Diplomski studij

Informacijska i komunikacijska tehnologija:

Telekomunikacije i informatika

Računarstvo:

Programsko inženjerstvo i informacijski sustavi

Računarska znanost

Ak.god. 2008./2009.

Raspodijeljeni sustavi

11.

Otpornost na neispravnosti

Creative Commons

Zavod za telekomunikacije

- **slobodno smijete:**
  - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **remiksirati** — 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. Najbolji način da to učinite je linkom na ovu internetsku stranicu.
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 licencije preuzet je s http://creativecommons.org/.

Sadržaj predavanja

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

Pripremljeno na temelju poglavlja 8, *Fault tolerance* iz

A. S. Tanenbaum, M. Van Steen: Distributed Systems: Principles and Paradigms, Second Edition, Prentice Hall, 2007.

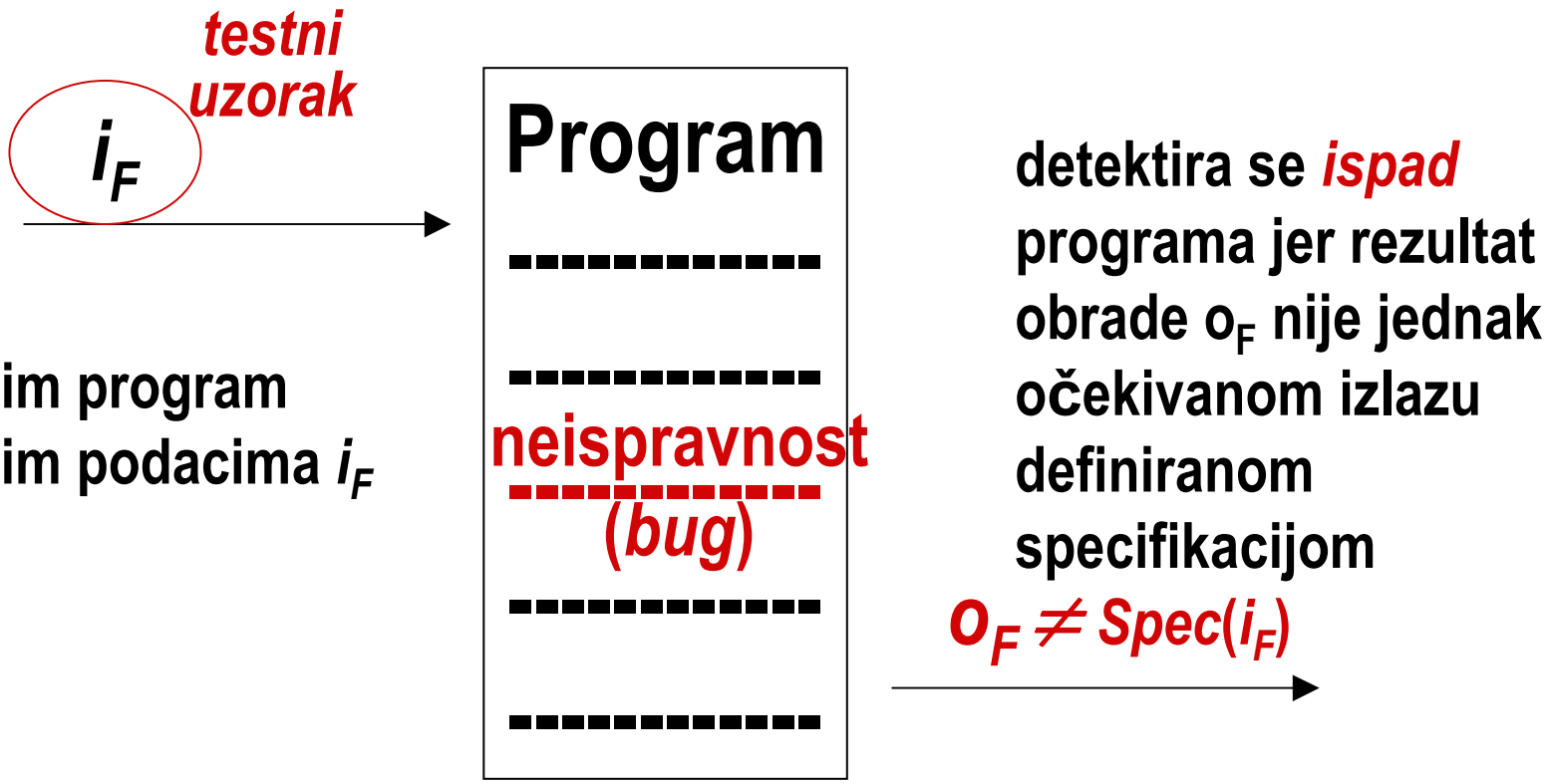
Otpornost na neispravnosti

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 performanse

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 testiranja programskog koda…

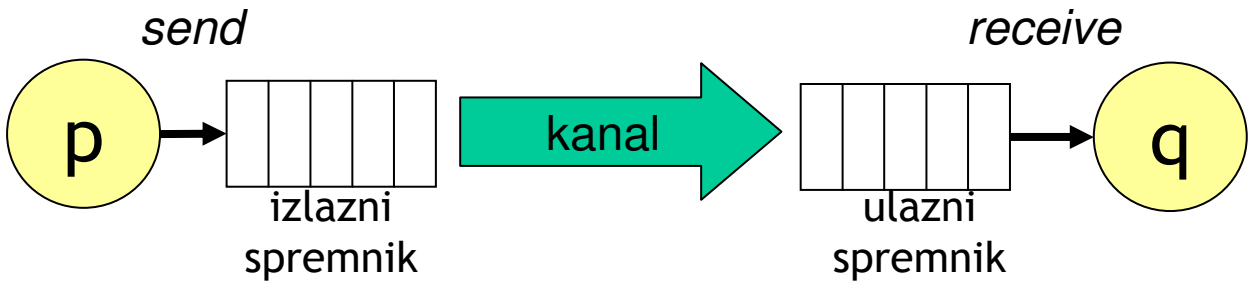


Osnovni pojmovi

- Ispad sustava (*failure*)
  - stanje sustava koje se detektira kroz nemogućnost korištenja jedne ili više njegovih usluga
  - posljedica neispravnosti, signalizira postojanje neispravnosti u distribuiranom sustavu
- 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

