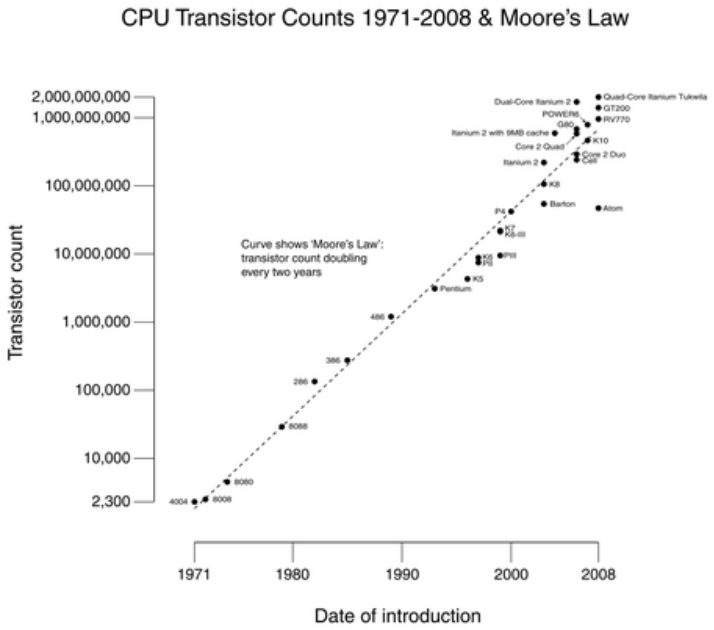
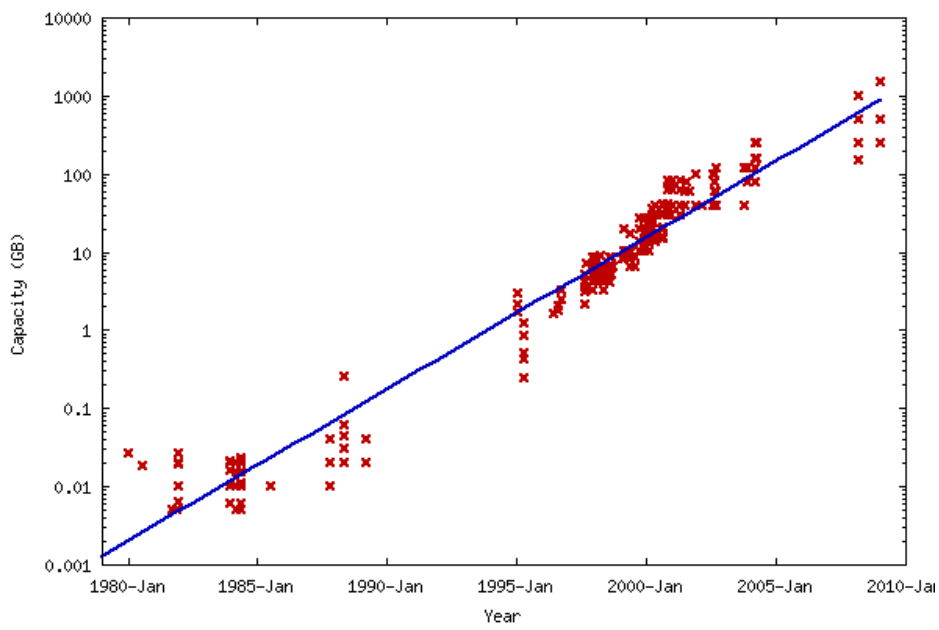
System plików BTRFS

Maciej Łaszcz

05.09.2009

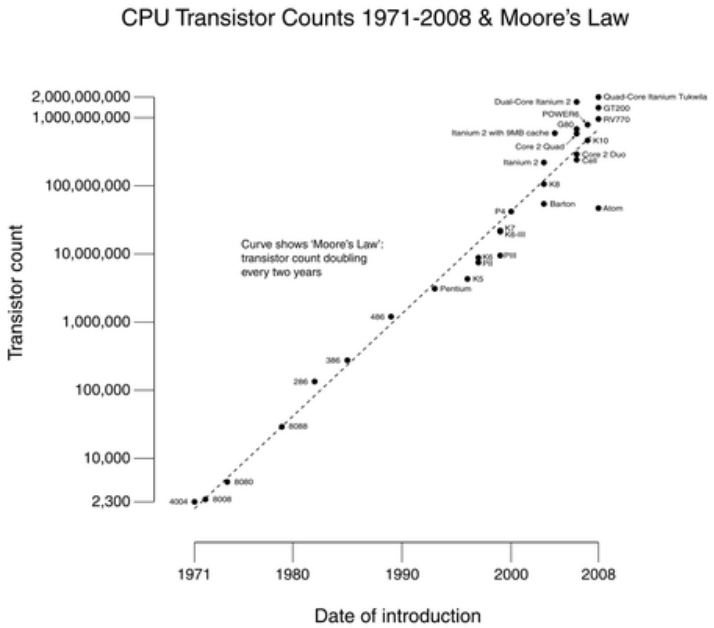
Prawo Moore'a a HDD

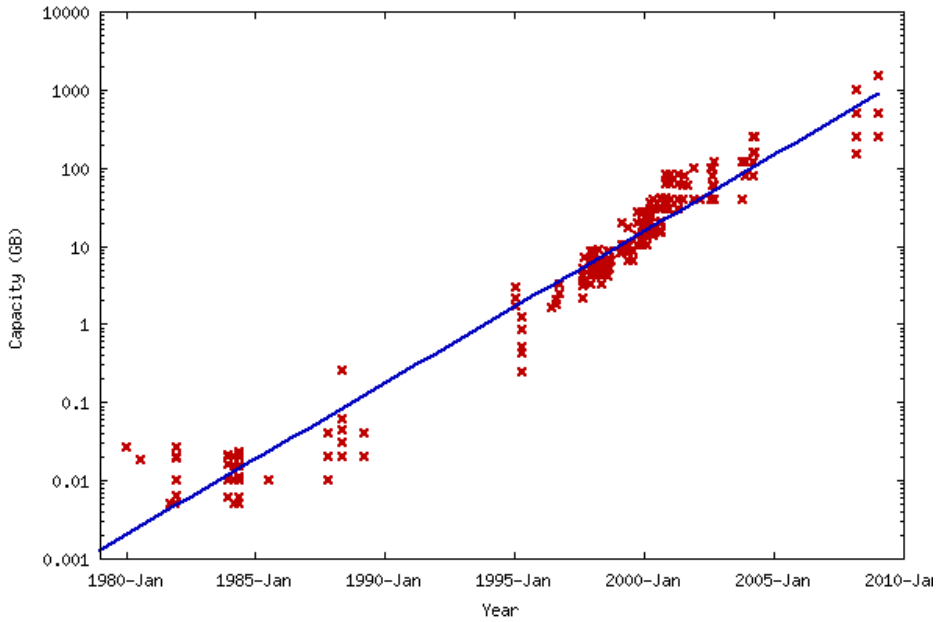




Ale czas dostępu do danych na mechanicznym dysku twardym cały czas trzyma się wartości od 6ms do 15ms i nic nie wskazuje na gwałtowne zmiany

Prawo Moore'a a HDD

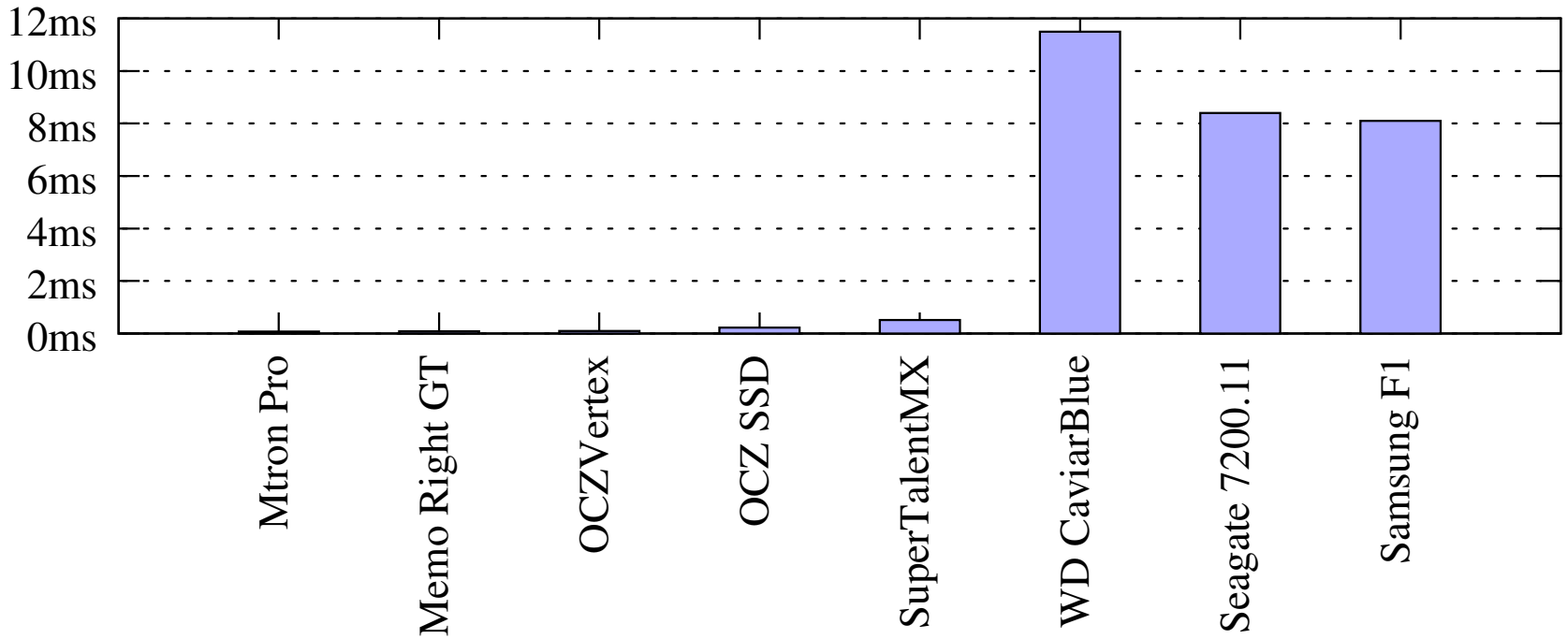




Ale czas dostępu do danych na mechanicznym dysku twardym cały czas trzyma się wartości od 6ms do 15ms i nic nie wskazuje na gwałtowne zmiany

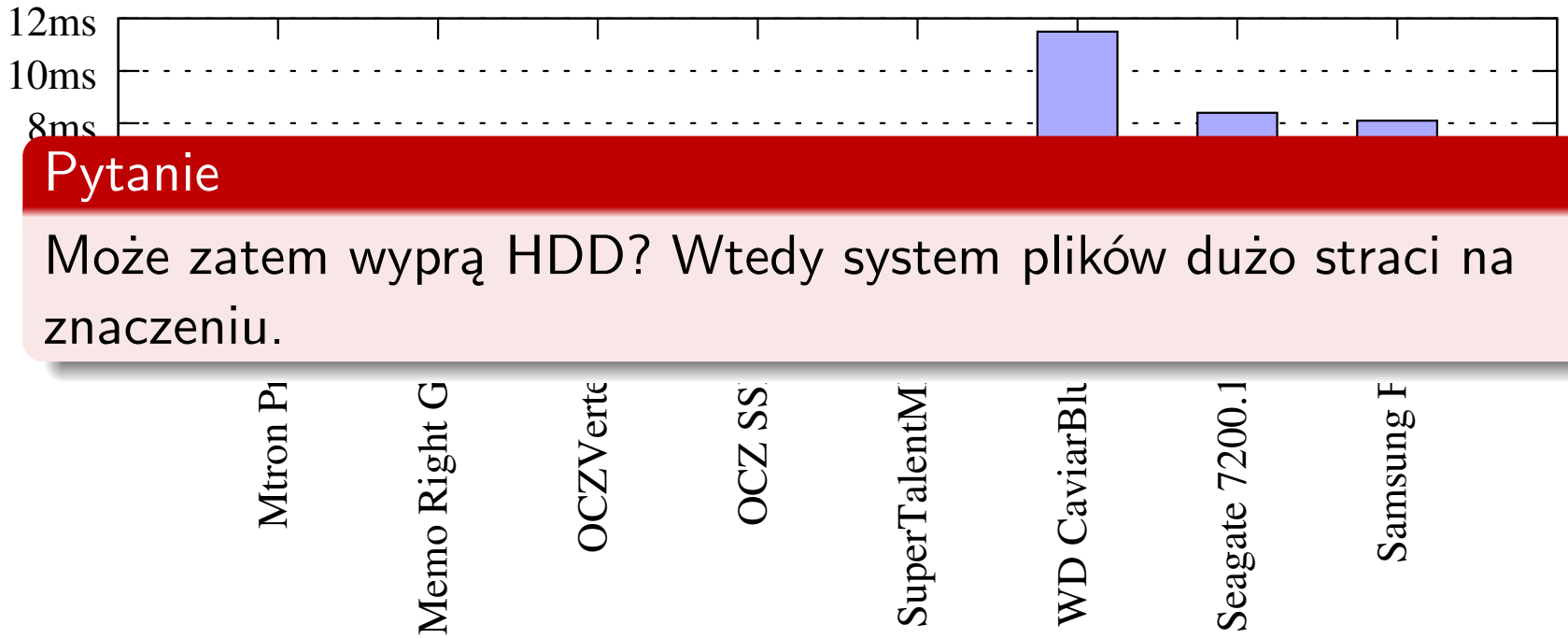
Alternatywa dla HDD

Dyski SSD mają zerowy czas dostępu do danych.



