High Velocity Kernel File Systems with Bento

Samantha Miller[1], Kaiyuan Zhang[1], Mengqi Chen[1], Ryan Jennings[1], Ang Chen[2], Danyang Zhuo[3], Tom Anderson[1]

[1] University of Washington, [2]Rice University, [3] Duke University

Kernel File System Development is Slow

- High development and deployment velocity are critical to modern cloud systems
- Linux kernel development and deployment are slow
- File systems are particularly affected due to advances in storage hardware and new demands by cloud systems

Existing Techniques aren't Sufficient

	Performance	Safety	General Programmability	Live Upgrade	Easy Debugging
VFS	✓	✗	✓	✗	✗
FUSE	✗	✓	✓	✗	✓
eBPF	✓	✓	✗	✗	✗
Bento	✓	✓	✓	✓	✓

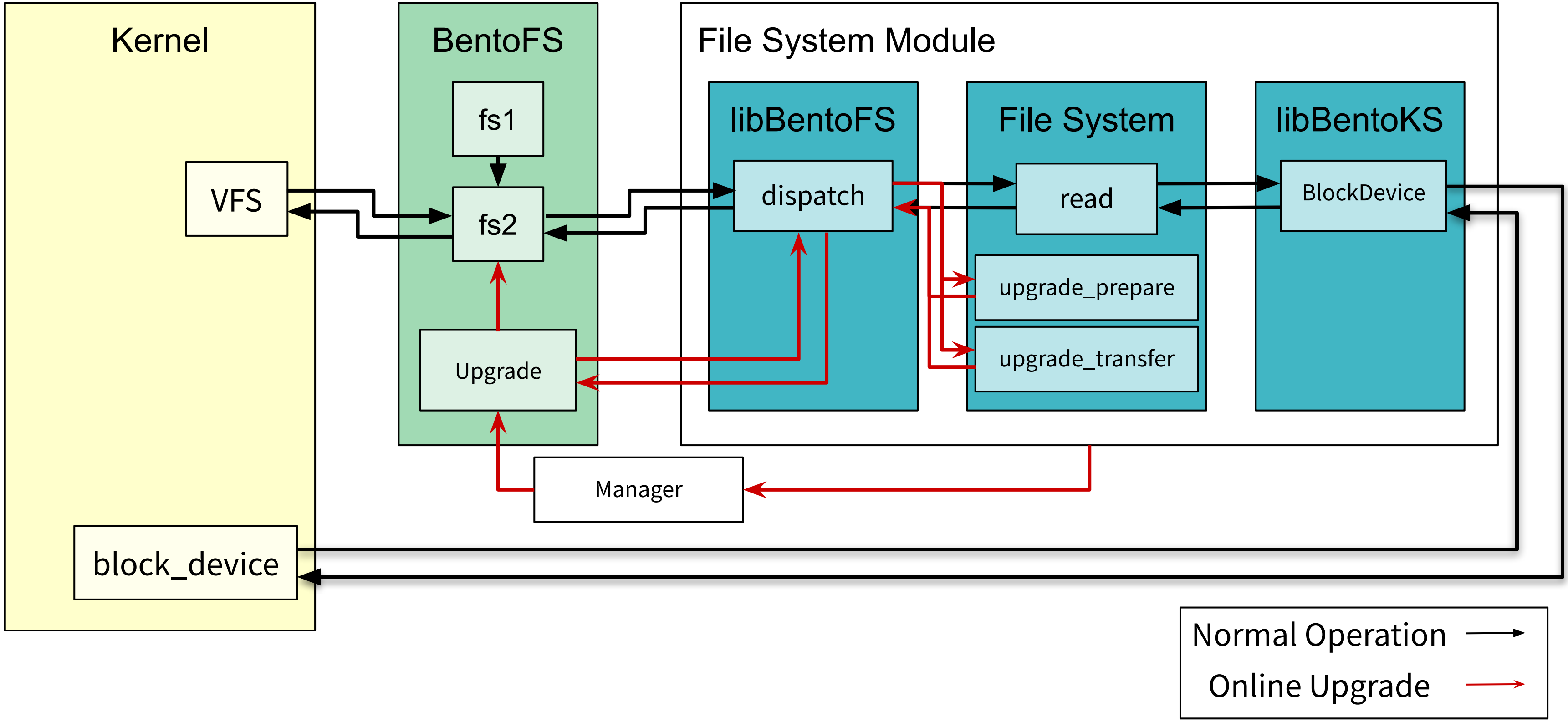
Bento Goals

- **High Performance:** Has low performance overhead
- **Safety:** Prevention of bugs in the file system
- **General Programmability:** Supports a wide variety of potential file systems
- **Compatibility:** Works with Linux and existing Linux binaries
- **Live Upgrade:** Can redeploy file systems without service downtime
- **User-level Debugging:** File system can be debugged easily with a variety of tools
- **Open Source:** Can be used by the community for research and development

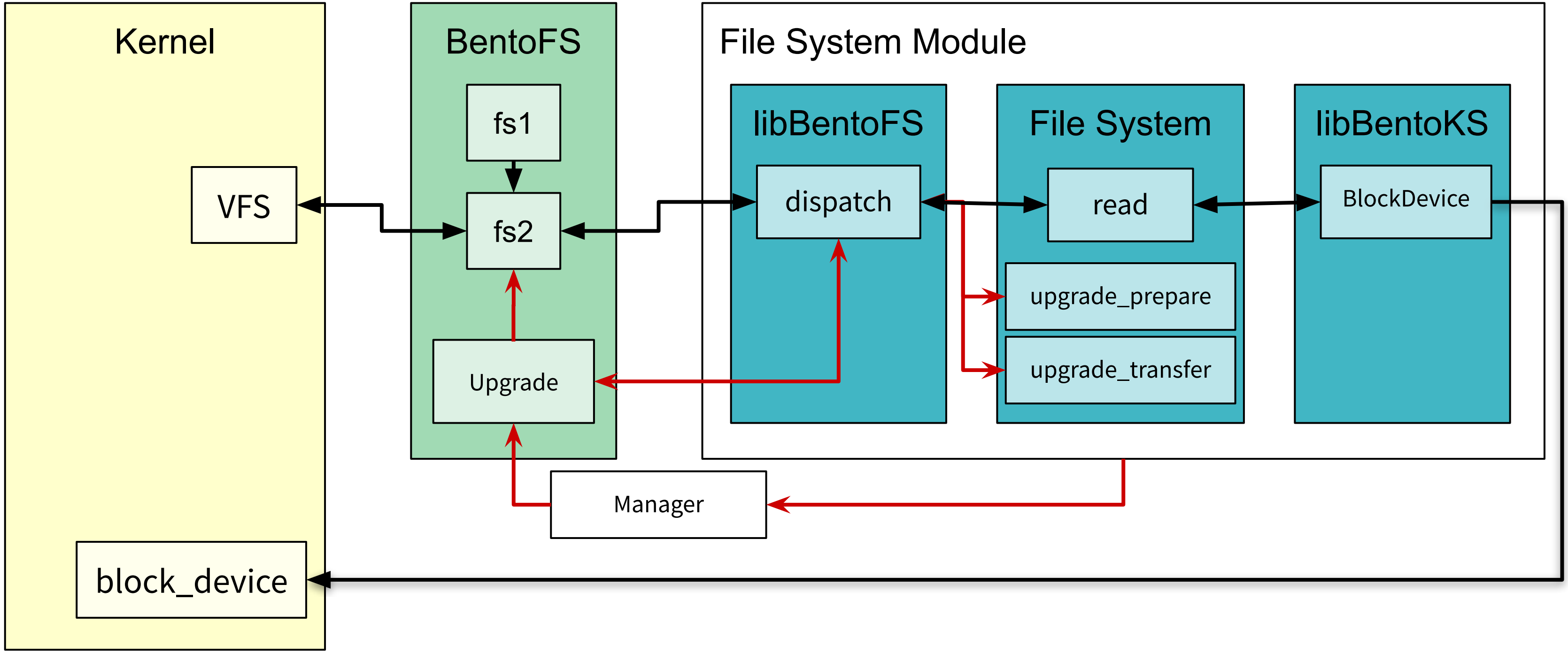
Our Approach: Bento

- File systems are implemented in safe Rust
    - Imposes little overhead and supports most designs
- How can we integrate a safe Rust file system in the kernel?
- How can we dynamically replace the file system?
- How can we support user-level execution?

