CRIWARE®

光ディスクを乗りこなせ！
ロード時間の短縮手法

2008年9月11日

㈱ CRI･ミドルウェア
押見正雄

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

CRIWARE®

はじめに

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

㈱ CRI・ミドルウェア

映像・音声を専門としたミドルウェア開発会社

1990年	人工知能・CD・音声・映像技術の研究開発	
	FM-Towns・メガCDなど	
1993年	サターン用CDシステムの開発	
1995年	サターン用ADXのリリース	５０タイトル
1997年	Dreamcast用ADX・Sofdecリリース	４５０タイトル
2008年	Xbox360・PS3・Wii・PSP・DS	
	Xbox・PS2・GC用ミドルウェア	1400タイトル以上

ゲーム機に特化したミドルウェア

ファイルマジックPRO

CRIWARE®

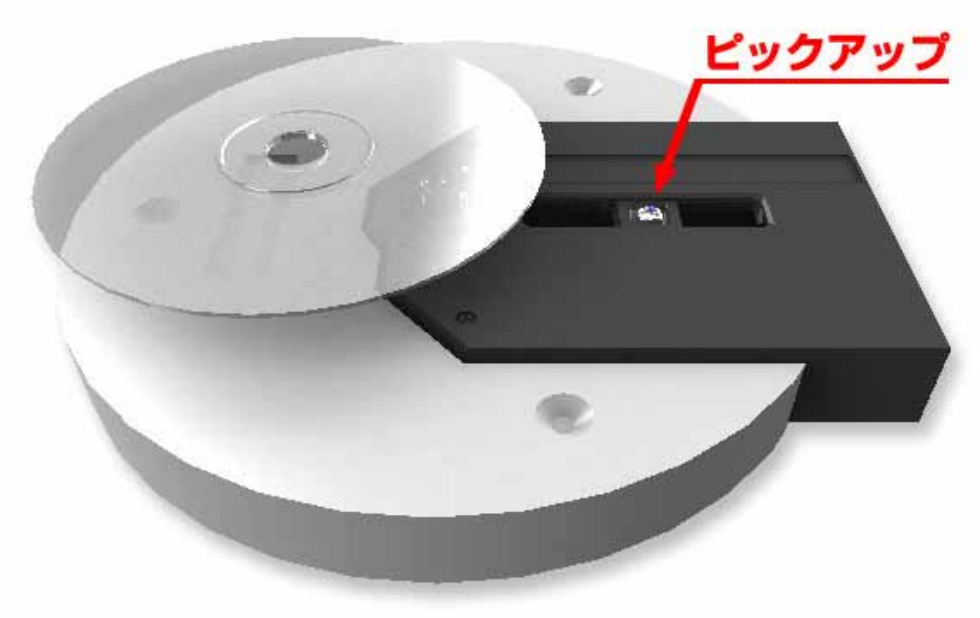
光ディスクドライブの基礎知識

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

光ディスクの仕組み1

■ ピックアップの読み込み動作

- ピックアップを目的のおおよそ場所まで移動。
- レーザーにより位置を特定し、データ読み込み開始。



光ディスクの仕組み2

■ 2種類のシーク方法

- スイング型シーク　：静かで速い。
- ストローク型シーク：シーク音が発生し、遅い。

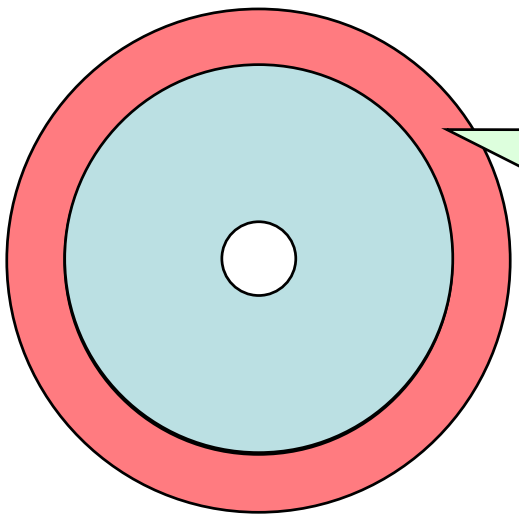
＜スイング型＞

＜ストローク型＞

光ディスクの仕組み3

■ リードエラーが起きたときの動作

- 初めは最高の回転速度でスピンドルをキック
- リードエラーが発生すると**低速回転で再度リトライ**。
- **経年劣化**したドライブや外周ではリードエラーがおきやすい。
- CAVで12センチのディスクをリードする場合、外周は内周の**約3倍**。



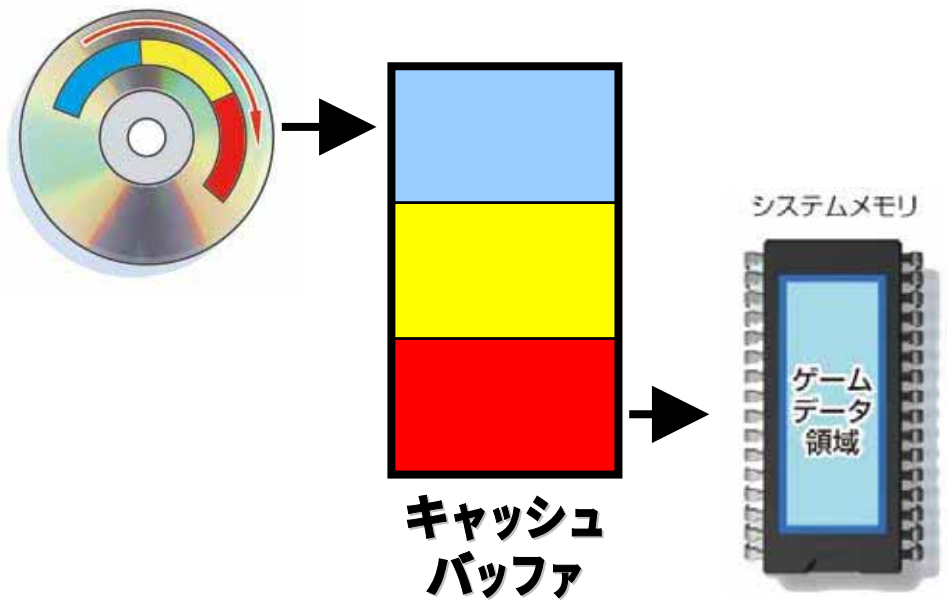
リードエラーが発生しやすく、読み込みが遅くなる。

➔この領域（1/4程度）には重要なデータは置かない方が良い。

光ディスクの仕組み4

■ ドライブ内のキャッシュバッファの役割

- ホストからの読み込み指示によってシークして読み込み開始。
- ホストから指示された範囲が読み終わっても、その先を読み続ける。
- 別の場所へシークすると**キャッシュバッファがクリア**される。




```
// 1回で100セクタ読み込む
fread(buf, 100, 2048, fp);

// 10回に分けて100セクタ読み込む
fread(buf, 10, 2048, fp);
fread(buf, 10, 2048, fp);
        ・
fread(buf, 10, 2048, fp);
```

➔ 読み込みにかかる時間は同じ

光メディアドライブとメモリ容量の移り変わり

■ **第5世代ゲーム機　(1995年頃：PS、サターン)**
転送速度　約300Kバイト/秒　メモリサイズ 約2Mバイト

■ **第6世代ゲーム機　(2000年頃：PS2, DC, GC, XBOX)**
転送速度　約3Mバイト/秒　メモリサイズ 約30Mバイト

■ **第7世代ゲーム機　(2005年頃：PS3、Xbox360)**
転送速度　約10Mバイト/秒　メモリサイズ 約500Mバイト

メモリサイズの増加　＞　光メディアの転送速度の増加
第7世代では、すべてのメモリ領域にロードするのに1分ぐらい。

光メディアドライブとシーク時間

■ シーク時間の比較

光メディアドライブ：　100～500ミリ秒
ハードディスク：　5～20ミリ秒　← 約20倍

■ シークによるロード時間の増加

もし１００個のファイルをランダムに読む場合
シーク時間：　100ミリ秒×100個 = 10秒失われる

■ リード速度の比較

光メディアドライブ：　約10Mバイト/秒
ハードディスク：　約20Mバイト/秒　← 約2倍

ドライブキャッシュ

■ ドライブキャッシュの効用

- メモリなどの一般的なキャッシュとは動作が異なるので要注意。
- ピックアップが読み始めると指定された範囲が読み終わっても、
  その先を読み続ける。
- ドライブキャッシュがフルになると、読み込み停止。
- **後方への多少のシークはペナルティがない**。
  ドライブキャッシュ内のデータを捨てるだけで、読み込みは継続する。

■ エミュレータについて

- 通常、エミュレータはドライブキャッシュのエミュレーションはしない。
  ➔ **実際の光メディアのほうが速い**。

CRIWARE®

ロード時間を短縮するためには

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

ロード時間を短縮するには

リード時間の短縮化 = 読み込むファイルのサイズを減らす → ファイルの「圧縮」+「展開」

シーク時間の短縮化 = 読み込むファイルを連続的に配置する → ファイルの「最適配置」

ファイルロードの手順

■ 一般的なファイル読み込みのシーケンス

1) 単純なファイルのロード
　　オープン ➔ リード ➔ クローズ
2) シーケンシャルアクセス（ストリーミング, パックされたファイル）
　　オープン ➔ リード ➔ リード ➔ リード ➔ クローズ
3) ランダムアクセス　（辞書的なデータ）
　　オープン ➔ シーク ➔ リード➔ シーク ➔ リード ➔ クローズ

※　ゲームアプリでは、1）または2）が多い。
モデルやモーションなどを1つのファイルにパックしてある場合は、
最初にヘッダを読んで、その後に各データを読み出すので、
2)のスタイルになることが多い。

各手順における光ディスクドライブの動作

■ オープン

- ディレクトリ情報（ディスク上の位置やサイズ）の取得。
- ディレクトリ情報は**常駐方式**と**キャッシュ方式**がある。
- キャッシュ方式の場合、メモリ上に読み込まれていなければ、ディレクトリ情報をロード。ディレクトリ情報は**再内周**にあることが多いためシーク量も多く、ロード時間に**大きなペナルティ**を与える。
- ディレクトリ情報のロードは、ロード時間の不安定要素となる。また、音声やビデオの**ストリーム再生の途切れ**る要因となる。

➔ **できる限りパッキングし、ディレクトリ情報は常駐させる。**

各手順における光ディスクドライブの動作

■ シーク

- 一般的には、ハンドル内のメンバーに書き込むだけの非常に軽い処理。
  実際のピックアップのシーク（移動）はリード時に行う。
- プラットフォームによっては、**実際にピックアップが移動**するので要注意。
- **後方への短距離のシーク**は、ほとんどペナルティが発生しない。
  例えば、パックされたファイルが2048バイトアライメントになっていて、
  アライメント調整のために読み飛ばしても大丈夫。
- ピックアップのシーク時間はリニアには増加せず、ある閾値を超えると
  リニアに増加していく。（DVDで数百Mバイト、BDで数Gバイト）

➔ **できる限りシークしないようにする。**
パッキングする場合は、ファイルを連続的に配置する。
単一ファイル内でも、連続的に読み込むようにデータを配置する。

各手順における光ディスクドライブの動作

■ リード

- 必要に応じてピックアップを移動し、ディスクからデータを読み出す。
- すでにドライブキャッシュに存在する場合は、高速に読み出し可能。
- ドライブキャッシュに存在しない、または足りない場合はさらに読み出す。
- 基本的には内周は遅く、外周は速い。
  どこでも同じ速度になっているプラットフォームもある。
- 数バイト単位で読み込むとオーバーヘッドが大きい。

➔ **圧縮し実際に読み取るデータ量を小さくする。**

できる限り連続的にリードする。
リードしたデータを処理した後に次のリードをしても、
処理時間が短時間であればドライブキャッシュが吸収してくれる。
数バイト単位で読む場合は、中間バッファを用意。