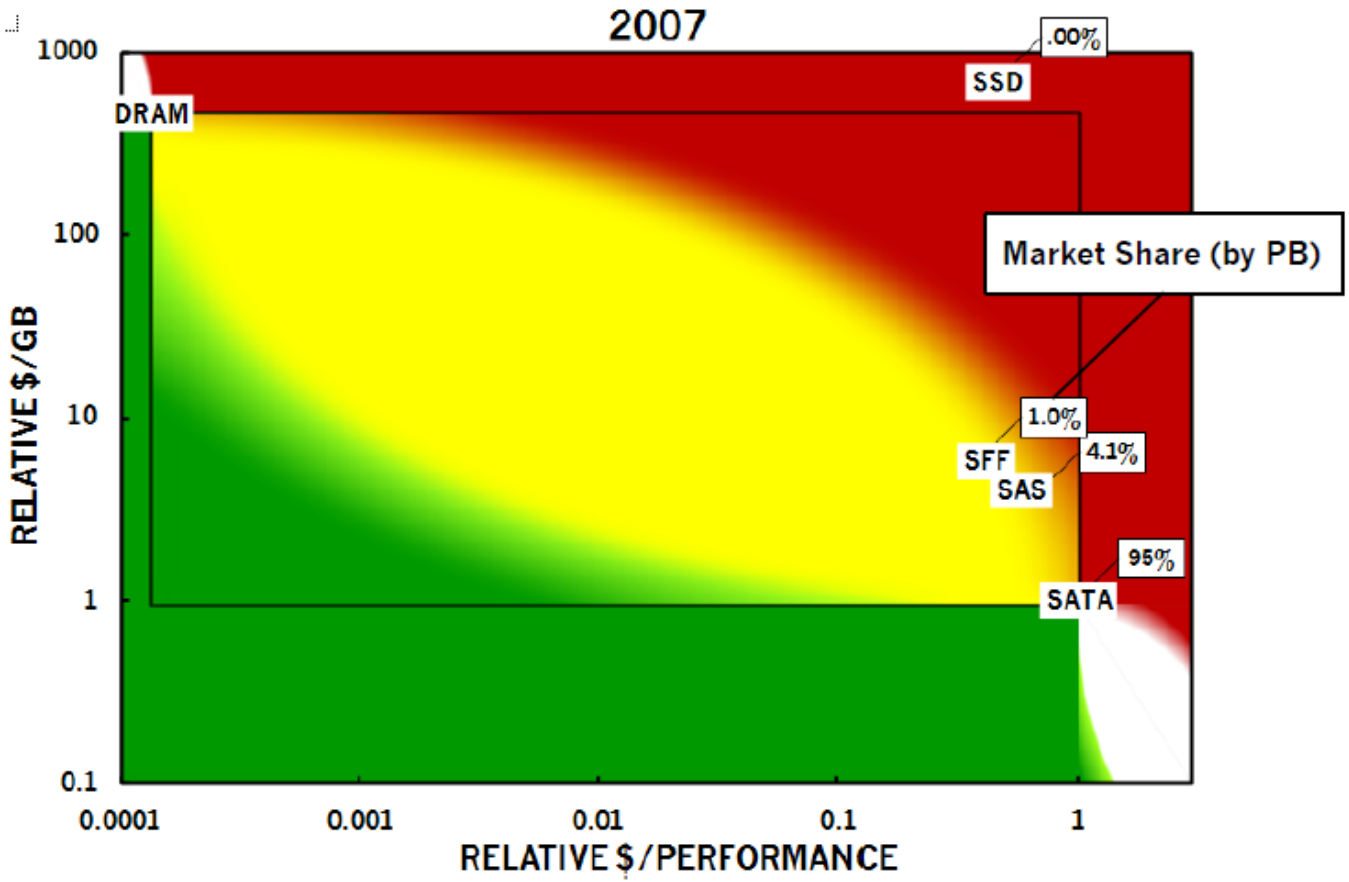
Alternatywa dla HDD

Dyski SSD mają zerowy czas dostępu do danych.

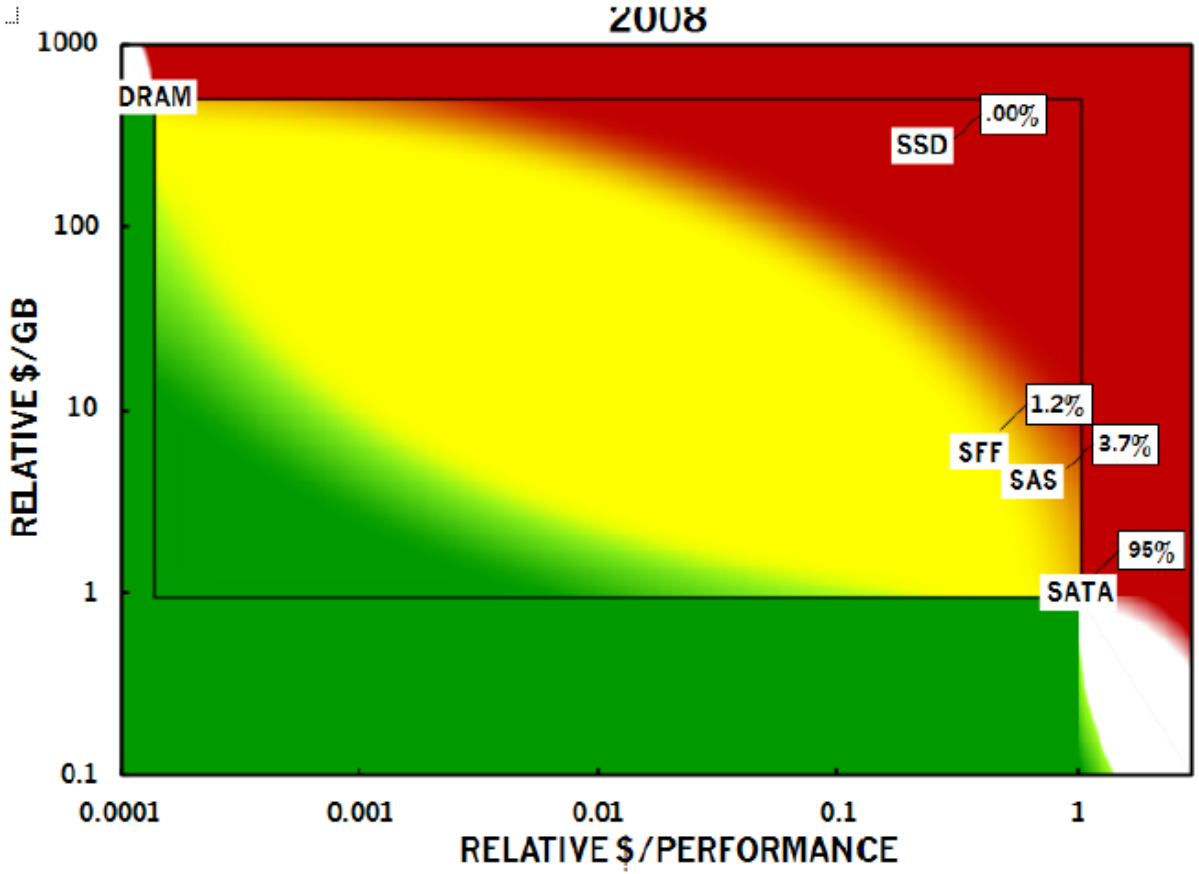


Pytanie

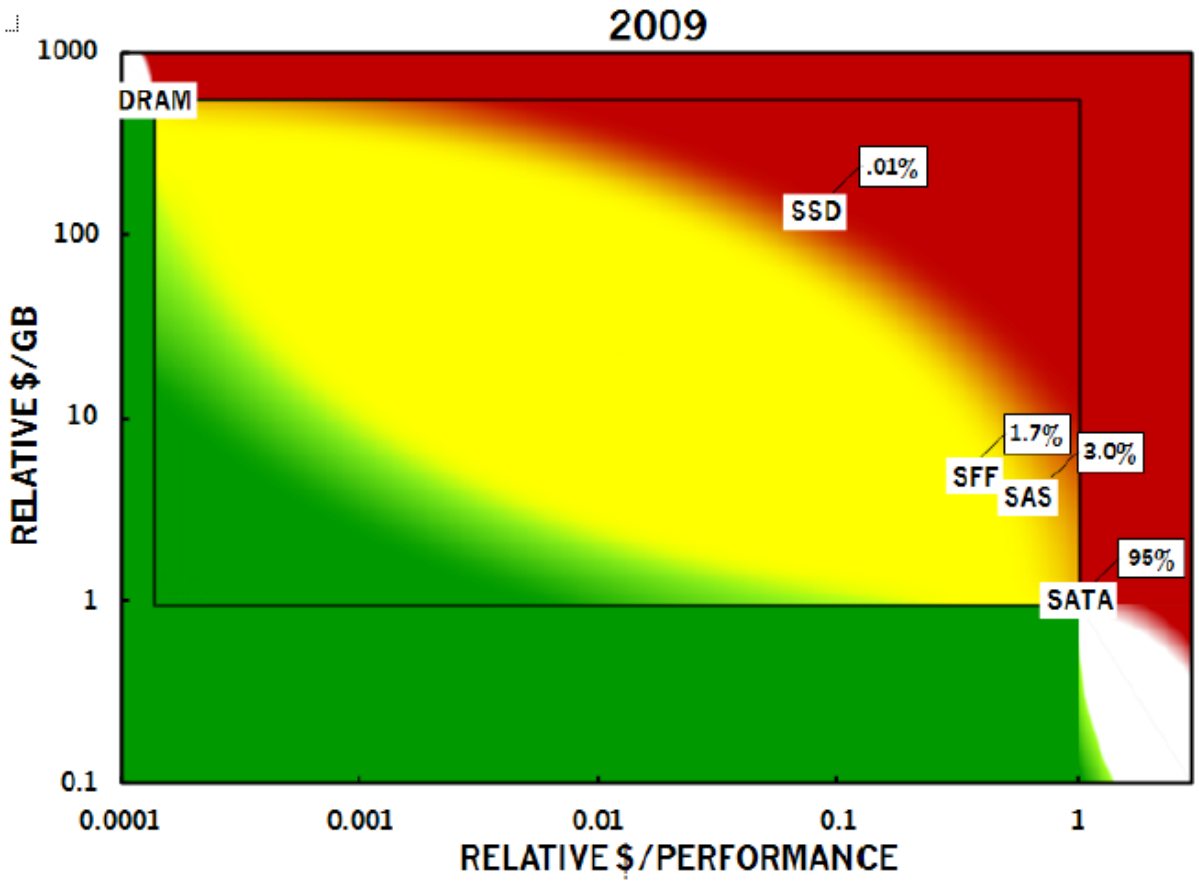
Może zatem wyprą HDD? Wtedy system plików dużo straci na znaczeniu.



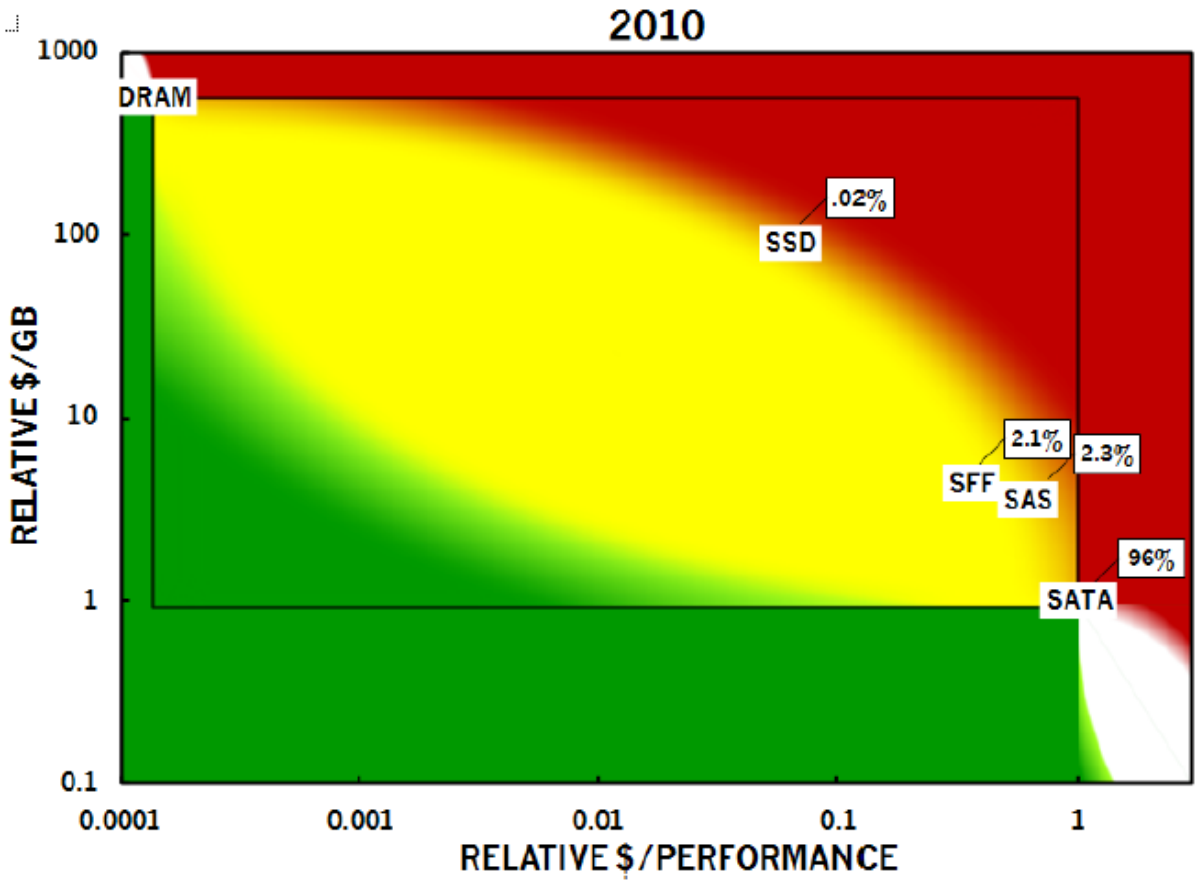
Na razie nie zanosi się na rewolucję na rynku dysków twardych.



