Btrfsってどんなファイルシステム?

btrfs +

2013-06-04 11:38:14

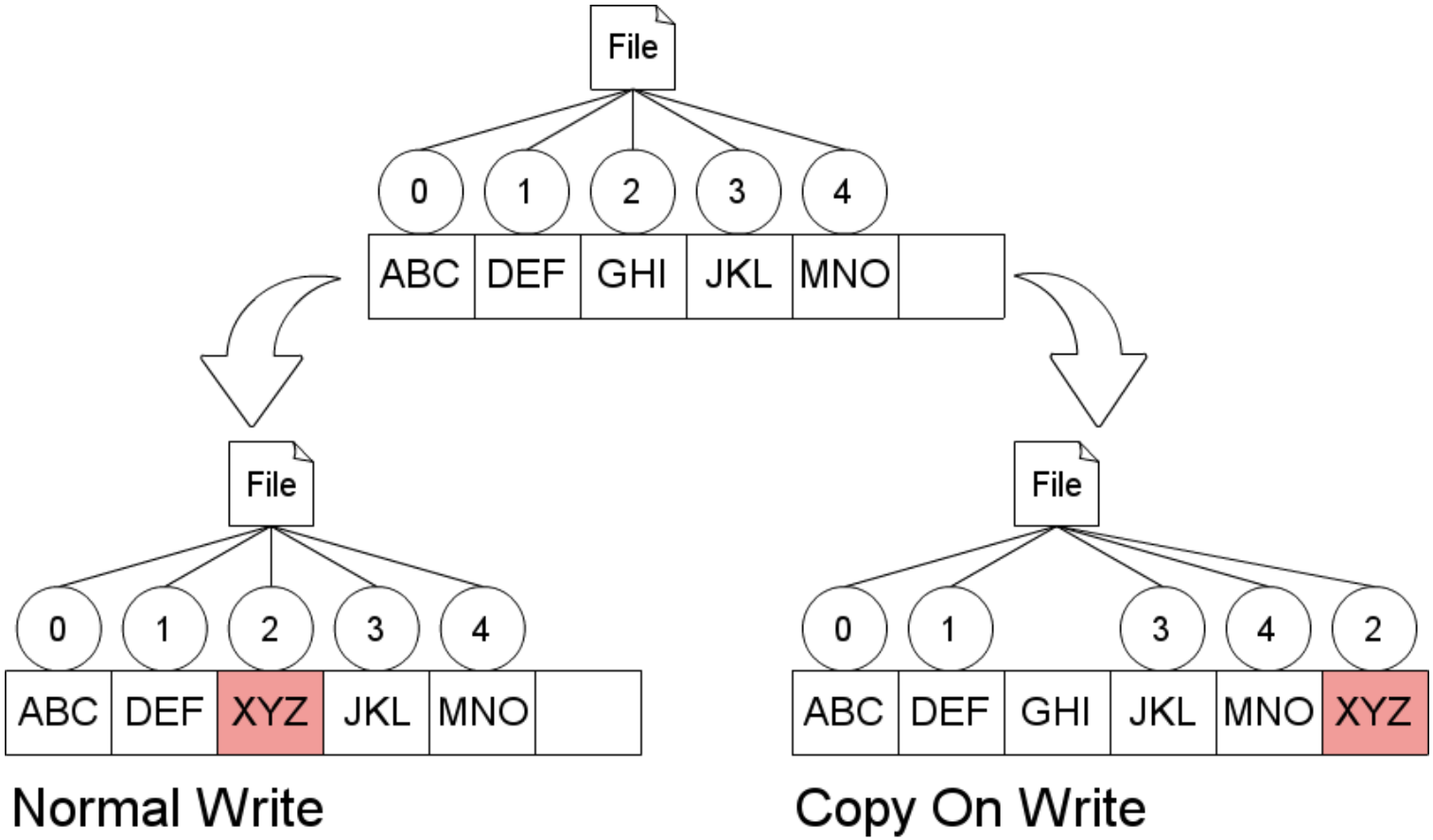
Btrfsは非常に安定した安心なファイルシステムです

Btrfsは非常に安定した安心なファイルシステムです

by naota344

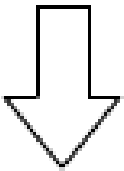
43 fav 11652 view いいね！ 25 Tweet 213 B! 36

Copy On Write



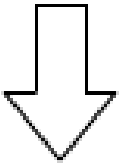
Journal

0	1	2	3	4	5
BtrFS	is	stable		Use	it.

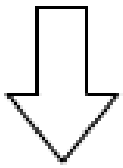


0	1	2	3	4	5
BtrFS	is	unstable	Don't	Use	it.

0	1	2	3	4	5
BtrFS	is	stable		Use	it.

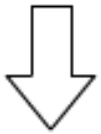


0	1	2	3	4	5
BtrFS	is	unstable		Use	it.



```
_人人人人人人人人人人人人_
>    突然のKernel Panic     <
￣Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y￣
```

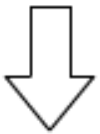
0	1	2	3	4	5
BtrFS	is	stable		Use	it.



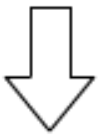
2に"unstable",
3に"Don't"を書きます

Journal Log

0	1	2	3	4	5
BtrFS	is	unstable		Use	it.



```
_人人人人人人人人人人人_
>   突然のKernel Panic    <
￣Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y￣
```



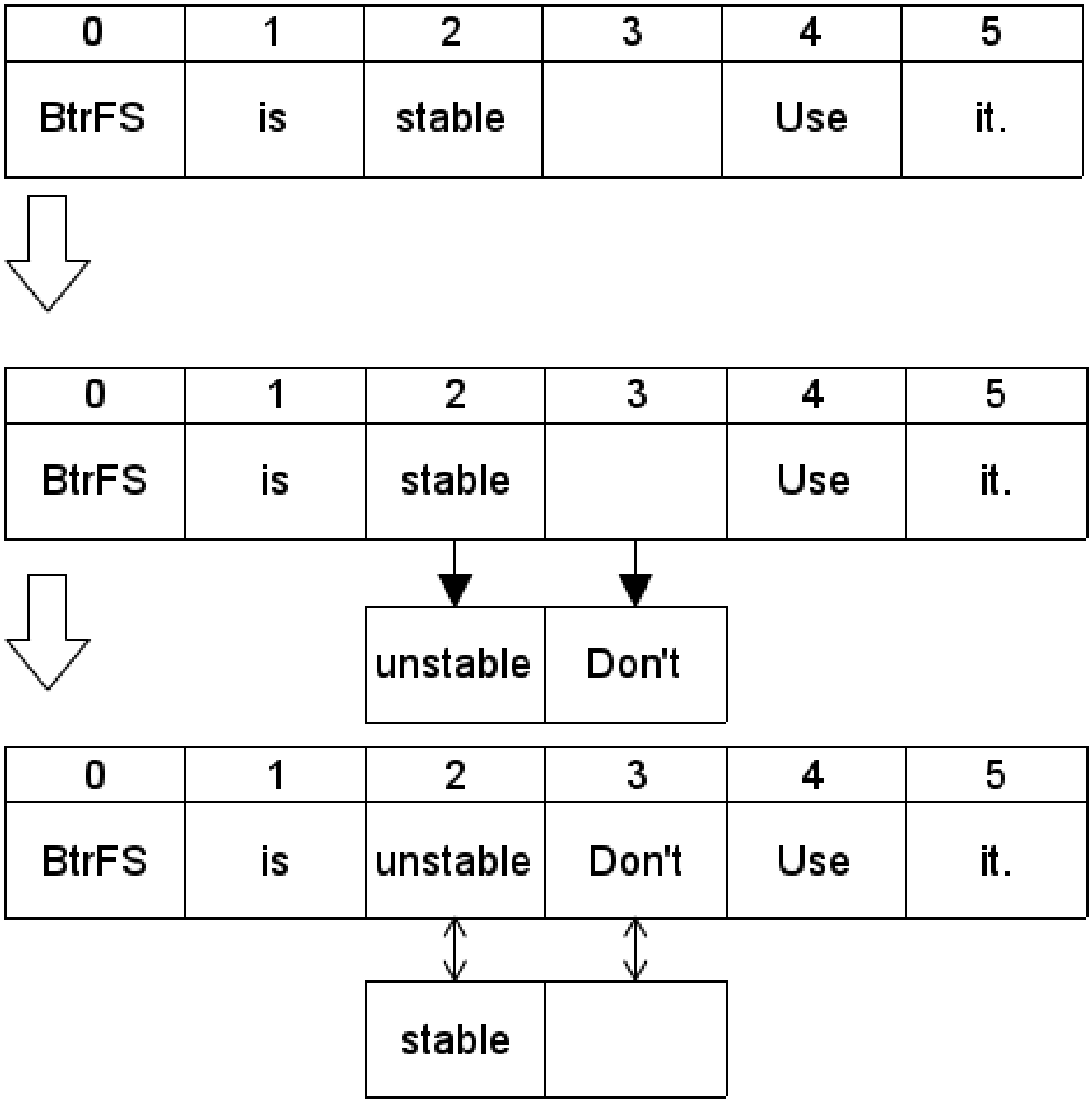
2に"unstable",
3に"Don't"を書きます

Journal Log

0	1	2	3	4	5
BtrFS	is	unstable	Don't	Use	it.

