MPICH-V

MPI Implementation for Volatile resources

Grand Large

INRIA

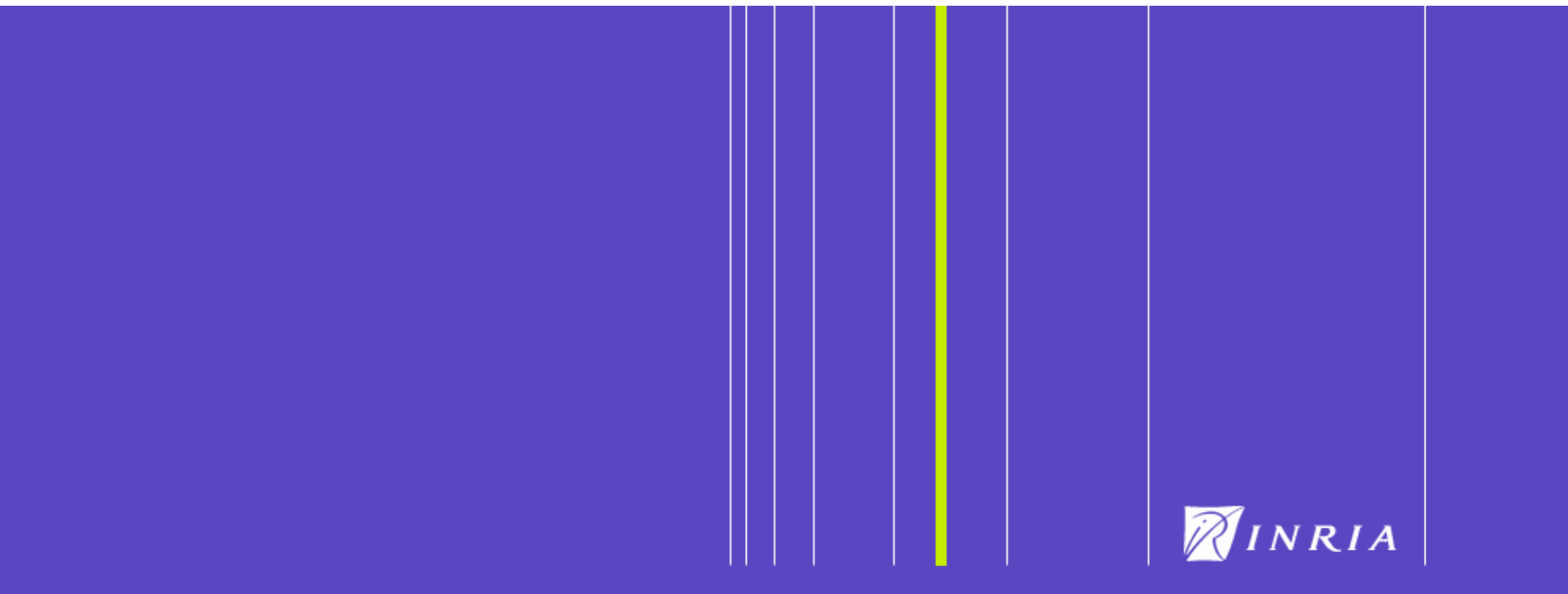
Tolérance aux défaillances dans les systèmes hautes performances

Aurélien Bouteiller - Séminaire ENS – 8 février 2006

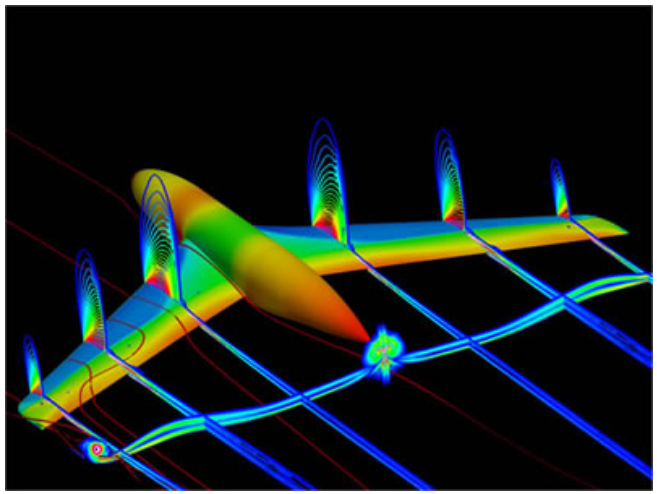
INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

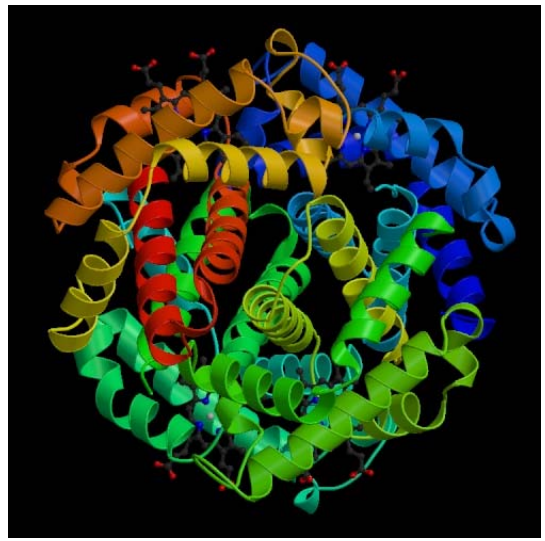
INRIA

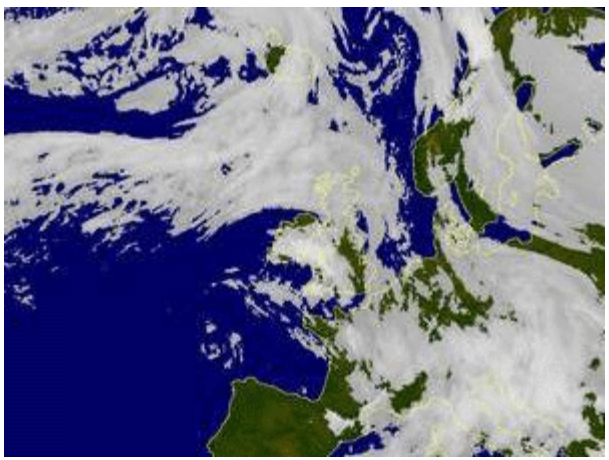
- **Qu’est ce que le calcul hautes performances ?**
- Système distribué : Modélisation
- Algorithmes de tolérance aux fautes
- Une solution pratique : MPICH-V

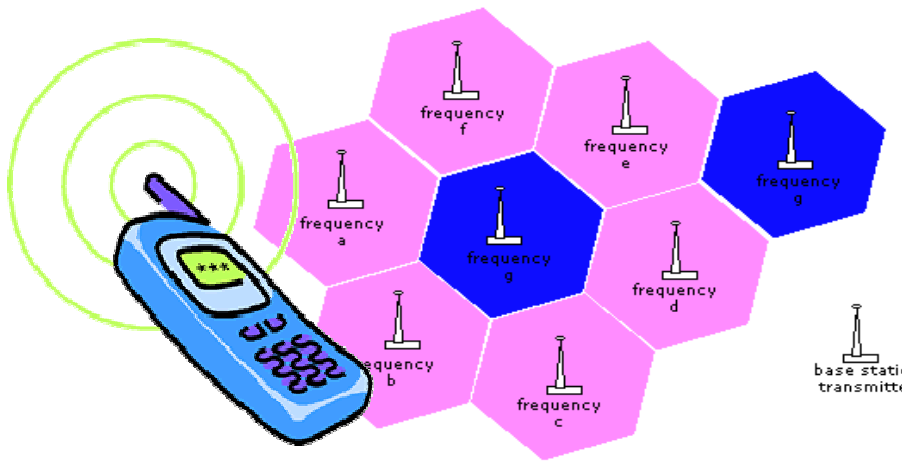


Calcul hautes performances
Pourquoi ?

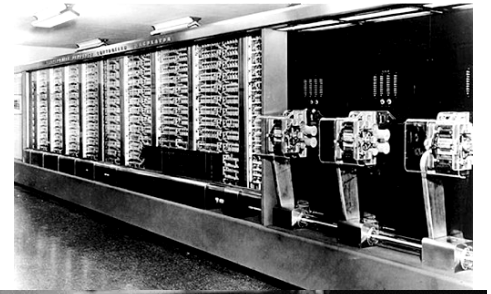








Calcul hautes performances
Un peu d'histoire



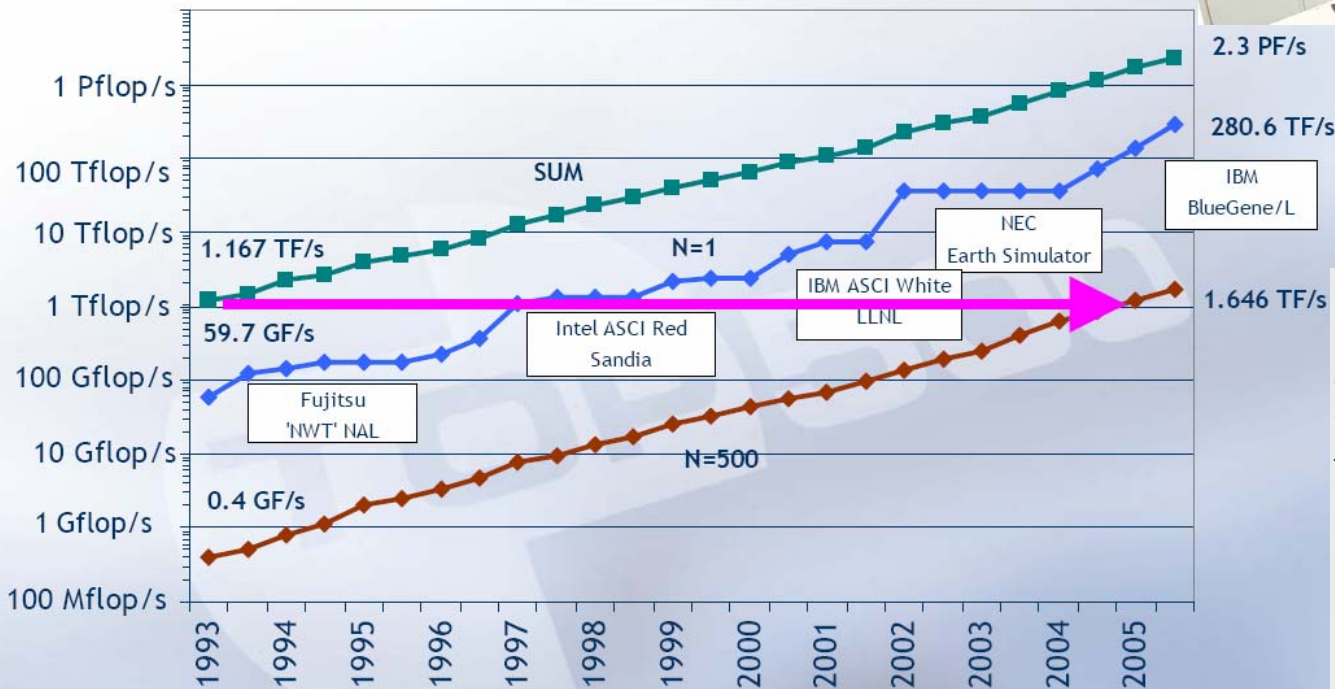
- 1944: Manhattan project : Harvard mark1
- 1955: Transistor, Univac 2 (4000 multiplications/s, Mémoire 1ko)
- 1973: Iliac IV, machine parallèle (30Mflop)
- 1974: Circuit intégré
- 1976: Cray 1, machine vectorielle (90Mflop)
- 1986: IBM 3090, machine à passage de message (8 processeurs - 1Gflop)

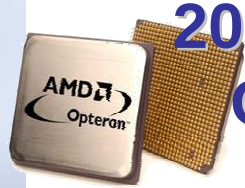
Calcul hautes performances
Époque moderne

1998
ASCI RED
1TFlop
10k Ppro@333

TOP500 SUPERCOMPUTER SITES

Performance Development



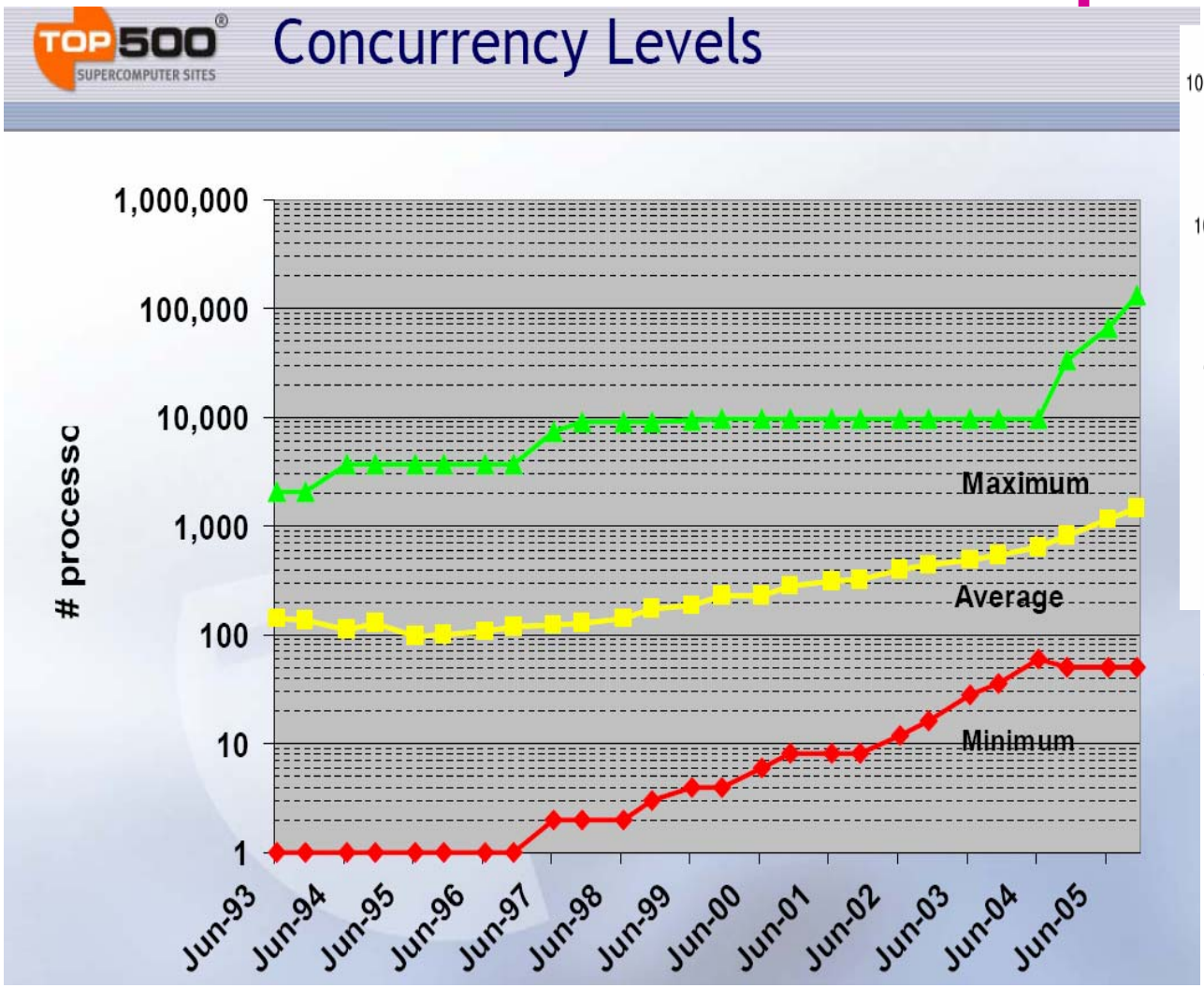


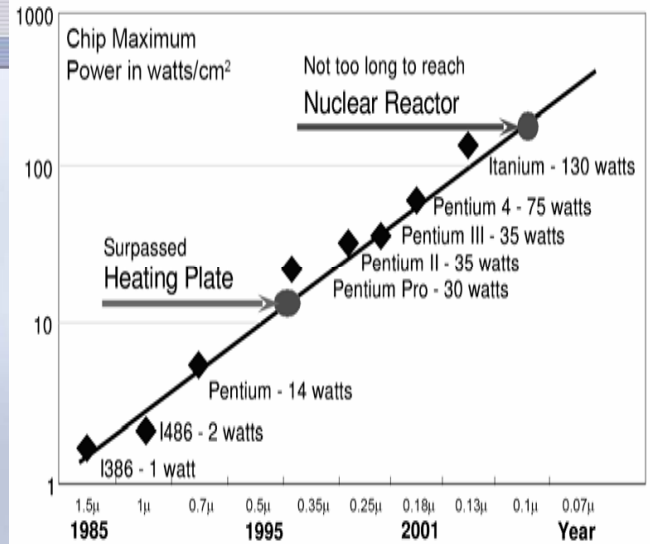
2006 AMD
Opteron
3Gflop

2006
Xbox 360
1TFlop
1 GPU

INRIA

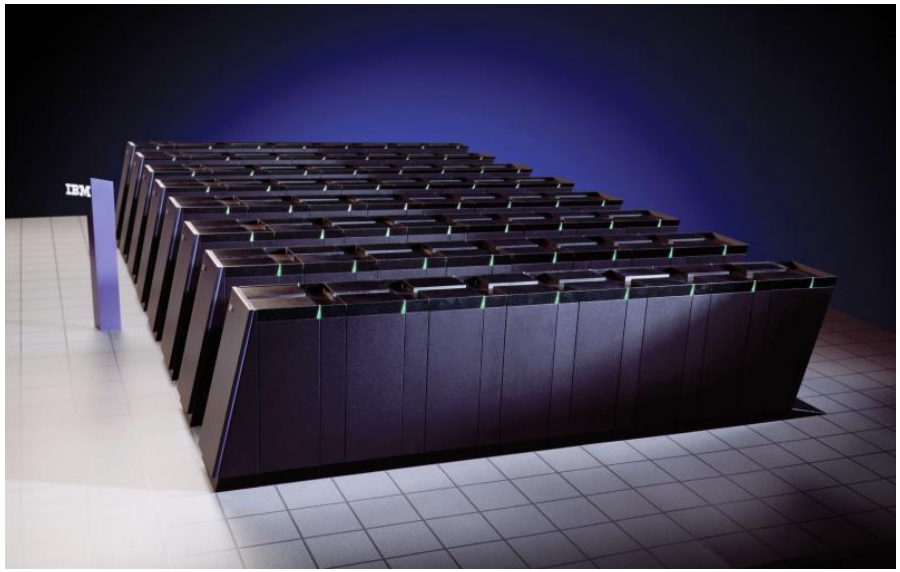
Calcul hautes performances
Évolution du nombre de processeurs



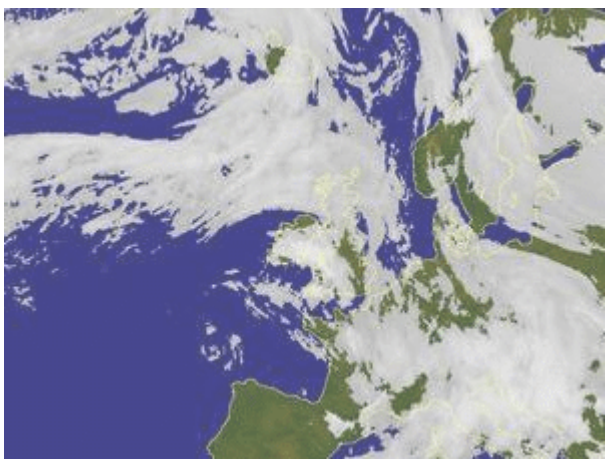


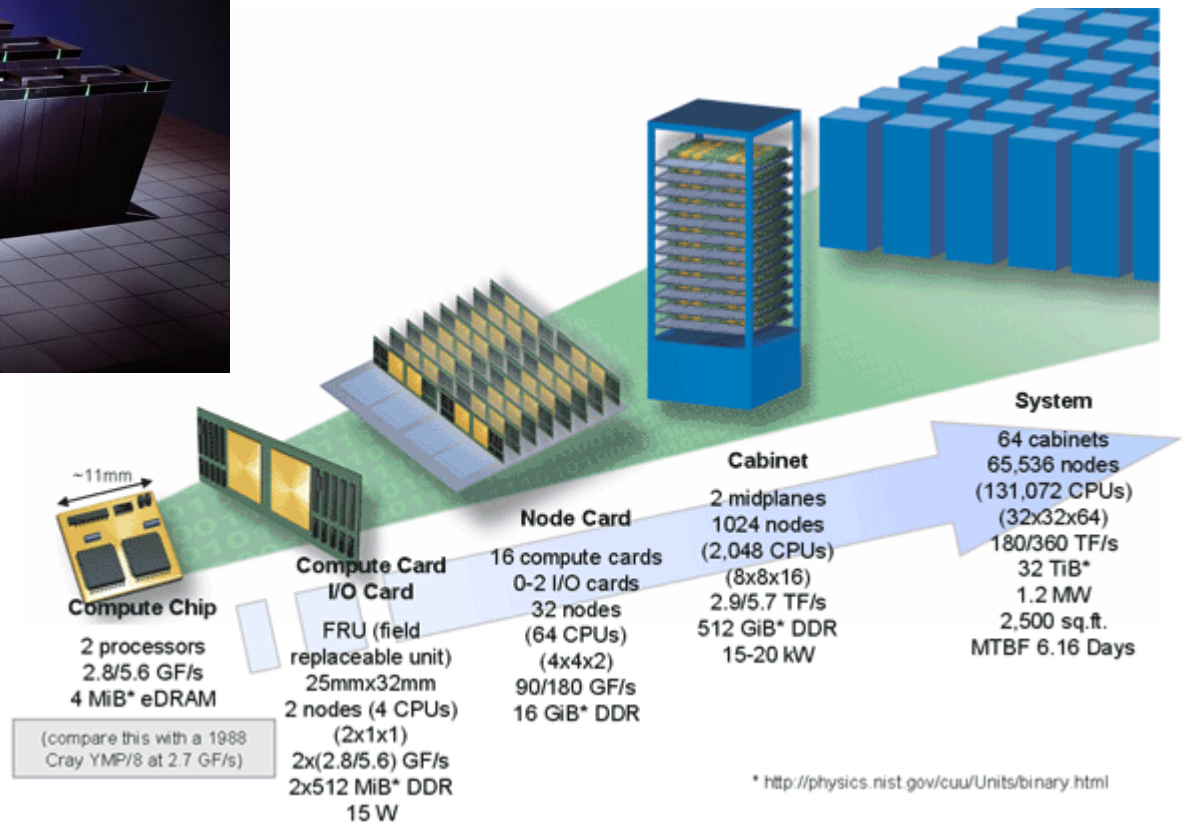
A64@2Ghz=P4@3.5Ghz
Cause: vitesse des e^-

