Sinkronizazio banatua

Sistema Banatuak

Mikel Larrea, KAT Saila

Sinkronizazio banatua

- Sarrera
- Elkarrekiko esklusioa
- Hautapen-algoritmoak
- Talde-komunikazioa
- Erreplikazioa
- Transakzio banatuak
- Adostasunaren arazoa (*Consensus*)

1 Sarrera

- Aplikazio banatu gehienetan sinkronizazioaren beharra dago. Adibideak:
  - baliabide banatuen aldi bereko atzipenaren kudeaketa (elkarrekiko esklusioa)
  - prozesu koordinatzailearen hautaketa, hutsegite baten ondoren adibidez (hautapen-algoritmoak)
  - hedapenak - *multicast* -, sendotasun semantika desberdinak bermatuz (talde-komunikazioa)
  - erreplikazioa, hutsegite-tolerantzia lortzeko. Kopia guztien sendotasuna bermatzeko estrategiak behar dira
  - transakzio banatuak: prozesuen arteko adostasuna

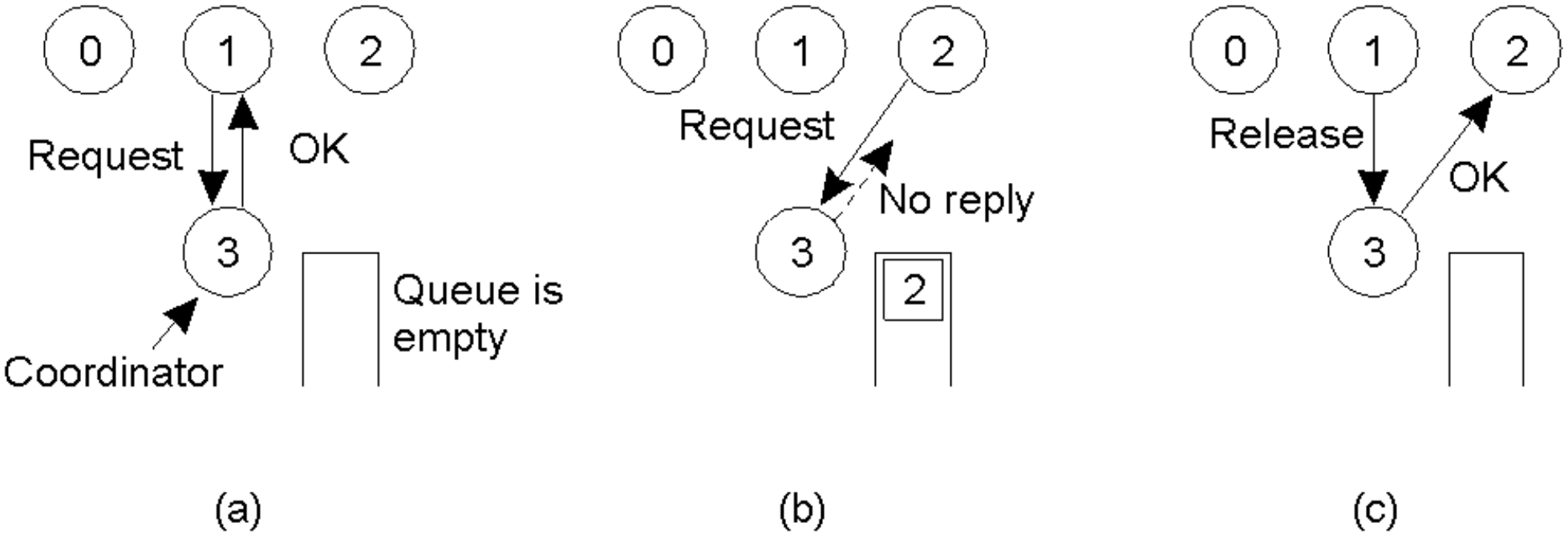
2 Elkarrekiko esklusioa

- Eredu sinplifikatua:
  - prozesu talde batek baliabide bakarra konpartitzen du
  - aldiko prozesu bakar batek erabil dezake baliabidea
- Arazoaren espezifikazioa (baldintzak):
  - i) prozesu bat baliabidea erabiltzen ari bada, askatu egin beharko du beste prozesu bati eman baino lehen
  - ii) baliabidea lortzeko eskariak egindako ordena (kausala) errespetatuz zerbitzatu behar dira
  - iii) baliabidea lortzen duten prozesu guztiek erabili ondoren askatu egiten badute, orduan eskari guztiak lehentxeago edo geroxeago zerbitzatuko dira

2 Elkarrekiko esklusioa (2)

- Algoritmo zentralizatuetan, sekzio kritikoa koordinatzaile prozesu batek kudeatzen du
  - sekzio kritikoan sartzeko eskaria:
    - koordinatzaileari mezua bidali eta konfirmazioaren zain gelditu
    - baliabidea libre badago, koordinatzaileak konfirmazio mezua bidaliz erantzuten du (konfirmazio mezua = baimena)
    - okupaturik badago, koordinatzaileak ez du erantzuten (edota ukatze mezua bidaltzen dio), eskaria kola baten sartuz
  - sekzio kritikotik irteterakoan:
    - koordinatzaileari mezua bidali irten dela abisatuz
    - sartzeko zain dauden prozesuen kola hutsik ez badago, lehendabizikoari konfirmazio mezua bidali, kolatik kanporatuz

Mutual Exclusion: A Centralized Algorithm



a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted

b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.

c) When process 1 exits the critical region, it tells the coordinator, which then replies to 2

2 Elkarrekiko esklusioa (3)

- Algoritmo zentralizatuen analisia:
  - eskalagarritasun eskasa: sistema handitan koordinatzailea itogunea bihur daiteke
  - hutsegite-tolerantzia eskasa: koordinatzaileak huts egiten badu sistema gelditu egiten da
    - ukatze mezurik gabe, huts egindako koordinatzailea ezin da desberdindu baimenik ematen ez duen batekin
  - eskarien ordena (*FIFO*): kausaltasuna ez da bermatzen
- Algoritmo banatuetan ez dago koordinatzailerik:
  - Ricart eta Agrawala-ren algoritmoa
  - Eraztunaren algoritmoa (*token ring*)

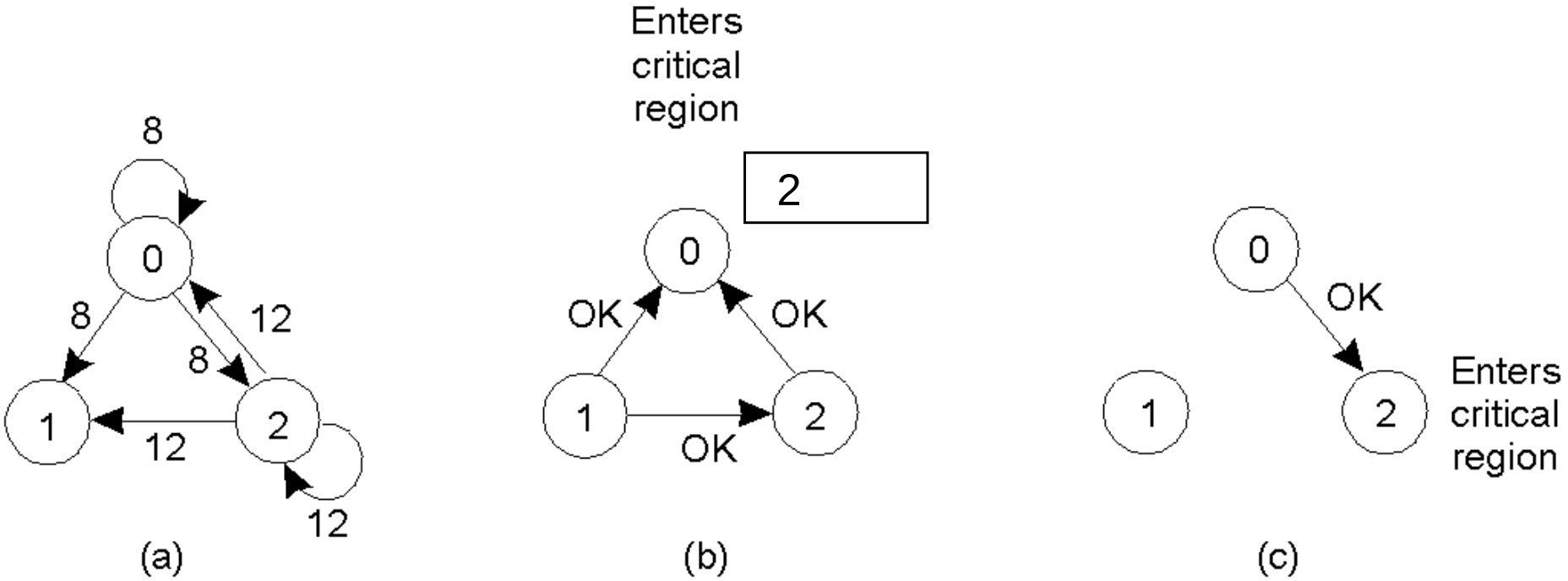
2 Elkarrekiko esklusioa (4)

- Ricart eta Agrawala-ren algoritmoa (1981):
  - gertaeren ordena behar du (erloju logikoak, indizeak)
  - sekzio kritikoan sartzeko eskaria:
    - prozesu guztiei mezu bat bidali: indizea + denbora marka
    - prozesu batek mezu hau jasotzean:
      - sekzio kritikoan ez badago, konfirmazio mezua bidaltzen dio
      - sekzio kritikoan badago, ez du erantzuten (edota ukatze mezua bidaltzen dio) eta eskaria kola baten sartzen du
      - sekzio kritikoan sartzeko zain badago, bere eskariaren denbora markarekin konparatzen du. Berea berriagoa bada, konfirmazioa bidaltzen dio; aldiz zaharragoa bada, ez du erantzuten (edota ukatze mezua bidaltzen dio) eta eskaria kola baten sartzen du
    - sekzio kritikoan sartzeko beste prozesu guztien konfirmazioa lortu behar da lehendabizi

2 Elkarrekiko esklusioa (5)

- Ricart eta Agrawala-ren algoritmoa (1981):
  - sekzio kritikotik irteterakoan:
    - kolan dituen prozesu guztiei konfirmazio mezua bidali, kolatik kanporatuz
- Analisia:
  - *2(n-1)* mezu (*n* hedapenerako euskarria balego)
  - edozein hutsegitek sistema blokeatzen du
    - ukatze mezuak bidaltzen badira, denbora-mugak (*timeout*) lagundu dezakete
  - talde dinamikoak: berrien sarrerak, irteerak, hutsegiteak (algoritmoaren exekuzioarekin bat gertatzen direlarik)
    - talde-komunikazioa lagundu dezake

Mutual Exclusion: A Distributed Algorithm



a) Two processes want to enter the same critical region at the same moment.

b) Process 0 has the lowest timestamp, so it wins.

c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

Ricart and Agrawala's algorithm

On initialization
    state := RELEASED;
To enter the section
    state := WANTED;
    Multicast *request* to all processes;
    T := request's timestamp;
    Wait until (number of replies received = (*N* – 1));
    state := HELD;

request processing deferred here

On receipt of a request $<T_i, p_i>$ *at* p_j $(i \neq j)$
    if (*state* = HELD or (*state* = WANTED *and* $(T, p_j) < (T_i, p_i)$))
    then
        queue *request* from p_i without replying;
    else
        reply immediately to p_i;
    end if
To exit the critical section
    state := RELEASED;
    reply to any queued requests;

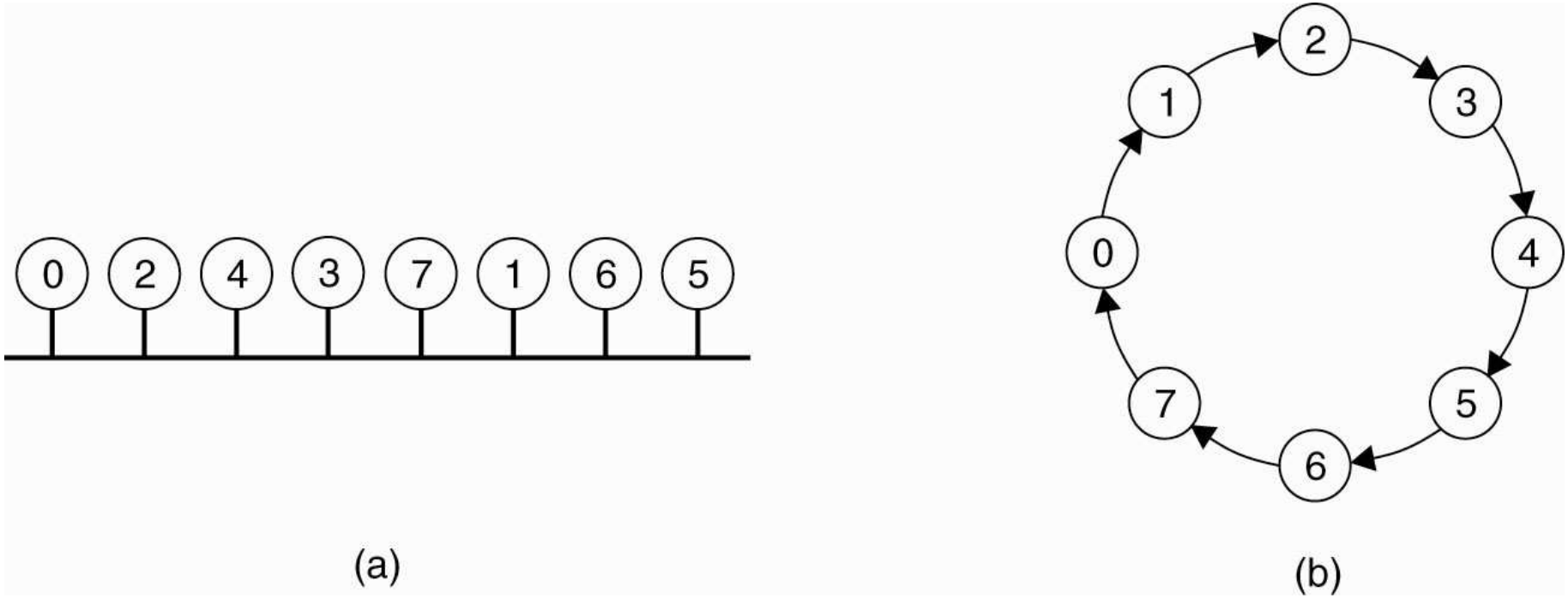
2 Elkarrekiko esklusioa (6)

- Tanenbaum-en liburutik (269. orria):
  - *“Ricart-Agrawala’s mutual exclusion algorithm is slower, more complicated, more expensive, and less robust than the original centralized one. Why bother studying it under these conditions? For one thing, it shows that a distributed algorithm is at least possible, something that was not obvious when we started. Also, by pointing out the shortcomings, we may stimulate future theoreticians to try to produce algorithms that are actually useful. Finally, like eating spinach and learning Latin in high school, some things are said to be good for you in some abstract way.”*

2 Elkarrekiko esklusioa (7)

- Eraztunaren algoritmoa (*token ring*):
  - prozesuen artean eraztun logikoa eraikitzen da, zentzu batean mezu berezi bat (testigua, *token*-a) transmitituz
    - hasieran, lehendabiziko prozesuak du testigua
    - testigua duen prozesuak sekzio kritikoan sartu nahi ez badu, testigua hurrengo prozesuari pasatzen dio
    - testigua duen prozesuak sekzio kritikoan sartu nahi badu, sartu egiten da, bertatik irten arte testigua kontserbatuz. Irteterakoan, testigua hurrengo prozesuari pasatzen zaio
    - sekzio kritikoan sartu nahi duen prozesu batek testigua izan arte itxaron egiten du
  - ahuleziak: testiguaren galera, hutsegiteak, kausaltasuna

Mutual Exclusion: A Token Ring Algorithm



a) An unordered group of processes on a network.

b) A logical ring constructed in software.

Mutual Exclusion Algorithms: Comparison

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 to 2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

A comparison of three mutual exclusion algorithms.

