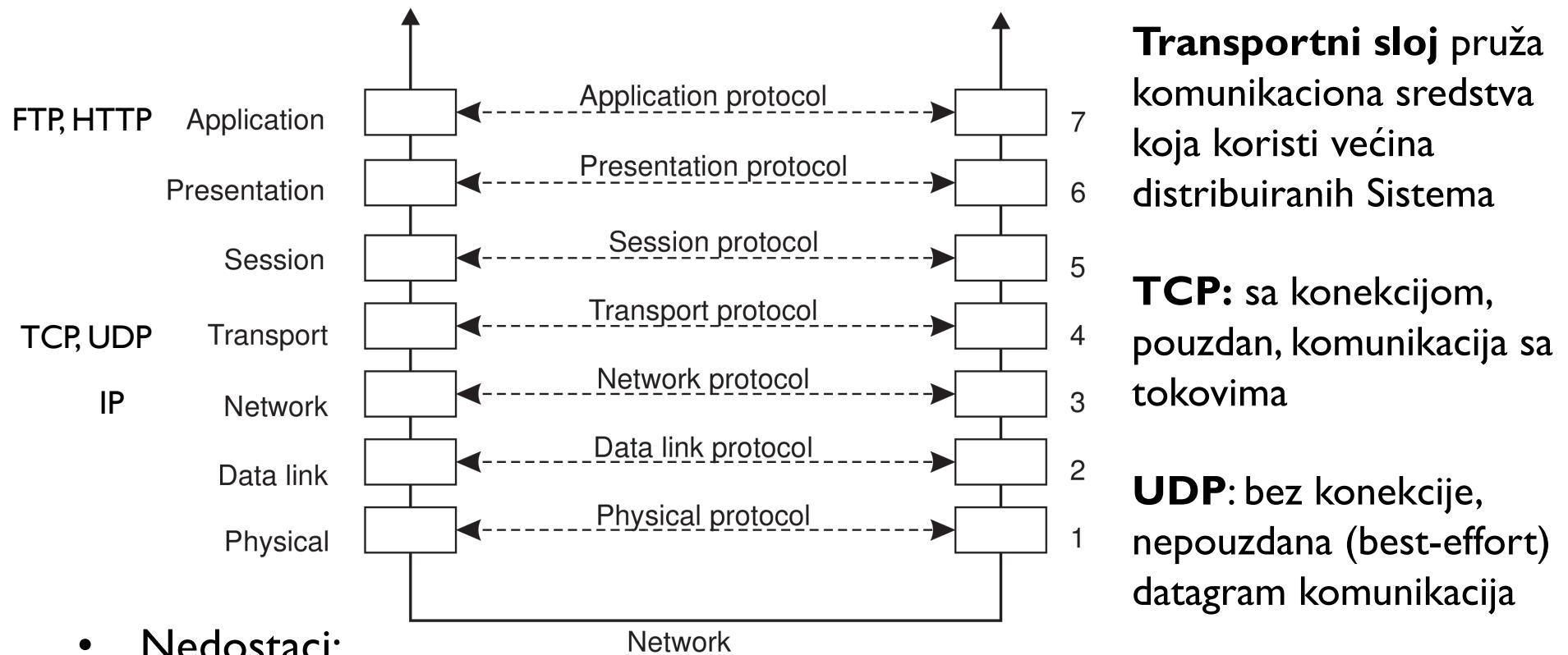


Komunikacija

Komunikacija

- **Međuprocesna komunikacija** (engl. **interprocess communication – IPC**) predstavlja **srce svih distribuiranih sistema**
- Savremeni distribuirani sistemi mogu biti sastavljeni od hiljada ili miliona procesa, raštrkanih širom mreže sa nepouzdanom komunikacijom kao što je Internet
- Skup pravila koja moraju poštovati procesi kako bi komunicirali predstavlja **komunikacioni protokol**, tipično **slojevito organizovan** (ISO OSI referentni model)
- Dva modela komunikacije:
 - **poziv udaljenih procedura** (engl. *Remote Procedure Call* – RPC)
 - **komunikacija orijentisana ka porukama** (soketi, MPI, MOM)

OSI referentni model



Transportni sloj pruža komunikaciona sredstva koja koristi većina distribuiranih Sistema

TCP: sa konekcijom, pouzdan, komunikacija sa tokovima

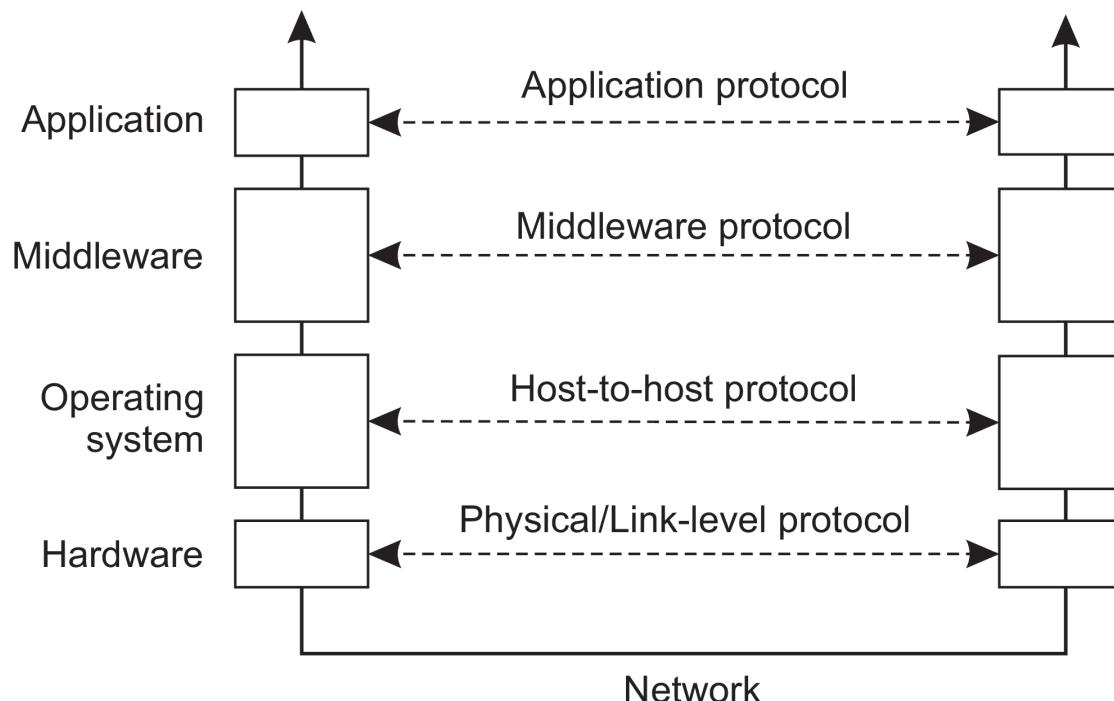
UDP: bez konekcije, nepouzdana (best-effort) datagram komunikacija

- **Nedostaci:**
 - fokus isključivo na slanju poruka
 - pruža često nepotrebne ili neželjene funkcionalnosti
 - krši transparentnost pristupa

Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Adaptirana slojevita šema

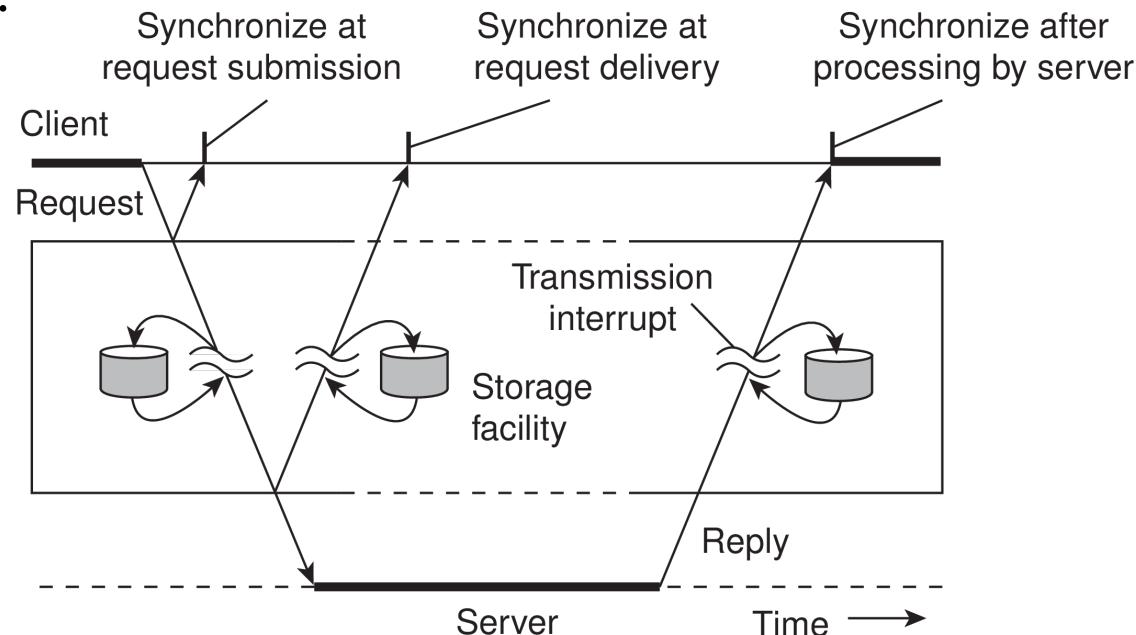
- **Midiver** je uveden u cilju pružanja **zajedničkih servisa i protokola različitim aplikacijama**
 - sadrži bogat skup **komunikacionih protokola**, (an)maršaling podataka, **protokoli za imenovanje** kako bi se lakše delili resursi, **bezbednosni protokoli**, mehanizmi za **skaliranje** kao što su **replikacija i keširanje**



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Tipovi komunikacije

- Vrste komunikacije:
 - **prolazna** (engl. *transient* – server odbacuje poruke ako ih ne može proslediti) i **perzistentna** (engl. *persistent* – server čuva poruku dok god je ne isporuči), **sinhrona** i **asinhrona**
- **Midiver kao posredni (distribuirani) servis u komunikaciji na nivou aplikacija:**



