# Storage Efficient Backup of Virtual Machine Images

Artur Hulestki, SPbAU
(hatless.fox@gmail.com)

## Motivation

- Backups have to be done frequently to minimize change of data loss
- Virtualization technologies are widely used by cloud services
- Virtual disk consumes most of the storage

Goal:

Minimize storage used by virtual disk backup

## Existent Approaches

- Backup organization:
  differential and incremental backups
- Features of virtualization software:
  VMware CBT; backup compressing in VirtualBox, etc.
- Proprietary Software:
  Veam Backup & Replication, EMC Avamar,
  Acronis Backup & Recovery, etc.

## Back to Problem...

- Cloud services provide many similar instances of virtual machines
- Frequent backups have to be performed for all these instances
- It is not common for the VM instance to use entire virtual disk space

## Unused Block Compression

Analysis of data stored on disk may lead better compression.

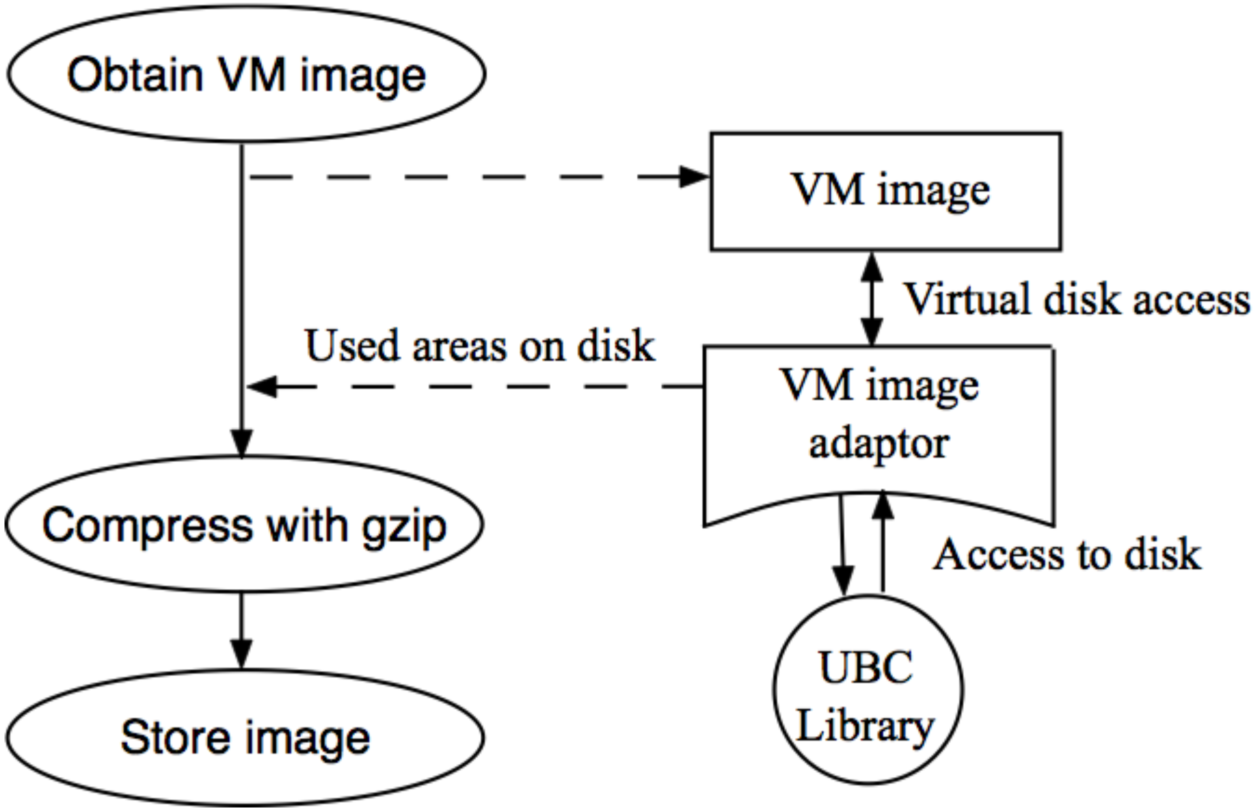
Idea: Want to analyze underlying File System to determine which blocks are actually used and store only them.

