Transparent DIFC: Harnessing Innate Application Event Logging for Fine-Grained Decentralized Information Flow Control

Jason Liu, Anant Kandikuppa, Adam Bates

7th IEEE European Symposium on Security and Privacy
June 9, 2022

Data Breaches Are Alarmingly Common

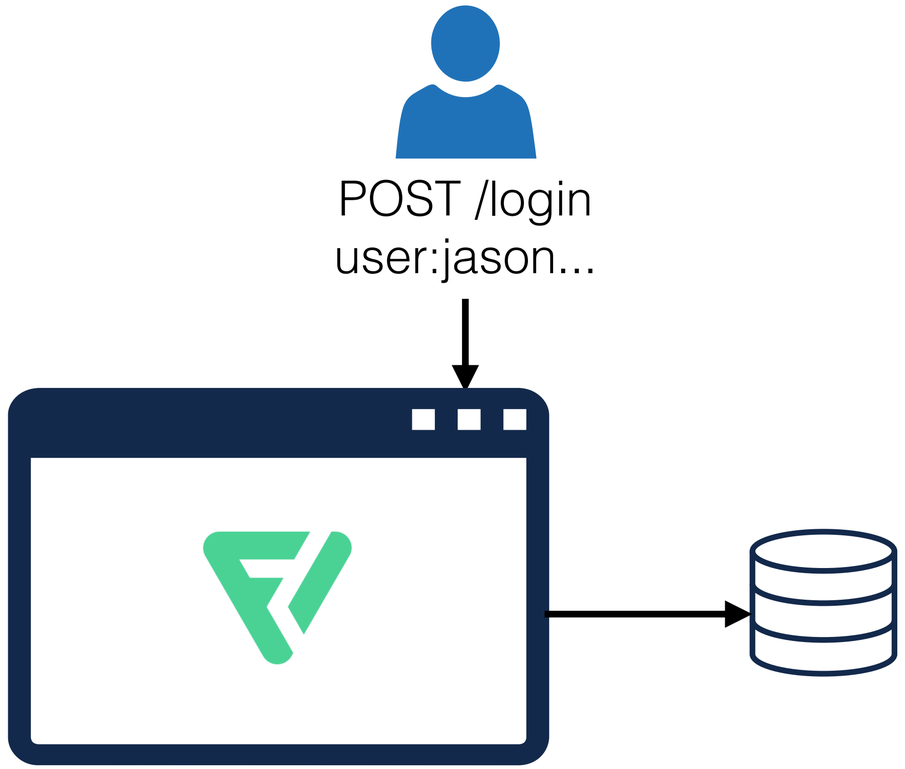
"We have recently notified all affected users of a security breach in Freepik Company, affecting Freepik and Flaticon. The security breach was due to a **SQL injection** in Flaticon that allowed an attacker to get some user's information from our database." [1]

Data of 8.3M users was stolen by one attacker!

[1] FreePik, Aug. 2020. https://www.freepikcompany.com/newsroom/statement-on-security-incident-at-freepik-company/
Flaticon icons created by Freepik - Flaticon

Data Breaches Are Alarmingly Common

"We have recently notified all affected users of a security breach in Freepik Company, affecting Freepik and Flaticon. The security breach was due to a **SQL injection** in Flaticon that allowed an attacker to get some user's information from our database." [1]

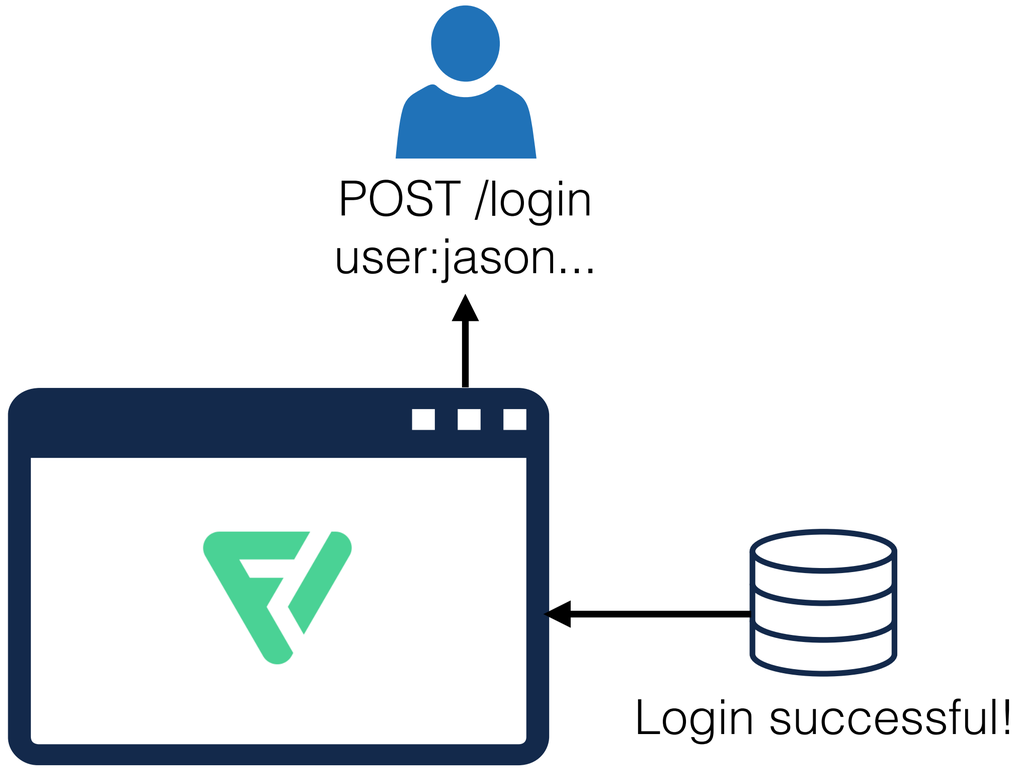


Data of 8.3M users was stolen by one attacker!

[1] FreePik, Aug. 2020. https://www.freepikcompany.com/newsroom/statement-on-security-incident-at-freepik-company/
Flaticon icons created by Freepik - Flaticon

Data Breaches Are Alarmingly Common

"We have recently notified all affected users of a security breach in Freepik Company, affecting Freepik and Flaticon. The security breach was due to a **SQL injection** in Flaticon that allowed an attacker to get some user's information from our database." [1]

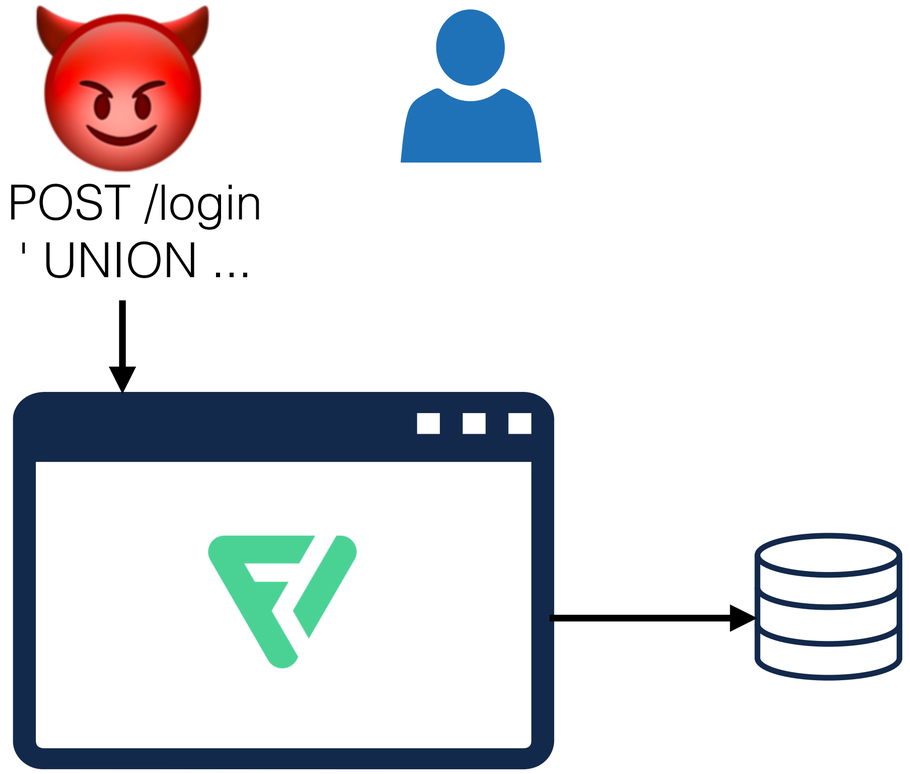


Data of 8.3M users was stolen by one attacker!

[1] FreePik, Aug. 2020. https://www.freepikcompany.com/newsroom/statement-on-security-incident-at-freepik-company/
Flaticon icons created by Freepik - Flaticon

Data Breaches Are Alarmingly Common

"We have recently notified all affected users of a security breach in Freepik Company, affecting Freepik and Flaticon. The security breach was due to a **SQL injection** in Flaticon that allowed an attacker to get some user's information from our database." [1]

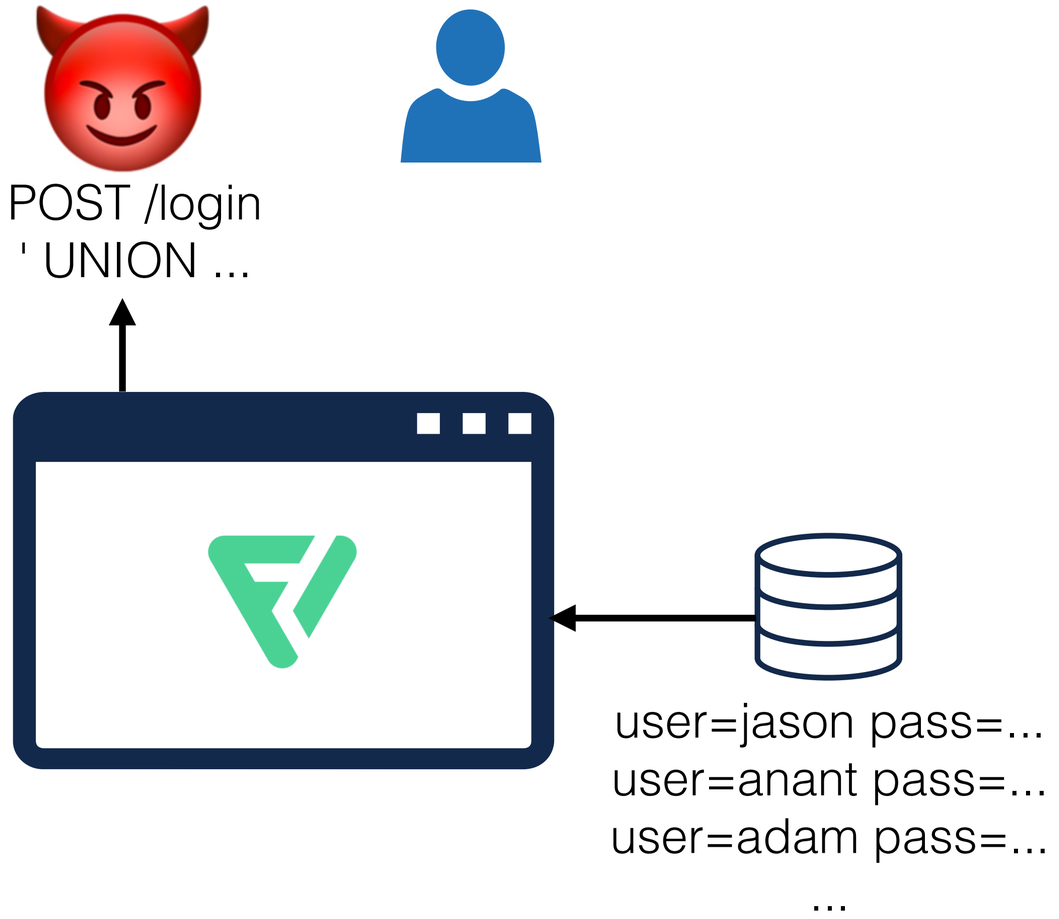


Data of 8.3M users was stolen by one attacker!

