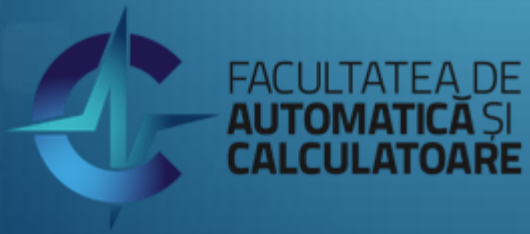
Arhitecturi Paralele
Consistency and Consensus

Prof. Florin Pop
As. Drd. Ing. Cristian Chilipirea
cristian.chilipirea@cs.pub.ro

Elemente preluate din cursul Prof. Ciprian Dobre




POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818

Cum se măsoară viteza unui proces și ce ne spune?

În cât timp va executa următorul program?

```
int main() {
    int i;

    for(i=0; i < 1000 * 1000 * 1000; i++);

    return 0;
}
```

În cât timp va executa următorul program?

```
int main() {
    int i;

    for(i=0; i < 1000 * 1000 * 1000; i++);

    return 0;
}
```

```
real    0m1.783s
user    0m1.766s
sys     0m0.000s
```

Time - precizie

Scientific Notation

Term	Symbol	Multiple	Sci Not^{n}
Terra	T	1 000 000 000 000	x 10^{12}
Giga	G	1 000 000 000	x 10^{9}
Mega	M	1 000 000	x 10^{6}
kilo	k	1 000	x 10^{3}
Units	Eg. meters	1	x 10^{0}
milli	m	1 / 1 000	x 10^{-3}
micro	μ	1 / 1 000 000	x 10^{-6}
nano	n	1 / 1 000 000 000	x 10^{-9}
pico	p	1 / 1000 000 000 000	x 10^{-12}

Timp transmitere pachet?

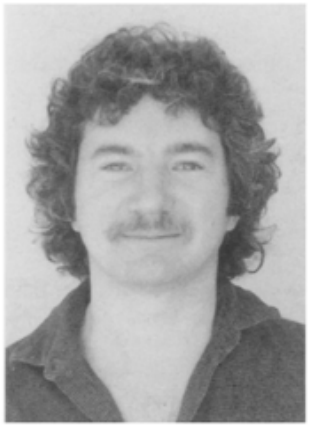
Timp transmitere pachet?

```
root@Nyx:/mnt/d/Dropbox/backupServer/apd-homework/rezults# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=120 time=15.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=120 time=13.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=120 time=15.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=120 time=14.5 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=120 time=32.5 ms
```

Sincronizarea timpilor între 2 sisteme

Probabilistic clock synchronization

Flaviu Cristian
IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA



Flaviu Cristian is a computer scientist at the IBM Almaden Research Center in San Jose, California. He received his PhD from the University of Grenoble, France, in 1979. After carrying out research in operating systems and programming methodology in France, and working on the specification, design, and verification of fault-tolerant programs in England, he joined IBM in 1982. Since then he has worked in the area of fault-tolerant distributed protocols and systems. He has participated in the design and implementation of a highly available system prototype at the Almaden Research Center and has reviewed and consulted for several fault-tolerant distributed system designs, both in Europe and in the American divisions of IBM. He is now a technical leader in the design of a new U.S. Air Traffic Control System which must satisfy very stringent availability requirements.

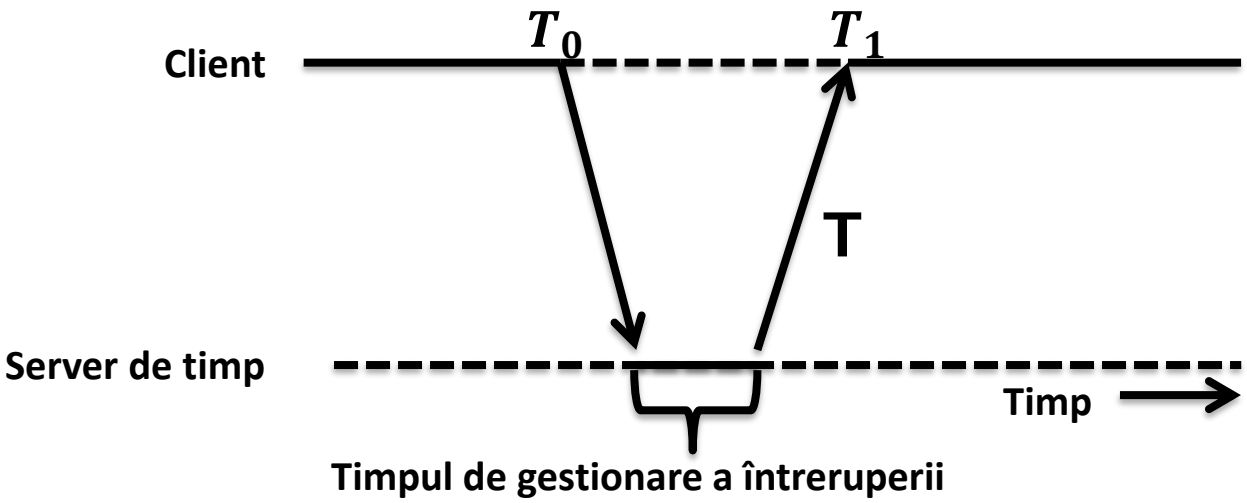
Abstract. A probabilistic method is proposed for reading remote clocks in distributed systems subject to unbounded random communication delays.

in some given maximum derivation from a time reference external to the system. *Internal* clock synchronization keeps processor clocks within some maximum relative deviation of each other. Externally synchronized clocks are also internally synchronized. The converse is not true: as time passes internally synchronized clocks can drift arbitrarily far from external time.

Clock synchronization is needed in many distributed systems. Internal clock synchronization enables one to measure the duration of distributed activities that start on one processor and terminate on another processor and to totally order distributed events in a manner that closely approximates their real time precedence. To allow exchange of information about the timing of events with other systems and users, many systems require external clock synchronization. For example external time can be used to record the occurrence of events for later analysis by humans, to instruct a system to take certain actions when certain specified (external) time deadlines occur, and to order the occurrence of related events observed by distinct sys-

Sincronizarea timpilor între 2 sisteme

T + RTT/2



Cu ce protocol se sincronizează ceasul calculatoarelor?

Cu ce protocol se sincronizează ceasul calculatoarelor?

Internet Engineering Task Force (IETF)
Request for Comments: 5905
Obsoletes: 1305, 4330
Category: Standards Track
ISSN: 2070-1721

D. Mills
U. Delaware
J. Martin, Ed.
ISC
J. Burbank
W. Kasch
JHU/APL
June 2010

Network Time Protocol Version 4: Protocol and Algorithms Sp

Abstract

The Network Time Protocol (NTP) is widely used to synchro
computer clocks in the Internet. This document describes
4 (NTPv4), which is backwards compatible with NTP version
described in RFC 1305, as well as previous versions of th
NTPv4 includes a modified protocol header to accommodate
Protocol version 6 address family. NTPv4 includes fundam
improvements in the mitigation and discipline algorithms
the potential accuracy to the tens of microseconds with m
workstations and fast LANs. It includes a dynamic server
scheme, so that in many cases, specific server configurat
required. It corrects certain errors in the NTPv3 design
implementation and includes an optional extension mechani

Într-un sistem distribuit cum stabilim ordinea operațiilor?

Modelarea unui sistem distribuit?

Modelarea unui sistem simplu/secvențial?

Modelarea unui sistem simplu/secvențial?

First Draft of a Report
on the EDVAC

by

John von Neumann

Modelarea unui sistem simplu/secvențial?

Un set de stări.

Operațiile sau acțiunile modifică starea

Modelarea unui sistem distribuit?

Modelarea unui sistem distribuit?

Programming Techniques — S. L. Graham, R. L. Rivest Editors

Communicating Sequential Processes

C.A.R. Hoare
The Queen's University
Belfast, Northern Ireland

This paper suggests that input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental program structuring method. When combined with a development of Dijkstra's guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.

Key Words and Phrases: programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays

CR Categories: 4.20, 4.22, 4.32

grams, three basic constructs have received widespread recognition and use: A repetitive construct (e.g. the **while** loop), an alternative construct (e.g. the conditional **if..then..else**), and normal sequential program composition (often denoted by a semicolon). Less agreement has been reached about the design of other important program structures, and many suggestions have been made: Subroutines (Fortran), procedures (Algol 60 [15]), entries (PL/I), coroutines (UNIX [17]), classes (SIMULA 67 [5]), processes and monitors (C
(CLU [13]), forms (ALPH

The traditional stored
been designed primarily f
single sequential program
speed has led to the intr
attempt has been made
programmer, either by ha
function units of the CD
in an I/O control packag
erating system). Howeve
technology suggest that a
structed from a number
essors (each with its ow
powerful, capacious, reli
machine which is disguise

In order to use such a
task, the component pro
municate and to synchr
methods of achieving this
adopted method of comm
updating of a common st
and many machine code
severe problems in the co

Într-un sistem distribuit cum stabilim ordinea operațiilor?

