Specifying and Checking File System Crash-Consistency Models

James Bornholt
Antoine Kaufmann
Jialin Li
Arvind Krishnamurthy
Emina Torlak
Xi Wang

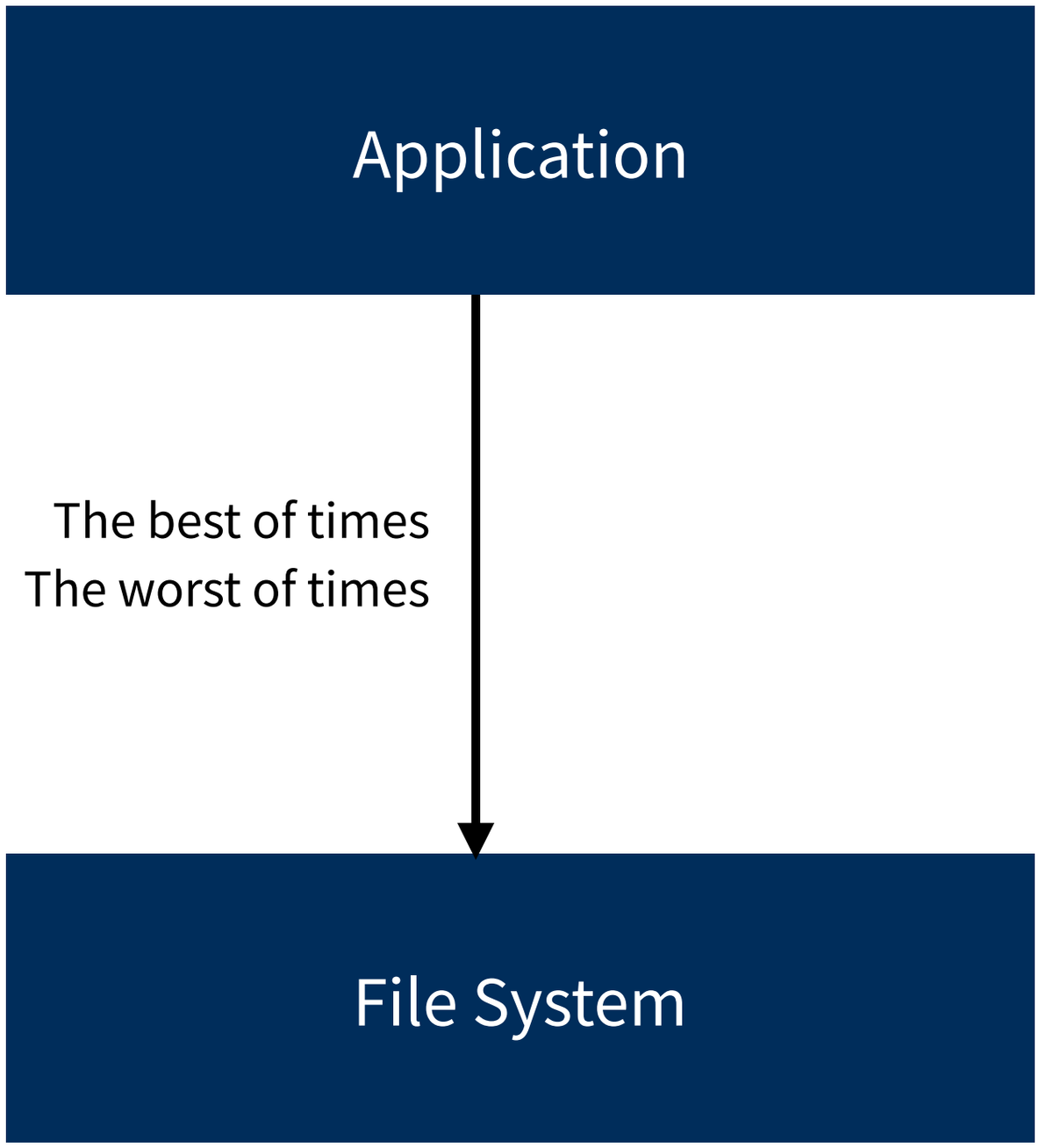
University of Washington

File systems persist our data

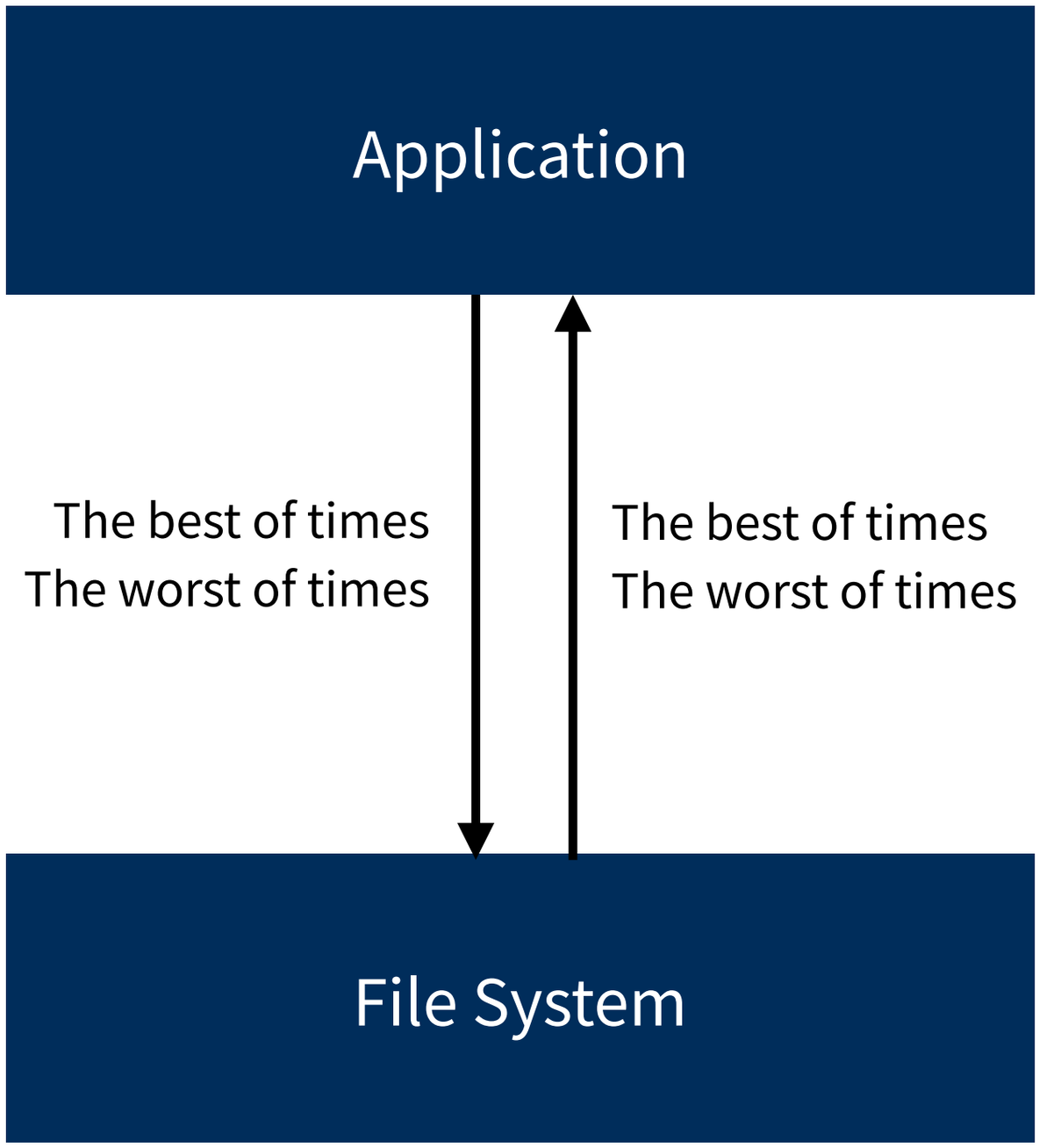
Application

File System

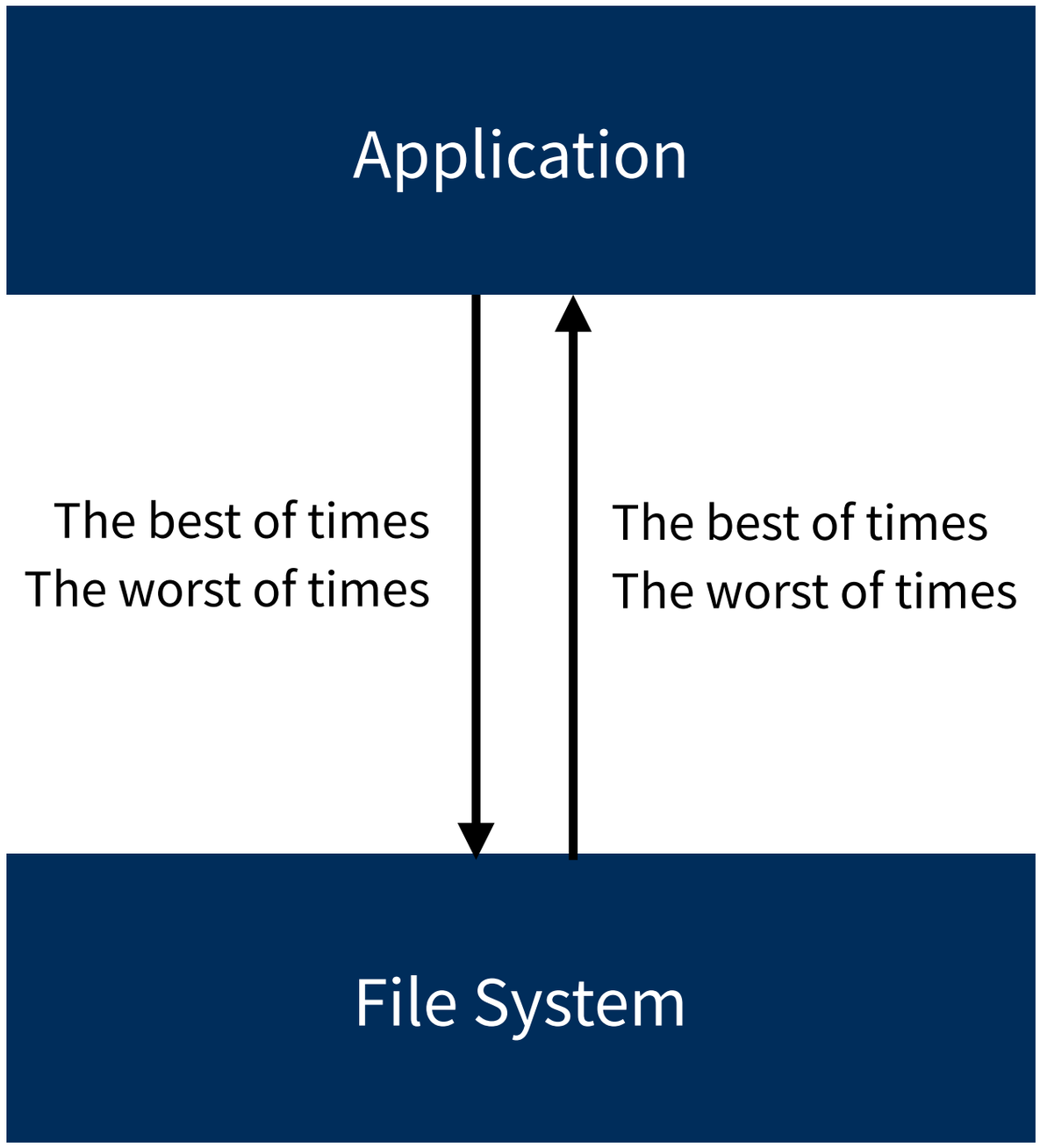
File systems persist our data



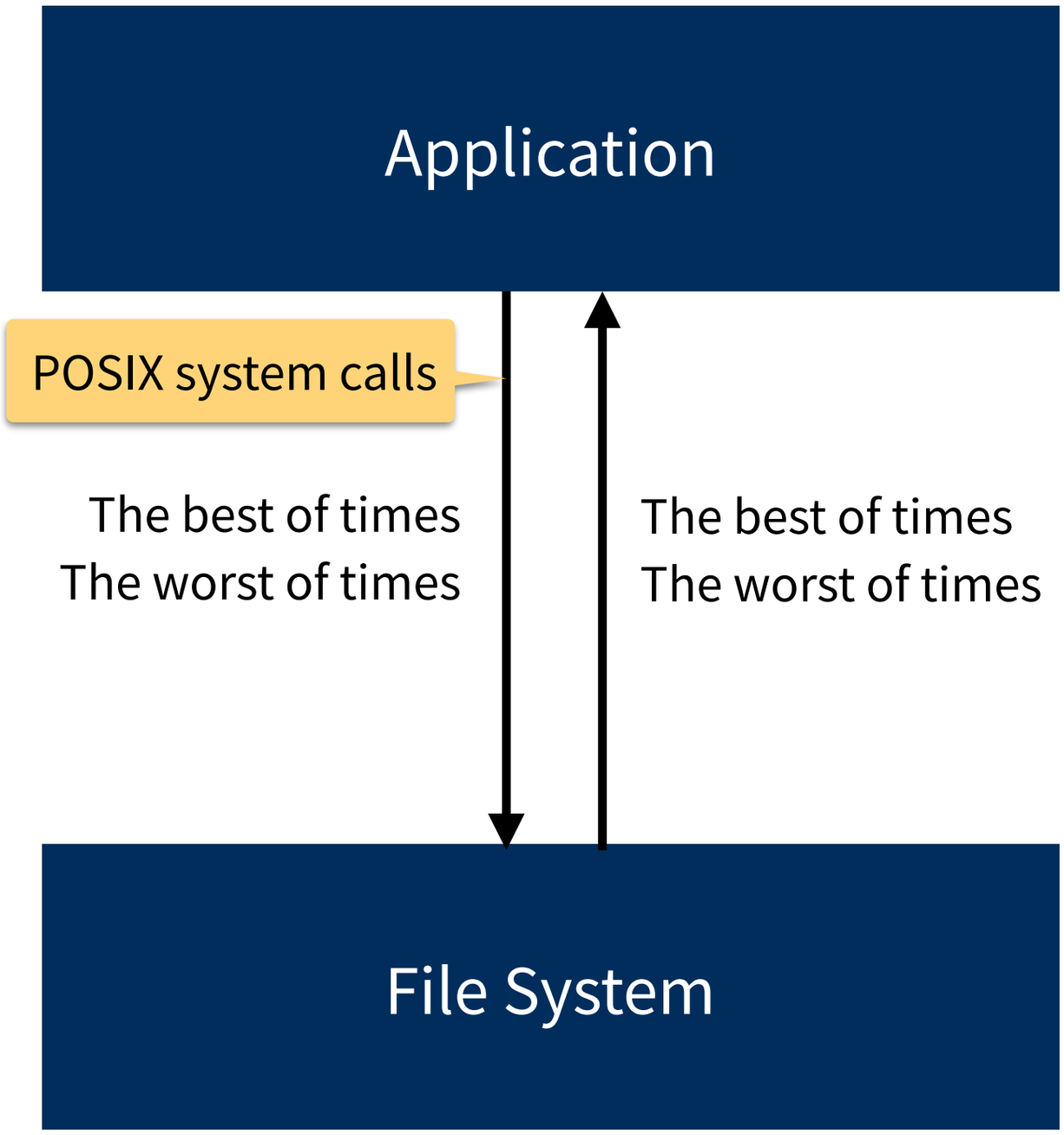
File systems persist our data



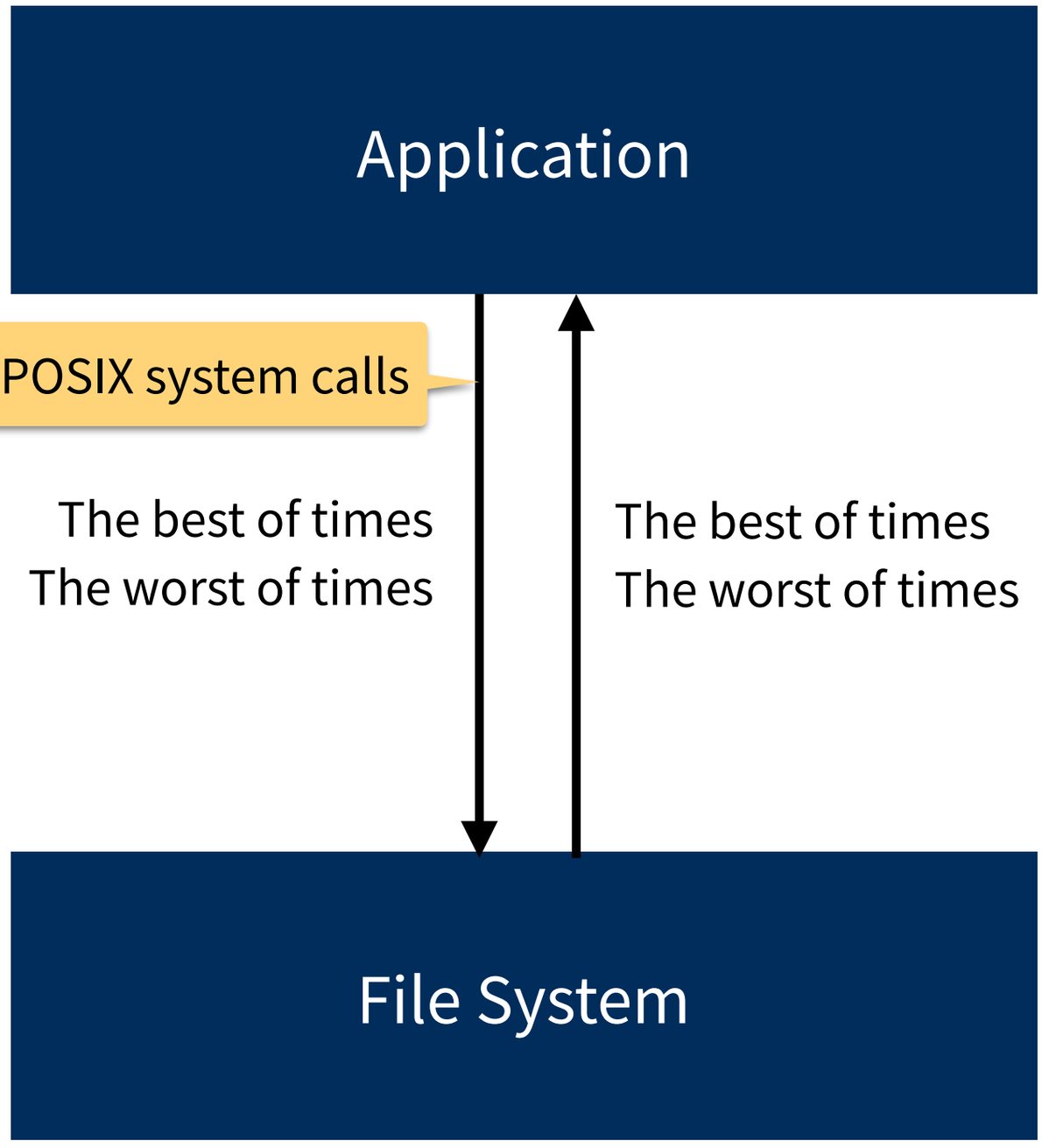
But what if the system crashes?



