# HARDFS: Hardening HDFS with Selective and Lightweight Versioning

Thanh Do, Tyler Harter, Yingchao Liu, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau

THE UNIVERSITY of WISCONSIN MADISON

Haryadi S. Gunawi

THE UNIVERSITY OF CHICAGO

# Cloud Reliability

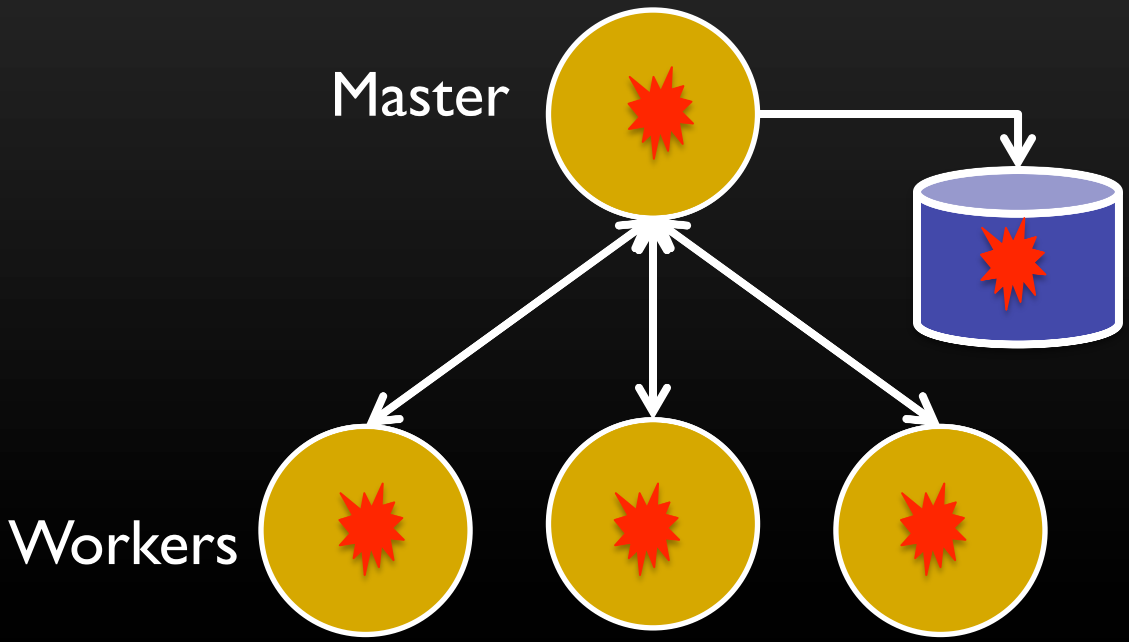
- Cloud systems
  - Complex software
  - Thousands of commodity machines
  - *"Rare failures become frequent"* [Hamilton]
- Failure detection and recovery
  - "… *has to come from the software*" [Dean]
  - "… *must be a first-class operation*" [Ramakrishnan et al.]

# Fail-stop failures

- Machine crashes, disk failures
- Pretty much handled
- Current systems have sophisticated crash-recovery machineries
  - Data replication
  - Logging
  - Fail-over

# Fail-silent failures

- Exhibits incorrect behaviors instead of crashing
- Caused by memory corruption or software bugs
- Crash recovery is useless if fault can spread



# Fail-silent failure headlines

GIGAOM

HOME APPLE CLEANTECH CLOUD DATA EUROPE MOBILE VIDEO

Jul 20, 2008 - 7:46PM PT

## S3 Outage Highlights Fragility of Web Services

Home ▸ .NET ▸ Azure ▸ News

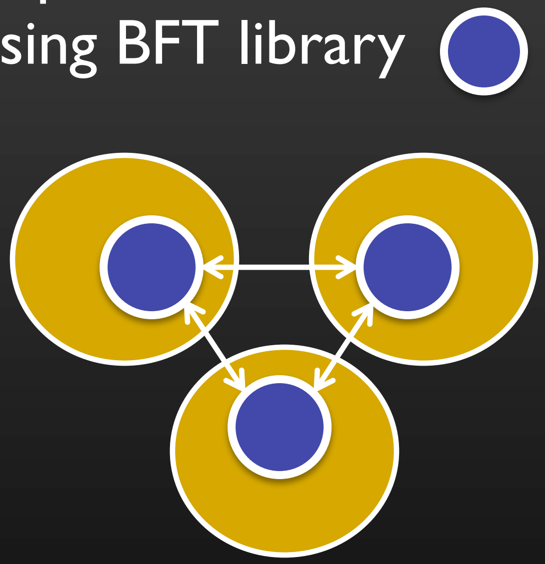
## Gmail data loss bug causes complete data loss, calls for tape backups

By Chris Alexander, published on 02 Mar 2011 | Filed in Security Azure    0 Comments    PDF

