PMA LAYOUT

DISK

MEMORY *

arena base** – $2^{45} = 0x2000.0000.0000$ – above exe/heap

* VIRTUAL MEMORY

** top – $2^{45} + 2^{46} = 0x6000.0000.0000$
– below shared libs/stack

SINGLE TREE MAX SIZE

$$\frac{2^{46} \text{ (arena bytes)}}{2^{14} \text{ (page bytes)}} = 2^{32} \text{ pages}$$

$$\frac{2^{32} \text{ pages}}{2^{10} \text{ entries/leaf}} = 2^{22} \text{ leaves}$$

$$\sim 2^{23} \text{ nodes}$$

$$2^{23} \text{ nodes} \cdot 2^{14} \text{ bytes/node} = 2^{37} \text{ bytes} = 2^{7} \text{GB} = 128 \text{GB}$$

NODE LAYOUTS

ROOT NODE

4 bytes - version 4
4 bytes - checksum 8
8 x 4 bytes - block bases 16
disk offsets of spaces for nodes
1 byte - block base extent (0-8) 17
1 byte - tree depth 18 = 69 words
1 byte - dirty flag 19 out of 4K
1 byte - padding 20 2 bytes entry count
256 bytes - dirty bitmap 276 2 bytes padding
MFMFM ... FM remainder
Ø
MAX_OFFSET

M and F 32-bit (4 byte) words

NODE LAYOUTS

~~ROOT~~ NODE

~~4 bytes - version~~ ~~4~~

~~4 bytes - checksum~~ ~~disk offsets of~~ ~~8~~

~~8x 4 bytes - block bases~~ ← ~~spaces for nodes~~ ~~16~~

~~1 byte - block base extent~~ (0-8) ~~17~~

~~1 byte - tree depth~~ ~~18~~ = 6~~9~~4 words

~~1 byte - dirty flag~~ ~~19~~ out of 4K

~~1 byte - padding~~ ~~20~~

256 bytes - dirty bitmap ~~276~~ 256

MFMFM ...FM remainder

M and F 32-bit (4 byte) words

128GB	– b[7]	(128GB)
32GB	– b[6]	(160GB)
8GB	– b[5]	(168GB)
2GB	– b[4]	(170GB)
512MB	– b[3]	(170GB + 512MB)
128MB	– b[2]	(170GB + 640MB)
32MB	– b[1]	(170GB + 672MB)
8MB	– b[0]	(170GB + 680MB)
2-MB	– fixed	(170GB + 682MB)

$$\frac{\sim 171\ GB}{64\ TB - \sim 171\ GB} =$$

$$\frac{171 \cdot 2^{30}}{64 \cdot 2^{40} - 171 \cdot 2^{30}}$$

$$\approx \frac{171 \cdot 2^{30}}{63 \cdot 2^{40}}$$

$$\approx \frac{171}{63} \cdot \frac{1}{2^{10}}$$

$$\approx 4 \cdot \frac{1}{2^{10}}$$

$$\approx \frac{1}{2^8} \quad \frac{1}{256}$$

< 1% overhead

2^{22} leaves — 64 GB $\cdot 2$ = 128 GB (128 GB slot)

\+

2^{12} $d-1$ — 64 MB $\cdot 2$ = 128 MB (128 MB slot)

\+

2^{2} $d \cdot 2$ — 64 KB $\cdot 2$ = 128 KB (8 MB slot)

\+

1 100+ — 16 KB $\cdot 2$ = 32 KB (2 MB slot)

Enough room to fully dirty the tree

(why not sparse files?)

Sparse files are designed to be transparent to userspace: holes are created by seeking past unused areas, and are read as blocks of zeroes. They can't be detected using standard userspace APIs, at least not yet — as pointed out by Stéphane Chazelas, at least Solaris and Linux support the `SEEK_DATA` and `SEEK_HOLE` `lseek(2)` flags which allow userspace programs to find holes, and these flags might be added to POSIX at some point.

fallocate(2) used for sparse file hole punching

This is a nonportable, Linux-specific system call. For the portable, POSIX.1-specified method of ensuring that space is allocated for a file, see posix_fallocate(3).

Unclear interaction between fallocate(), mmap(), msync()

DELETION

Deletion without Rebalancing in Multiway Search Trees*

Siddhartha Sen[1] and Robert E. Tarjan[1,2]

[1] Princeton University
{sssix,ret}@cs.princeton.edu
[2] HP Laboratories, Palo Alto CA 94304

Abstract. Many database systems that use a B^+ tree as the underlying data structure do not do rebalancing on deletion. This means that a bad sequence of deletions can create a very unbalanced tree. Yet such databases perform well in practice. Avoidance of rebalancing on deletion has been justified empirically and by average-case analysis, but to our knowledge no worst-case analysis has been done. We do such an analysis. We show that the tree height remains logarithmic in the number of insertions, independent of the number of deletions. Furthermore the amortized time for an insertion or deletion, excluding the search time, is $O(1)$, and nodes are modified by insertions and deletions with a frequency that is exponentially small in their height. The latter results do not hold for standard B^+ trees. By adding periodic rebuilding of the tree, we obtain a data structure that is theoretically superior to standard B^+ trees in many ways. We conclude that rebalancing on deletion can be considered harmful.

