

Asinkroni model mreže automata

Ivana Podnar

Zavod za telekomunikacije

Fakultet elektrotehnike i računarstva

Sveučilište u Zagrebu

Sadržaj predavanja

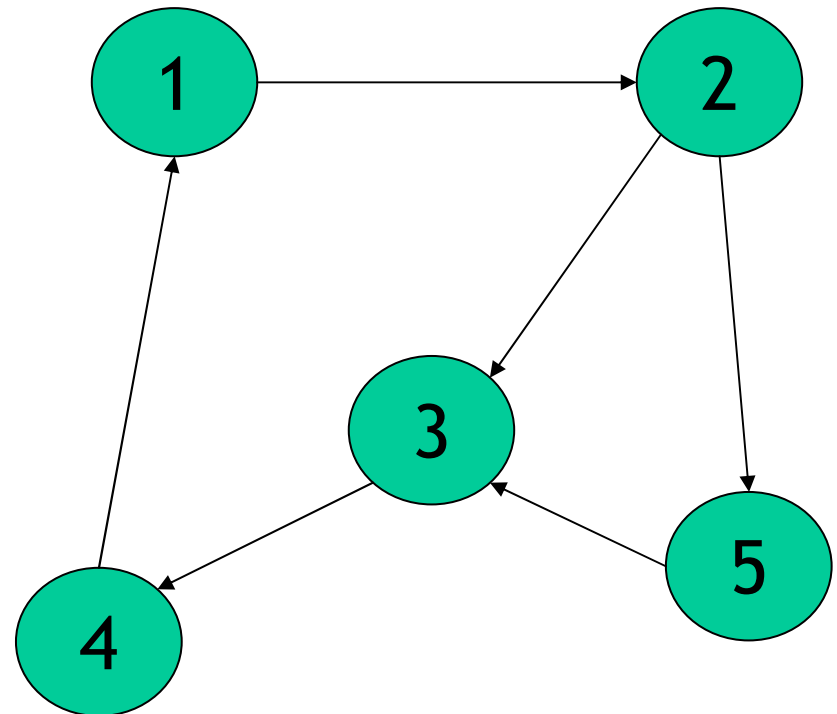


Zavod za telekomunikacije

- ◆ Asinkroni model mreže
- ◆ Primjeri algoritama
- ◆ Sinkronizacija vremena u distribuiranim sustavima
- ◆ Logičko vrijeme

Asinkroni model

- ♦ usmjereni graf $G = (V, E)$
- ♦ $v_i \in V$, čvor modelira proces
- ♦ $e_j \in E$, grana modelira kanal
- ♦ $out-nbrs_i$ - izlazni susjedi
- ♦ $in-nbrs_i$ - ulazni susjedi
- ♦ asinkronost izvođenja procesa i komunikacije (razlika u odnosu na sinkroni model)
- ♦ svaki proces i svaki kanal se modeliraju I/O automatom



- ◆ formalni model za komponente asinkronog sustava
- ◆ I/O automat modelira komponentu distribuiranog sustava koja je u interakciji s ostalim komponentama
- ◆ prijelazi su vezani uz **akcije**
- ◆ akcije mogu biti *ulazne*, *izlazne* ili *interne*

I/O automat A se sastoji od sljedećih komponenti

- ◆ $sig(A)$ - signatura (opis ulaznih, izlaznih i internih akcija)
- ◆ $states(A)$ - skup stanja
- ◆ $start(A)$ - skup početnih stanja, $start(A) \neq 0$
- ◆ $trans(A)$ - funkcija prijelaza
- ◆ $tasks(A)$ - particija zadataka (niti)

- ◆ $S = sig(A)$ - opis ulaznih, izlaznih i unutarnjih akcija
- ◆ $S = (in(S), out(S), int(S))$
 - $in(S)$ - ulazne akcije
 - $out(S)$ - izlazne akcije
 - $int(S)$ - interne akcije
- ◆ $ext(S) = in(S) \cup out(S)$ - eksterne akcije
- ◆ $local(S) = out(S) \cup int(S)$ - lokalne akcije
- ◆ sučelje automata $(in(S), out(S), 0)$

- ◆ $trans(A) \subseteq states(A) \times acts(sig(A)) \times states(A)$
- ◆ za svako stanje s i svaku ulaznu akciju π postoji prijelaz $(s, \pi, s') \in trans(A)$

Izvođenje prijelaza

- ◆ automat A se izvodi kao konačan ili beskonačan slijed stanja i akcija

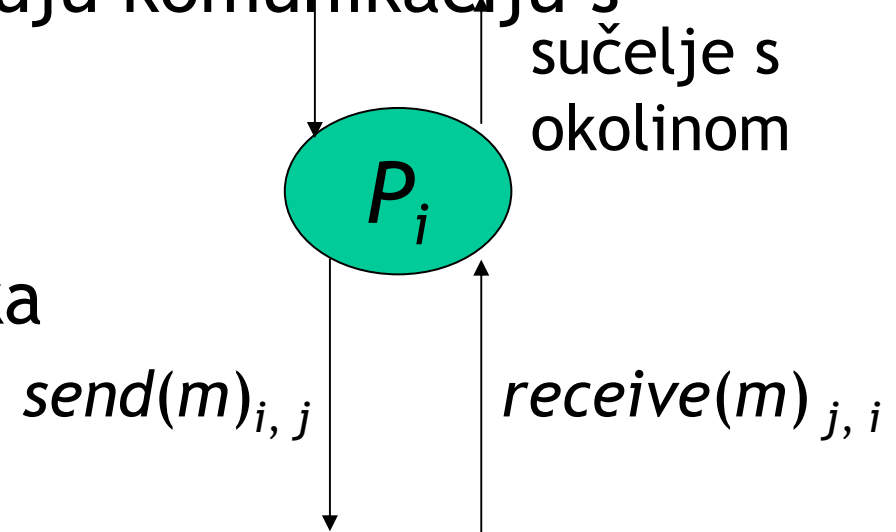
$$s_0, \pi_1, s_1, \pi_2, s_2, \dots, \pi_k, s_k, \dots$$

