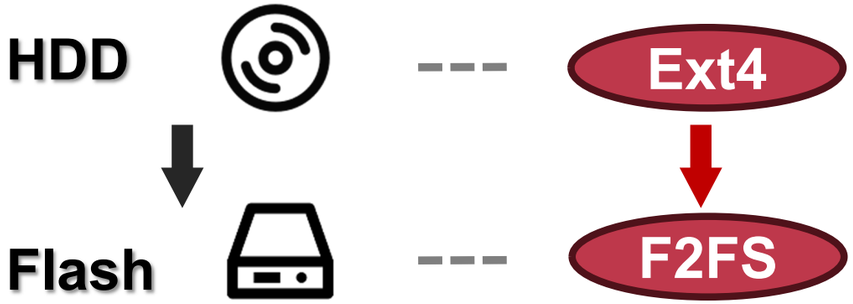
Sharpen the Spec, Cut the Code: A Case for Generative File System with SYSSPEC

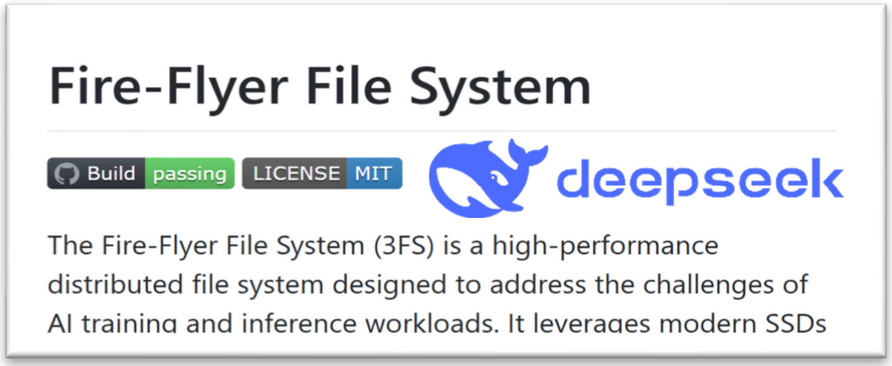
Qingyuan Liu, Mo Zou, Hengbin Zhang, Dong Du, Yubin Xia, Haibo Chen

