TECHNISCHE UNIVERSITÄT DRESDEN

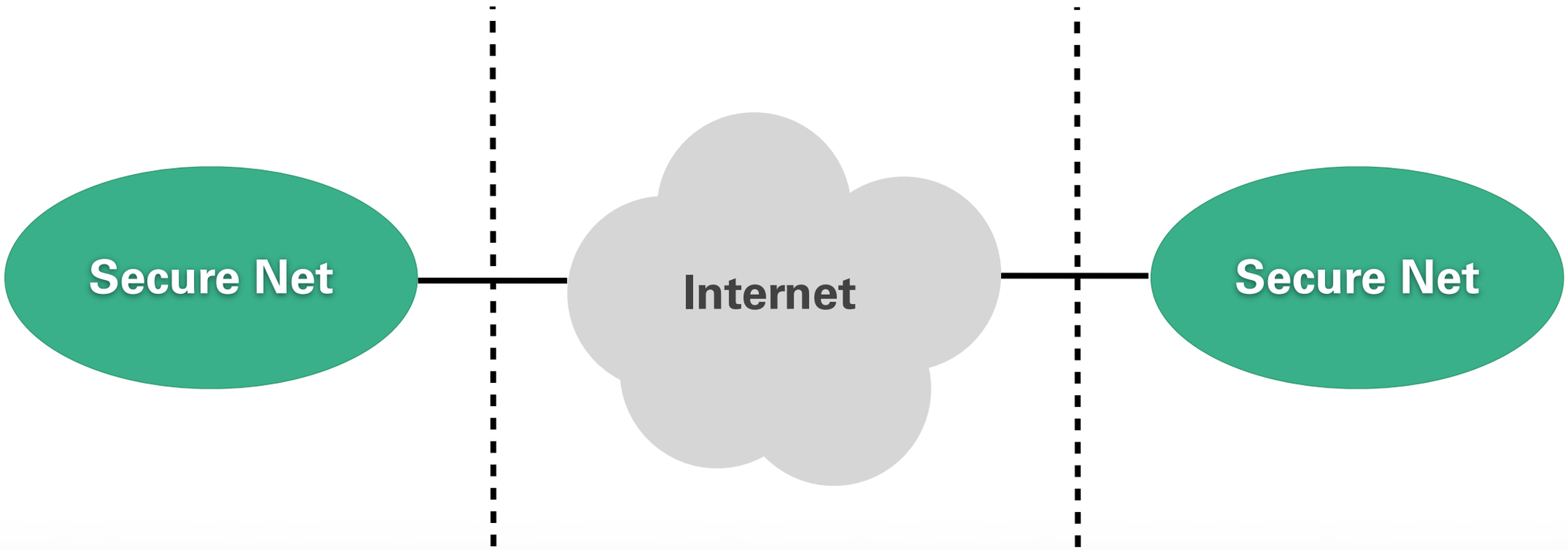
Department of Computer Science Institute of Systems Architecture, Operating Systems Group

jVPFS: Adding Robustness to a Secure Stacked File System with Untrusted Local Storage Components

