Forensic Analysis of Multiple Device BTRFS Configurations using The Sleuth Kit

Fraunhofer Institute for Communication, Information Processing and Ergonomics

Jan-Niclas Hilgert* jan-niclas.hilgert@fkie.fraunhofer.de

Martin Lambertz martin.lambertz@fkie.fraunhofer.de

Shujian Yang yang_shujian@hotmail.com

Forensic Analysis of Multiple Device BTRFS Configurations using The Sleuth Kit

- The Sleuth Kit for Pooled Storage File Systems
  - See our paper @ DFRWS USA 2017
- BTRFS Basics
  - Multiple Device Support
  - Documenting the address mapping used by BTRFS
- Implementing BTRFS into TSK
  - Forensic Analysis of BTRFS
  - Snapshots, File Recovery, Missing Storage Devices

https://github.com/fkie-cad/sleuthkit

Fraunhofer
FKIE

The Sleuth Kit

- Open-source forensic toolkit for volume and file system analysis
  - **`mmls:`** Display the partition layout of a volume system (partition tables)
  - **`fsstat:`** Display the details associated with a file system
  - **`fls:`** List file and directory names in a disk image
  - **`istat:`** Display details of a meta-data structure (i.e. inode)
  - **`icat:`** Output the contents of a file based on its inode number

Fraunhofer FKIE

The Sleuth Kit

- Open-source forensic toolkit for volume and file system analysis
- No file system specific background knowledge required

The Sleuth Kit

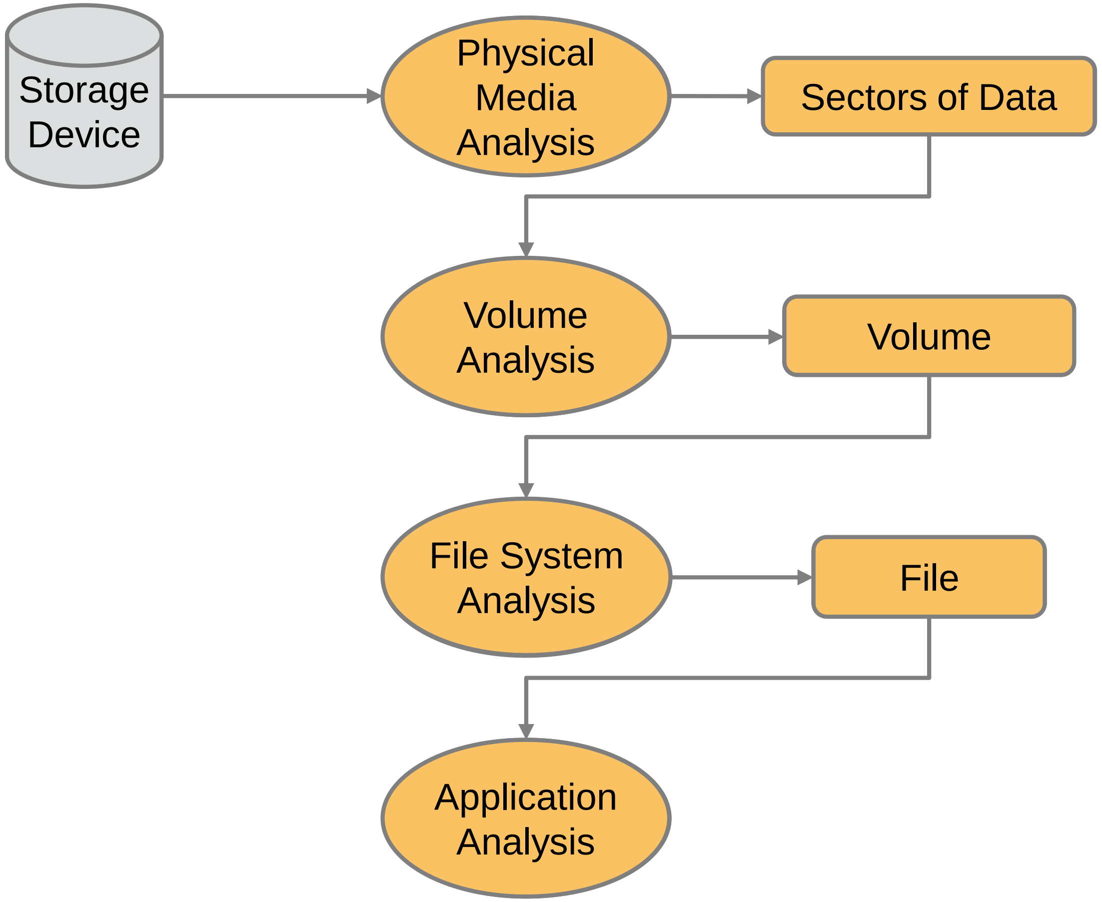
- Open-source forensic toolkit for volume and file system analysis
- No file system specific background knowledge required
- Support for multiple contemporary file systems

Input Data

- Analyzes raw (i.e. **dd**), Expert Witness (i.e. EnCase) and AFF file system and disk images. (Sleuth Kit Informer #11)
- Supports the NTFS, FAT, ExFAT, UFS 1, UFS 2, EXT2FS, EXT3FS, Ext4, HFS, ISO 9660, and YAFFS2 file systems (even when the host operating system does not or has a different endian ordering).
- Tools can be run on a live Windows or UNIX system during Incident Response. These tools will show files that have been "hidden" by rootkits and will not modify the A-Time of files that are viewed. (Sleuth Kit Informer #13)

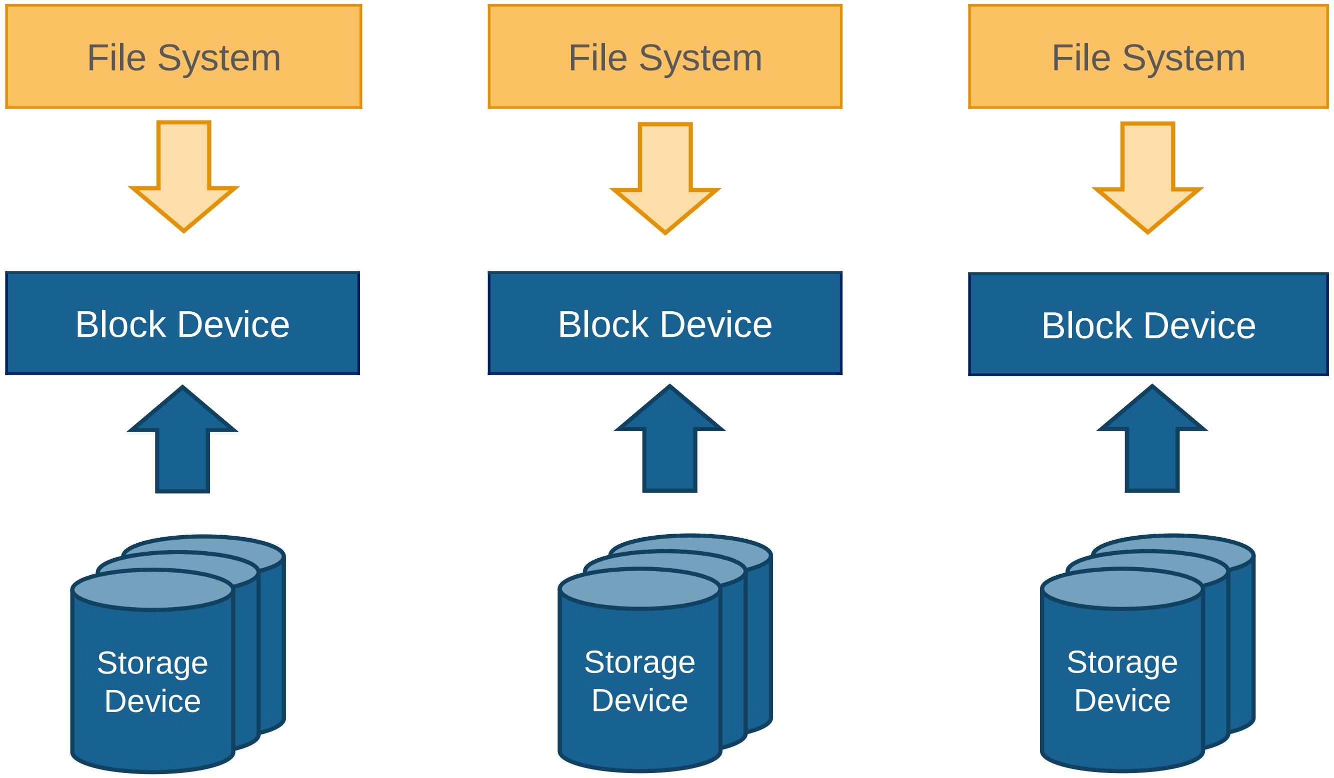
The Sleuth Kit – Theoretical Model

1. Data is aquired during the physical media analysis as a sequence of bytes
2. Volumes like partitions and multiple disk configurations are detected in the volume analysis
3. File system analysis searches the volumes for a file system on top of it
4. Application analysis is used for the analysis of files after their extraction or recovery



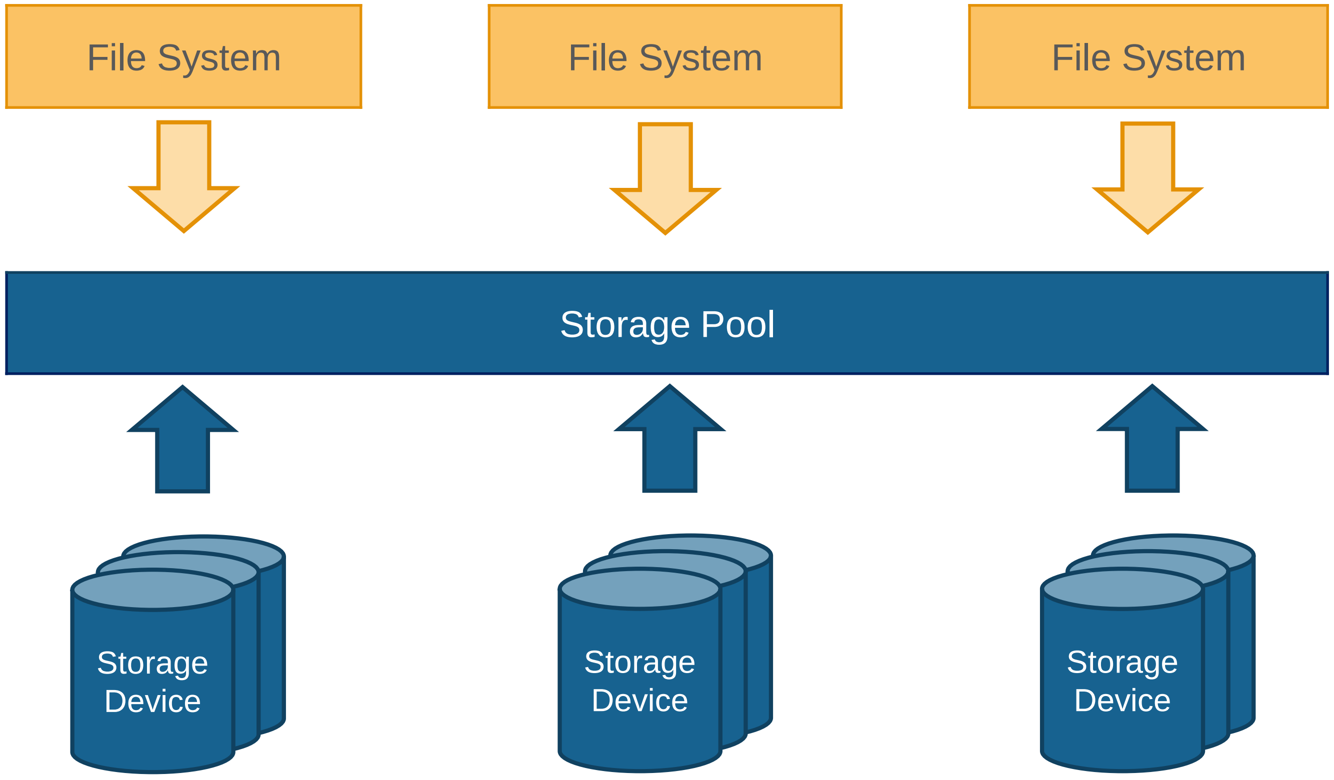
Pooled Storage File Systems

- Old file system mapping
  - Storage devices are somehow combined to block devices
  - One file system is assigned to exactly one block device



Pooled Storage File Systems

- Old file system mapping
  - Storage devices are somehow combined to block devices
  - One file system is assigned to exactly one block device
- Pooled storage file systems
  - Storage devices (or block devices) are combined to a storage pool
  - File systems share the available space of the storage pool