- ◆ $(s_k, \pi_k, s_{k+1}) \in trans(A)$, za svaki $k \geq 0$

- ◆ proces P_i se modelira I/O automatom (automat modelira interakciju procesa s okolinom)
- ◆ ograničenje automata obzirom na sučelje (ulazne i izlazne akcije koje omogućuju komunikaciju s okolinom)

- ◆ izlaz: $send(m)_{i,j}$
 j - izlazni susjed, m - poruka

- ◆ ulaz: $receive(m)_{j,i}$
 j - dolazni susjed



- ◆ ispad zaustavljanja (*stopping failure*)
 - zaustavlja se izvođenje svih akcija procesa
 - modelira se tako da se procesu dodaje ulazna akcija *stop_i*
- ◆ bizantinski ispad (*Byzantine failure*)
 - modelira se tako da se proces P_i može zamijeniti proizvoljnim I/O automatom s ekvivalentnim vanjskim sučeljem

- ◆ $C_{i,j}$ se modelira I/O automatom
- ◆ ulaz : $send(m)_{i,j}$
- ◆ izlaz: $receive(m)_{j,i}$, j - dolazni susjed
- ◆ ograničenja izvođenja: poslana poruka morala je prethodno biti primljena
- ◆ primjeri kanala:
 - pouzdani FIFO
 - pouzdani kanal s promjenjivim redoslijedom
 - nepouzdan kanal



- ◆ $sig(C_{i,j}) = (send(m)_{i,j}, receive(m)_{i,j}, 0), m \in M$
- ◆ states:
 - *queue*, a FIFO queue
- ◆ trans:
 - $send(m)_{i,j}$ - dodaj m na zadnje mjesto u *queue*
 - $receive(m)_{i,j}$ - preduvjet: m je 1. element iz *queue*, posljedica: briši m iz *queue*
- ◆ tasks: $\{ receive(m)_{i,j} : m \in M \}$

Primjeri izvođenja

- ◆ $[\text{null}], \text{send}(1)_{i,j}, [1], \text{receive}(1)_{i,j}, [\text{null}], \text{send}(2)_{i,j}, [2], \text{receive}(2)_{i,j}, [\text{null}]$
- ◆ $[\text{null}], \text{send}(1)_{i,j}, [1], \text{send}(1)_{i,j}, [11], \text{send}(1)_{i,j}, [111] \dots$

Definicija izvođenja:

Postoji funkcija *cause* koja povezuje akciju *receive* s prethodnom akcijom *send* tako da vrijedi:

1. za svaki *receive* = π , π i *cause*(π) prenose istu poruku *m*: *uvjet za isporuku ispravne poruke*
2. *cause* je surjektivna funkcija: *nema gubitka poruke*
3. *cause* je injektivna funkcija (one-to-one): *nema dupliciranja poruke*
4. *cause* čuva poredak, ne postoje akcije *receive* π_1 i π_2 takve da π_1 prethodi π_2 i da *cause*(π_2) prethodi *cause*(π_1): *nema promjene redoslijeda*

Pouzdani kanal s promjenjivim redosljedom



Zavod za telekomunikacije

- ◆ garantira isporuku svih poruka jednom i samo jednom
- ◆ ne čuva redosljed poruka
- ◆ vrijede prva tri aksioma (1. do 3.) s prethodnog slajda osim 4., ne čuva se poredak

Za nepouzdana kanal mogući su sljedeći ispadi

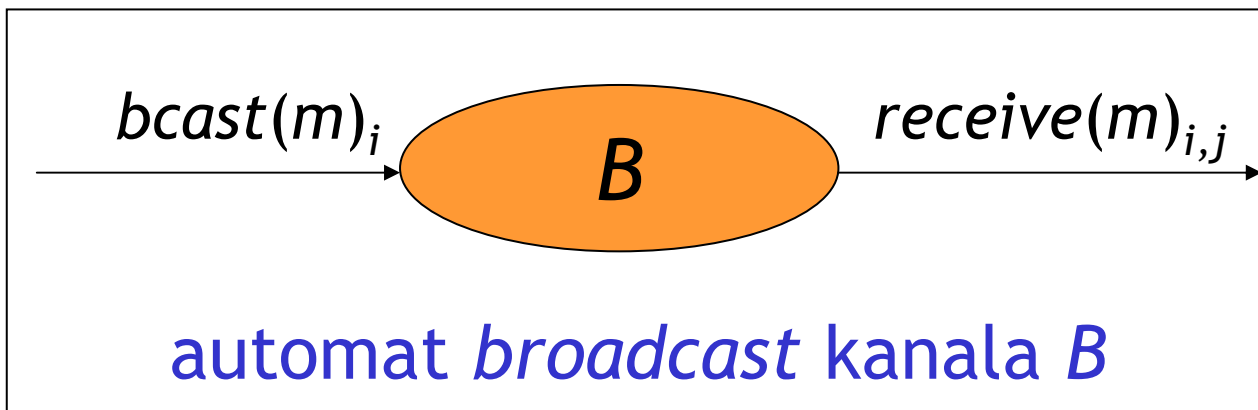
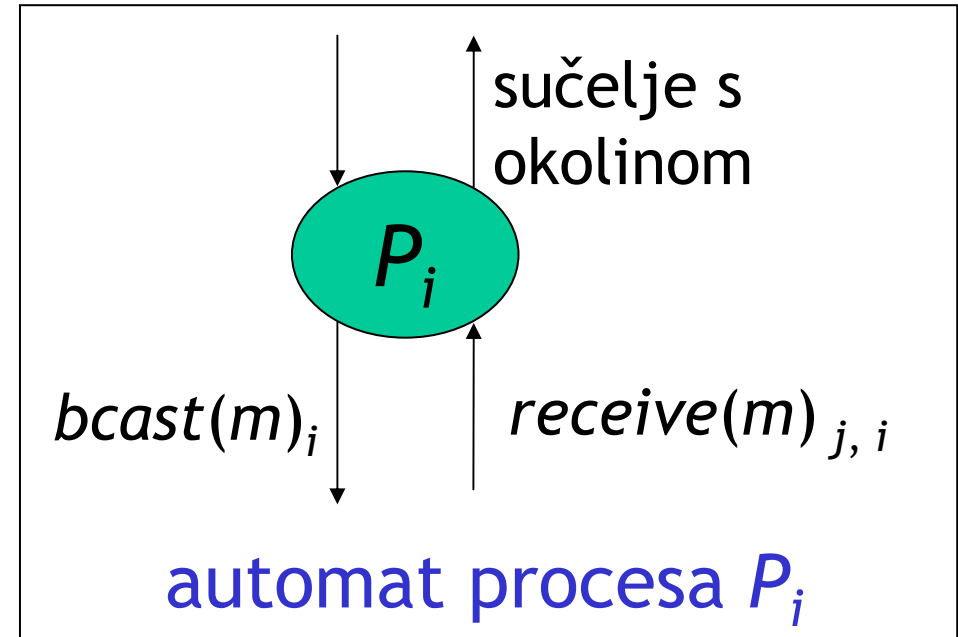
- ◆ gubitak poruke
- ◆ dupliciranje poruke

