Parameterised Light Atomic Broadcast and Membership Protocol (PLAMP)

Jean Fanchon
LAAS−CNRS

Introduction
- **Besoins**
- **Problèmes**

L'algorithme
- **Sur réseau synchrone fiable**
- **Crash de processus, pertes de messages**
- **Variables, paramétrage**

Implémentation et architecture

Conclusions

Coopération, coordination d'applications distribuées

Tolérance aux fautes

Membership

- **Listes de membres, join quit**
- **Notifications d'évènements, nouveaux, processus fautifs**

Atomic Broadcast

- **Diffusion de messages**
- **Livraison des mêmes messages dans le même ordre (total)**

Utilisations

- **Gestion de replica**
- **Partage d'objets par des applications distribuées**
- **Gestion distribuée de ressources**
- **Allocation unique de noms, décisions, etc....**

Ex: TSF OLC RST .....

Problèmes similaires mais contextes et utilisations différentes

−−> Code ouvert, homogène, accessible , adaptable

Problème du consensus (AB, Memb,etc..)

Difficultés intrinsèques

Asynchrone: impossibilité avec 1 seul processus fautif (FLP)

Synchrone : t fautes requièrent t+1 phases (T)

Réseau non−fiable

−−> Restreindre les hypothèses de fautes

−−> Décomposition du problème et résolution par modules (FD)