各手順における光ディスクドライブの動作

■ クローズ

- 通常は、ほとんど何もせずにハンドル領域を解放するだけ。
- プラットフォームよっては、まれにブロックされるので要注意。

➔ パッキングしてクローズ処理をしないようにする。
クローズ処理を別スレッドで処理する。

ロード時間を短縮するための手法

■パッキング
オープン、クローズの負荷を無くす。

■ファイル配置の最適化
できる限り連続的に読むようにファイルを配置し、シーク時間を短縮する。

■データ圧縮
ファイル圧縮によって読み込む量を削減、リード時間を短縮する。

■ 音声やビデオの効率良いストリーミング
データのロード途中で音声ファイルやビデオファイルへのアクセスを
少なくする。

CRIWARE®

「侍道3」のロード時間の短縮

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

「侍道3」について

侍道3
WAY OF THE SAMURAI 3

販売： 株式会社 スパイク

開発： 株式会社 アクワイア

2008年11月13日発売予定

非常に自由度の高い剣術アクションゲーム

「侍道3」のロード時間短縮への道のり

■ 背景

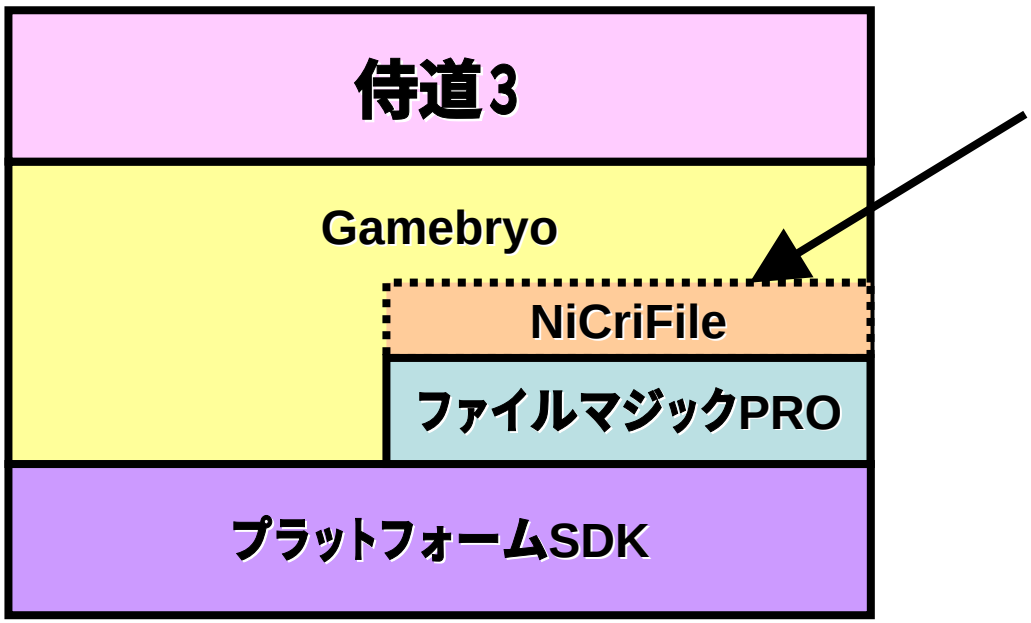
- エマージェント社のゲームエンジン「Gamebryo」を使用。
- PC上では5秒のロード時間がPLAYSTATION®3では25秒に。
- アクセス解析を行い、ロード時間を短縮。
- 69個のファイル（トータル25Mバイト）を約600回で分割リード。

■ ロード時間短縮の過程

- オリジナル	25秒	
- 音声のストリーミング再生の停止	17秒	8秒改善
- パッキング	14秒	3秒改善
- ファイル配置の調整	10秒	4秒改善
- 圧縮	7秒	3秒改善

「侍道3」（Gamebryo）への「ファイルマジックPRO」の組み込み

■ 「ファイルマジックPRO」の組み込み
- ゲームエンジン「Gamebryo」のファイルアクセスレイヤの置き換え。



ファイルアクセスクラス
NiFileの派生クラス
NiCriFileを作成。
簡単に置き換え可能。

短縮のための手法1　(音声のストリーミング再生の停止)

■ 音声ストリーミングの停止

- データロード時に他社ミドルウェアを使用し音楽を再生。
- ストリーム用バッファへ読み込むために、1秒間に数回のアクセス。
- データファイルのロード中に割り込んで、音声データを読み込む。
  ➔ シークが頻繁に発生していた。
- ドライブキャッシュが破棄され、約1/3のパフォーマンスが失われていた。

➔ データロードが25秒から17秒になり、8秒短縮。

短縮のための手法2 （パッキング）

■ パッキングによるオープン･クローズ処理の除去

- オープンに時間がかかる場合とそうでない場合が存在。
  ➔ ディレクトリ情報のアクセスが起きているのときに遅いのでは。
- 約9000個のデータファイルを1つのファイルにパッキング。
- 初期化時に、ディレクトリ情報をロードするようにシステムを修正。
- 起動時にオープンを1回だけ行い、2度とオープン･クローズ処理はしない。

➔ 17秒が14秒になり、3秒改善された。

短縮のための手法3　(ファイル配置の調整)

■ シークが起きないようにファイルを配置

- ファイルアクセスのログから、シークが起きないようにファイルを配置。
  ➔ パックされたファイル内を連続的に読み出すようになった。
- 「Gamebryo」は、ファイル内を数十回にわけて細かくリードするが、ファイル内ではシークせずに連続的に読み出している。
- 256Kバイトの内部バッファを持ち、細かいリードのオーバヘッドを緩和した。
- 最初にエミュレータを用いて計測したときは12秒だった。
- 実際にBDを焼いて計測すると10秒となった。(ドライブキャッシュの効用)

➔ 14秒が10秒になり、4秒改善された。

ドライブキャッシュの効用

■ データ処理時間を隠蔽

- データ処理後のリード時間が非常に短くなる。
  ➔ ドライブキャッシュからシステムメモリに転送するだけ。
- ドライブキャッシュ上のデータが無くなると通常の読み取り速度に戻る。

＜アクセス履歴の概略＞

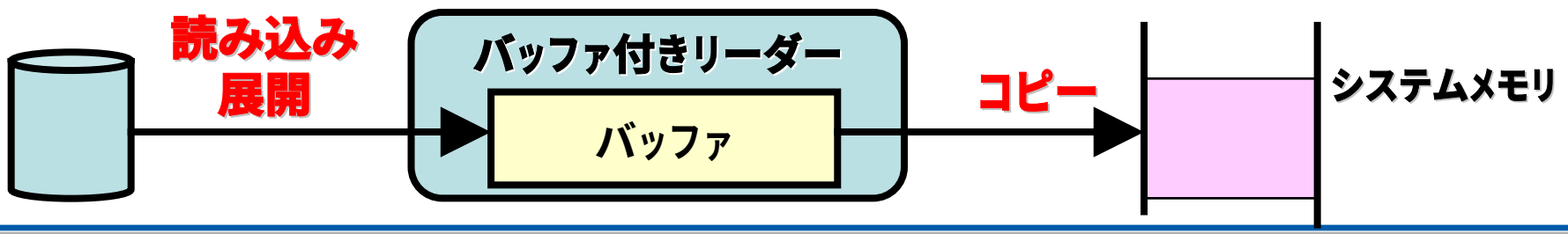
Load	15 msec	通常の読み込み
Load	15 msec	
＜ロードされたデータを処理＞	40 msec	キャッシュに貯まる
Load	6msec	キャッシュから転送
Load	6msec	
Load	11msec	
Load	15msec	

短縮のための手法4　(ファイルの圧縮)

■ ファイルを圧縮

- 内部バッファサイズを5Mバイトに増やした。
  (「Gamebryo」のグラフィックデータのほとんどは5Mバイト以下だったため)
- 5Mバイト以下のファイルを圧縮。圧縮率は60%程度。
- 圧縮ファイルを内部バッファに読み込み・展開し、「Gamebryo」からの要求に従って、内部バッファからコピーして渡した。
- 圧縮ファイルの読み込みと展開のオーバーラップはできなかった。

➔ 10秒が7秒になり、3秒改善された。



CRIWARE®

パッキングについて

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

パッキングの問題点

■ プログラマ vs グラフィッカー

- プログラマはパフォーマンスのためにパッキングしたいが、
  グラフィッカーはすぐに確認したいのでパッキングするのはいやだ。

■ 圧縮によるパッキング時間の増加

- パッキングだけならまだしも、再圧縮すると非常に時間がかかる。

■ パッキングされていないファイルも同じAPIでアクセスしたい

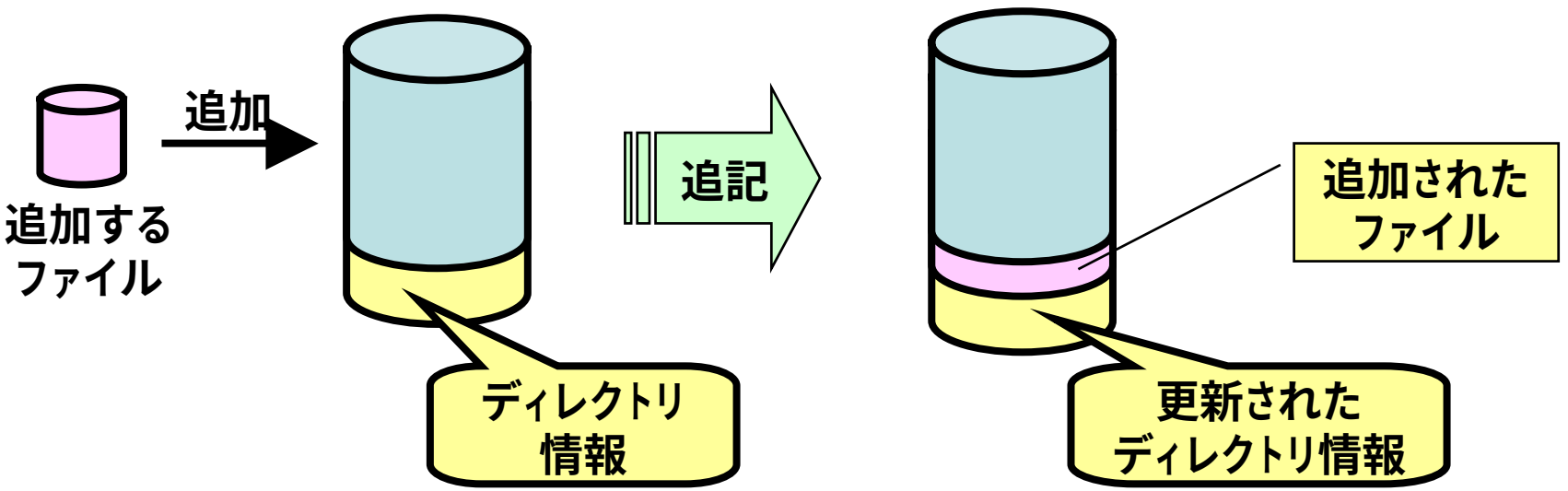
- 制作工程上、どうしてもパッキングできないファイルがでてくる。
- プログラマは、これらのファイルに同じAPIで簡単にアクセスしたい。

追記機能

■ 追記機能によるデバッグの効率化

- 数Gバイトを超えるパッキングには数時間必要。
- 追記機能によりターンアラウンド時間を短くでき、効率良くデバッグできる。
- α版やβ版などの納期間際には重宝。