Lamport timestamps

Operating Systems — R. Stockton Gaines, Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

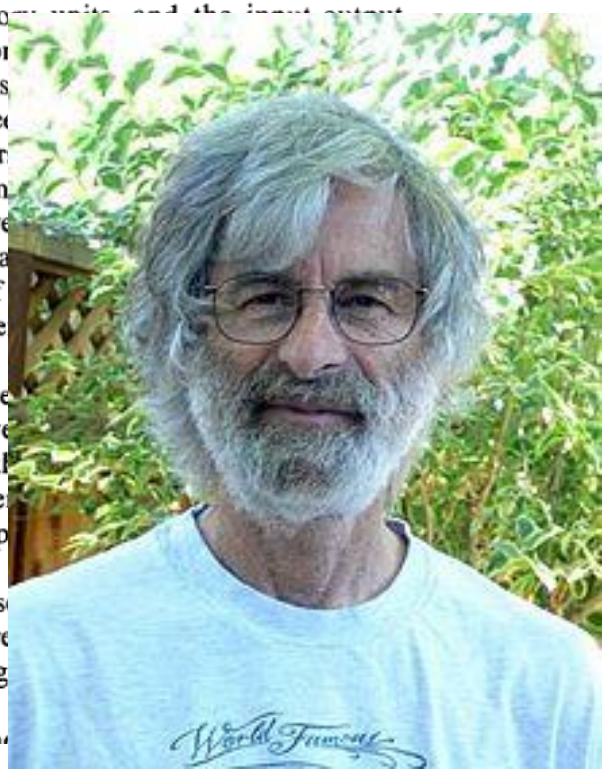
The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

A distributed system consists of a collection of distinct processes which are spatially separated, and which communicate with one another by exchanging messages. A network of interconnected computers, such as the ARPA net, is a distributed system. A single computer can also be viewed as a distributed system in which the central control unit, the memo
channels are separate pr
if the message transmiss
pared to the time betwe

We will concern our
spatially separated com
remarks will apply more
tiprocessing system on a
lems similar to those of
the unpredictable orde
occur.

In a distributed syste
say that one of two eve
"happened before" is th
of the events in the syste
often arise because peop
and its implications.

In this paper, we dis
by the "happened before
algorithm for extending
of all the events. This
mechanism for implem



Ordine parțială

- **Timp relativ** versus **timp real**
- **Ceasuri logice**
- petrecut înainte (→)
  - dacă **a** şi **b** sunt evenimente din acelaşi proces şi **a** îl precede in timp pe **b**,
    - atunci a → b
  - când **a** reprezintă transmiterea unui mesaj de către un proces, iar **b** recepţia
    - aceluiaşi mesaj de către un altul, atunci a → b
  - dacă a → b şi b → c, atunci a → c

Ordine parțială

- Reguli:
  - la producerea unui eveniment intern:
    - cl este incrementat
    - valoarea ceasului logic este asociată evenimentului ca amprentă de timp
  - la transmiterea unui mesaj :
    - incrementează cl al procesului transmitator cu 1
    - actualizează tt := cl
  - la primirea unui mesaj cu amprenta de timp tt:
    - actualizează cl := maxim(cl, tt) + 1

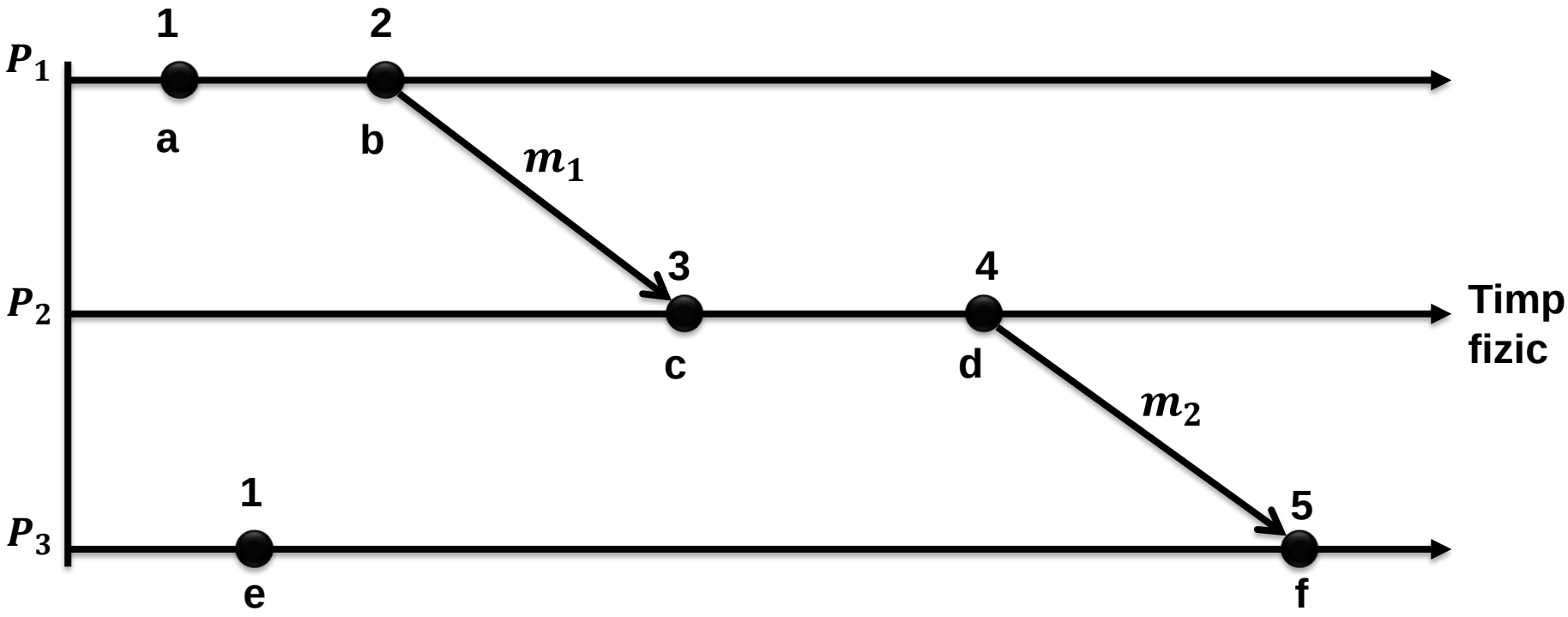
Ordine parțială

- Reguli:
  - la producerea unui eveniment intern:
    - cl este incrementat
    - valoarea ceasului logic este asociată evenimentului ca amprentă de timp
  - la transmiterea unui mesaj :
    - incrementează cl al procesului transmitator cu 1
    - actualizează tt := cl
  - la primirea unui mesaj cu amprenta de timp tt:
    - actualizează cl := maxim(cl, tt) + 1
- Ordonare parțială: $a \rightarrow b \Rightarrow cl(a) < cl(b)$

Ordine parțială

- Reguli:
  - la producerea unui eveniment intern:
    - cl este incrementat
    - valoarea ceasului logic este asociată evenimentului ca amprentă de timp
  - la transmiterea unui mesaj :
    - incrementează cl al procesului transmitator cu 1
    - actualizează tt := cl
  - la primirea unui mesaj cu amprenta de timp tt:
    - actualizează cl := maxim(cl, tt) + 1
- Ordonare parțială: $a \rightarrow b \Rightarrow cl(a) < cl(b)$

Exemplu



Ceasuri logice vectoriale

Cu soluţia Lamport:

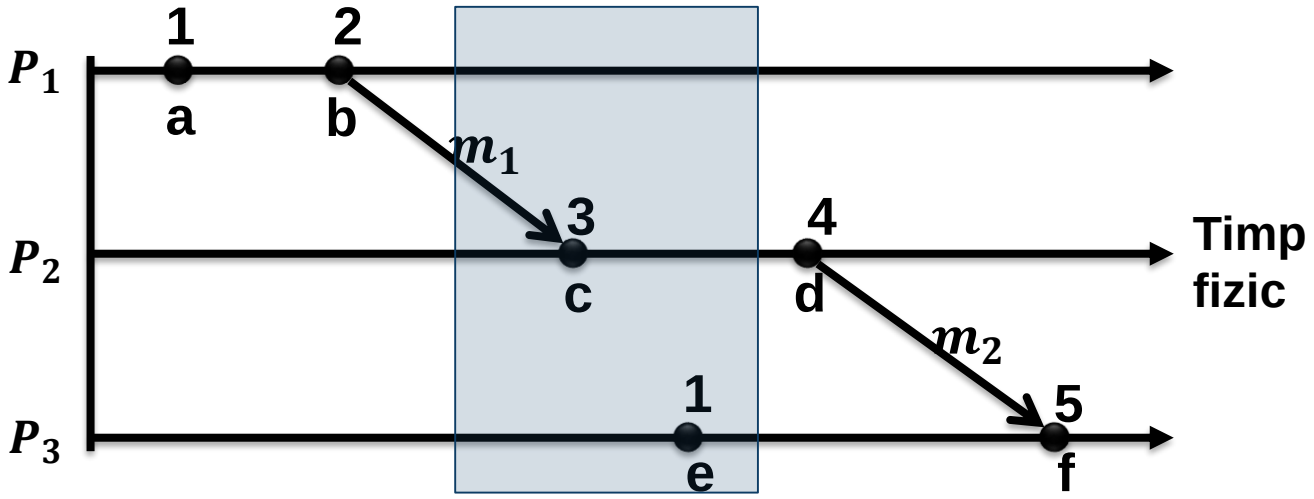
e precede f ⇒ amprenta_logică (e) < amprenta_logică (f)

Dar

amprenta_logică (e) < amprenta_logică (f) ⇏ e precede f

Ex: se poate spune ca **e** precede **c** ?