3 Hautapen-algoritmoak

- Zerbitzu banatu askok koordinatzaile prozesu baten oinarritzen dira. Adibideak:
  - elkarrekiko esklusio zentralizatua
  - transakzio banatuak
  - erreplikazioa (*primary-backup* ereduan)
  - Zerbitzu hauetan, koordinatzailearen hutsegitea detektatu behar da, berria hautatzeko
- Hautatzen den prozesua bakarra izan behar da beti
- Suposaketa: prozesuak guztiz ordenaturik daude
- Adibideak: borrokazalearen algoritmoa, eraztunaren algoritmoak

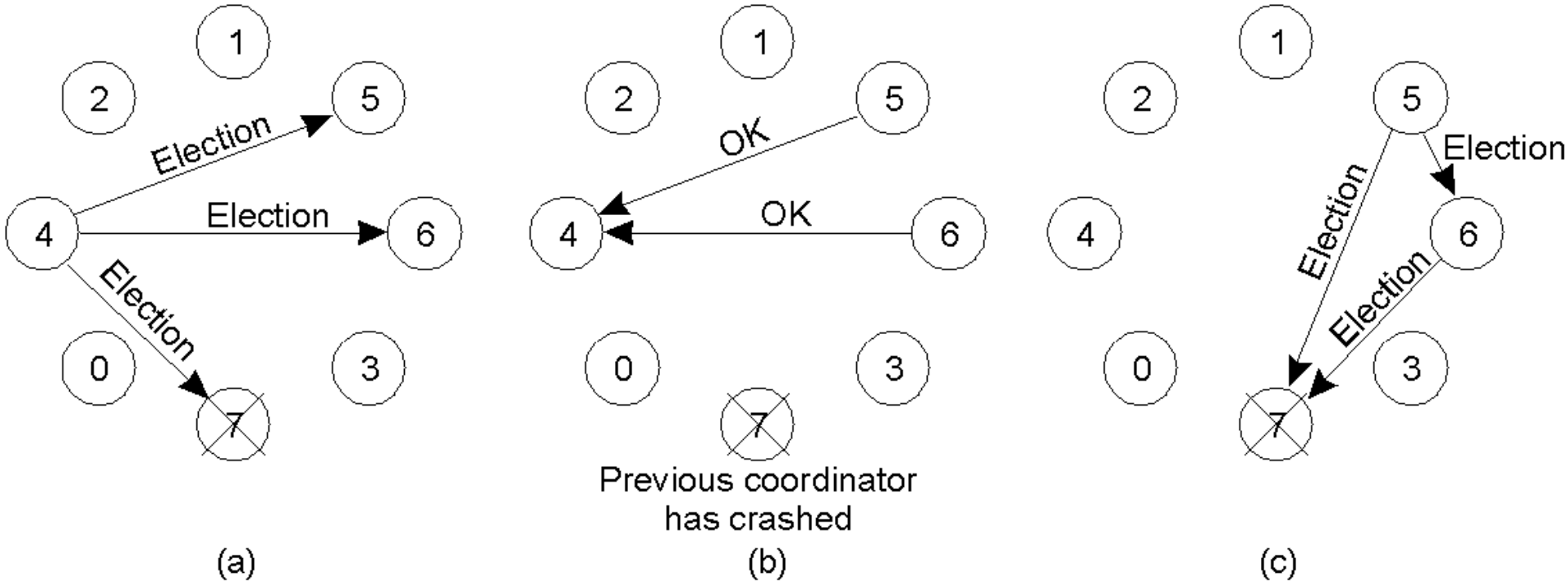
3 Hautapen-algoritmoak (2)

- Eredua eta baldintzak:
  - edozein prozesuk has dezake hautapen-algoritmoa
    - prozesu batek aldiko hautapen bakarra has dezake. Printzipioz, N prozesuek N hautapen konkurrente has dezakete
  - prozesuek hautapen baten partaideak ala ez partaideak dira
  - p_i prozesu bakoitzak $hautatua_i$ aldagaia du. Hautapen baten partaide izatera pasatzen denean, $hautatua_i = \perp$ ipiniko du, oraindik inor hautatu ez duela adierazteko
  - i) partaide bakoitzak $hautatua_i = \perp$ edota $hautatua_i = P$ izango du, azken kasuan P hutsegin ez duten prozesuen artean indizerik handiena duena delarik
  - ii) prozesu guztiek parte hartzen dute hautapenean, azkenean $hautatua_i \neq \perp$ ipiniz, edota huts egiten dute

3 Hautapen-algoritmoak (3)

- Borrokazalearen algoritmoa (García-Molina, 1982)
  - hautapena hasten duen prozesuak bera baino indize handiagoa dutenei hautapen-mezua bidaltzen die, aukeragarri bihurtuz
  - inork ez badio erantzuten, berak irabazten du eta koordinatzaile berria izatera pasatzen da, prozesu guztiei hautatua-mezua bidaliz hori adierazteko
    - prozesu batek hautatua-mezua jasotzerakoan, mezuan adierazten den prozesua koordinatzaile bezala ezartzen du
  - prozesu batek hautapen-mezua jasotzerakoan (indize txikiagoa duen prozesu batengandik), erantzun-mezua bidaltzen dio, eta bera bihurtzen da aukeragarri (hautapen-mezua bera baino indize handiago dutenei bidaliz...)
  - azkenean gelditzen den prozesu bakarrak irabazten du

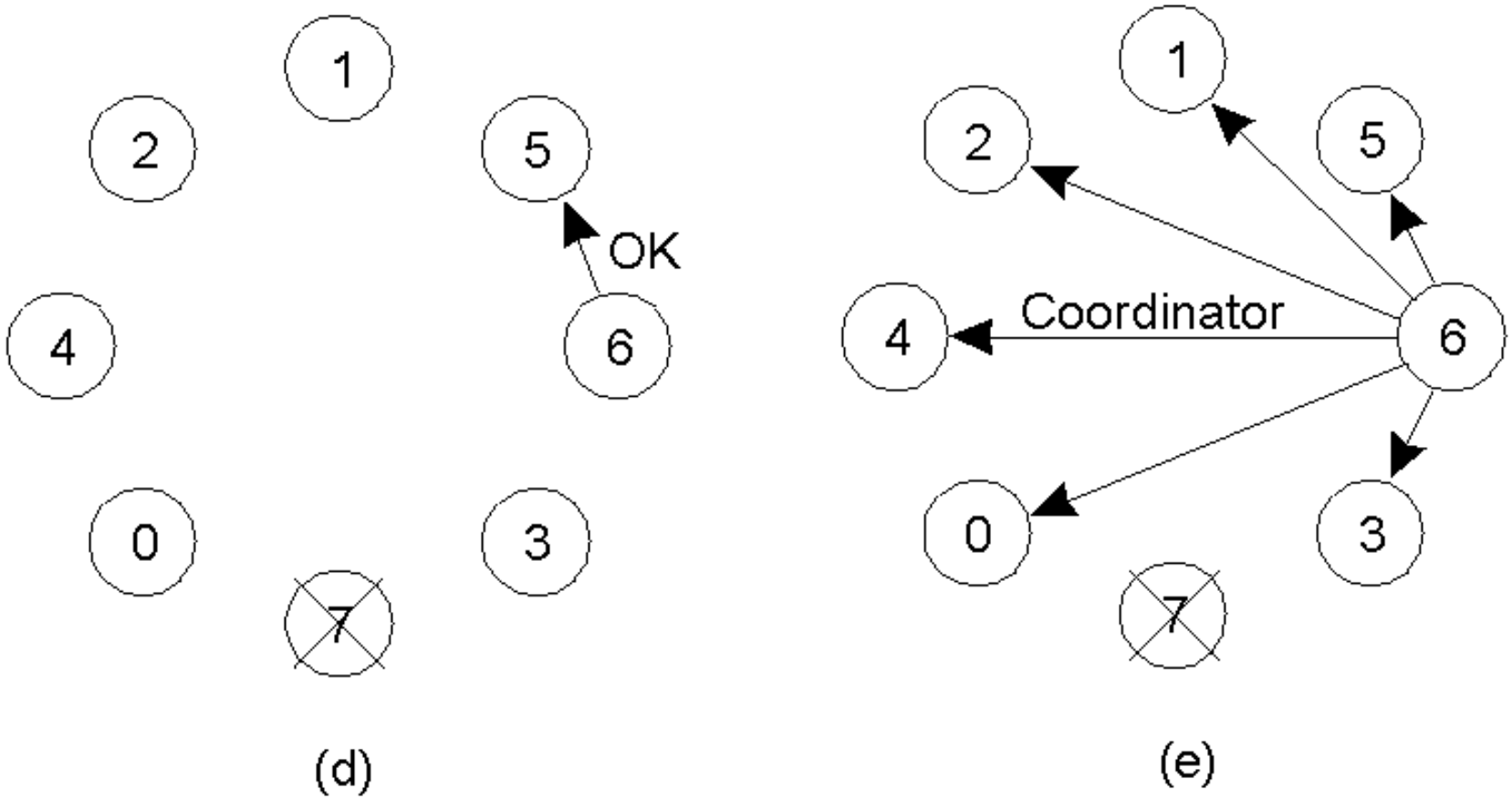
The Bully Algorithm (1)



The bully election algorithm

- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election

The Bully Algorithm (2)



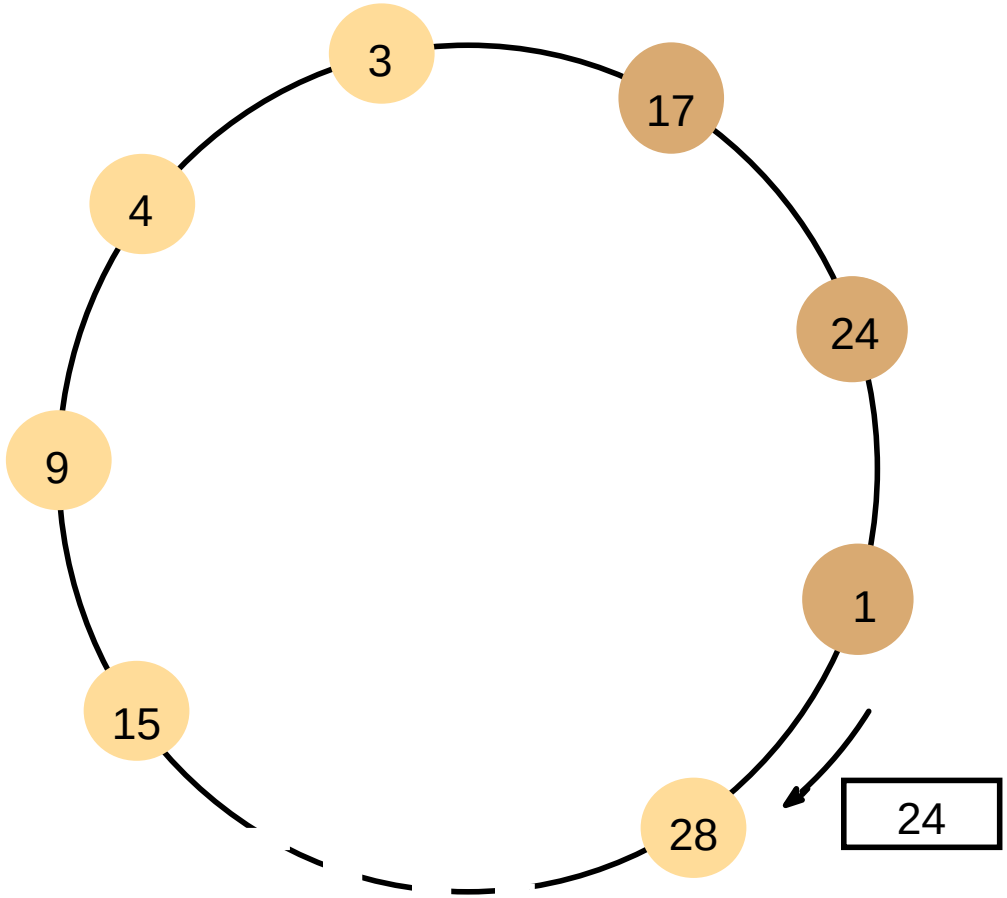
d) Process 6 tells 5 to stop

e) Process 6 wins and tells everyone

3 Hautapen-algoritmoak (4)

- Eraztunaren algoritmoa (Chang eta Roberts, 1979)
  - hautapena hasten duen prozesuak hurrengoari hautapen-mezua bidaltzen dio bere indizearekin, eta partaide bihurtzen da
  - prozesu batek hautapen-mezua jasotzerakoan, bere indizea handiagoa bada berea bidaliko du hurrengoari, partaide bihurtuz (lehendik partaide bazen, ez du ezer bidaliko). Aldiz bere indizea txikiagoa bada, jasotakoa bidaliko du
  - partaide batek bere indizea duen hautapen-mezua jasotzerakoan koordinatzaile bihurtzen da, hurrengoari hautatua-mezua bere indizearekin bidaliz
  - partaide batek hautatua-mezua jasotzerakoan, bere burua ez partaide bezala markatzen du, eta mezuan adierazten den prozesua koordinatzaile bezala ezartzen du. Koordinatzailea bera ez bada, hautatua-mezua hurrengoari bidaltzen dio

A ring-based election in progress



Note: The election was started by process 17.
The highest process identifier encountered so far is 24.
Participant processes are shown darkened.

3 Hautapen-algoritmoak (5)

- Eraztunaren algoritmoa (Tanenbaum)
  - hautapena hasten duen prozesuak hurrengoari hautapen-mezua bidaltzen dio bere indizearekin
  - prozesu batek hautapen-mezua jasotzerakoan, mezuak bere indizea dakarren ala ez aztertzen du
    - ez badakar, bere indizea eransten dio, hurrengoari bidaliz
    - aldiz mezuak bere indizea badakar, orduan koordinatzailea nor izango den kalkulatu dezake (adibidez, indizerik handiena duena). Beraz, hautapen-mezua hautatua-mezua bihurtzen du, hurrengoari bidaliz
    - hautatua-mezuak eraztunari buelta osoa ematen dionean, prozesu guztiek badakite nor den koordinatzaile berria
      - baita koordinatzaile berriak ere ;-)

4 Talde-komunikazioa

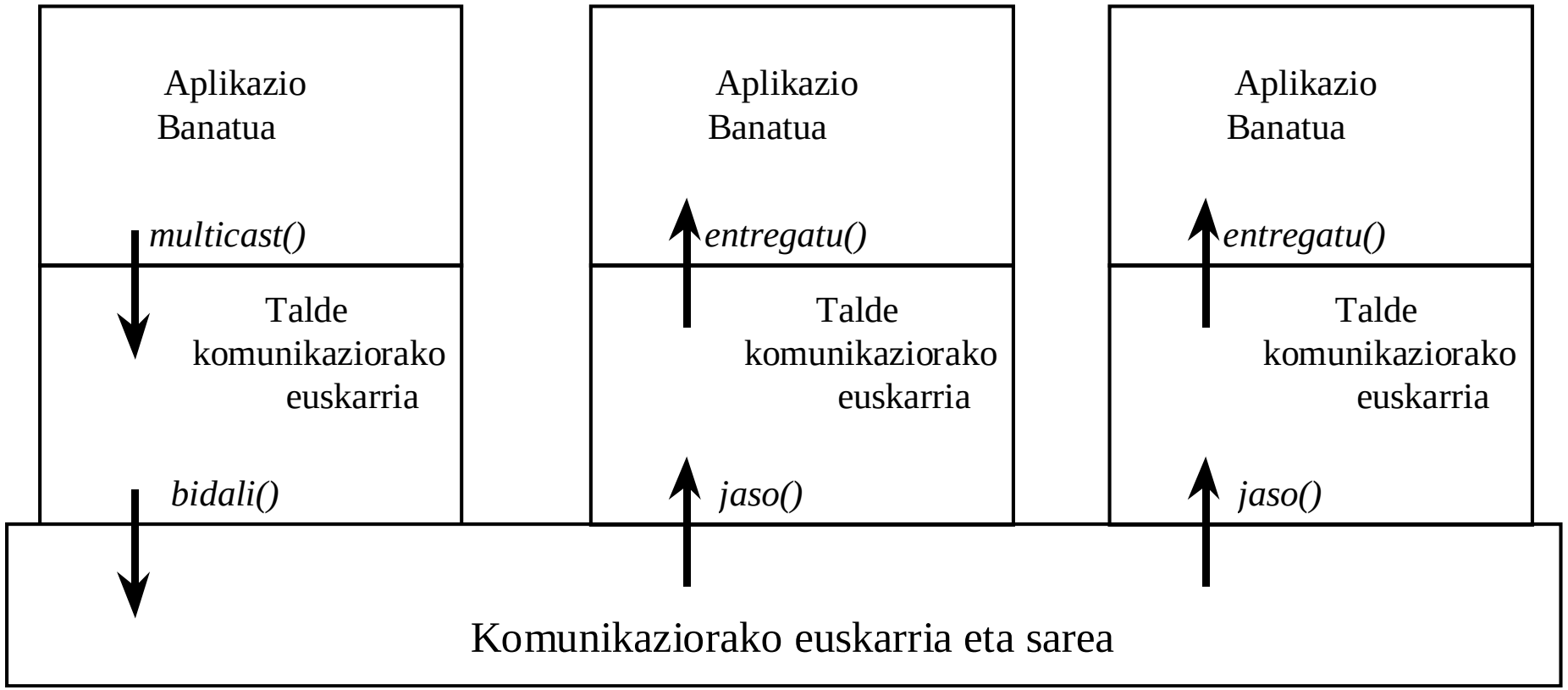
- Eredua:
  - prozesu taldeak
  - hedapena (1:N komunikazioa), semantika desberdinak
  - *broadcast* eta *multicast*
- Aplikazioak:
  - gertaeren notifikazioa
  - zerbitzu urrunen bilaketa
  - zerbitzuen erreplikazioa, hutsegite tolerantziagatik edota eskuragarritasunagatik/eraginkortasunagatik
- Adibideak: *Isis*, *Horus*, *Ensemble*, *JGroups*, *Transis*, *IP-multicast*, *Spread*, *Appia*…