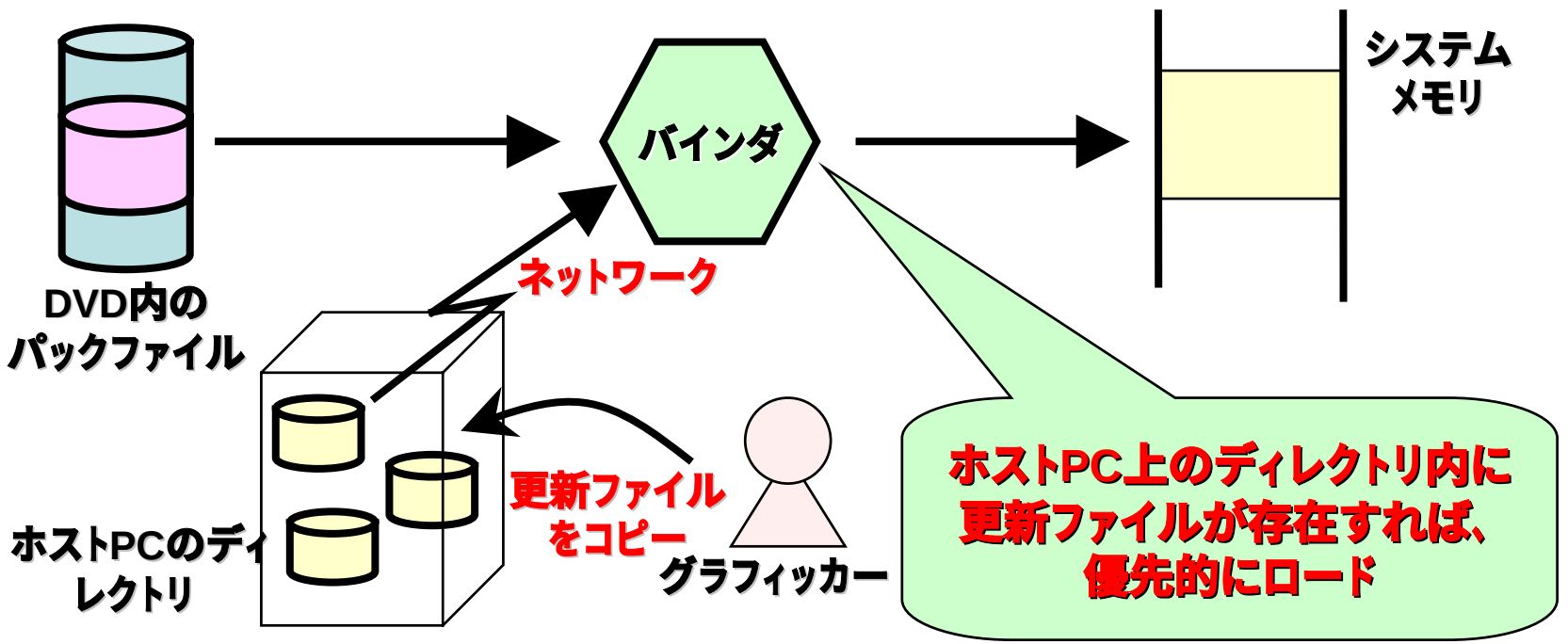
<追記の仕組み>



マルチバインド機能によるデバッグの効率化

■ 更新された個別ファイルの自動ロード

- ディスク上のパックファイルとホストPC上のディレクトリをマルチバインド。
- プログラムを変更せずに、リアルタイムにデータの差し替えができる。

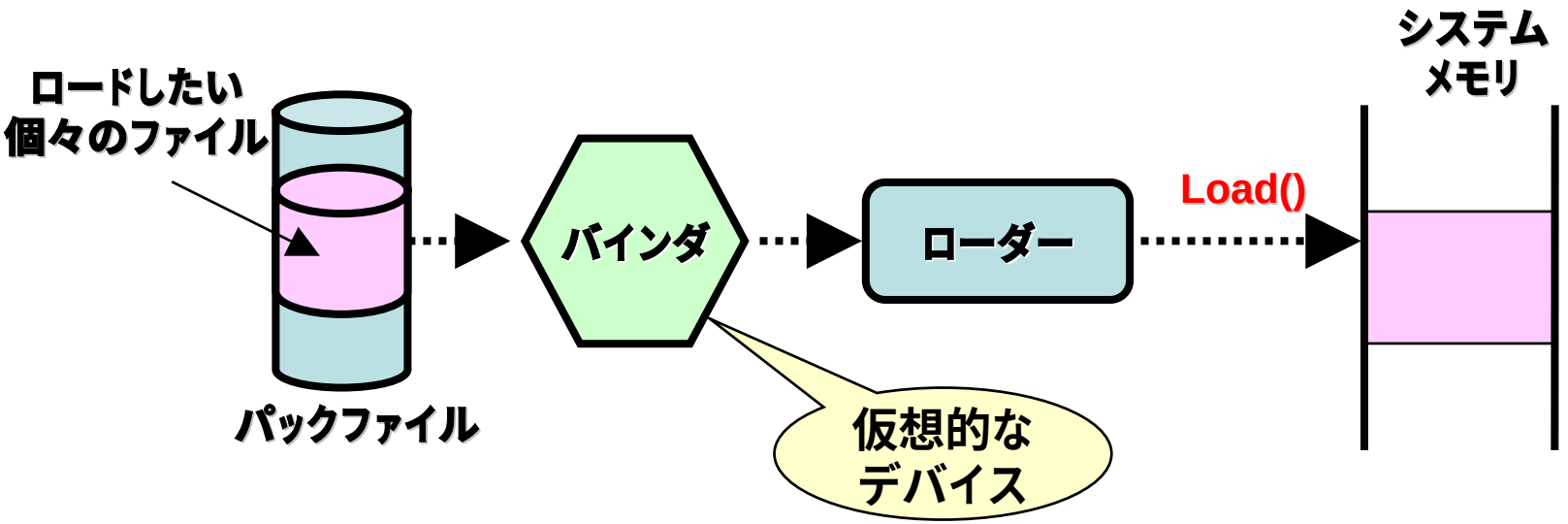


バインダとマルチバインド1

■ バインダ

- プログラマは、パックしたファイル内の個々のファイルをロードするためにバインダとファイル名を指定する。

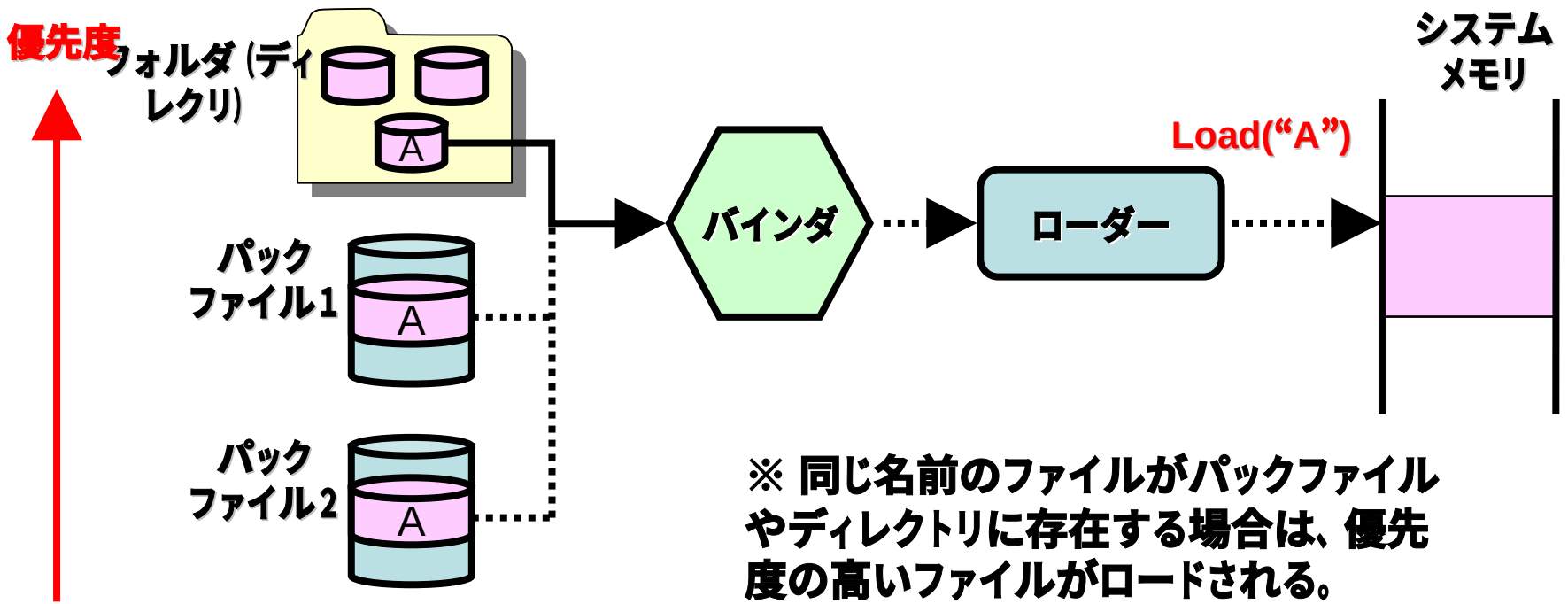
Load（ローダー, バインダ, ファイル名, 読み込み先バッファ）;



バインダとマルチバインド2

■ マルチバインド

- 複数のパックファイルやプラットフォームのフォルダ（ディレクトリ）をバインド。



※ 同じ名前のファイルがパックファイルやディレクトリに存在する場合は、優先度の高いファイルがロードされる。

バインダとマルチバインド3

■ マルチバインドの効用

- 再パッキングせずにデータの更新が可能。
- ゲームプレイ中でもリアルタイムに差し替えることができる。
- ハードディスクやメモリースティック上のパックファイルやディレクトリをバインドすることによって、ロード時間を短縮。
- パックファイルを分割しておくことによって、パッキング時間を短縮。
- 2層のメディアであっても、ひとつのバインダで管理できる。
- RAM上のパックファイルをバインドすることによって、RAMディスクのような機能を実現可能。

パッキングファイルフォーマットの工夫

■ アライメント調整

- セクター（2048バイト）のアライメント。
- DMA (4, 32, 64, 512バイトなど)によるアライメント。
- より小さくしたい場合は1バイトアライメント。

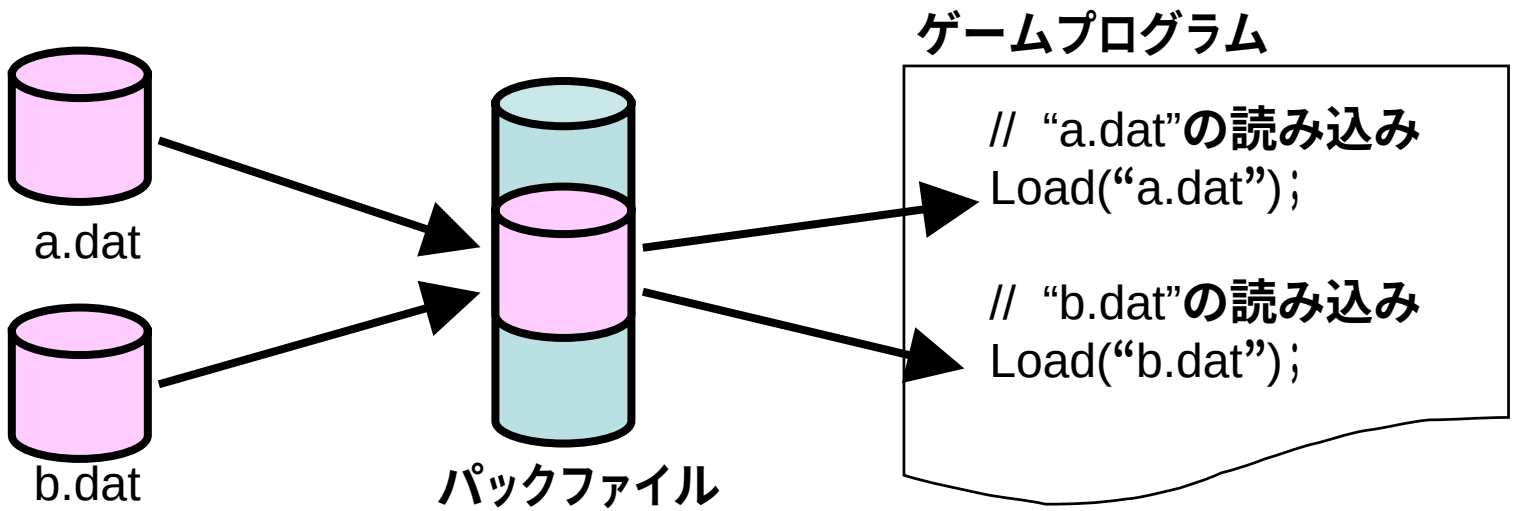
■ ファイルマジックPROのディレクトリ情報フォーマット

- ファイル識別子によって 6or10バイト/ファイルでファイルを管理可能。
- 64Kバイト超えるファイルの場合は、ファイル管理情報が自動的に10バイト/ファイルとなり、管理用の領域をコンパクトに抑える。
- 展開後のファイルサイズを保持するため、事前にオリジナルのファイルサイズを取得できる。

重複データの一元化機能

■ 異なるファイルであっても重複データを一元化

- ファイル名が異なっていても、内容を比較する。
- 内容が同じであれば、パックファイル内では実体は1つしかもたない。
- ランタイムライブラリでは異なるファイル名で同一データをアクセス。



“a.dat”と”b.dat”の内容は同じ

CRIWARE®

ファイル配置の最適化

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

シークの問題点

■ シーク時間によるロード時間の増加

- 100～500ミリ秒と非常に遅い。(ハードディスクの10倍以上)
- 50ファイル読むと平均150ミリ秒で7.5秒失われる。

■ 光ディスクドライブの実力

- シークせずにリードするとドライブキャッシュの効用によって、
  ピックアップは最高速で読み続ける。
- ハードディスクは、FATなどで管理されるためシークが発生。
- 最高速で読み続ける場合、ハードディスクの約50%ぐらい。
- 圧縮技術を併用すると、80%ぐらいに肉迫。