But what if the system crashes?



But what if the system crashes?



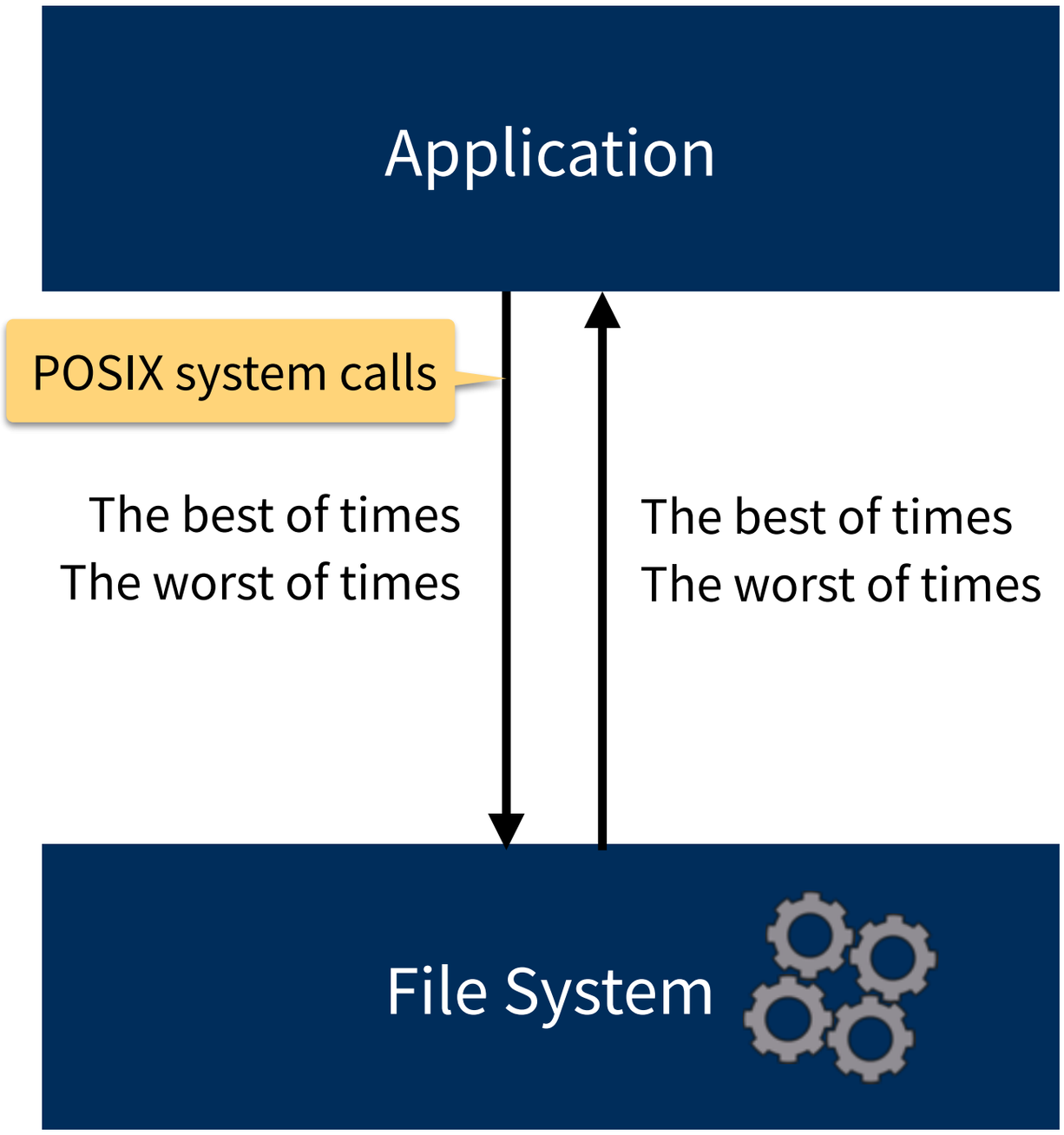
This provides **roughly the same level of guarantees** as ext3.

Linux kernel ext4 documentation

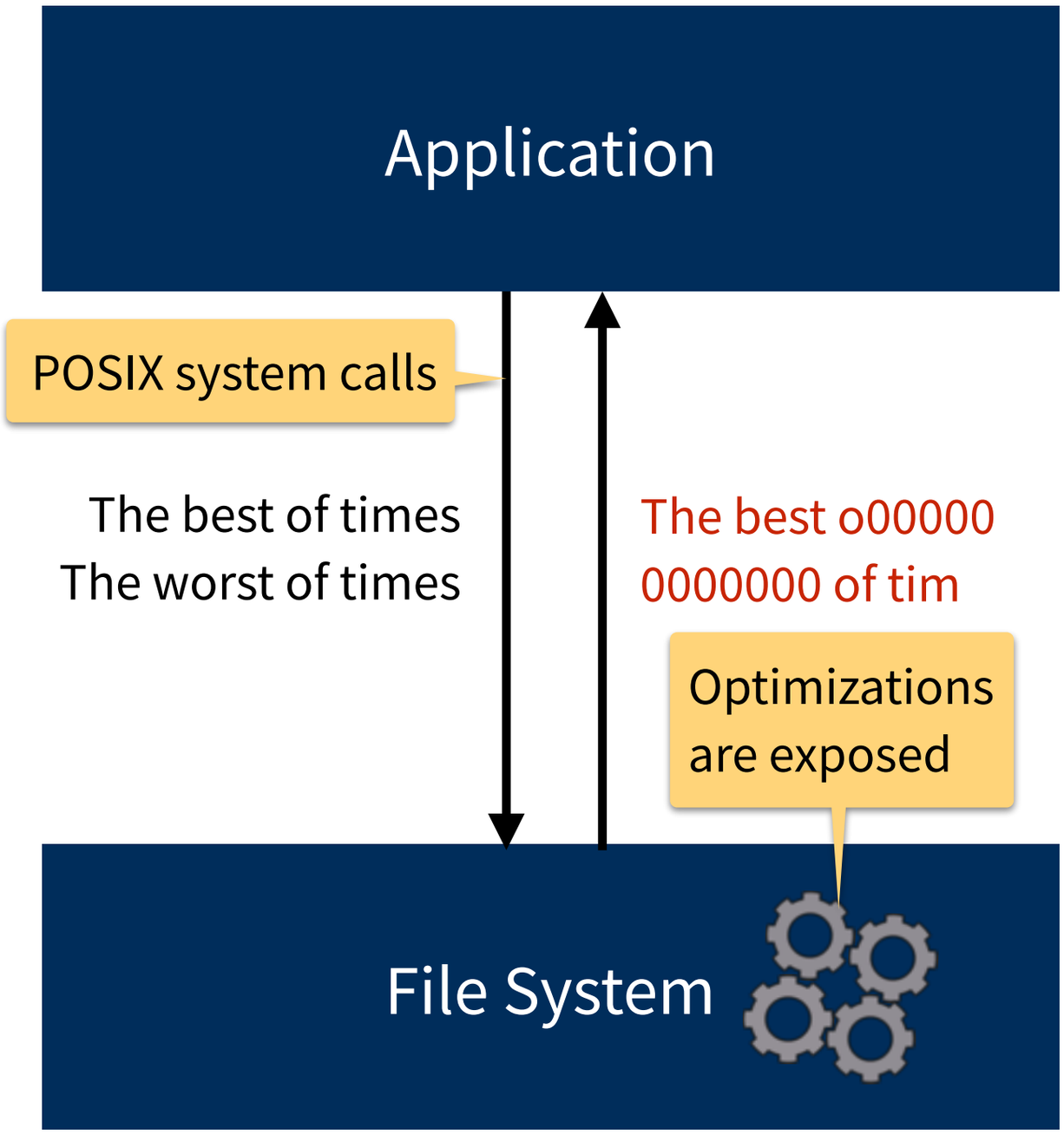
If the file system is inconsistent after a crash it is **usually** automatically checked and repaired when the system is rebooted

Proposed POSIX *fsync* documentation

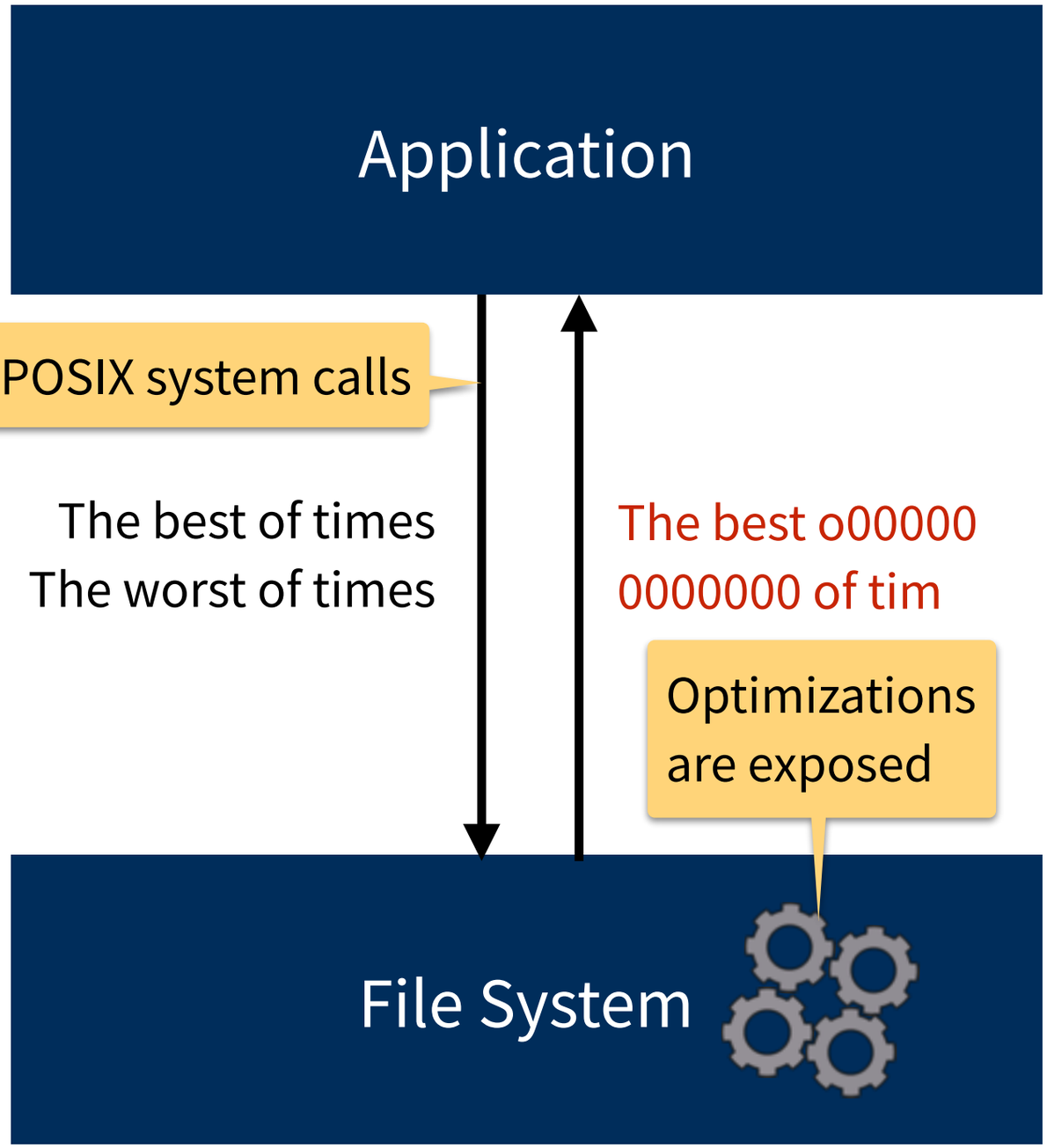
But what if the system crashes?



But what if the system crashes?



But what if the system crashes?



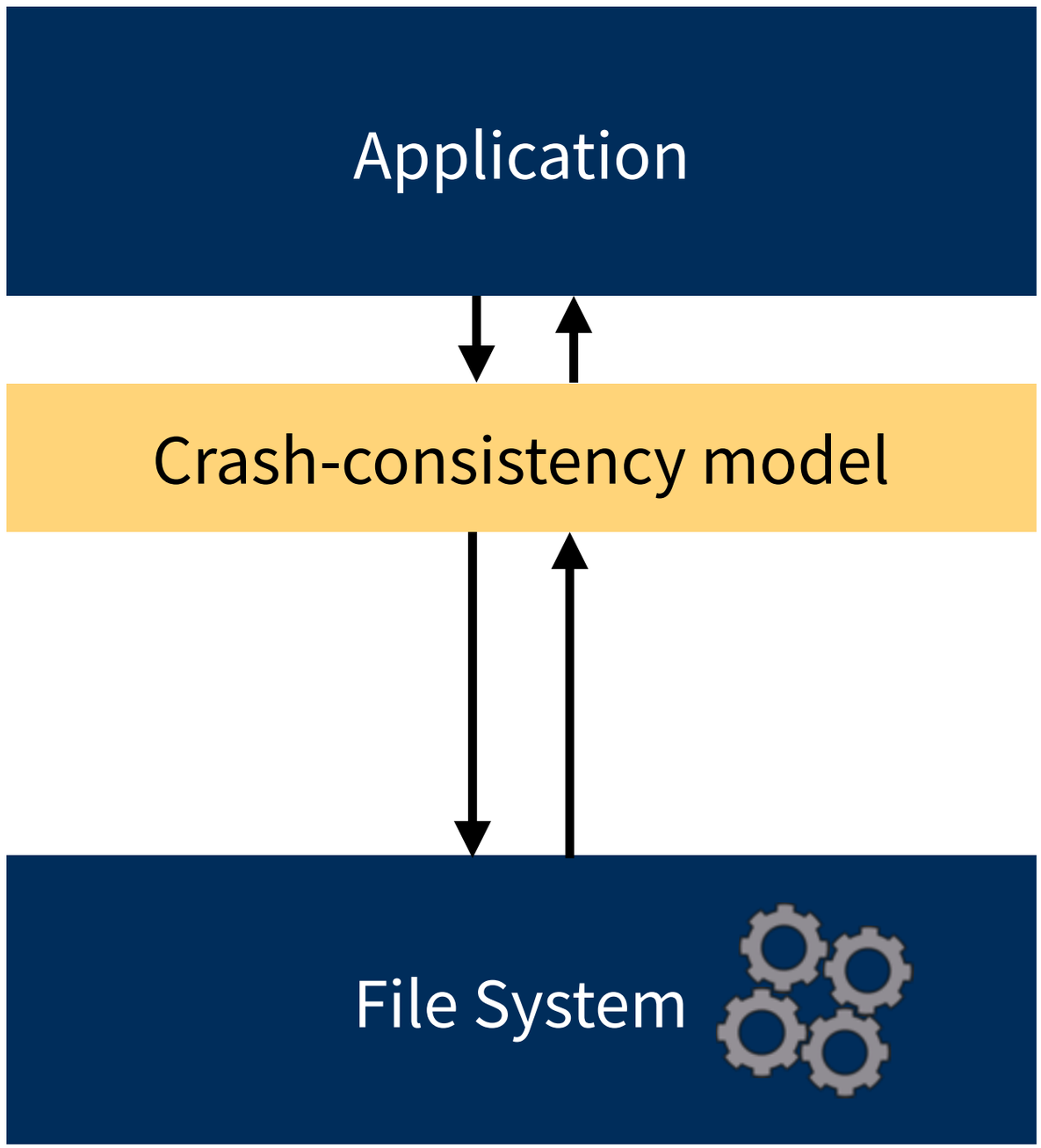
When gradually appending to a file, the content gets corrupted, causing **Chrome** to crash

