# ReplicaWatcher: Training-less Anomaly Detection in Containerized Microservices

**Asbat El Khairi**, Marco Caselli, Andreas Peter, Andrea Continella


UNIVERSITY OF TWENTE.

Munich RE

SIEMENS

CARL VON OSSIETZKY UNIVERSITÄT OLDENBURG

# The Core Limitation of Anomaly Detection

- Dependence of baselines
- Baselines gradually age and lose effectiveness

# Amplifying the Challenge: Microservices

- Supports agile development and continuous updates

# Amplifying the Challenge: Microservices

- Supports agile development and continuous updates
- Frequent **normality shifts**


Netflix makes **hundreds** of production changes to their microservices **per day.**

# Normality Shift in System Calls

# Normality Shift in System Calls

```php
<?php
$end_time = time() + 600;
while (time() < $end_time) {
   echo "Current time: " . date('H:i:s');
   sleep(10); }
?>
```

# Normality Shift in System Calls

```php
<?php
$end_time = time() + 600;
while (time() < $end_time) {
   echo "Current time: " . date('H:i:s');
   sleep(10); }
?>
```

Dockerize the app

```
FROM php:8.1.18-apache-buster
COPY script.php /var/www/html/script.php
CMD ["php", "/var/www/html/script.php"]
```

# Normality Shift in System Calls

```php
<?php
$end_time = time() + 600;
while (time() < $end_time) {
   echo "Current time: " . date('H:i:s');
   sleep(10); }
?>
```

## Dockerize the app

```
FROM php:8.1.18-apache-buster
COPY script.php /var/www/html/script.php
CMD ["php", "/var/www/html/script.php"]
```

## Monitor at Runtime

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > nanosleep interval=10000000000(10s)
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

# Normality Shift in System Calls

```php
<?php
$end_time = time() + 600;
while (time() < $end_time) {
   echo "Current time: " . date('H:i:s');
   sleep(10); }
?>
```

Dockerize the app

```
FROM php:8.1.18-apache-buster
COPY script.php /var/www/html/script.php
CMD ["php", "/var/www/html/script.php"]
```

Baseline

| Normal Syscalls |
| --- |
| **write**<br>**write**<br>**nanosleep**<br>**ppoll** |

Monitor at Runtime

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > nanosleep interval=10000000000(10s)
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

# Normality Shift in System Calls

```php
<?php
$end_time = time() + 600;
while (time() < $end_time) {
   echo "Current time: " . date('H:i:s');
   sleep(10); }
?>
```

Pre-update

```
FROM php:8.1.18-apache-buster
COPY script.php /var/www/html/script.php
CMD ["php", "/var/www/html/script.php"]
```

Post-update

```
FROM php:8.1.18-apache-bullseye
COPY script.php /var/www/html/script.php
CMD ["php", "/var/www/html/script.php"]
```

# Normality Shift in System Calls

Pre-update

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > nanosleep interval=10000000000(10s)
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

Post-update

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > clock_nanosleep
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

# Normality Shift in System Calls

Pre-update

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > nanosleep interval=10000000000(10s)
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

Post-update

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > clock_nanosleep
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

# Normality Shift in System Calls

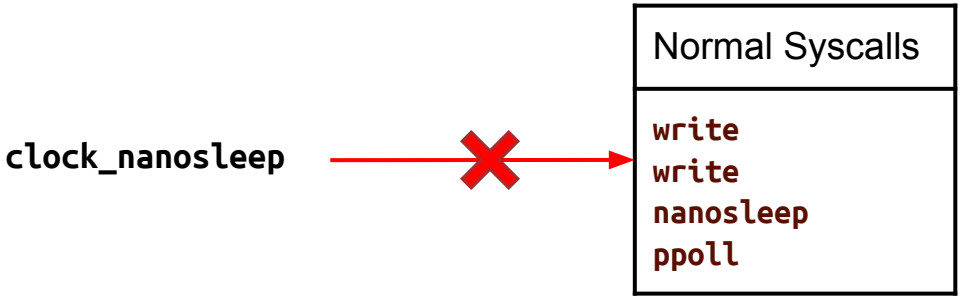
Pre-update

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > nanosleep interval=10000000000(10s)
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

Post-update

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > clock_nanosleep
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

Baseline



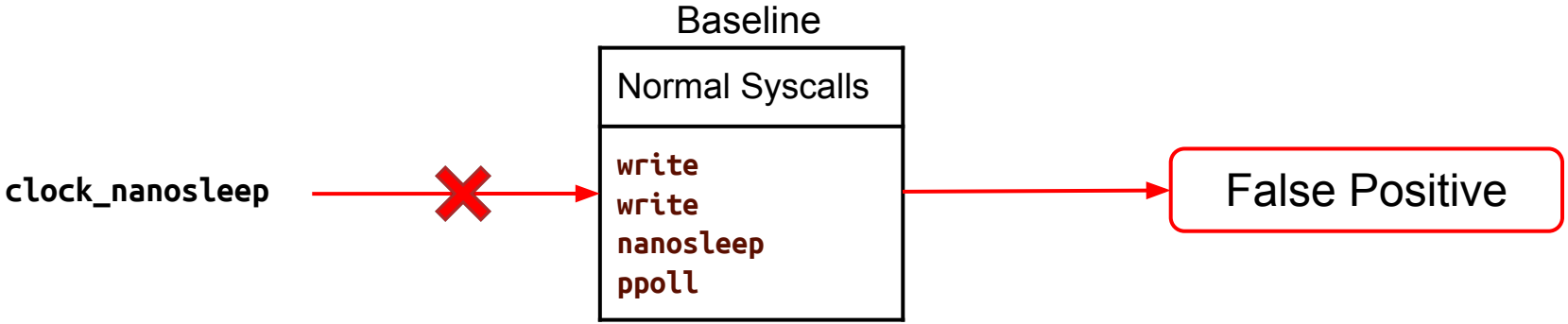
# Normality Shift in System Calls

Pre-update

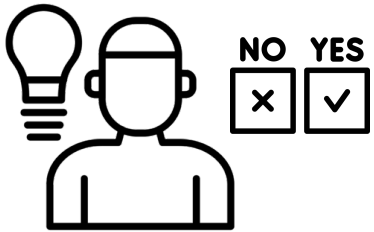
```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > nanosleep interval=10000000000(10s)
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

Post-update

