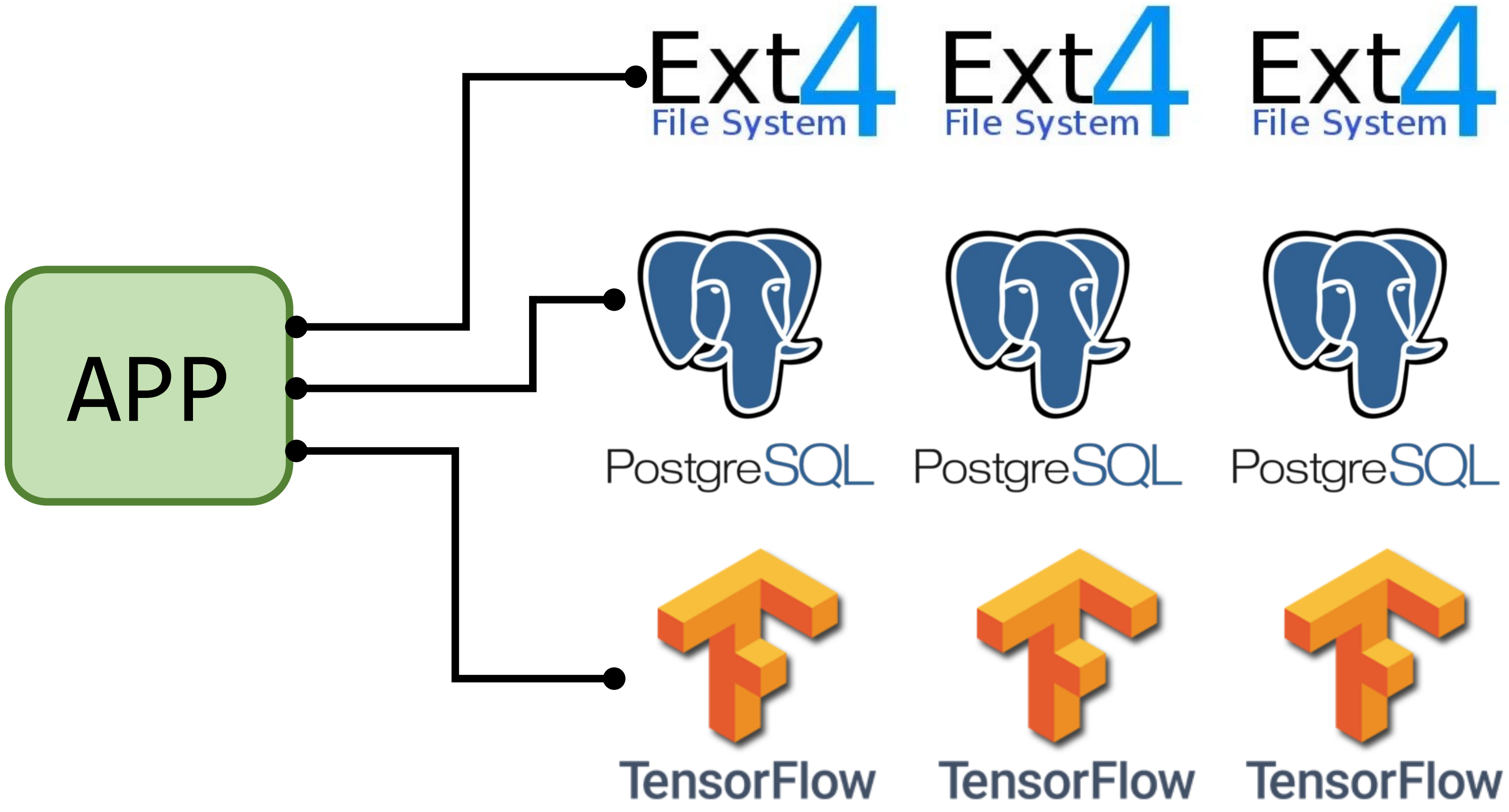
Push-Button Reliability Testing for Cloud-Backed Applications with Rainmaker

Yinfang Chen, **Xudong Sun**, Suman Nath, Ze Yang, Tianyin Xu

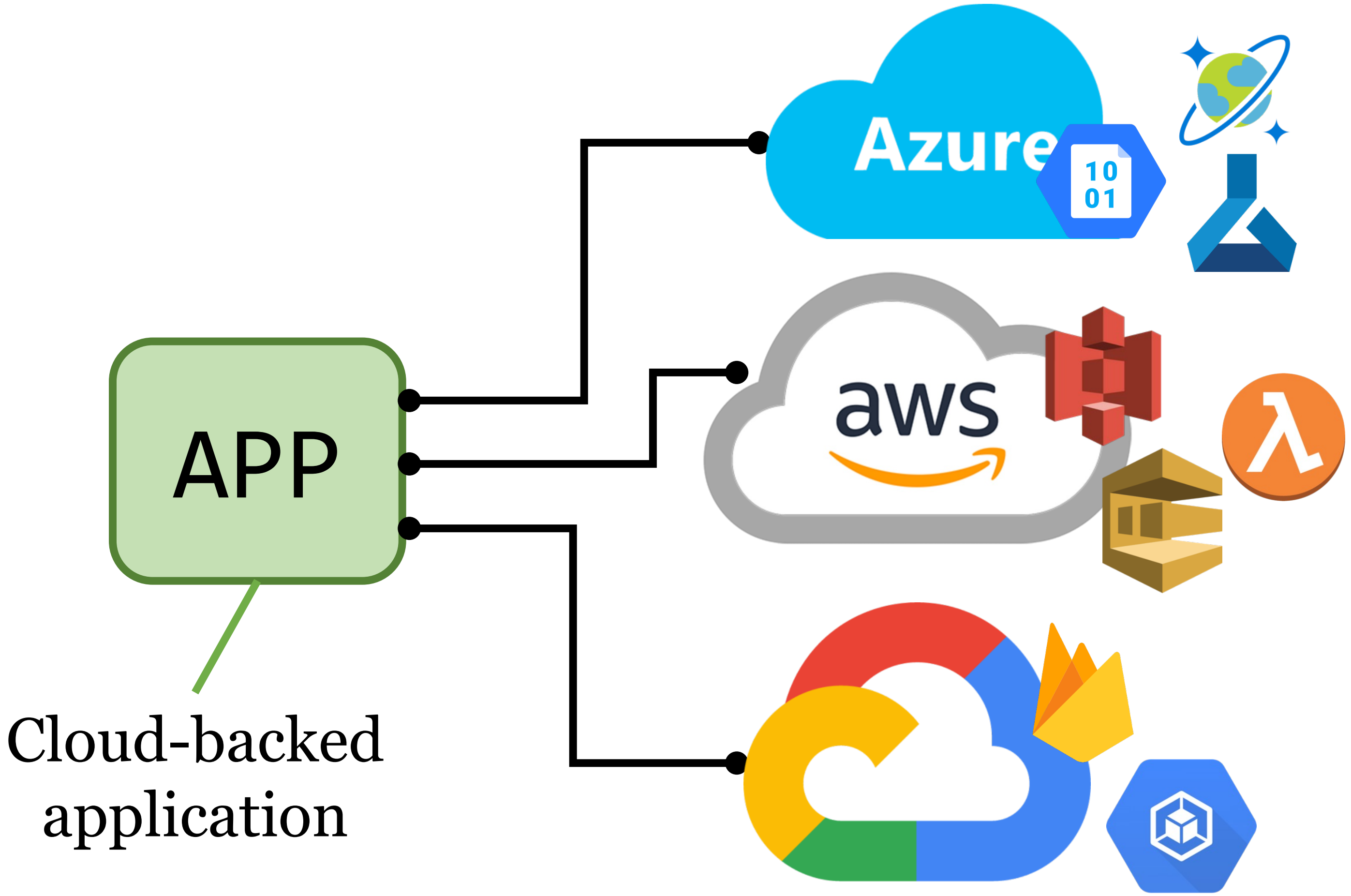
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Microsoft

The emerging cloud-based programming model



The emerging cloud-based programming model



- Azure has over **700 million** users
- Azure storage SDK (.NET) has **~80K** daily downloads