Journal の弱点

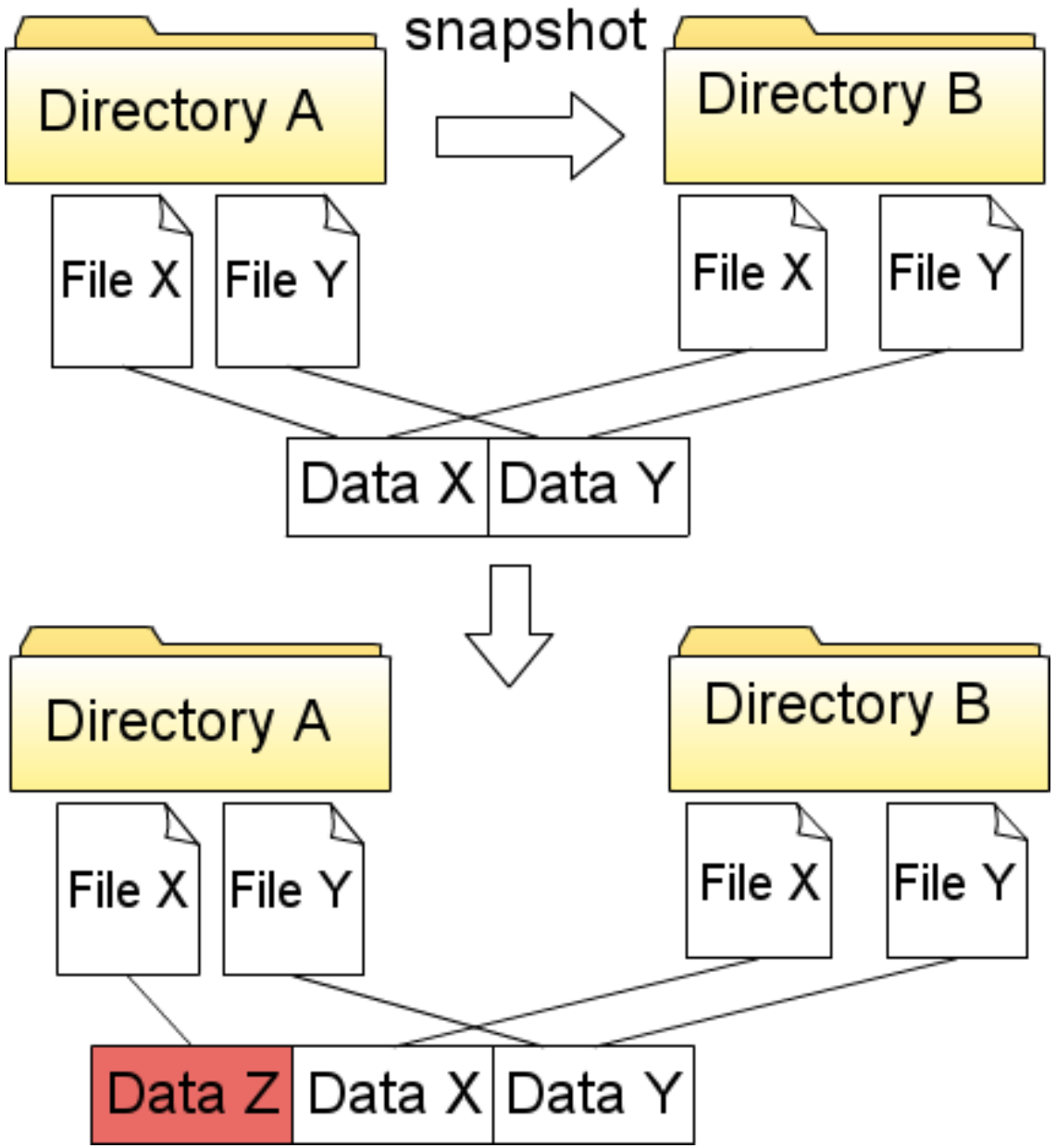
- 「ジャーナル」と「本体」と二回書かなければいけない
  - データもジャーナル化するのは効率が悪い
  - メタデータならいける


0
1
2
3
4
5
BtrFS
is
stable
Use
it.
unstable
Don't

Copy On Write

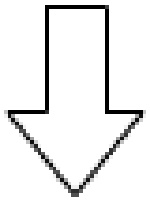
- 一回しか書かなくていい
- データ・メタデータ両方の整合性を保つのに使える

スナップショット


snapshot
Directory A
Directory B
File X
File Y
File X
File Y
Data X
Data Y
Directory A
Directory B
File X
File Y
File X
File Y
Data Z
Data X
Data Y

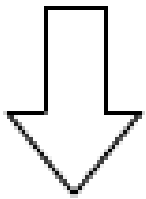
デフラグ

1	2	3	4	5		
A	B	C	D	E		



2=Fを書く

1		3	4	5	2	
A	B	C	D	E	F	



4=Gを書く

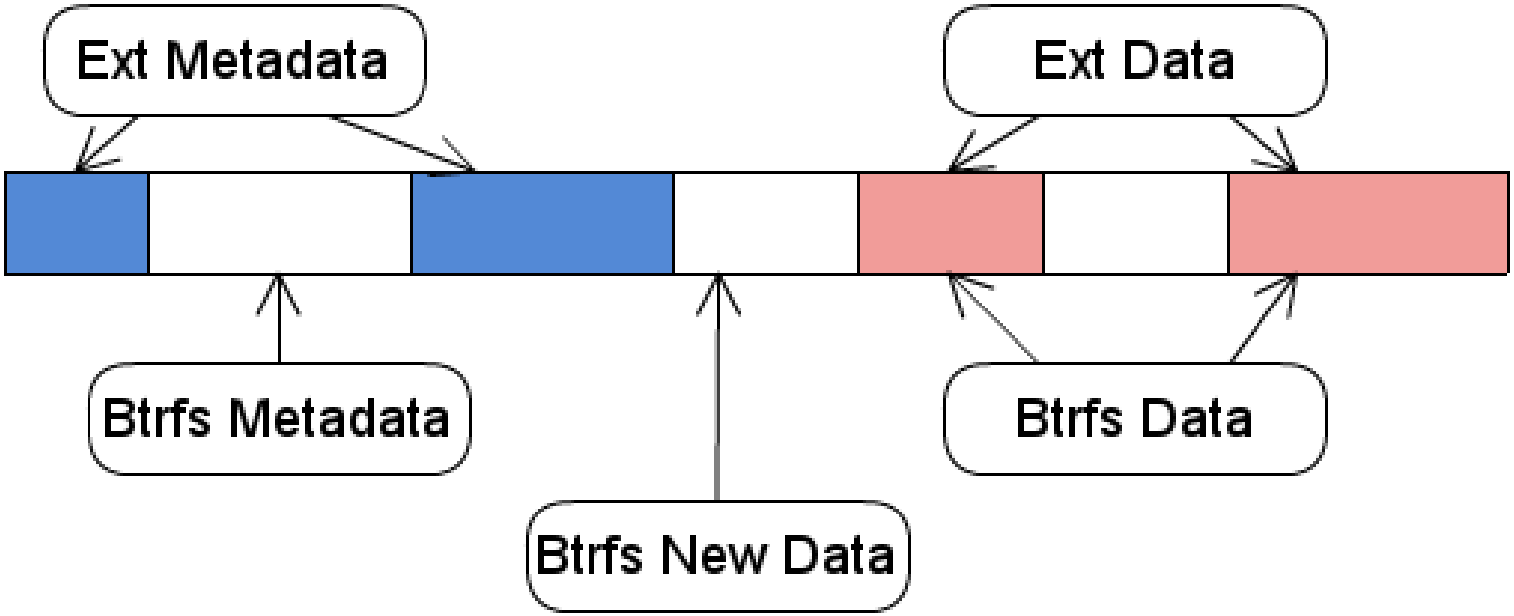
1		3		5	2	4
A	B	C	D	E	F	G

Quota

- QGroup と呼ばれる特殊な quota
  - スナップショットがあるので「共有データ」が存在する
  - 「全体のサイズ」と「共有されていないサイズ」による制限ができる
- 去年の Software Design10 月号に載ってます

Ext2/3 からの convert

- ext2/3 のデータがそのまま変換できます
- 気にいらなければ元に戻せます



その他

- 16EiB まで使える (XFS の 2 倍!)
- checksum があるからブロックの整合性チェック OK!
- 透過的圧縮で容量節約!
- RAID で信頼性向上!
- send/receive で効率的なバックアップ!
- Hot add/remove でらくらくディスク交換!
- dedup で重複排除!
- SSD 用最適化もあるよ!