Calcul hautes performances
Le cas de BlueGene/L

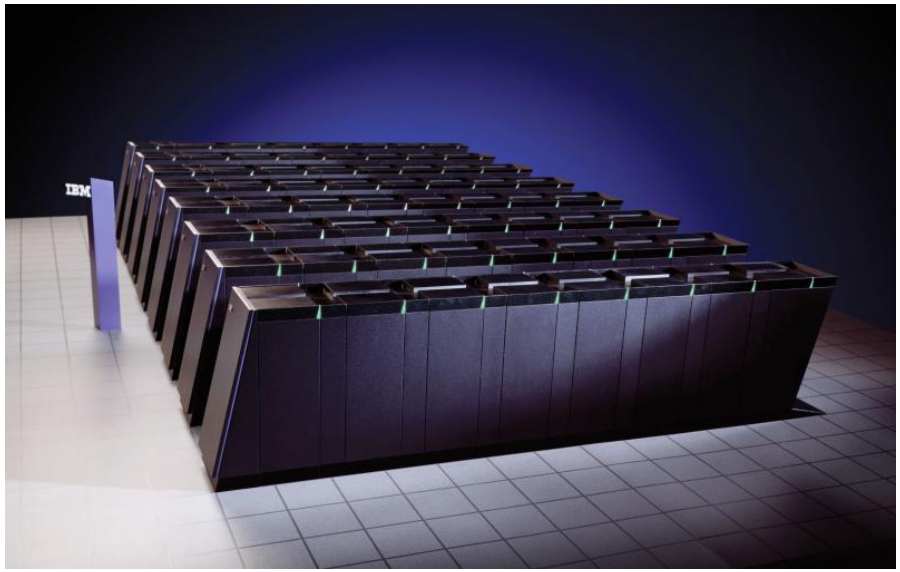


132000 CPU
280Tflops

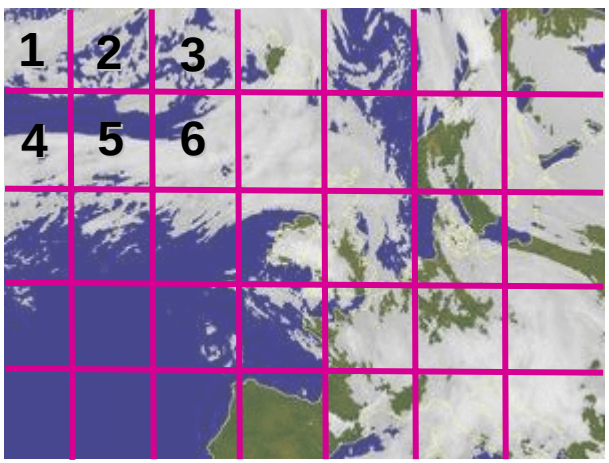


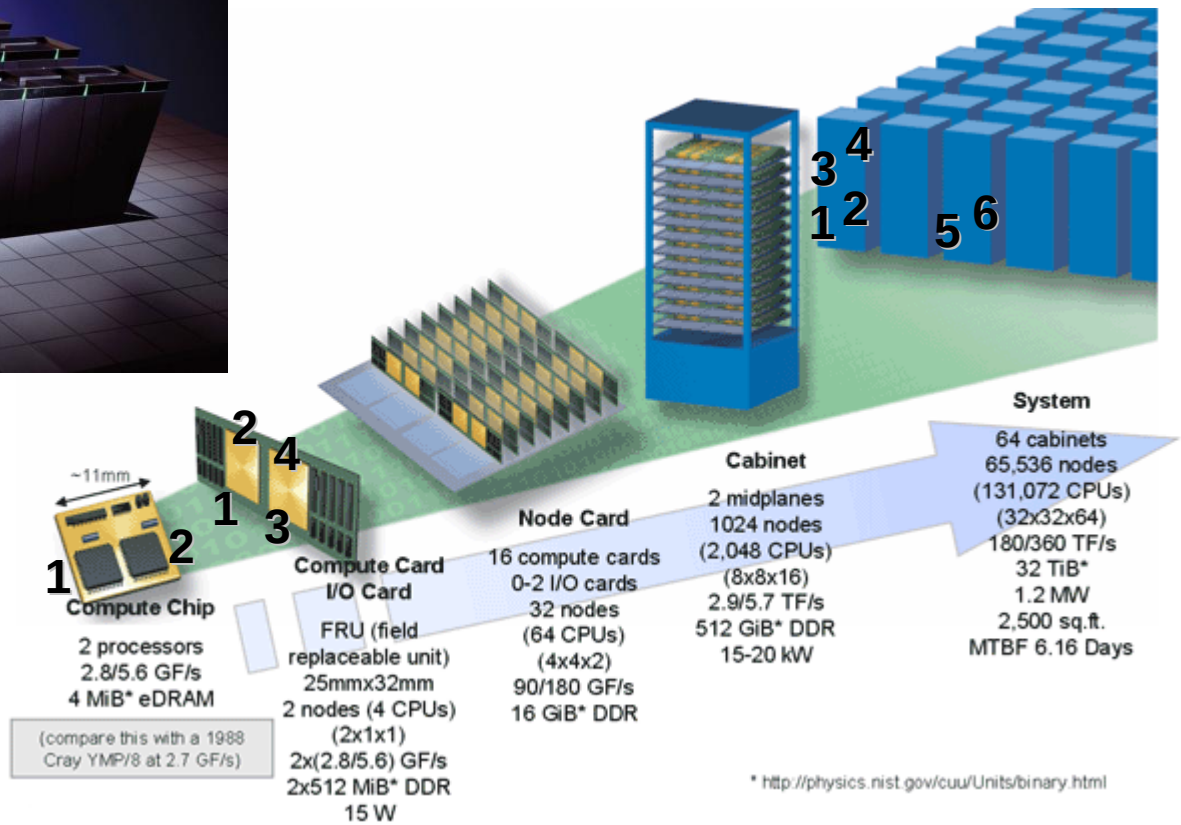


Calcul hautes performances
Le cas de BlueGene/L



132000 CPU
280Tflops





Calcul hautes performances
Temps moyen entre deux fautes

Accurate Comparisons?

Sandia National Laboratories

Systems	CPUs	Reliability & Availability
ASCI Q	8,192	MTBI: 6.5 hrs. 114 unplanned outages/month. ◆ HW outage sources: storage, CPU, memory.
ASCI White	8,192	MTBF: 5 hrs. (2001) and 40 hrs. (2003). ◆ HW outage sources: storage, CPU, 3rd-party HW.
NERSC Seaborg	6,656	MTBI: 14 days. MTTR: 3.3 hrs. ◆ SW is the main outage source. Availability: 98.74%.
PSC Lemieux	3,016	MTBI: 9.7 hrs. Availability: 98.33%.
Google	~15,000	20 reboots/day; 2-3% machines replaced/year. ◆ HW outage sources: storage, memory. Availability: ~100%.

MTBI: mean time between interrupts; MTBF: mean time between failures; MTTR: mean time to restore

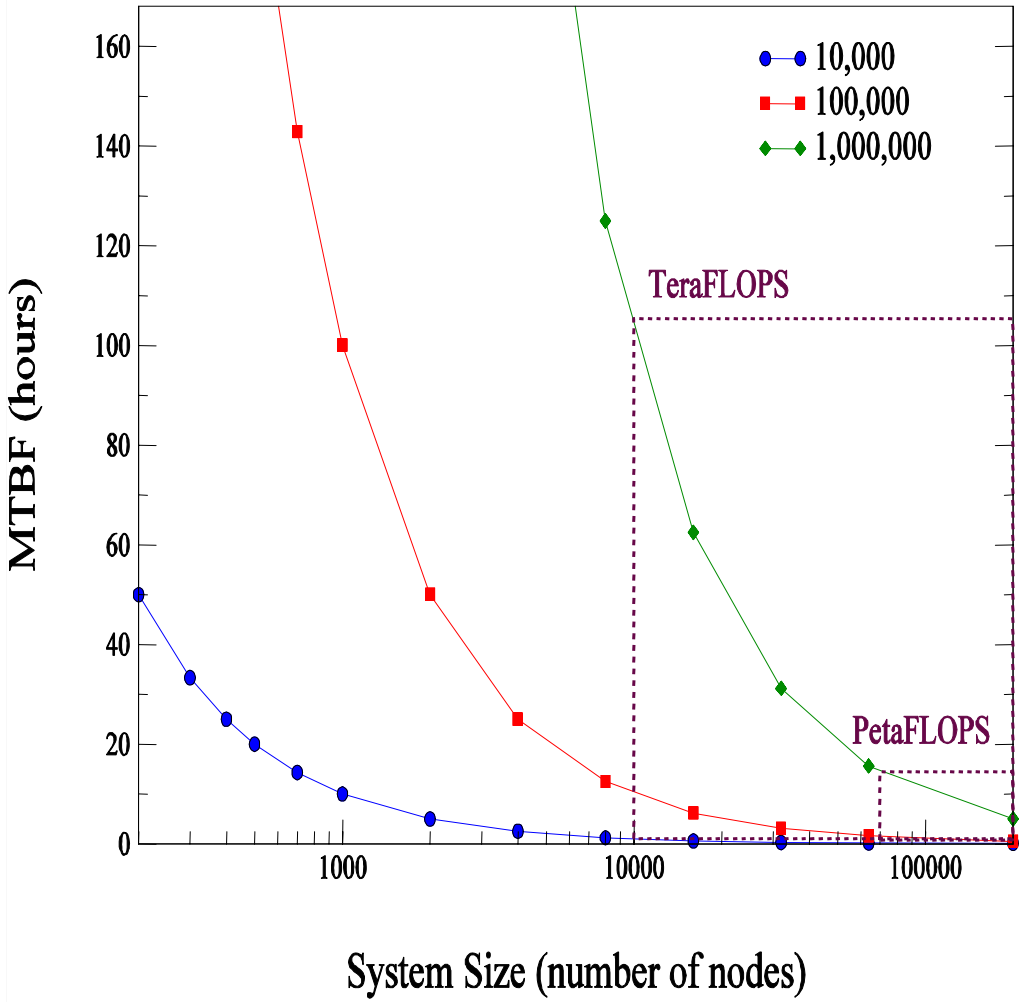
Source: Daniel A. Reed, UNC (via Chung-Hsing Hsu, LANL)

INRIA

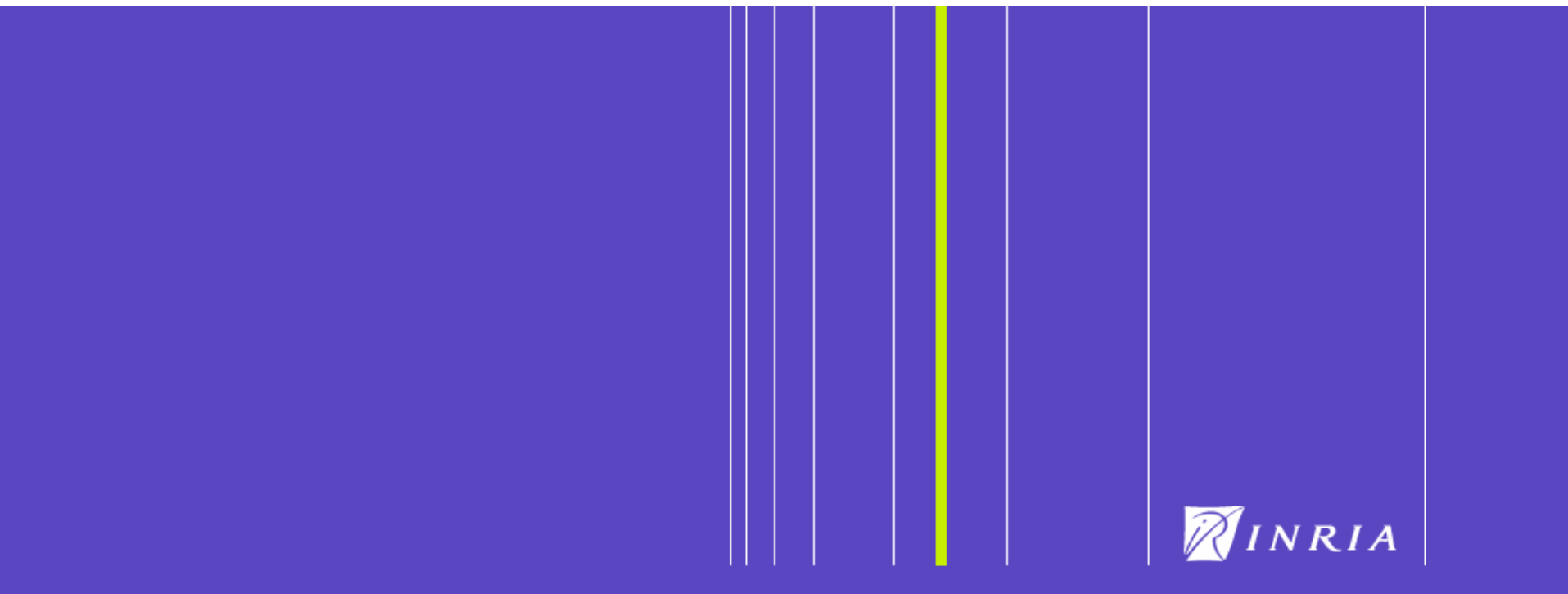
Calcul hautes performances
Évolution du MTBF dans le futur

1 PetaFlop = 200k 5Gflop CPU

Current reliable hardware (ASCI white) would lead to more than 1 fault every hour on a Petaflop scale machine



- Qu’est ce que le calcul hautes performances ?
- **Système distribué : Modélisation**
- Algorithmes de tolérance aux fautes
- Une solution pratique : MPICH-V



Système distribué : modélisation
Relation de Lamport

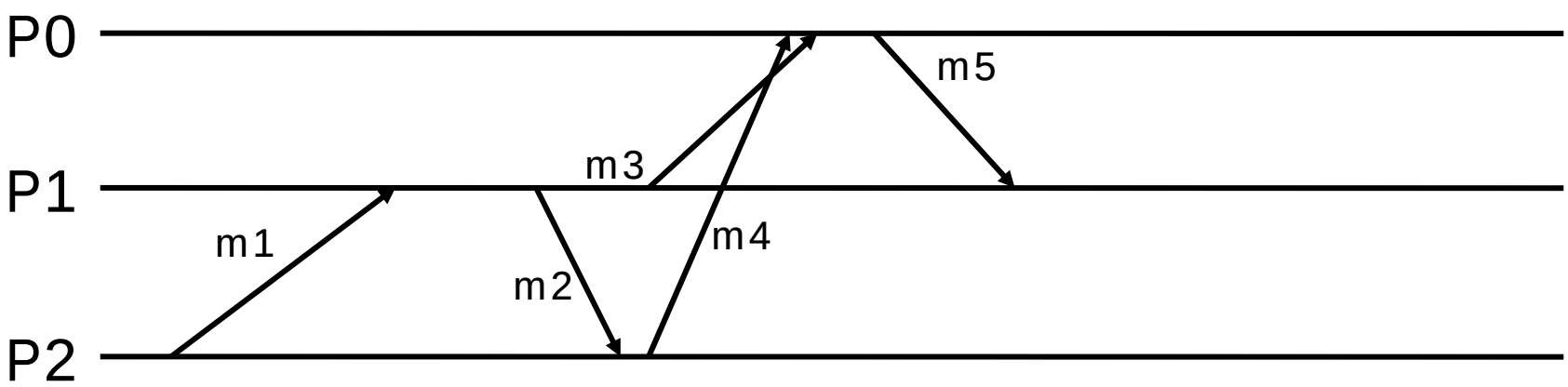
Définition : **relation de précédence de Lamport**

sur un ensemble **E** est une relation << qui vérifie :

$\forall\, T \in E, \neg\, (T << T)$ Non reflexive

$\forall\, T_1,\ T_2 \in E, \neg\, ((T_1 << T_2) \wedge (T_2 << T_1))$ Non symetrique

$\forall\, T_1,\ T_2,\ T_3 \in E, ((T_1 << T_2) \wedge (T_2 << T_3)) \Rightarrow (T_1 << T_3)$ Transitive



Système distribué : modélisation
Relation de causalité

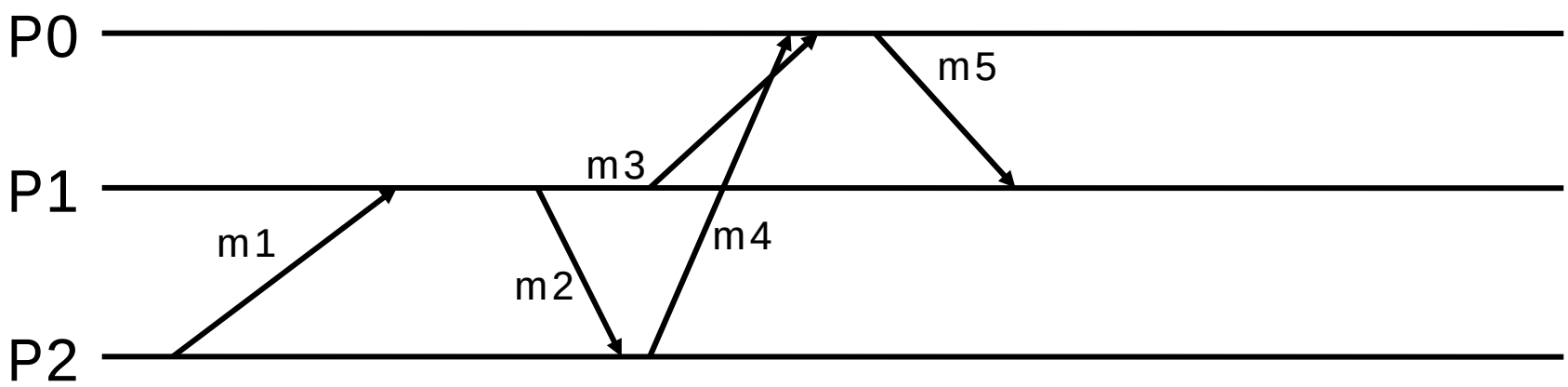
Définition : **relation de causalité** associée à Pi dans l'état j (O^i_j) est la plus petite relation de précédence qui contienne la relation → telle que

$$\forall\ i,j,k,\ (j < k) \Leftrightarrow (O^i_j \rightarrow O^i_k)$$

Ordre total par processus

$$\forall\ i,j,k,l,\ i \neq j\ (O^i_k \rightarrow O^j_l) \Leftrightarrow\ (O^i_k,\ O^j_l)$$

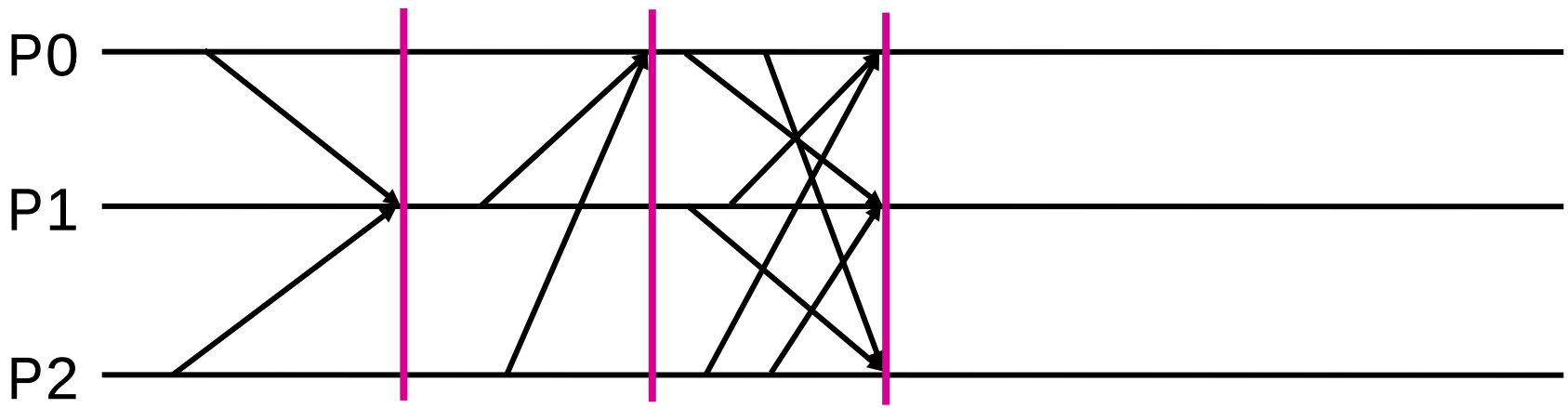
est le couple formé par l'émission par Pi et la réception par Pj d'un message (emission précède la réception).