[1] FreePik, Aug. 2020. https://www.freepikcompany.com/newsroom/statement-on-security-incident-at-freepik-company/
Flaticon icons created by Freepik - Flaticon

Data Breaches Are Alarmingly Common

"We have recently notified all affected users of a security breach in Freepik Company, affecting Freepik and Flaticon. The security breach was due to a **SQL injection** in Flaticon that allowed an attacker to get some user's information from our database." [1]



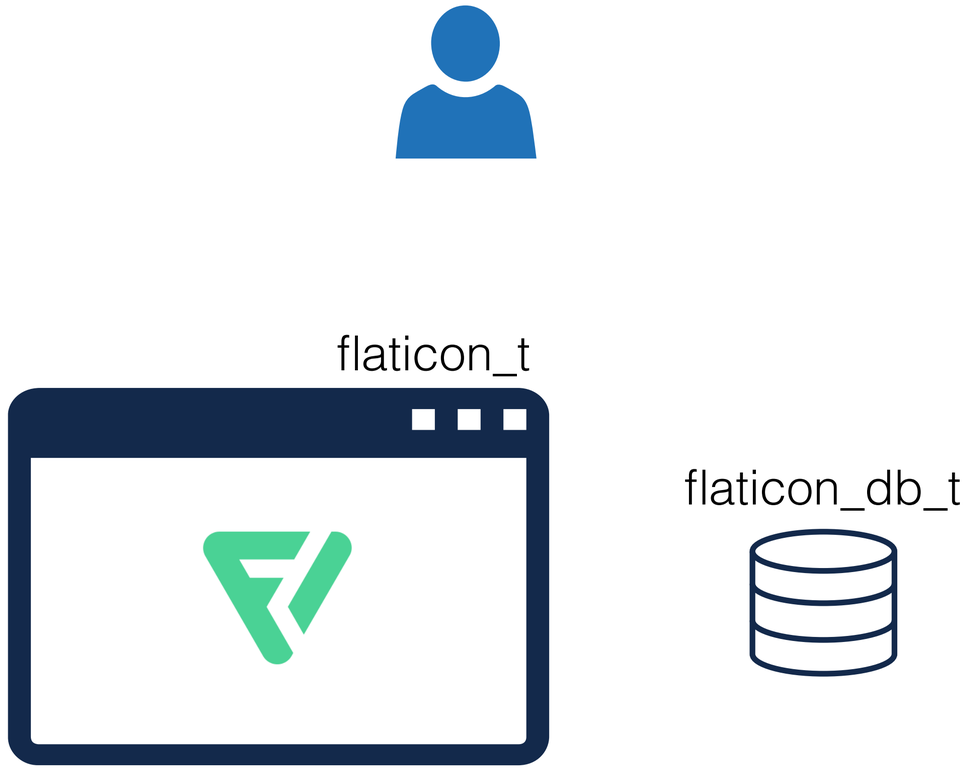
Data of 8.3M users was stolen by one attacker!

[1] FreePik, Aug. 2020. https://www.freepikcompany.com/newsroom/statement-on-security-incident-at-freepik-company/

Flaticon icons created by Freepik - Flaticon

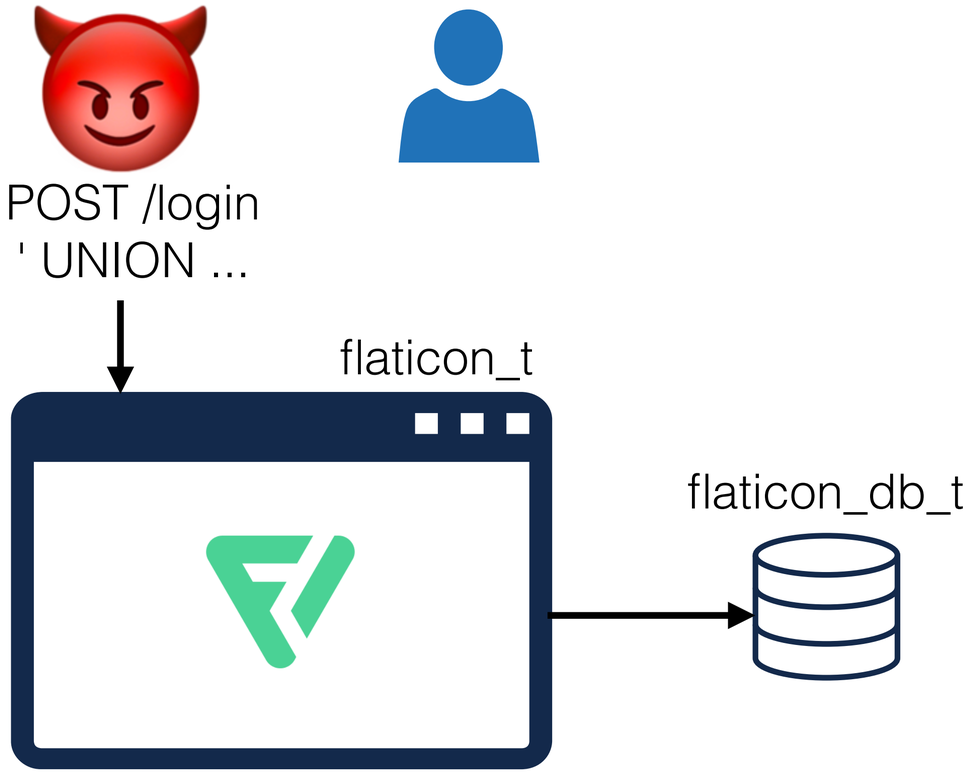
What Went Wrong?

- Data confidentiality from MAC
  - e.g., SELinux, AWS IAM
- `allow flaticon_t flaticon_db_t:db_table *;`
- Traditional MAC cannot distinguish application-level users or data!



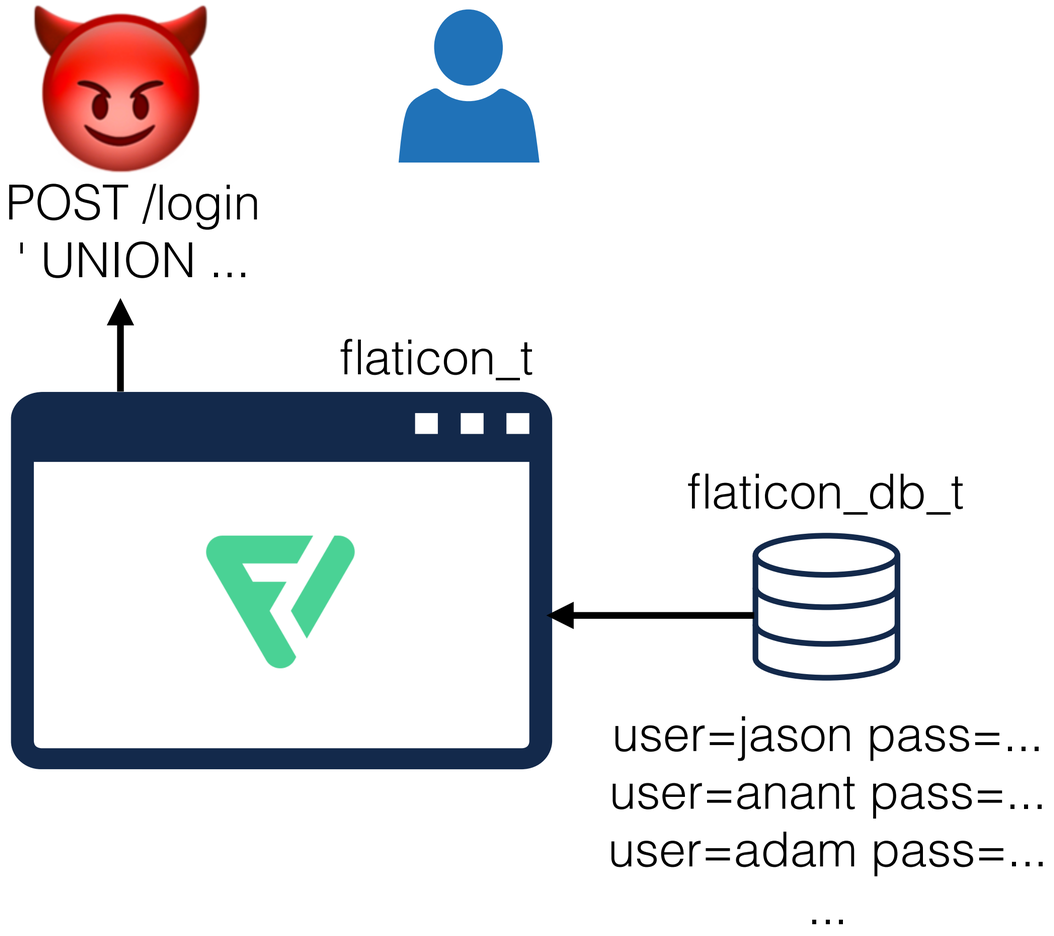
What Went Wrong?

- Data confidentiality from MAC
  - e.g., SELinux, AWS IAM
- `allow flaticon_t flaticon_db_t:db_table *;`
- Traditional MAC cannot distinguish application-level users or data!



What Went Wrong?

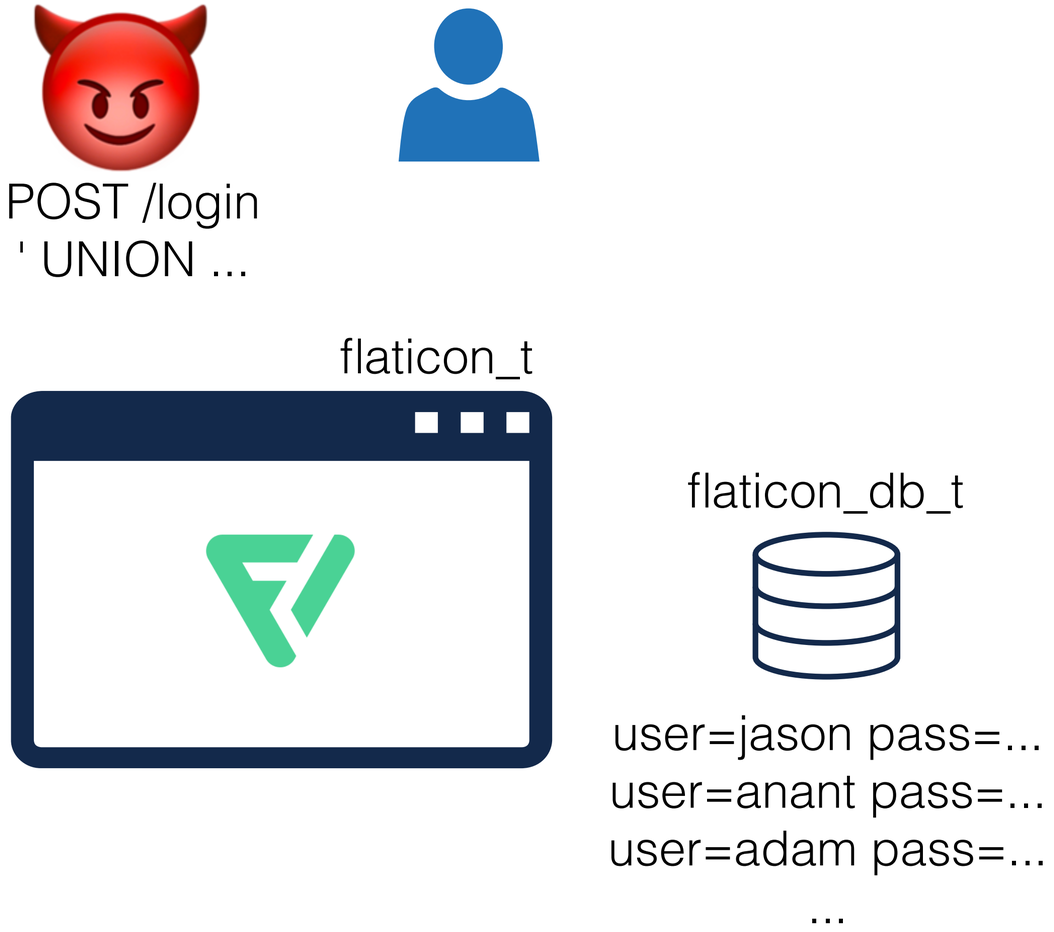
- Data confidentiality from MAC
  - e.g., SELinux, AWS IAM
- `allow flaticon_t flaticon_db_t:db_table *;`
- Traditional MAC cannot distinguish application-level users or data!



What Went Wrong?

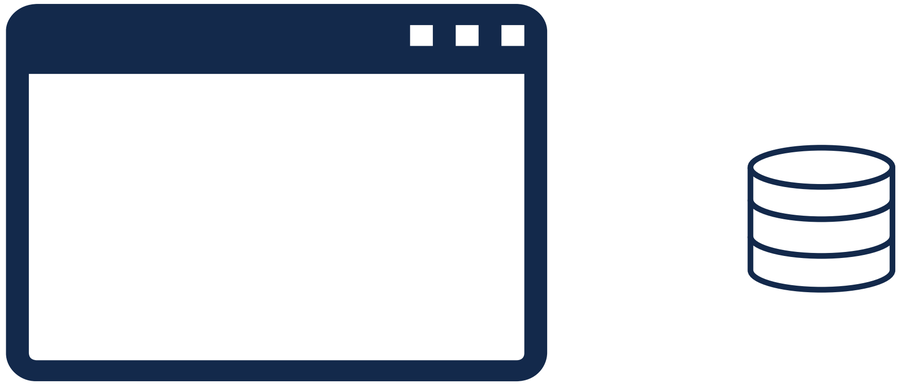
- Data confidentiality from MAC
  - e.g., SELinux, AWS IAM
- `allow flaticon_t flaticon_db_t:db_table *;`
- Traditional MAC cannot distinguish application-level users or data!

Actually, this problem was already solved in 1997...



