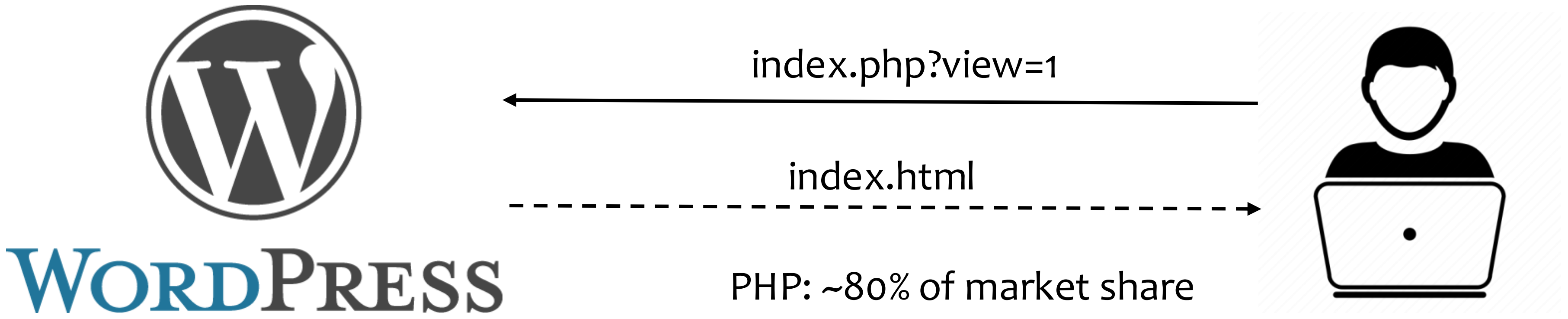
FuzzCache: Optimizing Web Application Fuzzing Through Software-Based Data Cache

Penghui Li[1] and Mingxue Zhang[2]

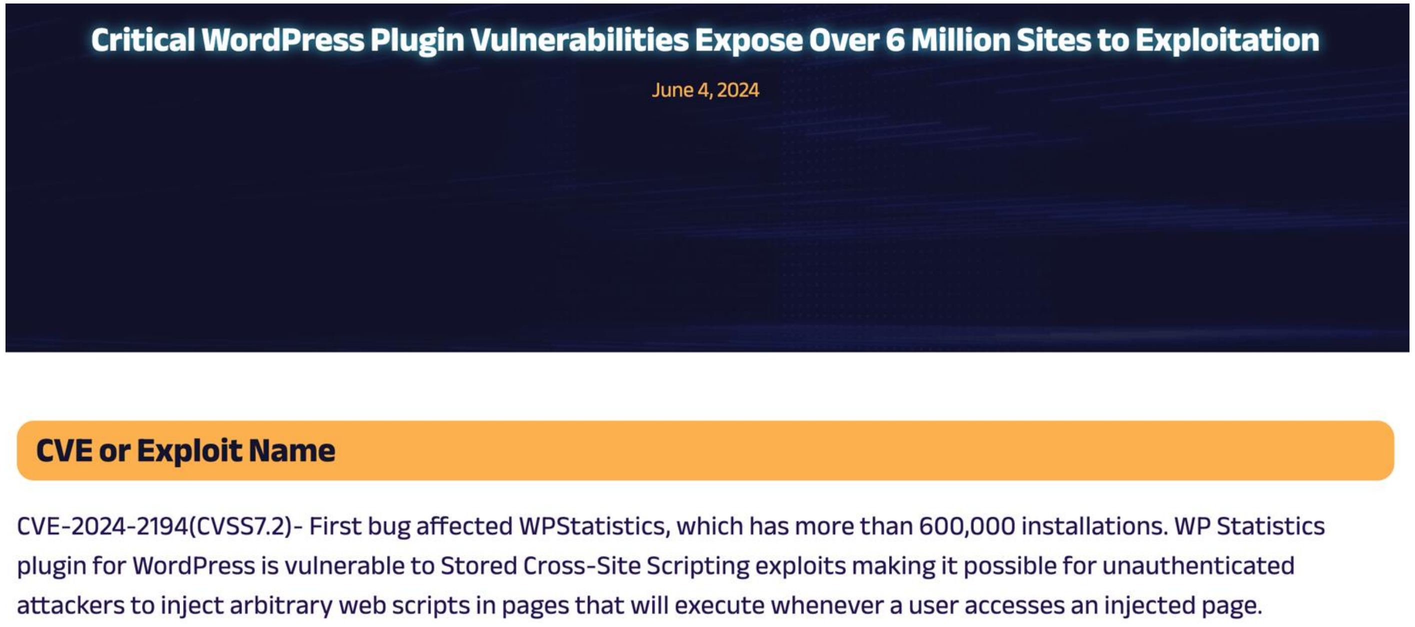
[1]Zhongguancun Laboratory [2]Zhejiang University

