Anarchy in the Database

Survey and Evaluation of Database Management System Extensibility

Abigale Kim
Carnegie Mellon University, TileDB Inc.
abigalekim0417@gmail.com

Collaborators

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

What is extensibility?

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

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!

PostgreSQL extensibility is impactful in industry

Cloud offerings support many extensions

Google Cloud

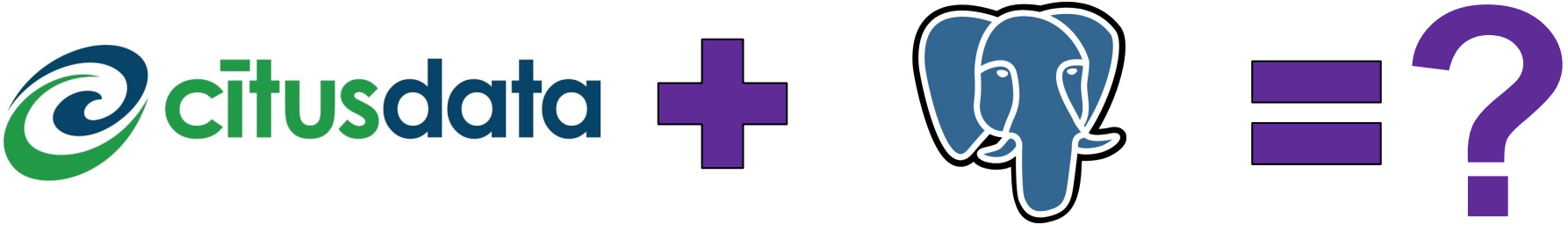
Many third party extensions

DBMS	Number of Extensions
DuckDB	30
MySQL	47
Redis OSS	57
SQLite	61
PostgreSQL	375+

What happens when we combine extensions?

Distributed PostgreSQL (as an extension)

auto_explain (execution logging)



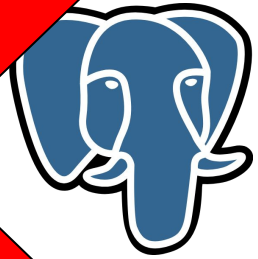
SQL Error [XX000]: ERROR: cache lookup failed for type 0 with pg auto explain #7596

Open StepanYankevych opened this issue 2 weeks ago · 2 comments

What happens when we combine extensions?

Distributed PostgreSQL (
an extension)

a plain
(on logging)



=?

SQL Error [XX000]: ERRO e lookup fai ype 0 with pg auto explain
#7596

Open StepanYankevych opened this issue 2 weeks ago · 2 comments

Research questions

- What design decisions caused conflicts like this (and similar)?
- How well-designed is DBMS extensibility?
- Can we design extensibility to not have these conflicts?
- What can we do to improve the design of DBMS extensibility?

Outline

- Motivation
- **Survey**
- Analysis
- Results
- Discussion

Survey goals

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

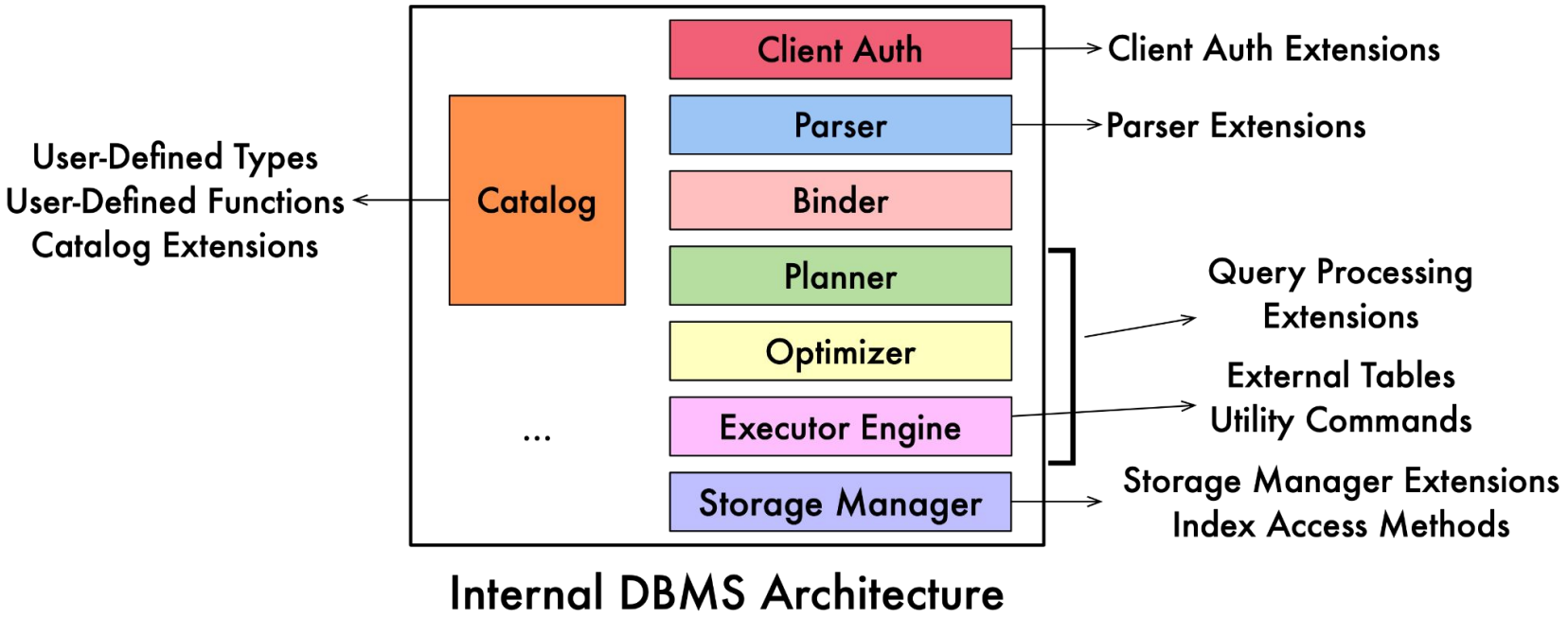
Solution: Create a taxonomy on DBMS extensibility!
Categorize existing systems using our taxonomy!