Système distribué : modélisation
Simplification avec le modèle synchrone

On suppose une horloge globale

- Fonctionnement par phases
- Durant la phase les processus calculent
- Au début de la phase suivante, tous les messages reçus sont disponibles



Système distribué : modélisation
Introduire les fautes dans le système

Panne crash (arrêt total)

- Un processus n'émet plus aucun message à partir de la faute
- C'est le type de faute le moins grave

Panne intermittente

- Un processus s'arrête de façon intermittente. Il omet l'émission de certains messages

Panne byzantine

- Un processus présente n'importe quel comportement arbitraire
- C'est le type de faute le plus grave (corruption mémoire, virus, etc).

Panne de réseau

- Dans le modèle synchrone, on considère simplement qu'un message est supprimé

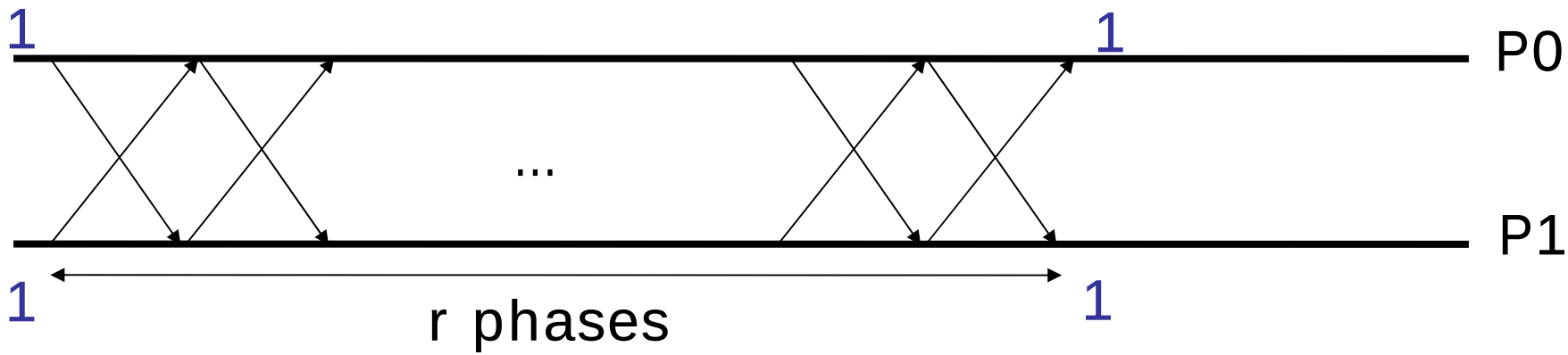
Système distribué : modélisation
Problème du concensus (synchrone)

Les généraux veulent coordonner leur attaque

- Chaque jour, ils envoient un messager qui informe les autres généraux de leur intention (attaquer ou non)
- Lorsque tous les généraux souhaitent attaquer, ils attaquent
- Problème : Les messagers peuvent se faire tuer par l'enemi!

3 conditions :

- Terminaison : Les généraux décident en un temps fini
- Agrément : les généraux décident tous la même valeur
- Validité : si initialement, tous les généraux veulent attaquer, ils attaquent. Si initialement aucun général ne veut attaquer, ils n'attaquent pas



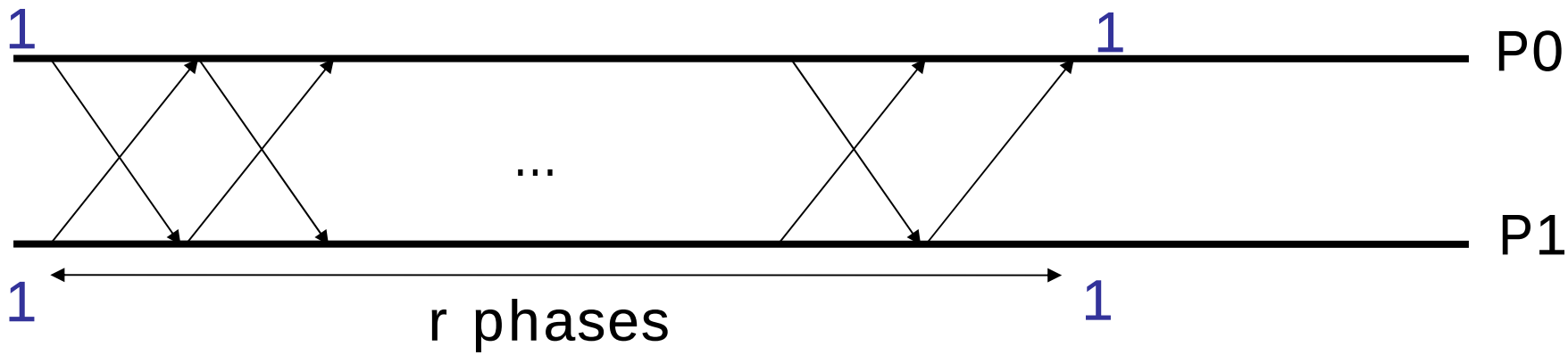
Système distribué : modélisation
Problème du concensus (synchrone)

Les généraux veulent coordonner leur attaque

- Chaque jour, ils envoient un messager qui informe les autres généraux de leur intention (attaquer ou non)
- Lorsque tous les généraux souhaitent attaquer, ils attaquent
- Problème : Les messagers peuvent se faire tuer par l'enemi!

3 conditions :

- Terminaison : Les généraux décident en un temps fini
- Agrément : les généraux décident tous la même valeur
- Validité : si initialement, tous les généraux veulent attaquer, ils attaquent. Si initialement aucun général ne veut attaquer, ils n'attaquent pas



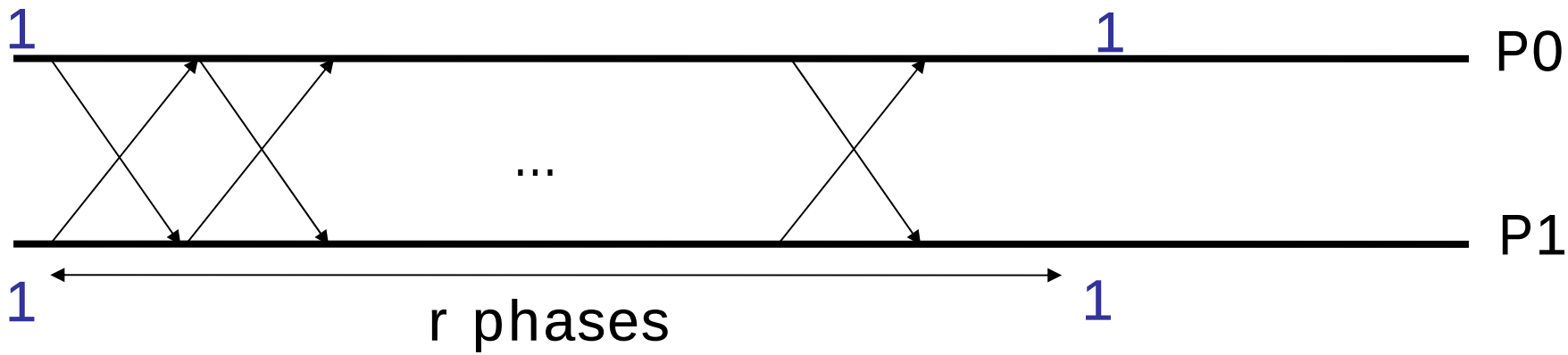
Système distribué : modélisation
Problème du concensus (synchrone)

Les généraux veulent coordonner leur attaque

- Chaque jour, ils envoient un messager qui informe les autres généraux de leur intention (attaquer ou non)
- Lorsque tous les généraux souhaitent attaquer, ils attaquent
- Problème : Les messagers peuvent se faire tuer par l'enemi!

3 conditions :

- Terminaison : Les généraux décident en un temps fini
- Agrément : les généraux décident tous la même valeur
- Validité : si initialement, tous les généraux veulent attaquer, ils attaquent. Si initialement aucun général ne veut attaquer, ils n'attaquent pas



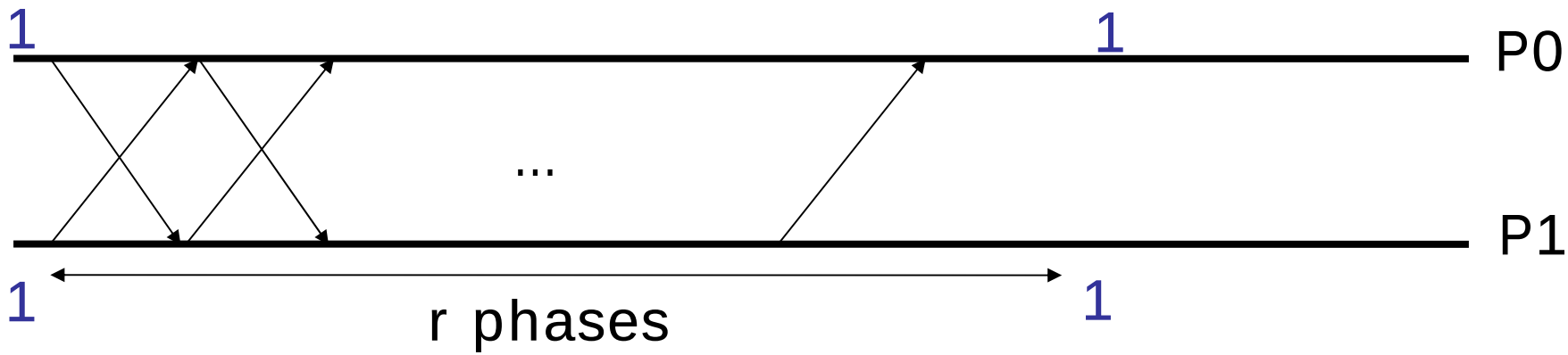
Système distribué : modélisation
Problème du concensus (synchrone)

Les généraux veulent coordonner leur attaque

- Chaque jour, ils envoient un messager qui informe les autres généraux de leur intention (attaquer ou non)
- Lorsque tous les généraux souhaitent attaquer, ils attaquent
- Problème : Les messagers peuvent se faire tuer par l'enemi!

3 conditions :

- Terminaison : Les généraux décident en un temps fini
- Agrément : les généraux décident tous la même valeur
- Validité : si initialement, tous les généraux veulent attaquer, ils attaquent. Si initialement aucun général ne veut attaquer, ils n'attaquent pas



Système distribué : modélisation
Problème du concensus (synchrone)

Les généraux veulent coordonner leur attaque

- Chaque jour, ils envoient un messager qui informe les autres généraux de leur intention (attaquer ou non)
- Lorsque tous les généraux souhaitent attaquer, ils attaquent
- Problème : Les messagers peuvent se faire tuer par l'enemi!

3 conditions :

- Terminaison : Les généraux décident en un temps fini
- Agrément : les généraux décident tous la même valeur
- Validité : si initialement, tous les généraux veulent attaquer, ils attaquent. Si initialement aucun général ne veut attaquer, ils n'attaquent pas

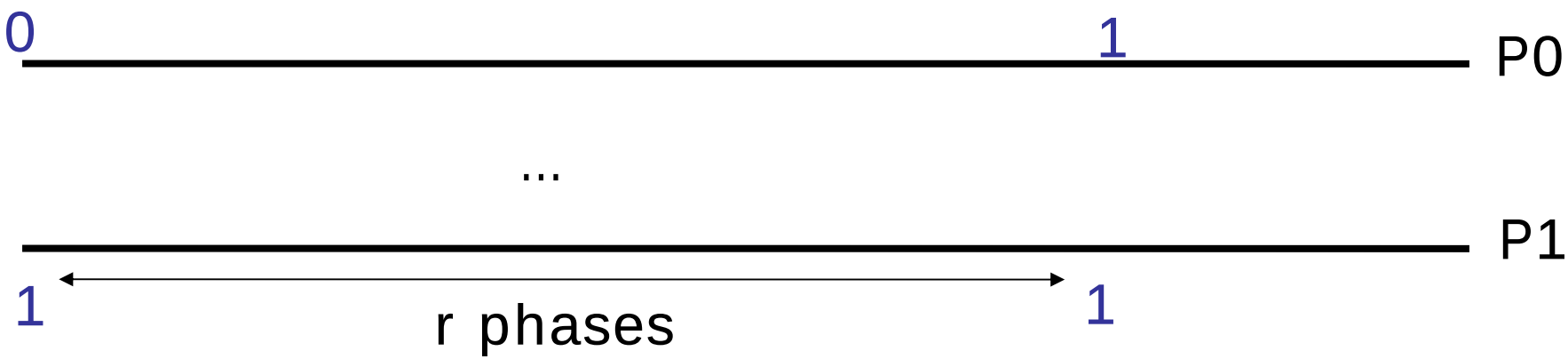
Système distribué : modélisation
Problème du concensus (synchrone)

Les généraux veulent coordonner leur attaque

- Chaque jour, ils envoient un messager qui informe les autres généraux de leur intention (attaquer ou non)
- Lorsque tous les généraux souhaitent attaquer, ils attaquent
- Problème : Les messagers peuvent se faire tuer par l'enemi!

3 conditions :

- Terminaison : Les généraux décident en un temps fini
- Agrément : les généraux décident tous la même valeur
- Validité : si initialement, tous les généraux veulent attaquer, ils attaquent. Si initialement aucun général ne veut attaquer, ils n'attaquent pas



Système distribué : modélisation
Problème du concensus (synchrone)

Les généraux veulent coordonner leur attaque

- Chaque jour, ils envoient un messager qui informe les autres généraux de leur intention (attaquer ou non)
- Lorsque tous les généraux souhaitent attaquer, ils attaquent
- Problème : Les messagers peuvent se faire tuer par l'enemi!

3 conditions :

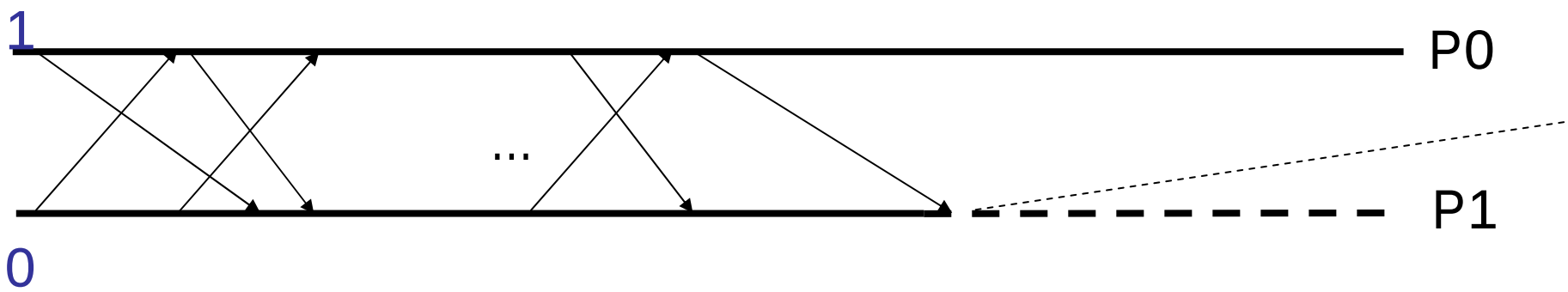
- Terminaison : Les généraux décident en un temps fini
- Agrément : les généraux décident tous la même valeur
- Validité : si initialement, tous les généraux veulent attaquer, ils attaquent. Si initialement aucun général ne veut attaquer, ils n'attaquent pas

Système distribué : modélisation
Concensus dans le cas général

Consensus asynchrone

- Terminaison : Les processus décident en un temps fini
- Agrément : tous les processus non défaillants décident d'une même valeur
- Validité : Si les processus partagent tous une même valeur initiale, ils décident de cette valeur
- Type de fautes : des processus peuvent s'arrêter totalement

Fischer, Lynch, Paterson, Impossibility of distributed concensus with one faulty process (journal of ACM 32(2), April 1985)



Système distribué : modélisation
Tolérer des fautes, Mission impossible ?

Ignorer le problème : l'expérience du projet Isis

- Group Membership service sur internet
- Lorsqu'un processus détecte qu'il a été suspecté, il se suicide
- Timeout de détection de faute augmenté jusqu'à 20 minutes et plus
- Fréquents scénarios de suicides collectifs…

Un calculateur haute performance n'est pas internet !

- La solution, Jim, est d'assouplir le modèle, sans faire d'hypothèses délirantes (horloge globale)

Système distribué : modélisation Pseudosynchronisme

Modèle synchrone

- Les processus réalisent une étape de calcul de façon synchronisée (Horloge globale).
- A la fin de la phase, tous les messages envoyés ont été reçus
- *On peut résoudre le problème*, mais ne correspond pas au monde réel.

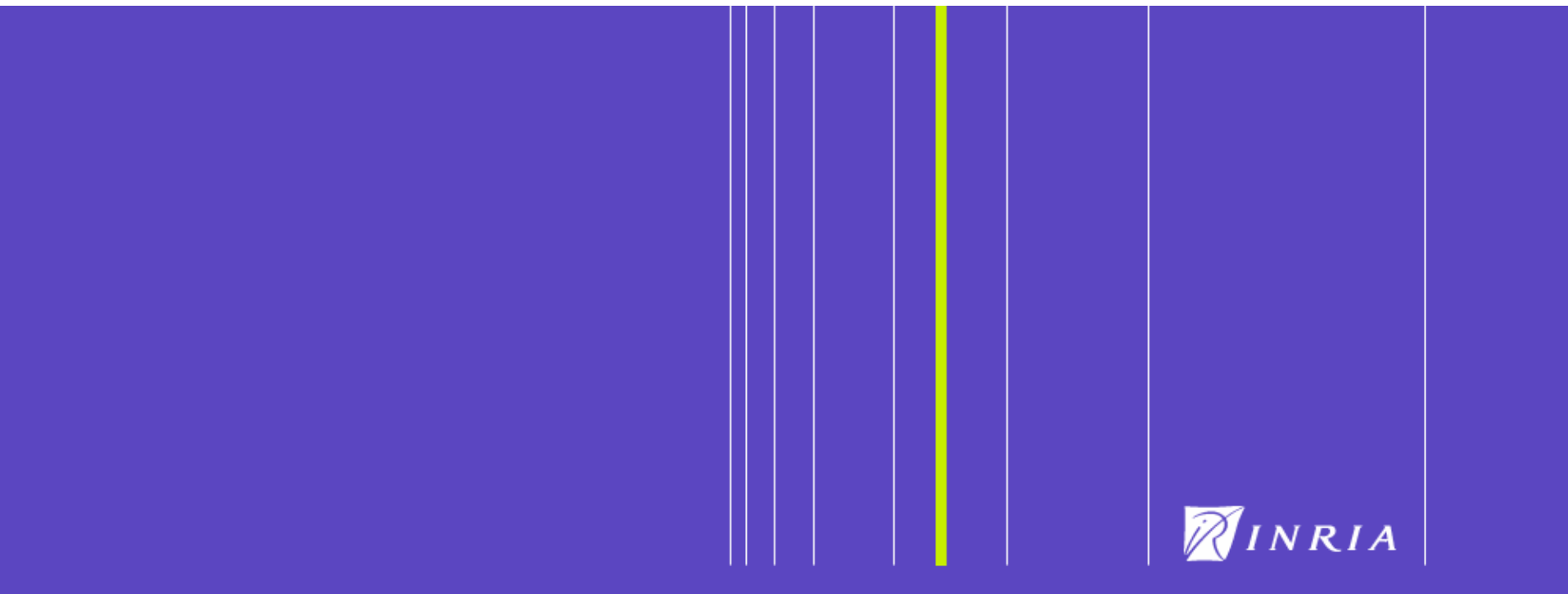
Modèle asynchrone

- pas d'horloge globale
- un message peut transiter un temps arbitrairement long dans un canal
- Modèle très expressif (Internet), mais on ne peut rien faire!

