Towards Family-Based Vulnerability Discovery for Highly-Configurable Software Systems

Meeting on Feature-Oriented Software Development 2026, Odense

Tim Bächle, Erik Hofmayer, Tobias Pett, Ina Schaefer | 24th March 2026

Test, Validation, Analysis

Context

```
1  void foo() {
2      int x = source(); // Attacker-controlled.
3      if(x < MAX){ // Does not enforce x >= 0.
4          int y = 0;
5  #ifdef CONFIG_PROCESS_INPUT
6          y = 2 * x;
7      #ifdef CONFIG_SEND_DATA
8          sink(y); // Security-sensitive operation.
9      #endif
10 #endif
11         // ...
12     }
13 }
```

A configurable C function adapted from the example provided by Yamaguchi et al. [Yam+14]

Context

```
1  void foo() {
2      int x = source(); // Attacker-controlled.
3      if(x < MAX){ // Does not enforce x >= 0.
4          int y = 0;
5  #ifdef CONFIG_PROCESS_INPUT
6          y = 2 * x;
7      #ifdef CONFIG_SEND_DATA
8          sink(y); // Security-sensitive operation.
9      #endif
10 #endif
11         // ...
12     }
13 }
```

A configurable C function adapted from the example provided by Yamaguchi et al. [Yam+14]

Variability-Induced Vulnerability (VIV)

A **vulnerability** that is **present in some but not all variants** derivable from a configurable software system [Bäc+25].

Context

```
1  void foo() {
2      int x = source(); // Attacker-controlled.
3      if(x < MAX){ // Does not enforce x >= 0.
4          int y = 0;
5  #ifdef CONFIG_PROCESS_INPUT
6          y = 2 * x;
7      #ifdef CONFIG_SEND_DATA
8          sink(y); // Security-sensitive operation.
9      #endif
10 #endif
11         // ...
12     }
13 }
```

A configurable C function adapted from the example provided by Yamaguchi et al. [Yam+14]

Popular Analysis Strategies [Thü+14]

- **Product-Based**: Derive variants and analyze them individually
- **Family-Based**: Analyze a configurable system as a whole

Variability-Induced Vulnerability (VIV)

A **vulnerability** that is **present in some but not all variants** derivable from a configurable software system [Bäc+25].

Context

```
1  void foo() {
2      int x = source(); // Attacker-controlled.
3      if(x < MAX){ // Does not enforce x >= 0.
4          int y = 0;
5  #ifdef CONFIG_PROCESS_INPUT
6          y = 2 * x;
7      #ifdef CONFIG_SEND_DATA
8          sink(y); // Security-sensitive operation.
9      #endif
10 #endif
11         // ...
12     }
13 }
```

A configurable C function adapted from the example provided by Yamaguchi et al. [Yam+14]

Variability-Induced Vulnerability (VIV)

A **vulnerability** that is **present in some but not all variants** derivable from a configurable software system [Bäc+25].

Popular Analysis Strategies [Thü+14]

- **Product-Based**: Derive variants and analyze them individually
- **Family-Based**: Analyze a configurable system as a whole