IPADS · Shanghai Jiao Tong University

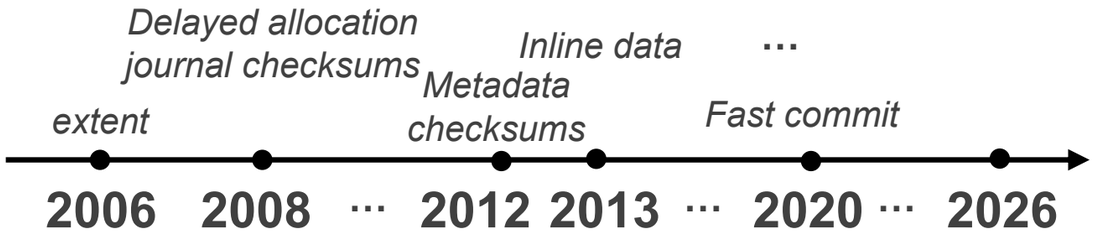
File System Keeps Evolving



File systems for new hardware



File systems for new application

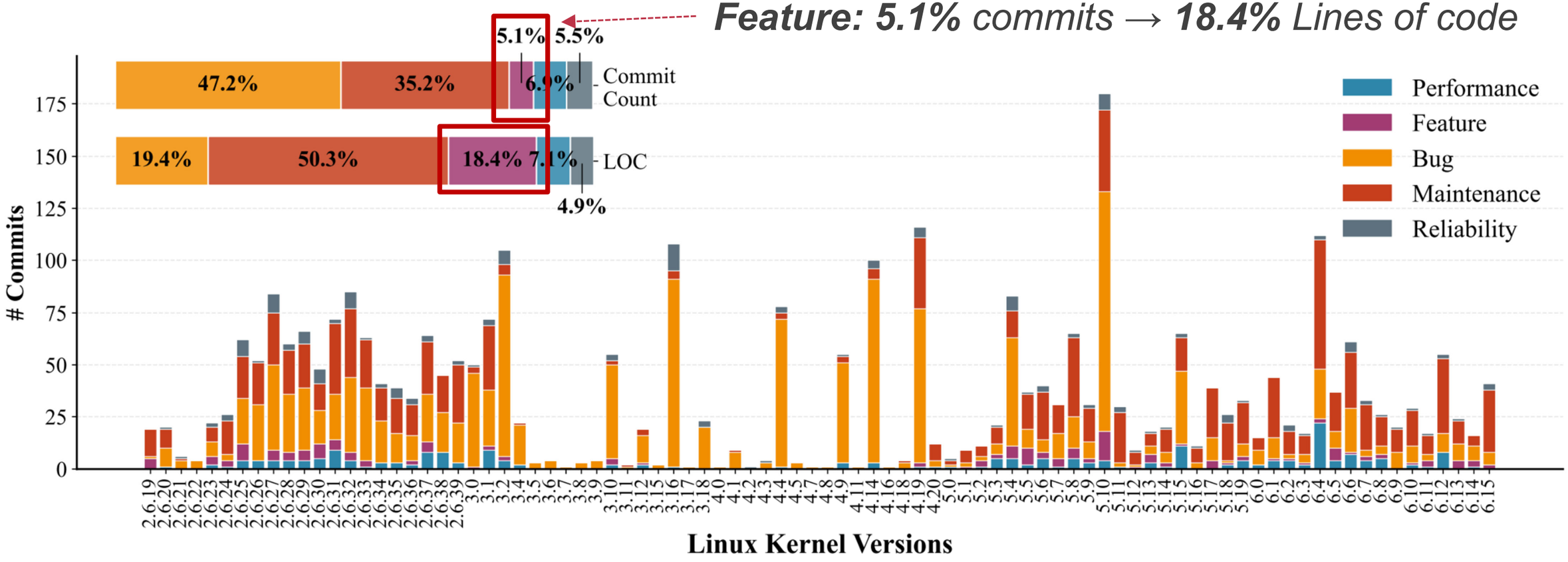


Mature file systems keep evolving

The Effort of File System Dev/Evo

*From Linux 2.6.19 to 6.15: ~**3,157** ext4-related commits*

***Feature: 5.1%** commits → **18.4%** Lines of code*

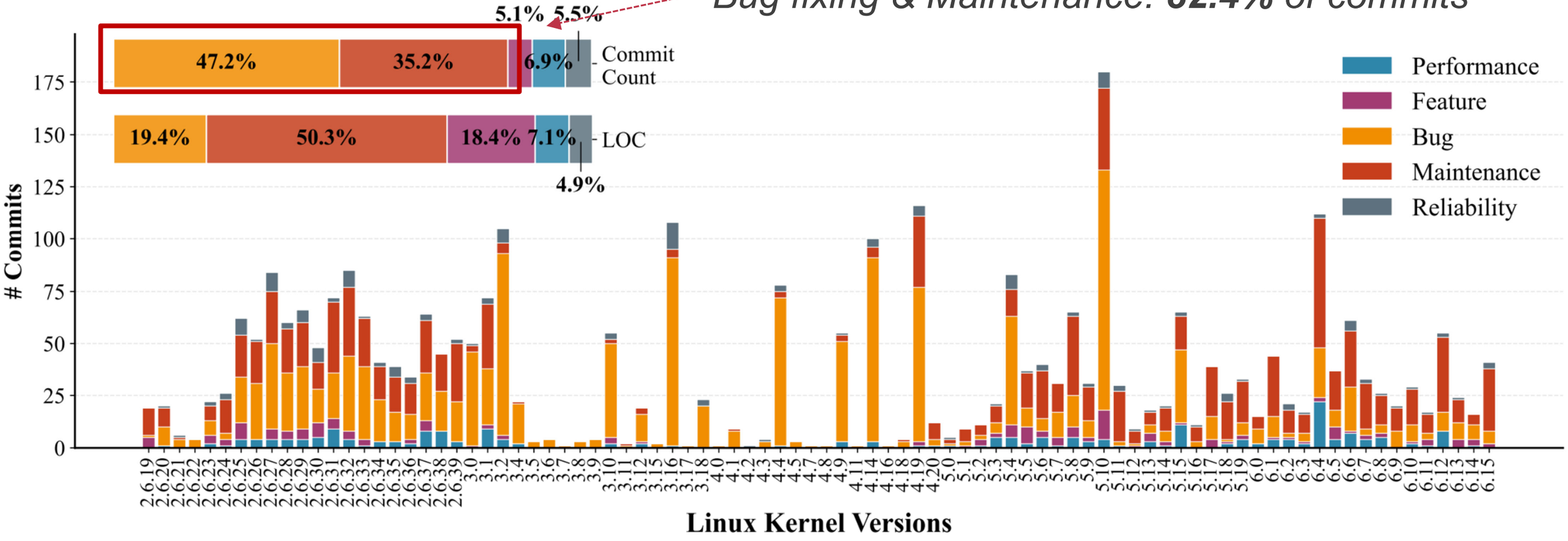


- Efforts on new features: **extensive coding**

The Effort of File System Dev/Evo

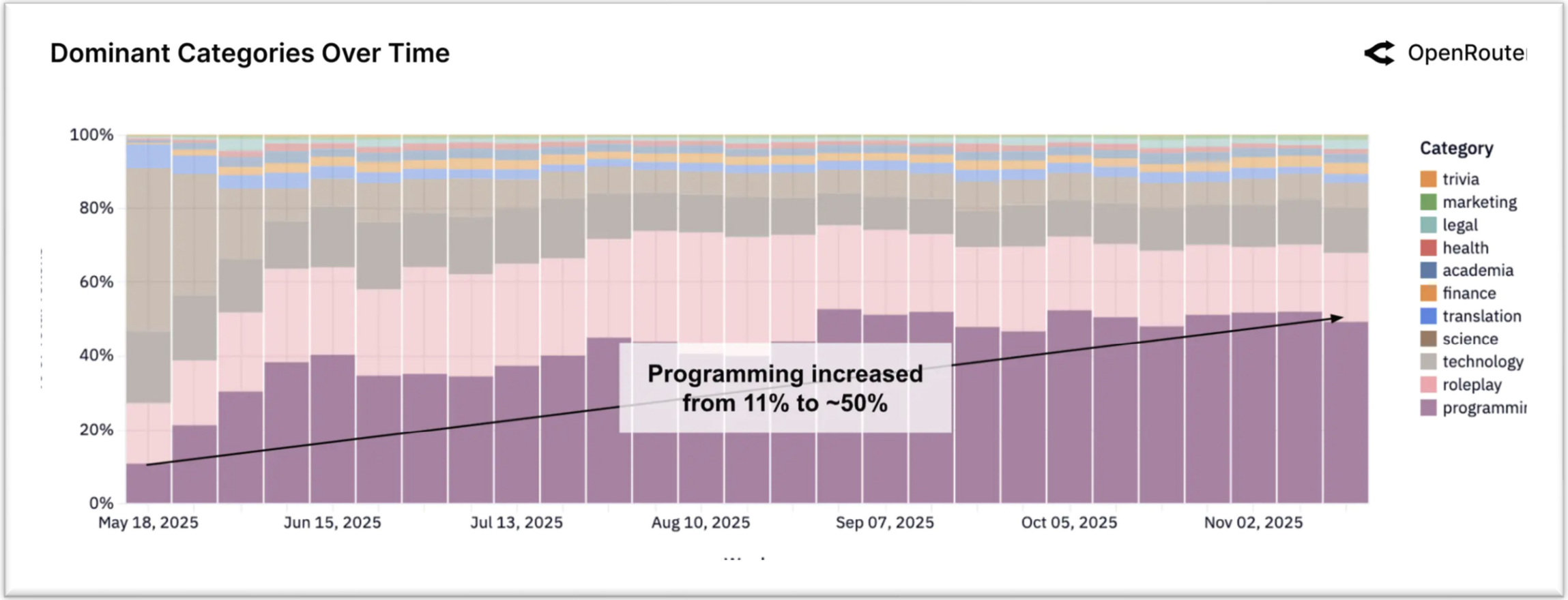
*From Linux 2.6.19 to 6.15: ~**3,157** ext4-related commits*

*Bug fixing & Maintenance: **82.4%** of commits*



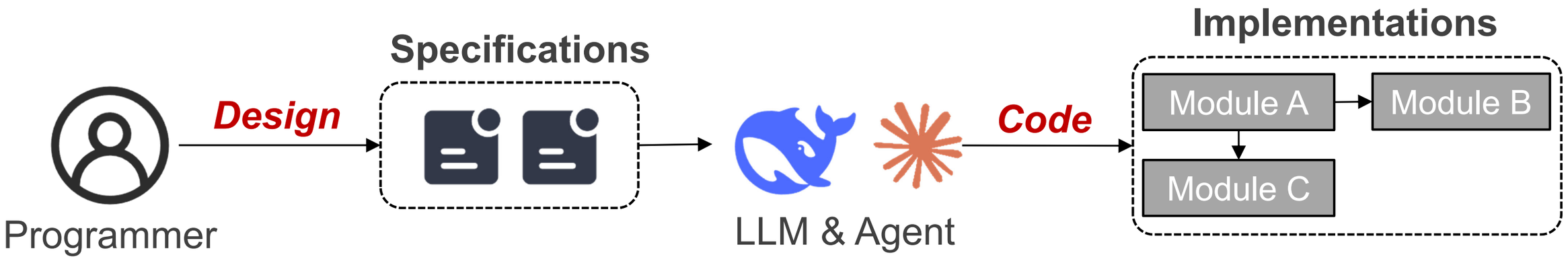
- Efforts on **long tail** maintenance and bug fixing

Rapid Advancement of LLM Coding



Reduce the efforts for file system Dev/Evo using the LLM?

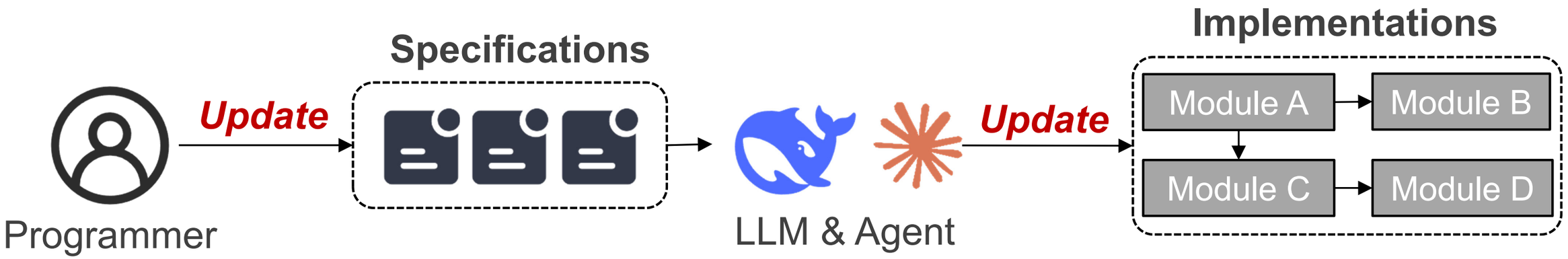
Our Vision: Generative File System



Write the file system with specifications

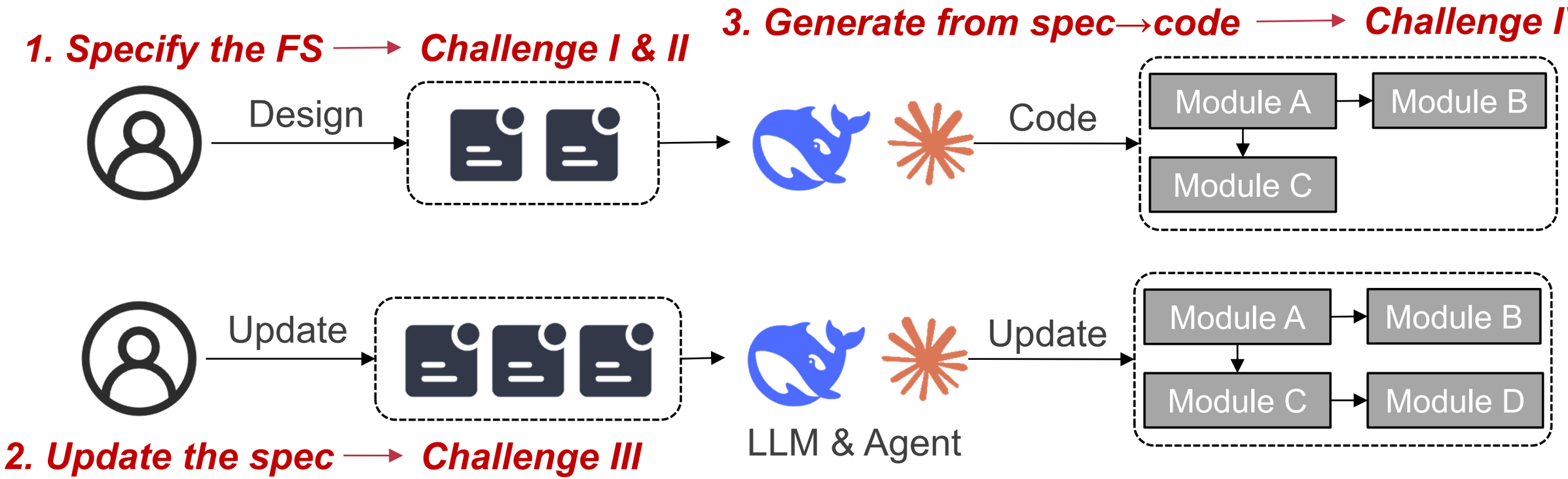
Leave the coding to LLMs

Our Vision: Generative File System



Update the specifications to evolve the file system

Challenges for Generative File System



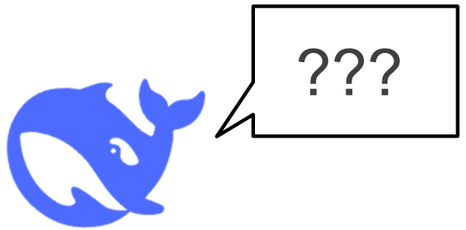
At least four challenges to achieving robust code generation

Challenges for Generative File System

- **Challenge I: Semantic Gaps**
  - Lacking a systematic methodology for specifying the **functionality** of programs

