BEYOND STORAGE APIS: **PROVABLE SEMANTICS FOR STORAGE STACKS**

RAMNATTHAN ALAGAPPAN

VIJAY CHIDAMBARAM

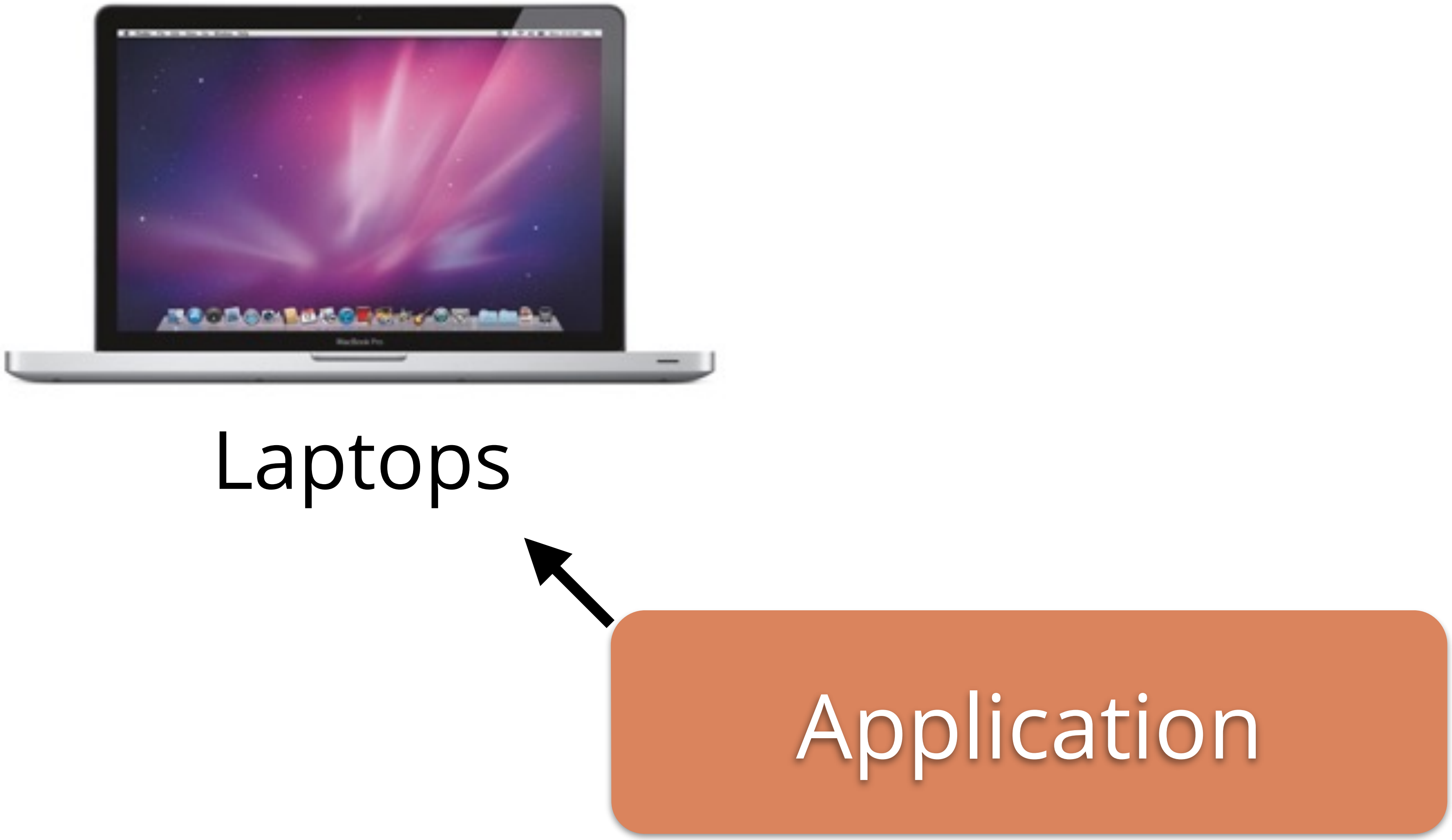
THANUMALAYAN PILLAI

AWS ALBARGHOUTHI

REMZI ARPACI-DUSSEAU

ANDREA ARPACI-DUSSEAU

UNIVERSITY OF WISCONSIN-MADISON


Laptops
Application

Laptops

Desktops

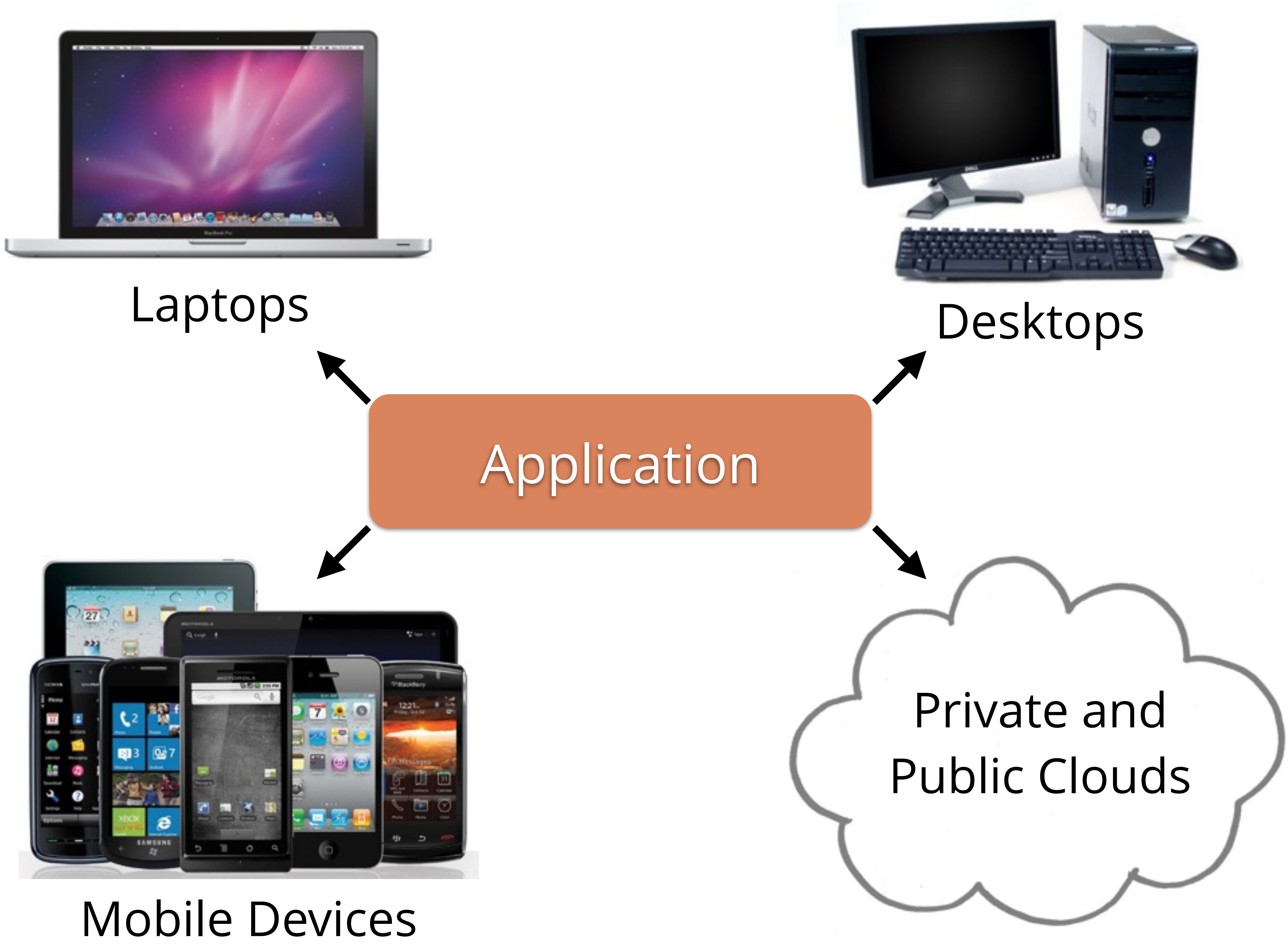
Application

Laptops

Desktops

Application

Mobile Devices


Laptops
Desktops
Application
Mobile Devices
Private and
Public Clouds

Laptops

Desktops

Heterogeneity of environments is increasing

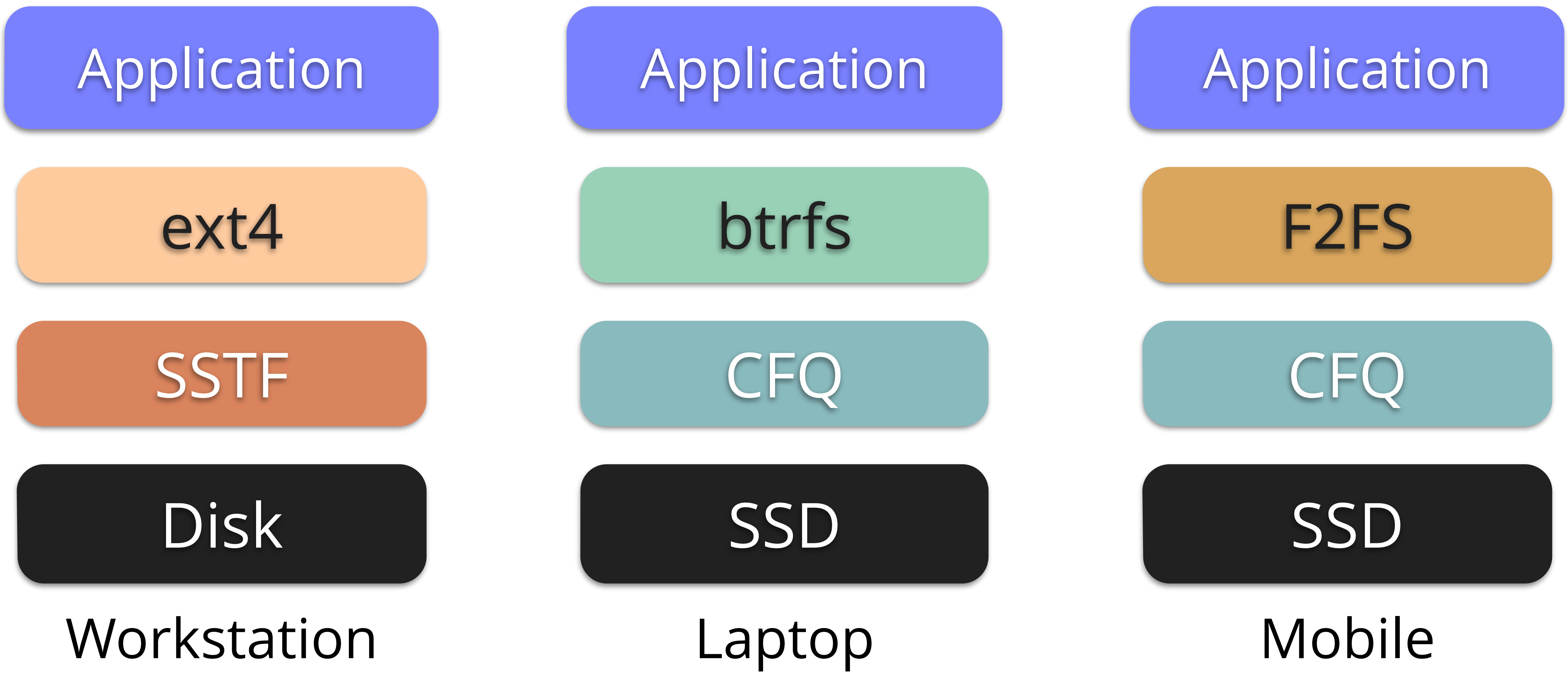
Mobile Devices

STORAGE STACKS: DEEP AND DIVERSE

Windows IO stack has **18** layers! [ThereskaSOSP13]

