中山大學
SUN YAT-SEN UNIVERSITY

ByteDance
字节跳动

ErrorPrism: Reconstructing Error Propagation Paths in Cloud Service Systems

Junsong Pu[1], Yichen Li[3], Zhuangbin Chen✉[1], Jinyang Liu[3],

Zhihan Jiang[2], Jianjun Chen[3], Rui Shi[3], Zibin Zheng[1], Tieying Zhang[3]

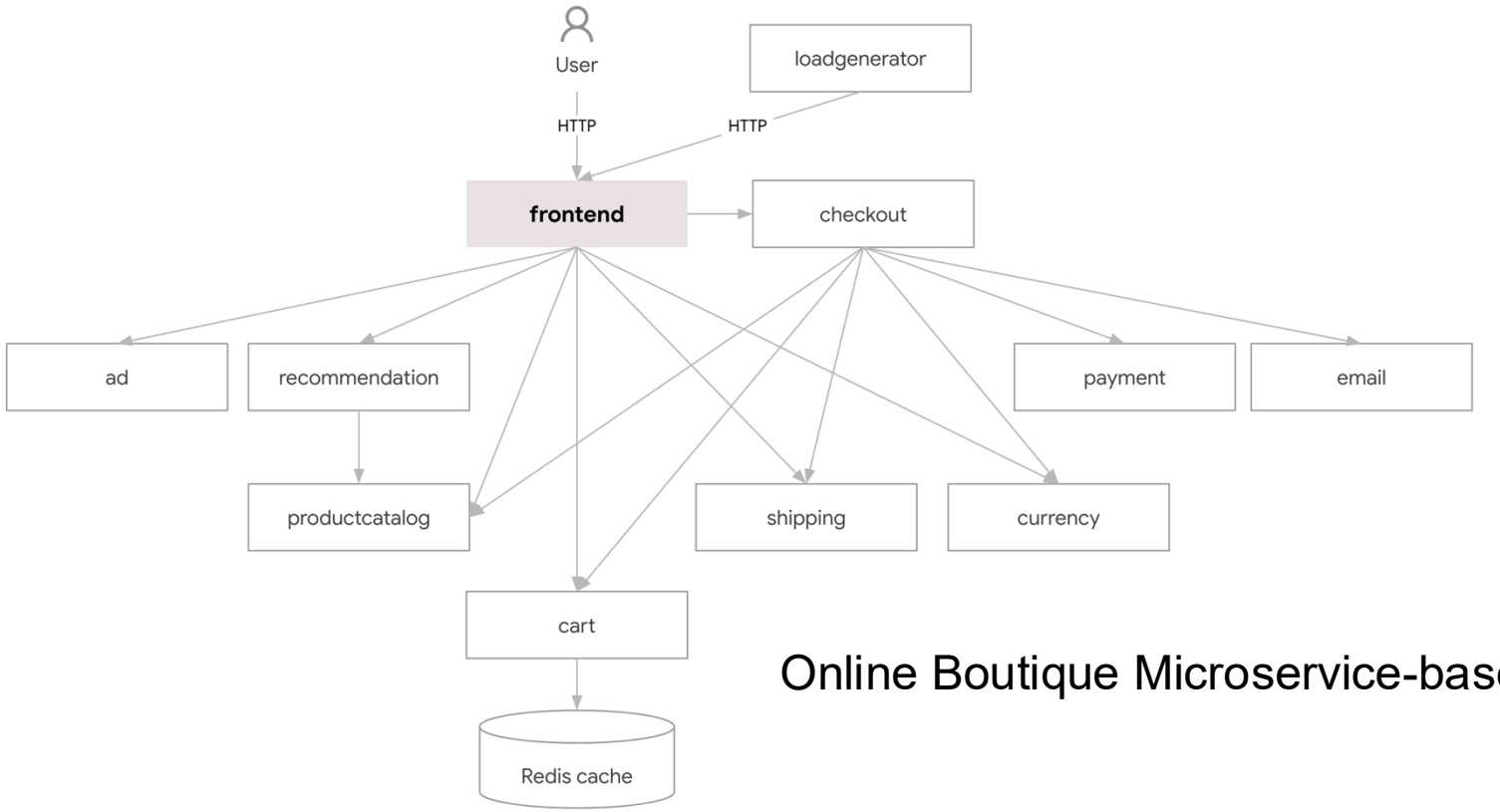
[1]Sun Yat-sen University

[2]The Chinese University of Hong Kong

[3]ByteDance

Cascading Failures in Cloud Systems

- Modern cloud systems are built on **Microservices**
- This architecture is scalable, but failures are complex and interconnected
- How are these cascading errors handled and reported at the code level?

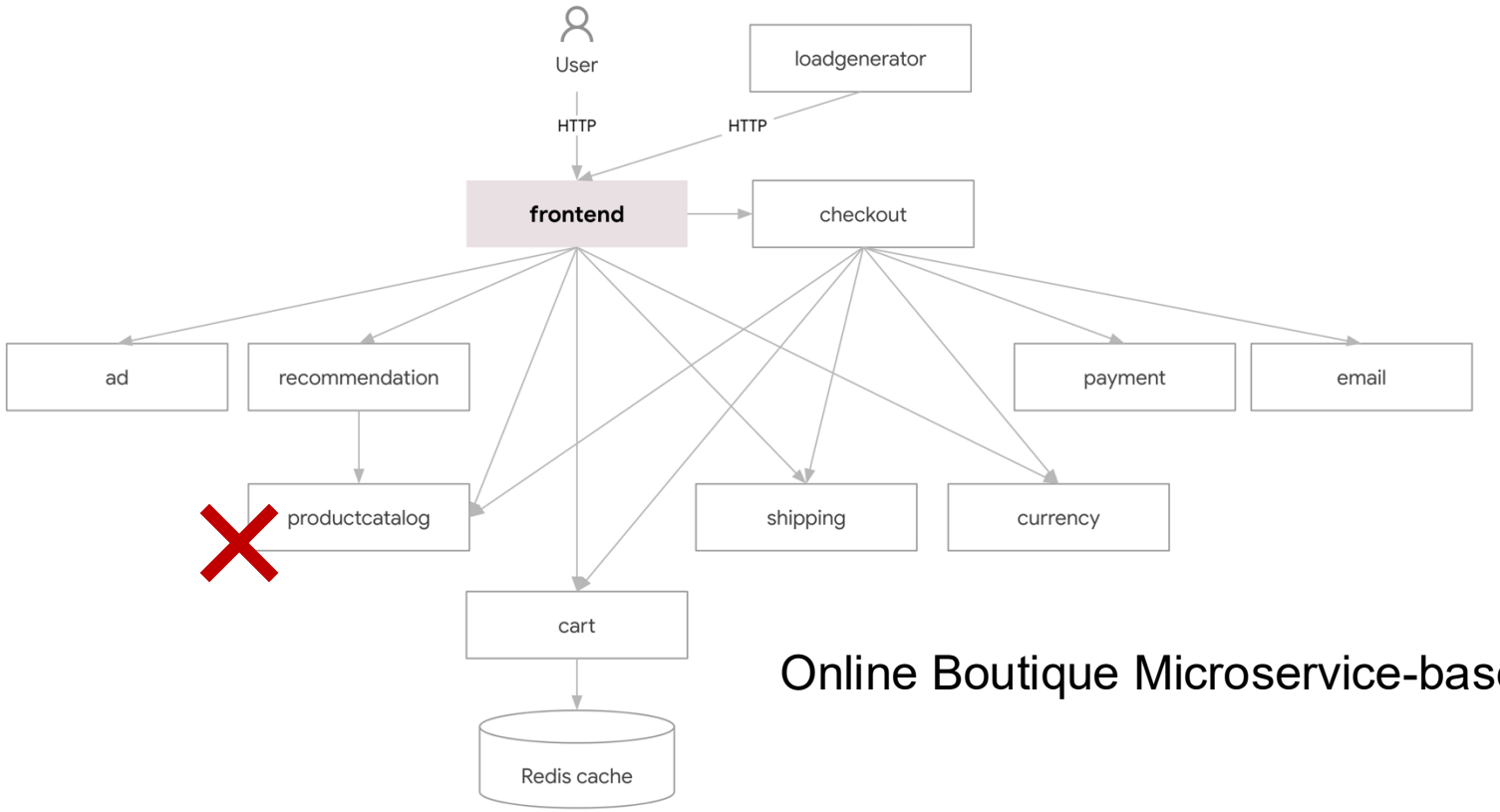


Online Boutique Microservice-based Application

Figure source: https://github.com/GoogleCloudPlatform/microservices-demo

Cascading Failures in Cloud Systems

- Modern cloud systems are built on **Microservices**
- This architecture is scalable, but failures are complex and interconnected
- How are these cascading errors handled and reported at the code level?

