March 11, 1983
T. Malloy

Specification for UnitHz - The Standard Storage Manager

Overview

UnitHz provides a client with routines to manage a piece of contiguous memory, called a heap zone (hz). In the case of Lisa, this is usually but not necessarily an MMU segment. UnitHz will manage an arbitrary number of such zones. The zone is made up of a zone header (ahz) followed by an arbitrary number of storage blocks (abk). The zone may be arbitrarily large.

A storage block consists of a header followed by usable memory. Blocks are always of even length within a specified minimum and maximum. The current minimum is 12 bytes, the length of a minimum free block. This could be reduced to 4 bytes if desired by recoding portions of UnitHz. The maximum size of an allocated block is currently 32K bytes. This could be increased to 65K bytes with minor enhancements or to 2^{31} by making the appropriate quantities a full 32 bits.

There are four types of storage blocks. They are free, non-relocatable, relocatable and named.

Free blocks are used by the implementation and not visible to the client. They are kept on a doubly linked list through fields in the free block header. Free blocks may be as large the entire zone.

Non-relocatable blocks behave like objects allocated on a Pascal heap with NEW and DISPOSE. The user references a non-relocatable block through a pointer to the first data byte.

Relocatable blocks, as the name suggests, may be moved around in memory (relocated) when necessary. There are two principal advantages to using relocatable blocks rather than non-relocatable blocks. First, relocatable blocks can change size dynamically. Second, better utilization of available memory is possible with a zone of relocatable blocks than non-relocatable blocks because the storage manager, when unable to find a free block large enough to satisfy an allocation request will move blocks around to create a large enough free block. In order to achieve this relocatablility the user references a relocatable block through a handle (h) which is a pointer to the pointer to the first data byte. There is precisely one pointer to the actual data, called the master pointer. The handle points to it. The storage manager remembers its location and updates it whenever the block is relocated. This unique pointer is normally located in an array of pointers at the end of the zone header but the user may specify, at zone initialization time, precisely one other block of memory in which he wishes to allocate these master pointers. This area is <= 32K bytes long. A pointer, once allocated in this area, must never move; so a master pointer cannot be allocated inside another relocatable object. The Word Processor uses the Pascal sysglobal area to store master pointers. (Figure 1)

Named objects provide the user a mechanism to implement an LRU caching mechanism on the collection of all named objects in the zone. The Word Processor uses this capability to implement a file page cache. The font manager uses it to implement

a font cache. A named object is referenced by its 32-bit name (n) which the storage manager attaches no significance to other than as a unique tag for the object. Using named objects requires a little more forethought than non-relocatable or relocatable blocks. The user must supply, at the time of zone initialization, several procedure variables which parameterize portions of the caching function. Through these routines the storage manager communicates to the user the intent to swap a named object out of the zone, request the user to copy a named object into the zone and inquire the amount of storage a named object will require. By parameterizing these three functions the storage manager may be used to cache different kinds of objects and need know nothing about the semantics of the objects themselves or where they exist when not cached in the zone.

Since named objects swap into and out of the zone automatically there is no primitive Pascal construct which directly references a named object the way a pointer, p, references a non-relocatable object and a handle, h, references a relocatable object. Instead the user must translate the object name, n, into a TEMPORARY pointer to the data of a named object using the storage manager routine, PMapN. PMapN will find the named object and return a pointer to it. If the object is not in the zone it will swap it into the zone with the aid of the procedure parameters mentioned above. PMapN finds in-zone objects by probing a hash table, rgpnob, which contains a pointer to every named object in the zone.

The storage manager maintains two state bits for each named object, fLock and fDirty, which affect the swapping characteristics of the object. The user may manipulate these flags through the named object interface. Setting fLock to TRUE makes the object memory resident. Put another way, when fLock is TRUE the named object will never be swapped out of the zone. **Beware - fLock does not make the object non-relocatable!!** When a named object is about to be swapped out of the zone the storage manager checks the "dirty bit", fDirty. If fDirty is TRUE then the user-supplied swapout routine is called before the object is deallocated. It is expected the user will take what ever action is necessary to guarantee the long term integrity of the object. IF fDirty is FALSE it is deallocated without the user being notified in any way.

Although not currently enforced by the storage manager it is a universally observed convention that non-relocatable objects not be mixed into a zone with relocatable and named objects. If this convention is not obeyed the user will sometimes find that the storage manager cannot satisfy an allocation request which would have been possible in an equivalent "pure relocatable" zone.

