SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Diplomski studij
Računarstvo
Znanost o mrežama
Programsko inženjerstvo i informacijski sustavi
Računalno inženjerstvo
Ostali (slobodni izborni predmet)

Raspodijeljeni sustavi

4. Formalni model raspodijeljenog sustava i primjeri raspodijeljenih algoritama

Ak. god. 2022./2023.

Creative Commons

- slobodno smijete:
  - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo
- pod sljedećim uvjetima:
  - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

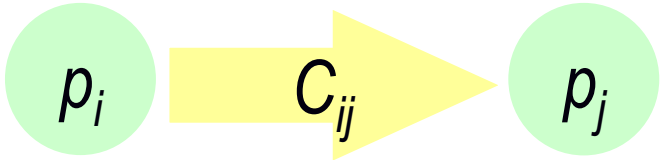
U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.
Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava. *Tekst licence preuzet je s http://creativecommons.org/*

Sadržaj predavanja

- Model raspodijeljenog sustava
  - model raspodijeljenog izvođenja
  - uzročna ovisnost događaja
  - globalno stanje raspodijeljenog sustava
  - raspodijeljeni algoritam
- Proširenje osnovnog modela raspodijeljenog sustava
  - sinkroni model
  - asinkroni model

Osnovni model raspodijeljenog sustava

- skup autonomnih procesa p_1, p_2, …, p_n
- C_{ij} –kanal koji povezuje procese p_i i p_j
- m_{ij} – poruka od p_i za p_j

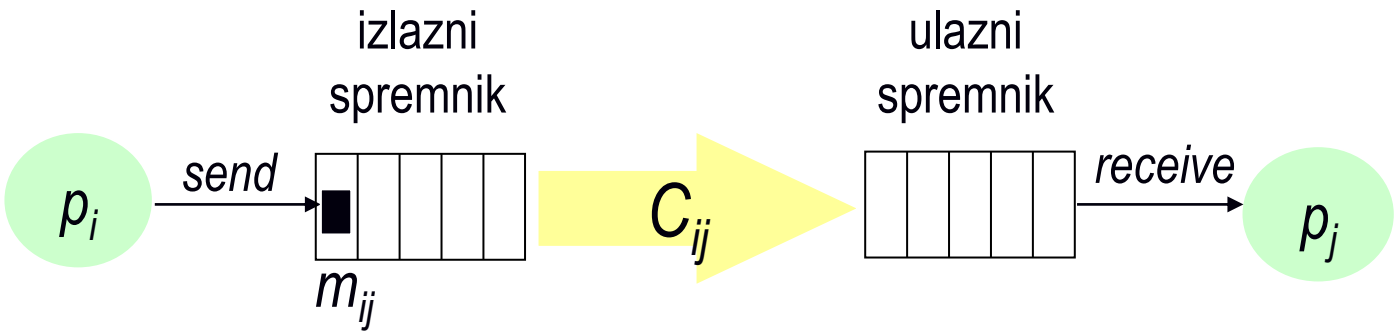


Svojstva

- Izvođenje procesa i prijenos poruka su asinkroni
- Procesi ne dijele zajednički memorijski prostor
- Pri komunikaciji procesa neminovno se javlja kašnjenje
- Procesi ne koriste jedinstveni globalni sat

Komunikacija procesa

- procesi međusobno komuniciraju razmjenom poruka (*message passing*) preko komunikacijskog medija (komunikacijske mreže)



- procesi koriste operatore *send* i *receive*
- *send*: pohranjuje poruku u izlazni spremnik i priprema za prijenos preko kanala
- *receive*: čita poruku iz dolaznog spremnika i prosljeđuje procesu

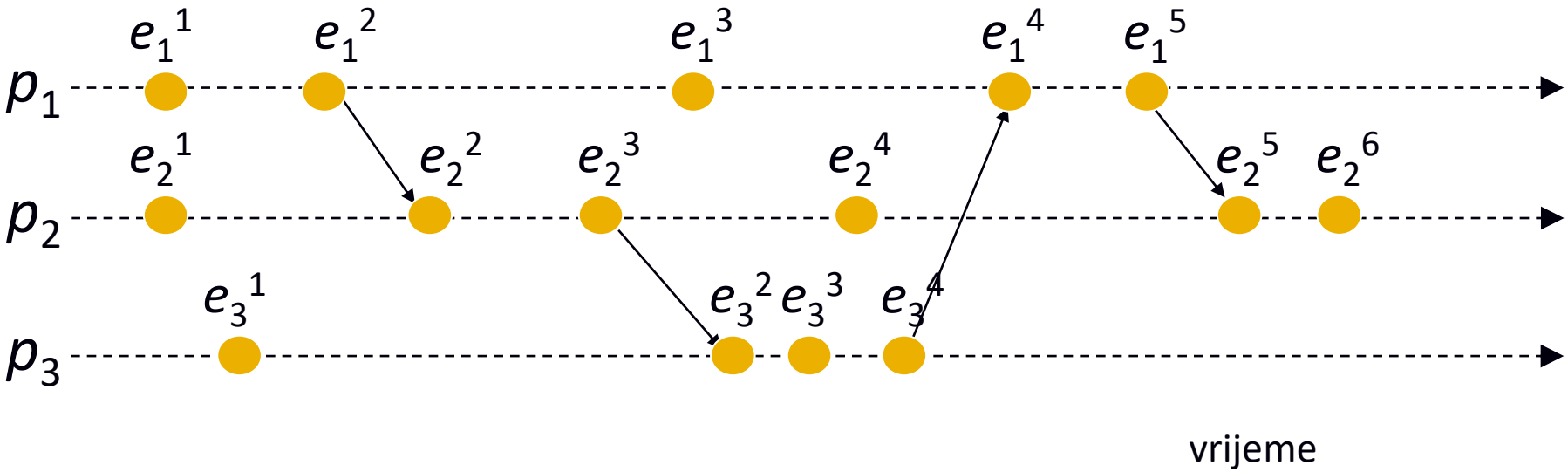
Model raspodijeljenog izvođenja

- Izvođenje procesa: slijedno izvođenje akcija procesa
- **Akcije** se modeliraju sljedećim događajima:
  - unutarnji događaj
  - slanje poruke
  - primanje poruke
- Događaj mijenja stanje procesa i komunikacijskog kanala
- Slijed događaja na procesu p_i:

$$e_i^1, e_i^2, e_i^3, \ldots, e_i^x$$

(e_i^2 se dogodio prije e_i^3)

Primjer raspodijeljenog izvođenja



Uzročna ovisnost događaja (1)

- Uzročna relacija ovisnosti događaja (oznaka: →)
  - izražava uzročnu ovisnost između dva događaja tijekom raspodijeljenog izvođenja, uzročnost može biti direktna ili tranzitivna
- $e_i^x \longrightarrow e_i^y$
  - događaj e_i^x je izvršen na procesu p_i prije događaja e_i^y te su oni uzročno povezani (e_i^x se nužno dogodio prije e_i^y)
- $send(m) \longrightarrow_{msg} receive(m)$
  - uzročna ovisnost vezana uz slanje i primanje poruke, da bi poruka bila primljena, mora prethodno nužno biti poslana na kanal
- $e_i^x \longrightarrow e_k^z \wedge e_k^z \longrightarrow e_j^y \Rightarrow e_i^x \longrightarrow e_j^y$
  - primjer tranzitivne uzročnosti događaja izvršenih na 3 različita procesa

Uzročna ovisnost događaja (2)

- Kada su 2 događaja uzročno ovisna?

$$e_i^x \rightarrow e_j^y \Leftrightarrow \begin{cases} e_i^x \rightarrow e_j^y, (i = j) \wedge (x < y) & \text{slijedni događaji na istom procesu} \\ e_i^x \rightarrow_{msg} e_j^y & \text{slanje i primanje poruke } msg \\ e_i^x \rightarrow e_k^z \wedge e_k^z \rightarrow e_j^y & \text{tranzitivna uzročnost} \end{cases}$$

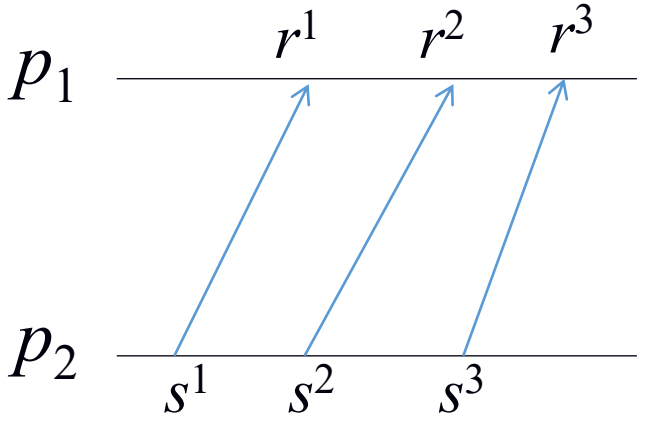
Uzročna neovisnost događaja

- Uzročna relacija neovisnosti dvaju događaja (oznaka: $\not\rightarrow$)
  - označava neovisnost dvaju događaja tijekom raspodijeljenog izvođenja
- $e_i \not\rightarrow e_j$
  - događaj e_j nije ovisan o događaju e_i
- Vrijede sljedeća pravila
  1. za 2 događaja e_i i e_j, $e_i \not\rightarrow e_j \not\Rightarrow e_j \not\rightarrow e_i$
  2. za 2 događaja e_i i e_j, $e_i \rightarrow e_j \Rightarrow e_j \not\rightarrow e_i$
  3. ako za 2 događaja e_i i e_j, vrijedi $e_i \not\rightarrow e_j$ i $e_j \not\rightarrow e_i$, onda su e_i i e_j konkurenti događaji i to možemo napisati na sljedeći način $e_i \,||\, e_j$

Model komunikacijskog kanala (1/3)

FIFO (first-in, first-out)

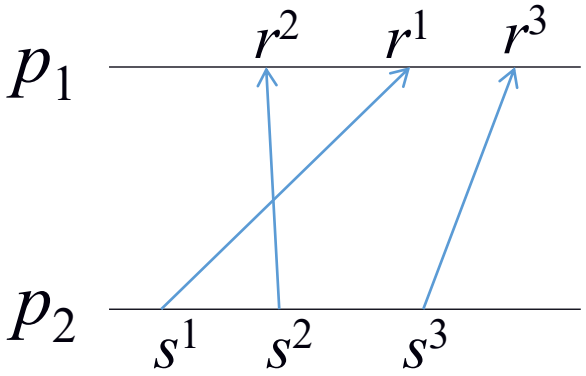
- kanal čuva slijednost poruka, ponaša se kao rep