ChromeOS “FS corruption on panic”, 2015

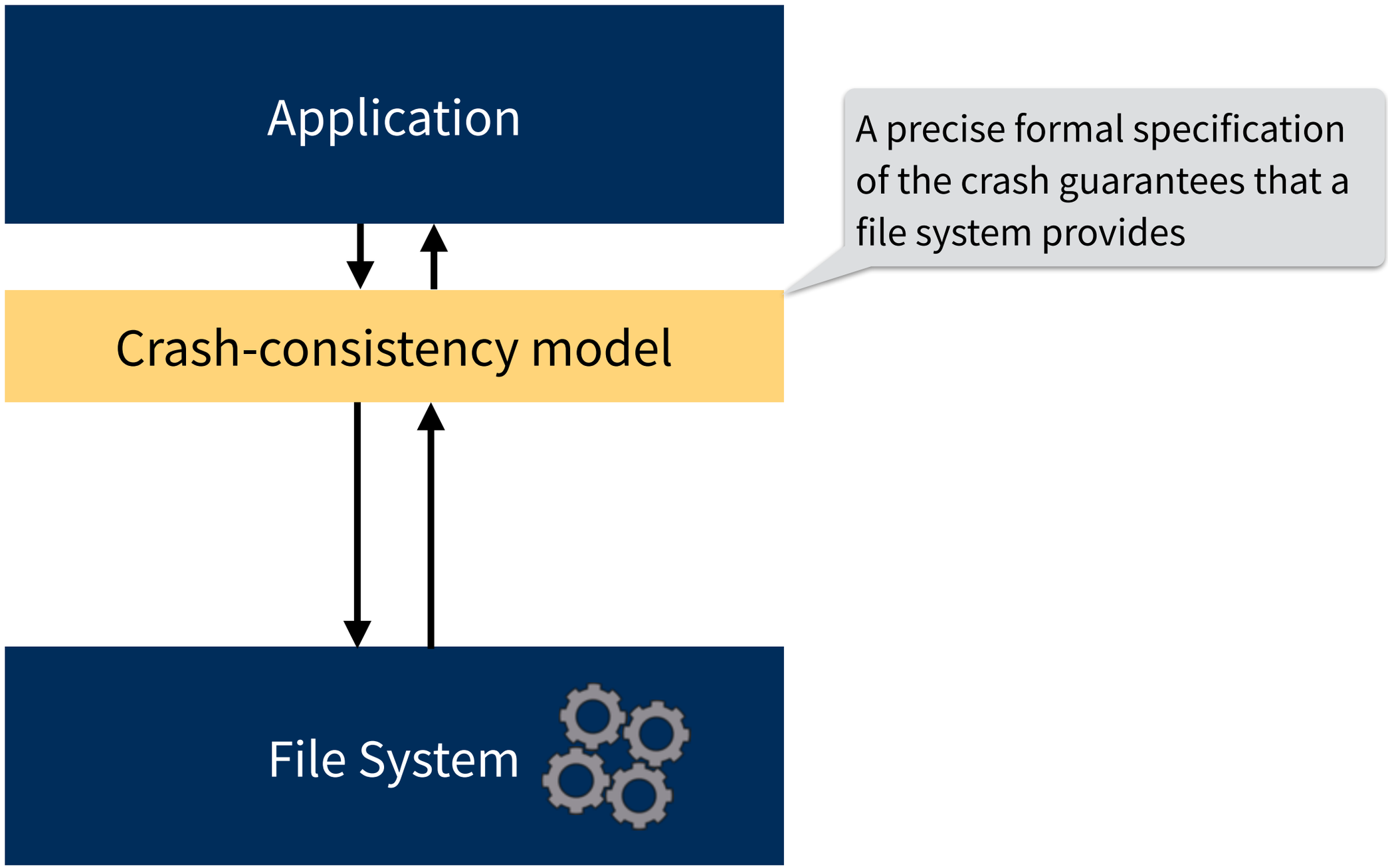
…some of the **KDE** core config files were reset. Also some of my **MySQL** databases were killed…

Ubuntu “ext4 data loss”, 2009

Crash-consistency models

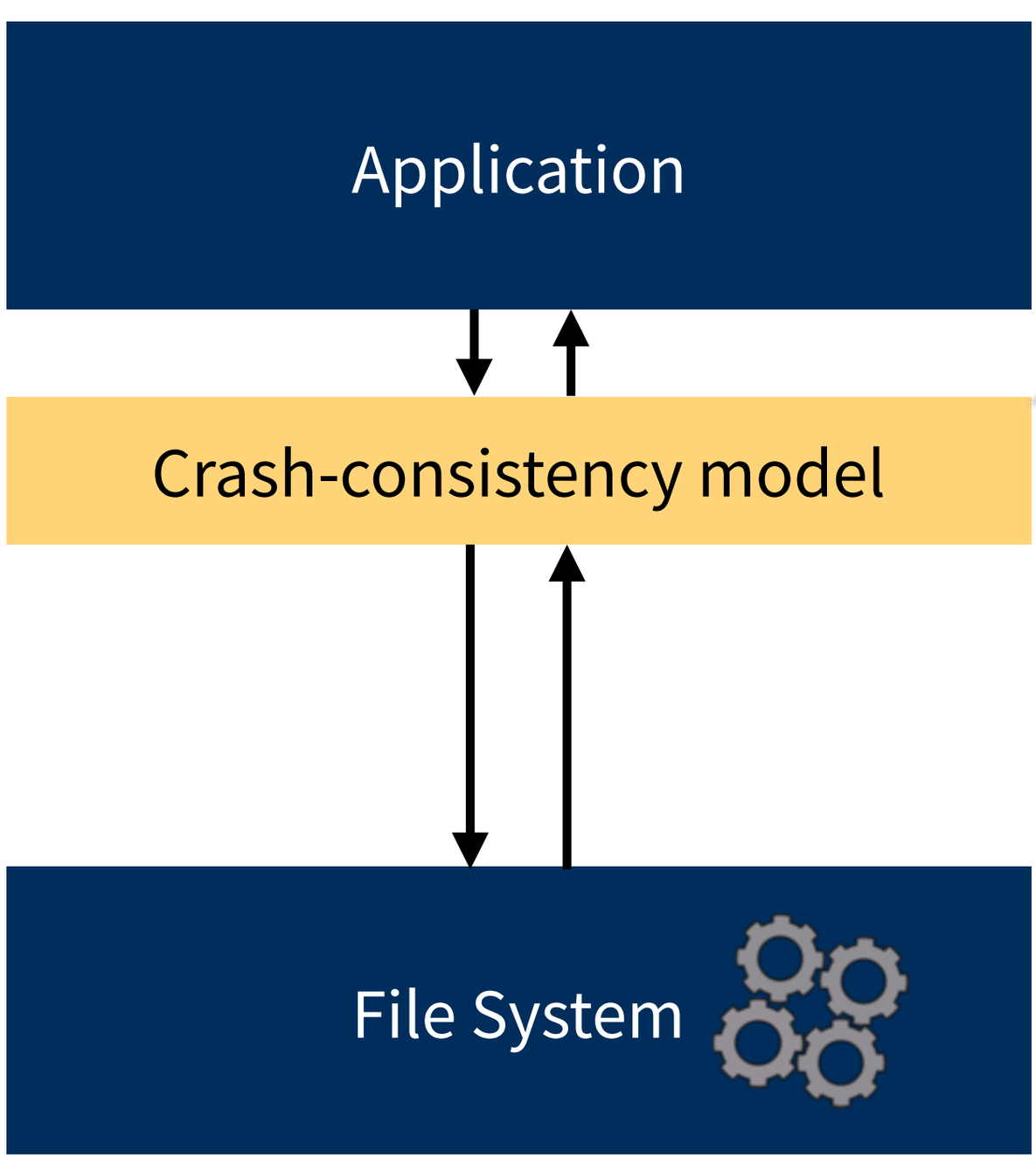


Crash-consistency models



A precise formal specification of the crash guarantees that a file system provides

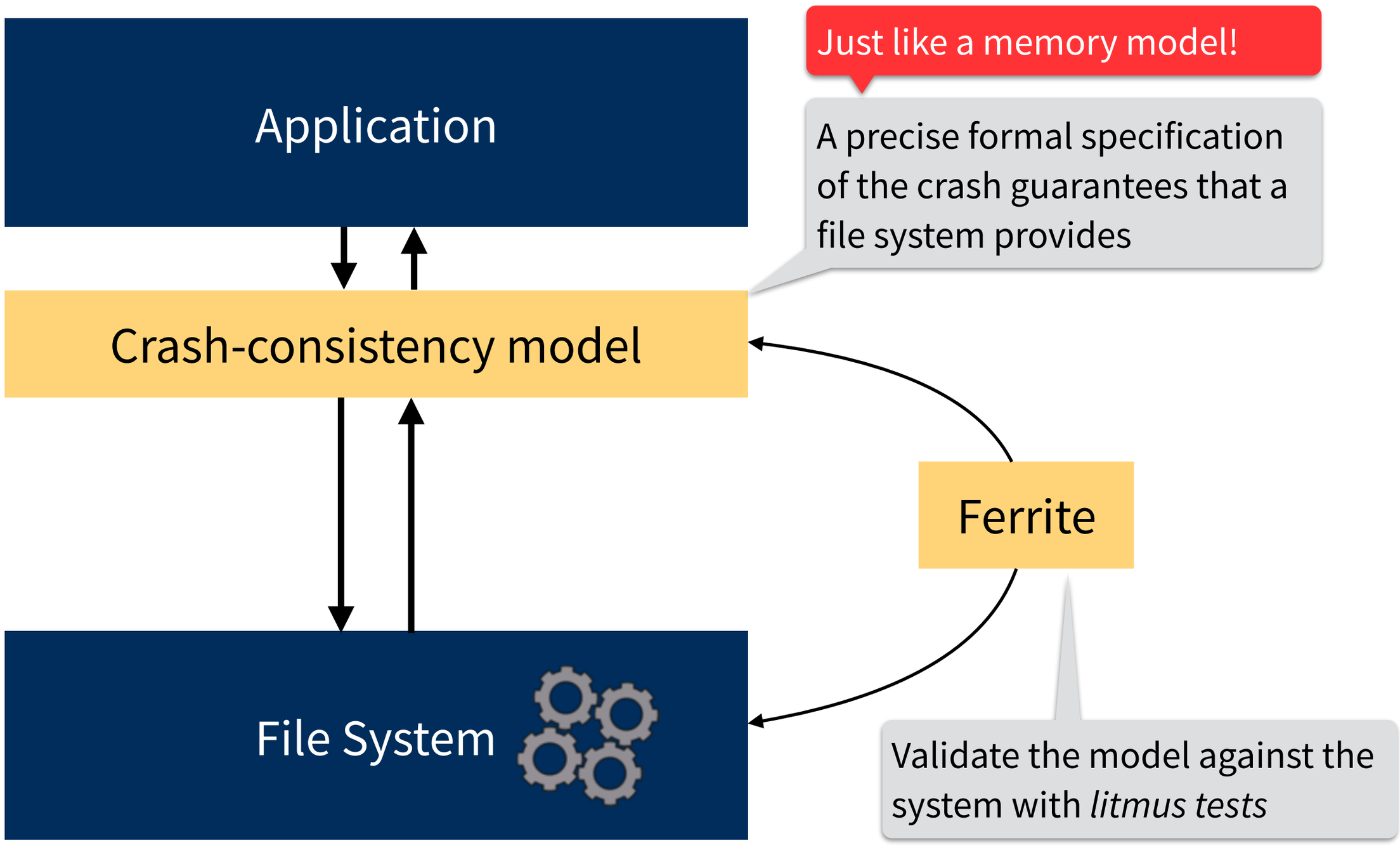
Crash-consistency models



Just like a memory model!

A precise formal specification of the crash guarantees that a file system provides

Crash-consistency models



Crash behavior of modern file systems

Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing crash-consistency models

Building crash-safe applications

Crash behavior of modern file systems

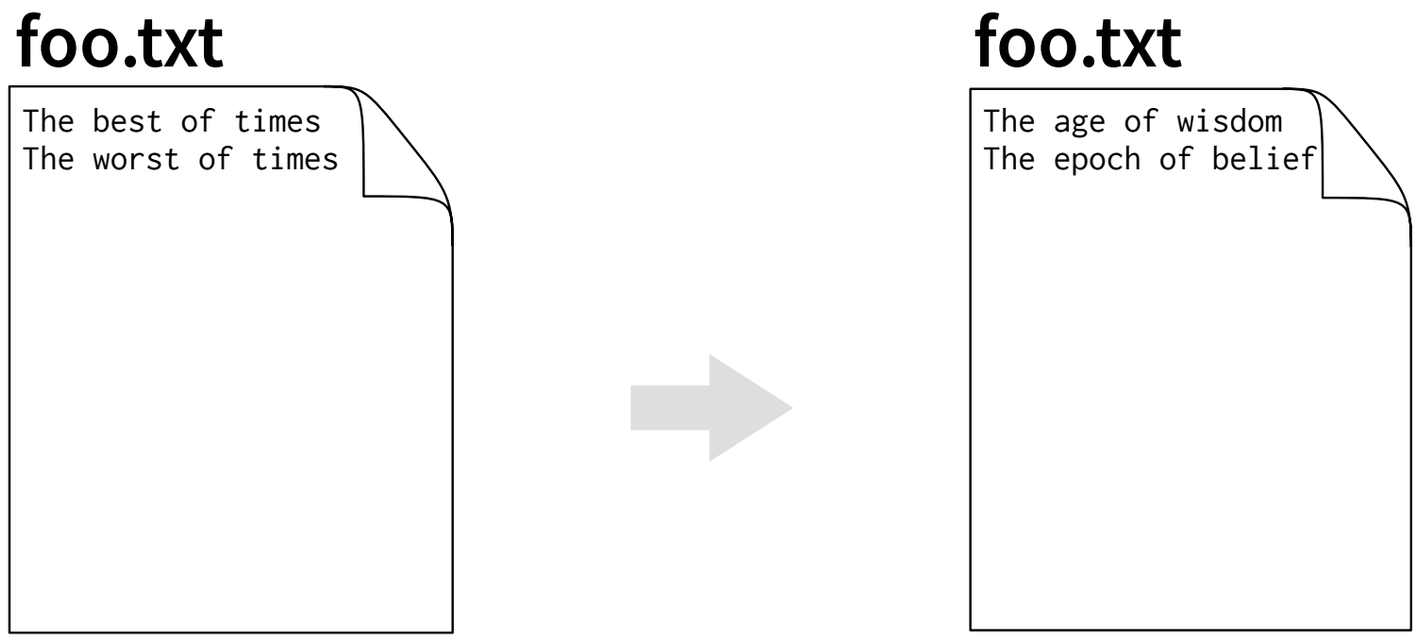
Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing crash-consistency models