Modeliranje nepouzdanog kanala

- ◆ gubitak poruke: vrijede aksiomi 1., 3. i 4., brisati aksiom 2.
- ◆ dupliciranja poruke: vrijede aksiomi 1., 2. i 4., brisati aksiom 3.
- ◆ gubitka i dupliciranja poruke: vrijede aksiomi 1. i 4., brisati aksiom 2. i 3.

Kako modelirati broadcast?

- ♦ *broadcast*: proces šalje poruku svim ostalim procesima u mreži
- ♦ skup procesa P_1, P_2, \dots, P_n i 1 jedan *broadcast* kanal



$$1 \leq i, j \leq n$$

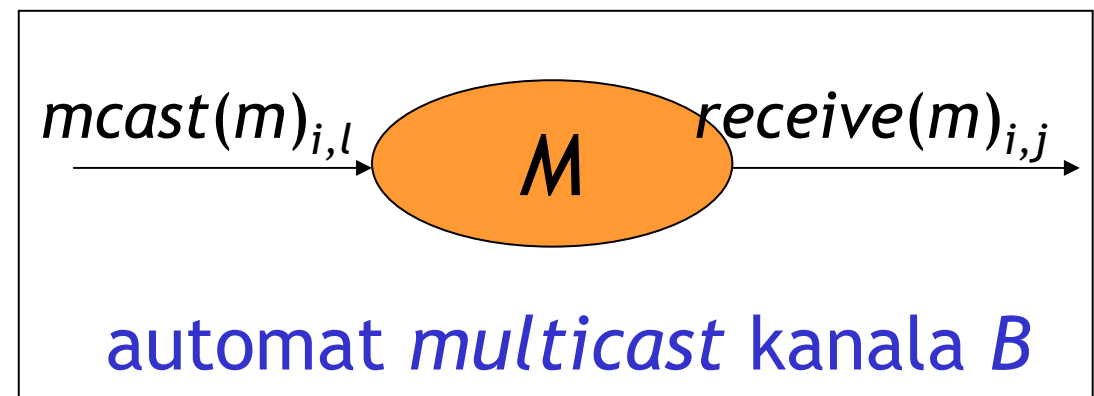
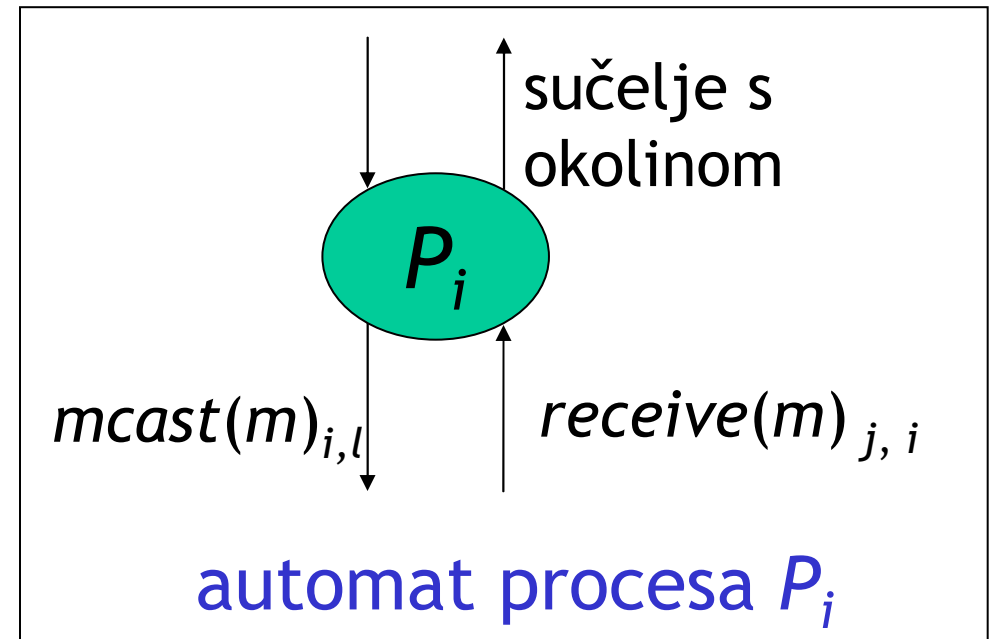
Definicija automata pouzdanog broadcast kanala

