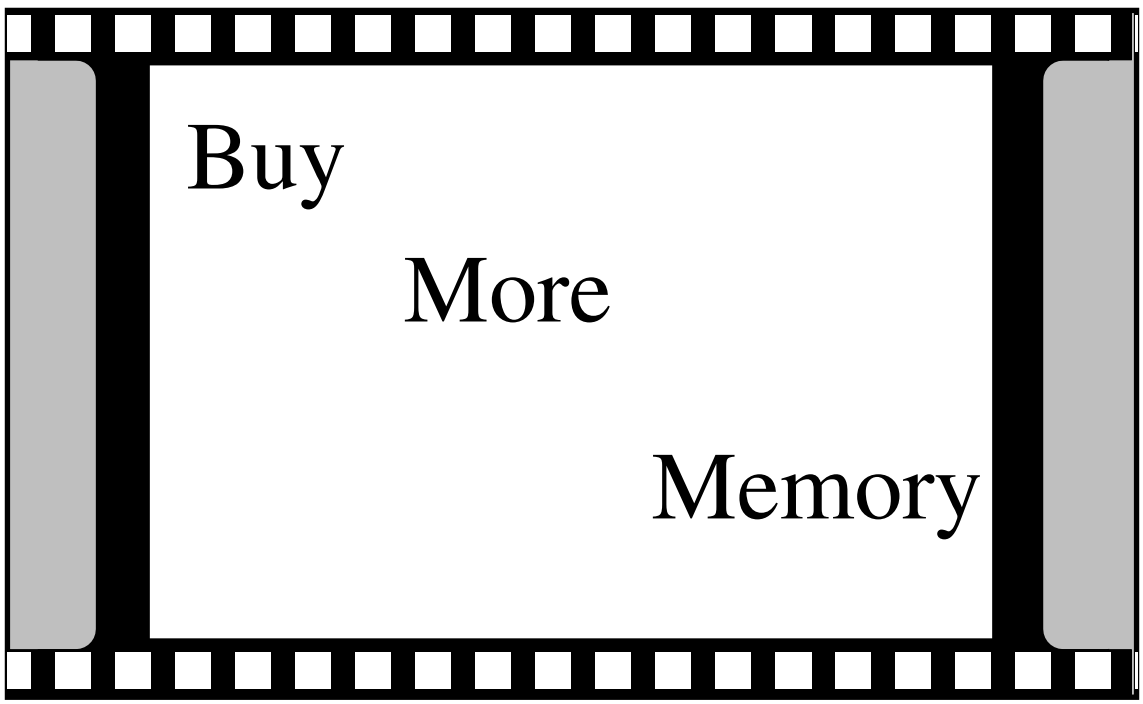
Nouveaux algorithmes de gestion de mémoire sous VMS

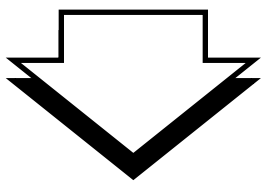
Overview

Bernard OURGHANLIAN

4-APR-1991

Traditionnellement....



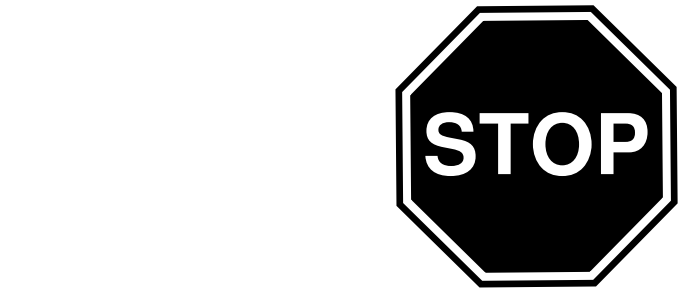


digital

Donc.....

digital

STOP !!!!



digital

Sommaire

1. Introduction :
   a. Pourquoi changer ?
   b. Comment changer ?
2. Les mécanismes :
   a. TROLLING
      (1) Overview
      (2) Implementation
   b. TICKING
      (1) Overview
      (2) Implementation
3. Paramètres système et quotas
4. Performances
5. Questions

Introduction
Auto-Tuning

- Un projet de recherche à long terme a démarré depuis novembre 1988 au sein de l'*engineering* VMS afin d'explorer des mécanismes permettant au système d'exploitation de s'adapter dynamiquement à sa charge de travail
- Les buts recherchés sont les suivants :
  - -- réduire la complexité et le niveau d'expertise système nécessaire pour gérer un système VMS
  - -- réduire les coûts de gestion manuelle du système (paramètres système et quotas utilisateurs)
  - -- réduire les temps d'arrêt système dus à un mauvais paramètrage

-- limiter (voire supprimer) les paramètres système VMS non dynamiques

- Un certain nombre de projets sont en cours :
  -- gestion dynamique de la mémoire
  -- gestion dynamique des *pools*
  -- élimination des causes de problèmes potentiels posés par :
    > la consommation complète de la *global page table*
    > la consommation complète de la *pool* paginée
    > la consommation complète de S0 (*balance slots*)
- Le résultat du premier projet : "gestion de la mémoire" est intégré avec la version V5.4-3 (?)

Introduction
Pourquoi changer ?

- La consommation mémoire des nouvelles applications augmente (applications orientées objet, DECwindows, Intelligence Artificielle,...)
- Aujourd'hui, un grand nombre de ressources de VMS sont statiques car "mappées" par la *system page table*
- La gestion de mémoire pratiquée jusqu'alors par VMS était essentiellement réactive et, dans certains environnements, inefficace :
  -- VMS ne réclame pas la mémoire à des processes inactifs tant que la mémoire libre n'est pas épuisée (la taille de la *free list* tombe en dessous de FREELIM)

-- La décrémentation automatique des des *working sets* (si elle est active : PFRATL <> 0) pourrait sembler non réactive, MAIS :

> ce mécanisme ne concerne PAS les "bons" processes

> la décrémentation automatique des *working sets* n'est effectuée que pendant l'expérience de *quantum end* (SCH$QEND dans RSE)

> les *processes* inactifs ne seront pas concernés car n'expérimentant **pas** *quantum end*

> ce sont les processes actifs (ou qui font très peu de *page faults*) qui sont concernés par ce mécanisme

> quand la taille de la *free list* tombe en dessous de FREELIM, VMS réclame le mémoire depuis TOUS les *processes* (premier niveau de

trimming effectué par le *swapper*) pour réduire la taille de leur *working set* jusqu'à WSQUOTA)

-- Ce premier niveau de *trimming* dépend donc de façon très importante de la valeur de WSQUOTA (très difficile à ajuster dans un cluster ayant un fichier SYSUAF unique pour des utilisateurs utilisant des applications ayant des profils de consommation mémoire très divers)

-- Les mécanismes actuels peuvent conduire à une dégradation non linéaire des performances en cas de charge importante

- Tous les mécanismes de régulation et de contrôle de l'utilisation de la mémoire sont régis par un *grand nombre de paramètres système* : autorise, voire encourage les erreurs

Introduction
Comment changer ?

- **Simple !! :**
  - -- quand il y a peu de mémoire disponible, on la prend aux *processes* inactifs pour la donner aux *processes* actifs
  - -- on tente de prendre cette mémoire le plus tôt possible grâce à une stratégie **non réactive**
  - -- on ne dépend plus de la "magie" de WSQUOTA
- **Égalitaire** !! :
  - -- on prend aux riches pour donner aux pauvres....
  - -- "The Robin-Hood of memory management!" (George Claborn)

Les mécanismes

- Le "Troller" : le "pêcheur" de pages
- Le "Ticker" : vise les *processes* ayant des activités périodiques
- **Hypothèse de base :**

Swapper n'est pas interdit !!! :

-- permet de libérer la mémoire avec très peu d'I/Os

-- permet de préserver l' investissement en *working set* des *processes* : après une opération d'*inswap*, le process est prêt à travailler sans avoir à regagner toutes ses pages en paginant

-- MAIS :

> considéré comme un "*last gasp reclaimer*" quand la mémoire est complètement épuisée : souvent c'est trop tard

> quand un process qui a été *outswapped* devient *computable*, il y a peu de chance que la mémoire ne soit pas encore complètement épuisée et que cela ne nécessite pas l'*outswap* d'un autre process : il peut donc en resulter un phénomène de *thrashing*

> quand un process est *outswapped*, celui-ci est *trimmé* jusqu'à SWPOUTPGCNT, donc le nombre d'entrées dans la *working set list* est réduite jusqu'à SWPOUTPGCNT ET la taille de la *working set list* est aussi réduite jusqu'à SWPOUTPGCNT : il en résulte que ce *process,* quand il

est à nouveau *inswapped*, doit justifier à nouveau la taille originale de son *working set list limit* à travers le mécanisme d'AWSA et donc à l'aide de nombreuses *page faults*