(PHP-Based) Web Applications



PHP: ~80% of market share

- SQL injection
- Cross-site scripting
- Cross-site request forgery



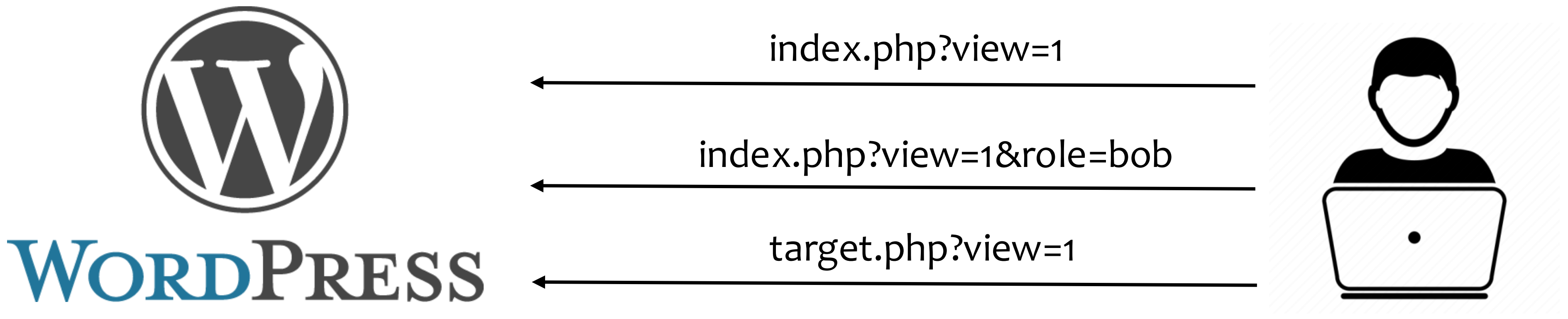
Critical WordPress Plugin Vulnerabilities Expose Over 6 Million Sites to Exploitation

June 4, 2024

CVE or Exploit Name

CVE-2024-2194(CVSS7.2)- First bug affected WPStatistics, which has more than 600,000 installations. WP Statistics plugin for WordPress is vulnerable to Stored Cross-Site Scripting exploits making it possible for unauthenticated attackers to inject arbitrary web scripts in pages that will execute whenever a user accesses an injected page.

Web Application Fuzzing



Fuzzing algorithms

- Improved state exploration & vulnerability oracles
  - Enemy of State (Security ’12)
  - Black-Widow (S&P ’21)
  - Witcher (S&P ’23)
  - Atropos (Security ’24)