Carsten Weinhold, Hermann Härtig

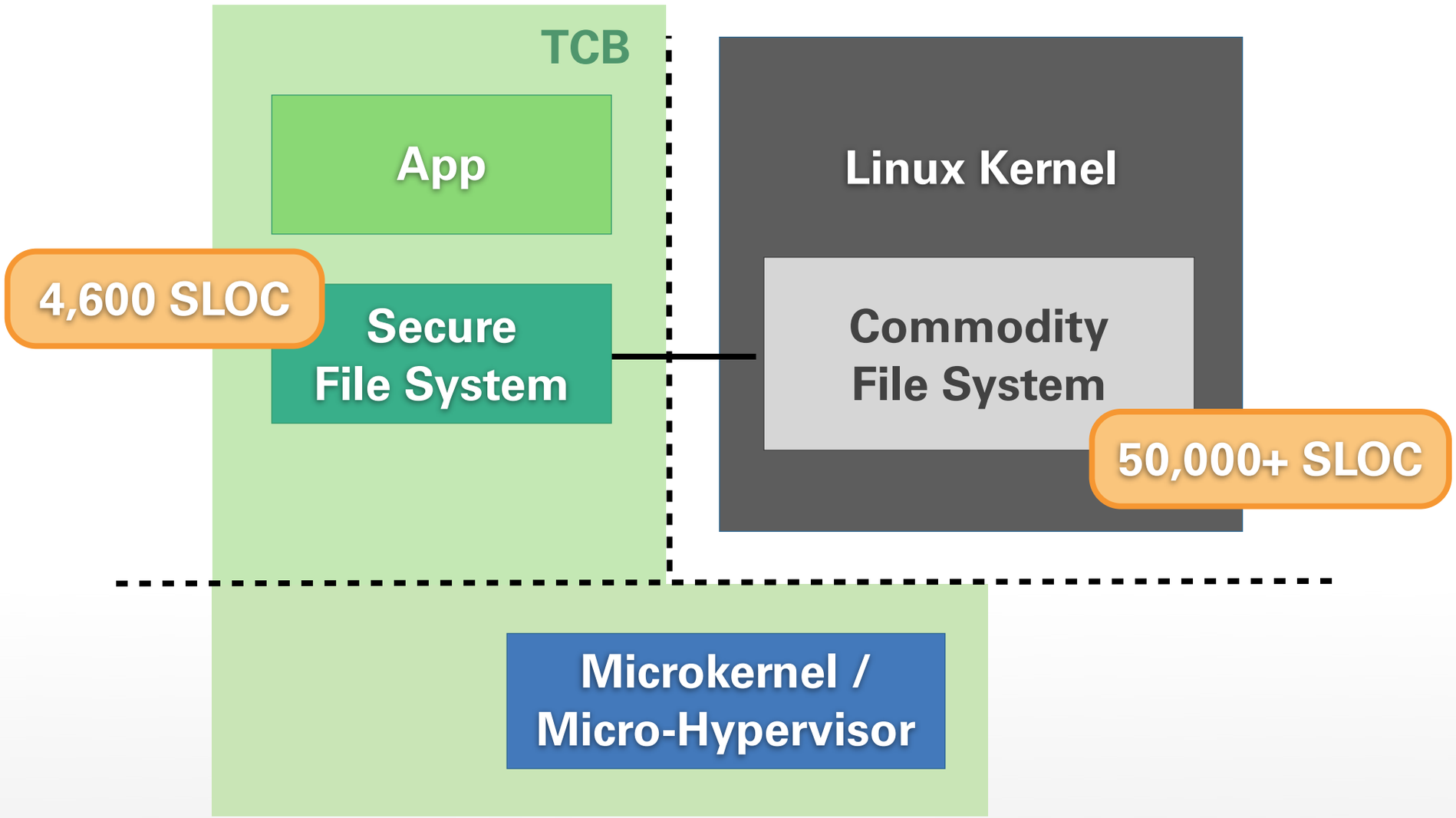
INTRODUCTION

VPN: Confidentiality, Integrity, ~~Availability~~



INTRODUCTION

VPFS: Confidentiality, Integrity, ~~Availability~~



[1] Weinhold, Härtig: *„VPFS: Building a Virtual Private File System With a Small Trusted Computing Base“*, EuroSys'08

INTRODUCTION

VPFS: Confidentiality, Integrity, ~~Availability~~

Secure File System

Secure File System Proxy

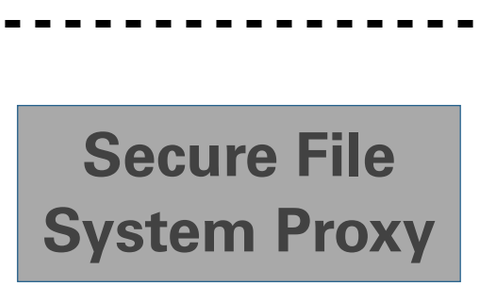
Commodity File System

[1] Weinhold, Härtig: *„VPFS: Building a Virtual Private File System With a Small Trusted Computing Base“*, EuroSys'08

