Docker Storage Drivers

A Comparative Analysis

Burke Libbey

@burkelibbey

TL;DR: Use overlay.

(probably)

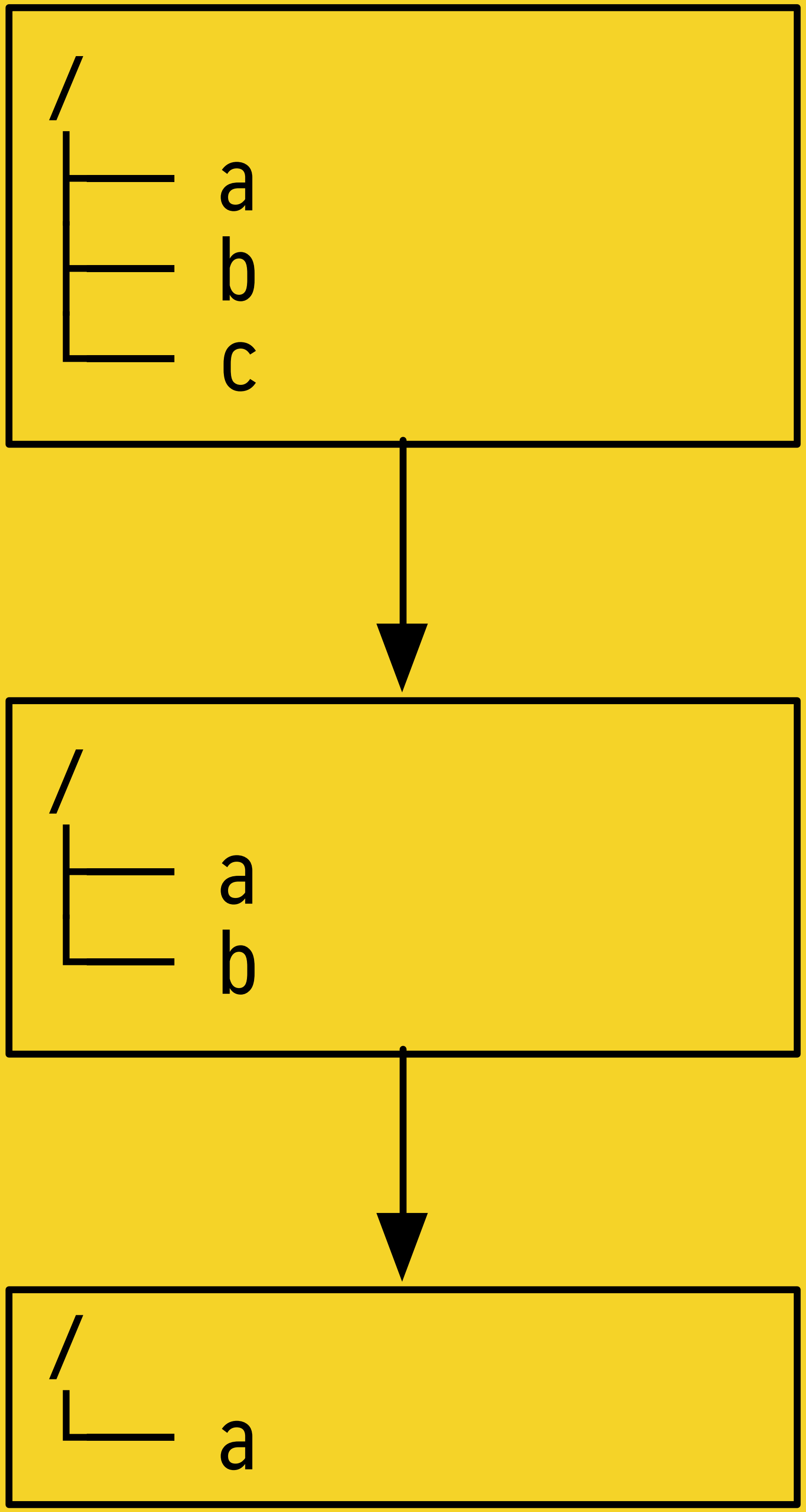
Overview

1. Storage Drivers in General
2. Implementation of each Storage Driver
3. Performance comparison
4. Recommendations

1

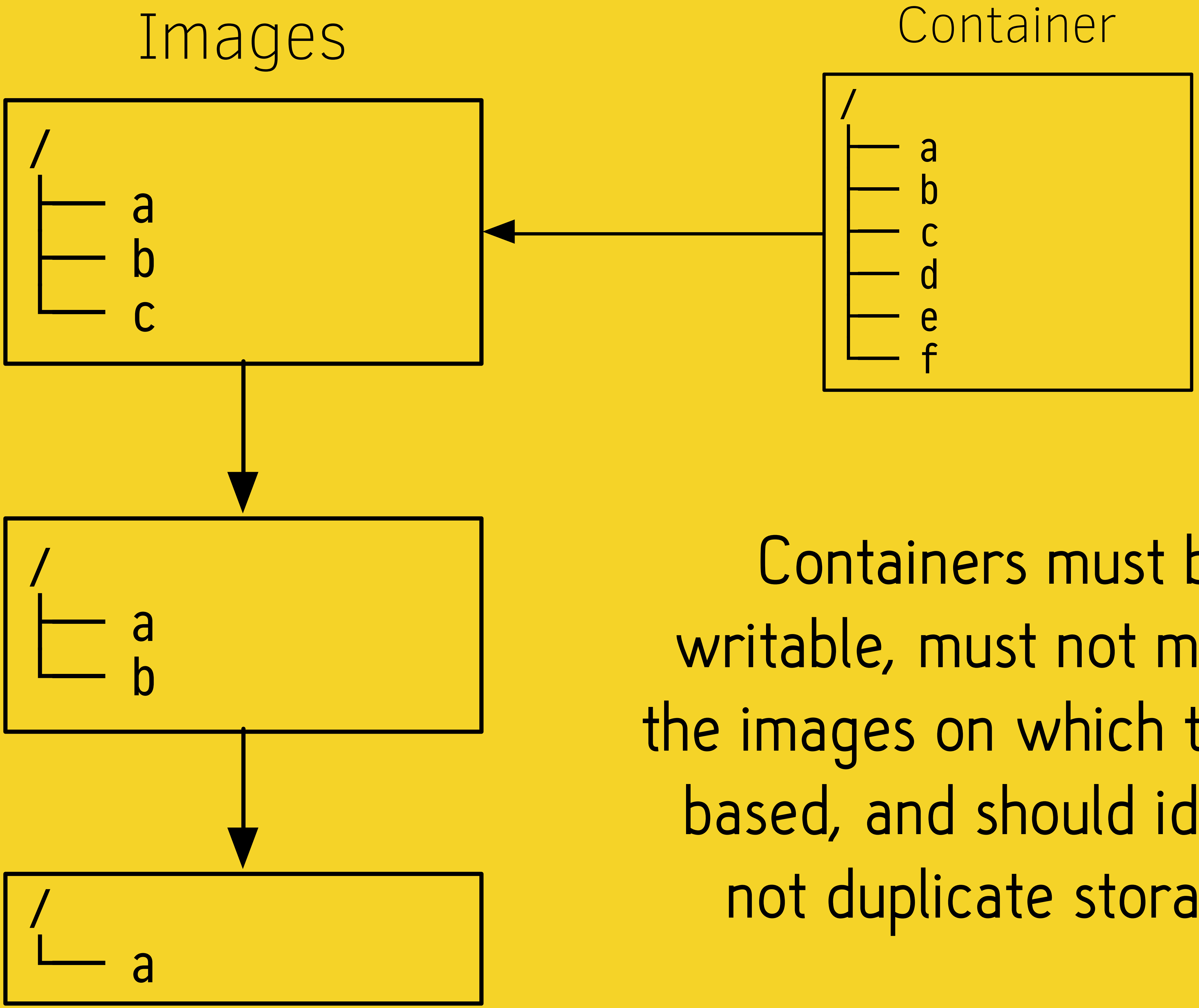
What is a **Storage Driver**?

Images



Images should share files
with their descendent images.

Storage Drivers are responsible
for this feature.



Containers must be writable, must not modify the images on which they`re based, and should ideally not duplicate storage.

Two primary actions*

1. Create descendent image
2. Create container from image

* at this conceptual level, anyhow

2

About Each Storage Driver

2.1

VFS

The Degenerate Case

Virtual **F**ile **S**ystem

Linux Kernel component and API

All other Filesystems conform to the very small VFS API.

Because of VFS, you don`t care the "ls /tmp" acts on a tmpfs.

Docker VFS

In context, "**vfs**" means
"doesn`t use any special features of specific filesystems"

which you can read as

"**really naïve** implementation"

VFS Driver

1. Create descendent image
   - Copy all the files

2. Create container from image
   - Copy all the files

VFS Driver

Dramatically **inefficient**, but obviously **correct**

VFS Driver

Mostly used to verify behaviour of other drivers.

