ファイル(2)

ファイルシステム

ファイルシステムの実現方法

ディレクトリ操作はあくまで概念の話

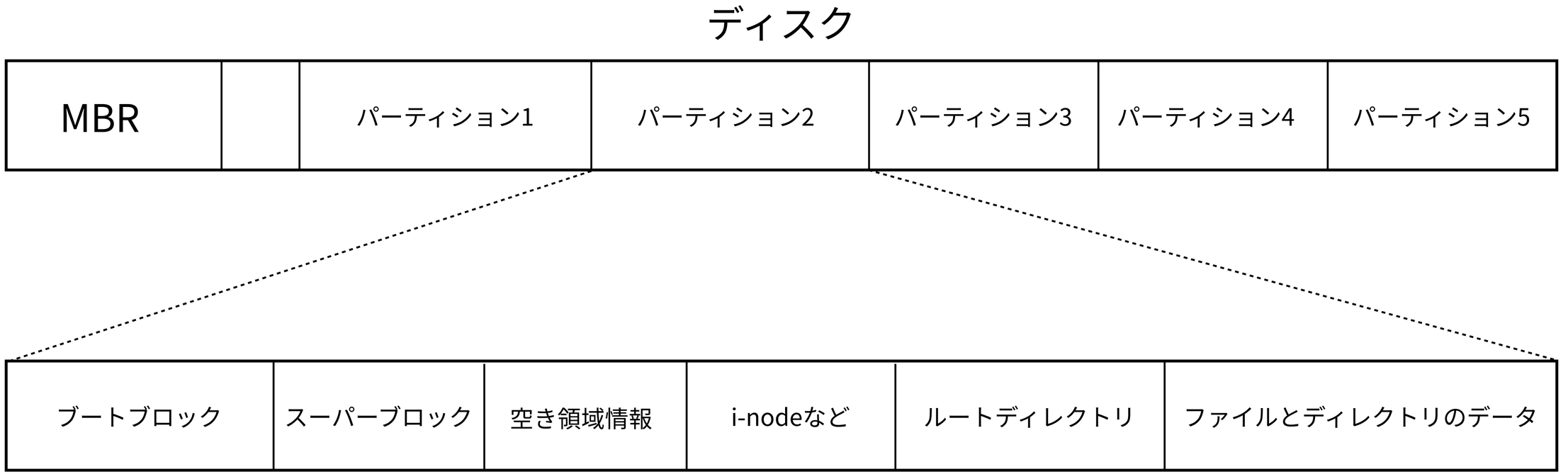
実際にデータを保存する場所は記憶装置（HDDやSSD、フラッシュメモリなど）

ファイルの保存

・補助記憶装置のメモリの値を書き換えてデータを保存

・ファイルは、データだけでなく構造も保存

補助記憶装置の中身

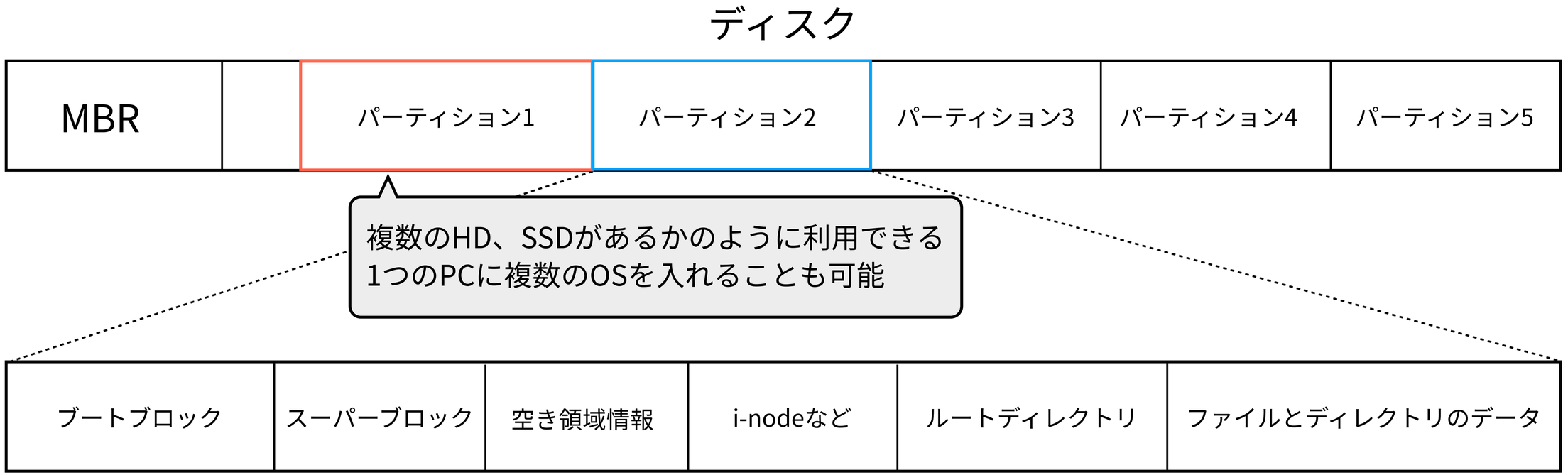


用語

- MBR(master boot record)
  コンピュータ起動のためのプログラムを保持

- パーティション
  データ領域分割のための壁、もしくは区切られた領域

補助記憶装置の中身



用語

- MBR(master boot record)
  コンピュータ起動のためのプログラムを保持

- パーティション
  データ領域分割のための壁、もしくは区切られた領域

パーティション内の要素

ブートブロック	スーパーブロック	空き領域情報	i-nodeなど	ルートディレクトリ	ファイルとディレクトリのデータ

ブートブロック: OSを呼ぶためのファイルを保持

スーパーブロック: ファイルシステムに関する重要なパラメータを保持

i-node: ファイル情報をまとめたデータ (後述)