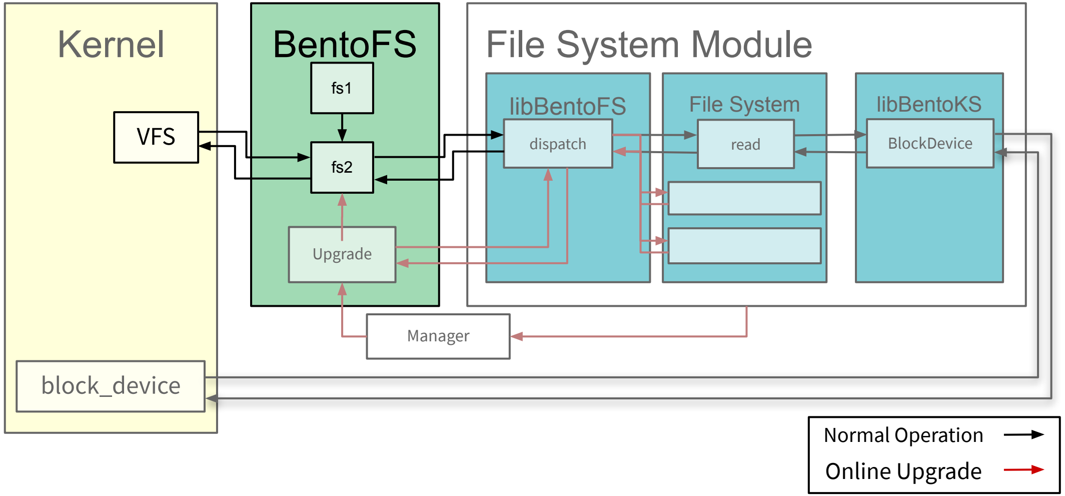
Bento


Kernel
VFS
block_device
BentoFS
fs1
fs2
Upgrade
File System Module
libBentoFS
dispatch
File System
read
upgrade_prepare
upgrade_transfer
libBentoKS
BlockDevice
Manager
Normal Operation
Online Upgrade

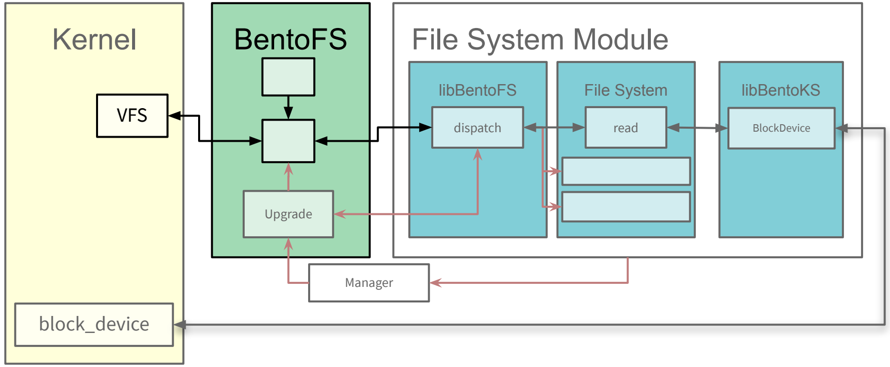
Bento


Kernel
VFS
block_device
BentoFS
fs1
fs2
Upgrade
File System Module
libBentoFS
dispatch
File System
read
upgrade_prepare
upgrade_transfer
libBentoKS
BlockDevice
Manager

BentoFS

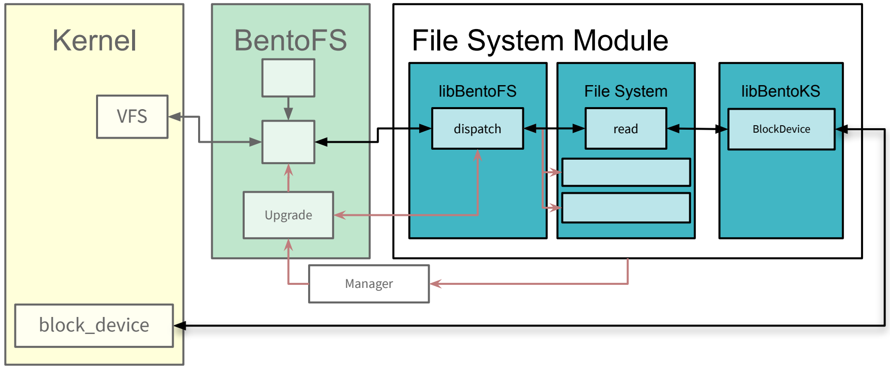

Kernel
VFS
block_device
BentoFS
fs1
fs2
Upgrade
Manager
File System Module
libBentoFS
dispatch
File System
read
libBentoKS
BlockDevice
Normal Operation
Online Upgrade

BentoFS



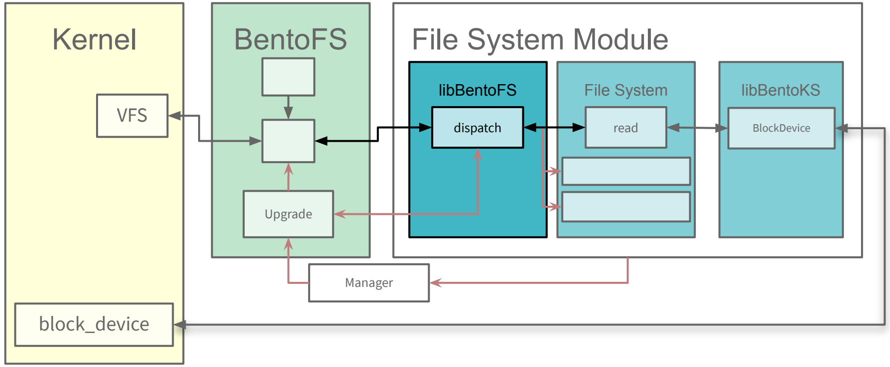
- Standalone C kernel module that hooks into VFS
- Maintains an active list of file systems
- Translates VFS calls to FUSE low-level API
- Forwards calls to the correct file system

FS Module



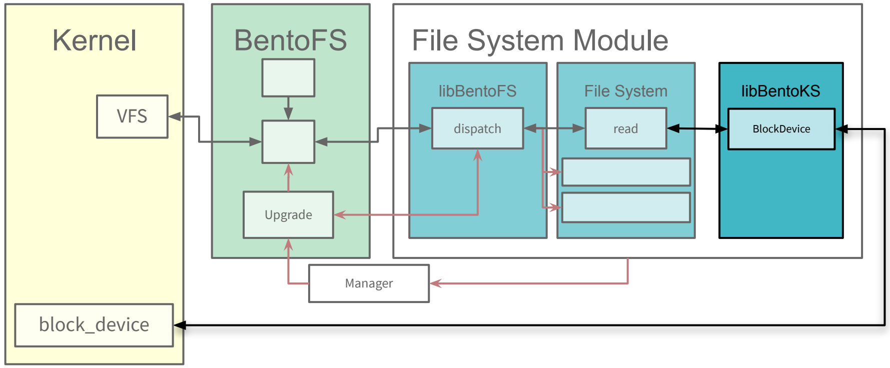
- Contains the file system and two Rust libraries: libBentoFS and libBentoKS

libBentoFS



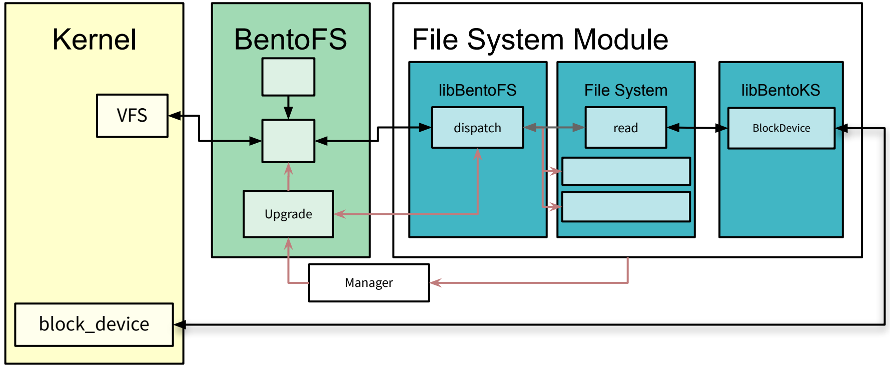
- Between BentoFS and the file system
- Converts unsafe C from BentoFS to safe Rust
  - Converts C types to Rust types
  - Converts pointers to references

libBentoKS



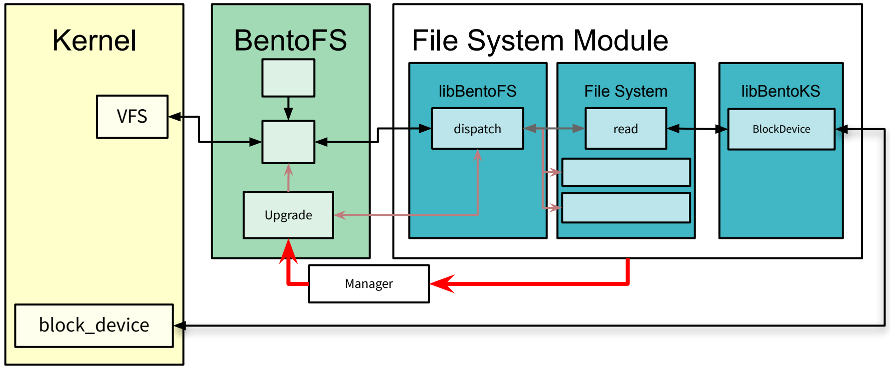
- Between the file system and the rest of the kernel
- Provides access to efficient, well-tested kernel services
- Provides safe abstractions around kernel services
  - Ensures correct ownership
  - Handles resource allocation and deallocation