Azure.Storage.Blobs

Prefix Reserved

.NET 6.0 .NET Standard 2.0

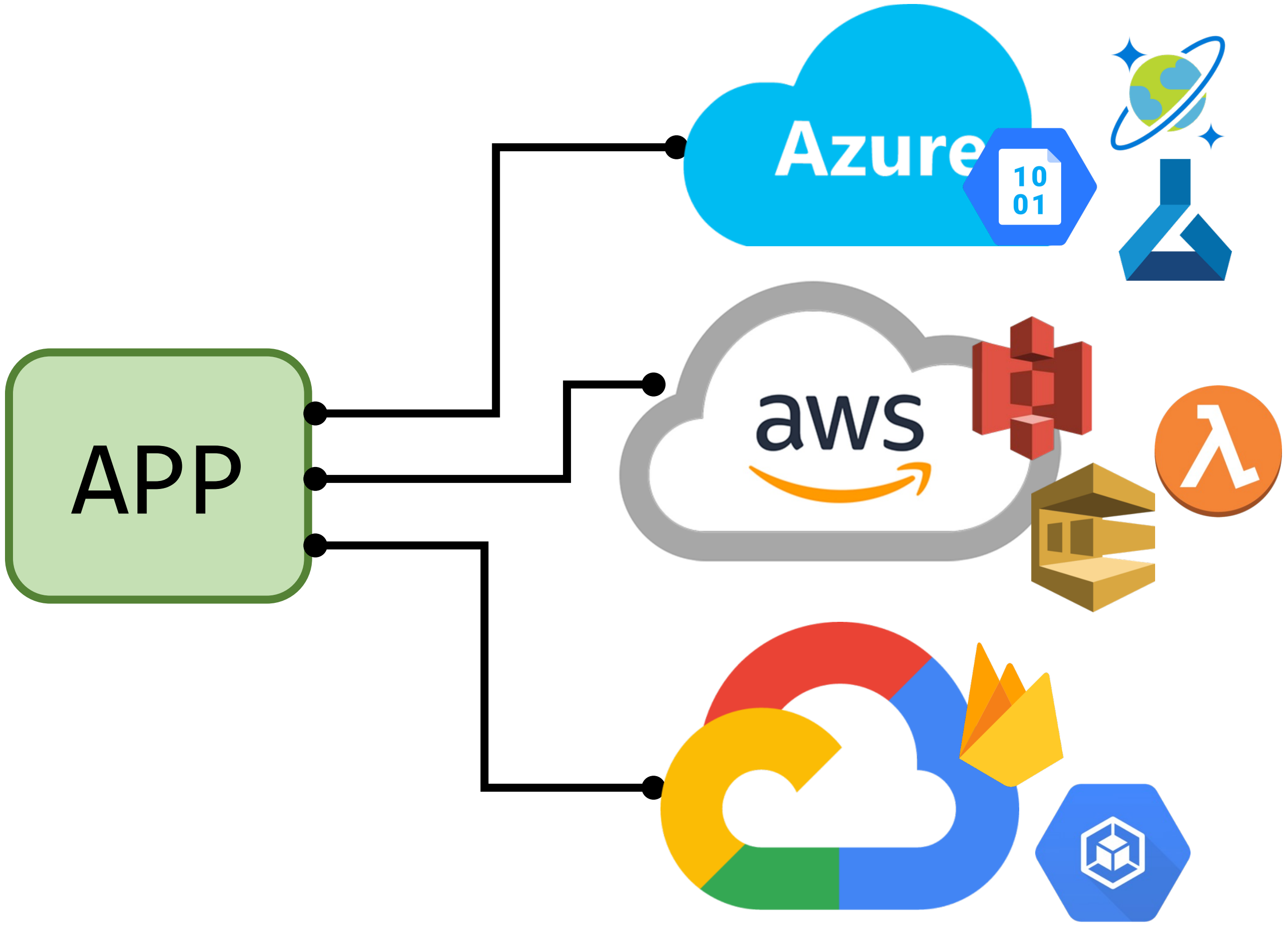
Downloads

Total **114.0M**

Current version **69.5K**

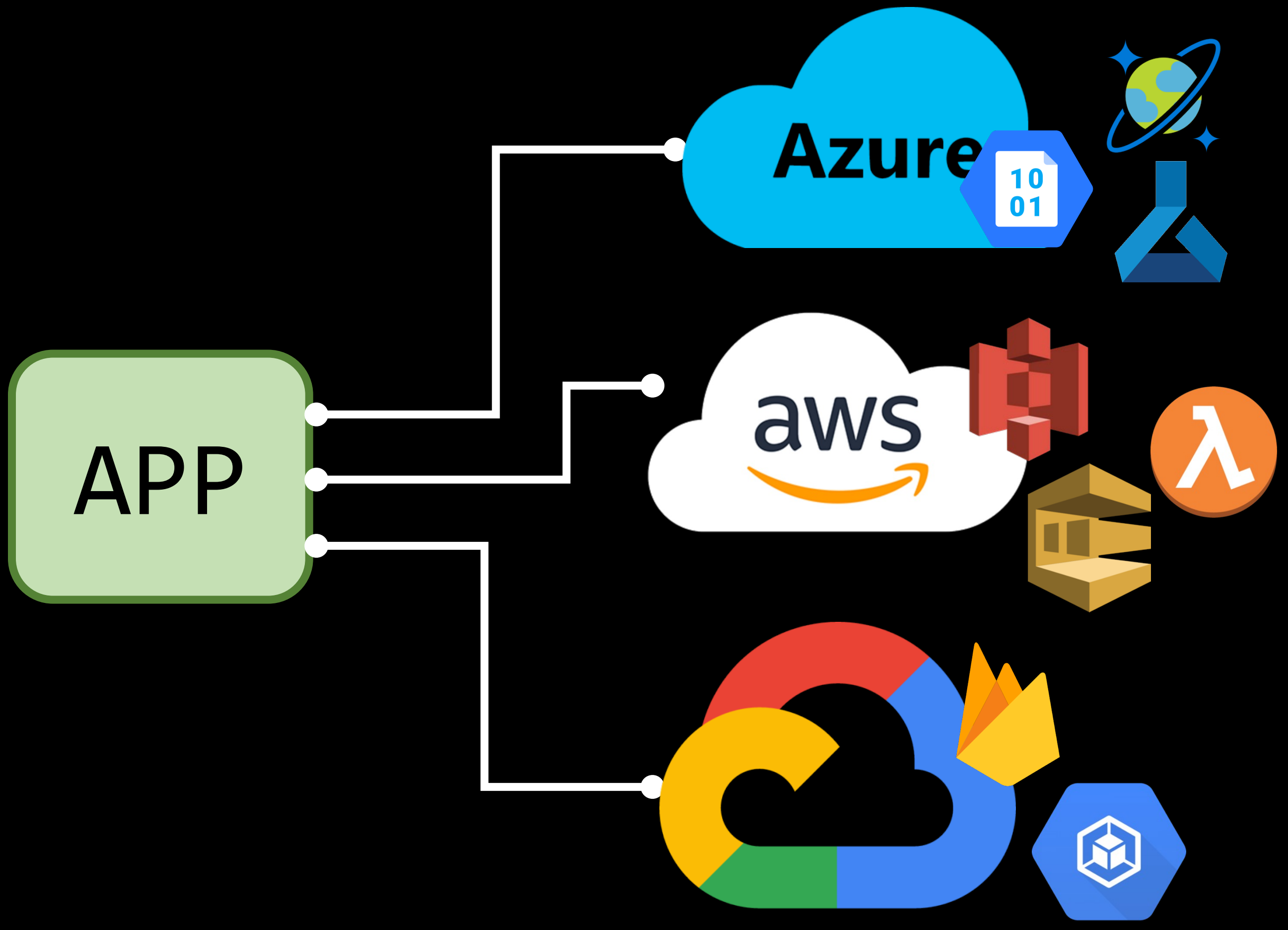
Per day average **79.7K**

Benefits of cloud-based programming



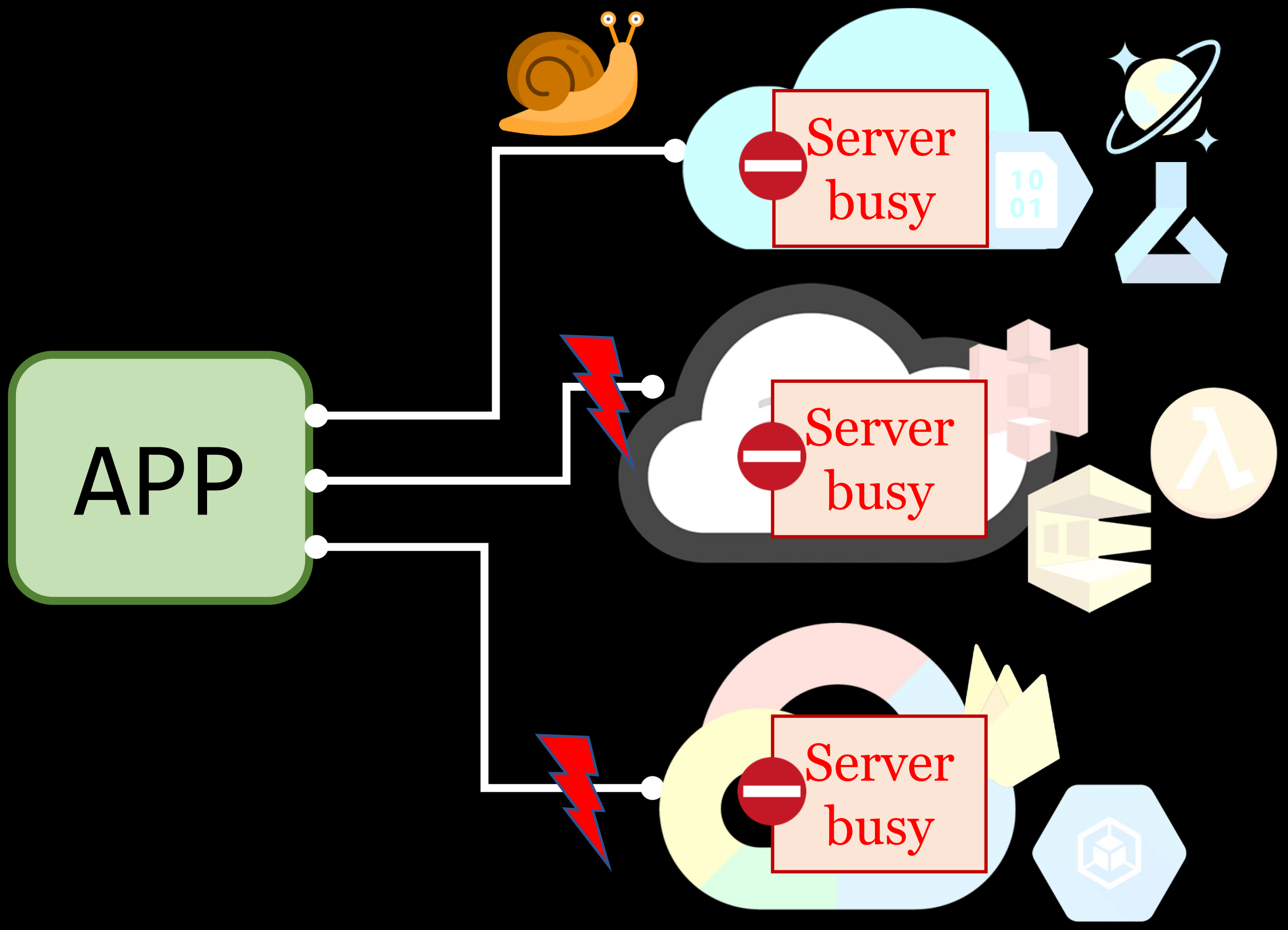
- Scalability
- Availability
- Flexibility
- Easy deployment

