オペレーティングシステム
第１８章 ZFS

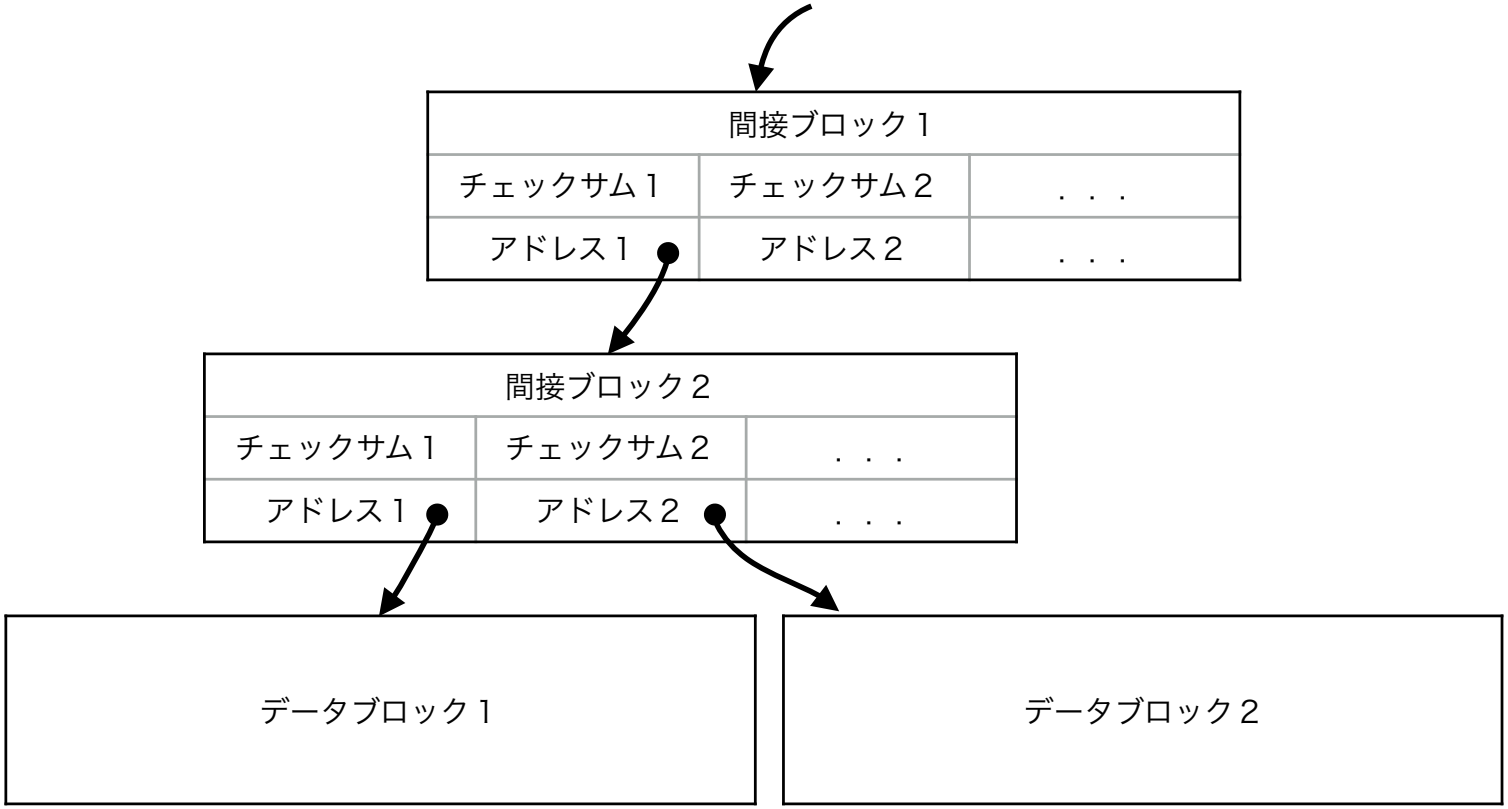
https://github.com/tctsigemura/OSTextBook

ZFS の特徴（1）

2005 年にサン・マイクロが OpenSolaris に実装して公開し，オープンソースで開発が続いているファイルシステム．FreeBSD，Linux 等に移植され Solaris 以外の OS でも使用できるようになっている．大きな主記憶と，高速なマルチプロセッサシステムを前提に設計されている．

- *COW*（*Copy On Write*）でデータやメタデータをハードディスク（以下ではデバイス）に書き込む．デバイスのブロックを上書きすることが無い．
- 一連の書き込み終了時点に Uberblock を書き込むと変更が反映される．
- Uberblock の書き込み前なら変更前の完全な状態，Uberblock の書き込み後なら変更後の完全な状態になり，変更途中の不完全な状態になることはない．
- チェックサムにより高い信頼性が確保されている．ファイルシステムのメタデータだけでなく，全てのデータ（ブロック）のチェックサムが，そのブロックを管理する 1 階層上のデータ構造に記録されている．(次ページの図)