Survey method

- Examined five 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



Types of extensibility pt. 1

- User Defined Types (UDTs)
  - Physical types with custom binary encoding and support functions
- User Defined Functions (UDFs)
  - Custom function that extends DBMS's functionality
- External Tables
  - Allow users to interact with data sources outside the DBMS
- Index Access Methods
  - Additional index implementations
- Catalog Modifications
  - Modify or access database metadata
  - DuckDB uses them as an external table

Types of extensibility pt. 2

- Client Authentication
  - Password validation, user privilege levels
- Parser Extensions
  - e.g. adding new syntax, query rewrite rules
- Query Processing Extensions
  - Planner, optimizer, executor
- Utility Commands
  - DDL commands, e.g. CREATE/DROP SCHEMA/DATABASE, user privileges, etc.
- Storage Manager Extensions
  - MySQL’s storage engines are a well known example

Types of extensibility in DBMSs

- 9/10 types of extensibility supported in PostgreSQL

	PostgreSQL	DuckDB	MySQL	MariaDB	SQLite	Redis
User-defined Functions	Yes	Yes	Yes	Yes	Yes	Yes
User-defined Types	Yes	Yes	No	No	No	Yes
External Tables	Yes	Yes	Yes	Yes	Yes	Yes
Utility Commands	Yes	No	No	No	No	No
Parser	No	Yes	Yes	Yes	No	No
Query Processing	Yes	Yes	No	No	No	No
Storage Engine	Yes	Yes	Yes	Yes	Yes	No
Index Access Methods	Yes	No	No	No	No	No
Client Authentication	Yes	No	Yes	Yes	No	No
Catalog	Yes	Yes	Yes	Yes	No	No
Number of Extensions	375+	30+	47+	65+	61+	57+