- ◆ input: $bcast(m)_i$, $1 \leq i \leq n$, $m \in M$
- ◆ output: $receive(m)_{i,j}$, $1 \leq i, j \leq n$, $m \in M$
- ◆ states: za svaki i, j , $1 \leq i, j \leq n$
 - $queue(i, j)$, a FIFO queue koji sadrži elemente iz M
- ◆ trans:
 - $bcast(m)_i$ - za svaki j dodaj m na zadnje mjesto u $queue(i, j)$
 - $receive(m)_{i,j}$ - preduvjet: m je 1. element iz $queue(i, j)$,
posljedica: briši m iz $queue(i, j)$
- ◆ tasks: za svaki i, j $\{ receive(m)_{i,j} : m \in M \}$

Multicast

- ◆ proces šalje poruku podskupu procesa u mreži
- ◆ skup procesa P_1, P_2, \dots, P_n i 1 jedan *multicast* kanal
- ◆ parovi (i, l) povezuju svaki izvor sa skupom odredišta
 - i je indeks procesa, a l skup indeksa odredišnih procesa
 - proces P_i može koristiti procese definirane skupom indeksa l kao odredišta

$$\begin{array}{l} 1 \leq i \leq n \\ 1 \leq j \leq l \end{array}$$



Definicija automata pouzdanog *multicast* kanala

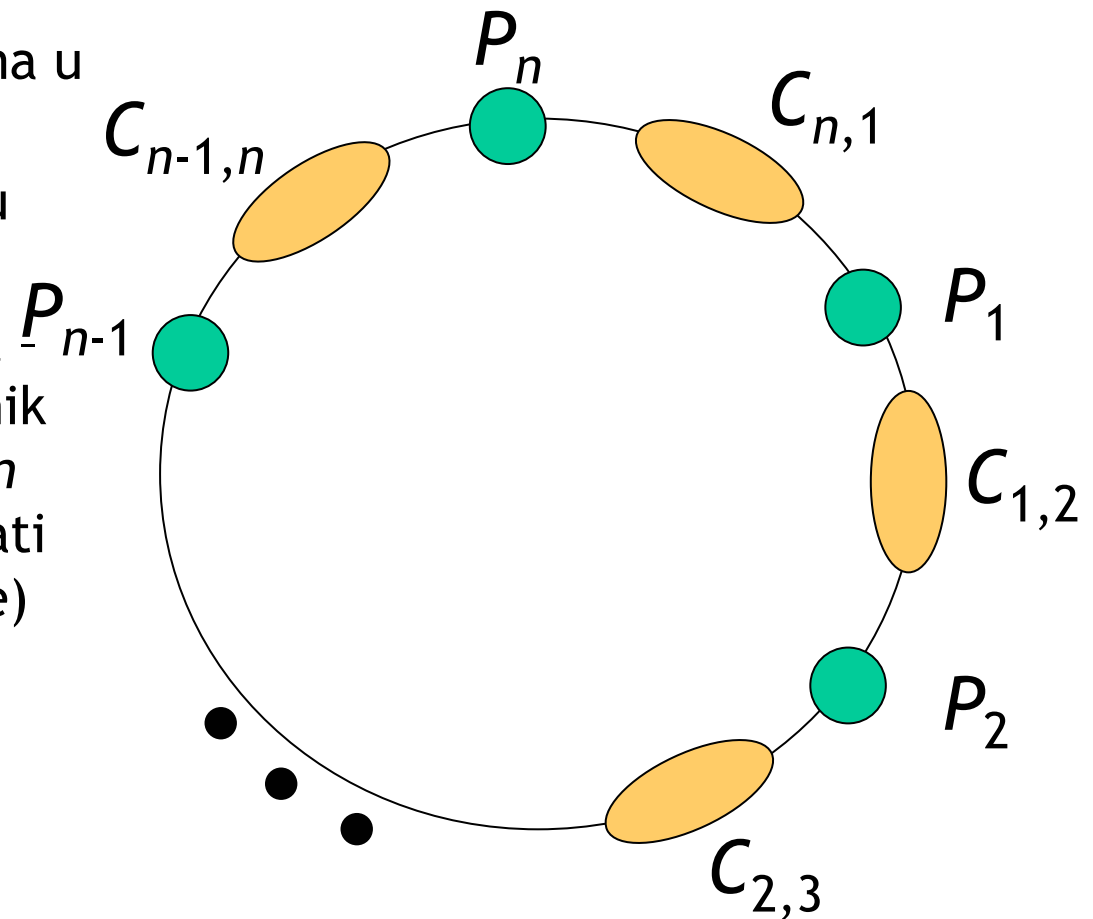
- ◆ input: $mcast(m)_{i,l}$, $(i, l) \in I$, $m \in M$
- ◆ output: $receive(m)_{i,j}$, $1 \leq i, j \leq n$, $m \in M$
- ◆ states: za svaki i, j , $1 \leq i, j \leq n$
 - $queue(i, j)$, a FIFO queue koji sadrži elemente iz M
- ◆ trans:
 - $mcast(m)_{i,l}$ - za svaki $j \in l$ dodaj m na zadnje mjesto u $queue(i, j)$
 - $receive(m)_{i,j}$ - preduvjet: m je 1. element iz $queue(i, j)$,
posljedica: briši m iz $queue(i, j)$
- ◆ tasks: za svaki i, j $\{ receive(m)_{i,j} : m \in M \}$

Primjeri algoritama

Odabir vođe u asinkronoj mreži

Definicija problema

- ◆ Izabrati “vođu” među procesima u mreži
- ◆ Samo 1 proces mijenja status u *leader*
- ◆ adaptacija sinkronog algoritma - svaki proces ima ulazni spremnik koji može primiti maksimalno n poruka (poruke se mogu gomilati zbog asinkronosti komunikacije)
- ◆ procesi: modelirani I/O automatom
- ◆ kanali: pouzdani FIFO