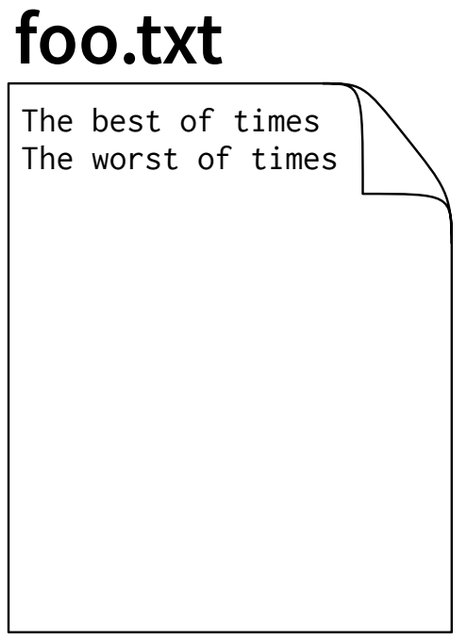
Building crash-safe applications

Replacing the contents of a file



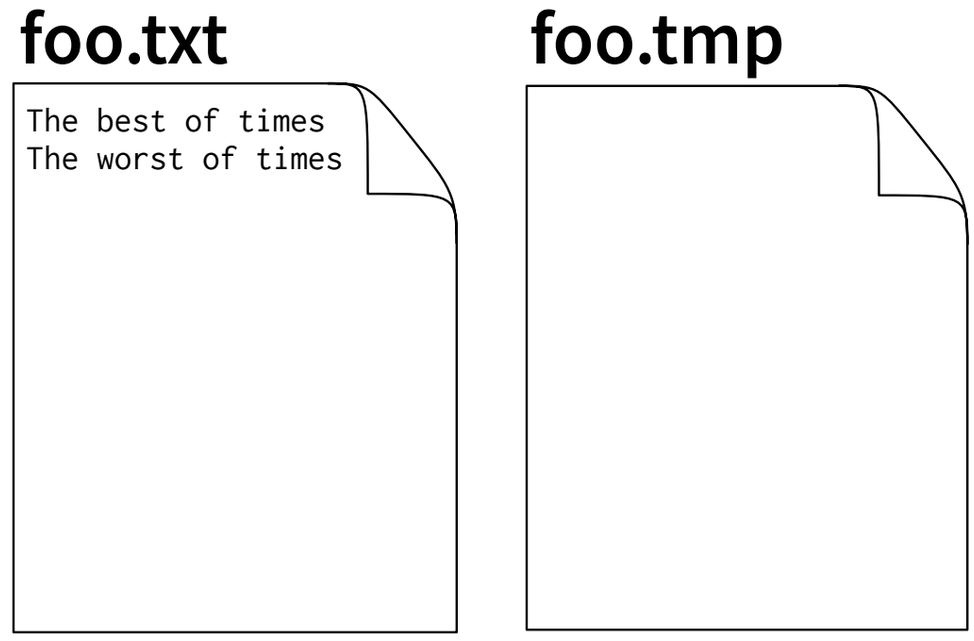
Atomic replace via rename

```
f = create(“foo.tmp”)
write(f, “The age of …”)
write(f, “The epoch of …”)
close(f)
rename(“foo.tmp”, “foo.txt”)
```



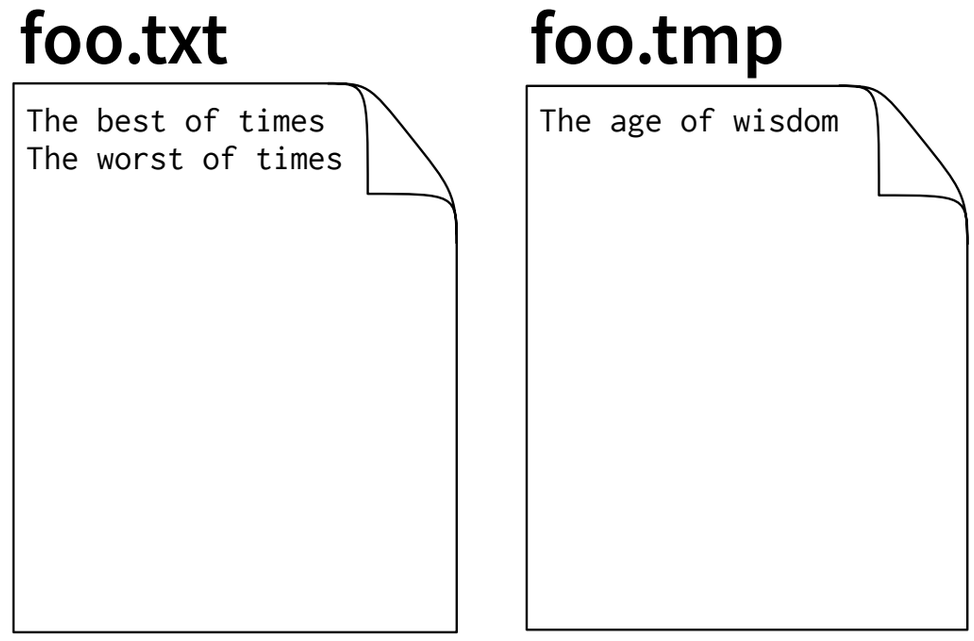
Atomic replace via rename

```
f = create(“foo.tmp”)
write(f, “The age of …”)
write(f, “The epoch of …”)
close(f)
rename(“foo.tmp”, “foo.txt”)
```



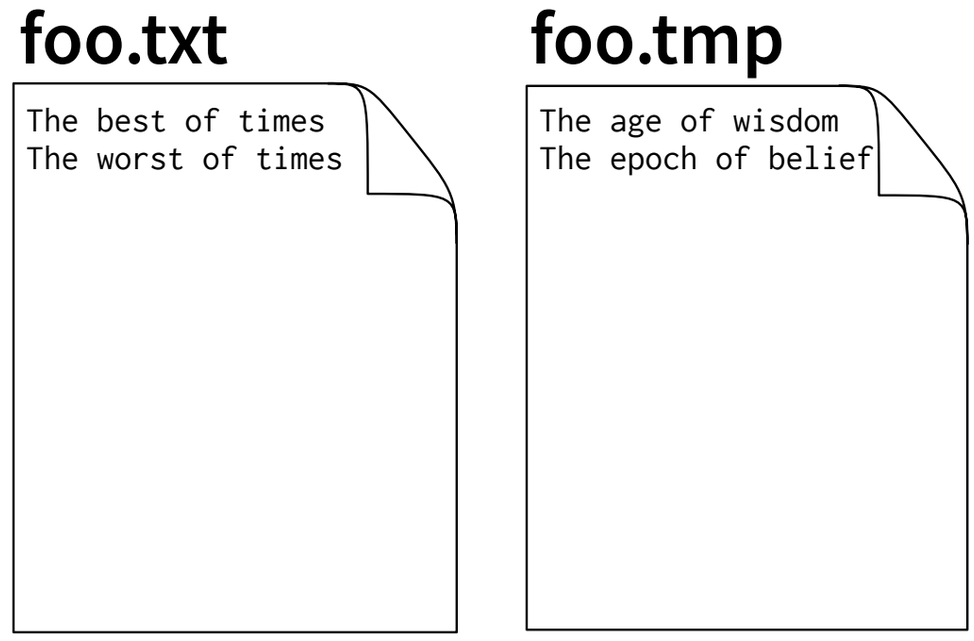
Atomic replace via rename

```
f = create(“foo.tmp”)
write(f, “The age of …”)
write(f, “The epoch of …”)
close(f)
rename(“foo.tmp”, “foo.txt”)
```



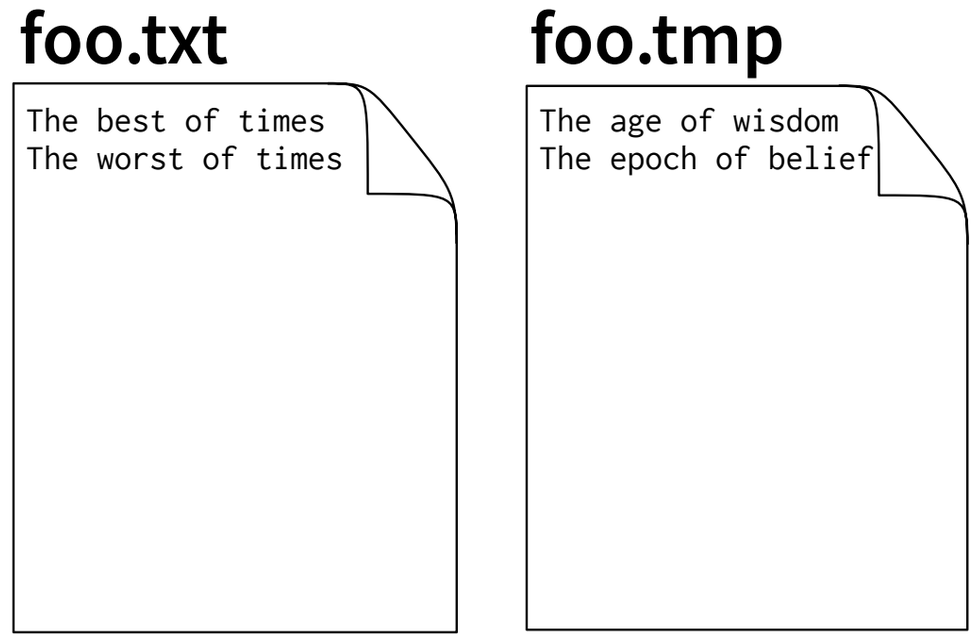
Atomic replace via rename

```
f = create(“foo.tmp”)
write(f, “The age of …”)
write(f, “The epoch of …”)
close(f)
rename(“foo.tmp”, “foo.txt”)
```



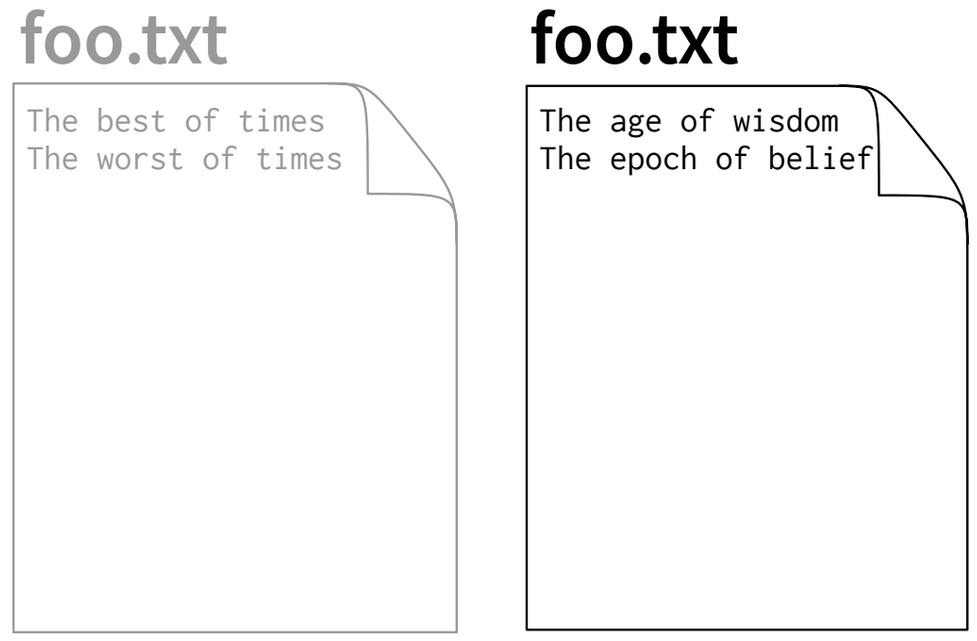
Atomic replace via rename

```
f = create(“foo.tmp”)
write(f, “The age of …”)
write(f, “The epoch of …”)
close(f)
rename(“foo.tmp”, “foo.txt”)
```



Atomic replace via rename

```
f = create(“foo.tmp”)
write(f, “The age of …”)
write(f, “The epoch of …”)
close(f)
rename(“foo.tmp”, “foo.txt”)
```



Atomic replace via rename

```
f = create(“foo.tmp”)
write(f, “The age of …”)
write(f, “The epoch of …”)
close(f)
rename(“foo.tmp”, “foo.txt”)
```

Atomic replace via rename

```
f = create(“foo.tmp”)
write(f, “The age of …”)
write(f, “The epoch of …”)
close(f)
rename(“foo.tmp”, “foo.txt”)
```

create(“foo.tmp”)

write(f, “The age of …”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

Atomic replace via rename

create(“foo.tmp”)

write(f, “The age of …”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

Atomic replace via rename

File operations

create(“foo.tmp”)

rename(“foo.tmp”, “foo.txt”)

Writes

write(f, “The age of …”)

write(f, “The epoch of …”)

Atomic replace via rename

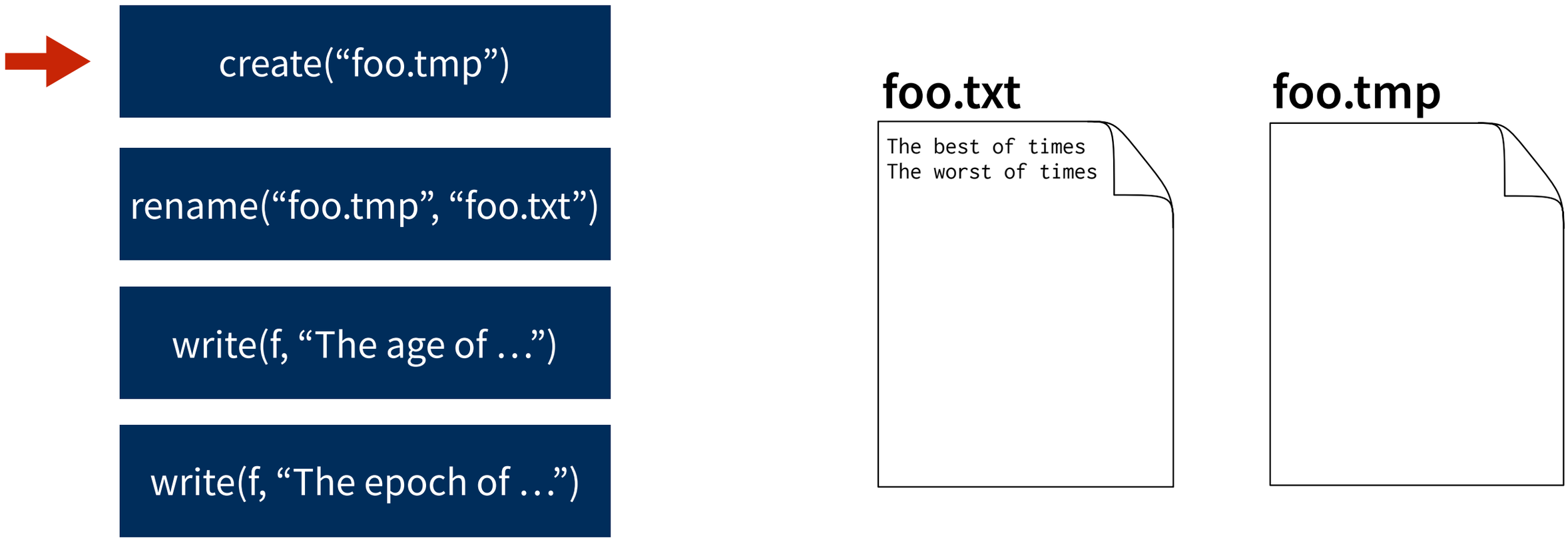
create(“foo.tmp”)

rename(“foo.tmp”, “foo.txt”)

write(f, “The age of …”)

write(f, “The epoch of …”)

Atomic replace via rename



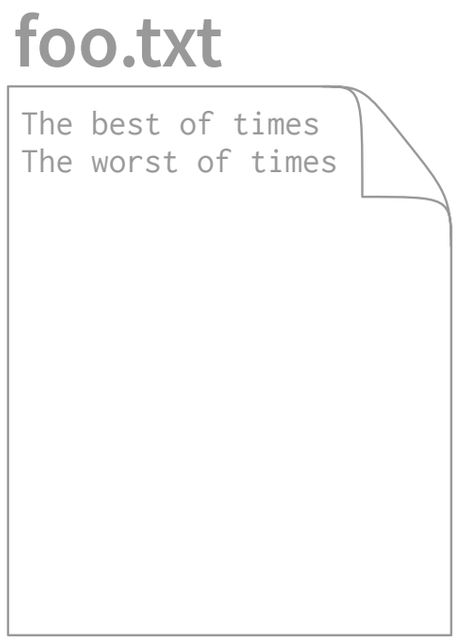
Atomic replace via rename

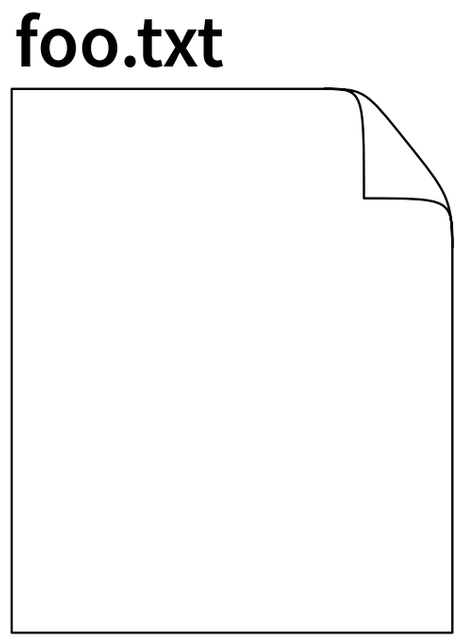
create(“foo.tmp”)

rename(“foo.tmp”, “foo.txt”)

write(f, “The age of …”)

write(f, “The epoch of …”)





Atomic replace via rename

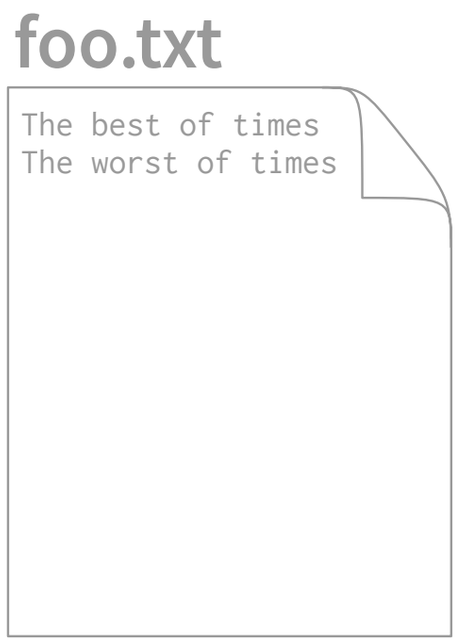
create(“foo.tmp”)

