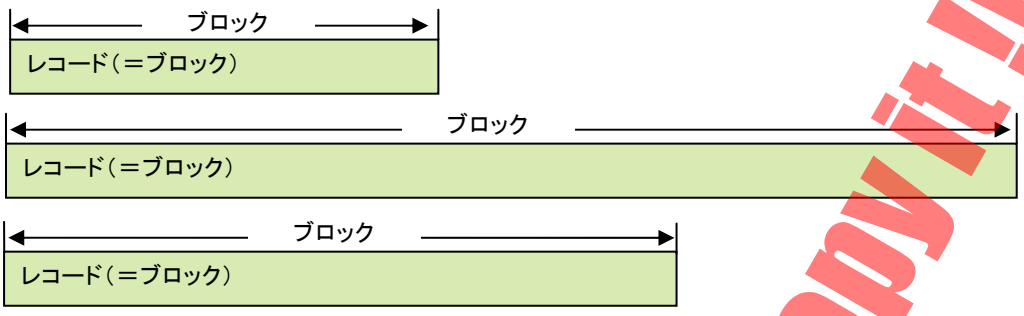
イスから転送された長さで MVS から通知されます。U 形式も読んでみなければ長さはわからないという特徴を持ち、ブロック単位でのみアクセスできます[5]。主にロードモジュール・ライブラリーに使用され、一般のユーザープログラムで使われることはほとんどありません。



RECFM=U

不定長形式は可変長ではあるものの長さを示すデータは一切持たない。ディスクあるいはテープ上の物理的な記録長で取り扱われる。

図 3-8 不定長レコード形式

▶ラージ・ブロック

z/OS ではテープデバイスに関してのみ 32760 バイトを超えるブロック長がサポートされており、LBI（ラージ・ブロック・インターフェース）と呼ばれます。実際のブロック長は装置がサポートする最大長に依存しプログラム側では指定できません。主に大容量テープへのアクセス速度の向上の目的で使われます。

デバイスとボリューム

データセットはディスクやテープに記録され、保存されます。ハードウェアとしてのディスク装置やテープ装置がデバイスです。デバイスには記録媒体が取り付けられ、実際のデータが記録されます。MVS ではこの記録媒体をボリュームと呼びます。ボリュームとはデータセットを格納する入れ物（器）と言ってよいでしょう。デバイスとボリュームは必ずしも 1 対 1 になりません。テープ装置や PC の CD-ROM を考えるとわかりやすいでしょう。機械の方がデバイスで、記録媒体となるテープやコンパクトディスクがボリュームに相当します。

▶デバイス

MVS ではデバイス（装置）には番号が付けられユニットとして管理されます。すべての入出力デバイスが対象になり、番号は 4 桁の 16 進数で構成されています。これを装置番号（Unit Number）と呼びます。

装置番号はしばしば装置アドレスと呼ばれることもあります。これは初期の MVS における呼び方の名残で、その頃はデバイスに CUU（チャネルアドレス+ユニットアドレス）を割り当てて管理していたからです。どのチャネルを通ってどのデバイスへ行くかの経路も OS 側で管理していました。MVS/XA によってチャネルは動的サブチャネル・システムへと変わり、経路の制御はチャネル側の仕事になりました。その結果、CUU の考え方はあてはまらなくなったので装置番号になったのです。

[5] ブロックは、1 つの論理レコードでのみ構成され、ブロック長=レコード長となります。

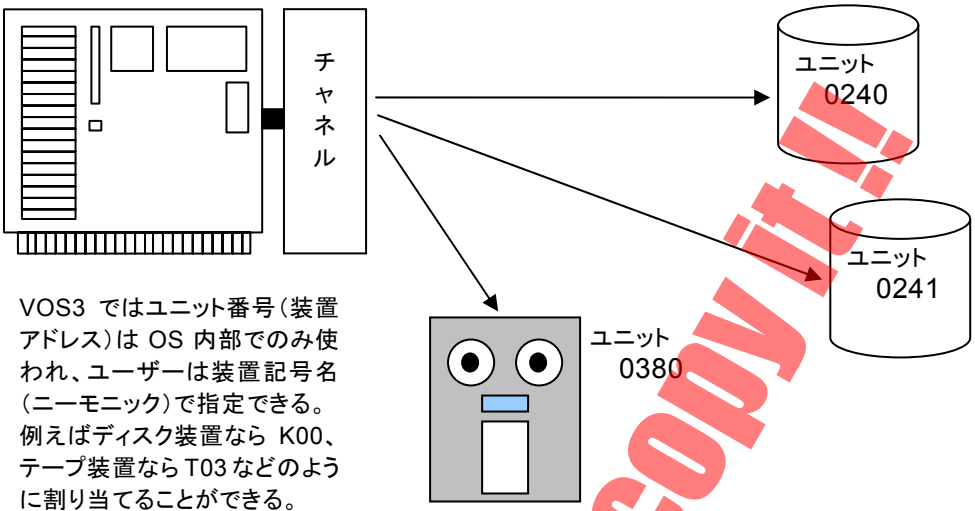


図 3-9 デバイスとユニット番号

▶ボリューム

テープ装置では取り外し可能な記録媒体が使われます。リールとかカートリッジと呼ばれるものです。同じ装置に時に応じて違う記録媒体が取り付けられるため、装置番号ではテープ上のデータセットを正しく指し示すことができません。そこで装置に取り付けられる記録媒体に名前を付けて識別できるようにしています。これがボリューム通し番号（ボリューム名）で、最大 6 文字までの英数字を使用します。ボリュームの考え方はディスク装置でも同じです。ディスクの場合は取り外し可能媒体ではありません[6]が、考え方として装置と記録媒体を分けているのです。