Dark side: new reliability challenges



- Diverse fault domains

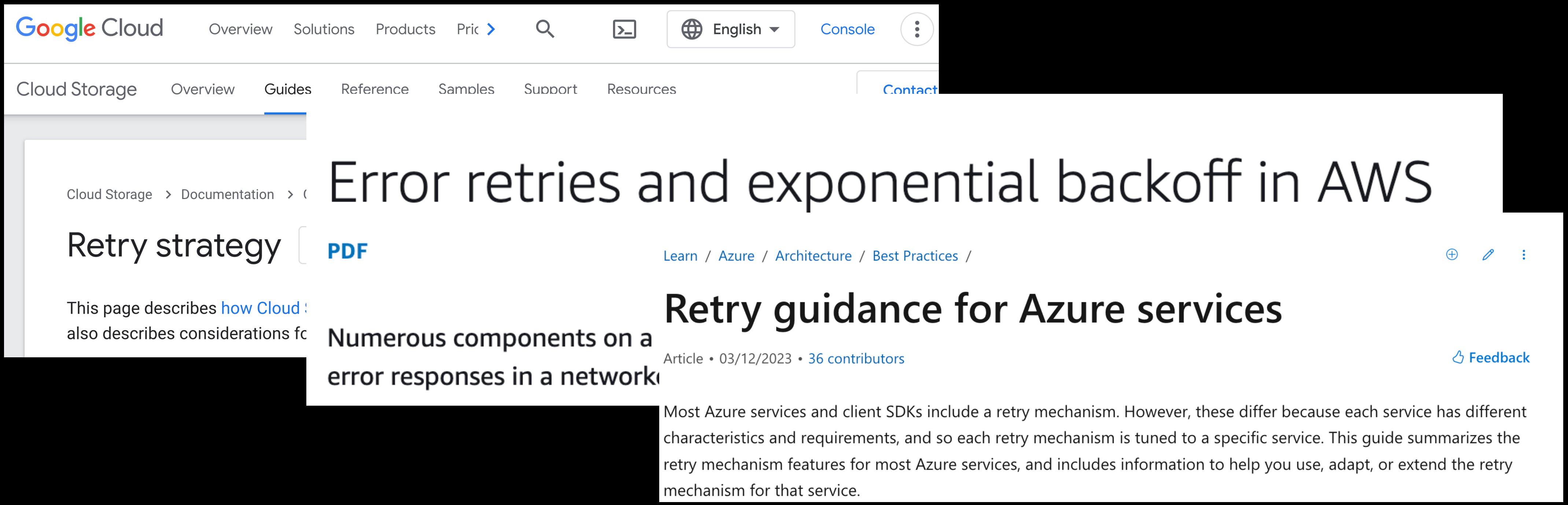
Dark side: new reliability challenges



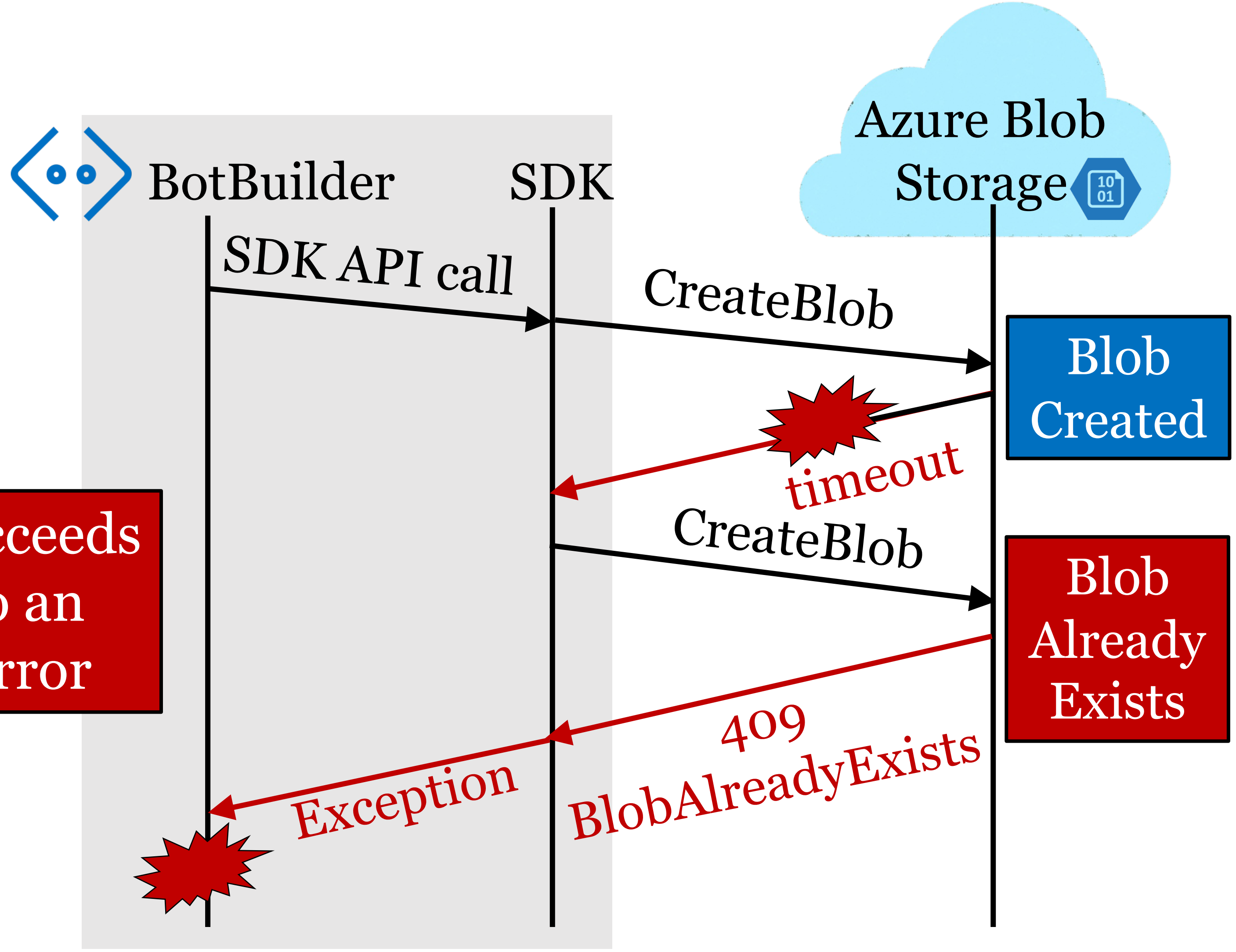
- Diverse fault domains
- A lack of standards
  - No standards such as POSIX
- Inconsistencies
  - E.g., AWS S3 SDKs in different languages treat “limit exceeded” error differently

