What PostgreSQL Can Teach Us About Composable Data Management Systems

Abigale Kim
University of Wisconsin–Madison
abigale@cs.wisc.edu

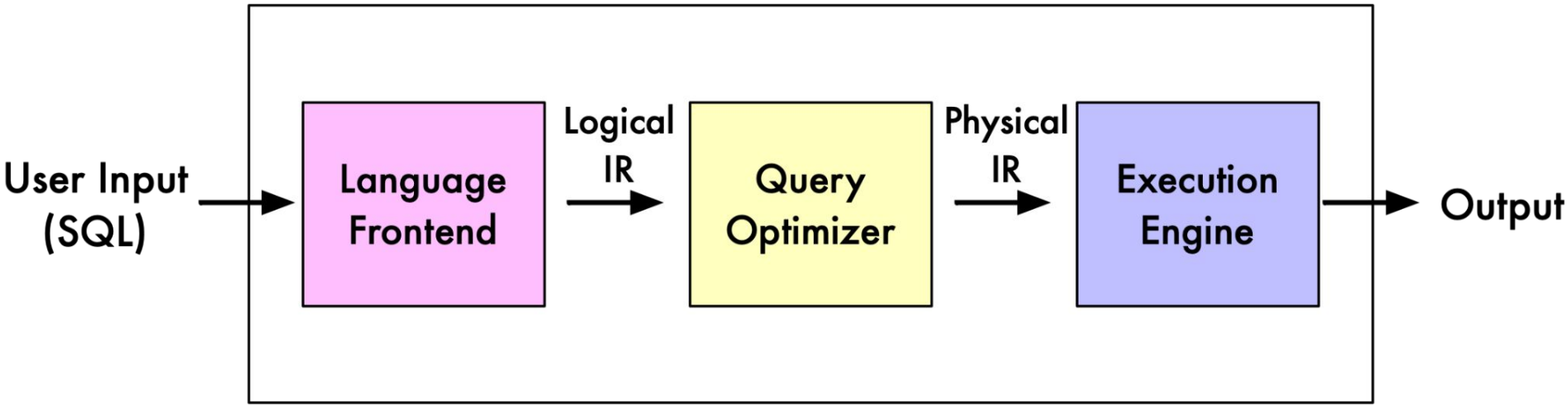
Collaborators
Andy Pavlo (pavlo@cs.cmu.edu)
Dave Andersen (dga@cs.cmu.edu)
Marco Slot (marco.slot@crunchydata.com)

Outline

- **Introduction**
- Survey
- Analysis Framework
- Findings
- Discussion
- Conclusion & Takeaways

What is composability?

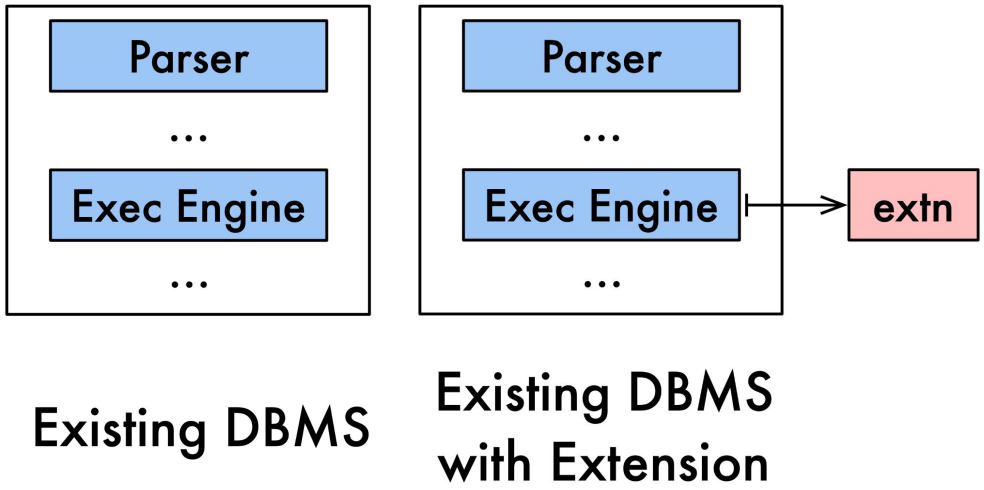
- Composability: the capability of constructing a new DBMS from a collection of its naturally well-defined components



P. Pedreira, O. Earling, K. Karanasos, S. Schneider, W. McKinney, S. Valluri, M. Zait, J. Nadeau. “The Composable Data Management System Manifesto”. VLDB 2023.

What is extensibility?

- Extensibility: the capability of a database system to let custom software extend its capabilities
- Extension: an instance of this software



Extensibility vs. composability

- Similarities
  - Both consist of reusable, modular components in a DBMS
- Differences
  - Extension adds one component to an existing DBMS
  - Composable DBMS consists of individual components added together

Research question: What lessons from extensibility can we take to composability?

Outline

- Introduction
- **Survey**
- Analysis Framework
- Findings
- Discussion
- Conclusion & Takeaways

Survey goals

- No larger understanding of DBMS extensibility
- Organize and classify extensibility design decisions
- Better understand DBMS extensibility design

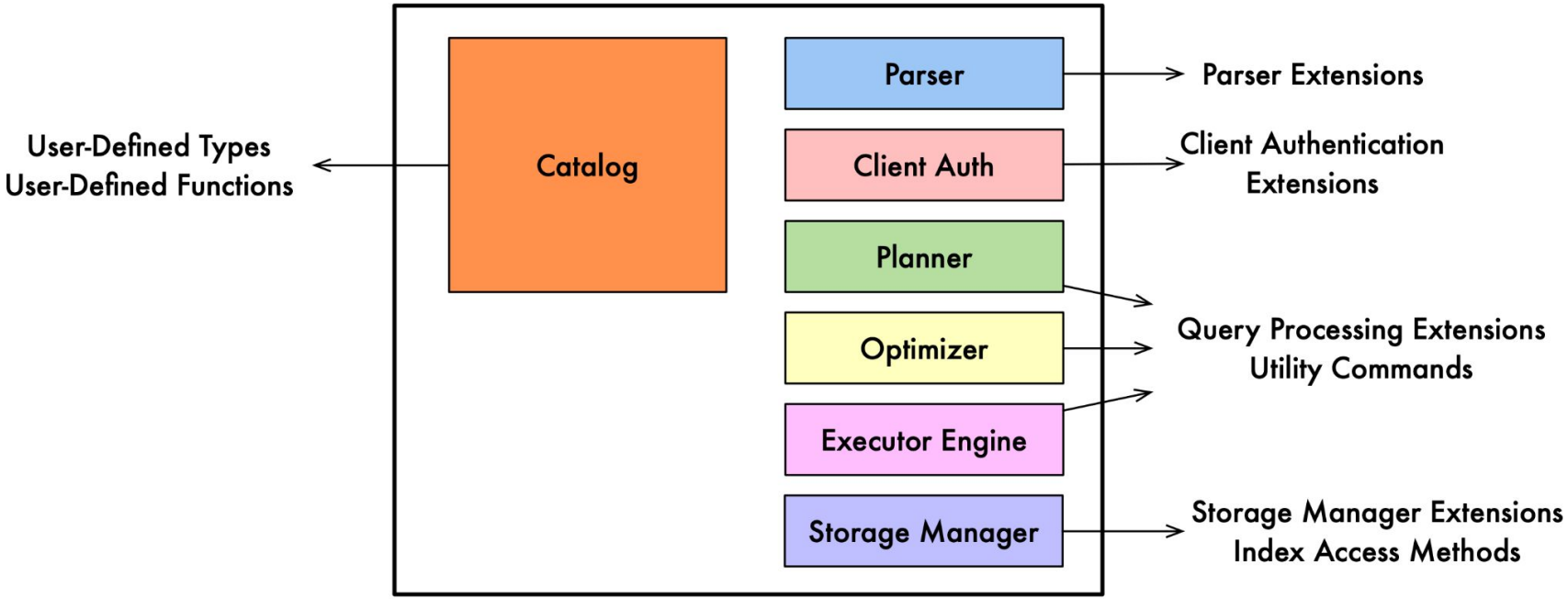
Solution: Create a taxonomy on DBMS extensibility!
Categorize existing systems using our taxonomy!