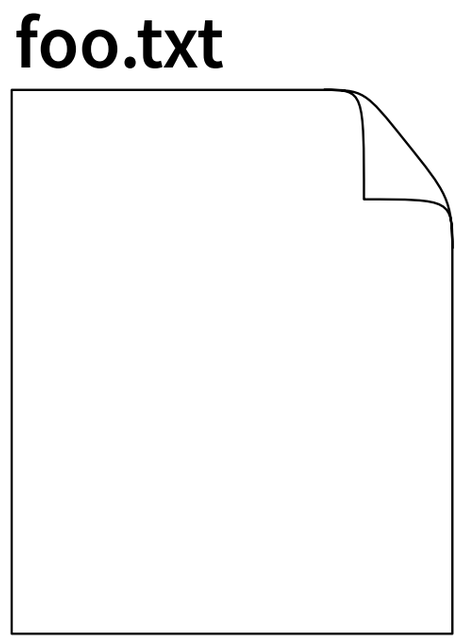
rename(“foo.tmp”, “foo.txt”)

Crash!

write(f, “The age of …”)

write(f, “The epoch of …”)



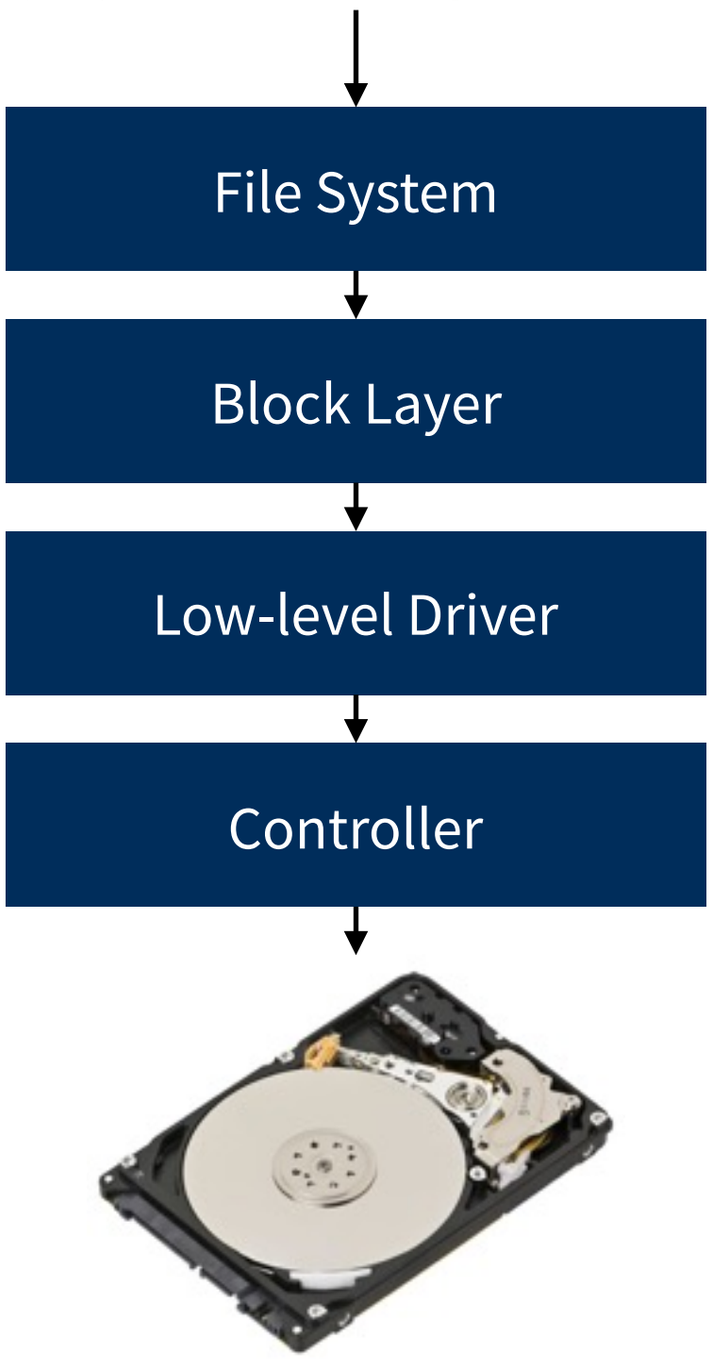


The storage stack

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```

The storage stack

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```



The storage stack

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```

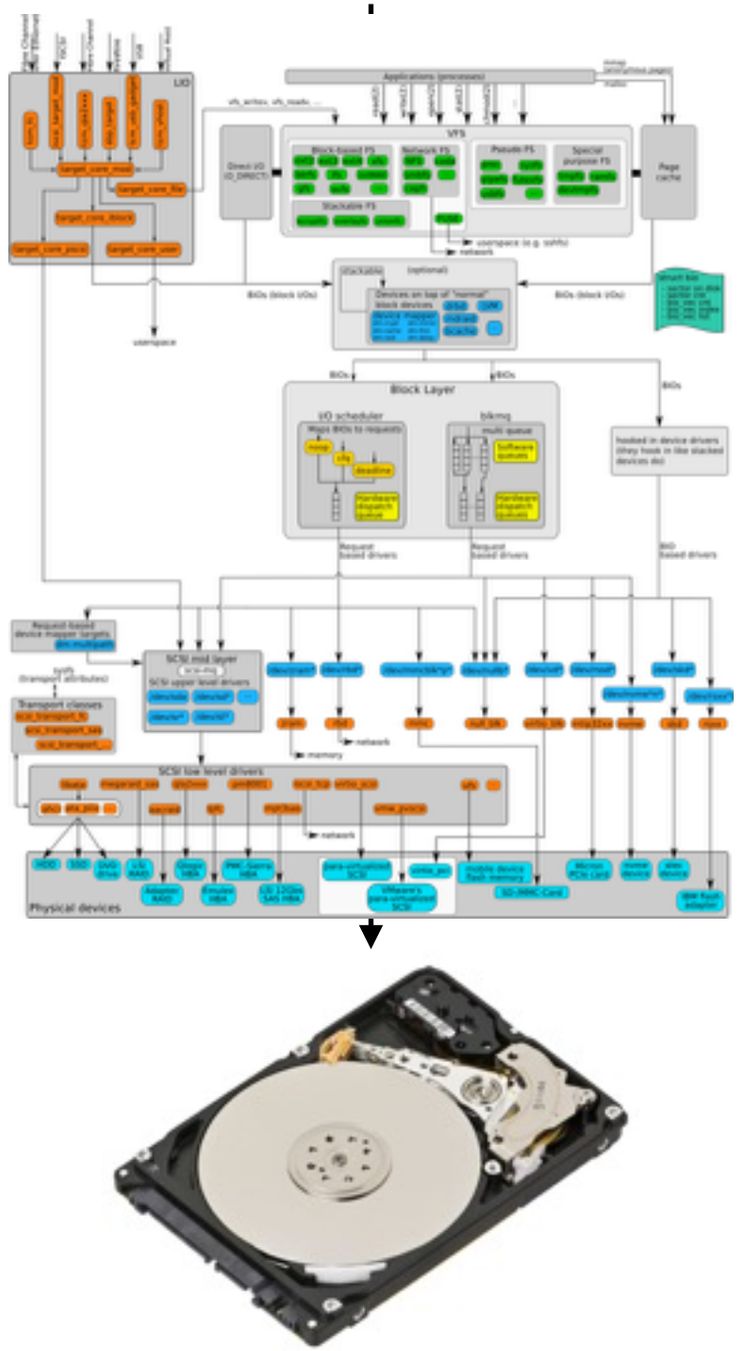
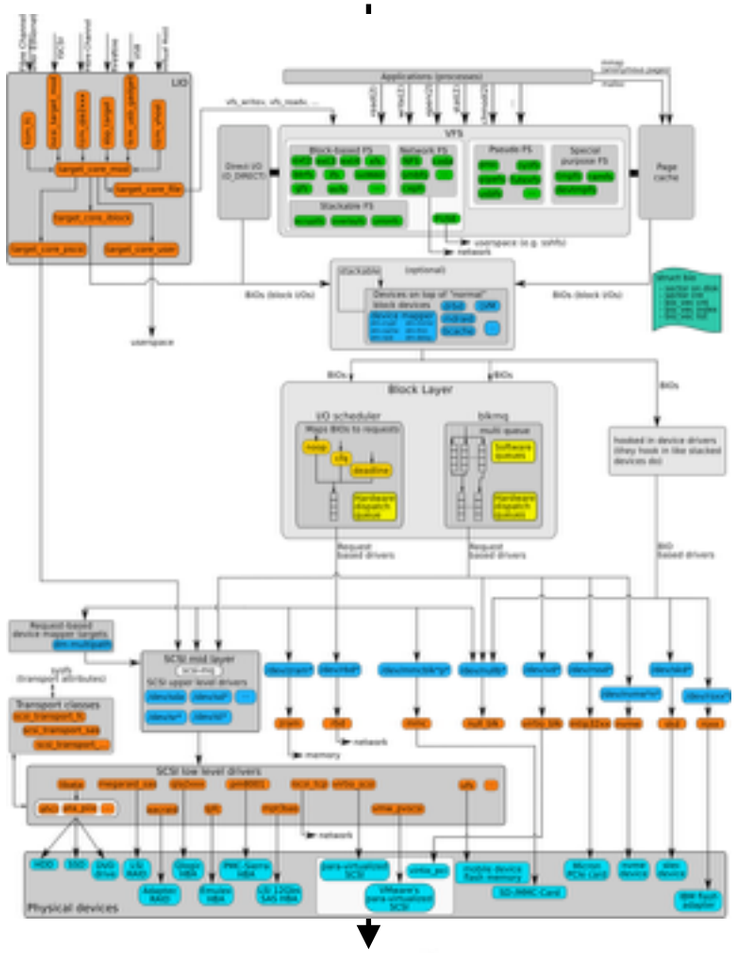


Diagram by Werner Fischer

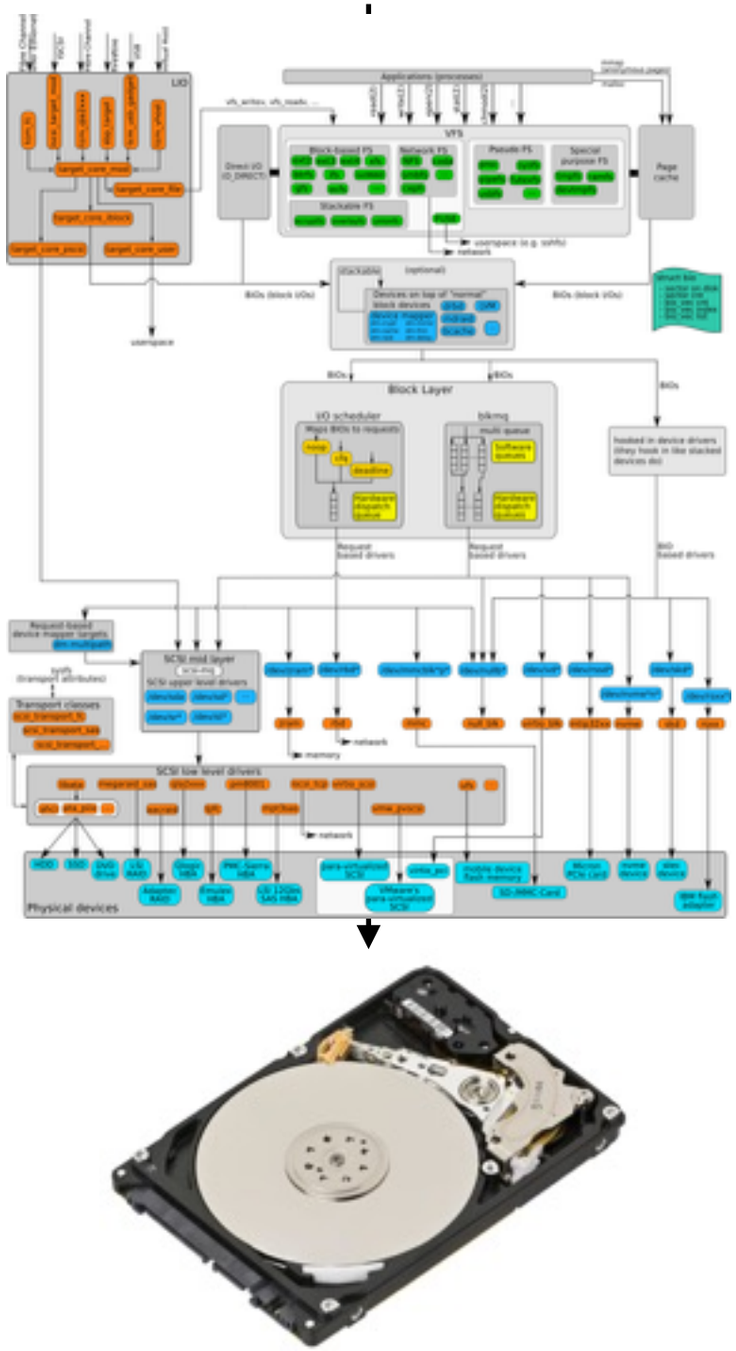
The storage stack

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```



The storage stack

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```

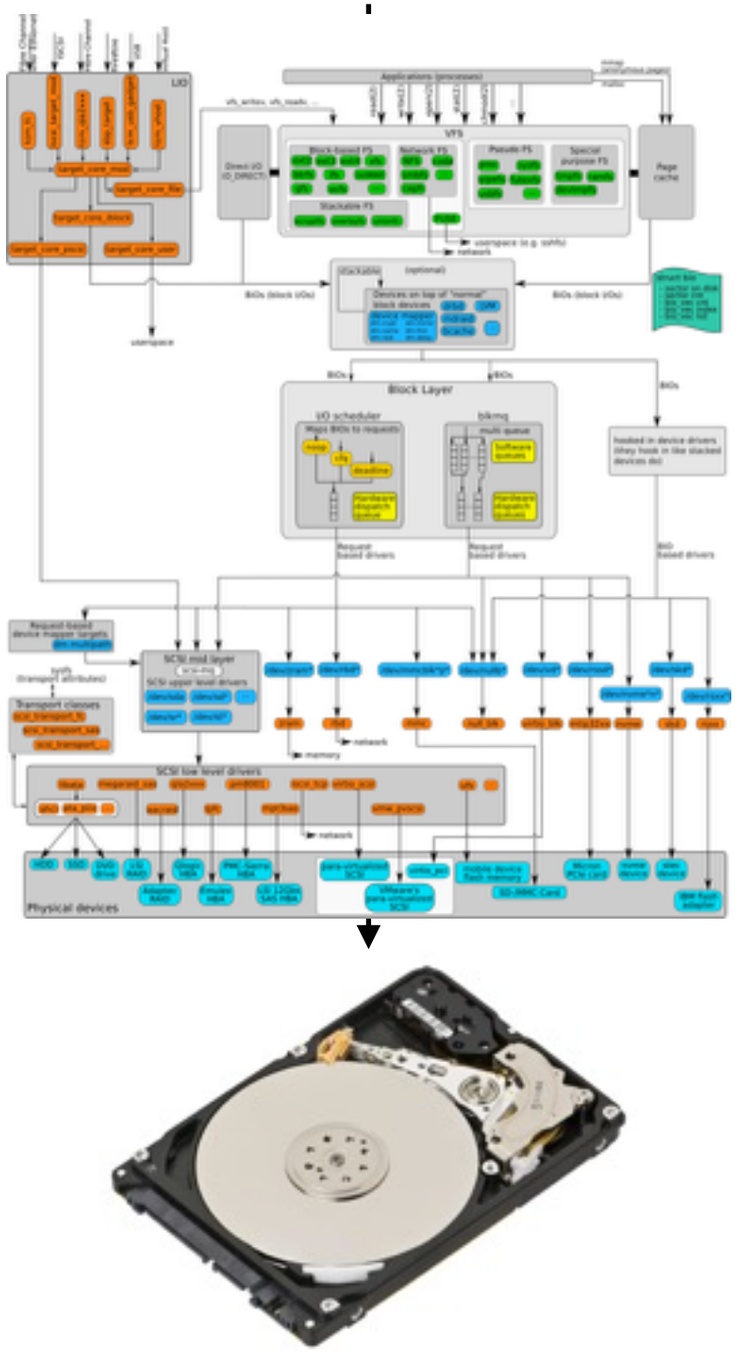


This provides **roughly the same level of guarantees** as ext3.

Linux kernel ext4 documentation

The storage stack

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```



This provides **roughly the same level of guarantees** as ext3.