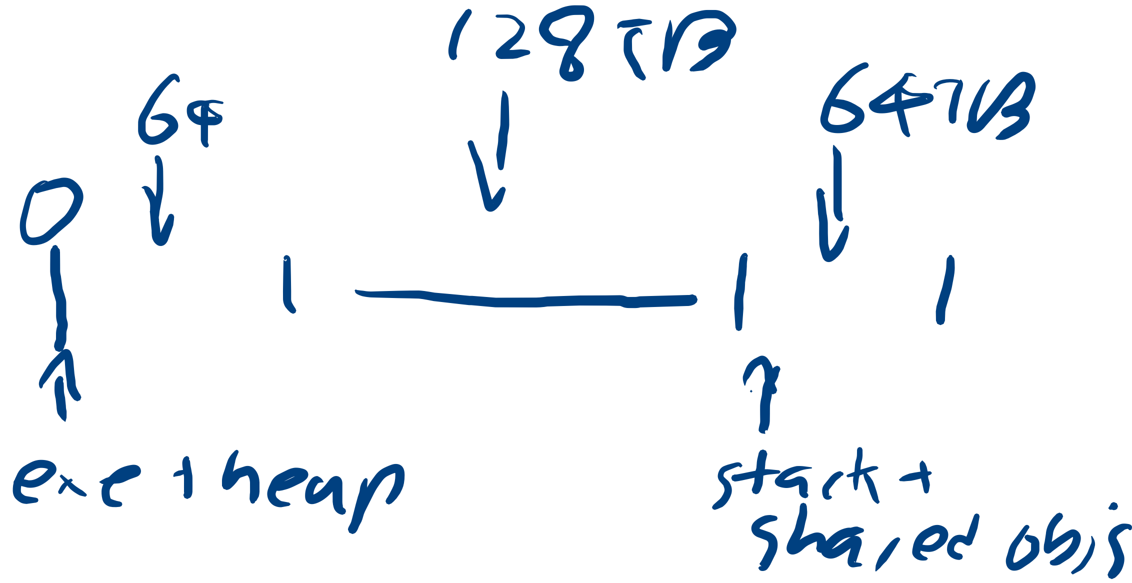
Just delete the entry; the leaf if now empty, recursively the nodes.

FREE LISTS

$$
\begin{aligned}
& \mathtt{0x6000.0000.0000} \\
- \; & \mathtt{0x2000.0000.0000} = 2^{45} \\
+ \; & \mathtt{0x4000.0000.0000} = 2^{46}
\end{aligned}
$$

- memory
- node disk
- pending node disk
- data disk
- pending data disk

in data memory



Unify pending just prior to snapshot:
allow no new dirtying from free list
until snapshot succeeds, process
crushes if snapshot fails

* what about online snapshotting?

1 - 2MB only
- init / find / insert / delete
(split)

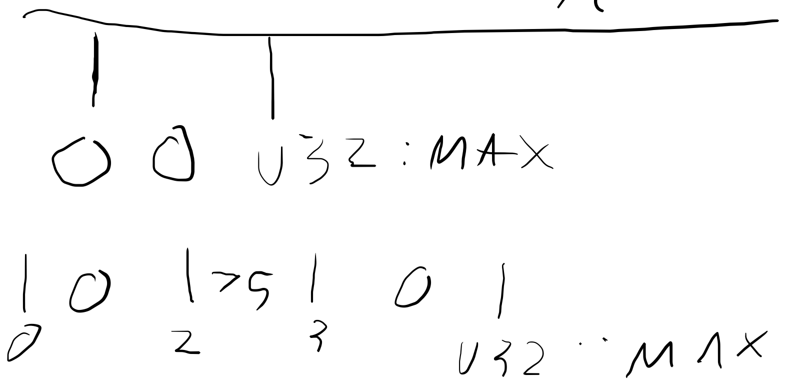
2 - memory management (fake disk offsets)
- data arena
- memory free lists
- malloc / free
- free can require
huge or allocation

3 - disk persistence
- disk free lists
- dirtying / COW
- msync / cleaning
- restoring /

4 - node partition striping

5 - range freeing

insert into empty range



insert into full range