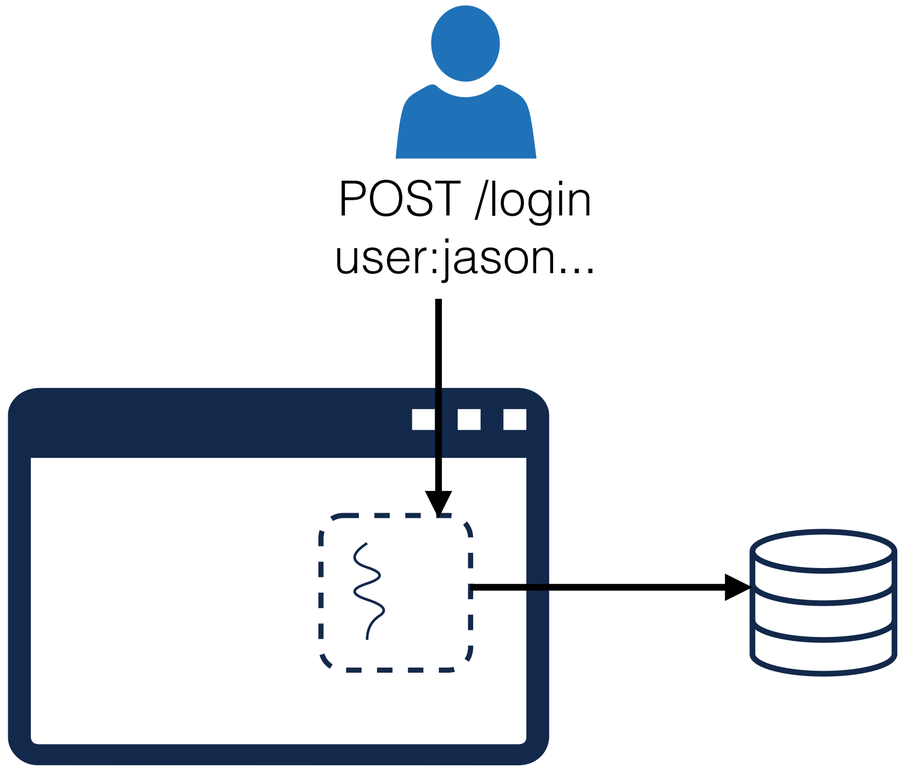
DIFC: Application-Level Security

- Decentralized Information Flow Control (DIFC): applications can define flow control rules
  - Finer-grained labeling of application-level users and data



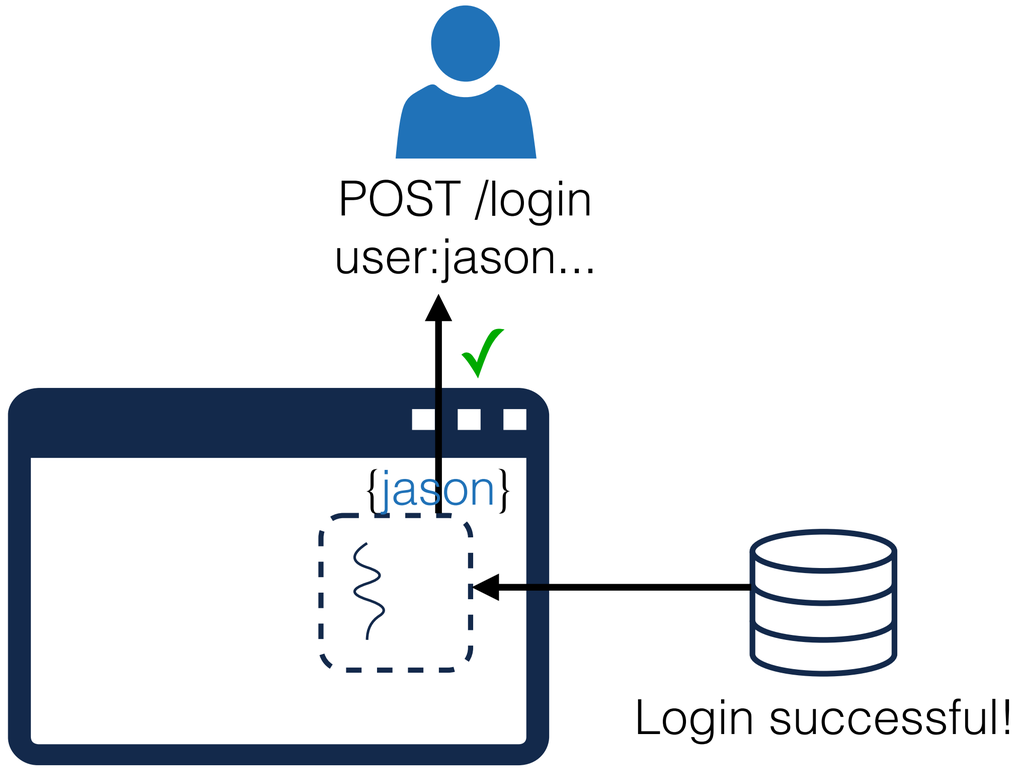
DIFC: Application-Level Security

- Decentralized Information Flow Control (DIFC): applications can define flow control rules
  - Finer-grained labeling of application-level users and data



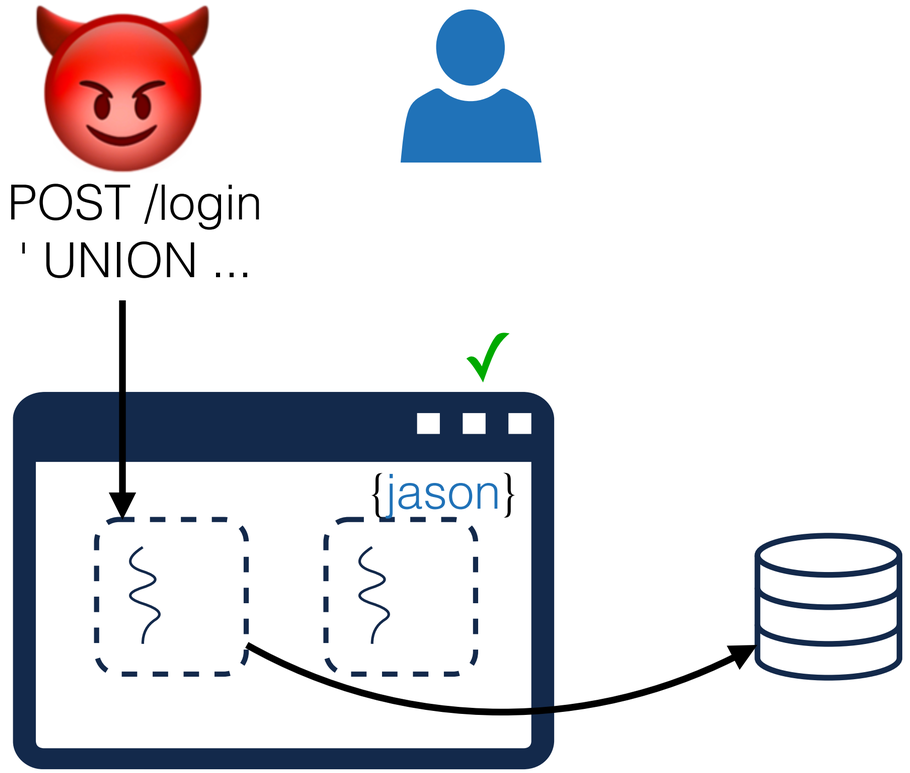
DIFC: Application-Level Security

- Decentralized Information Flow Control (DIFC): applications can define flow control rules
  - Finer-grained labeling of application-level users and data



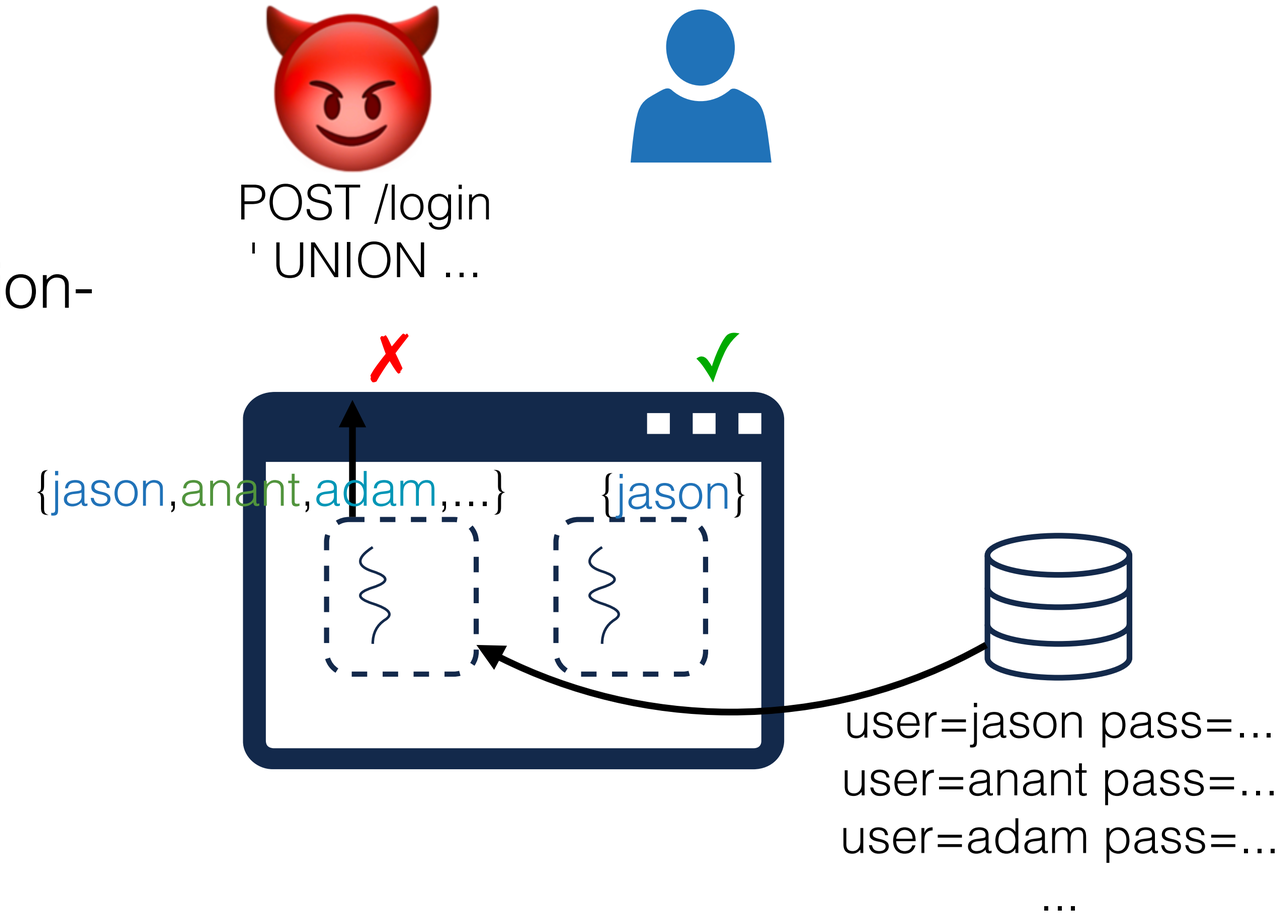
DIFC: Application-Level Security

- Decentralized Information Flow Control (DIFC): applications can define flow control rules
  - Finer-grained labeling of application-level users and data



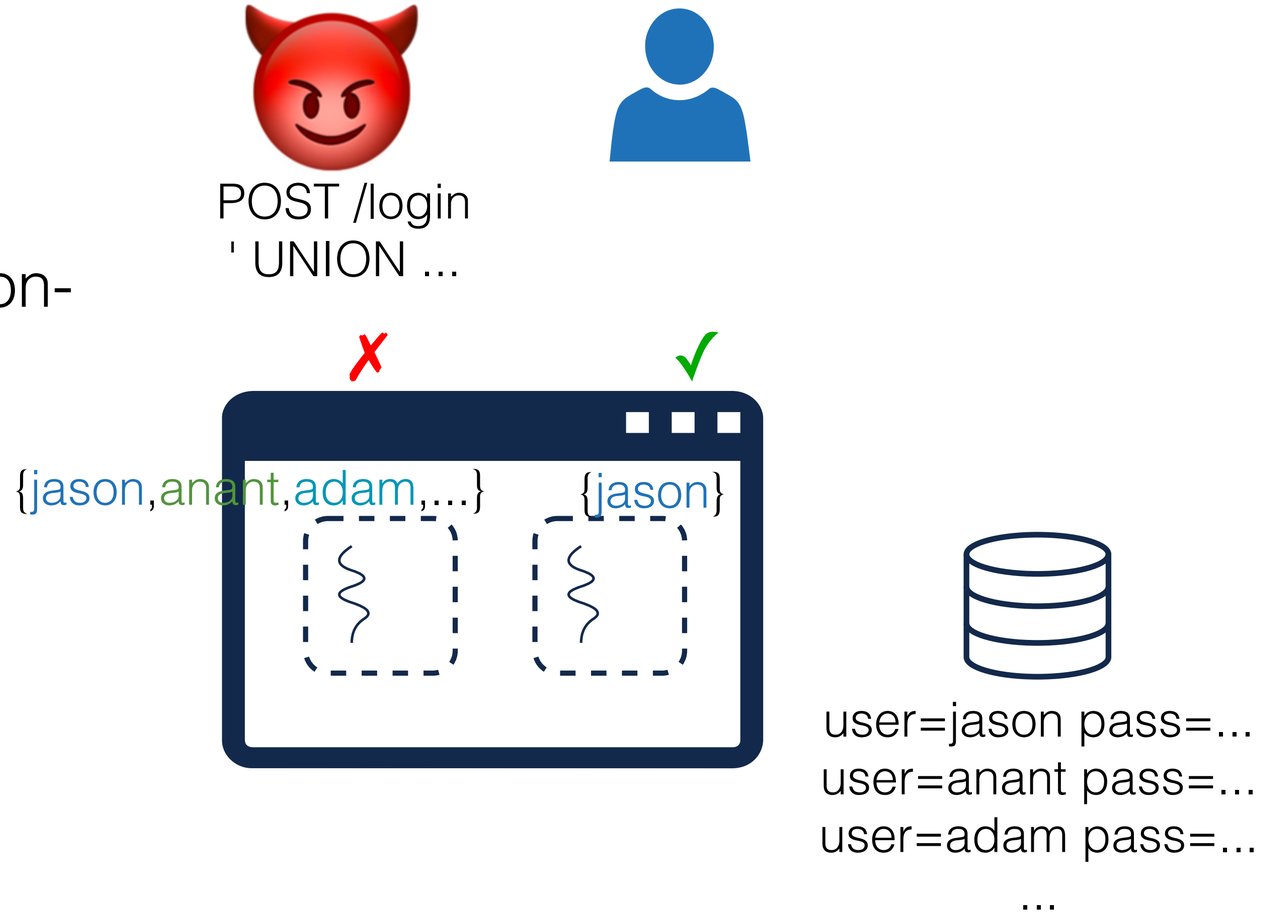
DIFC: Application-Level Security

- Decentralized Information Flow Control (DIFC): applications can define flow control rules
  - Finer-grained labeling of application-level users and data



