

Cursul 1

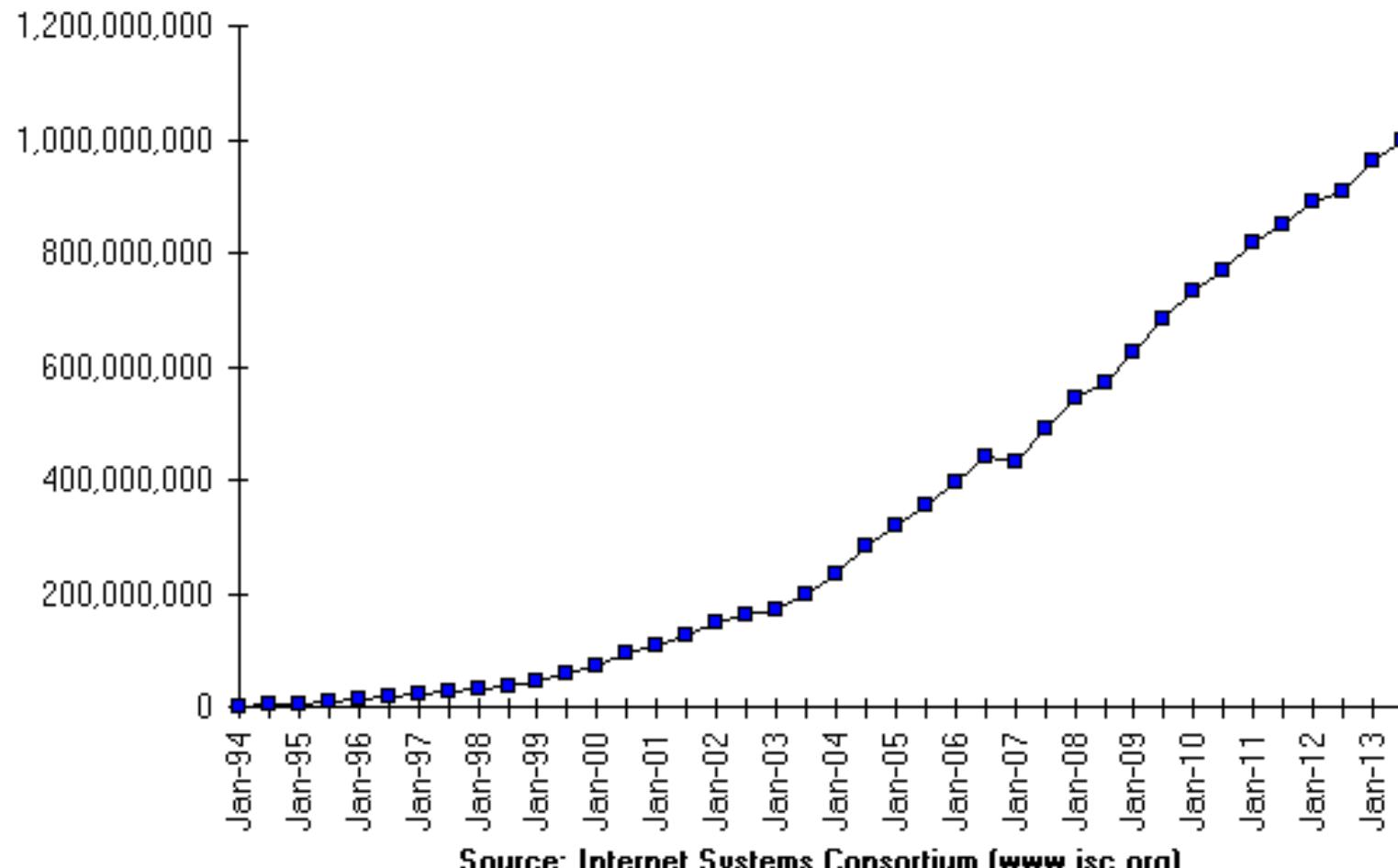
Sisteme Distribuite

Bibliografie recomandată

- Grigoraș D., "Modele de calcul distribuit", Ed. Specturm, Iași, 1999.
- Muhl, G., Fiege L, Pietzuch P., „Distributed event based systems”, Springer-Verlag Berlin Heidelberg, Germania, 2006
- Păstravanu O. "Sisteme cu evenimente discrete – Tehnici calitative bazate pe formalismul rețelelor Petri", Ed. Matrix Rom, București, 1997.
- Pfister G., "In search of clusters", Prentice Hall, 1995.
- Raynal, M. , "Distributed Algorithms and Protocols", John Wiley & Sons Ltd, 1988.
- Raynal, M., Helary,J.M., "Syncronisation and Control of Distributed Systems and Programs ", John Wiley & Sons Ltd, 1988.
- Sloam, M., Kramer, J. "Distributed Systems and Computer Networks ", Prentice Hall 1987.
- Tanenbaum A. S. "Rețele de calculatoare", Computer Press Agora, 1997.
- Tanenbaum A. S., „Modern Operating Systems“, Prentice Hall, USA,1992.
- Tanembaum A. S., "Distributed Systems", Prentice Hall 1993.
- Zaharia M. H., "Fluxuri de producție în medii distribuite", Ed. F.C. "Renașterea Română",Iași,2000.

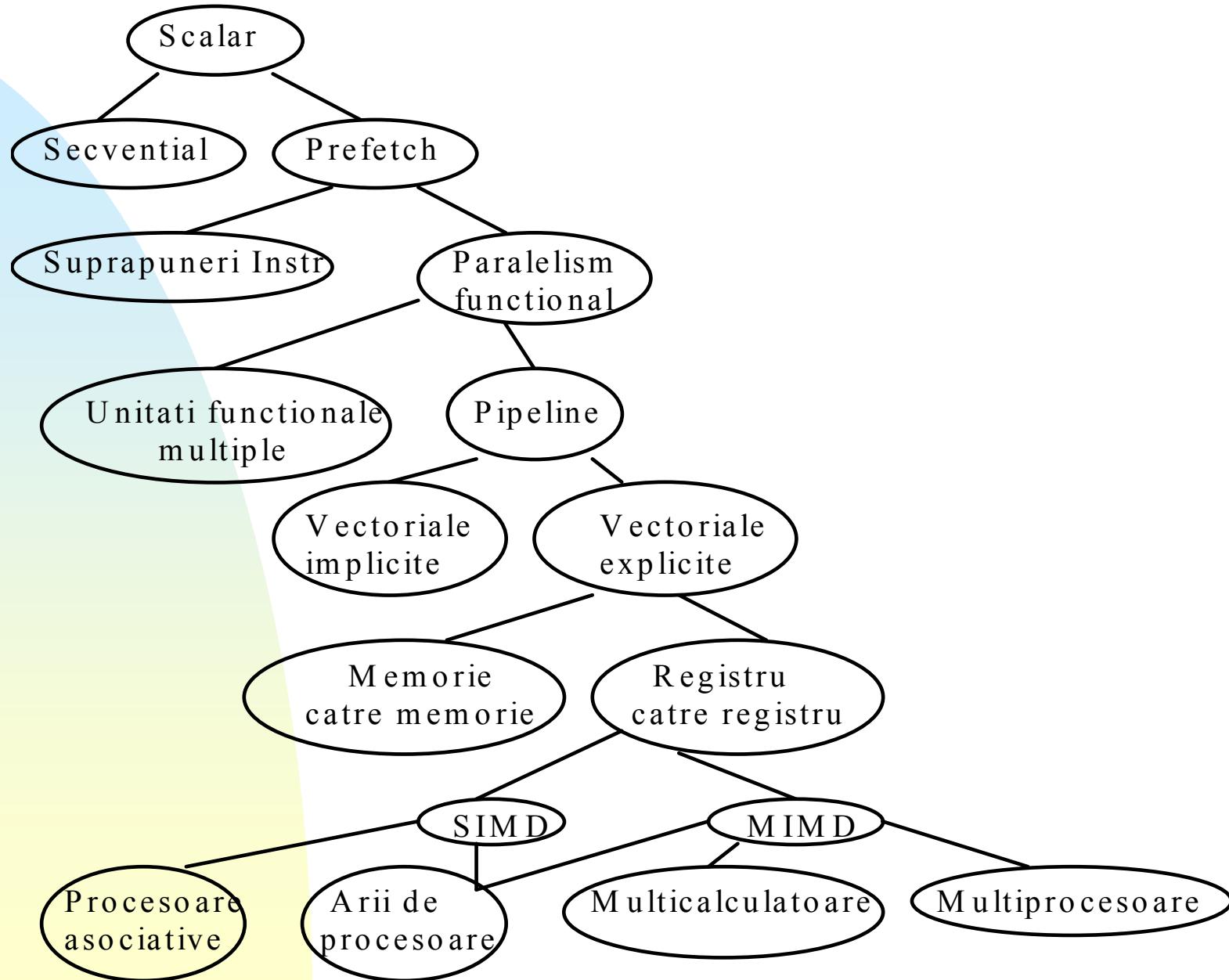
Cresterea numarului de *host-uri* din Ianuarie 1994 până în 2013

Internet Domain Survey Host Count



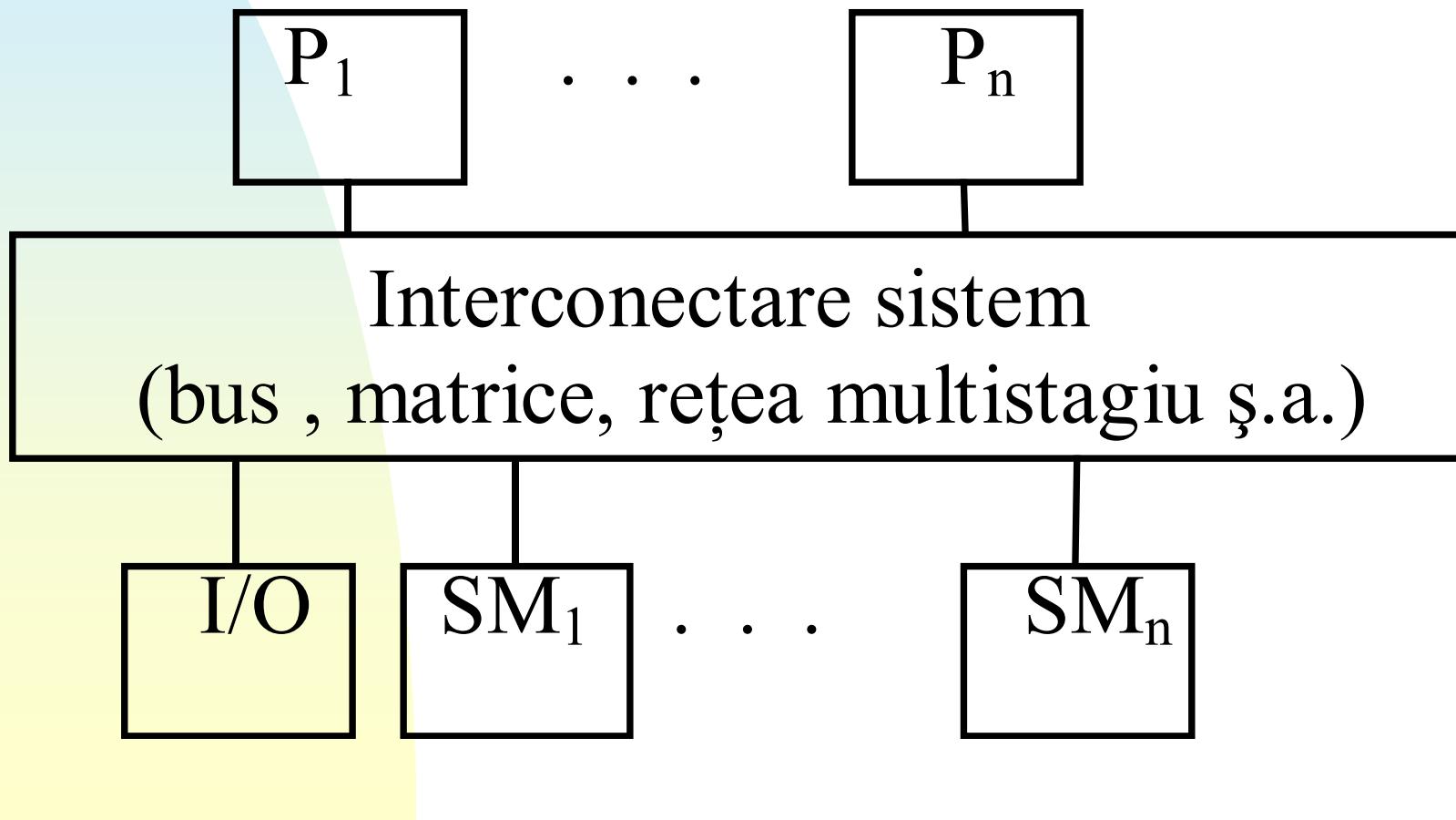
Source: Internet Systems Consortium (www.isc.org)

Evoluția principalelor clase de arhitecturi.



Modelul cu acces uniform la memorie

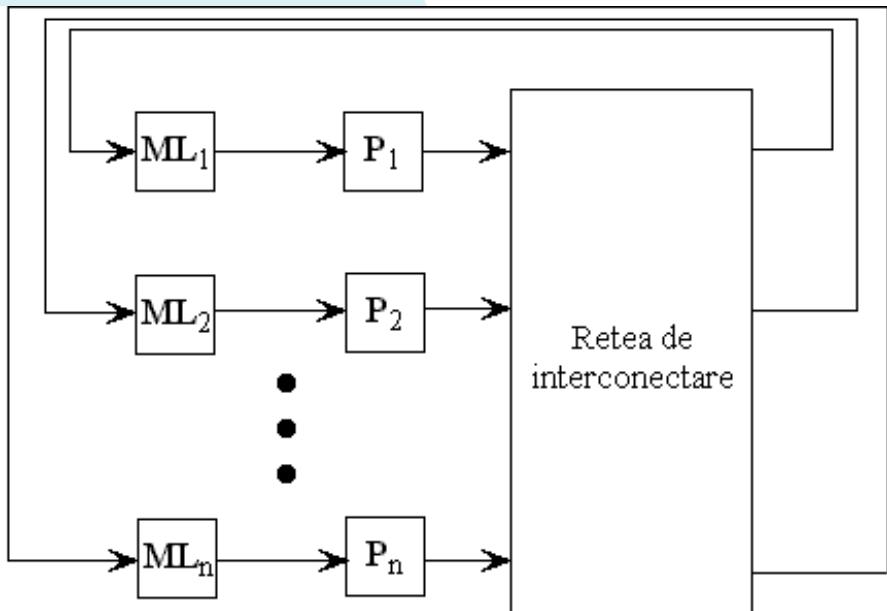
- De obicei este cunoscut sub acronimul UMA (“Uniform Memory Acces”).
- Structura specifică este prezentată în figura



Modelul cu acces neuniform la memorie

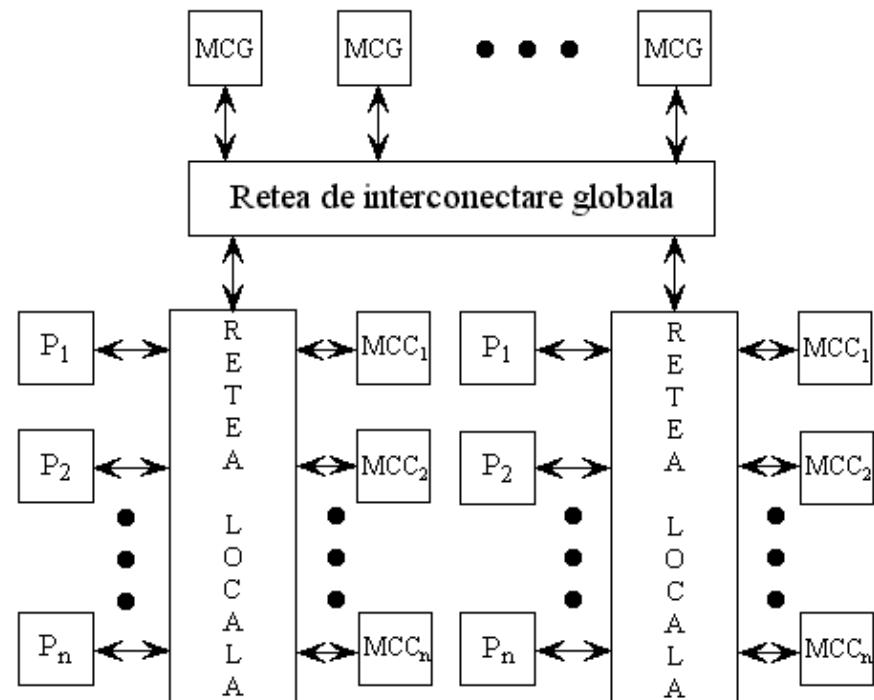
De obicei este referit ca NUMA (“Non-Uniform Memory Acces”) și are două variante:

1. *NUMA cu memorii locale distribuite*
2. NUMA ierarhic



ML = Memorie Locală; P = Procesor

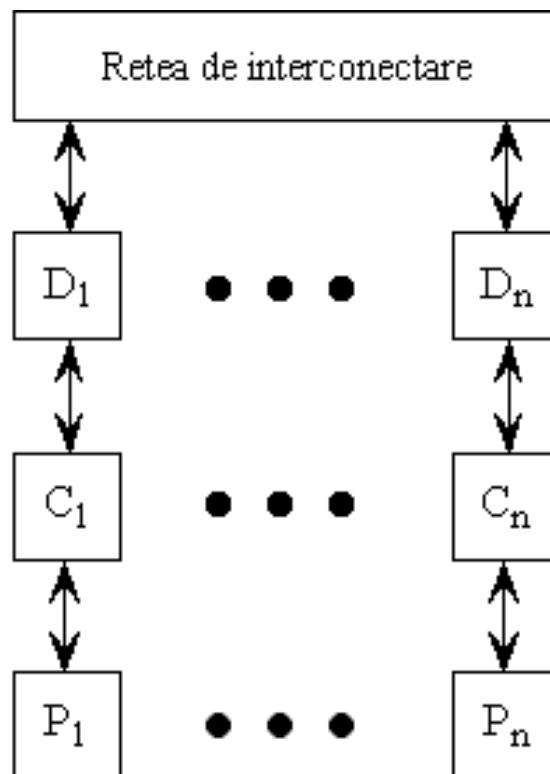
a. Cu memorii locale distribuite



MCG = Memorie Comuna Generală; MCC = Memorie Comuna în Cluster;
P = Procesor

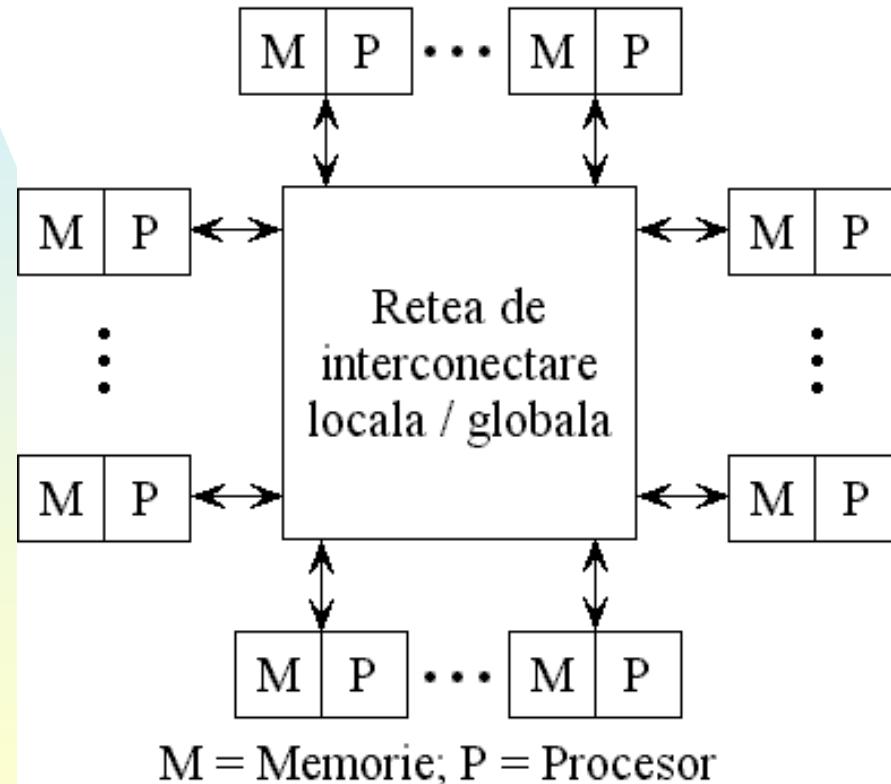
b. Ierarhic cu clustere

- **Modelul cu acces la memorie cache**
- În literatura de specialitate este denumit COMA (“Cache Only Memory Acces”) este prezentat în



D = Director; C = Cache; P = Procesor

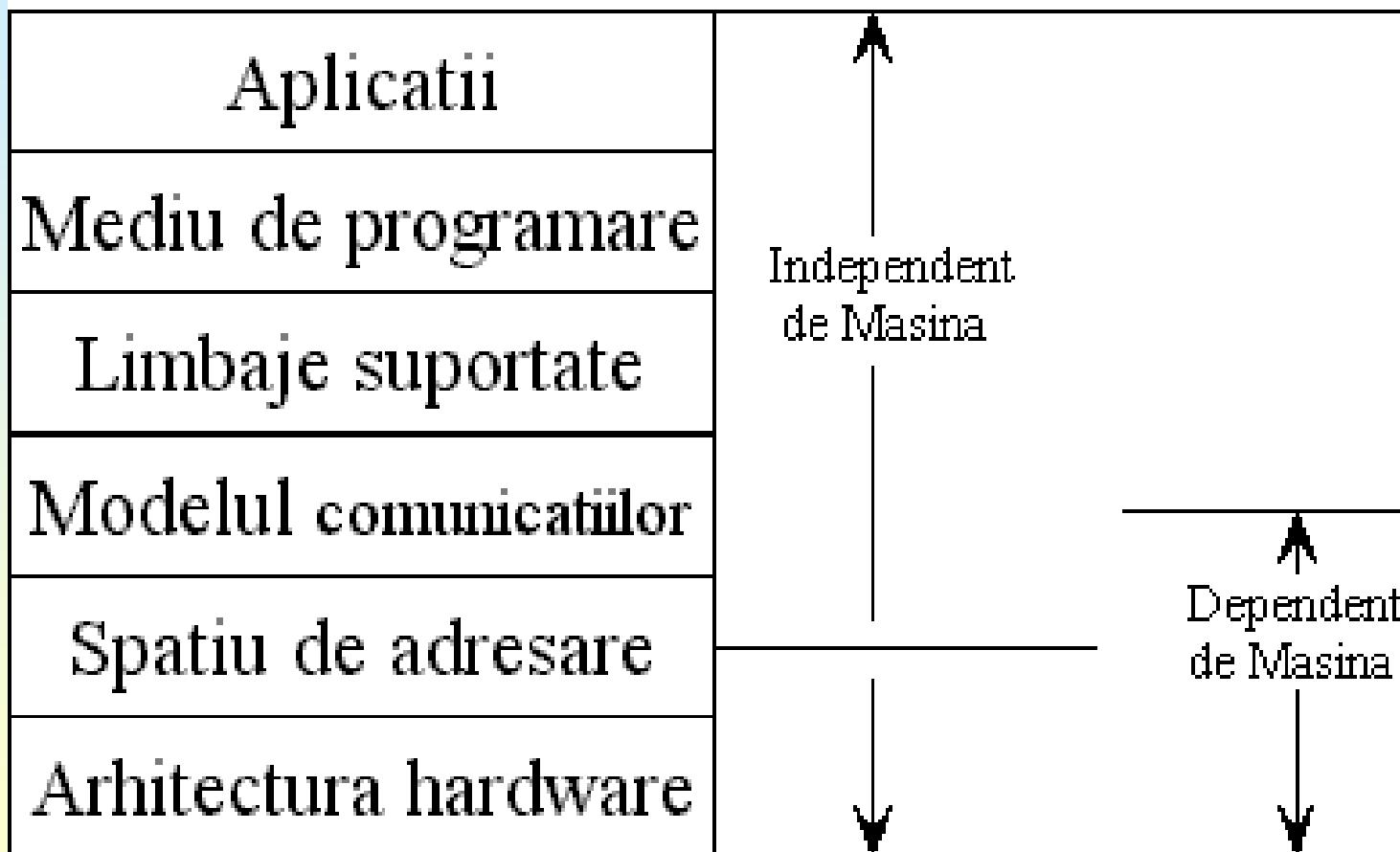
- **Modelul sistemelor slab cuplate**
- Este cel mai general model arhitectonic



Multicalcătoare cu memorie distribuită

Introducere

- Nivelul până la care urcă dependența de mașină, propus de Li (mai jos)



Introducere

- Apare întrebarea "toate cele trei dimensiuni de definire a unui sistem de calcul trebuie să fie distribuite pentru a putea declara un sistem ca distribuit" sau este una suficientă.
- La această întrebare s-au încercat mai multe răspunsuri, dintre care unul îl reprezintă modelul lui Enslow (1978)

Introducere

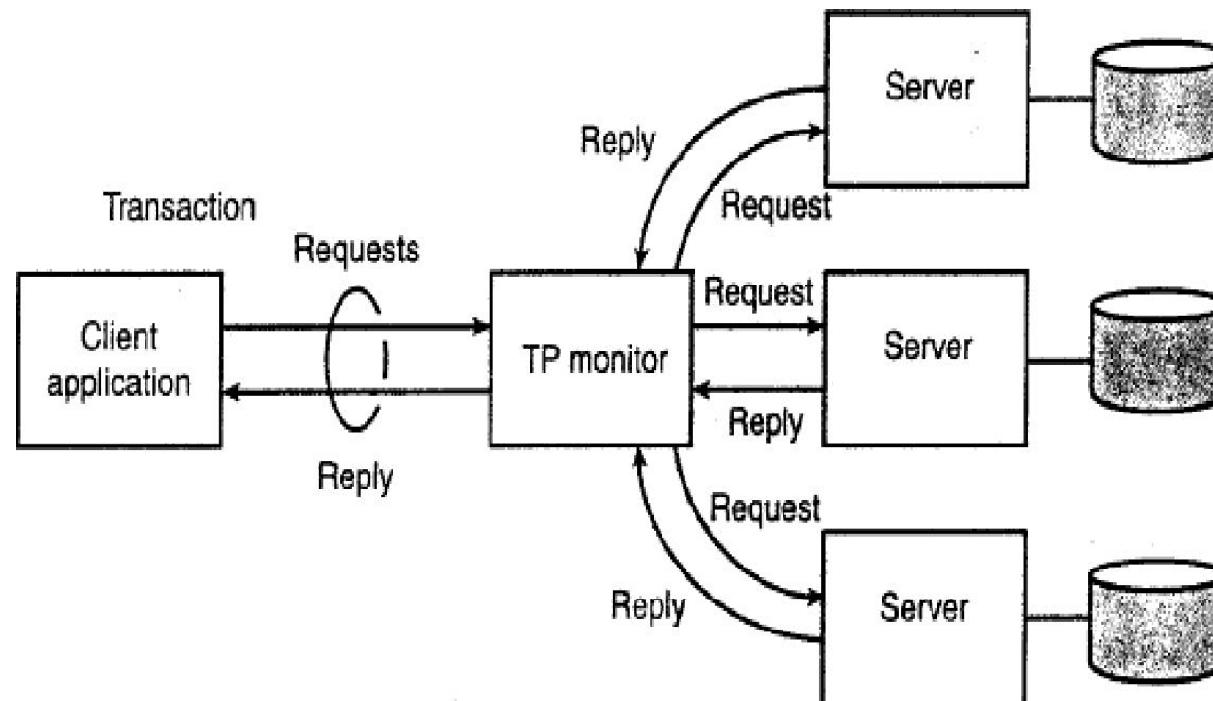
- Din acest punct de vedere o definiție mai practică este cea prezentată de Sloam & Kramer și anume:
 - *"Un sistem de calcul distribuit este acela în care un număr de procesoare și zone de stocare de date autonome suportă procesări și / sau interacțiuni cu bazele de date în scopul cooperării comune pentru a obține un rezultat comun. Procesele coordonează activitățile, schimbă informații prin intermediul unei rețele de comunicații".*

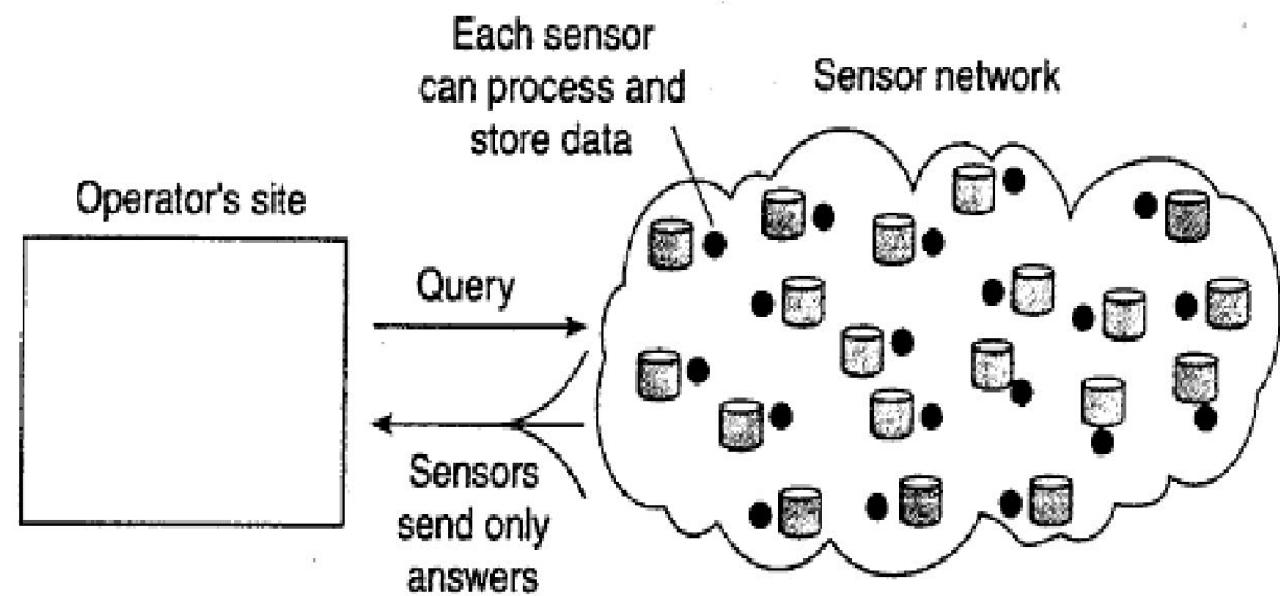
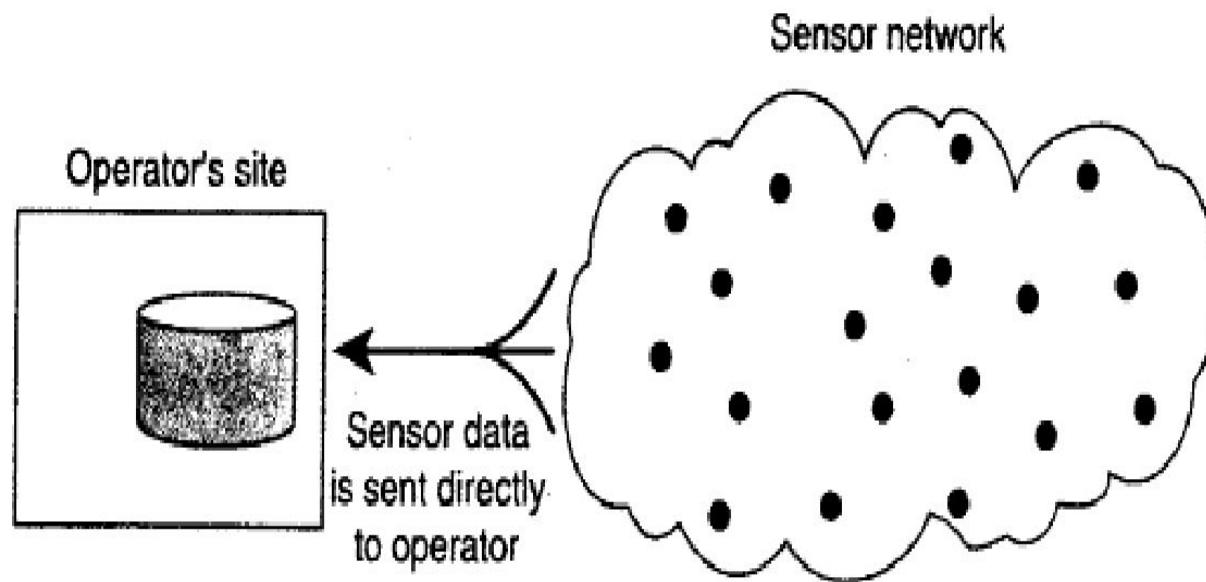
Introducere

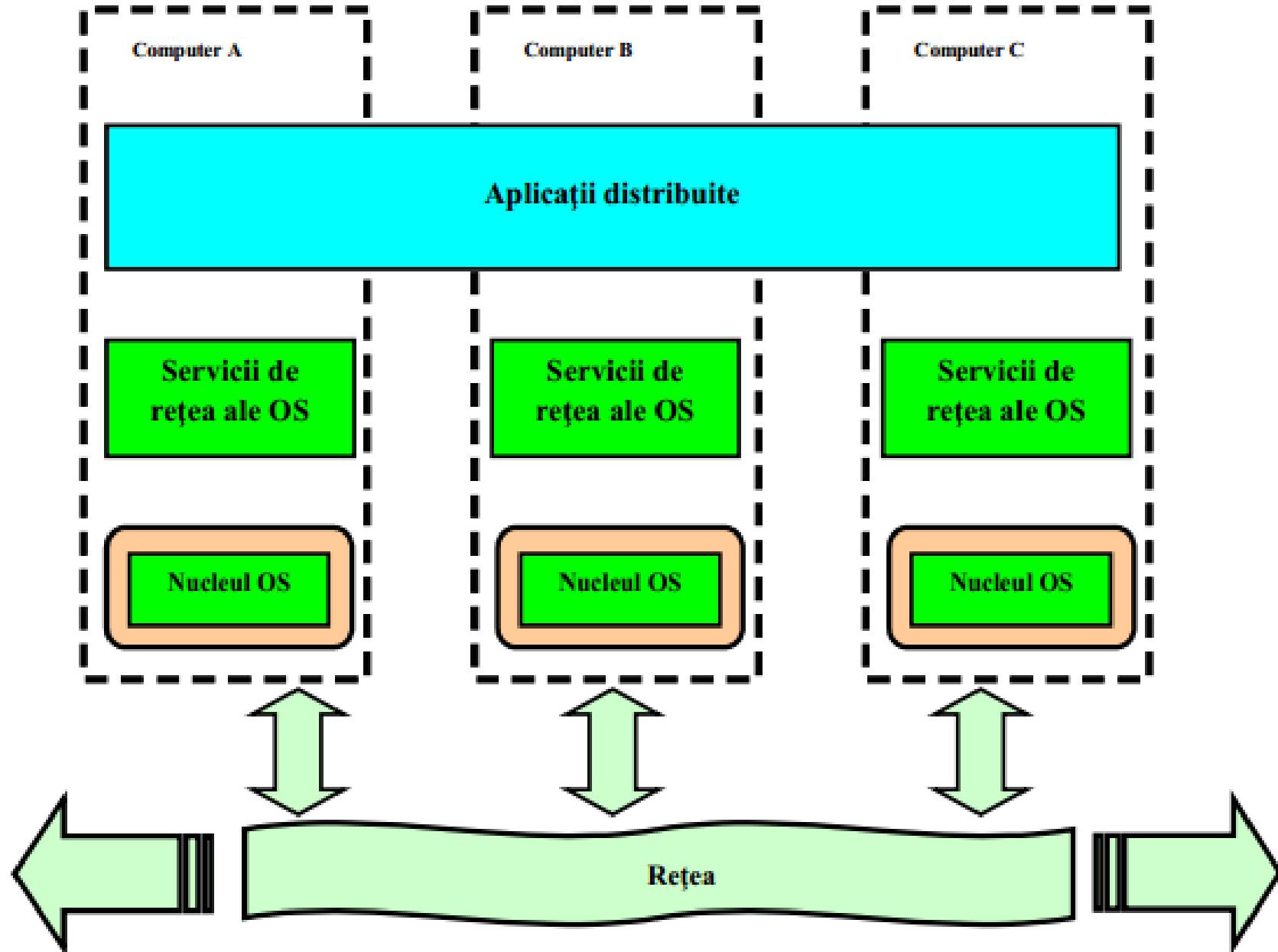
- Tanenbaum propune următoarea definiție:
- “*Un sistem este distribuit doar dacă existența nodurilor autonome este transparentă pentru utilizatorii sistemului – se comportă virtual ca un singur calculator*”.

- "You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done."
- (Leslie Lamport, Distribution email, May 28, 1987,
- Cum il inteleag aftomatistii?...

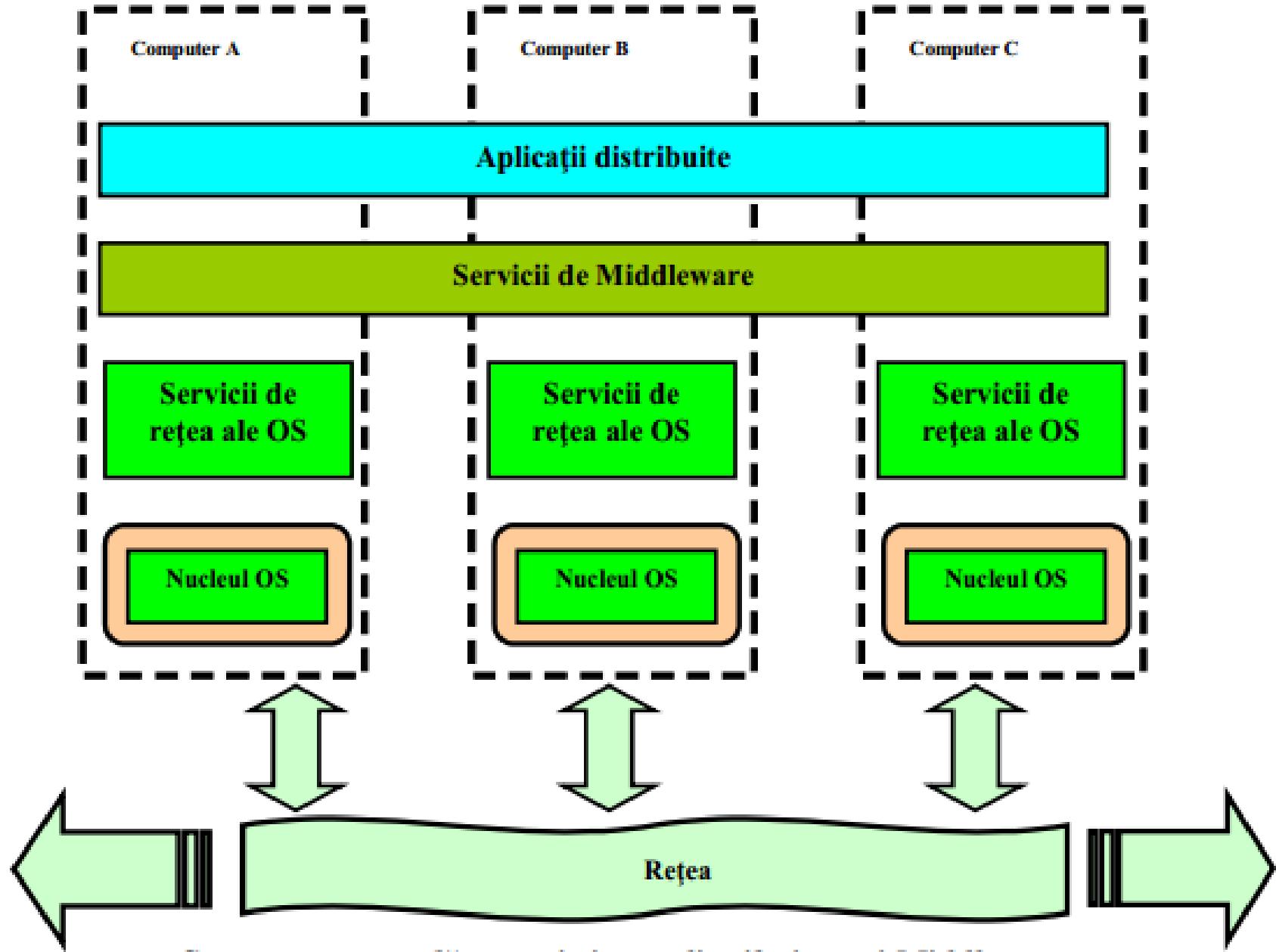
- **Sisteme de calcul distribuit** (*Distributed Computing Systems*)
 - ◆ Sisteme folosite pentru realizare de task-uri ce necesita putere mare de calcul
- Sisteme distribuite de informatii (*Distributed Information Systems*)



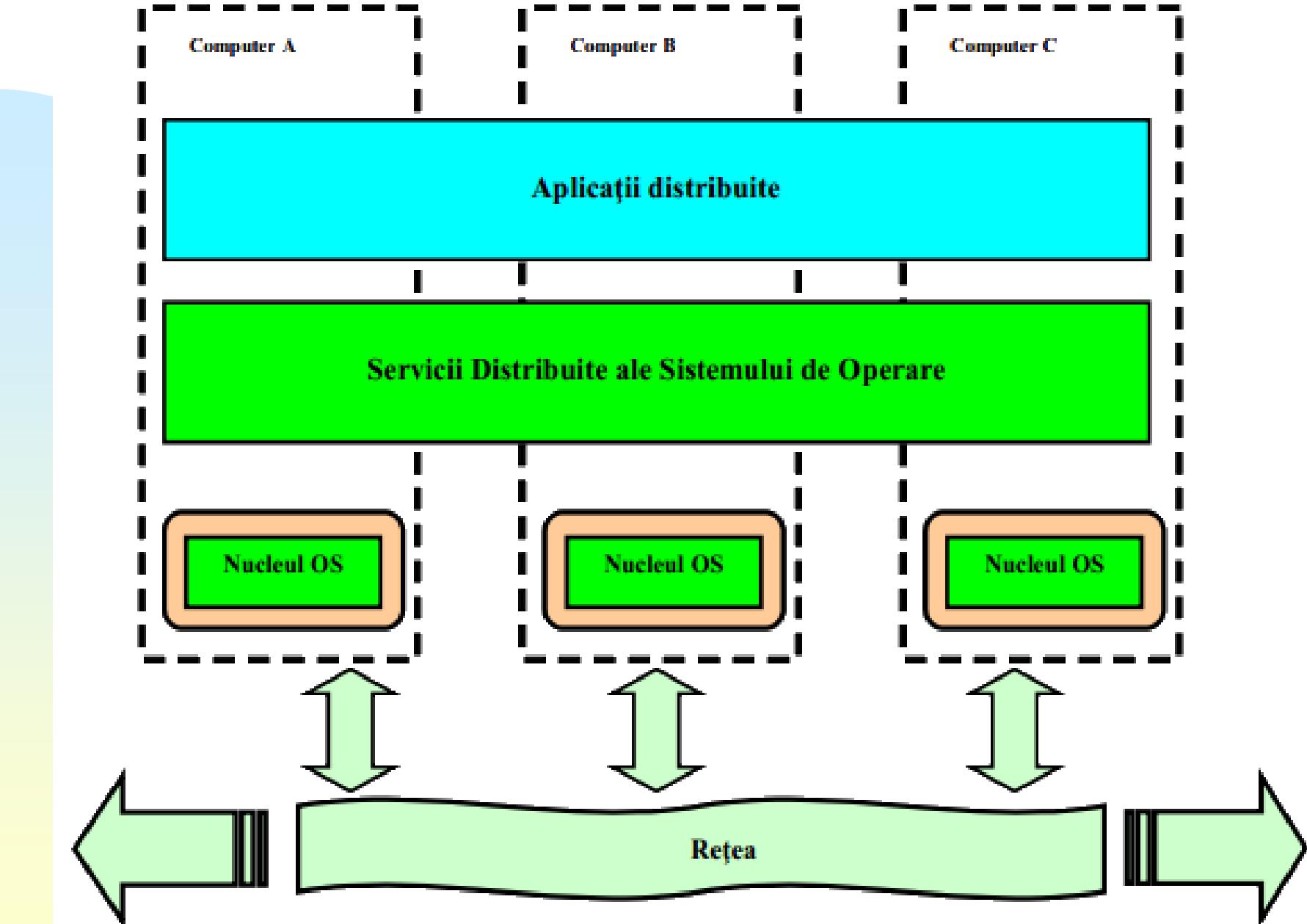




Structura generală a unui sistem de operare în rețea

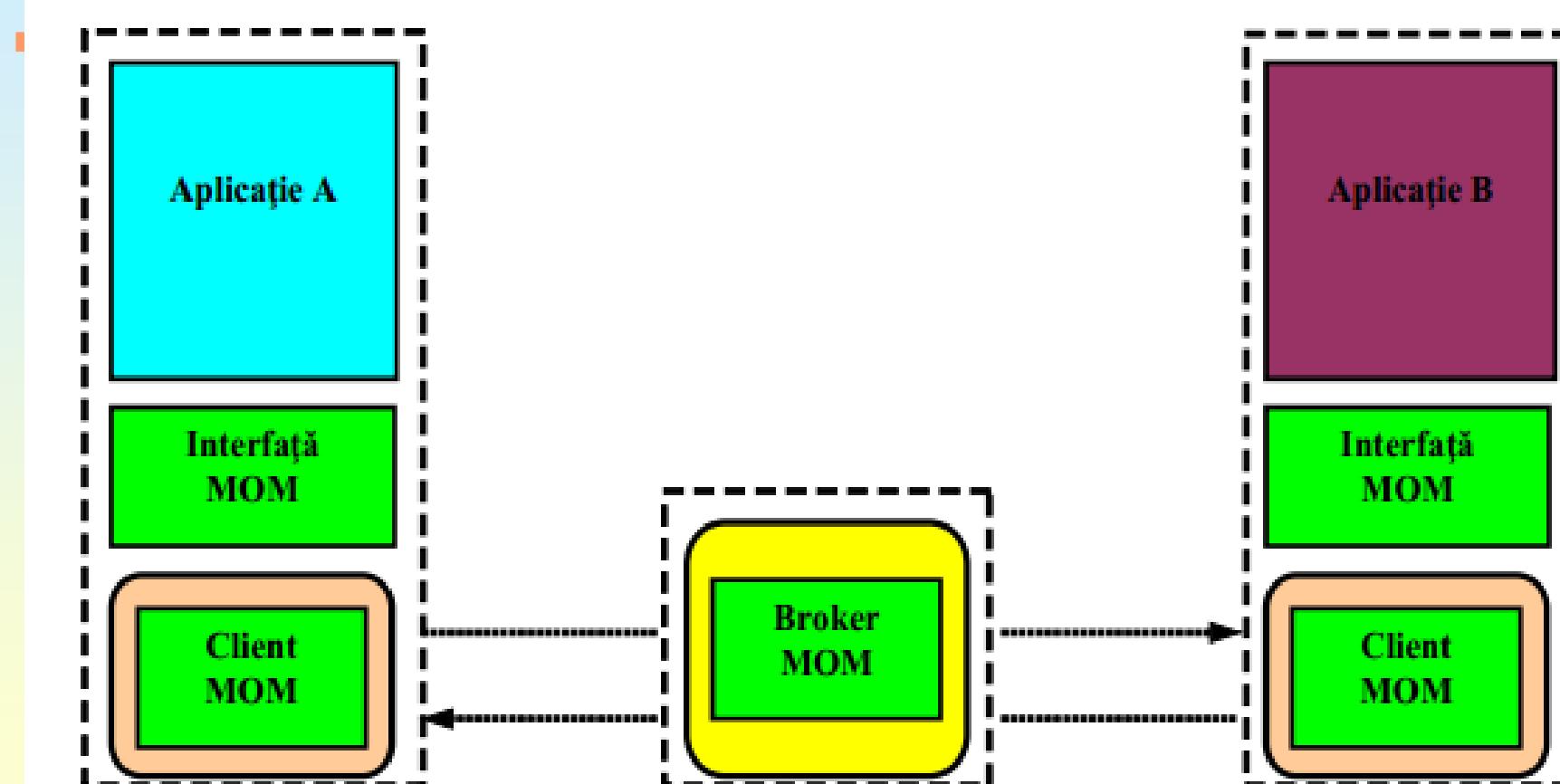


Structura generală a unui sistem distribuit ca și Middleware



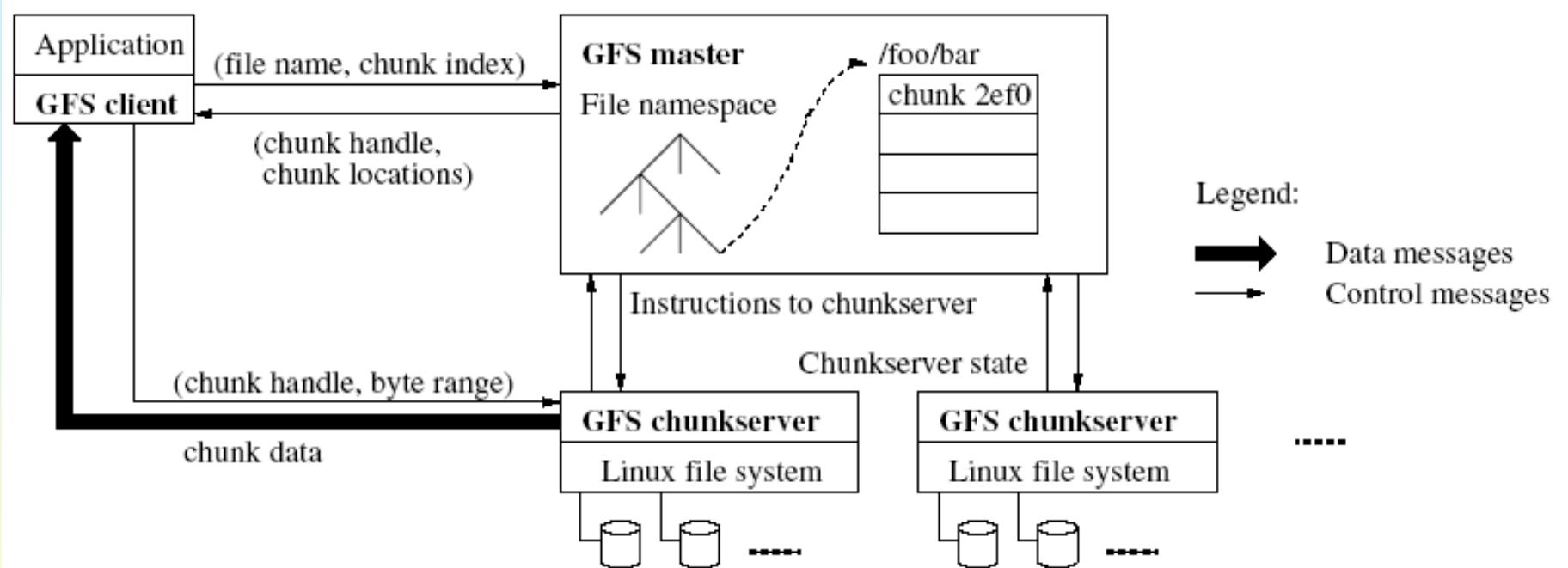
Structura de ansamblu a unui sistem de operare multicompiler

- Posibilitatea integrării unor sisteme eterogene este unul din principalele aspecte în care
- platformele de mesagerie joacă un rol cheie. Un Message Oriented Middleware (MOM)



Comunicarea între aplicații prin intermediul unui middleware orientat pe mesaje

GFS: Architecture (2)



Avantajele oferite de MOM sunt următoarele:

- posibilitatea integrării de componente eterogene, rulând pe platforme diferite, într-un sistem informatic distribuit;
- reducerea strangulărilor în sistem – decuplarea aplicațiilor, prin interschimbul asincron de mesaje,
 - ◆ dacă necesarul de prelucrare este mai ridicat pentru anumite componente, acestea pot, în mod dinamic, beneficia de multiple instanțe ale aceluiași proces;
- Scalabilitatea
 - ◆ Hw
 - ◆ sw

Avantaje

Costuri reduse

- S-a demonstrat economic că este mult mai ieftină să se folosească mai multe sisteme de calcul la rezolvarea unei probleme complexe, decât un supercalculator.
- Aceasta are un cost foarte mare și este dedicat, în general, numai unui flux continuu de probleme de calcul de foarte mari dimensiuni.
- În aplicațiile de control industrial apare o mare economie de cablu prin plasarea procesorului lângă sursa datelor, rezultatele putând fi vehiculate serial.

Avantaje

Modularitate și simplitate a softului

- Sistemele distribuite pot fi construite într-o manieră foarte modularizată, unde fiecare componentă permite interfațarea cu restul sistemului (celealte componente) sau servicii foarte bine definite.

Flexibilitate și extensibilitate

- Modularitatea sistemului permite schimbarea sau modificarea elementelor de calcul fără a deranja major operațiile în curs de desfășurare.

Avantaje

Fiabilitate și integritate

- Sistemele distribuite au proprietatea de a putea continua operațiunile indiferent de starea unei părți a sistemului.
- Folosirea mai multor noduri de calcul permite ca, în cazul defectării sau blocării unei resurse, sistemul să continue activitatea resursele critice, în acest caz, pot avea control dublu sau triplu.

Avantaje

Performanță

- Performanța este în general definită în termenii timpului de răspuns la un anumit grad de încărcare.
- Timpul de răspuns poate fi scăzut, în mod particular dacă majoritatea procesării este făcută local.

Dezavantaje

Lipsa cunoștințelor despre starea globală

- În fluxul de control al unui algoritm centralizat deciziile se pot lua în funcție de starea întregului sistem.
- Chiar dacă starea sistemului se poate deduce din valorile pe care le au mai multe variabile, aceasta poate fi determinată în mod corect, deoarece în procesul de inspectare a valorilor lor, nici una dintre variabile nu poate fi modificată de un alt proces.

Dezavantaje

Lipsa unui timp global

- Evenimentele care constituie executarea unui algoritm centralizat formează o mulțime total ordonată conform apariției lor temporale. Relația de ordonare temporală indusă asupra evenimentelor care constituie execuția unui algoritm distribuit nu este o relație de ordine totală.
- Există perechi de evenimente pentru care se poate decide care dintre ele s-a produs primul și care al doilea, dar acest lucru nu este valabil întotdeauna.

Dezavantaje

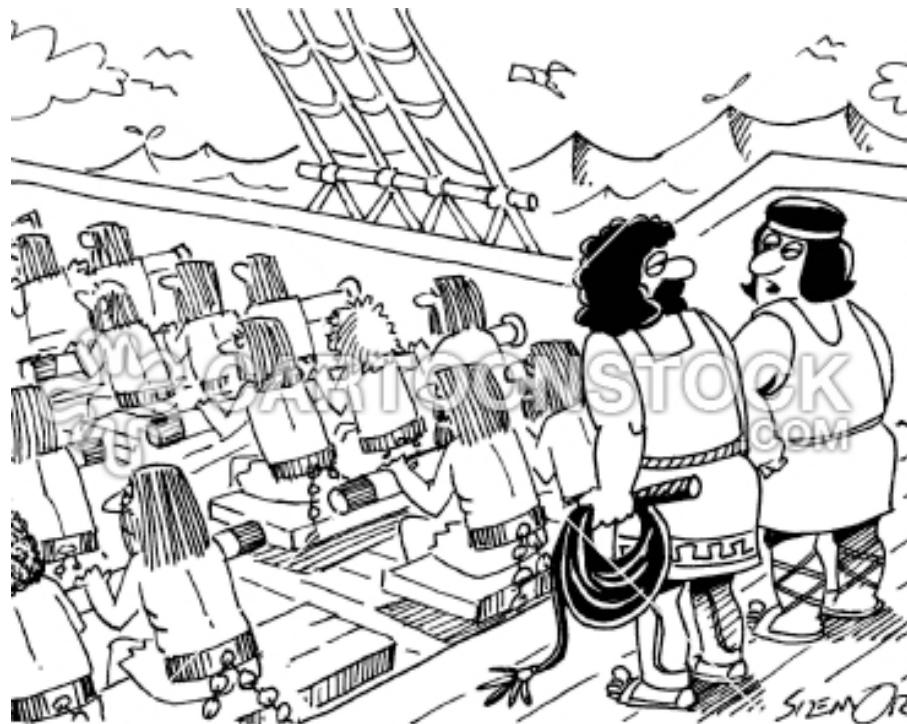
Nedeterminismul

- Comportamentul unui program centralizat poate fi descris în funcție de datele de intrare.
- Pentru un anumit set de date de intrare, comportamentul este identic.
- Spre deosebire de aceasta, comportamentul unei aplicații distribuite este de obicei nedeterminist.

Comunicațiile

Probleme specifice comunicațiilor

- Din punct de vedere al comunicației un sistem distribuit trebuie să se poată realiza:
 - ◆ Un schimb *sigur* de informații între noduri; această proprietate este asigurată prin ajungerea la destinație a mesajelor transmise, lipsa erorilor în mesaje ca și a copiilor multiple.



Centralised versus computing



Distributed Computing

Cursul 2

Model?
Modele?

Model

- Modelul unui sistem trebuie să răspundă următoarelor întrebări:
 - ◆ Care sunt principalele interacțiuni din sistem?
 - ◆ Cum interacționează acestea?
 - ◆ Care sunt caracteristicile care afectează comportamentul lor individual și global?

Model

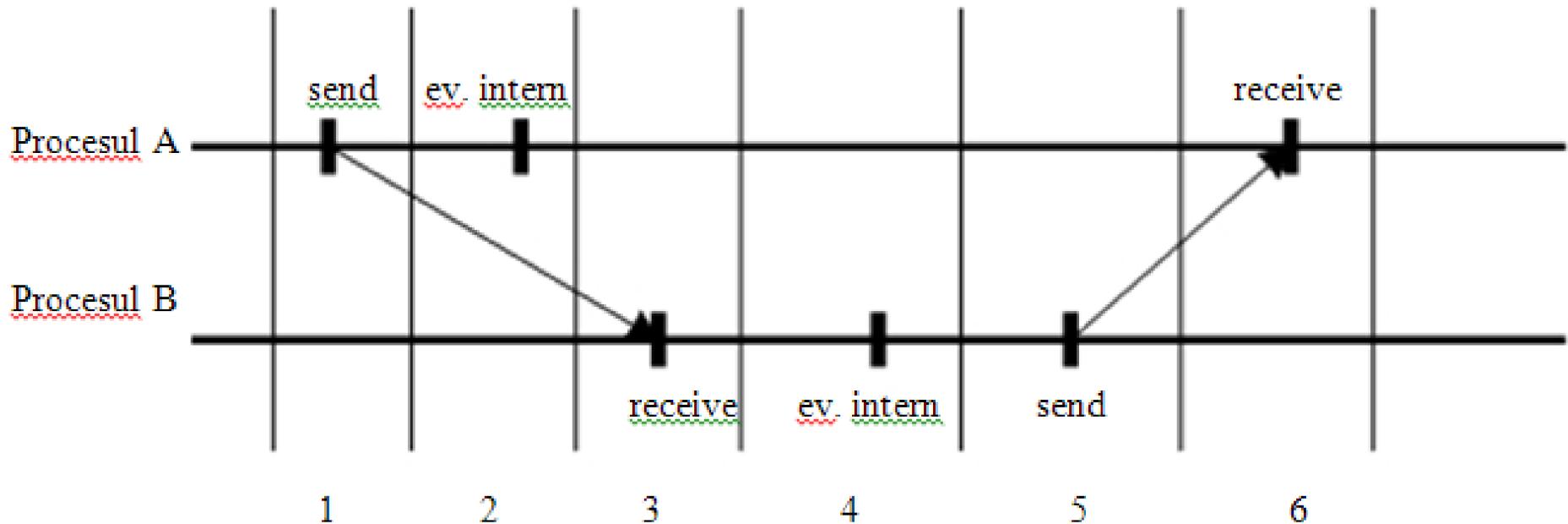
- În cazul particular al modelării sistemelor de calcul paralel și distribuit orice model din această categorie trebuie să includă:
 - ...
- Conceptul de proces combină activitatea acestuia cu protecția spațiului de memorie rezervat lui și are următoarele caracteristici:

Modelul ..

- 1. are propria stare care, de obicei, se referă la setul abstract de date aflat în procesare de acesta.
- 2. În modelul cu transfer de mesaje
- 3. În modelul cu memorie comună

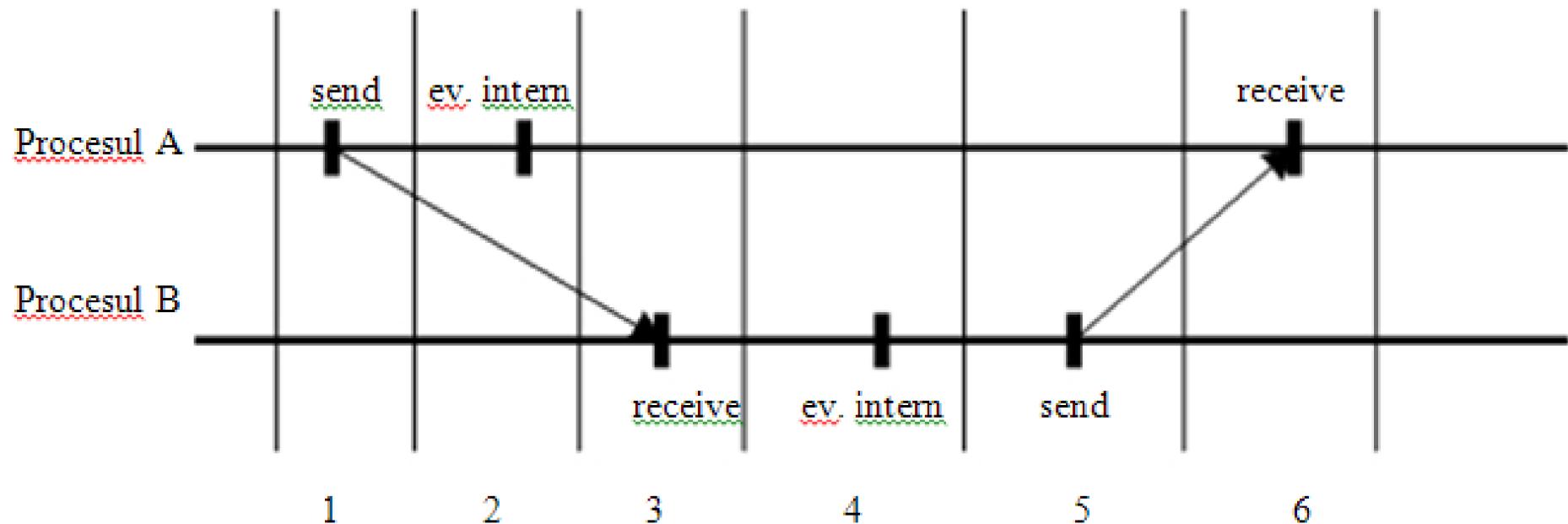
- Un sistem distribuit este format din entități (procese, procesoare, calculatoare) care comunică și care își modifică starea ca urmare a producerii unui eveniment intern, sau a unei operații de comunicație - emisie (send), recepție (receive).

sistemului distribuit ca sistem cu tranziții



- Sistemul distribuit parcurge lanțul de configurații
1, 2, 3, 4, 5, 6, 7,...

- În figura s-au trasat evoluțiile în timp a două procese, A și B, care aparțin aceluiași sistem distribuit.
- Ele sunt observabile din exterior, dar un proces nu poate să cunoască starea sistemului.
- Procesul A își schimbă starea ca urmare a producerii unor evenimente, în secvența emisie, intern și, apoi, recepție.
- Procesul B își modifică starea ca urmare a producerii evenimentelor recepție, intern și emisie.



- O configurație a sistemului constă din mulțimea tuturor stărilor proceselor componente și colecția mesajelor în tranzit. Astfel, pe figura , se poate observa că în configurația a sistemului, există un mesaj în tranzit, de la procesul A către procesul B.

- Evoluția în pași discreți a unui sistem distribuit sugerează adoptarea ca model a unui sistem cu tranziții.
- Aceasta este definit prin mulțimea configurațiilor sistemului, mulțimea configurațiilor inițiale, care este o submulțime a mulțimii configurațiilor și o relație de tranziție binară, definită și cu valori pe mulțimea configurațiilor.

- La un sistem cu tranziții ordonarea evenimentelor se face pe baza relației de cauzalitate și nu a unui timp global, care, aşa cum s-a precizat anterior, nu poate fi folosit din punct de vedere practic.
- Aceasta înseamnă că dacă, în cadrul aceluiași proces, un eveniment se produce înaintea altuia, se spune că îl precede cauzal.

Model cand

- Din punct de vedere al modelului anterior descris se observă că este perfect aplicabil, evident cu anumite diferențieri, atât în cazul calculului paralel cât și al celui distribuit.
- Din punct de vedere al schimbului de informații între elementele de calcul (care pot fi atât hard cât și soft) există două clase de modele: cu schimb de mesaje și cu memorie comună.

Model aplicații

- Există următoarele modele de organizare a aplicațiilor distribuite:
 1. *aplicații tip “client-server”:*
 2. *aplicații pe “cluster”:*
 3. *aplicații distribuite colaborative:*

Model aplicații

- ◆ “*metacalculul*”:
- Se observă că aceste categorii au apărut prin creșterea gradată a distribuției geografice, a numărului de utilizatori și a numărului de resurse implicate.

Model client server

- Este cel mai utilizat.
- Există procese specializate
- În timpul execuției diferite procese pot fi atât client, cât și server; de exemplu serverul de fișiere poate deveni client pentru serverul care oferă timpul curent.

Model client server

- Activitățile din timpul execuției unei aplicații distribuite sunt asociate, de obicei, abstractizării de proces și ele nu vor putea să existe decât ca părți ale aplicației.
- Problema care se pune sub acest aspect poate fi privită tot ca o problemă de descompunere și anume:
 - Ce reprezintă fiecare *activitate (proces)* într-o aplicație distribuită și cum se servește aceasta de *diferitele ei unități (obiecte)*?
 - Două tipuri de scheme de bază sunt curent întâlnite în organizarea activităților din aplicațiile distribuite: scheme procedurale și scheme modulare.

Model client server - RPC

Schemele procedurale - apel de procedură la distanță

- Schema procedurală este clasică și este cea mai răspândită în prezent pentru organizarea aplicațiilor.
- Adoptarea unei organizări procedurale în aplicațiile distribuite ridică însă o serie de aspecte noi, toate pornind de la faptul că distribuirea face inherentă cerința ca procesele să se poată “propaga” și la distanță, dincolo de granițele fizice ale unei singure mașini.

Model client server - RPC

- Trebuie observat că interacțiunile care au loc într-o organizare procedurală a proceselor sunt interacțiuni de tipul “master-slave”.
- Când este programată o interacțiune client-server, fiecare dintre procesele implicate, clientul și serverul, trebuie în esență să transmită fiecare câte o pereche de mesaje.

Model client server - RPC

- Din acest motiv apar complicații referitoare la realizarea transferului transparent la distanță al argumentelor de tip pointer și la apelurile prin referință. (Java why?)
- Sub aspectul sincronizării, RPC are prin tradiție un caracter sincron. Sunt posibile și forme mai generale de RPC, cum sunt RPC asincron și RPC de grup.

Model client server

Schema modulară, transfer explicit de mesaje

- Schema modulară sau cu activități “rezidente” prezintă o analogie evidentă cu arhitectura distribuită de bază, ceea ce permite o mare flexibilitate în descrierea interacțiunilor dintre procesele unei aplicații.
- Procesele au fiecare o memorie proprie și comunică între ele prin transfer explicit de mesaje.

Model client server

- Constrângerile rezultate asupra structurii obiectelor care servesc drept “rezidență” pentru procese sunt mai puternice decât schema procedurală, apărând probleme specifice de descompunere și control.
- Cerințele de modularitate și de distribuire fac în aşa fel încât procesele “găzduite” pe diferitele unități ale aplicației să nu disponă de mijloace隐含的 de comunicație, astfel că este definit și utilizat un mijloc neconvențional de comunicație, *mesajul*.

Model client server

- Diferențierea modelelor de transfer de mesaje poate fi făcută din perspective diverse cum ar fi:
 - ◆ schema de desemnare utilizată pentru specificarea sursei și a destinației;
 - ◆ momentul când se decide care sunt sursa și destinația unui mesaj;

Model client server

- Cea mai simplă schemă de desemnare este *desemnarea directă sau explicită*: sursa și destinația sunt specificate prin nume de procese.
- Schema este ușor de implementat și de utilizat, constituind o bună cale pentru implementarea algoritmilor de tip *pipeline*, deoarece fluxul informației este constant pe toată durata execuției programului distribuit.

Model client server

- Desemnarea prin nume globale este de regulă asociată conceptului de cutie poștală (“mailbox”) care permite mai multor procese să poată trimite mesaje către aceasta și, la rândul lor, mai multe procese să recepționeze mesaje din respectiva zonă.

Model client server

- O a doua cale de diferențiere a schemelor de transfer de mesaje este cea legată de momentul în care se decide sursa și destinația unui mesaj.
- Dacă sursa și destinația se fixează în momentul compilării, se spune că se realizează o desemnare *statică* a canalului logic.

Model client server

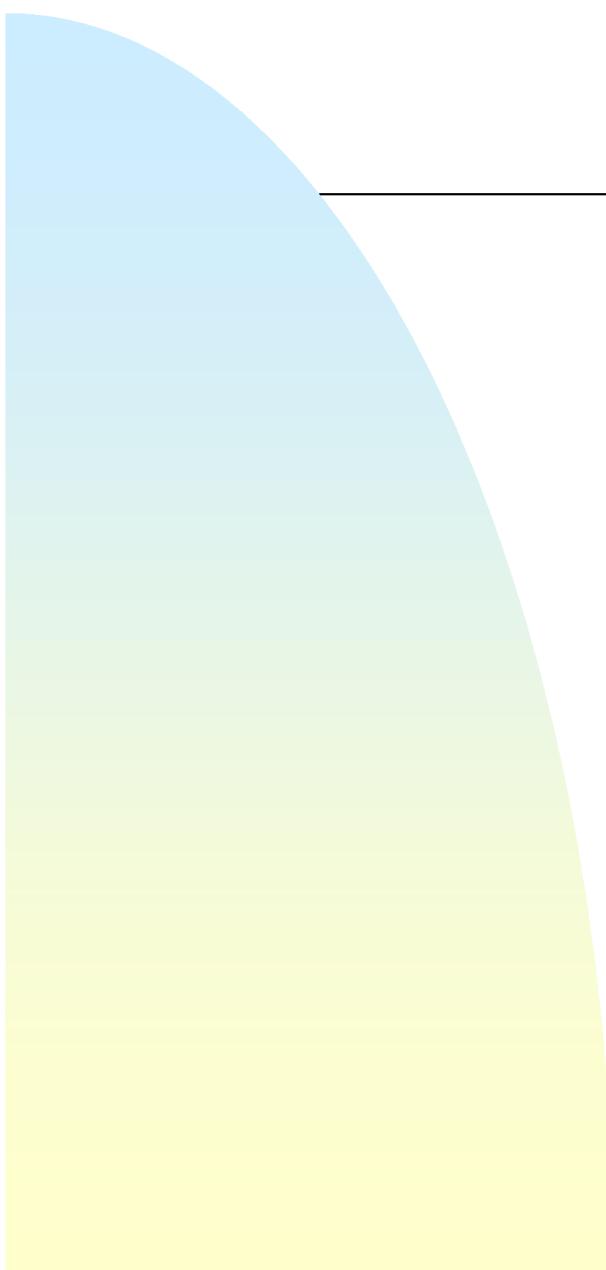
- Un alt aspect important, utilizat pentru diferențierea modelelor de transfer de mesaje, îl constituie structura de comunicație facilitată de model. Pot exista mai multe alternative:
 - ◆ structuri de tip “unu la unu”;
 - ◆ structuri de tip “mai mulți către unu”;
 - ◆ structuri de tip “mai mulți către mai mulți”;
 - ◆ structuri de tip “unu către mai mulți”.

Model client server

- Structurile de comunicație “mai mulți către unul” sunt utile în special dacă procesele comunicante cooperează în maniera client-server, mai mulți clienți pot beneficia de serviciile unui singur server.
- Structurile de tip “unul către mai mulți” sunt tipice pentru comunicații de grup (“multicast”), în particular “broadcast” la nivelul aplicațiilor distribuite.

Model client server

- Un sistem client-server este realizat pe trei nivele, cel al prezentării, cel al aplicației și cel al accesului la date.
- Separarea fizică a celor trei nivele se numește partiționarea aplicației. Întotdeauna nivelul prezentării va fi implementat de către client.
- Împărțirea celorlalte două nivele între client și server conduce la cinci stiluri client - server diferite:
- Multilayer multitier - comments



Gestiunea Resurselor

Alocarea resurselor

- În general, funcția de alocare a resurselor sistemului nu este încredințată unui singur locator, ci este distribuită între mai multe locatoare, situate pe elemente de prelucrare distribuită.
- Pentru implementarea schemelor de alocare, sunt necesare informații privind starea resurselor existente în sistem.

Alocarea resurselor

- Există două clase mari de soluții privind gestiunea reprezentărilor de stare de care au nevoie procesele dedicate alocării de resurse.
- Una dintre ele este menținerea unui catalog unic.
- În acest caz o reprezentare unică a stării resurselor este partajată între toate procesele dedicate

Alocarea resurselor

- O a doua modalitate este distribuirea catalogului stării.
- Aceasta poate fi distribuit în întregime sau parțial. În prima variantă, catalogul este replicat pe fiecare element de prelucrare, situație în care este necesară menținerea consistenței copiilor multiple ale catalogului de stare.
- În a doua variantă, pe fiecare element de prelucrare este întreținut un catalog al stării resurselor locale.

Alocarea resurselor singulare

- Alocarea unei resurse singulare localizată pe un element de prelucrare al sistemului poate fi făcută pentru a fi posibil un acces exclusiv sau concurrent.
- Accesul exclusiv la o resursă singulară poate fi implementat de un locator (numit server), care este însărcinat cu gestiunea respectivei resurse.

Alocarea resurselor singulare

- Serverul S poate fi programat să implementeze orice strategie referitoare la satisfacerea cerințelor:
 - ◆ controlul drepturilor de acces,
 - ◆ tratarea cererilor într-o anumită ordine
 - ◆ etc.
- Această strategie nu poate asigura nici o formă de paralelism la accesare

Alocarea resurselor singulare

- Implementarea acestei scheme de alocare poate fi făcută în două moduri:
 - cererile de la procesele solicitante
 - cererile de acces

Alocarea resurselor multiple

- Execuția unei aplicații compusă din mai multe procese ce pot fi executate în paralel, poate genera cereri de acces la o multitudine de resurse, posibil localizate pe elemente de prelucrare distincte.
- tranzactia

Alocarea resurselor multiple

- Cazul accesului la resurse multiple poate fi privit și din punct de vedere al concurenței apare problema creării proceselor concurente.
- Crearea proceselor concurente se realizează prin detectarea secvențelor din program care sunt independente, unele față de celelalte.

Alocarea resurselor multiple

- Relația de ordine este indicată prin dependență de date:
 - dependență de curs (“flow dependence”)
 - antidependență (“antidependence”)
 - dependență de ieșire (“output dependence”)
 - dependență I/E (“I/O dependence”)
- O relație de condiționare suplimentară este dictată de ordinea de execuție.

Alocarea resurselor multiple

- Fiind date două procese, P1 și P2, cu datele de intrare I1 , respectiv I2 și datele de ieșire O1, respectiv O2, ele se pot executa concurrent dacă sunt independente și nu creează rezultate confuze (condițiile lui Bernstein):

$$I1 \cap O2 = \emptyset, I2 \cap O1 = \emptyset \text{ și } O1 \cap O2 = \emptyset$$

- *Execuția concurentă a celor două procese produce aceleași rezultate, indiferent dacă sunt executate secvențial, în orice ordine, sau în paralel.*

Alocarea resurselor multiple

- Printre resursele cele mai importante ale sistemului cu prelucrare distribuită se numără procesoarele.
- Gestiunea lor adecvată este cheia obținerii unor performanțe corespunzătoare obiectivelor urmărite de sistem.
- O soluție ar fi prelucrarea arborescentă, această metodă nu este complet distribuită, dar funcționează bine în practică și este și tolerantă la erori.

Alocarea resurselor multiple

- Procesoarele executante se află numai pe ultimul nivel al arborelui pentru a crește toleranța la defecte: la defectarea un master mai există altă ramură cu un master disponibilă.
- În vederea satisfacerii cererilor de alocare, fiecare procesor conducător întreține un catalog cu informațiile de stare relativ la toate procesoarele pornind de la el în jos pe arbore.

Alocarea resurselor multiple

- Dacă acesta constată că numărul de procesoare subordonate este insuficient pentru realizarea calculelor, atunci el trimite cererea către procesorul superior lui.
- Dacă nici acolo nu sunt suficiente capacitateți la dispoziție, cererea se va propaga pe arbore însus până la satisfacerea ei, din acel moment alocarea proceselor la procesoare devenind efectivă.

Alocarea resurselor multiple

- Alocarea se realizează conform unui algoritm de tip licitație.
- În momentul în care un server primește mesajul cerere, acesta răspunde cu un mesaj ce conține condițiile în care poate satisface cererea (costul serviciului său), în funcție de încărcarea resursei gestionate de el.
- Solicitantul alege oferta cea mai convenabilă și anunță ofertantul sau oferanții respectivi.

Problema excluderii mutuale

- Este în specială sistemelor care folosesc zone de memorie comună pentru a realiza comunicarea între procese.
- Înainte de a accesa o zonă de memorie comună un proces trebuie să aibă asigurată atât excluderea mutuală cât și coerența datelor.
- Mono vs cluster

Abordarea centralizată

- O posibilitate este simularea metodelor specifice unui sistem monoprocesor.
- În acest caz se alege un proces ca fiind coordonator.
- Principiile de alegere a mașinii gazdă pentru acesta sunt destul de variate, totuși o metodă destul de răspândită este plasarea lui pe sistemul cu cea mai mare adresă din rețea.
- În momentul apariției unei noi cereri se testează dacă aceasta este vidă, caz în care respectivul proces primește permisiunea folosirii zonei.

Abordarea centralizată

- În caz contrar cererea se inserează în coadă iar procesul este anunțat. În acest caz procesul poate intra automat în aşteptare sau își poate continua lucrul.
- Deși abordarea centralizată suferă de problema supraîncărcării la nivelul procesului de control precum și de o rată scăzută a toleranței la defecte deoarece toată aplicația este blocată dacă procesul conducător cade, ea prezintă următoarele avantaje:

Abordarea distribuită Ricart și Agrawala

- 1981.
- Metoda pornește de la premisa existenței unei ordini totale a tuturor evenimentelor din sistem.
- Când un proces dorește accesul într-o regiune critică el va crea un mesaj care va conține identifierul propriu, numele zonei de acces precum și timpul la care s-a realizat respectiva cerere.

Abordarea distribuită Ricart și Agrawala

- În cazul recepționării unui astfel de mesaj tratarea lui este realizată funcție de regiunea critică cerută.
- Dacă procesul receptor nu accesează momentan respectiva zonă atunci va trimite un mesaj de validare.
- În caz contrar cererea va fi introdusă într-o coadă de așteptare.

Abordarea distribuită Ricart și Agrawala

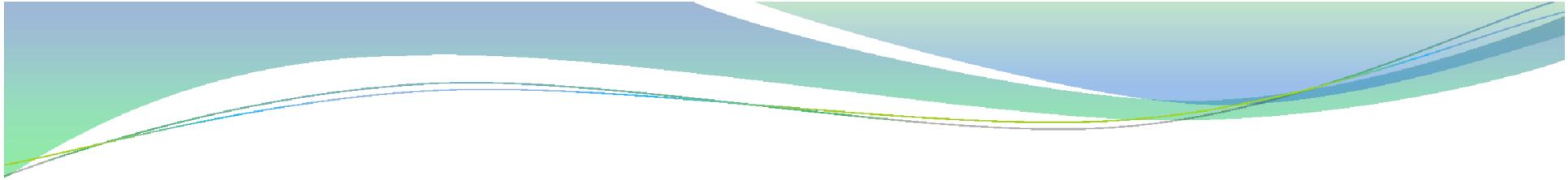
- La ieșirea din zonă acest proces va trimite tuturor un anunț care va debloca cozile de aşteptare algoritmul reluându-se.
- Și acest algoritm este fără blocaje.
- Dar este mult mai puțin tolerant la defecte deoarece acum există n puncte critice. Problema poate fi corectată cu prețul dublării numărului de mesaje.
- Dacă un proces nu este de acord el va trimite această înștiințare către cel care a generat cererea.

Abordarea distribuită algoritm pe inel

- Datorită problemelor legate de asigurarea unei ordini temporale s-au încercat și alte abordări care să nu țină cont de aceasta.
- Pentru fiecare resursă critică se va asocia un permis de acces unic.
- Procesele care folosesc această resursă sunt organizate într-un inel.
- Ordonarea proceselor poate fi realizată pe baza adresei mașinii de origine.

Abordarea distribuită algoritm pe inel

- Dacă are nevoie o va folosi și la terminare va trimite mai departe permisul.
- În caz contrar permisul va fi trimis imediat la vecinul din listă.
- Funcție de direcția de parcurgere a listei vecinul va fi cel din dreapta sau cel din stânga.
- Detectarea pierderii permisului este dificilă deoarece nu este prevăzută o cantă temporală fixă pentru folosirea respectivei resurse critice.

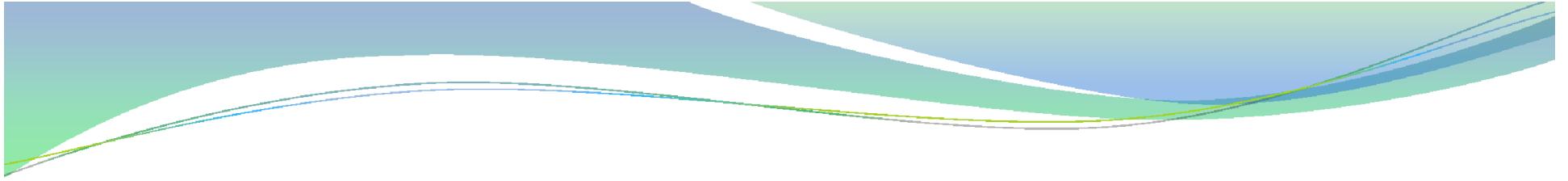


Sisteme distribuite de fisiere

Cursul 3

Intro

- Stocarea distribuită, uzual sub formă de fișiere, a devenit rapid importantă o dată cu succesul arhitecturilor de tip rețea și stații de lucru.
- Dacă fiecare stație necesită spațiul propriu de stocare, atunci schimbul de date este minimal.
- Oricum nu acesta este cazul într-un sistem distribuit.



- O dată ce utilizatorilor le este permisă conectarea la orice mașină și schimbul liber de fișiere și se așteaptă ca mediul de lucru să nu fie alterat atunci modelele clasice de rețele nu mai sunt viabile.
- Soluțiile pentru aceste probleme se află în sistemele de stocare distribuite.
-

Caracteristici - Acces transparent

- Ideal, un DFS ar trebui să apară clienților ca pe un sistem convențional, centralizat, de fișiere.
- Multiplicitatea și dispersia serverelor și a dispozitivelor de stocare ar trebui să fie transparentă.
- Asta înseamnă că interfața client a unui DFS nu ar trebui să facă diferență între fișiere locale și remote.

Datele comune

- Un spațiu de stocare distribuit trebuie să satisfacă un număr de cerințe generale:
- Este necesară asigurarea unei metode de a avea acces în comun la o zonă de date între mai multe aplicații.
- De obicei aceasta se rezumă la un acces comun la citire unde copiile primite de aplicații sunt manipulate independent.

Transparentă locației

- Localizarea fizică propriu zisă a datei trebuie să fie ascunsă pentru programator aceasta neavând o poziție neapărat stabilă.
- Este evident că această mobilitate trebuie să fie supusă restricțiilor implicite de securitate
- Există mai multe moduri de numire a schemelor DFS.

Controlul distribuit

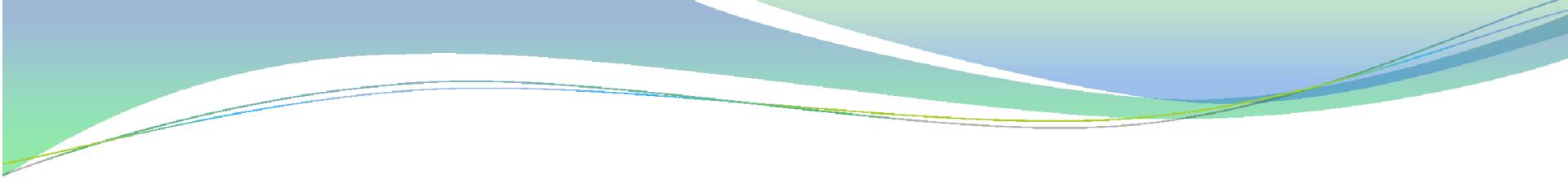
- Trebuie ca să nu existe o structură unică de control pentru respectivul spațiu.
- Dacă acest aspect nu pune probleme deosebite la structurile de dimensiuni mici și medii în cazul sistemelor mari, cum ar fi zone comune de stocare între corporații diferite, poate deveni problematică.
- De obicei controlul datelor se preferă a rămâne separat peste o anumită dimensiune.

Securitatea

- Rețelele sunt nesigure.
- Chiar dacă li se poate garanta integritatea în sisteme mici, sistemele mari care folosesc rețele accesibile publicului sunt sensibile la atacuri.
- Deci este necesară protecția datelor confidențiale împotriva accesului neautorizat și protecția sistemului împotriva simulării unui utilizator și mașină recunoscute de acesta

Stabilitate și fiabilitate

- Un astfel de sistem trebuie să facă față căderilor ce pot apărea la mașinile din rețea.
- Căderile trebuie acceptate/tolerate fără ca datele să fie afectate.
- De asemenea este de dorit ca problemele apărute să poată fi gestionate dinamic



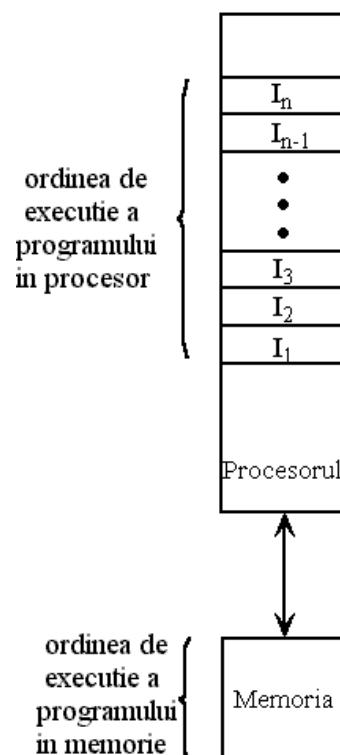
Folosirea tehnicielor de cache

- Cererile de acces ale unui fișier remote sunt realizate prin două metode complementare.
- Cu servicii remote, cererile de accesare sunt livrare serverului.
- Serverul le accesează și rezultatul e trimis clientului.

Atomicitate și ordinea evenimentelor

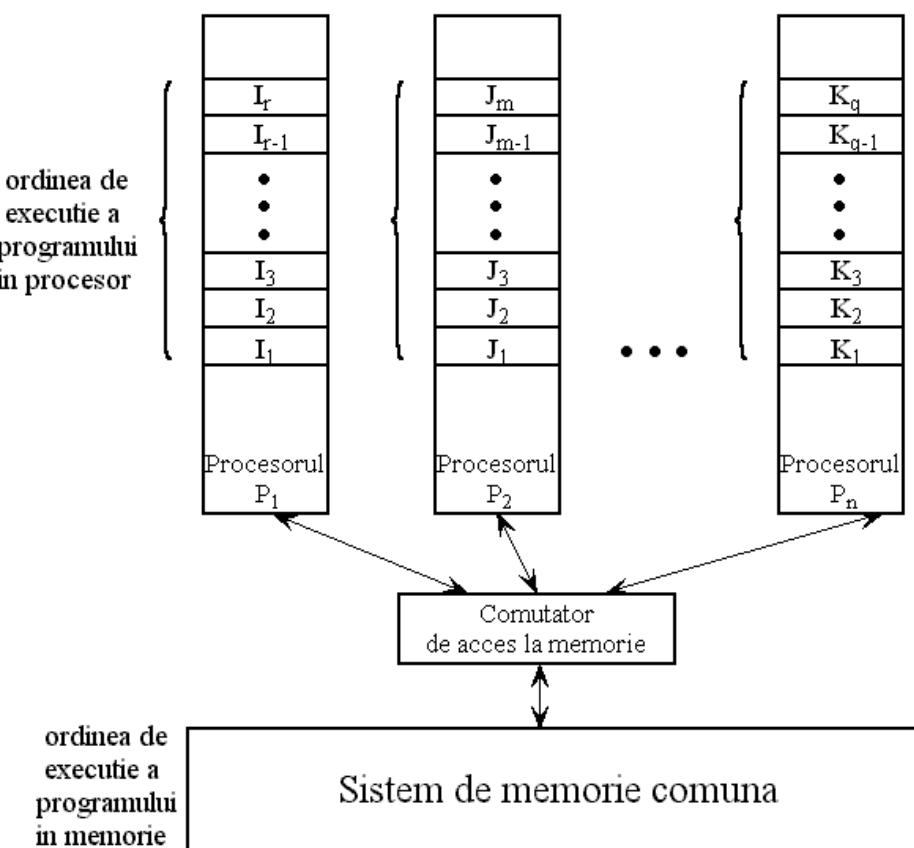
Problema inconsistenței la nivelul memoriei poate apărea atunci când accesele la memorie ajung să apară în altă ordine decât cea specifică execuției programului

Executia in cazul SISD



a)

Executia in cazul MIMD



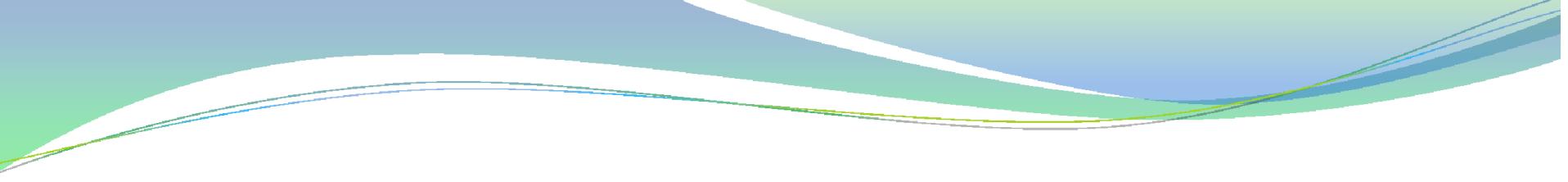
b)

- Modul în care este observat de către procesor comportamentul memoriei comune într-un sistem multiprocesor care implementează acest sistem, se numește model de memorie.
- Specificațiile de proiectare ale unui astfel de model trebuie să răspundă la următoarele întrebări fundamentale:
 - Comportament
 - Ambiguitati
 - general

- Ordinea programului este ordinea în care apar accesele la memorie în cazul execuției unui singur proces în cazul în care nu s-a realizat vreo operație de control superior (nivel sistem de operare) asupra execuției programului respectiv.

Dubois a definit în 1986 trei operații primitive de memorie ce pot fi folosite pentru a putea defini modelele de consistență a memoriei:

Se consideră a fi realizată ***o operațiune de citire*** de către procesul P_i , ținând cont de procesul P_k , numai în momentul în care acesta încearcă să realizeze o scriere în aceeași zona iar rezultatul acestei scrierii nu afectează valoarea citită de P_i .



- Se consideră realizată ***o operațiune de scriere*** de către procesul P_i , ținând cont de P_k , numai în momentul în care o cerere de citire realizată de acesta, la aceeași locație, întoarce valoarea depusă la scriere.
- ***O citire*** se consideră a fi valabilă la ***nivel global*** dacă este realizată ținând cont de toate procesele și dacă scrierea în acea zona a fost de asemenea realizată ținând cont de toate procesele.

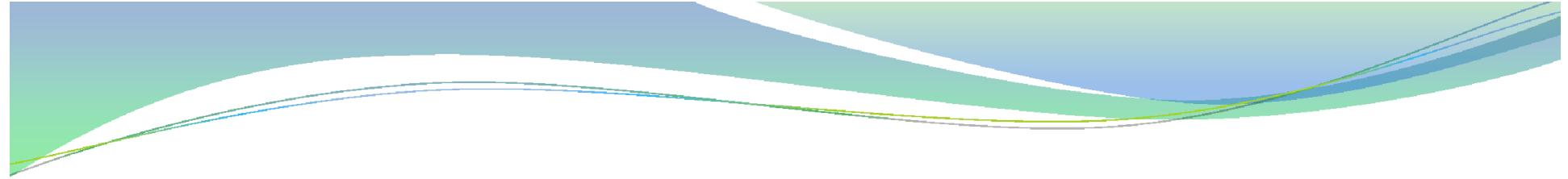


Modele de memorie specifice arhitecturilor multiprocesor scalabile cu memorie comună distribuită



Modelul consistenței procesorului

- A fost introdus de Goodman în 1992 și se referă la faptul că operațiunile de scriere inițiate de fiecare procesor respectă întotdeauna ordinea program.
- Totuși ordinea operațiilor de scriere inițiate de două procesoare poate să nu respecte ordinea programului.
- Cu alte cuvinte consistența la scriere este observată de fiecare procesor dar ordinea citirilor de la fiecare procesor nu este restricționată atâtă timp cât nu implică alte procesoare.



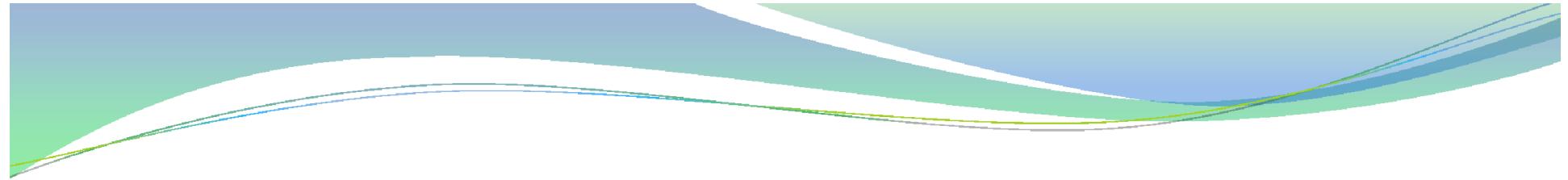
- 1. Înainte de a permite o citire, ținând cont și de celelalte procesoare, trebuie ca toate accesele de citire anterioare să fi fost terminate.
- 2. Înainte de a permite o scriere, ținând cont și de celelalte procesoare, trebuie ca toate accesele anterioare de citire sau scriere să fi fost terminate.
- De fapt acest model permite ca operațiile de citire care urmează unei operații de scriere să evite operația de scriere.

Consistență relaxată ("Release consistency")

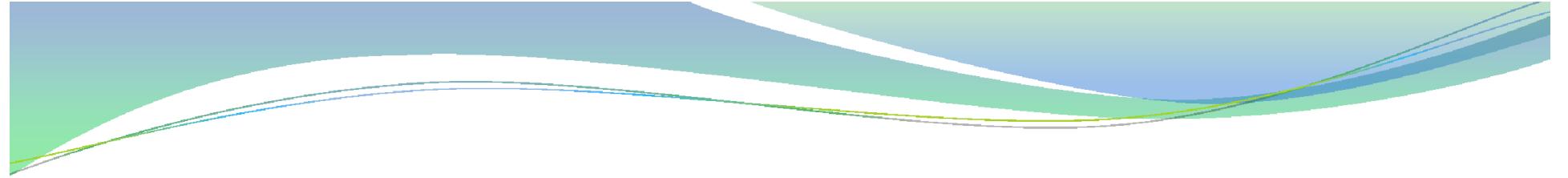
Unul dintre cele mai lejere modele de memorie a fost propus de Gharachorloo în 1990.

Acesta necesită ca accesele pentru sincronizare dintr-un program să fie identificate și clasificate ca rechiziționate ("acquire" - cum ar fi o operațiune de "lock") sau eliberate ("release" - cum ar fi "un-lock").

O operație de rechiziționare este o operație de citire care capătă accesul la un set de date în timp ce eliberarea este o operație de scriere care dă o astfel de permisiune (de acces pentru citire la un set de date).

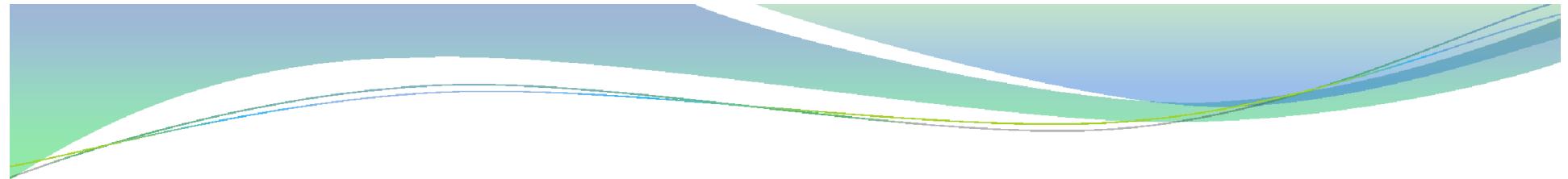


- 1. Înainte de execuția unei operațiuni obișnuite de citire sau scriere trebuie ca toate accesele anterioare de tip rechiziționare să fi fost realizate.
- 2. Înainte de execuția unei operațiuni de eliberare trebuie încheiate toate operațiunile obișnuite de citire și scriere.
- 3. Accesele speciale sunt consistente numai la nivel procesor unul față de celălalt.



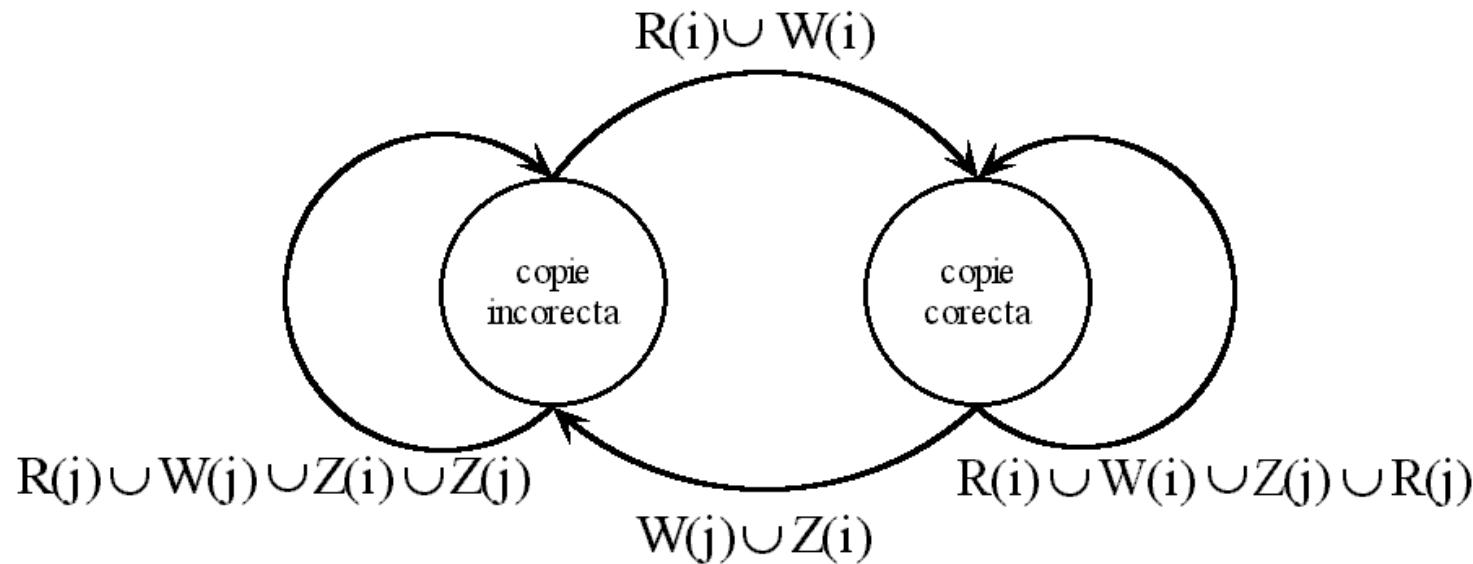
Problema coerenței cache-ului

- Într-o ierarhie de memorie specifică unui sistem multiprocesor pot apărea inconsistențe la nivelul datelor atât la nivele adiacente de memorie cât și în cadrul aceluiași nivel.
- Exemplul tipic este cel dat de copiile locale din cache față de cele din zona din memoria comună.
- În momentul în care ne referim la aspecte legate de asigurarea coerenței la nivele ce implică memoria cache apare o problemă referită ca problema coerenței cache-ului.



Protocol cu scriere imediată (“Write-Through”)

- În acest caz starea copie din cache se schimbă ținând cont de operațiunile de citire, scriere sau înlocuire.
- În figura sunt prezentate cele două stări posibile ale unui bloc din cache care poate fi o copie a blocului din memoria comună și poate afla în două stări: valid sau invalid în cazul folosirii tehnicii de scriere imediată.
- Fie un alt procesor j (cu $j \neq i$) care încearcă să lucreze cu respectivul bloc. Toate procesoarele pot realiza citirea ($R(i)$, $R(j)$) fără probleme.