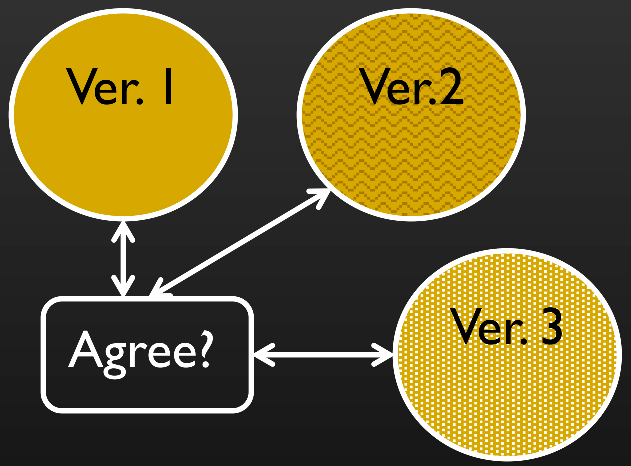
+1 Tweet Like 1    Share

# Current approaches

Replicated state machine using BFT library

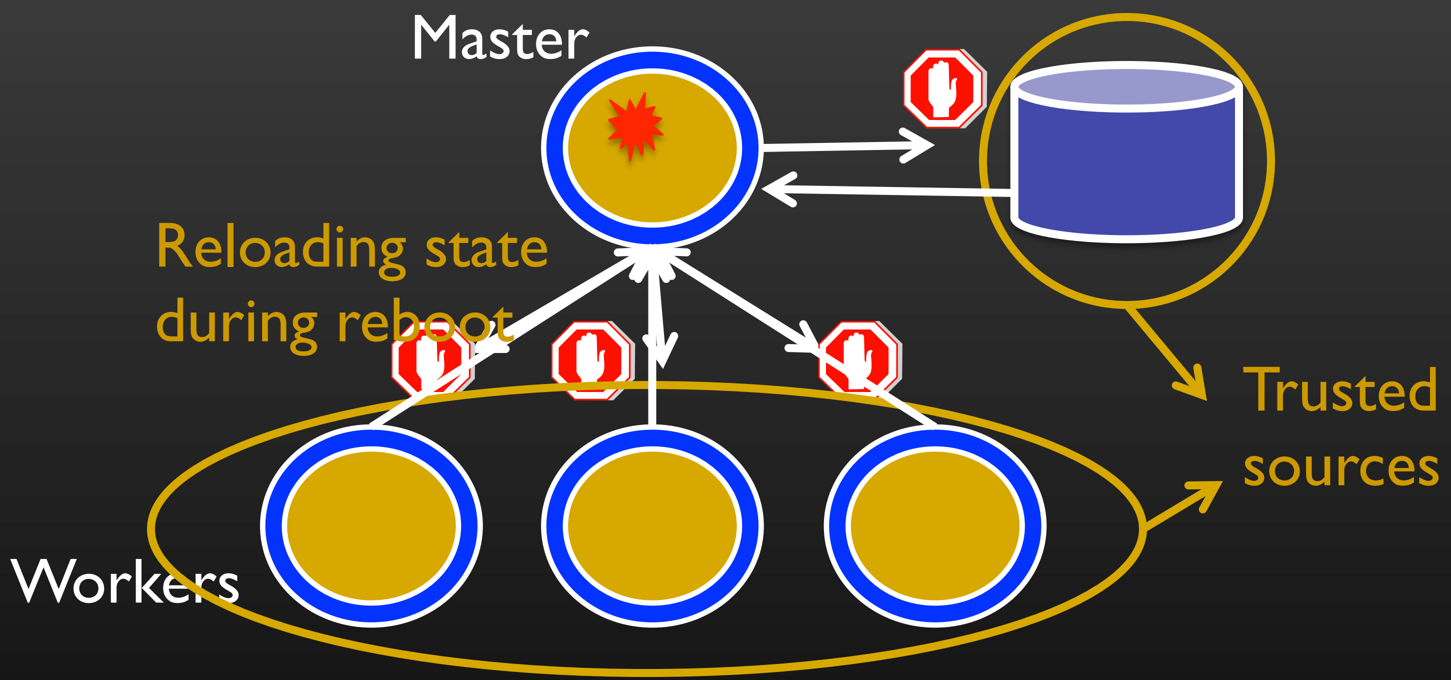


N-Version programing




- High resource consumption
- High engineering effort
- Rare deployment

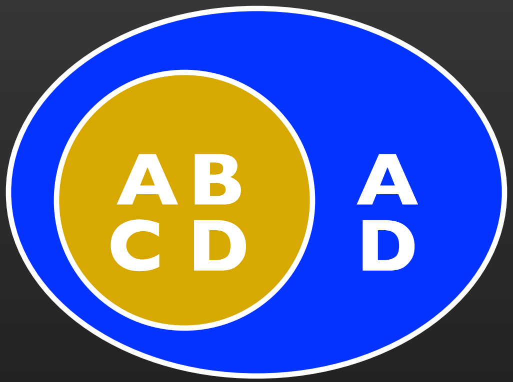
# Selective and Lightweight Versioning (SLEEVE)



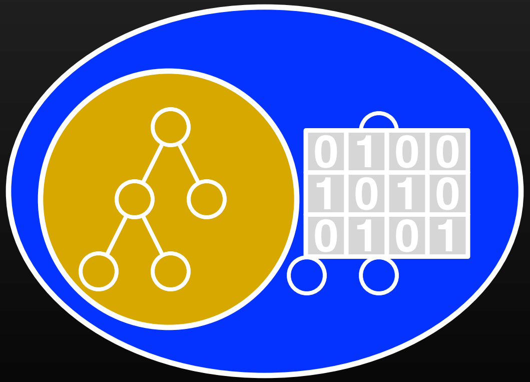

- 2nd version models basic protocols of the system
- Detects and isolates fail-silent behaviors
- Exploits crash recovery machinery for recovery

# Selective and lightweight versioning (SLEEVE)

- Selective
  - Goal: small engineering effort
  - Protects important parts
    - Bug sensitive
    - Frequently changed
    - Currently unprotected




- Lightweight
  - Avoids replicating full state
  - Encodes states to reduce space



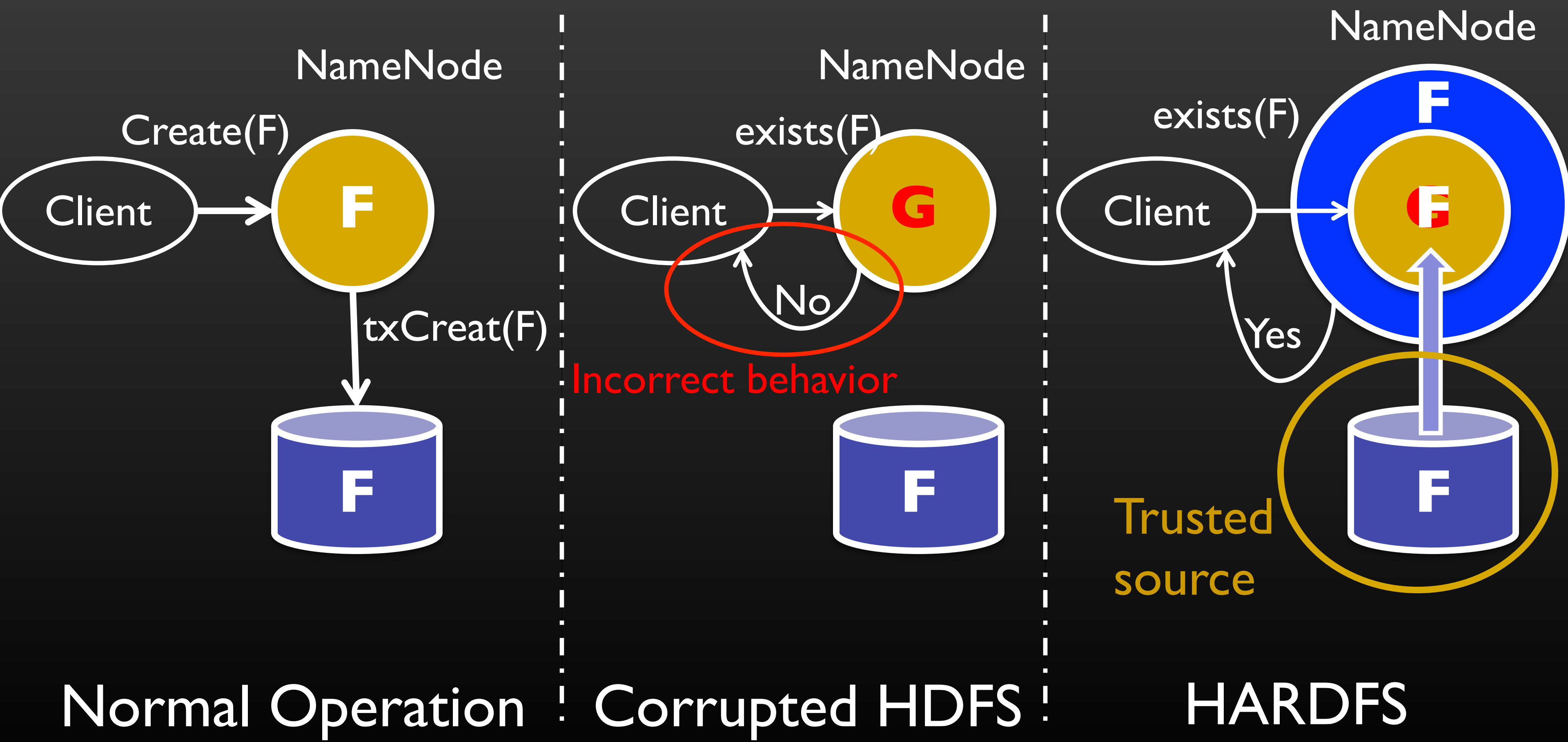
# HARDFS

- HARDFS - hardened version HDFS:
  - Namespace management
  - Replica management
  - Read/write protocol
- HARDFS detects and recovers from:
  - 90% of the faults caused by random memory corruption
  - 100% of the faults caused by targeted memory corruption
  - 5 injected software bugs
- Fast recovery using micro-recovery
  - 3 orders of magnitude faster than full reboot
- Little space and performance overhead