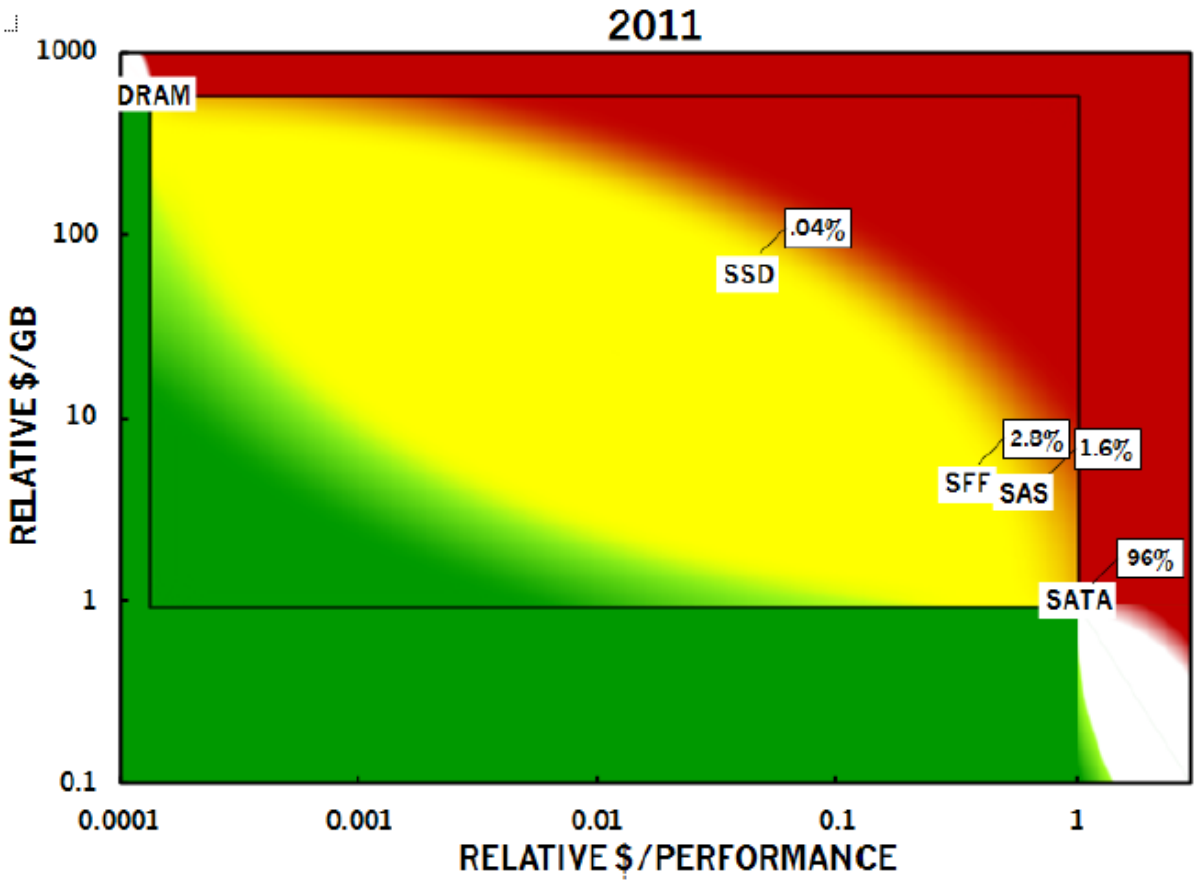
Na razie nie zanosi się na rewolucję na rynku dysków twardych.



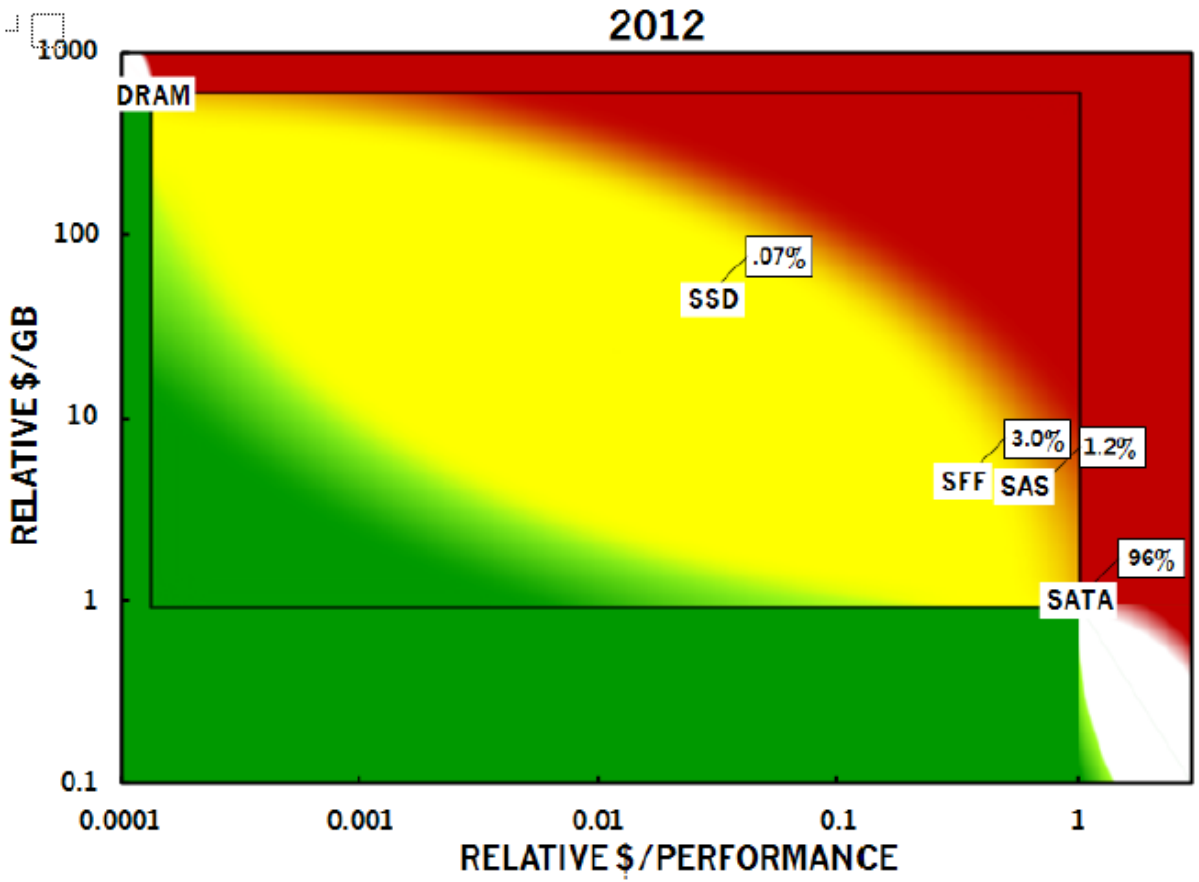
Na razie nie zanosi się na rewolucję na rynku dysków twardych.



Na razie nie zanosi się na rewolucję na rynku dysków twardych.



Na razie nie zanosi się na rewolucję na rynku dysków twardych.



Na razie nie zanosi się na rewolucję na rynku dysków twardych.

Czego wymagamy od systemów plików?

- Szybkości (także w miarę starzenia się systemu)
- Niezawodności
- Wydajnego użycia przestrzeni

W praktyce czyli dzisiejsze problemy

- Stabilne i szybkie odzyskiwanie danych
- Kopie zapasowe masowych danych
- Migrowanie
- Dużo plików w jednym folderze
- Ogromne pliki
- Macierze wielodyskowe

Jak ewoluowały systemy plików

- 1993 – EXT2
- 1994 – XFS
- 1996 – FAT32
- 1999 – EXT3
- 2001 – ReiserFS
- 2004 – ZFS
- 2006 – EXT4 (?)
- 2006 – NTFS

BTRFS

rok 2007 — REISERFS wstrzymany, EXT4 nie zapowiada się rewolucyjnie, a SUN ma ZFS...

08.2007 — FAST&LSF: Ohad Rohde przedstawia B+Drzewa z COW

09.2007 — UKUUG: Chris Mason przedstawia projekt BTRFS

06.2008 — BTRFS v0.15

01.2009 — v0.18 jest w 2.6.29-rc1

06.2009 — v0.19 jest w 2.6.31. Automatyczna zmiana formatu.

BTRFS

rok 2007 — REISERFS wstrzymany, EXT4 nie zapowiada się rewolucyjnie, a SUN ma ZFS...

08.2007 — FAST&LSF: Ohad Rohde przedstawia B+Drzewa z COW

09.2007 — UKUUG: Chris Mason przedstawia projekt BTRFS

06.2008 — BTRFS v0.15

01.2009 — v0.18 jest w 2.6.29-rc1

06.2009 — v0.19 jest w 2.6.31. Automatyczna zmiana formatu.

BTRFS

rok 2007 — REISERFS wstrzymany, EXT4 nie zapowiada się rewolucyjnie, a SUN ma ZFS...

08.2007 — FAST&LSF: Ohad Rohde przedstawia B+Drzewa z COW

09.2007 — UKUUG: Chris Mason przedstawia projekt BTRFS

06.2008 — BTRFS v0.15

01.2009 — v0.18 jest w 2.6.29-rc1

06.2009 — v0.19 jest w 2.6.31. Automatyczna zmiana formatu.

BTRFS

rok 2007 — REISERFS wstrzymany, EXT4 nie zapowiada się rewolucyjnie, a SUN ma ZFS...

08.2007 — FAST&LSF: Ohad Rohde przedstawia B+Drzewa z COW

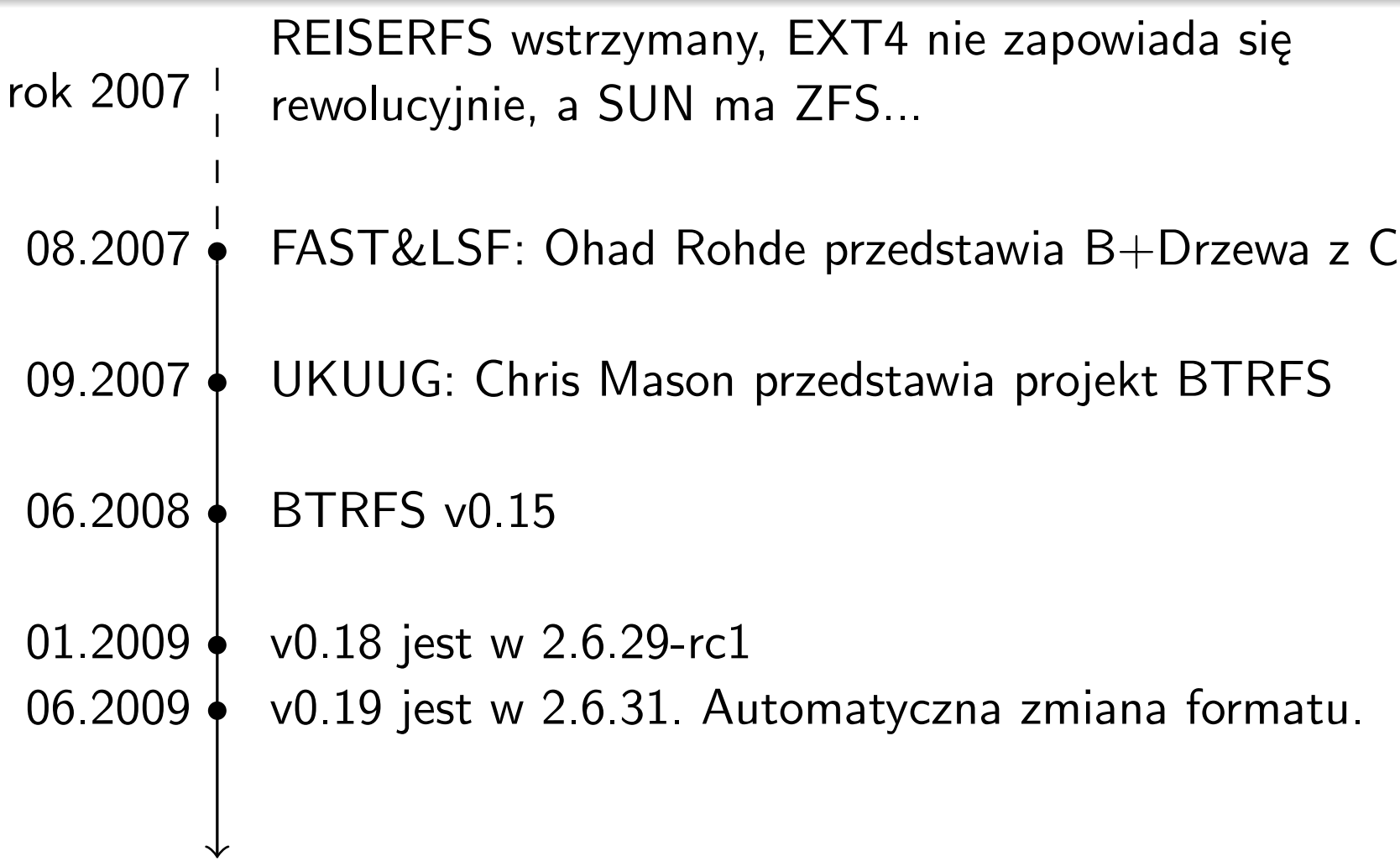
09.2007 — UKUUG: Chris Mason przedstawia projekt BTRFS