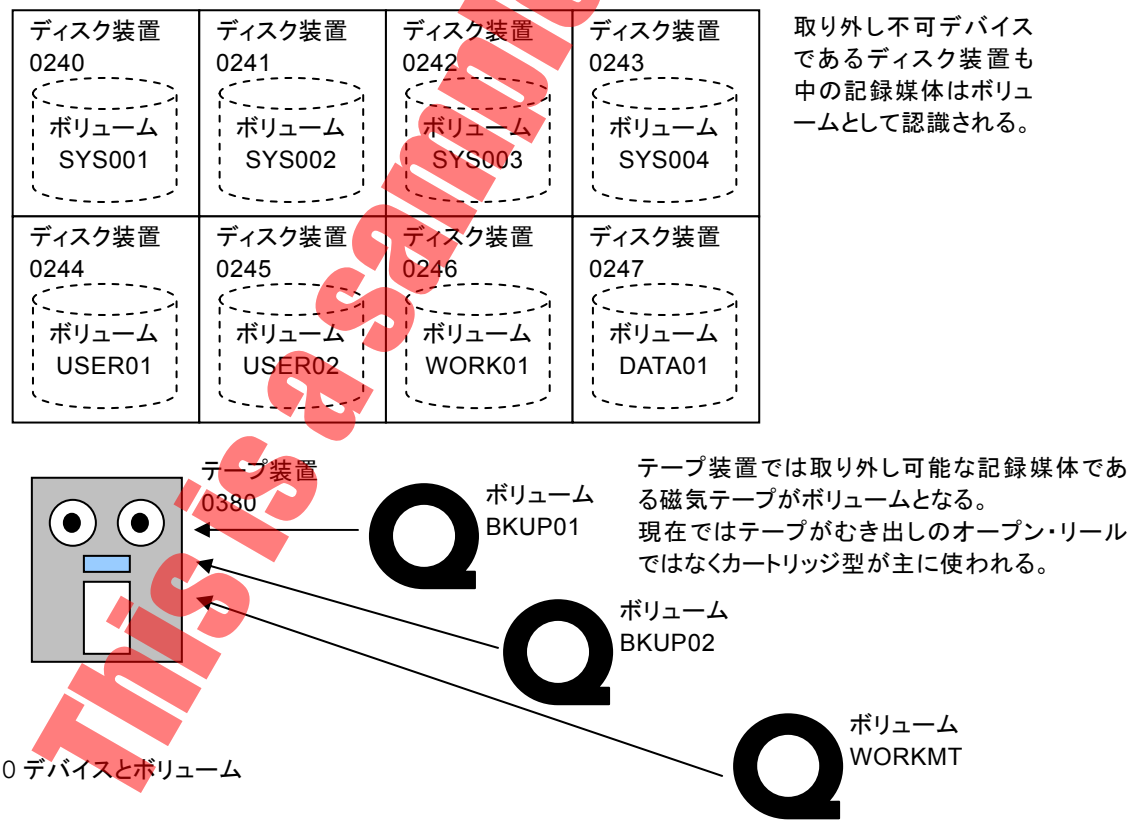
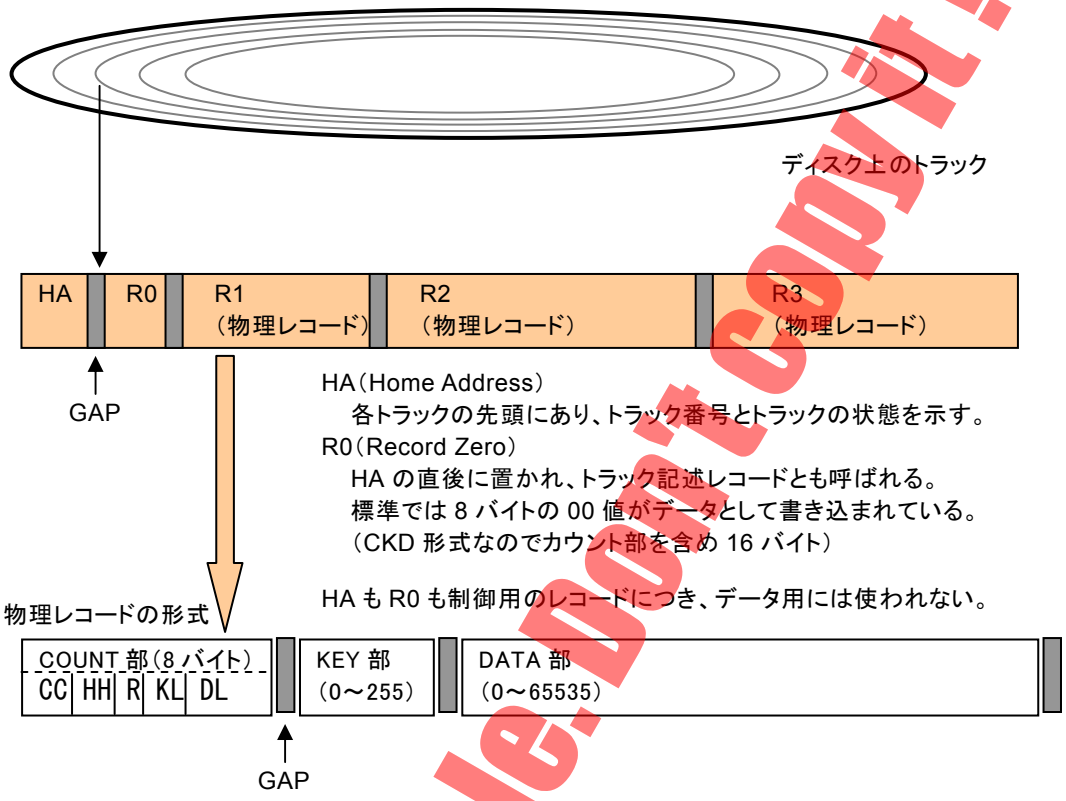


図 3-10 デバイスとボリューム

[6] 実際には初期のディスク装置（IBM3330 型など）では取り外し可能なものもあり、記録媒体はディスクパックと呼ばれました。その名残から今でもディスクをパックと呼ぶこともあります。

データの長さが記録され、キー部[16]とデータ部にはユーザーデータが記録されます。CKD ではトラックは長さに応じて必要な量だけ使われるため、スペース効率が高まりますが、直接ディスクをアクセスする場合、プログラミングは複雑になります。IBM には別に、Windows などでも使われているセクター固定長記録方式もあり、FBA（Fixed Block Architecture）と呼ばれ、VSE という OS でサポートされます。



カウント部（COUNT） … 物理レコードのトラック上の位置を示す。
CC：シリンダー番号（x0000～）
HH：トラック番号（x0000～x000E）
R：　レコード番号（x00～xFF）（ユーザーデータは x01 以降となる）
KL：キー部の長さ（x00～xFF）
DL：データ部の長さ（x0000～xFFFF）
ユーザーから見た場合、ブロック（物理レコード）の長さと考えてよい。
また実際の最大長はディスク装置の仕様で決められる。

現在のディスクは 1 シリンダー当たりのトラック数は 15 となっています。よって、トラック番号は 0 から始まり 14（x000E）までとなります。以降は、繰り上がってシリンダー番号が 1 増え、トラック番号は 0 に戻ります。CC-HH=0000-000E の次は CC-HH=0001-0000 となります。通常のプログラムでデータセットをアクセスするにはこのような知識は不要ですが、DASD の保守ユーティリティなどを使う際には知っておくと便利です。

キー部（KEY） … キー付きデータセットのキー内容が格納される。
データ部（DATA） … データセットのブロック内容が格納される。
プログラムやエディターなどで書き込んだデータはここに入る。