4 Talde-komunikazioa (2)

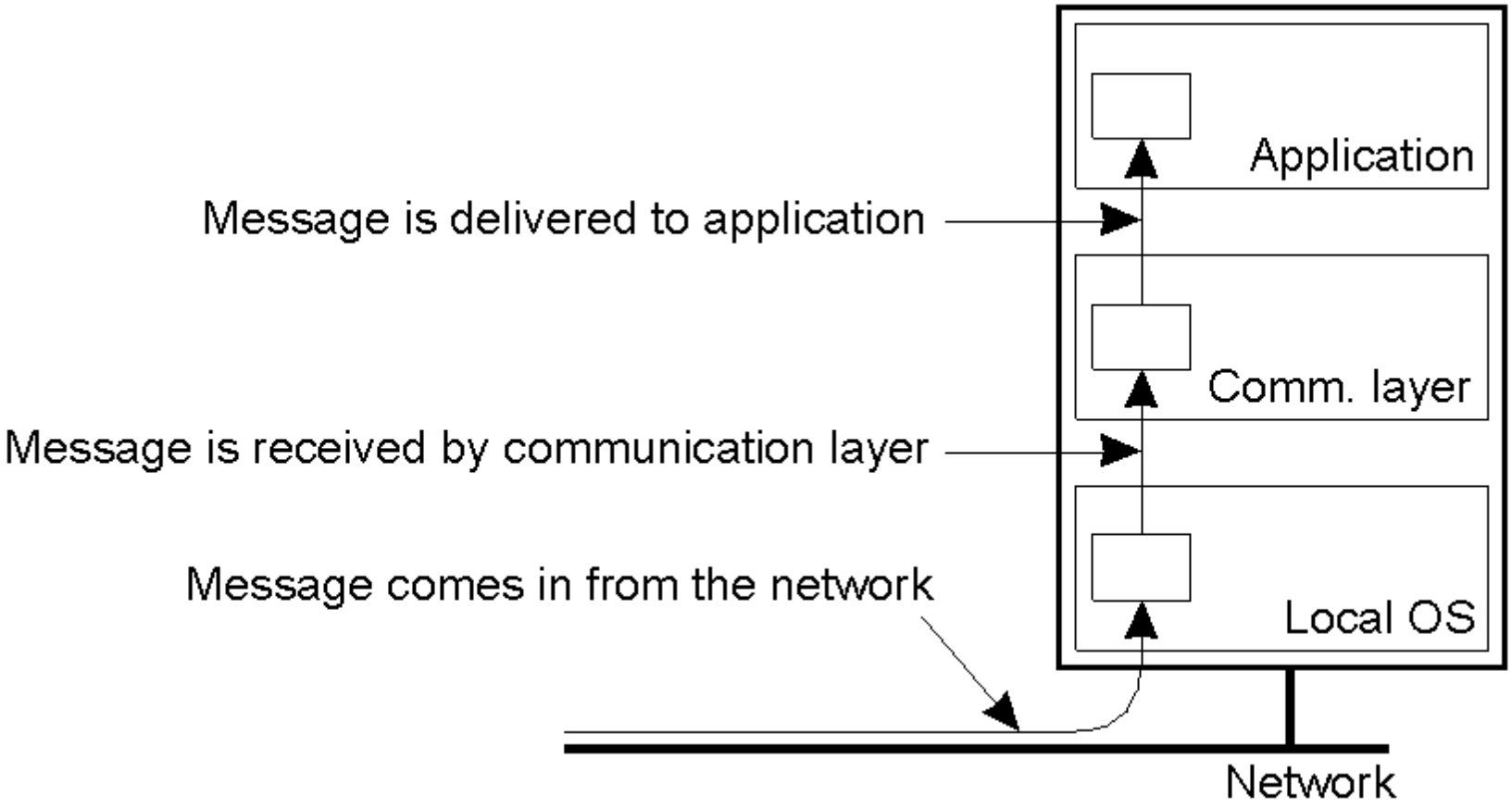
- Talde komunikaziorako euskarria (*middleware)*:
  - taldeen identifikazioa eta atzipena
  - taldeko kideei mezuak iritsaraztea, erabilitako hedapenaren semantika errespetatuz
  - taldeko konposizioa kudeatu, irteerak (hutsegiteengatik) eta sarrerak (errekuperazioengatik) sendoki kudeatuz
- Komunikazio eredua:
  - ***multicast****(G, m)*. *m* mezua *G* taldeari hedatzen zaio. Suposaketa: mezuaren igorlea taldeko kidea da
  - ***entregatu****(m)*. *m* mezua *entregatu* exekutatzen duen prozesuari ematen zaio

4 Talde-komunikazioa (3)

- Komunikazio eredua:

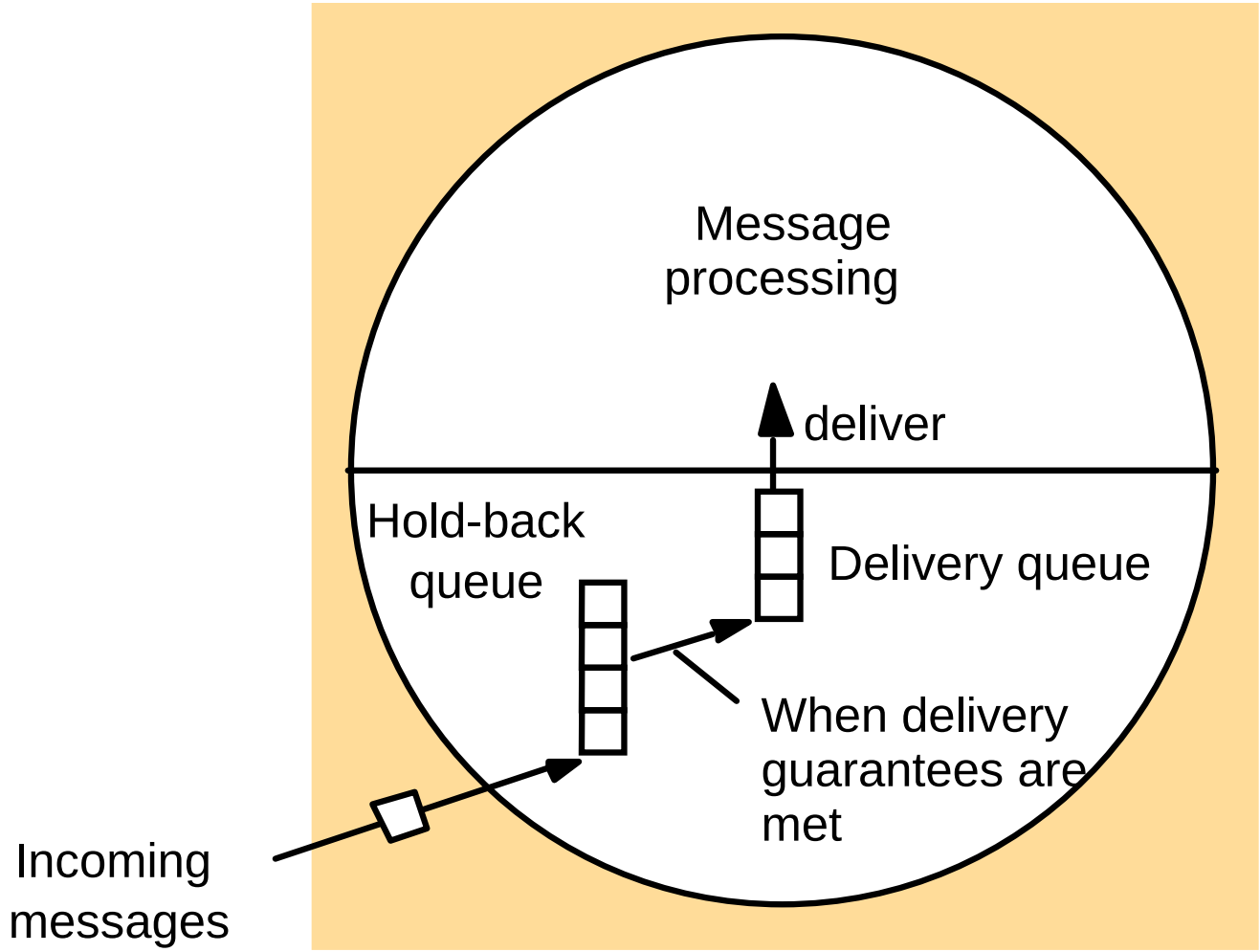


Message Receipt and Delivery



The logical organization of a distributed system to distinguish between message receipt and message delivery

The hold-back queue for arriving multicast messages



4 Talde-komunikazioa (4)

- Komunikaziorako euskarriak hardware hedapena eskaini dezake (adibidez, *Ethernet* sareetan)
- Prozesuek hutsegin dezakete
  - bi motako prozesuak: zuzenak eta okerrak
- Komunikazio kanalak ia-fidagarriak dira
  - prozesu zuzenen artean bidalitako mezu guztiak lehenago edo beranduago jaso egingo dira
  - kanal guztiz fidagarriak igorle okerrak bidalitako mezuak (jakina, hutsegin baino lehen) jasotzen direla bermatzen dute
    - mezu-trukean oinarritutako sistemetan ez da praktikoa

4 Talde-komunikazioa (5)

- Hedapen semantikak. Bi alderdi nagusi:
  - fidagarritasuna
  - entregatzeko ordena
- Oinarrizko inplementazioa (ez fidagarria):
  - prozesu batek *multicast(G, m)* exekutatu nahi duenean: $\forall P \in G$, *bidali(P, m)* exekutatzen du
  - *P*-k *jaso(m)* exekutatzen duenean: *P*-k *entregatu(m)* exekutatzen du
    - hedapen ez fidagarria: *m* mezua prozesu zuzen batzuk entregatu dezakete eta beste prozesu zuzen batzuk ez (igorleak *multicast* exekutatzen ari den bitartean huts egiten badu)

4 Talde-komunikazioa (6)

- Hedapen fidagarria (“denak ala inor ez”):
  - baliotasuna: prozesu zuzen batek *m* mezua hedatzen badu, orduan prozesu honek *m* entregatuko du
  - akordioa: prozesu zuzen batek *m* mezua entregatzen badu, orduan *m* prozesu zuzen guztiak entregatuko dute
    - hedapen ez fidagarriak bermatzen ez duen propietatea
  - zintzotasuna: edozein mezurentzako, prozesu zuzen guztiek gehienez behin entregatuko dute, eta bakarrik aurretik hedatua izan bada

– Inplementazioa (*R-multicast*, *R-entregatu*):
  - P_i-k *R-multicast(G, m)* exekutatzerakoan: *multicast(G, m)*
  - P_j-k *jaso(m)* lehendabiziko aldiz exekutatzen duenean:
    1. $P_j \neq P_i$ bada: *multicast(G, m)*
    2. P_j-k *R-entregatu(m)* exekutatzen du

Reliable multicast algorithm

On initialization
    Received := {};

For process p to R-multicast message m to group g
    B-multicast(*g*, *m*); // $p \in g$ is included as a destination

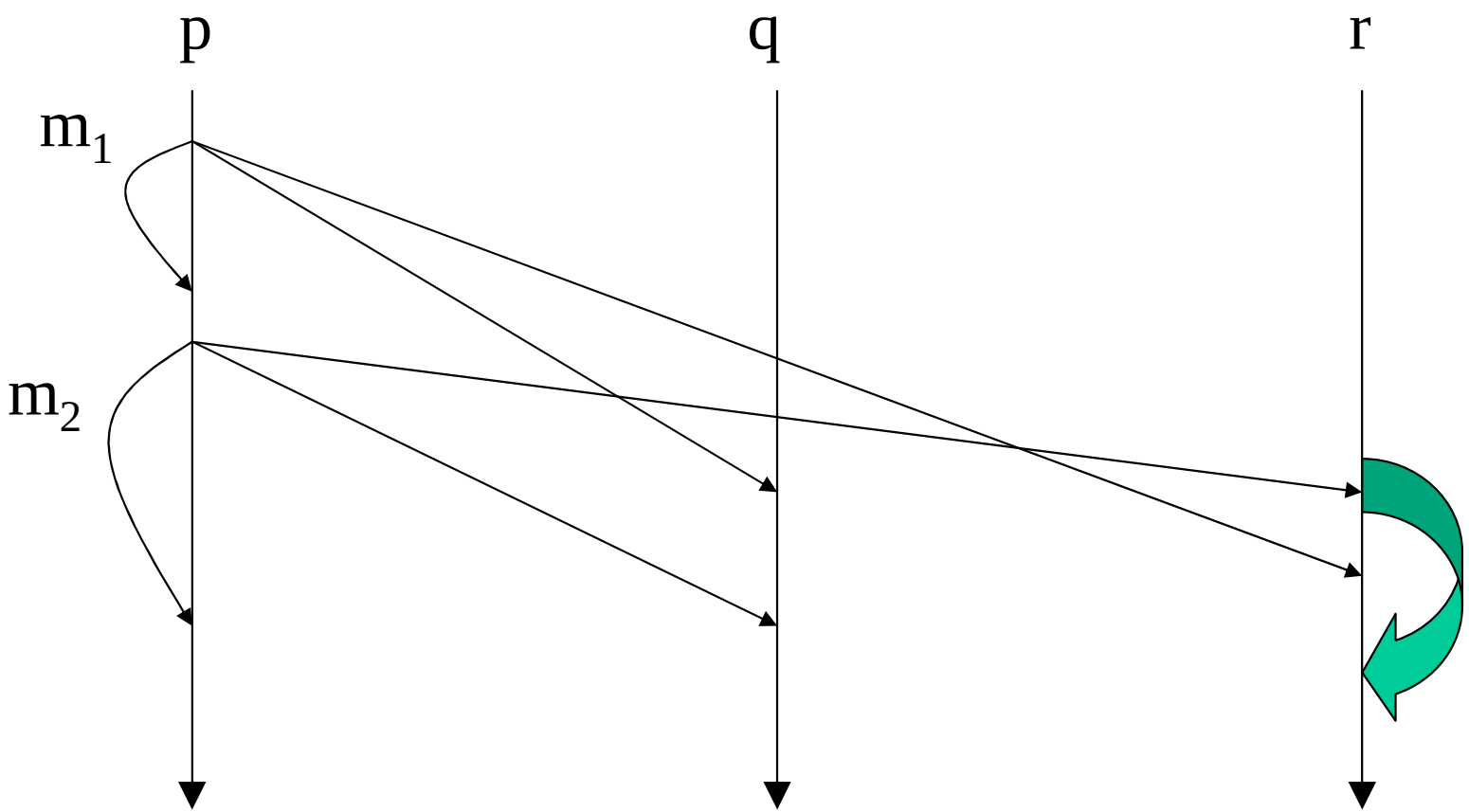
On B-deliver(*m*) *at process q with g = group*(*m*)
    if ($m \notin$ *Received*)
    then
        Received := *Received* $\cup$ {*m*};
        if ($q \neq p$) *then B-multicast*(*g*, *m*); *end if*
        R-deliver m;
    end if

4 Talde-komunikazioa (7)

- Hedapen (fidagarriak) ordenatuak:
  - *FIFO* ordena: prozesu batek m_1 mezua m_2 mezua baino lehenago hedatzen badu, orduan prozesu zuzenen artean inork ez du m_2 entregatuko lehenago m_1 entregatu ez badu
  - orden kausala: m_1 mezuaren hedapena m_2 mezuarena baino lehen (kausalki) gertatu bada, orduan prozesu zuzenen artean inork ez du m_2 entregatuko lehenago m_1 entregatu ez badu
    - orden kausala *FIFO* ordena baino sendoagoa da
  - orden osoa: zuzenak diren bi prozesuk P_i eta P_j bi mezu m_1 eta m_2 entregatzen badute, orduan P_i-k m_1 m_2 baino lehenago entregatzen ditu baldin eta P_j-k ere m_1 m_2 baino lehenago entregatzen baditu
    - orden osoa duen hedapenari hedapen atomikoa ere deitzen zaio. Sistema errepikatuetan sendotasuna bermatzeko erabiltzen da