⇨

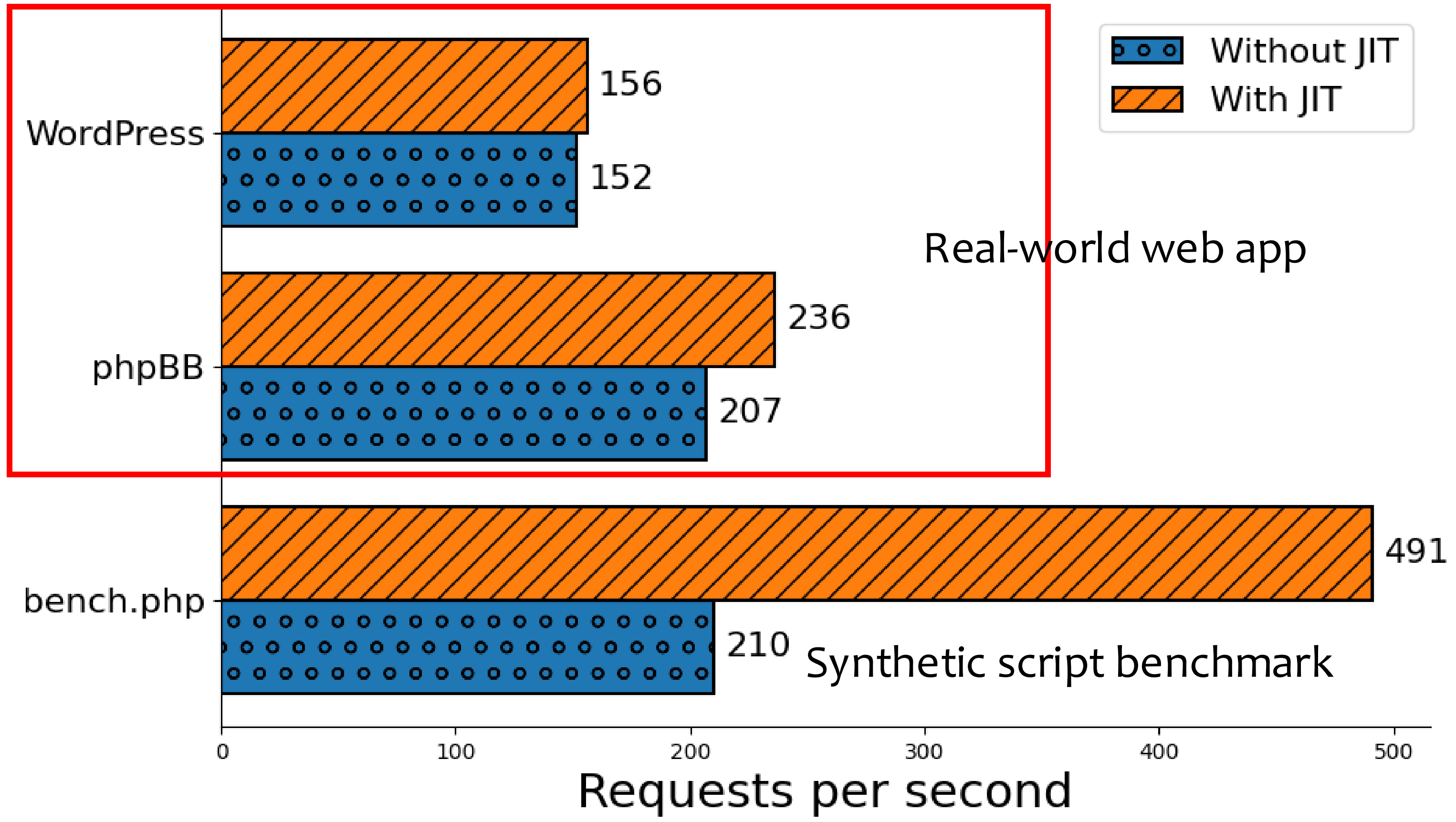
Hundreds of new vulnerabilities discovered

Can we improve web application fuzzing through system optimizations?