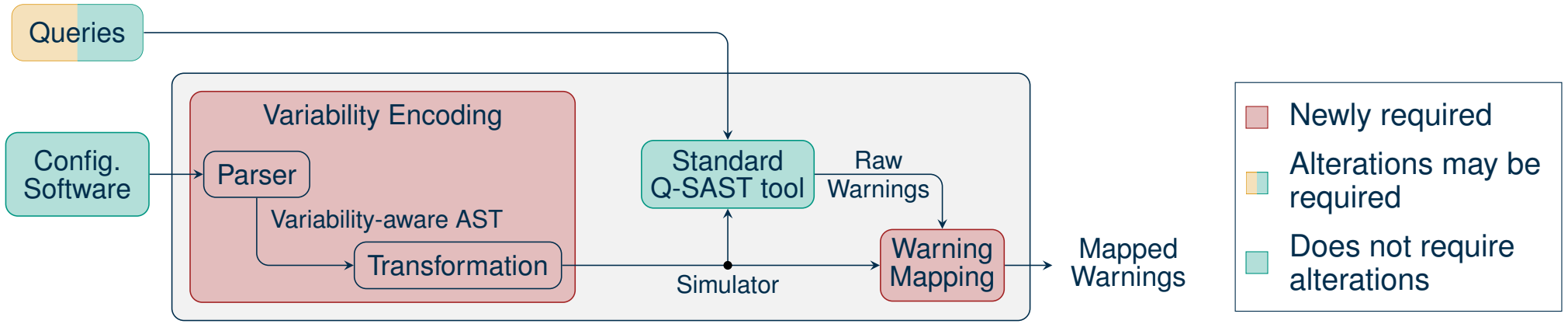
Family-Based Analysis

- \+ Promises **Completeness** [Ape+13; Lie+13; von+18]
- \+ Promises **Scalability** [Ape+13; Lie+13; von+18]
- ϟ Static **analysis tools** typically **cannot cope with variability** [Fer+16; Pat23; Sch+22]

How can we change that for query-based (Q-SAST) tools?

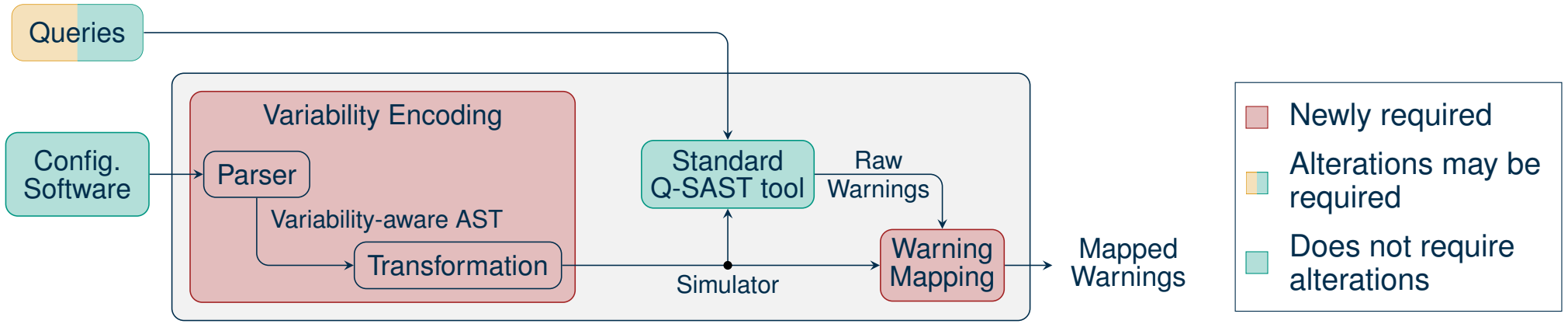
Lifting Strategies [los+17; Pat23; Thü+12; von+16]

Lifting by Simulation

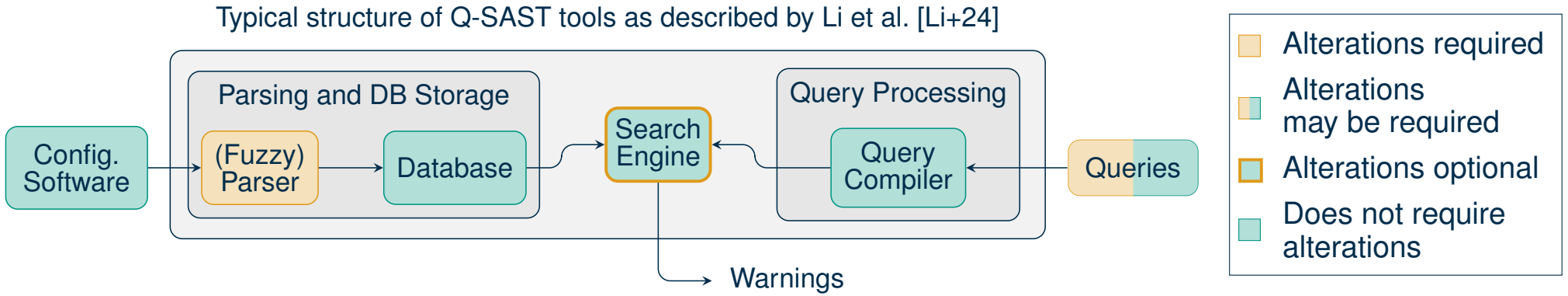


Lifting Strategies [los+17; Pat23; Thü+12; von+16]