Soluția: ceasuri logice vectoriale.




POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818

Vector Clocks

Timestamps in Message-Passing Systems That Preserve the Partial Ordering

Colin J. Fidge
Department of Computer Science, Australian National University, Canberra, ACT.

ABSTRACT

Timestamping is a common method of totally ordering events in concurrent programs. However, for applications requiring access to the global state, a total ordering is inappropriate. This paper presents algorithms for timestamping events in both synchronous and asynchronous message-passing programs that allow for access to t herent in a parallel system. The algorithms do not change the cor require a central timestamp issuing authority.

Keywords and phrases: concurrent programming, message-passing, timesta
CR categories: D.1.3

INTRODUCTION

A fundamental problem in concurrent programming is determining the different processes occurred. An obvious solution is to attach a number repres a permanent record of the execution of each event. This assumes that each proc clock, but practical parallel systems, by their very nature, make it difficult to

Vector Clocks

Virtual Time and Global States of Distributed Systems *

Friedemann Mattern †

Department of Computer Science, University of Kaiserslautem
D 6750 Kaiserslautern, Germany

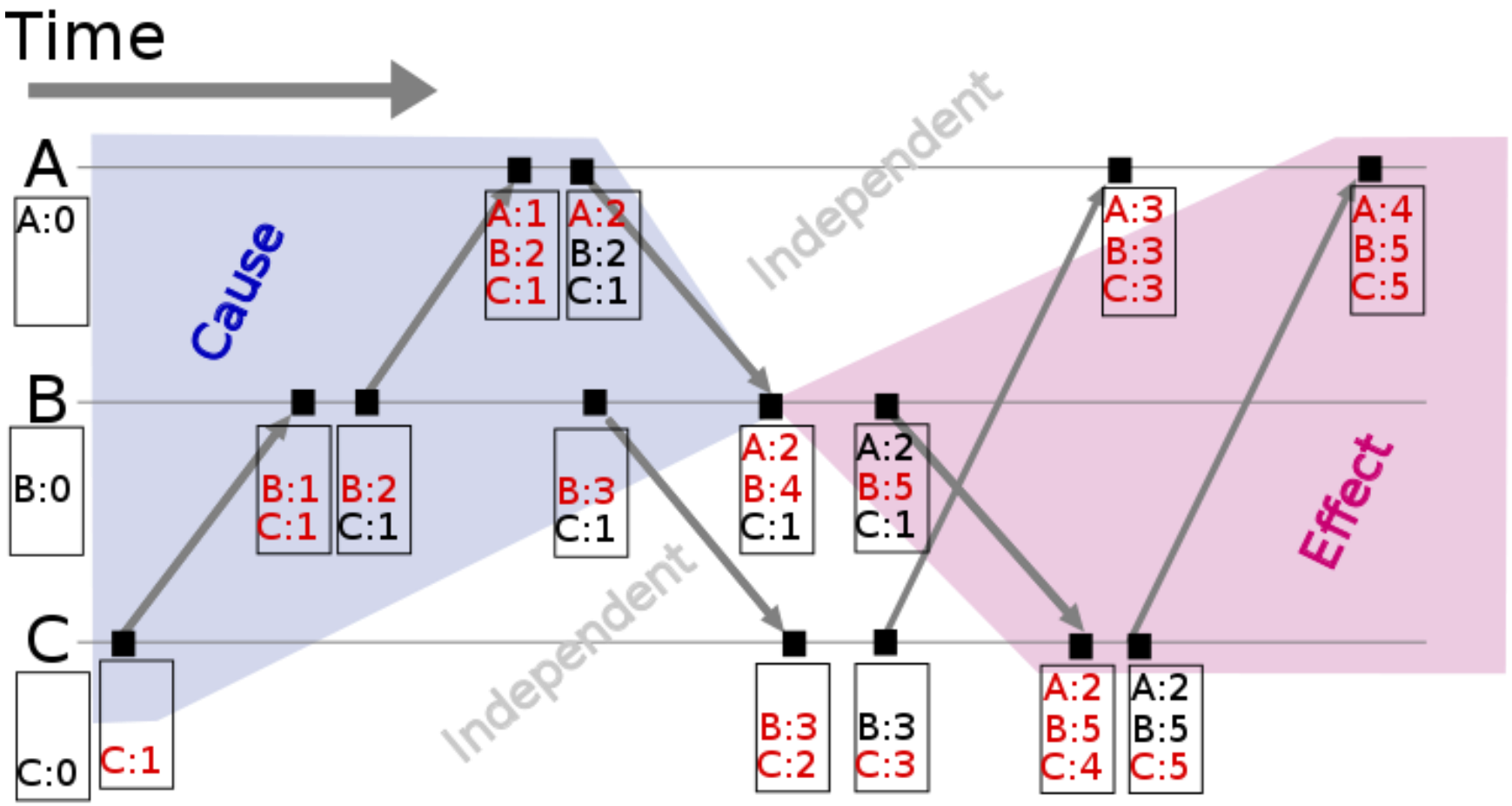
Abstract

A distributed system can be characterized by the fact that the global state is distributed and that a common time base does not exist. However, the notion of time is an important concept in every day life of our decentralized "real world" and helps to solve problems like getting a consistent population census or determining the potential causality between events. We argue that a linearly ordered structure of time is not (always) adequate for distributed systems and propose a generalized non-standard model of time which consists of vectors

view of an idealized
access to all process

The fact that *a*
view of the global s
not exist is the cau
tributed systems.
and database syste
detection, and conc
cult to solve in a dis
sical centralized env
of distributed contr
found to be wrong.
centralized systems

Vector Clocks

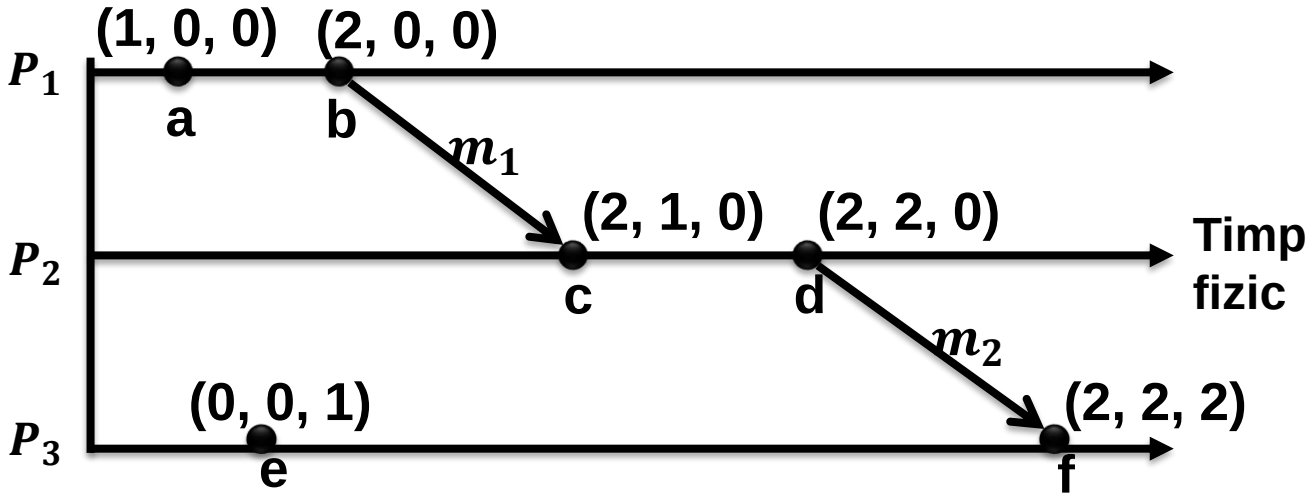


Ceasuri logice vectoriale

- Fiecare P_i are asociat un tablou $V_i[1..n]$ în care:
  - $V_i[i]$ este numărul de evenimente produse în procesul P_i
  - $V_i[j]$ este numărul de evenimente despre care P_i ştie *(a aflat)* că au avut loc la P_j.
- Procesul P_i actualizează V_i la fiecare **eveniment** din P_i
  - e.g. pentru procesorul 3, (1,2,1,3) → (1,2,2,3)
- Când P_i **transmite** mesajul *m* *(eveniment send m):*
  - P_i incrementează $V_i[i]$
  - P_i adaugă V_i la *m* ca vector de amprente de timp curent $\boldsymbol{vt_m}$
- Când P_j **primeşte** *m* și $\boldsymbol{vt_m}$ *(eveniment receive m):*
  - P_j ajustează: $V_j[k] = max\{V_j[k], vt_m[k]\}$ pentru fiecare k

    e.g., P_2 primește un mesaj cu timpul (3,2,4) iar timpul curent al lui P_2 este (3,4,3), atunci P_2 ajusteaza timpul la (3,4,4)
  - P_i incrementează $V_j[i]$cu 1

