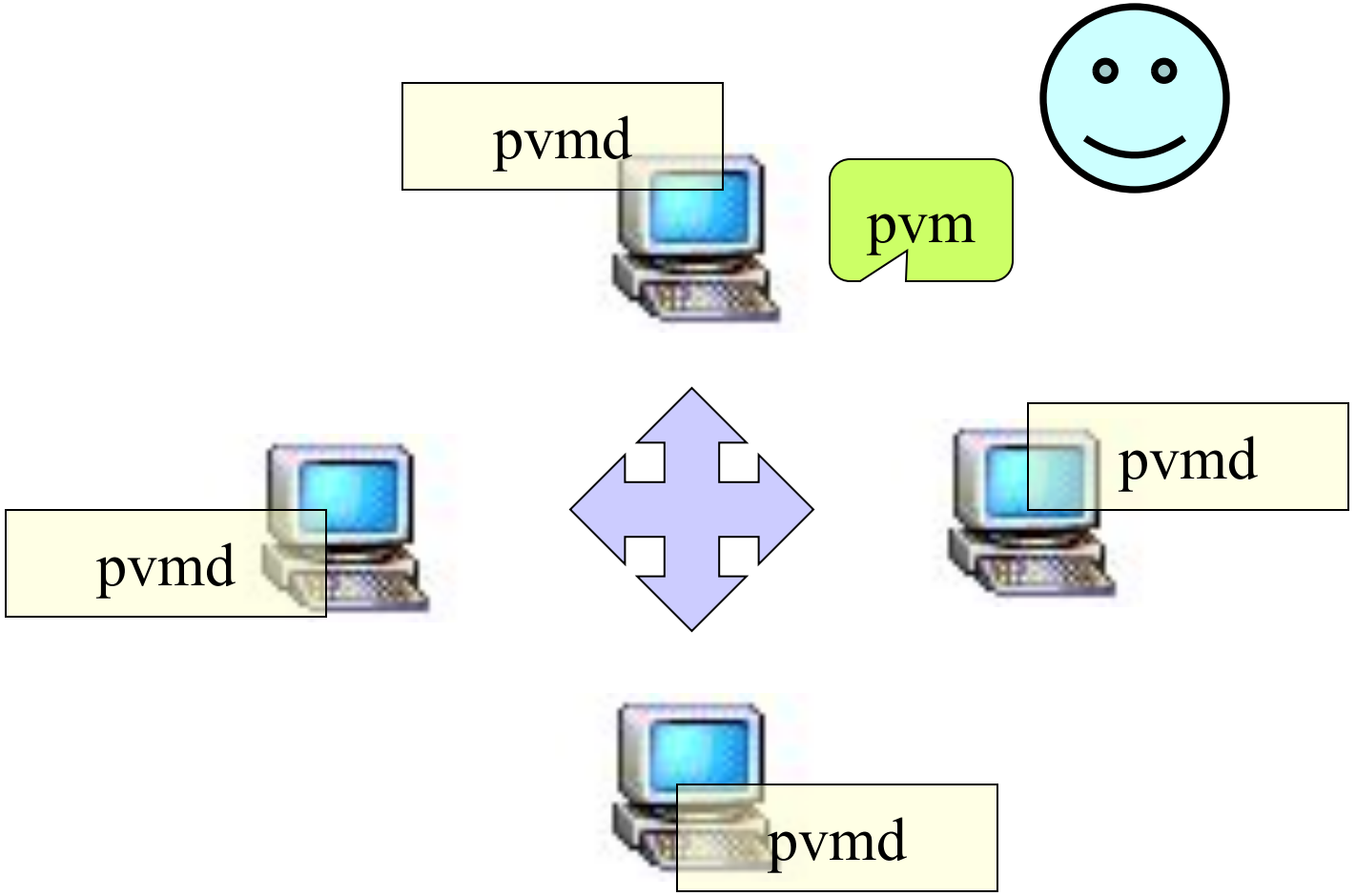
Parallel Virtual Machines

История создания

Проект PVM был начат в 1989 году в Oak Ridge National Laboratory
Первый релиз – Март 1991.

Библиотека была полностью переписана в 1993 году. Версия 3.3, которая и будет рассматриваться далее.