Vrste ispada

Osnovni model: 2 procesa povezana komunikacijskim kanalom, procesi komuniciraju koristeći operatore *send* i *receive*



- Ispad procesa
  - proces ne mijenja stanja (ne izvode se prijelazi) premda se ne nalazi u završnom stanju
  - narušeno svojstvo životnosti (*liveness*)
- Ispad kanala
  - proces *p* je poslao poruku procesu *q*, ali *q* poruku ne prima
  - narušeno svojstvo životnosti (*liveness*) i sigurnosti (*safety*)

Vrste ispada poslužitelja za model klijent-poslužitelj

Vrsta ispada	**Opis**
Ispad poslužitelja	Poslužitelj neočekivano ulazi u stanje zaustavljanja i ne odgovara na nove zahtjeve.
Pogreška u komunikaciji *pogreška primanja* *pogreška slanja*	Poslužitelj ne odgovara na primljeni zahtjev. Poslužitelj ne prima zahtjev. Poslužitelj ne šalje odgovor.
Vremenska pogreška	Odgovor je poslan 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”	Poslužitelj može proizvesti proizvoljan odgovor u proizvoljnom trenutku.

Redundancija

Ključna tehnika za prikrivanje neispravnosti distribuiranog sustava.

Primjeri redundacije:

- 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, definicija pojmova
- Otpornost procesa na ispade
- Pouzdana komunikacija grupe procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak

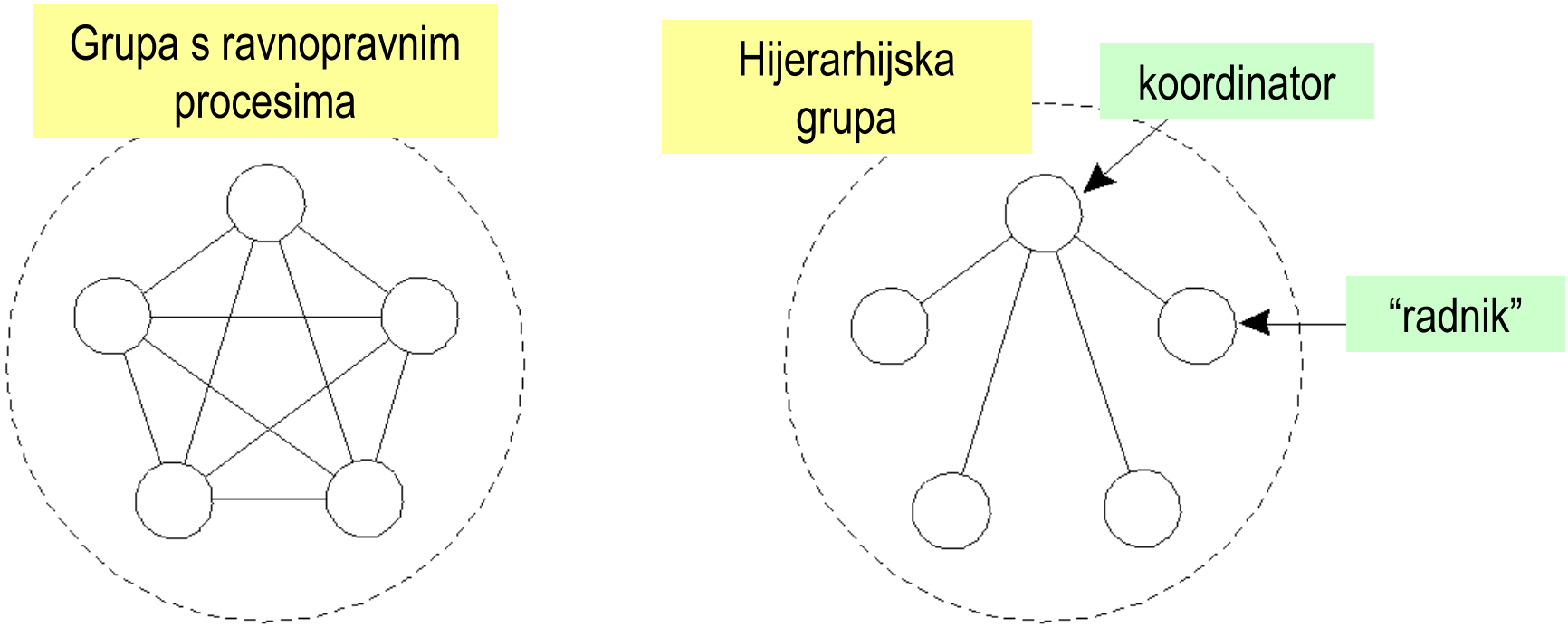
Grupa procesa

- replikacija procesa radi prikrivanja ispada
- **tolerancija *k* ispada**: grupa može "preživjeti" ispad najviše *k* procesa
  - dovoljan je ***k* + 1** proces da se osigura tolerancija *k* ispada, jedan proces može preuzeti poslove grupe
  - ako pretpostavimo *k* bizantskih ispada, potrebno je **2*k* + 1** procesa (*k* + 1 proces će "nadglasati" *k* neispravnih)
- osigurati isporuku poruke svim članovima grupe
- grupe procesa su dinamične (slično sustavima P2P), jedan proces može biti član više grupa