Survey method

- Examined six different DBMSs (PostgreSQL, MySQL, SQLite, DuckDB, Redis OSS)
  - Open source, more comprehensive support for extensibility
- Read extensibility implementation and extensions
- Focused on well-recognized extension code + DBMS code

For the sake of this presentation, we'll focus on the PostgreSQL findings.
Ask me about the other DBMSs!

Types of extensibility



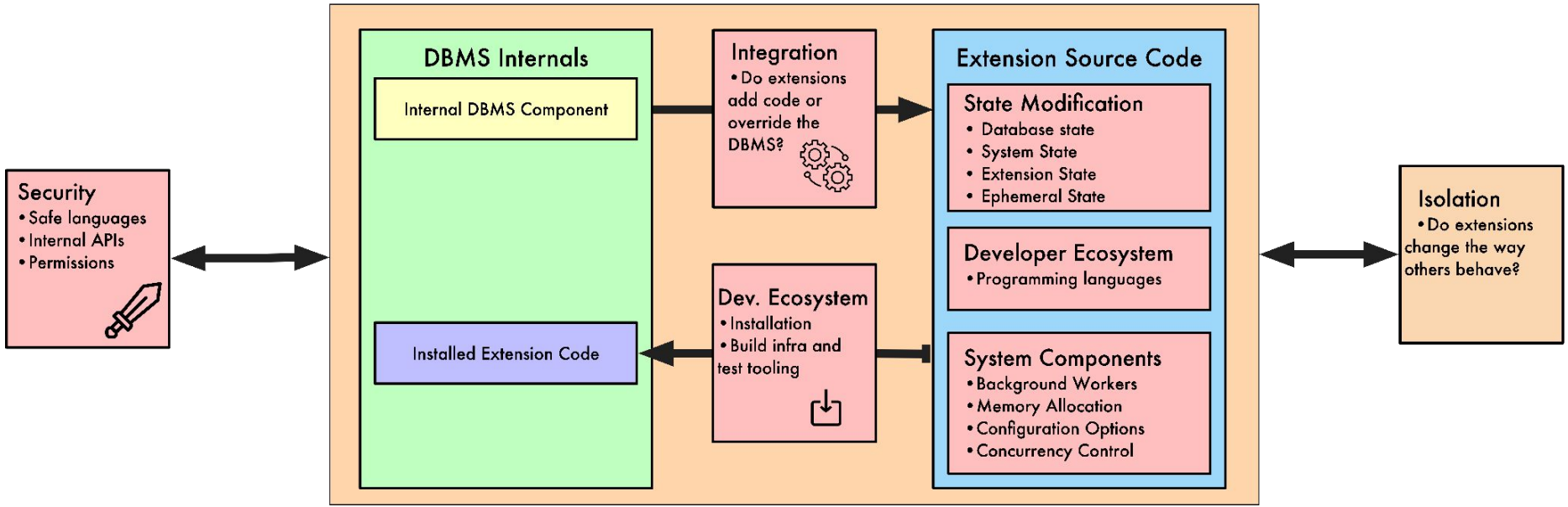
Internal DBMS Architecture

Types of extensibility in DBMSs

- 7/8 types of extensibility supported in PostgreSQL

	PostgreSQL	MySQL	MariaDB	SQLite	Redis	DuckDB
User-Defined Functions	**Yes** (408)	**Yes** (2)	**Yes** (1)	**Yes** (79)	**Yes** (57)	**Yes** (41)
User-defined Types	**Yes** (139)	No	**Yes** (13)	No	No	**Yes** (4)
Utility Commands	**Yes** (43)	No	No	No	No	No
Parser Modifications	No	**Yes** (2)	**Yes** (1)	No	No	**Yes** (4)
Query Processing	**Yes** (46)	**Yes** (7)	**Yes** (5)	No	No	**Yes** (4)
Storage Managers	**Yes** (44)	**Yes** (13)	**Yes** (18)	**Yes** (43)	No	**Yes** (9)
Index Access Methods	**Yes** (67)	No	No	No	No	**Yes** (3)
Client Authentication	**Yes** (17)	**Yes** (3)	**Yes** (10)	No	No	No
Version Examined	[illegible]	v8	v11	v3	v7	v1
Number of Extensions	441	29	68	98	57	44+

