

Pouzdanost računalnih sustava

Auditorne vježbe – 3. ciklus

Fakultet elektrotehnike i računarstva

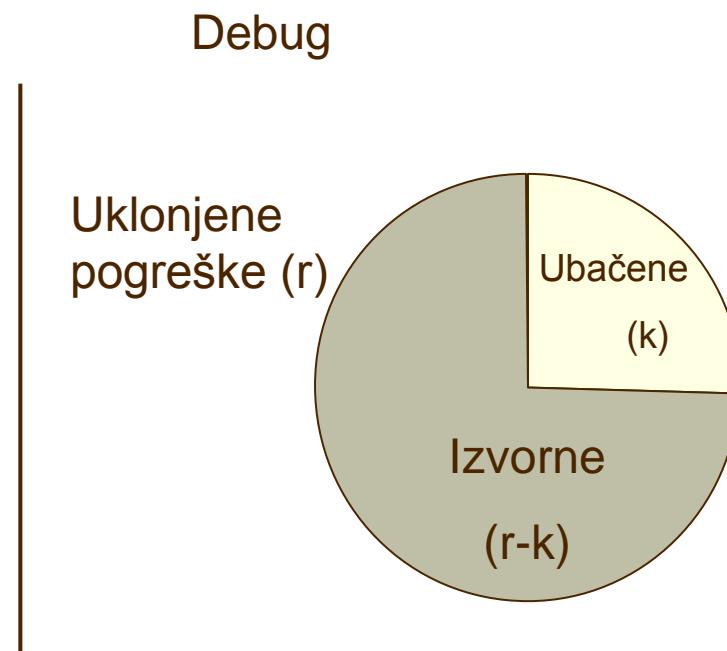
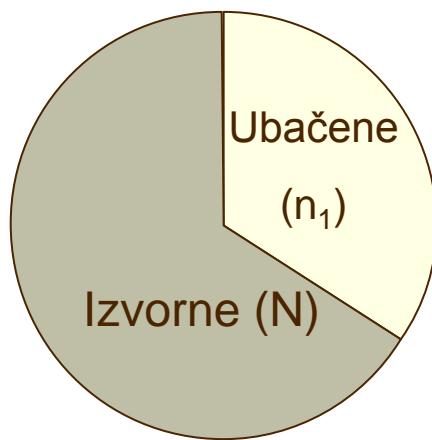
Zavod za elektroniku, mikroelektroniku, računalne i intelligentne sisteme

Zadatak 1. Millsov model pouzdanosti programske podrške

- Za danu programsku podršku određuje se pouzdanost na temelju Millsova modela. Neka je broj ubačenih pogrešaka **10**, a ukupan broj uklonjenih pogrešaka **15** od kojih je **10** bilo ubačenih.
- Odredite broj pogrešaka programske podrške.

Zadatak 1. Millsov model pouzdanosti programske podrške

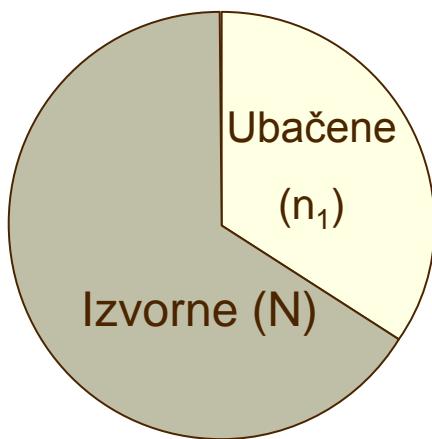
- Za danu programsku podršku određuje se pouzdanost na temelju Millsova modela. Neka je broj ubačenih pogrešaka 10, a ukupan broj uklonjenih pogrešaka 15 od kojih je 6 bilo ubačenih. Odredite broj pogrešaka programske podrške.



Zadatak 1. Millsov model pouzdanosti programske podrške

- $n_1 = 10$
- $r = 15$
- $k = 6$
- $N = ?$

MLE
procjena



Millsov model

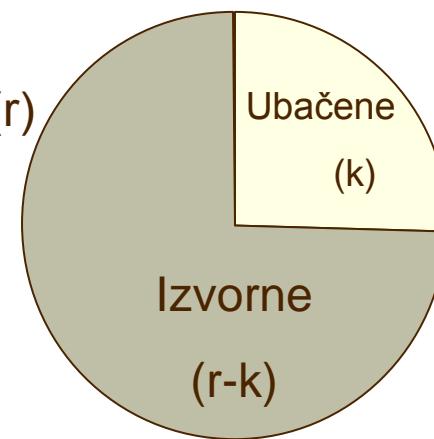
$$\hat{N} = \lfloor N_0 \rfloor + 1$$

$$N_0 = \frac{n_1(r-k)}{k} - 1$$

$$P(k; N, n_1, r) = \frac{\binom{n_1}{k} \binom{N}{r-k}}{\binom{N+n_1}{r}}, k = 1, 2, \dots, r$$

Debug

Uklonjene
pogreške (r)



Zadatak 1. Millsov model pouzdanosti programske podrške

- **$n_1 = 10$**
- **$r = 15$**
- **$k = 6$**
- **$N = ?$**

Millsov model

$$P(k; N, n_1, r) = \frac{\binom{n_1}{k} \binom{N}{r-k}}{\binom{N+n_1}{r}}, k = 1, 2, \dots, r$$

MLE
procjena

$$\hat{N} = \lfloor N_0 \rfloor + 1$$

$$N_0 = \frac{n_1(r-k)}{k} - 1 = \frac{10(15-6)}{6} - 1 = 14$$

Ukupan broj pogrešaka = 15

Zadatak 2. Jelinski-Moranda model pouzdanosti programske podrške

- Za zadani sustav programske podrške, tablica I. prikazuje intervale između zatajenja dobivenih testiranjem. Načiniti Jelinski-Moranda model pouzdanosti programa, uz uvjet da je poznat koeficijent kojim pojedina pogreška utiče na sustav, $c=0.0007$.
- Metodom procjene maksimalne sličnosti odrediti inicijalan broj pogrešaka u programu.

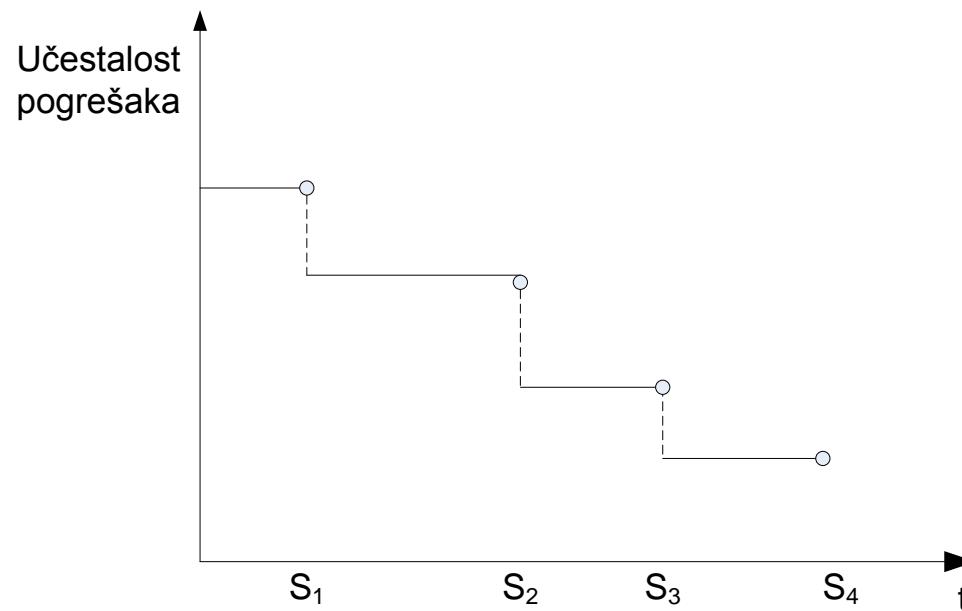
Zadatak 2. Jelinski-Moranda model pouzdanosti programske podrške

■ Tablica I.

Rbr	Interval	Rbr	Interval	Rbr	Interval
1	14,75	11	7,99	21	64,25
2	43,99	12	28,09	22	40,90
3	9,87	13	11,80	23	3,07
4	0,07	14	1,78	24	0,75
5	5,80	15	12,50	25	13,36
6	7,89	16	73,08	26	23,02
7	28,79	17	42,60	27	143,31
8	170,15	18	9,18	28	55,46
9	26,83	19	49,43	29	75,57
10	36,15	20	9,19	30	34,31

Zadatak 2. Jelinski-Moranda model pouzdanosti programske podrške

- **Vjerojatnosni model pouzdanosti programa**
- **Temelji se na učestalosti pogrešaka u programu**
 - **# pogrešaka – diskretna fija**
 - **Učestalost pogrešaka – diskretna fija sa diskontinuitetima u trenucima kvara**



Zadatak 2. Jelinski-Moranda – pretpostavke modela

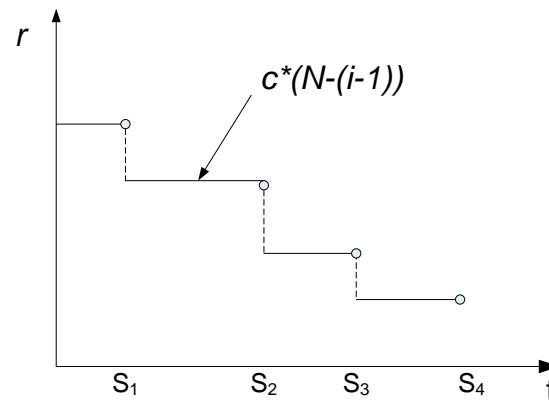
- **Fiksani broj pogrešaka N**
- **Pogreške su NEZAVISNE i jednako doprinose zatajenju**
- **Intervali u kojima se dešavaju pogreške međusobno su NEZAVISNI**
- **Prilikom svakog zatajenja otklanja se kvar i ispravljanje je savršeno**

Zadatak 2. Jelinski-Moranda – prepostavke modela

- **Učestalost pogrešaka $r(t_i - S_{i-1})$**
 - konstantna tijekom intervala $(t_i - S_{i-1})$
 - proporcionalna broju preostalih pogrešaka u programu sa koeficijentom c

$$r(t_i - S_{i-1}) = c * [N - (i - 1)]$$

- **N – inicijalni broj kvarova u programu**
- **$N - (i - 1)$ – preostali broj pogrešaka u programu**
- **CILJ: procjena inicijalnog broja pogrešaka MLE metodom**



Zadatak 2. Jelinski-Moranda – MLE procjena N

- **$f(t_i)$ – fija gustoća vjerojatnosti = ?**
 - učestalost pogrešaka (brzina/intenzitet kvara) $r = \text{konst}$ na INTERVALU
- **Podsjetnik: eksponencijalna fija**
- **Brzina kvarenja λ - konstantna tijekom vremena**

$$f(t) = \lambda e^{-\lambda t}$$

- \rightarrow na intervalu t_i

$$f(t_i) = r(t_i) e^{-\int_0^{t_i} r(x_i) dx_i} = c(N-i+1) e^{-c(N-i+1)t_i}$$


 $r(t_i)$

Zadatak 2. Jelinski-Moranda – MLE procjena N

■ MLE: funkcija procjene maksimalne sličnosti $L(N)$

$$\begin{aligned} L(N) &= \prod_{i=1}^n f(t_i) \\ &= \prod_{i=1}^n c(N-i+1)e^{-c(N-i+1)t_i} \\ &= c^n \left(\prod_{i=1}^n (N-i+1) \right) e^{-c \sum_{i=1}^n (N-i+1)t_i} \end{aligned} \quad \left. \right\} \text{Umnožak eksp. fija = eksp. fija sa zbrojem eksponenata}$$

$$\ln L(N) = n \ln c + \sum_{i=1}^n \ln(N-i+1) - c \sum_{i=1}^n (N-i+1)t_i$$

$$\frac{\partial}{\partial N} \ln L = \sum_{i=1}^n \frac{1}{N-i+1} - c \sum_{i=1}^n t_i = 0$$

Zadatak 2. Jelinski-Moranda – MLE procjena N

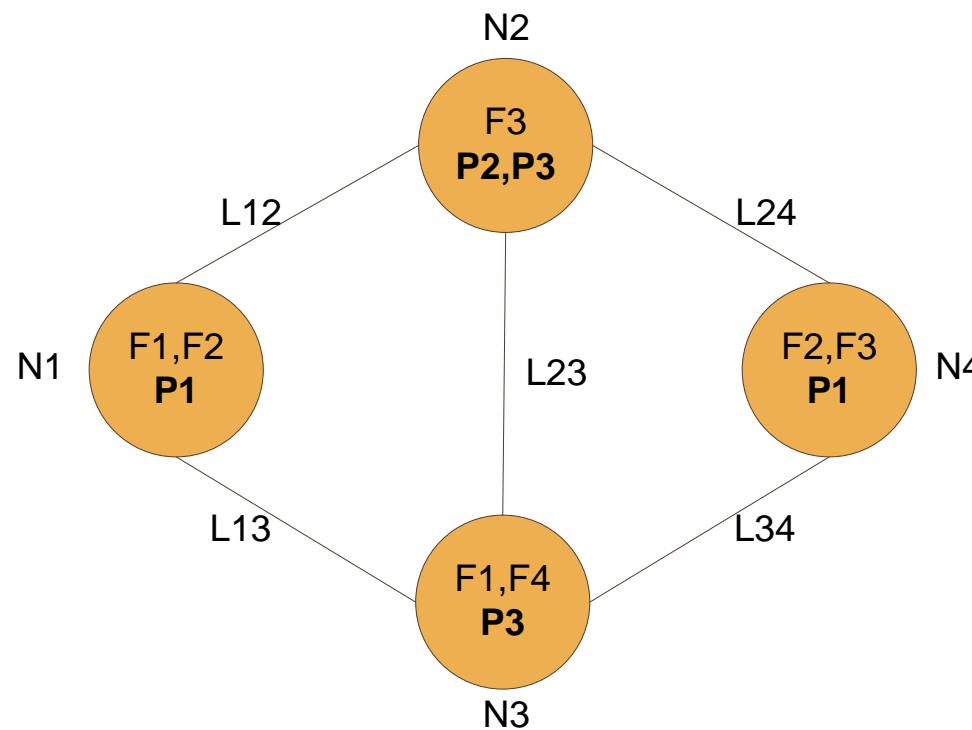
- **Zadatak**
- **$c = 0.0007$**

Rbr	Interval	Rbr	Interval	Rbr	Interval
1	14,75	11	7,99	21	64,25
2	43,99	12	28,09	22	40,90
3	9,87	13	11,80	23	3,07
4	0,07	14	1,78	24	0,75
5	5,80	15	12,50	25	13,36
6	7,89	16	73,08	26	23,02
7	28,79	17	42,60	27	143,31
8	170,15	18	9,18	28	55,46
9	26,83	19	49,43	29	75,57
10	36,15	20	9,19	30	34,31

- **$N = 54$**

Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- Neka je dan raspodijeljeni sustav prikazan na slici. Za dan i sustav potrebno je opisati princip određivanja pouzdanosti primjenom Kumarova algoritma te pokazati izvođenje algoritma na primjeru programa P1.