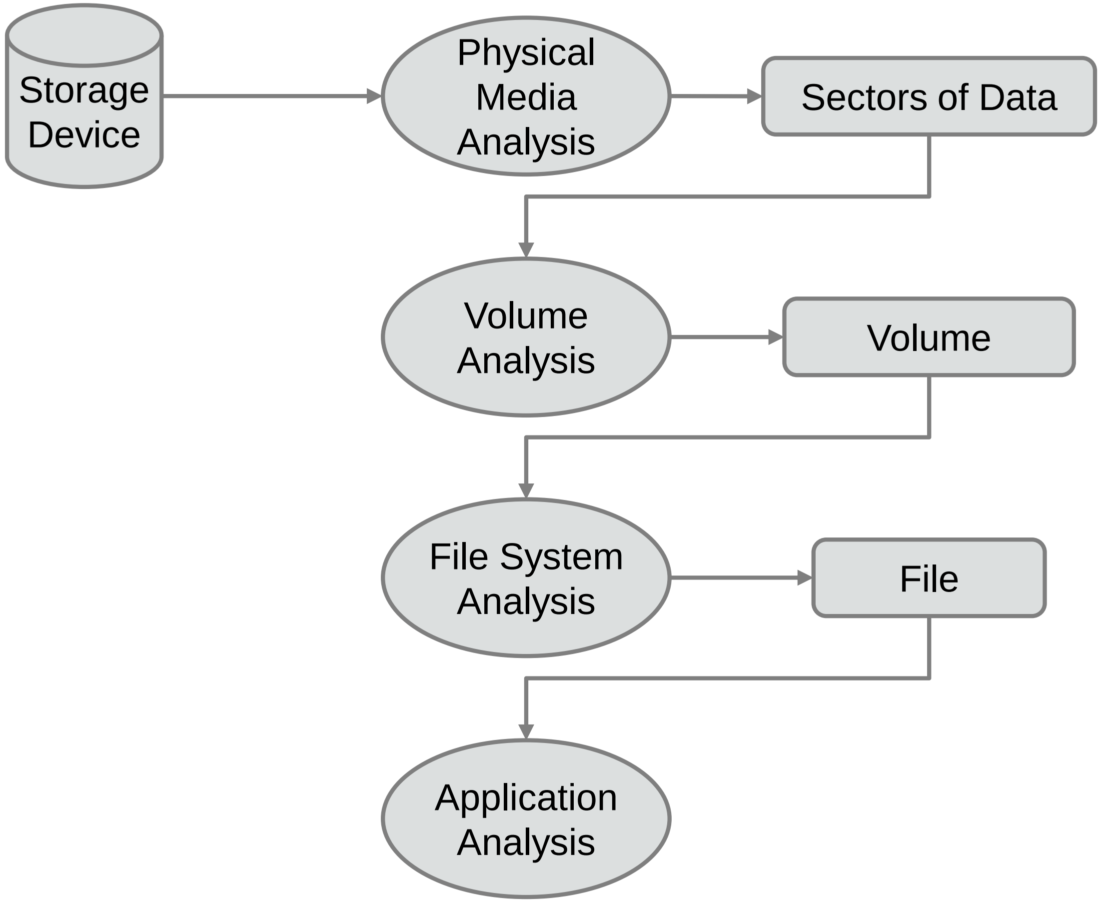


Fraunhofer FKIE

Pooled Storage File Systems

Recap: The Sleuth Kit

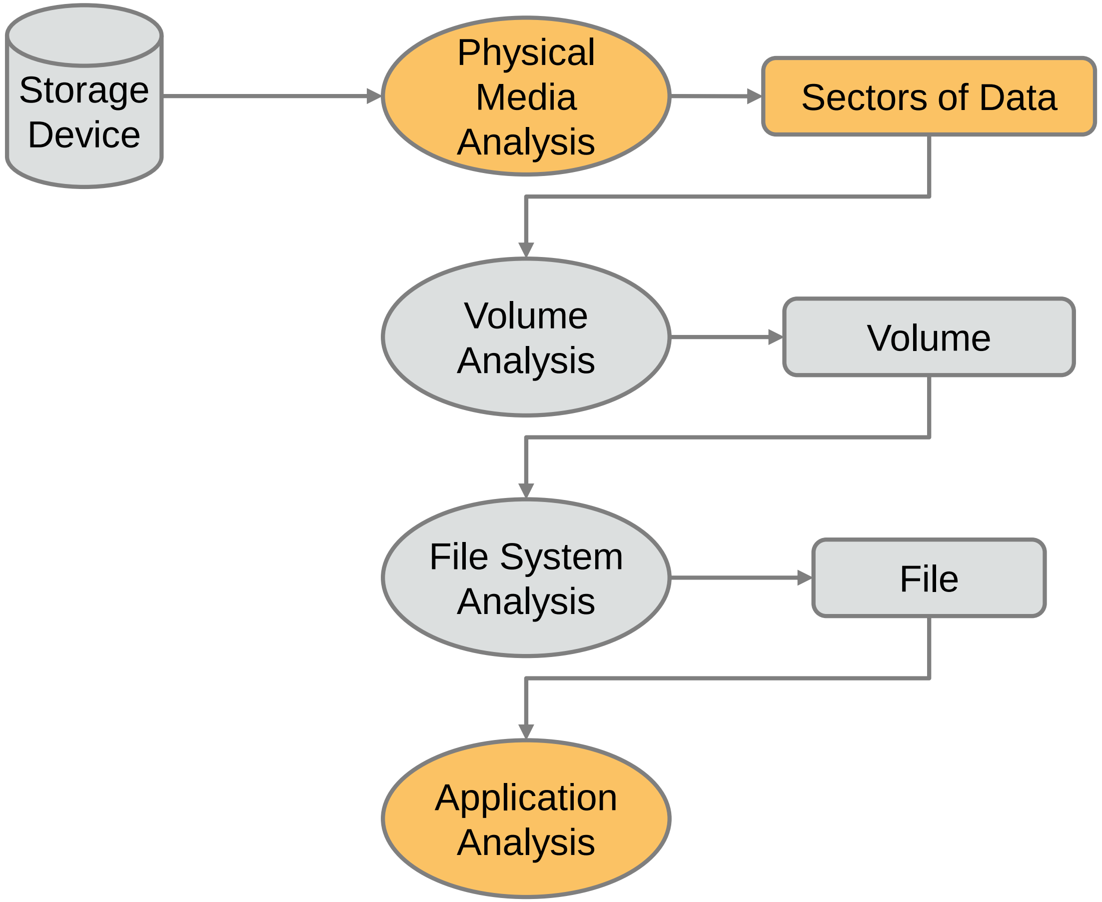
- Model needs an update to support pooled storage file systems (see talk @ DFRWS USA 2017)



Pooled Storage File Systems

Recap: The Sleuth Kit

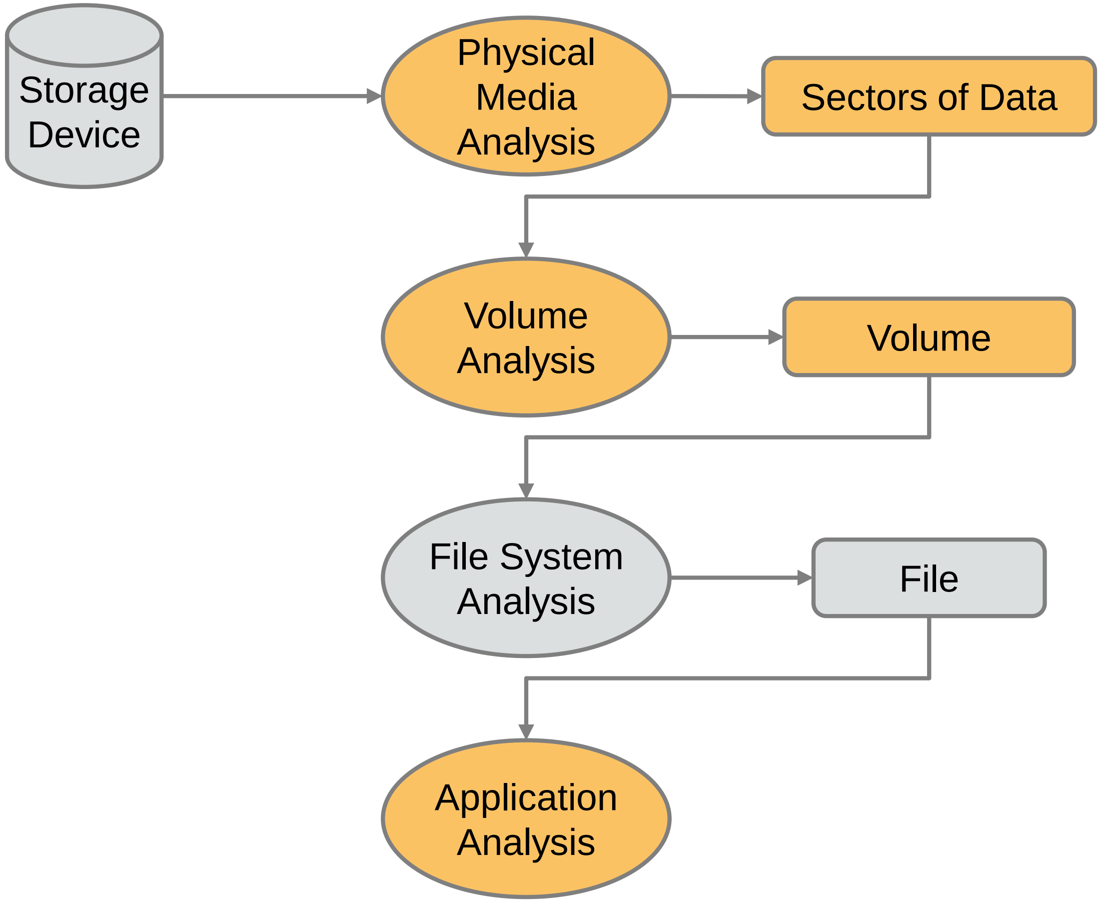
- Model needs an update to support pooled storage file systems (see talk @ DFRWS USA 2017)
  - Physical media analysis and application analysis are file system independent



Pooled Storage File Systems

Recap: The Sleuth Kit

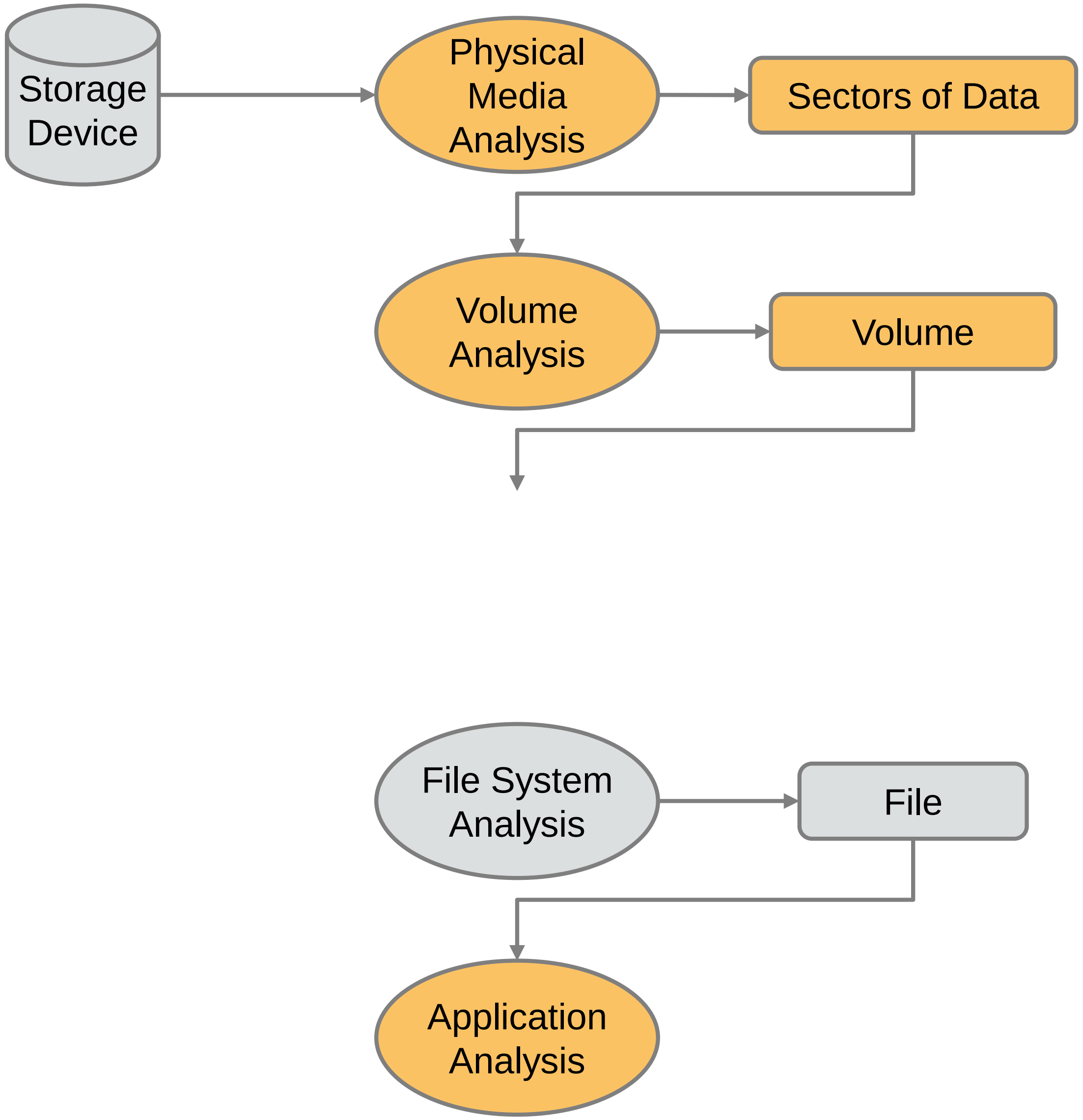
- Model needs an update to support pooled storage file systems (see talk @ DFRWS USA 2017)
  - Physical media analysis and application analysis are file system independent
  - Volume analysis is still required