DBMS Extensibility Taxonomy



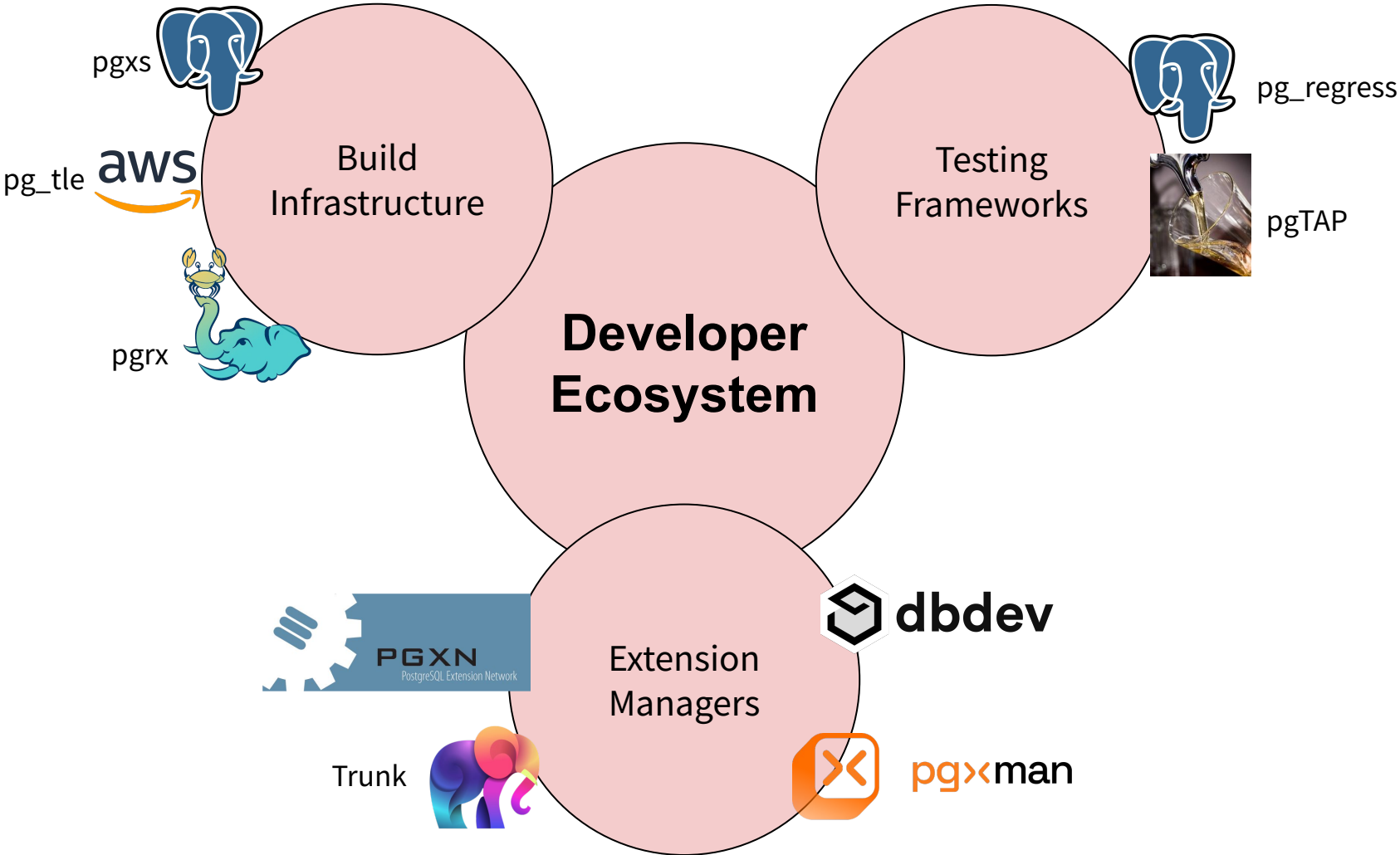
Interface design decisions

- Adding vs. overriding host functionality
  - PostgreSQL supports both
- State modification (database state, system state, extension state, ephemeral state)
  - PostgreSQL allows extensions to modify all four
- Protection
  - PostgreSQL has limited extension isolation and security
  - Superuser vs. not superuser extensions

Common system components

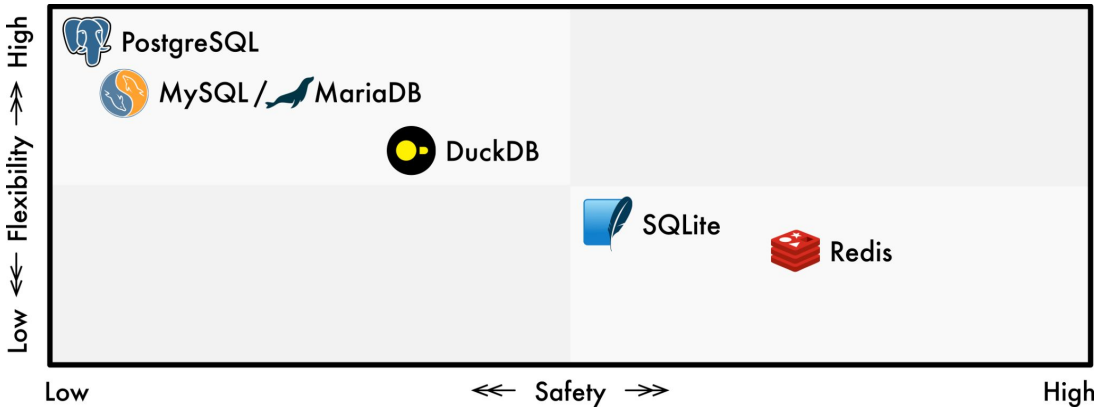
1. Background workers
2. Memory allocation mechanisms
3. Configuration options
    a. Modifying postgresql.conf/pg_hba.conf, GUC variables
4. Custom concurrency control
    - Advisory locks

PostgreSQL has support for 4/4 system components


pgxs
pg_tle
aws
pgrx
Build
Infrastructure
Testing
Frameworks
pg_regress
pgTAP
Developer
Ecosystem
PGXN
PostgreSQL Extension Network
Extension
Managers
dbdev
Trunk
pgxman

Comparison to other DBMSs

- Supports the most extensibility mechanisms
  - Types of extensibility
  - Mechanisms for building extensibility
  - Building and testing infrastructure
- Similar design to MySQL
  - Support lots of extensibility via overriding function pointers
- Redis OSS is the safest, only allows command-extensibility using their DBMS



PostgreSQL survey takeaways

- PostgreSQL has a very flexible extensibility interface
  - Allows for both extending and overriding
  - Extensions can modify all state
  - PostgreSQL offers limited security and isolation for extensions
- PostgreSQL supports a variety of extensibility mechanisms
  - 7/8 identified types of extensibility (and more!)
  - All system components
- PostgreSQL has robust extension building and testing infrastructure
  - pgxs, pg_tle, pgrx
  - pg_regress for testing

Outline

- Introduction
- Survey
- **Analysis Framework**
- Findings
- Discussion
- Conclusion & Takeaways