OUTLINE

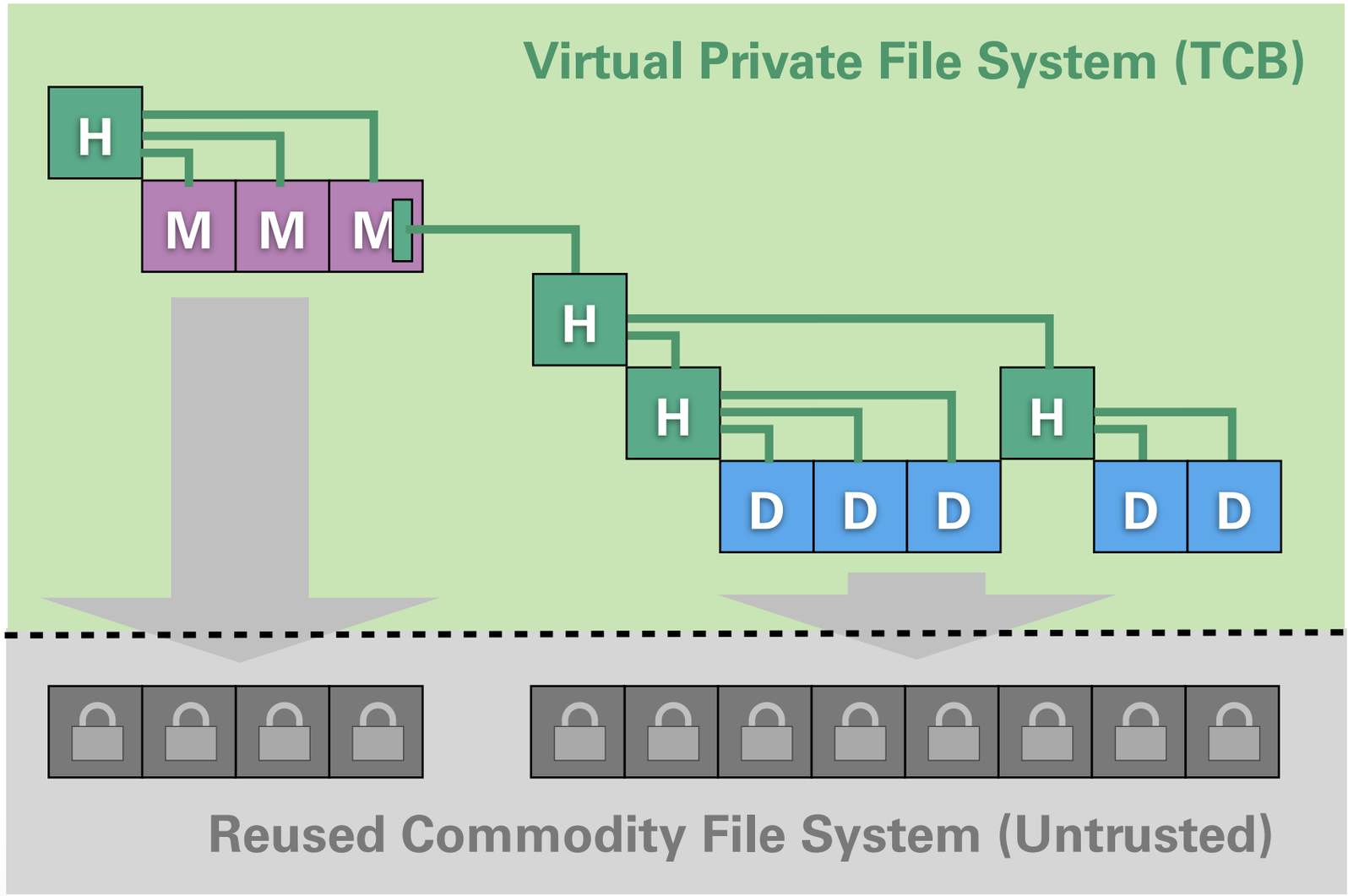
- Introduction
- VPFS: **V**irtual **P**rivate **F**ile **S**ystem
- **j**VPFS: Adding robustness securely
- Evaluation
- Lessons learned