Pooled Storage File Systems

Recap: The Sleuth Kit

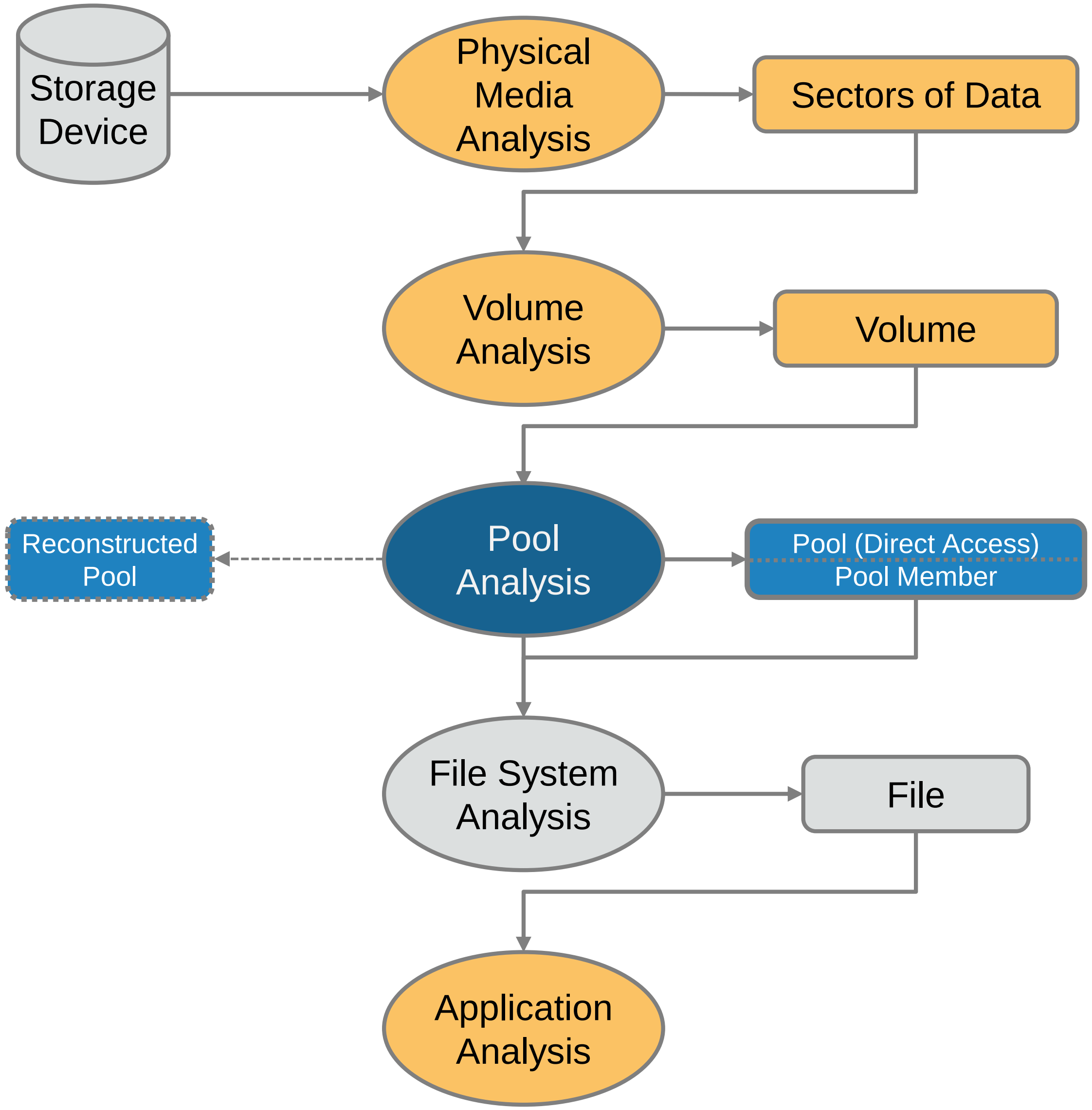
- Model needs an update to support pooled storage file systems (see talk @ DFRWS USA 2017)
  - Physical media analysis and application analysis are file system independent
  - Volume analysis is still required



Pooled Storage File Systems

Recap: The Sleuth Kit

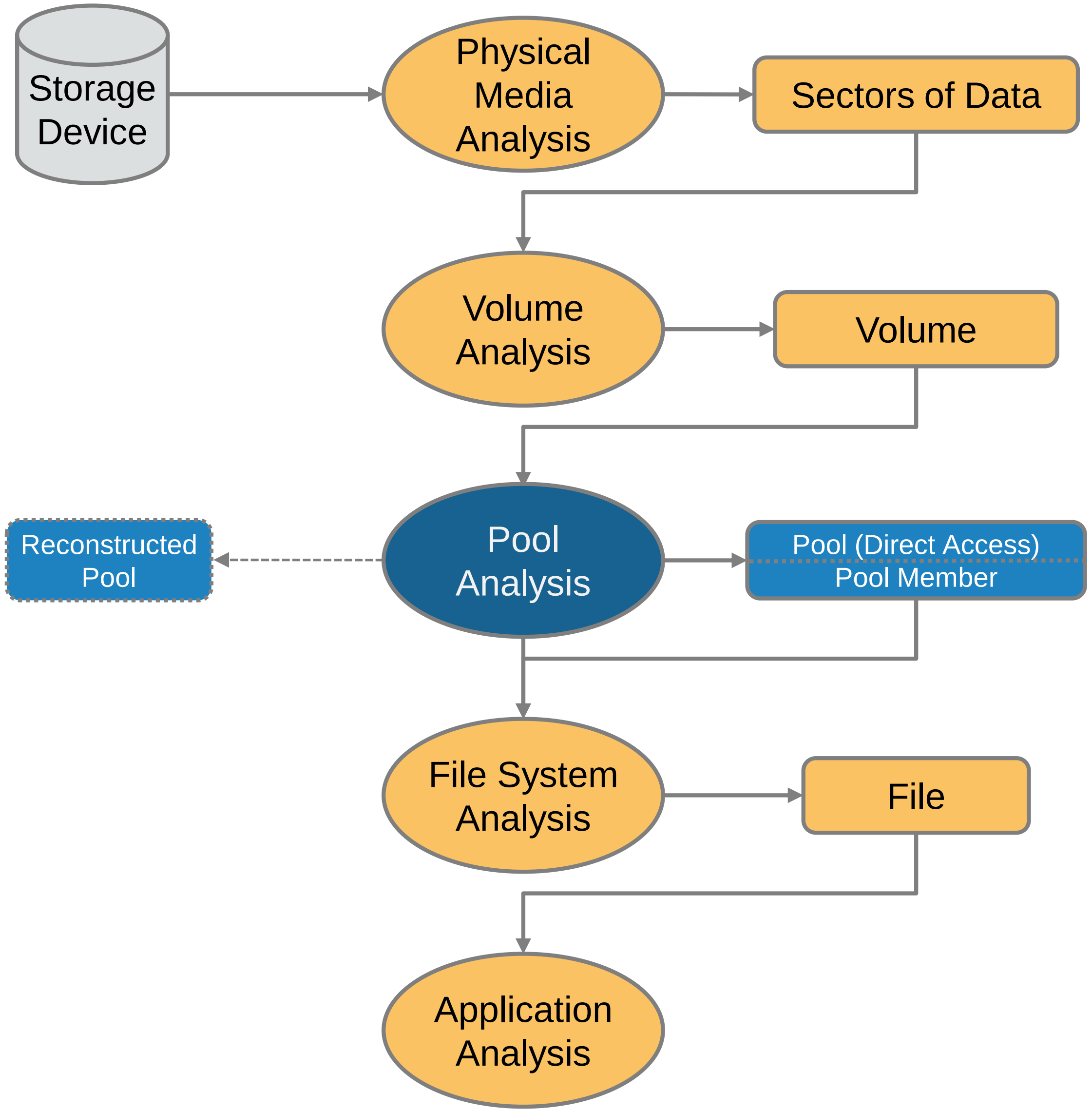
- Model needs an update to support pooled storage file systems (see talk @ DFRWS USA 2017)
  - Physical media analysis and application analysis are file system independent
  - Volume analysis is still required
  - Pool analysis becomes an additional step (performs the logical to physical mapping)



Pooled Storage File Systems

Recap: The Sleuth Kit

- Model needs an update to support pooled storage file systems (see talk @ DFRWS USA 2017)
  - Physical media analysis and application analysis are file system independent
  - Volume analysis is still required
  - Pool analysis becomes an additional step (performs the logical to physical mapping)
  - File system analysis is performed on a pool with direct access



Pooled Storage File Systems
The Sleuth Kit

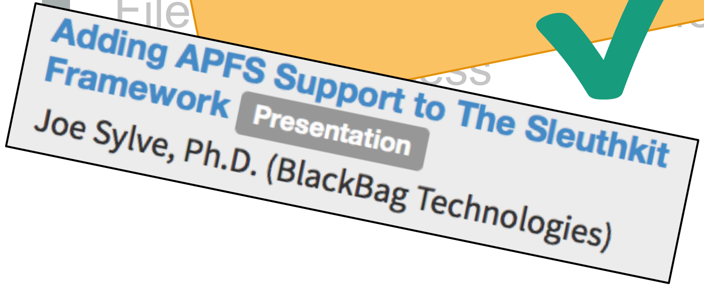
- Model needs an update to support pooled storage file systems (see talk @ DFRWS USA 2017)
  - Physical media analysis and application analysis are file system independent
  - Volume a… …ired
  - Po… …dditional step
  - File … …med on a p… …ss