PHP 8 Introduced Just-in-Time Compilation

- Server-side code is repeatedly executed

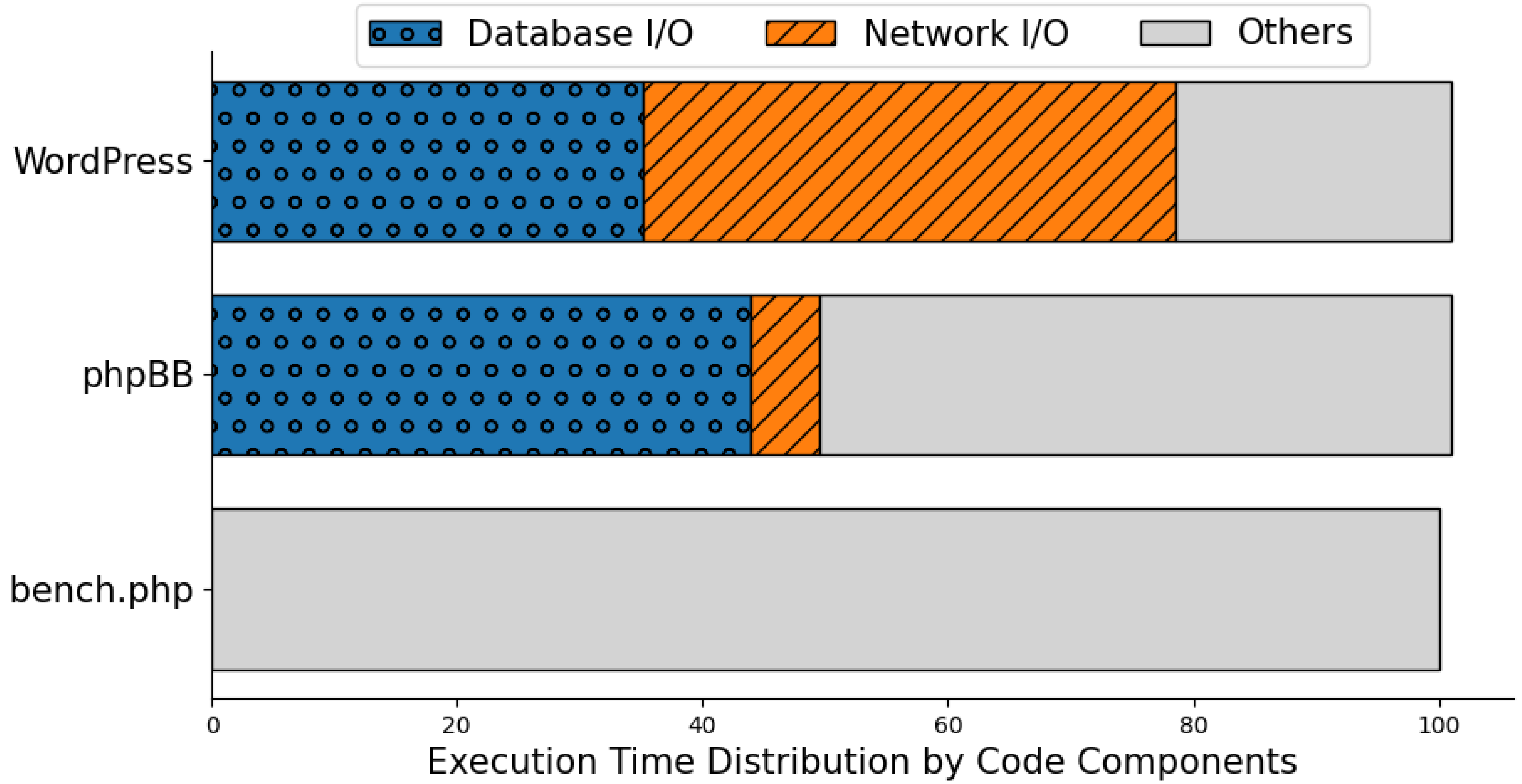
<u>Enabling JIT offered little improvement for real-world web applications</u>



* Experimented with ab tool, PHP OpCache enabled

JIT in Real-World Web Applications

- JIT accelerates code execution but not I/O
  - Two major I/O operations: 78.5% in WordPress and 49.5% in phpBB



I/O During Fuzzing

- Mainly fetch database and network data via built-in functions
  - Repeatedly called
  - Same function arguments and outputs