ルートディレクトリ: ファイルシステムのルートディレクトリの情報を保持

ファイルとディレクトリ: ファイルとディレクトリの実データを保持

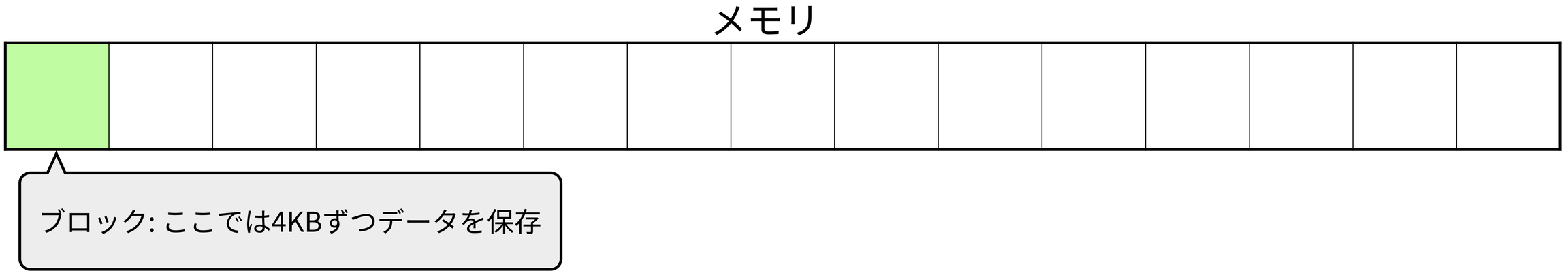
ファイルの実装

データをファイルとして記憶装置に保存する方法

(1) 連続ブロック割り当て

ブロック: 一定量のデータを保存するかたまり

例) 1ブロック4KBの記憶装置に対して53KBのファイルを保存する場合



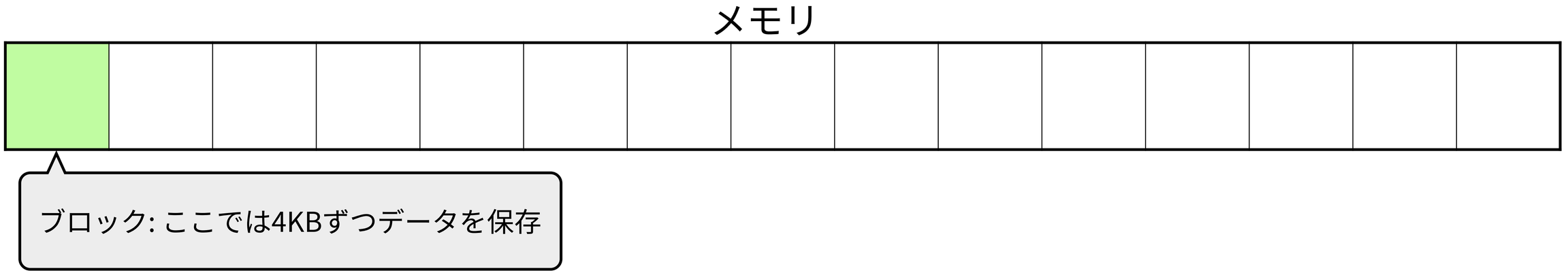
ファイルの実装

データをファイルとして記憶装置に保存する方法

(1) 連続ブロック割り当て

ブロック: 一定量のデータを保存するかたまり

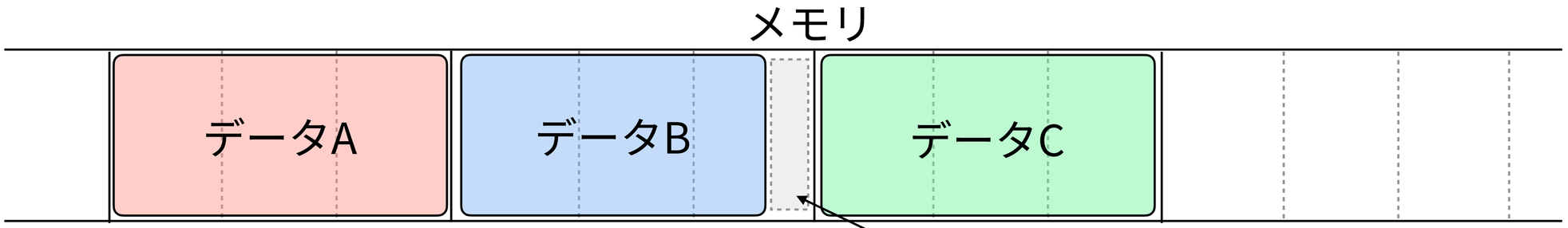
例) 1ブロック4KBの記憶装置に対して53KBのファイルを保存する場合



53 ÷ 4 = 13 あまり 1なので、