Definicija automata procesa P_i

- ◆ input: $receive(v)_{i-1,i}$, v je UID
- ◆ output: $send(v)_{i,i+1}$; $leader_i$
- ◆ $states_i$:
 - u - UID, inicijalno UID za i
 - $send$ - FIFO queue UID-ova veličine n , inicijalno sadrži UID za i
 - $status \in \{unknown, chosen, reported\}$, inicijalno $unknown$
- ◆ trans:
 - $send(v)_{i,i+1}$ - preduvjet: v je 1. element iz $send$, posljedica: briši v iz $send$
 - $leader_i$ - preduvjet: $status = chosen$, posljedica: $status := reported$
 - $receive(v)_{i-1,i}$
 - if $v > u$: add v to $send$
 - if $v = u$: then $status := chosen$
 - if $v < u$: do nothing

- ◆ kreiranje stabla s izvorišnim čvorom i_0
- ◆ mreža je modelirana grafom $G(V, E)$ koji je neusmjeren i povezan
- ◆ procesi ne znaju dijametar mreže
- ◆ svaki proces treba odrediti prethodnika (*parent*)
- ◆ Skica algoritma
 - Inicijalno je i_0 označen. i_0 šalje *search* svim izlaznim susjedima. Kada proces primi *search* taj proces postaje označen, odabire jedan od susjeda od kojih je primio poruku za *parent* i šalje *search* svim svojim susjedima.

Definicija automata procesa P_i

- ◆ input: $receive("search")_{j,i}$, $j \in nbrs$
- ◆ output: $send("search")_{i,j}$, $j \in nbrs$; $parent(j)_i$, $j \in nbrs$
- ◆ $states_i$:
 - $parent \in nbrs \cup \{null\}$, inicijalno $null$
 - $reported$ - boolean, inicijalno $false$
 - za svaki $j \in nbrs$ postoji
 $send(j) \in \{search, null\}$, inicijalno $search$ ako je $i = i_0$ inače $null$
- ◆ trans:
 - $send("search")_{i,j}$ - preduvjet: $send(j) = search$, posljedica: $send(j) := null$
 - $parent(j)_i$ - preduvjet: $parent = j$, $chosen = false$, posljedica: $reported := true$
 - $receive("search")_{j,i}$
 - if $i \neq i_0$ and $parent = null$
 - $parent := j$
 - for all $k \in nbrs \setminus \{j\}$
 - $send(k) := search$

- ◆ *broadcast*

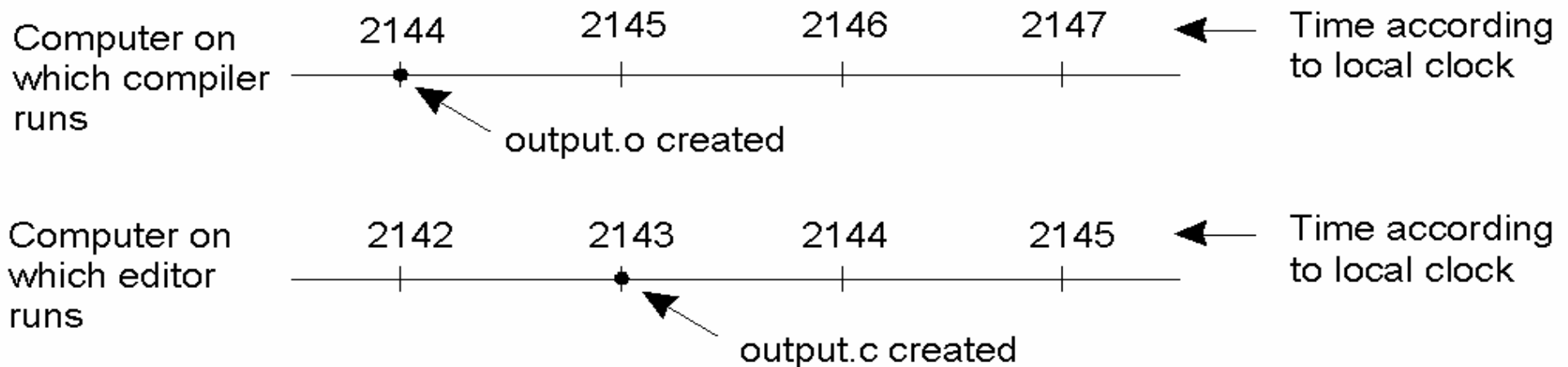
- poruku treba prenijeti zajedno sa *search*

- ◆ *convergecast*

- usmjeravanje poruka od bilo kojeg procesa u mreži prema i_0 , svaki proces prima poruke od svih čvorova sljedbenika, kreira jedinstvenu poruku na temelju primljenih i prosljeđuje je svom prethodniku
- dodati svakom procesu pokazivače na čvorove sljedbenike (“*child node*”)
- implementacija: na svaki primljeni *search* proces odgovara s *parent* ili *non-parent*

