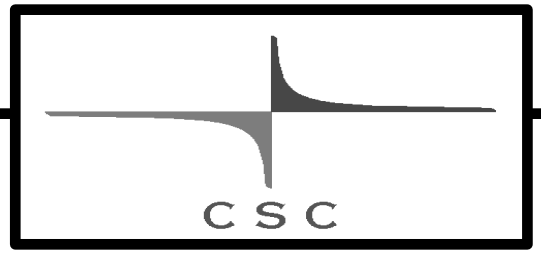
EASYPVM

EnhAnced Subroutine librarY for PVM

by

Sami Saarinen
Center for Scientific Computing
P.O.Box 405
Finland

e–mail: sbs@csc.fi



18–APR–94/sbs

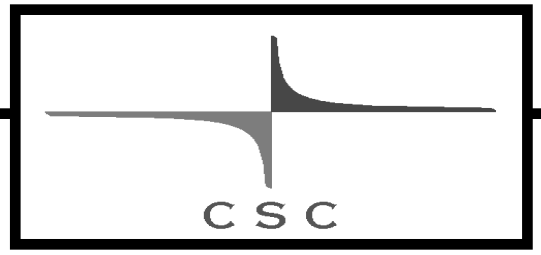
What is EASYPVM ?

– **a simplified message passing interface for PVM programmers who aim to test, port or run MPP–programs on their cluster of workstations**

– **built on the top of PVM 3.2 library**

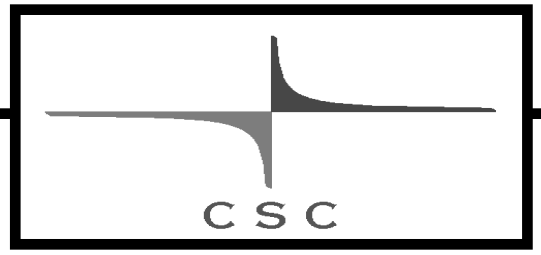
– **being developed at CSC by Sami Saarinen**

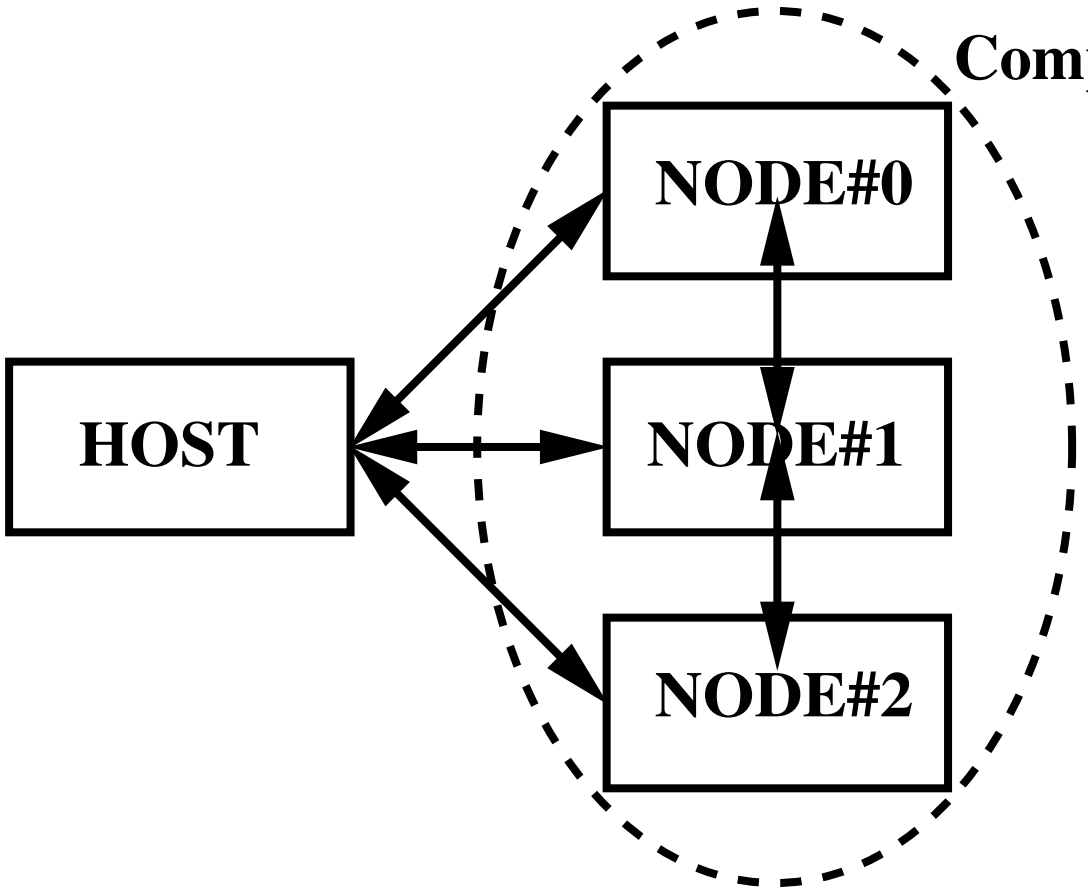
– **current platforms: SGI, SUN4, IBM RS6K, CRAY, CONVEX. Soon: HP PA–RISC & DEC ALPHA**