ファイル配置によるロード時間の短縮

■ ファイルアクセスログからファイル配置を調整

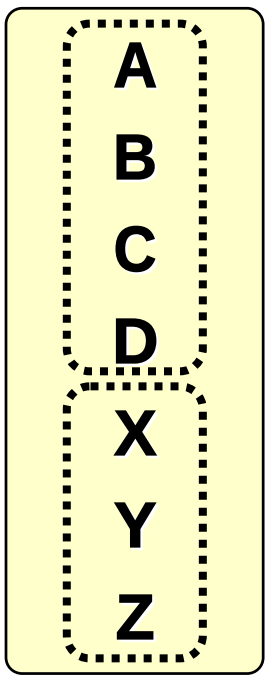
- ロード時間の大敵である「シーク時間」を削減。

単純な配置

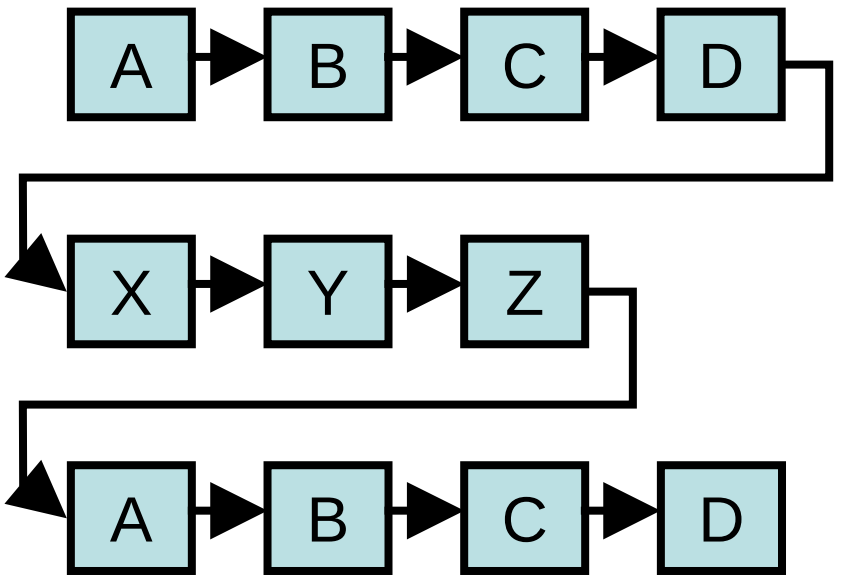
■ ファイルのグループ化

- ファイルをグループ化することで、だいぶシーク時間を軽減できる。

ファイルリスト



アクセスログ



グルーピングによるファイル配置

■ シーンやキャラクタなどの複数データをグループ化

- グループ定義情報に従ってファイルを近接配置
  ➔ アクセスログからグループを自動的に定義

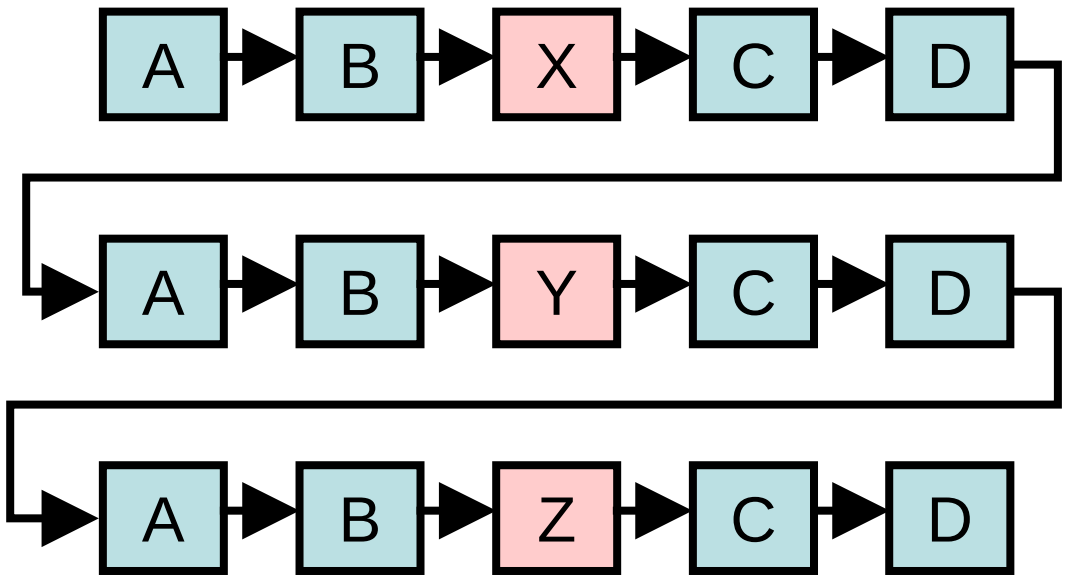
問題1

■ 下記のログから最適な配置を考えてみましょう。

ファイルリスト

A
B
C
D
X
Y
Z

アクセスログ



File MANA ： File Majik Analytics (仮)

■ 人工知能

- 天文学的な組み合わせをヒューリスティックに探索すること。
- 適用例：チェス・オセロ・将棋など

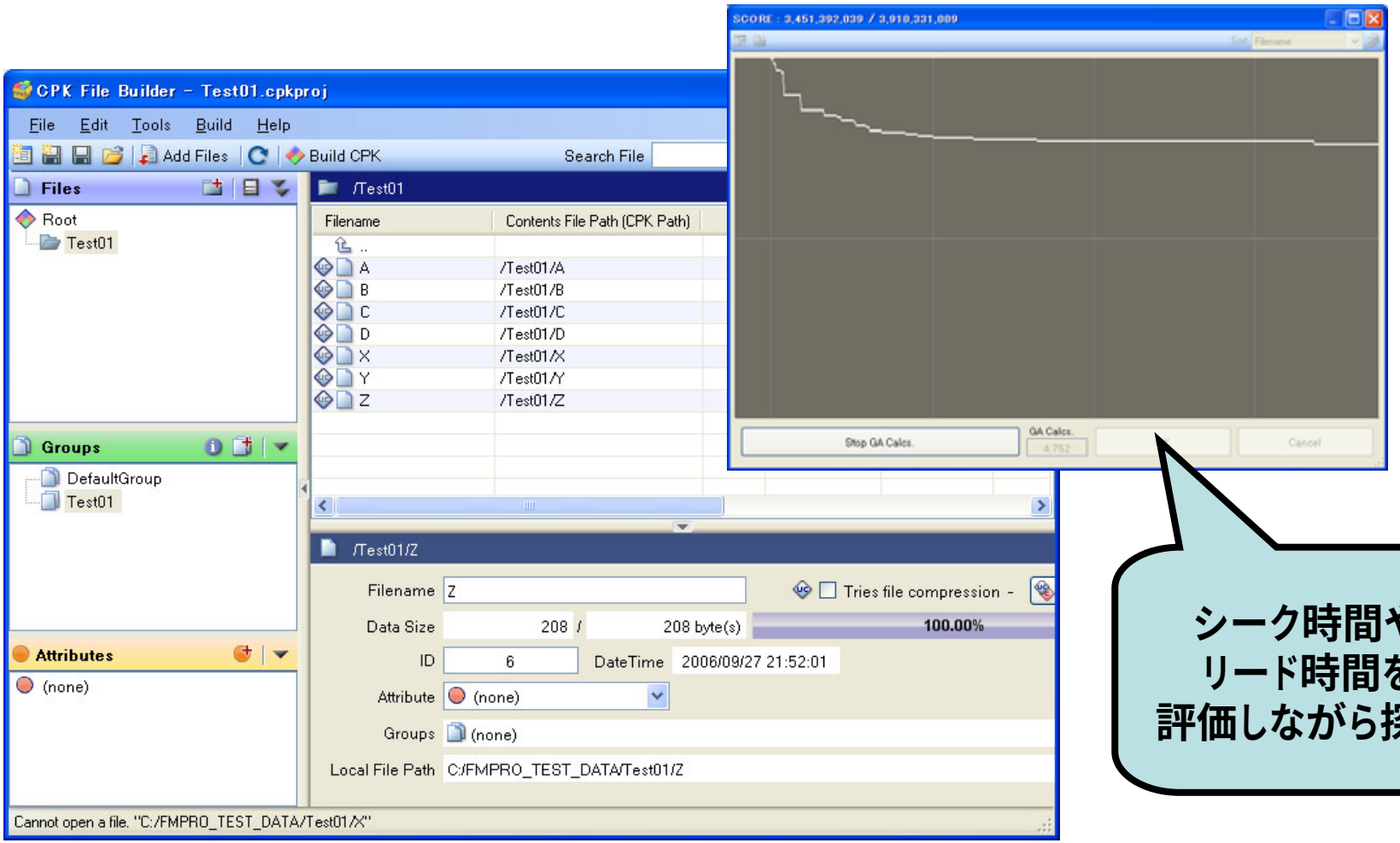
■ ファイルの並べ方は何通り？

- 少なければ総当りでも大丈夫。
- 「順列」、覚えていますか？階乗です。
- 例えば、1万個のファイルの並べ方は

  $_{10000}P_{10000}$ = 約300…(0が35659個) … 通り
- なにかしらの人工知能てきなアプローチが必要。

  ➔ File MANA (ファイル・マナ)

File MANA のデモンストレーション



シーク時間や
リード時間を
評価しながら探索

解答1

■「ABCDXYZ」の場合

- AB➔X➔CD➔AB➔Y➔CD➔AB➔Z➔CDで8回です。
  (「➔」はシークを表します。)

■「ABXCDYZ」の場合

- ABXCD➔AB➔Y➔CD➔AB➔Z➔CDで6回です。

■「YCDABXZ」の場合

- ABX ➔CDAB➔YCDAB➔Z➔CDで4回です。(正解)

※ 「ZCDABXY」などX、Y、Zが「CDAB」の外側にあれば同じです。

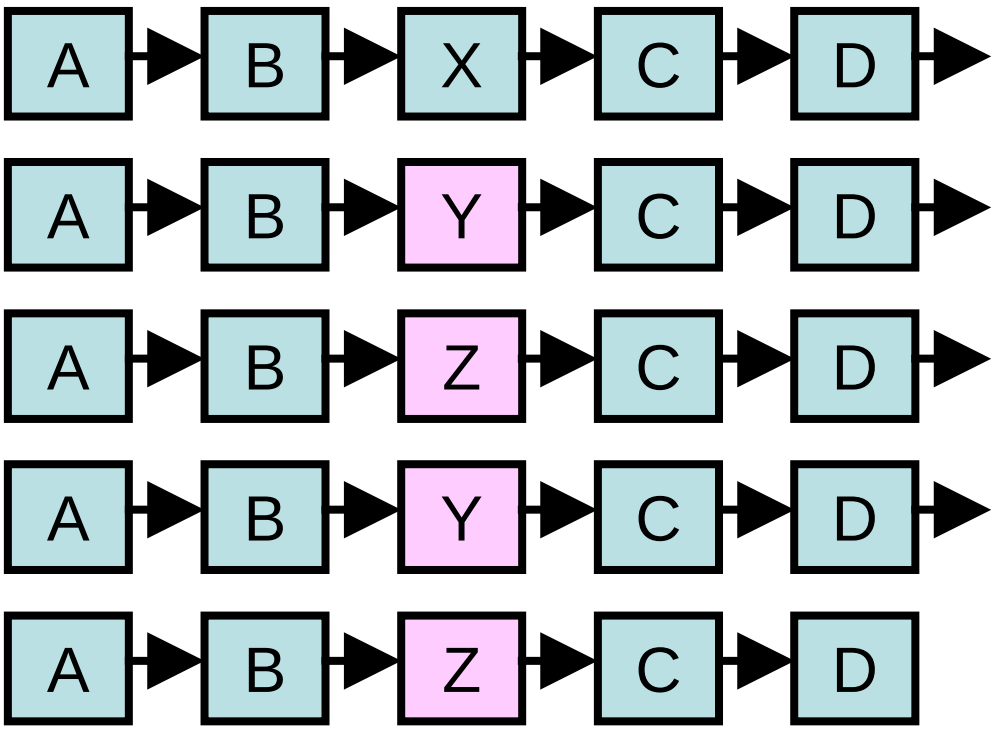
問題2

■ 下記のログから最適な配置を考えてみましょう。

ファイルリスト

A
B
C
D
X
Y
Z

アクセスログ



解答2

■「YCDABXZ」の場合

- ABX →CDAB→YCDAB→Z→CDAB→YCDAB→Z→CD

で7回です。

■「YCDABZX」の場合

- AB→X →CDAB→YCDABZ→CDAB→YCDABZ→CD

で6回です。 (正解)