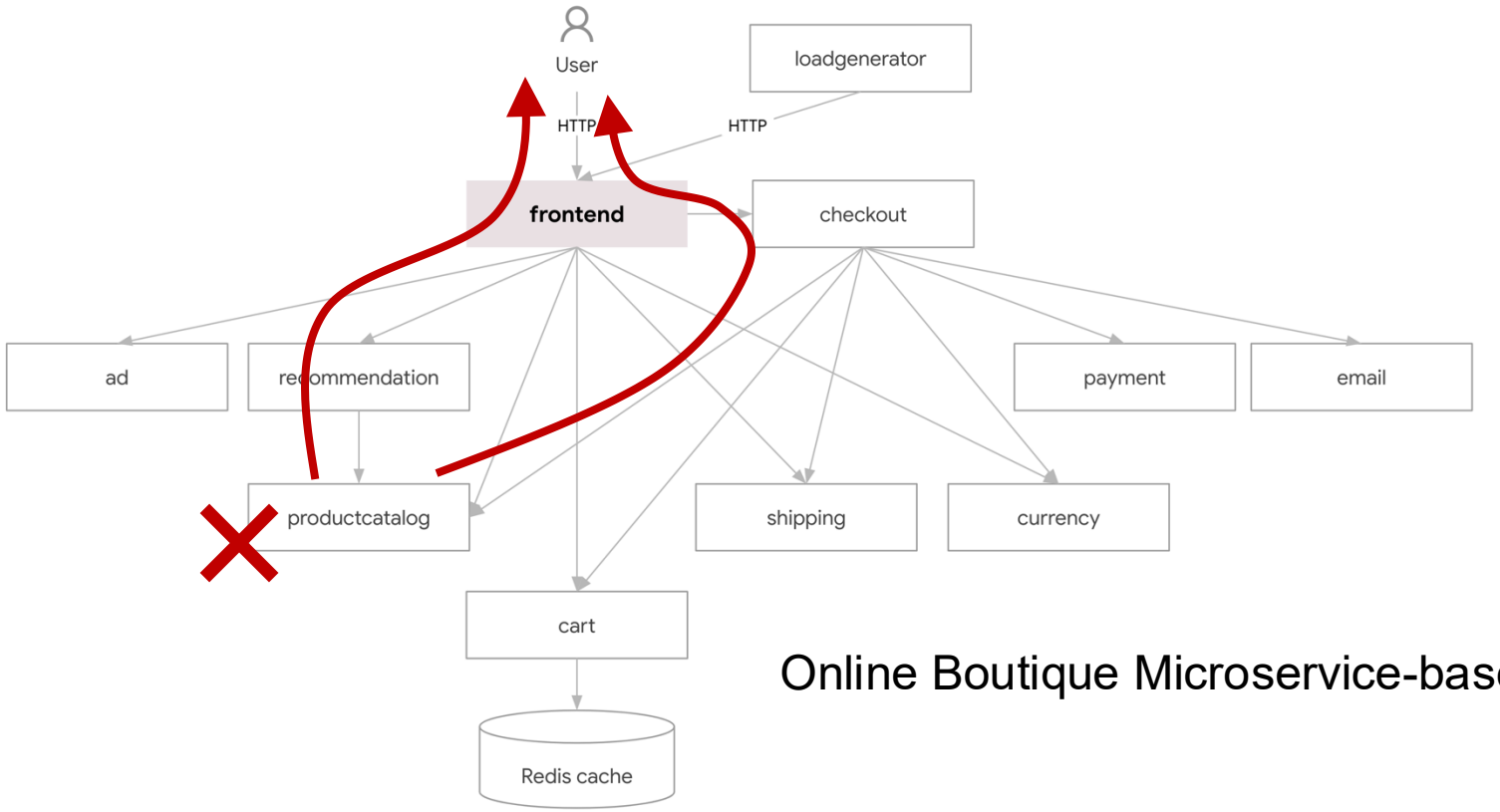


Online Boutique Microservice-based Application

Figure source: https://github.com/GoogleCloudPlatform/microservices-demo

Cascading Failures in Cloud Systems

- Modern cloud systems are built on **Microservices**
- This architecture is scalable, but failures are complex and interconnected
- How are these cascading errors handled and reported at the code level?



Online Boutique Microservice-based Application

Figure source: https://github.com/GoogleCloudPlatform/microservices-demo

Error Wrapping Mechanism

- Modern languages (*e.g.* **Go** and **Rust**) treat errors as **explicit return values**, instead of exceptions as in Java and Python
- **Error Wrapping** mechanism allows functions to add context to *any* error

```
func ReadRemoteFile() (string, error) {
    data, err := remote.ReadFile("config.txt")
    if err != nil {
        return Err: "file read error: %w", err
    }
    if len(data) == 0 {
        return Err: "parsing: empty file"
    }
    return string(data), nil
}
```

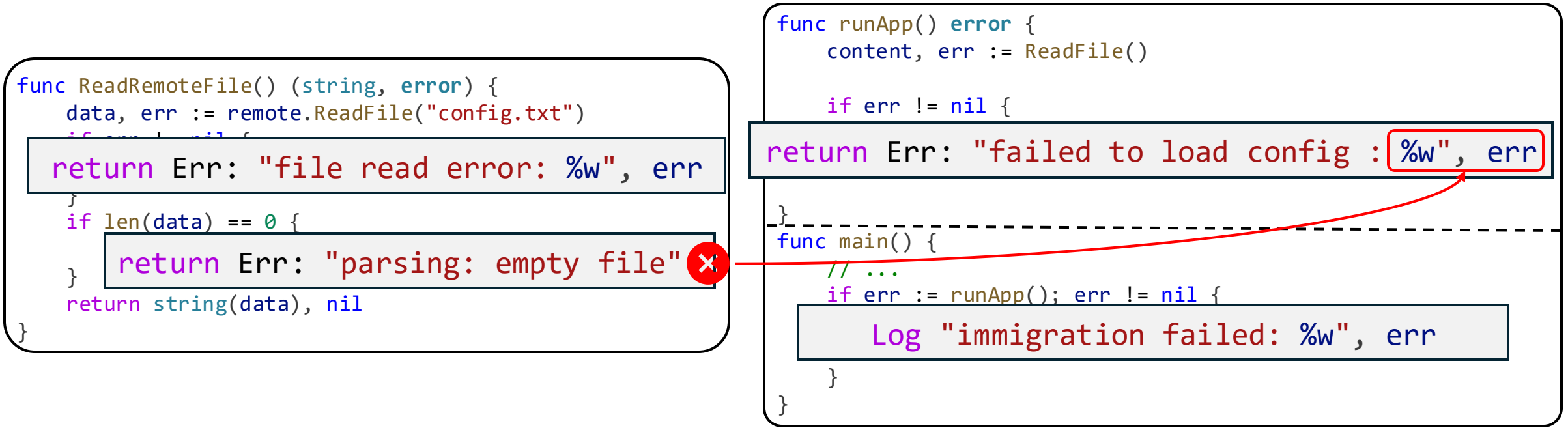
```
func runApp() error {
    content, err := ReadFile()

    if err != nil {
        return Err: "failed to load config : %w", err
    }
}
```

```
func main() {
    // ...
    if err := runApp(); err != nil {
        Log "immigration failed: %w", err
    }
}
```

