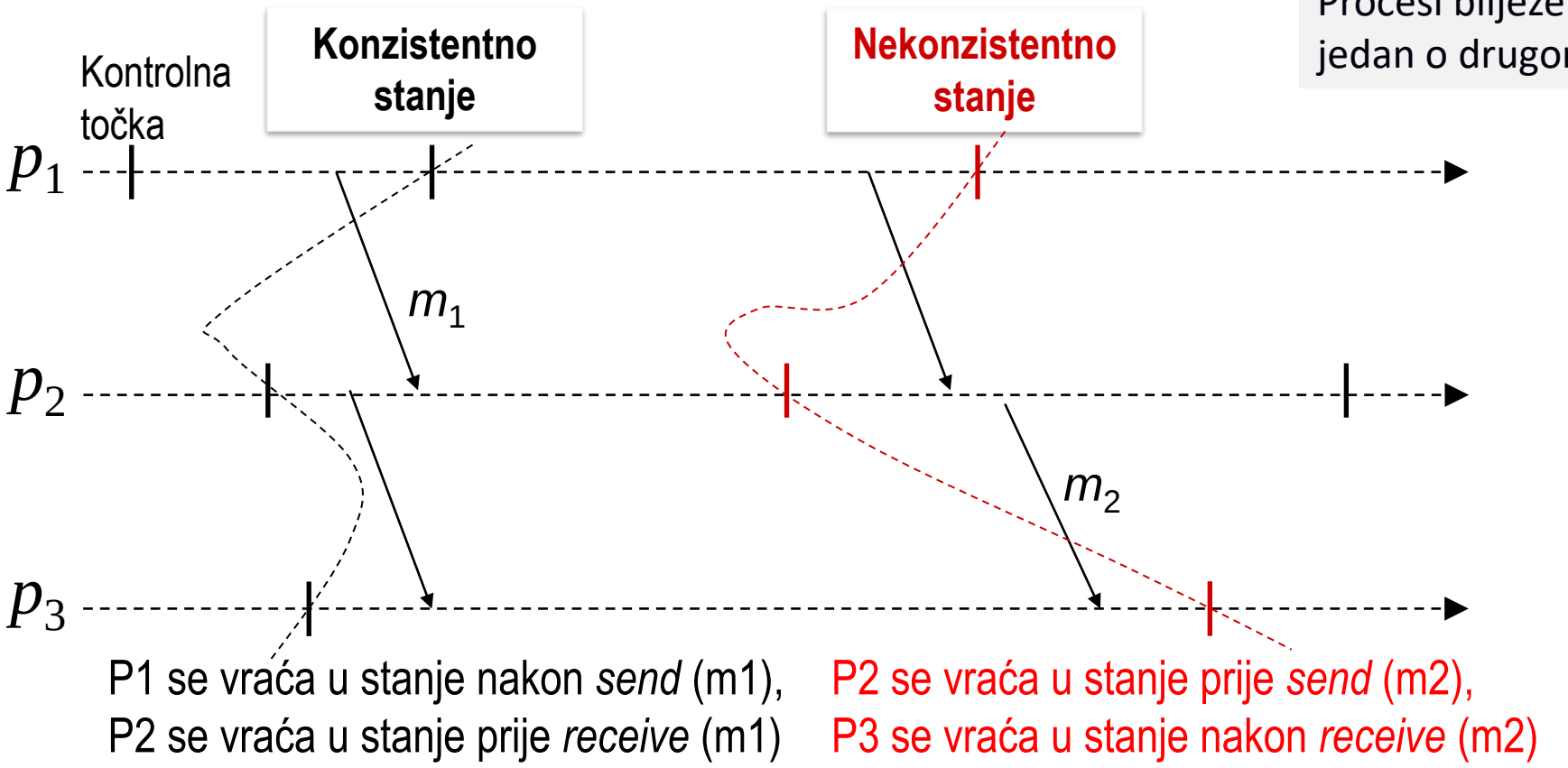
Oporavak nakon ispada

- Nakon ispada procesa nužan je njegov oporavak i povratak u ispravno stanje
- Oporavak unazad
  - vratiti sustav u ispravno stanje u prošlosti
  - potrebno je s vremena na vrijeme pohraniti stanje sustava (kontrolne točke, *checkpoint*)
  - zapisuju se poslane i primljene poruke u dnevnički zapis (*log*)
- Oporavak korištenjem dnevničkog zapisa
  - proces u ispadu vraća se u prethodno ispravno stanje nakon čega izvodi akcije iz dnevničkog zapisa