06.2008 — BTRFS v0.15

01.2009 — v0.18 jest w 2.6.29-rc1

06.2009 — v0.19 jest w 2.6.31. Automatyczna zmiana formatu.

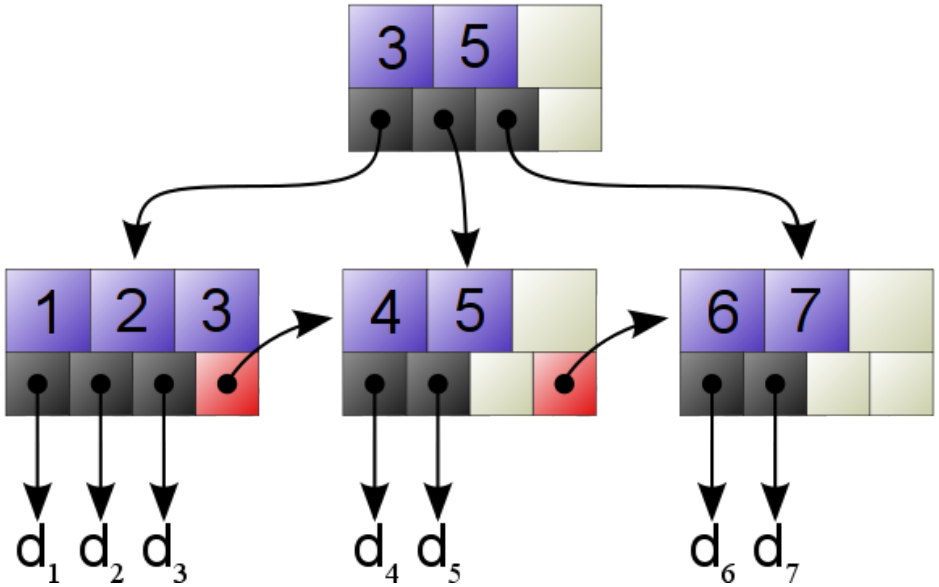
BTRFS


rok 2007
REISERFS wstrzymany, EXT4 nie zapowiada się rewolucyjnie, a SUN ma ZFS...
08.2007
FAST&LSF: Ohad Rohde przedstawia B+Drzewa z COW
09.2007
UKUUG: Chris Mason przedstawia projekt BTRFS
06.2008
BTRFS v0.15
01.2009
v0.18 jest w 2.6.29-rc1
06.2009
v0.19 jest w 2.6.31. Automatyczna zmiana formatu.

B+Drzewa

B+Drzewo

Popularna w systemach plików i bazach danych modyfikacja BDrzew. Tylko liście zawierają dane, a węzły wewnętrzne służą jako indeks.



B+Drzewa

B+Drzewo

Popularna w systemach plików i bazach danych modyfikacja BDrzew. Tylko liście zawierają dane, a węzły wewnętrzne służą jako indeks.

Struktura używana w wielu systemach plików i implementacjach baz danych.

- NTFS, ReiserFS, NSS, XFS, JFS ← przechowywanie metadanych
- IBM DB2, Informix, Microsoft SQL Server, Oracle 8, Sybase ASI, PostgreSQL, MySQL ← pozwalają na tworzenie indeksów

B+Drzewa

B+Drzewo

Popularna w systemach plików i bazach danych modyfikacja BDrzew. Tylko liście zawierają dane, a węzły wewnętrzne służą jako indeks.

Struktura używana w wielu systemach plików i implementacjach baz danych.

- NTFS, ReiserFS, NSS, XFS, JFS ← przechowywanie metadanych
- IBM DB2, Informix, Microsoft SQL Server, Oracle 8, Sybase ASI, PostgreSQL, MySQL ← pozwalają na tworzenie indeksów

B+Drzewa

B+Drzewo

Popularna w systemach plików i bazach danych modyfikacja BDrzew. Tylko liście zawierają dane, a węzły wewnętrzne służą jako indeks.

Struktura używana w wielu systemach plików i implementacjach baz danych.

- NTFS, ReiserFS, NSS, XFS, JFS ← przechowywanie metadanych
- IBM DB2, Informix, Microsoft SQL Server, Oracle 8, Sybase ASI, PostgreSQL, MySQL ← pozwalają na tworzenie indeksów

B+Drzewa a BDrzewa

Jakie korzyści wynikają z używania B+Drzew?

- Duże rozgałęzienie $\longrightarrow$ wczytywanych jest mniej węzłów
- Liście połączone są w listę $\longrightarrow$ ułatwia przetwarzanie zakresów
- Mniejsza defragmentacja danych $\longrightarrow$ . . . mniejsza deframentacja danych

B+Drzewa a BDrzewa

Jakie korzyści wynikają z używania B+Drzew?

- Duże rozgałęzienie $\longrightarrow$ wczytywanych jest mniej węzłów
- Liście połączone są w listę $\longrightarrow$ ułatwia przetwarzanie zakresów
- Mniejsza defragmentacja danych $\longrightarrow$ . . . mniejsza deframentacja danych

B+Drzewa a BDrzewa

Jakie korzyści wynikają z używania B+Drzew?

- Duże rozgałęzienie $\longrightarrow$ wczytywanych jest mniej węzłów
- Liście połączone są w listę $\longrightarrow$ ułatwia przetwarzanie zakresów
- Mniejsza defragmentacja danych $\longrightarrow$ . . . mniejsza deframentacja danych

B+Drzewa a BDrzewa

Jakie korzyści wynikają z używania B+Drzew?

- Duże rozgałęzienie $\longrightarrow$ wczytywanych jest mniej węzłów
- Liście połączone są w listę $\longrightarrow$ ułatwia przetwarzanie zakresów
- Mniejsza defragmentacja danych $\longrightarrow$ . . . mniejsza deframentacja danych

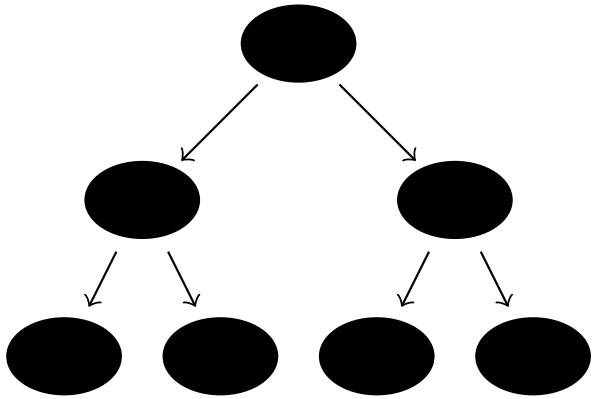
Co to jest *COW*?

copy-on-write czyli leniwe kopiowania. W ogólności współdzielenie identycznych fragmentów pomiędzy różnymi obiektami, ale także mechanizm zabezpieczający przed nadpisaniem i utratą danych.

Jak dodać *COW* do B+Drzewa?

Pomysł

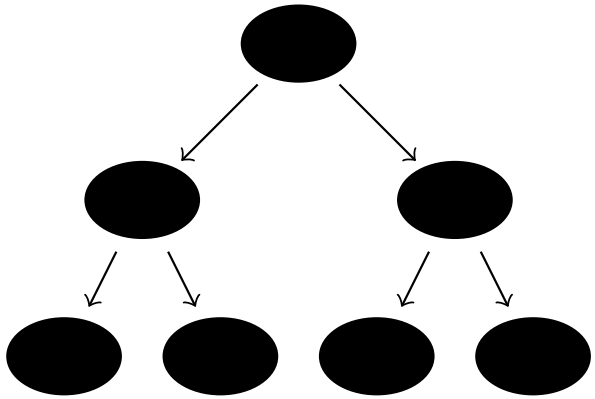
- Wszystkie węzły są stałego rozmiaru (jedna strona)
- Nie nadpisujemy stron



Jak dodać *COW* do B+Drzewa?

Pomysł

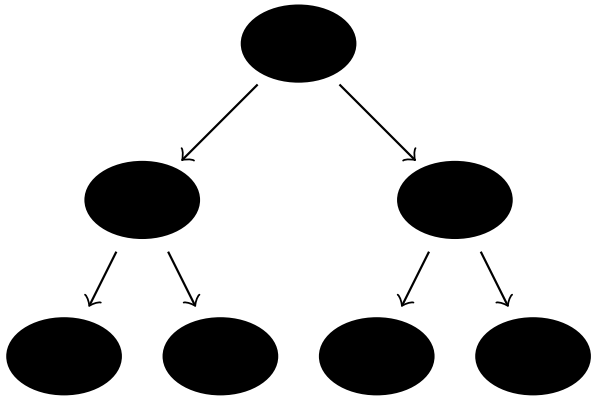
- Wszystkie węzły są stałego rozmiaru (jedna strona)
- Nie nadpisujemy stron



Jak dodać *COW* do B+Drzewa?

Pomysł

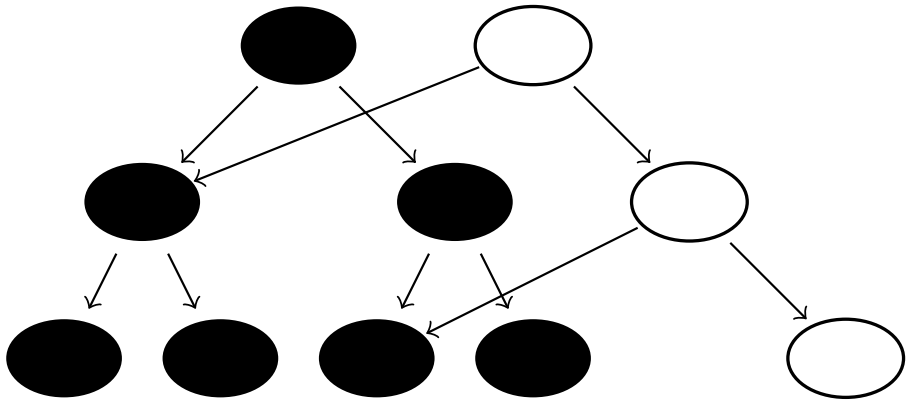
- Wszystkie węzły są stałego rozmiaru (jedna strona)
- Nie nadpisujemy stron



Jak dodać *COW* do B+Drzewa?

Pomysł

- Wszystkie węzły są stałego rozmiaru (jedna strona)
- Nie nadpisujemy stron



Jak to zrealizować?

Operacje wykonywane są iteracyjnie. Dla każdej z nich:

- Logujemy informację o wykonywanej operacji
- Wykonujemy suboperacje na nowych stronach *na boku*
- Między suboperacjami odsyłamy strony na dysk jako punkty kontrolne operacji
- Na końcu podmieniamy korzeń na ostateczny punkt kontrolny

W razie awarii możemy odtworzyć ostatni punkt kontrolny i zgodnie z logiem ponowić próbę.

Jak to zrealizować?

Operacje wykonywane są iteracyjnie. Dla każdej z nich:

- Logujemy informację o wykonywanej operacji
- Wykonujemy suboperacje na nowych stronach *na boku*
- Między suboperacjami odsyłamy strony na dysk jako punkty kontrolne operacji
- Na końcu podmieniamy korzeń na ostateczny punkt kontrolny

W razie awarii możemy odtworzyć ostatni punkt kontrolny i zgodnie z logiem ponowić próbę.

Jak to zrealizować?