STORAGE STACKS: DEEP AND DIVERSE

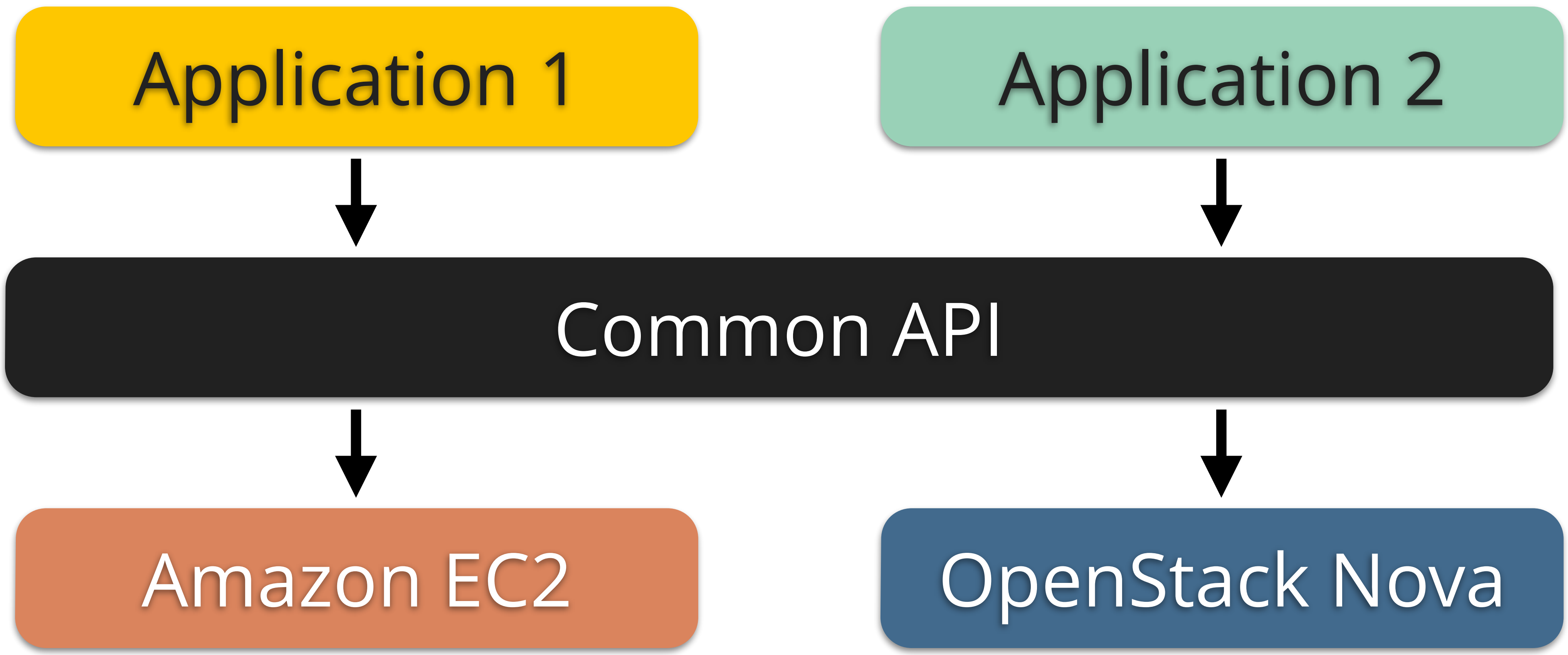
Windows IO stack has **18** layers! [ThereskaSOSP13]



APPLICATION PORTABILITY

Applications should be portable between environments

- Reduce development effort and bugs
- Avoid **vendor lock-in**



API Compatibility is **not enough**!

API Compatibility is **not enough**!

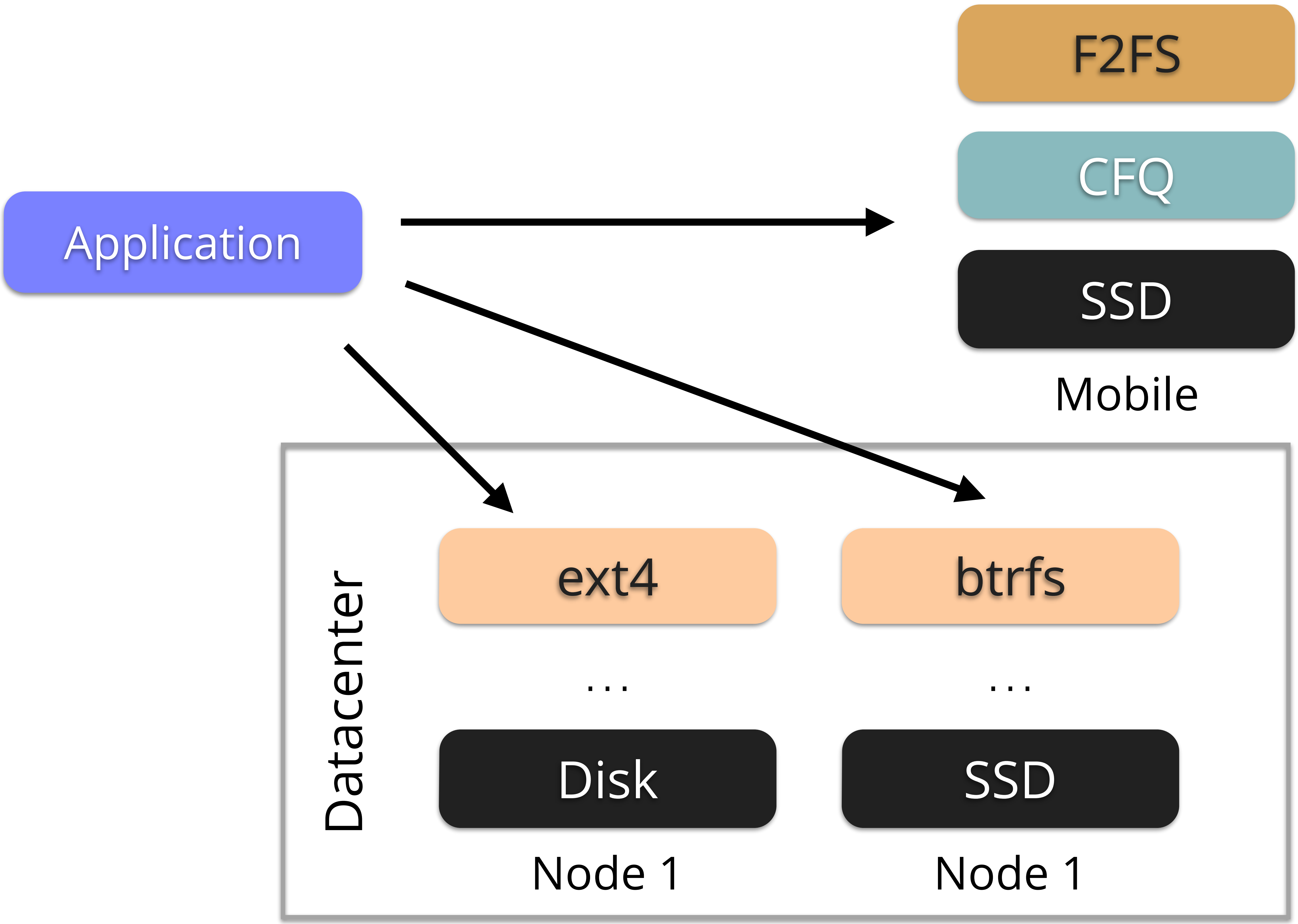
Application correctness depends upon **unspecified properties** of the storage stack

API Compatibility is **not enough**!