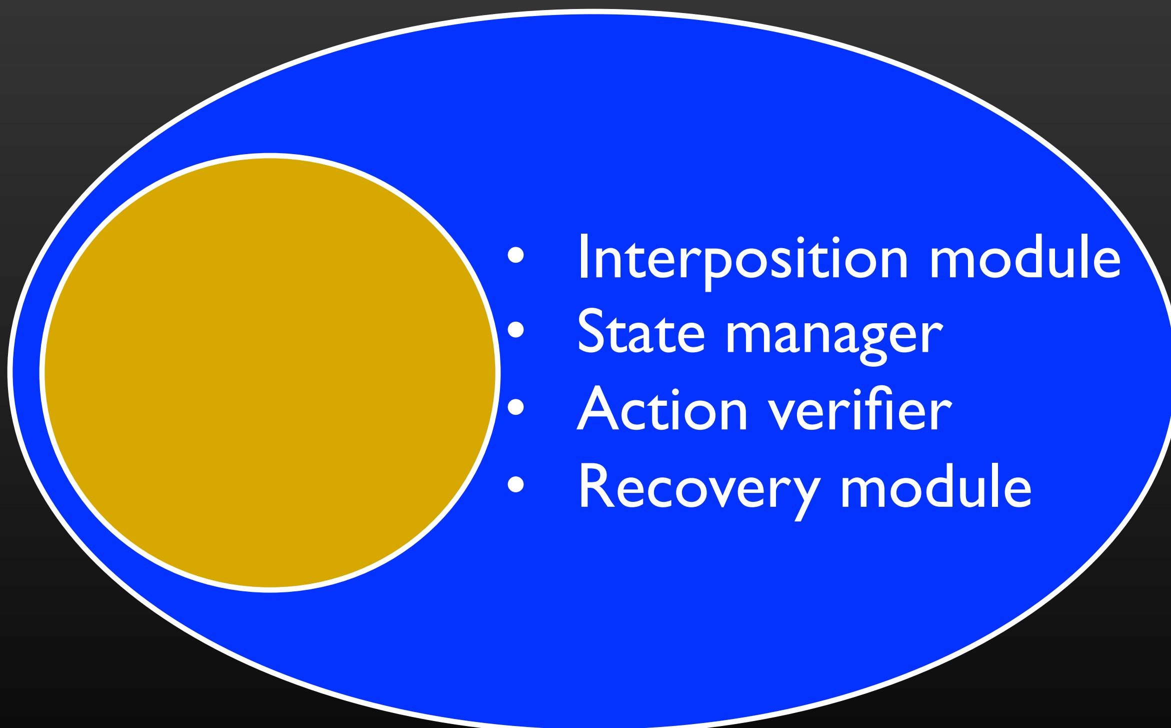
# Outline

- ✓ Introduction
- **HARDFS Design**
- HARDFS Implementation
- Evaluation
- Conclusion

# Case study: namespace integrity



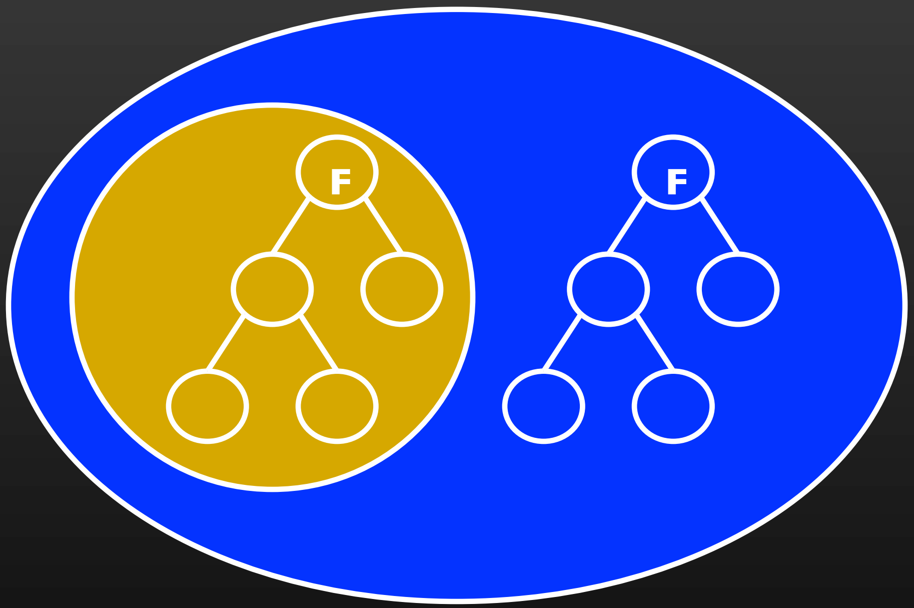
# SLEEVE layer components



# State manager

- Replicates subset of state of the main version
  - Directory entries without modification time
- Adds new state incrementally
  - Adds permissions for security checks
- Understands semantics of various protocol messages and thread events to update state correctly
- Compresses state using compact encoding

# Naïve: Full replication



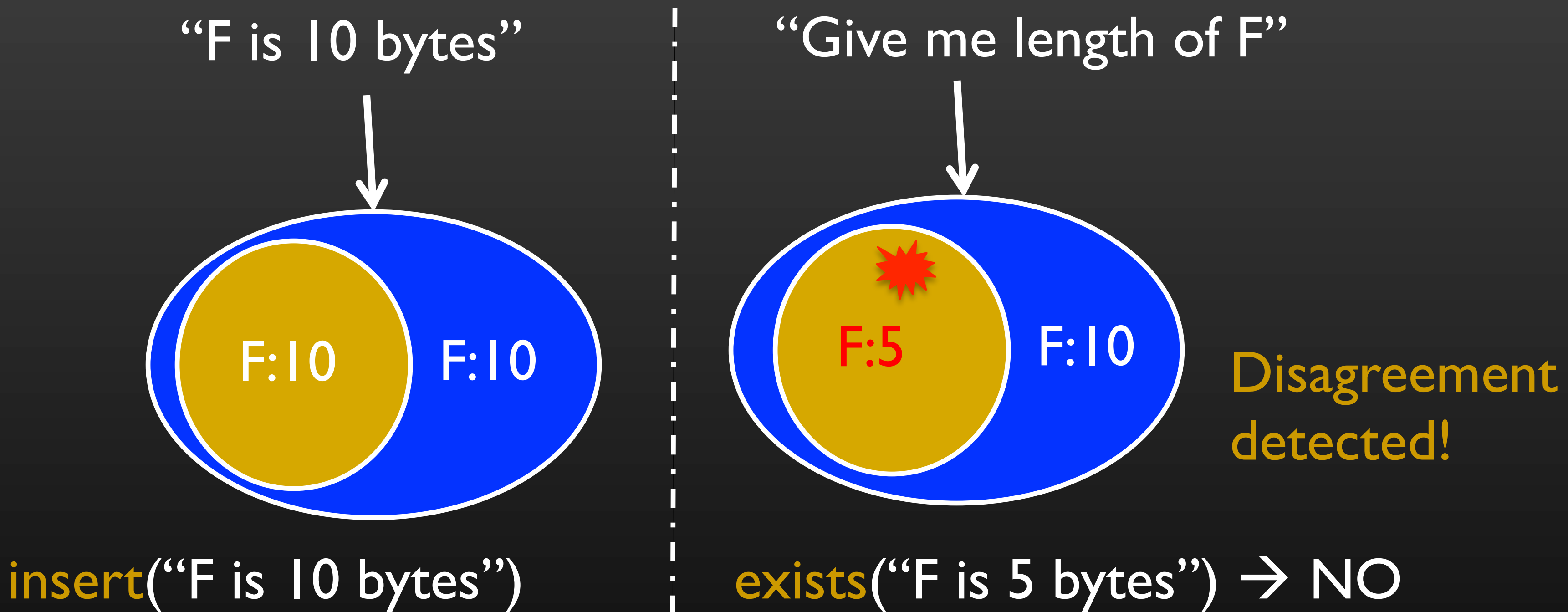

100% memory overhead

- HDFS master manages millions of files
- 100% memory overhead reduces HDFS master scalability [;login; '11]

# Lightweight: Counting Bloom Filters

- Space-efficient data structure
- Supports 3 APIs
  - insert(“A fact”)
  - delete(“A fact”)
  - exists(“A fact”)

# Lightweight: Counting Bloom Filters



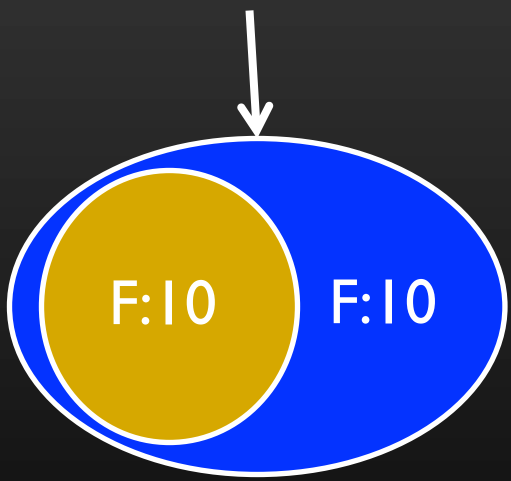

- Suitable for boolean checking
  - Does F exist?
  - Does F has length X?
  - Has block B been allocated?