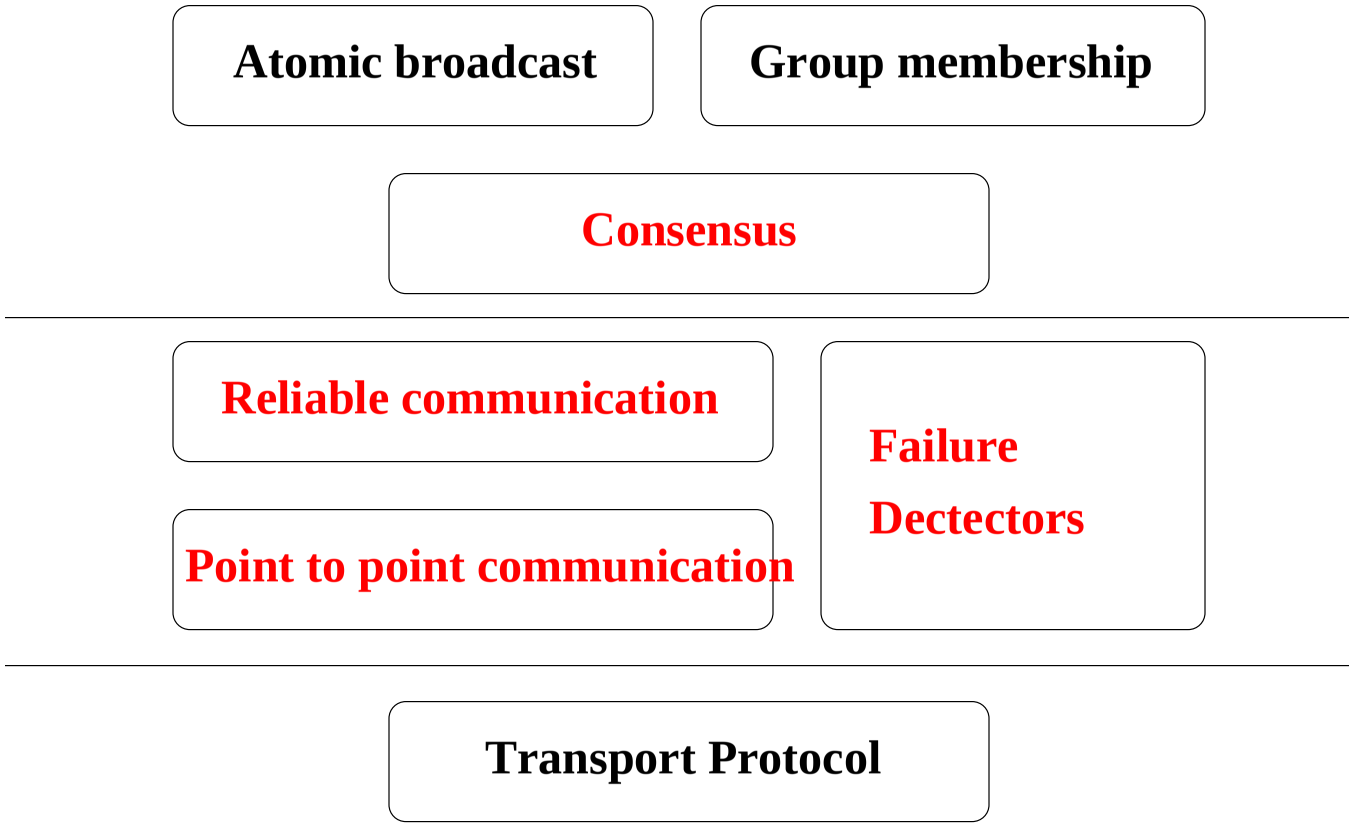
Difficultés d'implémentation

Chaque module nécessite des échanges de messages

Crash de composants vs. crash de sites

Interfaces, codes hétérogènes, maintenance, taille..

Modules / Fonctions



Membership

Agreement on group membership: if two correct processes belong to group then they have the same view of the membership

Total ordering of change notifications: any two processes deliver membership changes notifications in the same order

Virtual synchrony : membership changes notifications are delivered at the same point in the application message streams

Atomic Broadcast

B−Termination : if a correct process executes broadcast(m)
then all correct process execute delivery(m)

B−Validity : if a process executes delivery(m)
then m has been sent by a process

RB−Termination : if a process executes delivery(m)
then all correct processes execute delivery(m)

ORB−Agreement: if two processes p and q deliver m1 and m2
p and q deliver m1 and m2 in the same order

Principes de PLAMP (1)

Broadcast ordre total avec réseau et processus fiables

chaque processus −−> | **identificateur unique**

liste des membres

Algorithme par phases

Phase = i, chaque processus émet vers tous les autres

Chaque processus recoit de tous les autres

Phase = phase +1

Pour tous processus k et h | phase(k)−phase(h) | <= 1

A la fin de chaque phase, livraison de la même liste de messages

Principes de PLAMP (2)

Prise en compte de crash de processus −−>Failure detector P

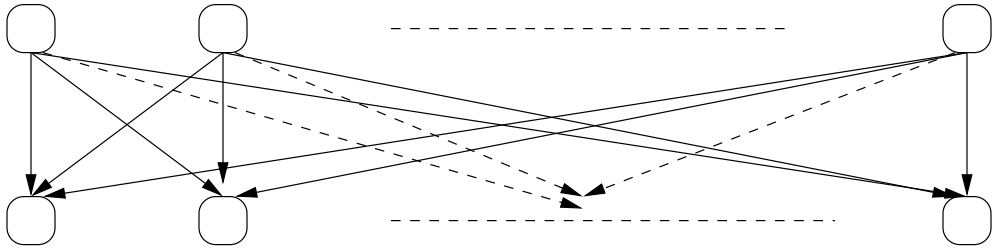
Crash de processus: arrête d'émettre en cours de phase
n'émet plus phases suivantes