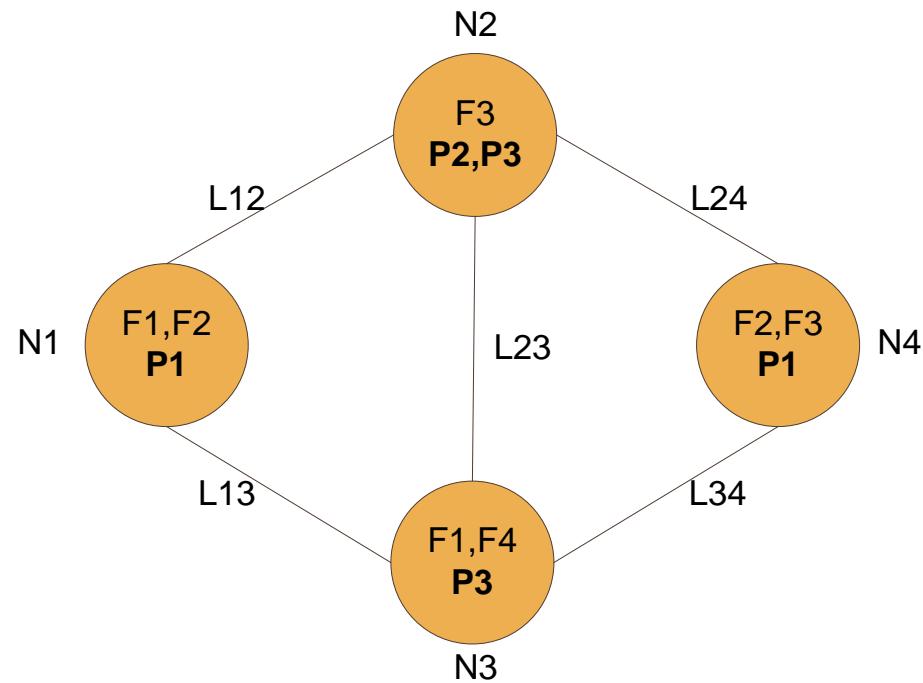
Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

Karakteristike RS:

- 4 čvora
- 3 programa: P1, P2, P3
- 4 datoteke: F1, F2, F3, F4

Izvođenje:

- $P1 \rightarrow \{F1, F2, F3\}$
- $P2 \rightarrow \{F1, F2, F4\}$
- $P3 \rightarrow \{F1, F2, F3, F4\}$



Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

■ Kumarov algoritam:

1. Za (sve programe P_i)

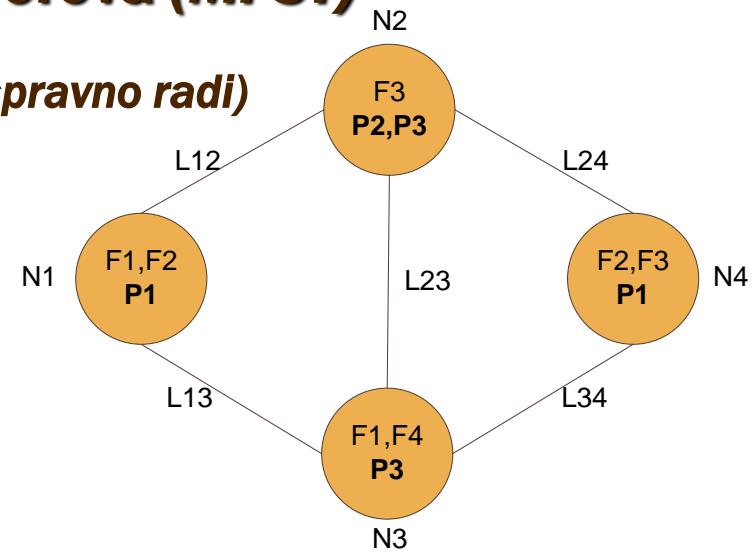
FST = Izgradi stabla čvorova koji sadrže potrebne datoteke za izvođenje (P_i);

$MFST$ = Pronađi u FST sva stabla koja sadržre minimalan potreban broj čvorova;

2. Algoritam evaluacije završnih čvorova ($MFST$)

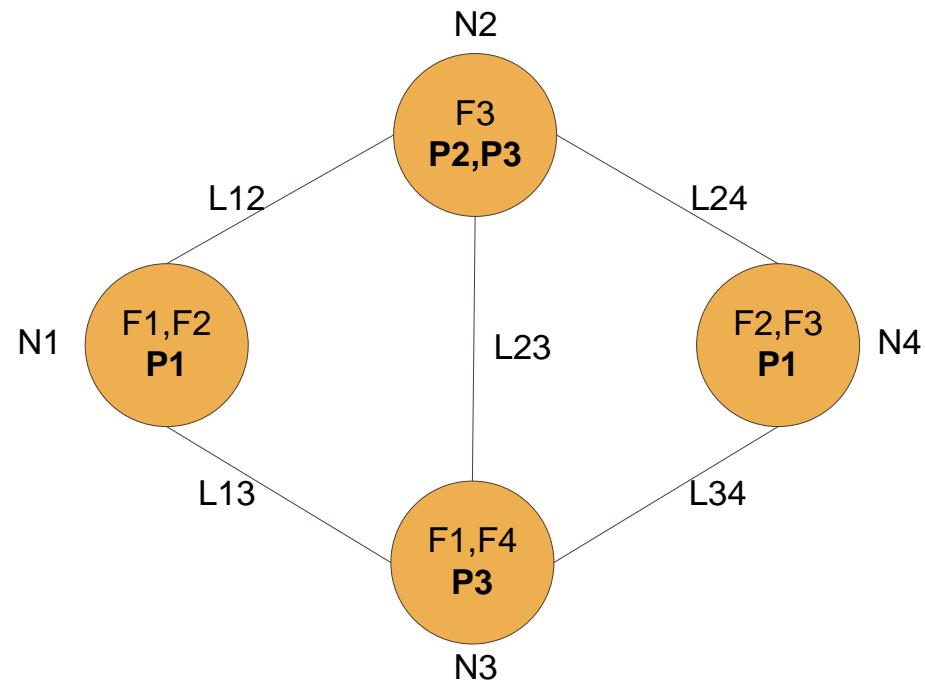
$DPR = P(\text{za svaki } P_i \text{ barem jedan } MFST \text{ ispravno radi})$

$$DPR = P\left(\bigcup_{j=1}^N MFST_j\right)$$



Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- **P1 na N1 i N4**
- **Izvođenje:**
- **$P1 \rightarrow \{F1, F2, F3\}$**

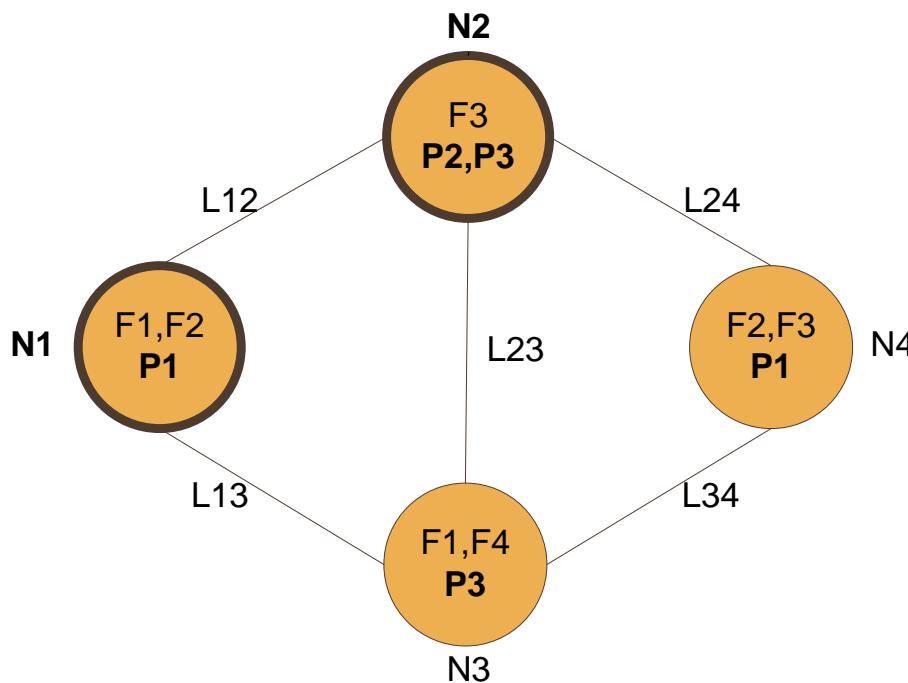


- **Stablo pokrivanja datoteka FST**
- → **čvorovi i veze koji sadrže sve datoteke potrebne za izvođenje nekog programa**

Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- **P1 na N1**
- **P1 → {F1, F2, F3}**
 - **Lokalno: F1, F2**
 - **F3 – N2, N4**

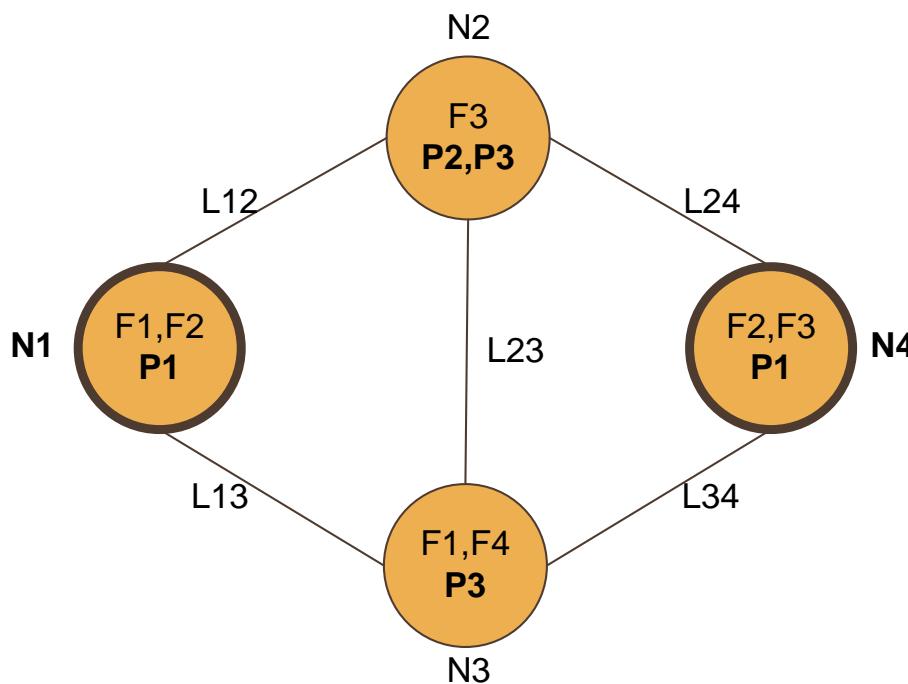
FST – do N2
N1, N2, L12
N1, N2, N3, L13, L23



Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- **P1 na N1**
- **P1 → {F1, F2, F3}**
 - **Lokalno: F1, F2**
 - **F3 – N2, N4**

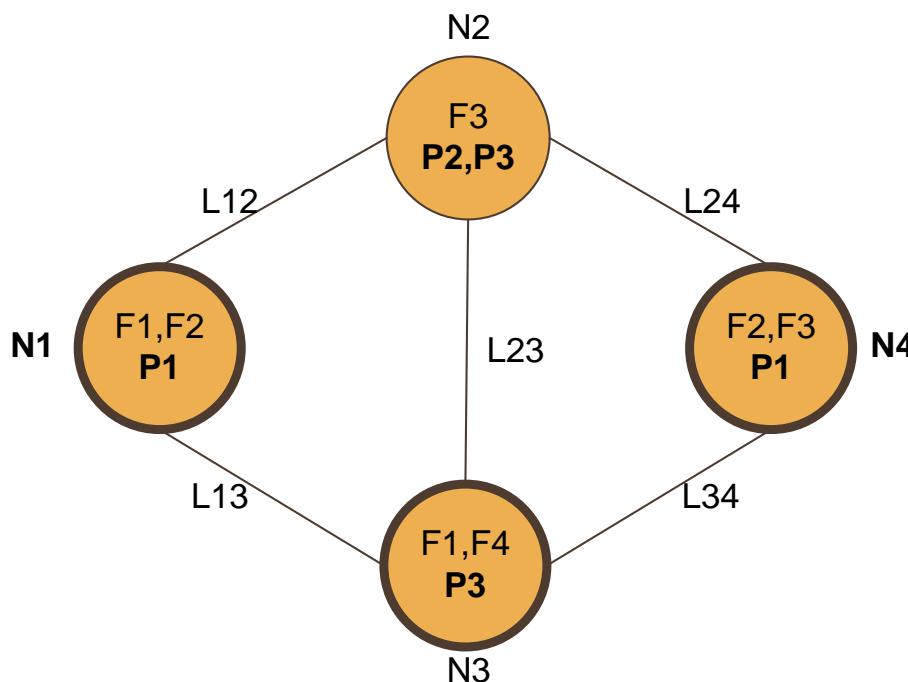
FST – N1 do N2
N1, N2, L12
N1, N2, N3, L13, L23



FST – N1 do N4
N1, N2, N4, L12, L24
N1, N3, N4, L13, L34
N1, N2, N3, N4, L12, L23, L34
N1, N2, N3, N4, L13, L23, L24

Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- **P1 na N4**
- **P1 → {F1, F2, F3}**
 - **Lokalno: F2, F3**
 - **F1 – N1, N3**