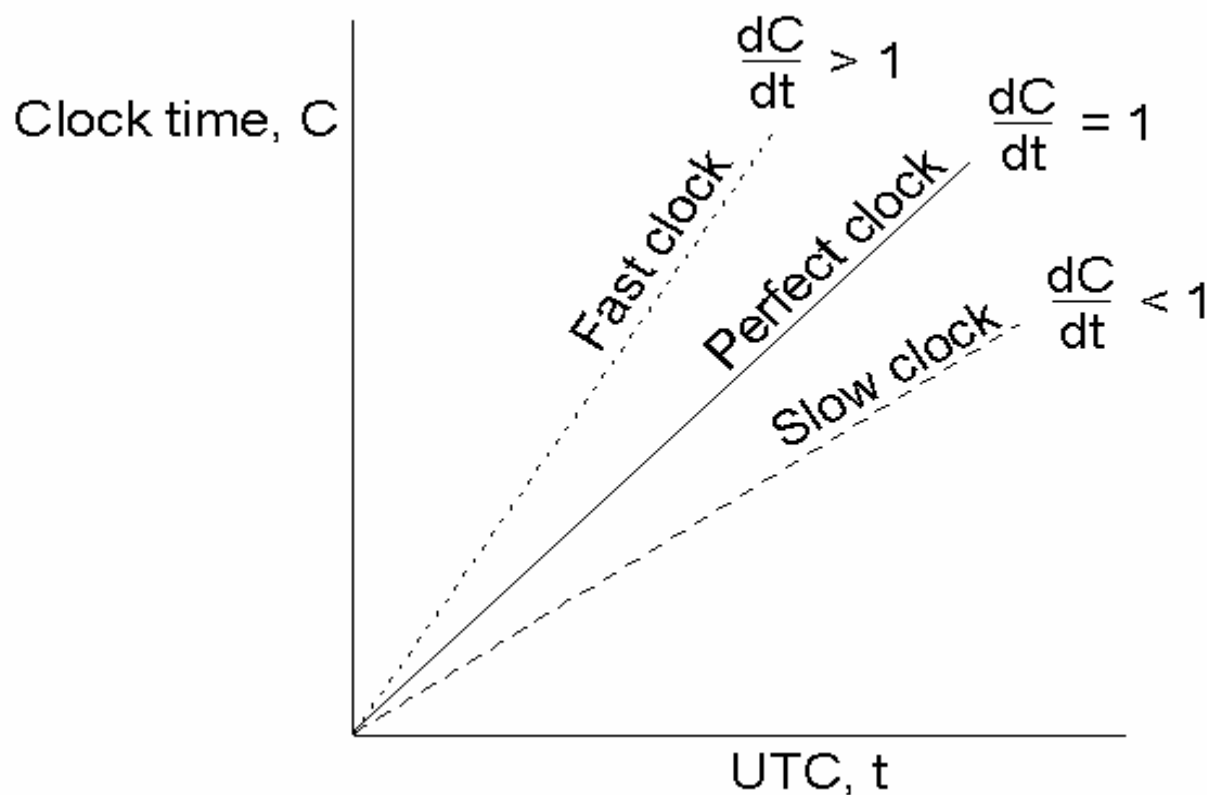
Sinkronizacija vremena u distribuiranim sustavima

Sinkronizacija vremena



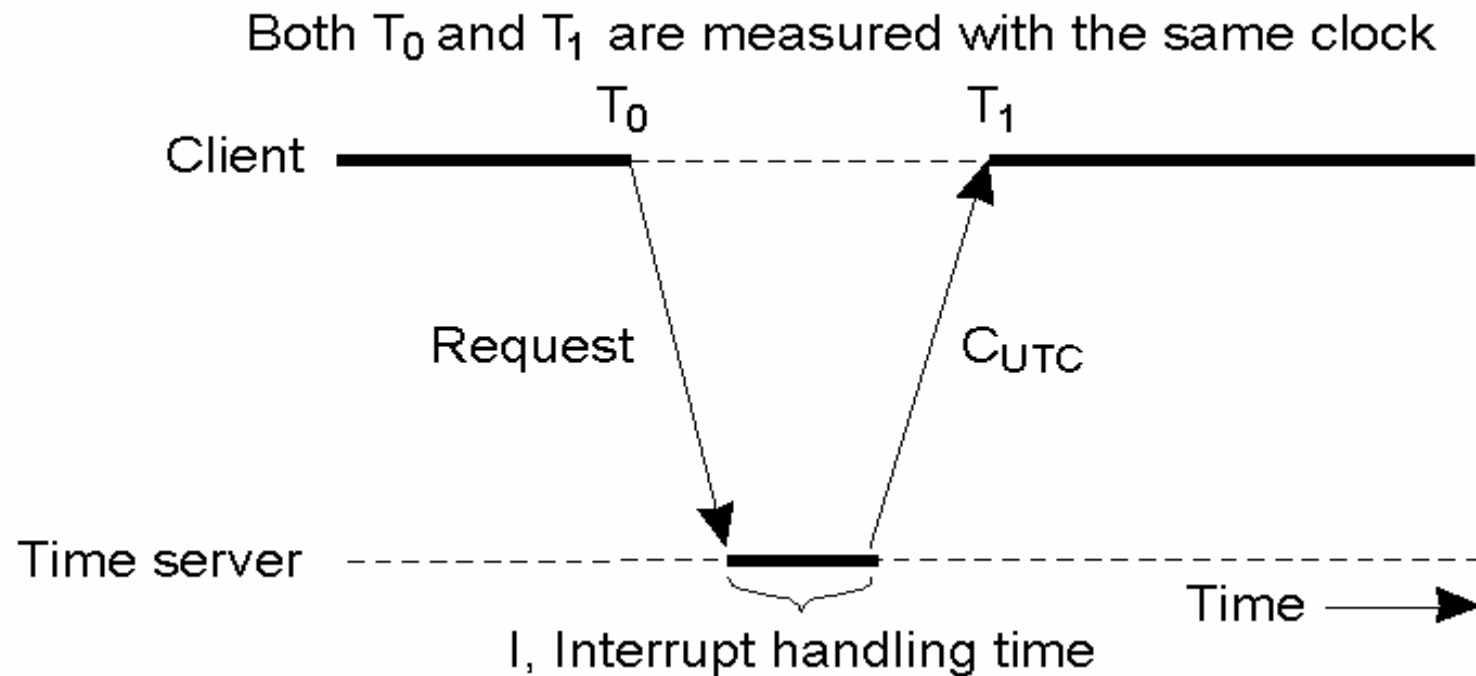
- ◆ obzirom da svako računalo ima vlastiti sat, događaj koji se zbio nakon nekog događaja u realnom vremenu može biti interpretiran kao da se dogodio prije