DIFC: Application-Level Security

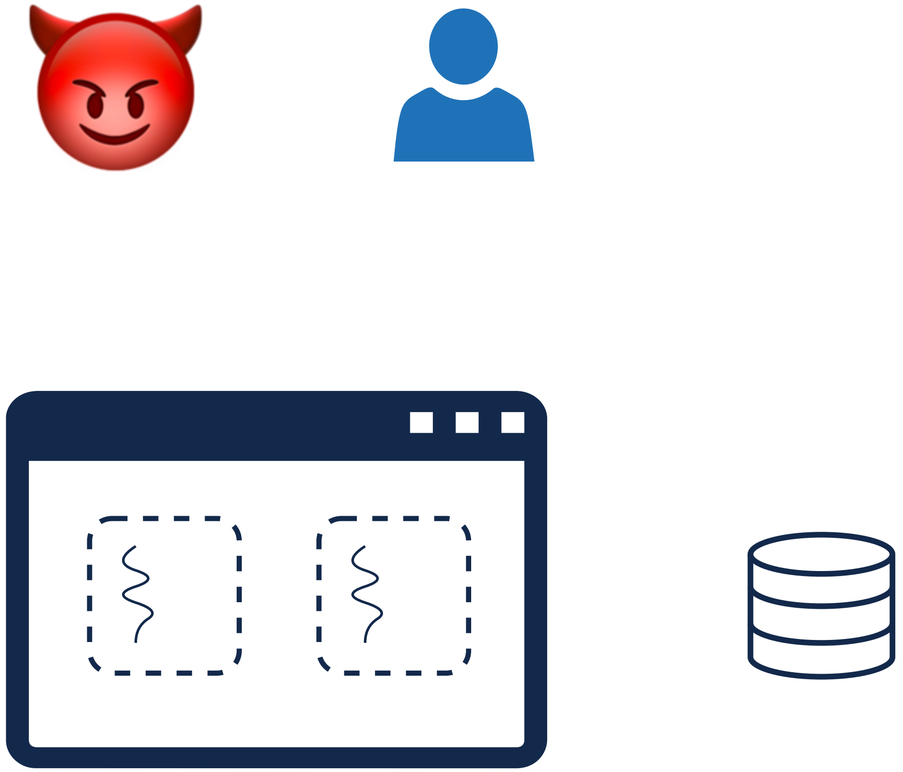
- Decentralized Information Flow Control (DIFC): applications can define flow control rules
  - Finer-grained labeling of application-level users and data



Why does DIFC remain unused despite its advantages?

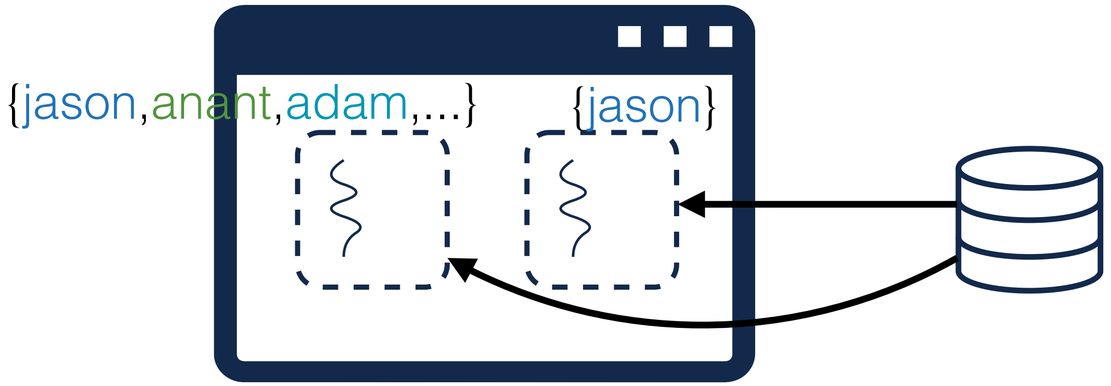
DIFC Requirements

- Modify code to:
  - Label threads and data
  - Specify allowable flows
  - Declassify data safely



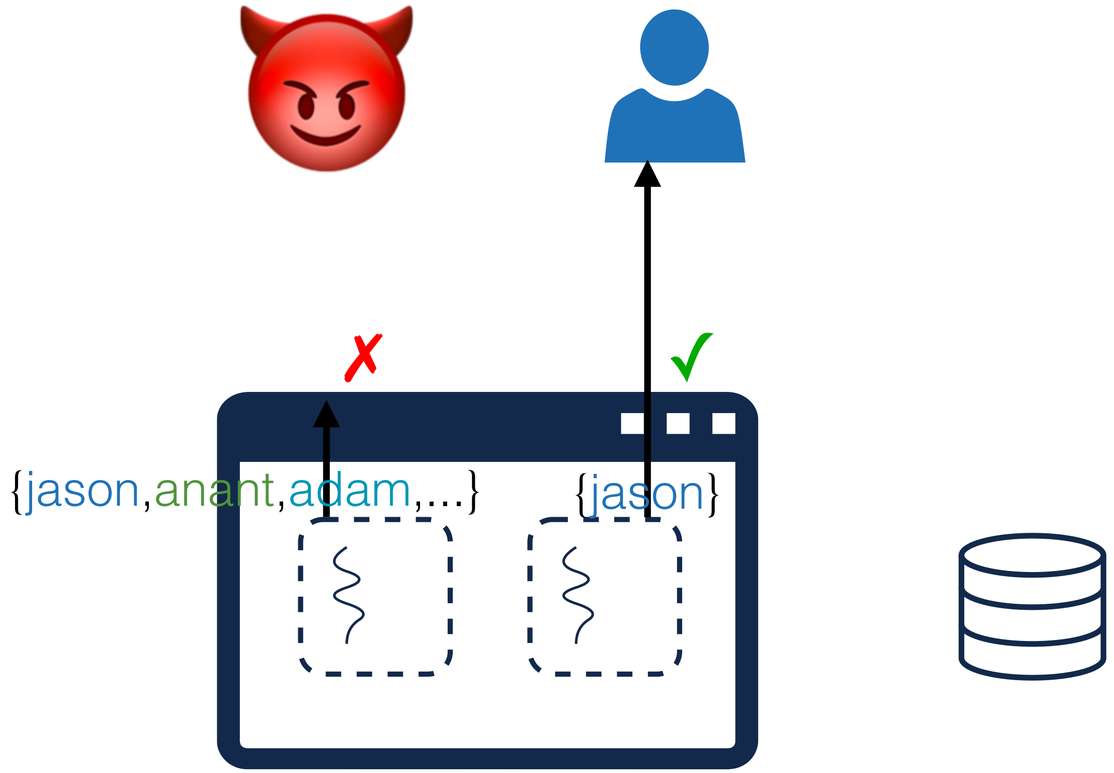
DIFC Requirements

- Modify code to:
  - Label threads and data
  - Specify allowable flows
  - Declassify data safely



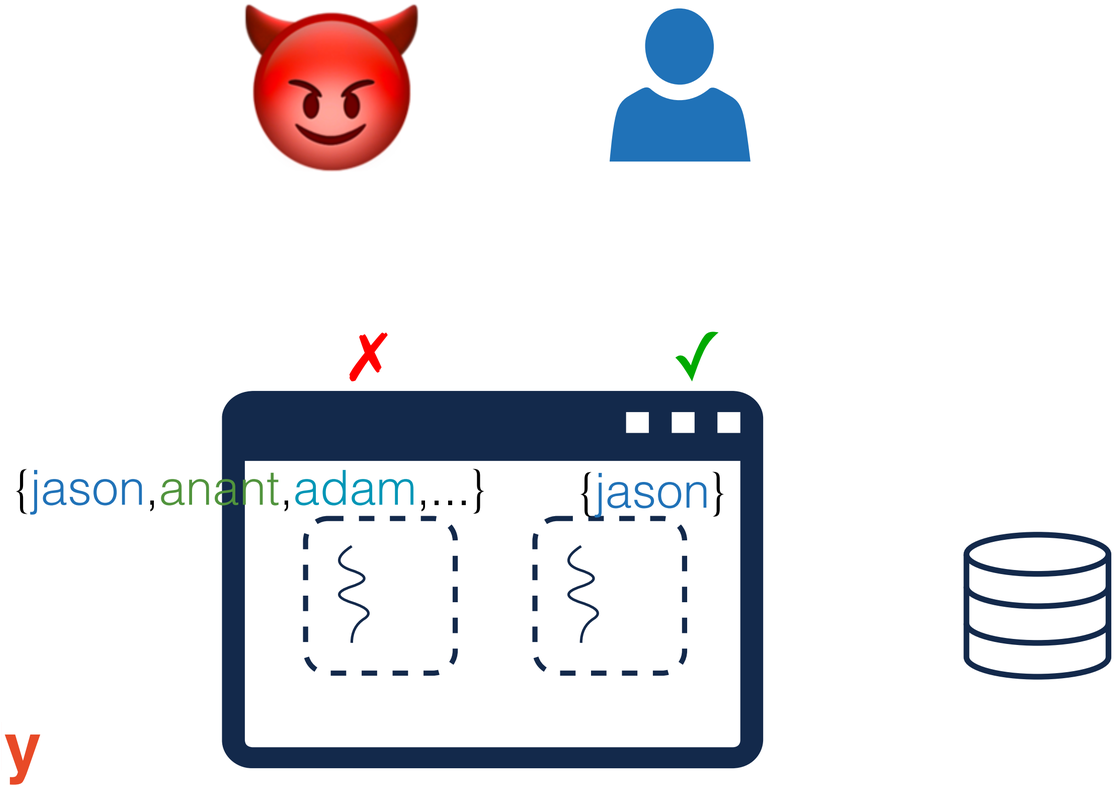
DIFC Requirements

- Modify code to:
  - Label threads and data
  - Specify allowable flows
  - Declassify data safely



DIFC Requirements

- Modify code to:
  - Label threads and data
  - Specify allowable flows
  - Declassify data safely