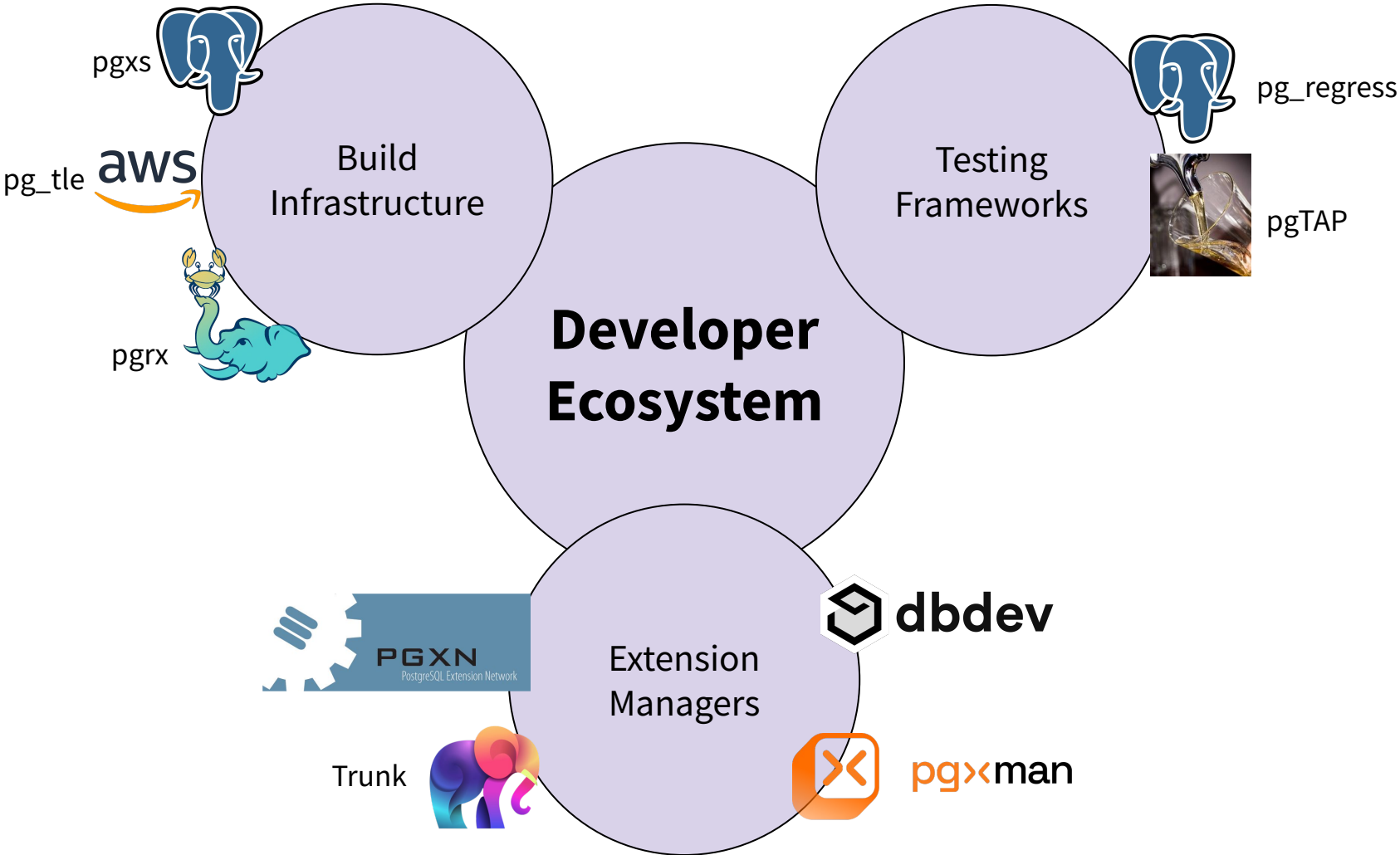
Interface design decisions

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

Common system components

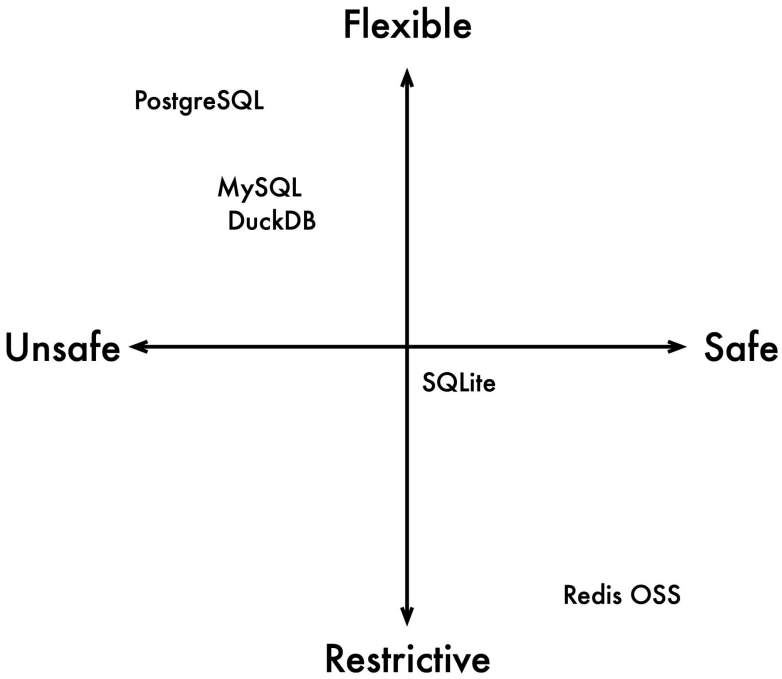
1. Background workers
2. Memory allocation mechanisms
3. Configuration options
    a. Modifying postgresql.conf/pg_hba.conf, GUC variables
4. Custom concurrency control
    - Advisory locks

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
  - 9/10 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

- Motivation
- Survey
- **Analysis**
- Results
- Discussion

Revisiting Citus and auto_explain

- Two well known, frequently used extensions still have bugs when used together
- How common is this problem?
- If this problem is common, why (from a design perspective)?

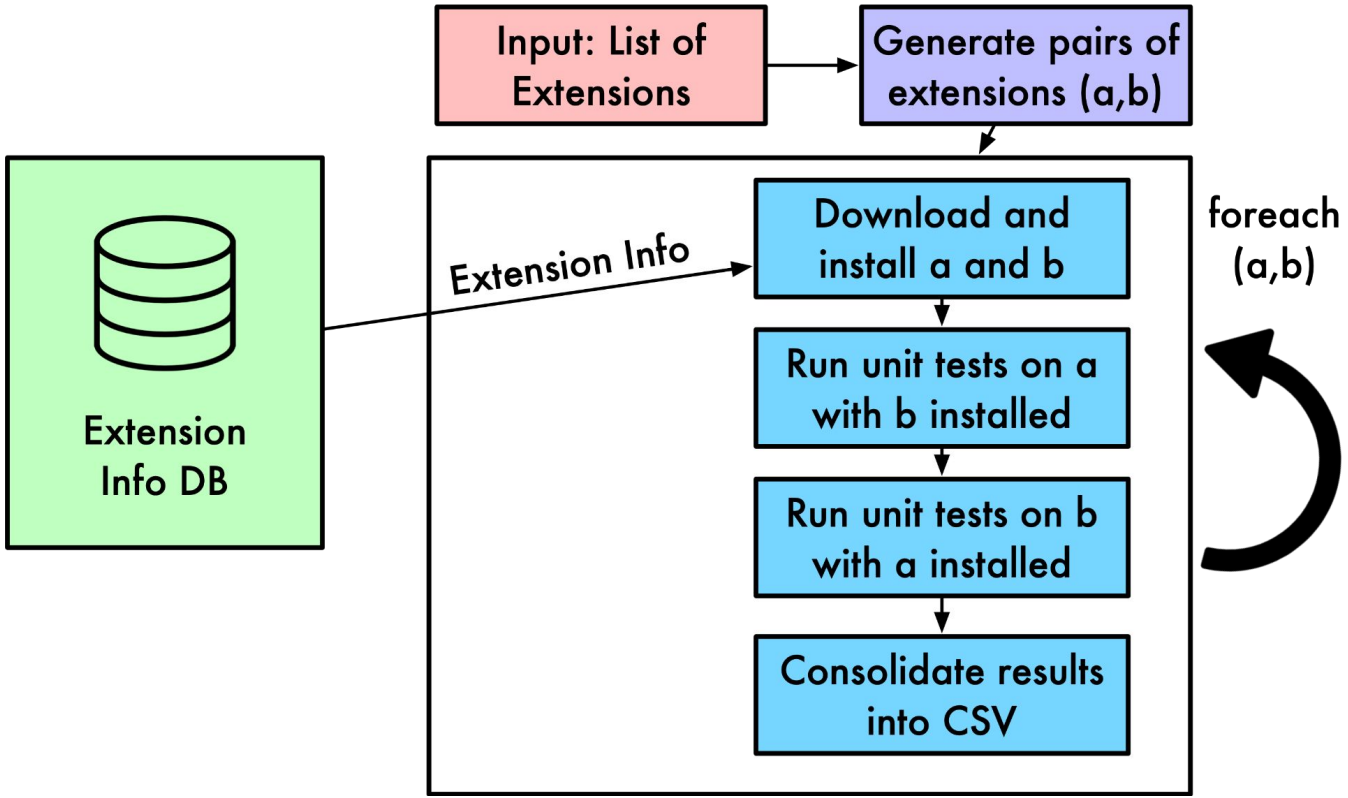
Analysis framework

- Tests compatibility between different extensions
  - Compatibility: two extensions work as intended when installed together
- Collects information about extensions
  - Both general information analysis and source code analysis
- Goal: to understand factors that cause incompatibility between extensions