Algoritmi za sinkronizaciju vremena



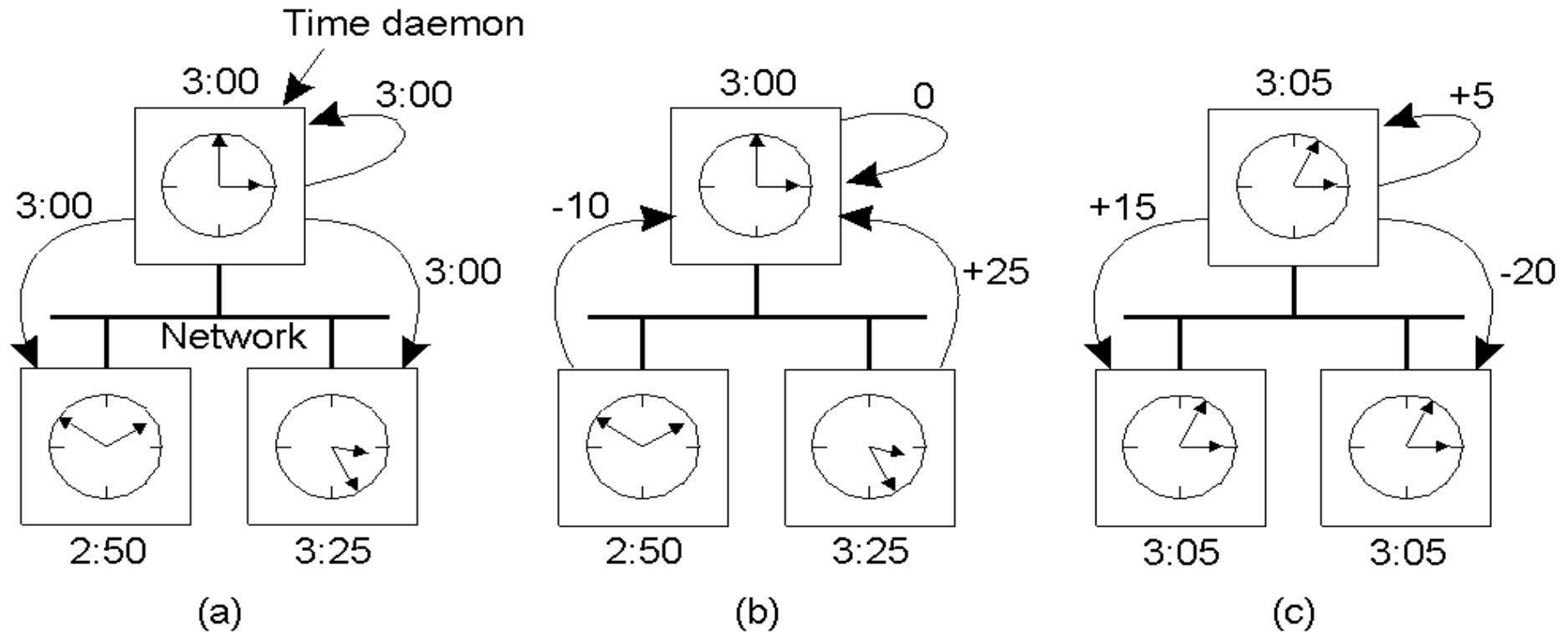
- ◆ Universal Coordinated Time (UTC)

Cristian's Algorithm



- ◆ problem ako je C_{UTC} manje od vremena na klijentu -> treba postepeno usporiti sat na klijentu
- ◆ procjena vremena za prijenos poruke $(T_1 - T_0)/2$

Berkeley Algorithm



- time daemon* šalje upit računalima u mreži o trenutnom vremenu
- računala šalju odgovor
- time daemon* na temelju primljenih vrijednosti (npr. srednje vrijeme) definira kako računala trebaju promijeniti vrijeme

Logičko vrijeme

- ◆ u model asinkrone mreže nije ugrađen pojam realnog vremena
- ◆ procesi ne moraju biti sinkronizirani u realnom vremenu, već međusobno sinkronizirani
- ◆ važan je redosljed događaja, procesi trebaju moći odrediti koji se događaj dogodio prije nekog drugog događaja (*partial order* \neq *total order*)
- ◆ svakom događaju se može pridijeliti logičko vrijeme iz skupa T , elementi iz T su u rastućem slijedu

Proces P_i opisan slijedom izvođenja α

1. dva događaja ne mogu imati jednako logičko vrijeme;