VFS Driver

Please don`t actually use this one :)

2.2

AUFS

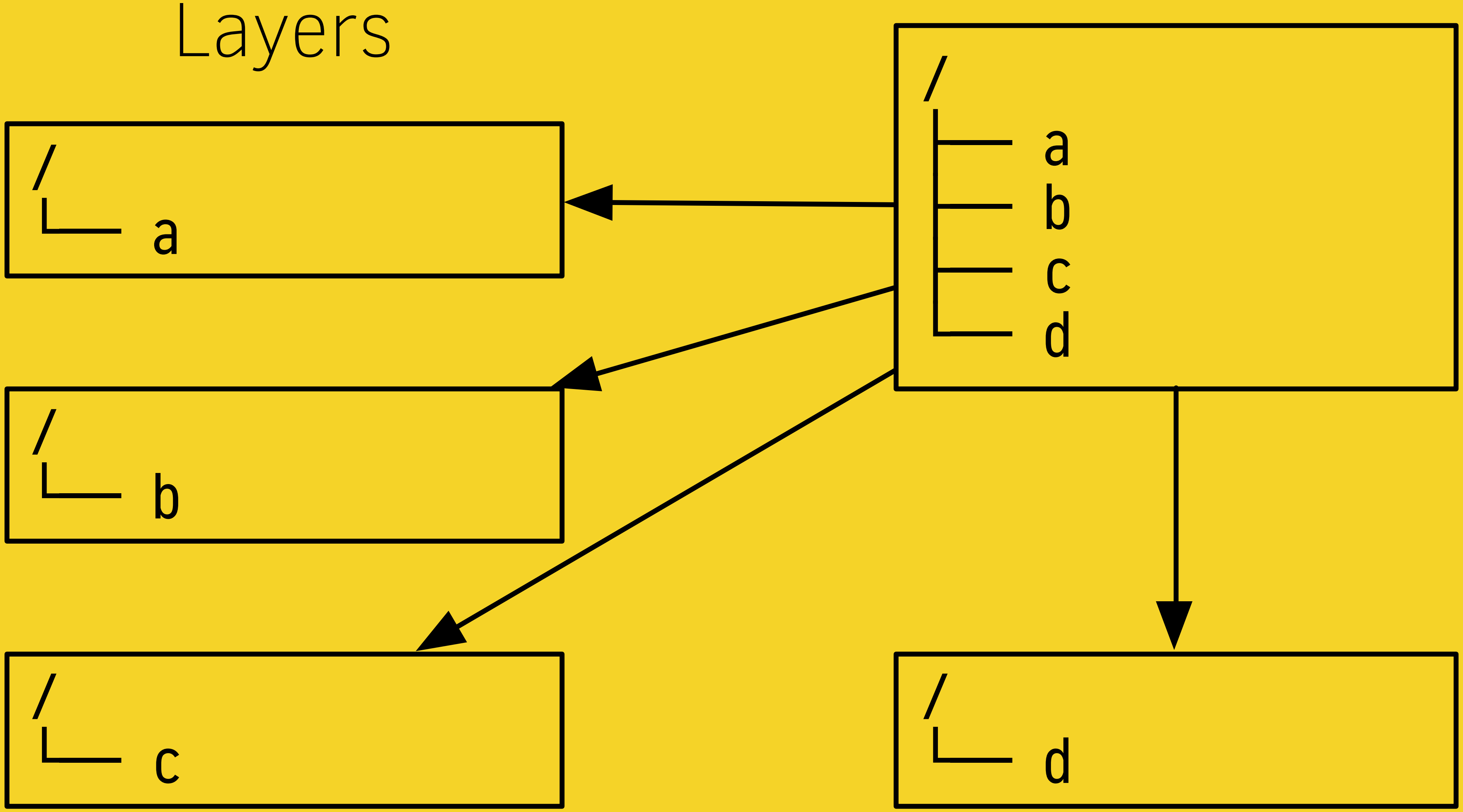
Almost worse than VFS*

*somehow.

AUFS

Individual Layers

AUFS Mount



Read/write layer

Initially empty

O(n) FS lookups
where n = number of layers

AUFS references:

1. a ″top‶ writable directory (″branch‶)

2. a variable number of other branches

A coalesced view is presented at the mount point. Changes are written to the top branch.