Live Upgrade



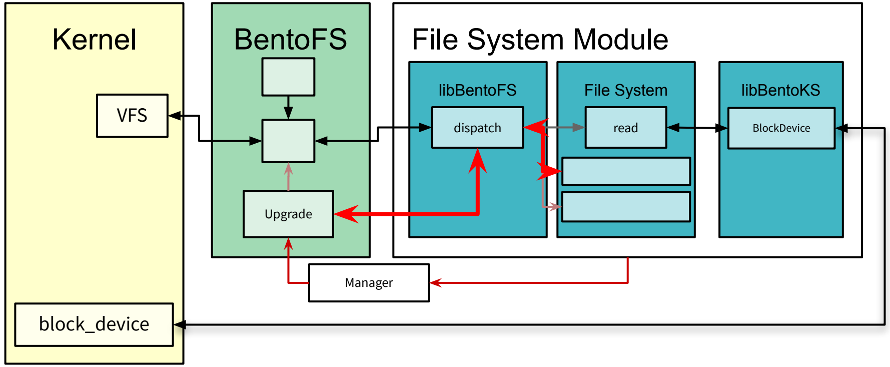
- Live upgrade component in BentoFS
- When an upgrade module is inserted:
  - BentoFS stops calls to the file system
  - Old file system cleans up and returns state struct
  - New file system initializes using state and returns
  - BentoFS swaps pointers and allows calls to proceed

Live Upgrade



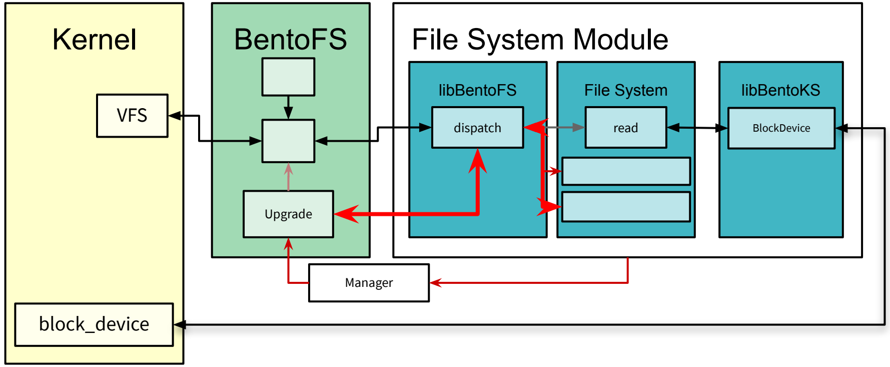
- Live upgrade component in BentoFS
- When an upgrade module is inserted:
  - **BentoFS stops calls to the file system**
  - Old file system cleans up and returns state struct
  - New file system initializes using state and returns
  - BentoFS swaps pointers and allows calls to proceed

Live Upgrade



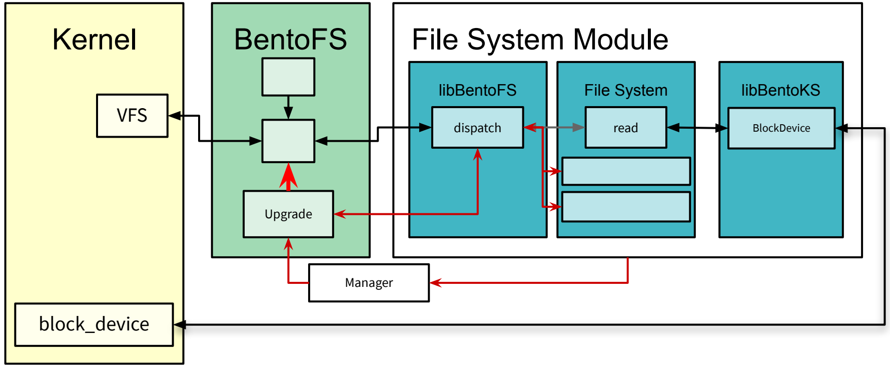
- Live upgrade component in BentoFS
- When an upgrade module is inserted:
  - BentoFS stops calls to the file system
  - **Old file system cleans up and returns state struct**
  - New file system initializes using state and returns
  - BentoFS swaps pointers and allows calls to proceed

Live Upgrade



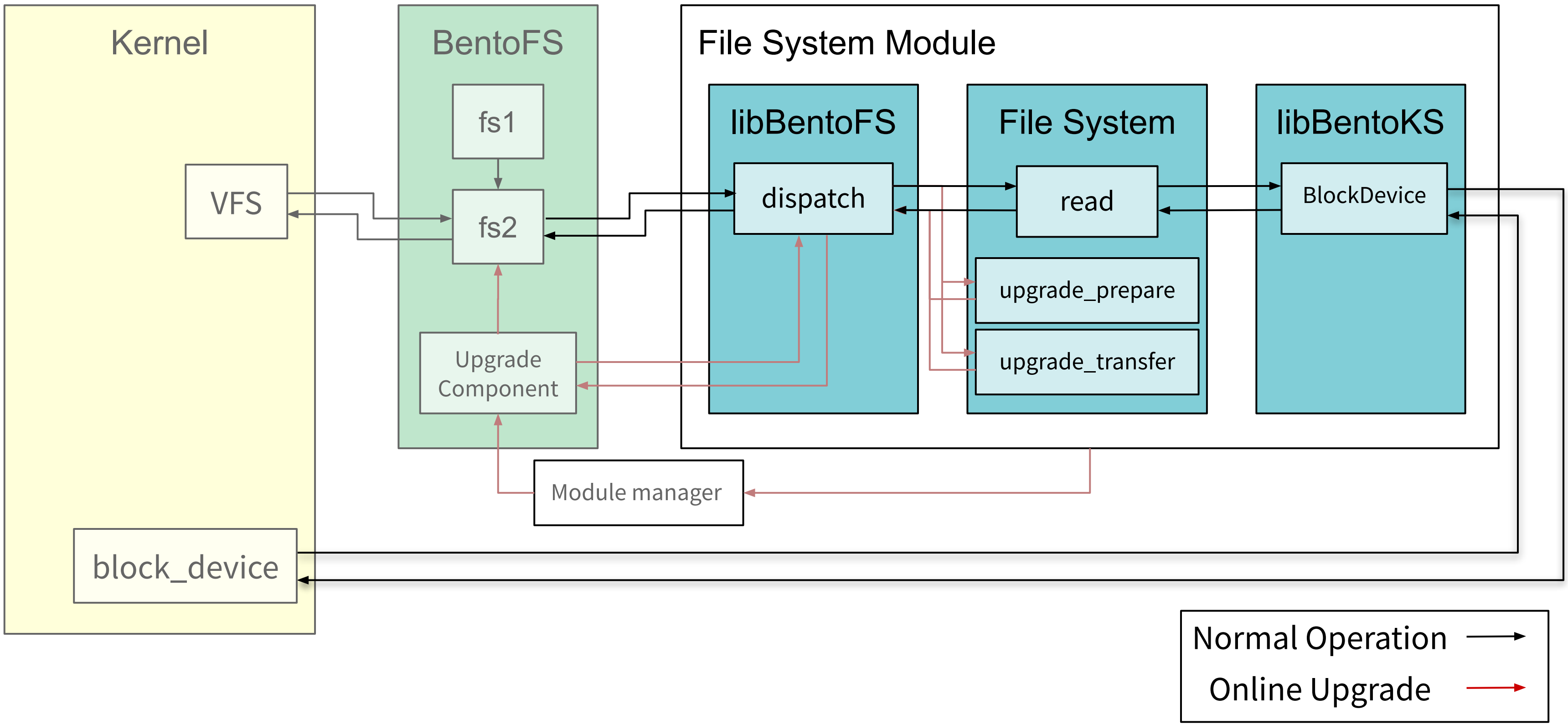
- Live upgrade component in BentoFS
- When an upgrade module is inserted:
  - BentoFS stops calls to the file system
  - Old file system cleans up and returns state struct
  - **New file system initializes using state and returns**
  - BentoFS swaps pointers and allows calls to proceed