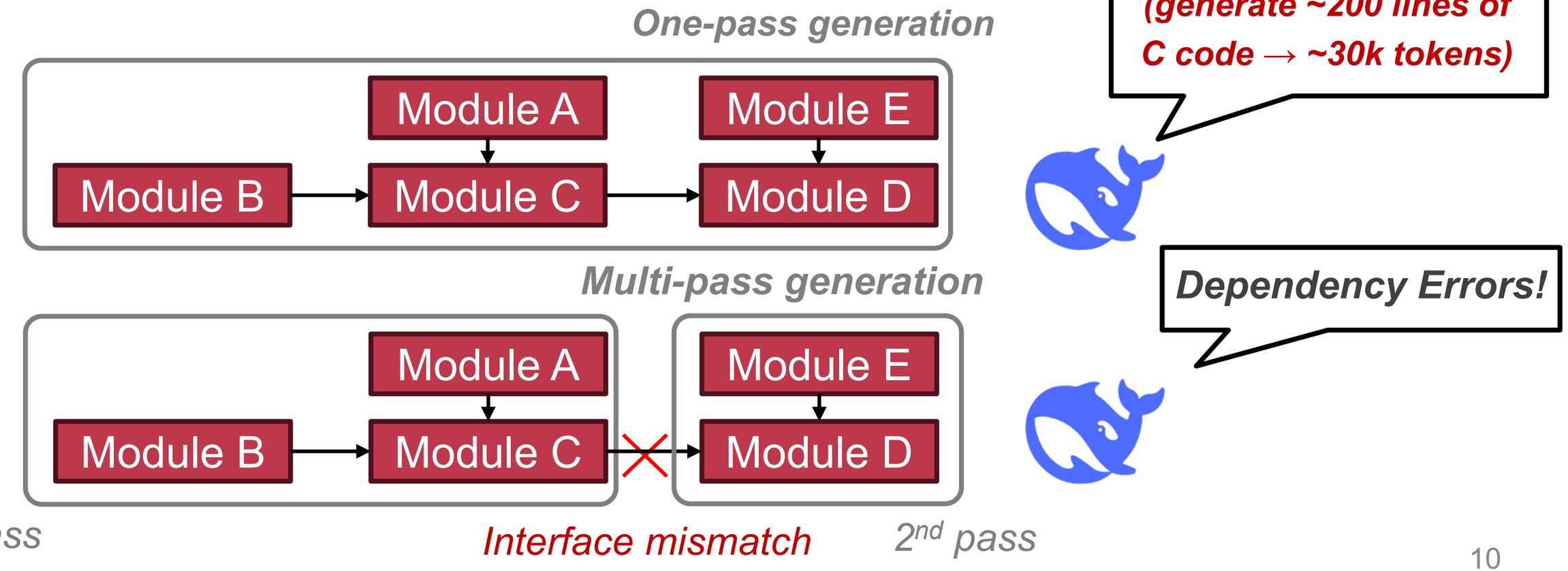
    *(Including **sequential** and **concurrent** logic)*

"Help me generate a file system"
- "No dependency error"
- "Avoid race conditions with locking"
- ……



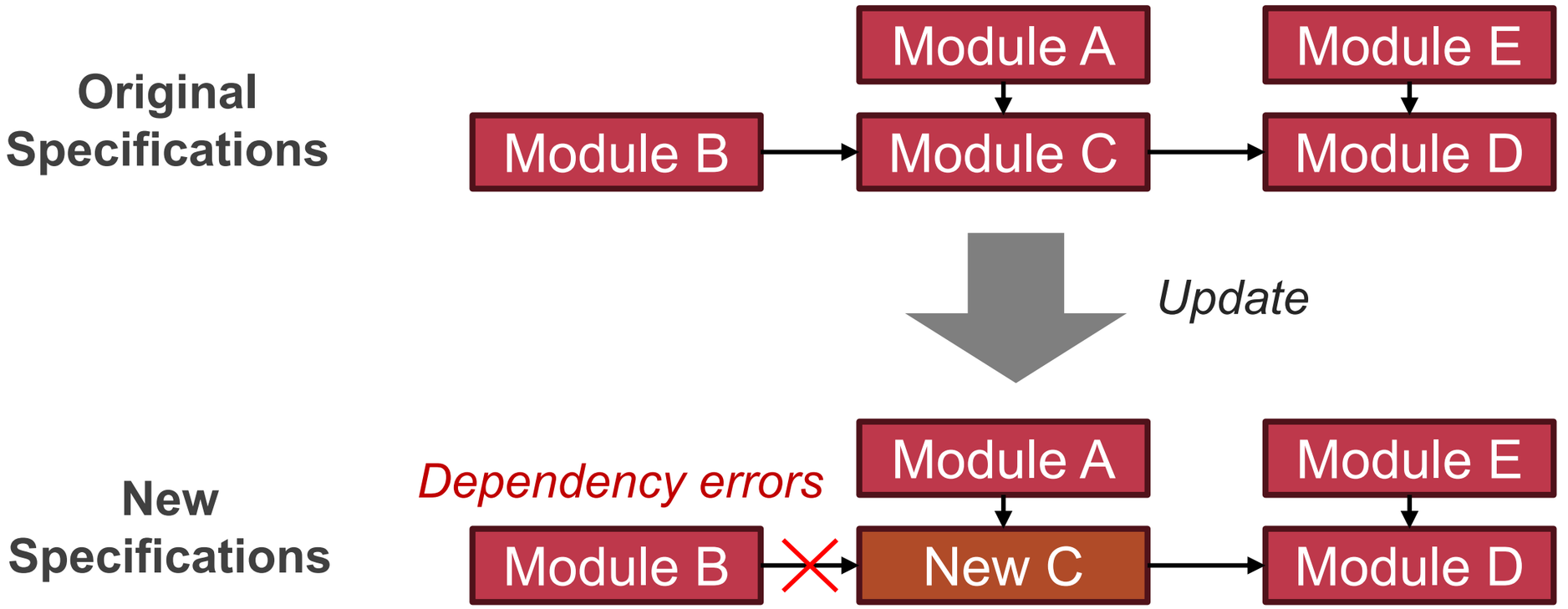
Challenges for Generative File System

- **Challenge II: Complicated Component Composition**



Challenges for Generative File System

- **Challenge III: Backward Compatibility**
  - How to avoid cascading dependency issues after specification updates?



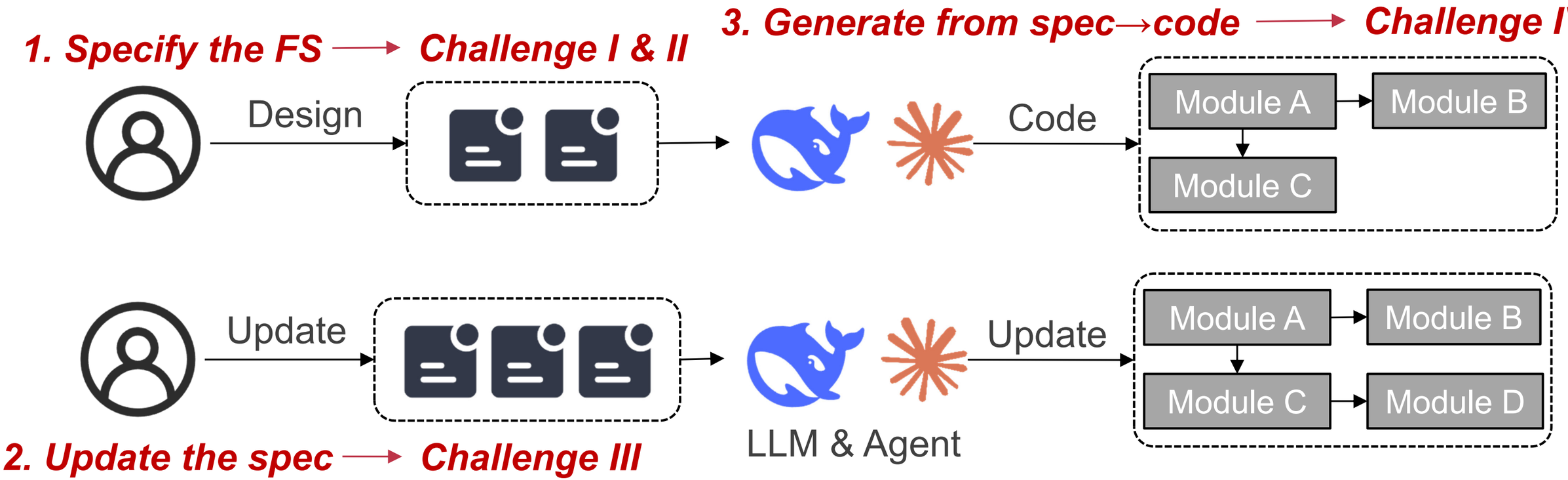
Challenges for Generative File System

- **Challenge IV: Unreliable LLM Capability**
  - How to reduce hallucinations and enhance the stability of code generation?

```
int atomfs_write(char* path[], const char* buf, unsigned size, unsigned offset) {
    // ❌ ERROR: Hallucinated safe state. Forgot lock(root_inum);
    struct inode *inum = locate(root_inum, path);
    if (inum == NULL) return -1;

    int res = check_file(inum);
    if (res == 1) return -1;

    int ret = inode_write(inum, buf, size, offset);
    unlock(inum);

    return ret;
}
```