This means that AUFS has an O(n) file lookup cost

2.3

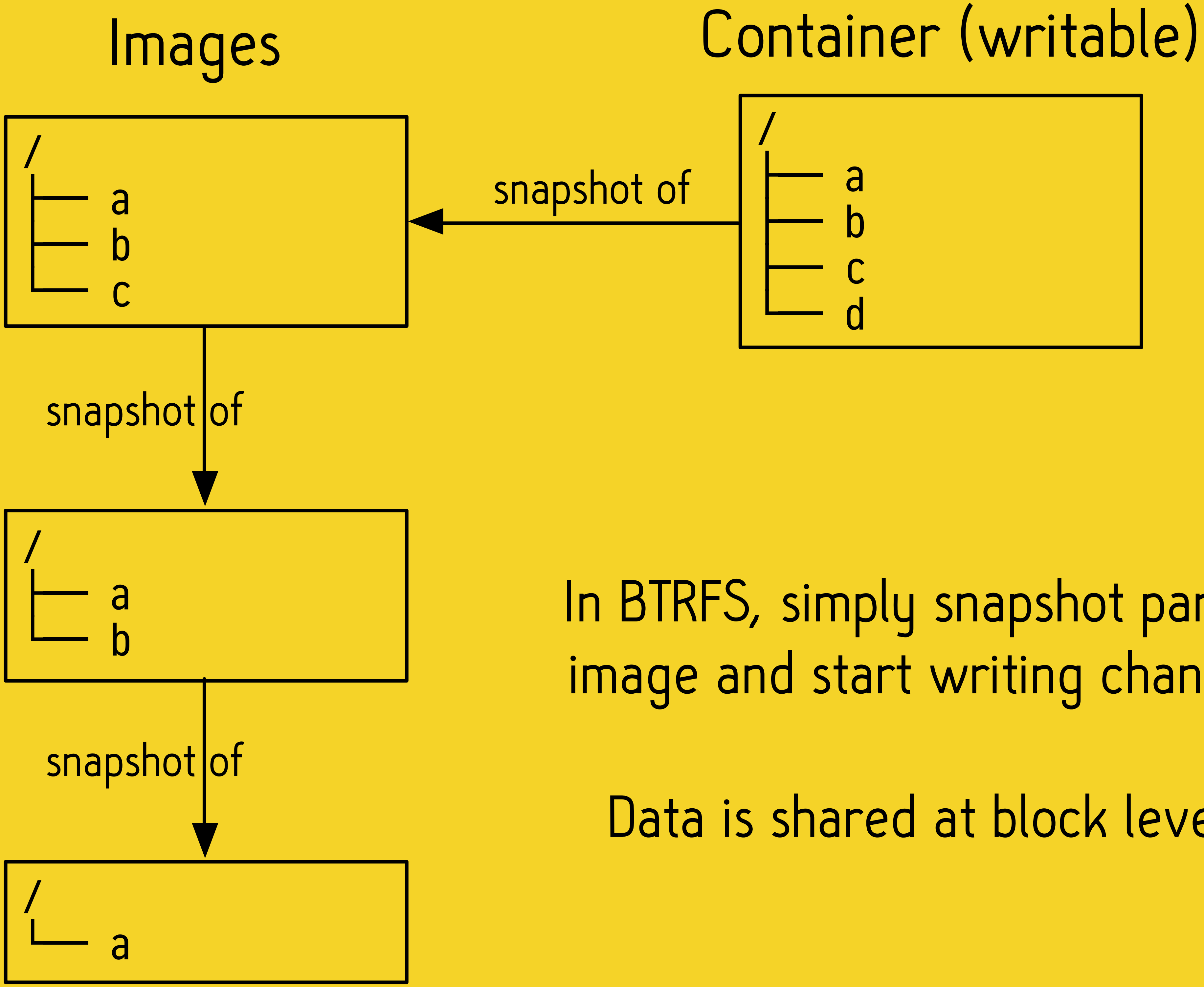
BTRFS

For carefully-selected definitions of “better“

BTRFS is more “advanced“ than AUFS*

*which isn`t necessarily a good thing

BTRFS uses filesystem snapshotting to implement content sharing.



In BTRFS, simply snapshot parent image and start writing changes

Data is shared at block level.