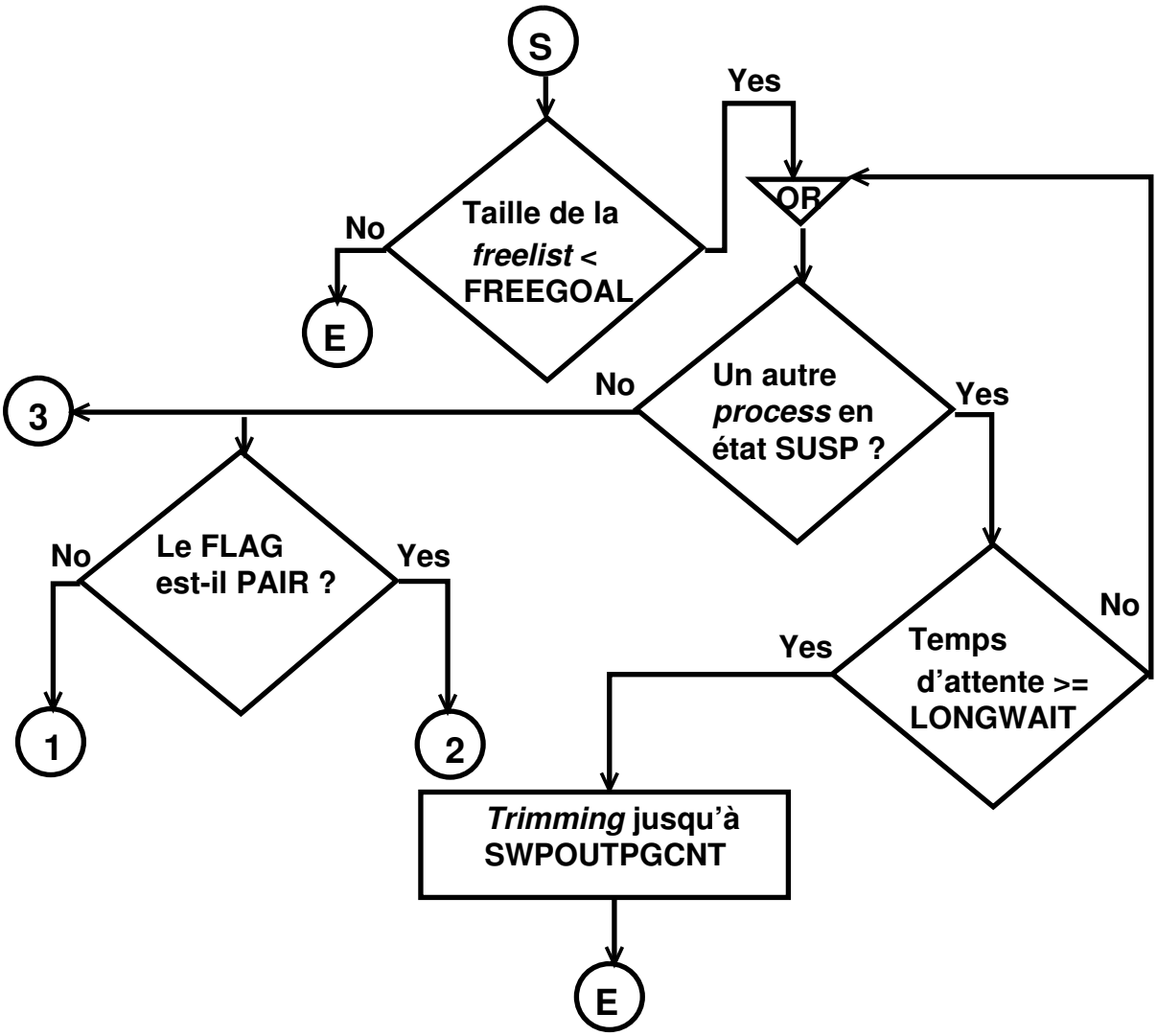
Le Troller
Overview

- Son but est de réclamer préventivement la mémoire
- Il démarre quand la taille de la *free list* tombe en dessous de FREEGOAL, PAS FREELIM
- Ceci permet de réclamer de la mémoire suffisamment tôt pour permettre de maintenir un cache de pages suffisant sur la *free list*
- Le ***Troller***, quand il est actif, recherche périodiquement les processes qui ont été inactifs dans les états SUSP, HIB, LEF ou CEF pendant plus de LONGWAIT secondes
- Si l'on trouve de tels processes et que l'on peut les *swapper*, leur *working set*

est *trimmé* jusqu'à SWPOUTPGCNT, puis, éventuellement, on les *outswap*

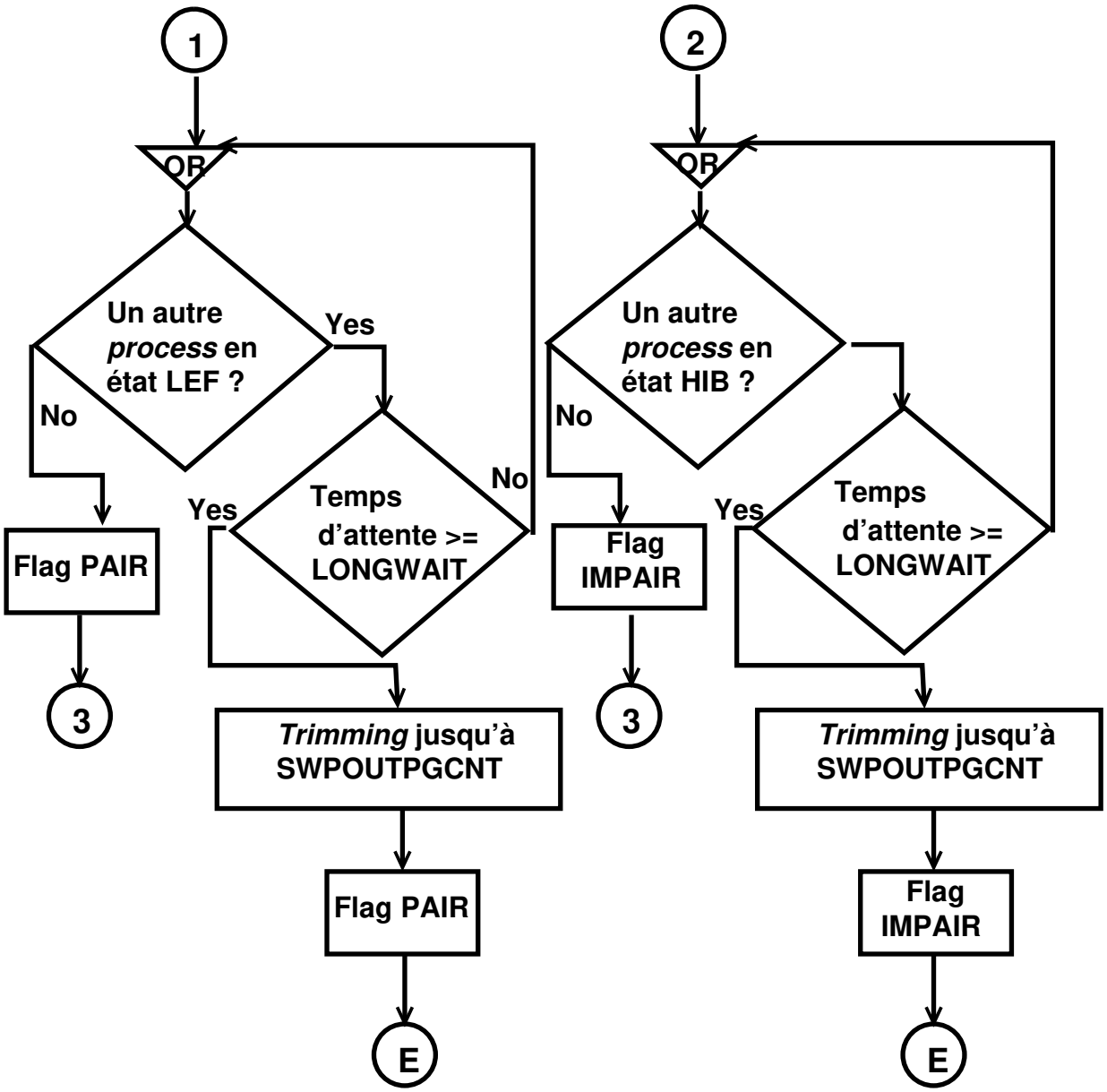
- Le *working set* est *trimmé* MAIS la taille du *working set* n'est PAS réduite : ceci permet ne pas avoir de avoir à re-justifier la taille de son *working set*
- Le ***Troller*** cherche toujours à éliminer le déficit =
  FREEGOAL- taille de la *freelist*
- S'il reste un déficit après avoir trouvé un process candidat et l'avoir *outswappé,* le ***Troller*** recherche un autre *process* candidat et le traite de la même façon
- Si le ***Troller*** ne trouve pas de *process* candidat (plus de *process* inactif pendant plus de LONGWAIT secondes), il détermine quand le prochain *process* candidat potentiel va devenir vraiment un *process* candidat, afin de savoir quand il sera nécessaire d'être réveillé

Le Troller
Overview



digital

Le Troller
Overview



digital

Le Troller
Implementation

- Pas de *process* séparé de type "Watch-dog" : toute la logique du ***Troller*** est implémenté dans une nouvelle routine du SWAPPER : TROLLER
- Modifications dans :
  - -- RSE (*Report System Event*)
  - -- SWAPPER
- RSE :
  - -- modification de routine SCH$SWPWAKE (routine de réveil du SWAPPER) en insérant après le code de vérification de l'existence d'un *powerfail* :
    - > test du fait que le *Trolling* soit autorisé

> si oui, test du fait qu'il reste plus de FREEGOAL pages sur la *free page list*

> si non, test du fait que 5 secondes se soient ecoulées depuis un dernier ***TROLL*** improductif

> si oui, réveil du SWAPPER

-- Le code :

```
        TSTL      G^EXE$GL_PFATIM              ; WAS THERE A POWER FAIL RE-
                                               ; COVERY?

******* NOUVEAU CODE INSÉRÉ *******

        BNEQ      10$                          ; BR IF SO
        BBC       S^#MMG$V_TROLL_ENABLE,-      ; IS TROLLING ENABLED ?
                  G^MMG$GB_CTLFLAGS,20$        ; BR IF NO
        CMPL      G^SCH$GL_FREECNT,-           ; IS FREECNT GEQ FREEGOAL ?
                  G^SGN$GL_FREEGOAL            ;
        BGEQ      20$                          ; BR IF YES
        CMPL      G^EXE$GL_ABSTIM_TICS,-       ; IS IT TIME FOR ANOTHER
                  G^SWP$GL_TROLL_TIME          ; TROLL ?

******* FIN D'INSERTION *******

        BLSSU     20$                          ; BR IF NO
```

- SWAPPER :
  -- Modification de la routine BALANCE (*Balance Free Page Count*) en insérant après le code de vérification de la nécessité de purger les PHDs :
    > test du fait que le *Trolling* soit autorisé
    > si oui, test du fait qu'il reste plus de FREEGOAL pages sur la *free page list*
    > si non, test du fait qu'il soit temps d'effectuer un nouveau ***Troll***
    > si oui, on positionne un flag et on passe le contrôle à la routine OUTSWAP du SWAPPER en utilisant un chemin standard du code