What extra, over PVM, does the EASYPVM offer ?

- **A simplified process creation and graceful exit**
- **Generic communication routines**
- **Global operations**
- **Broadcast, Multicast, Store&Fwd, SendRecv**
- **Operations with matrix sub–blocks**
- **HOST–NODE or NODE–only (SPMD & non–SPMD)**
- **Performance monitoring via PICL/ParaGraph**
- **Fortran, C & C++ –support**





CSC

18-APR-94/sbs

Process creation **[HOST part]**

- **Routines will take care of all the burden that is associated in plain PVM process initialization**

- **Examples: [HOST–program calls]**

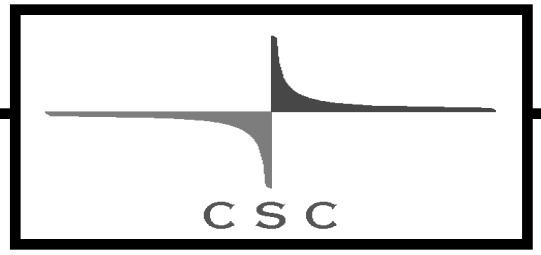
```
CALL CREATEPROC('ndffile')
```

  - **'ndffile' is a Node Definition File**

or

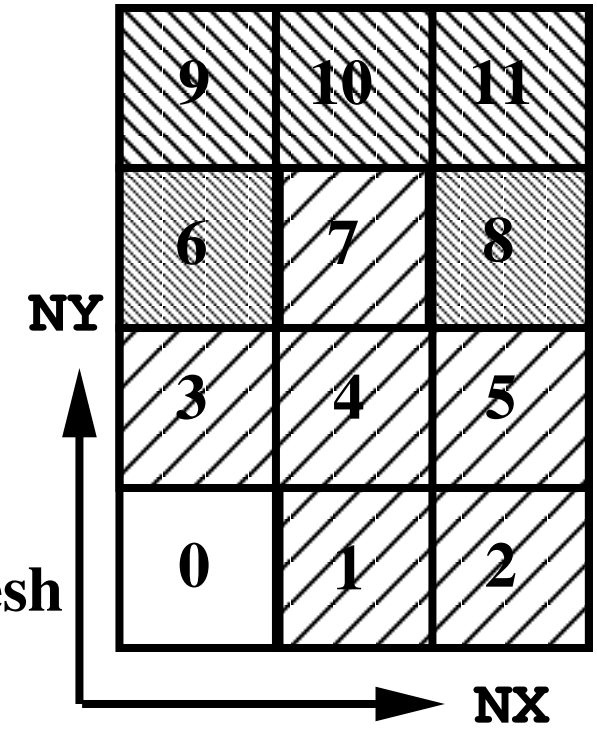
```
CALL LOADPROC('a.out@wstat',10)
```

  - **will start 10 copies of 'a.out' on machine called 'wstat'**



Node definition file (assume 3D-torus)

```
3   4   1   # Total 12 NODEs
0       !hostname node0
1-5,7   cray node1.exe
6,8     SUN4 node2 300 200
9-*     * a.out
```