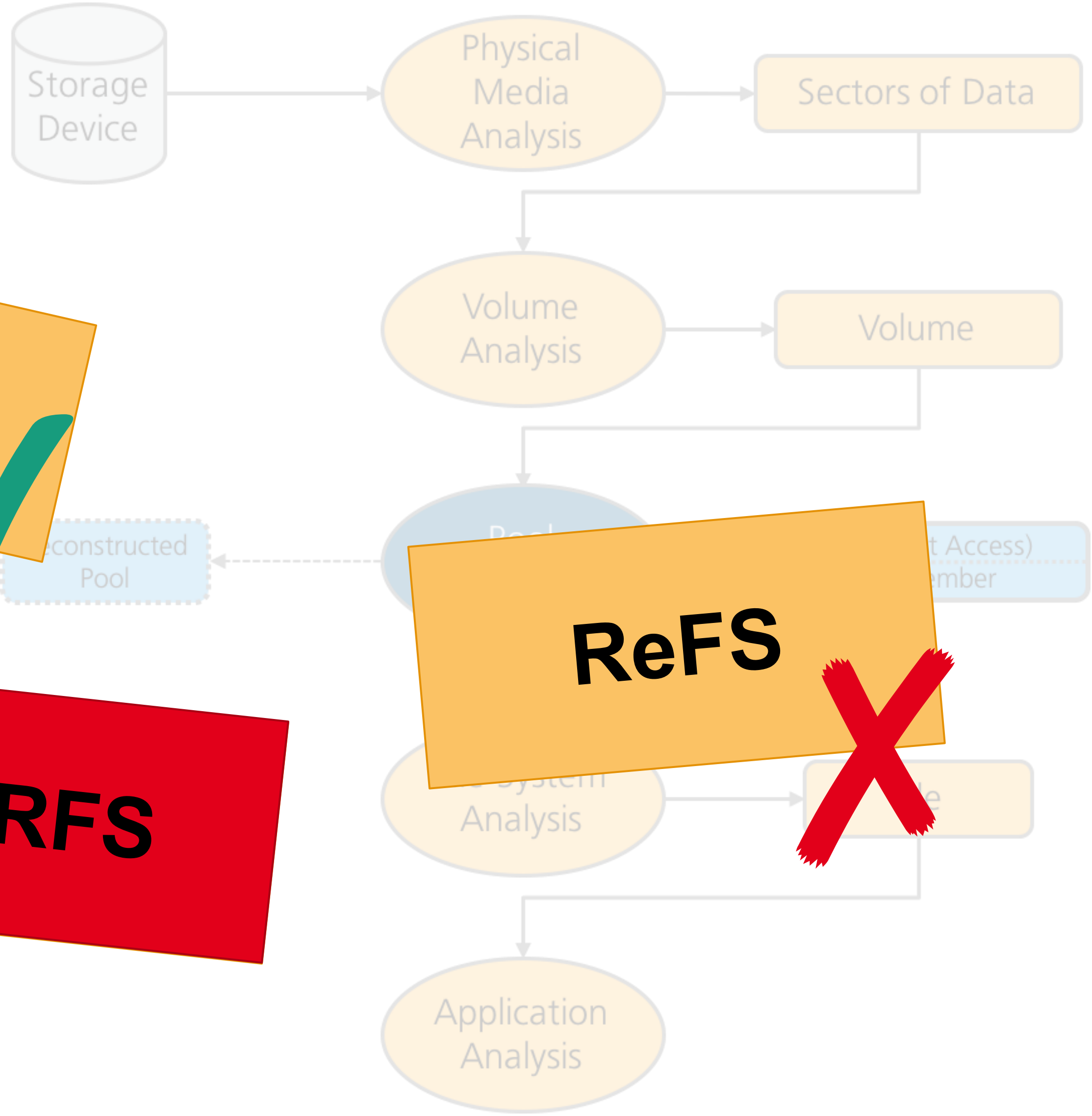
ZFS ✓

APFS ✓

BTRFS

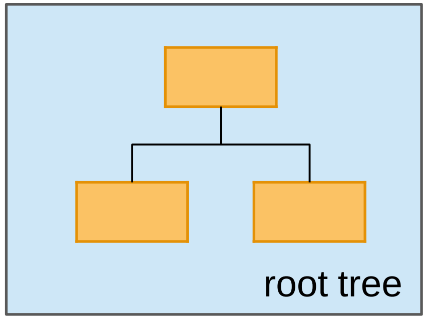
ReFS ✗





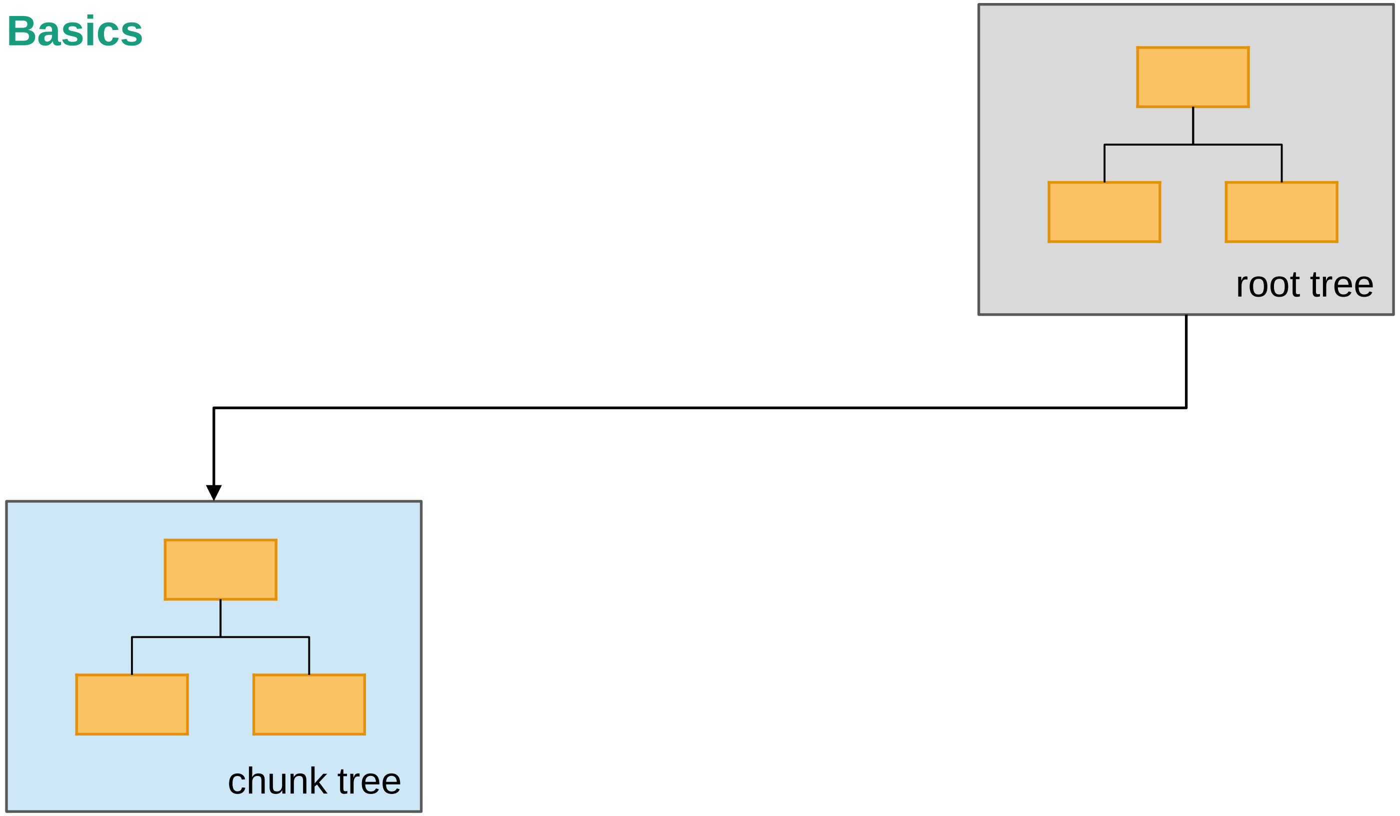
Fraunhofer FKIE

BTRFS
Basics



Stores the addresses of
the roots of the trees

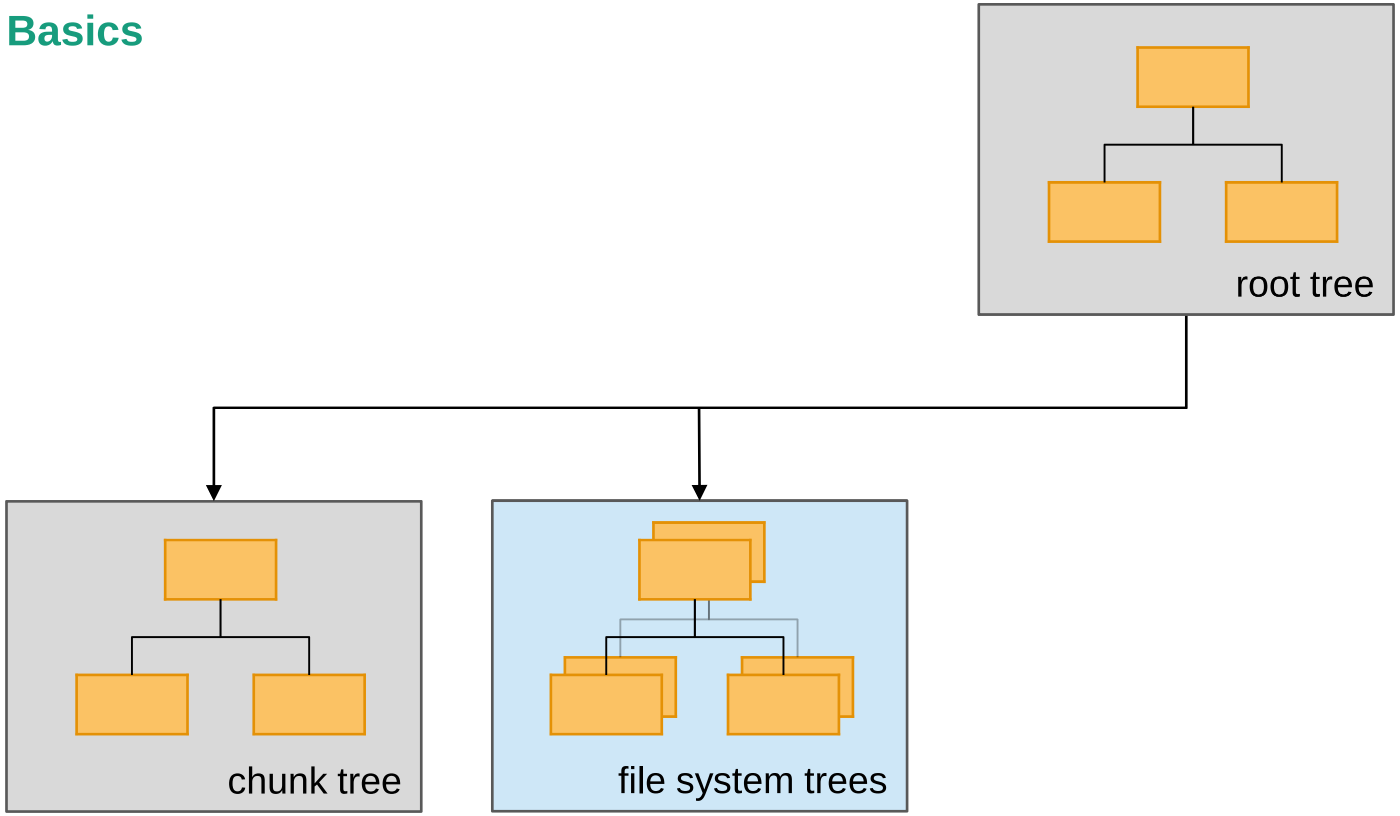
BTRFS
Basics



Defines chunks used for the mapping from logical to physical addresses

BTRFS

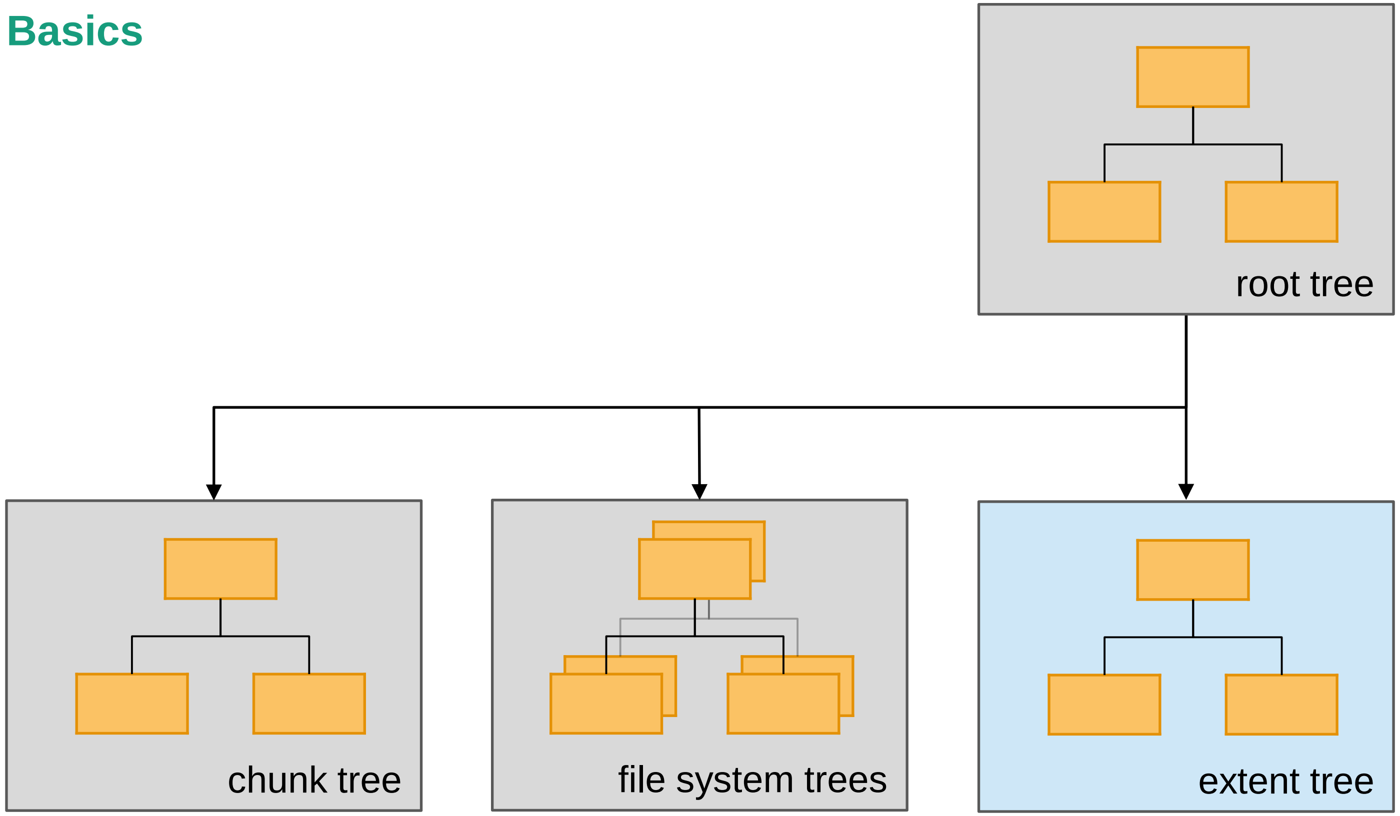
Basics



Stores the file and directory hierarchy of file systems, snapshots and subvolumes