Otkrivanje ispada procesa

- 2 osnovne metode:
  - 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 grupe

Organizacija grupe



Administriranje grupe procesa

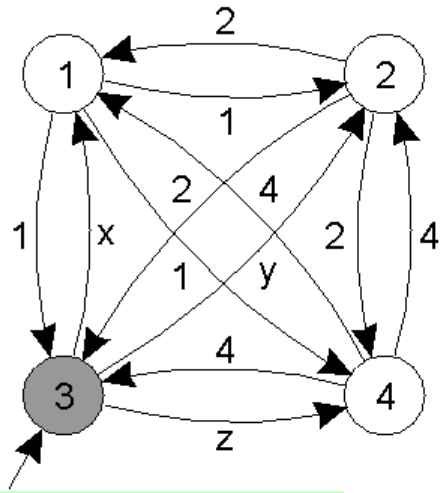
- usluga koja omogućuje kreiranje grupe, dodavanje procesa u grupu i izlazak procesa iz grupe

Sporazum grupe procesa

Byzantine agreement problem [Lamport et al. 1982]

- Problem: grupa procesa treba postići sporazum o neovisnim vrijednostima za svaki proces
- Pretpostavke:
  - *k* neispravnih procesa (**bizantski ispad**)
  - *N* procesa
  - proces *i* šalje vrijednost v_i ostalim procesima u grupi
  - svaki proces treba kreirati vektor **V** duljine *N* takav da vrijedi **V**[i] = v_i

Primjer (*Byzantine agreement problem*)



neispravan proces

1. korak:
Ispravni procesi 1, 2 i 4 šalju svoj identifikator ostalim procesima, dok 3 šalje proizvoljne vrijednosti.

***N* = 4**
***k* = 1**

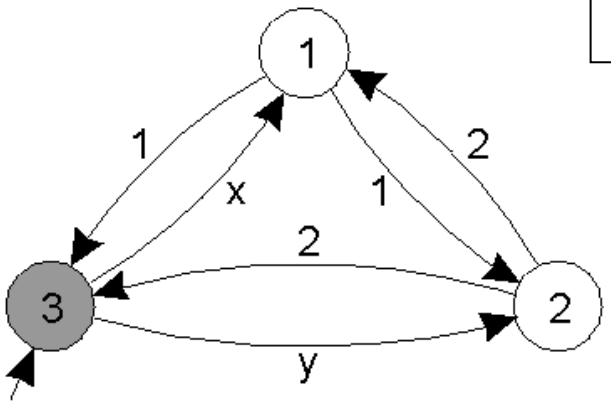
1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

2. korak:
Svi procesi prikupljaju podatke i spremaju ih u vektor **V** (svaki proces ima vlastiti vektor).

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

3. korak:
Svaki proces šalje ostalima svoj vektor (3 ponovo šalje proizvoljne vrijednosti). Konačno svaki proces uspoređuje sve primljene vektore i odlučuje se za većinsku vrijednost **(1, 2, ?, 4)**.

Primjer (*Byzantine agreement problem*)



***N* = 3**
***k* = 1**

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

neispravan proces

Mogu li procesi donijeti odluku za prikazani slučaj?

Za sporazum u slučaju *k* neispravnih procesa, potrebno $2k + 1$ ispravnih, tj. ukupno $N = 3k + 1$ procesa!