Meaning:

– Create 12 nodes for 3x4x1 3D–mesh

–#0 @localhost; exename node0

–#1..5 & 7 @cray; node1.exe

–#6 & 8 @any SUN4; node2 + params

–the rest @any machine; exe is a.out

```
NX = 3
NY = 4
NZ = 1
```

CSC

Process attachment [NODE]

– To become a member of a computational cluster, every NODE must identify itself to EASYPVM

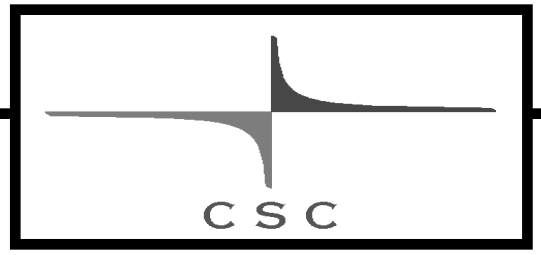
– explicitly: `CALL ATTACHPROC(me,nprocs)`

– implicitly thru some EASYPVM–funcs:

```
me = MYNODE()
```

or

```
nprocs = NUMNODES()
```



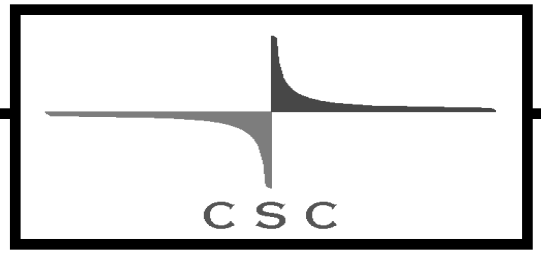
Generic communication routines

- **messages are sent/recv'd by the rank of NODE, not TASKIDs as in PVM**
- **send() and recv() routines operate with ANY predefined type available also in PVM**
- **Data items are accessed by elements, *not bytes*:**

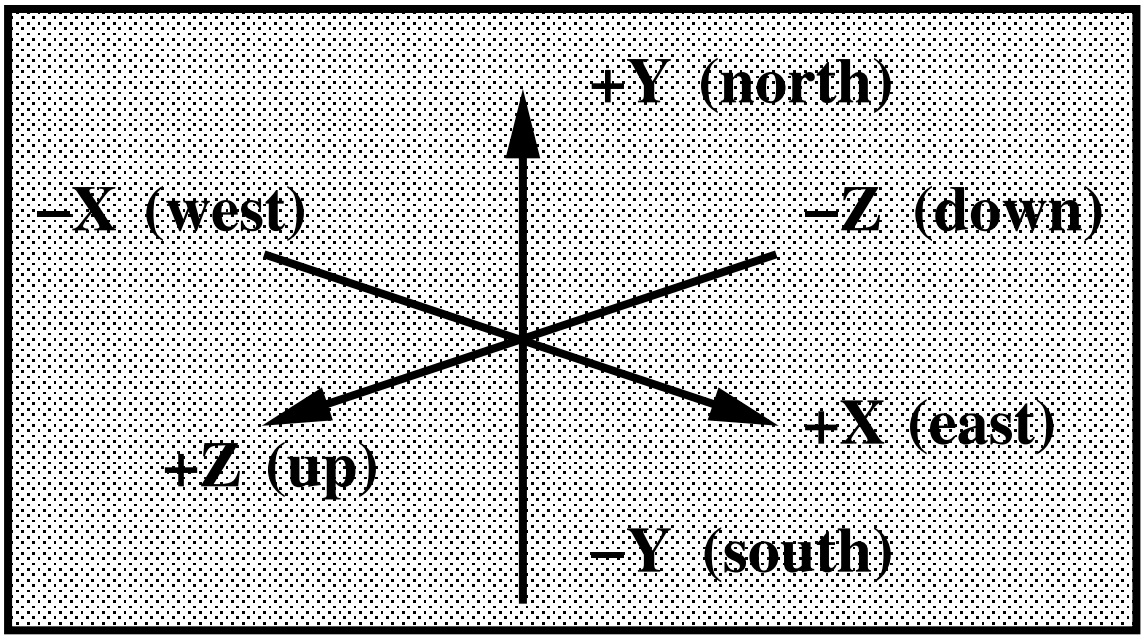
```
REAL*8 DP_TABLE(N)
prevstride = setstride(1)
prevtype  = setdatatype(REAL8)
CALL SEND(to,msgtag,DP_TABLE,N)
CALL RECV(from,msgtag,DP_TABLE,N)
```

