# Sisteme distribuite – Teorie
# 10. Toleranta la defecte

# Defecte

- Un sistem are un defect daca nu satisface specificatiile sale
- Gravitate:
  - Un sistem distribuit de emitere de ordine pentru un supermarket – un defect poate rezulta in lipsa la un moment dat a conserve de fasole
  - Intr-un sistem distribuit de control a traficului aerian, un defect poate fi catastrofic
- Tipuri:
  - Defecte de componente
  - Defecte ale sistemelor distribuite

# Defectele componentelor

- Se pot defecta datorita unor erori in procesor, memorie, dispozitiv, cablu, sau software.
- Un *defect* este o functionare proasta, posibil cauzata de
  - Erori de proiectare,
  - Eroare de manufacturare,
  - Eroare de programare,
  - Stricaciune fizica,
  - Deteriorare in timp,
  - Conditii proaste de mediu (a nins pe calculator),
  - Intrari neasteptate,
  - Eroare de operare,
  - Rozatoarele au manca o parte,
  - etc.
- Nu toate defectele duc (imediat) la defecte de sistem, dar unele da.

# Defectele componentelor - clasificare

- *Defecte tranzitorii* apar o data si dispar
  - Daca operatia este repetata defectul dispare
  - O pasare care trece prin dreptul unui transmitator poate cauza o pierdere de biti in anumite retele
  - Daca timpul de transmitere expira si se reincearca, probabil va fi functiona a doua oara.
- Intr-un *defect intermitent*, apare neasteptat, dispare, reapare etc.
  - Un contact care este pierdut va cauza adeasea un defect intermitent.
  - Defectele intermitente cauzeaze probleme grave pentru ca sunt greu de diagnosticat.
  - Tipic, cand apare doctorul, sistemul lucreaza perfect.
- Un *defect permanent* este unul care continua sa existe pana cand componenta este reparata.
  - Chip ars, bug de software, crash de hard disk.

# Scopul proiectarii sistemelor tolerante la defecte

- Asigurarea faptului ca intregul continua sa functioneze corect, chiar si in prezenta defectelor
- Aborarea clasica: analiza statistica a defectelor componentelor electrice
- Pe scurt:
  - Daca o componenta are probabilitatea *p* de functionare proasta intr-o secunda, timpul mediu de defectare este = l/*p*
  - De exemplu, daca probabilitatea de defectare este $10^{-6}$ per secund, timpul mediu de defectare este $10^{6}$ sec sau 1 1.6 zile.

# Defecte ale sistemului

- SD critice: sist trebuie sa supravietuiasca defectelor componentelor (in particular, procesoarelor!), decat sa fie faca acest lucru improbabil.
- Defectele procesoarelor sau caderile pot fi intelese fie ca defecte de procesor sau buguri software.
- Combinatii de defect de procesor cu defecte de linii de comunicare pot fi considerate,
  - Deoarece protocoalele standard pot ajuta recuperarea din erori de linii in modalitati predictibile,

    vom examina numai defecte de procesor!

# Tipuri de defecte de of procesor

- *Defecte silentioase:*
  - Un procesor se opreste si nu raspunde la intrari succesive sau nu produce alte iesiri, exceptand posibil ca nu nu mai produce.
  - Numite de asemenea *defecte-stop*.
- *Defecte bizantine:*
  - Procesorul care produce defect va continua sa ruleze, transmitand raspunsuri gresite la intrebari, posibil lucrand impreuna malitios cu alte procesoare cu defect dand impresia ca lucreaza corect cand nu este cazul
  - Bugurile software nedectare adesea produc defecte bizantine.
  - Tratarea defectelor bizantine este mult mai dificila decat cazul defectelor silentioase.
  - Termenul “bizantin" se refera la imperiul bizantin din perioada 330-1453 din Balcani in care conspiratiile, intrigile si  necredinta erau comune in cercurile de conducere