ユーザー視点でのロード時間

■ ユーザーの体感するロード時間

- 10秒が5秒になると短くなると体感できる。
- 1秒が0.5秒になっても「サクサク」感は上がるが、ロード時間が短縮したとは感じない。

■ ピンポイントでのチューニングが必要

- めだって長いロード時間をピンポイントで短縮するとより効果的。

■ 使用頻度の高いキャラクタやステージに着目

- 人気のあるキャラクタや最初の方のステージのロード時間を短くする。
- 10万人のプレーヤの総ロード待ち時間が短くなる。

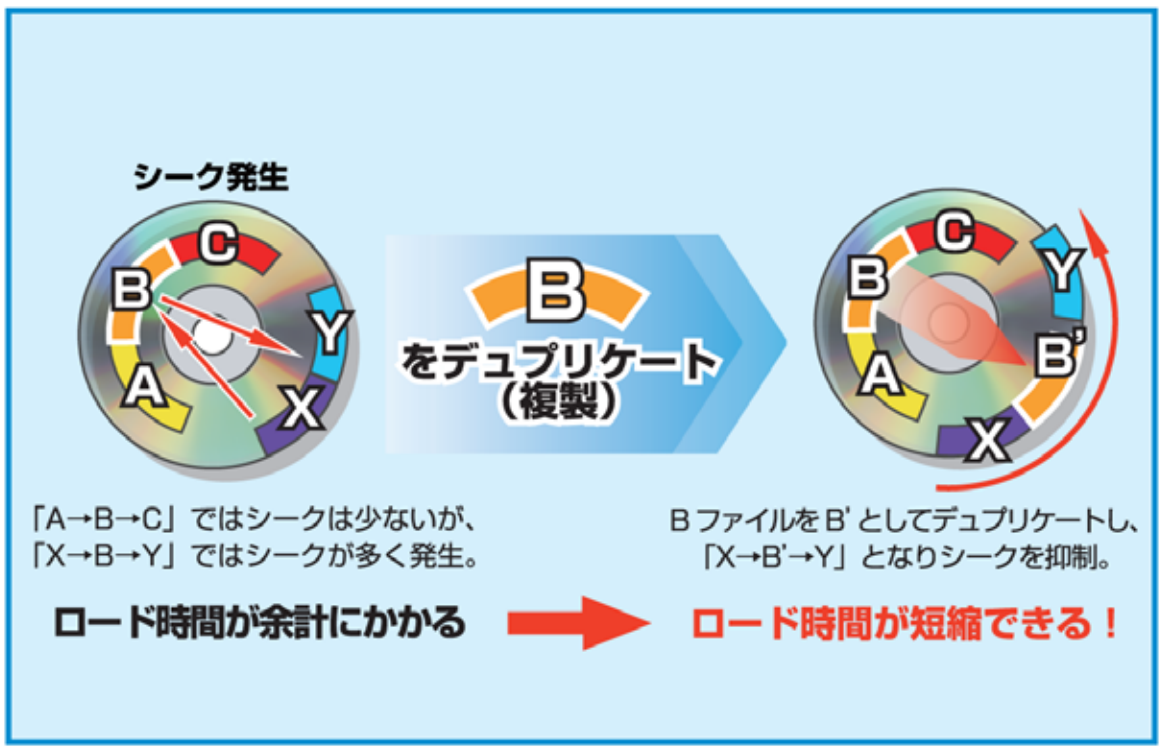
■ ゲームの盛り上がりを意識してロード時間を調整

- ボスキャラ退治後のロードはあまり気にならない。
- 盛り上がる手前でロードはNG。

デュプリケート機能によるさらなるロード時間の削減

■ 複数のグループに属するファイルを自動的にデュプリケート

- メディア上の空き領域を利用してロード時間を削減。

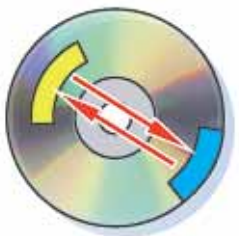


CRIWARE®

圧縮によるロード時間の短縮

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

ロード時間を短縮するには



ロード ＝ リード ＋ シーク

圧縮

リード時間の短縮化 ＝ 読み込むファイルのサイズを減らす → ファイルの「圧縮」+「展開」

シーク時間の短縮化 ＝ 読み込むファイルを連続的に配置する → ファイルの「最適配置」

圧縮技術を導入するための課題1

■ 圧縮率
- 可逆圧縮なので40～60%程度が目安。

■ 展開速度
- 読み込み時間よりも短時間で展開できる必要。

■ ワークサイズ
- 辞書などの展開処理に必要なワークメモリ。
- 圧縮データを一時的に保持するテンポラリメモリ。

■ スレッドアーキテクチャ
- ドライブからの読み込み処理は、メインスレッドより高くする。
- デコード処理は、ゲーム処理の空き時間に行うために、メインスレッドよりも低くする。

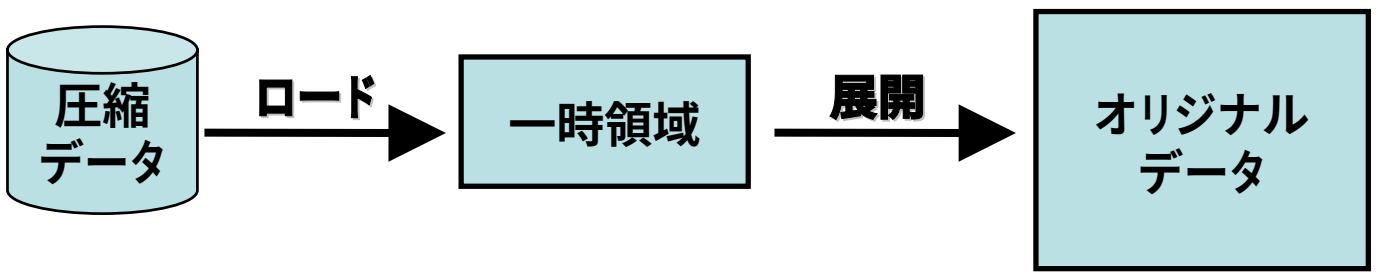
圧縮技術を導入するための課題2

■ **圧縮したデータを展開するための一時バッファが必要**
- 圧縮データを一時バッファにロード ➔ 目的の領域へ展開。

■ **圧縮してみないとサイズがわからない**
- 圧縮できない場合もある。（大きくなってしまうことも．．．）

■ **オリジナルのファイルサイズが知りたい**
- データ領域を何バイト確保するかをロードする前に知りたい。

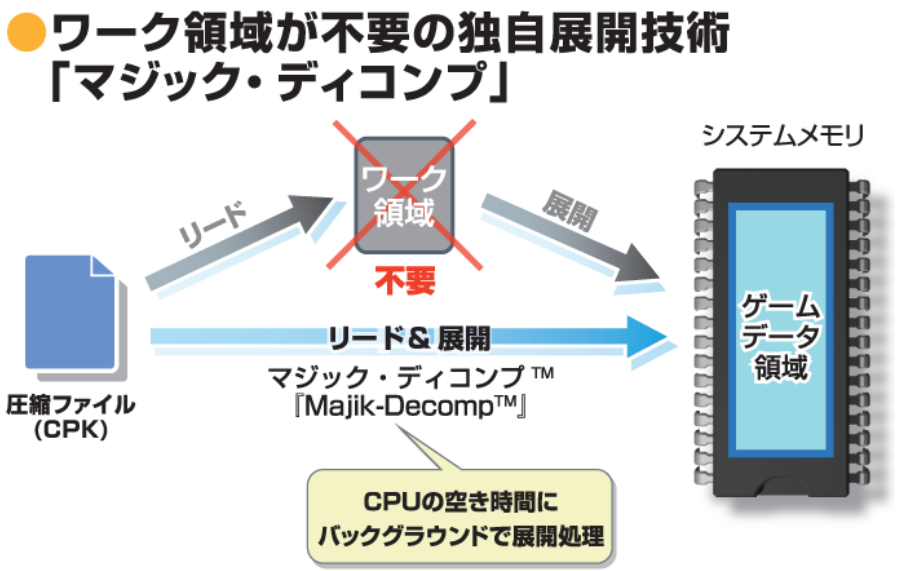
一般的な圧縮データの展開方法



『ファイルマジックPRO』による圧縮・展開

■ **展開時に一時バッファを必要としない『自己エリア展開』**
- 展開用の一時バッファを使用せずに、ロードされた領域で展開。

■ **プログラマは圧縮されているかどうかを意識せずにロード**
- Load関数を呼ぶだけで、圧縮されているデータは自動的に展開される。

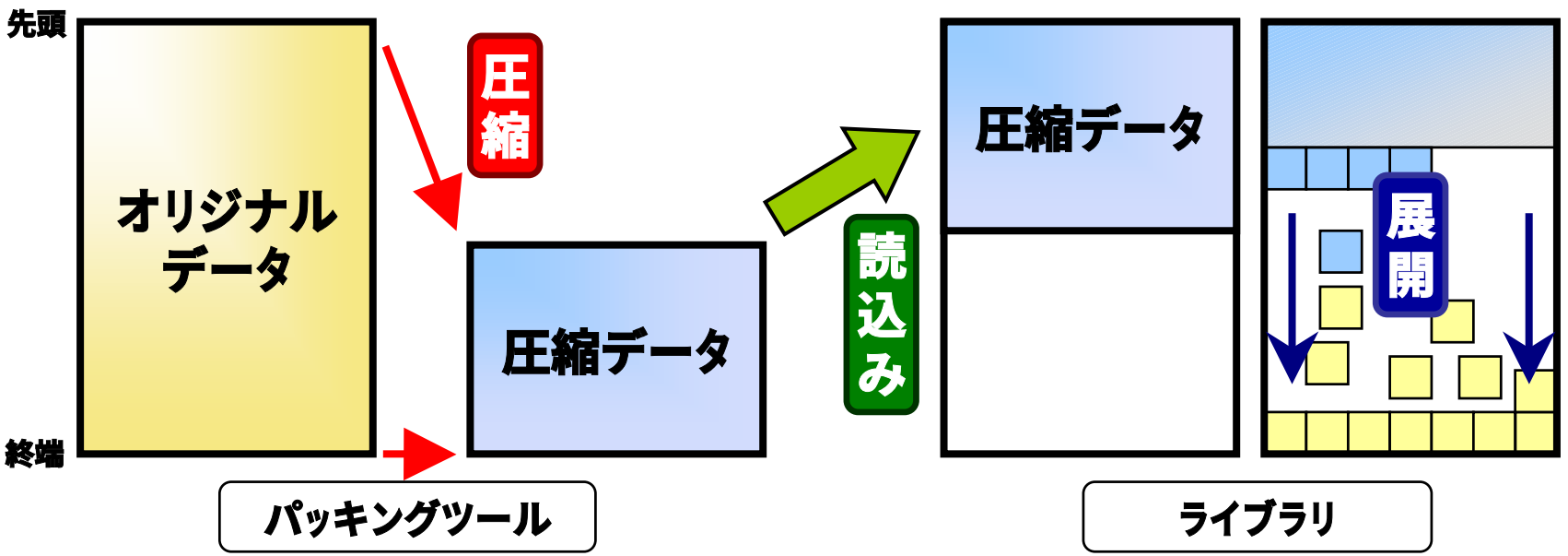


マジック・ディコンプ™のアルゴリズム1

■ ファイル後方より圧縮・展開

- ファイルデータの終端から圧縮。ロードしたデータは後方から展開。

➔ 高速な『自己エリア展開』を実現。

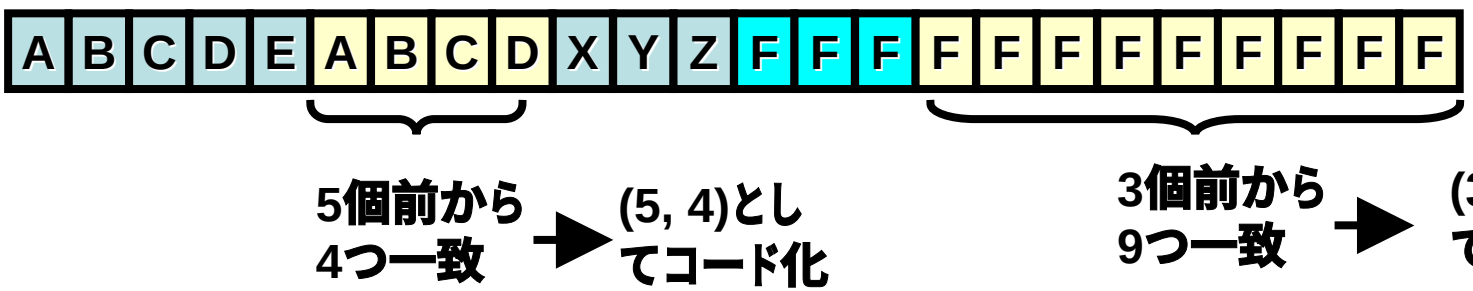


マジック・ディコンプ™のアルゴリズム2

■ スライド辞書による圧縮

- 13ビットのウィンドウ長　(一致する文字列の探索範囲:8195バイト)
- 一致長はVLC (可変長コード) を使用。最低一致長は3バイト。
- ZIPやLHAは、圧縮できなかったデータや一致長をハフマン圧縮
  ➔ 辞書領域が必要になってしまうことと、処理が遅くなってしまうので
  ハフマン圧縮はしていない。圧縮率も数パーセント程度の低下。
- ZLIBの2倍程度の展開速度。(Xbox360で測定)