Ceasuri logice vectoriale



Aplicarea regulilor ceasurilor logice vectoriale

Reguli:

- $VT_1 = VT_2 \Leftrightarrow VT_1[i] = VT_2[i]$, pentru $i = 1, \dots, N$
- $VT_1 \leq VT_2 \Leftrightarrow VT_1[i] \leq VT_2[i]$, pentru $i = 1, \dots, N$
- $VT_1 < VT_2 \Leftrightarrow VT_1[i] \leq VT_2[i]$, și $VT_1 <> VT_2$ *(de exemplu (1,2,2) < (1,3,2))*

Fie vt(a) şi vt(b) vectorii de amprente de timp asociaţi ev. a şi b. Atunci:

vt(a) < vt(b) $\Rightarrow$ **evenimentul a precede cauzal b**

vt(a) $\not<$ **vt(b) and vt(a)** $\not>$ **vt(b) and vt(a)** $\neq$ **vt(b)** $\Leftrightarrow$ **evenimentele a şi b sunt concurente**

Aplicație: Ordonare Cauzală Multicast

- Procesele unei colecţii P comunică între ele doar prin mesaje cu difuzare
- Se cere ca mesajele să respecte **dependenţa cauzală**

$$m \rightarrow m' \Rightarrow livrarep(m) \rightarrow livrarep(m')$$

Protocolul *(vectori de timp)*:

- fiecare proces $P_i = (i = 1..n)$ are asociat un vector $V_i[1..n]$, cu toate elementele iniţial 0
- se numara doar operatiile de transmitere de mesaje
- $V_i[i]$, este nr ev. transmitere de mesaje produse de P_i;
- $V_i[j]$ este nr ev. transmitere de mesaje despre care $\boldsymbol{P_i}$ ştie (a aflat) că au avut loc la P_j.
- Procesul P_i actualizează V_i la fiecare eveniment de trimitere sau recepţie de mesaj din P_i.

Aplicație: Ordonare Cauzală Multicast

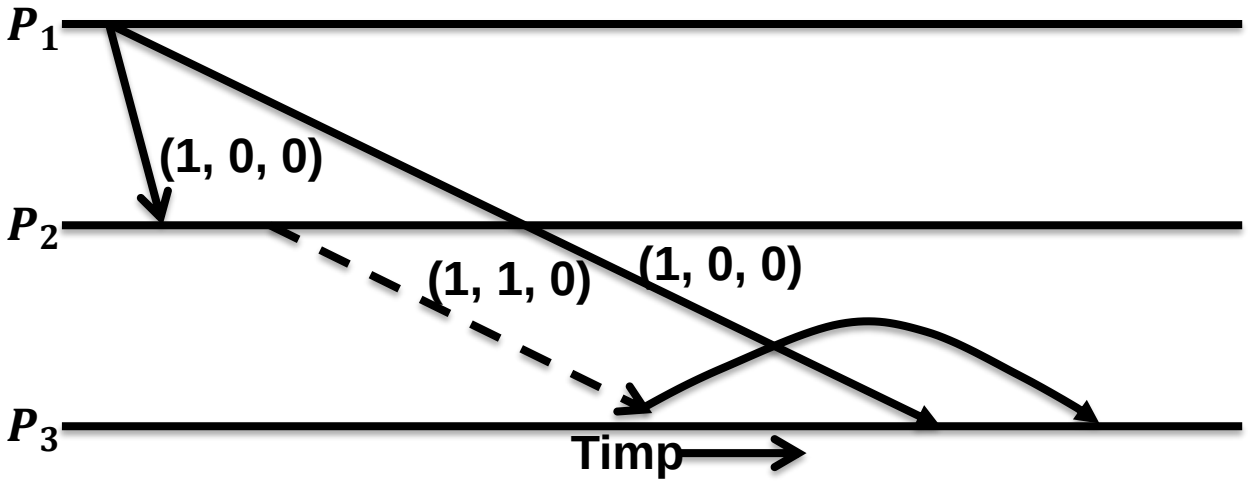
- Când P_S **transmite** mesajul *m*:
  - $\boldsymbol{P_S}$ **incrementează** $\boldsymbol{V_S[s]}$
  - $\boldsymbol{P_S}$ **adaugă** $\boldsymbol{V_s}$ **la** ***m*** **ca vector de amprente de timp curent** $\boldsymbol{vt_m}$
    - **Obs: Pentru ordonare cauzală, incrementarea lui $V_s[s]$ se face doar la transmiterea de mesaje de către s**
    - vt_m spune receptorului câte evenimente (din alte procese) au precedat **m** şi ar putea influenţa cauzal pe **m**.

- Când P_d **primeşte** mesajul *m* împreună cu vt_m, mesajul este păstrat într-o coadă de **întârziere** şi este **livrat** doar dacă:
  - $\boldsymbol{vt_s[s] = V_d[s] + 1}$ (acesta este următorul timestamp pe care **d** îl așteaptă de la **s**)
  - $\boldsymbol{vt_m[k] \leq V_d[k]\ pentru\ k <> s}$ (**d** a văzut toate mesajele ce au fost văzute de **s** la momentul când a trimis mesajul **m**)

- Când mesajul este livrat, V_d este actualizat conform regulilor vectorilor de timp:

  $\boldsymbol{V_d[k] = max\{V_d[k],\ vt_m[k]\}}$ **pentru fiecare** $\boldsymbol{k = 1, n.}$

Aplicație: Ordonare Cauzală Multicast



Alte acţiuni ale protocolului la livrarea mesajelor:

- dacă $vt_m[k] > V_d[k]$ pentru un oarecare k atunci se întârzie m
  P_S a primit mesaje de care mesajul curent poate fi cauzal dependent, dar pe care P_d încă nu le-a primit;
- dacă $vt_m[s] > V_d[s] + 1$ atunci se întârzie m
  mai sunt mesaje de la P_S pe care P_s nu le-a primit
  (asigură ordinea FIFO pentru canale nonFIFO)
- dacă $vt_m[s] > V_d[s]$ atunci rejectează m
  m este un duplicat al unui mesaj primit anterior


POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818

Byzantine Generals

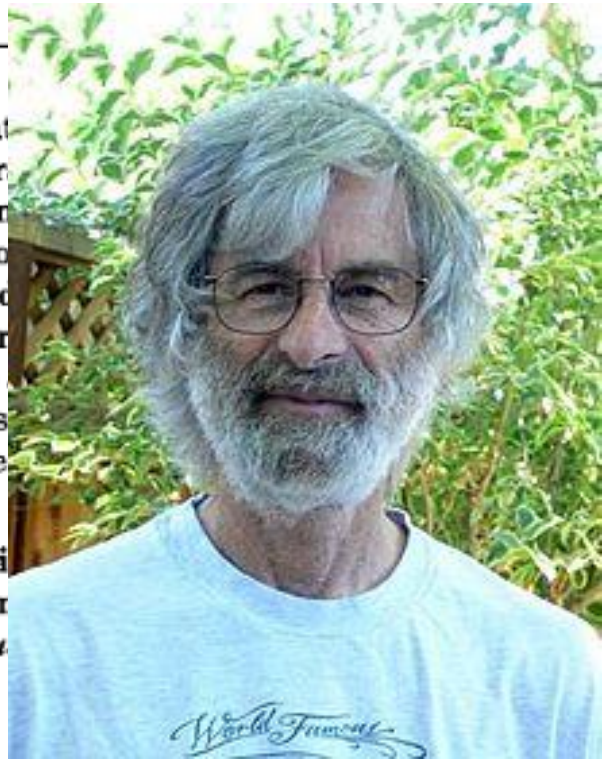
The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that
to different parts of the system. This situation can be expressed abstr
generals of the Byzantine army camped with their troops around an ener
by messenger, the generals must agree upon a common battle plan. Ho
may be traitors who will try to confuse the others. The problem is to find
the loyal generals will reach agreement. It is shown that, using only or
solvable if and only if more than two-thirds of the generals are loyal; so
two loyal generals. With unforgeable written messages, the problem is
generals and possible traitors. Applications of the solutions to reliable
discussed.