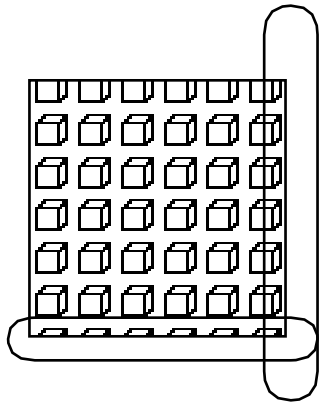
- **wildcards are also available:**

```
CALL SEND(ANYBODY,100,...)
CALL RECV(7,NOCARE,...)
```



Operations on logical 3D–torus





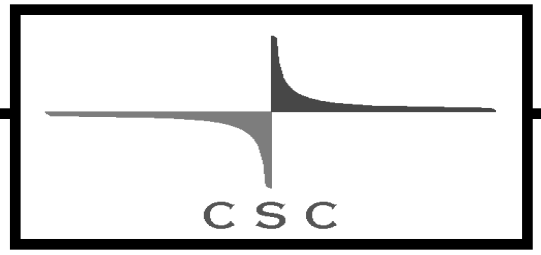
Wrap–around

```
CALL GETPROCDIM(nx,ny,nz)
CALL GETPROCPOS(my_x,my_y,my_z)
CALL SEND(east(),111,my_east,N)
CALL RECV(west(),111,your_east,N)
CALL SETWRAP(x_yes,y_yes,z_no)
```

CSC

Global operations in EASYPVM

– **An important extension to the PVM scope**
– **Imitates INTEL NX–library calls**
– **Always operates *within* NODE–cluster**
– **Routines are *generic* i.e. for any PVM–datatype**
– **syncronization:** `GSYNC()`
– **local vector combine:** `GCOL(..)` **or** `GCOLX(..)`
– **global min (or max):** `GMIN(x,N,y,where)`
– **global sum,prod,dotprod:** `GSUM,GPROD,GDOT`
– **complete exchange:** `GCXCH`

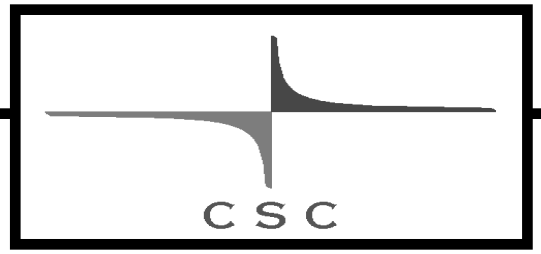


Example#1: **Global summing**

- **3 consecutive processors own one number in sequence {1, 2, 3}, respectively**
- **to calculate sum of these number EACH proc must call:**

```
CALLGSUM(one_number,1,result)
```

- **the 'result' is 6 !**



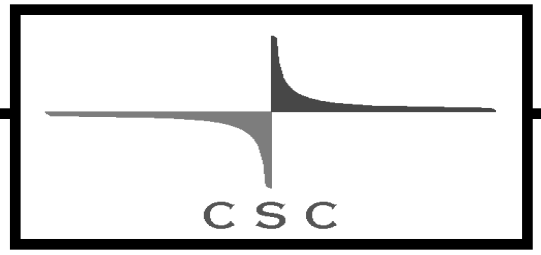
Example#2: **Global concatenation (combine)**