Application correctness depends upon **unspecified properties** of the storage stack

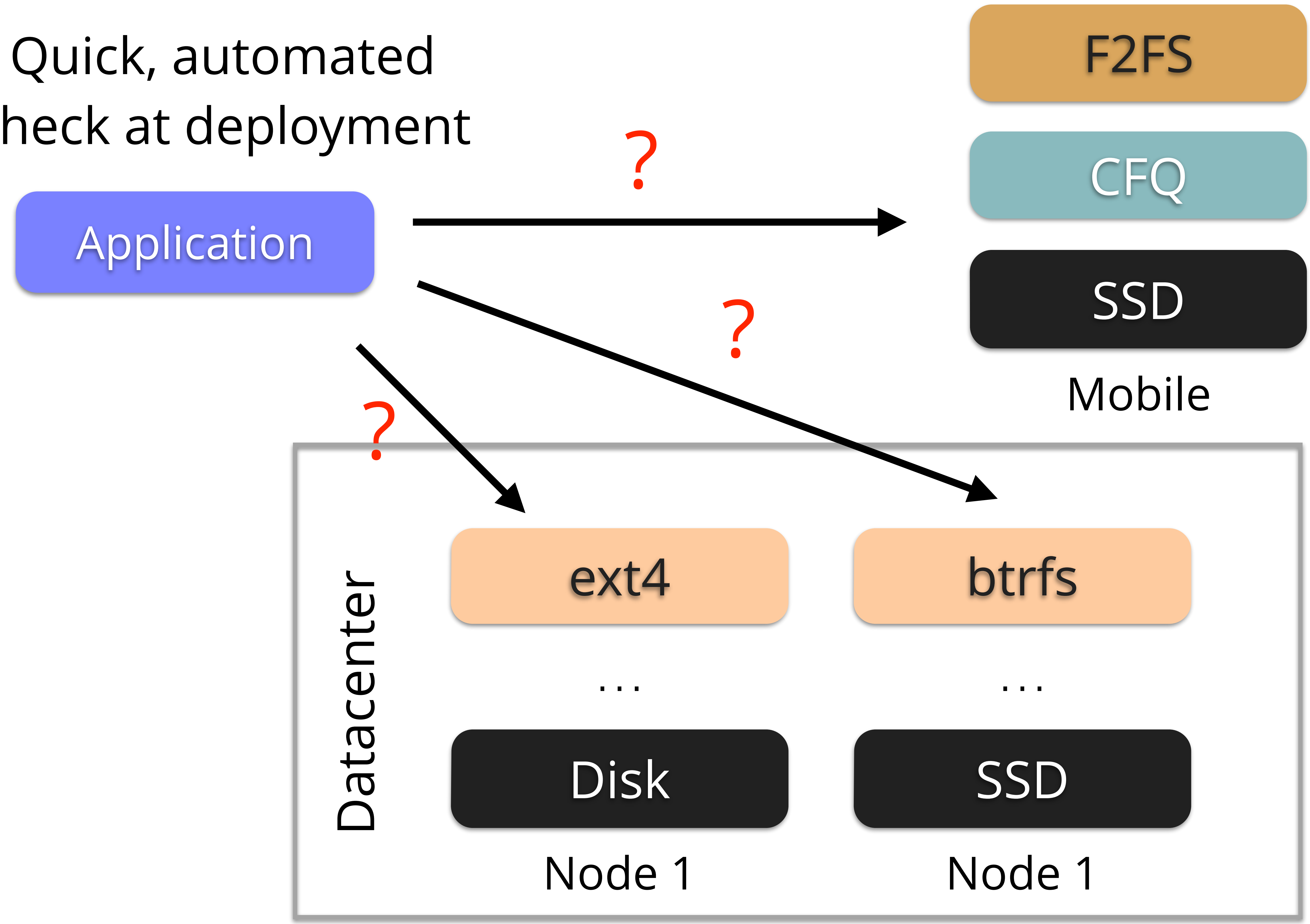
Results: data corruption, data loss, unavailability [PillaiOSDI14]

THE VISION


F2FS
CFQ
SSD
Mobile
Application
Datacenter
ext4
btrfs
...
...
Disk
SSD
Node 1
Node 1