Categories and Subject Descriptors: C.2.4. [**Computer-Communicati**
Systems—*network operating systems*; D.4.4 [**Operating Systems**]: Com
network communication; D.4.5 [**Operating Systems**]: Reliability—*fau*

General Terms: Algorithms, Reliability



Tipuri de defecte în sistemele distribuite

- Valorea comună poate *să nu fie atinsă* din diferite cauze
  - pierderea mesajelor în mediul de comunicație
  - procesele pot produce rezultate greșite
- Tipuri de defecte ale proceselor
  - ***crash***: procesul devine nefuncțional
  - ***byzantine***: procesul trimite mesaje cu un conținut arbitrar
- O analogie pentru astfel de situații este *Problema Generalilor Bizantini*

Problema generalilor bizantini

- Câteva divizii ale *Armatei bizantine*, fiecare sub comanda unui *General*, înconjoară inamicul
- Generalii bizantini trebuie să ajungă la o înțelegere în privința atacului
- Este crucial să se ajungă la o înțelegere, deoarece doar un atac simultan din partea tuturor diviziilor poate conduce la o victorie
- Diviziile sunt dispersate geografic, așa că Generalii își comunică între ei observațiile asupra inamicului prin intermediul mesagerilor

- Astfel, cauzele care pot îngreuna stabilirea unei înțelegeri pot fi:
  - Mesagerii pot fi prinși de către inamic – așadar, mesajele nu ajung la destinație
  - Generalii pot fi ***trădători***, încercând să deruteze Generalii ***loiali***

- În continuare, vom presupune că mesagerii ajung la destinație

Problema generalilor bizantini

Generalii au nevoie de un algoritm care să garanteze că:

A: Toți generalii loiali decid același plan de acțiune.

Nu e suficient: vrem același plan pentru toți, însă dorim ca și planul să fie unul adecvat.

B: Un număr mic de trădători nu pot influența decisiv Generalii loiali.

Problema generalilor bizantini

- Fie $v(i)$ – informația comunicată de către Generalul cu numărul *i*
- Fiecare General folosește o metodă pentru a combina valorile $v(1)$, $v(2)$, …, $v(n)$ într-un plan de acțiune

- Condiția *A* este îndeplinită prin faptul că toți generalii folosesc aceeași metodă de a combina cele *n* valori
- Condiția *B* este îndeplinită prin asigurarea că metoda folosită este una robustă

- Dacă decizia care poate fi luată este *atac* sau *retragere*, $v(i)$ este opțiunea Generalului *i* dintre cele două variante, iar decizia finală pentru fiecare General se bazează pe votul majoritar (dintre cele *n* valori)

Problema generalilor bizantini

- Generalii își comunică unii altora valorile $v(i)$
- Generalii trădători pot trimite valori diferite celorlalți Generali
- Satisfacerea condiției *A* presupune ca fiecare General să ia decizia finală bazată pe aceeași mulțime $v(1)$, $v(2)$, ..., $v(n)$

- Astfel, pentru îndeplinirea condiției *A*:

1. Fiecare General loial trebuie să obțină aceleași informații $v(1)$, ..., $v(n)$.

Problema generalilor bizantini

1. Fiecare General loial trebuie să obțină aceleași informații $v(1)$, …, $v(n)$.

- Condiția *1* implică faptul că un General poate să nu folosească valorea $v(i)$ obținută direct de la Generalul *i*, de vreme ce un General trădător poate să trimită valori diferite
- Astfel, se introduce riscul ca un General să nu folosească o valoare $v(i)$, deși Generalul *i* este unul loial

- Se introduce următoarea cerință:

2. Dacă Generalul *i* este loial, atunci valoarea trimisă de el trebuie să fie folosită de către fiecare General loial.

- Condiția 1 se poate rescrie astfel:

1'. Oricare doi Generali loiali folosesc aceeași valoare pentru $v(i)$.

Problema generalilor bizantini

- Restrângem abordarea problemei la modul în care *un* General își trimite valoarea celorlalți Generali
- Termenii problemei se schimbă: un General comandant trimite ordinul Locotenenților, obținându-se astfel problema:

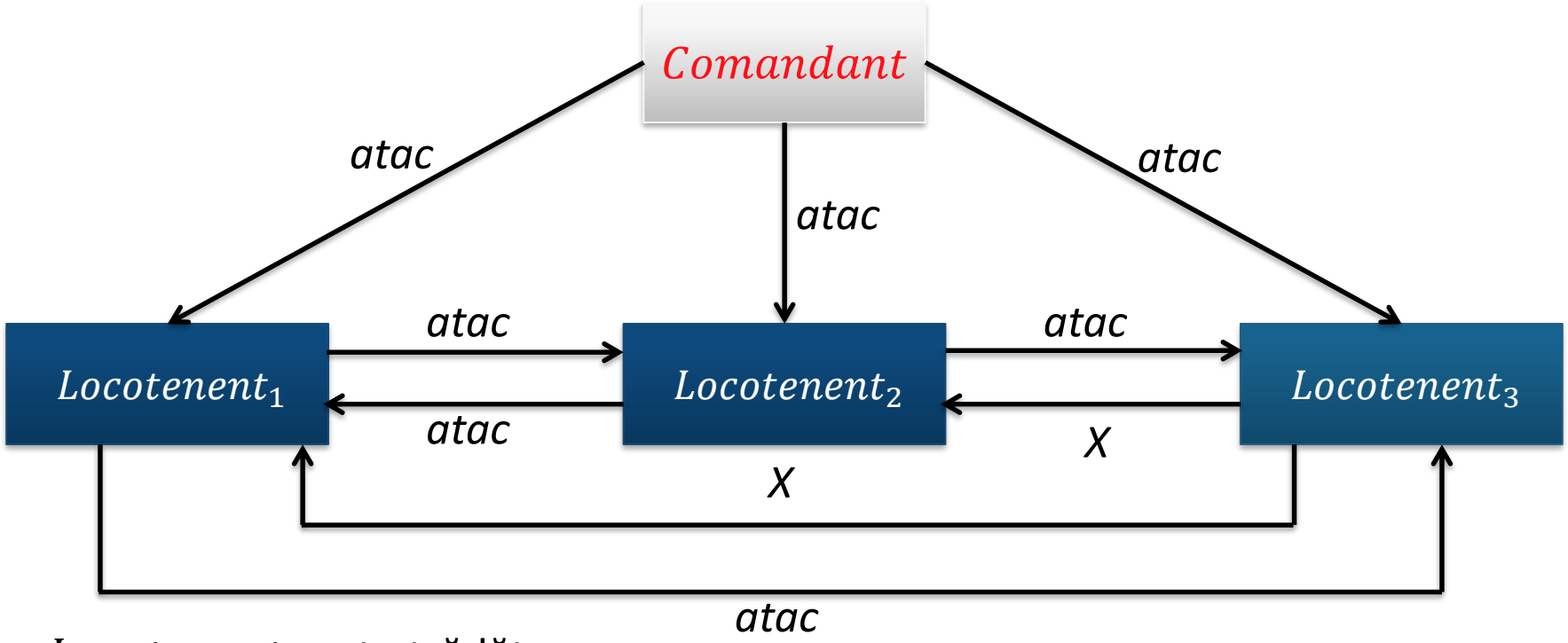
- ***Problema Generalilor Bizantini***:
  - Un General comandant trebuie să trimită un ordin celor n – 1 Locotenenți astfel încât:
  - IC_1: Toți Locotenenții loiali se supun aceluiași ordin
  - IC_2: Dacă Generalul comandant este loial, atunci fiecare Locotenent loial se supune ordinului Generalului comandant

- IC_1, IC_2 – *interactive consistency conditions*
- Se observă că, pentru un Comandat loial, IC_1 rezultă din IC_2; totuși, Comandantul nu este în mod necesar loial