-- Le code :

```
BRW       50$                    ; FOUND SOME, FREE THEM UP

******* NOUVEAU CODE INSÉRÉ *******

15$:    BBC         S^#MMG$V_TROLL_ENABLE,-      ; IS TROLLING ENABLED ?
                    G^MMG$GB_CTLFLAGS,20$        ; BR IF NO

          CMPL      G^SCH$GL_FREECNT,-           ; IS FREECNT > FREEGOAL ?
                    G^SGN$GL_FREEGOAL            ;
          BGTR      20$                          ; BR IF YES

          CMPL      G^EXE$GL_ABSTIM_TICS,-       ; IS IT TIME FOR ANOTHER ?
                    G^SWP$GL_TROLL_TIME          ; TROLL ?
          BLSSU     20$                          ; BR IF NO

          MOVL      #1,R3                        ; ELSE, SET TROLL REQUEST
                                                 ; FLAG

          BRB       60$                          ; AND GO DO IT !

******* FIN D'INSERTION *******

20$:      ; Nothing to do - unlock MMG and SCHED databases RECOVERY?
```

-- Modification de la routine OUTSWAP:

> vérification au point d'entrée de la routine si l'on a effectué une requête de TROLL. Si c'est le cas, on saute la partie de code qui vérifie l'existence de PHD libre et on appelle directement OSWPSCHED pour effectuer le travail d'*outswap*

> le code :

```
OUTSWAP:                                ; TRY TO OUTSWAP
         TSTL     FP                    ; ARE WE HERE TO PERFORM A TROLL ?
         BGTR     CALL_OSWPSCHED        ; SKIP PHD SCAN IF SO !
```

> OUTSWAP utilise FP comme argument d'entrée : si FP est positif, cela indique une tentative de TROLL (ici FP contient 1)

> Lors du retour d'OSWPSCHED le code (déjà existant) vérifie si l'on a trouvé un candidat à l'opération d'*outswap*

-- Modification de la routine OWSPSCHED:

> test du fait que le *Trolling* soit autorisé

> si oui, on se branche sur la routine qui effectue l'opération de ***Trolling***

> le code :

```
SCH$OSWPSCHED::                         ; SCHEDULE OUTSWAP

        BBC     S^#MMG$V_TROLL_ENABLE,-     ; IS TROLLING ENABLED ?
                G^MMG$GB_CTLFLAGS,8$        ; BR IF NO

        JSB     TROLLER                     ; ELSE, GO TROLL

        TSTL    R4                          ; DID WE FIND A PROCESS TO
                                            ; OUTSWAP ?
        BEQL    6$                          ; BR IF NO
```

> la routine TROLLER effectue un parcours abrégé de la table OSWPSTATE pour trouver les *processes* se trouvant dans des états SUSP, HIB, LEF ou CEF depuis plus de LONGWAIT secondes

> si l'on trouve un tel *process*, et que l'on peut le *swapper*, celui-ci est *trimmé* à SWPOUTPGCNT pages par la routine MMG$FREE_PAGES,

> l'adresse de son PCB est retourné dans R4 pour permettre le traitement par le reste de la routine OUTSWAP. Si l'on n'a pas trouvé

de candidat, R4 est retourné avec un contenu égal à 0.

> MMG$FREE_PAGES remplace l'appel à MMG$SHRINKWS par un appel à MMG$FREEWSLX. Cette routine ne change PAS WSSIZE.

> le code :

```
          .SBTTL    TROLLER
;++
; FUNCTIONAL DESCRIPTION:
;         THIS ROUTINE FINDS A SUITABLE PROCESS FOR OUTSWAP USING THE TROLL-
;         STATE
;         TABLE FOR GUIDANCE.
;
; CALLING SEQUENCE:
;         JSB       TROLLER
;
; INPUT PARAMETERS:
;         FP - IF FP = %X80000000, THEN WE'RE HERE TO FREE A BALANCE SET SLOT
;              IF FP IS NEGATIVE, THEN WE'RE HERE TO REPLENISH THE FREE PAGE
;              LIST
;              IF FP IS POSITIVE, THEN WE'RE HERE TO TROLL
;
;
; OUTPUT PARAMETERS:
;         R4 = ADDRESS OF OUTSWAP CANDIDATE
;
; SIDE EFFECTS:
;         R0,R1,R2,R3,R4,R5,R6,R7,R8,R9 CHANGED
;
;--
TROLLER:

          MOVZWL    G^SCH$GW_LONGWAIT,R0 ; GET DYNAMIC SYSGEN PARAMETER
                                         ; LONGWAIT
```

```
        CMPL    G^SCH$GL_FREECNT,-  ; IS THE FREE LIST DEPLETED ?
                G^SCH$GL_FREELIM    ;
        BGEQ    5$                  ; BR IF NO

        ASHL    #-1,R0,R0           ; USE ONE HALF LONGWAIT IF YES

5$:     MULL3   #100,R0,W^LONGWAIT  ; CONVERT TO 10 MS UNITS
        MOVAB   TROLLSTATE,R3       ; START SCAN AT TOP OF TROLLSTATE
                                    ; TABLE

10$:    MOVL    R3,R2               ; SETUP FOR SCAN
        BEQL    50$                 ; RETURN TO CALLER IF WE'VE HIT THE
                                    ; END OF THE TROLLSTATE TABLE
        BSBW    CANDIDATE           ; GO FIND A CANDIDATE

20$:    BEQL    10$                 ; IF NONE FOUND, TRY NEXT SECTION

30$:    BSBW    SWAPQUANCHK         ; SEE IF PROCESS IS SWAPPABLE
        BLBC    R9,40$              ; CONTINUE TROLLING IF NOT

        BSBW    SHRINK_FOR_TROLL    ; SHRINK WSL TO APPROPRIATE SIZE

        BSBW    GET_SWAP_SPACE      ; GET SWAP SPACE
        BLBS    R9,60$              ; SKIP AHEAD IF SUCCESSFUL

40$:    JSB     @(SP)+              ; CONTINUE TROLL
        BNEQ    30$                 ; CANDIDATE FOUND, CHECK IT OUT
        BRB     10$                 ; CANDIDATE NOT FOUND, CONTINUE TROLL

50$:    CLRL    R4                  ; SIGNAL FAILURE TO FIND CANDIDATE
        BRB     70$                 ; SKIP COROUTINE ADDRESS POP

60$:   ADDL     #4,SP               ; THROW AWAY COROUTINE ADDRESS

70$:    BSBW    SET_NEXT_TROLL_TIME; SET NEXT TROLL TIME

        RSB                         ; RETURN TO CALLER

SHRINK_FOR_TROLL:

        MOVQ    R6,-(SP)            ; SAVE R6 & R7

        MOVL    G^SWP$GL_SWPPGCNT,R7        ; ASSUME THAT WE'RE NOT LOOK-
                                            ; ING FOR A BALANCE SET SLOT
        ASHL    #1,FP,R6            ; ARE WE LOOKING FOR A BALANCE SET
                                    ; SLOT ? (I.E. IS FP = 80000000)
        BNEQ    10$                 ; SKIP AHEAD IF NOT

        SUBL3   PHD$L_WSLIST(R5),-  ; TRIM PROCESS TO WSQUOTA IF WE'RE
                PHD$L_WSQUOTA(R5),R7; TRYING TO FREE UP A BALANCE SET
```

```
        INCL    R7                  ; SLOT

10$:    ADDL3   PCB$L_PPGCNT(R4),-  ; GET PROCESS'S CURRENT WS SIZE
                PCB$L_GPGCNT(R4),R6;

        SUBL    R7,R6               ; GET NUMBER OF PAGES TO RELEASE
        BLEQ    20$                 ; RETURN IF LEQ 0

        BSBW    MMG$FREE_PAGES      ; GO TRIM PROCESS

        MOVB    #255,PHD$B_PRVCPU(R5) ; FORCE ASN-QUALIFIED TB ENTRY
                                      ; FLUSH ON NEXT CONTEXT LOAD OF
                                      ; THIS PROCESS
        MOVL    PCB$L_VP_CONTEXT(R4),R6 ; GET ADDRESS OF VECTOR CONTEXT
                                        ; AREA
        BEQL    20$                 ; IF EQL NOT A VECTOR CONSUMER

        BISL    #VCTX$M_VTB,VCTX$L_FLAGS(R6); SET FLAG TO FLUSH VECTOR-TB

20$:    MOVQ    (SP)+,R6            ; RESTORE R6 & R7

        RSB                         ; RETURN

SET_NEXT_TROLL_TIME:

        MOVL    G^EXE$GL_ABSTIM_TICS,R0     ; CURRENT TIME -> R0
        MOVL    R0,R2                       ; CURRENT TIME -> R2
        MOVAB   G^SCH$GQ_SUSP,R1    ; ADDRESS OF SUSP WQHDR -> R1
        MOVQ    (R1),R7             ; FLINK -> R7, BLINK -> R8
        CMPL    R7,R8               ; IS SUSP Q EMPTY ?
        BEQL    20$                 ; IF SO, THEN GO CHECK HIB Q

10$:    BBS     #PCB$V_PSWAPM,-     ; BRANCH AHEAD IF PROCESS ISN'T
                PCB$L_STS(R8),15$   ; SWAPPABLE
        BRW     100$                ; IF PROCESS IS SWAPPABLE, THEN GO AND
                                    ; SET NEXT TROLL TIME TO CURRENT TIME
                                    ; OR CURRENT TIME + 5 SECONDS DEPEND-
                                    ; ING ON WHETHER LAST TROLL WAS SUC-
                                    ; CESSFUL
15$:    MOVL    PCB$L_SQBL(R8),R8   ; OTHERWISE BLINK TO NEXT OLDEST
                                    ; PROCESS IN SUSP QUEUE
        CMPL    R8,R1               ; HAVE WE LOOKED AT ALL SUSPS ?
        BNEQ    10$                 ; NO, GO SEE IF PROCESS IS SWAPPABLE

20$:    MOVL    R0,R2               ; CURRENT TIME -> R2
                                    ; (ASSUME NO SWAPPABLE HIBS)
        MOVAB   G^SCH$GQ_HIBWQ,R1   ; ADDRESS OF HIB WQHDR -> R1
        MOVQ    (R1),R7             ; FLINK -> R7, BLINK -> R8
        CMPL    R7,R8               ; IS HIB Q EMPTY ?
        BEQL    50$                 ; IF SO, THEN GO CHECK LEF Q
```

```
30$:    BBC     #PCB$V_PSWAPM,-     ; GO AND GET WAIT TIME IF PROCESS IS
                PCB$L_STS(R8),40$   ; SWAPPABLE
        MOVL    PCB$L_SQBL(R8),R8   ; OTHERWISE BLINK TO NEXT OLDEST
                                    ; PROCESS IN HIB QUEUE
        CMPL    R8,R1               ; HAVE WE LOOKED AT ALL HIBS ?
        BEQL    50$                 ; IF S0, GO CHECK LEF Q
        BRB     30$                 ; ELSE, GO SEE IF PROCESS IS SWAPPABLE

40$:    MOVL    PCB$L_WAITIME(R8),R2 ;WAITIME OF OLDEST SWAPPABLE
                                    ; HIB -> R2

50$:    MOVL    R0,R3               ; CURRENT TIME -> R3
                                    ; (ASSUME NO SWAPPABLE LEFS)
        MOVAB   G^SCH$GQ_LEFWQ,R1   ; ADDRESS OF LEF WQHDR -> R1
        MOVQ    (R1),R7             ; FLINK -> R7, BLINK -> R8
        CMPL    R7,R8               ; IS LEF Q EMPTY ?
        BEQL    80$                 ; IF SO, THEN THEN SKIP AHEAD

60$:    BBC     #PCB$V_PSWAPM,-     ; GO AND GET WAIT TIME IF PROCESS IS
                PCB$L_STS(R8),70$   ; SWAPPABLE
        MOVL    PCB$L_SQBL(R8),R8   ; OTHERWISE BLINK TO NEXT OLDEST
                                    ; PROCESS IN LEF QUEUE
        CMPL    R8,R1               ; HAVE WE LOOKED AT ALL LEFS ?
        BEQL    80$                 ; IF SO, SKIP AHEAD
        BRB     60$                 ; ELSE, GO SEE IF PROCESS IS SWAPPABLE

70$:    MOVL    PCB$L_WAITIME(R8),R3 ; WAITIME OF OLDEST SWAPPABLE
                                    ; LEF -> R3

80$:    MOVL    #SCH$C_HIB,R1       ; ASSUME THAT OLDEST SWAPPABLE PROCESS
                                    ; IS A HIB
        CMPL    R2,R3               ; COMPARE WAITIME OF OLDEST SWAPPABLE
                                    ; HIB TO WAITIME OF OLDEST SWAPPABLE
                                    ; LEF
        BLSSU   90$                 ; ENSURE THAT R2 CONTAINS THE EARLIER
        MOVL    R3,R2               ; OF THE TWO WAITIMES
        MOVL    #SCH$C_LEF,R1       ; OLDEST SWAPPABLE PROCESS IS A LEF

90$:    ADDL    W^LONGWAIT,R2       ; CALCULATE THE EARLIEST TIME AT WHICH
                                    ; THE OLDEST SWAPPABLE HIB/LEF COULD
                                    ; BECOME A LONGWAITER
        CMPB    R1,TROLLSTATE+3     ; DOES THE SCHEDULING STATE OF THE
                                    ; OLDEST SWAPPABLE PROCESS APPEAR AS
                                    ; THE SECOND ENTRY
                                    ; IN THE TROLLSTATE TABLE ?
        BEQL    100$                ; YES, NO NEED TO TRANSPOSE HIB/LEF
                                    ; ENTRIES

        XORB    TROLLSTATE+6,-      ; THESE 3 INSTRUCTIONS TRANSPOSE THE
```

```
                TROLLSTATE+3        ; HIB AND LEF ENTRIES IN THE
                                    ; TROLLSTATE
        XORB    TROLLSTATE+3,-      ; TABLE.
                TROLLSTATE+6        ;
        XORB    TROLLSTATE+6,-      ;
                TROLLSTATE+3        ;

100$:   MOVL    R0,R3               ; CURRENT TIME -> R3
        TSTL    R4                  ; WAS LAST TROLL SUCCESSFUL ?
        BNEQ    110$                ; SKIP AHEAD IF SO
        ADDL    #500,R3             ; ELSE, ADD 5 SEC TO CURRENT TIME TO
                                    ; DETERMINE THE EARLIEST TIME AT WHICH
                                    ; ANOTHER TROLL SHOULD BE PERFORMED

110$:   CMPL    R2,R3               ; COMPARE THE EARLIEST TIME AT WHICH
                                    ; THE OLDEST SWAPPABLE HIB/LEF COULD
                                    ; BECOME A LONGWAITER WITH THE EARLI-
                                    ; EST TIME AT WHICH THE NEXT TROLL AT-
                                    ; TEMPT
                                    ; IS DESIRED.
        BGTRU   120$                ; ENSURE THAT R2 CONTAINS THE LATER OF
        MOVL    R3,R2               ; THESE TWO TIMES

120$:   MOVL    R2,G^SWP$GL_TROLL_TIME ; STORE TIME FOR NEXT TROLL.WHEW !
        RSB

        .END
```