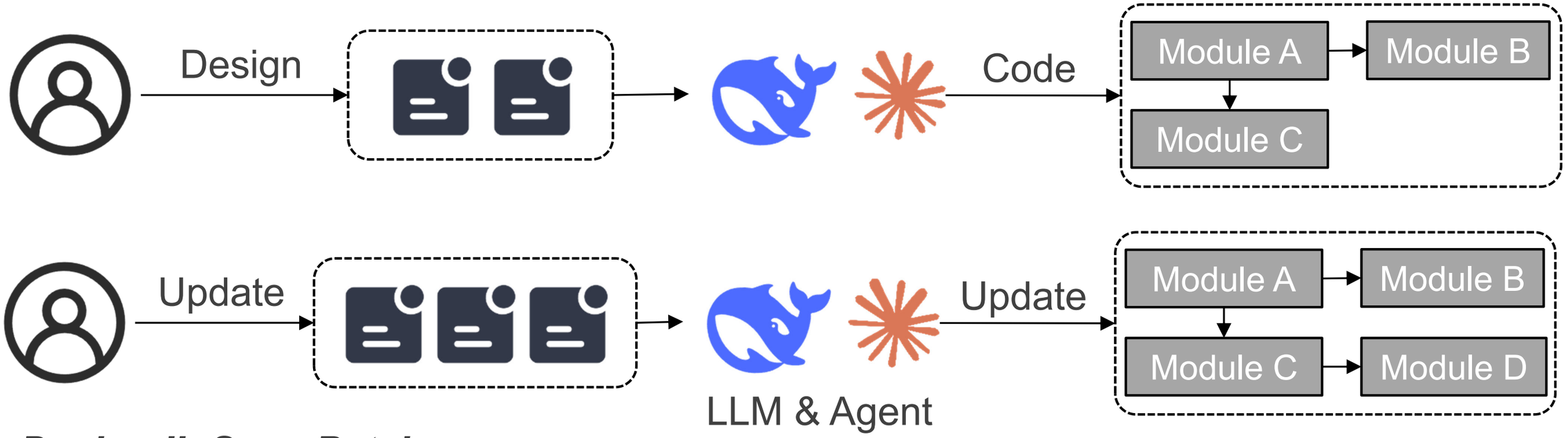
Occasionally happens

Challenges for Generative File System



SysSpec: Design Overview

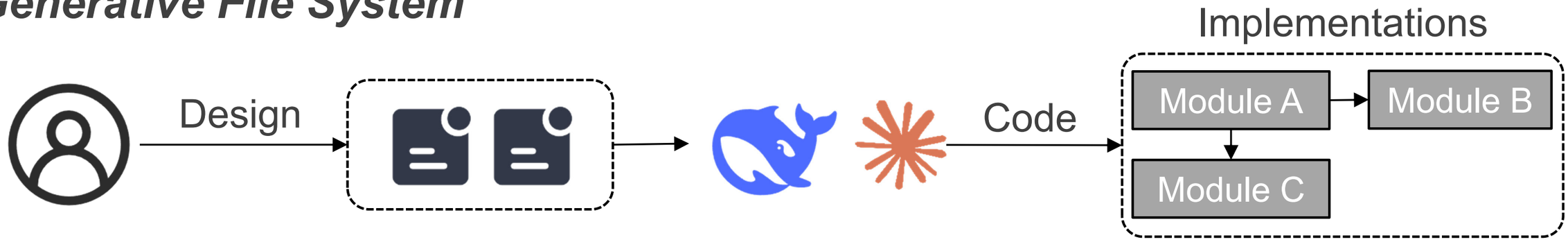
- ***Design I: SysSpec Specification***
  - *Addressing Challenge I & II*
- ***Design III: SysSpec Toolchain***
  - *Addressing Challenge IV*



