A filesystem attack vector

for backdoors, rowhammer-like attacks, and more.

Anil Kurmus

with Nikolas Ioannou, Matthias Neugschwandtner, Nikolaos Papandreou and Thomas Parnell

IBM Research - Zurich

This talk

Introduces filesystem-tricks that can be used in two attack scenarios (on ext3):

1. Persistence without tampering binaries/config
2. Privilege escalation assuming rowhammer-like attacks on storage media

Outline

1. Indirect block manipulation on ext3
2. Persistence backdoor
3. Privesc for rowhammer-like attacks

Primer on ext3

… and similar indirect-block-based filesystems

Inodes

Inode Table

inode 1	…	inode n	…
		mode	
		owner	
		…	
		direct blocks	
		indirect blocks	

…

…

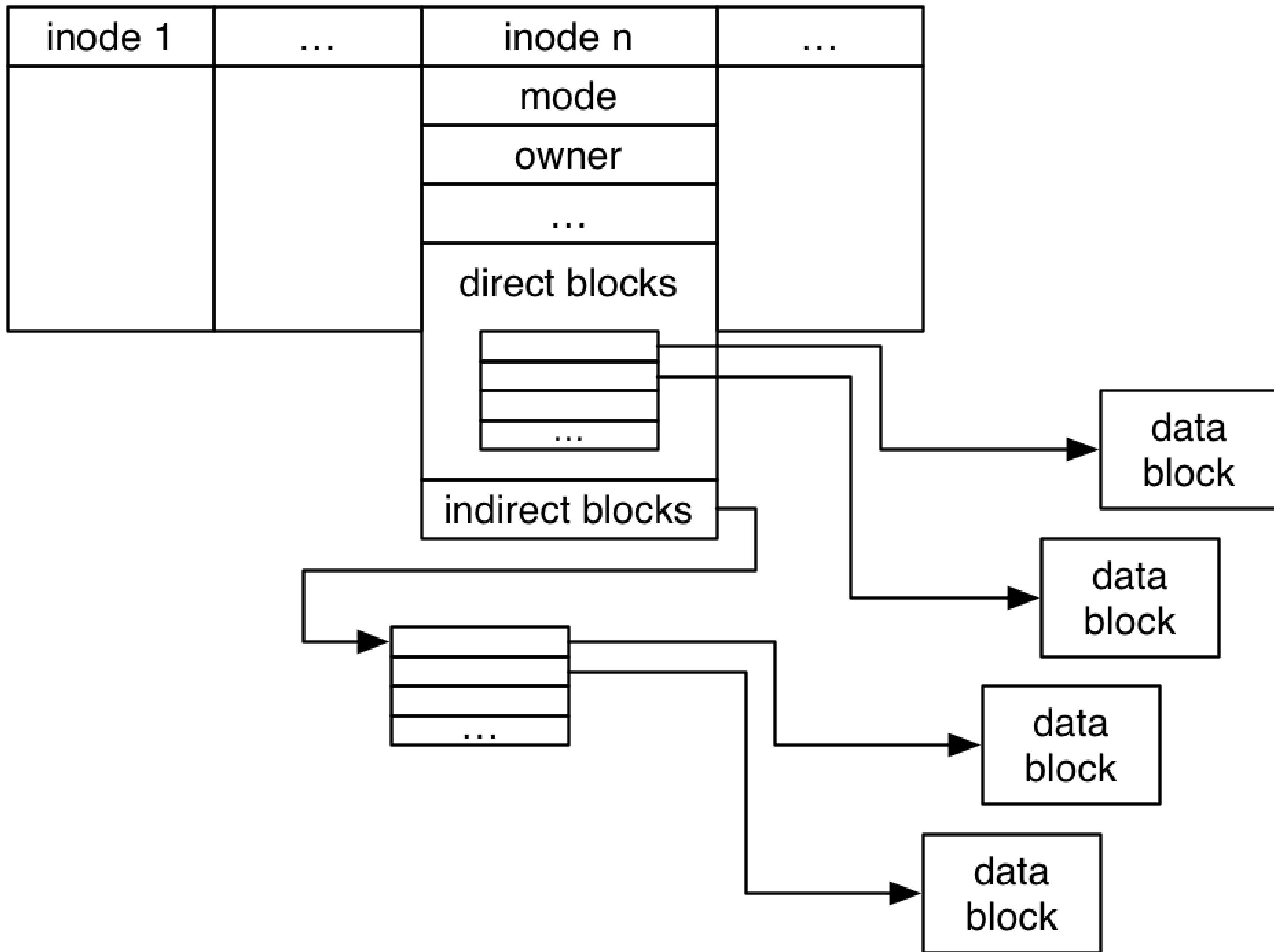
data block

data block

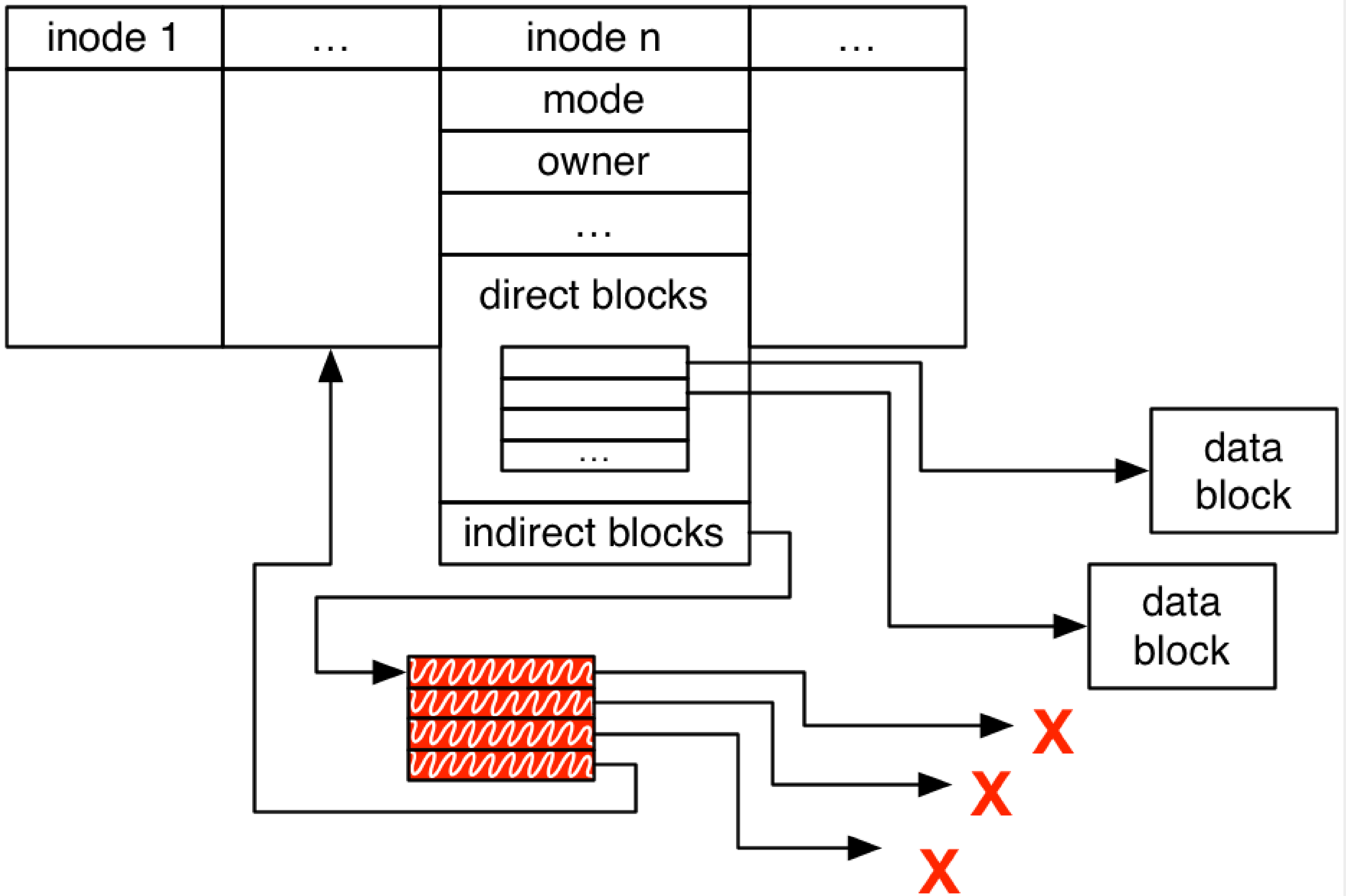
data block

data block

Indirect blocks



Indirect block manipulation



It's a pointer... we can and should corrupt it!

Application #1

Persistence backdoor

Implant a backdoor to persist root access across reboots *without* modifying system files, binaries, config files.

Threat model

Assume attacker has raw disk access (root access)

Idea

- Create "backdoor" file
- Update its inode: indirect block points to inode table
- Persistence achieved!