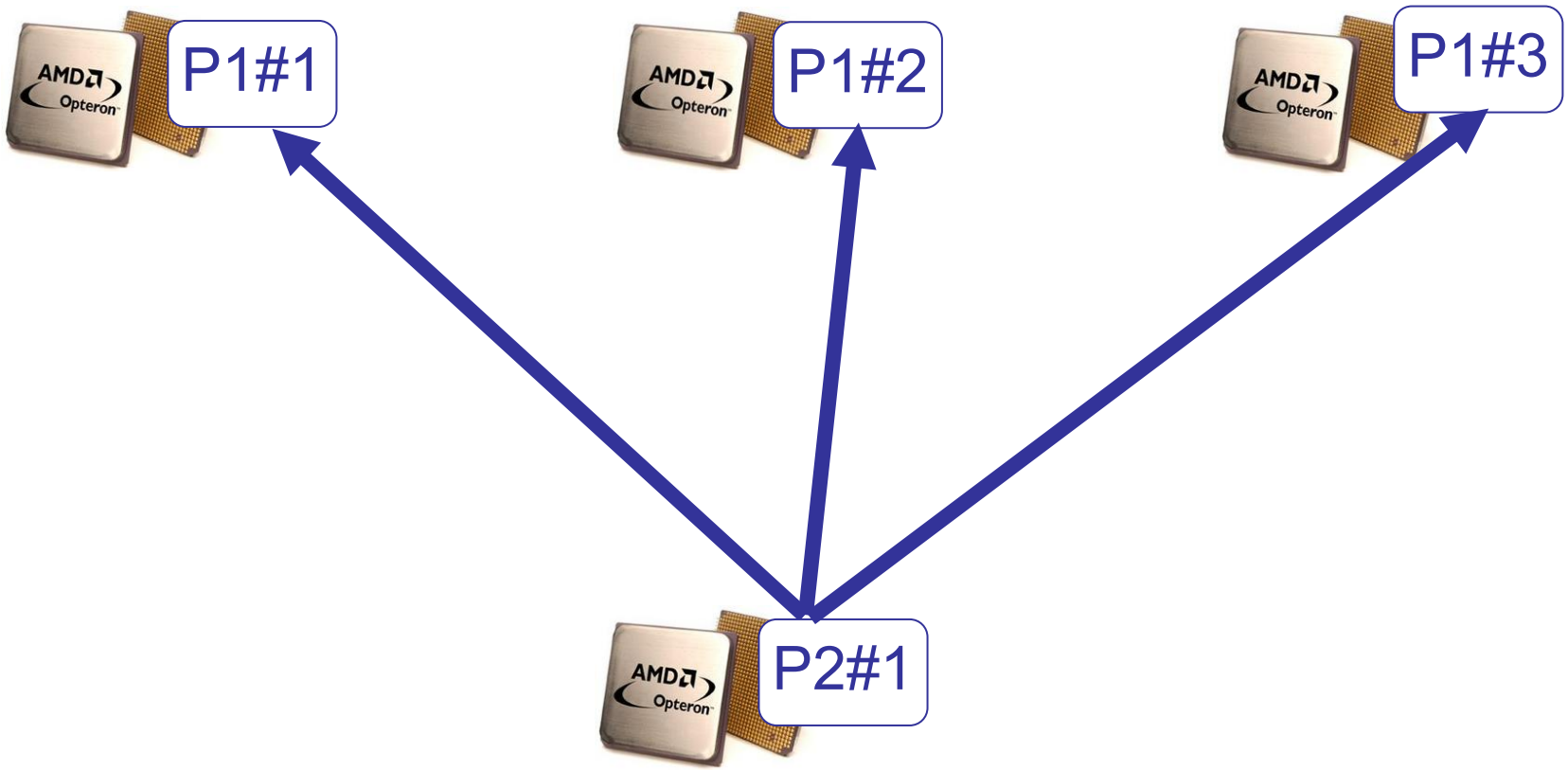
Modèle(s) Pseudosynchrone(s)

- **Pas d'horloge globale**
- **Il existe une borne sur le temps de transit d'un message dans un canal**
- **Equivalent avec l'existence d'un détecteur de défaillances (appelé Oracle) (Chen, Toueg, Aguilera: On the QoS of failure detectors, proc. Of ICDSN/FTCS-30, June 2000)**
- **Représente bien les réseaux de diamètre connu constitué de compostants temps réels (typiquement LAN/Clusters), permet de résoudre le problème! (Chen, Toueg: Unreliable failure detectors for reliable distributed systmes, Journal of the ACM 43(2) march 1996)**

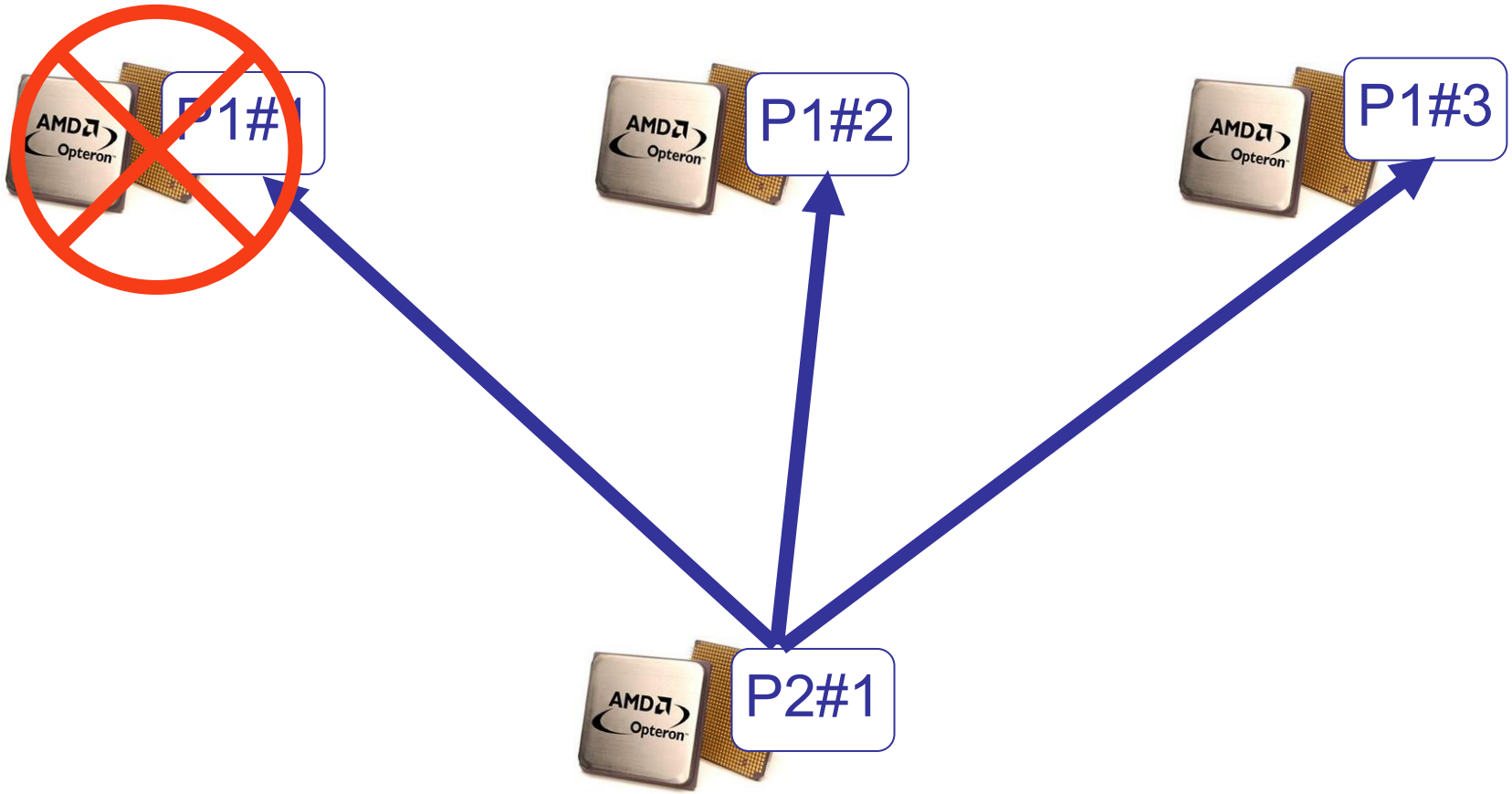
- Qu’est ce que le calcul hautes performances ?
- Système distribué : Modélisation
- **Algorithmes de tolérance aux fautes**
- Une solution pratique : MPICH-V



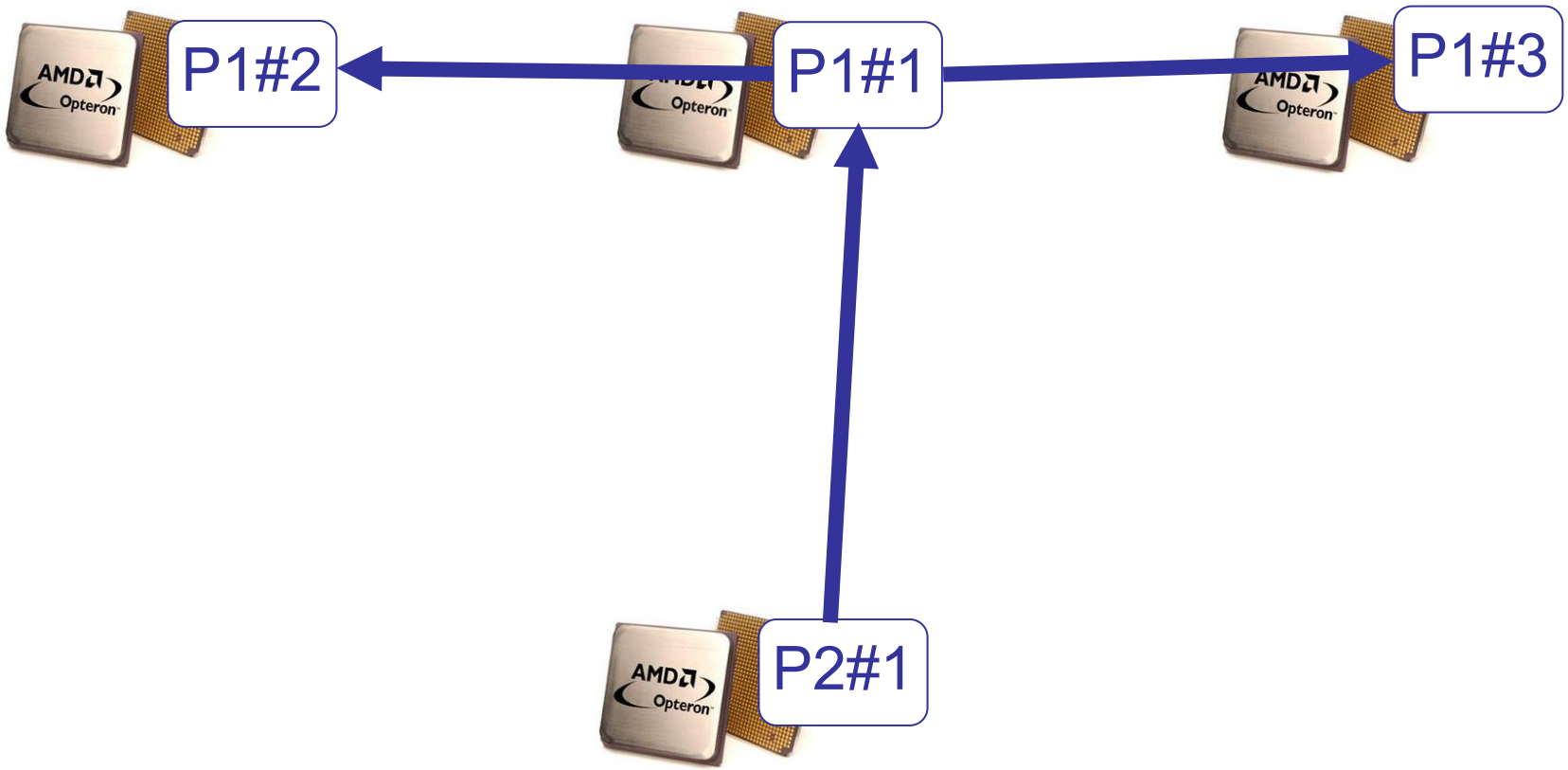
Algorithmes de tolérance aux fautes
Réplication active



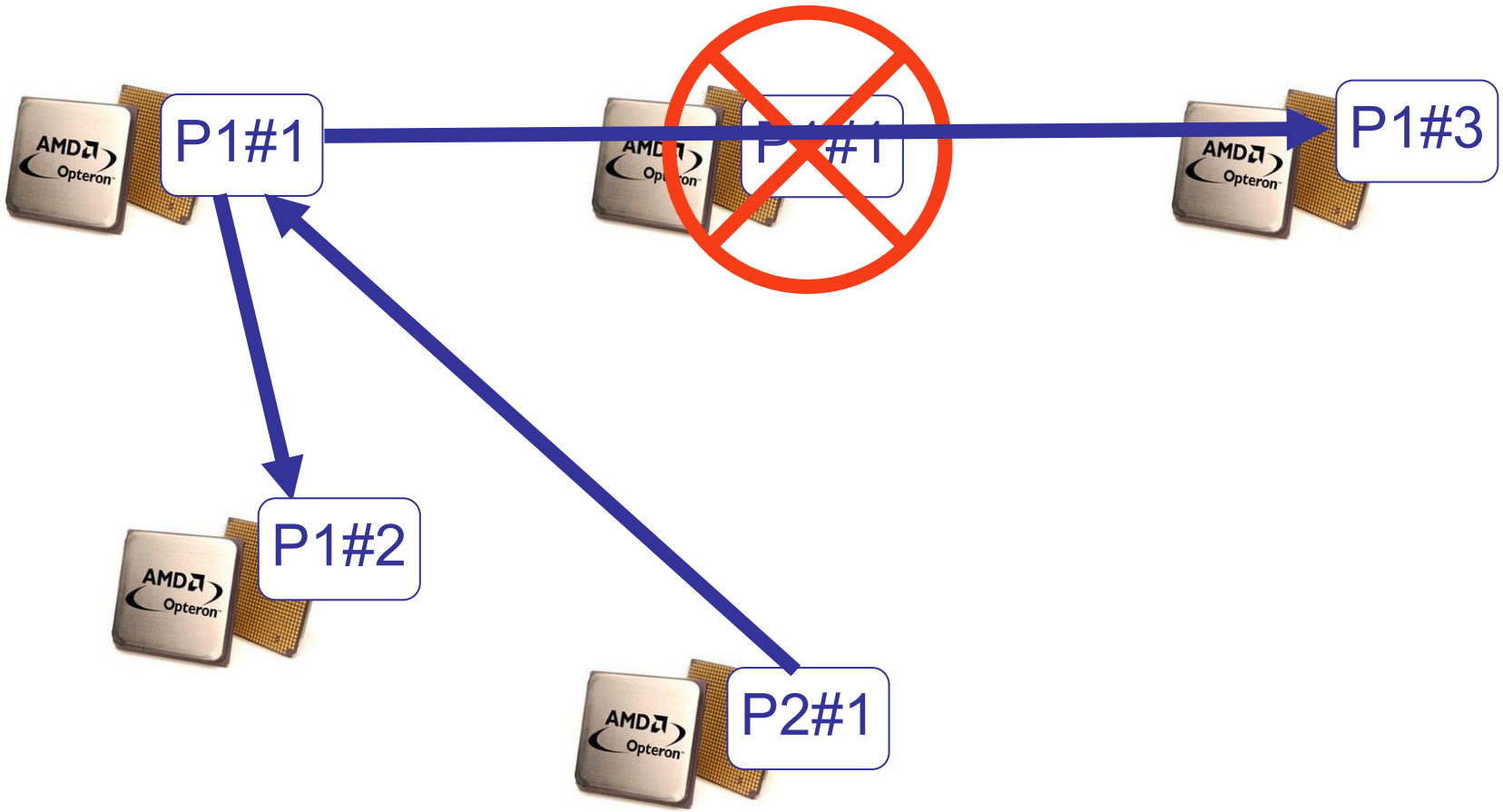
Algorithmes de tolérance aux fautes
Réplication active



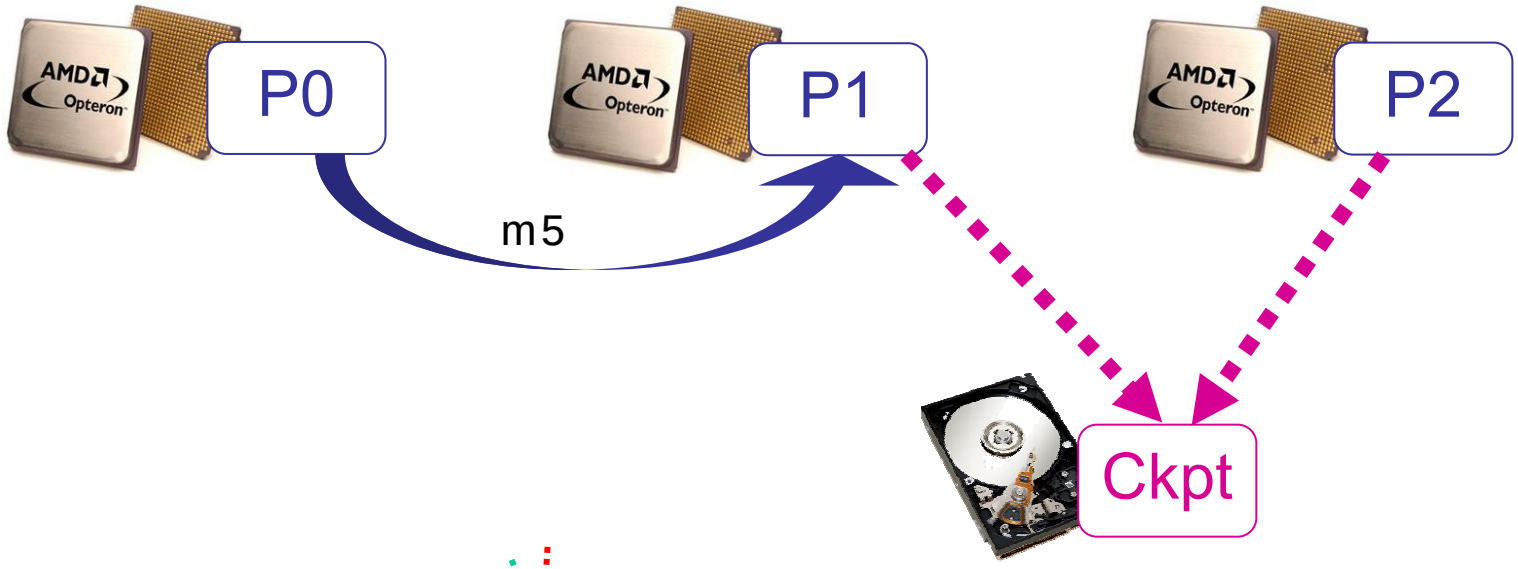
Algorithmes de tolérance aux fautes
Réplication passive

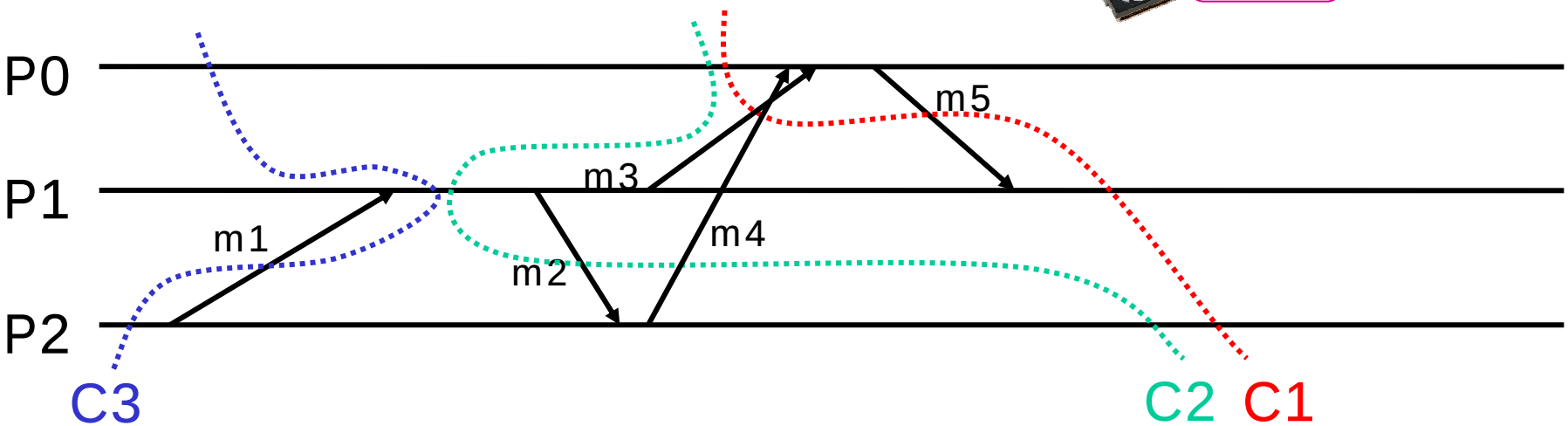


Algorithmes de tolérance aux fautes
Réplication passive

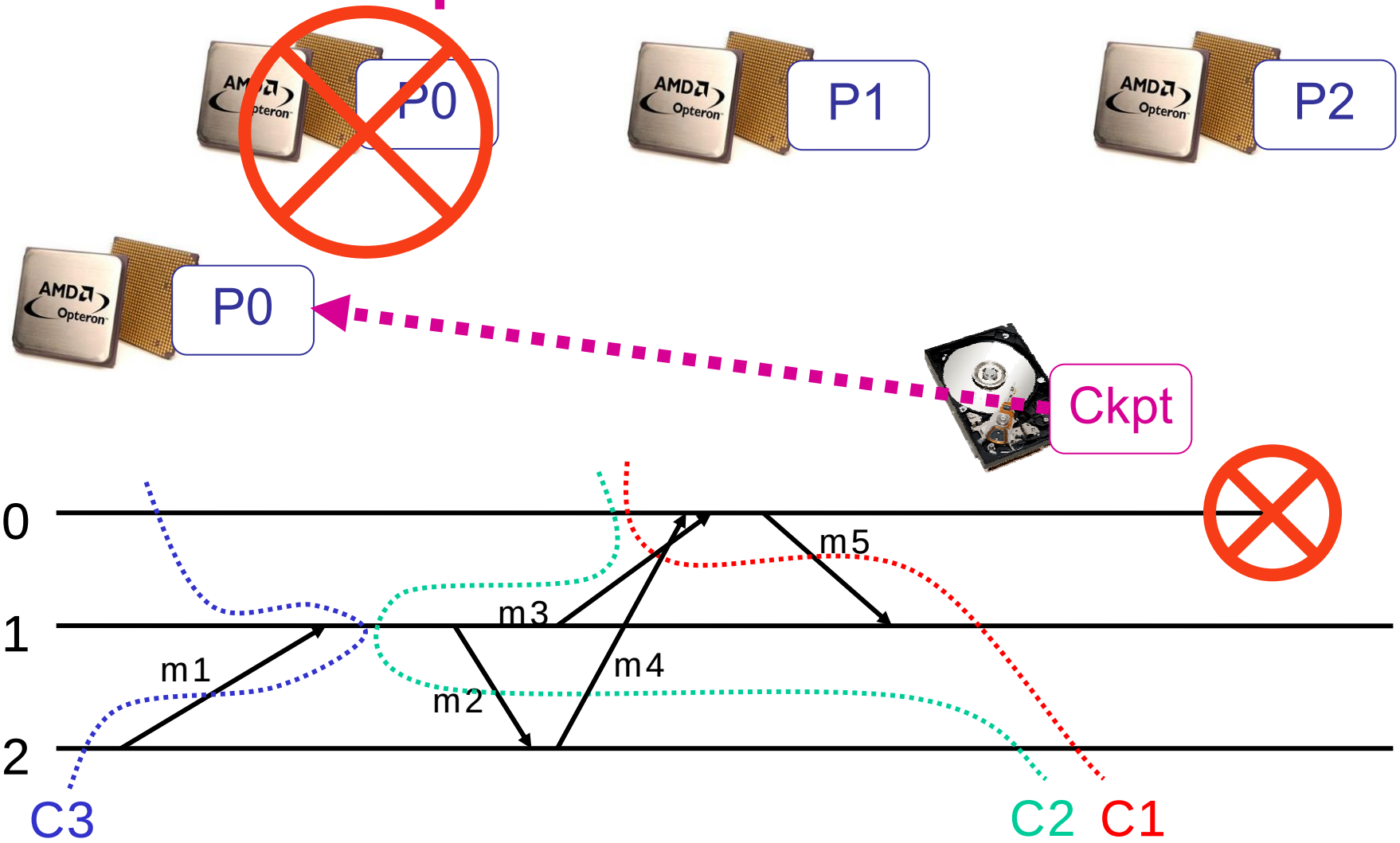


Algorithmes de tolérance aux fautes
Points de reprise





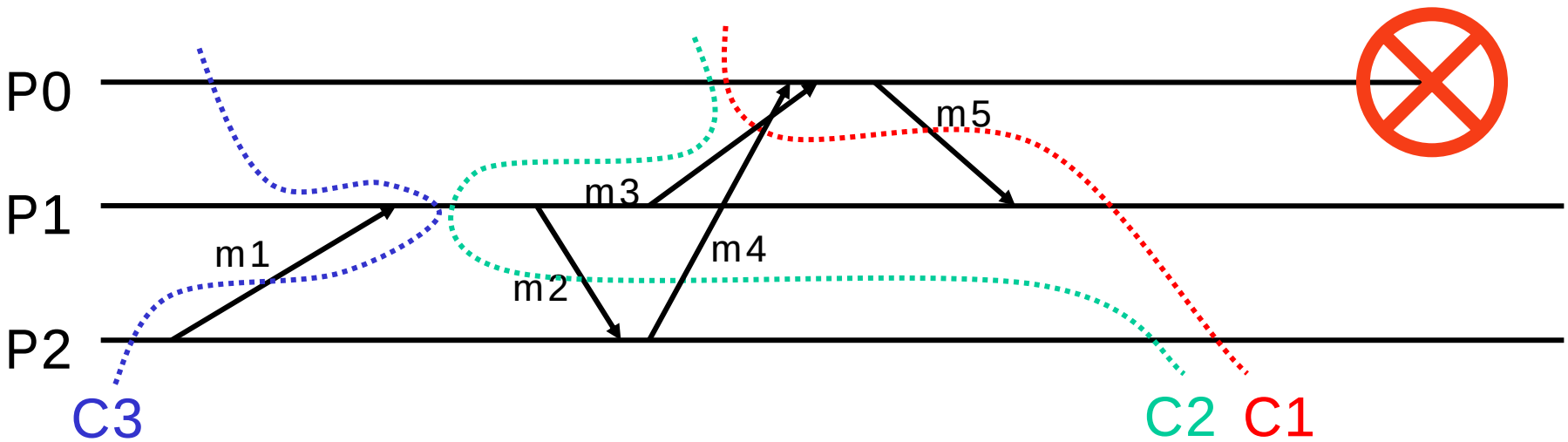
Algorithmes de tolérance aux fautes
Points de reprise