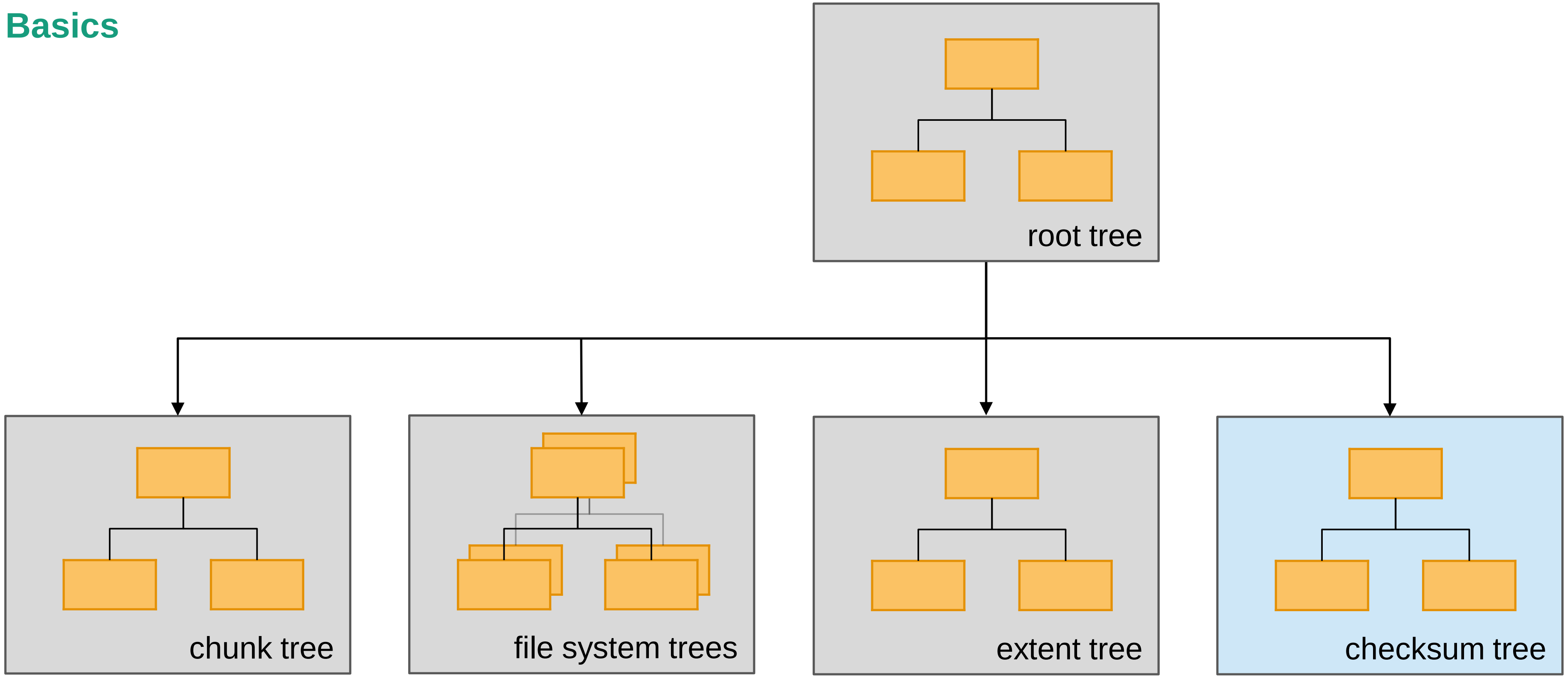
BTRFS

Basics



Keeps record of
the allocation
in BTRFS

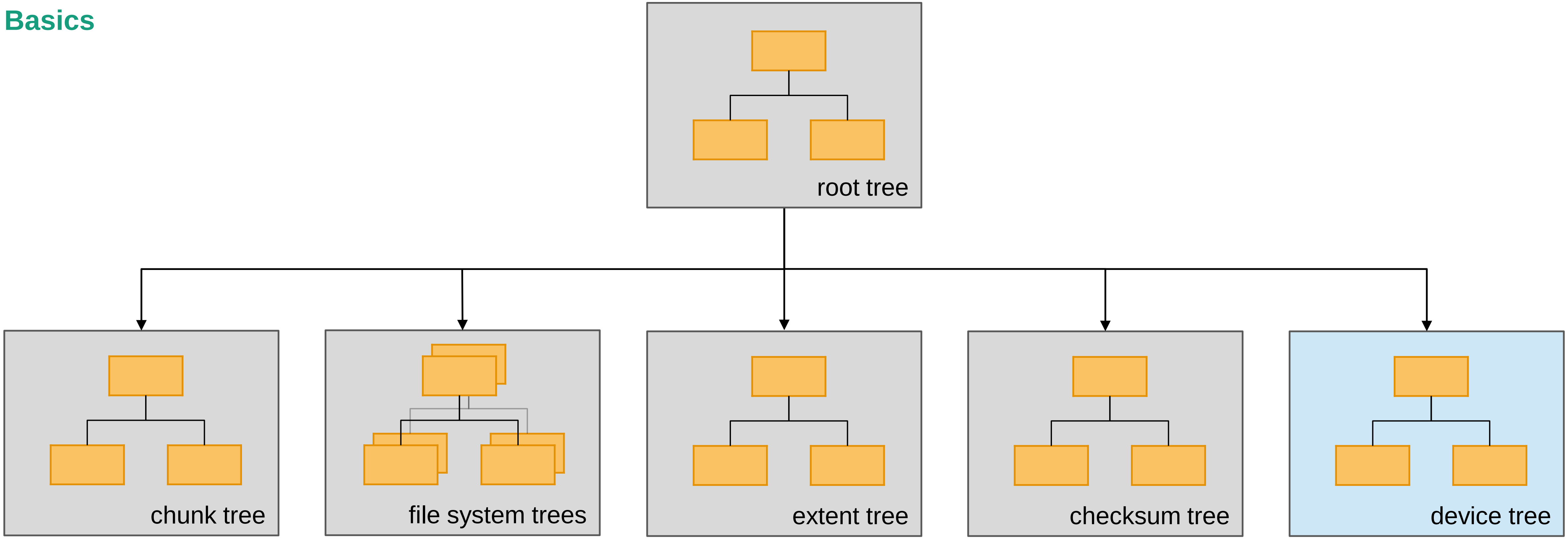
BTRFS
Basics



Stores checksums
for each block

BTRFS

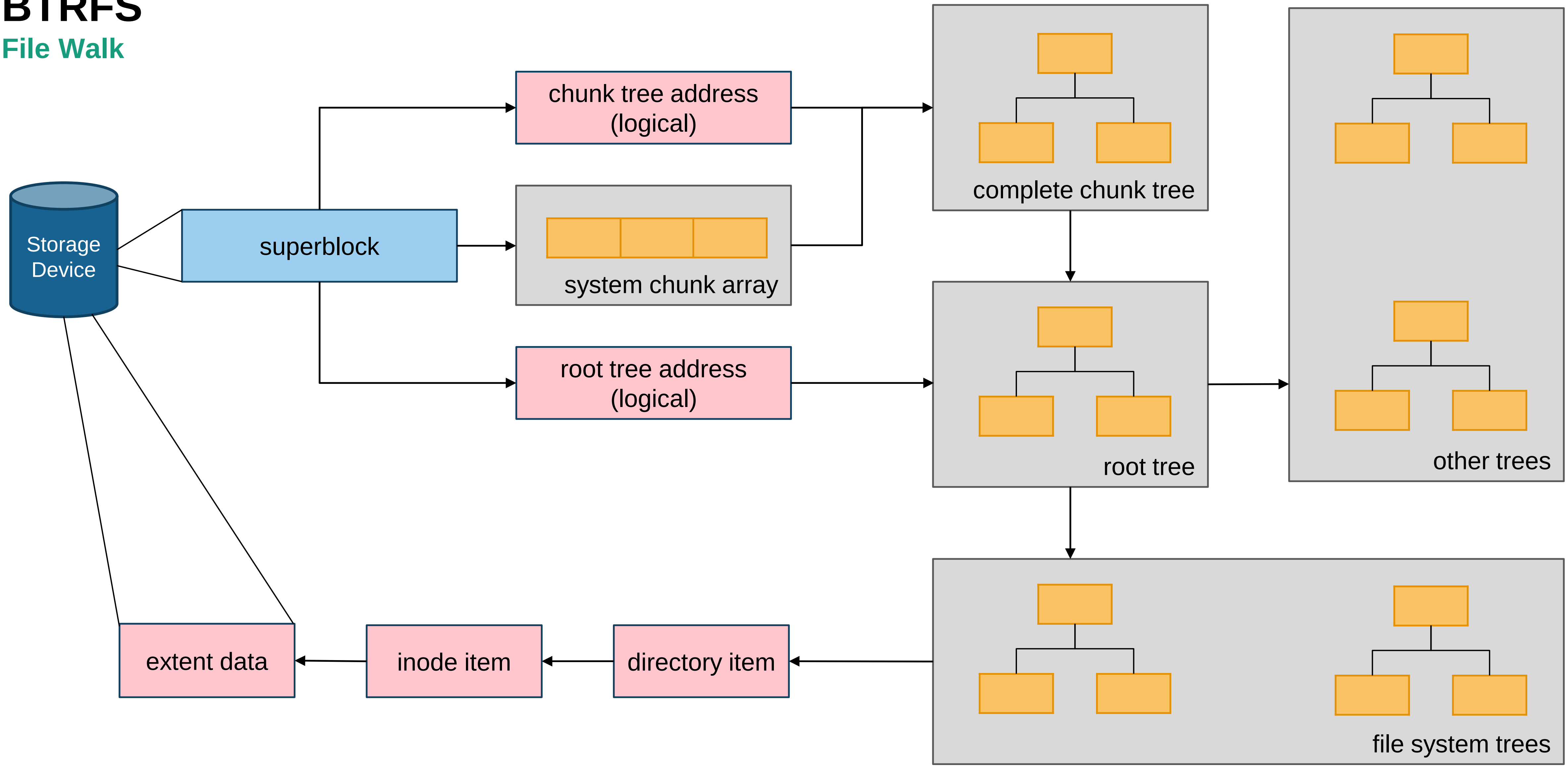
Basics



Used for the mapping from physical to logical addresses

Fraunhofer FKIE

BTRFS
File Walk



BTRFS

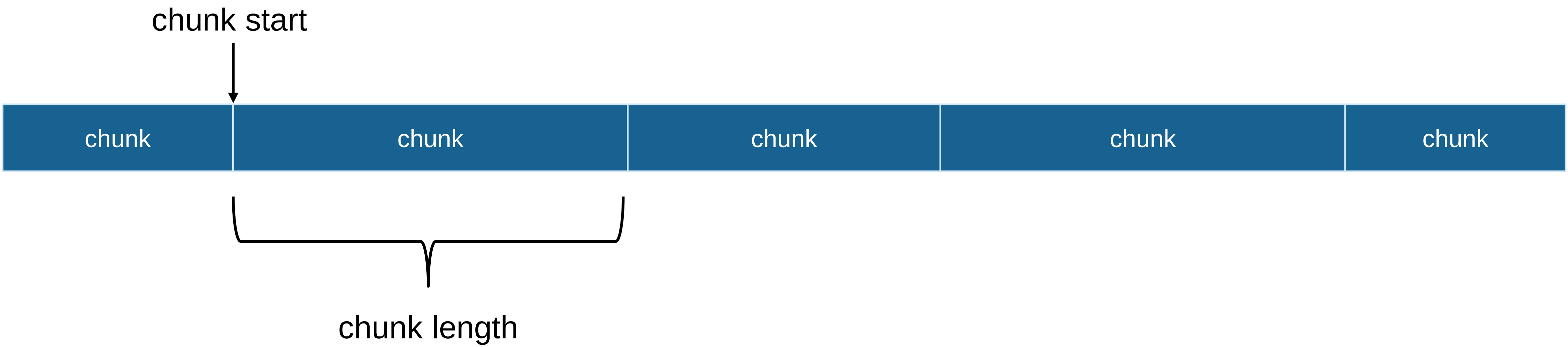
Multiple Device Support

- BTRFS‘ logical address space is divided into chunks defined in the chunk tree

BTRFS

Multiple Device Support

- BTRFS‘ logical address space is divided into chunks defined in the chunk tree



BTRFS

Multiple Device Support

- BTRFS‘ logical address space is divided into chunks defined in the chunk tree

chunk	chunk	chunk	chunk	chunk

```
chunk_start 34380000
chunk_length  800000
stripe_len     10000
num_stripes        4
type      DATA|RAID0
```

BTRFS

Multiple Device Support

- BTRFS‘ logical address space is divided into chunks defined in the chunk tree

chunk	chunk	chunk	chunk	chunk

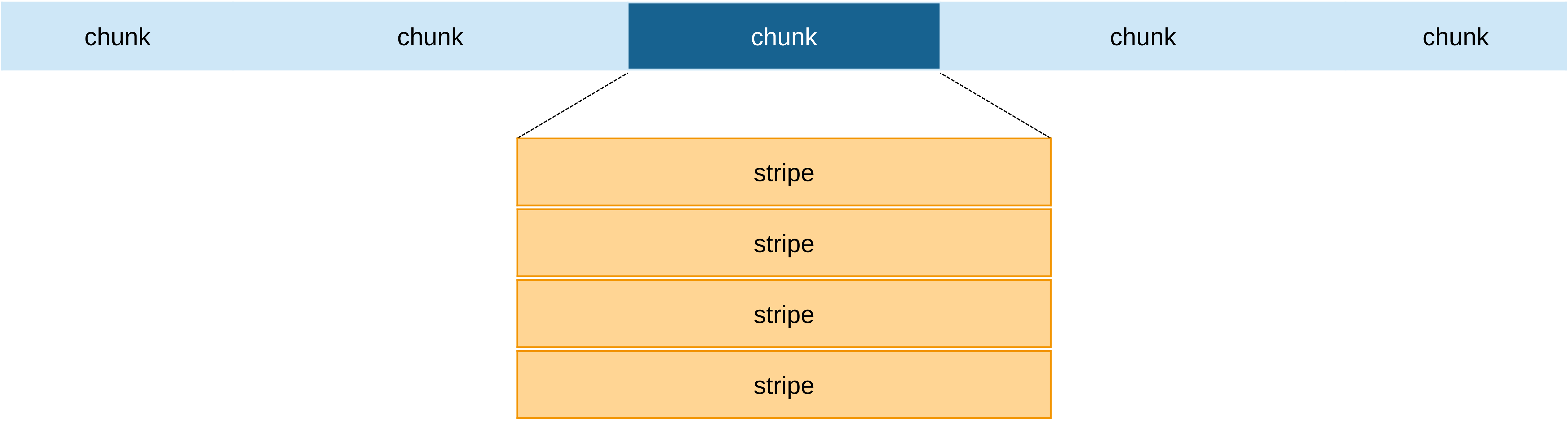
```
chunk_start 34380000
chunk_length  800000
stripe_len     10000
num_stripes        4
type      DATA|RAID0
```

What is stored inside of the chunk?

How is it stored inside of the chunk?

BTRFS
Multiple Device Support

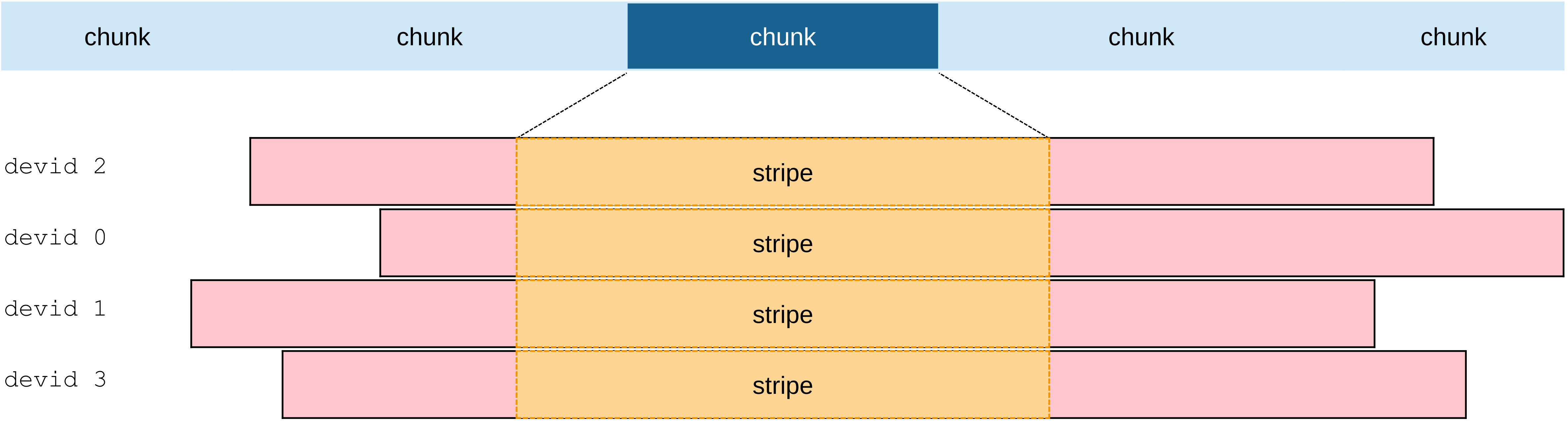
- BTRFS‘ logical address space is divided into chunks defined in the chunk tree
- Each chunk utilizes a certain number of stripes for mapping its data