Bilježenje kontrolnih točaka

- Procesi mogu kontrolne točke bilježiti neovisno ili koordinirano
- **neovisno** zahtjeva bilježenje ovisnosti među procesima koji razmjenjuju poruku (šalju i primaju istu poruku) radi povratka u konzistentna stanja – pokazano na sljedećem primjeru
- **koordinirano** koristi koordinatora i jednostavnije je za implementaciju, koordinator šalje naredbu procesima da pohrane stanje sustava gotovo istovremeno (prije toga moraju primiti poruke u tranzitu i privremeno zaustaviti pripremljena slanja poruka)

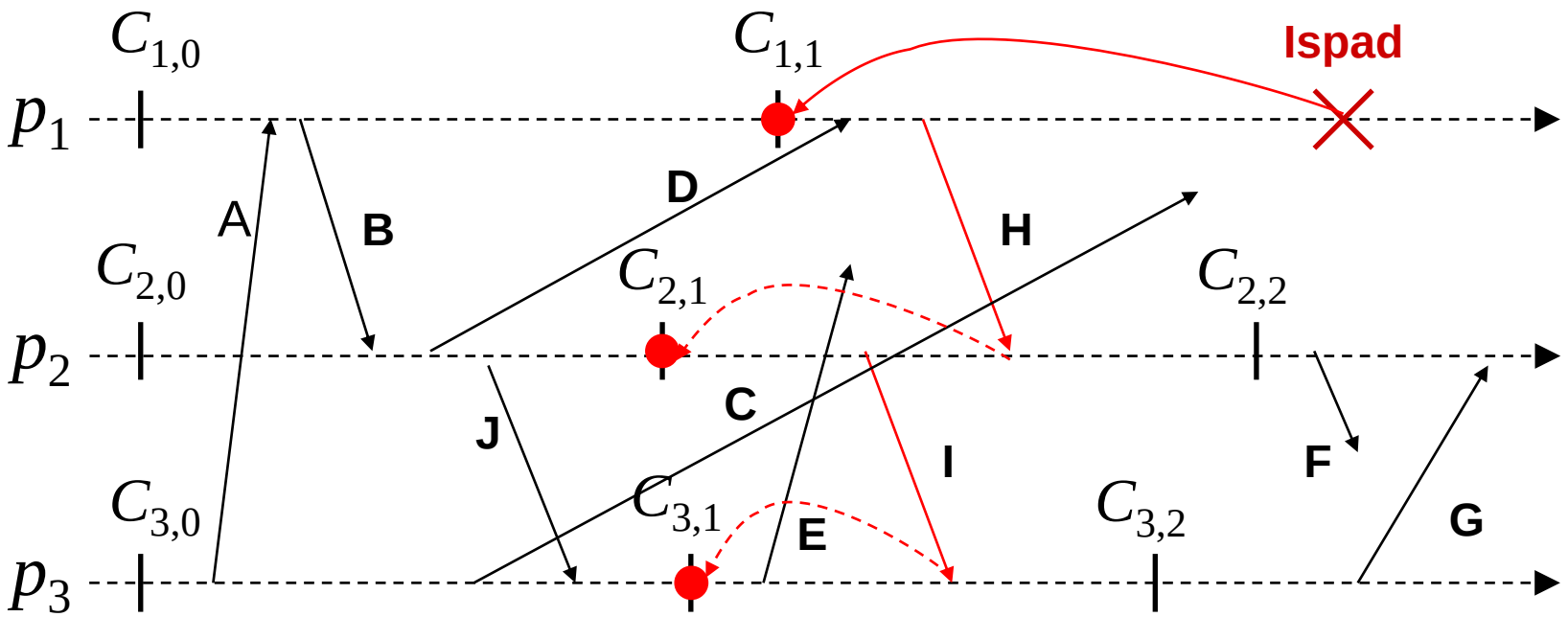
Konzistentna i nekonzistentna stanja



Procesi bilježe kontrolne točke neovisno jedan o drugom!

Ako se proces P1 vraća u stanje prije send(*m*), onda se P2 mora vratiti u stanje prije receive(*m*)!