# Sisteme “sincrone” vs. “asincrone”

- Presupunere:
  - Un sistem in care daca un procesor trimite un mesaj la altul, se garanteaza faptul ca se primeste un raspuns in timpul *T* cunoscut in avans.
  - Defectul de a obtine un raspuns pentru a obtine un raspuns inseamna ca sistemul receptor a esuat.
  - Timpul *T* include timp suficient pentru a trata pierderea mesajelor (transmitandu-le de n ori).
- Un sistem …
  - … care are proprietatea ca intotdeauna sa existe un raspuns la un mesaj intr-o margine cunoscuta daca este functional se spune ca este *sicron*.
  - … neavand aceasta proprietate se spune ca este *asincron*.
- Aceasta terminolgie intra in conflict cu utilizarea traditionala a lor; este utilizat in masura mare in tolerarea defectelor.
- Sistemele asincrone sunt mai complicat de tratat dect cele sincrone
  - Daca un procesor poate trimite un mesaj si cunoastre absenta raspunsului in T secunde inseamna ca recipientul s-a defectat -> poate lua o actiune corectiva
  - Nu exista o limita superioara pentru cat este necesar rapsunsului sa fie receptionat -> determinarea daca a existat un defect va fi o problema.

# Utilizarea redundantei ca abordare generala

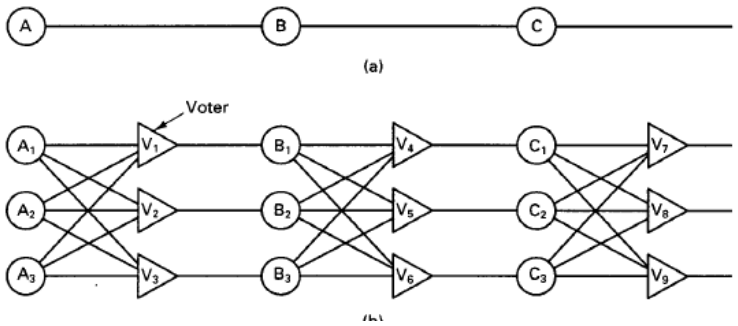
1. Redundanta informatiei
   - Biti suplimentari sunt adaugati pentru a permite recuperarea bitilor pierduri
   - E.g. un cod Hamming poate fi adaugat pentru transmiterea datelor pentru a permite recuperarea din zgomotele de pe linia de transmitere
2. Redundanta in timp,
   - Este efectuata o actiune, si apoi, daca este necesar, este efectuata din nou.
   - Exemple: utilizarea tranzactiilor atomice – daca tranzactia este abortata, poate fi re-efectuata fara a produce probleme.
   - Util in mod special cand defectele sunt tranzitorii sau intermitente.
3. Redundanta fizica

# Redundanta fizica

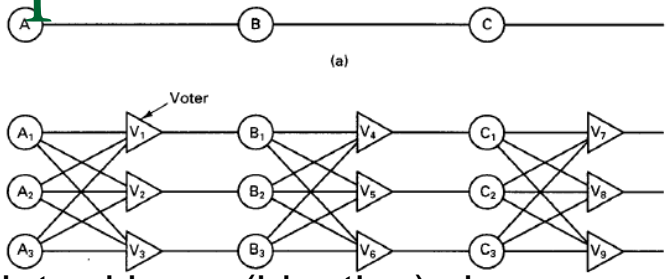
- Echipament suplimentar este adaugat pentru a face posibil ca intregul sistem sa tolereze pierderea sau functionarea proasta a anumitor componente.
- Exemplu: procesoare suplimentare pot fi adaugate la sistem astfel incat numai o parte din ele se defecteaza, sistemul poate inca functiona corect.
- Doua modalitati de organizare acestor procesoare extra:
  - Replicare activa
  - Backup primar.
- Exemplu: cazul unui server.
  - Cand este utilizata replicarea activa, toate procesoarele sunt utilizate tot timpul ca servere (in paralel) pentru a aascunde complet defectele
  - Schema de backup primar utilizeaza doar un procesor, inlocuindu-l cu cel de backup cand se defecteaza.