Secure
File System


Secure File
System Proxy

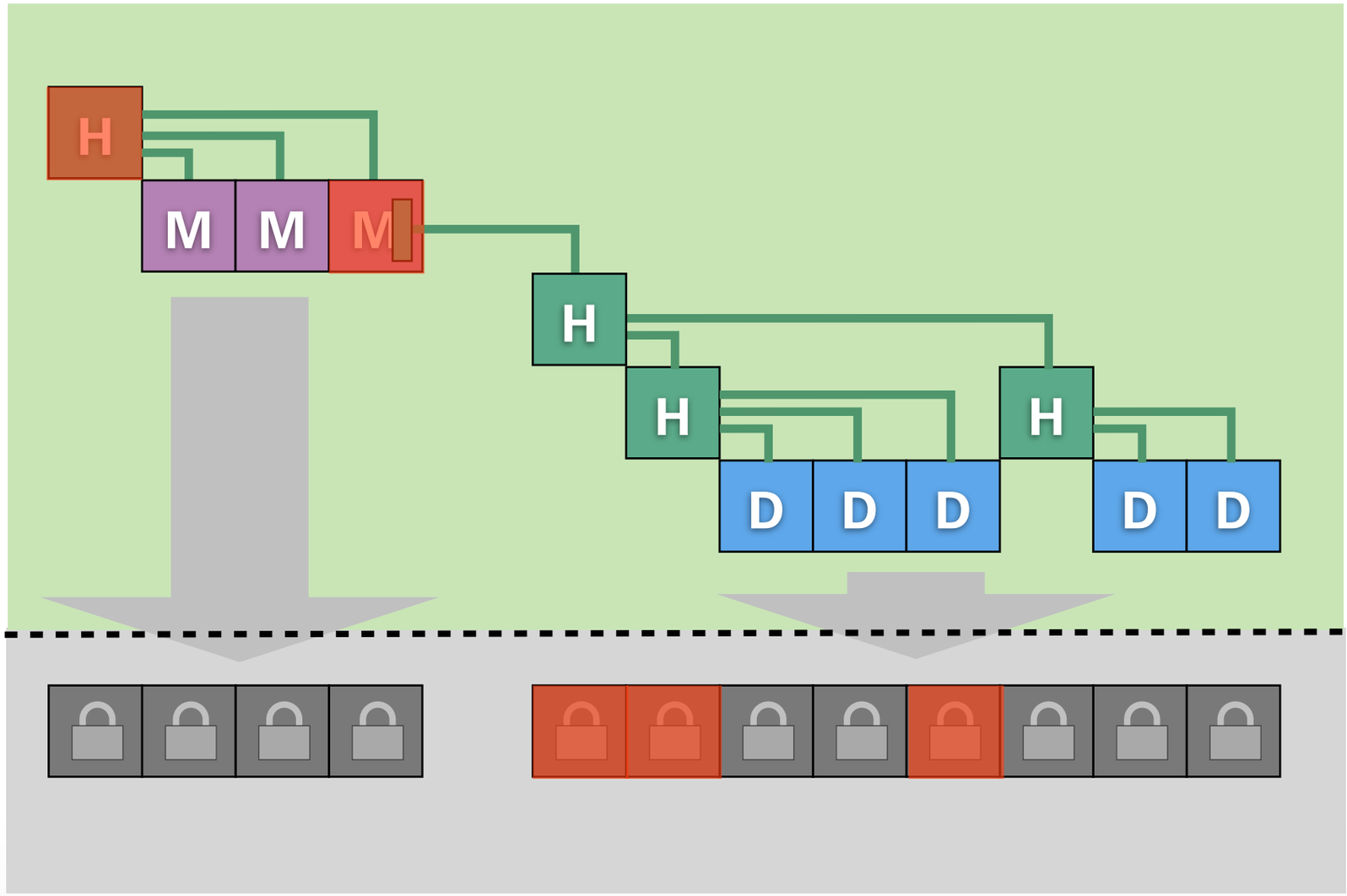

Commodity
File System

FILES & FILE CONTAINERS



- Encrypted files in commodity file system
- Merkle **hash tree** to detect tampering