$R(i)$ = Procesorul i citeste din cache-ul curent

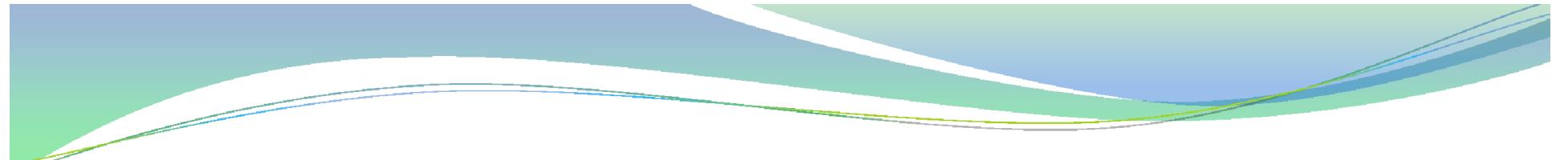
$W(i)$ = Procesorul i scrie in cache-ul curent

$Z(i)$ = modificare din exterior al cache

$A(j)$ = actiune facuta de procesorul $j \neq i$ in cache j , $A=\{R,W,Z\}$

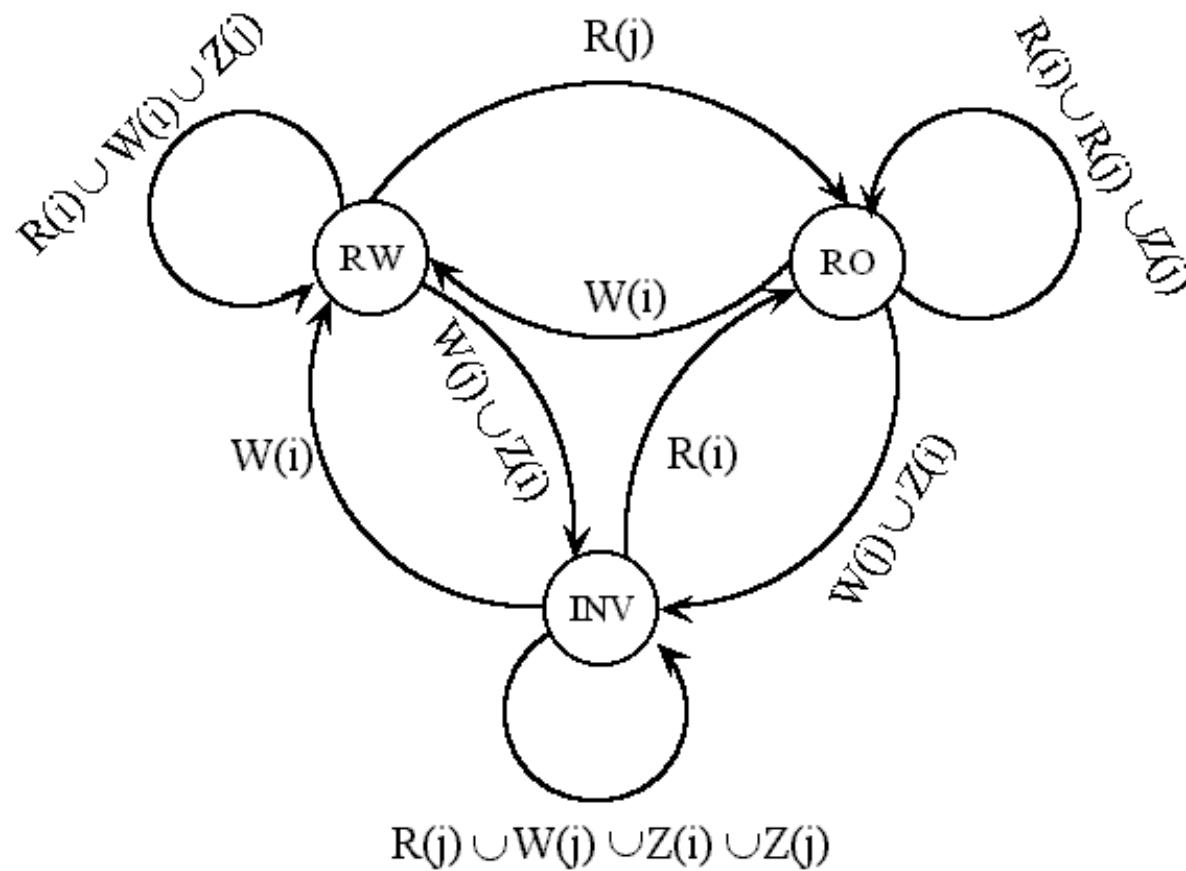
Protocol cu scriere imediată

- De fiecare dată când un alt procesor scrie $W(j)$ în copia lui din cache toate celelalte copii sunt invalidate.



Protocol cu scriere întârziată (“Write-Back”)

- În acest caz starea validă a unui cache care folosește această politică poate fi împărțită în două stări distincte una de scriere-citire (RW) și una numai de citire (ROnly) după cum se observă în figura .
- Starea invalidă (INV) corespunzătoare cazului “dirty” sau când copia este incorrectă sau nu există este echivalentă cu starea invalidă de la modelul menționat anterior.
- Această schemă este specifică unui protocol de proprietate (în sensul controlului momentan absolut “ownership protocol”).

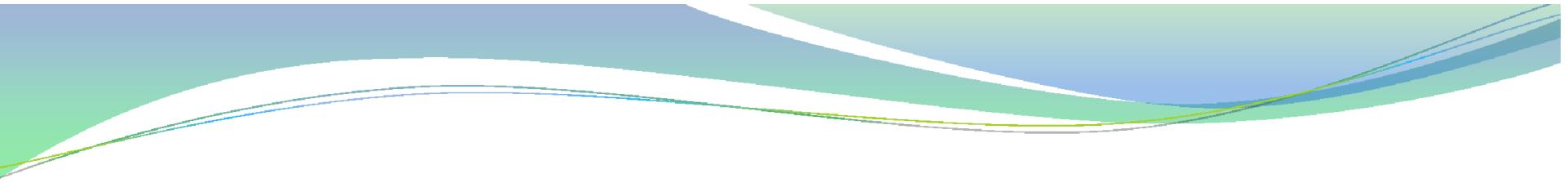


RO = Read Only

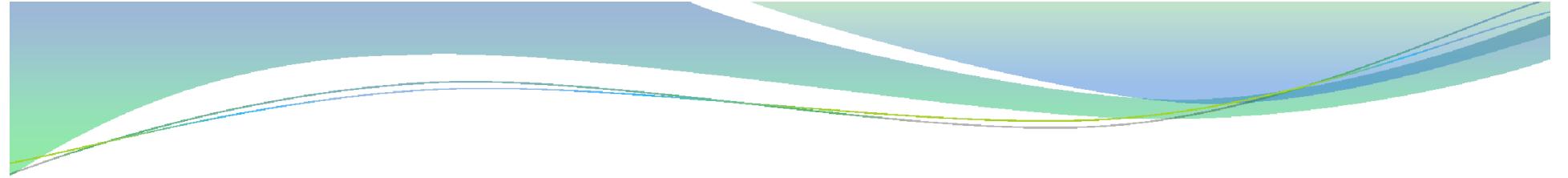
RW = Read - Write

INV = Invalidat sau nu există în cahe

Protocol cu scriere întârziată

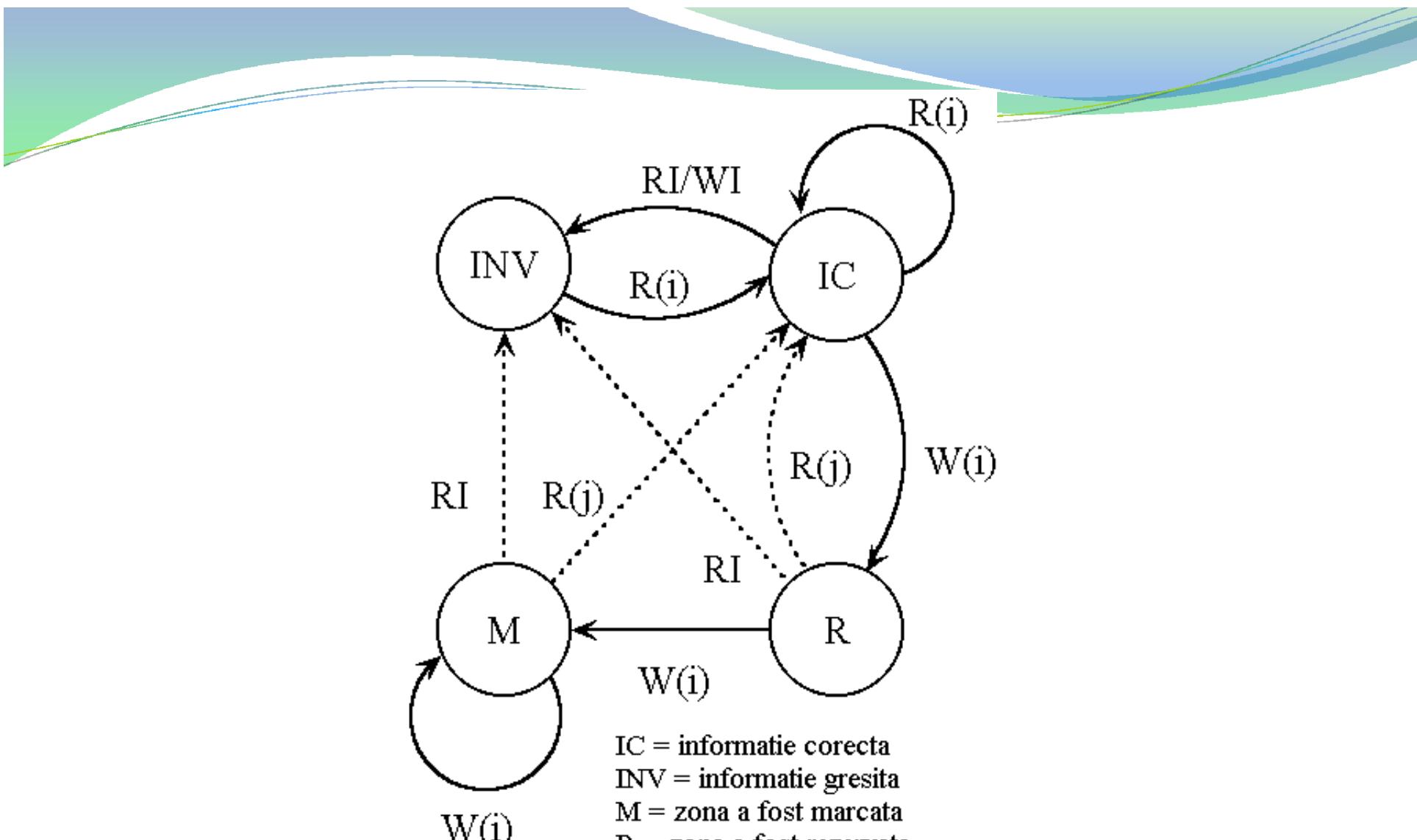


- În starea INV se intră numai când un alt procesor își modifică $W(j)$ copia locală sau își înlocuiește $Z(i)$ acea copie. Starea RW corespunde situației în care numai această copie (de pe procesorul i) există în sistem.
- Este evident că sunt permise numai operațiunile de locale de scriere sau citire ($R(i)$, $W(i)$).
- Înainte ca un bloc să poată fi modificat trebuie obținut controlul pentru acces exclusiv prin lansarea unei operațiuni de tip RO pe magistrală în scopul anunțării restului de procesoare, prin intrarea cache-urilor locale în starea RO, despre preluarea controlului.



Protocol cu scriere unică ("Write Once")

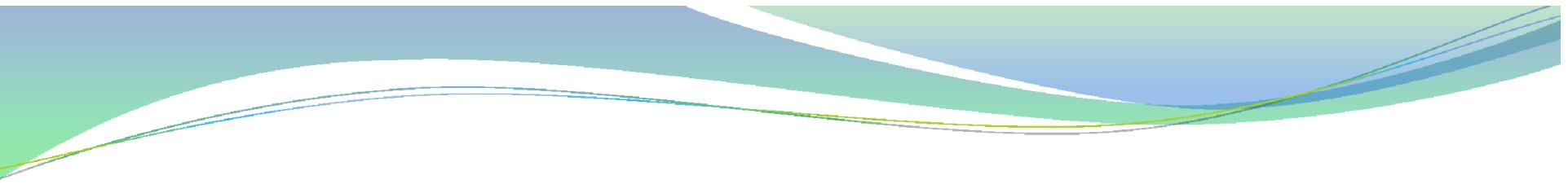
- A fost propus în 1983 de Goodman J. și îmbină avantajele ambele metode anterior prezentate.
- În scopul reducerii traficului pe magistrală prima operațiune de scriere a unui bloc din cache folosește tehnica de scriere imediată.
- Aceasta va conduce la invalidarea celorlalte copii cât și la transmiterea modificării în memoria comună.



Protocolul cu scriere unică

- Schema de lucru este prezentată în figura, unde prin linii continue sunt referite comenzi inițiate de procesorul local iar prin linii punctate sunt indicate comenzi inițiate de alte procesoare prin intermediul magistralei sistem, sub forma unui graf ce conține următoarele stări:
 - **Valid**: blocul din memoria cache care este identic cu cel din memoria principală a fost citit din aceasta și nu a fost modificat
 - **Invalid**: blocul fie nu există în memoria cache fie nu coincide cu copia din memoria principală
 - **Rezervat**: datele au fost scrise exact o singură dată de la începerea citirii din memoria comună. Copia din cache este identică cu cea din memoria comună care este singura copie existentă.
 - **Dirty**: blocul din cache a fost modificat mai mult de o dată iar respectiva modificare nu a fost propagată. În acest caz avem o copie unică în sistem.

- Din punct de vedere al celorlalte procesoare există următoarele posibile comenzi externe primite de la magistrală: “read invalidate” care citește din bloc și invalidează toate copiile și “write invalidate” care invalidează toate copiile unui bloc.
- Lipsa unui bloc din cache (“read miss”) apare atunci când procesorul dorește să citească un bloc care nu se află în cache.
- În aceste condiții se va iniția o operațiune de citire pe magistrală.
- Dacă nu există copii marcate ca interzise (“dirty”) înseamnă că memoria comună deține o copie corectă și deci o va furniza.

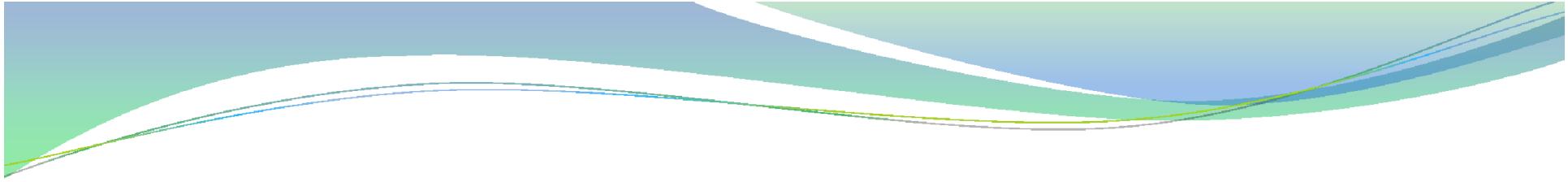


- În cazul în care există un bloc în cache și poate fi scris (“write hit”), dacă copia este “dirty” sau rezervată, atunci operațiunea de scriere este realizată local și noua stare este “dirty”.
- Dacă noua stare este validă atunci toate celelalte copii sunt invalidate, memoria comună trece la scriere imediată iar starea rezultată este rezervată după prima scriere.
- Când un procesor nu poate realiza scrierea în cache-ul local (“write miss”) aceasta înseamnă că copia trebuie să sească fie din memoria comună fie dintr-un cache “dirty”.

- Când blocul dorit este disponibil pentru citire (“read hit”): se poate oricând realiza într-un cache local fără a pune probleme de coerență.
- Atunci când avem o situație de înlocuire a blocului (“block replacement”) dacă o copie este “dirty” înseamnă că trebuie întâi scrisă în memoria comună prin înlocuirea blocului. Dacă copia este “clean”, situație când aceasta este validă, rezervată sau invalidă, nu se va face nici o înlocuire.
- Se observă că acest protocol necesită o linie specială a magistralei care să poată realiza inhibarea memoriei comune când o copie este invalidă și o operațiune de citire a magistralei este necesară după o situație de inexistență a blocului dorit la citire.

Servicii cu stare versus servicii fără stare

- Există două moduri de abordare a problemei stocării informației pe server în momentul în care un client accesează fișiere de pe alt system:
 - ori serverul urmărește fiecare fișier accesat de fiecare client,
 - sau pur și simplu pune la dispoziție blocuri pe măsură ce sunt cerute de client, fără a-l interesa cum sunt folosite acele blocuri.
- În primul caz avem de-a face cu un serviciu bazat pe stare,
- în cel de-al doilea cu servicii care nu sunt bazate pe stare.
-



- Un serviciu fără stare este descris ca o conexiune între client și server în timpul unei sesiuni.
- Ori prin închiderea fișierului sau prin mecaismul de garbage-collection serverul trebuie să își curețe spațiul principat de memorie folosit de clienții care nu mai sunt activi.
- Punctul principal legat de toleranța la erori într-un serviciu necaracterizat de stare este acela că serverul menține informația legată de clienți.
- Un serviciu de fișiere necaracterizat de stare evită informațiile despre stare făcând în aşa fel încât fiecare cerere să conțină informații despre sine.

Replicarea fișierelor

- Replicarea fișierelor pe diferite mașini într-un sistem de fișiere distribuit este un mod bun de a îmbunătăți disponibilitatea.
- Replicarea pe mai multe mașini poate crește și performanța: selectarea unei replici mai apropiate poate duce la timpi de acces mai mici.
- Cerințele de baza ale schemei de replicare este ca replici diferite ale aceluiași fișier să existe pe mașini independente la căderi.

- Este de dorit să se ascundă detaliiile replicării față de useri.
- Maparea unui fișier replicat într-o replică particulară este un task al schemei de nume.
- Existența replicii ar trebui să nu fie vizibilă la nivele superioare.
- La nivele inferioare replicile trebuie să poată să fie diferențiate prin nume diferit.
- Altă cerință legată de transparentă este să se realizeze controlul replicilor la un nivel superior.
- Controlul replicării include determinarea nivelului replicării și a locului unde se depun replicile. În aceste situații, vrem să expunem aceste detalii userilor.

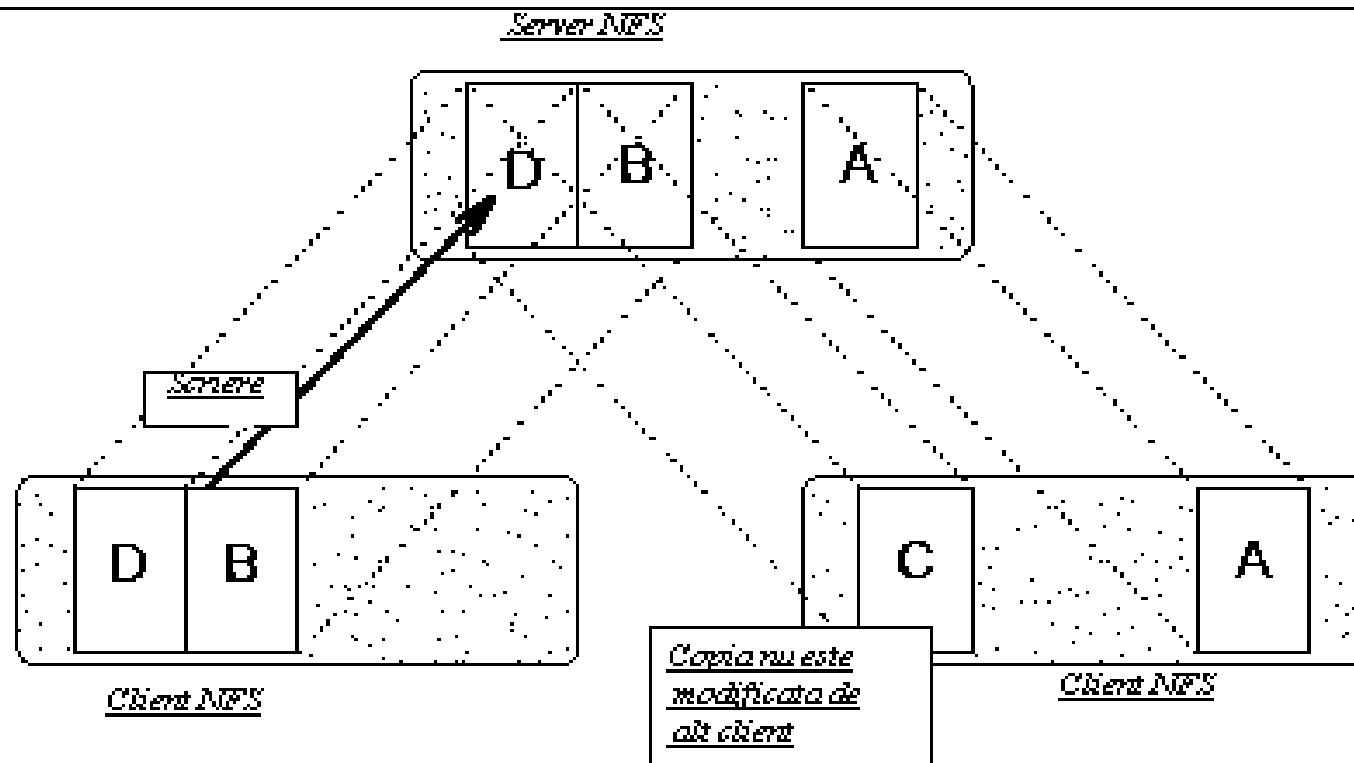
- Principala problemă a replicării este legată de updateare.
- Din punctul de vedere al utilizatorului, replica unui fișier trebuie să denote aceeași entitate logică și, de aceea, un update asupra oricărei replici trebuie să se reflecte asupra tuturor celorlalte replici.
- Mai exact, coherența semantică trebuie păstrată între accesul replicilor.
- Dacă coherența nu este de importanță majoră, atunci ea poate fi sacrificată în avantajul disponibilității și a performanței

Sistemul NFS(Network File Sistem)

- NFS a fost proiectat de Sun Microsystems din SUA, pentru a crea un sistem distribuit de stocare pentru mașinile lor.
- Aceasta permite o bună transparență la acces, toate fișierele sistem au fost combinate într-un arbore sistem conținut de un singur fișier,
- În practică totuși această abordare este destul de complicată ne nevoia de a furniza date specifice la locații fixe (în UNIX) fiecare sistem necesită date diferite dar provenind din același loc.

- Din nefericire NFS nu oferă o soluție completă.
- Există o slăbiciune la nivelul securității.
- Până recent, schimbul de informații între mașini se făcea la nivel de fișiere text adică datele puteau fi citite de oricine.
- Criptarea poate furniza protecția datelor precum și verificarea/autorizarea accesului la date
- Deci accesul unei mașini pe alta este permis doar cu cheia de criptare/decripare specifică.

Problema inconsistentei cache în NFS



- Această politică de tip write-through garantează securitatea datelor dar implică modificarea simultană a copiilor existente în rețea.
- NFS nu gestionează foarte bine cazul căderilor mașinii.
- Prin definiție UNIX -ul o dată ce face o scriere în fișier (apel `write()`) și nu sunt raportate erori asta înseamnă că data a fost scrisă corect.

NFS V4

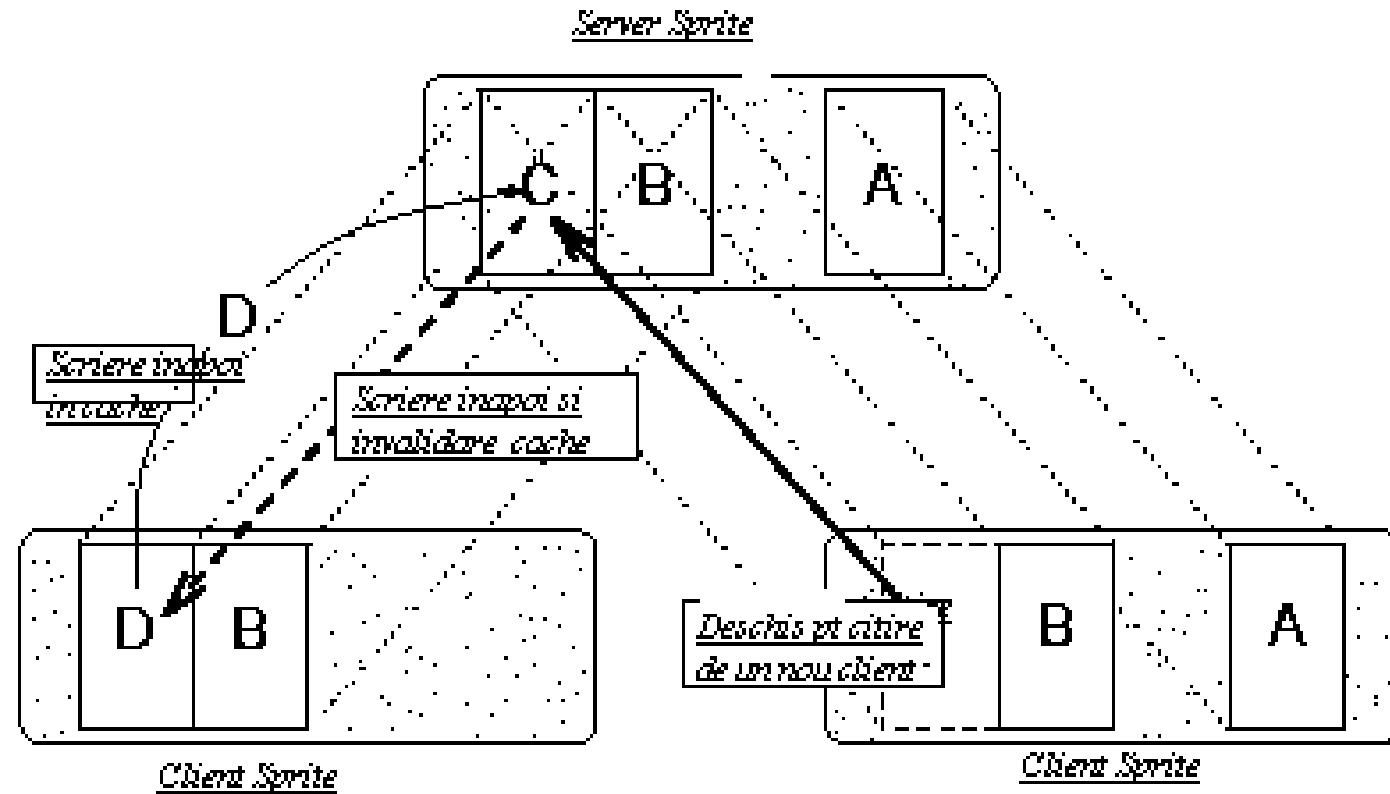
- Diferă fundamental de versiunile anterioare.
- Cea mai semnificativă schimbare este aceea că protocolul este acum bazat pe stare, asta însemnând că serverul menține starea sesiunii clientului din momentul în care este deschis remote un fișier și până la închiderea sa.
- De aceea, protocolul NFS are acum operațiile `open()` și `close()`; versiunile anterioare de NFS, care nu sunt bazate pe stare, nu pun la dispoziție astfel de operații.
- Mai mult, versiunile anterioare specifică protocoale separate pentru montarea sistemelor de fișiere remote și pentru a le bloca.

- Adițional, V4 a mărit abilitatea clientilor de a face cache cu fișierele de date local.
- Această capabilitate îmbunătățește performanțele sistemului distribuit de fișiere, dându-le clientilor abilitatea de a rezolva mai multe accese din cacheul local, în loc de a-i pune să treacă prin server.
- V4 le oferă abilitate clientilor de a cere lockuri pe fișierele de pe servere. Dacă serverul acordă permisiunea, clientul menține lockul până când este eliberat sau până îl expiră perioada.

SNF Sprite Network File system

- Acest sistem de operare a fost creat la Berkley SUA pentru a îndeplini funcții similare ca și NFS oferind soluții similare în problema transparenței accesului și a transparenței locațiilor. Proiectanții lui au ținut cont de problemele NFS -ului și au încercat să le corecteze fără a pierde din performanță sau flexibilitate. Cele trei probleme majore rezolvate sunt:
- 1. Evitarea ca fișierele temporare să fie scrise înapoi dacă nu este necesar
- 2. Evitarea întârzierii operațiunii de scriere datorită vitezei rețelei
- 3. Garantarea faptului că cea mai recentă scriere va fi văzută de toți clienții.
-

- Pentru obținerea acestor țeluri Sprite folosește servere de stare care mențin informația despre cine accesează rețeaua și cum o accesează.
- Când un fișier este accesat de un client, serverul informează despre operațiune împreună cu tipul operației.
- Când un fișier este deschis de un client nu mai apar probleme indiferent de modul de lucru pentru că datele sunt prelucrate local în cache deci nu sunt comune nimănuí.



Mecanismul "callback" în SPRITE

- Din punct de vedere al stabilității Sprite este mai bun ca NFS.
- Căderile și refacerea datelor după acea sunt rezolvate prin folosirea unei combinații între o bună structurare a fișierelor sistem și o stare distribuită menținută de fișierele sistem ale clienților.
- Un server NFS se poate bloca și reporni fără a avea cunoștință despre programele clienților.
- Sprite folosește serverele de stare care mențin informații despre fiecare client și dacă este necesară încercarea de recuperare a datelor, sau nu, în caz de cădere a serverului.
-



DFS

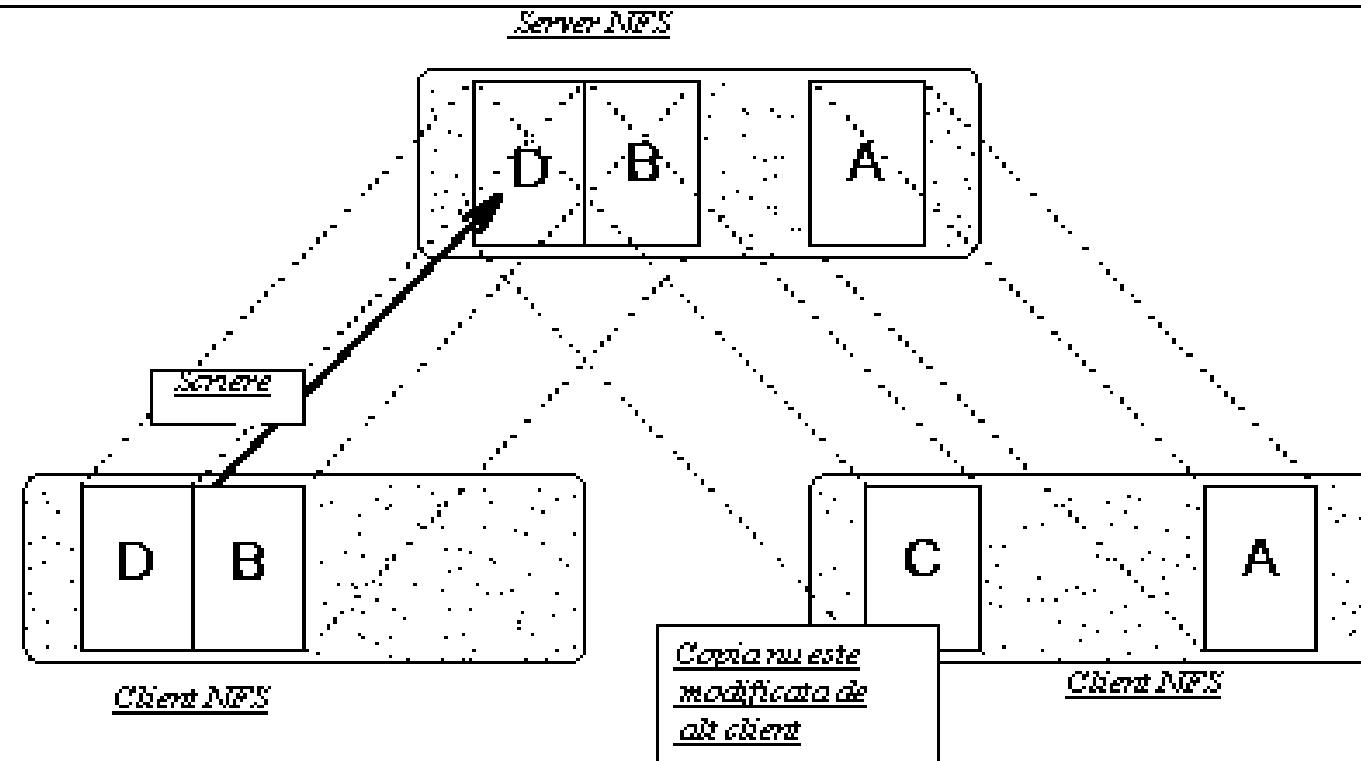
Curs 4

Sistemul NFS(Network File Sistem)

- NFS a fost proiectat de Sun Microsystems din SUA, pentru a crea un sistem distribuit de stocare pentru mașinile lor.
- Aceasta permite o bună transparență la acces, toate fișierele sistem au fost combinate într-un arbore sistem conținut de un singur fișier,
- În practică totuși această abordare este destul de complicată ne nevoia de a furniza date specifice la locații fixe (în UNIX) fiecare sistem necesită date diferite dar provenind din același loc.

- Din nefericire NFS nu oferă o soluție completă.
- Există o slăbiciune la nivelul securității.
- Până recent, schimbul de informații între mașini se făcea la nivel de fișiere text adică datele puteau fi citite de oricine.
- Criptarea poate furniza protecția datelor precum și verificarea/autorizarea accesului la date
- Deci accesul unei mașini pe alta este permis doar cu cheia de criptare/decripare specifică.

Problema inconsistenței cache în NFS



- Această politică de tip write-through garantează securitatea datelor dar implică modificarea simultană a copiilor existente în rețea.
- NFS nu gestionează foarte bine cazul căderilor mașinii.
- Prin definiție UNIX -ul o dată ce face o scriere în fișier (apel write()) și nu sunt raportate erori asta înseamnă că data a fost scrisă corect.

- O situație similară apare atunci când se face citirea unei date din cache iar serverul nu răspunde.
- După cum se observă o rezolvare posibilă a acestor probleme este abandonarea operațiunii în curs de desfășurare după un timp fără reacție de la server-client, din păcate această metodă nu prezintă interes.
- Sunt rezolvate totuși câteva probleme de management distribuit.
- De exemplu protecția este asigurată de atributie de fișier tip standard UNIX owner group etc.
- Acest lucru dă rezultate când rețeaua este administrată centralizat și fiecărui client din rețea î se poate da o identificare unică în rețea.

NFS V4

- Diferă fundamental de versiunile anterioare.
- Cea mai semnificativă schimbare este aceea că protocolul este acum bazat pe stare, asta însemnând că serverul menține starea sesiunii clientului din momentul în care este deschis remote un fișier și până la închiderea sa.
- De aceea, protocolul NFS are acum operațiile `open()` și `close()`; versiunile anterioare de NFS, care nu sunt bazate pe stare, nu pun la dispoziție astfel de operații.
- Mai mult, versiunile anterioare specifică protocoale separate pentru montarea sistemelor de fișiere remote și pentru a le bloca.

- Adițional, V4 a mărit abilitatea clientilor de a face cache cu fișierele de date local.
- Această capabilitate îmbunătățeste performanțele sistemului distribuit de fișiere, dându-le clientilor abilitatea de a rezolva mai multe accese din cacheul local, în loc de a-i pune să treacă prin server.
- V4 le oferă abilitate clientilor de a cere lockuri pe fișierele de pe servere.
- dacă serverul acordă permisiunea, clientul menține lockul până când este eliberat sau până îi expiră perioada.

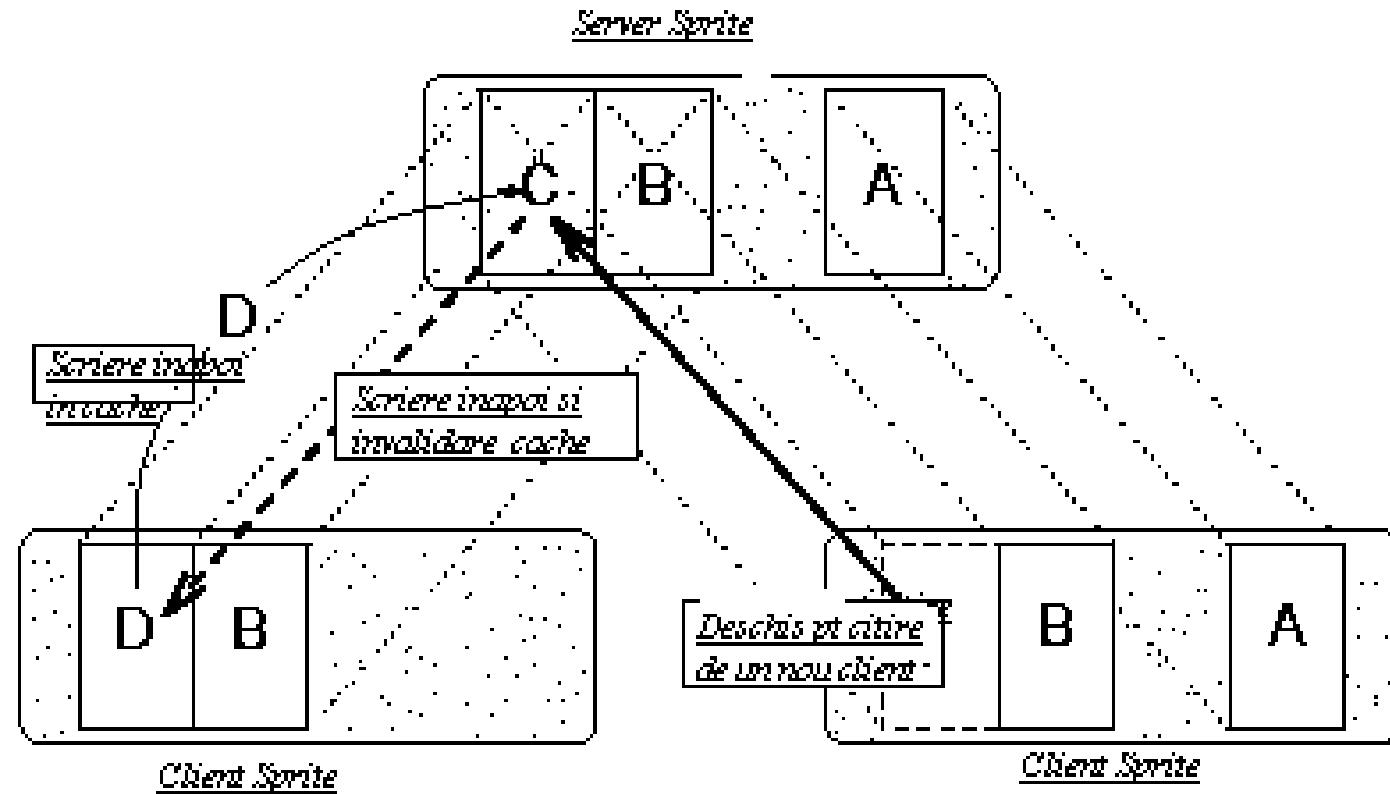
- Noile mecanisme de locking și cacheing sunt bazate pe conceptual delegării, unde serverul delegă responsabilități pentru un lock pe fișier și conținut clientului care a cerut lockul.
- Clientul delegate menține în cache versiunea curentă a fișierului și alți clienți pot să ceară clientului delegate un lock și conținutul fișierului până când fișierul delegat va elibera lockul și delegarea.
-

SNF Sprite Network File system

- Acest sistem de operare a fost creat la Berkley SUA pentru a îndeplini funcții similare ca și NFS oferind soluții similare în problema transparenței accesului și a transparenței locațiilor.
- Proiectanții lui au ținut cont de problemele NFS -ului și au încercat să le corecteze fără a pierde din performanță sau flexibilitate. Cele trei probleme majore rezolvate sunt:
 - 1. Evitarea ca fișierele temporare să fie scrise înapoi dacă nu este necesar
 - 2. Evitarea întârzierii operațiunii de scriere datorită vitezei rețelei
 - 3. Garantarea faptului că cea mai recentă scriere va fi văzută de toți clienții.

- Pentru obținerea acestor țeluri Sprite folosește servere de stare care mențin informația despre cine accesează rețeaua și cum o accesează.
- Când un fișier este accesat de un client, serverul informează despre operațiune împreună cu tipul operației.
- Când un fișier este deschis de un client nu mai apar probleme indiferent de modul de lucru pentru că datele sunt prelucrate local în cache deci nu sunt comune nimănuí.

- Sprite de asemenea folosește o politică de scriere întârziată. În loc să trimită, pentru modificare imediată, la o scriere către fișierul de pe server, modificările sunt făcute în cache-ul local și trimise prin rețea pentru o reală modificare numai atunci când s-au terminat toate modificările în respectivul fișier.
- Există ocazii când Sprite face scriere înainte de terminarea tuturor modificărilor.
- Una din cazuri, și cel mai simplu este atunci când cache-ul clientului este plin și pentru a face spațiu se trimit modificările procesul reluându-se.



Mecanismul "callback" în SPRITE

- Din punct de vedere al stabilității Sprite este mai bun ca NFS.
- Căderile și refacerea datelor după acea sunt rezolvate prin folosirea unei combinații între o bună structurare a fișierelor sistem și o stare distribuită menținută de fișierele sistem ale clienților.
- Un server NFS se poate bloca și reporni fără a avea cunoștință despre programele clienților.
- Sprite folosește serverele de stare care mențin informații despre fiecare client și dacă este necesară încercarea de recuperare a datelor, sau nu, în caz de cădere a serverului.

- Altă problemă curentă este legată de resincronizarea fișierelor sistem UNIX cu clienții după căderea unui server, apărând necesitatea verificării consistenței fișierelor sistem.
- Pe un Sun 4 cu harddisk de 2 Gb spațiu pe disc această procedură poate dura cel puțin 15 minute.
- Sprite încearcă să remedieze această problemă prin folosirea fișierelor de înregistrări sistem structurate în locul celor bazate pe o modificare convențională.
- Acest fișier de înregistrări împreună cu verificarea periodică a lui formează un sistem de fișiere similar sau mai bun ca performanțe decât UNIX-ul, posedând în plus și avantajul scăderii timpului de refacere

- Periodic zonă de verificare este scrisă pe disc într-o poziție fixată (de fapt sunt menținute două asemenea zone care sunt modificate alternativ în scopul evitării erorilor de scriere din diverse motive).
- Această zonă conține suficiente informații pentru a permite înregistrărilor să fie indexate și citite în mod eficient. și, de asemenea face ca sistemul să posede puncte în care consistența să fie garantată.
- Fără aceste informații citirea unui fișier de pe disc ar necesita examinarea întregului fișier de înregistrări pentru a putea refacă fișierul în cazul unei căderi a sistemului.

Sistemul de operare Decorum

- Decorum folosește principii similare cu Sprite pentru a furniza un sistem de operare de tip UNIX dar capabil să lucreze pe mașini distribuite.
- Pentru interfațare un sistem virtual de fișiere (VFS Virtual File System) este atașat sistemului .
- Interfața permite de asemenea ca diferite fișiere la nivel fizic să fie folosite simultan de clienți care pot vedea același fișier în aceiași manieră.
- Un avantaj major este faptul că fișiere NFS pot fi folosite de clienții Decorum fără modificări.

- Sunt folosite mai multe tipuri de etichete:
 - de date (“data token”) - permit clientului să citească sau să scrie (depinzând de subtipul etichetei) un număr de octeți dintr-un fișier.
 - de stare (“status token”) - permit clientului să citească sau să scrie (depinzând de subtipul etichetei) informația de stare asociată cu un fișier.
 - de blocare (“lock token”) - permit clientului blocheze citirea sau scrierea (depinzând de subtipul etichetei) unui număr de octeți dintr-un fișier.
 - de deschidere (“open token”) - dă clientului dreptul de a deschide un fișier într-un mod particular (depinzând de subtipul etichetei).

- Etichetele de diferite tipuri nu interacționează între ele atât timp cât se referă la diferite elemente (proprietăți) ale fișierului.
- Acordarea și revocare acestor etichete este similară cu mecanismul de rechemare (callback) existent în Sprite.
- Din acest punct de vedere putem spune că Sprite posedă o singură etichetă care spune dacă cu un fișier se poate lucra în stilul cache sau nu.
- În comparație cu Sprite, Decorum are o granularitate mult mai mică relativ la controlul lucrului cu cache.

- Protecția este simplă fiind asigurată de funcțiile UNIX standard.
- În plus sunt adăugate liste de control al accesului (ACL Acces Control List).
- Consistența datelor este menținută prin copii pe servere.
- Aceste servere sunt responsabile de mirroring-ul (crearea unei copii a unei zone sau chiar a întregului dispozitiv de stocare, în scopul creșterii toleranței la erori a sistemului) continuu al sistemului folosind o politică de replicare lentă, care garantează ca va face copii ale datelor într-o perioadă maximă de timp.

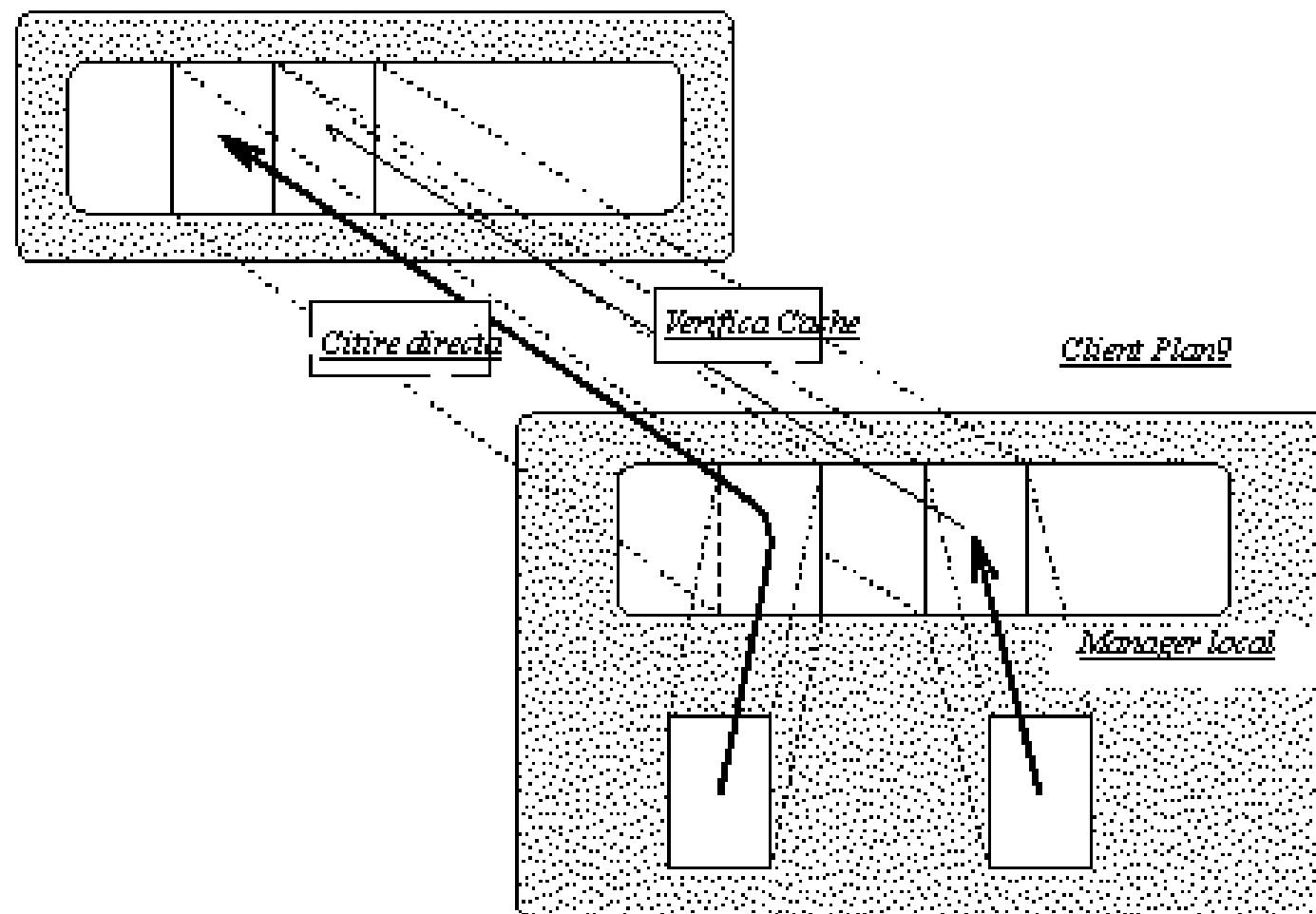
- În general nu trebuie ca durata de replicare să fie mică deci eforturile se îndreaptă în continuare spre scăderea costului de multiplicare.
- Decorum îndeplinește multe din cerințele unui sistem de stocare distribuit.
- Capacitatea de a replica fișierele sistem aşa cum sunt îl face foarte puternic în prevenirea căderilor de servere sau de rețea.
- Sistemul lui de protecție previne bine accesul utilizatorilor nedoriți.

Sistemul de fișiere plan 9.

- Este un sistem de operare distribuit dezvoltat de laboratoarele Bell de către proiectanții UNIX-ului.
- Deși nu este compatibil cu UNIX el este proiectat ca să suporte următoarele mașini în rețea
- Servere extrem de puternice (de obicei multiprocesor) furnizează puterea de calcul pentru utilizatori
- Servere de fișiere - furnizează zonele de stocare date
- Terminale grafice dedicate - permite o interfață grafică pentru utilizatori

- Plan 9 nu are o transparentă a datelor coerentă și nici protocol de coerență a datelor.
- În multe cazuri comunicația dintre serverul de date și serverul procesor este presupus a fi suficient de rapid ca să nu mai trebuiască să lucreze cu cache.
- Fără cache dispar problemele legate de coerență.
- Oricum Plan 9 este proiectat să lucreze cu o varietate de tipuri de rețea incluzând și cele lente .

- Când un fișier de pe server este deschis de către un client indiferent dacă este de date sau executabil se returnează o cheie de 64 biți către client.
- O parte a acestei chei identifică fișierul iar restul identifică tipul fișierului.
- Această cheie este comparată cu fiecare din cheile ținute de orice secțiune aflată în secțiunea cache a fișierului clientului.
- Dacă cheile sunt identice atunci datele din cache pot fi sterse iar data trebuie recitită de pe server.



- Soluția abordată pentru a realiza transparență locațiilor și a accesului este ca fiecare grup de procese să aibă propriul “namespace” (identifier a spațiului de memorie comun) care îi permite să construiască și să modifice cum dorește zona proprie de lucru.
- Este posibil pentru orice utilizator să construiască un mediu de lucru (environment) similar neavând importanță de unde se face accesul față de locația datelor.
- Această abilitate de a construi “namespace” individuale ajută managementul distribuit, din moment ce fiecare aplicație poate construi un fișier sistem care îi trebuie relativ la ceea ce serverul poate oferi.

- Pentru a realiza pornirea sistemului se folosește un identificator special de utilizator “none”.
- Schema de acces nu este ca în NFS unde o mașină considerată ca de “neîncredere” este automat mapată ca alții.
Oricum folosirea numelor textuale și a unei politici orientată după numele utilizatorului permite controlul distribuit simplu a utilizatorului .
- Plan 9 atacă problema sistemelor mari distribuite prin relaxarea multora dintre semanticile UNIX-ului clasic. Soluția autentificării este adekvată dar nu atât de flexibilă cât ar trebui.

- Andrew este un mediu computațional distribuit, gândit și implementat la Universitatea Carnegie Mellon. Sistemul de fișiere Andrew constituie mecanismul de schimb de informații între clienții din acel mediu.
- Transarc Corporation a preluat dezvoltarea AFS. Ei au fost apoi preluăți de IBM.
- IBM a produs de atunci câteva implementări comerciale de AFS.
- AFS a fost apoi ales ca sistemul de fișiere distribuit pentru o coaliție din industrie, rezultând Transarc DFS, care e parte a sistemului computațional distribuit a organizației OFS.

- Majoritatea vendorilor de UNIX, precum și Microsoft, suportă sistemul DCE, precum și DFS-ul său care este bazat pe AFS. Se continuă lucrul în încercarea de a face din DCE un DFS cross-platform general acceptat.
- Cum AFS și Transarc DFS sunt similare, vom descrie AFS în cele ce urmează și ne vom referi la Transarc DFS atunci când va fi specificat explicit acesta.
- AFS încearcă să rezolve multe dintre problemele DFS-urilor mai simple, cum ar fi NFS, și este cel DFS-ul neexperimental cu cele mai multe facilități.
- Conține un spațiu de nume uniform, un sistem de sharing independent de locație, cacheing pe client oferind consistență cacheului și autentificare securizată folosind Kerberos.

- AFS face distincția între mașini client (uneori numite stații de lucru) și serverele dedicate.
- Serverele și clienții rulau la început doar versiunea 4.2 de BSD UNIX, dar AFS a fost portat și pe alte sisteme de operare.
- Clienții și serverele sunt interconectați de o rețea de LANuri sau WANuri.
- Clienții au un spațiu de nume de fișiere partitiorat: un spațiu de nume local și unul shared.
- Serverele dedicate, numite colectiv Vice, după numele softwareului pe care îl rulează, pun la dispoziția clientilor spațiul de nume sharat sub forma unui sistem de fișiere ierarhic omogen, identic, transparent din punctul de vedere a locației.

- Tot locale sunt fișierele temporare și fișierele pe care posesorul stației, din motive de asigurare a intimității, dorește explicit să le salveze local.
- Văzute mai în amănunt, clienții și serverele sunt structurate în clustere interconectate de un WAN. Fiecare cluster este constituit dintr-o colecție de stații de lucru dintr-un LAN și un reprezentant Vice, numit server de cluster, și fiecare cluster este conectat la WAN printr-un router.
- Descompunerea în clustere este făcută în special pentru a adresa problema scalabilității.
- Pentru performanțe optimale, stațiile trebuie să folosească serverul din clusterul lor în cea mai mare parte din timp, de aceea referințele între clustere sunt relativ infrecvente.

- Pe scurt, câteva probleme adiționale legate de designul AFSului:
- Mobilitatea clientului. Clienții sunt capabili să acceseze orice fișier în spațiul de nume sharat de orice stație de lucru. Un client poate observa o degradare a performanțelor inițiale din cauza chacheingului fișierelor când se accesează fișiere de pe o altă stație decât cea obișnuită.
- Securitatea. Interfața Vice este considerată ca fiind o limitare a siguranței, pentru că nici un program client nu e executat pe mașina Vice. Autentificarea și transmiterea securizată sunt realizate ca parte a comunicației bazate pe pachete a paradigmelor RPC.

- Protecția. AFS pune la dispoziție liste de acces pentru a proteja directoare și fișierele normale UNIX pentru protecția fișierelor.
- Lista de acces poate conține informații despre acei useri care au dreptul să acceseze un director, precum și informații despre userii care nu au acest drept.
- De aceea este ușor de specificat că oricine cu excepția, să zicem, a lui Jim, are dreptul să acceseze un director.
- AFS oferă suport pentru acțiuni ca citire, scriere, căutare, inserare, administrare, blocare și ștergere.

Spațiul de nume comun

- Spațiul de nume comun al AFS este constituit din componente unitare, numite volume.
- Volumele sunt componente unitare mici.
- Tipic, ele sunt asociate cu fișierele de pe un singur client.
- Puține volume pot sta pe o singură partitie și ele sunt capabile să crească (până la o anumită cotă) sau să scadă ca mărime.
- Conceptual, volumele sunt legate între ele printr-un mecanism similar cu mecanismul mount din UNIX.
- Totuși, diferența de granularitate e semnificativă, deoarece în UNIX doar o partitie de disc (care conține sistemul de fișiere) poate folosi mount.

- Identifierul unic face posibilă reutilizarea numărului nodului, conținând niște structuri de date compacte.
- Fid-urile sunt transparente la locații, de aceea mutarea fișierelor de la server la server nu invalidează conținutul directoarelor din cache.
- Informațiile despre locații sunt ținute într-o bază de date replicată pe fiecare server.
- Un client poate identifica locația fiecărui volum din sistem prin interogarea bazei de date.
- Agregarea fișierelor în volume face posibilă menținerea bazei de date la valori acceptabile.

- În timpul transferului volumului, serverul care transmite poate încă continua să realizeze updateuri, care sunt transmise mai tarziu spre noul server.
- La un moment dat, volumul este dezactivat pentru puțin timp, astfel încât modificările să fie procesate; apoi, noul volum devine disponibil din nou la noua locație.
- Operația de transmitere a volumului este atomică.
- Dacă oricare dintre servere cade, operația este oprită.

Operații pe fișiere și consistență semantică

- Principiul arhitectural fundamental al AFS este cachingul fișierelor întregi de pe servere.
- În mod corespunzător, o stație client interacționează cu serverul Vice numai în timpul operațiilor de deschidere și închidere a fișierelor, și chiar și această interacțiune nu este tot timpul necesară.
- Citirea și scrierea fisierelor nu duce la interacțiuni remote (în contrast cu metoda remote-service).
- Aceasta distincție cheie are ramificații impornatnte pentru performanță, și pentru semantică și operații cu fișiere.

- Venus ar putea contacta Vice numai când un fișier e fișier e deshis sau închis; citirea și scrierea byte-ilor individuali ai fișierelor sunt făcute direct pe copia din cache și trec peste Venus.
- Aceasta face ca citirea de pe unele locații nu sunt vizibile imediat în alte locații.
- Cacheingul este mai apoi exploatat pentru alte deschideri ale fișierului care a fost încarcat în cache.
- Venus presupune că intrările în cache (fie ele fișiere sau directoare) sunt valide doar dacă altceva nu se specifică.

- Când un client face cacheul unui fișier sau al unui director, serverul își updatează informația despre stare pentru a înregistra această cacheing.
- Spunem că clientul are un callback pe acel fișier.
- Serveul notifică clientul înainte să ii permită altui client să modifice fișierul.
- Într-un asemenea caz, spunem că serverul îndepărtează callbackul de pe fișier de la fostul client.
- Un client poate folosi un fișier din cache doar pentru a-l deschide atâta timp cât fișierul are callback.

- De aceea, singura ocazie în care Venus contactează Vice este atunci când se face deschiderea unui fișier care ori nu e în cache sau care a avut callbackul revicat și la închiderea locală modifică fișierul.
- În esență, AFS implementeză semantica sesiunii.
- Singura excepție este aceea a altor operații cu fișiere decât readurile și writeurile primitive (de exemplu schimbarea protecției la nivel de director), care sunt vizibile oriunde pe rețea imediat după ce operația se termină.
- În ciuda mecanismului de callback, traficul cauzat de validarea cacheului este scăzut, de obicei pentru a înlocui callbackurile pierdute din cauza căderilor mașinii sau a rețelei.

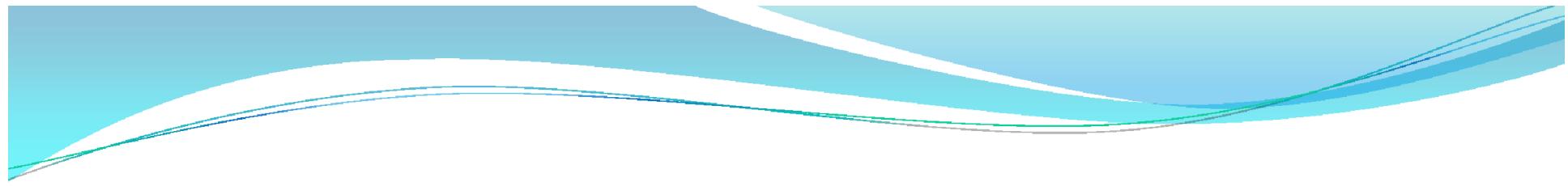
- Venus face cache și pentru directoare și linkuri simbolice, pentru traducerea căilor.
- Fiecare componentă din spațiul de nume din directoare este preîncărcat pentru el, dacă nu este deja în cache sau dacă clientul nu are un callback.
- Venus realizează căutări pe directoarele preîncărcate, folosind fidurile.
- Nicio cerere nu este trimisă de la un server la altul.
- Următoarele cereri de deschidere asupra acestui fișier nu vor provoca comunicare în rețea, cu excepția cazului în care un callback este oprit pe o componentă a numelui căii.

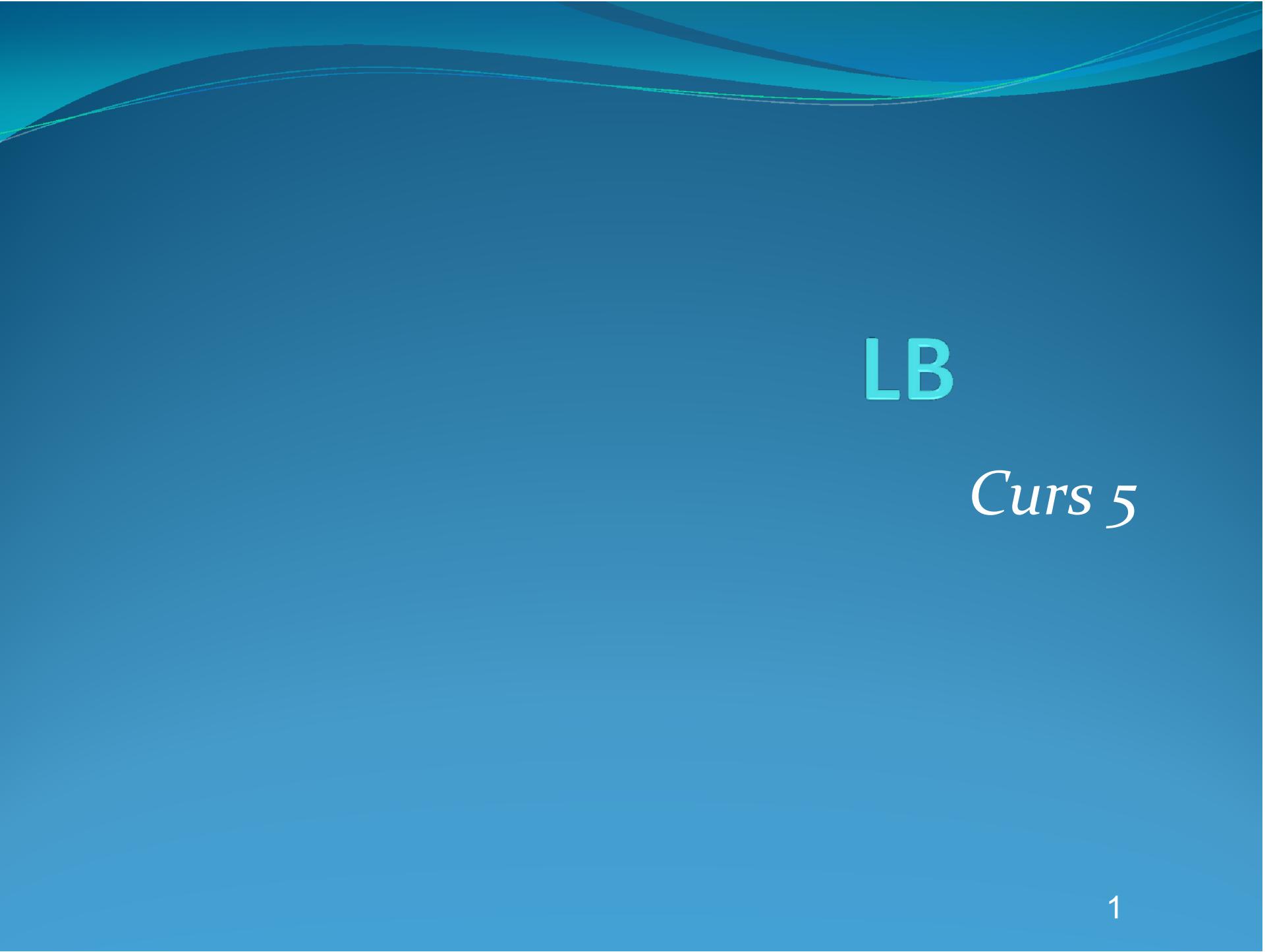
Implementarea

- Procesele client sunt interfațate cu un kernel UNIX cu un set ușual de cereri de sistem.
- Kernelul este puțin modificat pentru a detecta referințele la fișierele Vice din operațiile relevante și pentru a înainta cererile la procesul client Venus de pe stația de lucru.
- Venus face o traducere a spațiului de nume componentă cu componentă, așa cum a fost descris mai sus. Are un cache de mapare care asociază volumele cu locațiile de pe server pentru a elimina riscul de interogare a serverului pentru o locație de volum deja cunoscută.
- Dacă un volum nu există în cache , Venus contactează orice server cu care are deja o conexiune, cere locația informației, și introduce informația în cacheul mapat.

- Sistemul de fișiere UNIX este utilizat ca un sistem de stocare de nivel scăzut atât pentru clienții cât și pentru serverele AFS. Cacheul clientului este un director local pe discul stației.
- În acest director sunt fișiere ale căror nume sunt surtături spre intrările din cache. Atât Venus cât și procesele de pe servere acceseză fișierile UNIX direct după litera nodului pentru a evita rutina de traducere cale-nume-nod.
- Pentru că interfața nodului intern nu e vizibilă la procesele de nivel client (atât Venus cât și procesele server sunt procese client), un set coerent de cereri adiționale de sistem au fost adăugate.

- Chacheul de status este menținut în memorie pentru a permite un serviciu stat() rapid.
- Datele din cache sunt rezidente pe discul local, dar sistemul de bufferare de intrare-ieșire al UNIX face un cacheing a blocurilor discului în memorie într-un mod transparent pentru Venus.
- Un singur proces de nivel client pe fiecare server de fișiere face toate cererile de fișiere pentru client.
- Acest proces folosește un pachet proces ușor cu o planificare nepreemptivă pentru a servi concurrent mai multe cereri ale clienților.





LB

Curs 5

- Problema echilibrării încărcării a apărut o dată cu dezvoltarea mașinilor de calcul paralel.
- Este poate una din cele mai importante probleme din domeniul calculului de mare performanță.

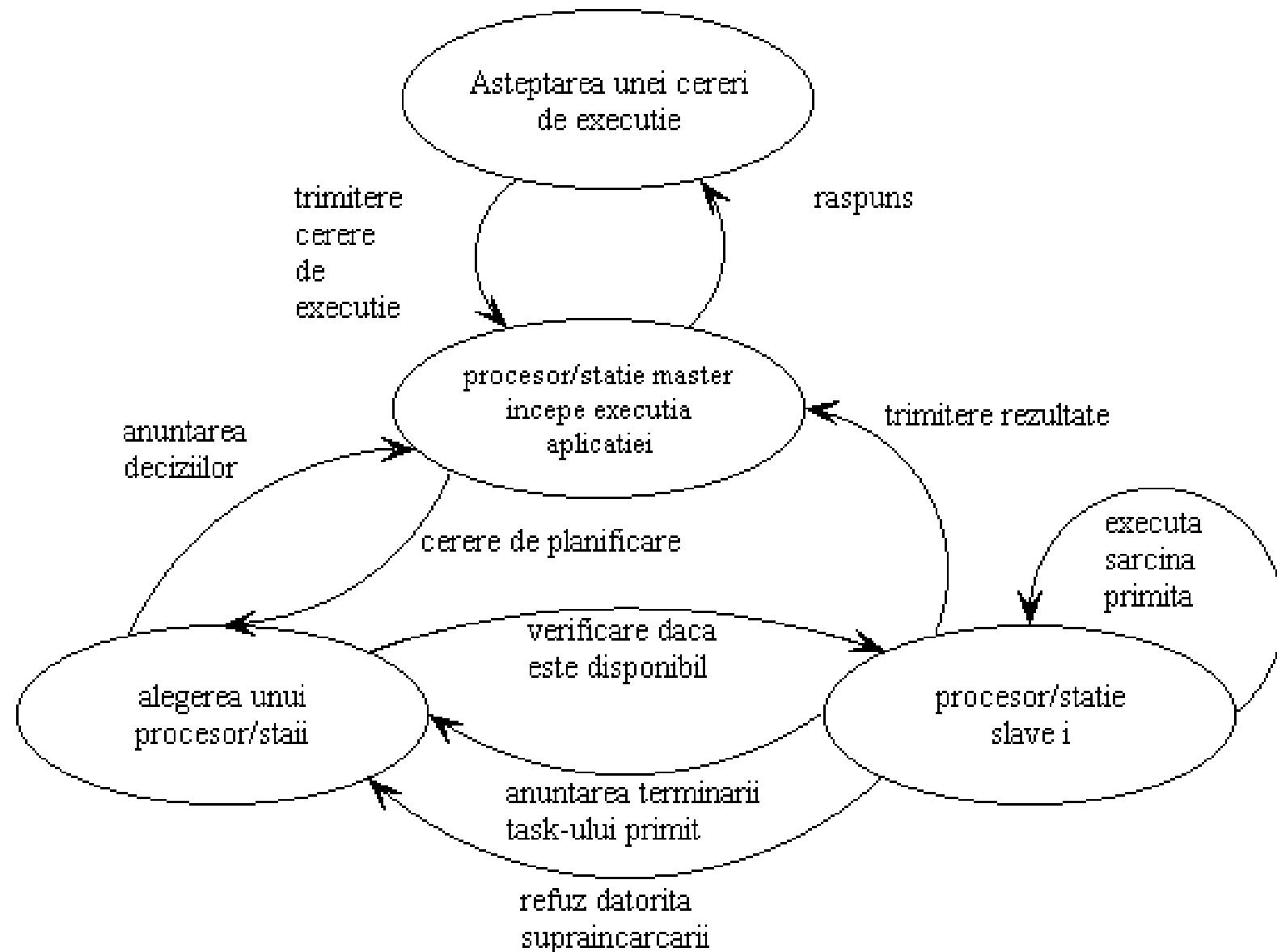
- La ora actuală, o dată cu dezvoltarea rapidă a Internetului și mai ales a ratelor de transfer suportate de acesta, s-a pus problema muncii cooperative, a mai multor stații în calculul aceleiasi aplicații, dar fără folosirea unui sistem centralizat

- În arhitecturile strâns cuplate una dintre problemele era dependența prea strânsă a performanțelor algoritmului de arhitectura hardware.
- La ora actuală aceste probleme specifice inițial numai calcului paralel se regăsesc și în sistemele de operare sau și aplicațiile de uz comun.

- În general prin încărcarea unui sistem se înțelege procentul de resurse folosite la un moment dat de o aplicație sau un grup de aplicații.
- Încărcarea unui sistem este legată de performanța acestuia.
- Putem spune că un sistem încărcat peste o anumită limită, chiar dacă mai poate accepta noi probleme spre rezolvare, nu mai satisface cererile în timpul lui normal de rezolvare.

- Deoarece echilibrarea încărcării este o tehnică de folosire optimă a resurselor de calcul rezultă că trebuie realizată o foarte fină analiză de rentabilitate înainte de alegerea unei tehnici anume.
- Evident că în forme rudimentare există implementată atât la nivel de sistem de operare cât și la nivel de aplicație.

Algoritmul generic al unei echilibrări a încărcării



Echilibrarea statică a încărcării

- Echilibrarea încărcării poate fi realizată atât static cât și dinamic.
- În cazul abordării statice toate calculele sunt realizate înainte de lansarea în execuție a aplicației.
- Acest tip de echilibrare mai este cunoscută în literatura de specialitate și sub denumiri ca problema mapării sau a planificării (“scheduling”).

- Chiar dacă prin analiza algoritmului de calcul s-a propus o metodă de mapare care să minimizeze timpii morți de execuție nimeni nu poate garanta că toate task-urile se vor termina cam în același interval temporal ceea ce conduce imediat la ideea că întârzierea mare a unui singur task poate anula efectele echilibrării în momentul în care se întâlnește un punct de sincronizare sau de colectare a rezultatelor.

- Cea mai mare parte a algoritmilor de planificare statică folosește una din metodele următoare: teoria grafurilor, programarea matematică, euristică.
- În primul caz, atât aplicația, cât și sistemul hardware sunt modelate sub forma unor grafuri, între care se încearcă găsirea unui omomorfism, sau o partiționare minimală (algoritmul “min cut”).

- Metodele din programarea matematică s-au impus, datorită ușurinței formulării obiectivului urmărit: minimizarea unei funcții cost, în prezența unor restricții.
- Funcțiile cost propuse se referă la minimizarea timpului total de comunicare între procese, sau a timpului total de comunicație la care se adaugă o componentă ponderată ce corespunde timpului de execuție a proceselor într-un sistem eterogen .

- Oricare ar fi funcția cost și restricțiile adoptate, problema poate fi rezolvată ca una de programare neliniară cu variabile bivalente.
- Metodele euristică încearcă să furnizeze o soluție apropiată de cea optimă, într-un timp rezonabil .

- În timp ce furnizează rezultate mai bune, algoritmii iterativi depind de soluția inițială .
- De aceea, se propune executarea unei variante “greedy”, pentru a obține o primă soluție, care este apoi îmbunătățită cu un algoritm iterativ.
- Cea mai răspândită metodă folosește gruparea proceselor într-o listă, conform unor anumite criterii fie în ordinea descrescătoare a timpilor de execuție, fie în ordinea impusă de terminarea execuției la termene fixe.

Cel mai apropiat vecin

- În această schemă fiecare procesor rămas fără job-uri, cere noi job-uri de la vecinii apropiati cum ar fi de exemplu într-un hipercub unde sunt $\log(n)$ vecini.

Round and Robin Asincron

- Fiecare procesor menține o variabilă țintă (“target”) independentă.
- De fiecare dată când procesorul rămâne în aşteptare, citeşte valoarea țintei locale şi trimite o cerere de treabă către un procesor particular.
-

Round and Robin Global

- În acest caz variabila întă este într-un prim procesor (“root”).
- Când un procesor cere un job, primește variabila întă, iar radacina incrementează respectiva variabilă.

Metode de planificare dinamică.

- În acest caz task-urile sunt alocate la procesoare în timpul execuției programului.
- Există două tipuri de abordări centralizată și descentralizată.

Abordarea centralizată

- Procesul conducător are imaginea globală asupra rezolvării problemei.

- Sarcini cu dimensiuni neprecizate apar cel mai des la aplicarea algoritmilor de căutare pentru care un task poate genera un număr oarecare de copii sau poate trimite imediat rezultatele către părintele lui.
- O abordare tipică în cazul implementării este folosirea unei cozi pentru task-urile care așteaptă să fie executate.

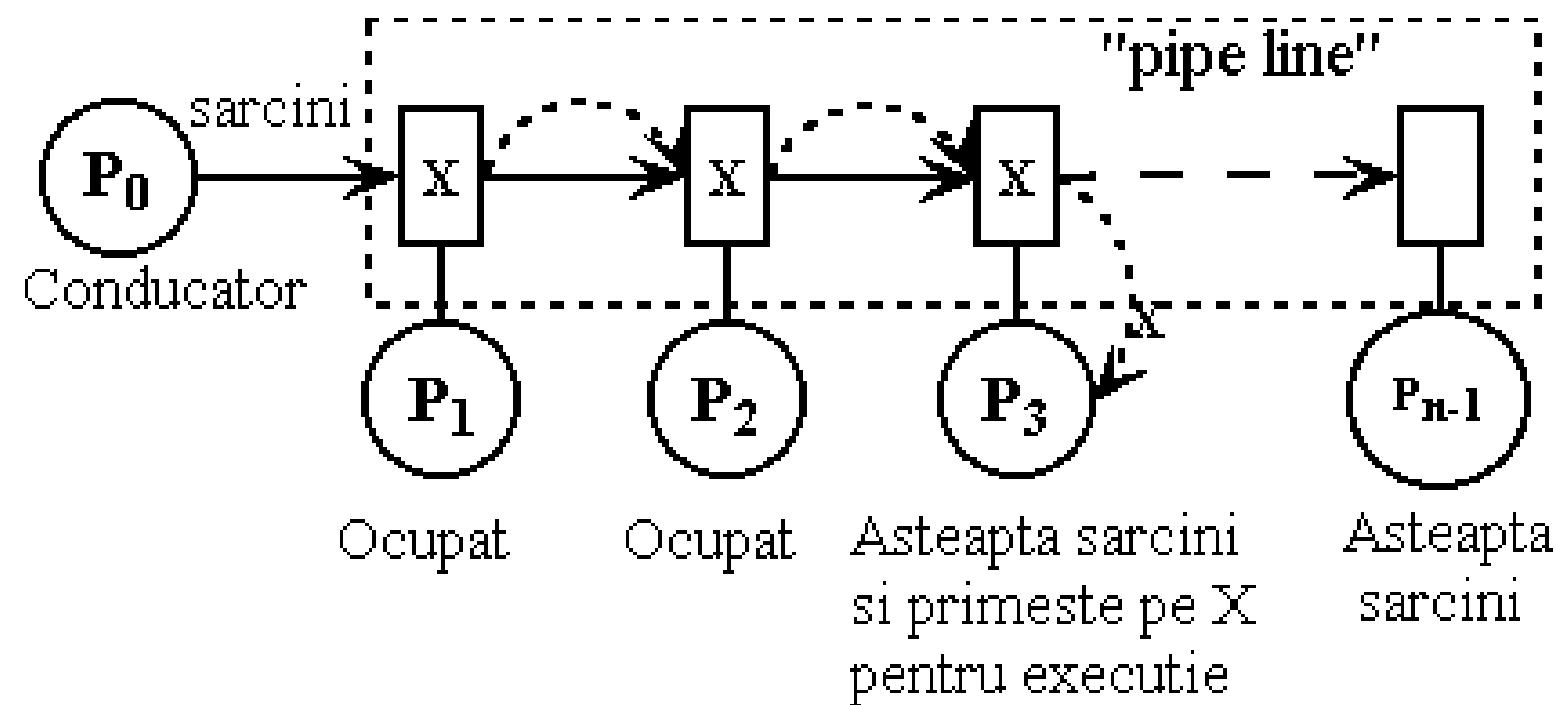
- Deși abordarea centralizată este foarte răspândită ea are o serie de dezavantaje dintre care cel mai important este încărcarea apărută la nivelul procesului conducător.
- Aceasta nu poate deservi decât un proces subordonat la un moment dat.
-

Abordarea descentralizată

- O primă soluție rezultă direct din modelul fermei de procesoare prin partaționarea atât a fermei cât și a problemei astfel încât fiecare conducător primește o subproblemă.
- Se observă că această abordare este parțial distribuită.
-

- În cealaltă situație se merge pe redirijarea sarcinilor suplimentare de către procesul sau procesorul care este supraîncărcat către receptorii care au o încărcare medie sau sub medie.
- Datorită calculului suplimentar necesar analizei permanente a încărcărilor locale din sistem această metodă este folosită în cazul sistemelor cu încărcare mică și mijlocie.
-

- O primă abordare relativ la implementare poate fi realizată folosind tehnica “round and robin”.



Etapele specifice realizării echilibrării dinamice

- O soluție practică pentru problema echilibrării încărcării dinamice poate implica următoarele etape:
- *Evaluarea încărcării* unde se realizează o estimare a încărcării procesorului trebuie realizată pentru a determina dacă există un eventual dezechilibru.

- *Determinarea profitabilității* unde se verifică dacă costul dezechilibrului depășește costul echilibrării încărcării, atunci echilibrarea încărcării poate fi inițiată.

Calcularea vectorului de transfer al încărcării

- Este calculat pe baza măsurătorilor făcute în prima fază când se calculează vectorul de transfer al încărcării pentru a echilibra procesoarele.
- Un algoritm ar fi SID propus de Willebeek. În acest caz se pornește de la premisa ca taskurile sunt infinit divizibile și încărcarea unui procesor este reprezentată de un număr real.

- Algoritmul DASUD se bazează pe algoritmul SID, care a fost îmbunătățit pentru a încorpora caracteristici noi, care detectează dacă un domeniu (toți vecinii imediați ai unui procesor) este echilibrat sau nu.
- Dacă domeniul nu este echilibrat, excesul de încărcare este distribuit vecinilor după diferite criterii, depinzând de distribuția încărcării lor.

- Comportamentul algoritmului DASUD poate fi divizat în trei etape, potrivit valorilor parametrilor anterioari, care servesc la măsurarea dezechilibrului unui domeniu la care procesorul i aparține.

Selectia taskurilor

- Se efectuează în acord cu vectorul de transfer calculat în faza anterioară.
- Există două opțiuni pentru satisfacerea vectorului de transfer dintre două procesoare.

Migrarea taskurilor

- Se realizează transferul efectiv al task-urilor de la un computer la altul.
- Aceasta trebuie să conserve integritatea stării unui task, incluzând orice mesaj SOSIT de pe canalele de comunicație.
-

Echilibrarea prin predicția încărcărilor

- Din nefericire, la proiectarea aplicațiilor specifice sistemele de operare Microsoft nu se poate atinge o finețe de echilibrare dorită, din cauza modului limitat în care o aplicație poate avea acces la resursele sistemului .
- Pentru multe aplicații proiectarea se bazează pe faptul că un task lansat la distanță (indiferent de modelul de calcul folosit) poate fi blocat sau întârziat suficient ca rezultatul muncii sale să nu mai fie folositor .

- Acest lucru poate fi evitat dacă proiectarea aplicației este abordată diferit.
- Cea mai comună situație este aceea în care serverul consideră un task neutilizabil (“dead”) în momentul în care, din cauza supraîncărcării stației pe care se află, nu mai lucrează normal.

Gestionarea taskurilor

- În proiectarea unei aplicații distribuite granularitatea acesteia trebuie luată întotdeauna în considerație.
- Totuși, influența acestui factor poate fi redusă, dacă se folosește o nouă abordare.
- În cazul sistemelor de operare Microsoft, controlul final alocării resurselor este scăzut față de UNIX.

- Decizia cu privire la cantitatea de muncă ce poate fi distribuită este complicată, de fapt reprezintă o problemă NP dificilă.
- De obicei soluțiile propuse rezolvă numai anumite subclase de probleme .

- Din nefericire, idea general susținută de rațiuni economice, este că un sistem de operare distribuit trebuie să fie transparent din punctul de vedere al folosirii resurselor locale de către alți utilizatori din rețea .
- De asemenea, este bine ca orice aplicație distribuită să urmeze aceeași gândire de prelucrare.

Predictia încărcării

- Idea de bază provine dintr-un model propus de Bhat [Bha98] legat de folosirea capacitaților de calcul ale unei stații atunci când proprietarul este plecat.
- Se definește un planificator S pentru furtul de timpi de lucru, unde $S=t_0, t_1, t_2, \dots$ și are definiția temporală prezentată în ecuația

$$\tau_k = \begin{cases} 0 & \text{if } k = 0 \\ T_{k-1} = t_0 + t_1 + \dots + t_{k-1} & \text{if } k > 0 \end{cases}$$

- Dacă utilizatorul stației B nu s-a reîntors în timpul $T_k = \tau_k + t_k$, cantitatea de muncă executată până în acest moment este w_k , dacă însă se întoarce înainte de scurgerea lui T_k atunci perioada de “împrumut” a resurselor ia sfârșit și munca efectuată până în acest moment este $w_0 + w_1 + \dots + w_{k-1}$.

$$w_k = t_k \Theta c, \quad \text{unde} \quad x \Theta y \stackrel{\text{def}}{=} \max(0, x - y)$$

- De asemenea sunt introduse două funcții de probabilitate p și q , după cum urmează:
- - **q este funcția de risc a unui episod și are proprietatea**

$$\int_{x \geq 0} q(x) dx = 1$$

- p este funcția de descreștere a duratei de viață pentru un episod:

- $$q^+(t) \stackrel{\text{def}}{=} \int_{x=t+1} q(x) dx$$

- Folosind funcția de risc marginal q^+ , putem reprezenta, pe p conform relației

$$p(t) = 1 - \int_{x=0}^t q(x)dx = 1 - \sum_{i=0}^t q^+(i)$$

În aceste condiții, mărcuș estimată pentru un episod de furt de timp este dată în relația

$$E(S, p) = \sum_{i \geq 0} (t_i \Theta c) p(T_i) = \sum_{i \geq 0} w_i p(T_i)$$



Cursul 6

DSM

- O alta abordare în calculul distribuit este memoria virtuală distribuită (Distributed Shared Memory DSM).
- Din nefericire mașinile paralele cu memorie comună fie au memoria fizică nescalabilă, fie lucrează în regim de transfer de mesaje, lucru care face programarea complexă și greoie.
- DSM oferă posibilitatea scalabilității fără pierderea ușurinței de programare.

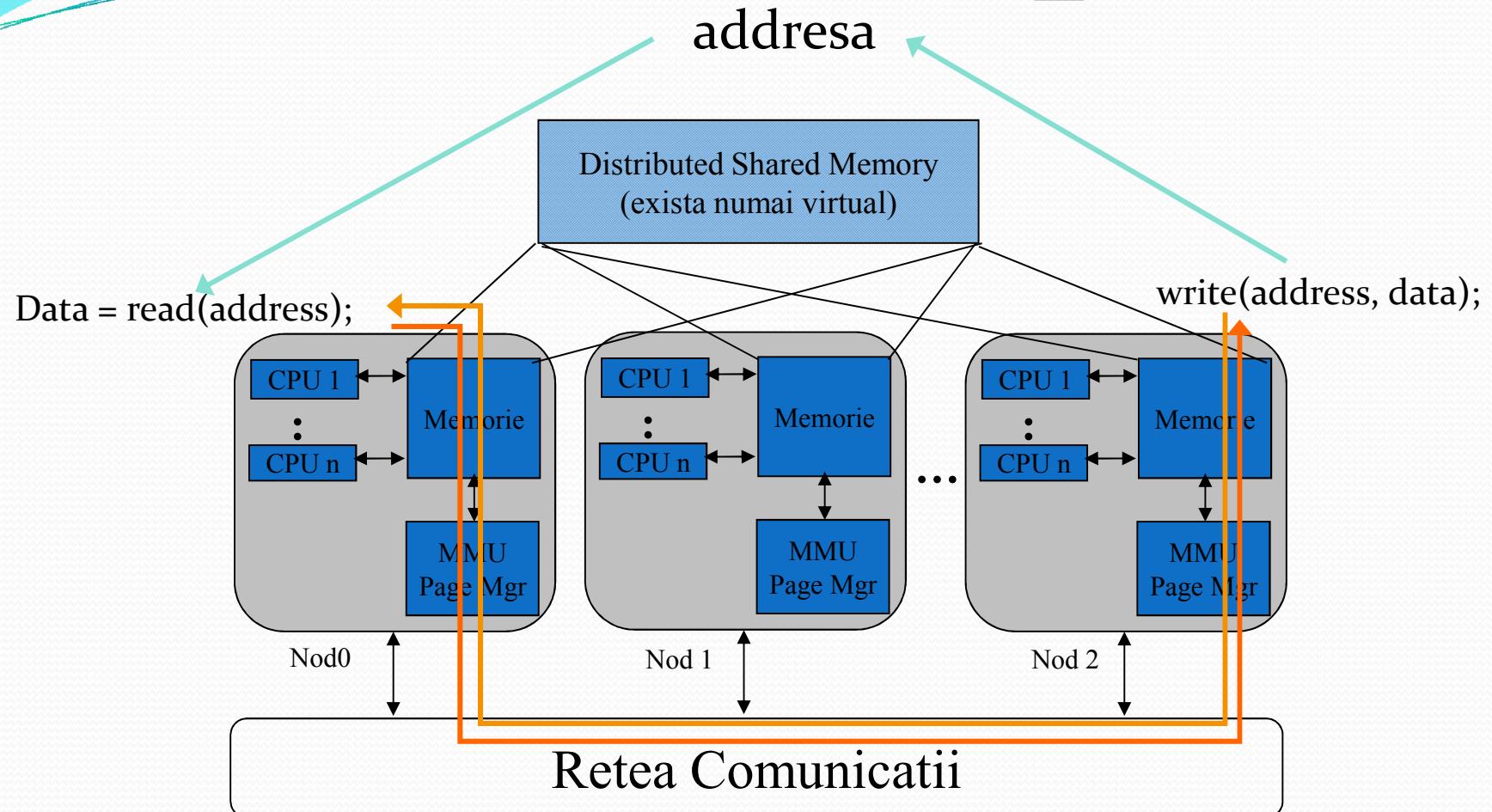
- 1. Un sistem de tip DSM trebuie să aibă cel puțin un model de coerență bine definit.
- 2. De asemenea trebuie să posedă un mecanism explicit de blocare a accesului la date,
- 3. Nu toate sistemele DSM pot fi scalabile.
- 4. Implementarea hard-soft.

- Unele sisteme de tip DSM sunt implementate la nivel hard, altele prin soft.
- Sistemele implementate la nivel hard sunt mai rapide dar cu prețul unei anumite rigidități din punct de vedere al modelelor de coerență suportate.
- Aici apar întrebările curente în cazul proiectării unui sistem:
 - ce compromis se va face ?
 - câte nivele vor fi implementate hard
 - și câte soft
- pentru a obține un sistem cât mai eficient ?.

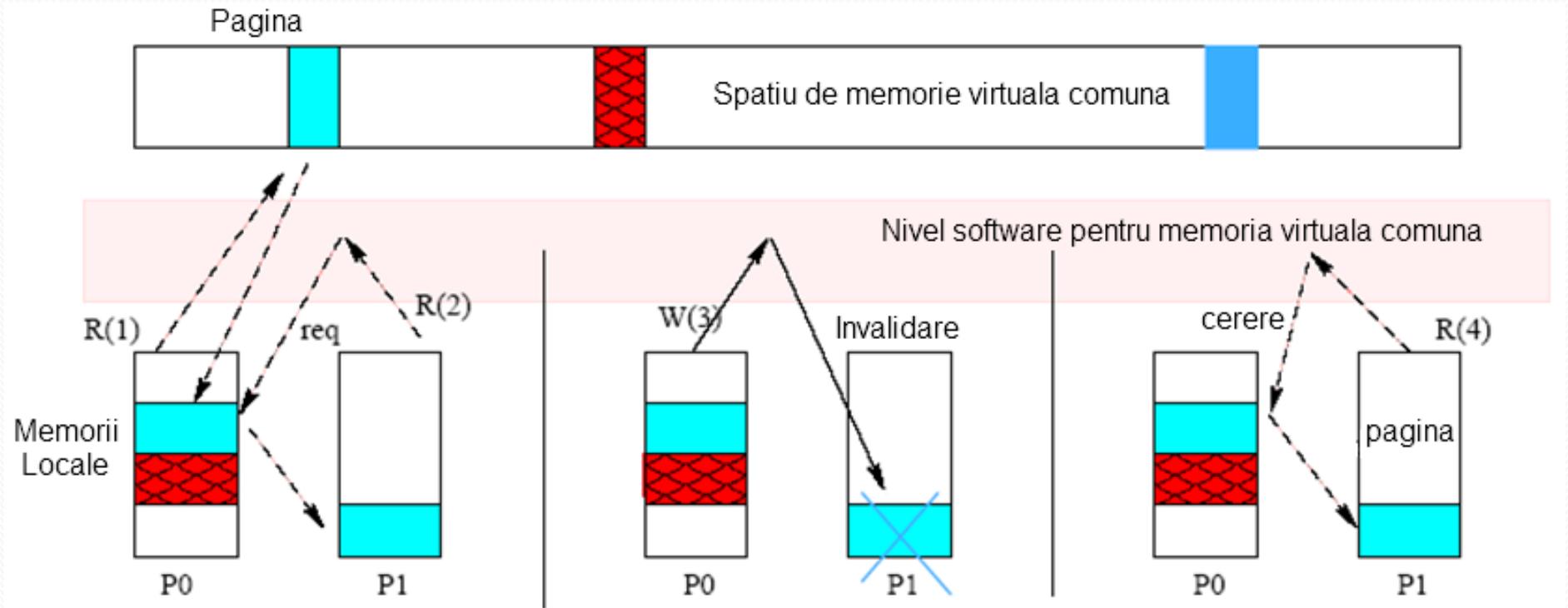
- În principiu, este de așteptat ca performanțele aplicațiilor ce folosesc DSM să fie mai rele decât în cazul folosirii tehnicilor de transfer de mesaje, atâta timp cât transferul de mesaje este o extensie directă a mecanismelor de comunicație sistem și datorită faptului că DSM este implementat ca un strat separat între aplicație și sistemul de transfer de mesaje.
- Totuși câteva implementări ale algoritmilor DSM au demonstrat că acesta poate fi competitiv cu transferul de mesaje în termeni de performanță pentru multe aplicații.

- În plus dacă aplicația are un grad rezonabil de localizare relativ la accesarea datelor, încărcarea de comunicații este amortizată de accesări multiple în memorie, fapt ce reduce accesările de comunicație.
- În al doilea, rând multe aplicații paralele / distribuite se execută în faze iar fiecare fază de calcul este precedată de o fază de schimb de date; timpul necesar schimbului de date este dictat de încărcarea existentă la nivelul comunicațiilor.