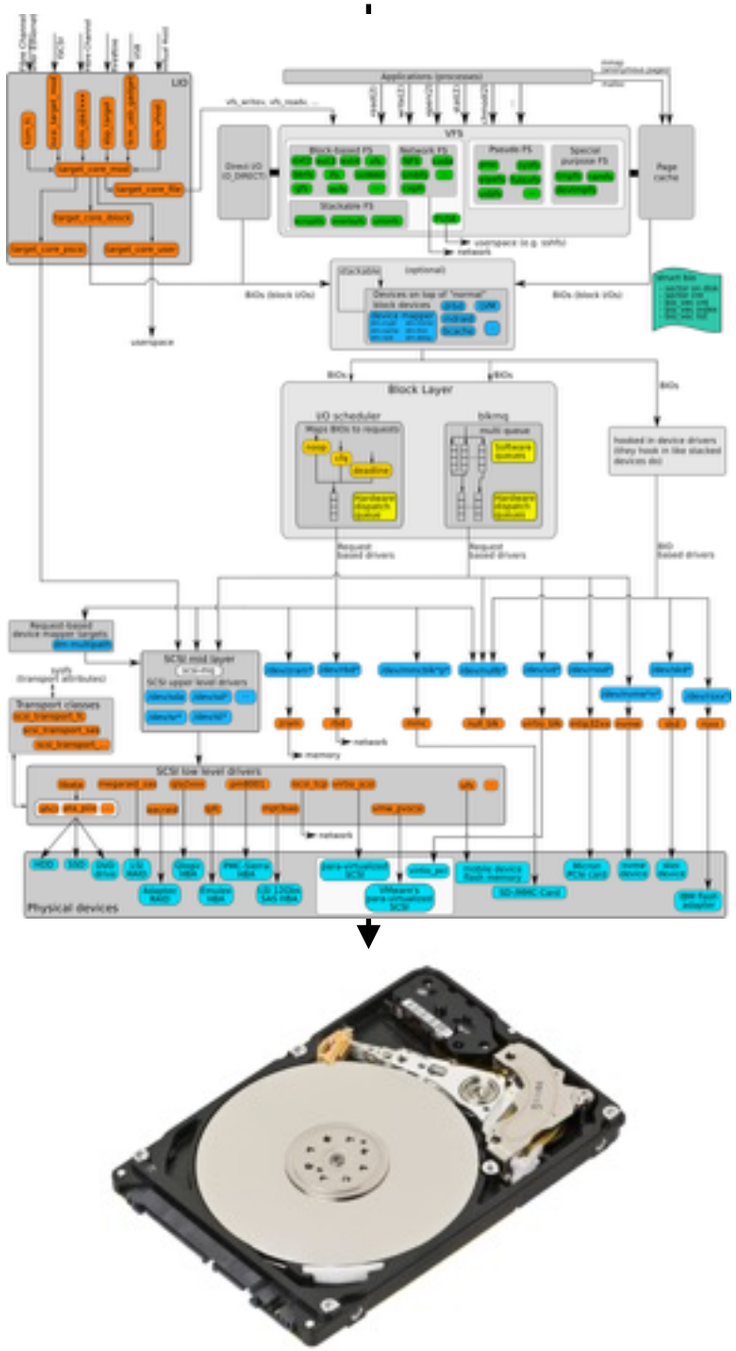
Linux kernel ext4 documentation

The key aspects of `fsync()` are **unreasonable to test** in a test suite

POSIX specification for *fsync*

Existing work

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```

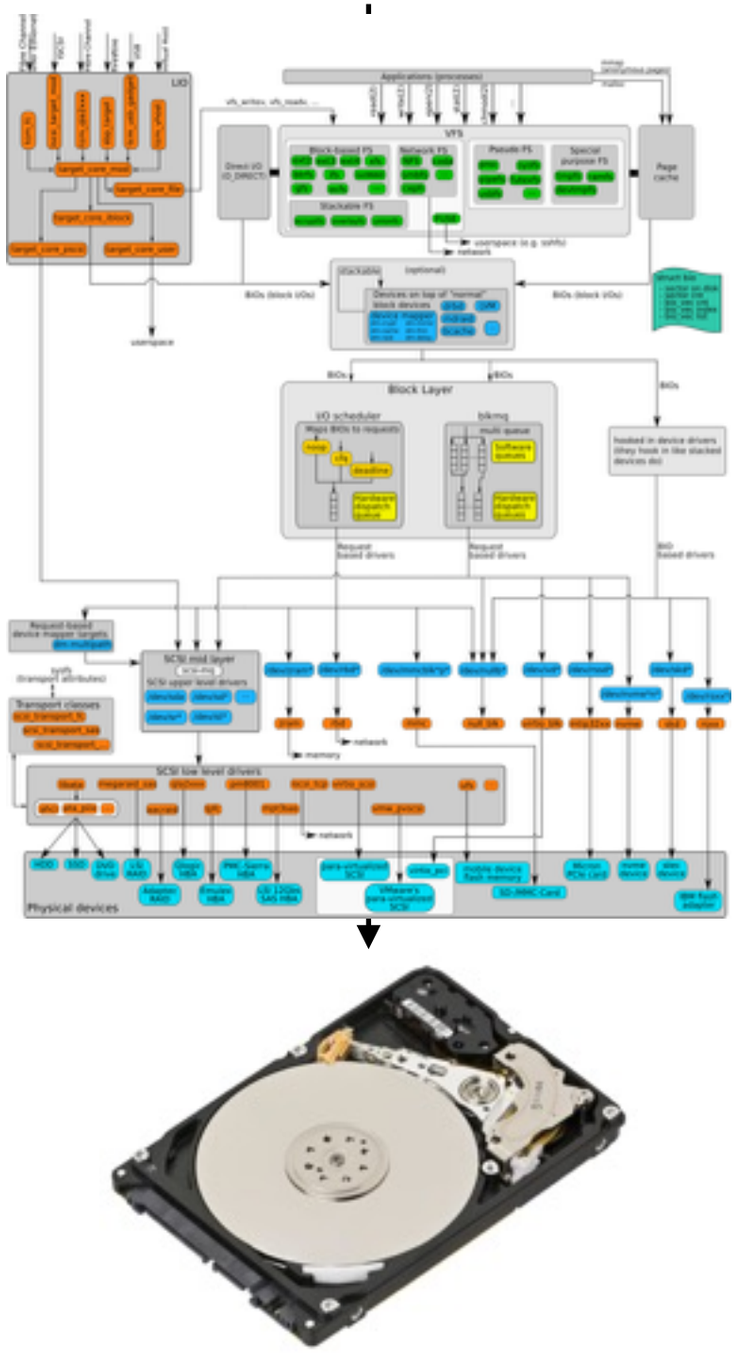


Formalize the existing POSIX interface (e.g. SibylFS [SOSP’15])

- But the interface says nothing about crash safety

Existing work

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```



Formalize the existing POSIX interface (e.g. SibylFS [SOSP’15])

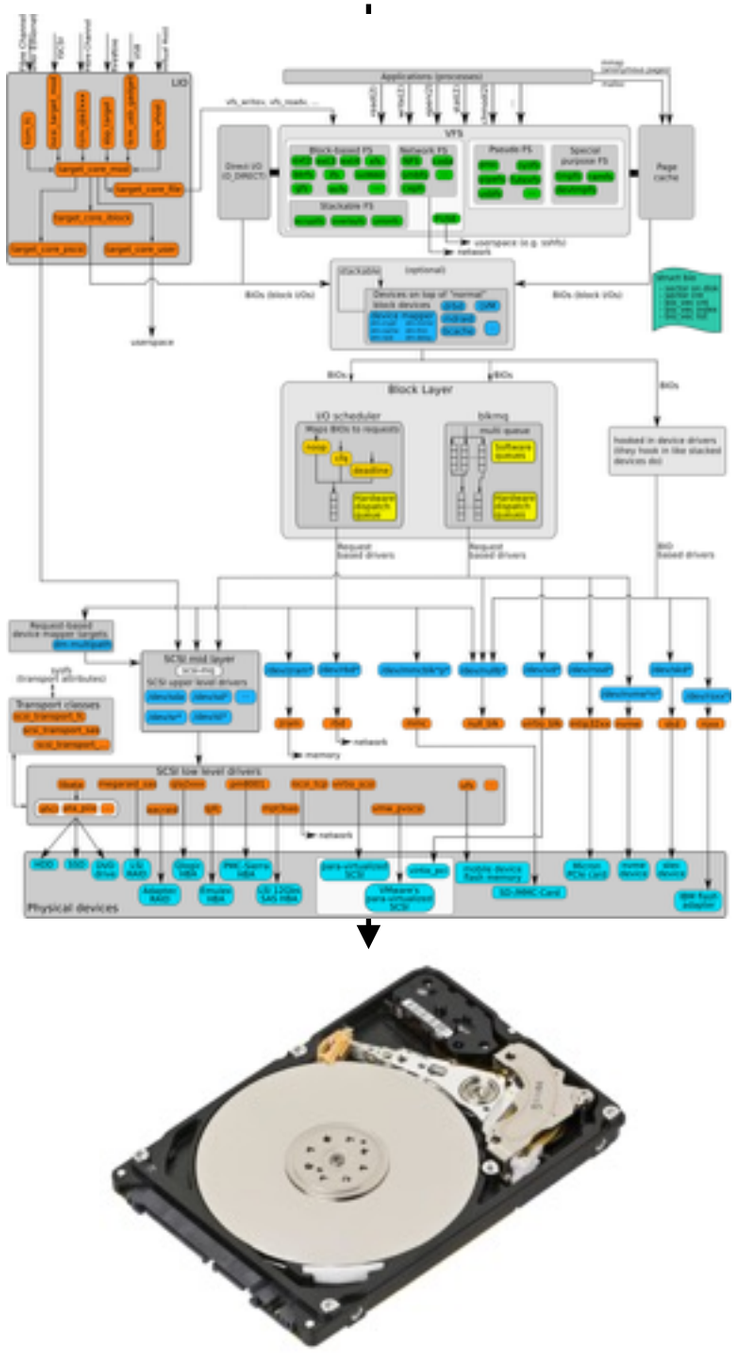
- But the interface says nothing about crash safety

Build a new crash-safe file system (e.g. FSCQ [SOSP’15])

- Comes with extremely high verification burden

Existing work

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```



Formalize the existing POSIX interface (e.g. SibylFS [SOSP’15])

- But the interface says nothing about crash safety

Build a new crash-safe file system (e.g. FSCQ [SOSP’15])

- Comes with extremely high verification burden

Find bugs in existing file systems (e.g. eXplode [OSDI’06])

- Ours is a complementary problem: precisely specifying actual behavior

Crash behavior of modern file systems

Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing crash-consistency models

Building crash-safe applications

Crash behavior of
modern file systems

Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing
crash-consistency models

Building crash-safe
applications

Crash-consistency models

Crash-consistency models

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

Crash-consistency models

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

Formal specifications

Axiomatic descriptions of crash consistency using first order logic

Crash-consistency models

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

Documentation for application developers

Formal specifications

Axiomatic descriptions of crash consistency using first order logic

Crash-consistency models

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

Documentation for application developers

Formal specifications

Axiomatic descriptions of crash consistency using first order logic

Automated reasoning about crash safety

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```

Initial setup
(cannot crash)

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```

Initial setup (cannot crash)

Main body (may crash at any point)

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

```
initial:
  f = create("file")
  write(f, old)

main:
  f = create("file.tmp")
  write(f, new)
  close(f)
  rename("file.tmp", "file")

exists?:
  content("file") != old & content("file") != new
```

Initial setup (cannot crash)

Main body (may crash at any point)

Check whether some (possibly crashing) execution satisfies predicates

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

```
initial:
  f = create("file")
  write(f, old)

main:
  f = create("file.tmp")
  write(f, new)
  close(f)
  rename("file.tmp", "file")

exists?:
  content("file") != old & content("file") != new
```

Initial setup (cannot crash)

Main body (may crash at any point)

Check for behavior that may **surprise** application writers

Check whether some (possibly crashing) execution satisfies predicates

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

memory parallelism

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a ~~file~~ system across ~~crashes~~

memory parallelism

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```

Initially `A = B = 0`

Thread 1	**Thread 2**
`A = 1`	`B = 1`
`r1 = B`	`r2 = A`

Can `r1 = 0 & r2 = 0`?

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

	Litmus test		
	Prefix append	Atomic replace via rename	Atomic create via rename

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

	Litmus test		
File system	Prefix append	Atomic replace via rename	Atomic create via rename
`ext4`	**Unsafe**	**Unsafe**	**Unsafe**

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

	Litmus test		
File system	Prefix append	Atomic replace via rename	Atomic create via rename
ext4	**Unsafe**	**Unsafe**	**Unsafe**
xfs	Safe	**Unsafe**	**Unsafe**
f2fs	**Unsafe**	**Unsafe**	**Unsafe**
nilfs2	Safe	**Unsafe**	**Unsafe**
btrfs	Safe	Safe	**Unsafe**
ufs2	**Unsafe**	**Unsafe**	**Unsafe**

Litmus tests

Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes

File system	Prefix append
`ext4`	**Unsafe**
`xfs`	Safe
`f2fs`	**Unsafe**
`nilfs2`	Safe
`btrfs`	Safe
`ufs2`	**Unsafe**

> We suspect that most modern filesystems exhibit the safe append property.
>
> SQLite Atomic Commit documentation

Crash behavior of
modern file systems

Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing
crash-consistency models

Building crash-safe
applications

Formal specifications

Axiomatic descriptions of crash consistency using first order logic

Ordering constraints on events in traces

```
P =   f = create(“file.tmp”)
      write(f, new)
      close(f)
      rename(“file.tmp”, “file”)
```

Formal specifications

Axiomatic descriptions of crash consistency using first order logic

Ordering constraints on events in traces

```
P =   f = create(“file.tmp”)
      write(f, new)
      close(f)
      rename(“file.tmp”, “file”)
```

create(“foo.tmp”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

Formal specifications

Axiomatic descriptions of crash consistency using first order logic

Ordering constraints on events in traces

```
P =   f = create(“file.tmp”)
      write(f, new)
      close(f)
      rename(“file.tmp”, “file”)
```

create(“foo.tmp”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

A *trace* is a sequence of file system events generated by an execution of P

Formal specifications

Axiomatic descriptions of crash consistency using first order logic

Ordering constraints on events in traces

```
P =   f = create(“file.tmp”)
      write(f, new)
      close(f)
      rename(“file.tmp”, “file”)
```

create(“foo.tmp”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

A *trace* is a sequence of file system events generated by an execution of P

A **crash-consistency model** is a *filter* on traces: it specifies which traces (and prefixes of traces) are allowed.

Formal specifications

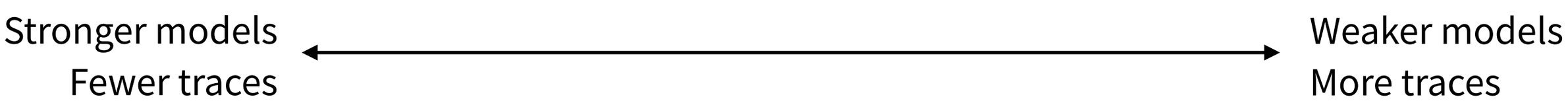
```
P  =   f = create(“file.tmp”)
       write(f, new)
       close(f)
       rename(“file.tmp”, “file”)
```

create(“foo.tmp”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

A **crash-consistency model** is a *filter* on traces: it specifies which traces (and prefixes of traces) are allowed.



Formal specifications

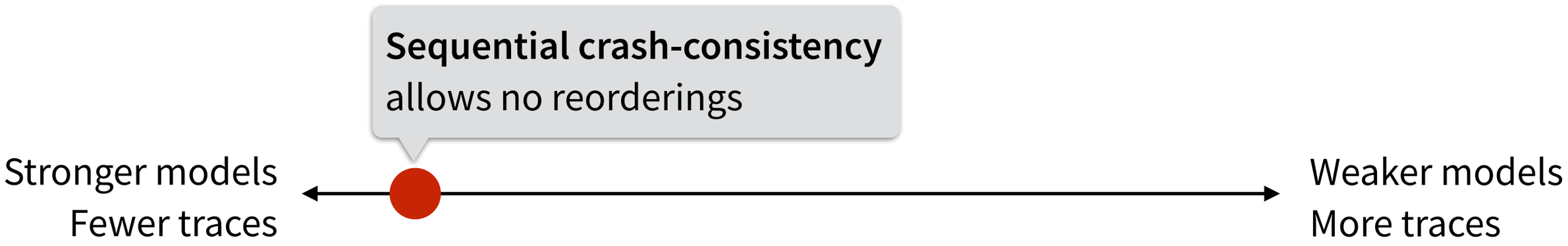
```
P  =  f = create(“file.tmp”)
      write(f, new)
      close(f)
      rename(“file.tmp”, “file”)
```

create(“foo.tmp”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

A **crash-consistency model** is a *filter* on traces: it specifies which traces (and prefixes of traces) are allowed.



