VMS V5.5

Overview de VMS FMR...

Bernard OURGHANLIAN

Symposium DECUS

4 Juin 1991

VMS 03

VMS V5.5

Sommaire

1. Introduction
2. Nouveau sous-système de *batch/print*
3. Nouveautés *cluster*
4. Support de POSIX
5. Support d'un "défragmenteur"
6. Support de DECdtm V1.1
7. Support des *hooks* DECnet PHASE V
8. Nouveautés RTL
9. DCL et divers

Remarque : certains sujets comme le nouveau *memory management* (qui a été déjà traité dans une réunion de SIG VAX) ne seront pas abordés ici, faute de temps....

Introduction

Les informations contenues dans cette présentation ne sauraient constituer un quelconque engagement de la part de DIGITAL sur des fonctionnalités ou des dates de disponibilité

Introduction

Un peu d'histoire... :

- La version actuelle de VMS est la version V5.4-2 :
  - -- N'introduit aucune nouvelle fonctionnalité ou correction de bugs par rapport à VMS V5.4-1a (à l'exception du kit spécial et optionnel LAT)
  - -- Version cumulative contenant TOUTES les nouvelles fonctionnalités et les corrections de problèmes contenues dans les versions V5.4-1 et V5.4-1a et installable sur les versions :
    - > V5.4 et V5.4-0A
    - > V5.4-1 et V5.4-1A

-- Introduit **uniquement** le support de nouveau *hardware* :
  - > VAX 4000 modèle 200
  - > L'interface KFMSA : *adapter* XMI/DSSI
  - > Disque DSSI : le RF31F
  - > Bandes : TF85, TZK10 et TA91
  - > CDROM : le RRD42
  - > Support d'autres disques non encore annoncés
-- Comprend un nouveau kit optionnel LAT corrigeant de nombreux problèmes présents en V5.4-1

- Les prochaines versions de VMS seront les V5.4-3 et V5.5 (VMS FMR)
- Ces 2 versions arriveront très probablement **en même temps**

- La version V5.4-3 :
  - -- 2 versions :
    - > version "complète" (*remastered kit*)
    - > version "update" cumulative
  - -- Version **complète** pour le support de nouveau *hardware* non encore annoncé
  - -- Version **cumulative** contenant TOUTES les nouvelles fonctionnalités et les corrections de problèmes contenues dans les versions V5.4-1 et V5.4-2 et installable sur les versions :
    - > V5.4 et V5.4-0A
    - > V5.4-1 et V5.4-1A
    - > V5.4-2
  - -- Correction de nombreux problèmes (Backup, *file system*, *driver*,...)

-- Introduit le support de nouveau *hardware*/*software* :

> L'interface DEMFA : *adapter* XMI/FDDI

> VAXft modèles 410, 610, 612

> Plusieurs *LAN adapters* par noeud dans un LAVc (voir plus loin)

> *Memory Management Autotuning* : permet aux *processes* actifs de réclamer la mémoire depuis les *idle processes* (voir la présentation sur ce sujet qui a été faite lors du dernier SIG VAX)

> Le *server* DECwindows V3.0 (utilise encore la version 2.1 pour la partie client)

> Le support de l'ISL (*Initial System Load*) depuis un InfoServer 100 pour les "petits" systèmes

> Le support de *hardware* non encore annoncé.....

- La version V5.5 est .....

Nouveau batch/print

L'ensemble du sous-système *batch/print* à été réécrit afin de répondre aux exigences suivantes :

- Améliorer la qualité et la disponibilité
- Améliorer les performances
- Permettre des améliorations futures

Nouveau batch/print

En V5.x (avant V5.5) :

- La base de données des *queues* était implantée sous la forme d'une base de données partagée entre tous les noeuds d'un *cluster*
- Ce fichier était accédé de façon aléatoire par les différents noeuds d'un *cluster*
- Un protocole de *locking* spécifique était utilisé pour coordonner l'accès à ce fichier depuis les différents noeuds

Il devenait urger de changer.....

car.....

Problèmes de performance :

- La charge de travail de *Job Control* augmente :
  - -- Beaucoup de temps CPU consommé par *Job Control* sur de nombreux sites
  - -- Entre 300-500 imprimantes sur certains sites; ce nombre augmente de 10 a 25% par trimestre
  - -- Soumission de 1000 jobs/heure sur certains sites
  - -- Augmentation du nombre de *queues* liées à la distribution de l'impression
- Les algorithmes de *Job Control* génèrent un nombre d'I/Os multiple du **(nombre de jobs en attente)**2**
- Le nombre d'I/Os nécessaire pour assurer la cohérence du cache de *Job Control* est un multiple du **nombre de noeuds dans le cluster**

- *Job Control* nécessite 30-50 mises à jour d'enregistrements dans le fichier JBCSYSQUE.DAT pour chaque *job*
- L'augmentation de la vitesse des disques ne suffit plus !!!

Améliorer la maintenabilité :

- Problèmes de corruptions du fichier JBCSYSQUE
- Boucles dans *Job Control*
- Problèmes de performances d'une base de données distribuée dans un LAVc

Nouveau batch/print

Comment changer ?

- Un seul *queue manager* accède à la base de données des *queues*
- Nouveau format de la base de données
- Séparation des fonctions du *Queue Manager* et du *Job Controller*

Nouveau batch/print

Un seul process "queue manager"

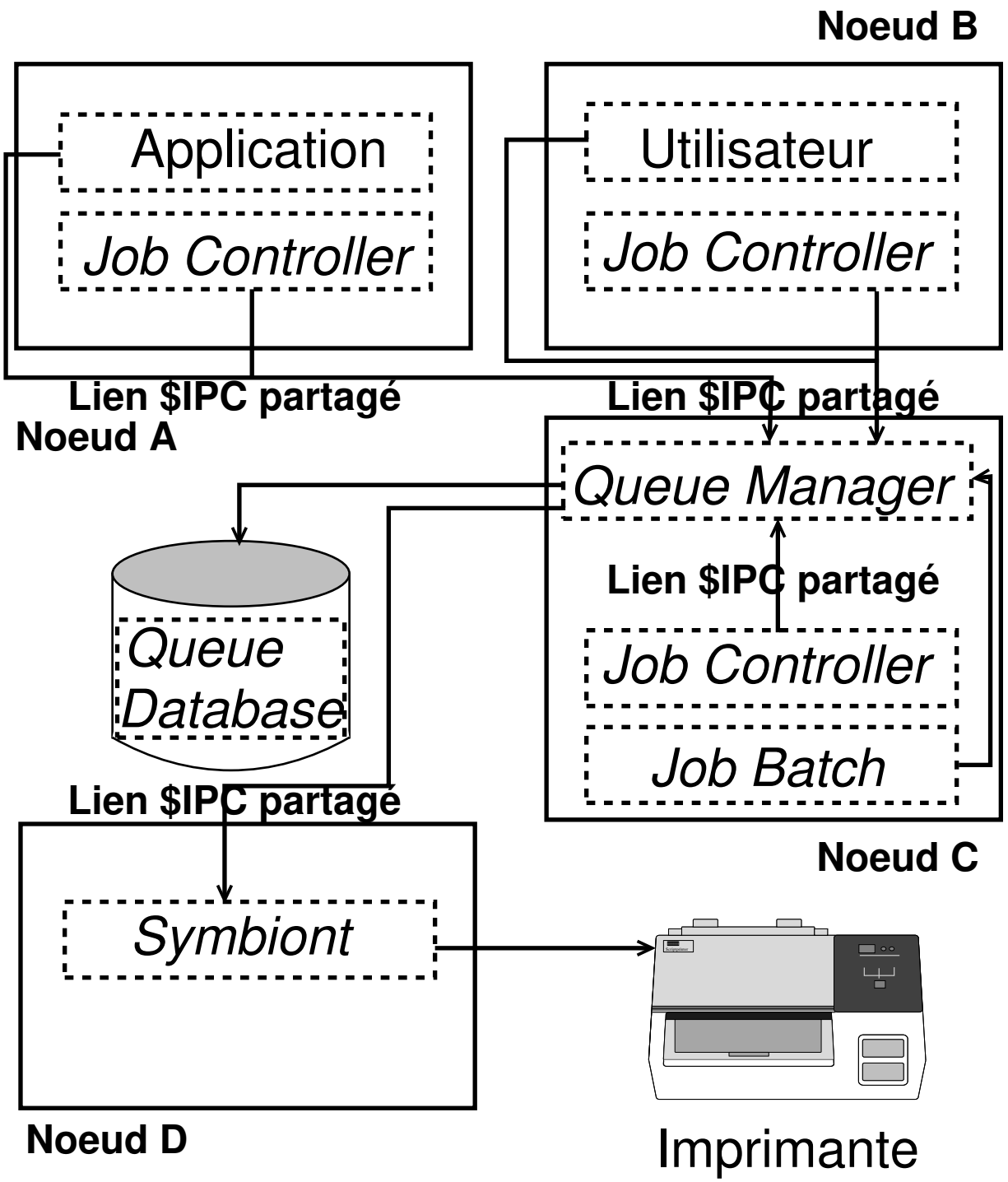
- Le *Queue Manager* est un *process* unique à un instant donné dans le *cluster* agissant comme un serveur *cluster-wide* :
  - -- Permet de réduire les activités disque nécessaires pour maintenir la cohérence du cache
  - -- Réduit la complexité du protocole de *locking*
- Les fonctions du *Queue Manager* sont séparées de celles du *Job Controller*

- Le *Job Controller* :
  - -- Effectue toutes les activités non directement liées au *Queue Manager*

- -- Crée et contrôle les *processes batch*, *symbiont* et le *Queue Manager* lui-même
- -- Traite la commande : $START/QUEUE/MANAGER
- -- Gère l'*accounting*
- -- Gère le *failover* du *Queue Manager* et s'assure que le *process Queue Manager* est toujours présent

- Les differentes composantes communiquent à travers $IPC :
  - -- Entre le *queue manager* et les *job controllers*
  - -- Entre le *queue manager* et les *symbionts*
- Les noeuds n'accedant pas à la base de données des queues permettent un *failover* automatique

Nouveau batch/print Un seul process "queue manager"



Nouveau batch/print
Nouvelle base de données

- Les informations de *queues*, *forms*, *characteristics* sont maintenues dans des fichiers RMS séquentiels indexés
  - -- Les *queues* ont des références aux *forms* et *characteristics*, pas des liens
- Les informations *jobs* et fichiers sont maintenues dans un fichier journal
  - -- L'espace dans le fichier journal est libéré lors d'opération de type *check-point*
- Toutes les informations sont maintenues en mémoire :
  - -- Les lectures sont seulement effectuées pendant le *startup*
  - -- Il n'y a pas d'I/O disque pour les requêtes $GETQUI

-- En général, il n'y a qu'une seule I/O d'écriture par requête $SNDJBC

-- Les indices qui changent fréquemment sont construits et modifiés en mémoire

-- Le *startup* du *Queue Manager* reconstruit tous les indices et peut donc corriger la plupart des formes de corruption

Base de données en mémoire :

- La plus grande partie de la base de données en mémoire est constituée de segments de mémoire :

  -- Ces segments sont pointés par référence à l'aide d'*ids* indépendants des adresses

  -- Une *queue*, un *job* ou une définition de *form* est une collection de ces segments mémoire avec un segment de base qui référence plusieurs autres segments

-- Pour trouver une *queue* par nom ou par numéro d'entrée, le *queue manager* a besoin d'une façon d'organiser ses données afin de trouver ces éléments :

> Pendant la phase de démarrage du *queue manager*, on construit un *balanced binary tree* permettant de trouver facilement ces données

> Ces arbres n'existent PAS sur disque : ils sont construits à partir de la lecture des informations lues depuis le disque

Bases de données disque :

- Le fichier JBCSYSQUE.DAT n'est plus utilisé : il est rempalcé par 3 fichiers sur disque :
  - -- Le *master file* qui est un fichier RMS séquentiel indexé à 2 clés contenant les informations relatives aux *forms*, *characteristics* et au *Queue Manager* lui-même. Il contient aussi le nom de toutes les *queues* définies sur le système
  - -- Le *queue file* qui est un fichier RMS séquentiel indexé à une clé contenant les définitions pour les *queues*
  - -- Le *journal file* qui est un fichier pré-alloué, séquentiel contenant toutes les définitions de *jobs* et les informations *cluster* dynamiques

- Fichier
  QMAN$MASTER.DAT :
  -- Fichier partagé
  -- Tous les *Jobs Controllers* et le *Queue Manager* ont ce fichier ouvert quand le *Queue Manager* est en fonctionnement
  -- Par défaut dans SYS$SYSTEM; pointé par le nom logique (/SYSTEM/EXEC) QMAN$MASTER
  -- Ce fichier contient :
  > La localisation des *queue* et *journal files*
  > La liste des noeuds autorisés à faire tourner le *Queue Manager*
  > Les définitions des *forms* et *characteristics*
  > La liste des noms de *queues*

-- Ce fichier ne change pas beaucoup et est destiné à permettre le support dans le futur de plusieurs *processes Queue Manager*, chacun d'entre-eux gèrant un ensemble différent de *queues* mais partageant les informations de *forms* et *characteristics*

- Fichier **SYS$QUEUE_MANAGER.QMAN$QUEUES** :
  -- Ce fichier n'est pas partagé et n'est lu que lors du *startup*
  -- Contient les définitions des *queues*
  -- Seul le *Queue Manager* peut écrire dans ce fichier
  -- Pendant le *startup*, ce fichier et le *master file* sont comparés pour être sûr que les deux fichiers sont d'accords sur les *queues* présentes sur le système : toute *queue* qui n'est pas trouvée dans les 2 fichiers est supprimée

- Fichier **SYS$QUEUE_MANAGER.QMAN$JOURNAL** :
  - -- Ce fichier journal contient :
    - > Un enregistrement de début liant l'ancien fichier journal au journal actif
    - > Des enregistrements de "fondation" contenant (entre autres) le *cluster foundation time*
    - > Des enregistrements de transactions contenant :
      - + Des segments mémoire
      - + Des enregistrements d'I/Os dans le *queue file*
      - + Des réponses aux requêtes $SNDJBC
    - > Des enregistrements de type *checkpoint*
    - > Un enregistrement de fin

-- Ce fichier est mis à jour en premier (avant le *queue file*) afin de garantir l'efficatité de la journalisation

-- Les définitions de *queues*, de *forms* et de *characteristics* ne sont pas journalisées

- Ces 2 derniers fichiers n'ont pas l'obligation de se trouver dans la même *directory* que le *master file*, mais ces 2 fichiers doivent se trouver dans la MÊME *directory* : cette *directory* est décrite lors de la commande :

  $START/QUEUE/MANAGER nom_de_directory

- *Checkpointing* : lors d'une opération de *checkpointing*, le *Queue Manager* :

  -- Renomme le fichier journal courant (_OLD)