Operacje wykonywane są iteracyjnie. Dla każdej z nich:

- Logujemy informację o wykonywanej operacji
- Wykonujemy suboperacje na nowych stronach *na boku*
- Między suboperacjami odsyłamy strony na dysk jako punkty kontrolne operacji
- Na końcu podmieniamy korzeń na ostateczny punkt kontrolny

W razie awarii możemy odtworzyć ostatni punkt kontrolny i zgodnie z logiem ponowić próbę.

Jak to zrealizować?

Operacje wykonywane są iteracyjnie. Dla każdej z nich:

- Logujemy informację o wykonywanej operacji
- Wykonujemy suboperacje na nowych stronach *na boku*
- Między suboperacjami odsyłamy strony na dysk jako punkty kontrolne operacji
- Na końcu podmieniamy korzeń na ostateczny punkt kontrolny

W razie awarii możemy odtworzyć ostatni punkt kontrolny i zgodnie z logiem ponowić próbę.

Jak to zrealizować?

Operacje wykonywane są iteracyjnie. Dla każdej z nich:

- Logujemy informację o wykonywanej operacji
- Wykonujemy suboperacje na nowych stronach *na boku*
- Między suboperacjami odsyłamy strony na dysk jako punkty kontrolne operacji
- Na końcu podmieniamy korzeń na ostateczny punkt kontrolny

W razie awarii możemy odtworzyć ostatni punkt kontrolny i zgodnie z logiem ponowić próbę.

Jak to zrealizować?

Operacje wykonywane są iteracyjnie. Dla każdej z nich:

- Logujemy informację o wykonywanej operacji
- Wykonujemy suboperacje na nowych stronach *na boku*
- Między suboperacjami odsyłamy strony na dysk jako punkty kontrolne operacji
- Na końcu podmieniamy korzeń na ostateczny punkt kontrolny

W razie awarii możemy odtworzyć ostatni punkt kontrolny i zgodnie z logiem ponowić próbę.

Migawki

- Wymagają przechowywania wskaźników do więcej niż jednego korzenia drzewa
- Każdy korzeń jest stanem systemu z jakiejś chwili
- Wystarczy nie podmieniać korzenia na końcu operacji

Obserwacja

Migawki nie muszą być tworzone w sposób liniowy, tzn. na migawce można oprzeć inną. Tego nie ma w ZFS.

Migawki

- Wymagają przechowywania wskaźników do więcej niż jednego korzenia drzewa
- Każdy korzeń jest stanem systemu z jakiejś chwili
- Wystarczy nie podmieniać korzenia na końcu operacji

Obserwacja

Migawki nie muszą być tworzone w sposób liniowy, tzn. na migawce można oprzeć inną. Tego nie ma w ZFS

Migawki

- Wymagają przechowywania wskaźników do więcej niż jednego korzenia drzewa
- Każdy korzeń jest stanem systemu z jakiejś chwili
- Wystarczy nie podmieniać korzenia na końcu operacji

Obserwacja

Migawki nie muszą być tworzone w sposób liniowy, tzn. na migawce można oprzeć inną. Tego nie ma w ZFS

A współbieżność?

Standardowe operacje na B+Drzewach znajdują liść i modyfikują drzewo w drodze do góry $\longrightarrow$ korzeń jest punktem kolizyjnym.

Rozwiązanie

Wszystkie operacje wykonywać od góry do dołu blokując maksymalnie trzy węzły.

Pozostał *tylko* problem z utrzymaniem balansu...

A współbieżność?

Standardowe operacje na B+Drzewach znajdują liść i modyfikują drzewo w drodze do góry $\longrightarrow$ korzeń jest punktem kolizyjnym.

Rozwiązanie

Wszystkie operacje wykonywać od góry do dołu blokując maksymalnie trzy węzły.

Pozostał *tylko* problem z utrzymaniem balansu...

A współbieżność?

Standardowe operacje na B+Drzewach znajdują liść i modyfikują drzewo w drodze do góry $\longrightarrow$ korzeń jest punktem kolizyjnym.

Rozwiązanie

Wszystkie operacje wykonywać od góry do dołu blokując maksymalnie trzy węzły.

Pozostał *tylko* problem z utrzymaniem balansu...

A współbieżność?

Standardowe operacje na B+Drzewach znajdują liść i modyfikują drzewo w drodze do góry $\longrightarrow$ korzeń jest punktem kolizyjnym.

Rozwiązanie

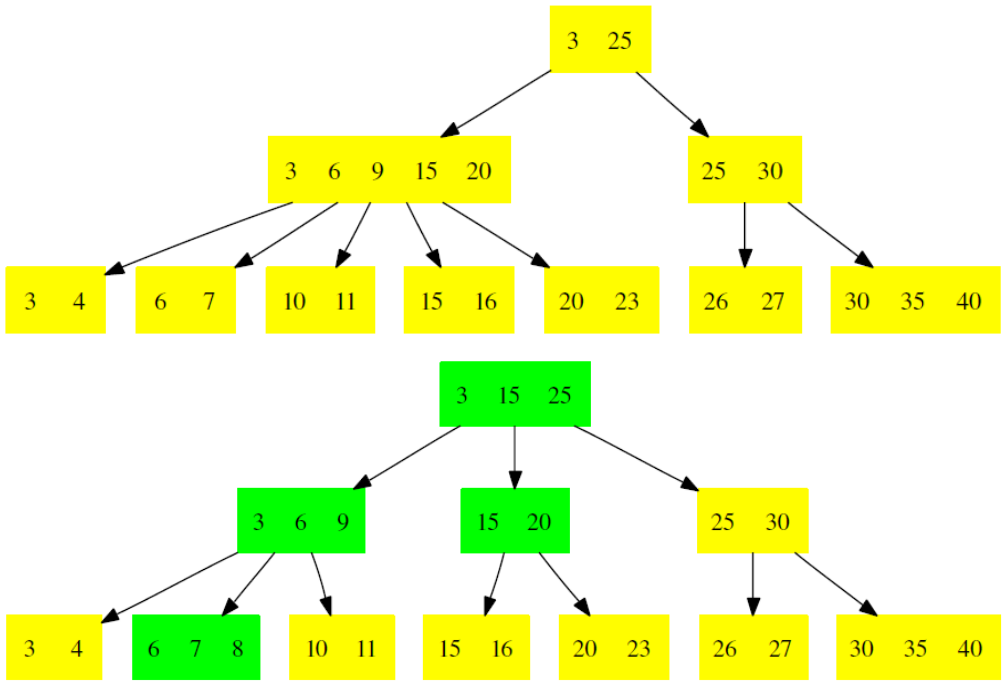
Wszystkie operacje wykonywać od góry do dołu blokując maksymalnie trzy węzły.

Pozostał *tylko* problem z utrzymaniem balansu...

Wstawianie węzła

- Modyfikowane elementy kopiowane są przy zagłębianiu się
- "Zapobiegawcze" rozdzielanie węzłów

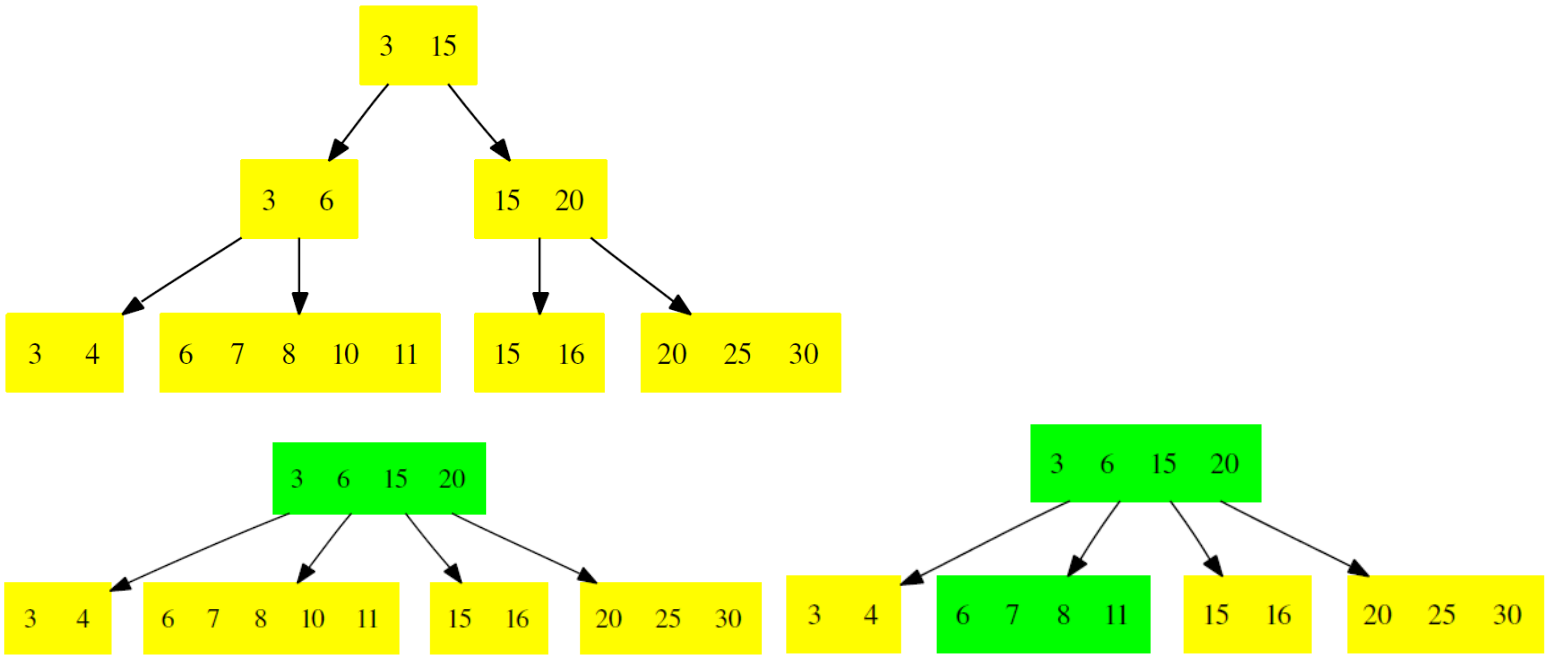
Przykład (Wstawienie 8):



Usuwanie węzła

- Modyfikowane elementy kopiowane są przy zagłębianiu się
- "Zapobiegawcze" łączenie węzłów

Przykład (Usunięcie 10):



Konsekwencje

- Drzewo jest mniej zrównoważone ($d \ldots 3d$)

- Nie ma zakleszczenia
- Algorytmy są proste

Klonowanie

A co jeśli chcemy móc tanio skopiować drzewo?

Wymagania

- Współdzielić wszystko co możliwe
- Klonowanie ma być szybkie
- Liczba klonów nie ma być zbyt mocno ograniczona
- Klony też mają być klonowalne

Pomysły

- Skopiowanie całej struktury
- Multizbiory bitowe
- Zliczanie referencji

Klonowanie

A co jeśli chcemy móc tanio skopiować drzewo?

Wymagania

- Współdzielić wszystko co możliwe
- Klonowanie ma być szybkie
- Liczba klonów nie ma być zbyt mocno ograniczona
- Klony też mają być klonowalne

Pomysły

- Skopiowanie całej struktury
- Multizbiory bitowe
- Zliczanie referencji

Klonowanie

A co jeśli chcemy móc tanio skopiować drzewo?

Wymagania

- Współdzielić wszystko co możliwe
- Klonowanie ma być szybkie
- Liczba klonów nie ma być zbyt mocno ograniczona
- Klony też mają być klonowalne

Pomysły

- Skopiowanie całej struktury
- Multizbiory bitowe
- Zliczanie referencji

Klonowanie: Zliczanie referencji

Każdy blok ma licznik użyć (0 oznacza wolną przestrzeń).

Aby skopiować drzewo wystarczy dokonać inkrementacji licznika użyć na kopiowanych węzłach.

Pomysł

Można to zrobić leniwie korzystając z COW.

Klonowanie: Zliczanie referencji

Każdy blok ma licznik użyć (0 oznacza wolną przestrzeń).