スライド辞書法の基本原理

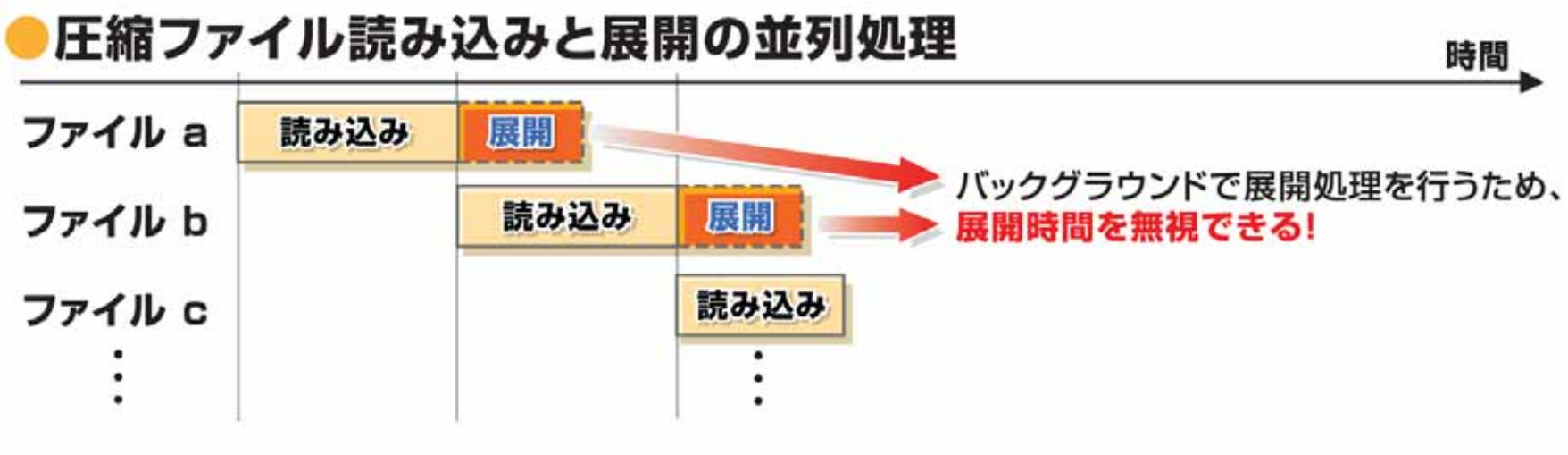


「読み込み」と「展開」の並列処理

■ 「読み込み」と「展開」をオーバーラッピング

- 展開処理を別スレッドで空き時間に実行。
- ファイル単位で「読み込み」と「展開」をオーバーラッピング

➔ **展開時間をほぼゼロに！**



圧縮によるリード時間の削減

■ 可逆圧縮によるリード時間の短縮

- ゲームデータは約40％ぐらいに圧縮可能 → リード速度が2.5倍に！

●ファイル圧縮によるファイルロード時間短縮の効果

プラットフォーム	メディア	ロードサイズ	非圧縮	圧縮（ファイルマジックPRO）	
Wii	DVD	50MB	10.9 秒 →	4.6 秒	2.4倍
Wiiウェア	Wii 本体保存メモリ	20MB	2.3 秒 →	1.1 秒	2.1倍
Xbox360	DVD	450MB	36.3 秒 →	15.5 秒	2.3倍
PLAYSTATION 3	ブルーレイディスク	200MB	22.8 秒 →	9.9 秒	2.3倍
PSP	UMD	20MB	11.1 秒 →	5.0 秒	2.2倍

圧縮による読み込みファイルサイズの最小化、リード&展開処理のオーバーラップ化により、ロード時間を大幅短縮！

2倍以上に改善！

※画像ファイル（BMP）を読み込んで測定。
ファイルマジックPRO使用時は、元ファイルを 43%に圧縮。

※非圧縮／圧縮とも、10 個のファイルを一括読み込みして計測。
表内のロードサイズは 10 ファイルの合計値。

※ロードサイズは、各プラットフォームの RAM 容量に合わせて設定。

※Wii と PSP は DVD-R、PLAYSTATION3 は BD-RE、Xbox360 はエミュレータ上で計測。

※なお、オプティカルメディアでないニンテンドーDS の場合でも、約 1.4 倍の速度向上が見込めます。

CRIWARE®

ランタイムライブラリによる高速化

CRI Middleware Co., Ltd.
Audio & Visual Solution Company

ランタイムライブラリの設計目標

■ **非同期型API**
- **裏読み**が簡単に実現できる使いやすいAPI。

■ **パックしたファイルとパックしていないファイルの切り替え**
- パックしたファイルも個別のファイルも同じAPIでロード。

■ **圧縮されたファイルか否かを区別せずにロード**
- プログラマが圧縮を意識せずにロードできる。

■ **スレッドアーキテクチャ**
- マルチスレッド環境、シングルスレッド環境でも同じAPI。
- ユーザのマルチスレッドシステムにも高い親和性を持つ。

「HTTPプロトコル」とファイルロードAPI

■ fopen, fread, fcloseの問題

- 3ステップに分かれていると、状態遷移の管理が難しい。
- 例えば、fopen, fclose中にディスクが取り出された場合などの異常系の処理が複雑になる。

■ HTTPプロトコルのシンプルさ

- Getコマンド ➔ ひたすらリードするのみ。

■ ローダー

- バインダとファイル名でリードリクエスト ➔ ロードの完了チェック

```
// ロードリクエスト
Load(ローダー, バインダ, ファイル名, バッファ, バッファサイズ);
// 状態チェック
while ( GetStatus(ローダー) != COMPLETE )     // 完了チェック
    ;
```

ノンブロッキングファイルシステム

■ ノンブロッキングなロード関数

- ローダーを作って、ロードリクエストを発行。(Load関数)
- ブロックしないのでゲームプレイ中の裏読みが可能。
- ローダーに対して完了したか否かをチェック。

```
CriFsLoader ldr;                                    //  ローダー

ldr = CreateLoader();                               //  ローダーの作成
Load(ldr, NULL,“a.dat”,buf, bufsize);               //  ロード開始
while(GetStatus(ldr) != COMPLETE )                  //  ロード完了チェック
        ;
DestroyLoader(ldr);                                 //  ローダーの消去
```

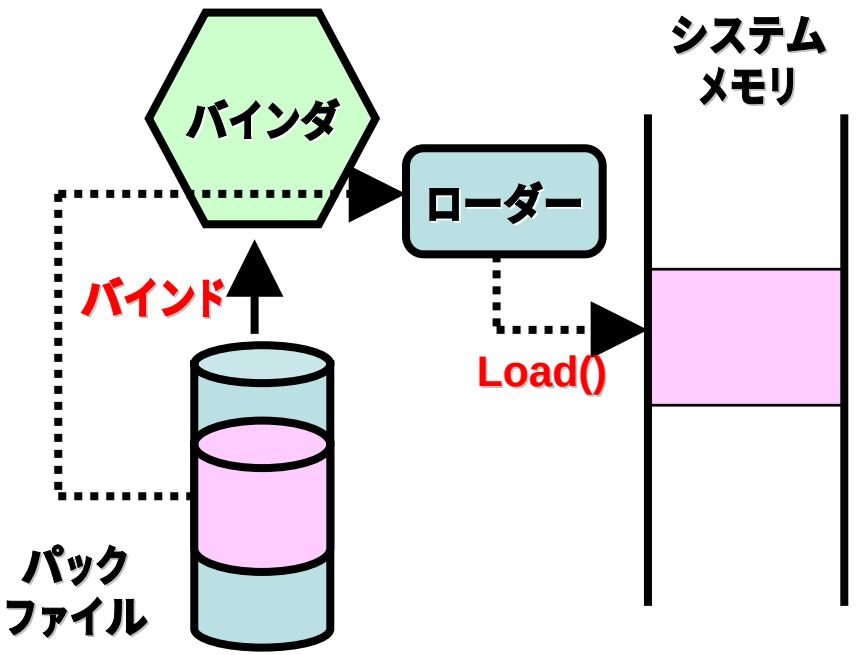
パッキングされたファイルの読み込み

■ パックファイルをバインダ（仮想的なデバイス）へ登録

- パックされたファイルをバインダに登録（バインド）。
- バインダとファイル名を指定してロード。

```
CriFsBinder bnd;  //  バインダ

//  CPKファイルのバインド
id = BindCpk(bnd, “game.cpk”);
//  バインド完了チェック
while(GetStatus(bnd) != COMPLETE )
        ;
//  バインダとファイル名によりロード
Load(ldr, bnd, “a.dat”,buf, bufsize);
//  ロード完了チェック
while(GetStatus(ldr) != COMPLETE )
        ;
```



パッキングしていないファイルの読み込み

■ バインダにNULLを指定

- パッキングしていないファイルも同じAPIで非同期にロード。
- バインダをNULLにすることで、プラットフォームのファイルシステムから直接、個々のファイルをロード。

＜パッキングしたファイル＞

```
Load(バインダ, ファイル名, buf, bufsize);
```

＜パッキングしていないファイル＞

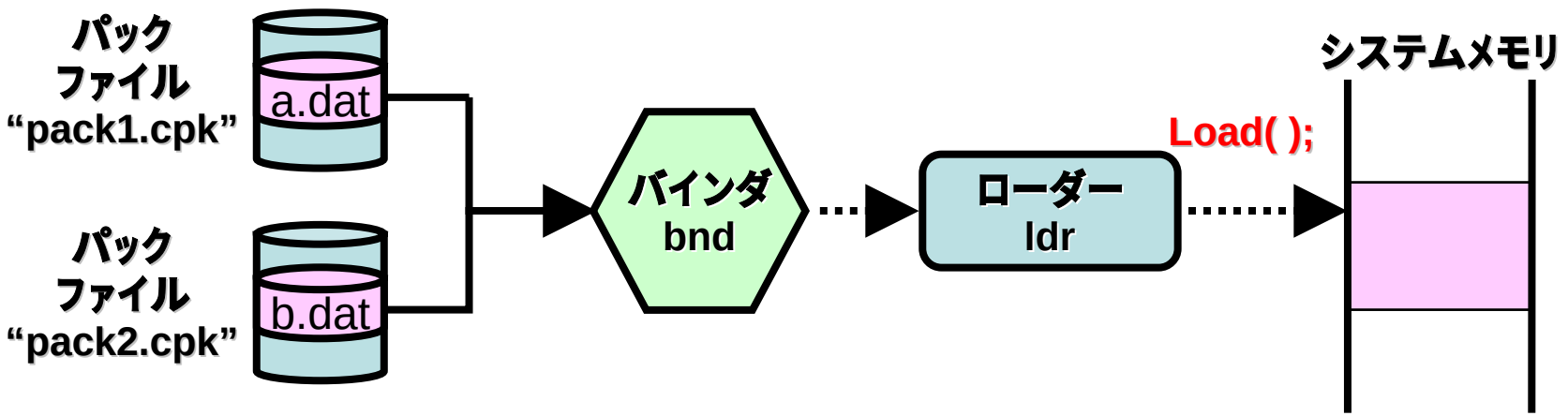
```
Load(NULL, ファイル名, buf, bufsize);
```

マルチバインド1

■ 複数のパックファイルからのロード

- バインダに複数のパックファイルをバインド。
- バインダは、バインドされた複数のパックファイルを検索しロード。

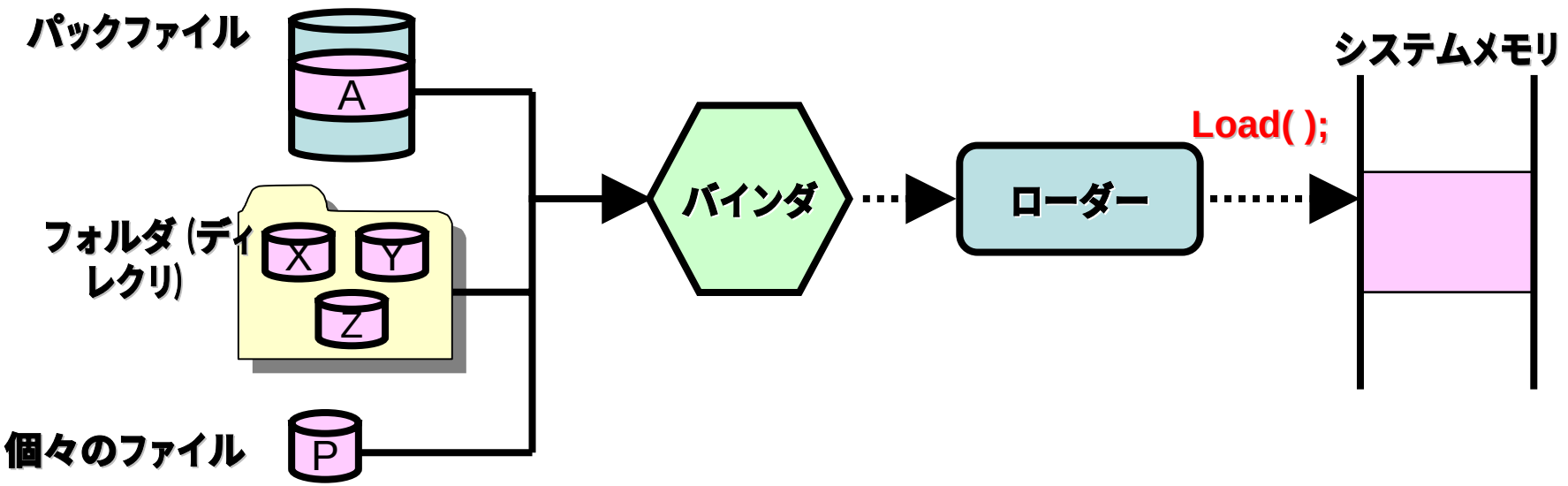
```
BindCpk(bnd, null, “pack1.cpk”);      //  bndにpack1.cpkをバインド
BindCpk(bnd, null, “pack2.cpk”);      //  bndにpack2.cpkをバインド
Load(ldr, bnd, “a.dat”, buf, bufsize); //  pack1.cpk内のa.datをロード
Load(ldr, bnd, “b.dat”, buf, bufsize); //  pack2.cpk内のb.datをロード
```