Live Upgrade

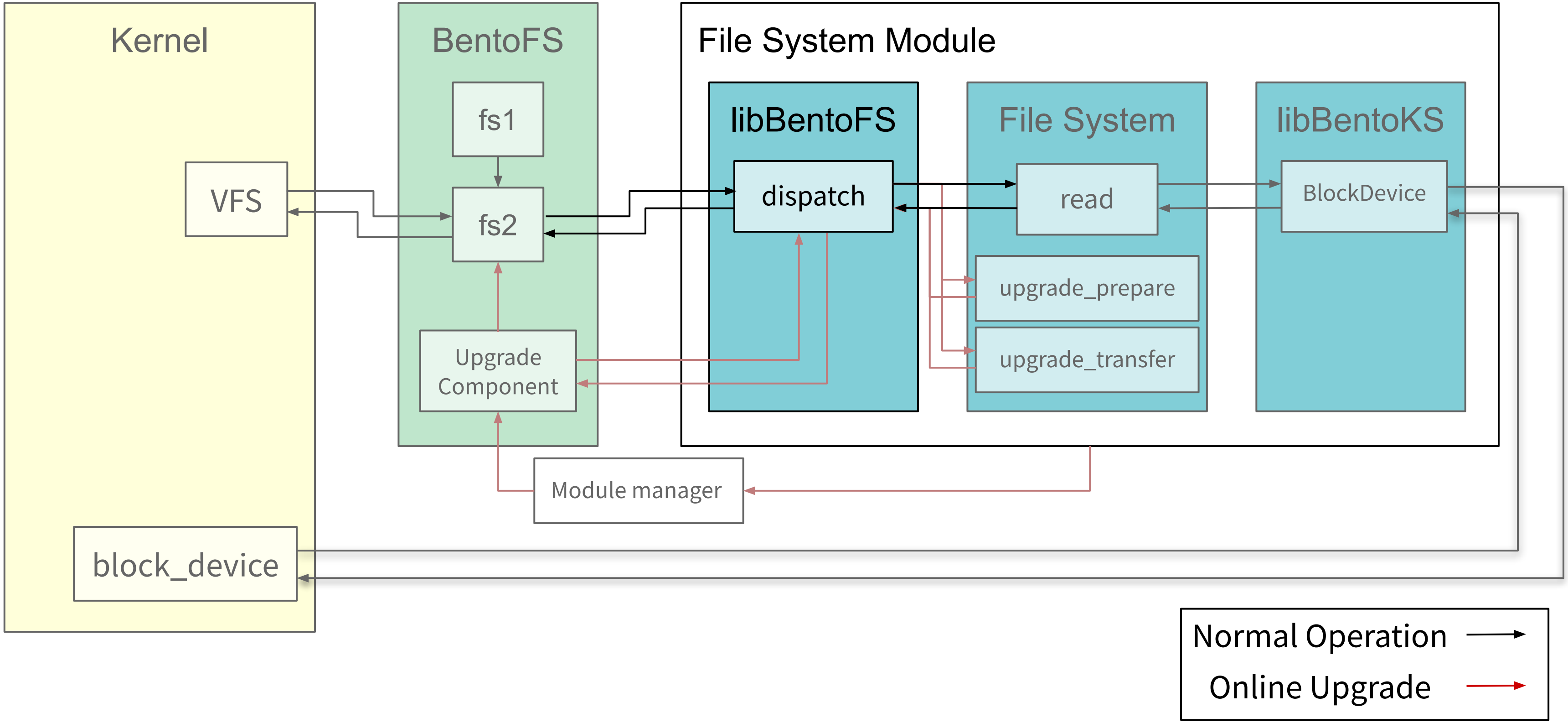


- Live upgrade component in BentoFS
- When an upgrade module is inserted:
  - BentoFS stops calls to the file system
  - Old file system cleans up and returns state struct
  - New file system initializes using state and returns
  - **BentoFS swaps pointers and allows calls to proceed**