FST – N4 do N3
N3, N4, L34
N2, N3, N4, L24, L23

FST – N4 do N1
N1, N2, N4, L12, L24
N1, N3, N4, L13, L34
N1, N2, N3, N4, L12, L23, L34
N1, N2, N3, N4, L13, L23, L24

Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- **Minimalno stablo pokrivanja datoteka $MFST$**
- **FST za koje ne postoji niti jedno drugo stablo FST_j**

$$FST \subseteq FST_j$$

FST – N4 do N3
N3, N4, L34
N2, N3, N4, L24, L23

FST – N4 do N1
N1, N2, N4, L12, L24
N1, N3, N4, L13, L34
N1, N2, N3, N4, L12, L23, L34
N1, N2, N3, N4, L13, L23, L24

FST – do N2
N1, N2, L12
N1, N2, N3, L13, L23

Pretraga: od najkraćih stabala prema duljima

Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- **Minimalno stablo pokrivanja datoteka $MFST$**
- **FST za koje ne postoji niti jedno drugo stablo FST_j**

$$FST \subseteq FST_j$$

FST – N4 do N3
N3, N4, L34
N2, N3, N4, L24, L23



FST – N4 do N1
N1, N2, N4, L12, L24
N1, N3, N4, L13, L34
N1, N2, N3, N4, L12, L23, L34
N1, N2, N3, N4, L13, L23, L24

FST – do N2
N1, N2, L12
N1, N2, N3, L13, L23

Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- **Minimalno stablo pokrivanja datoteka $MFST$**
- **FST za koje ne postoji niti jedno drugo stablo FST_j**

$$FST \subseteq FST_j$$

FST – N4 do N3
N3, N4, L34
N2, N3, N4, L24, L23



FST – N4 do N1
N1, N2, N4, L12, L24
N1, N2, N3, N4, L13, L23, L24

FST – do N2
N1, N2, L12
N1, N2, N3, L13, L23

Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

- **Minimalno stablo pokrivanja datoteka $MFST$**
- **FST za koje ne postoji niti jedno drugo stablo FST_j**

$$FST \subseteq FST_j$$

FST – N4 do N3
N3, N4, L34
N2, N3, N4, L24, L23

FST – N4 do N1
N1, N2, N4, L12, L24

FST – do N2
N1, N2, L12
N1, N2, N3, L13, L23



Zadatak 3. Pouzdanost programa u raspodijeljenim sustavima

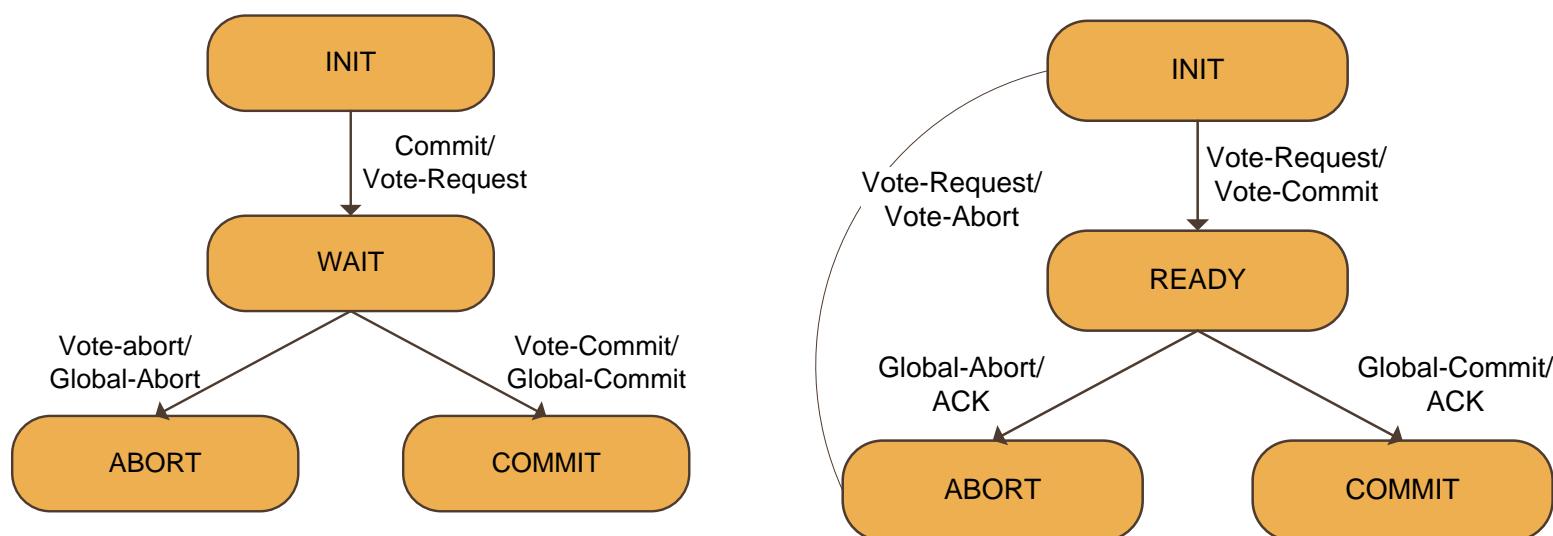
- **Skup minimalnih stabala – 4 stabla**
 - {N3, N4, L34},
 - {N2, N3, N4, L24, L23},
 - {N1, N2, L12},
 - {N1, N2, N3, L13, L23}
- **P1 se uspješno izvodi ako su svi čvorovi i veze barem jednog od stabala u ispravnom stanju**
- **DPR – poznate su pouzdanosti pojedinih čvorova i veza**
 - **Vjerojatnost da za svaki program postoji barem jedan MFST na kojem su svi čvorovi i veze ispravni**

Zadatak 4. Protokoli potvrđivanja u raspodijeljenim sustavima

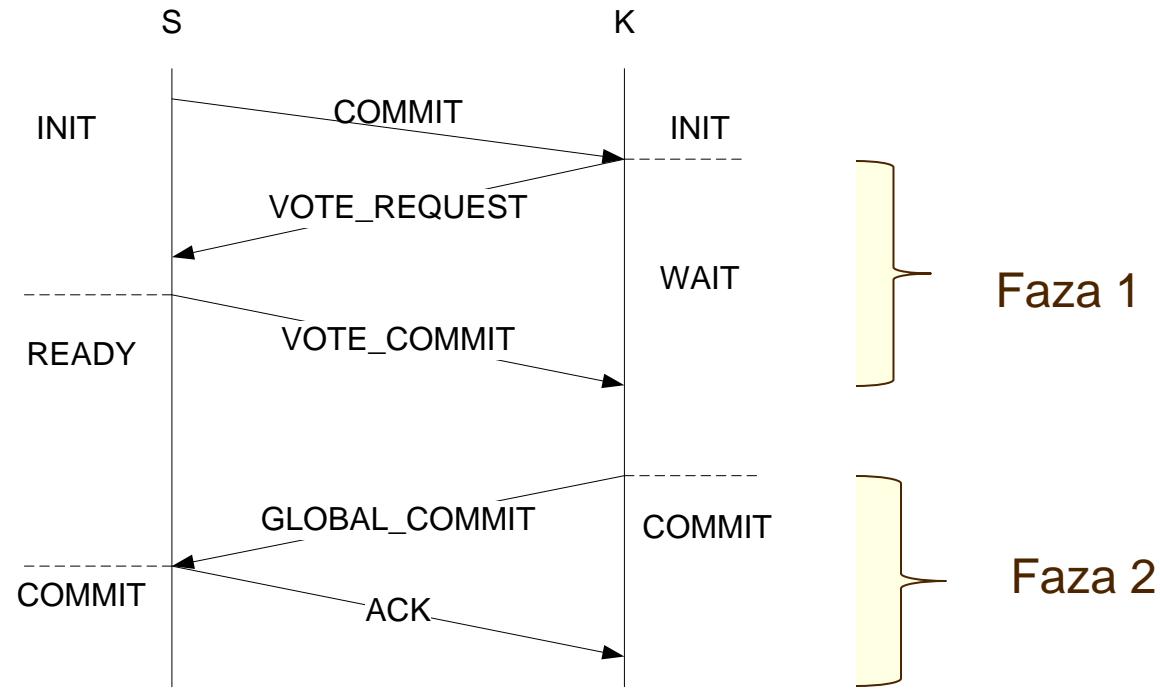
- **Opisati protokole potvrđivanja u 2 faze (2PC) te u 3 faze (3PC) te prikazati odgovarajuće mehanizme oporavka u slučaju ispada:**
 - Čvora sudionika
 - Čvora koordinatora
 - Čitavog segmenta mreže
- **Objasniti potencijalne probleme ovih protokola**

Zadatak 4. Protokoli potvrđivanja u raspodijeljenim sustavima

- **Arhitektura RS: centralizirana sa globalnim koordinatorom**
- **Protokol potvrđivanja u 2 faze:**
 - **Glasanje**
 - **Odluka o potvrđivanju**



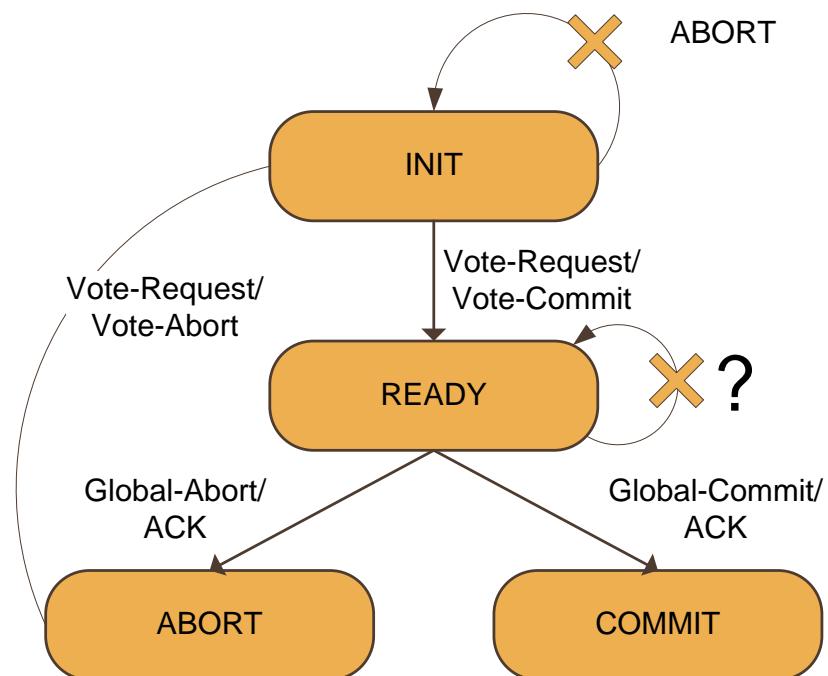
Zadatak 4. 2 PC



Zadatak 4. 2 PC – blokirajuća stanja

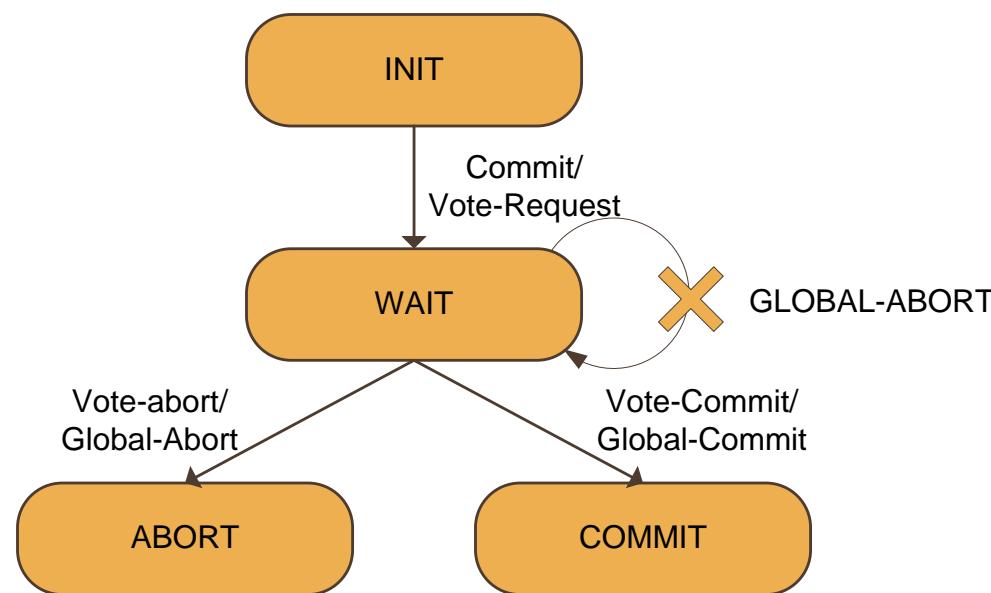


- **Rješenje: mehanizam timeout-a**
- **Čvor sudionik:**
- **INIT – timeout → ABORT**
- **READY – timeout → pita ostale sudionike**