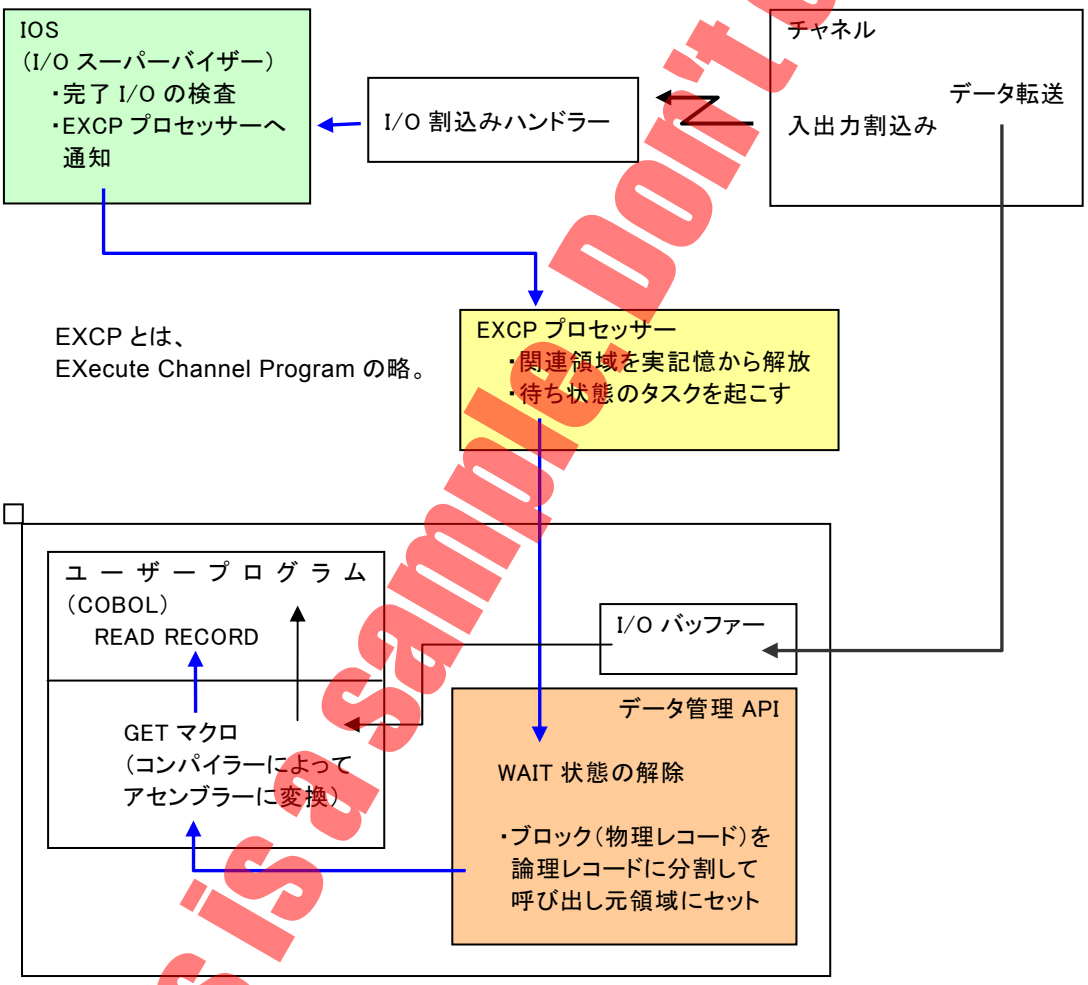
CKD 形式のレコードは R1 から順次可変長で書き込まれます。トラックの残り容量が書き込もうとするレコードの長さを満たさない場合は、そのレコードは次のトラックの R1 として書き込まれます。この場合の可変長はディスクの物理的な記録方式のことで、データセットのレコード形式（FB、VB、U など）とは別のものです。また、1 つの CKD レコードは複数のトラックにはまたがりません。

図 3-17CKD 方式によるトラックへのデータの格納

[16] データベースでいうキーと同様の概念で、ハードウェアの機能でレコードを検索できるキー付きデータセットというものを利用するときに使われましたが、現在では VSAM やデータベースが一般的になり、ユーザー用にはほとんど使われません。

⑩　I/O スーパーバイザー　EXCP プロセッサーへ I/O 完了を通知する
⑪　EXCP プロセッサー　・固定したコマンドとデータ転送領域を実記憶から解放
　　・タスクを起こす（データ管理 API 内で待ち状態）
⑫　データ管理 API　I/O バッファーに読み込まれたブロックをデブロッキング
⑬　コンパイラー　レコードをユーザープログラム領域に書き込む
⑭　ユーザープログラム　READ 完了

本書で説明しない用語がいくつか出てきましたが、COBOL によるユーザープログラムであれば READ ステートメントに対応した I/O モジュールから先は、データ管理 API（アクセスメソッド）、EXCP プロセッサー、I/O スーパーバイザーという 3 つの MVS コンポーネントを介して、チャネルへたどり着きます。COBOL のような高級言語では READ、WRITE ステートメントで読み書きの動作を指定しますが、実際の I/O 処理はもっと低レベルで複雑です。OS のコンポーネントの階層が下がるにつれ、より低いレベル（抽象的・論理的から具体的・物理的に）の処理が行われている、ということを理解してください。



この例は一般のプログラムが DASD や TAPE 内のデータセットを論理レコード単位でアクセス（読み込み）するものである。アクセスするデバイスや方法により幾種類ものアクセス方法があるが、「プログラム→アクセス法（API）→EXCP プロセッサー→I/O スーパーバイザー」の順に制御が流れて行くことが基本となる。
また、一部のシステム系プログラムではアクセス法によらず直接 EXCP プロセッサーを呼び出している。IOS を直接呼び出す必要はほとんどないが、オフライン状態のデバイスをアクセスするような一部の DASD ユーティリティでは IOS を直接呼び出している。

図 3-23 入出力処理の流れ②

▶拡張区分データセット（PDSE）

PDSE は従来の PDS の欠点が改良された新しいタイプの区分データセットで、DFSMS の機能によって提供されます。今までの PDS アクセスメソッドで利用できるため、プログラム互換を持ちます。PDSE には次の利点があります。

- ディレクトリには索引とキャッシュが付き、PDS より高速なディレクトリ・エントリーの探索ができる。
- ディレクトリ部は必要に応じて自動的に拡張される。
- 圧縮の必要がない（必要に応じて内部で自動的に行われる）。
- メンバー名は 8 バイトだが、別名として 1024 バイトまでの長い名前を付けることができる。
- 複数のジョブで同時にメンバーを登録・更新できる。
- メンバーは格納時には最適なサイズでブロック化されるが、読み出し時にはプログラムが指定するブロックサイズで渡るように調整される。

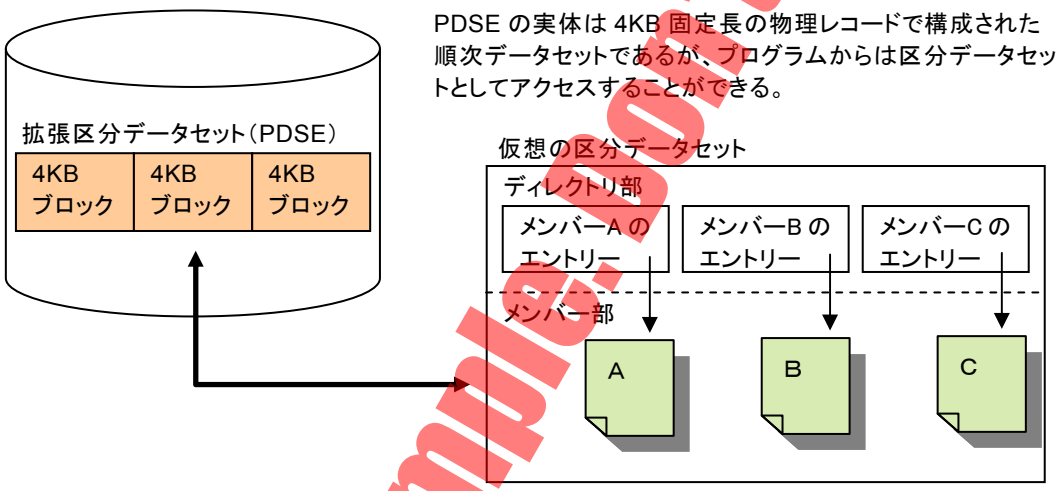


図 3-30 拡張区分データセット

VSAM データセット

VSAM(Virtual Storage Access Method)もデータをレコード単位に転送し、管理するための仕組みですが、他のデータセットと異なり、OS/360 からの互換ではなく、S/370 による仮想記憶の出現によって実装されました。当初から DASD と仮想記憶の使用を前提に設計されたもので、従来からあるデータセット編成とアクセスメソッドに比べ、次のような特徴を持ちます。

- 1 つのアクセスメソッド（VSAM）で目的に応じた複数の編成を持つファイルをサポートする。
- デバイスの物理的特性を意識する必要がない（従来は装置の特性や、装置上の物理的な記録フォーマットを意識する必要があった。特に基本アクセス方式）。
- アクセス効率の向上。
- カタログによるデータセット管理（データセットの構成情報は VTOC だけでなく、カタログも使用するため、データセットの構造が VTOC の仕様に制限されない）。
- データの機密保護や保全性の向上。