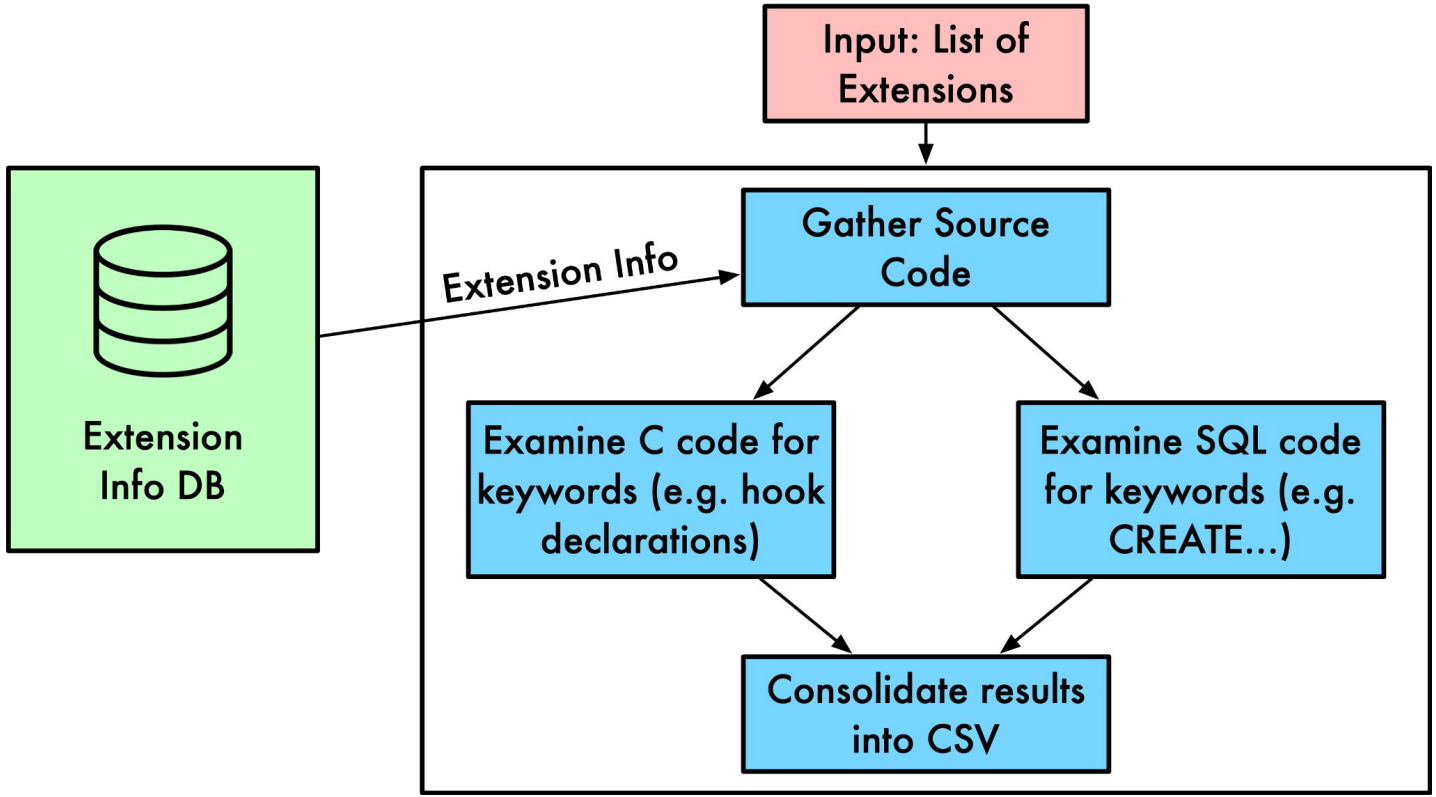
Framework logistics

- Postgres v15.3
- Tested on 114 extensions (one of the following)
  - **`contrib`** directory
  - AWS + Azure + Google Cloud extension offerings
  - 2000+ stars on Github
- Python + Bash scripts
- Utilizes pg_regress for running extension tests