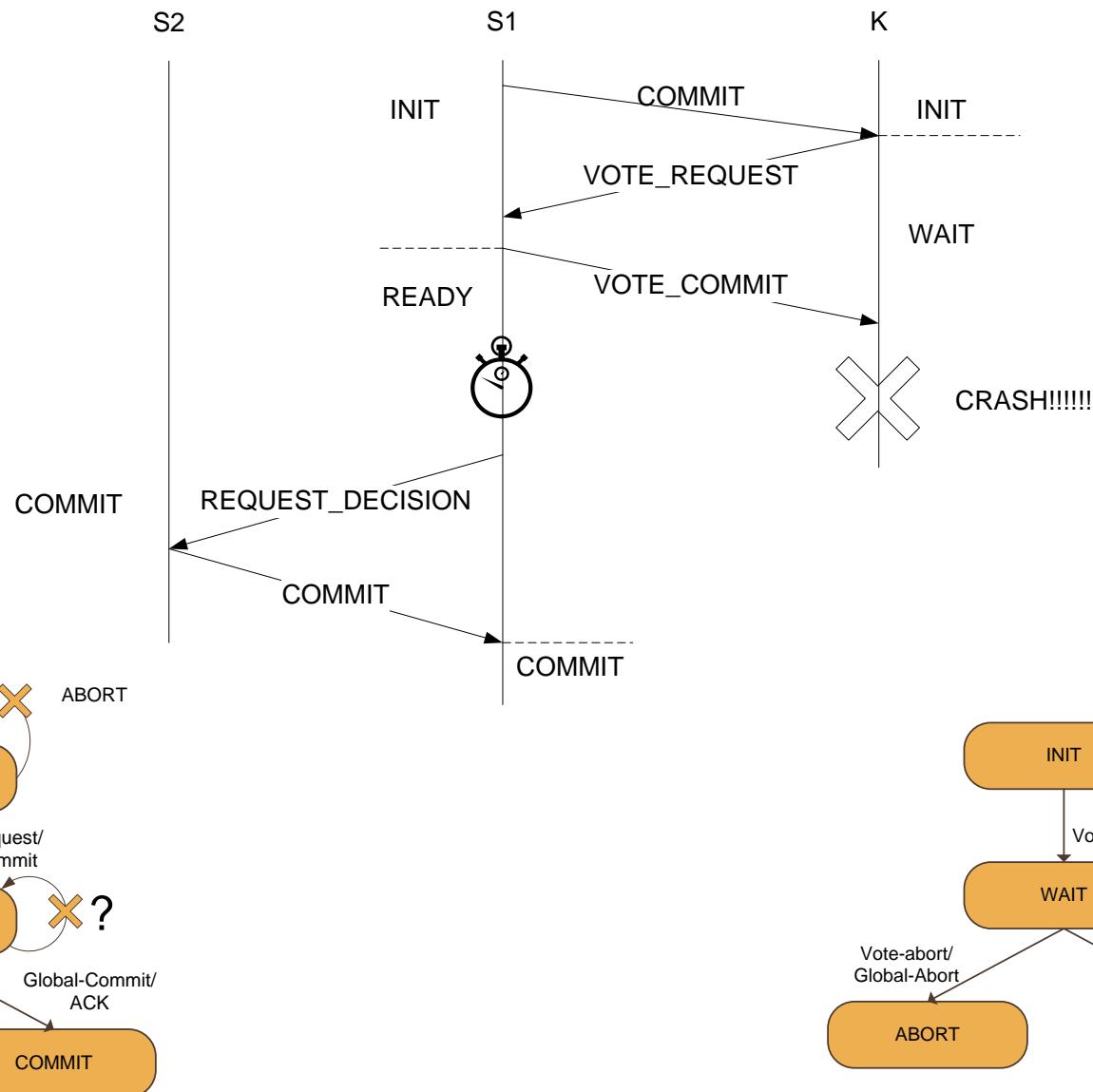


Zadatak 4. 2 PC – blokirajuća stanja

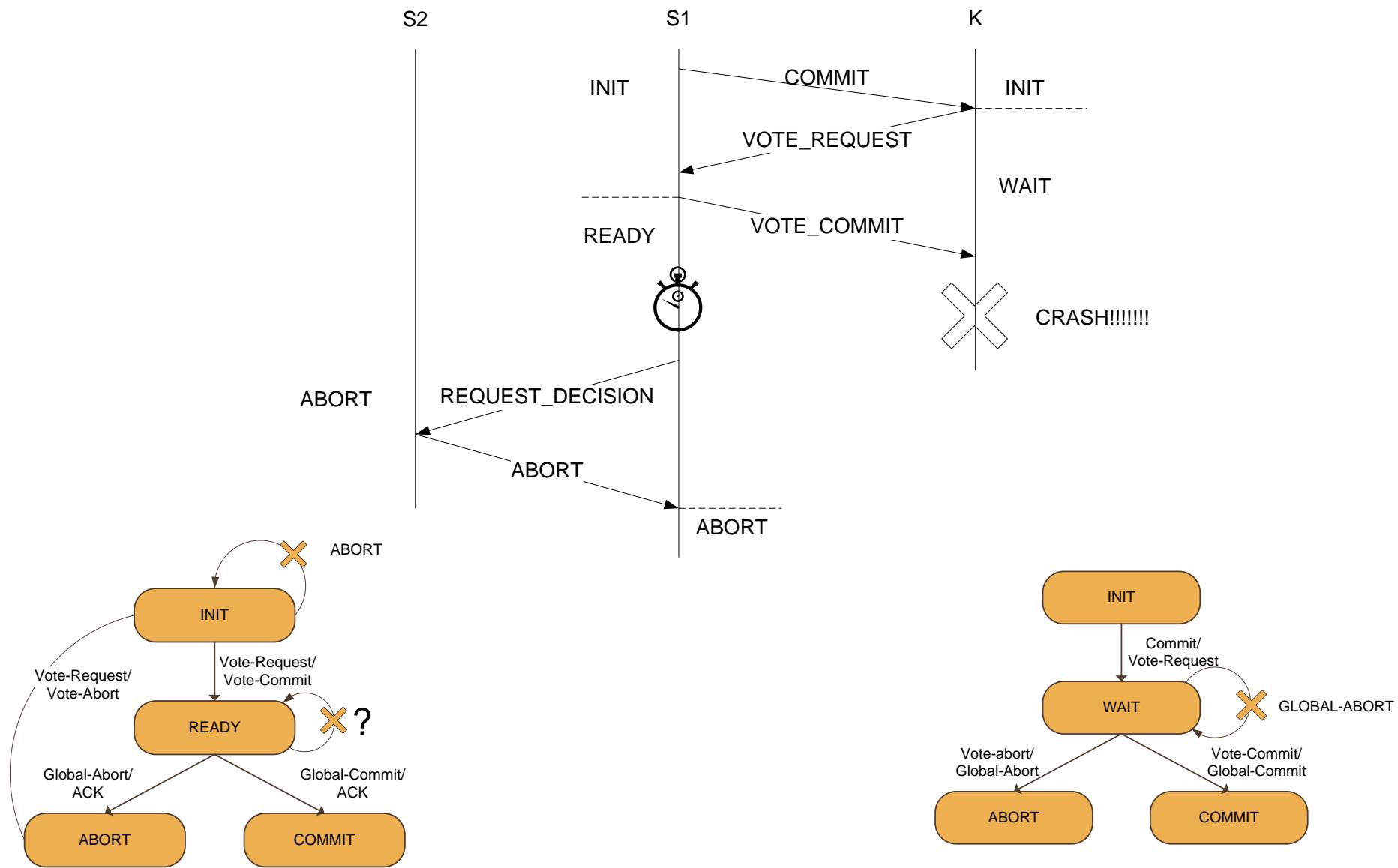
- **Rješenje: mehanizam timeout-a**
- **Koordinator:**
- **WAIT – timeout → GLOBAL-ABORT**