```
Process </> Syscall Args
...
php > write fd=1(<p>pipe:[843685]) size=34
php < write res=34 data=Current time: ..
php > clock_nanosleep
php > ppoll next=0 pgft_maj=213 pgft_min=2634
...
```

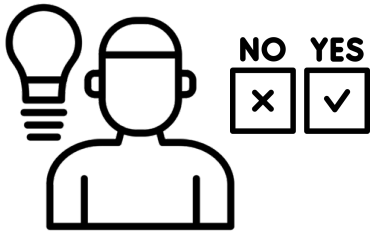


# Retraining Challenges

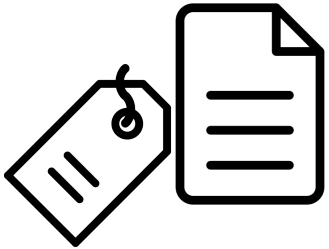



Necessity for Security Expertise

# Retraining Challenges

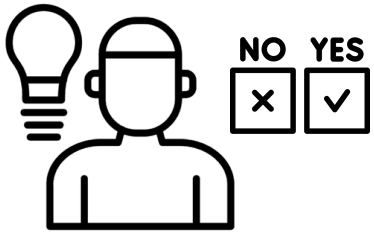



Necessity for Security Expertise

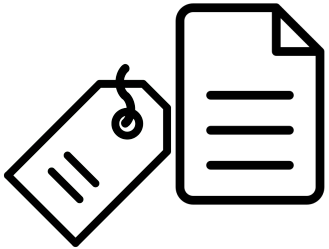


Data Labeling

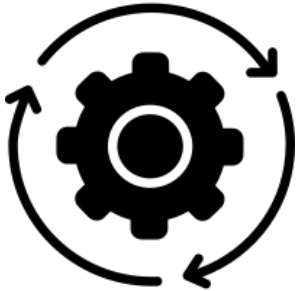
# Retraining Challenges




Necessity for Security Expertise

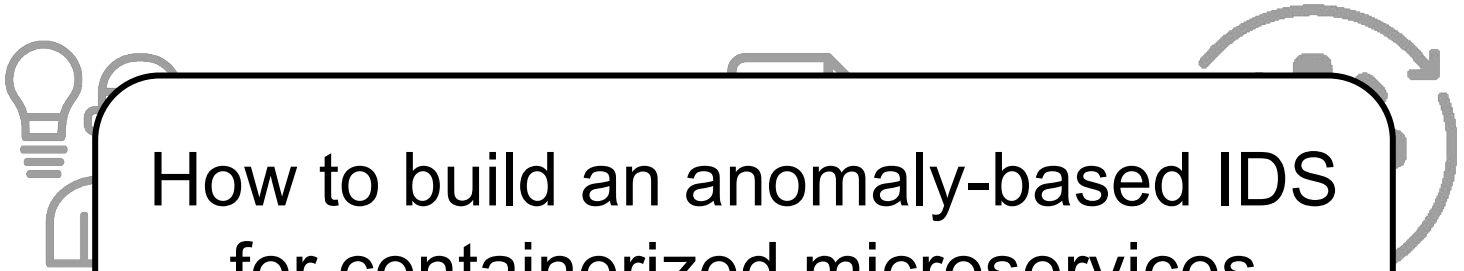


Data Labeling



Frequent Retraining

# Retraining Challenges

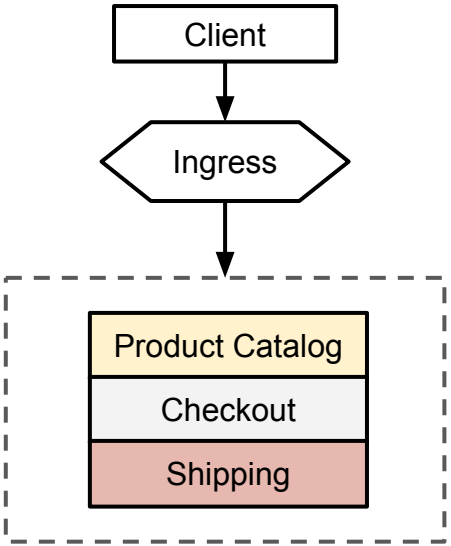


Necessity for aining

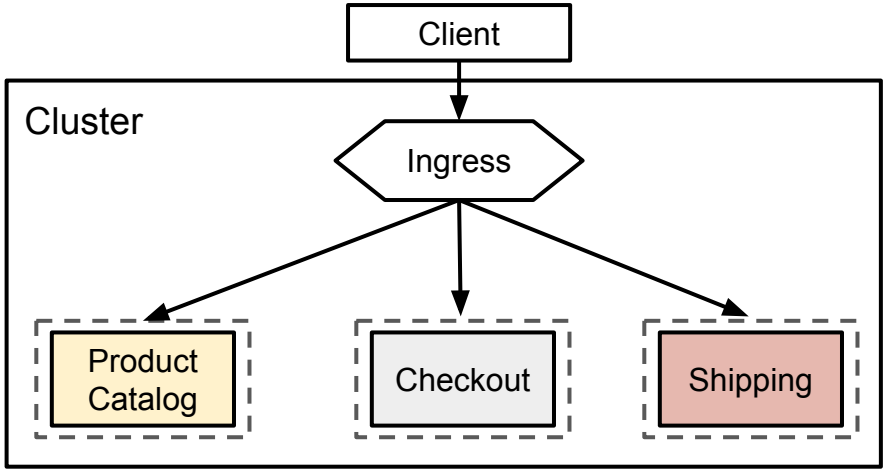
How to build an anomaly-based IDS for containerized microservices **without training**?

# Separation of Concerns

- Containerized microservices are designed for **specific** and **narrow** tasks



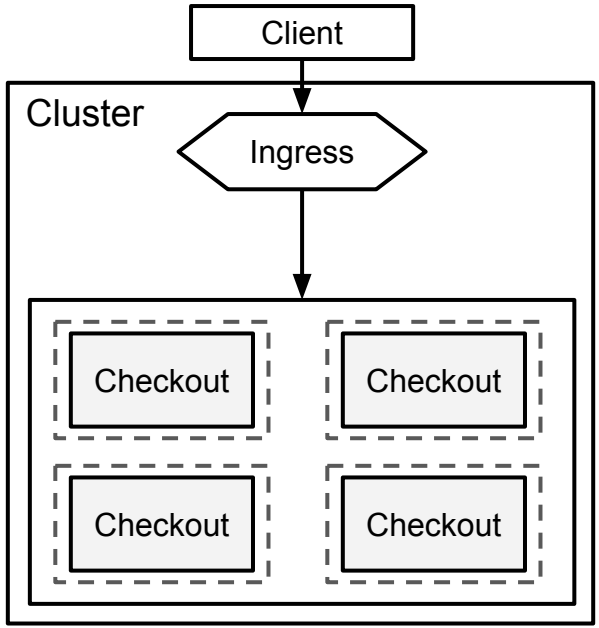

Monolithic



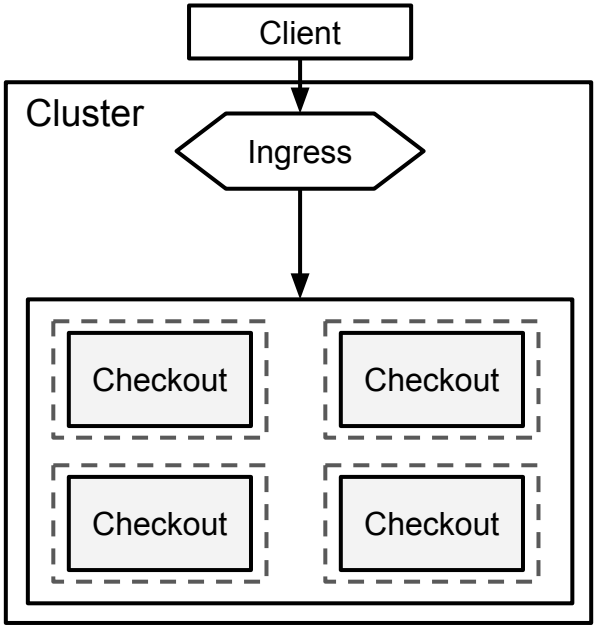

Microservices