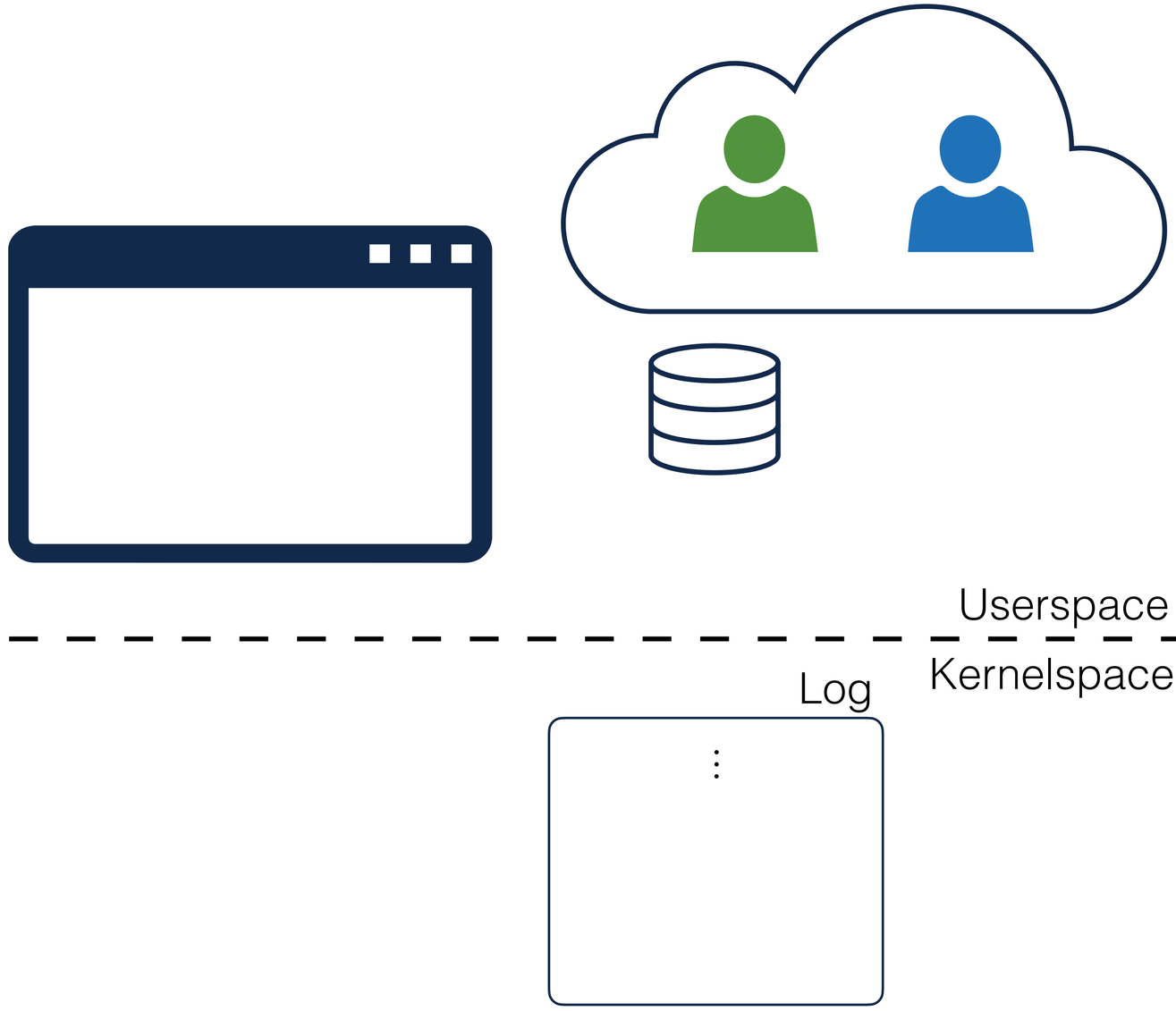


Such code changes are too costly to justify widespread deployment

Can we enforce DIFC policies transparently?

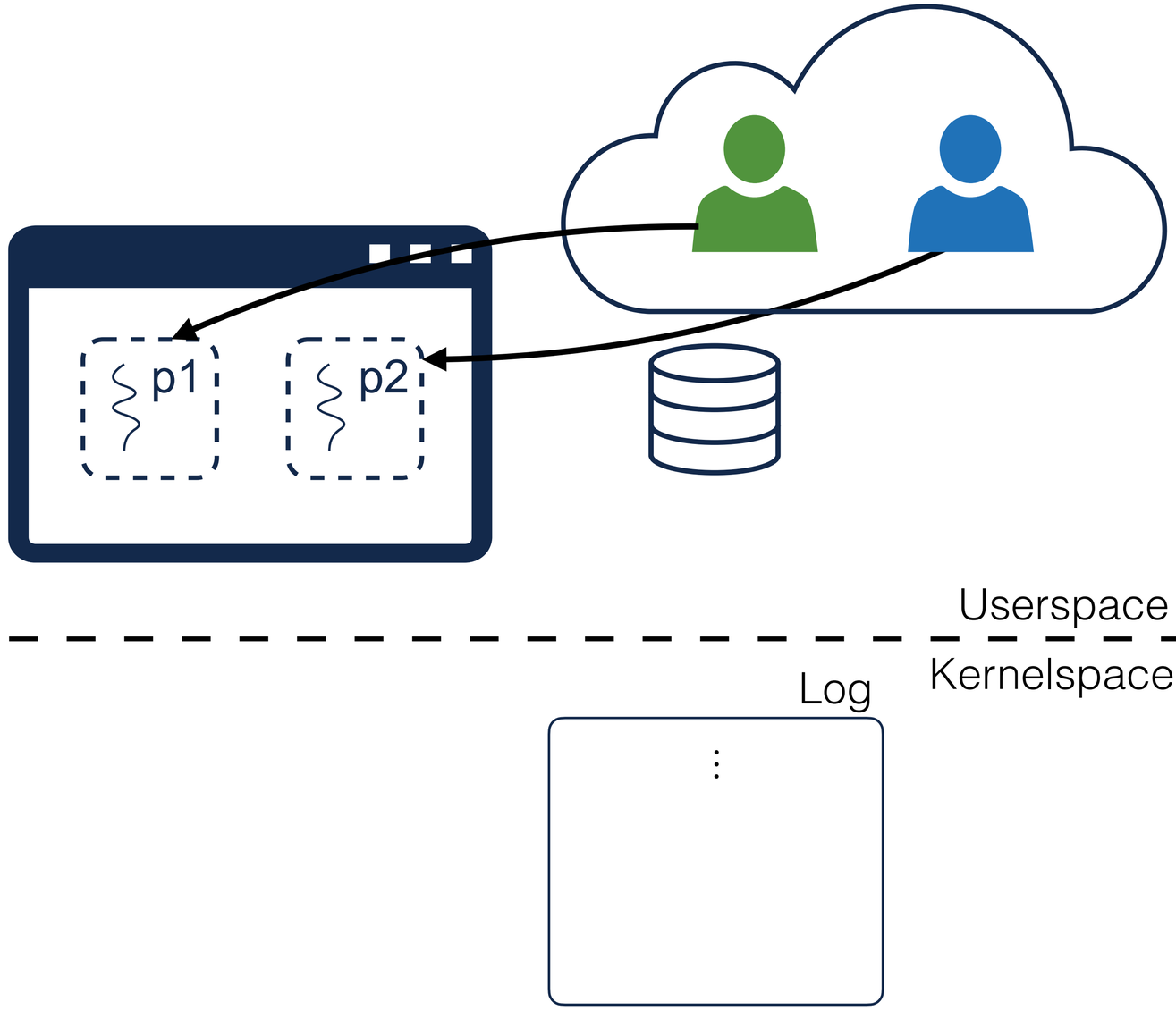
Devs Have Already Done (Some of) the Work

- Best practices dictate logging key events
- Logs contain application-level information
  - Including threading!
- Applications convey this to the reference monitor via write syscalls



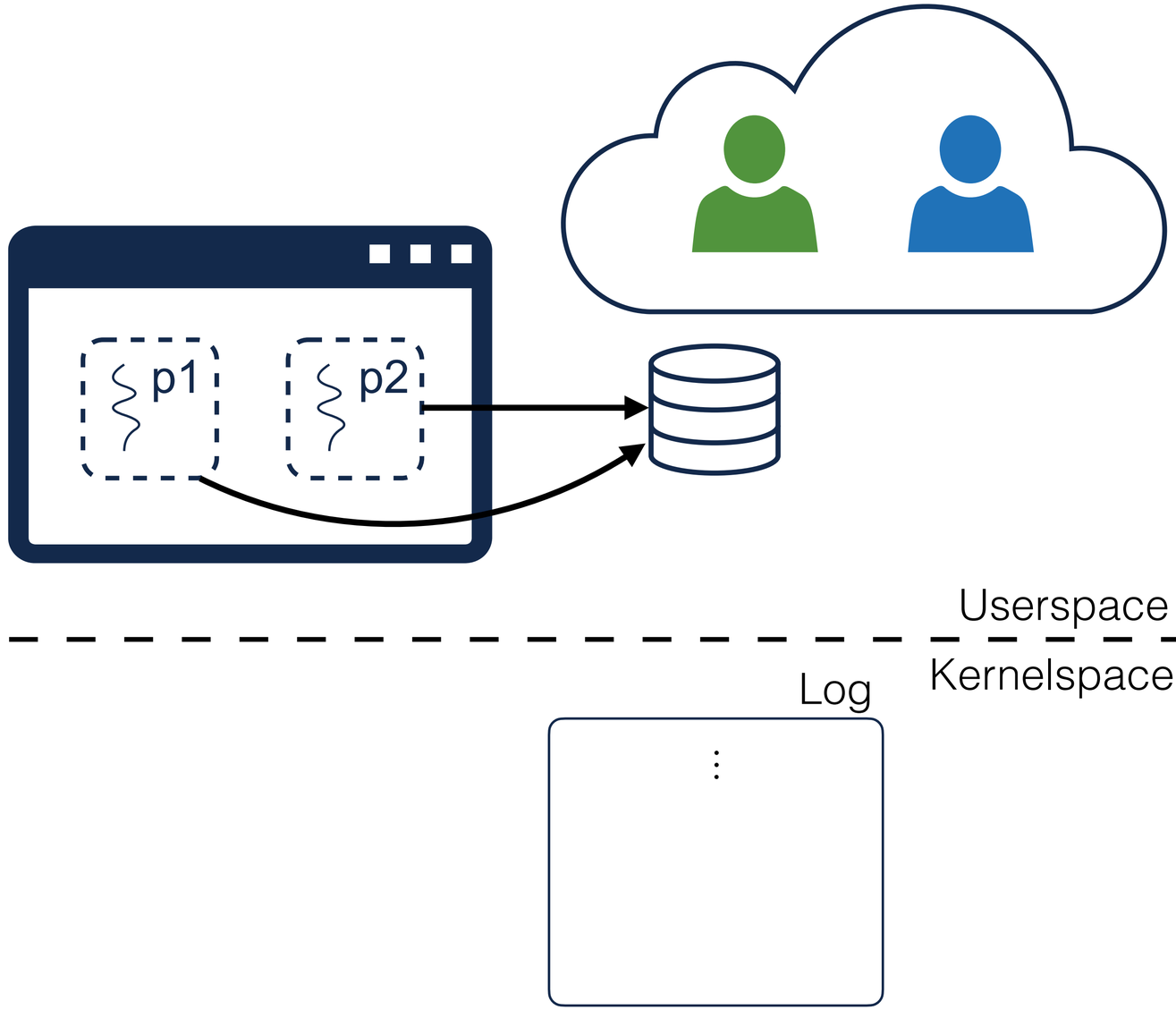
Devs Have Already Done (Some of) the Work

- Best practices dictate logging key events
- Logs contain application-level information
  - Including threading!
- Applications convey this to the reference monitor via write syscalls



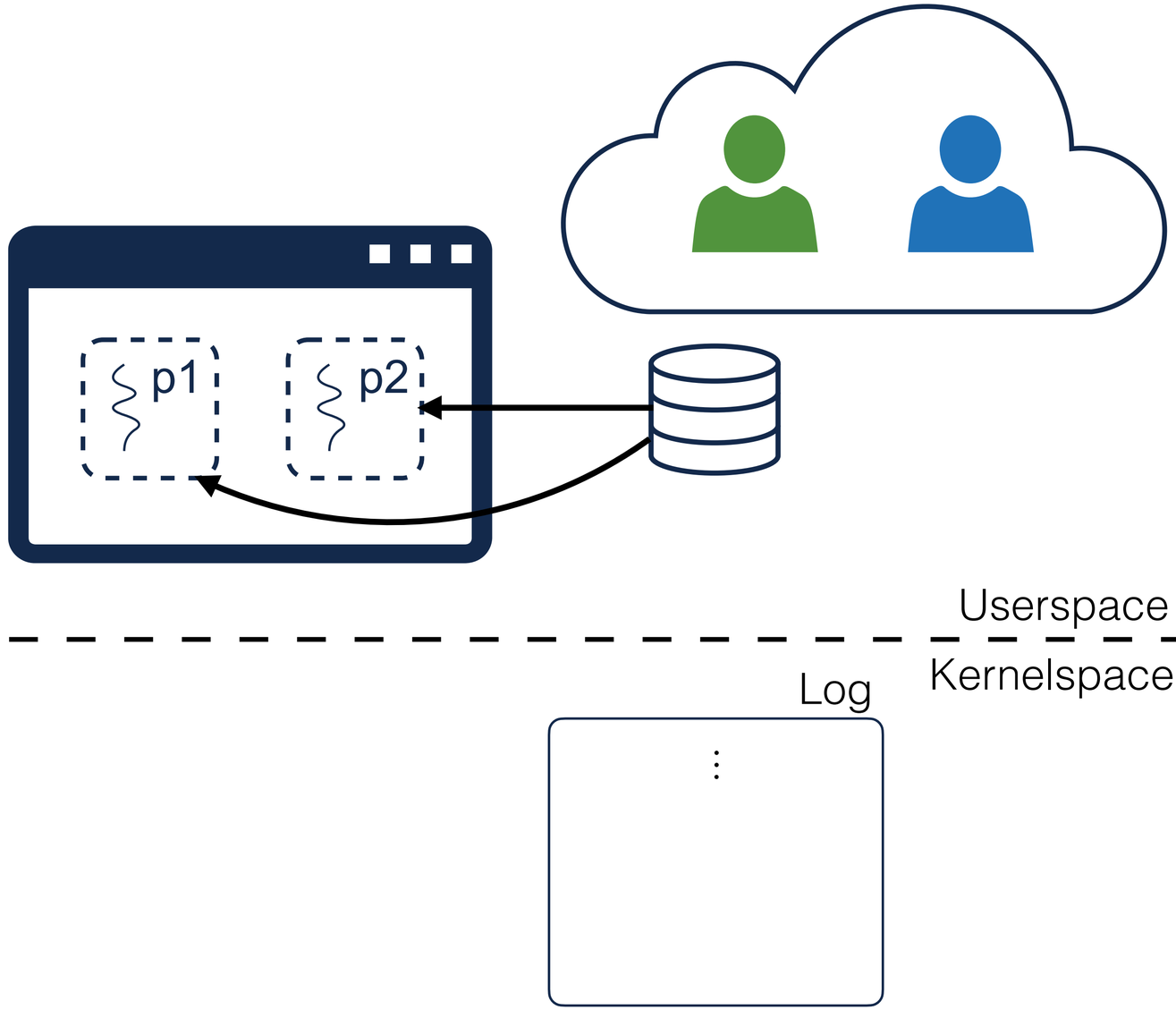
Devs Have Already Done (Some of) the Work