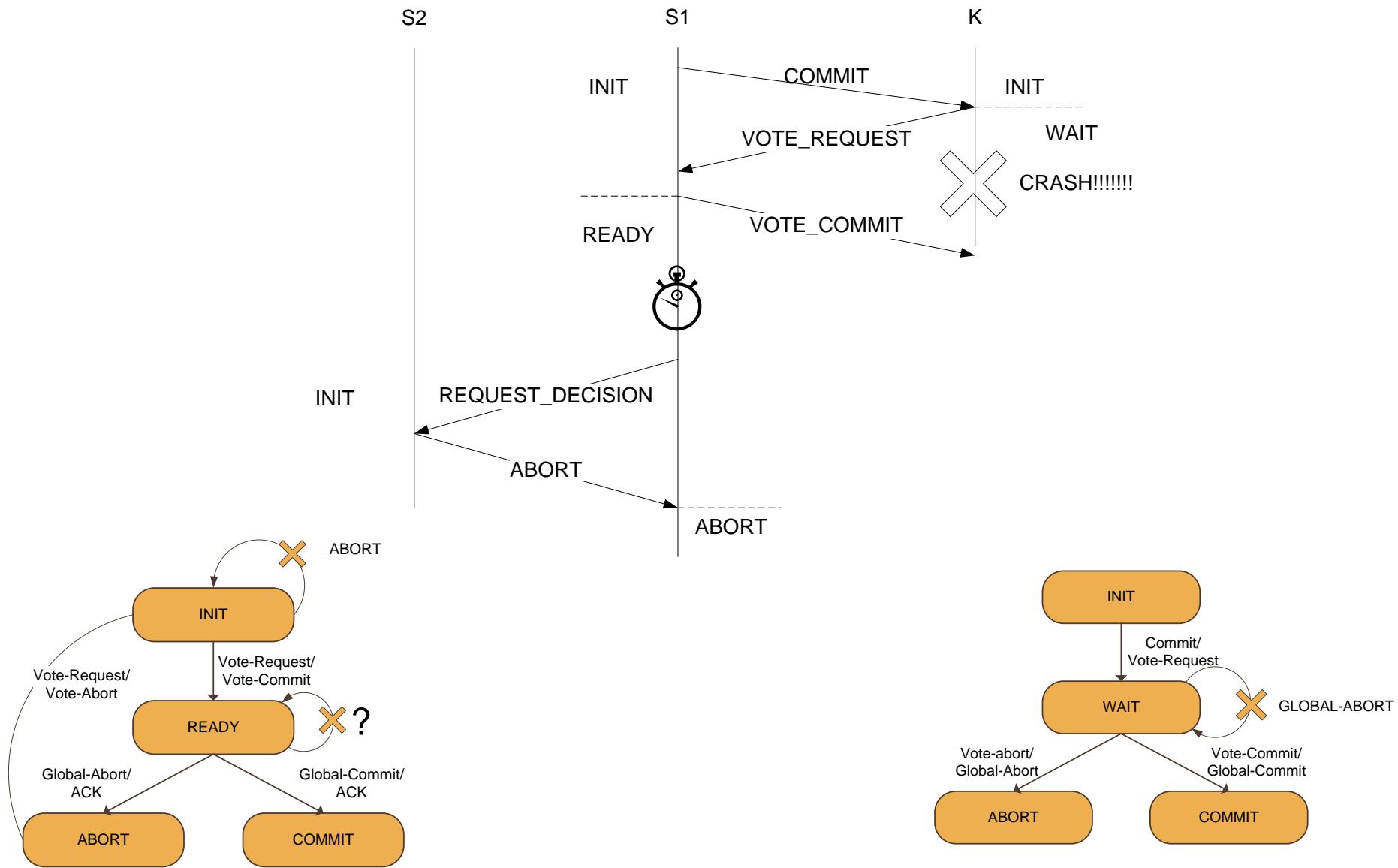
Zadatak 4. 2 PC – ispad koordinatora



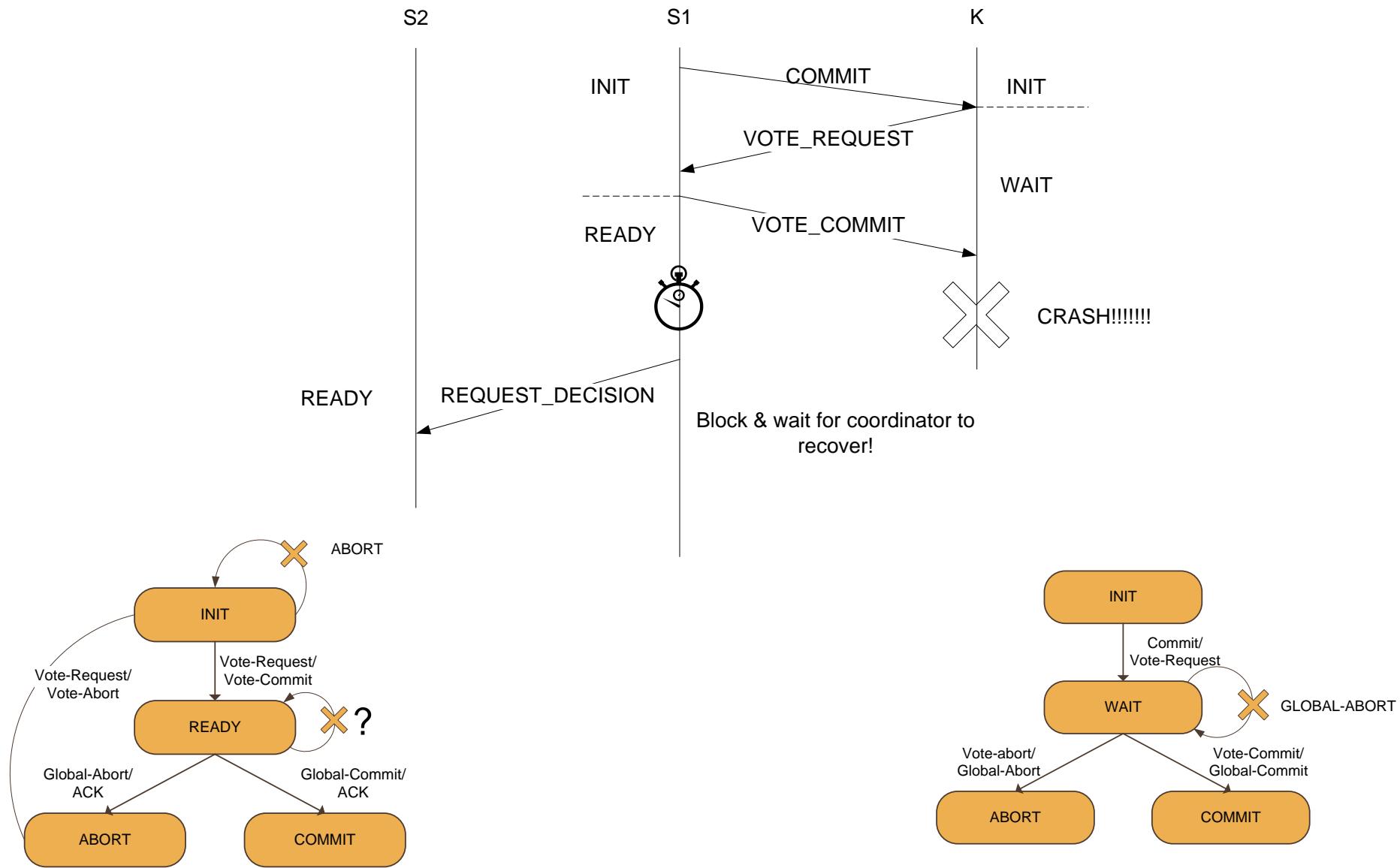
Zadatak 4. 2 PC – ispad koordinatora



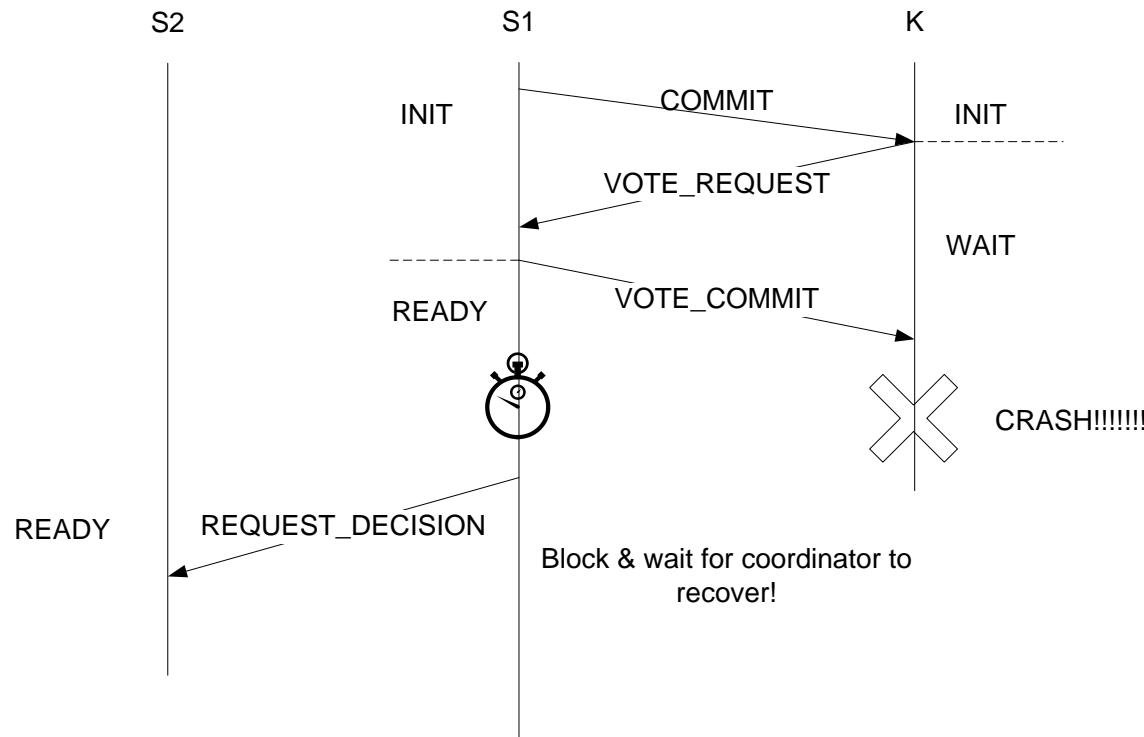
Zadatak 4. 2 PC – ispad koordinatora



Zadatak 4. 2 PC – ispad koordinatora



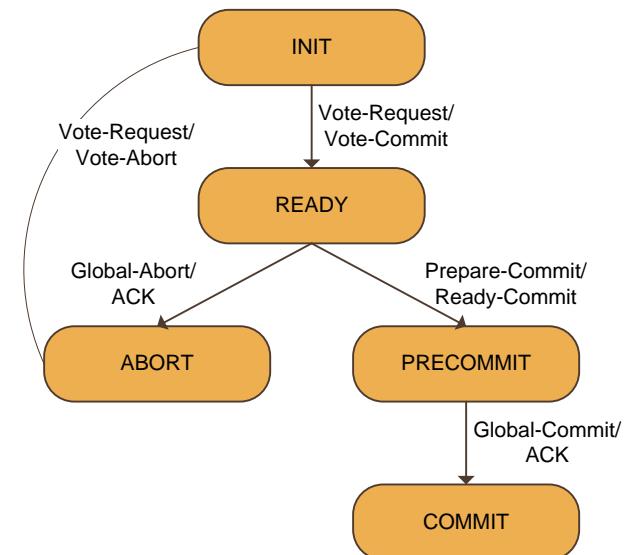
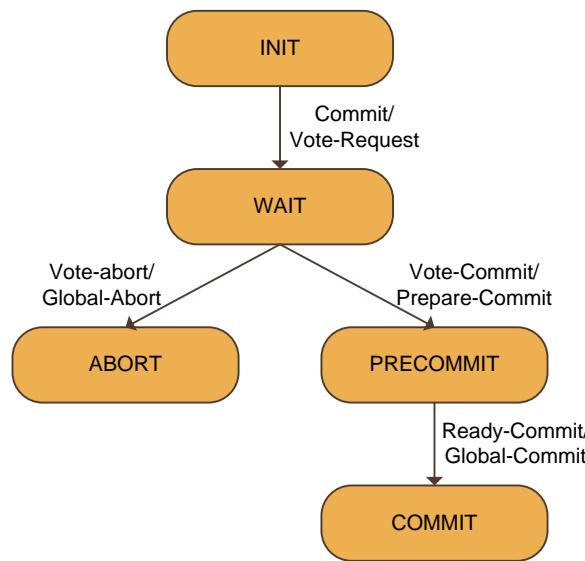
Zadatak 4. 2 PC – ispad koordinatora



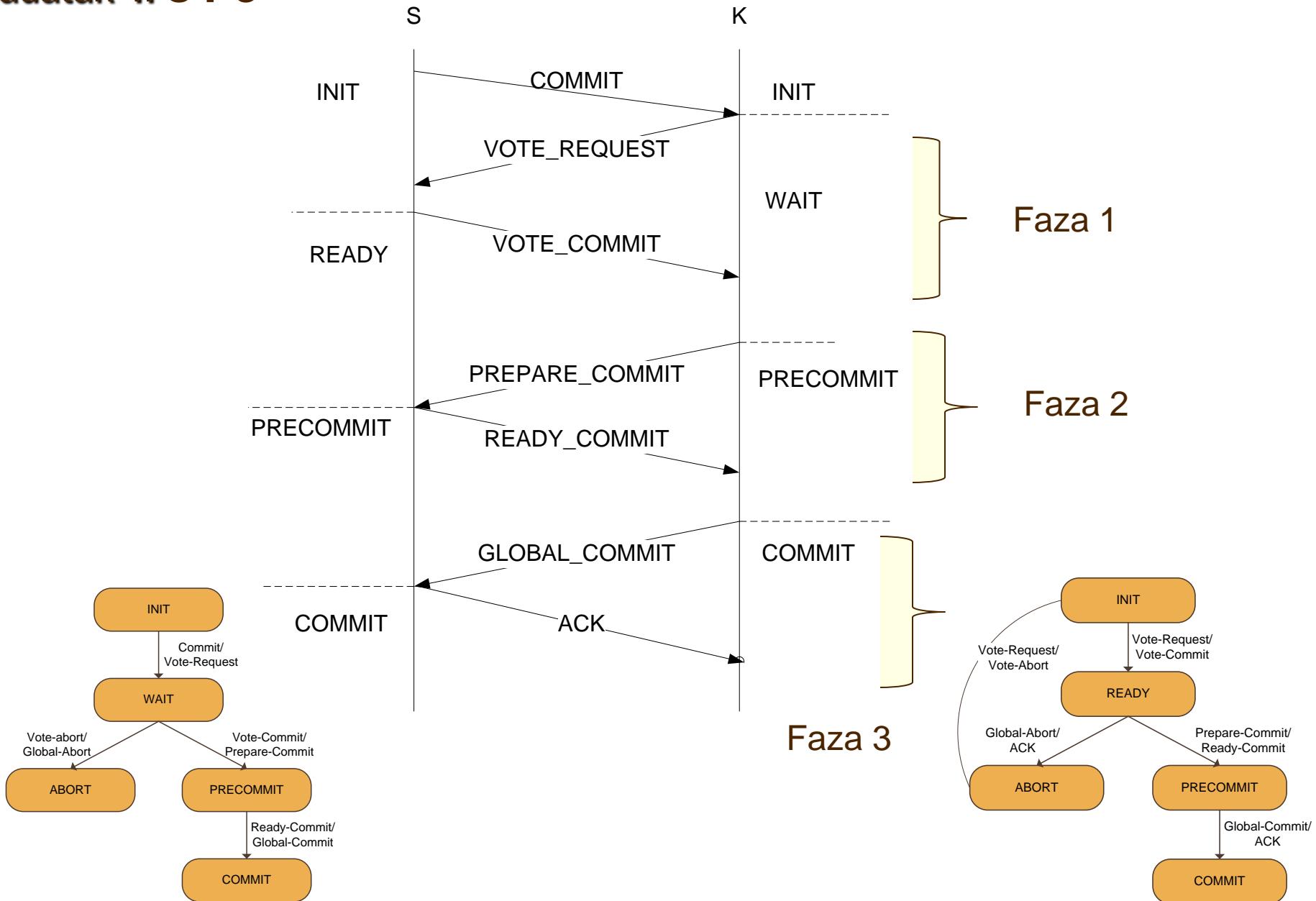
- Čvorovi ne mogu potvrditi transakciju iako su svi spremni za potvrđivanje
- ZAŠTO?
- Možda je neki od sudionika ispao – ne mogu znati kako bi on glasao!

Zadatak 4. Rješenje: protokol potvrđivanja u tri faze (3PC)

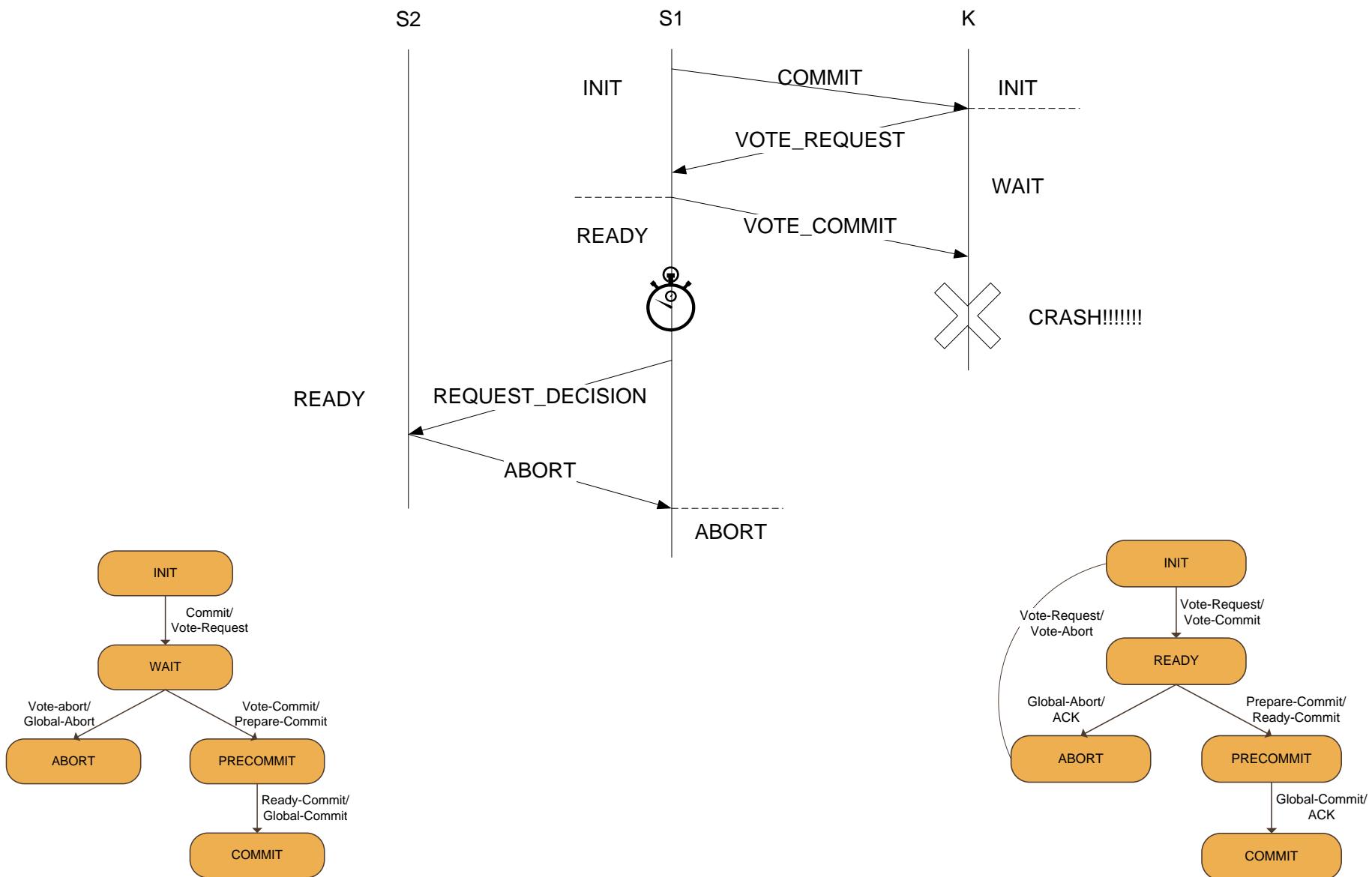
- **Zahtjevi:**
 - Ne postoji stanje iz kojeg se direktno može u COMMIT/ABORT**
 - Ne postoji stanje iz kojeg se može donijeti konačna odluka, a da se iz njega može napraviti prijelaz u COMMIT/ ABORT**