Problema generalilor bizantini

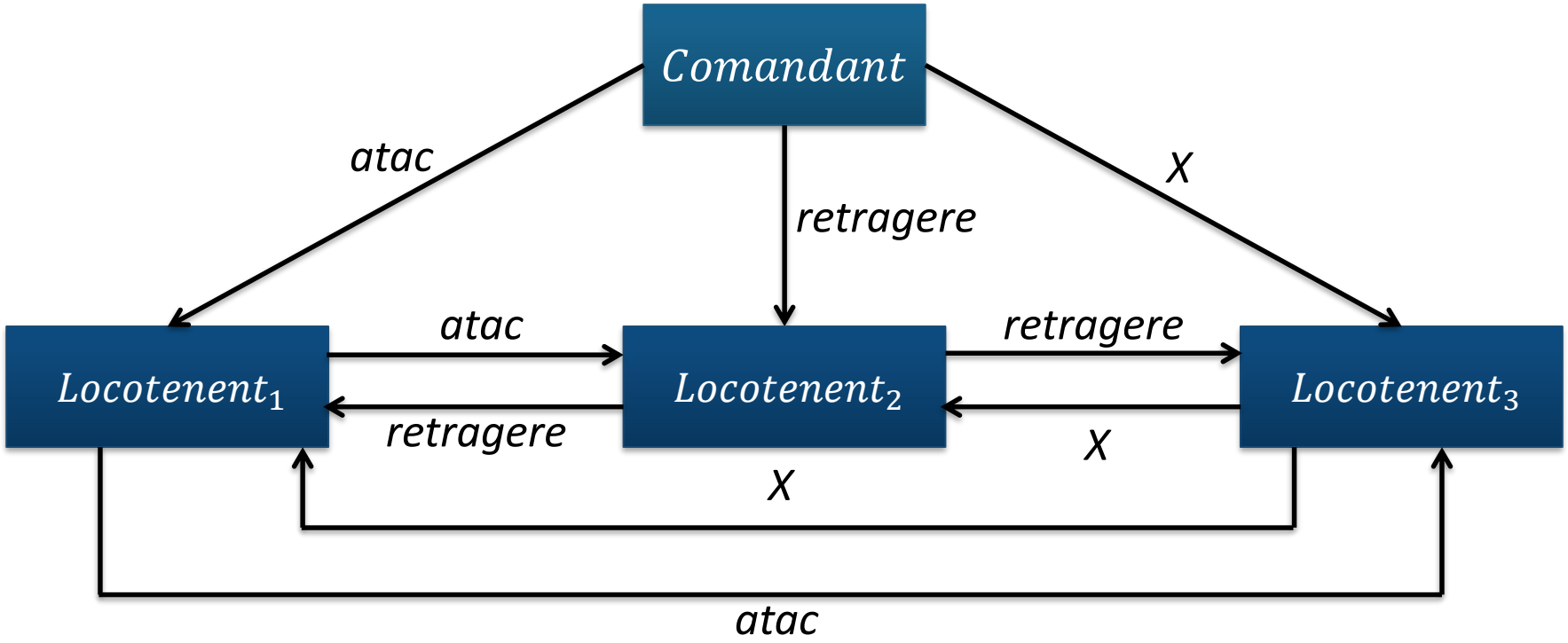
- Evenimentele sunt următoarele:
  - Comandantul trimite ordinul tuturor Locotenenților
  - Un Locotenent trimite celorlalți Locotenenți mesajul primit de la Comandant
  - După primirea mesajului de la Comandant și a copiilor de la ceilalți Locotenenți, un locotenent decide *prin vot majoritar* ce decizie va lua

Problema generalilor bizantini
Exemplul 1



- $Locotenent_3$ este trădător
- $X \in \{atac, retragere\}$
- Oricare ar fi mesajul transmis de trădător, cei doi Locotenenți loiali vor lua aceeași decizie (*atac*)

Problema generalilor bizantini
Exemplul 2

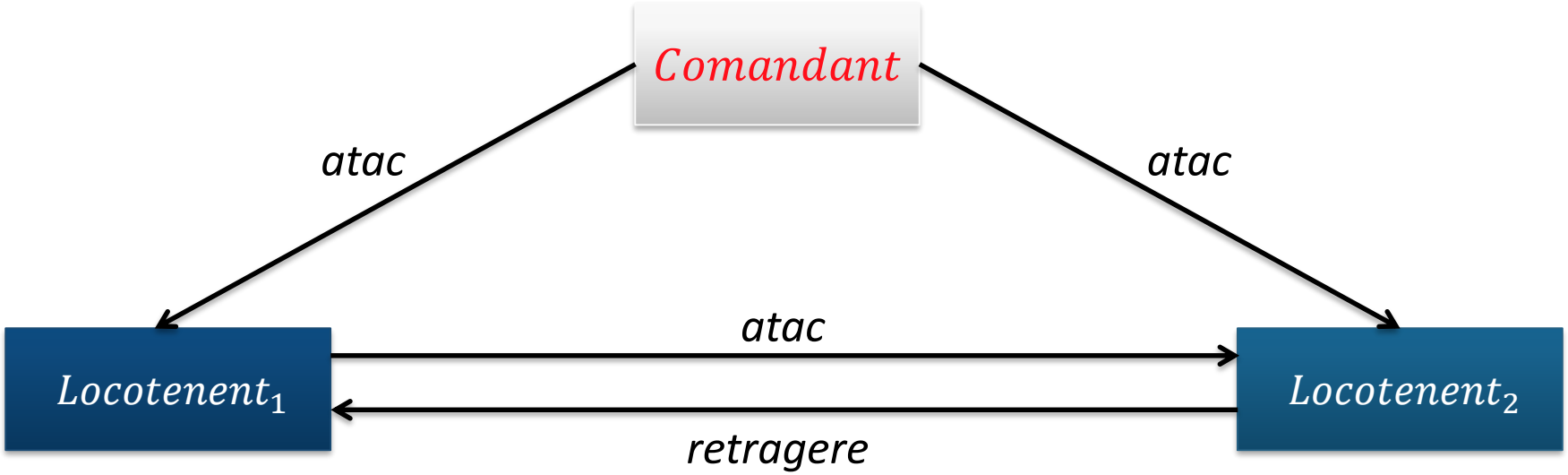


- $Comandant$ este trădător
- $X \in \{atac, retragere\}$
- Oricare ar fi mesajul transmis de trădător, toți Locotenenții vor lua aceeași decizie X

Problema generalilor bizantini

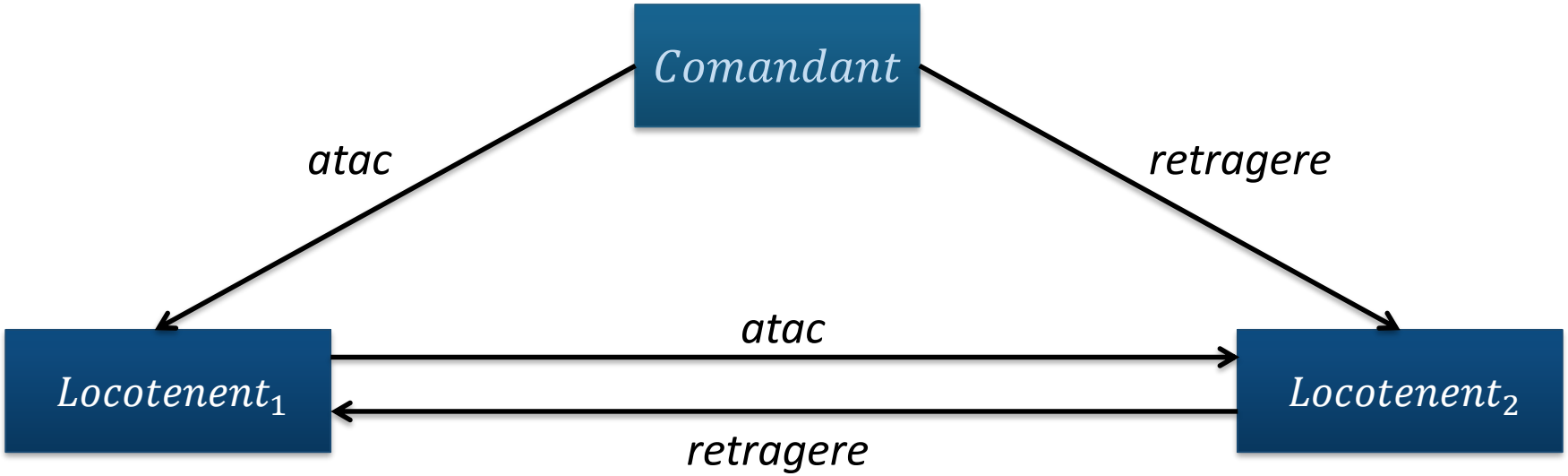
- Dificultatea problemei constă în faptul că:
  - Dacă Generalii (Locotenenții) pot trimite doar mesaje orale, atunci nu există soluție decât pentru cazul în care *2/3 din Generali sunt loiali*
- Un mesaj *oral* este aflat complet sub controlul emițătorului, deci un trădător poate trimite orice mesaj
- Mesajul trimis este *atac* / *retragere*
- Următoarele exemple: nu există soluție pentru 3 Generali, din care un trădător

Problema generalilor bizantini
Exemplul 3



- $Locotenent_2$ este trădător

Problema generalilor bizantini
Exemplul 4



- $Comandant$ este trădător

Problema generalilor bizantini
Soluția cu mesaje orale

- Mesajele orale îndeplinesc următoarele condiții:
  - A_1: Fiecare mesaj trimis ajunge corect la destinație
  - A_2: Receptorul mesajului cunoaște autorul mesajului
  - A_3: Absența unui mesaj poate fi detectată

- Fiecare General (Locotenent) poate trimite mesaje oricărui alt General
- $A_1, A_2 \rightarrow$ un trădător nu intervine în comunicarea dintre alți doi Generali
- $A_3 \rightarrow$ un trădător nu poate influența decizia prin a nu trimite un mesaj
- Un Comandant trădător poate decide să nu trimită ordine, așa că este nevoie de o decizie implicită pentru Locotenenți
- Decizia implicită este: *retragere*

Problema generalilor bizantini
Soluția cu mesaje orale

- Se definește algoritmul pentru soluția cu mesaje orale: *OM(m)*
- *OM(m)* rezolvă Problema Generalilor Bizantini pentru minim *3m + 1* Generali și cel mult *m* trădători

- Algoritmul folosește funcția *majority*$(v(1), \ldots, v(n-1))$
  - Dacă majoritatea valorilor $v(i)$ este v, atunci *majority* este v
- Funcția *majority*$(v(1), \ldots, v(n-1))$ va returna:
  - Valoarea majoritară din $v(i)$, dacă aceasta există; altfel, se returnează *retragere*
  - Mediana valorilor $v(i)$, presupunând că mulțimea este sortată

Problema generalilor bizantini
Soluția cu mesaje orale

- ***OM(0)***

  1) Comandantul își trimite valoarea tuturor Locotenenților.