Interface Abstractions

hz — pointer to an area of memory managed as a storage zone (TYPE THz = rTAhz). The thing that hz points to, ahz, is in the interface but considered private. It is included in the interface because of the lack of "opaque types".

p — pointer to a non-relocatable storage object (TYPE TP = rINTEGER).

h — handle to a relocatable storage object (TYPE TH = rTP). Such a handle is a pointer to a pointer to the data you allocated. So, if a user allocates a record of type TFoo in a relocatable storage object she would save the handle in a variable hfoo (say) of type rrTFoo. Access to a field baz in a record of type TFoo is hfoorr.baz.

n 32 bit name of a named (i.e. cached) object (TYPE TN = LONGINT).

Operations

Zone Initialization

```
HzInit(pFst: TP; pLim: TP; pBase: TP; ipPoolMac: TC; logIpnLim: TProc;
      procCbMore: TProc; procCbOfN: TProc; procFSwapInN: TProc;
      procSwapOutN: TProc)
```

Initializes the interval of memory in [pFst .. pLim) to be a storage zone and returns a pointer to it as a result. pBase is a pointer to the beginning of the 32K byte area in which the user may want to allocate master pointers. If pBase = pNil then the zone has no alternative master pointer area; all relocatable objects must me allocated with master pointers in the zone's master pointer pool with HAllocate. ipPoolMac is the number of initial master pointers for relocatable objects allocated in the internal area of master pointers. This number is automatically increased when needed. The initial hash table used for named objects has $2^{logIpnLim}$ entries. procCbMore is the procedure called when the storage manager fails to satisfy an allocation request because of insufficient memory. This procedure has the interface of the generic procedure:

```
FUNCTION CbMore(hz: THz; cbNeed: TC) : TC;
```

The storage manager must be able to add at least cbNeed bytes of memory to the end of the zone, hz, in order to successfully satisfy the allocation request. The return value is the amount the client has allocated immediately after the end of the zone to be added to the zone. If cbNeed > return parameter then the storage manager returns without success. A default routine will be provided if procNil is passed as the parameter but experience has shown that clients almost always need there own to handle boundary conditions in application specific ways.

procCbOfN, procFSwapInN and procSwapOutN are procedure parameters which must be supplied if named objects are to be used in this zone. procCbOfN is a procedure variable which, given a named object n, returns the size (in bytes) of that object. The generic interface is:

```
FUNCTION CbOfN(n: TN) : TC;
```

The procedure variable procFSwapInN is called whenever the storage manager fails to find a named object in the cache (see PMapN). The user is expected to get the object into the zone at the location specified. The generic interface is:

```
FUNCTION FSwapInN(n: TN; pDst: TP) : TF;
```

n is the name of the object; pDst is the destination to "swap" the object to; return value indicates success (TRUE) or failure (FALSE)

The procedure variable procSwapOutN is called immediately before the

storage manager tosses a named object out of the zone (i.e. reclaims its memory). It is called only if the dirty bit, fDirty, is set in the named object header. The user is expected to get the object into the zone at the location specified. The generic interface is:

```
PROCEDURE SwapOutN(n: TN; pSrc: TP);
```

n is the name of the object; pSrc is the location of the cached object

Implementation Note:

The pDst and pSrc parameters have proved awkward for the font manager since in the implementation of FSwapInN and SwapOutN other allocation requests are made which can relocate the new named object and hence invalidate the pointers. In the future you might want to eliminate these parameters and force the routines to map from name to pointer with PMapN or export the hashing function IpnMapN (see below).

Operations on Non-relocatable objects

```
PAllocate(hz: THz; cb: TC) : TP;
```

Allocates a non-relocatable storage object in zone, hz. The actual allocated data area in the object is at least cb bytes long and may be slightly larger. Returns a pointer to the first data byte in the object. This pointer is guaranteed to be on an even byte so that access to 16 bit fields in records will work correctly. Returns pNil if unable to satisfy the allocation request.

```
FreeP(hz: THz; p: TP);
```

Deallocates the non-relocatable storage object pointed at by p in zone hz. The pointer itself is not enough info to find the containing zone which is why the user must pass it in as a parameter.