# Challenges of using Counting Bloom Filters

- Hard to check stateful system
- False positives

# Non-boolean verification

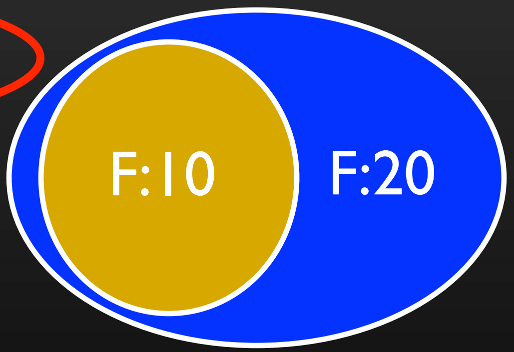
"F is 20 bytes"




Before

Bloom filter does not support this API

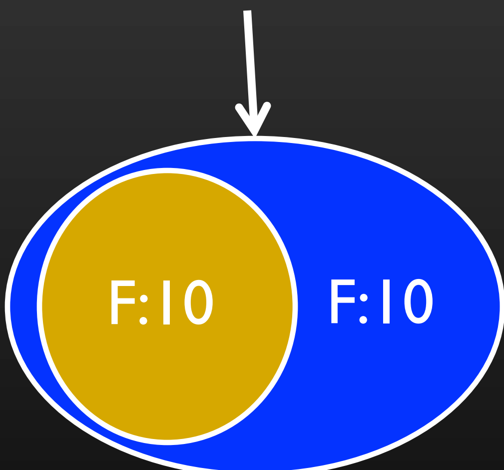
X = returnSize(F)
delete(F:X)
insert(F:20)




After

# Non-boolean verification

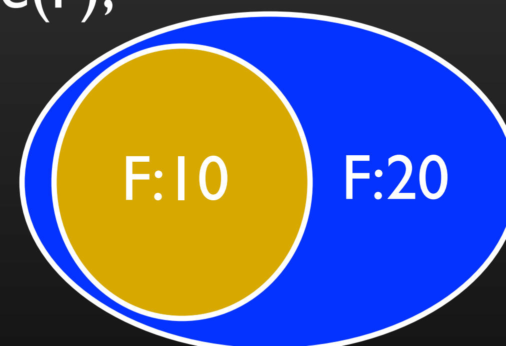
“F is 20 bytes”




Before

## Ask-Then-Check

X 🡐 MainVersion.returnSize(F);
IF exists(F:X)
        delete(F:X);
        insert(F:20);
ELSE
        initiate recovery;




After

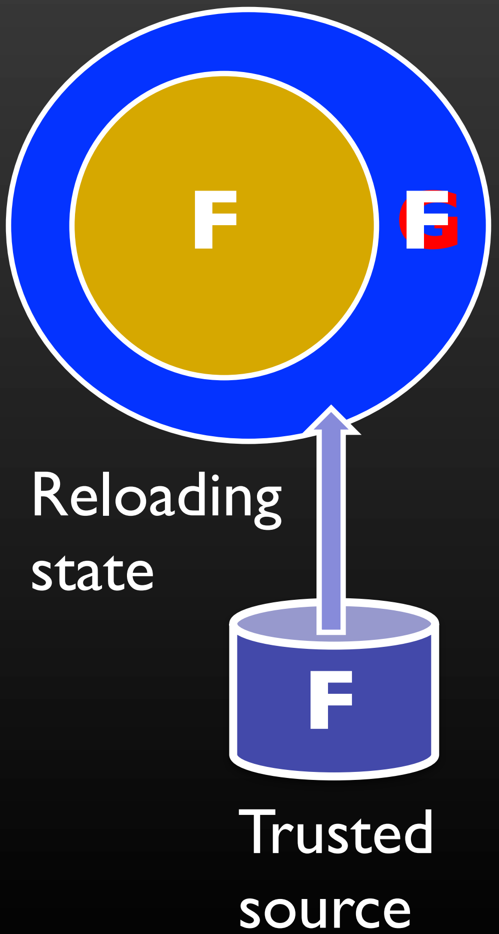
# Stateful verification

Ask Then Check

Checking stateful systems

Bloom Filter (boolean verification)

# Dealing with False positive

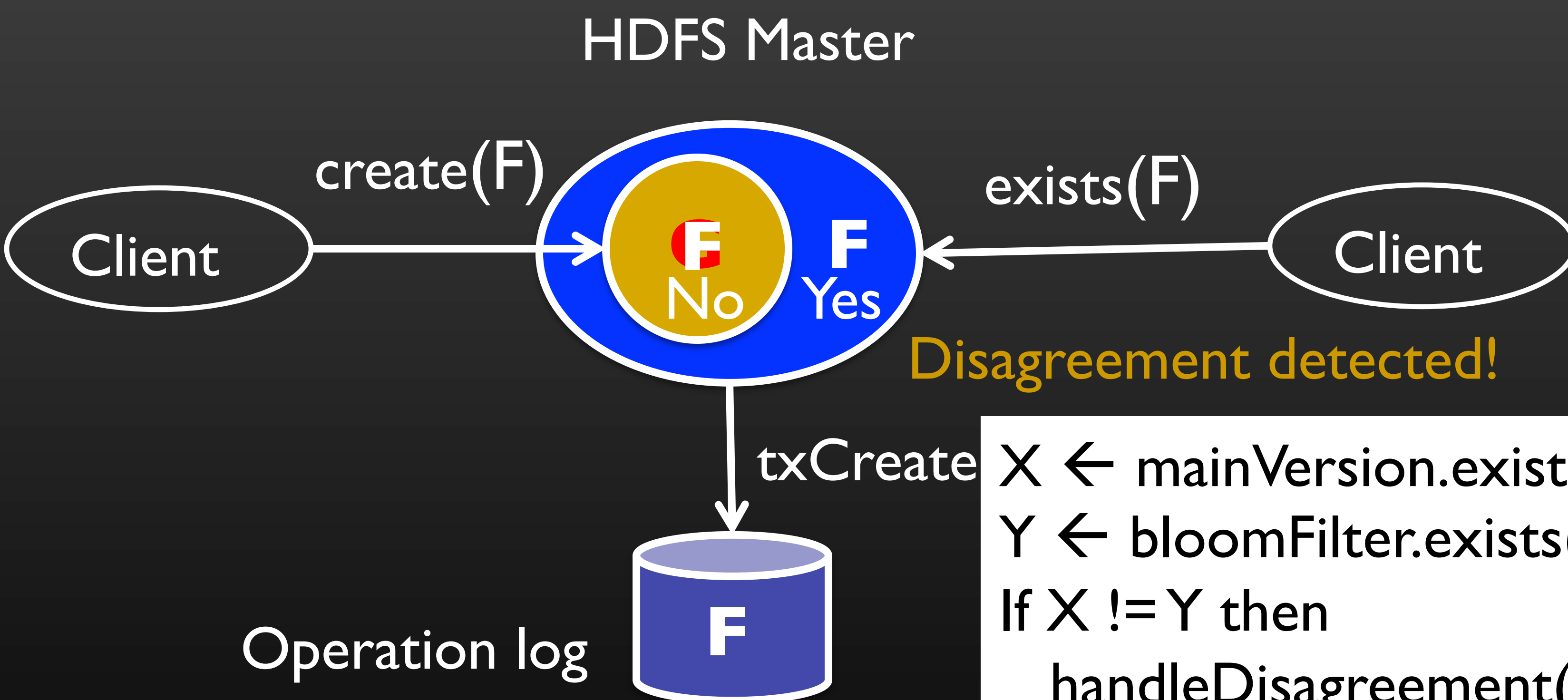



- Bloom filters can give false positive
  - 4 per billion
  - 1 false positive per month (given 100 op/s)
- Only leads to unnecessary recovery