Drawback: Unused blocks are wiped.

# Plan

1. UBC adoption
2. Ext parsing
3. NTFS parsing
4. Incremental backups

## UBC adoption



## UBC library summary

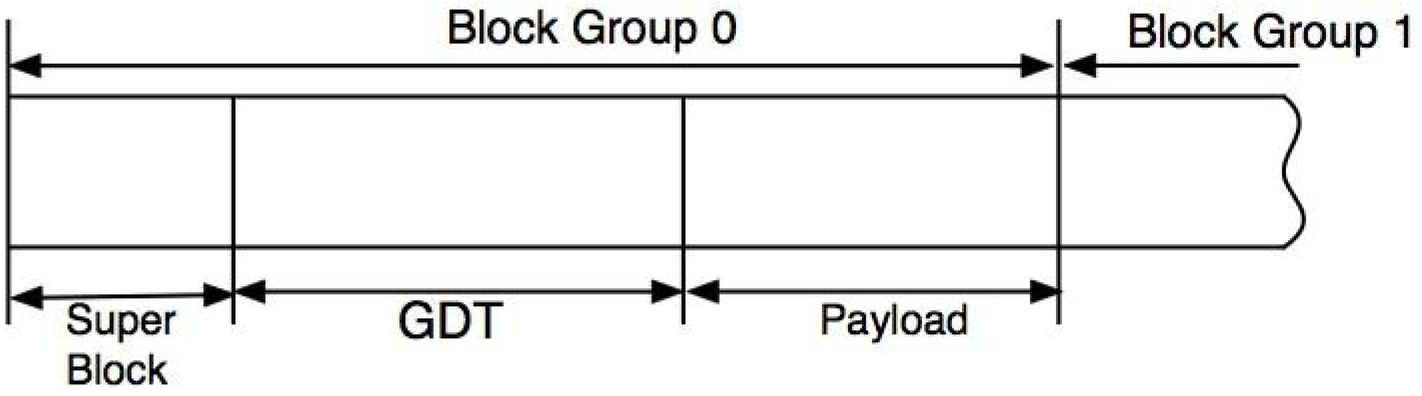
Supported File Systems:

- Ext[3, 4]
- NTFS

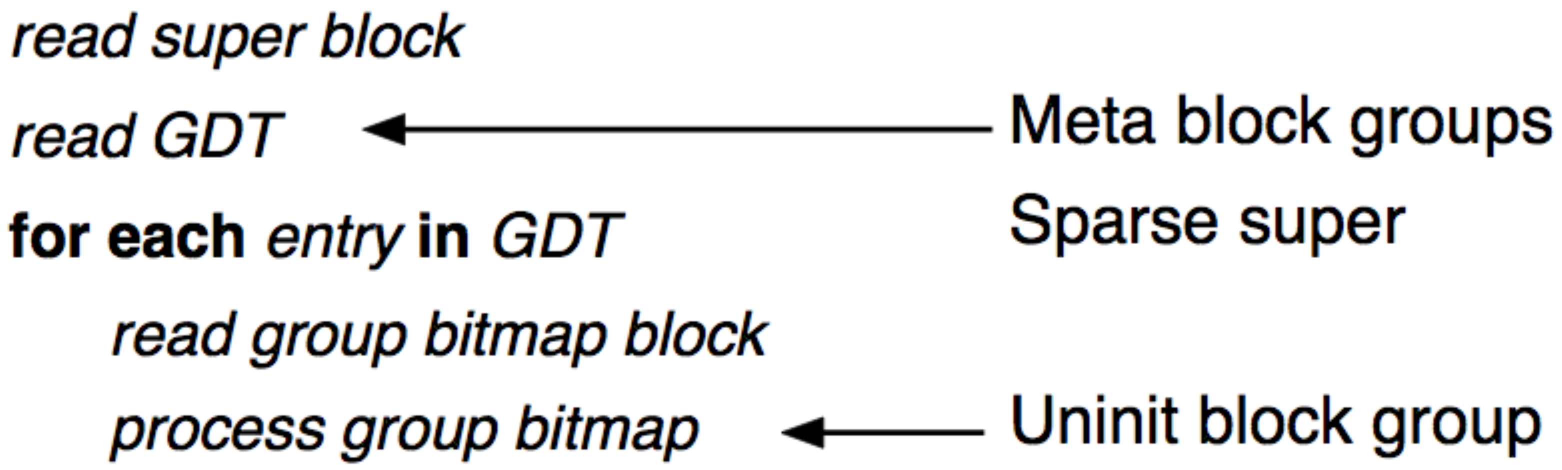
Assumption: FS structures are in consistent state