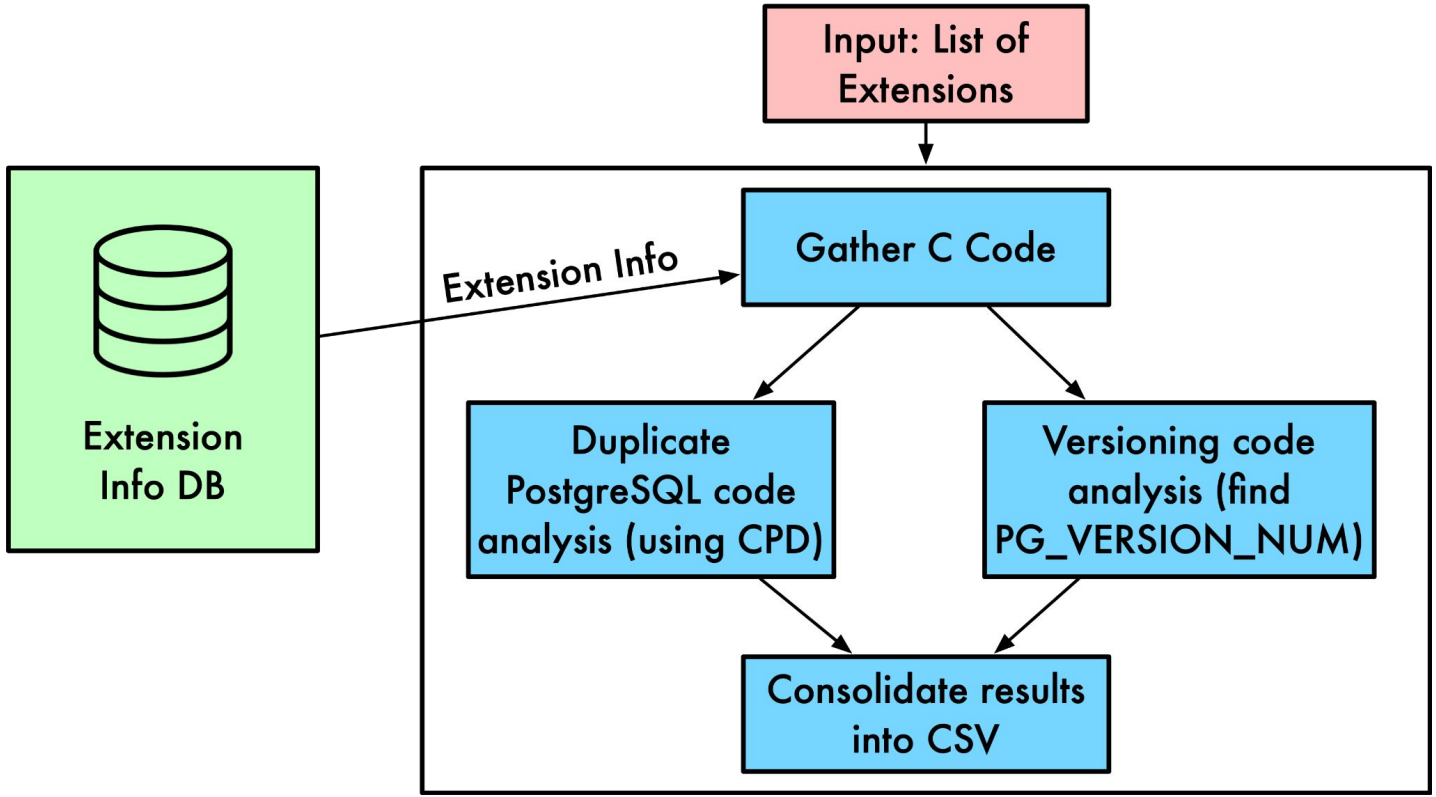
Compatibility analysis



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
. . .
```

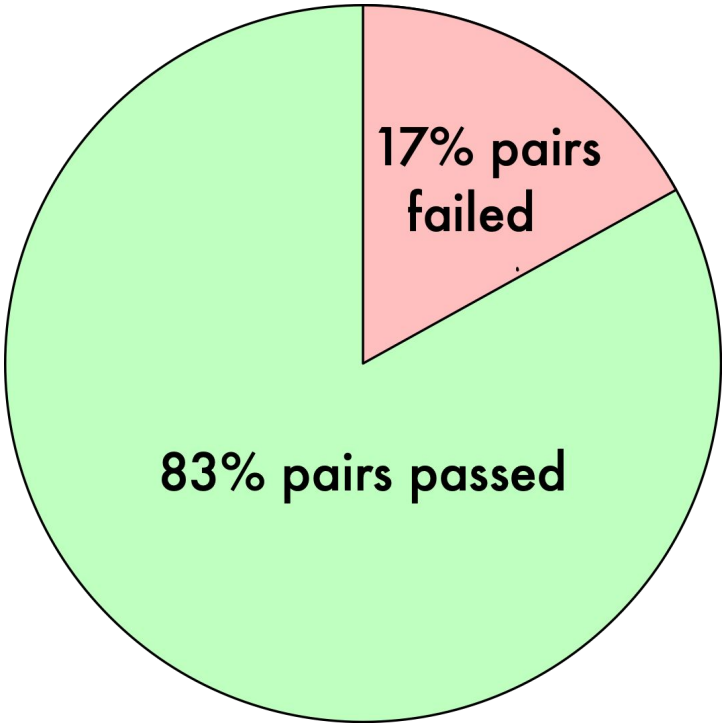
Outline

- Motivation
- Survey
- Analysis
- **Results**
- Discussion

Analysis metrics

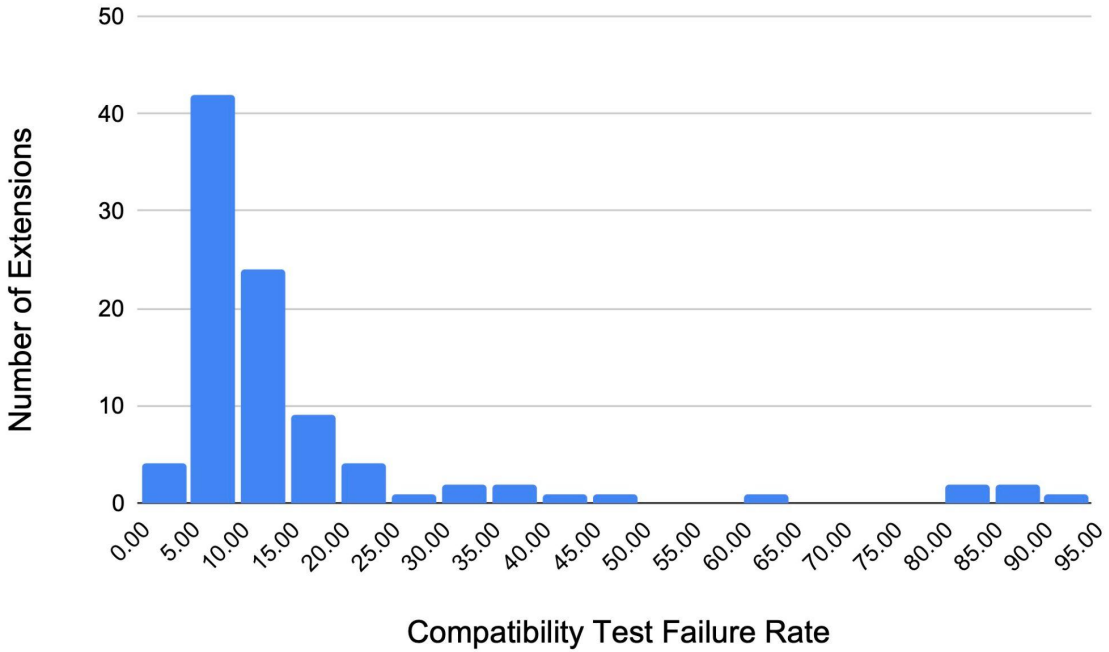
- How common is the incompatibility problem?
  - How often tests failed (in general and per extension)
  - # Pairs where tests failed/# Pairs where an extension was included = Extension failure rate
- What factors could contribute to higher incompatibility rates?
  - Components utilized (functions, types, access methods, external tables, client authentication, query processing, utility commands)
  - Metrics on extensions copying source code
  - Metrics on extensions utilizing versioning

Compatibility analysis



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

Compatibility analysis histogram



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

Compatibility error messages

- Fun reasons they failed