Operations on Relocatable Objects

```
HAllocate(hz: THz; cb: TC) : TH;
```

Allocates a relocatable storage object in zone, hz. The actual allocated data area in the object is at least cb bytes long and may be slightly larger. Returns a handle to the first data byte in the object. This pointer is guaranteed to be on an even byte so that access to 16 bit fields in records will work correctly. Returns hNil if unable to satisfy the allocation request.

```
FreeH(hz: THz; h: TH);
```

Deallocates the non-relocatable storage object referred to by h in zone, hz.

```
ChangeSizeH(hz: THz; h: TH; cbNew: TC);
```

One of the unique advantages of a relocatable object is that the user may change its size after it has been created. This is the routine to accomplish it. The relocatable object, h, in zone, hz, is altered to have

cbNew data bytes. This may require moving it.

CbDataOfH(hz: THz; h: TH) : TC;

Returns the size, in bytes, of the data area of object, h, in zone hz. Note that CbDataOfH(hz, HAllocate(hz, cb)) >= cb. That is, the actual size of the data area may be greater than what you asked for.

HzFromH(h: TH) : THz;

If a relocable object is allocated with HAllocate then it is possible to algorithmically derive the containing zone. When this is of interest to the user he may find it by calling this function.

Operations on Named Objects

PMapN(hz: THz; n: TN) : TP;

Locates the object named by n in the zone, hz, swapping it into the zone if necessary. Returns a pointer to the first data byte of the object. THIS POINTER IS ONLY VALID UNTIL THE NEXT CALL ON THE STORAGE MANAGER!! Returns pNil if the named object cannot be found or swapped in with FSwapInN.

PCreateNob(hz: THz; n: TN; cbData: TC) : TP;

Creates a new named object in the zone, hz. Used instead of PMapN when the named object does not yet exist outside the cache. For instance, when the Word Processor creates a new file it allocates the pages of the file in the storage zone with PCreateNob.

SetFDirty(hz: THz; n: TN; fDirty: TF);

Sets the state of the "dirty bit" of the named object, n, to the parameter, fDirty.

HLockN(hz: THz; n: TN; fNeedH: TF) : TH;

Makes the named object memory resident; i.e. sets fLock to TRUE. fNeedH and the return parameter are unimplemented hooks. The intent is that when locking a named object the client may also request a handle be allocated to efficiently access the locked object.

UnlockN(hz: THz; n: TN);

Makes the object, n, swappable; i.e. sets fLock to FALSE.

Miscellaneous Zone Operations

CbOfHz(hz: THz) : TL;

Returns the total number of bytes of memory in the zone, hz.

FCheckHzOk(hz: THz; VAR cBkStd: TC) : TF;

A check routine to test the internal consistency of a zone, hz. Returns TRUE if the zone appears to be OK, FALSE otherwise. The return parameter

cBkStd is set to the number of relocatable blocks in the zone.

PxHz(hz: THz);

A print routine to dump a description of the storage zone to the Console.

EnlargeHz(hz: THz; cbMore: TL);

A client will sometimes want to give more space to a zone. This space must immediately follow the zone since zones themselves are not relocatable. EnlargeHz will add to the zone cbMore bytes of memory immediately following the zone. This routine simply reconfigures the zone to include the extra space.

CbShrinkHz(hz: THz; cbLess: TL) : TL;

It MAY be possible for a client to reclaim some of the memory managed by the storage manager for some other use. Using CbShrinkHz the user requests that the last cbLess bytes of memory in the zone, hz, be removed from the zone. The function result is the number of bytes the storage manager successfully stripped from the zone.

AllocBk(hz: THz; hDst: TH; cb: TC; tybk: TTybk);

AllocBk is the lower level routine to which all allocation requests (PAllocate, HAllocate, PCreateNob, etc.) are eventually reduced. It is included for the knowledgable user who wishes, for instance, to allocate a relocatable block with the handle allocated in the interval [pBase .. pBase + 32K). A block of size cb bytes and type tybk is allocated and the pointer to it in hDstr. tybk is one of tybkNrel (non-relocatable), tybkStd (relocatble) or tybkN (named).

FreeBk(hz: THz; h: TH; tybk: TTybk);

Like AllocBk, a lower level routine for FreeP, FreeH and FreeN. The object of type, tybk, pointed at by hr is deallocated.

Dunsels (i.e. accumulated useless trash)

PLstFree(hz: THz) : TP;

ReleaseBkNrel(hz: THz; pFstRelease: TP);

These routines were added to support Pascal heaps with MARK and RELEASE. The Pascal system has never adopted this memory manager.