VSAM でもデータの実体は DASD 上のデータセットとして格納されます。しかし、順次データセットや区分データセットのように、物理的な構造がそのままプログラムにマッピングされません。DASD 上のデータセットには VSAM による内部フォーマットによってデータが整理されて格納されています。この中のデータをプログラムで扱うレコードの形式に再構成して仮想記憶上に展開するのが、アクセスメソッドとしての VSAM です。

VSAM は次の 4 種類のタイプのデータセットをサポートします。

- KSDS（キー順データセット）
- ESDS（エントリー順データセット）
- RRDS（相対レコードデータセット）
- LDS（リニアデータセット）

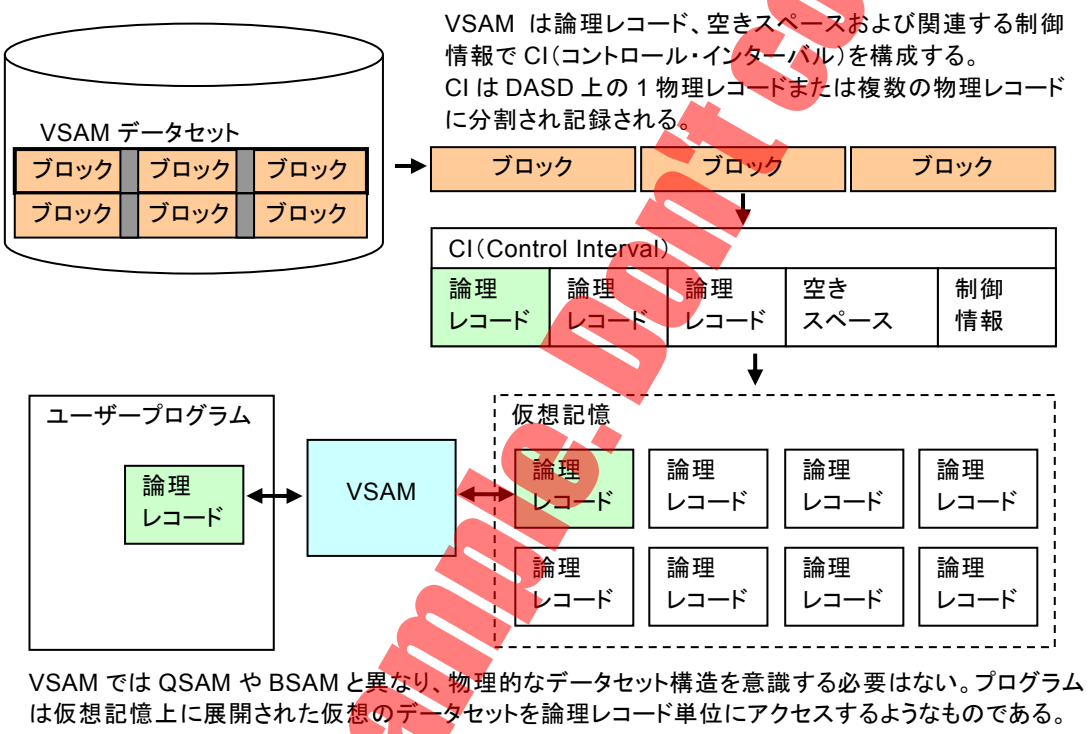


図 3-31VSAM データセットの概要

▶KSDS（Key Sequenced Data Set）

KSDS は、レコードの一部分がキーとして定義され、レコードがキーの値で順番に並んでいる[18]データセットです。データセットはキーを管理するインデックス部とレコードを格納するデータ部によって構成されます。レコードは、キーの順番で順次にアクセスすることもできますし、キーの値で直接アクセスすることもできます。

KSDS はデータ管理が従来からサポートしていた索引順次データセット（ISAM）の代替としても使用できます。VSAM の KSDS はキー探索の速さなどデータベースの原型と言えるでしょう。

[18] 実際に順番に並んでいるのはインデックス部で、データ部の並びは必ずしもキー順ではありません。

SRM の目標は大きく 2 つです。

- ジョブのプログラムの応答時間を短縮する（資源を優先的に配分する）。
- 全体のスループットを向上する（資源の使用効率を高める）。

しかし、この 2 つは相反する関係にあります。1 つのジョブを優先すれば、他は動けません（資源に遊びが生まれる）。すべてを平等に扱えば、みんなで遅くなります（資源を取り合い、競合する）。

そこで必要になるのが「妥協」です。お互いに大きな支障にならない程度にバランス良く対処するわけです。SRM の役割は、この妥協点を見いだすことでもあります。元々はオペレーターの作業分野ですが、この種の作業は「勘と経験」を頼りにした高度な技能を必要とします。SRM は、これを OS 自身で肩代わりするものです。

SRM では、管理の対象を次のように分けます。

■ サービス…… CPU、メモリー、I/O など使われる資源です。
■ ワークロード …… バッチや STC タスク、TSO ユーザーです。

MVS ではユーザーの仕事の単位はジョブですが、SRM ではこれを「トランザクション」と呼びます。バッチや STC タスクではジョブ、TSO では 1 回のコマンド入力がトランザクションに対応します。IMS や CICS など、オンライン・リアルタイム処理を行うシステムでは、ジョブ全体のサービス制御に加えて、DC システムのトランザクションを SRM のトランザクションに対応させることもできます（バッチや TSO でもジョブ名やユーザーID で細分化できます）。

▶WLM（Workload Manager）

ワークロードマネージャーは、一定の周期で個々のトランザクションが受けているサービス量の配分状況を調べて、サービス率を監視します。なお、CPU のサービス量は単純な使用時間の積み上げではありません。CPU は機種によって能力が変るので、例えばプログラムが命令を百万回連続して実行しても、CPU モデル A と CPU モデル B では実行に要する実時間は変ってきます。プログラムが要求するのは、百万個の命令を処理できる CPU のサービスであって、○○秒間 CPU を使うことではありません。そこで、WLM は CPU の使用時間を機種ごとの定数で性能差を補正（時間÷モデル別定数）したものをサービス量としています。こうすることで、機種が変っても同じサービス配分になるように調整されるわけです。

OS/390 以降の MVS では WLM は独立したアドレス空間で動作し（それ以前の MVS と MSP、VOS3 では、WLM は SRM の内部プログラムで独立した空間は持たない）、個々のトランザクションに与えられたパフォーマンス目標を達成するのに必要なサービス量を、ダイナミックに算出して配分されるように SRM に働きかけます。これが GOAL モードと呼ばれるものです。それ以前の MVS では初期設定パラメーター（IPS、ICS）によって、どのトランザクションにどの程度の量でサービスを配分したり、ディスパッチの優先順位を決めたりするかを指定しておく必要がありました。SRM はその定義に基づいてサービスを配分していたのです。

しかしながら、ジョブがどの程度の量の CPU や実記憶を使うかを正確に見積もるのは非常に困難です。ましてやそのジョブが動くときに資源にどの程度の余裕があるかなどは予測できません。何となくオンラインだから優先度を上げ、バッチだから下げると言ったようにかなりアバウトな設定がなされる例が多かったのです。もちろん、きめ細かく定義をしているユーザーもいますが、それはシステム管理者が高度なスキルを持っていればこそ可能なことでした。GOAL モードでは目標さえ示せば、その目標を達成すべく WLM がシステムの負荷状況に応じて適切なサービス配分を自動的に行ってくれます。なお、MSP と VOS3 では以前の MVS 同様に初期設定パラメーターによって、サービスの配分と優先度が定義されます。