  -- Crée un nouveau fichier pré-alloué

  -- Écrit 8 blocks de segments mémoire, entremêlé avec le traitement des requêtes

-- Écrit les informations liées aux liens $IPC et aux messages non envoyés

-- Supprime l'ancien fichier journal

- Pendant l'exécution du *startup*, le *Queue Manager* construit la base de données en mémoire en :

  -- Lisant toutes les données depuis le *queue file* et en chargeant ces données en mémoire

  -- Lisant le fichier journal :

  > Les segments mémoire sont chargés en mémoire

  > Toutes les I/Os enregistrées dans le fichier journal sont vérifiées par rapport à ce qui est en mémoire

  > Les reponses à $SNDJBC non acquittées sont réémises

  -- Vérifiant que toutes les références des segments mémoires sont valides

-- Reconstruisant les *binary trees*

-- Vérifiant que toutes les références à tous les objets sont valides

- Quand il démarre, le *queue manager* a besoin de savoir quels *processes* sont dans le *cluster*. Pour cela, il :

  -- Vérifie le *cluster foundation time* par rapport au *journal file*

  -- Supprime les informations liées aux *processes* pour les noeuds manquants

  -- Envoie la liste des PIDs actifs à chaque *Job Controller* en leur demandant d'effectuer une vérification

  -- Reçoit un message de suppression de *process* pour chaque PID non actif mais présent dans la liste des PIDs de l'un des *Job Controllers*

Nouveau batch/print

Les communications du Queue Manager

Ce qu'il faut savoir.... :

- Tous les *processes* sur un noeud partagent le même lien $IPC avec le *Queue Manager*
- Chaque message a un *identifier* unique (*quadword*)
- Chaque message contient le dernier *message id* reçu depuis l'autre coté du lien
- Le *Queue Manager* enregistre toutes les réponses dans le fichier journal (*shadow)*
- Chaque *process* maintient une *shadow copy* de toute requête $SNDJBC ou $GETQUI non traitée dans son espace P1

- Ces id's et ces *shadow copies* permettent d'éviter la perte de messages pendant le *failover* du *queue manager*

Démarrage de *process*, réconstruction des liens :

- Quand il démarre, le *Queue Manager* établit un lien avec tous les noeuds du *cluster*
- Il fournit ensuite les dernier *message id* traité pour chacun des noeuds
- Chaque noeud lui fournit en retour le dernier *message id* reçu
- Le *Queue Manager* renvoit toutes les réponses non reçues
- Chaque noeud renvoit chaque requête non traitée

Job Controllers **et** ***Symbionts*** :

- Le *Job Controller* et les *processes Symbiont* sont des *processes* spéciaux pour le *queue manager* mais ils utilisent encore $SNDJBC pour :
  - -- Envoyer les messages de demande de traitement au *Queue Manager*
  - -- Envoyer les messages au *Queue Manager* en réponse aux requêtes de traitement émanant du *Queue Manager*
- Le *Job Controller* contient toute la logique nécessaire pour :
  - -- Lire les information du *Queue Manager* depuis le *master file*
  - -- Déterminer si le *Queue Manager* doit être démarré
  - -- Déterminer quel est le meilleur choix (meilleur noeud) pour démarrer le *Queue Manager*

-- Vérifie si un START/QUEUE/MANAGER a besoin d'une réponse

-- Vérifie s'il est nécessaire d'effectuer un *failover*

Nouveau batch/print
Nouvelles fonctionnalités

- Changement dans le démarrage et l'arrêt du *Queue Manager*
- Nouvelle fonctionnalité AUTO_START
- Changement dans le *longword* QUI$_QUEUE_STATUS
- Nouvelles fonctionnalités utilisateur
- Réécriture de $SNDSMB et $SNDACC (utilisation de $SNDJBC)

Nouveau batch/print
Commandes Queue Manager

- START/QUEUE/MANAGER
  -- Est maintanant une commande *clusterwide*
  -- Est enregistrée dans la base de données du *queue manager* : n'a donc besoin d'être effectuée qu'une fois : le *queue manager* démarre automatiquement à chaque *reboot*
  -- /ON_NODE=(A,B,...) permet de spécifier une liste des noeuds impliqués dans les opérartions de *failover* (par défaut, tous les noeuds le sont)

- STOP/QUEUE/MANAGER/CLUSTER
  - -- Arrête le *clusterwide queue manager*
  - -- Les transitions *cluster* ne changent pas l'état du *Queue Manager* : il faut le redémarrer
- STOP/QUEUE/ON_NODE[=nom_de_noeud]
  - -- Remplace la commande STOP/QUEUE/MANAGER
  - -- Arrête toutes les queues sur un noeud
  - -- Rend STOP/QUEUE/MANAGER obsolète
  - -- Force un éventuel *failover* des *queues* quand c'est nécessaire

Nouveau batch/print
Fonctionnalité AUTO_START

- But :
  - -- Simplifier le *startup*
  - -- Procurer un mécanisme permettant une haute disponibilité pour les imprimantes réseau
- ENABLE AUTO_START [/QUEUES][/ON_NODE=nom de noeud]
  - -- Permet de démarrer la fonctionnalité AUTO_START sur un noeud
  - -- Notifie le *Queue Manager* du fait que le noeud a progressé assez dans sa procédure de *startup* pour permettre à ses *jobs batch* et *print* de démarrer
  - -- Démarre toutes les *queues* en mode AUTO_START sur le noeud concerné

- DISABLE AUTO_START [/QUEUES][/ON_NODE=nom de noeud]
  - -- Marque toutes les *queues* de type AUTO_START en "stop_pending" pour préparer un *shutdown* planifié : ceci laisse continuer les *jobs* en cours d'exécution
  - -- Empêche les *queues* en mode AUTO_START d'effectuer un *failover* vers le noeud concerné
  - -- Force un *failover* éventuel lors de la complétude des *jobs* s'exécutant sur l'une des *queues* en mode AUTO_START
  - -- Après cette commande, il faut encore arrêter toutes les *queues* qui ne sont pas en mode AUTO_START

- START/QUEUE /AUTOSTART_ON=(node::device,...)
  - -- Paramètre une queue pour être automatiquement démarrée quand le mode AUTO_START est mis en service sur un noeud
  - -- Donne une liste de noeuds vers lesquels effectuer un *failover*
- STOP/QUEUE/NEXT ou /RESET
  - -- Arrête une queue et marque cette queue "auto-start inactive"; dans ce cas, celle-ci ne sera pas démarrée automatiquement

Nouveau batch/print
QUI$_QUEUE_STATUS

- Ce *status longword* a toujours contenu une combinaison d'information et de *status*. Un problème existait jusqu'à présent pour tester avec $GETQUI si une *queue* était arrêtée (tester si stopped=1 ne suffisait pas, il fallait aussi tester idle=1 et stopping=0)
- Avec le nouveau *batch/print* certains de ces bits ont été définis comme bits d'état : ainsi, pour tester si une *queue* est arrêtée, il suffit maintenant de tester si stopped=1
- Mais, par exemple, la condition stopped=1 ET idlle =1 ne sera jamais vraie car les bits sont devenus des bits d'état : il faudra dans certains cas changer les programmes

Nouveau batch/print

Nouvelles fonctionnalités utilisateur

- Modifications de SHOW ENTRY :
  - -- Accepte les noms de *jobs* (*wildcards* possibles)
  - -- Modification du format de *display* : le numéro d'entrée apparait en pre-mière colonne
- PRINT, SUBMIT ou SET ENTRY admettent /RETAIN=(always,default,error,until=)
  - -- Permet à l'utilisateur de préciser le critère de rétention par *job* et non plus uniquement par queue
- SUBMIT/NOTE="..." (255 caractères)
  - -- Permet de spécifier un message pour impression sur la *flag page*

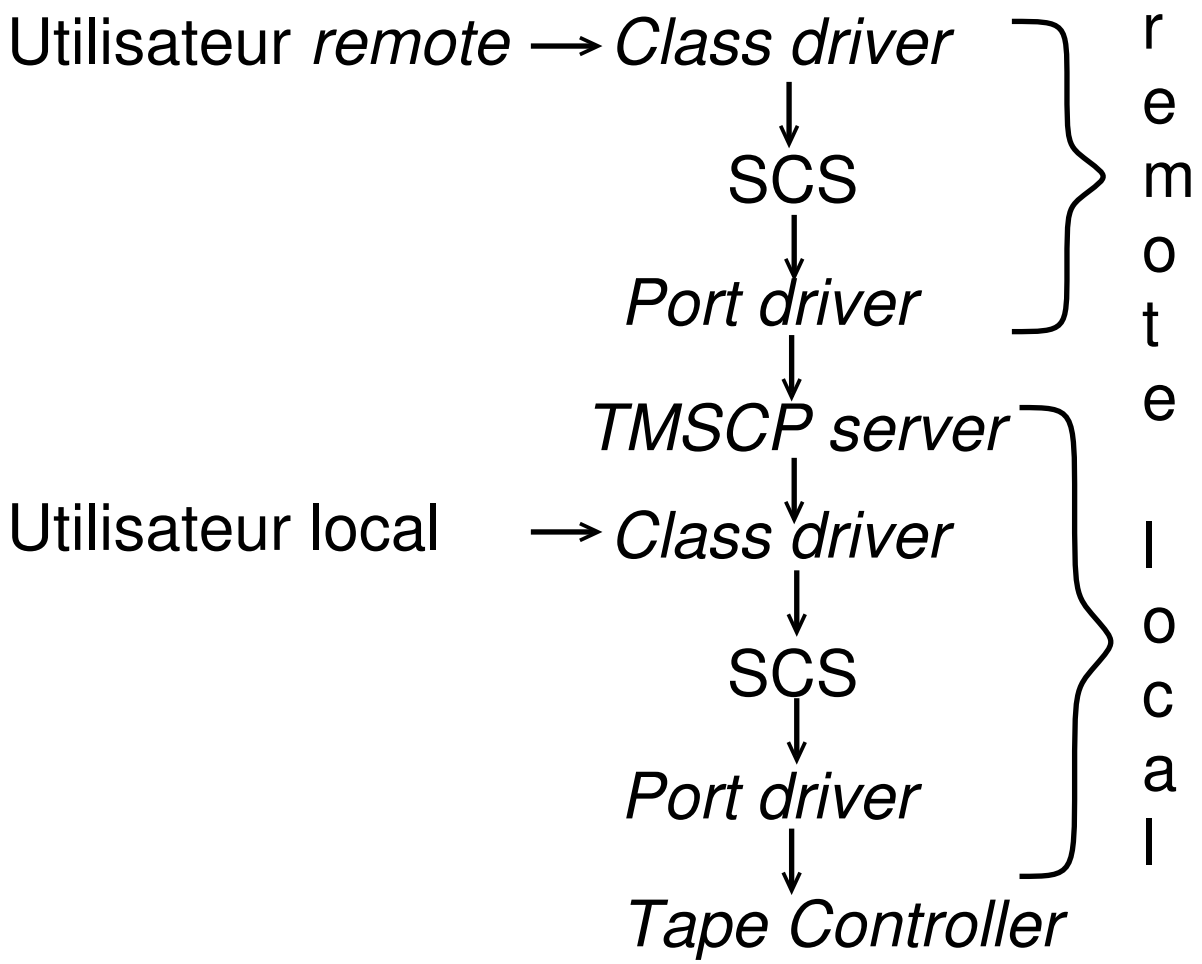
-- Peut servir de description de *job batch* (visualisé dans un SHOW QUEUE/FULL)

Nouveautés cluster
TMSCP server

- L'image TMSCP.EXE implante un *Tape MSCP Server*
- Permet de servir des bandes locales vers des systèmes *remote*
  - -- Supporte, par exemple, l'accès depuis un satellite à des bandes connectées derrière un HSC dans un *cluster* mixte
  - -- Permet l'accès *clusterwide* à un *controller* local
- Permet à un système VAX/VMS d'émuler un *controller* TMSCP
- **Les bandes non TMSCP NE PEUVENT PAS être servies** (SCSI ou TK50 sur VS2000 par exemple)

- Implanté sous la forme d'une SYSAP SCS (MSCP$TAPE)
- Contrôlé par un nouveau paramètre SYSGEN TMSCP_LOAD :
  - -- 0 (défaut) : ne pas charger le *server* et ne pas servir les bandes locales
  - -- 1 : charger le *server* et servir toutes les bandes locales TMSCP
- Le *server* est chargé au moment du *boot*
- Les bandes sont paramètrées "servies" pendant le *boot*
- Il n'y a pas de possibilité de choisir les bandes que l'on veut servir
- Il est cependant possible d'utiliser la commande $SET DEVICE/SERVED pour servir une bande qui n'était pas disponible pendant le *boot*

- Le flux d'une requête d'I/O *remote* :
  - -- La requête de $QIO est envoyée en direction du TU *tape class driver*
  - -- TUDRIVER construit un *packet* et l'envoit au *port driver* local
  - -- Le *port driver* envoit ce *packet* au noeud TMSCP *server* en utilisant SCS
  - -- Le TMSCP *server remote* l'envoit au TU *class driver*
  - -- Le *class driver remote* l'envoit à son *port driver* en utilisant SCS
  - -- Le *port driver* l'envoit à son *tape controller*



Dans le cas de la communication *TMSCP server* --> *Class driver* on n'utilise pas les *FDT routines*. Par contre lors d'une I/O "standard" on utilise les points d'entrées "standard" du *class driver*

Nouveautés cluster
Shadowing SCSI

- La version V5.4-1 a apporté le supporte du *host based shadowing* (*shadowing Phase II*) pour les disques DSA
- La version V5.5 étend le *shadowing Phase II* pour supporter le *shadowing* des disques SCSI
- Les fonctionnalités du *shadowing* pour les disques SCSI sont les mêmes que celles pour les disques DSA (à l'exception du support de *command assist*)
- Les disques SCSI supportés sont ceux étant capables de comprendre les 2 commandes SCSI :
  -- READ LONG
  -- WRITE LONG

- Ces 2 commandes ne font pas partie de la norme ANSI SCSI-1 (X3.131-1986) mais font partie du *draft* 10b de la norme ANSI SCSI-2
- Ces commandes permettre de lire ou écrire un secteur de données disque avec un ECC (sur 6 bytes) et sont utilisées par le *class driver* SCSI et le *shadowing drivers* pour gérer la possibilité d'écriture de *forced error flag* :
  -- Lors de UNIT_INIT, le *class driver* SCSI teste la possibilité d'envoyer de telles commandes au disque (*routine* TEST_HBS)
  -- En cas d'échec (test avec des tailles de transferts de 0,1, 512 et 1024 bytes), le bit DEV$V_NOFE est positionné indiquant que le disque n'est pas supporté dans un environnement *shadowing*

- Tous les disques DIGITAL supportent ces commandes mais cela n'est pas obligatoirement le cas pour les disques compatibles; ceci peut être vérifié sous SDA :

```
SOS4$DKA300                          RZ22              UCB address:  8036B390

Device status:    00000010 online
Characteristics: 1C454008 dir,fod,shr,avl,elg,idv,odv,rnd
                 05000221 clu,mscp,nnm,scsi,nofe  ===> NOFE !!!

Owner UIC [000000,000000]   Operation count     110760  ORB address  8036B550
      PID         00000000   Error count              0  DDB address  80875600
Alloc. lock ID    0C000000   Reference count          0  DDT address  803683D8
Alloc. class             4   Online count             0  VCB address  8038DA30
Class/Type           01/30   Retry cnt/max        16/16  CRB address  80875580
Def. buf. size         512   BOFF                 0000  I/O wait queue  empty
DEVDEPEND         03080821   Byte count           0000
DEVDEPND2         00000000   SVAPTE           00000000
FLCK index              34   DEVSTS               0004
DLCK address      00000000
```

- Si l'on tente de monter un disque SCSI en *shadow set* et que le disque ne le supporte pas, l'opération échoue et un message d'information est envoyé :

  $mount/sys dsa100/shad=$4$DKA300 test
  %MOUNT-I-DEVNOFE, device does not support FORCED ERROR handling.