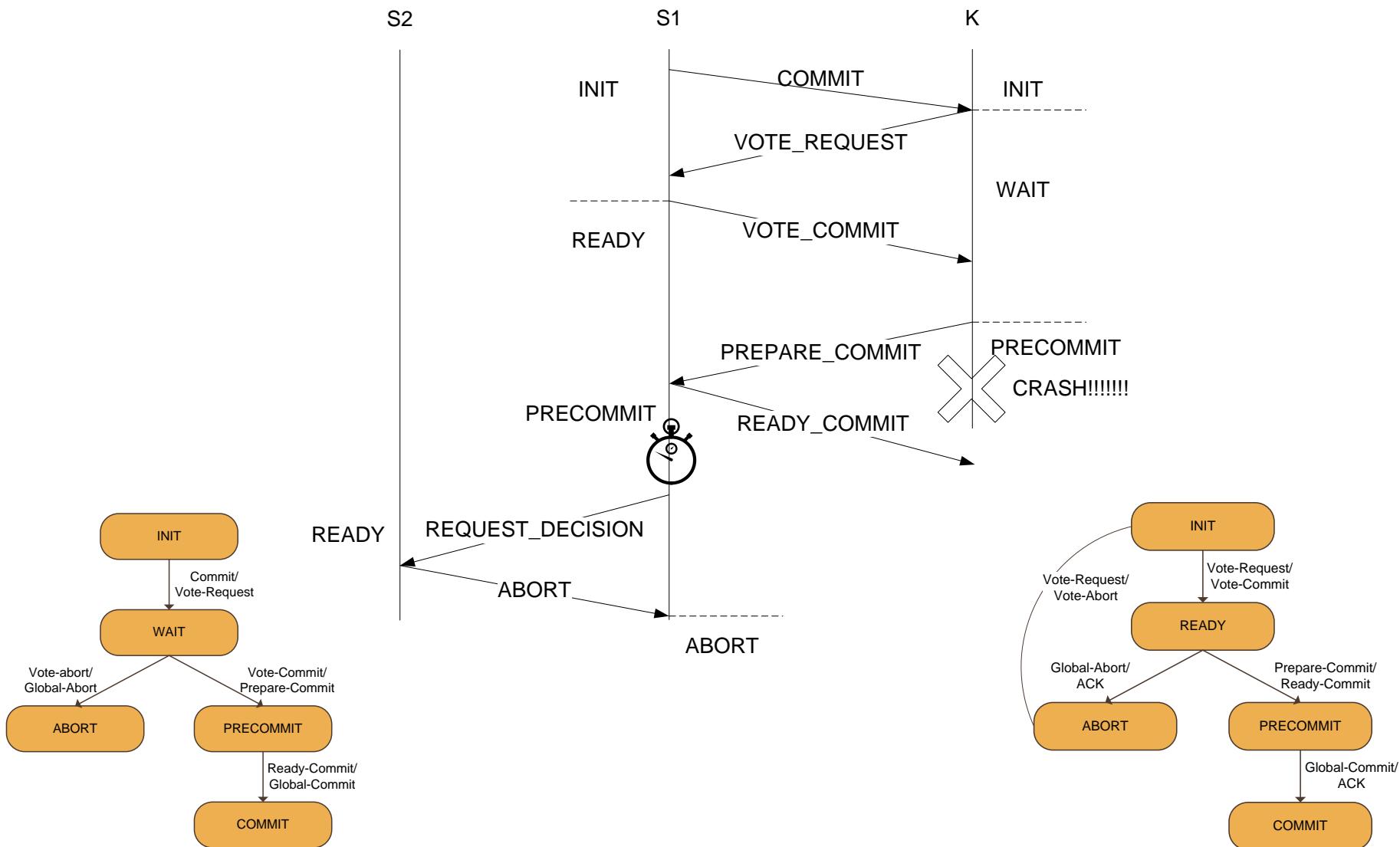
Zadatak 4. 3 PC



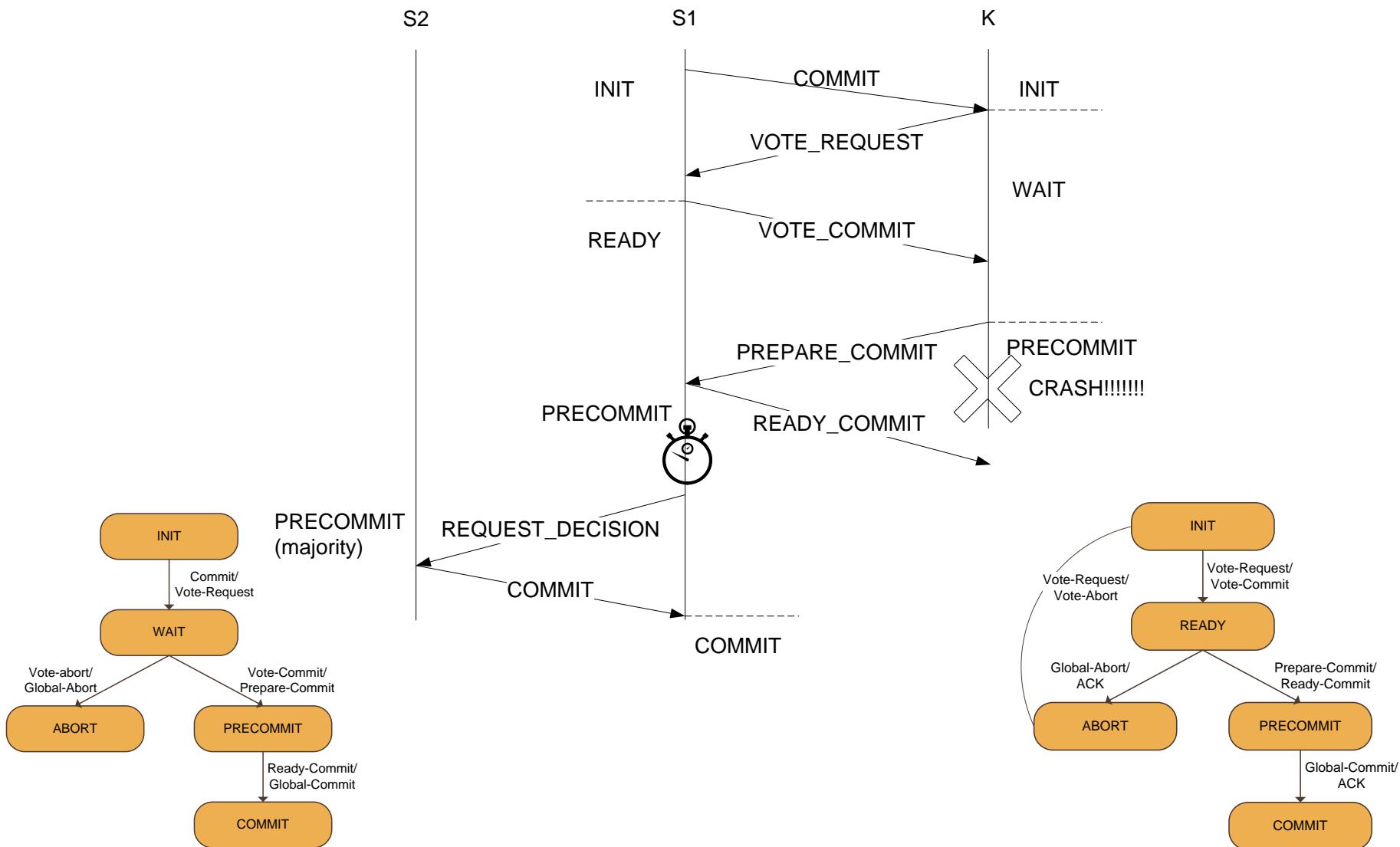
Zadatak 4. 3 PC – ispad koordinatora



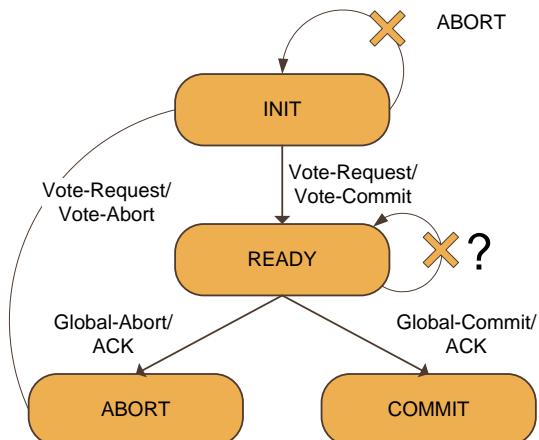
Zadatak 4. 3 PC – ispad koordinatora



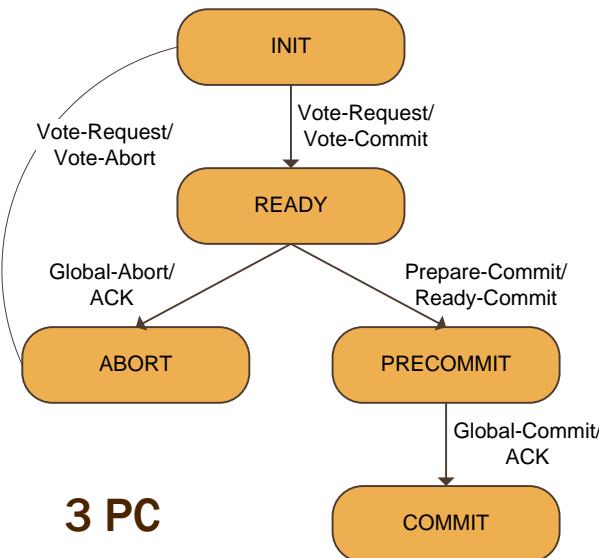
Zadatak 4. 3 PC – ispad koordinatora



Zadatak 4. 3 PC – ispad čvora sudionika



2 PC

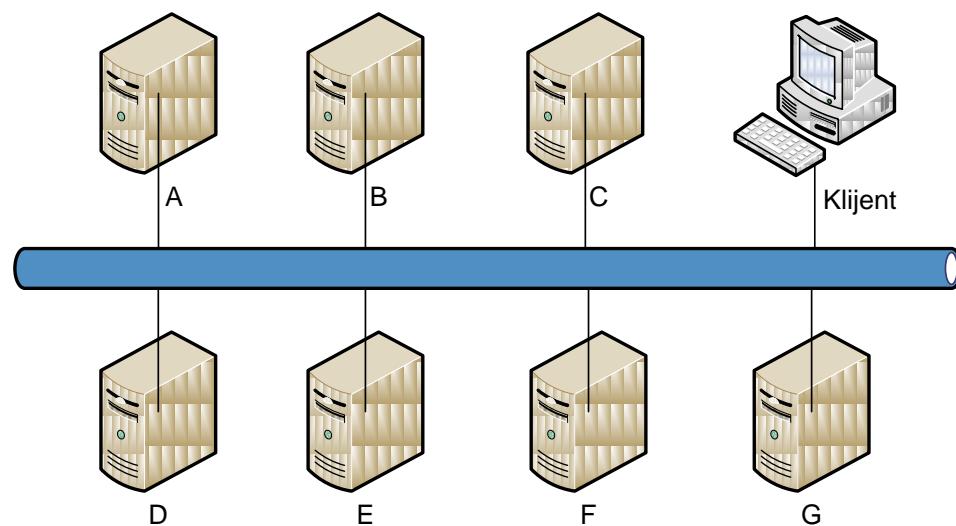


3 PC

- RAZLIKA U ODNOSU NA 2 PC:
- Čvor se ne može oporaviti u COMMIT stanje!

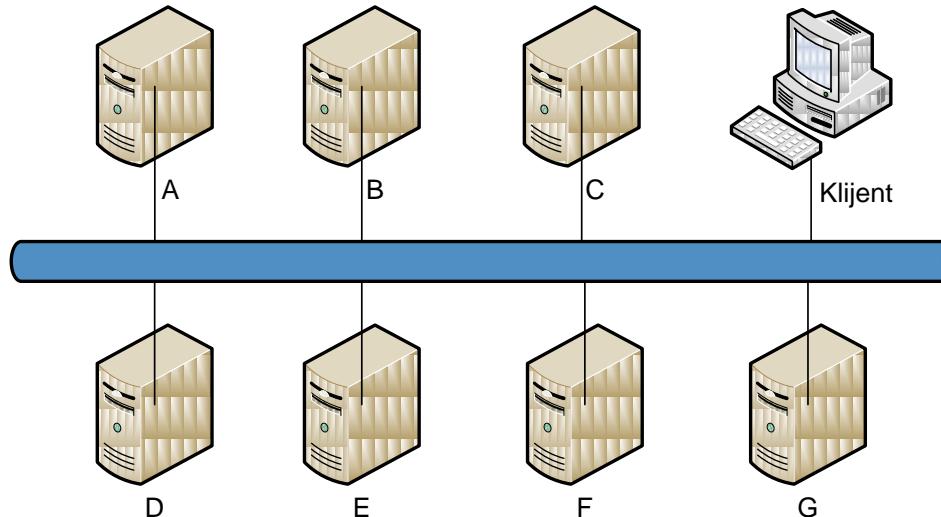
Zadatak 5. Protokol konsenzusa kvorumom

- Neka je dan raspodijeljeni datotečni sustav kojeg sačinjava 7 datotečnih poslužitelja prikazanih na slici. Operacije pisanja i čitanja sinkronizirane su prema metodi koncenzusa kvorumom. Odredite koje od sljedećih shemi omogućavaju rad bez konfliktata:
- $N_w = 3, N_r = 4$
- $N_w = 4, N_r = 3$
- $N_w = 4, N_r = 4$
- $N_w = 7, N_r = 1$
- $N_w = 4, N_r = 6$



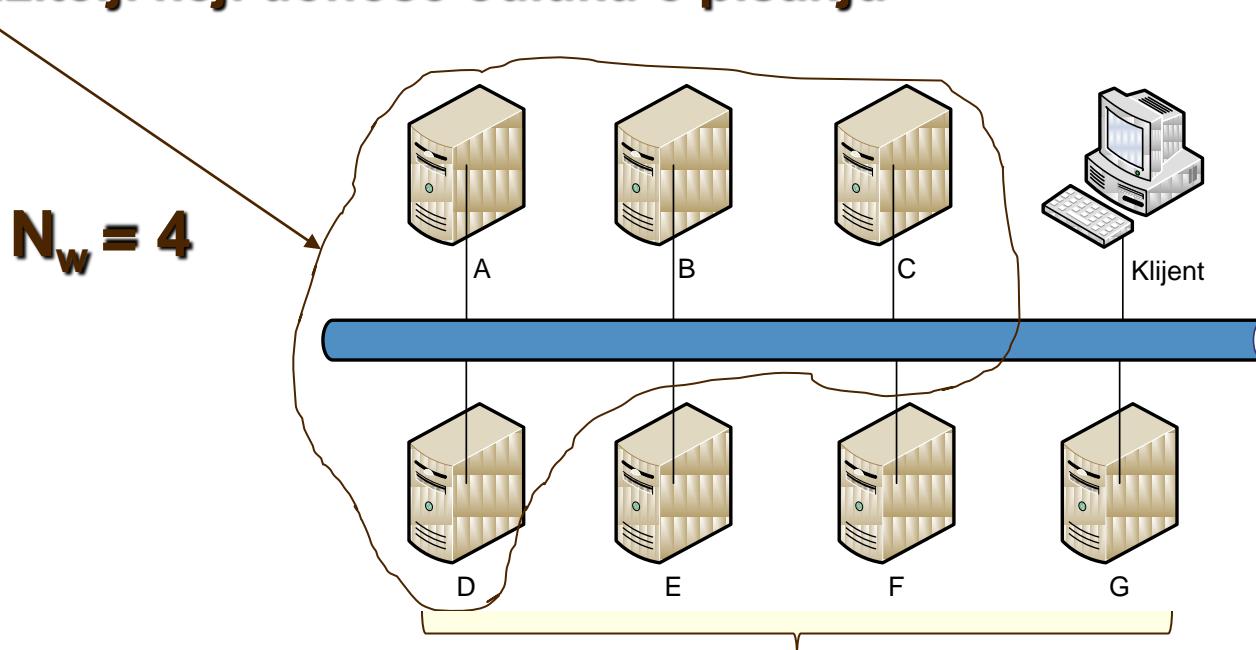
Zadatak 5. Protokol konsenzusa kvorumom

- **$N = 7$ datotečnih poslužitelja → 7 kopija datoteke**
- **Klijenti – prije čitanja / pisanja šalju upit poslužiteljima**
- **Kvorum za pisanje $| N_w |$**
 - **Poslužitelji koji donose odluku o pisanju**
- **Kvorum za čitanje $| N_r |$**
 - **Poslužitelji koji donose odluku o čitanju**