i-node 全ブロックにわたるチェックサム

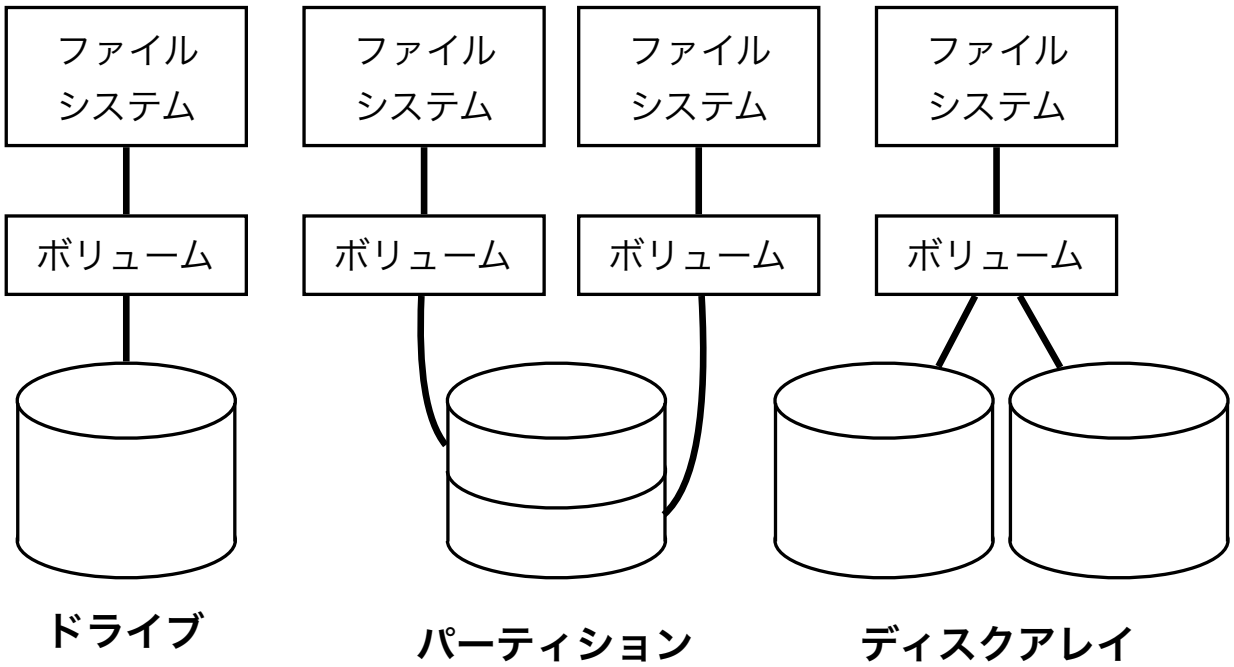


- ブロックポインタがチェックサムを持つ．
- チェックサムの不整合が見つかった場合，データの２重化（ミラー）がされていれば，自動的にミラーからデータを修復する．

ZFS の特徴（2）

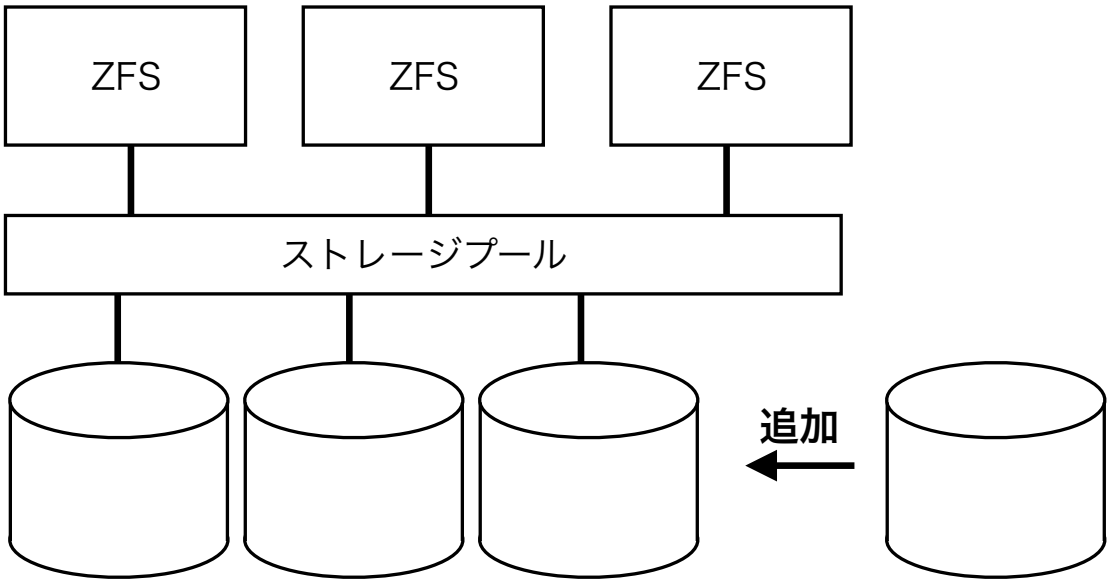
- **スナップショット**や**クローン**の作成が一瞬で完了する．その後は COW の手法を使用し，コピーとオリジナルに違いが出た時点で，違いが出たブロックとそれの親だけのコピーが作られる．デバイスの容量も無駄にならない．（前の図で最上位だけコピーするイメージ）
- ボリュームの代わりにストレージプールと呼ばれるソフトウェアの層をデバイスとファイルシステムの間にはさんでいる．（次々ページ）
- ファイルサイズ等の制約が事実上無くなった．ファイルサイズは最大 2^{64} バイト，ストレージプールサイズは最大 2^{70} バイト（$Zetta = 2^{70}$ ）
- ストレージプールは，ミラーや RAID-Z 等によりデバイスの故障に対する信頼性・可用性を向上する．