# Tolerarea defectelor utilizand replicarea activa

- Utilizata in
  - biologie (mamiferele au doi ochi, dou urechi, doi plamani, etc.),
  - avion (Boing 747 are patru motoare dar poate zbura cu trei), and
  - sport (referinte multiple in cazul pierderii unui eveniment).
- Anumiti autori se refera la replicarea activa ca fiind abordarea starii masinii
- Utilizata pentru tolerarea defectelor in circuite electrice
  - Circuitul in (a):
    - Semnalul trece prin dispozitivele A, B si C, in secventa.
    - Daca unul esueaza, rezultatul final va fi probabil gresit.
  - Circuitul in (b),
    - Fiecare dispozitiv este replicat de trei ori.
    - In fiecare etapa din circuit este un votant tri-plicat.
    - Fiecare votant este un circyit cu trei intrari si o iesire.
    - Daca doua sau trei dintre intrari sunt aceleasi, iesirea este egala cu acea intrare.
    - Daca toate intrarile sunt diferite, iesirea este nedefinita
    - Acest tip de proiectare este cunoscut sub numele de TMR (Triple Modular Redundancy).



# Redundanta modulara tripla




- Presupunem ca elementul A2 se defecteaza:
  - Fiecare dintre votati, V1, V2 si V3 primesc dou intrari bune (identice) si una gresita, si fiecare dintre ele scot valoarea corecta in etapa a doua.
  - In esenta, efectul defectului lui A2 este complet mascat, a.i. intrarile la B1, B2 si B3 sunt exact la fel ca si cand n-ar fi existat defect.
- Sa consideram in plus ca si B3 si C1, produc defect, in plus fata de A2.
  - Aceste efecte sunt de asemenea mascate, a.i. cele trei iesiri finale sunt corecte.
- La prima vedere nu este evident de ce sunt necesari trei votanti la fiecare etapa.
  - De fapt, un singur votant pote detecta si pasa informatia majoritara.
    - Totusi, un votant este de asemenea o componenta care poate esua.
  - Pp., de exemplu, ca V1 functioneaza prost.
    - Intrarea B1 va fi gresita, insa B2 & B3 vor produce aceeasi iesire si V4, V5 & V6 vor produce toate rezultatul corect in etapa 3.
  - Un defect in V1 nu este diferit de un defect in B1 care produce o iesire gresita, dar in ambele caz nu este votat mai tarziu.
- TMR poate fi aplicat recursiv,
  - De exemplu, pentru a face un cip de incredere prin utilizarea TMR in acesta

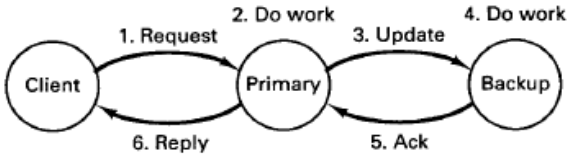
# Cata replicare este necesara?

- Raspunsul depinde de cantitatea de tolerare a erorii care este dorita
- Un sistem se spune ca este *k-tolerand la defecte* daca poate supravietui defectelor in *k* componente si inca sa corespunda specificatiilor
- Daca componentele, fie procesoarele, esueaza silentios,
  - Atunci este suficicinet sa exista *k* + 1 procesoare pentru a oferi *k*-toleranta la defecte.
    - Daca *k* dintre ele se opresc, atunci raspunsul de la cel ramas poate fi.
- Daca procesoarele au defecte bizantine, continuand sa ruleze si cand sunt defecte si trimit reapsunsuir eronate sau aleatore,
  - Un minim de 2*k* + 1 procesare sunt necesare pentru a atinge *k*-toleranta la defecte.
    - In cel mai rau caz, *k* procesoare defecte pot genera accidental (sau chiar intentionat) acelasi raspuns.
    - Totusi, cele *k* + 1 ramase vor produce acelalasi rapsuns, a.i. clientul sau votantul se poate increde in majoritate.

# Tolerarea defectelor utilizand backup primar

- Ideea esentiala a metodei de backup primara este aceea ca exista un singur  server care este primar si realizeaza ceea ce se cere
- Daca acest primar es defecteaza, backupul preia sarcinile.
- Ideall, acesta schimbare trebuie sa se faca intr-o modalitate cat mai clara si vzibila numai sistemelui de operare al clientului, nu programelor aplicatii.
- Aceasta schema este des utilizata in lumea reala:
  - Exemple: guvernare (Vice-presedinte), aviatie (co-pilot), automobil (roata de rezerva), generatoare de rezerva in camerele de operatii din spitale.
- Toleranta da defecte bazate pe primar-backup are doua avantaje majore fata de replicarea activa:
  - Este mai simpla in timpul operatiilor normale pentru ca mesajele merg inspre un server (primarul) si nu catre intregul grup.
  - In practica este necesar un numar mic de masini, deoarece la un moment dat este necesar un primar si un secundar
- Dezavantaj
  - Lucreaza slab in prezenta defectelor bizantine in care primarul declara in mod eronat ca lucreaza bine.
  - Recuperea dintr-o defectare a primarului poate fi complex si consumatoare de timp.