# Ext File System

- Storage is split on block groups
- Block group meta data is stored in Global Descriptor Table
- Used blocks info is stored in bitmap per-group



# Ext Parsing

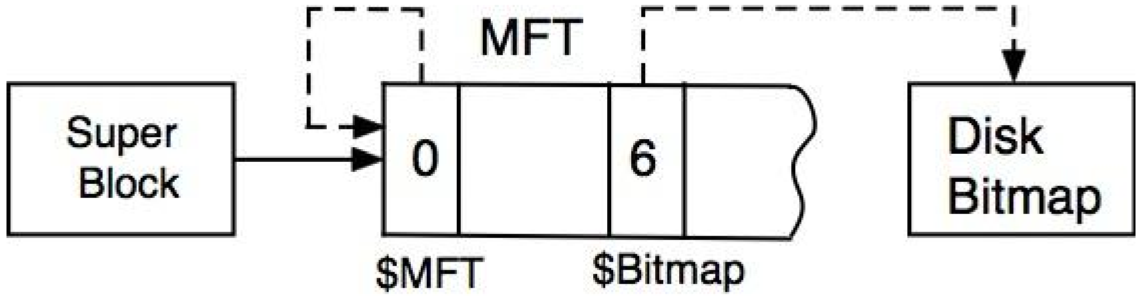


```
read super block
read GDT
for each entry in GDT
    read group bitmap block
    process group bitmap
```

Unsupported modes: 64-bit

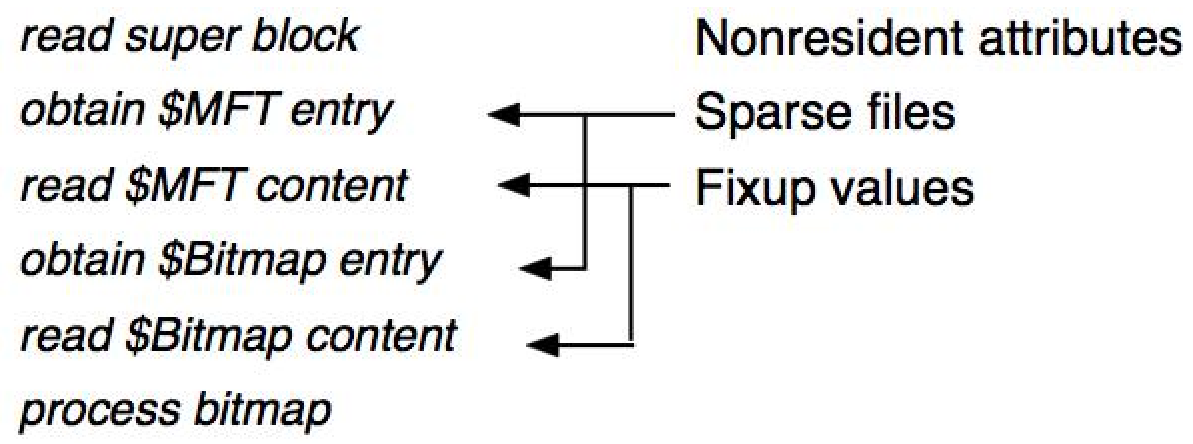
# NTFS

- Everything is stored in “files”
- Master File Table contains file descriptors
- File meta-data is stored in descriptors using attributes
- Usage bitmap of entire disk is stored in special file



A

# NTFS Parsing



Looks simple but ...