Lifting by Simulation



Lifting by Extension



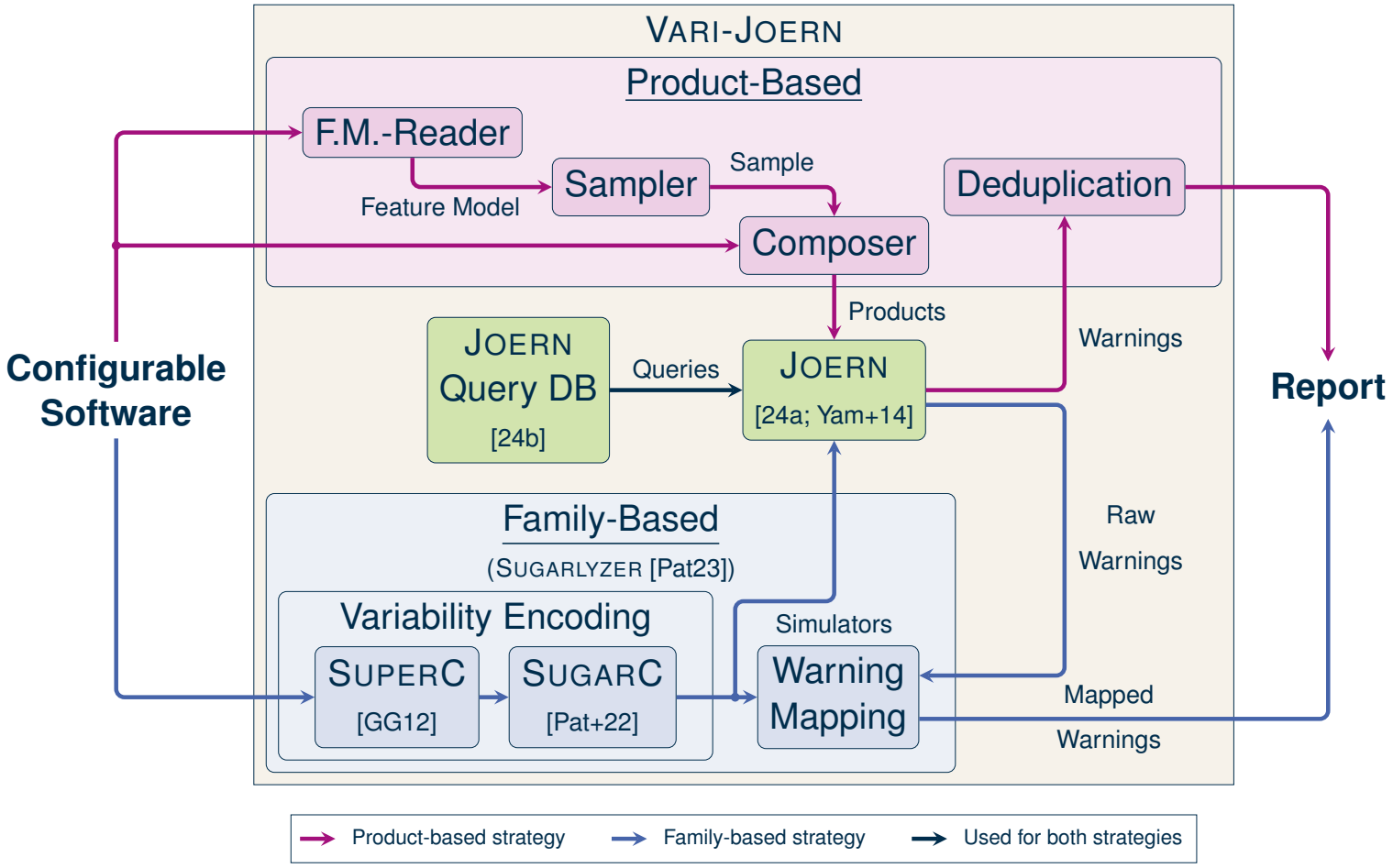
Implementing Lifting by Simulation

Idea

- **Extend VARI-JOERN** [Bäc+25] with a family-based analysis strategy based on lifting by simulation
- **Realized using SUGARC** by Patterson et al. [Pat+22] for variability encoding

Evaluation

- **Compare the capabilities** between product- and family-based vulnerability discovery



Implementing Lifting by Simulation

Idea

- **Extend VARI-JOERN** [Bäc+25] with a family-based analysis strategy based on lifting by simulation
- **Realized using SUGARC** by Patterson et al. [Pat+22] for variability encoding

Evaluation

- **Compare the capabilities** between product- and family-based vulnerability discovery

Comparison of vulnerability warnings between product-based (PB) and family-based (FB) analysis:

System	t	Share Type 1	Share Type 2	Share Type 3	Total
AXTLS	2, 3	95.28% (1010)	0.94% (10)	3.77% (40)	1060