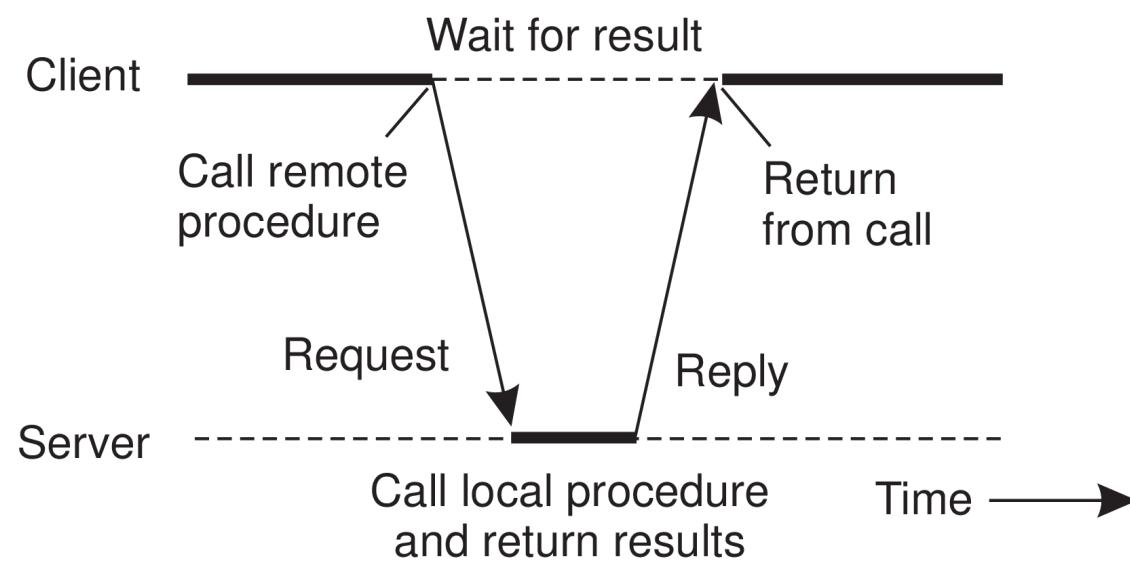
Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Tipovi komunikacije

- **Klijent/server** model se bazira na **prolaznoj sinhronoj komunikaciji**:
 - klijent i server moraju biti aktivni u vreme komunikacije
 - klijent šalje zahteve i blokira se dok ne dobije odgovor
 - server čeka na dolazeće zahteve i obrađuje ih
 - **mane sinhrone komunikacije**: klijent je blokiran dok čeka na odgovor, otkazi se moraju odmah rešavati jer klijent čeka, sam model nekad ne odgovara (mejl, vesti)
- **Midver orientisan ka porukama (MOM)** se bazira na **perzistentnoj asinhronoj komunikaciji**:
 - procesi šalju poruke koje se smeštaju u red, pošiljalac ne mora da čeka na odgovor, midver osigurava otpornost na greške

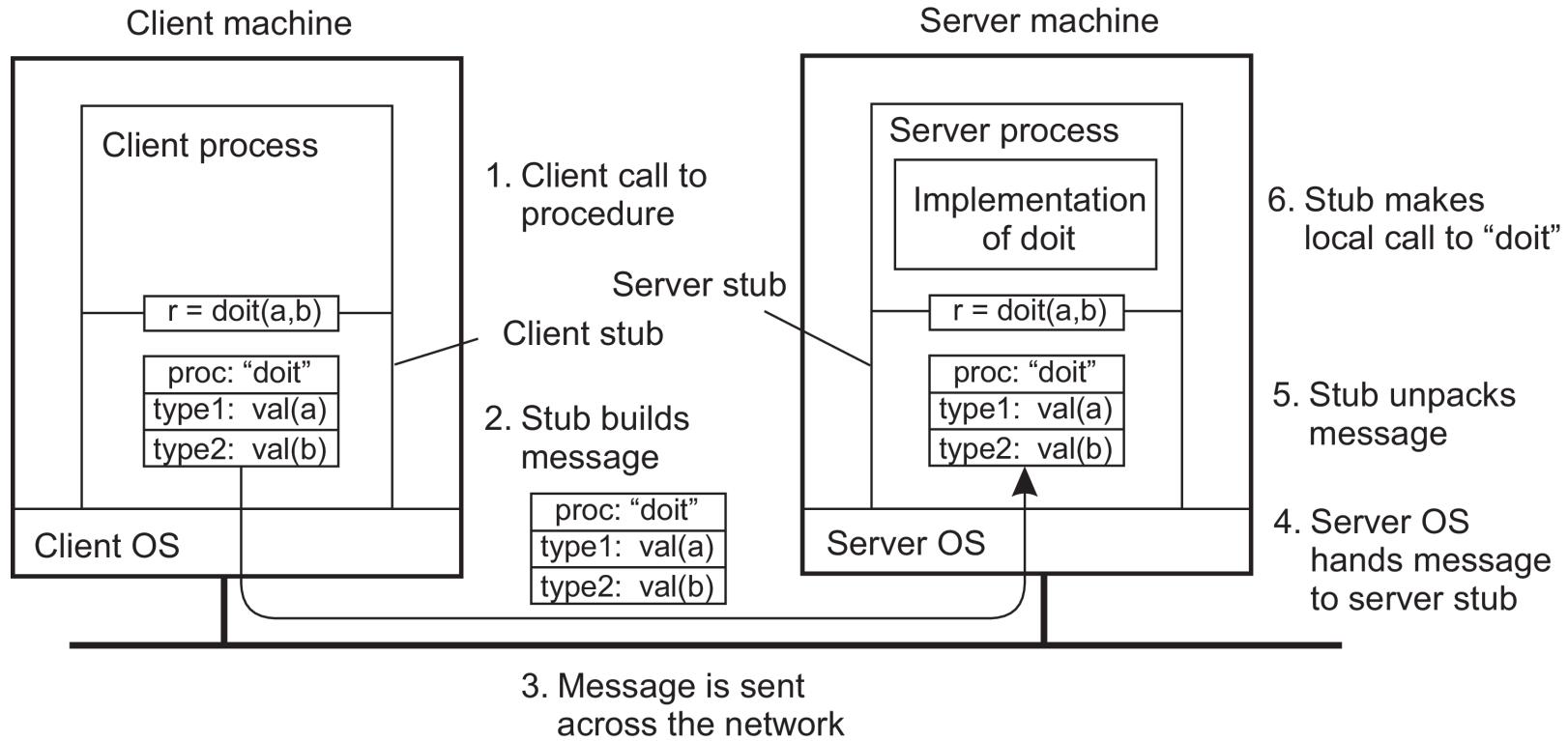
RPC

- Programeri su dobro upoznati sa jednostavnim modelom poziva procedura
- Dobro napravljene procedure rade izolovano (princip crne kutije) – nema razloga da se ne mogu izvršavati na posebnim mašinama
- Komunikacija između pozivaoca i pozvanog može se sakriti mehanizmom poziva procedura



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Osnovne RPC operacije



- 1. klijentska procedura poziva klijentski isečak
- 2. isečak pravi poruku i poziva lokalni OS
- 3. lokalni OS šalje poruku udaljenom OS-u
- 4. udaljeni OS prosleđuje poruku isečku
- 5. isečak raspakuje parametre i poziva server
- 6. server pravi lokalni poziv i vraća rezultat isečku
- 7. isečak pravi poruku i poziva lokalni OS
- 8. OS servera šalje poruku OS-u klijenta
- 9. OS klijenta prosleđuje poruku isečku
- 10. isečak raspakuje rezultat i vraća ga klijentu

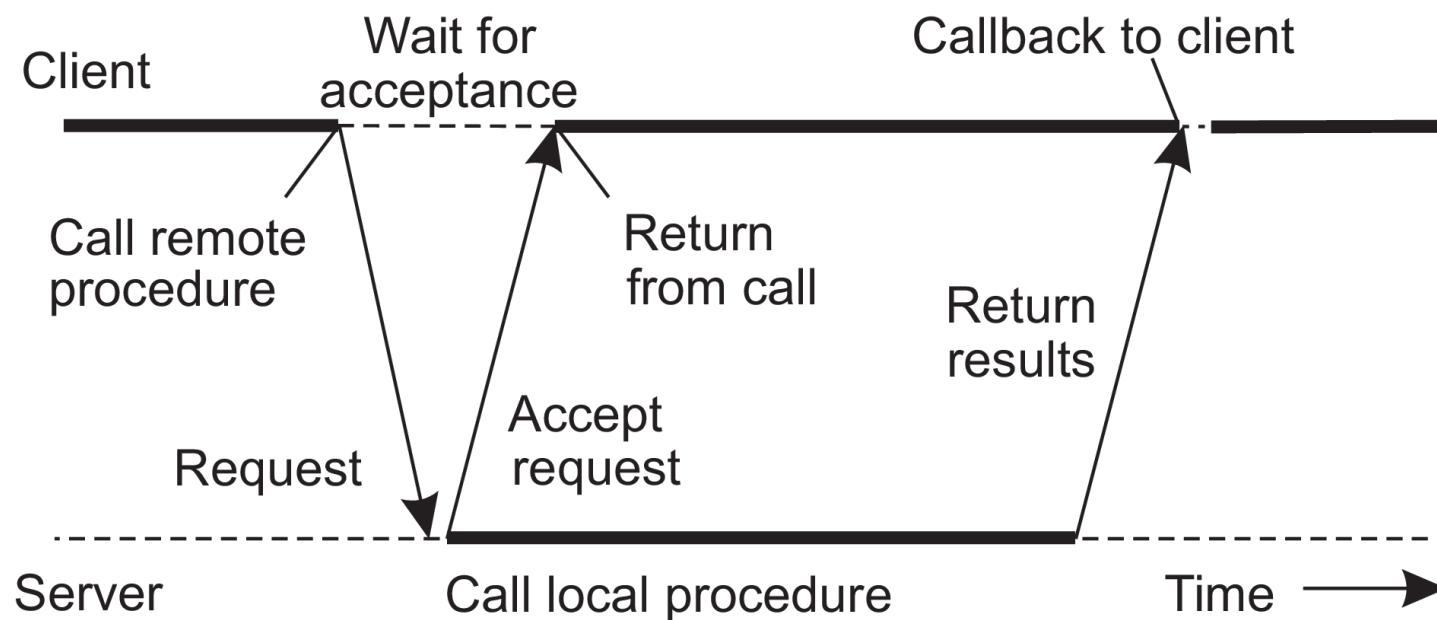
Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