## NTFS ambiguity

Ambiguities related to $MFT reading:

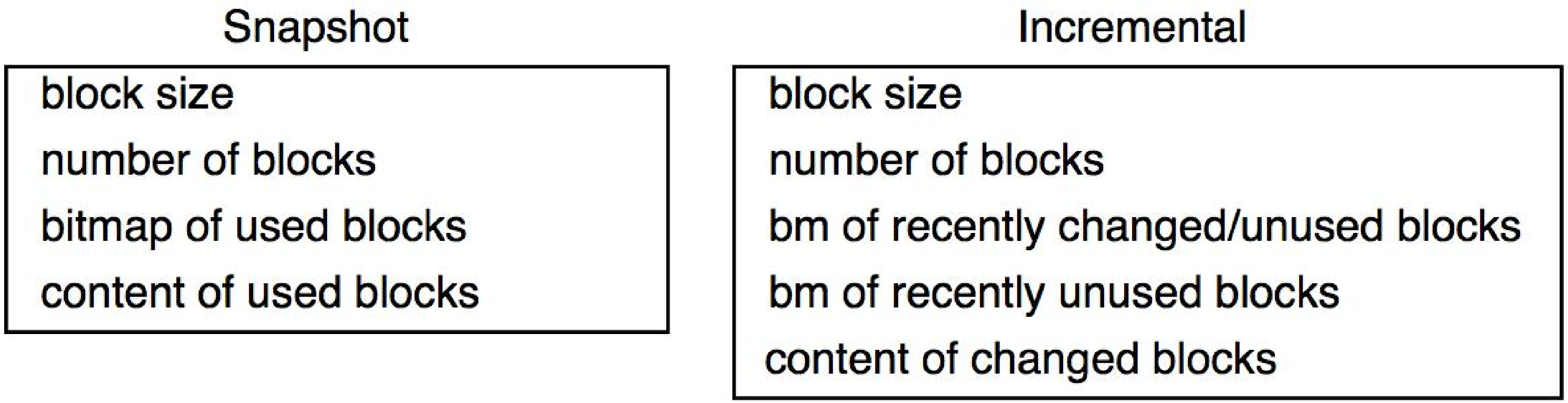
- Compressed core file system files
- Custom attribute types (value of $DATA)
- Multi-record MFT entries (fragmented $MFT)

Reason: No open specification/implementation from Microsoft

C

## Incremental Backups

## Backups



Snapshot

- block size
- number of blocks
- bitmap of used blocks
- content of used blocks

Incremental

- block size
- number of blocks
- bm of recently changed/unused blocks
- bm of recently unused blocks
- content of changed blocks

Naive restore (patching approach):

find the most recent snapshot

apply all later incremental backups

D

# Sketch of Restore Algorithm

```
init unrestored bitmap (as filled)          /* unrstd - blocks that */
init target disk image                       /* have to be restored */
while not unrestored bitmap empty
        get next backup                      /* bu from LIFO container */
        if incremental backup
                update changed blocks        /* unrstd &= !bu.ch_unusd */
                update unrestored bitmap
        if snapshot backup
                update blocks from snapshot
                clear unrestored backup
```

**Core Ideas**

- Incremental backups are applied in reverse order
- Progress is tracked with a bitmap

Benefits:

Every used block is written only once

(“patching” approach doesn’t provide such guarantee)

## Summary

Implemented:

- Library (diskube) that allows to incorporate UBC approach
- Scaffolding for testing: command line tool, tiny DSL for disk image creation and configuration

Future:

- Add support for other file systems (e.g. XFS)
- Implement PoC adaptor (e.g. for VirtualBox images)
- Implement and measure proposed algorithm for restore

Q&A