Réseau fiable synchrone

Algorithme modifié

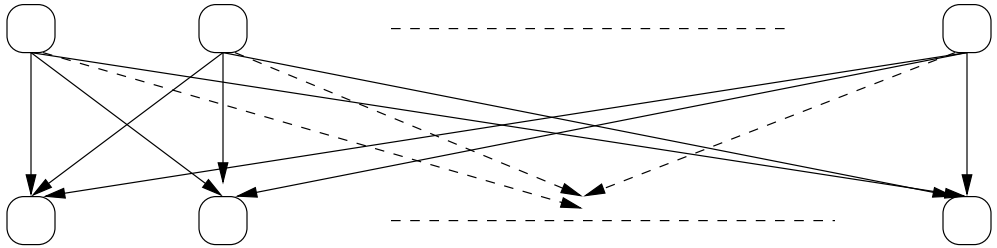
- **Fin de phase: réception de messages de tous ou bien timeout**
- **Détection de processus fautifs**
- **Message de la phase i acquittement de la phase i−1, liste de fautifs**
- **A la fin de phase i , décision quant à la phase i−1 (i−P)**

Phases et décisions



Phase i

Decide (i−1) **(i − P)**



Phase i+1

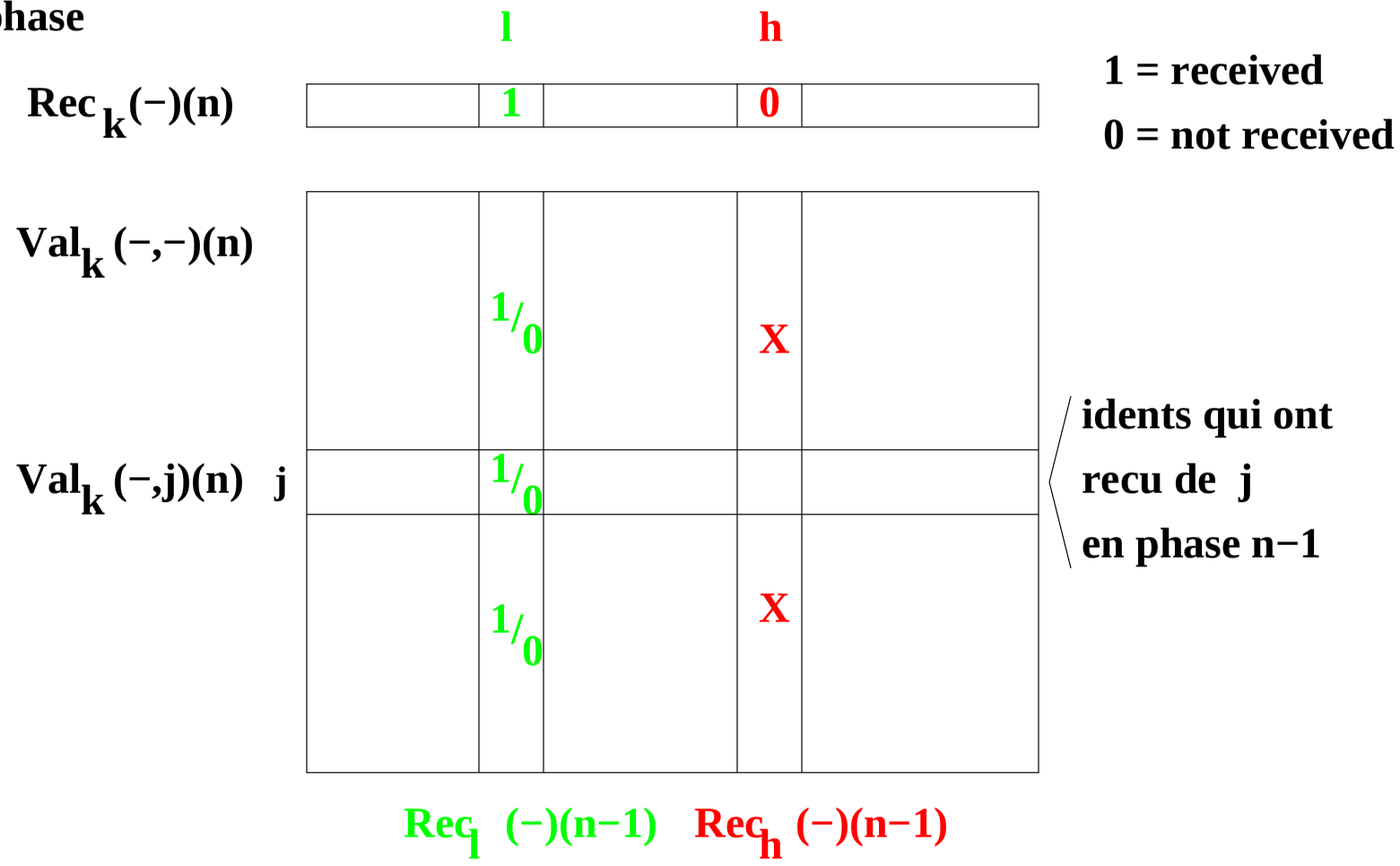
Decide (i) **(i−P+1)**

Decision: 1) processus fautifs 2) liste de messages à livrer

Variables locales

j,k,l,h: ident

n: phase



Décisions

Pas 2 phases consécutives avec faute(s)

Décision pour phase i−1

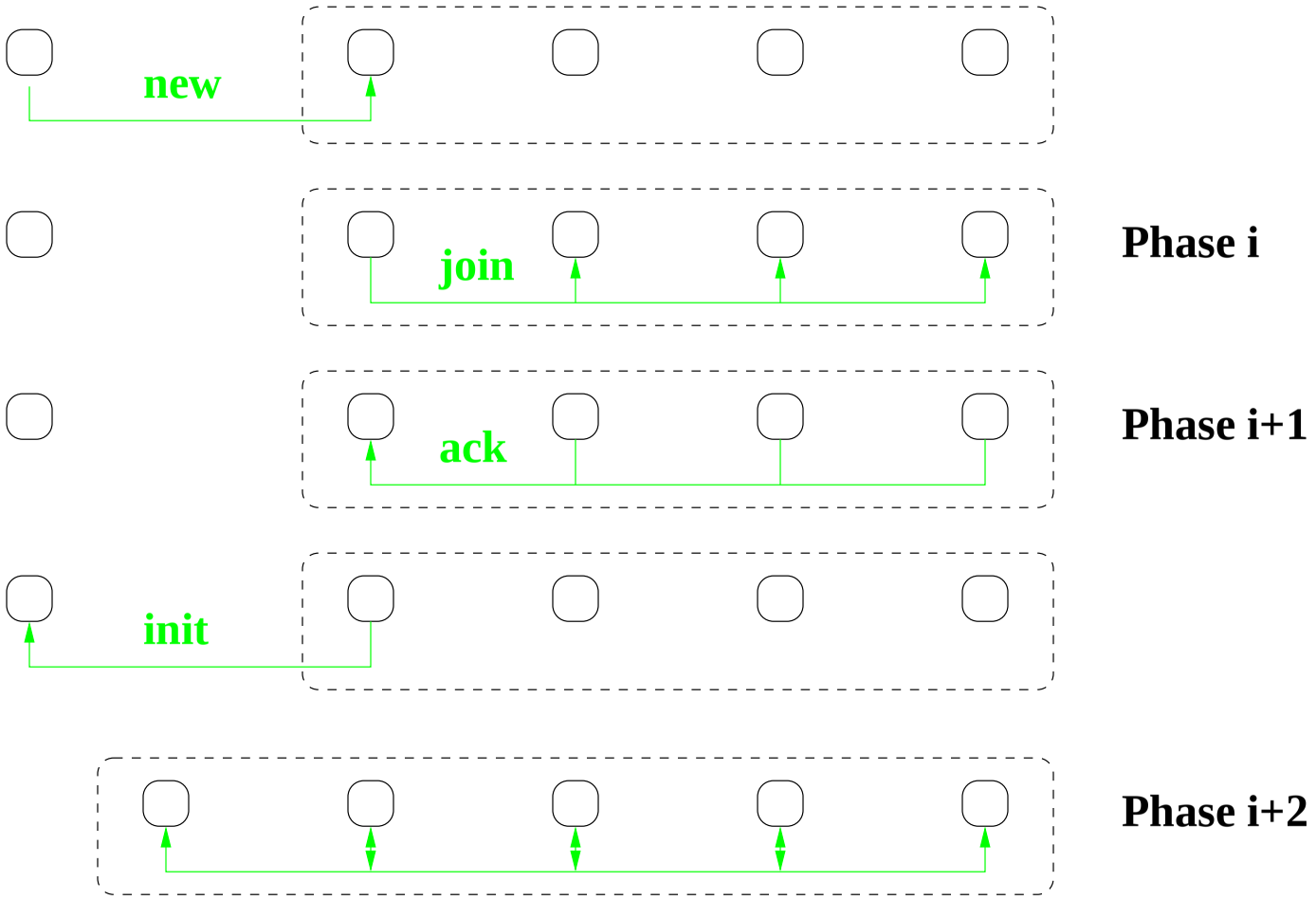
$| h, \underset{k}{Val} \ (h,j)(n) = 0 | = 0$