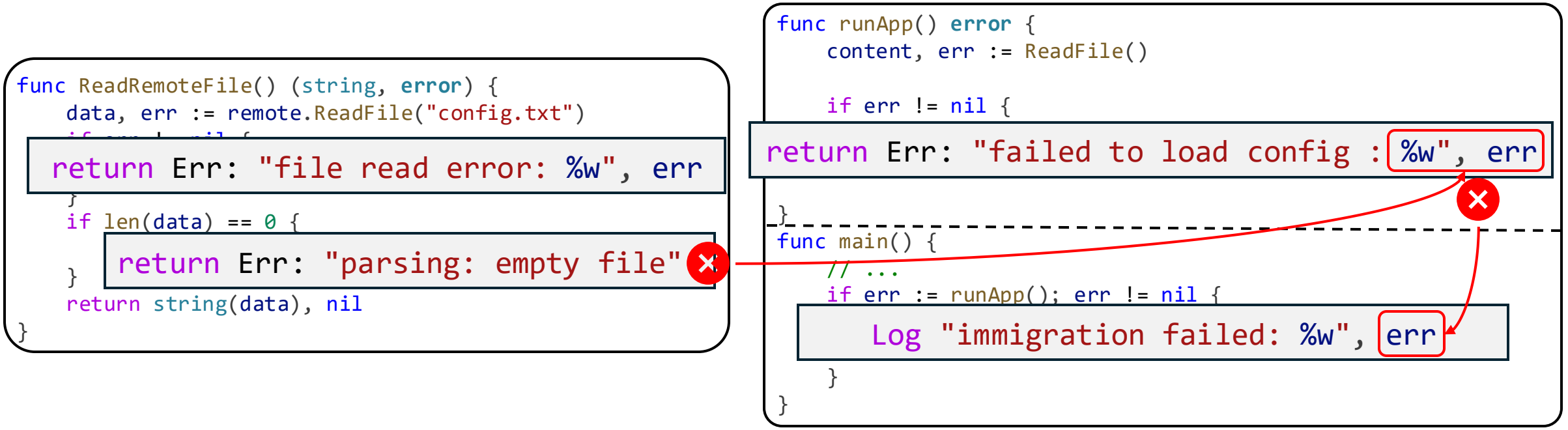
Error Wrapping Mechanism

- Modern languages (*e.g.* **Go** and **Rust**) treat errors as **explicit return values**, instead of exceptions as in Java and Python
- **Error Wrapping** mechanism allows functions to add context to *any* error



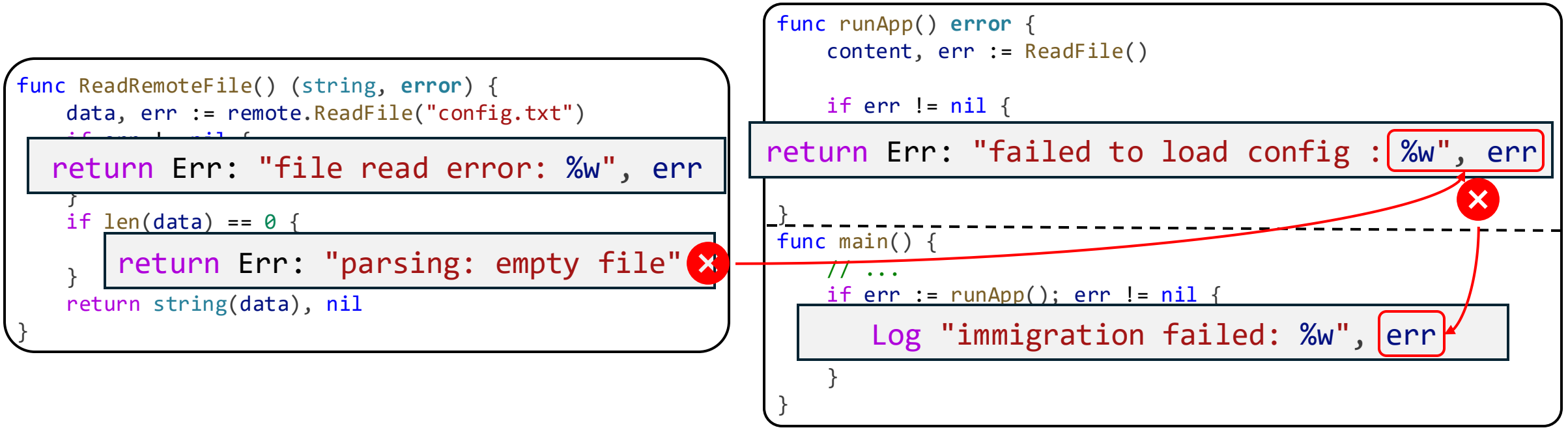
Error Wrapping Mechanism

- Modern languages (*e.g.* **Go** and **Rust**) treat errors as **explicit return values**, instead of exceptions as in Java and Python
- **Error Wrapping** mechanism allows functions to add context to *any* error



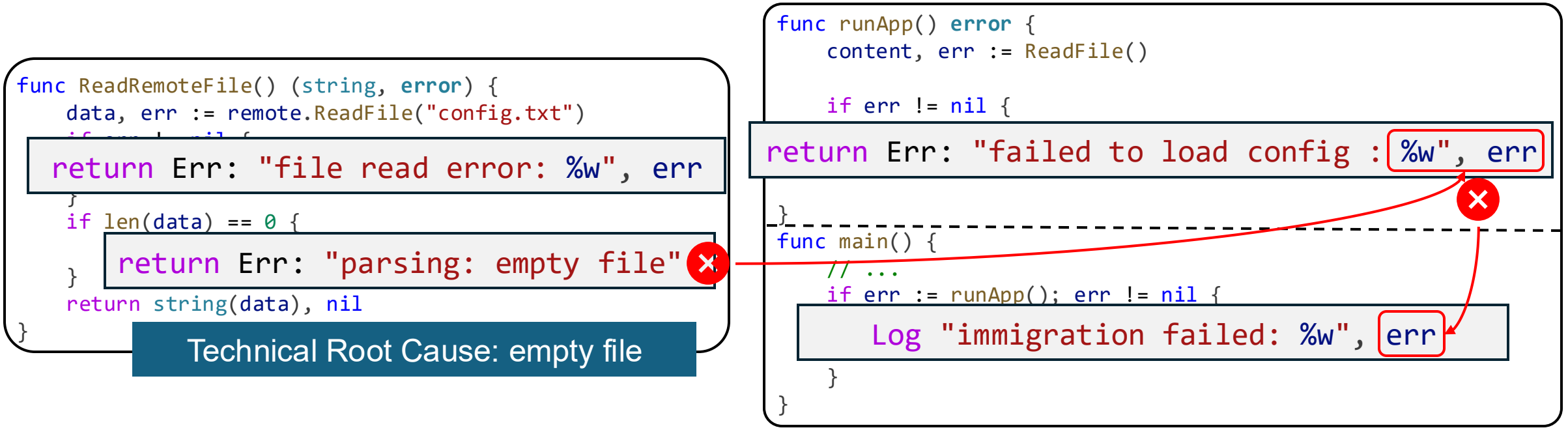
Error Wrapping Mechanism

- Modern languages (*e.g.* **Go** and **Rust**) treat errors as **explicit return values**, instead of exceptions as in Java and Python
- **Error Wrapping** mechanism allows functions to add context to *any* error
- The error propagation chain precisely describes a failure with structured context from its technical origin to its business impact



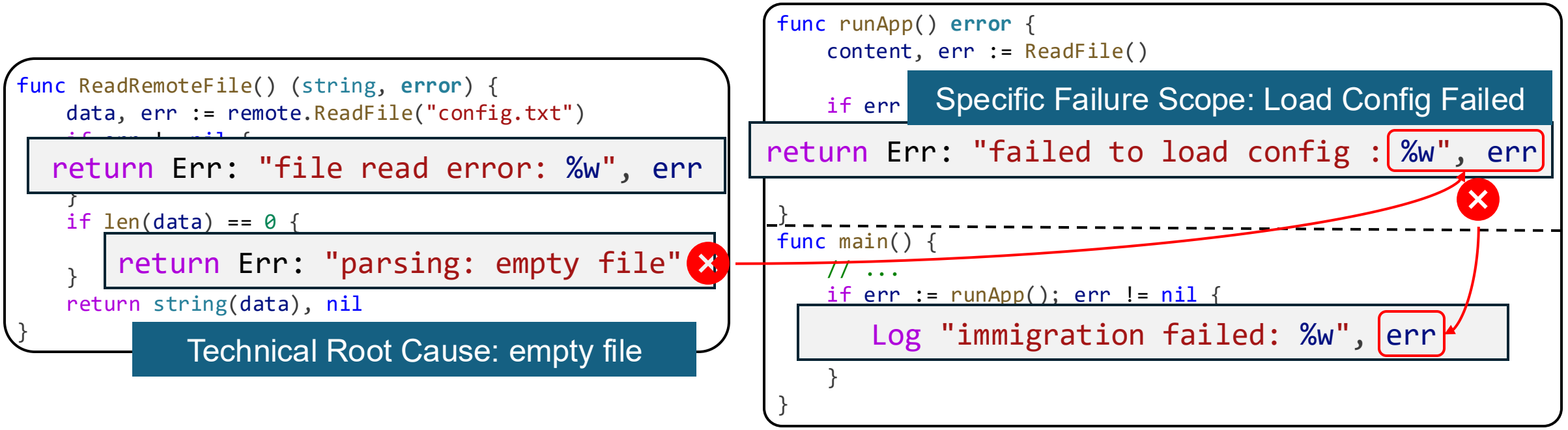
Error Wrapping Mechanism

- Modern languages (*e.g.* **Go** and **Rust**) treat errors as **explicit return values**, instead of exceptions as in Java and Python
- **Error Wrapping** mechanism allows functions to add context to *any* error
- The error propagation chain precisely describes a failure with structured context from its technical origin to its business impact