```
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
2023-07-25 05:04:56.945 UTC [687074] DETAIL:  Process 687074 waits for AccessShareLock on relation 17466 of database 16384; blocked by process 687073.
        Process 687073 waits for AccessExclusiveLock on relation 17471 of database 16384; blocked by process 687074.
        Process 687074: truncate
        Process 687073: truncate table t;
2023-07-25 05:04:56.945 UTC [687074] HINT:  See server log for query details.
```

```
INFO:  operator family "gin_bigm_ops" of access method gin contains function gin_bigm_compare_partial(text,text,smallint,internal) with wrong signature for support number 5
 amname |   opcname
--------+--------------
 gin    | gin_bigm_ops
(1 row)
```

```
2023-07-26 11:57:04.922 UTC [2087903] FATAL:  requested tranche is not registered
2023-07-26 11:57:04.925 UTC [2087903] LOG:  database system is shut down
```

Compatibility error messages

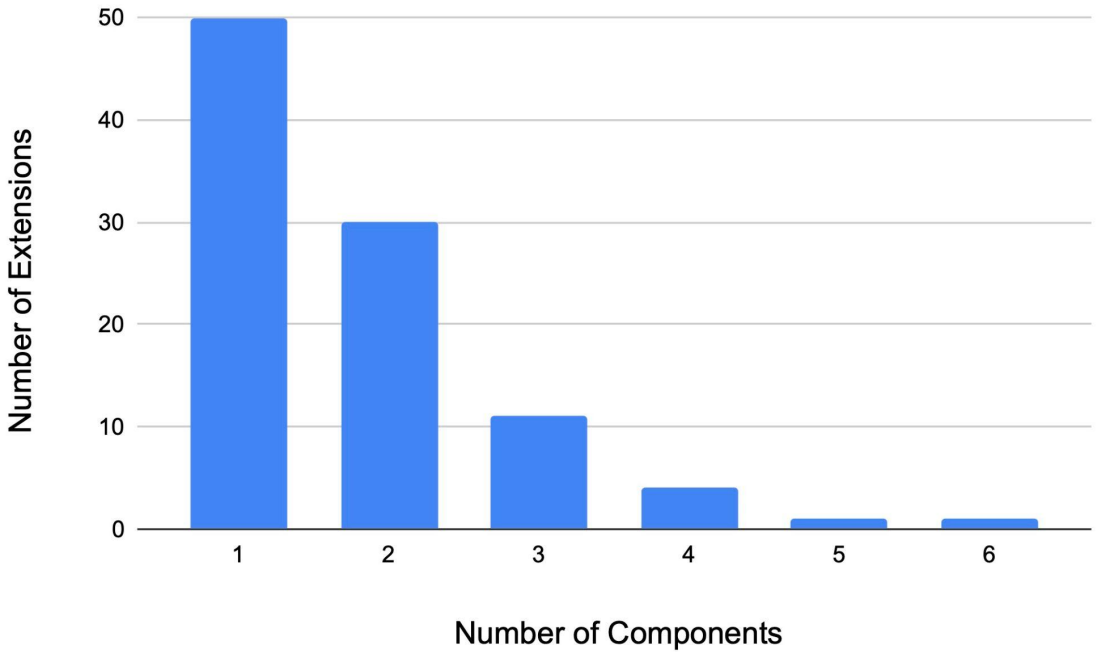
- Fun reasons they failed

```
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
2023-07-25 05:04:56.945 UTC [687074] DETAIL:  Process 687074 waits for AccessShareLock on relation 17466 of database 16384; blocked by process 687073.
        Process 687073 waits for AccessExclusiveLock on relation 17471 of database 16384; blocked by process 687074.
        Process 687074: truncate
```