# Un protocol simplu primar-backup pt. o operatie de scriere




- Un client expediaza un mesaj catre primar, care realizeaza sarcina si apoi expediaza un mesaj de actualizare catre backup.
- Cand backupul primeste mesajul, efectueaz sarcina si expediaza un mesaj de confirmare catre primar.
- Cand vine confirmarea, primarul expediaza mesaj de raspuns la client.
- Efectul caderii primarului in diferite momente ale RPC?
  - Daca primarul cedeaza inainte sa efectueze sarcina (pas 2), nu se strica nimic.
    - Clientul va intra in time out si va reincerca.
    - Daca incearca destul, si eventual va reusi sa se efectueze sarcina o singura data.
  - Daca primarul cedeaza dupa ce efectueaza sarcina dar dupa expedierea actualizarea, cand backupul preia si cerintele vin din nou,
    - Lucrul va fi realizat de doua ori – daca acesta are efecte secundare, poate fi o problema.
  - Daca primarul cedeaza dupa pasul 4 dar inainte de pasul 6,
    - Lucrul se poate termina efectuandu-se de trei ori, odat la primar, odata la backup ca rezultta al pasului 3, si odata ce backupul devine primar.
  - Daca cerinntele poarta identificatori -> posibil sa fie asigurat ca sarcina este realizata numai de doua ori
    - Asigurarea ca este realizat numai o singura data este imposibil.

# Cand sa se treaca de la primar la backup?

- Backupul poate sa trimita mesaje: “Esti viu?" periodic catre primar.
- Daca primarul nu raspunde intr-un anumit timp, backupul poate prelua.
- Daca primarul nu s-a defectat, dar e mai incet,
  - Intr-un sistem asincron nu exista nici o modalitate de a distinge intre un primar incet si unul care s-a defectat.
  - Solutia cea mai buna este un mecanism hardware in care backupul poate opri sau reboota primarul.
  - Toate schemele primar-backup necesita un acord, care este greu de atins, pe cand replicarea activa nu cere intotdeauna un protocol pentru acord (ex. TMR).
  - O alta solutie este utilizarea un disk cu port dual intre primar si secundar.
    - Cand primarul obtine o cerere, scrierea cererea pe disk inainte de a efectua sarcina si de asemenea scrie rezultatele pe disk.

# Acord in sistemele cu defecte

- In numeroase SDuri exista o necesitate de a ajunge la un acord asupra unui lucru
  - Exemple sunt: alegerea unui coordonator, decizia de a efectua o tranzactie sau nu, impartirea sarcinilor intre lucratori, sincronizare, etc.
- Scopul algoritmilor de acord distribuit: pentru a face ca toate procesoarele defectuoase sa ajunga la consens in anumite teme, dupa un numar finit de pasi.
- Cazuri diferite sunt posibile depinzand de parametrii sistemului, incluzand:

1. Sunt mesajele livrate cu incredere tot timpul?
2. Pot procesele sa cedeze, si in acest caz, se defecteaza silentios sau bizantin?
3. Este sistemul sincron sau asincron?

# Cazul simplu

- Procesoare perfecte + Liniile de comunicare pot pierde mesaje
- O problema faimoasa, cunoscuta ca ***problema celor doua armate***,
  - Ilustreaza dificultatea de a pune doua procesoare perfecte de acord asupra unui singur bit de informatie

  - Armata rosie, cu 5000 de ostasi, este campata intr-o vale.
  - Doua armate albastre, fiecare cu 3000 de ostasi, sunt campate in jurul dealurilor inconjuratoare care domina valea.
  - Daca cele doua armate albastre pot sa-si coordoneze atacul aupra armatei rosii, vor fi victorioase.
  - Totusi, daca ataca singure, vor fi macelarite.
  - Scopul armatelor albastre este de a ajunge la un acord privind atacul.
  - Problema este aceea ca pot comunica numai pe un canal nesigur: expediind un mesager care este susceptibil de a fi capturat de armata rosie.