# Replication

- Orchestration platforms (e.g., Kubernetes) provide **replicas** for fault tolerance or scalability reasons

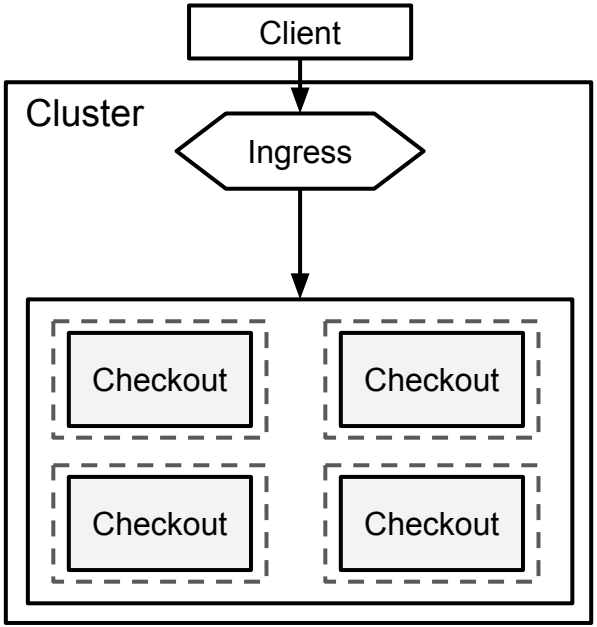


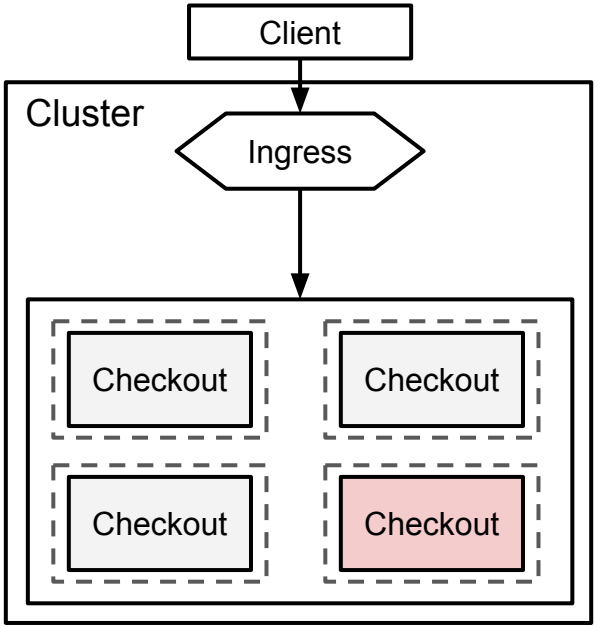
# Intuition




Consistent behavior

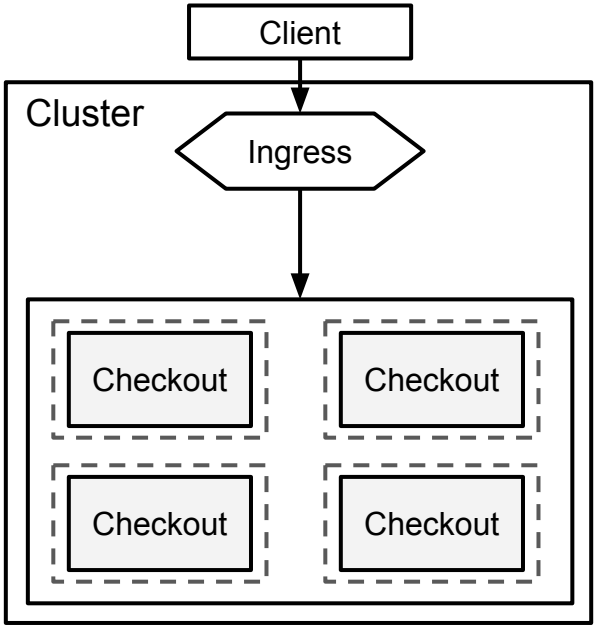
# Intuition



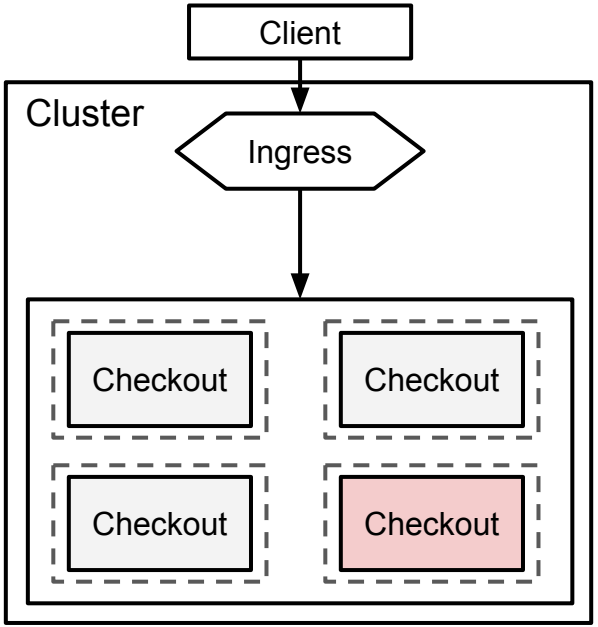




Consistent behavior

# Intuition

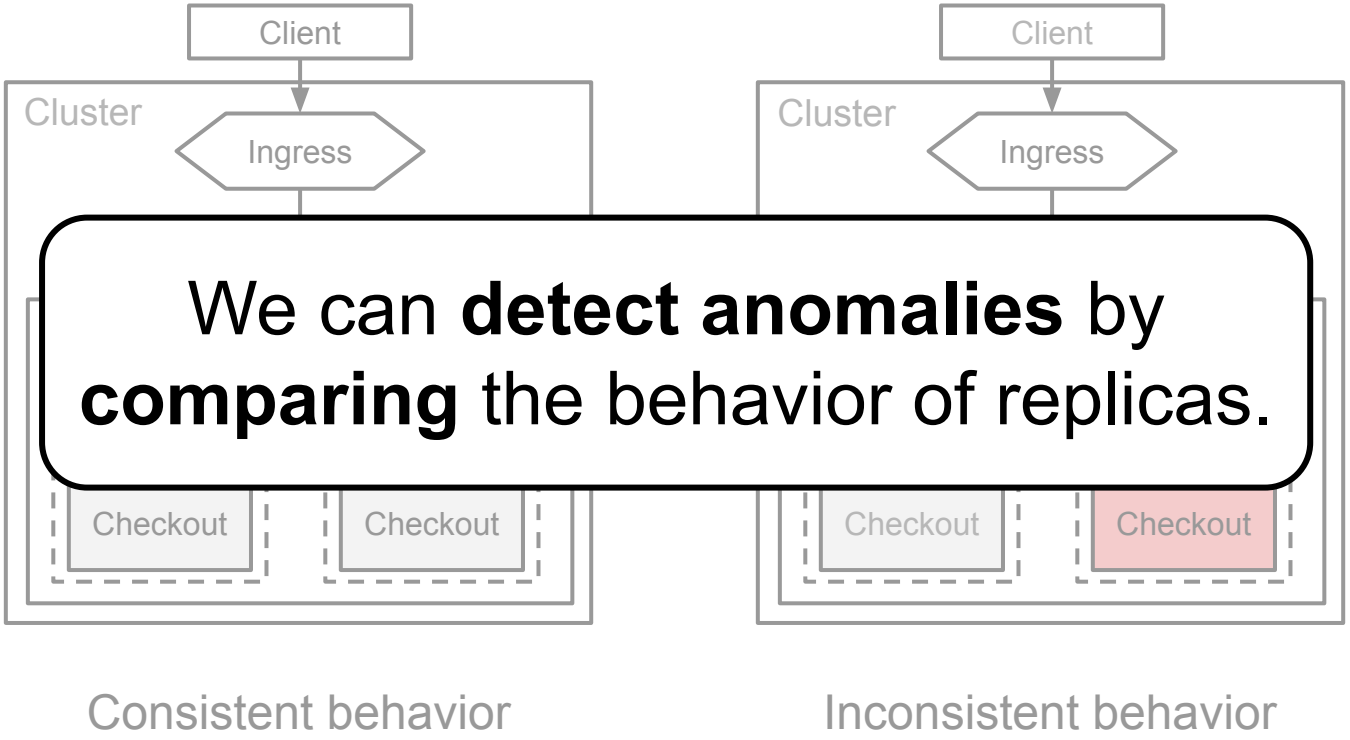



Consistent behavior




Inconsistent behavior

# Intuition


Client
Cluster
Ingress
Checkout
Checkout
Client
Cluster
Ingress
Checkout
Checkout
We can **detect anomalies** by **comparing** the behavior of replicas.
Consistent behavior
Inconsistent behavior

# Not so easy …

# Not so easy …

- Background noise
  - Different traffic loads
  - Differences in user inputs
  - Resource contention

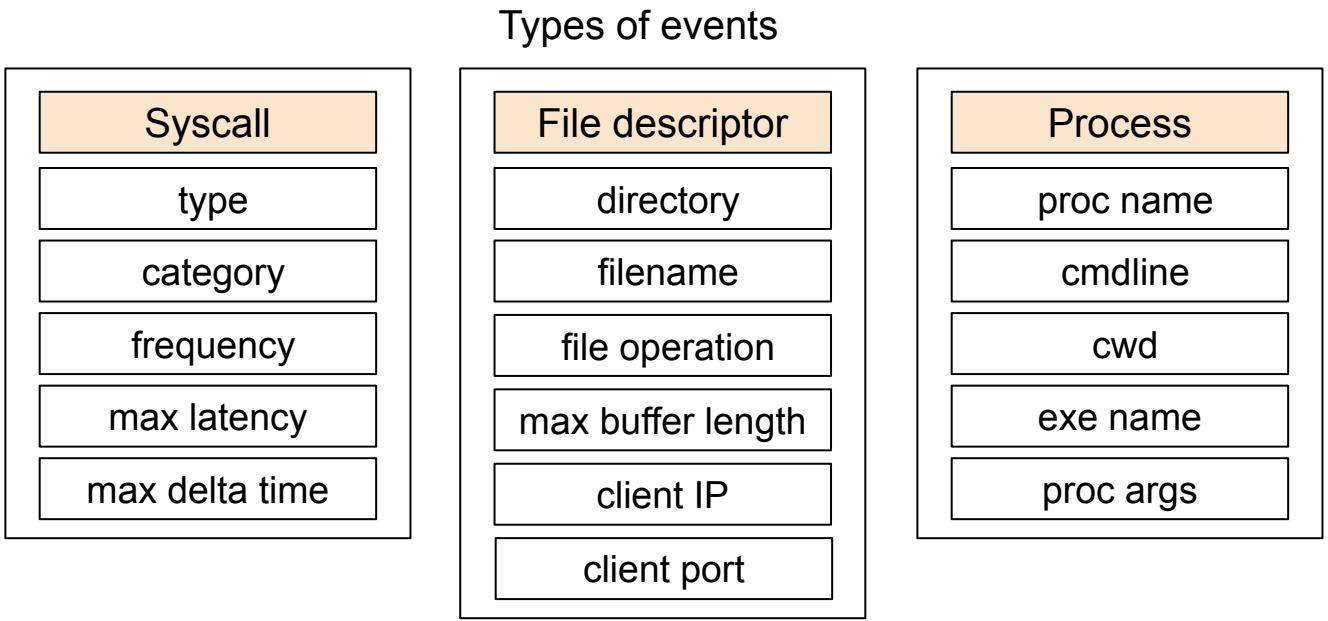
We found that background noise can be controlled, making comparison between replicas possible.

# Controlling Background Noise

- Select observables (i.e., kernel events) that
  - Show less sensitivity to noise

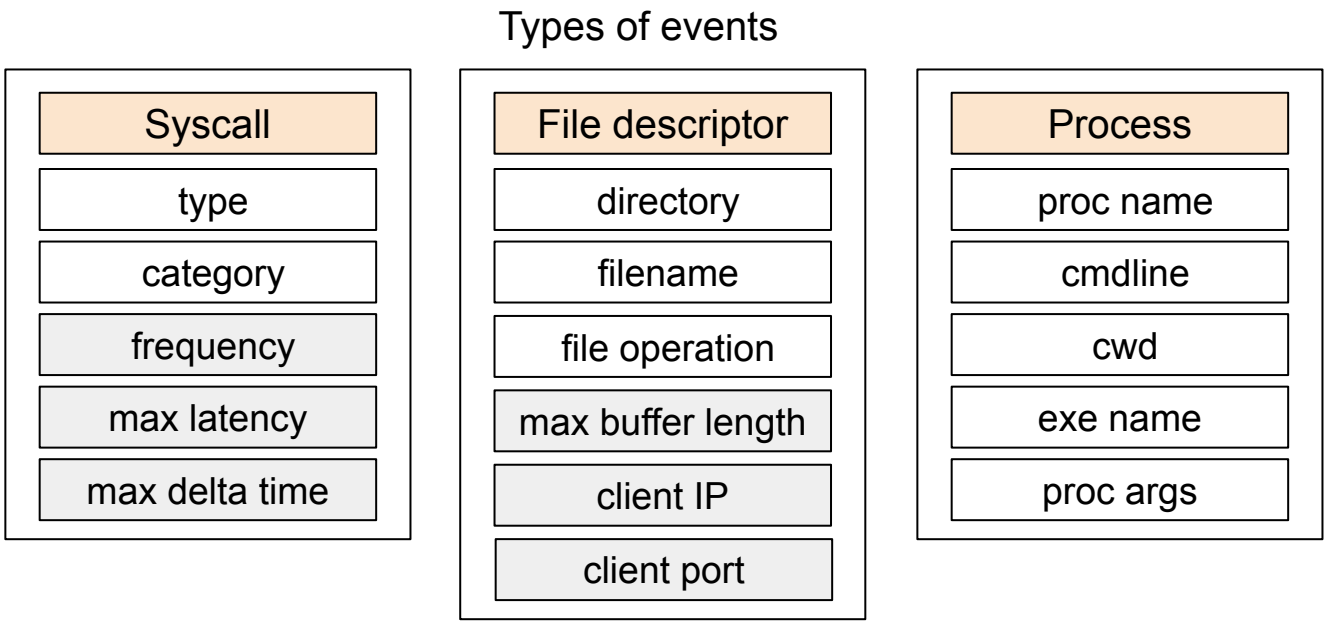