```
-- 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';
```

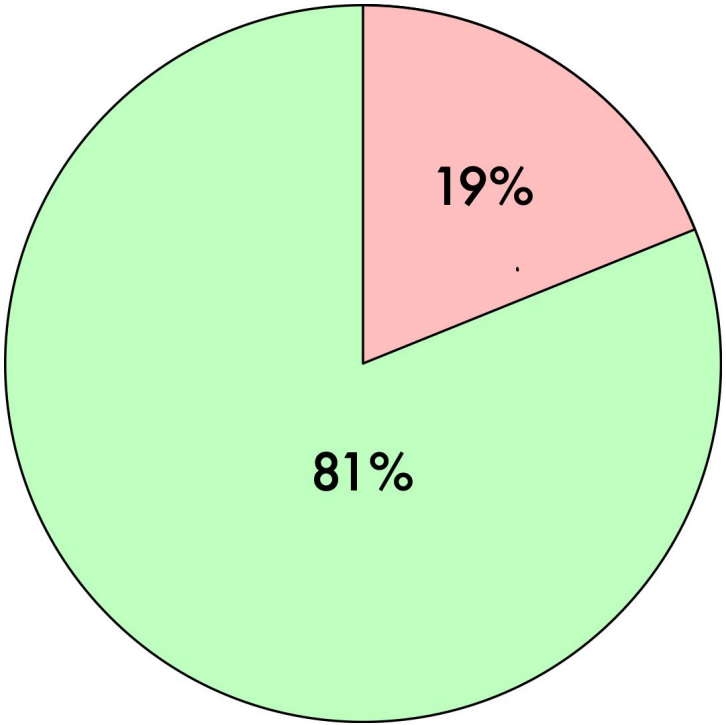
```
2023-07-26 11:57:04.925 UTC [2087903] LOG:  database system is shut down
```

Number of components



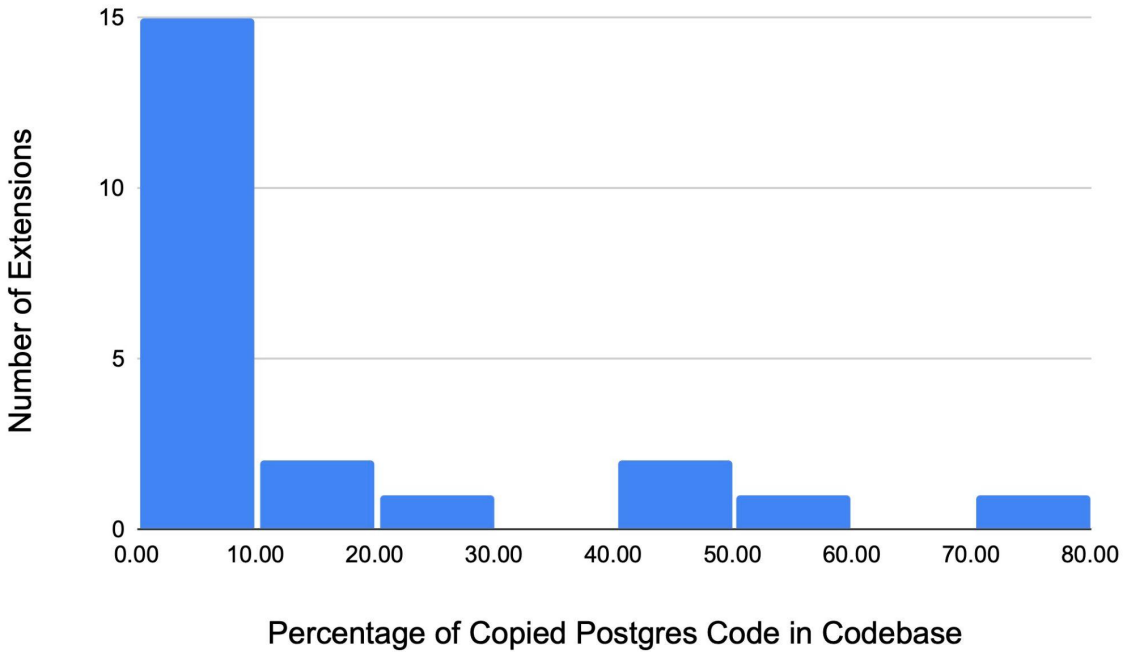
- 86.8% of extensions incorporate UDFs
- **Takeaway: Most extensions utilize a small number of components**

Copied PostgreSQL code



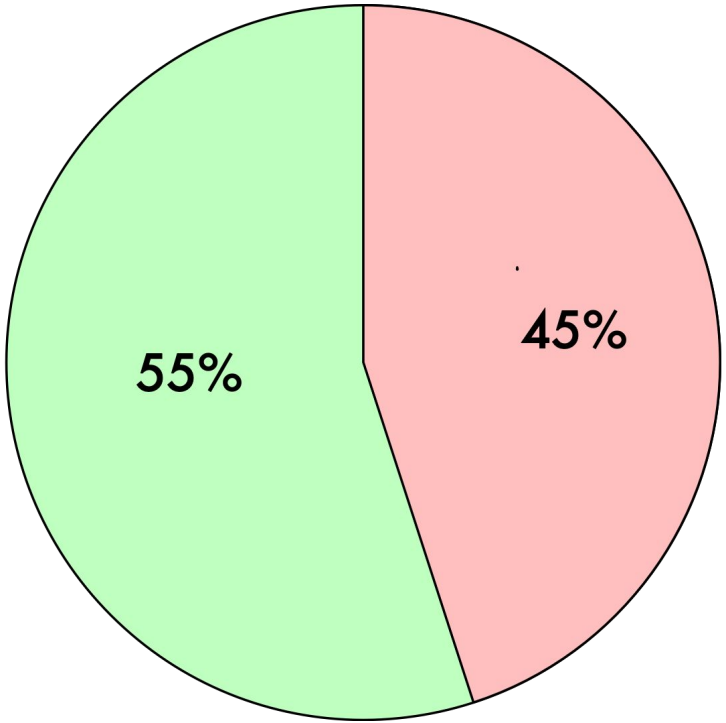
- 19% of extensions copy PostgreSQL code
- **Takeaway: Extensions copying PostgreSQL code is a common phenomenon**

Copied PostgreSQL code histogram



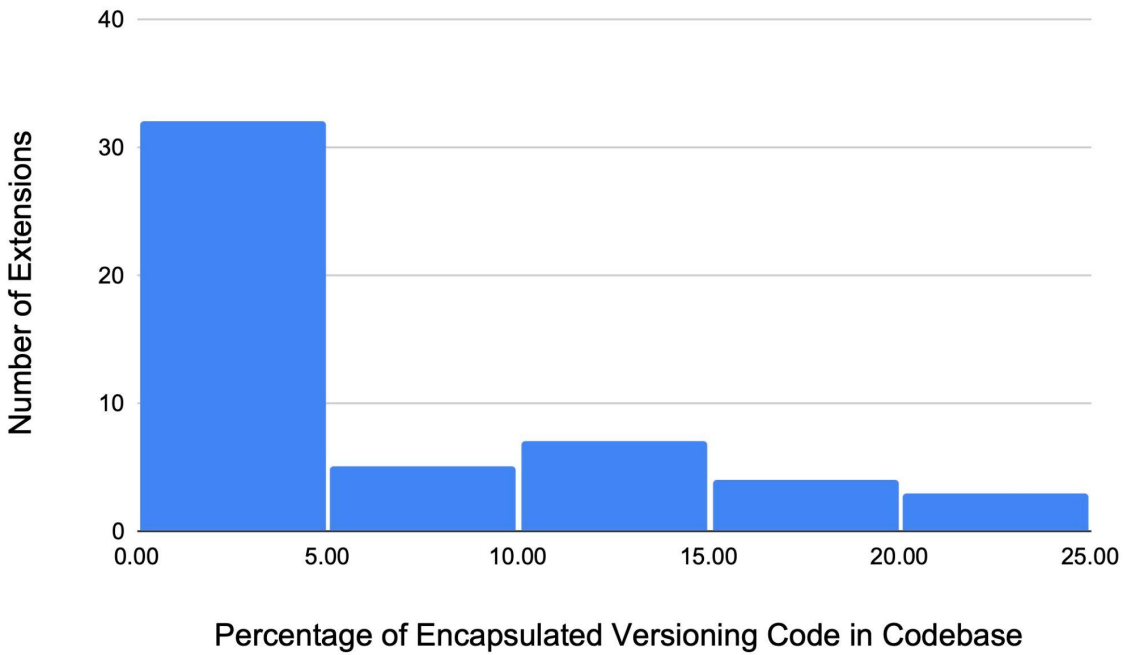
- Measured percentage of extension's C codebase consisting of copied code
- Histogram consists of extensions with > 0% copied code
- Max: pageinspect (75.5%)
- **Takeaway: Most extension codebases do not have lots of copied code**

Versioning code logic



- 45% utilize versioning code logic
- **Takeaway: Extensions utilizing versioning logic is a common phenomenon.**

Versioning code logic histogram



- Measured percentage of LOC encapsulated in versioning macros
- Max: hypopg (23.85%)
- **Takeaway: Most don’t use versioning logic often.**

What is the relationship between compatibility failure rates and extension properties?

Connecting properties to incompatibility

- Ran T-paired test on incompatibility data with extension characteristics
- Found four properties with correlation:
  - Number of types of extensibility utilized
  - Had more than 500 or 750 lines of source code
  - Utilization of versioning logic
  - Usage of ProcessUtility_hook

Outline

- Motivation
- Survey
- Analysis
- Results
- **Discussion**

PostgreSQL: a double edged sword

- Other DBMSs (MySQL/MariaDB, DuckDB, SQLite) also support extensibility