Zadatak 5. Protokol konsenzusa kvorumom

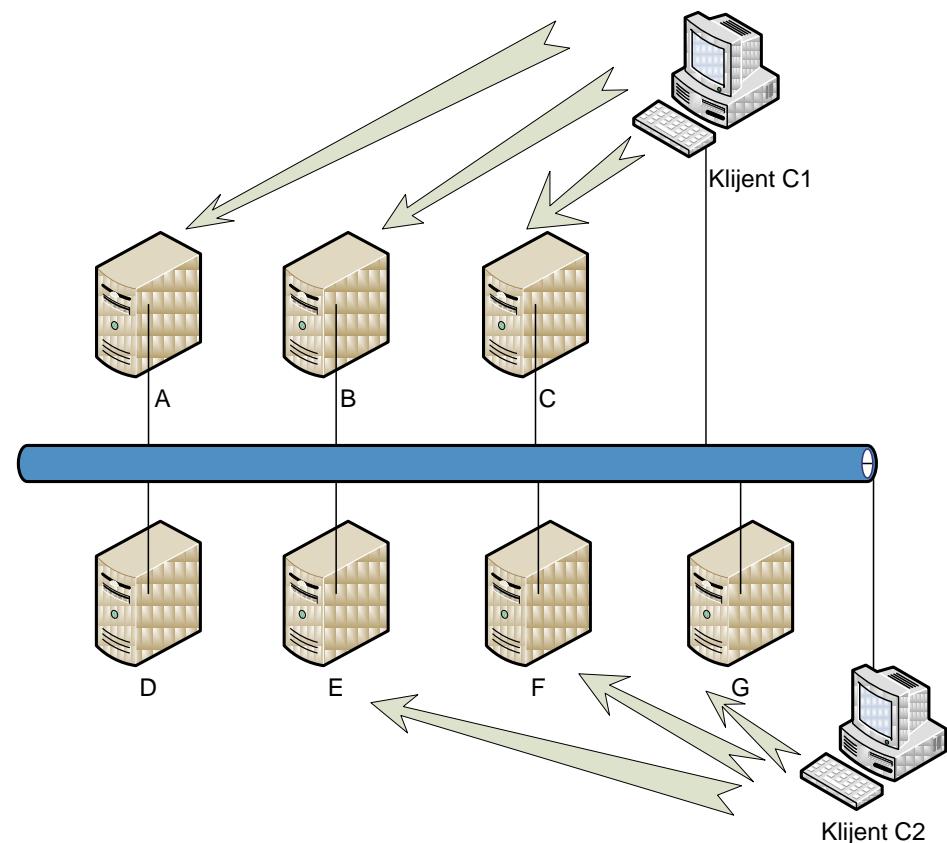
- **$N = 7$ datotečnih poslužitelja → 7 kopija datoteke**
- **Klijenti – prije čitanja / pisanja šalju upit poslužiteljima**
- **Kvorum za pisanje $|N_w|$**
 - **Poslužitelji koji donose odluku o pisanju**



- **Kvorum za čitanje $|N_r|$** $N_r = 4$
 - **Poslužitelji koji donose odluku o čitanju**

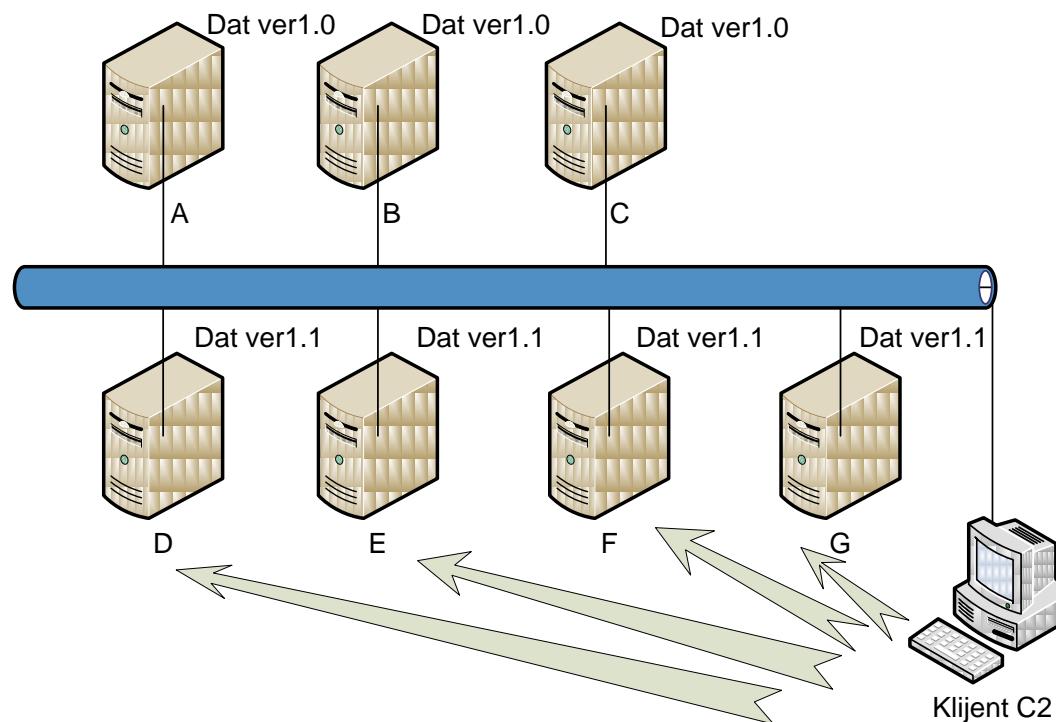
Zadatak 5. Protokol konsenzusa kvorumom – pisanje (osvježavanje)

- **Pisanje (osvježavanje) uvišestručenih datoteka**
- **Veličina kvoruma za pisanje: $N_w > N / 2$**
- → **ne mogu 2 transakcije istovremeno osvježavati**
- **$N_w = 3$**
- **Klijent C1 → A, B, C**
- **Klijent C2 → E, F, G**
- → **oba imaju dovoljno glasova**



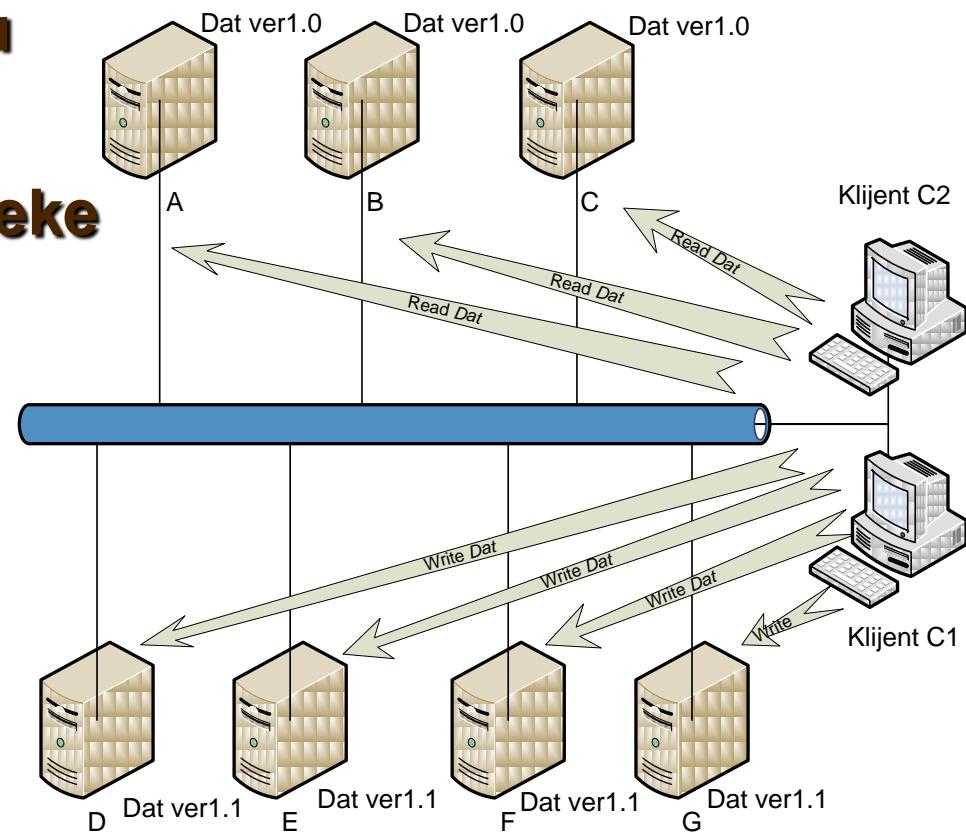
Zadatak 5. Protokol konsenzusa kvorumom – pisanje (osvježavanje)

- $N_w = 4 > N / 2 (3)$
- samo jedan klijent!
- Pisanje → $N_w = 4$ poslužitelja mijenja se verzija datoteke
Dat iz ver 1.0 u ver 1.1



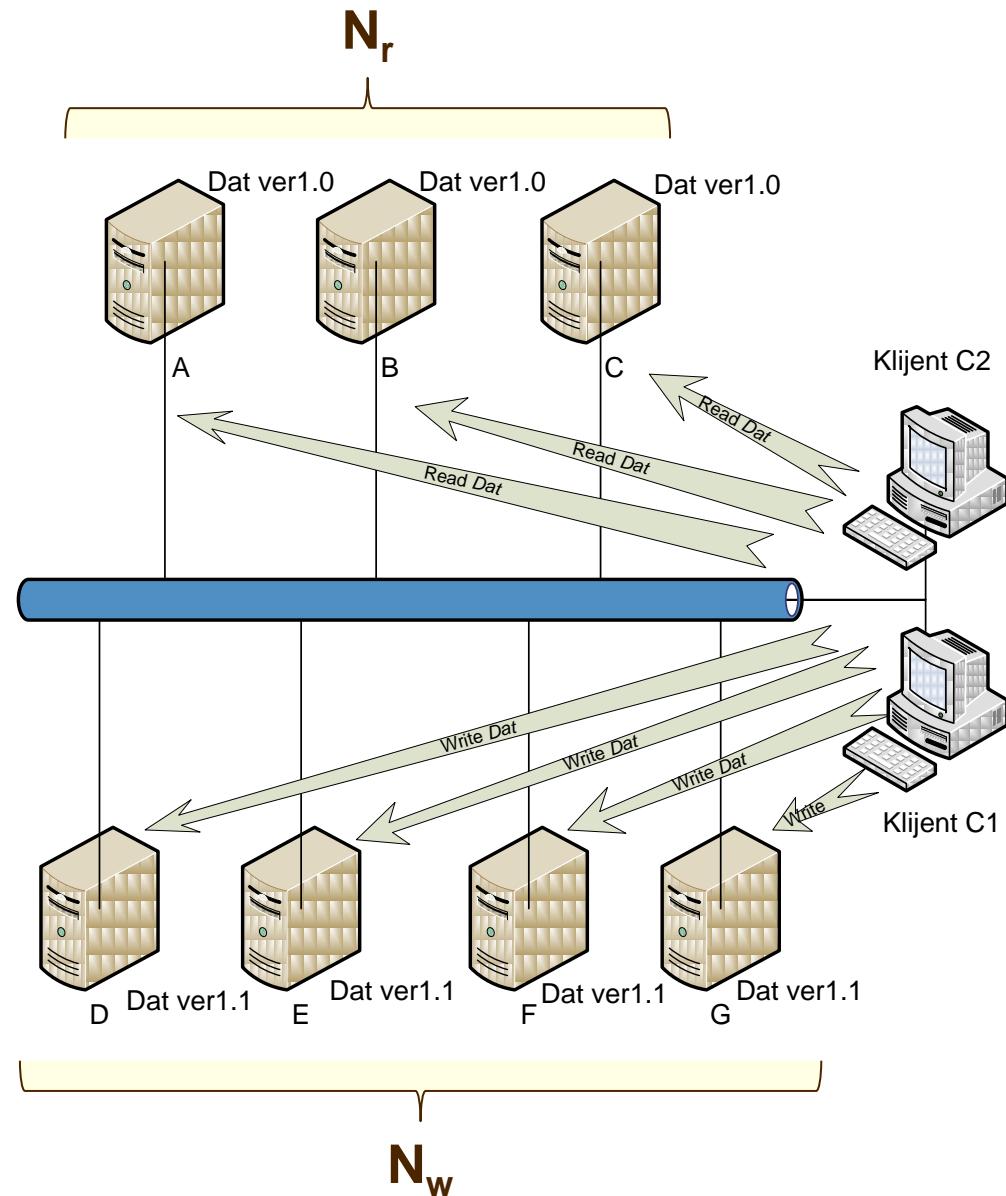
Zadatak 5. Protokol konsenzusa kvorumom – čitanje

- $N_w + N_r > N$
- preklapanje kvoruma za čitanje i pisanje
- → konzistencija čitanja i pisanja!
- → barem 1 poslužitelj u kvoru za čitanje ima najsvježiju verziju datoteke
- $N_w = 4, N_r = 3$



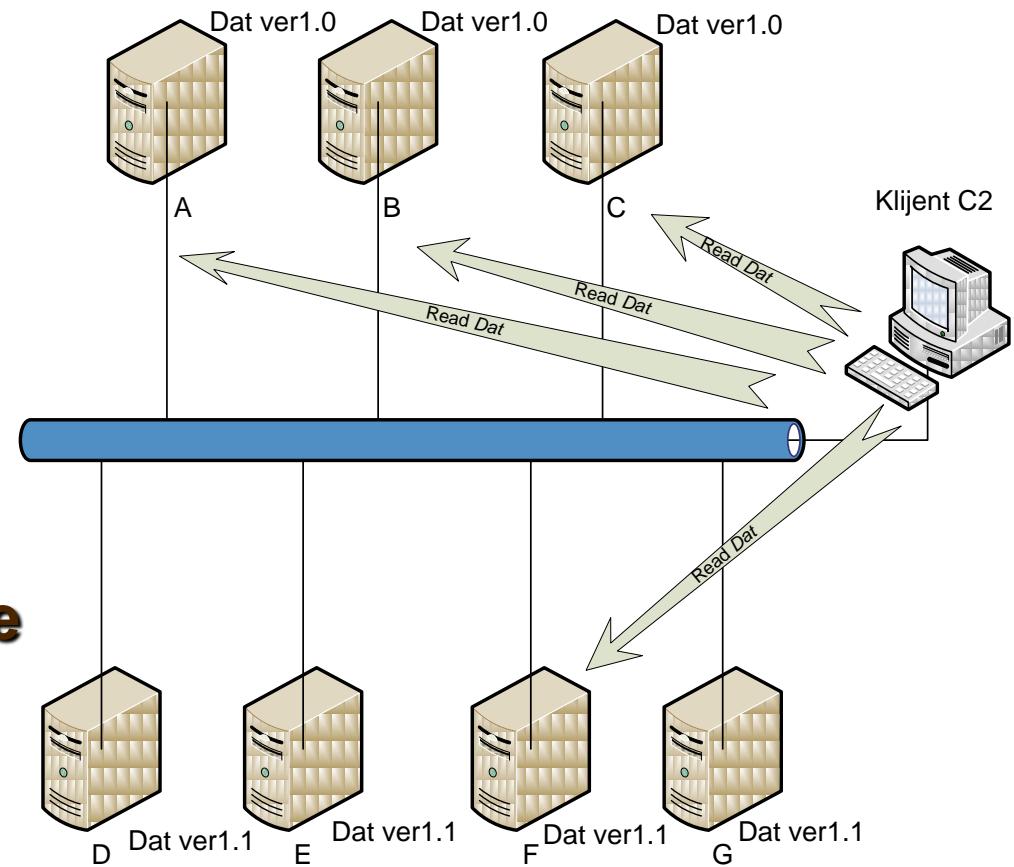
Zadatak 5. Protokol konsenzusa kvorumom – čitanje