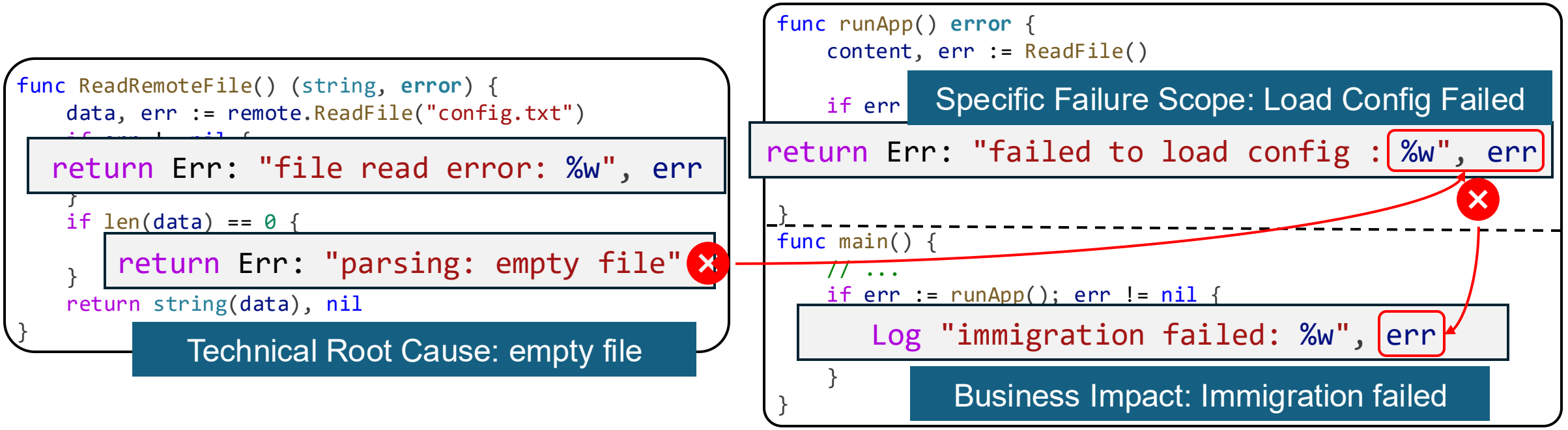
Error Wrapping Mechanism

- Modern languages (*e.g.* **Go** and **Rust**) treat errors as **explicit return values**, instead of exceptions as in Java and Python
- **Error Wrapping** mechanism allows functions to add context to *any* error
- The error propagation chain precisely describes a failure with structured context from its technical origin to its business impact



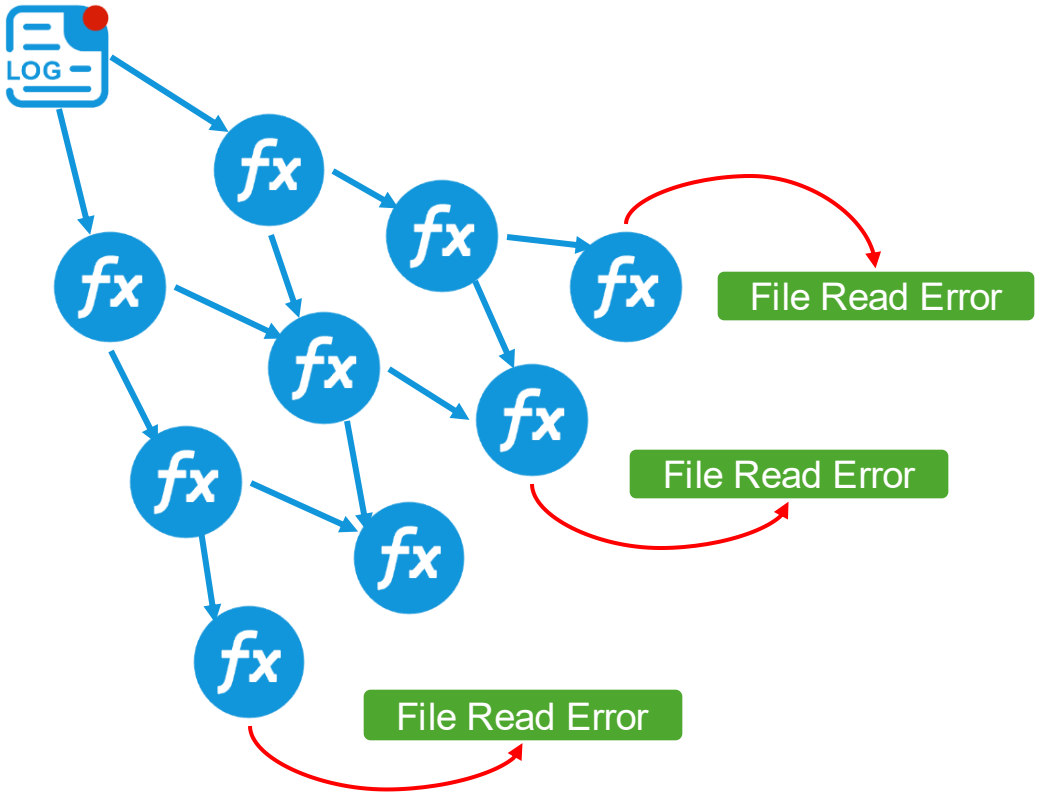
Error Wrapping Mechanism

- Modern languages (*e.g.* **Go** and **Rust**) treat errors as **explicit return values**, instead of exceptions as in Java and Python
- **Error Wrapping** mechanism allows functions to add context to *any* error
- The error propagation chain precisely describes a failure with structured context from its technical origin to its business impact



New Problem: **Error Obfuscation**

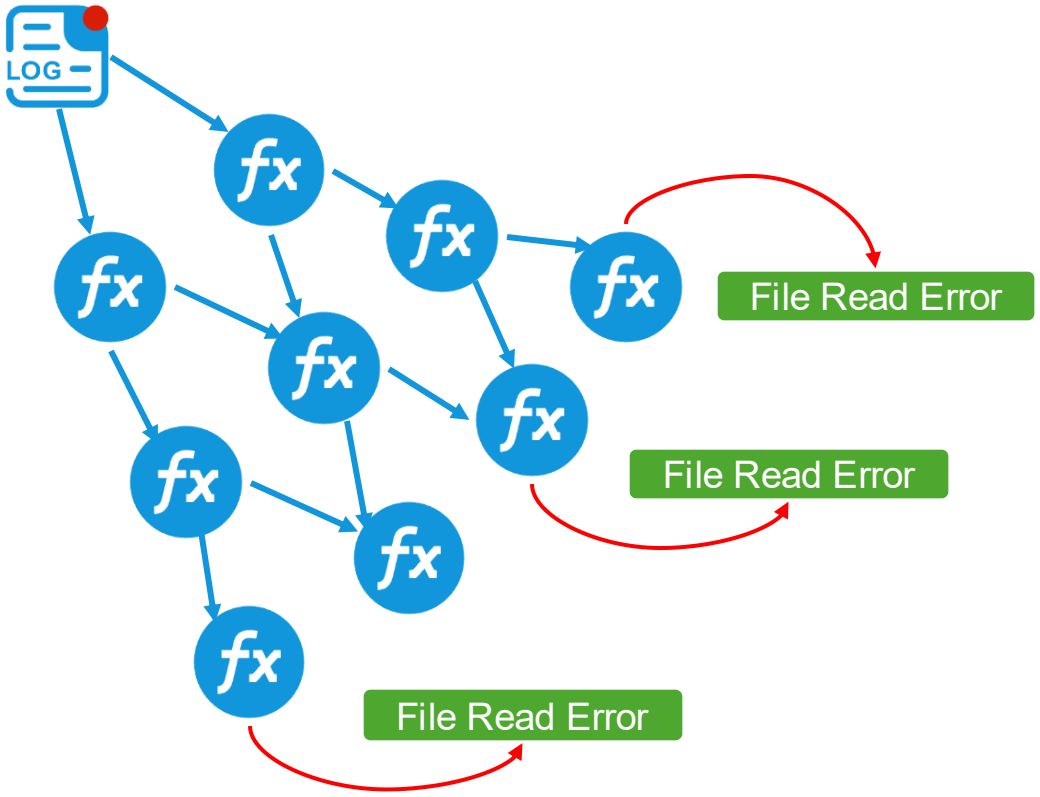
- **Definition**: The final error chain which includes context from *multiple functions*, is flattened into a single log string that **hides the runtime path*** in the final service.