Note: The reader should not venture further without a listing of UnitHz

Private Abstractions

ahz — header of a storage zone. This record contains all of the structural information for the storage zone. The record is considered private even though it is included in the interface. It is included in the interface because of the lack of "opaque types".

- bkFst — pointer to the first storage block in the zone (in memory address order).
- bkLst — pointer to the last storage block in the zone (in memory address order). This block is always a "dummy" free block at the end of the zone which is never allocated. Its size is cbMinFree. Its existence simplifies, ever so slightly, the management of the free list.
- bkfFst — pointer to the free block which is at the head of the doubly linked free chain.
- pBase — pointer to the user master pointer allocation area (see discussion of relocatable objects).
- argpPool — The array of internal master pointers used by HAllocate. This is at the end of the header and of variable length.
- ipPoolMac — Size of argpPool. argpPool has valid entries allocated from argpPool[0] to argpPool[ipPoolMac-1].
- hFstFree — All unallocated master pointers in argpPool are linked together on a free list. This is the head of that list.
- rgpnob — Hash table for locating named objects in the zone. Pointer to an array of pointers to named objects, (nob). This array is allocated as a relocatable object in this zone.
- mskIpnLst — Size of rgpnobr. rgpnob has valid entries allocated in [rgpnobr[0] .. rgpnobr[mskIpnLst]].
- ipnCur, ubtCur — Variables used to implement LRU caching.
- procCbMore, procCbOfN, procFSwapInN, procSwapOutN — Procedure variables; see HzInit for discussion.

abk — header of a storage block. The format of the block is variant, dependent on the 2-bit type field, tybk, found at the beginning of the block.

If tybk = tybkFree then the header has three four-byte fields: cwFree, the size of the block in words; bkfNxt, a pointer to the next free block on the linked list of free blocks; and bkfPrv, a pointer to the previous free block on the list. Note that cwFree can be accessed as a long integer even though its top two bits are the tybk field because tybkFree = 0.

If tybk <> tybkFree then the first word of a block is always interpreted as a 2-bit type field, tybk, followed by a 14-bit size field (in words), cw. If tybk = tybkNrel (non-relocatable object) these two fields represent the entire header and the data portion begins at byte 2.

If tybk = tybkStd (relocatable object) then the second word of the header is a 16-bit reference to the single pointer to the block (oh). If abk.oh >= 0 then the pointer is found by adding oh to the pointer to the zone, hz (i.e. h := hz + oh). If oh < 0 then the pointer is found by subtracting oh from pBase (i.e. h := hzr.pBase - oh). Client data begins at byte 4.

If tybk = tybkN (named object) then the second word of the header is a 6-byte record describing the object. The first field is the name of the object, n. Following the name is a status word, stn, which contains the flags, fDirty and fLock as well as an 8-bit LRU time, ubt. Client data begins at byte 6.

Private Operations

MakeBkf(hz: THz; bk: TBk; cb: TL);

Changes the block, bk, into a free block of size cb. Chains the block onto the head of the doubly-linked free list.

DeleteBkf(hz: THz; bkf: TBk);

Removes a free block, bkf, from the free list.

BkFindCb(hz: THz; cb: TL) : TBk;

Searches the free list for a block whose size is at least cb bytes. Returns a pointer to it as a result or bkNil if it fails.

BkfLow(hz: THz; bkMin: TBk) : TBk;

Returns a pointer to the first free block (in memory address order) which is greater or equal to the bkMin parameter. Used, for instance by the compactor to find where to begin compaction. Implemented by chasing the links of the free list, NOT by scanning in address order from bkMin.

BkCompactCb(hz: THz; cb: TL; bkLst: TBk) : TBk;

This procedure is the main compaction routine. It is an incremental compactor in the sense that it has very little overhead (a single call on BkfLow dominates the overhead) and it will terminate any time either of two parameterized conditions is met: when it creates a contiguous free block of cb bytes or when it reaches bkLst. The procedure simply iterates the blocks in address order from BkfLow(hz, ...) removing free blocks and pushing allocated blocks towards the front. References to the allocated data are updated on the fly and a single free block is created at the completion of the iteration. A pointer to this free block is returned as the result.

CbMakeBkfBefore(hz: THz; bkAfter: TBk; cbNeed: TL) : TL;

BkCompactCb always moves blocks toward the front of the zone.