RPC: prosleđivanje parametara

- Nije dovoljno samo spakovati parametre u poruke:
 - klijentska i serverska mašina mogu imati **različite reprezentacije podataka** (npr. različito uređenje bajtova – Intel little endian, ARM big endian)
 - pakovanje parametara podrazumeva **transformaciju vrednosti u sekvencu bajtova**
 - klijent i server se moraju **složiti oko određenog kodiranja**
 - kako se predstavljaju **osnovni tipovi podataka** (integer, float, character)
 - kako se predstavljaju **složeni tipovi podataka** (nizovi, strukture, unije)
- Neophodno je da **klijent i server ispravno interpretiraju poruke**, transformišući ih u odgovarajuće mašinski zavisne reprezentacije

Asinhroni RPC

- Izbegava se striktno ponašanje tipa zahtev-odgovor, klijent može da nastavi sa radom bez čekanja na odgovor od strane servera



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

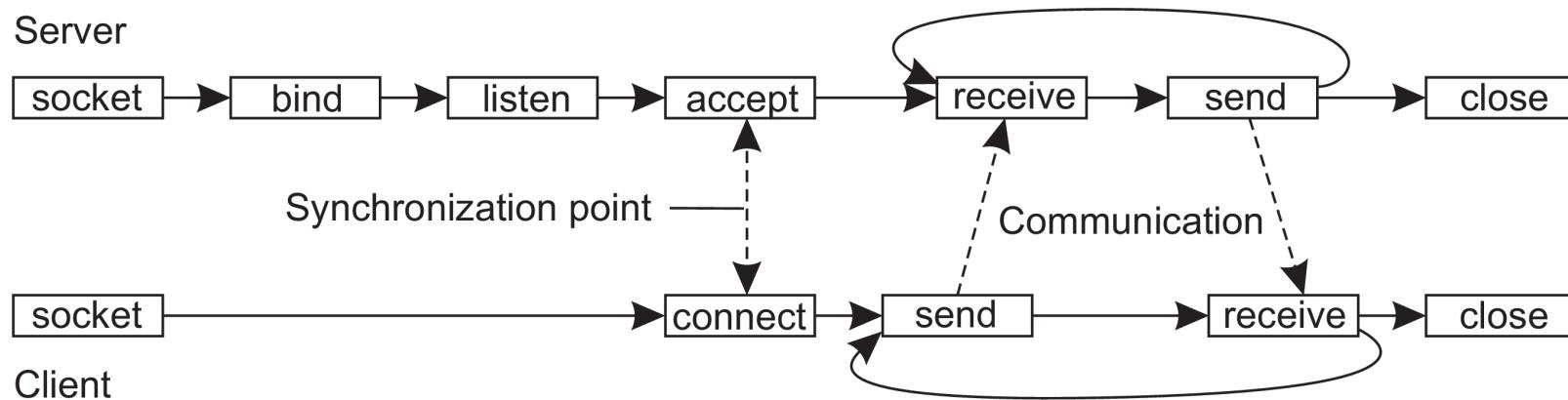
Komunikacija orijentisana ka porukama

- **Soket** (engl. *socket*) interfejs je uveden 1970-tih u Berkley UNIX-u, i kasnije je usvojen kao POSIX standard, podržavaju prolaznu (engl. *transient*) komunikaciju
- **Soket je krajnja tačka** (engl. *end point*) **u komunikaciji** na kojoj aplikacije mogu upisivati podatke za slanje putem mreže i pre kojih mogu primati dolazeće podatke,
- **Soket je apstrakcija stvarnog porta** kojeg koristi operativni sistem za određeni transportni protokol
- Kod TCP/IP, soketu se pridružuje **adresa soketa** koja se sastoji od para (*IP adresa, broj porta*) na lokalnom čvoru, neophodna je i adresa soketa i na drugom čvoru

```
Socket socket = getSocket(type = "TCP")
connect(socket, address = "1.2.3.4", port = "80")
send(socket, "Hello, world!")
close(socket)
```

Soketi

Soket operacija za TCP/IP	Opis
socket	kreiraj novu komunikacionu krajnju tačku
bind	pridruži lokalnu adresu soketu
listen	reci OS koji je max dozvoljeni broj čekajućih zahteva za konekciju
accept	blokiraj pozivaoca dok ne stigne zahtev za konekcijom
connect	aktivno pokušaj da uspostaviš konekciju
send	pošalji određene podatke preko konekcije
receive	primi određene podatke preko konekcije
close	prekini konekciju



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

MPI

- **Message Passing Interface (MPI)**, standard za slanje poruka, projektovan za paralelnu obradu i prolaznu komunikaciju, pruža fleksibilnost u HPC primenama

Operation	Description
<code>MPI_bsend</code>	Append outgoing message to a local send buffer
<code>MPI_send</code>	Send a message and wait until copied to local or remote buffer
<code>MPI_ssend</code>	Send a message and wait until transmission starts
<code>MPI_sendrecv</code>	Send a message and wait for reply
<code>MPI_isend</code>	Pass reference to outgoing message, and continue
<code>MPI_issend</code>	Pass reference to outgoing message, and wait until receipt starts
<code>MPI_recv</code>	Receive a message; block if there is none
<code>MPI_irecv</code>	Check if there is an incoming message, but do not block

Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

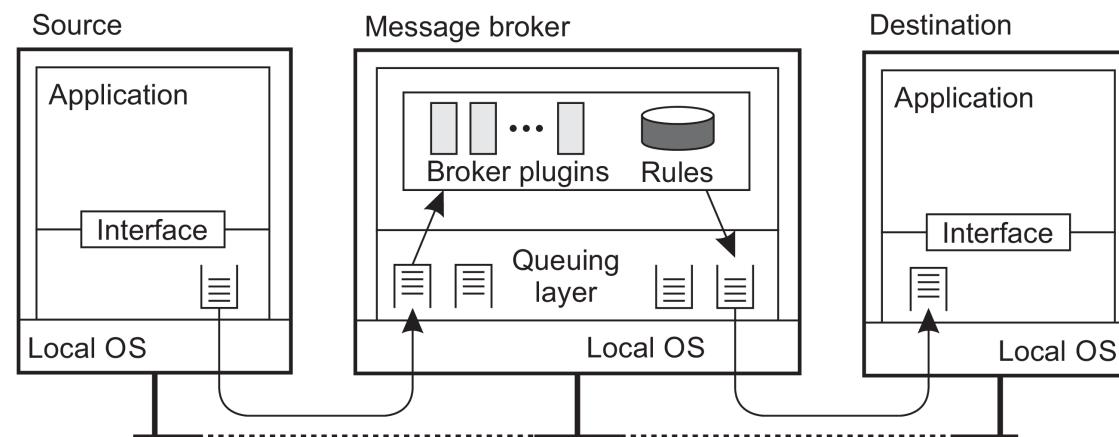
Midlver orientisan ka porukama (MOM)

- **MOM** omogućava **asinhronu perzistentnu komunikaciju** putem **redova** (engl. *queues*) na **nivou midlvera**, **redovi** odgovaraju **baferima na komunikacionim serverima**, za razliku od soketa i MPI, MOM sistemi za prenos poruka omogućavaju da transfer traje minutima
- **Osnovna ideja u sistemima sa redovima poruka** (engl. *message-queuing systems*) je da aplikacije komuniciraju ubacivanjem poruka u redove, redovima upravljuju **menadžeri redova** koji izvlače poruke iz lokalnih redova i rutiraju ih, osnovni interfejs za rad sa redom:

Operacija	Opis
put	dodaj poruku u određeni red
get	blokiraj dok specificirani red nije prazan, ukloni prvu poruku
poll	proveri da li specificirani red ima poruke i ukloni prvu, nikada ne blokira
notify	instaliraj upravljač koji se poziva kada se poruka smesti u specificirani red

Brokeri poruka

- **Sistemi sa redovima poruka** podrazumevaju **zajednički protokol za poruke**, sve aplikacije moraju se složiti oko formata poruke (strukture i reprezentacije podataka)
- **Broker poruka** (engl. *message broker*) transformiše dolazeće poruke tako da ih odredišne aplikacije mogu razumeti
 - ponaša se kao kapija (engl. *gateway*) na aplikacionom nivou
 - pruža rutiranje prema temama (engl. *subjects*) (tj. objavi-preplati se mogućnosti)
 - omogućava naprednu integraciju poslovnih aplikacija (EAI), odgovoran i za uparivanje aplikacija na osnovu poruka koje se razmenjuju