TOYBOX	2, 3	78.83% (5770)	14.75% (1080)	6.42% (470)	7320
BUSYBOX	2	34.59% (4991)	59.82% (8632)	5.60% (808)	14431
	3	34.76% (5019)	59.83% (8639)	5.41% (781)	14439

Type 1: Both PB and FB Type 2: Only PB Type 3: Only FB

Implementing Lifting by Simulation

Idea

- **Extend VARI-JOERN** [Bäc+25] with a family-based analysis strategy based on lifting by simulation
- **Realized using SUGARC** by Patterson et al. [Pat+22] for variability encoding

Evaluation

- **Compare the capabilities** between product- and family-based vulnerability discovery

Comparison of vulnerability warnings between product-based (PB) and family-based (FB) analysis:

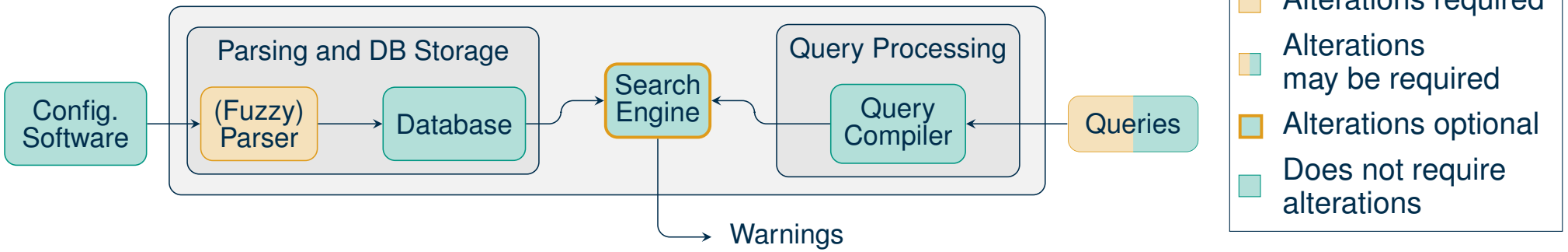
System	t	Share Type 1	Share Type 2	Share Type 3	Total
AXTLS	2, 3	95.28% (1010)	0.94% (10)	3.77% (40)	1060
TOYBOX	2, 3	78.83% (5770)	14.75% (1080)	6.42% (470)	7320
BUSYBOX	2	34.59% (4991)	59.82% (8632)	5.60% (808)	14431
	3	34.76% (5019)	59.83% (8639)	5.41% (781)	14439