# Controlling Background Noise

- Select observables (i.e., kernel events) that
  - Show less sensitivity to noise



# Controlling Background Noise

- Preliminary assessment results



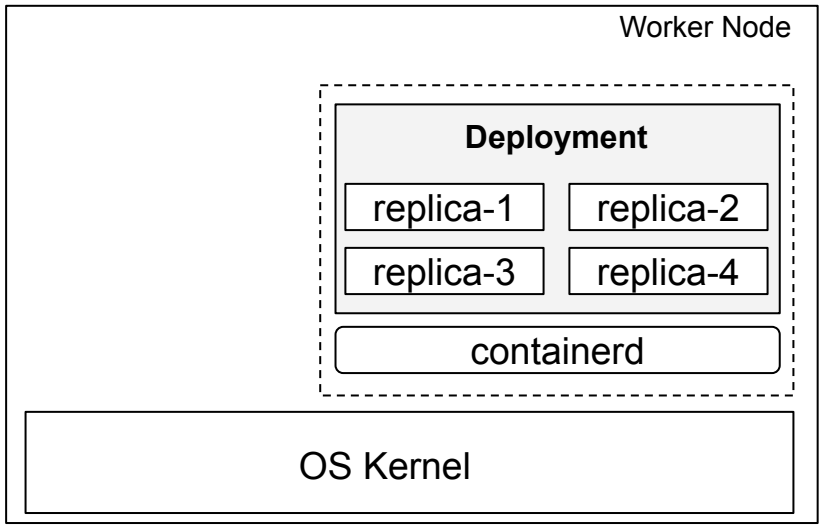
# Controlling Background Noise

- Select optimized comparison interval that
  - Promotes consistent behavior across replicas
  - Flattens out outliers

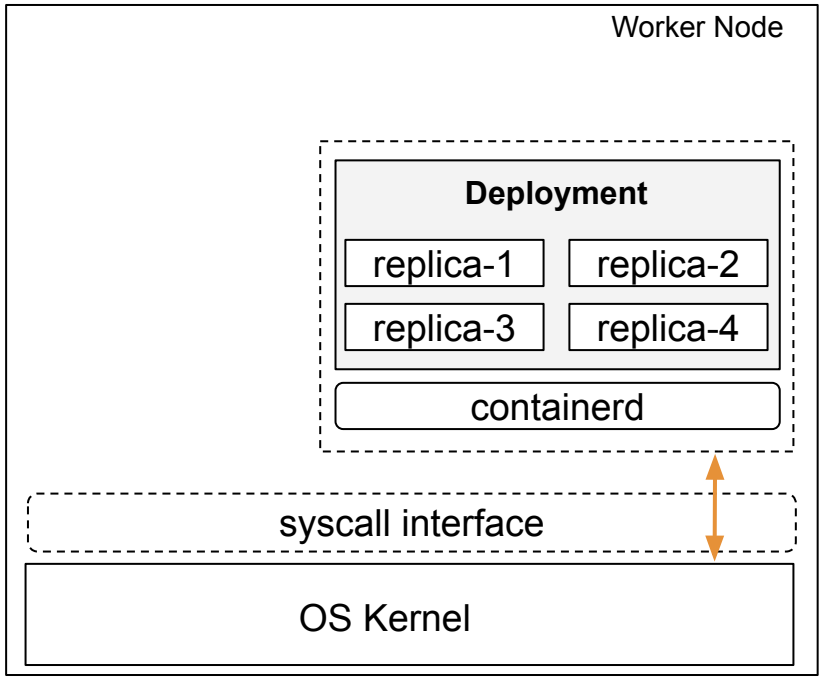
# Controlling Background Noise

- Select optimized comparison interval that
  - Promotes consistent behavior across replicas
  - Flattens out outliers
- Preliminary assessment results
  - Short intervals – low consistency
  - Long intervals  – high consistency
  - 30-second interval allows for consistent behavior

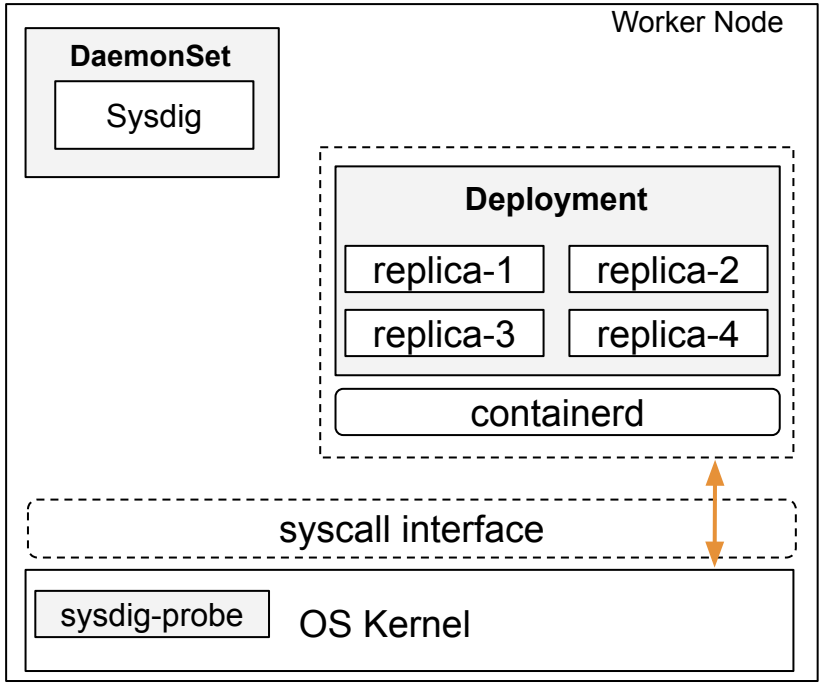
# ReplicaWatcher - Overview



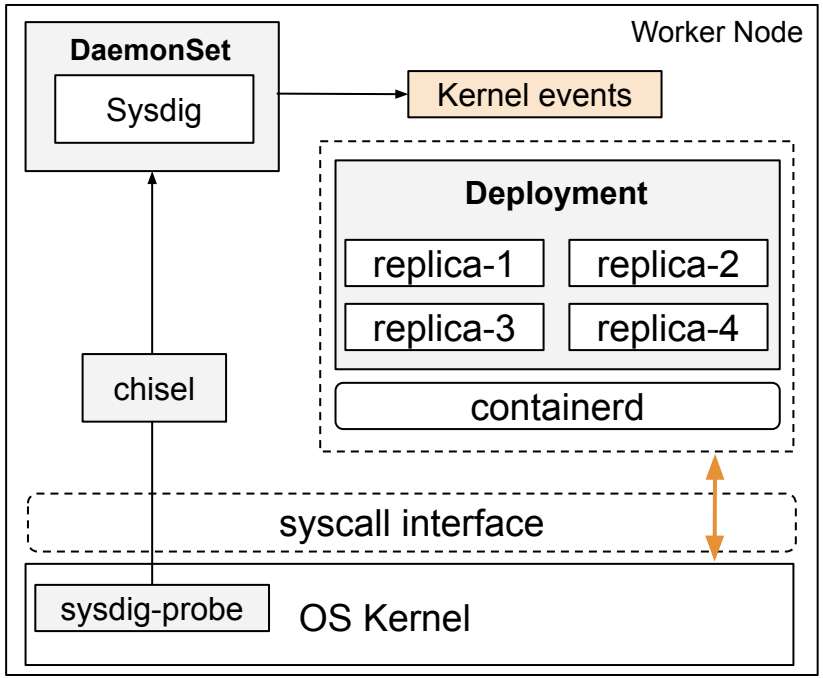
# ReplicaWatcher - Overview



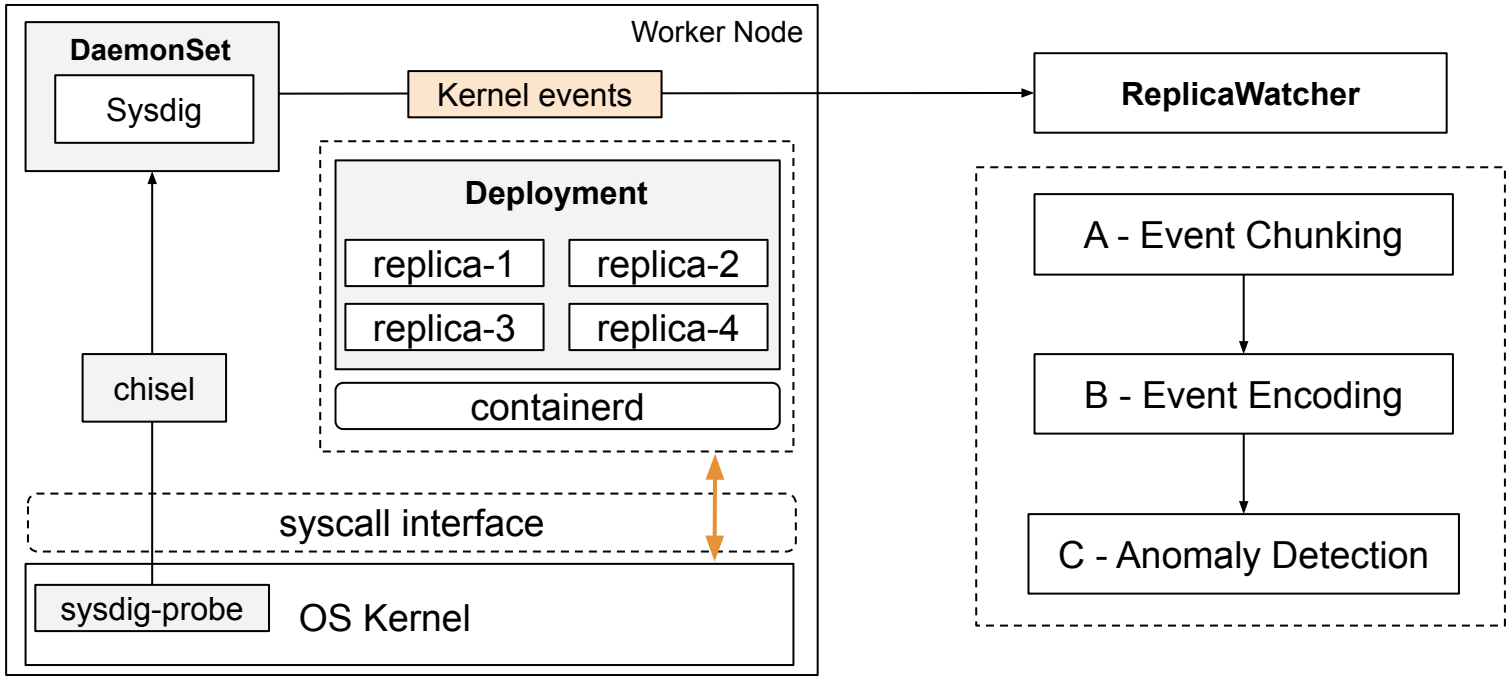
# ReplicaWatcher - Overview



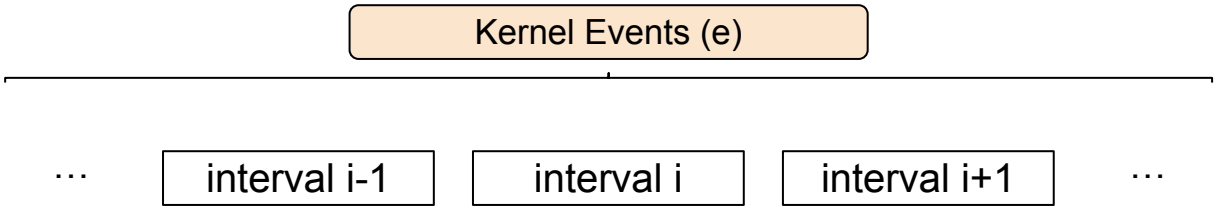
# ReplicaWatcher - Overview



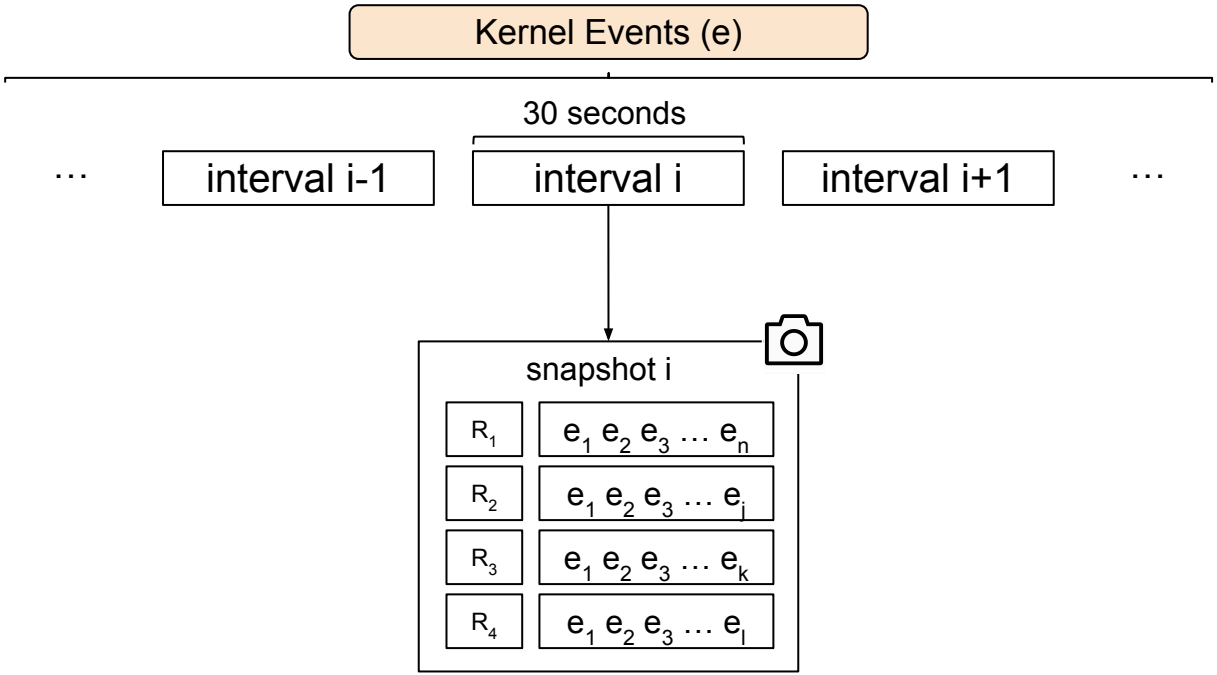
# ReplicaWatcher - Overview



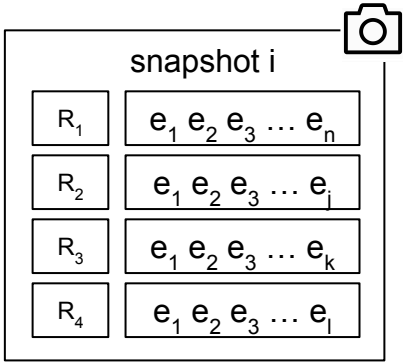