FIFO

non-FIFO

- kanal ne čuva slijednost poruka, ponaša se kao skup

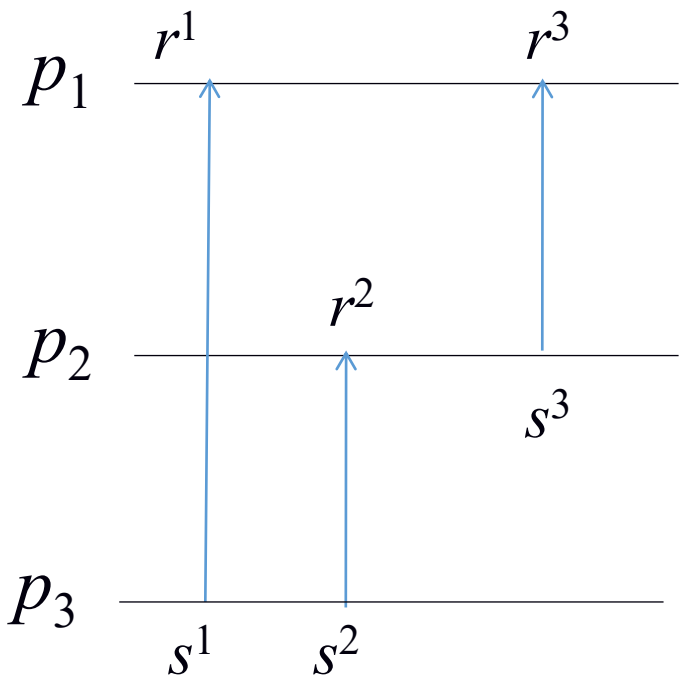


non-FIFO

Model komunikacijskog kanala (2/3)

sinkrona slijednost

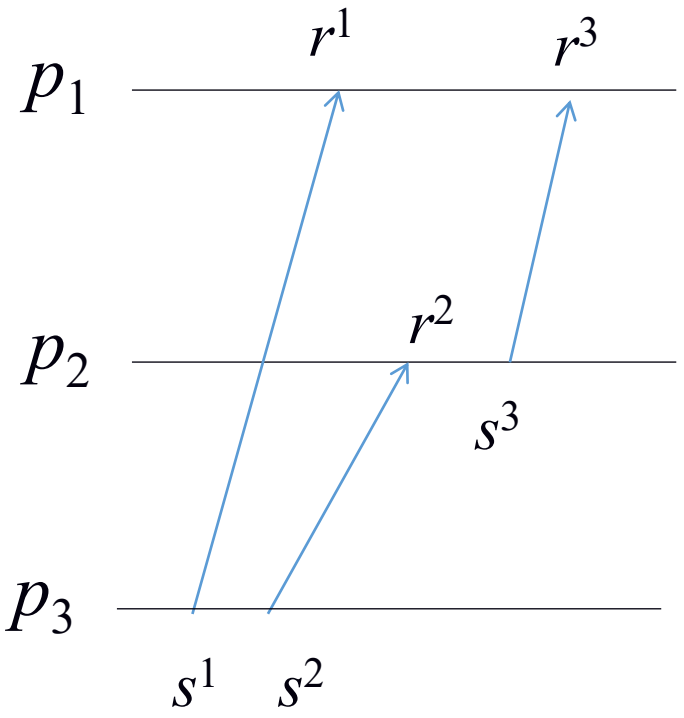
- slanje i primanje poruke događa se istovremeno



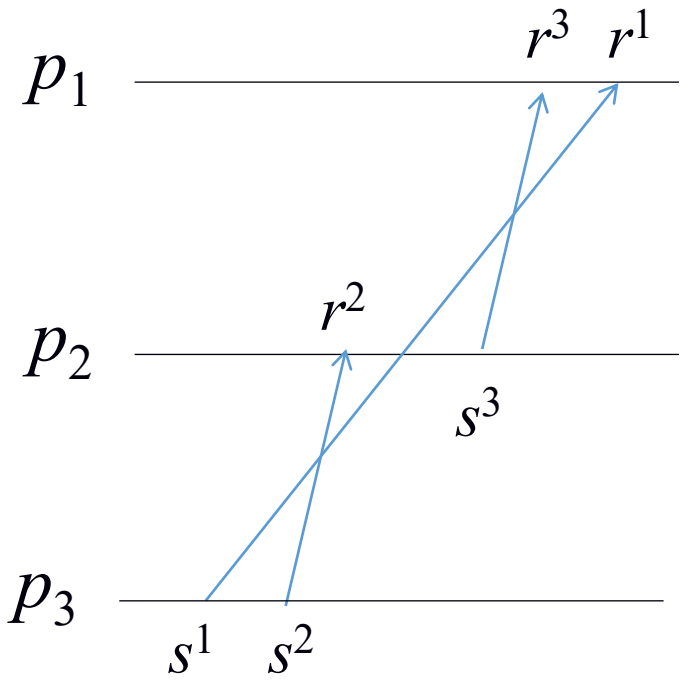
Model komunikacijskog kanala (3/3)

uzročna slijednost (*causal ordering*, CO)

- osigurava da uzročno povezani događaji slanja dviju poruka istom primatelju rezultiraju primanjem u slijedu kojim su poslani
  - ako su dva događaja slanja poruke istom primatelju uzročno povezana (kao s^1 i s^3) onda primanje poruka mora slijediti redoslijed slanja (r^1 se mora dogoditi prije r^3 kako bi bilo zadovoljeno svojstvo CO)
- koristan za razvoj distribuiranih algoritama, pojednostavljuje razvoj jer ima ugrađeni mehanizam „sinkronizacije”
  - npr. replicirana baza podataka, svaki proces koji osvježava repliku mora primiti zahtjeve za update u istom slijedu (važno zbog konzistentnosti baze)

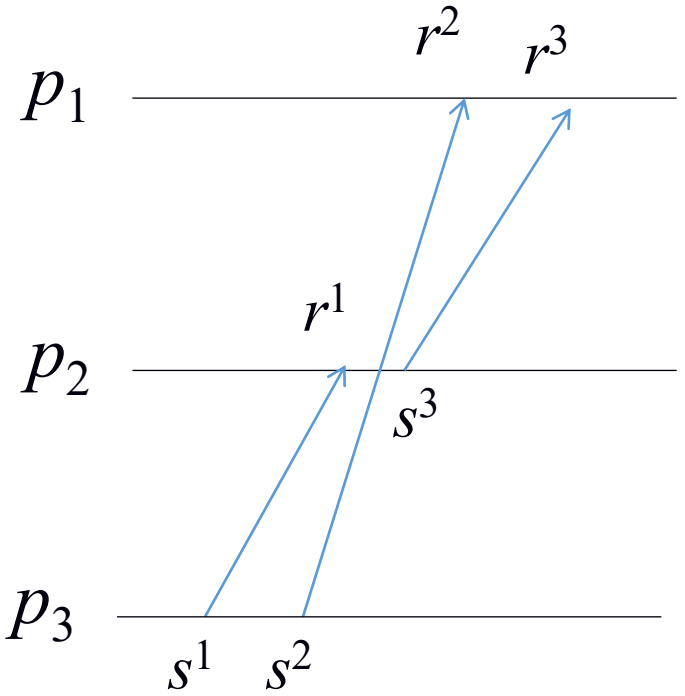


Primjer izvođenja non-CO i CO



non-CO

$s^1 \rightarrow s^3$, jer na p_1 se dogodilo $r^3 \rightarrow r^1$



CO

$s^1 \rightarrow s^2$, $s^1 \rightarrow s^3$ ali odredišta poruka se razlikuju, a kada analiziramo r^2 i r^3, poruke slanja s^2 i s^3 su neovisne pa je njihov redoslijed irelevantan

Globalno stanje raspodijeljenog sustava

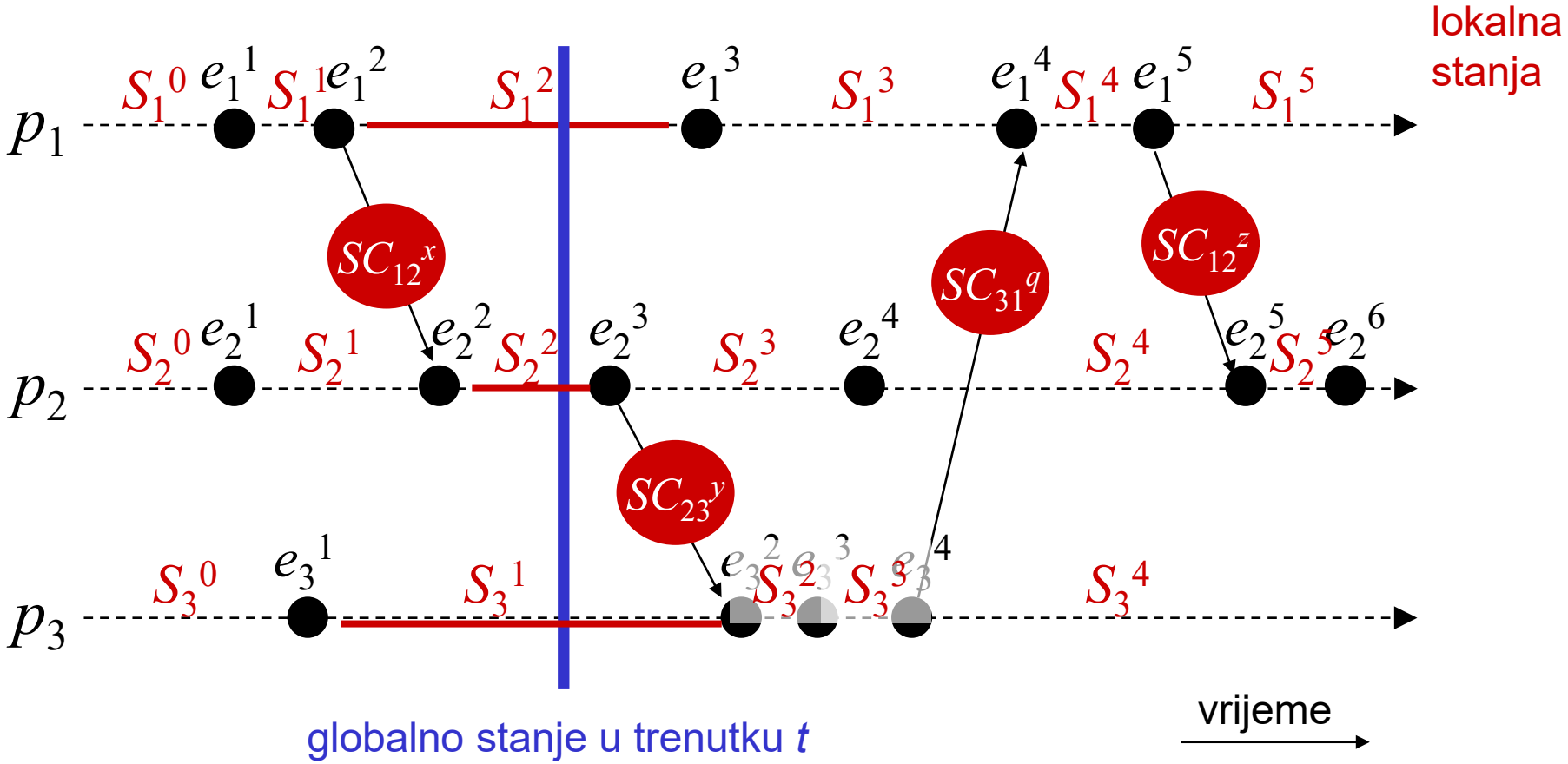
Lokalno stanje procesa ili kanala

- Stanje procesa određeno je stanjem lokalne memorije i izvođenjem unutarnjih događaja, lokalno stanje procesa je potpuno privatno (ne može ga mijenjati drugi proces)
- Stanje kanala određeno je skupom primljenih i poslanih poruka