Formal specifications

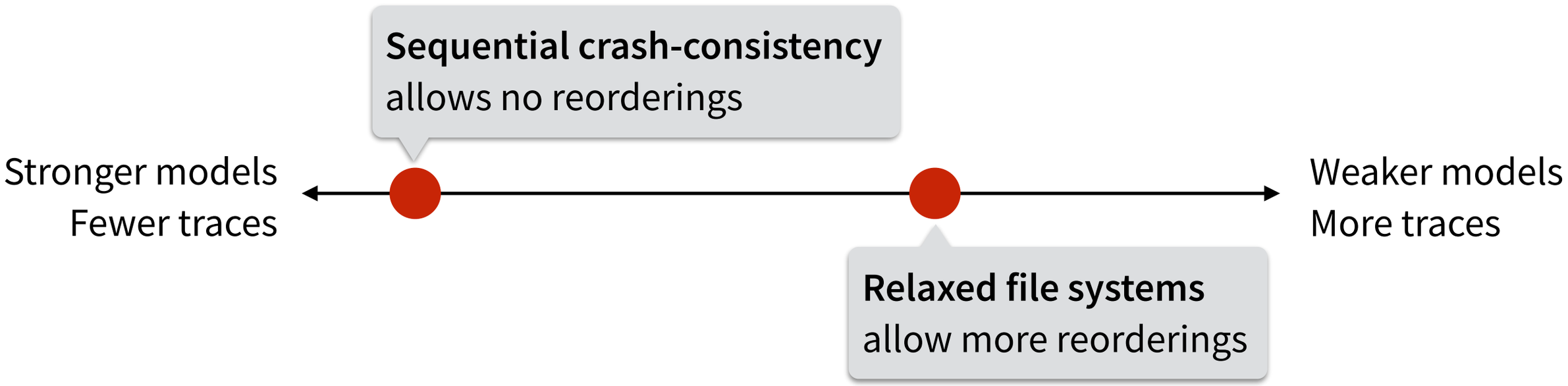
```
P  =  f = create(“file.tmp”)
      write(f, new)
      close(f)
      rename(“file.tmp”, “file”)
```

create(“foo.tmp”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

A **crash-consistency model** is a *filter* on traces: it specifies which traces (and prefixes of traces) are allowed.



ext4 crash-consistency

```
P =  f = create(“file.tmp”)
     write(f, new)
     close(f)
     rename(“file.tmp”, “file”)
```

create(“foo.tmp”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

Definition 7 (ext4 Crash-Consistency). Let t_P be a valid trace and τ_P the corresponding canonical trace. We say that t_P is ext4 *crash-consistent* iff $e_i \leq_{t_P} e_j$ for all events e_i, e_j such that $e_i \leq_{\tau_P} e_j$ and at least one of the following conditions holds:

1. e_i and e_j are metadata updates to the same file: $e_i = \mathit{setattr}(f, k_i, v_i)$ and $e_j = \mathit{setattr}(f, k_j, v_j)$.
2. e_i and e_j are writes to the same block in the same file: $e_i = \mathit{write}(f, a_i, d_i)$, $e_j = \mathit{write}(f, a_j, d_j)$, and $\mathit{sameBlock}(a_i, a_j)$, where *sameBlock* is an implementation-specific predicate.
3. e_i and e_j are updates to the same directory: $\mathit{args}(e_i) \cap \mathit{args}(e_j) \neq \emptyset$, where $\mathit{args}(\mathit{link}(i_1, i_2)) = \{i_1, i_2\}$, $\mathit{args}(\mathit{unlink}(i_1)) = \{i_1\}$, and $\mathit{args}(\mathit{rename}(i_1, i_2)) = \{i_1, i_2\}$.
4. e_i is a write and e_j is an extend to the same file: $e_i = \mathit{write}(f, a_i, d_i)$ and $e_j = \mathit{extend}(f, a_j, d_j, s)$.

ext4 crash-consistency

```
P =  f = create(“file.tmp”)
     write(f, new)
     close(f)
     rename(“file.tmp”, “file”)
```

create(“foo.tmp”)

write(f, “The epoch of …”)

rename(“foo.tmp”, “foo.txt”)

Definition 7 (ext4 Crash-Consistency). Let t_P be a valid trace and τ_P the corresponding canonical trace. We say that t_P is ext4 *crash-consistent* iff $e_i \leq_{t_P} e_j$ for all events e_i, e_j such that $e_i \leq_{\tau_P} e_j$ and at least one of the following conditions holds:

1. e_i and e_j are metadata updates to the same file: $e_i = \mathit{setattr}(f, k_i, v_i)$ and $e_j = \mathit{setattr}(f, k_j, v_j)$.
2. e_i and e_j are writes to the same block in the same file: $e_i = \mathit{write}(f, a_i, d_i)$, $e_j = \mathit{write}(f, a_j, d_j)$, and $\mathit{sameBlock}(a_i, a_j)$, where *sameBlock* is an implementation-specific predicate.
3. e_i and e_j are updates to the same directory: $\mathit{args}(e_i) \cap \mathit{args}(e_j) \neq \emptyset$, where $\mathit{args}(\mathit{link}(i_1, i_2)) = \{i_1, i_2\}$, $\mathit{args}(\mathit{unlink}(i_1)) = \{i_1\}$, and $\mathit{args}(\mathit{rename}(i_1, i_2)) = \{i_1, i_2\}$.
4. e_i is a write and e_j is an extend to the same file: $e_i = \mathit{write}(f, a_i, d_i)$ and $e_j = \mathit{extend}(f, a_j, d_j, s)$.

ext4 crash-consistency: allows traces that respect ordering of:

1. Metadata updates to the same file
2. Same-block writes
3. Same-directory operations
4. Write-append operations

Crash-consistency models

Like memory consistency models but for describing file system crashes

Litmus tests	Formal specifications
Small programs that demonstrate allowed or forbidden behaviors of a file system across crashes	Axiomatic descriptions of crash consistency using first order logic
Documentation for application developers	**Automated reasoning** about crash safety

Crash behavior of
modern file systems

Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing
crash-consistency models

Building crash-safe
applications

Crash behavior of
modern file systems

Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing
crash-consistency models

Building crash-safe
applications

The storage stack is complex

```
write(f, “The age of …”)
write(f, “The epoch of …”)
```

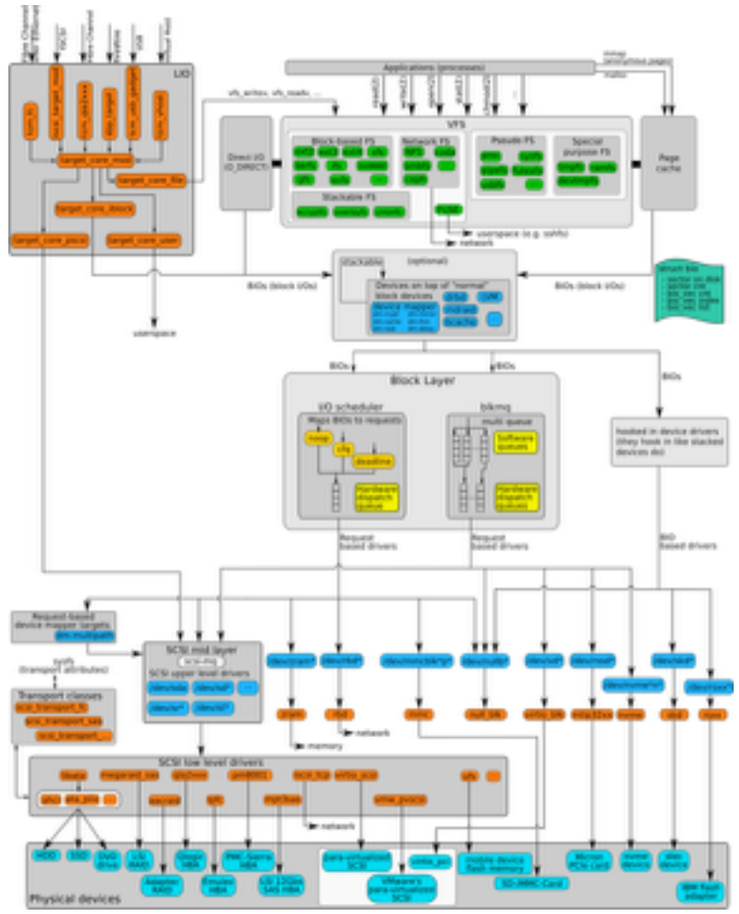


Diagram by Werner Fischer

Building models with Ferrite

Ferrite

Litmus tests

File System
(via QEMU)

Building models with Ferrite

Litmus tests

System calls

File System
(via QEMU)

Ferrite

Building models with Ferrite

Ferrite

Litmus tests

System calls

File System
(via QEMU)

Storage stack

Building models with Ferrite

Ferrite

Litmus tests

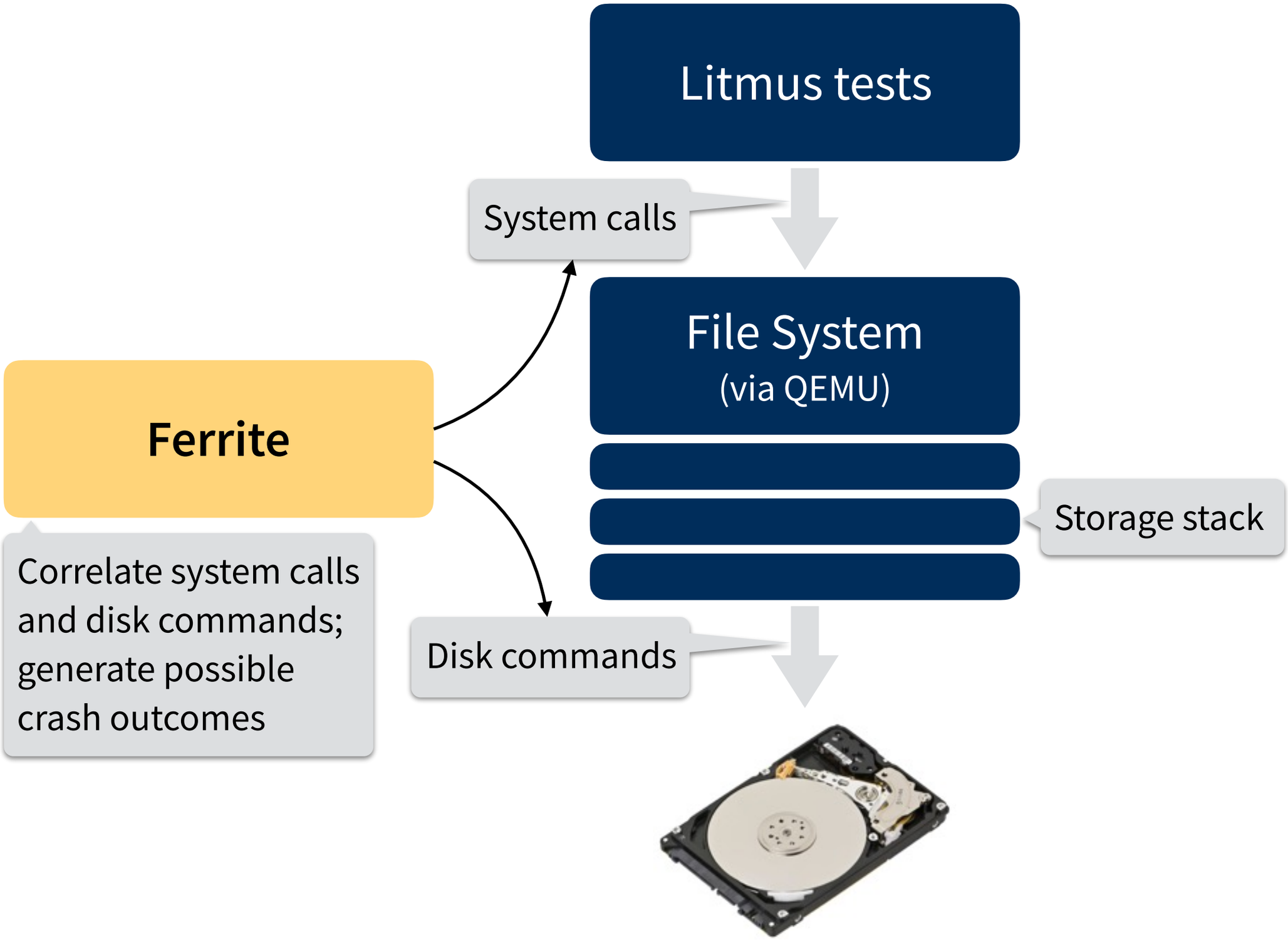
System calls

File System
(via QEMU)

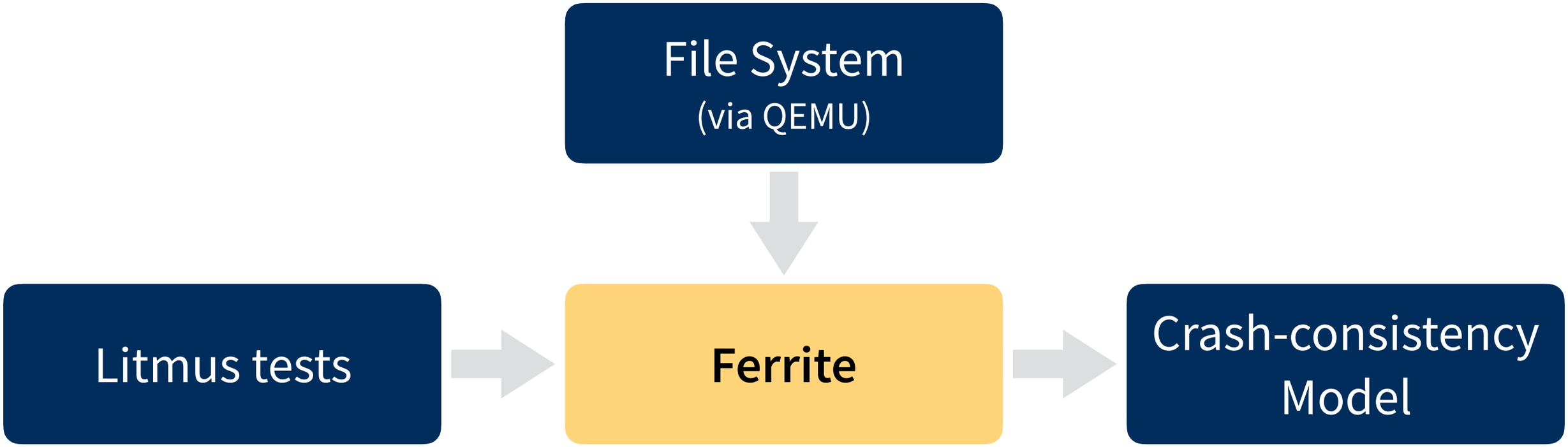
Storage stack

Disk commands

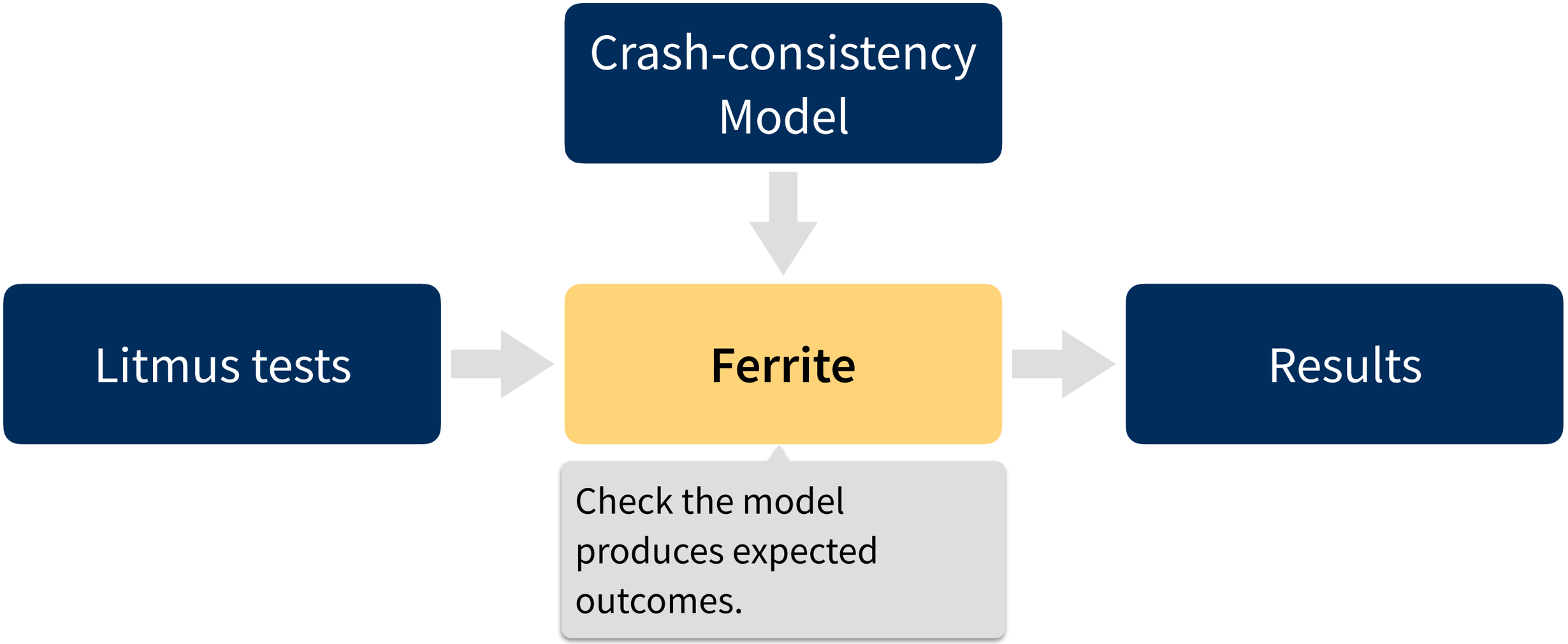
Building models with Ferrite



Building models with Ferrite



Checking models with Ferrite



Crash behavior of
modern file systems

Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing
crash-consistency models