Dark side: new reliability challenges

It is challenging for application developers to anticipate all faulty scenarios and write comprehensive error-handling code

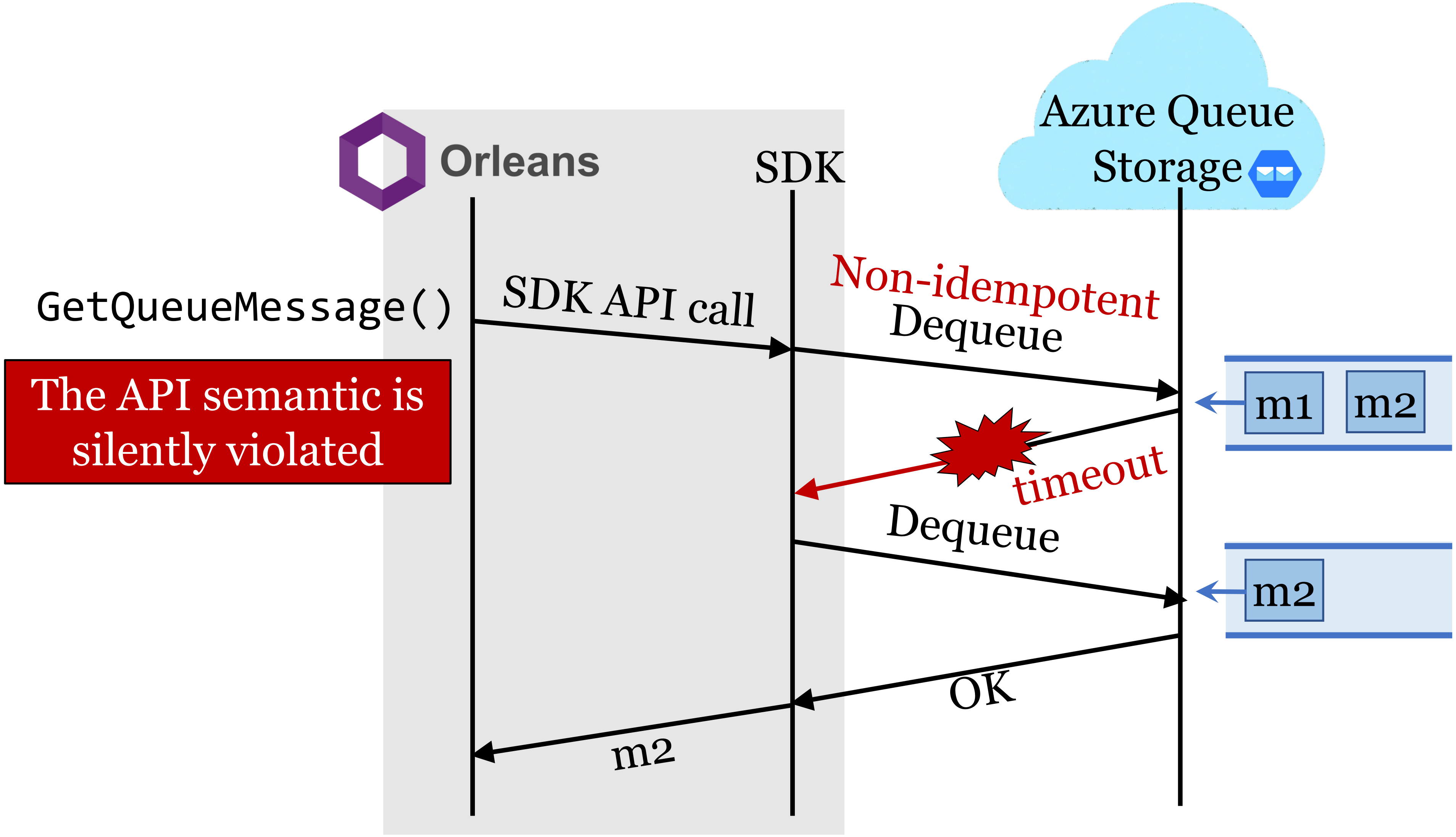


Does retry solve all the problems?



The request succeeds but it leads to an unexpected error

Does retry solve all the problems?



How can applications address the emerging reliability challenges of cloud-based programming?

Contribution

- A call for attention of the emerging reliability challenges of cloud based programming
- A taxonomy of error-handling bugs triggered by transient faults
- Rainmaker: Push-button reliability testing for cloud-backed apps
  - Systematically exercise error-handling code under common faults
  - Detected **73** new bugs in 11 cloud-backed apps (**51** fixed)
  - Released at https://github.com/xlab-uiuc/rainmaker

GitHub

Design goals of Rainmaker

- **Effective**: Detect error-handling bugs of different patterns
- **Easy-to-use**: Directly applied to existing testing environment
- **Efficient**: Efficiently finish testing while ensuring coverage

Design goals of Rainmaker

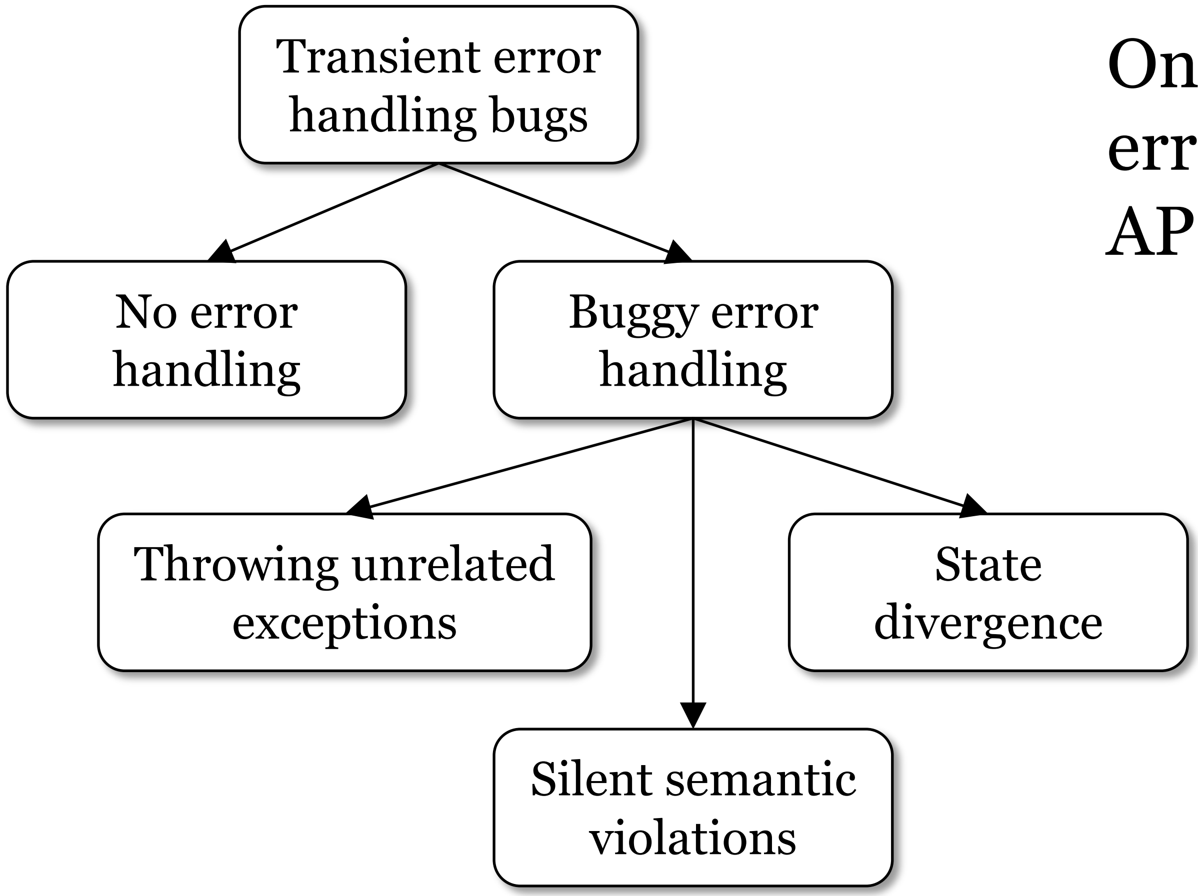
- **Effective**: Detect error-handling bugs of different patterns