Main reason for Type 2: **Limitations of variability encoding**

Implementing Lifting by Extension

Challenges

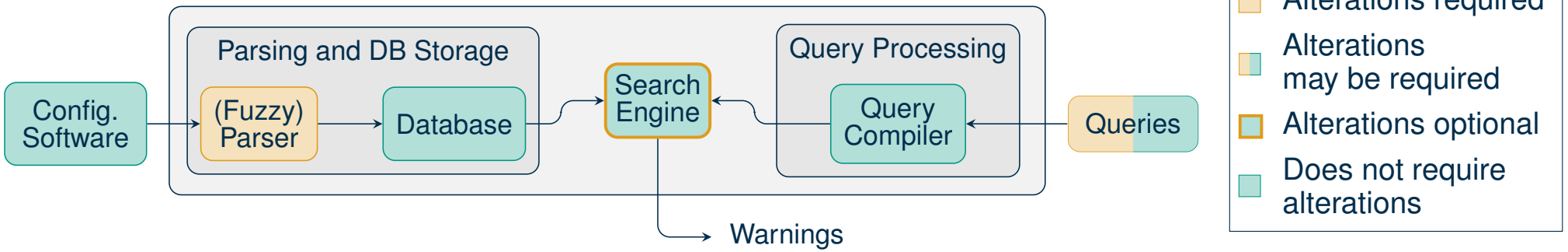
Typical structure of Q-SAST tools as described by Li et al. [Li+24]



Implementing Lifting by Extension

Challenges

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



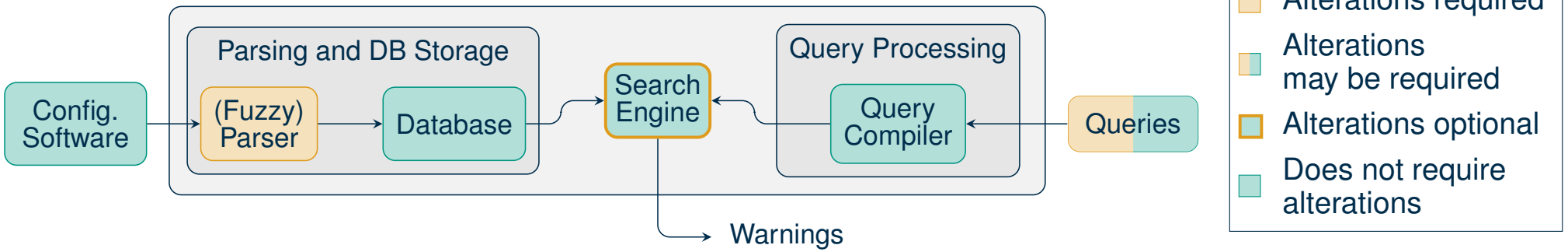
1. Var.-Aware Data Structure

How can **variability** be **integrated into** the **source code representation** employed by the chosen Q-SAST tool?

Implementing Lifting by Extension

Challenges

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



1. Var.-Aware Data Structure

How can **variability** be **integrated into** the **source code representation** employed by the chosen Q-SAST tool?

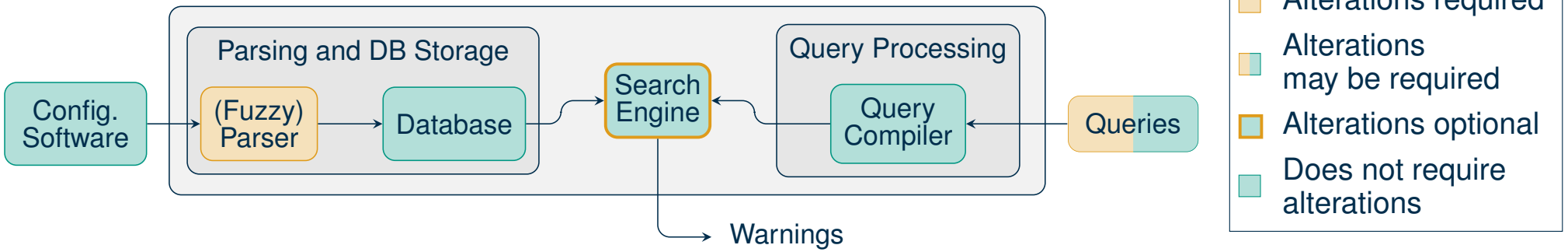
2. Parsing

How can the **variability-aware source code representation be constructed** from variable source code?