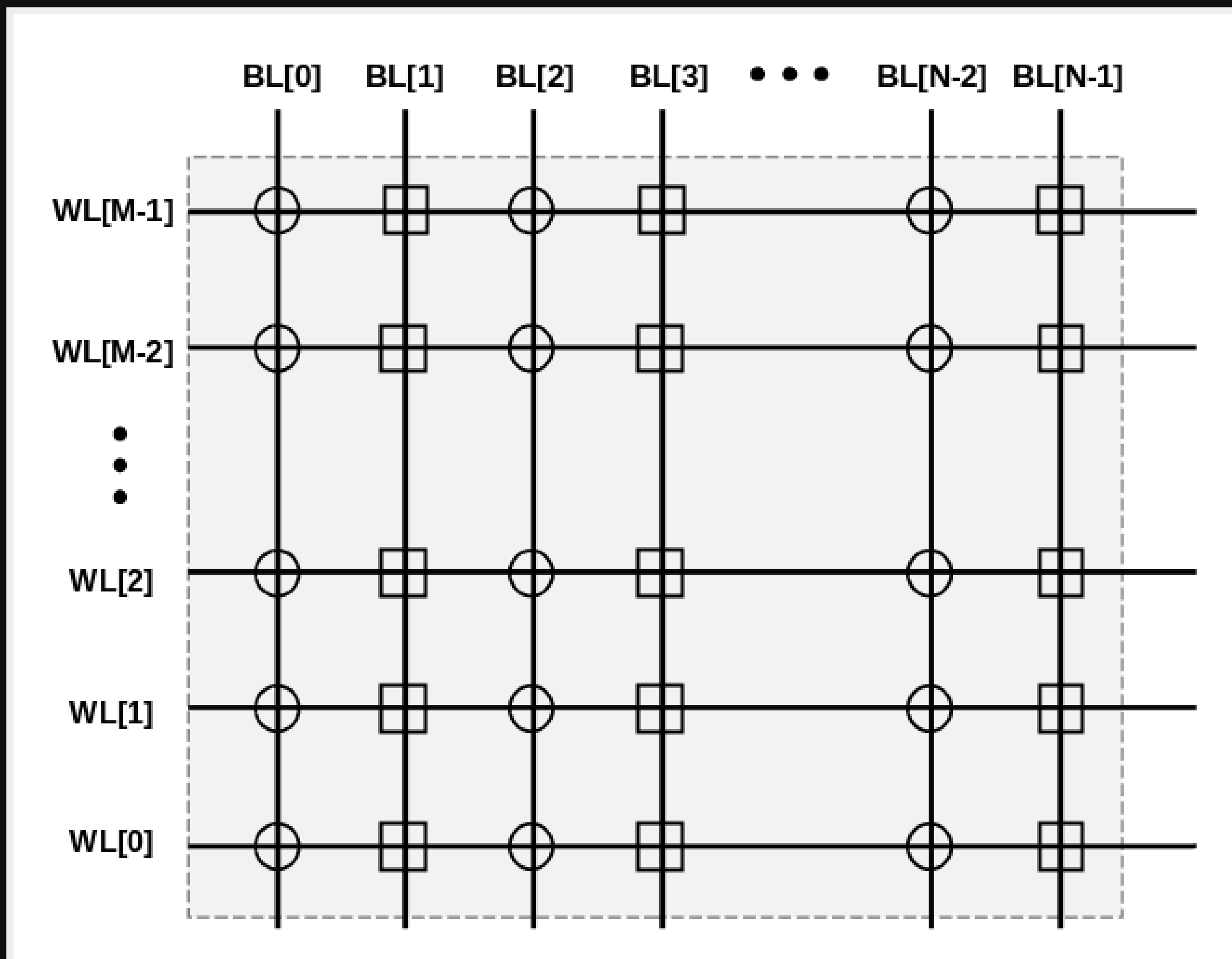
On reboot

- Write into the "backdoor" file
- Updates inode of another file: for example a shell
- Get root by creating a suid-root shell
- Done!