- Best practices dictate logging key events
- Logs contain application-level information
  - Including threading!
- Applications convey this to the reference monitor via write syscalls



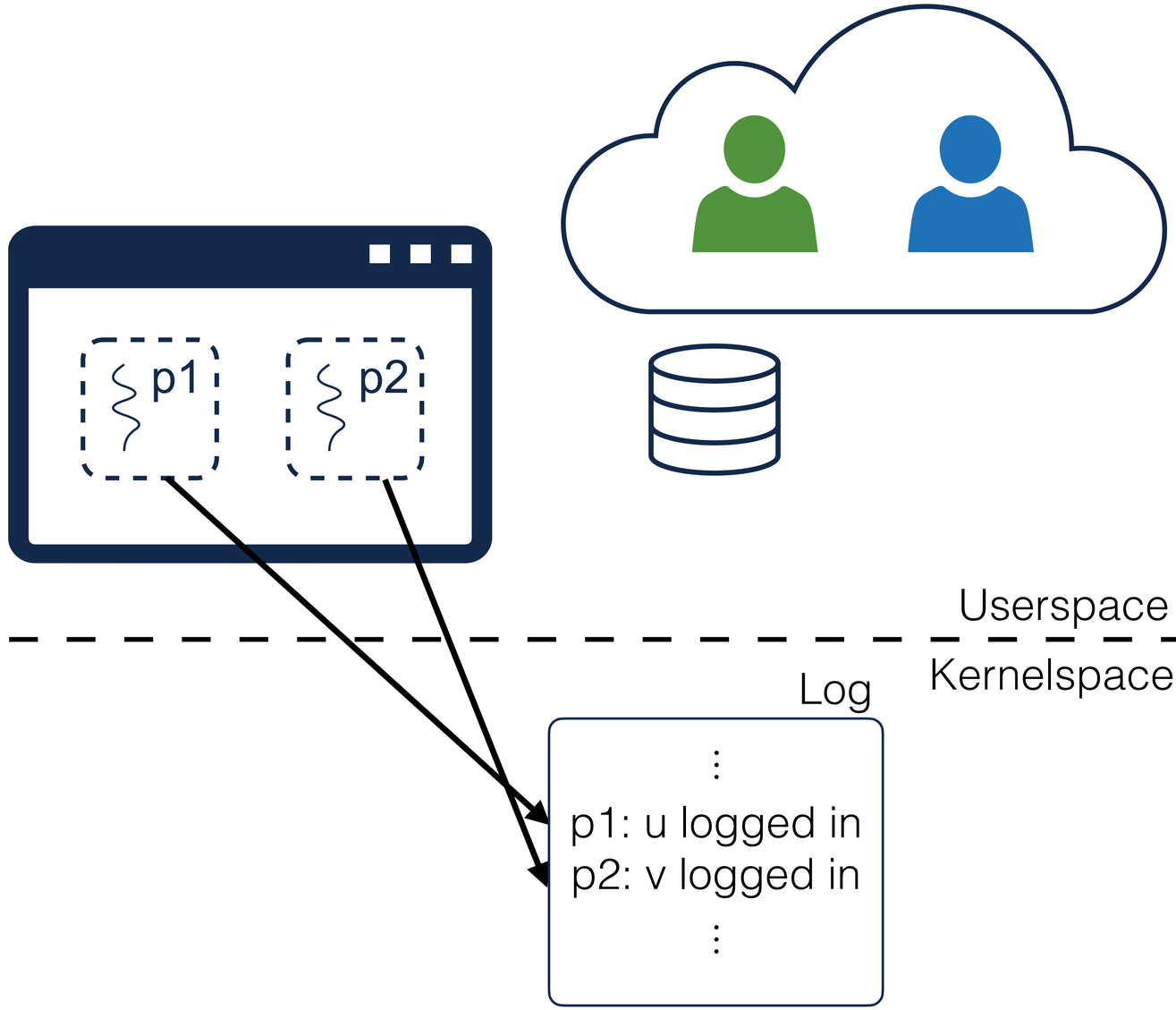
Devs Have Already Done (Some of) the Work

- Best practices dictate logging key events
- Logs contain application-level information
  - Including threading!
- Applications convey this to the reference monitor via write syscalls



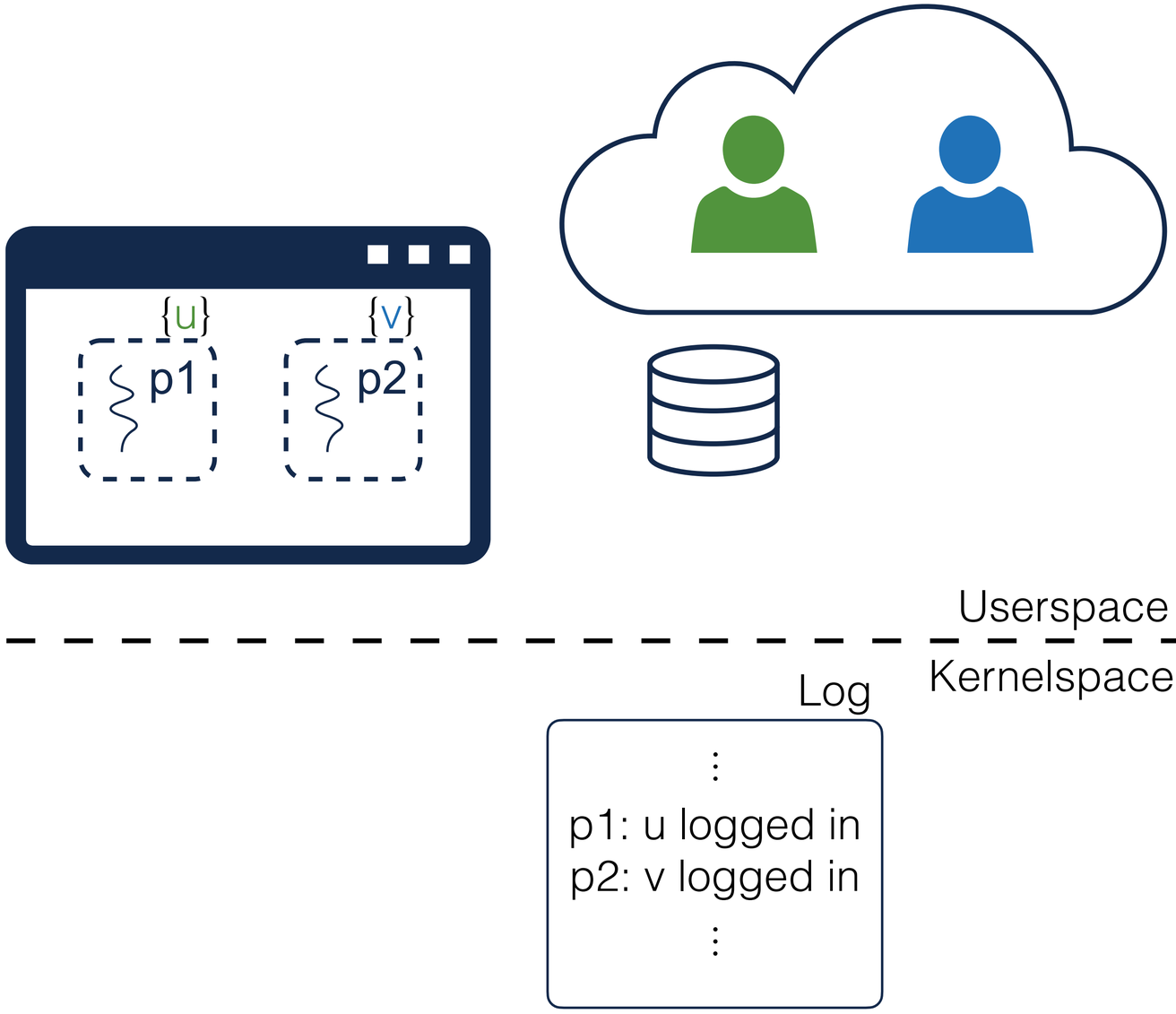
Devs Have Already Done (Some of) the Work

- Best practices dictate logging key events
- Logs contain application-level information
  - Including threading!
- Applications convey this to the reference monitor via write syscalls

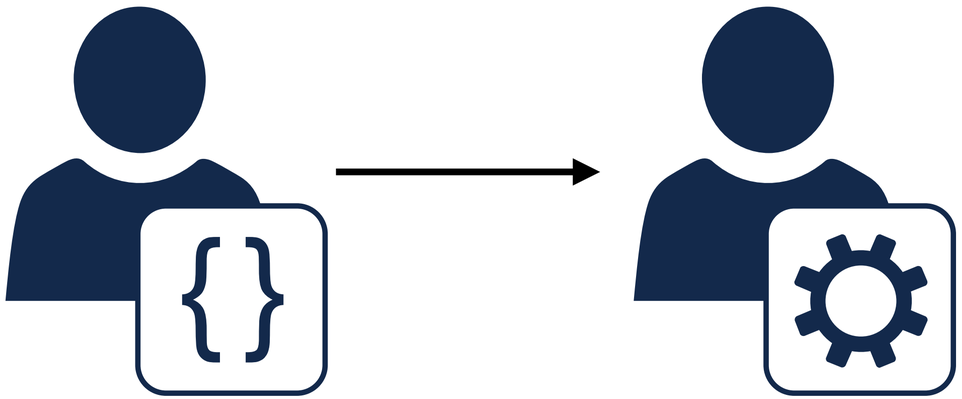


Devs Have Already Done (Some of) the Work

- Best practices dictate logging key events
- Logs contain application-level information
  - Including threading!
- Applications convey this to the reference monitor via write syscalls



From Devs to SysAdmins



- Only need to create policies for programs when required
- Can specialize policies to exact needs & deployments
- Easy to write policies spanning multiple programs