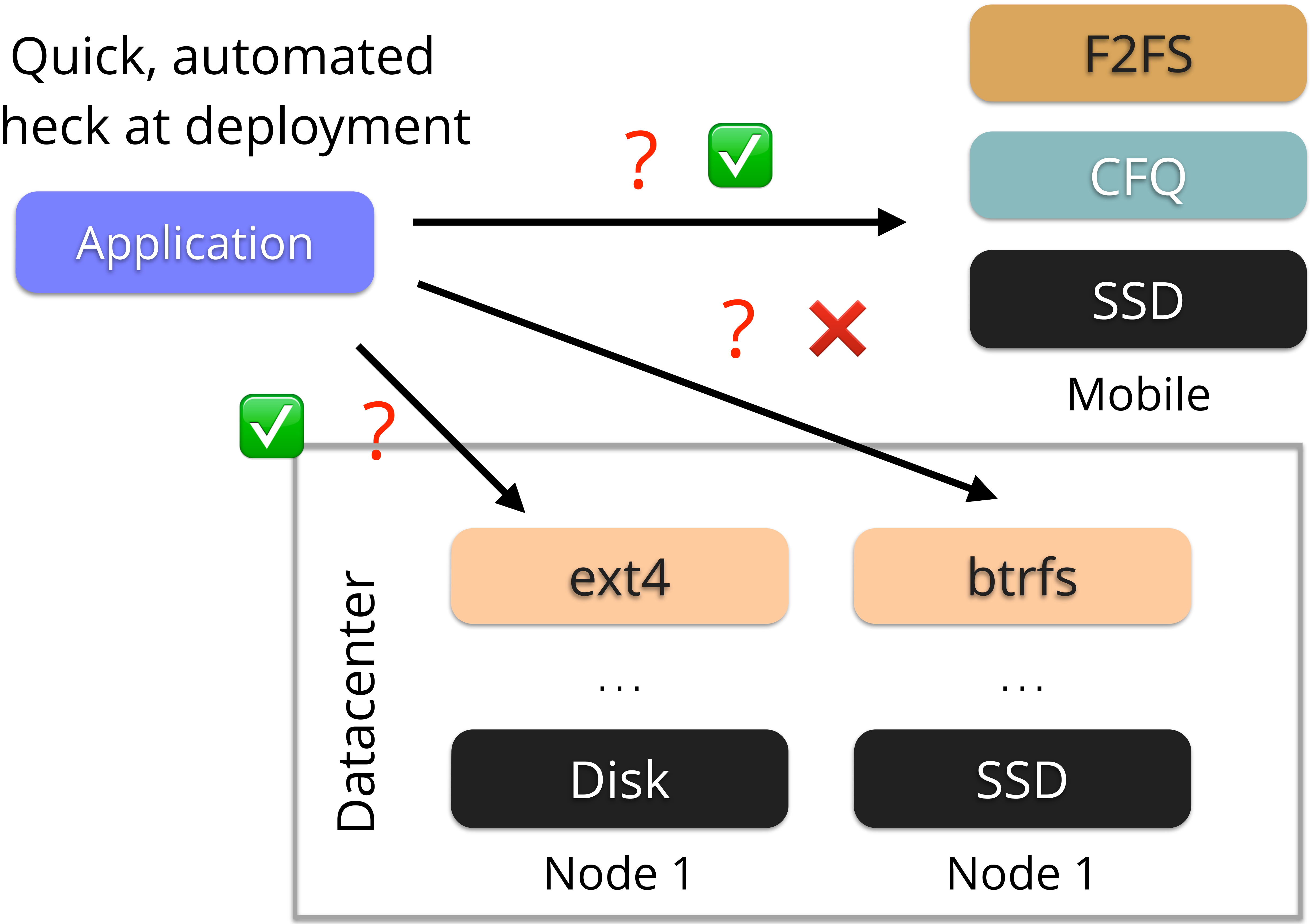
THE VISION

Quick, automated check at deployment

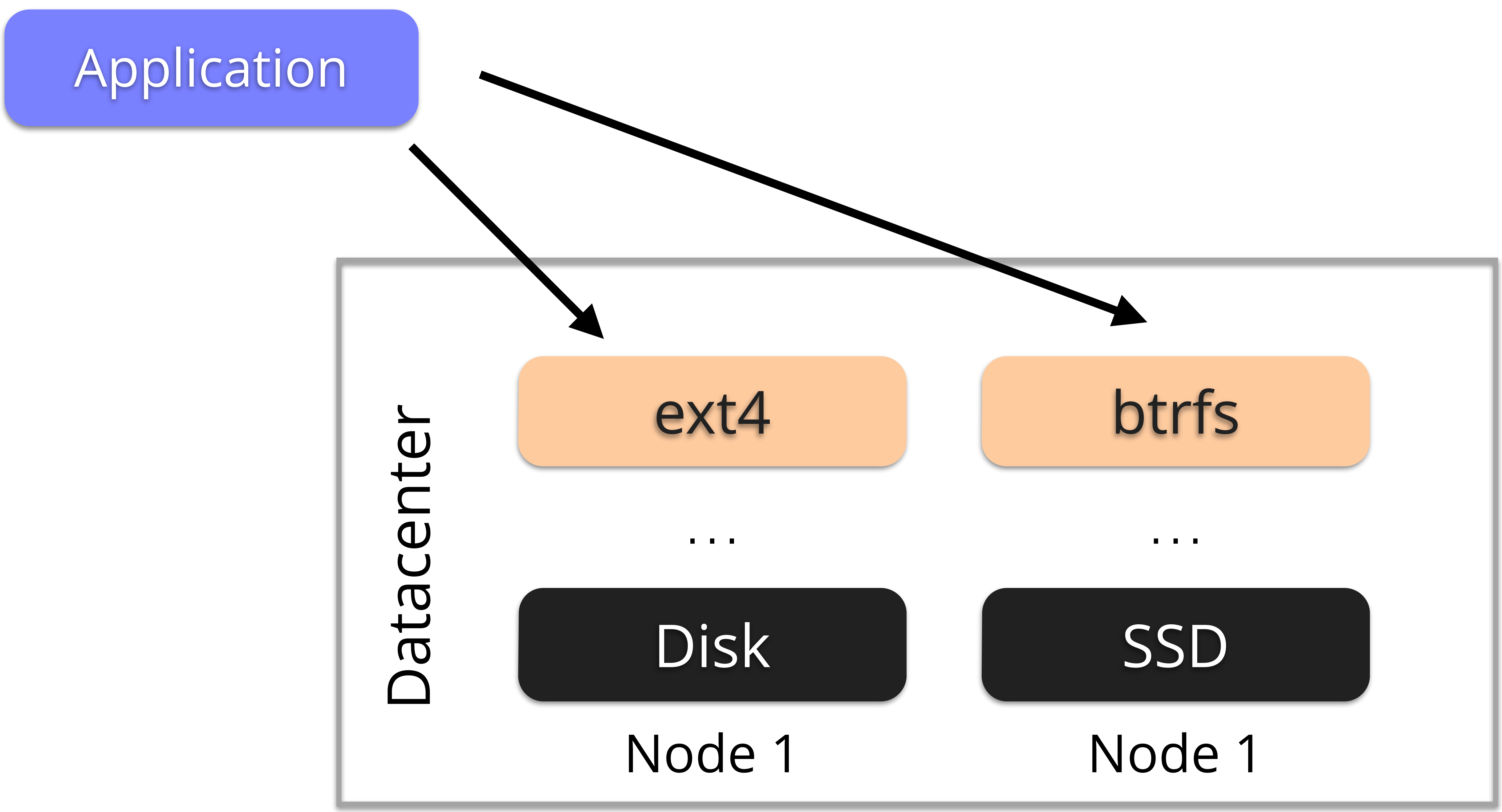


THE VISION

Quick, automated check at deployment

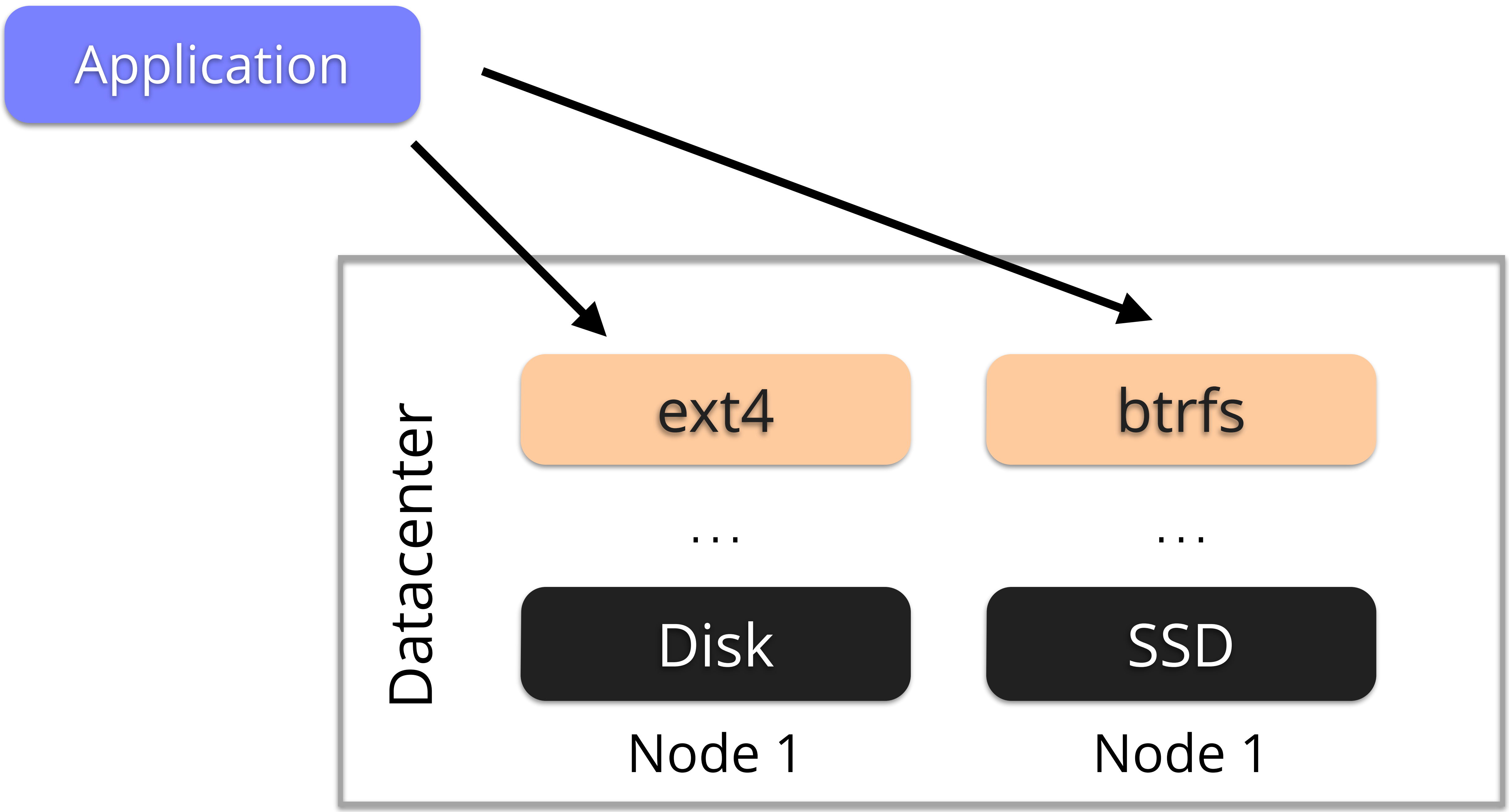


THE VISION



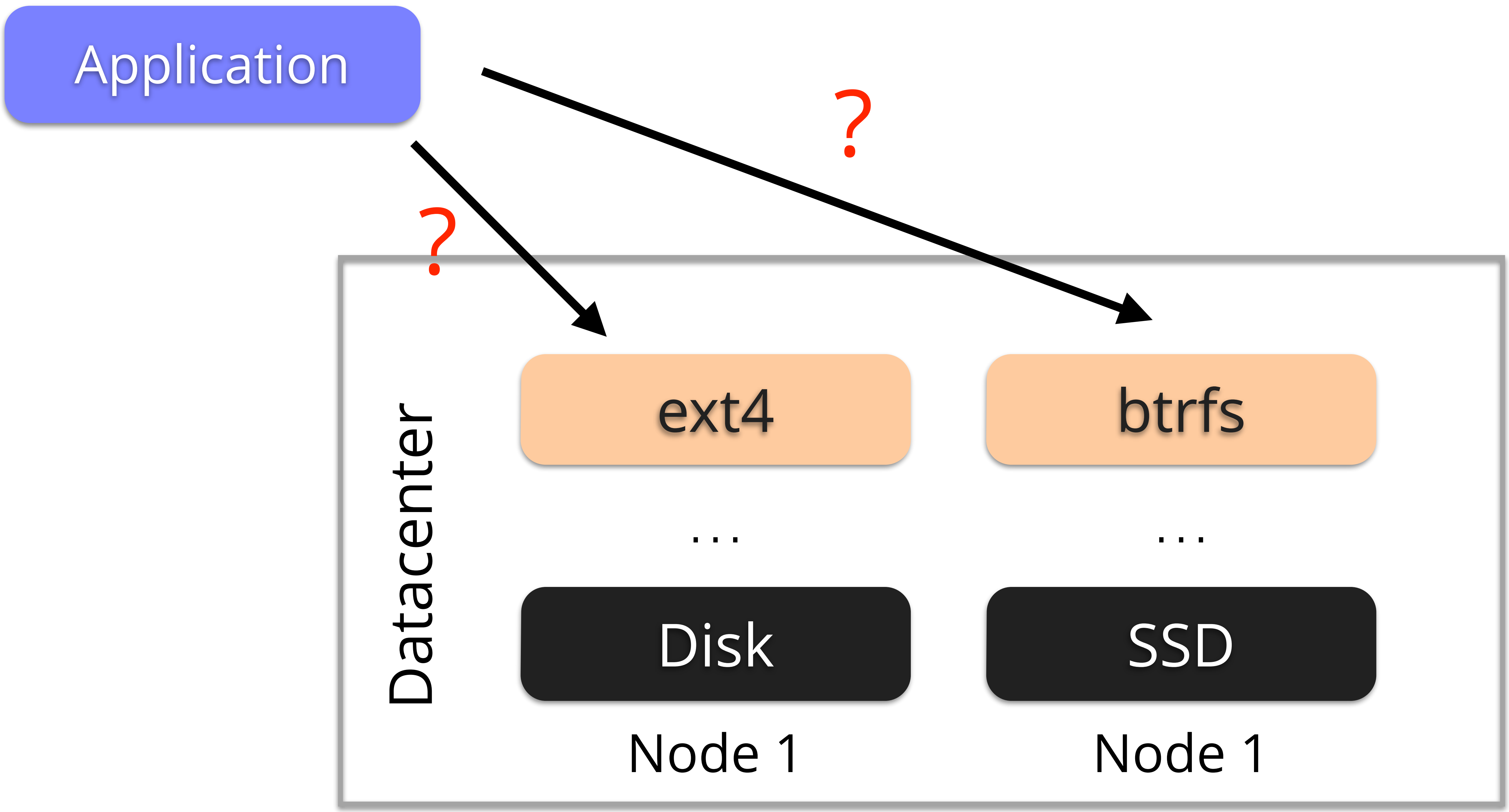
THE VISION

Which is the **best** node to deploy this application to?



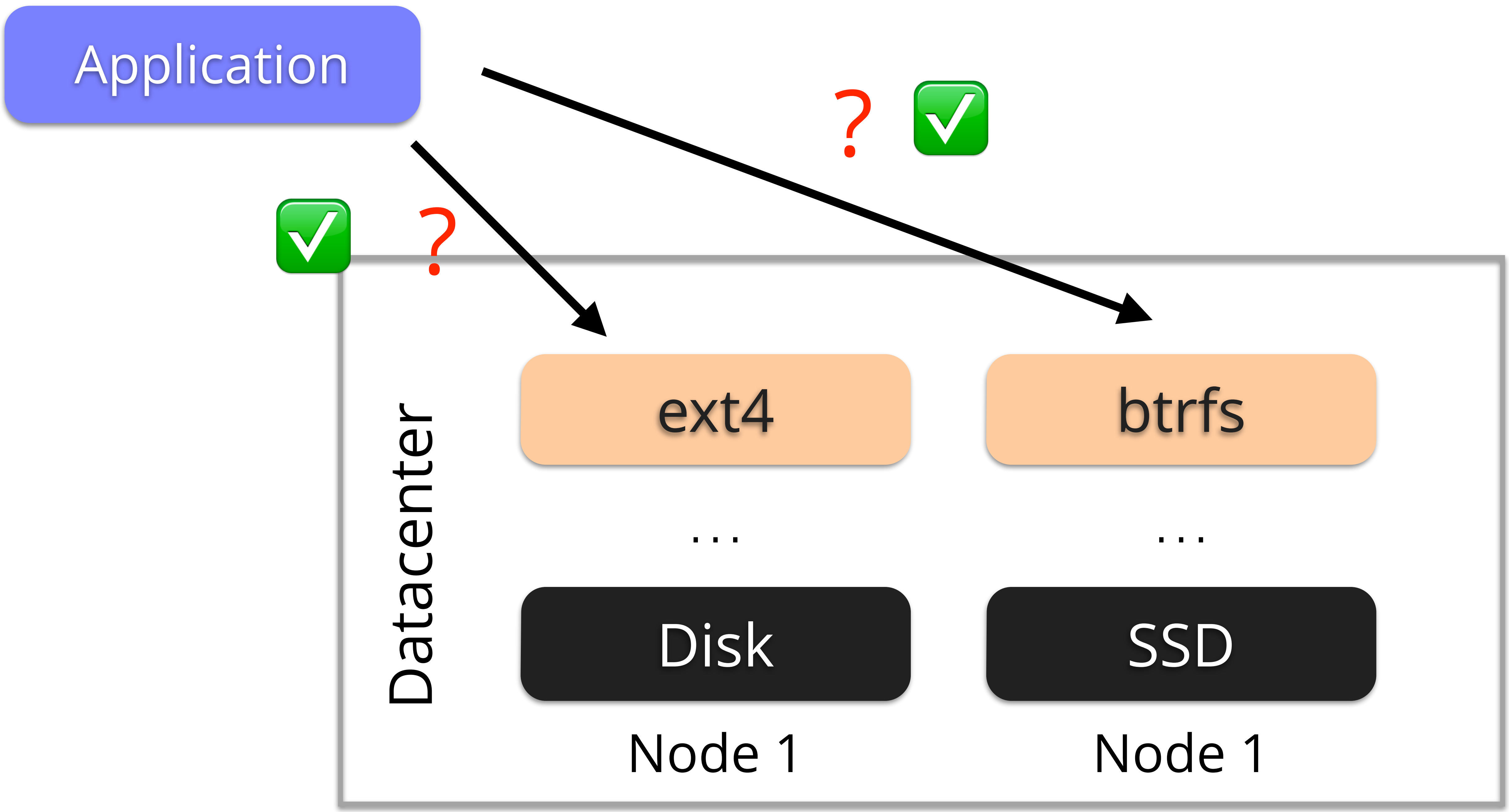
THE VISION

Which is the **best** node to deploy this application to?