53KBのファイルを保存するには、14ブロック用意する必要がある

(1) 連続ブロック割り当て

メリット: 無駄なくデータを保存可能



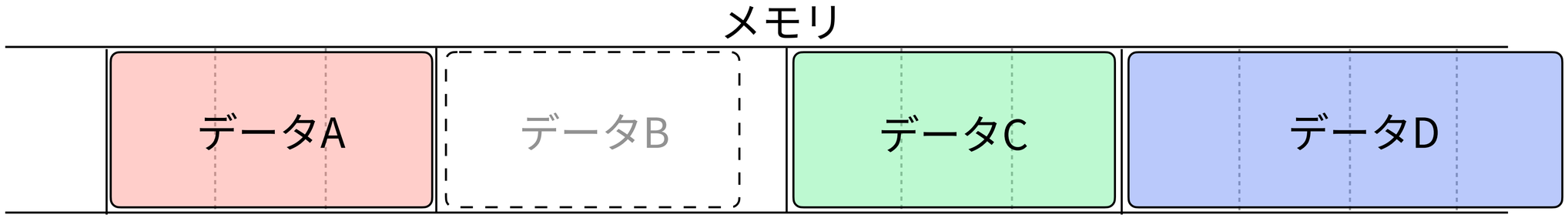
例) 1ブロック5KBの場合

データA: 15KB
データB: 13KB
データC: 15KB

データA + データB + データC + α の容量を用意すれば十分

(1) 連続ブロック割り当て

デメリット:データの書き換えに制約がある



例) 13KBのデータBを削除して17KBのデータDを保存する場合
　　データBが置かれていた領域に配置できるデータは15KB以下
　　→17KBのデータDはそのほかの領域に配置する必要がある

連続ブロック割り当てが使用される補助記憶装置

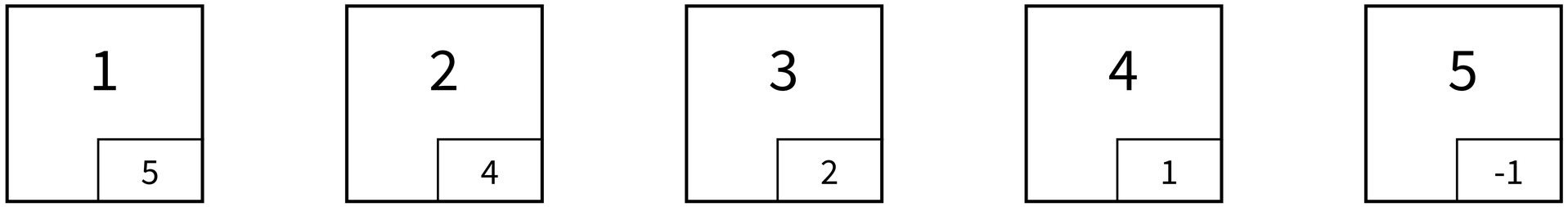
　CD-RやDVDなど (データの書き換えを想定しない)