BTRFS

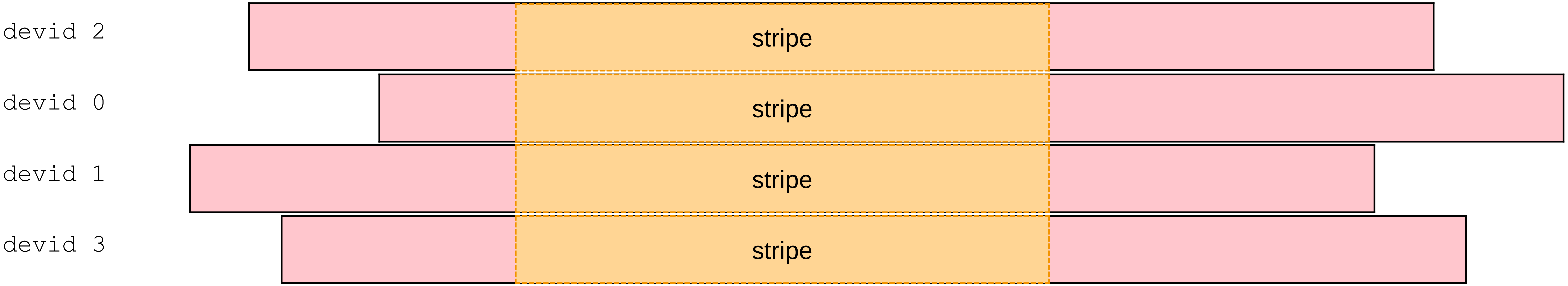
Multiple Device Support

- BTRFS' logical address space is divided into chunks defined in the chunk tree
- Each chunk utilizes a certain number of stripes for mapping its data
- Stripes are physical areas on devices of the pool starting at a given offset



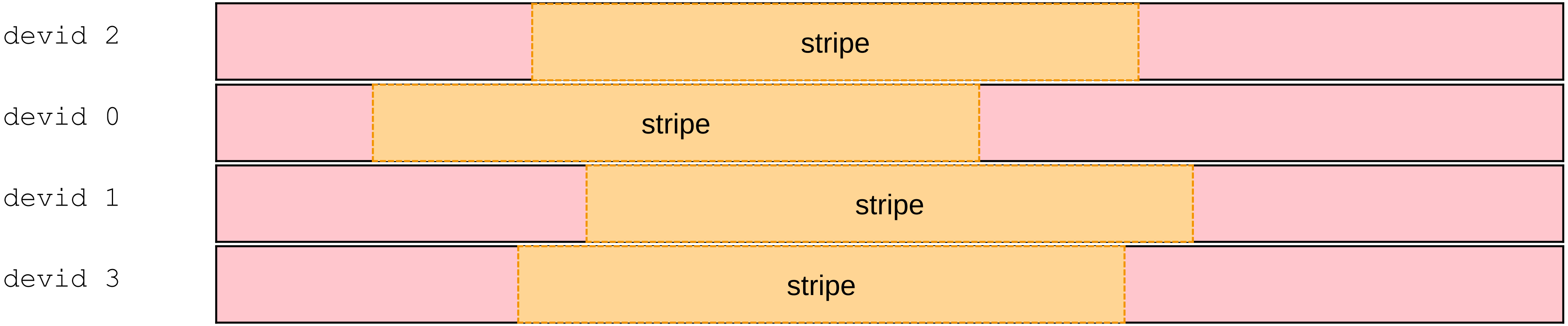
BTRFS
Multiple Device Support

- BTRFS' logical address space is divided into chunks defined in the chunk tree
- Each chunk utilizes a certain number of stripes for mapping its data
- Stripes are physical areas on devices of the pool starting at a given offset



BTRFS
Multiple Device Support

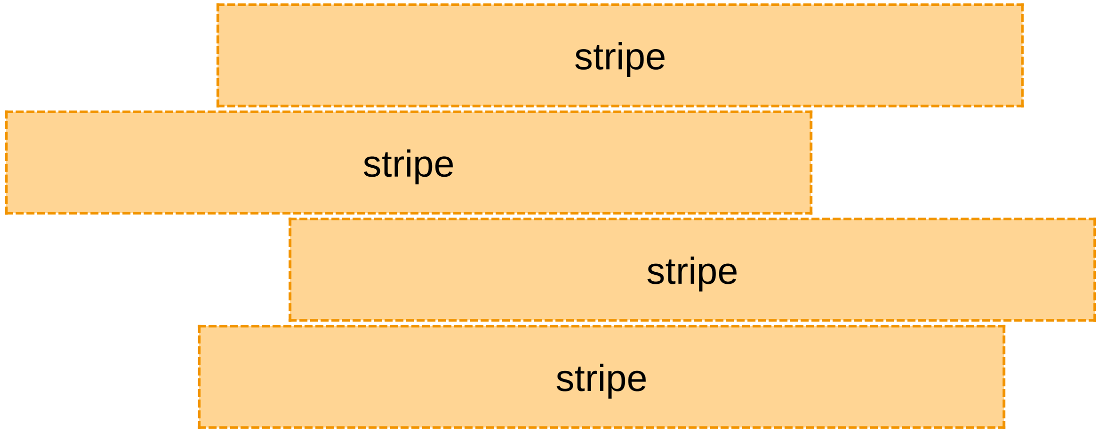
- BTRFS‘ logical address space is divided into chunks defined in the chunk tree
- Each chunk utilizes a certain number of stripes for mapping its data
- Stripes are physical areas on devices of the pool starting at a given offset



BTRFS

Multiple Device Support

- BTRFS‘ logical address space is divided into chunks defined in the chunk tree
- Each chunk utilizes a certain number of stripes for mapping its data
- Stripes are physical areas on devices of the pool starting at a given offset
- Stripes are furthermore divided into equally sized “stripe units“

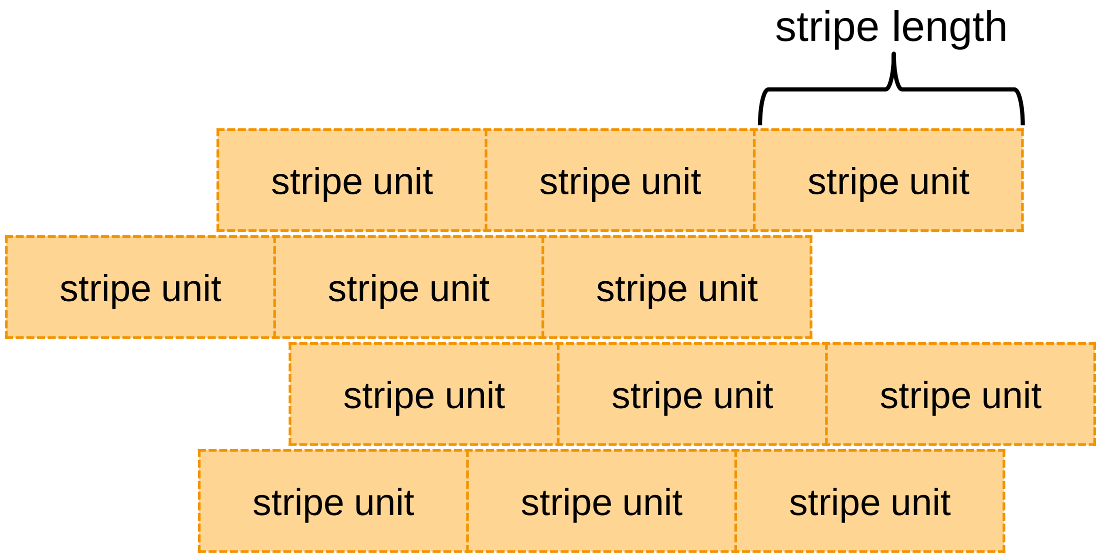


Fraunhofer FKIE

BTRFS

Multiple Device Support

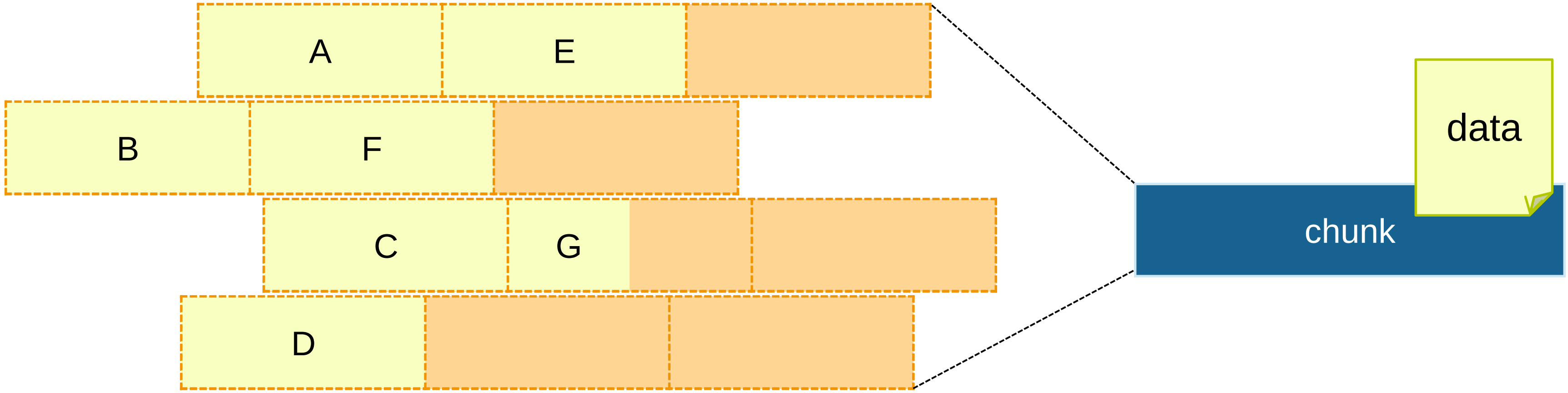
- BTRFS‘ logical address space is divided into chunks defined in the chunk tree
- Each chunk utilizes a certain number of stripes for mapping its data
- Stripes are physical areas on devices of the pool starting at a given offset
- Stripes are furthermore divided into equally sized “stripe units“



BTRFS

Multiple Device Support: RAID0

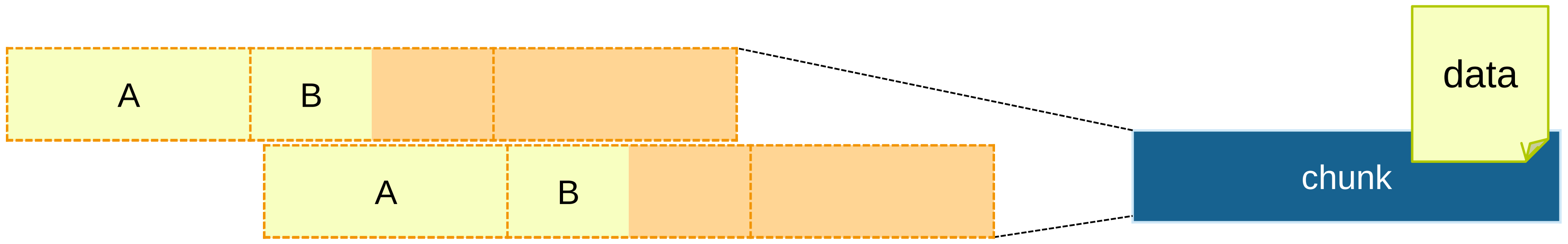
- RAID0 stripes the data across all stripes of the chunk
- BTRFS uses all available devices for a RAID0 chunk configuration
- Missing disk leads to definite data loss



BTRFS

Multiple Device Support: RAID1

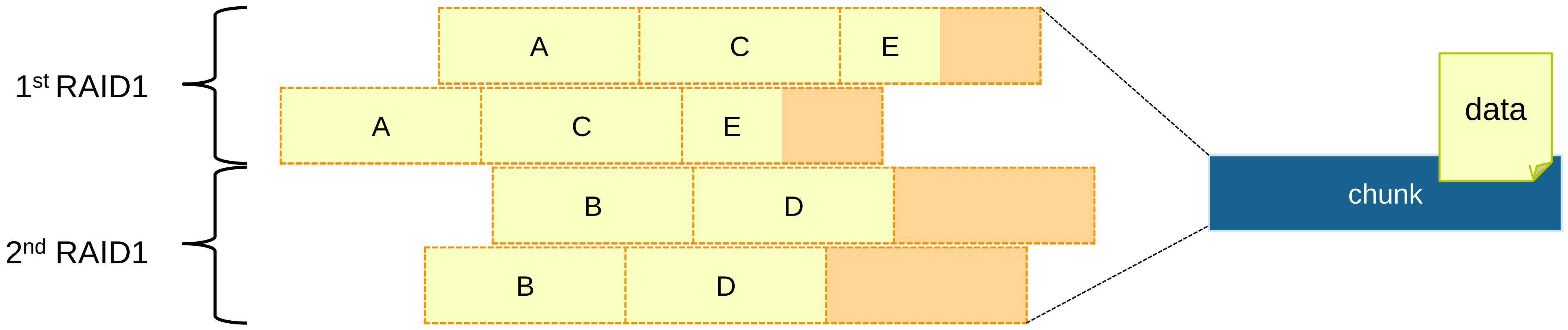
- RAID1 uses a pair of stripes for each chunk item
- Data is mirrored on both of these stripes