Fault injection during testing, before production

- **Efficient**: Efficiently finish testing while ensuring coverage

What faults to inject? ***When*** to inject them?

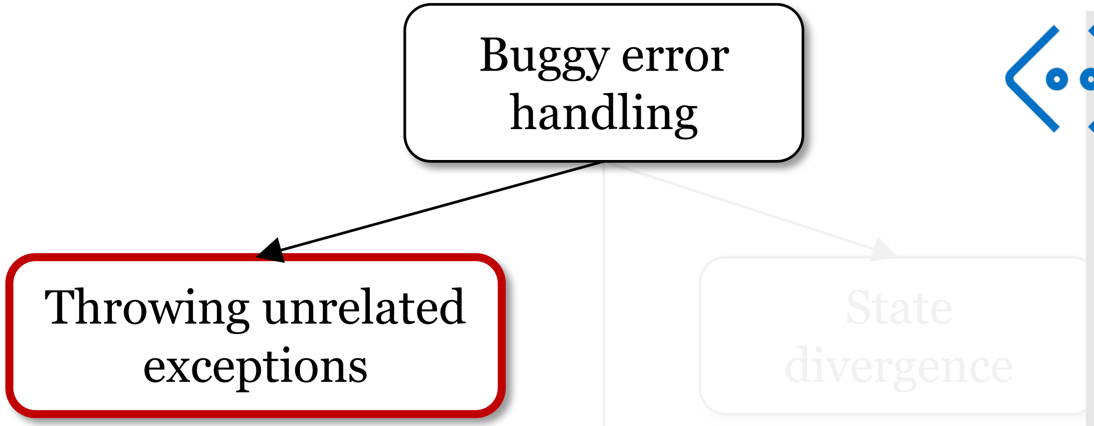
A taxonomy of error-handling bugs



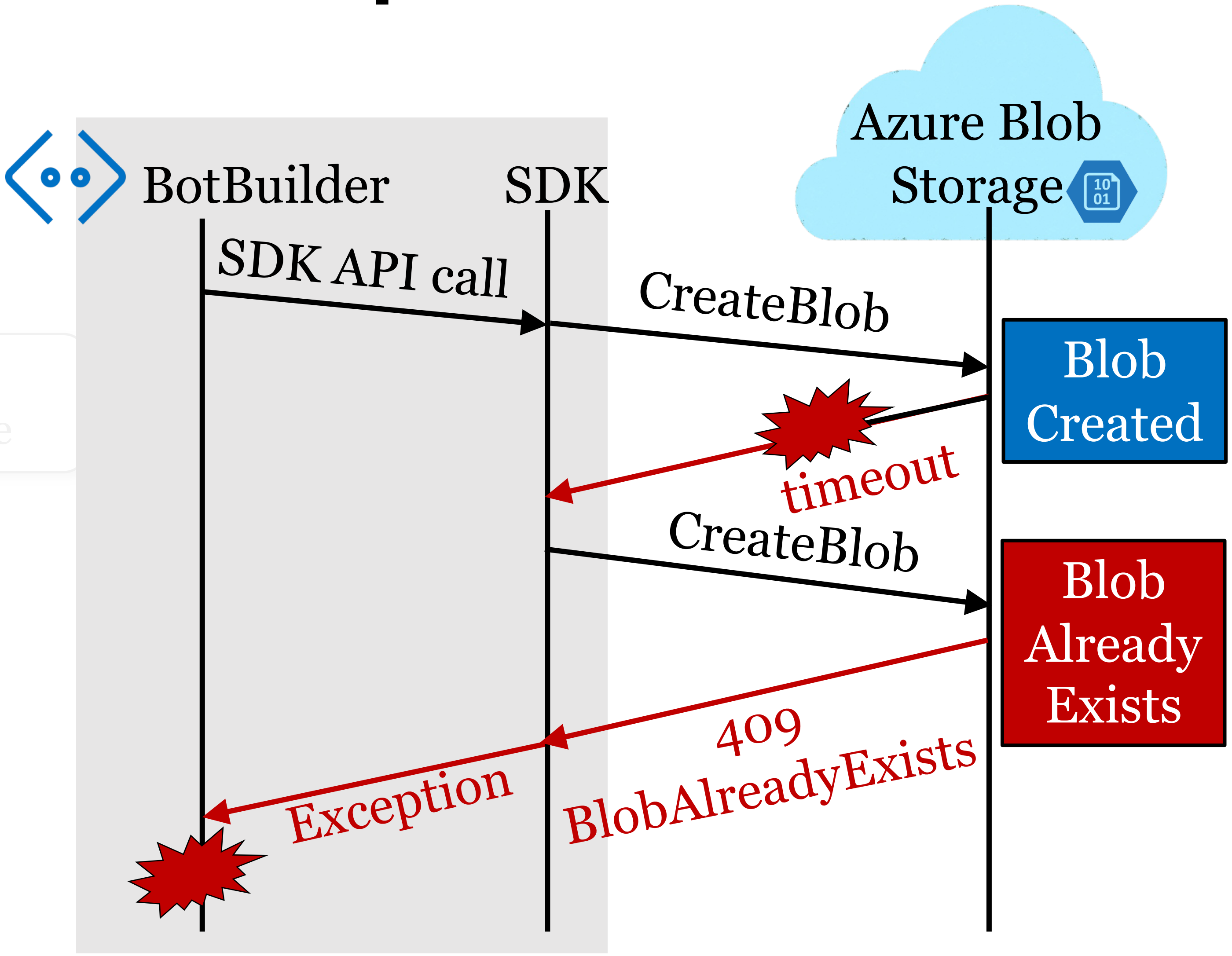
Only consider **realistic transient** error(s) that occur during **one** REST API call interaction