Globalno stanje

- Određeno (trenutnim) lokanim stanjima svih procesa i kanala, kontinuirano se mijenja uslijed akcija tj. događaja na procesima i kanalima
- Izvođenje događaja mijenja lokano stanje procesa/kanala te istovremeno mijenja i globalno stanje raspodijeljenog sustava

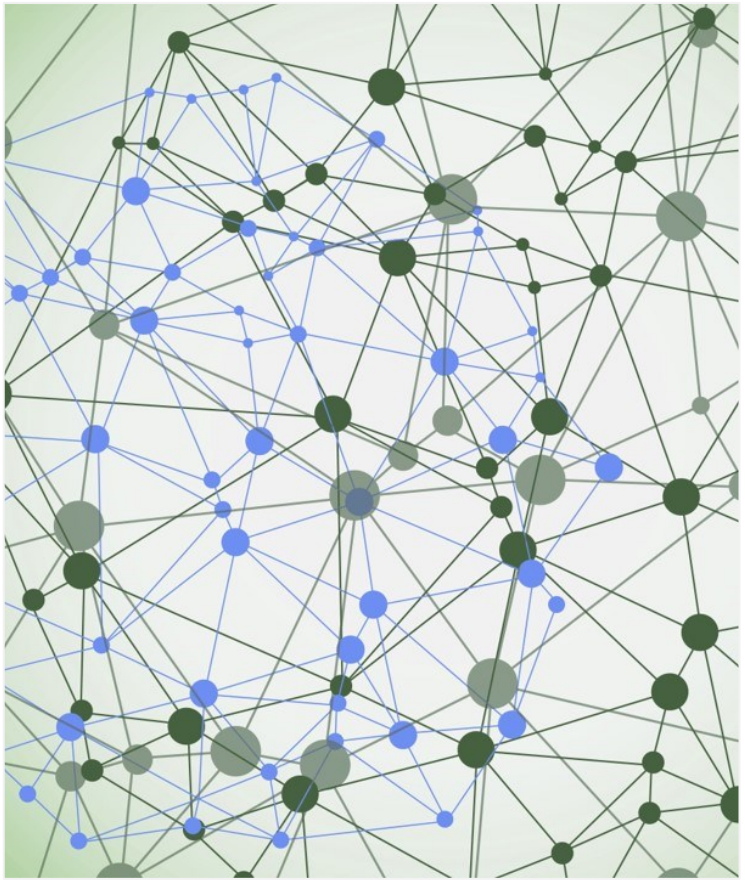
Primjer lokalnog/globalnog stanja



$$GS(t) = \{S_1^2, S_2^2, S_3^1, SC_{12}^x, SC_{23}^a, SC_{31}^b\}$$

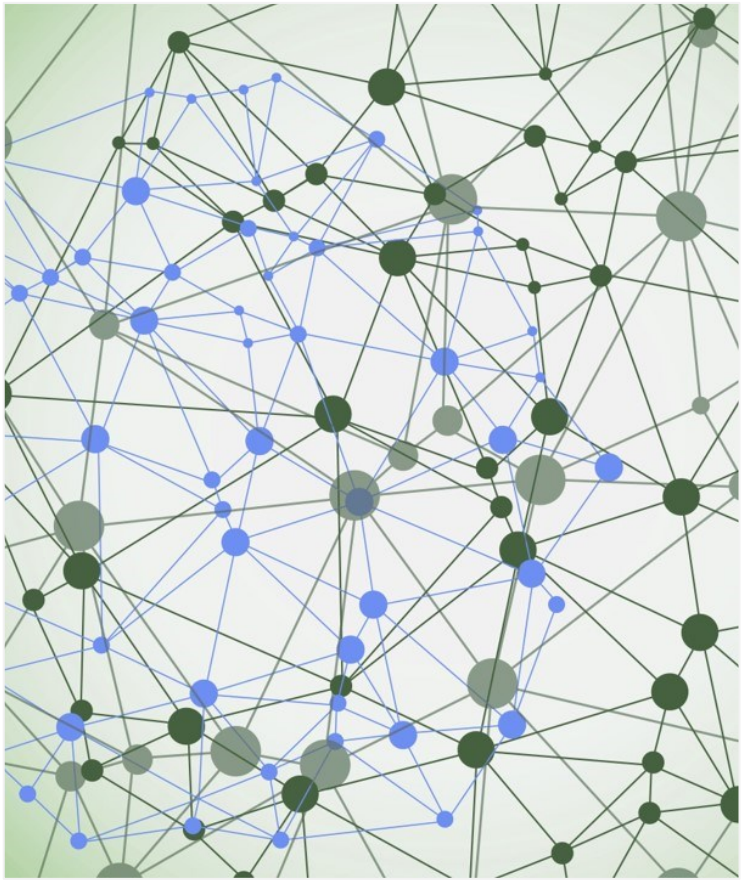
Raspodijeljeni (distribuirani) algoritam

- Algoritam koji se izvodi u raspodijeljenoj okolini na 2 ili više procesa koji komuniciraju razmjenom poruka
- Definira akcije koje izvodi svaki proces sustava, uključujući prijenos poruka između procesa. Poruke se prenose radi prijenosa informacija i omogućuju koordinaciju aktivnosti procesa koja vodi zajedničkom cilju – implementacija funkcionalnosti koju nudi raspodijeljeni algoritam.



Raspodijeljeni (distribuirani) algoritam

- izvodi se na većem broju računala povezanih mrežom
- nije jednostavno odrediti početak i kraj izvođenja algoritma, koliko procesa izvodi raspodijeljeni algoritam u svakom trenutku, globalno stanje sustava
- treba biti otporan na ispade procesa jer je to djelomični ispad sustava
- **komunikacija složenost**: brojimo poruke koje se generiraju tijekom izvođenja algoritma



Sadržaj predavanja

- Model raspodijeljenog sustava
  - model raspodijeljenog izvođenja
  - uzročna ovisnost događaja
  - globalno stanje raspodijeljenog sustava
  - raspodijeljeni algoritam
- Proširenje osnovnog modela raspodijeljenog sustava
  - sinkroni model
  - asinkroni model

Proširenje osnovnog modela (sinkroni i asinkroni)

Sinkroni model

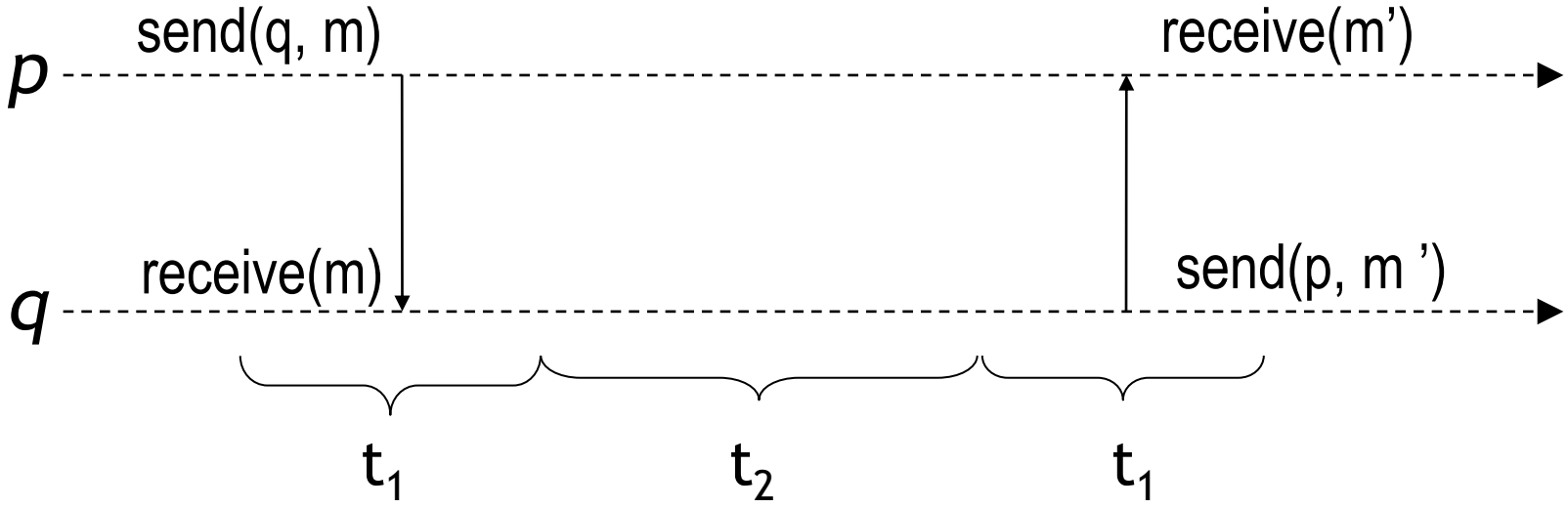
- pretpostavka: svi procesi raspodijeljenog sustava izvode događaje (tj. korake) istovremeno
- pojednostavljenje koje nije realno za raspodijeljene sustave, ali može biti korisno za njihovo razumijevanje i analizu

Asinkroni model

- pretpostavka: procesi izvode događaje u proizvoljnom slijedu
- postoji neodređenost vezana uz slijed događaja
- realna situacija

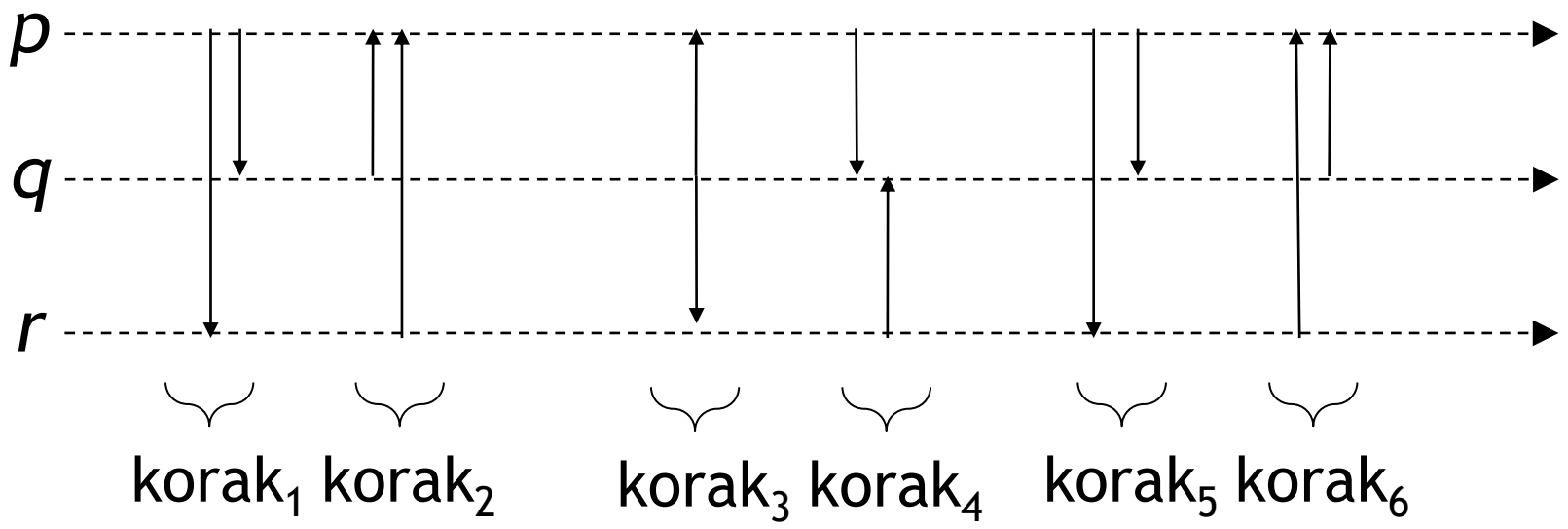
Sinkroni model

- Poznata je gornja vremenska granica za
  - trajanje prijenosa poruke kanalom (t_1) i izvođenje prijelaza nekog procesa (t_2)
- Pretpostavka
  - procesi imaju potpuno sinkronizirana lokalna vremena