Парадигма PVM



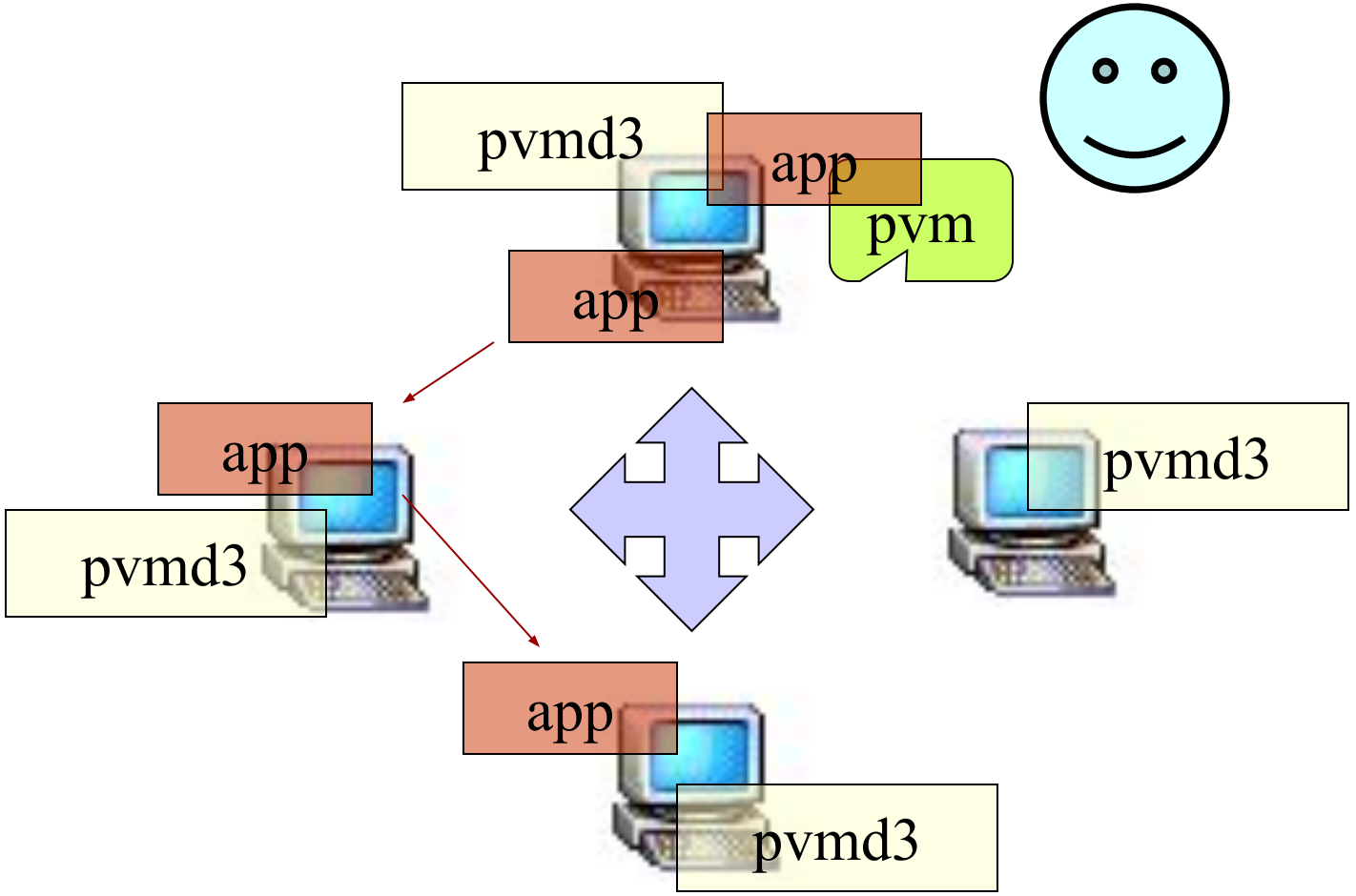
«Демон» pvmd

- «Отвечает» за работу на вычислительном узле.
- Используется для запуска, контроля и завершения заданий.

Работа пользователя в PVM

- Запуск PVM.
- Создание виртуальной машины.
- Запуск процесса приложения, который порождает другие процессы.

Запуск приложения



Консоль PVM

Консоль PVM предоставляет возможность контроля всех процессов виртуальной машины. Она используется для

- конфигурирования параллельной машины;
- запуска и завершения процессов виртуальной машины;
- получения информации о запущенных процессах;

Консоль является отдельным процессом PVM, может быть запущена или завершена в любой момент.