Live demo

Application #2

Flash primer

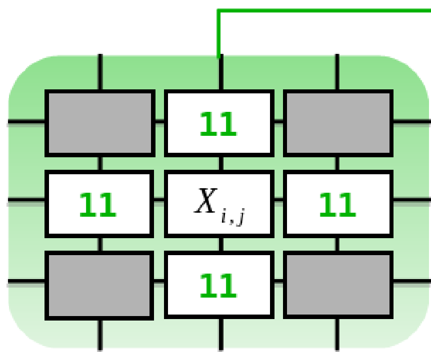


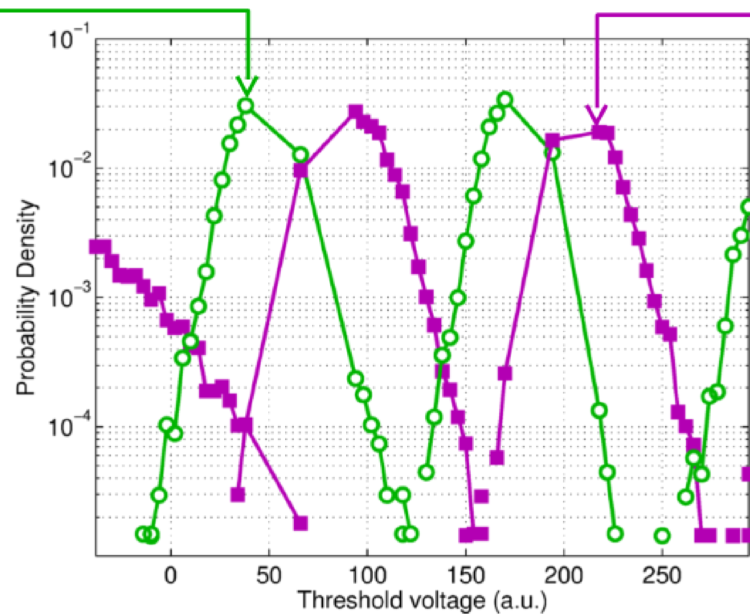
Flash weaknesses

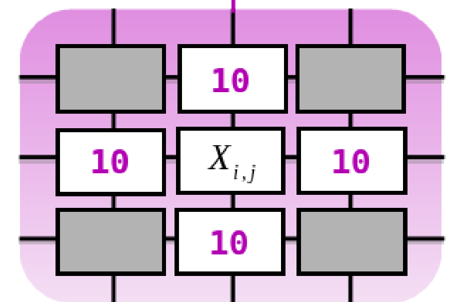
- Program/Erase wear
- Charge loss over time
- Cell-to-cell interference
- Read disturb

All were demonstrated, characterized

Cell-to-cell interference







Mitigations

- Scrambler
- Block allocation/wear leveling
- Error correcting codes (ECC)

Implemented in SSDs

Flash storage layers

1. Flash chip
2. Flash controller
3. SSD controller
4. OS (filesystem/driver)
5. User

A path to rowhammer-like attacks on flash

1. Flash chip: cell-to-cell interference.
2. Flash controller: scrambler and ECC bypass.
3. SSD Controller: wear leveling and block placement algorithm.
4. OS: filesystem caching and error detection bypass.
5. User: privilege escalation payload.

Prior work

1. **[Flash chip: cell-to-cell interference.]**
2. Flash controller: **[scrambler]** and ECC bypass.
3. SSD Controller: wear leveling and block placement algorithm.
4. OS: filesystem caching and error detection bypass.
5. User: privilege escalation payload.

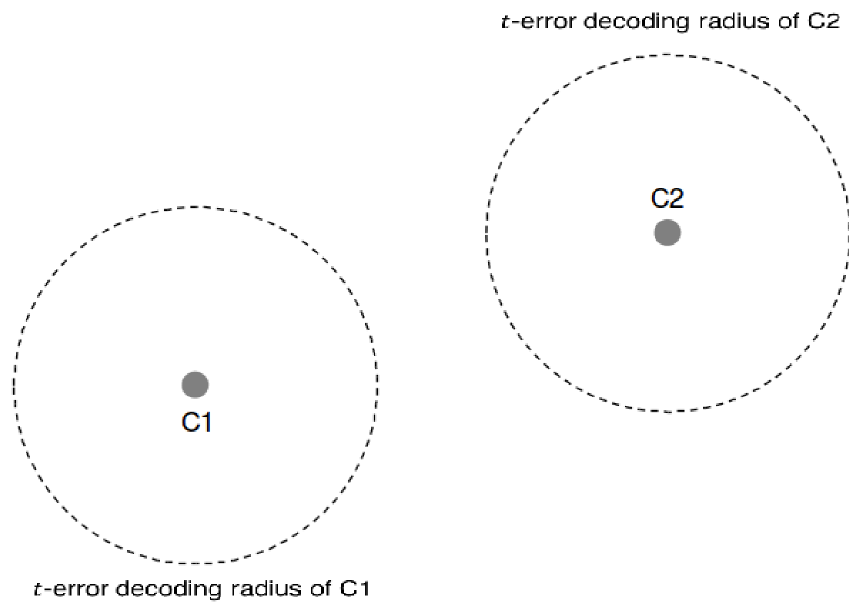
Our WOOT paper

1. Flash chip: cell-to-cell interference.
2. Flash controller: scrambler and ECC bypass.
3. SSD Controller: wear leveling and block placement algorithm.
4. **OS: filesystem caching and error detection bypass.**
5. **User: privilege escalation payload.**