- **proc#0 owns two numbers x={1,2} and proc#1 owns three numbers x={3,4,5}**
- **After global concatenation EACH of them will have vector v={1,2,3,4,5}, in this order**
- **Code:**

```
INTEGER x(my_xlen), v(5)
CALL GCOL(x,my_xlen,v,5)
```

or

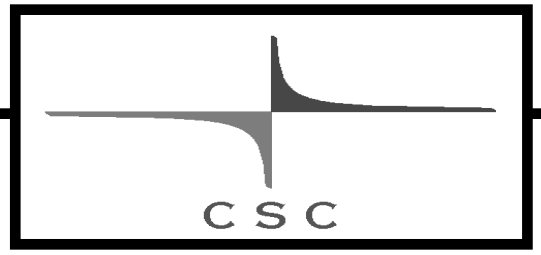
```
INTEGER nlens(0:nproc-1)
data nlens /2, 3/
CALL GCOLX(x,nlens,v)
```

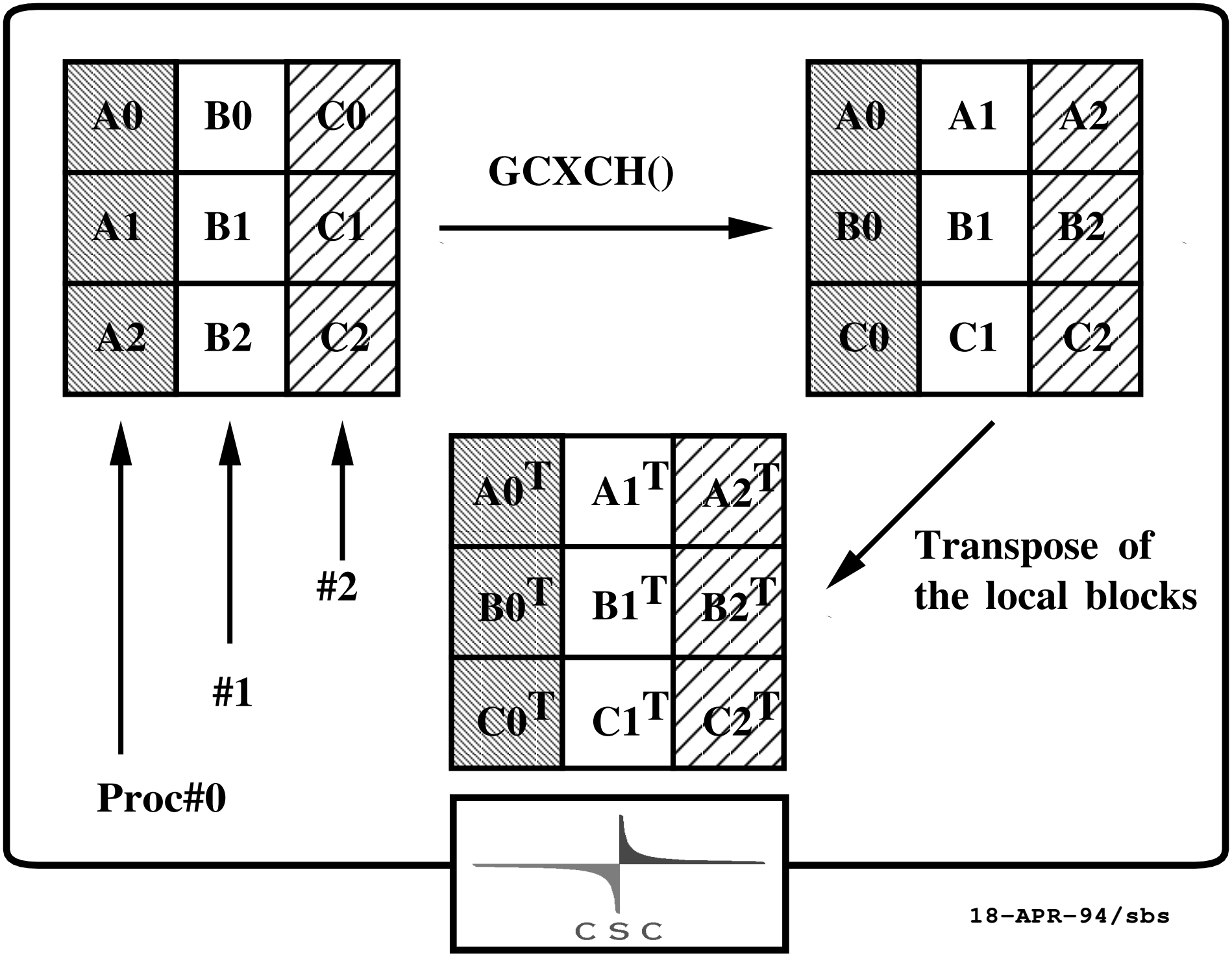


Example#3: **Matrix transpose**

- **Each proc owns 'nproc' `rxr` –matrix subblocks**
- **Using complete exchange, the matrix transpose is a "straight forward" operation:**

```
      REAL*8 A(r,r,nproc),ATR(r,r,nproc)
      CALL GCXCH(A,r*r,ATR)
      do 100 k=1,nproc
         do 100 j=1,r
            do 100 i=j+1,r
               swap = ATR(i,j,k)
               ATR(i,j,k) = ATR(j,i,k)
100            ATR(j,i,k) = swap
```