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

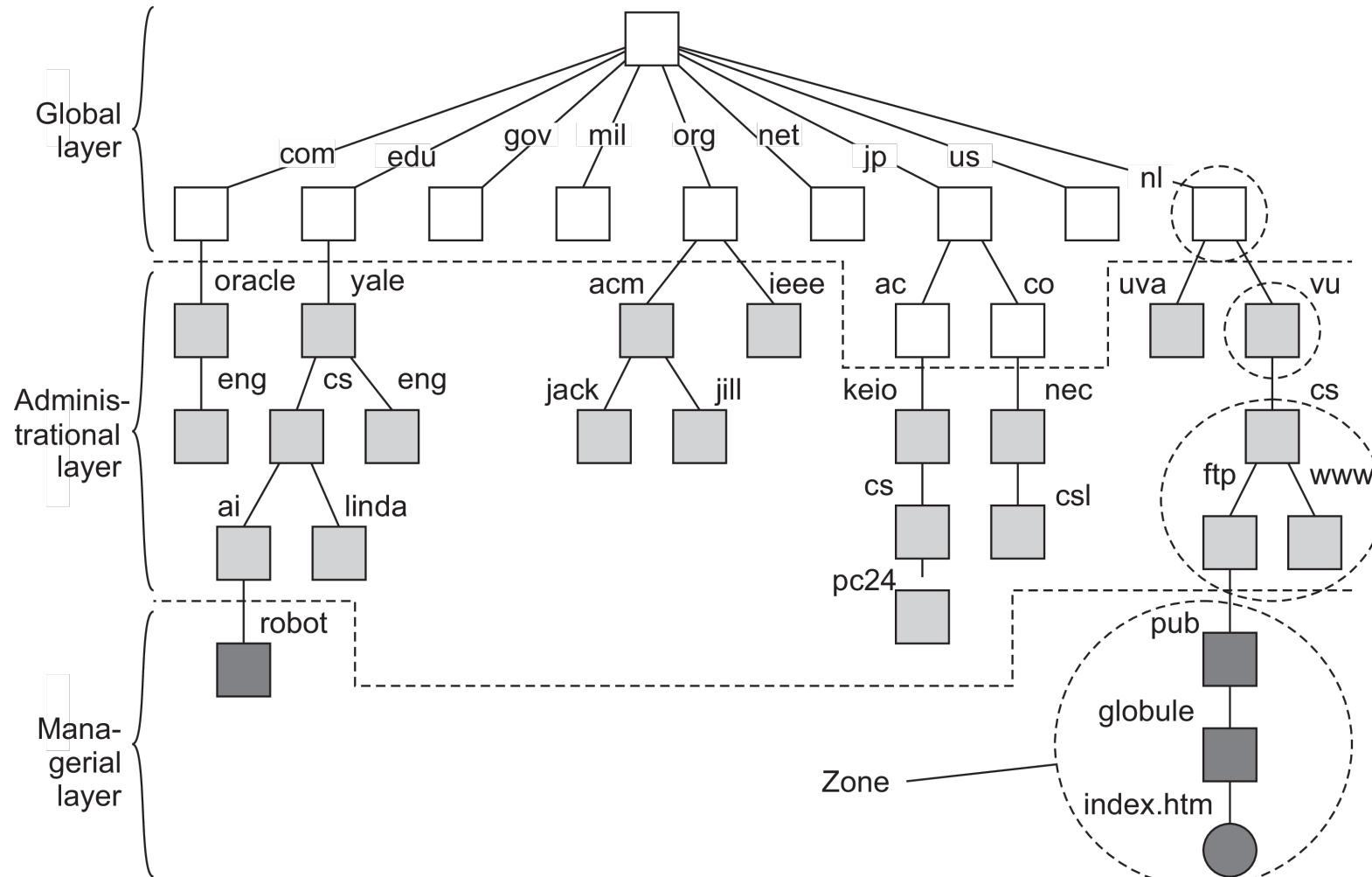
Multikast

- **Multikast** (engl. *multicast*) komunikacija podrazumeva **slanje podataka višestrukim primaocima**, podrazumeva organizaciju čvorova u distribuiranom sistemu u **prekrivajuću mrežu**
- Ako svaki čvor u prekrivajućoj mreži treba da primi poruku, onda se radi o **brodkastu** (engl. *broadcast*)
- Vrste multikasta:
 - **Multikast na nivou aplikacija zasnovan na stablima i mrežama**, za diseminaciju podataka koriste se stabla (engl. *tree*) čvorova ili meš mreža (engl. *mesh networks*) kod koje je neophodan neki vid rutiranja
 - **Multikast zasnovan na plavljenju** (engl. *flooding*), čvor P šalje poruku m svim svojim susedima, ako ranije nije video poruku m , svaki od suseda prosleđuje dalje m svim svojim susedima osim P
 - **Multikast zasnovan na ogovaranju** (engl. *gossip*), tzv. **epidemijski protokoli** (engl. *epidemic protocols*), lenja propagacija, na kraju svako ažuriranje dospeva do svake od replika, dve vrste epidemija: **anti-entropija** i **širenje glasina** (engl. *rumor spreading*)

Imenovanje u distribuiranim sistemima

- **Ime** u distribuiranom sistemu je **niz bitova ili karaktera** koji se koristi za **obraćanje nekom entitetu** kao što su **resursi** (host, printer, disk, fajl), **procesi, poruke, konekcije**, ...
- **Pristupna tačka** (engl. *access point*) za rad sa entitetima je posebna vrsta entiteta u distribuiranim sistemima čiji naziv je poznat kao **adresa**
- **Identifikatori** su posebna vrsta imena koja na jedinstven način identifikuju entitete. Svaki identifikator odnosi se na najviše jedan entitet, na svaki od entiteta referiše se samo jedan identifikator, identifikator se takođe uvek odnosi na isti entitet
- **Tri klase sistema za imenovanje** (engl. *naming systems*):
 - **ravno imenovanje** (engl. *flat naming*), povezivanje identifikatora sa adresom pridruženih entiteta, pogodno za mašine, primer DHT kod Chord
 - **strukturirano imenovanje** (engl. *structured naming*), imenski prostori, primer DNS
 - **imenovanje zasnovano na atributima** (engl. *attribut-based naming*), koriste parove (*atribut, vrednost*), poznati i kao **directory services**, primer LDAP, **directory information base** i **tree** (DIB i DIT)

Primer: DNS implementacija imenskog prostora

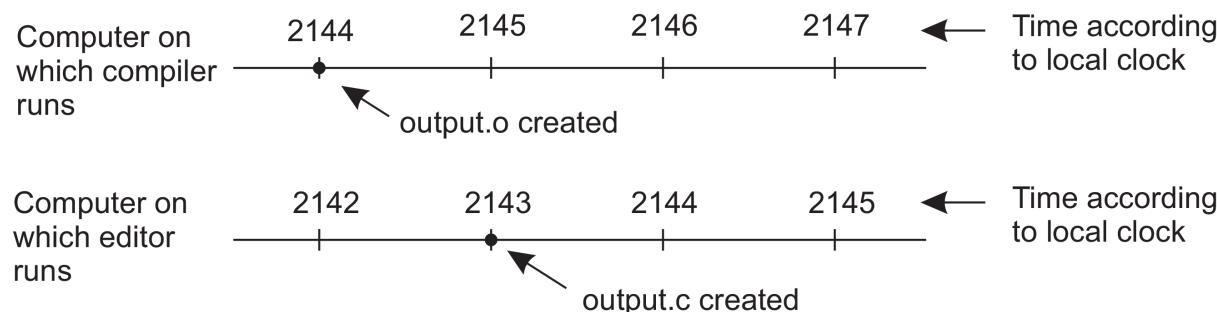


Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Koordinacija procesa

Sinhronizacija i koordinacija

- Problemi: procesi rade sa **deljenim resursima** koji se ne mogu simultano koristiti, često je potrebno da se procesi slože o **redosledu događaja**
- **Sinhronizacija i koordinacija** su blisko povezani pojmovi:
 - **Sinhronizacija procesa** obezbeđuje da se procesi “čekaju” kada je neophodno da neki od njih završe određene operacije pre nego distribuirani sistem nastavi sa daljim radom
 - **Sinhronizacija podataka** obezbeđuje da dve kopije nekog skupa podataka sadrže identične vrednosti
 - Cilj **koordinacije** je upravljanje interakcijama i zavisnostima u distribuiranim sistemu. Može se reći da **koordinacija obuhvata** između ostalog i **sinhronizaciju**
 - U centralizovanim sistemima vreme je jednoznačno određeno. Proces dobija od OS informaciju o trenutnom vremenu, primer: korišćenja make alata kod UNIX/Linux OS



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Fizički satovi

- Svi računari imaju kolo koje prati protok vremena – **sat** ili **tajmer**
- **Tajmer** se obično zasniva na precizno konstruisanom oscilatoru na bazi kvarcnog kristala SiO_2 i pridruženih registara za **brojač** (engl. counter) i **čuvanje vrednosti** (engl. holding register)
- Kada brojač odbroji do nule generiše se **prekid** (engl. interrupt) i brojač se ponovo učitava, svaki prekid odgovara jednom **otkucaju sata** (engl. clock tick)
- Međunarodno atomsko vreme (engl. International Atomic Time – TAI), zasnovano na oscilacijama cezijuma 133
- **Universal Coordinated Time** (UTC) – TAI proširen sa preskočnim sekundama (engl. leap seconds), dobija se kao vrednost oko 50 cezijumskih časovnika širom sveta
- UTC se emituje kratkim talasima putem radija i satelita
- Tačnost UTC otprilike ± 0.5 ms

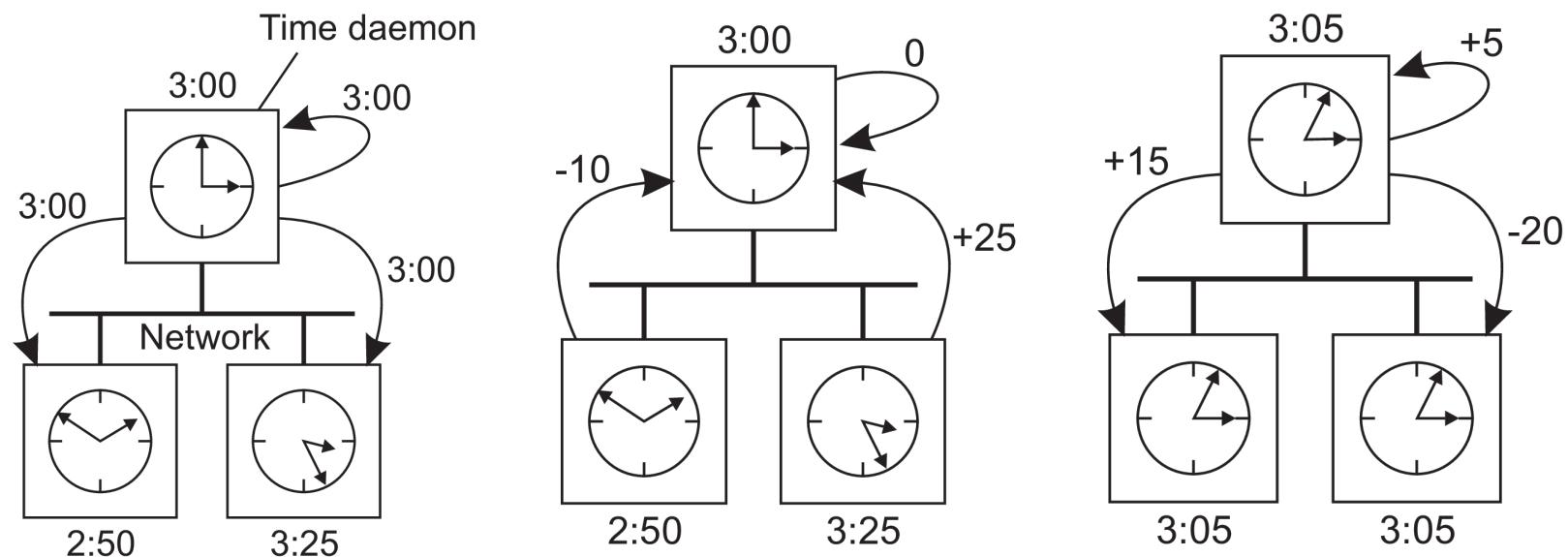


Sinhronizacija satova

- **Preciznost** (engl. *precision*):
 - Cilj je **održati devijaciju između dva sata** na bilo koje dve mašine unutar određene granice, poznate kao preciznost Π :
$$\forall t, \forall p, q : |C_p(t) - C_q(t)| \leq \Pi$$
gde je $C_p(t)$ izračunato vreme sata mašine p u UTC vremenu t
- **Tačnost** (engl. *accuracy*):
 - Kod tačnosti, cilj je **očuvati sat unutar granice α** :
$$\forall t, \forall p : |C_p(t) - t| \leq \alpha$$
- **Sinhronizacija**:
 - **interna**: očuvati satove **preciznim**
 - **eksterna**: očuvati satove **tačnim**