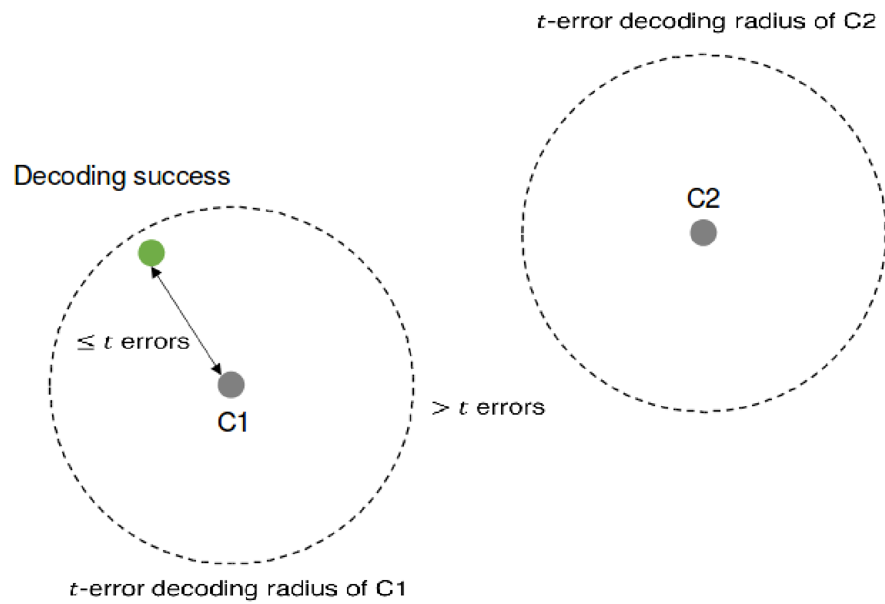
Flash ECC

- Long codewords (e.g., >1KB)
- High correction capability (e.g., >50 bits)