従来の方式



- ファイルシステムの初期化以前にボリュームを決定し，
- 後でサイズの変更などはできない．

ストレージプール

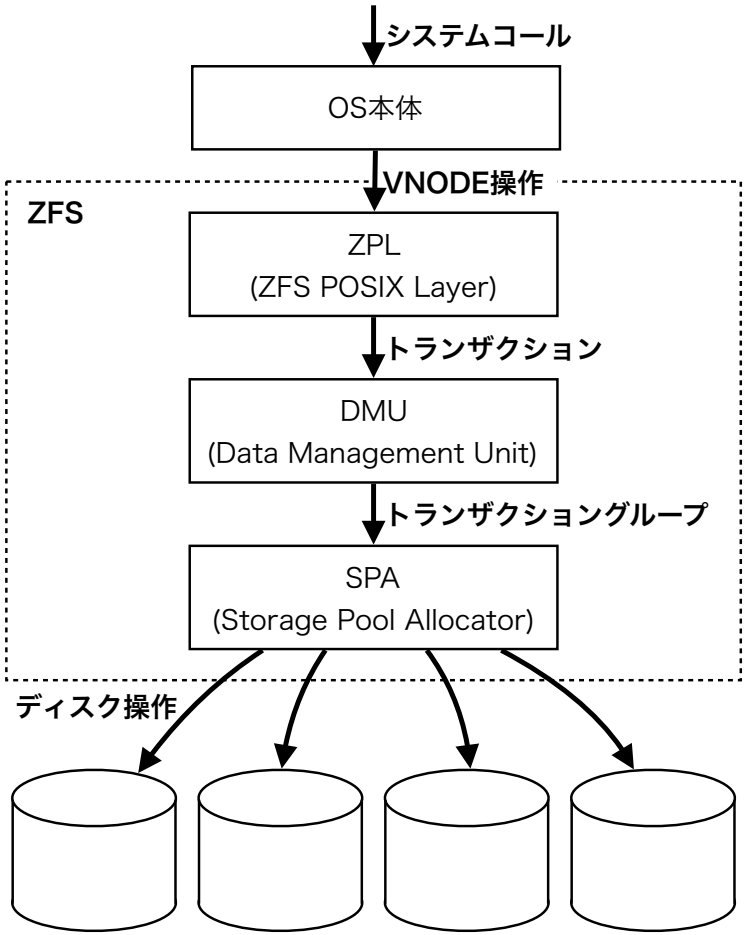


- ストレージプールは沢山のデバイスを収容する.
- ZFS からの要求に応じてデータブロックを割り付ける.
- C 言語プログラムの `malloc()` や `free()` に似ている.
- ストレージプールに後でデバイスを追加することも可能できる.

ZFS の特徴（3）

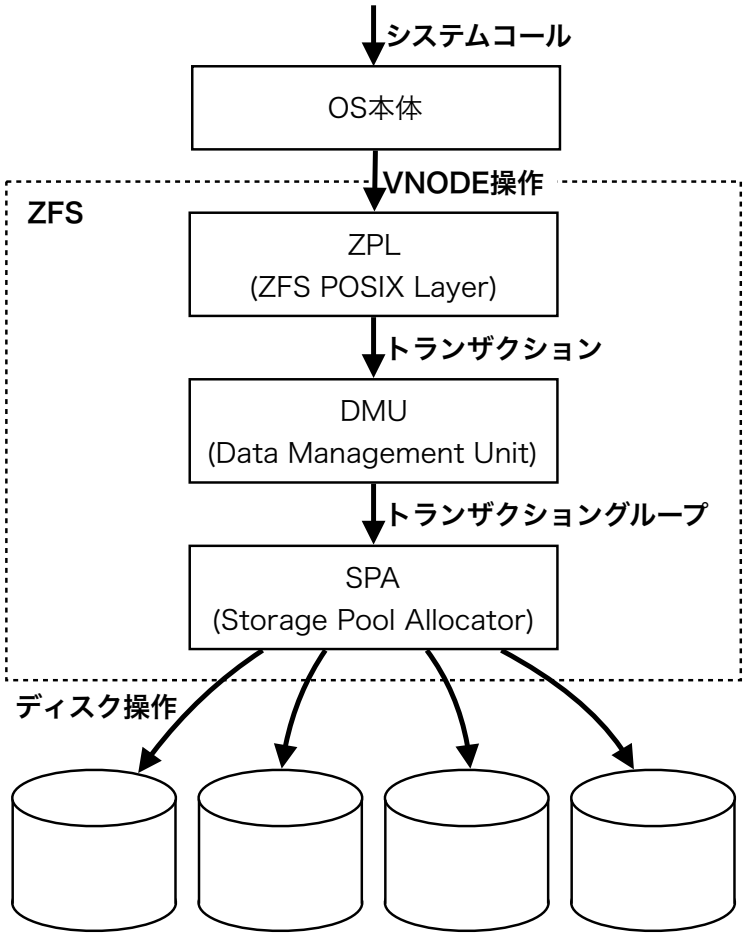
- 仮想記憶のページキャッシュと統合されていない．
- CPU やメモリの利用率が高い．64 ビット CPU でないと ZFS に十分なメモリを提供できない．（FreeNAS では最低 8GiB のメモリ）