2.4

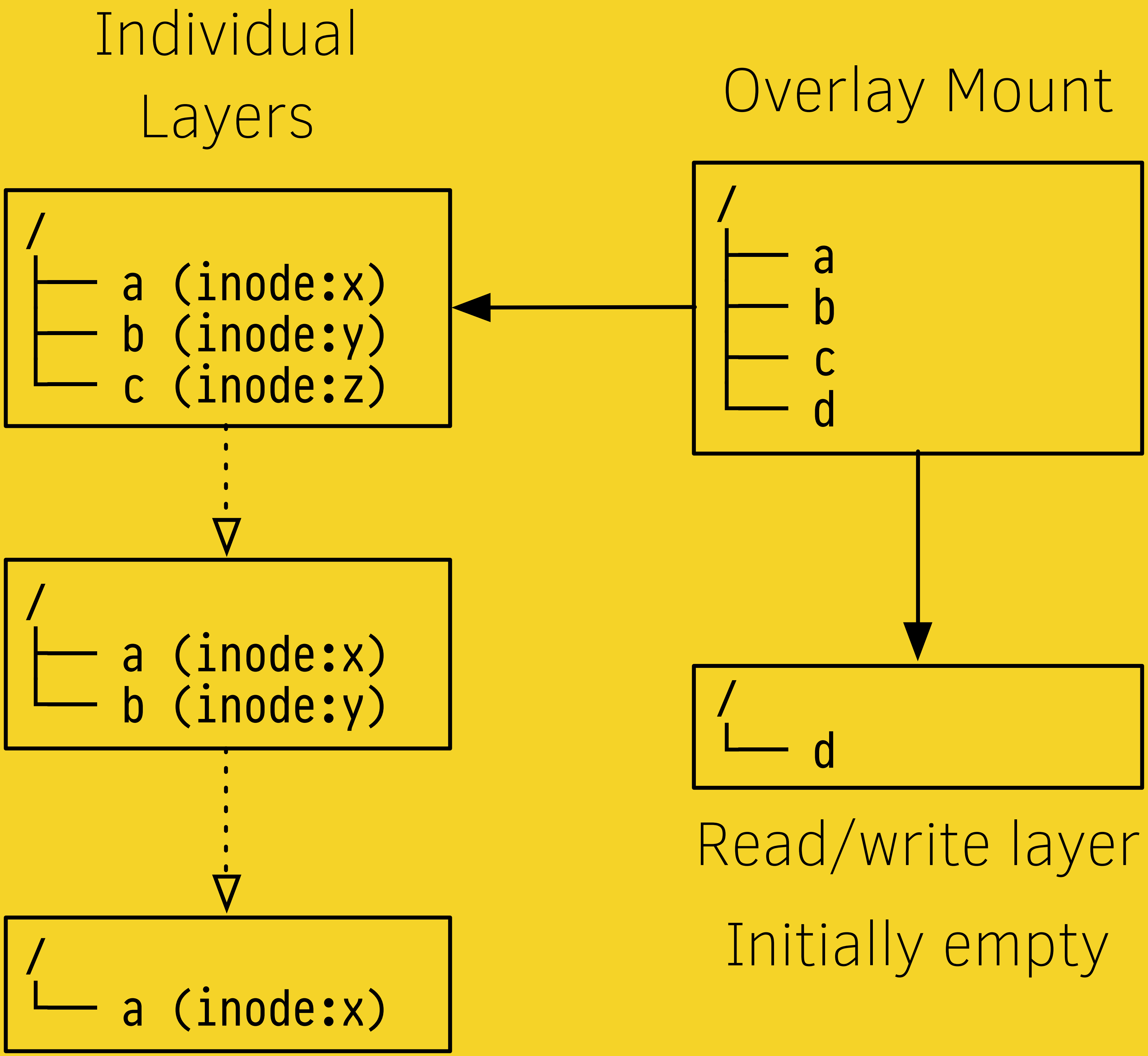
DeviceMapper

Probably stable, but slow

2.5

OverlayFS

The holy grail?



Individual Layers

```
/
├── a (inode:x)
├── b (inode:y)
└── c (inode:z)
```

```
/
├── a (inode:x)
└── b (inode:y)
```

```
/
└── a (inode:x)
```