- Timeout
- Server-busy error

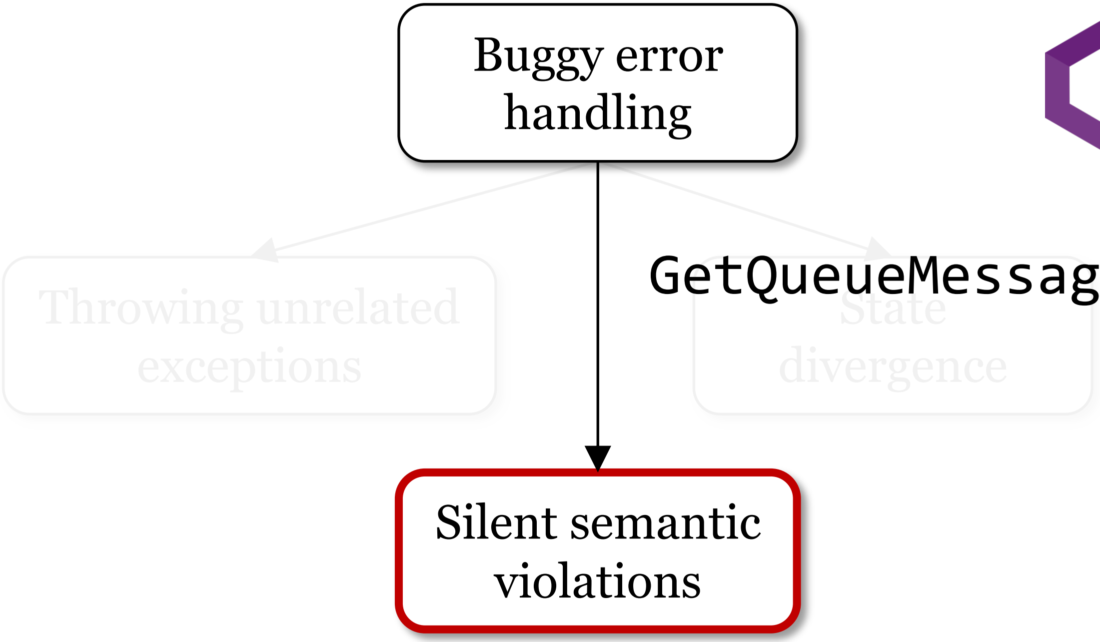
Throwing unrelated exceptions

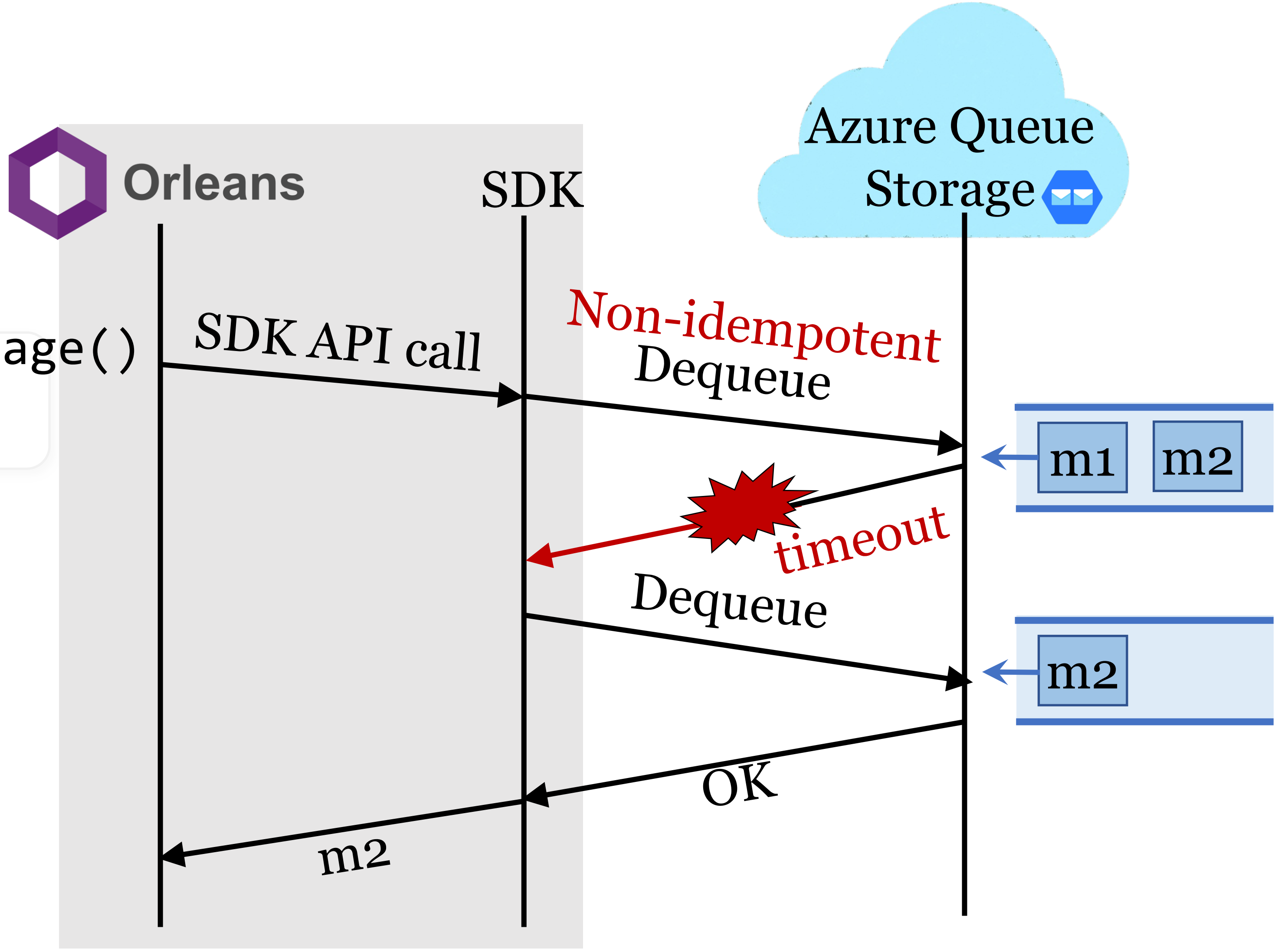


Key: Mishandling leads to a new error unrelated to the root cause error



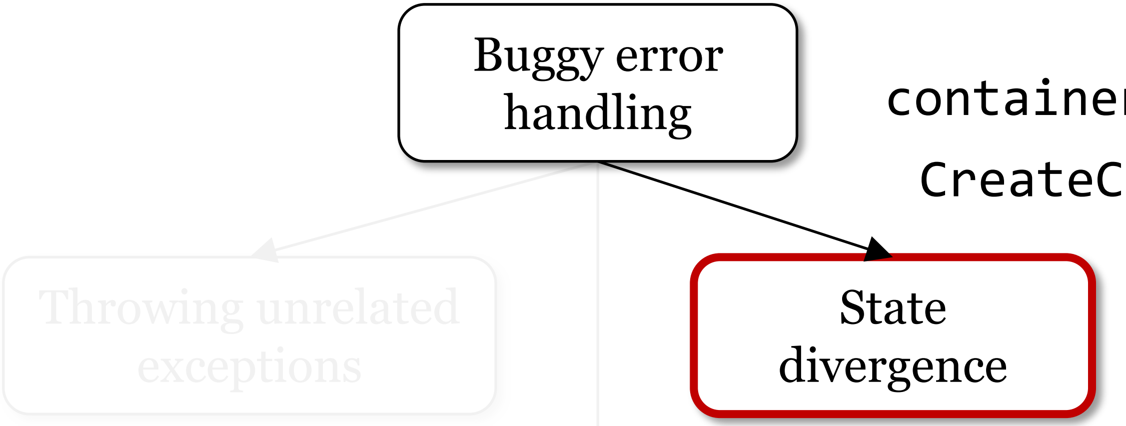
Silent semantic violations



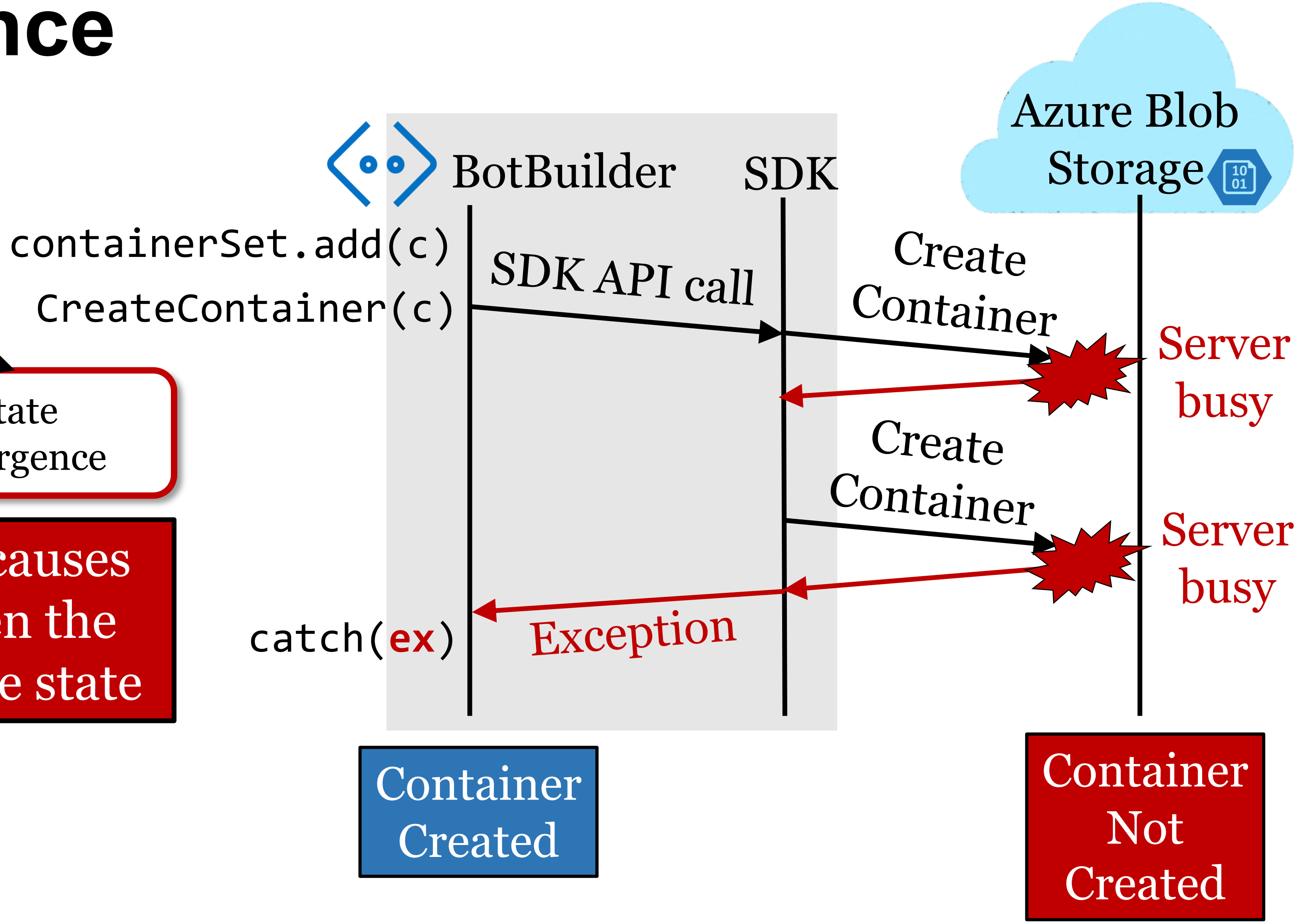