ZFS のソフトウェア構成（1）



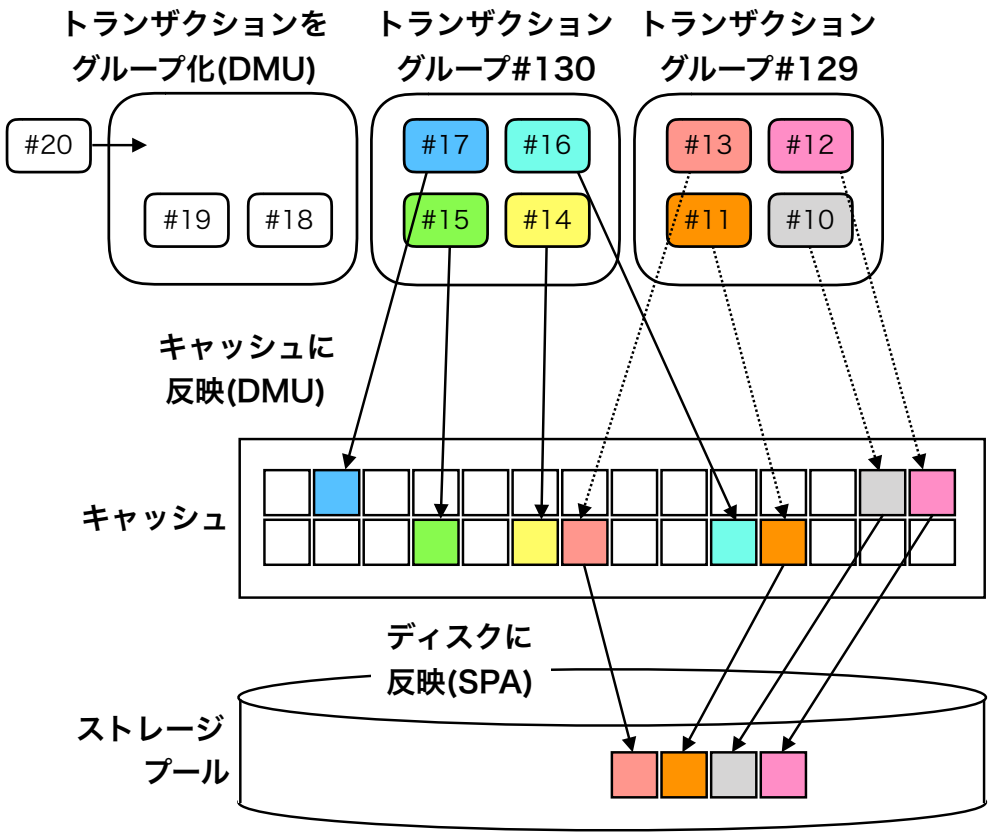
1. システムコールは，OS カーネル本体が VNODE 操作に変換する．
2. ZPL は VNODE 操作を ZFS のトランザクションに変換する．1 つのシステムコールが 1 つのトランザクションに変換される．

ZFS のソフトウェア構成（2）



3. DMU は複数トランザクションを**トランザショングループ**にする.
4. SPA は，DMU がトランザクショングループをキャッシュに書込み終わると，キャッシュの内容をデバイスに反映させる.

ZFS のソフトウェア構成（3）



3. DMU は複数トランザクションを**トランザションググループ**にする．
4. SPA は，DMU がトランザクショングループをキャッシュに書込み終わると，キャッシュの内容をデバイスに反映させる．（バースト）

ストレージプールの構造（概要）

デバイス内部の構造

VL_1（256KiB）
VL_2（256KiB）
ブートコード (3.5MiB)
データ領域
VL_3（256KiB）
VL_4（256KiB）

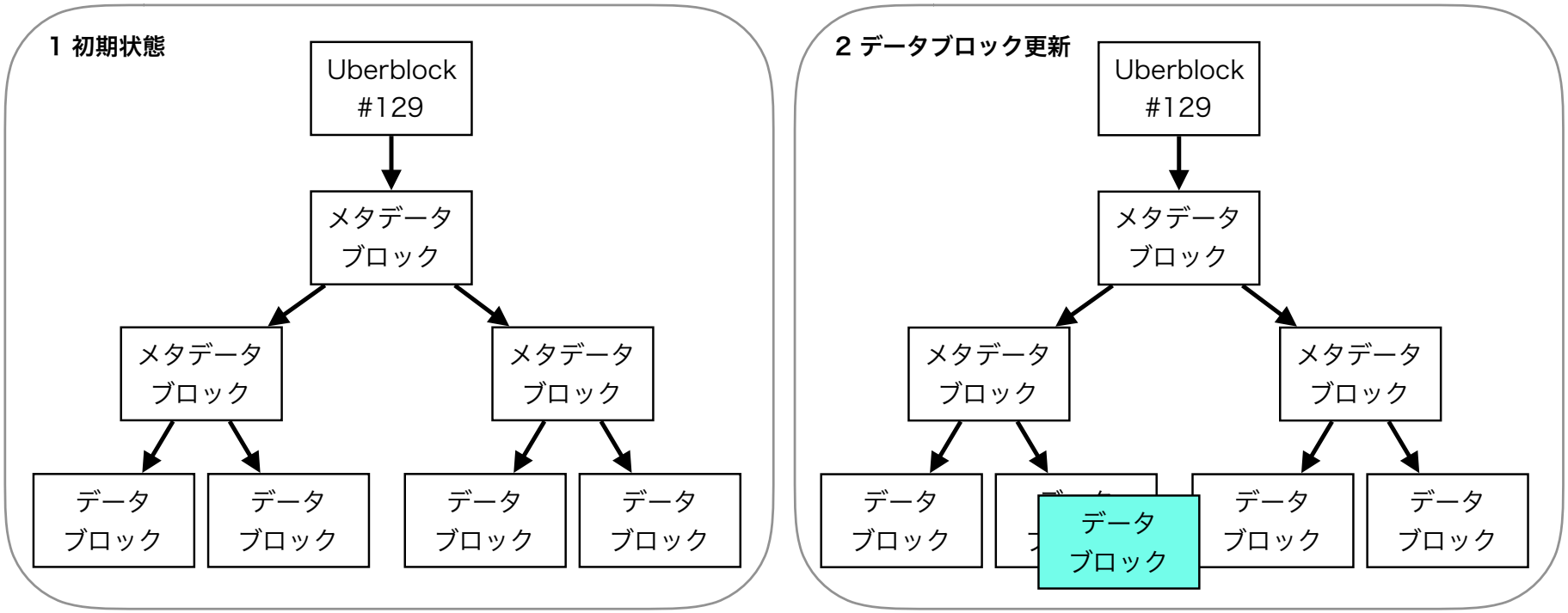
VL_n：ボリュームラベル

ボリュームラベル

デバイス情報など 名前/値ペア （128KiB）
Uberblock[0]（1KiB）
Uberblock[1]（1KiB）
Uberblock[2]（1KiB）
...
Uberblock[127]（1KiB）