*Naïve methods like hook Error type are unfeasible. Because it alters an error's original structure, it easily breaks the internal error-handling mechanisms of third-party frameworks (like Tauri), causing compatibility failures.

New Problem: **Error Obfuscation**

- **Definition**: The final error chain which includes context from *multiple functions*, is flattened into a single log string that **hides the runtime path*** in the final service.



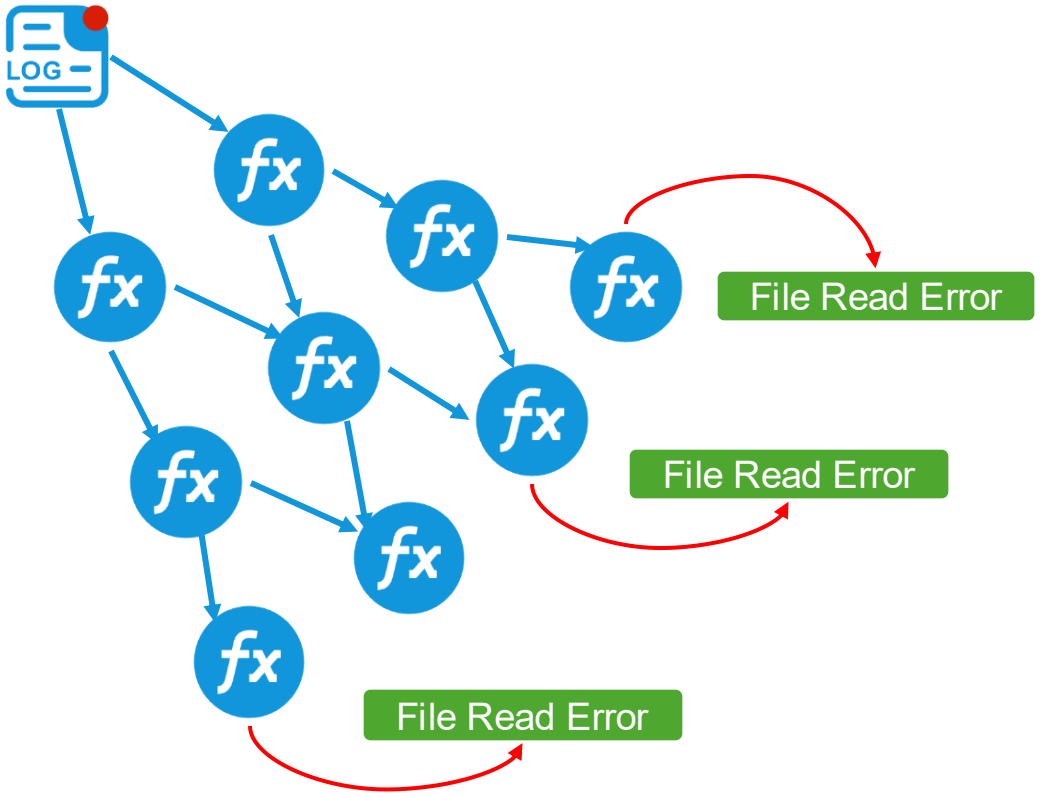
Example Error Message:

Immigration failed: failed to load config:
file read error: <*>

*Naïve methods like hook Error type are unfeasible. Because it alters an error's original structure, it easily breaks the internal error-handling mechanisms of third-party frameworks (like Tauri), causing compatibility failures.

New Problem: **Error Obfuscation**

- **Definition**: The final error chain which includes context from *multiple functions*, is flattened into a single log string that **hides the runtime path*** in the final service.



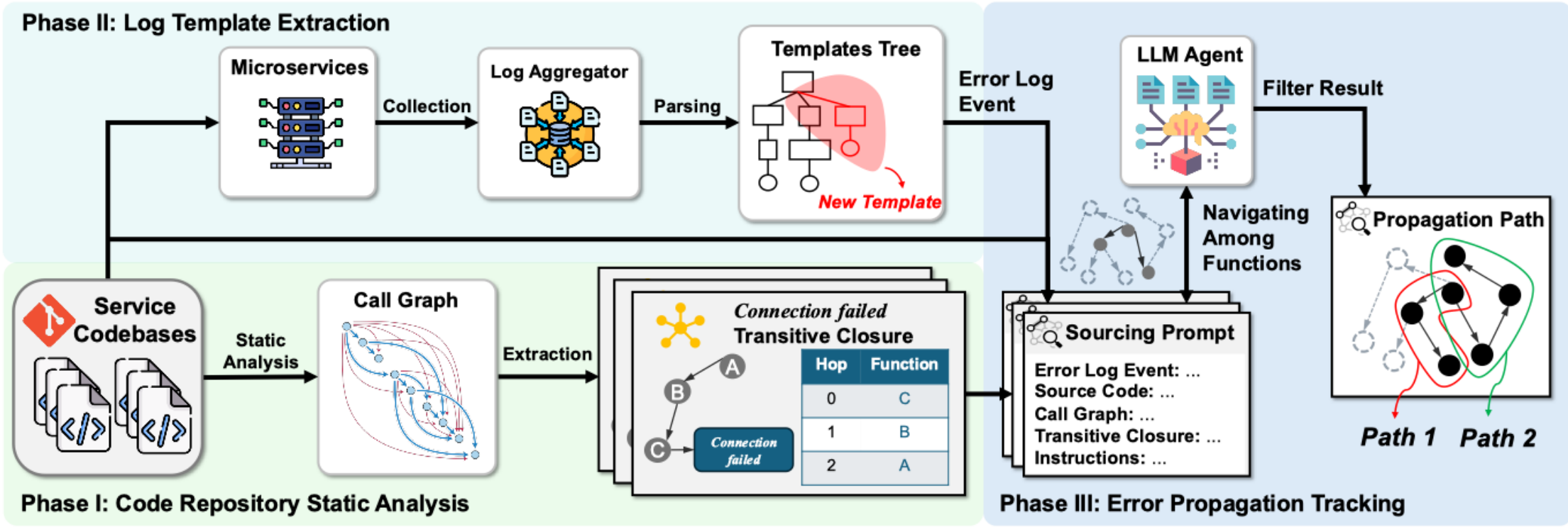
Example Error Message:

Immigration failed: failed to load config: **file read error**: <*>

There is too many paths that could lead to a file read error !

*Naïve methods like hook Error type are unfeasible. Because it alters an error's original structure, it easily breaks the internal error-handling mechanisms of third-party frameworks (like Tauri), causing compatibility failures.

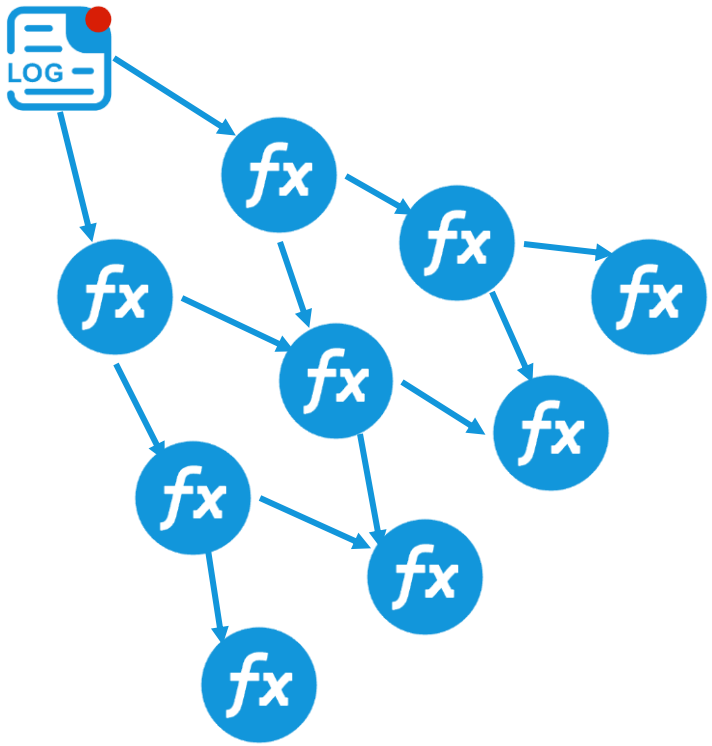
ErrorPrism: Overview



Offline (Phase I): Perform a one-time static analysis on selected functions to build a call graph and an error-string index