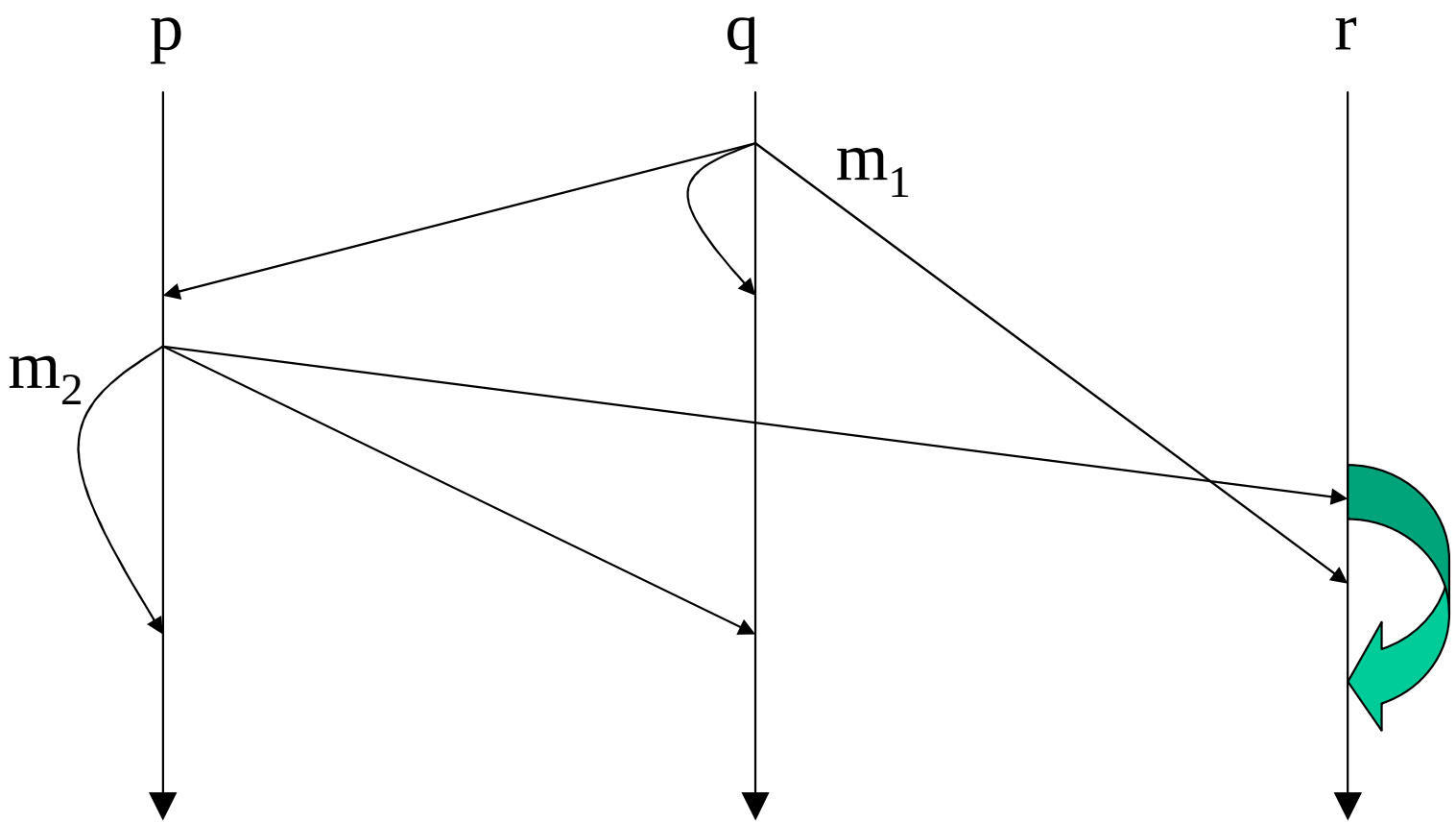
4 Talde-komunikazioa (8)

- *FIFO* hedapena:



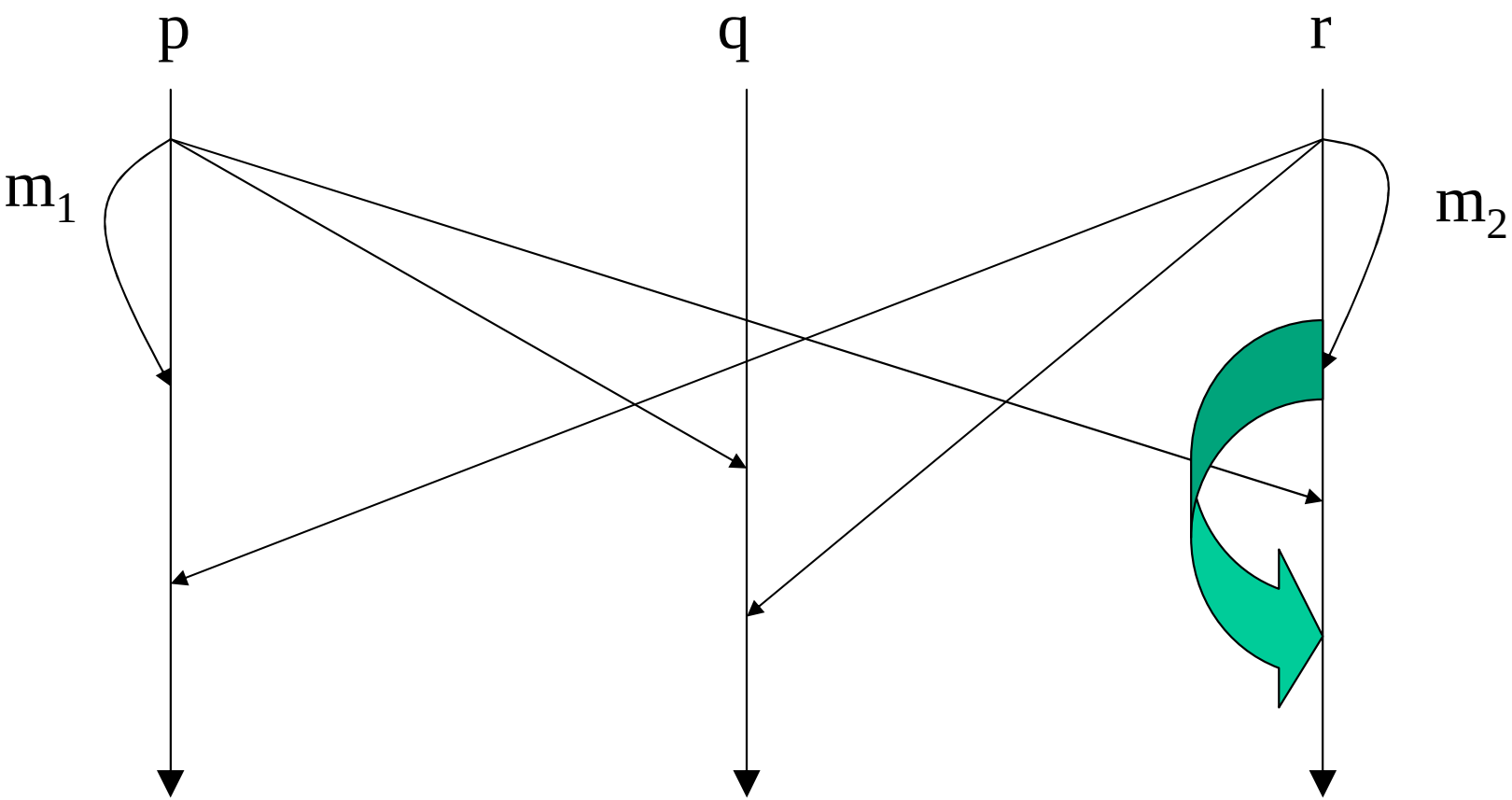
4 Talde-komunikazioa (9)

- Hedapen kausala (kausala $\rightarrow$ *FIFO*):



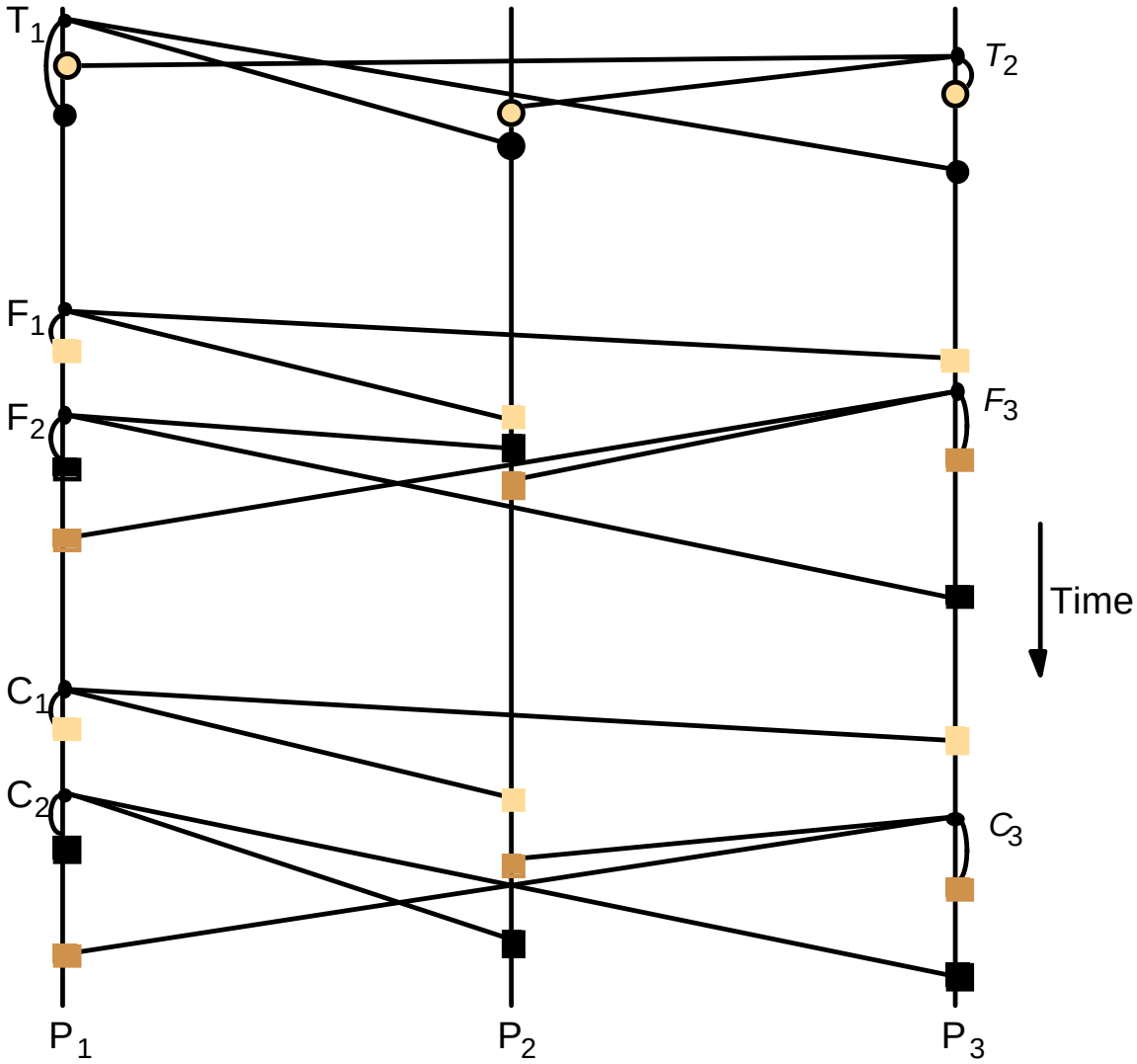
4 Talde-komunikazioa (10)

- Hedapen atomikoa:



Total, FIFO and causal ordering of multicast messages

Notice the consistent ordering of totally ordered messages T_1 and T_2,
the FIFO-related messages F_1 and F_2 and the causally related messages C_1 and C_3
– and the otherwise arbitrary delivery ordering of messages.



4 Talde-komunikazioa (11)

- Hedapen motak:
  - hedapen ez fidagarria (adibidez, *IP-multicast*)
  - hedapen fidagarria
    - ordenarik gabekoa
    - FIFO hedapena
    - hedapen kausala (bide batez, FIFO dena)
    - hedapen atomikoa
    - hedapen atomikoa eta FIFO
    - hedapen atomikoa eta kausala (bide batez, FIFO dena)
- Adibidea: *USENET News*
    - fidagarria (*TCP*), *FIFO* (ez kausala, ezta atomikoa)
    - kausala praktikan: erantzunetan mezu originala bidaliz

Display from bulletin board program

Bulletin board: *os.interesting*

Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

Alice

Bob

Bulletin board: *os.interesting*

Item	From	Subject
23	A.Hanlon	Mach
24	T.L'Heureux	RPC performance
25	M.Walker	Re: Mach
26	A.Hanlon	Re: Microkernels
27	G.Joseph	Microkernels
end		

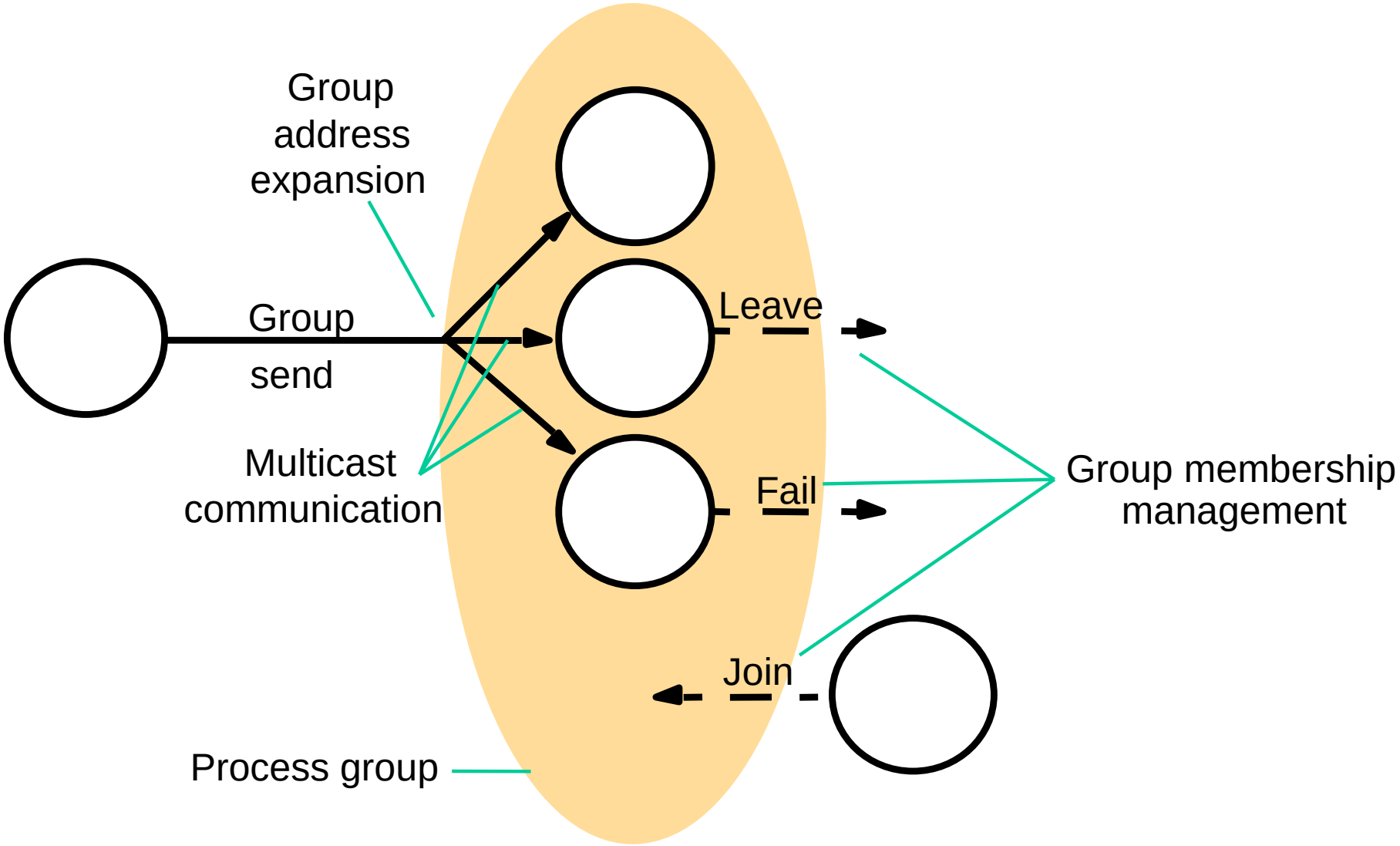
4 Talde-komunikazioa (12)

- Hedapen ordenatuen inplementazioa:
  - *FIFO* hedapena: igorleak mezuak zenbatuz
  - hedapen kausala: denborazko bektoreak erabiliz
  - hedapen atomikoa sinkronoa:
    - P_i-k *A-multicast(G, m)* exekutatzerakoan: *R-multicast(G, m)*, *m*-ri *t(m)* denbora marka lokala erantsiz
    - P_j-k *R-entregatu(m)* exekutatzen duenean: *A-entregatu(m)* $t(m)+\Delta$ denbora lokalerako planifikatzen du
      - Δ: sisteman mezuak transmititzeko behar den denbora maximoa + erlojuen arteko desbideratze maximoa da (erlojuak sinkronizatuak suposatzen dira)
      - *A-entregatu* planifikatutako denboran ezin bada exekutatu (mezua beranduegi *R-entregatu* delako), igorleak hutsegin duela suposatuko da