Sistemul



Memoria distribuită este simulață



Exemplu proces care scrie in memorie

```
#include "world.h"
struct shared { int a,b; };
```

Program Writer:

```
main()
{
    int x;
    struct shared *p;
    methersetup();          /* Initialize the Mether run-time */
    p = (struct shared *)METHERBASE;      /* overlay structure on METHER segment */
    p->a = p->b = 0;           /* initialize fields to zero */
    while(TRUE) {               /* continuously update structure fields */
        p->a = p->a + 1;
        p->b = p->b - 1;
    }
}
```

Exemplu proces care citeste din memorie

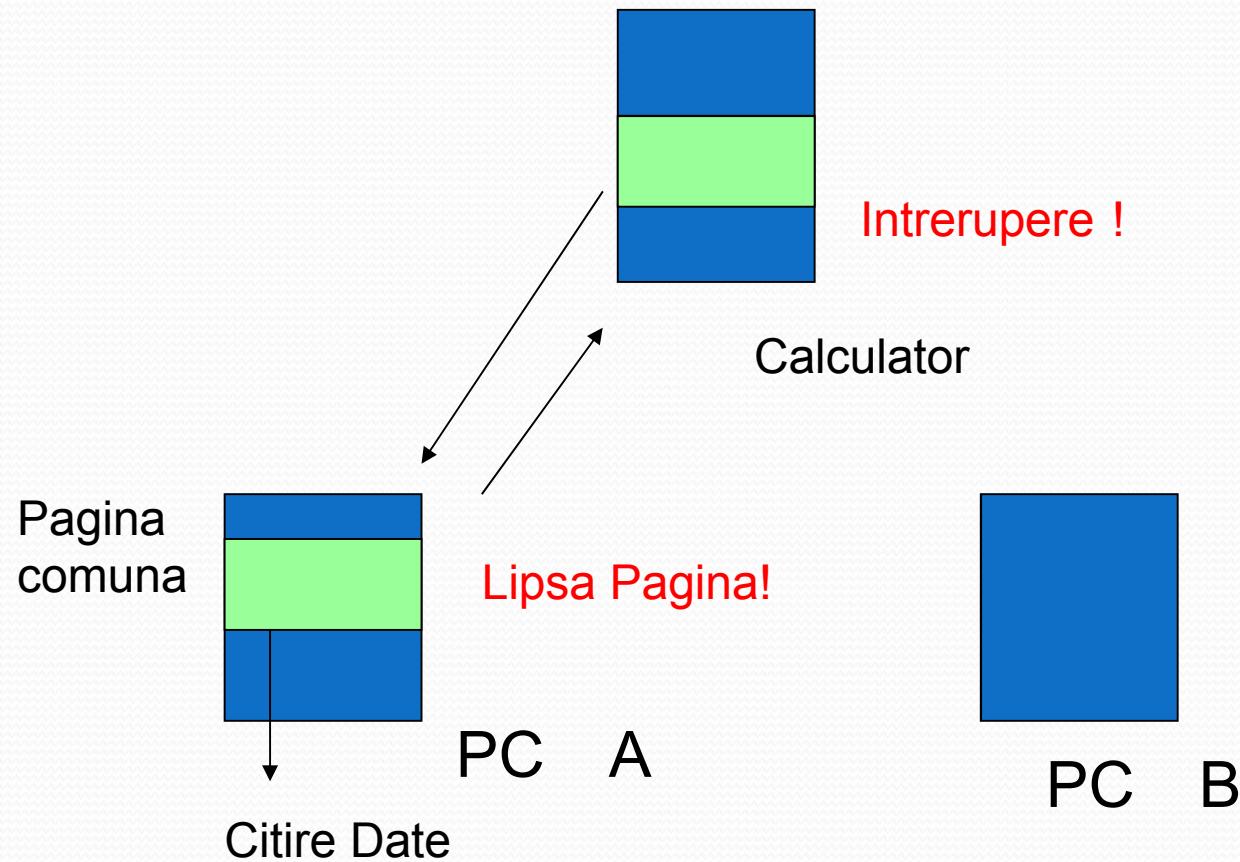
Program Reader:

```
main()
{
    struct shared *p;
    methersetup();
    p = (struct shared *)METHERBASE;
    while(TRUE) { /* read the fields once every second */
        printf("a = %d, b = %d\n", p->a, p->b);
        sleep(1);
    }
}
```

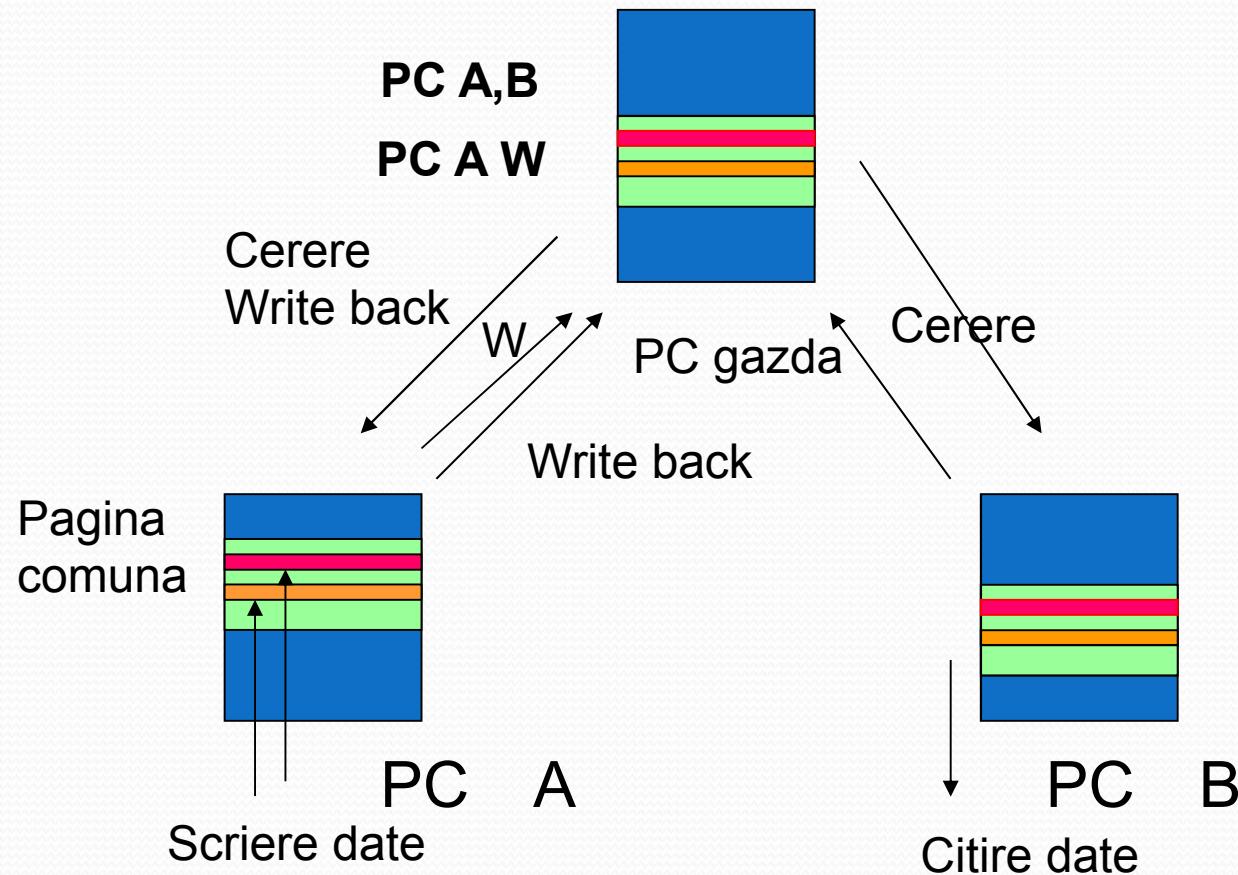
Memorie comuna distribuita realizata software (Virtual shared memory)

- Se foloseste mecanismul de management al memoriei virtuale pentru gestiunea memoriei comune
 - IVY (U.of Irvine), TreadMark (Wisconsin U.)
 -
 -
 -
 -

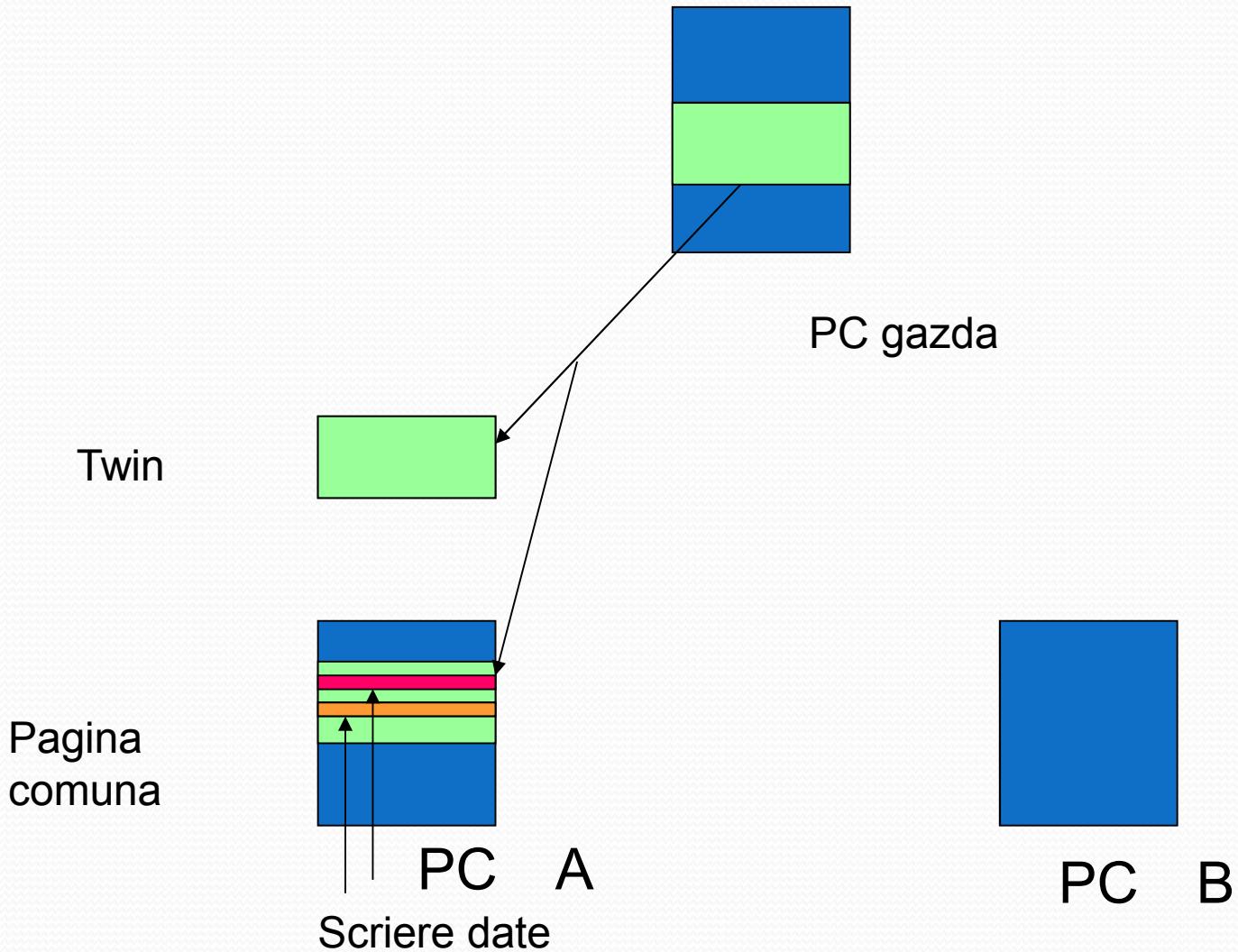
Un exemplu simplu de memorie comună realizata soft



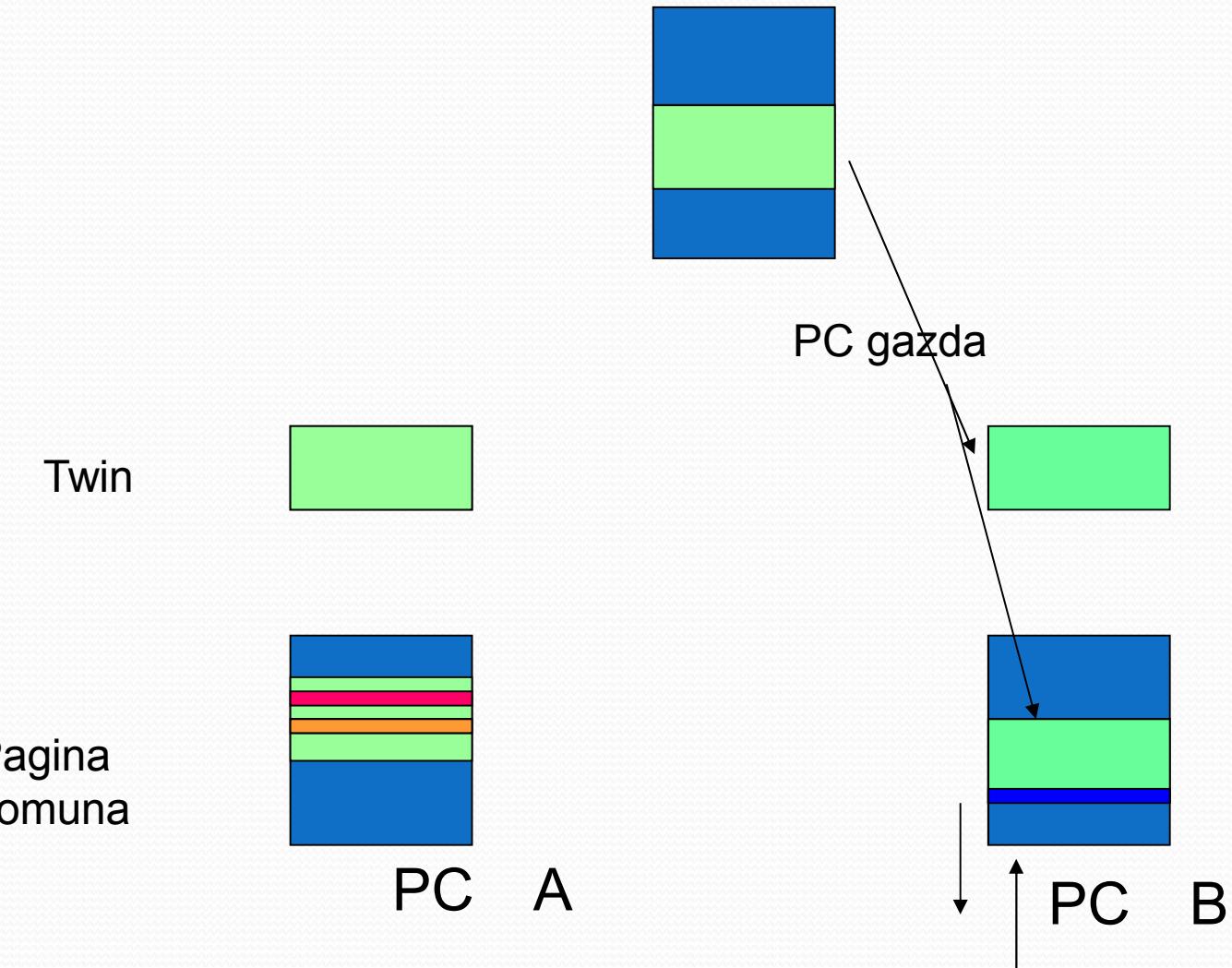
Protocol de tip “Single Writer”



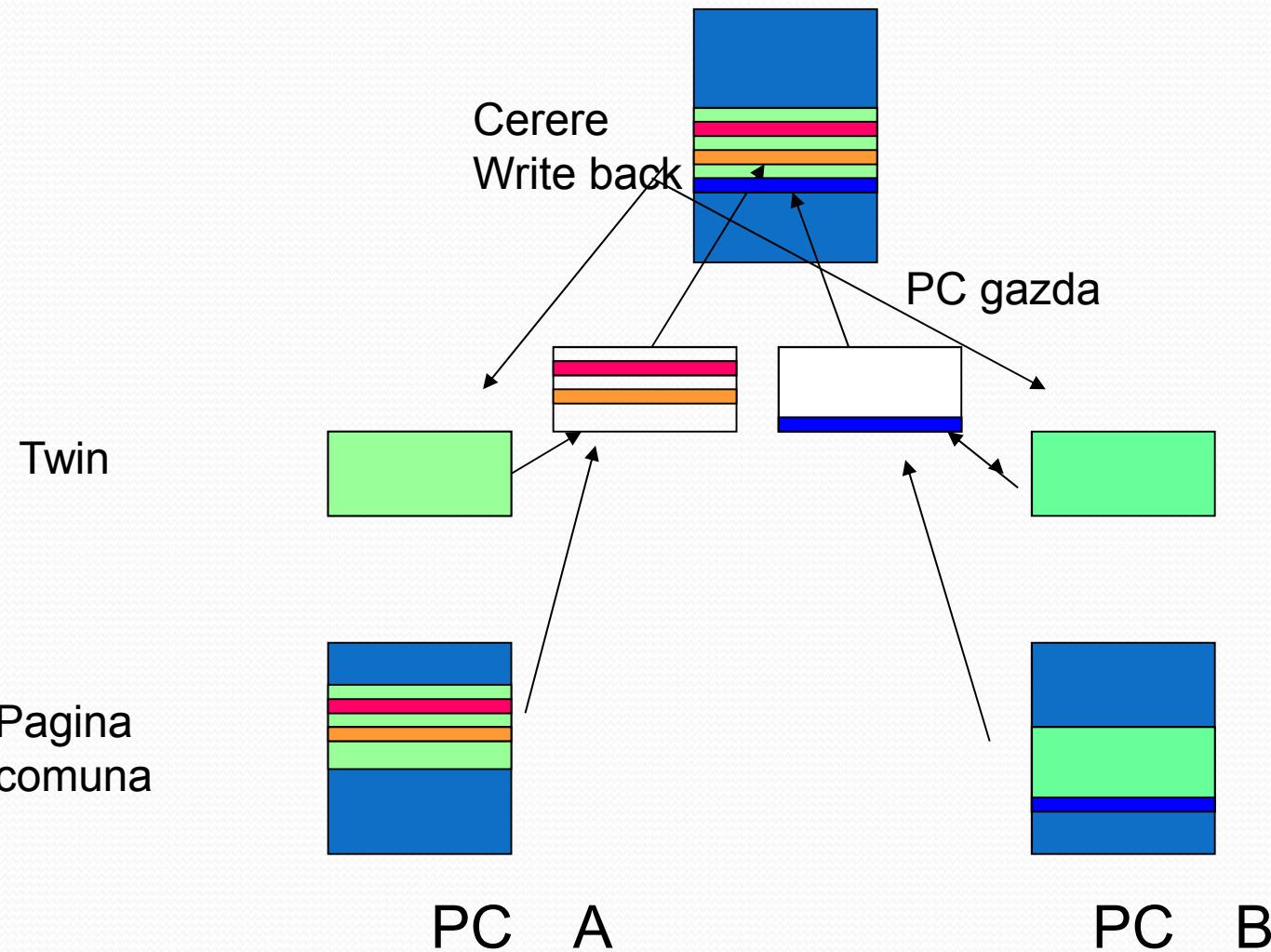
Protocol de tip “Multiple Writers”



Protocol de tip “Multiple Writers”



Protocol de tip “Multiple Writers”



Modele de consistenta a memoriei pentru memoria comuna distribuita implementata software (SDSM)

- SDSM

→ Gestiunea implica resurse procesor
Imposibilitatea gestiunii intreruperilor



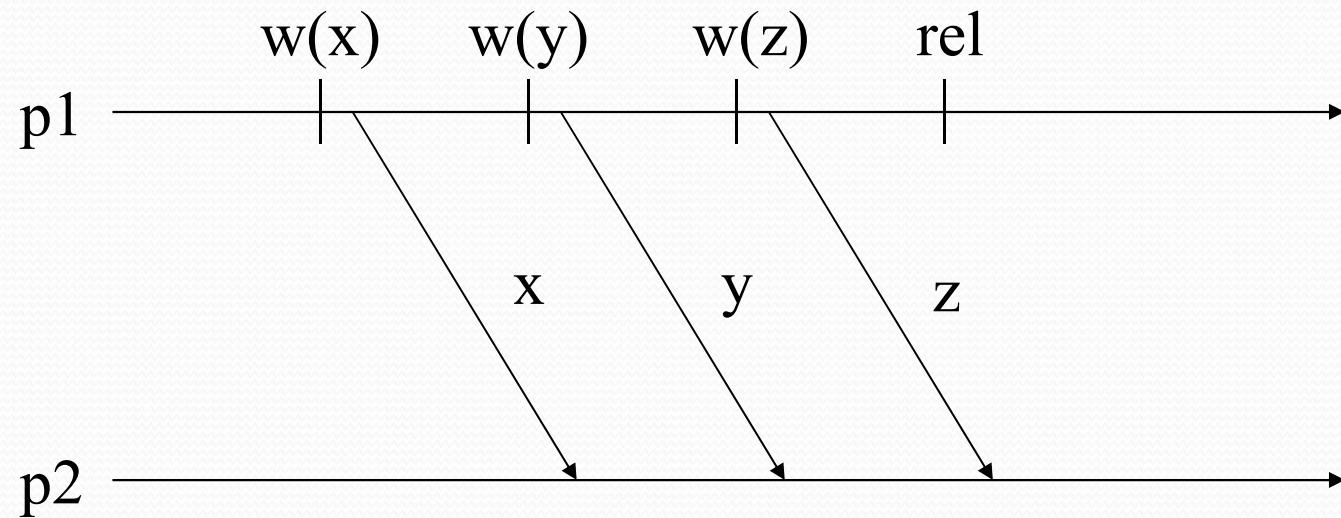
Numarul de mesaje trebuie redus



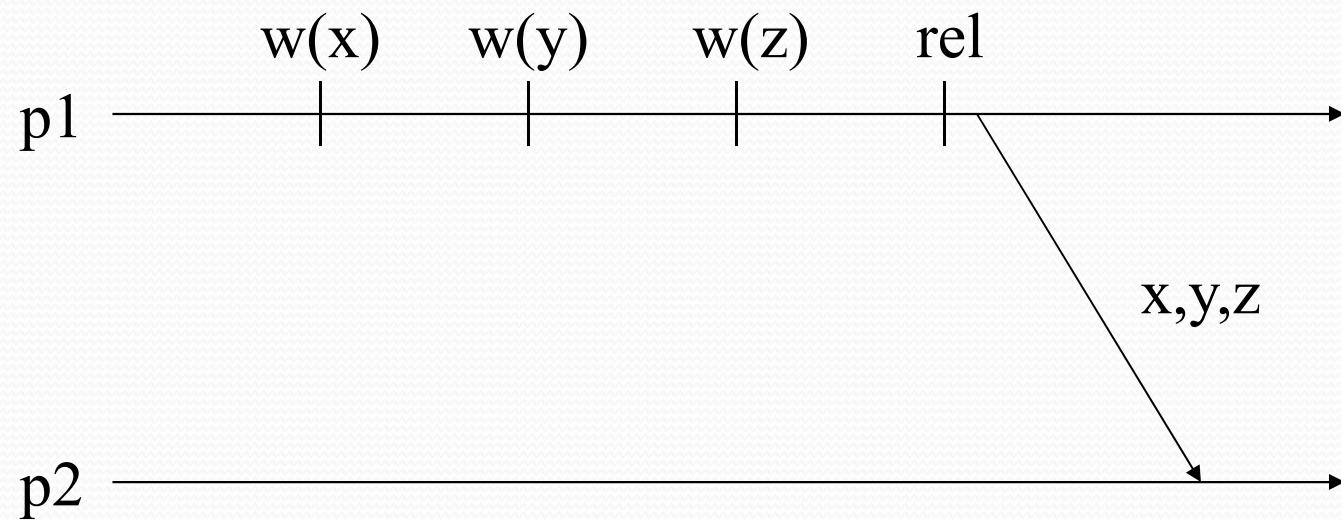
- Modele de consistenta relaxata extinse

- Eager Release Consistency (Munin)
- Lazy Release Consistency (TreadMarks)
- Entry Release Consistency (Midway)

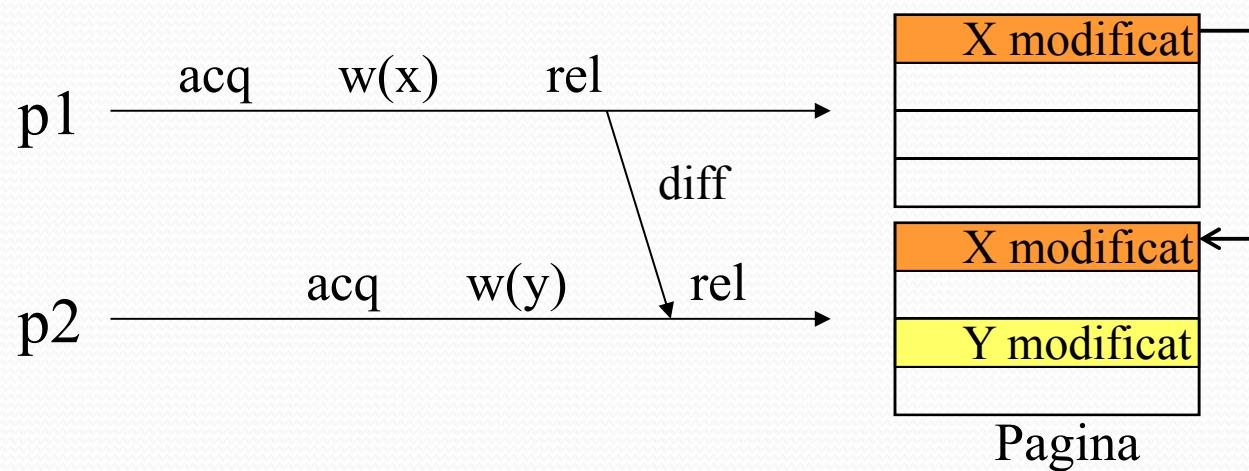
Eager Release Consistency



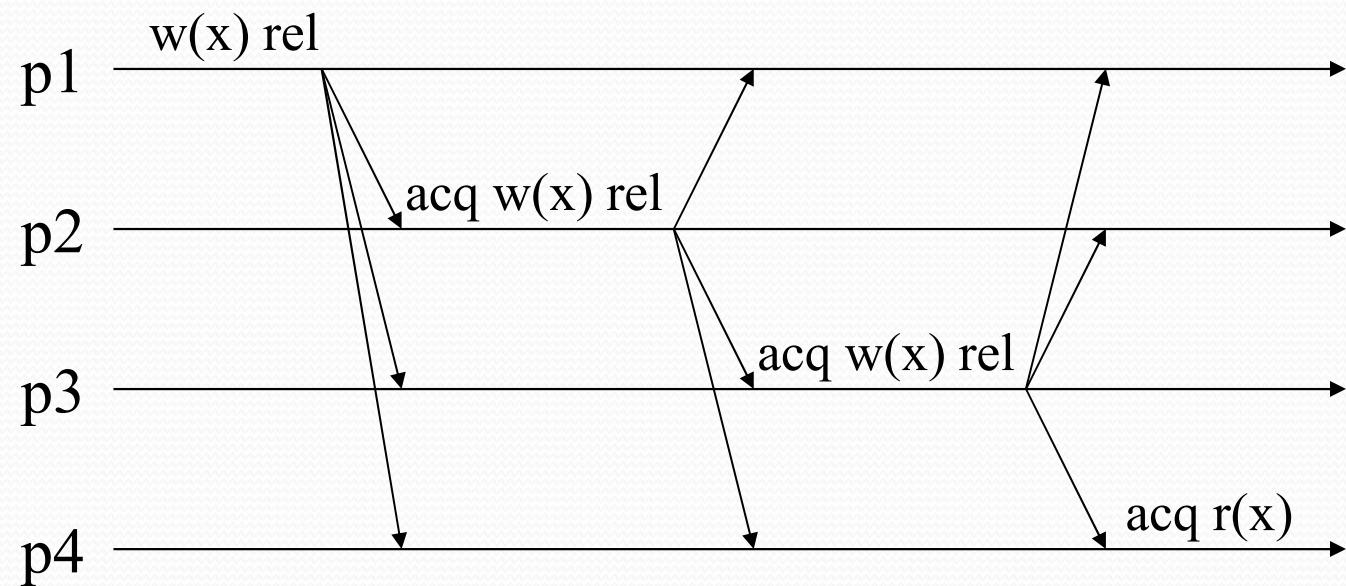
Eager Release Consistency



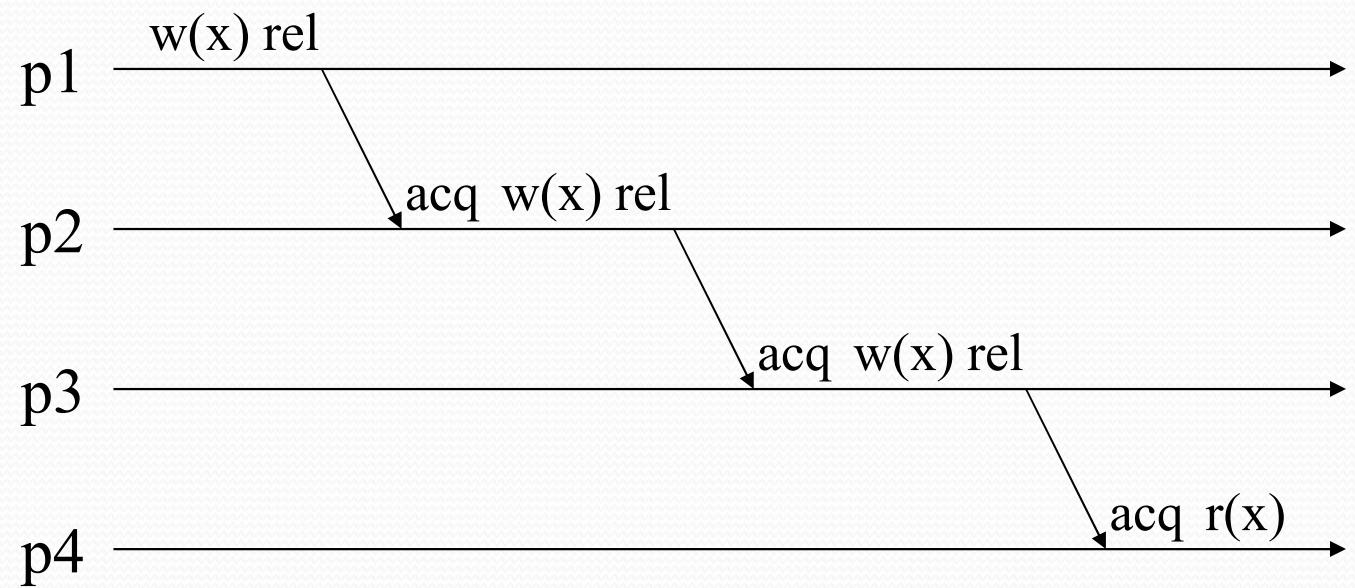
Eager Release Consistency



Lazy Release Consistency



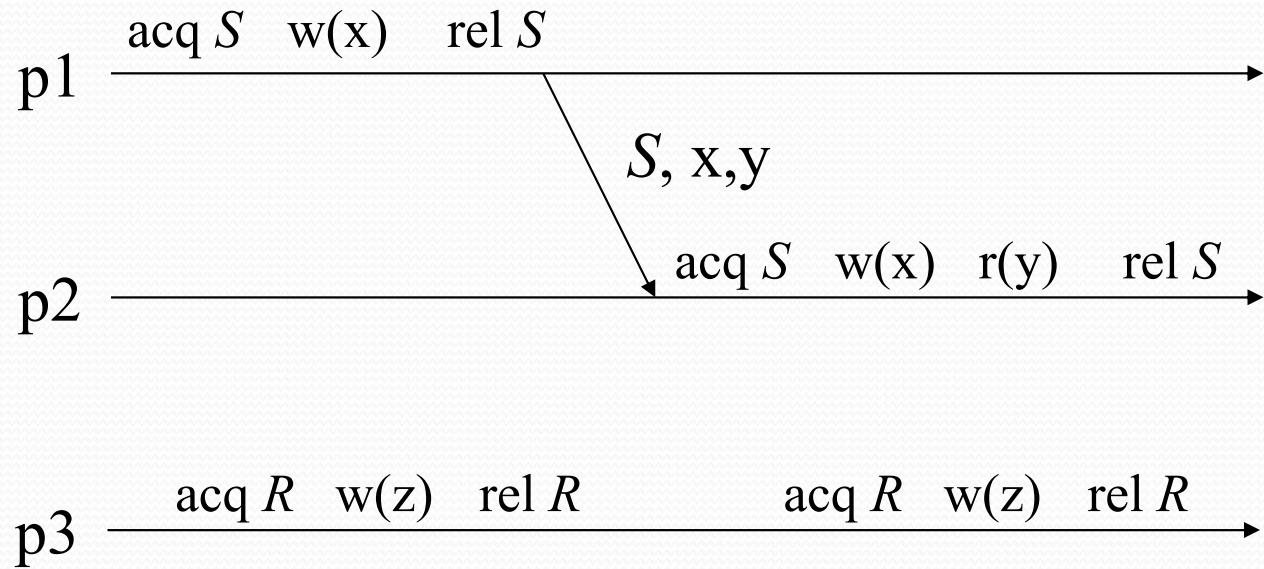
Lazy Release Consistency



Entry Release Consistency

- datele comune si obiectele de sincronizare sunt asociate
- achizitia si eliberarea se executa pe un obiect de sincronizare
- prin introducerea in cache a obiectului de sincronizare
- blocuilor lipsa in cache (miss) vor fi reduse prin realizarea unei asociieri intre obiect si datele comune corespunzatoare

Entry Release Consistency



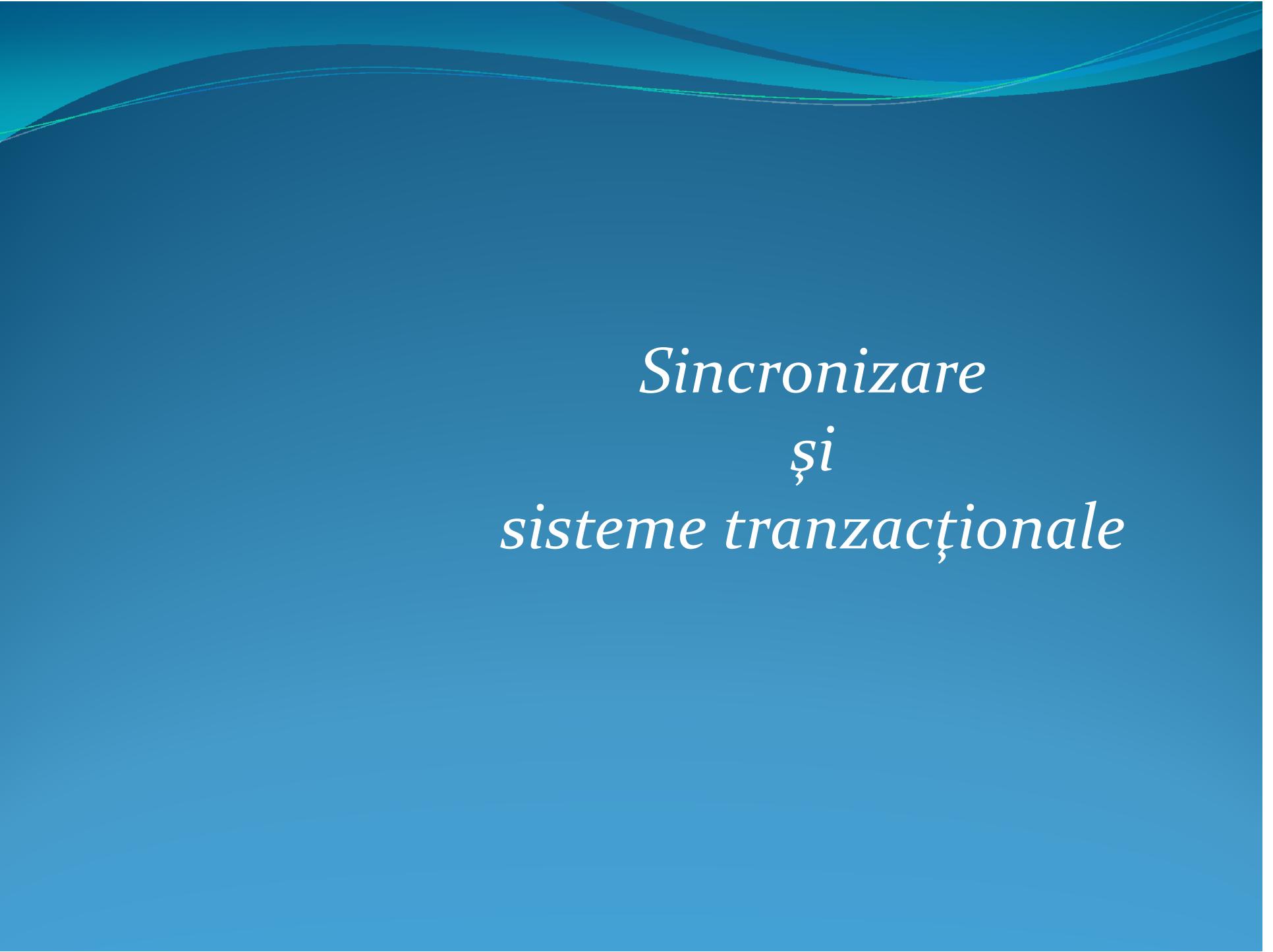
- Cel mai cunoscut algoritm pentru implementarea DSM este cel al lui Li care este foarte bun pentru o clasă mare de algoritmi.
- În algoritmul lui Li cunoscut ca SVM spațiul comun de adrese este partaționat în pagini, iar copii ale paginilor sunt distribuite între gazde, folosind un protocol de tipul multipli cititori/un singur scriitor.

- Un avantaj al algoritmului lui Li este că acesta poate fi integrat ușor în memoria virtuală menținută de sistemul de operare a gazdei.
- Dacă o pagină de memorie comună este stocată local pe o gazdă ea poate fi mapată în spațiul virtual de adrese al aplicației de pe gazdă
- O accesare la o pagina inexistentă (fault page) transferă controlul unui gestionar (handler) de erori de acest tip .

- Protocolul de menținere a coerenței sistemului este similar cu cel folosit la menținerea consistenței memoriilor cache în sistemele multiprocesor cu memorie comună.
- Pentru aplicații paralele și distribuite, DSM poate ascunde complexitatea comunicației față de aplicație
- În acest tip de sisteme, DSM obține transparentă totală

- Eterogenitatea într-un sistem distribuit apare într-un număr de forme.
- Arhitectura hard a mașinilor poate fi diferită, chiar și setul de instrucțiuni, reprezentarea datelor, mărimea paginilor și numărul de procesoare ale gazdei.
- De asemenea sistemul de operare, limbajele de programare și compilatoarele lor, chiar și protocoalele de comunicații pot fi definite.

- Memoria comună distribuită în sisteme eterogene este folositoare pentru aplicații paralele și distribuite în scopul exploatarii resurselor existente pe diverse tipuri de gazde în același timp.
- De exemplu aplicațiile CAM ce realizează controlul unei linii de producție automatizate în timp real.



*Sincronizare
și
sisteme tranzacționale*

Sincronizarea în sistemele distribuite

- Într-un sistem strâns cuplat, regiunile critice, excluziunea mutuală, și alte probleme de sincronizare sunt rezolvate ușual folosind semafoare și monitoare.
- Aceste metode nu se pot aplica prea bine în sistemele distribuite deoarece ele sunt tipice existenței unei memorii comune.
- Una din problemele majore în realizarea unei sincronizări este timpul, deci trebuie realizată o sincronizare a ceasurilor mașinilor din sistem.

Sincronizarea în sistemele distribuite

- În general un algoritm distribuit are următoarele caracteristici:
 1. Informația relevantă este împărtășiată pe diverse mașini;
 2. Procesele iau decizii bazându-se pe informațiile locale;
 3. Un singur punct de cădere în sistem poate fi evitat;
 4. Nu există ceas comun sau o resursă care să genereze un timp global.

Sincronizarea în sistemele distribuite

- Din primele trei puncte rezultă că nu putem face colectare centralizată de informații.
- De exemplu pentru a realiza alocarea resurselor nu este în general folosită metoda trimiterii tuturor cererilor la un singur proces care să le examineze ca apoi să accepte sau să respingă cererile bazându-se pe propriile tabele de date.
- Aceasta deoarece într-un sistem mare s-ar realiza o supraîncărare a respectivului proces, fără a lua în considerare de cazul în care acest proces s-ar bloca și de implicațiile acestui fapt.

Ceasuri logice

- Indiferent cât de precise ar fi ceasurile CMOS, existente pe diverse stații de lucru ce operează într-un sistem distribuit, totuși există diferențe în timp între ele.
- Atât timp cât lucrează numai cu procese și fișiere de pe acea mașină deviația referinței de timp nu deranjează, însă la un sistem distribuit acest lucru conduce la pierderea sincronizării globale.
- O modalitate de a rezolva problema a fost propusă de Lamport.

Ceasuri logice

- De fapt problema este mai complexă.
- Nu se poate renunța complet la sincronizarea temporală dar se poate conveni între mașini că este ora X chiar dacă nu este exact acea oră la toate.
- Pentru clasele de algoritmi care suportă această metodă de lucru acest tip de ceas este denumit ceas logic.
-

Ceasuri logice - Lamport

- Pentru a realiza sincronizarea ceasurilor logice Lamport a definit o funcție de tipul "s-a întâmplat înainte" notată cu \rightarrow .
- De exemplu $a \rightarrow b$, a s-a întâmplat înainte de b.
- Această relație se poate observa în două cazuri:
 - Dacă avem a și b evenimente aparținând aceluiași proces și a apare înainte de b atunci $a \rightarrow b$ este adevărată.
 - Dacă a este un eveniment al unui mesaj transmis de un proces și b este un eveniment al unui mesaj recepționat de alt proces de asemenea $a \rightarrow b$ este adevărată.

Ceasuri logice - Lamport

- Dacă avem nevoie de o cale de a măsura timpul pentru fiecare eveniment a atunci acestuia î se poate asigna o valoare temporală $T(a)$, cu proprietatea că
$$a \rightarrow b \text{ implică } T(a) < T(b).$$
- Lamport a asignat timpi la evenimente. Procesele rulează pe diferite mașini fiecare cu propriul său ceas rulând la viteza proprie mașinii respective.

Ceasuri logice - Lamport

- 1. Dacă a apare după b în același proces:
 $T(a) > T(b);$
 2. Dacă a și b reprezintă emisarea și receptia unui mesaj $T(a) < T(b);$
 3. Pentru toate evenimentele a și b
 $T(a) \neq T(b).$

Ceasuri fizice

- Din motive ca eficiență și redundanță se folosesc ceasuri fizice multiple, lucru care pune două probleme:
 - Cum să realizăm sincronizarea lor cu lumea reală?
 - Cum să le sincronizăm între ele?

Ceasuri fizice

- În primul caz se folosește sincronizarea:
 - cu frecvența de alimentare a rețelei
 - prin recepția semnalelor generate de stații pe unde scurte
 - prin recepția semnalelor generate prin satelit.
- Fiecare mașină are un numărător intern care produce o întrerupere de câte H ori pe secundă. La depășirea variabilei temporale este incrementată.
- Să notăm cu T această valoarea timpului intern și cu $T_p(t)$ timpul notat de mașina p atunci când timpul real este t , deci $T_p(t)=t$ pentru orice p și t.

Ceasuri fizice

- Din punct de vedere matematic dacă există o constantă ρ care satisface relația de mai jos atunci se consideră că lucrăm în cadrul specificat. ρ este specificat de fabricanți și denumit rata maximă de eroare.

$$1 - \rho = \frac{dC}{dt} \leq 1 + \rho$$

- Dacă două ceasuri au eroarea în direcții inverse față de timpul real la timpul Δt după ce au fost sincronizate atunci se vor depărta cu $2\rho\Delta t$.
- .

Algoritmul Cristian

- O metodă de abordare este algoritmul lui Cristian care este folosit în cazul sincronizărilor în Internet folosind un server de timp.
- Este suficientă realizarea periodică de cereri către serverul de ceas pentru a se obține răspunsul cu timpul real.
- Algoritmul prezintă unele probleme.
- Prima este timpul scurs între cerere și răspuns.

Algoritmul Berkeley

- În acest caz există un demon sau server de timp care întreabă din timp în timp toate mașinile din sistem cât este ceasul după care le trimite modificările care trebuie făcute astfel încât să se alinieze la timpul global deci serverul este activ.
- Toate metodele descrise până acum au dezavantajul centralizării.
- Se cunosc și algoritmi descentralizați.

Algoritm descentralizat

- Datorită faptului că mașinile nu lucrează la aceeași viteză emisia acestui timp nu se petrece simultan.
- După ce emite timpul propriu ea pornește un ceas local pentru a colecta toate mesajele care apar în timpul unui interval S.
- Când au venit toate mesajele se calculează un nou timp din valorile primite.
- Algoritmul cel mai simplu este realizarea mediei timpilor recepționați.

Tranzacții atomice

- Majoritatea tehniciilor folosite în realizarea sincronizării sunt de nivel scăzut și necesită cunoștințe detaliate ale programatorului.
- În aceste condiții necesitatea unui mediu de programare mai eficient, care să permită programatorului focalizarea pe algoritm și modul de interacțiune a proceselor, este necesar.

Model tranzactional

- Acet model este compus din:

- Sistem
- Procesor
- Comunicatie
- Persistenta

Primitivele tranzacțiilor

- Aceste elemente de bază trebuie să existe implementate fie la nivelul sistemului de operare, fie să fie puse la dispoziție de bibliotecile limbajului.

Primitivile tranzacțiilor

- Termenul de tranzacție provine din cauza faptului că modelul tranzacțional este provenit din analiza situațiilor existente la nivelul schimburilor economice
- Vom prezenta mai jos câteva din cele mai importante primitive atomice:
 - BEGIN_TRANSACTION
 - END_TRANSACTION
 - ABORT_TRANSACTION
 - READ
 - WRITE

Primitivele tranzacțiilor

- Denumirea de primitive provine din faptul că indiferent de tipul și caracteristicile unei tranzacții aceste operații sunt general întâlnite.
- Într-un sistem tranzacțional execuția unei tranzacții trece printr-o serie de etape:
 - stabilirea condițiilor
 - verificarea validității acestora
 - execuția propriu zisă a tranzacției.

Proprietățile tranzacțiilor

- Din punct de vedere formal orice tranzacție se consideră a fi corect definită dacă ea respectă următoarele proprietăți.
 - ◆ Atomicitatea:
 - ◆ Consistența:
 - ◆ Izolarea:
 - ◆ Durabilitatea:
 - ◆ În literatura de specialitate aceste proprietăți sunt cunoscute sub denumirea de ACID.

Folosirea spațiului privat de lucru

- La inițializarea unei tranzacții de către un proces acesta va avea un spațiu privat de lucru care conține copii ale tuturor fișierelor sau/și obiectelor necesare.
- Această abordare necesită resurse foarte mari de calcul.

Folosirea spațiului privat de lucru

- Chiar și această abordare necesită un număr mari de replicări ale fișierelor.
- Cum sistemele tranzacționale sunt specifice bazelor de date este de dorit ca să fie gestionate cât mai puține copii.
- De aceea în unele cazuri nu se mai realizează copierea întregului fișier ci numai a indexului acestuia.

Metoda jurnalului posibilei execuții (“writeahead log”)

- Această metodă mai este numită uneori și cea a listei de intenții.
- Idea de pornire provine din faptul că terminarea corectă a unei tranzacții este mult mai probabilă decât cazul contrar.
- De asemenea necesită resurse mult mai mici de calcul.

Metoda jurnalului posibilei execuții (“writeahead log”)

- Dacă o tranzacție a avut succes atunci acest lucru este specificat în acest jurnal al execuției și se continuă.
- În caz contrar starea anterioară este refăcută pe baza informațiilor existente.
- Această metodă de restaurare a stării anterioare este cunoscută și sub numele de “roll back”.

Protocol cu execuția în două etape

- Această abordare propusă de Gray în 1978 este tipică implementării tranzacțiilor în cazul sistemelor distribuite și este și cel mai folosit de proiectanții de sisteme tranzacționale.
- Procesul care inițiază tranzacția este ales de obicei coordonator.
- Desfășurarea protocolului începe atunci când procesul coordonator face o mențiune în jurnalul execuției prin care anunță că aceasta s-a demarat.
- Totodată el își va anunța și copiii de acest lucru.

Controlul concurenței în sistemele tranzacționale

- Una din metodele clasice este interzicerea accesului.
- Este simplă și eficientă însă nu are o foarte bună scalabilitate.
- Când un proces preia controlul unui obiect sau fișier în cursul unei tranzacții el blochează accesul la această resursă.
- Acest lucru poate fi realizat folosind abordare centralizată sau distribuită parțial sau total funcție de sistemul de operare.

Controlul concurenței în sistemele tranzacționale

- Își acordă funcție de sistemul de operare fișierul poate fi tratat la nivel general sau pe blocuri.
- Cu cât granularitatea blocajului este mai mare cu atât se obține o concurență mai eficientă.
- Totuși accesul la nivel de bloc va crește mult numărul de blocaje controlate de sistemul de operare fapt ce va conduce la supraîncărcarea acestuia și de asemenea crește și riscul de cădere.

Controlul concurenței în sistemele tranzacționale

- Abia după aceea începe execuția.
- Eliberarea accesului poate fi făcută treptat pe măsură ce nu mai este nevoie de respectivele fișiere sau în aceeași manieră o dată tranzacția încheiată.
- Deși are avantajul că garantează menținerea proprietăților ACID metoda poate conduce la blocarea aplicației.

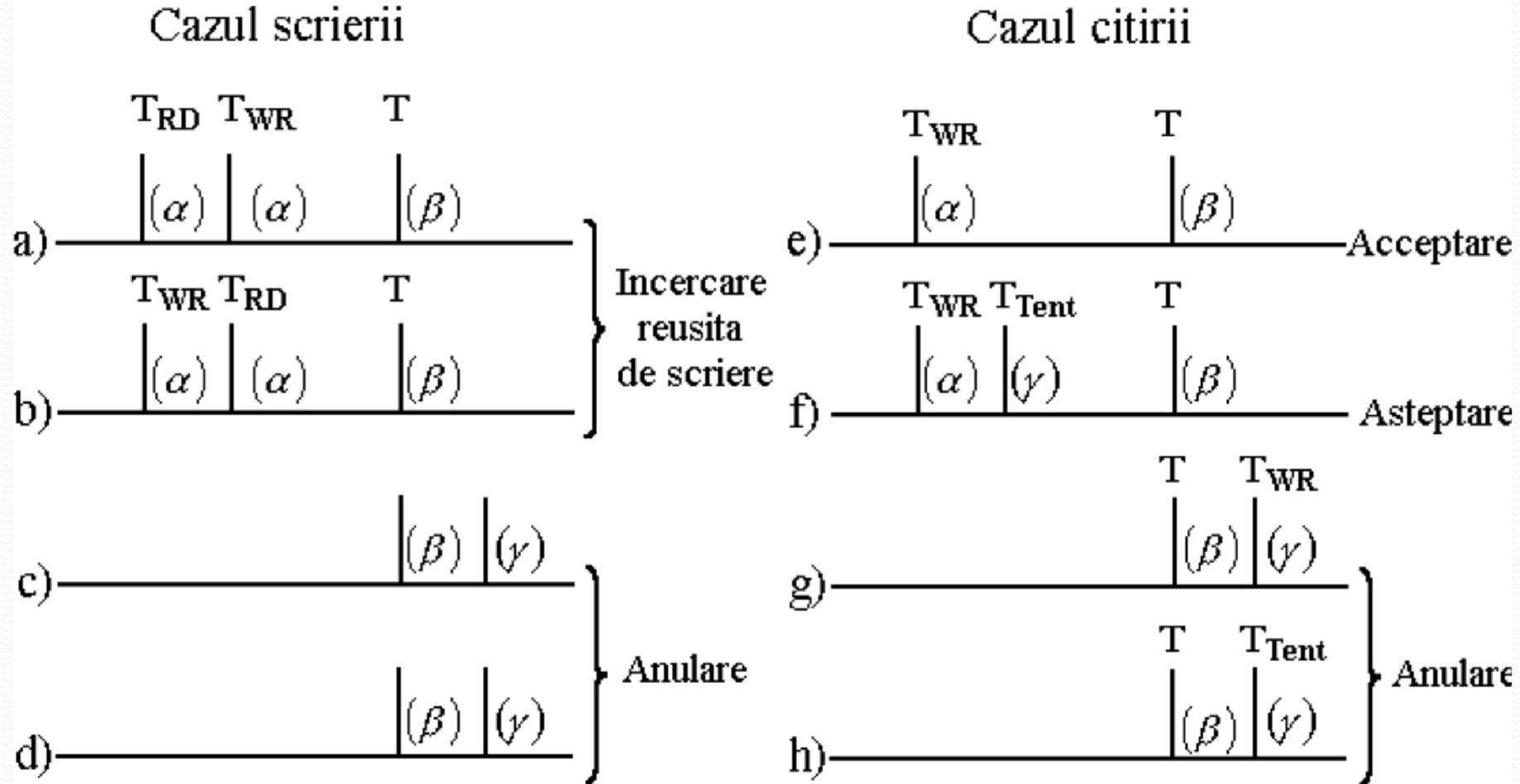
Controlul concurenței în sistemele tranzacționale

- O altă cale de a rezolva această problemă este cea propusă de King și Robinson în 1981 și se bazează pe analiza probabilității de apariție.
- Se verifică doar dacă e posibil să apară un conflict.
- Pentru aceasta la setul propriu de condiții mai apare una implicită pentru toate și anume verificarea dacă există cel puțin un fișier modificat după inițierea tranzacției.

Controlul concurenței în sistemele tranzacționale

- Eficiența ei este dată de rata de apariție a accesului simultan la un fișier ceea ce înseamnă că în sistemele tranzacționale foarte încărcate, ea va scădea.
- Mai mult în cazul încărcărilor mari se va ajunge la dominarea încercărilor de reluare a unor tranzacții.

Controlul concurenței în sistemele tranzacționale



Referinte

- www.cs.iit.edu/~cs550/lectures/10_distributed_shared_memory.ppt
- [Distributed Shared Memory](#)
- <http://courses.washington.edu/css434/slides/DSM.ppt>
- <ftp://ftp.cs.wisc.edu/pub/techreports/1989/TR825.pdf>

Cursul

Cluster

De ce calcul paralel ?

- Pentru că efortul considerabil de cercetare și dezvoltare a produs:
 - ✉ arhitecturi performante
 - ✉ sisteme de operare adecvate
 - ✉ limbaje și medii de programare/dezvoltare de aplicații.
- dezvoltarea semnificativă în domeniul rețelelor creează premizele calculului eterogen.

Paradoxul calculului paralel

- ➡ Dezvoltarea unei aplicații paralele complexe consumă jumătate din timpul de viață al unui sistem paralel.

De aceea,

- ➡ Sunt necesare soluții noi, mai ieftine și cu un grad mai mic de perisabilitate.
- ➡ Noroc caexistă rețelele de calculatoare !

Justificare apariției conceptului de cluster

- Studiile arată că utilizarea ciclurilor de procesor pentru stațiile de lucru este sub 10%.
- Performanțele stațiilor de lucru și PC-urilor cresc rapid.
- În timp ce performanțele cresc, procentul de utilizare va descrește și mai mult.
- Organizațiile nu încurajează achiziționarea de supercomputere datorită costului mare și timpului de viață scurt.

Cluster

- *O colecție de stații de lucru sau PC-uri, interconectate.*
- *O colecție de calculatoare interconectate, lucrând împreună ca o resursă unică de calcul.*

Cluster management software

- Asigură în primul rând disponibilitatea resurselor de calcul și creșterea indicelui de utilizare.
- Cele mai importante rezultate sunt:
 - ◆ biblioteca PVM (Parallel Virtual Machine), dezvoltată la Oak Ridge National Lab
 - ◆ standardul MPI (Message Passing Interface), care are mai multe implementări
 - ☞ – MPICH (Argonne National Lab),
 - ☞ LAM (Ohio Supercomputing Centre),
 - ☞ CHIMP (Edinburgh Parallel Computing Centre) și...

- Există mai multe concepte nou introduse
 - ◆ *mașină paralelă virtuală*,
 - ◆ *domeniu de execuție*, pentru a preciza resursele de calcul din rețea care participă la execuția unei aplicații,
 - ◆ *cluster*.
- Un *cluster* (ciorchine în traducere liberă) de calculatoare reprezintă alternativa cea mai ieftină față de un calculator masiv paralel.
- Elementele ce diferențiază cluster-ul, în cadrul soluțiilor arhitecturale multi-calculator, sunt

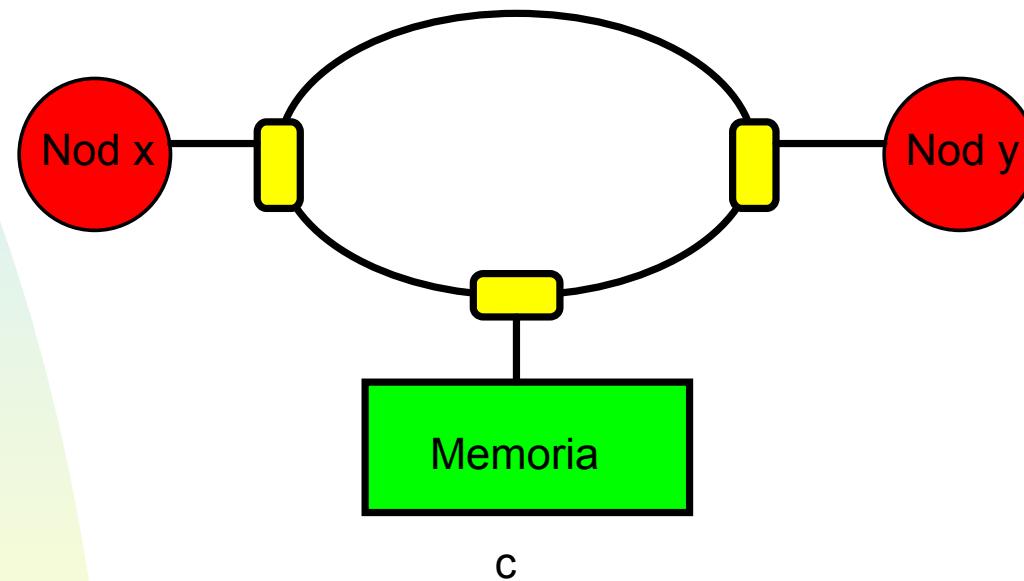
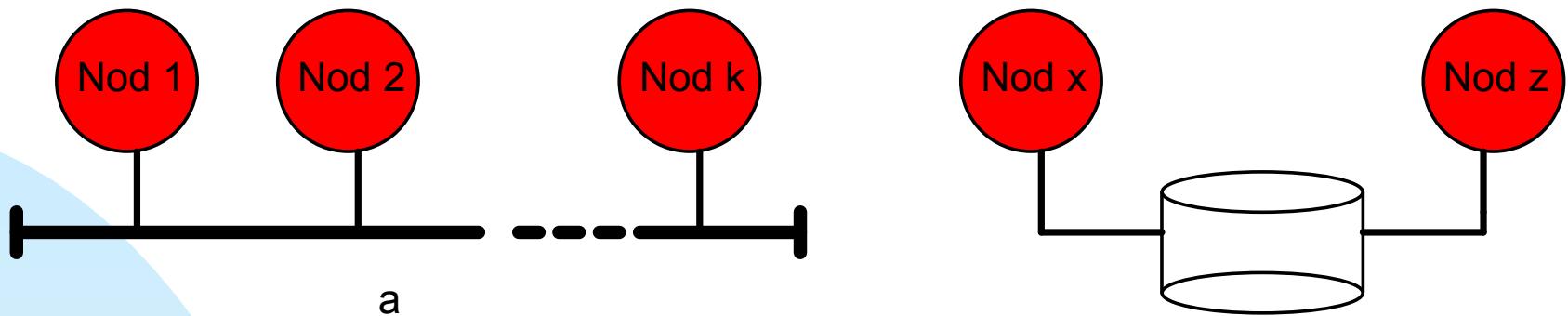
- ◆ *fiecare nod este un calculator de sine stătător, nodurile comunică printr-o rețea obișnuită, de exemplu ethernet, deși există și cluster-e (produse comerciale) care folosesc rețelele de mare performanță;*
- ◆ *interfața de rețea este atașată magistralei de I/E și nu celei a memoriei;*
- ◆ *există un disc local la nivelul fiecărui nod;*
- ◆ *cluster-ul este administrat ca o resursă unică de calcul,*

- ◆ *disponibilitate ridicată*
- ◆ *performanță foarte bună*
- Proprietatea cea mai importantă a calculatoarelor care formează un cluster este că, în timp ce pot fi folosite și de sine stătător, cel mai adesea sunt utilizate ca un ansamblu de calculatoare ce formează o resursă unică de calcul.
-

Atributele folosite la clasificarea cluster-elor

Atribut	Valoare	
Asamblare	compact	repartizată
Control	centralizat	descentralizat
Omogenitate	omogen	eterogen
Securitate	închisă	expusă
Exemplu	cluster dedicat	cluster de întreprindere

- **Asamblarea.** Nodurile unui cluster pot fi plasate toate într-un rack, sau în mai multe rack-uri care se află în aceeași cameră și nu au periferice.
- **Controlul** poate fi centralizat, mai ales pentru un cluster compact, sau descentralizat.
- Comunicația din interiorul unui cluster poate fi **expusă**, sau **închisă**.



- Variante de comunicare a nodurilor unui cluster:
 - ◆ a. comunicare prin rețea,
 - ◆ b. partajarea unui disc,
 - ◆ c. memorie comună.

- Nodurile unui cluster pot fi interconectate prin
 - ◆ legarea la rețea (figura 1, a),
 - ◆ prin partajarea unui disc figura 1.b
 - ◆ prin partajarea memoriei ca în figura 1.c.
- Caracteristicile de cluster care sunt date de prezența mecanismelor pentru:
 - ◆ asigurarea *disponibilității*;
 - ◆ crearea *imaginii unice de sistem*;
 - ◆ *gestiunea job-urilor și a utilizatorilor*;
 - ◆ *comunicații eficiente*.
-

Suportul pentru disponibilitate

- Pentru un sistem robust, cu un grad înalt de disponibilitate se consideră necesară asigurarea unor valori ridicate pentru trei parametri
 - ◆ *fiabilitatea*
 - ◆ *disponibilitatea*
 - ◆ *servisabilitatea*

- Există două posibilități de creștere a disponibilității unui sistem
 - ◆ prin creșterea timpului mediu până la pană (MTTF – mean time to failure),
 - ◆ reducerea timpului mediu de reparație (MTTR - mean time to repair).

tehnici pentru asigurarea disponibilității, folosite la cluster-e.

Redundanța izolată

- Este o tehnică elementară este de a folosi componente redundante.
- Când una cade, funcția sa este preluată de alta.
- Aceste două componente trebuie izolate, în sensul că nu trebuie să poată fi afectate de aceeași sursă de cădere.
- Această soluție are avantaje multiple, printre care și acela că cele două componente se pot testa reciproc și că în timp ce una funcționează, cealaltă poate fi reparată.

- Un alt exemplu este cel al strategiei de proiectare a sistemelor software *critice N-version programming* (NVP).
- Soft-ul este implementat de N echipe izolate, care folosesc algoritmi, limbaje, medii de programare și platforme diferite.
- Într-un sistem tolerant la defecte, cele N versiuni se execută simultan, iar rezultatele lor se confruntă periodic.

Preluarea (failover)

- Când o componentă cade, restul sistemului preia serviciile oferite de componenta respectivă.
- Un mecanism de preluare trebuie să asigure diagnoza, notificarea și refacerea.
- O tehnică comună pentru diagnoză este *heartbeat*.
- Nodurile își trimit mesaje heartbeat.

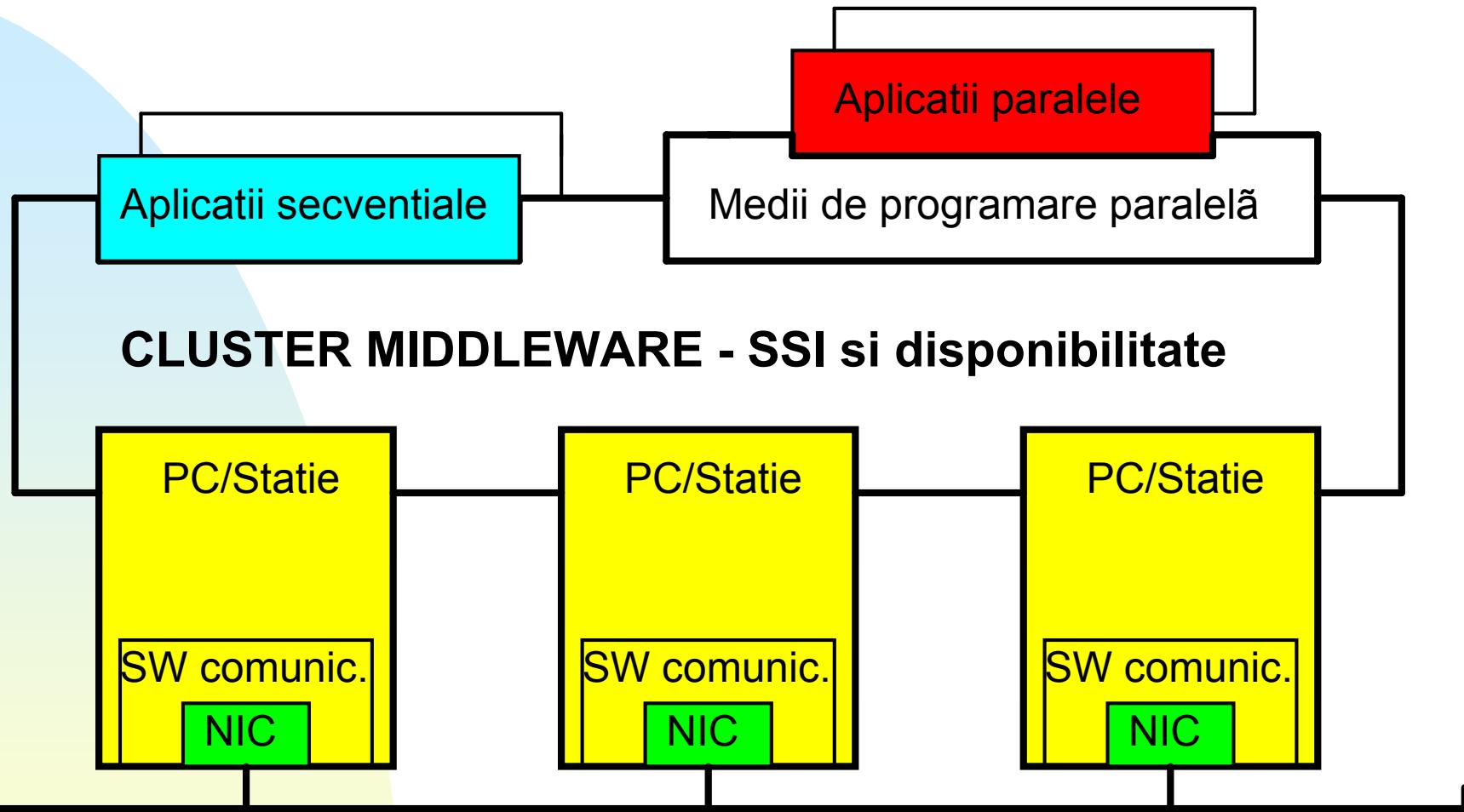
Scheme de refacere (recovery schemes)

- Există două tehnici de refacere.
 1. constă în salvarea periodică a stării proceselor ce se execută (backward recovery - checkpoint).
 2. Dacă timpul de execuție este critic, se poate folosi o schemă de *forward recovery*.

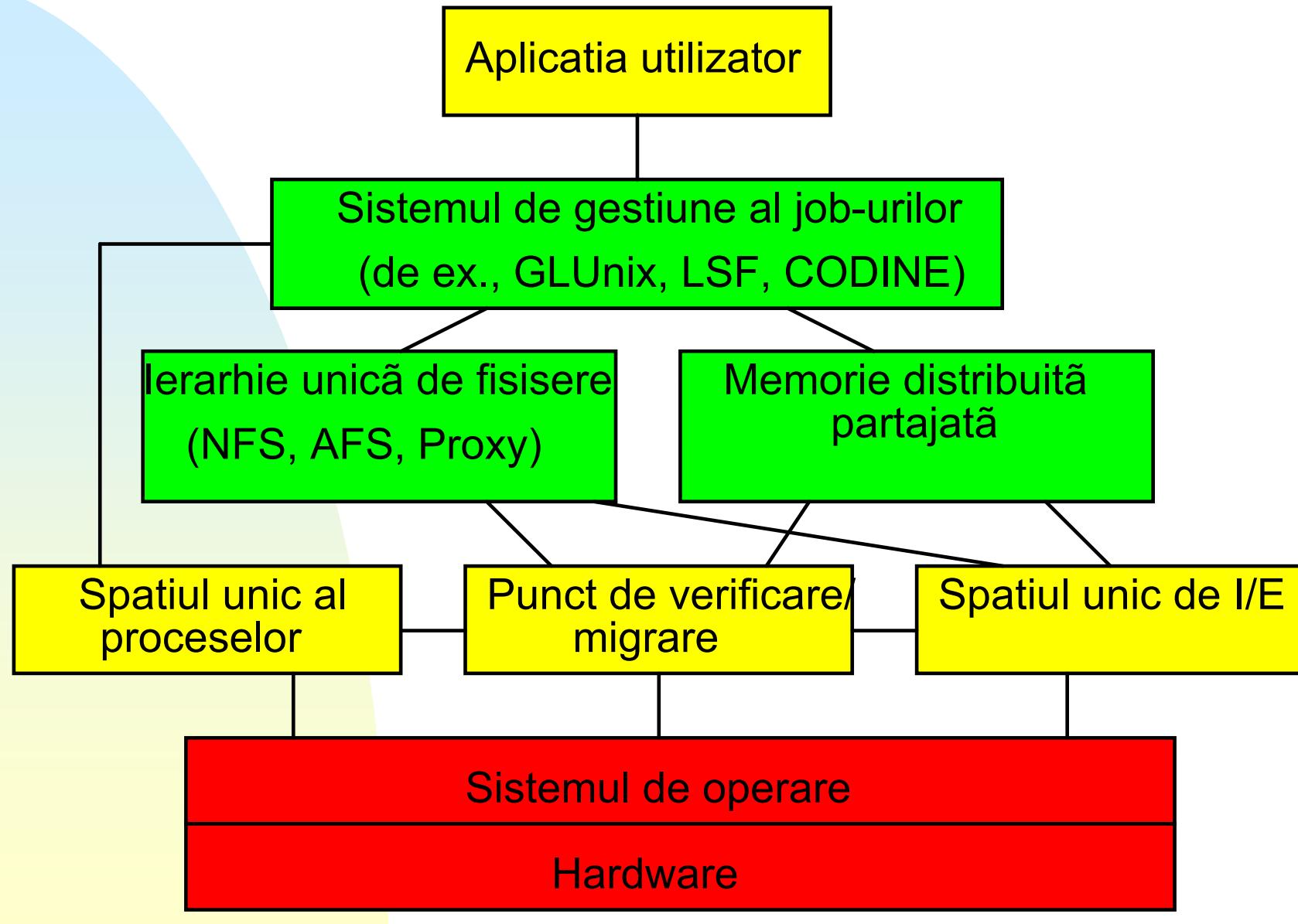
Imagine unică de sistem

- Illuzia imaginii unice a sistemului poate fi obținută la diferite nivele:
 - ◆ **nivelul aplicației** –
 - ◆ **nivelul hardware sau kernel** –
 - ◆ **nivelul situat deasupra kernel-ului** –

Arhitectura unui cluster



Relațiile dintre componentele software/hardware, la un nod dintr-un cluster



- **Punct de intrare unic** înseamnă că un utilizator se poate conecta la un cluster ca la un singur calculator (gazdă).
- **Ierarhie unică de fișiere** înseamnă producerea unei imagini care include discurile locale, globale sau alte dispozitive.

- **Punct de control unic** înseamnă că administratorul de sistem poate configura, monitoriza, testa și controla atât cluster-ul, cât și fiecare nod în parte, de la un singur punct.
- **Spațiul de memorie unic** produce iluzia unei memorii principale mari, care în realitate este un ansamblu de memorii locale.

Gestiunea job-urilor în cluster-e

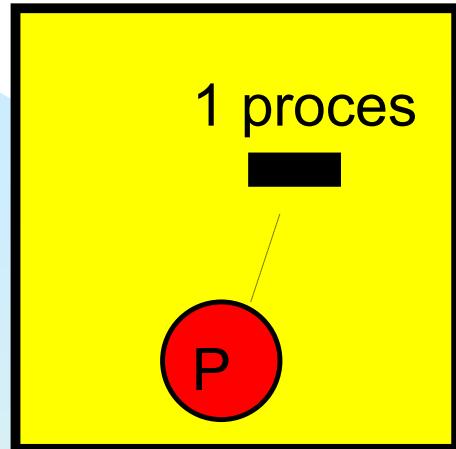
- Un sistem de gestiune a cluster-elor trebuie să conțină trei componente,
 - ◆ *un server pentru utilizatori,*
 - ◆ *un planificator de aplicații/job-uri,*
 - ◆ *un gestionar de resurse,*

- Pe un cluster se pot executa mai multe tipuri de job-uri.
 - ◆ Un *job serial* se execută pe un singur nod,
 - ◆ *job-urile paralele* folosesc mai multe noduri
 - ◆ pe cluster se execută și job-uri care sunt lansate local, neintrând în interacțiune cu sistemul de gestiune al cluster-ului.
 - ◆ Un astfel de job, denumit local sau străin, aşteaptă o tratare imediată.

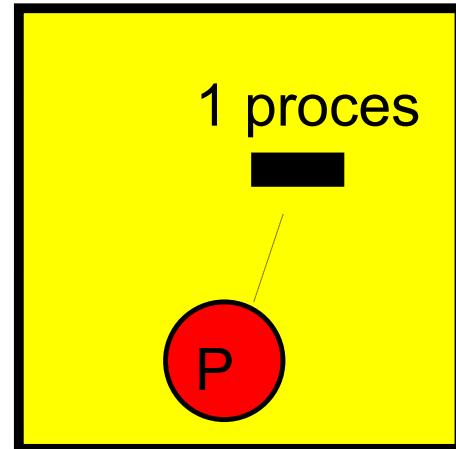
modul dedicat

- În *modul dedicat*, la un moment dat pe cluster se execută un singur job și cel mult un proces al job-ului este alocat la un moment dat unui nod (procesor).
- Acest job se execută până la terminare, după care un alt job poate prelua controlul.
- În acest caz, job-ul este considerat de prioritate maximă, primind în consecință toate resursele cluster-ului.

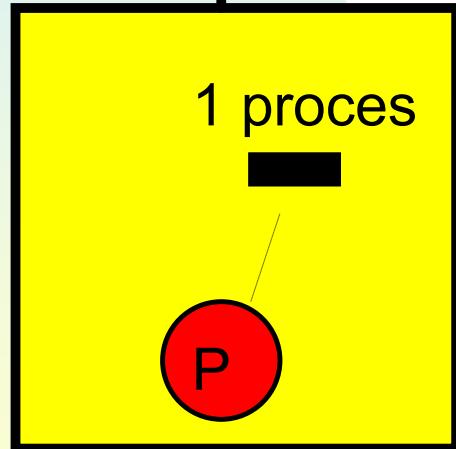
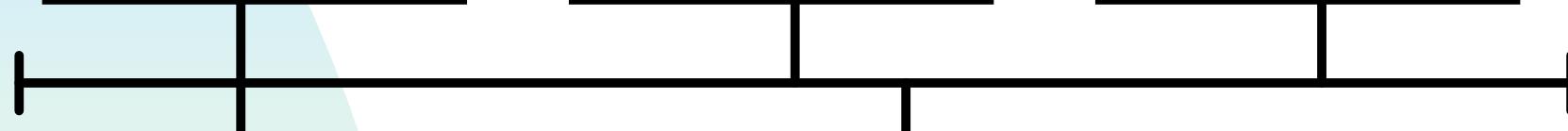
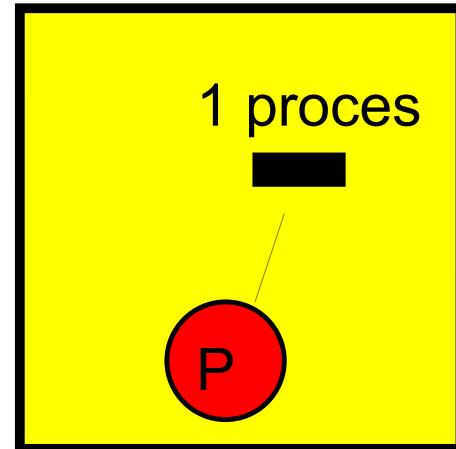
Statia #1



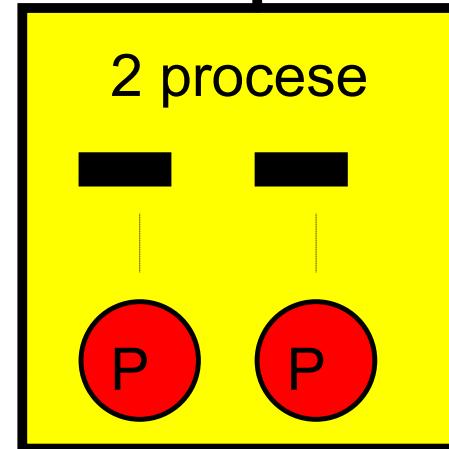
Statia #2



Statia #3

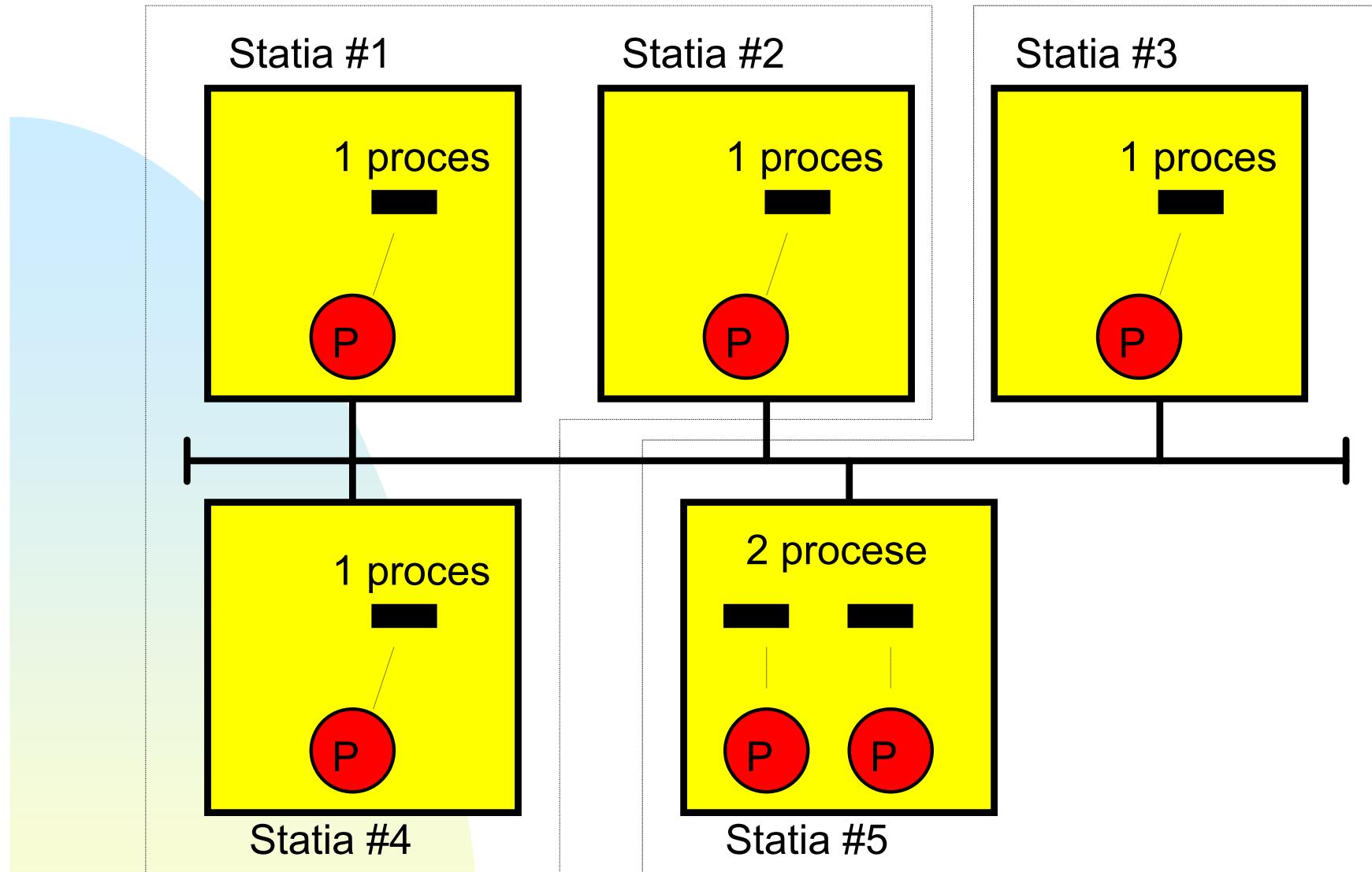


Statia #4



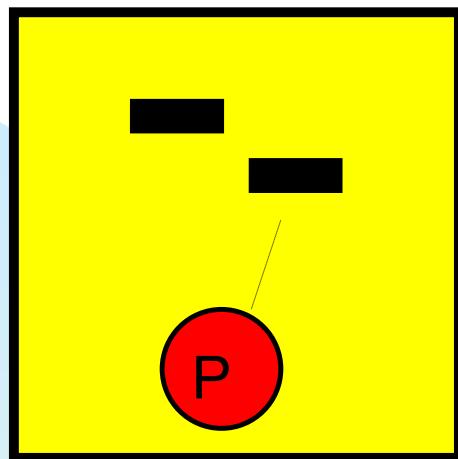
Statia #5

- Cluster folosit în modul dedicat

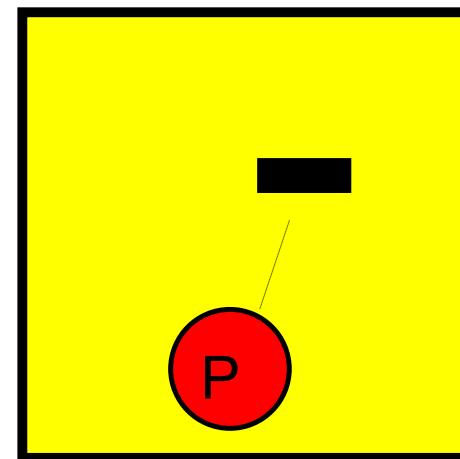


- **Cluster cu partajare spațială. Se observă crearea a două partiții**

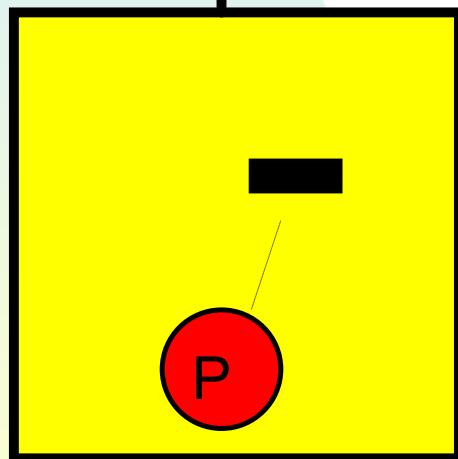
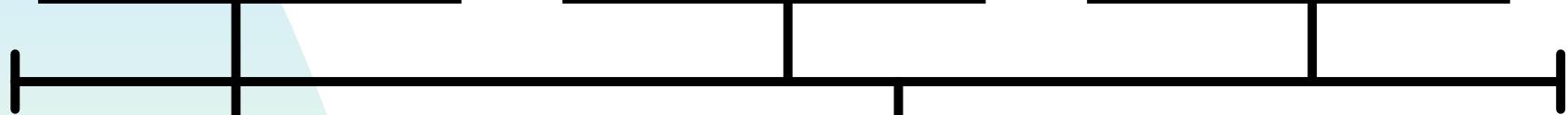
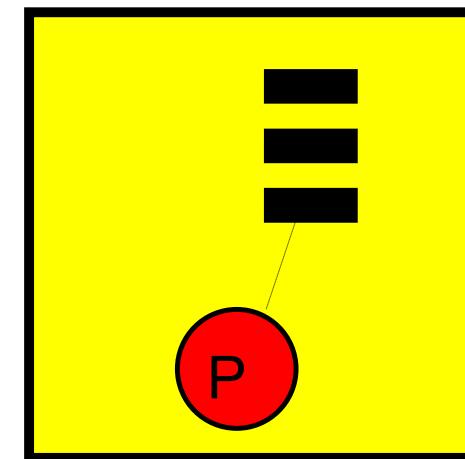
Statia #1



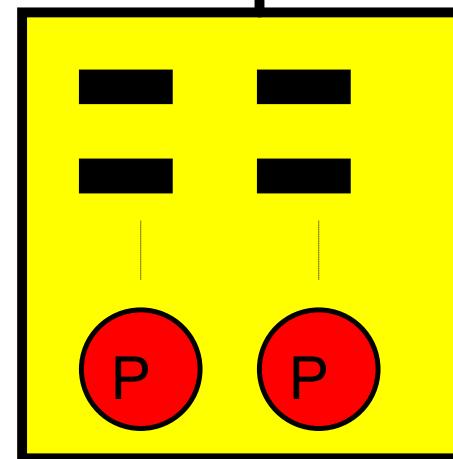
Statia #2



Statia #3



Statia #4



Statia #5

- Cluster folosit în modul cu partajarea timpului

- Prima și ultima dintre strategii sunt mai simple, dar nu garantează un timp de execuție minim.
- Dimpotrivă, prioritatea mai mare a proceselor locale, față de care utilizatorul cluster-ului nu are cum interveni, poate conduce la un timp inacceptabil de mare.
- De aceea, aceste variante pot fi utilizate în perioadele de inactivitate – noaptea, la sfârșit de săptămână, etc.

Tratarea eterogeneității

- Din ce în ce mai mult, rețelele folosite sunt eterogene, chiar dacă folosesc aceeași platformă software: diferă ca tip de procesor, număr de procesoare, memorie instalată, resurse de I/E, etc.
- Tratarea eterogeneității nu este un aspect minor, chiar dacă ne gândim doar la posibilitatea migrării și execuției codului de pe o mașină pe alta.
- Există diferențe mari de performanțe între mașinile existente, chiar atunci când sunt compatibile la nivel de cod.

- selectarea pentru execuție a nodurilor unui sistem distribuit (domeniu de execuție), poate fi făcută pe baza unui parametru (*factorul de interferență*).
- Aceasta ține cont atât de caracteristicile statice de performanță (hard), cum ar fi tipul procesorului, dimensiunea memoriei principale, resurse de I/E, tipul sistemului de operare, cât și de o serie de parametri dinamici.
- Parametrii dinamici se referă la gradul de încărcare al nodului și tipul proceselor în execuție, sau așteptare – interactive, intensive din punct de vedere al calculelor, sau al operațiilor de I/E etc.

- **Acest factor include în anumite forme:**
 - ◆ performanțele nodului destinație, susceptibil de a fi ales pentru execuție, în raport cu cele ale nodului sursă;
 - ◆ caracteristicile dinamice ale nodului destinație și ale canalelor de comunicație;
 - ◆ caracteristicile proceselor existente;
 - ◆ caracteristicile proceselor ce vor fi transferate.

- Decizia de distribuție a aplicației în rețea se ia după calcularea factorilor de interferență, pentru toți candidații (din vecinătatea) și, pe această bază, a funcției de utilitate care controlează, în final, operația de distribuție.
- Există două aspecte importante, influența proceselor nou venite pe un nod asupra celor existente și câștigul de performanță ce se poate obține în mod real.
- Nu este clar cum influențiază această interacțiune timpul total de răspuns pentru o stație folosită în time-sharing.

Clustere Beowulf

- Au apărut inițial ca un proiect inițiat 1993 de Donald Becker și Thomas Sterling
- Clusterele uzuale (clasa I) au următoarele caracteristici
 - ◆ hardware disponibil din diverse surse (preturi mici, depanare etc);
 - ◆ - nu există dependență de un singur producător de hardware;
 - ◆ - drivere pentru Linux;
 - ◆ - standarde uzuale (IDE, SCSI, Ethernet etc).
- Spre deosebire, clusterele de clasa II sunt de regula clustere omogene și sunt proiectate special pentru a obține performanță (hardware scump, tehnologii proprietar etc).

Cluster IBM

- Are abilitatea de a face disponibile la nivelul întregii întreprinderi resursele neutilizate unde și când este nevoie de ele, rezultând un singur sistem, virtual.
- În centrele de date de astăzi, clusterele de servere care rulează aplicații de afaceri nu reușesc adesea să se descurce cu suprasarcini neprevăzute.
- Un server poate să stea nefolosit, în timp ce altul poate să fie exploatat la maximum.

- IBM a dezvoltat primul produs software de tipul "gestionar de trafic" (traffic cop-like), care monitorizează intelligent și automat suprasarcinile de aplicații și direcționează traficul spre un server sau altul, în funcție de încărcarea acestuia la un moment dat.
- Acest software permite unui cluster format din mai multe servere - de la câteva zeci la sute - să opereze ca un singur mediu care se adaptează automat la schimbările brusete, funcționând de o maniera foarte apropiata rețelelor de energie electrică.

WebSphere Performance Advisor:

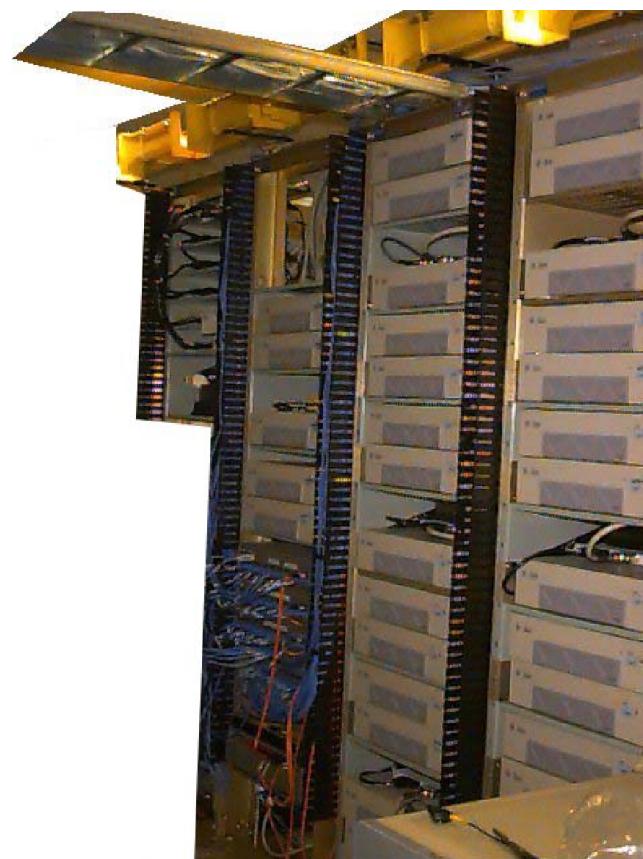
- acesta simplifică munca administratorului colectând date în timp real dintr-un sistem în lucru pentru a analiza schimbările intervenite și pentru a recomanda acțiuni de îmbunătățire a performanțelor aplicației.
- Automatic Backup Clusters:
- Cu WebSphere, clientii pot să-si configureze automat sistemul pentru a instala un cluster de servere de rezerva, pentru cazul în care cluster-ul principal cedează - fără să fie nevoie să se scrie vreun cod.

Proiectul NOW de la Universitatea Berkeley

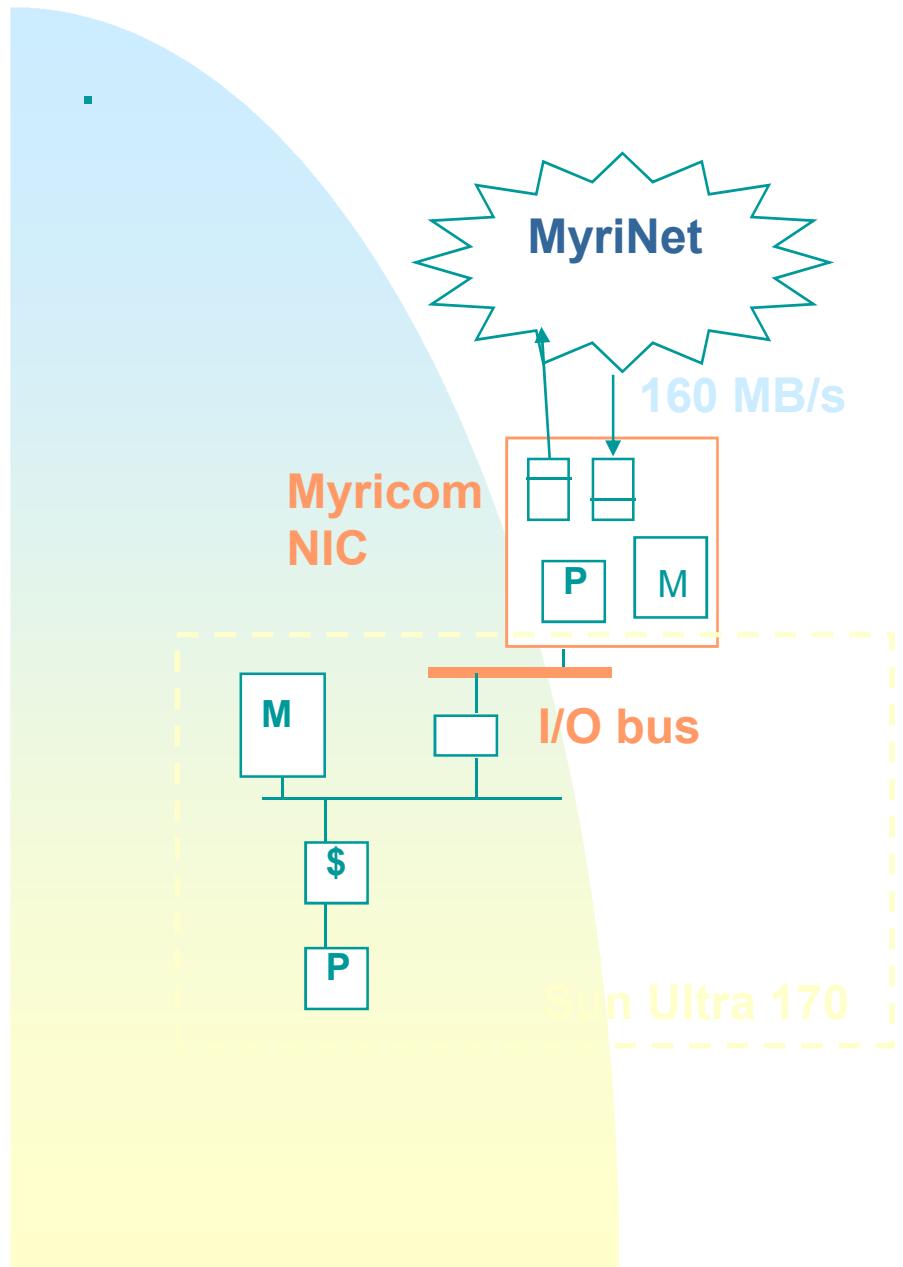
- Unul din proiectele cele mai importante de software pentru gestiunea cluster-ului a fost proiectul NOW – Network of Workstations.
- Acest proiect, încheiat în anul 1998, și-a propus dezvoltarea de tehnici pentru cluster-e de întreprindere, aplicabile și cluster-elor dedicate.
- Obiectivul propus era ca un cluster să răspundă atât job-urilor interactive cu viteza unei stații de lucru, cât și să execute job-uri prea mari pentru o singură stație.
- Pentru atingerea acestui obiectiv, s-a proiectat un strat software de tip *cluster middleware* denumit GLUnix, care se aşează deasupra sistemelor de operare existente.

- În faza finală, GLUnix a realizat numai o parte din mecanismele propuse inițial.
- Permite lansarea în execuție de job-uri la orice nod al sistemului.
- Poate activa procese *worker* componente pe orice alt nod. Planificarea lor este de tip gang.
- Păstrează informații despre încărcarea fiecărui nod, trimițând procese nodurilor mai puțin încărcate.
- Un semnal care este trimis unui proces este multiplexat tuturor proceselor *worker*.
- Ieșirile către *stdout* și *stderr* sunt puse în pipe la nodul startup, iar caracterele trimise la *stdin* sunt multiplexate către toate procesele *worker*.

Berkeley NOW: 100 Sun UltraSparc lucrează împreună



Componente de bază



Cluster pentru crearea unui spațiu mare pe discuri



Unitatea de bază:
2 PC-uri cu patru
SCSI a 8 discuri
fiecare

Cluster cu SMP-uri (CLUMPS)



4 Sun E5000s

8 procesoare

4 Myricom NICs
fiecare

Multiprocesor, Multi-NIC, Multi-Protocol

NPACI => Sun 450s

Millennium PC CLUMPS



Cluster ieftin, ușor de administrat

Multiplicat în multe departamente

Prototip pentru clustere din PC-uri, foarte mari.

Windows server

- **Comunicare și securitate**
- Protocolul de comunicație care sta la baza sistemului de operare este TCP/IP
- Alte protocole, care sunt câteodată necesare pentru comunicarea cu alte rețele sau aparate: IPX, AppleTalk, NetBEUI etc.
- De asemenea PPP în versiunea extinsă PPTP (Point To Point Tunneling Protocol) folosit pentru crearea de rețele private virtuale (VPN) pe linii dial-up.
- Suportul pentru VPN-uri este extins prin folosirea și a protocoalelor IPSec (IP Security) și L2TP (Layer 2 Tunneling Protocol).

- Legat de comunicare, apare în sfârșit „controlul de calitate” al serviciului (QoS - Quality of service), prin care administratorul rețelei poate aloca, pentru portiuni ale rețelei, priorități diferite pentru diferite servicii.
- De exemplu, traficul de e-mail-uri poate fi prioritar față de browsing, dar să cedeze întâietatea pentru VoIP (telefonie peste IP), care necesită o rata de transfer constantă.

- Autentificarea se face folosind serviciul Kerberos, dezvoltat de MIT.
Prin Kerberos, fiecare utilizator din rețea primește o cheie unică (numită „tichet”), care este inclus în fiecare mesaj al său, pentru a-l identifica.
- Modelul de autorizare se bazează pe asocierea unei liste de control a accesului (ACL - Access Control List) pentru fiecare resursă și pentru fiecare utilizator sau grup de utilizatori.

- Sistemul de operare compară drepturile utilizatorului de pe lista cu cele din ACL-ul fișierului și dacă există o corespondență, utilizatorul primește acces.

■ **Serviciul de directoare**

- Active Directory, serviciul de directoare din Windows server, este o componentă esențială a sistemului de operare.

- Oferă transparentă pentru utilizator din punct de vedere al structurii fizice a rețelei precum și protocoalele ce o ghidează, pentru ca acesta să poată accesa o resursă fără a-i cunoaște locația fizică (desigur, dacă are dreptul...).
- Serviciul de directoare este și baza sistemului de management și de securitate al rețelei.

- Active Directory este o structură arborescentă, ierarhică, a resurselor, asemănătoare sistemului de fișiere bazat pe directoare și fișiere, dar în care fiecare resursă reprezintă un obiect.
- **Servicii de fișiere și imprimare**
- Prin cotele (quota) de spațiu pe disc, administratorul aloca pentru utilizatori și grupuri un spațiu maxim pe volumele NTFS din rețea.

- Managementul ierarhic al resurselor de stocare (HSM) permite mutarea automată a datelor între mediile de stocare scumpe (și cu timp de acces mic) și cele ieftine (cu timp de acces mare, dar și de mare capacitate).
 - ◆ HSM salvează datele pe unitățile de backup, iar la nevoie le readuce pe harddisk-urile serverului.
 - ◆ HSM se bazează pe Remote Storage (de la Seagate), un serviciu care monitorizează spațiul liber de pe volumele serverului.

- Tot de la Seagate vine și utilitarul de backup din Windows server.
- Aceasta este integrat în serviciile centrale ale sistemului de operare:
 - ◆ Active Directory,
 - ◆ serviciul de replicare
 - ◆ servicii certificare.
- și tipărirea în rețea plug&play pentru imprimante, organizarea logica și nu fizica a imprimantelor în Active Directory, tipărirea prin IPP (Internet Print Protocol).

Windows server Advanced Server

- Windows server Advanced Server asigură scalabilitate integrată a sistemului prin multiprocesare simetrică îmbunătățită (hardware scaling sau scaling up).
- **Scalabilitate pentru multiprocesare simetrică îmbunătățită (Symmetric Multiprocessing Scalability- SMP)**

■ Serviciul Cluster

- Windows server Advanced Server furnizează servicii de sistem pentru clustering de servere ca și componentă standard a produsului. Un cluster de servere este un set de servere independente care sunt gestionate împreună.
- Serviciul de clustering minimizează timpii morți și reduce costurile IT prin furnizarea unei arhitecturi care menține sistemele rulând în cazul în care unul dintre servere se defectează.

- Există mai multe moduri de repartizare a sarcinii pe un cluster de două noduri, dar două dintre acestea sunt mai comune:
 - Plasarea sistemelor de producție tranzacționale pe un nod iar dezvoltarea și testarea pe celălalt nod
 - Plasarea logicii comerciale pe un nod și a serviciilor de baze de date pe al doilea nod (într-o aplicație pe 3 nivele)

- Tehnologia serviciului Cluster încorporată în Windows server Advanced Server permite conectarea a două servere într-un cluster pentru disponibilitate mai mare și gestionare mai ușoară a resurselor de server.
- Cele două servere nu trebuie să aibă neapărat aceeași mărime sau aceeași configurație. Caracteristicile serviciului Key Cluster din Advanced Server includ:
- **Suport pentru actualizare din mers.**

- **Suport pentru Active Directory și integrare MMC.**
- Serviciul cluster pentru Windows server folosește serviciul Active Directory pentru publicarea informațiilor despre clustere.
- Integrarea cu Microsoft Management Console (MMC) face configurarea ușoară și permite administratorilor monitorizarea vizuală a stării tuturor resurselor din cluster.

- **Recuperare în cazul blocării rețelei.** Serviciul cluster pentru Windows server implementează un algoritm sofisticat pentru detectarea și izolarea defectiunilor survenite în rețea și pentru îmbunătățirea acțiunilor de recuperare în cazul de blocare a rețelei.
- **Monitorizarea stării.** Serviciul cluster monitorizează starea aplicațiilor standard și serverelor și poate recupera automat date critice și aplicații din multe tipuri obișnuite de defectiuni - de obicei în mai puțin de un minut.

- **Unitatea de monitorizare și tratare erori** este un serviciu sau o aplicație.
- **Suport Plug and Play pentru rețele și discuri hard.**
- Folosind tehnologia Plug and Play încorporată în Windows server, serviciul Cluster detectează adăugarea și înlăturarea adaptoarelor de rețea, stivelor de rețea TCP/IP și discurilor fizice partajate.

- **Suport WINS, DFS și DHCP.** Serviciul Cluster suportă acum protocolele Windows Internet Name Service (WINS) și Dynamic Host Configuration Protocol (DHCP), precum și Distributed File Services ca resurse cluster-aware care suportă trecerile peste erori și recuperarea automată.
- O resursă de fișiere poate servi acum ca rădăcină a sistemului de fișiere distribuit (distributed file system - DFS) sau își poate diviza folderul în subdirectoare pentru gestionarea eficientă a unui mare număr de resurse de fișiere corelate.

- **Suport COM pentru cluster API.**
- Serviciul cluster al Windows server Advanced Server include un API standard pe mai multe platforme pentru dezvoltarea și suportul aplicațiilor cluster-aware.
- Acest API poate fi folosit pentru a crea aplicații scalabile cluster-aware care pot echilibra automat sarcinile pe serverele multiple din cluster și poate fi accesat de Windows Scripting Host pentru a controla comportamentul clusterului și pentru a automatiza mai multe sarcini de administrare.

Echilibrarea sarcinii în rețea (Network Load Balancing - NLB)

- Echilibrarea sarcinii în rețea permite organizațiilor să formeze clustere cu servere rulând Windows server Advanced Server pentru a distribui în mod egal traficul de intrare în rețea în timp ce monitorizează de asemenea starea serverului și a interfeței de rețea (NIC).
- Avantajul dublu al scalabilității simple, incrementale, combinate cu disponibilitatea ridicată fac NLB ideal pentru utilizarea cu aplicații critice comerciale și de e-commerce, găzduire și aplicații Terminal Services.

- Avantajele lui Network Load Balancing includ:
- **Performanță scalabilă**
- Reduce timpii morți planificați prin suportul pentru actualizare completă în mers.

■ Disponibilitate ridicată

- Detectează și recuperează automat date de la un sistem blocat sau offline.
- Redistribuie automat sarcina în rețea când se schimbă setările clusterului.
- Recuperează și redistribuie sarcina de lucru
- Manevrează împărțirea inadecvată în subretele și realăturarea rețelei clusterului.

■ Capacitate de control

- Specifică echilibrarea sarcinii pentru un singur port sau grup de porturi IP folosind reguli de gestionare directă a portului care se potrivesc cu sarcina de lucru a fiecărui server în parte.
- Suportă sesiuni cliente și SSL.

■ Ușurință utilizării

- Integrare cu infrastructura de rețea a Windows server Advanced Server.
- Nu necesită hardware specializat.
- Permite clientilor accesul la cluster folosind un singur nume logic Internet și adresă IP, în timp ce reține numele individuale pentru fiecare calculator.

- **Sortare de înaltă performanță**
- Windows server Advanced Server optimizează sortarea comercială a seturilor mari de date.
- Această sortare va fi utilizată în mod tipic pentru a pregăti datele pentru încărcarea în depozitele de date, pentru aplicații de tip piată de date și pentru a pregăti operațiile de tipărire cu suport de sortare și batch.
- **Suport pentru memorie îmbunătățit**

- **Arhitectura *Cluster service***
- *Cluster service* controlează toate operațiile ce se execută într-un sistem cluster. El rulează și utilizează driverele de rețea, cele pentru discuri, procesele de gestionare a resurselor și este implementat ca un serviciu care constă în mai multe componente puternic interconectate:
 - ◆ **checkpoint Manager** — salvează application registry keys ai aplicației într-un director al clusterului stocat pe o quorum resource

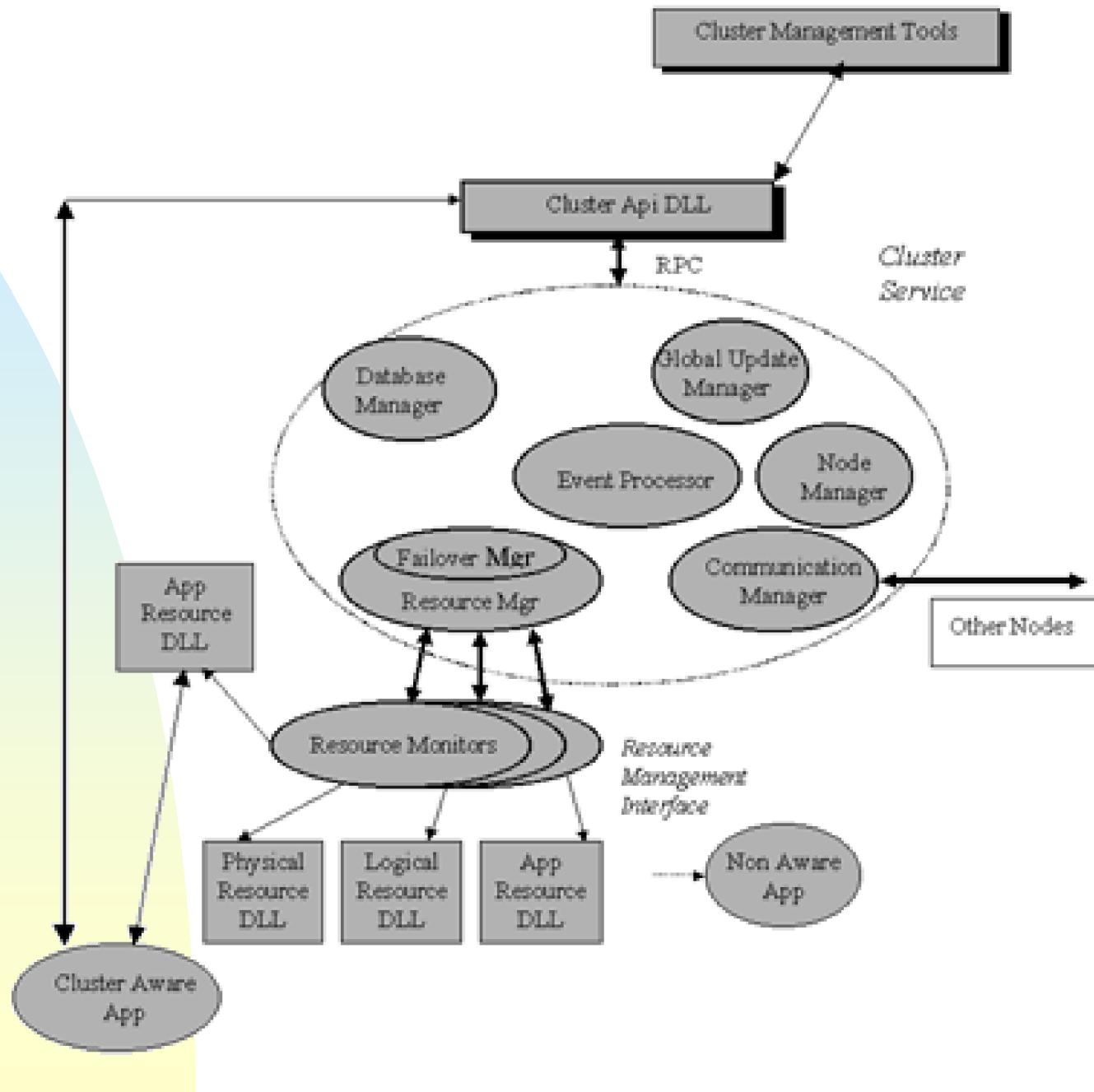
- ◆ **Communications Manager** — asigură comunicarea între nodurile clusterului.
- ◆ **Configuration Database Manager** — menține informațiile cu privire la configurația clusterului.
- ◆ **Event Processor** — recepționează mesajele de la resursele clusterului cum ar fi modificări ale stării acestora sau cereri ale aplicațiilor de a deschide, închide, accesa obiecte ale clusterului.
- ◆ **Event Log Manager** — replichează event log-urile de la un nod la toate celelalte noduri ale clusterului.