- $N_w + N_r > N$
- $N_w = 4, N_r = 3$
- Čitanje → A, B, C
- Pisanje → D, E, F, G
- Obje operacije bile bi dozvoljene
- Čitanje – dohvata stare verzije datoteke



Zadatak 5. Protokol konsenzusa kvorumom – čitanje

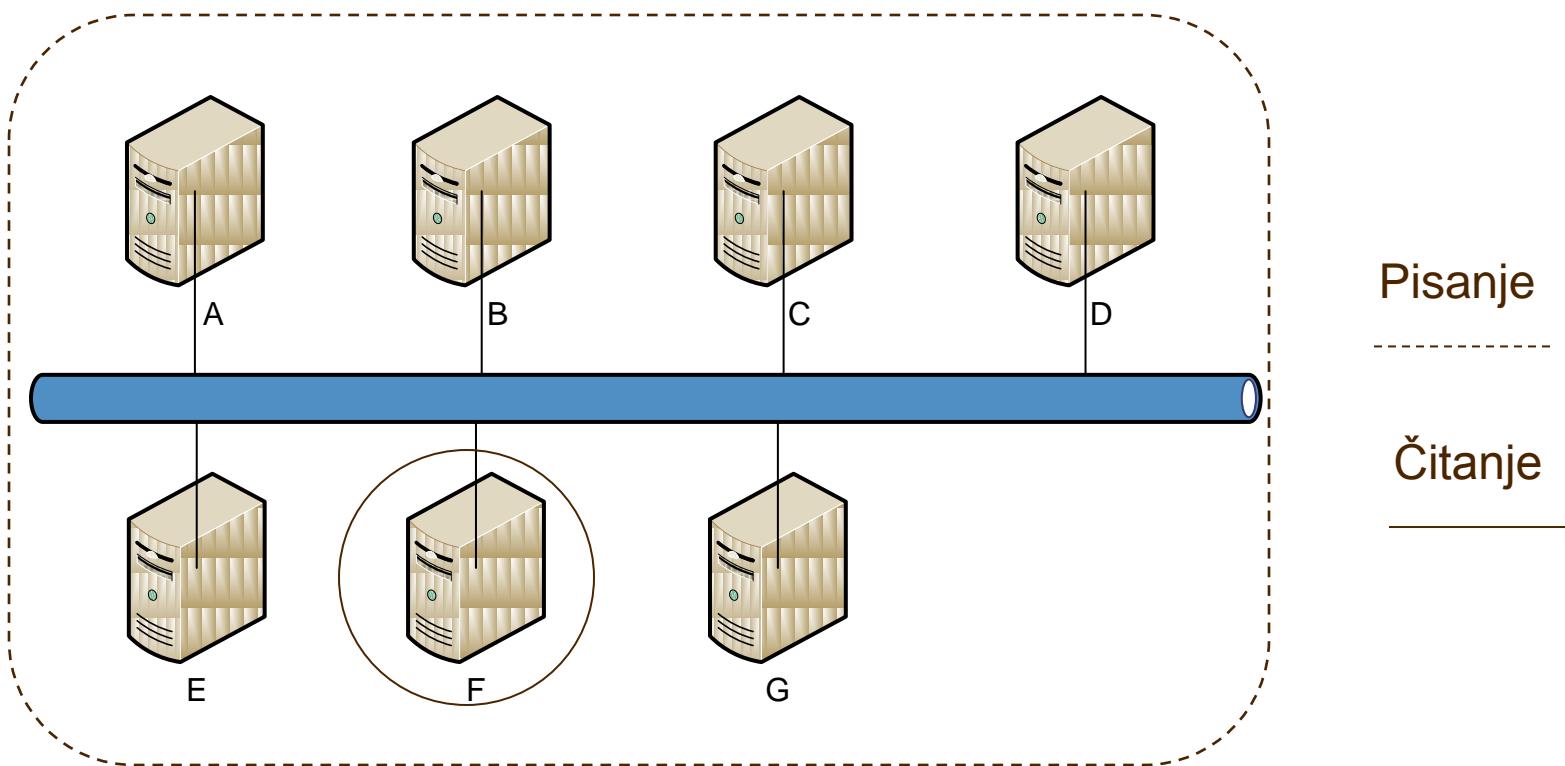
- $N_w + N_r > N$
- $N_w = 4, N_r = 4$
- Čitanje → A, B, C, F
- Pisanje → D, E, F, G



- F – ili čitanje ili pisanje
- F – najsvježa verzija datoteke

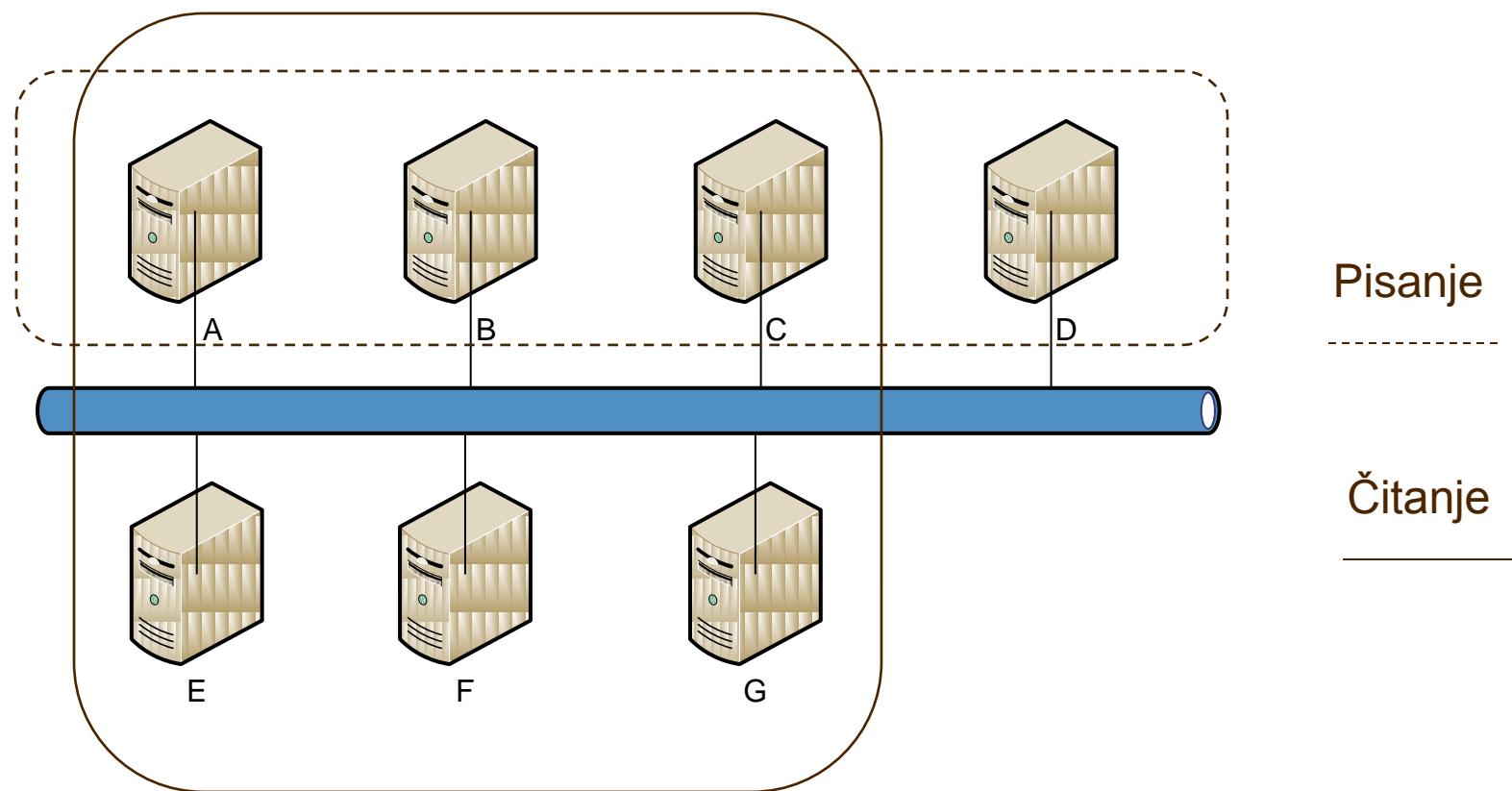
Zadatak 5. Protokol konsenzusa kvorumom - performanse

- $N_r = 1$ – najbolje performanse čitanja
- Da bi $N_r + N_w > 7$
- $N_w = 7$ – osvježavaju se sve kopije



Zadatak 5. Protokol konsenzusa kvorumom

- $N_w = 4, N_r = 6$
 - $N_w + N_r = 4 + 6 = 10 > 7$
 - $N_w = 4 > 3$
- Konzistentno čitanje i pisanje!



Zadatak 6. Oporavak metodom kontrolnih točki

- **Opisati koncept oporavka od pogreške uporabom kontrolnih točki. Za dan program implementiran programskom jeziku C primjeniti opisanu metodu.**

```
#include <stdio.h>
#include <limits.h>

int main() {
    int counter;
    for (counter = 0; counter < INT_MAX; counter++) {
        printf("PID: %d, Ja brojim: %d ! \n", getpid(), counter);
    }
    return 0;
}
```

Zadatak 6. Oporavak metodom kontrolnih točki

- **Uspostavljanje prethodno sačuvanog stanja u slučaju zatajenja**
- **Informacije koje se spremaju:**
 - **Varijable procesa**
 - **Okolina**
 - **Vrijednosti registara**
- **Mogućnosti:**
 - **Potpuno - spremanje čitavog stanja**
 - **Inkrementalno - samo dijelovi koji su promijenjeni od posljednje kontrolne točke**
- **Učestalost:**
 - **Slučajan odabir**
 - **Fiksni intervali**
 - **Broj uspješno obavljenih transakcija**

Zadatak 6. Oporavak metodom KT u raspodijeljenim sustavima

- **Asinkroni oporavak – nekoordinirano među čvorovima**
 - Veća učestalost kontrolnih točki
 - Jednostavnije – koristi se kada je:
 - Vjerojatnost zatajenja manja
 - Ograničena komunikacija među čvorovima
- **Sinkroni oporavak**
 - Koordinirano postavljanje kontrolnih točaka
 - Potreban manji broj kontrolnih točaka

Zadatak 6. Oporavak metodom KT – primjer implementacije

■ Spremanje stanja objekta prije kritične točke

```
try{  
    T old = obj;  
    alternate(obj);  
    if(accept(obj)){  
        return;  
    }  
}  
catch(){  
    obj = old;  
    continue;  
}
```

Zadatak 6. Oporavak metodom KT – primjer implementacije

■ Promjena objekta u slučaju uspješnog obavljanja kritične operacije

```
try{  
    T newObj = alternate(obj);  
    if(accept(newObj)) {  
        obj = newObj;  
        return;  
    }  
}  
} catch() {  
    continue;  
}
```



fija $T \text{ alternate}(T \text{ obj})$ ne mijenja originalni objekt

■ Usporedba implementacija – u slučaju da nije došlo do pogreške:

- Inicijalizacija objekta *old* ✓
- Inicijalizacija objekta *newObj* & pripadavanje *newObj* objektu *obj*

Zadatak 6. Oporavak metodom kontrolnih točki

- **“Za dan program implementiran u C++ programskom jeziku primjeniti opisanu metodu.”**

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int main() {
    int counter;
    for (counter = 0; counter < INT_MAX; counter++) {
        printf("PID: %d, Ja brojim: %d ! \n", getpid(), counter);
    }
    return 0;
}
```

Zadatak 6. Oporavak metodom kontrolnih točki

- **Pohrana vrijednosti varijable counter u datoteku checkpoints.dat**
- **Kontrolna točka – nakon svakih 10 uspješno obavljenih operacija**

```
fd = fopen("checkpoints.dat", "w+");
if(fd != NULL){
    for (counter = 0; counter < INT_MAX; counter++) {
        if((counter % 10) == 0){
            if(fwrite(&counter, sizeof(counter), 1, fd) != 1){
                perror("Fatal! Fatal! Ne mogu zapisati kontrolnu tocku :(");
            }else{
                fflush(fd);
                printf("Kontrolna tocka: %d ! \n", counter);
            }
        }
        printf("PID: %d, Ja brojim: %d ! \n", getpid(), counter);
    }
} else{
    perror("Fatal! Fatal! Ne mogu otvoriti datoteku za zapis kontrolnih tocaka");
    return -1;
}
fclose(fd);
```

Zadatak 6. Oporavak metodom kontrolnih točki

■ Program ubojica.c

```
int main(int argc, char *argv[]){
    int pid = atoi(argv[1]);
    int sendSignal = SIGKILL;
    if (argc > 2 ){
        sendSignal = atoi(argv[2]);
        printf("Saljem %d signal procesu PID = %d \n", sendSignal, pid);
    }else{
        printf("Ubijam proces PID = %d \n");
    }
    srand ( time(NULL) );
    int timeToSleep;
    int timeToSleepMin = 1, timeToSleepMax = 3;
    timeToSleep = rand() % (timeToSleepMax - timeToSleepMin) +
        timeToSleepMin;
    sleep(timeToSleep);

kill(pid, sendSignal);
}
```

Zadatak 6. Oporavak metodom kontrolnih točki

- pretp. da možemo predviđjeti određeni broj pogrešaka (signala)
- rutina za oporavak od pogreške **void checkpointRecovery(int sig)**

```
int main() {
    int i;
    for (i = 2; i < SIGRTMIN; i++) {
        sigset(i, checkpointRecovery);
    }
    printf("Ja sam mali demo za kontrolne tocke PID=%d \n", getpid());
    fd = fopen("checkpoints.dat", "w+");
    if(fd != NULL) {
        for (counter = 0; counter < INT_MAX; counter++) {
            ...
        }
    }
}
```

Zadatak 6. Oporavak metodom kontrolnih točki

■ rutina za oporavak od pogreške **void checkpointRecovery(int sig)**

```
void chechpointRecovery(int sig){  
    sighold(sig);  
    printf("Pocinjem oporavak od pogreske... \n");  
    sleep(3);  
    fseek ( fd , -sizeof(int) , SEEK_END );  
    int result = fread (&counter,sizeof(int),1,fd);  
    if (result != 1) {  
        perror("Fatal! Fatal! Ne mogu citati iz datoteke kontrolnih  
        tocki! ");  
        exit (3);  
    } else{  
        printf("Ja brojim: %d\n", counter);  
    }  
    printf("Uspjesan oporavak od pogreske! \n");  
    sleep(3);  
    sigrelse(sig);  
}
```