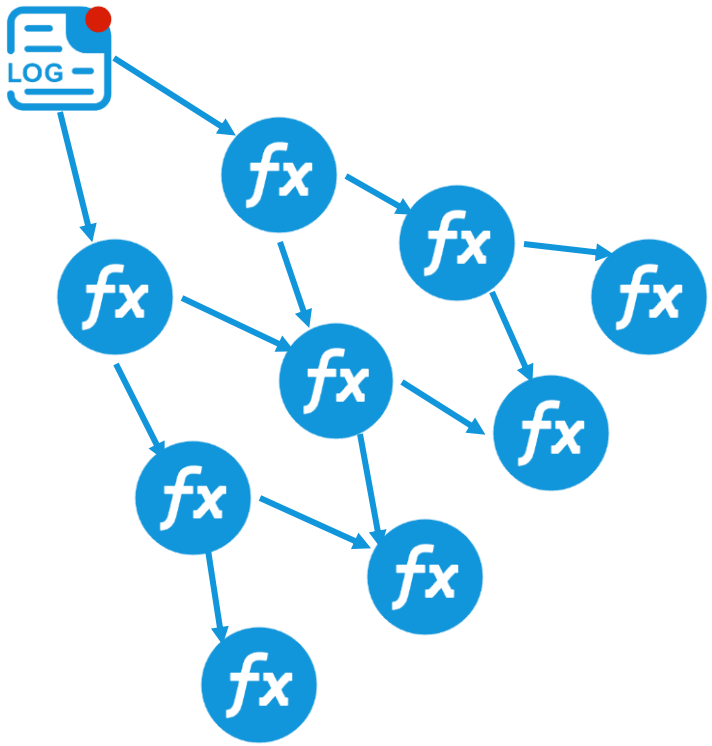
Online (Phase II & III): When a new log event is extracted, LLM agent uses pre-computed error-string index and call graph to perform a semantic, backward search and reconstruct the final error path

Static Analysis



Error log:
Immigration failed: failed to load config:
file read error: <*>

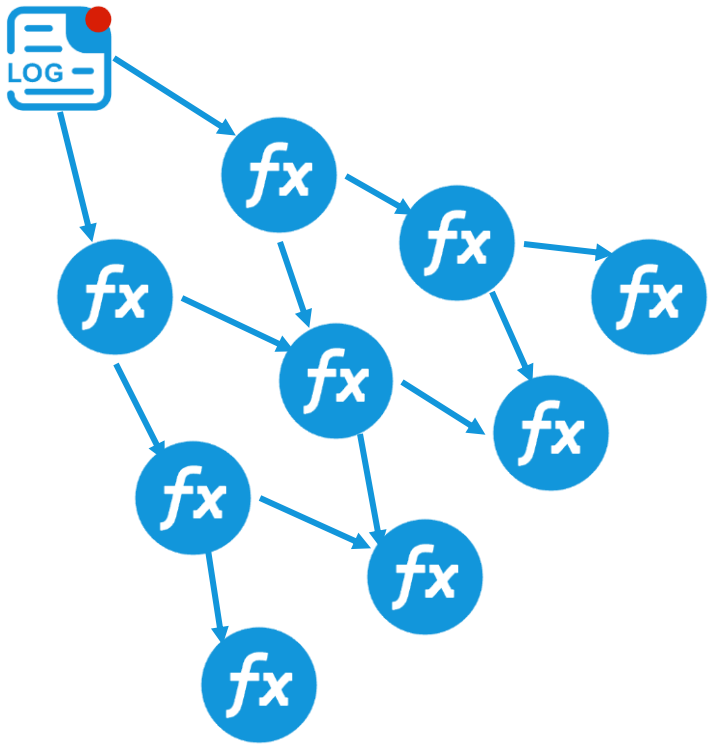
Static Analysis



Error log:
Immigration failed: failed to load config: file read error: <*>

- We build Function Call Graph first
- Too many paths in FCG
- **Need to Prune the vast majority of impossible paths**

Static Analysis

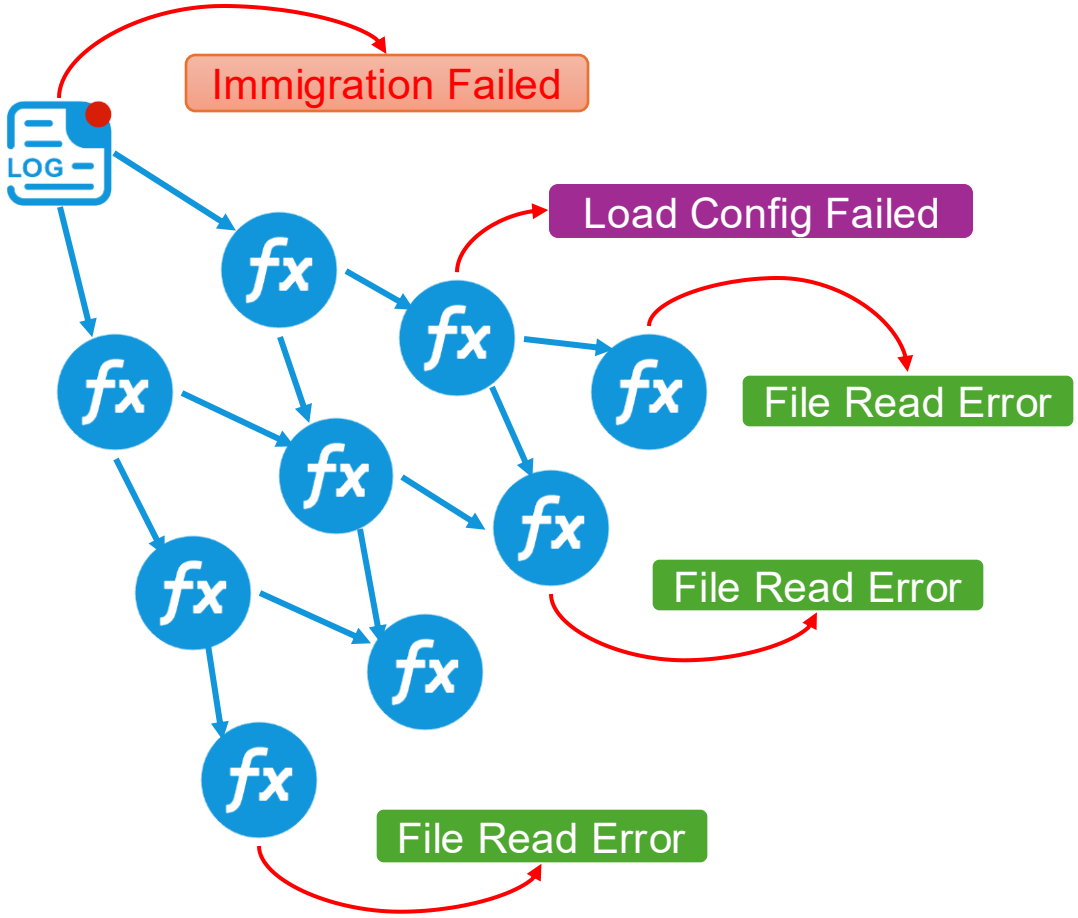


Error log:
Immigration failed: failed to load config: file read error: <*>

- We build Function Call Graph first
- Too many paths in FCG
- **Need to Prune the vast majority of impossible paths**

- A path is only *valid* if its functions contain the "Error Strings" we see in the final log