Primjer sinkrone komunikacije

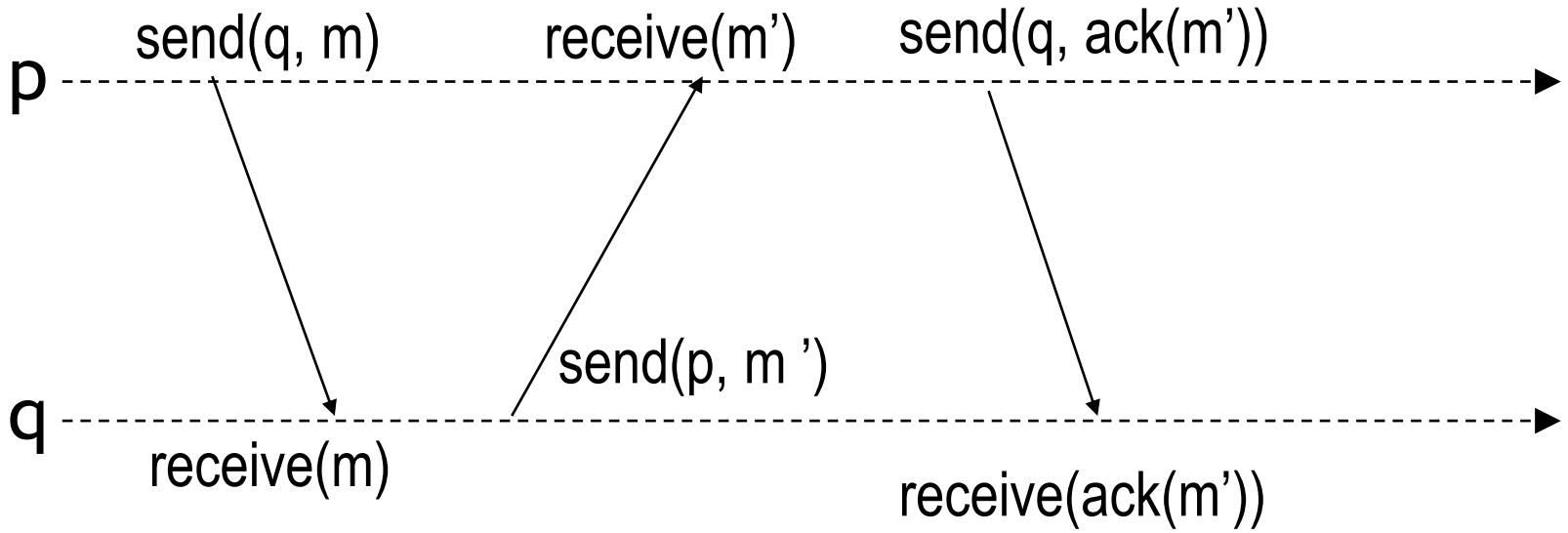
- izvođenje algoritma u sinkronom sustavu organizirano je u koracima
  - pošalji poruke procesima u sustavu
  - primi poruke od drugih procesa u sustavu
  - izvođenje prijelaza: promijeni stanje na temelju primljenih poruka



Asinkroni model

- Ne postoji gornja vremenska granica za
  - izvođenje prijelaza nekog procesa (no trajanje prijelaza je uvijek konačno)
  - trajanje prijenosa poruke kanalom
- Pretpostavka
  - procesi **nemaju** sinkronizirana lokalna vremena
- Realni slučaj koji ćemo najčešće razmatrati, znatno komplicira model i analizu distribuiranog algoritma raspodijeljenog sustava

Primjer asinkrone komunikacije

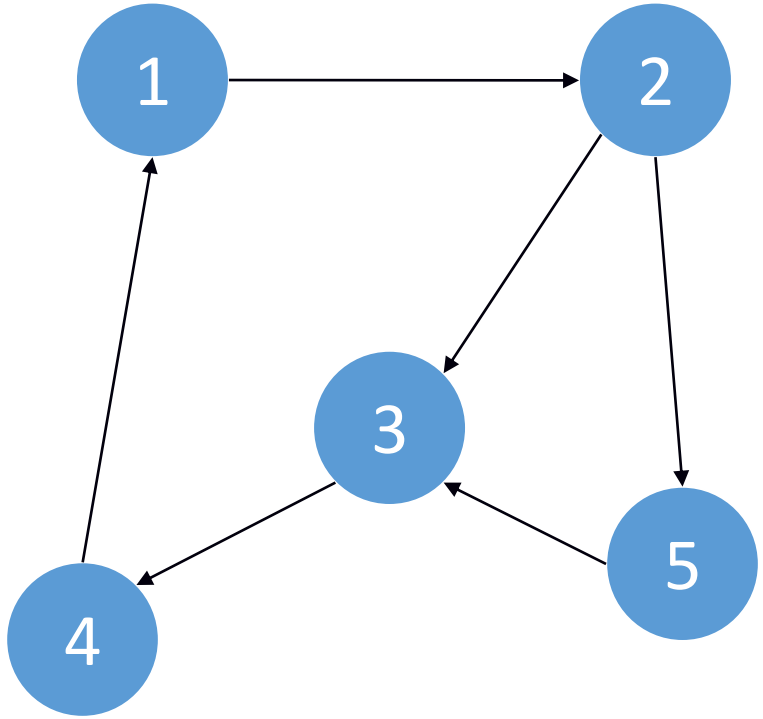


nepouzdani komunikacijski medij, potrebno je modelirati vjerojatnost gubitka poruke na kanalu

Sinkroni model raspodijeljenog sustava

Sinkroni model

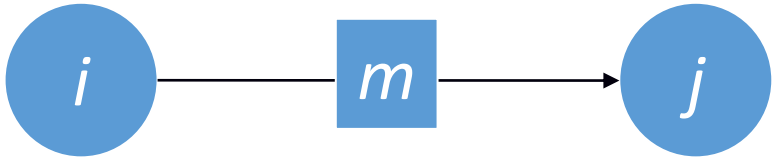
- usmjereni graf $G = (V, E)$
- $v_i \in V$, čvor modelira **proces**
- $e_j \in E$, grana modelira **kanal**
- ***M*** je skup poruka, *null* ako na kanalu nema poruka
- $out\text{-}nbrs_i$ – izlazni susjedi
- $in\text{-}nbrs_i$ – ulazni susjedi
- $distance(i, j)$ – najkraći put između i i j u G, $(i, j \in V)$
- $diameter(G)$ – max $distance(i, j)$ za sve parove (i, j)



Model procesa

- svaki se proces vezan uz čvor $v_i \in V$ modelira kao uređena četvorka: ($states_i$, $start_i$, $msgs_i$, $trans_i$)
- $states_i$ – skup mogućih stanja procesa
- $start_i$ – skup početnih stanja
  - $start_i \subset states_i$, $start_i \neq 0$
- $msgs_i$ – funkcija za generiranje poruka
  - određuje izlaznu poruku za svakog susjeda na temelju trenutnog stanja procesa
  - $states_i \times out\text{-}nbrs_i \rightarrow M_i \subset M \cup \{\text{null}\}$
- $trans_i$ – funkcija prijelaza, određuje sljedeće stanje na temelju trenutnog stanja i primljenih poruka od ulaznih susjeda

Model kanala

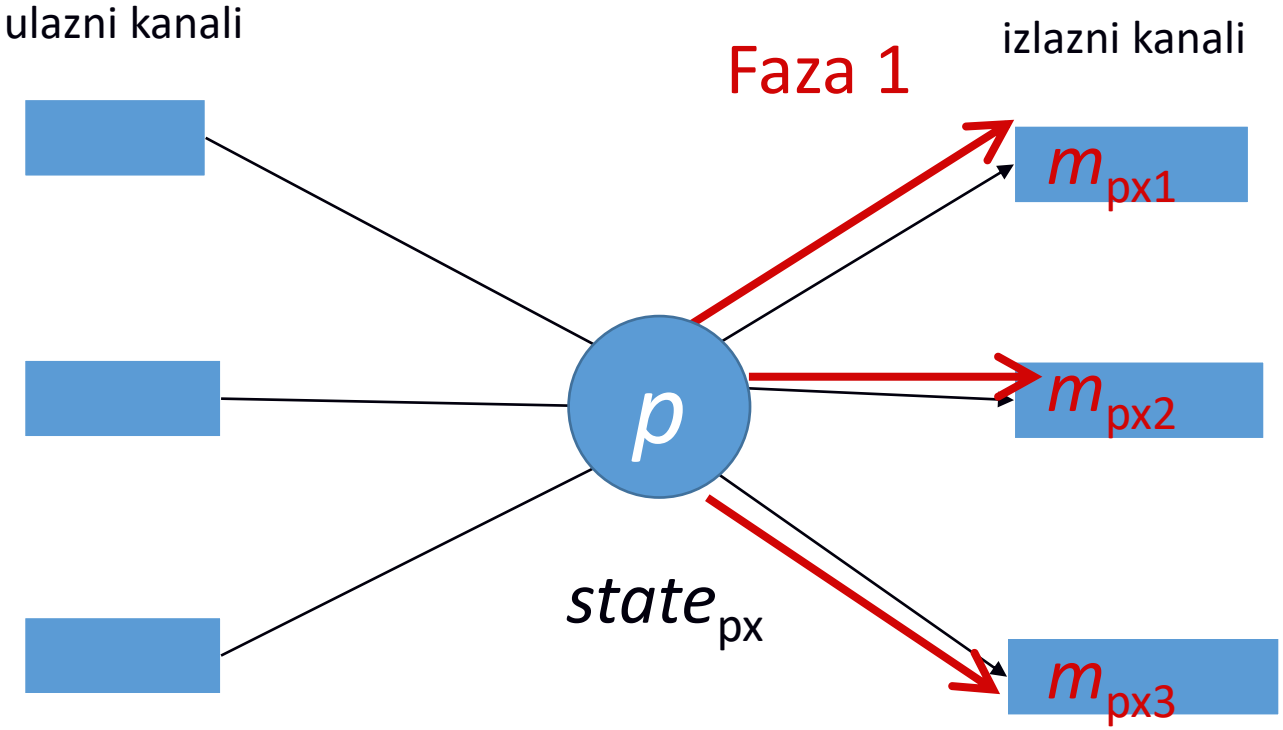


- modelira ga grana između para čvorova (*i*, *j*) iz *G*
- može primiti poruku *m* iz definiranog skupa poruka *M* ili *null*
- *null* označava praznu poruku