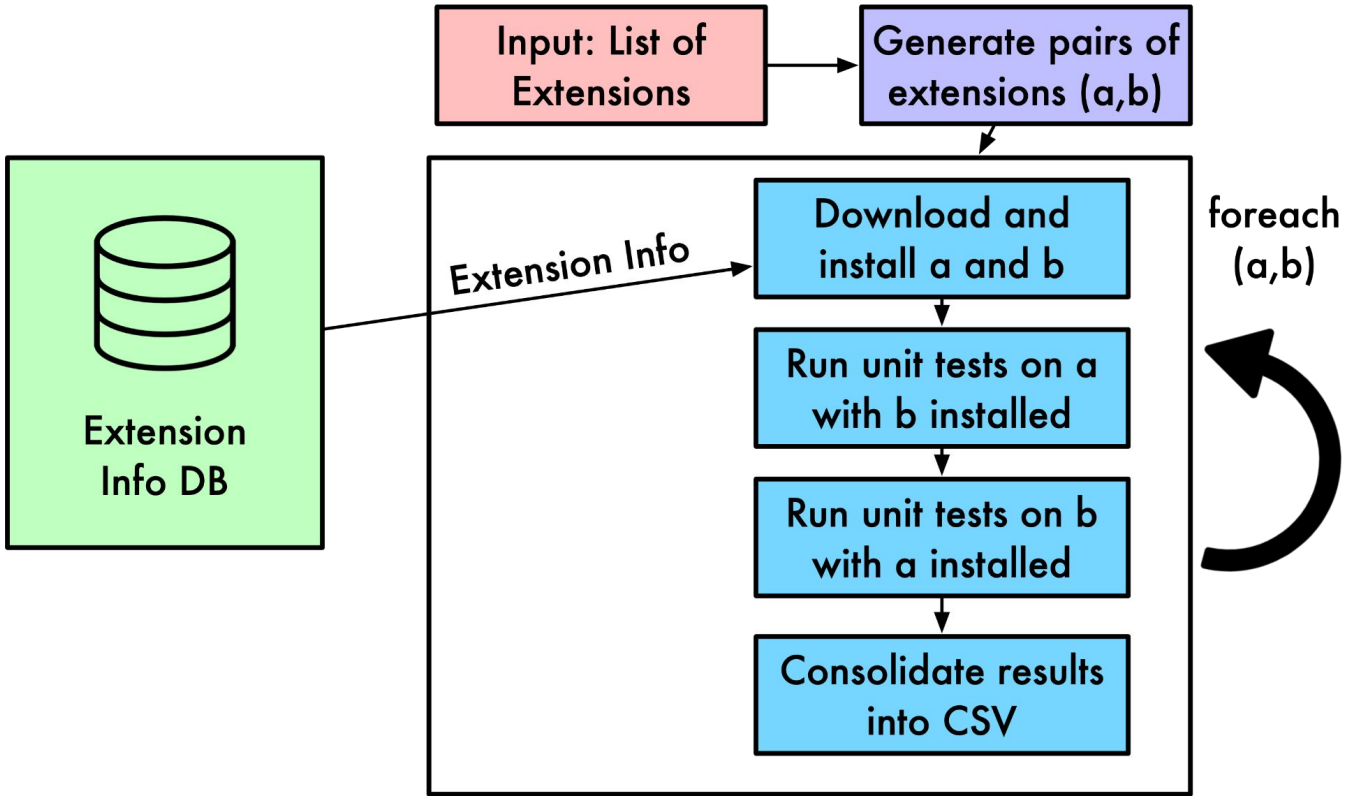
ExtAnalyzer!

- Tests compatibility between different extensions
  - Compatibility: two extensions work as intended when installed together
- Collects information about extensions
  - Types of extensibility and system components used
  - Source code analysis on bad practices (e.g. duplicate code in PostgreSQL and extension codebases)

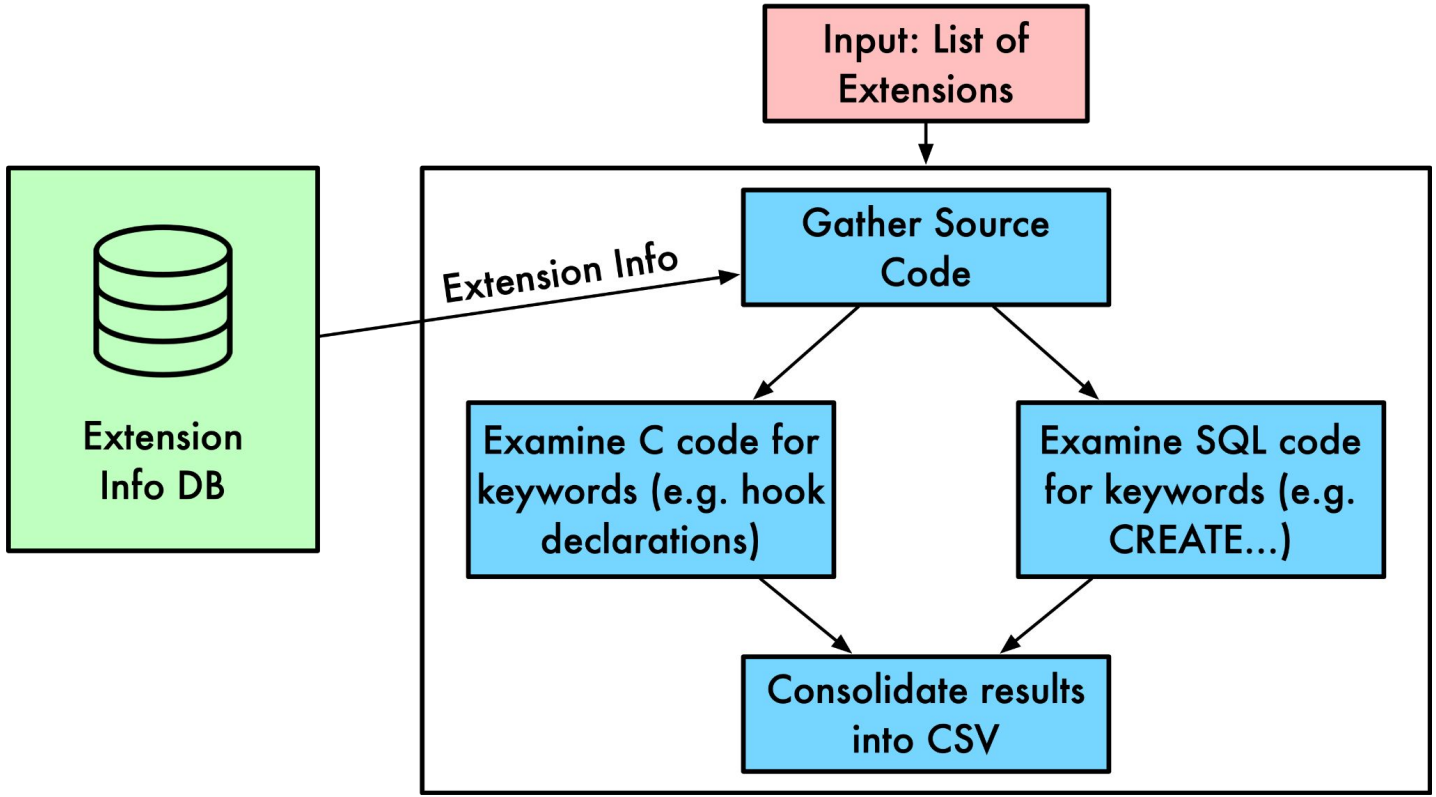
Framework logistics

- Postgres v15.3
- Tested on 441 extensions (one of the following)
  - **`contrib`** directory
  - AWS + Azure + Google Cloud extension offerings
  - Listed on PGXN (PostgreSQL Extension Registry)
- Python + Bash scripts
- Utilizes pg_regress for running extension tests