- PostgreSQL's extensibility ecosystem surpasses all in involvement
  - Very flexible interface via hooks
  - Lots of internal support for extensibility
  - Internal building and testing infrastructure
- Lots of safety and maintenance concerns
  - No guarantees that PostgreSQL extensions work with one another
  - Versioning logic to get PostgreSQL extensions working with multiple versions

Suggestion: extension manager

- PostgreSQL does not have an extension manager
- Problems an extension manager could solve:
  - Extensions are responsible for calling other extensions' hooks
  - Extensions need to be loaded in certain orders to work properly
  - Streamline disabling and enabling extensions
  - Matching extension installations to versions

Suggestion: compatibility tool

- Users typically utilize > 1 extension on DB instances
- Smoke tests (or more rigorous tests) of extension compatibility are useful
- Tool helps find undiscovered bugs in your + other extensions
- My tool showed many compatibility errors on established PostgreSQL extensions

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 usage of ProcessUtility_hook and versioning logic
- Suggestions: Extension manager, analysis tools for PostgreSQL

Github Repo: Analysis Framework Tool

Email: abigalekim0417@gmail.com

Appendix: pg_regress

- pg_regress validates by text output
- Some errors in my testing framework are caused by different outputs, but the extensions are both working as intended
- For smoke test purposes a different validation system for pg_regress is necessary

Appendix: Acknowledgements

Thankful to Andy, Marco, Dave and the CMU DB group for helping me out with this presentation and research!