マルチバインド2

■ バインド対象とその動作

1) パックファイル　　ディレクトリ情報の読み込み常駐させる。
2) フォルダ　　ファイルオープン時の検索対象となる。
3) 個々のファイル　　ファイルをオープンしておき、ロードの負荷を下げる。

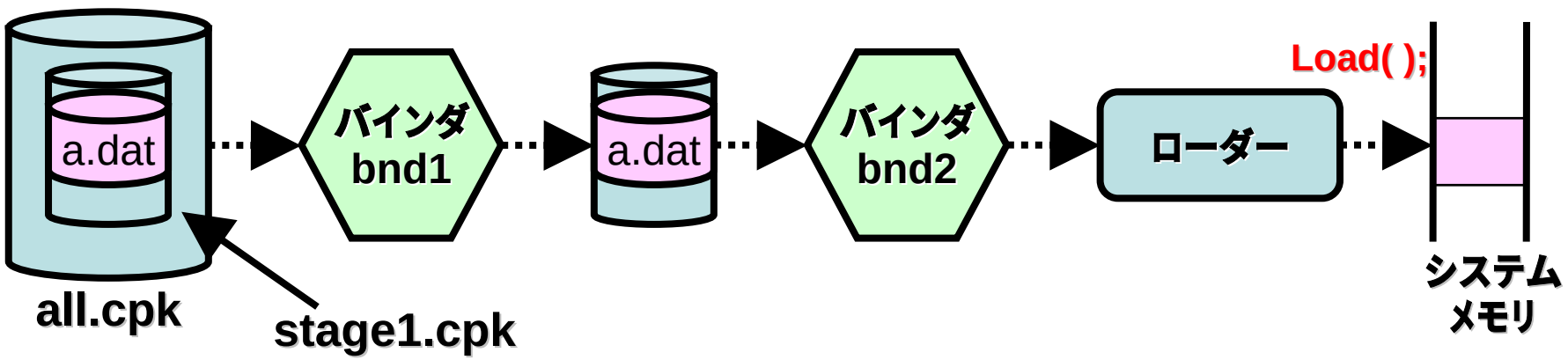


マルチバインド3

■ バインダ内ファイルのバインド

```
Bind(bnd1, NULL, “all.cpk”);          // all.cpkは独立したファイル
Bind(bnd2, bnd1, “stage1.cpk”);       // stage1.cpkはall.cpkの中
Load(ldr, bnd2, “a.dat”);
```

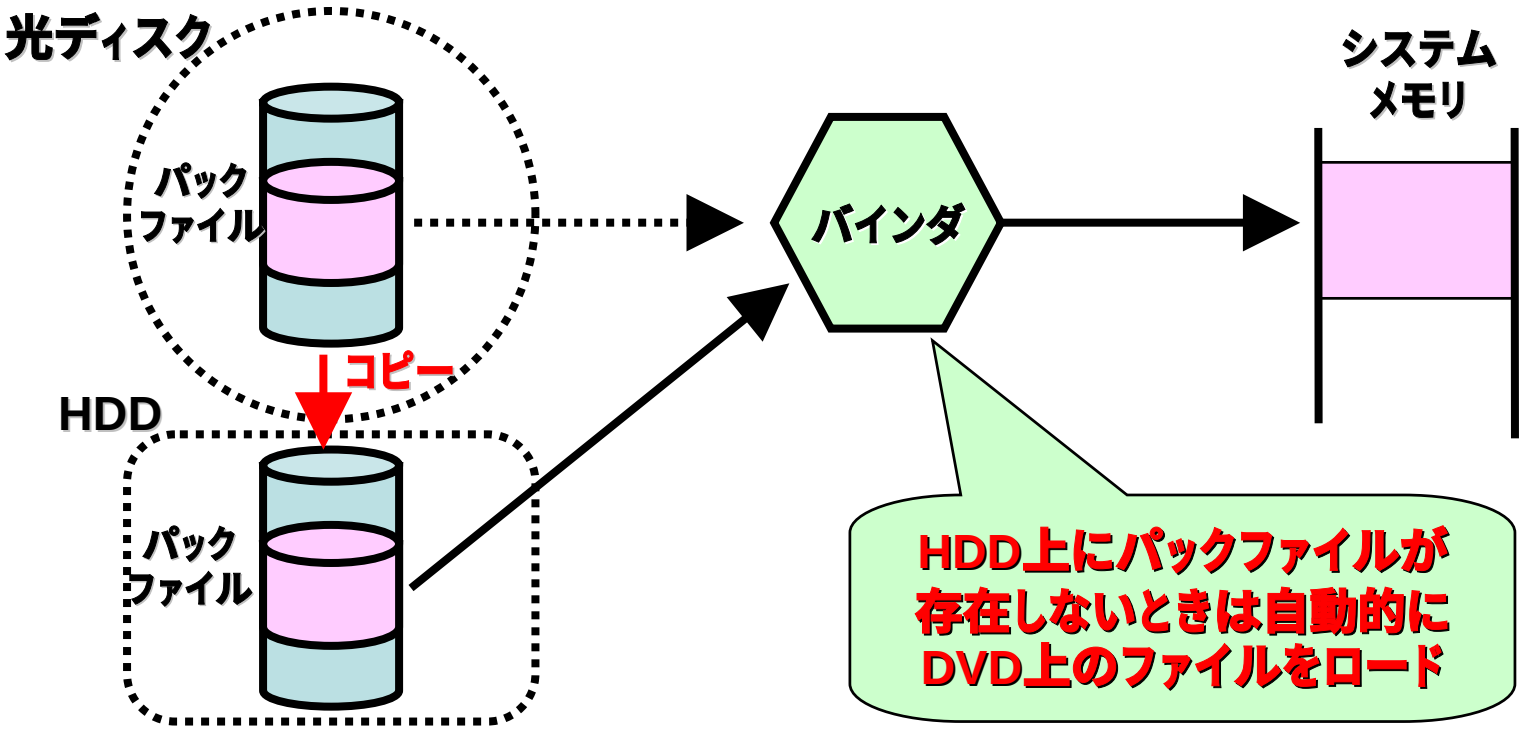
- 入れ子構造が持てることによって、様々なデータをパッキング可能。
- 単一ファイル、フォルダ、メモリを組み合わせることによって、
  非常に柔軟なデータ管理ができる。



ファイルインストール機能による読み込みの高速化

■ 高速な補助記憶装置へパックファイルをコピーしロードを高速化

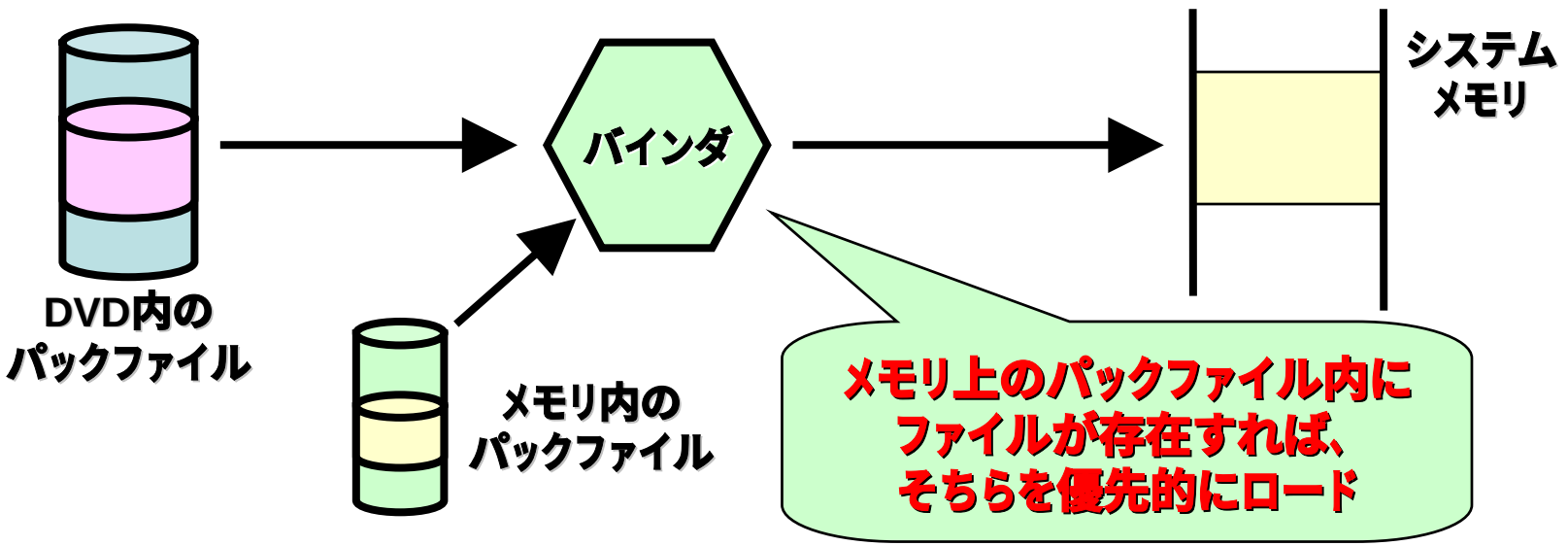
- ハードディスクやメモリースティックにパックファイルをコピー
- パックファイルが存在しないときは、ディスク上のデータをロード。



RAMディスク機能による高速化

■ メモリ上にパックファイルを常駐

- 使用頻度の高い小さめなファイルをメモリに常駐。
  ➔ シークが減り、ロード時間が大幅に改善。
- 「マルチバインド機能」により、プログラムを変更せずにメモリの空き容量に合わせて調整可能。優先的にロードするパックファイルを指定できる。



複数ファイルのロード

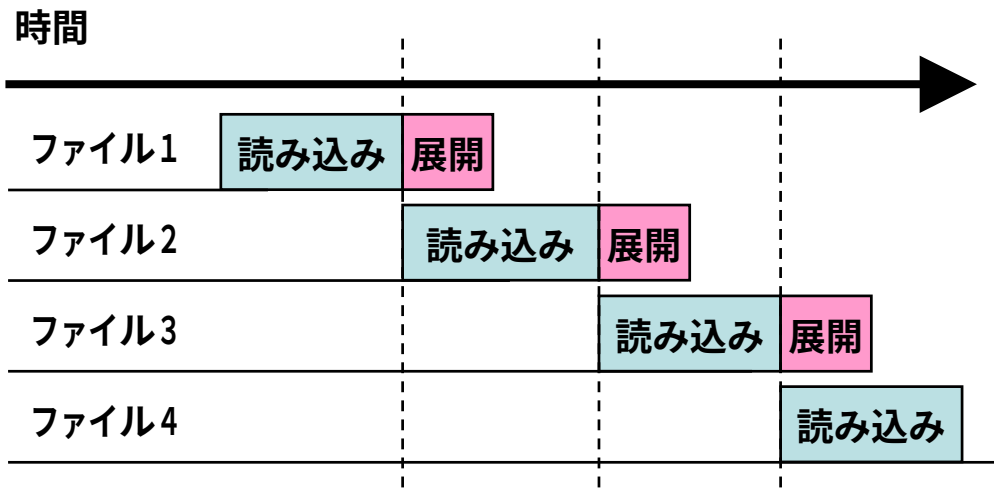
■ 複数のローダーにロード関数を実行

- 一時バッファを必要としない「マジック・ディコンプ」の効果により、複数のファイルに対して一度にロード関数を実行可能。
- ディスクからのリードと圧縮データの展開が並行動作。

```
//  ロード開始
Load(ldr1, “a.dat”,buf1, bsize1);
Load(ldr2, “b.dat”,buf2, bsize2);
Load(ldr3, “c.dat”,buf3, bsize3);
Load(ldr4, “d.dat”,buf4, bsize4);

//  ロード完了チェック
while(IsAllComplete() != COMPLETE )
        ;
```