  2) Fiecare Locotenent folosește valoarea primită de la Comandant, sau *retragere* dacă nu primește nimic.

- ***OM(m), m > 0***

  1) Comandantul își trimite valoarea tuturor Locotenenților.

  2) Pentru fiecare *i*, fie $v(i)$ valoarea primită de Locotenentul *i* de la Comandant (sau valorea implicită – *retragere*). Locotenentul *i* ia rolul de Comandant în *OM(m – 1)*, pentru a trimite valorea celorlalți Locotenenți.

  3) Pentru fiecare *i*, pentru fiecare *j*, $j \neq i$, fie $v(j)$ valoarea pe care Locotenentul *i* a primit-o de la Locotenentul *j* în pasul *2)* (*OM(m – 1)*) sau valoarea implicită.

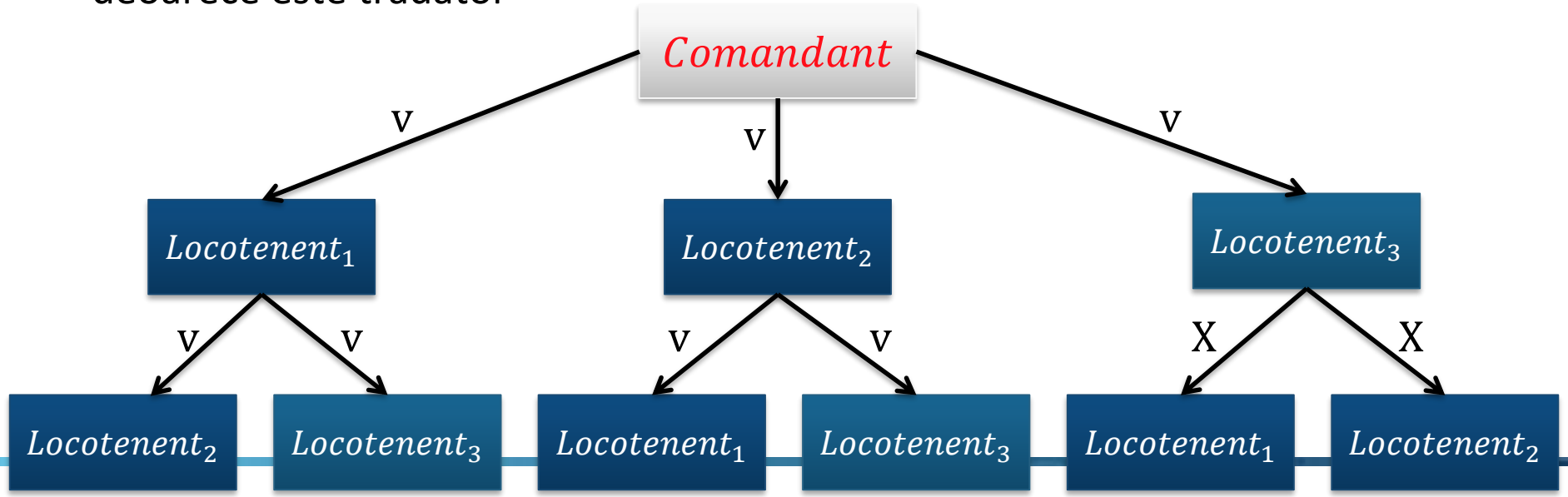
     Locotenentul *i* folosește *majority*$(v(1), \ldots, v(n-1))$.

Soluția cu mesaje orale Exemplu

- m = 1, n = 4 → 1 Comandant, 3 Locotenenți (Locotenentul cu nr.3 = trădător)
- *OM(1)*
  - Comandantul trimite valoarea v
  - Pentru fiecare *i*
    - $v(1) = v$ $\quad Locotenent_1 = \{v\}$
    - *OM(0)* $\quad Comandant = Locotenent_1$
      - Comandantul își trimite valorea v
        - $Locotenent_2 = \{v\}$
    - $v(2) = v$ $\quad Locotenent_2 = \{v, v\}$
    - *OM(0)* $\quad Comandant = Locotenent_2$
      - Comandantul își trimite valorea v
        - $Locotenent_1 = \{v, v\}$
    - $v(3) = v$
    - *OM(0)* $\quad Comandant = Locotenent_3$
      - Comandantul își trimite valorea X
        - $Locotenent_1 = \{v, v, X\}$
        - $Locotenent_2 = \{v, v, X\}$

Soluția cu mesaje orale Exemplu

- Pentru fiecare *i*
  - $Locotenent_1$ folosește *majority*(v, v, X) = v
  - $Locotenent_2$ folosește *majority*(v, v, X) = v
  - $Locotenent_3$ este trădător
- Observație: parcursul pentru cel de-al treilea Locotenent nu a fost urmărit, deoarece este trădător



Soluția cu mesaje orale Complexitate

- Complexitate:
  - Pas 1: $(n-1)$ x $OM(m-1)$ mesaje
  - Pas 2: $(n-1)$ x $(n-2)$ x $OM(m-2)$ mesaje
  - Pas 3: $(n-1)$ x $(n-2)$ x $(n-3)$ x $OM(m-3)$ mesaje
  - Deci, în pasul k: $(n-1)$ x $(n-2)$ x … x $(n-k)$ x $OM(m-k)$ mesaje
    - $k = m$: $(n-1)$ x $(n-2)$ x … x $(n-m)$ x $OM(0)$
    - Pasul m + 1: se trimit $(n-1)$ x $(n-2)$ x … $(n-m-1)$ mesaje
    - Număr total de mesaje: $O(n^{m+1})$

Corectitudine

LEMA 1. Pentru orice m şi k, UM(n,m) satisface IC2 dacă numărul n de generali este mai mare de 2k+m şi sunt cel mult k trădători.

TEOREMA 1. Pentru orice m, algoritmul UM(n,m) satisface condiţiile IC1 şi IC2 dacă numărul de generali n este mai mare de 3m şi sunt cel mult m trădători.

IC1. Toţi locotenenţii loiali se supun aceluiaşi ordin.

IC2. Dacă comandantul este loial, atunci fiecare locotenent loial se supune ordinului transmis de acesta.

ipoteză: n > 2k + m ➔ n - 1 > 2k + (m - 1)

ipoteza inducţie satisfacuta ➔ fiecare locotenent loial obţine vj = v pentru fiecare locotenent loial j.

Sunt cel mult k trădători şi n - 1 > 2k + (m - 1) >= 2k ➔ majoritatea dintre n-1 locotenenţi sunt loiali

Algoritmul UM(n,m), m > 0.

(1) Comandantul trimite valoarea sa fiecărui l

➔ fiecare locotenent loial obţine o majoritate de valori vj = v ➔ în pasul (3) se obţine *majority(v1 . . . . , vn-1)* = v ➔ IC2 satisfacuta

Lema se refera la IC2 (comandant loial). Dem inductie. Pentru m=0: comandant loial trimite v; fiecare mesaj transmis este livrat corect (cf. propr. A1) ➔ UM(n,0) satisface IC2

ceilalţi n - 2 locotenenţi folosind *UM(n - 1,m - 1)*

Pp. proprietatea IC2 indeplinită ptr. m-1, m>0 şi probăm ptr m.

În pas (1), comandant loial trimite v celor n-1 locotenenţi.

În (2), fiecare locotenent loial aplică UM(n-1,m-1) cu n-1 generali.

majority (v1 . . . . . vn-1).

Algoritmul UM(n,0).

TEOREMA 1. Pentru orice m, algoritmul UM(n,m) satisface condiţiile IC1 şi IC2 dacă numărul de generali n este mai mare de 3m şi sunt cel mult m trădători.

ef

Presupunem teorema adevărată ptr m-1; probăm pentru m > 0.

Comandant loial: punem k=m în Lema 1 (n>2k+m ~ n>3m) ➔ UM(n,m) satisface IC2, iar IC2 ➔IC1

locotenent loial se supune ordinului transmis de el).

(2) **For each** Locotenent i

fie *vi* valoarea primită de la comandant, sau Vdef dacă nu primeste nici o valoare.

Dem IC1 ptr. comandant trădător:

Max. m trădători şi comandant trădător ➔ max. m-1 locotenenţi trădători.

Sunt > 3m generali ➔ > 3m-1 locotenenţi. Deoarece 3m-1 > 3(m - 1) ➔ ipoteza de inducţie satisfacuta ➔ UM(n-1,m-1) satisface condiţiile IC1 şi IC2.

➔ ptr. fiecare j, orice doi locotenenţi loiali obţin aceeaşi valoare vj în pasul (3).