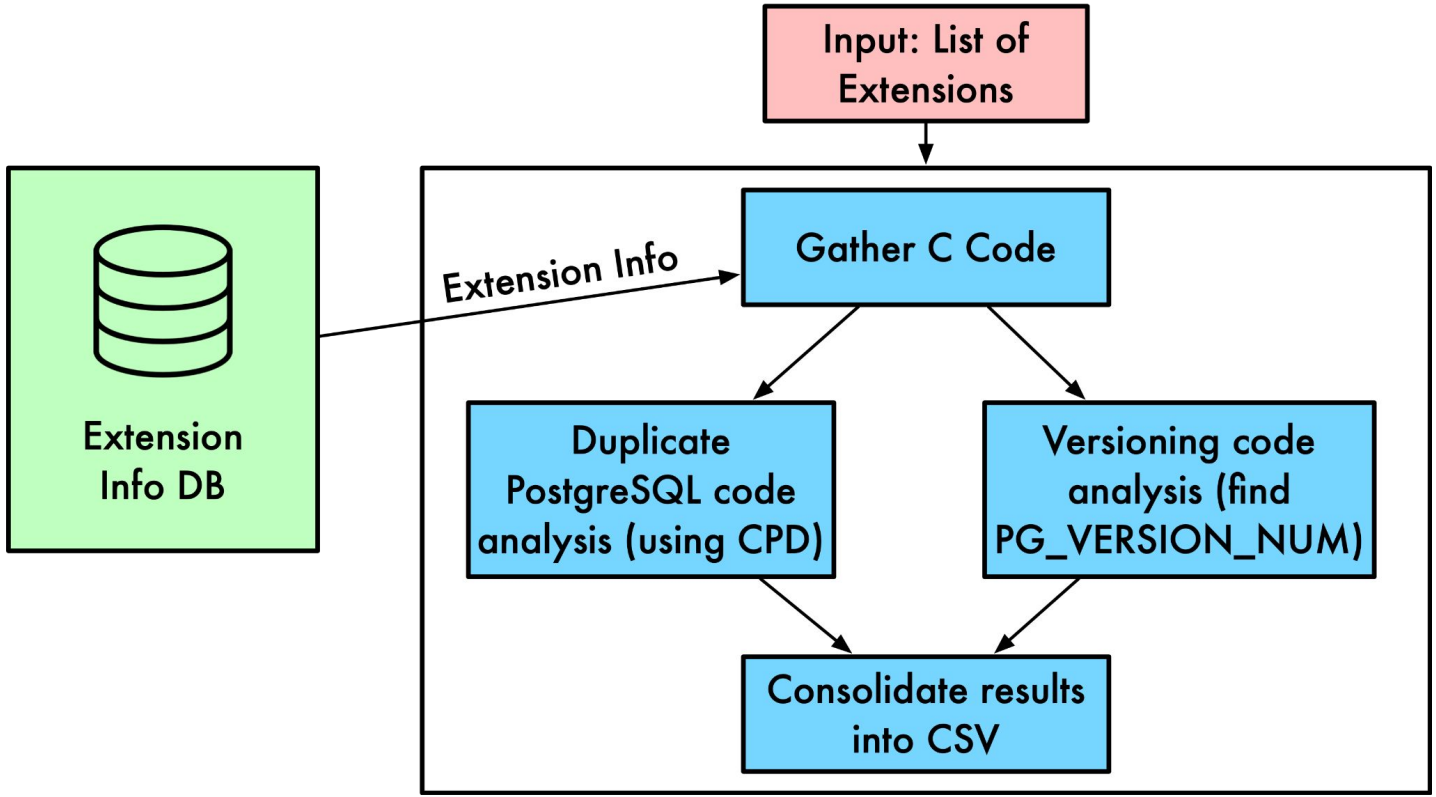
Compatibility analysis



Information analysis



Source code analysis



Duplicate code

- Extensions copy code from PostgreSQL to use in their own extensions
  - Static functions that they can't call from core PostgreSQL but they want to use
  - Significant portions of complicated logic (e.g. switch statements)
- In our analysis: counted instances with more than 100 tokens
  - Tokens: Identifiers, constants, special keywords, special symbols
- Used **PMD Copy-Paste Detector (CPD)** tool

core_postgresql.c

```
static void foo(void) {
  . . .
}
```

extension.c

```
void foo(void) {
  . . .
}

int a(void) {
. . .
foo();
. . .
}
```

Versioning

- PostgreSQL keeps a internal version macro PG_VERSION_NUM
- Extensions utilize this to support different logic for each version
- Results in overly complex, hard-to-read code
- PostgreSQL versions vary greatly from one another

```
#if PG_VERSION_NUM > 14
. . .


#else
. . .
```

Outline

- Introduction
- Survey
- Analysis Framework
- **Findings**
  - **Compatibility Analysis**
  - **Information Analysis**
  - **Source Code Analysis**
- Discussion
- Conclusion & Takeaways

Fun Monstrosities in PostgreSQL

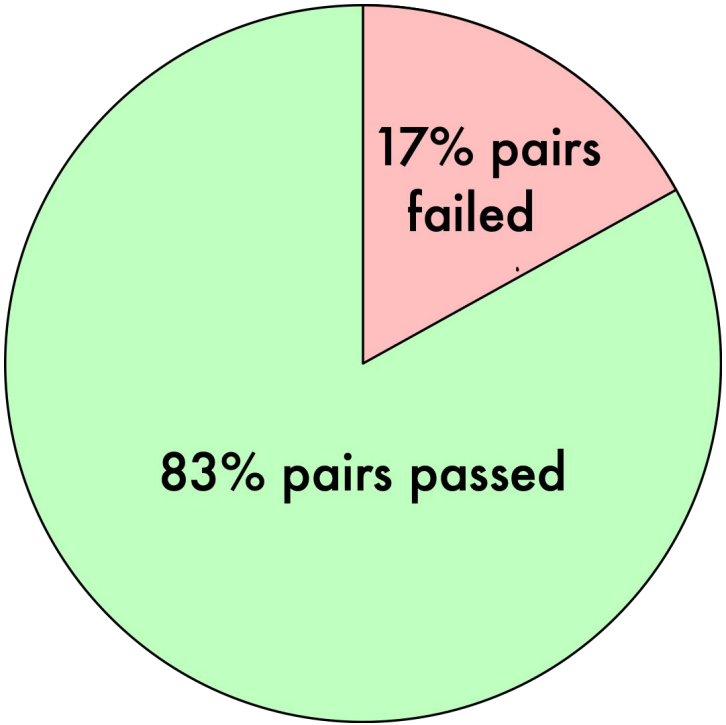
```
#if PG_VERSION_NUM>=150000
static shmem_request_hook_type  PreviousShmemRequestHook = NULL;
#endif
```

```
2023-07-25 05:04:55.945 UTC [687073] STATEMENT:  truncate table t;
2023-07-25 05:04:56.945 UTC [687074] ERROR:  deadlock detected
```

```
-- Create first test user
CREATE USER user1 password 'password';
ERROR:  password must contain both letters and nonletters
ALTER ROLE user1 SET pgaudit.log = 'ddl, ROLE';
```