- ◆ **Failover Manager** — realizează managementul resurselor și inițiază acțiunile corespunzătoare ca startup, restart și failover.
- ◆ **Global Update Manager** — asigură un serviciu global de actualizare folosit de componentele clusterului.
- ◆ **Log Manager** — scrie modificările în recovery logs stocate pe quorum resource.
- ◆ **Membership Manager** — menține componența clusterului și monitorizează nodurile clusterului.
- ◆ **Node Manager** — atribuie proprietatea resurselor de group nodurilor, luând în considerație liste de preferință și disponibilitatea nodurilor.

- ◆ **Object Manager** — realizează managementul tuturor obiectelor *Cluster service*.
- ◆ **Resource Monitors** — monitorizează fiecare resursă a clusterului folosind codurile de eroare ale resurselor DLL. Resource Monitors rulează într-un proces separat și comunică cu *Cluster service* prin apeluri de procedură la distanță (RPCs) pentru a proteja *Cluster service* de la erori individuale ale resurselor clusterului.

Components

or



Formarea unui cluster

- Pentru a forma un cluster, un nod trebuie sa poata capata posesia exclusiva a *quorum resource*.
- Quorum resource menține integritatea datelor și unitatea clusterului și joacă un rol critic în rularea clusterului.
- El trebuie să fie prezent pentru operațiile nodurilor – cum ar fi formarea și adăugarea unui nod.
- Stările *Cluster service* sunt:
 - ◆ **Offline.** Nodul nu este un membru activ al clusterului. Nodul și *Cluster service* pot sau nu pot fi pornite.

- ◆ **Online.** Nodul este membru activ al clusterului. El onorează actualizările bazei de date, contribuie la algoritmul de votare, menține mecanismul de heartbeats și poate fi posesorul unui grup de resurse.
- ◆ **Paused.** Node este membru activ al clusterului. El onoroază actualizările bazei de date, contribuie la algoritmul de votare, menține mecanismul de heartbeats dar nu poate fi posesorul sau nu poate să ruleze resurse decat cele proprii

Adăugarea unui nod la un cluster

- Pentru adaugare trebuie ca serverul să ruleze *Cluster service* și să detecteze cu succes un alt nod al clusterului.
- Ulterior acest nod primește o copie a bazei de date conținând configurația clusterului.

Părăsirea unui cluster

- Un nod poate părăsi un cluster că acesta sau *Cluster service* este oprit. În orice caz un nod poate fi deasemeni eliminat dintr-un cluster când acesta eșuează în rularea unor operații cum ar fi actualizarea bazei de date conținând configurația nodului.
- Când un nod părăsește un cluster el trimite un mesaj de tipul **ClusterExit** la ceilalți membri ai clusterului. Nodul nu așteptă nici un răspuns de confirmare a primirii acestui mesaj de la celelalte noduri.

Cursul

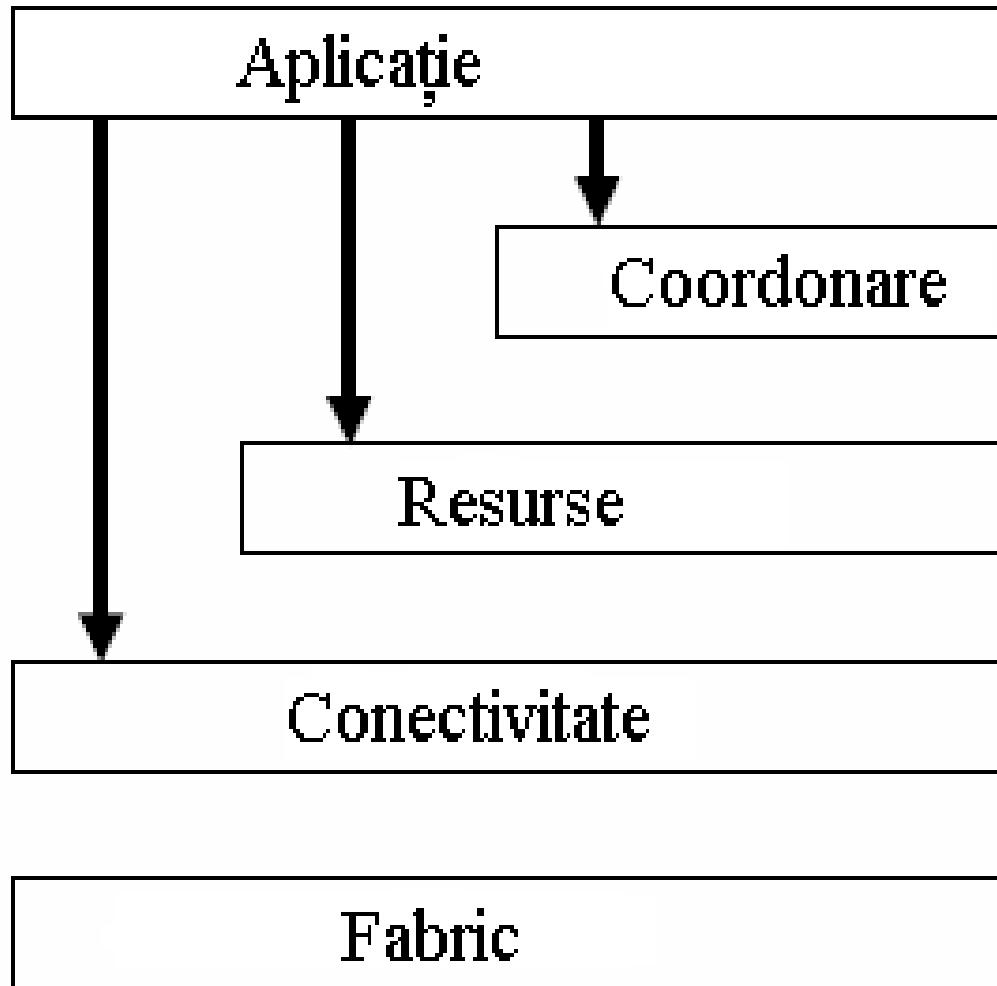
Grid

- Calculul Grid este o formă de calcul distribuit care implică calcule distribuite și coordonate, aplicații, informații, stocări sau resurse de rețea pentru organizațiile dinamice și disperse geografic.
- Bazându-se pe calculul paralel, mai multe organizații au organizat studii experimentale pentru a unifica resursele distribuite geografic, astfel încât să se comporte ca un singur calculator foarte puternic

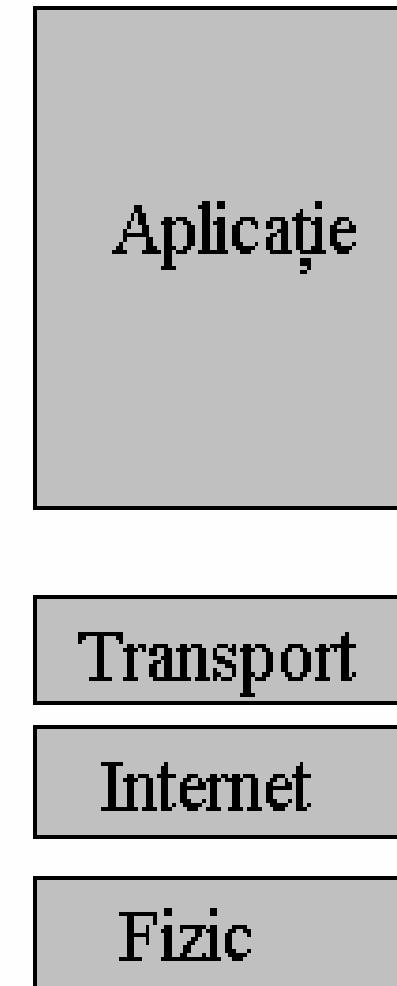
Descrierea Arhitecturii Grid

- Nivelele modelului Grid și legăturile cu protocolul Internetului
- Acest model este unul conceptual și este organizat pe cinci nivele ierarhice.
- Acest fapt permite introducerea de standarde independente pentru fiecare nivel în parte.

Descrierea nivelerelor arhitecturii Grid



Arhitectura Protocolului Grid



Arhitectura Protocolului Internet

Nivelul Fabric (Fabric)

- Acest nivel oferă resurse individuale la care accesul este mediat de către protocoalele Grid.
- Astfel de resurse
 - ◆ sisteme de calcul independente,
 - ◆ clustere de calcul paralel,
 - ◆ sisteme de stocare a datelor,
 - ◆ cataloage, etc
-

- Apare totuși o interdependentă între funcțiile implementate la nivel Fabric și operațiile distribuite care sunt suportate.
- Nivelul Fabric este capabil să funcționeze pe mai multe operații complexe distribuite.

Tipuri servicii în fabrică

- Se impun cel puțin două tipuri de servicii pentru nivelul Fabric:
 - ◆ serviciul de raportare a resurselor disponibile, folosit de nivelul superior pentru a descoperi și monitoriza structura, starea și abilitățile specifice;
 - ◆ servicii de management a resurselor, mecanism care permite garantarea unei calități a serviciului asumat;

Tipuri de resurse:

Resursele de calcul:

- ◆ Interfața de management a resurselor de calcul pentru lansarea în execuție a programelor, pentru monitorizarea și controlul execuției cât și a proceselor rezultante.

Resurse de stocare:

- ◆ Spatiul de stocare a fișierelor și a rezultatelor unui proces trebuie să fie guvernate de un sistem de management a acestora la nivel global

Resurse de rețea:

- Inter-operabilitatea în cadrul VO; ca atare funcționarea sistemului Grid în ansamblu
- Aceasta funcționează pe baza setului de protocoale TCP/IP și oferă servicii de tip „best effort” (efortul cel mai bun).
- Pentru o structură Grid este nevoie să se asigure, la nivelul rețelei, funcționarea unui mecanism care să interacționeze cu nivelele superioare ale arhitecturii Grid

Alte resurse:

- Pentru funcționarea unui sistem Grid, o serie de alte resurse sunt necesare și, mecanisme de control și raportare a disponibilității acestora.
- De exemplu:
 - sisteme de stocare a surselor programelor
 - de control a versiunii disponibile
 - sisteme de evidență a modificărilor acestora;
 - baze de date specializate pentru stocarea diverselor tipuri de date, etc.

Nivelul Conectivitate (Connectivity)

- Nivelul Conectivitate îndeplinește două funcții importante: transportul informației și sistemul de securitate și autentificare.
- Transportul informației se realizează între resursele Nivelului Fabric.
- Sistemul de autentificare este bazat pe transportul informației pentru a prevedea mecanisme de criptografie sigure folosite la verificarea identității utilizatorilor și a resurselor.

Protocole de baza

- Protocole ale nivelului rețea:
 - ◆ Internet Protocol (IP) și Internet Control Message Protocol (ICMP)
 - ◆ protocole pentru asigurarea calității serviciilor (ex. Resource Reservation Protocol - RSVP).
- Protocole ale nivelului transport:
 - ◆ Transport Control Protocol (TCP)
 - ◆ User Datagram Protocol (UDP).
- Protocole ale nivelului aplicație:
 - ◆ sistem de nume (DNS),
 - ◆ sistem de acces la fișiere (NFS, AFS, etc).

Soluțiile pentru autentificare trebuie să respecte o serie de condiții

- **Autentificare o singură dată (Single Sign On):**
 - ◆ este un sistem care asigură accesul la resursele definite ale nivelului Fabric pentru care un anume utilizator este autorizat în urma unui singur proces de autentificare la nivelul Grid.
- **Delegarea (Delegation):**
 - ◆ un program al unui utilizator trebuie să poată avea acces la toate resursele la care ar avea acces acel utilizator.

- **Integrarea cu soluții locale de securitate** fiecare participant în structura Grid poate utiliza diverse soluții locale de securitate.
- Soluțiile pentru asigurarea securității Grid trebuie să interacționeze cu sistemele locale de securitate.



■ Legături de încredere între utilizatorilor:

- ◆ arhitectura Grid trebuie să asigure accesul unui utilizator la resursele alocate în diverse secțiuni ale infrastructurii Grid fără a implica intervenția administratorilor fiecărei secțiuni în parte.
- ◆ Este nevoie de un mecanism global care să interacționeze independent și automat cu sistemul de securitate la nivel local

Nivelul Resurse (Resource)

- Numărul de protocoale disponibile la nivelul Resurse trebuie să fie menținut la minimul necesar pentru a asigura o interfață simplă nivelului Aplicație și nivelului Coordonare.
- Nivelul Resurse este bazat pe funcțiile nivelului Conectivitate pentru definirea protocoalelor, precum API și SDK, pentru negocierea, inițierea, controlul și taxarea operațiilor ce implică utilizarea resurselor locale ale nivelului Fabric.
- Nivelul Resurse este responsabil cu gestiunea individuală a resurselor și nu cu distribuirea globală a acestora, funcție specifică nivelului Coordonare.

- Se disting două clase principale de protocoale ale nivelului Resurse:
 - ◆ **Protocoale de interogare:** folosite pentru centralizarea informațiilor privind structura și starea resurselor disponibile (tipul și configurația resurselor locale, gradul de utilizare, disponibilitatea, costul, etc).
 - ◆ **Protocoale de management:** acestea permit negocierea procedurilor de acces la resurse, specificarea cerințelor privind calitatea serviciilor și a operațiilor necesare pentru crearea proceselor și accesul la date.

■ Nivelul Coordonare (Collective)

- ◆ Funcția principală a nivelului Coordonare este de a coordona utilizarea resurselor multiple.
- ◆ Spre deosebire de nivelul Resurse care operează cu resurse individuale

Servicii ale nivelului Coordonare

- **Servicii de informare:**

- ◆ acordă dreptul participanților unei VO de a determina existența și/sau proprietățile resurselor VO.
- ◆ Un serviciu de informare dă dreptul utilizatorilor de a se informa despre numele și/sau atributele resurselor.

- **Co-alocarea, listarea și împărțirea serviciilor:**

- ◆ Acordă dreptul participanților unei VO de a cere alocarea uneia sau mai multor resurse pentru un scop anume precum și lista sarcinilor resurselor alocate

- **Monitorizarea și diagnosticarea serviciilor:**
 - ◆ suportă căutarea resurselor avariate ale unei VO, atacurile adversarilor, supraîncărcarea resurselor, etc.
- **Serviciul de răspuns al informațiilor:**
 - ◆ suportă managementul memorării resurselor pentru a maximiza performanțele accesului la informație, cu respectarea metricilor

- **Sistemele Grid de autorizare a programelor:**
 - ◆ autorizează folosirea exemplelor de programe cele mai frecvente în mediul Grid, folosind diverse servicii Grid, adresarea descoperirii resurselor, securitatea, alocarea resurselor și altor legături.
- **Sistemele de administrare a încărcărilor și cadrelor de lucru colaborative:**
 - ◆ prevede descrierea, folosirea și managementul pentru pași multipli, sincronizarea componentelor multiple ale fluxurilor tehnologice.²¹

- **Serviciul de descoperire a pachetului de programe:**
 - ◆ descoperirea și selectarea celei mai bune implementări software și executarea platformelor bazate pe parametrii problemelor care au fost rezolvate.
- **Serverul de autorizații al comunității:**
 - ◆ impune comunității politica de guvernare a accesului la resurse, generând capacitatea membrilor comunității de a accesa resursele comunității.

- **Serviciul de gestionare a conturilor și plășilor:**
 - ◆ strâng informații despre resursele folosite în scopul justificării, plășirii și/sau limitării folosirii resurselor de către membrii comunitășii.
- **Serviciul de colaborare:**
 - ◆ suportă coordonarea schimbului de informații la nu mai mult de un potențial larg al comunitășii de utilizatori, fie ei sincroni sau asincroni.
 - ◆ La acest nivel se pot dezvolta serviciile specifice a unei VO sau a unui domeniu de activitate.

■ Nivelul Aplicație (Application)

- ◆ Nivelul Aplicație include aplicațiile utilizatorilor și aplicații specifice care operează în cadrul VO.
- ◆ Aplicațiile utilizator apelează serviciile nivelelor inferioare, motiv pentru care acestea trebuie standardizate și documentate corespunzător.
- ◆ Ar fi de preferat ca nivelul Aplicație să interacționeze numai cu nivelul Coordonare, dar o astfel de abordare a modelului Grid nu este disponibilă în acest moment.

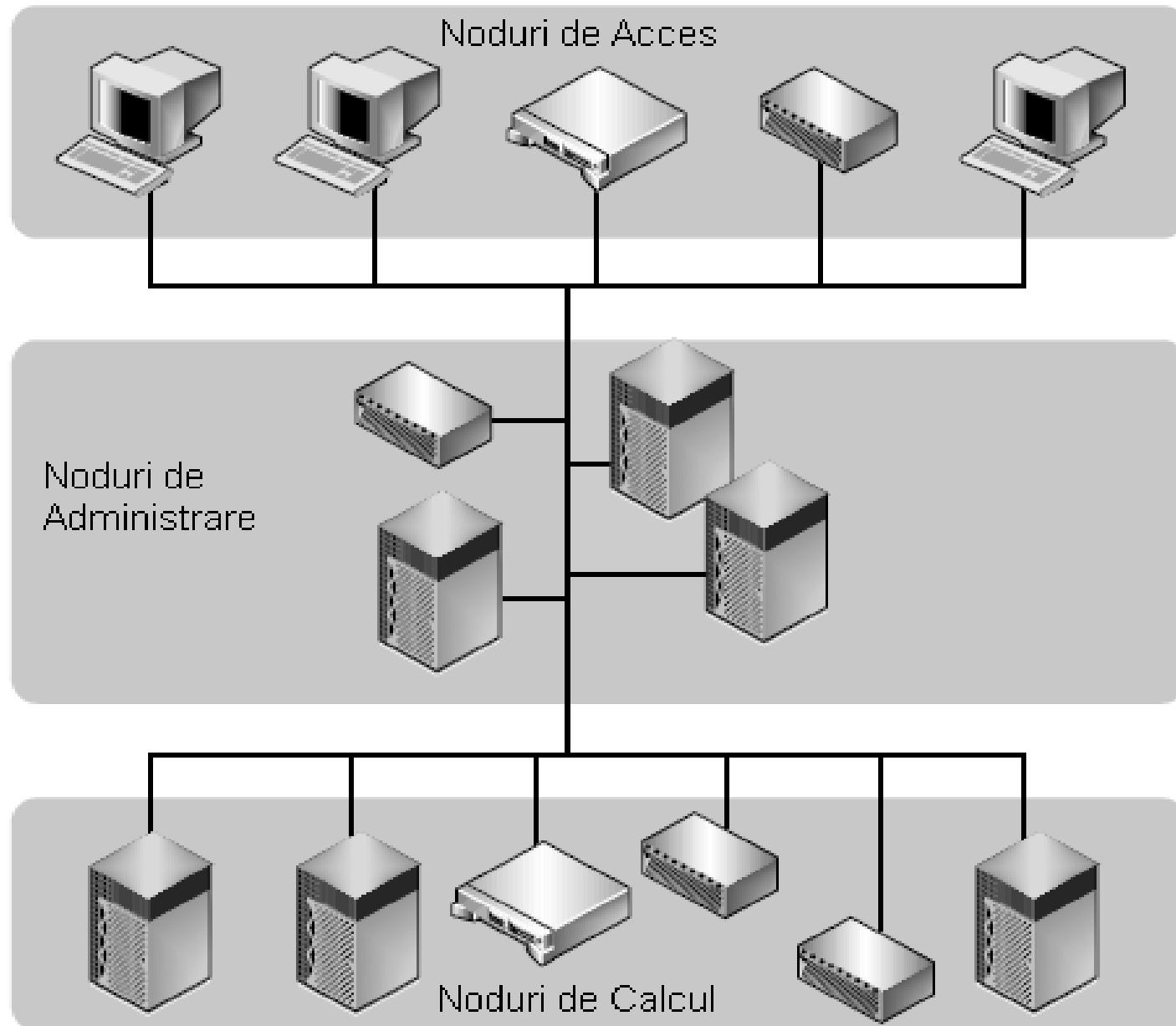
Clasificări ale Tehnologiilor Grid

- Calculul Grid prevede un mediu în care resursele rețelei sunt vizualizate pentru a activa un model de calcul util.
- Calculul Grid revede servicii foarte accesibile cu un înalt grad de transparență pentru utilizatori.
- Calculul Grid poate fi divizat în următoarele trei nivele de dezvoltare:
 - ◆ **Cluster Grid**
 - ◆ **Campus Grid**
 - ◆ **Global Grid**

Cluster Grid

- Un Cluster Grid este cea mai simplă formă de Grid, care prevede un serviciu de calcul pentru nivelul grupului sau departamentului.
- Beneficiul cheie al arhitecturii de Cluster Grid este de a maximiza folosirea resurselor de calcul, și creșterea rezultatelor problemelor utilizatorilor.
- Un Cluster Grid este un set superior de alte resurse tehnice de calcul

- Arhitectura unui Cluster Grid este divizată în trei nivele logice neierarhice:
 - ◆ Acces
 - ◆ Administrare
 - ◆ Calcul
- Fiecare nivel este definit de către furnizorul de servicii.
 - ◆ Nivelul de acces
 - ◆ Nivelul de administrare
 - ◆ Nivelul de calcul
- Fiecare nivel poate fi considerat independent de celelalte



■ **Nivelul acces**

- ◆ furnizează accesul și autentificarea serviciilor pentru utilizatorii clusterului Grid.
- ◆ Orice metodă de acces trebuie să fie capabilă să se integreze cu schema comună de autentificare cum ar fi NIS sau LDAP.
- ◆ Acest nivel poate fi utilizat pentru integrarea cu Global Grid.

■ **Nivelul de administrare**

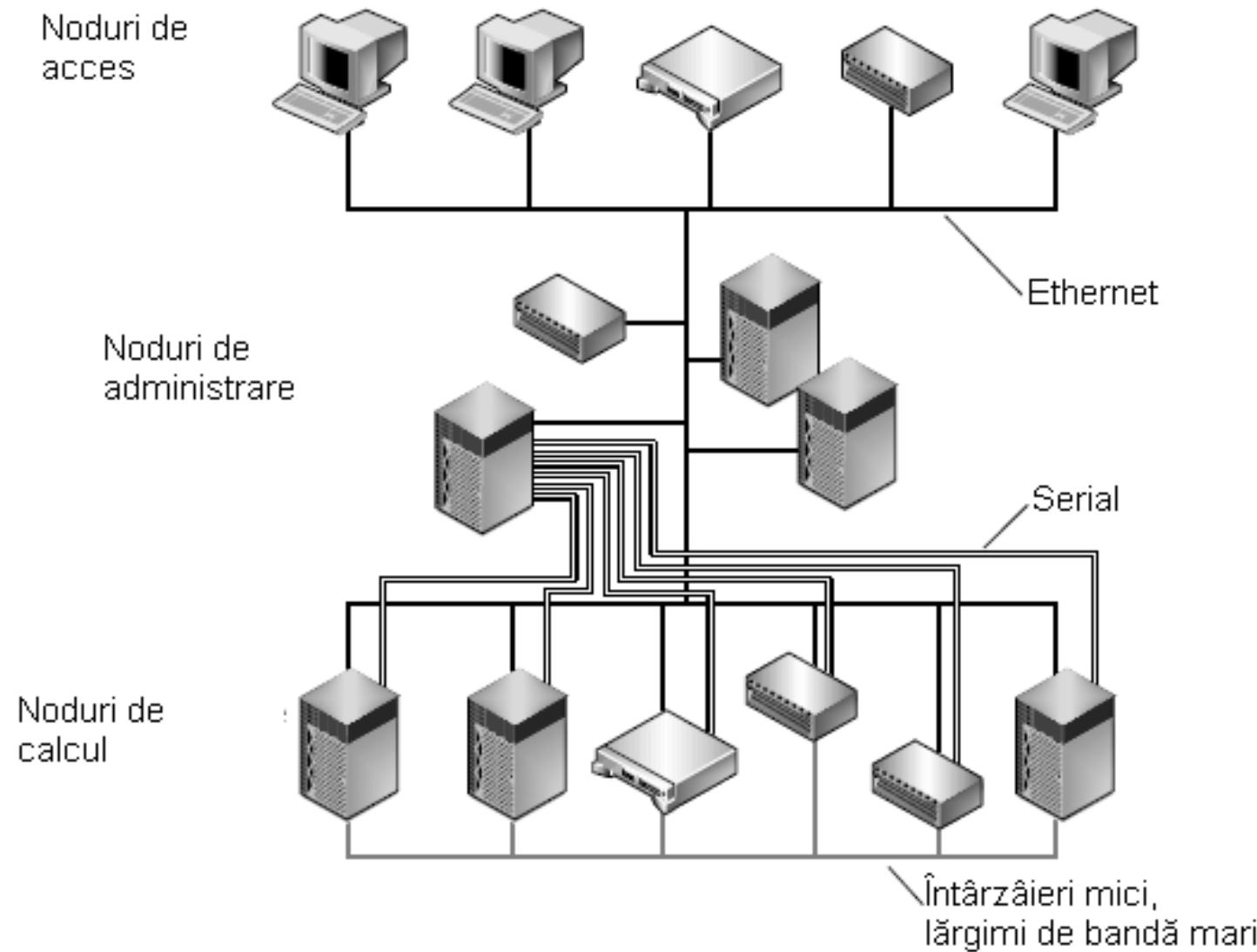
- ◆ este responsabil cu furnizarea serviciilor majore pentru cluster Grid:

- Pe lângă aceste servicii, mai există și altele secundare:
 - ◆ *Serviciul fișierelor* –
 - ◆ *Serviciul de validare a licenței* –
 - ◆ *Administrarea susținerilor*
 - ◆ *Serviciul de instalare* –

Nivelul de calcul

- Asigura puterea de calcul pentru un Cluster Grid.
- Problemele transmise de nivelele superioare din arhitectură sunt planificate să ruleze pe unul sau mai multe noduri în nivelul de calcul.
- Nivelul de calcul comunică cu nivelul de administrare prin primirea sarcinilor și prin raportarea stării de completare a problemelor și detaliile de continuare.

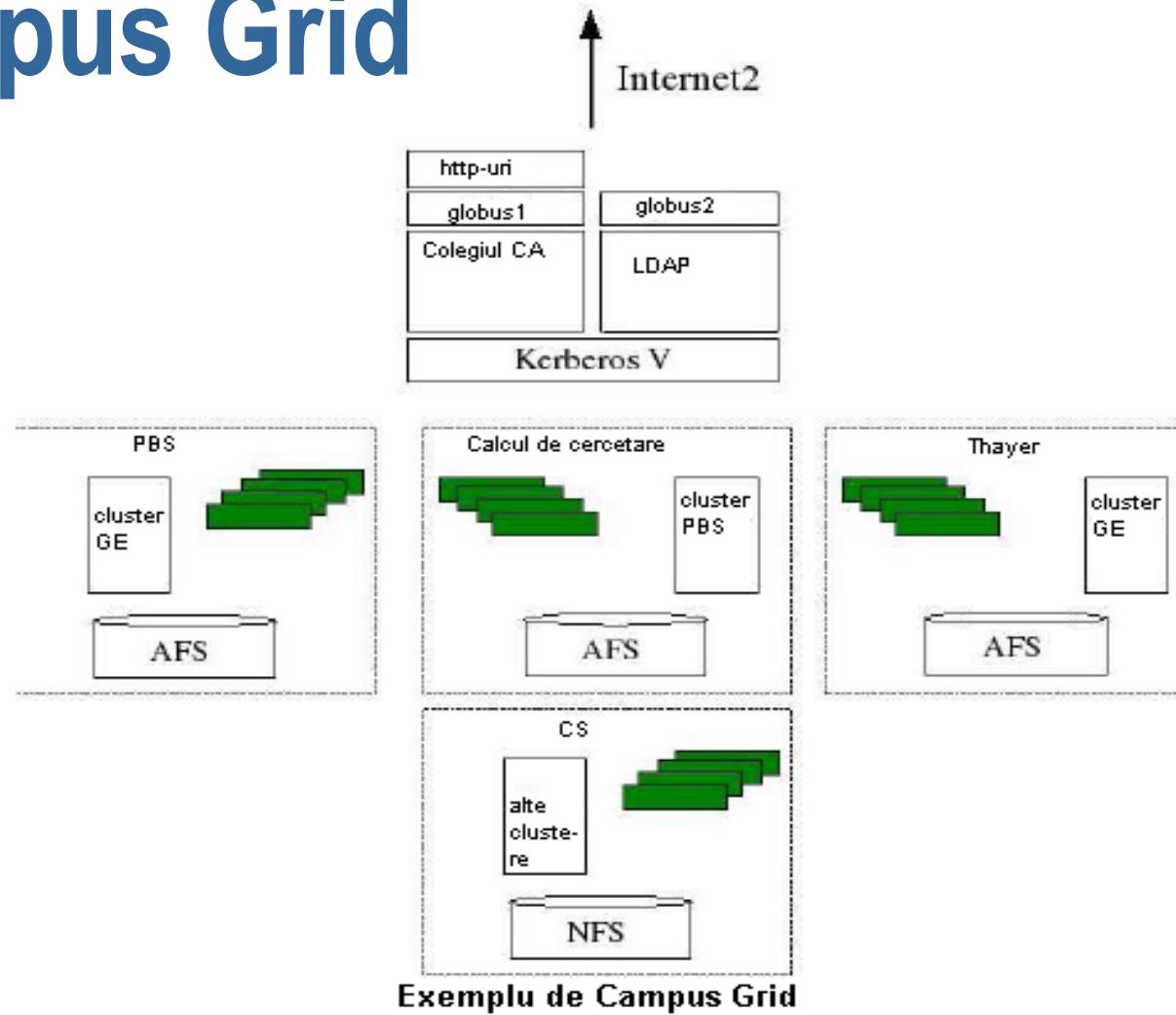
■ Nivelul de calcul poate fi heterogen în funcție



Campus Grid

- Această clasă de Grid activează proiecte multiple sau departamentale printr-o organizație de distribuire a resurselor.
- Organizațiile pot folosi Campus Grid pentru a rezolva o largă varietate de probleme.
- Sunt cerute explicit politici de administrare și negociere pentru a putea suporta mediul dinamic.

Campus Grid



- **Campus Grid** poate susține un număr mare de calculatoare și clustere, poate integra colecții de informații științifice și echipamente de cercetare.

Global Grid

- Componentele principale ale Global Grid sunt:
 - ◆ Interfata/prezentarea – permite accesul în cadrul Grid al utilizatorilor obișnuiți. Câteva “suprafete” de încredere sunt bazate pe crearea unor portaluri web.
 - ◆ Agent – o planificare automată a problemelor, bazată pe politicile proprii utilizatorului. Astfel de politici pot descrie prioritățile utilizatorilor în termeni de cereri ale problemelor, timpul cerut, și altele.
- Securitate,
- Garantarea și numărarea resurselor

- O caracteristică importantă a Global Grid pentru utilitatea comunicărilor este transparenta transferului de informații.
- Prințipiile arhitecturale pentru Global Grid:
 - ◆ Folosirea unei infrastructuri de rețea comune;
 - ◆ Transportul oricărui tip de trafic;
 - ◆ Integrarea cu ușurință a unor transporturi de informații multiple(cum ar fi comunicațiile de voce, radio, video, etc.);
 - ◆ Adaptarea la schimbări;
 - ◆ Prevede siguranța serviciilor.

- Mecanisme de realizare a principiilor arhitecturale:
 - ◆ Transportul informațiilor printr-o infrastructură bazată pe IP;
 - ◆ Aplicarea metodologiei arhitecturii pe nivale;
 - ◆ Comunicările comerciale și tehnologiile rețelelor extensibile.

Problema programarii

- Ce ar trebui pentru apuesta dezvolta aplicatii care sa fie robuste, sigure cu cliclu mare debiata si performante bune peste un grid eterogen
 - ◆ Abstractiuni si modele
 - ◆ Instruente dedicate
 - ◆ Maniera de “Code/tool sharing” –

Examples of Grid Programming Technologies

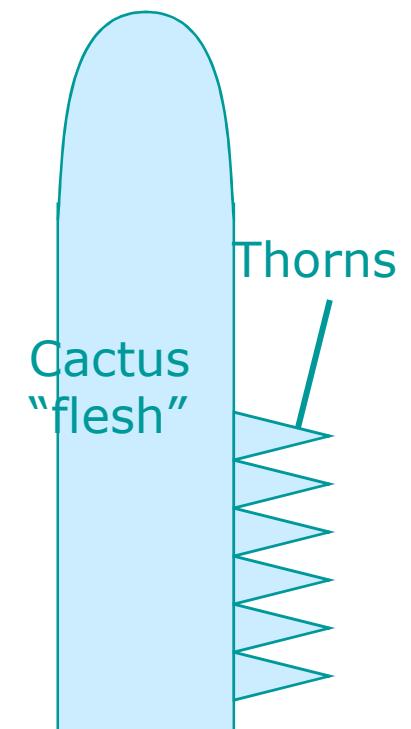
- MPICH-G2:
- CoG Kits, GridPort:
- GDMP, Data Grid Tools, SRB:
- Condor-G:
- Legion:
- Cactus:

MPICH-G2: MPI pe grid

- Reprezinta o implementare completa a Message Passing Interface (MPI) pentru medii eterogen distribuite fizic pe mari distante
 - ◆ Bazata pe implementarea MPI a Argonne MPICH
- Programele sunt execute pe arii mari fara modificari (scalabilitate)
- Altele: MetaMPI, PACX, STAMPI, MAGPIE

Cactus (Allen, Dramlitsch, Seidel, Shalf, Radke)

- Este un mediu moduli portabil pentru a realiza simulari multidimensionale in paralel
- Codul este construit prin linking
 - ◆ Small core (flesh): management
 - ◆ Module neceare app (thorns) pentru:
 - Metode numerice
 - Decompozitie a domeniului
 - Vizualizare. Etc



Calcul de mare incarcare si Condor

- Ce inseamna calcul de mare incarcare
 - ◆ Utilizarea ciclurilor CPU in fiecare zi/sapt/luna/an in conditii neideale (suna cunoscut nu?)
 - ◆ De cate ori se poate executa o simulare X intr-o luna folosind toate masinile disponibile
- Pune accentul pe fiabilitate si fiabilitate

Abordari OOP

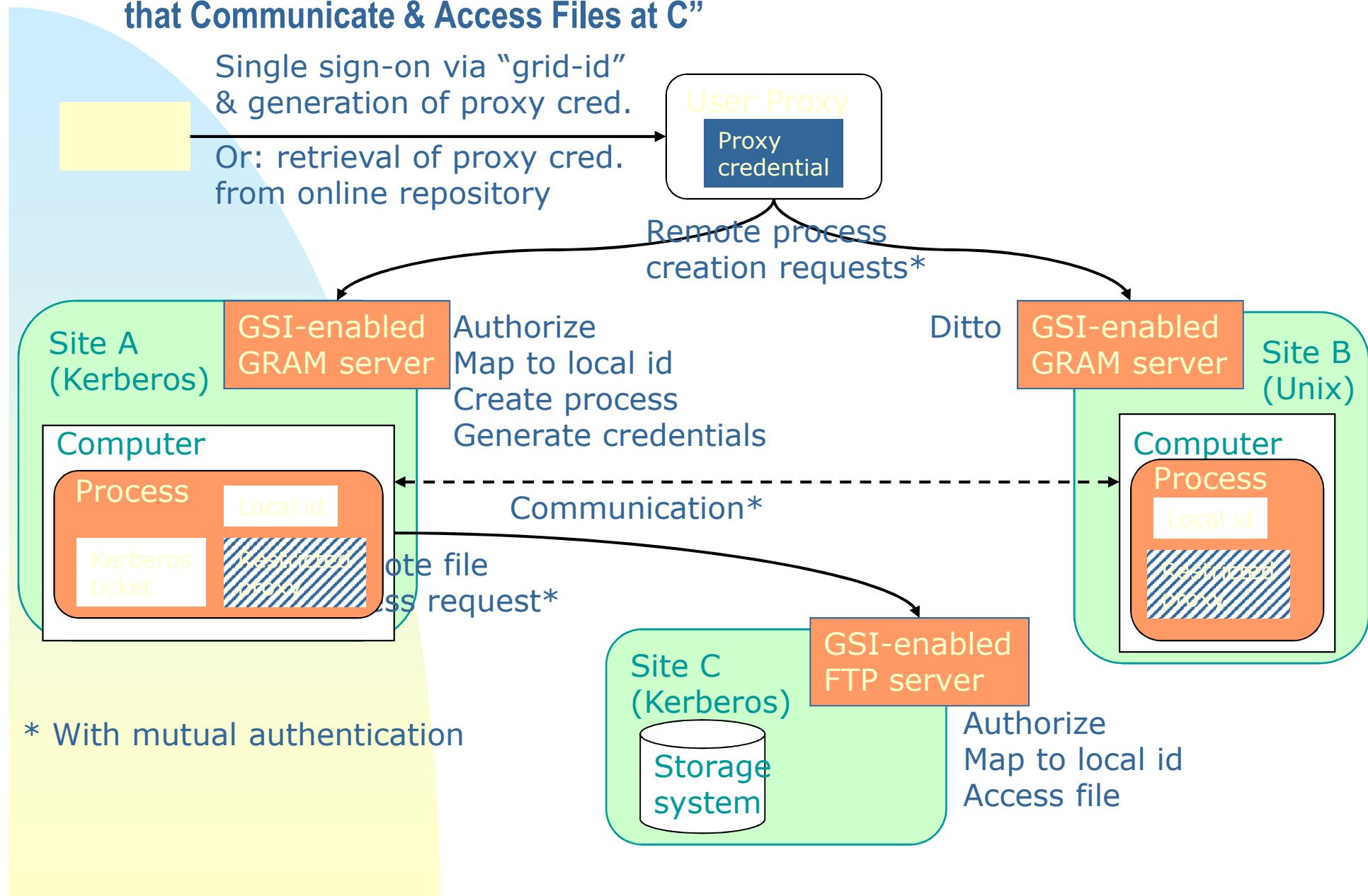
- CORBA pe grid
 - ◆ NASA Lewis, Rutgers, ANL, others
 - ◆ CORBA wrappers pentru protocoale Grid
 - ◆ Exista NISTE SUCCESSE INITIALE
- Legion
 - ◆ U.Virginia
 - ◆ MODELE OBIECT PENTRU COMPOONENTE GRID
(e.g., “vault”=storage, “host”=computer)

Portaluri

- Sunt arhitecturi N-tier care permit ca clienti de mici dimensiuni sa fie conectati cu nivelel intermediare din GRID
 - ◆ Prin clienti de mici dimensiuni se intreleg aplicatii ce pot fi executate sub navigatoare internet
 - ◆ Prin nivel intermediare se intelege Java Server Pages, cu Java CoG Kit, sau utilitati gen GPDK, GridPort
 - ◆ La baza se afla resursele standard GRID
- Sunt numeroase proiecte si aplicatii
 - ◆ Unicore, Gateway, Discover, Mississippi Computational Web Portal, NPACI Grid Port, Lattice Portal, Nimrod-G, Cactus, NASA IPG Launchpad, Grid Resource Broker, ...

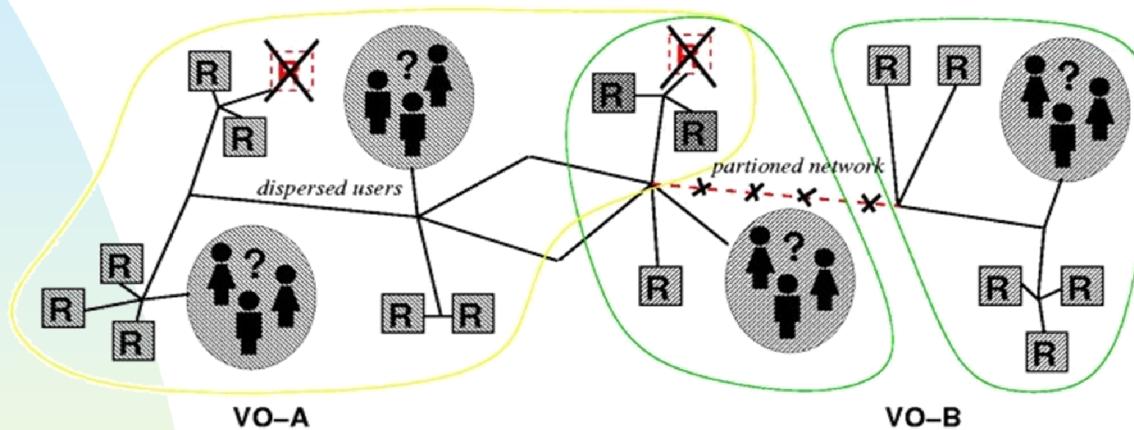
GSI in Action

“Create Processes at A and B that Communicate & Access Files at C”



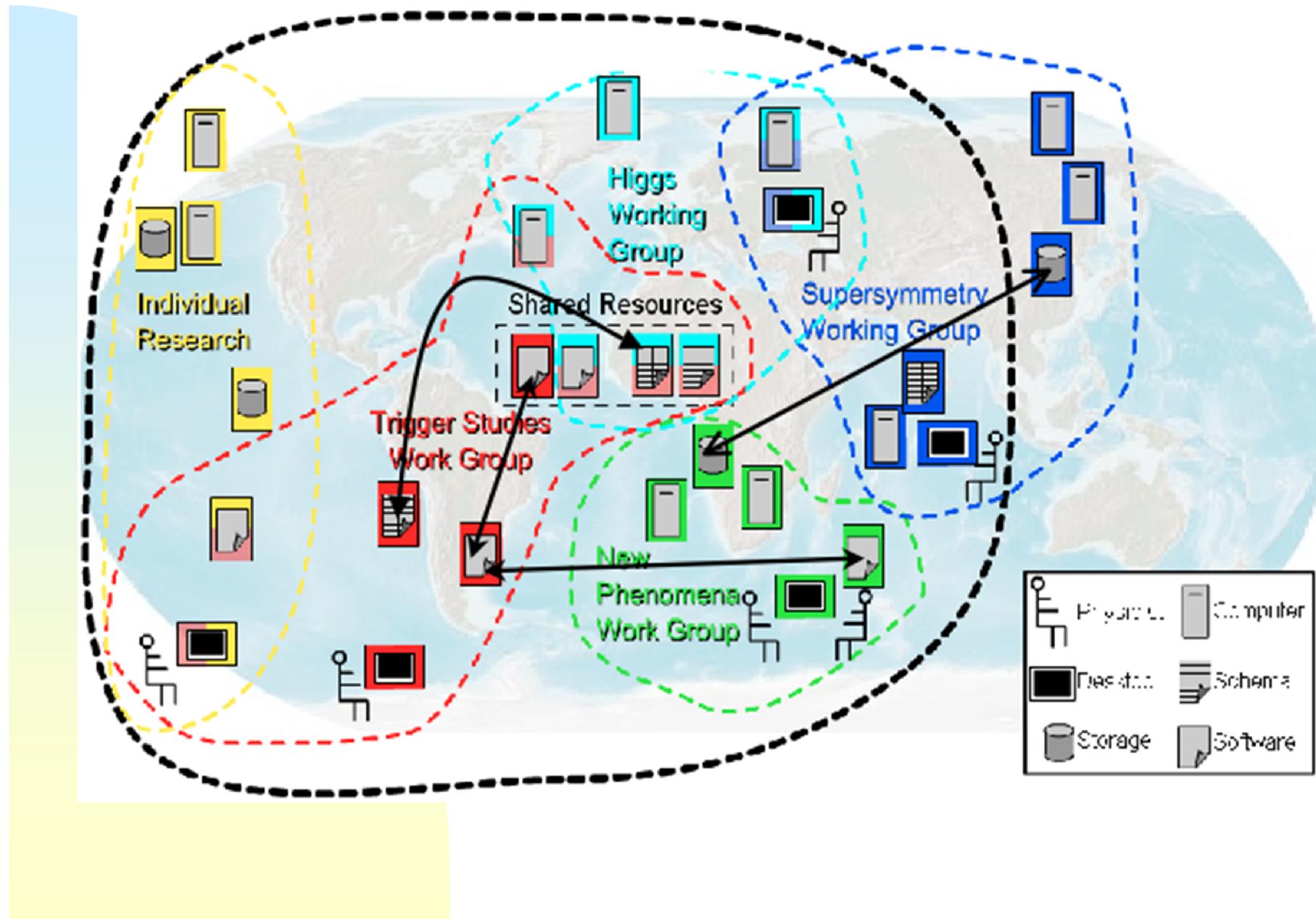
Organizatie virtuala

VO reprezinta un set de indiizi sau organizatii care nu se afla sub un singur control ierarhic,

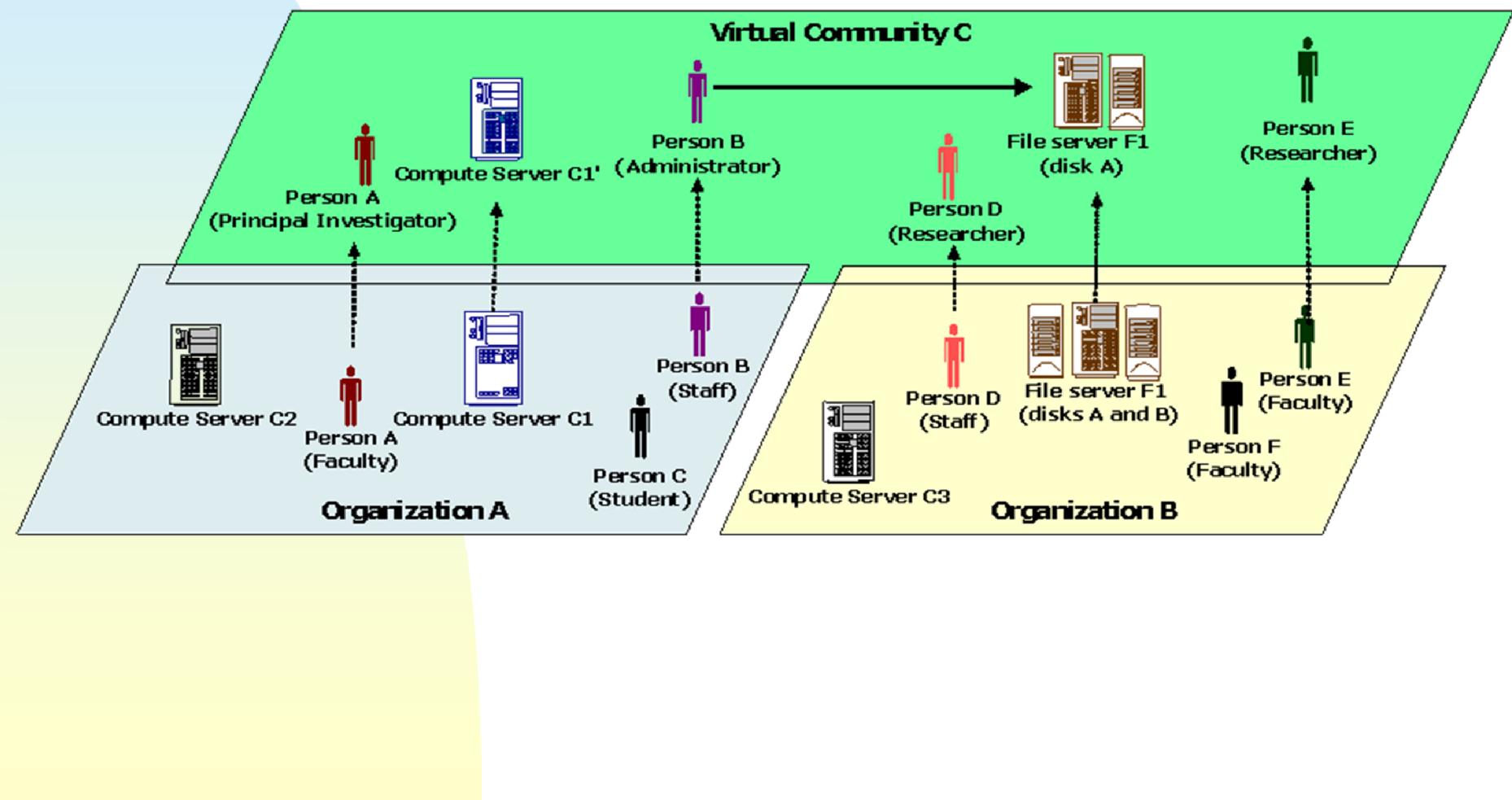


- Utilizatorii sunt de obicei membri la mai multe VO

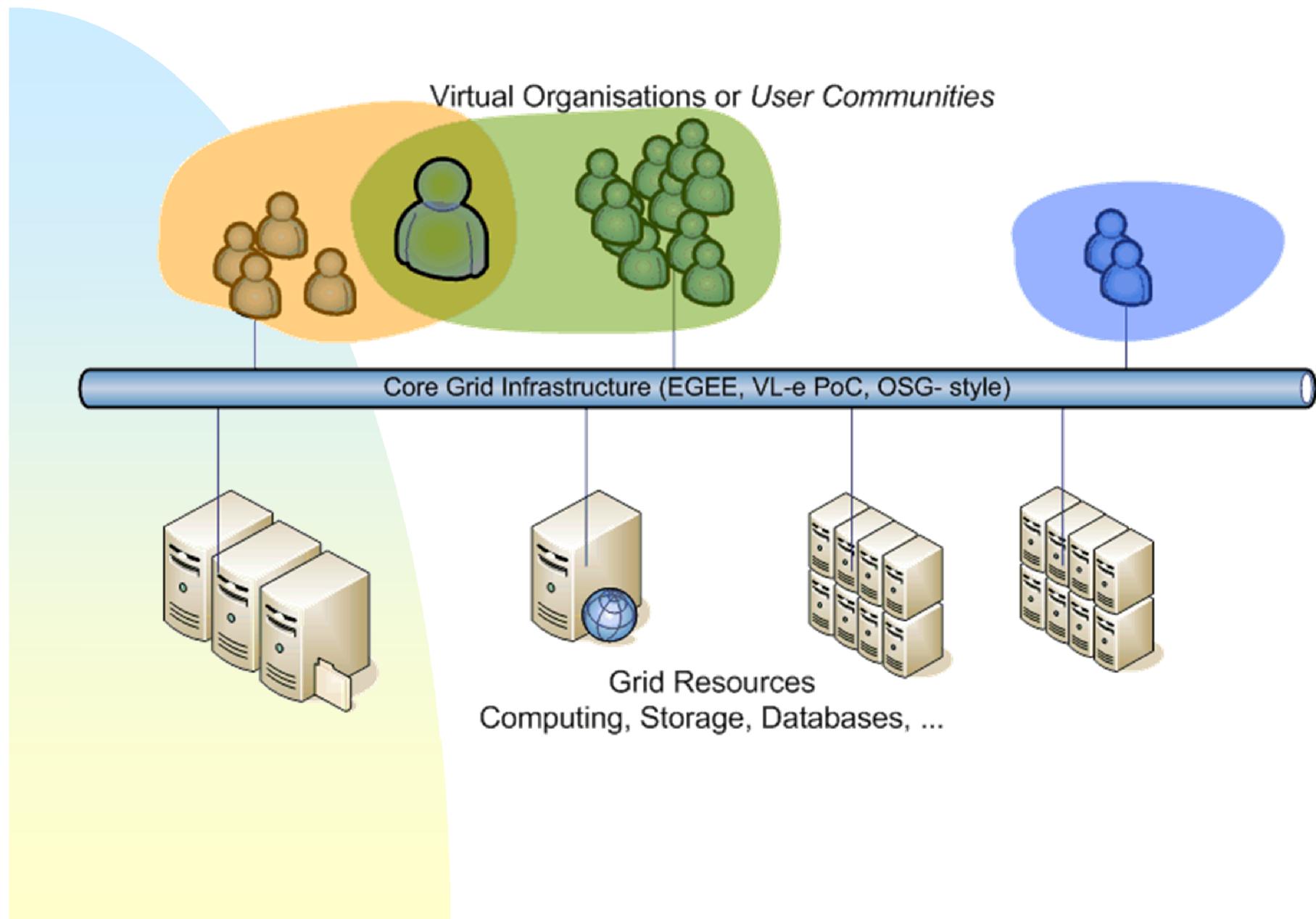
Structura unei organizatii virtuale



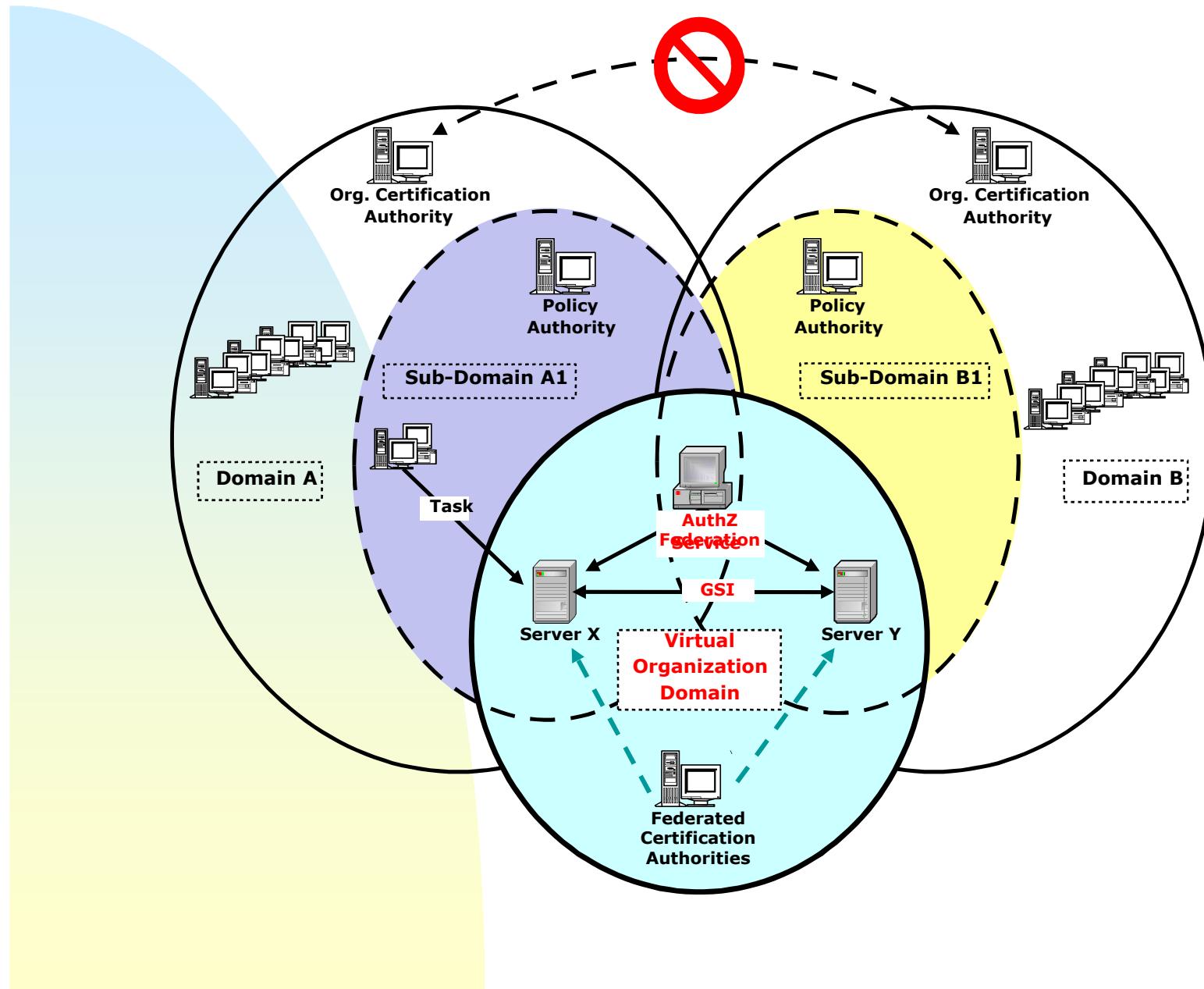
Virtual vs. Organic structure



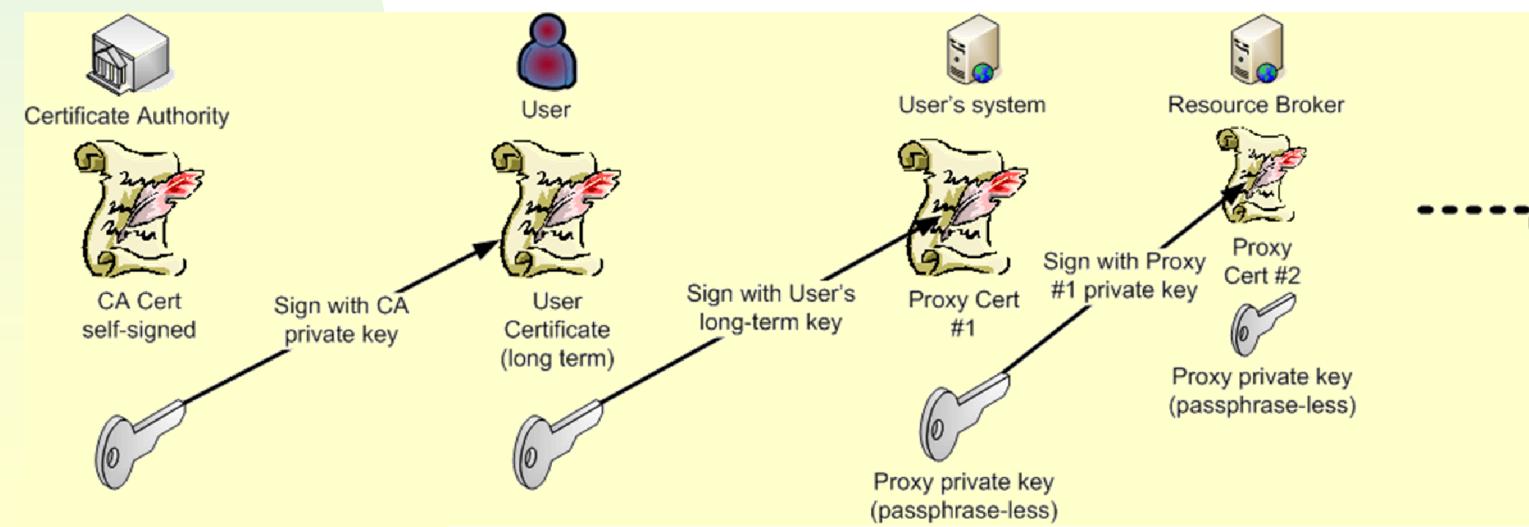
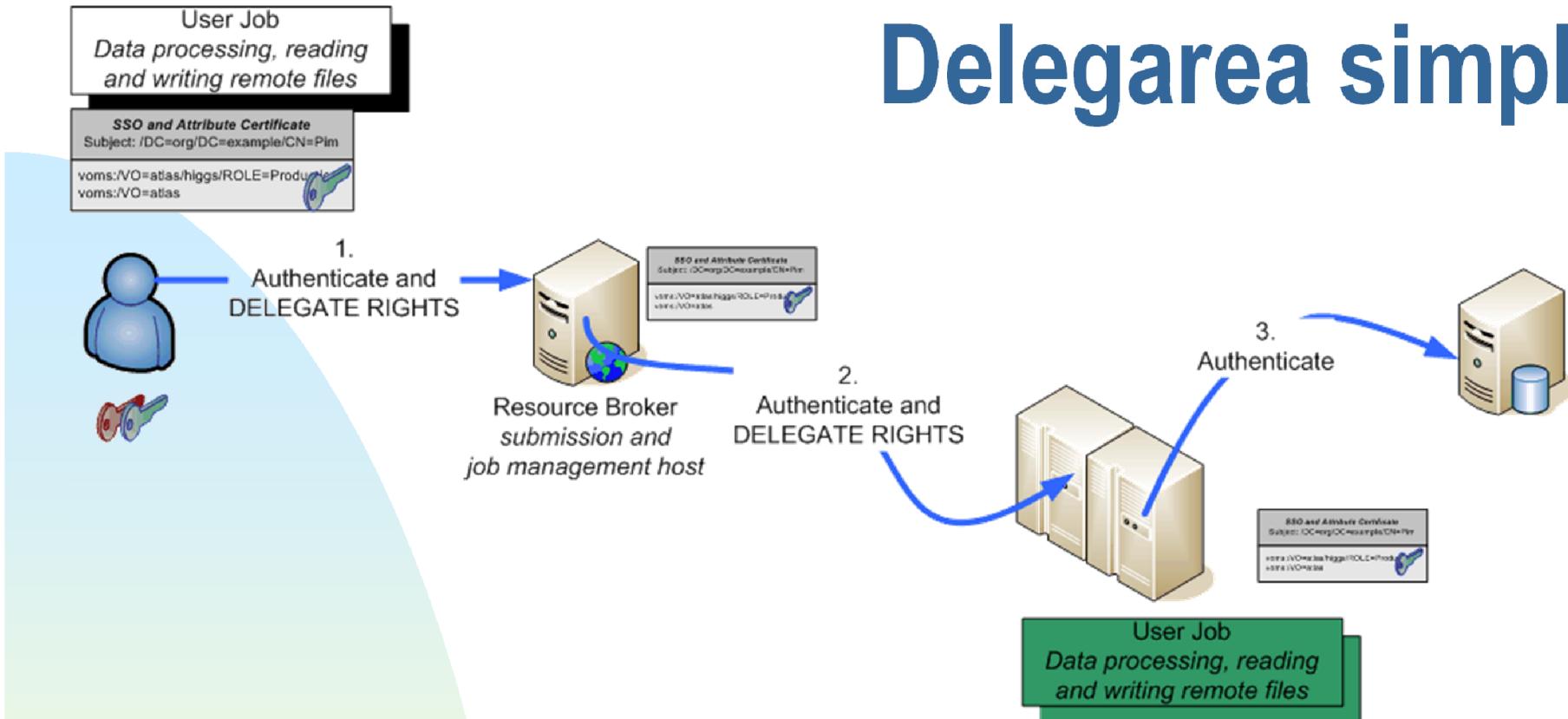
VO-uri si infrastructura



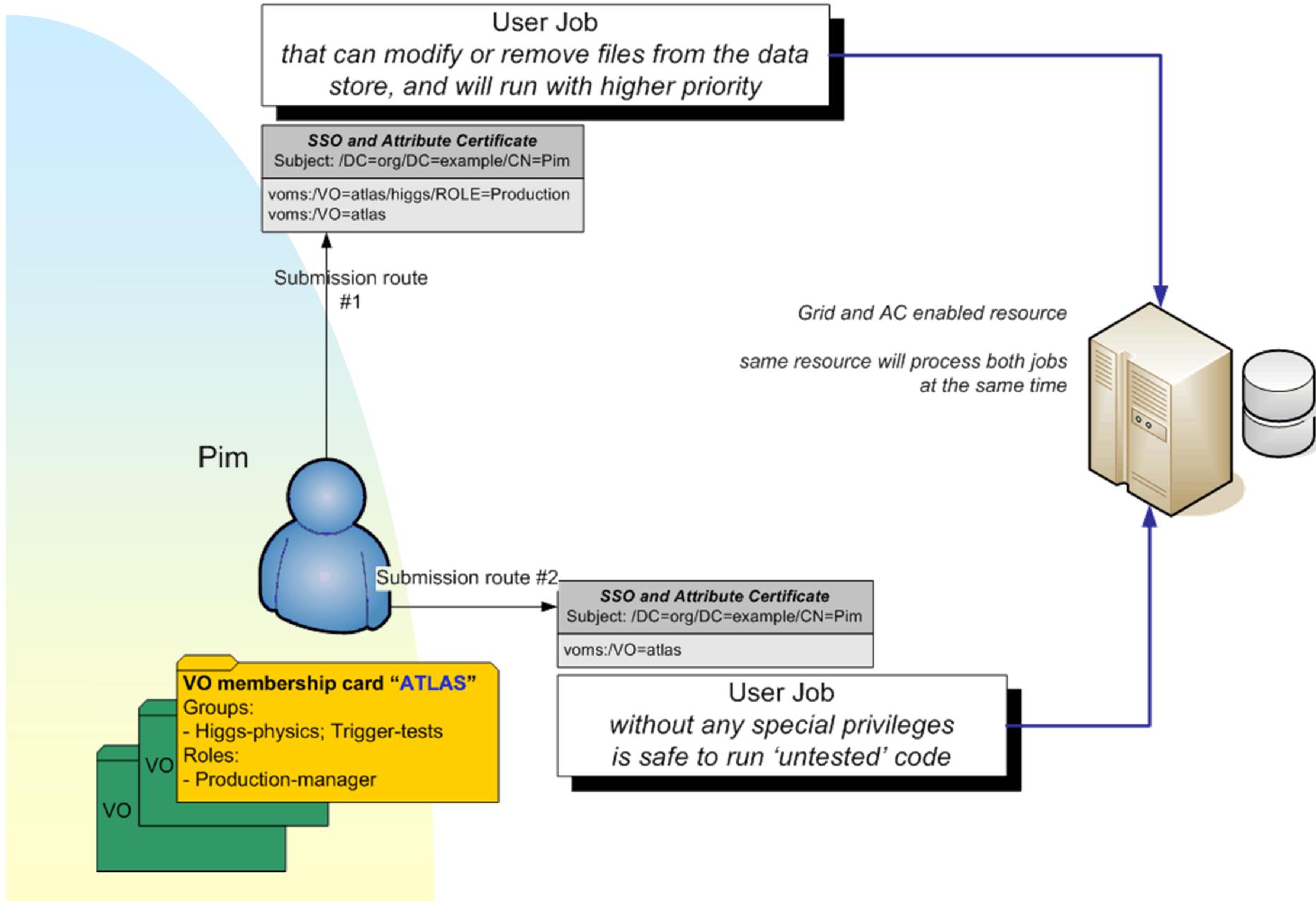
Relatii de incredere



Delegarea simplă



Delegarea specializata sau restrictionata



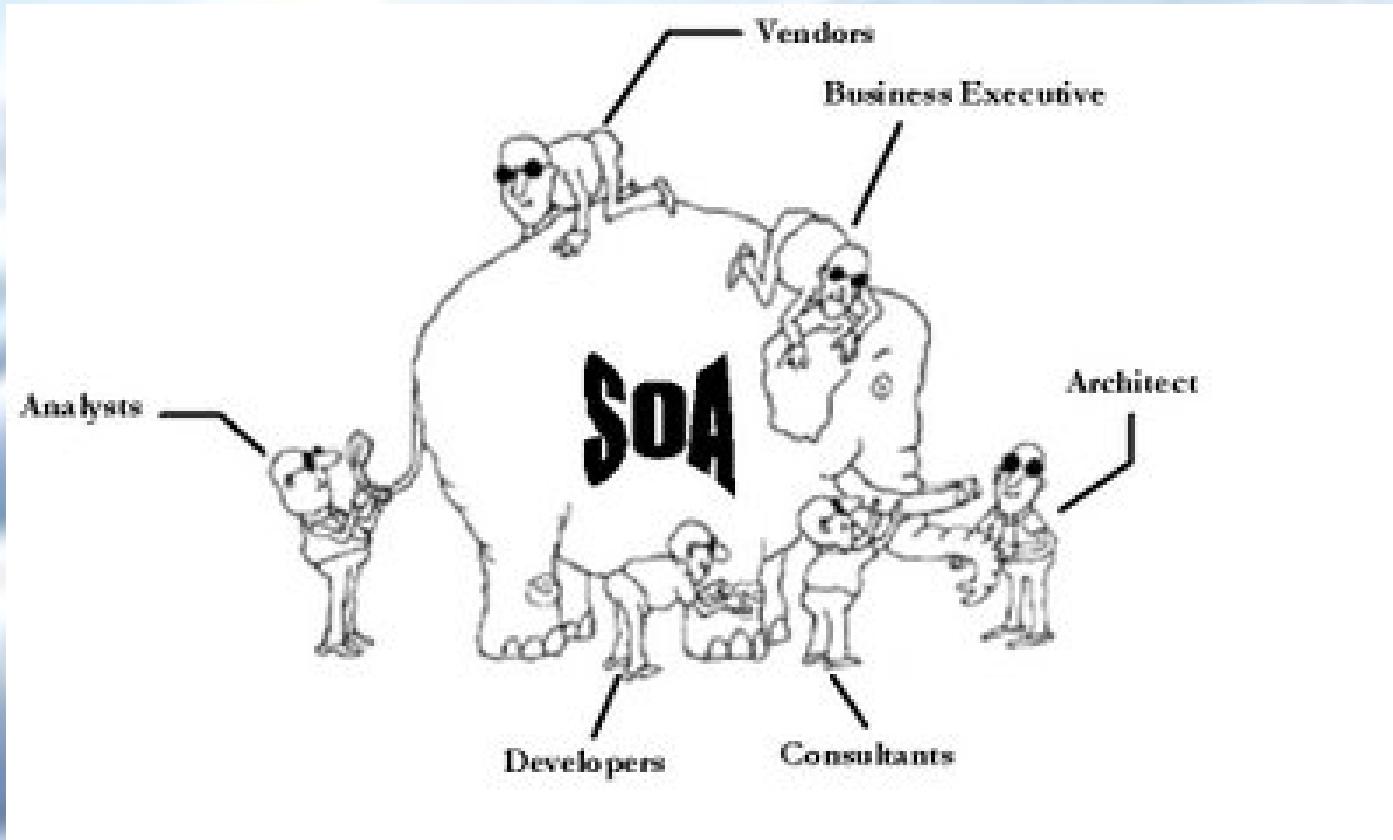
Separararea responsabilitatilor

- Se foloseste un token unic pentru autentificare (“passport”)
 - ◆ Furnizeaza un identificator persistent de incredere (ceva in genul clearance)
 - ◆ El este emis de o parte dar acceptat de toata lumea
 - ◆ Este recunoscut si de furnizorii de resurse si VO
- Autorizare tip Per-VO (“visa”)
 - ◆ Este oferita unei persoane sau serviciu prin intermediul unui VO
 - ◆ Este bazata pe identificator
 - ◆ Este acceptata de catre detinatorii resurselor
 - ◆ Permite obtinerea unui control larg bazat pe rolul in organizatie

Referinte

- <http://web.cecs.pdx.edu/~howe/cs410/>
- <http://www.cs.utsa.edu/~korkmaz/teaching/cs7123/>
- <http://www.nordugrid.org/>
- <https://www.nsf.gov/div/index.jsp?div=ACI>
- <http://www.cloudbus.org/course/>
- <http://toolkit.globus.org/toolkit/>
- <http://www.slac.stanford.edu/econf/C0303241/>
- <http://grid.itim-cj.ro/vo.html>
- http://profs.info.uaic.ro/~alaiba/mw/index.php?title=Sistemul_de_calcul_pe_grid_Globus_Toolkit
- <https://www.globus.org/alliance/publications/papers/anatomy.pdf>

Cursul 9



SOA???

- SOA este un design pattern architectural aparut tot din necesitati practice. A fost propus in 1994 de analistul lui Gartner Yefim V. Natis.
- El spunea ca:
“SOA este o arhitectura software care pleaca de la definirea unei interfete si construieste o topologie a aplicatiei bazata pe compunerea interfetelor, a implementarilor lor sau apelurilor acestor implementari”

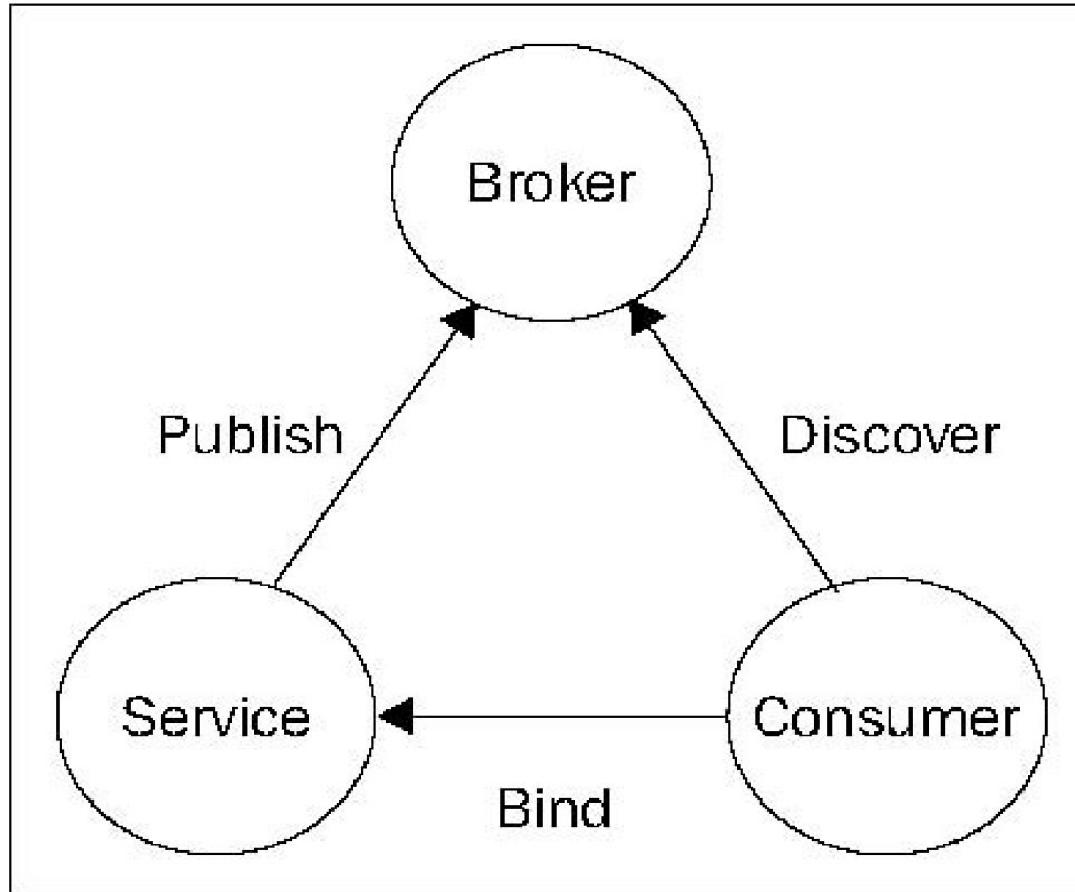
- SOA este bazat pe
 - ◆ Servicii (web)
 - ◆ Mesaje
 - ◆ Descoperire dinamica
- Se porneste de la abordarea deja cunoscuta in proiectarea nivelului de afaceri ca o descompunere de solutii web.

- SOA este un *stil de proiectare care ghideaza toate aspectele crearii si utilizarii serviciilor prin tot ciclul lor de viata* .
- SOA este o modalitate de a defini si oferi o infrastructura IT pentru a *permite aplicatiilor diferite sa schimbe date*

SOA

- Astazi mediile companiilor sunt complexe datorita utilizari unei varietati mari de platforme software si hardware, comunicare bazata pe Internet, etc.
- SOA utilizeaza serviciul ca si componenta reutilizabila
- Serviciile au o granularitate mai mare decat componente

Fundamentele SOA



Abstractizarea serviciului

- Metadatele serviciului specifică:
 - ◆ Localizarea în rețea (adresa de rețea a serviciului)
 - ◆ Descrierea într-un format inteligibil a mesajelor pe care le receptionează și optional returnează.
 - ◆ Defineste ce sabloane suportă în termeni de schimb de mesaje
 - ◆ ...

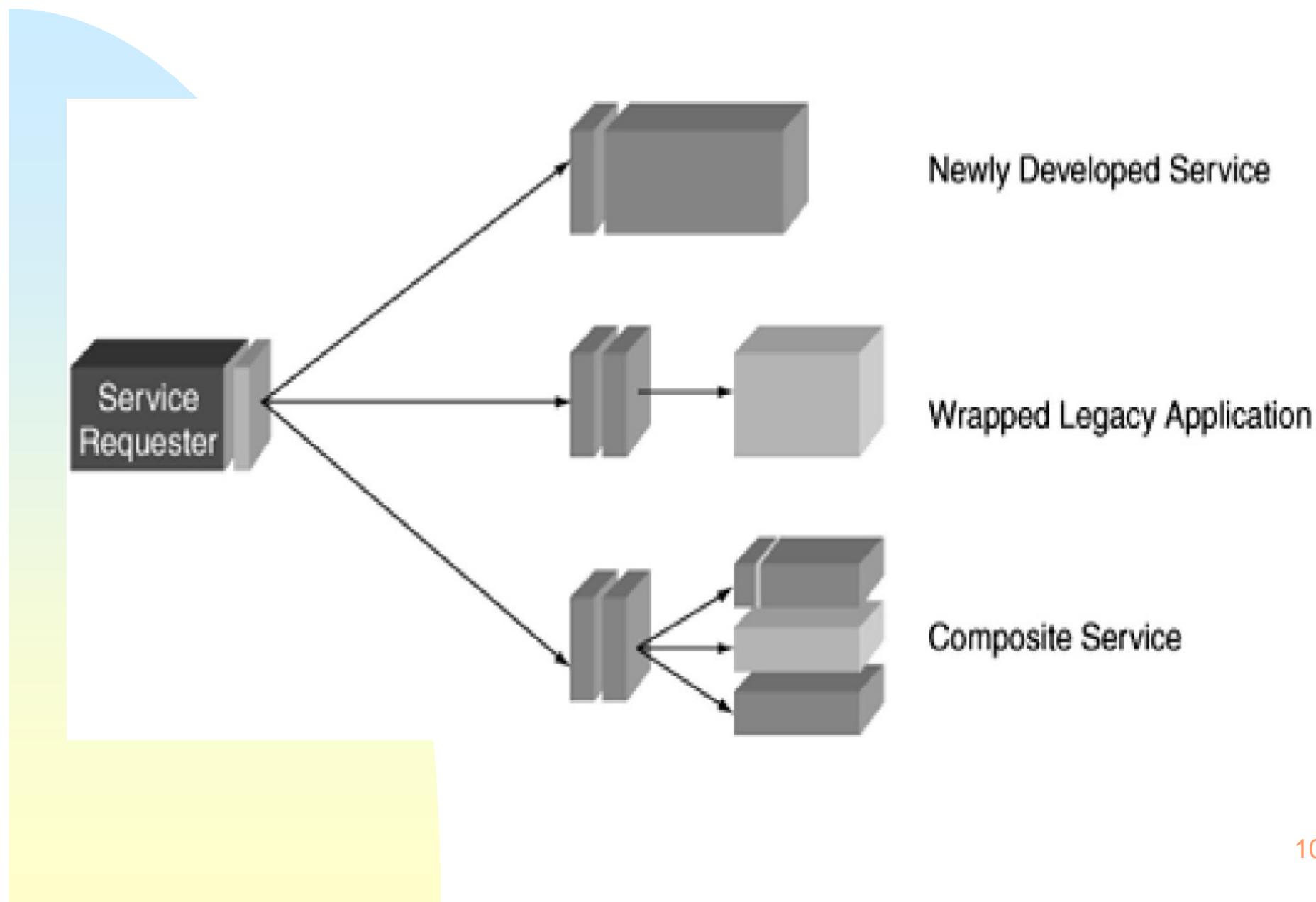
Agentul executabii si stratul de mapare

- Implementarea serviciului este numita si *agent executabil*.
 - ◆ Acesta se executa intr-un mediu de executie
 - ◆ Descrierea serviciului este separata de agentul executabil:
 - ◆ O descriere poate avea mai multi agenti executabili asociati cu aceasta
 - ◆ Un agent poate suporta descrieri multiple

Gestionarul (Handler) serviciilor

- Serviciile sunt publicate de 'furnizor' si sunt legate de 'consumator' prin 'handler'-ul de serviciu.
- Se comporta ca un agent de colaborare intre furnizor si consumator
- Cand serviciul este solicitat, se cauta intre numeroase cai de transmitere de mesaje si prin handleri multipli

Solicitant si furnizor



Tranziția la SOA

- Principala problema in implementarea SOA este complexitatea solutiilor
- Descompunerea proceselor curente in servicii mici este o provocare mare in sine
- Abordari: *Top-down: & bottom up*
- *SOA vs. OO si CBD:*

SOA este implementata

- Obiecte distribuite CORBA, J2EE, COM/DCOM.
- Middleware orientat spre mesaje (MOM)
- WebSphere MQ, Tibco Rendezvous.
- Monitoare CICS, IMS, Encinia, Tuxedo.
- Platforme B2B precum ebXML, RosettaNet.
- Servicii Web
- WebSphere MQ

CORBA pentru SOA (pe vremuri)

- Este un standard deschis.
- Suporta RMI (adica RPC), transmitere de mesaje asincrona, si comunicare de grup
- Ofera securitate, servicii de numire, administrarea tranzactiilor si mesagerie de incredere
- Suporta limbaje de programare multiple
- Ofera CORBA IDL utilizat ca un limbaj de definire a serviciilor

Java si tehnologiile J2EE

- Au numeroase avantaje si dezavantaje
similarare cu CORBA in ceea ce priveste
implementari SOA.
- Similaritati cu CORBA:
 - ◆ Sunt standarde deschise.
 - ◆ sunt tehnologii pentru obiecte distribuite care
ofera suport excelent pentru invocarea de
metode la distanta
 - ◆ solicitantul si furnizorul de serviciu sa utilizeze
aceeasi tehnologie (J2EE respectiv CORBA).

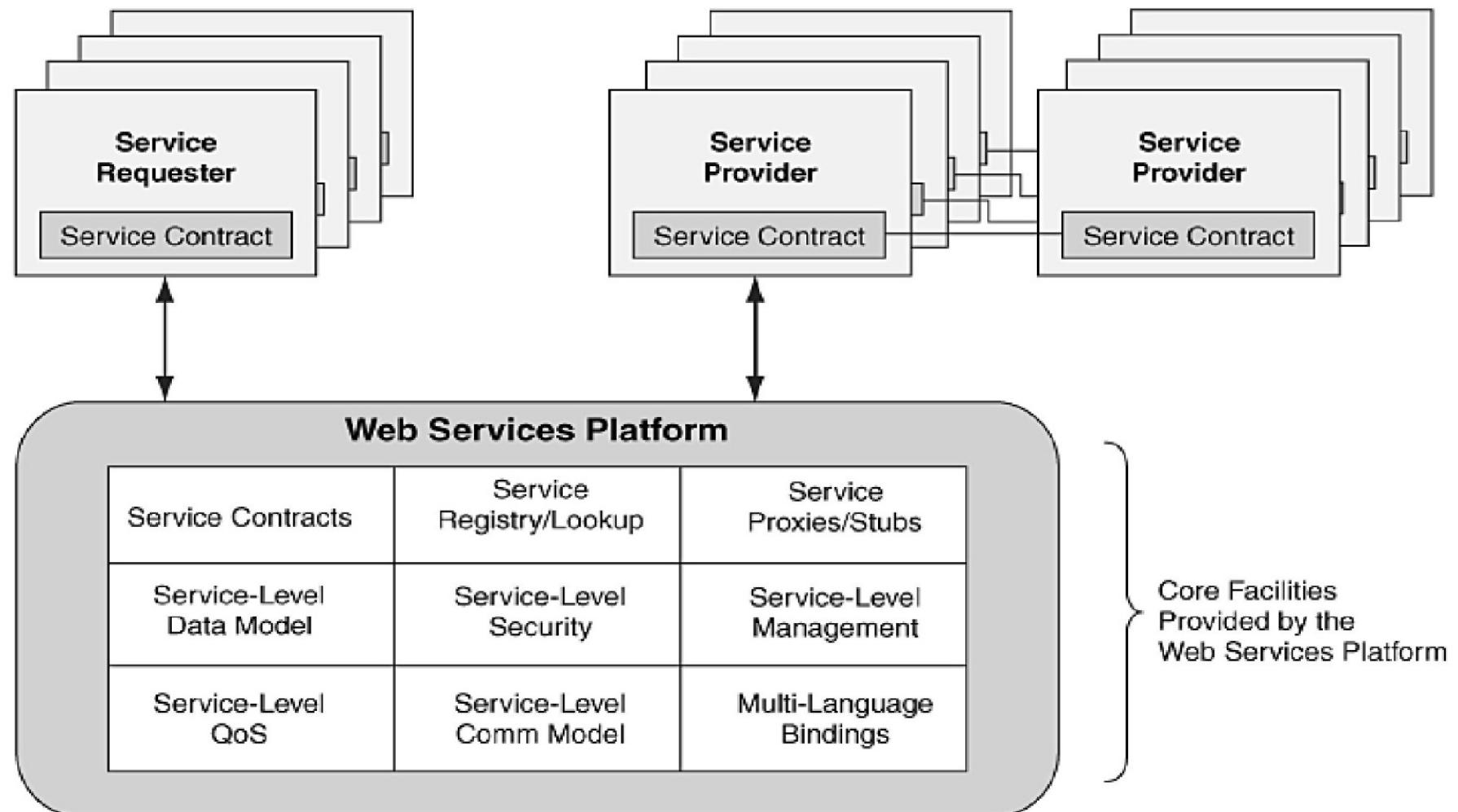
Diferente J2E fata de CORBA:

- CORBA suporta limbaje de programare multiple.
- CORBA ofera CORBA IDL ca un limbaj explicit pentru interfete
- Serviciile Web J2EE comunica nativ utilizand XML si SOAP, pe cand CORBA WSDL mapping comunica utilizand CDL si IIOP.

SOA utilizand Servicii Web

- Avantajul major in implementarea SOA utilizand serviciile Web:
 - ◆ WS sunt peste tot, simple, si neutre la platforma
- Alte avantaje deriva din faptul ca se bazeaza pe succesul WWW:
 - ◆ Utilizarea document markup language precum HTML (sau XML) poate oferi o solutie de interoperabilitate puternica

Platforma de servicii Web



Probleme pana la aparitia norilor

- Companiile IT investeau mult pentru cresterea capacitatii de calcul
 - ◆ Reducerea investitiei initiale
 - 👉 Reducerea cheltuielilor de capital
- Lipsa agilitatii pentru infrastructura IT
- Costuri suplimentare (ridicate) in caz de erori hardware...

Solutii?

- Solutie tip outsourcing
 - ◆ "Cineva" face managementul cererii mele de calcul sau de stocare
 - ◆ "Cineva" imi ofera aceste resurse oricand
 - ◆ "Cineva" se ocupa de nivelul hardware
 - ◆ "Cineva" se ocupa de performanta
 - ◆ "Cineva"...

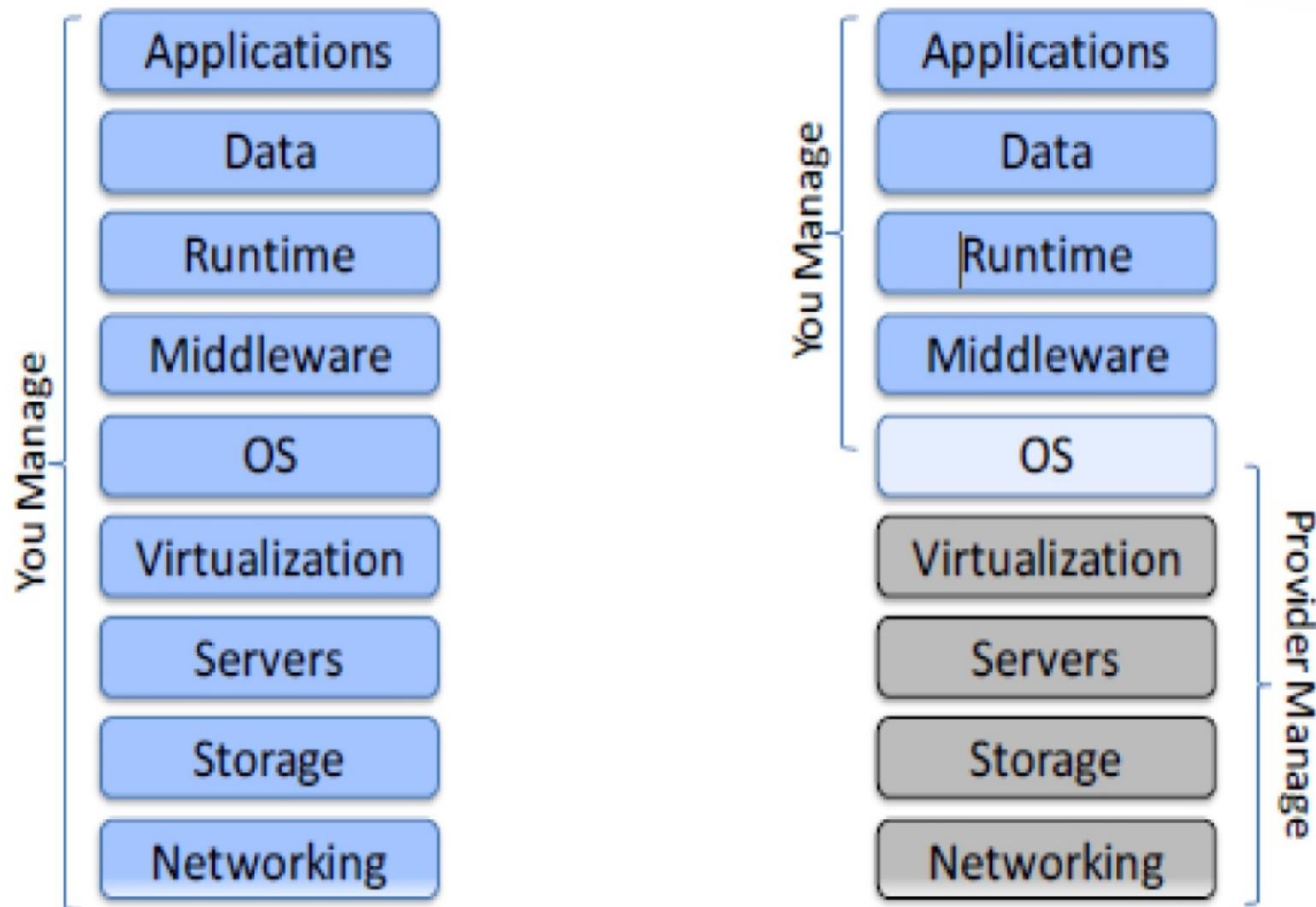
Solutia: IaaS - Infrastructure as a Service

- Furnizorul de IaaS
 - ◆ Are grija de aspecte ce tin de complexitatea infrastructurii IT
 - ◆ Furnizeaza functionalitatile aferente infrastructurii
 - ◆ Garanteaza serviciile aferente infrastructurii furnizate
 - ◆ Calculeaza un pret raportat la resursele utilizate
- Problemele tipice ale unui furnizor de IaaS - Infrastructure as a Service se refera la faptul ca clienti: ...

Strategii posibile de rezolvare

- *Strategie 1:* Alocarea cate unei masini fizice pentru fiecare client ?
- *Strategie 2:* Pregatirea unui cluster neomogen cu noduri pre-instalate pentru diverse potentiiale cereri
- *Strategie 3:* Utilizarea unui grid .situatia se imbunatateste dar sun probleme de sistem de operare si adaptarea aplicatii
- *Strategie 4:* Utilizarea virtualizarii in oricare din mediile anterior mentionate.
- Aceasta ofera:... dar nu raspunde bine cresterii de butere de calcul
- *Strategia 5:* utilizarea norulet (virtualizare in nor)

Modelul Traditional (*on-premise*) versus IaaS



Ce este noruleul

Definiția NIST

- Cloud computing este un model de plată funcție de utilizare care permite accesul, pe baza de rețea, la cerere, convenabil, disponibil, la o grupare de resurse de calcul configurabile (ex., retele, servere, stocare, aplicații, servicii care pot fi oferite rapid și cu un efort de administrare minimal sau cu interacțiune minima cu furnizorul de serviciu.