BTRFS

Multiple Device Support: RAID10

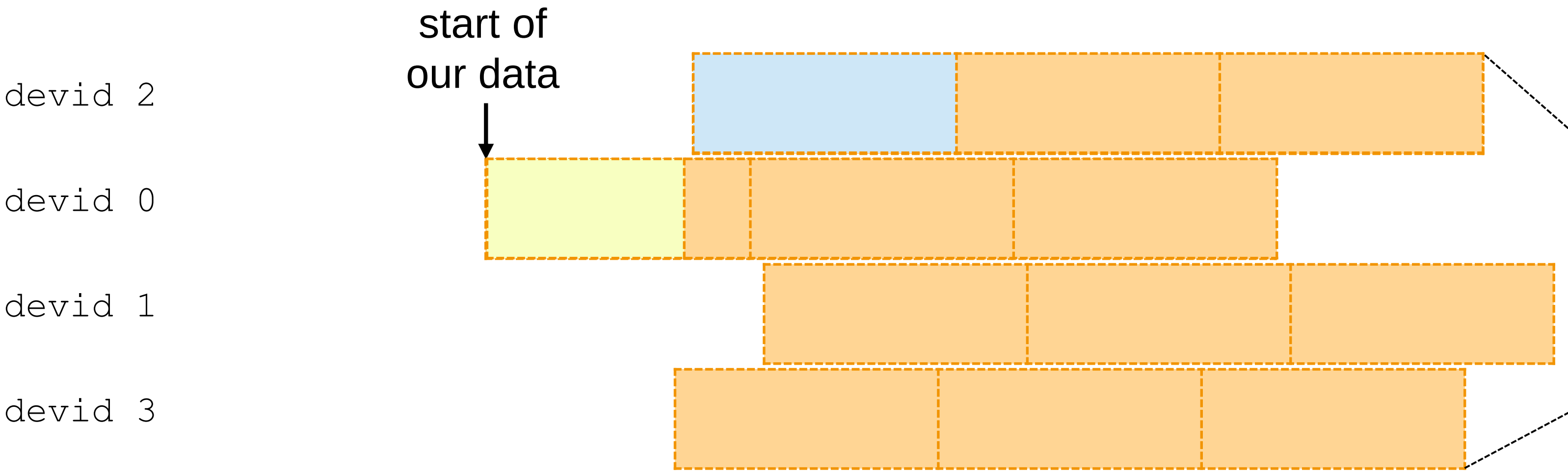
- All available stripes are split into RAID1 configurations
- Data is then striped across these configurations
- BTRFS uses two sub stripes for each RAID1 configuration



BTRFS

Logical-to-physical: RAID0 Example

- Accessing file with logical address 34390000 and length 8000
  1. Locate the corresponding chunk item
  2. Calculate the offset into the chunk (10000)
  3. Find the corresponding stripe, its device id (0) and physical offset
  4. Check if data fits into one stripe unit



devid	2
offset	7800000
devid	0
offset	2700000
devid	1
offset	7800000
devid	3
offset	3140000
chunk_start	34380000
chunk_length	800000
stripe_len	10000
num_stripes	4
type	DATA|RAID0

chunk

BTRFS

Logical-to-physical: In detail

1. Locate the **chunk item** containing the given logical target address ($\mathtt{t}_{\mathtt{log}}$) in the chunk tree. This gives us the **logical start address of the chunk** ($\mathtt{c}_{\mathtt{log}}$).
2. Calculate the **difference** (Δ) between the logical target address and the logical start address of the chunk.

   $$\Delta = \mathtt{t}_{\mathtt{log}} - \mathtt{c}_{\mathtt{log}}$$

   This difference represents the offset of the target address within the chunk item.
3. Use Δ and the stripe length (`stripeLen`) to compute the total number of stripe units preceding our target address (`preStripeUnits`):

   $$\mathtt{preStripeUnits} = \left\lfloor \frac{\Delta}{\mathtt{stripeLen}} \right\rfloor$$

4. Find out on **which stripe** (`targetStripe`) our logical address (and thus the start of the data) lies by calculating the total number of preceding units modulus the number of stripes (`nStripes`).

   $$\mathtt{targetStripe} = \mathtt{preStripeUnits} \bmod \mathtt{nStripes}$$

5. Knowing the corresponding stripe gives us the **physical start offset** (`phyStripeOff`) of the data on the device specified in the chunk item.
6. Calculate the **number of units** (`nStripeUnits`) that have already been allocated on our stripe by dividing the total number of units already filled by the number of available stripes.

   $$\mathtt{nStripeUnits} = \left\lfloor \frac{\mathtt{preStripeUnits}}{\mathtt{nStripes}} \right\rfloor$$

7. Calculate the offset within the unit (`unitOff`) on our stripe.

   $$\mathtt{unitOff} = \Delta \bmod \mathtt{stripeLen}$$

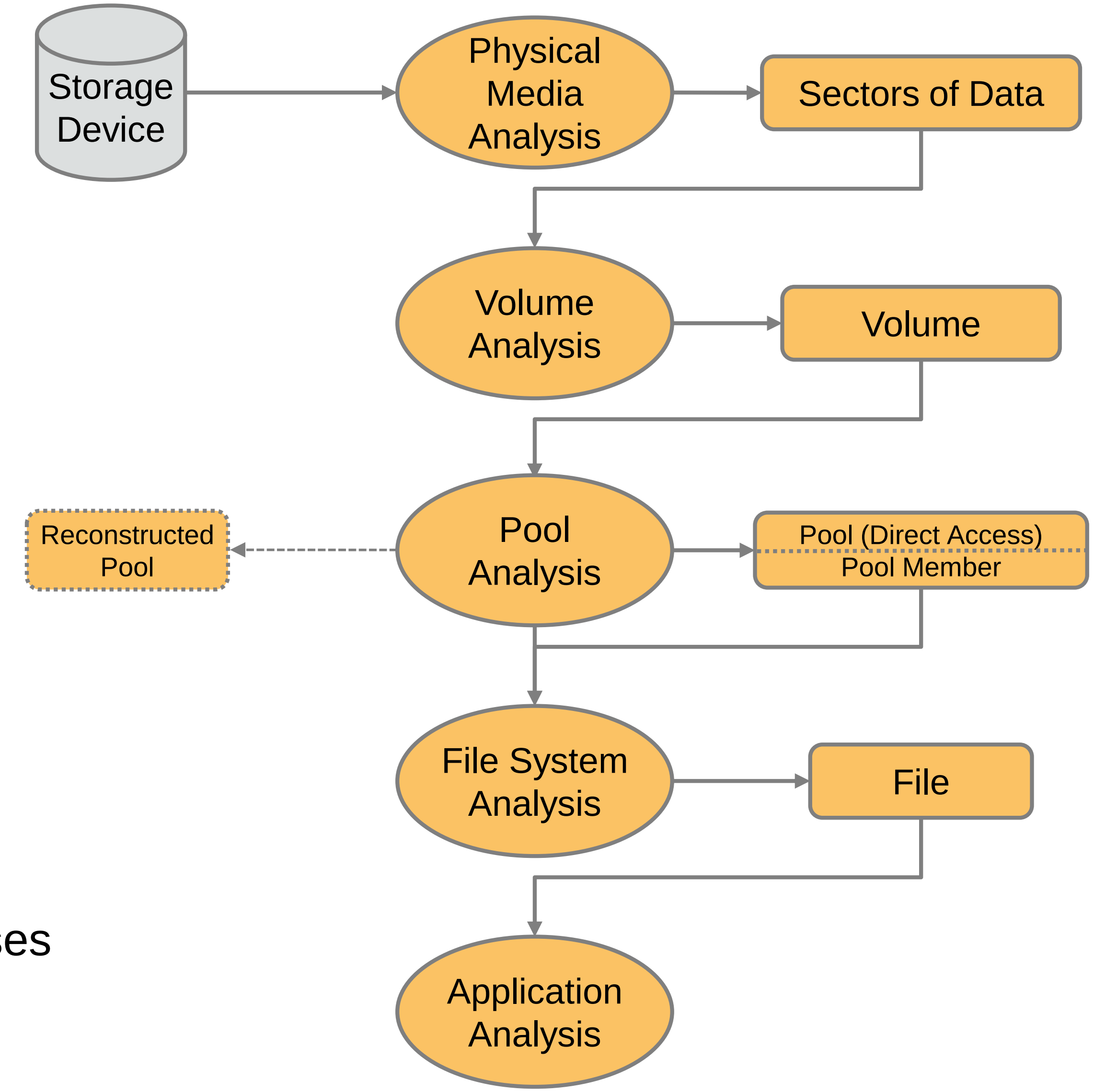
8. Adding the calculated values results in the **final physical offset** (`phyOff`)

   $$\begin{aligned}\mathtt{phyOff} = \; & \mathtt{phyStripeOff} \\ & + \mathtt{nStripeUnits} \cdot \mathtt{stripeLen} \\ & + \mathtt{unitOff}\end{aligned}$$

BTRFS

Into The Sleuth Kit

- ✓ First and last step are file system independent
- ✓ Not only raw storage devices can be part of a BTRFS storage pool
- ✓ Reconstructing a BTRFS file system only gives access to the most recent version of the file sytem
- ✓ File system analysis needs direct access to the pool
- ✓ Pool analysis needs to:
  - ✓ Detect BTRFS members and their configuration
  - ✓ Perform the logical-to-physical mapping of addresses
  - ✓ Deal with missing members



Forensic Analysis of BTRFS

- pls command is used for the pool and pool membership detection

```
$ pls /tmp/BTRFS
FSID:                     0B8BF06F-B379-4051-83AC-E8A0C30F7124
System chunks:            RAID1 (1/1)
Metadata chunks:          RAID1 (1/1)
Data chunks:              RAID0 (1/1)
Number of devices:   3 (3 detected)
        ID:          1
        GUID:        FFA6CCA6-F221-48AE-968E-82F8355690C5
        ID:          2
        GUID:        A1A41337-1BD4-4DA4-9E54-957D48F76F19
        ID:          3
        GUID:        52DA169B-FC06-4D43-AE18-42727930E0CB
```

Forensic Analysis of BTRFS

- pls command is used for the pool and pool membership detection
- Common TSK tools support pools by using –P
- Still work on existing file systems as well

```
$ fsstat -P /tmp/BTRFS
Label:
File system UUID:        0B8BF06F-B379-4051-83AC-E8A0C30F7124
Root tree root address:           30179328
Chunk tree root address:          20987904
Log tree root address:            0
Generation:              13
Chunk root generation:            5
Total bytes:             3145728000
Number of devices:                3
Total size:              2GB
Used size:               91MB
```

Forensic Analysis of BTRFS

What about deleted files?

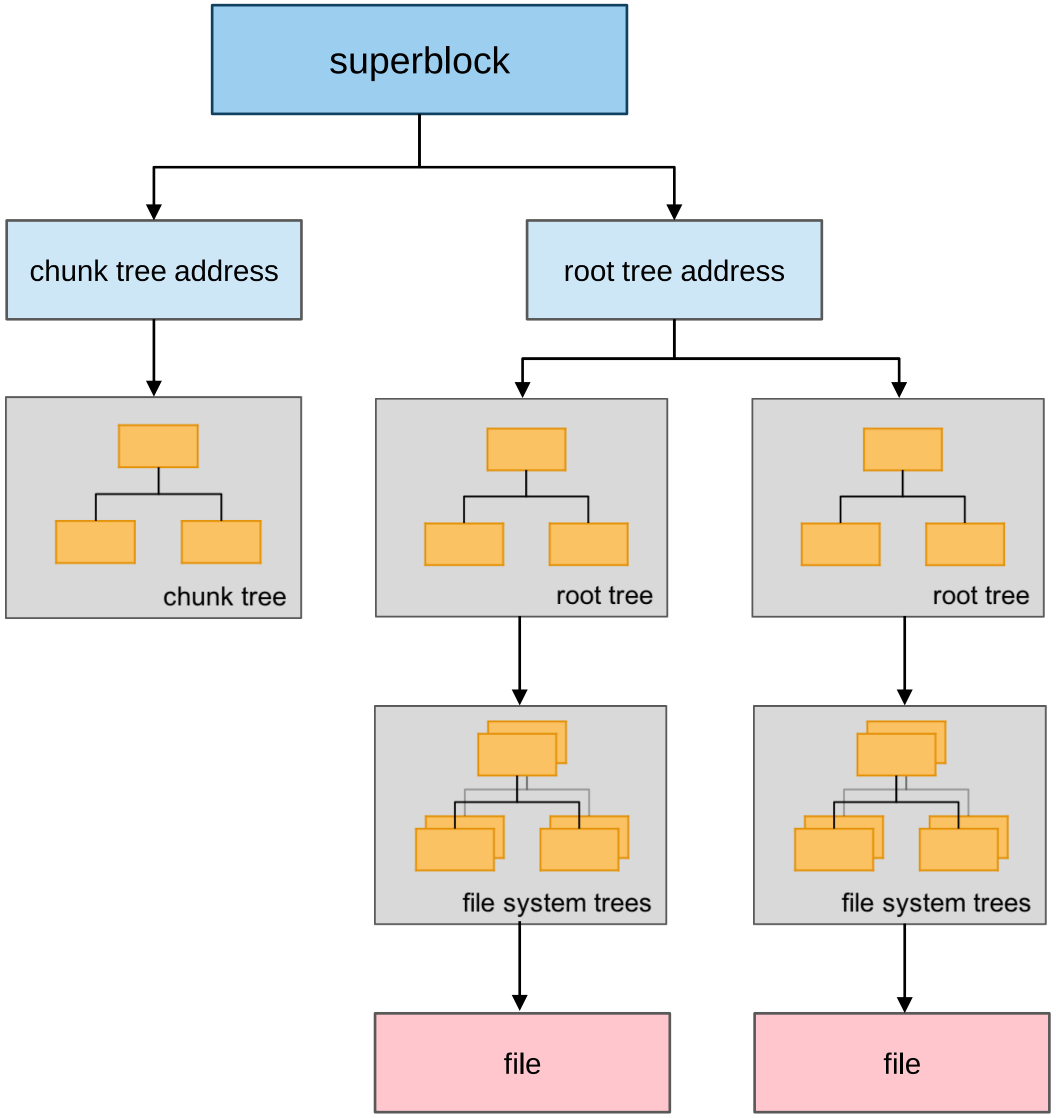
- pls command is used for the pool and pool membership detection
- Common TSK tools support pools by using –P
- Still work on existing file systems as well

```
$ fls -P /tmp/BTRFS
d/d 257:    data
+ r/r 258:    network_capture.pcap
+ r/r 259:    application01.dmg
+ r/r 260:    application02.dmg
d/d 261:    home
+ d/d 262:    user
++ d/d 263:    images
+++ r/r 264:    img00032.jpg
+++ r/r 266:    img00034.jpg
+++ r/r 267:    img00031.jpg
[...]
```