Le Ticker
Overview

- Son but est de compléter le mécanisme mis en place par le ***Troller***
- Créé pour traiter le cas des stations DECwindows sur lesquelles une bonne partie de la mémoire est détenue par des *processes* inactifs dont le comportement n'est pas celui visé par le mécanisme du ***Troller*** :
  - -- ces *processes* entrent dans un état d'attente quand ils sont inactifs MAIS se réveillent périodiquement pour exécuter une routine afin de voir s'il y a quelque chose à faire
  - -- ces *processes* ne paraissent pas inactifs pour le mécanisme de ***Trolling*** car alternant en permanence entre 2 états

-- typique des *processes* DECwindows :

> Clock

> Calendar

> Mail

> View

> ....

- Différences entre ***Troller*** et ***Ticker*** :

Troller	***Ticker***
. Réclame la mémoire depuis les *processes* inactifs depuis long-temps	**. Réclame la mémoire depuis les *processes* peu actifs**
. *Swappe* les *pro-cesses*	**. *Trimme* les *pro-cesses***

- Quand le ***Ticking*** est autorisé, les *processes* sont évalués quand ils entrent volontairement dans l'état HIB, LEF ou CEF pour voir s'ils ont accumulé 30 secondes d'*idle time* :
  - -- si non, le *process* rentre en *wait state* comme d'habitude
  - -- si oui, on calcule le rapport :

    CPU time / Wait time
- Si :
  - -- ce rapport est < 1/100
  - -- le *process* n'a pas de *direct I/O* en attente (car sinon, il y aurait de fortes chances que le *process* en question fasse bientôt quelque chose)
  - -- le *process* n'est pas un *process temps réel*
- Alors le *process* en question est candidat au mécanisme de ***Ticking***

- Dans ce cas, son *working set* est réduit (*trimmed*) de 25% MAIS la taille du *working set* n'est pas réduite : 25% a été choisi comme un compromis entre la réclamation maximale de la mémoire et une trop grande agresssivité pour un *process* donné
- Le ***Ticker*** démarre quand la taille de la *free list* tombe en dessous de 2*FREEGOAL :
  -- la raison pour choisir 2*FREEGOAL est liée au fait que le processus de TICKING réclame la mémoire plus lentement que le processus de TROLLING car le TICKING ne réclame de la mémoire QUE lorsque le *process* entre dans un état d'attente volontairement
  -- si les 2 mécanismes se mettaient en route lors du dépassement de la même valeur de seuil, le TROLLING réclamerait la mémoire si rapidement

que le TICKING n'aurait aucune chance de réclamer de la mémoire : ceci voudrait dire qu'une demande de pages sur la FPL ne pourrait pas être satisfaite par le mécanisme de TROLLING (si tous les *processes* de type LONGWAIT avaient déjà été *outswappés*) et que le mécanisme de TICKING ne serait pas assez rapide pour réclamer la mémoire, laissant la FPL avec trop peu de pages

Le Ticker
Implementation

- Pas de *process* séparé de type "Watch-dog" : toute la logique du ***Ticker*** est implémenté dans RSE, SYSWAIT et SYSPCNTRL
- RSE :
  - -- modification de routine SCH$ONE_SEC (routine appelée toutes les secondes par EXE$TIMEOUT pour effectuer l'ensemble des traitements périodiques du *scheduler*) :
    - > on vérifie si le ***ticking*** est autorisé
    - > si oui, test du fait qu'il reste plus de 2*FREEGOAL pages sur la *free page list*
    - > si non, positionnement d'un flag global

(bit dans MMG$GL_RECLAIM_FLAGS) qui est examiné dans la branche de code qui effectue l'opération de ***ticking*** conditionnel

> le code :

```
SCH$ONE_SEC::

        CLRL    G^MMG$GL_RECLAIM_FLAGS  ; CLEAR MEMORY RECLAMATION FLAGS

        BBC     S^#MMG$V_TICK_ENABLE,-  ; IS TICKING ENABLED ?
                G^MMG$GB_CTLFLAGS,1$    ; BR IF NO

        ASHL    #1,G^SGN$GL_FREEGOAL,R0 ; GET FREEGOAL AND DOUBLE IT

        CMPL    G^SCH$GL_FREECNT,R0     ; IS FREECNT GEQ 2 X FREEGOAL ?

        BGEQ    1$                      ; BR IF YES

        BBSS    S^#MMG$V_TICK_ACT,-     ; ACTIVATE TICKING MECHANISM
                G^MMG$GL_RECLAIM_FLAGS,1$ ;
1$:
```