UPDATING HASH TREE



- High overhead: many writes + crypto ops
- Hash tree updates must be atomic

CONSISTENCY PROBLEM

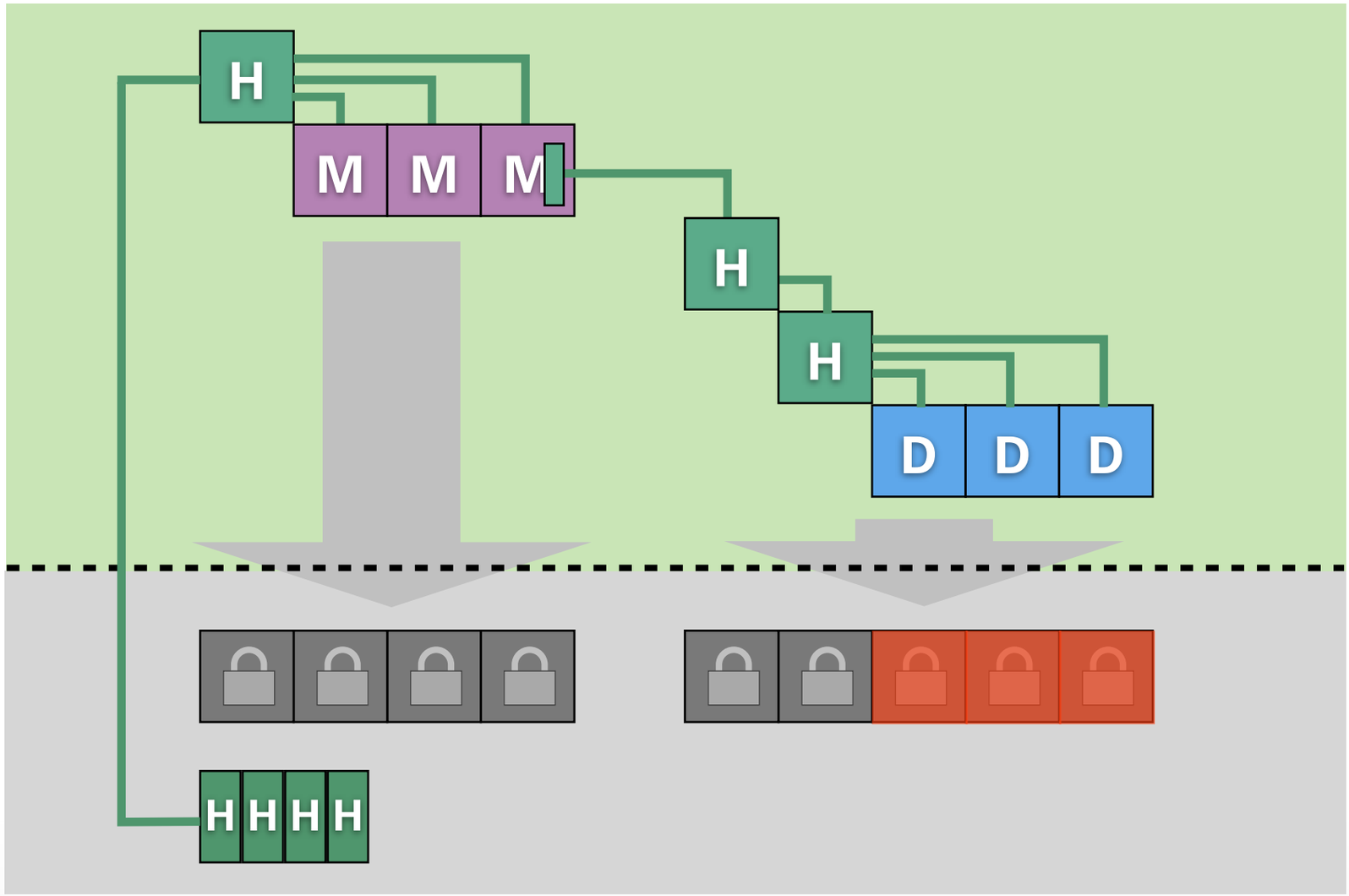
- File-system consistency is complex problem:
  - Correct implementation is difficult[2,3]
  - Bugs often in corner cases, error checking[4]
  - Widely used file systems affected, too
- **Goal:** keep complexity out of the TCB

[2] Yang et al.: *„Using Model Checking to Find Serious File System Errors“*, ACM TOCS Vol.24 Issue 4, 2006

[3] Prabhakaran et al.: *„Model-Based Failure Analysis of Journaling File Systems“*, DSN'05

[4] Gunawi et al.: *„EIO: Error Handling is Occasionally Correct“*, FAST'08

HASH TREE + JOURNAL



- Record new hash sums in journal
- Recovery: valid hash either in tree or journal

BLOCK WRITE BACK

Security critical

- Calculate hash + encrypt block
- Put ciphertext + hash into shared ring buffer

- Do ordered write to legacy file system
  - Append hash sums to journal
  - Write blocks afterwards
- Optimizations

Critical only for Availability

JOURNALING METADATA

- **Approach:** log operations, not blocks
  - Code reuse: replay during recovery via API
  - Simple dependency tracking
  - Non-intrusive implementation
- **Dependencies:**
  - *New files:* inode, name, parent dir
  - *Updated files:* file size, hash sums
  - *Unlinked / moved files:* name, parent dir