Algorithmes de tolérance aux fautes
Points de reprise

Uncoordinated checkpoint : the problem of inconsistent states

- Order of message receptions are non deterministic events
- ➡ messages received but not sent are inconsistent
- **Domino effect** can lead to rollback to the begining of the execution in case of even a single fault
- Possible loose of the whole execution and unpredictive fault cost



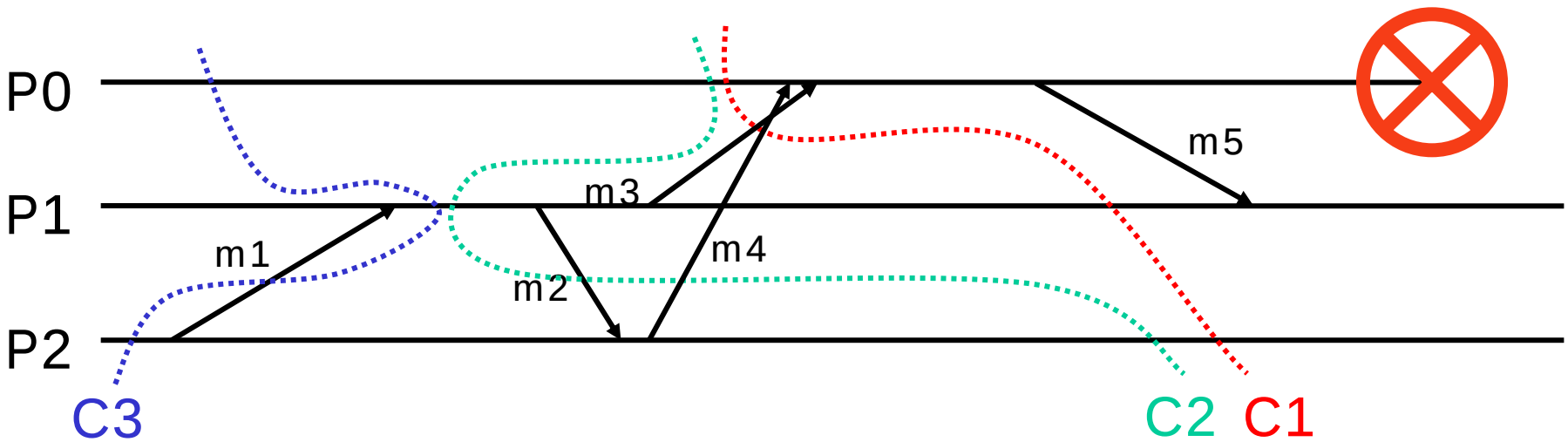
Algorithmes de tolérance aux fautes
État global incohérent

Messages types with respect to a snapshot C

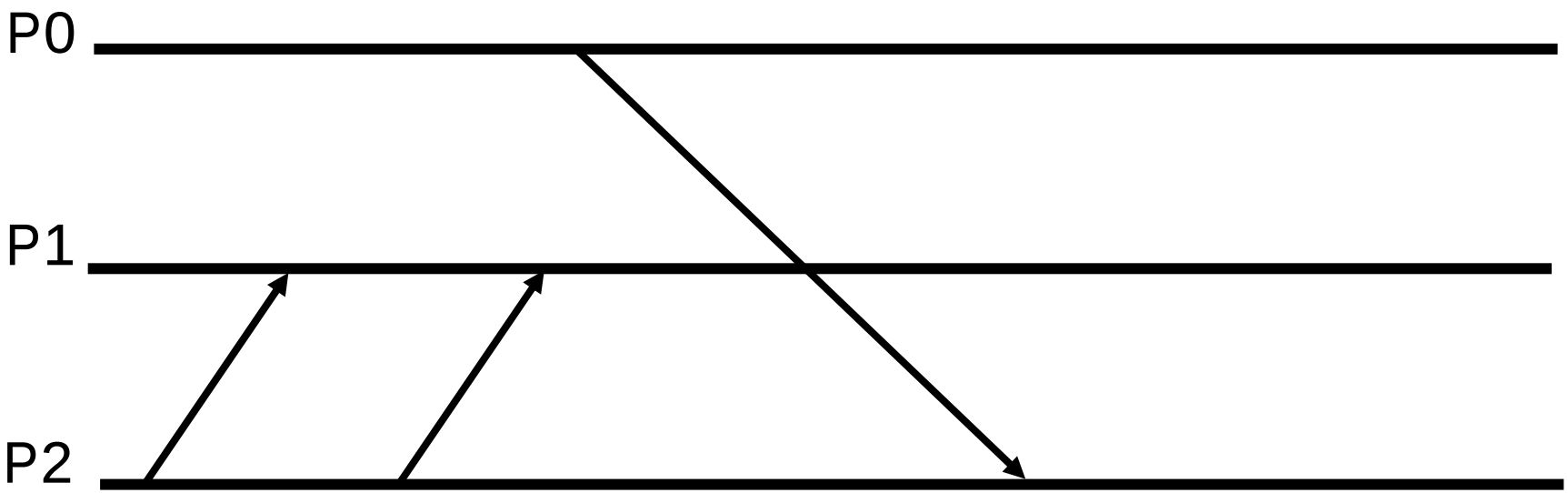
- Past: emission and reception are before C
- Future: emission and reception are after C
- In-transit: emission is before C, reception is after C
- Orphan: emission is after C, reception is before C

Definition: Coherent snapshot

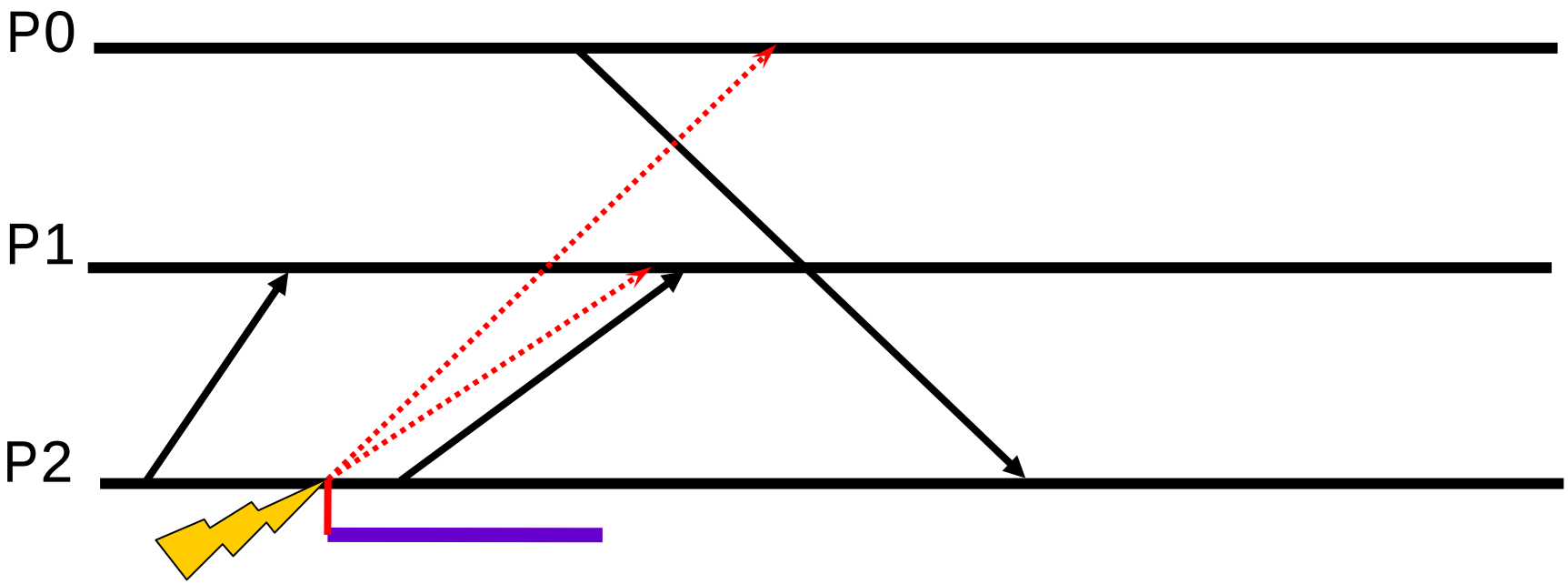
- No orphan messages
- All in-transit messages are logged



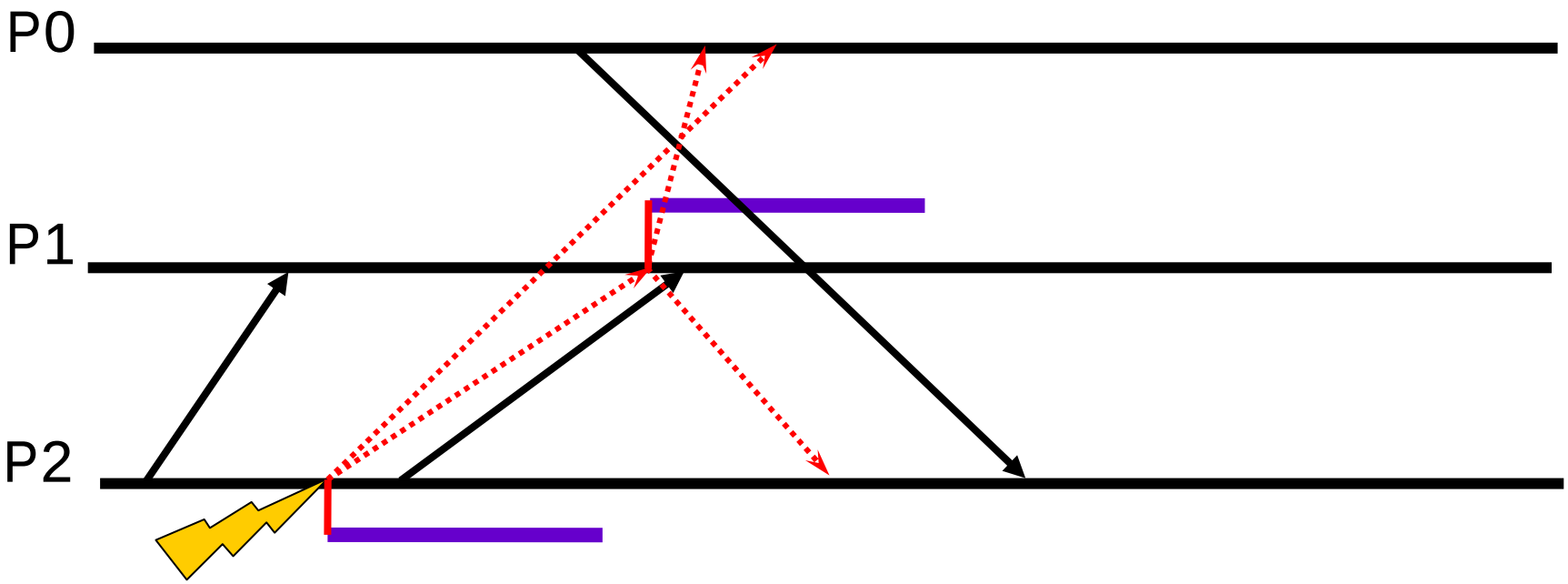
Algorithmes de tolérance aux fautes
Coordinated snapshot (Chandy&Lamport)



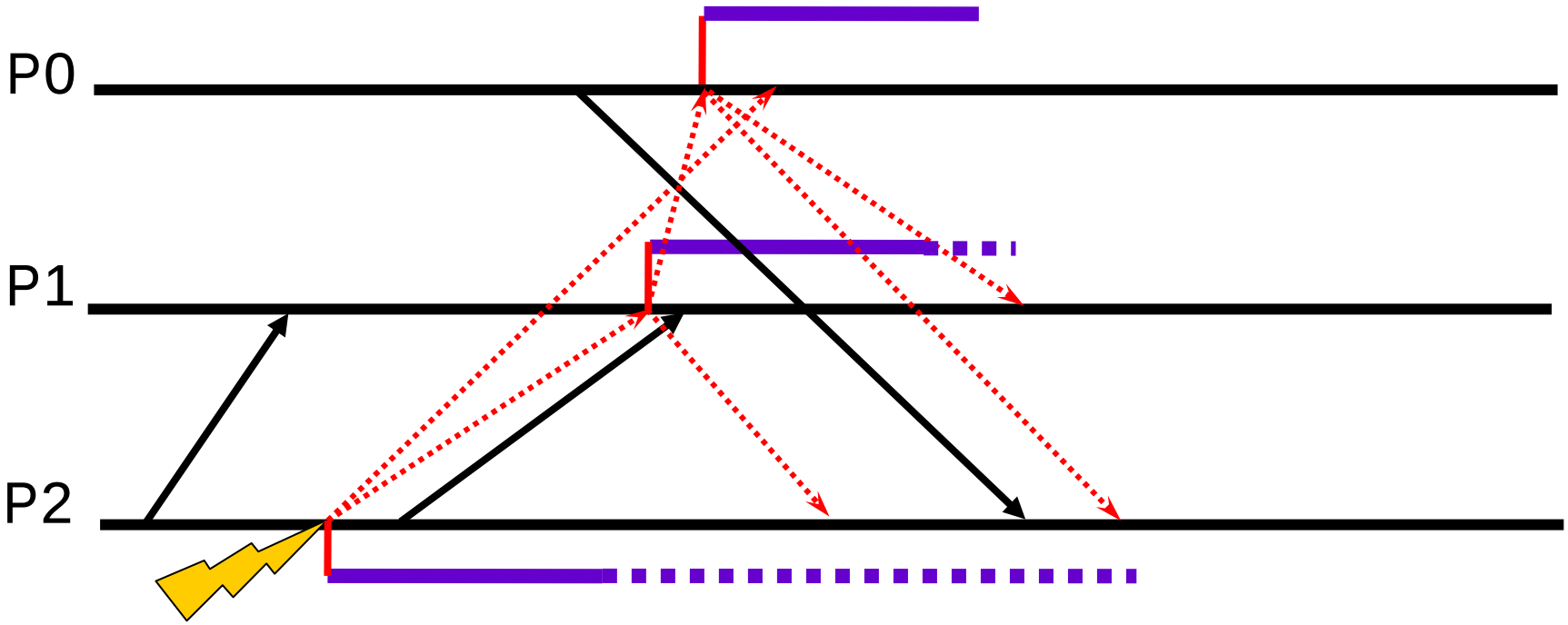
Algorithmes de tolérance aux fautes
Coordinated snapshot (Chandy&Lamport)



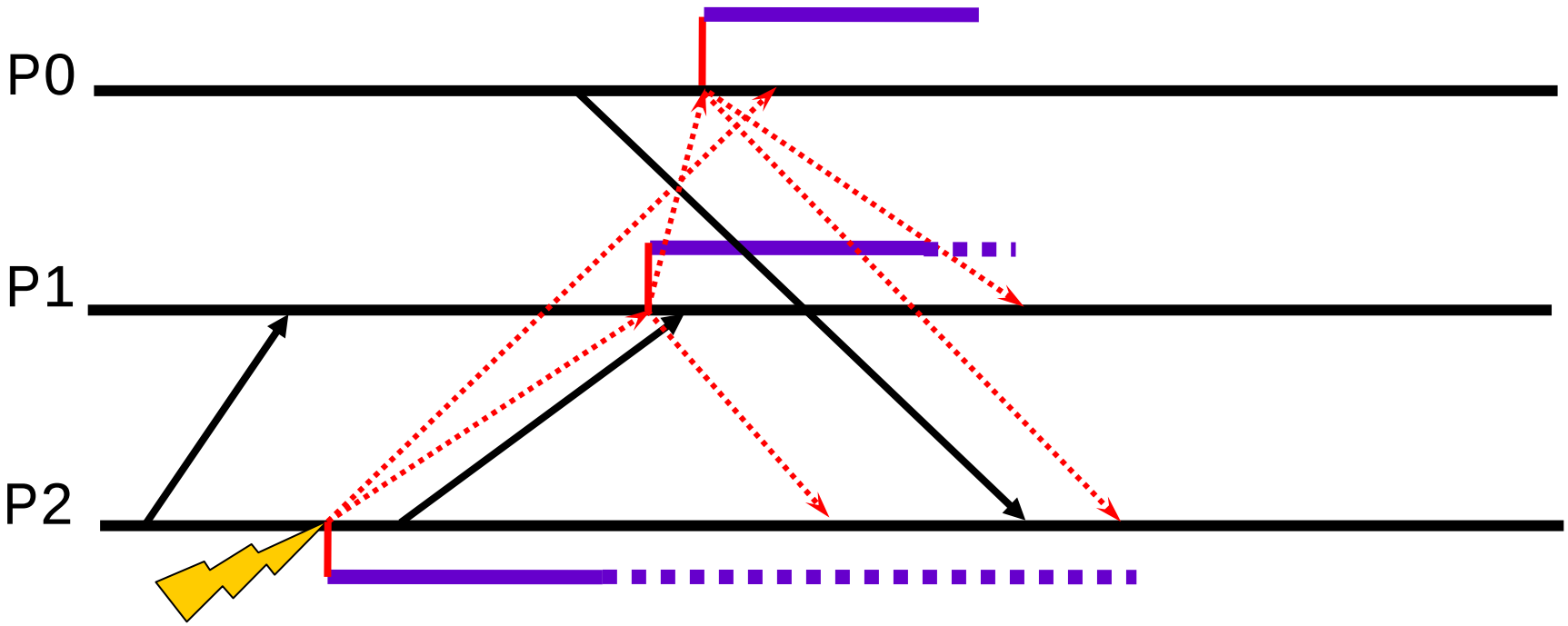
Algorithmes de tolérance aux fautes
Coordinated snapshot (Chandy&Lamport)



Algorithmes de tolérance aux fautes
Coordinated snapshot (Chandy&Lamport)



Algorithmes de tolérance aux fautes
Coordinated snapshot (Chandy&Lamport)