4 Talde-komunikazioa (13)

- Partaidetzaren kudeaketa (talde dinamikoetan):
  - irteeren (hutsegiteengatik) eta sarreren (errekuperazioengatik) kudeaketa
    - ***join***(P_i, G)
    - ***leave***(P_i, G)
  - hutsegiteak detektatzeko mekanismoa beharrezkoa da
  - bista kontzeptua (*view*): taldearen partaidetza
    - *join()* edo *leave()* exekutatzerakoan, bista aldaketa egiten da
    - taldea bista sekuentzia baten bidez modelatzen da: $v_0, v_1, ..., v_k$
    - bista berria instalatzeko: ***install***(v_{k+1})
    - bisten sendotasuna bermatzeko protokoloa behar da
    - bistetan baliatuz, koordinatzailearen hautaketa zuzena da

Services provided for process groups

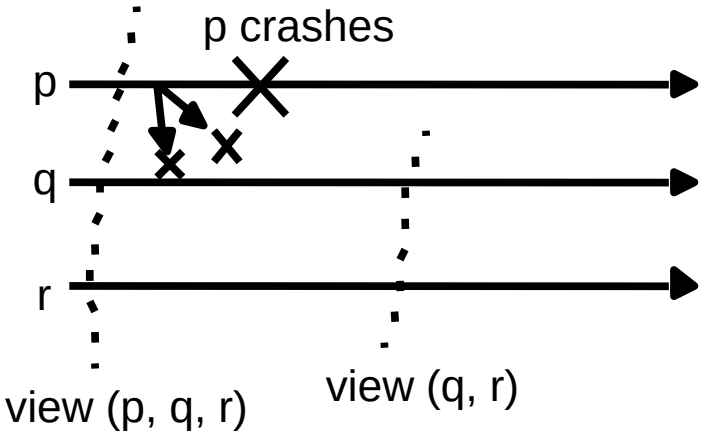


4 Talde-komunikazioa (14)

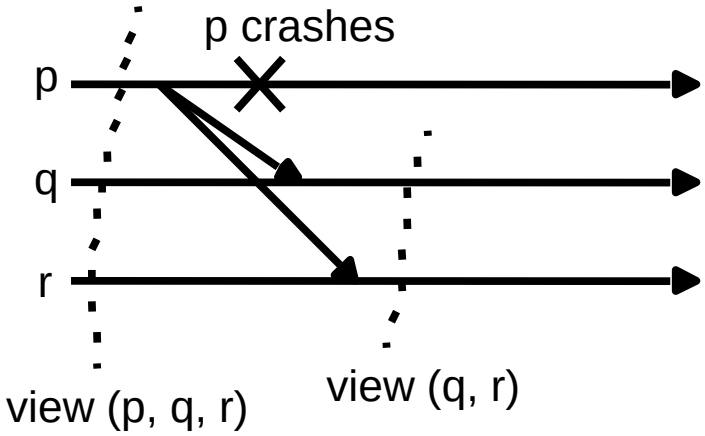
- Bisten kudeaketaren propietateak:
  - amaiera: prozesu baten hutsegiteak edota *join()* edo *leave()* funtzioen exekuzioak lehenago edo beranduago bista berri baten instalazioa ekartzen dute
  - akordioa: prozesu zuzen guztiek mezu berdinak entregatzen dituzte bista baten barruan
  - baliotasuna: bedi P_i prozesu zuzen bat v_k bistan m mezua entregatu duena. Baldin badago prozesuren bat $P_j \in v_k$ m mezua v_k bistan entregatzen ez duena, orduan P_i-k instalatuko duen v_{k+1} bistan P_j ez da egongo

View-synchronous group comm.

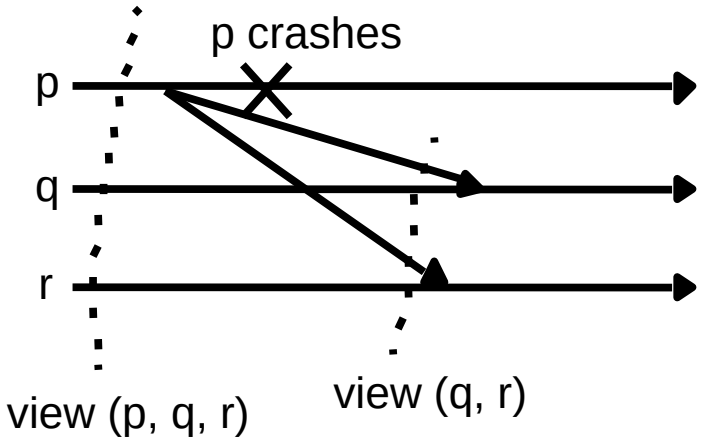
a (allowed).



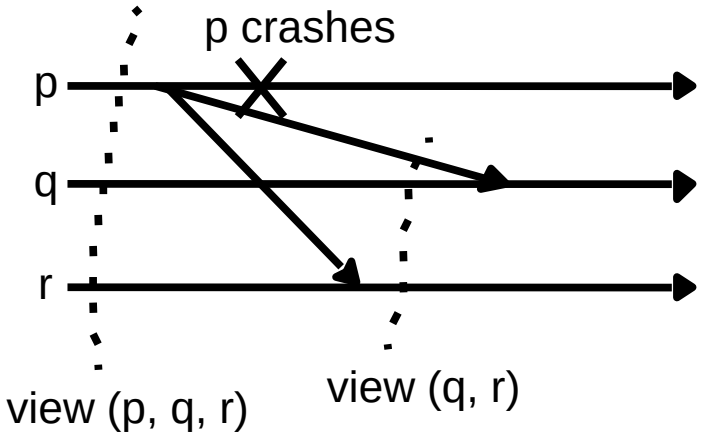
b (allowed).



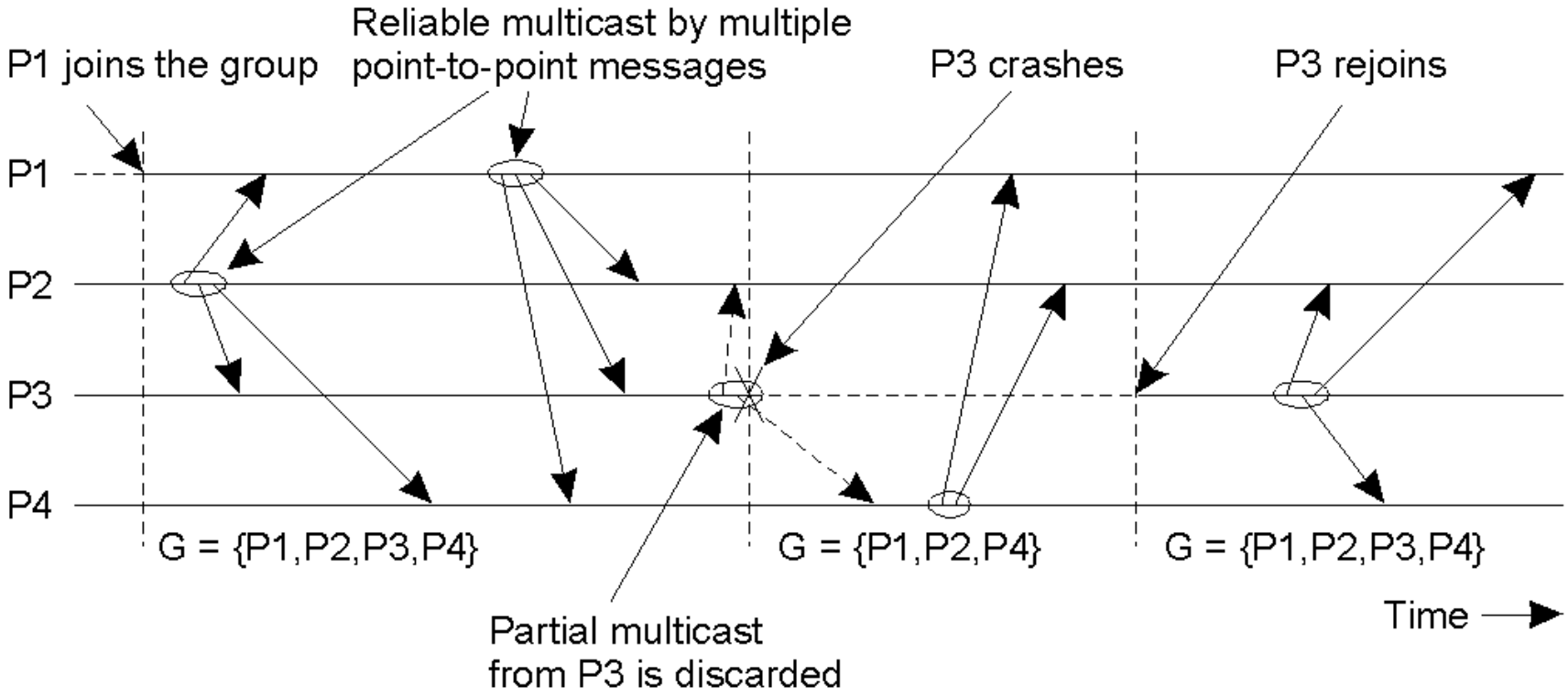
c (disallowed).



d (disallowed).

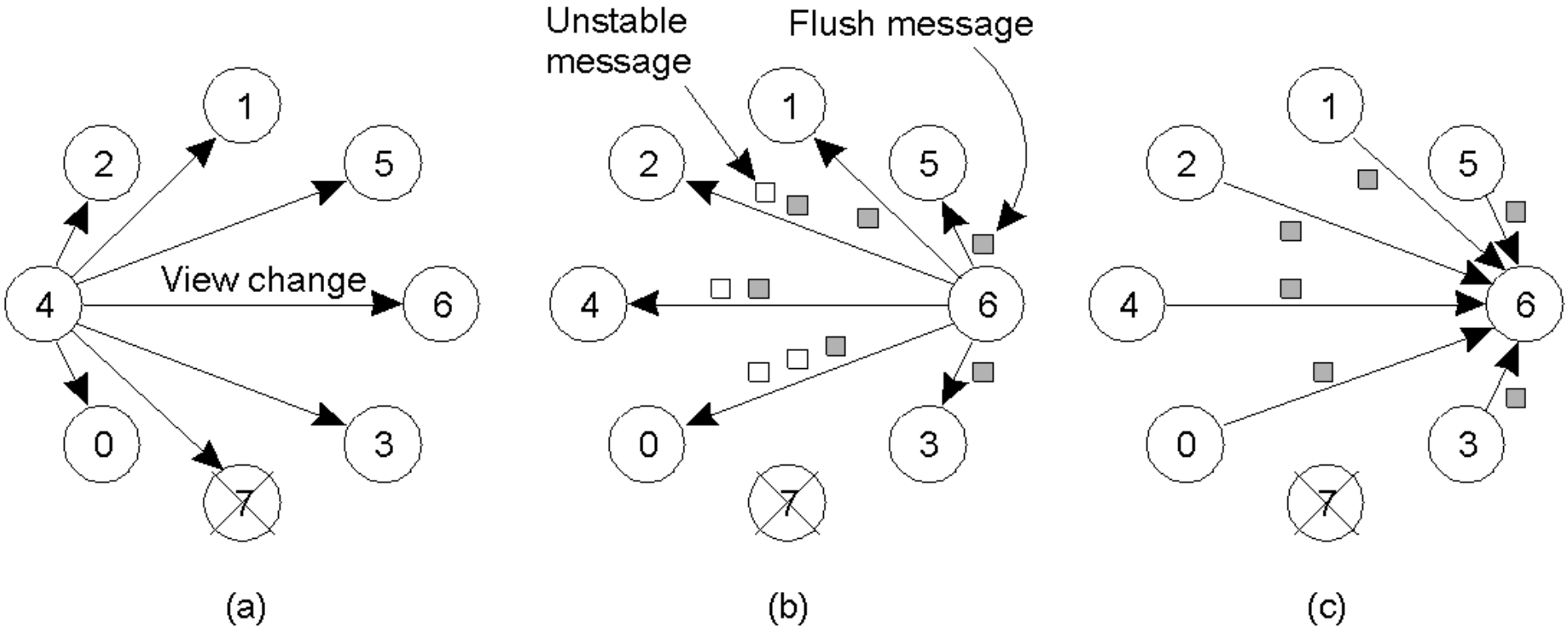


Virtual Synchrony



The principle of virtual synchronous multicast

Implementing Virtual Synchrony



a) Process 4 notices that process 7 has crashed, sends a view change

b) Process 6 sends out all its unstable messages, followed by a flush message

c) Process 6 installs the new view when it has received a flush message from everyone else

4 Talde-komunikazioa (15)

- *ISIS* sistema:
  - talde-komunikaziorako *middleware*-a
  - *UNIX* sisteman garatua (Cornell University, 1983)
  - prozesu taldeak + bisten kudeaketa + hedapenak
  - *FBCAST*: *FIFO* hedapena
  - *CBCAST*: hedapen kausala
    - denborazko bektoreetan oinarritua
  - *ABCAST*: hedapen atomikoa eta kausala
    - prozesu koordinatzailean oinarritua
  - *GBCAST*: bisten kudeaketarako
    - bista-aldaketak gauzatzeko (*flush* protokoloa)

Causal ordering using vector timestamps (ISIS)

Algorithm for group member p_i ($i = 1, 2\ldots, N$)

On initialization

$V_i^g[j] := 0$ ($j = 1, 2\ldots, N$);

To CO-multicast message m to group g

$V_i^g[i] := V_i^g[i] + 1$;

B-multicast(g, $<V_i^g, m>$);

On B-deliver($<V_j^g, m>$) *from* p_j, *with* $g = group(m)$

place $<V_j^g, m>$ in hold-back queue;

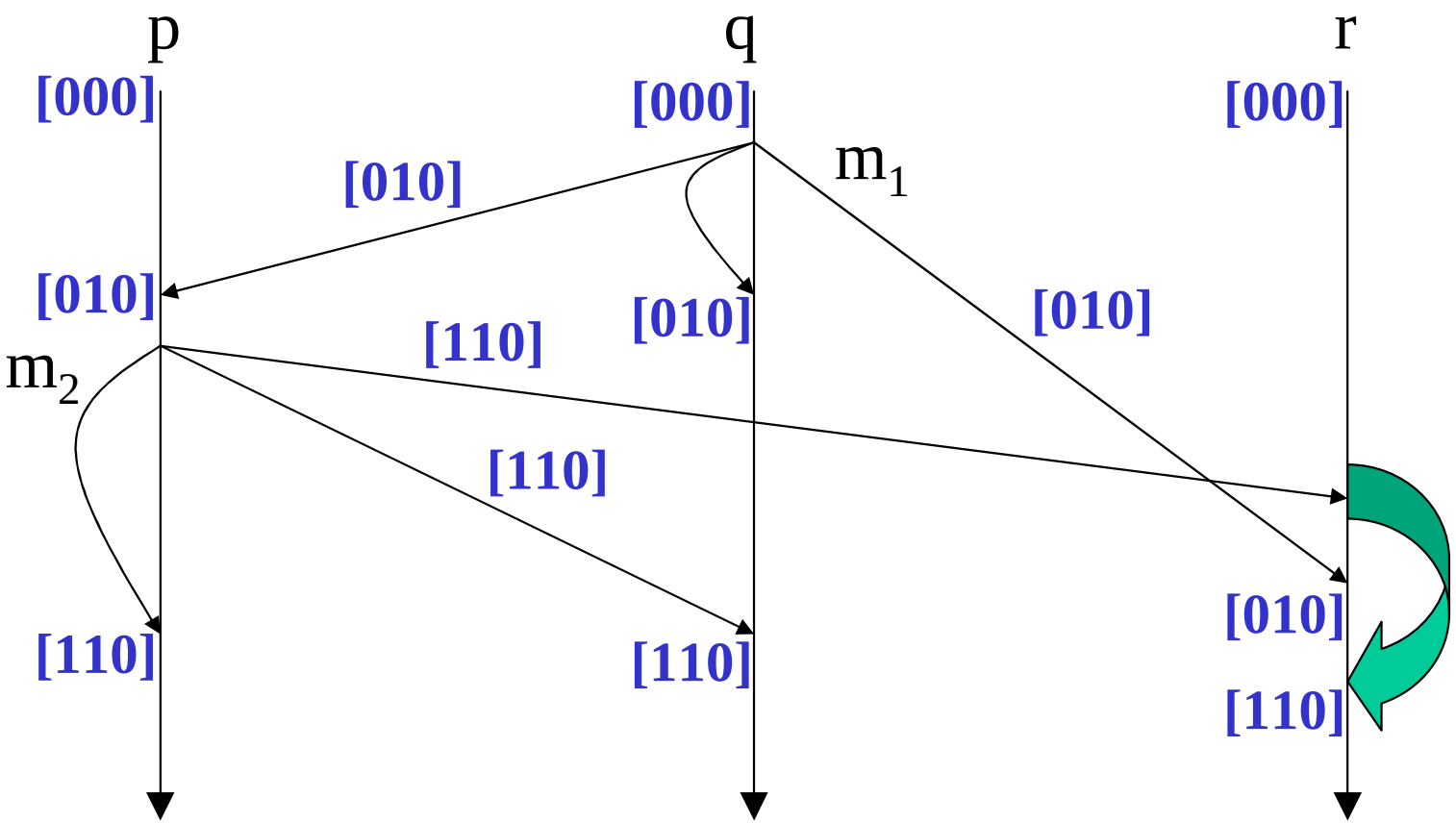
wait until $V_j^g[j] = V_i^g[j] + 1$ and $V_j^g[k] \leq V_i^g[k]$ ($k \neq j$);