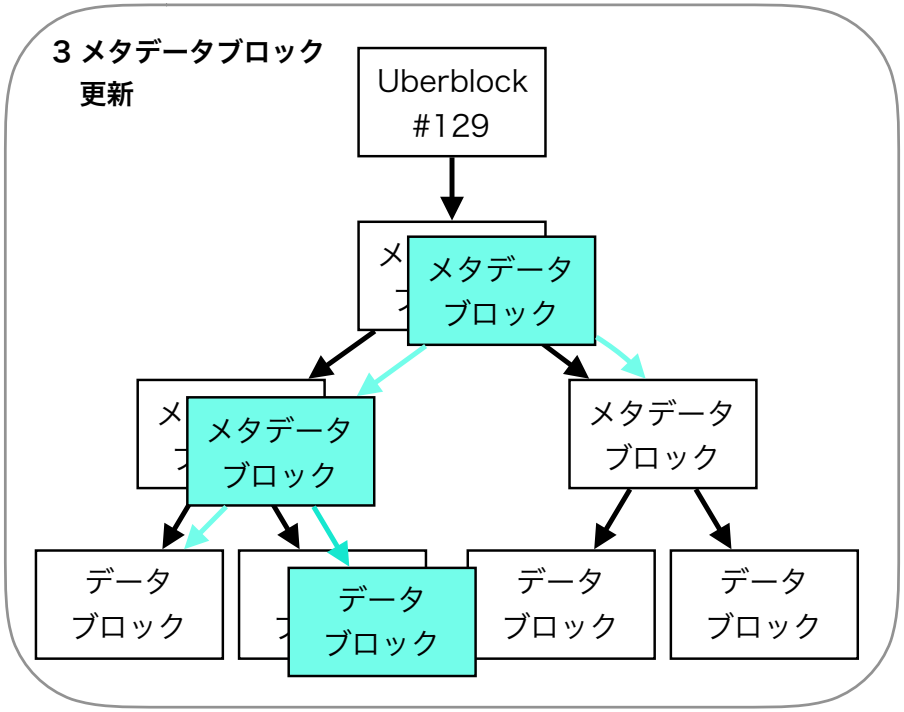
- デバイス（ディスク）の 4 箇所に同じボリュームラベルを書く.
- ボリュームラベルには 128 個の Uberblock を格納できる.
- Uberblock はトランザクショングループ番号を含んでいる.
- Uberblock はトランザクショングループ番号を 128 で割った余りの位置に書く.

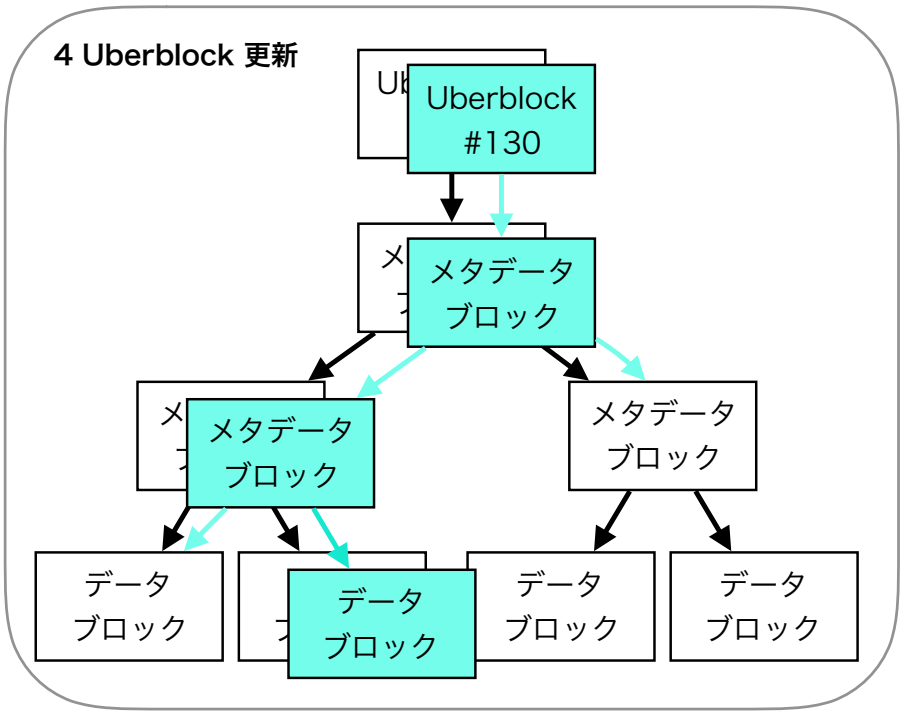
ストレージプールの更新（1）



1. Uberblock 起点の木構造でブロックは記録されている．
2. 変更するには，新しいブロックを確保し内容を書き込む（COW）．

ストレージプールの更新（2）





3. メタデータブロックも COW で更新する.
4. Uberblock を新しい領域に書き込む.
   (トランザクショングループ番号が最新の Uberblock が有効)
   (古い世代のブロックは解放され，再利用される.)