将来的に

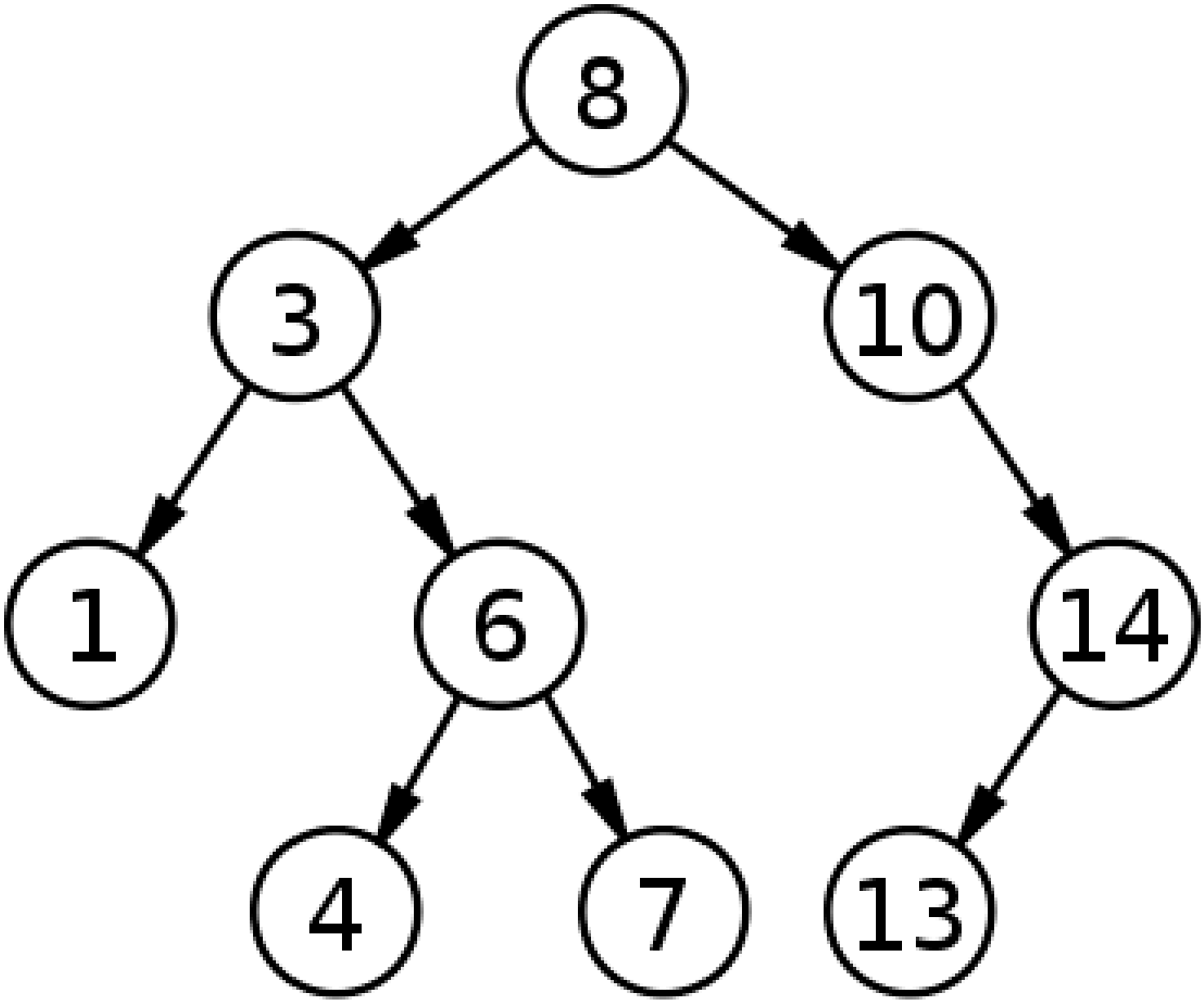
- 書きこみと同時に自動 dedup
- hot data tracking でよくアクセスするファイルを SSD にキャッシュ!
- fsck?

使いたくなってきたでしょ？

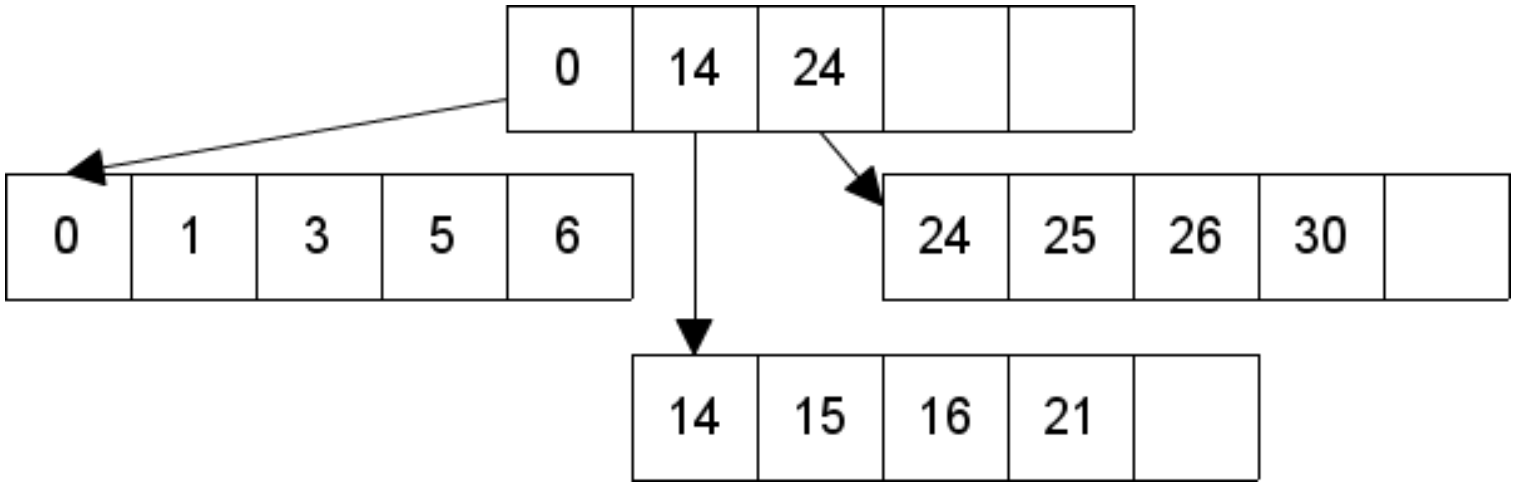
B 木

- Btrfs のほとんどいたるところで使われるデータ構造
- これがわからないと Btrfs がわからない
- なんといっても “BtrFS” = “B-Tree File System” ですからな!