Primjer oporavka povratkom unazad



Konzistentno stanje sustava:

$C_{1,1}, C_{2,1}, C_{3,1}$

A, B, J: normalne poruke (poslane i primljene)

C: zakašnjela poruka (može stići prije, tijekom ili nakon oporavka)

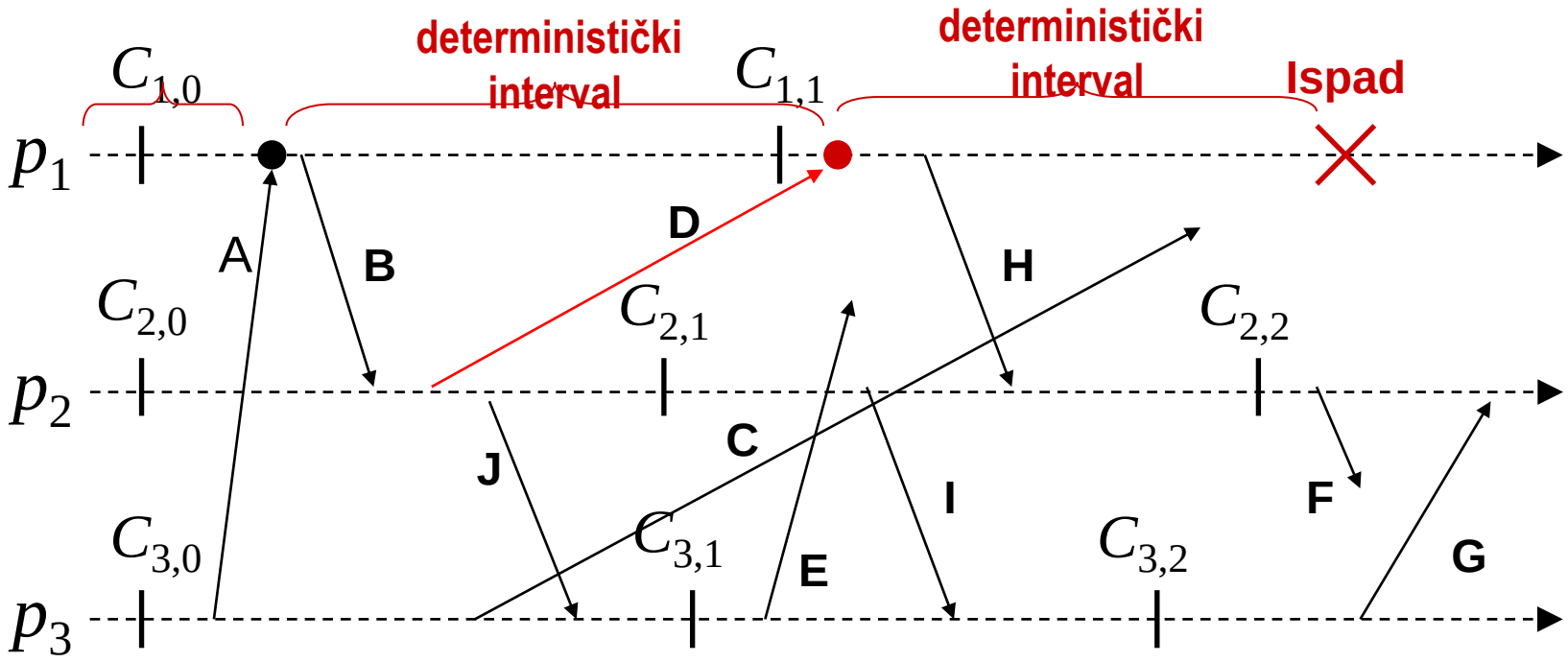
D: izgubljena poruka (p_2 poslao, p_1 nije zabilježio prijam)

G, H, I: poništene poruke, ponovit će se slanje i primanje

E i F: zakašnjele suvišne poruke (ako/kad stignu na odredište treba ih odbaciti, jer će se ponoviti njihovo slanje i primanje)

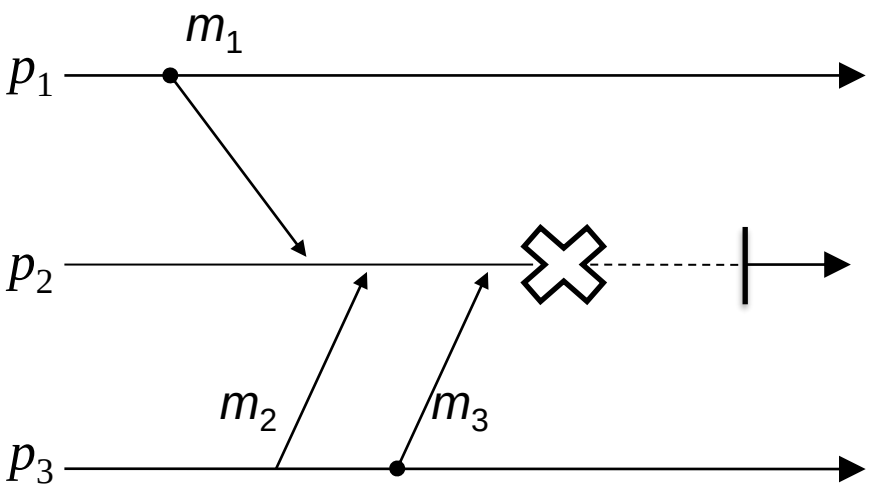
Oporavak korištenjem dnevničkog zapisa

- Koristi kombinirano kontrolne točke i dnevnički zapis
- Deterministički interval: počinje i završava nedeterminističkim događajem (prijam poruke) koji se zapisuje u dnevnički zapis