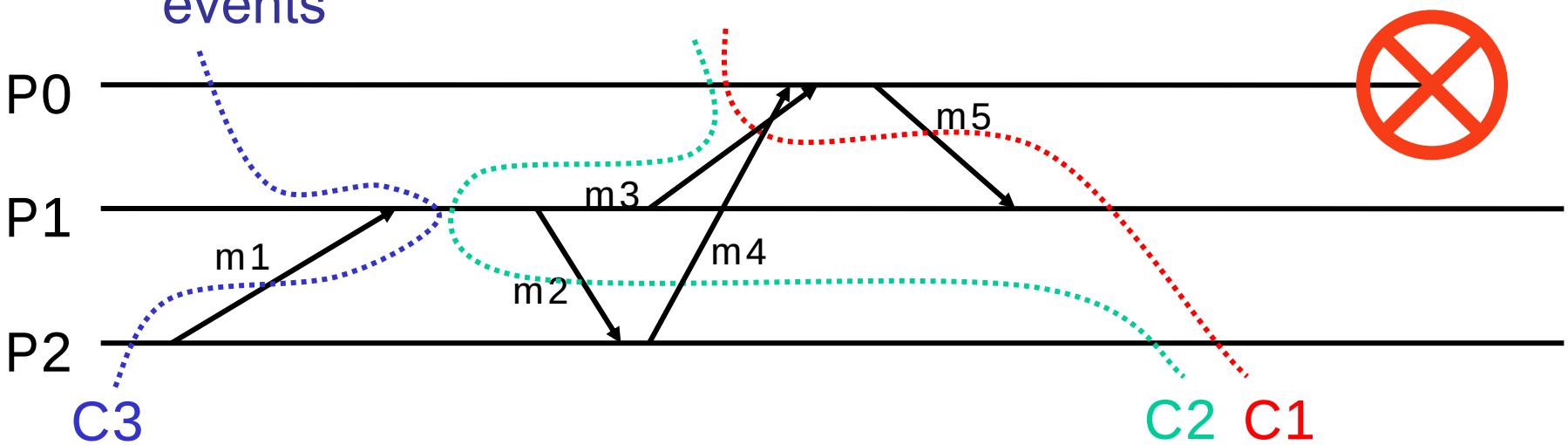


- Negligible overhead on fault free execution
- Requires global synchronization (may take a long time to perform checkpoint because of checkpoint server stress)
- In the case of a single fault, all processes have to roll back to their checkpoints : high cost of fault recovery

Algorithmes de tolérance aux fautes
Enregistrement de messages

Uncoordinated checkpoint : the problem of inconsistent states

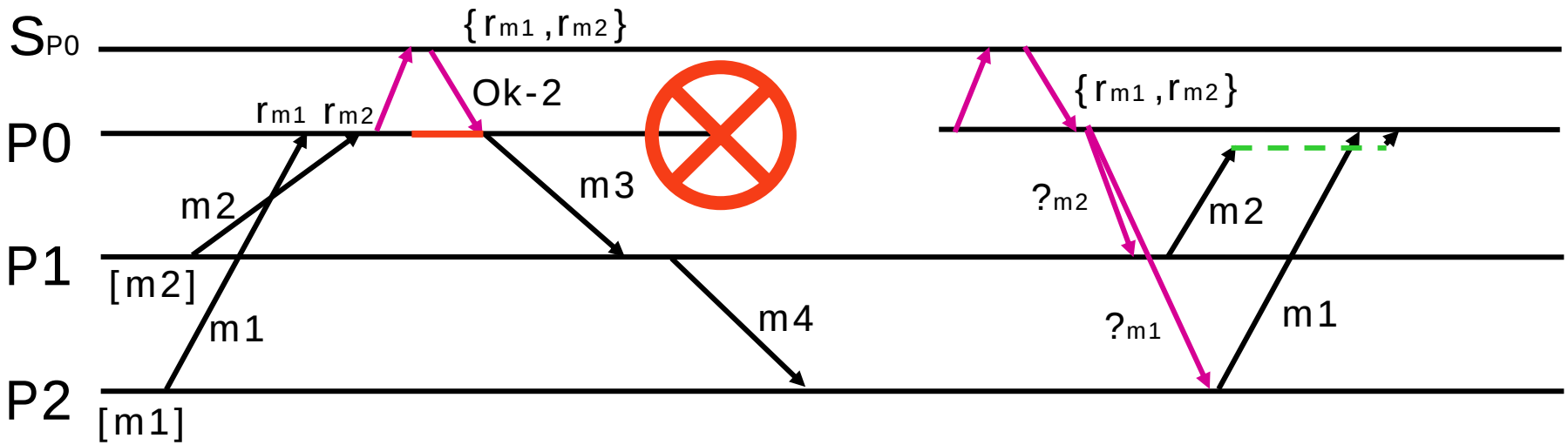
- If process is piecewise deterministic (PWD), recalling of non deterministic events allows a deterministic roll forward
- Order of message receptions are non deterministic events



Algorithmes de tolérance aux fautes
Enregistrement pessimiste sender-based

3 problèmes :

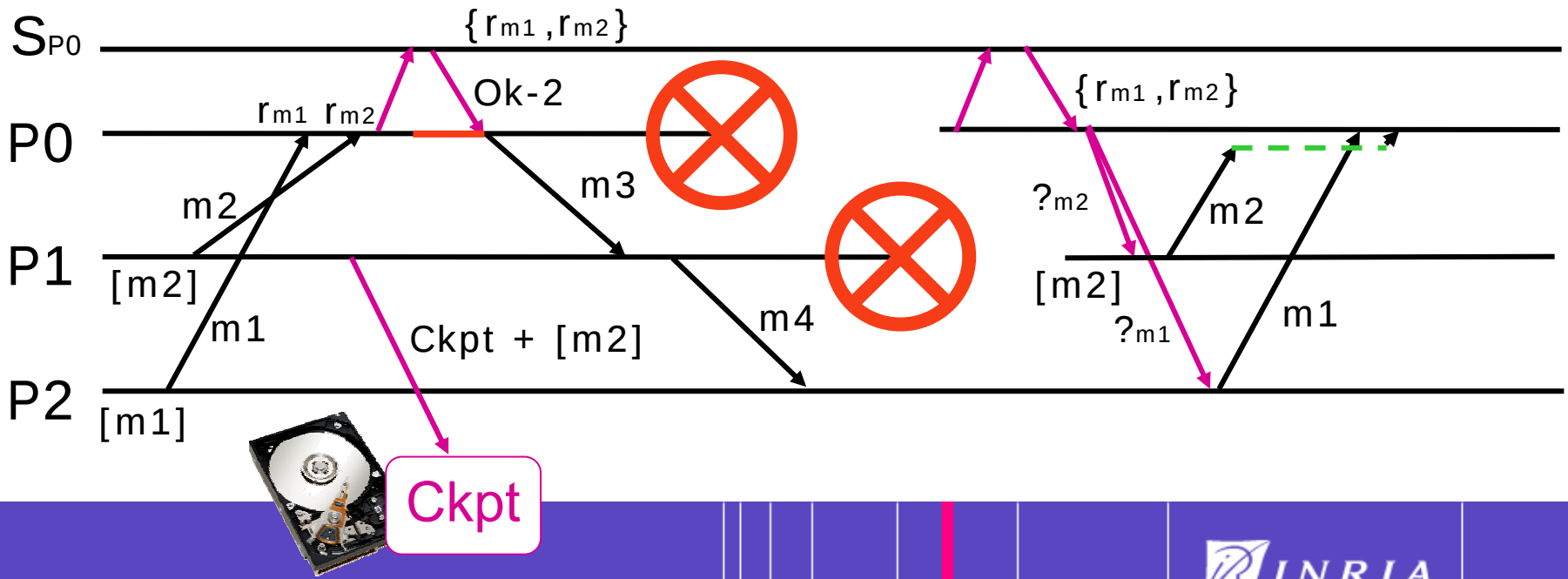
- Messages en-transit : enregistrement du message sur l'émetteur
- Ordre des réceptions non déterministe : enregistrement de l'ordre des évènements sur un support stable
- Propagation transitive de dépendance entre évenements non déterministes : Retardement des émissions



Algorithmes de tolérance aux fautes
Enregistrement pessimiste sender-based

Messages en-transit et fautes multiples :

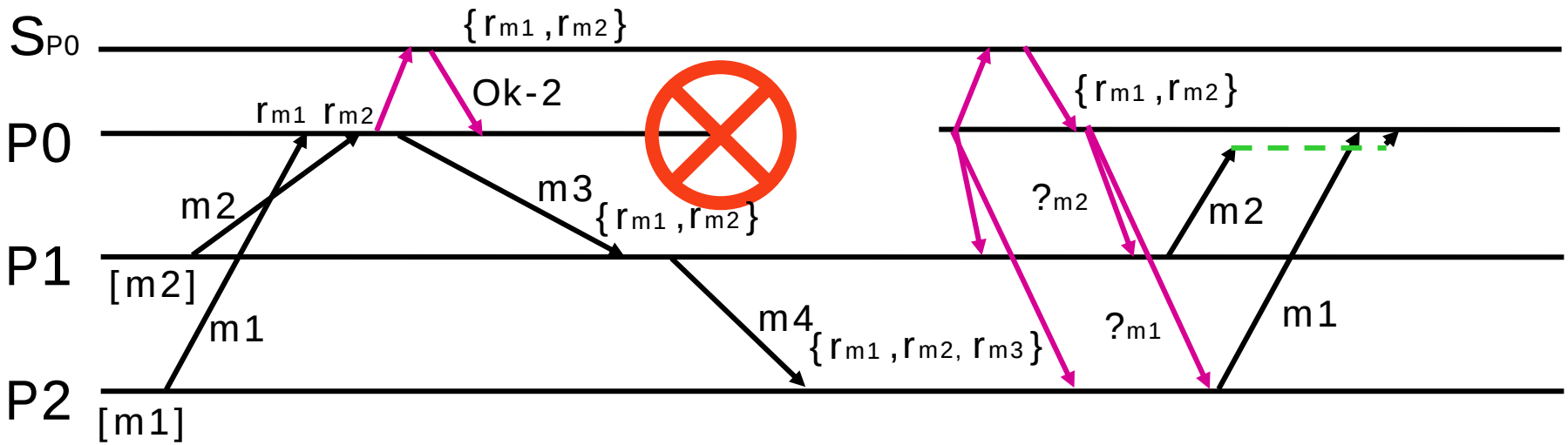
- Messages en-transit : enregistrement du message sur l'émetteur
- Si l'émetteur disparaît, les messages disparaissent : enregistrement en même temps que le point de reprise
- En l'absence de point de reprise, le message est regénéré



Algorithmes de tolérance aux fautes
Enregistrement causal

problème :

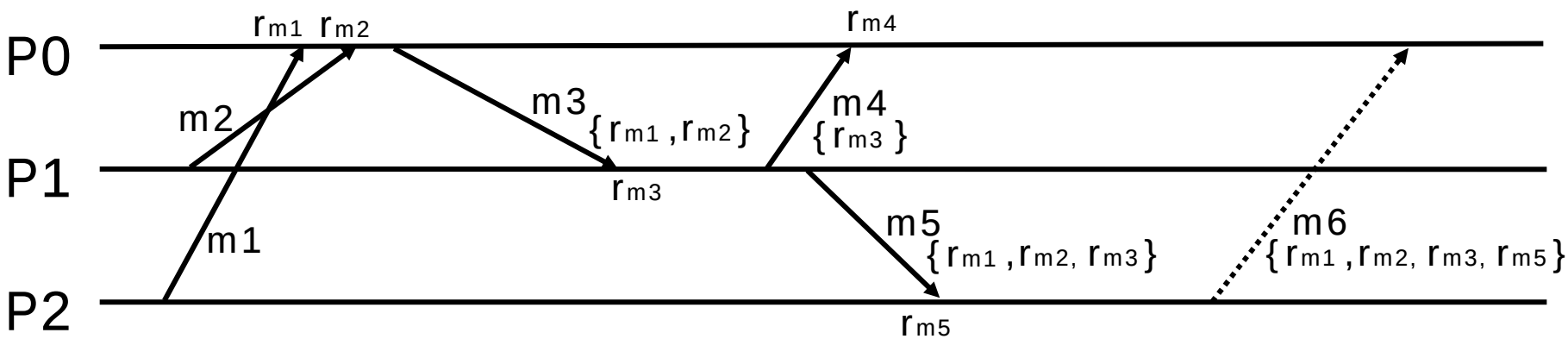
- Interdiction des dépendences transitives : Latence multipliée par 3 avec algo pessimiste
- Lever l'interdiction : propager les évènements vers ceux qui en dépendent



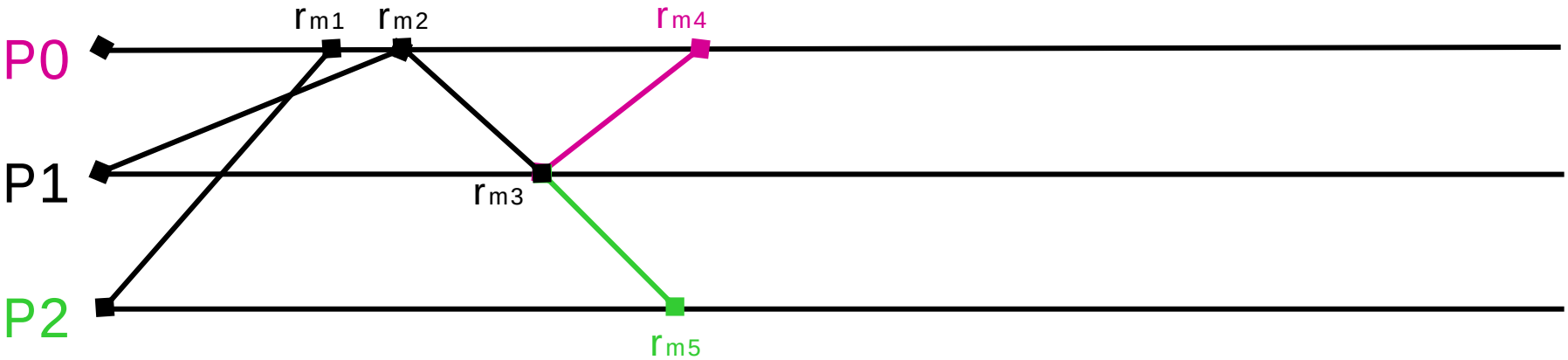
Algorithmes de tolérance aux fautes
Enregistrement causal : graphe

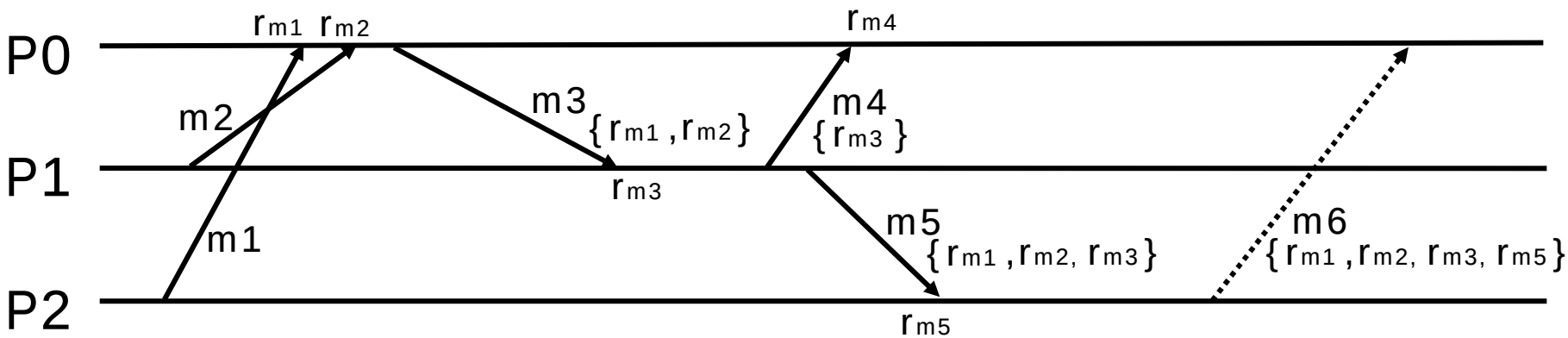
problème :

- La propagation d'évènements coûte cher en bande passante
- Certains évènements envoyés sont déjà connus du destinataire
- La structure de graphe de causalité permet de détecter de tels évènements

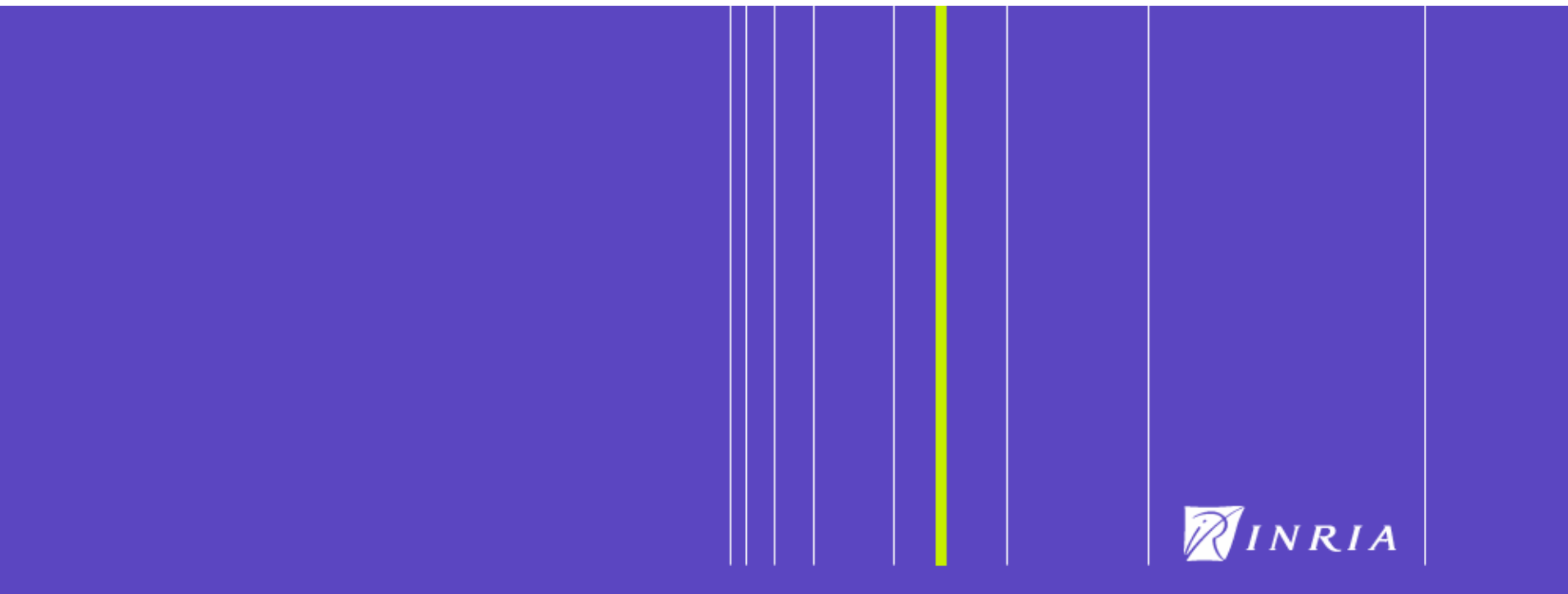


Algorithmes de tolérance aux fautes
Enregistrement causal : graphe





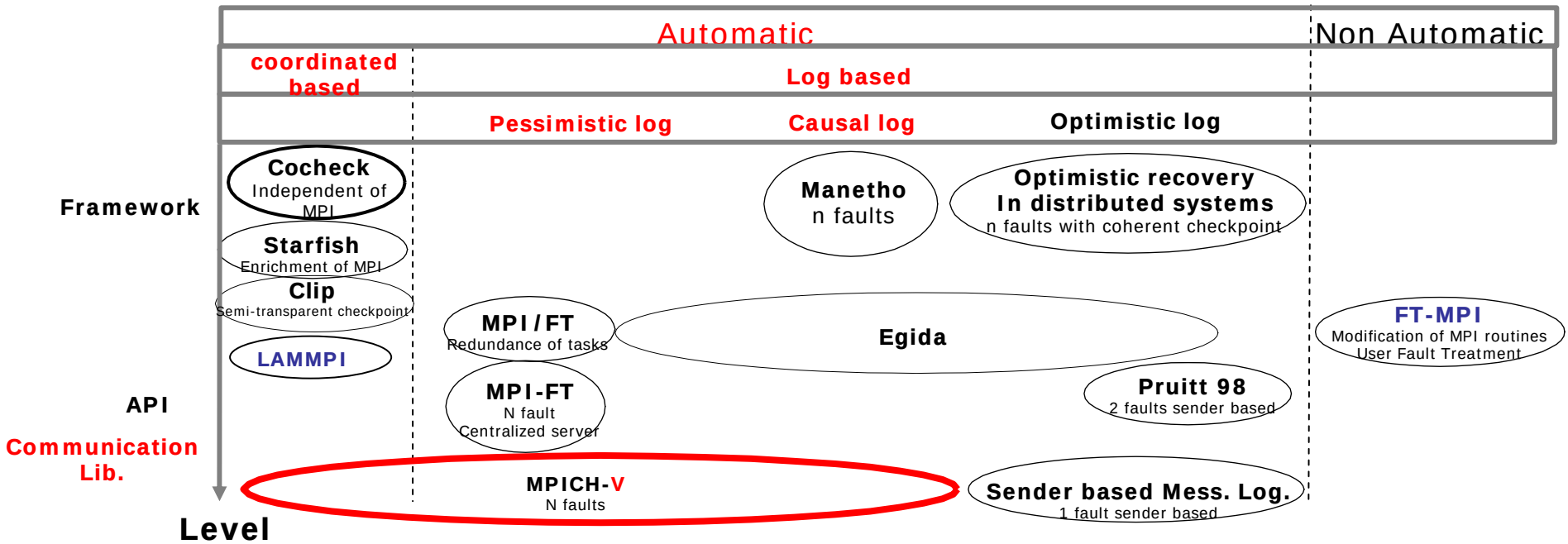
- Qu’est ce que le calcul hautes performances ?
- Système distribué : Modélisation
- Algorithmes de tolérance aux fautes
- **Une solution pratique : MPICH-V**



Une solution pratique : MPICH-V
Fault tolerant MPI implementations

Classification des implémentations suivant deux critères

- Niveau dans la pile logicielle MPI
- Algorithme de tolérance aux défaillances utilisé