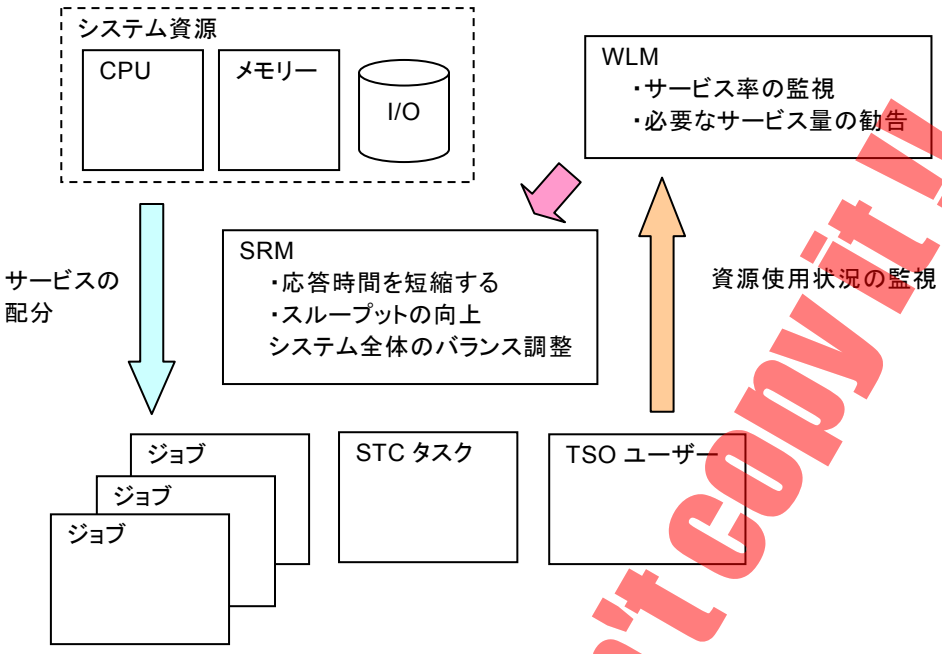


図 3-40 SRM によるシステム資源の配分

SRM は WLM の勧告に基づき具体的な資源の配分調整を行います。具体的には次のような制御を行います（抜粋）。

▶スワッピング

スワッピングは空間（ジョブ）の多重度を調整するために行われます。サービス率が上昇したジョブのスワップ・イン優先度を下げ、低下したジョブの優先度を上げます。ジョブはサービス率が上昇したまま資源を使い続けるとスワップ・アウトされやすくなります。スワップ・アウトによってジョブが使用していた CPU とメモリー（実記憶）は解放されます。

▶競合待ちの最小化

リザーブされた DASD や排他使用中のデータセットの使用待ちジョブがあるとき、SRM は排他使用中の資源を持っている方のジョブを優先的にスワップ・インします。

▶ディスパッチング

他の実行可能な空間があれば優先順位に従い、CPU ディスパッチを切り替えます。

▶ストレージ不足の防止

補助記憶装置の空き容量が一定の割合を下回ると、新しいアドレス空間の生成を停止します（TSO へのログオン、STC タスクの起動が抑止されます）。

▶DASD の割り振り

一時的データセットなどを割り振るときに、特定のボリュームに偏らないよう、利用可能ボリュームを平等に使うことを試みます。

これらに加え、マルチ CPU のプロセッサーでは、ディスパッチャーは次にディスパッチするタスクを決めたら、CPU ワークロードがなるべく平準化されるように、割り当てる CPU を決定します。ディスパッチャー自身も CPU を使用しているため、ディスパッチャー自身が実行されている CPU とは別の CPU でタス

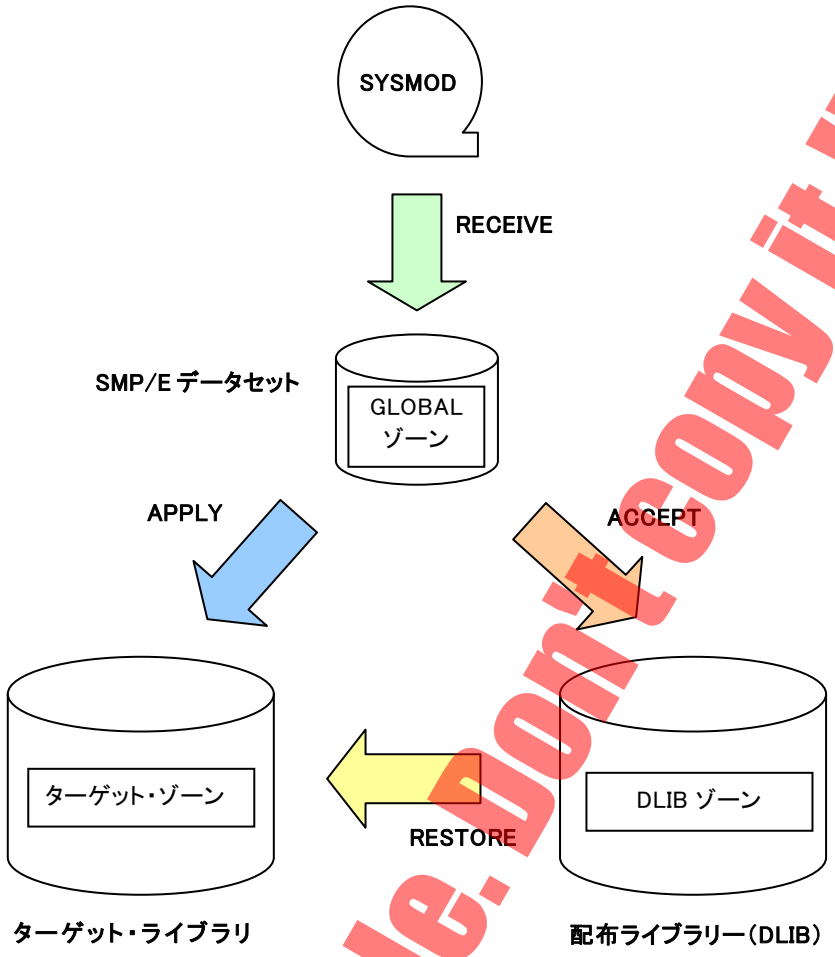


図 3-41 SMP によるシステム修正のインストール

RECEIVE

SYSMOD を SMP/E データセットにロードして、インストール（適用）できるようにします。

APPLY

SYSMOD をターゲット・ライブラリー（稼動しているプログラム）にインストールします。

ACCEPT

SYSMOD を配布ライブラリー（DLIB）にインストールします。ACCEPT 済みのシステム修正は取り消すことができません。

RESTORE

DLIB からターゲット・ライブラリーを復元して、APPLY した SYSMOD を取り消します。APPLY する直前のレベルのターゲット・ライブラリーに戻すことができます。

SMP/E データベースは、ソースコード、オブジェクトモジュール、ロードモジュールの関係と、個々のシステム修正（SYSMOD）の依存関係を蓄えます。例えば、SYSMOD①で修正した問題に、新たな対応策を追加するために SYSMOD②が提供されたとします。この場合、SYSMOD②を適用するためには、あらかじめ SYSMOD①を適用しておく必要があります。これをシステム修正の依存関係といいます。