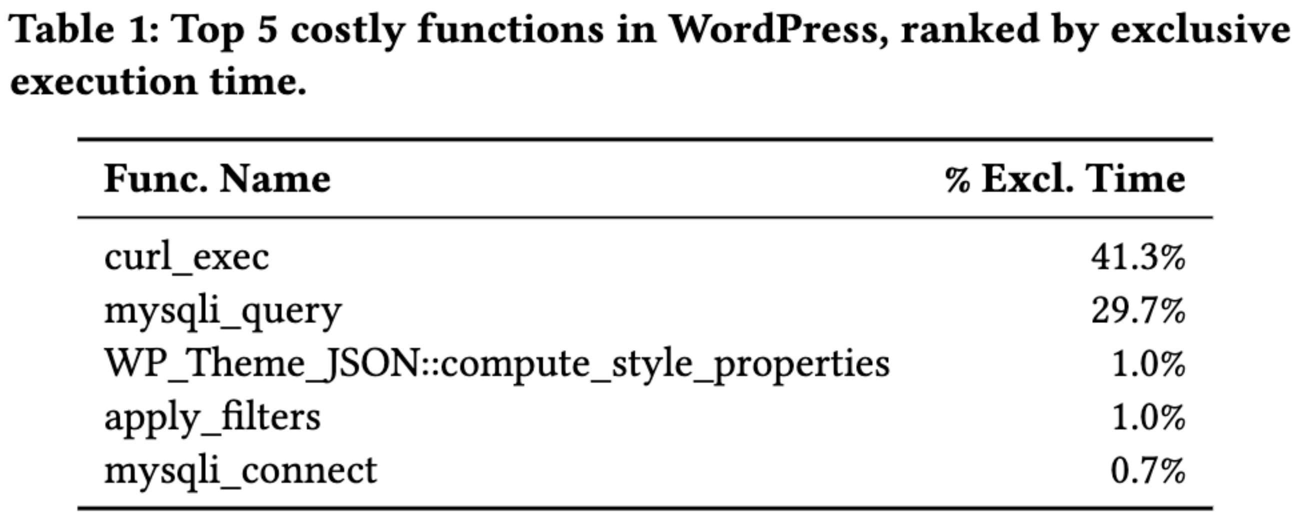
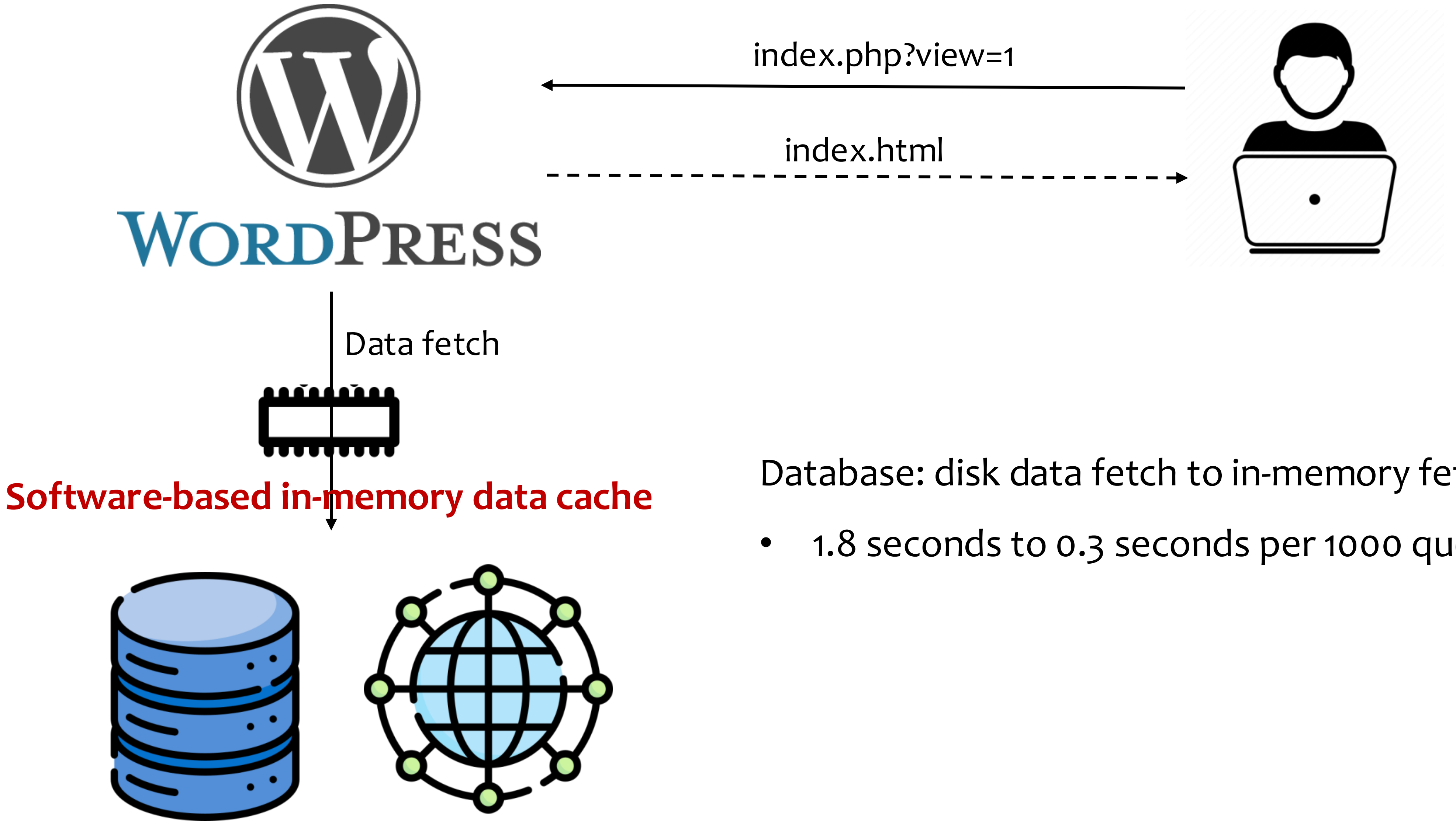