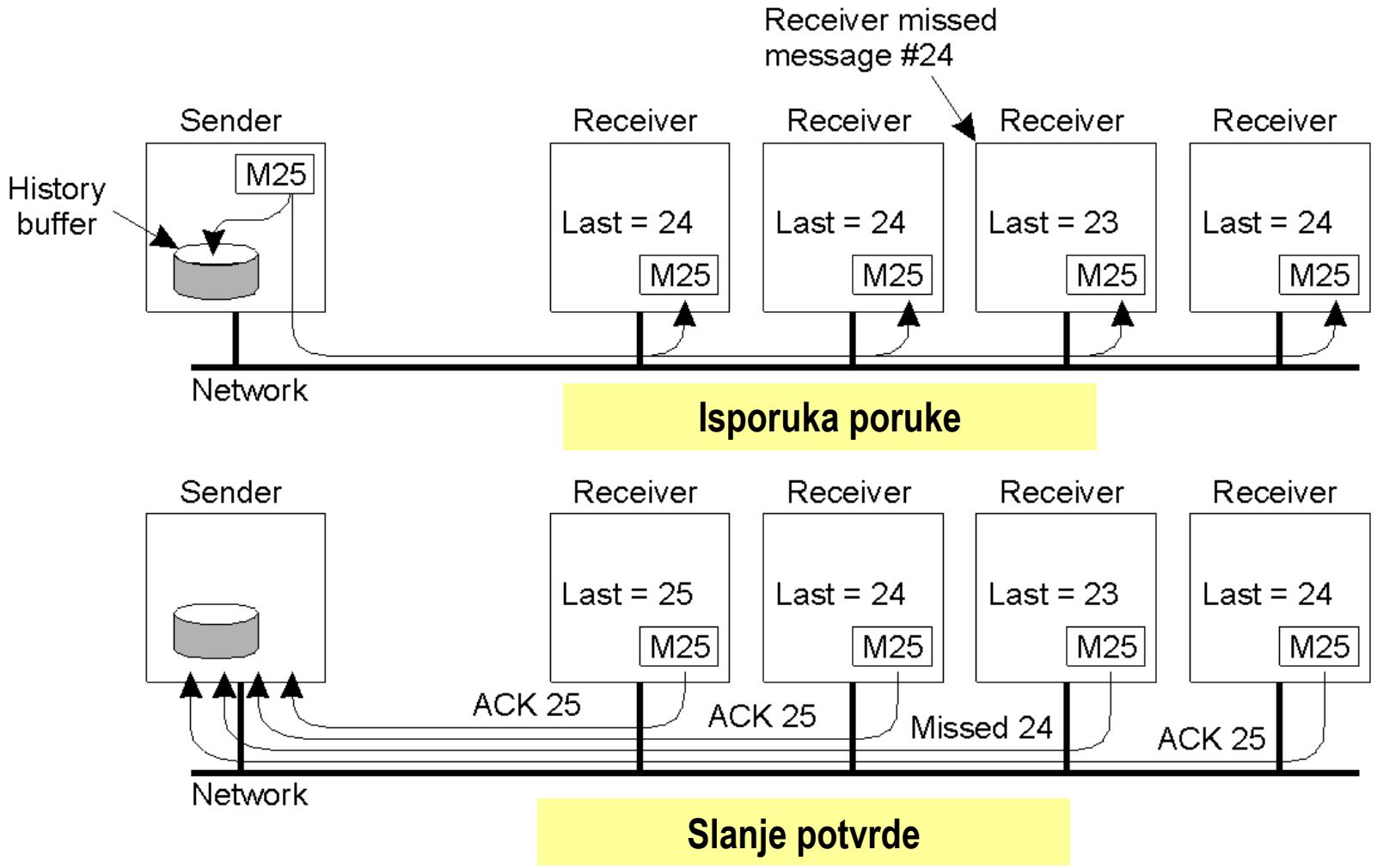
Sadržaj predavanja

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

Pouzdana komunikacija grupe procesa

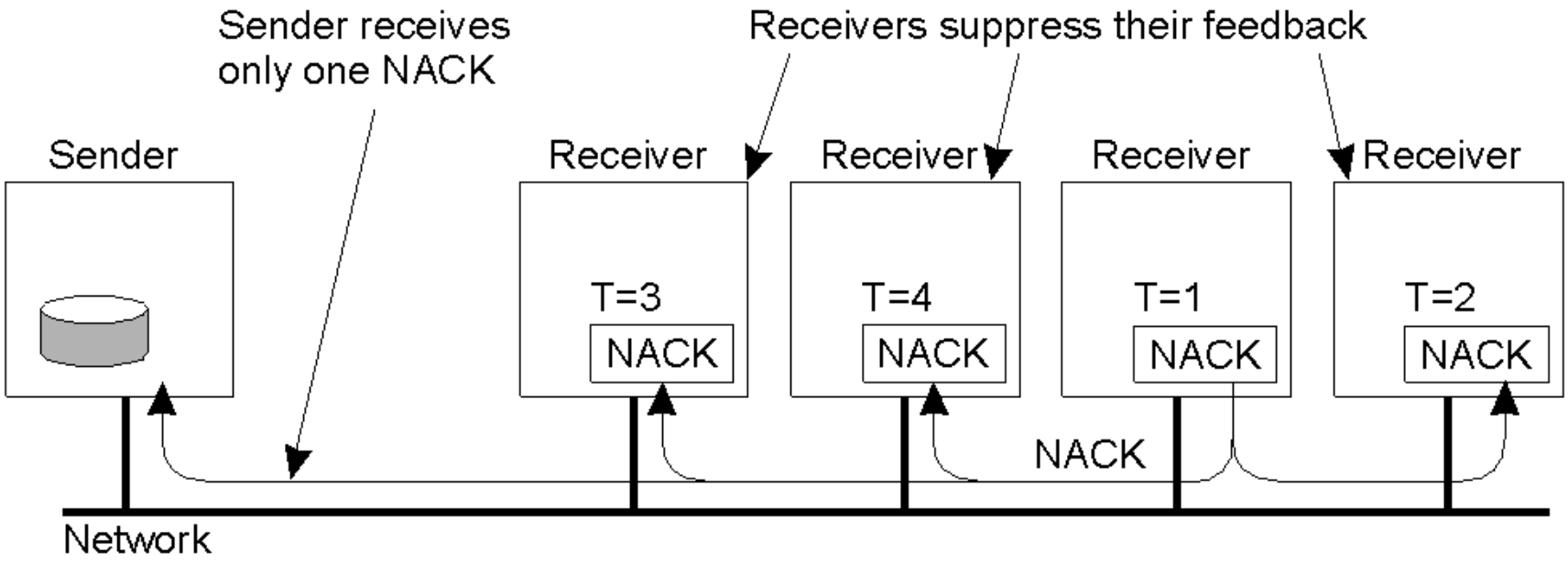
- garantira isporuku poruke svim procesima u grupi
- problemi
  - Koji procesi čine grupu u trenutku slanja poruke?
  - Što se događa ako novi proces ulazi u grupu procesa dok je isporuka poruke grupi u tijeku?
  - Što se događa ako se dogodi ispad pošiljatelja poruke tijekom isporuke poruke ostalim procesima?
  - Što se događa ako jedan od primatelja ispadne tijekom isporuke poruke?
- najjednostavnija praktična implementacija
  - pouzdana komunikacija od točke do točke (npr. TCP) između svakog para procesa iz grupe

Pouzdana komunikacija bez mogućih ispada procesa (1)



Pouzdana komunikacija bez mogućih ispada procesa (2)

Potisnuta potvrda



Pouzdana komunikacija s ispadima procesa

- garantira isporuku poruke svim ispravnim i dostupnim procesima u grupi ili niti jednom
- potrebno je osigurati i isporuku poruka u određenom slijedu