SYSMOD の依存関係や、どのモジュールがどの CSECT[29]で構成されているかという関係については、SMP/E 特有の「++」で始まる言語（SMP/E 修正制御ステートメント）で登録する方法がほとんどです。しかし、システム生成によるユーザー環境の違いなどにより、プログラムの構成要素が異なるものは、システム生成の際に使われた JCL を直接読み込ませて、DLIB からターゲット・ライブラリーを生成した情報を登録するものがあります。このような方法を JCLIN と呼びます。

SMP/E でシステムを修正する作業は必要ですが、慎重にやらねばなりません。例えば、定期メンテナンスなどにおいて、複数の SYSMOD を適用している途中で、ワークエリアが一杯になるなどの理由でジョブが異常終了すると、リカバリーは困難になります。事前にバックアップを取るなどして、慎重に作業を行います。

[29] Control Section の略。実際の機械命令や処理データなど、プログラムの実行コード部を構成するひとつのまとまり。アセンブラー言語では、実行コード部を定義するための命令でもあります。1 つのモジュールは 1 つまたは複数の CSECT で構成されます。

作成された VTOC のアドレス（CCHHR）は、ボリューム・ラベルに設定され、DASD アクセス時に VOL ラベルから VTOC へと参照されます。VTOC の大きさは、そのボリュームに格納されるデータセットの数によって決まります。1 つのデータセットは、最低でも 1 つの DSCB が使用されます。データセットが拡張され、エクステントが 3 を超えると、さらに 1 つの DSCB が必要になります。

VTOC は、DSCB を 1 トラック当たり、3390 型 Disk の場合 50 個、3380 型では 53 個収容できるので、これに基づき必要な VTOC トラック数を計算します。また、エクステント拡張や制御用の DSCB 数も考慮して、さらに 1～2 トラック程度を追加します。

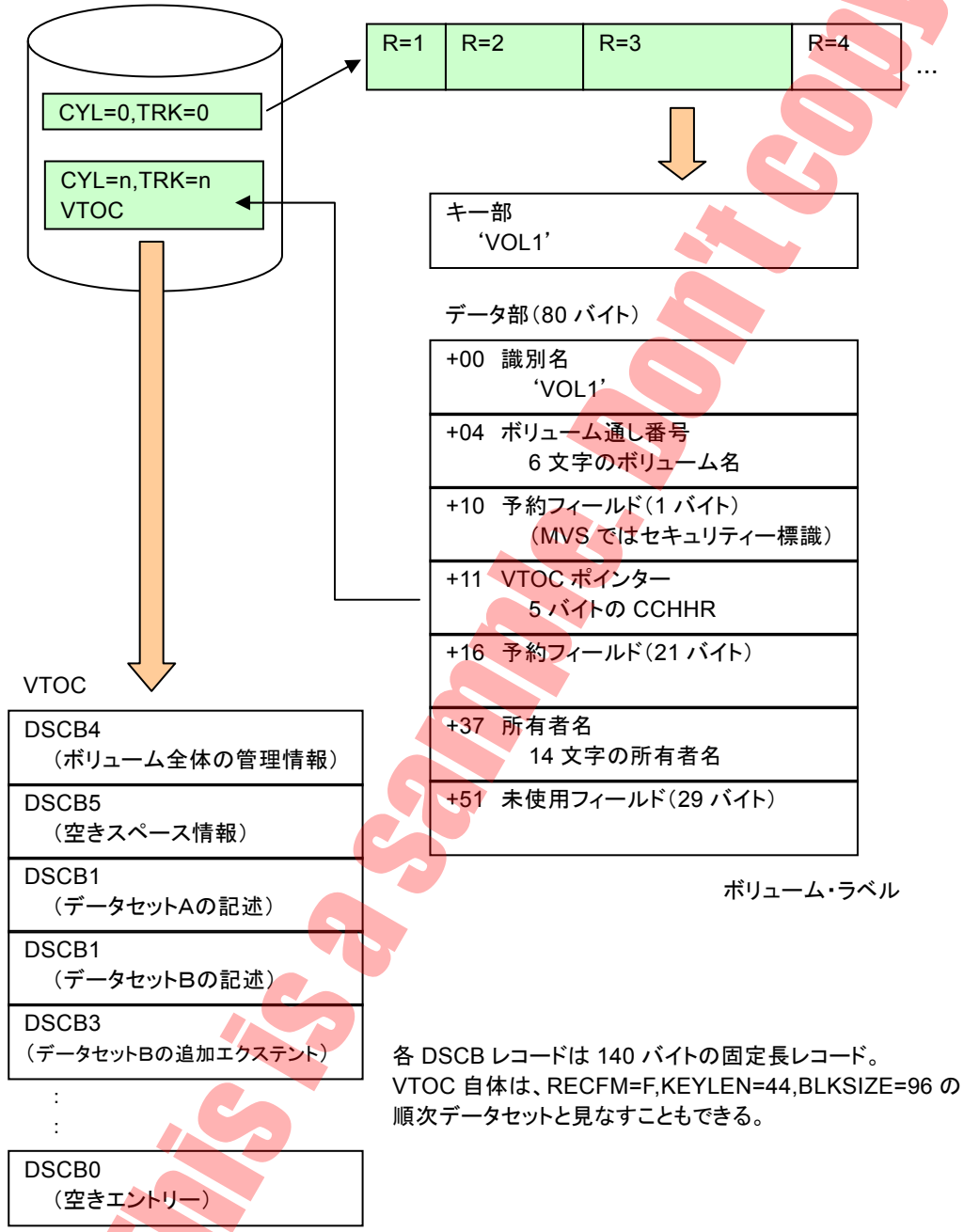


図 8-1 ボリューム・ラベルと VTOC

▶DSCB レコード

VTOC の DSCB には、用途別に 0、1、3、4、5 および 7 の 6 種類があります（DSCB2 および DSCB6 は廃止されています）。

DSCB4

VTOC の先頭に位置し、ボリューム全体の管理情報と VTOC 自身のエクステントを示します。キー部には、44 バイトの x04 が設定されています。

DSCB5

ボリューム内の空きスペースを管理します。最初の DSCB5 は、DSCB4 の直後に置かれています。1 つの DSCB5 は、26 の空きスペースを管理できます。ボリューム内の空きスペースの数が 26 を超えると、新たな DSCB5 が作成されます。インデックス VTOC では、空きスペースは VTOC インデックス・データセットで管理されるため、最初の DSCB5 のみ存在しますが、使用されません。

DSCB1

データセットごとに作成され、データセット領域を構成するエクステントのアドレス、レコード形式、ブロック長、レコード長などのデータセット属性、およびその他の制御情報を持ちます。キー部には、44 バイトのデータセット名が設定されています。

DSCB3

データセットの拡張されたエクステントを管理します。DSCB1 では、3 つまでのデータセット・エクステントが管理できますが、これを超えて拡張されると DSCB3 が作成されます。1 つの DSCB3 は、13 のエクステントを管理できます。順編成や区分編成などのデータセットは、DSCB1 による 3 エクステントと、DSCB3 による 13 エクステントで、最大 16 のエクステントを持つことができます。VSAM や PDSE などでは、さらに DSCB3 をチェインして作成することができ、最大で 123 のエクステントを持つことができます。

DSCB7

大容量ボリュームの空きスペースを管理します。3390-9、27 および 54 型（それぞれ 10017、32760、65520 シリンダーの容量を持つ）など、4369 を超えるシリンダー（総トラック数が 65535 を超える）を持つ大容量ボリュームが非インデックス VTOC を使用する場合は、ボリューム内の空きスペースを従来の DSCB5 では管理できません。代わりに DSCB7 が使われます。1 つの DSCB7 当たり 16 の空きスペースを管理できます。