CO-deliver m; // after removing it from the hold-back queue

$V_i^g[j] := V_i^g[j] + 1$;

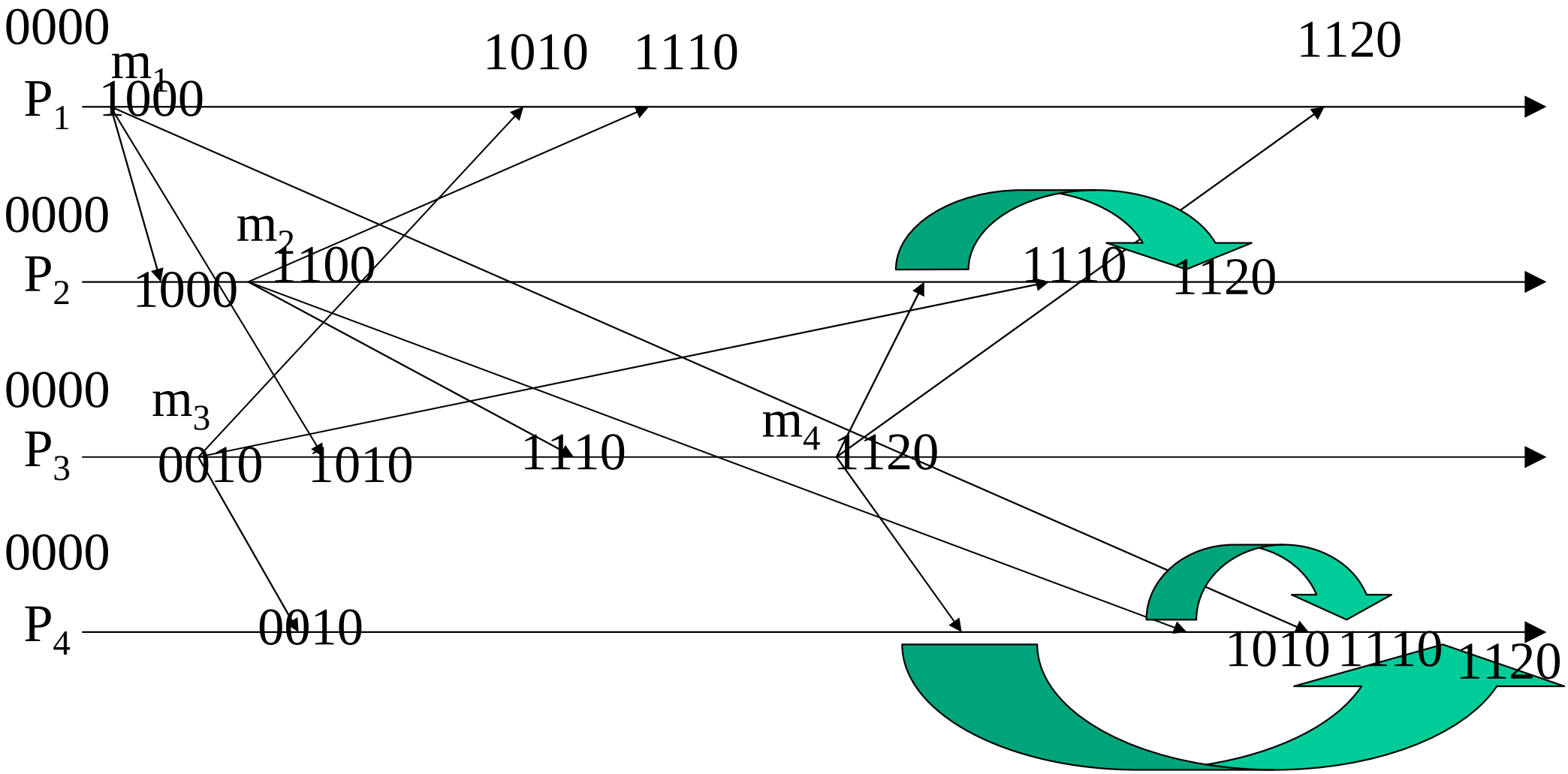
4 Talde-komunikazioa (16)

- Hedapen kausalaren adibidea:



4 Talde-komunikazioa (17)

- Hedapen kausalaren ariketa:



Total ordering using a sequencer

1. Algorithm for group member p

On initialization: $r_g := 0$;

To TO-multicast message m to group g
    B-multicast($g \cup \{sequencer(g)\}$, <*m, i*>);

On B-deliver(<*m, i*>) *with* $g = group(m)$
    Place <*m, i*> in hold-back queue;

On B-deliver(m_{order} = <"order", *i, S*>) *with* $g = group(m_{order})$
    wait until <*m, i*> in hold-back queue and $S = r_g$;
    TO-deliver m; // (after deleting it from the hold-back queue)
    $r_g = S + 1$;

2. Algorithm for sequencer of g

On initialization: $s_g := 0$;

On B-deliver(<*m, i*>) *with* $g = group(m)$
    B-multicast(*g*, <"order", *i*, s_g>);
    $s_g := s_g + 1$;

5 Erreplikazioa

(1) Erreplikazio sendoa

- helburu nagusia: hutsegite-tolerantzia
- nodo bakoitzak kopia osoa kudeatzen du (kudeaketa banatua)
- nodoak gertu egoten dira (sare lokalean normalean)
- sendotasuna bermatzea oinarrizkoa da
- adibidea: banketxeetako sistema informatikoak

(2) Zerbitzuen banaketa

- helburu nagusia: latentzia (itxaron-denbora) hobetzea
- erreplikazio partziala edota osoa (kudeaketa zentralizatua)
- nodoak geografikoki urrunak
- sendotasuna bermatzea ez da hain garrantzitsua
- adibideak: datu-base banatuak, *mirroring* Internet-en

5 Erreplikazioa (2)

(3) *Caching*

- informazioa lokalki (edota gertu) gordetzea
- helburu nagusia: eraginkortasuna hobetzea
- erreplikazio partziala (kudeaketa bezeroaren ardura da)
- behin-behinekotasuna
- sendotasuna bermatzea ez da hain garrantzitsua
- adibideak: NFS

– Erreplikazio motak konbinatu daitezke (adib: *web*-a):

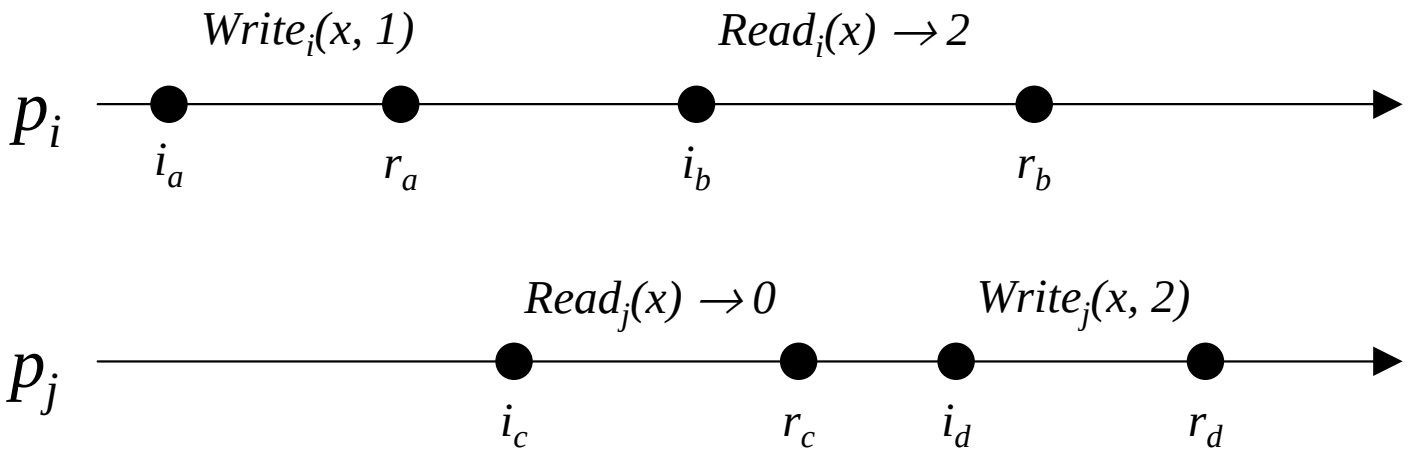
- zerbitzari errepikatuak sare lokalean
- geografikoki urrutiko kopiak (*mirroring*)
- bezeroek duela gutxi eskatutako orrien kopia lokalak mantentzen dituzte (*caching*)
- *proxy*-ak ere egon daitezke (*caching* hierarkikoa)

5 Erreplikazio sendoa

- Eredua:
  - bezeroek eskariak egiten dizkiote zerbitzariari
    - printzipioz, zerbitzua bakarra ala errepikatua izan daiteke
  - operazio bakoitza o_k bi gertaerez osaturik dago: (i_k , r_k)
    - eskaria (*invocation*): i_k , erantzuna (*reply*): r_k
  - bezeroak sinkronoak dira: bezero bakoitzaren operazio sekuentzian [o_1 , o_2 , ..., o_k , ...], i_{k+1} ez da gertatzen r_k gertatu arte
  - adibidea: *x* aldagaia
    - operazioak: *Read(x)* $\rightarrow$ *v*, *Write(x, v)*
    - ohizko semantika: *Read(x)* aldagaian idatzitako azken balioa bueltatzen du, *Write(x, v)* aldagaian *v* balioa idazten du

5 Erreplikazio sendoa (2)

- Adibidea (hasieran, $x = 0$):



  - operazio sekuentziak: p_i: [o_a , o_b], p_j: [o_c , o_d]
  - sekuentzia globala: $\tau = [i_a , r_a , i_c , i_b , r_c , i_d , r_b , r_d]$
  - τ-ko gertaerak permutatuz exekuzioak lortzen dira

    legala: $\sigma_1 = [Read_j(x) \rightarrow 0, Write_i(x, 1), Write_j(x, 2), Read_i(x) \rightarrow 2]$

    ilegala: $\sigma_2 = [Write_i(x, 1), Read_j(x) \rightarrow 0, ...]$

5 Erreplikazio sendoa (3)

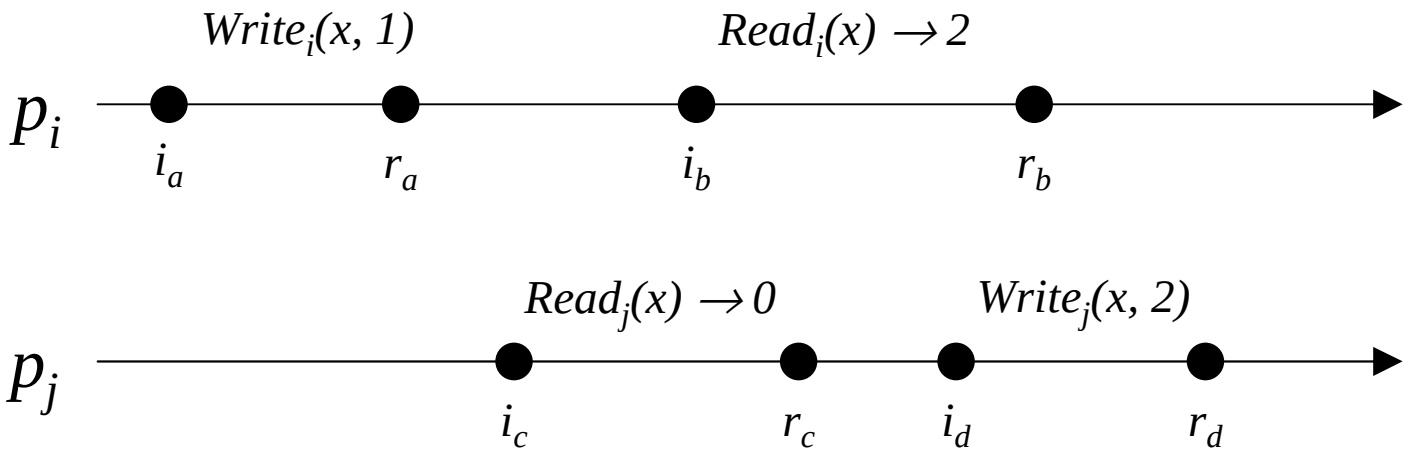
- Sendotasun motak (τ sekuentziatik exekuzio legalak lortzeko jarritako baldintzen arabera):
  - sendotasun sekuentziala: σ exekuzio legalak τ-rekiko sendotasun sekuentziala du baldin eta prozesu bakoitzaren operazioen ordena σ-n eta τ-n berdina bada
    - adib.: [$Read_j(x) \rightarrow 0$, $Write_i(x, 1)$, $Write_j(x, 2)$, $Read_i(x) \rightarrow 2$]
    - denborazko ordena ez da errespetatzen: r_a denboran i_c baino lehen gertatu arren, o_c o_a baino lehen exekutatzen da σ-n
      - egoera hau zerbitzu errepikatuetan gerta daiteke, eguneratu gabeko kopia bat irakurtzen bada
    - aplikazio batzutan sendotasun sekuentziala ez da nahikoa
      - adibidea: banku kontuen kudeaketan, diru-sartze eta zordunketen ordena ezin da aldatu (azken egoera desberdina izan daiteke)

5 Erreplikazio sendoa (4)

- Sendotasun motak:
  - lerrokadura: σ exekuzio legala lerrokagarria da sendotasun sekuentziala izateaz gain, edozein o_k o_h bi operazioentzako, r_k τ-n i_h baino lehen gertatu bada, orduan o_k σ-n o_h baino lehen exekutatzen da
    - lerrokadura izateko operazioen denborazko ordena errespetatu behar da. Adibidean, $Write_i(x, 1)$ $Read_j(x)$ baino lehen exekutatu behar da, r_a τ-n i_c baino lehen gertatu baita
    - [$Read_j(x) \rightarrow 0$, $Write_i(x, 1)$, $Write_j(x, 2)$, $Read_i(x) \rightarrow 2$]: EZ
    - [$Write_i(x, 1)$, $Read_j(x) \rightarrow 1$, $Write_j(x, 2)$, $Read_i(x) \rightarrow 2$]: BAI
      - legala izateko, $Read_j(x) \rightarrow 0$ $Read_j(x) \rightarrow 1$ bihurtu dugu
    - [$Write_i(x, 1)$, $Read_j(x) \rightarrow 1$, $Read_i(x) \rightarrow 1$, $Write_j(x, 2)$]: BAI
      - legala izateko, $Read_i(x) \rightarrow 2$ $Read_i(x) \rightarrow 1$ bihurtu dugu

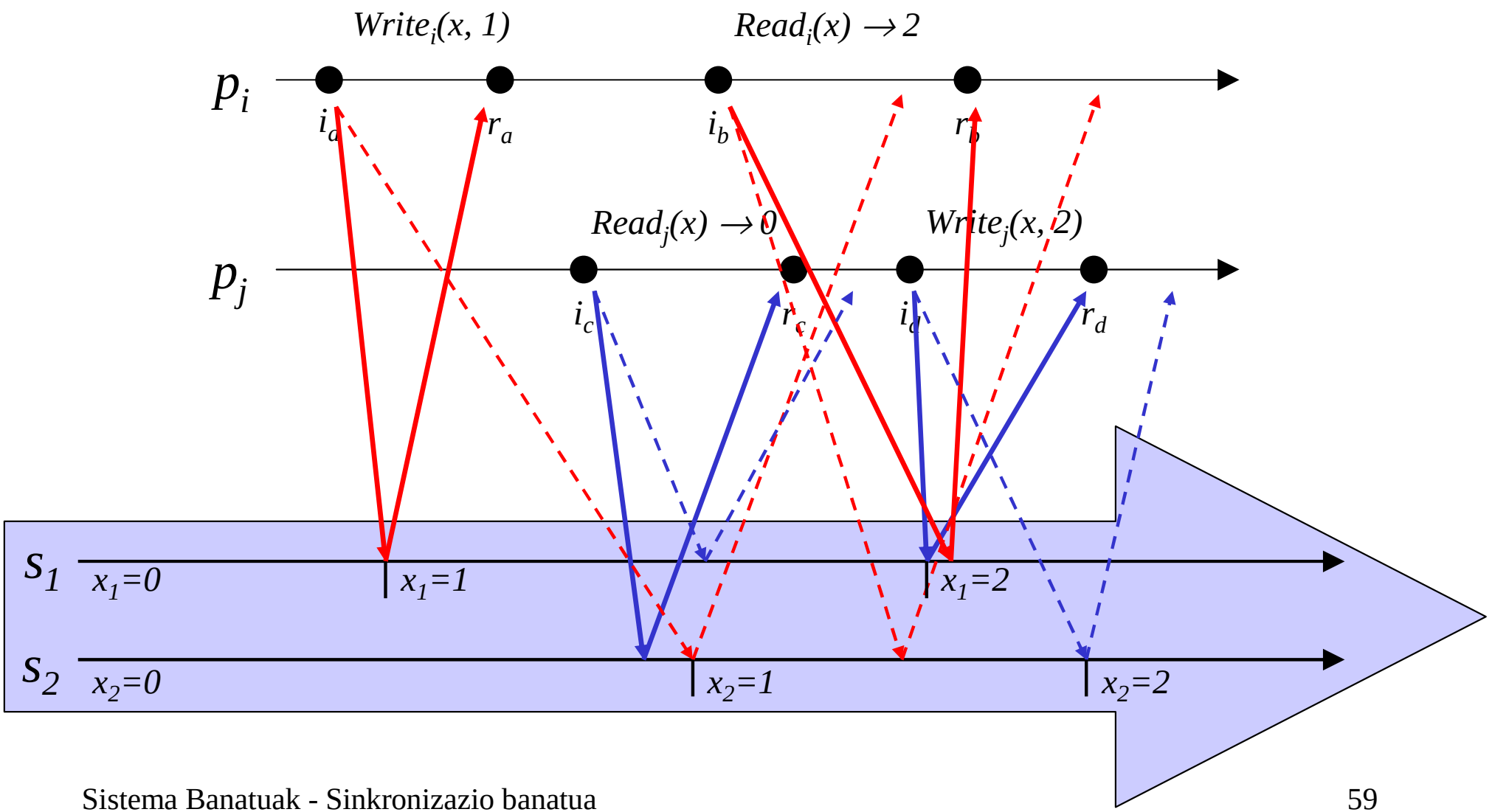
5 Erreplikazio sendoa (5)