Key: Mishandling causes semantic violations of the application

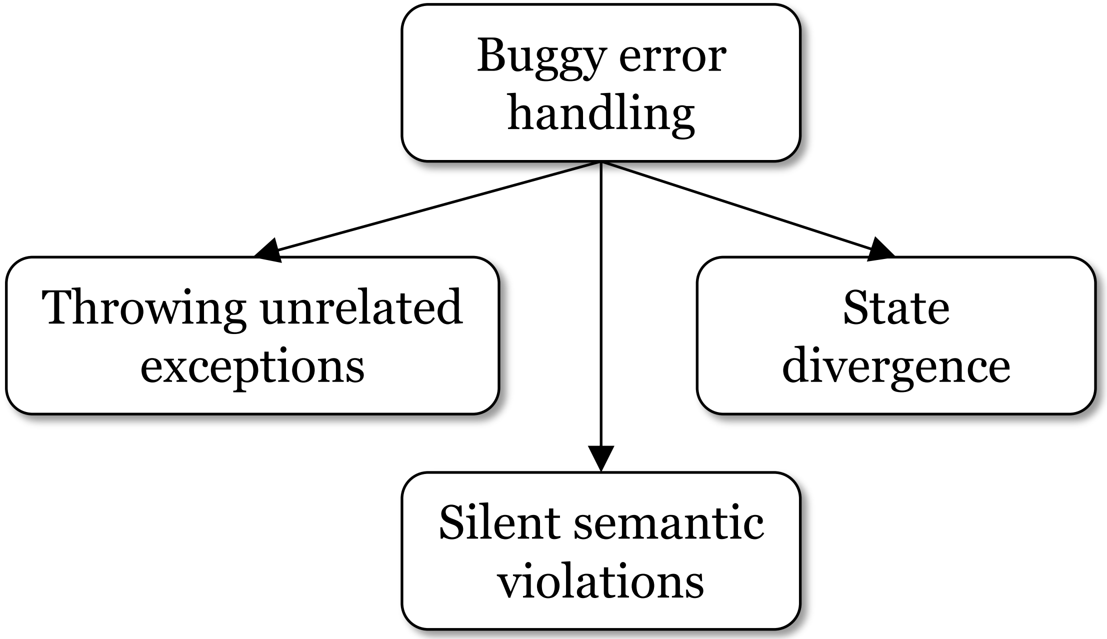
State divergence



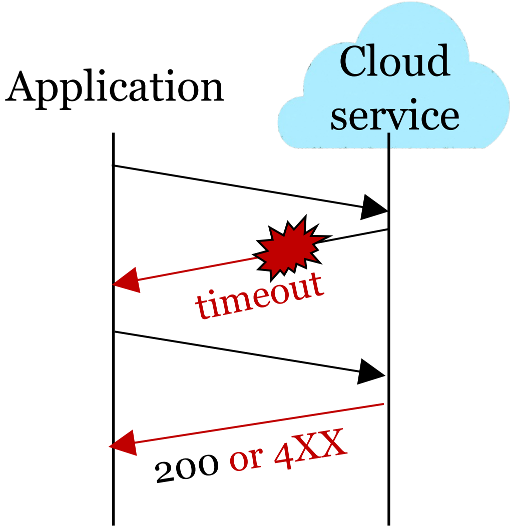
Key: Mishandling causes divergence between the local and the remote state



Rainmaker’s fault injection policies



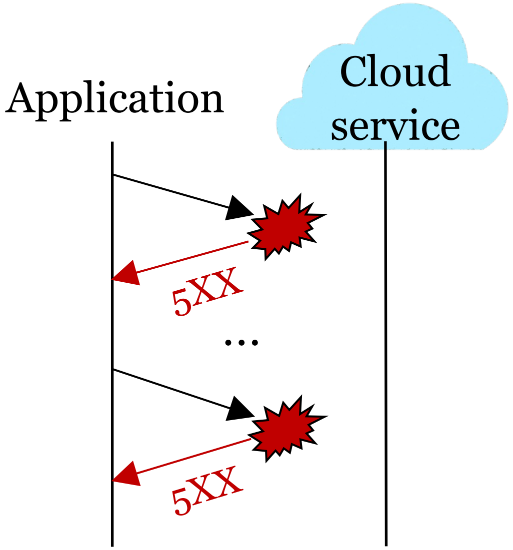
Rainmaker has more policies to trigger bugs