Zapisivanje poruka u dnevnički zapis

- Pravilo: Sve nedeterminističke događaje tj. primitak poruke s pripadnim informacijama (pošiljatelj, primatelj, redni broj poruke) treba bilježiti u dnevnik



Primjer neispravnog zapisa u dnevnički zapis

Proces p_2 prima m_1, m_2, i m_3, ali m_2 ne zapisuje u dnevnički zapis.

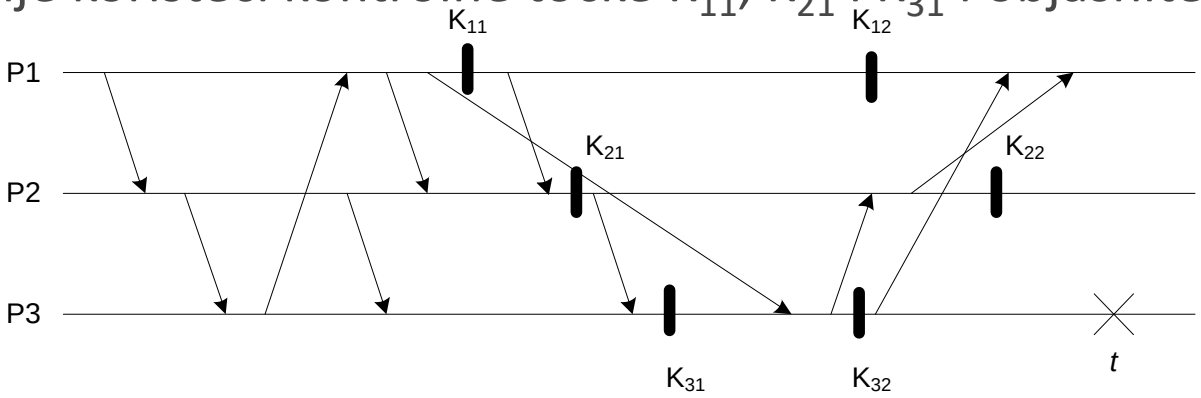
Što se događa nakon ispada i oporavka procesa p_2?

p_2 čita i rekonstruira m_1 iz zapisa
p_2 nema m_2 u zapisu i ne može rekonstruirati tu poruku, (može rekonstruirati samo primanje poruke m_3) =>

p_2 se ne može vratiti u ispravno stanje neposredno prije ispada

Pitanja za učenje i ponavljanje

- Pretpostavite da postoji skupina od *m* jednakih procesa. Navedite toleranciju ove skupine procesa na ispade ako pretpostavite „obični” ispad ili bizantski ispad.
- Identificirajte i objasnite blokirajuća stanja protokola 2PC.
- Slika prikazuje tri procesa i razmjenu poruka među njima. Svaki proces neovisno o drugim procesima bilježi svoja stanja u označenim kontrolnim točkama. U trenutku *t* dolazi do ispada procesa P3. Možemo li sustav od tri procesa na slici nakon ispada procesa P3 vratiti u konzistentno stanje koristeći kontrolne točke K_{11}, K_{21} i K_{31} i objasnite zašto je to moguće ili nije moguće?



Literatura

- A. S. Tanenbaum, M. Van Steen: Distributed Systems: Principles and Paradigms, Second Edition, Prentice Hall, 2007. (poglavlje 8, *Fault tolerance*)
- Ajay D. Kshemkalyani, Mukesh Singhal, Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press, 2008.
- Rachid Guerraoui, André Schiper, "Software-Based Replication for Fault Tolerance," Computer, vol. 30, no. 4, 68-74, Apr., 1997.
- Lamport, L., Shostak, R., and Pease, M. “The Byzantine Generals Problem,” *ACM Trans. Program. Lang. Syst.,* vol. 4, no. 3, 382-401, Jul. 1982.
- Défago, X., Schiper, A., and Urbán, P. “Total order broadcast and multicast algorithms: Taxonomy and survey,” *ACM Comput. Surv.* vol. 36, no. 4, 372-421, Dec. 2004.