(2) 連結リスト割り当て

ブロックの関係をリストとして保持

2種類の実現方法

① 各ブロックを連結リストとする場合

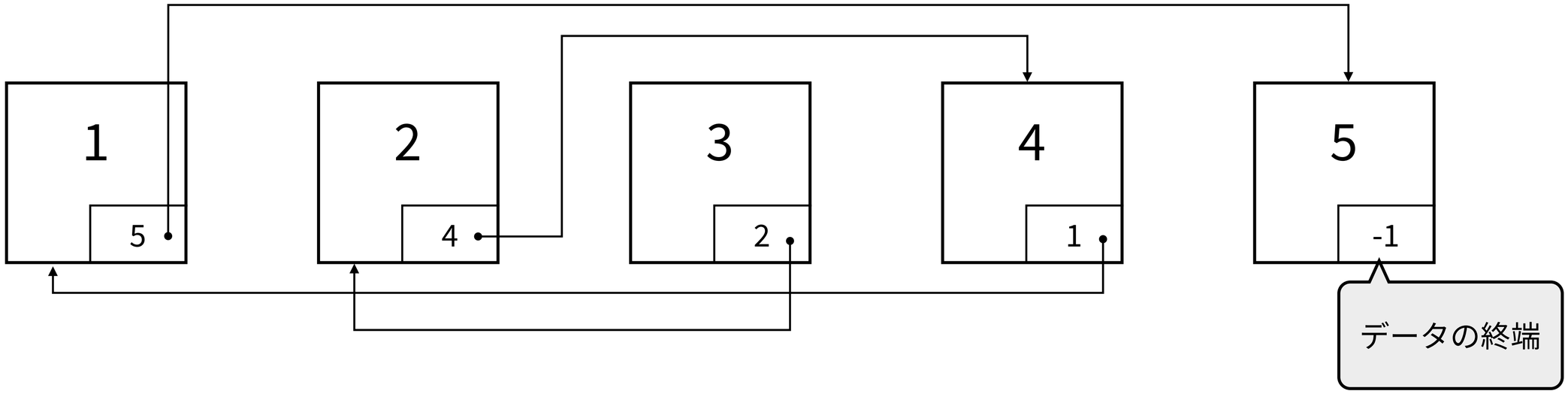


(2) 連結リスト割り当て

ブロックの関係をリストとして保持

2種類の実現方法

① 各ブロックを連結リストとする場合

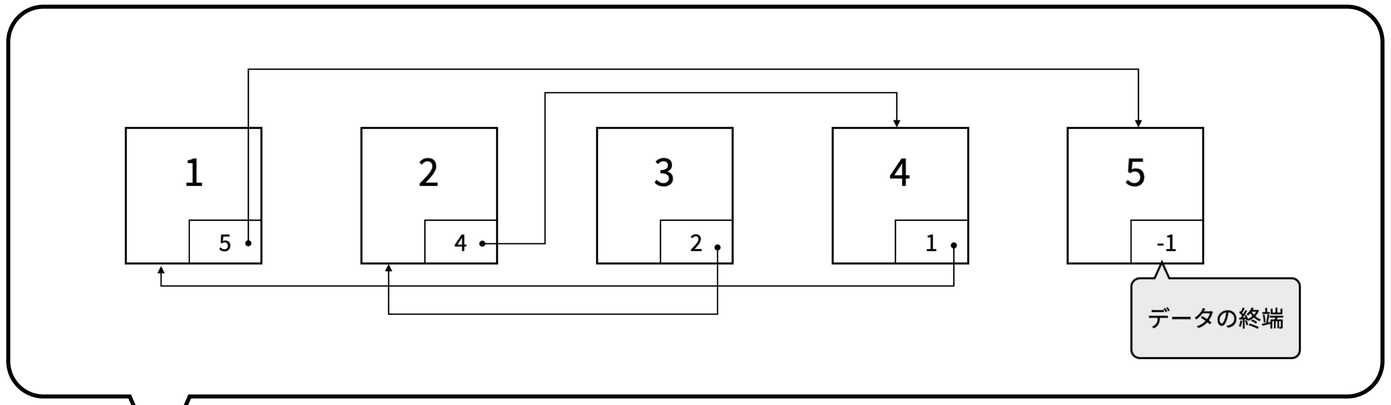


(2) 連結リスト割り当て

ブロックの関係をリストとして保持

2種類の実現方法

① 特定のブロックに連結リストテーブルを用意する場合



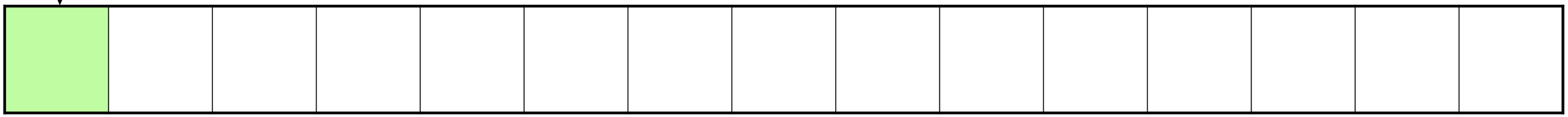
メリット

無駄なデータを保持しない

デメリット

探索に時間がかかる

メモリ



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

ブロック番号

i-nodeによる管理

i-node: ファイルやディレクトリのメタデータを管理するデータ構造
ファイルやディレクトリはデータそのものとデータに関する属性情報から構成

i-nodeが格納する情報

- ファイルのタイプ (通常ファイル)
- ファイルサイズ
- 所有者
- アクセス権
- タイムスタンプ
- ブロックポインタ
  実際のデータが格納されているディスク上のブロックへの参照ポインタ