Izvođenje sinkronog modela

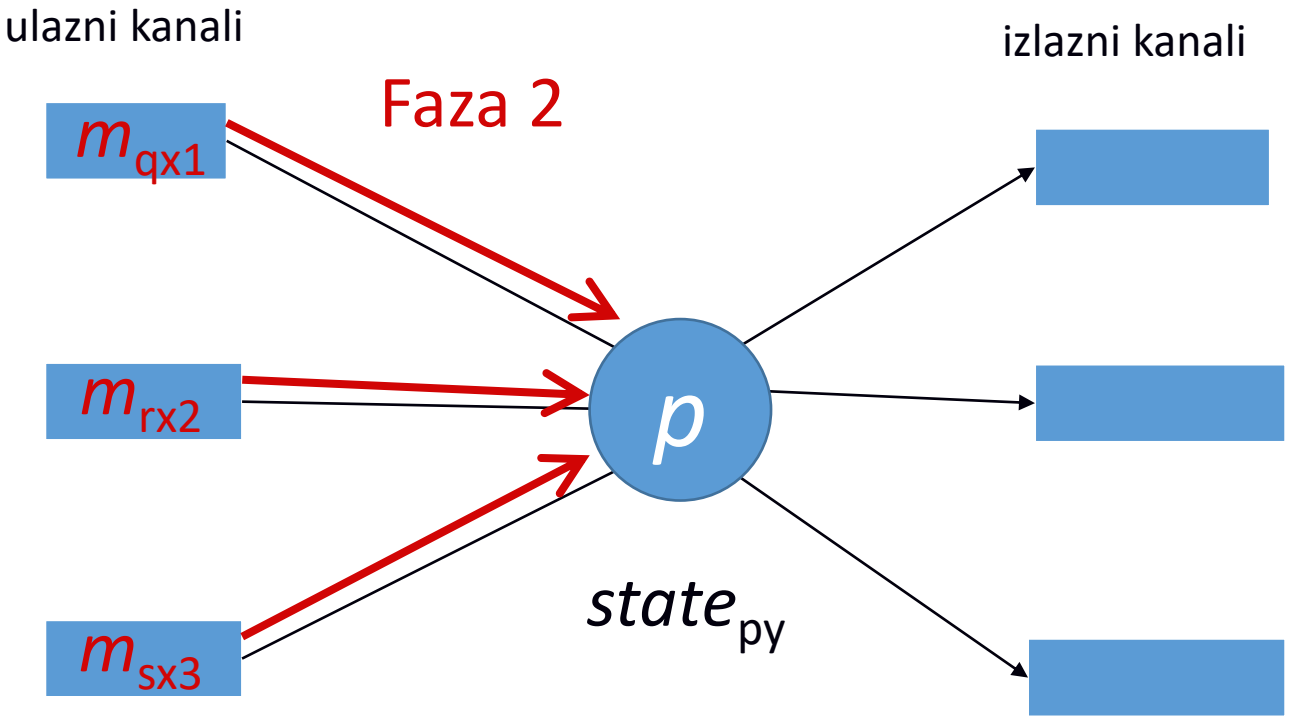
- Algoritmi za sinkrone modele se izvode u koracima (*round*). Inicijalno su svi procesi u proizvoljnom početnom stanju i svi su kanali prazni. Nakon toga se izvode koraci.
- Korak se sastoji od 2 faze
  - Faza 1: Za svaki proces primijeni funkciju za generiranje poruka (*msg*), a na temelju trenutnog stanja. Generiraj poruke koje će biti poslane izlaznim susjedima i postavi te poruke na izlazne kanale.
  - Faza 2: Primijeni funkciju prijelaza (*trans*) koja će na temelju trenutnog stanja i primljenih poruka odrediti sljedeće stanje procesa. Briši sve poruke na kanalima.

Izvođenje (faza 1)



na temelju trenutnog stanja generiraj poruke i postavi ih na izlazne kanale

Izvođenje (faza 2)



primijeni funkciju prijelaza koja na temelju primljenih poruka određuje sljedeće stanje procesa

Formalni model izvođenja

- Beskonačni slijed:

$$C_0, M_1, N_1, C_1, M_2, N_2, C_2, \ldots$$

- C_k - stanje svih procesa nakon k koraka
- M_k – poslane poruke na svim kanalima nakon k koraka
- N_k – primljene poruke na svim kanalima nakon k koraka
- $M_k \neq N_k$ – ako dođe do ispada na nekom kanalu, inače je u sustavima bez gubitaka poruka $M_k = N_k$ za svaki k

Složenost sinkronog algoritma

- vremenska složenost
  - mjeri se brojem izvedenih koraka (*rounds*, *r*) koji dovodi do završnog stanja algoritma tj. do stanja u kome su svi procesi zaustavljeni ili kada se više ne proizvode novi izlazi
- komunikacijska složenost
  - mjeri se broj kreiranih i poslanih poruka na kanalima

(Pri određivanju mjere složenosti uvijek se analizira najgori mogući scenarij izvođenja algoritma!)

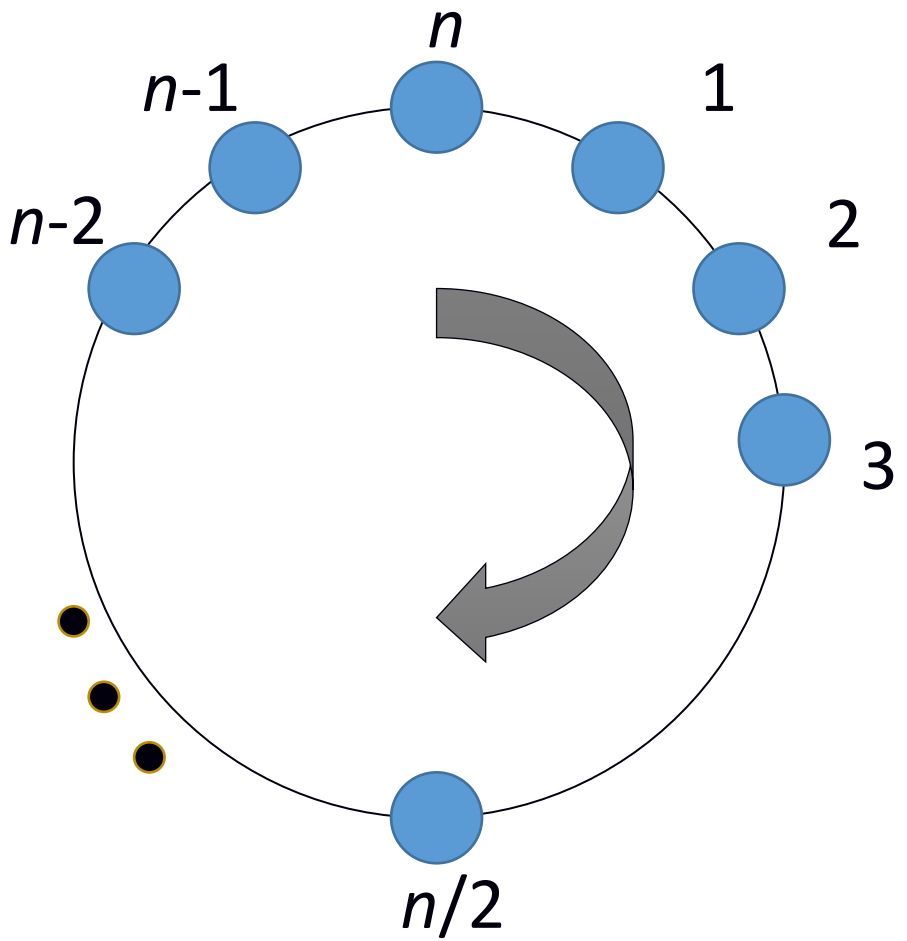
(Usporedite s mjerama složenosti algoritama, vremenska i prostorna)

Primjeri sinkronog modela

Problem 1: Odabir vođe u sinkronom prstenu

Definicija problema

- Izabrati jedinstvenog “vođu” među procesima u mreži
- U bilo kojem koraku samo 1 proces može postati vođa i promijeniti status u *leader*
- Pretpostavka jednostavne mreže od *n* čvorova
  - *token ring*
  - svaki čvor je označen brojem od 1 do *n*



Dodatni zahtjevi

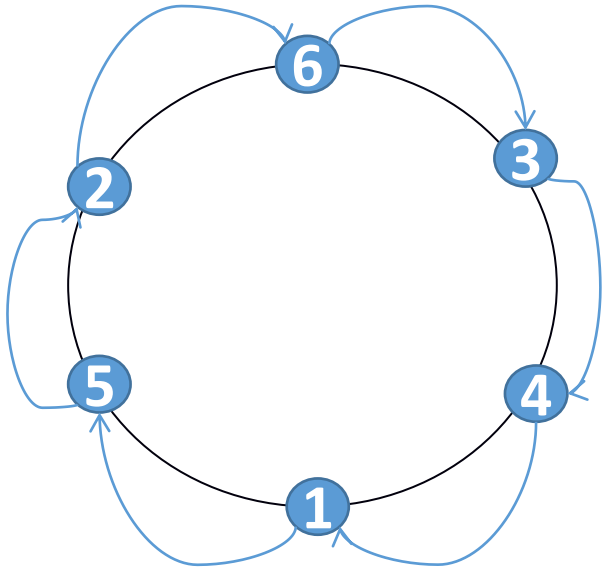
- Svi procesi su identični osim po identifikatoru
  - svaki ima jedinstven identifikator (UID - *Unique* ID)
  - UID nisu sljedbenici u prstenu
  - UID se mogu međusobno uspoređivati
  - (to omogućuje odabir vođe, inače bi svi procesi bili jednaki i vođa se ne bi mogao odabrati)
- Procesi znaju svoje susjede (ulazni ili izlazni)
- Broj procesa u prstenu (*n*) može biti poznat ili nepoznat svim procesima