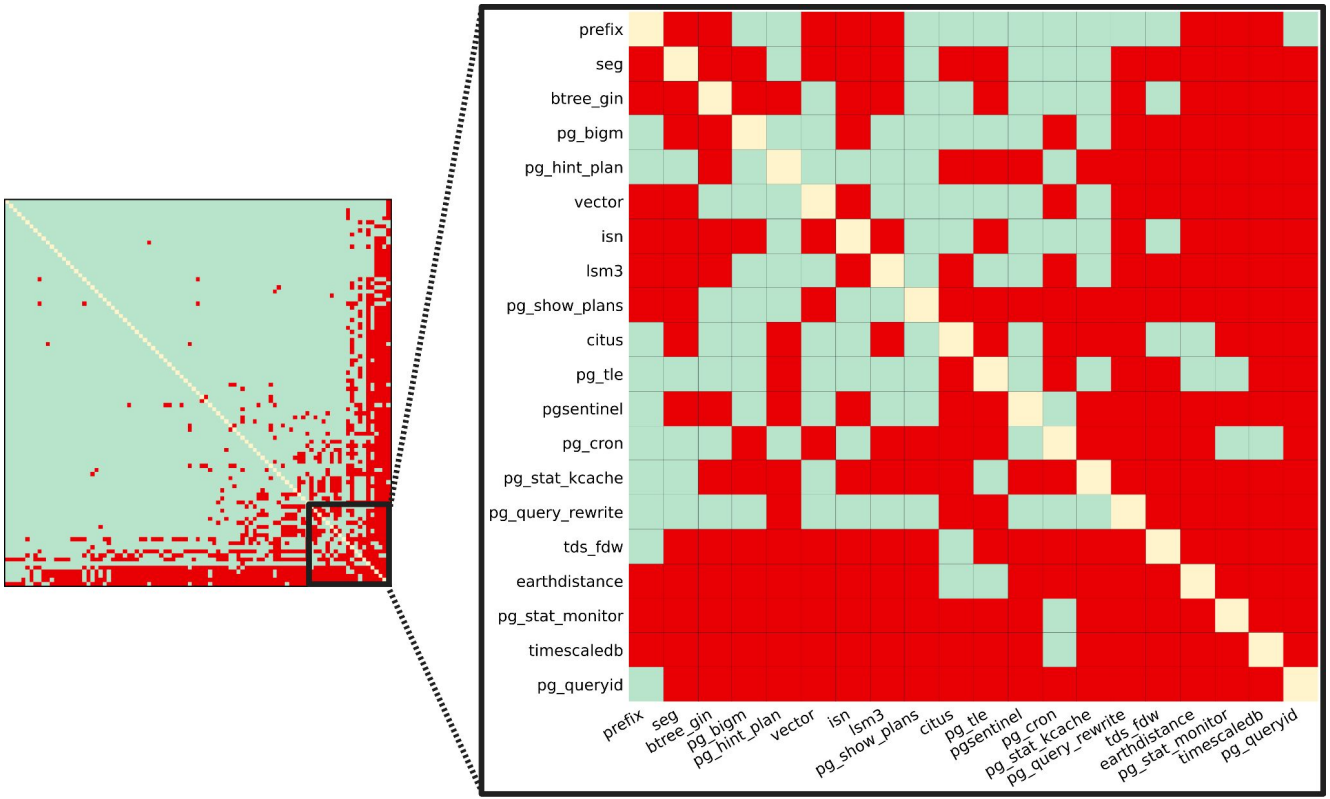
```
2023-07-26 11:57:04.925 UTC [2087903] LOG:  database system is shut down
```

Compatibility Analysis



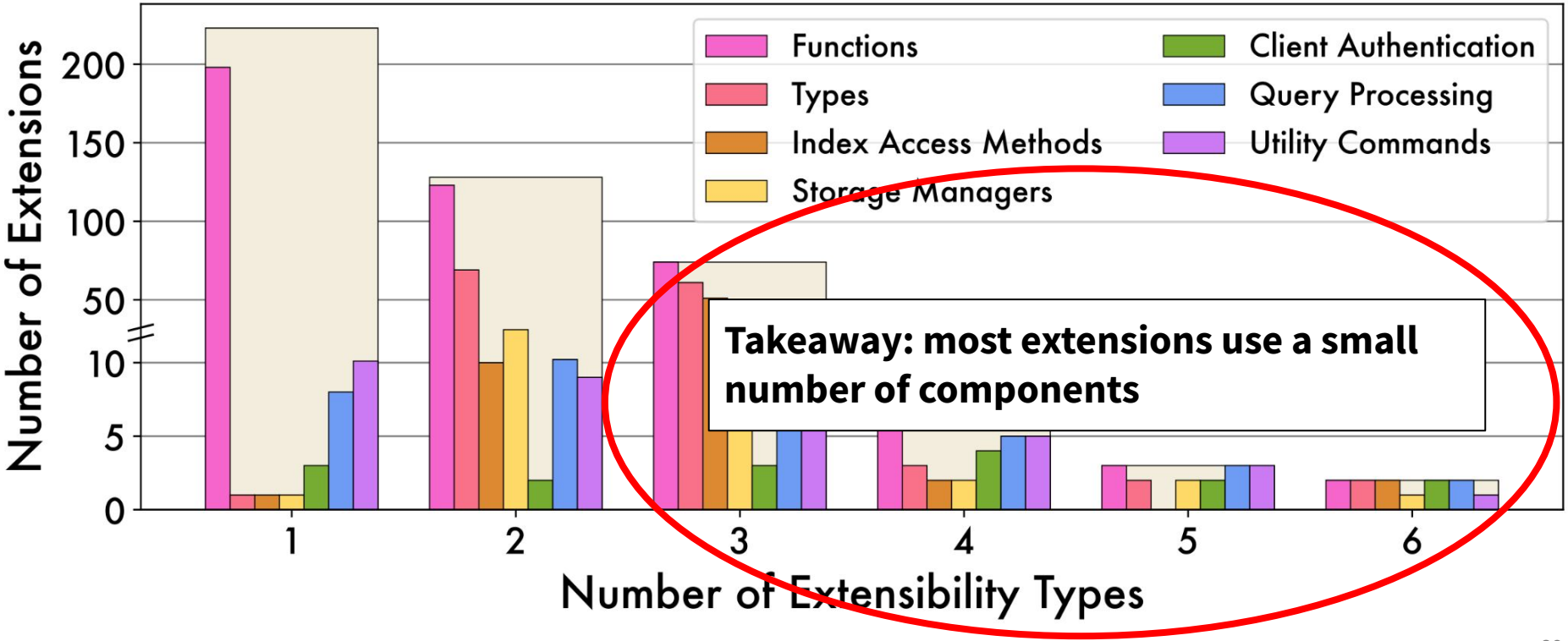
- Recorded extension pair failures, not specific tests failures
- **Takeaway: Extension compatibility is a common problem**

Compatibility Failure Matrix



- **Takeaway: Most extensions do not produce conflicts, but some produce a lot of compatibility conflicts**