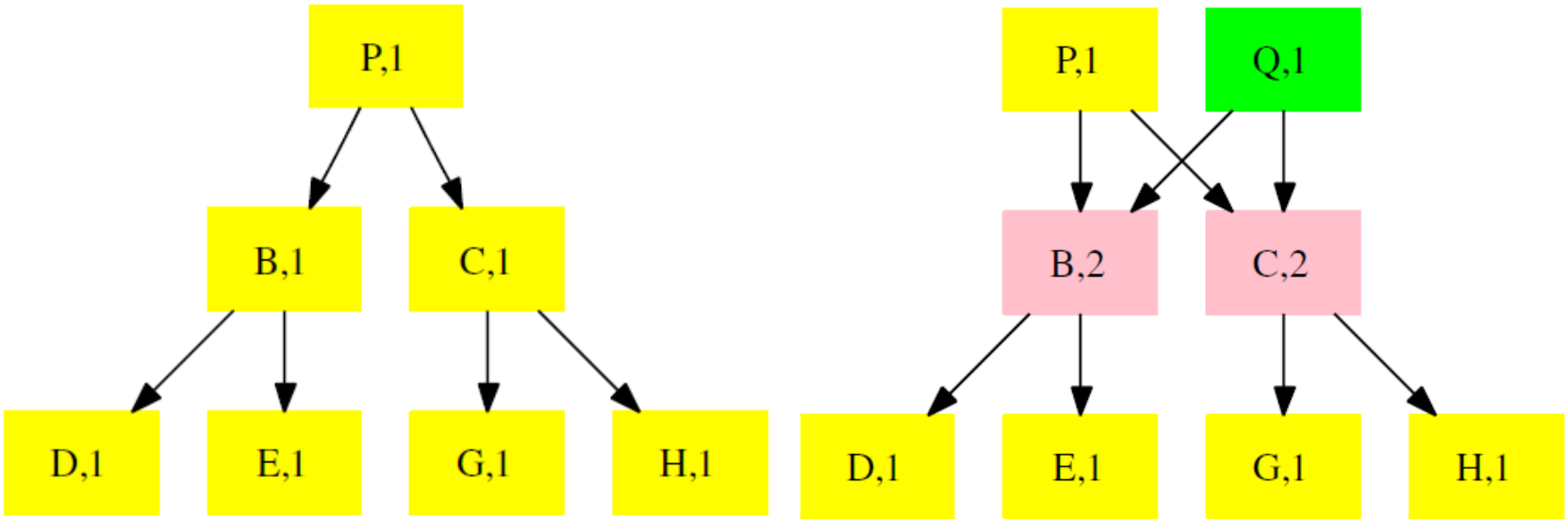
Aby skopiować drzewo wystarczy dokonać inkrementacji licznika użyć na kopiowanych węzłach.

Pomysł

Można to zrobić leniwie korzystając z COW.

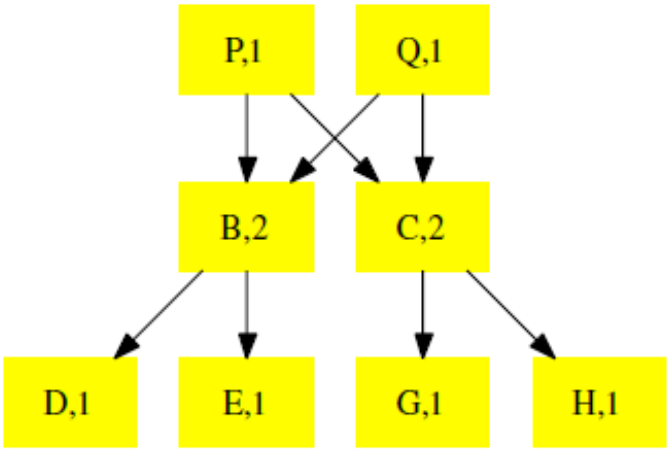
Klonowanie: przykład

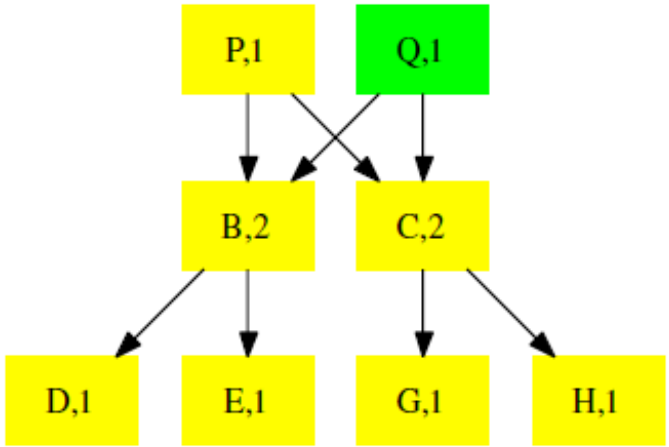
Klonujemy P:

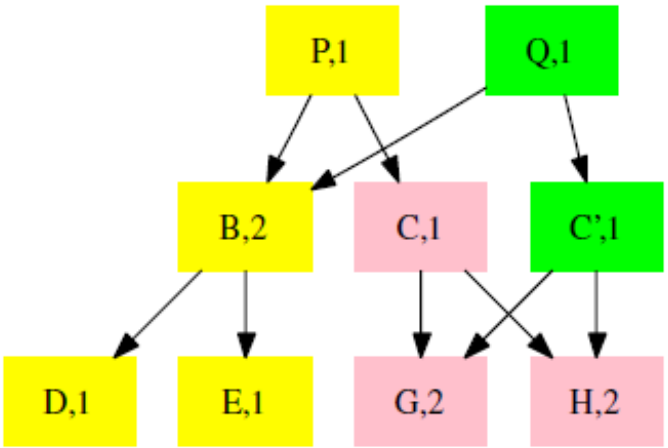


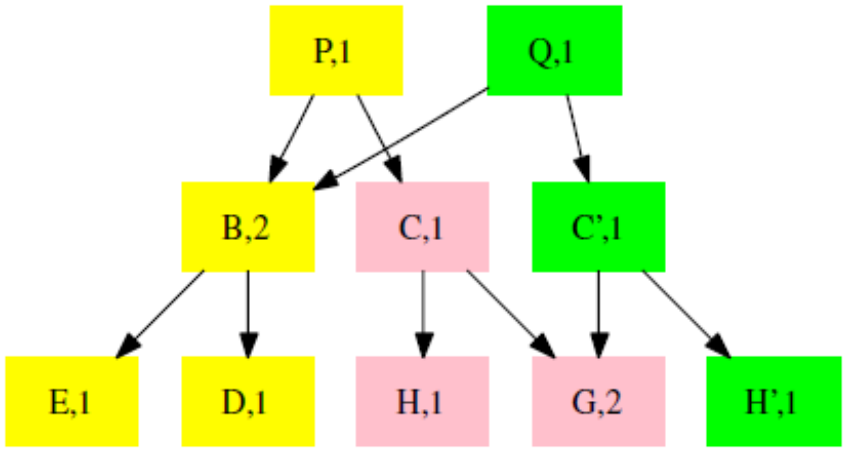
Wstawianie do klona

Wstawiamy do liścia H w drzewie Q:

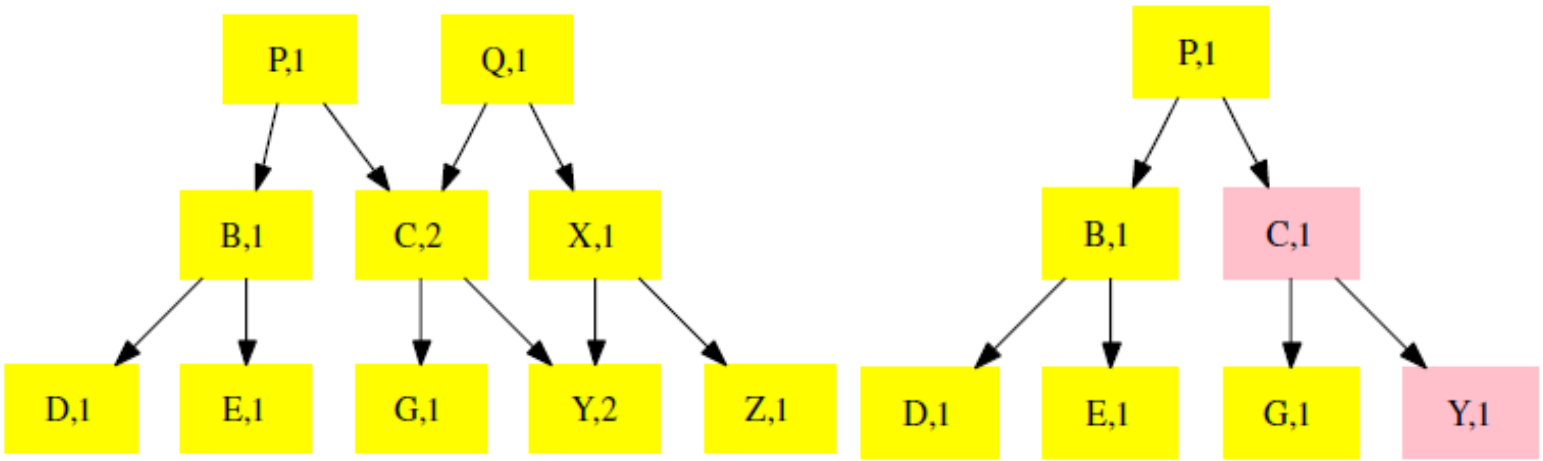








Usuwanie klona



Jak zrobić z tego system plików?

Pomysł

Wrzucić wszystko do B+Drzewa z *COW*. W tym informację o przestrzeni dyskowej zajętej przez drzewo.

Efekt

Jeden kod obsługuje wszystkie elementy i ułatwia implementację m.in.: migawek, klonowania, kompresji i automatycznej weryfikacji sum kontrolnych

Jak zrobić z tego system plików?

Pomysł

Wrzucić wszystko do B+Drzewa z *COW*. W tym informację o przestrzeni dyskowej zajętej przez drzewo.

Efekt

Jeden kod obsługuje wszystkie elementy i ułatwia implementację m.in.: migawek, klonowania, kompresji i automatycznej weryfikacji sum kontrolnych

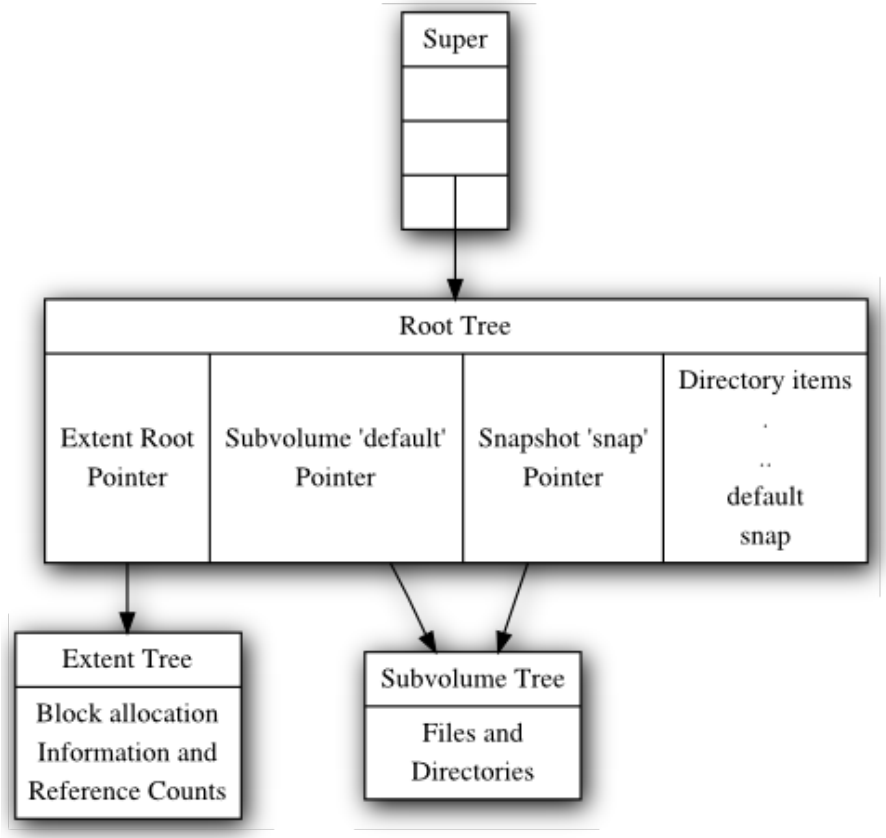
Jak zrobić z tego system plików?

Pomysł

Wrzucić wszystko do B+Drzewa z *COW*. W tym informację o przestrzeni dyskowej zajętej przez drzewo.

Efekt

Jeden kod obsługuje wszystkie elementy i ułatwia implementację m.in.: migawek, klonowania, kompresji i automatycznej weryfikacji sum kontrolnych



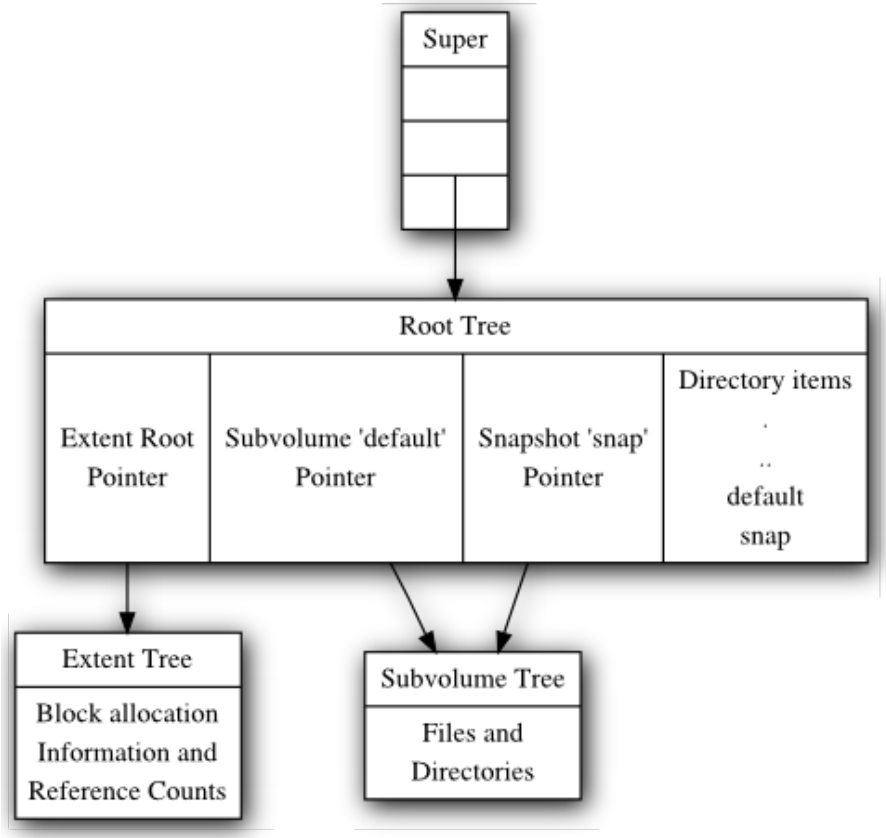
Jak zrobić z tego system plików?