など

ディレクトリは、ファイル名とi-nodeのペアを持っている

i-nodeの特性およびデータの管理方法

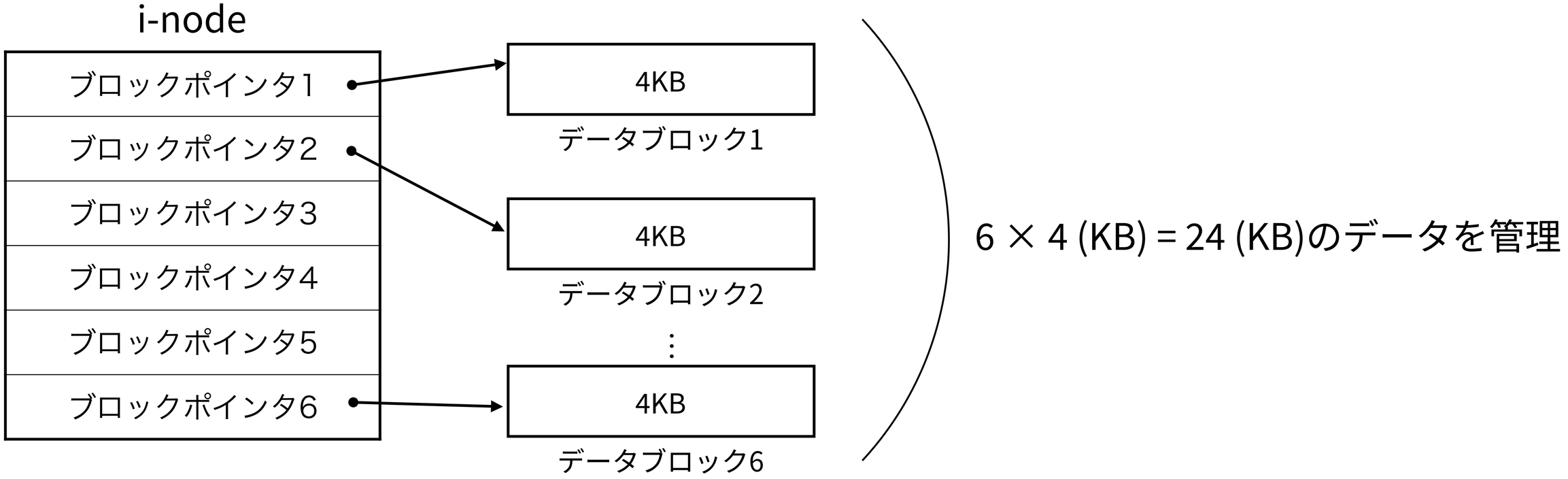
i-nodeの特性

データ分離: ファイル名とデータが分離されている

ファイル名を変更してもi-nodeやデータへの影響はない

データの管理方法

i-nodeが持つブロックポインタに従いデータを管理

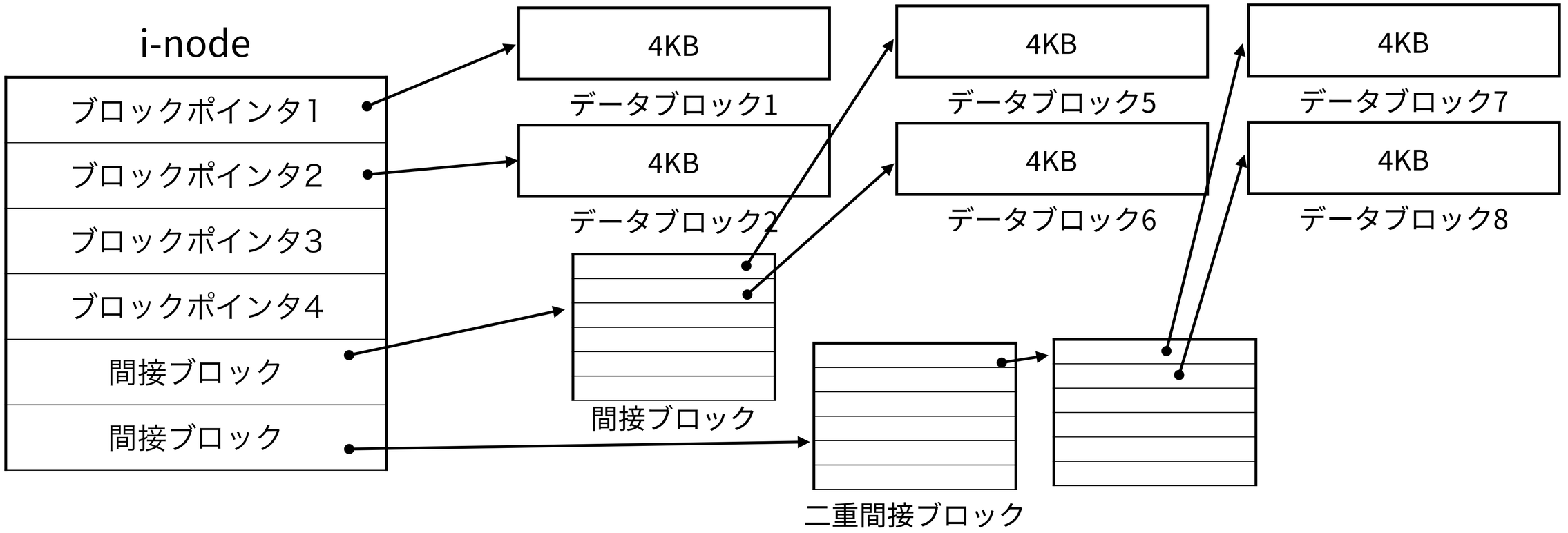


i-nodeの特性およびデータの管理方法

大きなデータを保持する場合には、間接ブロックを使用

間接ブロックを使うことで、より多くのデータブロックを参照可能