- Il est possible de monter quand même un disque SCSI non supporté avec la commande /override=no_forced_error :

  $mount/sys/over=no_ forced_error dsa100 -
  _$/shad=$4$DKA300 test
  %MOUNT-I-DEVNOFE, device does not support FORCED ERROR handling.

- Un disque SCSI mounté avec un /OVER=No_Forced_Error est supprimé du *shadow set* pendant une opération de *full copy* si ce disque est destination de la *full copy* et que l'on rencontre un *bad block* sur le disque source
- Quand un *bad block* est rencontré :
  - -- Si la donnée peut être récupérée, le *class driver* SCSI "revectorise" le *block* et retourne un statut de succès
  - -- Si la donnée ne peut pas être récupérée, une erreur de parité est retournée (pas un *forced error*: cela n'est pas supporté par SCSI) par le

class driver SCSI et un bit d'erreur et positionné dans l'UCB; le fait qu'il n'y ait pas de notion de *forced error* sur un disque SCSI a une conséquence importante : on ne peut pas "revectoriser" les *blocks* ayant rencontré une *uncorrectable error* lors d'une **lecture**

-- Comme on ne sait pas gérer la notion de *forced error flag* sur un disque SCSI, celle-ci est "simulée" en générant une *uncorrectable ECC error* de telle façon que toute tentative de lecture ultérieure vers ce *block* se traduira par une erreur

-- Cette *ECC error* est générée en effectuant un READ LONG, en **inversant** les *bytes* de ce *block* et en effectuant un WRITE LONG pour réécrire ce *block*

-- L'opération de "revectorisation" n'est effectuée que lorsqu'une écriture est

effectuée sur le *block* et le *class driver* ”revectorise” d’abord puis écrit ensuite la donnée (opération non atomique vu du *shadowing*)

-- Pour se protéger contre une corruption potentielle, le disque ayant le *"bad block"* est marqué *full copy target* avant la "revectorisation" et repositionné *member* après la "revectorisation"

-- Un rappel important : le *shadowing* permet d’améliorer la disponibilité des données, pas leur intégrité

Nouveautés cluster

Divers shadowing

- Le *shadowing driver* (SHDRIVER) a été modifié de façon à retourner dans la partie *device dependant* de l'IOSB, le nombre de membres courants d'un *host based volume shadow set*
- Permet une notification immédiate pour les applications du fait qu'un membre a été supprimé du *shadow set*
- Allure de l'IOSB :

BCNT	I/O status
#membres	R1

- Formalisation du paramètre VMSD3 sous la forme du paramètre SYSGEN

SHADOW_MBR_TMO qui permet de contrôler le temps d'attente pendant lequel *Volume Shadowing Phase II* attend un membre perdu du *shadow set*

-- Dynamique, défaut 5 secondes

-- Si un VAX perd son *path* vers un membre d'un *shadow set* , le *shadowing phase II* fait passer les I/Os en *stall* et tente de trouver un autre *path.* Si le VAX trouve un autre *path* vers le membre perdu ET que ce membre n'est pas le seul membre valide du *shadow set,* le *shadowing Phase II* supprime ce membre du *shadow set* et redémarre les I/Os

-- Si l'on tourne en mode compatible, le *shadowing Phase I* necessite que le paramètre HSC **drive_timeout** soit au moins positionné à 5 secondes : cette valeur est passée aux disques *online* du HSC pour déterminer combien de temps ceux-ci doivent atten-

dre avant de se déclarer *available* au controleur sur l'autre port

-- Le *shadowing phase II* n'est pas concerné par le paramètre **drive_timeout** du HSC; cependant si *shawowing Phase I et II* coexistent sur le même HSC, il est préférable de positionner SHADOW_MBR_TMO à la même valeur

-- Pour positionner SHADOW_MBR_TMO à une valeur différente de la valeur par défaut, le bit de poids fort doit être positionné et le *byte* de poids faible doit contenir le nombre de secondes à attendre

-- Exemple : 10s,
SHADOW_MBR_TMO =%X8000000A

Nouveautés cluster

Support de plusieurs adapters dans un LAVc

- Un LAVc avec plusieurs *adapters* permet :
  - -- Une plus grande disponibilité
  - -- De meilleures performances
- Avec le support de plusieurs *controllers* Ethernet, VMS V5.5 (en fait V5.4-3) introduit le support de :
  - -- Jusqu'à 4 *controllers* Ethernet (ou FDDI à travers le DEMFA)
  - -- *Clusters* à travers des *LAN bridges*
  - -- *Routines* spécifiques permettant de démarrer ou arrêter l'utilisation des *adapters* par LAVc
  - -- D'outils de *debug*

Nouveautés cluster

Support de plusieurs adapters dans un LAVc

Guide de configuration :

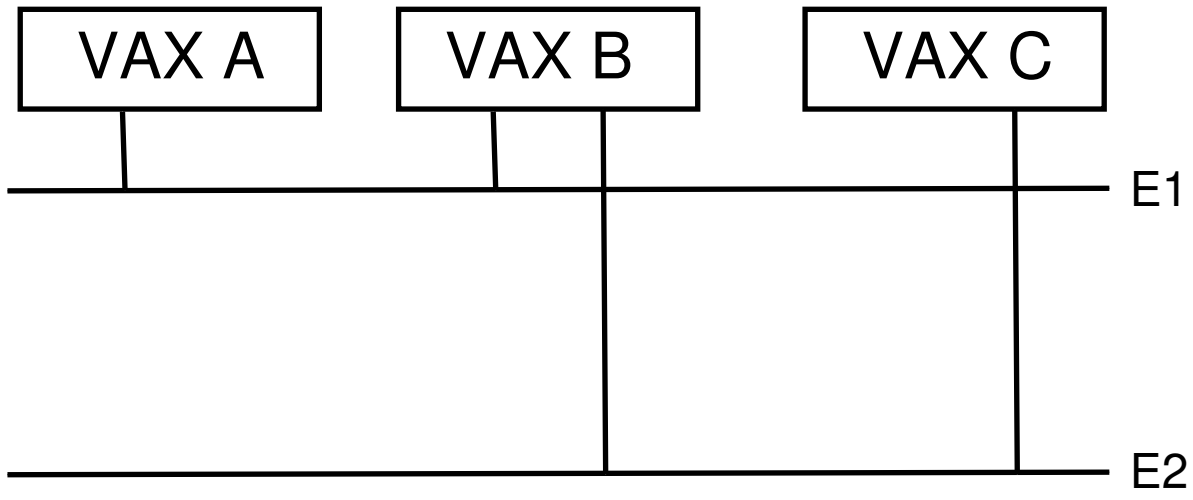
Quand on désire configurer un LAVc pour permettre le support d'*adapters* multiples :

- Tous les noeuds doivent tourner au moins VMS V5.4-3
- Le support initial permis est de 4 *LAN adapters* (peut évoluer dans le temps en fonction des tests)
- Chaque *LAN adapter* doit être connecté à un LAN **différent** (meilleure disponibilité et meilleur débit)
- Ces segments de LAN peuvent être "bridgés"; ceci n'est pas obligatoire mais peut permettre de meilleures

performances et une meilleure disponibilité

- Ces segments peuvent être Ethernet ou FDDI
- Il faut distribuer les *LAN adapters* des satellites de façon équitable sur ces segments
- Si l'on désire une configuration hautement redondante, il faut :
  -- "Bridger" les segments de LAN ensemble afin de former un seul LAN étendu afin de pouvoir tolérer la perte d'un "bridge"
  -- Configurer les *LAN bridges* de façon à faire passer les protocoles SCA et MOP

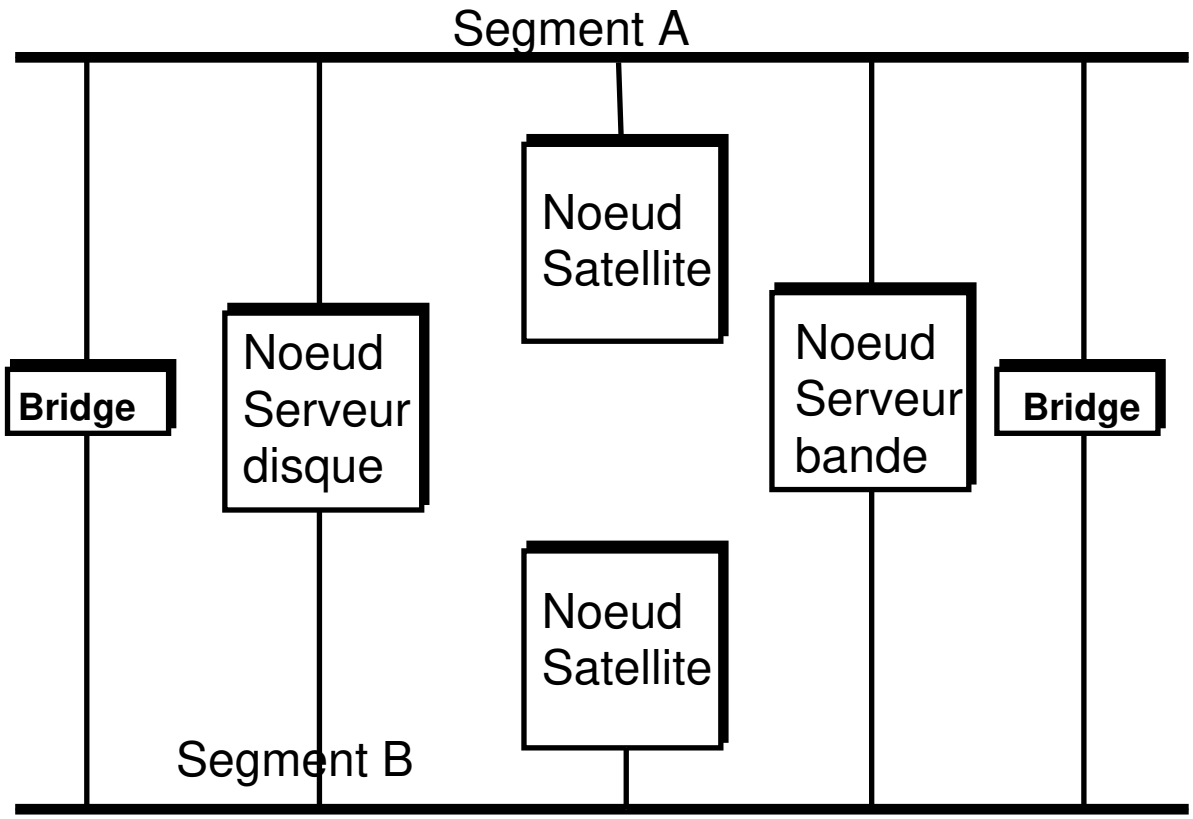
- Un configuration telle que celle-ci n'est pas supportée :



car les VAXes A et C ne peuvent pas communiquer à travers Ethernet et que le protocole SCS n'autorise pas la notion de *routing* (ici à travers le VAX B). Pour modifier la configuration afin d'être supporté, il faut soit :

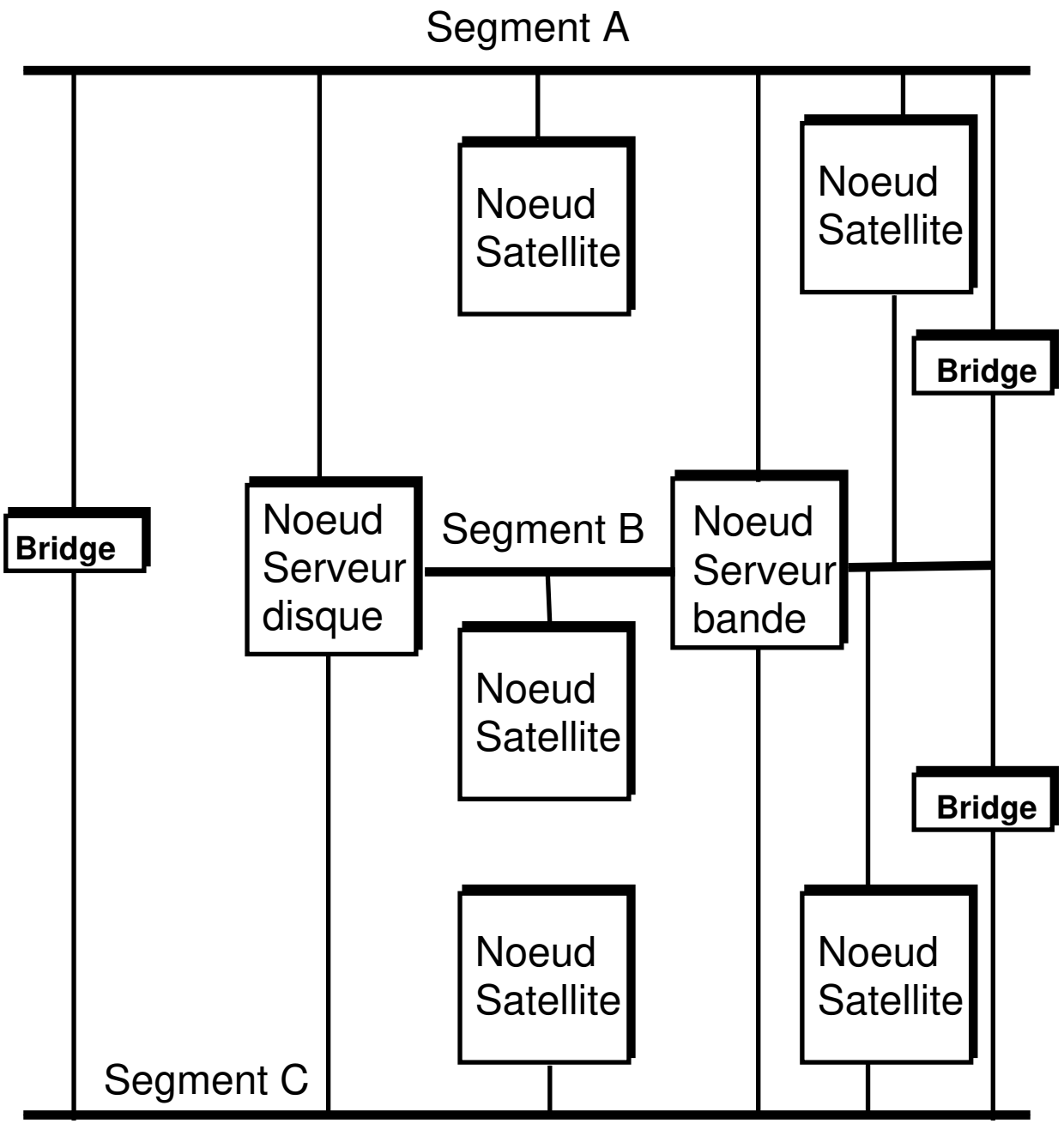
1. mettre un *bridge* entre E1 et E2
2. ajouter des connexions au VAX A vers E2 et au VAX C vers E1