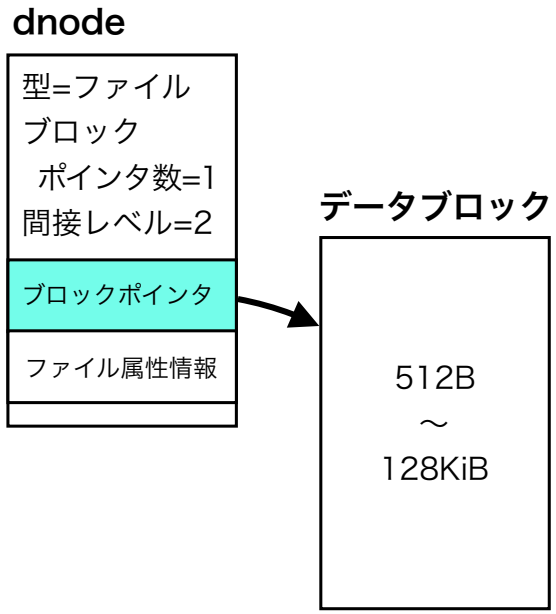
ブロックポインタ

図中でブロックを指していた**矢印**を表現するデータ構造を**ブロックポインタ**と呼ぶ．ブロックポインタはデータ多重化のために最大３組のアドレスを記録できる．ブロックポインタの内容は以下の通り．

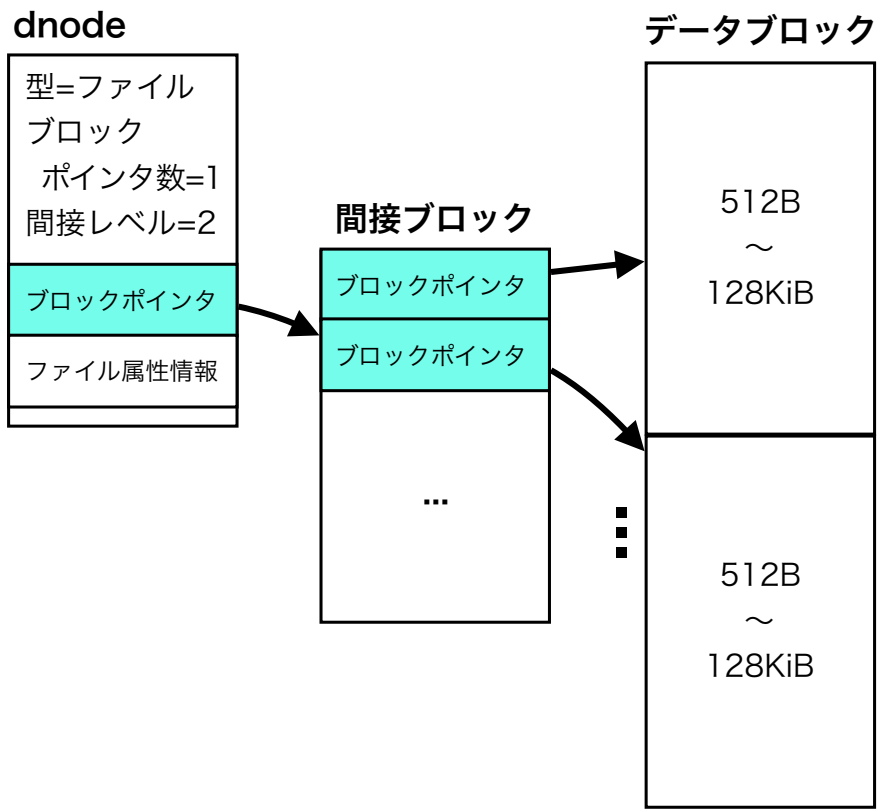
- **サイズ**：ブロックの大きさに関する情報
- **チェックサム**（64 ビット）：ブロックのチェックサム（最大３個）
- **ブロックのアドレス**：ブロックのストレージプール内での格納位置に関する情報（最大３個）（デバイス，デバイス内アドレス）
- **タイムスタンプ**：ブロックを作成したトランザクショングループの番号（ブロックが削除される時にスナップショットと比較）
- **その他**：チェックサム計算に使用するアルゴリズムの種類，データ圧縮に使用するアルゴリズムの種類，圧縮後のサイズなど...

Dnode（1）

ストレージプール内のあらゆるオブジェクトを表現する 512 バイトのデータ構造である．USF の i-node に似ているが，ファイルやディレクトリだけでなく，ファイルシステムなども表現する．