THE VISION

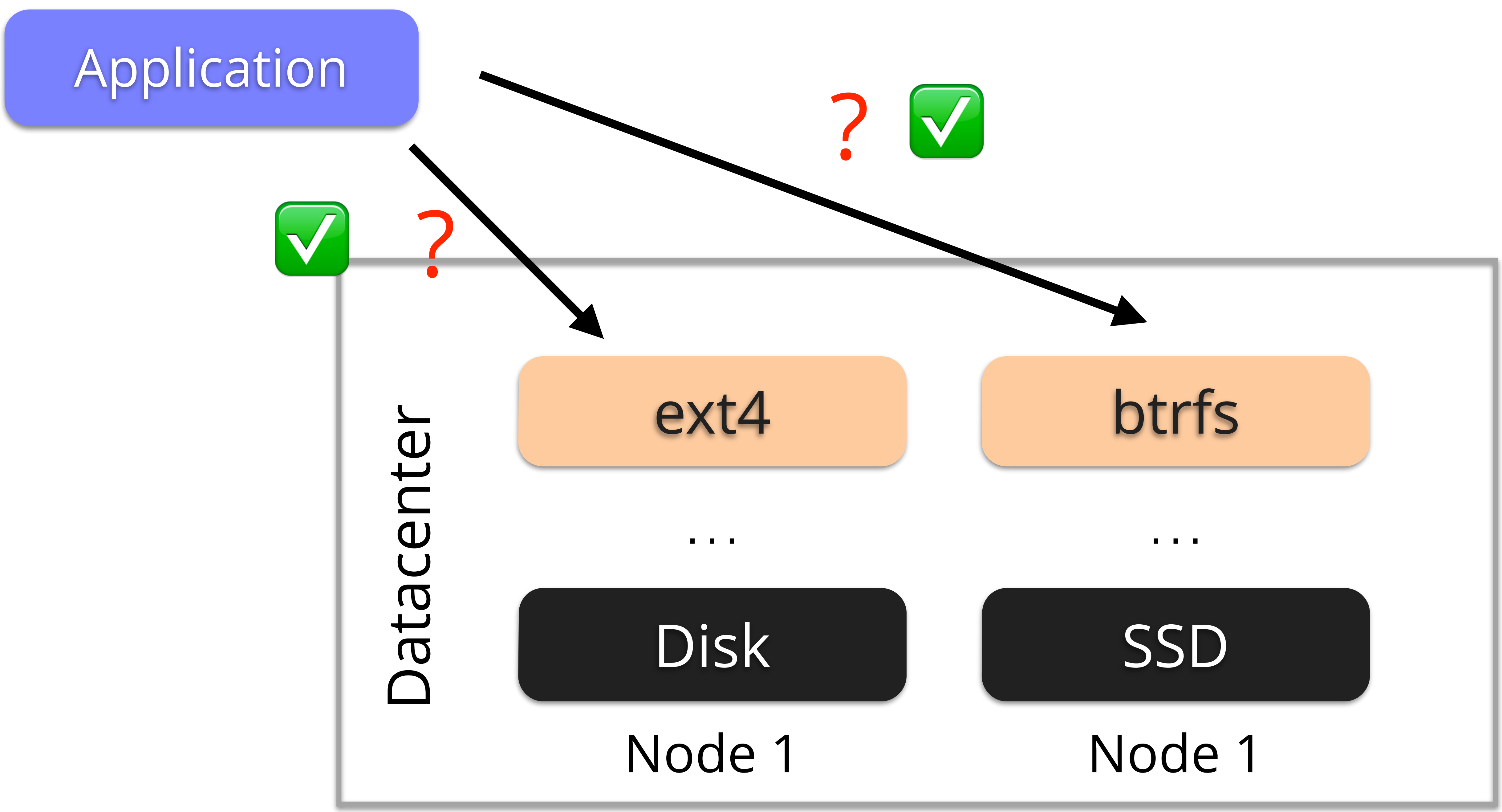
Which is the **best** node to deploy this application to?



THE VISION

Which is the **best** node to deploy this application to?

Best: least # of stack layers, least utilized, etc.



OUTLINE

1. Introduction
2. Portability Bug Study
3. First Steps Toward The Vision
4. The Road Ahead

PORTABILITY BUG STUDY

Portability bug: bug that occurs when an application is **moved** to a **different** environment

Studied public bug databases

- Android deployed on different mobile devices
- Applications run on cloud platforms and on NFS

Performed our own experiments based on previous work [PillaiOSDI14]