Static Analysis

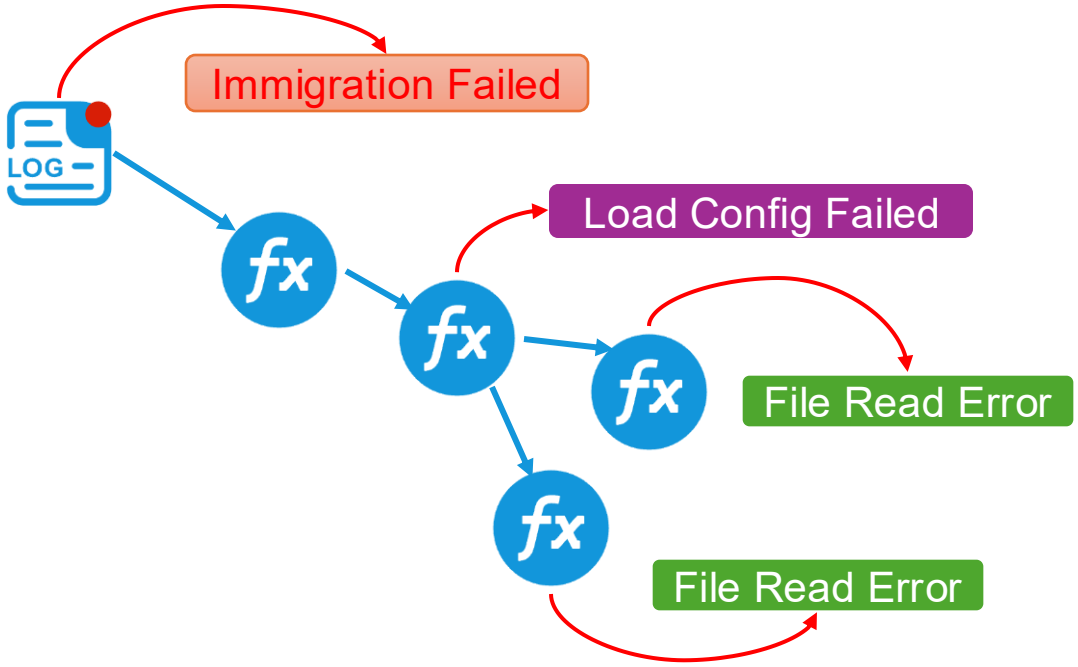


Error log:
Immigration failed: failed to load config: file read error: <*>

- We build Function Call Graph first
- Too many paths in FCG
- **Need to Prune the vast majority of impossible paths**

- A path is only *valid* if its functions contain the "Error Strings" we see in the final log

Static Analysis

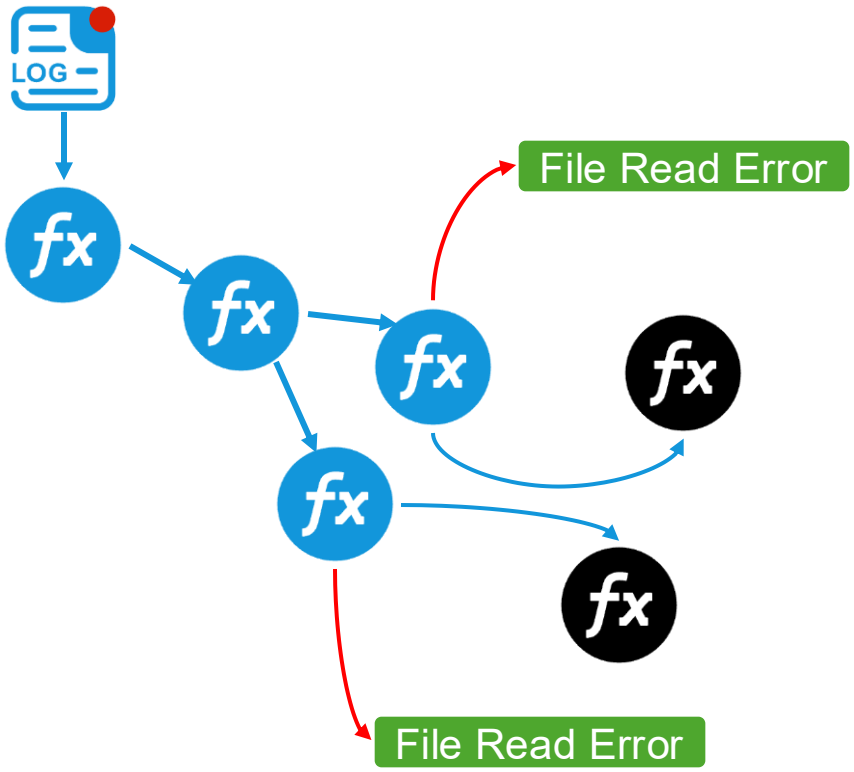


Error log:
Immigration failed: failed to load config: file read error: <*>

- We build Function Call Graph first
- Too many paths in FCG
- **Need to Prune the vast majority of impossible paths**

- A path is only *valid* if its functions contain the "Error Strings" we see in the final log

LLM Agent Reasoning

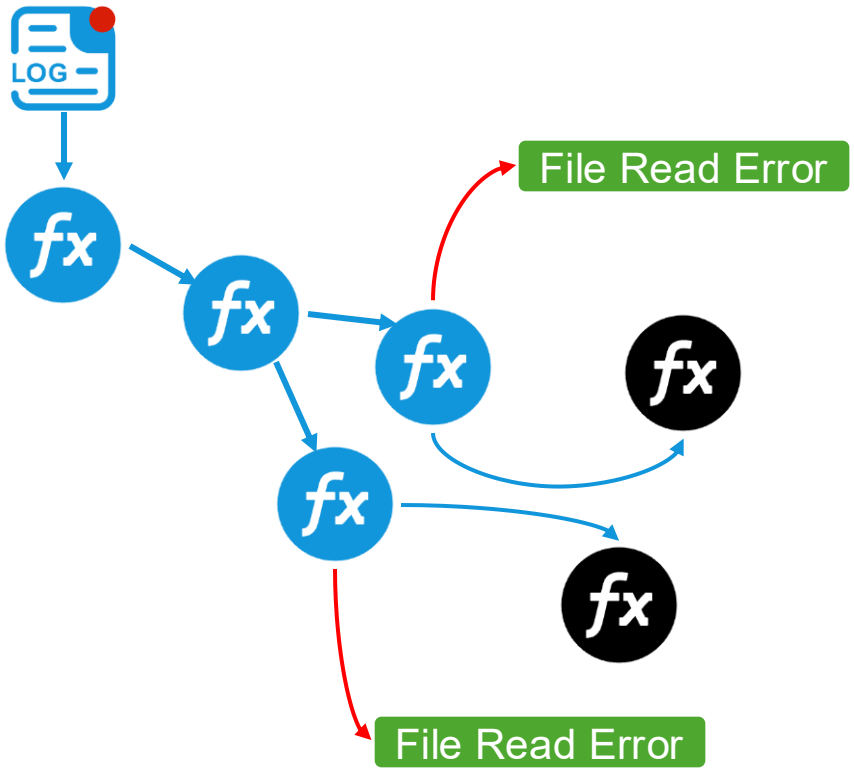


However, there can be multiple paths are structurally possible

Static analysis can't understand the intent of the code, so it gets stuck.

Error Templates:
Immigration failed: failed to load config: file read error: Failed to fetch cluster configuration…(**NOT IN INDEX**)

LLM Agent Reasoning



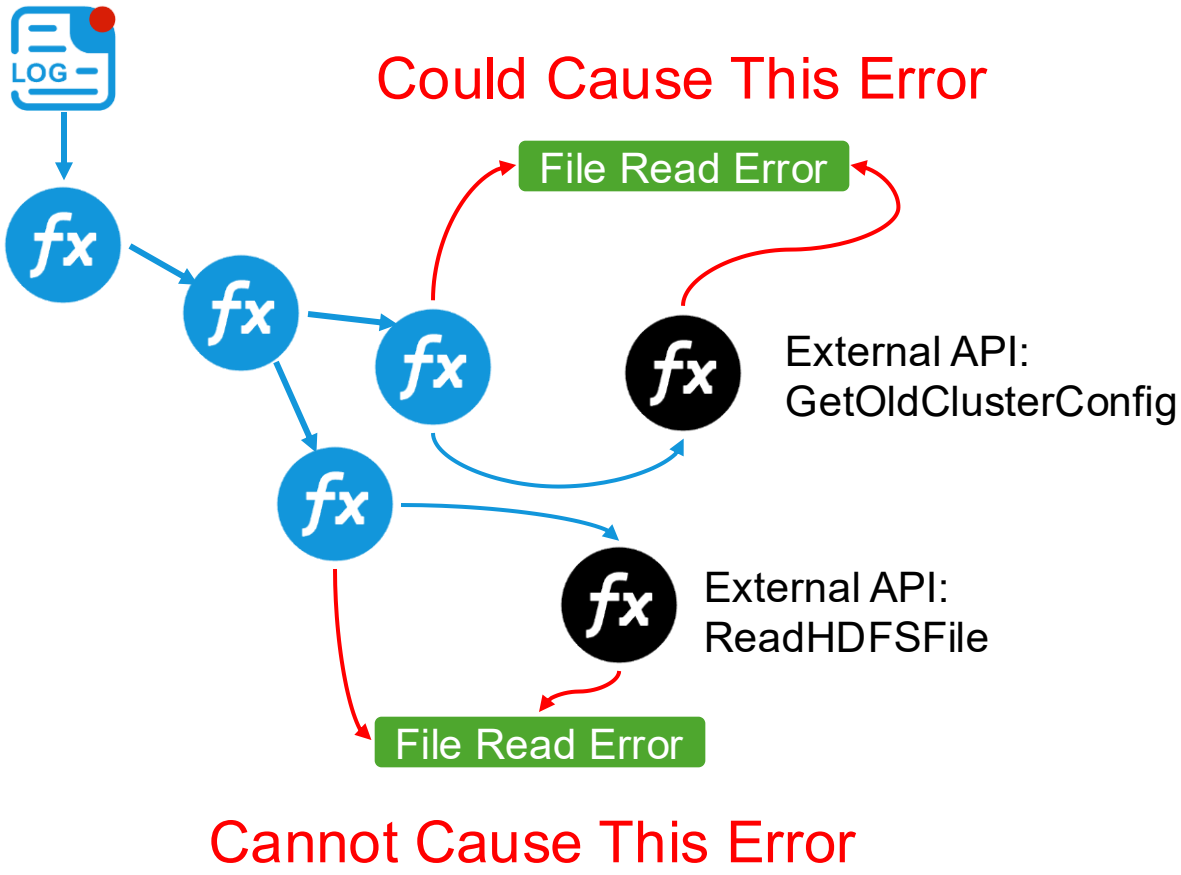
However, there can be multiple paths are structurally possible