# Event Chunking



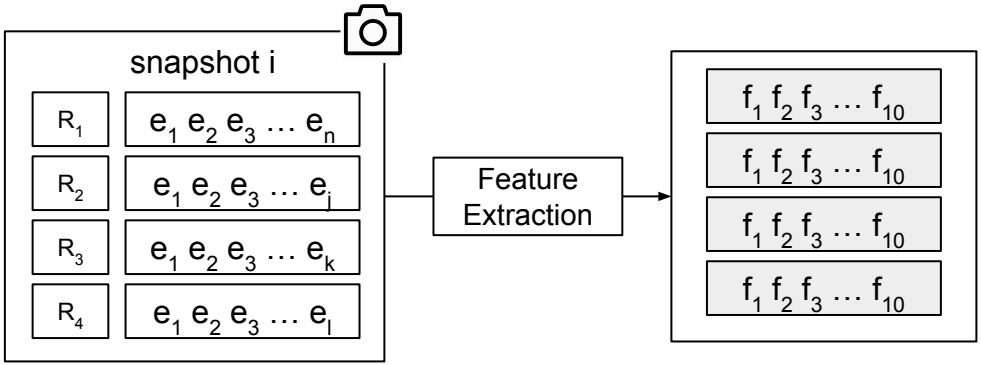
# Event Chunking



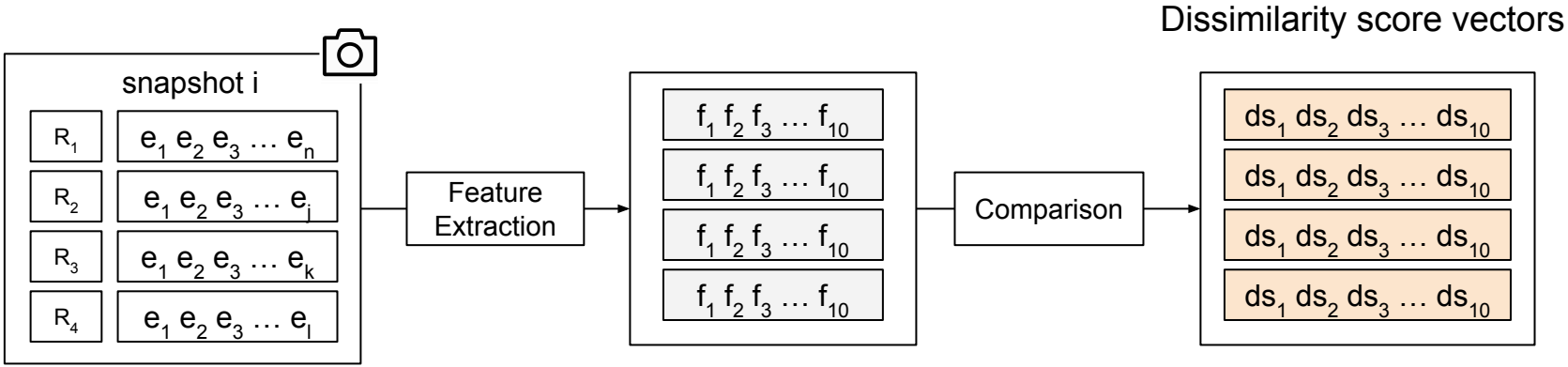
# Event Encoding



# Event Encoding



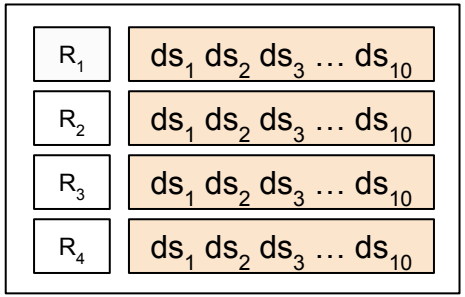
# Event Encoding




$$DS_{f_k}(R_i) = 1 - \frac{1}{n-1} \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{|R_{i_{f_k}} \cap R_{j_{f_k}}|}{|R_{i_{f_k}} \cup R_{j_{f_k}}|}$$

# Anomaly Detection

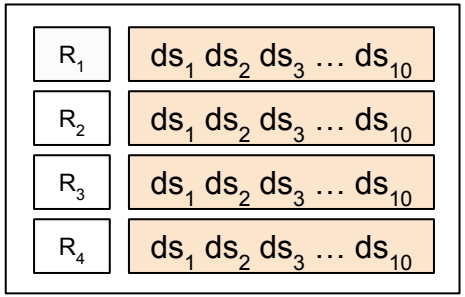
Dissimilarity score vectors



# Anomaly Detection

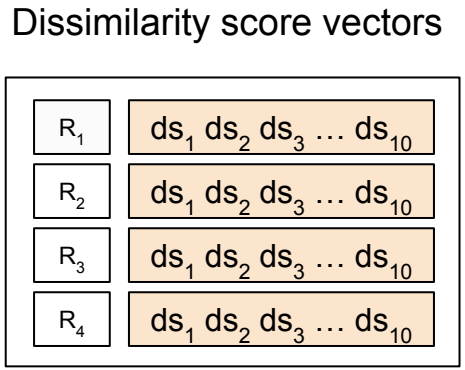
- Replicas should behave **the same**

Dissimilarity score vectors



# Anomaly Detection

- Replicas should behave **the same**
- Allow **minor** differences in behavior

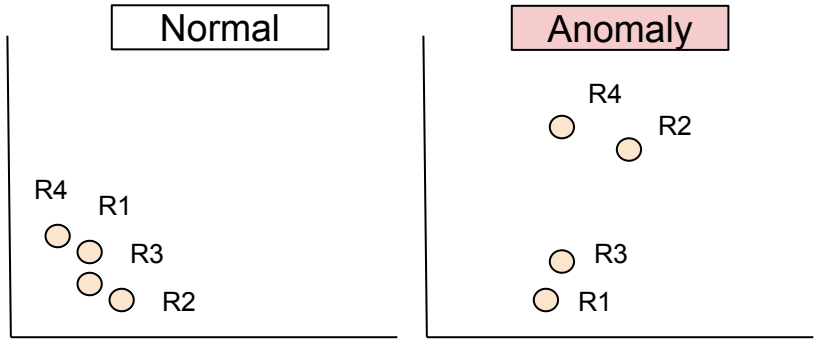


# Anomaly Detection

- Replicas should behave **the same**
- Allow **minor** differences in behavior
- Replicas' vectors should **converge** to **origin**

Dissimilarity score vectors

| | |
|---|---|
| $R_1$ | $ds_1\ ds_2\ ds_3\ \ldots\ ds_{10}$ |
| $R_2$ | $ds_1\ ds_2\ ds_3\ \ldots\ ds_{10}$ |
| $R_3$ | $ds_1\ ds_2\ ds_3\ \ldots\ ds_{10}$ |
| $R_4$ | $ds_1\ ds_2\ ds_3\ \ldots\ ds_{10}$ |