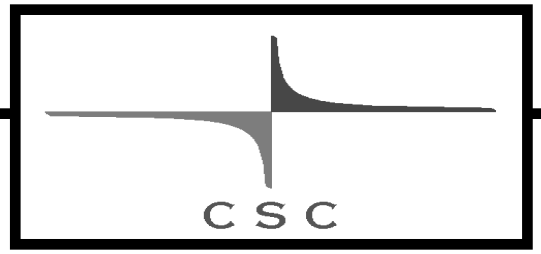


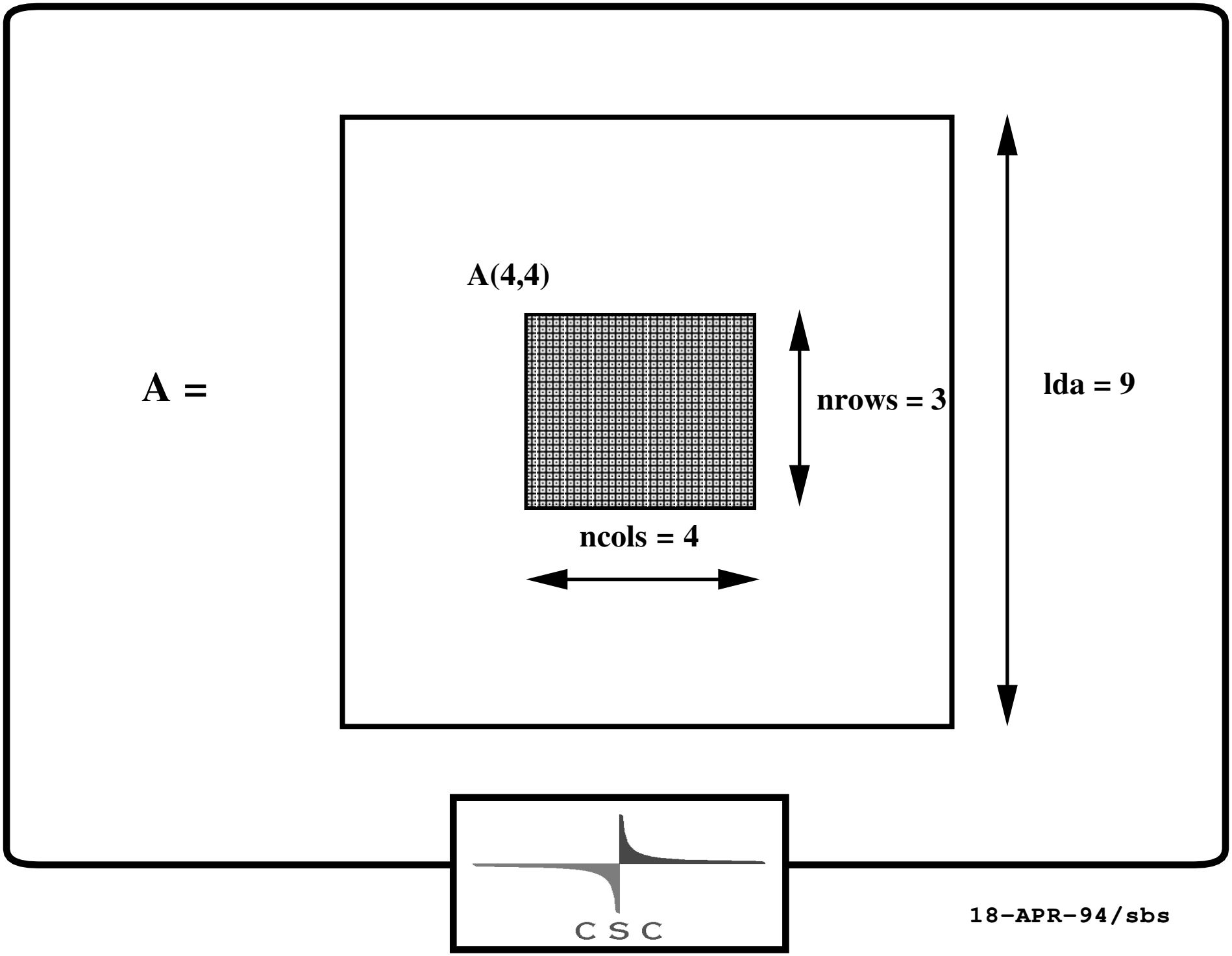

A0
B0
C0
A1
B1
C1
A2
B2
C2
GCXCH()
A0
A1
A2
B0
B1
B2
C0
C1
C2
#2
#1
Proc#0
A0T
A1T
A2T
B0T
B1T
B2T
C0T
C1T
C2T
Transpose of
the local blocks
CSC
18-APR-94/sbs

Operations on matrix sub–blocks

– Communication routines can access the matrix sub–blocks directly
– Save lots of coding burden and are reliable
–Example: recv() 3x3 sub–block starting at A(4,4):

```
REAL*4  A(9,9)
nrows = 3
ncols = 4
lda = 9
CALL RECV2D(2,100,A(4,4),nrows,ncols,lda)
```





18-APR-94/sbs