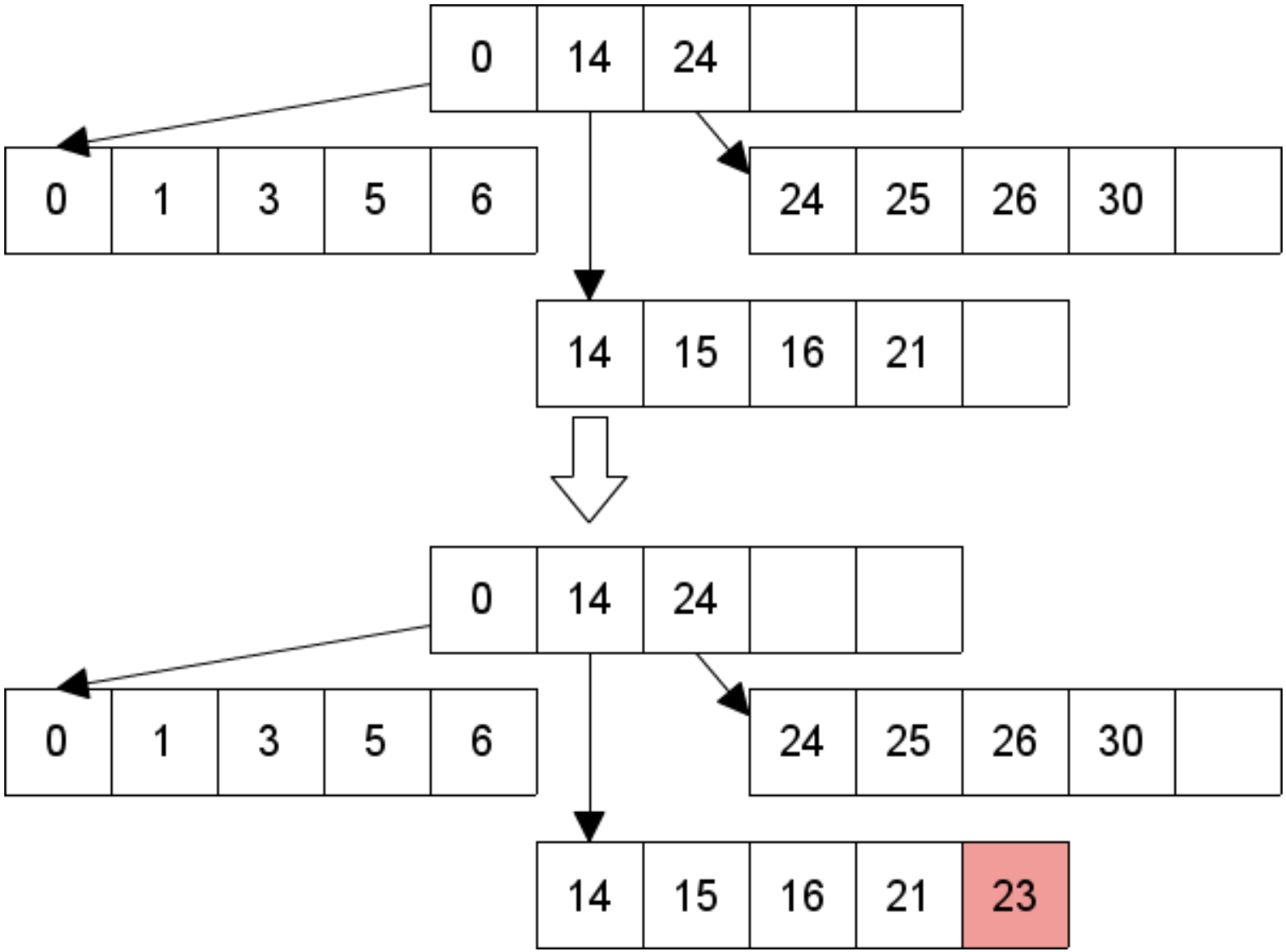
二分探索木



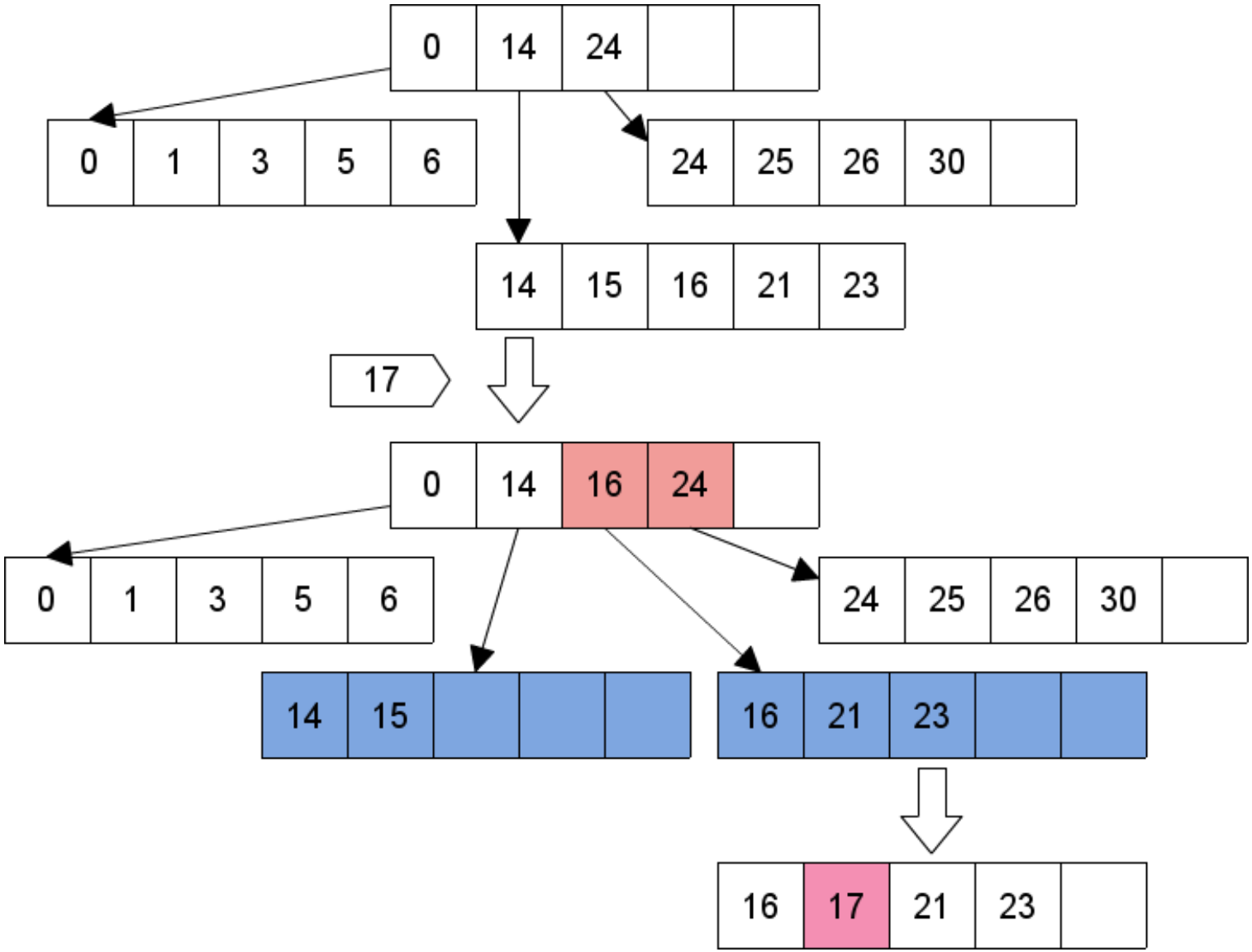
B木



B 木への挿入

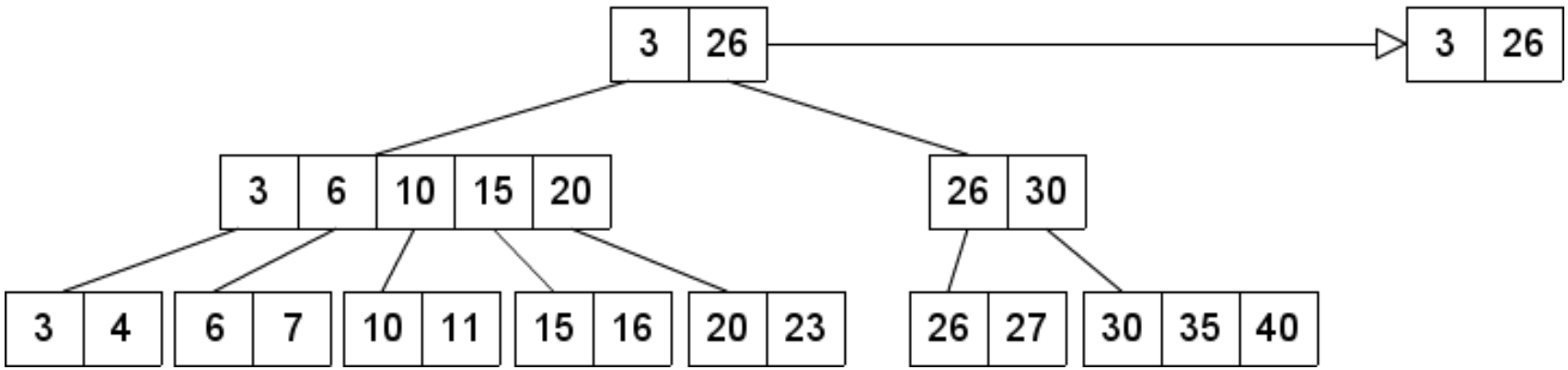

0
14
24
0
1
3
5
6
24
25
26
30
14
15
16
21
0
14
24
0
1
3
5
6
24
25
26
30
14
15
16
21
23

分割

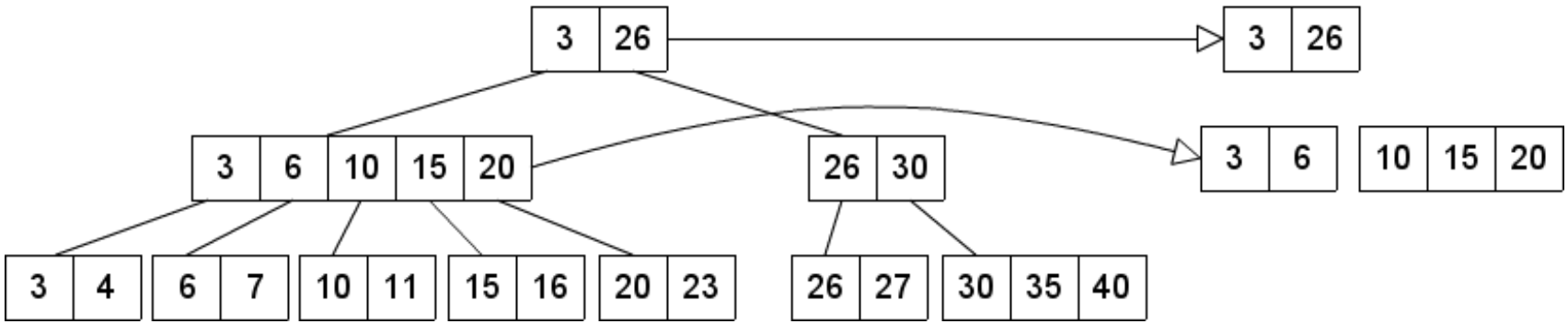

0
14
24
0
1
3
5
6
24
25
26
30
14
15
16
21
23
17
0
14
16
24
0
1
3
5
6
24
25
26
30
14
15
16
21
23
16
17
21
23