Table 1: Top 5 costly functions in WordPress, ranked by exclusive execution time.

Func. Name	% Excl. Time
curl_exec	41.3%
mysqli_query	29.7%
WP_Theme_JSON::compute_style_properties	1.0%
apply_filters	1.0%
mysqli_connect	0.7%

```
curl_exec("https://api.wordpress.org/core/versioncheck/1.7/?version=6.4.2&php=8.2.13")
myqsli_query("SELECT wp_posts.* FROM wp_posts")
```

Data Cache for Fuzzing



Database: disk data fetch to in-memory fetch

- 1.8 seconds to 0.3 seconds per 1000 queries (6x)

Multi-Phase Database Data Fetch

Data dependencies across phases

```
$conn = mysqli_connect(...)

$result = mysqli_query($conn, $query)

$row = mysqli_fetch_assoc($result)

$fields = mysqli_fetch_fields($result)

$all = mysqli_fetch_all ($result)
```

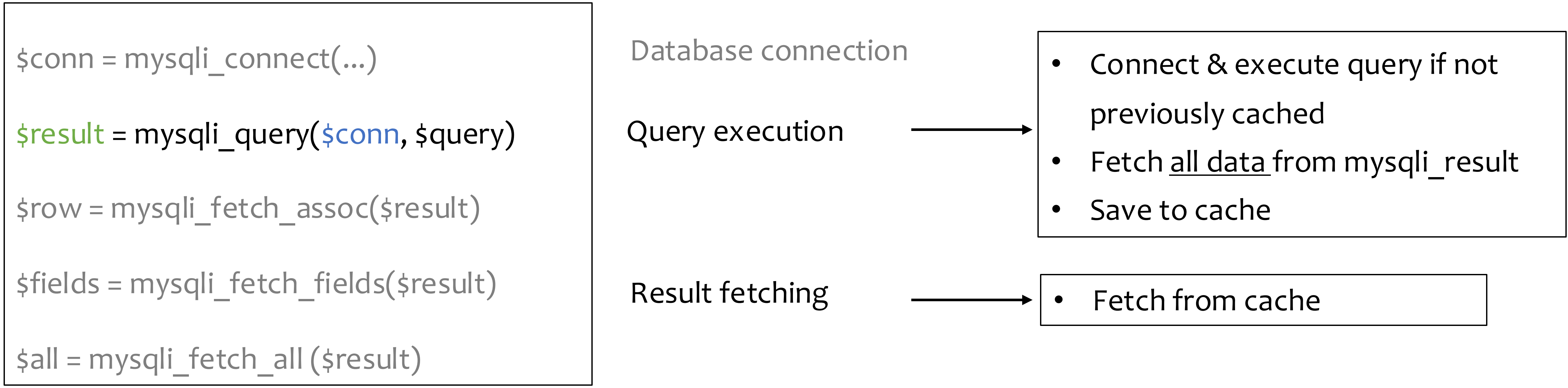
Database connection Query execution	Expensive, should be optimized whenever possible
Result fetching	Multiple (partial) fetches

Database Cache

- Postpone database connection
  - Till necessary query execution

Query-centric design: a cache entry corresponds to a query

- Data prefetch
  - mysqli_result object encompasses active connection and could not be saved



```
$conn = mysqli_connect(...)

$result = mysqli_query($conn, $query)

$row = mysqli_fetch_assoc($result)

$fields = mysqli_fetch_fields($result)

$all = mysqli_fetch_all ($result)
```

Database connection

Query execution →

- Connect & execute query if not previously cached
- Fetch <u>all data</u> from mysqli_result
- Save to cache