Notacija

- p - proces
- G – grupa, skup procesa (*group view*)
- m – generirana poruka
- vc – poruka koja prenosi informaciju o dolasku ili odlasku procesa iz grupe (*view change*)

Virtualna sinkronost

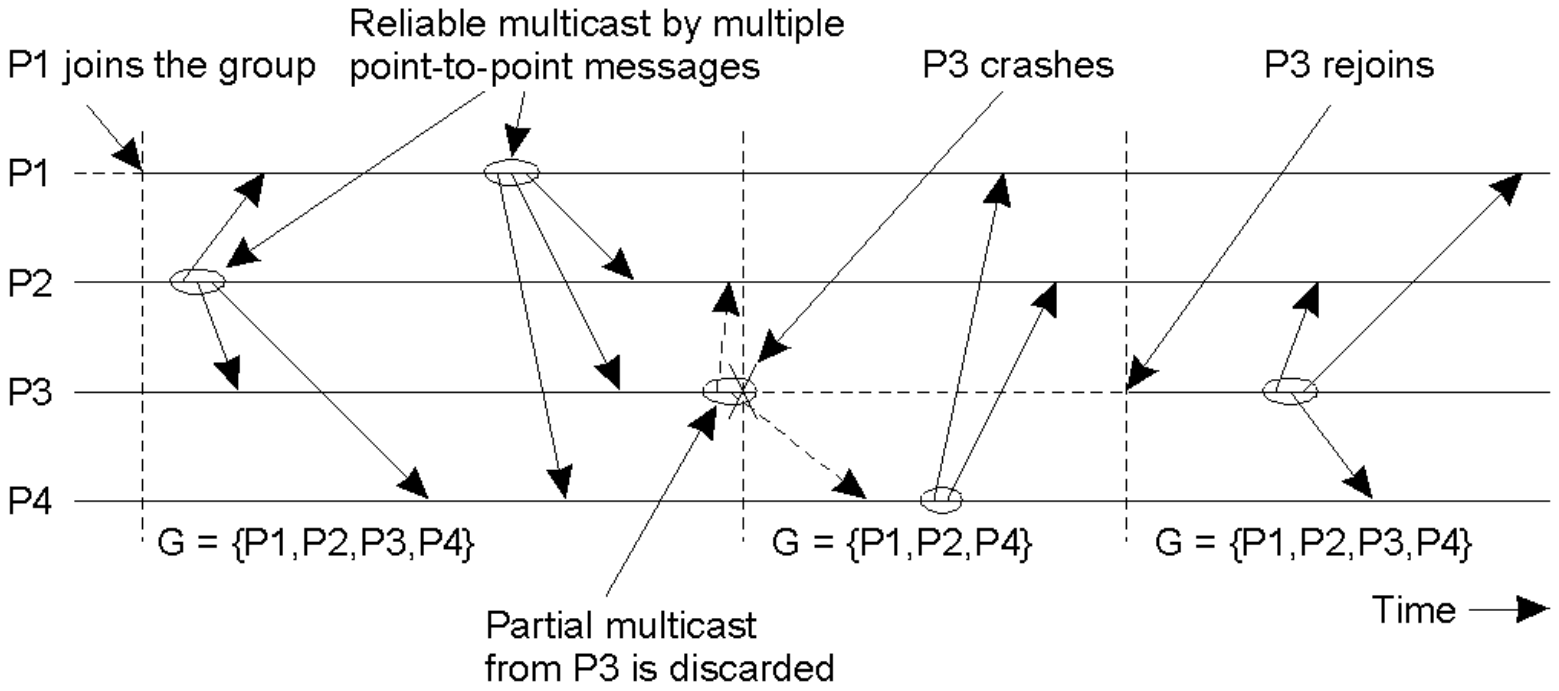
Scenarij 1

- Proces p šalje poruku m, u tome trenutku postoji grupa procesa G
- Tijekom isporuke m novi proces se uključuje u grupu i generira se poruka vc (view change) koja se opet šalje svim G
- Posljedica: p 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 iz G, ostali procesi ignoriraju m (ne treba osigurati isporuku ostalim ispravnim procesima, kao da je p ispao prije slanja m)

Primjer virtualne sinkronosti



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

Implementacija nije trivijalna.

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

Process P1	Process P2	Process P3
sends m1	receives m1	receives m2
sends m2	receives m2	receives m1

Nije važan redoslijed kojim procesi primaju poruke.

Pouzdani neuređeni multicast – to je pouzdani multicast koji je istovremeno i **virtualno sinkron**.

Slijed poruka FIFO

Process P1	Process P2	Process P3	Process P4
sends m1	receives m1	receives m3	sends m3
sends m2	receives m3	receives m1	sends m4
	receives m2	receives m2	
	receives m4	receives m4	

Poruke koje dolaze od istog procesa moraju se isporučiti u redoslijedu kojim su poslane, npr. kako P1 šalje prvo m1 a zatim m2, m1 mora biti isporučen prije m2. Isto vrijedi za m3 i m4. Procesi P2 i P3 primaju poruke u različitom slijedu, važno je poštivati redoslijed isporuke poruka koje dolaze od istog procesa.