Implementing Lifting by Extension

Challenges

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



1. Var.-Aware Data Structure

How can **variability** be **integrated into** the **source code representation** employed by the chosen Q-SAST tool?

2. Parsing

How can the **variability-aware source code representation be constructed** from variable source code?

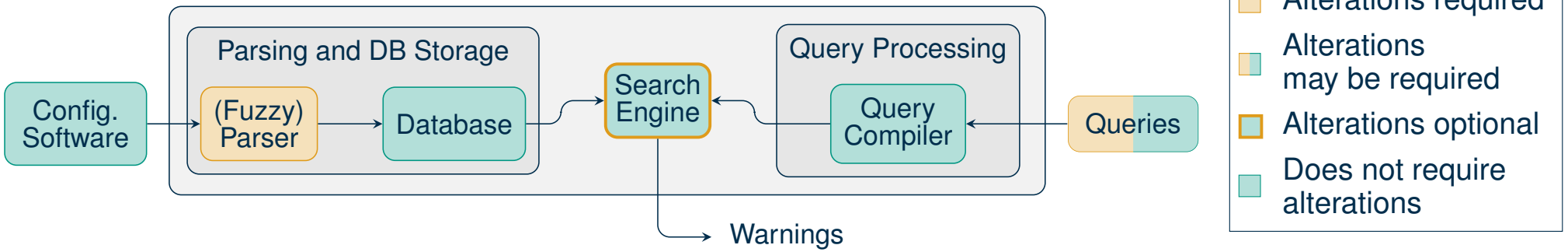
3. Querying

How can the **variability-aware source code representation be queried** for potentially vulnerable source code patterns?

Implementing Lifting by Extension

Challenges

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



1. Var.-Aware Data Structure

How can **variability** be **integrated into** the **source code representation** employed by the chosen Q-SAST tool?

2. Parsing

How can the **variability-aware source code representation be constructed** from variable source code?

3. Querying

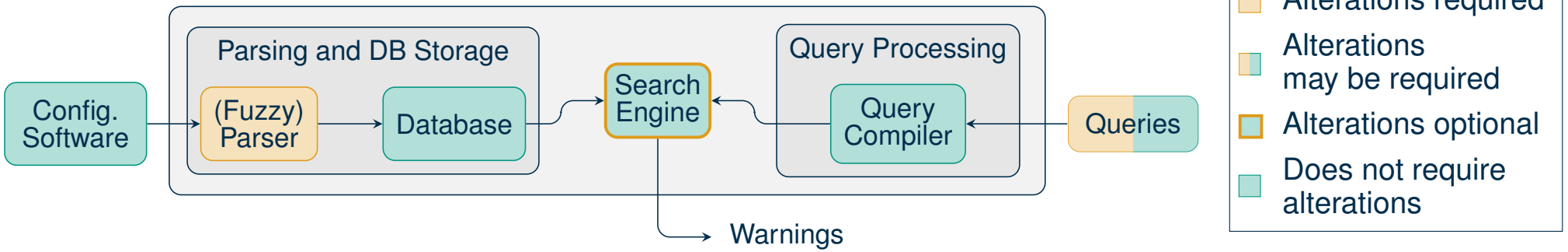
How can the **variability-aware source code representation be queried** for potentially vulnerable source code patterns?

How can these challenges be addressed for the Q-SAST tool JOERN?

Implementing Lifting by Extension

Challenges

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



1. Var.-Aware Data Structure

How can **variability** be **integrated into** the **source code representation** employed by the chosen Q-SAST tool?