# Examplu pt.problema celor doua armate

- Comandantul armatei albastre 1, Gen. Alexandru, trimite un mesaj la comandantul armatei albastre 2, Gen.Bonaparte: “Intentionez sa atac – hai sa atacam in zori de zi."
- Mesagerul trece si Bonaparte il trimite inapoi cu o nota spunand: “Idee splendida, Alex. Ne vedem maine in zori."
- Mesagerul ajunge cun bine la baza sa, livreaza mesajul si Alexander spune trupelor sale sa se pregateasca pentru batalie in zori.
- Totusi, mai tarziu in acea zi, Alexandru isi da seama ca  Bonaparte nu stie ca mesagerul a ajuns inapoi in siguranta si necunoscand aceasta, poate nu va indrazni sa atace.
- In consecinta, Alexandru spune mesageruli sa mearga sa-l spuna lui Bonaparte ca mesajul sau (a lui Bonaparte) a ajuns si batalia este pregatita.
- Din nou mesagerul trece si livreaza raspunsul.
- Dar acum Bonaparte se ingrijoreaza ca Alexandru nu conoaste ca raspunsul a sosit. Bonaparte gandind ca partnereul poate gandi ca mesagerul a fost capturat si poate sa nu fie sigur de planul de atac, asa incat trimite mesagerul inapoi.

!!! *Chiar daca mesagerul a reusit sa treaca de fiecare data, este simplu sa fie demonstrat ca Alexandru si Bonaparte nu vor ajunge niciodata la un acord, indiferent de numeroarele raspunsuri care le trimit* !!!

# Analiza exemplului celor doua armate

- Pp. ca exista un anumit protocol care se termina intr-un numar finit de pasi.
- Inlatura orice pas extra de la sfarsit si se poate obtine un prtocol care functioneaza.
- Un anumit mesaj este in acest caz ultimul si este esential pentru acord (deoarece este protocolul minim).
- Daca mesajul nu ajunge din cauza unui defect, razboiul inceteaza.
- Expeditorul ultimului mesaj nu cunoaste daca ultimul mesaj a ajuns.
- Daca nu stie, protocolul nu este complet si celalalt general nu va ataca.
- Astfel expeditorul ultimului mesaj nu poate cunoaste ca razboiul este planificat sau nu, sii deci nu poate implica trupele sale.
- Deoarece receptorul ultimului mesaj cunoaste ca expeditorul nu poat fi sigur, nu va risca a posibila moarte, si nu exista un acord.

!!!! *Chiar cu procesoare ne-defectuoase (generali), acordul intre chiar si numai doua procese nu este posibila in cazul unei comunicare care nu este de incredere.* !!!

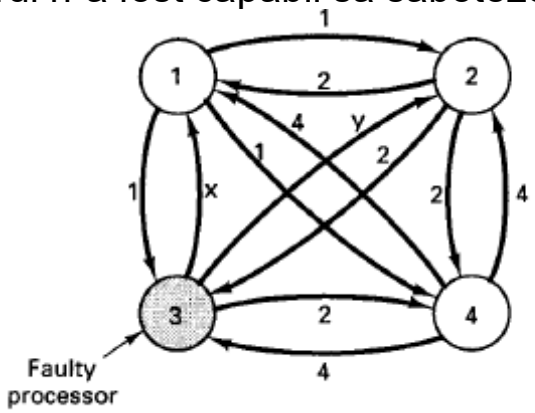
# Caz secund

- Comunicarea este perfecta, dar procesoarele nu-s.
- Problema clasica si in acest caz apare dintr-o perspectiva militara si este numita ***problema generalilor bizantini***.
  - Armata rosie este si in acest caz campata in vale, dar exista *n* generali albastrii care conduc armate aflate pe dealurile apropriate.
  - Comunicarea este realizata prin telefon si este perfecta,
  - Dar *m* generali sunt tradatori (defectuosi) si incearca activ sa previna generalii loiali sa ajunga la un acord prin furnizarea de informatii contradictorii si incorecte (modeleaza proc.care functioneaza prost)
  - Intrebare: pot generalii loiali sa ajunga la un acord?
  - Fiecare general stie numarul de ostasi pe care-l are.
  - Scopul problemei: pentru generali sa schimbe numarul de ostasi, a.i. la sfarsitul algoritmului, fiecare general are un vector de lungime *n* corespunzand tuturor trupelor albastre.
  - Daca generalul *i* est loial, atunci elementul *i* este valoare corecta a trupei sale; altfel, este  nedefinit.