- ***Design II: Spec Patch***
  - *Addressing Challenge III*

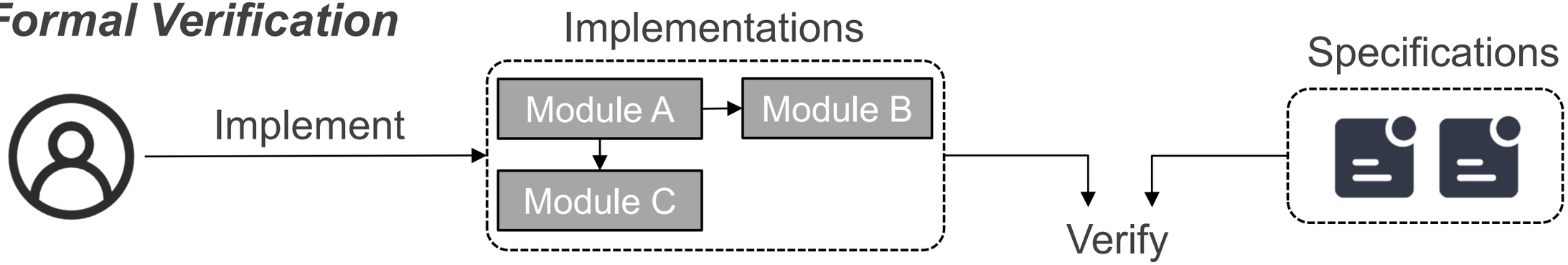
SysSpec Specification: Key Insight

Generative File System



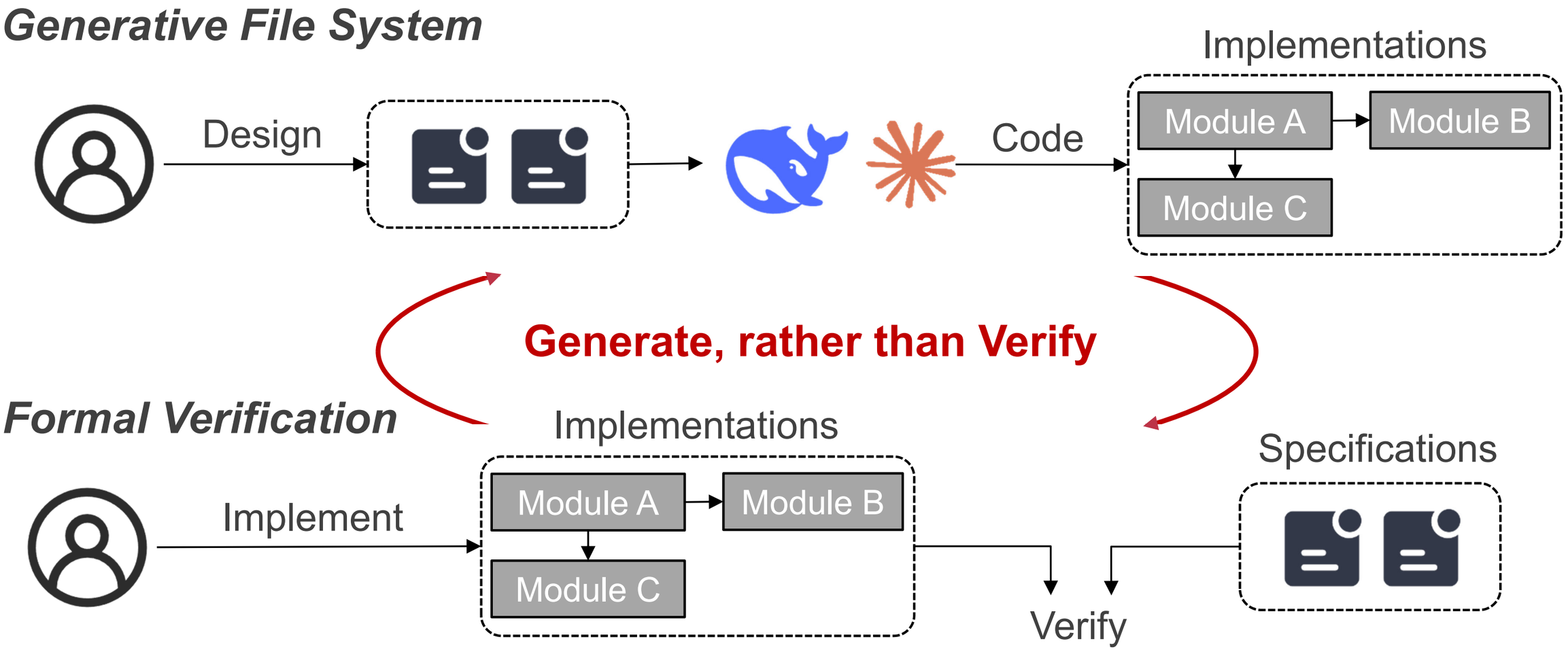
Generate non-existence Impl with Spec

Formal Verification



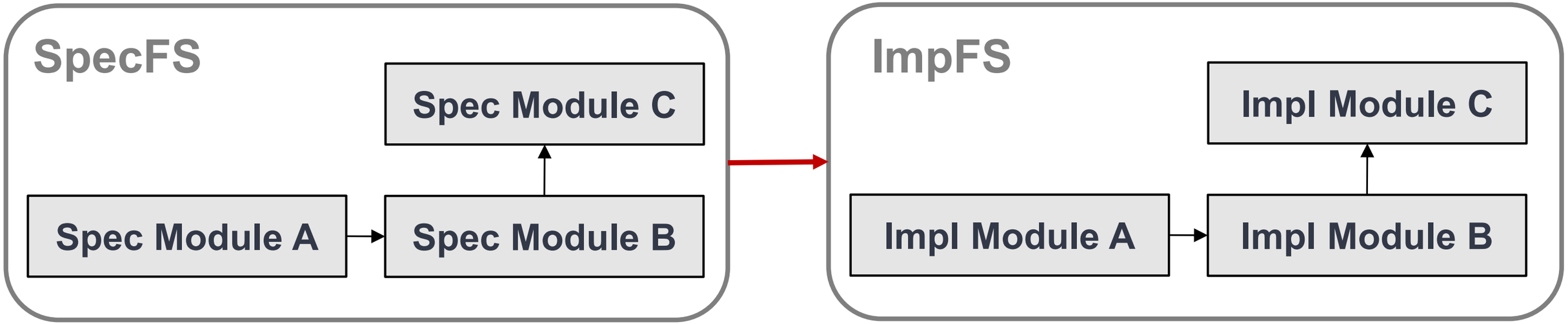
Verify existence Impl with Spec

SysSpec Specification: Key Insight



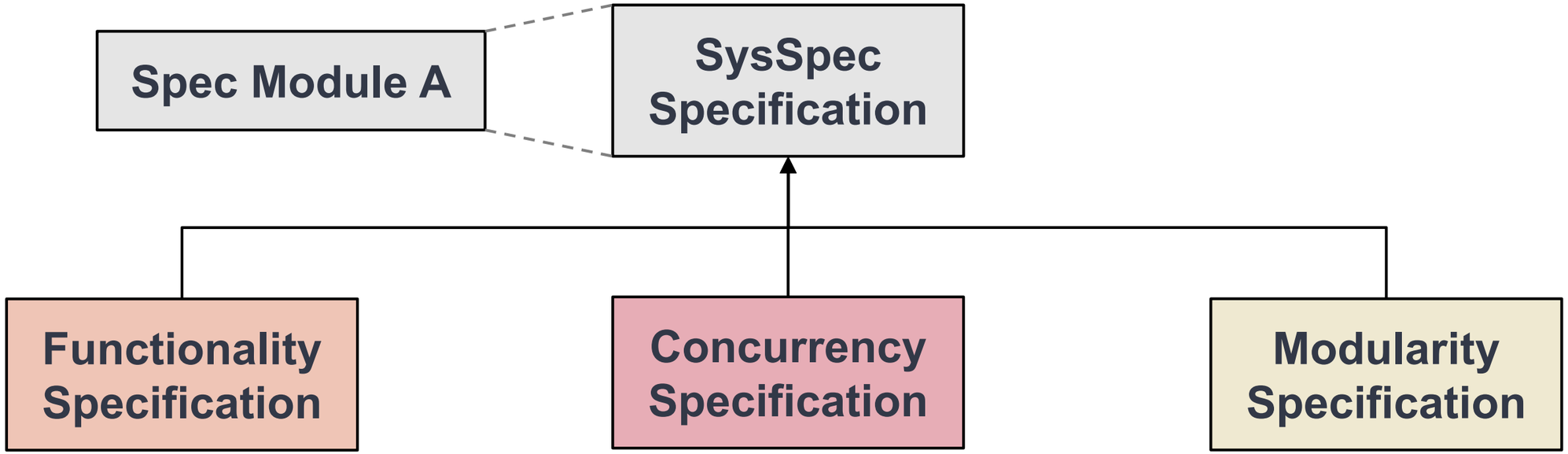
Borrow methodologies from formal verification

SysSpec Specification: Basic Unit



Module: basic unit of specification and code generation

SysSpec Specification Structure



SysSpec Specification: consists of 3 types of specifications

SysSpec Specification Structure

- **Functionality Specification**
  - Define sequential functionality of each module.
- **Concurrency Specification**
  - Specify concurrent behavior and interactions.

Address Challenge I

- **Modularity Specification**
  - Describe inter-module composition and dependencies.

Address Challenge II

Functionality Specification

[Pre-condition]:
Required state before execution

[Post-conditon]:
Guaranteed state upon completion

[Invariants]:
properties that must hold true across state transitions

Hoare Style Specification

```
[Pre-condition]
    path: a NULL-terminated string array
    name: a valid string
[Post-condition]
    Case 1 Successful traversal and insertion
        - New inode created
        - Entry inserted into target directory
        - Return 0
    Case 2 Traversal or insertion failure
        Return -1
[Invariants]
    The root inum always exists
```

Functionality Specification of `atomfs_ins`

Borrow Hoare logic from formal verification

Optional Semantics of Intent & System Algorithm: details in the paper

Concurrency Specification

The state of locking of each function call

```
[Locking Specification of locate]
    Pre-condition: cur is locked.
    Post-condition: suppose return target.
    - If target is NULL: no lock owned
    - If target is not NULL: only target is owned
[Locking Specifications of check_ins]
    ……
[Locking Specification of atomfs_ins]
    ……
```

Concurrency Specification of `atomfs_ins`

Focus on concurrent behavior

Modularity Specification

[Rely]:
Predefined external dependencies

[Guarantee]:
Behavior guaranteed by the implementation

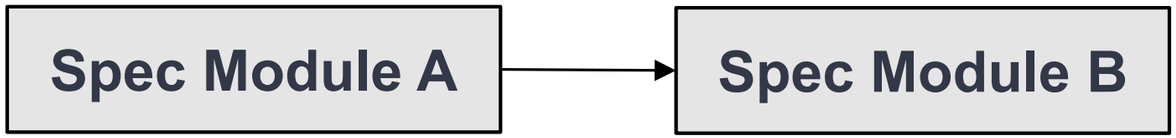
Constrained module sizes: typically <500 LoC

```
[Rely]
    struct inode {…};
    struct inode* root_inum;
    void lock(struct inode*);
    void unlock(struct inode*);
    struct inode* locate(struct inode* cur, char*
    path[]); // Traverse path under cur
    void insert(struct inode*, struct inode*,
char*) // …
[Guarantee]
int atomfs_ins(char*[], char*, int, unsigned,
unsigned)
```

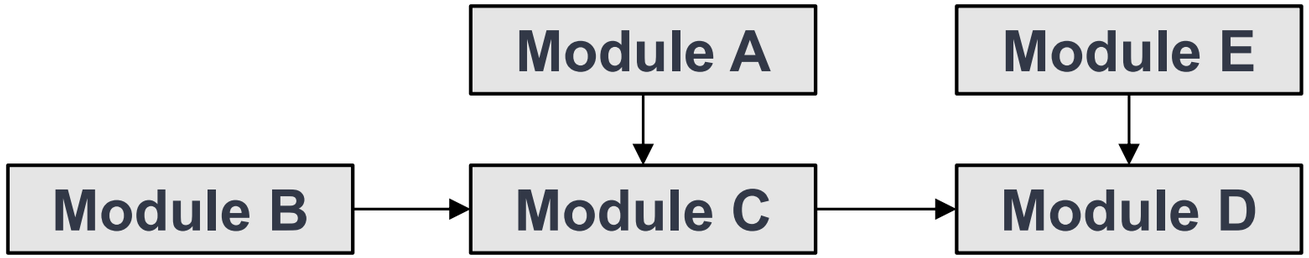
Modularity Specification of `atomfs_ins`

Borrow the rely/guarantee from formal verification

Inherent Dependencies through Modularity Specification

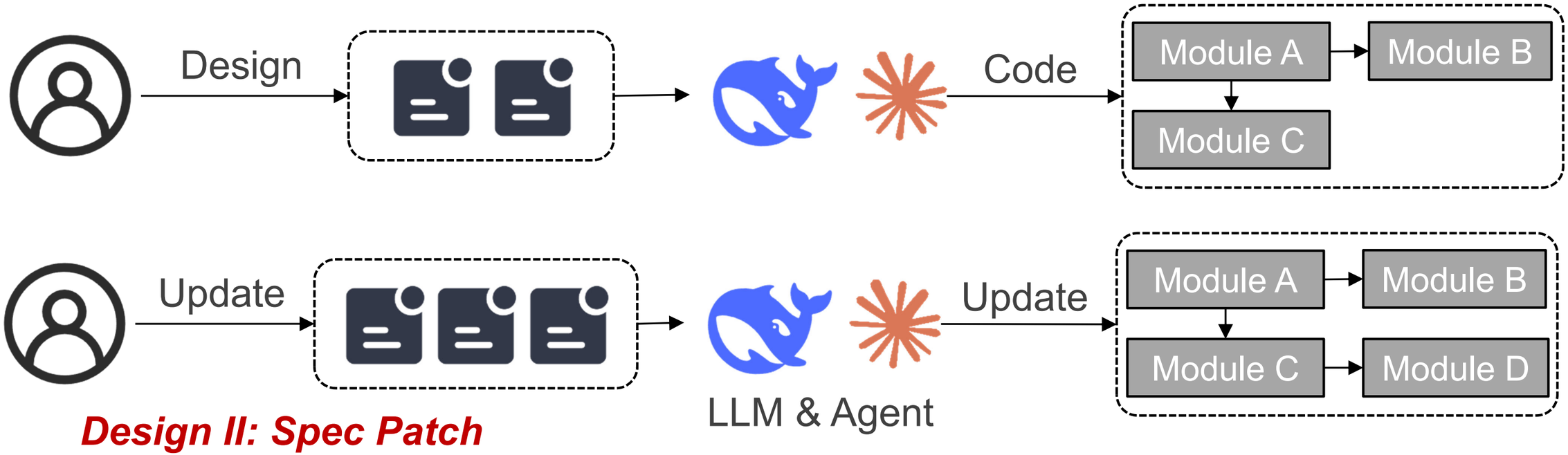


“Module B relies on Module A’s guarantee”