TRACKING NEW FILES

file table

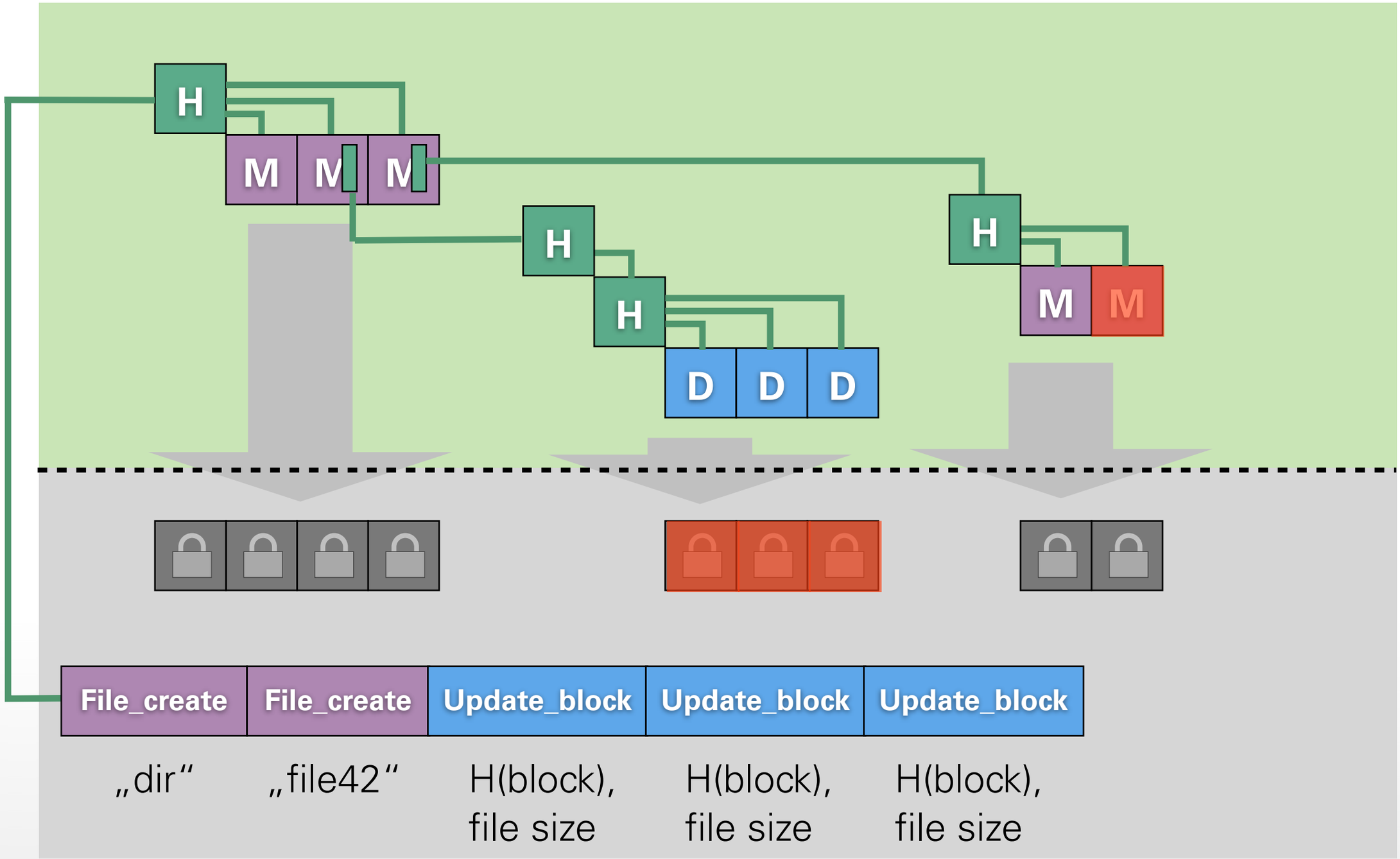
	fd
0	File*
1	
2	File*
3	File*
4	File*
...	
511	

new-file table

	p_fd	p_inode	name
0			
1			
2	1	0	„dir“
3	2	0	„file23“
4	2	113	„file42“
...			
511			