Pouzdani FIFO multicast

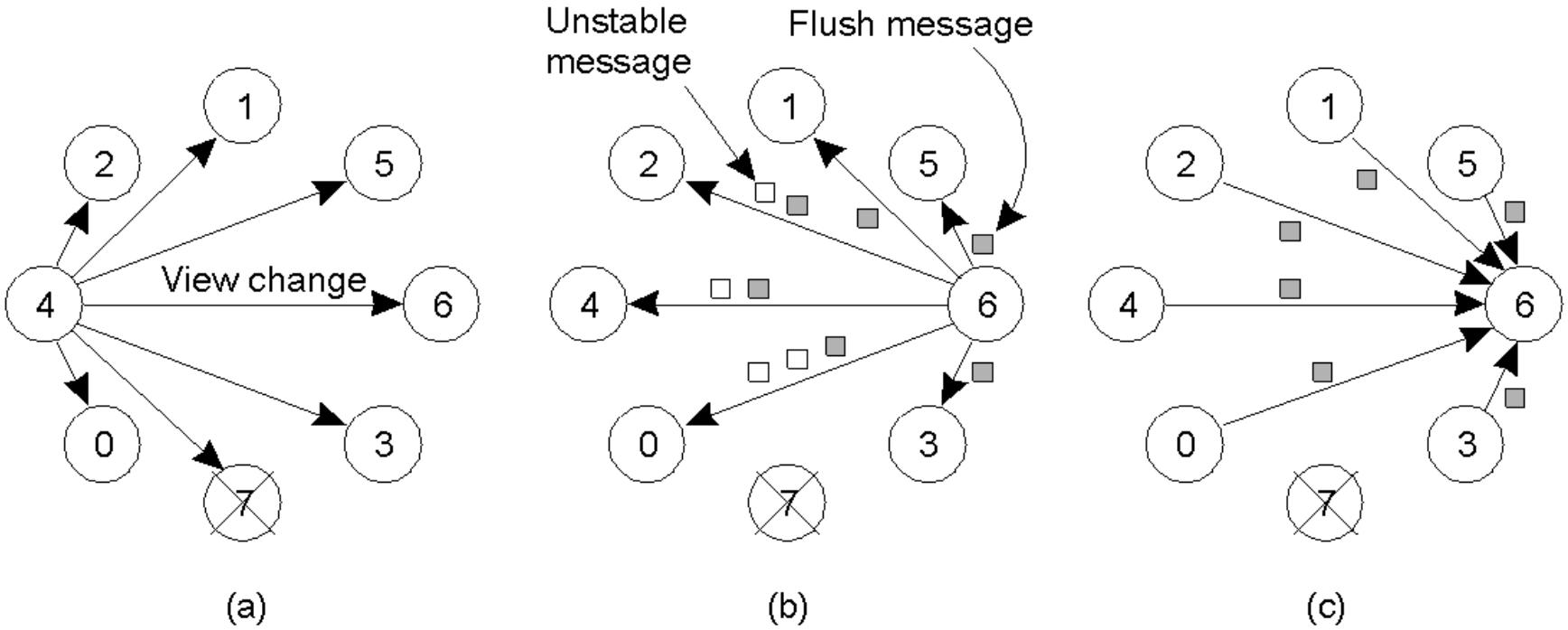
Potpuno uređen slijed poruka

Process P1	Process P2	Process P3	Process P4
sends m1	receives m1	receives m1	sends m3
sends m2	receives m3	receives m3	sends m4
	receives m2	receives m2	
	receives m4	receives m4	

Poruke se isporučuju u istom redoslijedu svim procesima u grupi (treba uzeti u obzir i FIFO redoslijed poruka koje dolaze od istog procesa, što znači da npr. m2 ne može biti isporučen prije m1).

Atomic multicast – pouzdani virtualno sinkroni multicast s potpuno uređenim slijedom poruka

Primjer implementacije: Isis



a) Proces 4 opaža ispad procesa 7 i šalje poruku vc.

b) Proces 6 šalje sve “nestabilne poruke” poruke, a nakon toga posebnu poruku *flush*.

c) Proces 6 kreira novi skup G kada prima poruku flush od svih ostalih procesa.

Sadržaj predavanja

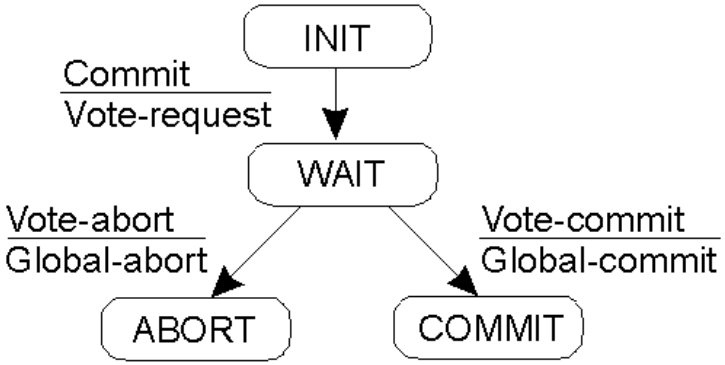
- Uvod, definicija pojmova
- Otpornost procesa na ispade
- Pouzdana komunikacija grupe procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak

Raspodijeljeno izvršavanje operacije

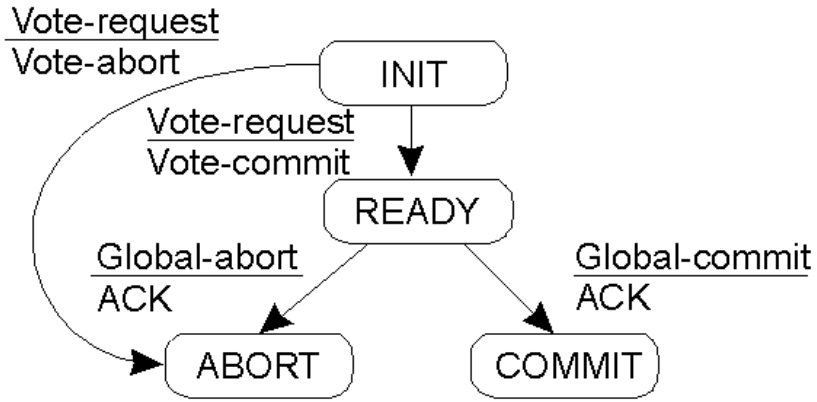
Distributed commit

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

Protokol *two-phase commit* (1)



Automat stanja koordinatora



Automat stanja procesa

Protokol *two-phase commit* (2)

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
  - Proces je blokiran u stanju READY čekajući konačnu odluku koordinatora – proces treba saznati koju je poruku koordinator poslao!