Sinhronizacija satova bez UTC

- **Berkli algoritam: server za vreme** (engl. *time server – time daemon*) periodično skenira sve mašine, računa srednju vrednost i potom informiše svaku mašinu kako da podesi svoje novo vreme u odnosu na svoje trenutno vreme, satovi rade brže ili sporije po potrebi
- Korišćenje servera za vreme:



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Relacija dešava-se-pre

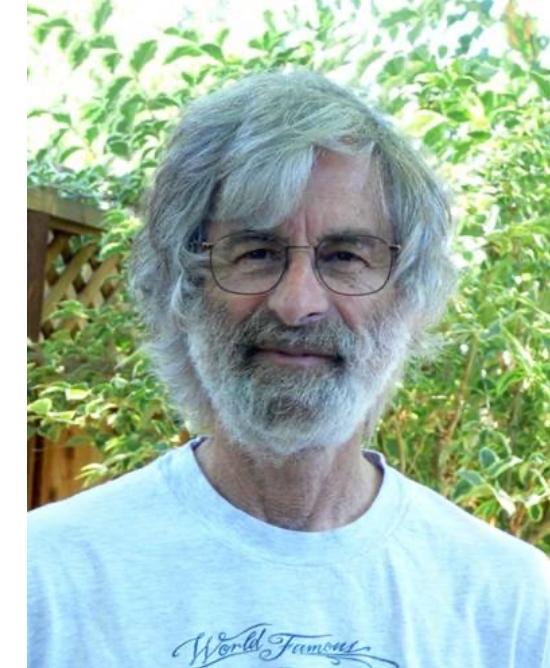
- Obično nije važno da se svi procesi slože oko tačnog vremena, već je dovoljno da se slože oko **redosleda događaja**, što podrazumeva uvođenje određenog **uređenja**
- **Relacija dešava-se-pre** (engl. *happens-before*):
 - ako su a i b dva događaja u istom procesu, i a se dešava pre b , tada važi $a \rightarrow b$
 - ako je događaj a slanje poruke i b događaj prijema te poruke, tada važi $a \rightarrow b$
 - ako je $a \rightarrow b$ i $b \rightarrow c$, onda je i $a \rightarrow c$
- Na ovaj način se uvodi **parcijalno uređenje događaja** u sistemu u kome se **procesi izvršavaju konkurentno**

Logički satovi

- **Problem:** kako održati konzistentan globalni pogled na ponašanje sistema ako koristimo dešava-se-pre relaciju?
- **Rešenje:** dodati **vremenski otisak** (engl. *timestamp*) $C(e)$ svakom događaju e , koji zadovoljava sledeće osobine:
 - S1: ako su a i b dva događaja u istom procesu i $a \rightarrow b$, tada se zahteva i da važi da je $C(a) < C(b)$
 - S2: ako a odgovara slanju poruke m i b prijemu te poruke, tada je takođe $C(a) < C(b)$
- **Problem:** kako dodati vremenski otisak događaju kada nema globalnog sata?
- **Rešenje:** uvesti i održavati konzistentan skup **logičkih satova** (engl. *logical clocks*), po jedan za svaki od procesa u sistemu

Leslie Lamport

- Leslie Lamport, „*Time, Clocks, and Ordering of Events in a System*“, CACM, 21(7):558-565, July 1978.
- 11153 citata, <https://amturing.acm.org/p558-lamport.pdf>
- Tjuringova nagrada 2013.
 - Happens-before relacija
 - Logički satovi
 - Sekvencijalna konzistencija
 - Paxos konsenzus algoritam
 - Algoritam pekare (*bakery*) za međusobno isključivanje
 - Chandy-Lamport algoritam za određivanje konzistentnog globalnog stanja (*snapshot*)
 - Lamportov potpis, jedan od prototipova sistema digitalnog potpisa
 - Temporalna logika akcija (TLA) – TLA⁺ jezik za specifikaciju i zaključivanje u konkurentnim sistemima, “Don Kihotovski pokušaj da se prevaziđe inženjerska antipatija prema matematici“



Operating
Systems
R. Stockton Gaines
Editor
**Time, Clocks, and the
Ordering of Events in
a Distributed System**
Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and it is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used for ordering the events. The use of the total ordering is illustrated with a method for reserving seats in an airplane. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchronization they can get.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocessors.

CR Categories: A.3.2, S.29

Introduction

The concept of time is fundamental to our way of thinking. It is derived from the more basic concept of the order in which events occur. We say that something happened at time t_1 if we have seen its effects after t_1 and before t_2 , that t_1 happened before t_2 . The concept of the temporal ordering of events pervades our thinking about systems. For example, we expect that a reservation for a flight that is made before the flight is filled. However, we will see that this is not necessarily true when considering events in a distributed system.

General permission to make fair use in teaching or research of all or part of this work is granted for users registered with Copyright Clearance Center (CCC) Transactional Reporting Service, provided that the fee of \$15 per article is paid directly to CCC, 27 Congress Street, Salem, MA 01970. The fee code is 0001-0732/78/0700-0558 \$0.75. © 1978 ACM 0001-0732/78/0700-0558 \$0.75.

558

A distributed system consists of a collection of distinct processes which are spatially separated, and which communicate with one another by exchanging messages. A network of interconnected computers such as the ARPANET is a distributed system. A single computer can also be viewed as a distributed system in which the central computer and its peripheral devices are separate processes. Channels are separate processes. A system is distributed if the message transmission delay is not negligible compared to the time required for local computation. We will concern ourselves primarily with systems of spatially separated computers. However, many of our results apply to distributed systems involving a single multiprocessor system on a single computer involves problems similar to those of distributed systems, except that the unpredictable order in which certain events can occur.

In a distributed system, it is sometimes impossible to

say that one of two events occurred first. The relation “happened before” is therefore only a partial ordering of events. This paper shows how this situation often arises because people are not fully aware of this fact and are not fully aware of what it means.

In this paper, we discuss the partial ordering defined by the “happened before” relation, and give a distributed algorithm for implementing this ordering among all the events. This algorithm can provide a useful mechanism for implementing a distributed system. We illustrate the use of a simple algorithm for solving synchronization problems. Unexpected, anomalous behavior can result if the ordering obtained by the algorithm differs from the ordering that would be obtained when considering events in a distributed system.

General permission to make fair use in teaching or research of all or part of this work is granted for users registered with Copyright Clearance Center (CCC) Transactional Reporting Service, provided that the fee of \$15 per article is paid directly to CCC, 27 Congress Street, Salem, MA 01970. The fee code is 0001-0732/78/0700-0558 \$0.75. © 1978 ACM 0001-0732/78/0700-0558 \$0.75.

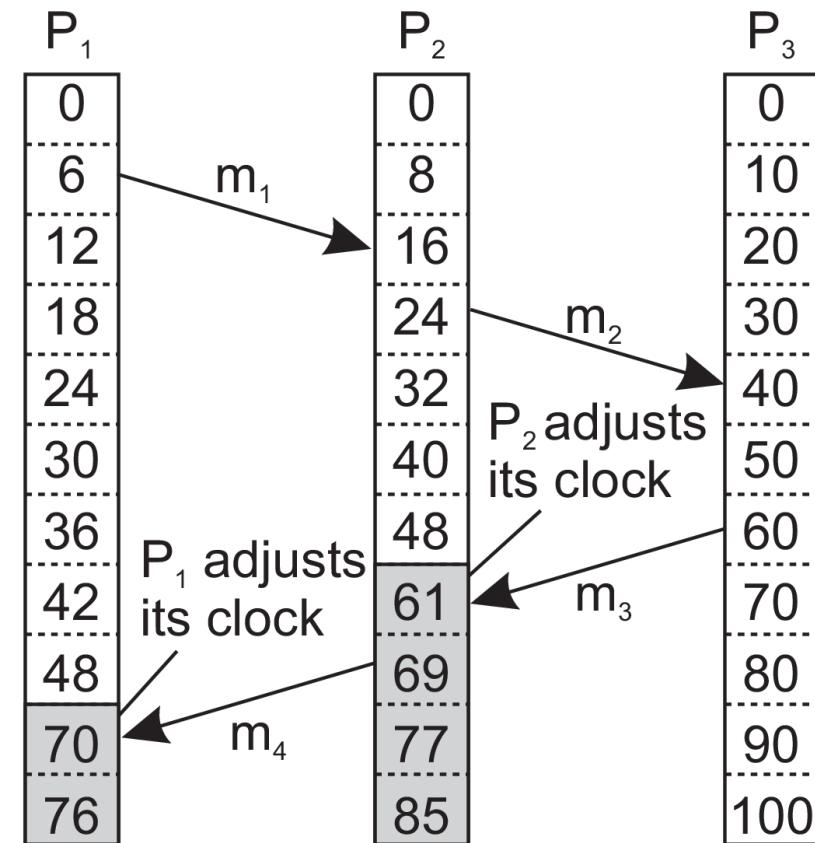
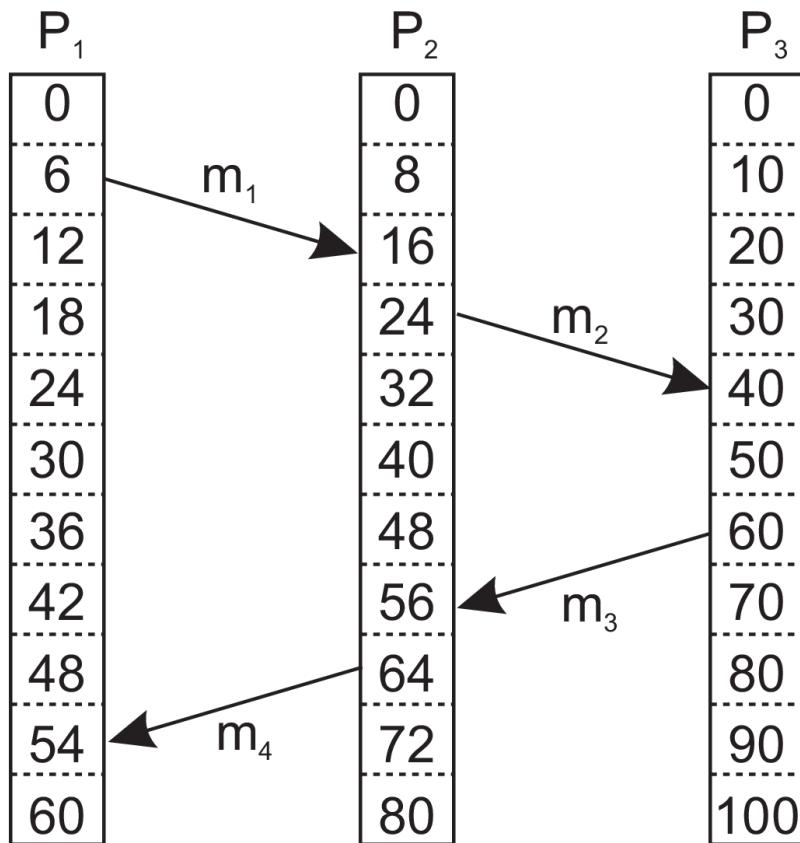
July 1978
Volume 21
Number 7