Result fetching →

- Fetch from cache

FuzzCache Tool

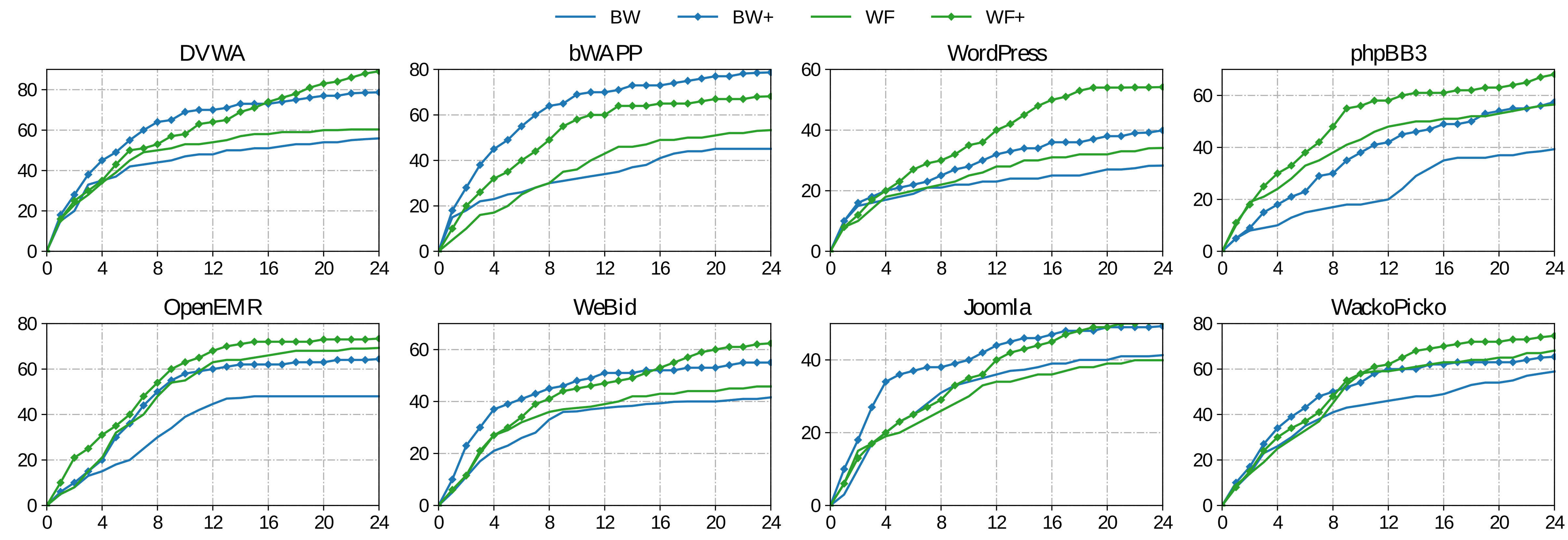
- Network data cache
  - Time-to-expire control

- Just-in-time compilation
  - Fine-tune JIT parameters

- Cross-process data maintenance
  - Inter-process shared memory
  - Persist data across processes or fuzzing trials