Protokol *two-phase commit* (3)

Stanje procesa Q	Akcije procesa P
COMMIT	Obavi prijelaz u COMMIT
ABORT	Obavi prijelaz u ABORT
INIT	Obavi prijelaz u ABORT
READY	Kontaktiraj drugi proces

Akcije procesa P kada se nalazi blokiran u stanju READY i kada kontaktira drugi proces Q iz G

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;
}
```

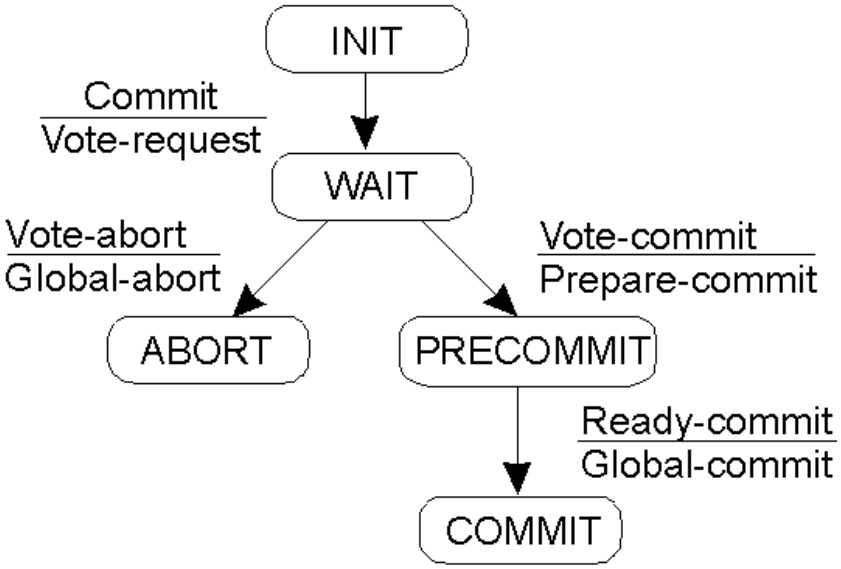
Algoritam za proces