2. Parsing

How can the **variability-aware source code representation be constructed** from variable source code?

3. Querying

How can the **variability-aware source code representation be queried** for potentially vulnerable source code patterns?

How can these challenges be addressed for the Q-SAST tool JOERN?

Implementing Lifting by Extension

1. Variability-Aware Data Structure

- JOERN uses **Code Property Graphs (CPGs)** as underlying data structure [Yam+14]
- A **CPG combines** a program's:
  - Abstract Syntax Tree (AST)
  - Control-Flow Graph (CFG)
  - Program Dependence Graph (PDG)

Implementing Lifting by Extension

1. Variability-Aware Data Structure

- JOERN uses **Code Property Graphs (CPGs)** as underlying data structure [Yam+14]
- A **CPG combines** a program's:
  - Abstract Syntax Tree (AST)
  - Control-Flow Graph (CFG)
  - Program Dependence Graph (PDG)

- AST, CFG, and PDG need to be made variability-aware
- Two popular options:
  - Dedicated **choice nodes** [GG12]
  - Edges with **presence condition labels** [Ken+10; Lie+13]

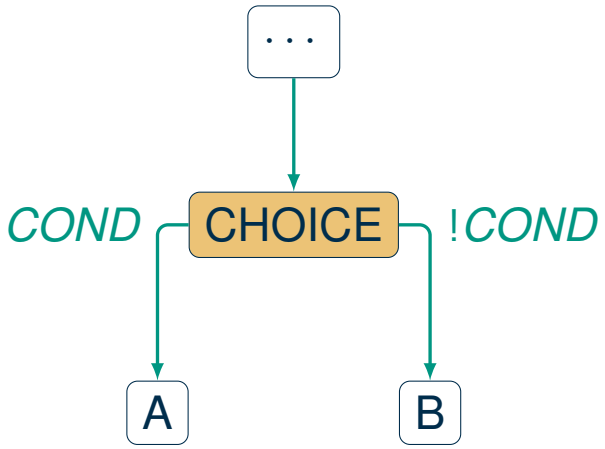
Implementing Lifting by Extension

1. Variability-Aware Data Structure

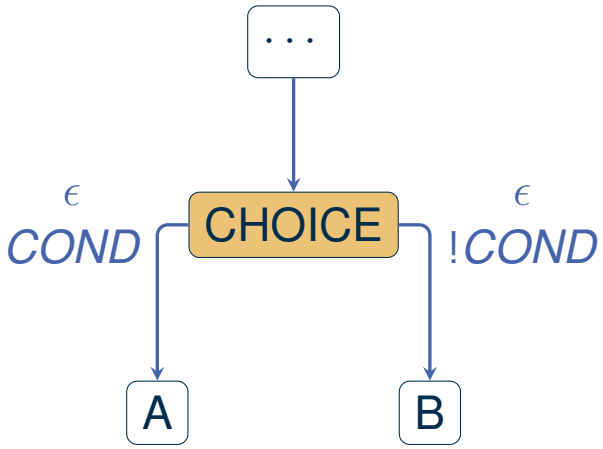
- JOERN uses **Code Property Graphs (CPGs)** as underlying data structure [Yam+14]
- A **CPG combines** a program's:
  - Abstract Syntax Tree (AST)
  - Control-Flow Graph (CFG)
  - Program Dependence Graph (PDG)

- AST, CFG, and PDG need to be made variability-aware
- Two popular options:
  - Dedicated **choice nodes** [GG12]
  - Edges with **presence condition labels** [Ken+10; Lie+13]