Décision sur phases successives dissymetriques

If f < n/2 $| \{ h , Val_k \ (h,j)(n) = 1 \}| > n/2$ =>

For all l $| \{ h , Val_l \ (h,j)(n) \in \{ 0,X\} \}| < n/2$

Join scenario



Principes de PLAMP (3)

Réseau non fiable, asynchrone --> FD $\diamond$S

Algorithme modifié

- En cours de phase, demandes de réémissions
  réémissions de message phase i ou i−1
- Fin de phase: réception de messages de tous ou bien timeout
- Détection de processus fautifs ou de messages perdus
- Message de la phase i acquittement de la phase i−1
- A la fin de phase i , décision quant à la phase i−1 (i−P)

Décisions

Pas 2 phases consécutives avec faute(s)

Décision pour phase i−1

$| h, Val_k (h,j)(n) = 0 | != 0$ ET $Rec_k(j)(n) = 0$

Pas t phases consécutives avec faute(s)

Décision pour phases i−1 à i−t

Décision sur phases successives dissymetriques

If f < n/2 $| \{ h , Val_k (h,j)(n) = 1 \}| > n/2$ =>

For all l $| \{ h , Val_l (h,j)(n) \in \{ 0,X\} \}| < n/2$

Paramétrage

Timeout et nombre des demandes de réémissions

Décision du failure detector

Nombre de phases utilisées

Protocole de transport (UDP−−> TCP)