T-DIFC Overview

Userspace

Kernelspace

Log

Label Propagation & Permissions

Log Interception

T-DIFC Overview

Configure
External Policy

Userspace

Kernelspace

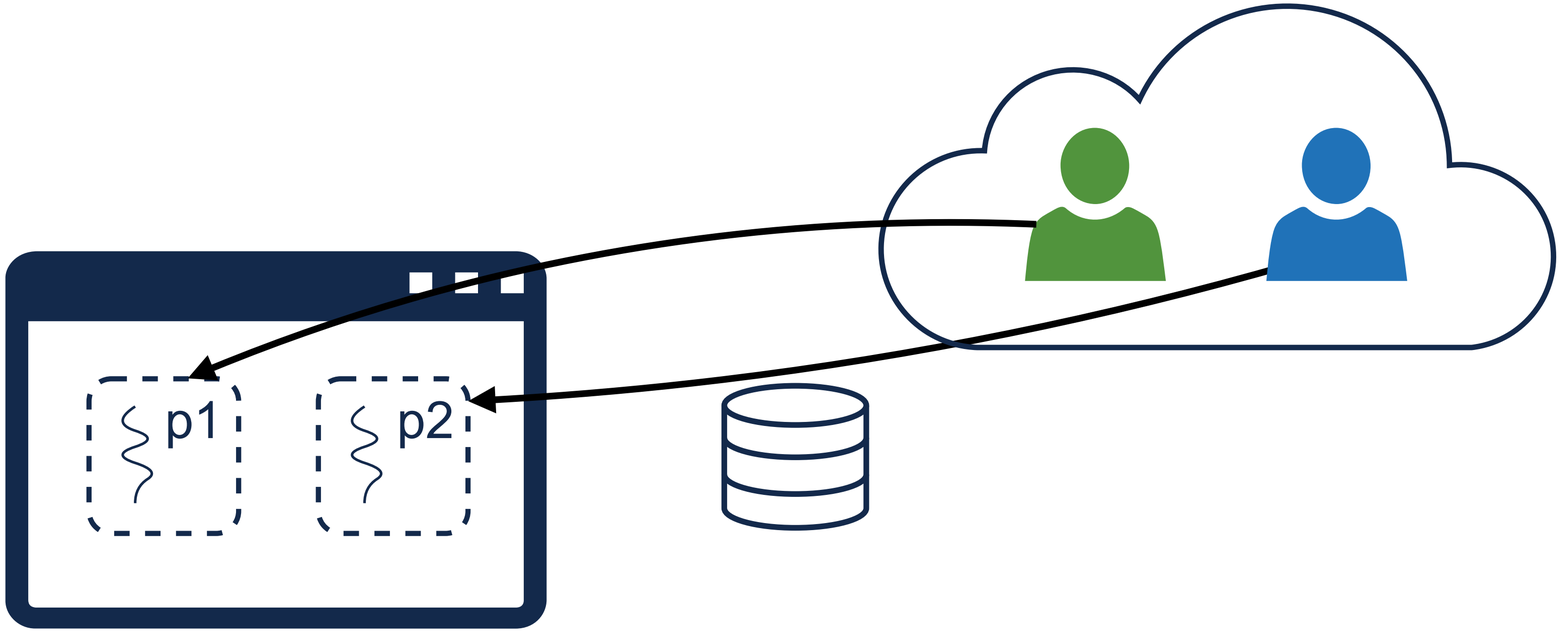
Log

Label
Propagation &
Permissions

Log
Interception

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

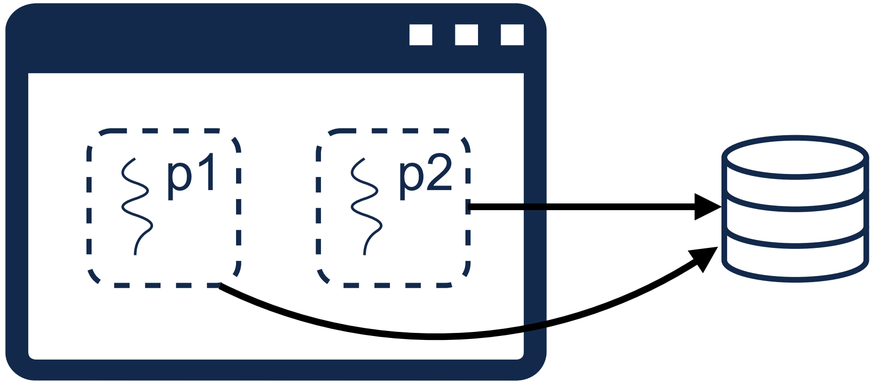
Log

Label
Propagation &
Permissions

Log
Interception

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

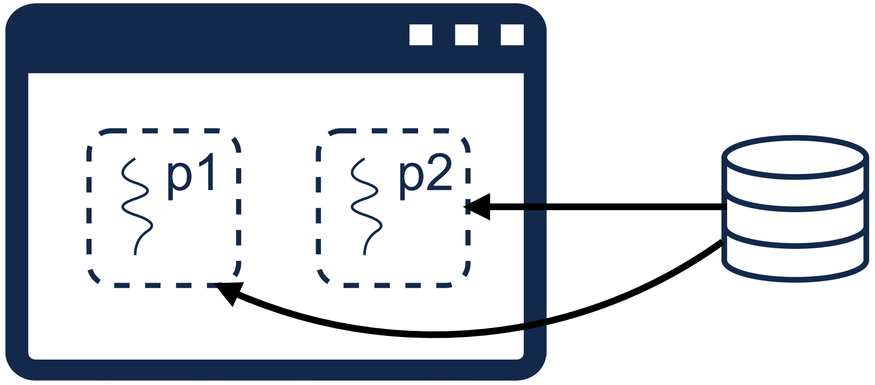
Log

Label
Propagation &
Permissions

Log
Interception

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

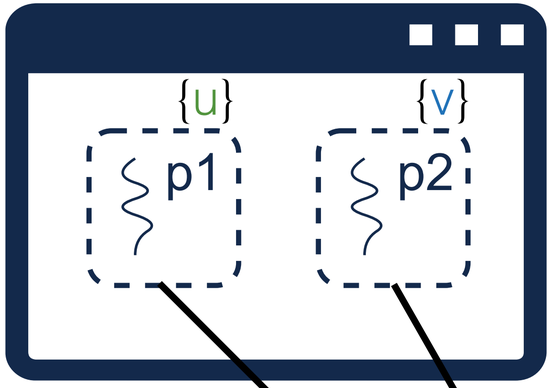
Log

Label
Propagation &
Permissions

Log
Interception

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

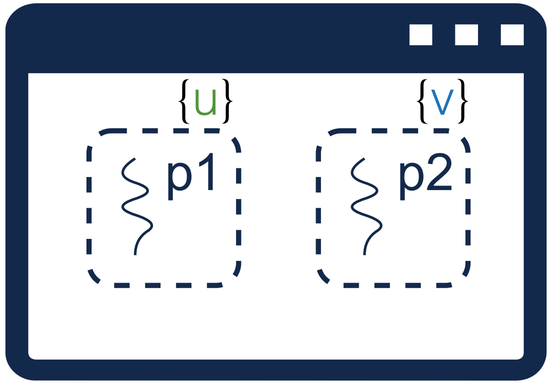
Log

Label
Propagation &
Permissions

Log
Interception

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

Label
Propagation &
Permissions

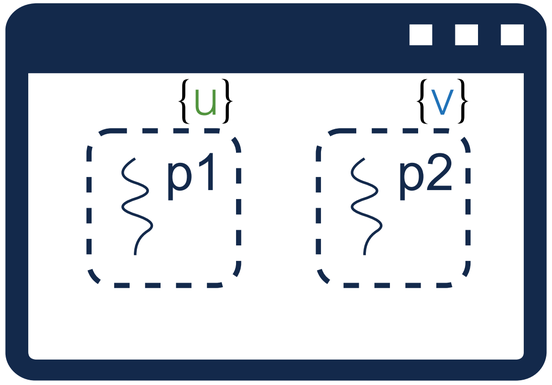
Log
Interception

Log

⋮
p1: u logged in
p2: v logged in
⋮

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

Label
Propagation &
Permissions

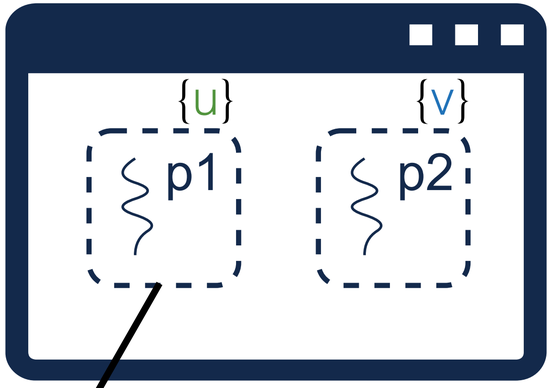
Log
Interception

Log

⋮
p1: u logged in
p2: v logged in
⋮

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

{v} {u}

Label
Propagation &
Permissions

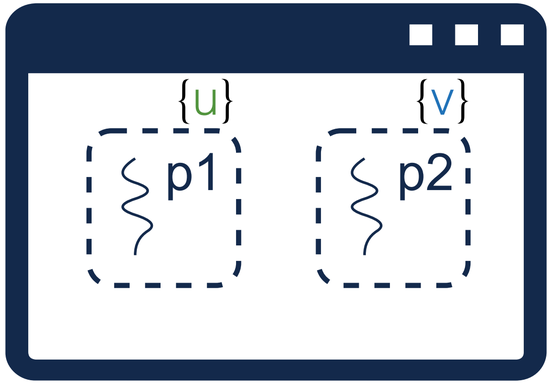
Log
Interception

Log

⋮
p1: u logged in
p2: v logged in
⋮

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

?

{u,v}

Label
Propagation &
Permissions

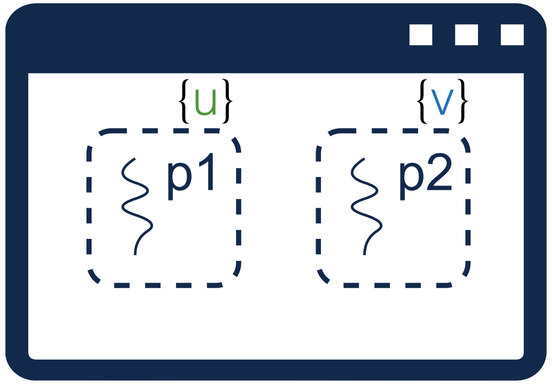
Log
Interception

Log

⋮
p1: u logged in
p2: v logged in
⋮

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

✗ {u,v}

Label
Propagation &
Permissions

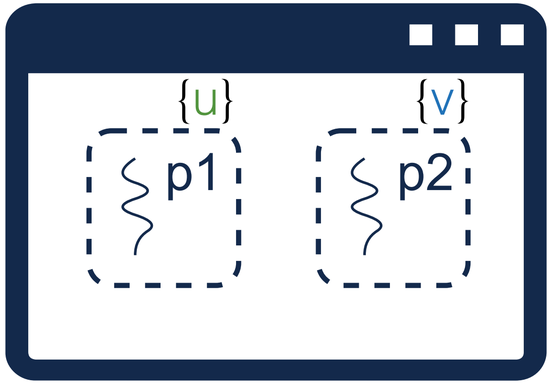
Log
Interception

Log

⋮
p1: u logged in
p2: v logged in
⋮

T-DIFC Overview

Configure
External Policy



Userspace

Kernelspace

✓

Label
Propagation &
Permissions

Log
Interception

Log

⋮
p1: u logged in
p2: v logged in
⋮

Labeling

```
match "<p[0-9]+>: <.+> logged in" {
   process <1> {
      settags tag(<2>);
   }
}
```

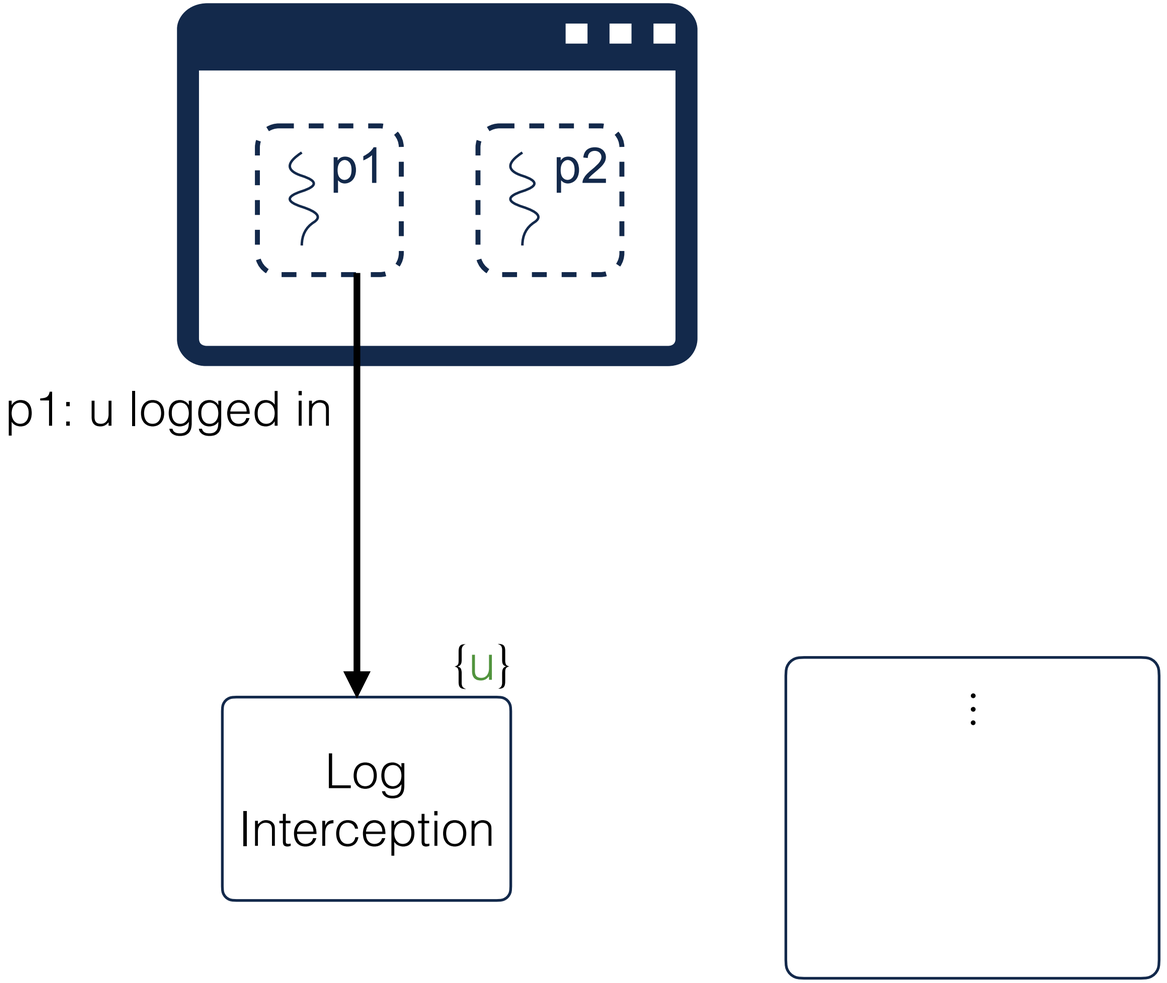
- Extract PID of current thread (`<p[0-9]+>`)
- Extract user as tag (`<.+>`)