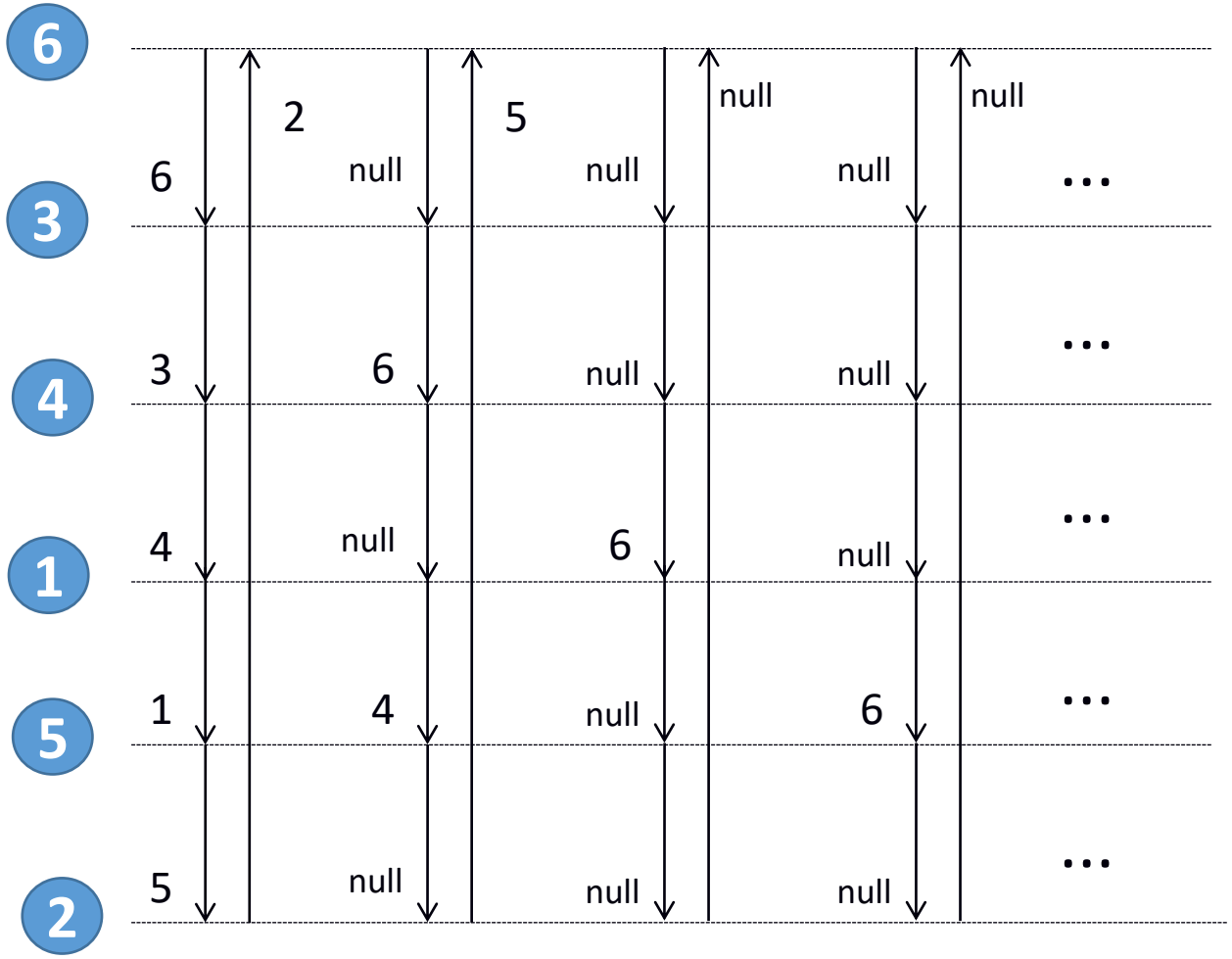
Osnovni algoritam za odabir vođe

- Pretpostavke
  - Jednosmjerna komunikacija među procesima u prstenu (usmjereni graf u smjeru kazaljke na satu)
  - Procesi ne znaju veličinu prstena *n*
  - Svaki proces ima jedinstveni identifikator UID iz skupa prirodnih brojeva, UID se procesu dodjeljuje na slučajan način
- Vođa je proces s najvećim UID
- Skica algoritma:

  Svaki proces inicijalno šalje svoj UID susjedu. Kada proces primi UID, ako je taj veći od njegovog UID-a prosljeđuje ga dalje, ako je primljeni UID malji od njegovog UID-a primljeni UID se odbacuje, a ako je primljeni UID jednak njegovom UID-u proces objavljuje sebe kao vođu

Primjer prstena i algoritma za odabir vođe





Formalni model osnovnog algoritma

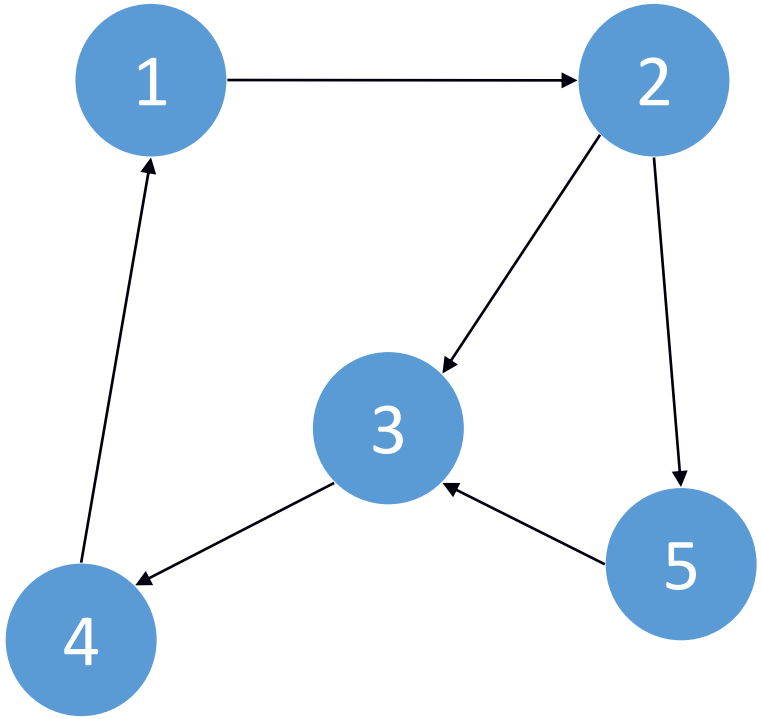
- *M* – skup poruka čini skup svih UID
- Za svaki proces *i*:
  - $states_i$ = (*u*, *send*, *status*)
    u – identifikator, inicijalno UID za *i*
    send – identifikator ili null, inicijalno UID za *i*
    status ∈ {*unknown*, *leader*}, inicijalno *unknown*
  - $start_i$ = (UID procesa *i*, UID procesa *i, unknown*)
  - $msgs_i$ – poslati vrijednost varijable *send* sljedećem procesu
  - $trans_i$ – receive *v*
    send := null (pobriši poruke na kanalima)
    if *v* > *u* then *send* := *v*
    if *v* = *u* then *status* := *leader*
    if *v* < *u* then *do nothing*

Složenost algoritma

- Vremenska
  - s obzirom da algoritam završava u slučaju kada čvor s najvećim UID ponovo primi vlastitu poruku, potrebno je *n* koraka da ta poruka stigne do vođe u prstenu s *n* čvorova
  - samo će proces koji je vođa znati da je algoritam završen (primio je poruku identičnu vlastitom UID), stoga može poslati posebnu poruku (*halt*) s obavijesti da je vođa izabran – potrebno je 2*n* koraka za pronalazak vođe i slanje poruka zaustavljanja
- Komunikacijska
  - u mreži se generira $O(n^2)$ poruka - pri svakom koraku svaki čvor potencijalno generira novu poruku ($n^2 = n$ poruka po koraku x n koraka do završetka algoritma)
  - Ako analiziramo max broj generiranih poruku uzimajući u obzir *null* poruke, onda je to za mrežu s padajućim UID u smjeru kazaljke na satu kada se za svaki korak generira $n+(n-1)+(n-2)+\ldots+1$, što je ukupno $n*(n+1)/2$ poruka, što je $O(n^2)$

Problem 2: Odabir vođe u usmjerenoj mreži

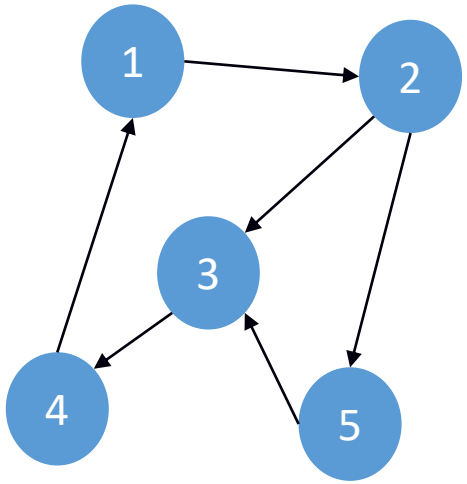
- Povezana usmjerena mreža, za svaki par čvorova postoji konačan *distance*(i, j)
- Svaki čvor ima jedinstveni identifikator UID
- Izabrati vođu među procesima u mreži
- Samo 1 proces mijenja status u *leader*
- *diameter*(G) = max *distance*(v_i, v_j) za sve parove (v_i, v_j) iz G



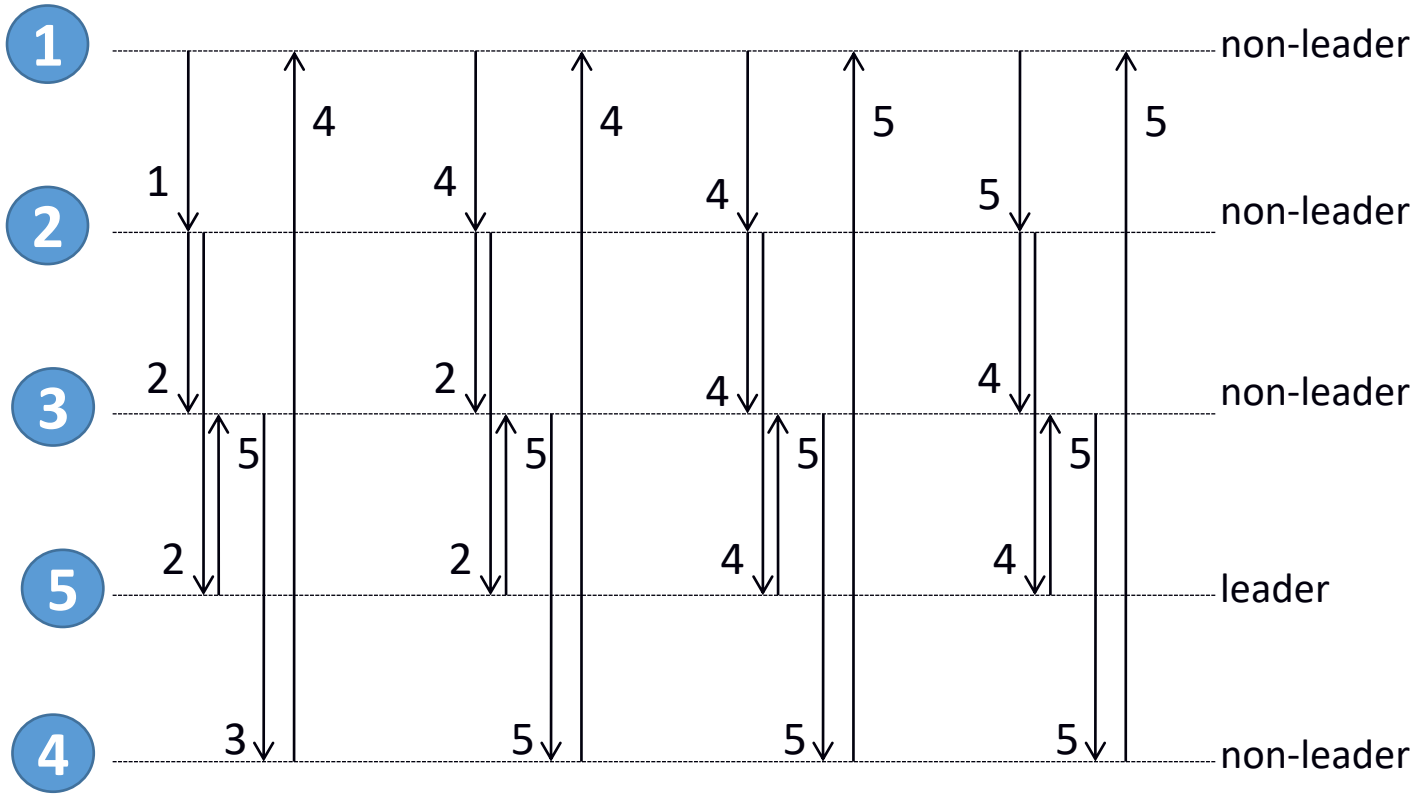
Rješenje problema 2: Algoritam preplavljivanja

- *Simple Flooding Algorithm* (SFA)
- Pretpostavke
  - Svaki proces ima UID iz skupa prirodnih brojeva
  - Svaki proces zna *diameter*(*G*) – (pažnja: ovo je globalno znanje!)
- Skica algoritma
  - Svaki proces bilježi max primljeni UID (inicijalno je to vlastiti UID). U svakom koraku proces šalje tu maksimalnu vrijednost na izlaznim kanalima svim susjedima. Nakon *diameter*(*G*) koraka ako je maksimalna vrijednost jednaka vlastitom UID, proces se proglašava vođom, a u suprotnom nije vođa.