Construct SpecFS with DAG-structured dependencies

Spec Patch: From Construction to Evolution

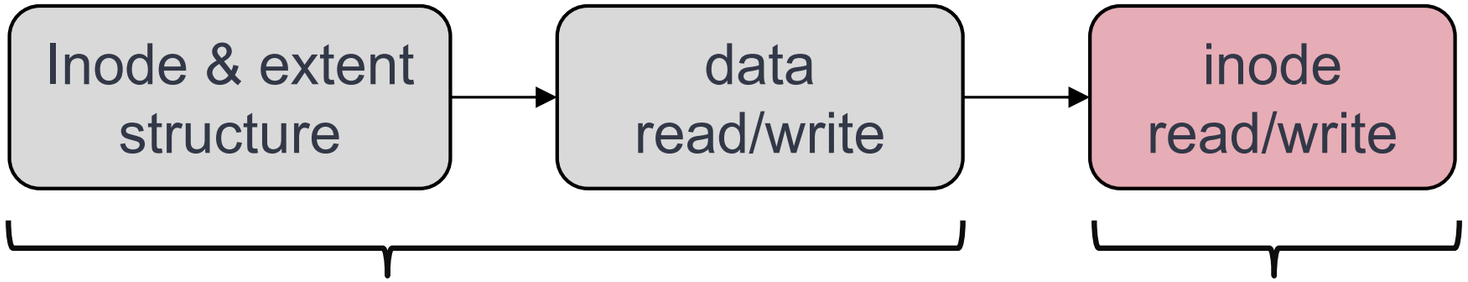


Design II: Spec Patch

From Construction to Evolution

- **The DAG structure of specifications is natural for evolution**
- **Organize new/modified specifications using DAG**
  - Merge into original specifications **atomically**

Example: part of the spec patch implementing **`Extent`**



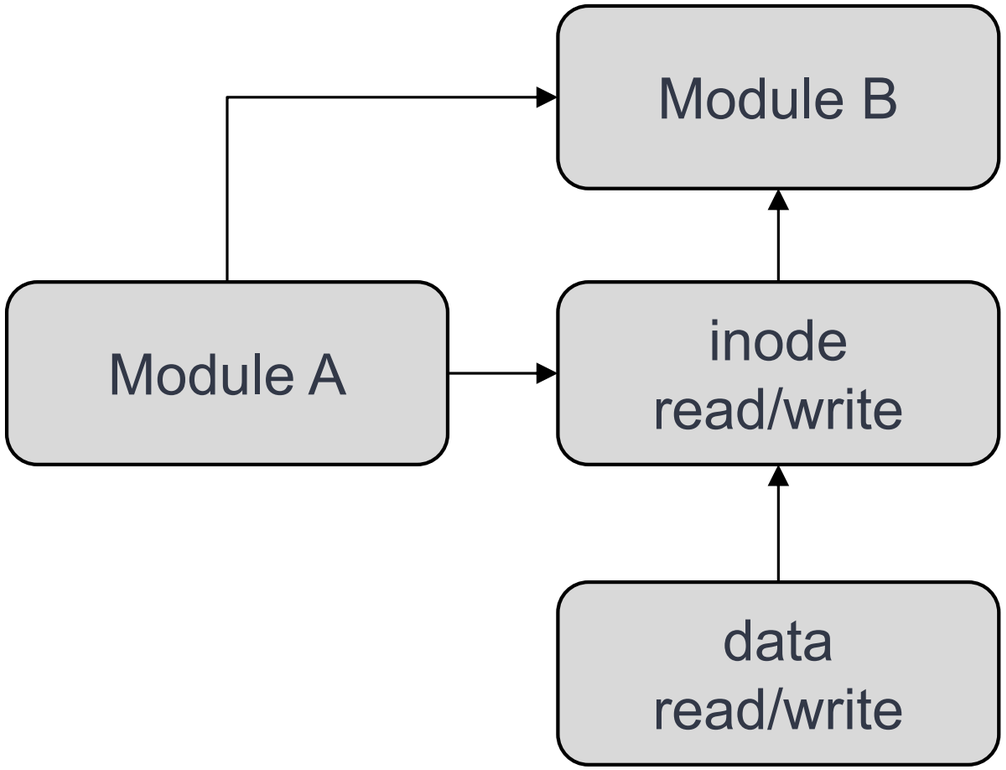
Non-root Node:

Implement new `guarantees`

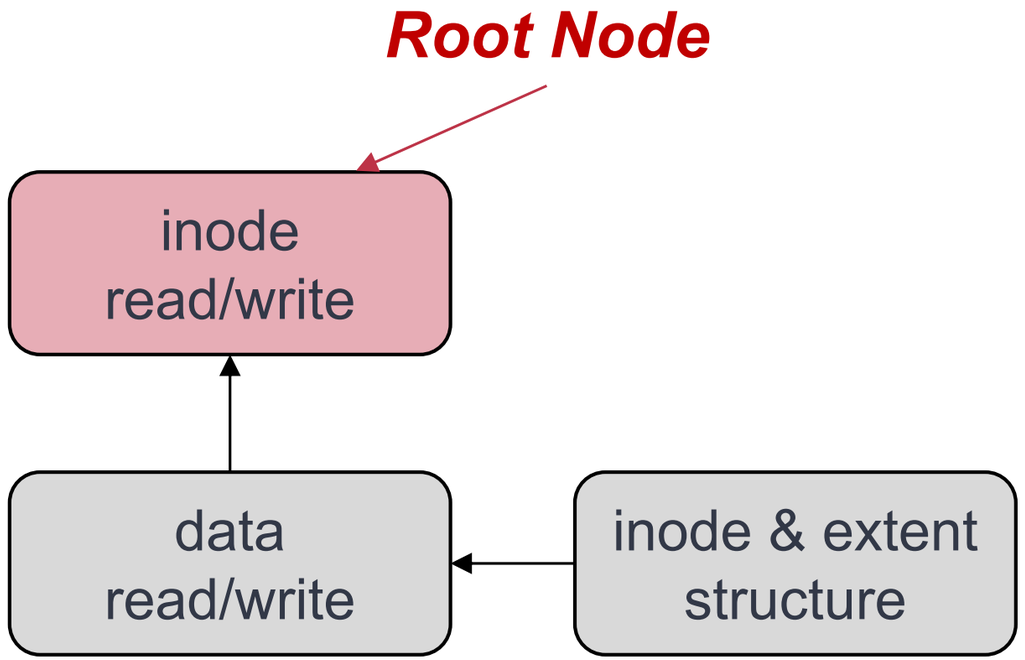
Root Node:

Rely on new `guarantees` but its `guarantees` do not change

Evolution Process: Spec Patch Merging

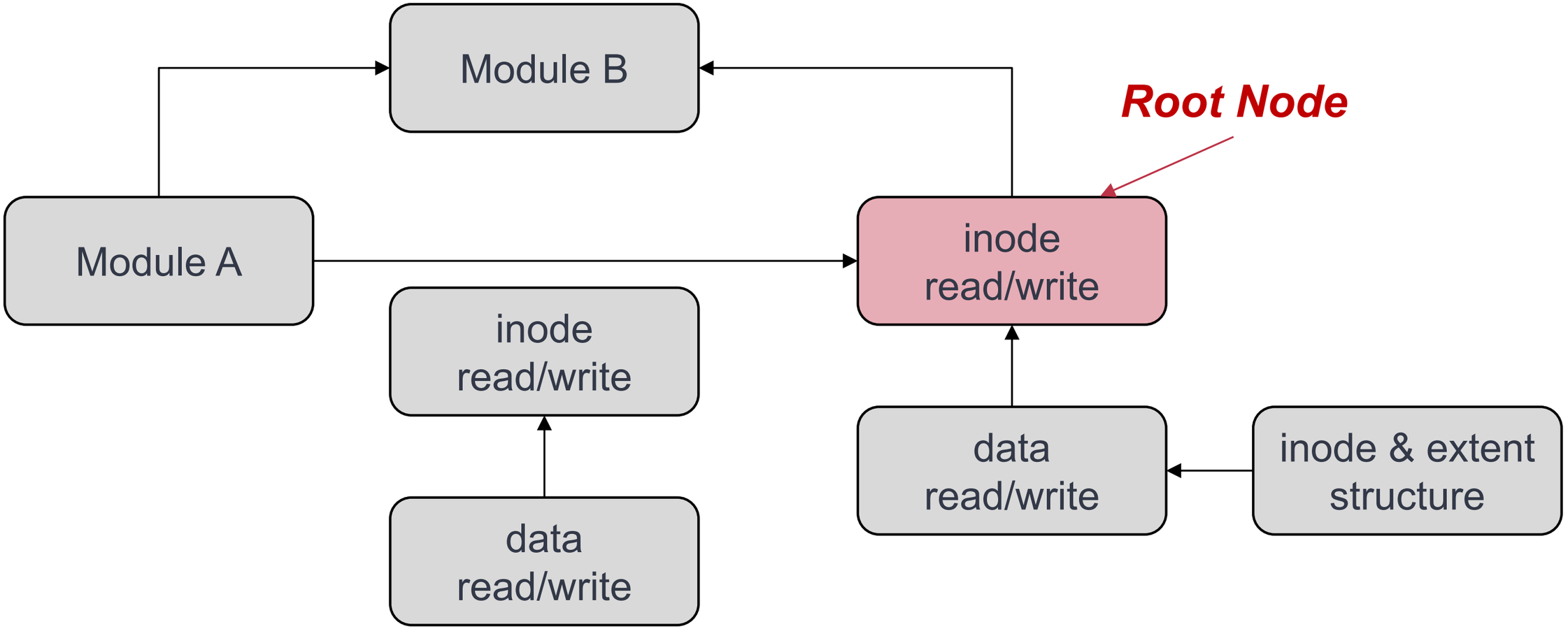


Original Spec

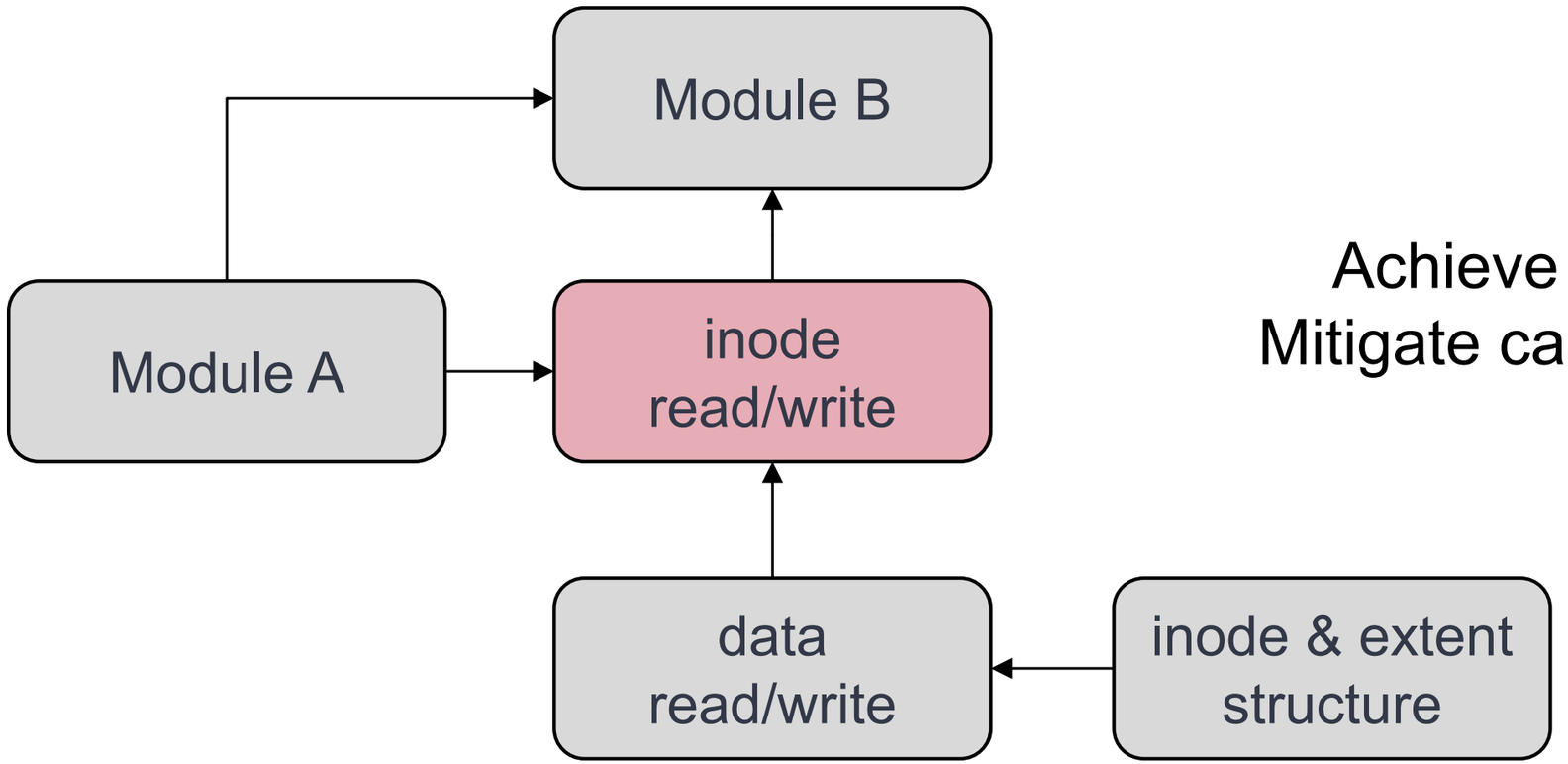


Spec Patch

Evolution Process: Spec Patch Merging



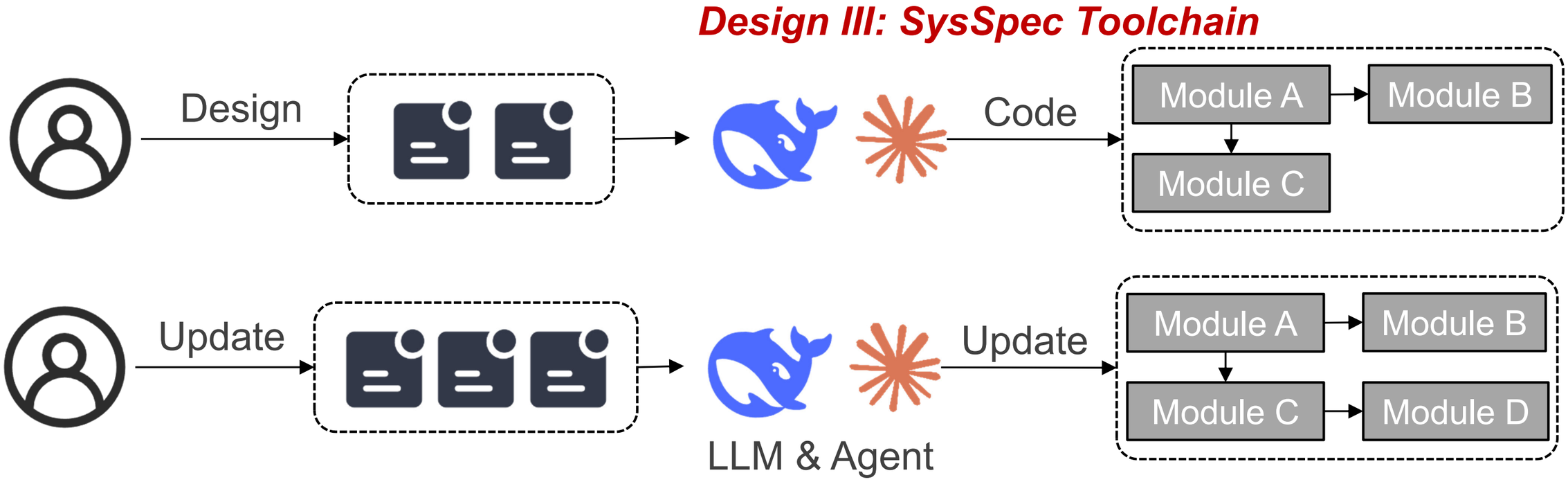
Evolution Process: Spec Patch Merging



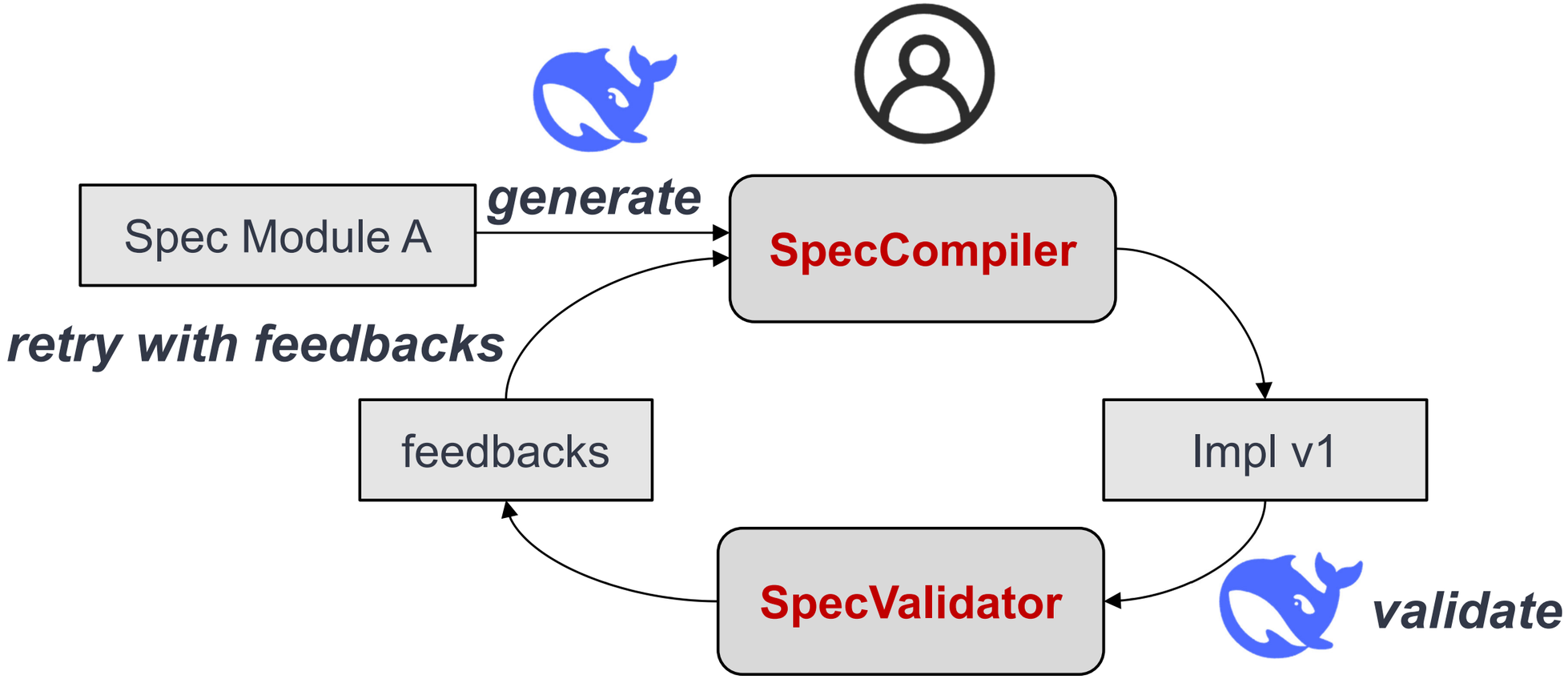
Achieve **atomic** system evolution
Mitigate cascading dependency issues