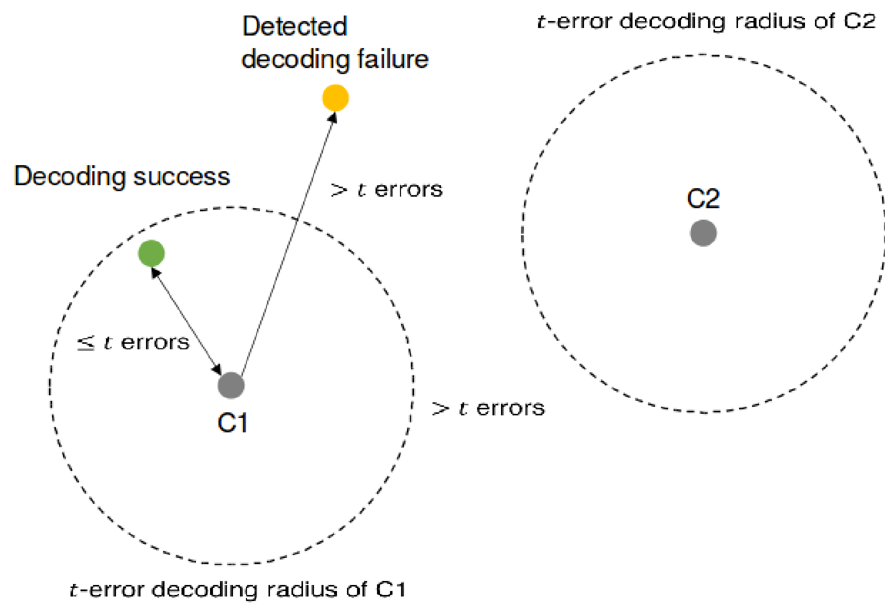
Codewords



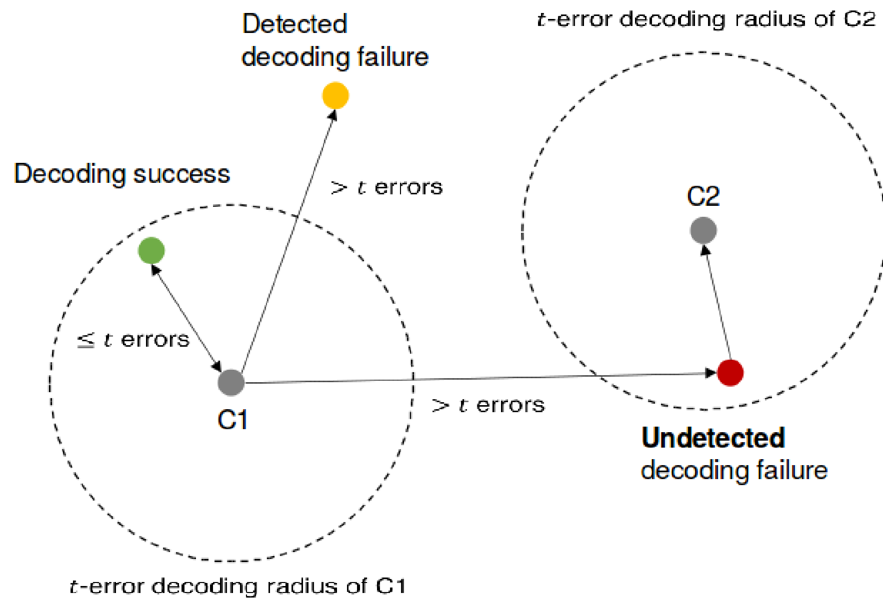
Codewords



Codewords



Codewords



Extremely difficult to achieve this!

The filesystem attack

Assumes:

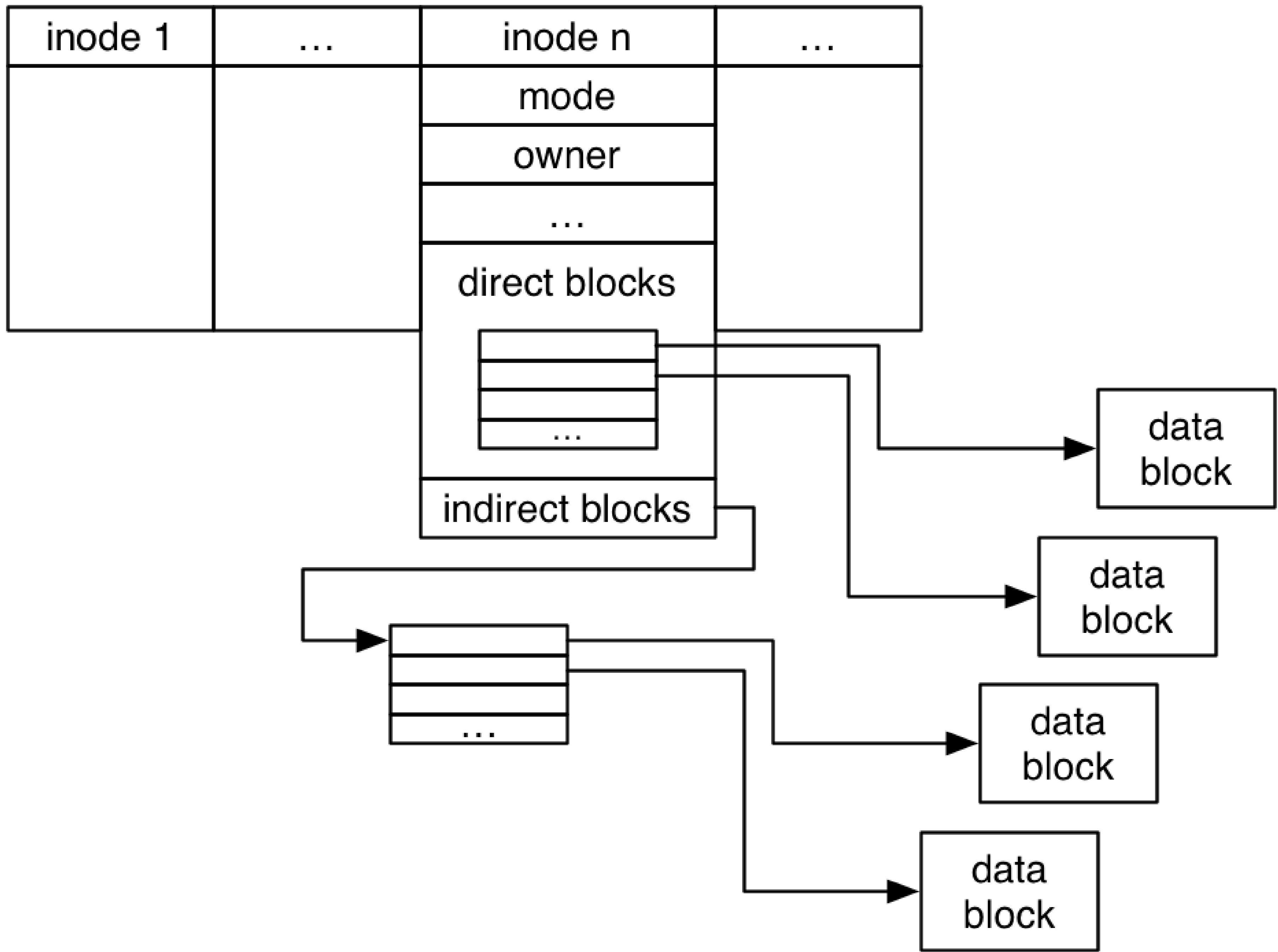
- Attacker can corrupt chosen block
- random contents (weaker)
- ext3 filesystem