Miscellaneous communication routines

- **broadcast: within NODE–cluster**

```
CALL BROADCAST(msgtag,x,N)
```

- **multicast: within HOST and/or some NODEs**

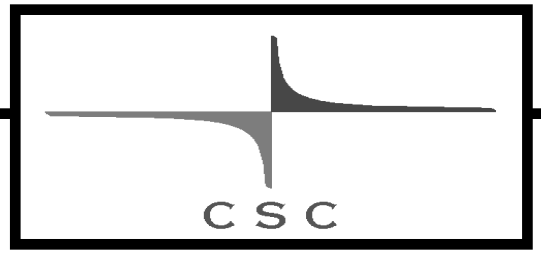
```
CALL MULTICAST(dest,ndest,msgtag,x,N)
```

- **store–and–forward: [Combined recv() + send()]**

```
CALL STOREFWD(...)
```

- **Combined send() + recv():**

```
CALL SENDRECV(to,10,x,N,from,20,y,M)
```



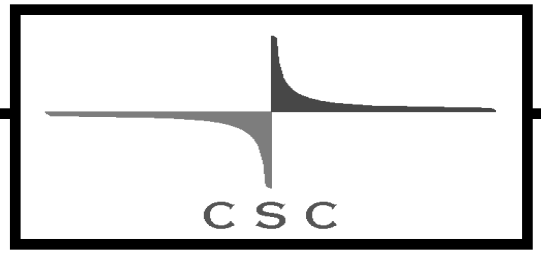
Compilers

– **Take care of compilation and inclusion of the correct header files and PVM object libraries**