# Outline

- ✓ Introduction
- ❑ **HARDFS Design**
  - ✓ Lightweight
  - Selective
  - Recovery
- ❑ HARDFS Implementation
- ❑ Evaluation
- ❑ Conclusion

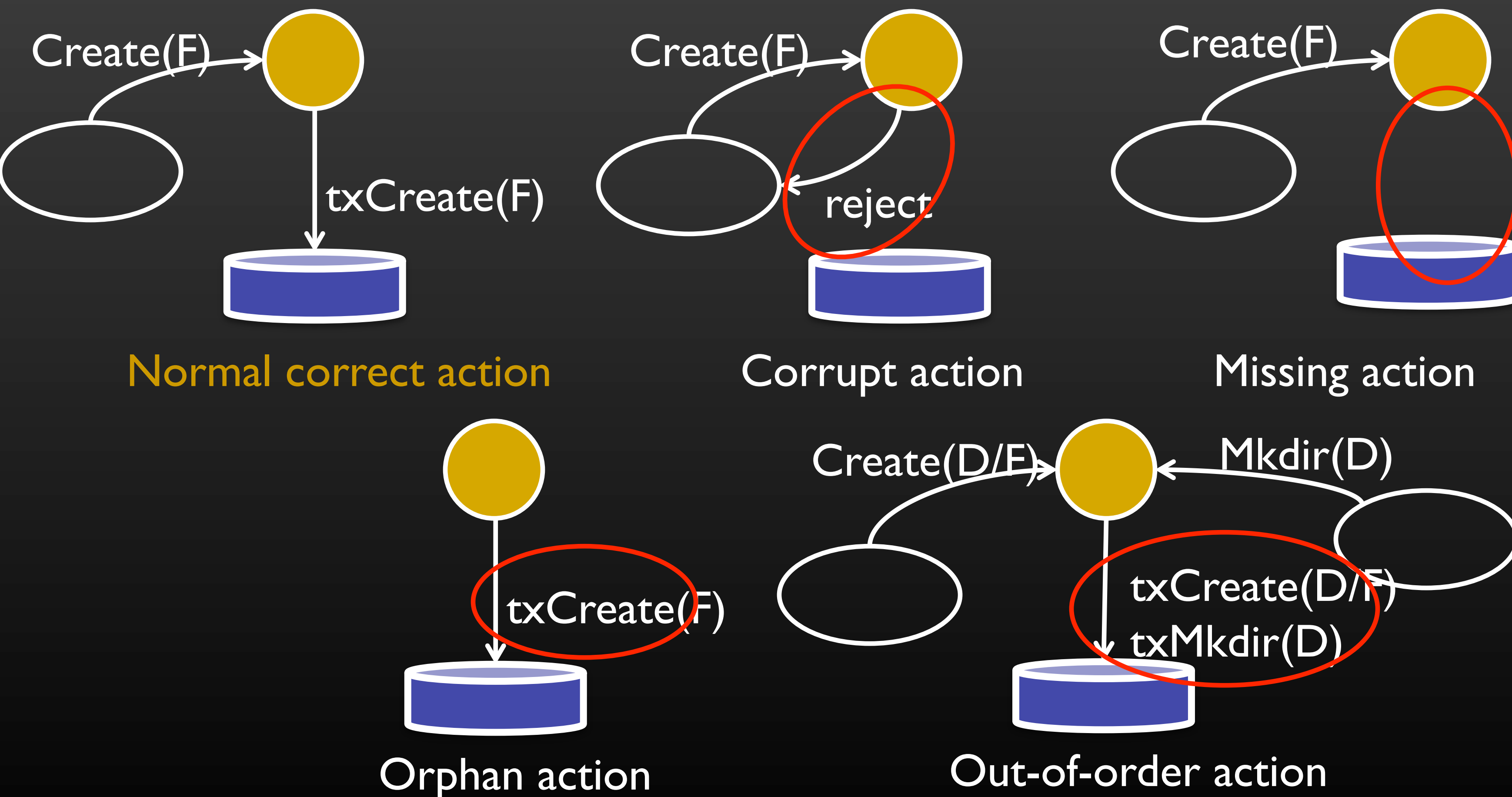
# Selective Checks




X ← mainVersion.exists(F);
Y ← bloomFilter.exists(F);
If X != Y then
  handleDisagreement();

- Goals: small engineering effort
- Selectively chooses namespace protection
- Excludes security checks

# Incorrect action examples




**All of these happen in practice**

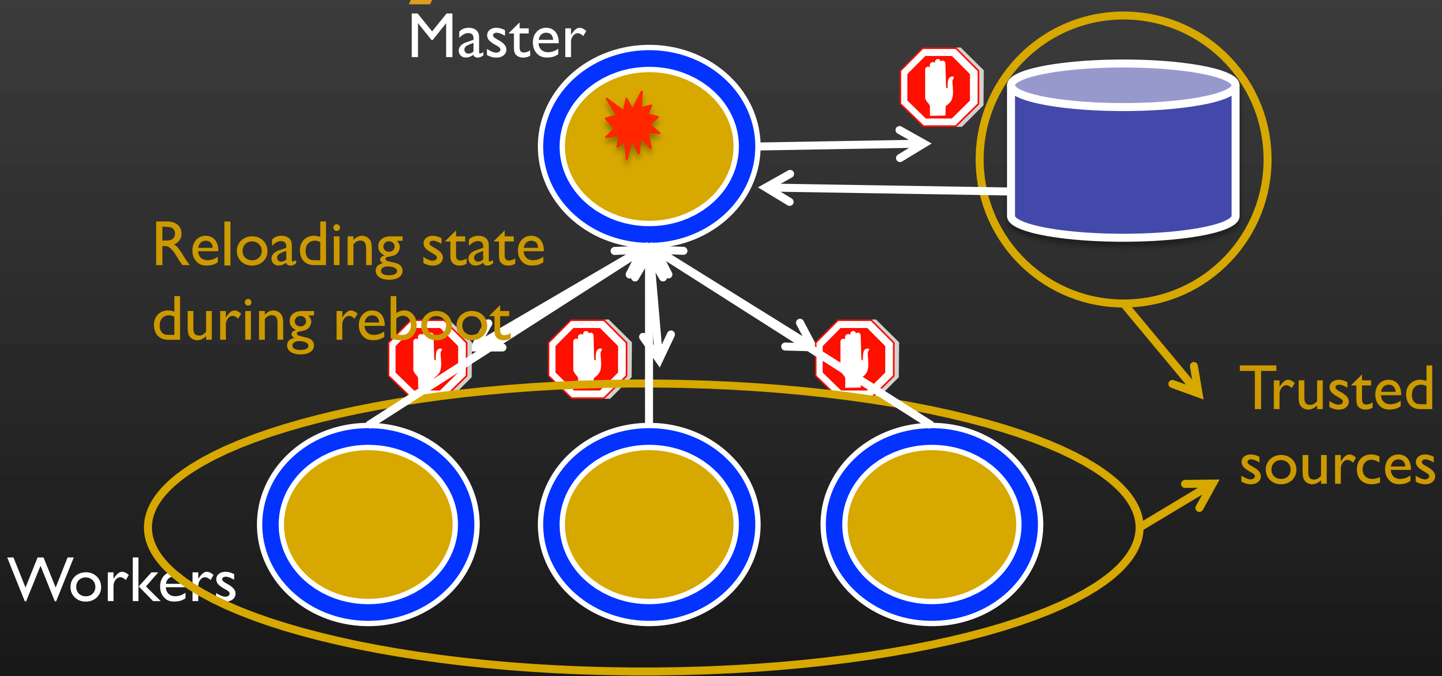
# Action verifier

- Set of micro-checks to detect incorrect actions of the main version
- Mechanisms:
  - Expected-action list
  - Actions dependency checking
  - Timeout
  - Domain knowledge to handle disagreement

# Outline

- ✓ Introduction
- ❑ **HARDFS Design**
  - ✓ Lightweight
  - ✓ Selective
  - ❑ Recovery
- ❑ HARDFS Implementation
- ❑ Evaluation
- ❑ Conclusion