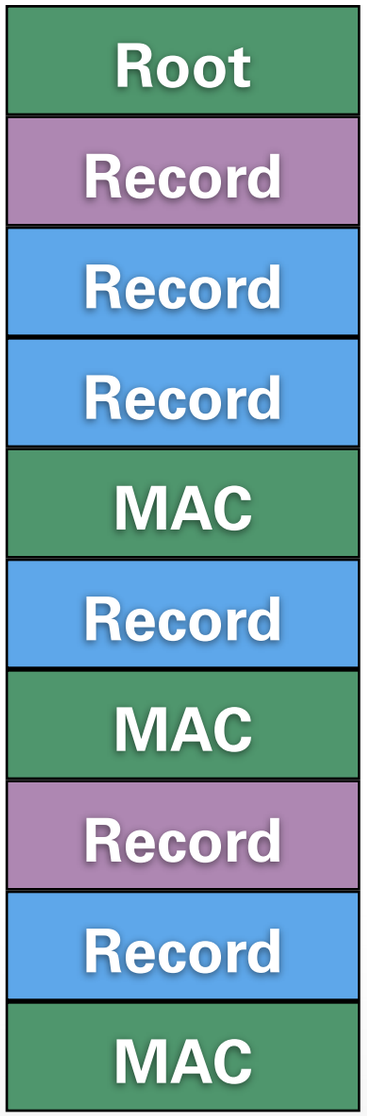
Pathname elements to log: .../**dir/file42**

JOURNAL CONTENT



JOURNAL SECURITY

- **Confidentiality:**
  - Filenames, inodes, etc.: encrypted
  - Block offset, type: plaintext
- **Tamper detection:**
  - Anchor of journal in „Sealed Memory“
  - Journal is continuously MAC’d
- **Record groups:**
  - All records between two MACs



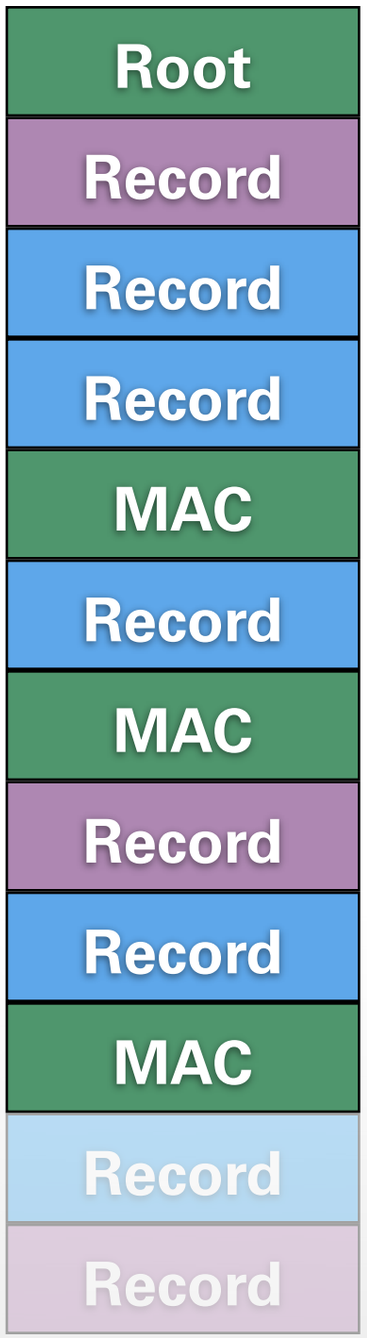
RECOVERY PROCEDURE

Critical only for Availability

1. Recover legacy file system
2. Find complete record groups in journal
3. Restore pre-journal versions of metadata blocks

4. Read root info (aka superblock)
5. Replay:
   a) Get complete record groups
   b) Check integrity + decrypt
   c) Re-execute operations:
      open(), unlink(), ...
   d) Repeat from a)

Security critical



COOPERATION

- **Extensive Reuse:**
  - Complete commodity file system
  - Existing consistency primitives:
    - Journaling, copy-on-write, ...
    - Write ordering, snapshots
- **More details in paper:**
  - Checkpoints + journal truncation
  - Flushing metadata blocks