-- modification de routine SCH$UNWAIT (permet de décrémenter le nombre de *processes* dans la *wait queue*) :

> ce code n'est executé que par les *processes* en *wait state*

> permet de calculer le temps pendant lequel le *process* a été dans un état d'attente et d'accumuler ce

temps dans le PCB (en PCB$L_ACC_WAITTIME)

> le code :

```
SCH$UNWAIT::                            ; DECREMENT PROPER WAIT COUNT
         MOVZWL    PCB$W_STATE(R4),R1  ; Get current process state number
         BBC       R1,B^WAITMSK,20$    ; SKIP OUT IF NOT WAIT STATE

******* NOUVEAU CODE INSÉRÉ *******

         SUBL3     PCB$L_WAITIME(R4),-; COMPUTE TIME THAT PROCESS WAS IN A
                   G^EXE$GL_ABSTIM_TICS,R0 ; WAIT STATE
         ADDL      R0,PCB$L_ACC_WAITIME(R4); ADD TO ACCUMULATED WAIT TIME

******* FIN D'INSERTION *******

         CMPW      #SCH$C_CEF,R1       ; CHECK FOR COMMON EVENT FLAG WAIT
```

-- nouvelle routine EXE$CHK_WAIT_BHVR qui effectue la fonction de ***ticking*** et est appelée conditionnellement dans SYSWAIT et SYSPCNTRL :

> calcul du rapport *:*
Wait time/CPU time

> si ce rapport est > 100, le *process* est considéré inactif et l'on effectue un certain nombre de vérifications supplementaires pour voir si l'on peut *trimmer* le *process* (test

du bit PHD$V_NO_WS_CHNG dans PHD$W_FLAGS)

> si l'on peut *trimmer* ce process, on réduit son *working set* de 25% sans changer sa taille en appelant MMG$FREE_PAGES

> le code :

```
EXE$CHK_WAIT_BHVR::

          MOVQ      R5,-(SP)            ; SAVE R5 & R6

          MOVL      PCB$L_PHD(R4),R5    ; GET ADDRESS OF PHD

          SUBL3     PCB$L_CPUTIME_REF(R4),- ; FIND OUT HOW MUCH CPU TIME HAS
                    PHD$L_CPUTIM(R5),R6     ; BEEN USED SINCE LAST TIME WE
                                            ; WERE HERE
          BEQL      10$                 ; SKIP AHEAD AND TRIM PROCESS IF ZERO

          DIVL3     R6,-                ; FIND RATIO OF WAIT TIME TO CPU TIME
                    PCB$L_ACC_WAITIME(R4),R6

          CMPL      R6,#100             ; COMPARE WITH "MAGIC" NUMBER
          BLEQ      30$                 ; SKIP TRIM IF RATIO OF WAIT TIME  TO
                                        ; CPU TIME IS LESS THAN MAGIC VALUE

10$:      LOCK      LOCKNAME=MMG,-      ; LOCK MMG DATABASE
                    PRESERVE=YES,-      ; PRESERVE R0
                    SAVIPL=-(SP)        ; SAVE IPL

          BBS       #PHD$V_NO_WS_CHNG,-; SKIP TRIM IF WE'RE NOT ALLOWED
                    PHD$W_FLAGS(R5),20$; TO TOUCH IT'S WSL

          ADDL3     PCB$L_PPGCNT(R4),- ; GET PROCESSES'S WS SIZE
                    PCB$L_GPGCNT(R4),R6;

          ASHL      #-2,R6,R6           ; FIND 25% OF WS SIZE
          BEQL      20$                 ; SKIP TRIM IF ZERO !

         JSB        G^MMG$FREE_PAGES    ; GO REPOSSESS SOME MEMORY !
```

```
20$:    UNLOCK   LOCKNAME=MMG,-      ; UNLOCK MMG DATABASE, LEAVE IPL
                 PRESERVE=YES,-      ; RESTORE R0
                 NEWIPL=(SP)+        ; RESTORE IPL

30$:    MOVL     PHD$L_CPUTIM(R5),- ; SAVE CPU TIME FOR FUTURE USE
                 PCB$L_CPUTIME_REF(R4) ;

        CLRL    PCB$L_ACC_WAITIME(R4) ; CLEAR ACCUMULATED WAIT TIME

        MOVQ     (SP)+,R5            ; RESTORE R5 & R6

        RSB                          ; RETURN

        .END
```

- SYSWAIT :
  - -- Insertion de code au label WAITCK pour tester si le ***ticking*** est autorisé
  - -- Si oui, on compare le champ du PCB où est accumulé le temps d'attente du process (PCB$L_ACC_WAITTIME) à 30 secondes
  - -- Si ce temps est supérieur à 30, on teste si le process n'est pas de type temps réel et s'il n'a pas de *direct I/Os* en cours
  - -- Si oui, on appelle la routine EXE$CHK_WAIT_BHVR

-- Le code :

```
WAITCK:                                 ;CHECK FOR WAIT SATISFIED

          ASSUME    MMG$V_TICK_ACT EQ 0; ENSURE CORRECTNESS OF BLBC
          BLBS      G^MMG$GL_RECLAIM_FLAGS,-; IS TICKING MECHANISM ACTIVE ?
                    10$                 ; BR IF SO
[...]
10$:      CMPL      PCB$L_ACC_WAITIME(R4),- ; HAS PROCESS ACCUMULATED 30 SECS
                    #3000               ; OF WAIT TIME YET ?
          BLSS      5$                  ; BR IF NO

          CMPB      PCB$B_PRI(R4),#15   ; IS PROCESS REAL TIME ?
          BLEQ      5$                  ; RETURN TO NORMAL WAIT PATH IF SO

          CMPW      PCB$W_DIOCNT(R4),-  ; DOES PROCESS HAVE A DIRECT I/O
                    PCB$W_DIOLM(R4)     ; OUTSTANDING ?
          BNEQ      5$                  ; RETURN TO NORMAL WAIT PATH IF SO

          JSB       G^EXE$CHK_WAIT_BHVR; GO SEE IF MEMORY SHOULD BE RECLAIMED
                                        ; FROM THIS PROCESS
          BRB       5$                  ; RETURN TO NORMAL WAIT PATH

          .END
```

- SYSPCNTRL :
  - -- Insertion de code au debut du *system service* $HIBER pour tester si le ***ticking*** est autorisé
  - -- Si oui, on compare le champ du PCB où est accumulé le temps d'attente du process (PCB$L_ACC_WAITTIME) à 30 secondes
  - -- Si ce temps est supérieur à 30, on teste si le process n'est pas de type

temps réel et s'il n'a pas de *direct I/Os* en cours

-- Si oui, on appelle la routine EXE$CHK_WAIT_BHVR

-- Le code :

```
        ASSUME  MMG$V_TICK_ACT EQ 0; ENSURE CORRECTNESS OF BLBC
        BLBS    G^MMG$GL_RECLAIM_FLAGS,-; IS TICKING MECHANISM ACTIVE ?
                15$                         ; BR IF YES
[...]
15$:    CMPL    PCB$L_ACC_WAITIME(R4),- ; HAS PROCESS ACCUMULATED 30 SECS
                #3000               ; OF WAIT TIME YET ?
        BLSS    5$                  ; BR IF NO

        CMPB    PCB$B_PRI(R4),#15   ; IS PROCESS REAL TIME ?
        BLEQ    5$                  ; RETURN TO NORMAL WAIT PATH IF SO

        CMPW    PCB$W_DIOCNT(R4),-  ; DOES PROCESS HAVE A DIRECT I/O
                PCB$W_DIOLM(R4)     ; OUTSTANDING ?
        BNEQ    5$                  ; RETURN TO NORMAL WAIT PATH IF SO

        BSBW    EXE$CHK_WAIT_BHVR   ; GO SEE IF MEMORY SHOULD BE RECLAIMED
                                    ; FROM THIS PROCESS
        BRB     5$                  ; RETURN TO NORMAL WAIT PATH
```

Paramètres système

- Le paramètre SYSGEN **MMG_CTLFLAGS** permet de contrôler les mécanismes d'Auto-Tuning :
  - -- Bit 0 =
    - > 0 : ***Ticker*** inactif
    - > 1 : ***Ticker*** actif
  - -- Bit 1 =
    - > 0 : ***Troller*** inactif
    - > 1 : ***Troller*** actif
  - -- Les autres bits du byte ne sont pas encore définis mais pourraient servir bientôt

-- D'une façon génerale, ce paramètre doit être vu comme un *switch* à 2 positions :

> 0 : Auto-Tuning inactif

> 255 : Auto-Tuning actif

-- Pour le moment, le défaut est 0

-- Le comportement final de la version V5.4-3 (?) n'est pas défini

- Le paramètre SYSGEN **FREEGOAL** permet de définir le nombre de pages que l'on essaye de maintenir sur la *free list*

  -- affecte directement les mécanismes d'Auto-Tuning

  -- si MMG_CTLFLAGS est <>0, AUTOGEN positionne FREEGOAL à 1/8 de la taille de la mémoire physique (borné entre 1600 et 12000 pages)

- Le paramètre SYSGEN **LONGWAIT** affecte l'"enthousiasme" du ***troller*** : une valeur plus grande du paramètre rend le

mécanisme de ***trolling*** moins agressif et une valeur plus faible rend le mécanisme plus agressif; le défaut (30 secondes) semble adéquat

- Les paramètres SYSGEN **FREELIM**, **GROWLIM** et **BORROWLIM** sont positionnés par défaut par AUTOGEN comme égaux :
  FREELIM=GROWLIM=BORROWLIM
  si le système est une station ou si MMG_CTLFLAGS est <>0.
  Ceci permet de maximiser la possibilité pour les *processes* d'étendre leur *working sets.* Ceci est déjà fait par AUTOGEN pour VMS V5.4
- Les paramètres SYSGEN **PQL_MWSDEFAULT, PQL_MWSQUOTA** et **PQL_MWSEXTENT** sont positionné par AUTOGEN à un minimum de 512 si MMG_CTLFLAGS est <>0

Quotas

- WSQUOTA devient "moins" important : il est moins vital de le positionner correctement en fonction de l'application ou du noeud du cluster
- WSEXTENT = WSMAX : l'Auto-Tuning fera le reste....