Characteristicile de baza

- Auto-service la cerere
 - ◆
- Acces de oriunde la retea
 - ◆
- Grupare a resurselor independente de locatie.
 - ◆
- Plata dupa cat consumi

Componente CC

- Clienti:
 - ◆
- Centre de date
- Servere virtualizate:
- Servere distribuite

Tipuri de Nori

- **Privat**



- **Public**



- **Hibrid**



Software as a Service (SaaS)

- Model in care o aplicație este gazduita ca serviciu pentru clienții care o acceseaza via Internet
- Furnizorul se ocupa de impachetare, actualizare si mentinerea infrastructurii in rulare
- Aplicatiile sunt livrate printr-un navigator la mii de clienti utilizand o arhitectura multi-utilizator

Tipuri de software care conduc la modelul SaaS

- Tipic, software-ul care efectueaza un task simplu fara a fi necesara o interactiune cu alte sisteme
- Pentru clienti car nu sunt inclinati sa efectueze dezvoltare de software dar au cerinte de aplicatii puternice
- Cunoscut ca si **Application-as-a-service (AaaS)**,
-

Aplicatii SaaS si exemple

- Administrarea resurselor clientilor (CRM)
- Conferinte video
- Administrare de servicii IT
- Gestiunea conturilor
- Analitice asupra Webului
- Administrare de continut Web

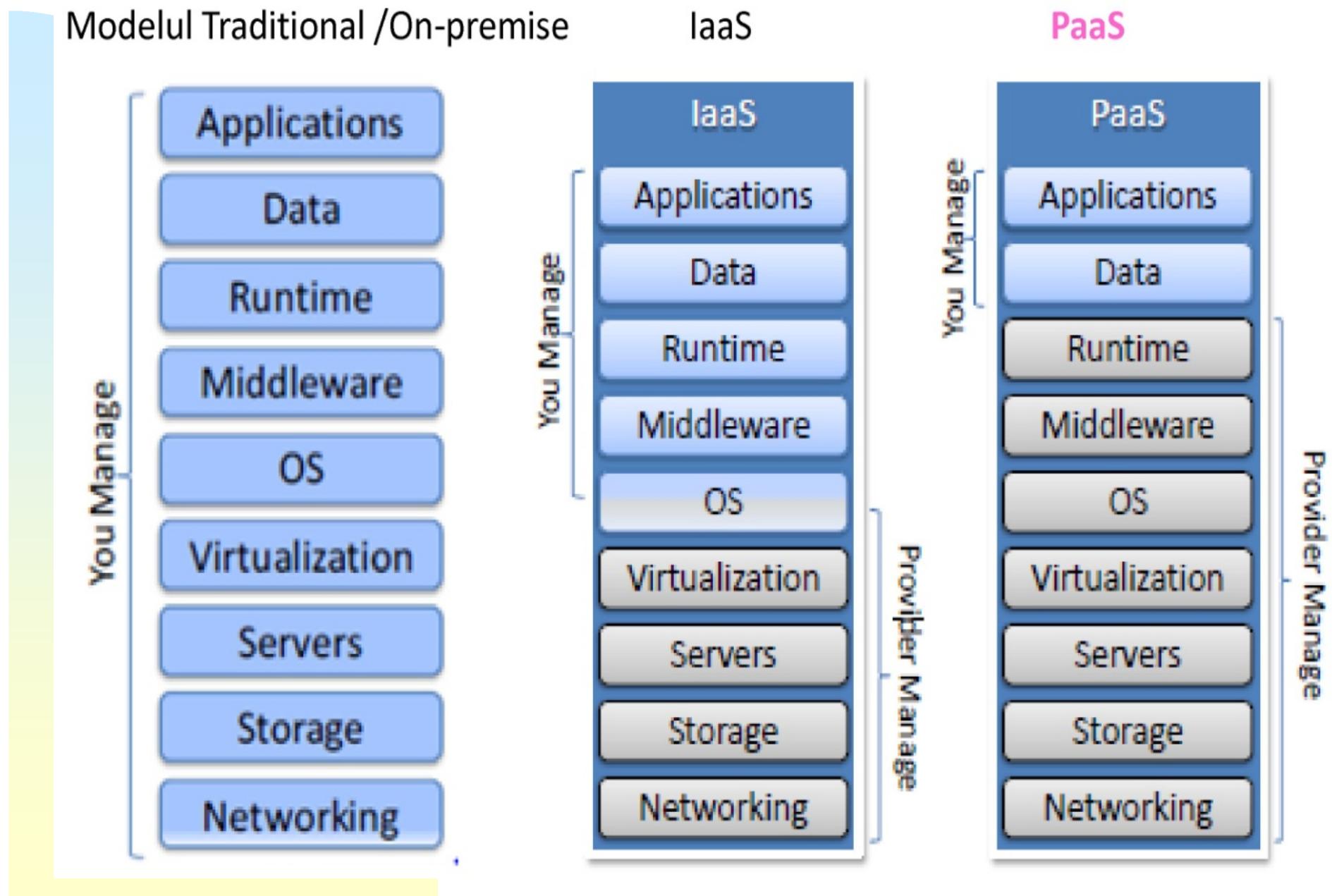
Organisation	Cloud Service
fluidOpS	eCloudManager
	SAP Edition
Google	Google Docs
Google	Google Maps
	API
Google	OpenSocial
Open ID Foundation	OpenID
Microsoft	Office Live
Salesforce	Salesfoice.com

Caracteristicile cheie ale SaaS

- Aplicatiile sau serviciile software sunt stocate la distanta
- Un utilizator poate accesa aceste servicii sau aplicatii software via Internet
- In majoritatea cazurilor, un utilizator nu trebuie sa instaleze nimic pe masina gazda,
- Tot ceea ce se cere este un navigator web pentru accesarea acestor servicii
-

Platform as a Service (PaaS)

- Un alt model de livrare a aplicatiilor, cunoscut si ca cloudware sau servicii web in nor
- Serviciile includ:
 - ◆ Proiectarea aplicatiilor, dezvoltare, testare, lansare, si gazduire.
 - ◆ Colaborarea echipelor, integrarea serviciilor Web,



Exemple

APIuri

- Oferă anumit suport pentru a ajuta la crearea interfetelor utilizator, și sunt în mod normal bazate pe HTML sau JavaScript.
- Suportă interfețe de dezvoltare Web precum SOAP și REST care permit construirea de servicii Web multiple, uneori numite mashups
-

Organisation	Cloud Service
Akamai	EdgePlatform
Facebook	Facebook Platform
Google	App Engine
Microsoft	Azure
Microsoft	Live Mesh
Net Suite	SuiteFlex
Salesforce	Force .com
Sun	Project Caro line
Zoho	Zoho Creator

- Se gaseste in una din urmatoarele tipuri de sisteme:
 - ◆ **Facilitati de dezvoltare add-on**
 - ◆ **Medii de sine statatoare -**
 - ◆ **Mediile de livrare numai a aplicatiilor -**
 - ◆ **Nu include ...**

Infrastructure as a Service (IaaS or HaaS)

Hardware as a Service (HaaS)

- ◆ SaaS si PaaS ofera aplicatii catre clienti, HaaS nu o face.
- ◆ Ofera hardware a.i. clientii pot sa instaleze ceea ce doresc
- In locul cumpararii, furnizorii de servicii inchireaza aceste toate resursele necesare;

Necesitati si beneficii ale HaaS

- Necesitati: Contracte de servicii (SLA)

Exemple

Organizatie	Serviciu Nor	Organizatie	Serviciu N or
Amazon	Elastic Compute Cloud (EC2)	Joyent	Accelerator
Amazon	Dynamo	Joyent	Connector
Amazon	Simple Storage Service (S3)	Joyent	BingoDisk
Amazon	SimpleDB	Nirvanix	Storage Delivery Network
Amazon	CloudFront	Openflow	OpenFlow
Amazon	SQS	Rackspace	Mosso Cloud Sites
App Nexus	AppNexus Cloud	Rackspace	Mosso Cloud Storage
Bluelock	Virtual Cloud Computing	Rackspace	Mosso Cloud Servers
Bluelock	Virtual Recovery	Skytap	Skytap Virtual Lab
Dropbox	Dropbox Cloud Storage	Tenemark	Infinistructure
Emulab	Emulab Network Testbed	Globus	Nimbus
ENKI	Virtual Private Data Centers	todo GmbH	flexIT
Reservoir	Open Nebula	UCSB	Eucalyptus
RexiScale	RexiScale Cloud Computing	Zimory	Zimory Public Cloud Market
GoGrid	Cloud Hosting	Zumodrive	Hybrid Cloud Storage
GoGrid	Cloud Storage	IOgen	Mongo DB
Google	Google Big Table	IOgen	Babble Application Server
Google	Google File System		
HP	iLO		
HP	Tycoon		

Availability si Reliability in IaaS

- Pentru resurse computaționale:
 - ◆ Monitorizarea
 - ◆ Backup permanent
 - ◆ Mutarea masinilor virtuale
- Pentru resurse de stocare:
 - ◆ Mentinerea replicilor
 - ◆ Backup regulat
 - ◆ Pentru resurse de comunicare: Construirea de conexiuni redundante

Manageability si Interoperability in IaaS:

- Clientii au control deplin asupra infrastructurii virtualizate care le-a fost alocata
- Resursele virtualizate pot fi rezervate printr-un proces automat ce respecta o politica de pre configurare
- Starea resurselor virtualizate trebuie permanent monitorizata
- Utilizarea resurselor este inregistrata si mai apoi convertita de catre un sistem de facturare (*billing system*)

Pentru resurse de calcul :

- Furnizarea de operatii de baza asupra masinilor virtuale: crearea, terminare, suspendare, *snapshot'*

Pentru resurse de stocare:

- Monitorizarea si inregistrarea spatiului folosit, precum si a accesului read/write pentru fiecare resursa virtuala de stocare
-

Pentru resurse de comunicare:

- Monitorizarea si inregistrarea latimii de banda consumata pentru fiecare legatura virtuala
-

Performance si Optimization in IaaS

- Resursele fizice trebuie sa fie utilizate la un nivel ridicat intre clienti diferiti
- Resursele fizice formeaza o ferma de resurse care furnizeaza putere de calcul pentru procesari paralele
- Pentru resurse de calcul:
 - ◆ "Ridicarea" unei masini virtuale luand in calcul aspecte legate de *load balancing*
 - ◆

Accesibility si Portability in IaaS

- Clientii trebuie sa aiba control si acces la infrastructura fara sa fie nevoiti sa instaleze soft local sau sa apeleze la un dispozitiv hardware special
-

Pentru resurse de calcul:

- ◆ Furnizorul de Cloud ofera un portal Web pentru gestiunea resurselor
- ◆

Pentru resurse de stocare:

- Furnizorul de Cloud ofera un portal Web pentru gestiunea resurselor de stocare

Tipuri de virtualizare:

- ***Bare metal:*** VMM (Virtual Machine Monitor)
- ***Hosted::***

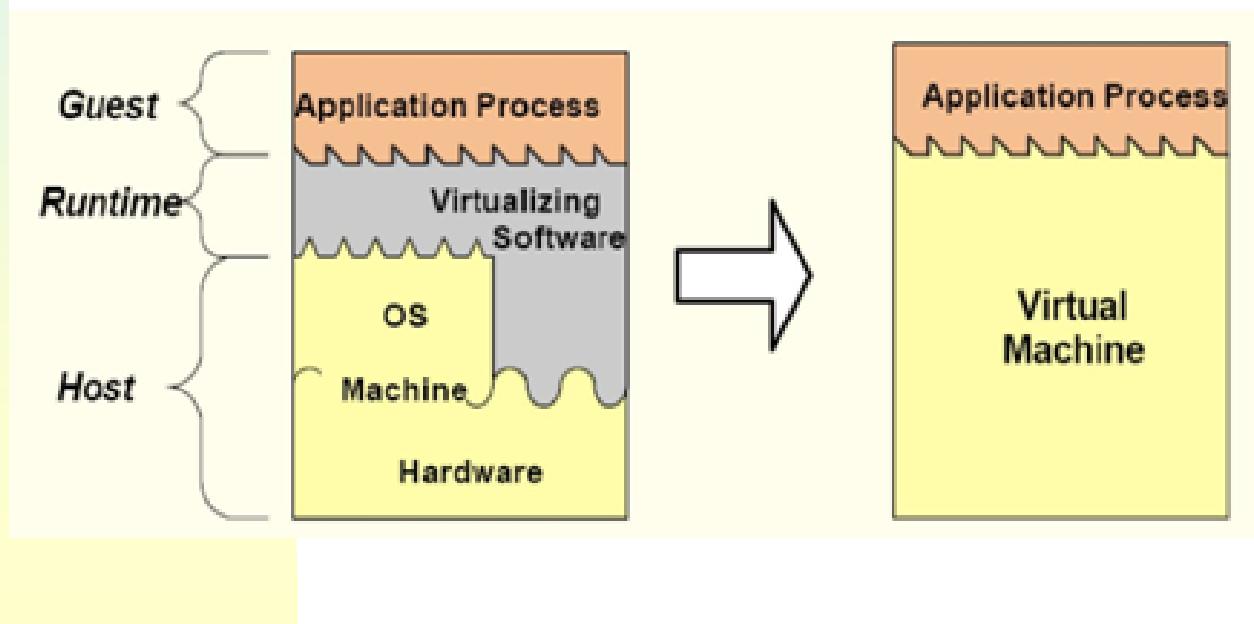
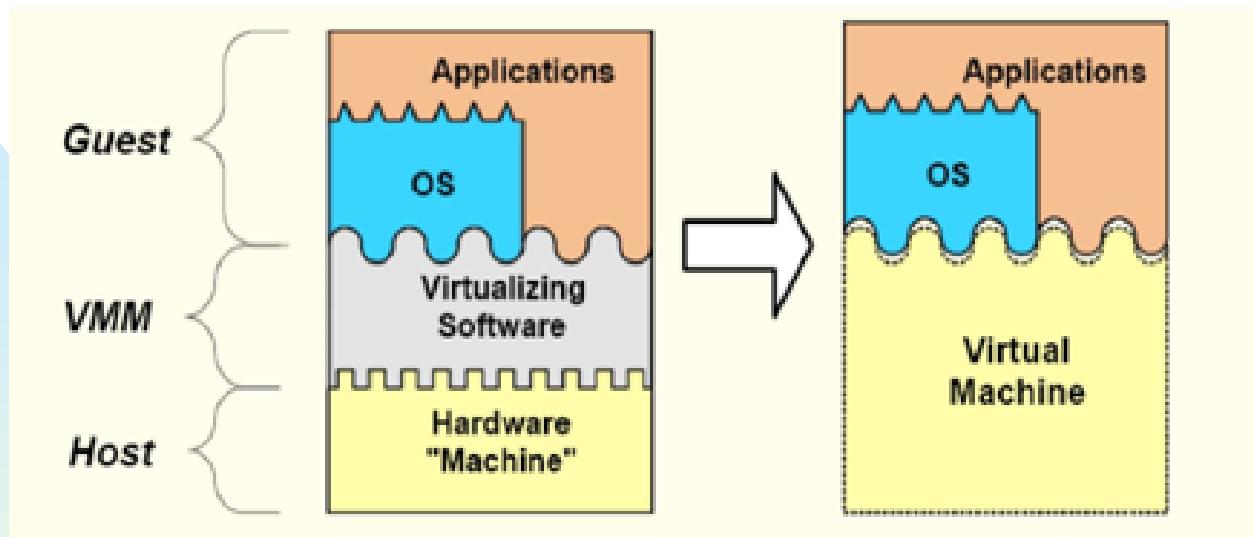
Abordari ale virtualizarii:

- ***Full-Virtualization::***
- ***Para-Virtualization::***

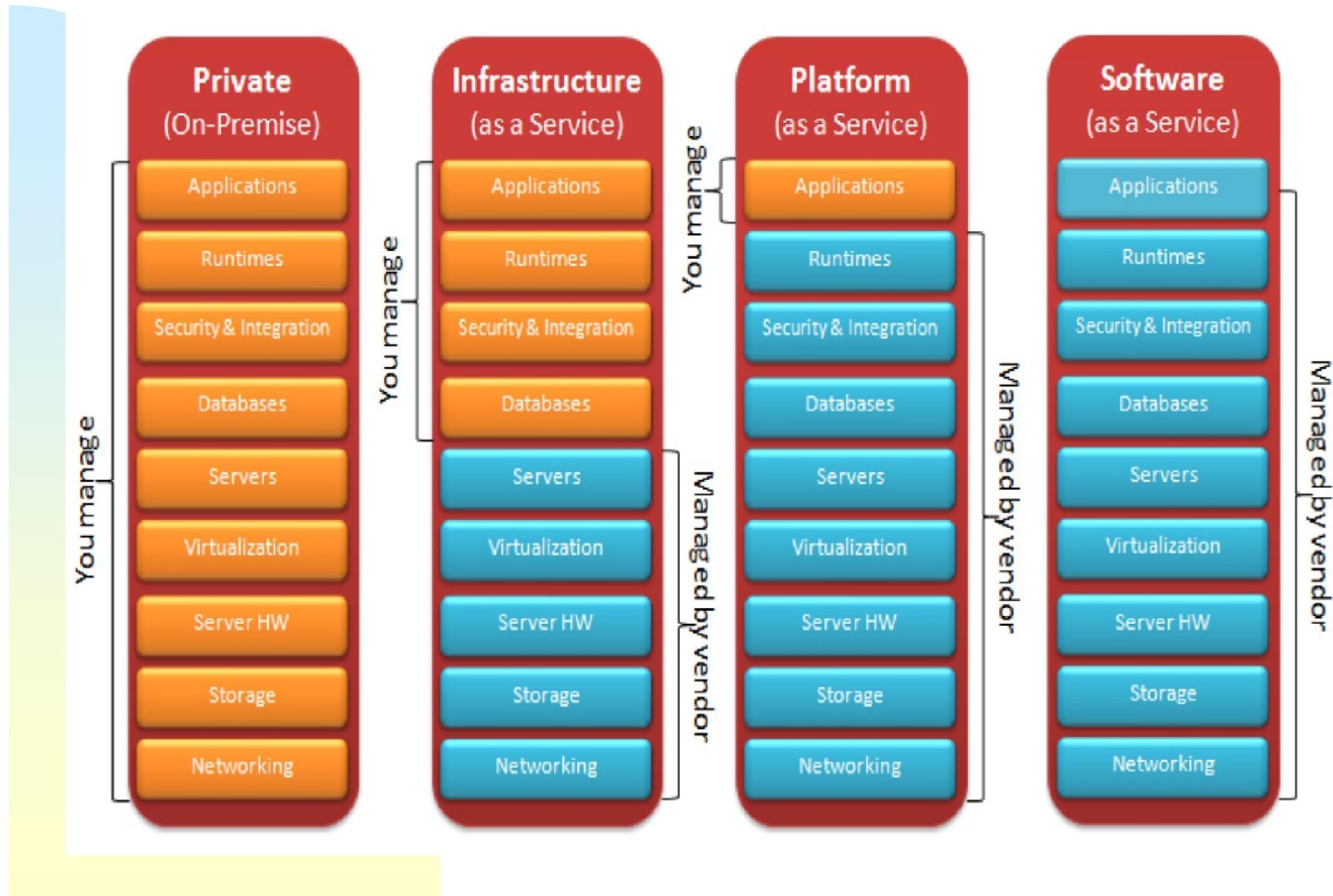
Virtualizare vs Paravirtualizare

- Utilizata pentru a accesa servicii in nor
- Centrul de date la distanta poate livra servicii in format virtualizat
- Virtualizarea completa are succes deoarece se supune urmatoarelor scopuri:
 - ◆ Partajarea
 - ◆ Izolarea
 - ◆ Emularea hardware
 - ◆ Intregul sistem este emulat (BIOS, drive etc)

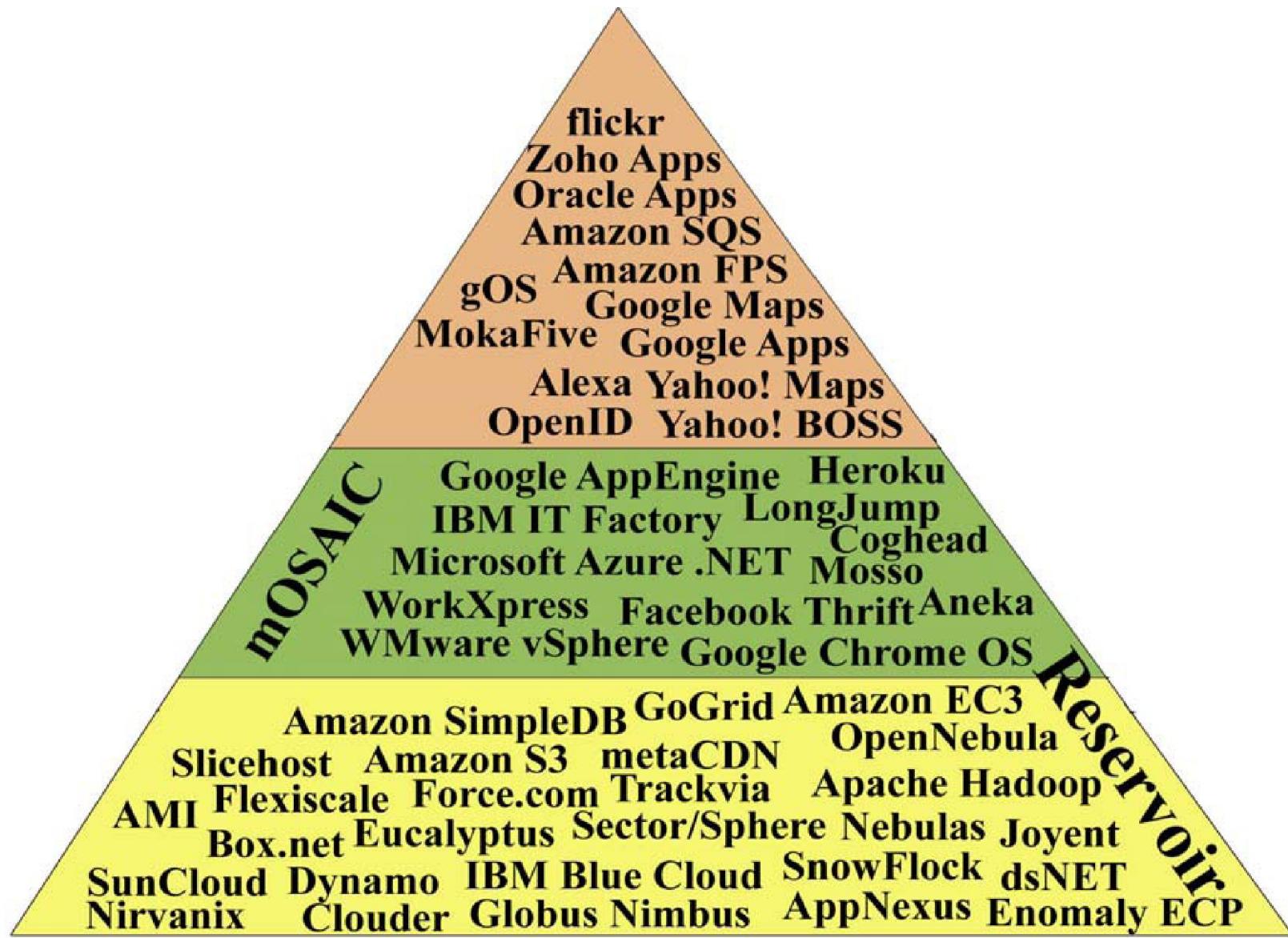
Tehnici de virtualizare in iaas



Furnizori de servicii Cloud (IaaS si/sau PaaS si/sau SaaS)



Piramida norului



Storage as a Service

- Cunoscuta si ca spatiu de disc la cerere
- Abilitatea de a oferi stocare care exista fizic distribuit
- Inseamna ca o terță parte inchiriază spațiu pe
- **Exemple de furnizori** Google Docs, provideri de email pe Web:Gmail, Hotmail, și Yahoo! Mail sau Flickr, Picasa, YouTube etc

Database as a Service (DaaS)

- Baze de date: depozitari de informatii cu legaturi care ajuta la cautari de date.
- Baze de date distribuite: Amazon SimpleDB,
- Au transparenta locatiei
- De fapt o baza de date distribuita cu algoritmi avansati data mining

Information as a Service

- Se referă la abilitatea de a consuma orice tip de informație gazduita la distanță
- Ex. Informatii de pret a articolelor, validarea adreselor, raportarea creditului,
- Printr-o interfață bine definită precum un API
-

Communication as a Service (CaaS)

- CaaS este o solutie de comunicare pentru companii gen outsource
- Oferă
 - ◆ Servicii de voce peste IP (VoIP),
 - ◆ Mesagerie instantană (IM), și
 - ◆ Facilități de conferințe video
 - ◆ Facilități avansate

Identity as a Service

- Ofera o identitate digitală—pentru a descrie utilizatorul
 - ◆
- Aplicațiile in-house se bazează pe servicii precum Active Directory pentru a oferi verificarea identității.
- Norii ar trebui să utilizeze propriile servicii de identitate.
 - ◆ Dacă sunt utilizate serviciile Amazon, se identifică utilizatorul folosind o identitate Amazon.
 - ◆ Google's App Engine necesită un cont Google
 - ◆ Windows utilizează Windows Live ID pentru aplicațiile CC Microsoft.

Monitoring as a Service (MaaS)

- Exemplu: oferirea de tip outsource de securitate, primar pe platforma de afaceri care permite realizarea de afaceri pe Internet.
 - ◆
- *Security-as-a-service*, este abilitatea de a oferi servicii de securitate de baza la distanta peste Internet.
 - ◆

Process as a Service Integration as a Service

Process-as-a-service

- Resursele la distanta care sunt cuplate impreuna, ex. servicii & date gazduite in aceeasi resursa CC sau la distanta pentru a crea procese de afaceri

Integration-as-a-service

- Abilitatea de a livra o stiva completa de integrare in nor incluzand interfantarea cu aplicatiile, medierea semantica, control al fluxului, proiectarea integrarii etc.

MaaS si TaaS

Management/governance-as-a-service (MaaS and GaaS)

- Orice serviciu la comanda care ofera abilitatea de a administra unul sau mai multe servicii
-

Testing-as-a-service (TaaS)

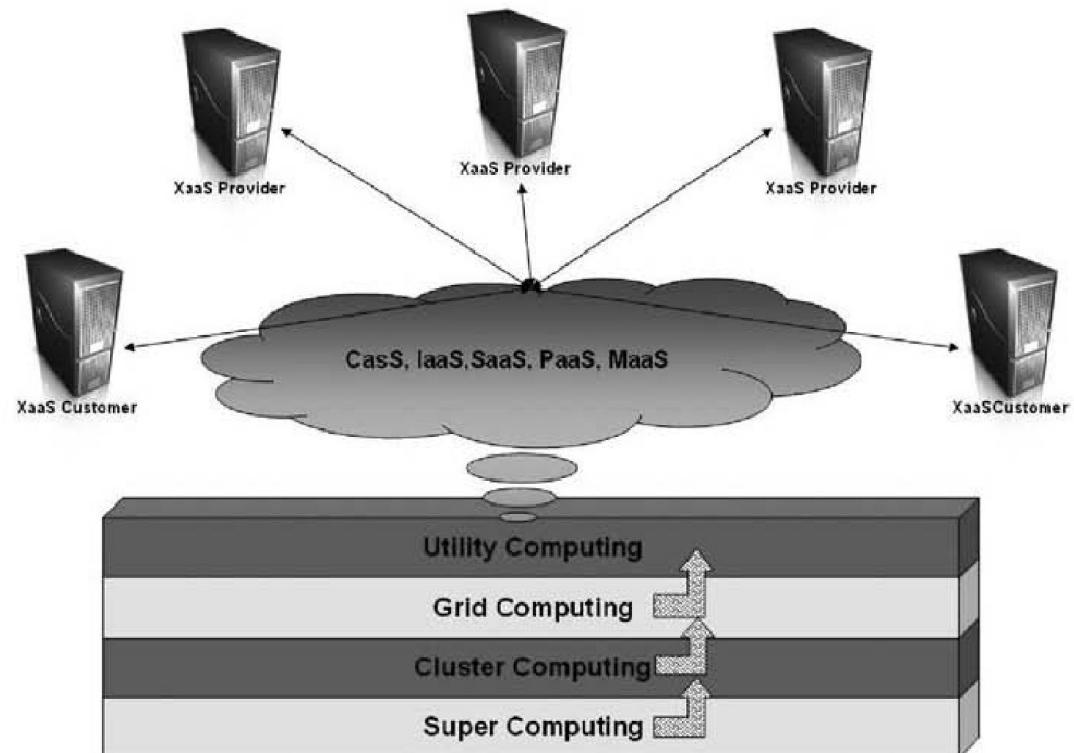
- Abilitatea de a testa local sau in sisteme de livrare CC utilizand software de testare si servicii care sunt gazduite la distanta.

Backup as a Service

- SaaS pentru backup online - EMC's Mozy
- Servicii de subsciere lunare
- Solutii:
 - ◆ Pentru consumator
 - ◆ Pentru companii.
 - ☞ Ofera backup offsite automatic pentru desktopurile clientilor, laptopuri, si servere Microsoft Windows
 - ☞ Administrare configurabila, lansare si centralizata via o consola administrative bazate pe web si multi-tenanta.

Orice ca serviciu (XaaS)

- Storage as a Service
- Database as a Service
- Communication as a Service
- Network as a Service
- Monitoring as a Service
- Testing as a Service
- HPC as a Service
- Human as a Service
- Process as a Service
- Information as a Service
- Identity as a Service
- Application as a Service
- Integration as a Service
- Governance as a Service
- Security as a Service
- Backup as a Service



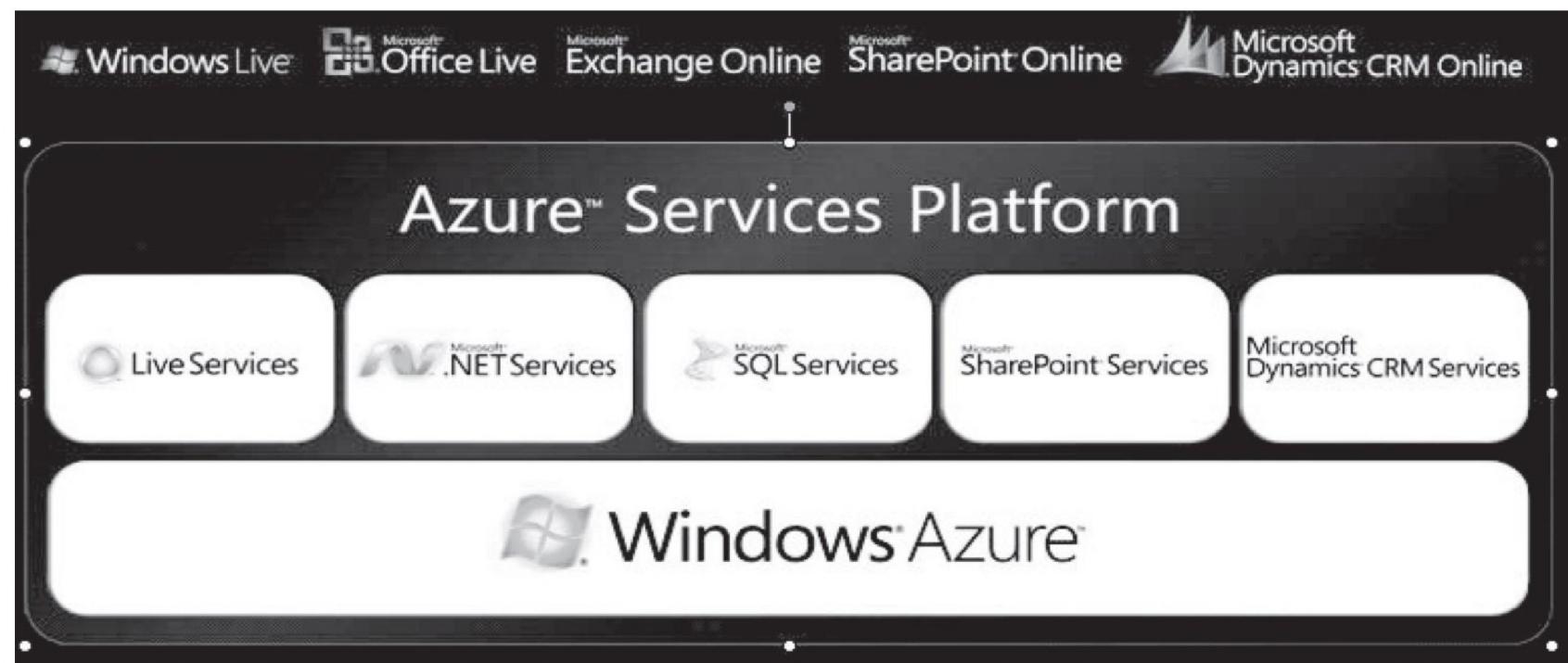
Serviciile Amazon

- Amazon a fost una dintre primele companii care au oferit servicii CC catre public
 - ◆ ***Elastic Compute Cloud (EC2)***
 - ◆ ***Simple Storage Service (S3)***
 - ◆ ***Simple Queue Service (SQS)***
 - ◆ ***SimpleDB***
 - ◆ ***CloudFront***

- Cloud-ul Amazon este recomandat daca:
 - ◆ Se doreste utilizarea de soft open-source de la un alt furnizor
 - ◆ Se dispune de un cod existent
 - ◆ Se doreste transferul aplicatiei web pe masina proprie mai incolo (aprox. *~zero lock-in*)
 - ◆ Portarea codului in alt limbaj
 - ◆ Se doreste control complet
 - ◆ Se doresc teste de stres/incarcare (e.g. 1000 de instante)

Windows Azure

- Ofera si servicii IaaS similar cu Amazon, dar ofera multe servicii la nivel PaaS.
- Multe aplicatii end-user Microsoft au fost modificate pentru a rula in cloud Obs.
- Platforma ofera si servicii SaaS (e.g. Office 365)



Referinte

- Binildas C. A. (2008) Service Oriented Java Business Integration Enterprise Service Bus integration solutions for Java developers
- Packt Publishing Ltd., Birmingham, UK
- <http://thor.info.uaic.ro/~adria/teach/courses/pcd/>
- http://cks.univnt.ro/uploads/cks_2013_articles/index.php?dir=4_IT_in_Social_Sciences%2F&download=cks_2013_it_003.pdf
- <http://banking.incloud.ro/wp-content/uploads/2013/02/GARTNER-Felix-enescu-v02.ppsx>
- <http://download.microsoft.com/download/B/5/F/B5F597E5-CBE7-420B-B8D8-5BC9EDC0C575/Introducing%20Windows%20Azure.ppt>
- <http://web.info.uvt.ro/~petcu/distrib.htm>

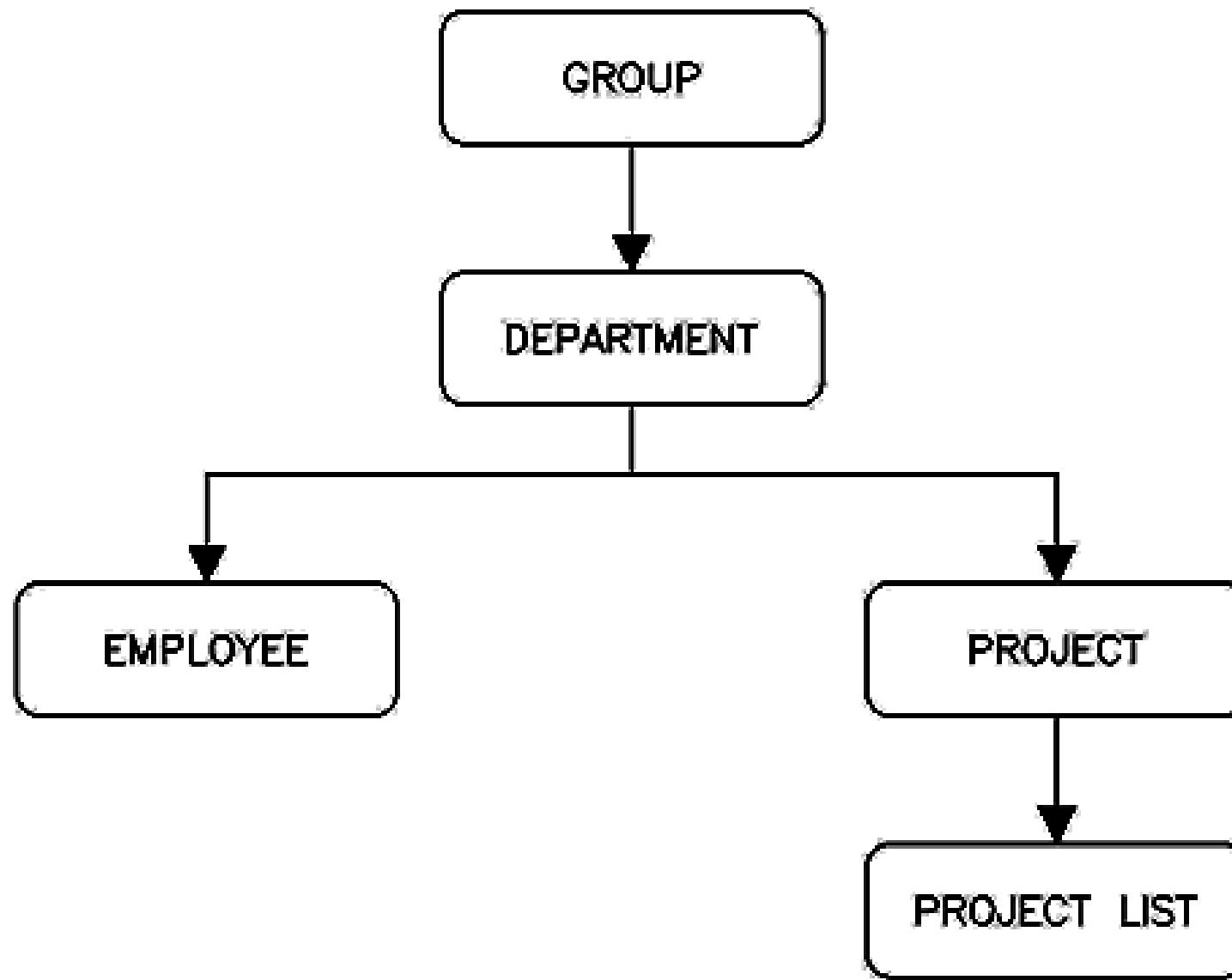
Cursul 10

Shared Information
systems

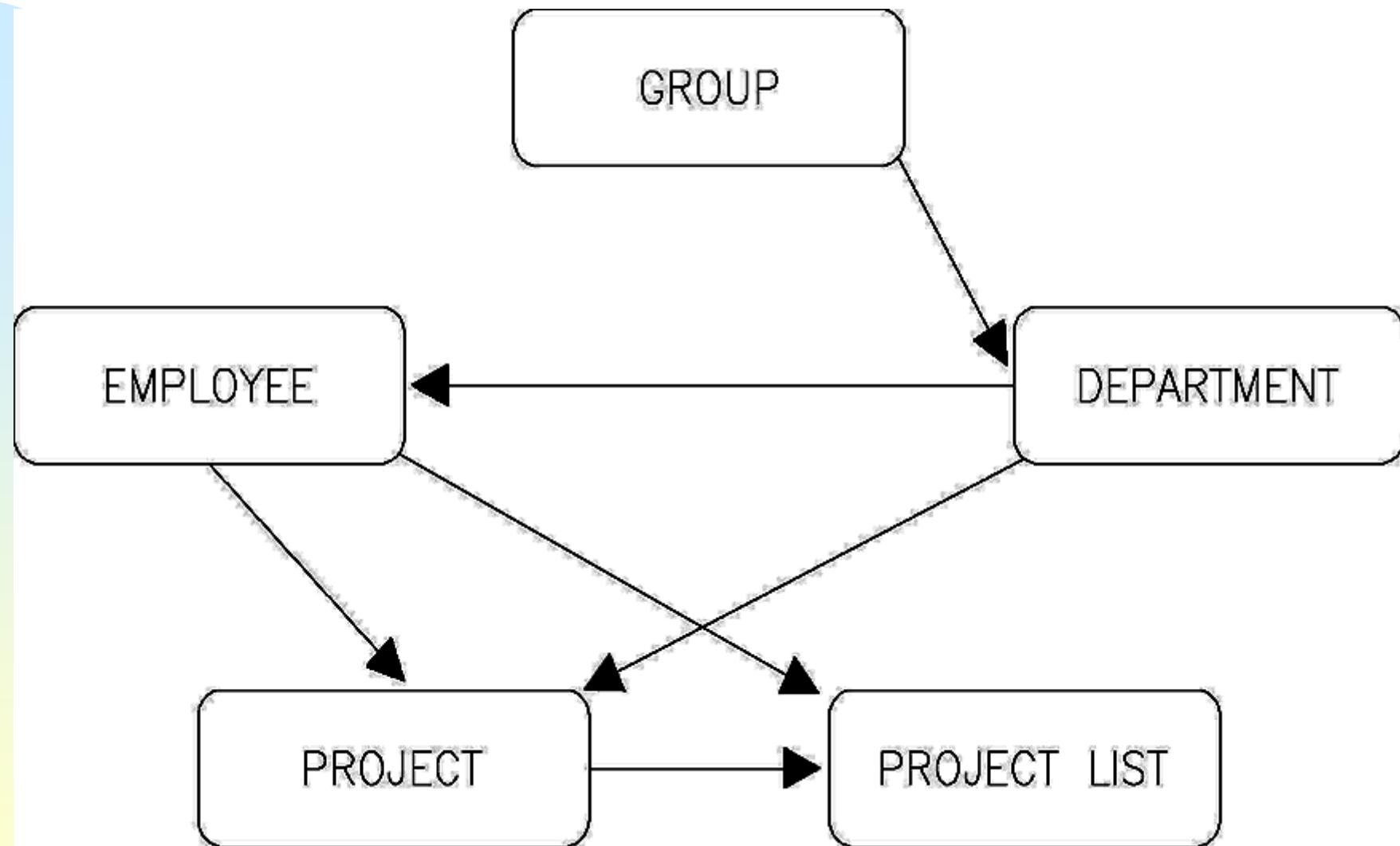
- Sistemele globale de informații pot fi clasificate în:
 - ◆ Baze de date distribuite
 - ◆ Baze de date multiple cu scheme globale
 - ◆ Baze se date federative
 - ◆ Sisteme omogene de baze de date multilimbaj
 - ◆ Sisteme interoperabile

- Acestea pot fi reduse pe baza asemănărilor la trei categorii:
 - ◆ Sisteme de baze de date distribuite
 - ◆ Sisteme de baze de date multiple
 - ◆ Sisteme de baze de date federative

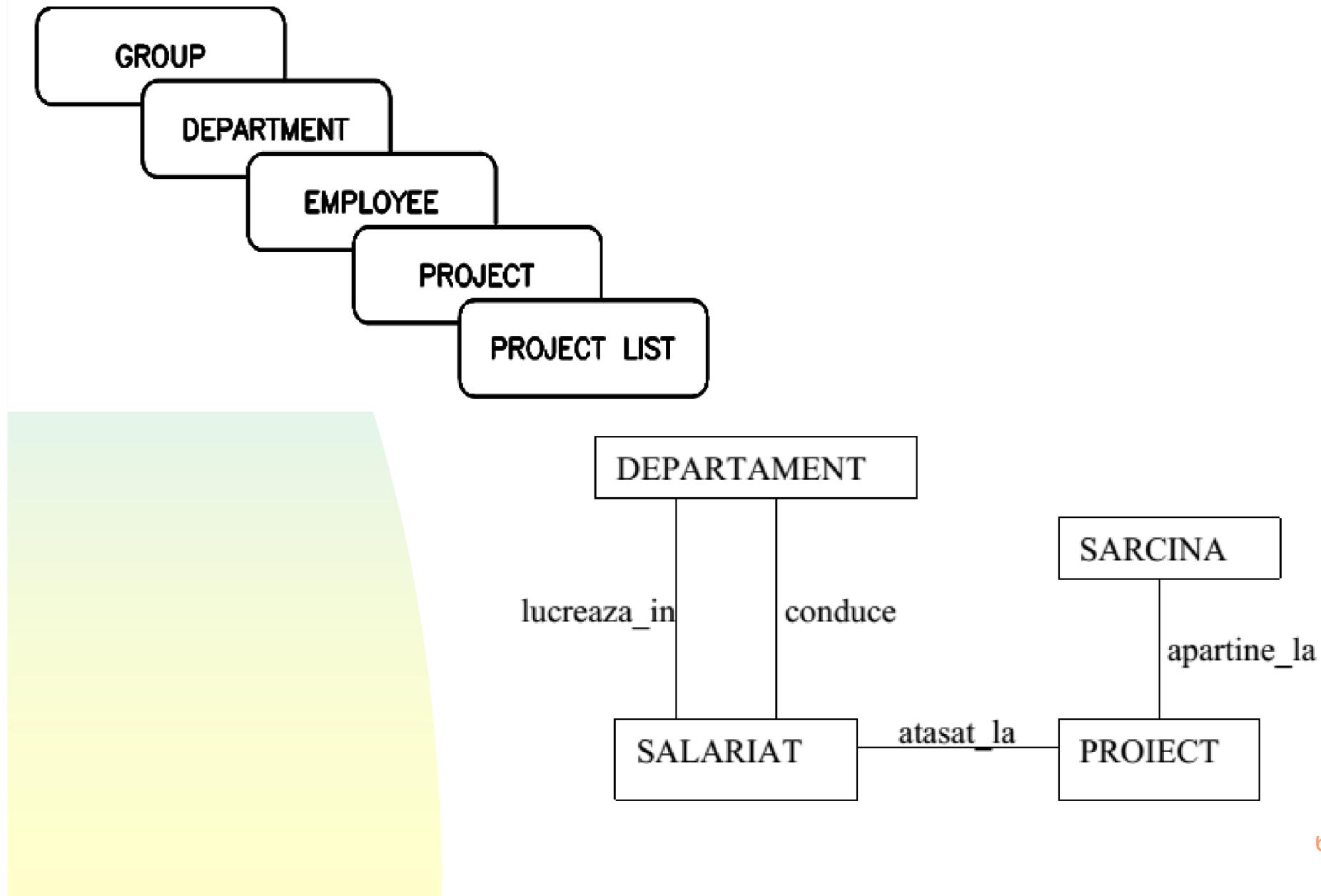
Hierarchical Model



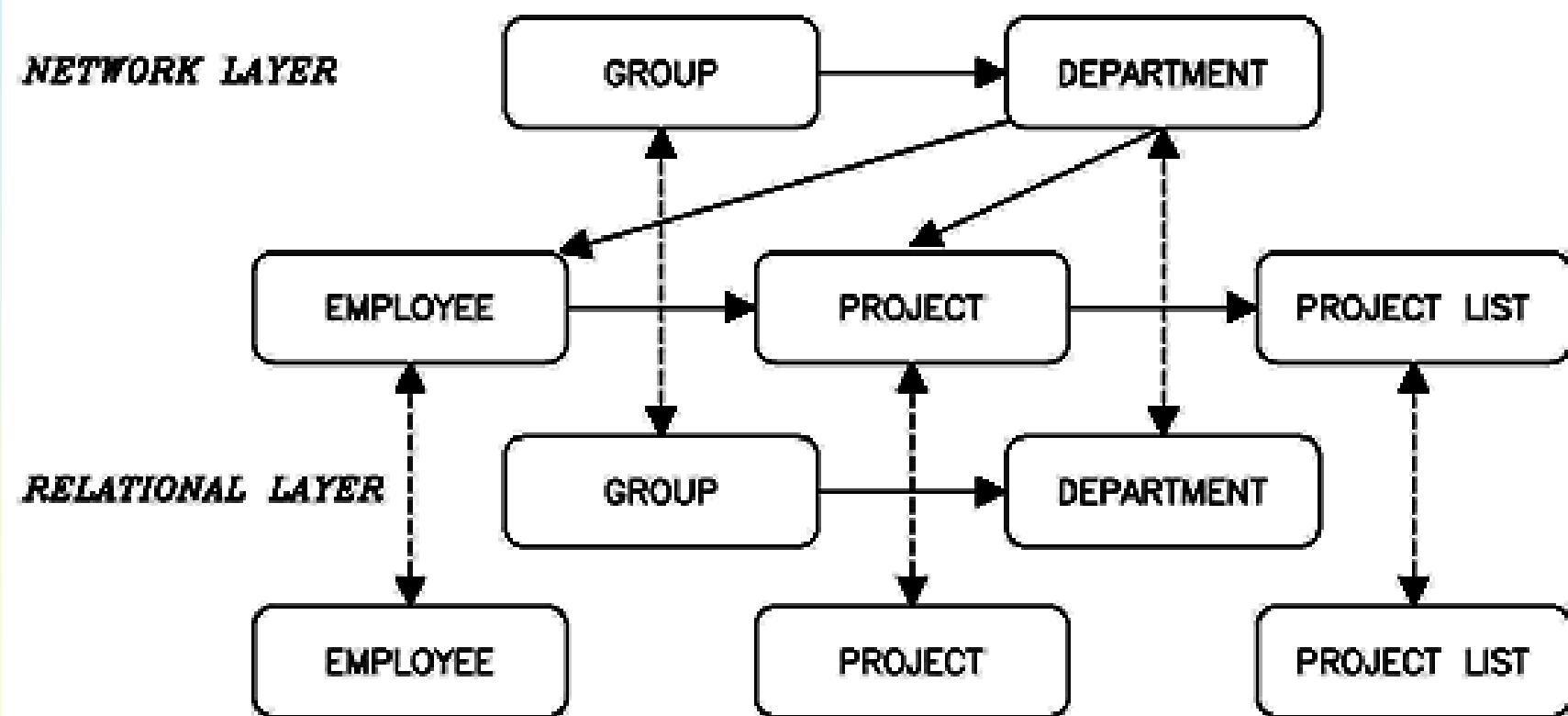
Network Data Model



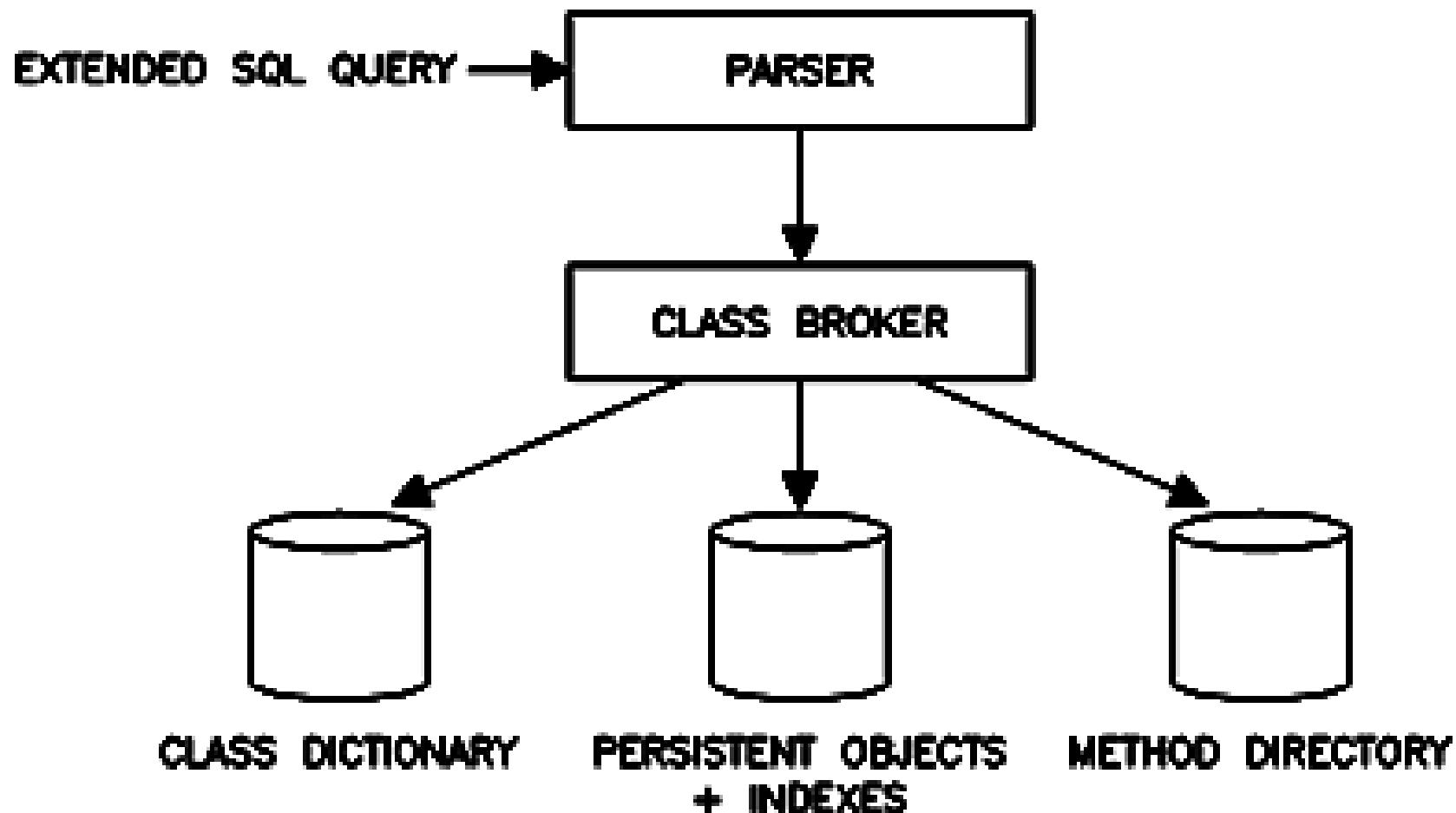
Relational Model



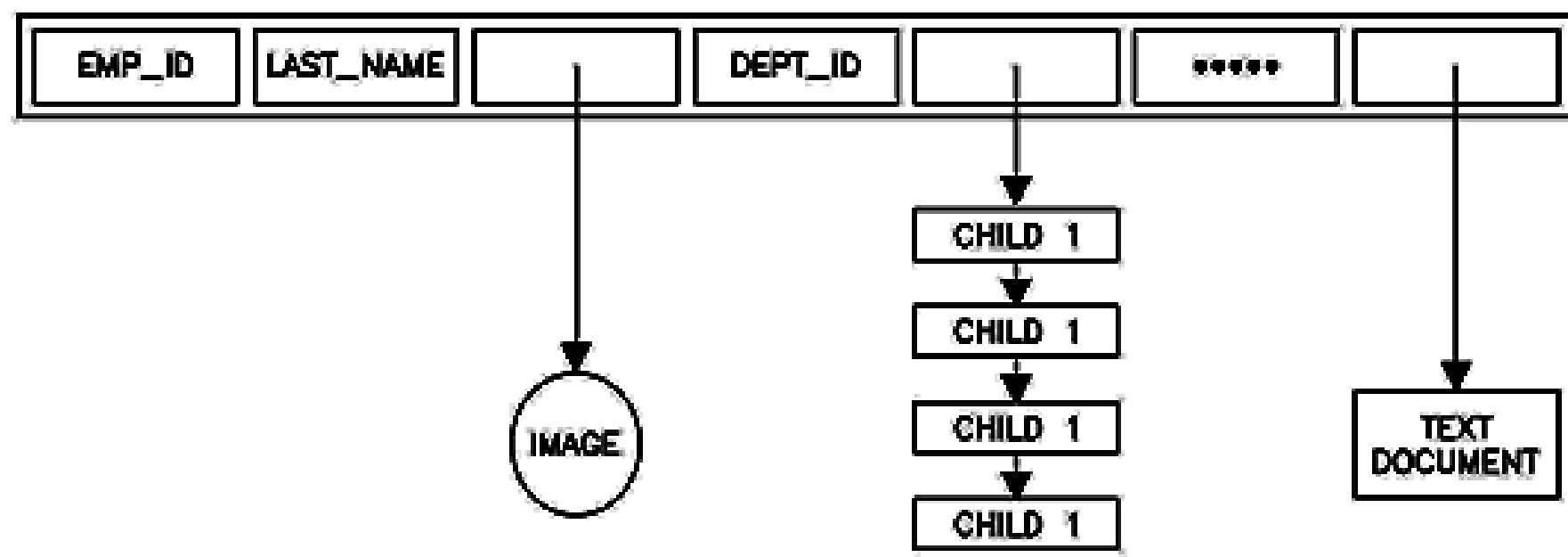
Hybrid Object/Relational Model prin compunerea in stiva a celor clasice

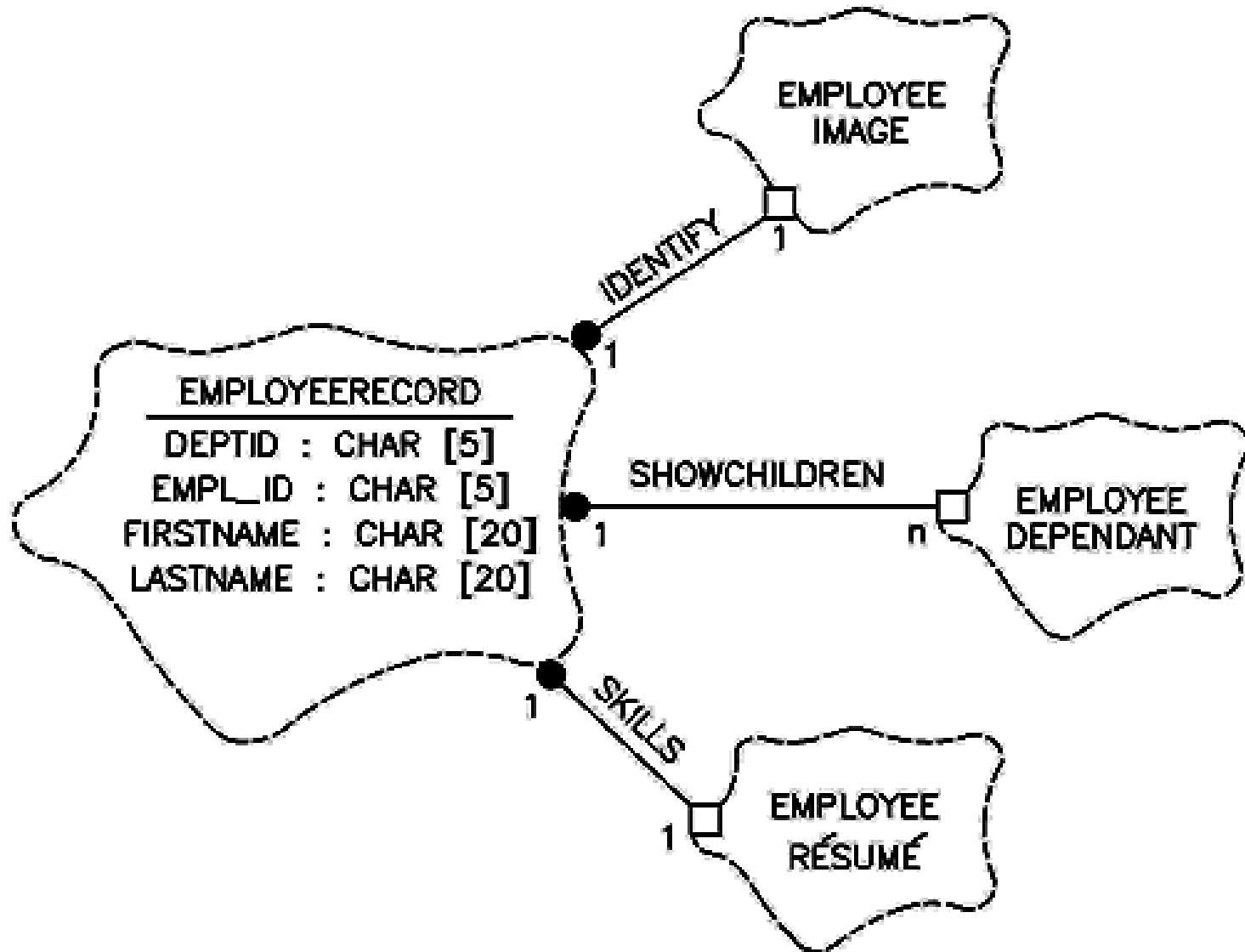


PURE OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS - OODBMS

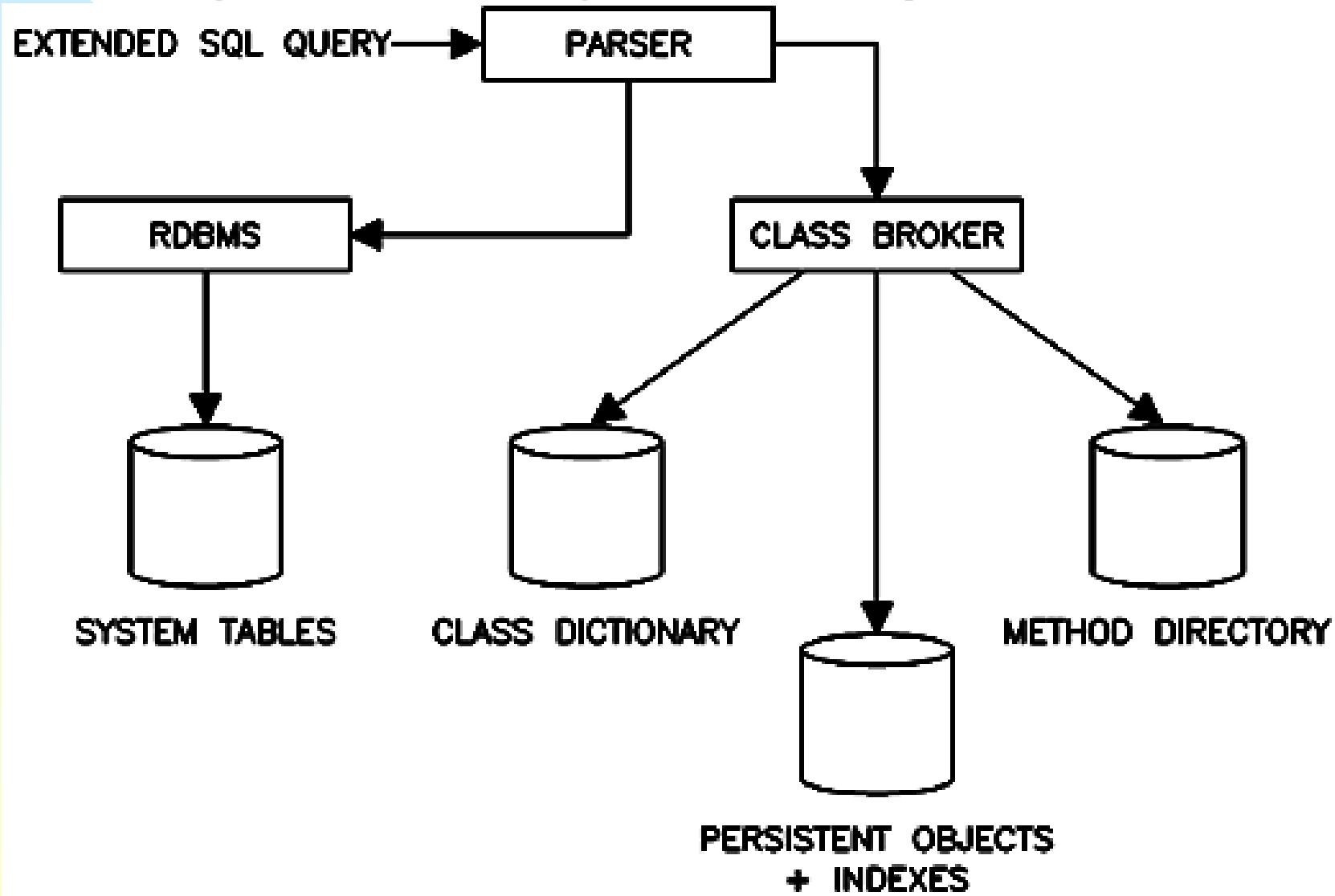


Complex employee record





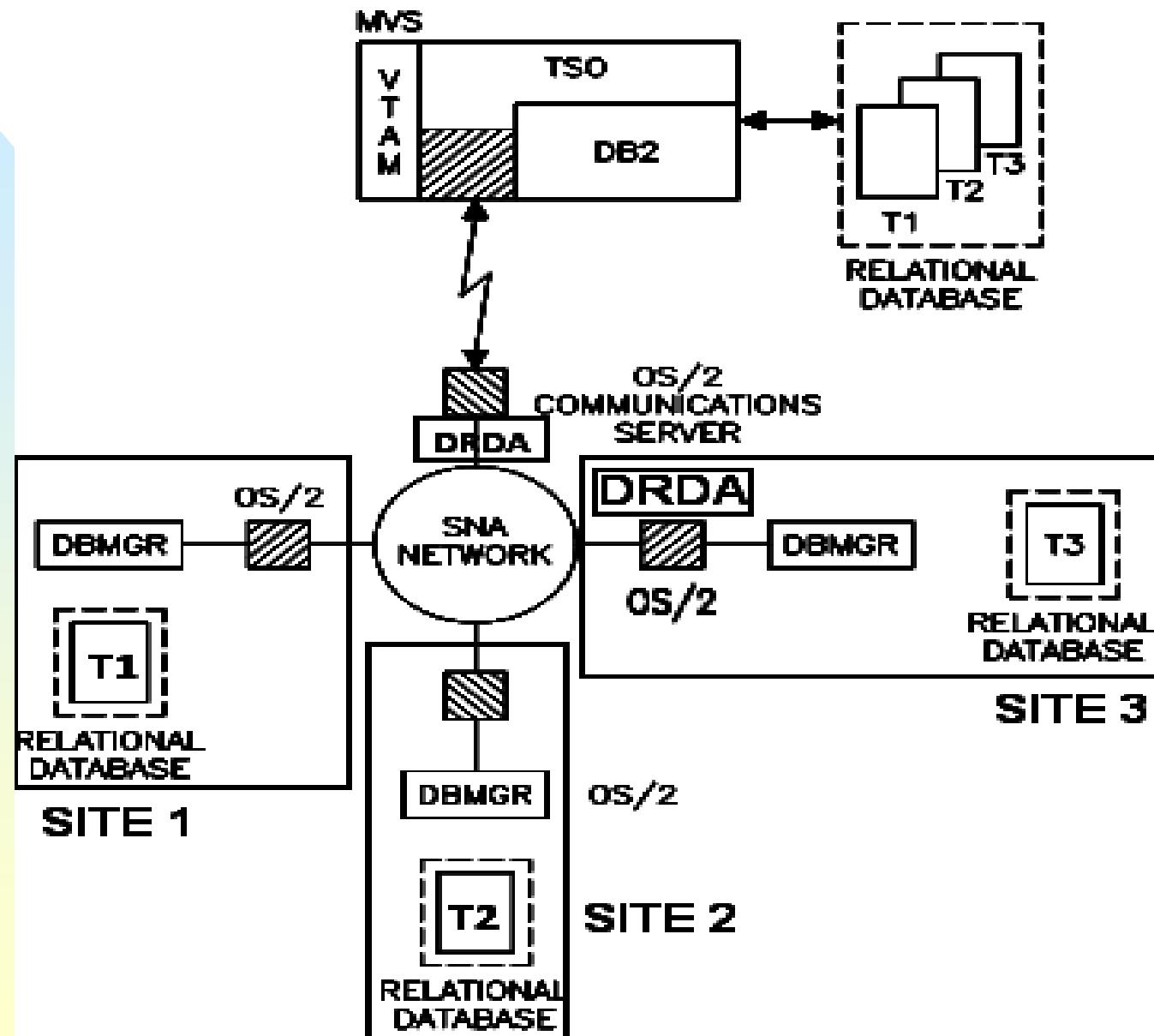
Hybrid Object/Relational Database Management Systems (HORDBMS)



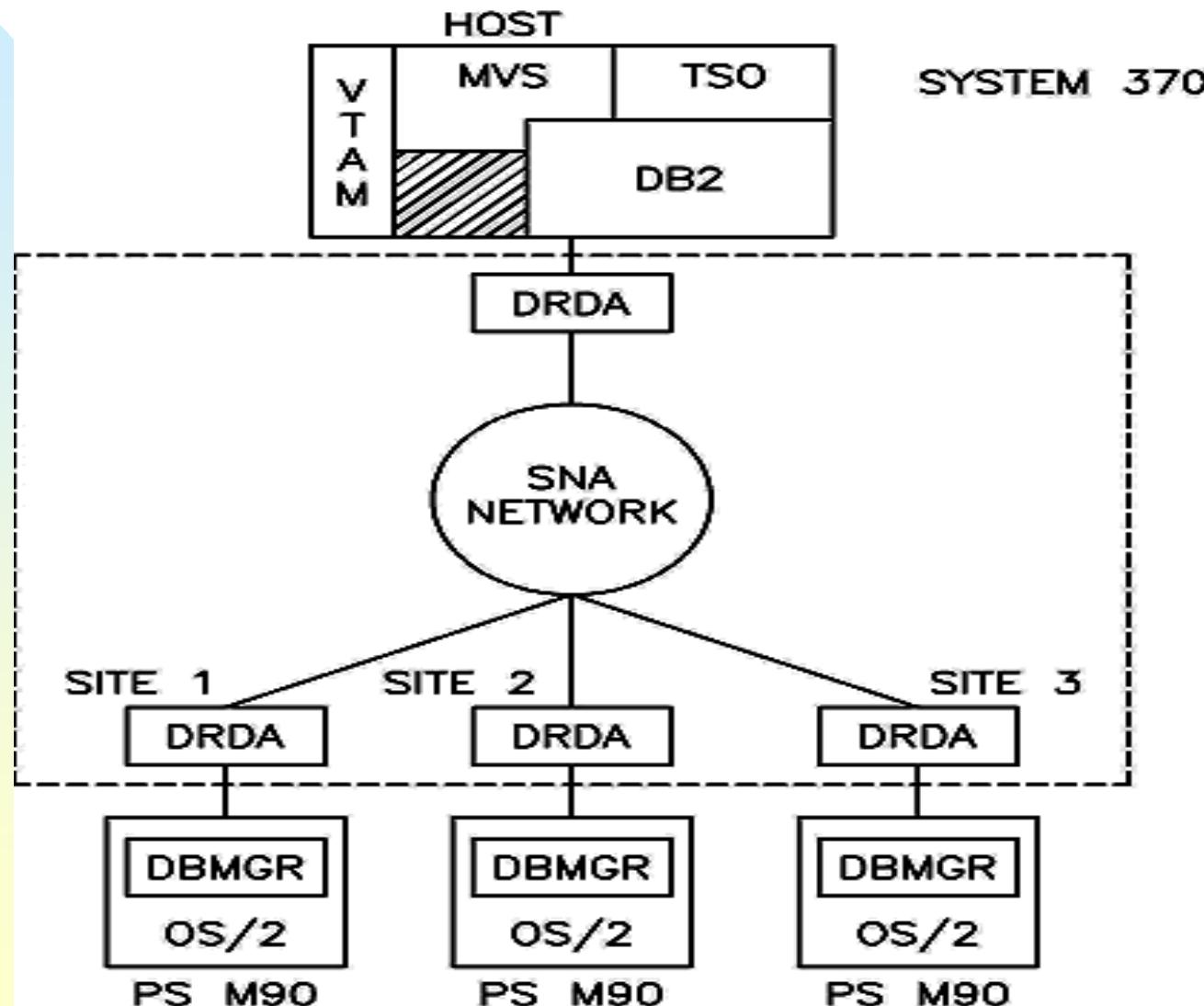
Sisteme de baze de date distribuite

In general terminologia refera pe cele omogene.

Aceasta se refera la faptul ca metodele de acces, strategiile de optimizare de gestiune a concurentei precum si modelele de date sunt similare indiferent de nodul integrat in sistem.



Arhitectura DRDA a IBM pentru situatia de BDD anterioara



Cum se utilizeaza asa ceva

EMP TABLE

EMP ID	LNAME	DEPT	SALARY
E1	SMYTHE	D1	50000
E2	BILLINGS	D1	40000
E3	WILLIAMS	D2	80000
E4	JONES	D3	90000

EMP TABLE FRAGMENT 1

EMP ID	LNAME	DEPT	SALARY
E1	SMYTHE	D1	50000
E2	BILLINGS	D1	40000

EMP TABLE FRAGMENT 2

EMP ID	LNAME	DEPT	SALARY
E3	WILLIAMS	D2	80000

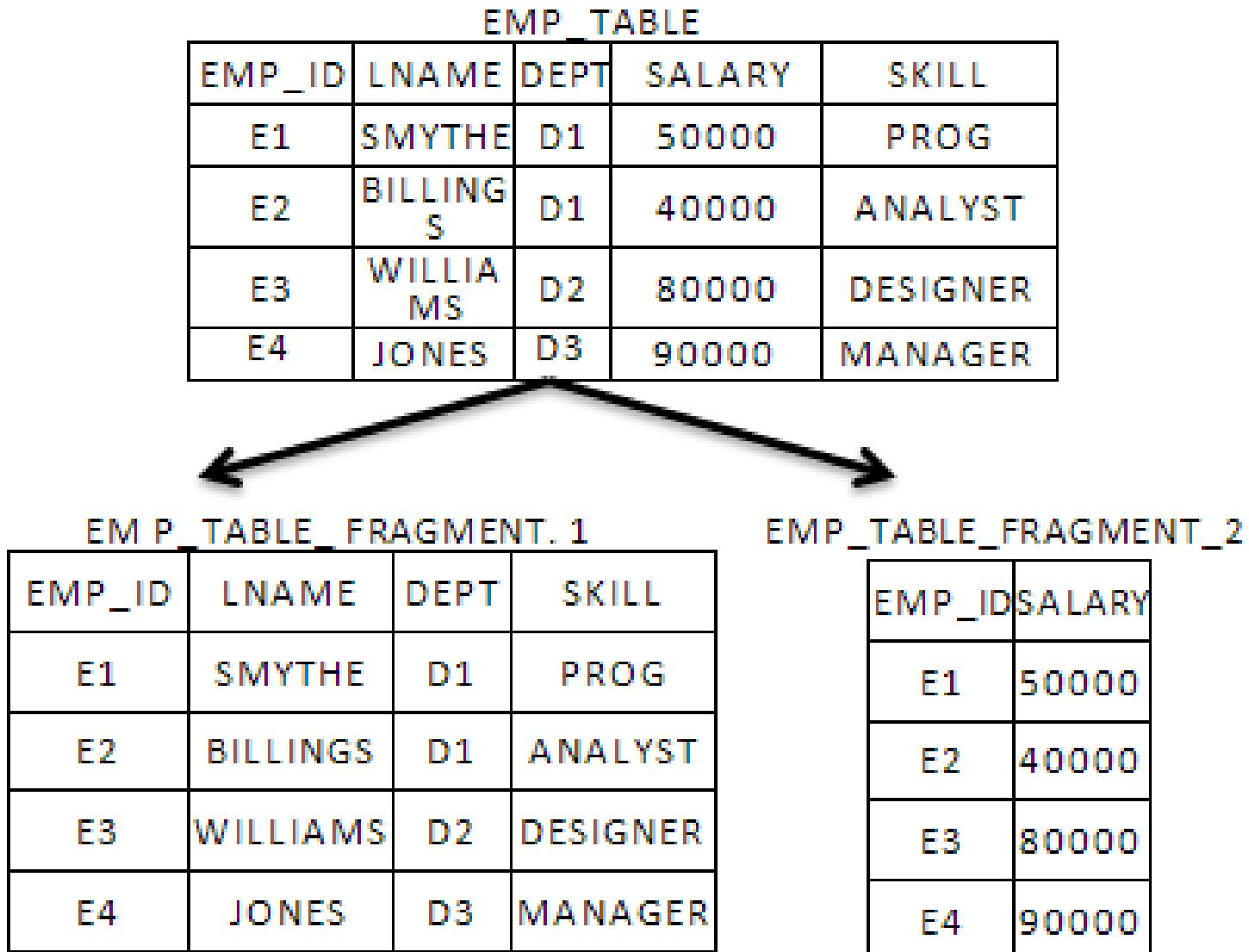
EMP TABLE FRAGMENT 3

EMP ID	LNAME	DEPT	SALARY
E4	JONES	D3	90000

Impartirea trebuie facuta explicit :

- EMP_TABLE_FRAGMENT_1 = SELECT (EMP_TABLE) WHERE DEPT = D1
- EMP_TABLE_FRAGMENT_2 = SELECT (EMP_TABLE) WHERE DEPT = D2
- EMP_TABLE_FRAGMENT_3 = SELECT (EMP_TABLE) WHERE DEPT = D3

Distributie verticala



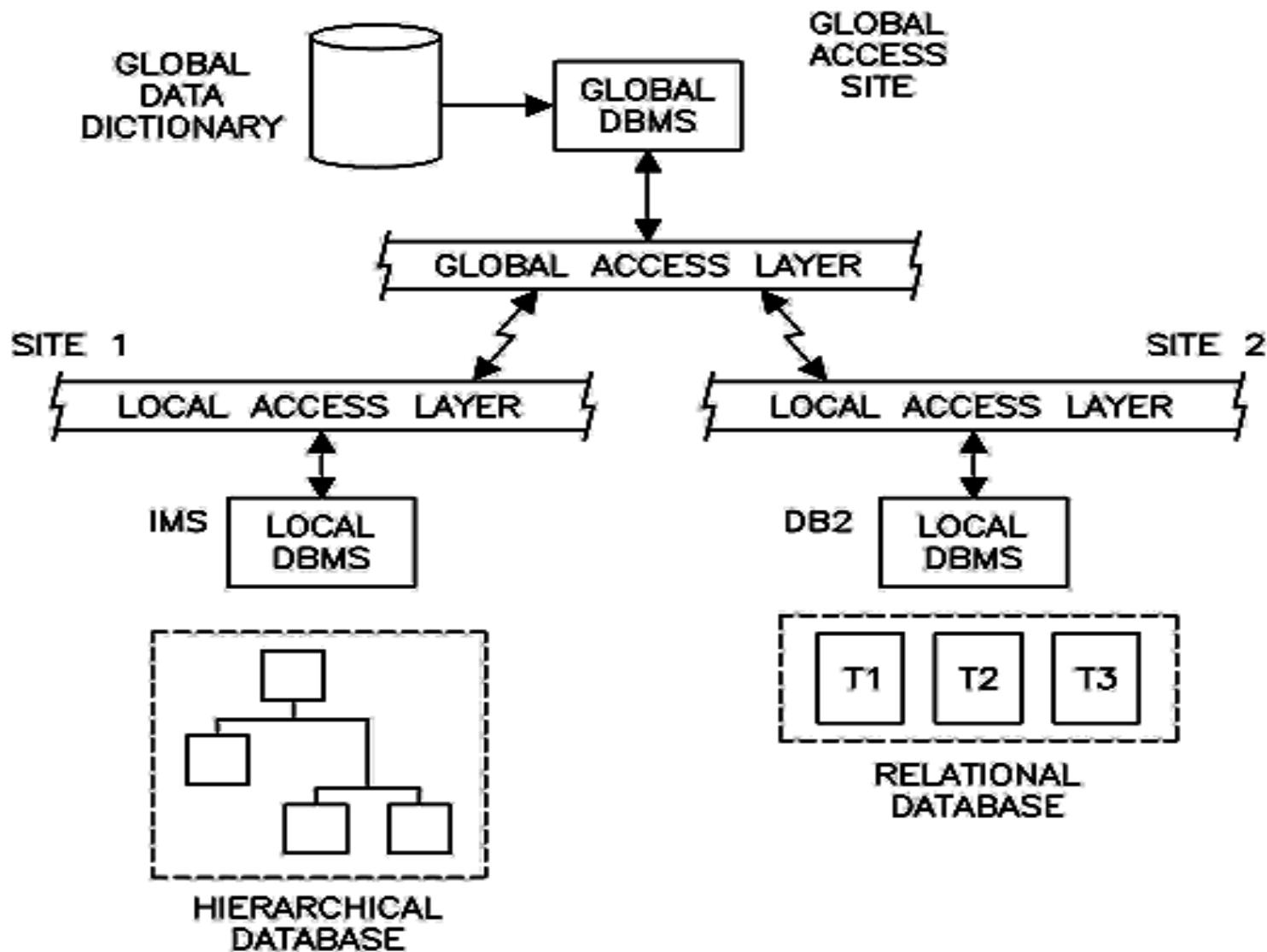
UN exemplu de interogare

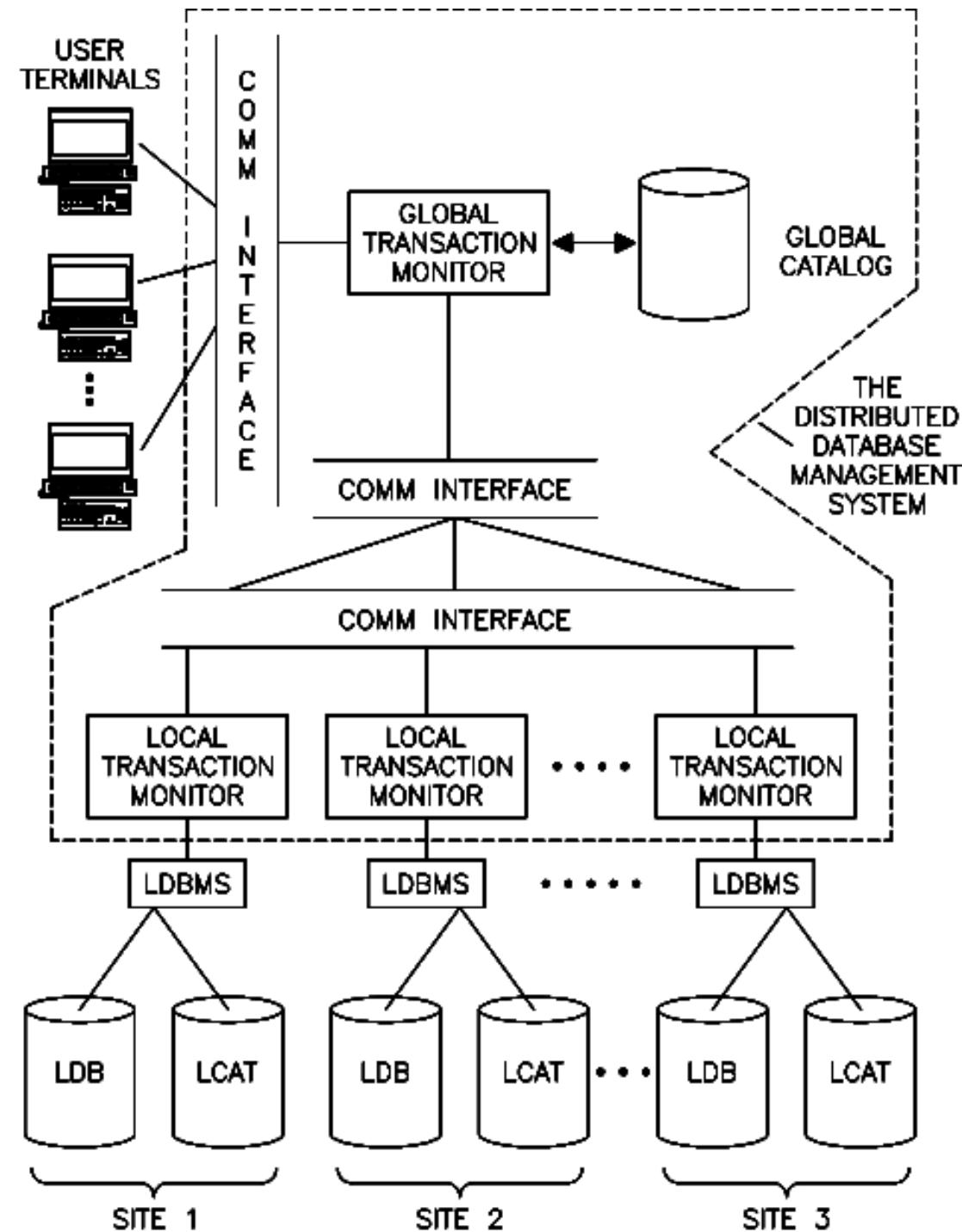
- `SELECT EMP_TABLE_FRAGMENT_1.EMP_ID,
EMP_TABLE_FRAGMENT_1.SKILL,
EMP_TABLE_FRAGMENT_2.SALARY, FROM
EMP_TABLE_FRAGMENT_1, EMP_TABLE
FRAGMENT_2 WHERE
EMP_TABLE_FRAGMENT_1.EMP_ID =
EMP_TABLE_FRAGMENT_2.EMP_ID AND
EMP_TABLE_FRAGMENT_2.SALARY > 80000`
- IN cazul nostru va rezulta:

EMP_ID	SKILL	SALARY
E4	MANAGER	90000

Sistemele de baze de date multiple

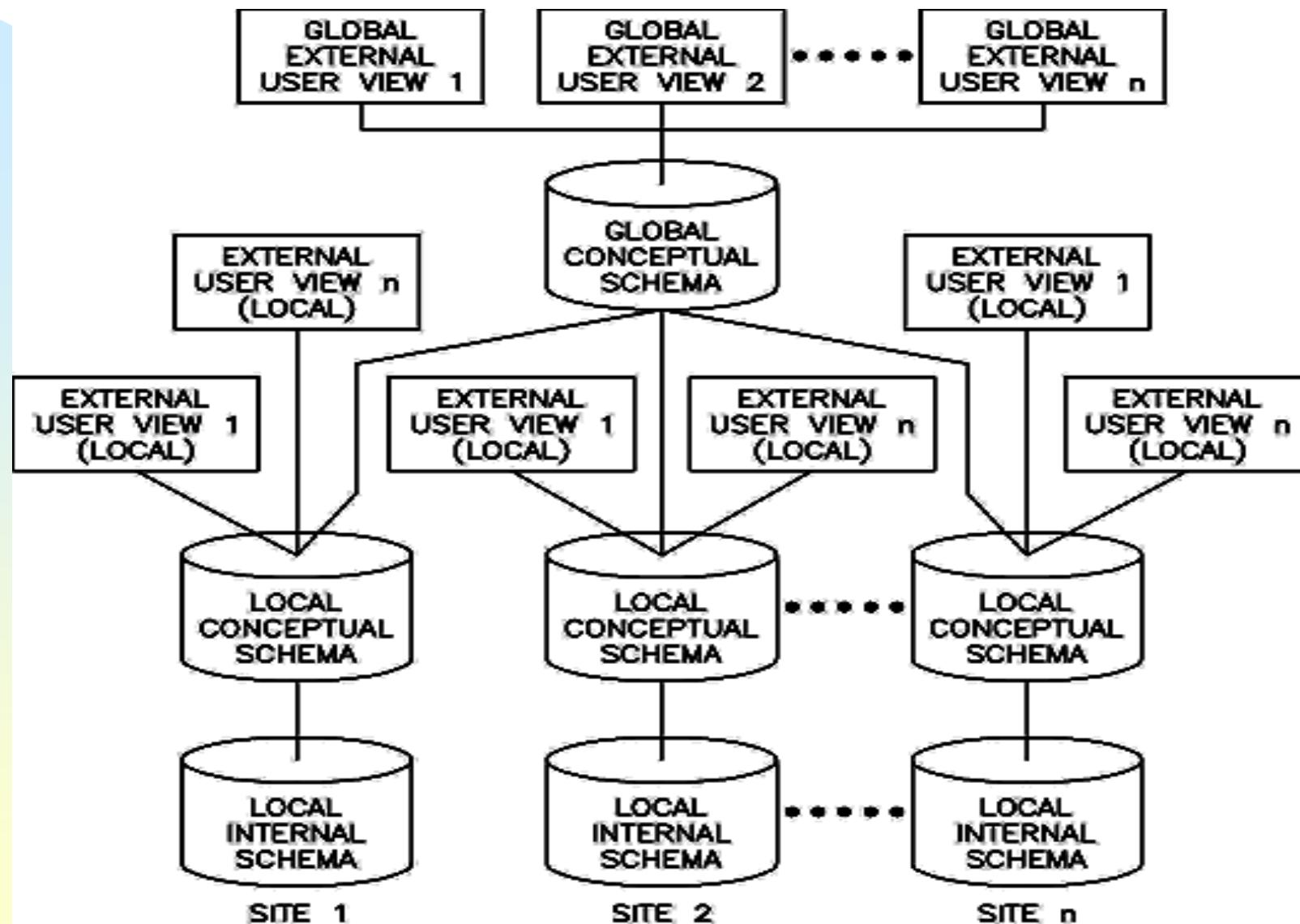
- Se referă la sistemele heterogene de baze de date.
- Acest tip de DBMS global este caracterizat de modele diferite de date.
- Spre deosebire de anterioarele (cele omogene) modelele de date pot fi de orice tip:
 - ◆ relational,
 - ◆ ierarhic
 - ◆ retea.





- O data ce interogarile distribuite sunt corecte din punct de vedere sintactic si grammatical ele sunt descompuse in bucati si trimise spre executie la nodurile locale.
- Daca nu apar erori atunci rezultatul executiei este trimis inapoi catre monitorul global.

Abordarea cu schema globala

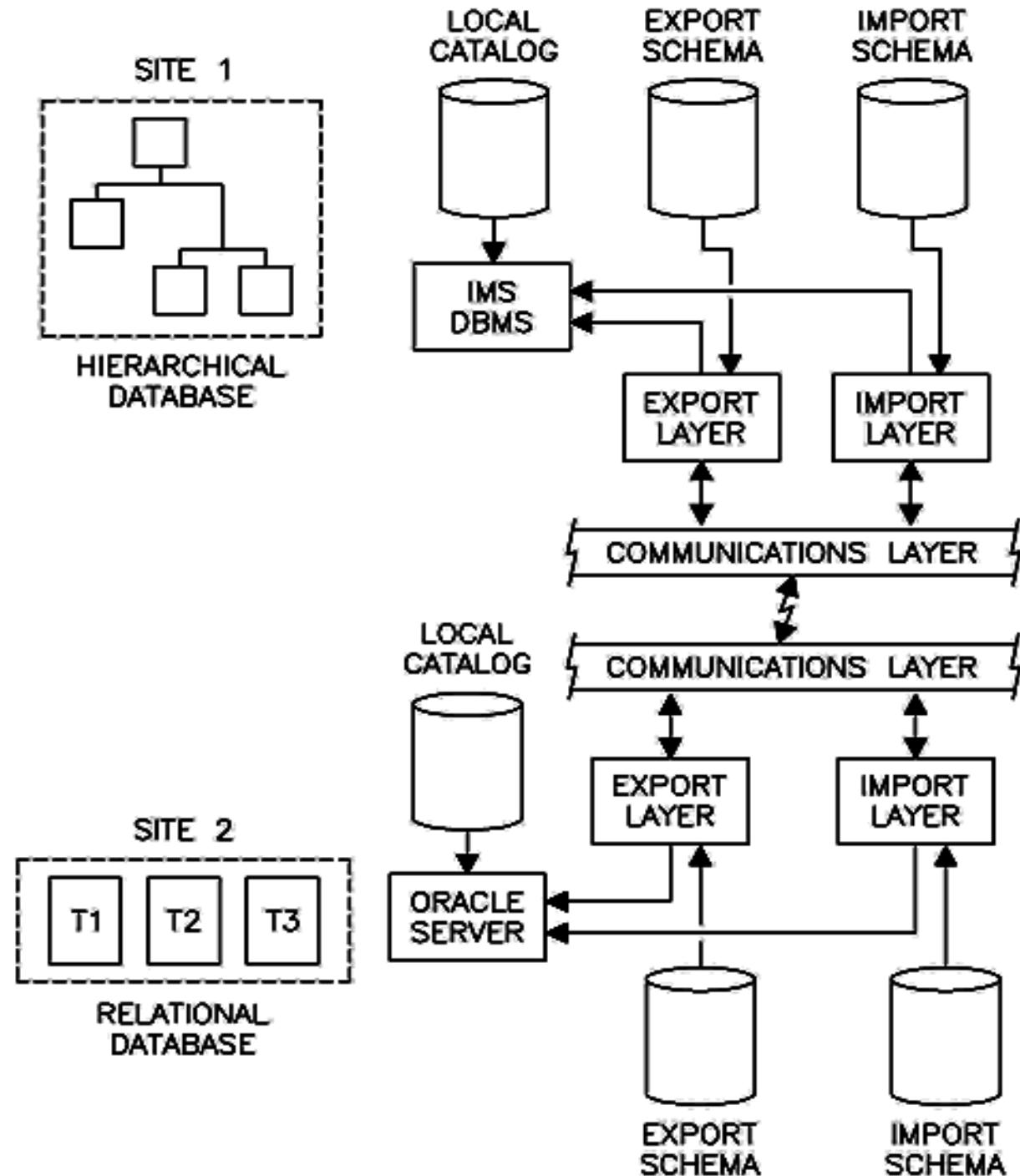


Abordarea cu schema multipla

- IN acest caz nu se folosesc scheme centralizate.
- Limbajele de interogare contin primitive care realizeaza automat translarile necesare intre modelele bazelor de date diferite care sunt in sistem.

Sisteme de baze de date distribuite federative

- Acestea sunt cazuri speciale ale sistemelor cu baze de date multiple.
- Ele sunt complet autonome.
- Nu se bazeaza pe dictionare globale pentru tratarea interogarilor distribuite.

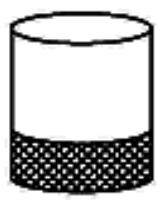


- Rezulta ca interogarile distribuite generate de fiecare nod sunt definite conform cu informatia prezenta in schema de import local.
- Membrii federatiei trebuie insa sa se puna de accord cu privire la protocoalele de comunicatii si metodele de distribuitie (routing) a interogarilor si datelor

LOCAL CATALOGS



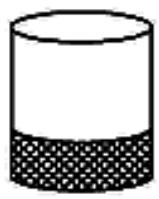
SYSTEM
TABLE
CATALOG



SYSTEM
COLUMN
CATALOG

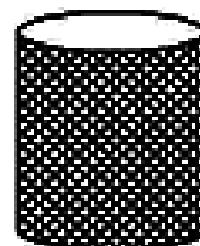


SYSTEM
VIEW
CATALOG

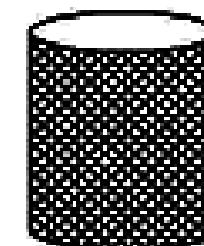


SYSTEM
INDEX
CATALOG

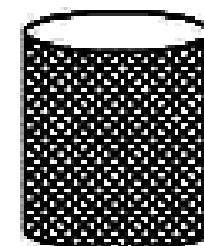
LOCAL EXPORT SCHEMA



EXPORT
COLUMN
CATALOG



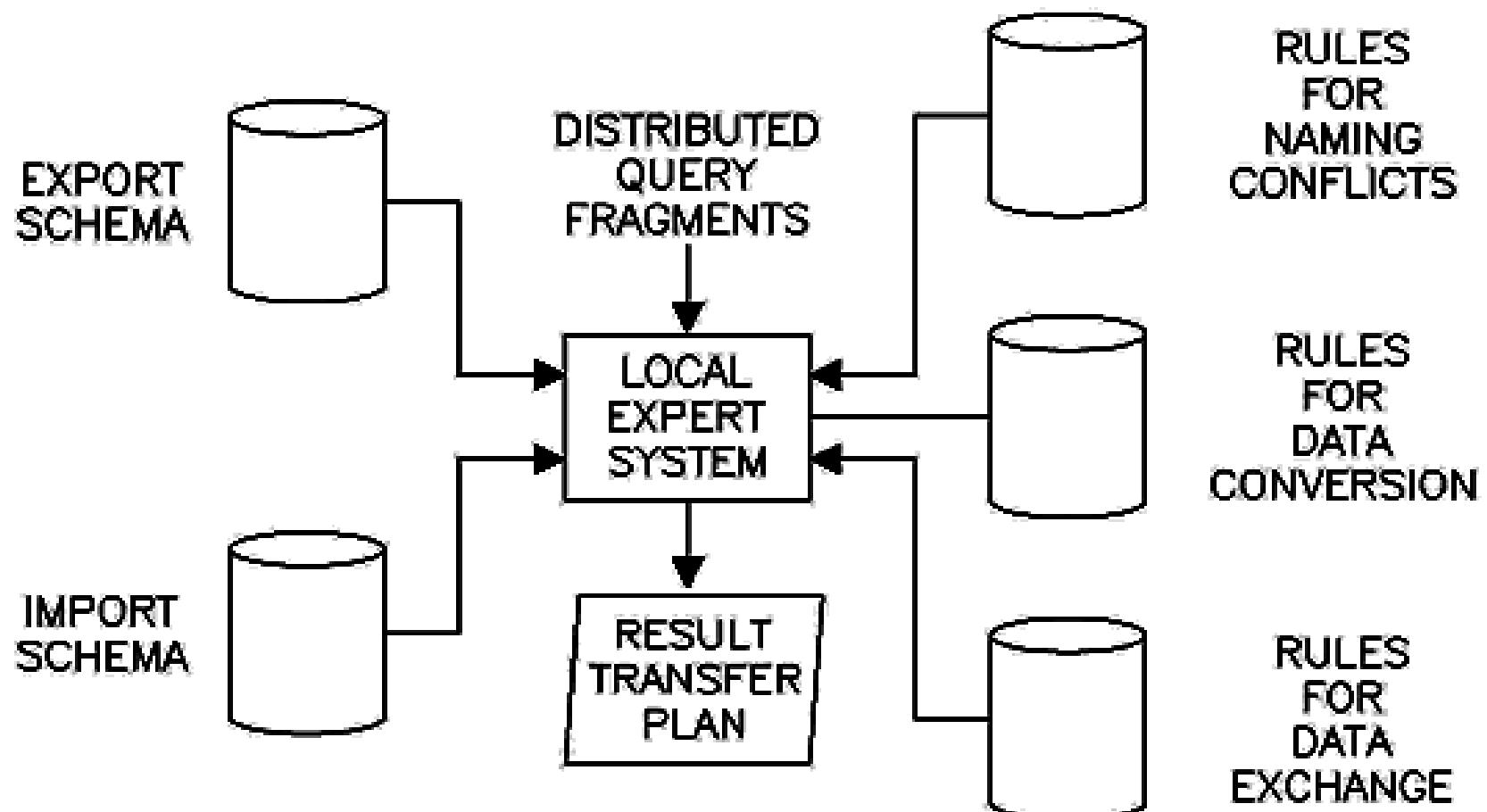
EXPORT
VIEW
CATALOG



EXPORT
INDEX
CATALOG

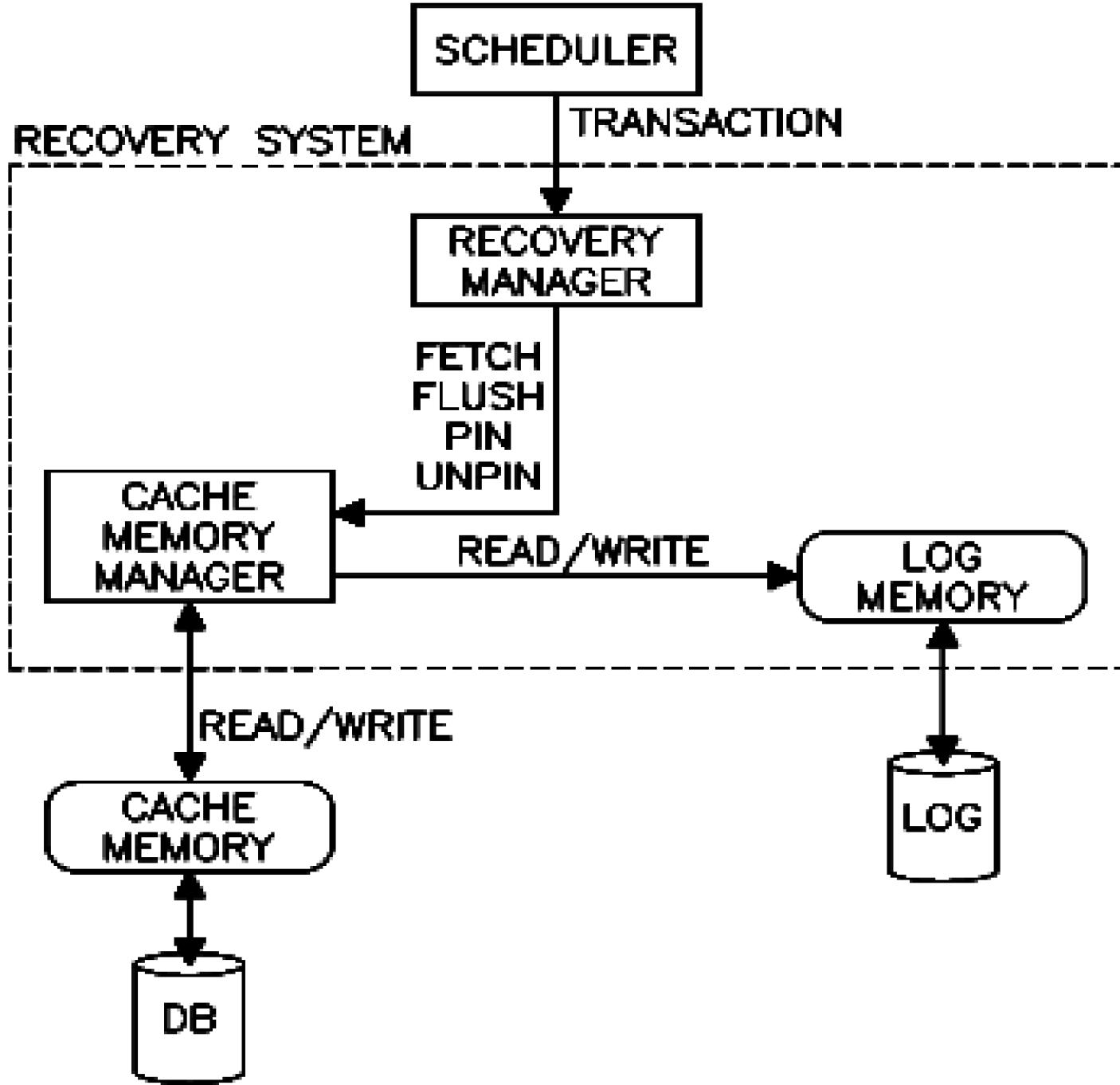
Sistemele expert

- Din ceea ce s-a discutat result ac transformarile intre diverse forme ale interogarilor sunt deosebit de complexe.
- În consecinta este indicate utilizarea de sisteme expert pentru a putea usura dezvoltarea acestor transformari.



Strategii de refacere locale sau distribuite

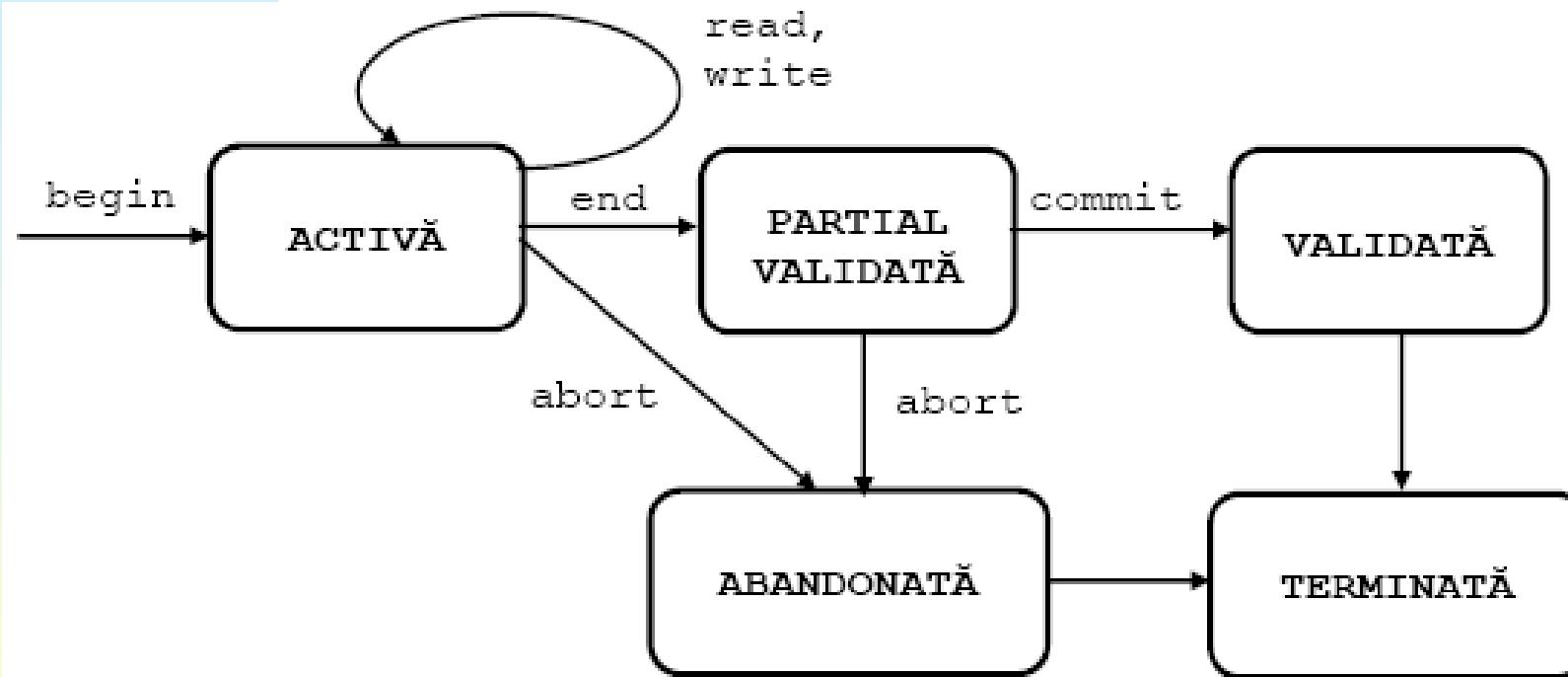
- Acestea sunt necesare pentru a trata urmatoarele tipuri de erori/caderi/abandonare
 - ◆ Caderi la nivelul tranzactiei
 - ◆ Caderi la nivelul sistemului
 - ◆ Caderi la nivelul suportului media

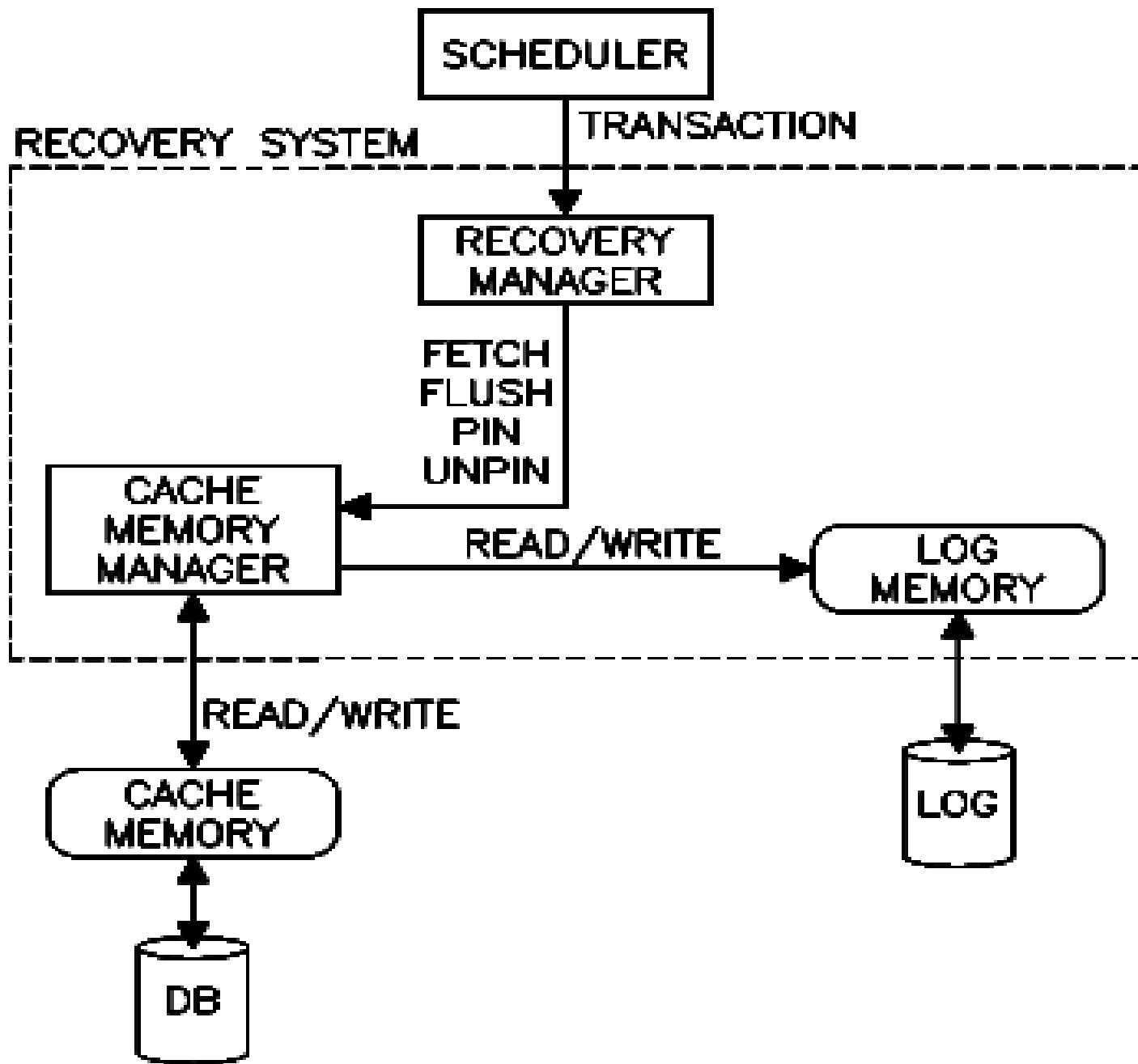


Operațiile specific tranzactiilor sunt

Operație	Semnificație
Begin	Indică începutul unei tranzacții
End	Indică sfîrșitul tranzacției
Abort	Sfîrșit nereușit de tranzacție; modificările dispar
Read	Citește valoarea unei variabile persistente
Write	Scrie valoarea unei variabile persistente

Diagrama de stare a unei tranzacții



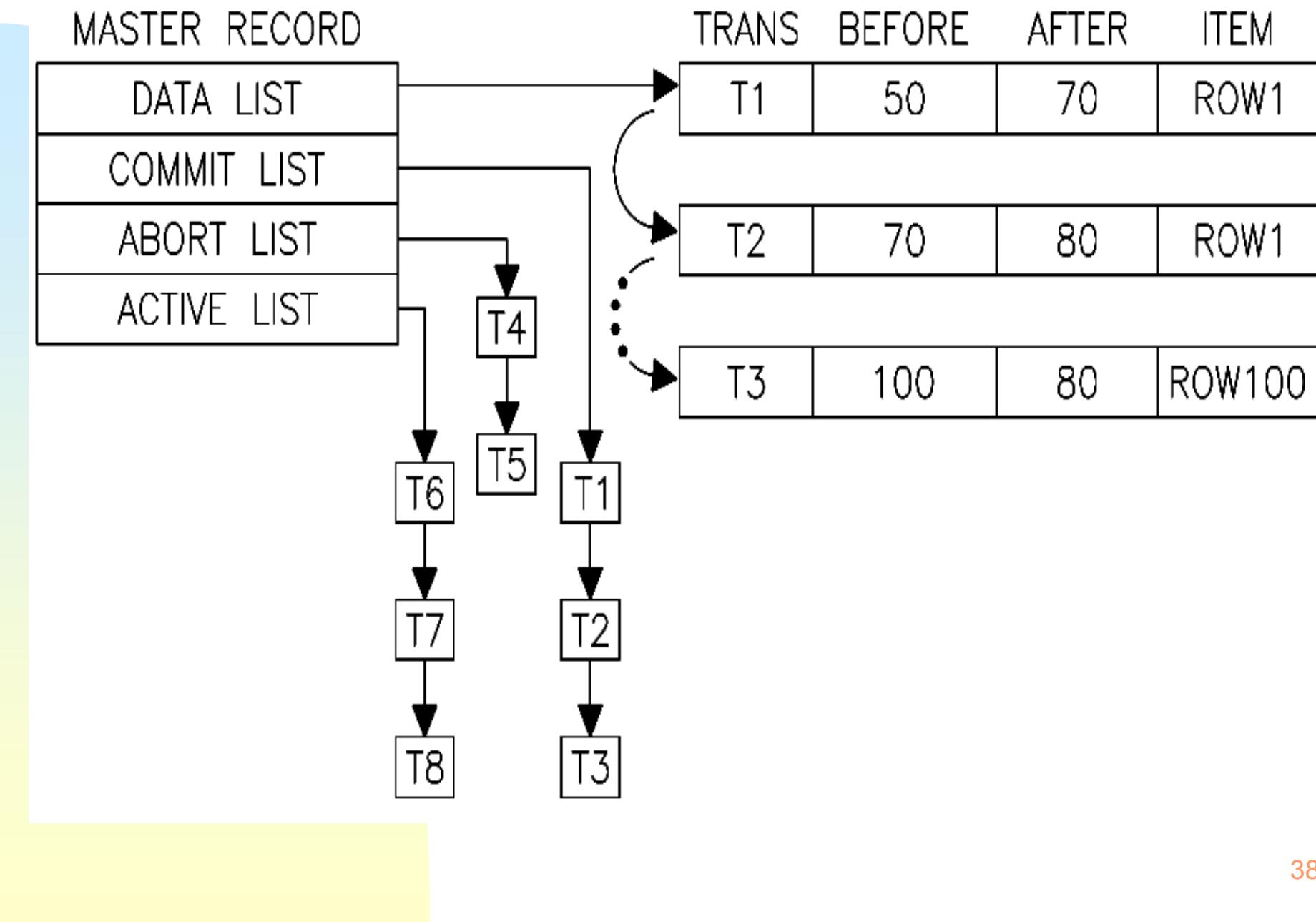


Proprietatile tranzactiei ACID

Atomicitate	nimic nu poate întrerupe o tranzacție în timpul executiei ei
Consistență	Tranzacțiile păstrează structurile de date corecte
Independență	Programe executate în paralel nu interferă
Durabilitate	Rezultatul unei tranzacții este permanent

Jurnalizare unica

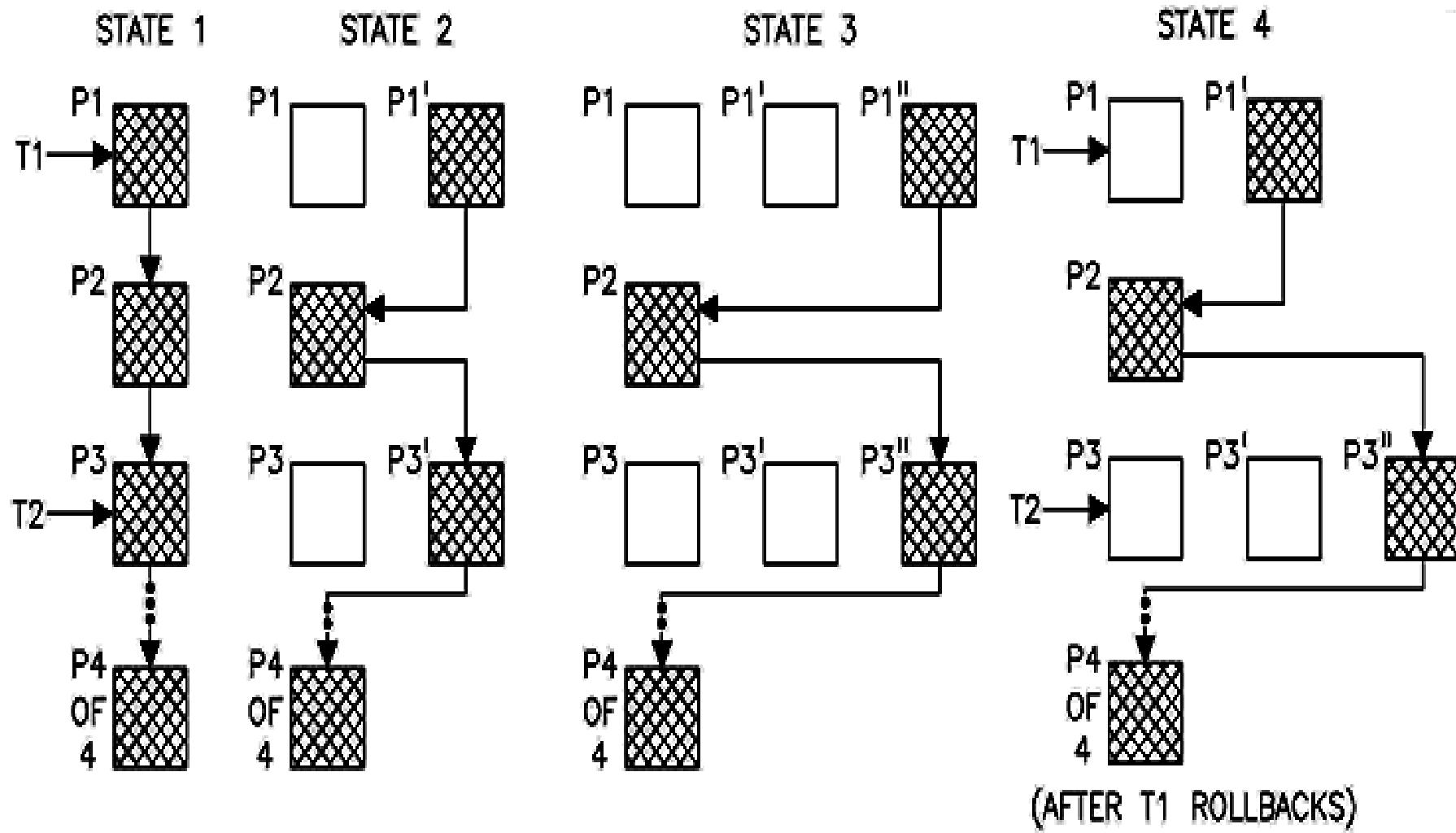
- TErmenul de “in-place logging” este folosit pentru a referi sistemele la care modificarile jurnalului (update) sunt scrise direct intr-o singura copie a acestuia.
- Cand sistemul ramane fara spatiu (se depaseste dimensiunea maxima stabilita a jurnalului) sistemul sterge vechile inregistrari si o ia de la capat.