Overlay Mount

```
/
├── a
├── b
├── c
└── d
```

```
/
└── d
```

Read/write layer

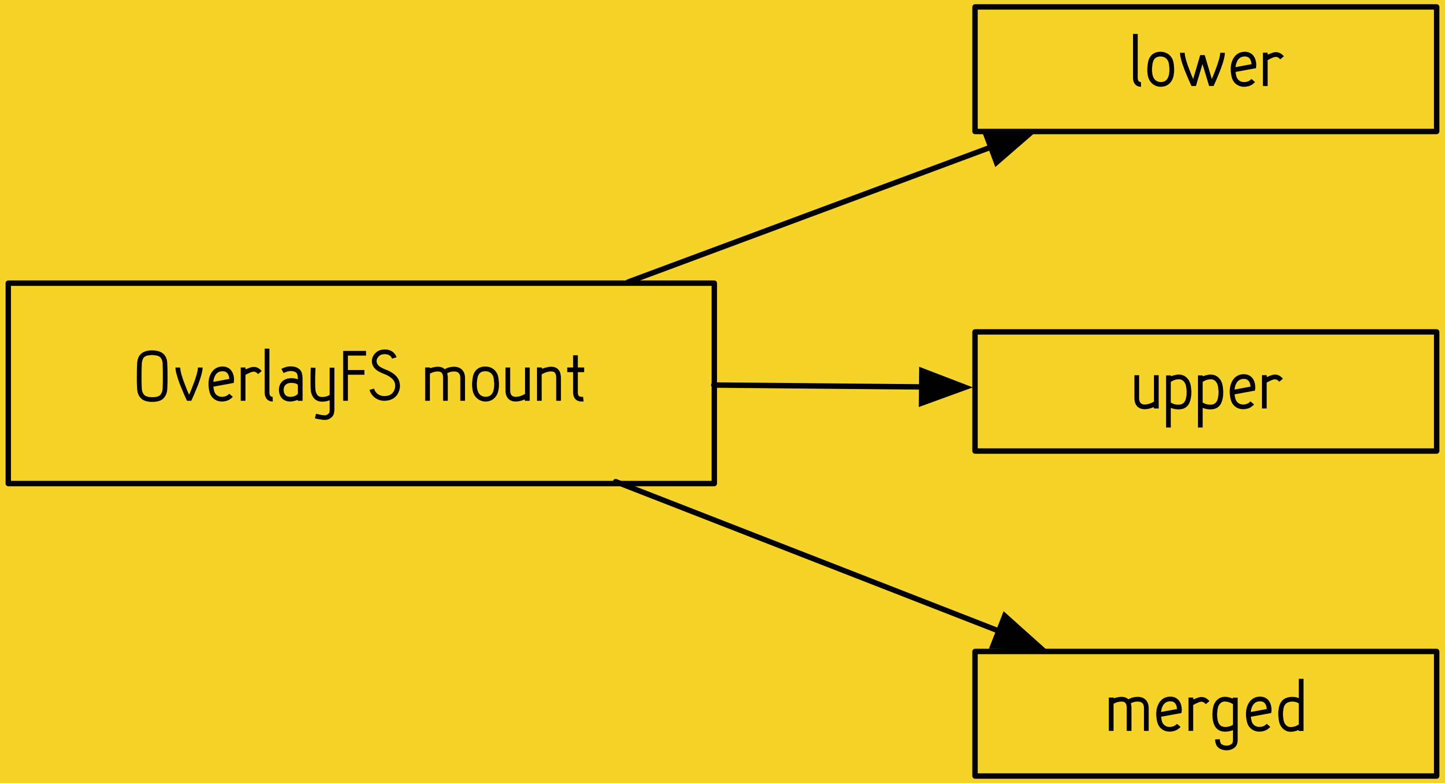
Initially empty

Unlike AUFS, O(1) FS lookups

Hardlinking is a good idea.

AUFS should be doing this too.

Anatomy of an OverlayFS mount



OverlayFS mount

lower

upper

merged

read-only "bottom" layer.
e.g. parent image

"top" layer.
i.e. divergence from lower

read-write coalesced view of
lower and upper

User interacts with "merged" directory.
Changes are written to "upper".
Unlike AUFS, does not immediately support coalescing n>2 directories.

OverlayFS sits on top of your actual FS — you can choose almost any underlying FS*

*but you should probably use ext4

Requires no additional configuration, but…

Requires kernel 3.18+

3

Performance

	FS Lookup operations (layers)	Data sharing	Inheritance Overhead (files or blocks)	Page cache sharing
AUFS	O(n)	File level	O(1)	Data & Metadata
BTRFS	O(1)	Block level	O(n)	Data
DeviceMapper	O(1) (?)	Block level (?)	(?)	(?)
VFS	O(1)	No	O(n)	No
OverlayFS	O(1)	File level	O(1)	Data & Metadata