Static analysis can't understand the intent of the code, so it gets stuck.

Error Templates:
Immigration failed: failed to load config: file read error: Failed to fetch cluster configuration…(**NOT IN INDEX**)

LLM Reasoning: Matching code intent to the **unknown** (**NOT IN INDEX**) string

LLM Agent Reasoning



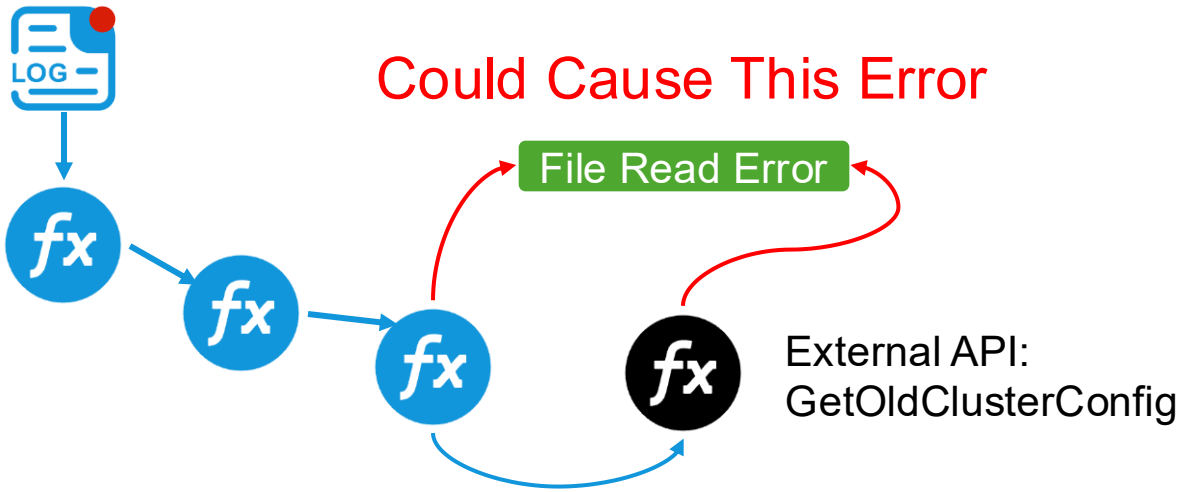
However, there can be multiple paths are structurally possible

Static analysis can't understand the intent of the code, so it gets stuck.

Error Templates:
Immigration failed: failed to load config: file read error: Failed to fetch cluster configuration…(**NOT IN INDEX**)

LLM Reasoning: Matching code intent to the **unknown** (**NOT IN INDEX**) string

LLM Agent Reasoning



However, there can be multiple paths are structurally possible

Static analysis can't understand the intent of the code, so it gets stuck.

Error Templates:
Immigration failed: failed to load config: file read error: Failed to fetch cluster configuration…(**NOT IN INDEX**)

LLM Reasoning: Matching code intent to the **unknown** (**NOT IN INDEX**) string

Evaluation: Dataset Construction

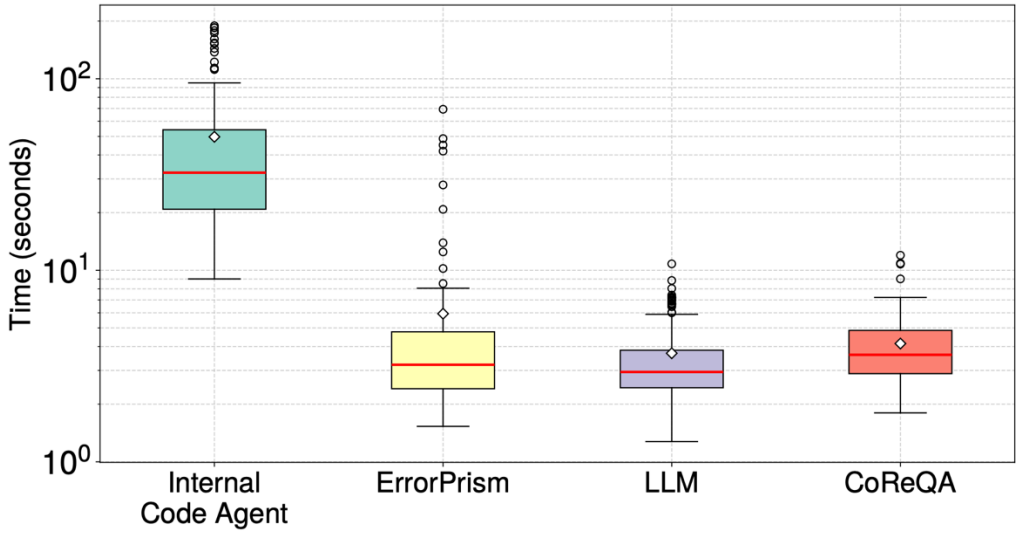
- **Environment:** Tested on a large-scale production platform at ByteDance.
- **Scope:** Analyzed **67** microservices, **~1 millon LOC** of Go code.
- **Ground Truth:** Manually collected and verified **102 real-world error events**.
- **Verification:** For each of the 102 events, engineers meticulously traced and recorded the complete, correct propagation path by hand.

Evaluation: Results

- **RQ1 (Accuracy*): ErrorPrism achieved 97.0% accuracy** in perfectly reconstructing the complete error paths. Accuracy remains high on long paths (>=3 hops) where other methods fail.
- **RQ2 (Efficiency): ErrorPrism** found the correct path **~8x faster** than the general-purpose code agent.

Method	Hop					Total
	0	1	2	3	≥4	
ErrorPrism	**100**	**100**	95.2	**100**	**85.7**	**97.0**
Static Analysis	100	90.4	**98.8**	72.9	66.1	90.7
Internal Agent	100	87.1	90.5	84.6	57.1	87.1
CoReQA	75	67.7	54.8	53.8	14.3	57.4
Pure LLM	100	64.5	45.2	30.8	0	50.5

The Performance of Error Propagation Tracking (%)



The Inference Time of Error Propagation Tracking

*_Accuracy_ means the predicted path **exactly matched** the ground truth, while _Precision_ (for static analysis) measures **the proportion of correct functions** in the candidate set (see paper for full formula).

Discussion: Why **ErrorPrism** Works

- **Static Pruning (Enables Efficiency):**
  - The offline static analysis (Phase I) *massively* reduces the search space.
  - This is the key to our high performance (5.9s) and high accuracy.
- **LLM Reasoning (Enables Accuracy):**
  - The LLM agent only needs to reason about a *few* candidate functions.
  - This allows it to perform deep semantic analysis to solve semantic blind spots that static analysis alone can't.

Conclusion & Contributions

- We Identified **Error Obfuscation** as a critical challenge for microservice reliability.
- We Proposed **ErrorPrism**, a novel hybrid framework, which use static analysis to prune the search space and an LLM agent to reason within that space.
- **ErrorPrism** proved to be highly effective (97% accuracy) and efficient (5.9s per request) in a large-scale industrial setting. This is a practical tool for automating Root Cause Analysis (RCA).

中山大學
SUN YAT-SEN UNIVERSITY

ByteDance
字节跳动

Thank you!

Junsong Pu, 蒲俊宋

School of Software Engineering, Sun Yat-sen University

E-mail: pujs@mail2.sysu.edu.cn / angrychow@qq.com