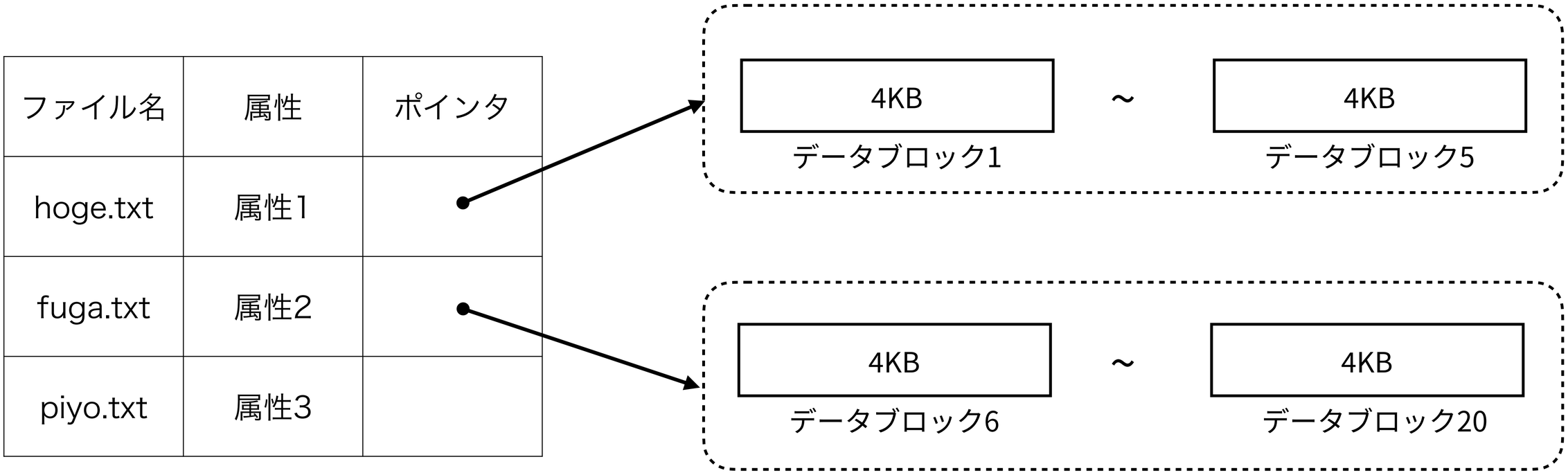
ただし，データにアクセスするための時間は増加する



ディレクトリの実装方法

代表的な実装方法

1）ファイル名と属性、ブロックアドレスへのポインタを保持する場合
2）ファイル名とメタデータへのポインタを保持する場合

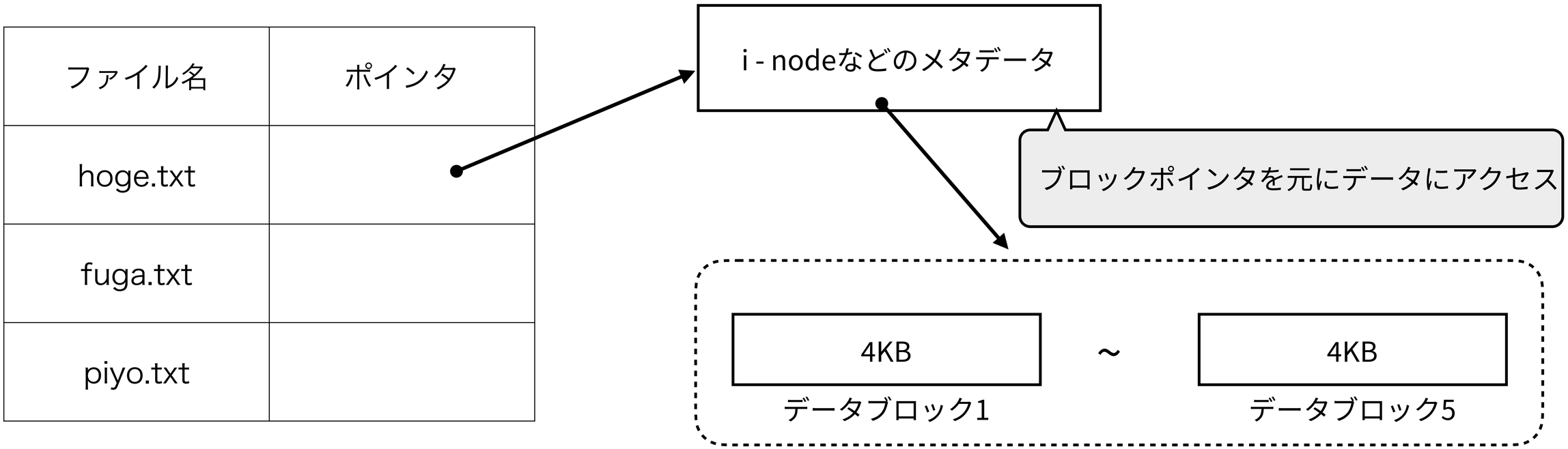


1）ファイル名と属性、ブロックアドレスへのポインタを保持する場合

ディレクトリの実装方法

代表的な実装方法

1）ファイル名と属性、ブロックアドレスへのポインタを保持する場合
2）ファイル名とメタデータへのポインタを保持する場合



2）ファイル名とメタデータへのポインタを保持する場合

リンク