Primjer algoritma za odabir vođe u usmjerenoj mreži



diameter = 4 jer je
distance (5,2) = 4



Formalni model algoritma preplavljivanja

- $states_i$ = (*u*, *max-uid*, *status*, *rounds*)
  u – UID, inicijalno UID za *i*
  max-uid – UID, inicijalno UID za *i*
  status $\in$ {*unknown*, *leader*, *non-leader*}, inicijalno *unknown*
  rounds – cijeli broj, inicijalno 0
- $msgs_i$ – if *rounds* < *diameter* then
  send *max-uid* to all $j \in$ *out-nbrs*
- $trans_i$ – *rounds* := *rounds* + 1
  receive set of UIDs *U* from neighbors
  max-uid := max({max-uid} $\cup$ *u*)
  if *rounds* = *diameter* then
  if *max-uid* = *u* then *status* := *leader*
  else *status* := *non-leader*

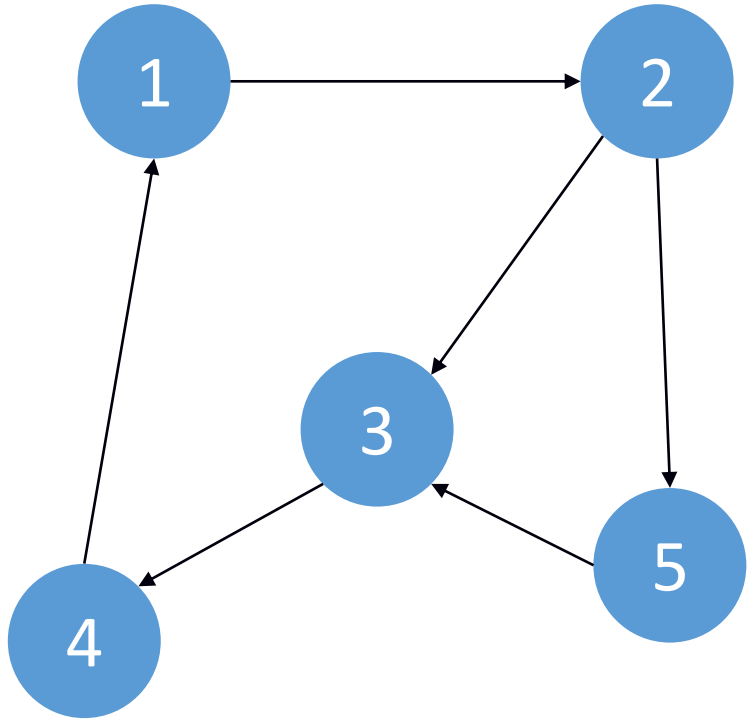
Složenost

- Vremenska
  - Određena vrijednošću *diameter*(*G*)
- Komunikacijska
  - broj poruka = diameter · |E|, gdje je |E| broj usmjerenih grana grafa
  - poruka se šalje na svaku granu za svaki korak algoritma
  - jednostavna optimizacija koja smanjuje broj poruka – proces šalje *max-uid* susjedima samo ako se vrijednost *max-uid* promijeni

Asinkroni model raspodijeljenog sustava

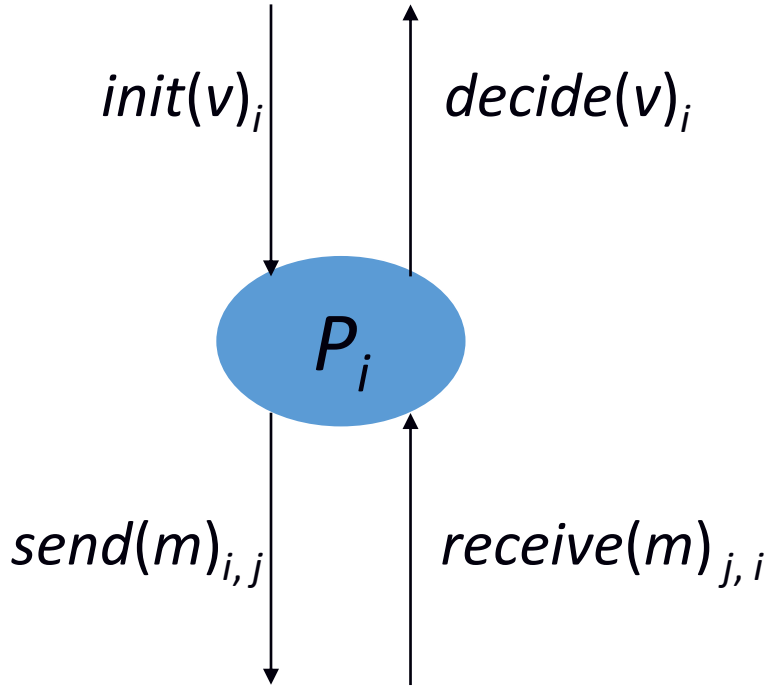
Asinkroni model

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



Ulazno/izlazni (I/O) automat

- formalni model za asinkrone sustave
- I/O automat modelira komponentu raspodijeljenog sustava koja je u interakciji s ostalim komponentama
- prijelazi su vezani uz **događaje**
- događaji mogu biti *ulazni*, *izlazni* ili *unutarnji*



primjer procesa u asinkronom raspodijeljenom sustavu

Formalna definicija I/O automata

I/O automat *A* se sastoji od sljedećih komponenti:

- *sig*(*A*) – signatura
  sig(A)={ *in*(*A*), *out*(*A*), *int*(*A*) } – opis ulaznih, izlaznih i unutarnjih događaja)
- *states*(*A*) – skup stanja automata
- *start*(*A*) – skup početnih stanja, *start*(*A*) $\neq 0$
- *trans*(*A*) – funkcija prijelaza,
  npr. (*s*, π, *s*') – *s* i *s*' su stanja, a π je događaj
  - za svako stanje *s* i svaki ulazni događaj π postoji prijelaz (*s*, π, *s'*) $\in$ *trans*(*A*)

Izvođenje automata

- automat *A* se izvodi kao konačan ili beskonačan slijed stanja i događaja, npr.

$$s_0, \pi_0, s_1, \pi_1, s_2, \pi_2, s_2, \ldots \pi_k, s_k, \ldots$$

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

Primjer: automat kanala FIFO (1)

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

Primjer: automat kanala FIFO (2)

primjeri izvođenja – tj. slijed događaja (engl. *trace*)

- [null], $send(1)_{i,j}$, [1], $receive(1)_{i,j}$, [null], $send(2)_{i,j}$, [2], $receive(2)_{i,j}$, [null]
- [null], $send(1)_{i,j}$, [1], $send(1)_{i,j}$, [11], $send(1)_{i,j}$, [111]...

Za "*trace*" su važna sljedeća 2 svojstva:

- *A **safety property** is often interpreted as saying that some particular "bad" thing never happens.*
- *A **liveness property** is often informally understood as saying that some particular "good" thing eventually happens.*

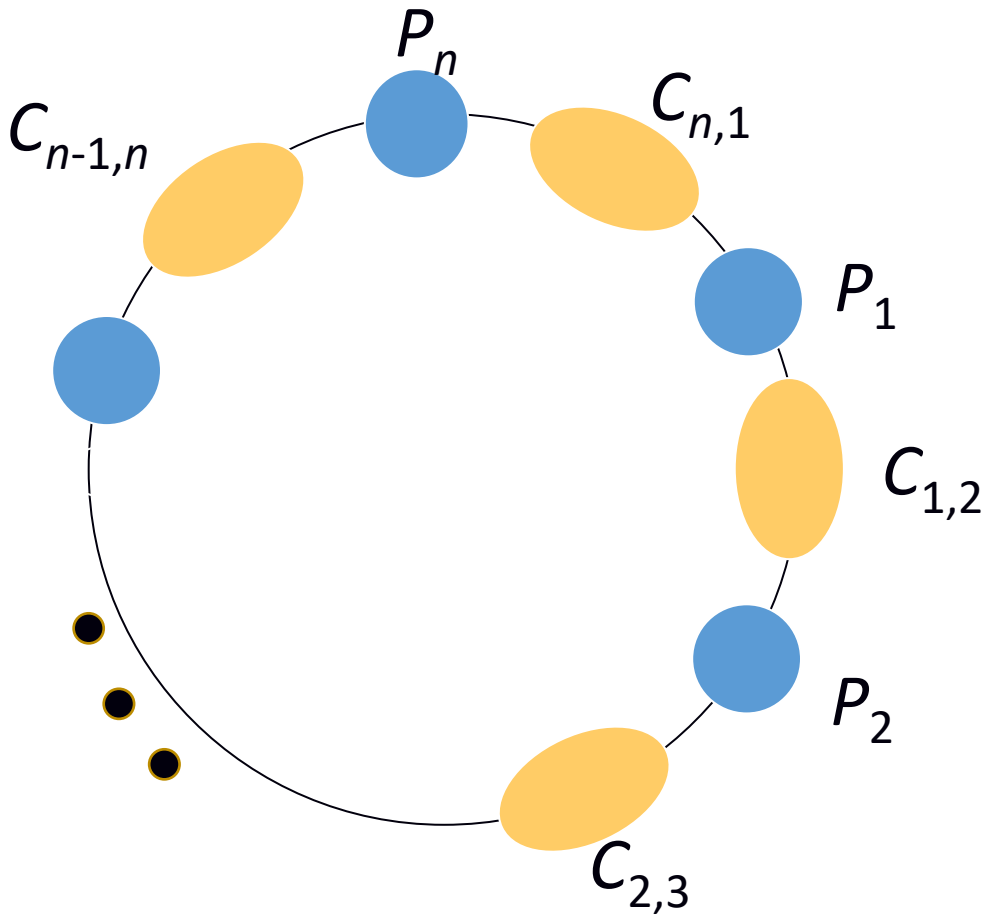
(prema N. Lynch)