−−> Adaptable au réseau

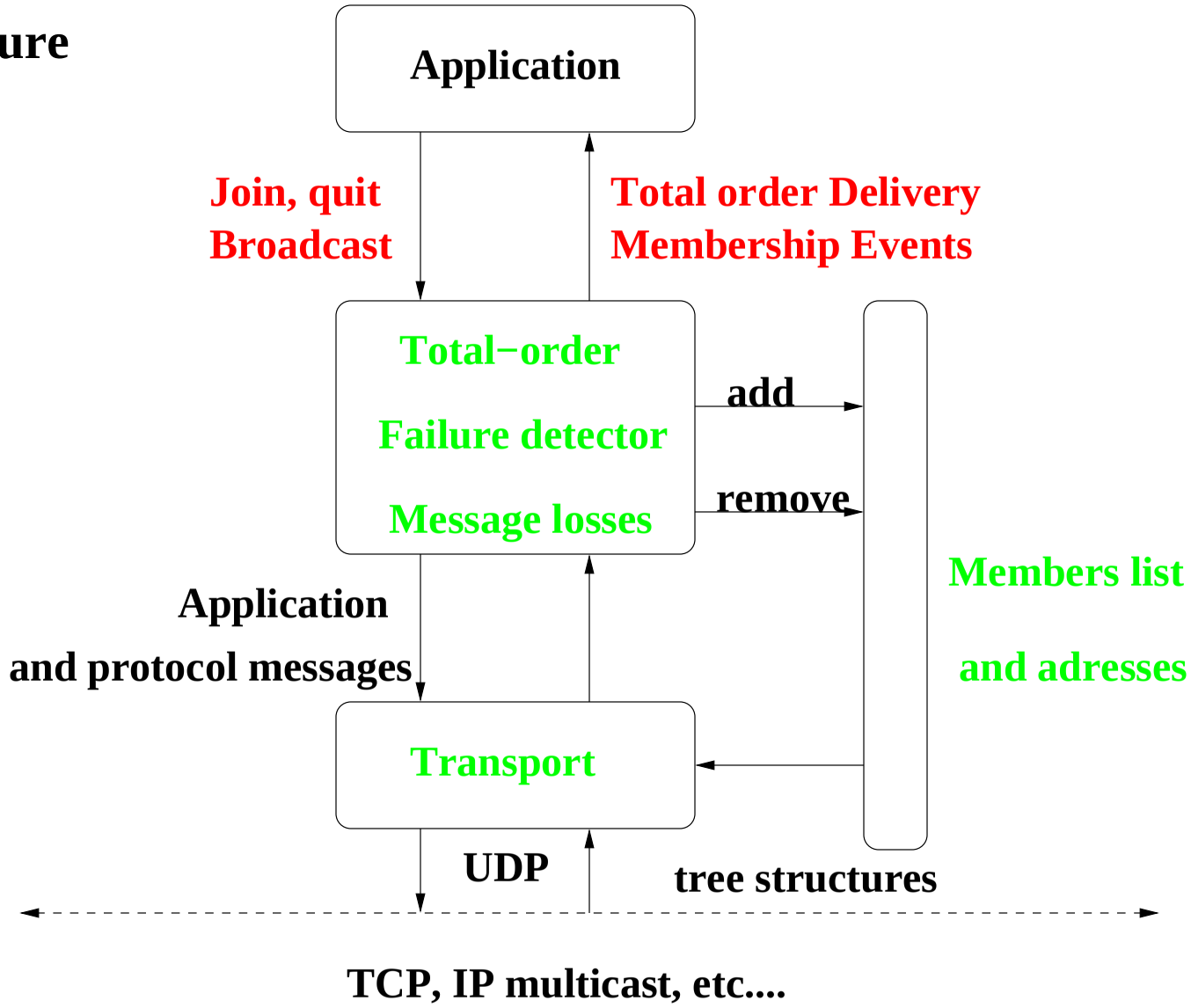
au type d'applications

au degré de développement

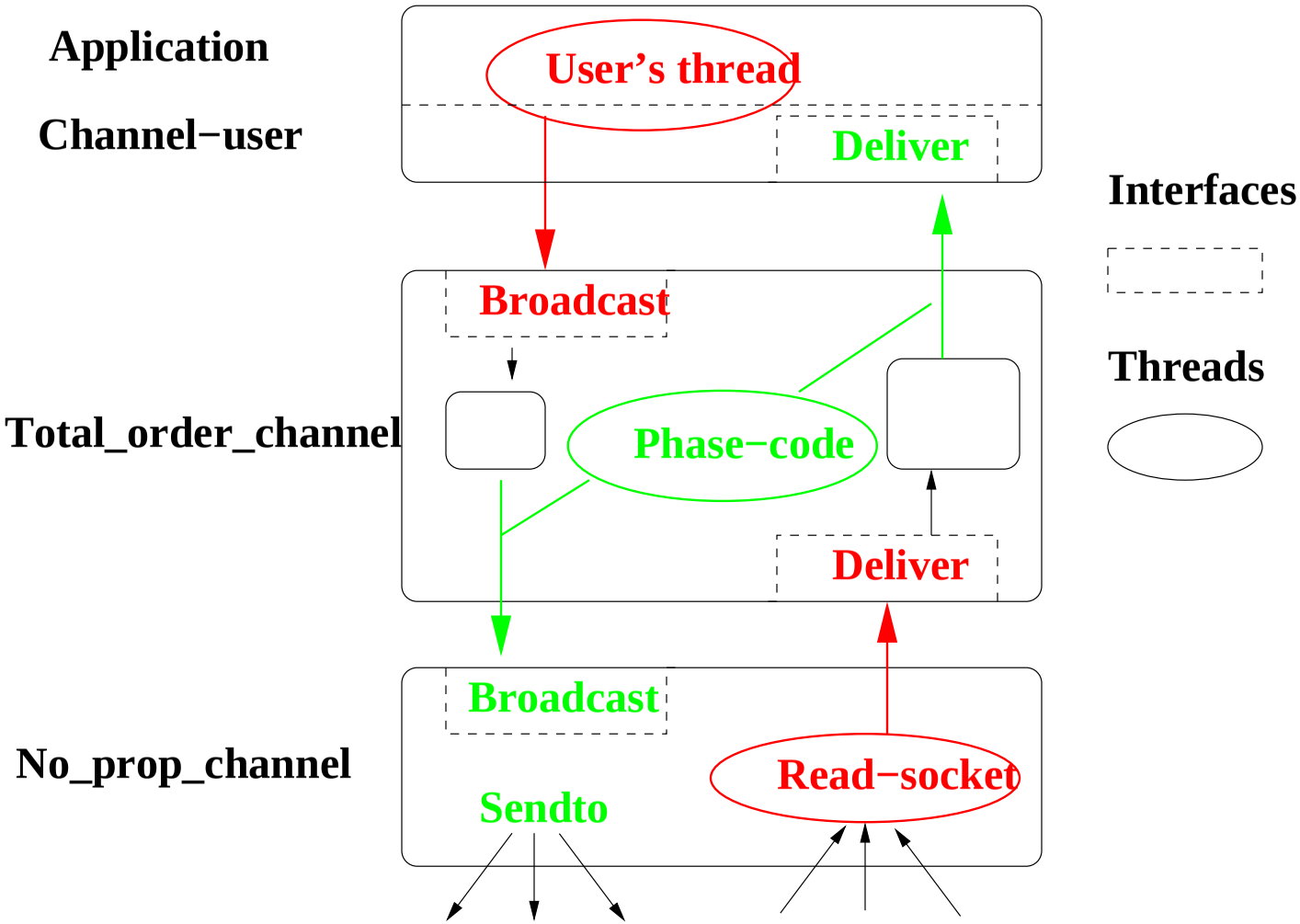
−−> Paramétrage à la main (FD)

autostabilisation (timeouts)

Structure


Application
Join, quit
Broadcast
Total order Delivery
Membership Events
Total−order
Failure detector
Message losses
add
remove
Members list
and adresses
Application
and protocol messages
Transport
UDP
tree structures
TCP, IP multicast, etc....

Classes et threads



Conclusions

Membership et atomic broadcast sur réseau non fiable

Paramétrable , ouvert, compact

Taille code: 1500 lignes C++

Taille éxecutable : 40KB

Outil de test pour algorithmes

Crash et recovery: implémentable au niveau applicatif

Compatible avec couche d' authentification

Arborescence de diffusion.