- Exemple de configuration à 2 segments:



- Les noeuds critiques tels que les serveurs disques ou bande ou même les satellites critiques sont connectés aux 2 segments
- Il est possible pour les noeuds critiques d'être des *routers* (afin de pouvoir utiliser la notion de *cluster alias)* mais dans ce cas, il ne faut **pas** démarrer DECnet sur les *2 adapters*

- Les satellites non critiques sont distribués de façon équitable pour équilibrer les charges sur les différents segments



- Paramètrage des *LAN bridges* permettant le support du *failover* :
  - -- Paramètre du *Root bridge* :
    - > LISTEN_TIME (défaut 15s)
    - > FORWARDING_DELAY (défaut 30s)
  - -- Temps de *self-test* du *bridge*
  - -- Si le *cluster* veut survivre à un *failover* entre 2 LAN bridges, la valeur du paramètre RECNXINTERVAL doit être supérieure au temps de *failover* du bridge =

    temps de *self-test* du *bridge* + LISTEN_TIME + FORWARDING_DELAY
- Le *LAN bridge* reçoit un *HELLO message* des autres *bridges* dans un intervalle spécifié par le paramètre du *bridge* HELLO_INTERVAL. Si le *LAN bridge* ne reçoit pas de *HELLO messages* dans les LISTEN_TIME secondes, il change la topologie du réseau et peut choisir un

autre *root bridge*; après ce changement de topologie, le *bridge* attend FORWARDING_DELAY secondes afin d'apprendre quelles adresses sont de quel coté puis commence à transmettre les *packets*

- Diminuer la valeur de LISTEN_TIME permet au *bridge* de détecter un changement de topologie plus rapidement. Dans ce cas , il faut aussi diminuer la valeur de HELLO_INTERVAL; dans ce cas, les *HELLO messages* sont envoyés plus souvent et le traffic sur le réseau augmente
- Diminuer la valeur de FORWARDING_DELAY peut conduire le bridge à transmettre des packets inutilement de l'autre coté du segment : ceci peut créer temporairement un traffic inutile sur les 2 segments avant que le *bridge* n'ait déterminé de quel coté sont les adresses

Nouveautés cluster

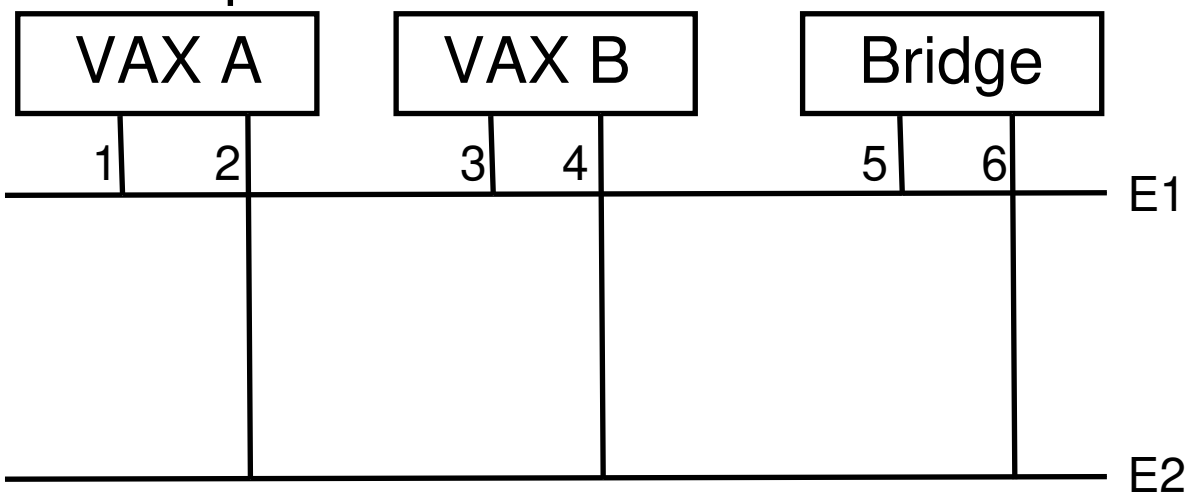
Plusieurs adapters dans un LAVc : Fonctionnement

Définition des termes :

- **BUS** : le nom utilisé par PEDRIVER pour un *LAN adapter*. La structure de données BUS contient les données utilisées par PEDRIVER pour gèrer, contrôler et surveiller un *LAN adapter.* Chaque structure de données BUS représente un *LAN adapter* différent
- **CHANNEL** : le nom utilisé par PEDRIVER pour désigner le chemin unique entre 2 noeuds dans un *cluster*. Un *channel* est représenté par une paire d'adresses LAN, de *LAN adapter* ou de structure de données BUS
- **PEDRIVER** : le *driver* qui permet les connexions *cluster* possibles sur Ether-

net ou FDDI. Ce *driver* est responsable pour fournir à SCS un chemin de données sans erreur entre 2 noeuds. Ce chemin est aussi appelé circuit virtuel

- **SCS** : *System Communication Services*: couche logicielle responsable des communications dans le *cluster*. Gère les connexions entre les noeuds dans le *cluster*
- Exemple :



VAX A a 4 *channels* vers VAX B :

1. *Channel* 1-3 : E1 seulement
2. *Channel* 1-4 : E1 (1 à 5), à travers le *bridge* E2 (6 à 4)
3. *Channel* 2-3 : E1 (2 à 6), à travers le *bridge* E2 (5 à 2)
4. *Channel* 2-4 : E2 seulement

Fonctionnement :

- PEDRIVER fournit à la couche SCS un seul *circuit virtuel* pour chaque système *remote* dans le *cluster*
- SCS choisit le *circuit virtuel* correspondant au noeud *remote* quand il envoit des données
- PEDRIVER récupère alors le *channel* depuis le *circuit virtuel* et l'utilise pour localiser le BUS (*LAN adapter)*
- Les données sont alors envoyées en utilisant ce *LAN adapter* vers le noeud *remote*
- Plusieurs *LAN adapters* peuvent être utilisés par PEDRIVER. Un nouveau *channel* est créé pour chaque paire de *LAN adapters* entre le noeud local et chacun des noeuds *remote*
- Ce *channel* reste ouvert tant que tous les composants du chemin ouvert entre

les 2 éléments de cette paire sont opérationnels

- Le *circuit virtuel*, lui, reste ouvert (utilisable par SCS) tant **qu'au moins un *channel*** est ouvert
- Chaque BUS envoit des *HELLO messages* toutes les 3 secondes : un *channel* est ouvert après qu'un *HELLO message* soit reçu d'un noeud *remote*
- Si un *channel* n'a **pas** reçu de message depuis le noeud *remote* depuis 8 secondes, il est considéré qu'il y a un problème et le *channel* est fermé : les *channels* qui dont fermés de cette façon sont regroupés ensemble pour permettre une analyse ultérieure quand le *LAVc network failure analysis* a été activé (voir plus loin)
- Il est important de noter que tant qu'un *channel* existe encore, le *circuit virtuel*

reste ouvert : **les SYSAPs SCS ne sont pas prévenues du problème**

- L'amélioration des performances dans le cas de l'utilisation de plusieurs *LAN adapters* est obtenue par une distribution de la charge sur les *LAN adapters* avec détection de congestion : ceci n'est pas du *load balancing* car **à un instant donné, un seul *LAN adapter* est utilisé pour communiquer avec une machine donnée** :
- Détection de congestion :
  -- Dans les message périodiques de type *HELLO messages,* un *timestamp* est placé. Quand ce message est reçu, on soustrait le temps contenu dans le message du temps de réception : ceci permet de calculer une approximation du temps de transmission. Ce temps de transmission est moyenné dans une moyenne glissante pour chacun des *channels*

et l'on compare ces différents temps (ce qui permet de ne plus tenir compte des différences des horloges des systèmes)

-- Quand le réseau est *idle*, les temps de transmission sont presque identiques; ce n'est pas le cas si l'un des réseau est chargé

-- On selectionne alors le *channel* le moins chargé que l'on garde sélectionné jusqu'à ce que la situation change

Nouveautés cluster

Plusieurs adapters dans un LAVc : le boot

- Lors du *boot* l'ensemble des *LAN adapters* est configuré (même s'il y en a plus de 4) en utilisant la routine de PEDRIVER LAN$GET_DEVICE
- Les satellites peuvent maintenant *booter* en utilisant les *LAN adapters* supplémentaires (>>> B XQB0)
- Notion de "*Solicit Booting*" :
  - -- Cette fonctionnalité a été introduite avec VMS V5.4
  - -- Le fonctionnement est le suivant :
    - > la requête de téléchargement MOP est transmise au *boot mode* sous la forme d'une requête *multi-cast*

> la satellite est chargé avec NISCS-LOAD.EXE **et** le *system disk ID* ($GETDVI avec DVI$_ALLDEVNAM) par NISCS_LAA.EXE

> le serveur du disque système est localisé :

+ le satellite envoit des requêtes d'accès au disque système à l'aide de messages *multicast*

+ l'un des *disk server* répond en donnant l'accès au disque système

> le satellite continue à *booter*

-- Le système qui sert le disque système peut être différent de celui qui a effectué le téléchargement

-- Les *dumps* sont écrits sur le même disque. Dans le cas du *host based shadowing*, les *dumps* sont écrits sur le *master member*

-- Si le *disk server* du disque système *crashe* pendant le *dump* du satellite, celui-ci trouve un autre *server* et continue à écrire son *dump*

- Les restrictions :

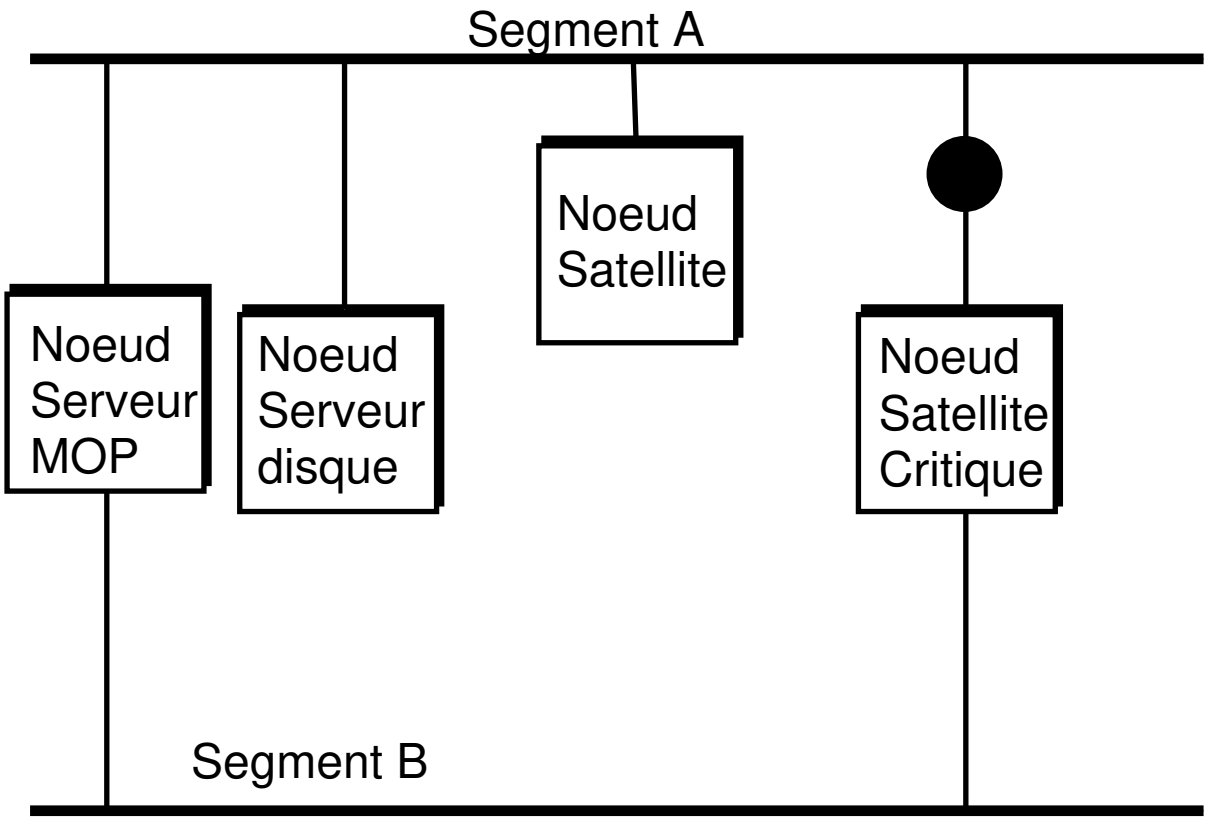
Restriction 1 :

Pas de support de *failover* d'*adapter* pendant le *boot* :

- Le *boot/dump driver* des satellites n'affectue pas de *failover* vers un autre *adapter* même si celui-ci est présent
- Un *Halt* est effectué dans ce cas
- Si *Halt/Restart* est en service, le satellite tente de *rebooter*; il est toujours possible de *rebooter* manuellement sur l'autre *adapter*
- En cas de *dump*, celui-ci est perdu : ceci n'est pas fréquent

- **Restriction 2 :**
- **Restriction de configurations pour *boot* et *disk server*** :
- Il est possible de créer des configurations dans lesquelles le *system disk server* est accessible pour le satellite à travers un *LAN adapter* **différent** de celui du *MOP server* : cette configuration n'est pas supportée et ne fonctionne pas
- En effet, le *boot driver* du satellite ne supporte par un *failover* vers un autre *LAN adapter*
- Donc, **le *disk server* du disque système doit être accessible sur le même *extended LAN* que le *boot server***
- Attention : si le *disk server* a un problème et que le **nouveau** *disk server* n'est accessible qu'à travers un *adapter* différent, le *dump* echouera

- Exemple :