CoW 挿入

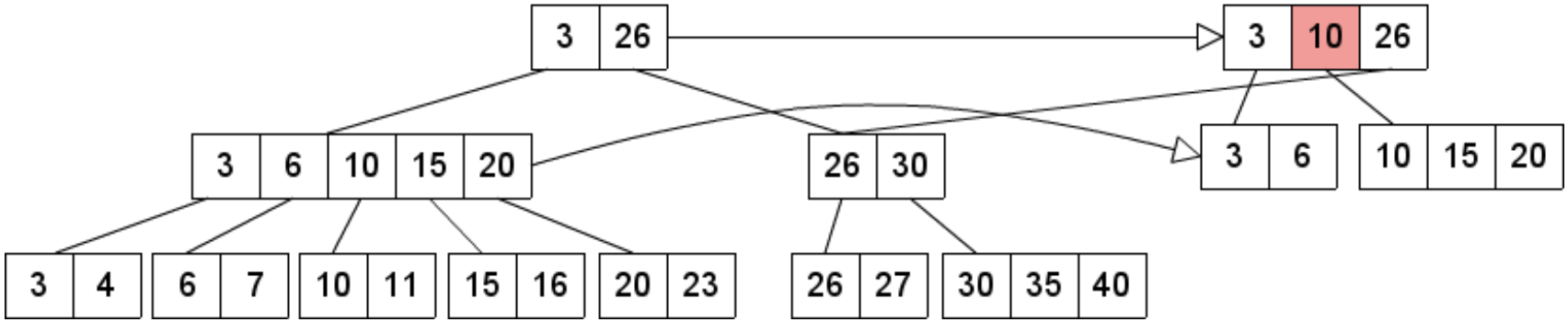
最大8ノードまで保持。「8」を追加



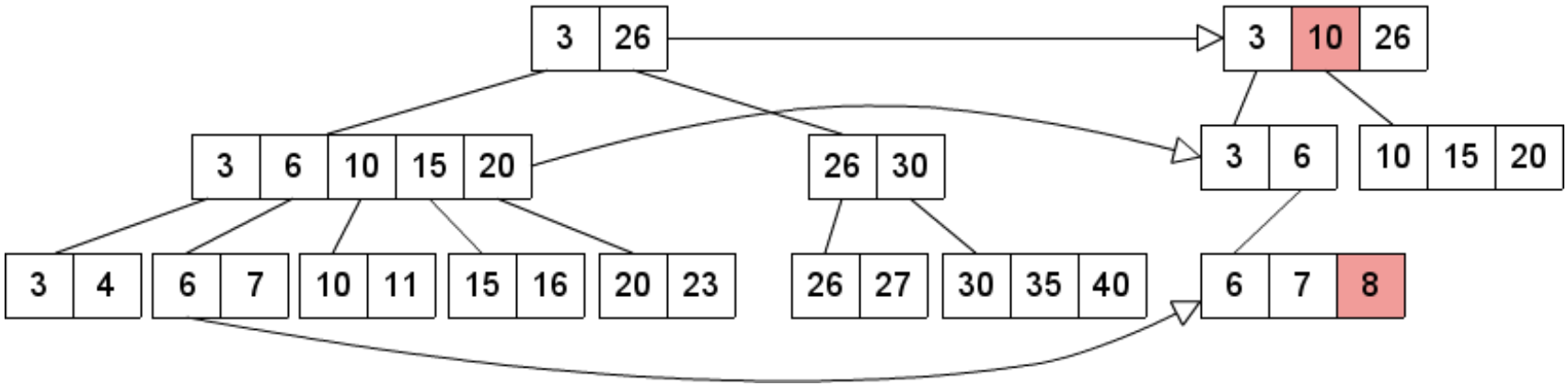
最大8ノードまで保持。「8」を追加。
(最大-3)以上を持つノードを分割



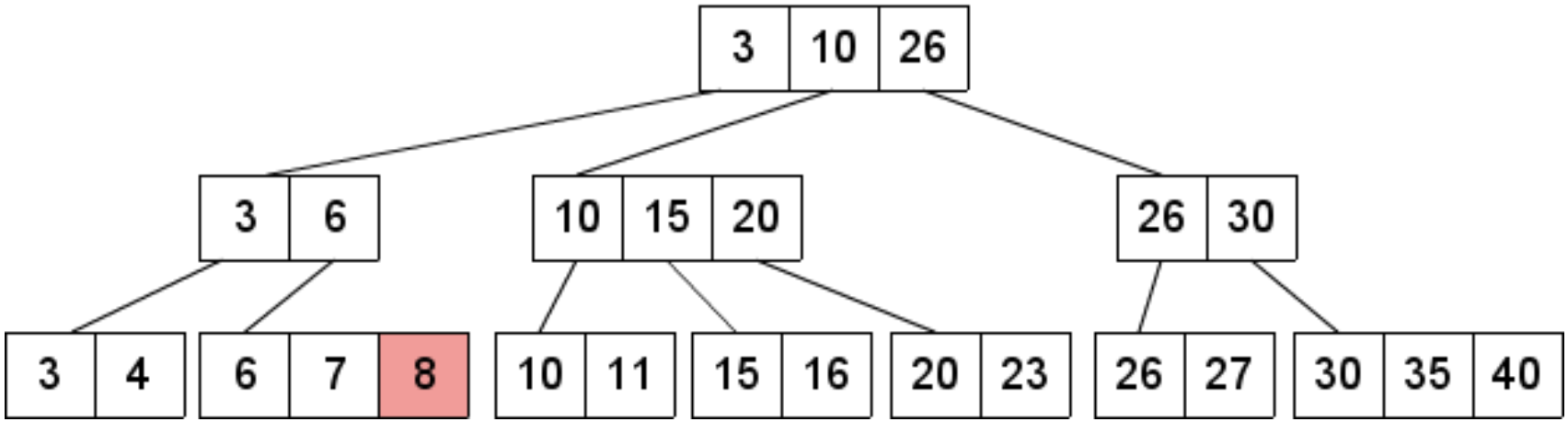
最大8ノードまで保持。「8」を追加。
(最大-3)以上を持つノードを分割



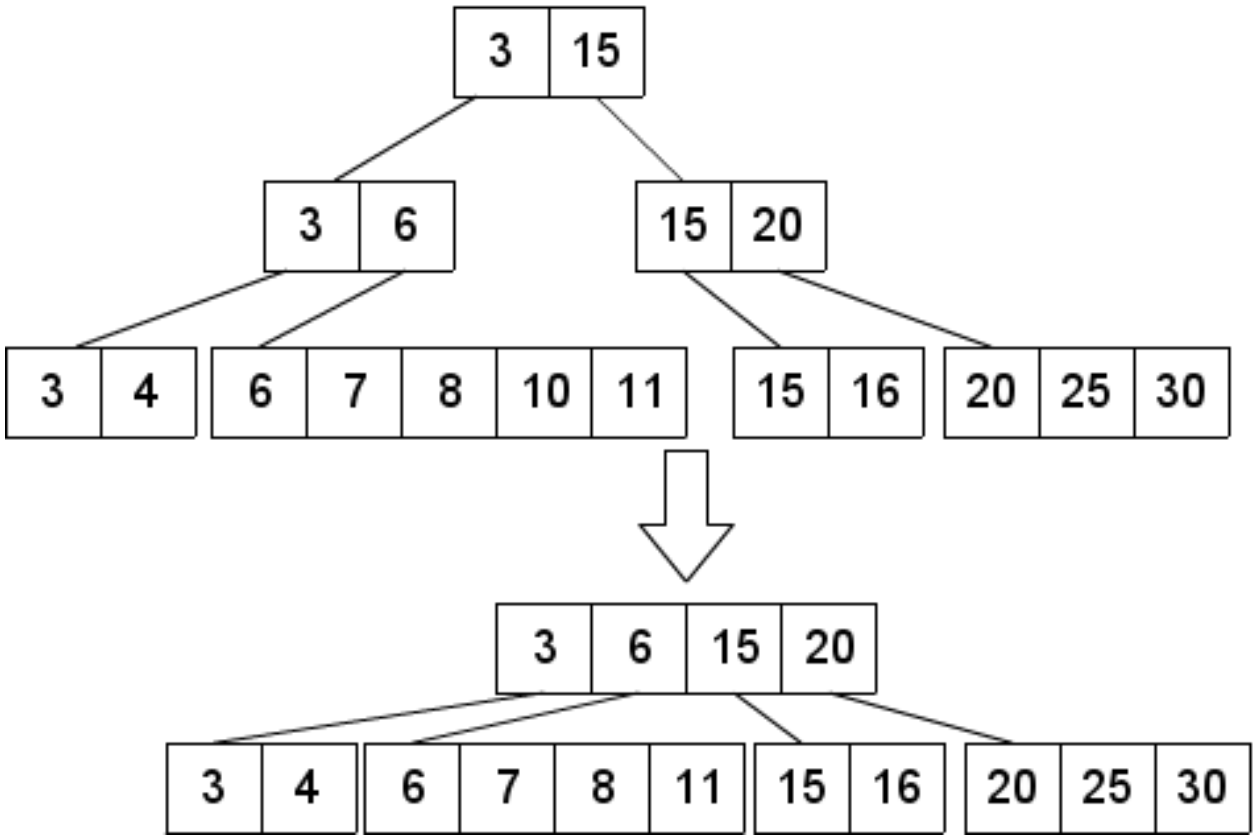
最大8ノードまで保持。「8」を追加。
(最大-3)以上を持つノードを分割

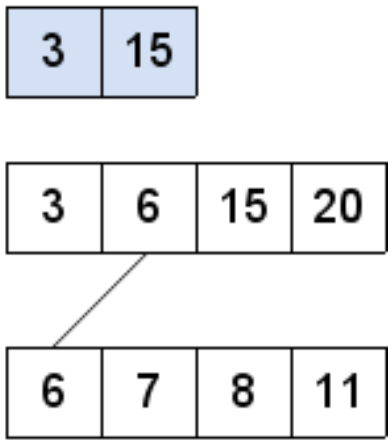


最大8ノードまで保持。「8」を追加。
(最大-3)以上を持つノードを分割



「10」を削除。

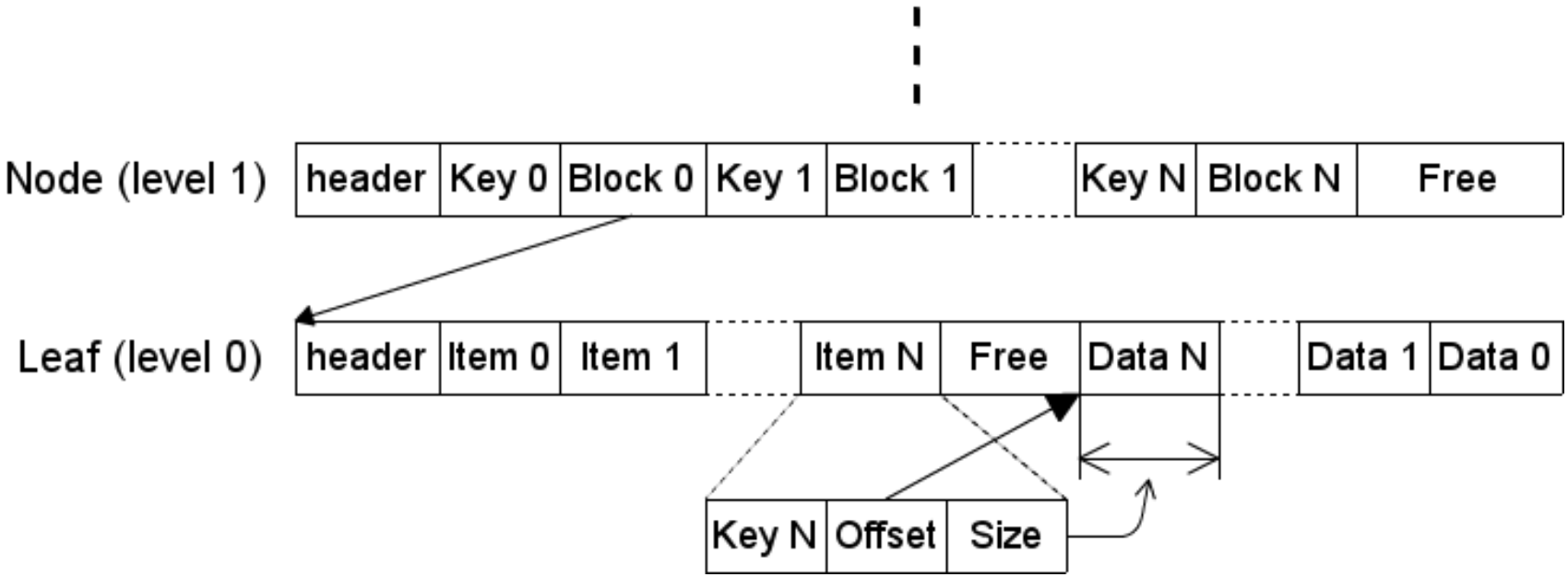

3
15
3
6
15
20
3
4
6
7
8
10
11
15
16
20
25
30
3
6
15
20
3
4
6
7
8
11
15
16
20
25
30


3
15
3
6
15
20
6
7
8
11


drwxr-xr-x 2 root r 48 2005- 16:59 bin
drwxr-xr-x 3 root 1 924 2005-11-27 16:42 boot
lrwxrwxrwx 1 root 11 2005-11-27 16:29 cdrom -> media/cdrom
drwxr-xr-x 11 root 576 2005-11-27 16:57 dev
drwxr-xr-x 52 ro 3072 2005-11-27 17:01 etc