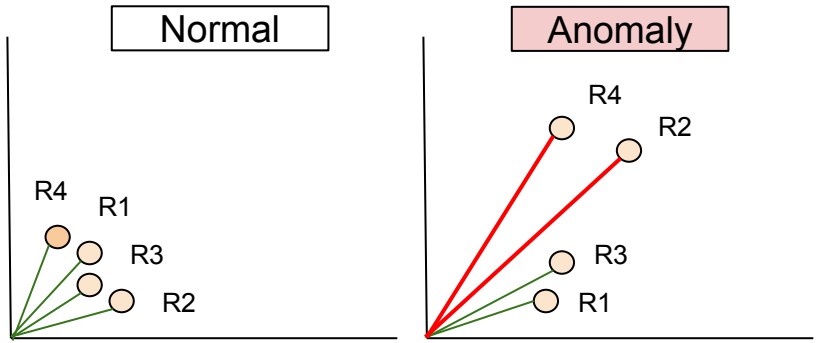


# Anomaly Detection

- Replicas should behave **the same**
- Allow **minor** differences in behavior
- Replicas' vectors should **converge** to **origin**
- Alarm if distance from origin **passes** threshold

Dissimilarity score vectors

| | |
|---|---|
| $R_1$ | $ds_1\ ds_2\ ds_3\ \ldots\ ds_{10}$ |
| $R_2$ | $ds_1\ ds_2\ ds_3\ \ldots\ ds_{10}$ |
| $R_3$ | $ds_1\ ds_2\ ds_3\ \ldots\ ds_{10}$ |
| $R_4$ | $ds_1\ ds_2\ ds_3\ \ldots\ ds_{10}$ |



# Evaluation

- How well does ReplicaWatcher work?

# Evaluation

- How well does ReplicaWatcher work?
  - Showing resilience to normality shifts

# Evaluation

- How well does ReplicaWatcher work?
  - Showing resilience to normality shifts
  - Detecting anomalies with minimal false positives

# Evaluation

- How well does ReplicaWatcher work?
  - Showing resilience to normality shifts
  - Detecting anomalies with minimal false positives
- Dataset
  - Tested on two e-commerce platforms (Google Online Boutique, Homebrew)

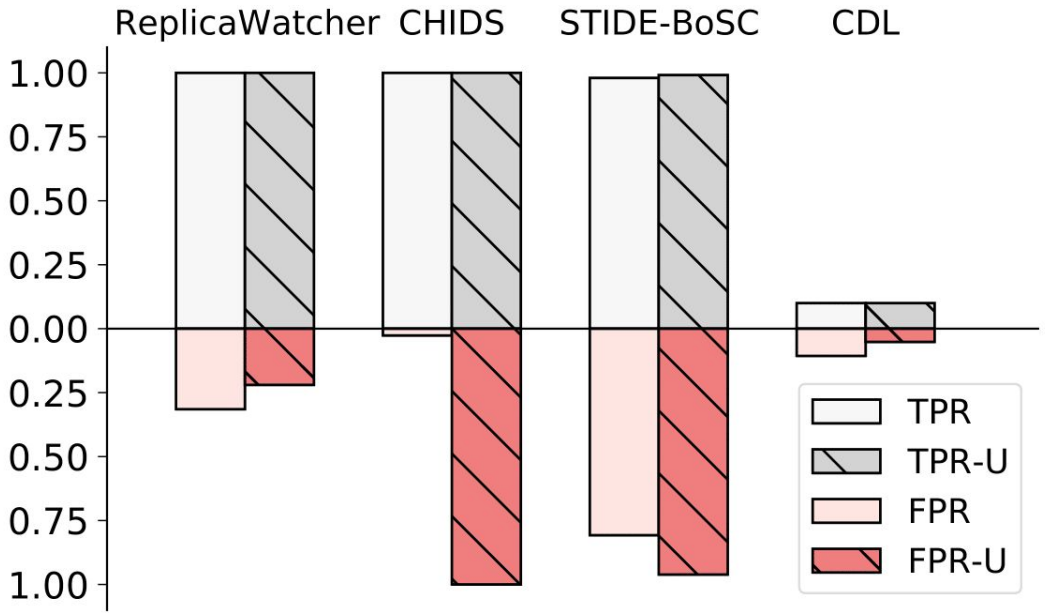
# Evaluation

- How well does ReplicaWatcher work?
  - Showing resilience to normality shifts