Log Interception

⋮

Labeling

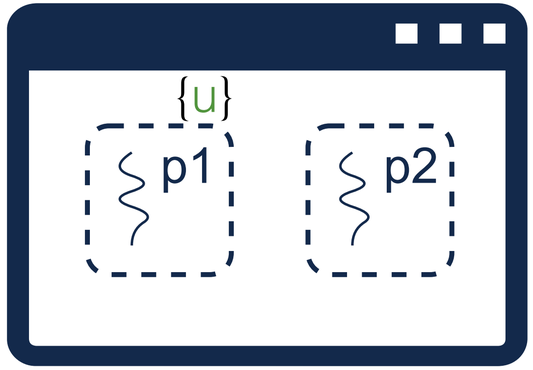
```
match "<p[0-9]+>: <.+> logged in" {
   process <1> {
      settags tag(<2>);
   }
}
```



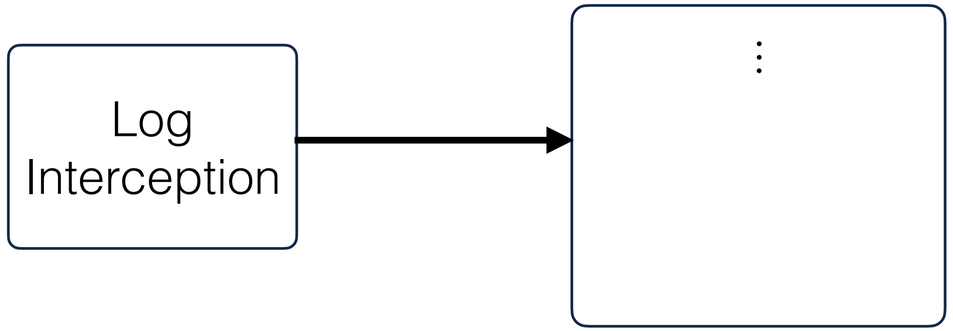
- Extract PID of current thread (`<p[0-9]+>`)
- Extract user as tag (`<.+>`)

Labeling

```
match "<p[0-9]+>: <.+> logged in" {
   process <1> {
      settags tag(<2>);
   }
}
```

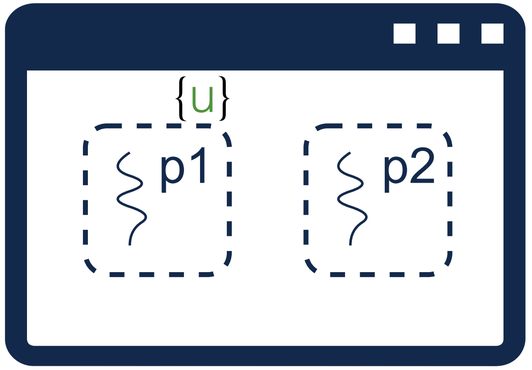


- Extract PID of current thread (`<p[0-9]+>`)
- Extract user as tag (`<.+>`)



Labeling

```
match "<p[0-9]+>: <.+> logged in" {
  process <1> {
     settags tag(<2>);
  }
}
```



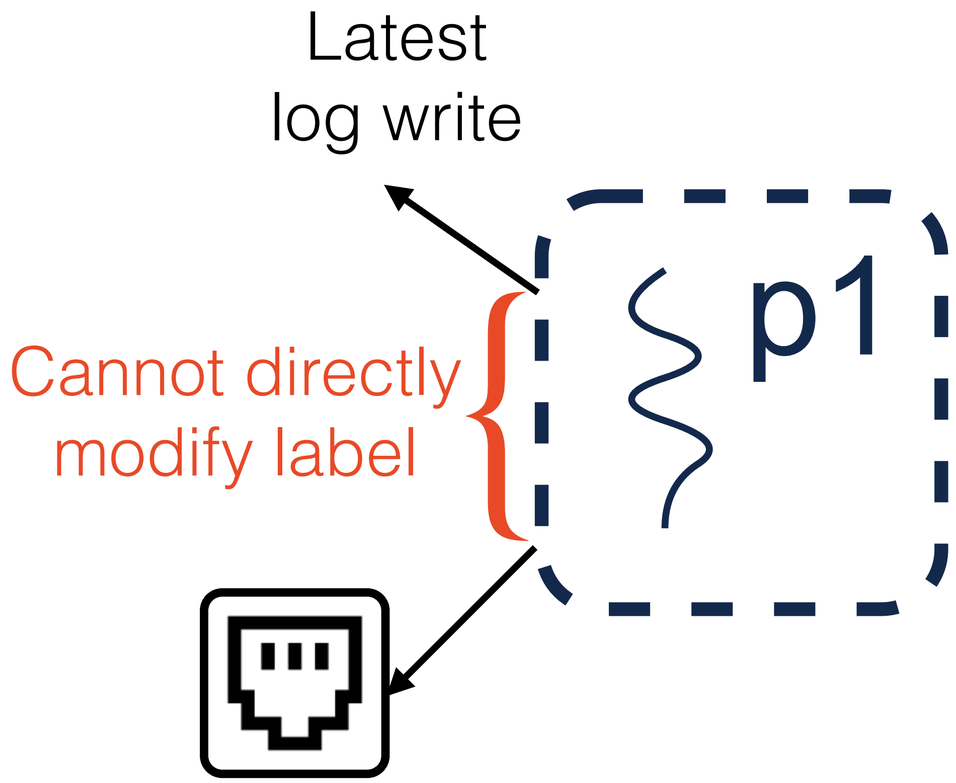
- Extract PID of current thread (`<p[0-9]+>`)
- Extract user as tag (`<.+>`)

Log Interception

⋮
p1: u logged in
⋮

Declassification

- Unlike existing systems, T-DIFC can only directly manipulate labeling state at **specific code locations** (logs)
- Allow specific implicit declassification using an **external policy rule**
  - Only declassify if label is small enough
  - Prevents smash-'n-grab attacks



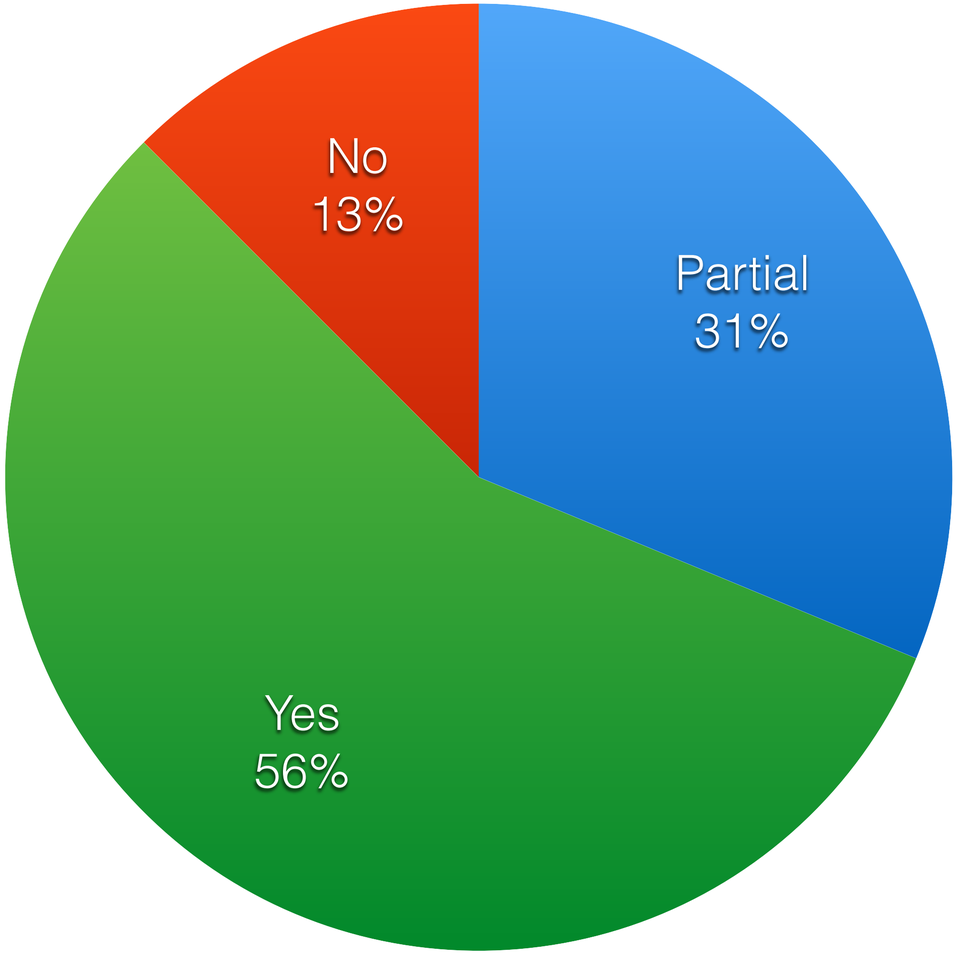
Evaluation Takeaways

Category	"Useful" Policy?	Sample Log
HTTP Servers	3/3	*host - user ...*
Other Servers	3/3	USER *user*: Login successful.
Load Balancing	0/1	Connect from *addr:port* to ...
Web Applications	2*/2	... Login for *user* succeeded ...
Databases	3*/3	... "get" "*key*"
Web Clients	3/4	Logging in as *user*... Logged in!

*Insufficient data partitioning

ProFTPD case study:
~154% overhead per log write
Negligible data transfer overhead

Surveyed Applications with Useful Policies



Open Challenges

- Difficulty in creating correct policies for a specific program
  - Translating high-level security policies to DIFC policies
  - Handling implementation details, e.g., fork before/after logging?
- Finer-grained partitioning for complex data structures (e.g., monolithic file-based or in-memory databases)
- Effectiveness on varying workloads

Conclusion

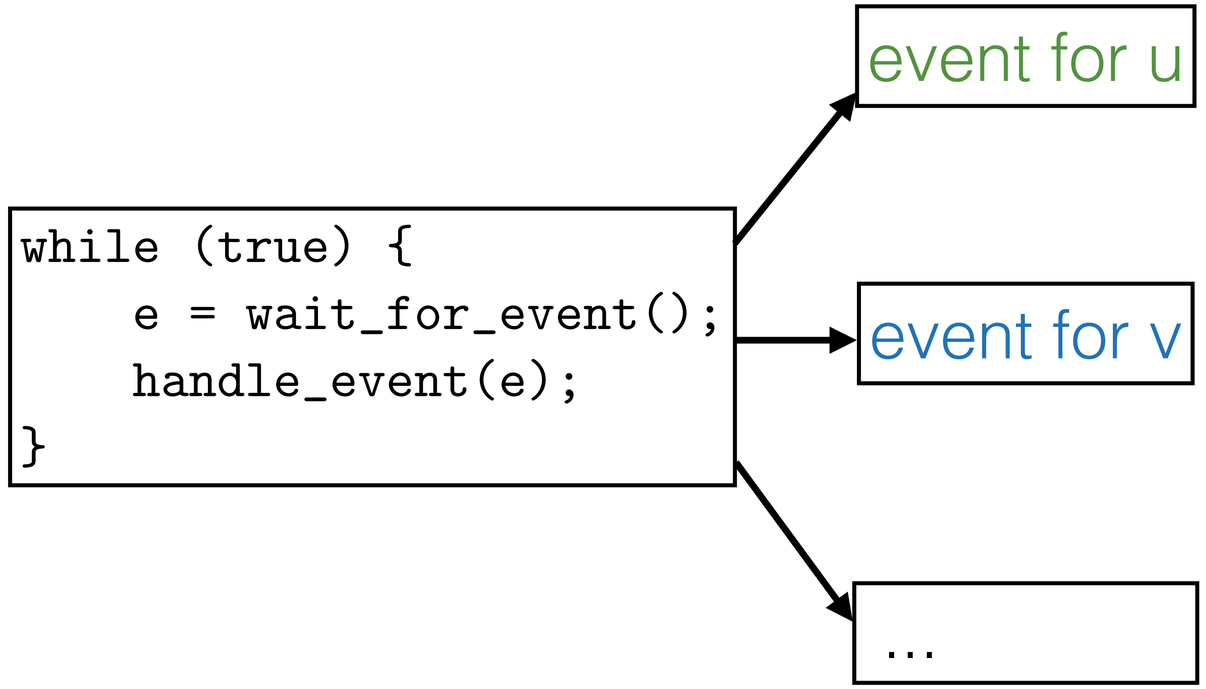
- One factor limiting DIFC is lacking compatibility with existing software
- Application logs can be used to partition processes and generate tags
  - We can express DIFC policies from most existing applications' logs
- We create T-DIFC, an OS-level DIFC system leveraging logs to achieve transparent DIFC

Thank you!

jdliu2@illinois.edu

Finer-Than-Process Granularity

- "Event process" abstraction from Asbestos [1]
- Can divide many servers into handlers for different user requests
  - Each handler is typically one iteration of an event-handling loop



[1] Efstathopoulos et al.. Labels and Events in the Asbestos Operating System. *SOSP '05.*

ProFTPD Example

```
id 21;
namespace unique;
logfile "/var/log/proftpd/proftpd.log";

max_process_label 1;

match ".*proftpd\[<[0-9]+>\].*:
    USER <[^:]+>: Login successful.\n" {
  process <1> {
    settags tag(<2>);
  }
}
```