様々なディレクトリからファイルやディレクトリにアクセスするための仕組み

2種類の方法

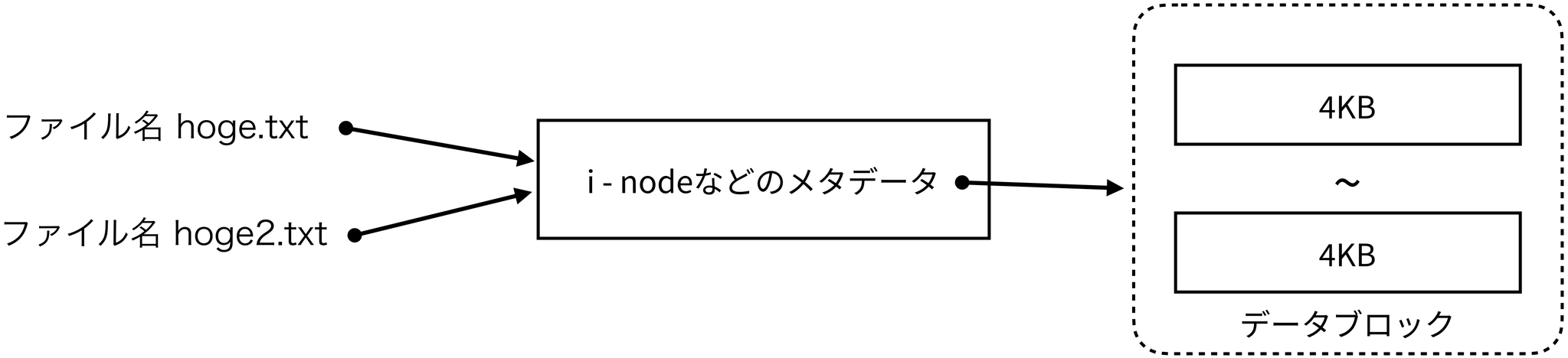
ハードリンク、ソフトリンク（シンボリックリンク）

ハードリンク

異なる名前や場所から同じデータにアクセスするための仕組み

複数のファイル名と特定のi-nodeを結びつける

異なるファイル名から同一のデータにアクセスすることができる



ハードリンク

ハードリンクの特性・メリット

特性

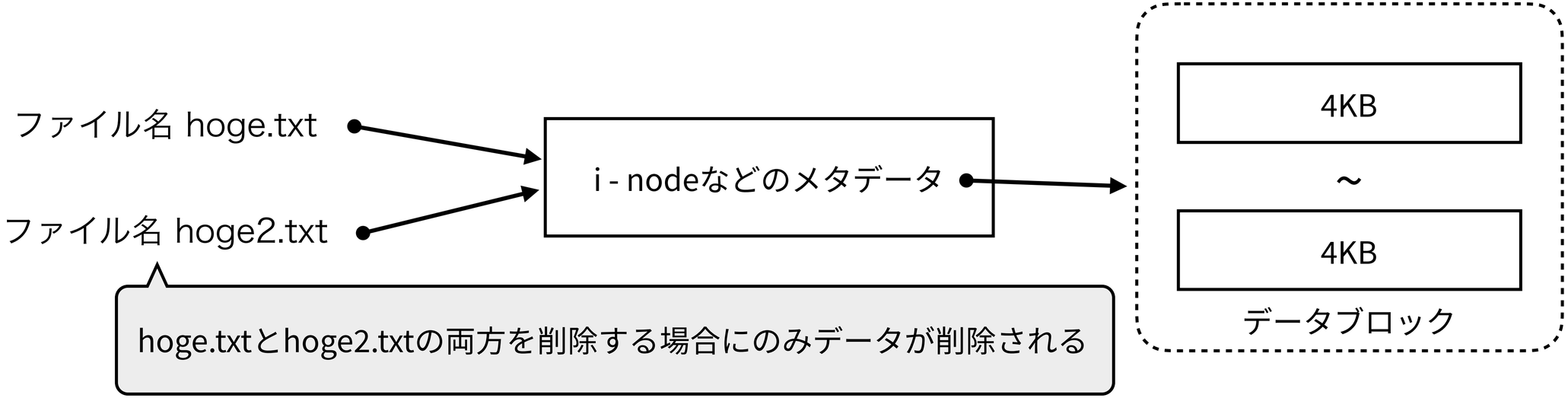
ディレクトリにはハードリンクを作成できない

一方のファイル名を削除してもデータは残り続ける

メリット

データの冗長化を抑えられる