drwxrwxr-x 4 roo 1024 2005-11-27 16:46 home
drwxr-xr-x 2 root root 1024 2005-11-27 16:32 initrd
lrwxrwxrwx 1 root root 28 2005-11-27 16:41 initrd.img -> boot/initrd.i
.4.27-2-386
drwxr-xr-x 8 root root 4096 2005-11-27 16:56 lib
drwxr-xr-x 2 root root 12288 2005-11-27 16:29 lost+found

BtrfsのB木

- Key
  - オブジェクト ID
  - タイプ
  - オフセット
- ノードとリーフ
  - 一番下がリーフ、残りがをノード
- リーフにキーに対応するデータが保管される


Node (level 1)
header
Key 0
Block 0
Key 1
Block 1
Key N
Block N
Free
Leaf (level 0)
header
Item 0
Item 1
Item N
Free
Data N
Data 1
Data 0
Key N
Offset
Size

様々な木

- Root tree
- FS tree
- extent tree
- chunk tree
- device tree
- CSum tree

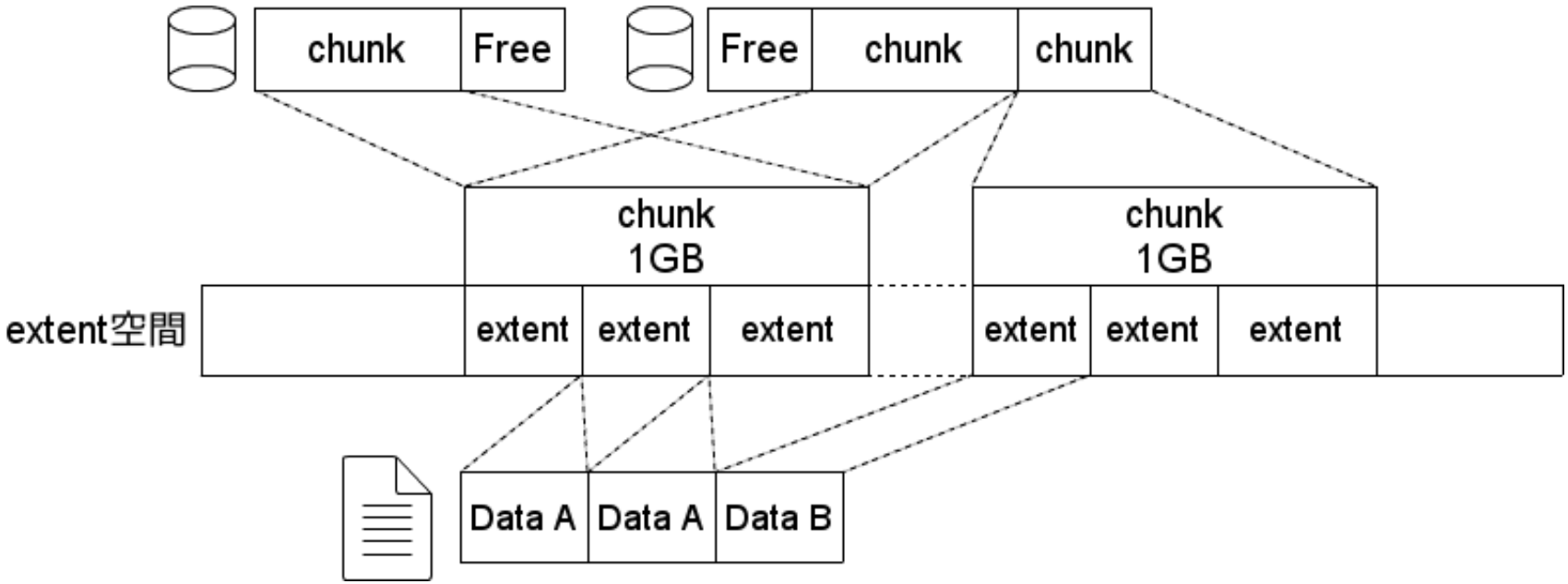
Root Tree

- 基本の木!
- 他の木の root を保持
- サブボリューム構造の取得

FS Tree

- サブボリューム・スナップショットごとに FS tree の root がある
- ディレクトリ構造
- i-node 情報
- ファイルデータ位置 (extent address)