Evaluation -- Code Coverage

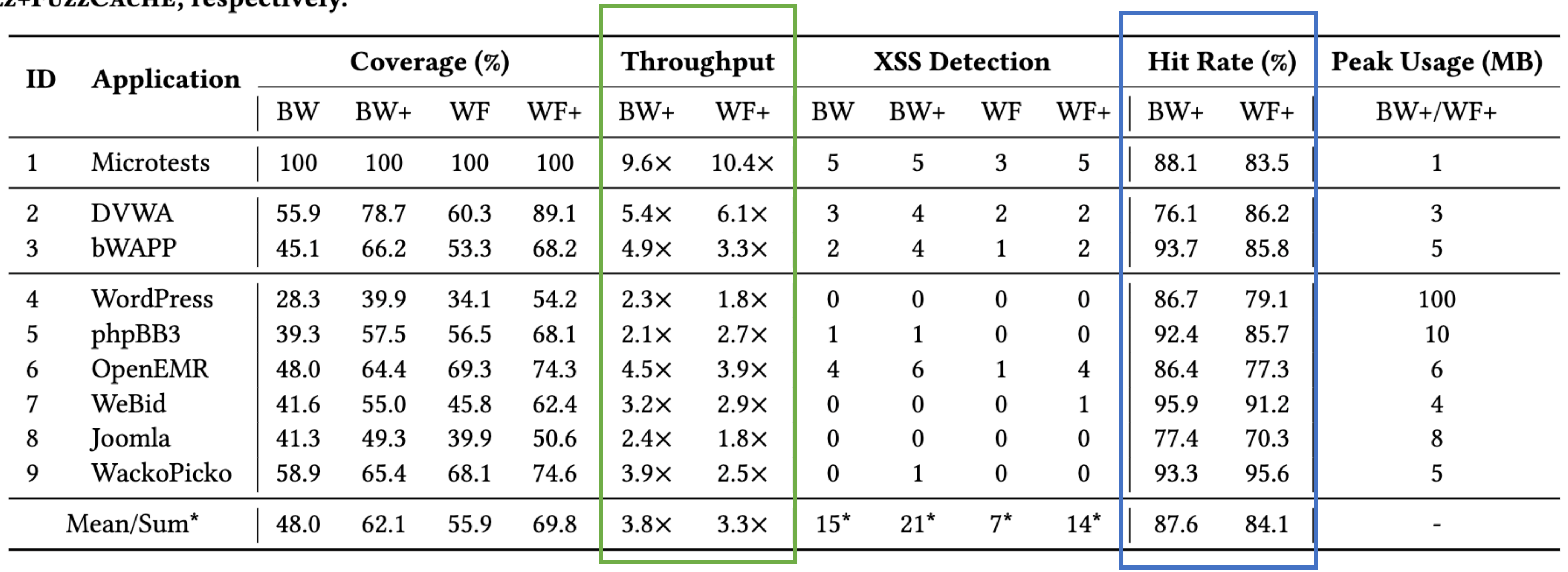
- FuzzCache generally improved Black-Widow and WebFuzz



Evaluation (Cont.)

- Contributed to ~3x throughput boost
- High cache hit rate

Table 4: Evaluation results of 24-hour experiments. BW, BW+, WF, and WF+ denote Black-Widow, Black-Widow+FuzzCache, WebFuzz, and WebFuzz+FuzzCache, respectively.



ID	Application	Coverage (%)				Throughput		XSS Detection				Hit Rate (%)		Peak Usage (MB)
		BW	BW+	WF	WF+	BW+	WF+	BW	BW+	WF	WF+	BW+	WF+	BW+/WF+
1	Microtests	100	100	100	100	9.6×	10.4×	5	5	3	5	88.1	83.5	1
2	DVWA	55.9	78.7	60.3	89.1	5.4×	6.1×	3	4	2	2	76.1	86.2	3
3	bWAPP	45.1	66.2	53.3	68.2	4.9×	3.3×	2	4	1	2	93.7	85.8	5
4	WordPress	28.3	39.9	34.1	54.2	2.3×	1.8×	0	0	0	0	86.7	79.1	100
5	phpBB3	39.3	57.5	56.5	68.1	2.1×	2.7×	1	1	0	0	92.4	85.7	10
6	OpenEMR	48.0	64.4	69.3	74.3	4.5×	3.9×	4	6	1	4	86.4	77.3	6
7	WeBid	41.6	55.0	45.8	62.4	3.2×	2.9×	0	0	0	1	95.9	91.2	4
8	Joomla	41.3	49.3	39.9	50.6	2.4×	1.8×	0	0	0	0	77.4	70.3	8
9	WackoPicko	58.9	65.4	68.1	74.6	3.9×	2.5×	0	1	0	0	93.3	95.6	5
	Mean/Sum*	48.0	62.1	55.9	69.8	3.8×	3.3×	15*	21*	7*	14*	87.6	84.1	-

Discussion

- Cache invalidation at table granularity
  - Identify table names in query strings
- Compatibility with recent oracles
  - Witcher & Atropos identify parsing errors as indicators of bugs
  - Execute a lightweight query syntax parser even when data is cached
- Targeted data cache vs. generic, adaptive data cache
  - Extend to other frequently accessed data, e.g., template loading

Summary

- Real-world web applications are data-centric
  - Data fetch is often repeated, redundant, and expensive

- Data cache is a generic optimization that complements existing fuzzers
  - Improve application execution speed transparently

Thank You!

Questions?

lipenghui315@gmail.com