Si l'*adapter* permettant l'accès au segment A est coupé, les satellites critiques ne peuvent plus *booter/dumper*
Solutions :
1. Ajouter un *bridge* entre les LANs A et B
2. Ajouter un *adapter* sur le *disk server* vers le segment B

Restriction 3 :

DECnet phase IV ne supporte qu'une adresse hardware par satellite pour le téléchargement

- Contournement possible :
  - -- Créer plusieurs noeuds DECnet qui pointent vers la même *root* mais avec des adresses *hardware* différentes; ces descriptions supplémentaires ne seront utilisées que pendant le *boot* des satellites
  - -- Maintenir des bases de données DECnet différentes sur les *boot servers*
  - -- Définir le même noeud dans plusieurs *clusters* en utilisant plusieurs adresses réseau : *booter* depuis plusieurs *LAN adapters* permettra de placer le satellite dans des *clusters* différents

Nouveautés cluster

Plusieurs adapters dans un LAVc : Divers

- On utilise maintenant les adresses Ethernet *hardware*; on n'utilise plus les adresses DECnet Phase IV
- PEDRIVER maintient un cache des messages reçus afin d'éviter l'arrivée de messages dans le désordre : ceci évite des retransmissions inutiles
- L'interface non documentée FFI (*Fast Fast Interface*) utilisée par PEDRIVER pour appeler les *drivers* Ethernet, n'est plus utilisée : elle est remplacée par l'interface (utilisée par DECnet Phase V) VCI (*VAX communication Interface*)

- Le support de DECnet PHASE IV pose un problème car DECnet PHASE IV ne supporte qu'un *adapter* ethernet par machine et par *extended LAN*; or il est possible ici d'utiliser plusieurs *adapter* ethernet et donc tous les *adapters* ne peuvent être initialisés en utilisant l'adresse DECnet. La solution utilisée est la suivante :
  - -- Les *datalink drivers* PHASE V permettent aux clients de changer l'adresse *hardware* lors du *startup*; de plus, un paramètre est disponible permettant les autres clients de changer cette adresse. Enfin, les clients qui s'exécutent sont prévenus lors du changement
  - -- On utilise donc les *datalink drivers* PHASE V :
    - > PEDRIVER initialise donc le *LAN adapter* avec l'adresse *hardware*

> DECnet est alors autorisé à changer l'adresse et notifie PEDRIVER

> Les *HELLO messages* en provenance de l'*adapter* provoquent la création de nouveaux *channels*

> Le fait de ne plus recevoir de *HELLO messages* en provenance de la vieille adresse de l'*adapter* provoque un *timeout* sur le *channel* mais **pas** sur le *circuit virtuel*

> Le basculement de l'adresse est alors terminé

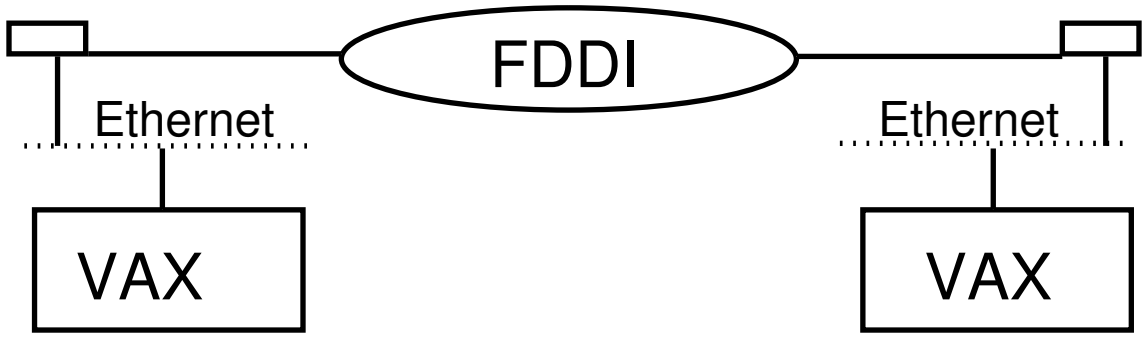
Nouveautés cluster

LAVc : Le support de FDDI

- FDDI supporte des *packets* de taille plus importante qu'Ethernet : 4468 *bytes* au lieu de 1498 *bytes*
- PEDRIVER a la possibilié d'utiliser cette taille de *packet* si le paramètre LRPSIZE > 4474
- Mais, seuls les noeuds connectés à FDDI sur le **même anneau** peuvent bénéficier de cette taille de *packet*
- Il est important de noter que PEDRIVER n'effecute pas d'assemblage de *packets* : il est donc inutile d'augmenter la taille de LRPSIZE si les transferts sont de petite taille
- Dans le cas où l'on utilise des configurations *cluster* avec des liens physiques multiples CI ou DSSI, ces configurations

ne peuvent pas tirer parti des tailles de *packet* plus grandes car le *path* CI ou DSSI est choisi de préférence au *path* FDDI : dans ces cas , il est inutile d'augmenter la taille de LRPSIZE

- Si l'on utilise une configuration dans laquelle un *packet* transite à travers Ethernet :



on ne peut pas utiliser des grandes tailles de *packets* : PEDRIVER détecte cela en testant la valeur du champ "priorité" dans le *frame control byte* du *header* FDDI : les *bridges* vendus par DEC positionnent cette valeur à 0; or PEDRIVER envoit tous les messages FDDI avec une priorité différente de 0 :

cela permet de détecter qu'un *packet* est passé par Ethernet

- Si un *bridge* ne positionne pas correctement la valeur à 0, PEDRIVER envoit une message d'erreur sur la console et casse la connexion avec les autres noeuds FDDI : dans ce cas, il faut positionner LRPSIZE à 1504 pour éviter l'utilisation de grande taille de *packet*

Nouveautés cluster
Les outils de *debug*

Contrôle du système de boot :

- Dans les versions précédentes de VMS, un système n'indiquait pas de messages de connexions quand il accédait au disque système pendant une phase de *boot remote* dans un LAVc
- À partir de VMS V5.4-3, il est possible d'obtenir ces messages en :
  - -- Positionnant NISCS_CONV_BOOT à 1 pour autoriser un *boot* conversationnel *remote*
  - -- Positionnant le bit 0 de R5 à 1
- Dans le cas où l'on a plusieurs *LAN adapters*, cela permet de déterminer quel est le système *remote* qui sert le disque système

Analyse des problèmes réseau à l'aide de PEDRIVER :

- Avec le support de plusieurs *LAN adapters*, on augmente la tolérance du *cluster* aux problèmes réseau mais on augmente aussi la probabilité d'avoir un problème sur l'un des composants de ce réseau
- Pour maintenir un niveau de disponibilité correct, un certain nombre de *hooks* ont été mis en place dans le code de PEDRIVER pour isoler ces problèmes réseau
- C'est en fait la seule façon de s'appercevoir que l'on a eu un problème sur un *channel* : il n'y a que lorsqu'il y a rupture du *circuit virtuel* que l'on est informé du problème

- Cette analyse d'erreur n'est pas simple :
  -- Le nombre de *channels* qui existe entre 2 noeuds peut être calculé en multipliant le nombre de *LAN adapters* sur un noeud par celui qui existe sur l'autre noeud
  -- Pour un LAVc de 96 noeuds et 4 *LAN adapters* par noeud, cela fait 96*4*4=1536 *channels*
  -- Si l'on a un problème réseau réduisant le nombre de *channel* de 2/3, cela fait encore 512....
  -- Cela n'est pas très simple de trouver le problème
- Il est donc possible de décrire physiquement la topologie de son réseau dans un programme fournit dans SYS$EXAMPLES:LAVC$FAILURE_ANALYSIS.MAR en décrivant :

- -- les composants du réseau (segments, *bridges*, *LAN adapters,* concentrateurs,*...)*
- -- la topologie du réseau (les chemins)

- L'ensemble de cette description est chargée en *non paged pool* : il faut donc, si l'on a un réseau important réserver l'exécution de ce programme sur les noeuds ayant suffisamment de mémoire
- Quand ce programme s'exécute,il :
  - -- collecte les problèmes *channel*
  - -- groupe les problèmes
  - -- met en correspondance ces problèmes avec la topologie du réseau
  - -- imprime les composants de la liste ayant une probabilité élevée d'être en cause dans les problèmes de *channels*

-- Si une portion de chemin ne peut pas être validée, le premier composant de ce chemin est considéré comme le "suspect principal" et un message OPCOM :

%LAVC-W-PSUSPECT, nom de composant

est envoyé à la console

-- Les autres composants de ce chemin sont considérés comme "suspect secondaires" et le message les concernant est :

%LAVC-W-ASUSPECT, nom de composant

Interface de programmation permettant de démarrer ou d'arrêter la protocole LAVc sur un *LAN adapter* :

- Il est possible d'utiliser 2 routines permettant de démarrer ou d'arrêter le protocole SCS sur un *LAN adapter* :
  - -- SYS$LAVC_START_BUS qui admet comme paramètre le nom de l'adapter passé par *reference*
  - -- SYS$LAVC_STOP_BUS qui admet comme paramètre le nom de l'adapter passé par *reference*
- Exemple simple :

```
PROGRAM start_bus
EXTERNAL Sys$LAVc_Start_Bus
INTEGER*4 Sys$LAVc_Start_Bus
INTEGER*4 status
status = Sys$LAVc_Start_Bus('XQA0:')
CALL Sys$Exit(%val(status))
END
```

- 2 exemples : LAVC$START_BUS.MAR et LAVC$STOP_BUS.MAR sont fournis dans SYS$EXAMPLES
- De plus, les routines SYS$LAVC_ENABLE_ANALYSIS et SYS$LAVC_DISABLE_ANALYSIS permettent de mettre en service ou d'arrêter les mécanismes d'analyse d'erreur cités plus haut
- La documentation de ces routines avec des exemples est donnée dans les *release notes* de VMS V5.4-3

Modification de SDA pour les *LAN adapters* multiples dans un LAVc :

- La commande SHOW PORT admet les *qualifiers* suivants :
  - -- /BUS[=BUS_id]
  - -- /CHANNEL[=CHANNEL_id]
  - -- /DEVICE
  - -- /MESSAGE
  - -- /VC[=VC_id]
- Exemples :

```
SDA> show port /bus=ESA
 VAXcluster data structures
 --------------------------
 --- BUS: 80B08090  (ESA)  Device: ES_LANCE  LAN Address: AA-00-04-00-33-FD ---
                                              LAN Hardware Address:
08-00-2B-12-AE-A1
 Status: 00000A03 run,online,xmt_chaining_disabled,restart
 ------- Transmit ------  ------- Receive -------  ---- Structure Addresses ---
 Msg Xmt          434107  Msg Rcv         1170090  PORT Address         80B091B0
 Mcast Msgs       103939  Mcast Msgs       859601  VCIB Addr            80B08248
 Mcast Bytes    13304192  Mcast Bytes    96272072  HELLO Message Addr   80B082D8
 Bytes Xmt      59789962  Bytes Rcv     146674695  BYE Message Addr     80B08468
 Outstand I/Os         0  Buffer Size        1424  Delete BUS Rtn Adr   8079E424
 Xmt Errors           75  Rcv Ring Size         8
 Last Xmt Error 00000334         Time of Last Xmt Error 25-MAR-1991 23:39:28.27
 --- Receive Errors ----  ------ BUS Timer ------  ----- Datalink Events ------
 TR Mcast Rcv          0  Handshake TMO  8079FA50  Last 22-MAR-1991 18:25:25.12
 Rcv Bad SCSID         0  Listen TMO     8079FA54  Last Event           00001202
 Rcv Short Msg         0  HELLO timer           1  Port Usable                 1
 Fail CH Alloc         0  HELLO Xmt err        38  Port Unusable               0
 Fail VC Alloc         0                         Address Change              1
 Wrong PORT            0                         Port Restart Fail           0
```

```
SDA> show port /vc=vc_SOS4
VAXcluster data structures
--------------------------
               --- Virtual Circuit (VC) 806CD6E0 ---
Remote System Name:  SOS4   (0:VAX)     Remote SCSSYSTEMID:  64856
Local System ID:  222 (DE)              Status: 0005 open,path
------ Transmit -------  ------ VC Closures ----  ---- Congestion Control ----
Msg Xmt          216686  SeqMsg TMO             0  UnAcked Msgs                1
Unsequence            3  CC DFQ Empty           0  Pipe Quota Reached         33
Sequence         149643  Topology Change        0  CMD Queue Len               0
ReXmt               545  NPAGEDYN Low           0  Max CMD Queue Len           5
Lone ACK          66495                             RSVP Threshold             15
Bytes Xmt       33309074                             Pipe Quota                 31
------- Receive -------  - Messages  Discarded -  ----- Channel Selection ----
Msg Rcv          194492  No Xmt Chan            0  Preferred Channel   80704320
Unsequence            1  Rcv Short Msg          0  Delay Time          FB7E6F80
Sequence         178905  Illegal Seq Msg        0  Buffer Size             1424
ReRcv                30  Bad Checksum           0  Channel Count              6
Lone ACK          15531  TR DFQ Empty           0  Channel Selections      3920
Cache                26  TR MFQ Empty           0  Protocol                1.3.0
Ill ACK               0  CC MFQ Empty           0  Open  1-JAN-1991 00:00:07.03
Bytes Rcv        52086897  Cache Miss             0  Cls  17-NOV-1858 00:00:00.00

VAXcluster data structures
--------------------------
         -- Channel Summary for Virtual Circuit (SOS4  ) 806CD6E0 --

Address     Type     Xmt Time Size Preferred    Best        Last State Change
--------  --------- -------- ---- ---------  --------   ----------------------
80704320  Preferred FB7E6F80 1424        812       617   22-MAR-1991 18:14:07.01
807043E0  Active    FB7E735E 1424         95         4   25-MAR-1991 20:01:15.18
807050D0  Active    FB7E7FED 1424        431         0   25-MAR-1991 20:01:15.18
806CD820  Active    FB7E728E 1424        868      1470   25-MAR-1991 20:01:15.18
80705010  Active    FB7E7043 1424        738         9   25-MAR-1991 20:00:58.17
806CD8E0  Active    FB7E7BB5 1424        976      1744   25-MAR-1991 20:00:31.17

SDA> show port /ch=ch_SOS4
VAXcluster data structures
--------------------------
  -- PEDRIVER Channel (CH:80704320) for Virtual Circuit (VC:806CD6E0) SOS4   --
State: 0004 open                Status: 0B path,open,rmt_hwa_valid
BUS: 80B008B0  (XQA)  Lcl Device: XQ_DELQA  Lcl LAN Address: 08-00-2B-0A-6A-6B
Rmt Name: XQB         Rmt Device: XQ_DEQTA  Rmt LAN Address: 08-00-2B-13-70-88
Rmt Seq #: 0002   Open:22-MAR-1991 18:14:07.01  Closed:17-NOV-1858 00:00:00.00
------- Transmit ------  ------- Receive -------  ----- Channel Selection ----
Lcl CH Seq #        0001  Msg Rcv          139205  Average Xmt Time    FB879740
Msg Xmt            66707  Mcast Msgs       103906  Remote Buffer Size      1424
Ctrl Msgs              1  Mcast Bytes    10182788  Max Buffer Size         1424
Ctrl Bytes            98  Ctrl Msgs             2  Best Channel             615
```

```
Bytes Xmt          9130385  Ctrl Bytes           196  Preferred Channel        810
Rmt Ring Size           31  Bytes Rcv       22654333  Retransmit Penalty         2
---------------  Channel Errors  ---------------  Xmt Error Penalty          12
Handshake TMO           0  Short CC Msgs          0  ------- Channel Timer ------
Listen TMO              0  Incompat Chan          0  Timer Entry Flink   8079FA3C
Bad Authorize           0  No MSCP Srvr           0               Blink   80705010
Bad ECO                 0  Disk Not Srvd          0  Last Ring Index            08
Bad Multicast           0  Old TR Msgs            0  Protocol                 1.3.0
Topology Change         0                            Supported Services  00000000

VAXcluster data structures
--------------------------

   -- Network Component List (CLST:80D36440) for Channel (CH:806DC420) --

COMP adr   COMP Type  Description
--------   ---------
------------------------------------------------------------
80D30010   NODE       SOS4
80CC9300   ADAPTER    ESA; Ethernet adapter on my station
80D3CDB0   COMPONENT  TIMA LAN - Cluster Segment DEMPR
80D40380   COMPONENT  TIMA LAN - Cluster Segment DELNI
80D36AD0   COMPONENT  LAN-WAN
80D2D4C0 P COMPONENT  DELNI to LAB
80CC2BE0 S ADAPTER    ESA; Ethernet adapter on node RONE37
80D323F0   NODE       RONE37
```