- Laburpena (hasieran, $x = 0$):



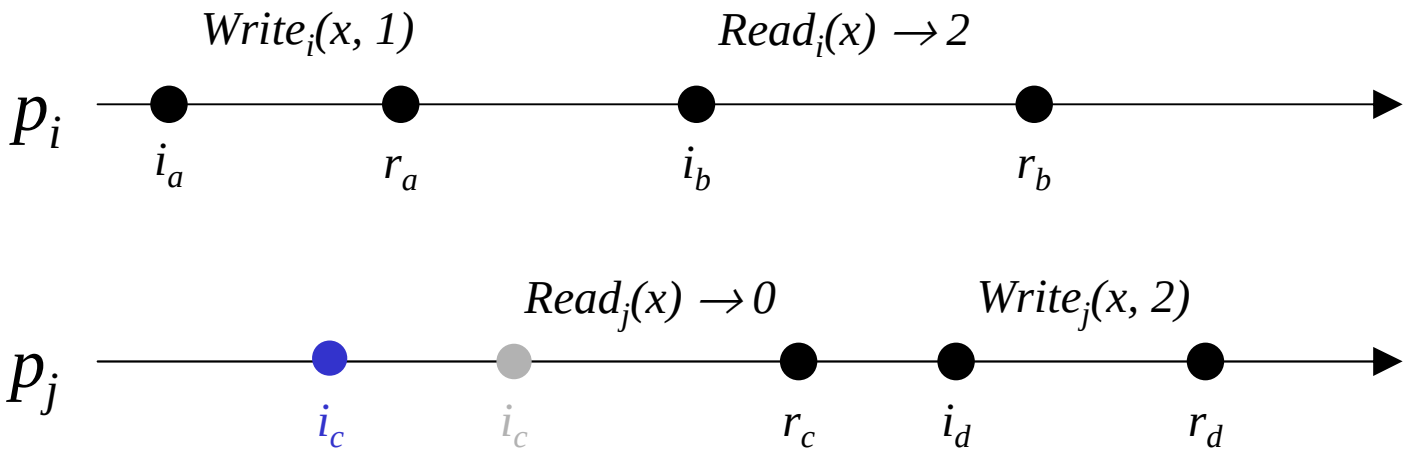
  - gertaeren sekuentzia globala: $\tau = [i_a, r_a, i_c, i_b, r_c, i_d, r_b, r_d]$
  - exekuzio ez legala: $\sigma_1 = [Write_i(x, 1), Read_j(x) \rightarrow 0, ...]$
  - exekuzio legala: $\sigma_2 = [Read_j(x) \rightarrow 0, Write_i(x, 1), Write_j(x, 2), Read_i(x) \rightarrow 2]$
    - sendotasun sekuentziala du
    - ez da lerrokadura
  - lerrokadurak diren exekuzioak:
    - $\sigma_3 = [Write_i(x, 1), Read_j(x) \rightarrow 1, Write_j(x, 2), Read_i(x) \rightarrow 2]$
    - $\sigma_4 = [Write_i(x, 1), Read_j(x) \rightarrow 1, Read_i(x) \rightarrow 1, Write_j(x, 2)]$

5 Erreplikazio sendoa (6)



5 Erreplikazio sendoa (7)

- Beste adibide bat (hasieran, $x = 0$):



  - gertaeren sekuentzia globala: $\tau' = [i_a, i_c, r_a, i_b, r_c, i_d, r_b, r_d]$
  - σ_2 exekuzioa: [$Read_j(x) \rightarrow 0$, $Write_i(x, 1)$, $Write_j(x, 2)$, $Read_i(x) \rightarrow 2$]
    - legala
    - sendotasun sekuentziala du
    - **lerrokadura!**

5 Erreplikazio sendoa (8)

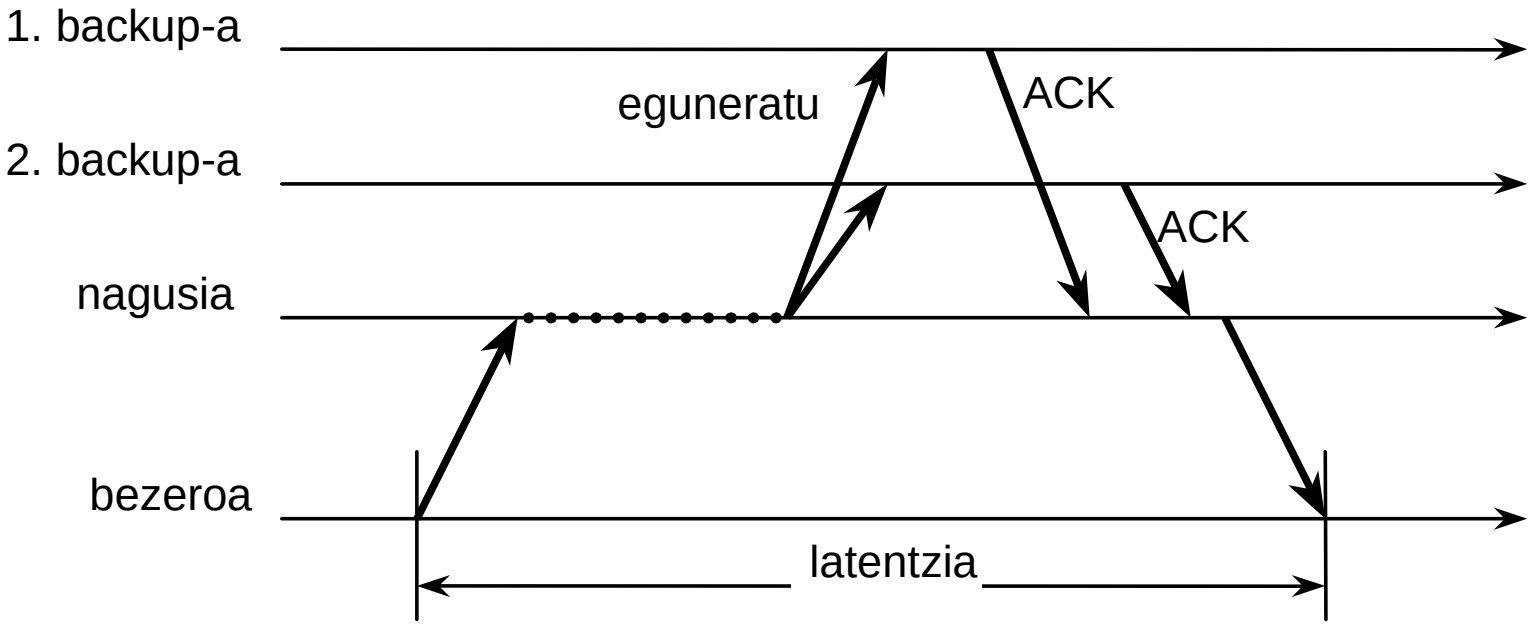
- Sistema errepikatuetan lerrokadurak bermatzeko nahikoa da bi kondizio hauek betetzea:

  (1) ordena: zerbitzu errepikatu baten kopia guztiek eskari guztiak orden berdinean exekutatzen dituzte

  (2) atomizitatea (“denak ala inor ez”): kopia batek eskari bat exekutatzen badu, orduan zuzenak diren (huts egiten ez duten) kopia guztiak ere lehentxeago ala beranduago eskari hura exekutatuko dute

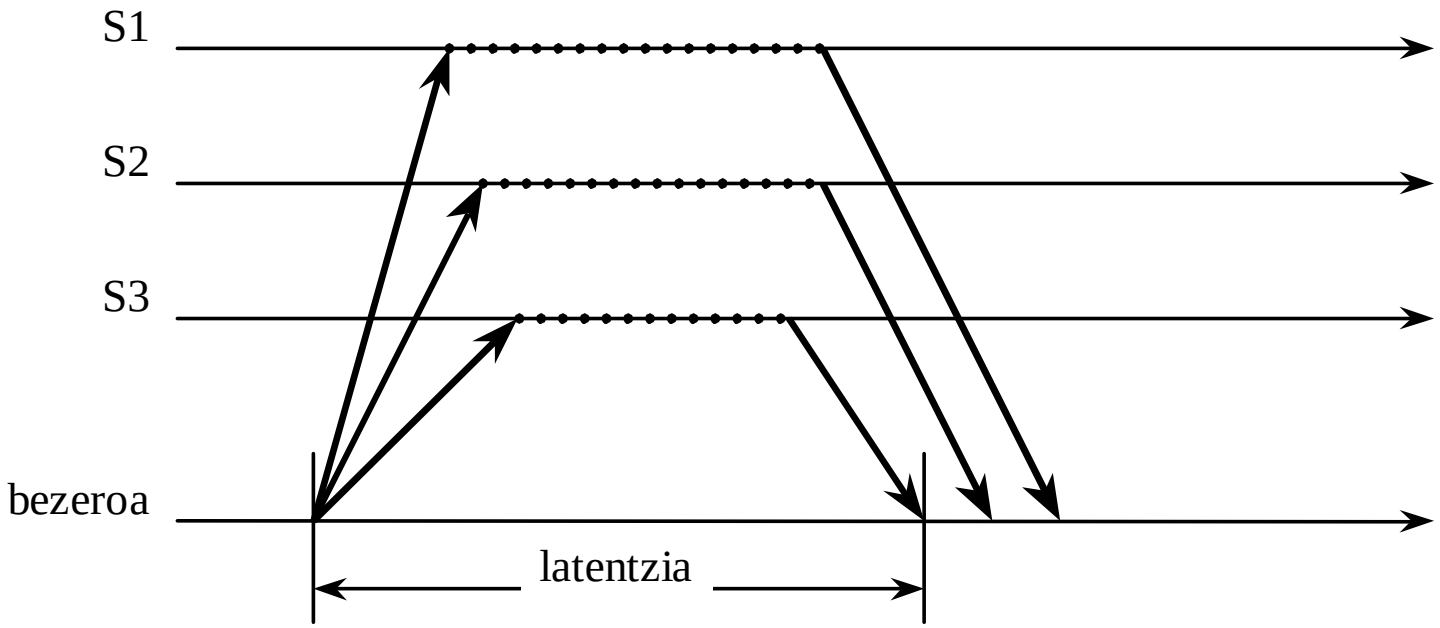
- Erreplikazio teknikak:
  - erreplikazio pasiboa (*primary-backup*)
  - erreplikazio aktiboa

5 Erreplikazio pasiboa



- ordena: nagusiak definitzen du
- atomizitatea: *backup*-en eguneratzea atomikoa izan behar da
- nagusiak huts egiten badu:
  - nagusi berriaren hautaketa (bista-aldaketa)
  - bezeroak eskaria berriro egin beharra gerta daiteke. Eskari bakoitza behin baino gehiago ez burutzea bermatu behar da

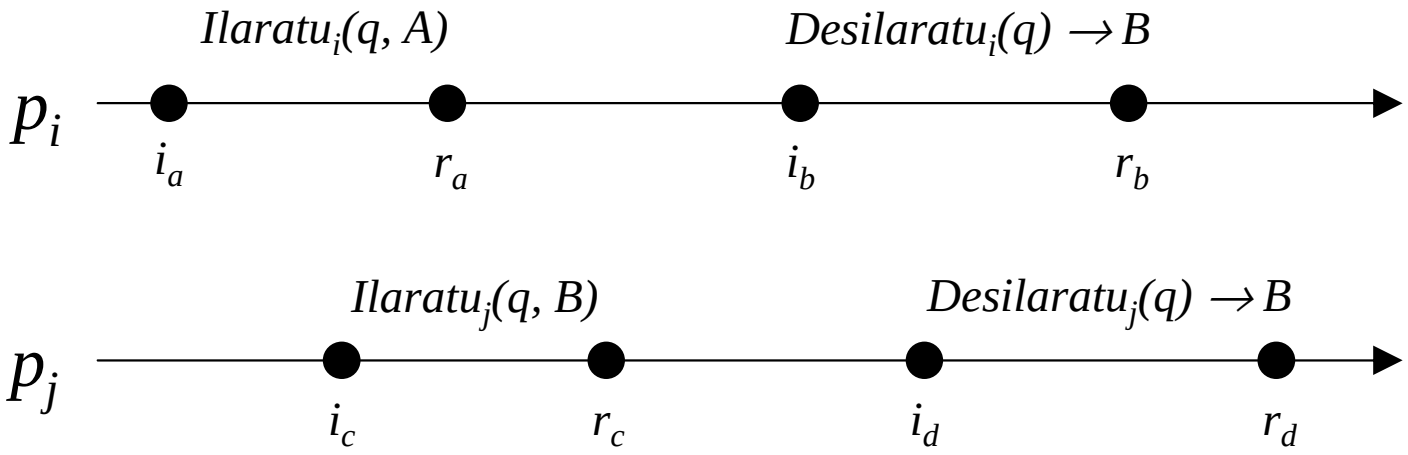
5 Erreplikazio aktiboa



- ordena eta atomizitatea: bezeroek eskariak hedapen atomikoa erabiliz egiten dituzte
- bezeroak lehendabizikoaren erantzunaren zain gelditzen dira
- zerbitzarien hutsegiteak guztiz gardenak dira bezeroentzat
- teorian, erreplikazio pasiboa baino latentzia hobea du
  - baina hedapen atomikoa garestiagoa da

5 Errereplikazio sendoa (9)

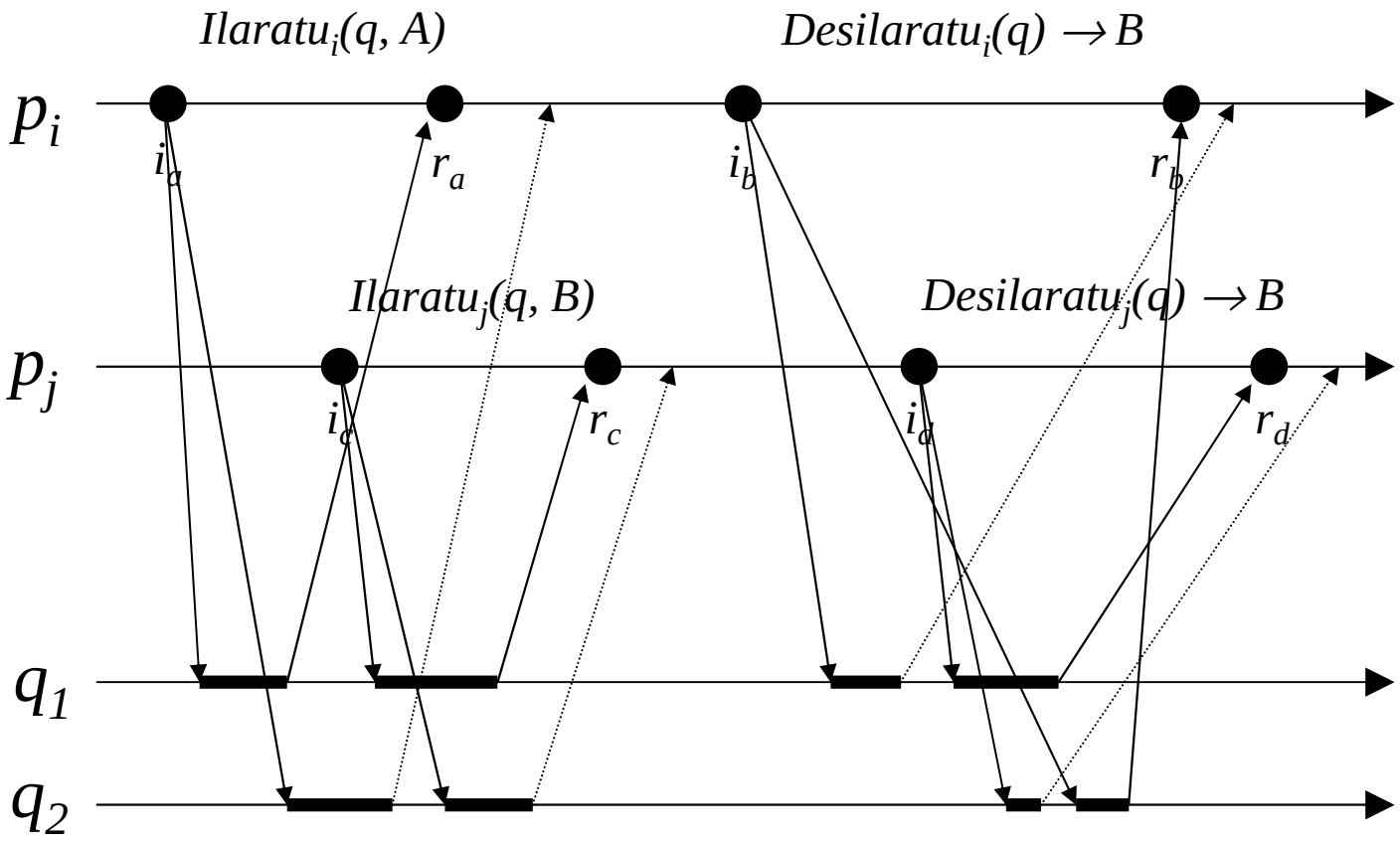
- Ariketa:



  - *q* kola errepikatua (2 kopia)
  - bezero eta zerbitzari bakoitzaren arteko komunikazio mota:
    - fidagarria eta *FIFO*
- Zein sendotasun mota du exekuzioak?
- Nola esplikatu daiteke gertatutakoa?