インデックス VTOC では、空きスペースは VTOC インデックス・データセットで管理されるため、DSCB7 は使用されません。また、インデックスの有無にかかわらず、最初の DSCB5 は存在しますが、これも使用されません。

DSCB0

空きの DSCB レコードです。未使用または不要になった DSCB は、ヌル（x00 値）でクリアーされ、必要な際に DSCB として再利用されます。

VTOC インデックス・データセット

VTOC インデックス・データセットは、「SYS1.VTOCIX.volser」の DSN で示され、ボリューム上のデータセットの探索効率を上げるために使用されます。大きな VTOC を持つボリュームでは、VTOC の先頭側

5 ····記憶クラスの表示・登録・更新
6 ····記憶グループの表示・登録・更新
7 ····ACS オブジェクトの登録・更新、ACS ソース元データセット名の表示
8 ····CDS データセットの定義・更新・検証・アクティブ化

SMS 定義に関する修正が終了したら、「8」でアクティブ化して反映させます。アクティブ化を実行すると、コンソールに確認のリプライ・メッセージが出力されるので、「Y」を応答します。その間、TSO 端末は応答できなくなるので注意してください。

ISMF ユーティリティの操作に関する詳細は、OS マニュアル「DFSMS 対話式記憶管理機能(ISMF)の使用法」および「DFSMSdfp ストレージ管理リファレンス」を参照してください。

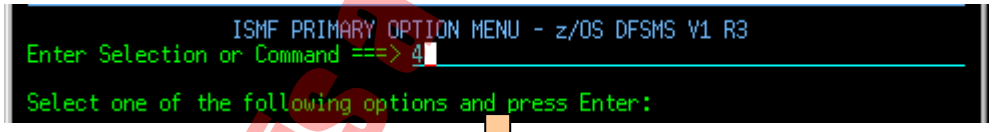
管理クラス

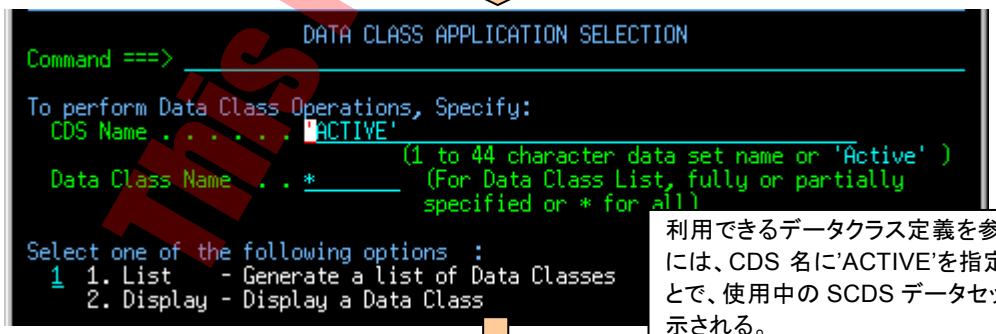
自動マイグレーション、自動バックアップ、保存日数などの属性を設定するクラスです。自動マイグレーションとは、一定期間参照されないデータセットを自動的にマイグレート（アーカイブ）することです。管理クラス名を JCL DD 文で指定する場合は、MGMTCLAS パラメーターを使用します。なお、管理クラスは SMS データセットを作成する上で必ずしも必要なものではありません。

データクラス

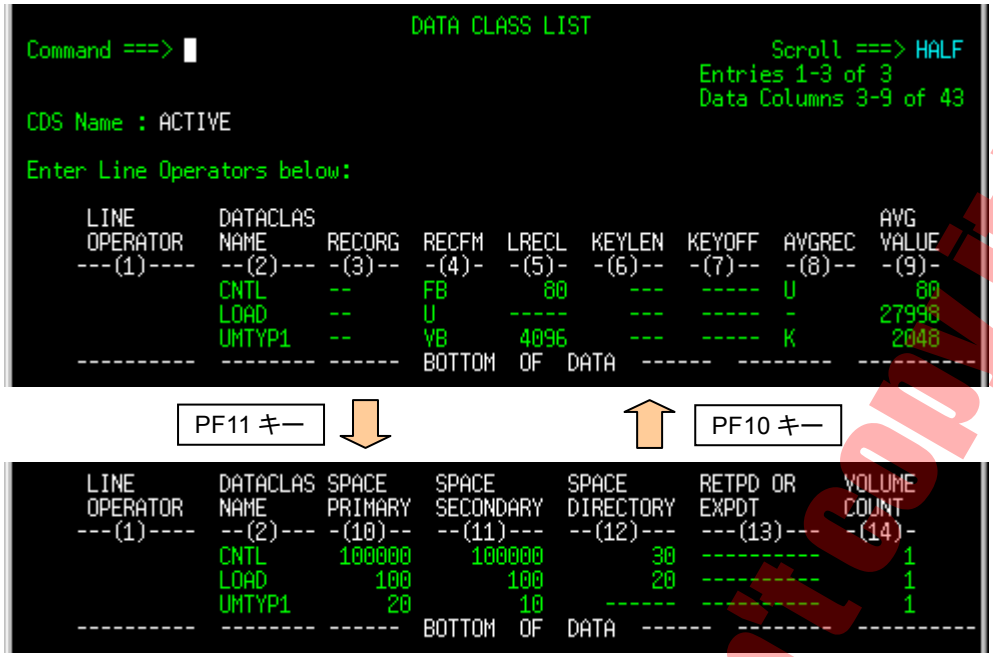
データセットの DCB および SPACE 割り振りに関するデフォルト属性を設定するクラスです。ISPF 3.2、TSO ALLOCATE コマンドや JCL DD 文で、これらに関するパラメーターを設定する代わりにデータクラスを指定します。JCL DD 文で指定する場合は、DATACLAS パラメーターを使用します。なお、DATACLAS と同時に、DCB および SPACE パラメーターを指定した場合は、DCB や SPACE パラメーターの値が優先します。DATACLAS は、これらを省略した際に補完される標準値と考えてよいでしょう。

通常は、データセットの使用目的に応じて複数種類のデータクラスを定義します。例えば、JCL やプログラムなどのライブラリー用、ロードモジュール用、業務用のマスターデータ・ファイル用などです。データクラスも SMS データセットを作成する上で必ずしも必要なものではありません。また、データクラスは非 SMS 管理データセットの作成時にも利用できます。ACS ルーチンと組み合わせれば、データセット名によって自動的にデータクラスを割り当てることもできます。これは、データセットの属性を標準化したり、誤った指定を防いだりすることにもなります。





利用できるデータクラス定義を参照するには、CDS 名に'ACTIVE'を指定することで、使用中の SCDS データセットが表示される。



データクラス CNTL では、RECFM=FB、BLKSIZE=80 および 1 次・2 次スペース量が 100,000 レコード分、ディレクトリ・ブロック数 30 の DCB と SPACE 情報が割り当てられることになる。

画面 8-17 ISMF によるデータクラスの参照

記憶クラス

記憶クラスは、パフォーマンス目標と可用性（Availability）要件を設定するクラスです。データセットを割り振るべきボリュームを単に空き容量や業務内容によって選択するのではなく、

- パフォーマンス目標を達成可能なボリューム
- 二重ボリューム、RAID ボリューム、ストライピング・ボリューム
- スペース保証（マルチボリュームの事前割り振りなど）
- 同期書き込み保証