  - Detecting anomalies with minimal false positives
- Dataset
  - Tested on two e-commerce platforms (Google Online Boutique, Homebrew)
  - Assessed across 13 attack scenarios spanning five threat families (information disclosure, remote code execution, etc.)

# Resilience against Normality Shifts

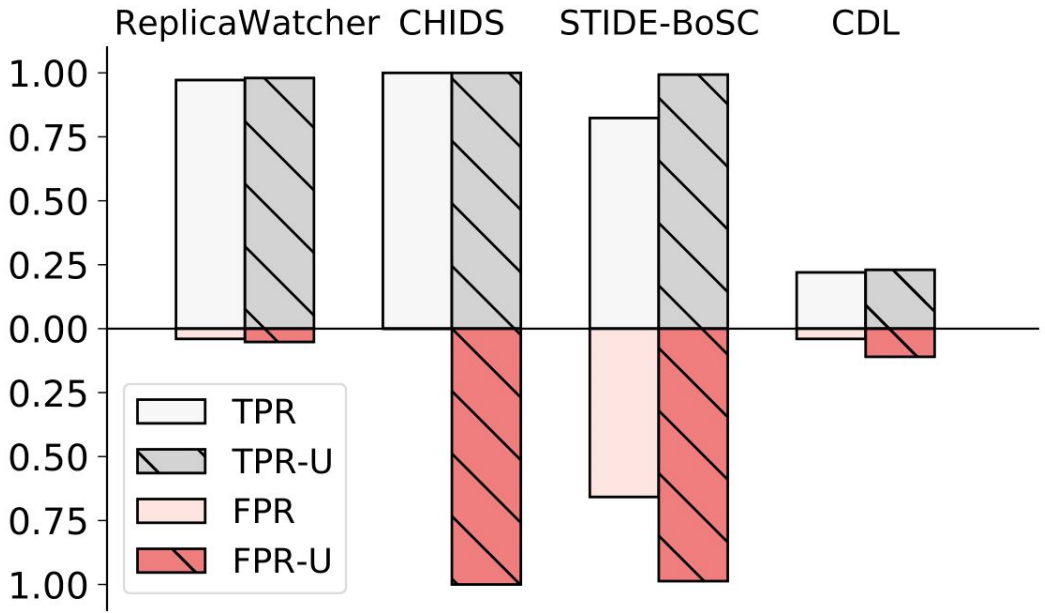
- Experiment
  - Upgrade of base OS image
  - Upgrade of dependency package
  - Change in application code

# Upgrade of Base OS image



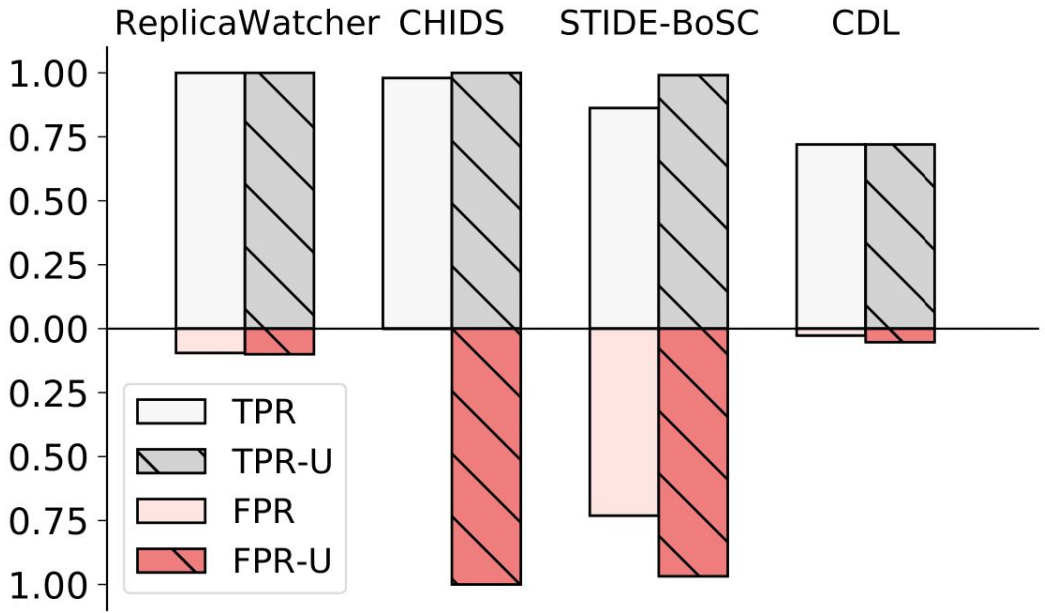

Assessment of Post-Update Performance. (-U) indicates post-update values.

# Upgrade of Dependency Package




Assessment of Post-Update Performance. (-U) indicates post-update values.

# Change in Application Code




Assessment of Post-Update Performance. (-U) indicates post-update values.

# Performance Results

| Threat Families | **Precision** | **Recall** | **F1-score** | **Accuracy** |
|---|---|---|---|---|
| Information Disclosure | 0.9708 | 0.9480 | 0.9587 | 0.9600 |
| Remote Code Execution | 0.9438 | 0.9960 | 0.9674 | 0.9640 |
| Command Injection | 0.8695 | 0.9928 | 0.9251 | 0.9171 |