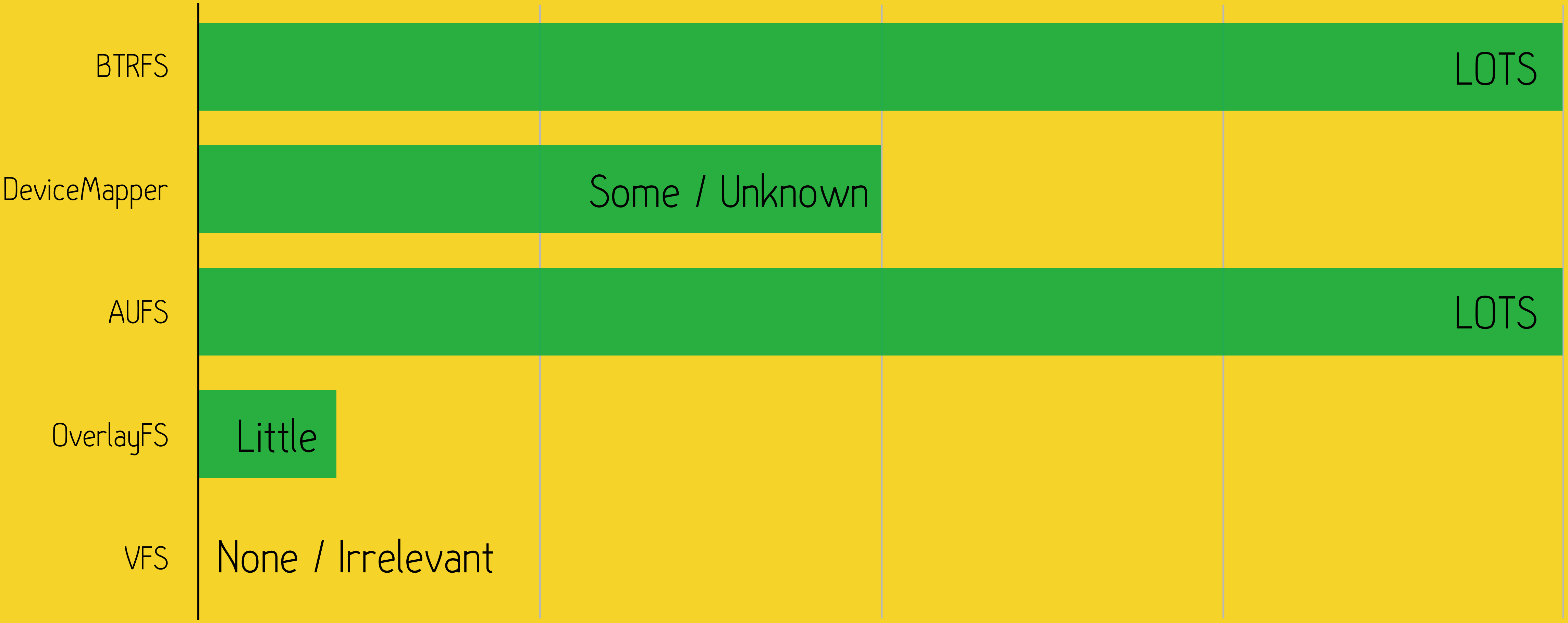
4

Recommendations

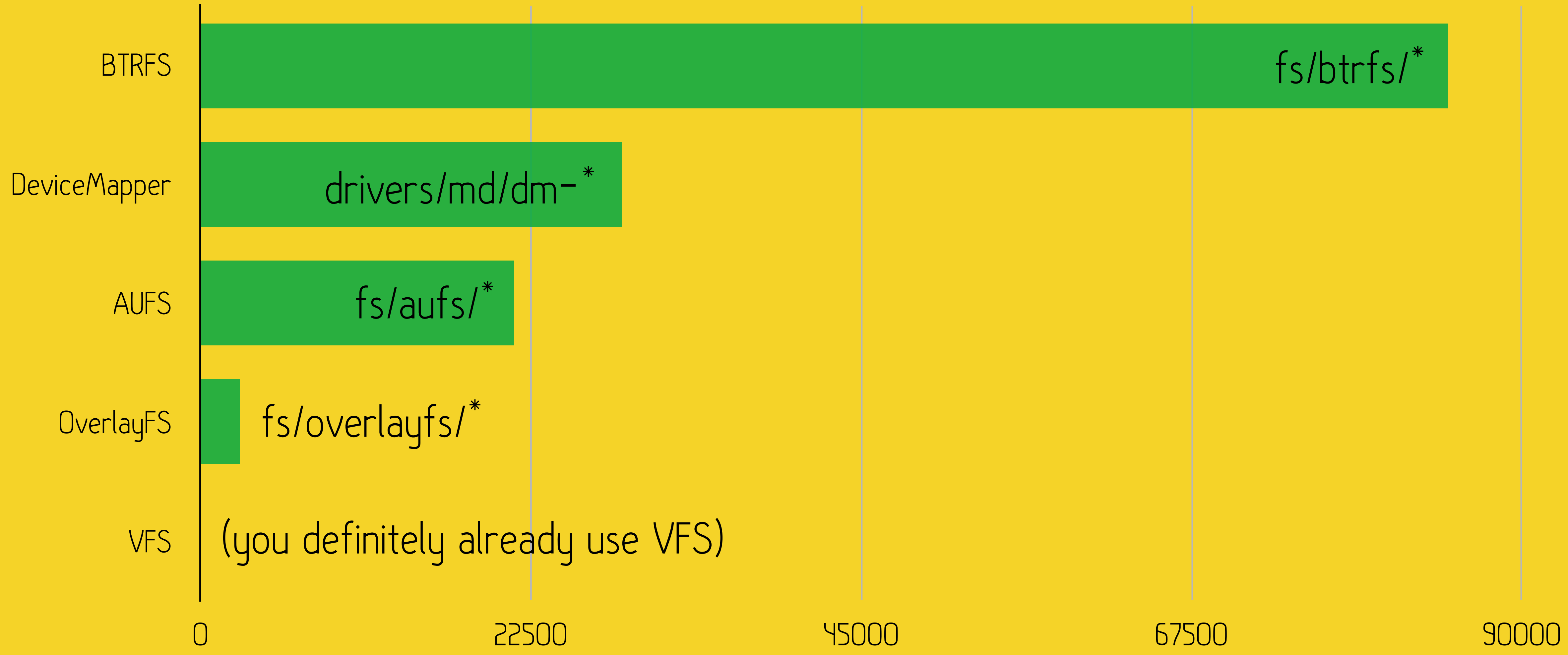
First, a note about complexity and stability