# Algoritmul recursiv a lui Lamport (1982)

- Exemplu pentru $n = 4$ si $m = 1$ => 4 pasi

1. Fiecare general expediaza un mesaj (de incredere) la toti generaill anuntand nr.ostasilor sai.
   - Generalii loiali spun adevarul; tradatorii pot spune fiecarui general o minciuna diferita.
   - (a) vedem ca generalul 1 raporteaza 1K, generalul 2 raporteza 2K, generalul 3 minte pe fiecare, dand *x, y,* si *z*, iar generalul 4 raporteaza 4K.
2. Rezultatele anunturilor din pasul 1 sunt colectate impreuna in vectorul (b)
3. Pasul 3 consta in pasarea de catre fiecare general a vectorului sau de la (b) la fiecare alt generalother general.
   - generalul 3 minte, inventand 12 valori noi,
   - Rezultatele pasului 3 sunt aratate in (c).
4. Pasul 4: fiecare general examineaza elementul *i*lea pentru fiecare a din vectorii noi primiti.
   - Daca valoare are o majoritate, valoarea este pusa in vectorul rezultat.
   - Daca nici o valoare nu are majoritatea, elemnetul corespunzator este marcat ca UNKNOWN.
   - (c): generalii 1, 2, si 4 ajung la acord asupra (1, 2, UNKNOWN, 4), rezultatul corect.
   - Tradatorul n-a fost capabil sa saboteze decizia.



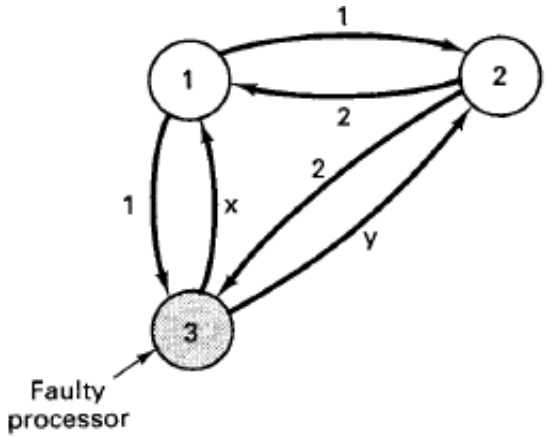

(a)

| 1 Got (1, 2, x, 4) | 1 Got | 2 Got | 3 Got |
|---|---|---|---|
| 2 Got (1, 2, y, 4) | (1, 2, y, 4) | (1, 2, x, 4) | (1, 2, x, 4) |
| 3 Got (1, 2, 3, 4) | (a, b, c, d) | (e, f, g, h) | (1, 2, y, 4) |
| 4 Got (1, 2, z, 4) | (1, 2, z, 4) | (1, 2, z, 4) | (i, j, k, l) |

(b) (c)

# Alt exemplu

- pentru $n = 3$ si $m = 1$, adica, numai 2 generali loiali si 1 tradator,
- In (c): nici un general loial nu vede o majoritate pentru elementul 1, elementul 2, sau elementul 3, a.i. toate sunt marcate UNKNOWN.
- Algoritmul a esuat sa produca un acord.




(a)

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

(b)

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

(c)

# Cazul general

- Lamport a demonstrat ca un sistem cu *m* procesoare defectuoase, acordul poate fi atins numai daca 2*m* + 1 procesoare corect functionale sunt prezente, dintr-un total de 3*m* + 1.
- *Acordul este posibil numai daca mai mult decat doua treimi din procesoare lucreaza corect.*
- Mai rau: Fischer a demonstrat ca intr-un SD

  cu procesoare asincrone + intarzieri nemarginite in transmitere,

  nu este posibil un acord chiar daca numai un procesor este defectuos (chiar daca procesorul se defecteaza silentios).
  - Problema sistemelor asincrone este aceea ca procesoarele arbitrar incete nu se disting de cele care sunt oprite.
- Numeroase rezultate teoretice sunt cunoscute pentru cazuri in care acordul este posibil si cand nu este posibil.