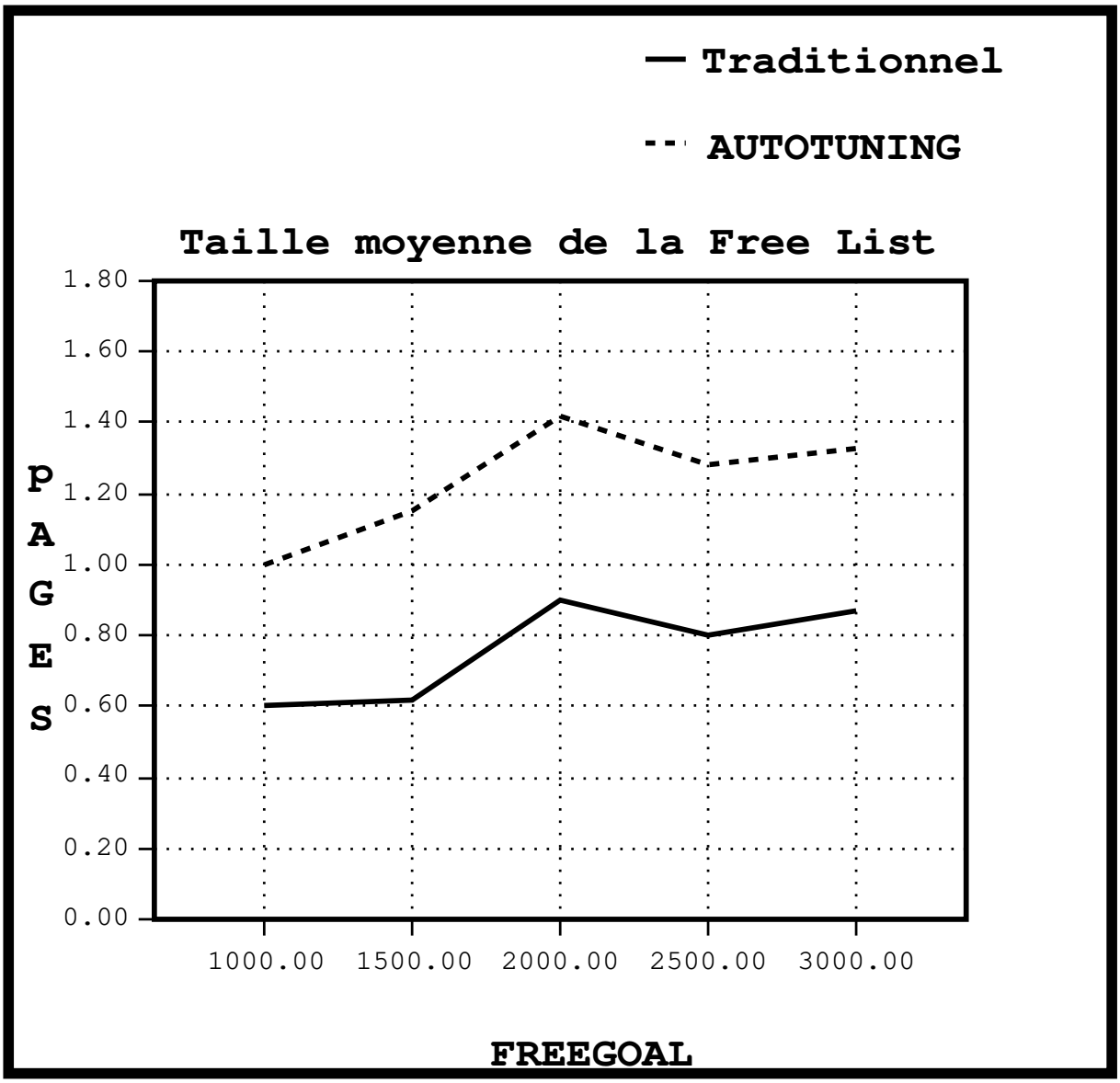
Remarque

- Attention au fait que la mise en marche de ces mécanismes conduit à une augmentation du nombre de *processes swappés*
- Il faut (éventuellement) augmenter la taille du fichier de *swap*

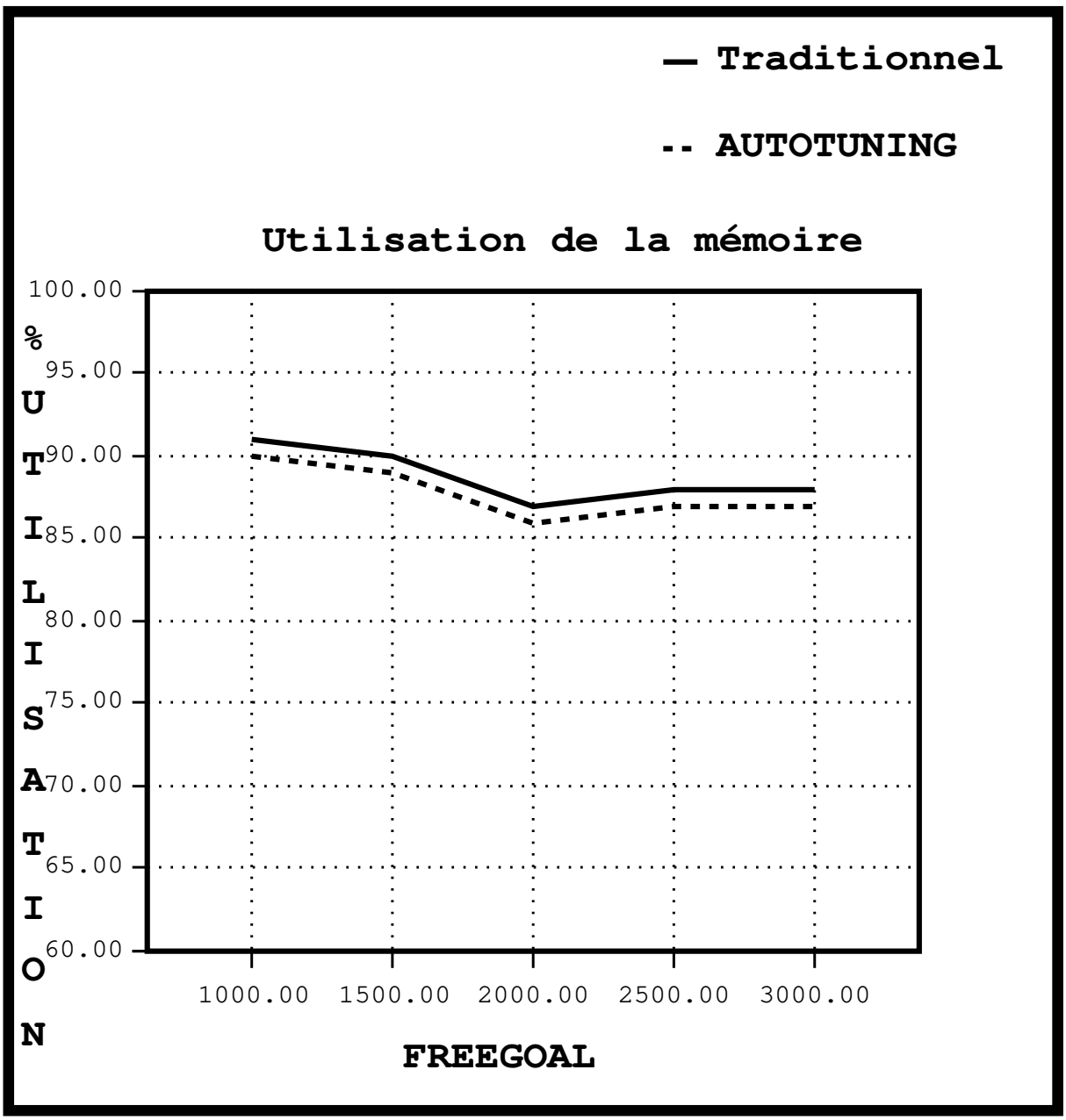
Performances

- Un certain nombre de tests de performance ont été effectués sur un LAVc constitué de :
  - -- un VAX 8700 *boot node* (2 RA60s) avec 192 Mbytes de mémoire, configuré comme noeud DECnet *end-node*
  - -- une VS2000 comme *satellite* (RD54 comme disque de pagination et *swapping*) avec 6 Mbytes de mémoire, configuré comme noeud DECnet *endnode*
  - -- sur un réseau Ethernet séparé
- Les applications choisies pour le test sont les applications standards DECwindows suivantes : Fileview, Calendar, Mail, DECterm et Bookreader