Forensic Analysis of BTRFS

File Recovery

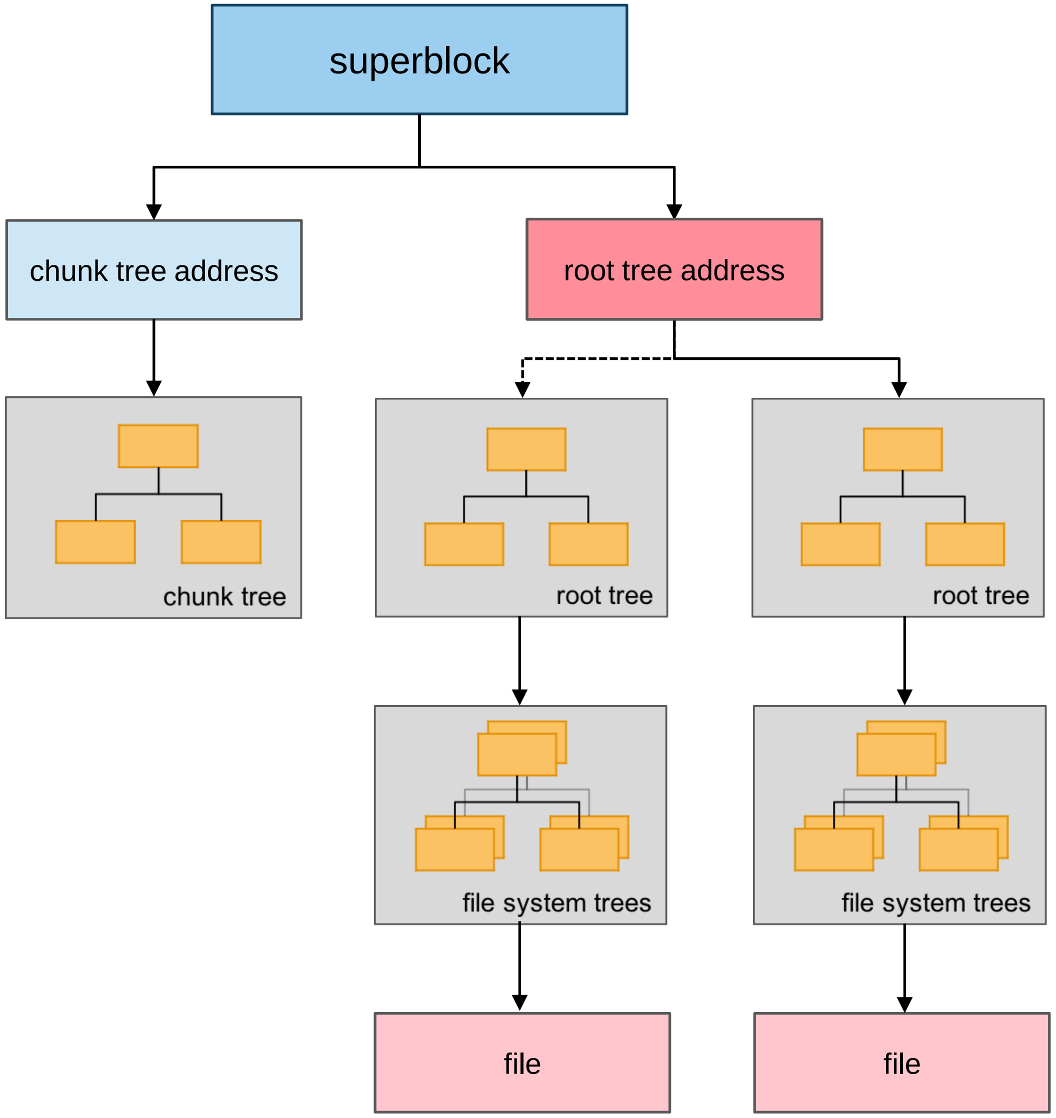
- Copy-on-write creates a lot of artifacts
- Old metadata and data are not part of the most recent version of the file system
- BTRFS refers to them as generations
- No inode table or FAT to search for unallocated metadata structures



Forensic Analysis of BTRFS

File Recovery

- Copy-on-write creates a lot of artifacts
- Old metadata and data are not part of the most recent version of the file system
- BTRFS refers to them as generations
- No inode table or FAT to search for unallocated metadata structures



Forensic Analysis of BTRFS

Metadata Based File Recovery

- BTRFS stores four backup roots
- More root tree addresses can be found by using btrfs-find-root

```
$ pls /tmp/BTRFS
[...]
Backup Roots:
1. tree root at 29687808 (generation: 10)
        chunk tree root at 20987904 (generation: 5)
2. tree root at 29933568 (generation: 11)
        chunk tree root at 20987904 (generation: 5)
3. tree root at 30048256 (generation: 12)
        chunk tree root at 20987904 (generation: 5)
4. tree root at 30179328 (generation: 13)
        chunk tree root at 20987904 (generation: 5)
```

Forensic Analysis of BTRFS

Metadata Based File Recovery

- BTRFS stores four backup roots
- More root tree addresses can be found by using btrfs-find-root
- By using –T another generation and thus older root trees can be used

```
$ fls –P /tmp/BTRFS –T 12
[...]
d/d 261:      home
+ d/d 262:      user
++ d/d 263:      images
+++ r/r 264:      img00032.jpg
+++ r/r 266:      img00034.jpg
+++ r/r 267:      img00031.jpg
+++ r/r 268:      img00040.jpg
+++ r/r 269:      img00041.jpg
[...]
```

Forensic Analysis of BTRFS

Metadata Based File Recovery

- BTRFS stores four backup roots
- More root tree addresses can be found by using btrfs-find-root
- By using –T another generation and thus older root trees can be used

```
$ fls –P /tmp/BTRFS
[...]
d/d 261:      home
+ d/d 262:      user
++ d/d 263:      images
+++ r/r 264:      img00032.jpg
+++ r/r 266:      img00034.jpg
+++ r/r 267:      img00031.jpg
++ d/d 280:      documents
+++ r/r 281:      presentation.pdf
[...]
```

Forensic Analysis of BTRFS

Metadata Based File Recovery

- BTRFS stores four backup roots
- More root tree addresses can be found by using btrfs-find-root
- By using –T another generation and thus older root trees can be used
- File recovery can then be performed using icat

```
$ icat –P /tmp/BTRFS –T 12 269 > img00041.jpg
```

Forensic Analysis of BTRFS

Snapshots

- Snapshots contain consistent metadata and data
- Always check snapshots first before performing metadata based recovery

```
$ fsstat –P /tmp/BTRFS
[...]
Following subvolumes or snapshots were found:
258       snapshot_2018_07_11
259       snapshot_2018_07_12
260       snapshot_2018_07_13
261       snapshot_2018_07_14
```

Forensic Analysis of BTRFS

Snapshots

- Snapshots contain consistent metadata and data
- Always check snapshots first before performing metadata based recovery
- Snapshots can be accessed using -S

```
$ fls –P /tmp/BTRFS –S snapshot_2018_07_11
[...]
d/d 261:      home
+ d/d 262:      user
++ d/d 263:      images
+++ r/r 271:      img00050.jpg
+++ r/r 272:      img00051.jpg
[...]
```


Fraunhofer FKIE

Forensic Analysis of BTRFS

Missing Disks

- 3 disks with metadata mirrored (raid1) but only striped data (raid0)
- BTRFS still displays the metadata (though read-only)

```
$ mount –o degraded /dev/sda /tmp/btrfs

BTRFS: missing devices(1) exceeds the limit(0), writeable mount is not allowed
```

Forensic Analysis of BTRFS

Missing Disks

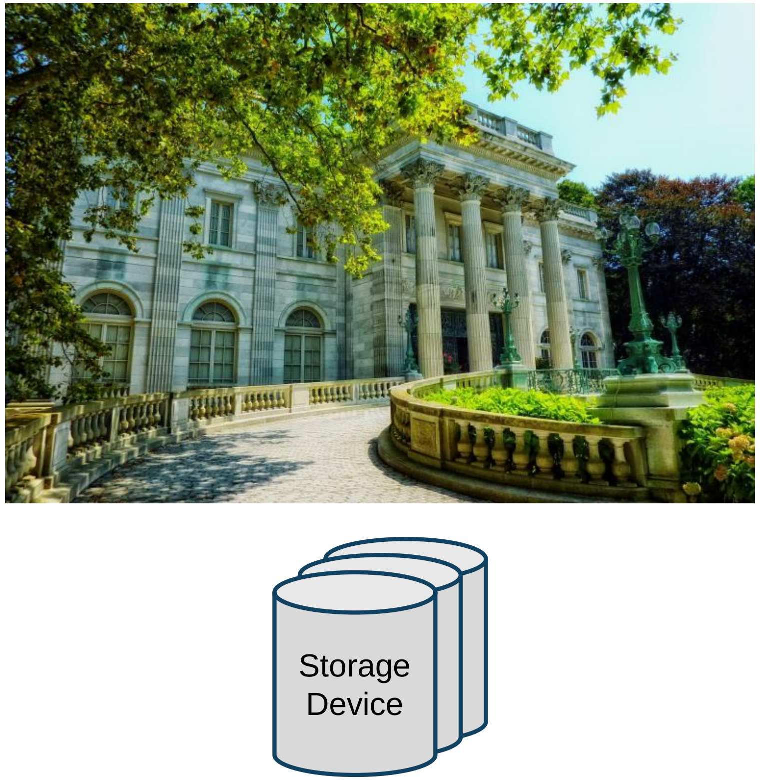
- 3 disks with metadata mirrored (raid1) but only striped data (raid0)
- BTRFS still displays the metadata (though read-only)
- Fails to open files

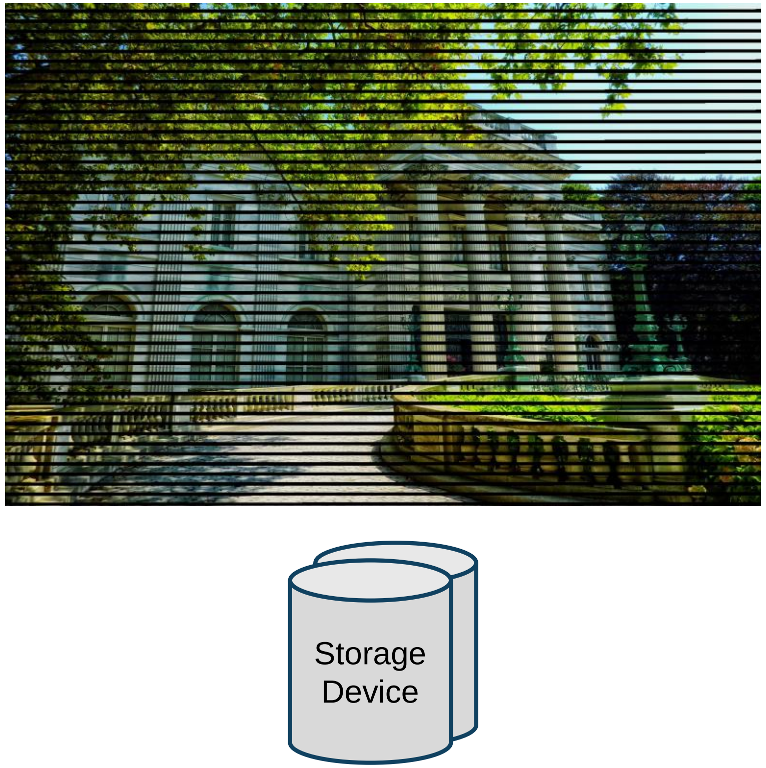
```
cp: error reading /mnt/btrfs/img00041.bmp: Input/output error
```

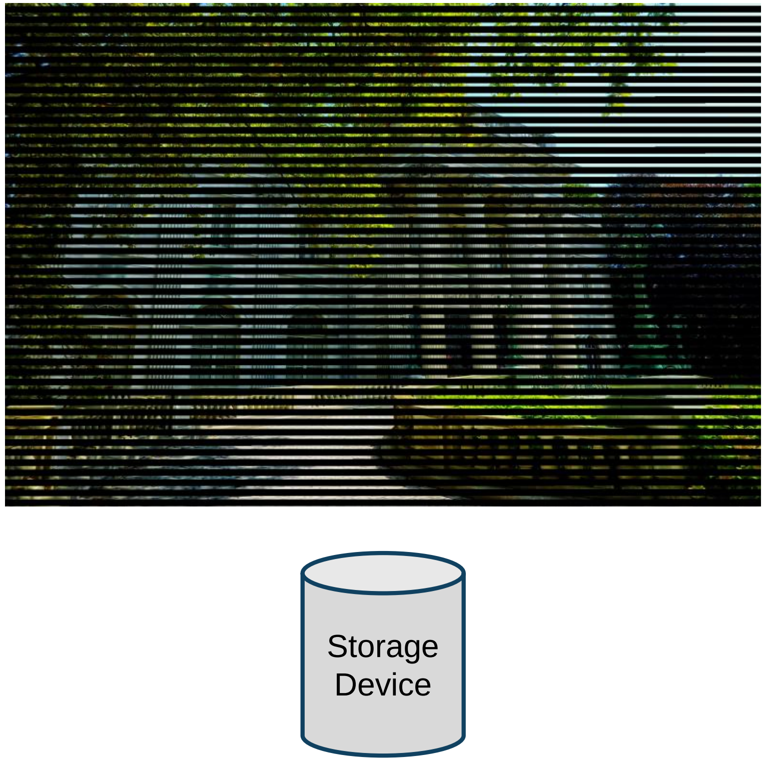
Forensic Analysis of BTRFS

Missing Disks

- Direct access to a pool makes it possible to pad missing data







Summary

- Examined the correctness of our extended model for BTRFS
- Documented the internal mapping scheme of BTRFS
- Implemented support for multiple device BTRFS configurations
  - Extended an existing BTRFS implementation working for single disk configurations
- Performed a forensic analysis of BTRFS
  - Snapshots
  - Metadata based file recovery
  - Missing pool members

Thanks for your attention!

https://github.com/fkie-cad/sleuthkit

Fraunhofer FKIE