```
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;
}
```

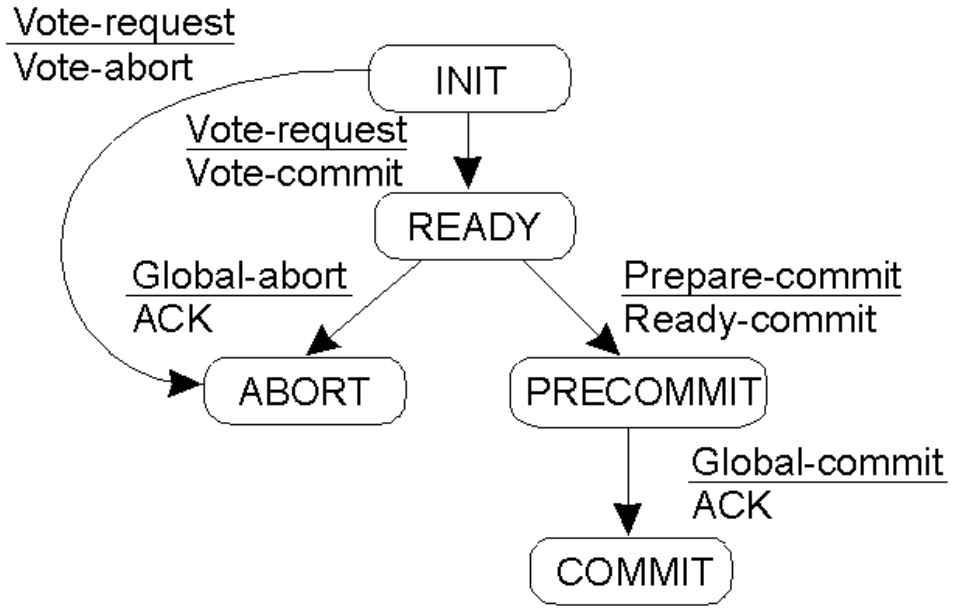
Protokol *three-phase commit* (1)

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

Protokol *three-phase commit* (2)



Automat stanja koordinatora



Automat stanja procesa

Protokol *three-phase commit* (3)

- Koordinator može biti blokiran u stanju PRECOMMIT zbog ispada jednog procesa, ali može ostalim procesima poslati GOBAL_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

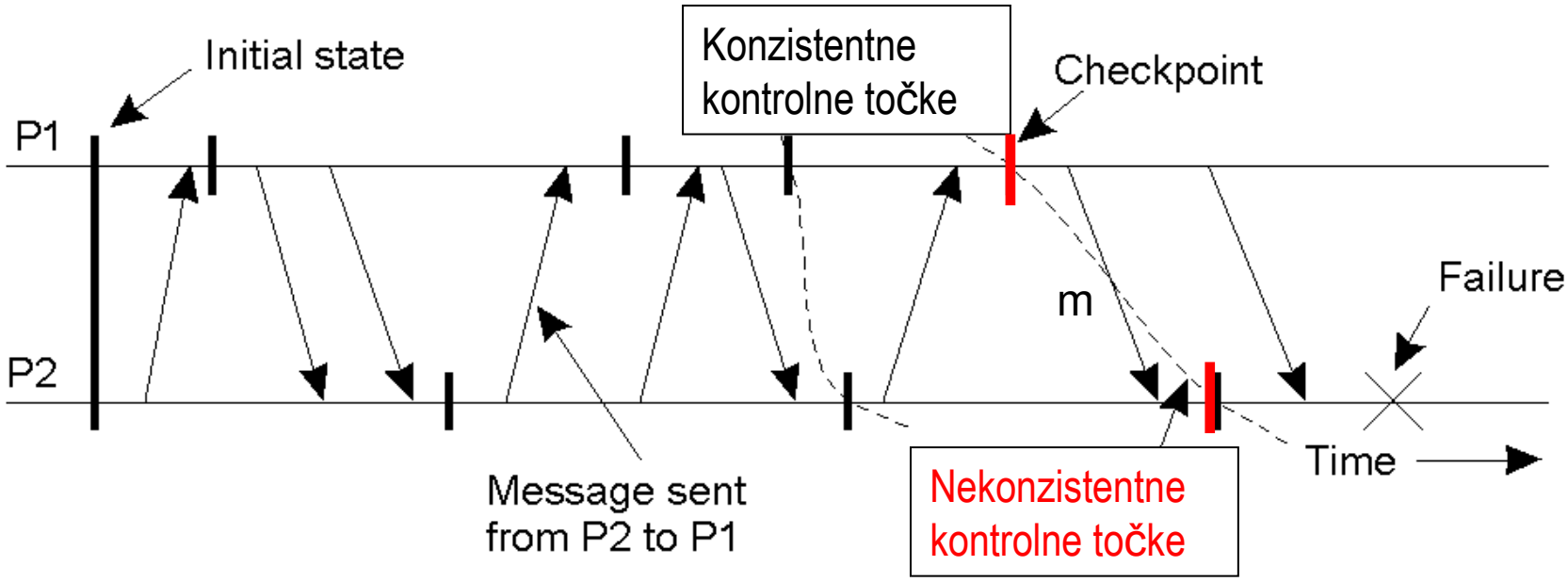
- Uvod, definicija pojmova
- Otpornost procesa na ispade
- Pouzdana komunikacija grupe procesa
- Raspodijeljeno izvršavanje operacije
- Oporavak

Oporavak

- 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 unaprijed
  - pokušava se sustav vratiti u novo ispravno stanje (treba unaprijed predvidjeti koji se ispadi mogu dogoditi!)

Kontrolne točke (1)

- Koriste se za oporavak unazad
- Procese je potrebno vratiti u prošlost, ali u konzistentna stanja



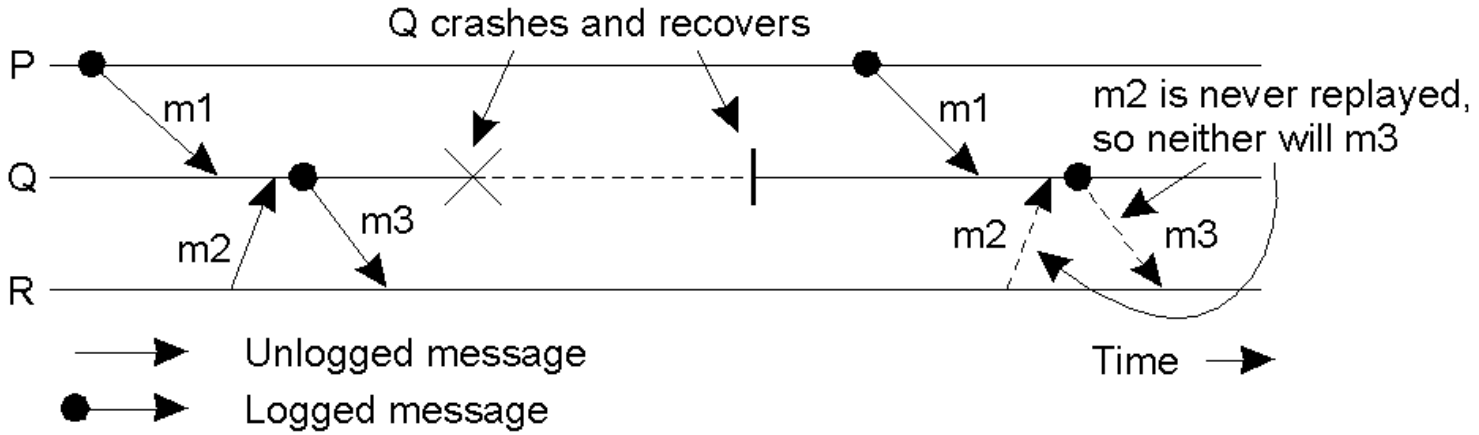
Primjer kontrolnih točaka koje P1 i P2 bilježe neovisno

Kontrolne točke (2)

- 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
- **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)

Zapisivanje poruka u dnevnički zapis (1)

- Svaka poruka sadrži sve potrebne informacije za ponovljeno slanje (pošiljatelj, primatelj, redni broj)
  - poruka je stabilna kada je zapisana u trajnu memoriju



Primjer neispravnog zapisivanja u dnevnički zapis

Zapisivanje poruka u dnevnički zapis (2)

Pravilo zapisivanja poruka u dnevnički zapis

- Svi procesi koji su ovisni o isporuci poruke *m* moraju pohraniti kopiju poruke *m*, tj. proces *P* ne smije slati niti jednu poruku nakon primitka *m* ako *m* nije zapisana u trajnu memoriju

Literatura

- 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.