Primjeri asinkronog modela

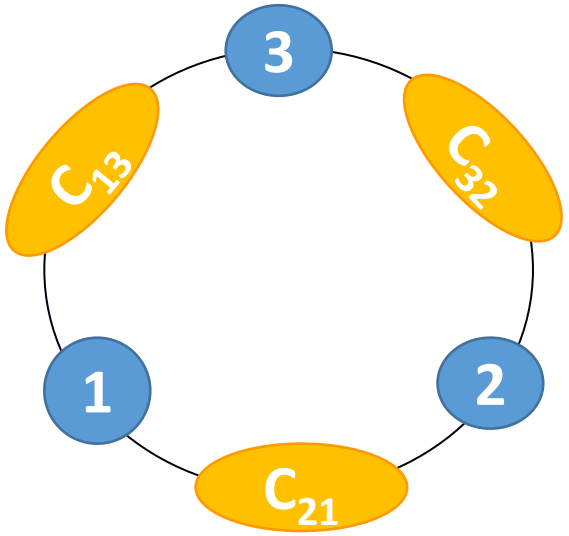
Odabir vođe u asinkronoj mreži

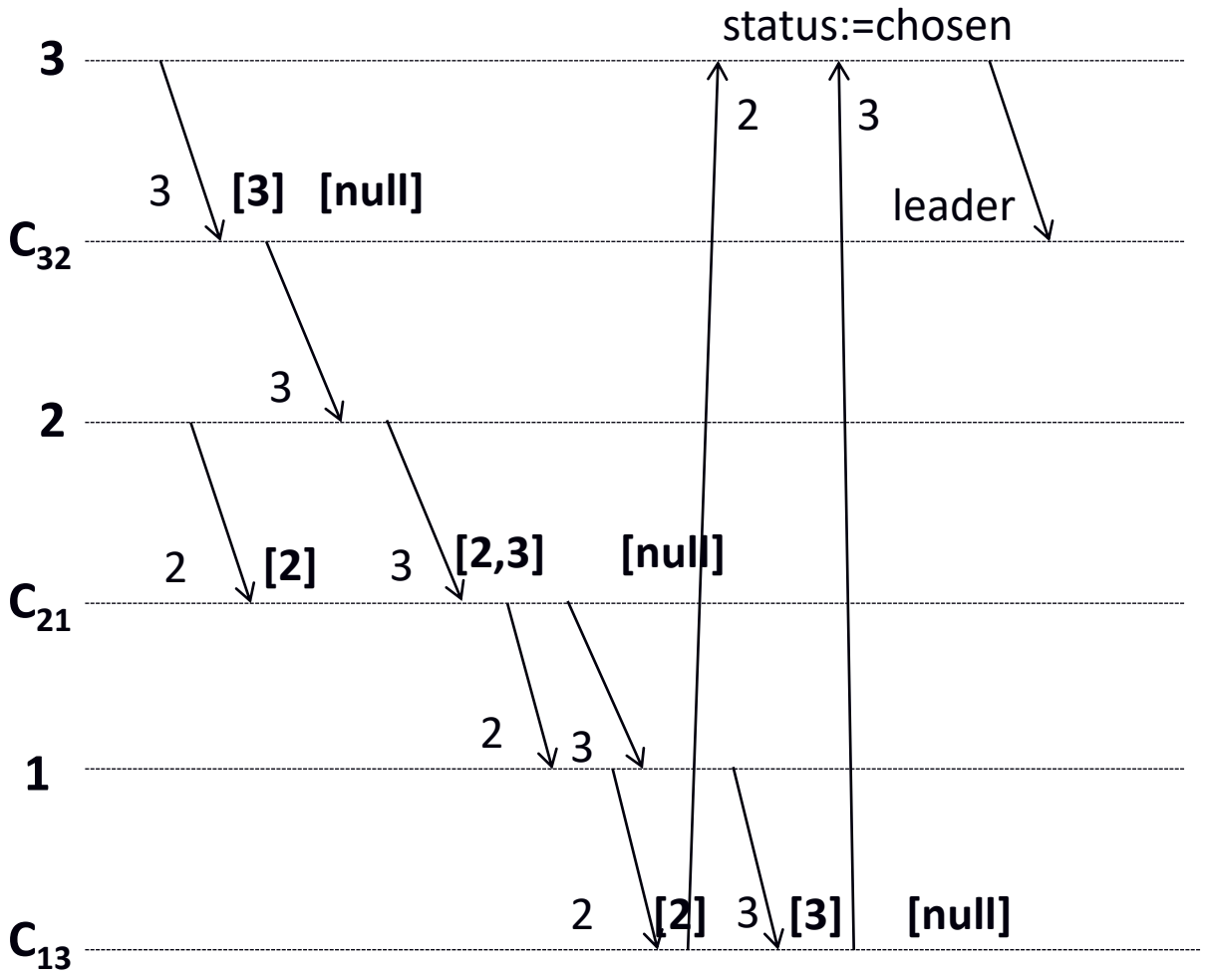
Definicija problema

- 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: pretpostavka je pouzdani FIFO



Primjer asinkronog prstena i algoritma za odabir vođe





Osnovni algoritam za asinkroni model

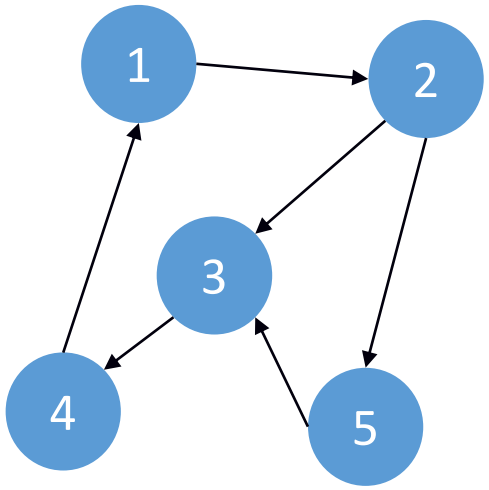
Definicija automata procesa P_i

- 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*

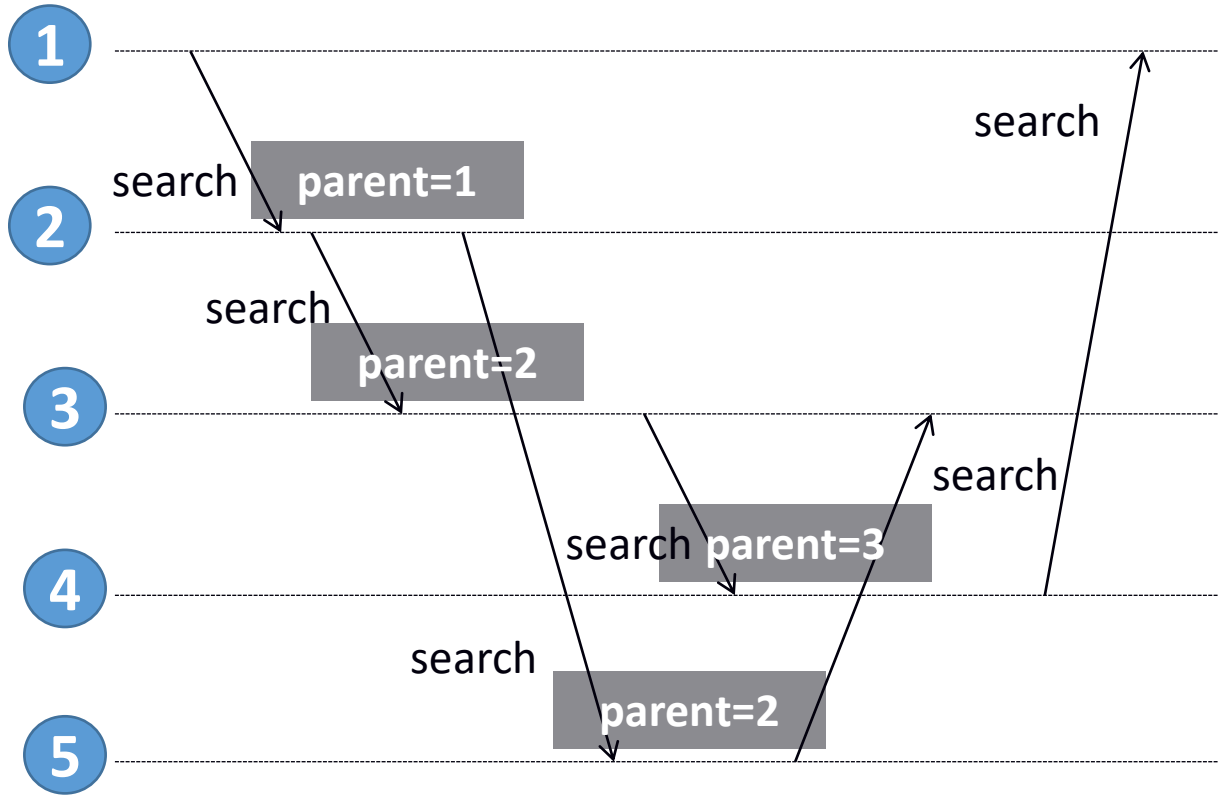
Kreiranje stabla u asinkronoj mreži

- kreiranje stabla s definiranim korijenskim čvorom i_0
- mreža je modelirana grafom $G(V, E)$ koji je usmjeren i povezan
- procesi ne znaju dijametar mreže
- cilj: svaki proces treba odrediti prethodnika (*parent*)
- Skica algoritma *AsynchSpanningTree*
  - 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.

Primjer algoritma *AsynchSpanningTree*



$i_0 = 1$



AsynchSpanningTree

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* \ {*j*}
    send(k) := *search*

Literatura

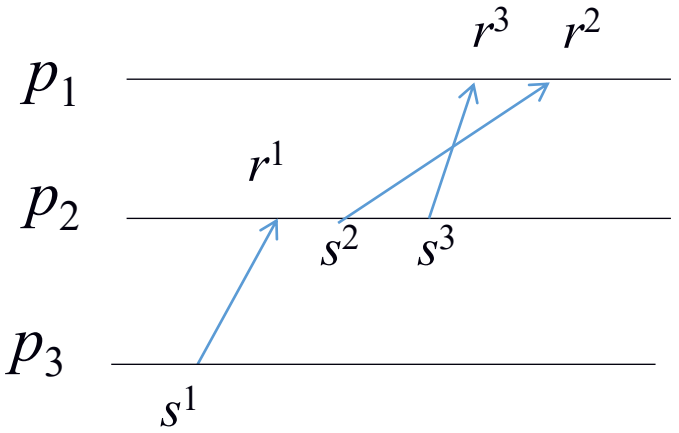
- A. D. Kshemkalyani, M. Singhal: Distributed Computing: Principles, Algorithms, and Systems, *Cambridge University Press*, 2008. poglavlja 2.1-2.4
- N. Lynch: Distributed Algorithms, *Morgan Kaufmann Publishers Inc.* 1996.
  - poglavlje 2: Modelling I: Synchronous Network Model
  - poglavlje 3: Leader Election in a Synchronous Ring (osnovni algoritam do 3.4)
  - poglavlje 8: Modelling II: Asynchronous System Model
  - poglavlje 15.1: Leader Election in a Ring

Pitanja za učenje i ponavljanje

1. Za koje je svojstvo raspodijeljenih sustava značajna komunikacijska složenost algoritama? Zašto?

   a) replikacijska transparentnost
   b) skalabilnost
   c) otvorenost

2. Objasnite model komunikacijskog kanala koji se temelji na uzročnoj slijednosti. Vrijedi li za primjer na slici CO ili non-CO i zašto?



3. Proučite formalnu definiciju algoritma *AsynchSpanningTree* na slajdu 60.