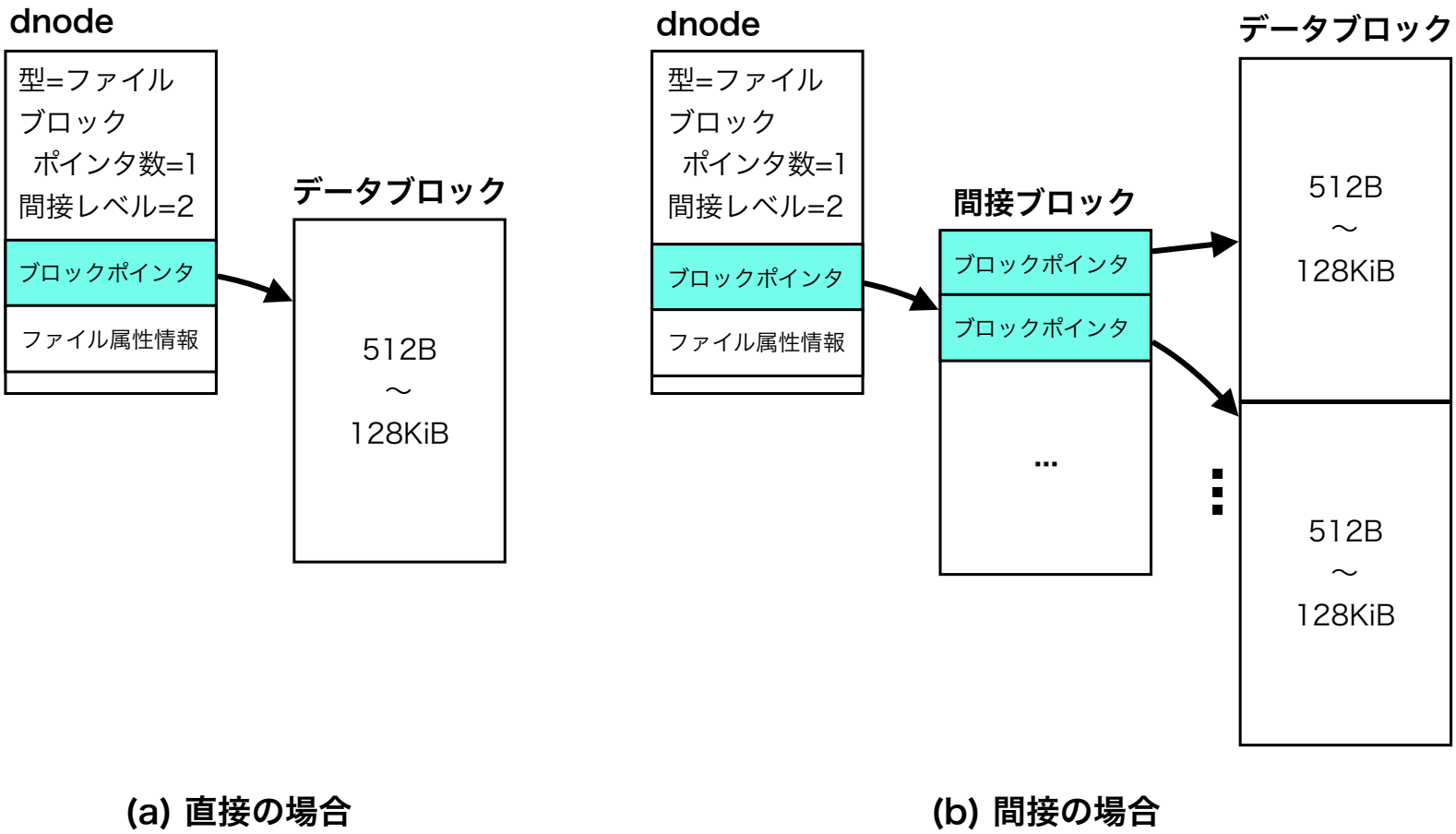


(a) 直接の場合



(b) 間接の場合

Dnode（2）

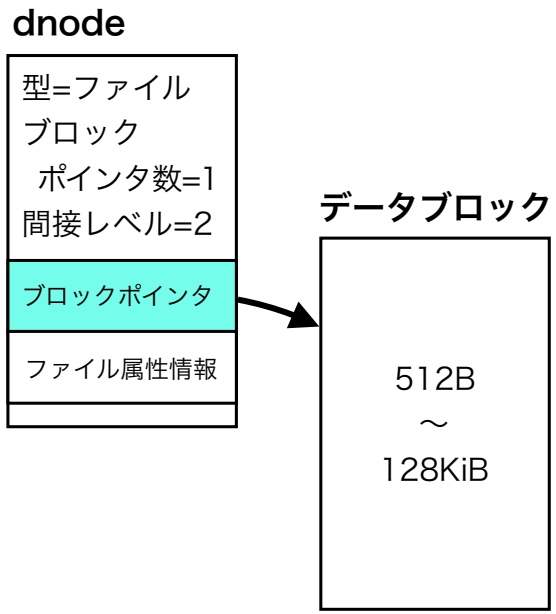


(a) 直接の場合

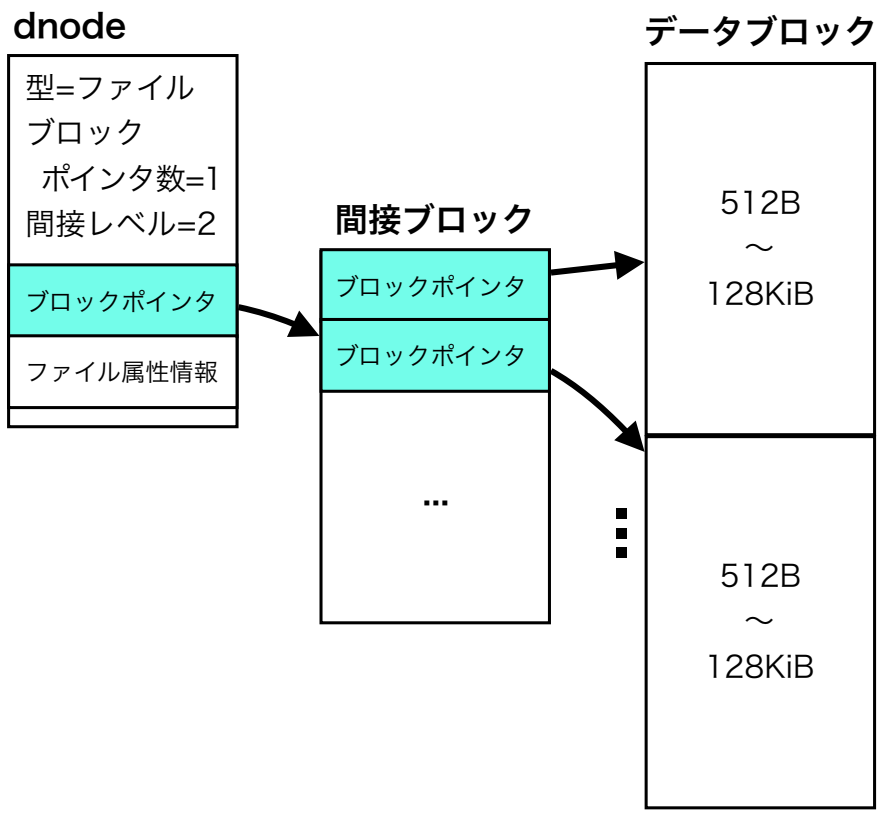
(b) 間接の場合

- dnode は三つ以内のブロックポインタを格納することができる.
- dnode は表現するオブジェクトに応じたデータを格納する領域を持っている.（この領域はブロックポインタと共用になっている）