Occasionally it is necessary to push blocks towards the end of the block. GrowHInPlace and HMakeMoreMasters are examples. This routine accomplishes that "reverse" compaction. It tries to create a free block of at least cbNeed bytes at location, bkAfter, pushing bkAfter and subsequent blocks towards the end of the zone to make room. The actual size is returned as the result. If the result is 0 nothing was moved.

HMakeMoreMasters(hz: THz) : TH;

If there are not free master pointers in argpPool for HAllocate to use it calls this routine to create some more. It "reverse" compacts all blocks in the zone to make some room at the end of the master pointer table. It then increments hzr.ipPoolMac and links the pointers on the list headed by hzr.hFstFree. Returns hNil if it fails to make any more masters.

GrowHInPlace(hz: THz; cb: TL; bkLst: TBk) : TBk;

GrowHInPlace is a subroutine for the use of ChangeSizeH. ChangeSizeH employs a mixed strategy to achieve its ends. If the block is to be shrunk by at least cbMinFree bytes then it simply creates a free block in the reclaimed space. If it is to grow and there is room in the zone to create a new block of the appropriate size and copy the contents then it tries this next. If all else fails ChangeSizeH calls GrowHInPlace to grow the object without moving it. This involves "reverse" compacting following blocks to make room for size of the new block.

IpnMapN(hz: THz; n: TN) : TC;

IpnMapN is the assembly language hashing function used in probing the named object hash table, rgpnob. Secondary probes are linear to allow for efficient deletion of entries. There is some indication that the function does not "scramble" the bits of n very well which may be resulting in an unnecessarily high average number of probes per call. Returns an index into rgpnob which contains a pointer to the object named by n. If the object is not found then rgpnobr[IpnMapN(...)] = NIL.

IpnChoose(hz: THz) : TC;

IpnChoose implements the named object replacement algorithm. It iterates over a fixed number of named objects (not necessarily all of them) looking for the oldest as measured by its ubt field. Returns an index to this object.

FreeIpn(hz: THz; ipn: TC);

Removes the named object at rgpnobr[ipn] from the zone. If the dirty bit is set the user's swapout routine (hzr.procSwapOutN) is called prior to deallocating the object.

SetCbFree(hz: THz; cbFree: TL; fEnlargeHz: TF);

SetCbFree tries to guarantee that the zone, hz, has at least cbFree bytes of free space. If fEnlargeHz is TRUE it will attempt to expand the zone, if necessary. First, however, it will try to achieve its goal by choosing old named objects and freeing them.