Information Analysis: Extensibility Distribution

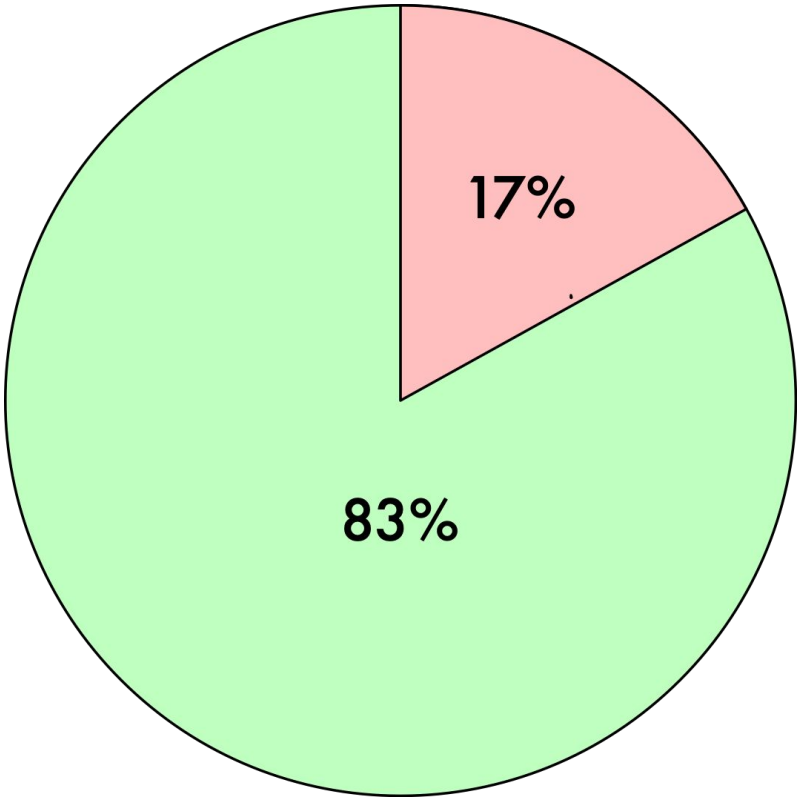


Complex Extensions (>= 3 extensibility types)

- Used K-modes clustering to determine types of complex extensions
- New indexes (67.9%)
  - UDFs, UDTs, index access methods
- Query processing features (21.0%)
  - UDFs, query processing, utility commands
- UDT-optimized query engine (4.9%)
  - UDFs, UDTs, Storage Manager, Query Processing
- New storage manager (4.9%)
  - UDFs, storage manager, utility commands
- Full-featured extensions (1.2%)
  - All extensibility types

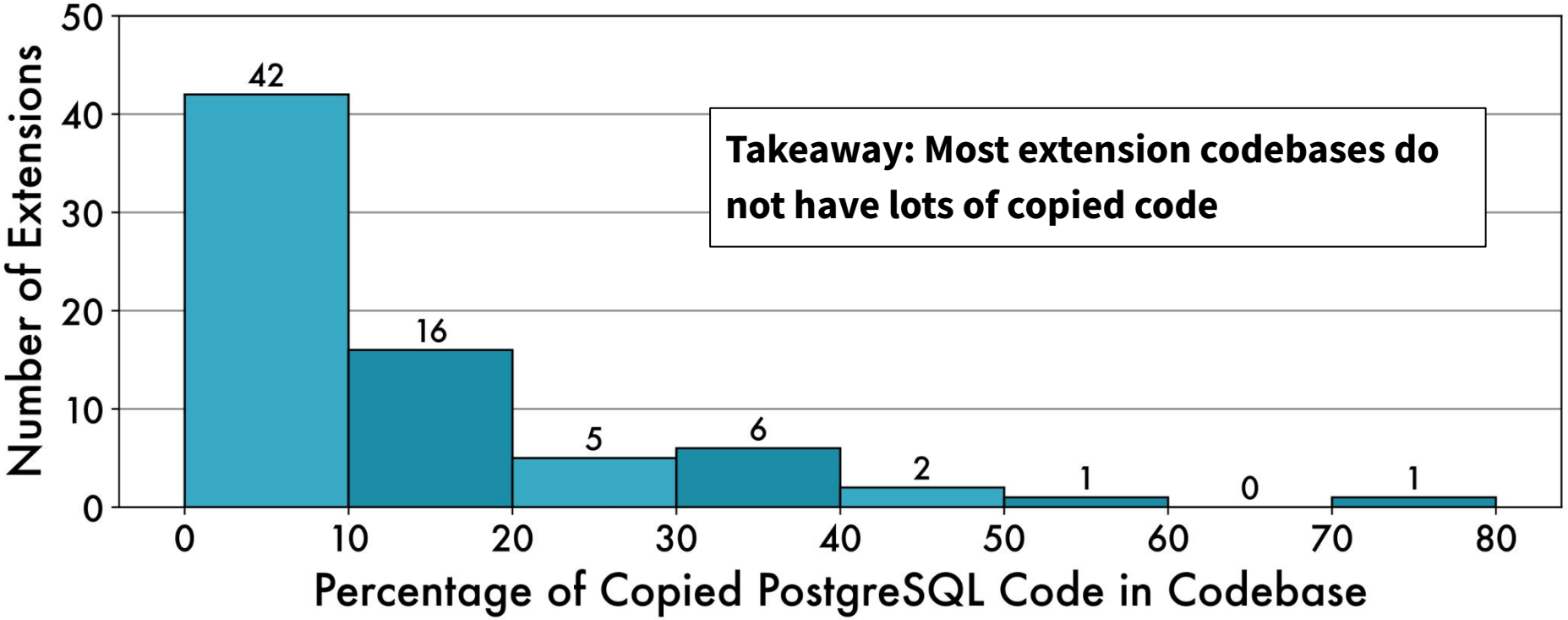
- **Takeaway: Most complex extensions use certain types of extensibility together**

Source Code Analysis: Copied PostgreSQL Code

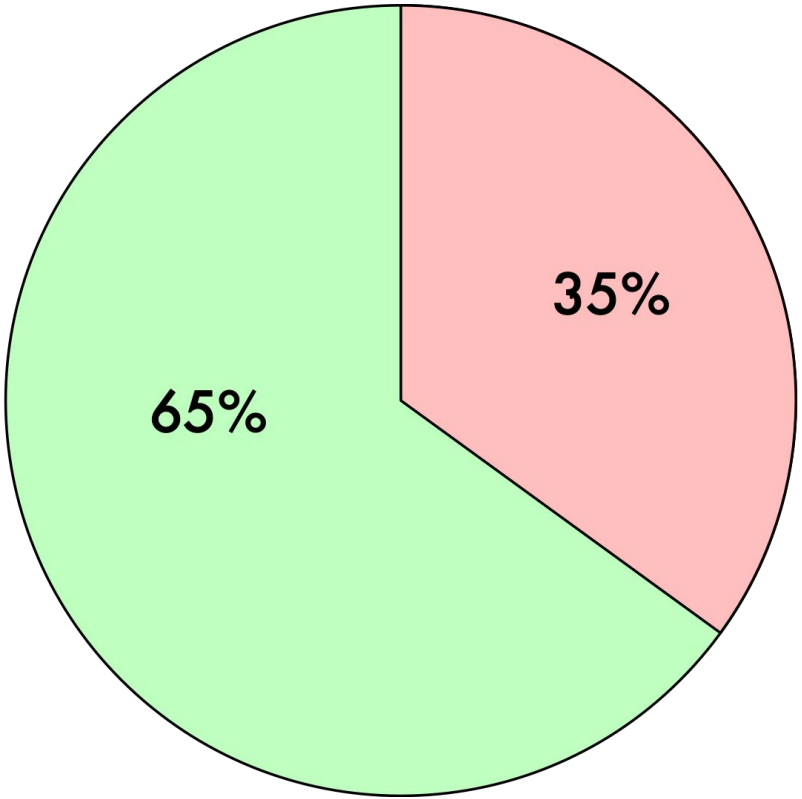


- 17% of extensions copy PostgreSQL code
- **Takeaway: Extensions copying PostgreSQL code is a common phenomenon**