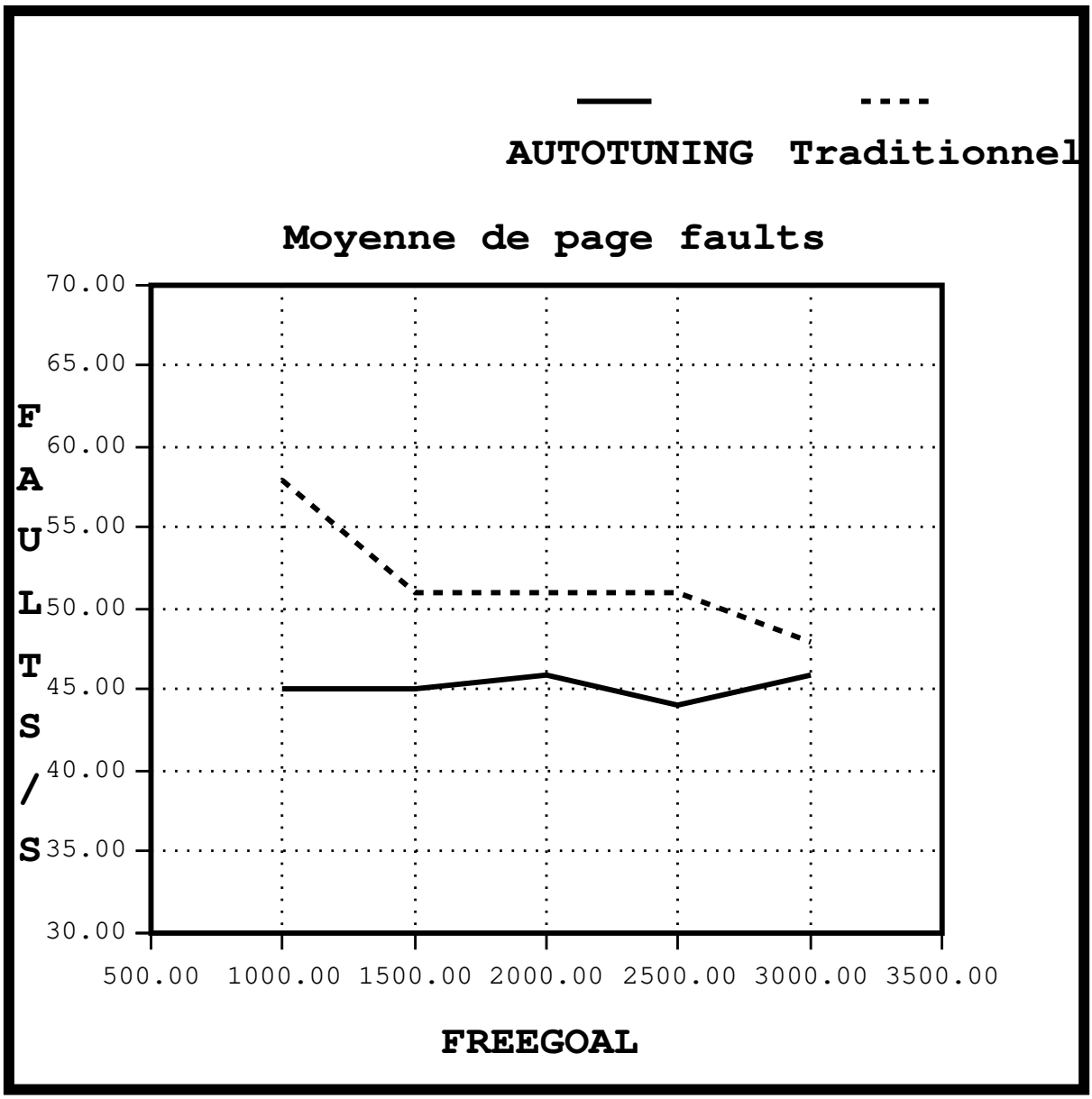
Performances



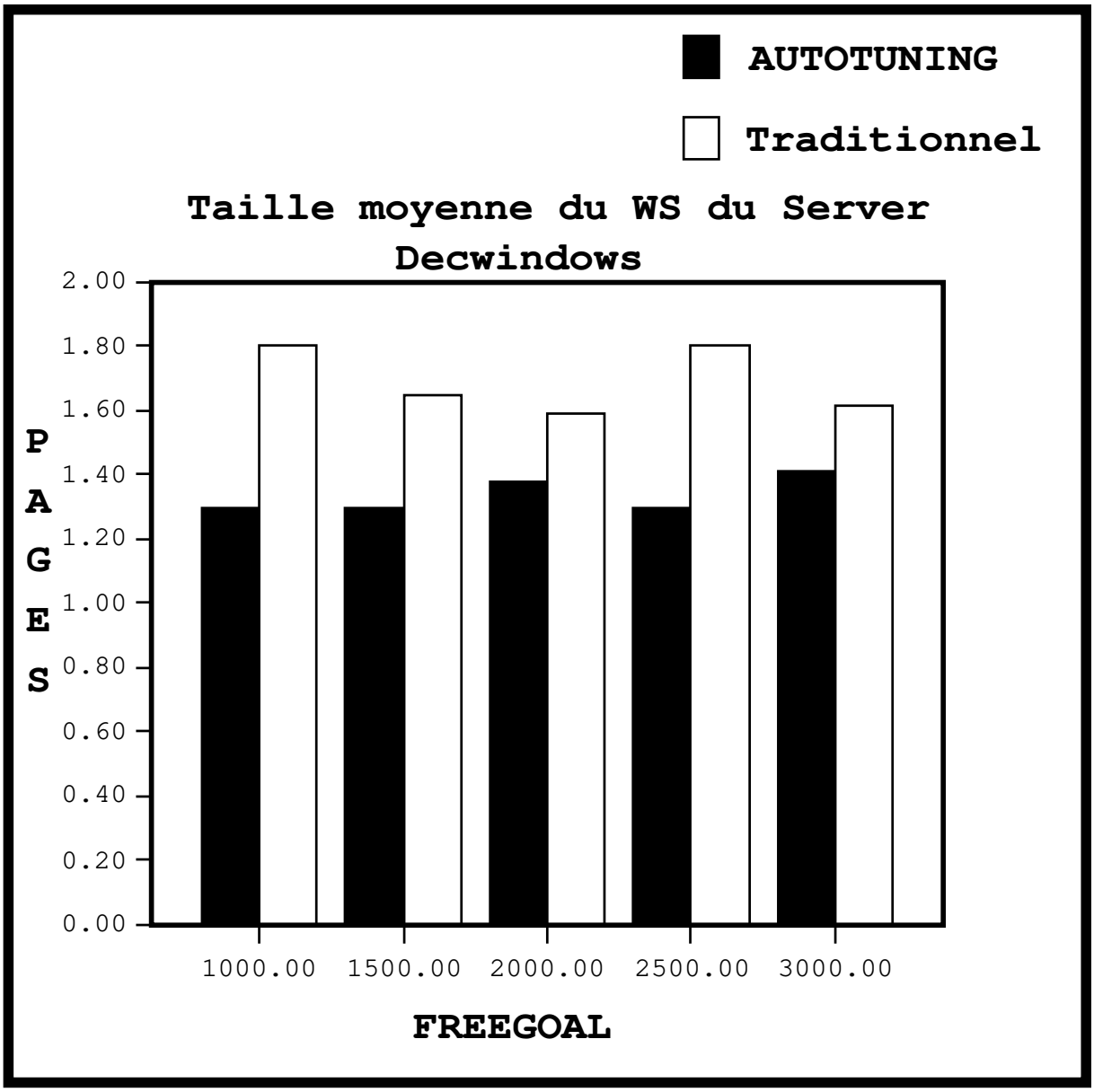
Performances



Performances



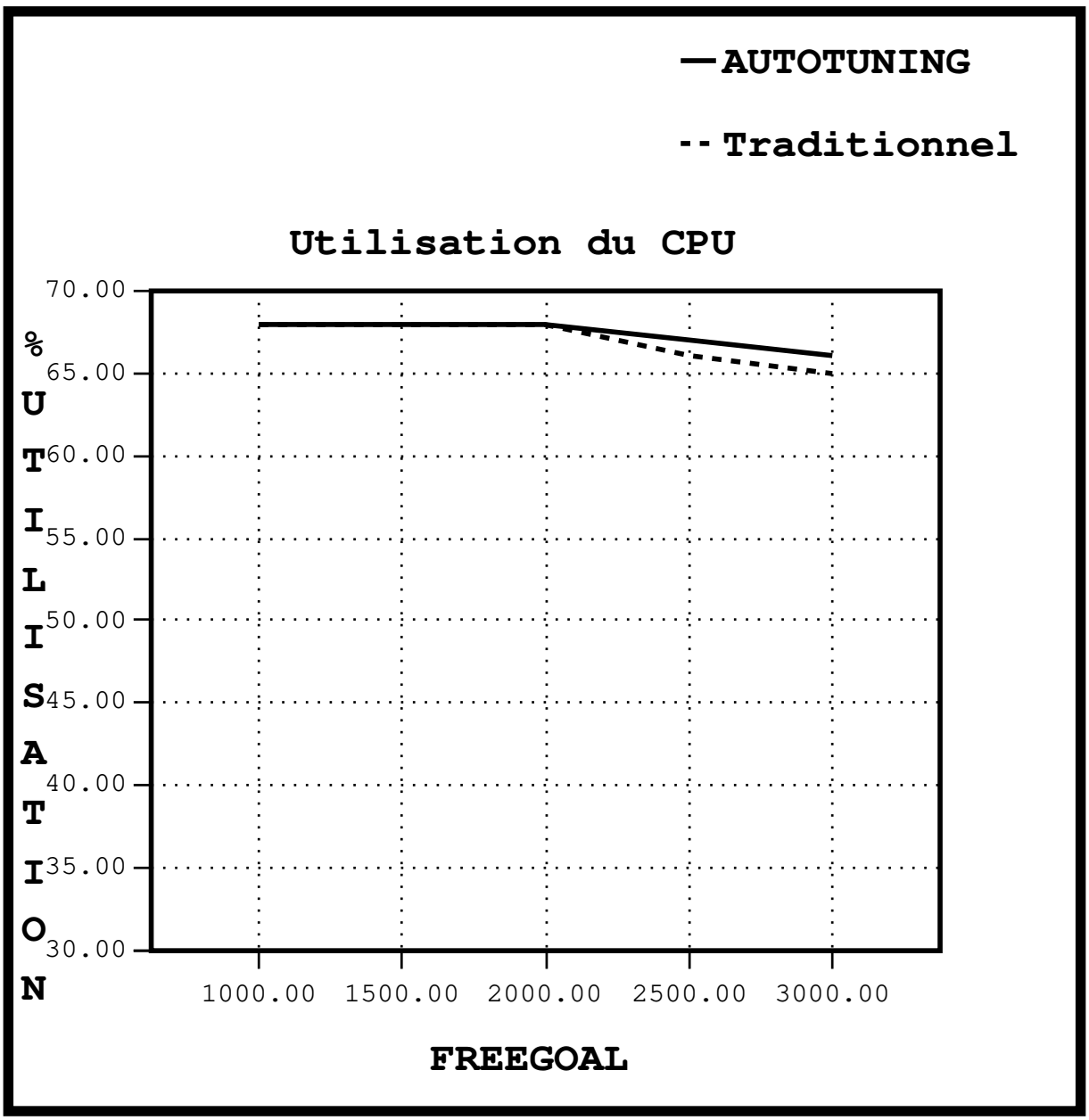
Performances



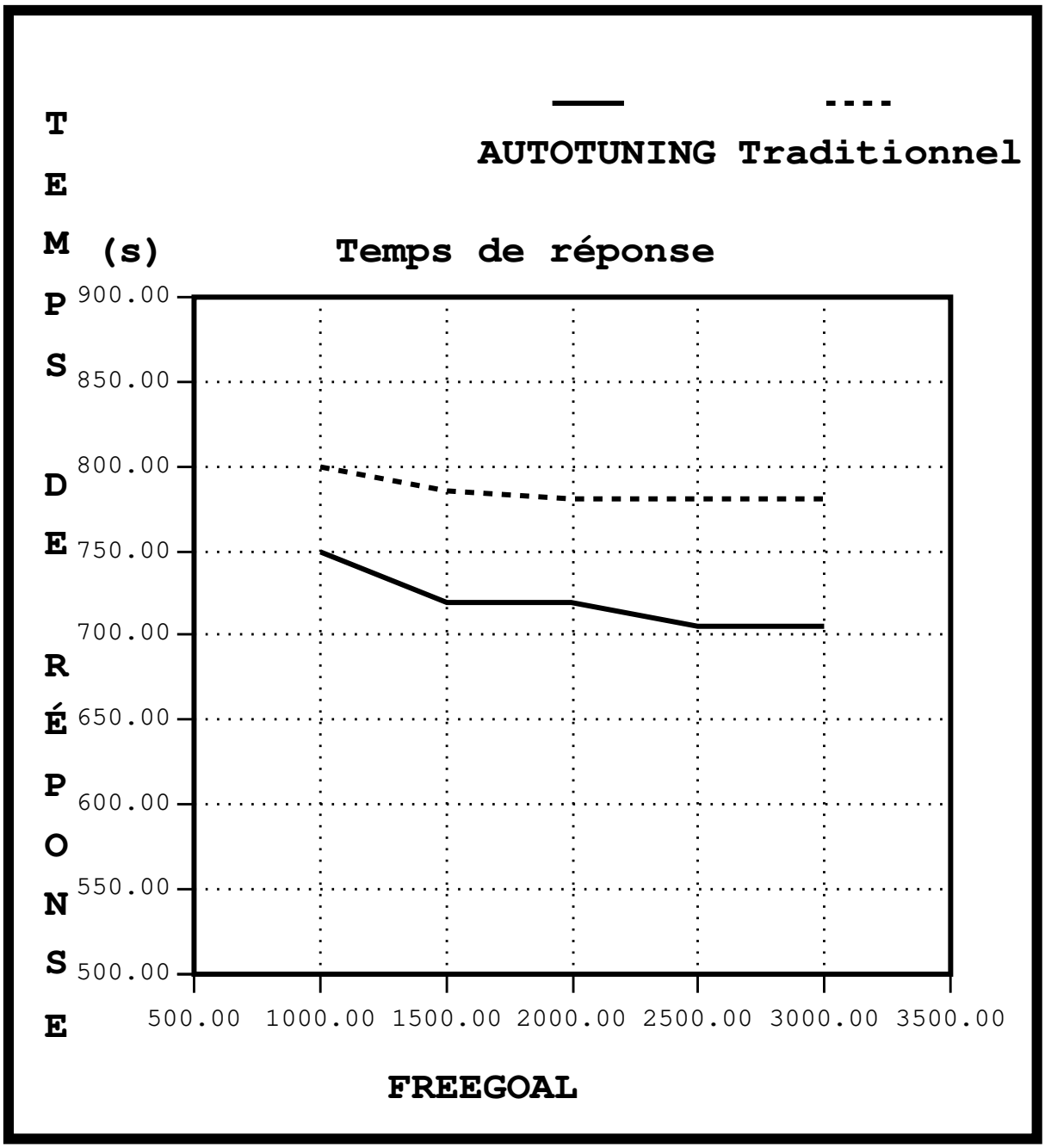
WSDEF=500, WSQUOTA=1300
WSEXTENT=3100

digital

Performances



Performances



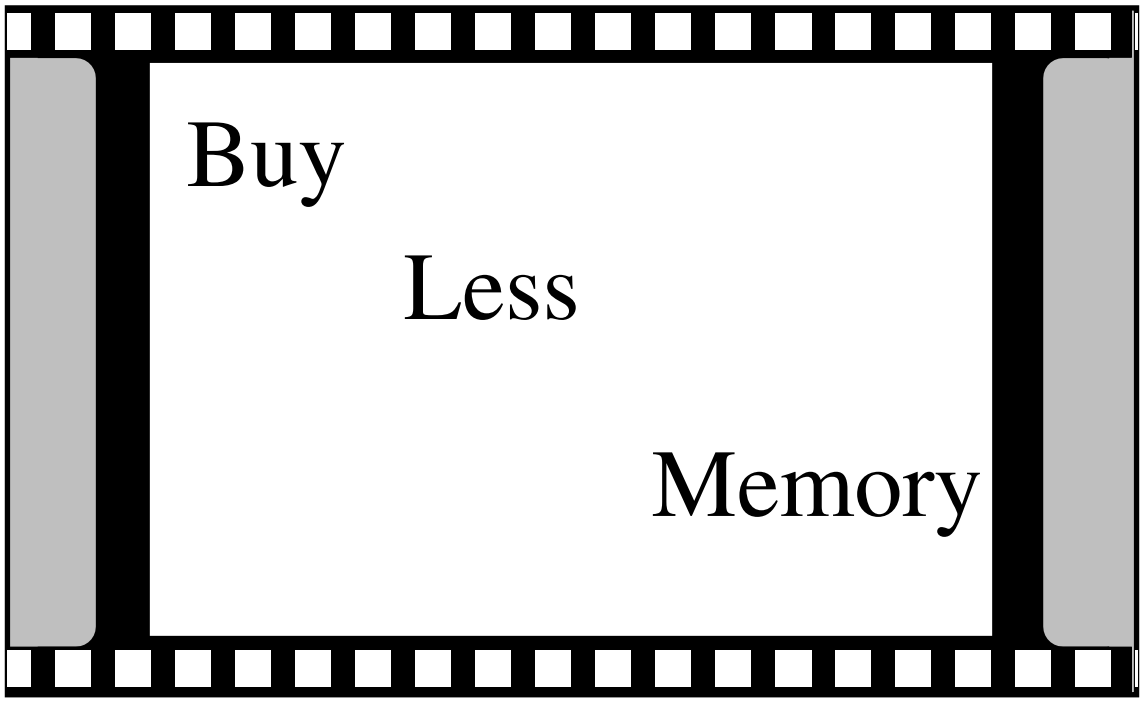
- Le temps de réponse est amélioré de 6.5% à 8.5%