といった観点で SMS に選択させることができます。

もちろん、これらの複雑な基準を適用することなく、単にグループ分けの目的でクラスを設定することもできます。この場合はクラス名を決めれば、設定内容は省略値や初期値のままで登録されます。この場合の記憶クラスは、データセットを適切な SMS ボリュームに割り振るために使用されます（SMS データセットは必ず記憶クラスを持ちます）。いずれにせよ、最低 1 つの記憶クラスが必要です。記憶クラスは、DD 文の STORCLAS パラメーターで指定されます。また、ACS ルーチンによってデータセット名から適切な記憶クラスを設定させることもできます。

記憶グループ

記憶グループは、SMS が管理する物理的なストレージ装置を定義します。SMS ボリュームにデータセットを割り当てる場合、最低 1 つの記憶グループが必要です。記憶グループは、タイプ（DASD プール、テープ等）によって分けられ、各々のグループには名前が付きます。SMS で管理される DASD ボリュームは、POOL タイプの記憶グループのどれかに属します。また、ボリュームを自動バックアップや自動マイグレーションの対象にするかなどの属性を設定することもできます。SMS 管理データセットがどのボリュームに

▶SMS 管理データセットの作成例

前提環境

記憶グループ SMS1SG にはボリューム SMSVOL が、SMS2SG にはボリューム SMS001 と SMS002 が登録されているものとします。クラスや ACS ルーチンは、サンプルで紹介した内容で定義されているものとします。

◆STORCLAS=SMS1 とした場合は、SMS 管理データセットとしてボリューム SMSVOL に作成されます。

```
//SYSUT2   DD DISP=(,CATLG),DSN=MY.TESTLIB,
//            STORCLAS=SMS1,SPACE=(CYL,(10,10,20)),
//            DCB=(RECFM=U,BLKSIZE=4096)
```

◆DSN が SMS2 で始まる場合は、SMS 管理データセットとしてボリューム SMS001 または SMS002 に作成されます。

```
//SYSUT2   DD DISP=(,CATLG),DSN=SMS2 TEST.SOURCE,
//            SPACE=(CYL,(10,10,20)),
//            DCB=(RECFM=FB,LRECL=80)
```

SMS データセットの場合は、記憶グループで作成されるボリュームが選択されます。また、SMS ボリュームには、指定されたデータセット以外は作成できません。記憶グループの割り当てをデータセット名などからの選択に限定した ACS ルーチンを作成すれば、空きスペースに規定外のデータセットが作成されるようなことを防ぐことができます。例えば、DASD ボリュームを業務やグループ（部署）などで分けて運用する場合、ルールの遵守を利用者自身に委ねる必要がなくなります。

▶データクラスを指定したデータセットの作成例

前提環境

記憶グループ SMS1SG にはボリューム SMSVOL が、SMS2SG にはボリューム SMS001 と SMS002 が登録されているものとします。クラスや ACS ルーチンはサンプルで示した内容で定義されているものとします。

◆データクラス CNTL には、次の属性が設定されているものとします。
RECFM=FB、LRECL=80、1 次スペース量 100000 レコード分、2 次スペース量 100000 レコード分、ディレクトリ・ブロック数 30

```
//SYSUT2   DD DISP=(,CATLG),DSN=MY.JCLLIB,
//            UNIT=SYSDA,VOL=SER=WRKVOL,
//            DATACLAS=CNTL
```

WRKVOL が 3390 型の場合、145 トラックのスペースが割り当てられます（80 バイトのレコードを 100,000 行と 30 個のディレクトリ・ブロックが格納できるトラック数が、SMS によって計算されます）。ブロック長も SMS によって最適化された値が設定されます。

◆データクラス UMTYP1 には、次の属性が設定されているものとします。
RECFM=VB、LRECL=4096、平均レコード長 2048 バイト、1 次スペース量 20K レコード分、2 次スペース量 10K レコード分

```
//SYSUT2   DD DISP=(,CATLG),DSN=GYOMU1.MASTER.DATA,
//            STORCLAS=SMS2,DATACLAS=UMTYP1
```

STORCLAS で記憶クラス SMS2 を指定したので、ACS ルーチンによって記憶グループ SMS2SG が割り当てられます。よって、SMS データセットとして SMS が選択したボリューム（SMS001 または SMS002）

に、750 トラックの順次データセットが割り振られます（2048 バイトのレコードを 20,480 行分格納できるトラック数が SMS によって計算されます）。

```
//SYSUT2   DD DISP=(,CATLG),DSN=GYOMU1.ALTER.DATA,
//            STORCLAS=SMS2,DATACLAS=UMTYP1,
//            SPACE=(CYL,1)
```

DATACLAS に SPACE パラメーターを追加した例です。この場合、割り振られるスペース量は SPACE パラメーターに従います。区分データセットのディレクトリ・ブロック数のみ変更する場合は、SPACE=(,(,,10)) のように指定することができます。

SMS を活用すると、データセットの作成と配置（ボリューム選択）を標準化したり、ルールで管理したりすることが容易になります。ボリューム数とデータセット数が膨大になる大規模なセンターでは、特に有用な機能です。なお、運用中の SMS 定義を変更するには、管理者権限がなければなりません。一般ユーザーの場合は、ISMF で現行の定義内容や ACS ルーチンを参照して、自分が利用可能なクラスがあれば、それらの定義済みクラスなどを利用して SMS の機能を試してみてください。

データセットのマイグレーションとリコール（HSM）

一定期間利用していないデータセット、あるいは、通常は使わないが何かのときのためにとっておきたいといったデータセットは、アーカイブしておくことができます。アーカイブされたデータセットは、指定された DASD または TAPE ボリュームに圧縮されて保管されます。アーカイブされたデータセットを再利用する場合、DFSMS は必要に応じてデータセットを自動的に復元します。

データセットをアーカイブすることをマイグレーション、アーカイブされたデータセットを復元することをリコールと呼びます。HSM は、このアーカイブの機能によってストレージをより効率よく利用できるように管理します。HSM は、ストレージをレベル 0～2 の 3 つの階層で管理します。

レベル 0 は、通常の DASD ボリュームであり、直接アクセスできるデータセットが入っています。レベル 1 は、マイグレートされたデータセットが入るボリュームであり、オンライン状態の特定の DASD ボリュームが利用されます。データセットとしては存在しますが、HSM 固有の形式に圧縮・変換されていますから、直接参照することはできません。レベル 2 はさらに利用頻度の少ないデータを移動するボリュームであり、テープまたは DASD が利用されます。たいていの場合、マウントされていないか、オフラインとなっている装置です。

このように、データの記憶場所をアクセスのしやすさで階層に分けて管理することから「Hierarchical Storage Manager」と呼ばれます。

> HSM におけるアーカイブはバックアップとは違います。バックアップは複製（コピー）であって、元のデータと同じ物、あるいは同じ内容に戻せる物を、別の場所に作っておくことです。しかしアーカイブは移動であって、元のデータを別の場所に移して、元の場所から消してしまいます。そのためデータは一箇所にしかありませんから、マイグレート先のボリュームが壊れてしまえばデータは消失してしまいます。そのためアーカイブされたデータセット（またはマイグレート先のボリューム）もバックアップは必要になります。

HSM のコマンド（パラメーター）設定による自動マイグレーションによらず、手動でデータセットをマイグレーションするには、TSO から次のコマンドを使用します。

```
HMIGRATE 'dsname'
または
HSENDCMD MIGRATE DSNAME(dsname)
```