Failure Detectors

Completness Tout processus crashé est suspecté par tous les corrects

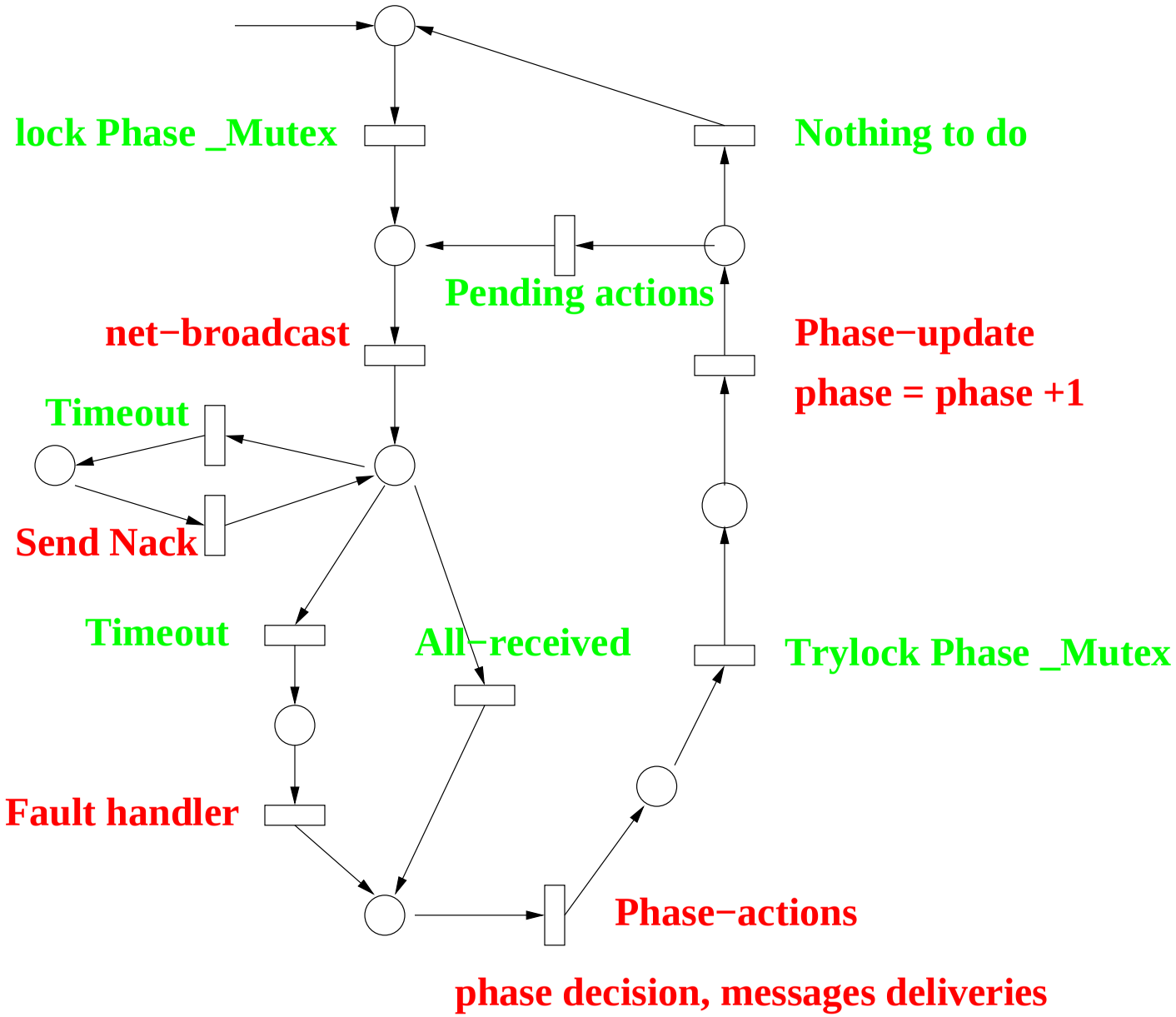
Accuracy

- $\diamond S$ Tout processus correct n'est plus suspecté apres un certain temps
- P^x Un processus ne suspecte faussement que au plus n−x−1 corrects
- P Un processus n'est jamais suspecté avant de crasher

P <---- Interactive consistency $D_k[i] \in \{v_i, \perp\}$

$\diamond S$ <---- Consensus pour t<n/2

Phase−code


lock Phase _Mutex
Nothing to do
Pending actions
net−broadcast
Phase−update
phase = phase +1
Timeout
Send Nack
Timeout
All−received
Trylock Phase _Mutex
Fault handler
Phase−actions
phase decision, messages deliveries