Source Code Analysis: Copied code histogram

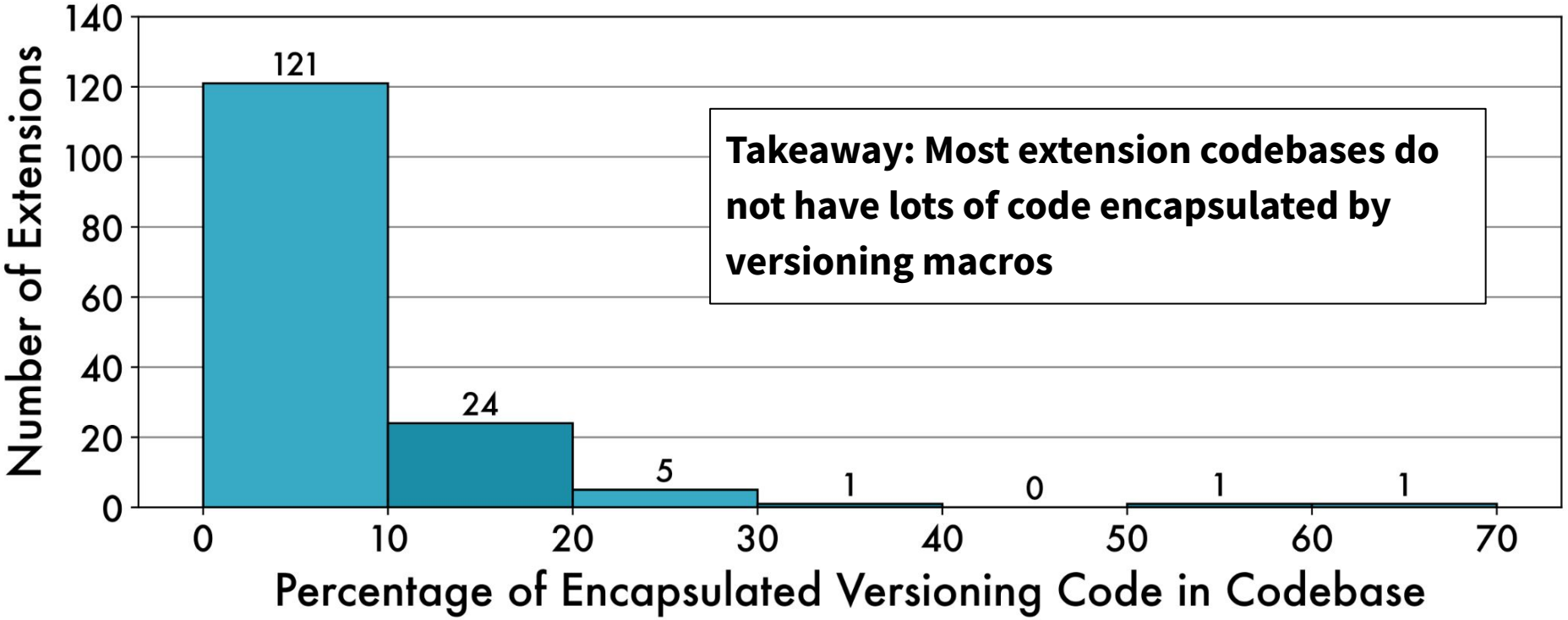


Source Code Analysis: Versioning Code Logic



- 35% utilize versioning code logic
- **Takeaway: Extensions utilizing versioning logic is a common phenomenon.**

Source Code Analysis: Versioning code histogram



Connecting properties to incompatibility

- Ran T-paired testing to find extensions with higher incompatibility
- Higher complexity causes higher incompatibility
  - Using many extensibility types, system components, hooks
  - Duplicate code usage, having a larger codebase
- Larger scope query planner and execution hooks
  - Modifying the planner or executor decreases compatibility

Outline

- Introduction
- Survey
- Analysis Framework
- Findings
- **Discussion**
- Conclusion & Takeaways

What lessons can DBMS composability take from extensibility?

1. Caution when transferring state

- PostgreSQL gives extensions large responsibility over internal state without helper APIs
  - Query execution hooks failing is due to lots of internal state (e.g. plans, intermediate results)
- Solution: create shared APIs meant for all modules to use
  - Allows modules to manipulate shared state in a standardized way

2. Composability in composable DBMSs

- Many extensions build on top of each other
  - Extensions in PostgreSQL rely on pg_stat_statements to collect planning and execution statistics
  - Extensions in MySQL collect statistics for InnoDB (a storage engine extension)
- No defined way of calling other extensions in PostgreSQL
  - Enforcing a load order/dependency graph to ensure modules can build on top of each other
- Allow customization in modules
  - Can cater to systems use cases

3. Composable system templating

- Our analysis showed that most complex extensions can be categorized into several use cases
- Composable systems would benefit from finding out similar patterns and use cases
  - SiriusDB (GPU SQL engine built using DuckDB planner & optimizer)

4. Building and testing infrastructure

- PostgreSQL is so popular because of its building and testing infrastructure
- Providing standardized building and testing infrastructure allows people to contribute more easily
  - Build problems in PostgreSQL: dependencies, versioning, language support
  - Test problems in PostgreSQL: takes lots of effort to integrate testing
  - No package manager, and no official registry

Takeaways

- Survey findings: PostgreSQL's flexible interface, comprehensive support, and usability results in significantly more prolific ecosystem
- Analysis findings: Extensions are commonly incompatible with each other, caused by higher complexity and query execution modules
- Suggestions: Internal state management, composability^2, system templating, infrastructure

Github Repo: ExtAnalyzer!

Website: https://db.cs.cmu.edu/pgexts-vldb2025/

Email: abigale@cs.wisc.edu

History of Extensibility

- Extensibility: 1970s
  - Ingres: Supported UDTs and UDFs
- Extensibility: 1980s
  - Starburst: extendible query processing
  - Exodus: modules (kernel libraries, storage manager, rule-based query optimizer) to help developers
  - Genesis: abstract interfaces filled in by developers
- PostgreSQL archaeology
  - 2007: planner hook added
  - 2008: execution hooks added
  - 2011: CREATE EXTENSION command supported
  - 2012 - now: extension development skyrocketed!