5 Erreplikazio sendoa (10)

- Ariketa (interpretazioa):



6 Transakzio banatuak

- Transakzioa: operazio sekuentzia, sistemaren egoera era sendo batean aldatzen duena
  - datu-basetan oso erabiliak dira, datuen aldi bereko atzipenak kontrolatzeko
    - datu-basea banatua bada, transakzio banatuak izango ditugu. Honetaz gain, errepikatua denean, kopien arteko sendotasuna bermatu beharko dute
    - eraginkortasuna (*throughput*) lehen helburua da
    - adibidea (*a* eta *b* banketxe kontuak dira):

```
BEGIN_TRANSACTION
    a.withdraw(1000€)
    b.deposit(1000€)
END_TRANSACTION
```

6 Transakzio banatuak (2)

- Transakzioak *ACID* propietateak bete behar dute:
  - *Atomicity* (atomizitatea, “dena ala ezer ez”): transakzio bat osorik burutzen da (*commit*), edo bestela ez du inolako eraginik sistemaren egoeran (*abort*)
  - *Consistency* (sendotasuna): transakzio baten exekuzioaren ondorioz, sistemaren egoera sendoa izaten jarraitzen du (operazio sekuentzia zuzena da)
  - *Isolation* (isolamendua): transakzioak aldi berean exekutatu arren, denak sekuentzian exekutatuko balira bezala gertatzen dira gauzak
  - *Durability* (iraunkortasuna): transakzio baten ondorioak iraunkorrak dira, hutsegiteek eraginik ez dutelarik

6 Transakzio banatuak (3)

- Eredua:
  - bezero batek transakzio bat exekutatu nahi duenean, operazio sekuentzia adierazten dio zerbitzariari
  - operazioak:
    - HASI_TRANSAKZIOA: transakzioaren hasiera markatzen du
    - BUKATU_TRANSAKZIOA: transakzioaren amaiera markatzen du, akordio batera iristeko saioa hasiz (*commit*)
    - ABORTATU_TRANSAKZIOA: transakzioa ez da burutzen, hasi aurreko egoera globalera bueltatzen delarik
    - IRAKURRI_OBJEKTUA / IDATZI_OBJEKTUA: informazioa kudeatzeko operazioak
  - bezeroaren transakzioa zerbitzari banatu guztietara hedatzen da, burutzeko denbora epea adieraziz

The Transaction Model

```
BEGIN_TRANSACTION
  reserve WP -> JFK;
  reserve JFK -> Nairobi;
  reserve Nairobi -> Malindi;
END_TRANSACTION
```

(a)

```
BEGIN_TRANSACTION
  reserve WP -> JFK;
  reserve JFK -> Nairobi;
  reserve Nairobi -> Malindi full =>
ABORT_TRANSACTION
```

(b)

a) Transaction to reserve three flights commits

b) Transaction aborts when third flight is unavailable

6 Transakzio banatuak (4)

- Eredua (jarraipena):
  - partaide bakoitzak transakzioa jasotzen duenean, berari dagokion operazioak burutzen saiatzen da. Arazorik gabe burutzea lortzen badu, bere botoa aldekoa izango da (BAI), eta bestela kontrakoa (EZ)
  - azkenik, partaide guztiek konpromiso atomikorako protokoloa exekutatzen dute, transakzioari buruzko azken erabakia hartzeko (*commit* / *abort*)
    - azken erabakia *commit* bada, orduan partaide guztiek bere egoera eguneratzen dute
    - azken erabakia *abort* bada, orduan inork ez du bere egoera eguneratzen, aurrekoa mantenduz

6 Transakzio banatuak (5)

- Konpromiso atomikorako protokoloen propietateak (sistema sinkronoa suposatzen da):
  - (1) erabakitzen duten partaide guztiek erabaki berdina hartzen dute
  - (2) erabakiak iraunkorrak dira
  - (3) *commit* bakarrik erabaki daiteke partaide guztiek BAI bozkatzen badute
  - (4) partaide guztiek BAI bozkatzen badute eta hutsegiterik ez badago, erabakia *commit* izango da
- Adibidea: 2PC (bi faseko konpromisoa)

6 Transakzio banatuak (6)

- Bi faseko konpromisoa:
  - lehen fasea:
    - koordinatzaileak partaide guztiei beraien botoa eskatzen die
    - partaide bakoitzak koordinatzailearen boto-eskaria itxaroten du. Iristen ez bada (*timeout*), koordinatzaileak hutsegin du eta *abort* erabakitzen du. Aldiz boto-eskaria iristen bada, orduan koordinatzaileari botoa bidaltzen dio
    - partaide guztien botoak BAI badira, koordinatzaileak *commit* erabakitzen du. Aldiz, botoren bat EZ bada edota partaideren bat erantzuten ez badu (*timeout*), orduan *abort* erabakitzen du
  - bigarren fasea:
    - koordinatzaileak partaide guztiei erabakia bidaltzen die
    - erabakia jasotzean, partaide bakoitzak erabaki egiten du

6 Transakzio banatuak (7)

- Bi faseko konpromisoa (hutsegiterik gabe):

```
if pi = coordinator then send <Vote?> to all
when (received <Vote?> from coordinator)
   send <vi,Vote> to coordinator
if pi = coordinator then
   wait until (received <vj,Vote> from every pj)
   if (received <no,Vote> from some pj) then
      decision := abort
   else
      decision := commit
   send <decision,Decision> to all
wait until (received <v,Decision> from coordinator
if pi ≠ coordinator then
   decision := v
```

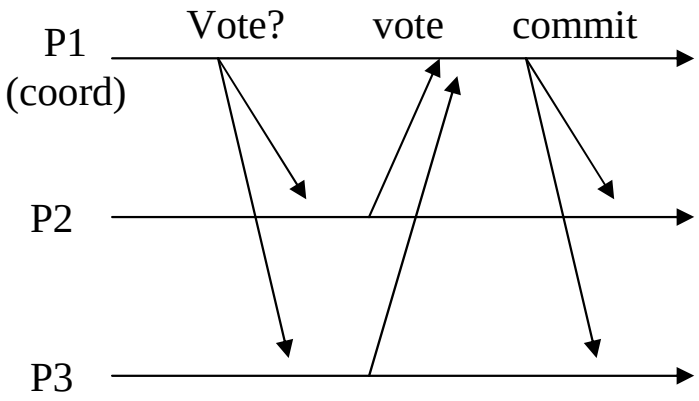
6 Transakzio banatuak (8)

- Bi faseko konpromiso banatua (D2PC, hutsegiterik gabe):

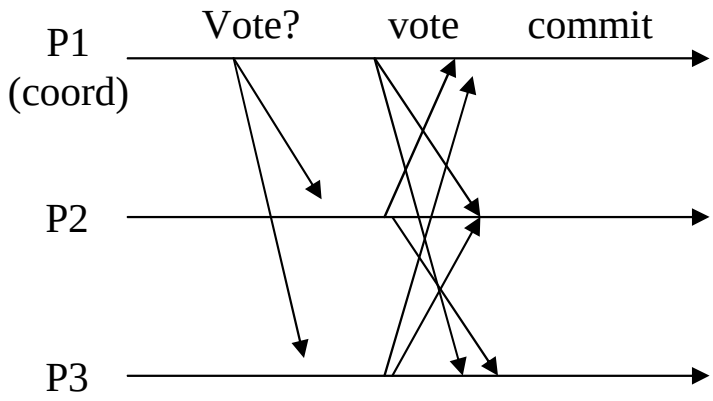
```
if pi = coordinator then send <Vote?> to all
when (received <Vote?> from coordinator)
   send <vi,Vote> to all
wait until (received <vj,Vote> from every pj)
if (received <no,Vote> from some pj) then
   decision := abort
else
   decision := commit
```

6 Transakzio banatuak (9)

- Bi faseko konpromisoa (2PC, D2PC):



2PC

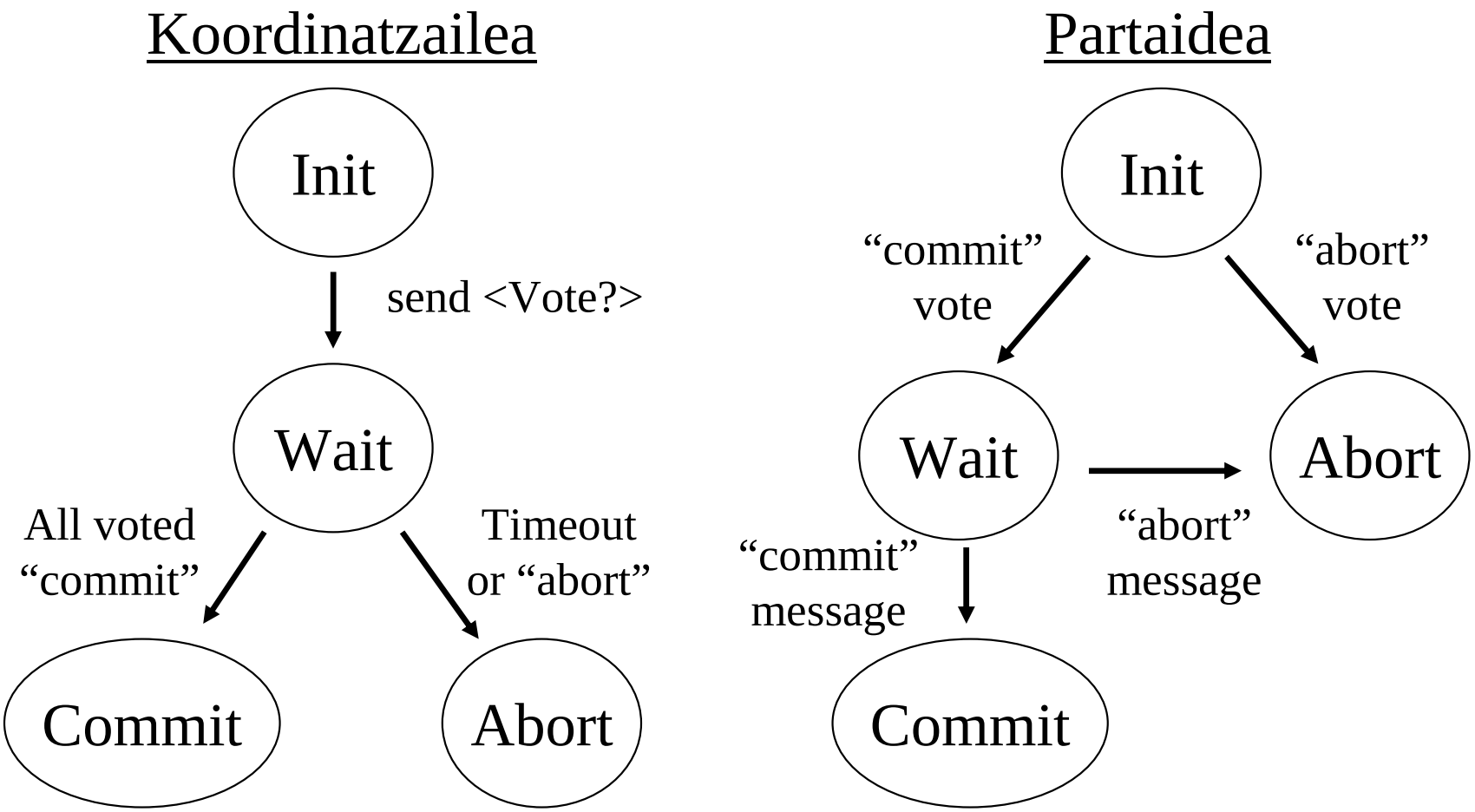


D2PC

- Ariketak:
  - (1) konprobatu biak zuzenak direla, hau da, konpromiso atomikorako propietateak betetzen dituztela
  - (2) konparatu bi protokoloak: mezu kopurua, latentzia

6 Transakzio banatuak (10)

- 2PC-ko egoera diagrama:



6 Transakzio banatuak (11)

- Protokolo ez-blokeagarria (propietate gehigarria):
  - (5) huts egiten ez duten partaide guztiek lehentxeago edo beranduago erabakitzen dute
- Protokolo blokeagarrietan, hutsegin ez duten partaideek hutsegin dutenen konponketaren zain gelditu beharra gerta daiteke
  - bitartean, datuen gaineko sarrailak ezin dituzte askatu
- 2PC eta D2PC protokoloak blokeagarriak dira!!
  - koordinatzaileak (2PC) edota edozein partaidek (D2PC) une kritiko batean huts egiten badu
- 3PC (hiru faseko konpromisoa): ez-blokeagarria

Two-Phase Commit

actions by coordinator:

```
write 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 {
        write 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;
}
```

Two-Phase Commit

actions by participant:

```
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;
}
```

Two-Phase Commit

actions for handling decision requests: /* executed by separate thread */

```
while true {
    wait until any incoming DECISION_REQUEST is received;
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting participant;
    else if STATE == INIT or STATE == GLOBAL_ABORT
        send GLOBAL_ABORT to requesting participant;
    else
        skip;   /* participant remains blocked */
```

7 Adostasunaren arazoa

- Adostasunaren arazoan, prozesu bakoitzak balio bat proposatzen du, eta huts egiten ez dutenek proposatutakoen arteko balio bat adosten dute
- Aplikazioak:
  - hedapen atomikoa: “zein ordenetan entregatu mezuak?”
  - talde partaidetzaren kudeaketa: “zein da partaidetza berria?”
  - hautaketa: “zein da koordinatzaile/lider berria?”
  - konpromiso atomikoa: “*commit* ala *abort* egingo dugu?”
- Espezifikazioa:
  - *proposatu(v)*: prozesuak *v* balioa proposatzen du
  - *erabaki(v)*: prozesuak *v* balio erabakitzen du

7 Adostasunaren arazoa (2)

- Adostasunaren arazoaren propietateak:
  - Bukaera: huts egiten ez duten prozesu guztiek lehentxeago edo beranduago erabakitzen dute
  - Baliotasuna: prozesu batek *v* erabakitzen badu, orduan *v* prozesuren batek proposatutako balioa da
  - Akordioa: huts egiten ez duten bi prozesuk ezin dute balio desberdinak erabaki
    - akordio uniformea: bi prozesuk (zuzenak ala okerrak) ezin dute balio desberdinak erabaki
- Sistema asinkronoetan adostasuna eta hedapen atomikoa baliokideak dira
  - bata bestearen bidez erresolbitu daiteke

7 Adostasunaren arazoa (3)

- FLP ezintasuna: sistema asinkronoetan hutsegiteak gerta badaitezke adostasunak ez du soluzio deterministarik
  - nahiz eta komunikazio kanalak guztiz fidagarriak izan, eta gehienez prozesu bakar batek hutsegin
  - ezintasunaren oinarria: sistema asinkronoetan ezin dira hutsegiteak era fidagarri batean detektatu
- Ezintasuna “gainditzeko” proposamenak:
  - sistema (partzialki) sinkronoak kontsideratu
  - soluzio ez-deterministak (probabilitatean oinarrituta)
  - hutsegite detektatzaileak (guztiz fidagarriak ez direnak)