OPERATIONS NOT SUPPORTED

SQLite creates temporary files
by opening a file and unlinking them

Not supported
by the daemon emulating FAT32
on the sd card

UNEXPECTED ERROR CODES

MySQL

↓

NFS

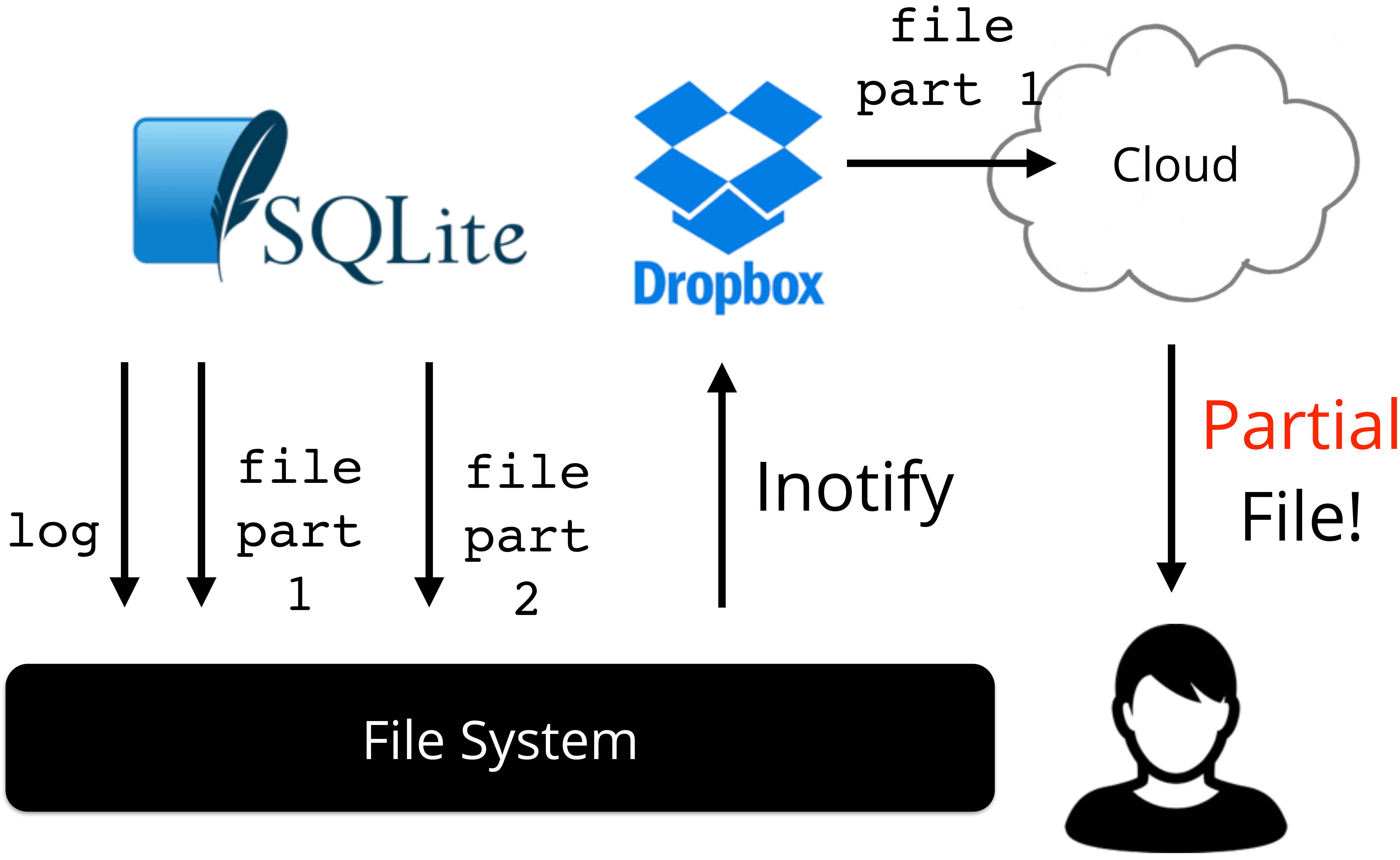
↓

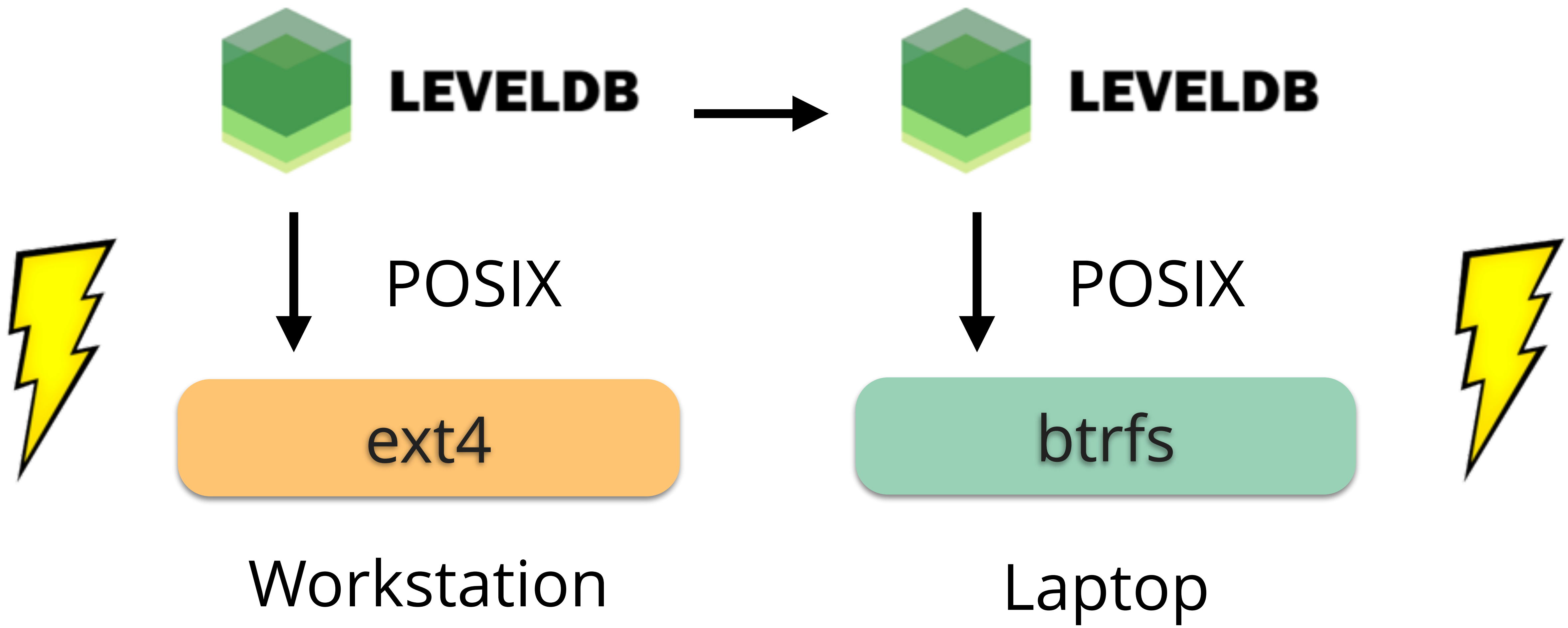
FreeBSD

`fsync()` on
FreeBSD returns `ENOLCK`
even on success

MySQL restarts when
it sees that error

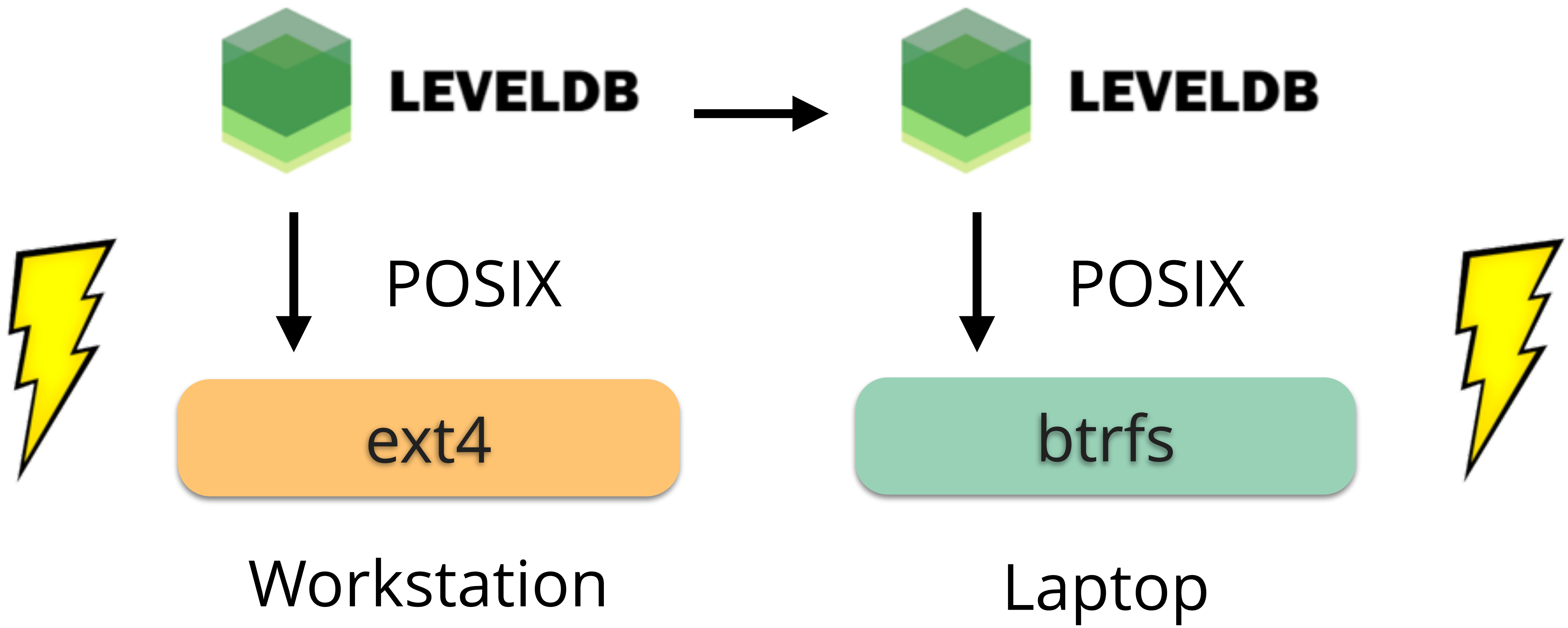
ORDERING REQUIREMENTS NOT MET





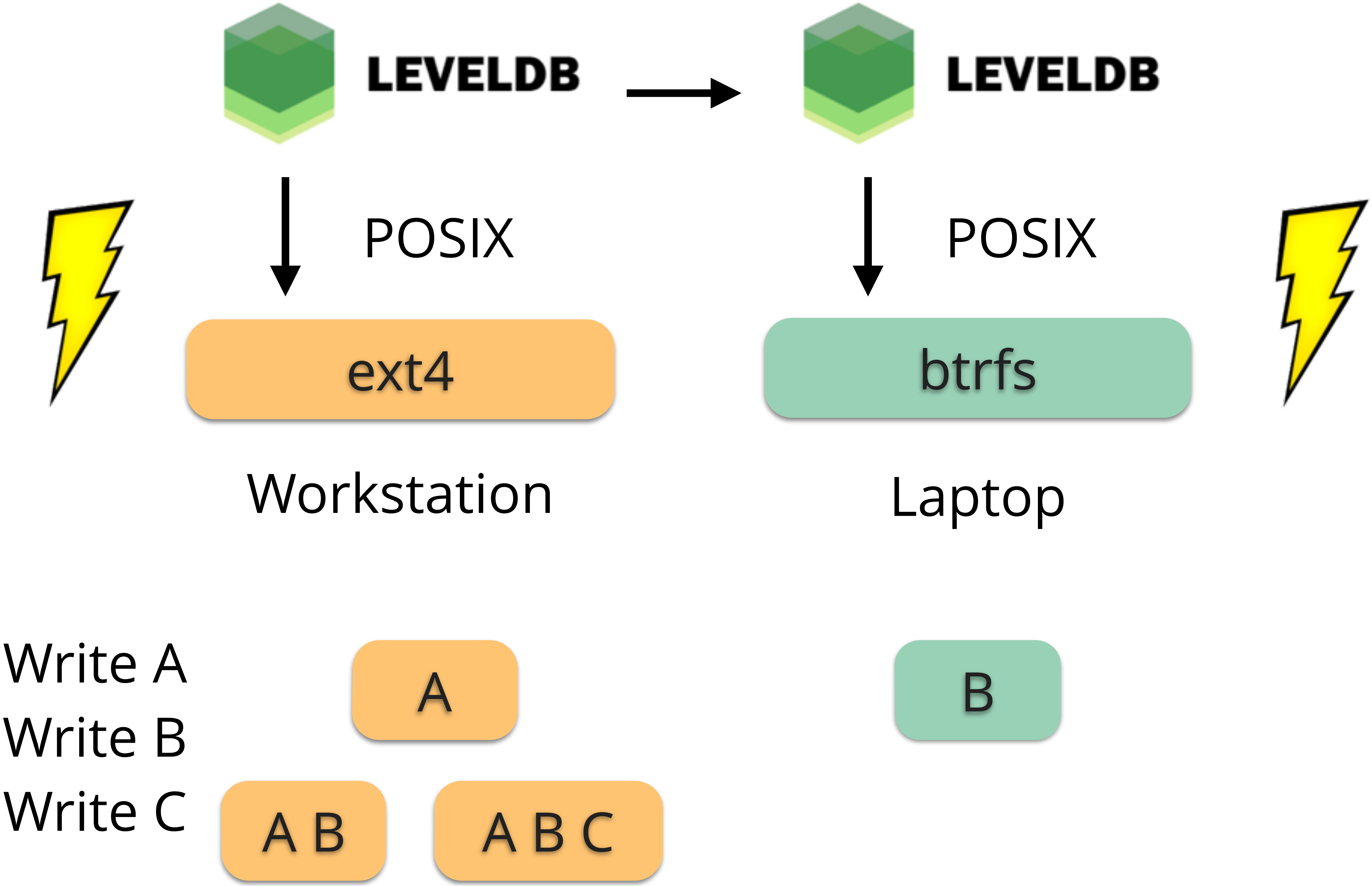
Guarantee: Committed data can always be read back after a crash

All file systems are not created equal: On the complexity of crafting crash-consistent applications, OSDI 2014