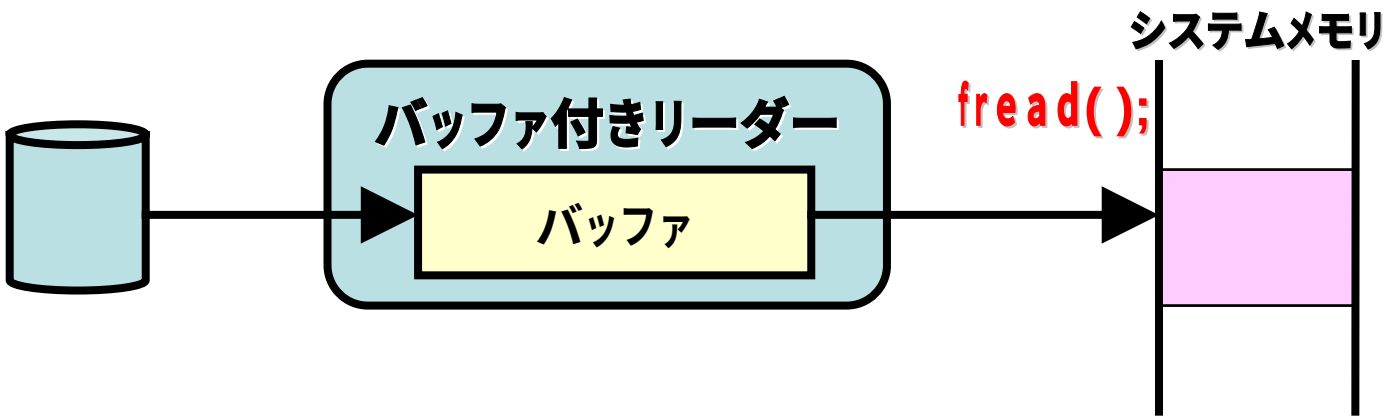
＜読み込みと展開のオーバーラップング＞



バッファ付きファイルリーダー（ANSI標準入出力API）

■ fopen, fread, fclose的APIの実装

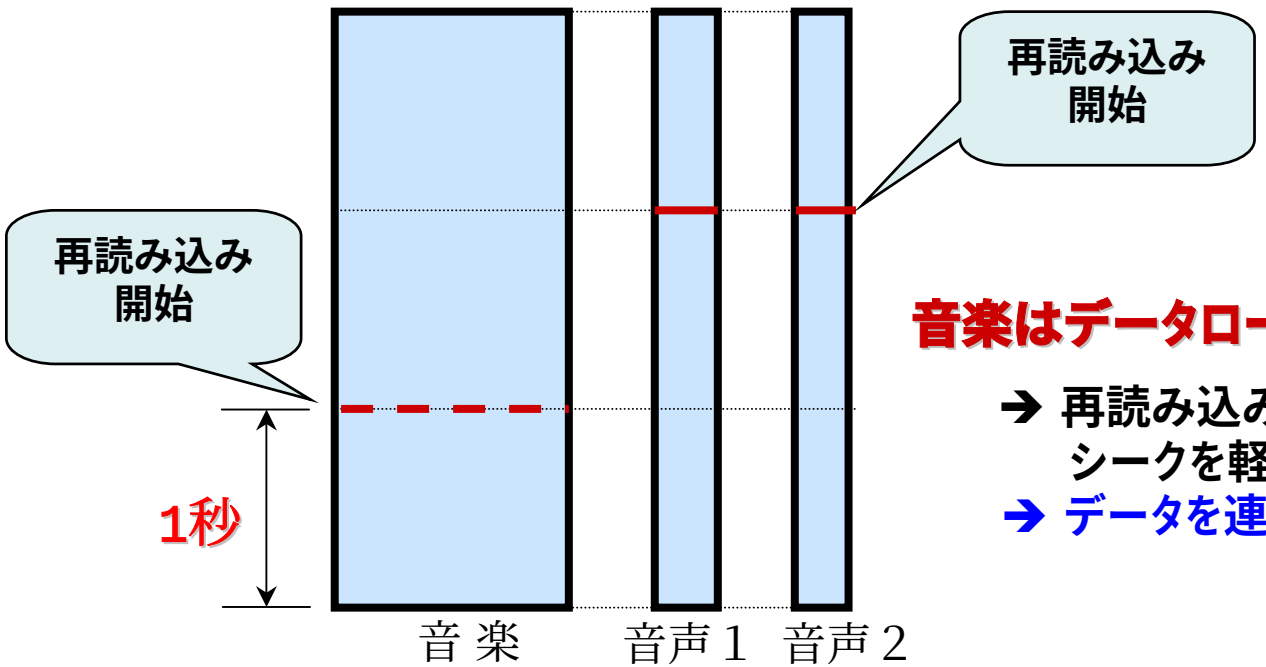
- ファイルフォーマットを解析しながらロードする場合は、バッファ付きファイルリーダーが必要。
- 光メディアの場合は、**バッファサイズを大きめ**にする。（256Kバイトぐらい）
  ➔ 小さいとドライブに対しリードリクエストがかからないことがある。



ストリーミング再生中のデータロード

■ ADXのストリームバッファの管理

- 500Kバイト程度（10秒ぐらい）のバッファを用意。
- 音楽用バッファの閾値を低く（1秒程度）に設定。



音楽はデータロード中も再生されている。

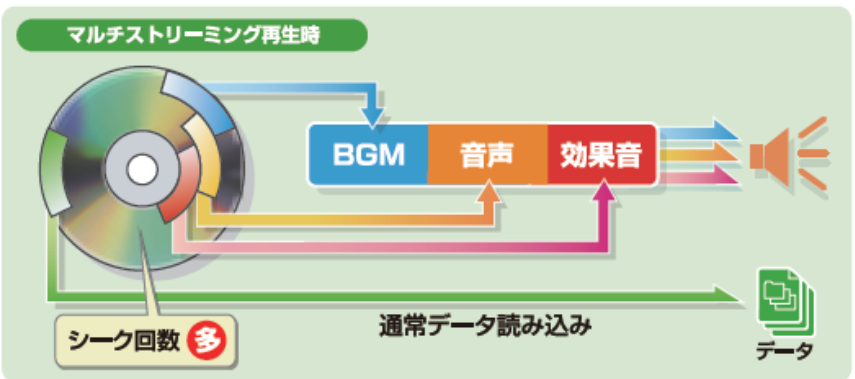
➔ 再読み込み開始の閾値を下げて、
シークを軽減できる。

➔ データを連続して読み込める。

ストリーミングバッファの効率的な利用

■ D-BAS™ (ダイナミックバッファアロケーションシステム)

- 音楽だけを再生しているときは、すべてのバッファを音楽用に割り当てる。
  - ➔ データの読み込みを途切れを少なくし、ロード時間が短縮。
- セリフや効果音が再生されたときは、音楽用バッファを分割して割り当てる。

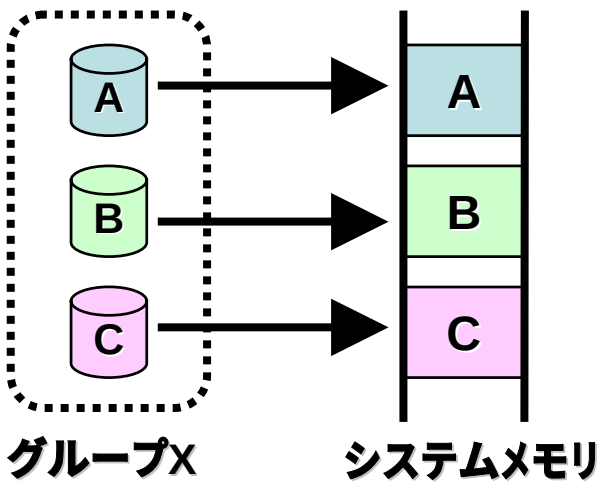


グループロード機能

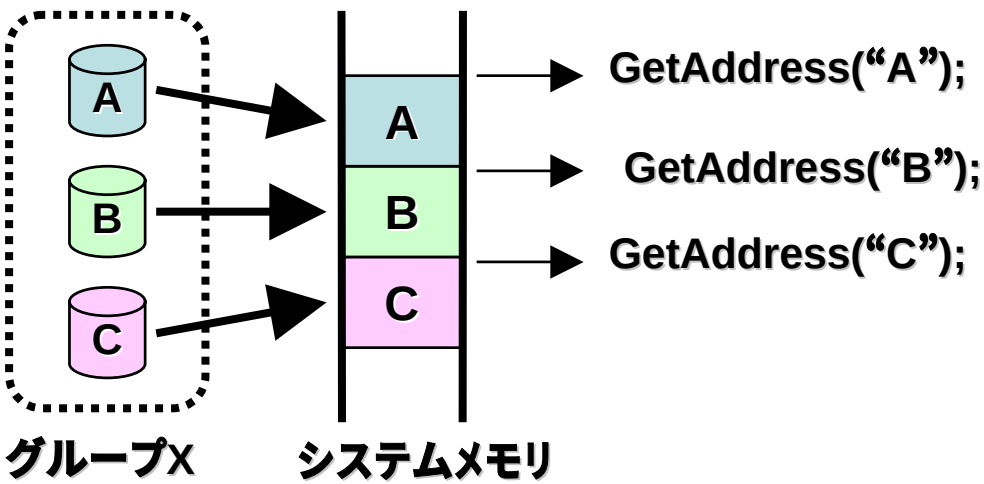
■ グループ名により複数のファイルを一括ロード

1) 各ファイル名と個々のロード領域を指定。
2) 一つの領域に一括ロードした後に、ファイル名からアドレスを取得。
3) 各ファイルをロードする前にコールバックがかかり、コールバック関数から逐次ロードアドレスを指定する。

1) 個々のロード領域を指定



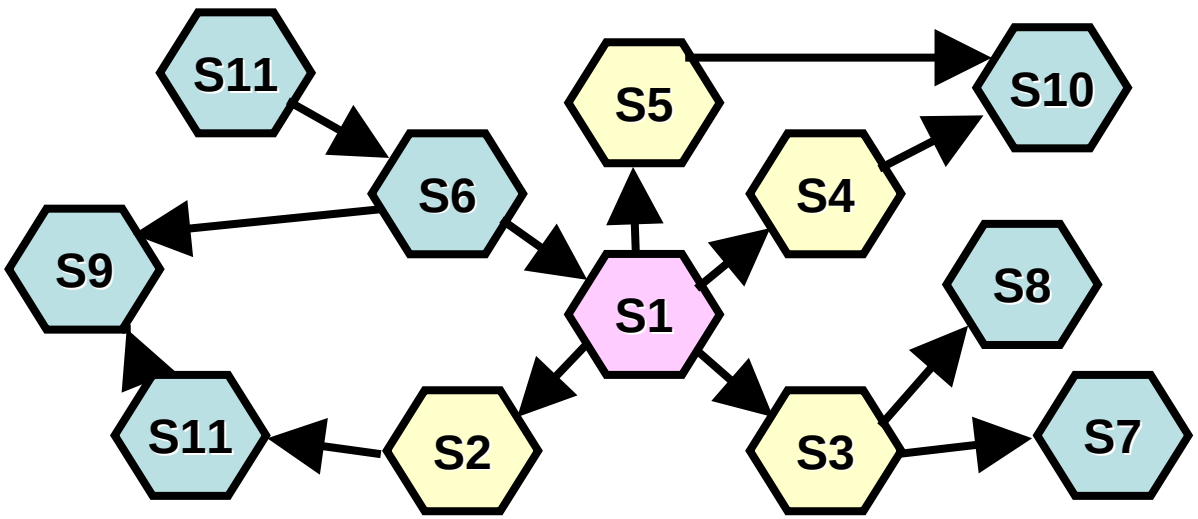
2) ファイル名からロードアドレスを取得



「Now Loading」撲滅のために

■ ゲームステートによるロード管理

- シームレスなゲームはメモリ管理（フラグメンテーション）との戦い。
- ゲームステートの遷移図から、近接するステートデータを先読み。
- ステート毎に使用するデータをグループ化 ➔ **グループロード**



※Googleマップのように現在Ｓ１にいるのであれば、周辺のデータをロード。ステート毎にメモリを割り当てて、フラグメンテーションを起こさないようにする。

まとめ

- 光メディアドライブはクセがあるので一筋縄ではいきません。
- とりあえずパッキングしましょう。
- グラフィッカーさんやサウンド屋さんと仲良くしましょう。
- 配置を気をつけると、いろいろなご利益があります。
- 圧縮すると容量を稼げるだけでなくロード時間も短くなります。
- ロード設計もゲームプレーヤーの視点が大切。

より“濃密なプレイ”の楽しめるゲームを！

お問い合わせ先

お問い合わせは…

www.cri-mw.co.jp/inquiry