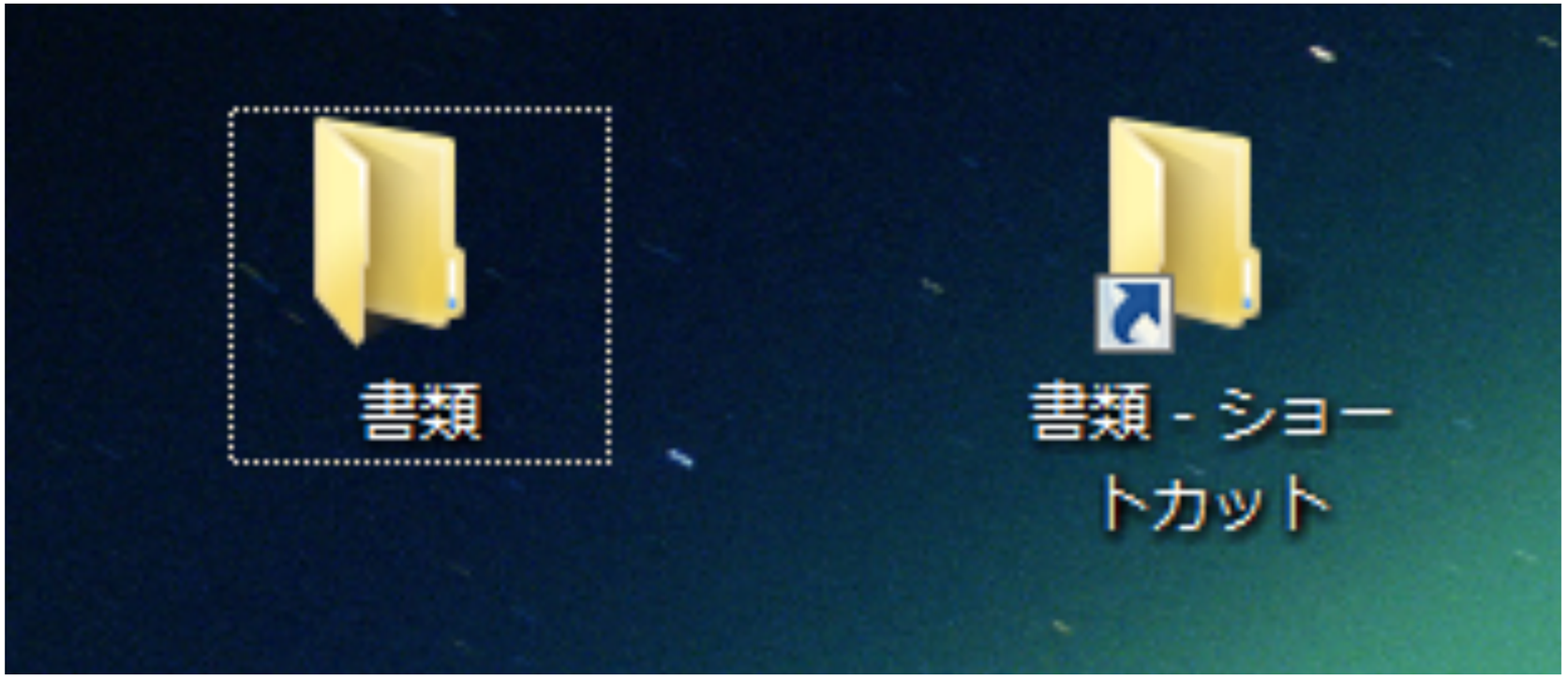
バックアップの作成やソフトウェアのバージョン管理



ソフトリンク(シンボリックリンク)

シンボリックリンク:

参照先のパスを記録するリンクを使用（いわゆるショートカットファイル）



ハードリンク、ソフトリンク(シンボリックリンク)の違い

	ハードリンク	ソフトリンク
仕組み	同一のi-nodeを共有	参照先のパス名を記録
ディレクトリに対するリンク	作成不可	作成可能
元ファイル削除時	他のリンクがあれば データは保持される	リンクが壊れる （アクセスできなくなる）
アクセス速度	速い	遅い
ディスク使用量	データは共有するため増えない	リンクを記録するファイルぶん増加