Dnode（3）



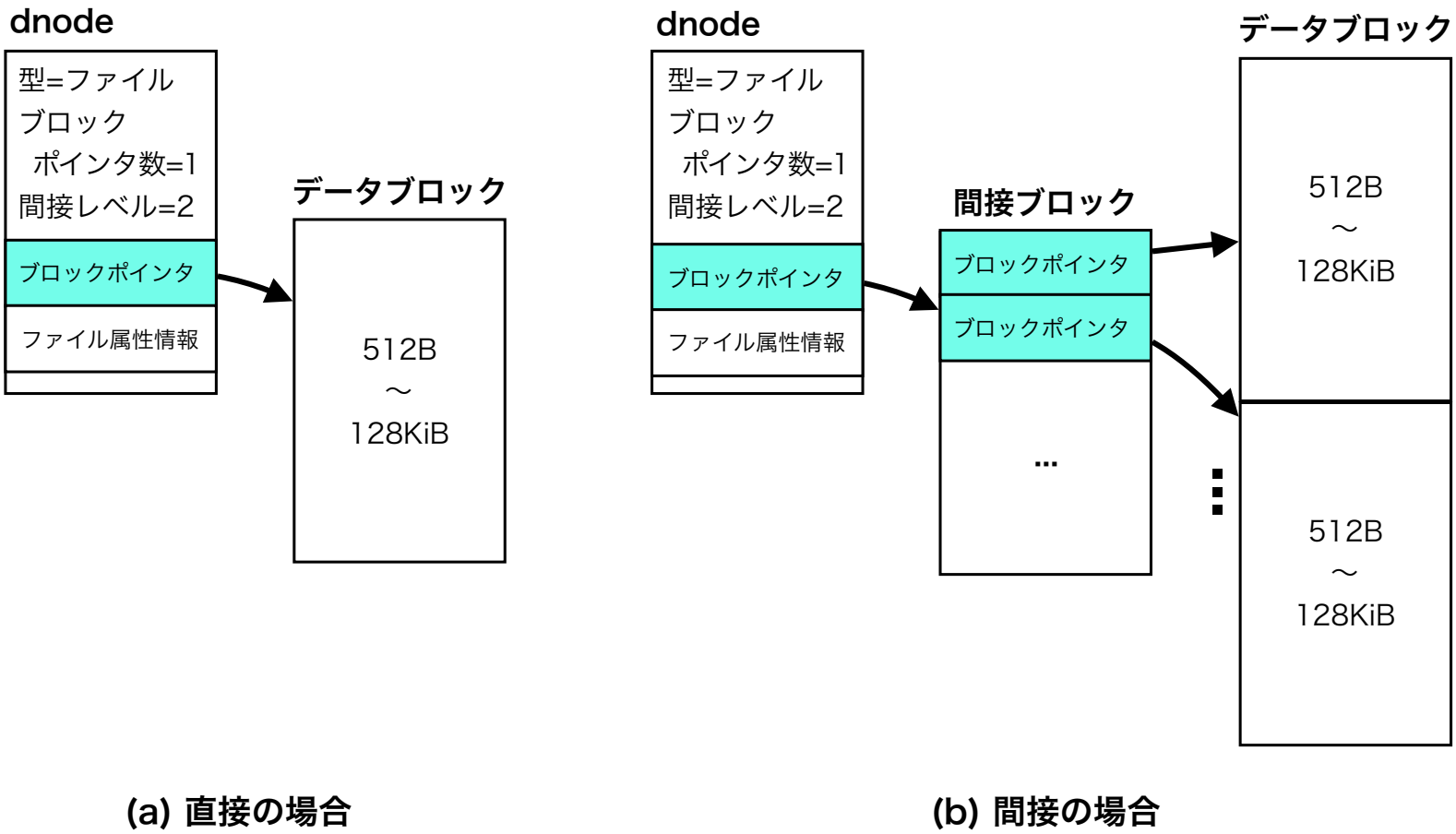
(a) 直接の場合



(b) 間接の場合

- データの大きさが 128KiB 以内の場合は直接参照（図の (a)）
- 大きさが 128KiB を超える場合は間接ブロック（図の (b)）

Dnode（4）



(a) 直接の場合　　(b) 間接の場合

- 128KiB の間接ブロックはブロックポインタを最大 1Ki 個格納できる．
- $128KiB \times 1Ki = 128MiB$ より大きなデーを表現する時は，多重の間接ブロック（最大 6 レベル）を用いる．