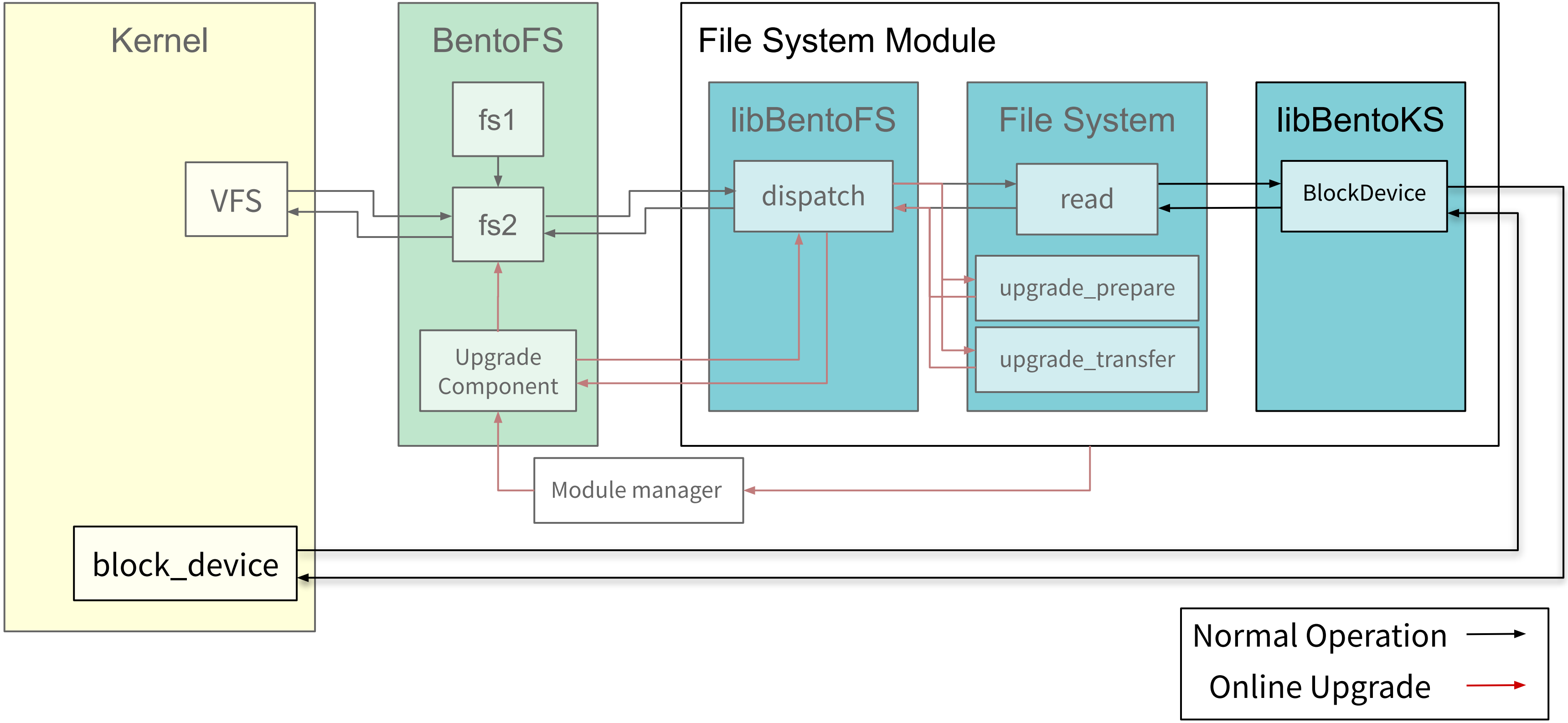
File System Module



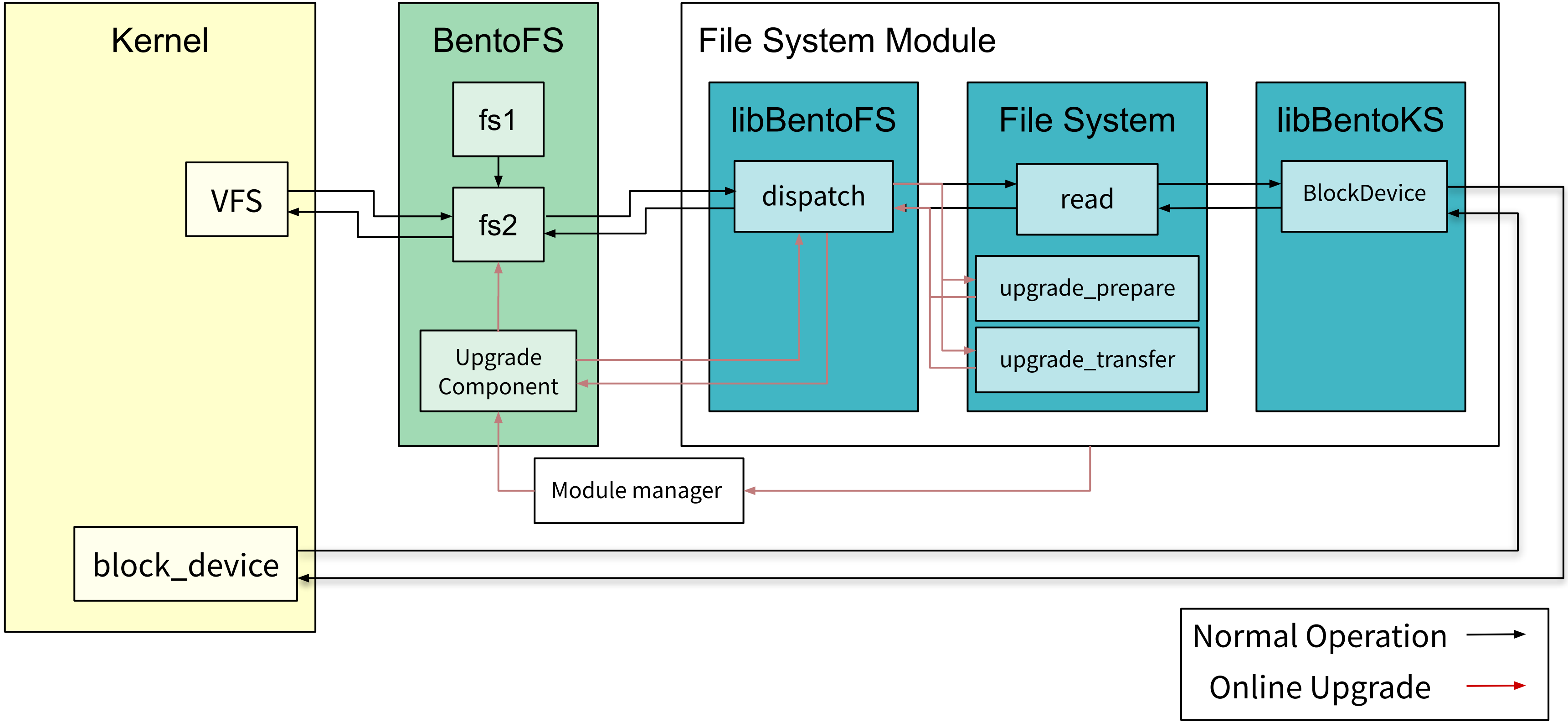
libBentoFS


Kernel
VFS
block_device
BentoFS
fs1
fs2
Upgrade Component
File System Module
libBentoFS
dispatch
File System
read
upgrade_prepare
upgrade_transfer
libBentoKS
BlockDevice
Module manager
Normal Operation
Online Upgrade

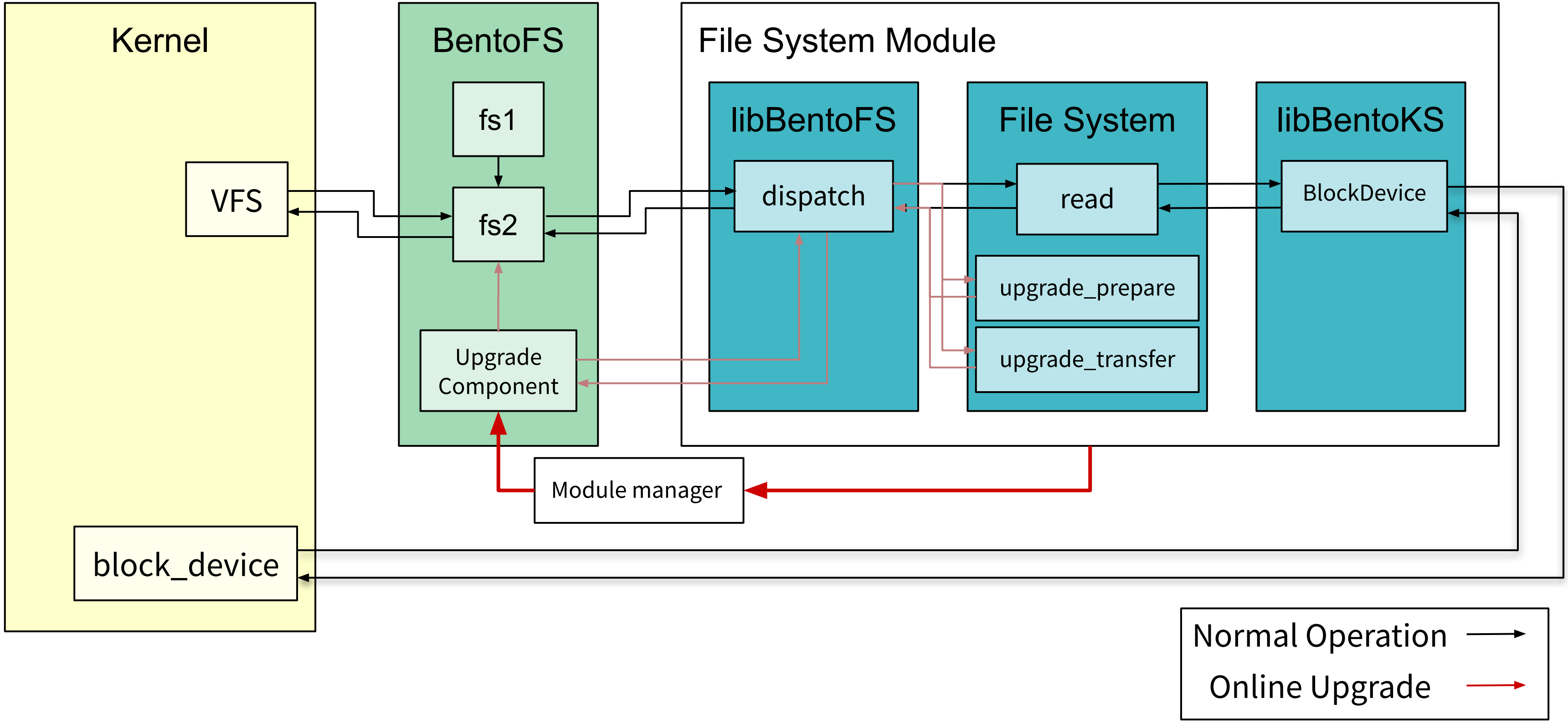
libBentoKS


Kernel
VFS
block_device
BentoFS
fs1
fs2
Upgrade Component
Module manager
File System Module
libBentoFS
dispatch
File System
read
upgrade_prepare
upgrade_transfer
libBentoKS
BlockDevice
Normal Operation
Online Upgrade