Figure 1 - Storage zone and associated structures

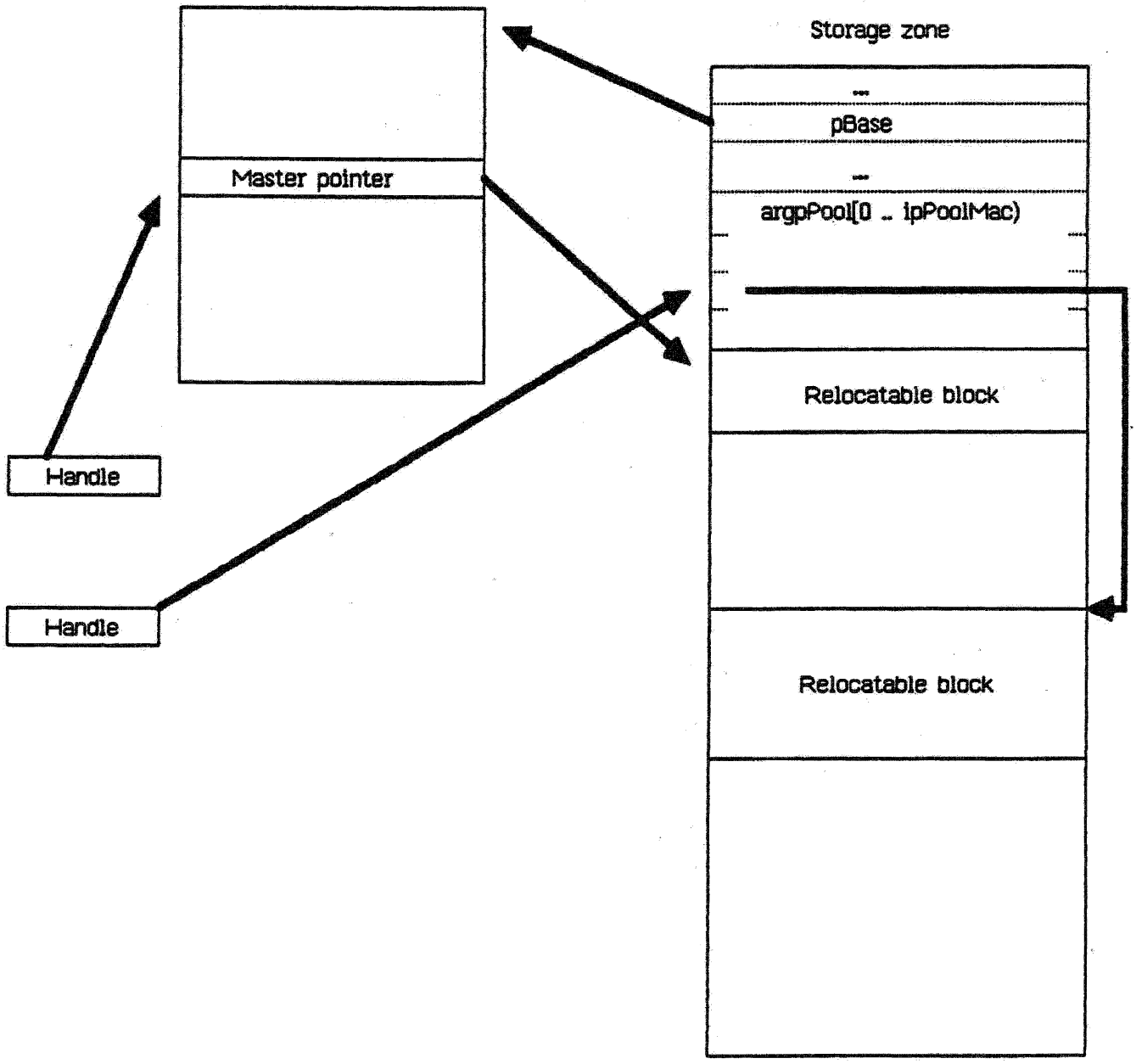


Figure 2 - Storage Zone with two free blocks

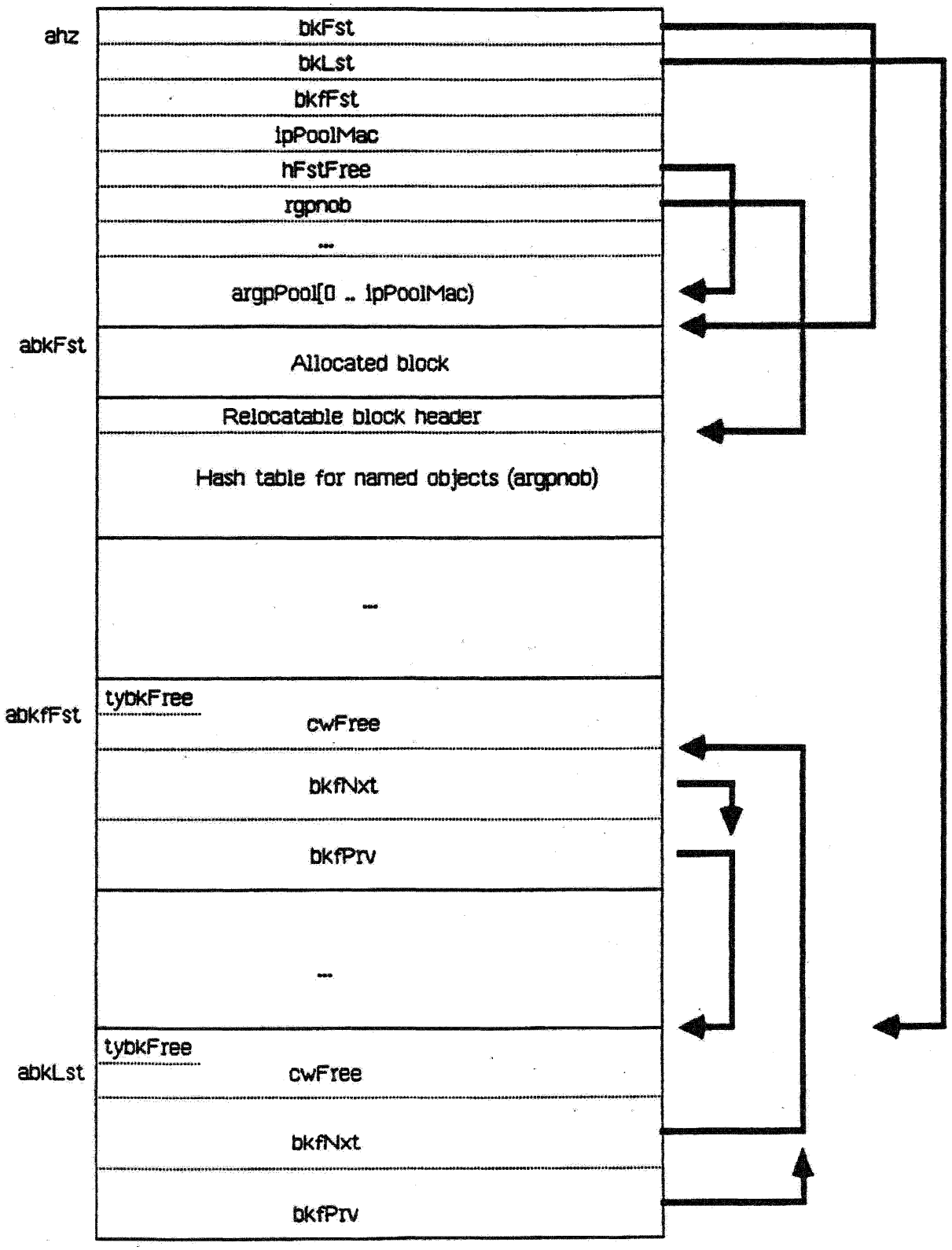
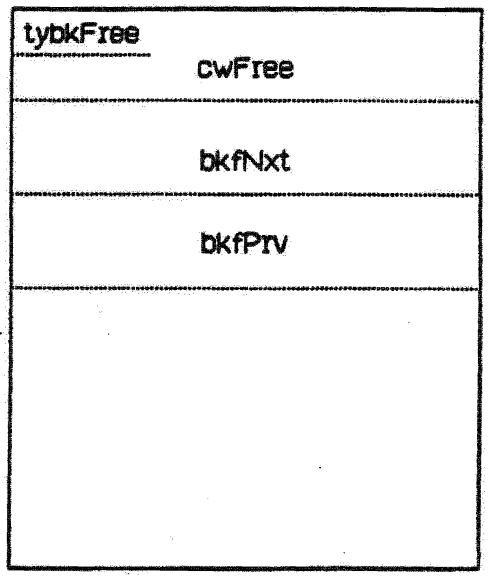