Filesystem attack

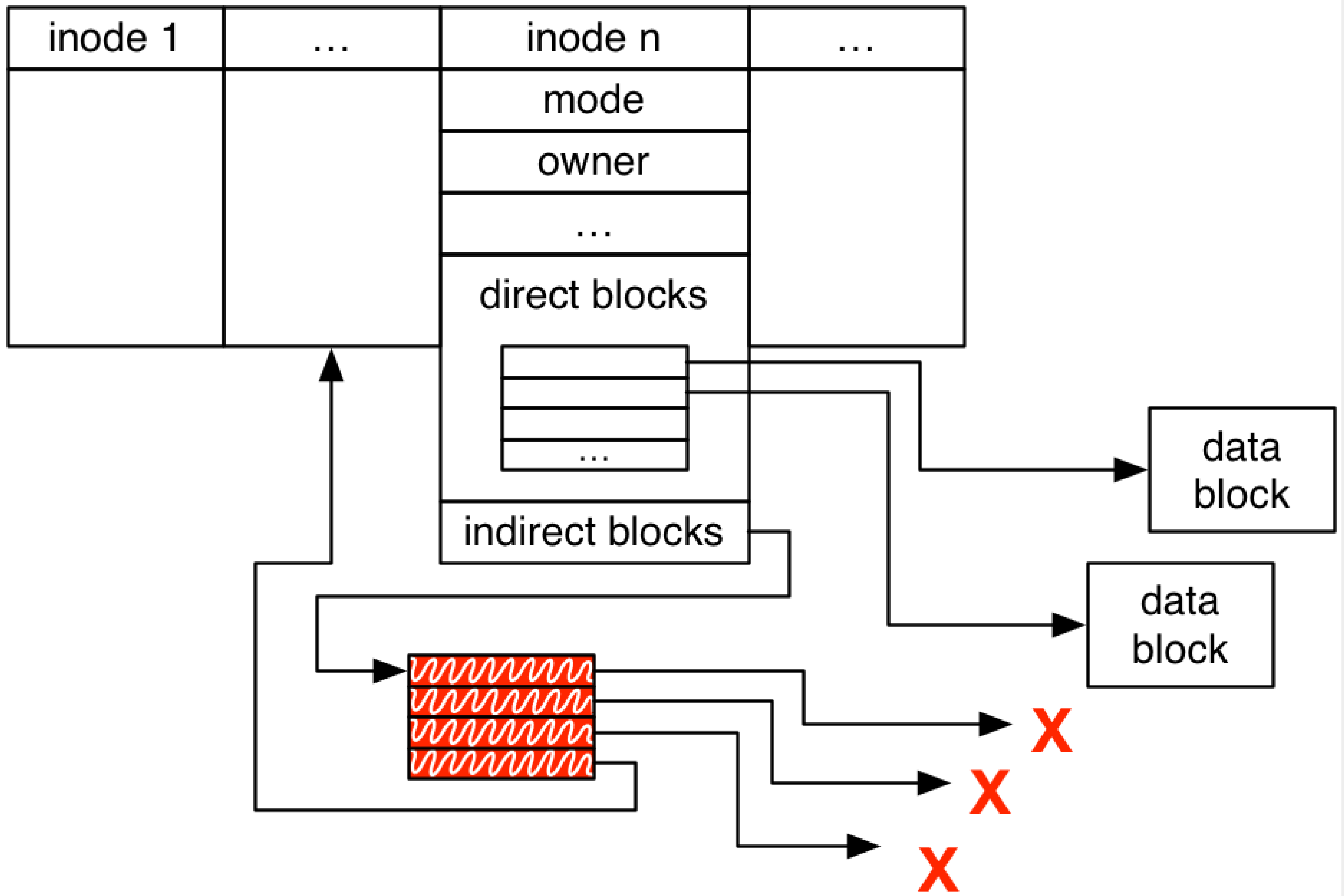
Main idea:

- Cause corruption of indirect block
- Good chance one pointer points to inode table
- Overwrite inode to set SUID-bit root
- Elevate by executing SUID-root shell