Building crash-safe
applications

Crash behavior of
modern file systems

Crash-consistency models

Litmus tests & formal specifications

Ferrite: developing
crash-consistency models

Building crash-safe
applications

Automating crash consistency

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)
```

```
exists?:
  content(“file”) != old & content(“file”) != new
```

Automating crash consistency

```
initial:
  f = create(“file”)
  write(f, old)

main:
  fsync(f)
  f = create(“file.tmp”)
  fsync(f)
  write(f, new)
  fsync(f)
  close(f)
  fsync(f)
  rename(“file.tmp”, “file”)
  fsync(f)

exists?:
  content(“file”) != old & content(“file”) != new
```

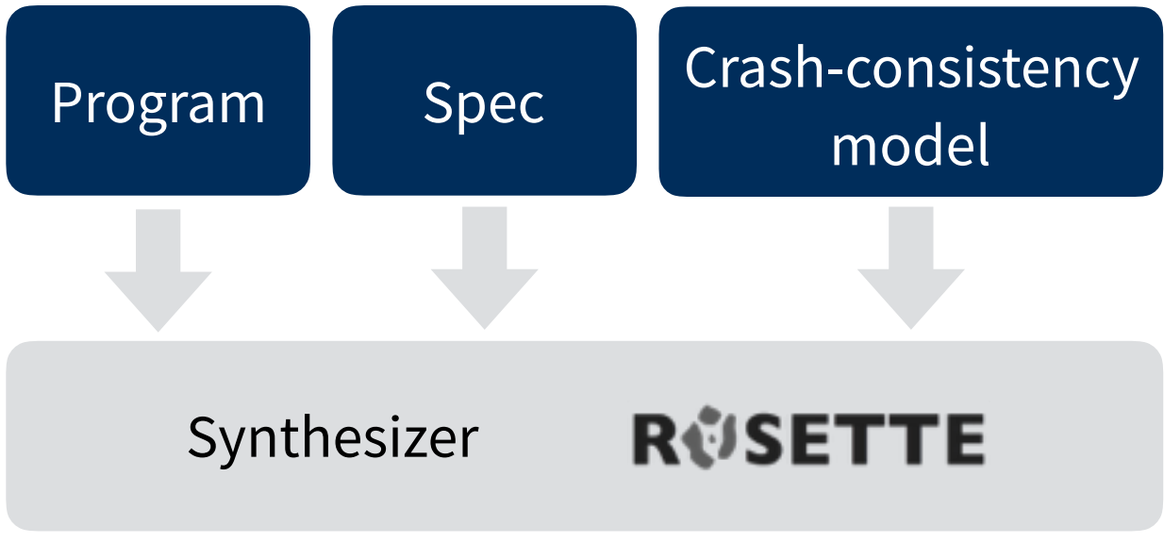
Synthesizing crash consistency


```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)
```

```
exists?:
  content(“file”) != old & content(“file”) != new
```

Synthesizing crash consistency

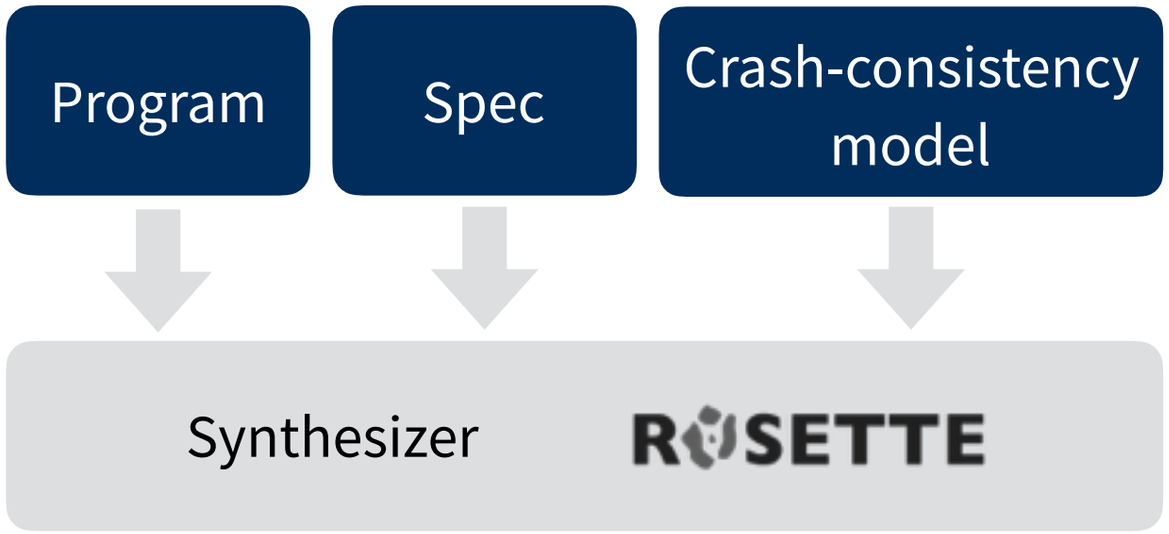
```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  close(f)
  rename(“file.tmp”, “file”)
```




```
exists?:
  content(“file”) != old & content(“file”) != new
```

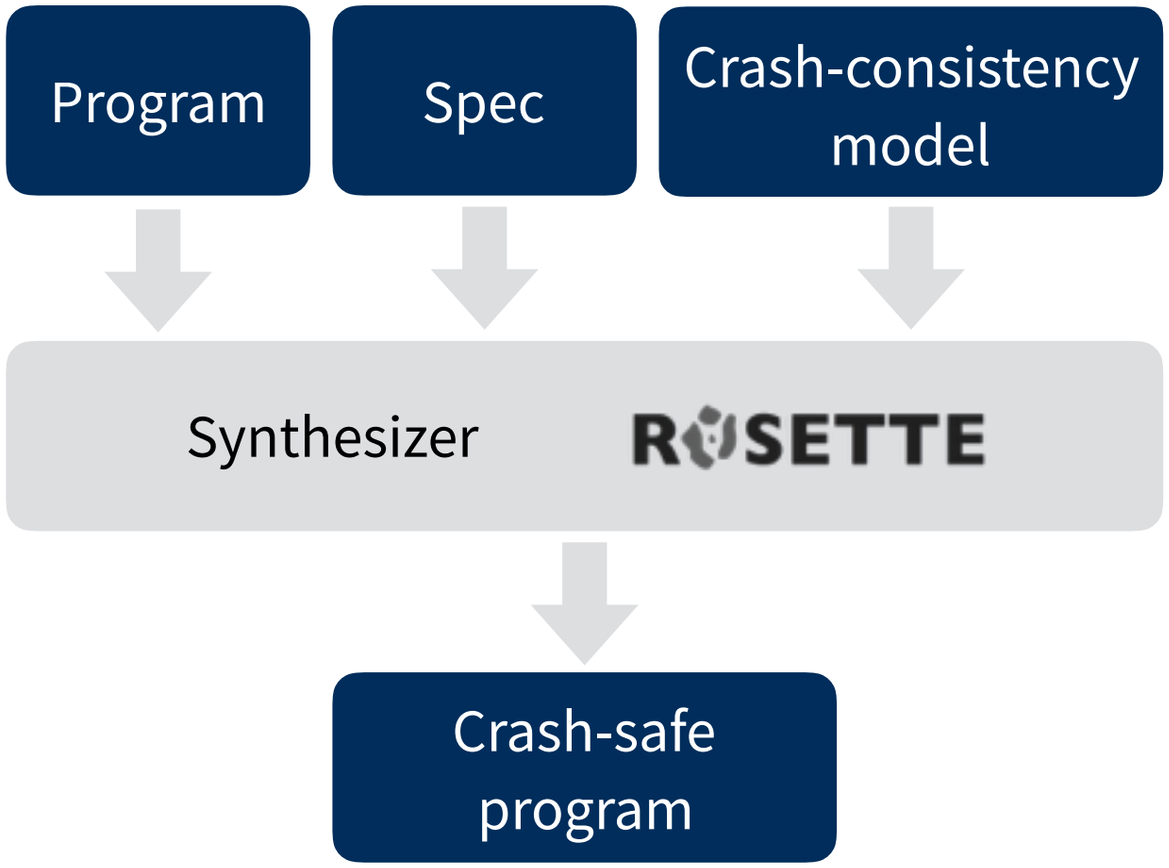
Synthesizing crash consistency

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  fsync(f)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```



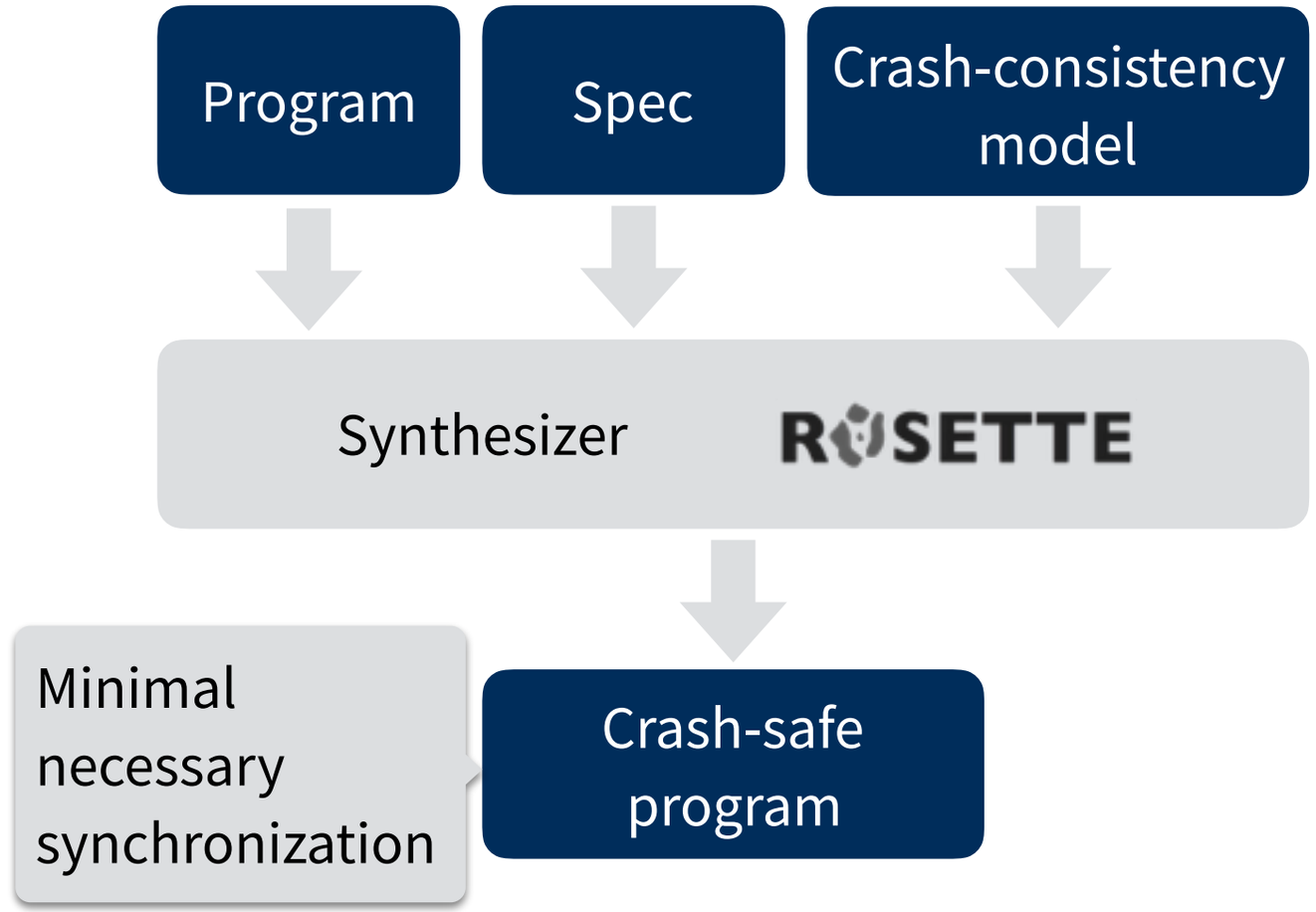
Synthesizing crash consistency

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  fsync(f)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```

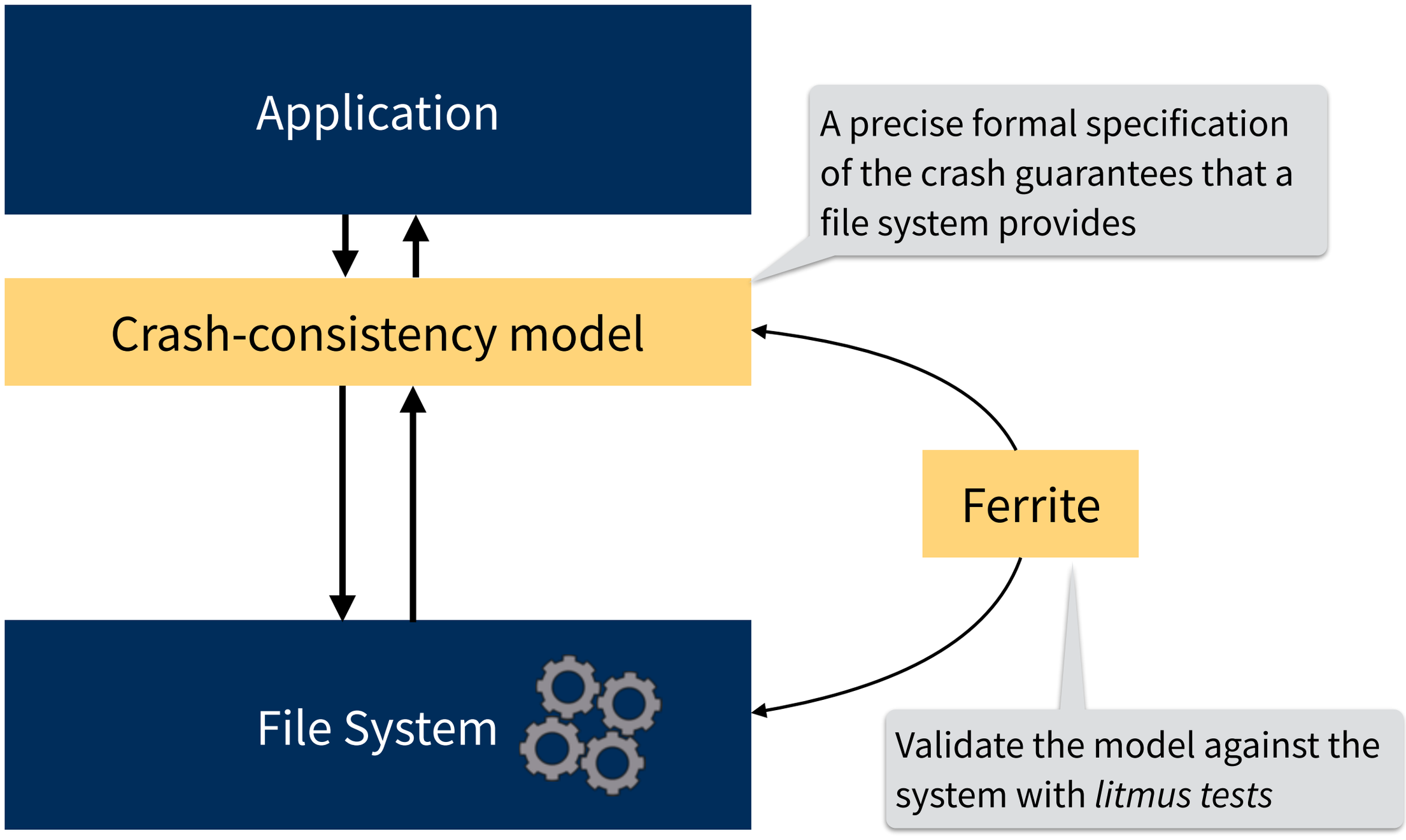


Synthesizing crash consistency

```
initial:
  f = create(“file”)
  write(f, old)

main:
  f = create(“file.tmp”)
  write(f, new)
  fsync(f)
  close(f)
  rename(“file.tmp”, “file”)

exists?:
  content(“file”) != old & content(“file”) != new
```



Crash-consistency models



Crash-consistency models

A DNA-Based Archival Storage System
Wednesday, right before lunch