Pomysł

Wrzucić wszystko do B+Drzewa z *COW*. W tym informację o przestrzeni dyskowej zajętej przez drzewo.

Efekt

Jeden kod obsługuje wszystkie elementy i ułatwia implementację m.in.: migawek, klonowania, kompresji i automatycznej weryfikacji sum kontrolnych



Drzewa systemu

Superblok wskazuje miejsce położenia drzewa korzeni (*tree of roots*).

Drzewo korzeni

Drzewo korzeni ma wskaźniki na drzewa:

- systemu plików
- alokacji obszarów
- alokacji dla poszczególnych urządzeń blokowych

Format węzła

Każdy węzeł jest zawarty w jednym bloku i zawiera nagłówek.
Węzły wewnętrzne składają się z list (klucz, wskaźnik na blok)
Liście przechowują tablice obiektów:

Format węzła

Każdy węzeł jest zawarty w jednym bloku i zawiera nagłówek.
Węzły wewnętrzne składają się z list (klucz, wskaźnik na blok)
Liście przechowują tablice obiektów:

Format węzła

Każdy węzeł jest zawarty w jednym bloku i zawiera nagłówek.
Węzły wewnętrzne składają się z list (klucz, wskaźnik na blok)
Liście przechowują tablice obiektów:

Format węzła

Każdy węzeł jest zawarty w jednym bloku i zawiera nagłówek.
Węzły wewnętrzne składają się z list (klucz, wskaźnik na blok)

Liście przechowują tablice obiektów:

Struktury danych

Drzewo zapisywane jest na dysku przy pomocy trzech struktur danych. Są to:

- Nagłówek bloku
- Klucz obiektu
- Obiekt

Struktury danych

Drzewo zapisywane jest na dysku przy pomocy trzech struktur danych. Są to:

- Nagłówek bloku
- Klucz obiektu
- Obiekt

Metadane weryfikujące:

- suma kontrolna bieżącego bloku
- UUID systemu plików
- poziom elementu w drzewie
- numer bloku w którym jesteśmy
- numer rewizji/generacji

Poza tym zawiera flagi i identyfikator obszaru

Struktury danych

Drzewo zapisywane jest na dysku przy pomocy trzech struktur danych. Są to:

- Nagłówek bloku
- Klucz obiektu
- Obiekt

- ID (64bity)
- typ (8 bitów)
- przesunięcie (64bity)

Struktury danych

Drzewo zapisywane jest na dysku przy pomocy trzech struktur danych. Są to:

- Nagłówek bloku
- Klucz obiektu
- Obiekt

- klucz obiektu
- rozmiar (32bity)
- przesunięcie (32bity)

Obiekty systemu

Najważniejszą częścią klucza jest identyfikator obiektu.

Elementy dotyczącego tego samego obiektu będą się grupowały blisko siebie.

iwęzły mają zawsze najniższy numer klucza dla danego obiektu (typ= 1,offset= 0). Z założenia mają być małe i zawierać tylko podstawowe informacje.

Obiekty systemu

Najważniejszą częścią klucza jest identyfikator obiektu.

Elementy dotyczącego tego samego obiektu będą się grupowały blisko siebie.

iwęzły mają zawsze najniższy numer klucza dla danego obiektu (typ= 1,offset= 0). Z założenia mają być małe i zawierać tylko podstawowe informacje.

Obiekty systemu

Najważniejszą częścią klucza jest identyfikator obiektu.

Elementy dotyczącego tego samego obiektu będą się grupowały blisko siebie.

iwęzły mają zawsze najniższy numer klucza dla danego obiektu (typ= 1,offset= 0). Z założenia mają być małe i zawierać tylko podstawowe informacje.

Jak trzymane są pliki?

- Dane małych plików mogą być zapisane bezpośrednio w drzewie.
- Duże pliki przechowywane są w obszarach (*extent*) poza drzewem. Odnośnik do obszaru (btrfs_file_extent_item) zawiera numer rewizji oraz umiejscowienie danych na dysku.
- Zapis do środka obszaru odbywa się bez odczytu danych.

Jak trzymane są pliki?

- Dane małych plików mogą być zapisane bezpośrednio w drzewie.
- Duże pliki przechowywane są w obszarach (*extent*) poza drzewem. Odnośnik do obszaru (btrfs_file_extent_item) zawiera numer rewizji oraz umiejscowienie danych na dysku.
- Zapis do środka obszaru odbywa się bez odczytu danych.

Jak trzymane są foldery?

Obiekt folderu nadrzędnego jest podwójnie indeksowany, tzn. posiada po dwa elementy dla każdego podfolderu lub pliku, który jest w nim zawarty.

- indeks po nazwach plików. Prawa część klucza elementów tego indeksu zawiera hash nazwy.

Directory Objectid	BTRFS_DIR_ITEM_KEY	64 bit filename hash

- indeks po (przybliżonej) kolejności na dysku. Każdy nowy element dostaje pierwszy wolny numer.

Directory Objectid	BTRFS_DIR_INDEX_KEY	Inode Objectid

Obszary danych

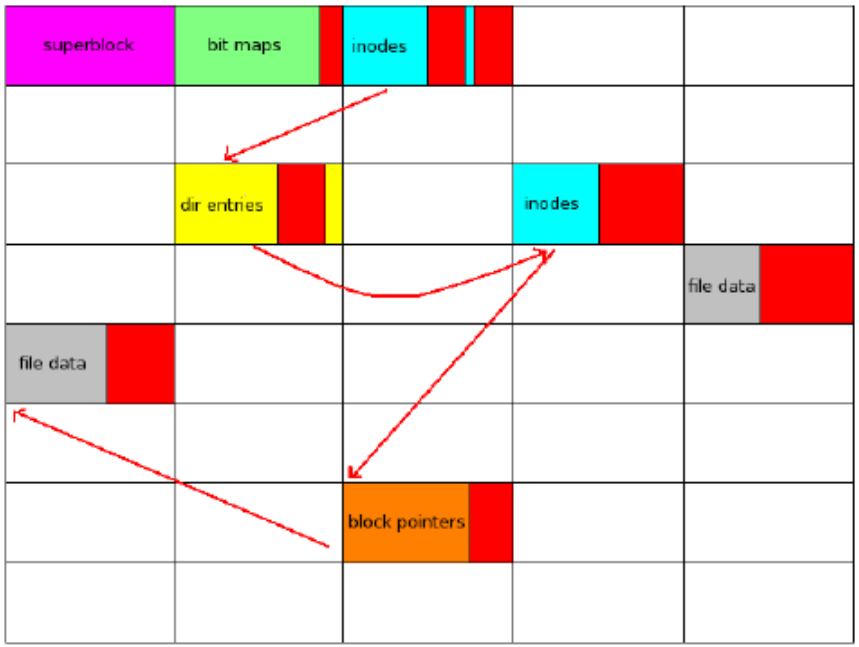
Obszary danych pogrupowane są w duże grupy (*block groups*).

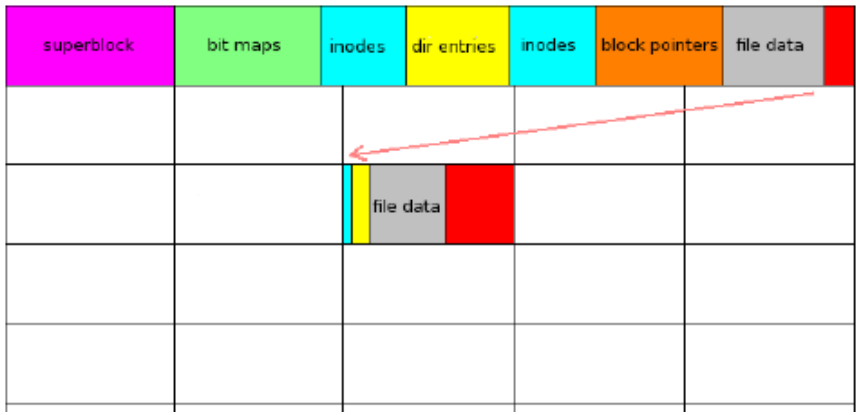
- Grupy są zmiennej wielkości (inaczej niż w EXT3)
- Naprzemiennie na dysku występują grupy z różnymi preferencjami (metadane lub dane)

Opisy wszystkich obszarów trzymane są w osobnym drzewie (*extent allocation tree*), a ich kluczami są adresy przesunięcia na dysku.

- Każdy obszar posiada listę wszystkich obiektów systemu plików, które go współdzielą
- Zawarta jest liczba referencji, która służy też jako opis wolnej przestrzeni

Wykorzystanie obszaru dysku





Zarządzanie przestrzenią i migracje

Dzięki drzewie obszarów łatwo odnaleźć obiekty zajmujące określone miejsce na dysku.

Możliwości

- Stabilna zmiana rozmiaru systemu
- Swobodna administracja obszarem dysku
- Wydajna defragmentacja

Zarządzanie przestrzenią i migracje

Dzięki drzewie obszarów łatwo odnaleźć obiekty zajmujące określone miejsce na dysku.

Możliwości

- Stabilna zmiana rozmiaru systemu
- Swobodna administracja obszarem dysku
- Wydajna defragmentacja

Zarządzanie przestrzenią i migracje

Dzięki drzewie obszarów łatwo odnaleźć obiekty zajmujące określone miejsce na dysku.

Możliwości

- Stabilna zmiana rozmiaru systemu
- Swobodna administracja obszarem dysku
- Wydajna defragmentacja

Zarządzanie przestrzenią i migracje

Dzięki drzewie obszarów łatwo odnaleźć obiekty zajmujące określone miejsce na dysku.

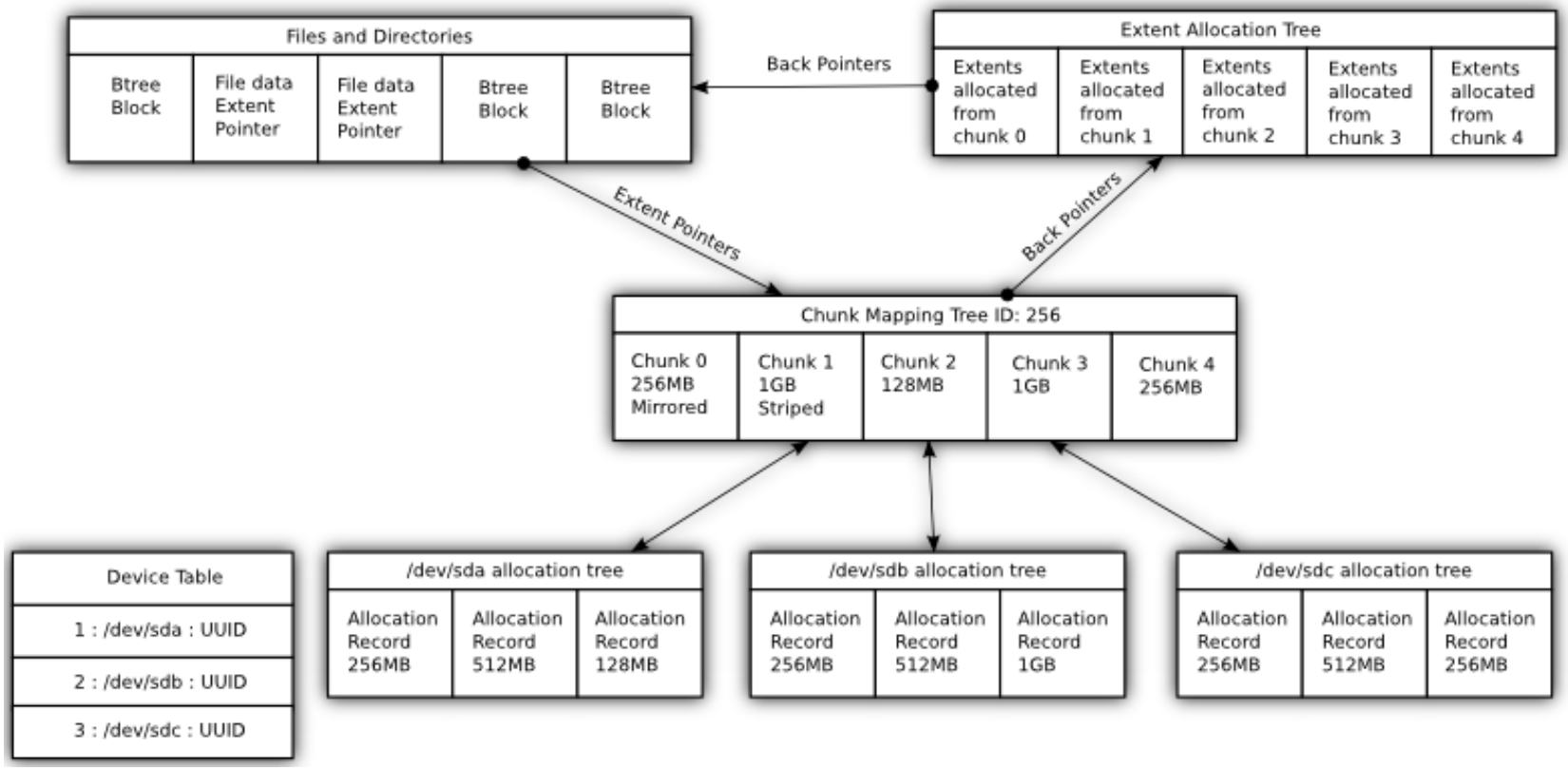
Możliwości

- Stabilna zmiana rozmiaru systemu
- Swobodna administracja obszarem dysku
- Wydajna defragmentacja

Macierze wielodyskowe

Obsługa wielu dysków zaimplementowana na poziomie systemu plików. Dla każdego urządzenia przynajmniej jedno drzewo alokacji obszarów.