Lamportovi logički satovi

- Svaki proces P_i čuva i podešava lokalni brojač C_i
- **Lamportovi logički satovi – algoritam :**
 1. Proces P_i , pre nego što izvrši bilo koji događaj (slanje ili prijem poruke, interni događaj), inkrementira C_i za 1
 2. Kada se poruka m šalje od strane procesa P_i , poruka dobija vremenski otisak $ts(m) = C_i$
 3. Kada se poruka m primi od strane procesa P_j , P_j podešava svoj lokalni brojač $C_j = \max\{C_j, ts(m)\}$; onda izvršava korak 1 i potom prosledi poruku m aplikaciji
- Svojstvo S1 sa slajda 7 zadovoljeno je korakom 1, a S2 koracima 2 i 3
- Još uvek je moguće da se dva događaja dogode istovremeno. Ovo se izbegava uvođenjem jedinstvenih identifikatora procesa – koristi se uređeni par (C_i, i) , npr. ako postoji $(42, i)$ i $(42, j)$, $i < j$, onda je $(42, i) < (42, j)$

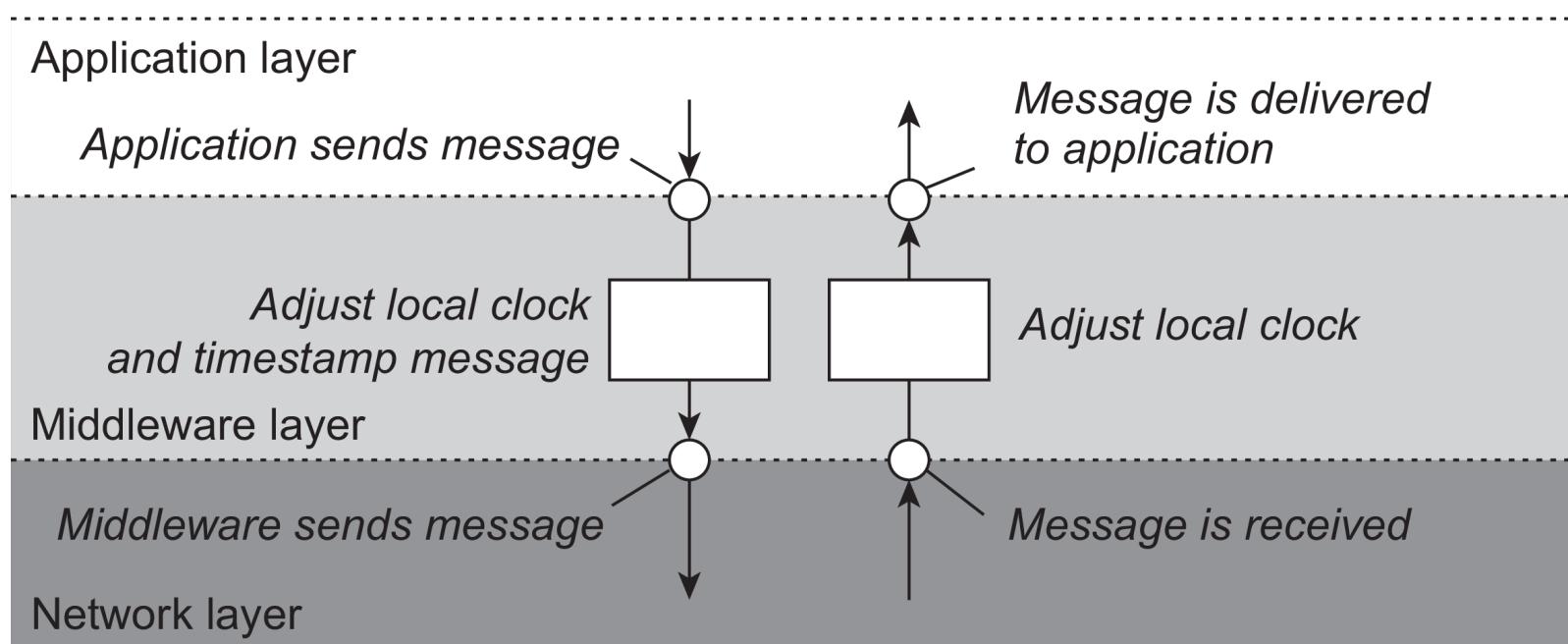
Primer: Lamportovi logički satovi

- Tri procesa sa logičkim satovima koji se inkrementiraju različitom brzinom:



Implementacija logičkih satova

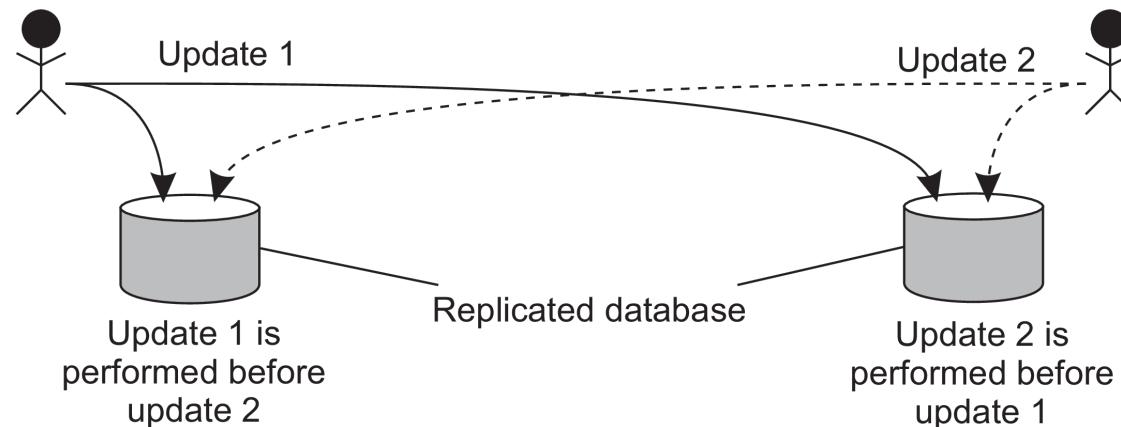
- Logički satovi ne garantuju kauzalnost između događaja, za kauzalne zavisnosti neophodni su **vektorski satovi** (engl. vector clocks), svaki od procesa pamti i vektor sa **kauzalnom istorijom**
- Sva podešavanja logičkih satova dešavaju se u midlver sloju:



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Primer: totalno-uređeni multikast

- Konkurentna ažuriranja (engl. *update*) replicirane baze podataka se moraju realizovati svuda u istom redosledu
 - P_1 dodaje 1000 RSD na račun (prethodno stanje je 10000 RSD)
 - P_2 uvećava iznos na računu za 1%
 - postoje dve replike baze podataka
 - ako ne postoji odgovarajući mehanizam za sinhronizaciju, može se desiti da nakon prethodnih operacija replike sadrže različite vrednosti stanja na računu



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Primer: totalno-uređeni multikast

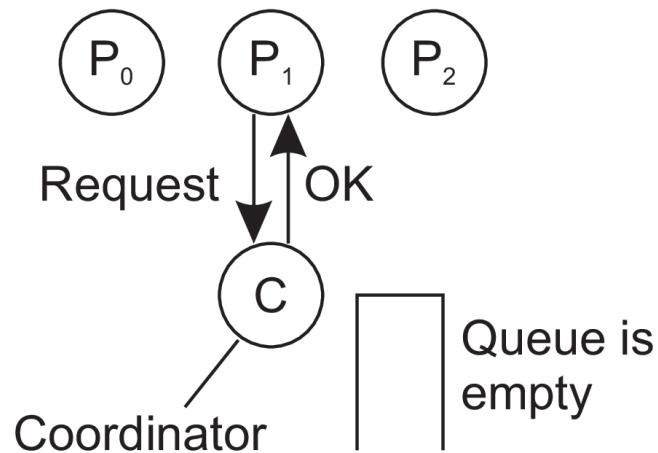
- **Rešenje:**
 - Proces P_i šalje poruku sa vremenskim otiskom m_i svim drugim procesima. Poruka se smešta u lokalni red red_i .
 - Svaka poruka koju dobija P_j se u skladu sa svojim vremenskim otiskom smešta u red red_j i prijem se potvrđuje svim ostalim procesima
 - P_j prosleđuje poruku m_i svojoj aplikaciji ako:
 - m_i se nalazi na početku reda red_j
 - za svaki proces P_k postoji poruka m_k u redu red_j sa većim vremenskim otiskom
 - Pretpostavka je da je komunikacija pouzdana i da je uređena po FIFO principu
- Replike se održavaju konzistentnim tako što se svuda izvršavaju iste operacije u istom redosledu, **replike prate iste prelaze u istom konačnom automatu – replikacija konačnih automata** (engl. *state machine replication*)

Međusobno isključivanje

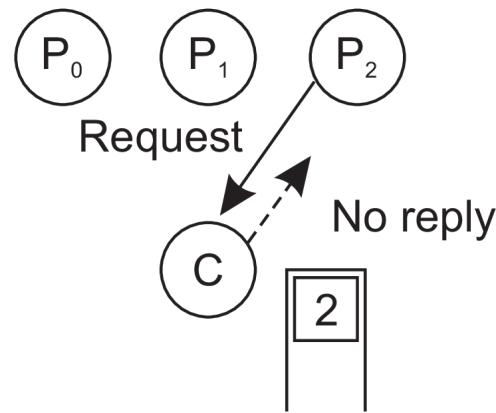
- Problem **međusobnog isključivanja** (engl. *mutual exclusion*) se javlja kada **više procesa u distribuiranom sistemu želi ekskluzivni pristup nekom resursu**
- Osnovna podela **algoritama za međusobno isključivanje**:
 - **zasnovani na dozvoli** (engl. *permission-based*): proces, koji želi da uđe u kritičnu sekciju (deo koda koji u nekom trenutku može izvršavati samo jedan proces) ili pristupi resursu, traži dozvolu od drugih resursa, primeri centralizovani algoritam i Ricart-Agarwala distribuirani algoritam
 - **zasnovani na tokenu** (engl. *token-based*): token se razmenjuje između procesa. Proces koji poseduje token može ući u kritičnu sekciju ili koristiti resurs ili, ako nije zainteresovan, može proslediti token dalje, primer token ring algoritam

Centralizovani algoritam zasnovan na dozvoli

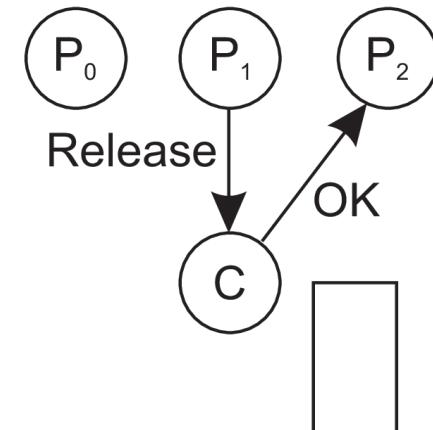
- **Koordinator** igra centralnu ulogu:



(a) Proces P_1 traži od koordinatora dozvolu za pristup deljenom resursu. Zahtev je odobren.