➔ obţin aceleaşi valori v1 . . . vn-1 ➔ calculeaza aceeaşi valoare majority(v1 . . . vn-1) în pasul (3)

Soluția cu mesaje semnate

- Se adaugă condiția:
  - A_4
    a) Semnătura unui general loial nu poate fi falsificată și orice alterare a mesajelor sale semnate poate fi detectată.
    b) Oricine poate verifica autenticitatea unei semnături.
- Trădătorii își pot falsifica semnăturile între ei
- Algoritmul cu mesaje semnate rezolvă problema pentru orice număr de Generali

Soluția cu mesaje semnate

- Desfășurare algoritm:
  - Comandatul trimite un mesaj semnat tuturor Locotenenților
  - Fiecare Locotenent își adaugă semnătura și trimite mesajul celorlalți Locotenenți

- Utilitatea semnăturii (de exemplu):
  - Comandatul este trădător și trimite mesaje diferite Locotenenților
  - Locotenenții, prin intermediul semnăturii, pot observa că mesajele diferite primite sunt cauzate chiar de Comandant – două mesaje diferite conțin semnătura Comandantului

Soluția cu mesaje semnate

- Algoritmul folosește o funcție *choice* care este aplicată unei mulțimi de ordine – pentru a se obține un *singur ordin*

- Cerințe:
  - Dacă mulțimea V conține un singur ordin v, atunci $choice(V) = v$
  - $choice(\emptyset) = RETRAGERE$, $\emptyset$ este mulțimea vidă

- O posibilă definiție pentru *choice* este elementul median al mulțimii V – presupunând că mulțimea este ordonată

Soluția cu mesaje semnate

- Notații:
  - x : i – valoarea x semnată de Generalul *i*
    - v : j : i – valoarea v semnată de Generalul *j*, apoi de Generalul *i*
  - Generalul *0* este Comandantul
  - V_i – mulțimea de ordine primite (corect semnate) de către Generalul *i*
    - Pentru un Comandant loial, această mulțime conține un singur element (Locotenenții loiali pot recunoaște mesajele false introduse de trădător)

- A nu se confunda *mulțimea de ordine* V_i cu mulțimea de mesaje primite de un General (mai multe mesaje pot conține același ordin)

Soluția cu mesaje semnate
Algoritmul *SM(m)*

- $V_i = \emptyset$
- Comandatul își semnează valoarea și o trimite fiecărui Locotenent
- Pentru fiecare *i*:
  - Dacă Locotenenentul *i* primește un mesaj de forma $\mathrm{v} : 0$ de la Comandant și mulțimea sa de ordine este vidă:
    - $V_i = v$
    - trimite $\mathrm{v} : 0 : \mathrm{i}$ celorlalți Locotenenți
  - Dacă Locotenenentul *i* primește un mesaj de forma $\mathrm{v} : 0 : j_1 : \ldots : j_k$ și v nu este în mulțimea V_i, atunci:
    - se adaugă v la V_i
    - dacă $k < m$, se trimite mesajul $\mathrm{v} : j_1 : \ldots : j_k : \mathrm{i}$ Locotenenților care nu sunt în mulțimea $\{j_1, j_2, \ldots, j_k\}$
- Pentru fiecare *i*: atunci când Locotenentul *i* nu mai primește mesaje, se va supune ordinului $choice(V_i)$

Soluția cu mesaje semnate
Algoritmul *SM(m)*

- Cum se dectează faptul că un Locotenent nu va mai primi mesaje?
  - folosirea unui timer
  - se poate arăta că pentru o secvență $\{j_1, j_2, \dots, j_k\}$, k ≤ m, un Locotenent poate primi cel mult un mesaj de forma v : 0 : j_1 : ... : j_k
    - se poate impune ca Locotenentul j_k să trimită un astfel de mesaj sau un mesaj care să indice că nu va trimite un astfel de conținut

Corectitudine

TEOREMA 2. *Pentru orice m, Algoritmul SM(m) rezolvă problema generalilor bizantini dacă există cel mult m trădători.*

Dacă comandantul este **loial**, atunci fiecare locotenent loial se supune ordinului transmis de el.

Iniţial Vi = Φ.

(1) Comandantul semnează şi trimite valoarea sa fiecărui locotenent

(2) **For each** i:

(A) **If** Locotenent i primeşte un mesaj de forma v: [illegible] primit încă nici un ordin **then**

(i) Vi := {v};

(ii) transmite mesajul v:0:i [illegible]

(B) **If** Locotenent i primeşte un [illegible]

(i) adaugă v la *Vi;*

(ii) **if** k < m **then** trimite mesaj v:0:j1: . . . :jk:i fiecărui locotenent diferit de j1 . . . jk.

(3) **For each** i:

when Locotener

Pas 1. Comandantul trimite ordinul semnat v:0

Fiecare locotenent loial primeşte ordinul v în pasul (2)(A).

Un locotenent neloial nu poate falsifica ordinul comandantului → un locotenent loial nu poate primi un alt ordin în pasul (2)(B)

IC1. Toţi locotenenţii loiali se supun aceluiaşi ordin. Analizam cazul "comandant **tradator**".

Dacă **i** primeşte ordinul v în pasul (2)(A), atunci el îl transmite lui **j** în pasul (2)(A)(ii); **j** îl primeşte (conform proprietăţii A1).

Iniţial Vi = Φ.

(1) Comandantul semnează şi trimite valoarea să fiecărui locotenent.

(2) **For each** i:

Dacă **i** adaugă v la *Vi* în pasul (2)(B)(i), atunci el trebuie să fi primit un mesaj de forma v:0:j1: . . . :jk. Dacă **j** este unul dintre *jr*, atunci (cf A4) el trebuie să fi primit deja ordinul v.

(A) **If** Locoten
primit înc

Daca j nu este unul din *jr* & k < m: **i** trimite mesajul v:0:j1: . . . :jk:i lui **j** ➔ **j** trebuie să primească ordinul v.

(B) **If** Locotenent i primeşte un mesaj de forma v:0:j1: … :jk şi v nu este în *Vi* **then**

(i) adaugă v la *Vi;*

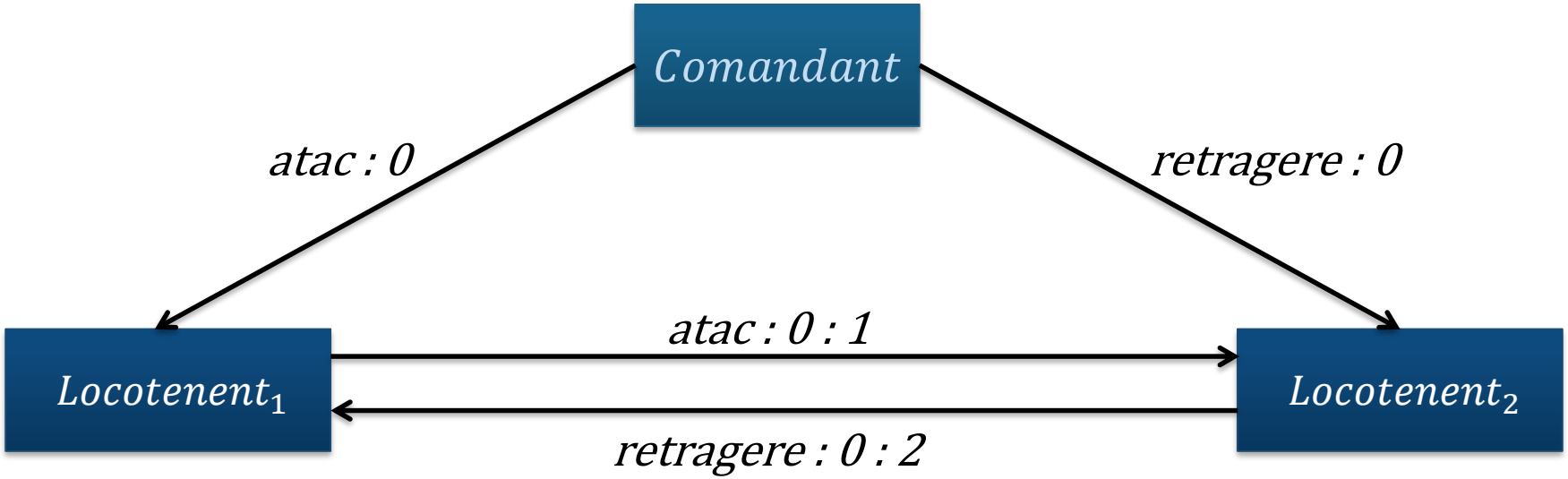
(ii) **if** k < m **then** trimite mesaj v:0:j1: . . . :jk:i fiecărui locotenent diferit de j1

(3) I

Daca j nu este unul din *jr* & k = m: comandant trădător ➔ cel mult m - 1 locotenenţi sunt trădători. Cel puţin unul dintre j1, . . . , jm este loial. El a trimis lui **j** valoarea v atunci când a primit-o prima dată.

➔ daca i are v atunci si j are v ➔ toti lt. loiali au aceleasi v-uri

Soluția cu mesaje semnate
Exemplu




POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818