Управление виртуальной машиной с консоли

pvm> *add hostname* добавление узла в виртуальную машину

pvm> *delete hostname* удаление узла из виртуальной машины

pvm> *conf* печать конфигурации виртуальной машины

pvm> *halt* завершить работу всех демонов и закрыть PVM

Работа с процессами

pvm> *spawn task* запуск процесса с именем *task*

pvm> *kill tid* прерывание выполнения задания

pvm> *ps* печать списка запущенных процессов

pvm> *reset* завершение всех запущенных задач

Другие команды консоли

pvm> *spawn task* запуск процесса с именем *task*

pvm> *kill tid* прерывание выполнения задания

pvm> *ps* печать списка запущенных процессов

Способы запуска приложения

- Из командной строки.
- Из консоли **pvm** на локальной или удаленной машине.
- Из приложения с помощью функции **pvm_spawn** на локальной или удаленной машине.

Управление виртуальной машиной из консоли

- На машине должна быть установлена библиотека PVM.
- Должны существовать права для удаленного доступа.
- pvm add hostname

Управление виртуальной машиной из программы

int pvm_add_hosts(char hosts, int nhosts, int* infos);**

hosts – список имен добавляемых машин

nhosts – число добавляемых машин

infos – коды ошибок (< 0 означает ошибку)

возвращает число корректно добавленных узлов

int pvm_del_hosts(char hosts, int nhosts, int* infos);**

Запуск процесса из консоли

- Предварительно скопировать в стандартный (или указанный для данной машины) каталог на нужной машине.
- spawn app_name – обычный запуск (вывод сохраняется в директории tmp)
- spawn -> app_name – запуск с перехватом вывода.

Запуск процесса

int numt = pvm_spawn(char *task, char **argv, int flag,
char *where, int ntask, int *tids)

Запуск **ntask** процессов с именем **task**.

task – имя загрузочного модуля
argv – аргументы командной строки
flag – способ запуска
PvmTaskDefault (0) - PVM выбирает машину
PvmTaskHost - **where** задает конкретную машину
…
tids – массив идентификаторов запущенных заданий (или кодов ошибок при частичном запуске).

numt – число реально запущенных заданий, значение меньше ноля означает ошибку; если **numt < ntask**, то запущено меньше заданий, чем ожидалось (ошибки в **tids**)

Различные примеры запуска процессов

```
numt = pvm_spawn( "host", 0, PvmTaskHost,
                  "sparky", 1, &tid[0] );
numt = pvm_spawn( "host", 0, (PvmTaskHost+PvmTaskDebug),
                  "sparky", 1, &tid[0] );
numt = pvm_spawn( "node", 0, PvmTaskArch,
                  "RIOS", 1, &tid[i] );
numt = pvm_spawn( "FEM1", args, 0, 0, 16, tids );
numt = pvm_spawn( "pde", 0, PvmTaskHost,
                  "paragon.ornl", 512, tids );
```

Завершение задания

Выход из параллельной машины:

```
int pvm_exit();
```

стандартная последовательность вызова:

```
pvm_exit();
exit();
```

Принудительное завершение процесса по идентификатору:

```
int pvm_kill(int tid);
```

Пример (hello.c)

```
main()
{
   int cc, tid, msgtag;
   char buf[100];

   printf("i'm t%x\n", pvm_mytid());

   cc = pvm_spawn("hello_other", (char**)0, 0, "", 1, &tid);
```

```
if (cc == 1) {
      msgtag = 1;
      pvm_recv(tid, msgtag);
      pvm_upkstr(buf);
      printf("from t%x: %s\n", tid, buf);
   } else
      printf("can't start hello_other\n");

   pvm_exit();
}
```

Пример (hello_other.c)

```
#include "pvm3.h"

main()
{
   int ptid, msgtag;
   char buf[100];

   ptid = pvm_parent();
```

```
    strcpy(buf, "hello, world from ");
    gethostname(buf + strlen(buf), 64);
     msgtag = 1;
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, msgtag);

    pvm_exit();
}
```

int tid = pvm_mytid(void)

Возвращает номер задания. Если процесс не включен
в PVM, то добавляет его (как и любой другой первый
вызов PVM).

int tid = pvm_parent(void)

Возвращает идентификатор процесса, запустившего
данный или PvmNoParent, если такого нет.