Jurnalizare externă

- Termenul “Out-of-Place Logging” se referă la arhitecturi și strategii care mențin copii externe ale bazei de date pe post de jurnal de execuție.
- Dacă sistemul esuează atunci imaginea anterioară este luată din copie unde se află salvată ultima stare stabila dinainte de a începe operatiunea esuată

Shadow Logging



Jurnalizare diferențială

ROW DATA	
1	10
2	15
3	20
4	10
5	15
6	20

READ ONLY
STABLE
DB FILE

DIFFERENTIAL FILES
FOR TRANSACTION 1 & 2
INSERTS

1	10
2	20
3	25
4	10
5	15
6	20

T1

1	10
2	20
3	25
4	10
5	15
6	20
*	
**	
7	10

T2

NOTE:

- * ROWS CHANGED BY CURRENT TRANSACTION
- ** INSERTED RECORD
- 1) UPDATES ARE PERFORMED BY DELETING A ROW AND THEN INSERTING THE UPDATED ROW

DIFFERENTIAL FILES
FOR TRANSACTION 1 & 2
DELETES

1	
4	
5	
6	

T1

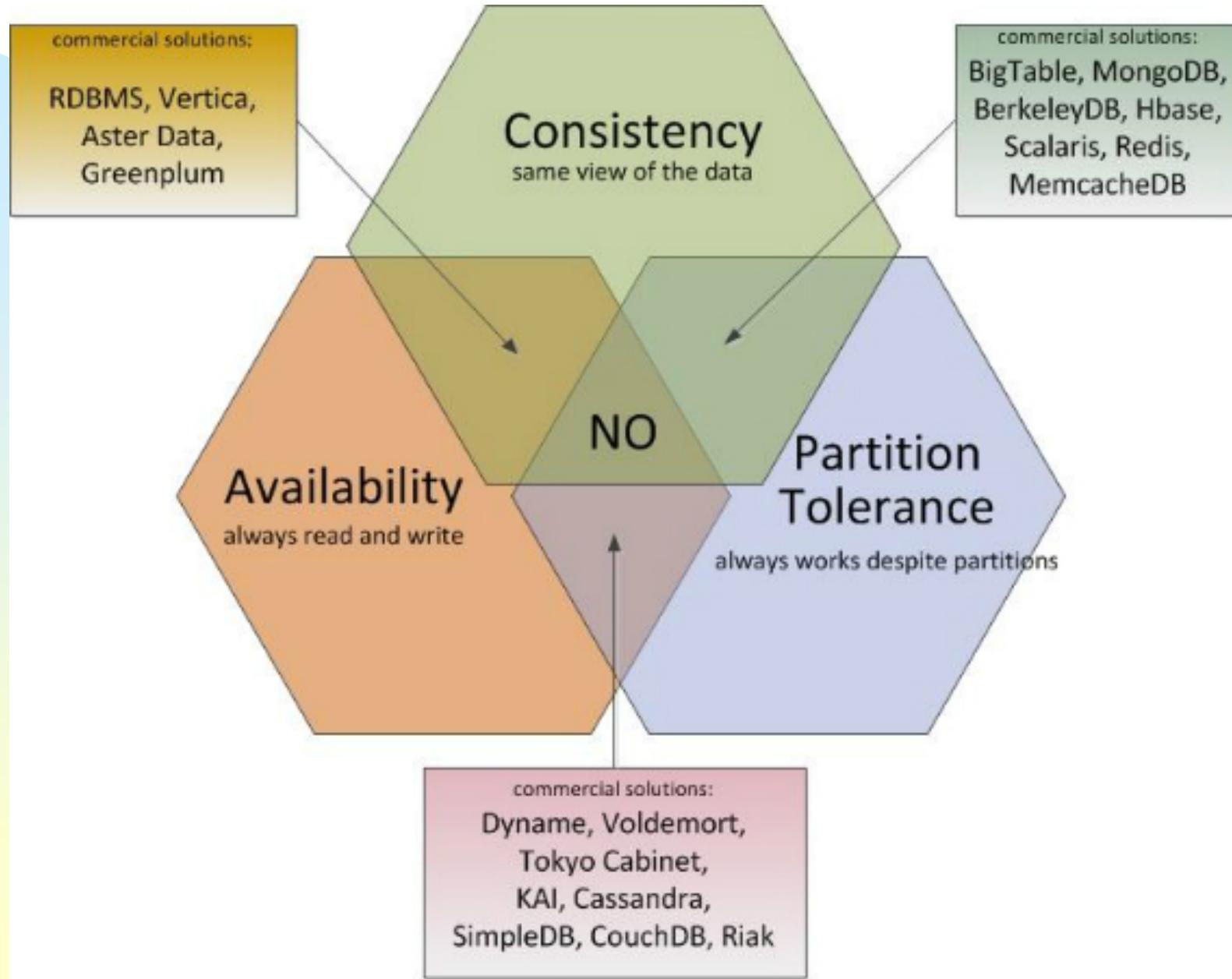
1	10
4	10
5	15

T2

Proprietati DDB

- Sistemele de baze de date distribuite trebuie să satisfacă un set de proprietăți, cum ar fi
 - ◆ Consistență (*Consistency*),
 - ◆ disponibilitatea (*Availability*)
 - ◆ toleranța la partiționare (*Partition tolerance*),

Scalabilitatea orizontală



Baze de date NoSQL

- O dată cu stabilizarea tehnologiilor legate de Cloud Computing, bazele de date de tipul NoSQL devin tot mai atractive, atât datorită suportului nativ pentru scalabilitate și arhitecturi distribuite, cât și pentru faptul că multe dintre acestea pot fi oferite ca și servicii, pe modelul *pay-as-you-go*.
- În funcție de tipul problemei care se dorește a fi rezolvată, există mai multe feluri de baze de date NoSQL. Tabelul următor exemplifică principalele tipuri, precum și cele mai cunoscute implementări comerciale.

Tipuri de baze de date NoSQL

Tip	Implementări	Descriere
Cheie-Valoare (Key-Value)	Amazon Dynamo , Voldemort , Tokyo Cabinet , Riak	Datele sunt stocate sub forma unui dicționar (<i>hash</i>), suportă operații de tip: <i>put</i> , <i>get</i> , <i>delete</i> .
Document	CouchDB , MongoDB	Extensie a sistemelor cheie-valoare, suportă ierarhii și imbricări de perechi cheie-valoare, organizate sub forma unui document.
Columnare	BigTable , HBase , HyperTable	Datele respectă o schemă predefinită, dar au avantajul unor tempi foarte mici de căutare și agregare datorită modului în care sunt stocate pe suportul fizic.
Graf	OrientDB , Neo4j , InfoGrid	Pentru reprezentarea datelor se folosesc elemente din teoria grafurilor, ca noduri, relații și proprietăți. Foarte eficiente pentru reprezentarea problemelor ce pot fi descrise prin grafuri

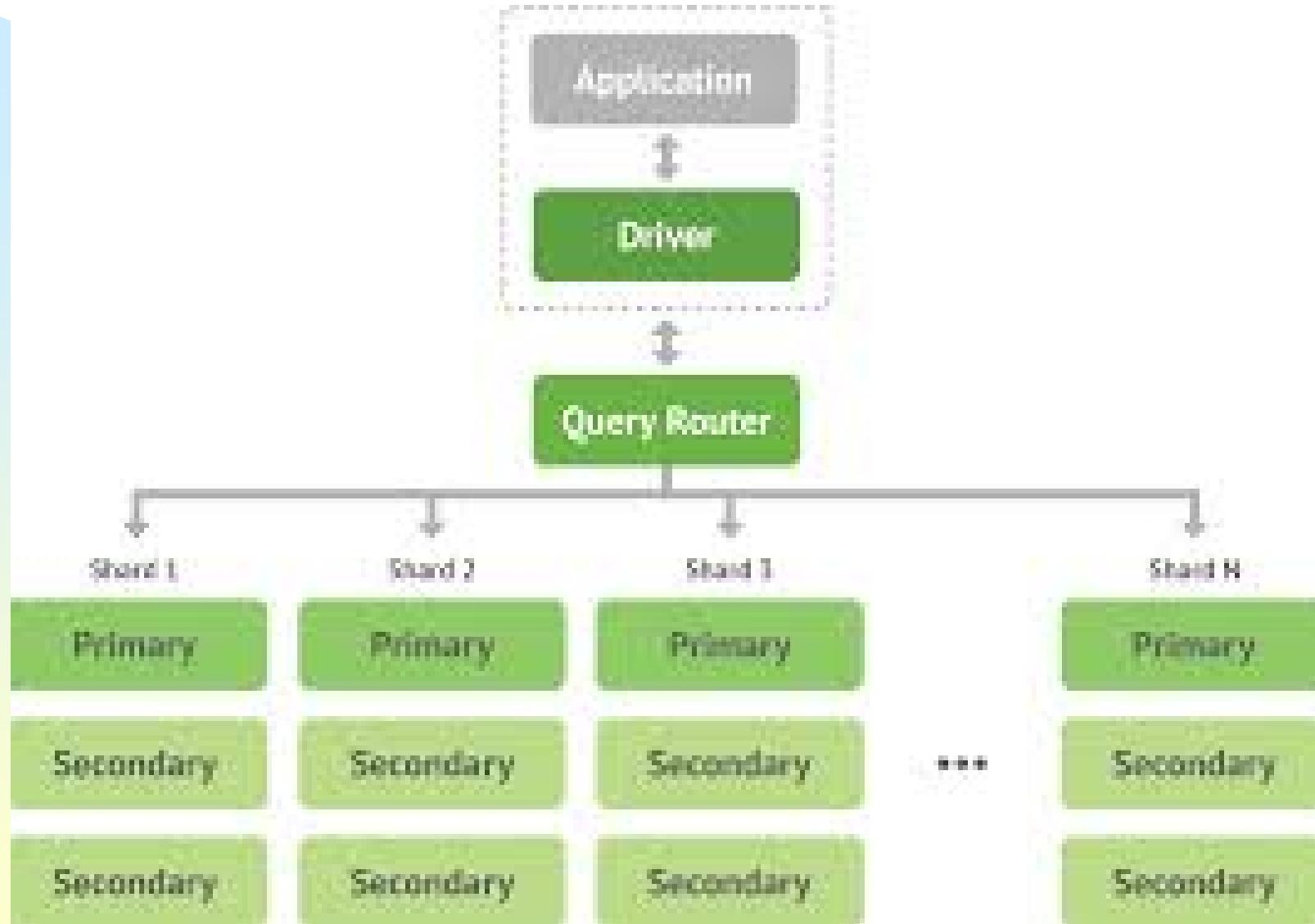
Baze de date “cloud”

- O baza de date “cloud” este o baza de date care ruleaza pe platforma de cloud computing, cum este Amazon EC2, GoGrid si Rackspace.
- Există două modele de implementare:
 - ◆ utilizatorii pot rula baza de date în “cloud” independent
 - ◆ utilizatorii pot plati pentru a avea acces la niste servicii de baze de date

Xeround

- Un exemplu de "cloud database" este Xeround, care inlocuieste baza de date MySQL
- Xeround functioneaza in "nor" si ofera utilizatorilor posibilitatea de a avea acces la aceste date chiar si in cazul in care serverul local se prabuseste.
- Are o arhitectura bazata pe doua nivele: nodul de accesare date si nodul de date.

MongoDB



Riak

- Riak este un Key-Value store distribuit, open-source cu suport enterprise, dezvoltat de Basho Technologies.
- Riak consideră toate nodurile din cluster egale, fără master, deci nu are Single Point Of Failure.
- Oferă availability, scalability, este tolerant la partitionarea rețelei și asigură replicarea datelor în 3 locații,
- Rezolvarea conflictelor de date este rezolvată cu ajutorul vector clock (ceas logic).

Hypertable

- Este bazat pe filozofia Google Big Table si concureaza cu HBase.
- Prin un API generic, suportă diverse sisteme de fișiere distribuite, cel mai utilizat fiind HDFS
- HDFS este sistemul distribuit de fișiere oferit de Hadoop si inspirat din Google File System

Cassandra

- Cassandra a porinit ca un proiect open-source de la Facebook, ajuns sub umbrela Apache. Implementarea îmbină concepte din Amazon Dynamo și Google BigTable.
- Poate fi caracterizat ca un key-value store eventually consistent.
- La fel ca Riak și Cassandra pornește de la conceptele publicate de Amazon în lucrarea ce descrie Dynamo.

Referinte

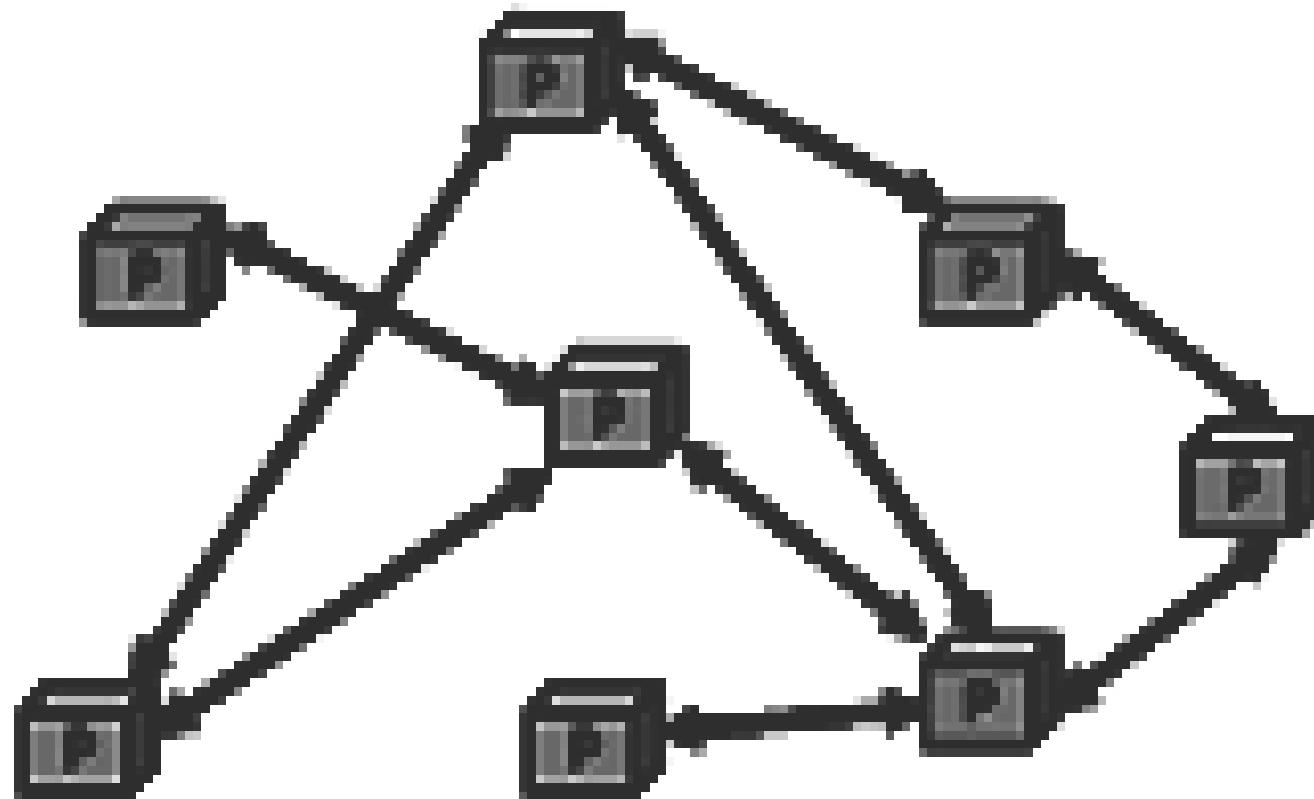
- Ozsu, M. Tamer, and P. Valduriez. *Principles of Distributed Database Systems*, Englewood Cliffs, NJ: Prentice Hall, 1991.
- Angelo R. Bobak, *Distributed and Multi-Database Systems*, Boston, US, Artech House, 1995
- <http://ctrl-d.ro/development/resurse-development/introducere-in-tematica-bazelor-de-date-in-cloud/>
- [http://www.todaysoftmag.ro/article/ro/10/Bazele de date NoSQL - o analiza comparativa 304](http://www.todaysoftmag.ro/article/ro/10/Bazele_de_date_NoSQL - o analiza comparativa 304)

Cursul 11

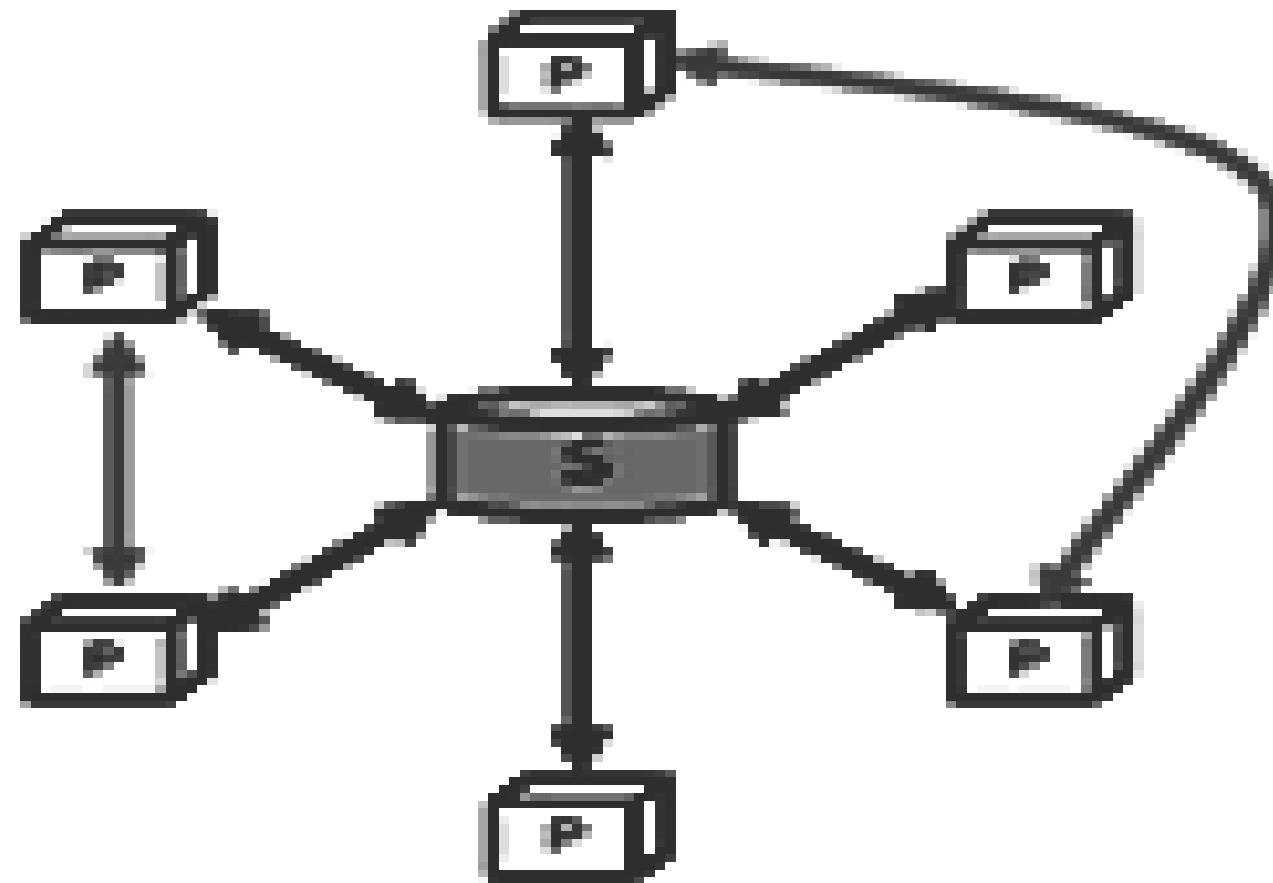
P2P

- Un sistem peer to peer (P2P) este un sistem cu autoorganizare format din entitati (peers) autonome ale caror scop este folosirea in comun a unor resurse distribuite prin intermediul unei retele de comunicatii eveitand folosirea unei resurse centralizate
- Aceste sisteme reprezinta o schimbare de paradigm la nivelul proiectarii sistemelor distribuite.

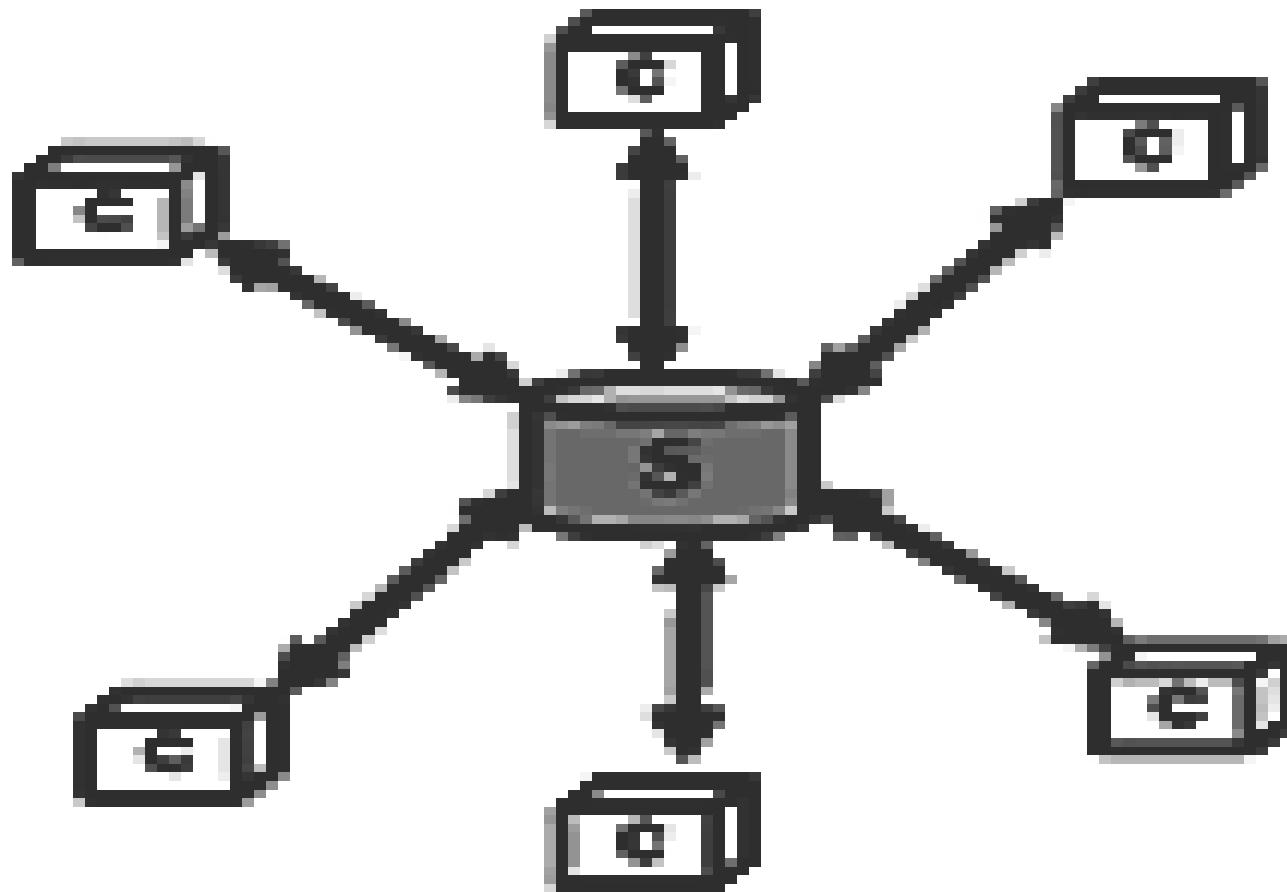
clasic



hibrid



Stil cli-ser



Utilizarea descentralizata a resurselor

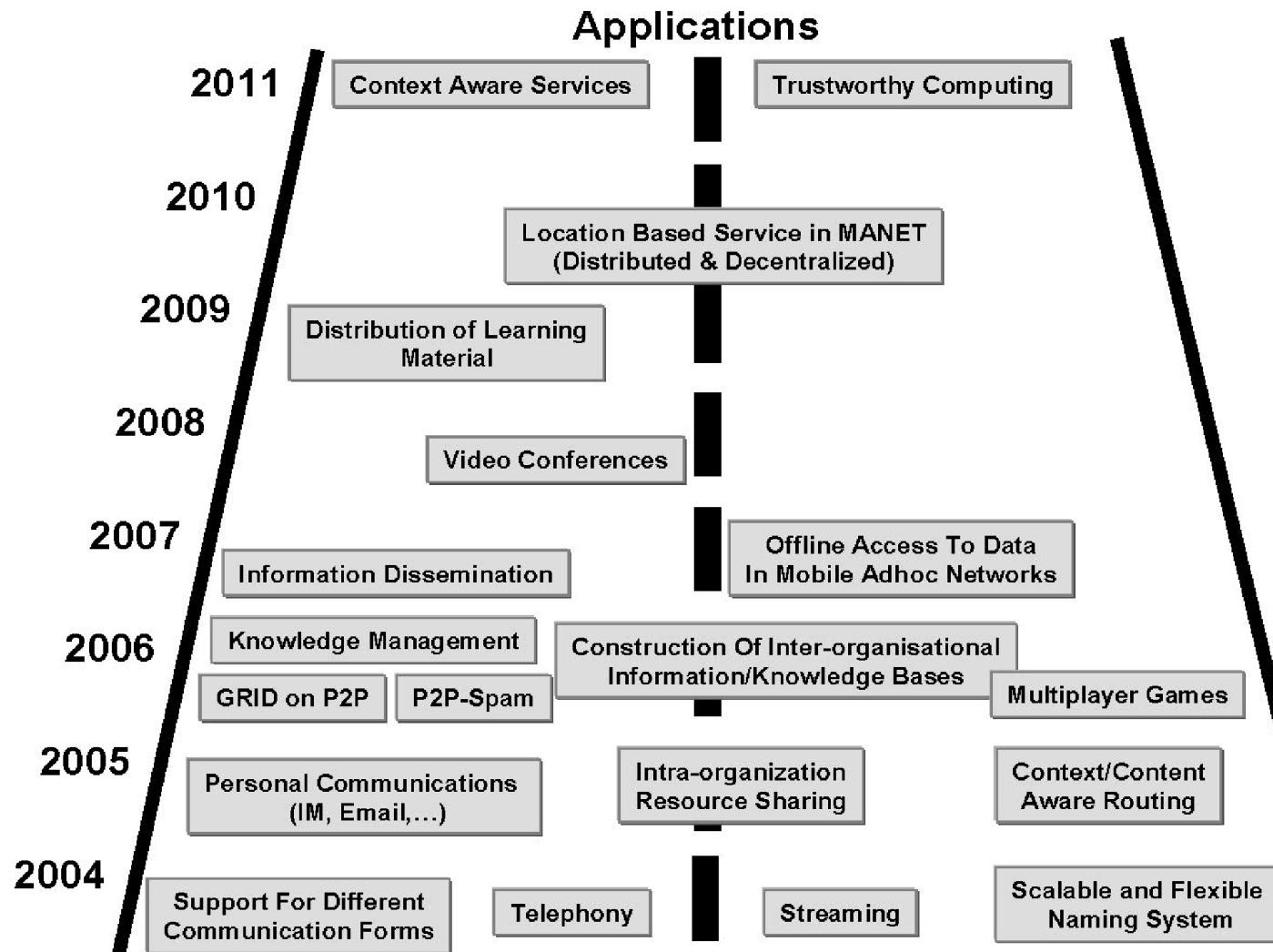
- Resursele in acest caz se refera la
 - ◆ dimensiune de banda disponibila pentru transfer
 - ◆ dispozitive de stocare si capacitatea libera (disponibila pentru cooperative)
 - ◆ putere de calcul

- Partenerii sunt distribuiți la nivel global și de obicei folosesc Internetul ca support de comunicare.
- Disponibilitatea unui partener nu poate fi analizată foarte bine deoarece adresa de internet se schimba dinamic deci nu poate fi folosită ca o referință de incredere.
-)

Autoorganizarea distribuita

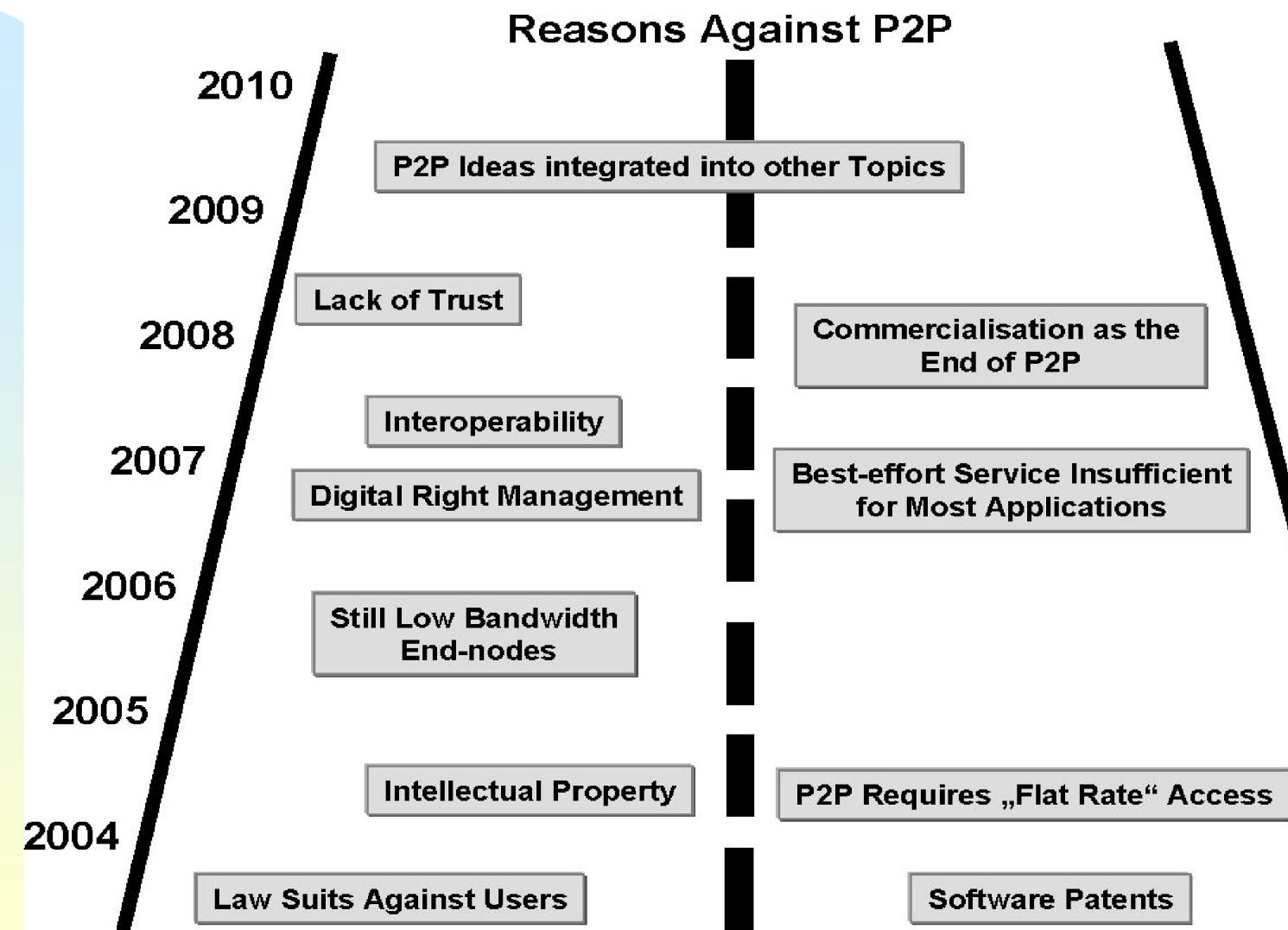
- Este esenta sistemelor P2P partenerii interactioneaza direct unui cu altii.
- In modelul original nu exista nici o forma de coordonare.
- Aceasta abordare reduce la maxim probabilitatea de aparitie a supraincarcarilor dar aduce si dezavantaje legate de o disponibilitate mai scazuta decat in cazul solutiilor client server clasice.

Aspecte privind dezvoltarea P2P



Aplicatii peste File-Sharing

Contra 2010



Pro&Contra

Sisteme P2P nestructurate

- De obicei erau de tip hibrid si se bazau pe existent unui server central care retina locatia fiecarei informatii din sistem.
- Dupa ce era obtinuta locatia sistemele incepeau sa comunice direct.
- Alte abordari ca Gnutella foloseau o tehnica de inundare

Sisteme P2P Structurate

- Avand in vedere problemele discutate anterior evolutia catre acest nou tip de sisteme era fireasca.
- Aici se urmareste dezvoltarea unor sisteme de stocare a datelor distribuite adresabile pe baza de continut.
- Pentru aceasta se vor folosi **Distributed Hash Tables**.

dht

Data

Fox

Hash function

DFCD3454

The red fox
runs across
the ice

Hash function

52ED879E

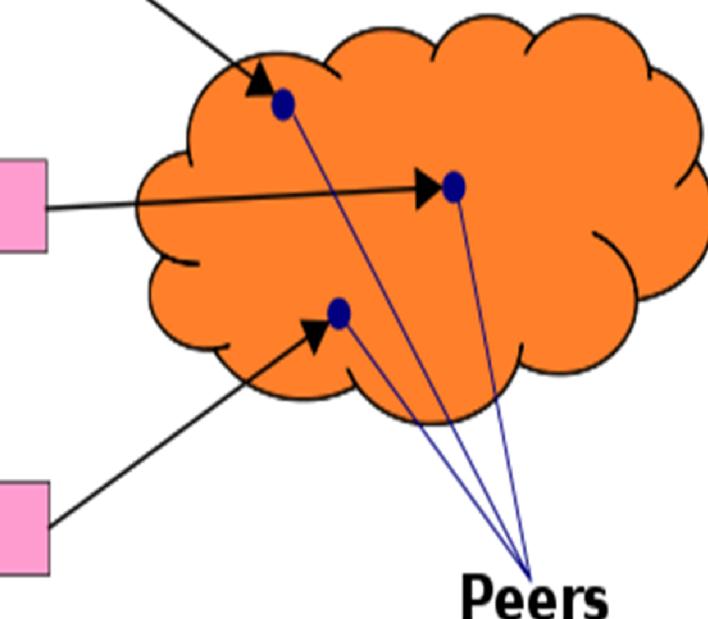
The red fox
walks across
the ice

Hash function

46042841

Key

Distributed Network



Arii de aplicare a sistemelor P2P

Presence Information

- Este esentiala pentru sistemele P2P deoarece permite entitatilor gasirea rapida a informatiilor despre vecini precum si a resurselor disponibile.
- De asemenea sunt interesante pentru distribuirea efectiva a cererilor de calcul pe baza crearii unor scenarii legate de gradul de disponibilitate ale unor noduri

Gestiunea documentelor (document management systems – DMS)

- DMS dedicate sunt de obicei bazate pe solutii centralizate.
- Ele permit accesul, procesarea si utilizarea in comun la datele (documente) gestionate
- Solutia are dezavantajul centralizarii
- Aceasta noua directive a aparut pornind de la observatia ca o mare parte din documentele gestionate de un DMS sunt distribuite pe o multitudine de noduri iar cererile catre un repository central sunt rare.

Colaborare

- Sistemele P2P pot fi considerate ca baza si pentru dezvoltarea unor sisteme colaborative de orice nivel pornind de la operarea in comun peste niste fisiere pana la o integrare totala intr-un spatiu virtual
- Totusi la ora actuala aplicatiile sunt inca separate si cele mai comune sunt din categoria virtual office sau de tip skype

Resursele gestionate de o aplicatie P2P

Fisierele

- Sunt specifice majoritatii aplicatiilor P2P se ocupa cam 70% din traficul pe Internet
- Problema principala – localizarea resurselor
 - ◆ Gnutella – flooded request
 - ◆ Napster – centralized directory
 - ◆ FreeNet – document routing

Gnutella

- Aceasta nu are autoritate centrală iar toți membrii sunt egali.
- Cererile de căutare care urmează modelul cu inundare la cerere (cererea este trimisă la un număr predestinat de membri).
- Dacă un membru nu poate satisface cererea el o va trimite la randul lui la alți membri

Napster - RIP

- Un sistem P2P hibrid în care indexarea este furnizată centralizat de o entitate coordonatoare care în urma unei cereri îi furnizează clientului lista cu locatii unde se gaseste replicata informatia dorita apoi clientul ia legatura direct

FreeNet

- Foloseste modelul rutarii documentelor. In acest caz fisierele nu sunt depozitate pe hdd-ul peer-ului care le pune la dispozitie ci in alte locatii din retea.
- Aceasta inseamna ca nici un peer nu stie ce documente stocheaza.
- Fisierele su utilizatorii au asociate un identificator numeric unic

Largimea de banda

- Lăsând P2P hibrid se poate obține o echilibrare a încarcării mult mai eficientă ca la soluțiile client server clasice
- La fel descarcarea fisierelor este mai bună deoarece parti diferite din fisier sunt aduse de la adrese diferite

Stocarea:

- O retea de stocare tip P2P este reprezentata de un cluster de computere creat peste reteaua deja existenta.
- Se utilizeaza pentru imbunatatirea vitezei de transfer si scaderea efortului gestiunii .
- Cateva sisteme sunt
 - ◆ PAST, Pasta, OceanStore, CFS, Farsite, si Intermemory.
- Pentru a participa intr-o astfel de retea fiecare partener primeste o pereche de chei publica/privata.

- Stocarea si cautarea in retea sunt dupa modelul rutarii documentelor.
- Se foloseste si replicarea cu un factor anume.
- Fiecare peer isi citeste din retea propria versiune de tabela de rutare care este folosita pentru stocare si cautare.

Utilizarea ciclurilor processor

- De obicei aceste task-uri sunt gestionate de un numar de noduri particulare care au performante superioare de calcul si disponibilitate crescuta.
- Pentru a gestiona caderile la nivelul acestor noduri se foloseste un algoritm Bizantin.

Furt cicluri idle

- Cel mai apropiat proiect ca idee a fost analiza distribuita realizata de SETI@home (Search for Extraterrestrial Intelligence) - University of California, Berkeley,
- Se observa abordarea SETI era tip client server, ramane doar idea furtului de cicluri idle

Proiectarea unei aplicatii P2P

- O astfel de aplicatie trebuie sa indeplineasca urmatoarele deziderate
 - ◆ Sa fie adaptabila la orice mediu – portabila
 - ◆ Trebuie sa poata interactiona cu celelalte aplicatii (sa respecte un standard) indifferent de forma de comunicare in retea (LAN, WAN..)
 - ◆ Trebuie sa fie scalabila
 - ◆ Desi telul primar este clar o arhitectura “deschisa” bazata pe plugin-uri pentru adaugarea usoara a unor facilitati utilizator dezvoltate in comunitate ar fi de dorit

- De asemenea aplicatia trebuie sa se supuna si la urmatoarele constrangeri.
- Una ar fi rezolvarea lucrului in spatele unui proxy
- Nu este suficiente utilizarea serviciilor de tip Whols deoarece la ora actuala sunt dese cazurile in care exista mai multe IP-uri mapate peste aceeasi adresa web

- O solutie simpla este prezentat la nivel de pseudo cod mai jos

Exista un IP unic obtinut prin interogare directa?

Daca nu exista macar un IP de domeniu?

Se converteste acest IP in reprezentarea ca octeti si se realizeaza o cautare intre adresa minima si adresa maxima. Se verifica cu aceasta ocazie daca este valid.

Daca nu IP-ul se considera a fi de domeniu

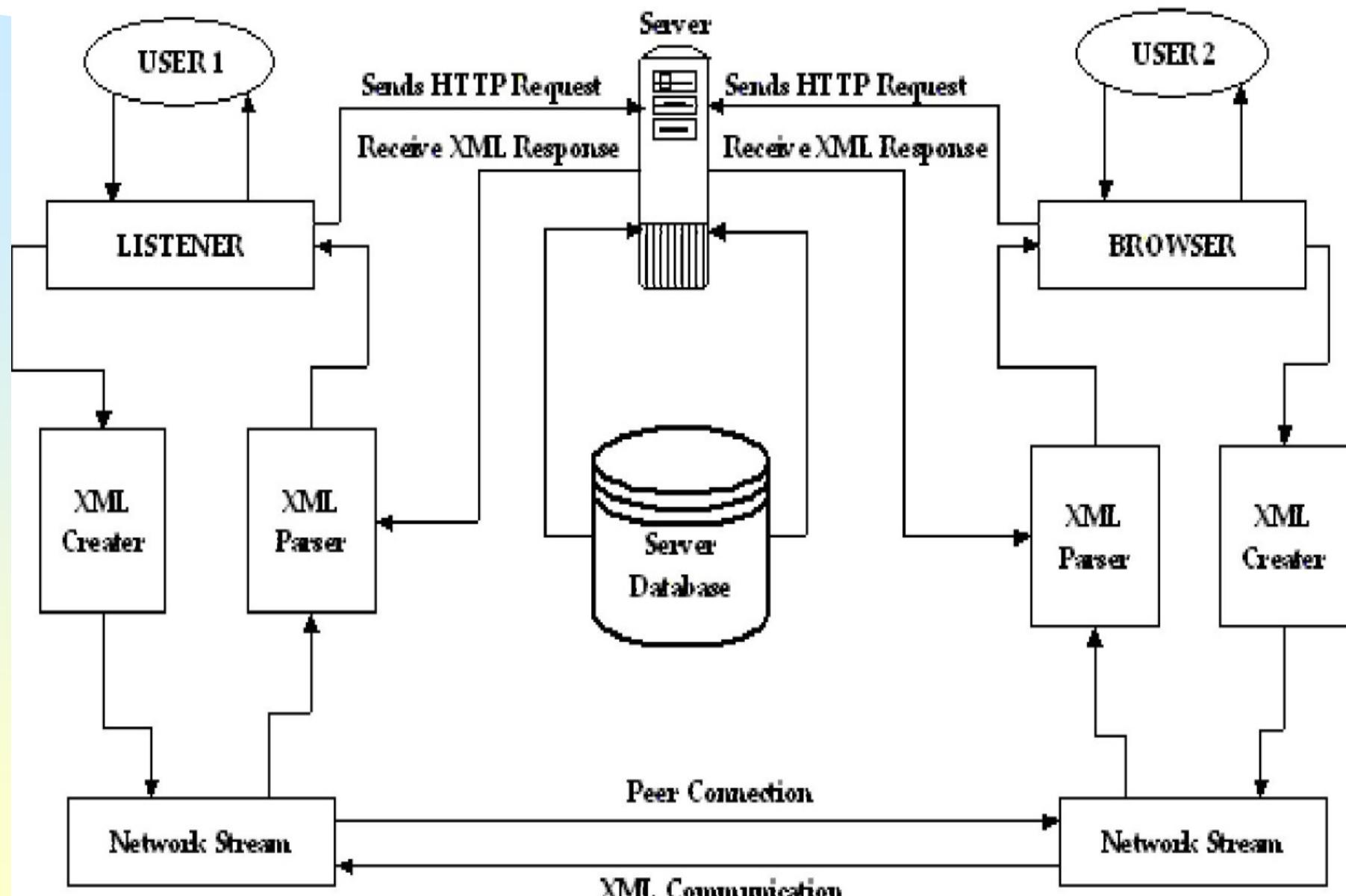
Probleme specific transferului de date

- Este de asteptat ca serverele de acest tip sa fie continuu supraincarcate datorita lipsei de reguli de selectie in acceptarea utilizatorilor.
- De asemenea apare o problema legata de stabilirea timpului de viata pentru o conexiune deja acceptata
- Timp scurt →
- Timp lung →

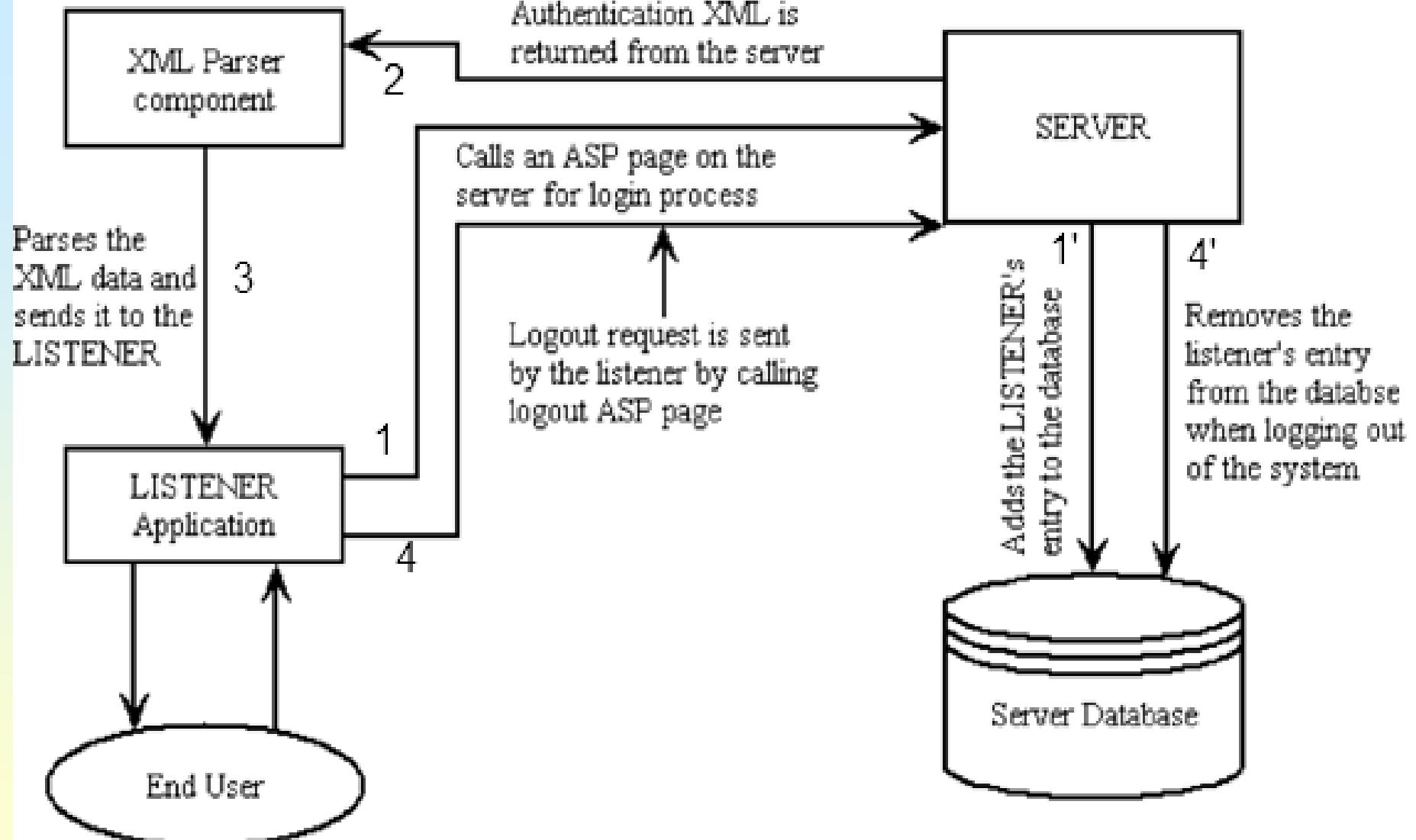
Securitatea

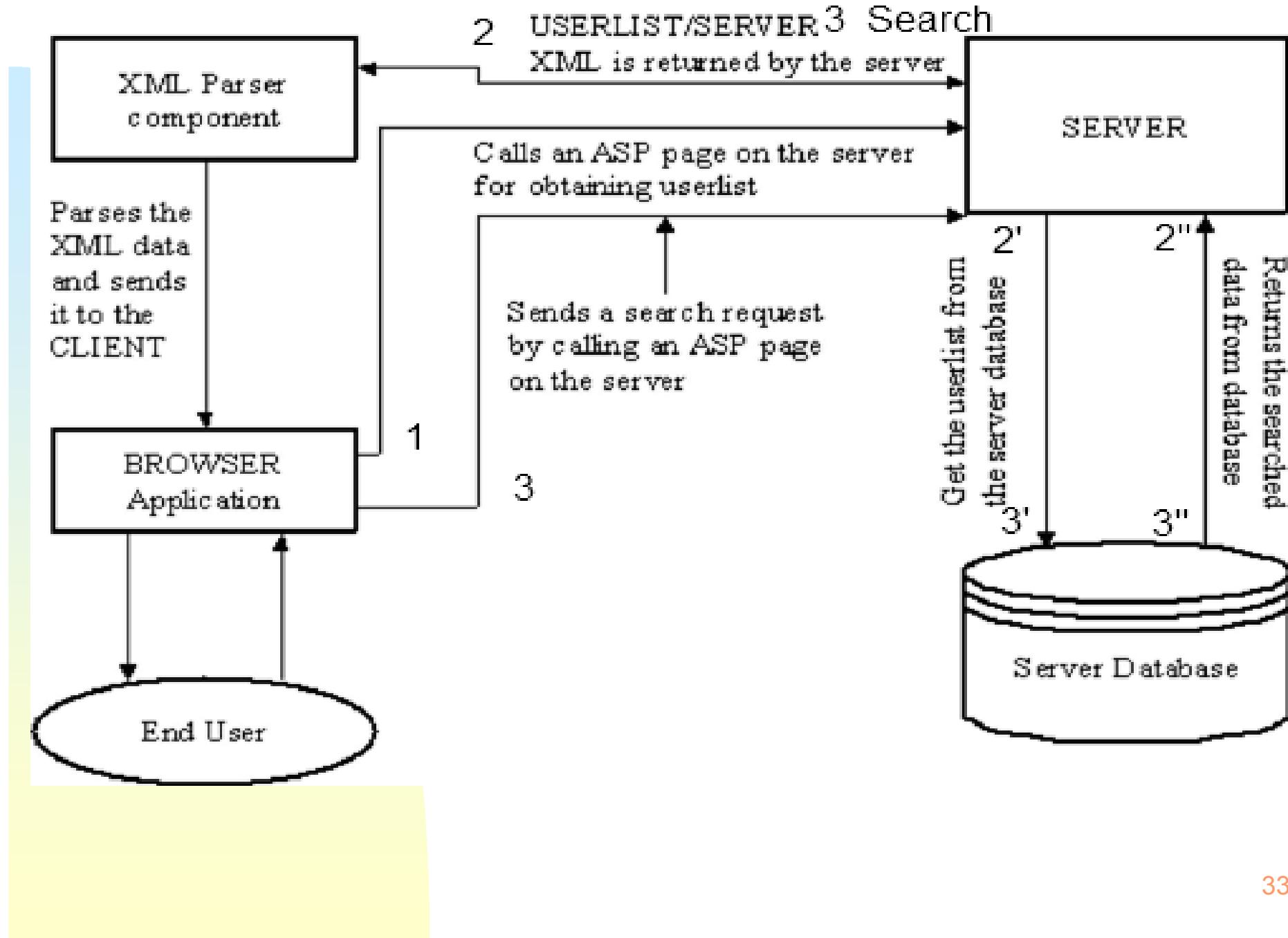
- Este bine sa se se foloseasca portul 7070 conform clasificarii IANA dar sa ofere totusi utilizatorului posibilitatea configurarii manual a acestuia
- Pentru a pastra un bun control dpdv al ISP-ului dar si al utilizatorului este bine sa se automatizeze procesul de setare al firewall
- Se poate folosi si transferal criptat bazat pe https dar numai la cererea explicita a utilizatorului
- Mai jos este prezentata o posibila structura pentru o aplicatie P2P

arhitectura



comunicare





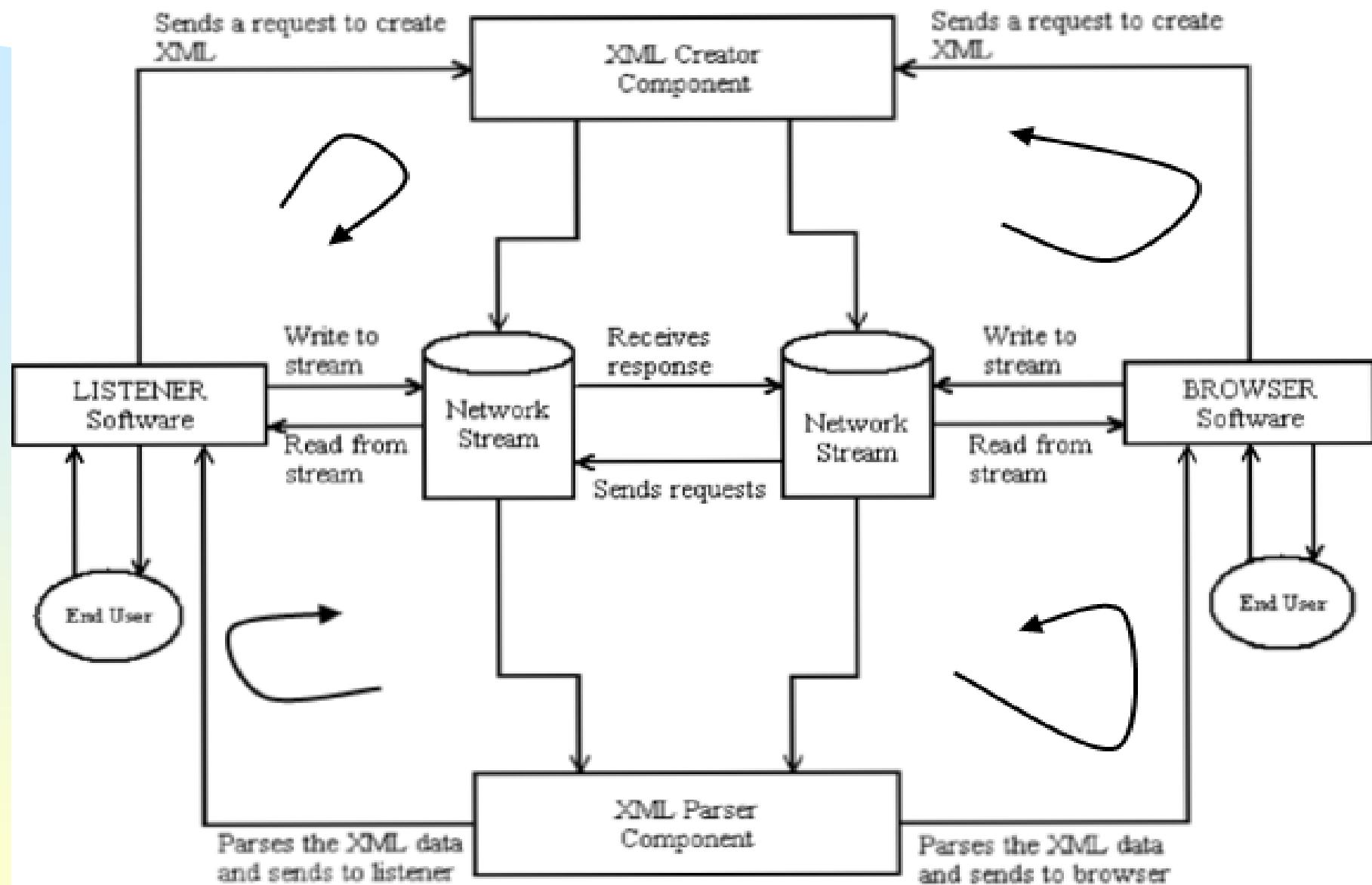


Tabela peer

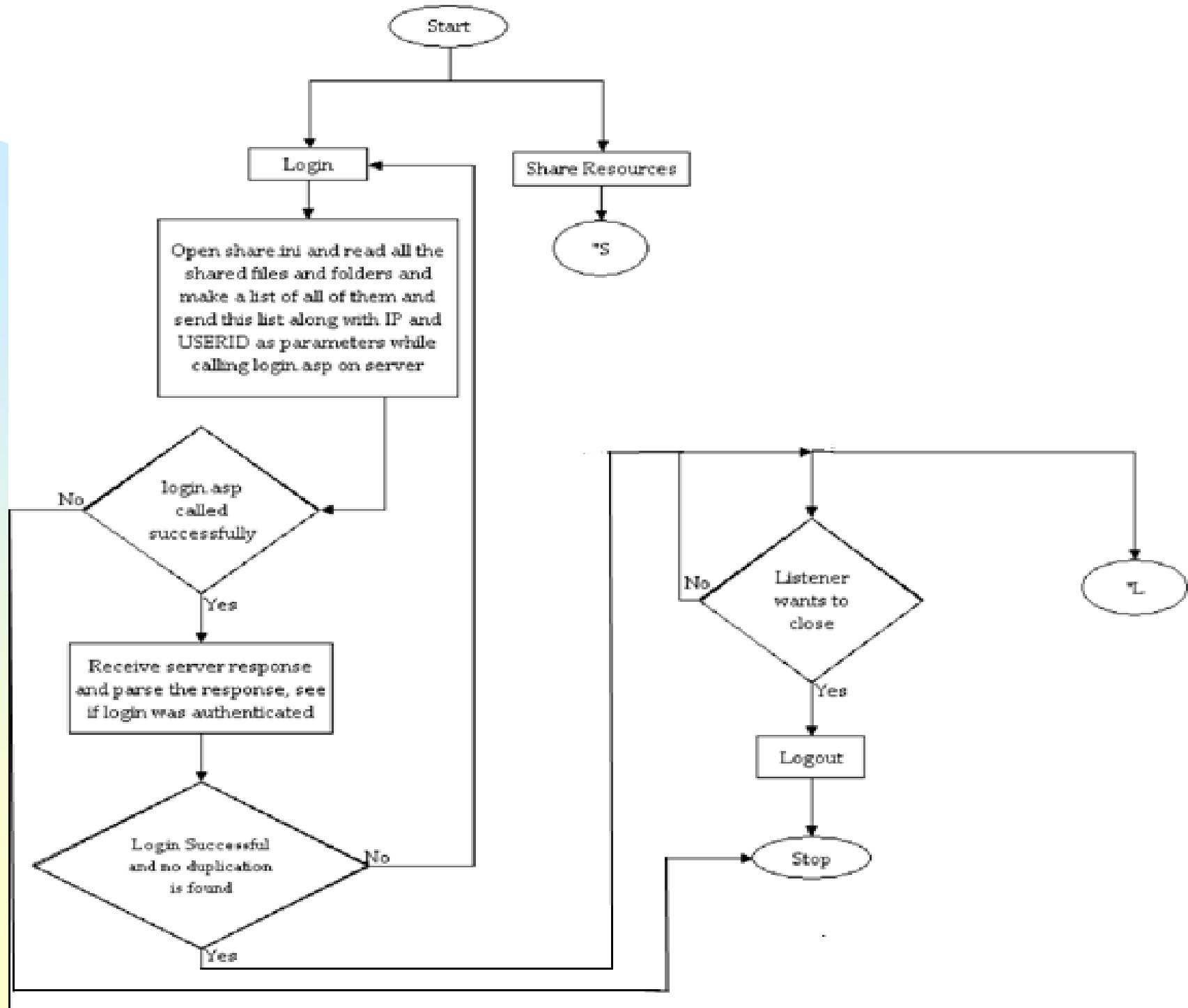
Tabela Peer

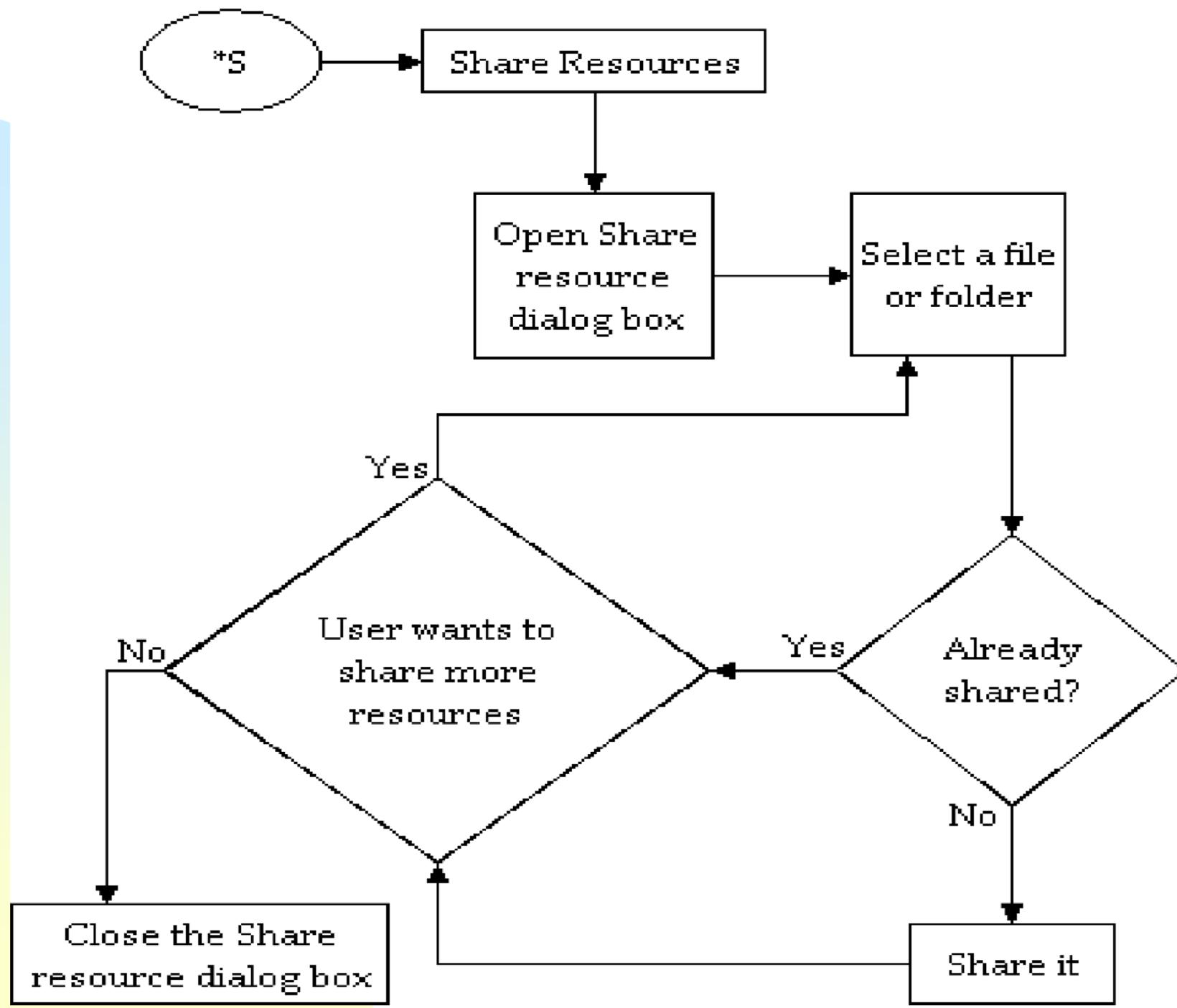
<i>Nume camp</i>	<i>Descriere</i>
ip_address	Adresa IP remote a utilizator
user_name	Numele utilizatorului
Status	Starea conexiunii (0 or 1)
connected_time	Timpul de conectare

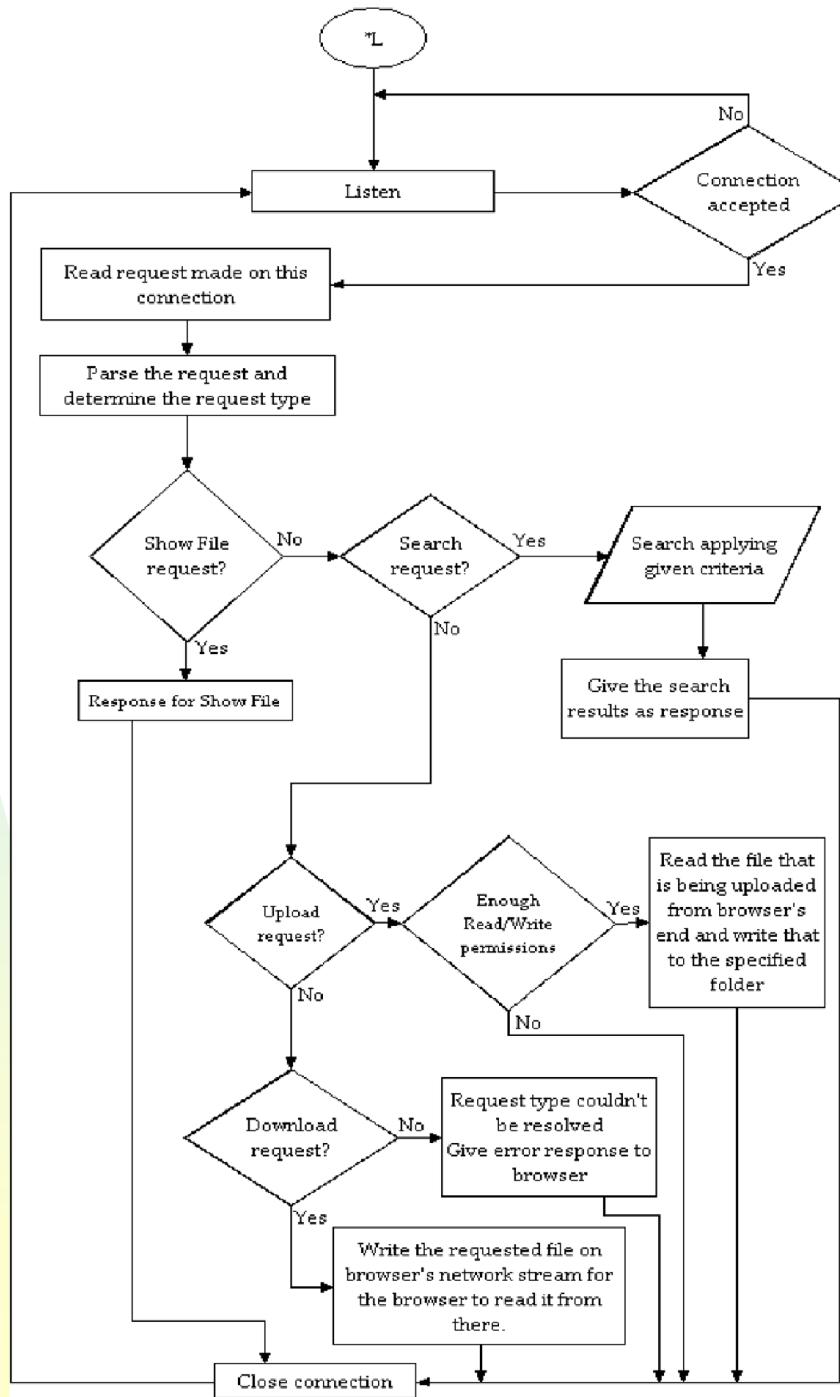
Tabela share

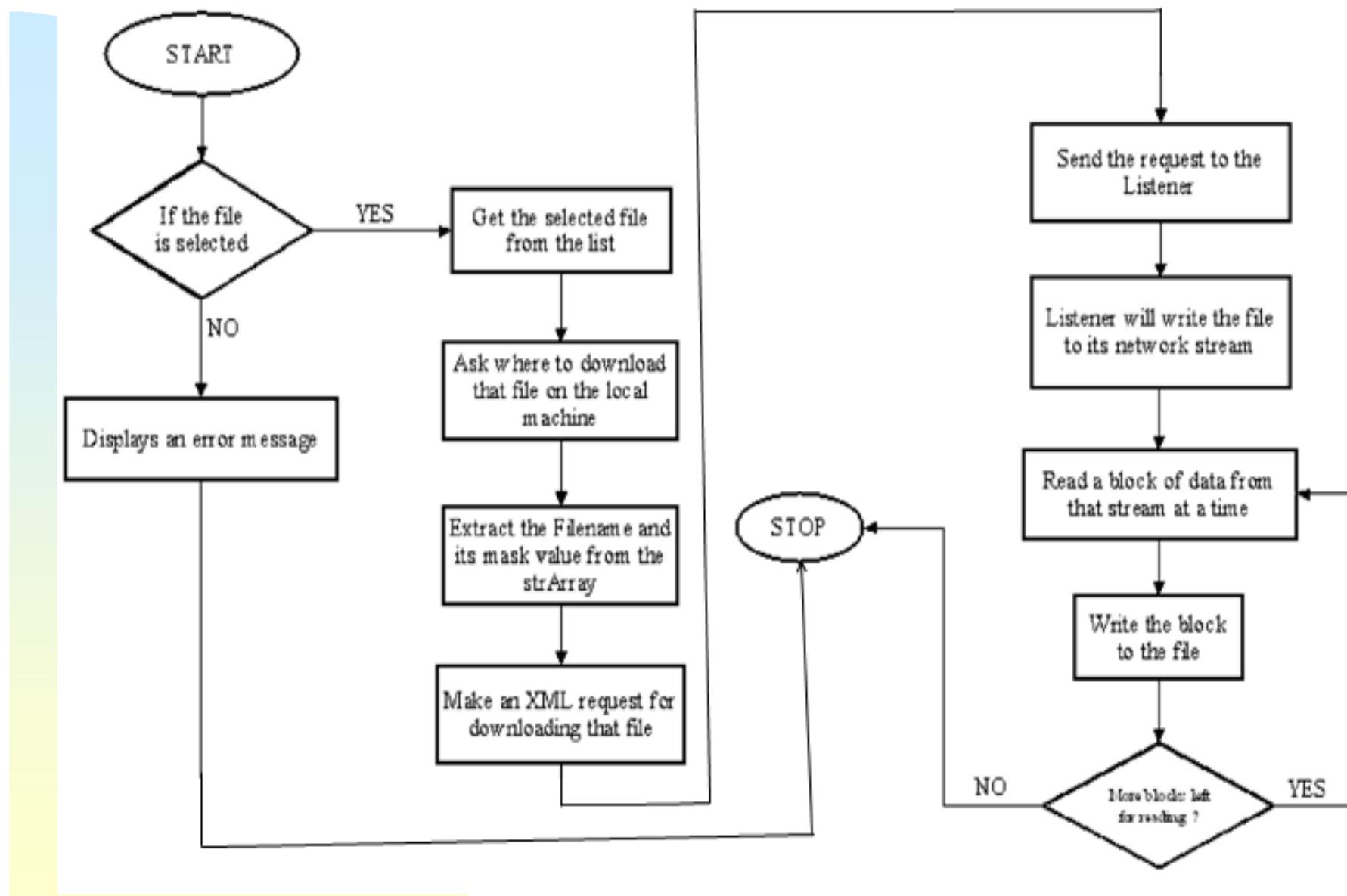
Tabela Share

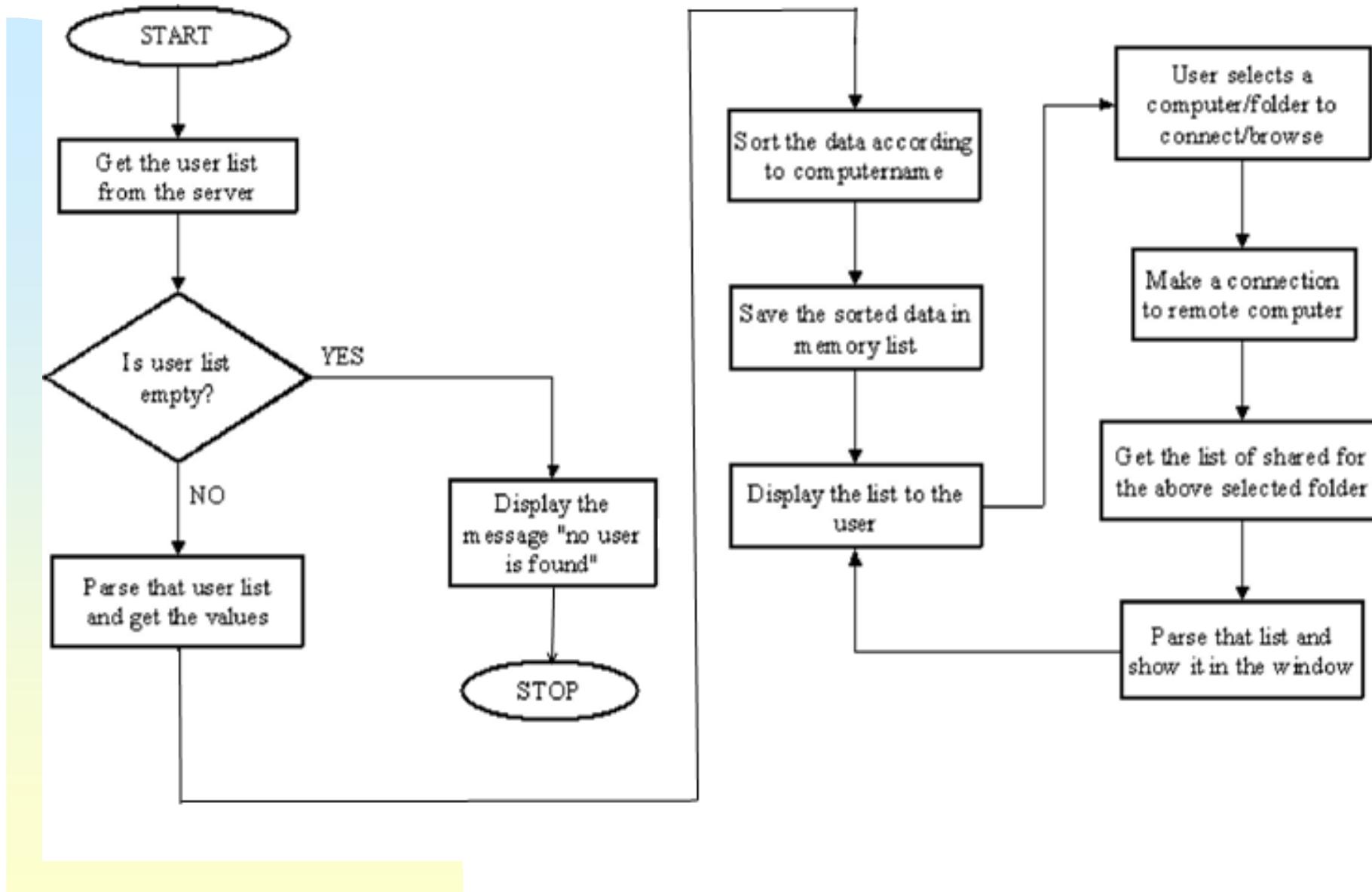
Nume Camp	Descriere
<u>ip_address</u>	Adresa IP remote a <u>utilizator</u>
<u>file_name</u>	<u>Fisierele puse la dispozitie de utilizator</u>

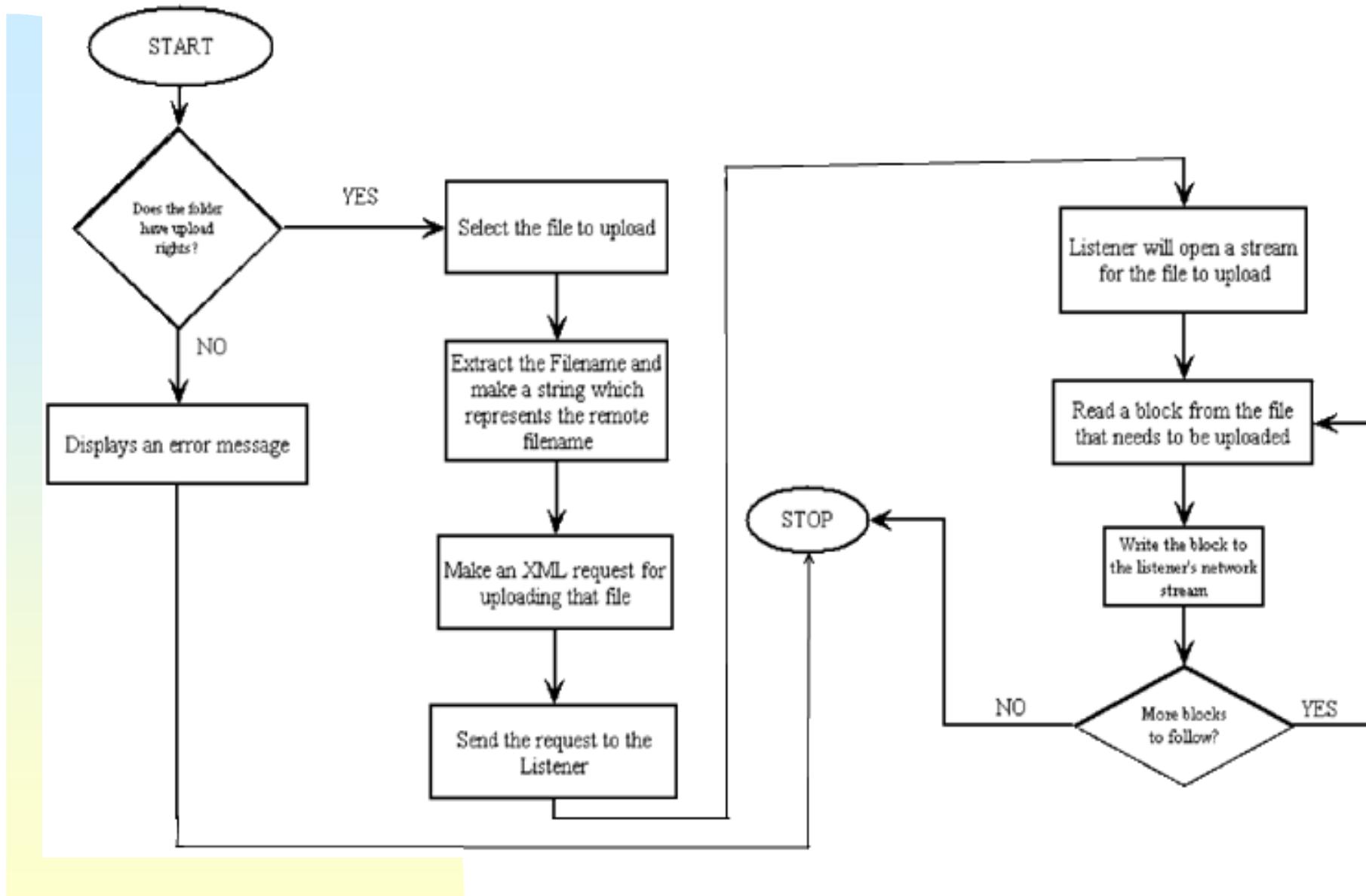










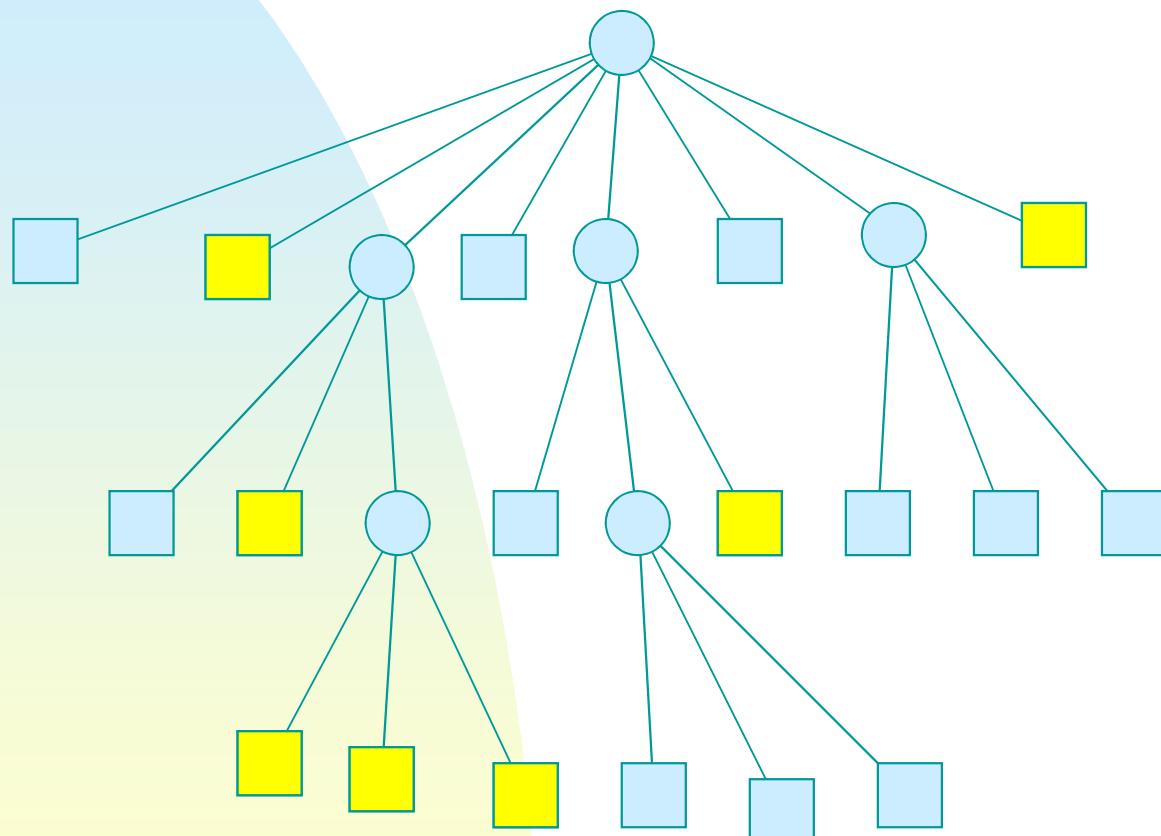


Clasificare documentelor

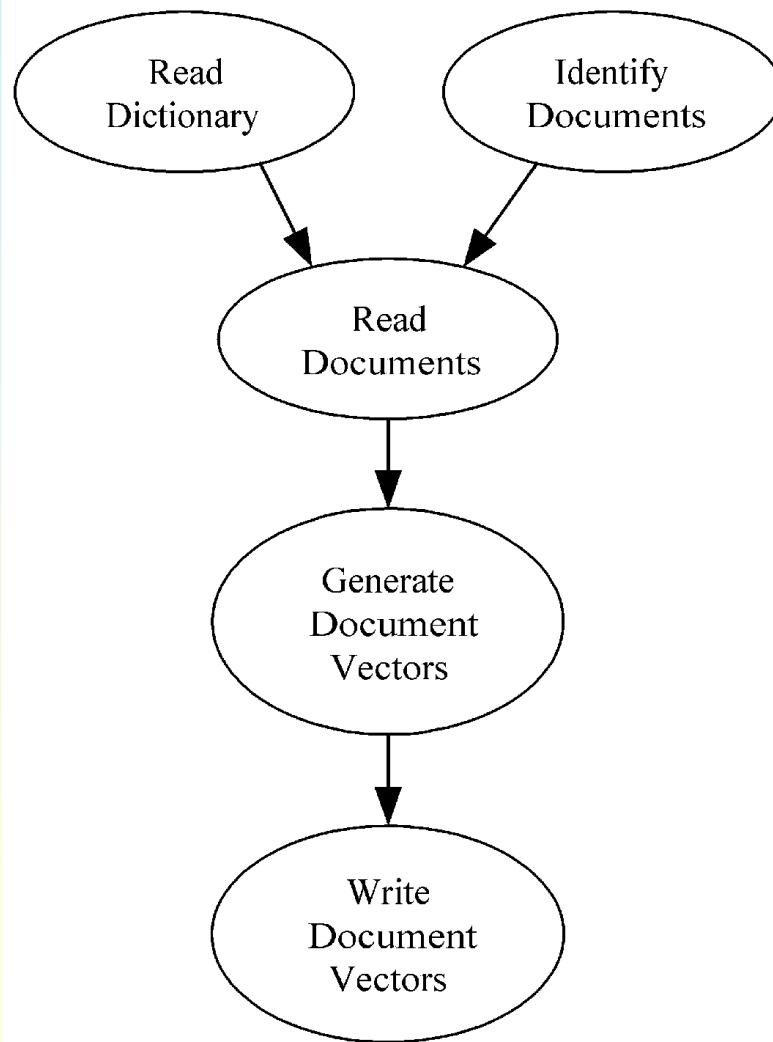
Problema clasificarii documentelor

- Se cauta in directoare, subdirectoare dupa documente (de ex. .html, .txt, .tex, etc.)
- Se foloseste un dictionar de cuvinte pentru a crea un vector profil pentru fiecare document
- Se stocheaza aceste profile
- Aceasta problema este interesanta deoarece permite o descompunere functionala

Problema clasificării documentelor



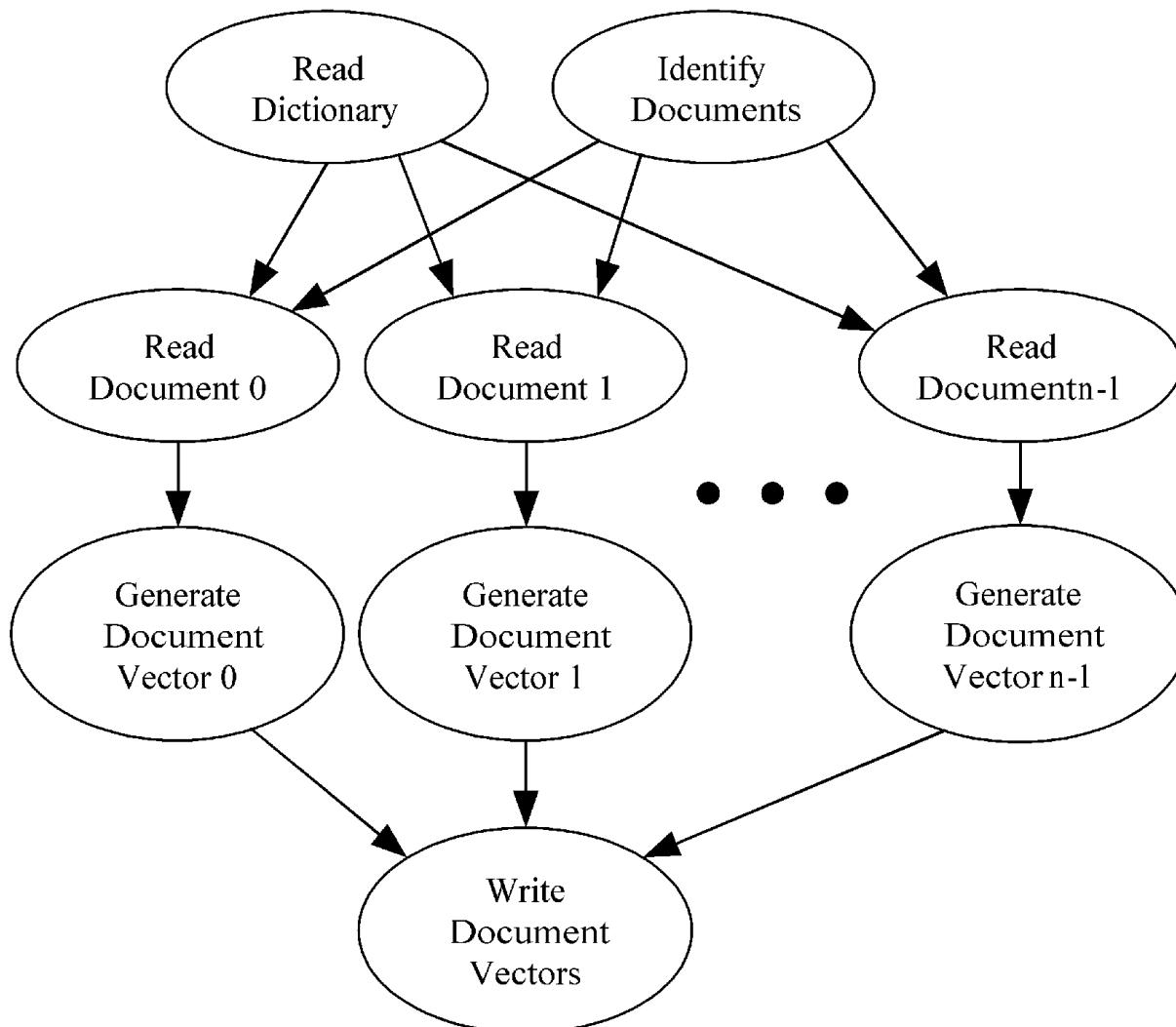
Graful de dependinte a datelor (1)



Partitionare si comunicare

- Cel mai mult timp se pierde cu citirea documentelor si generarea vectorului profil
- Deci trebuie create doua task-uri primare pentru fiecare document
 - ◆ 1) citeste fisierul ce contine documentul
 - ◆ 2) genereaza vectorul

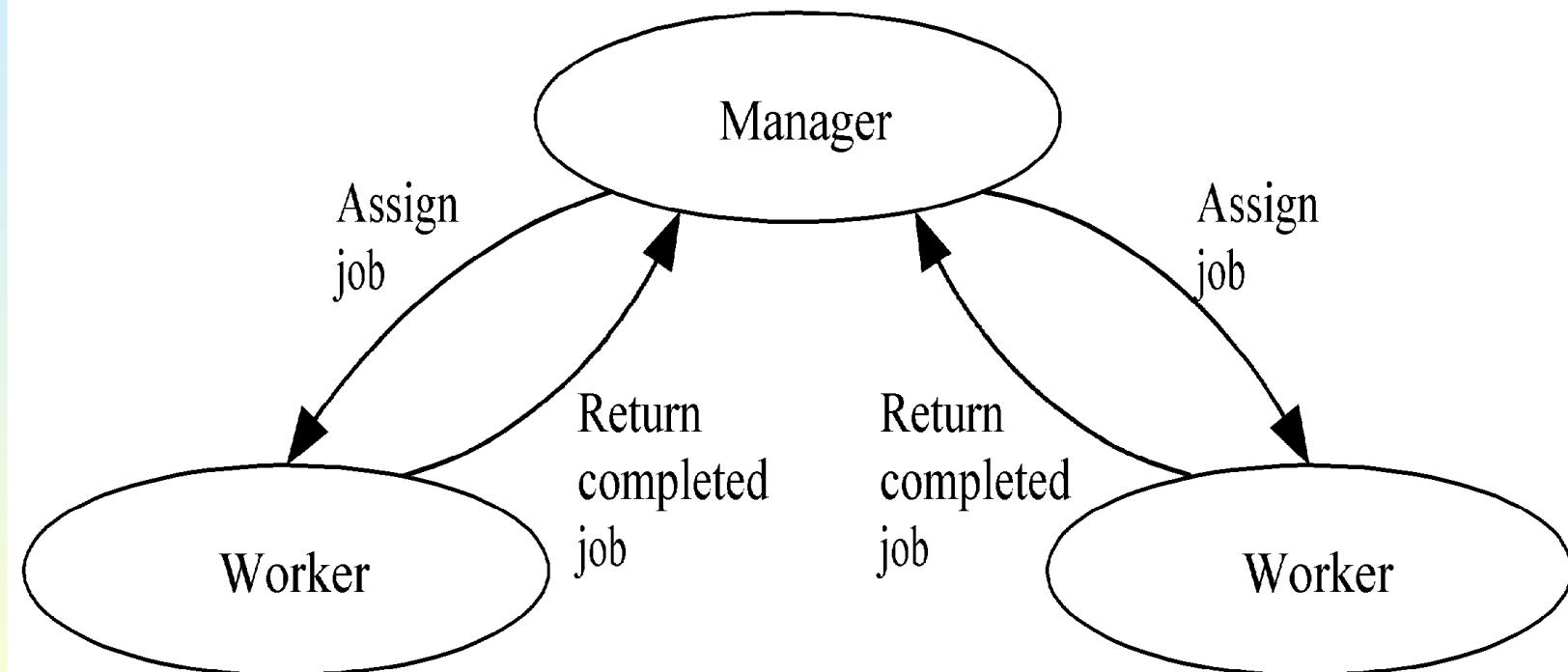
Graful de dependinte a datelor (2)



Aglomerarea si maparea

- Numarul de task-uri nu este cunoscut in momentul compilarii
- Task-urile nu comunica intre ele
- Timpul necesar pentru executia task-urilor variaza mult.
- Documentele difera in dimensiune si cteva documente sunt mai greu de procesat

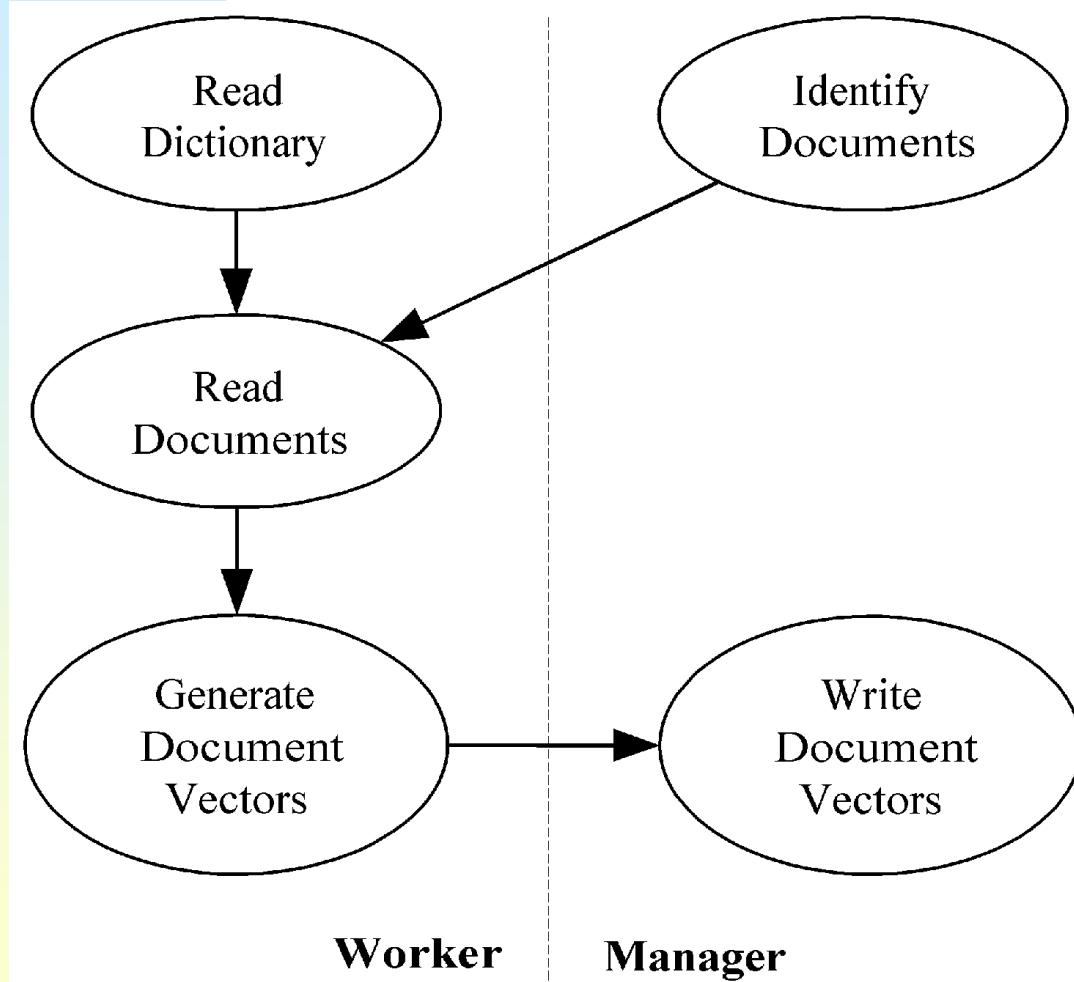
Algoritm de tip Manager/worker



Manager/Worker versus SPMD

- SPMD (single program multiple data)
 - ◆ Fiecare proces executa aceasi functie
 - ◆ Aplicatiile anterioare pot fi usor folosite
- Manager/worker
 - ◆ Procesul manager are responsabilitati diferte fata de cel worker de tip
 - ◆ Un program MPI de tip manager/worker program are o partitionare imediata a fluxului de program

Rolurile entrului Manager si Workers



Proiectarea algoritm

- **Exista doua posibilitati in proiectarea algoritmului pornind de la paradigma worker/manager**
 - **Cind managerul initiaza ciclul de lucru,**
 - **Cind workerul initiaza ciclul de lucru**

Pseudocodul pentru Manager

Identifica n documente in directorul specificat de utilizator
Receptioneaza dimensiunea k, care reprezinta numarul de elemente in fiecare document de la worker 0
Rezerva o matrice de n,k pentru a retine vectorii specifici
repeat

```
    receptioneaza mesaj de la worker
    if mesaj contine vector document
        stocheaza-l
    endif
    if daca mai sunt documente atunci trimite catre worker
        un nou nume de document
    else trimite la worker mesaj de terminare
    endif
until all toti workerii au terminat
```

Pseudocod Worker

Scrie vectorii document intr-un fisier

Trimte prima cerere de sarcini catre manager
if worker 0 then

citeste disctionarul din fisier

endif

Broadcast dictionar intre workeri

Creaza o tabela hash din dictionar

if worker 0 then

trimite dimensiunea dictionar catre manager

endif

Pseudocod Worker

repeat

 citeste numele fisier de la manager
 if nume fisier == NULL then termina
 endif
 citeste document, genereaza vectorul
 document
 trimite vectorul document catre
 manager

forever

Tabele hash

- In acest caz este necesar sa se determine daca un cuvant dintr-un document este in dictionarul de cuvinte dorite sau nu
- In aceste cazuri formele clasice de cautare nu sunt eficiente
- In consecinta trebuie folosite tabele hash
- Desi exista multe moduri de a implementa un tabel hash una din cea mai simple forme este cu ajutorul listelor “bucket”

Coduri Hash

- UN cod hash este o functie care ia un cuvint, in acest caz, si il converteste la un intreg care apartine unu anumit interval de exemplu 0..k
- Un cod hash bun are urmatoarele proprietati
 - ◆ **1) este usor de calculat**
 - ◆ **2) numerele intre 0 si k trebuie folosite cit mai apropiat de distributia uniforma**
 - ◆ **3) exista un numar minim de coliziuni, cum ar fi de ex pt word1 si word2, sa am**
 $\text{hash(word1)} = \text{hash(word2)}$

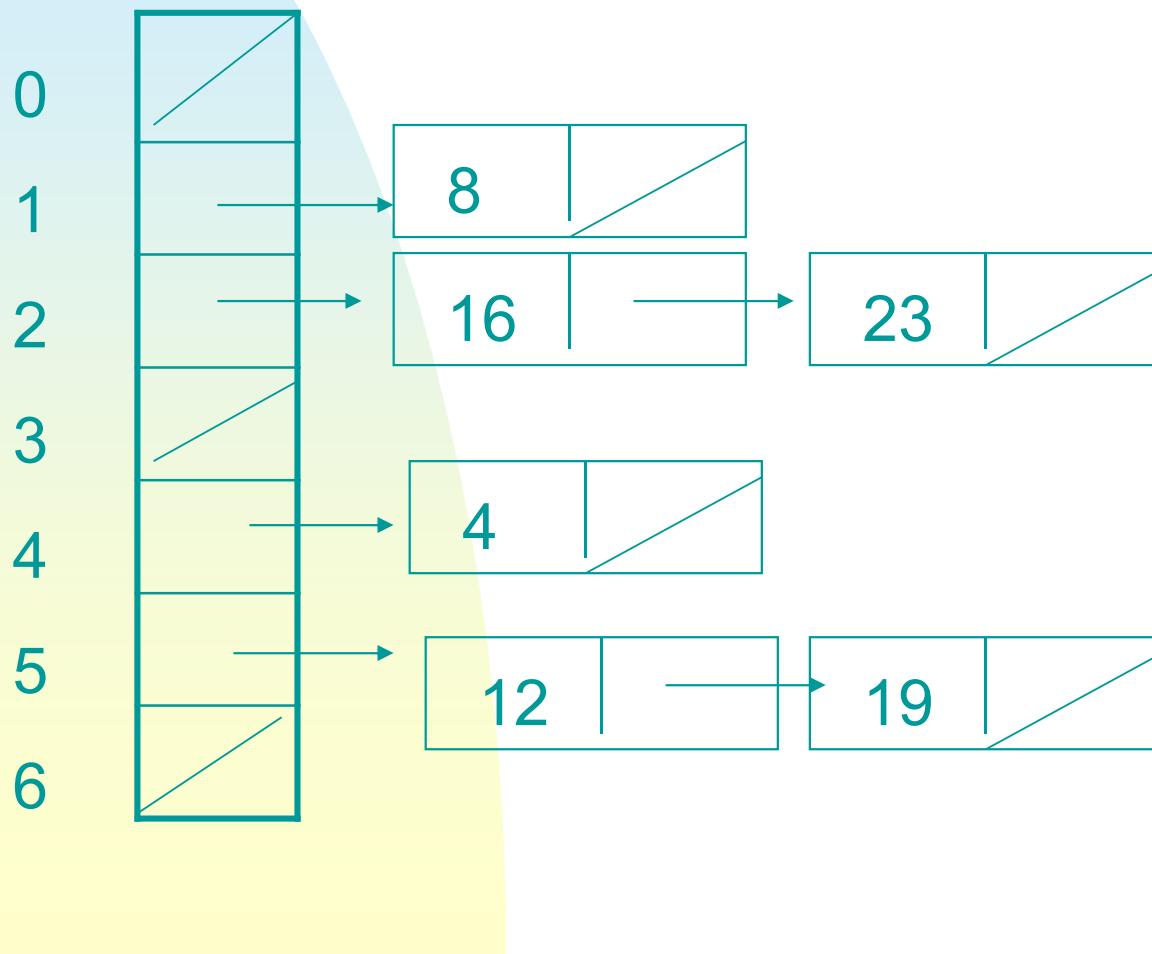
Implementarea

- **Sa presupunem ca avem un cod hash h care mapeaza cuvintele in interval $[0, k]$.**
- **Atunci cuvintele vor fi puse intr-o tabela hash dupa cum urmeaza :**
 - ◆ **Se foloseste un tablou de pointeri cu indicii de la 0 la k**
 - ◆ **Fiecare pointer retine capul unei liste inlantuite numita si "bucket" unde sunt stocate cuvintele**
 - ◆ **Indiferent daca insertia in lista se face orfonat sau nu singurul criteriu ce trebuie respectat este ca un cuvant este stocat in lista la $h(\text{word})$**
 - ◆ **Pentru a gasi un cuvant se cauta inexul tabloului $h(\text{word})$ si se cauta in lista referita de acesta**

Exemplu cu numere

Fie $h(\text{numar}) = \text{numar} \bmod 7$

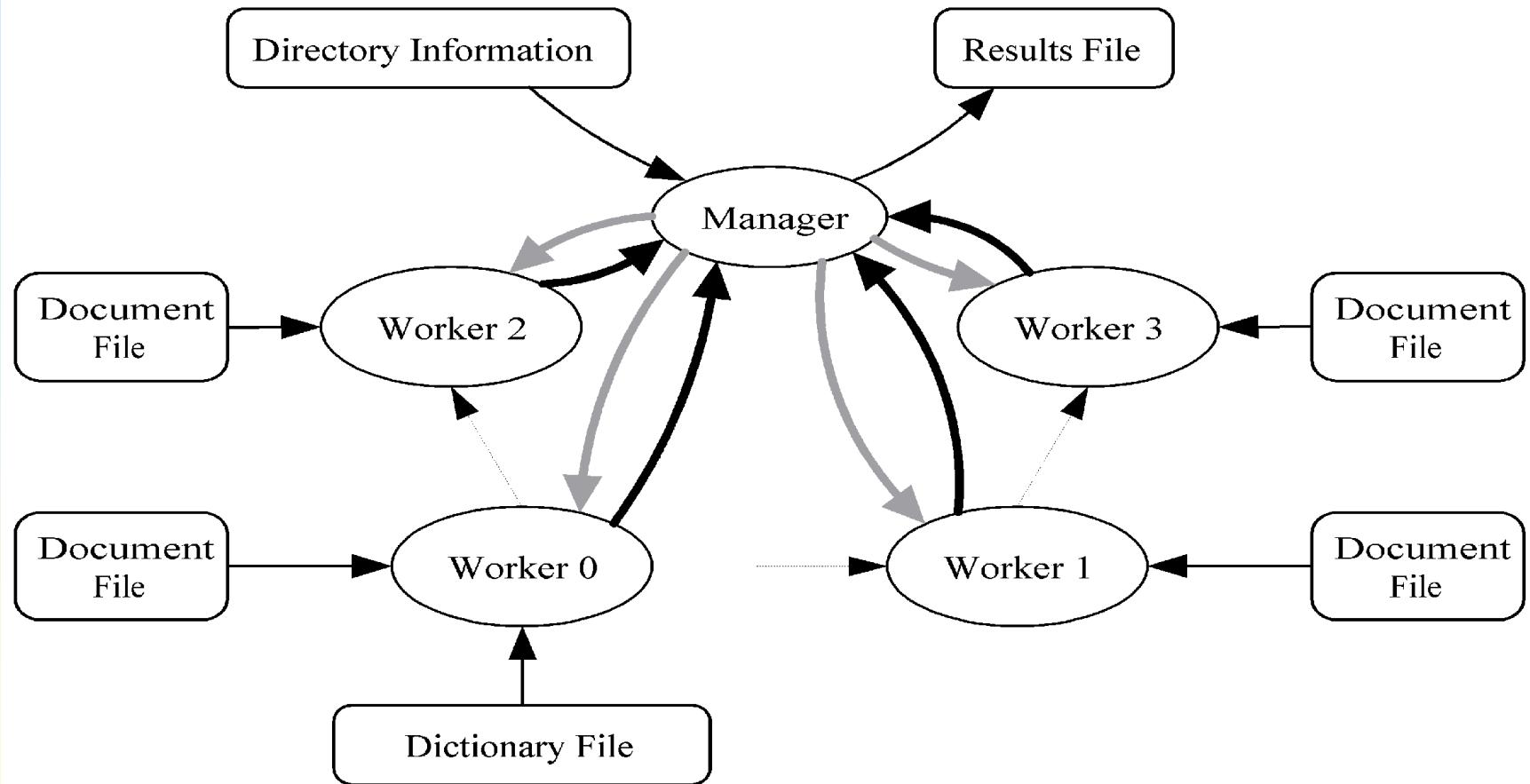
Se arata mai jos cum vor fi stocate numerele
8, 12, 4, 23, 16, 19.