Merged Spec

SysSpec Toolchain: From Specification to Implementation



From Specification to Implementation



Retry-with-feedback loops

(~30% modules retry at least once)

Case Study: The Generation and Evolution of SpecFS

- **SpecFS: a concurrent file system generated from scratch**
  - Modeled after **AtomFS [SOSP'19]**
- **System Evolution: Add 10 ext4 features to original SpecFS**
  - Together: ~4k LoC, ranked 42/82 in Linux 6.1.10

Type	Feature	Propose	Launch	Brief Description	Release
I	Indirect Block (Ext2/3)	/	/	One-to-one block mapping via multi-level pointers	/
	Extent	2006	2006	Contiguous block ranges reducing metadata by 50%	2.6.19
	Inline Data	2011	2013	Store small files in inode's unused space	3.8
II	Multi Block Pre-Allocation	2006	2008	Benefit large files by allocating blocks in groups	2.6.25
	Delayed Allocation	2006	2008	Deferred block allocation for reducing I/O operations	2.6.27
	rbtree for Pre-Allocation	2022	2023	rbtree to organize the pre-allocated block pool	6.4
III	Metadata Checksums	2011	2012	Checksummed filesystem metadata structures	3.5
	Encryption	2015	2015	Per-directory encryption with low overhead	4.1
	Logging(jbd2)	2006	2006	Journaling support for 64-bit filesystems	2.6.19
IV	Timestamps	2006	2006	Nanosecond resolution timestamps in inode structure	2.6.19

Evaluation: Functional Correctness

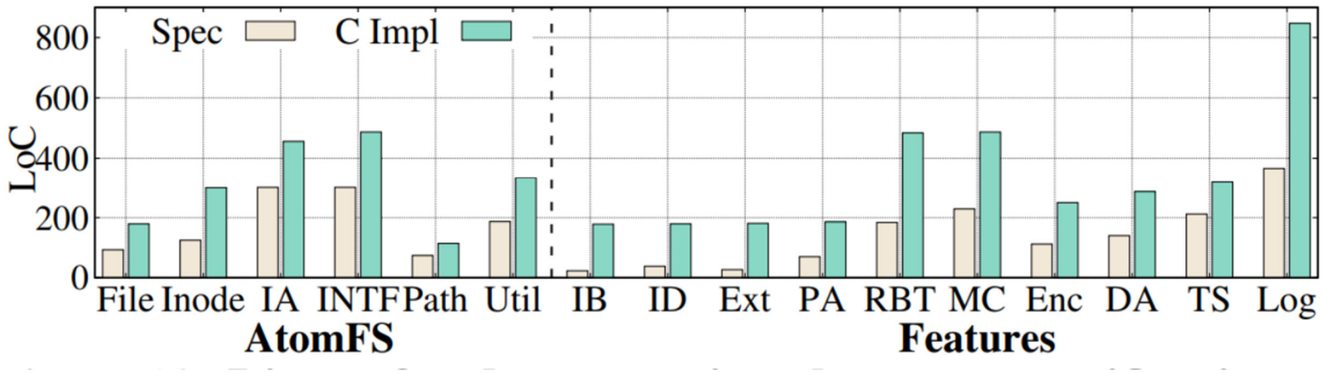
- **Definition of *Correct*:**
  - Logically equivalent to AtomFS through manual inspection and passes all tests

LLM	AtomFS	Features
Gemini-2.5/3.0-pro	*100% (45/45)*	*100% (70/70)*
Deepseek-V3.1	*100% (45/45)*	*100% (70/70)*
GPT-5-minimal	*100% (45/45)*	*100% (70/70)*
Qwen3.5-397B-A17B	*100% (45/45)*	*100% (70/70)*
Qwen3-32B	*82.2% (37/45)*	*92.9% (65/70)*

Errors mainly results from concurrency

Show equivalent level of correctness as human-written AtomFS

Evaluation: Improved Productivity



Specification:
Fewer LoC compared with C impl

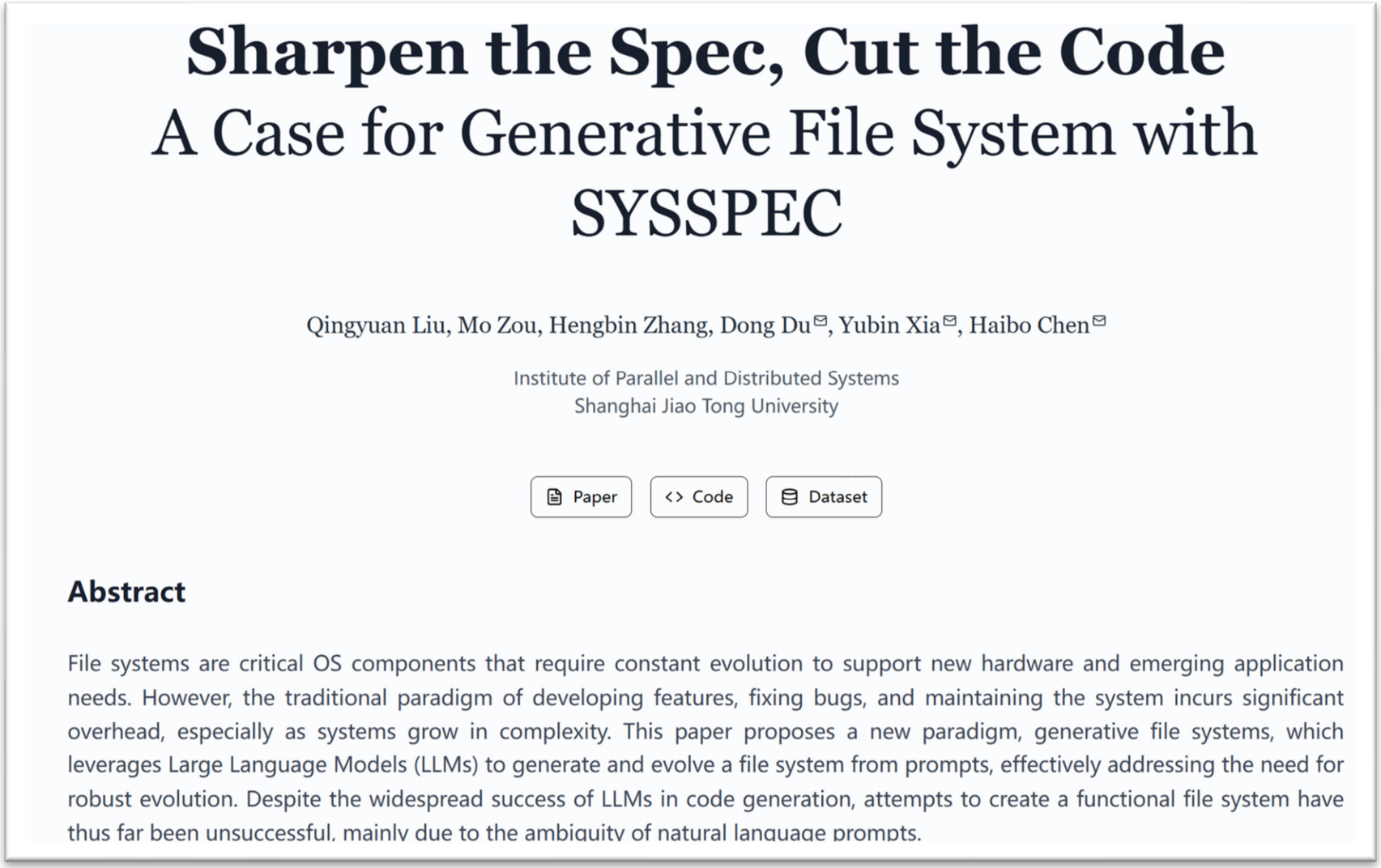
Development Costs	Extent	Rename
Manual	4.5h (3.0×)	13h (5.4×)
Ours	1.5h	2.4h

Improve productivity compared with manual implementation

Summary

- **Generative File System: *Sharpen the spec, cut the code***
  - Save the effort of file system dev/evo with LLMs
- **SysSpec and SpecFS**
  - Holistic methodology for specifying, constructing, and evolving a file system
  - **Case study:** Implement **SpecFS** based on **SysSpec** and integrate 10 ext4 features
- **Feasibility Validation**
  - Improve functional correctness of LLM-based code generation
  - Improve productivity

Main Page: *https://ipads.se.sjtu.edu.cn/projects/specfs*



Sharpen the Spec, Cut the Code
A Case for Generative File System with SYSSPEC

Qingyuan Liu, Mo Zou, Hengbin Zhang, Dong Du✉, Yubin Xia✉, Haibo Chen✉

Institute of Parallel and Distributed Systems
Shanghai Jiao Tong University

Paper Code Dataset

Abstract

File systems are critical OS components that require constant evolution to support new hardware and emerging application needs. However, the traditional paradigm of developing features, fixing bugs, and maintaining the system incurs significant overhead, especially as systems grow in complexity. This paper proposes a new paradigm, generative file systems, which leverages Large Language Models (LLMs) to generate and evolve a file system from prompts, effectively addressing the need for robust evolution. Despite the widespread success of LLMs in code generation, attempts to create a functional file system have thus far been unsuccessful, mainly due to the ambiguity of natural language prompts.

Thanks!
Q&A

Main Page: *https://ipads.se.sjtu.edu.cn/projects/specfs*