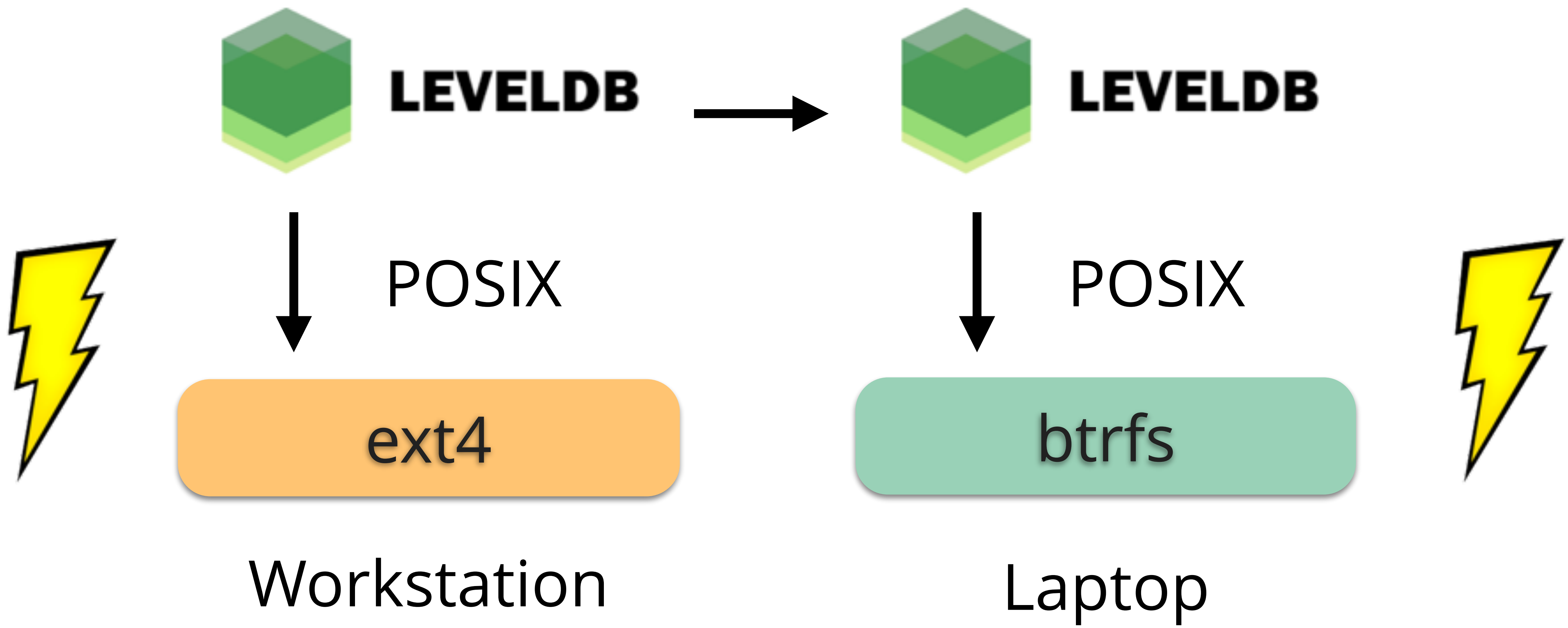


~~Guarantee: Committed data can always be read back after a crash~~

All file systems are not created equal: On the complexity of crafting crash-consistent applications, OSDI 2014



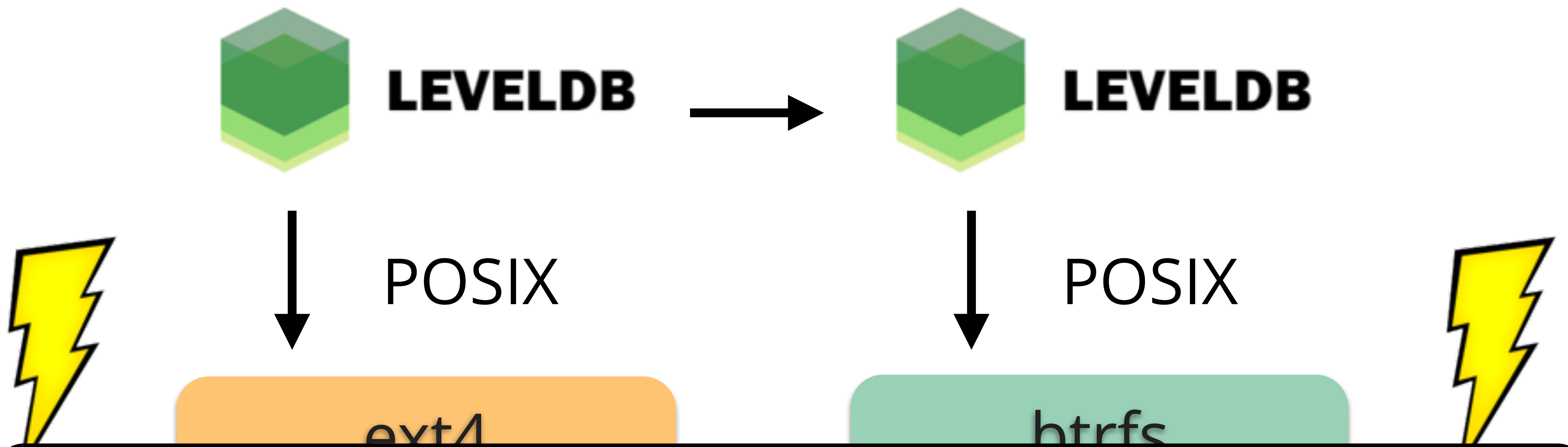
All file systems are not created equal: On the complexity of crafting crash-consistent applications, OSDI 2014



We term this an

Application Crash Vulnerability

All file systems are not created equal: On the complexity of crafting crash-consistent applications, OSDI 2014



API compatibility is not enough!

We term this an

Application Crash Vulnerability

All file systems are not created equal: On the complexity of crafting crash-consistent applications, OSDI 2014

OUTLINE

1. Introduction
2. Portability Bug Study
3. First Steps Toward The Vision
4. The Road Ahead

Formally verify that an application will run correctly on a **given storage stack**

What application **requires**	**<=**	What storage stack **provides**

WHY IS THIS HARD?

What application **requires** <= What storage stack **provides**

Application requirements can be complex

- e.g., append("AB") should result in file containing A or AB
- if then else form

Binary or numerical checks are not sufficient

WHY IS THIS HARD?

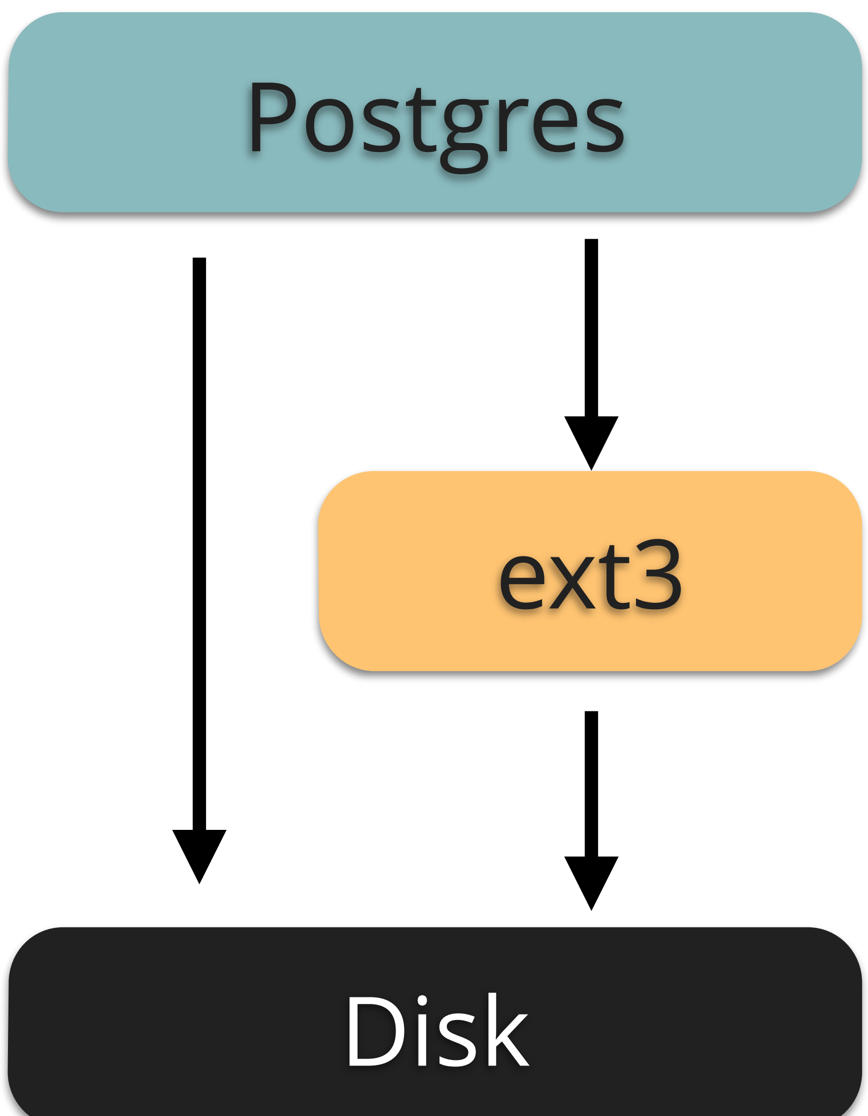
What application **requires** <= What storage stack **provides**

Need **expressive language** for specifying application requirements

Binary or numerical checks are not sufficient

WHY IS THIS HARD?