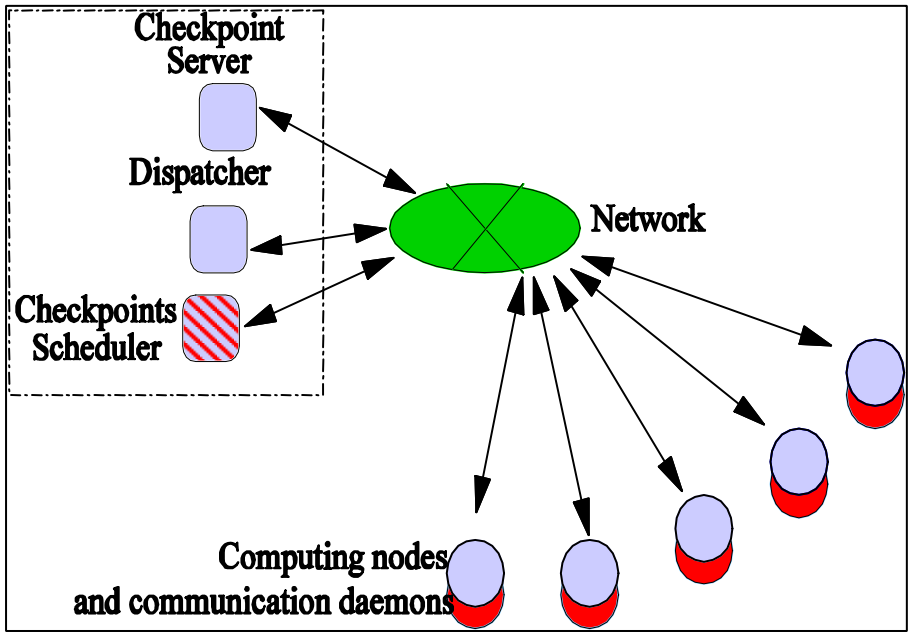


Several protocols to perform fault tolerance in MPI applications with N faults and automatic recovery : Global checkpointing, Pessimistic/Causal Message log

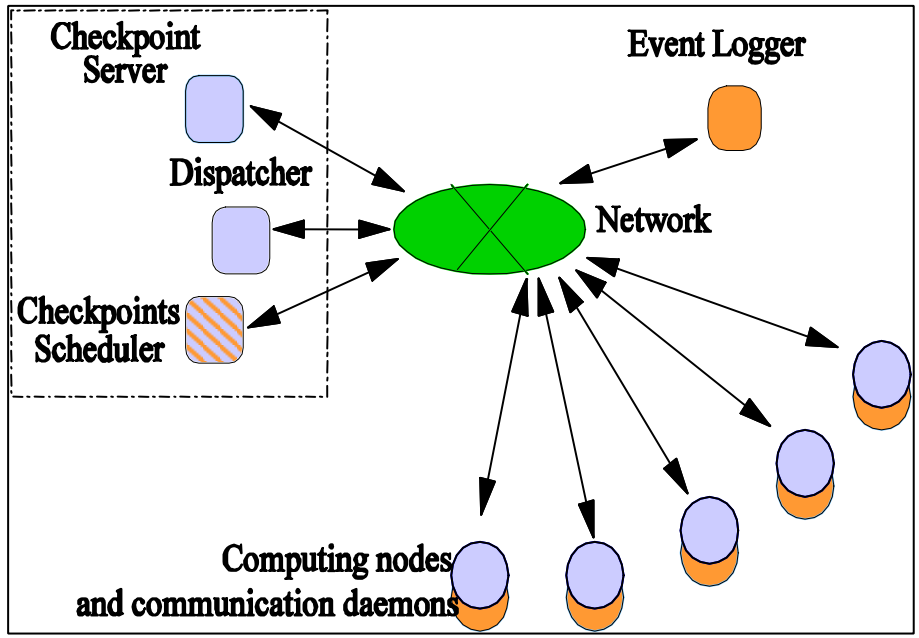
compare fault tolerant protocols for a single MPI implementation

Une solution pratique : MPICH-V
Fault tolerant MPI implementations

We designed MPICH-V to perform a fair comparison of coordinated checkpoint and pessimistic message log



MPICH-Vcl
Chandy&Lamport algorithm
Coordinated checkpoint



MPICH-V for message logging protocols (causal, pessimist)

Une solution pratique : MPICH-V
Generic device in MPICH-1.2.5.2

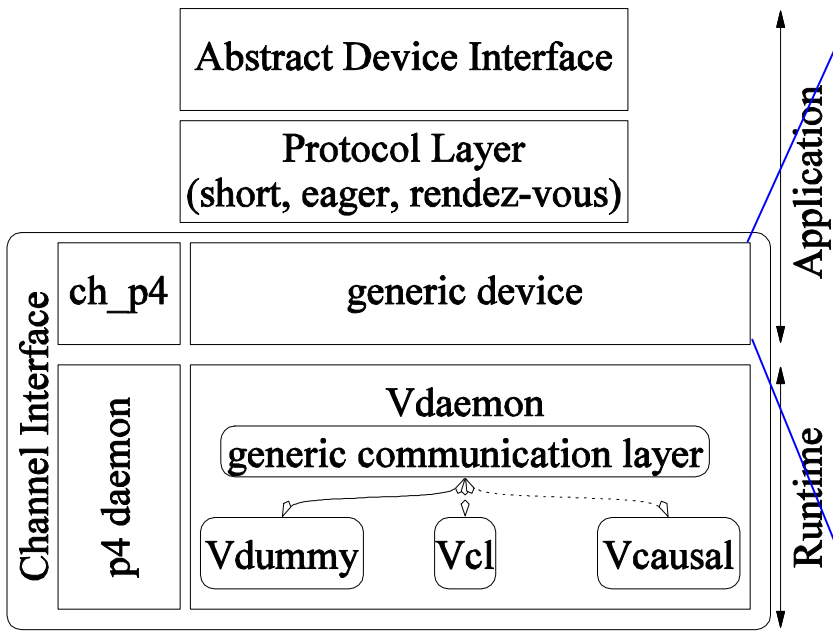
- A new device: 'ch_v' device
- All ch_v device functions are blocking communication functions built over TCP layer
- Non blocking communications sits in the communication daemon

MPI_Send

MPID_SendControl
MPID_SendChannel

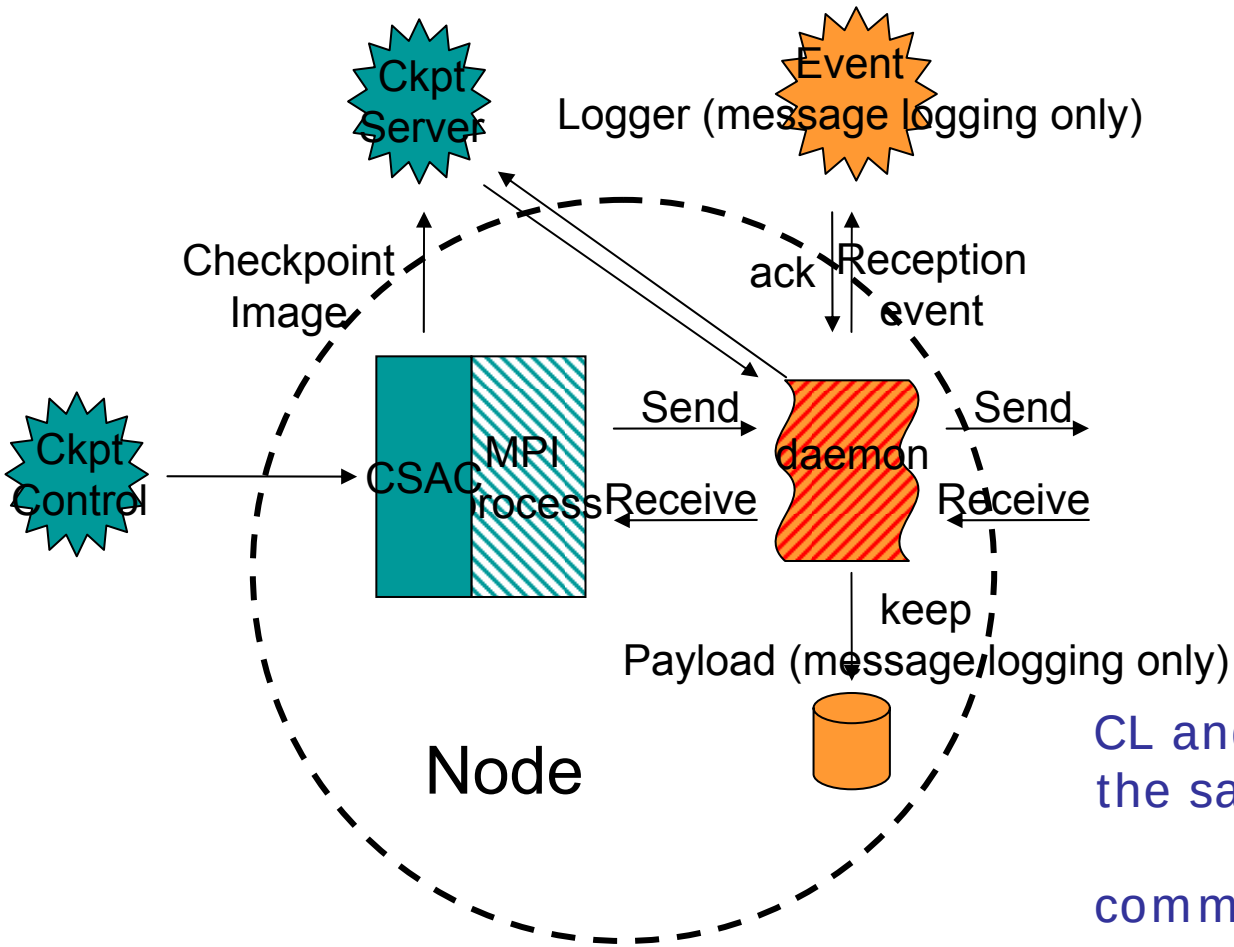
Binding

_bsend



_bsend	- blocking send
_brecv	- blocking receive
_probe	- check for any message avail.
_from	- get the src of the last message
_Init	- initialize the client
_Finalize	- finalize the client

Une solution pratique : MPICH-V
Role of the Vdaemon



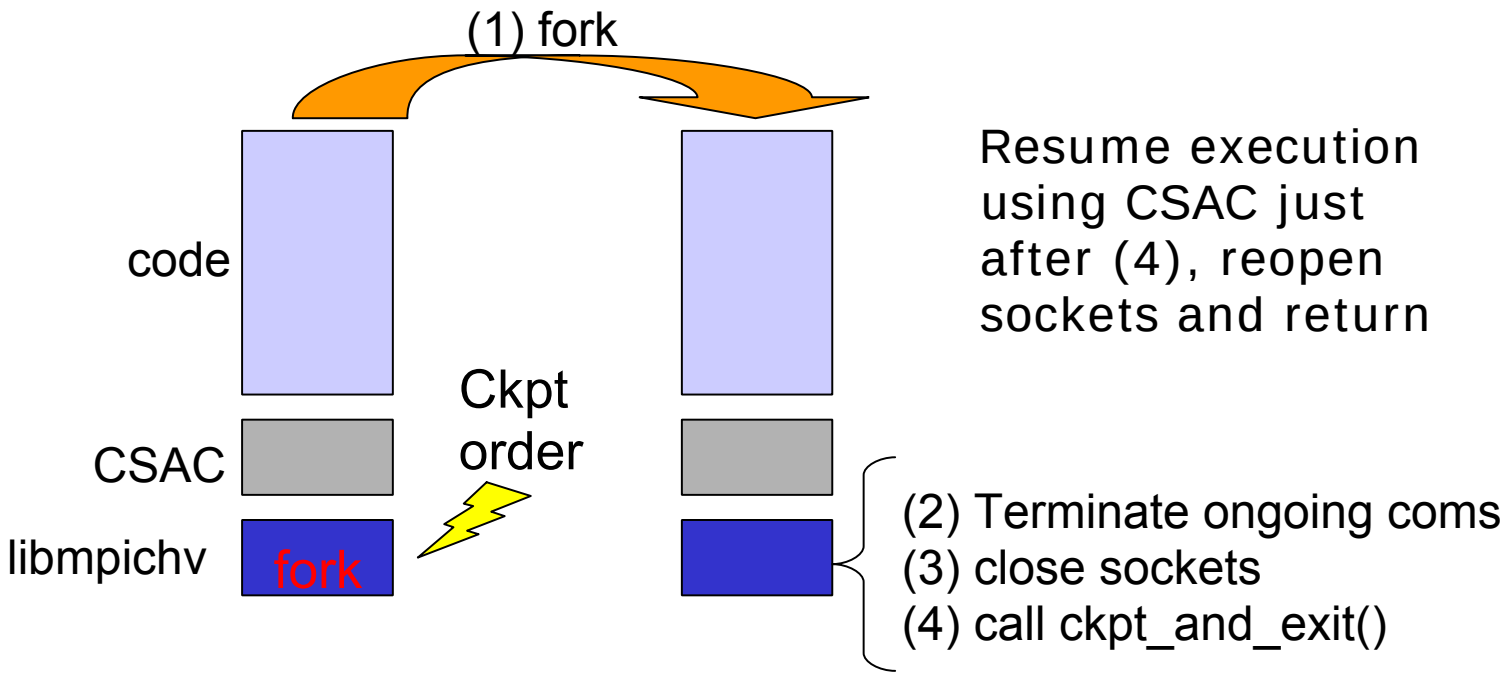
CL and Vpessimist/Vcausal share the same architecture

communication daemon includes protocol specific actions

Une solution pratique : MPICH-V
Taking checkpoints

User-level Checkpoint : Condor Stand Alone Checkpointing

Clone checkpointing + non blocking checkpoint



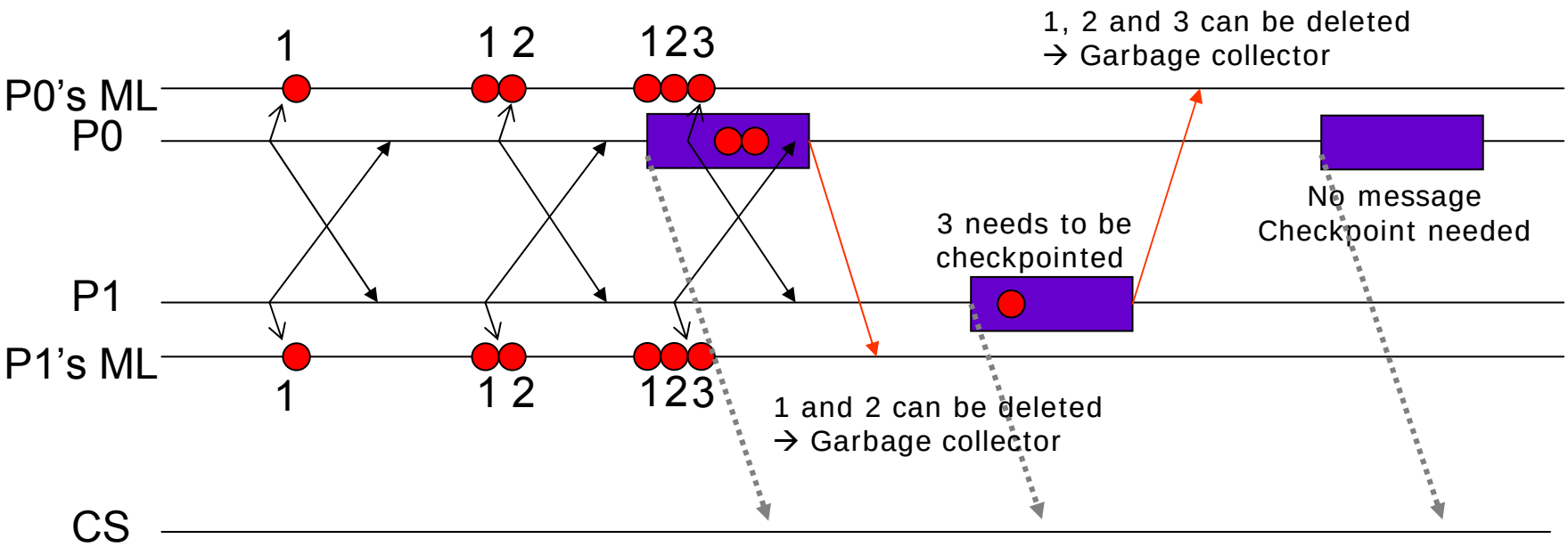
- Checkpoint image is sent to reliable CS on the fly

Une solution pratique : MPICH-V
Scheduling checkpoints

Uncoordinated checkpoint implies all messages are possibly in-transit messages

- Checkpoint traffic should be flattened
- Checkpoint size can be reduced by removing message logs
- Message received before a checkpoint are no more in-transit

Checkpoint scheduling should evaluate the cost and benefit of each checkpoint



Une solution pratique : MPICH-V
Experimental conditions

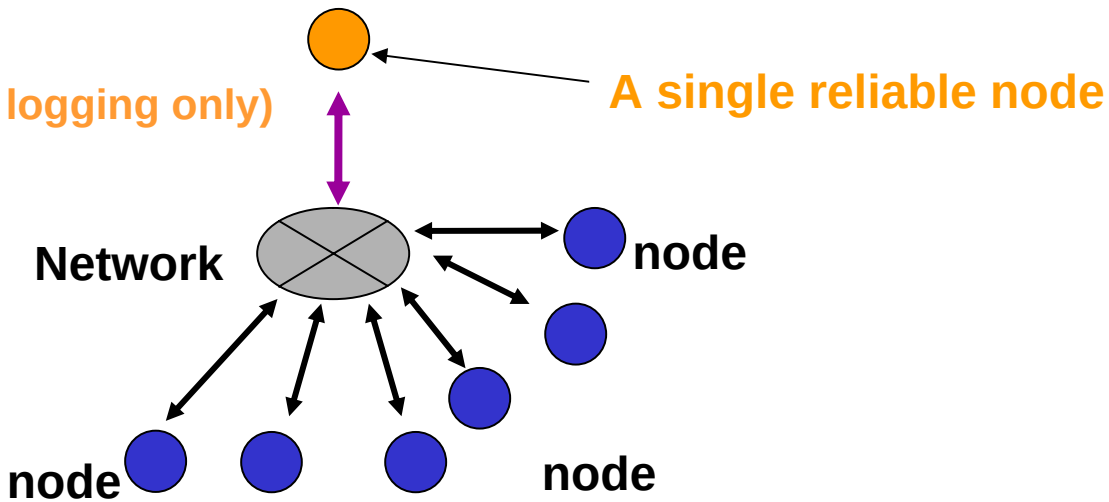
Ethernet experiments
- 32 2800+ AthlonXP CPU, 1 GB DDR SDRAM, 70GB ATA100 IDE Disc
- 100Mbits/s Ethernet card connected by a single Fast Ethernet Switch

Myrinet experiments:
- 8 2200+ AthlonXP-MP CPU, 1 GB DDR SDRAM, 70GB ATA100 IDE Disc
- Myrinet2000 connected by a single 8-port myrinet switch

SCI experiments
- 32 2800+ AthlonXP CPU, 1 GB DDR SDRAM, 70GB ATA100 IDE Disc
- 2D-torus topology SCI

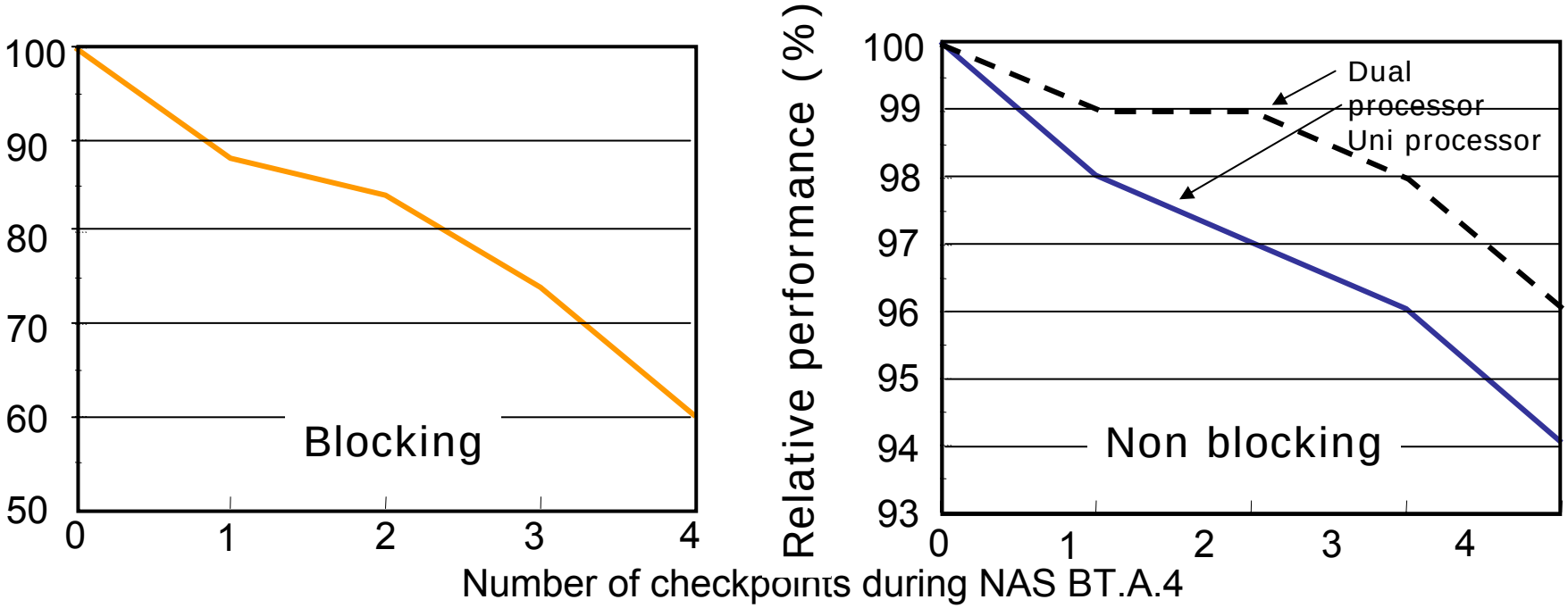
Linux 2.4.20, GCC 2.96 (-O3), PGI Fortran <5 (-O3, -tp=athlonxp)

Checkpoint Server
+Event Logger (message logging only)
+Checkpoint Scheduler
+Dispatcher



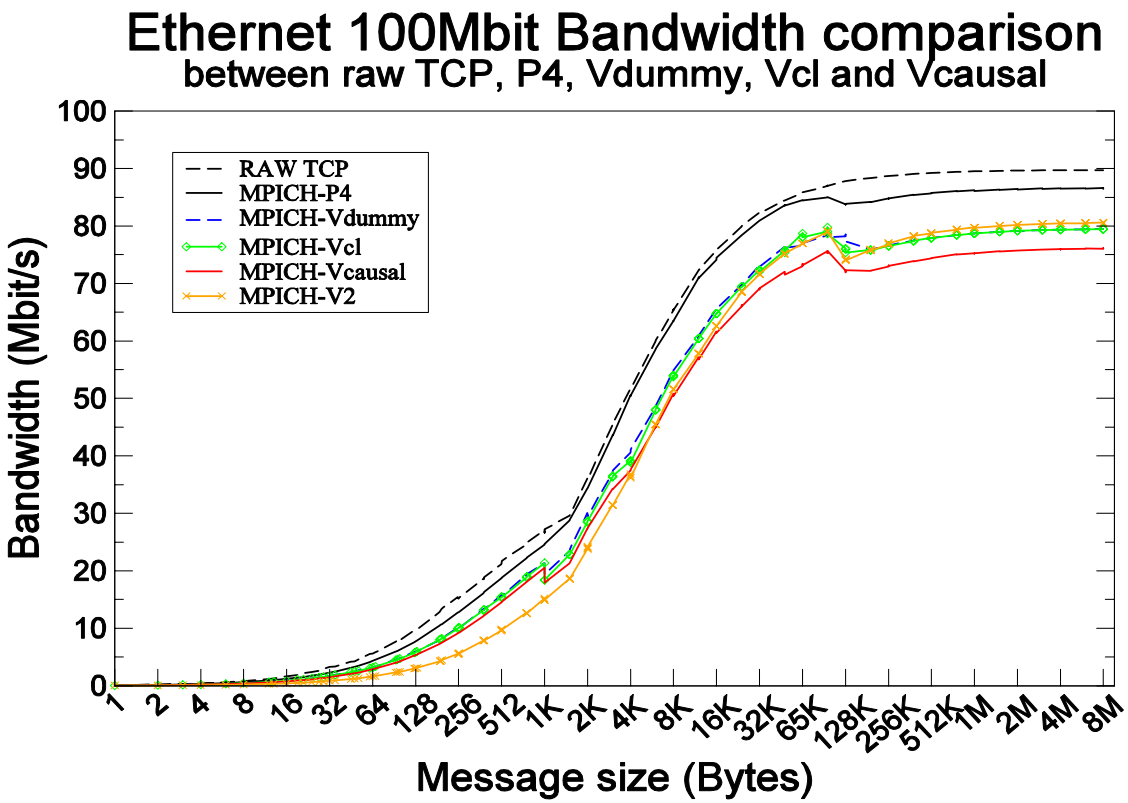
Une solution pratique : MPICH-V
Impact of checkpointing on app. perf.