# Recovery

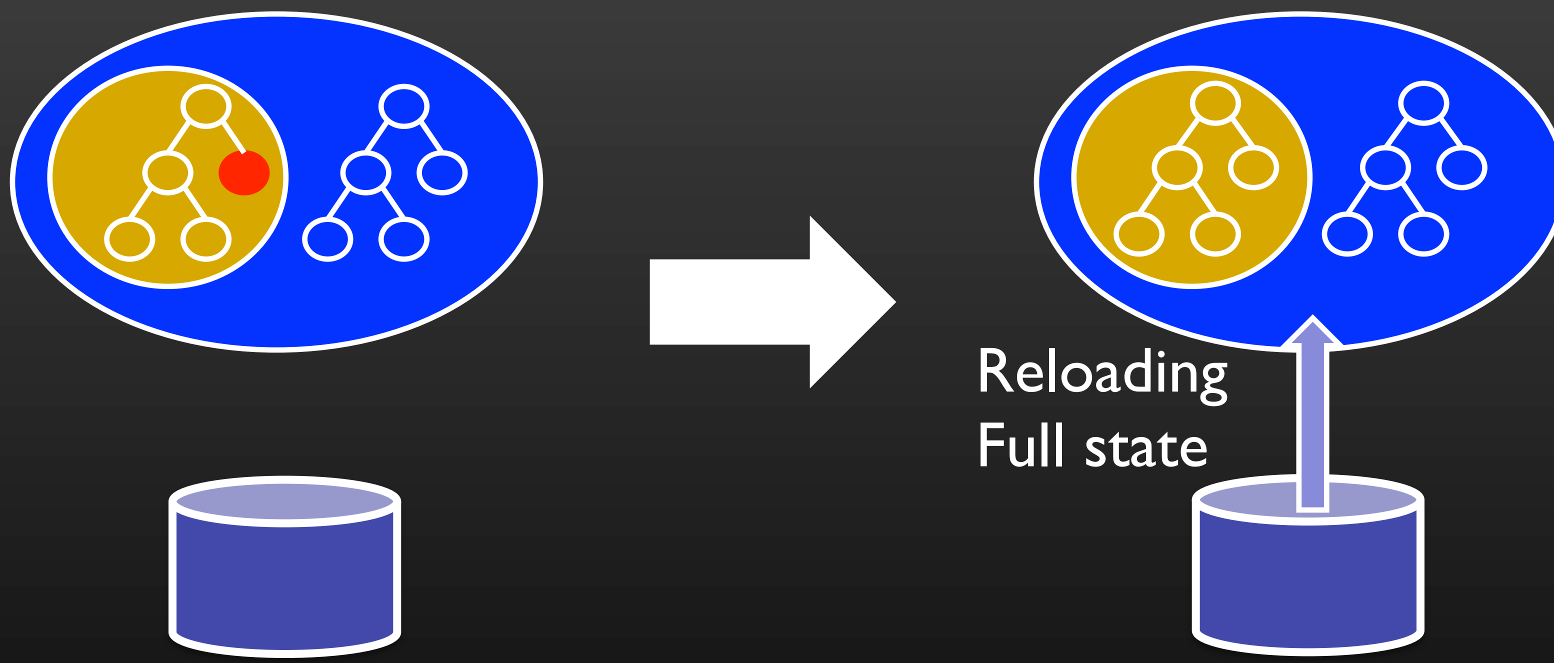



- Crash is good provided no fault propagation
- Detects and turns bad behaviors into crashes
- Exploits HDFS crash recovery machineries

# HARDFS Recovery

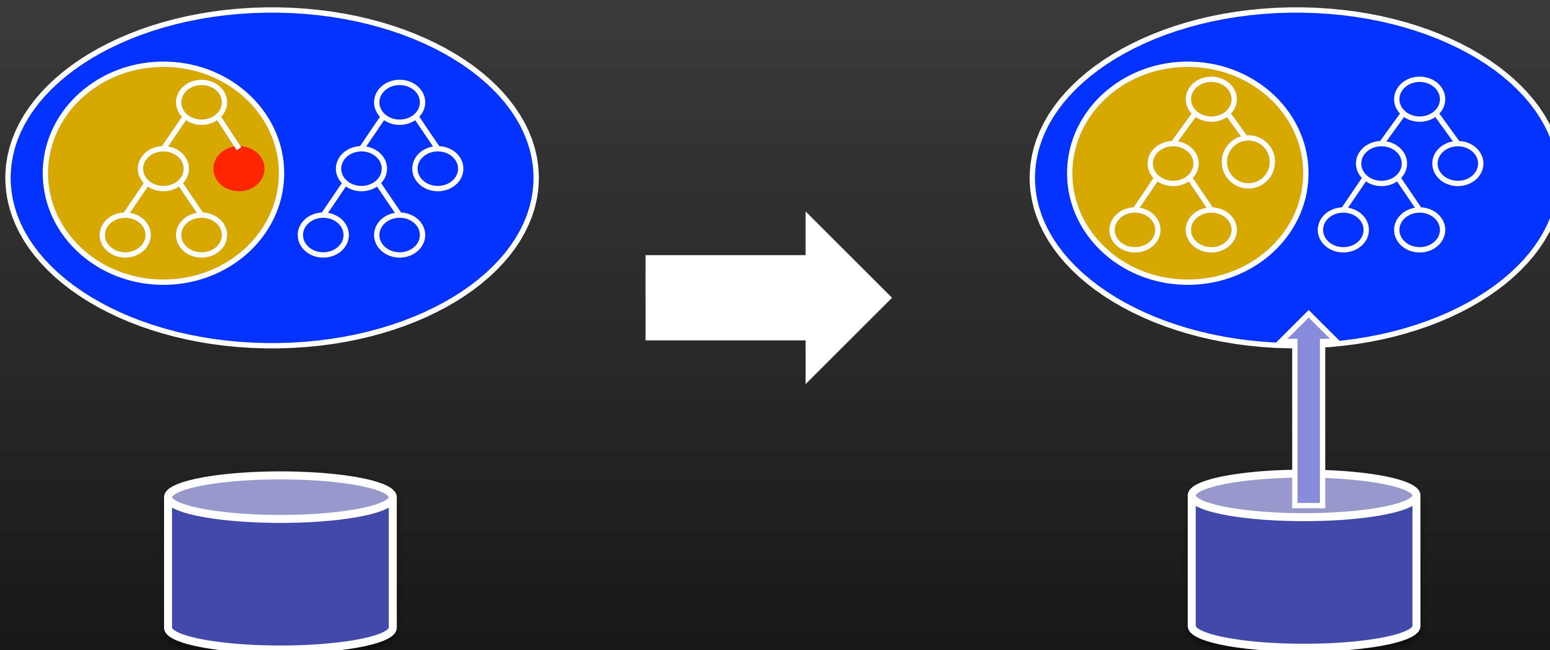
- Full recovery (crash and reboot)
- Micro-recovery
  - Repairing the main version
  - Repairing the 2nd version