Daca codul hash este bun si k este bine ales atunci listele vor fi scurte

Ca rezultat timpul de cautare este constant

Graful final de procesare



Crearea unui worker numai pentru comunicare

- Dictionarul este schimbat intre ei de workeri
- Pentru a realiza aceasta va trebui realizat un worker dedicat care sa se ocupe de aceasta
- Se poate folosi fie broadcast la toti fie comunicare P2P

- Partitioneaza procesele unui comunicator intr-unul sau mai multe subgrupuri
- Construieste un comunicator pentru fiecare subgrup
- Permite proceselor din fiecare subgrup sa isi realizeze propriile comunicatii colective
- Este necesar sa se permita emisia numai pentru worker in acest caz
- Aceasta abordare nu este unica

Ce face managerul Manager

- **Dupa cum am observat anterior exista trei faze:**
- **Faza 1:**
 - ◆ **1) Gaseste fisierele in structura de director indicata de user**
 - ◆ **2) Primeste dimensiunea dictionarului de la worker 0**
 - ◆ **3) rezerva tabloul bidimensional care este folosit pentru stocarea vectorilor profil .**
- **Faza 2:**
 - ◆ **Trimite documente la workeri si primeste vectorii profil.**
- **Faza 3:**
 - ◆ **Scrie intregul set de vectori profil intr-un fisier.**

Pseudocode pentru manager ce foloseste pipeline pentru alocarea sarcinilor

```
a ← 0 {sarcini asignate}
j ← 0 {sarcini disponibile}
w ← 0 {workeri care asteapta sarcini}
repeat
    if (j > 0) and (w > 0) then
        assigneaza sarcina la worker
        j ← j - 1; w ← w - 1; a ← a + 1
    else if (j > 0) then
        gestioneaza un mesaj primit de la workeri
        incrementeaza w
    else
        ia alta sarcina
        incrementeaza j
    endif
until (a = n) and (w = p)
```

Referinte

- Quang Hieu Vu, Mihai Lupu, Beng Chin Ooi, Peer-to-Peer Computing Principles and Applications, Springer Heidelberg, 2010
- Ralf Steinmetz Klaus Wehrle, Peer-to-Peer Systems and Applications, Springer Heidelberg, 2005
- Dmitry Korzun, Andrei Gurkov, Structured Peer-to-Peer Systems - Fundamentals of Hierarchical Organization, Routing, Scaling, and Security, Springer Heidelberg, 2013
- Wolfgang Effelsberg, Ralf Steinmetz, Thorsten Strufe, Benchmarking Peer-to-Peer Systems Understanding Quality of Service in Large-Scale Distributed Systems, Springer Heidelberg, 2013
- Vikas Gupta, Avnish Dass, Harpreet Singh Matharu, Ankur Verma, Yashraj Chauhan, Peer-to-Peer Application Development - Cracking the Code, Hungry Minds, Inc., New York, 2002



Cursul 12

FTDS



Introducere

- Complexitatea sistemelor de calcul creste si o data cu ea sansa de aparitie a unei erori nemascabile atat la nivel hardware cat si software.
- Unele categorii de calcul sunt proiectate ca sa fie tolerante la erori in cazul defectarii unei componente sau macar sa ascunda (mascheze) aparitia acelei erori din punct de vedere al utilizatorului

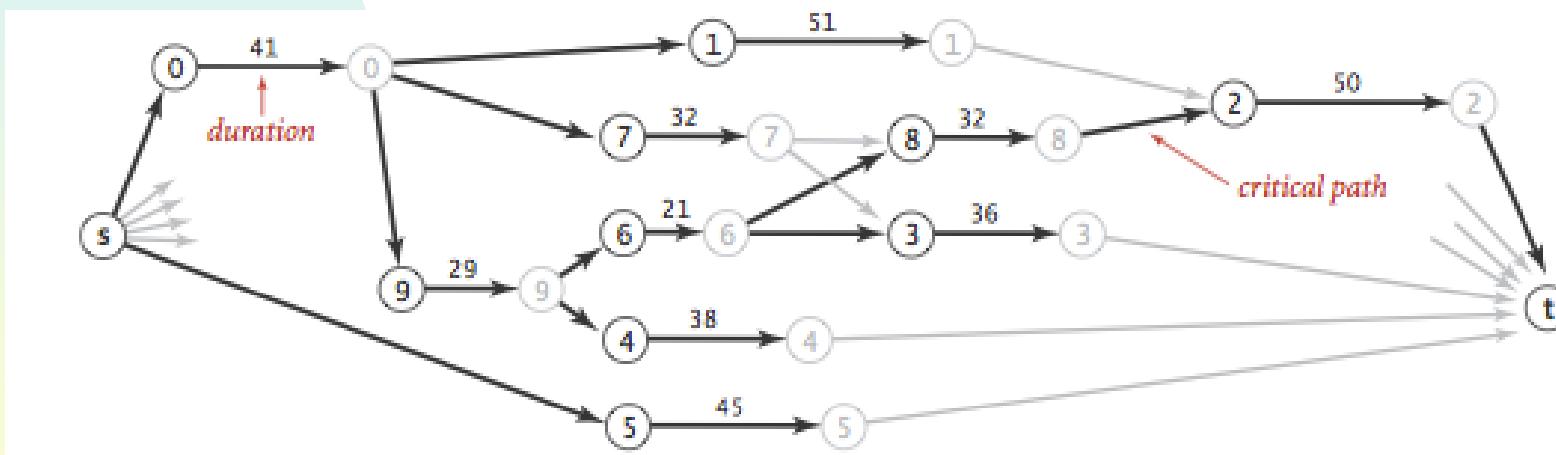
- Sistemele distribuite (indiferent de maniera de interpretare sau implementare a conceptului) se intreprind cu maniera de organizare actuala a majoritatii activitatilor unei societati mai avansate tehnologic.
- Discutiile de pana acum au pus in evidenta caracterul discret al imaginii de ansamblu pentru un astfel de sistem de unde rezulta faptul ca este destul de dificil si costisitor (dpdv al resurselor suplimentare folosite) sa se proiecteze un sistem distribuit tolerant la erori.

Servere, servicii si relatia lor de dependenta

- UN serviciu poate fi definit ca o colectie de operatii a caror executie poate fi declansata (implicit) de trecerea unei cuante de timp sau explicit la cererea beneficiarului.
- Un server poate fi definit ca entitatea care executa suita de operatii folosite in definirea conceptului anterior

Relatia de dependenta

- Un server u poate depinde de un server r daca corectitudinea comportamentului lui u depinde de corectitudinea comportamentului lui r.



Clasificarea caderilor

- **Eroare prin omisiune:** cand serverul nu raspunde la o intrare
- **Eroare temporală :** serverul raspunde corect dar nu in timpul necesar (cel din specificatii)
 - ◆ **Early timing failures** (caderi precoce)
 - ◆ **Late timing failure** (de performanta) prea tarziu

- Un raspuns eronat apare atunci cand serverul
 - ◆ Ofera o valoare incorrecta (value failure)
 - ◆ Starea in care se afla (din FSM) nu e cea corecta (state transition failure)
- Daca dupa prima aparitie a omisiunii raspunsului serverul nu va mai raspunde nici la celelalte cereri pana cand acesta este repornit atunci avem o cadere catastrofala (crash failure)

- Functie de starea serverului dupa repornirea (in cazul unui crash) putem avea:
 - ◆ **Amnezia crash:** serverul se restarteaza dintr-o stare initiala care nu are legatura cu intrarile dinaintea caderii (de ex. reboot os)
 - ◆ **Pause crash:** serverul se reporneste exact in starea anterioara producerii caderii
 - ◆ **Halting crash:** serverul nu mai porneste (chiar dupa un restart soft)

De exemplu

- Fie un client u care realizeaza o cerere de serviciu sr printr-un canal de comunicatie I catre un server r.
- Fie d timpul maxim necesar ca I sa transporte sr catre r
- Fie p timpul maxim necesar lui r pentru receptionare, procesare si emiterea raspunsului catre cererea sr.
- Daca proiectantul lui u cunoaste faptul ca linia I are erori de omisiune (de ex UDP) dar nu si de performanta rezulta ca daca nu primesc un raspuns la o cerere in $2^*(d+p)$ unitati temporale (ticks, secunde si diviziuni, etc) atunci voi avea o eroare.

- O posibilitate are fi ca dupa scuregerea acestui timp p noua cerere **sr'** sa fie emisa.
- In acest caz nu este necesara mentinerea locala a unor date pentru a putea discerne daca raspunsul primit este pentru cererea **sr** sau **sr'**.
- Daca insa I are si probleme de performanta atunci aceste informatii sunt necesare.

Propagarea ierarhica a erorilor si mascarea acestora

- Comportamentul la erori poate fi clasificat numai daca se lucreaza relativ abstract si se tine cont si de o serie de specificatii ale serverului
- Daca un server depinde de alte servere de nivel mai scazut atunci o cadere pe un nivel inferior poate conduce la o serie de caderi in cascada spre nivelele superioare

- De exemplu o daca stratul de date (immediat peste stratul fizic) foloseste un cod corector de minim 2 erori si sunt doi biti eronati din nivel fizic atunci mesajul va fi eliminat si eroarea este clasificata ca o eroare prin emisiune
- Daca un server u depinde de o resursa r a carui semantica de failure este oarecare atunci prin tranzitivitate su u o va avea la fel. O exceptie poate aparea in situatia in care u are la dispozitie o serie de metode pentru a verifica corectitudinea rezultatelor furnizate de r.

Mascarea erorilor prin utilizarea unor grupuri de servere

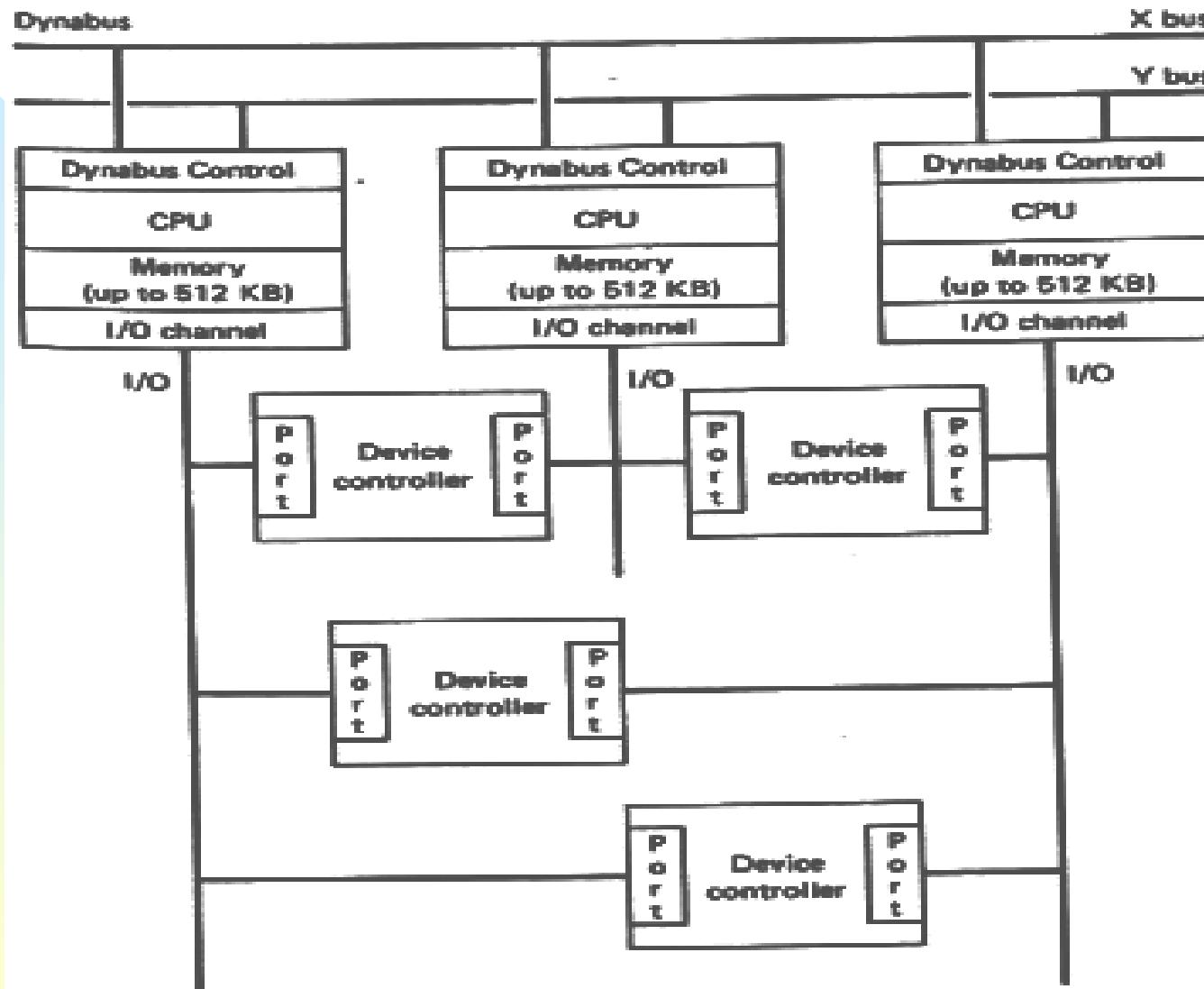
- Pentru a asigura ca un serviciu ramane disponibil clientilor in ciuda caderilor care pot sa apară se poate implementa respectivul serviciu ca un grup redundant independent dpdv fizic de servere.
- Ca rezultat grupul va masca caderea unui membru **m** prin suplinire automata
- UN grup de servere care este capabil sa mascheze caderea a k membri se numeste k – tolerant la erori

Abordari hardware

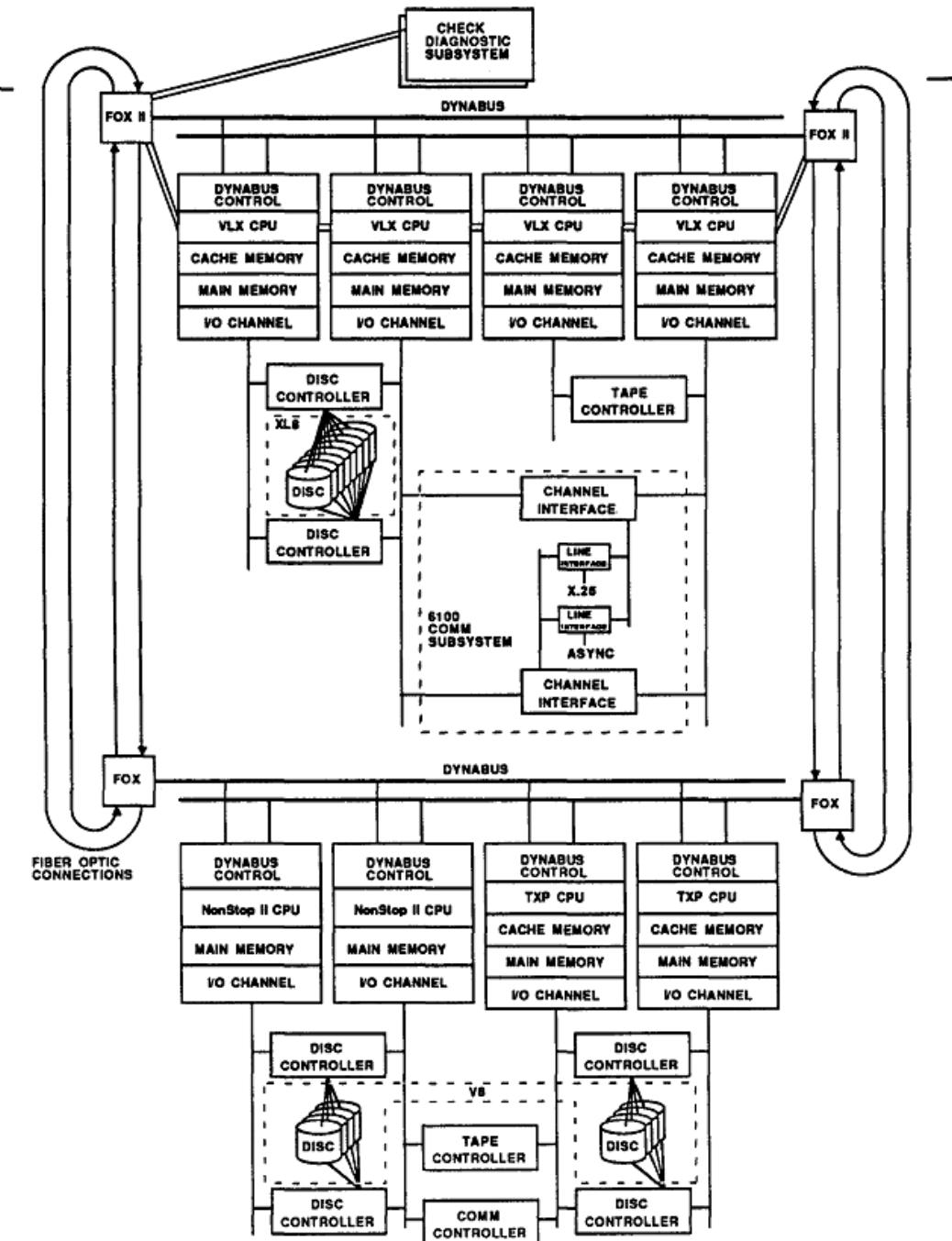
- Majoritatea arhitecturilor produse au o redundanta minimala de multe ori inexistentă, la acest nivel, datorita constrangerilor economice.
- IN momentul in care toleranta la erori este justificata (dpdv economic) se poate trece la replicari judicios alese pentru diverse componente.

- Aceste componente se grupeaza in doua categorii
 - ◆ Care necesita prezenta unui personal autorizat la inlocuire
 - ◆ Si care nu (adica le schimba userul fara pregatire speciala)
- Caracteristicile si complexitatea unitatilor modulare sunt strans legate de granularitatea masinii.

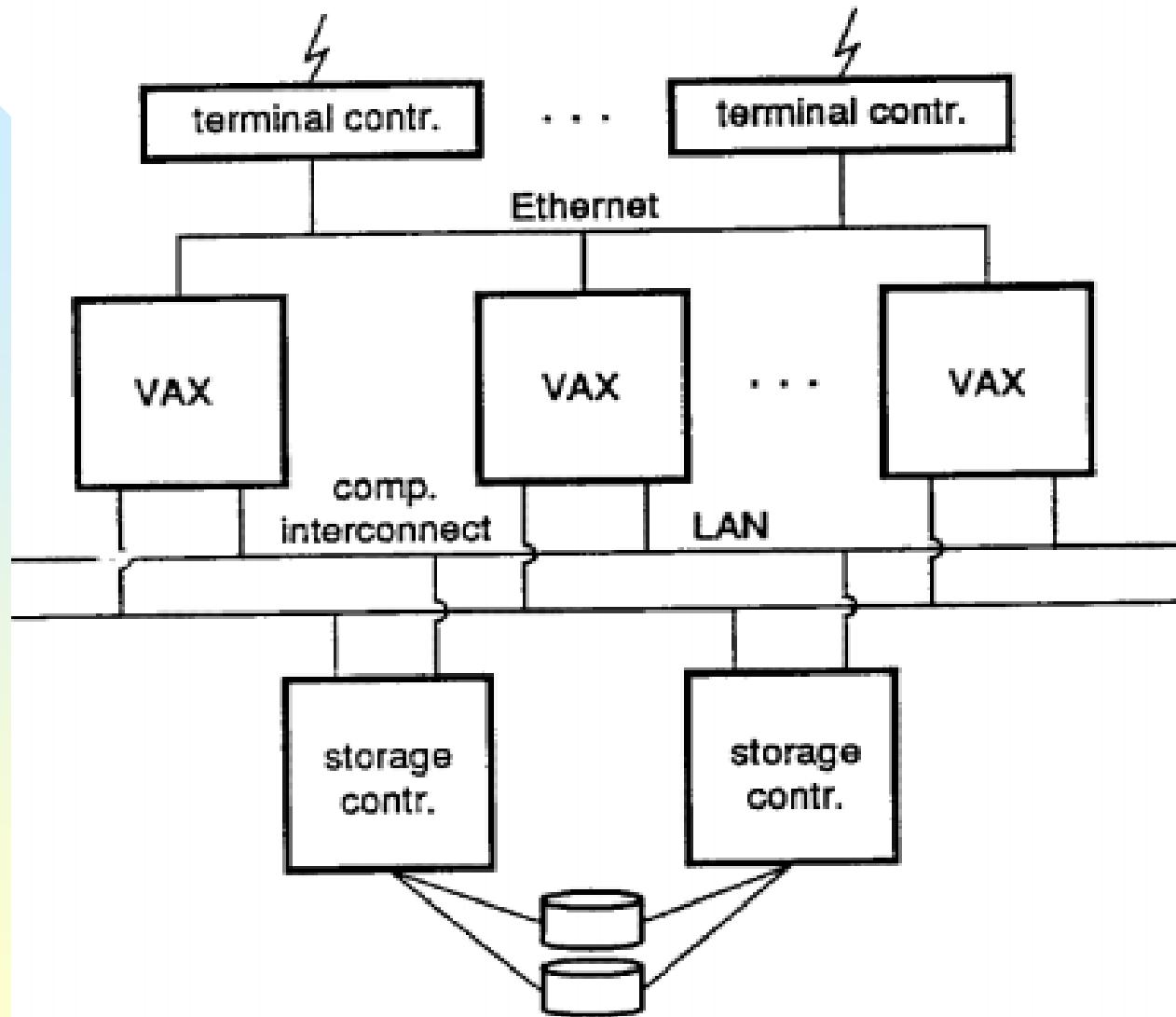
Arhitectura cu procesoare in tandem



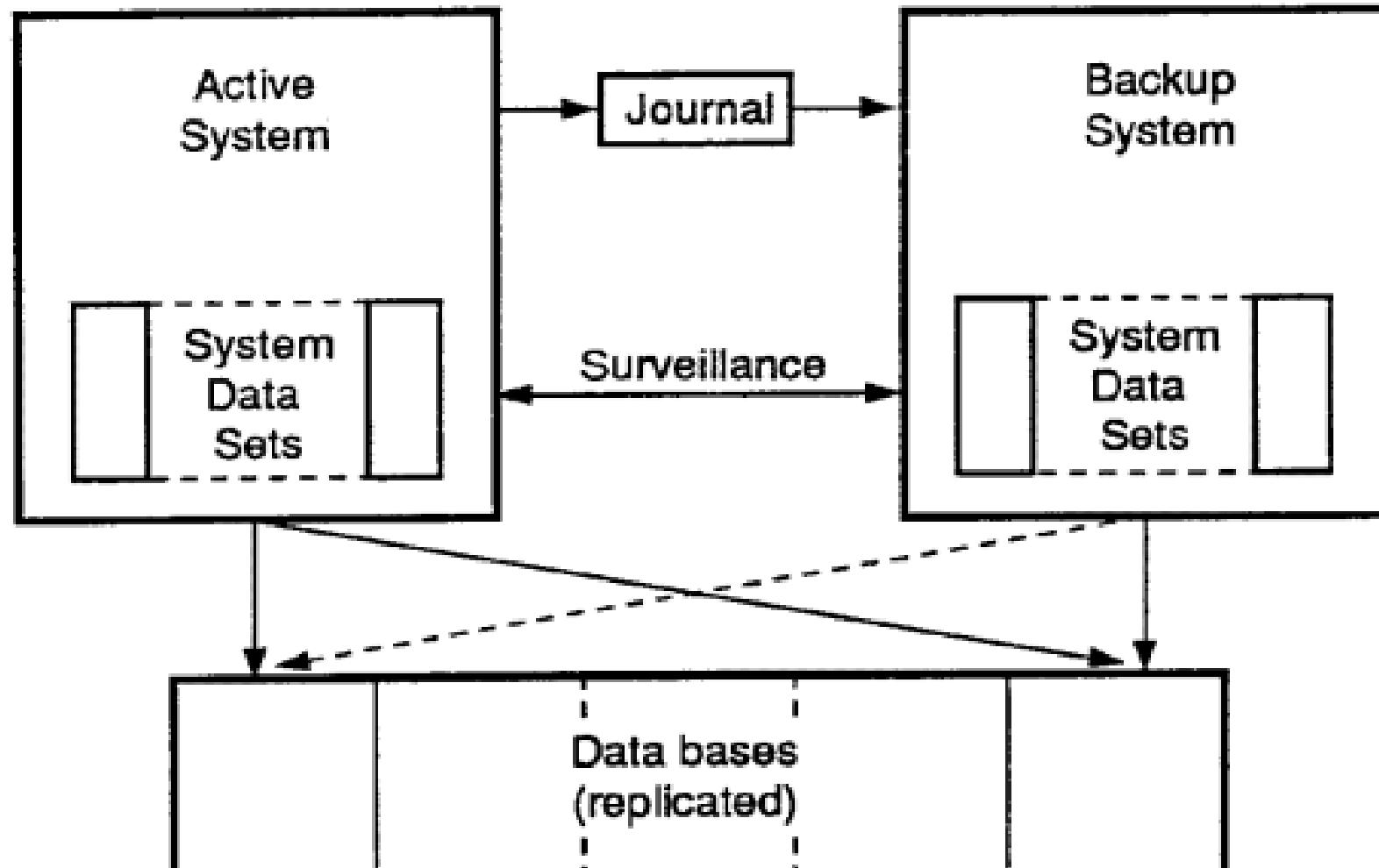
Arhitectura tandem din '76



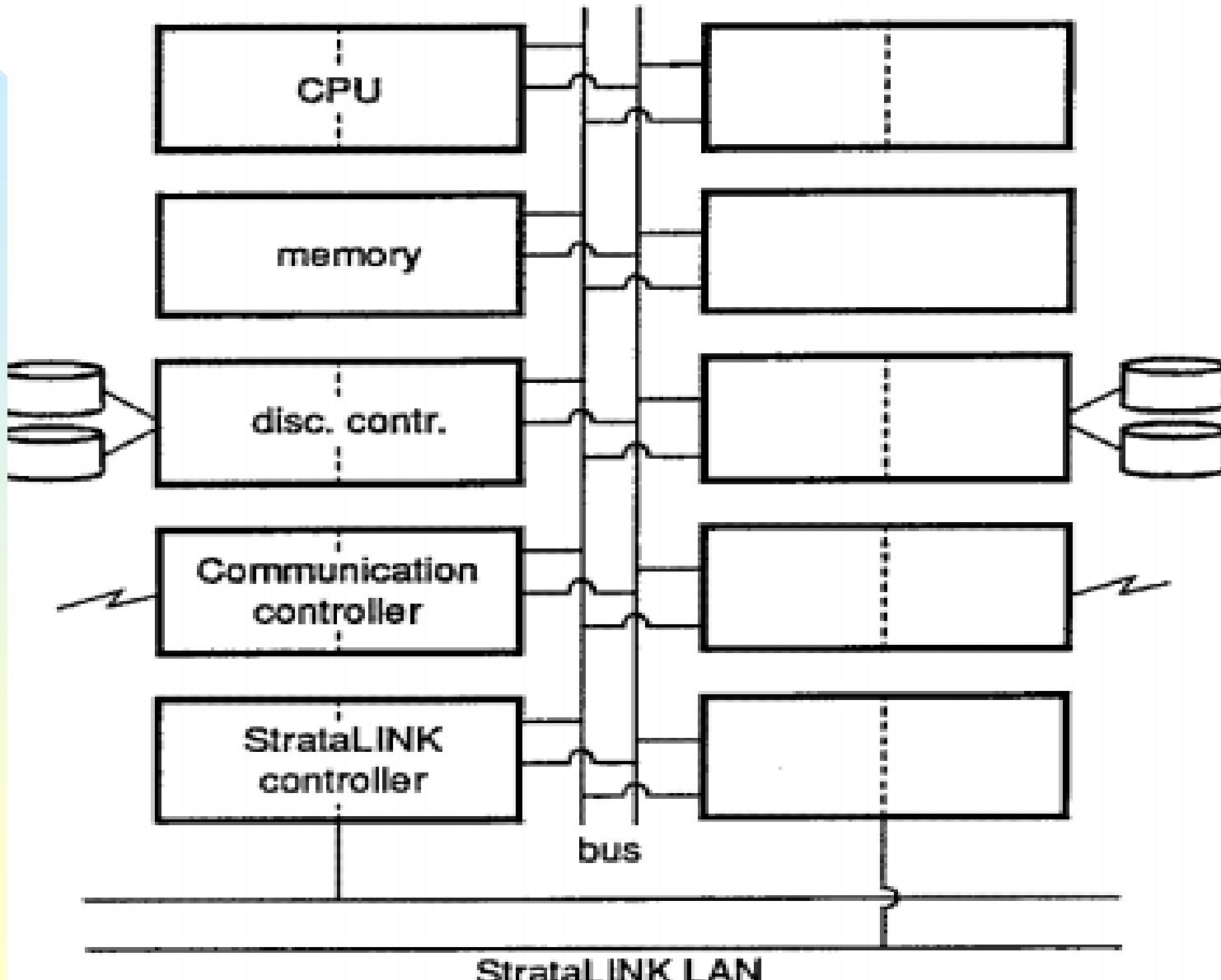
Arhitectura VAX cu cluster de procesoare



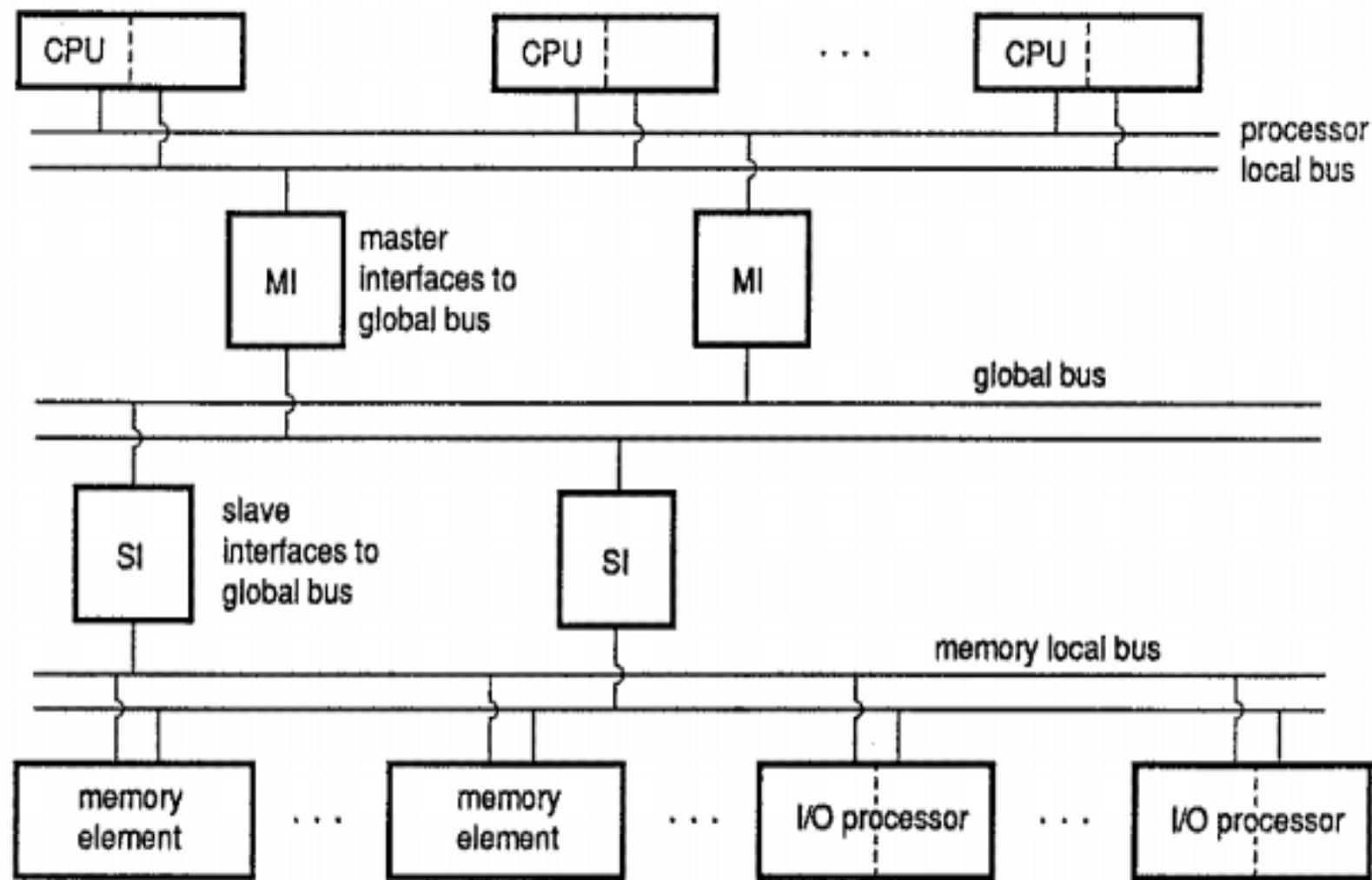
IBM XRF/1



Arhitectura Stratus



Arhitectura Sequoia /1



Sincronizarea membrilor unui grup tolerant la erori

Pentru orice serviciu politica de sincronizare a grupului de servere care il ofera descrie gradul local de sincronizare existenta intre acestea

Sincronizare stransa

- Mai este numita si sincronizare activa sau mascata.
- In acest caz toate cererile de servicii se executa in paralel si trec prin aceleasi stari (FSM comun)
- Daca apare o eroare majoritatea va vota iar rezultatul votului va fi considerat raspunsul valid

Sincronizare relaxata

- I se mai spune si redundanta dinamica sau in asteptare.
- In aceasta situatie membrii sunt clasificati functie de cat de sincronizati sunt la serviciul curent.
- Serverul cu gradul cel mai mare este numit primar.

Puncte de salvare a executiei si refacerea

- Exista mai multe abordari pentru refacerea prin restaurarea unei stari anterioare (rollback recovery)
 - Restarurarea sistemului intr-o stare stabila de dinaintea caderii
 - Obtinerea tolerantei la erori prin salvarea periodica a starii proceselor in timpul unei executii fara caderi
 - Analiza unei aplicatii distribuite ca fiind o colectie de procese care comunicapeste o retea
- Punctele de salvare/verificare a executiei (Checkpoints)
 - Starile salvate a unui proces

- Daca fiecare proces isi face salvarele de stare independent atunci sistemul nu poate evita aparitia fenomenului domino
 - Aceasta abordare este numita checkpointing independent sau necordonat
- Tehnici pentru evitarea efectului domino
 - Utilizarea refacerii starii bazate pe checkpointing coordonat
 - ☞ In acest caz procesele isi coordoneaza salvarele de stare astfel incat sa se mentina o stare globala consistenta a sistemului
 - Utilizarea refacerii bazate pe un checkpointing care tine cont de comunicarea intre procese
 - ☞ Pentru fiecare proces punctele de salvare sunt bazate si pe informatiile de la celelalte procese

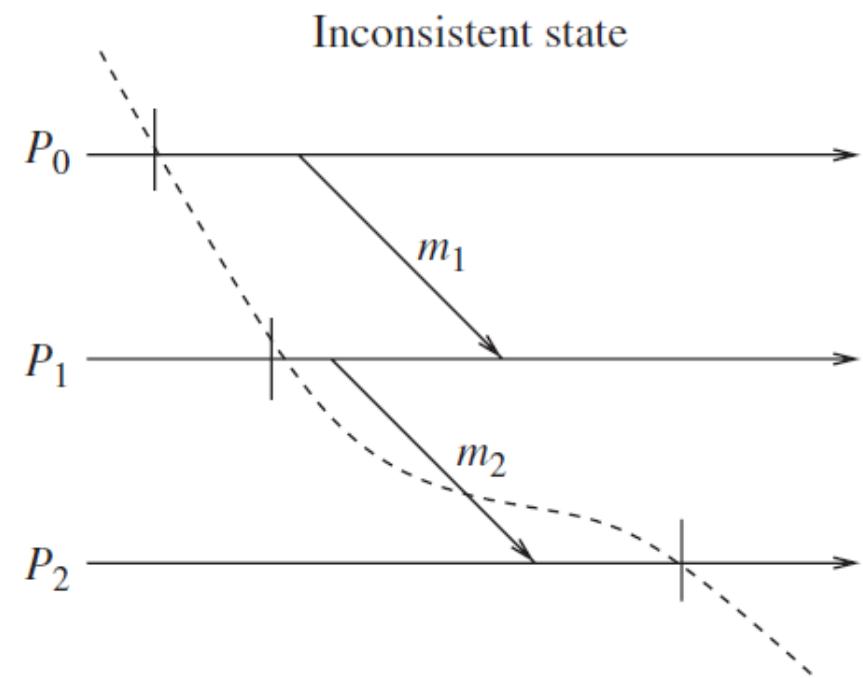
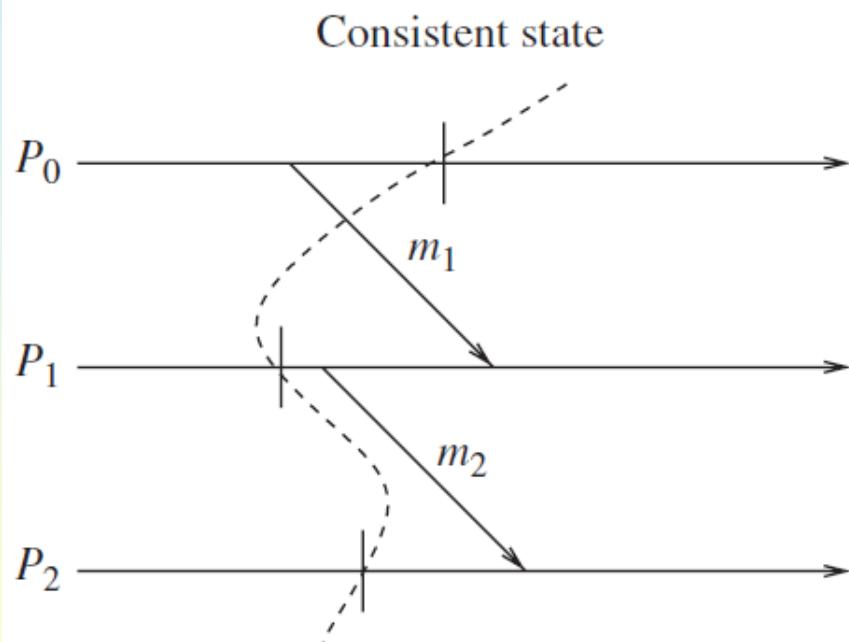
Punct local de salvare (checkpoint)

- Toate procesele isi salveaza starile locale la anumite perioade de timp
- Un punct de salvare local este o radiografie (snapshot) a starii procesului dintr-o instantă anume
- Presupuneri
 - Un proces stocheaza toate punctele de refarere locale intr-un tabel persistent
 - Un proces este capabil de restaurare catre oricare din punctele de restaurare salvate (existente)
- $C_{i,k}$
 - Este al k -lea punct de salvare pentru procesul P_i
- $C_{i,0}$
 - Procesul P_i realizeaza prima salvare a starii $C_{i,0}$ inainte de a-si incepe executia

Stari stabile

- O stare globala a unui sistem distribuit
 - O colectie a tuturor starilor individuale pentru toate procesele participante precum si starea canalelor de comunicatii
- O stare globala stabila
 - Este o stare globala care poate aparea in timpul unei executii lipsite de erori a unui calcul distribuit
 - Daca starea unui proces reflecta si mesajul receptionat atunci si starea corespunzatoare emitatorului corespunzator trebuie sa reflecte si trimiterea acestui mesaj
- UN punct de salvare globala
 - Este format dintr-un set de puncte de salvare locala cate unul pentru fiecare proces
- Un punct de salvare global si consistent

Exemple de stari consistente



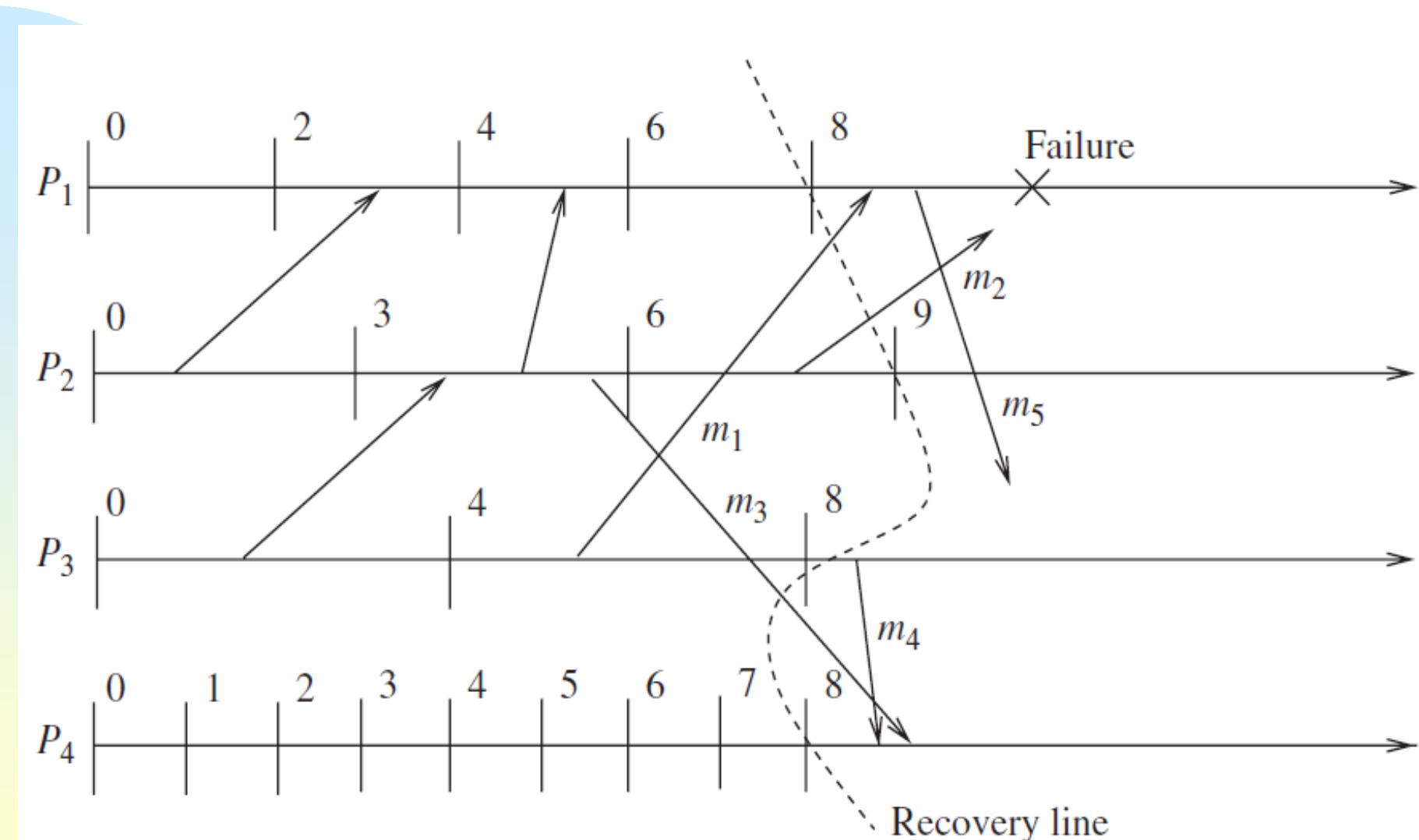
Interactiuni cu lumea exterioara

- Un sistem distribuit interactioneaza adeseori cu lumea de afara pentru a receptiona date de intrare sau a livra rezultatele calculelor (sub orice forma)
- Procesul lumii de afara (Outside World Process (OWP))
 - Este un proces special care interactioneaza cu resptul sistemului prin transfer de mesaje
- O abordare comună
 - Orice mesaj de intrare este salvat (in zona de persistență) înainte de a se permite procesarea lui
- Simbolul “||”
 - Reprezinta o interactiune cu lumea de ad=fara pentru a trimite catre aceasta rezultatul unui calcul

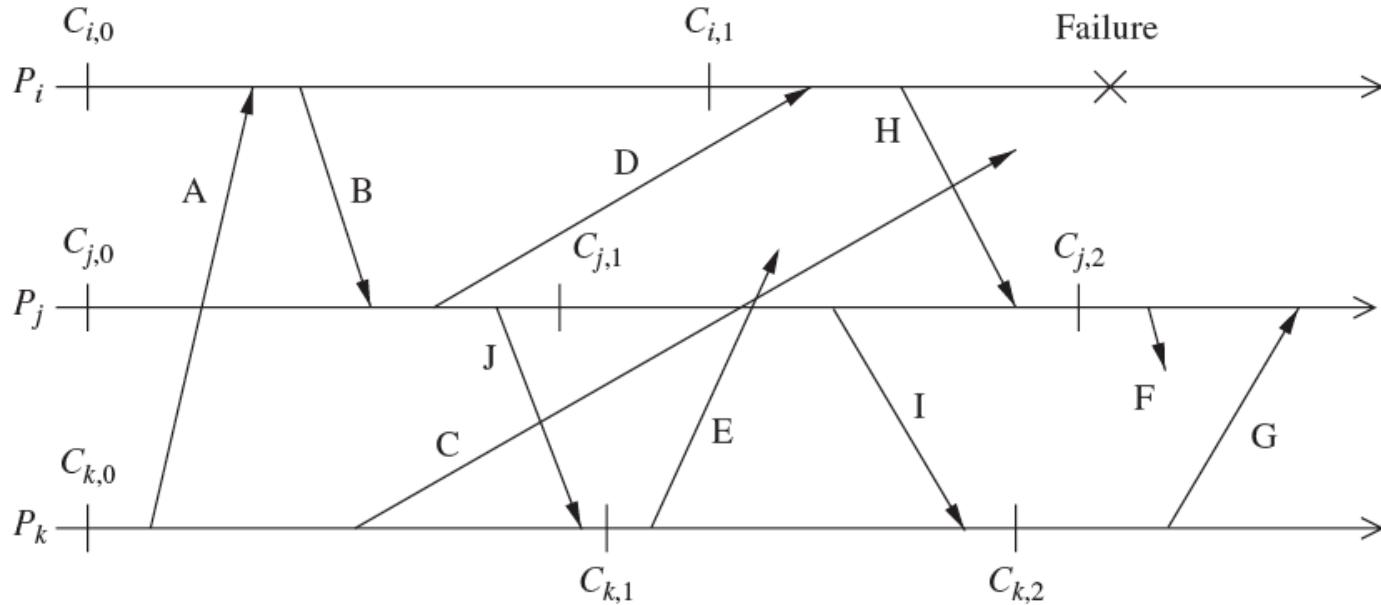
Mesajele

- Mesaje in tranzit
 - Mesajele care au fost trimise dar nu au fost inca receptionate la destinatie
- Mesaje pierdute
 - Mesajele care au fost trimise dar receptia lor este anulat de la restaurarea sistemului
- Mesaje intarziante
 - Mesajele ale caror primire nu a fost inregistrata deoarece procesul receptor a fost fie inchis fie mesajul a venit dupa restaurare
- Mesaje orfane
 - Mesajele care au fost inregistrate ca receptionate dar nu au fost inregistrate ca trimise
 - Teoretic nu apar daca sistemul este resaturat la o stare globala consistenta
- Mesaje duplicate
 - Apar datorita inregistrarii mesajelor si reparitia lor dupa restaurarea procesului

Exemple message



Refacerea după eroare



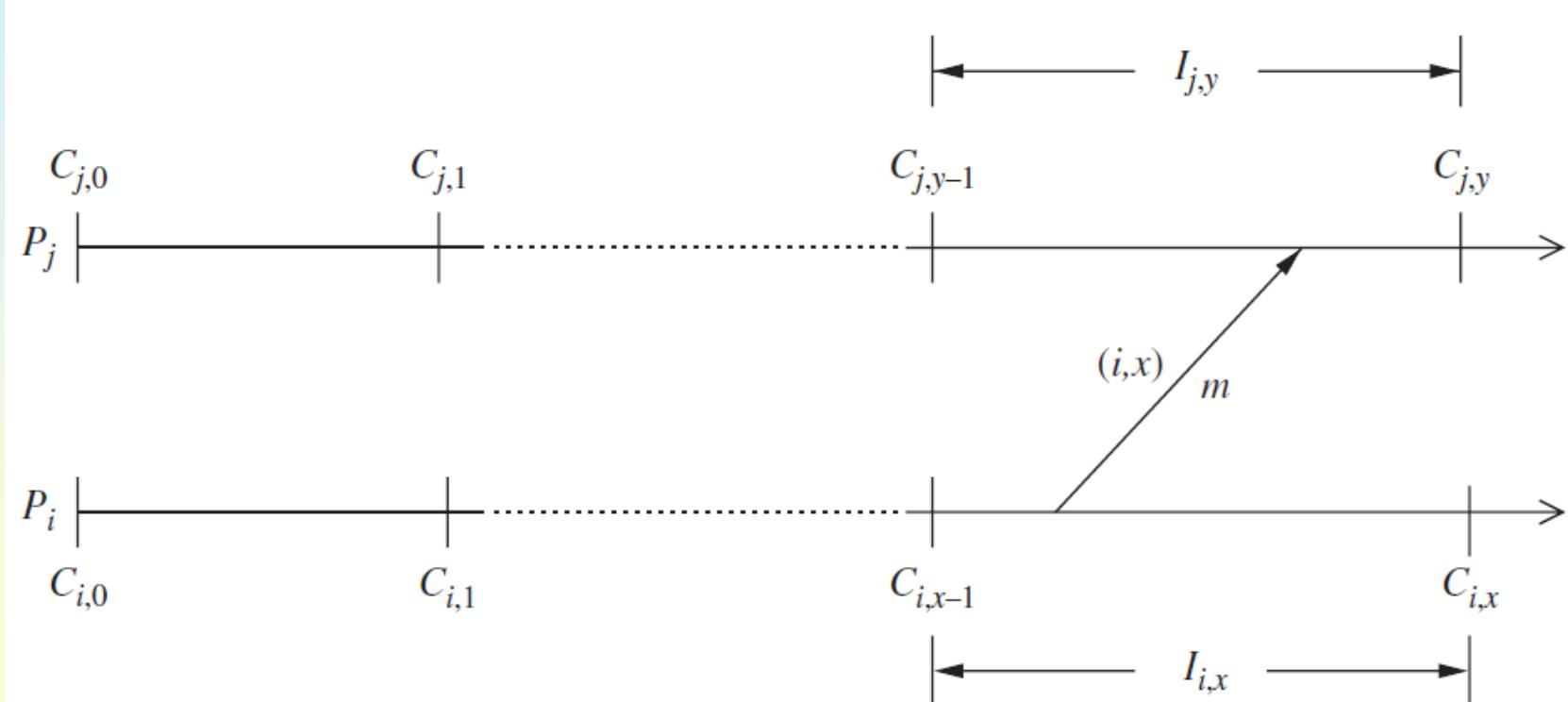
- Checkpoint-uri : $\{C_{i,0}, C_{i,1}\}$, $\{C_{j,0}, C_{j,1}, C_{j,2}\}$, and $\{C_{k,0}, C_{k,1}, C_{k,2}\}$
- Mesaje : A - J
- Starea globală consistentă restaurată este : $\{C_{i,1}, C_{j,1}, C_{k,1}\}$

Checkpointing necordonat

- Fiecare proces are autonomie de decizie cu privire la momentul salvării stării
- Pro
 - Încarcare minima în cazul executiei
- contra
 - Efectul de domino în timpul restaurării
 - Refacerea după o cadere este lenta deoarece procesele trebuie să facă mai mulți pași pentru a găsi un set coherență de puncte de salvare

Tehnica urmaririi dependentei directe

- Sa presupunem ca fiecare proces P_i isi porneste executia de la punctul initial $C_{i,0}$
- Fie $I_{i,x}$: un interval intre doua puncte de salvare $C_{i,x-1}$ si $C_{i,x}$

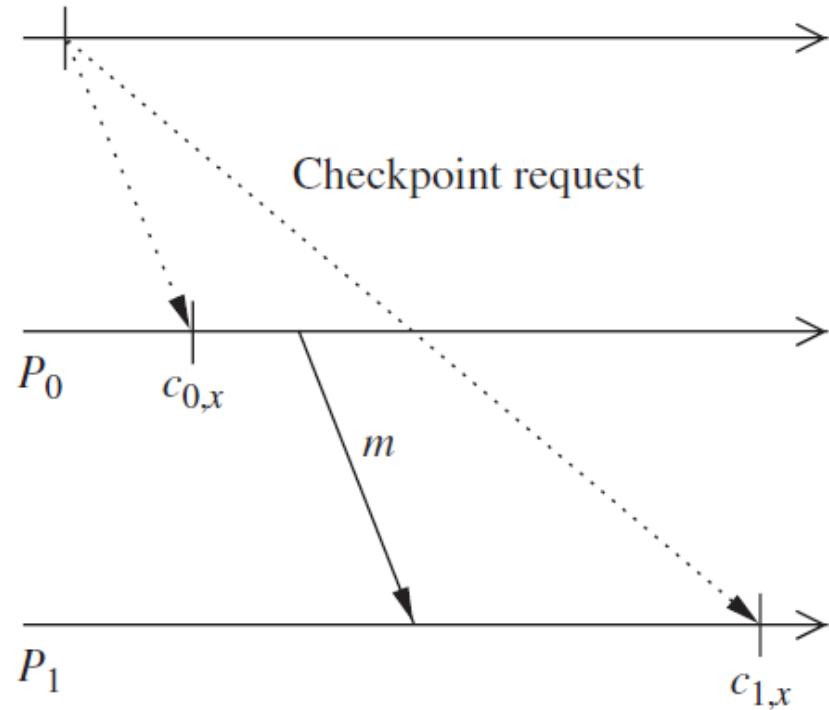


Checkpointing coordonat

- Checkpointing blocant
 - Dupa ce un proces realizeaza un checkpoint local, el va ramane blocat pana cand procesul de salvare este complet. Acest lucru previne aparitia mesajelor orfani
 - Dezavantaje
 - Timpul pierdut cu blocarea in timpul operatiei de salvare a starii
- Checkpointing neblocant
 - Procesele nu se opresc pentru checkpoint
 - Trebuie gasita o metoda de coordonare globala a efectuarii salvarilor pentru a impiedica receptia mesajelor in timpul checkpointing deoarece aparitia lor ar face checkpoint-ul inconsistent.

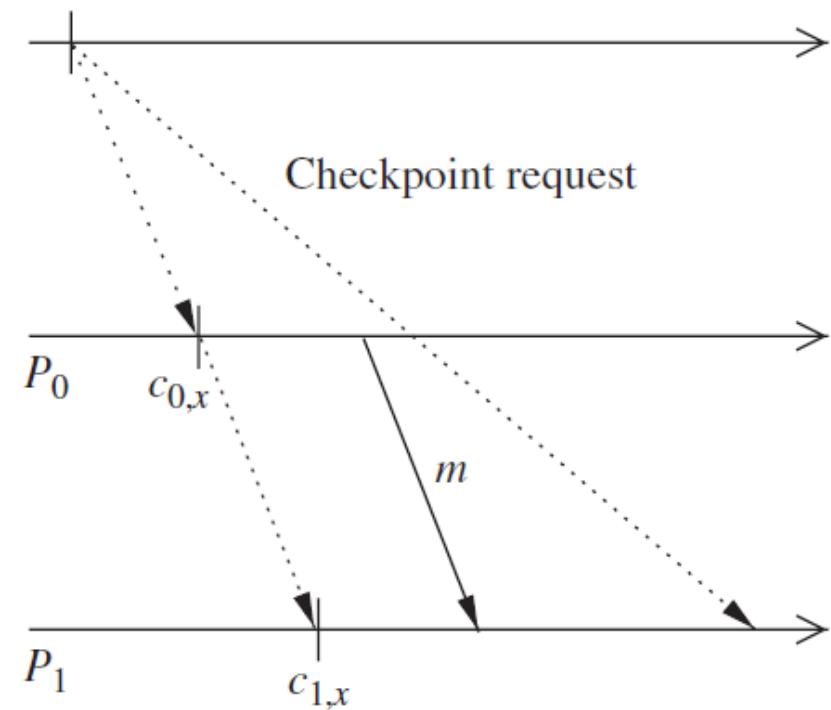
Checkpointing coordinate

Initiator



(a)

Initiator

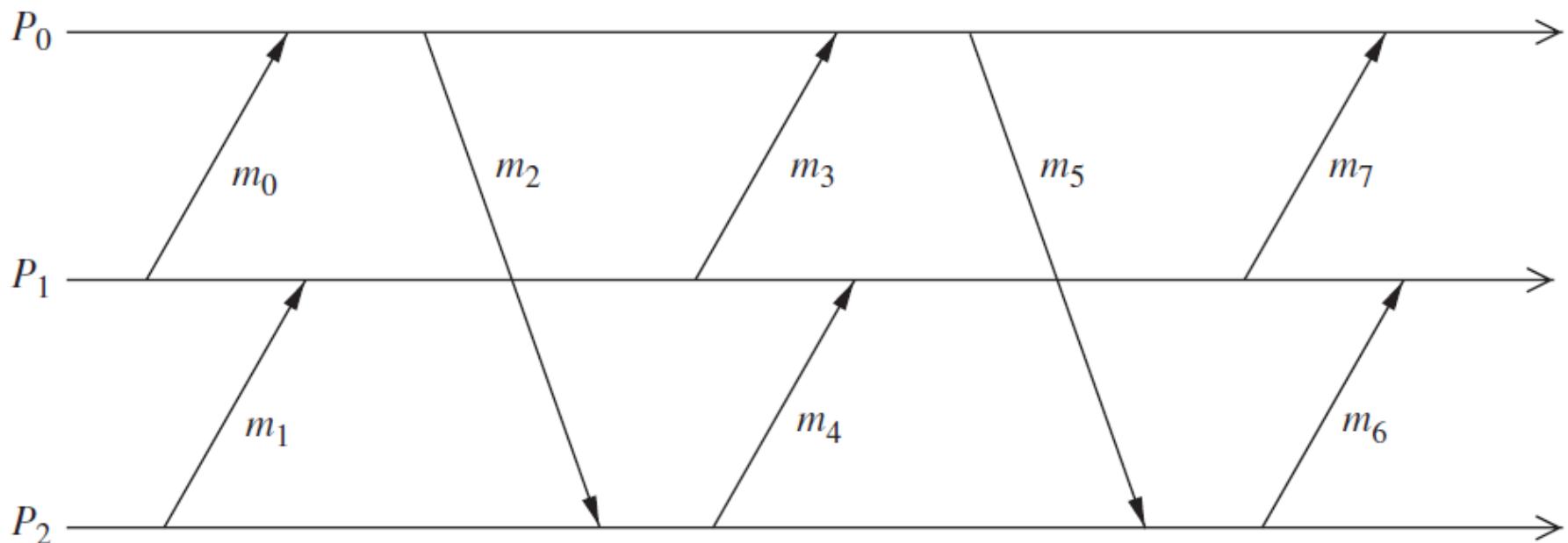


(b)

Checkpointurile induse de comunicatie

- Există două tipuri de checkpoint-uri
 - autonome și fortate
- Checkpointurile induse de comunicatie
fură/inspectează/află informații despre mesajele fiecarei aplicații
- Receptorul fiecarui mesaj aplicație folosește aceasta informație pentru a determina dacă ar fi bine să facă un checkpoint mai devreme decât regulile stabilită global pentru aceasta operatie
- Aceste salvari fortate în avans trebuie realizate înainte de receptia mesajelor

Log-based Rollback Recovery

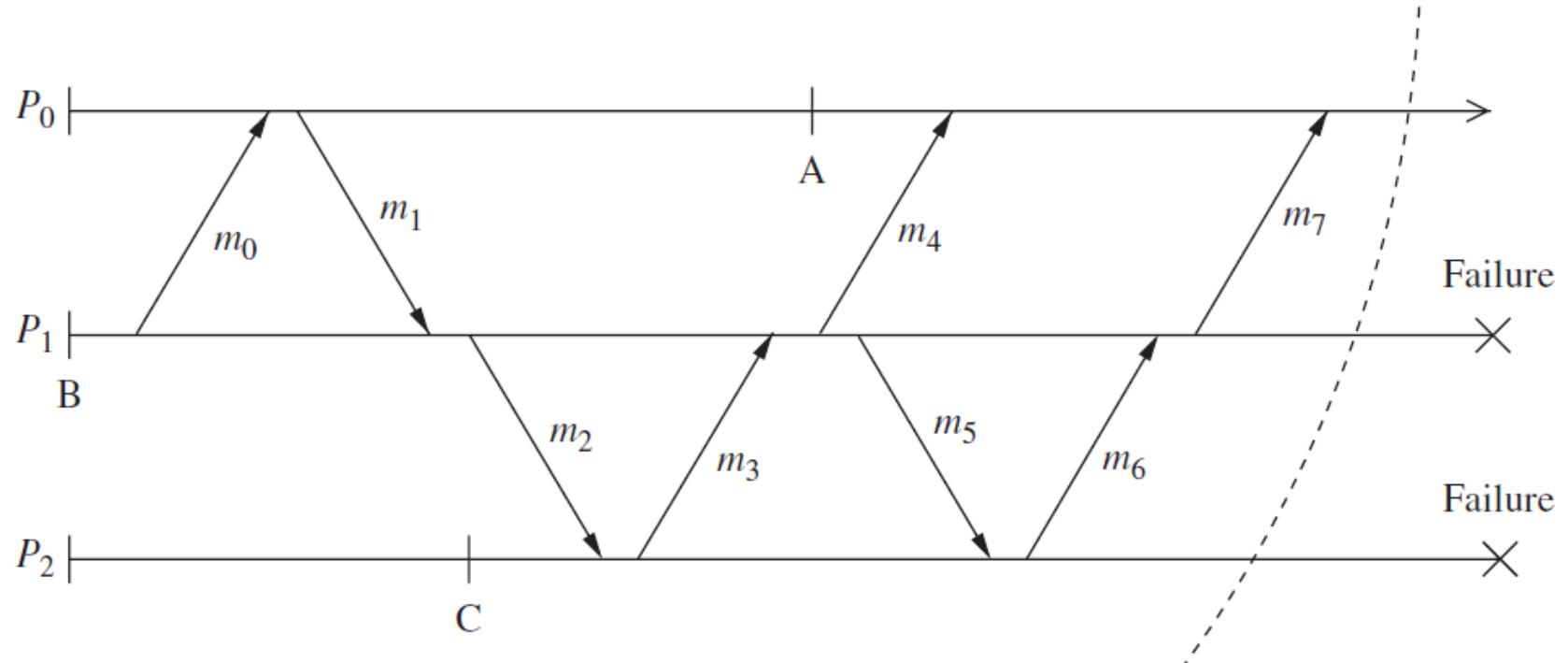


Conditia de consistenta pentru a evita orfanii

- Fie e un eveniment nedeterministic care apare in cadrul procesului p
- **Depend(e)**
 - Setul de procese care sunt afectate de un eveniment nedeterministic e . Acest set consta in p , si orice procese a caror stare depinde de evenimentul e conform relatie s-a “*intamplat mai inainte*” a lui Lamport
- **Log(e)**
 - Un set de procese care au inregistrat o copie a lui e (determinantul lui) in memoria lor volatila
- **Stable(e)**
 - Un predicat care este true daca determinantul lui e este inregistrat in zona de persistenta
- **Atunci conditia ca s nu existe orfani se poate descrie ca**
$$\forall(e) : \neg \text{Stable}(e) \Rightarrow \text{Depend}(e) \subseteq \text{Log}(e)$$

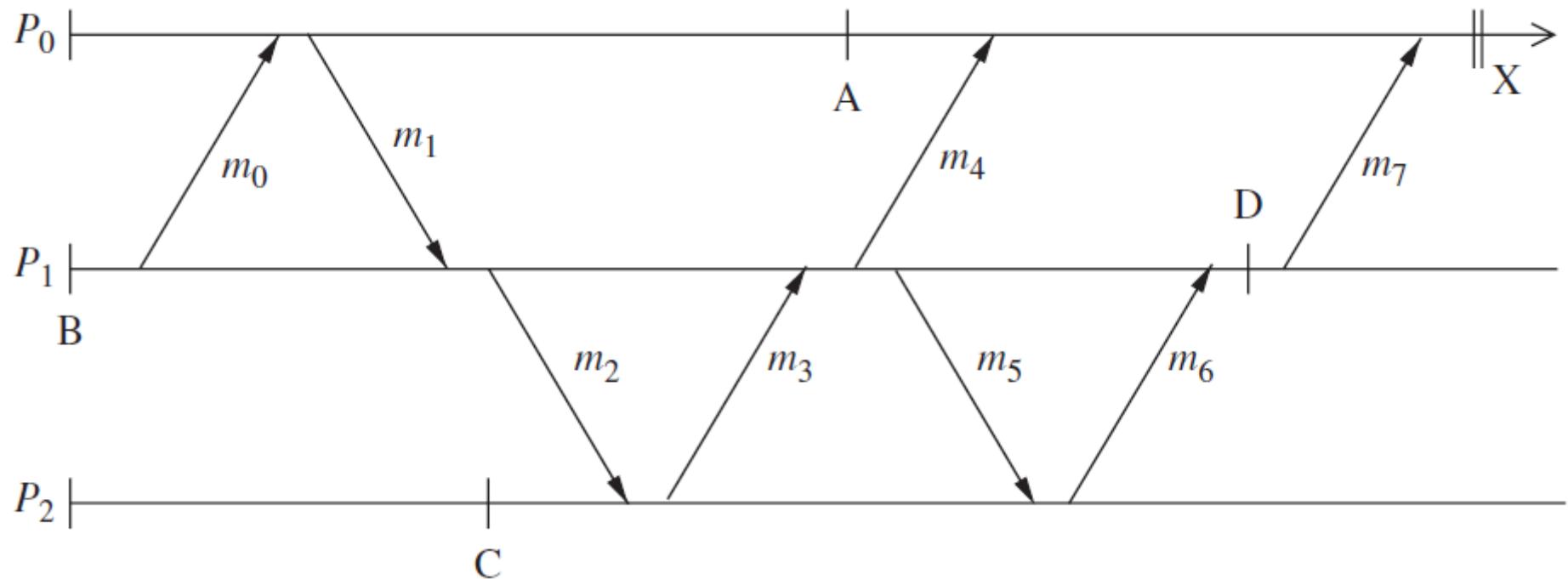
Inregistrare pesimista - Pessimistic Logging

Maximum recoverable state

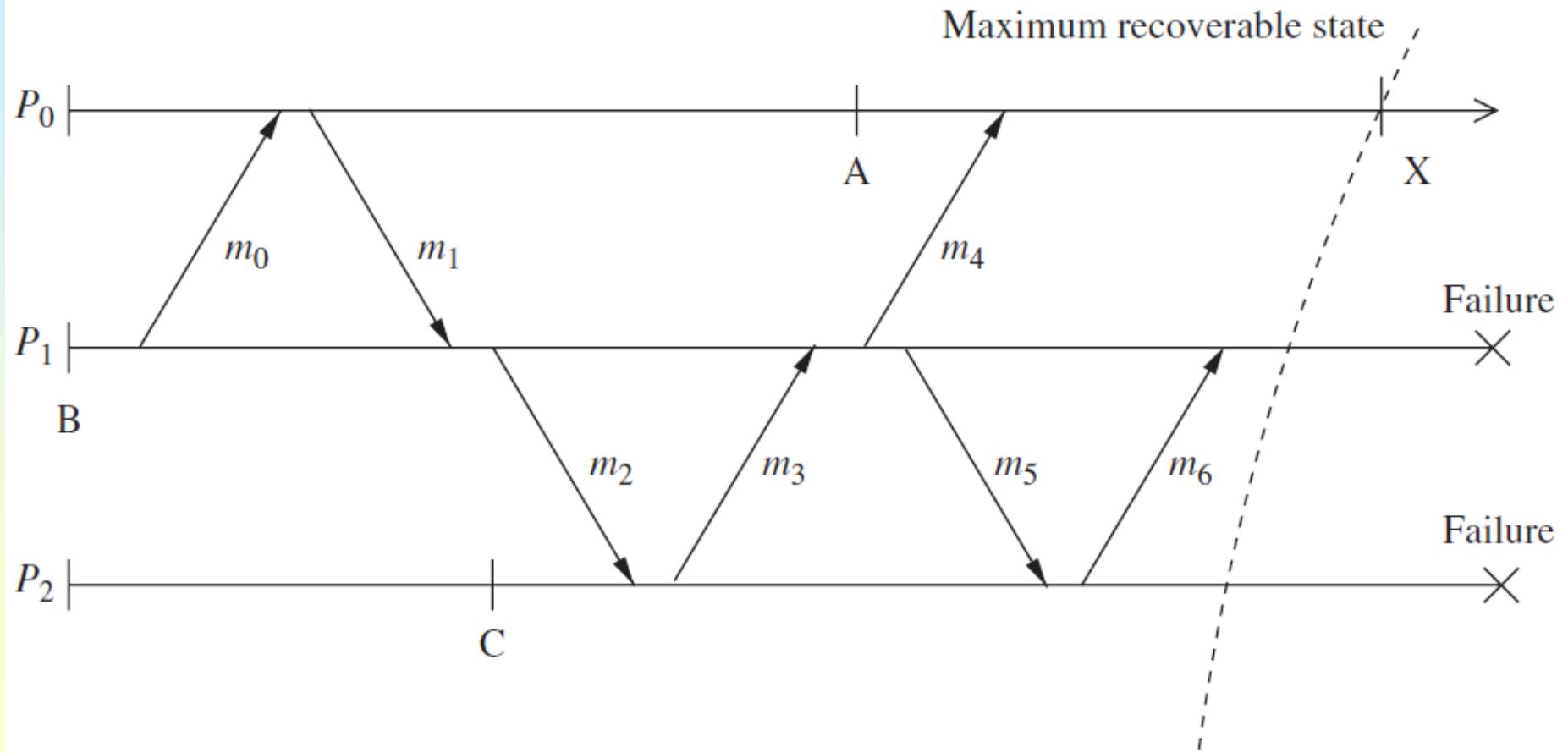


- Sa presupunem ca procesele P_1 si P_2 cad ca in figura si apoi repornesc din punctele de salvare B si C, si apoi genereaza aceasi secventa de mesaje (pe baza inregistrarilor determinant) ca inainte de cadere
- O data ce restaurarea este completa ambele procese vor fi consistente cu starea lui P_0 care include si receptionarea mesajului m_7 de la P_1

Inregistrare optimista - Optimistic Logging



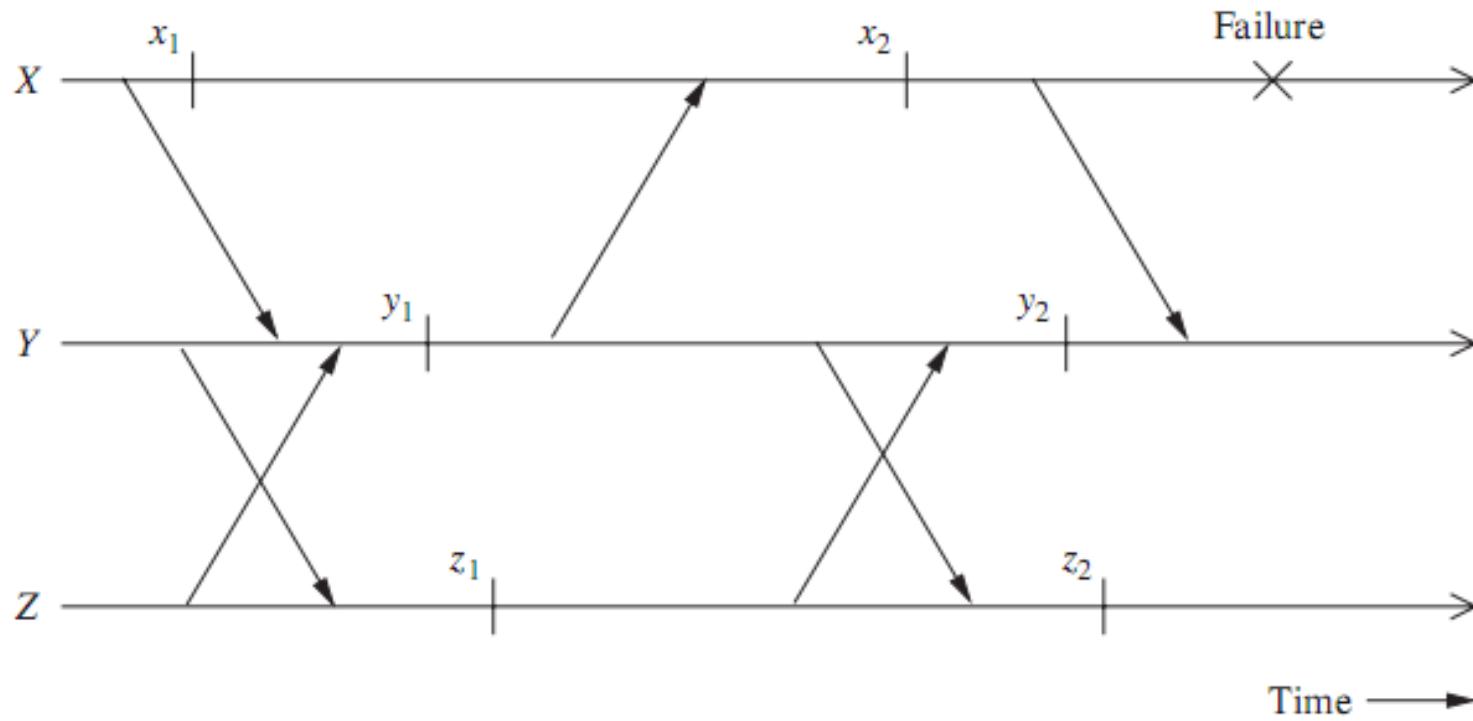
Causal Logging



Algoritmul Koo-Toueg pentru salvare coordonata

- Reprezinta o tehnica pentru salvarea coordonata care garanteaza ca punctele de salvare sunt consistente evita efectul de domino sau blocajele in timpul refacerii
- Este compus din doua faze
 - ◆ Faza-Algoritm de salvare
 - ◆ Faza-Algoritm restaurare

- Algoritmul de salvare are si el doua faze/etape
 - ☞ Procesul care initiaza aceasta faza incearca sa salveze un punct de refacere si in acelasi timp cere si celorlalte procese sa faca acelasi lucru
 - ☞ Fiecare proces nu mai poate emite mesaje dupa ce incearca sa salveze un punct de restaurare
 - ☞ Toate procesele vor ajunge in final la aceasi decizie operez salvarea sau o anulez
 - ☞ O data terminata aceasta votare/consultare procesul initiator trimit catre celelalte decizia finala si se comporta ca atare



- Correctitudine: se continua dintr-o stare consistenta in caz de cadere
- Optimizare: s-ar putea ca recuperarea/restaurarea sa nu fie completa pentru fiecare proces deoarece unele nu avea motiv (nu s-a schimbat nimic)

Algoritmul Juang-Venkatesan pentru salvare si restaurari asincrone

- Presupuneri: canalele de comunicatie sunt de incredere (reliable), livrarea mesajelor se face dupa ordinea FIFO, buferele sunt infinite, intarzierile in transmiterea unui mesaj exista sunt arbitrate dar finite
- Calculele sau aplicatiile sunt de tip event-driven
- Procesul P aflat in starea **s**, receptioneaza mesajul **m**, il proceseaza, tranziteaza in starea **s'** si emite alte mesaje.
- Deci tupla (**s, m, msgs_sent**) va reprezenta starea lui P
- Exista doua tipuri de jurnale mentinute
 - ◆ Volatil: timpi mici de acces dar se pierde in caz de eroare procesor sau alta eroare catastrofica. Este mutat periodic in zona de persistenta (jurnalul stabil) – write back va zice ceva?
 - ◆ Jurnal stabil: salvat in zona de persistenta

Algoritmul Juang-Venkatesan

Procedura RollBack_Recovery: procesorul p_i executa urmatoarele:

PAS (a)

if procesorul p_i isi revine dupa o cadere then

$CkPt_j \leftarrow$ ultimul eveniment salvat in zona de persistenta

else

$CkPt_j \leftarrow$ ultimul eveniment care a avut loc in p_i ,

 {ultimul eveniment al p_i se poate gasi in memoria temporara sau in zona de persistenta}

end if

PAS (b)

for $k = 1$ to N { N este numarul de procesoare din sistem} do

 for each procesor vecin p_j do

 compute $SENT_{i \rightarrow j}(CkPt_i)$

 send un mesaj $ROLLBACK(i, SENT_{i \rightarrow j}(CkPt_i))$ catre p_j

 end for

 for oricare mesaj $ROLLBACK(j,c)$ receptionat de la vecinul j do

 if $RCVD_{i \leftarrow j}(CkPt_i) > c$ {inseamna ca exista un mesaje orfani} then

 gaseste ultimul eveniment e care satisface $RCVD_{i \leftarrow j}(e) = c$

 {un astfel de eveniment e s-ar putea sa fie ori in memoria temporara ori in cea persistenta}

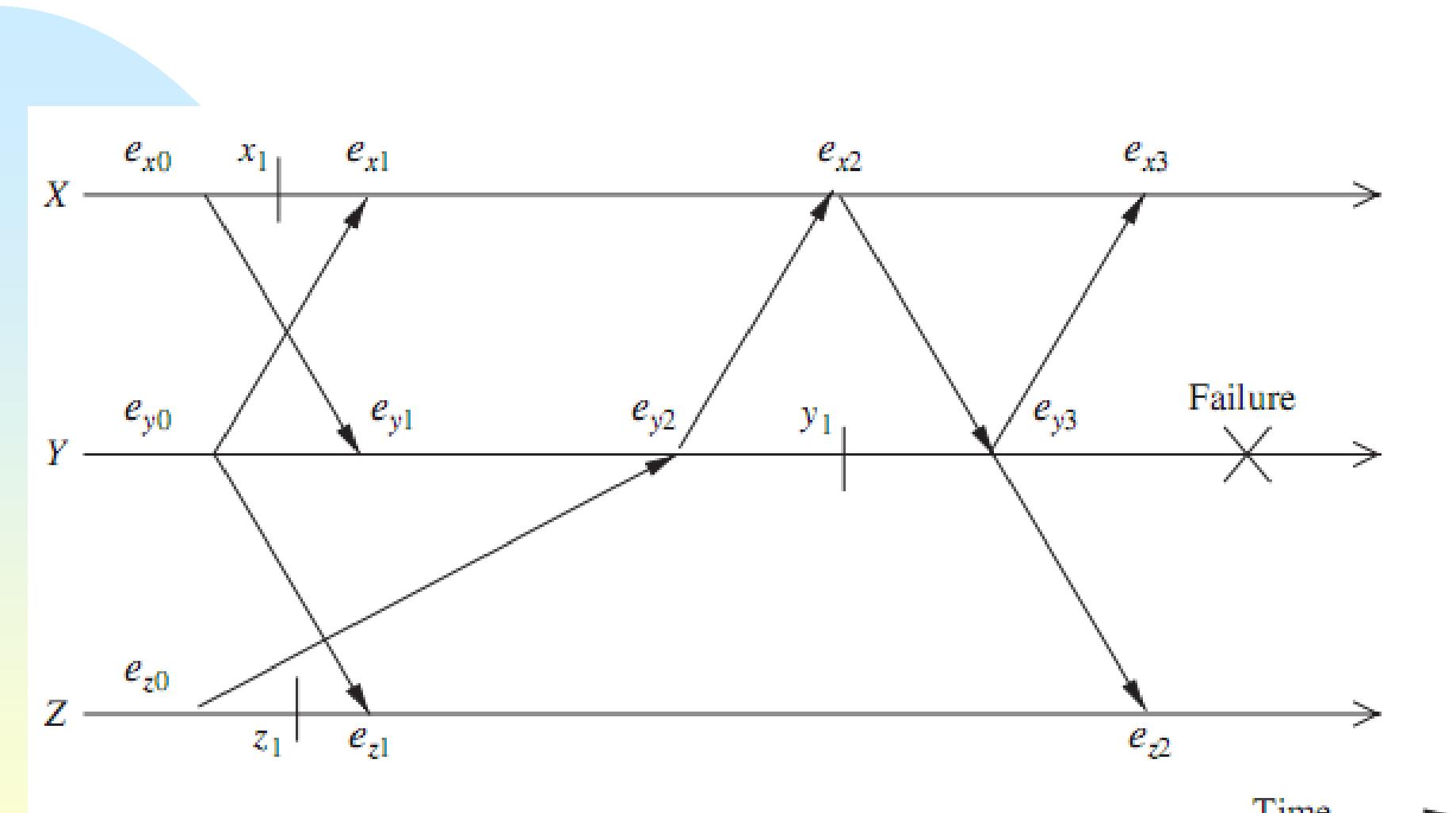
$CkPt_i \leftarrow e$

 end if

 end for

end for {for k }

Exemplu Juang-Venkatesan



Referinte

- A.D. Kshemkalyani, M. Singhal, Distributed Computing: Principles, Algorithms, and Systems, ISBN: 9780521189842, paperback edition, Cambridge University Press, March 2011

Calcul mobil

Si problemele specifice

Limitarile dispozitivelor mobile

- Implicatiile portabilitatii pentru dispozitivele mobile sunt
 - marimile
 - greutatile reduse
 - dependenta de puterea bateriei.
 - Memoria limitata

- Bateriile sunt printre cele mai mari dispozitive in greutate intr-un nod mobil.
- Consumul de putere este proportional cu CV^2f ,
 - C este capacitatea dispozitivelor si conexiunilor inter-dispozitive,
 - V este variația voltajului,
 - f este frecvența de ceas.

- Puterea poate fi economisita
 - (1) crescand nivelul de integrare VLSI astfel incat sa scada C ,
 - (2) reproiectand cip-urile sa lucreze la un voltaj V mai redus
 - (3) reducand frecventa de ceas dinamic pentru a negocia viteza de calcul pentru economia de putere.

- Transmisia wireless, receptia, retransmisia si operatiile de avertizare, toate acestea consuma putere.
- Multe protocoalele de routare existente folosesc transmisii periodice ale mesajelor de updateare a rutei pentru a mentine acuratetea tabelei de rutare.

Proprietatile sistemului distribuit mobil

- Disponibilitate
- confidentialitate

- De asemenea, este necesara o a treia parte, de ex. *Certificatin Authority*, pentru managementul cheii sau cheile trebuie sa fie transmise anticipat.

■ *Integritatea.*

■ *Securitatea.*

Contextul

- In retelele cablate, adresarea clasica a comunicatiei se face ascunzand layer-ul de inferior al comunicatiei, pe care majoritatea metodelor de tratare a erorilor nu 1-ar lua in considerare (reamintim ca pentru retelele cablate, erorile de legatura si host sunt considerate a fi evenimente foarte rare).

- Astfel, a aparut o tendinta opusa, de ex., expunerea evenimentelor unei retele mobile aplicatiei, care trebuie sa fie responsabila cu tratarea evenimentelor.
- Mergand mai departe, aplicatiile si-au dat seama de mediul inconjurator in care functioneaza si trebuie sa fie capabile sa se adapteze la el.

Retele senzor

- Retele senzor reprezinta o categorie aparte a retelelor ad hoc.
- Perceperea datelor inconjuratoare este realizata prin efortul comun al unui numar mare de noduri senzor, effort ce consta in perceperea, procesarea datelor si comunicarea componentelor.

- Pot fi amplasate aleator pe terenuri inaccesibile, precum operatiuni pe teren accidentat, deci protocoalele si algoritmii retelei trebuie sa detina capacitate de auto-organizare.

Diferente intre R. Senzor si R. Clasice

- 1. Numarul de noduri senzor intr-o retea senzor poate fi mai cu magnitudine mai mare de cateva ori.
- 2. Nodurile senzor sunt amplasate compact.
- ...

- 5. Nodurile senzor pot sa nu aiba identificare globala.
- 6. Mobilitatea senzorului poate fi limitata.

- În timp ce retelele traditionale tind să atingă calitatea de dispozitii ale serviciilor foarte înalte (high quality of service QoS), protocoalele de rețea senzor trebuie să se concentreze în principal pe conservarea puterii.

Modele de mobilitate

- Cercetatorii au propus mai multe scheme predictive de mobilitate pentru a prezice viitoarea disponibilitate a link-urilor wireless.

- Ca istoric, primele modele de mobilitate utilizate pentru retelele ad hoc au fost variante ale modelului random-walk, care defineste miscarile individuale ale nodului si este bazat pe directii si viteze intamplatoare (random).

■ Intr-un muzeu, vizitatorii merg cu pasi diferiti si pe traекторii diferite, varind in functie de obiectele lor de interes, dar mobilitatea paternurilor tinde sa se focalizeze in punctele comune de interes, cum ar fi pictura.

- Se poate observa ca miscarile nodurilor bazate pe grup (group-based) cauzeaza partitionarea retelei.
- Sa consideram o retea ad hoc formata din mai multe miscari de grup, a carei noduri sunt initial dispersate si amestecate; paternul de miscare distinct al fiecarui grup cauzeaza separarea grupurilor si in final, partitionarea retelei.

- in timp ce miscarile independente ale unui singur nod pot cauza doar ruperea sporadica si la intamplare a link-urilor.
- S-a aratat ca atitudinea mobilitatii de grup a utilizatorilor mobili cauzeaza partitionarea frecventa a retelelor, iar partitiile rezultate sunt miscari separate de grup.

- De exemplu, la timpul t , locatia unui nod din intr-un grup j este data de urmatorii vectori de pozitie:
 - 1. loc de referinta $Y_j(t)$;
 - 2. dizlocare locala $Z_{ji}(t)$;
 - 3. locul nodului $X_{ji}(t) = Y_j(t) + Z_{ji}(t)$.

- Modelul RPGM genereaza locatia fizica a nodurilor mobile, dar utilizarea ei poate sa nu identifice cu acuratete grupurile mobile.
- De exemplu, sa consideram o topologie a retelei generata de modelul RPGM unde exista cateva grupuri mobile cu punct de referinta comun si cu arii de acoperire suprapuse.

Protocole de routare ad-hoc

- Datorita gradului limitat de transmisie al interfatelor retelelor wireless, pot fi necesare mai multe hop-uri pentru a schimba date intre noduri in cadrul aceleiasi retele.

Protocole proactive

- În protocolele proactive, fiecare nod menține unul sau mai multe tabele pentru a stoca informații de routare.
- Aceste informații sunt întărite up-to-date cu ajutorul schimbului de mesaje periodic. .

Destination-Sequenced (DSDV)

- Protocoloul de routare vector-distanță se bazează pe mecanismul de routare Bellman-Ford, care a fost îmbunătățit pentru a evita buclele în tabelele de routare.
- Încărcătorul din rețea mobilă menține o tabelă de routare pentru toate destinațiile posibile din rețea.

Clusterhead Gateway Switch Routing (CGSR)

- Protocolul se bazeaza pe DSDV, dar solicita o structura a retelei inghesuita (clustered) si foloseste cateva scheme de routare euristice.
- In fiecare cluster un nod este ales ca si cluster head initialinzand un algoritm de selectie a cluster head-ului.

Protocole reactive

- Protocolele reactive presupun o abordare diferită , creând rute doar la cererea nodului sursă .
- Atunci când un nod dorește o rută către o destinație , inițiază *un proces de descoperire* a rutei, care este terminat atunci când este găsită o rută sau toate permutările dintre rute au fost examineate .

Protocolul *Dynamic Source Routing (DSR)*

- Este bazat pe conceptul rutării sursei .
- Când nodul primește un pachet de cerere a rutei (conținând adresele nodurilor sursă și destinație) , acesta verifică dacă “știe” deja calea spre destinație , și , dacă nu , adaugă propria adresă la *route record*-ul conținut în pachet și înațează pachetul spre celelalte link-uri ale sale .

- Un dezavantaj al acestui protocol este că suferă o problemă de scalabilitate datorată naturii procesului de *source routing* .
- Odată ce rețeaua devine mai mare , pachetele de control ..

Ad Hoc On-Demand Distance Vector(AODV)

- Protocolul de rutare se bazează pe algoritmul DSDV și minimizează redundanța acestuia creând rute numai la cerere .
- În locul rutării sursei , AODV se bazează pe intrările din tabelele generate dinamic în nodurile intermediare.

- Un pachet unidirecțional(RREP) este trimis către sursă permitând fiecărui nod din cale să stabilească un *forward pointer* către nodul de la care sosește pachetul RREP .

Algoritmul *Temporally Ordered Routing (TORA)*

- *Algoritm* este bazat pe conceptul de inversare a link-ului și operează într-o rețea cu un grad înalt de mobilitate .
- Conceptul cheie al TORA este localizarea mesajelor de control la un set mic de noduri din apropierea locului în care se produce o modificare topologică .

Associativity-Based Routing (ABR)

- Protocolul folosește ca unitate de măsură *gradul stabilității asociativității* .
- Fiecare nod generează periodic un semnal de control pentru a indica vecinilor prezența sa .
- Pentru fiecare semnal primit , contorul asociativ al nodului curent este incrementat .

- Toate nodurile care primesc mesajul de interogare își asociază propria adresă și contorul de asociativitate la mesaj .
- Un nod successor va șterge contorul vecinului său din aval cu excepția intrării referitoare la el însuși .

Signal Stability-Based Adaptive Routing (SSR)

- Protocolul selectează ruta pe baza puterii semnalului între noduri și stabilitatea locațiilor nodurilor pentru a alege ruta cu cea mai puternică conectivitate .
- Puterea semnalului este stabilită prin testarea link-urilor cu nodurile vecine .

Protocole hibride

- Protocolele de rutare ad-hoc hibride combină rutarea locală proactivă cu rutarea globală reactivă pentru atingerea unor eficiențe și scalabilități mai înalte .
- În cadrul *Zone Routing Protocol (ZRP)*, procesul de descoperire a rutei este inițiat la cerere .