What application **requires** <= What storage stack **provides**



Postgres provides
ACID transactions

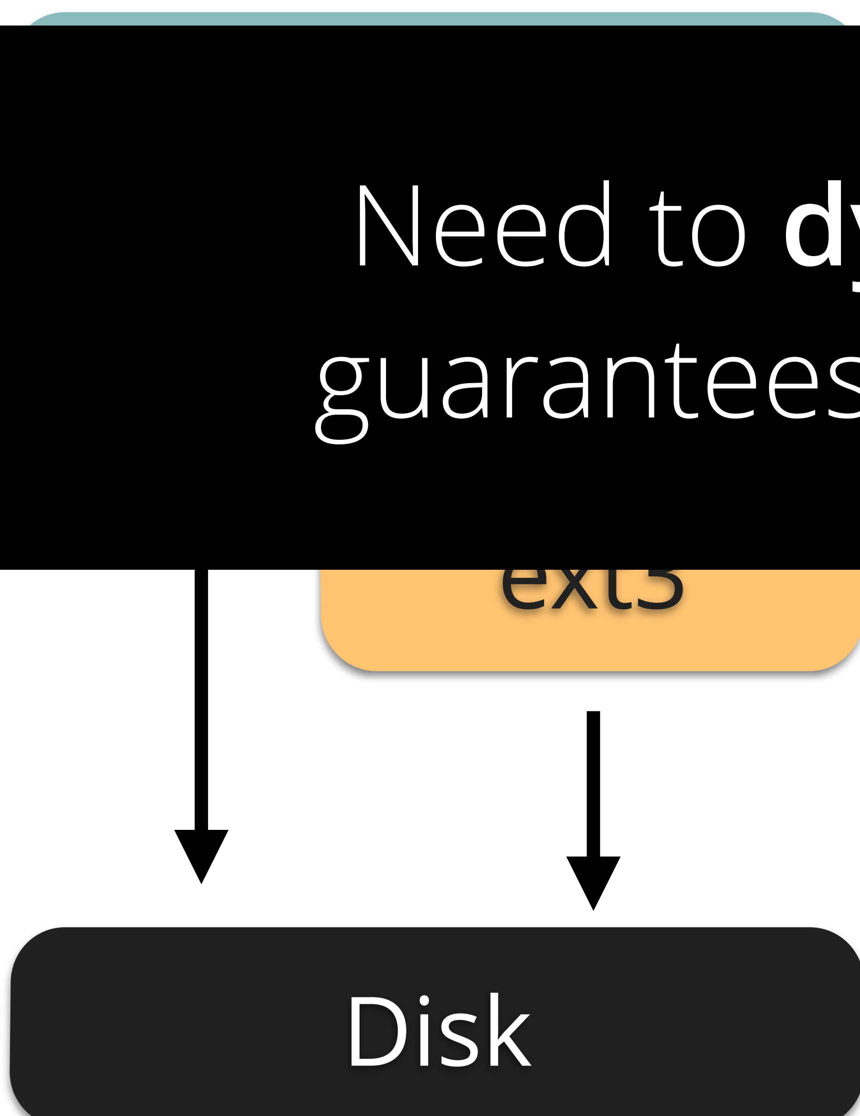
File system provides
atomic metadata operations

Disk provides
atomic reads and writes

WHY IS THIS HARD?

What application **requires** <= What storage stack **provides**

Need to **dynamically compute** guarantees provided by the stack



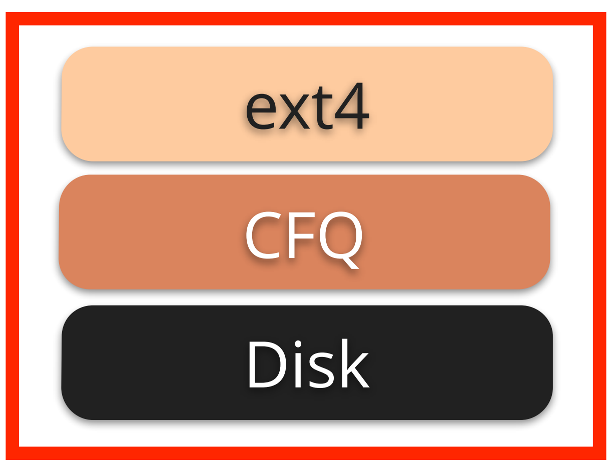
atomic metadata operations

Disk provides atomic reads and writes

OVERVIEW

Model guarantees the application requires from storage as a **theorem**

Ex: application will be crash-consistent if all writes are **ordered** and **atomic**



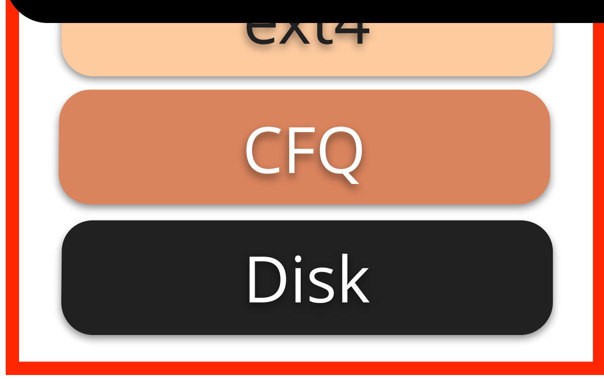
Model guarantees provided by each layer of the storage stack as **axioms**

Ex: disk guarantees **sector-level** reads and writes are **atomic** even with crash

OVERVIEW

Model guarantees the application requires from storage as a **theorem**

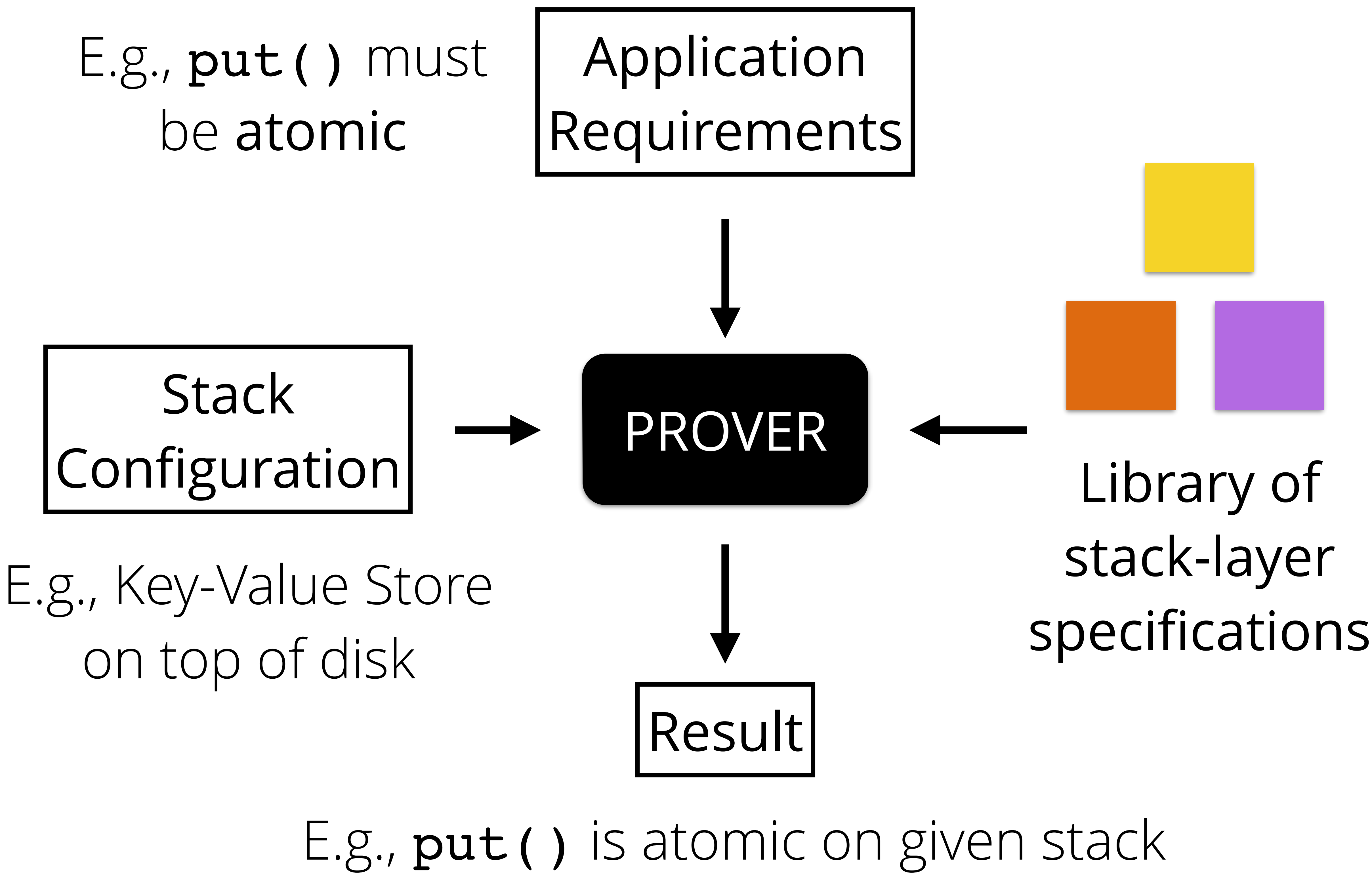
Prove application theorem using axioms from storage stack



each layer of the storage stack as **axioms**

Ex: disk guarantees **sector-level** reads and writes are **atomic** even with crash

SYSTEM


E.g., put() must be atomic
Application Requirements
Stack Configuration
PROVER
Library of stack-layer specifications
E.g., Key-Value Store on top of disk
Result
E.g., put() is atomic on given stack

SYSTEM

E.g., **`put()`** must be **atomic**

Application Requirements

Stack Configuration

E.g., Key-Value Store on top of disk

Library of stack-layer specifications

Result

E.g., **`put()`** is atomic on given stack

SYSTEM

E.g., **`put()`** must be **atomic**

Application Requirements

Use the proof assistant to manually write machine-checked proofs

E.g., Key-Value Store on top of disk

Library of stack-layer specifications

Result

E.g., **`put()`** is atomic on given stack

EXPERIENCE WITH ISABELLE

Modelled a simple 2-layer stack

- Key-value store on top of a disk

Proved `put()` is atomic

About 160 lines of code (lots of trial and error)

Code available at: https://github.com/ramanala/StorageStackSemantics

EXPERIENCE WITH ISABELLE

Modelled a simple 2-layer stack

Key value store on top of a disk

```
(* Atomic KV puts *)
theorem kv_put_atomic_theorem:
shows   "length d \<ge> 2 \<and> (Suc idx) < length d  \<and> (list_contains after (kv_put s
         \<Longrightarrow>
         after!idx=d!idx \<and> after!(Suc idx)=d!(Suc idx)
         \<or>
         after!idx=s1 \<and> after!(Suc idx)=s2"
apply(auto)
done
```

StorageStackSemantics

ramanala / StorageStackSemantics   Watch 2

OUTLINE

1. Introduction
2. Portability Bug Study
3. First Steps Toward The Vision
4. The Road Ahead

CHALLENGES

Obtaining specifications

- Developer provides/written by grad students
- How to figure out automatically?

Automatic proofs

- Use Z3 instead of Isabelle?

Proofs without specifications

- Know a layer provides guarantees, without knowing how

Verifying implementations

CONCLUSION

The promise of software-defined storage

- Increases in performance, flexibility, and utilization
- Unspoken aspect: **application correctness**!

Simply ensuring API compatibility is not enough

- Storage semantics are complex and nuanced

PL tools like SMT solvers/proof assistants can help match application to diverse storage stacks

Interesting, significant challenges on path ahead

THANK YOU!

QUESTIONS?





SOURCE CODE AT:
HTTP://CS.WISC.EDU/~VIJAYC

VIJAY CHIDAMBARAM
UNIVERSITY OF WISCONSIN MADISON
VIJAYC@CS.WISC.EDU | HTTP://CS.WISC.EDU/~VIJAYC