Обмен сообщениями

Посылка:

1. Инициализация буфера.
2. Упаковка в буфер данных.
3. Посылка данных.

Прием:

1. Прием данных в буфер.
2. Распаковка данных из буфера.

Посылка сообщений.

int bufid = pvm_initsend(int encoding)

Очищает буфер посылки сообщений и инициализирует кодировку.

encoding:

PvmDataDefault – кодировка XDR (самый общий случай)

PvmDataRaw – кодировка отсутствует (однородная платформа)

PvmDataInPlace – копирования не производится (сохраняются
указатели и размеры данных пользователя)

Управление несколькими буферами

PVM допускает существование нескольких буферов, из которых в данный момент может быть активен только **один** буфер для отправки и в точности **один** буфер для приема сообщений. Активный буфер можно менять.

Функции для управления несколькими буферами

int pvm_mkbuf(int encoding) – создает буфер возвращает идентификатор созданного буфера

int pvm_freebuf(int bufid) – освобождает буфер

int pvm_getsbuf() – возвращает активный буфер для отправки сообщений

int pvm_getrvuf() – возвращает активный буфер для приема сообщений

int pvm_setsbuf(int bufid) – устанавливает активный буфер для отправки сообщений (возвращает id предыдущего буфера)

int pvm_setrbuf(int bufid) – устанавливает активный буфер для приема сообщений (возвращает id предыдущего буфера)

Возможное применение нескольких буферов

За счет переключения между буферами удается передавать сообщения без перекодировки:

```
bufid = pvm_recv( src, tag );
oldid = pvm_setsbuf( bufid );
info = pvm_send( dst, tag );
info = pvm_freebuf( oldid );
```

Упаковка сообщений

int info = pvm_pkbyte(char *cp, int nitem, int stride)
int info = pvm_pkcplx(float *xp, int nitem, int stride)
int info = pvm_pkdcplx(double *zp, int nitem, int stride)
int info = pvm_pkdouble(double *dp, int nitem, int stride)
int info = pvm_pkfloat(float *fp, int nitem, int stride)
int info = pvm_pkint(int *np, int nitem, int stride)
int info = pvm_pklong(long *np, int nitem, int stride)
int info = pvm_pkshort(short *np, int nitem, int stride)
int info = pvm_pkstr(char *cp)
int info = pvm_packf(const char *fmt, ...)

Посылка сообщений

int info = pvm_send(int tid, int msgtag)

tid – идентификатор процесса приемника
msgtag – тэг сообщения

int info = pvm_mcast(int *tids, int ntask, int msgtag)

Посылка асинхронная.

Посылка сообщений

int info = pvm_psend(int tid, void* p, int msgtag,
int cnt, int typ)

tid – идентификатор процесса приемника;
msgtag – тэг сообщения;
cnt – число посылаемых элементов данных;
typ – тип элементов данных;

Типы

- PVM_STR
- PVM_BYTE
- PVM_SHORT
- PVM_INT
- PVM_LONG
- PVM_ULONG
- PVM_FLOAT
- PVM_CPLX
- PVM_DOUBLE
- PVM_DCPLX
- PVM_UINT
- PVM_USHORT

Прием сообщений

int bufid = pvm_recv(int tid, int msgtag)

tid – идентификатор процесса отправителя
msgtag – тэг сообщения

(Значение тэга или идентификатора сообщения, равное –1, воспринимается как шаблон.)
Блокирующий прием сообщения. Освобождает старый (если последний
не был сохранен с помощью pvm_setrbuf) и создает новый активный буфер, который заполняет принятым сообщением и возвращает его.

Прием сообщений

int bufid = pvm_trecv(int tid, int msgtag, struct timeval* timeout)

Прием по таймауту.
tid – идентификатор процесса отправителя
msgtag – тэг сообщения
timeout – максимальное время блокировки
возвращает 0, если сообщение не доставлено после прошествия
периода времени

int bufid = pvm_nrecv(int tid, int msgtag)

Неблокирующий прием – возвращает 0 в случае
если сообщение не пришло неготовности отправителя

Распаковка

int info = pvm_upkbyte(char *cp, int nitem, int stride)
int info = pvm_upkcplx(float *xp, int nitem, int stride)
int info = pvm_upkdcplx(double *zp, int nitem, int stride)
int info = pvm_upkdouble(double *dp, int nitem, int stride)
int info = pvm_upkfloat(float *fp, int nitem, int stride)
int info = pvm_upkint(int *np, int nitem, int stride)
int info = pvm_upklong(long *np, int nitem, int stride)
int info = pvm_upkshort(short *np, int nitem, int stride)
int info = pvm_upkstr(char *cp)

int info = pvm_unpackf(const char *fmt, ...)