- Le temps de réponse pour une valeur de FREEGOAL de 1000 pour les premières et secondes invocations d'une opération de type *mail_dialog_up* sont de :

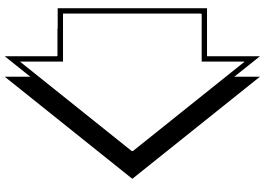
  -- 8.7s et 5.3s sans Auto-Tuning

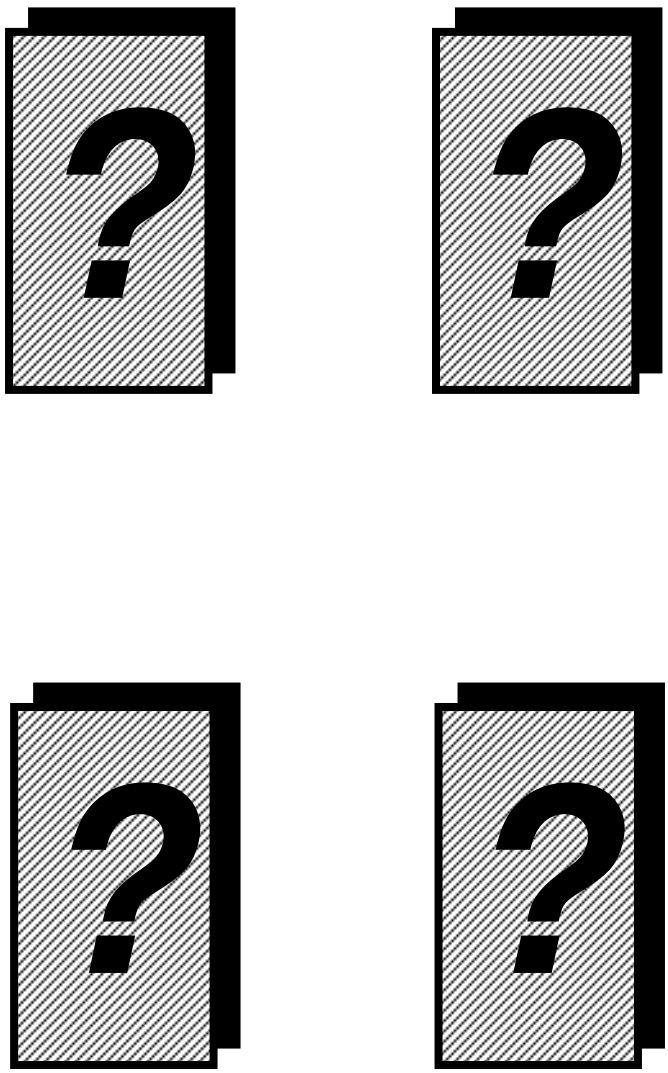
  -- 6.6s et 2.2s avec Auto-Tuning

  soit une amélioration respective de 24% et 59%

Maintenant.....





Maintenant.....

Questions