Protocole bazate pe pozitie

- Protocolele bazate pe poziție necesită disponibilitatea informației despre poziționarea fizică a nodurilor ad-hoc.
- Fiecare nod își poate determina propria poziție prin GPS sau o altă metodă de localizare.

- Rutarea bazată pe poziție nu necesită stabilirea sau menținerea rutelor.
- Un avantaj în plus este că aceste rutine suportă trimiterea pachetelor către toate nodurile dintr-o zonă geografică dată.

- LAR presupune că transmіtătorul cunoaște localizarea destinației și viteza de transmitere .
- Pe baza acestei informații se poate defini o zonă estimată a destinației .

- De exemplu în proiectul *Terminodes* se combină rutarea ierarhică și rutarea bazată pe poziție într-o ierarhie pe două niveluri.
 - Pachetele sunt rulate conform unui vector de distanță proactiv pentru distanțe mici și printr-o abordare reactivă *greedy-position* pentru distanțe mari (*Anchored Geodesic Packet Forwarding*).

- Fiecăru i nod i se cere să-și cunoască propria poziție .
- Pentru cazul în care nu este disponibil un serviciu GPS , se propune un *Self Positioning Algorithm(SPA)*.

- De aceea, dacă toate nodurile nu aparțin unei aceleiași entități administrative, utilizatorii tind să fie egoiști : folosesc serviciile oferite de alții dar nu oferă la rândul lor servicii comunității.

Protocole care țin cont de consumul de energie

- Trebuie eliminate pe cât posibil coliziunile cu stratul MAC care presupun efectuarea unor retransmisii cu consecințe asupra consumului de energie.
- Un exemplu în acest sens este protocolul EC-MAC .
-

Rutarea în rețelele deconectate

- O abordare a tratării rețelelor deconectate ad-hoc este de a lăsa gazda mobilă să aștepte pasivă reconectarea la rețea.
- Acest lucru poate duce la întârzieri inacceptabile propunându-se în acest sens limitarea acestora prin exploatarea și controlarea mobilității nodului .

- Vahdat și Becjer propun un protocol *epidemic* pentru rețelele deconectate.
- Mecanismul de rutare derivă din algoritmii epidemici care oferă consistență bazelor de date replicate fără a fi nevoie de disponibilitatea unei anumite replici la un anumit moment de timp.

- Un dezavantaj al acestei abordări este necesitatea unei capacități sporite de buffer-izare.
- Chatzigiannakis propune un protocol şarpe , în care o secvență de purtători împerecheați de mesaje asemănatori unui şarpe se mișcă într-un mod determinat de capul şarpelui.
-

Rutarea bazată pe agenți în rețele ad-hoc

- Amin și Mikler propun un algoritm numit *Agent-based Distance Vector Routing (ADVR)* .
- Pentru fiecare sesiune , numărul de mesaje schimbate în rețea este limitat de numărul de agenți prezenți în rețea.
-

- Această strategie de migrare folosește o combinație de *Stigmergy*, ca formă de comunicare indirectă, și o căutare în adâncime.
- *Stigmergy* este un mecanism prin care insectele comunică între ele în funcție de schimbările de mediu.
-

Broadcasting și multicasting deficitar

- Protocole pentru problema broadcasting-ului și multicasting-ului deficitar.
- Acestea nu oferă siguranță în sensul că un mesaj poate fi pierdut dacă topologia rețelei se schimbă în timpul unui proces multicasting
- *Simple Flooding*

Probability Based Methods

Area Based Methods

Neighbor Knowledge Methods

- Zhou și Sing propun o varianta numită *Content Based Multicasting (CBM)* pentru rețelele ad-hoc.
- În CBM , conținutul datelor care suferă un proces multicasting împreună cu mobilitatea receptorilor determină setul multicasting.

- Receptorii “trag” apoi informația de la nodurile din calea acesteia.
 - Protocolul presupune maparea și împărțirea zonei de operații în regiuni .

Serviciul de apartenenta la grup

- Un protocol de apartenenta la grup se ocupa cu formarea si intretinerea unui set de procese care formeaza un grup.

- In general, un proces poate parasi un grup din cauza ca a esuat, din cauza ca a cerut sa paraseasca grupul sau din cauza ca a fost eliminat fortat de catre alti membri ai grupului. In mod similar, un proces poate deveni membru al unui grup

- (ex: in urma unei selectii pentru a juca rolul de replica pentru un alt proces din grup).

Problema “Consensului”

- Aceasta problema nu poate fi rezolvata in mod deterministic in sisteme asincrone, chiar in cazul in care comunicarea este buna, un singur proces poate esua, iar aceasta se poate intampla numai prin aparitia unei erori.

1. in apartenenta la grup,
2. consensul

- Totusi, pentru pentru cazul serviciilor apartenente la grup care urmaresc sa mentina o imagine unica “in acord” a componentei curente a grupului, s-a demonstrat ca apartenenta la grup nu este rezolvabila deterministic in sisteme asincrone in care comunicarea este fiabila si in care cel mult un proces poate termina anormal

- Pentru a evita acest rezultat de imposibilitate, au fost propuse asa numitele servicii de apartenenta la grup “partitionabile”.

- Astfel de servicii de apartenenta la grup admit “separarea grupului” (ex: cand reteaua se partitioneaza) si “reunirea grupului” (ex: cand comunicarea intre partitii este restaurata).

- Cristian propune trei specificatii de apartenenta la grup pentru “modelul asincron temporizat” cu trei garantii de consistenta diferite:
 - “inteligerea in grup”,
 - “inteligerea majoritatii” si
 - “inteligerea stricta”.

- Procesele schimba mesaje prin intermediul unui sistem de comunicare de tip datagrama.
- Mesajele se pot pierde si intarzierile mesajelor sunt nemarginite, dar, cele mai multe mesaje ajung la destinatie intr-o constanta de intarziere bine definita pentru comunicare uni-directionala, **d**.

- Pentru ceasurile hardware se presupune modelul de defectarea prin terminare anormală și, în plus, un proces care nu s-a încheiat eronat are un ceas hardware corect.

- Cand intarzierile de programare depasesc s , serverele sufera de scaderi de performanta.
- Astfel, serverele au un model de defectare de tip *eroare/performanta*.

$$\delta = s + D + s$$

- Doua procese p si q sunt *conectate* intr-un interval de timp $[t, t']$ daca sunt corecte (ex: nu au terminat anormal si sunt temporizate) si orice mesaj trimis intre ele in $[t, t' - \delta]$ ajunge la destinatie in limita a δ unitati de timp.

- De remarcat, ca orice pereche de procese se poate afla in unul si numai unul din modurile anterioare.
- Un set de procese care sunt conectate doua cate doua formeaza o *partitie fizica*.

- De remarcat ca un sistem stabil constă în una sau mai multe partitii fizice separate.
- Se presupune că sistemul alternează între lungi perioade de stabilitate și perioade comparativ scurte de instabilitate, ex: comunicarea asincronă poate fi stabilită în mare parte majoritatea timpului.

- Perioadele de instabilitate tranzitională sau mai multe perioade de deconectare între diferite parti ale rețelei pot rezulta în crearea mai multor grupuri paralele.

- Doua grupuri G si G' sunt considerate *paralele* daca nici unul dintre ele nu este successorul celuilalt. Procesele p si q sunt partitionate (logic) la momentul t daca intra in componenta a doua grupuri paralele diferite la momentul t .
- Un grup G este un *grup majoritar* daca setul $mem(G)$ al membrilor sai contine majoritatea membrilor echipei P , adica

$$|mem(G)| > \frac{|P|}{2}$$

- Generic vorbind, cand un grup G este instalat pe un proces p, p este informat asupra grupului precedent $\text{pred}(q, G)$ al fiecarui proces q care este membru al lui

- In acest caz, ei ar fi putut aplica update-uri conflictuale asupra starilor lor locale si astfel, ar fi putut devia.
- Daca se detecteaza devierea de stare, starea initiala a noului grup G trebuie sa rezolve update-urile conflictuale.

Probleme de autostabilizare

- Dolev and Schiller propun un algoritm aleator pentru implementarea unui serviciu de apartenenta la grup autostabilizant in sisteme asincrone.

- Un proces poate trimite un mesaj catre toti vecinii lui intr-o singura operatie de comunicare.
- Procesele pot sa termine anormal si sa isi revina in timpul executiei.

Problema partitionarii

- Prin natura lor, aplicatiile de retea pentru calcul mobil implica o cooperare intre mai multe situri.
- Pentru aceste aplicatii, caracterizate de cerinte fiabilitate si reconfigurabilitate, posibila partitionare a retelei de comunicare este un aspect extrem de important al mediului.

- În mod intuitiv, partitiile corespund unui număr maximal de componente conectate ale grafului logic care reprezintă relația de “accesibilitate” între procese.
- Astfel, ele pot fi definite doar în contextul unor primitive de comunicare specifice.

- Natura partitionarii va determina calitatea aplicatiei in functie de ce servicii sunt disponibile, unde si la ce nivale de performanta.

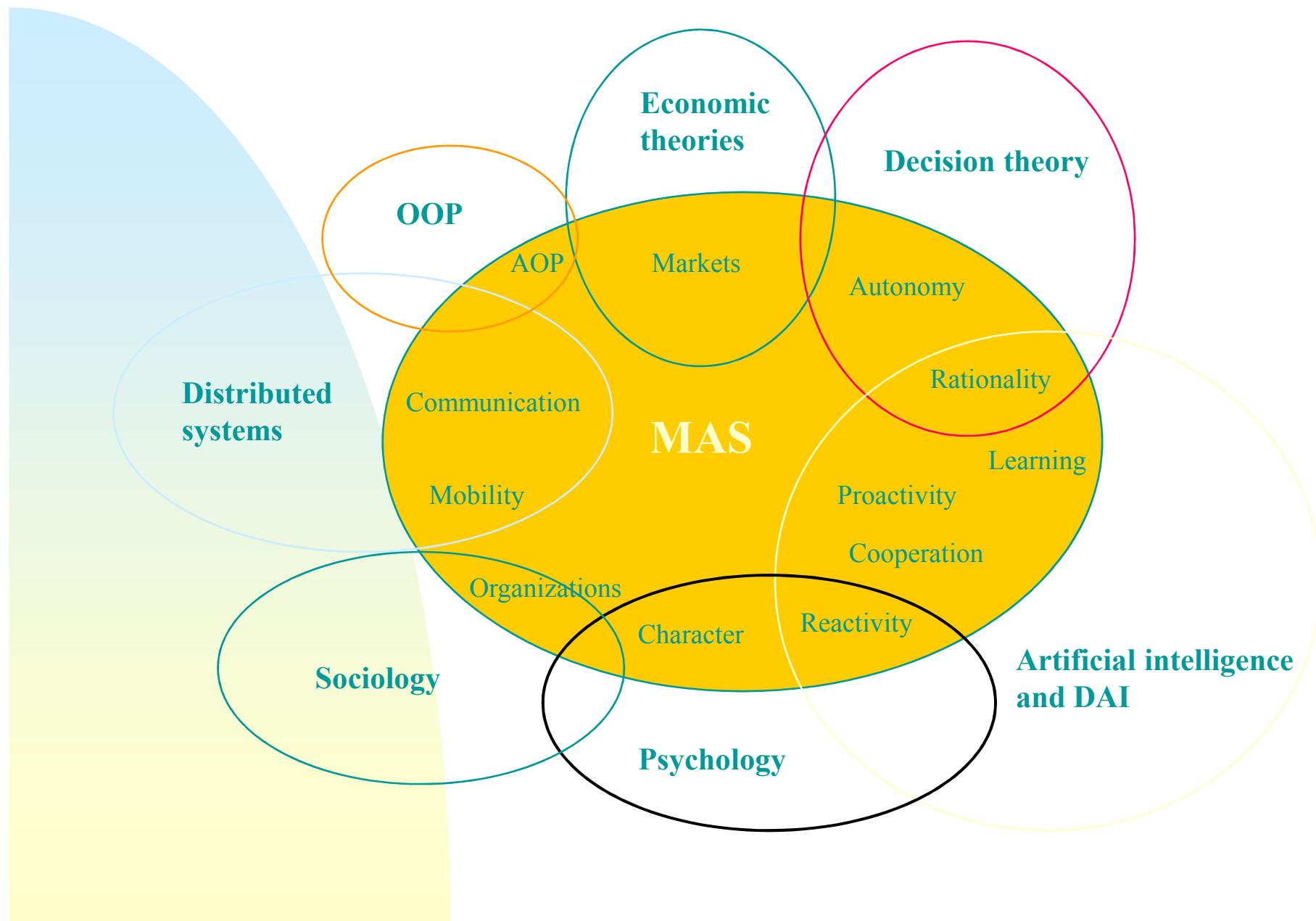
- Reducerea si degradarea serviciilor depind in mod esential de semantica aplicatiei.
- Pentru anumite clase de aplicatii cu constrangeri severe de consistenta, se poate ajunge la cazul in care toate serviciile sa fie suspendate complet in toate partitiile, cu exceptia uneia.

Cursul

*Inteligenta artificială
distribuită*

- Termenul a aparut pentru prima oara in anii'50 la MIT.
- o **entitate care** functioneaza **permanent** si **autonom** intr-un mediu in care exsita alte procese si/sau alti agenti" (Shoham, 1993)
- "Un **agent este o entitate care** **percepe** mediul in care se afla si **actioneaza** asupra acestuia" (Russell, 1997)

Legaturi cu alte discipline



- Totusi analizand literatura se observa o serie de caracteristici commune mai multor definitii dupa cum urmeaza:
 - ◆ Autonomia
 - ◆ Perceptia mediului prin senzori
 - ◆ Posibilitatea modificarii mediului prin efectori
 - ◆ Exista cu ajutorul unor medii support (frameworks)

- “Agentul poate fi privit ca o ființă vie iar mediul support poate fi privit ca lumea înconjuratoare acesteia”
- Ca și în cazurile reale dacă **tinem cont de definitia inteligenței:**
- „capacitatea de adaptare la mediu precum și de modelare a acestuia pentru a-și urmări scopurile de bază”
- **se observă agentii inteligenți suporta într-o formă abstractă și aceasta definitie**

- Agentii pot avea urmatoarele caracteristici
 - ◆ Reactia:
 - ◆ Autonomia:
 - ◆ Colaborarea:
 - ◆ Inferenta:



- ◆ Continuitate:

- ◆ Personalitate:

- ◆ Adaptabilitatea:

- ◆ Mobilitate:

- Tipuri de agenti
 - ◆ Agenti colaborativi:
 - ◆ Agenti pentru Interfata:

- Agenti mobili:
- Agenti informationali:
- Agenti reattivi:

- Agenti hibrizi:
- Agenti eterogeni:

Taxonomia A

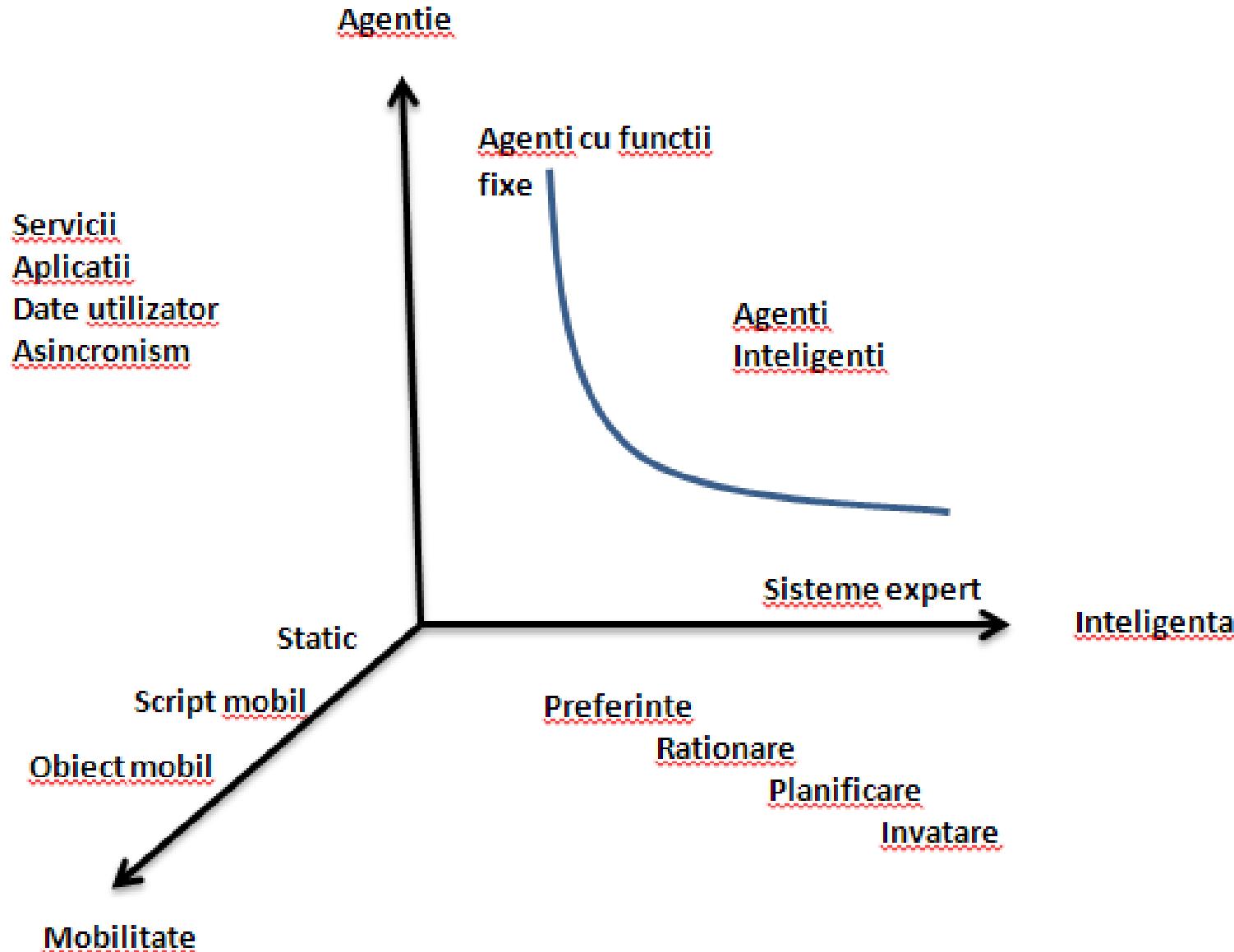
- Clasificare dupa atributele primare
 - ◆ Inteligenti
 - ◆ Colaborativi
 - ◆ De interfata
 - ◆ ...
- Dupa mobilitate
 - ◆ Statici
 - ◆ Mobili

- Dupa existenta sau lipsa unirii rationament simbolic
 - ◆ Deliberativi
 - ◆ Reactivi
- Dupa roluri
 - ◆ Informativi
 - ◆ Distribuiti pe Internet

Taxonomia B

- Agenti biologici
- Agenti robotici
- Agenti care simuleaza viata artificiala
- Agenti software
 - ◆ Cu sarcini specifice
 - ◆ De divertisment
 - ◆ Virusi

Si acestia pot fi integrati intr-un spatiu tridimensionale



Agentii mobili

- Reprezinta o entitate software executata intr-un mediu suport dedicat (framework) si respecta urmatoarele modele:
 - ◆ Al ciclului de viata
 - ◆ Computational
 - ◆ De securitate
 - ◆ De comunicatii
 - ◆ De navigare

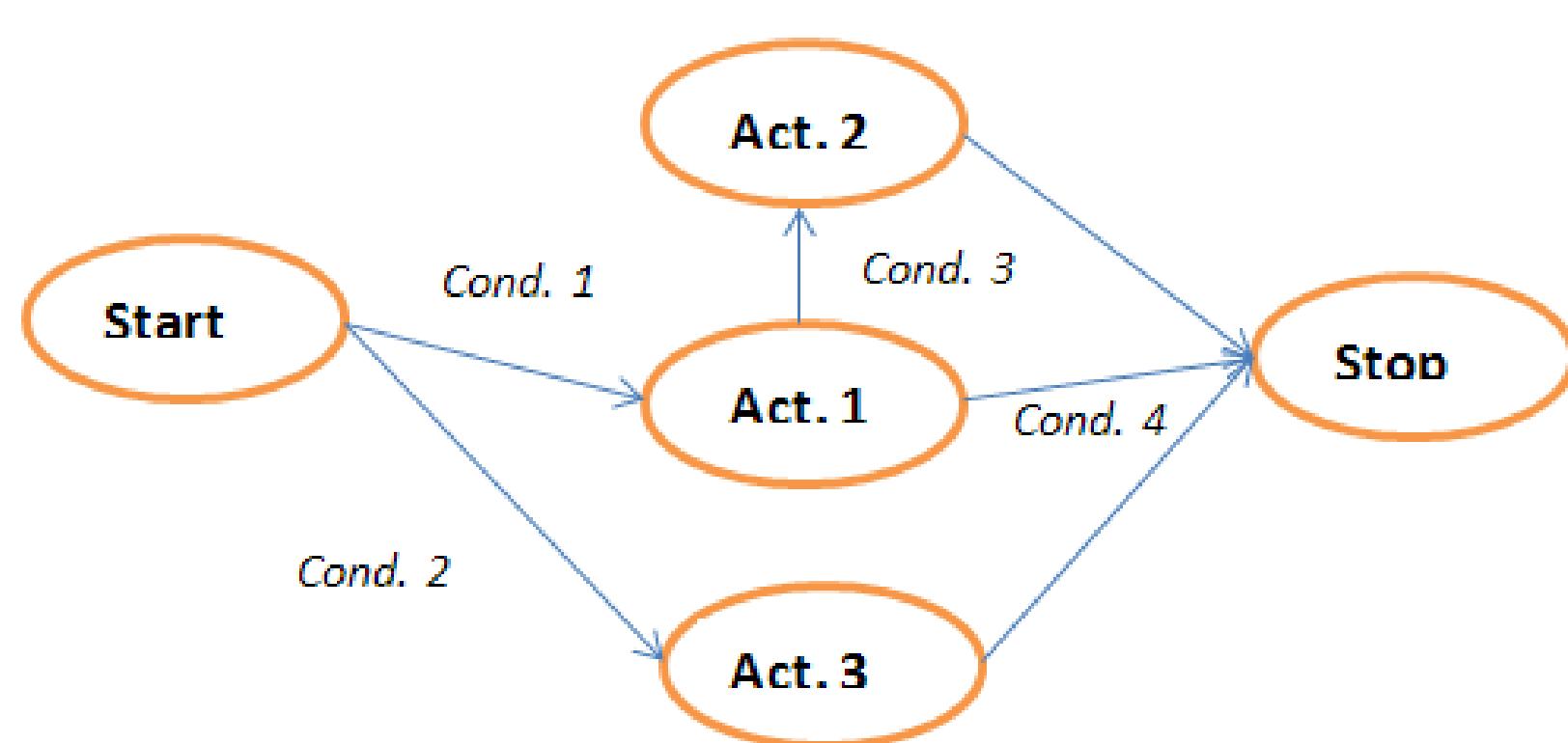
Mediul de lucru

- Poate fi static (modele matematice riguroase) sau dinamic (paralelismul problemei).
- O aplicatie distribuita avand ca arie de instalare un cluster dinamic neomogen.
- Practic avem de a face cu un sistem deschis cu urmatoarele caracteristici:
 - ◆ Structura retelei evolueaza dinamic
 - ◆ Modulele pot fi dezvoltate pentru diverse platforme la diverse momente de timp
 - ◆ Resursele sunt distribuite

- De fapt există două abordări care influențează parțial funcționarea mediului de lucru
 - ◆ Abordarea în context distribuit:
 - ◆ Abordarea folosind sisteme multiagent:

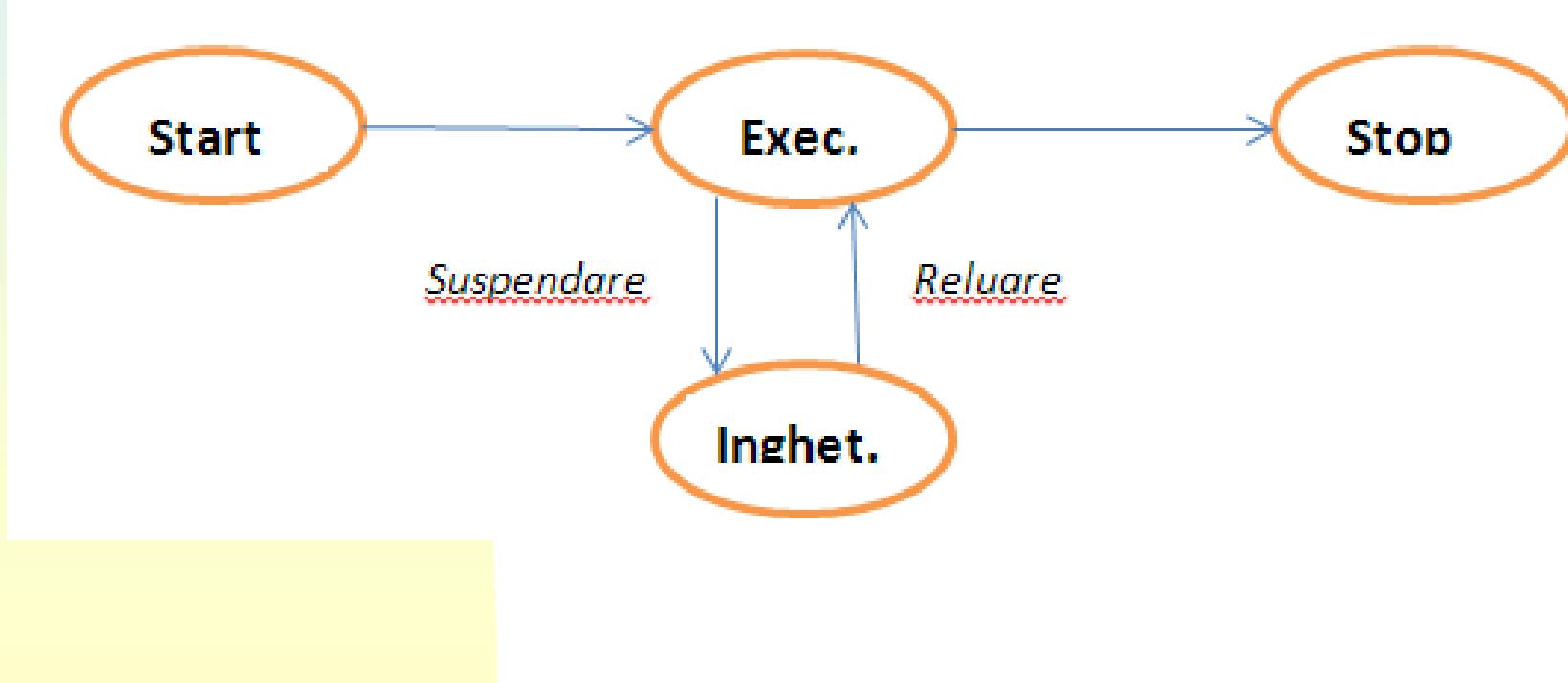
Modelul ciclului de viata

- Model bazat pe procese persistente (Telescript si Agent TCL)



Model bazat pe actiuni

- Se observa similitudinea cu modelul de viata a unui thread



Modelul computational

- Dintre acestea cele mai importante sunt
 - ◆ Pentru agenti mobili Tck/Tk, Action, Telescript, Linda,....
 - ◆ Pentru agenti cu capacitatii cognitive: Agent 0, Concurrent Metatem, KQML

Modelul securitatii

- Se observa aceleasi probleme ca in cazul oricarei aplicatii distribuite.
- Mai multe capacitatile cognitive ale agentilor pot duce la riscuri de securitate aproape imposibil de estimat

Comunicare în SMA

- Comunicare indirectă
- Comunicare directă
 - ◆ ACL
 - ◆ Limbaje pentru conținut
 - ◆ Teoria actelor de vorbire
 - ◆ KQML
 - ◆ FIPA and FIPA-ACL
- Protocole de interacțiune

Comunicare in SMA

Comunicare agenti

- nivel scazut
- nivel inalt
- Implica interactiuni

Protocole de comunicare

Protocole de interactiune → conversatii = schimb structurat de mesaje

Scop → comunicarea permite agentilor:

- coordinarea actiunii si comportarii
- schimbarea starii altor agenti
- determina agenti sa faca actiuni

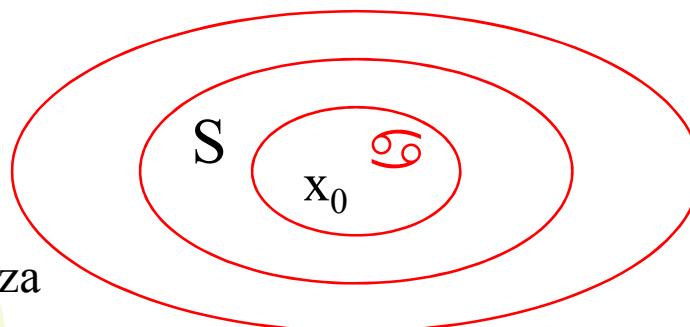
Comunicare indirecta

In general pt agenti reactivi

- Comunicare prin semnale

$$V(x) = V(x_0) / \text{dist}(x, x_0)$$

Agent A
(stimulus genreaza
comportare P)



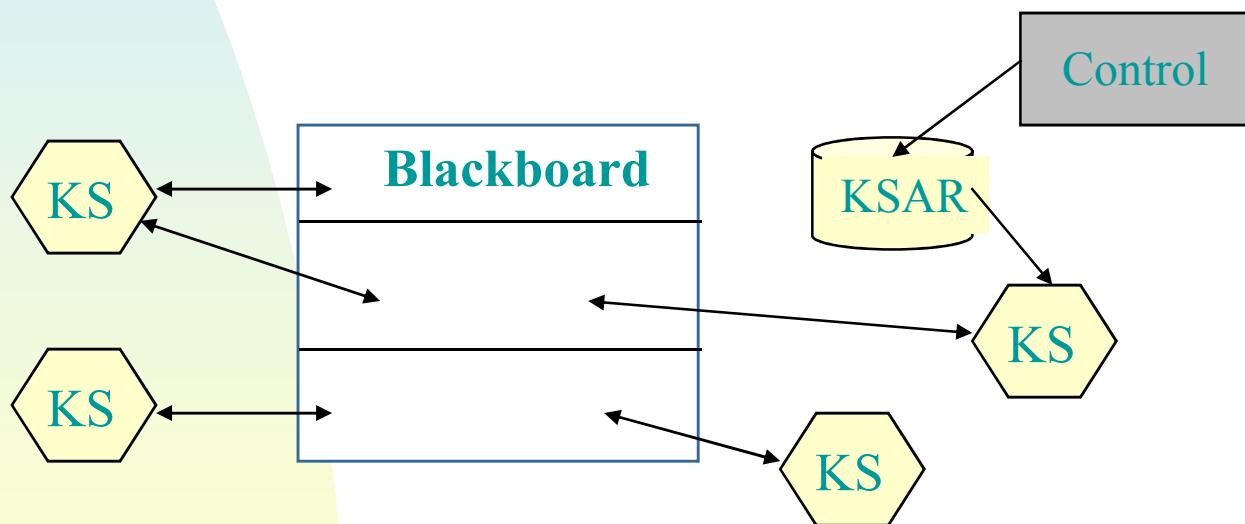
S - stimulus

Agent B
(stimulus genreaza
comportare P)

- Comunicare prin "urme" lasate in mediu

Comunicare indirectă

- Comunicare în sisteme tip "blackboard"



Comunicare directă

- SMA – limbaje de nivel înalt
- Presupun în general agenti BDI

- ACL = *Agent Communication Languages*
- Comunica cunoștinte → rep. cunoștinte
- Intelegerea mesajului în context → ontologii
- Comunicare vazuta ca o actiune – **acte de vorbire
(de comunicare)**

ACL – limbaje comunicare interagent

Concepte

- UN mesaj ACL descrie o stare dorita intr-un limbaj declarativ (decat sa foloseasca o procedura sau un RPC)
- ACL gestioneaza propozitii, reguli si actiuni (in loc de obiecte fara semantica asociata)
- ACL sunt bazate in general pe teoriile BDI (agentii BDI incearca sa-si comunice starea in scopul alterarii starii vecinilor (agenti cognitivi))
- ACL sunt bazate pe Speech Act Theory
- ACL se refera la Ontologii comune ***Ontologies***

3 nivele ale comunicarii

- **Primitive si protocol**

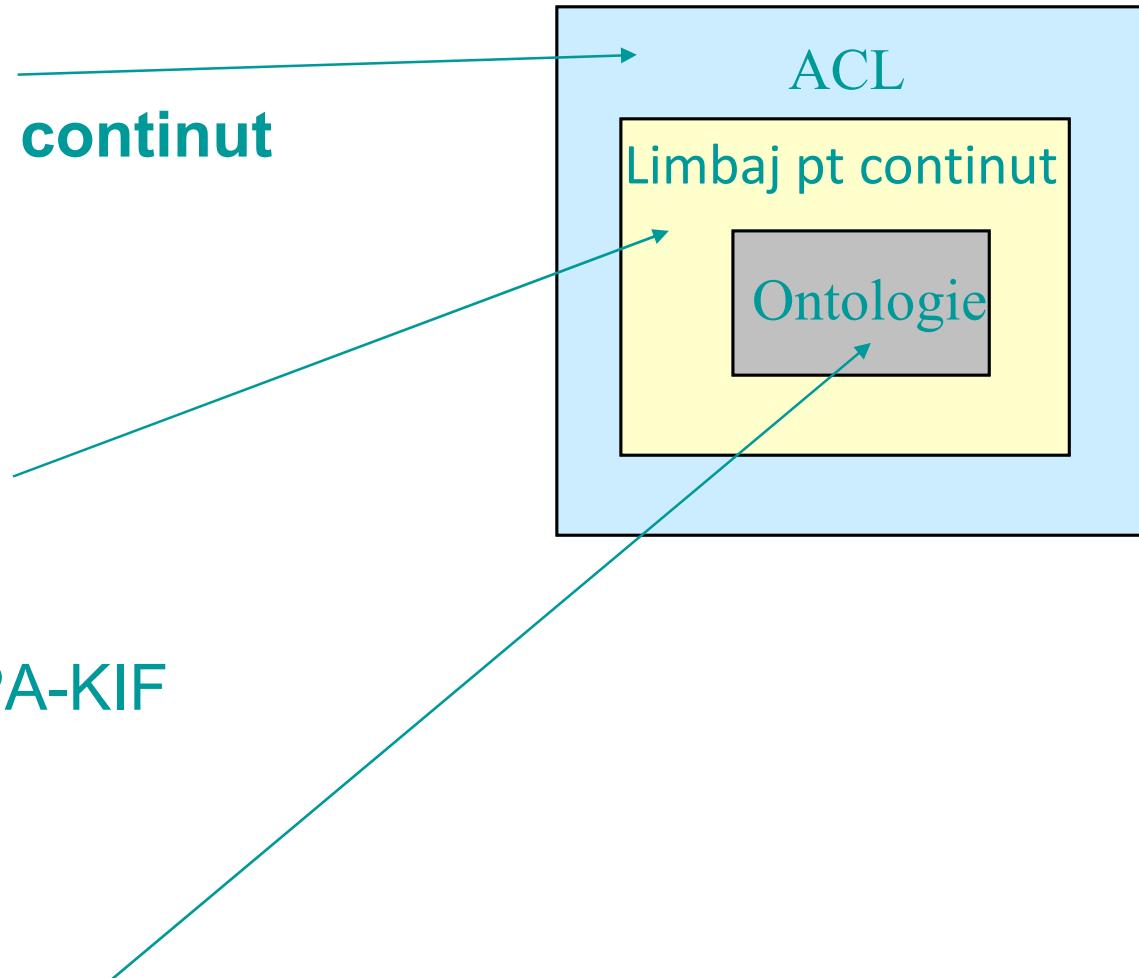
- KQML
- FIPA

- **Limbaje pentru continut**

- ◆ KIF
- ◆ Prolog
- ◆ Clips
- ◆ SQL
- ◆ DL
- ◆ FIPA-SL, FIPA-KIF

- **Ontologii**

- ◆ DAML
- ◆ OWL



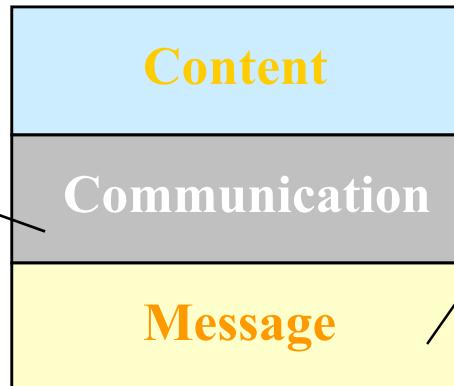
Primitive ACL

- Bazate pe acte de comunicare / acte de vorbire
 - J. Austin - *How to do things with words*, 1962,
 - J. Searle - *Speech acts*, 1969
- Cele 3 straturi separă:
 - ◆ continutul și semantica mesajului
 - ◆ semantica comunicării (acte vorbire) – independentă de domeniu
- Un ACL are o semantica formala bazata pe un formalism logic
- 2 ACL-uri care s-au impus:
 - ◆ KQML
 - ◆ FIPA-ACL
- Pot include definitii de protocoloale

KQML-Knowledge Query Manipulation Language

Parametrii comunicarii

- identitate emitator si receptor
- identificator unic asociat comunicarii



KQML de baza

- protocol retea
- act de vorbire

Optional

- limbajul continutului
- ontologie

Tipuri de performative

- **Queries** - **ask-one, ask-all, ask-if, stream-all,...**
- **Generative** - **standby, ready, next, rest, discard, generate,...**
- **Response** - **reply, sorry ...**
- **Informational** - **tell, untell, insert, delete, ...**
- **Capability definition** - **advertise, subscribe, recommend...**
- **Networking** - **register, unregister, forward, route, ...**

Exemplu KQML

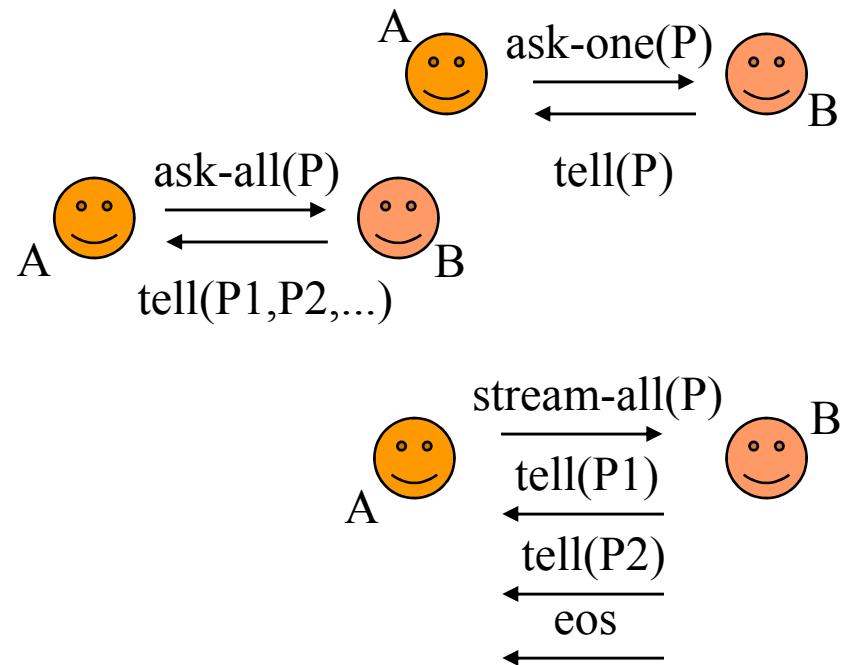
intrebare

```
(ask-one :sender joe  
         :receiver ibm-stock  
         :reply-with ibm-stock  
         :language PROLOG  
         :ontology NYSE-TICKS  
         :content (price ibm ?price) )
```

(tell

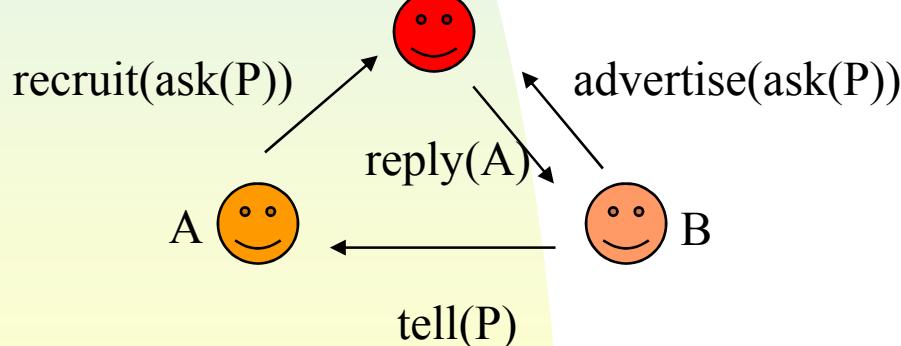
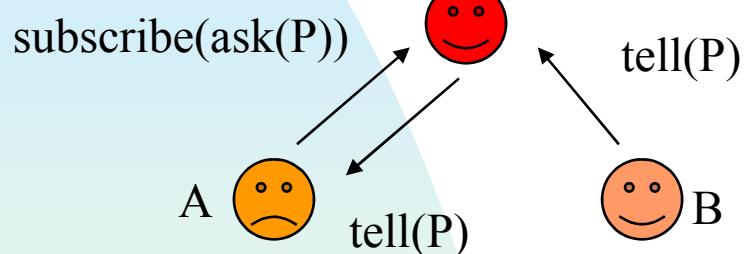
```
:sender willie  
:receiver joe  
:reply-with block1  
:language KIF  
:ontology BlockWorld  
:content (AND (Block A) (Block B) (On A B)) )
```

informare

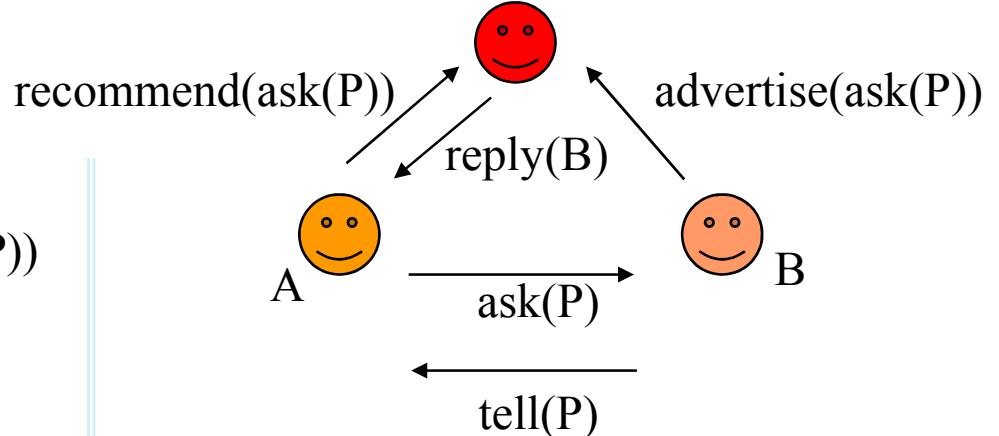
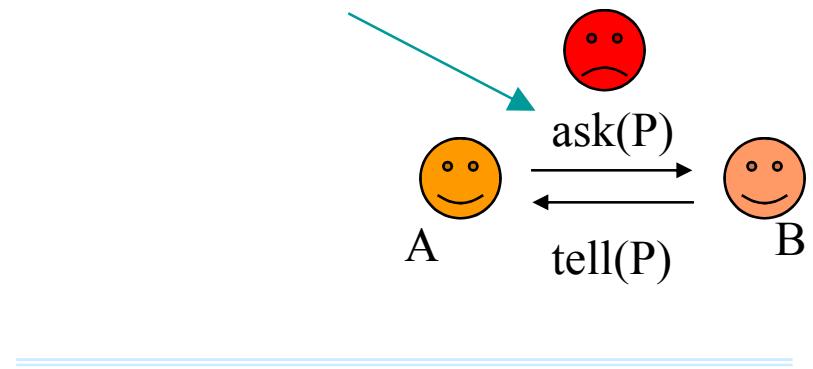


Agent facilitator

capacitate



intrebare point-to-point



FIPA ACL

- Asemanator cu KQML
- Primitive oarecum diferite
- Semantica semnificativ diferita

(inform

:sender (agent-identifier :name i)

:receiver (set (agent-identifier :name j))

:content "weather (today, raining)"

:language Prolog)

Exemplu FIPA

```
(request :sender (agent-identifier :name i)
         :receiver (set (agent-identifier :name j)
                      :content ((action (agent-identifier :name j)
                                         (deliver box7 (loc 10 15))))
                     :protocol fipa-request
                     :language fipa-sl
                     :reply-with order56 )
```

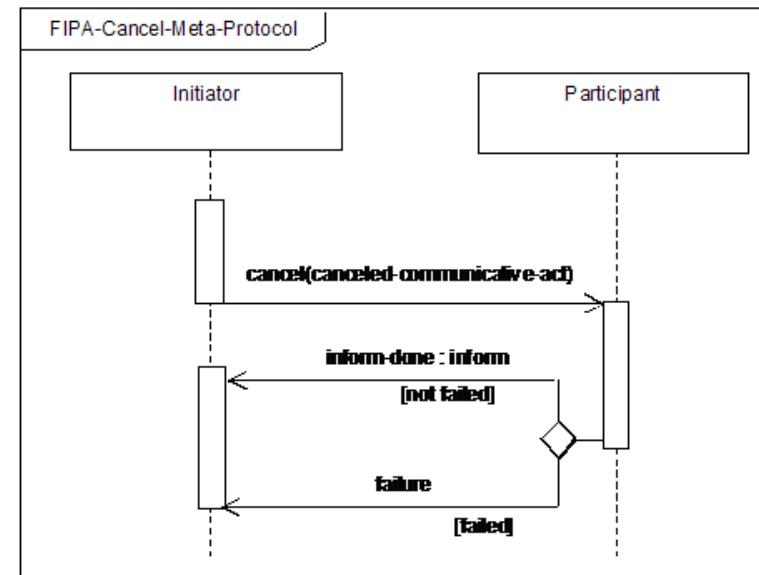
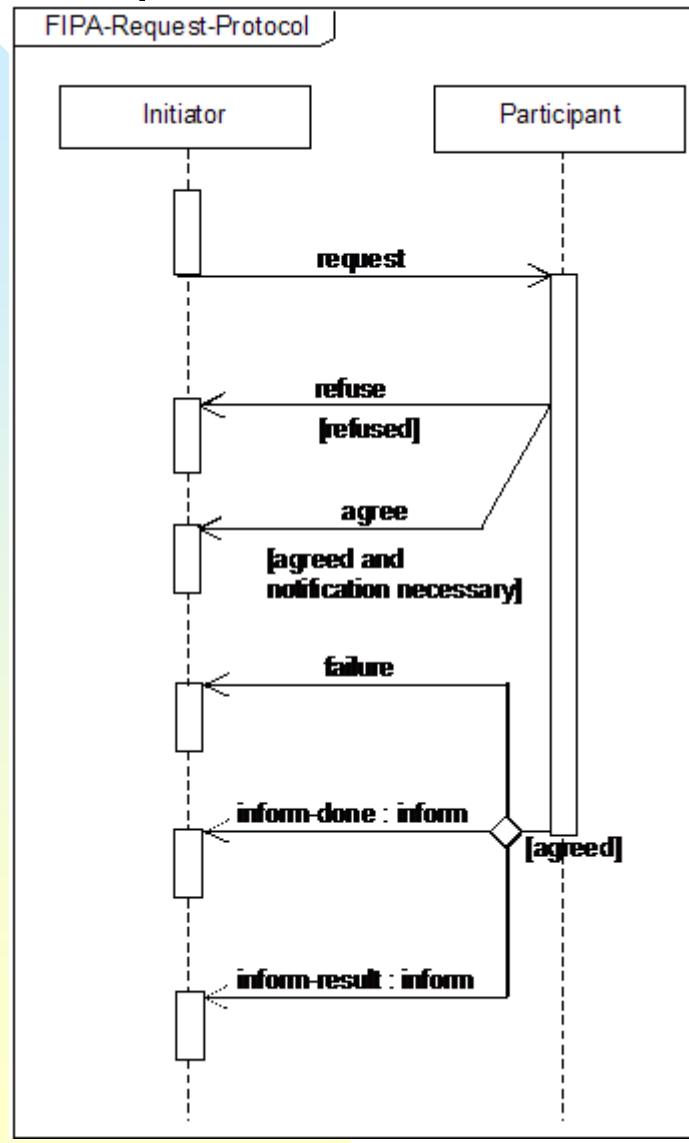
```
(agree    sender (agent-identifier :name j)
           :receiver (set (agent-identifier :name i)
                          :content ((action (agent-identifier :name j)
                                            (deliver box7 (loc 10 15))) (priority order56
low))
                     :protocol fipa-request
                     :language fipa-sl
                     :in-reply-to order56 )
```

Primitive FIPA

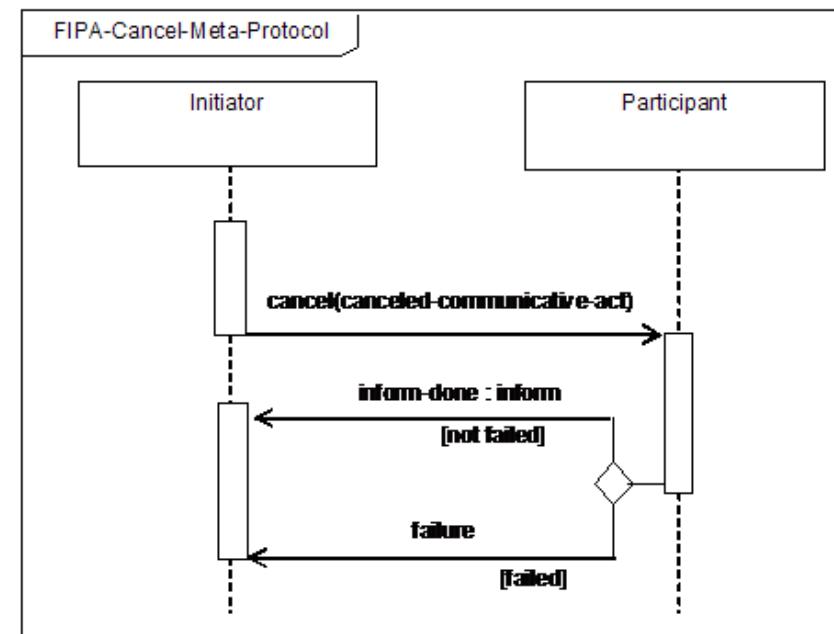
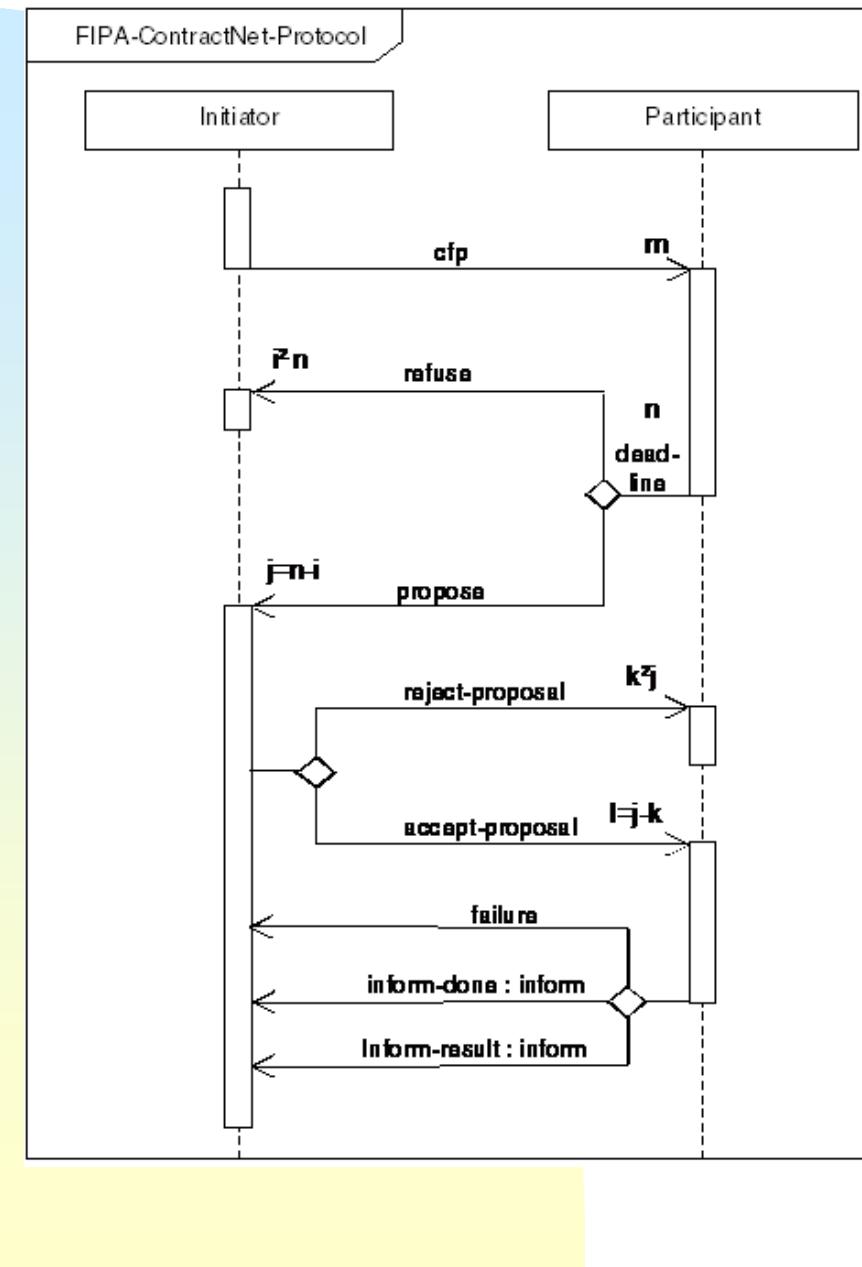
- **FIPA – acte de comunicare**
- **Informativ**
 - `query_if`, `subscribe`, `inform`, `inform_if`, `confirm`, `disconfirm`,
`not_understood`
- **Distributie taskuri**
 - `request`, `request_whenever`, `cancel`, `agree`, `refuse`, `failure`
- **Negociere**
 - `cfp`, `propose`, `accept_proposal`, `reject_proposal`

FIPA - Protocole

■ FIPA - Request



FIPA - Contract net



Limbaje pentru continut

- ◆ KIF
- ◆ Prolog
- ◆ Clips
- ◆ SQL
- ◆ FIPA-SL, FIPA-KIF

Knowledge Interchange Format (KIF)

■ Facts

```
(salary 015-46-3946 john 72000)
(salary 026-40-9152 michael 36000)
(salary 415-32-4707 sam 42000)
```

■ Asserted relation

```
(> (* (width chip1) (length chip1))
  (* (width chip2) (length chip2)))
```

■ Rule

```
(=> (and (real-number ?x)
           (even-number ?n))
      (> (expt ?x ?n) 0))
```

Procedure

```
(progn (fresh-line t)
       (print "Hello!")
       (fresh-line t))
```

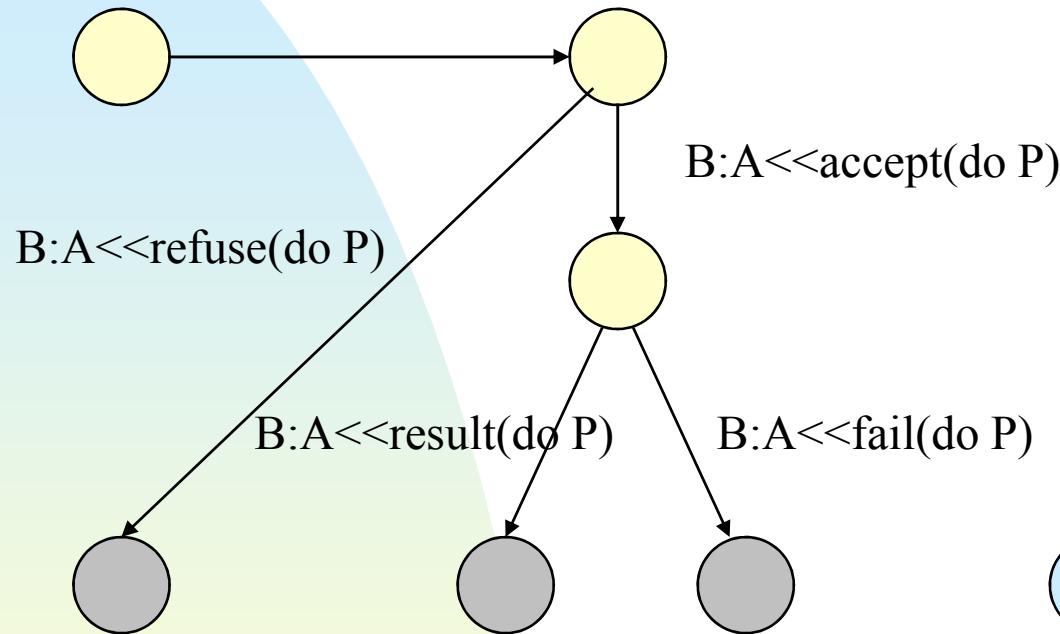
Protocole de interactiune

Permit agentilor realizarea de conversatii = schimburi structurate de mesaje

- Automate finite
- Conversatii in KQML
- Retele Petri

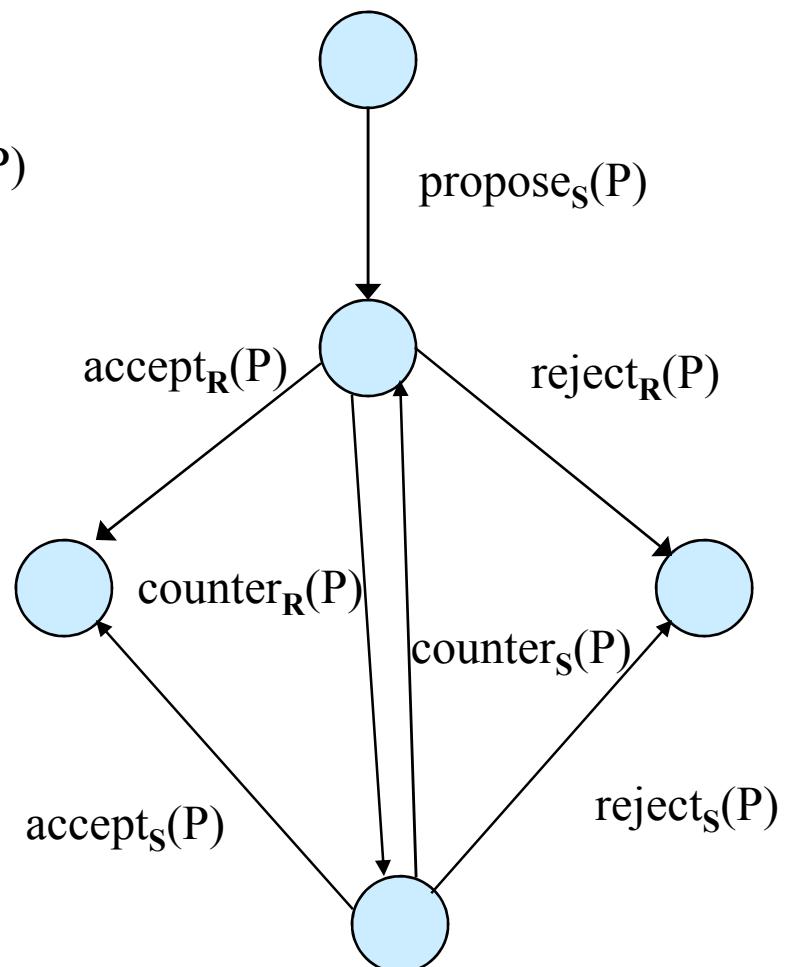
Automate finite

A:B<<ask(do P)



Winograd, Flores, 1986

COOL, Barbuceanu, 95



Conversatii in KQML

Definirea clauzelor gramaticale

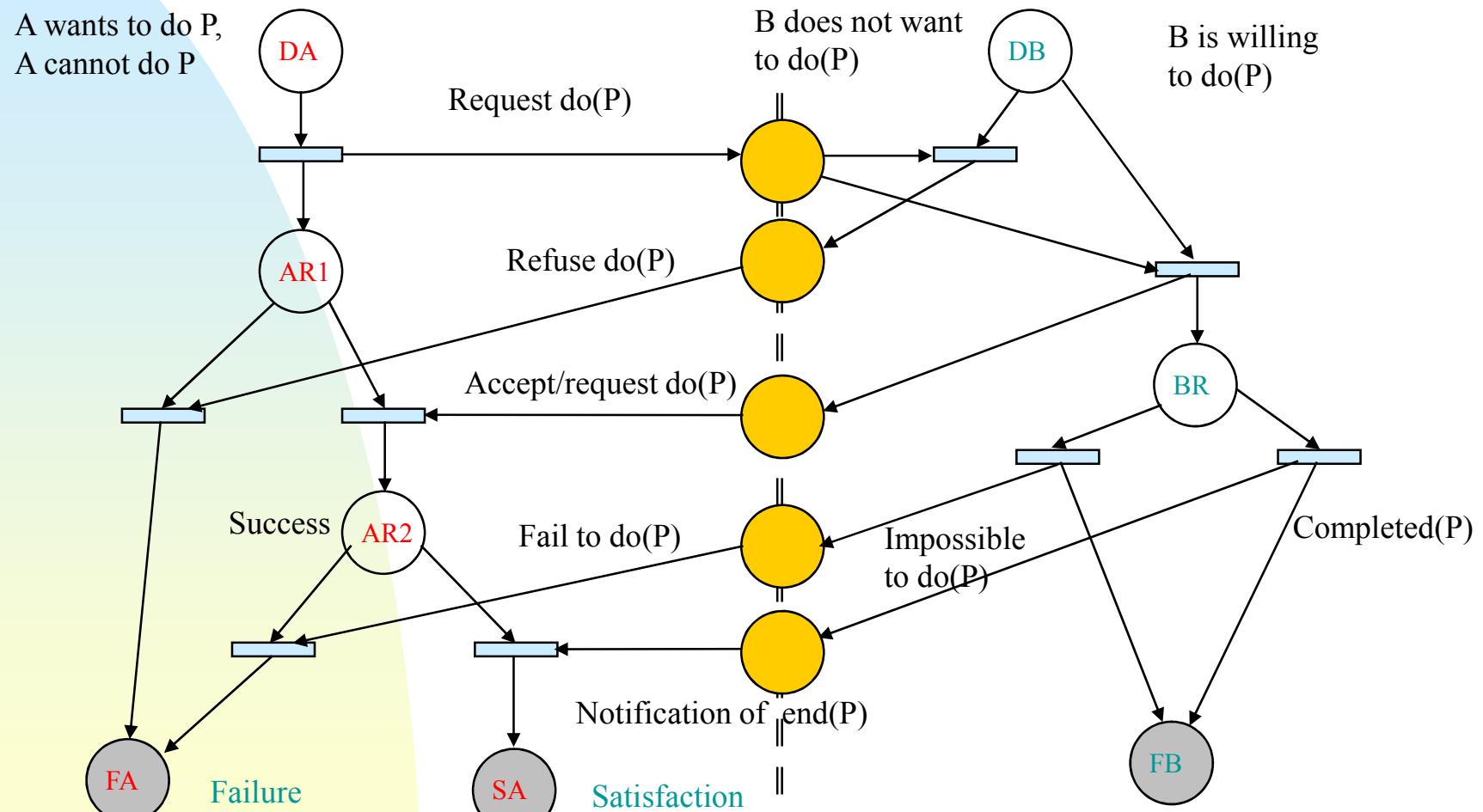
$S \rightarrow s(\text{Conv}, P, S, R, \text{inR}, \text{Rw}, \text{IO}, \text{Content}), \{\text{member}(P,$
 $\text{[advertise, ask-if]}\}$

$s(\text{Conv}, \text{ask-if}, S, R, \text{inR}, \text{Rw}, \text{IO}, \text{Content}) \rightarrow$
 $[\text{ask-if}, S, R, \text{inR}, \text{Rw}, \text{IO}, \text{Content}] \mid$
 $[\text{ask-if}, S, R, \text{inR}, \text{Rw}, \text{IO}, \text{Content}], \{\text{OI is } \text{inv}(\text{IO})\},$
 $r(\text{Conv}, \text{ask-if}, S, R, _, \text{Rw}, \text{OI}, \text{Content})$

$r(\text{Conv}, \text{ask-if}, R, S, _, \text{inR}, \text{IO}, \text{Content}) \rightarrow$
 $[\text{tell}, S, R, \text{inR}, \text{Rw}, \text{IO}, \text{Content}] \mid$
 $\text{problem}(\text{Conv}, R, S, \text{inR}, _, \text{IO})$

Retele Petri

Ferber, 1997



- Modelul navigarii
 - ◆ Stabilirea unor conventii de numire a agentilor dintr-un sistem de agenti mobili
 - ◆ Accesul la informatii privind mediile de agenti mobili aflate la distanta
 - ◆ Abilitatea de a muta un agent mobil aflat in stare de suspendare pe o alta masina in vederea executiei
 - ◆ Abilitatatea de a primi un agent suspendat si de a-l reconstrui intr-un nou mediu de calcul

Tehnici comune de rezervare a task-urilor

- CNP – Contract Network Protocol: In acest caz agentul cere realizarea unui set de subtask-uri. Agentul contractor primește cereri și oferte. Apoi se realizează alocarea dinamică a task-urilor
- PGP – Partial Global Planning: un fel de plan comun și scopuri comune
- FAC – Functionally Accurate Model: agentii obțin informația dorită asincron prin întrebări directe
- Joint Interaction Network: O intenție comună a mai multor agenti pentru a rezolva un task

Modele arhitecturale de agenti

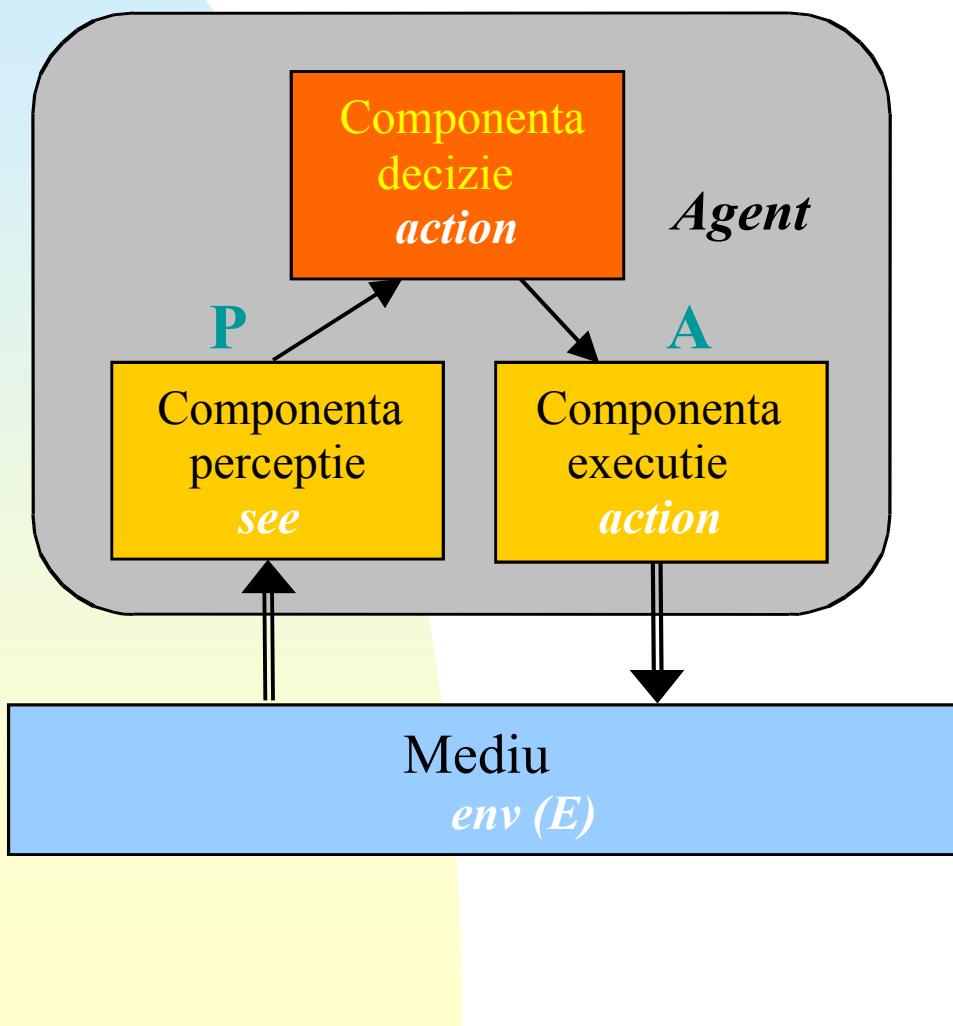
- Structura conceptuala a agentilor
- Arhitecturi de agenti cognitivi
- Arhitecturi de agenti reactivi

Structura conceptuală a agentilor

Rationalitatea unui agent

- Ce înseamnă rationalitatea unui agent
- *Cum putem măsura rationalitatea unui agent?*
- O măsura a performanței

Modelare agent reactiv

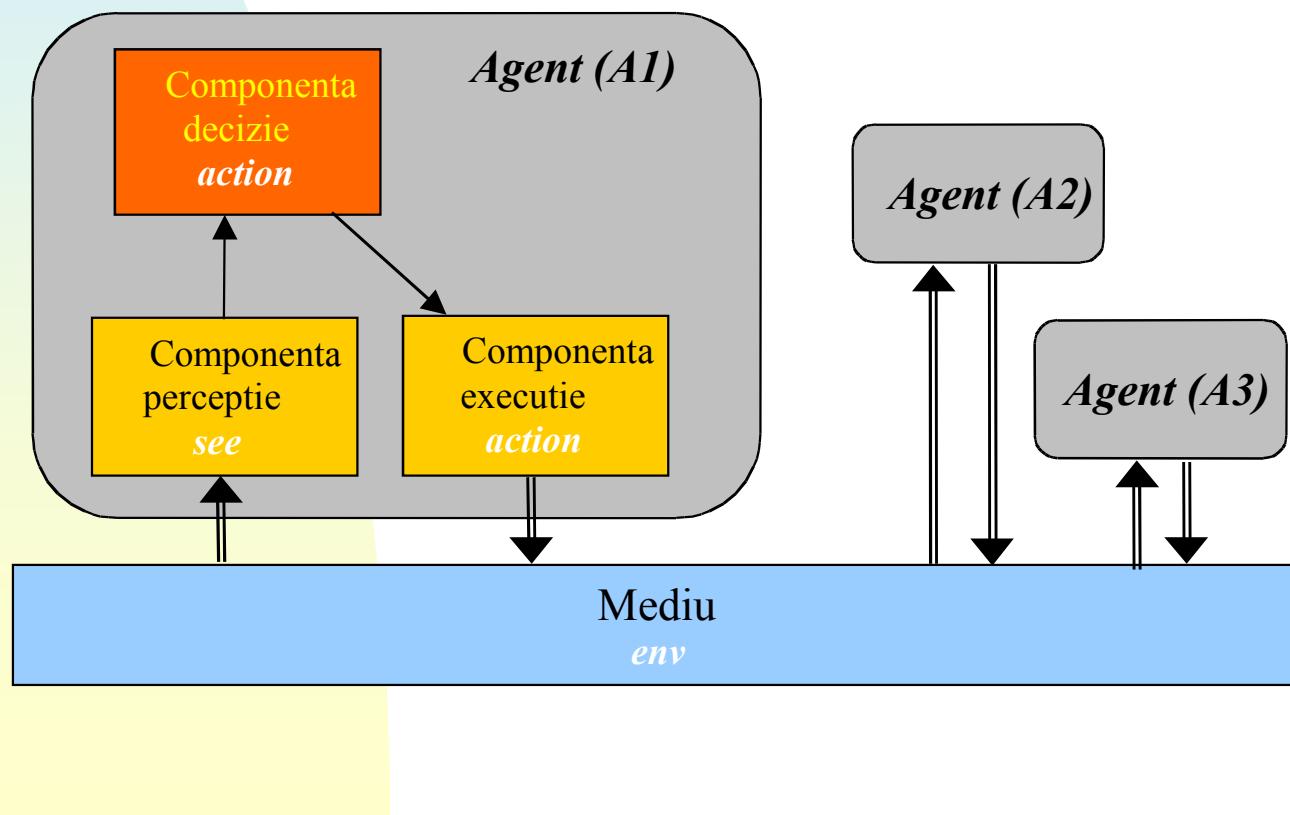


$$\begin{aligned} E &= \{e_1, \dots, e, \dots\} \\ P &= \{p_1, \dots, p, \dots\} \\ A &= \{a_1, \dots, a, \dots\} \end{aligned}$$

Agent reactiv
see : $E \rightarrow P$
action : $P \rightarrow A$
 $\text{env} : E \times A \rightarrow E$
 $(\text{env} : E \times A \rightarrow P(E))$

Modelare agenti reactivi

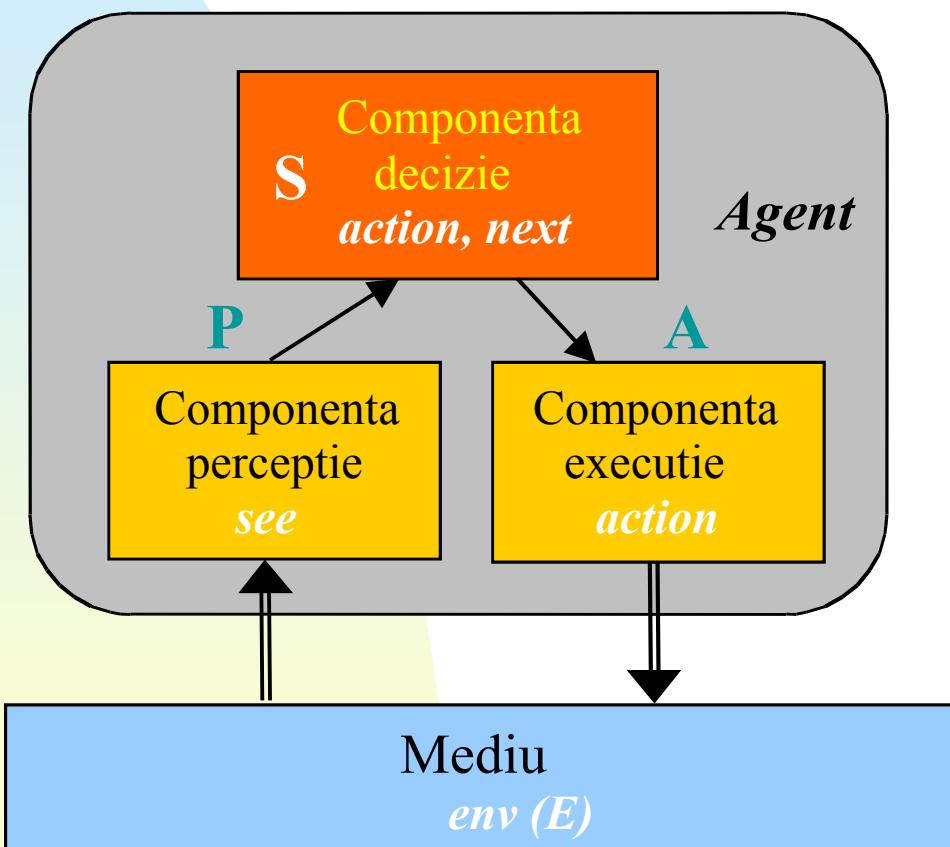
A_1, \dots, A_i, \dots
 P_1, \dots, P_i, \dots
(de obicei identice)



Mai multi agenti reactivi

$$\begin{aligned} see_i : E &\rightarrow P_i \\ action_i : P_i &\rightarrow A_i \\ env : E \times A_1 \times \dots \times A_n &\rightarrow P(E) \end{aligned}$$

Modelare agent cognitiv



$$\begin{aligned} E &= \{e_1, \dots, e, \dots\} \\ P &= \{p_1, \dots, p, \dots\} \\ A &= \{a_1, \dots, a, \dots\} \\ S &= \{s_1, \dots, s, \dots\} \end{aligned}$$

Agent cu stare
see : $E \rightarrow P$
next : $S \times P \rightarrow S$
action : $S \rightarrow A$
env : $E \times A \rightarrow P(E)$

Modelare agenti cognitivi

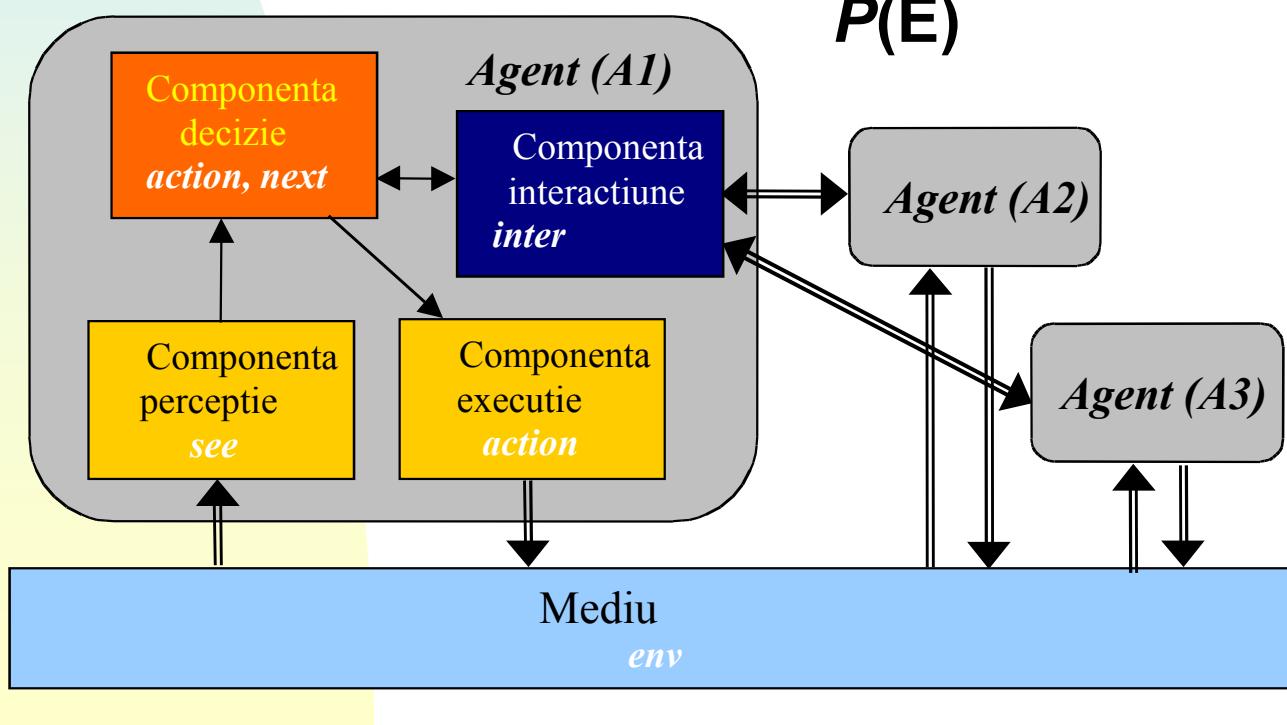
S_1, \dots, S_i, \dots

A_1, \dots, A_i, \dots

P_1, \dots, P_i, \dots

(nu intotdeauna identice)

$I = \{i_1, \dots, i_k, \dots\}$



Mai multi agenti cognitivi

$\text{see}_i : E \rightarrow P_i$

$\text{next}_i : S_i \times P \rightarrow S_i$

$\text{action}_i : S_i \times I \rightarrow A_i$

$\text{inter}_i : S_i \rightarrow I$

$\text{env} : E \times A_1 \times \dots \times A_n \rightarrow P(E)$

Modelare agent cognitiv

Agenti cu stare si scopuri

$goal : E \rightarrow \{0, 1\}$

Agenti cu utilitate

$utility : E \rightarrow R$

Mediu nedeterminist

$env : E \times A \rightarrow P(E)$

Probabilitatea estimata de un agent ca rezultatul unei actiuni (a) executata in e sa fie noua stare e'

$$\sum_{e' \in env(e, a)} prob(ex(a, e) = e') = 1$$

Modelare agent cognitiv

Agenti cu utilitate

Utilitatea estimata (*expected utility*) a unei actiuni a intr-o stare e , dpv al agentului

$$U(a, e) = \sum_{e' \in env(e, a)} prob(ex(a, e) = e') * utility(e')$$

Principiul utilitatii estimate maxime
Maximum Expected Utility (MEU)

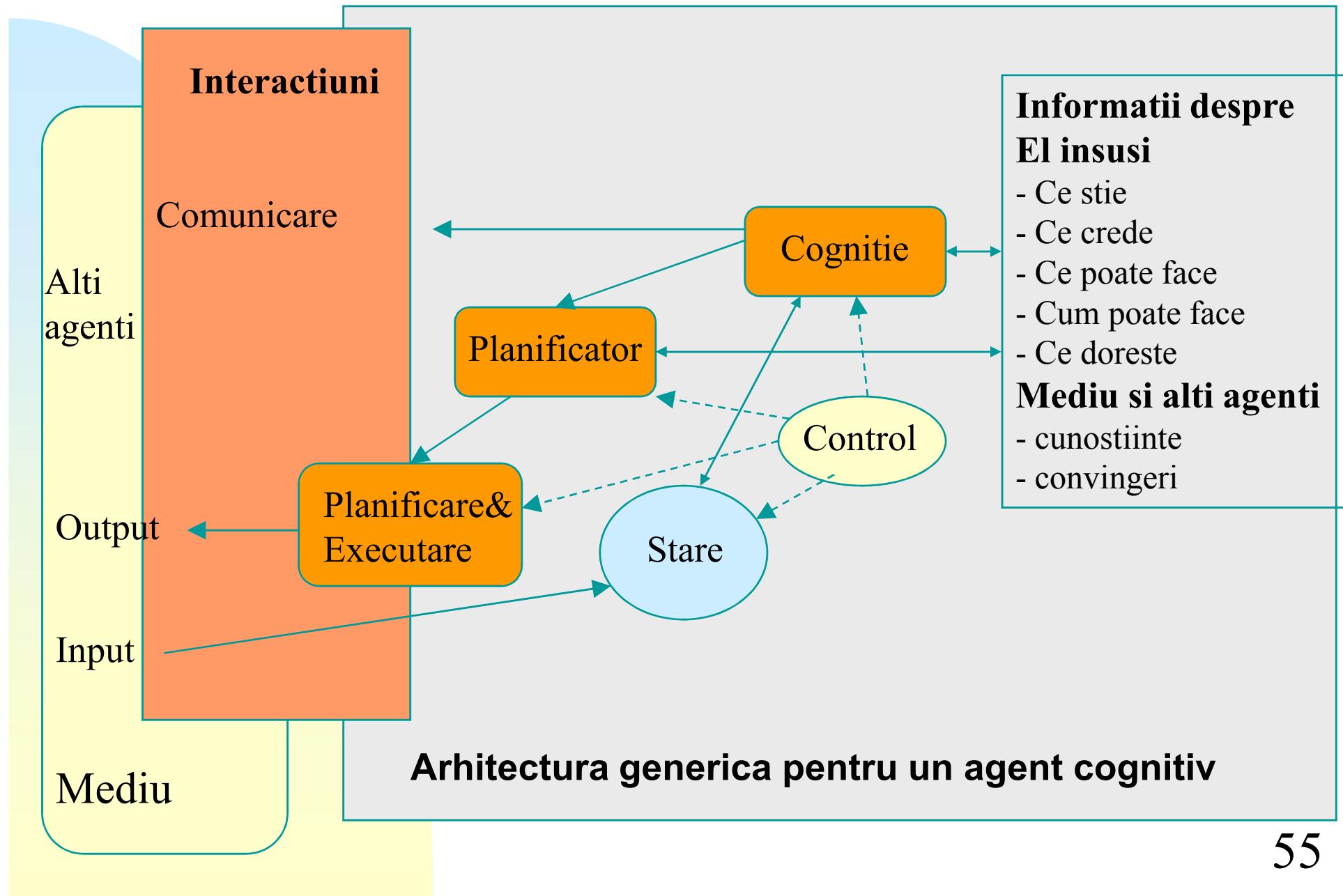
Masura a performantei

Arhitecturi de agenti cognitivi

Comportare ratională

Problema 1 = deliberare/decizie vs. actiune/proactivitate

- Problema 2 = limitarea resurselor



Modele

- Reprezentare simbolica + inferente – demonstrarea teoremelor pt a afla ce actiuni va face agentul
- Abordare declarativa
- **(a)Reguli de deductie**

Predicate At(x,y), Free(x,y), Wall(x,y), Exit(dir), Do(action)

Fapte si axiome despre mediu

At(0,0)

Wall(1,1)

$\forall x \forall y \text{ Wall}(x,y) \rightarrow \neg \text{Free}(x,y)$

Reguli de deductie

$\text{At}(x,y) \wedge \text{Free}(x,y+1) \wedge \text{Exit(east)} \rightarrow \text{Do(move_east)}$

Actualizare automata a starii curente si test pt starea scop

At(0,3)

(b) Utilizarea calcului situational = descrie schimbari utilizand formalismul logic

- **Situatie** = starea rezultata prin executarea unei actiuni

Result(Action, State) = NewState **At**(location, situation)

$\text{At}((x,y), S_i) \wedge \text{Free}(x,y+1) \wedge \text{Exit(east)} \rightarrow$

$\text{At}((x,y+1), \text{Result(move_east}, S_i))$

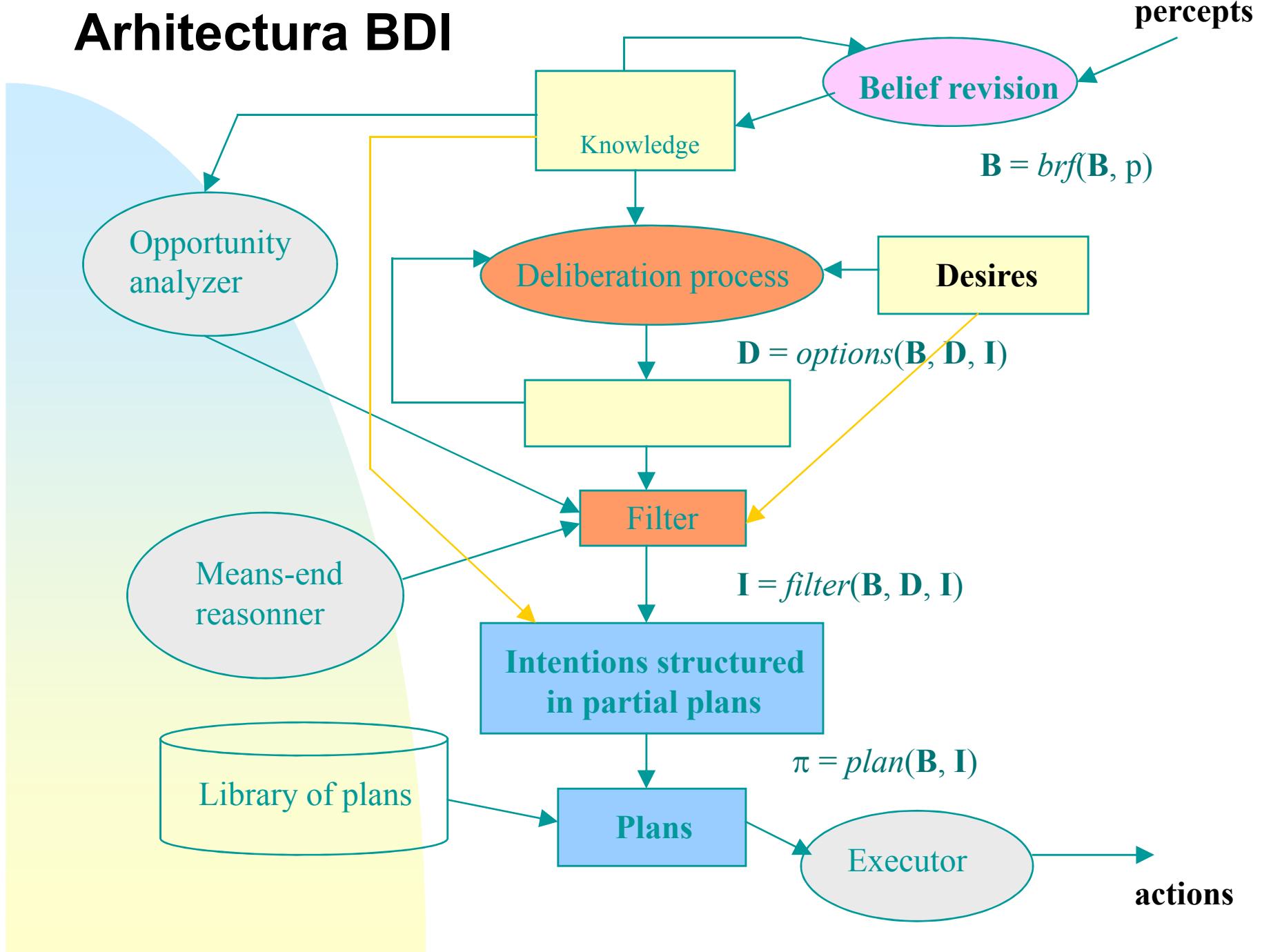
Scop **At**((0,3), _) + actiuni care au condus la scop

means-end analysis

Agentii inteligenți

- Există mai multe tipuri de arhitecturi cognitive specifice:
 - ◆ Bazate pe o reprezentare simbolica:
 - ◆ Societatea de agenti:

Arhitetura BDI



Arhitectura Belief Desire Intention – BDI:

```
Init_stare()
Repeat
{
    optiune=generator_optiuni(coada de
evenimente)
    optiune selectata=delibereaza (optiuni)
    modifica intențiile (noile optiuni selectate)
    executa()
    preia_noi_evenimente_externe()
    adauga_noi_intenții_posibile()
    adauga_noi_intenții_imposibile()
}
```

Bucla de control a agentului

B = B₀ **I = I₀** **D = D₀**

while true **do**

- get next percept p
- B = brf(B,p)**
- D = options(B, D, I)**
- I = filter(B, D, I)**
- π = plan(B, I)**
- execute(π)

end while

Strategii de angajare (Commitment strategies)

- Optiune aleasa de agent ca intentie – **agentul s-a angajat pentru acea optiune**
- Persistenta intențiilor

Intrebare: Cat timp se angajeaza un agent fata de o intenție?

- **Angajare oarba (Blind commitment)**
- **Angajare limitata (Single minded commitment)**
- **Angajare deschisa (Open minded commitment)**

```

B =  $B_0$   

I =  $I_0$  D =  $D_0$   

while true do  

    get next percept p  

    B = brf(B,p)  

    D = options(B, D, I)  

    I = filter(B, D, I)  

     $\pi$  = plan(B, I)  

    while not (empty( $\pi$ ) or succeeded (I, B)) do  

         $\alpha$  = head( $\pi$ )  

        execute( $\alpha$ )  

         $\pi$  = tail( $\pi$ )  

        get next percept p  

        B = brf(B,p)  

        if not sound( $\pi$ , I, B) then  

             $\pi$  = plan(B, I) ← Reactivitate, replanificare  

    end while  

end while

```

Bucla de control BDI

angajare oarba

```

B = B0
I = I0 D = D0
while true do
    get next percept p
    B = brf(B,p)
    D = options(B, D, I)
    I = filter(B, D, I)
     $\pi = \text{plan}(B, I)$ 
    while not (empty( $\pi$ ) or succeeded (I, B) or impossible(I, B)) do
         $\alpha = \text{head}(\pi)$ 
        execute( $\alpha$ )
         $\pi = \text{tail}(\pi)$ 
        get next percept p
        B = brf(B,p)
        if not sound( $\pi$ , I, B) then
             $\pi = \text{plan}(B, I)$ 
        end if
    end while
end while

```

Bucla de control BDI

angajare limitata

*Renuntarea la intențiile care sunt imposibile
Sau s-au realizat deja*



Reactivitate, replanificare

$B = B_0$
 $I = I_0 \quad D = D_0$
Bucla de control BDI
angajare deschisa

```

while true do
    get next percept p
     $B = \text{brf}(B, p)$ 
     $D = \text{options}(B, D, I)$ 
     $I = \text{filter}(B, D, I)$ 
     $\pi = \text{plan}(B, I)$ 
    while not (empty( $\pi$ ) or succeeded ( $I, B$ ) or impossible( $I, B$ )) do
         $\alpha = \text{head}(\pi)$ 
        execute( $\alpha$ )
         $\pi = \text{tail}(\pi)$ 
        get next percept p
         $B = \text{brf}(B, p)$ 
        if reconsider( $I, B$ ) then
             $D = \text{options}(B, D, I)$ 
             $I = \text{filter}(B, D, I)$ 
             $\pi = \text{plan}(B, I)$ 
← Replanificare
    end while
end while

```

- Nu exista o unica arhitectura BDI
- PRS - Procedural Reasoning System (Georgeff)
- dMARS
- UMPRS si JAM – C++
- JACK – Java
- JADE – Java
- JADEX – XML si Java,
- JASON – Java

Agenti emotionali????

- Inteligenta afectiva
- Actori virtuali
 - ◆ recunoasterea vorbirii
 - ◆ gesturi, sinteza de vorbire
- Emotii:
 - ◆ Aprecierea unei situatii sau a unui eveniment: **bucurie, suparare;**
 - ◆ valoarea unei situatii care afecteaza pe alt agent: **bucuros-pentru,, gelos, invidios, suprat-pentru;**
 - ◆ Aprecierea unui eveniment viitor: **speranta, frica;**
 - ◆ Aprecierea unei situatii care confirma o asteptare: **satisfactie, dezamagire**
- Controlarea emotiilor prin temperament

Arhitecturi de agenti reactivi

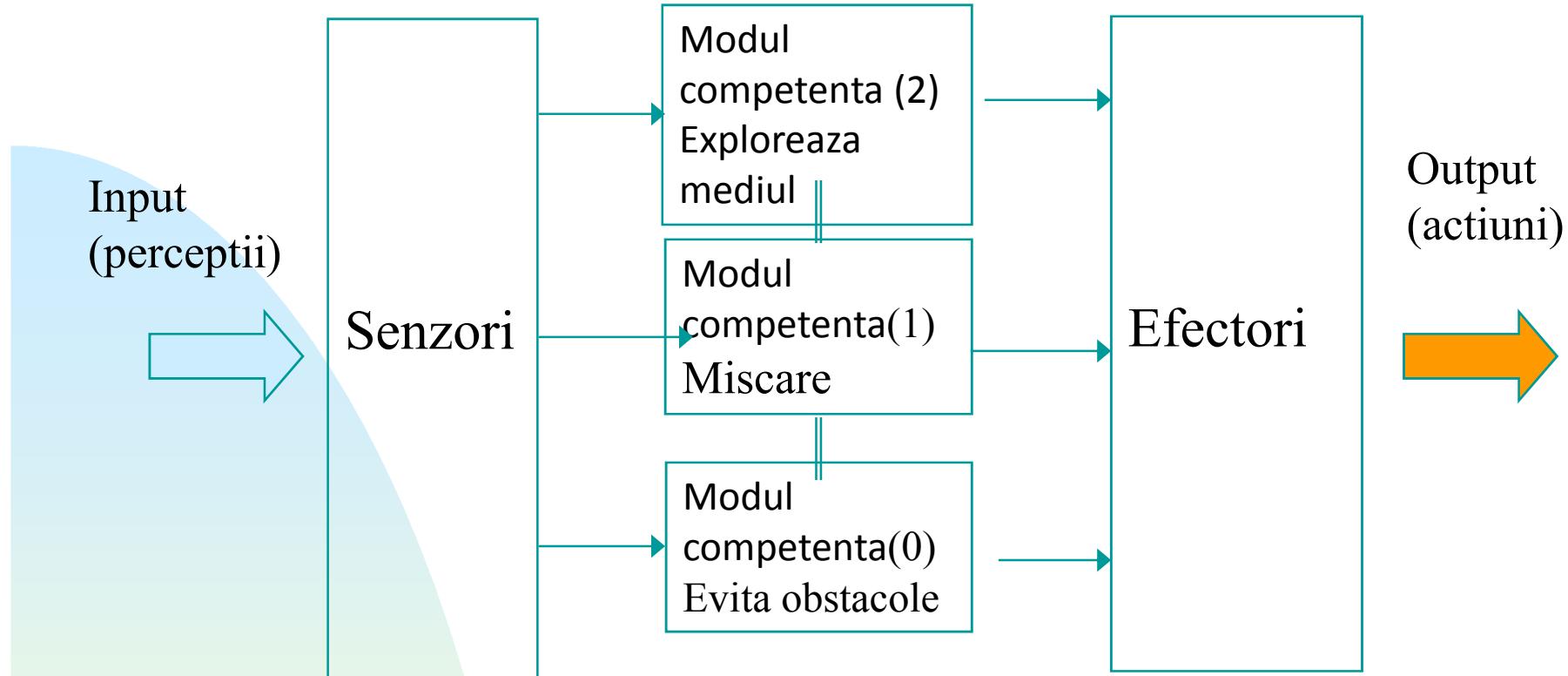
Arhitectura de subsumare - Brooks, 1986

- (1) Luarea deciziilor = {TAB - Task Accomplishing Behaviours}
 - ◆ Fiecare comportare (behaviour) = o functie ce realizeaza o actiune
 - ◆ TAB – automate finite
 - ◆ Implementare: *situatie* → *actiune*
- (2) Mai multe comportari pot fi activate in paralel

Arhitectura de subsumare

- Un TAB este reprezentat de un modul de competenta (c.m.)
- Fiecare c.m. executa un task simplu
- c.m. opereaza in paralel
- Nivele inferioare au prioritate fata de cele superioare
- c.m. la nivel inferior monitorizeaza si influenteaza intrarile si iesirile c.m. la nivel superior

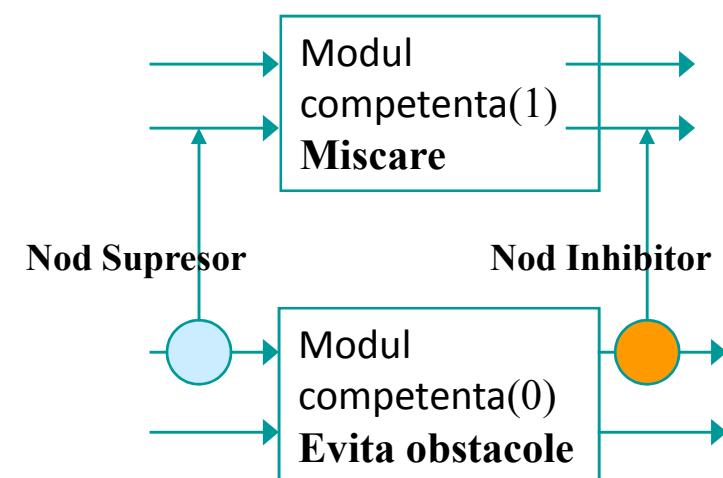
→ *subsumtion architecture*



$M_1 = \text{se misca cu evitarea obstacolelor} \supset M_0$

$M_2 = \text{exploreaza mediul cautand obiecte de interesa aflate in departare while moving around} \supset M_1$

- Incorporarea functionalitatii unui c.m. subordonat de catre un c.m. superior se face prin noduri **supresoare** (modifica semnalul de intrare) si noduri **inhibitoare** (inhiba iesirea)



Comportare

(c, a) – conditie-actiune; descrie comportarea

$R = \{ (c, a) \mid c \subseteq P, a \in A \}$ - multimea reguli de comportare

$\angle \subseteq R \times R$ – relatie binara totala de inhibare

```
function action( p: P)
var fired: P(R), selected: A
begin
    fired = { (c, a) | (c, a) ∈ R and p ∈ c}
    for each (c, a) ∈ fired do
        if  $\neg \exists (c', a') \in \text{fired}$  such that  $(c', a') \angle (c, a)$  then return a
    return null
end
```

Ne aflam pe o planeta necunoscuta care contine aur. Mostre de teren trebuie aduse la nava. Nu se stie daca sunt aur sau nu. Exista mai multi agenti autonomi care nu pot comunica intre ei. Nava transmite semnale radio: gradient al campului

Comportare

- (1) **Daca** detectez obstacol **atunci** schimb directia
- (2) **Daca** am mostre **si** sunt la baza **atunci** depune mostre
- (3) **Daca** am mostre **si** nu sunt la baza **atunci** urmez campul de gradient
- (4) **Daca** gasesc mostre **atunci** le iau
- (5) **Daca** adevarat **atunci** ma misc in mediul

(1) \angle (2) \angle (3) \angle (4) \angle (5)

Agentii pot comunica indirect:

- Depun si culeg boabe radiocative
- Pot seziza aceste boabe radioactive

- (1) **Daca** detectez obstacol **atunci** schimb directia
- (2) **Daca** am mostre **si** sunt la baza **atunci** depune mostre
- (3) **Daca** am mostre **si** nu sunt la baza **atunci** depun boaba radioactiva **si** urmez campul de gradient
- (4) **Daca** gasesc mostre **atunci** le iau
- (5) **Daca** gasesc boabe radioactive **atunci** iau una **si** urmez campul de gradient
- (6) **Daca** adevarat **atunci** ma misc in mediu

- (1) < (2) < (3) < (4) < (5) < (6)