Прием + распаковка

int pvm_precv(int tid, int msgtag, void *p, int cnt, int typ, int * rtid, int* rcnt, int * rtag)

Комбинирует прием и распаковку сообщения, состоящего из
однородных элементов. Возвращает (через параметры rtid, rcnt,
rtyp) информацию о числе элементов в принятом сообщении,
идентификаторе процесса-передатчика и тэге сообщения.

Нотификация о событии

int pvm_notify(int what, int msgtag, int cnt, int* tids)

what – тип события (**PvmTaskExit**, **PvmHostDelete**, **PvmHostAdd**)

msgtag – тэг, который будет использован для нотификации

cnt – число элементов в массиве **tids**

tids – массив идентификаторов процессов, события которых подвергаются мониторингу (пуст, если **PvmHostAdd**)

Нотификация о событии

При возникновении запрашиваемого события процессу, вызывавшему **pvm_notify**, посылается сообщение с указанным тэгом.

Сообщение содержит id процесса, для которого произошло событие (одно сообщение на событие):

PvmTaskExit – id процесса, завершившего работу;

PvmHostDelete – id удаленного pvmd;

PvmHostAdd – id добавленного pvmd (не более cnt сообщений).

```
/*
    Failure notification example
    Demonstrates how to tell when a task exits
*/

/* defines and prototypes for the PVM library */
#include <pvm3.h>

/* Maximum number of children this program will spawn */
#define MAXNCHILD   20
/* Tag to use for the task done message */
#define TASKDIED        11


int
main(int argc, char* argv[])
{

    /* number of tasks to spawn, use 3 as the default */
    int ntask = 3;
    /* return code from pvm calls */
    int info;
    /* my task id */
    int mytid;
    /* my parents task id */
    int myparent;
    /* children task id array */
    int child[MAXNCHILD];
    int i, deadtid;
    int tid;
    char *argv[5];
```

```
/* find out my task id number */
mytid = pvm_mytid();

/* check for error */
if (mytid < 0) {
    /* print out the error */
    pvm_perror(argv[0]);
    /* exit the program */
    return -1;
    }
/* find my parent's task id number */
myparent = pvm_parent();


/* exit if there is some error other than PvmNoParent */
if ((myparent < 0) && (myparent != PvmNoParent)) {
    pvm_perror(argv[0]);
    pvm_exit();
    return -1;
    }
```

```
/* if i don't have a parent then i am the parent */
if (myparent == PvmNoParent) {
    /* find out how many tasks to spawn */
    if (argc == 2) ntask = atoi(argv[1]);
    /* make sure ntask is legal */
    if ((ntask < 1) || (ntask > MAXNCHILD)) { pvm_exit(); return 0; }

    /* spawn the child tasks */
    info = pvm_spawn(argv[0], (char**)0, PvmTaskDebug, (char*)0,
        ntask, child);

    /* make sure spawn succeeded */
    if (info != ntask) { pvm_exit(); return -1; }

    /* print the tids */
    for (i = 0; i < ntask; i++) printf("t%x\t",child[i]); putchar('\n');
```

```
        /* ask for notification when child exits */
        info = pvm_notify(PvmTaskExit, TASKDIED, ntask, child);
        if (info < 0) { pvm_perror("notify"); pvm_exit(); return -1; }

        /* reap the middle child */
        info = pvm_kill(child[ntask/2]);
        if (info < 0) { pvm_perror("kill"); pvm_exit(); return -1; }

        /* wait for the notification */
        info = pvm_recv(-1, TASKDIED);
        if (info < 0) { pvm_perror("recv"); pvm_exit(); return -1; }
        info = pvm_upkint(&deadtid, 1, 1);
        if (info < 0) pvm_perror("calling pvm_upkint");

        /* should be the middle child */
        printf("Task t%x has exited.\n", deadtid);
        printf("Task t%x is middle child.\n", child[ntask/2]);
        pvm_exit();
        return 0;
        }

    /* i'm a child */
    sleep(63);
    pvm_exit();
    return 0;
}
```