# Crash and Reboot



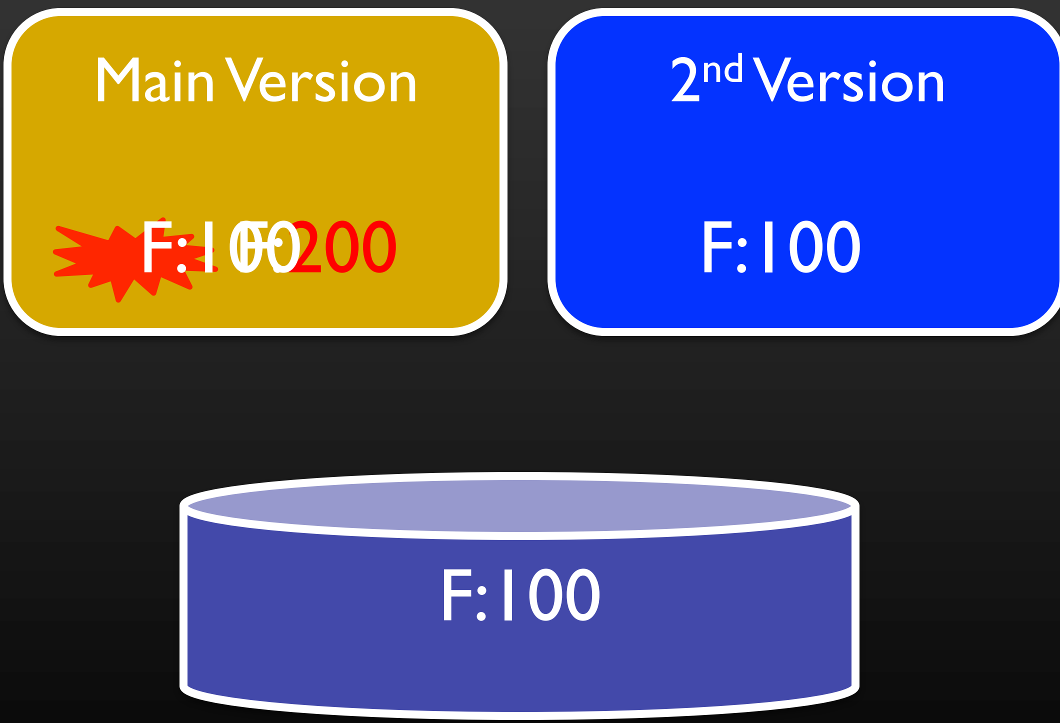

- Full state is reconstructed from trusted sources
- Full recovery may be expensive
  - Restarting an HDFS master could take hours

# Micro-recovery



- Repairs only corrupted state from trusted sources
- Falls back to full reboot when micro-recovery fails

# Repairing main version

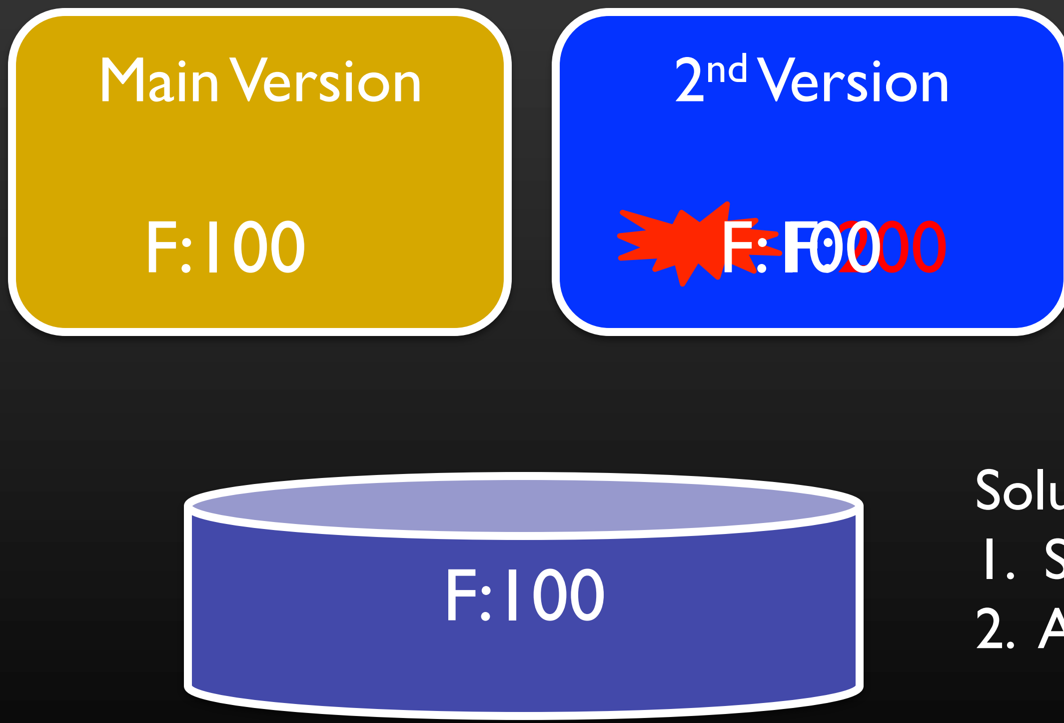



Trusted source: checkpoint file

Direct update

F:200 ← F:100

# Repairing 2nd version




Trusted source: checkpoint file

Must:

1. Delete("F is 200 bytes")
2. Insert("F is 100 bytes")

Solution:

1. Start with an empty BF
2. Add facts as they are verified

# Outline

- ✓ Introduction
- ✓ HARDFS Design
- ❑ **HARDFS Implementation**
- ❑ Evaluation
- ❑ Conclusion

# Implementation

- Hardens three functionalities of HDFS
  - Namespace management (HARDFS-N)
  - Replica management (HARDFS-R)
  - Read/write protocol of datanodes (HARDFS-D)
- Uses 3 Bloom filters API
  - insert("a fact"), delete("a fact"), exists("a fact")
- Uses ask-then-check for non-boolean verification

# Protecting namespace integrity

- Guards namespace structures necessary for reaching data:
  - File hierarchy
  - File-to-block mapping
  - File length information
- Detects and recovers from namespace-related problems:
  - Corrupt file-to-block mapping
  - Unreachable files

# Namespace management

| Message | Logic of the secondary version |
|---|---|
| Create(F):<br>*Client request NN to create F* | **Entry**:<br>  **If** ***exists***(F) **Then** reject;<br>  **Else**<br>    ***insert***(F);<br>    generateAction(txCreate[F]);<br>**Return**: check response; |
| AddBlock(F):<br>*client requests NN to allocate a block to file F* | **Entry**:<br>  F:X = ***ask-then-check***(F);<br>**Return**:<br>  B = addBlk(F);<br>  **If** ***exists***(F) & !***exists***(B) **Then**<br>    $X' = X \cup \{B\}$;<br>    ***delete***(F:X);<br>    ***insert***(F:X′)<br>    ***insert***(B@0);<br>  **Else** declare error; |

# Outline

- ✓ Introduction
- ✓ HARDFS Design
- ✓ HARDFS Implementation
- ❑ Evaluation and Conclusion

# Evaluation

- Is HARDFS robust against fail-silent faults?
- How much time and space overhead incurred?
- Is micro-recovery efficient?
- How much engineering effort required?