Nouveautés cluster

Nouveautés lock manager

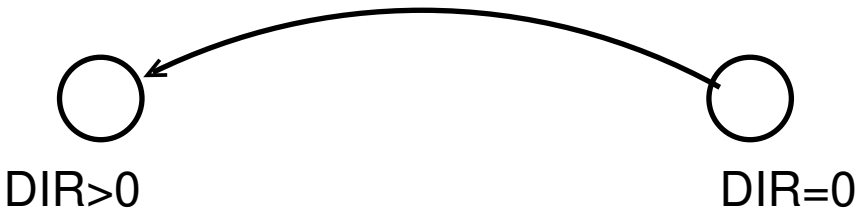
Dynamic Remastering :

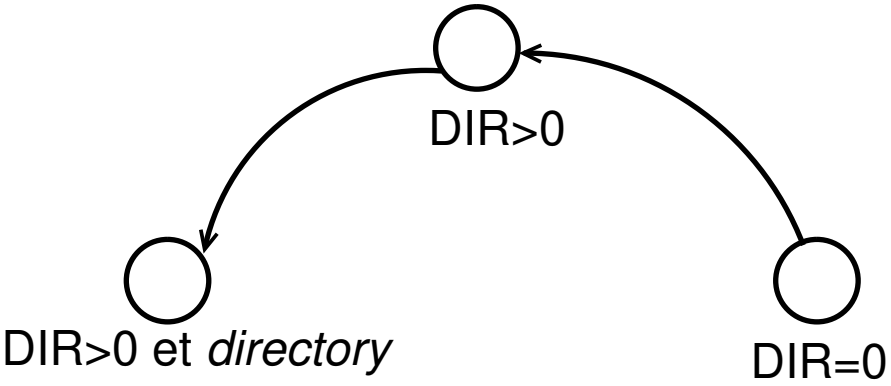
- La fonction de *dynamic remastering* du *distributed lock manager* est constituée de 2 parties principales :
  - -- Un mécanisme pour déplacer les ressources pendant les opérations normales
  - -- Une règle pour déplacer ces ressources en fonction de l'*activity*
- VMS V5.2 a pour la première fois introduit une notion de *dynamic remastering* limitée fondée sur la valeur du paramètre LOCKDIRWT :
  - -- *Activity* = 0, si LOCKDIRWT = 0

-- *Activity* = 1, si LOCKDIRWR > 0 et que le noeud **n'est pas** le noeud *directory* de la ressource

-- *Activity* = 2, si LOCKDIRWR > 0 et que le noeud **est** le noeud *directory* de la ressource

- En V5.2 et V5.4, l'*activity* n'est pas réellement fondée sur les activités de *locking*
- On a les déplacements suivants possibles de la fonction de *master* :





- La version V5.5 introduit les améliorations suivantes :
  - -- Le noeud *remote* devient *master* si c'est le seul noeud ayant des *locks* sur la ressource concernée
  - -- Le nouvel algorithme de *remastering* est fondé sur l'activité de *locking* réelle
- Parcours rapide de l'algorithme :
  - -- Chaque noeud effectue les taches suivantes :
    - > *monitoring* des requêtes de *locks*
    - > stockage du nombre total de requêtes par intervalle dans un champ du *ressource block* (RSB$W_NACT)
    - > le total est augmenté à chaque fois que :
      - + un nouveau *lock* est demandé

+ un *lock* existant est converti
+ un *lock* est *dequeued*

-- Périodiquement, le contenu de RSB$W_NACT est intégré dans un compteur de moyenne historique RSB$W_OACT

-- La valeur de l'*activity* représente l'historique des requêtes récentes de *lock* :

> permet une décroissance graduelle si l'activité tombe à 0

> permet un croissance graduelle si l'activité augmente

-- Un des buts du *design* est d'eviter les montés ou les descentes TROP brusques

- La procédure de *remastering* est une procédure en 2 étapes qui est de la responsabilité du *master* courant :

-- 1. Déterminer le nouveau *master* :

> Lors d'une requête de *lock*, on reçoit la valeur *remote* de RSB$W_OACT

> Si :

+ la valeur *remote* de LOCKDIRWT est >= à celle trouvée sur le noeud *master*

+ la valeur d'*activity remote* est > *system threshold*

+ la valeur d'*activity remote* est > *activity* locale + *activity threshold*

> Alors :

+ le noeud *remote* peut être un meilleurs noeud *master*

+ on positionne alors le bit RM_PEND

-- 2. Déterminer les bénéfices potentiels de performance :

> on cherche à calculer les bénéfices d'un déplacement de *mastership*

> la liste des ressources à déplacer est parcourue

> Si RM_PEND est positionné :

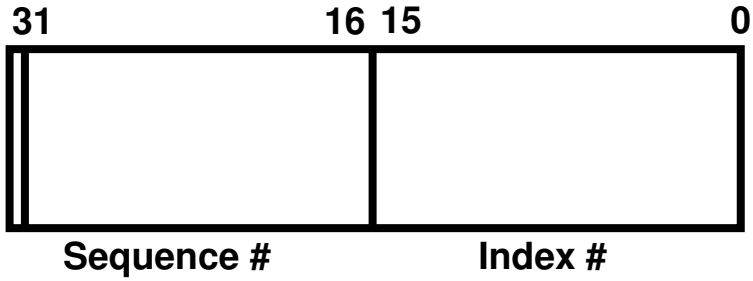
+ on détermine le cout du déplacement de l'arbre des *locks* en fonction du nombre total de *resource locks*

+ on détermine les bénéfices potentiels si l'arbre était déplacé

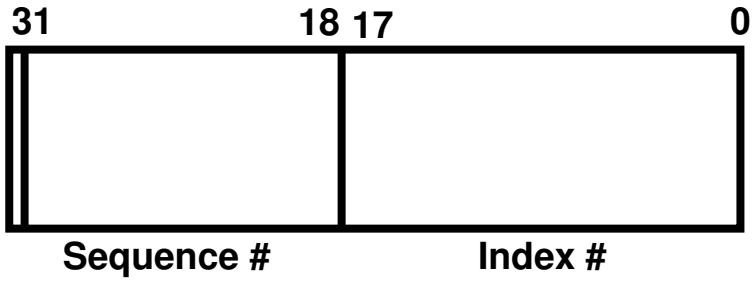
+ si le bénéfice est > au coût, la ressource est *remastered* sur le noeud *remote*

Extension de la table des LKIDs :

- Avant V5.2-1, la table de Lock IDs avait le format suivant :

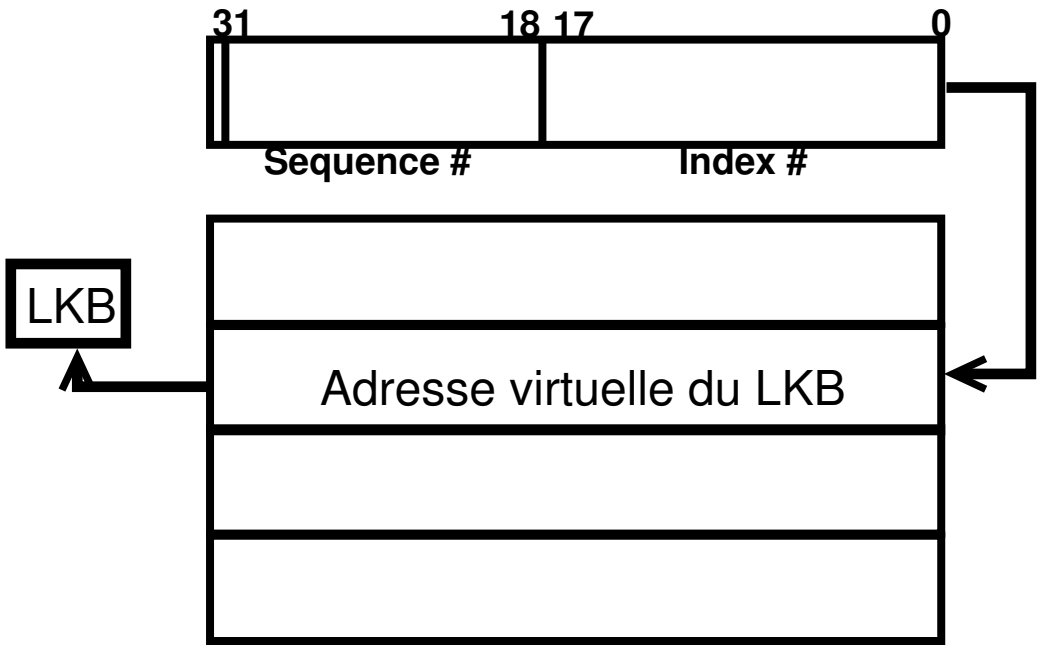


- Avec V5.2-1, la table de Lock IDs est devenue :



permettant de faire passer le nombre maximum de Lock IDs de 65535 à 262144

- Aujourd'hui, la table de Lock IDs pointe sur l'adresse du LKB :



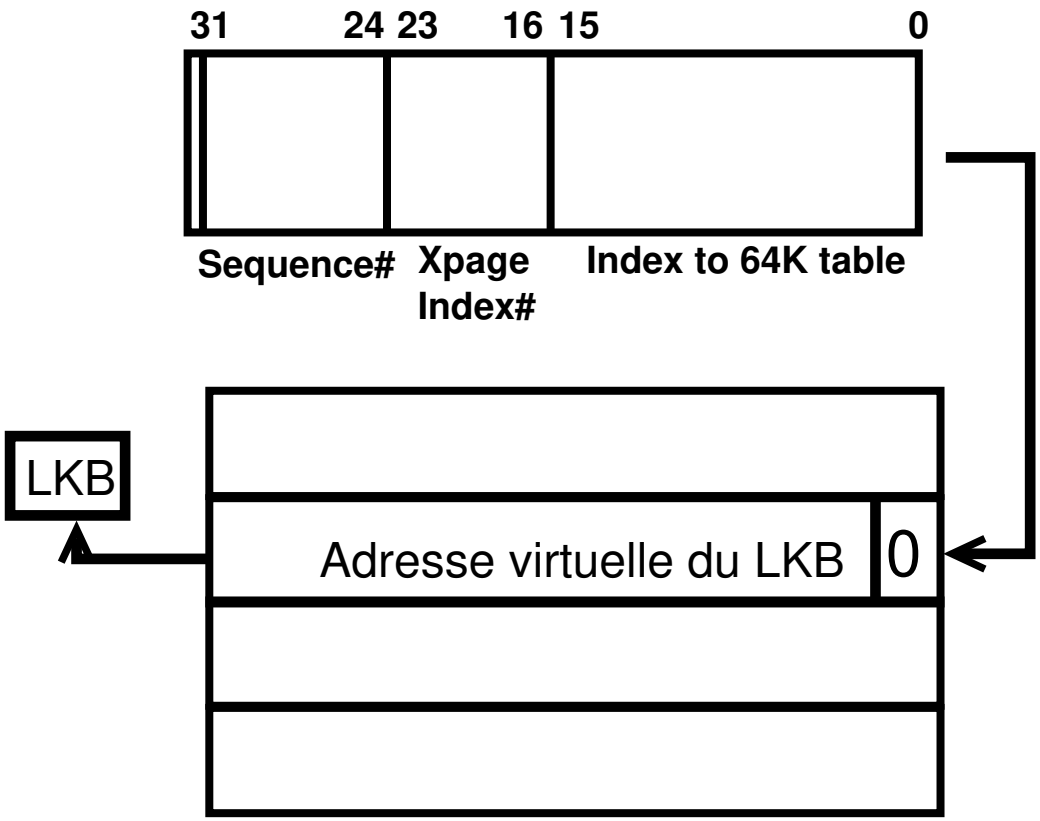
- Aujourd'hui la table est initialisée pendant la phase de *boot* et l'algorithme d'extension :
  - -- Alloue (taille actuelle + valeur de LOCKIDTBL) depuis la *non paged pool* : lors de la première extension, la taille de la table devient 2*LOCKIDTBL
  - -- Copie l'ancienne table dans la nouvelle table