(b) Proces P_2 traži od koordinatora dozvolu za pristup istom resursu. Koordinator ne odgovara.



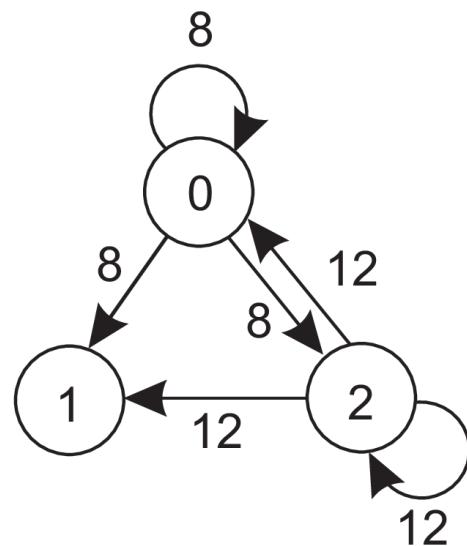
(c) Kada P_1 oslobodi resurs, javlja koordinatoru, koji potom odgovara P_2

Ricart-Agrawala distribuirani algoritam

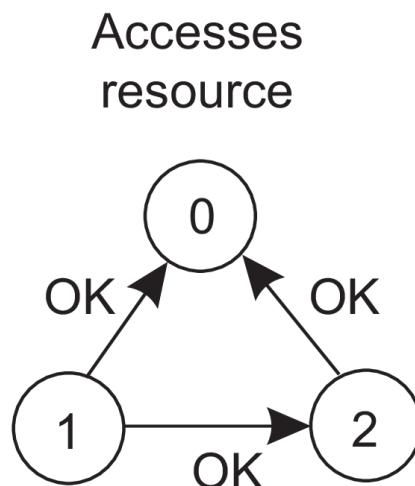
- **Distribuirani algoritam** zasnovan na Lamportovim **logičkim satovima**, slanje poruka se smatra pouzdanim (nema izgubljenih poruka)
- Proces šalje **poruku svim procesima** sa nazivom resursa, svojim brojem i trenutnim logičkim vremenom. Kada proces primi poruku, **akcija zavisi od njegovog sopstvenog stanja u odnosu na resurs** naveden u poruci:
 - ako primalac ne pristupa resursu i ne želi pristup, šalje pošiljaocu *OK* poruku
 - ako primalac već ima pristup resursu, jednostavno ne odgovara i stavlja zahtev u red
 - ako i primalac želi pristup resursu ali ga još uvek nema, poredi vremenski otisak u poruci sa onim u poruci koju je on poslao svima. Ako primljena poruka sadrži nižu vrednost vremenskog otiska, proces šalje pošiljaocu *OK* poruku. U slučaju da njegova poruka ima nižu vrednost otiska, primalac stavlja zahtev u red i ne odgovara ništa
- **Proces odgovara na zahtev** samo kada **nema interesa za resursom ili kada čeka na resurs, ali ima niži prioritet**. U ostalim slučajevima, **odgovor se odlaze**

Ricart-Agrawala distribuirani algoritam

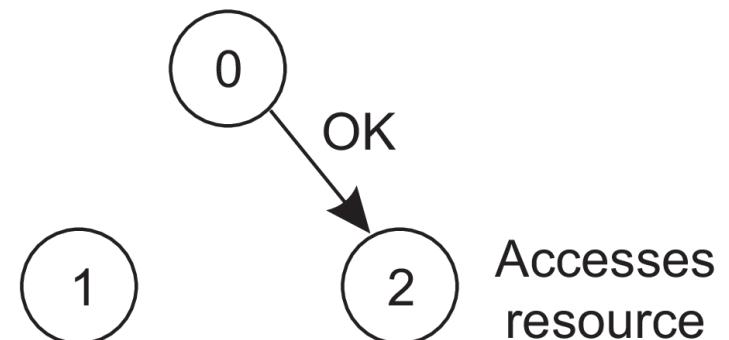
- Primer sa tri procesa:



(a) Dva procesa žele pristup istom resursu u isto vreme



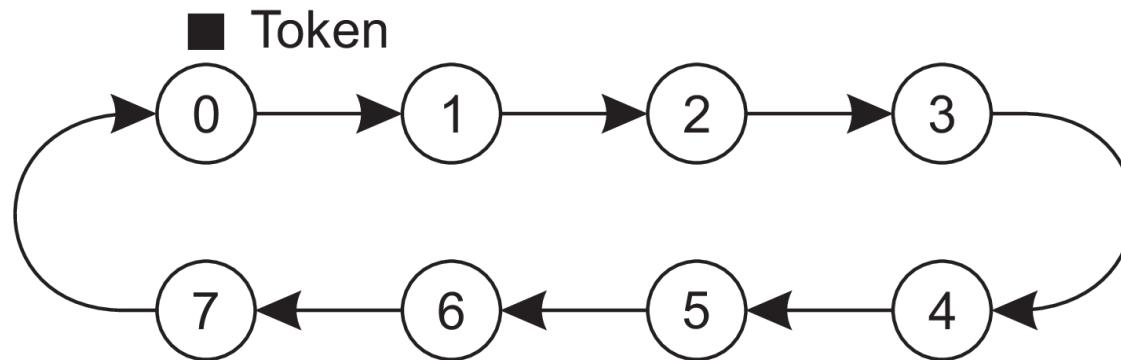
(b) Proces P_0 ima manju vrednost vremenskog otiska i pobeduje



(c) Kada P_0 završi, šalje OK poruku, tako da P_2 sada može da pristupi

Token ring algoritam

- Procesi se organizuju u **logički prsten** (engl. *ring*) u kome se razmenjuje **token**. Proses koji ima token može ući u kritičnu sekciju ili koristiti deljeni resurs (ako to želi)
- Prednosti: očigledna tačnost – samo jedan proces može imati token, ne može doći do izgladnjivanja (engl. *starvation*)
- Nedostaci: ako se token izgubi, mora biti ponovo generisan, detekcija gubitka tokena je komplikovana
- **Prekrivajuća mreža** se konstruiše kao **logički prsten sa cirkulišućim tokenom**:



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Decentralizovano međusobno isključivanje

- Prepostavimo da je svaki resurs repliciran N puta i da svaka replika ima svog koordinatora. Pristup resursu se zasniva na **algoritmu glasanja** (engl. *voting algorithm*) i zahteva većinski glas $m > N/2$ koordinatora
- Koordinator uvek trenutno odgovara na zahtev. Kada se koordinator sruši, brzo se oporavlja, ali zaboravlja na dozvole koje je poslao
- Robustnost sistema:
 - neka je $p = \Delta t/T$ verovatnoća da se koordinator resetuje tokom vremenskog intervala Δt , dok mu je životni vek T
 - verovatnoća $P[k]$ da se k od ukupno m koordinatora resetuje tokom istog intervala je
$$\mathbb{P}[k] = \binom{m}{k} p^k (1-p)^{m-k}$$
 - f koordinatora se resetuje \Rightarrow tačnost je narušena samo kada postoji manjina neotkazanih koordinatora: kada je $m - f \leq N/2$ ili $f \geq m - N/2$
 - verovatnoća narušenja tačnosti je $\sum_{k=m-N/2}^N \mathbb{P}[k]$

Decentralizovano međusobno isključivanje

- **Verovatnoća narušenosti tačnosti** za različite vrednosti parametara:

N	m	p	Violation
8	5	3 sec/hour	$< 10^{-15}$
8	6	3 sec/hour	$< 10^{-18}$
16	9	3 sec/hour	$< 10^{-27}$
16	12	3 sec/hour	$< 10^{-36}$
32	17	3 sec/hour	$< 10^{-52}$
32	24	3 sec/hour	$< 10^{-73}$

N	m	p	Violation
8	5	30 sec/hour	$< 10^{-10}$
8	6	30 sec/hour	$< 10^{-11}$
16	9	30 sec/hour	$< 10^{-18}$
16	12	30 sec/hour	$< 10^{-24}$
32	17	30 sec/hour	$< 10^{-35}$
32	24	30 sec/hour	$< 10^{-49}$

- Zaključak: **verovatnoća da se naruši tačnost** može biti tako **niska** da je **zanemarljiva u odnosu na druge tipove otkaza**
- Ako proces ne dobije dovoljno glasova, sledeći pokušaj se dešava nakon slučajno odabranog vremena.
- Nedostatak: ako **veliki broj procesa** pokuša istovremeno da pristupi **resursu**, može doći do situacije da **nijedan ne dobije dovoljno glasova**, a da se **resurs u stvari ni ne koristi**

Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Poređenje algoritama

- Poređenje četiri posmatrana algoritma za međusobno isključivanje, point-to-point poruke, dva parametra – broj poruka potrebnih kako bi proces pristupio resursu i oslobodio ga, čekanje pre pristupa:

Algorithm	Messages per entry/exit	Delay before entry (in message times)
Centralized	3	2
Distributed	$3 \cdot (N - 1)$	$2 \cdot (N - 1)$
Token ring	$1, \dots, \infty$	$0, \dots, N - 1$
Decentralized	$2 \cdot m \cdot k + m, k = 1, 2, \dots$	$2 \cdot m \cdot k$