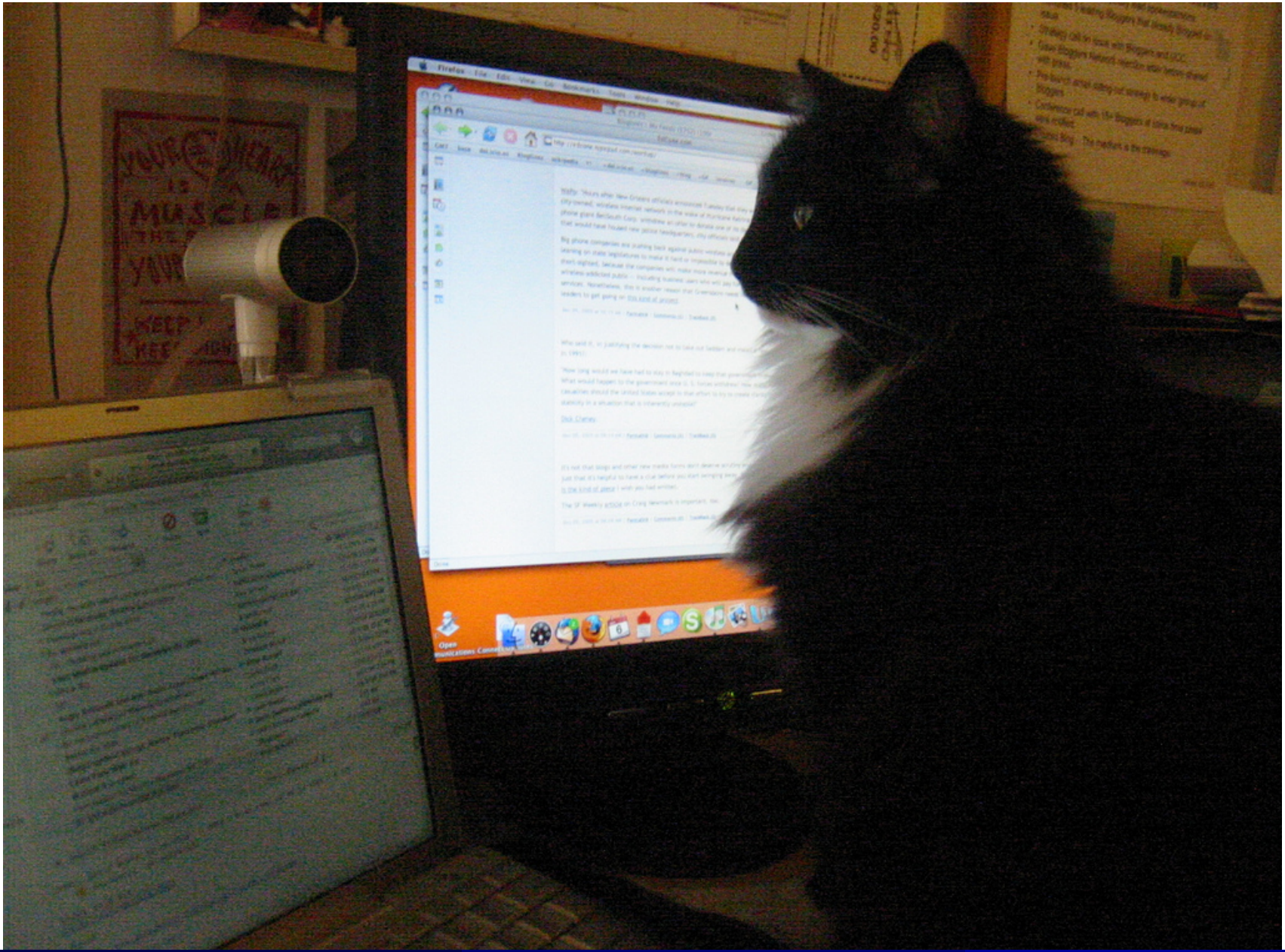
chunk と extent



device tree・CSum tree

- device tree
  - Btrfs に登録されているデバイスの管理
- CSum tree
  - 4KB ごとの checksum

探してみよう

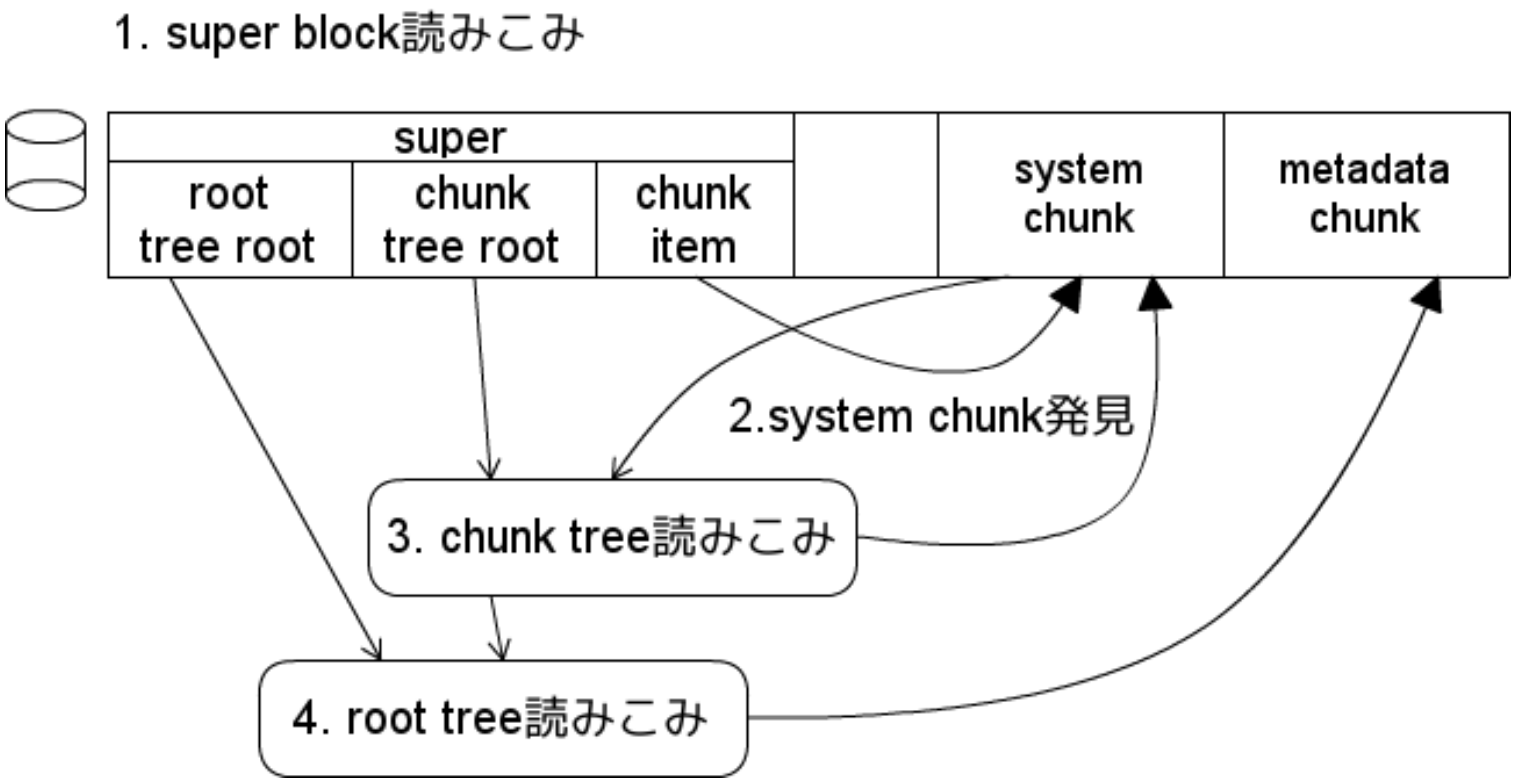


- Btrfs の tree もファイルも全て extent address(論理アドレス) でアクセス
- Root tree も extent address でアクセスされる
- chunk tree 自身も
- どうやって最初の extent address を物理アドレスにマッピングするの?

superblock

- btrfs で唯一物理的にアドレスが決まっている
  - パーティション 先頭から 6410KiB, 6410MiB, 25610GiB, 1PiB
- root の extent address
  - root tree
  - chunk tree
- system chunk の chunk 情報
  - chunk tree のノード, リーフは system chunk から割り当てられる