P_1: Timeout the first response

Throwing unrelated exceptions

Silent semantic violations



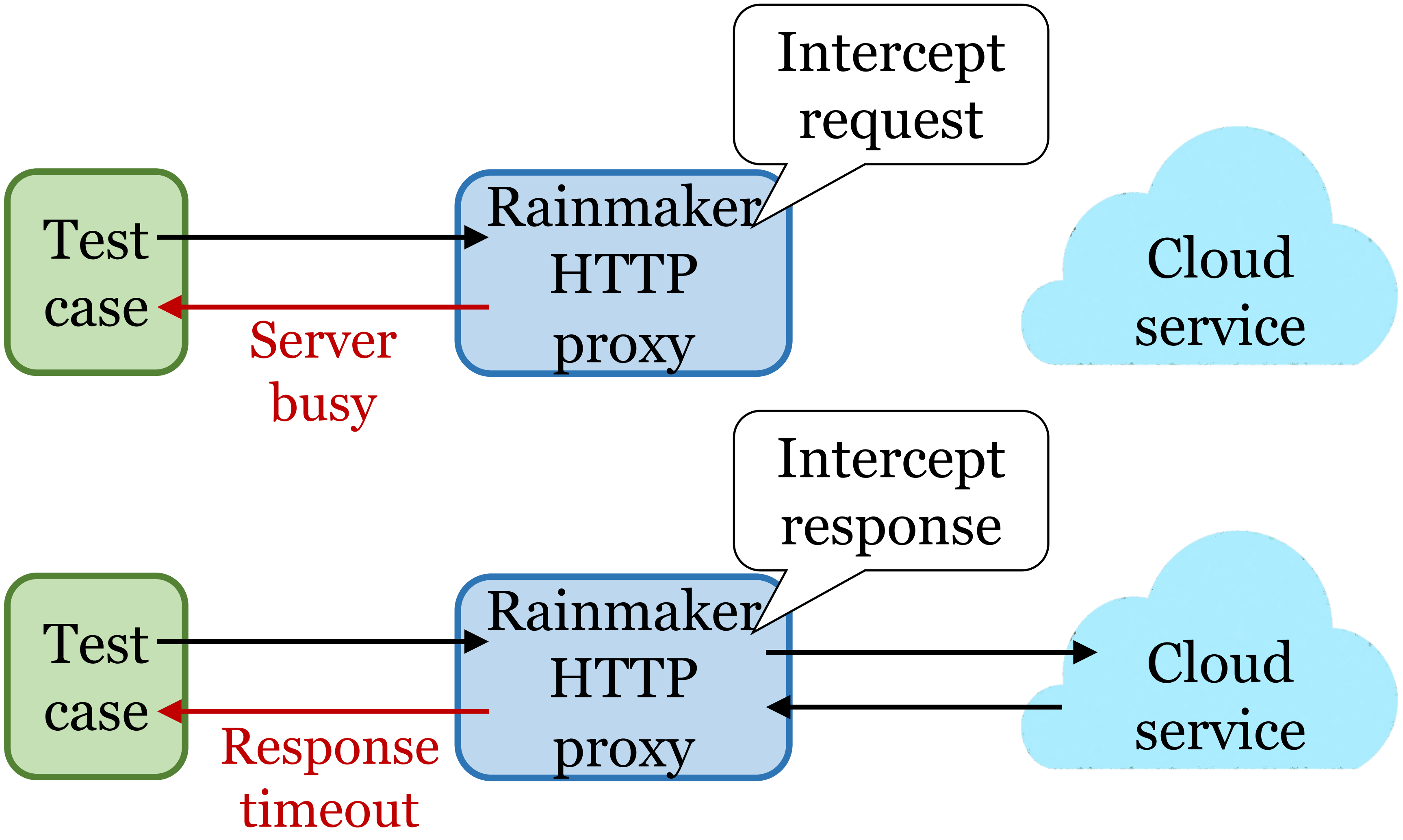
P_2: Return 5XX to all requests

State divergence

Design goals of Rainmaker

- **Effective**: Detect error-handling bugs of different patterns
- **Easy-to-use**: Directly applied to existing testing environment
- **Efficient**: Efficiently finish testing while ensuring coverage

Rainmaker performs HTTP layer injection



Rainmaker reuses existing test oracles

- Naively reusing oracles could lead to false alarms
- Analyze test execution and output to capture only true alarms

```
// test code
fn unit_test() {
  // set up set env
  SDK.CreateBlob();
  ...
  // call app code
  ...
}
```

The test failure does not point to any error-handling bug in application code.

Solution: Rainmaker checks the stack trace of the exception. If the SDK is directly invoked by test code, it does not report an alarm.

Design goals of Rainmaker

- **Effective**: Detect error-handling bugs of different patterns
- **Easy-to-use**: Directly applied to existing testing environment
- **Efficient**: Efficiently finish testing while ensuring coverage

Ensure coverage while being efficient

- Test 1
  - SDK.API1()
  - SDK.API2()
    - REST call by 1^{st} invocation
    - REST call by 2^{nd} invocation
    - REST call by 3^{rd} invocation
    - …
  - …
- Test 2
  - SDK.API2()
  - SDK.API3()
  - …
- …
- Test n
  - SDK.API1()
  - SDK.API3()
  - …

Injecting to every REST call takes **588 days** to test Orleans!

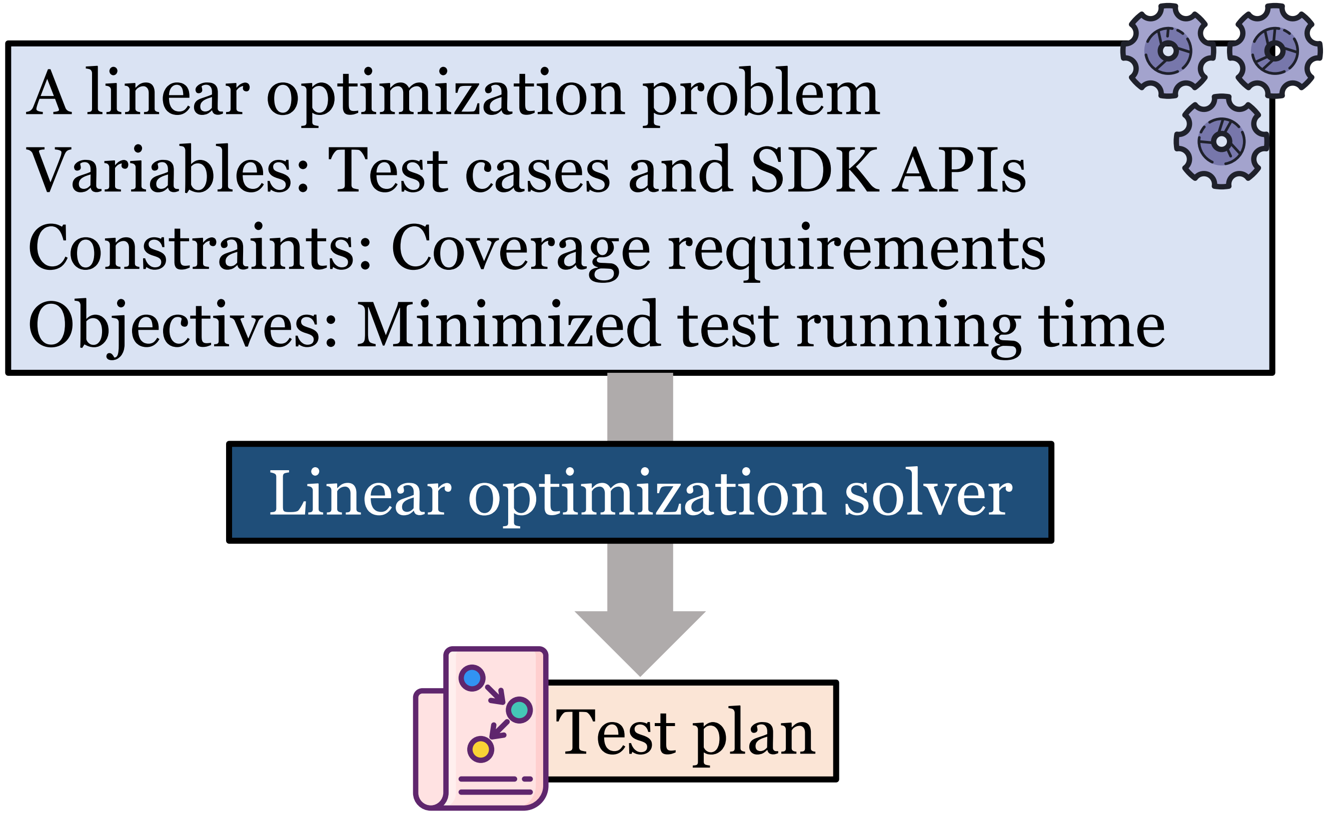
↓

~56 hours

Coverage Metric	
Cover all combinations of (test case, SDK API*)	Default
Cover all test cases and SDK API respectively	Weaker Faster
Cover all SDK API	
Cover all test cases	

Generating test plans

- Rainmaker generates the test plan that achieves the coverage with minimized test running time for each coverage metric



Evaluation

- We applied Rainmaker to 11 popular cloud-backed applications
- Rainmaker finds **73** new bugs with severe consequences
- Rainmaker has a low false-positive rate **1.96%**
- Rainmaker reduces on average **64.47%** of test runs compared to exhaustively injecting to every REST call

Finding *new* bugs

55 confirmed; 51 fixed

Azure Storage

Cosmos DB

AWS S3

AWS SQS

Cloud-backed application	No Error Handling	Unrelated Exception	Semantic Violation	State Divergence	Total
Alpakka	0	0	1	1	2
AttachmentPlugin	0	0	0	2	2
BotBuilder	0	2	0	2	4
DistributedLock	0	2	0	0	2
EF Core	7	0	0	0	7
FHIR Server	11	0	0	0	11
Insights	0	10	0	0	10
IronPigeon	0	1	0	0	1
Orleans	0	5	2	11	18
Sleet	0	2	0	0	2
Storage.NET	11	1	1	1	14
Total	**29**	**23**	**4**	**17**	**73**

Conclusion

- A call for attention of the emerging reliability challenges of cloud based programming
- A taxonomy of error-handling bugs triggered by transient faults
- Rainmaker: Push-button reliability testing for cloud-backed apps
  - Effective, easy-to-use, efficient
  - Released at https://github.com/xlab-uiuc/rainmaker