Anectodally:

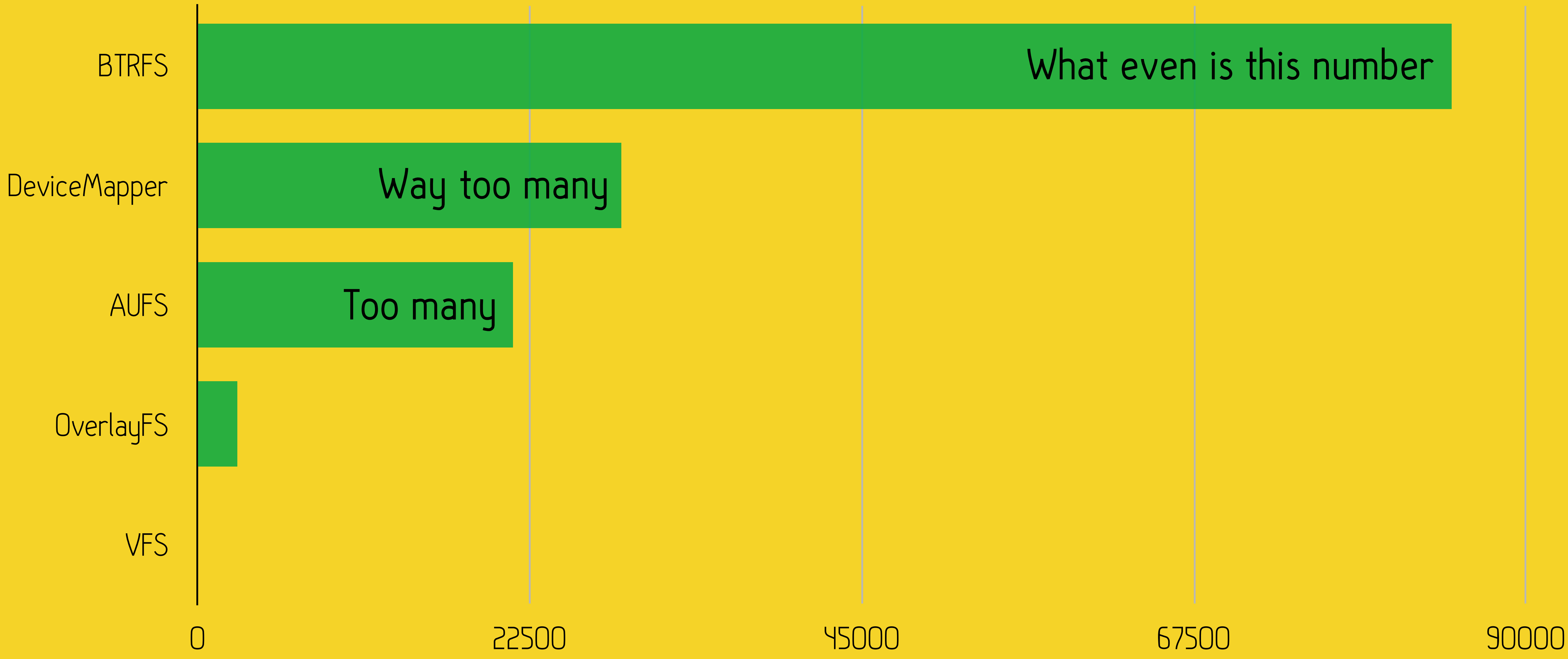
Shopify`s **mistrust** of drivers



New Source Lines of Code



New Source Lines of Code



We used AUFS in production from May to July 2014.

It wasn`t a positive experience. Performance was terrible and we frequently had to reboot deadlocked nodes.

We used BTRFS from July 2014 to February 2015.

It was an improvement, but still unstable. BTRFS is not mature. It frequently blocks on writes while starting containers, sometimes deadlocks.

We've been using OverlayFS since February

Huge improvement. Much more efficient and stable in every way.

Overall recommendations

If you can possibly run kernel 3.18+, use OverlayFS.

If not, try harder.

If still no, use BTRFS.