– **Usage:**

Fortran: `% pvmf77 prog.f -o prog`

C: `% pvmcc prog.c -o prog`



Running applications manually

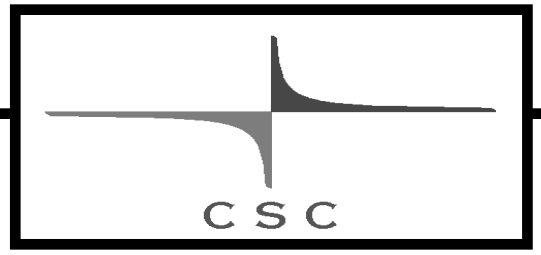
1. **Create PVM hostfile and name the machines etc. or use defaults for local machine**

2. **Start PVM–daemons so that ALL machines are up and running before start of the application:**

```
% pvminit [my_hostfile]
```

3. **Decide which approach to use**
   - **HOST – NODE or**
   - **NODE –only**

(cont'd)



Running applications manually **(cont.)**

4. HOST – NODE:

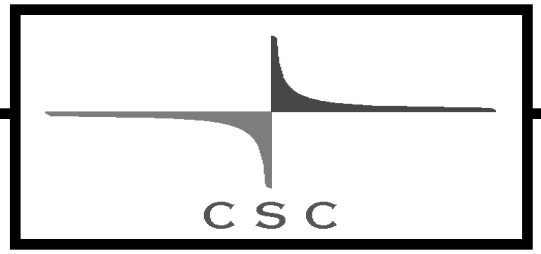
```
% my_host  [params]
```

5. NODE – only:

```
% pvmserver -n4 my_node [params]
```

6. Clean up the session and kill PVM–daemons:

```
% pvmclean
```



Running applications automatically

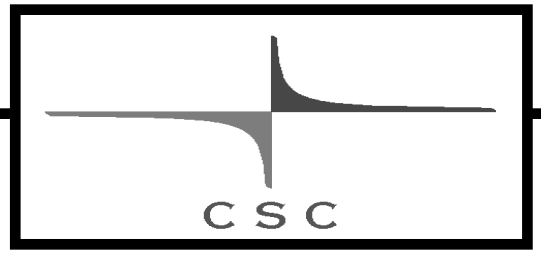
– Scripts '`pvmrun`' & '`pvmfork`' take care of initialization, run and clean up

– HOST–NODE:

```
% pvmrun [-h hostfile] my_host[params]
```

– NODE–only: [two main alternatives]

```
% pvmfork -n4 my_node [params]
% pvmfork -f ndffile
```



Performance tracing & monitoring

– **EASYPVM is enable to produce PICL–tracefiles for NODE–processes for use by *ParaGraph***

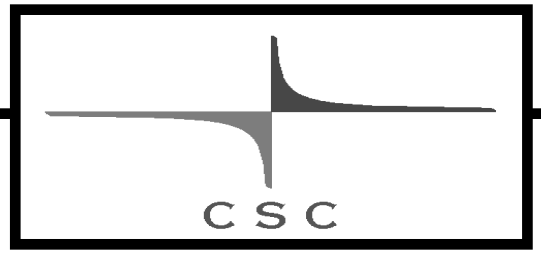
– **Activated via '–t' flag in 'pvmrun' or 'pvmfork':**

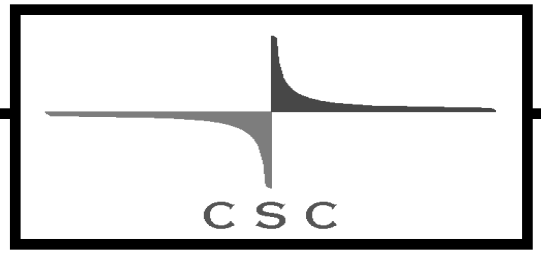
```
% pvmfork -t -n9 nodeprog
```

– **"Permanent" tracefile generation via env–param:**

```
% setenv EASY_PICLTRACE
```

– **Location of the tracefiles after run:**
machine:$HOME/picl/<hhmmss>/node–*



How to obtain the EASYPVM ?

– **EASYPVM is a *public domain* product available at anonymous `ftp.funet.fi` (`128.214.248.6`) under directory `/pub/csc/software/easypvm`**