| Privilege Escalation | 0.9877 | 0.9640 | 0.9757 | 0.9760 |

# Evaluation - Performance Results

| Threat Families | Precision | Recall | F1-score | Accuracy |
|---|---|---|---|---|
| Information Disclosure | 0.9708 | 0.9480 | 0.9587 | 0.9600 |
| Remote Code Execution | 0.9438 | 0.9960 | 0.9674 | 0.9640 |
| Command Injection | 0.8695 | 0.9928 | 0.9251 | 0.9171 |

- Compared **ReplicaWatcher** with training-based solutions
  - Precision of **0.91** vs 0.97 (CHIDS), 0.73 (CDL), 0.59 (Stide-BoSC)
  - Recall of **0.98** vs 0.99 (CHIDS), 0.41 (CDL), 0.94 (Stide-BoSC)

# Limitations

- Uniform attacks (with the same input) across all replicas
- Brute-force Attacks
- Scalability in multi-cluster setups

# Conclusion

- ReplicaWatcher
  - Requires **no training**
  - Shows **resilience** against **normality shifts**
  - Performs **comparable** to **training-based solutions**

# Conclusion

- ReplicaWatcher
  - Requires **no training**
  - Shows **resilience** against **normality shifts**
  - Performs **comparable** to **training-based solutions**

`https://github.com/utwente-scs/Replicawatcher`

Thank you & Questions