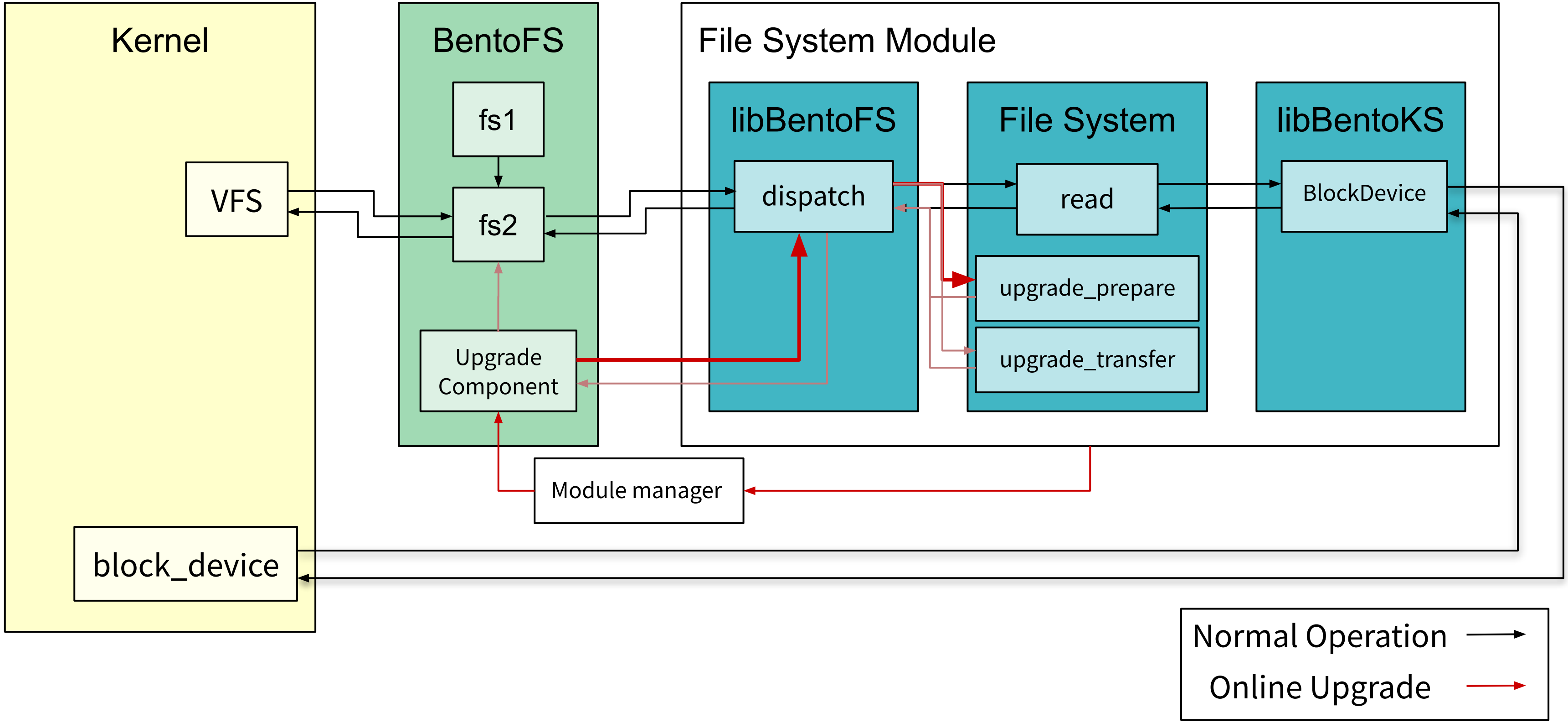
Live Upgrade



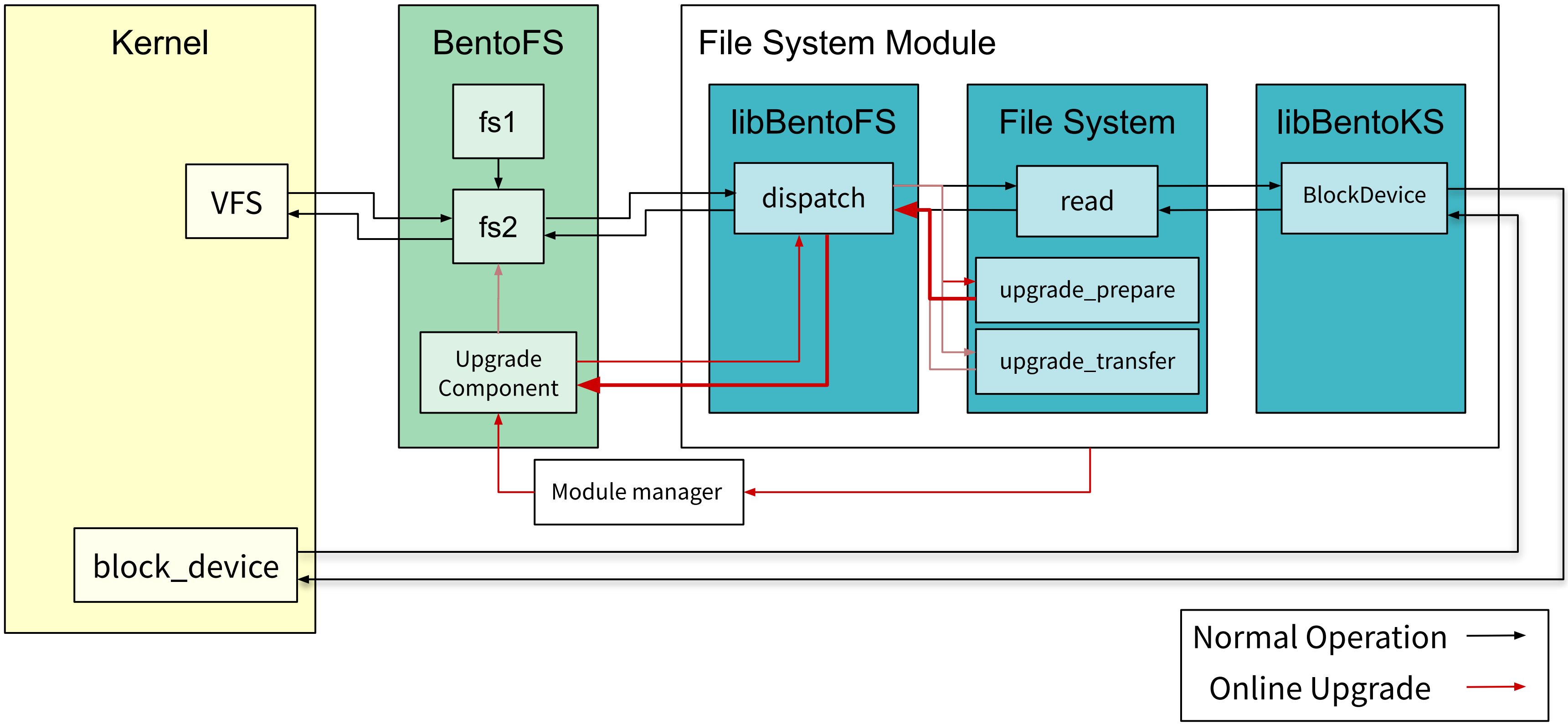
Live Upgrade



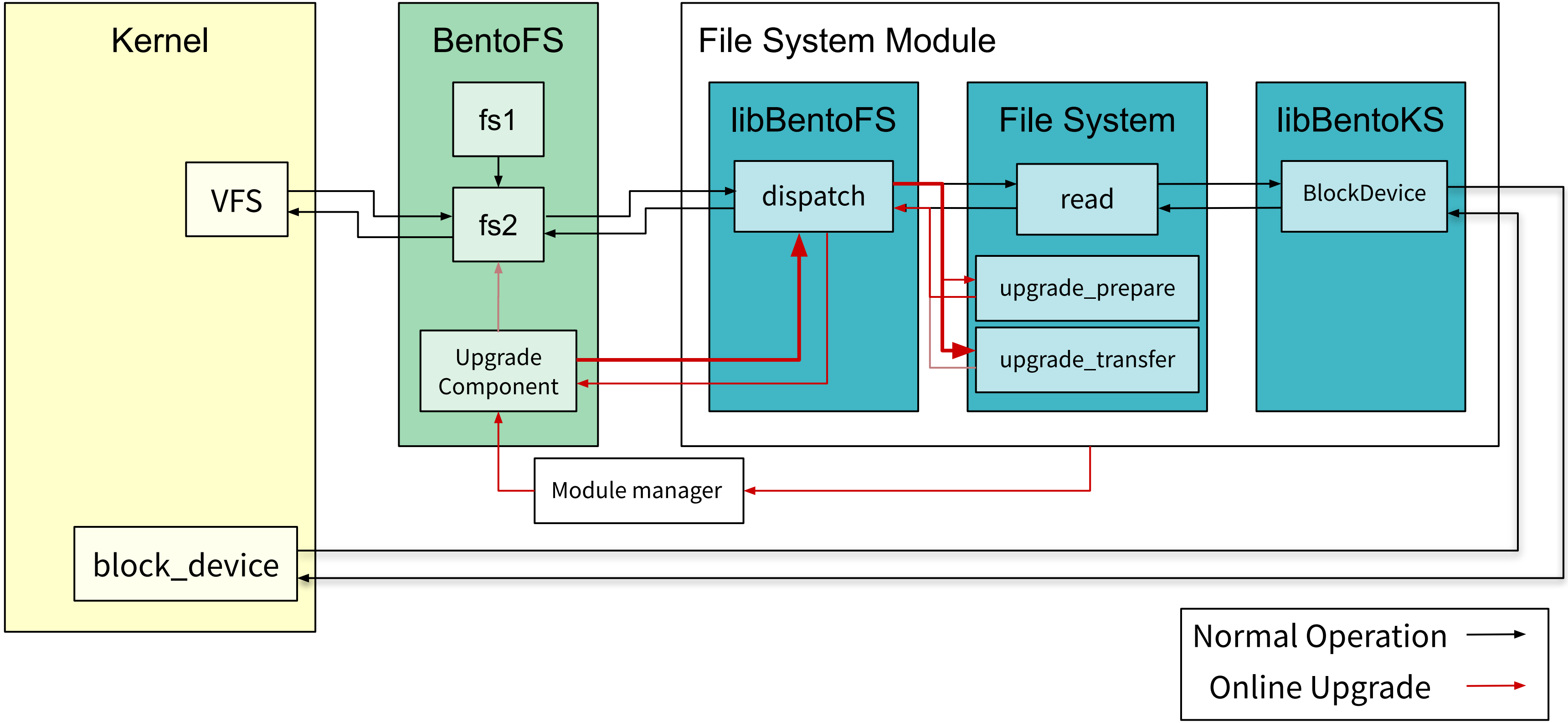
Live Upgrade



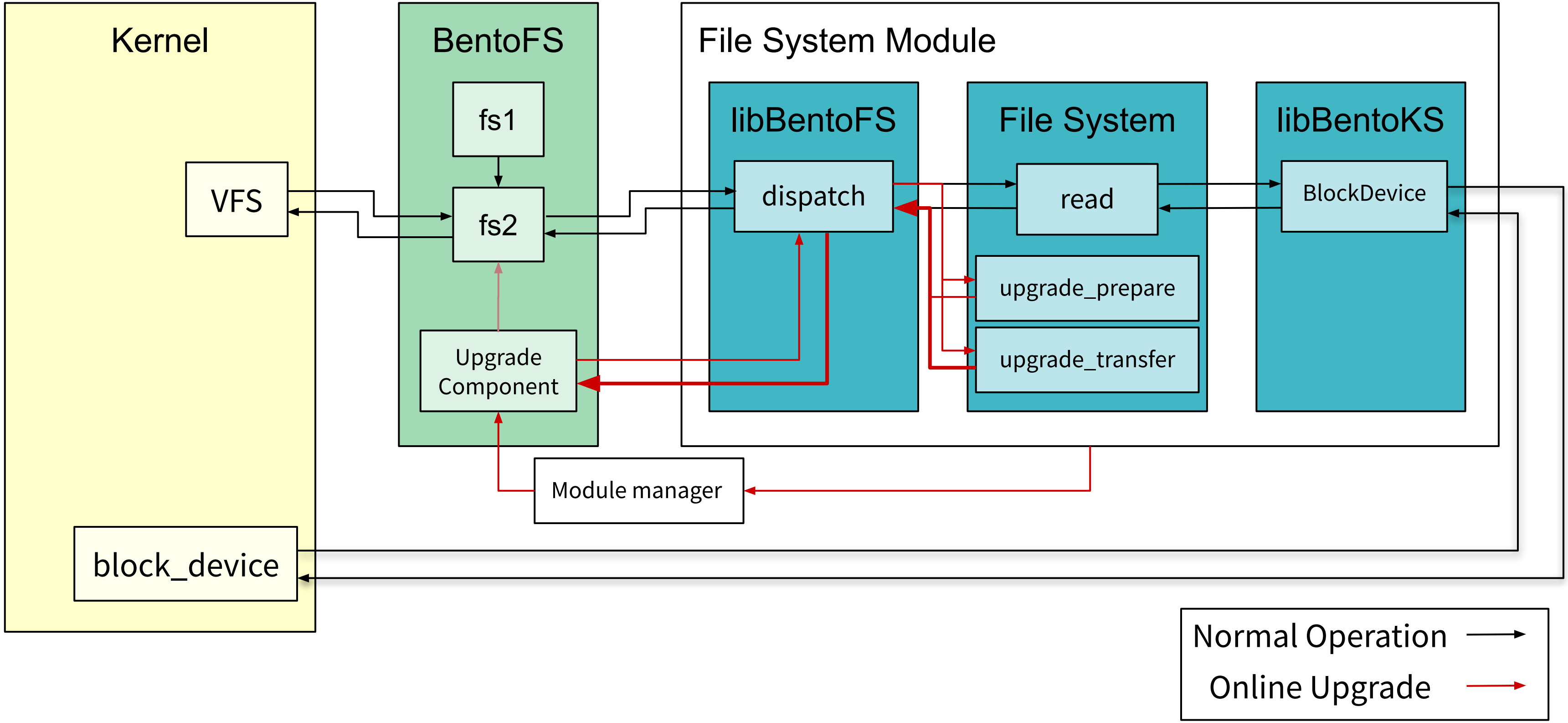
Live Upgrade



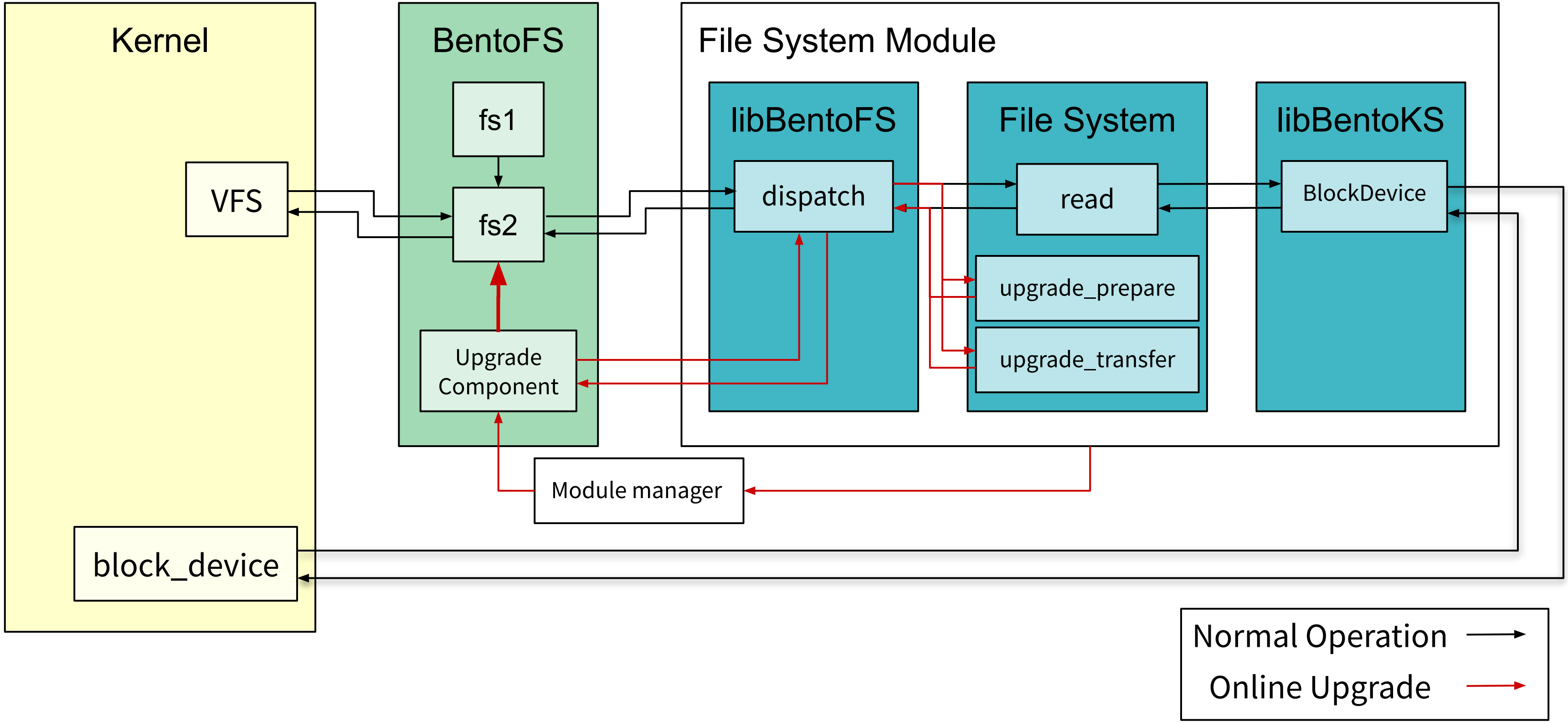
Live Upgrade



Live Upgrade



Live Upgrade



Safe Interfaces for a Safe Language

- Safe interfaces provided by Bento enable the file system to be written in safe Rust
- BentoFS and libBentofs translate the VFS interface to a safe Rust interface inspired by message passing interfaces
  - This interface is based on the FUSE low-level interface
- LibBentoKS provides safe wrappers around kernel services such as the buffer cache, kernel lock implementations, and the kernel TCP stack

Live Upgrade

- A live upgrade component in BentoFS manages the live upgrade process, allowing the old file system to transfer state to the new one and swapping function pointers
- When a file system upgrade module is inserted:
  a. The new file system registers itself with BentoFS as an upgrade
  b. BentoFS acquires a lock on the old file system to pause new operations
  c. BentoFS sends a message to the old file system to cause it to prepare for an upgrade
  d. The old file system performs necessary clean up and returns state to BentoFS
  e. BentoFS sends a message to the new file system with the state from the old one
  f. The new file system initializes itself using the state
  g. BentoFS updates function pointers to point to the new file system and releases the lock