-- Désalloue l'ancienne table

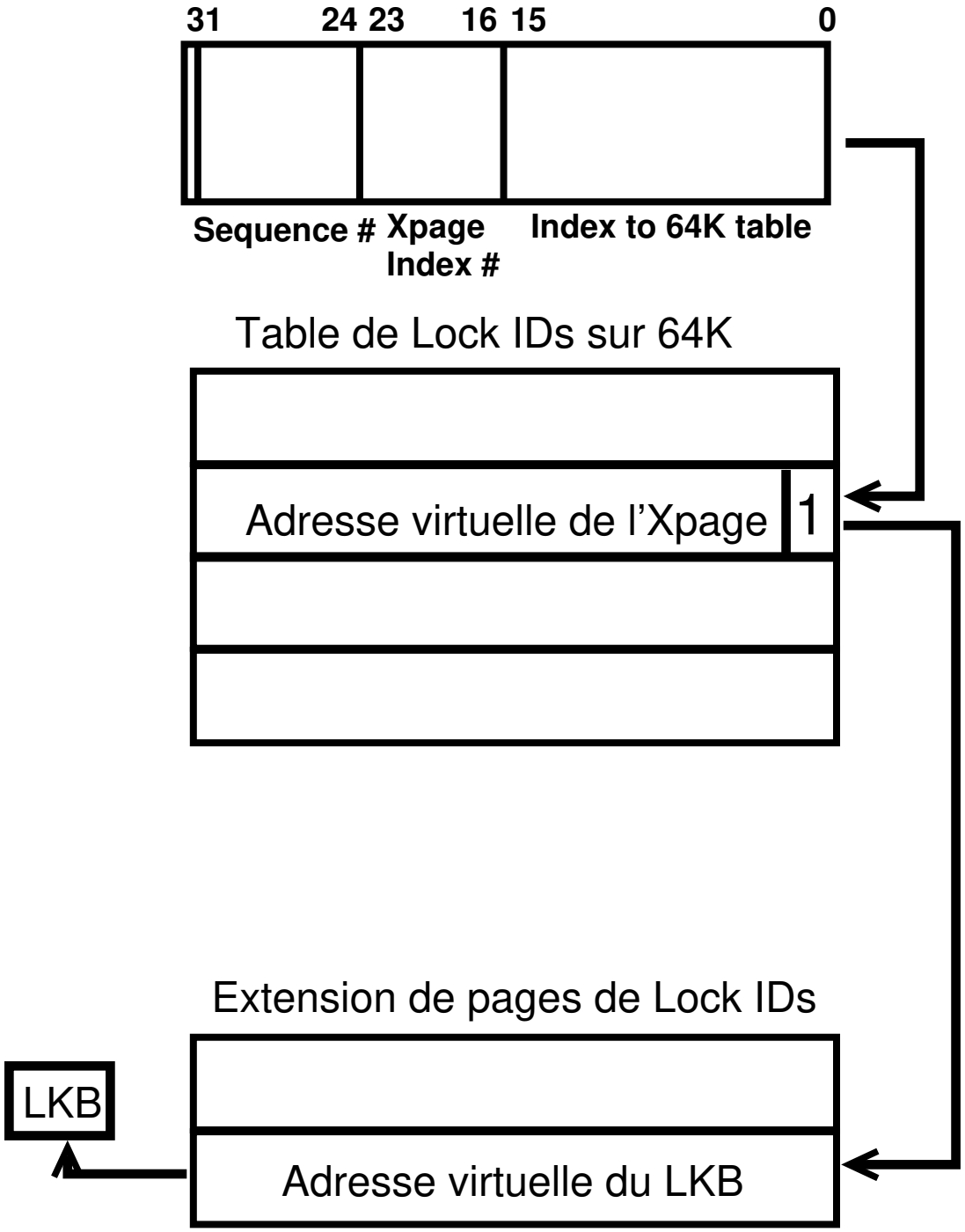
-- Initialise le reste de la nouvelle table

- Ceci ne suffit pas car le *design* actuel a des limitations :
  -- on dépend de la possibilité de trouver de gros *blocks* **contigus** de *non paged pool*
  -- le nombre de *Lock IDs* est limité
  -- il y a un coût élevé à payer lors de l'extension de la table des *Lock IDs*
  -- il a des limitations de performances en SMP

- Le format des lock IDs devient si le nombre de LKIDs est < 64K :



- Au dela de 64K, le format de Lock IDs devient :



- Nouvel algorithme d'initialisation et d'extension :
  - -- Allocation de SPTEs pour la table de 64K lors du *boot* (4 pages)
  - -- Allocation de la mémoire pour initialiser la table (initialisation avec les LKIDs jusquà LOCKIDTBL + 256 pour la gestion des *sequence numbers*)
  - -- Extension de la table 1 page par 1 page jusqu'à 64K (l'extension intervient quand le nombre de LKIDs libres tombe en dessous de 256)
  - -- Extension de la table 2 pages contigues par 2 pages après 64K

Support du *byte range locking* :

- Permet le support de la fonctionnalité *byte-range locking* POSIX
- Permet l'implantation des services systèmes POSIX :
  - -- fcntl() avec les flags F_GETLK, F_SETLK et F_SETLKW
- Permet l'utilisation de $ENQ [range] ou "*range*" est l'adresse d'un RGB (*Range Block*)
- Un *range block* est une structure de données formée de 2 *quadwords* :

- Aucun changement dans le comportement du reste du code du *lock manager*: le chemin de code exécuté est différent

- Il est possible d'effectuer une opération de *range conversion* :
  - -- Exemple : convertir le *lock* sur les bytes 10-20 (EX) en *lock* sur les bytes 30-40 (PW)
  - -- Nécessite l'implantation d'un protocole de type *2 phase commit*
  - -- Permet la conversion simultanée de :
    - > *range*
    - > *mode*

Nouveautés cluster

Amélioration des temps de "failover"

- Introduction du paramètre SYSGEN TIMVCFAIL permettant de spécifier le temps de détection maximum d'un problème de port ou de vircuit virtuel SCS
- La valeur minimum autorisée de TIMVCFAIL est de 1 seconde

Support de POSIX

Un certain nombre de *hooks* et de nouvelles fonctionnalités ont été ajoutés à VMS pour le support de POSIX :

- 1003.1 : *base operating system services*
- 1003.2 : *command line interface* (*shell*)
- 1003.4 : *realtime extension*

Hooks :

- EVENT_FLAGS_AND_ASTS.EXE pour supporter les *signals* POSIX
- IO_ROUTINE.EXE pour supporter un $ASSIGN *session-wide*
- PAGE_MANAGEMENT.EXE pour supporter les nécessités de vérrouillage mémoire 1003.4

- PROCESS_MANAGEMENT.EXE pour le support de fork(), de la notion de *session* POSIX et les mécanismes de *scheduling* 1003.4
- WORKING_SET_MANAGEMENT.EXE pour le support de fork() et le support des nécessités de vérrouillage mémoire 1003.4
- SYSTEM_PRIMITIVES.EXE : support de l'*accounting* POSIX

Selective cancel I/O on channel :

- Création du nouveau service système SYS$CANCEL_SELECTIVE afin de permettre à l'appelant d'effectuer un *cancel I/O* sur une liste de requêtes adressées à un périphériques
- On doit :
  - -- spécifier un *channel*
  - -- identifier l'IOSB comme étant celui qui a été spécifié lors de l'I/O

- L'opération *cancel* est effectuée au niveau *driver*
- Le service système s'assure que le périphérique est capable d'effectuer cette opération. Dans ce cas, le périphérique a :
  - -- un *extended DDT* contenant à l'adresse DDT$CANCEL_SELECTIVE l'adresse de la routine effectuant l'opération
  - -- un bit DPT$V_EXTENDED_DDT positionné dans DPT$L_FLAGS
- Utilisé effentiellement pour le *terminal driver* POSIX

Extention du *Mailbox driver* :

- Opérations unidirectionnelles :
  - -- Il devient possible d'assigner des *channels* unidirectionnels à une boite-aux-lettres à travers $CREMBX et $ASSIGN

- -- Un *channel* unidirectionnel est un *channel* qui est *read-only* ou *write-only*
- -- Jusqu'à présent, les boites-aux-lettres étaient bi-directionnelles

- Possibilité de type *read streams* :
  - -- Il devient possible de spéficier un nombre de *bytes* à lire en utilisant une requête IO$M_STREAM
  - -- Jusqu'à présent, il n'était possible que de lire les données d'une boite-aux-lettres en mode *record*
- Possibilités d'attendre l'assignation d'un *channel* en lecture/écriture :
  - -- Un appel $QIO en direction d'une boite-aux-lettres autorise l'utilisateur à attendre qu'un *read channel* ou un *write channel* soit assigné à la boite-aux-lettres

  - -- 2 type de requêtes : IO$M_READERWAIT et IO$M_WRITERWAIT sur une opération IO$_SETMODE
  - -- Si l'on poste ce genre de requêtes sur un *channel* bi-directionnel, la requête se termine immédiatement puisque le *channel* attendu existe déjà
- Possibilité de *room notification* :
  - -- Un utilisateur peut être averti du fait qu'il y a de la place libre dans la boite-aux-lettres
- Complétude de $QIO quand un *channel* unidirectionnel est "désassigné" :
  - -- Une requête $QIO WRITE sur un *channel* uni-directionnel *read-only* peut utiliser un modifier IO$M_READERCHECK. Dans ce cas, le $QIO se termine avec le *status* SS$_NOREADER si tous les *channels* as-

signés en lecture sont "desassignés" avant la requête d'ecriture

-- Une requête $QIO READ sur un *channel* unidirectionnel *write-only* peut utiliser un modifier IO$M_WRITERCHECK. Dans ce cas, le $QIO se termine avec le *status* SS$_NOWRITER si tous les *channels* assignés en écriture sont "desassignés" avant la requête de lecture

- Interface SENSEMODE :
  -- Permet de déterminer les informations suivantes :
    > nombre de messages dans la boite-aux-lettres
    > nombre de *bytes* dans la boite-aux-lettres
    > l'absence de *channel* en lecture ou en écriture assigné sur la boite-aux-lettres

Modification de TTDRIVER :

- Ajout d'un ensemble de *vector entries* contenant les adresses d'un certain nombre de routines spéciales pour appeler le *terminal driver* POSIX
- Ces *vector entries* sont toujours disponibles mais uniquement utilisées si le *terminal driver* POSIX est chargé et que le terminal est paramètré POSIX
- Un ensemble de nouveaux codes de fonctions pour SET, SENSE, READ et WRITE afin de supporter une nouvelle méthode de contrôle de flux : COMMSYNC
  - -- le but de cette méthode de contrôle de flux est de pouvoir DSR/CTS au lieu de XON/XOFF
  - -- les 2 modes COMMSYNC et MODEM sont mutuellement exclusifs

-- En mode COMMSYNC :

> TTDRIVER positionne DTR et RTS et attend que les signaux CTS et DSR soient positionnés

> Alors, TTDRIVER émet sur la ligne jusqu'à ce que l'un des signaux CTS, DSR (ou les deux) soit perdu

> S'il n'y a pas de *status* d'erreur à ce moment, TTDRIVE cesse d'émettre :

+ COMMSYNC ne fonctionne que sur des lignes *full modem*, pas sur des lignes DEC423

+ TTDRIVER "fait de son mieux" pour cesser d'émettre : il peut y avoir quelques caractères perdus

+ Certaines imprimantes (lors de leur mise sous tension) positionnent le signal CTS et DSR avant d'être **réellement** prêtes :

dans ce cas, il peut y avoir des pertes de données

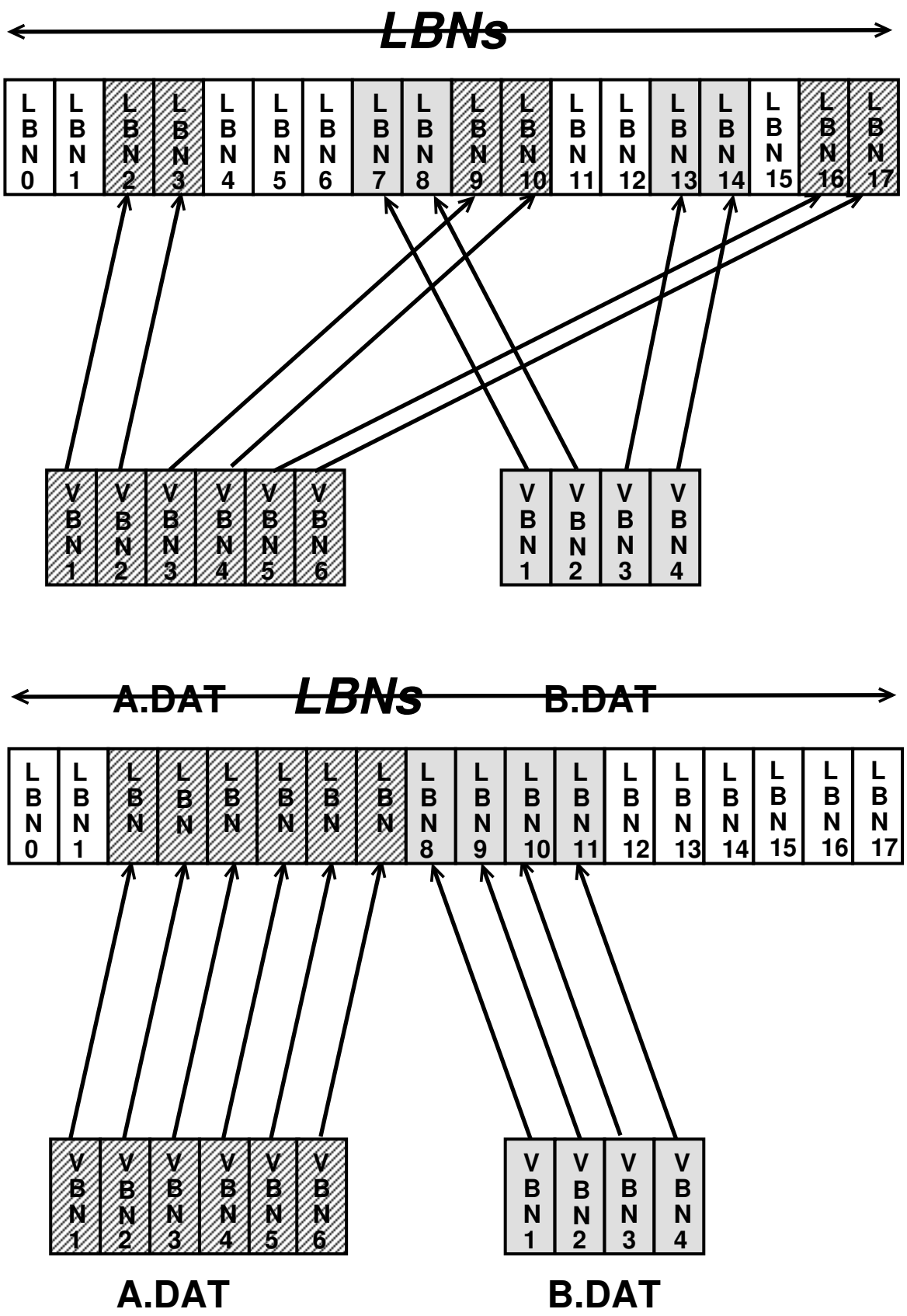
> Les applications qui désirent savoir quand une imprimante est *stalled* peuvent utiliser un algorithme du genre :

```
$SETTIMR(EFN,time,0,reqid,0)
$QIO(EFN,chan,IO$_WRITE,iosb,0,0,buff,bufsiz,0,0,0,0)
$WAITFR(EFN)
        IF (IOSB.IOSTAT==0) THEN
                PRINT("Printer Stalled")
        ELSE
                $CANCTIM(reqid,0)
                Check status.....
        ENDIF
```

Support d’un "défragmenteur"