- Administrator może oznaczać pliki, które mają być duplikowane
- Niezależność od sprzętu
- Ułatwienie migracji na nowy dysk


Files and Directories
Btree Block
File data Extent Pointer
File data Extent Pointer
Btree Block
Btree Block
Back Pointers
Extent Allocation Tree
Extents allocated from chunk 0
Extents allocated from chunk 1
Extents allocated from chunk 2
Extents allocated from chunk 3
Extents allocated from chunk 4
Extent Pointers
Back Pointers
Chunk Mapping Tree ID: 256
Chunk 0 256MB Mirrored
Chunk 1 1GB Striped
Chunk 2 128MB
Chunk 3 1GB
Chunk 4 256MB
Device Table
1 : /dev/sda : UUID
2 : /dev/sdb : UUID
3 : /dev/sdc : UUID
/dev/sda allocation tree
Allocation Record 256MB
Allocation Record 512MB
Allocation Record 128MB
/dev/sdb allocation tree
Allocation Record 256MB
Allocation Record 512MB
Allocation Record 1GB
/dev/sdc allocation tree
Allocation Record 256MB
Allocation Record 512MB
Allocation Record 256MB

Sprawdzanie danych: Btrfsck

Właściwości

- Wysoka tolerancja na błędy
- Niewymagane odmontowanie
- Wydajne rozmieszczenie struktur powoduje, że głównym ograniczeniem jest prędkość wczytywania danych

Btrfsck potrzebuje 3 razy więcej ramu od ext2fsck

Sprawdzanie danych: Btrfsck

Właściwości

- Wysoka tolerancja na błędy
- Niewymagane odmontowanie
- Wydajne rozmieszczenie struktur powoduje, że głównym ograniczeniem jest prędkość wczytywania danych

Btrfsck potrzebuje 3 razy więcej ramu od ext2fsck

Redundancja

Każdy blok posiada w nagłówku bardzo dużo powtórzonych informacji.

- Zwiększa to prawdopodobieństwo wykrycia błędów.
- Ułatwia odzyskiwanie danych z uszkodzonego systemu plików

Kompresja

Obszary danych plików mogą korzystać z transparentnej kompresji. Obecnie zaimplementowana jest to przy pomocy zliba, ale w planach jest wsparcie LZOPa.

Konwersja EXT3 ↔ BTRFS

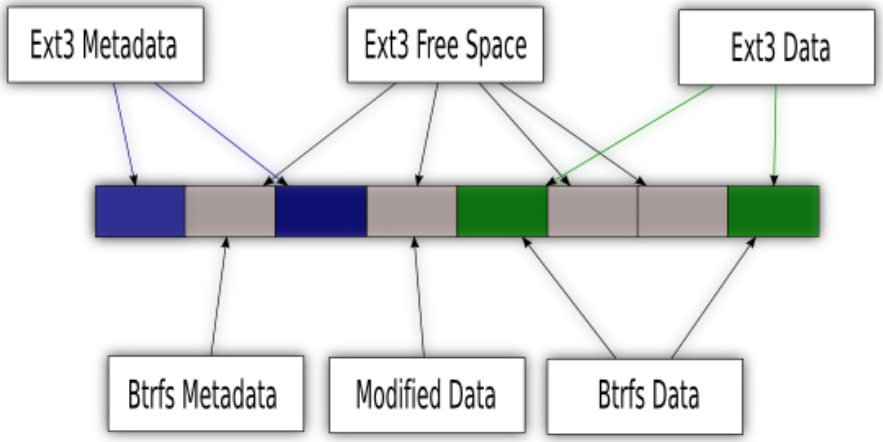
Obserwacja

BTRFS ma bardzo mało elementów o stałym umiejscowieniu na dysku.

Konwersja EXT3 ↔ BTRFS

Obserwacja

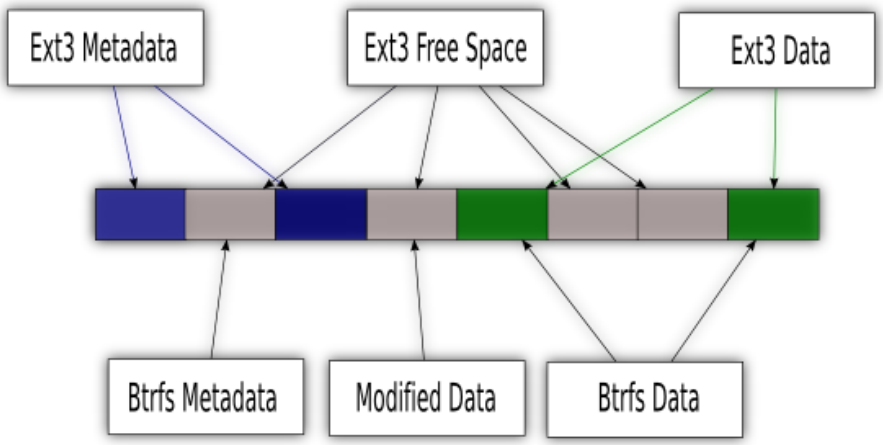
BTRFS ma bardzo mało elementów o stałym umiejscowieniu na dysku.



Wniosek

Struktury systemu plików można umiejscowić w wolnych blokach konwertowanego systemu.

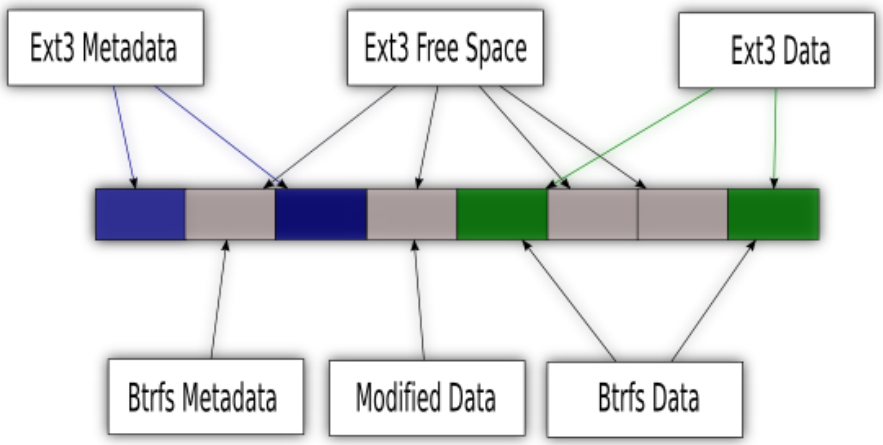
Konwersja EXT3 ↔ BTRFS, c.d.



Algorytm

1. Skopiowanie 1MB danych z początku pliku
2. Zduplikowanie struktury katalogów i inode'ów do BTRFS
3. Podłączenie bloków danych EXT3 do plików BTRFS
4. Oznaczenie metadanych EXT3 jako dane BTRFS

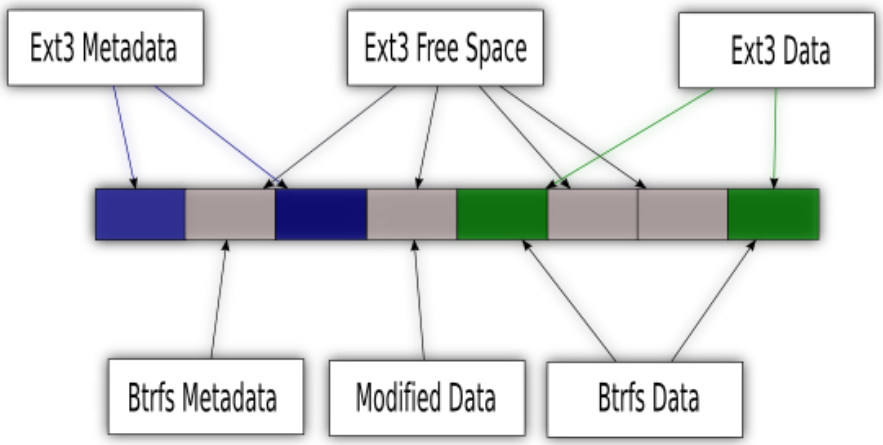
Konwersja EXT3 ↔ BTRFS, c.d.



Algorytm

1. Skopiowanie 1MB danych z początku pliku
2. Zduplikowanie struktury katalogów i inode'ów do BTRFS
3. Podłączenie bloków danych EXT3 do plików BTRFS
4. Oznaczenie metadanych EXT3 jako dane BTRFS

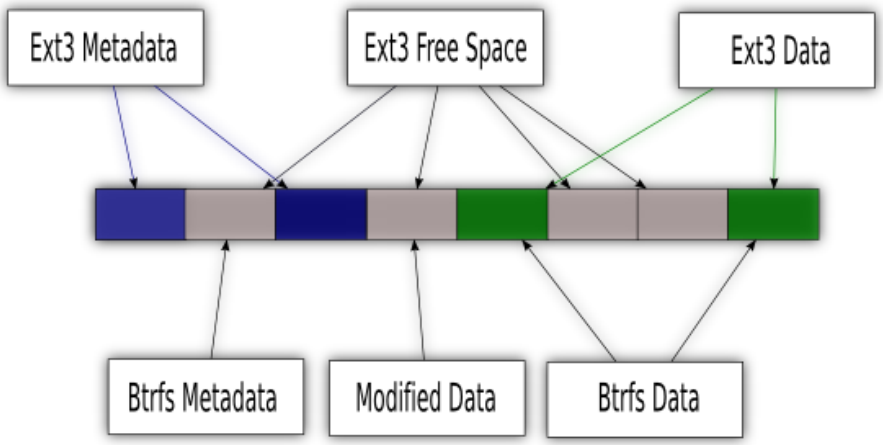
Konwersja EXT3 ↔ BTRFS, c.d.



Algorytm

1. Skopiowanie 1MB danych z początku pliku
2. Zduplikowanie struktury katalogów i inode'ów do BTRFS
3. Podłączenie bloków danych EXT3 do plików BTRFS
4. Oznaczenie metadanych EXT3 jako dane BTRFS

Konwersja EXT3 ↔ BTRFS, c.d.



Algorytm

1. Skopiowanie 1MB danych z początku pliku
2. Zduplikowanie struktury katalogów i inode'ów do BTRFS
3. Podłączenie bloków danych EXT3 do plików BTRFS
4. Oznaczenie metadanych EXT3 jako dane BTRFS

Zaimplementowano

- Migawki, Migawki migawek
- Używanie jednego systemu plików jako Migawki dla drugiego
- Dodawanie i usuwanie urządzeń dyskowych w locie
- Defragmentacja online
- Kompresja w locie
- Obsługa wielu dysków (RAID1 i RAID0)
- Transakcje z poziomu użytkownika
- Wsparcie dla SSD (*wear leveling*)

W planach

- Operacje asynchroniczne
- Bardziej zaawansowana obsługa macierzy (RAID5 i RAID6)
- Automatyczne kopie inkrementalne

- Dobrze się zapowiada
- Wiele ciekawych udogodnień
- Nadaje się nie tylko do serwerów

http://www.cs.tau.ac.il/ ohadrode/papers/btree_TOS.pdf

http://www.cs.tau.ac.il/ ohadrode/papers/LinuxFS_Workshop.pdf

http://www.linuxconf.eu/2007/papers/Mason.pdf

http://oss.oracle.com/projects/btrfs/dist/documentation/btrfs-ukuug.pdf

http://dclug.tux.org/200908/BTRFS-DCLUG.pdf

http://www.caiss.org/docs/DinnerSeminar/TheStorageChasm20090205.pdf

http://lwn.net/Articles/342892/

http://en.wikipedia.org/wiki/Btrfs

http://btrfs.wiki.kernel.org/