$$t(a) < t(b) \Rightarrow a \text{ se dogodio prije } b$$

$$t(a) = t(b) \Rightarrow a = b$$

2. logička vremena procesa su uvijek u rastućem nizu

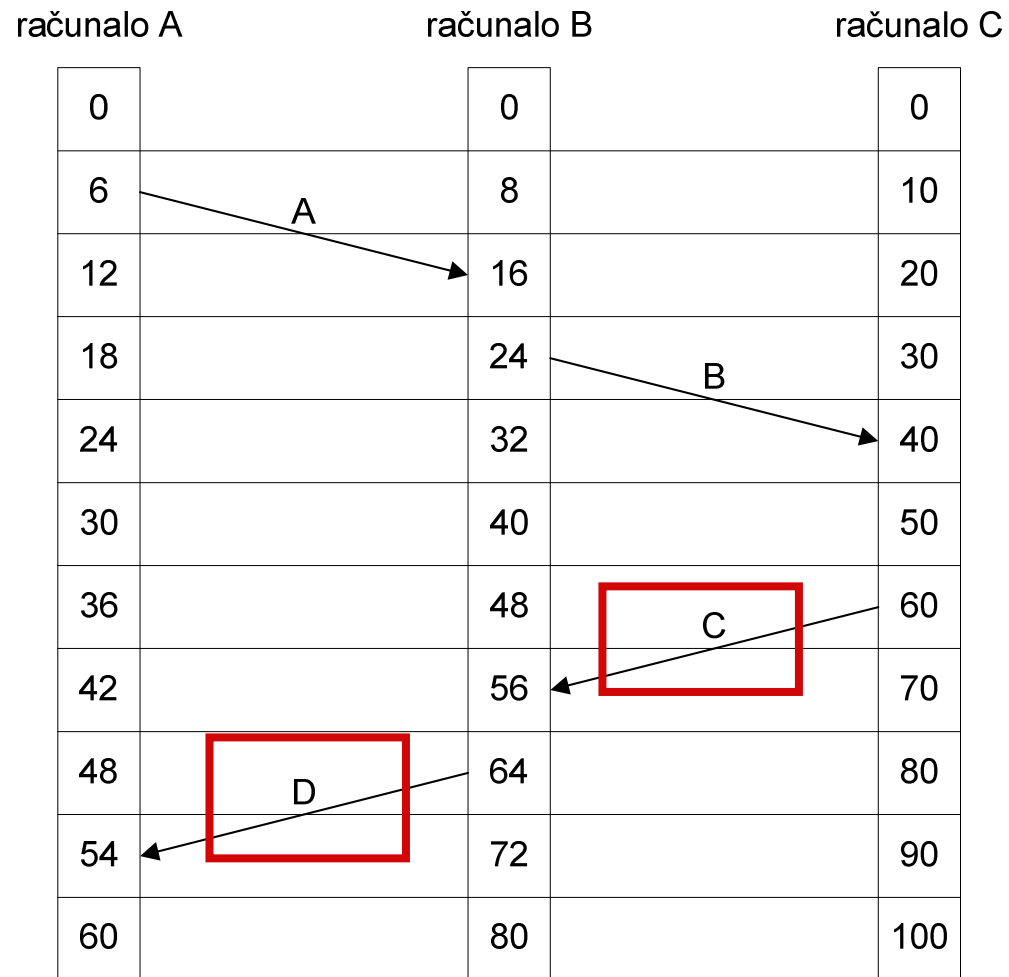
Mreža send/receive

1. logičko vrijeme za *send* je uvijek manje od odgovarajućeg *receive* (poruka ne može biti primljena prije nego je poslana)
2. u nekom trenutku t postoji uvijek konačan broj događaja kojima su pridijeljena logička vremena manja od t

Mreža broadcast

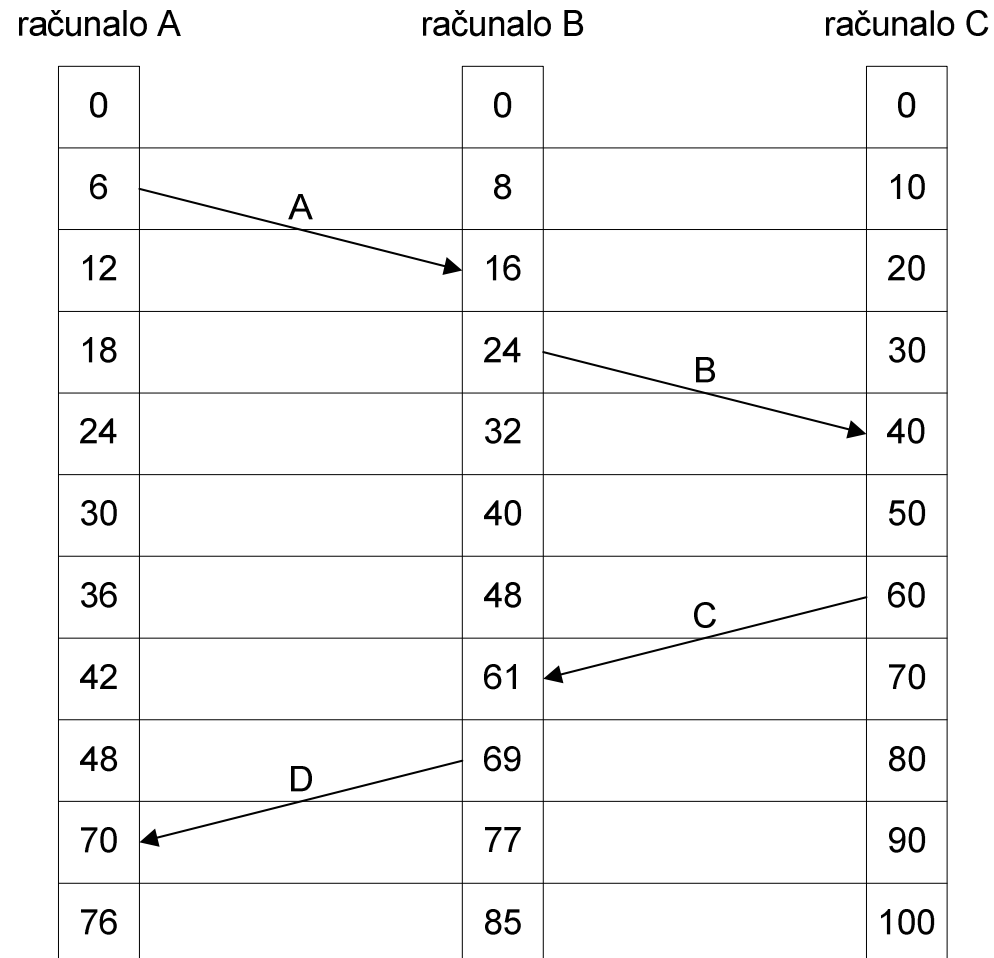
1. logičko vrijeme za *bcast* je uvijek manje od odgovarajućeg *receive*

Lamport Timestamps



Lamport Timestamps

računalo A	računalo B	računalo C
0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	61	70
48	69	80
70	77	90
76	85	100



LamportTime

- ◆ proces $LamportTime(A)_i$ uz stanje procesa A_i održava i lokalnu varijablu $clock$ (inicijalno je 0)
- ◆ $clock$ se povećava za 1 kod svakog događaja procesa A_i
- ◆ kada proces izvodi $send$, šalje $v = clock + 1$ zajedno s porukom
- ◆ kada proces izvodi $receive$, povećava $clock$ tako da uvijek bude veći od primljenog v

WelchTime

- ◆ svaki proces održava lokalnu varijablu *clock*
- ◆ kada proces izvodi *send*, šalje *clock* zajedno s porukom
- ◆ svaki proces *i* održava FIFO *receive-buffer* koji sadrži poruke s vremenima koja su veća od lokalne varijable *clock*
- ◆ svaka poruka kojoj je timestamp manji od lokalne varijable *clock*-a se odmah obrađuje, ostale čekaju u spemniku