Userspace Execution

- Bento file system can be run in userspace without any changes to the code
- Most interfaces provided by libBentoFS and libBentoKS are identical to existing userspace interfaces
  - We provide userspace implementations for the other interfaces
- File system can be compiled to run in userspace using a build flag

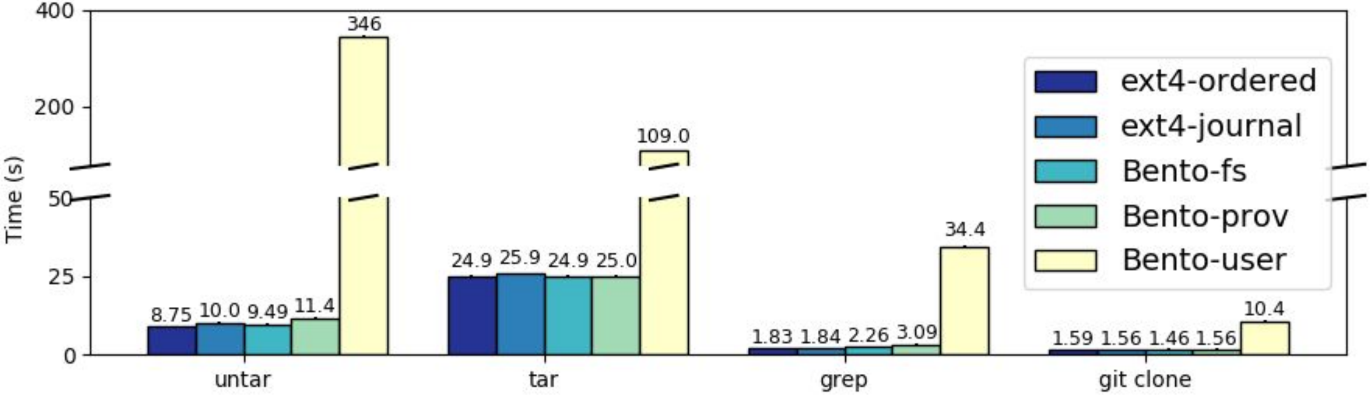
Implementation

- We used Bento to build a file system: Bento-fs
- Bento-fs is like xv6 but includes optimizations to be competitive with ext4
- 3038 lines of safe Rust code
- Passes all seq-2 CrashMonkey crash consistency tests
- Also implemented Bento-prov, a version of Bento-fs with file provenance tracking

Evaluation

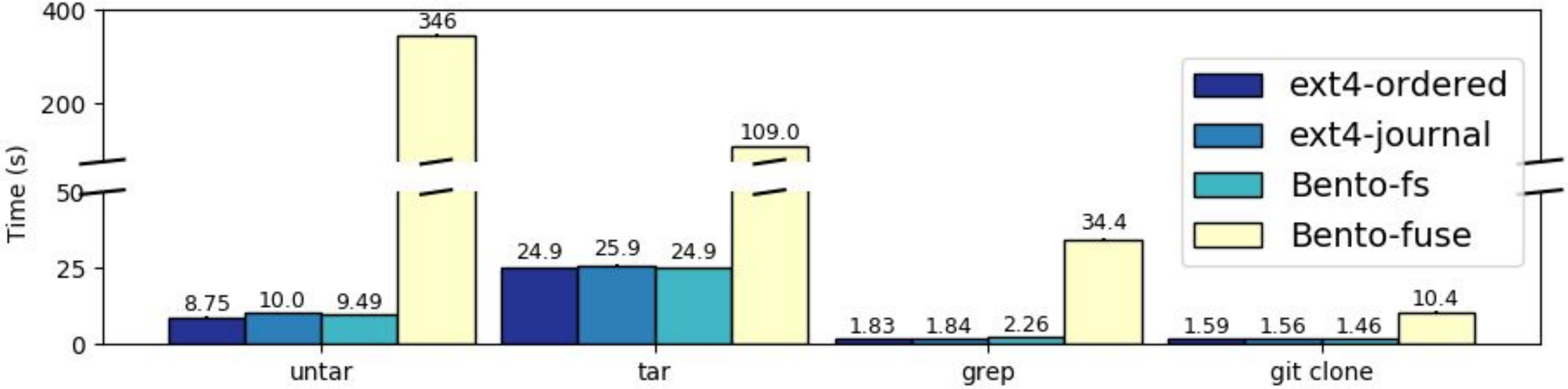
- Can Bento support competitive performance?
- How does live upgrade impact availability?
- Machine setup
  - Intel Xeon Gold 6138 CPU using 40 hyperthreads
  - 96 GB DDR4 RAM
  - Intel Optane 900P Series NVMe SSD with 2.5 GB/s read speed and 2 GB/s write speed
- Baselines
  - Ext4 with data journaling (`data=journal`)
  - Ext4 with default metadata journaling (`data=ordered`)

Filebench, Redis & RocksDB, Apps



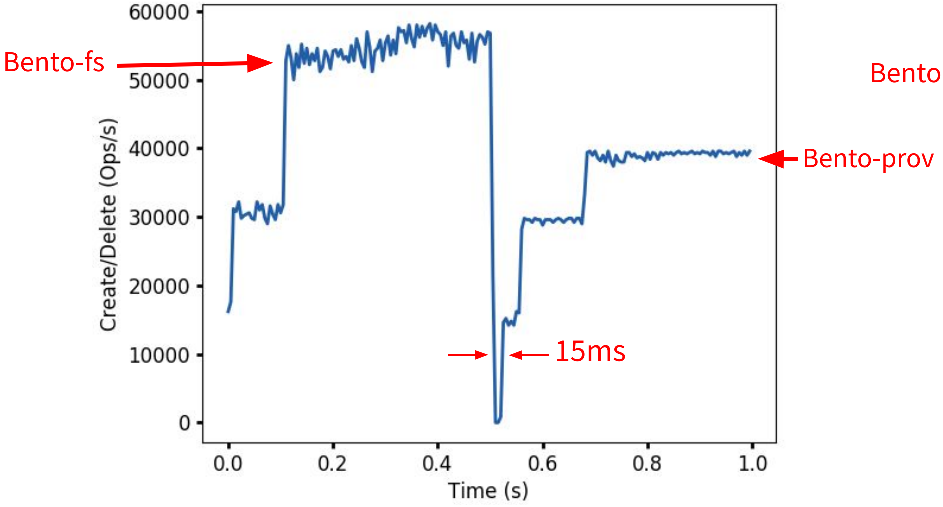
Application Benchmarks

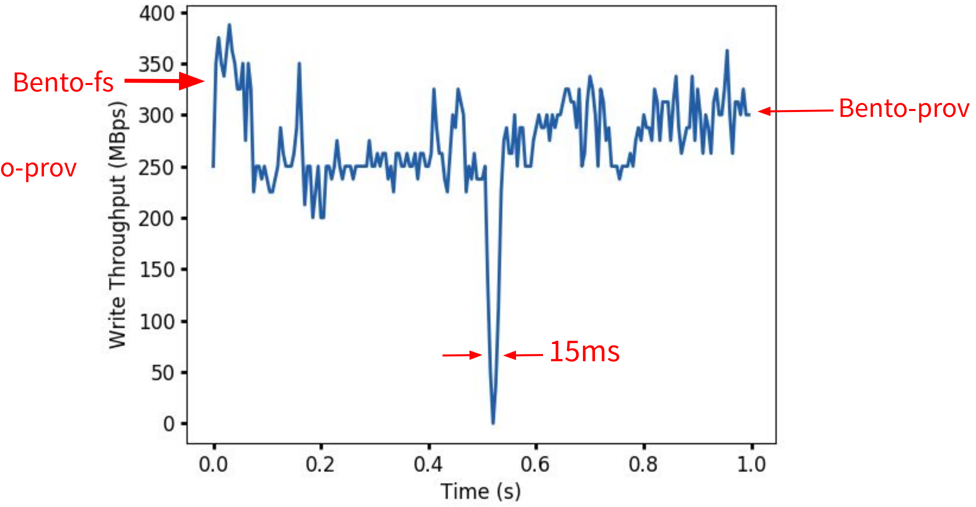
- Untar, tar, and grep on Linux. Git clone on xv6
- Bento-fs similar to both ext4
- FUSE file system is much slower



Live Upgrade Evaluation

- Benchmarks: Pairs of creates and deletes and synced writes with 10 threads
- Performance recovers after 15ms of downtime





Conclusion

- Built Bento, a framework for high-velocity Linux kernel file systems
- A Bento file system performs competitively with ext4 and can be upgraded with only 15ms of downtime
- Code: https://gitlab.cs.washington.edu/sm237/bento

Thank you

Contact: sm237@cs.washington.edu