- Centralizovani algoritam; najprostiji i najefikasniji
- Distribuirani algoritam: $N - 1$ zahteva, $N - 1$ odobrenja
- Token ring algoritam: promenljivo, zavisi od interesovanja procesa za resurs
- Decentralizovani algoritam: zahtev i izlaz ide ka m koordinatora
- Jedinica za kašnjenje: message transfer time unit – MTTU

Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Algoritmi za izbore (*Election Algorithms*)

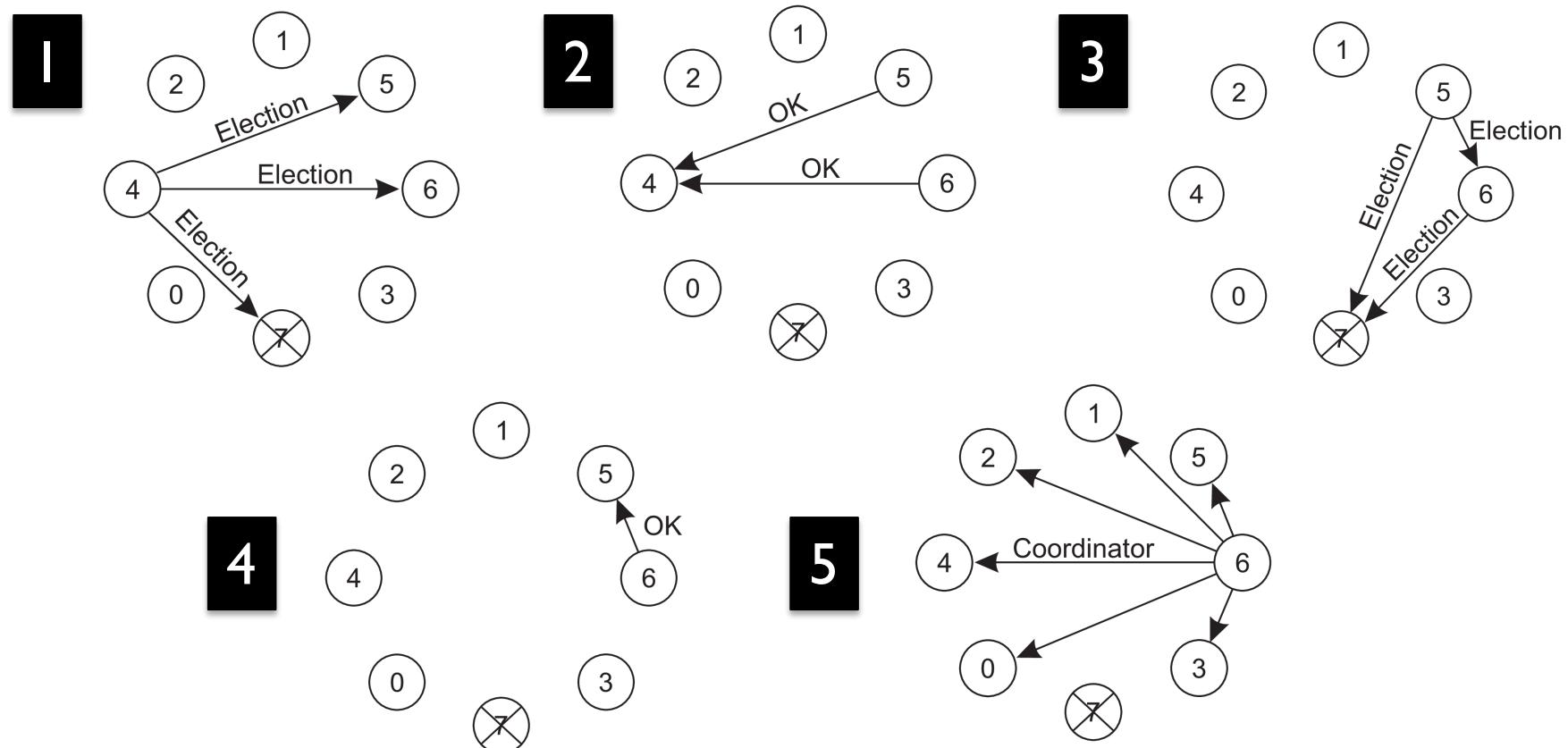
- Mnogi distribuirani algoritmi zahtevaju da se neki **proces** ponaša kao koordinator, inicijator ili ima neku drugu **posebnu ulogu**. Pitanje **dinamičkog izbora ovog posebnog procesa** rešavaju **algoritmi za izbore** (engl. *election algorithms*)
- U mnogim sistemima, koordinator se bira manuelno (npr. fajl serveri). Ovakav pristup vodi **centralizovanim rešenjima** koja imaju **jednu tačku otkaza** (engl. *Single Point of Failure* – SPoF)
- Osnovne pretpostavke:
 - svi procesi imaju **jedinstvene identifikatore**
 - svi procesi **znaju identifikatore svih drugih procesa** u sistema (ali ne i da li su pokrenuti ili ne)
 - **proces izbora** (engl. *election*) podrazumeva **identifikaciju procesa** koji imaju **najveću vrednost identifikatora** i trenutno se izvršava

Algoritam siledžije

- **Algoritam siledžije** (engl. *bully algorithm*) predložio je Garcia-Molina 1982.
- **Proces sa najvećom vrednošću identifikatora uvek pobeduje**
- Posmatramo N procesa $\{P_0, \dots, P_{N-1}\}$, gde je $id(P_k) = k$. Kada proces P_k primeti da koordinator više ne odgovara na zahteve, on inicira **izbore**:
 1. P_k šalje *ELECTION* poruku svim procesima sa većim vrednostima identifikatora: $P_{k+1}, P_{k+2}, \dots, P_{N-1}$
 2. Ako niko ne odgovori, P_k osvaja izbore i postaje koordinator
 3. Ako neki od procesa sa većim identifikatorom odgovori sa *OK* porukom, on preuzima dalje i održava izbore, a P_k odustaje. Na kraju svi procesi odustanu, osim jednog koji je novi koordinator

Algoritam siledžije

- Primer rada algoritma siledžije sa osam procesa čiji su identifikatori od 0 do 7, prethodno je proces 7 bio koordinator, ali se upravo srušio:



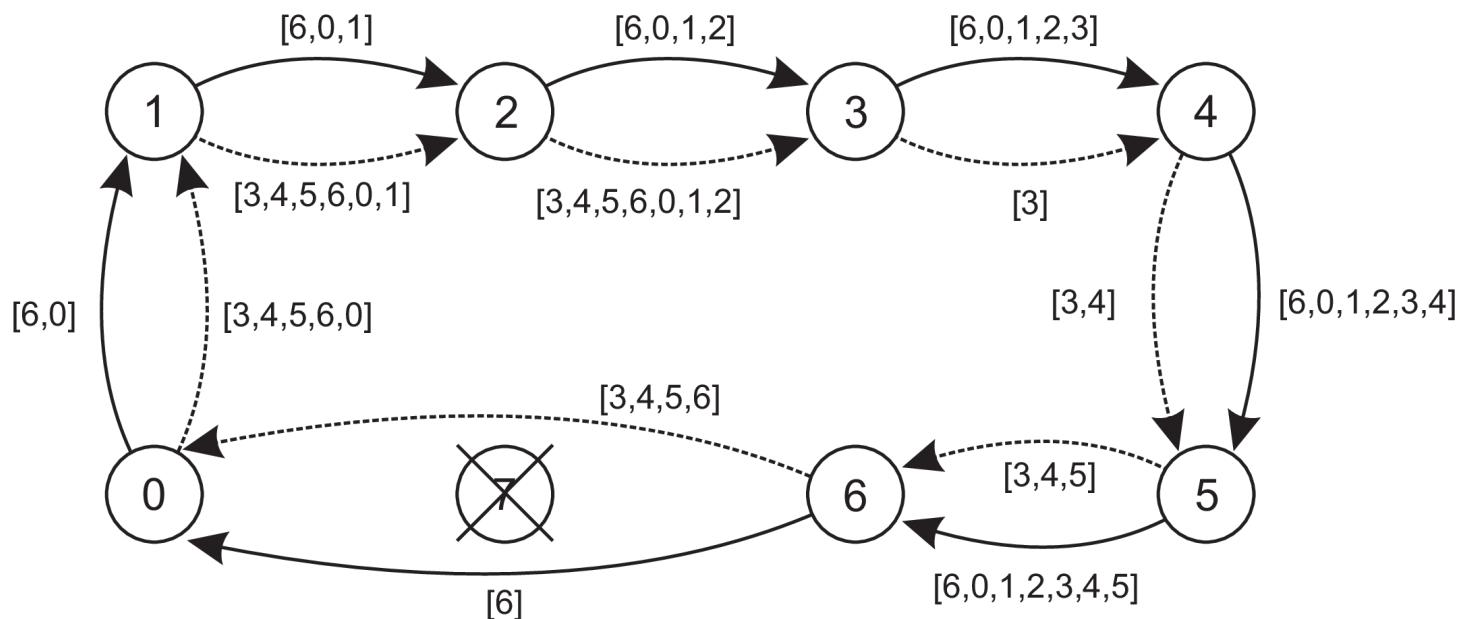
Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Algoritam izbora u prstenu

- Kod **algoritma izbora u prstenu** (engl. *ring algorithm*) prioritet procesa se dobija organizacijom procesa u (logički) prsten. **Proces sa najvišim prioritetom** biće **izabran za koordinatora**
- Pretpostavka je da svaki od procesa zna svog **sledbenika** (engl. successor)
- Bilo koji proces može započeti izbore slanjem *ELECTION* poruke svom sledbeniku. Ako se sledbenik ne izvršava, poruka se prosleđuje sledećem sledbeniku
- Prilikom prosleđivanja poruke, pošiljalac dodaje sebe u listu u okviru poruke čime se praktično kandiduje za koordinatora. Kada poruka stigne nazad do inicijatora, svi procesi su imali priliku da ukažu na svoje prisustvo
- Inicijator potom šalje *COORDINATOR* poruku koja sadrži listu svih “živih” procesa kroz prsten. Novi koordinator je proces sa najvišim identifikatorom

Algoritam izbora u prstenu

- Primer rada algoritma izbora u prstenu kada procesi P_3 i P_6 istovremeno otkriju da se prethodni koordinator (P_7) srušio. Oba kreiraju *ELECTION* poruke koje obe prođu čitav krug. P_3 i P_6 ih potom konvertuju u *COORDINATOR* poruku sa identičnim članovima i u istom redosledu (isprekidana linija odgovara poruka iniciranim od P_3 , puna linija od P_6):



Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>