A single checkpoint server for 4 MPI tasks (P4 driver)
Ckpt is performed at random time on each node (no sync.)



→When 4 checkpoints are performed per process performance is about 94% the one of a non checkpointed execution.
→Several nodes can use the same CS

Une solution pratique : MPICH-V
Ethernet Bandwidth Comparison



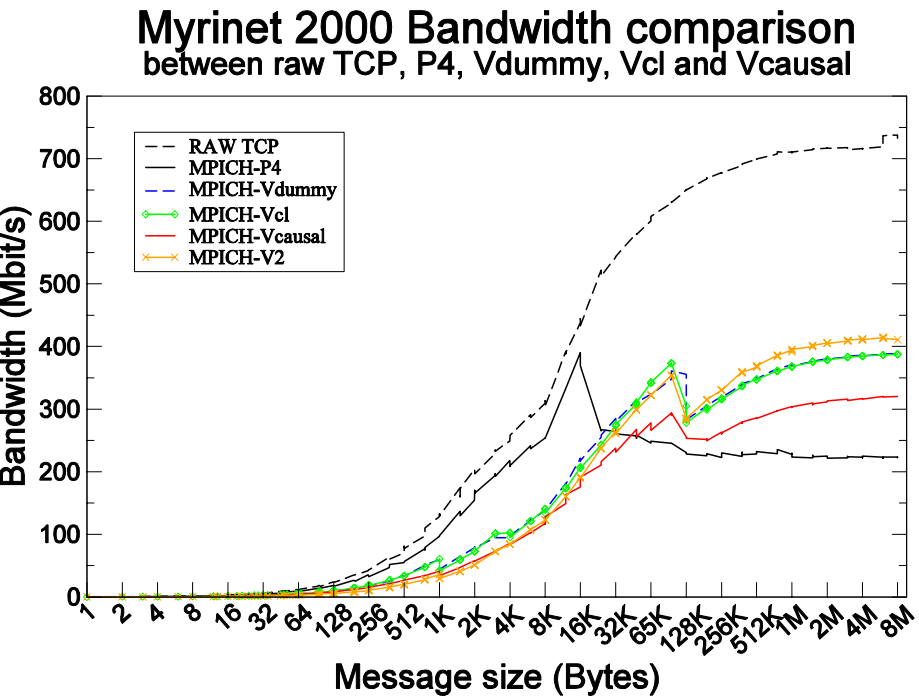
Latency for a 1 byte message :

TCP	(75us)
MPICH-P4	(100us)
MPICH-V	(135us)
MPICH-Vcl	(138us)
MPICH-Vcausal	(157us)
MPICH-V2	(291us)

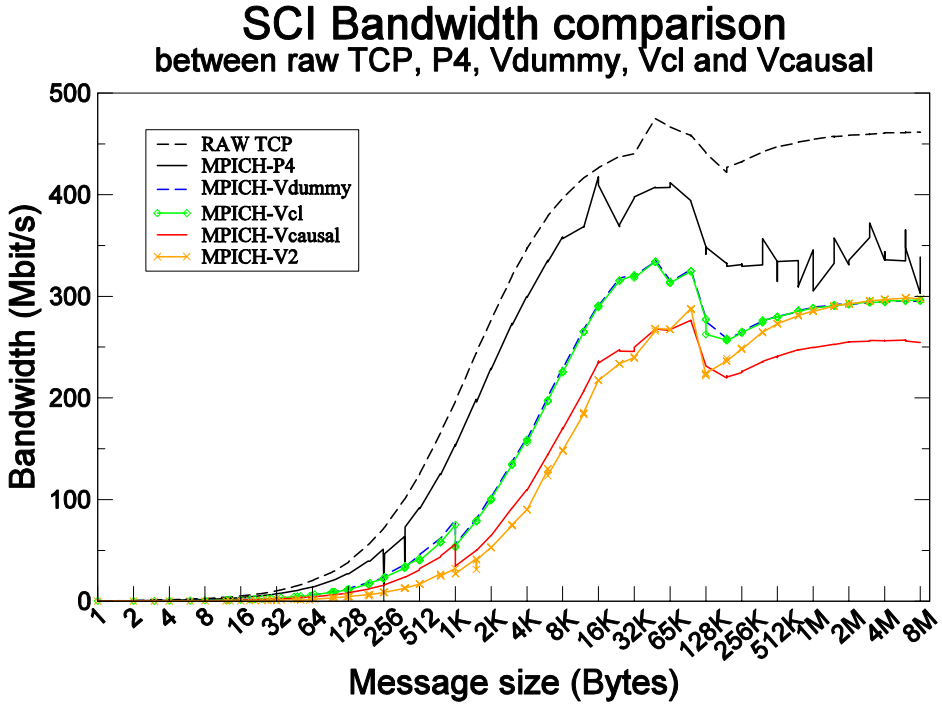
Latency is high in MPICH-V2 due to the event logging.

→ A receiving process can send a new message only when the reception event has been successfully logged (3 TCP messages for a communication)

Une solution pratique : MPICH-V
High perf. NIC Bandwidth Comparison

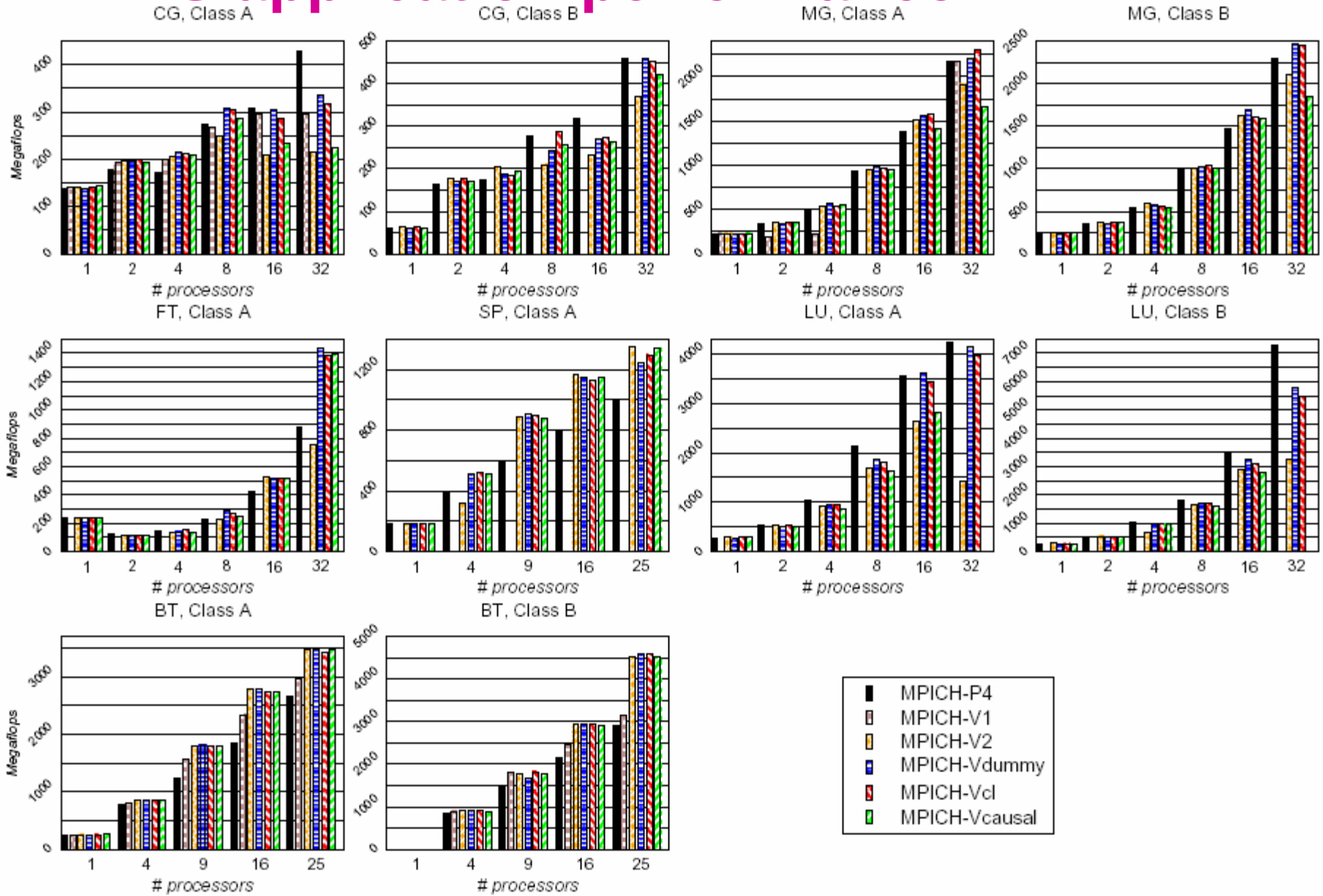


TCP (43us)
MPICH-P4 (53us)
MPICH-V (94us)
MPICH-Vcl (99us)
MPICH-Vcausal(112us)
MPICH-V2 (183us)

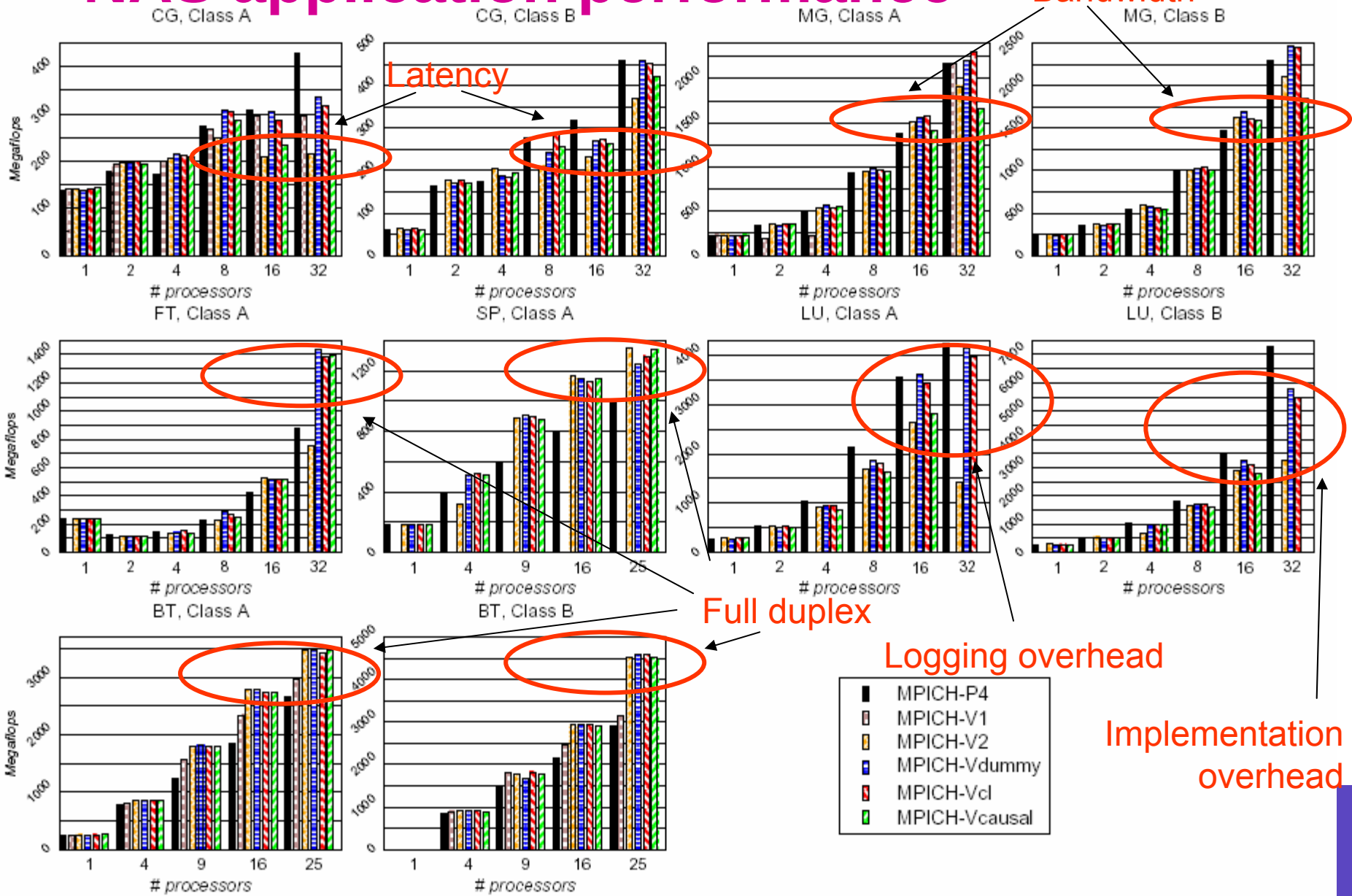


TCP (23us)
MPICH-P4 (34us)
MPICH-V (76us)
MPICH-Vcl (81us)
MPICH-Vcausal(116us)
MPICH-V2 (355us)

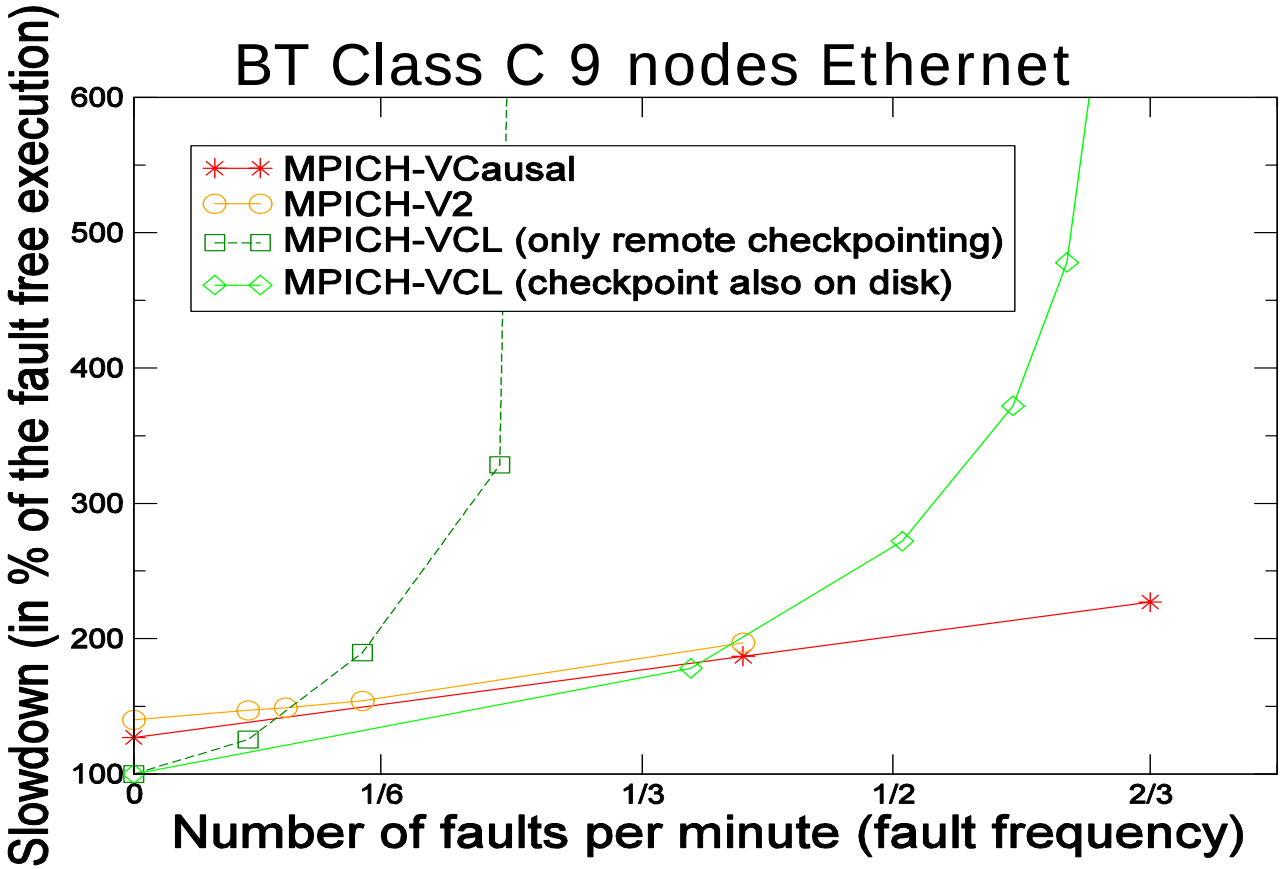
Une solution pratique : MPICH-V
NAS application performance



Une solution pratique : MPICH-V
NAS application performance



Une solution pratique : MPICH-V
Impact of fault frequency



If we consider a 1GB memory occupation for every process, an extrapolation expects the crosspoint to appear around one fault every 9 hours.

- 20% overhead for fault free execution of Vcausal. (40% for pessimistic implementation).
- Crosspoint between Vcl and Vcausal at 0.006 faults per second (0.002 for the crosspoint between pessimistic and remote-checkpoint Vcl)

Conclusions

MPICH-V
MPI Implementation for Volatile resources

La complexité fractale des problèmes motive une augmentation exponentielle de la puissance requise

- Pour obtenir plus que le meilleur processeur disponible : parallélisme massif
- Tendance lourde : l'augmentation de perf. Individuelle des CPU atteint sa fin
- Implique la diminution du MTBF vers moins de quelques heures

Tolérance aux fautes pour les machines hautes performances

- Machines à passage de message : MPI
- Les machines hautes perf. ne sont pas internet : pseudosynchronisme
- Protocole par vue globale coordonnée (Chandy&Lamport)
- Enregistrement de messages
  - Pessimiste
  - Causal

Comparaison de performances entre les protocoles

- Avantage pour CL pour la performance hors faute
- Avantage pour Pessimiste en terme de résistance aux fautes
- Causal est un bon compromis

Travaux futurs
OpenMPI-V

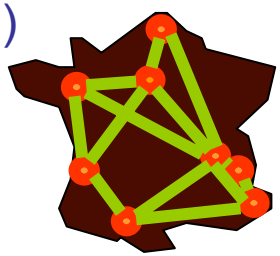
L'ordre des réceptions n'importe pas toujours

- Détection des évènements déterministes
- Collaboration avec les couches hautes de OpenMPI (opérations collectives) pour détecter les évènements non déterministes sémantiquement déterministes

La performance de CL dépend fort de l'enregistrement des points de reprise

- Décentralisation de l'enregistrement des points de reprise pour CL (Reed-Solomon + incremental ckpt)
- Investigation : problème de passage à l'échelle structurel de la vague de synchro CL ? (Grid5000)

Grid'5000



MPICH-V

MPI Implementation for Volatile resources

Grand Large

INRIA

Tolérance aux défaillances dans les systèmes hautes performances

Aurélien Bouteiller - Séminaire ENS – 8 février 2006

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

QUESTIONS ?