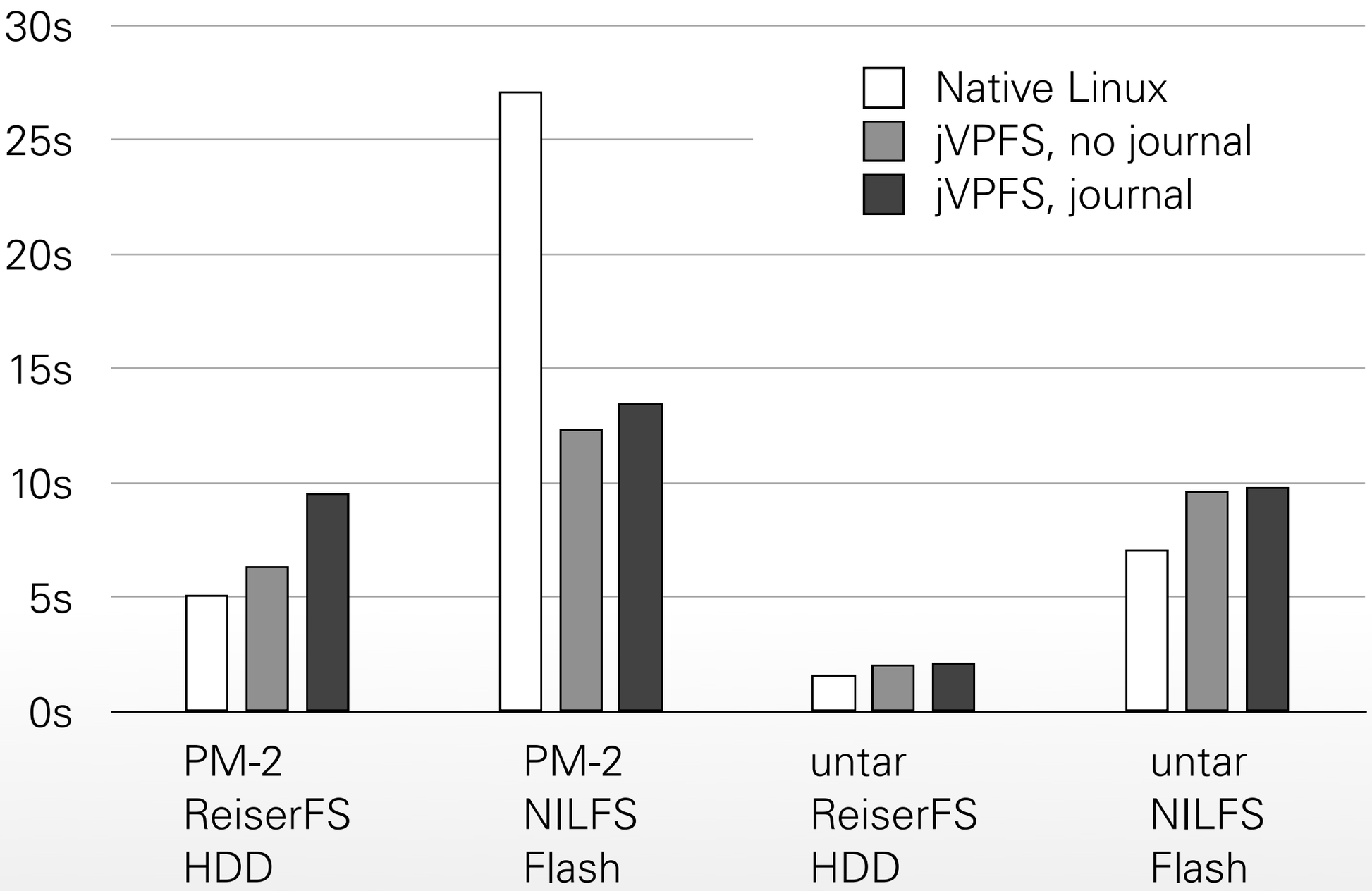
SOURCE COMPLEXITY

		SLOC	
Subsystem	ReiserFS	Ext4	jVPFS
Journal + replay	~3,200	~5,000	325
Basic persistency	16,500+	24,000+	404
Core functionality			2,444
Crypto algorithms			667

TESTING RECOVERY

- **Testcase for recovery:**
  - Unpack tar archive (3,000+ files, 70 MB)
  - Power-cycle machine, interrupt write back
  - Recover jVPFS + try to open + read all files:
    - *NILFS+Flash:* successful
    - *ReiserFS+HDD:* successful
- **Example run:** replay **1.2 MB** journal in **5.1s**
- **Restored: 2,710** files, **40 MB** user data

PERFORMANCE



LESSONS LEARNED

- **jVPFS:** Less than **350 SLOC** in TCB to **make secure** file system **robust**
- Security-critical core for journaling + replay:
  - Log API-level operations, replay via API
  - Code reuse, simple dependency tracking
- Move complexity to untrusted file system
- Reuse existing consistency primitives