# Random memory corruption results

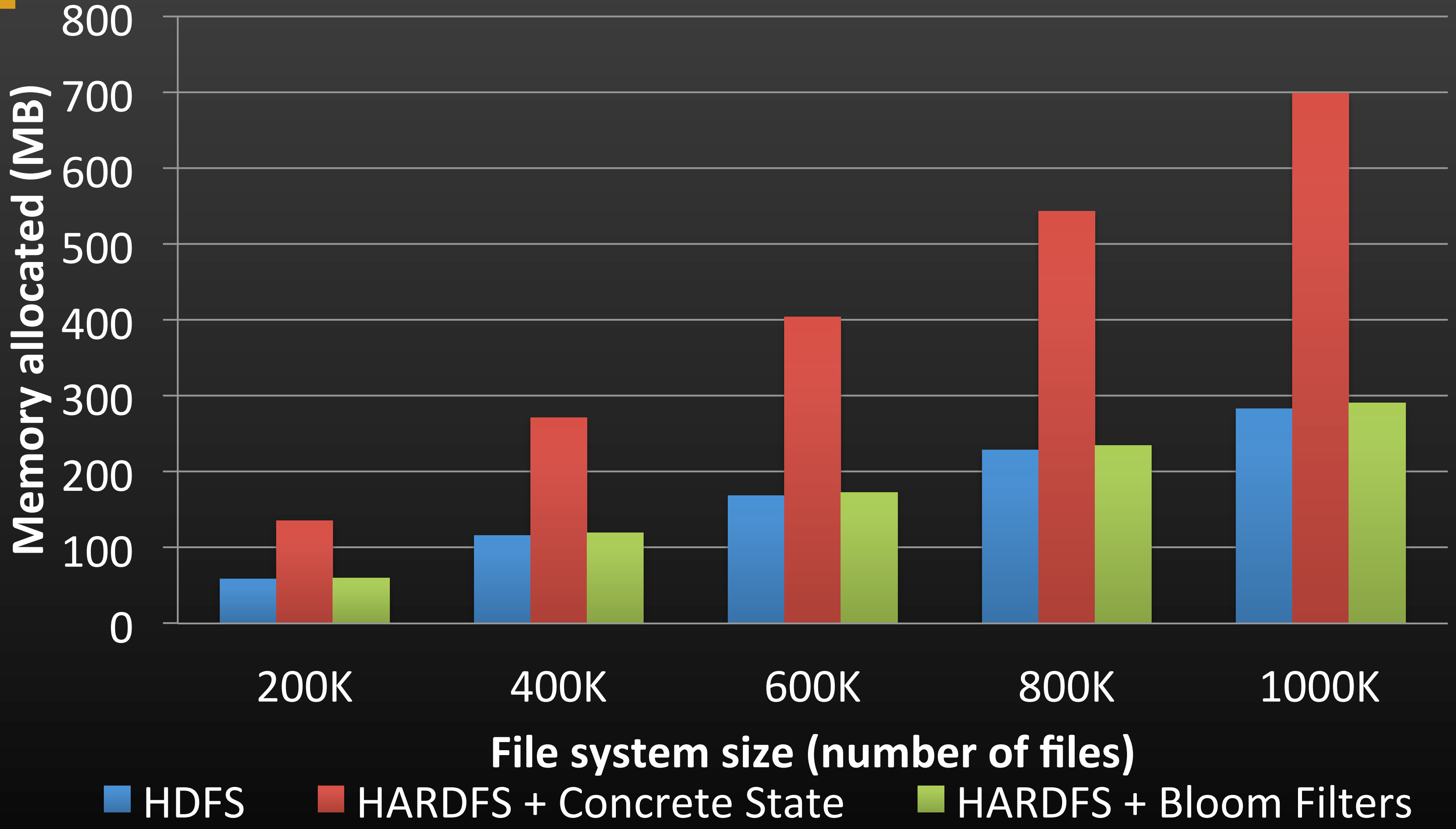
| Outcome | HDFS | HARDFS |
|---|---|---|
| Silent failure | 117 | 9 |
| Detect and reboot | - | 140 |
| Detect and micro-recover | - | 107 |
| Crash | 133 | 268 |
| Hang | 22 | 16 |
| No problem observed | 728 | 460 |

- # fail-silent failures reduced by factor of 10
- Crash happens twice as often

# Silent failures

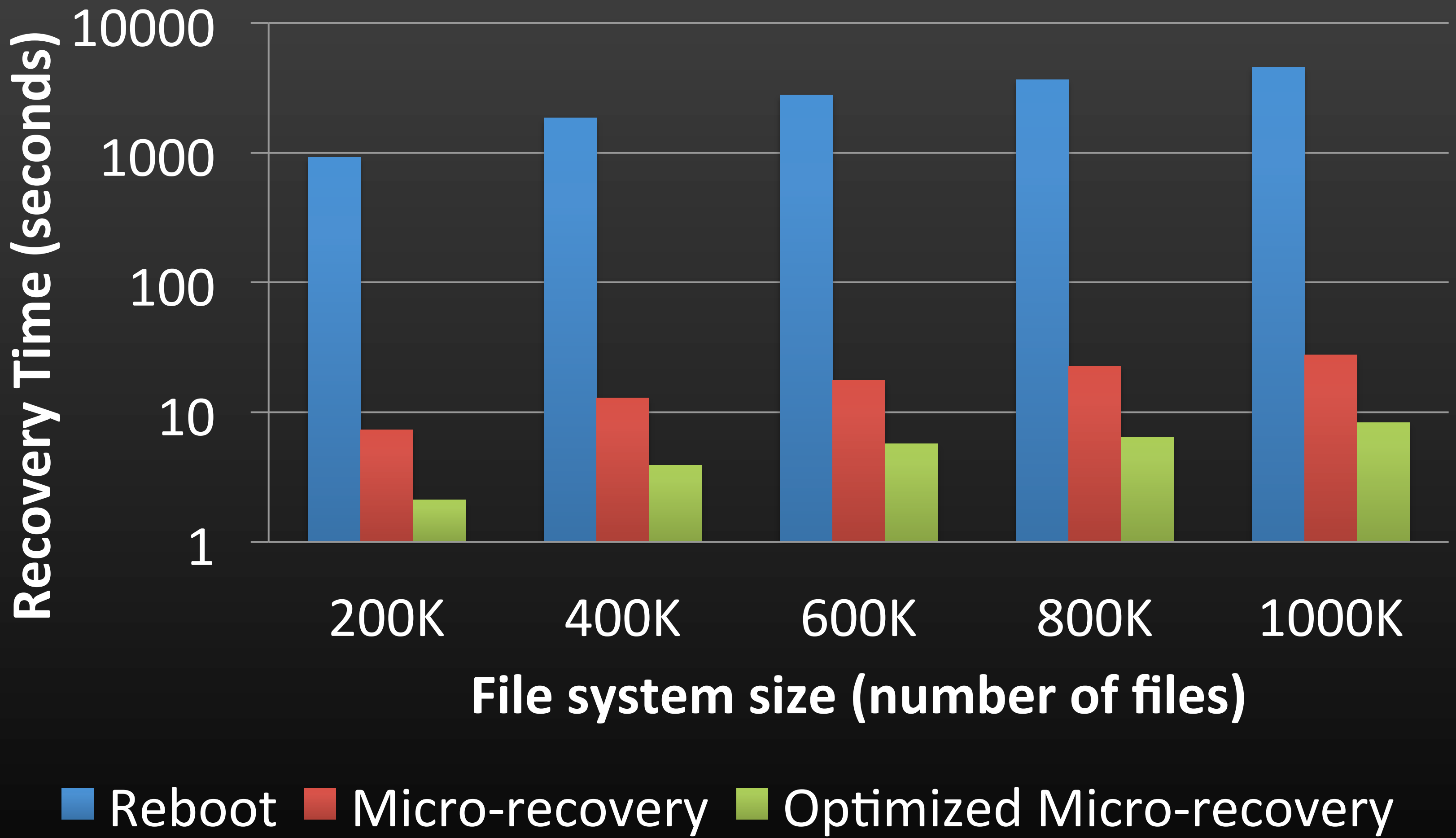
| FIELD | HDFS | HARDFS |
|---|---:|---:|
| pathname | 95 | 0 |
| replication | 1 | 0 |
| modification time | 6 | 8 |
| permission | 3 | 0 |
| block size | 12 | 1 |

# Namepsace management Space Overhead




**HARDFS with Bloom filter incurs little space overhead (2.6%)**

# Recovery Time




- Rebooting NameNode is expensive
- Micro-recovery is 3 order of magnitude faster

# Complexity (LOC)

| Functionality | HDFS | HARDFS | |
|---|---|---|---|
| Namespace management | 10114 | 1751 | 17% |
| Replica management | 2342 | 934 | 40% |
| Read/write protocol | 5050 | 944 | 19% |
| Others | 13339 | - | - |

- **Lightweight versions are smaller**

# Injecting software bugs

| Bug | Year | Priority | Description |
|---|---|---|---|
| HADOOP-1135 | 2007 | Major | Blocks in block report wrongly marked for deletion |
| HADOOP-3002 | 2008 | Blocker | Blocks removed during safemode |
| HDFS-900 | 2010 | Blocker | Valid replica deleted rather than corrupt replica |
| HDFS-1250 | 2010 | Major | Namenode processes block report from dead datanode |
| HDFS-3087 | 2012 | Critical | Decommission before replication during namenode restart |

# Conclusion

- Crashing is good
- To die (and be reborn) is better than to lie
- But lies do happen in reality
- HARDFS turns lies into crashes
- Leverages existing crash recovery techniques to resurrect

# Thank you!
# Questions?

ADSL

WISDOM

http://research.cs.wisc.edu/adsl/

http://wisdom.cs.wisc.edu/

UCARE

http://ucare.cs.uchicago.edu/