system chunk



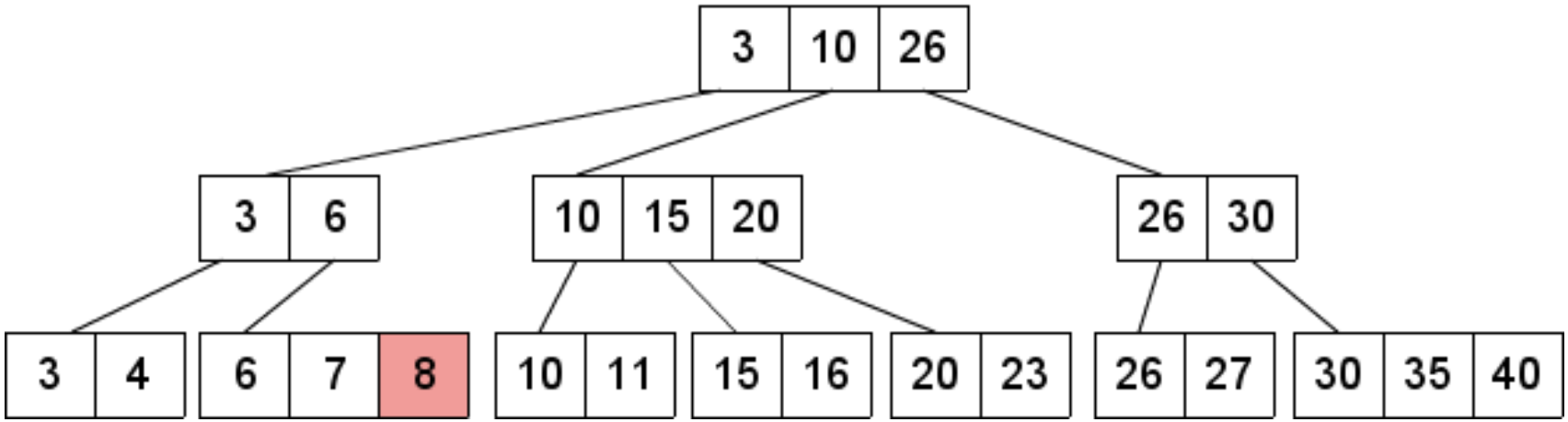
最後にこの１月ぐらいふみまくってる assert について話します

dump

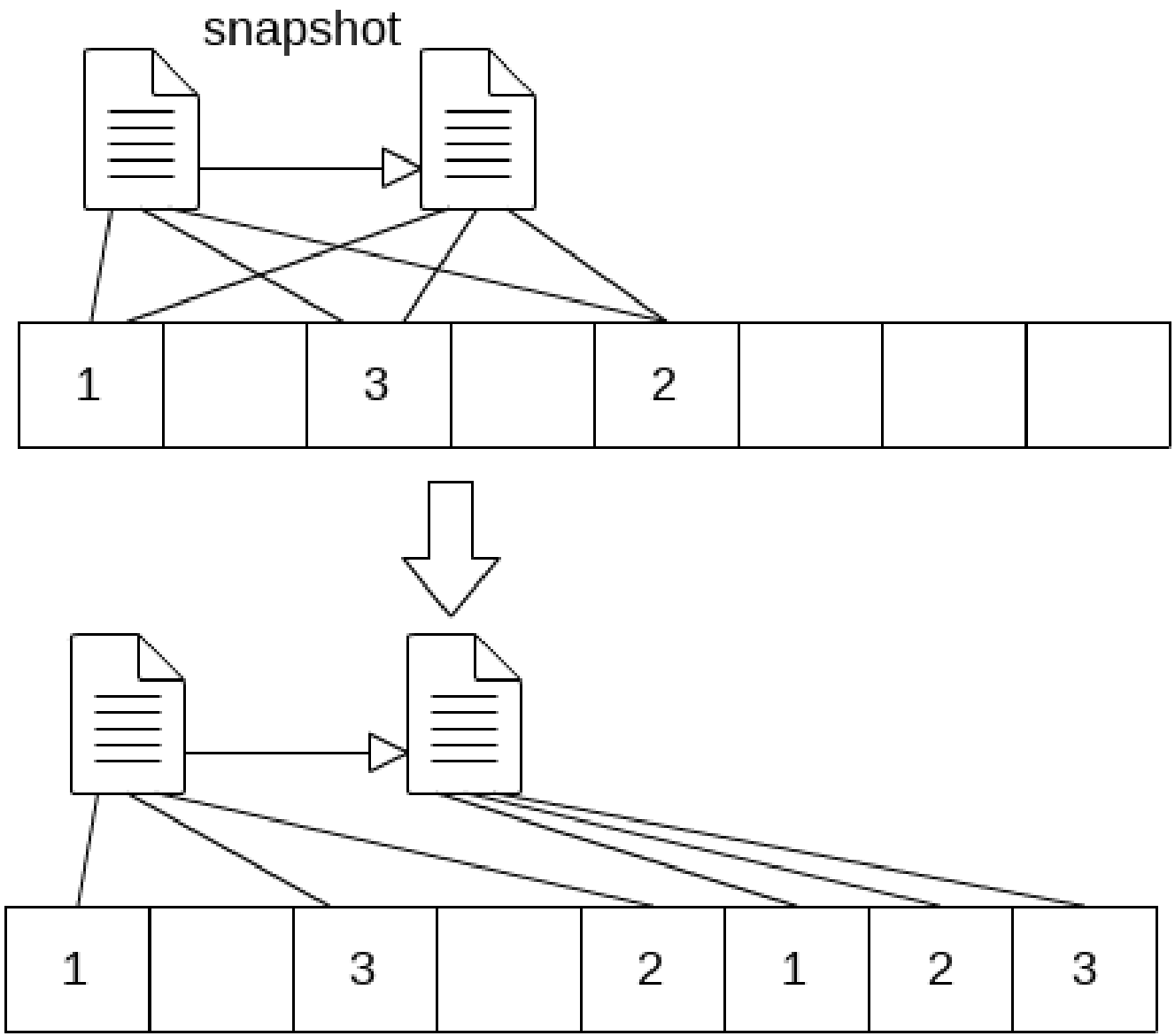
```
[42604.796633] BTRFS assertion failed: !memcmp_extent_buffer(
b, &disk_key, offsetof(struct btrfs_leaf, items[0].key),
 sizeof(disk_key)), file: fs/btrfs/ctree.c, line: 2444
```

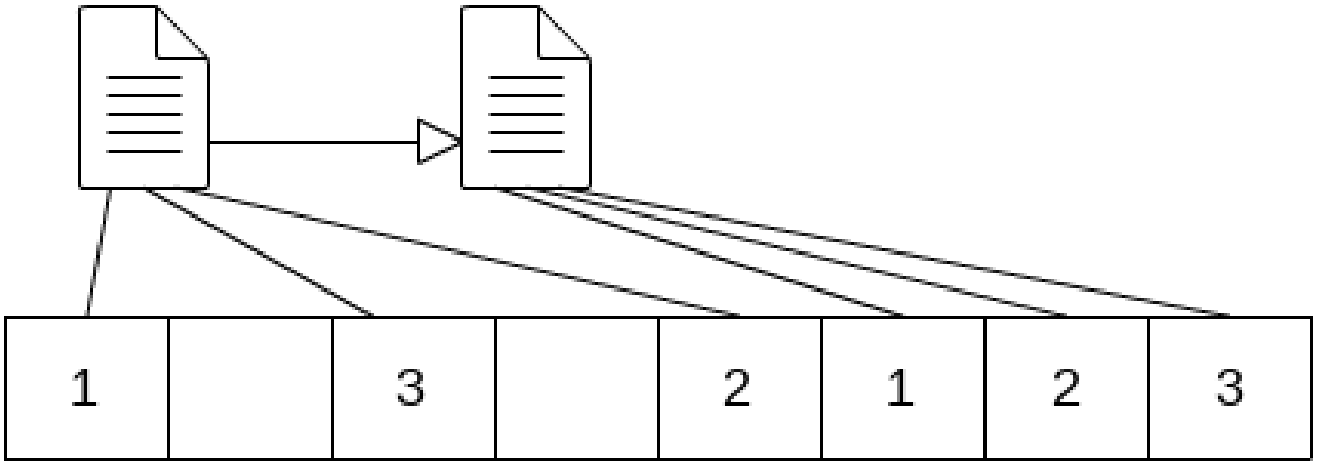
キー探索の最適化

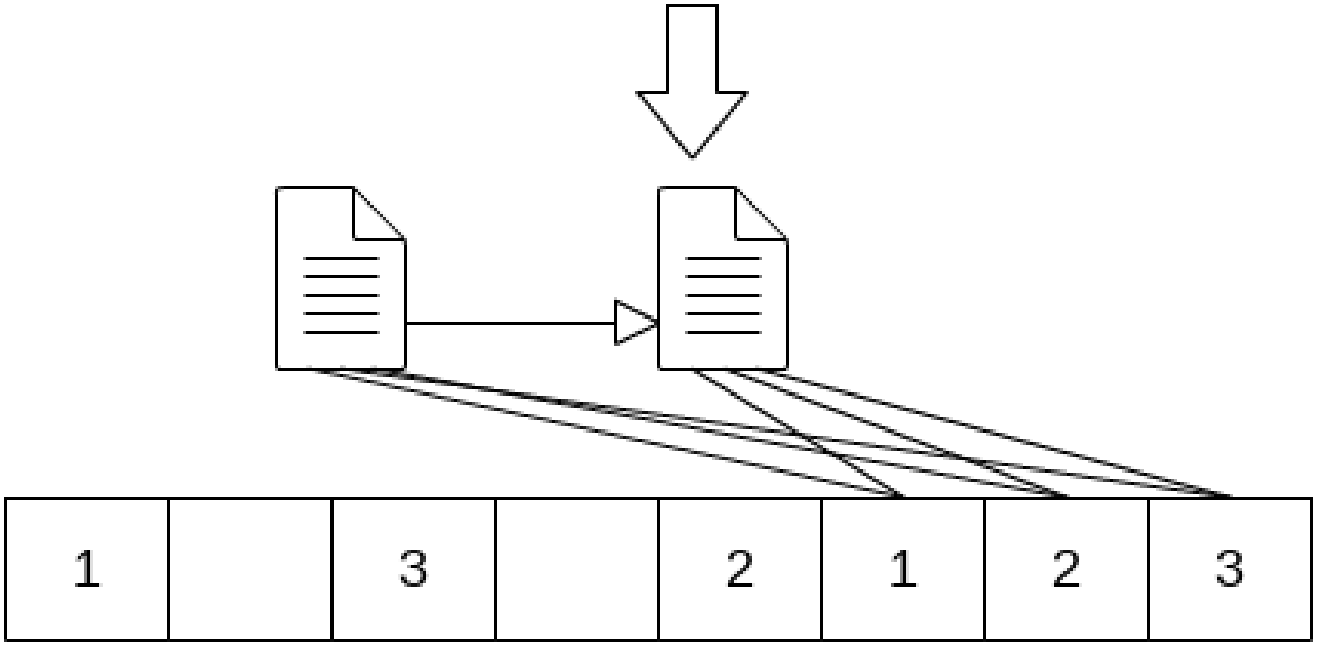
最大8ノードまで保持。「8」を追加。
(最大-3)以上を持つノードを分割



snapshot aware defrag




1
3
2
1
2
3


1
3
2
1
2
3

tree mod log

- B-tree の変更時に変更前にどういう値が入っていたかを記録
- transaction ごとにリセットされる
- メモリ上だけでディスクには書かれない
- これで「昔の B-tree」をとりだすがそれがバグってる?

原因は…なんでだろう?

まあとりあえず「自動デフラグ」切っておけば (デフォルト:オフ) いいと思うよ?

おしまい

参考文献

- https://btrfs.wiki.kernel.org/
- https://events.linuxfoundation.org/sites/events/files/slides/LinuxCon_2013_NA_Eckermann_Filesystems_btrfs.pdf
- http://people.redhat.com/lczerner/files/btrfs_lczerner.pdf
- https://www.usenix.org/legacy/event/lsf07/tech/rodeh.pdf

画像

- http://www.flickr.com/photos/artofphotography-ramsner/9592744354
- http://www.flickr.com/photos/surferbill/2506950772/
- http://www.flickr.com/photos/rore/1304728223/
- http://ja.wikipedia.org/wiki/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB:Binary_search_tree.svg
- http://www.flickr.com/photos/protohiro/85504626/
- http://www.flickr.com/photos/rubyji/74176893/
- http://www.flickr.com/photos/rore/1304728223/
- http://www.flickr.com/photos/dolfiedream/5060030894/