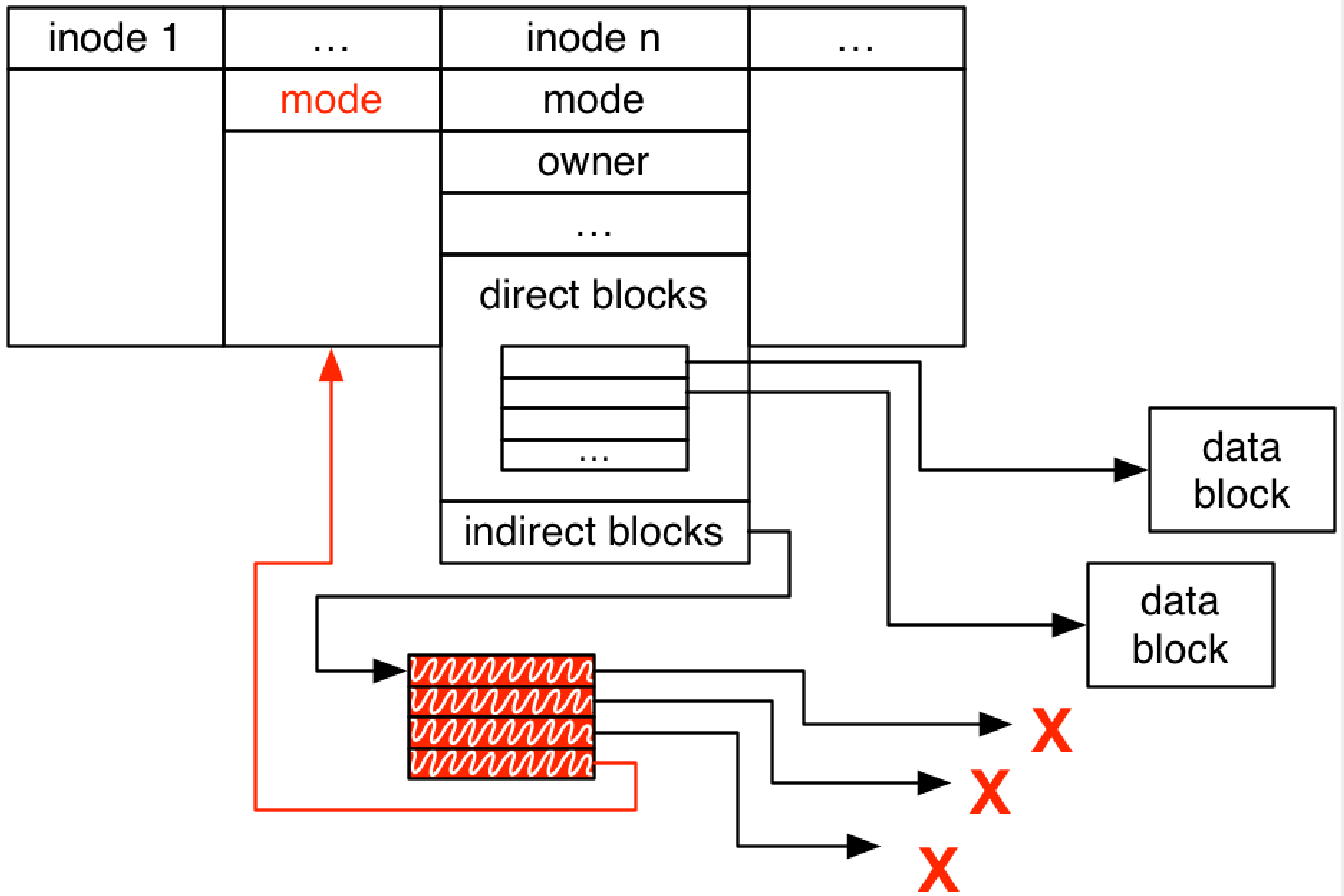
Inodes, indirect blocks



Indirect block corruption



Overwrite inode entries



Attack demo

https://www.youtube.com/watch?v=Mnzp1p9Nvw0

Improved attack

- Uses double indirect block instead
- Achieves full filesystem read/write
- 99% success

Limitations & Mitigations

- unclear whether applicable beyond ext3
- data integrity checks prevent the attack (ZFS)
- more in the paper

Conclusion 1/2

- Random corruption of a chosen block implies privilege escalation
  - with high probability on ext3
- There is a path to rowhammer-like attacks on SSDs
  - but none demonstrated yet
- This is one piece of the puzzle: the filesystem part.

Conclusion 2/2

- The technique is nevertheless applicable in other domains:
  - persistence without modifying binaries/config files
  - active attacks against XTS encryption? (future work!)
- We can do neat attacks by manipulating fs pointers!