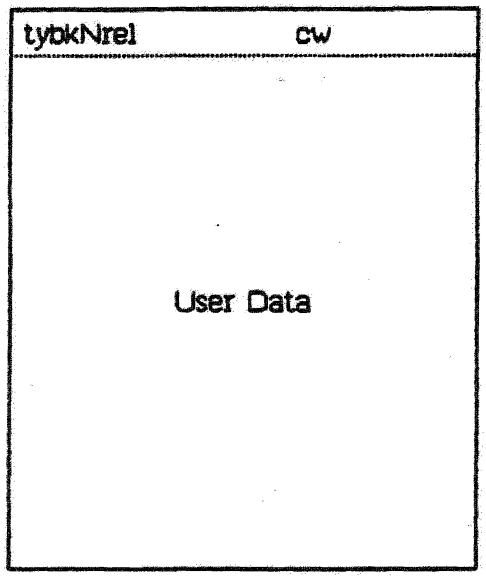


Figure 3 - Block Formats

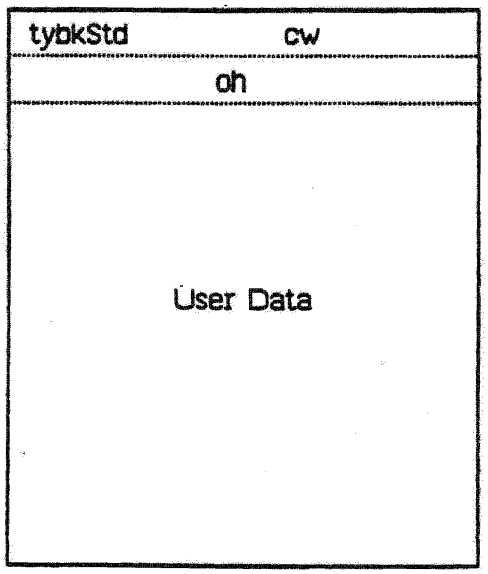
(a) Free block



(b) Non-relocatable block



(c) Relocatable block



(d) Named block