- XQP a été modifié pour ajouter une fonctionnalité **majeure** au *file system* de VMS afin d’autoriser les déplacements de fichier de façon sécurisée
- Cette fonctionnalité permet de **déplacer** les LBNs vers lesquels les VBNs d’un fichier sont mappés

Support d'un "défragmenteur"



- Autorise l'implantation sécurisée d'un "defragmenteur"
- Implanté sous la forme d'un *modifier* de la fonction $QIO IO$_MODIFY : IO$M_MOVEFILE
- Cette primitive permet de déplacer un fichier ou un ensemble de VBN consécutifs d'un même fichier à un nouvel emplacement
- Pour pouvoir opérer, les conditions suivantes doivent être remplies :
  - -- Accès en écriture au fichier
  - -- Fichier fermé
  - -- Opérations de déplacement autorisées
  - -- Opération non interrompue : si une opération *movefile* est interrompue par une autres opération (lecture ou écriture), le fichier est laissé en l'état

  - -- L'opération *movefile* a besoin de 12800 *bytes* comme *user buffer* minimum pour fonctionner
  - -- Le périphériques source et destination doivent être identiques
- Une nouvelle commande DCL : $SET FILE/[NO]MOVE permet d'exclure les fichiers de l'action de la primitive MOVEFILE
- Cette opération de marquage est une nouvelle fonction $QIO ACP
- DIRECTORY et DUMP permette de visualiser cela

```
$ SET FILE/NOMOVE TEST.DAT
$ DIR/FULL TEST.DAT

Directory USER1:[BERNARD]
TEST.DAT;1                          File  ID:   (4629,1,0)
...
File attributes:    Allocation:3, Extend: 0, Global buffer count: 0
                    No version limit, MoveFile disabled
```

- Pendant la procédure d’installation de VMS, certains fichiers système sont automatiquement /NOMOVE

Support de DECdtm V1.1

- DECdtm V1.1 introduit les nouvelles fonctionnalités suivantes :
  - -- La notion de *commit domain*
  - -- De nouveaux services système
  - -- Le support d'*extended reason code* lors d'un *abort*
  - -- Le support de *timeout* sur les transactions
  - -- De nouveaux items pour $GETJPI
  - -- De nouvelles commandes SDA
- Notion de *commit domain* :
  - -- Un *commit domain* est l'ensemble formé par un *transaction manager* **et** son *log file*
  - -- Essentiellement équivalent à un nom de noeud

-- Ces *commit domains* sont nommés SYSTEM$nom_de_noeud

-- Ces *commit domains* sont identifiés par leur nom afin de pouvoir être utilisés par $ADD_BRANCH et $START_BRANCH

-- Pendant son exécution, une transaction peut couvrir plusieurs *commit domains*

- Nouveaux services systèmes :

  -- Les services systèmes suivants sont introduits par DECdtm V1.1 :

  > $ADD_BRANCH : permet une transaction de devenir un segment subordonné dans une transaction distribuée

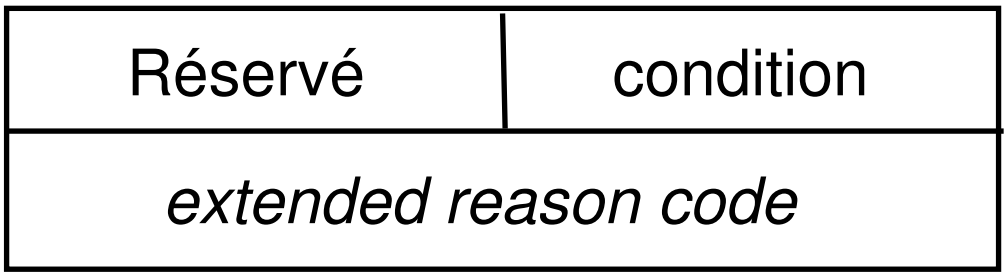
  > $START_BRANCH(W) : permet d'inclure un segment subordonné dans une transaction distribuée

> $READY_TO_COMMIT(W) : permet d'informer le *transaction local* qu'une transaction est terminée

-- En V1.0, DECdtm permettait à **une** application de manipuler plusieurs ressources (Rdb, RMS,...) sur plusieurs noeuds dans le contexte d'une seule transaction; en V1.1, DECdtm étend cette possibilité à **plusieurs** applications

- Support d'*extended reason code* :
  -- Permet de mieux faire la différence entre les différentes raisons d'échec de transaction
  -- Permet à une application de préciser "sa" raison pour affectuer un *abort*
  -- Implanté sous la forme d'un argument additionnel à $ABORT_TRANS(W)
  -- Cet *abort reason code* est retourné dans l'IOSB de $ABORT_TRANS(W), END_TRANS(W) et $READY_TO_COMMIT

-- Format de l'IOSB :



- Support de *timeout* sur les transactions:
  -- Il devient possible de spécifier un *timeout* sur une transaction en indiquant un temps limite
  -- Pour établir ce *timeout*, il faut appeler $START_TRANS avec un argument supplémentaire : TIMEOUT
  -- Cette valeur est une valeur borne pour que la transaction atteigne un point de *commit*
- Nouveaux items pour $GETJPI :
  -- Valable aussi pour F$GETJPI

- -- JPI$_CD_NAME : nom du *commit domain* du *process*
- -- JPI$_DEFAULT_TID : *transation id* par défaut du *process*

- Nouvelles commandes SDA :
  - -- Les commandes SDA permettant de supporter DECdtm ont été isolées dans une image partageable SDATP$SHARE.EXE spécifique invoquée dynamiquement lors du premier appel à une commande concernant DECdtm
  - -- Les commandes sont les suivantes :
    - > SHOW TRANSACTIONS
    - > SHOW COMMIT DOMAIN
    - > SHOW LOGS
    - > SHOW PROCESS

- Architecture du support des UID (*Unique Identifier)* Phase V nécessaire pour identifier de façon unique une transaction (tid) :
  - -- 3 nouvelles routines de la *Run Time Library* :
    - > LIB$ASCII_TO_UID : convertit une chaine de caractères ASCII en UID sur 48 bits
    - > LIB$UID_TO_ASCII : convertit un UID sur 48 bits en une chaine de caractères ASCII
    - > LIB$COMPARE_UID : compare 2 UIDs
  - -- 2 nouveaux paramètres spéciaux :
    - > IEEE_ADRESS et IEEE_ADDRESSH
    - > Paramètres dynamiques permettant de composer une adresse unique sur 48 bits qui est utilisée pour générer des UIDs uniques

dans le service système $CREATE_UID

> Positionnés par TP_SERVER

Support de hooks DECnet Phase V

Un certain nombre de *hooks* et de nouvelles fonctionnalités ont été ajoutés à VMS pour le support de DECnet PHASE V:

- *Hooks* :
  - -- CLUSTER_CONFIG.COM
  - -- AUTHORIZE
  - -- SHOW NETWORK
  - -- VMS$CONFIG-050_VMS.COM
  - -- LOGINOUT
  - -- SDA
  - -- SET TERMINAL
  - -- *Datalink drivers*
- Les 2 services système $DNS et $DNSW qui ont été introduits avec VMS

V5.3 étaient implantés sous la forme d'une *privileged shareable library* : ils sont intégrés comme services systèmes dans une *loadable image*

- Support de DTSS :
  -- Un nouvelle cellule système EXE$GQ_TDF (*time differential factor*) a été ajoutée; cette cellule maintient le *timezone offset* par rapport au méridien de Greenwhich
  -- Cette cellule est gérée par DTSS

Nouveautés RTL

Run Time Library COBOL :

- Un certain nombre de nouvelles routines de la *Run Time Library* COBOL ont été ajoutées pour étendre les améliorations de performances de la V4.4 du compilateur COBOL pour le traitement des données de type *packed decimal* sur les VAXes qui émulent les instructions travaillant sur ce type de données
- Amélioration sur les opérations effectuées sur des données binaires de 18 digits :
  - -- Stoquées sur un quadword (64 bits)
  - -- Ces opérations nécessitent une représentation intermédiaire plus importante (64 bits * 64 bits -> 128 bits)

-- Avant V4.4, pour multiplier 2 quad-words :

COB$CVTQP Convert quad->packed
MULP Multiply packed
COB$CVTPQ Convert packed->quad

-- Avec V4.4, pour multiplier 2 quad-words :

EXTV Extend quadword to octaword
COB$MULO Multiply octaword
COB$CVTOQ Convert octaword to octaword

-- Amélioration des performances par un facteur 20

Run Time Library PPL :

- Amélioration de PPL$UNIQUE_NAME pour retourner un nom unique à une application (un ensemble de *processes*) et non plus unique à un *process*
- Amélioration de PPL$WAIT_AT_BARRIER, PPL$DECREMENT_SEMAPHORE, PPL$REMOVE_WORK_ITEM pour admettre une nouvelle option permettant aux

processes d'effectuer un *spinwait* au lieu d'"hiberner" quand ceux-ci sont bloqués sur une attente de synchronisation

Création d'une ***Fast-Vector Math Library***

- Cette nouvelle RTL (partie de MTHRTL) permet d'améliorer considérablement les performances des traitements mathématiques vectoriels par l'appel de routines mathématiques spécialisées
- Ces routines sont appelées automatiquement lors de l'utilisation des *qualifiers* /VECTOR et /MATH_LIBRARY=FAST lors de la compilation en utilisant le compilateur FORTRAN HPO V1.2
- Ces routines permettent essentiellement des calculs trigonométriques, des calculs de racines carrées et des calculs d'exponentielles

- La précision de ces routines n'est **pas** garantie :
  - -- Erreur maximum de 2 LSBs sur les routines F_float
  - -- Erreur maximum de 4 LSBs sur les routines D_float ou G_float
- Améliorations potentielles des performances : entre 15 et 300% en fonction des routines et des applications

Nouveautés DCL

Nouvelles commandes DCL :

- $ INITIALIZE/SIZE=n
  - -- Utilisé pour le produit DECram
  - -- Permet de spécifier la taille d'un disque virtuel en mémoire
  - -- n <= 524280 *blocks*
  - -- n=0 permet de "désallouer" la place occupée en mémoire
- $ SET VOLUME/REBUILD[=FORCE]
  - -- Permet de remettre à jour de façon non conditionnelle le *free block count* dans le *lock value block* du volume disque
  - -- Introduit depuis environ 6 mois dans *Télépatch*

- $ SET MAGTAPE/RETENTION :
  - -- Permet de déplacer la bande dans la cartouche jusqu'à l'EOT et de la re-positionner au début
  - -- Doit être utilisé régulièrement afin d'aider au maintien de l'intégrité des cartouches TZK10
  - -- Doit être utilisé avec /REWIND ou /UNLOAD
- $ BACKUP/RELEASE_TAPE :
  - -- Permet de démonter une bande et de la décharger après une opération de sauvegarde
  - -- Si l'on utilise ce nouveau *qualifier* avec les *qualifiers* /DELETE ou /RECORD, cela permet de relacher la bande (et donc le dérouleur) immédiatement après l'écriture et AVANT l'opération "DELETE" ou "RECORD"

-- Introduit depuis environ 4 mois dans *Télépatch*

- $ BACKUP/IGNORE= :
  -- Admet un nouveau mot clé : ACCESSIBILITY
  -- Permet à BACKUP de "monter" une bande protégée par un caractère d'accès ou une bande crée par un HSC

Fonction lexique :

- F$MESSAGE :
  -- admet un nouvel argument : F$MESSAGE(status,message_component)
  -- Ce nouvel argument admet les 4 valeurs suivantes :
    > FACILITY
    > SEVERITY

- > IDENT
- > TEXT

-- Exemple :

```
$ERROR_INFO=F$MESSAGE(%X1C,"TEXT")
$SHOW SYMBOL ERROR_INFO
$ ERROR_INFO="EXCEEDED QUOTA"
```

Batch Log Time Stamps :

- Le nouveau sous-système de *batch/print* introduit la notion de *time stamp* dans les fichiers "log" du batch
- Implanté à l'aide de la nouvelle commande DCL :

  $ SET [NO]PREFIX *string*

  -- Remplace le préfixe de vérification par un préfixe spécifié dans la chaine de caractère *string*

  -- Utilise une syntaxe $FAO avec quelques restrictions

-- Admet des directives de date et de temps (!%T et !%D)

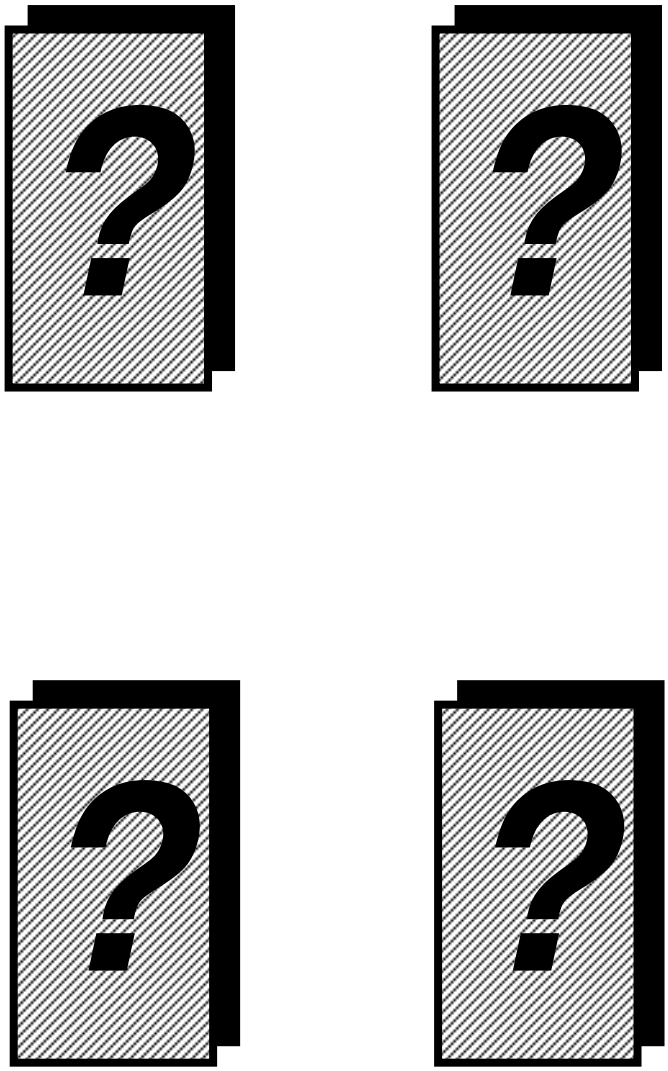
-- La chaine de caractère résultante doit avoir moins de 64 caractères

-- Le mécanisme de *time stamp* intervient pour chaque commande **vérifiée** ($SET VERIFY)

- Exemple :

```
$ SET VERIFY
$ SET PREFIX "(!5%T) "
$  @TEST
$ (14:18) $ SET DEFAULT SYS$LOGIN
$ (14:18) $ SHOW DEFAULT
  USER1:[BERNARD]
```

VMS FMR

Visible à l'exposition

Questions