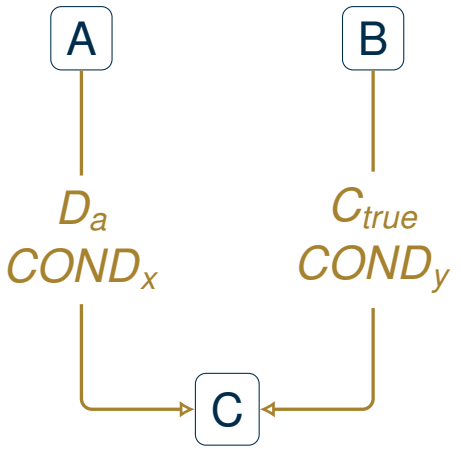
Abstract Syntax Tree



Control Flow Graph



Program Dependence Graph

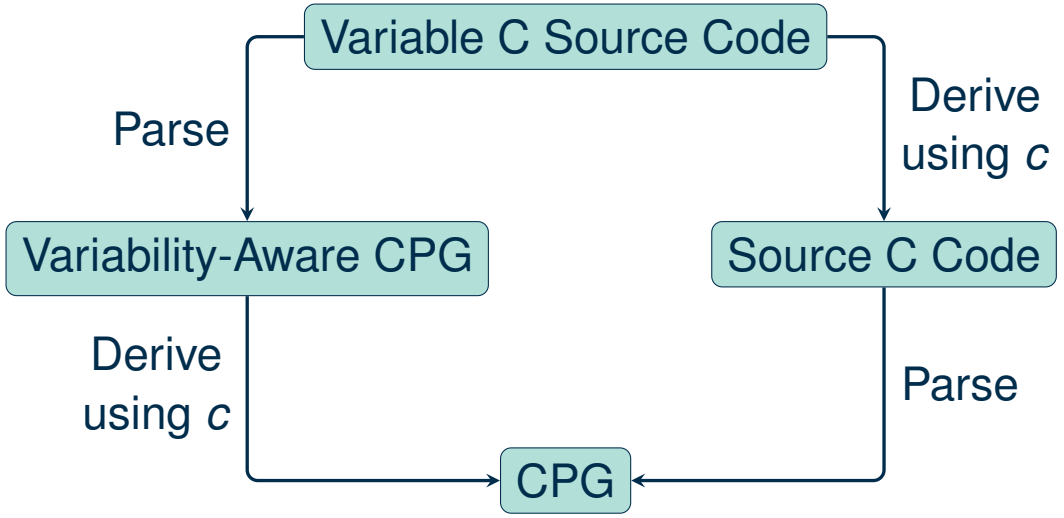


Implementing Lifting by Extension

1. Variability-Aware Data Structure

A Note on Correctness

- **Ideally**, we would show that the **commuting diagram holds**
- Intuition suggests this
- A formal proof has yet to be performed

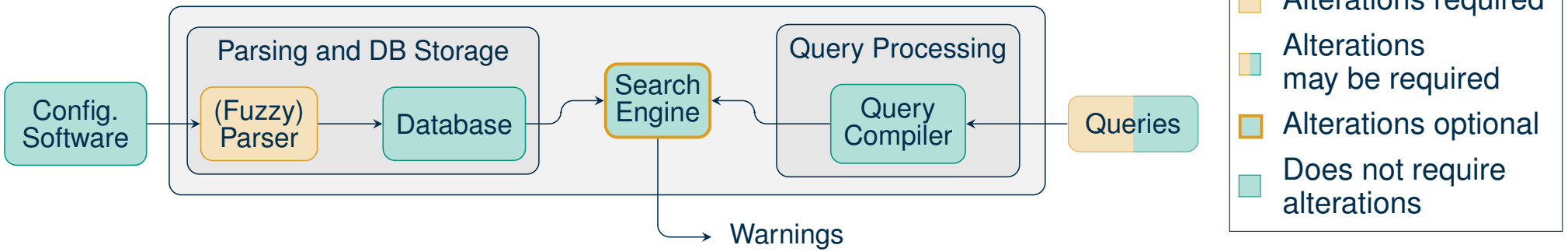


For arbitrary variable source code and an arbitrary (but valid) configuration *c*

Implementing Lifting by Extension

Challenges

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



1. Var.-Aware Data Structure

How can **variability** be **integrated into** the **source code representation** employed by the chosen Q-SAST tool?

2. Parsing

How can the **variability-aware source code representation be constructed** from variable source code?

3. Querying

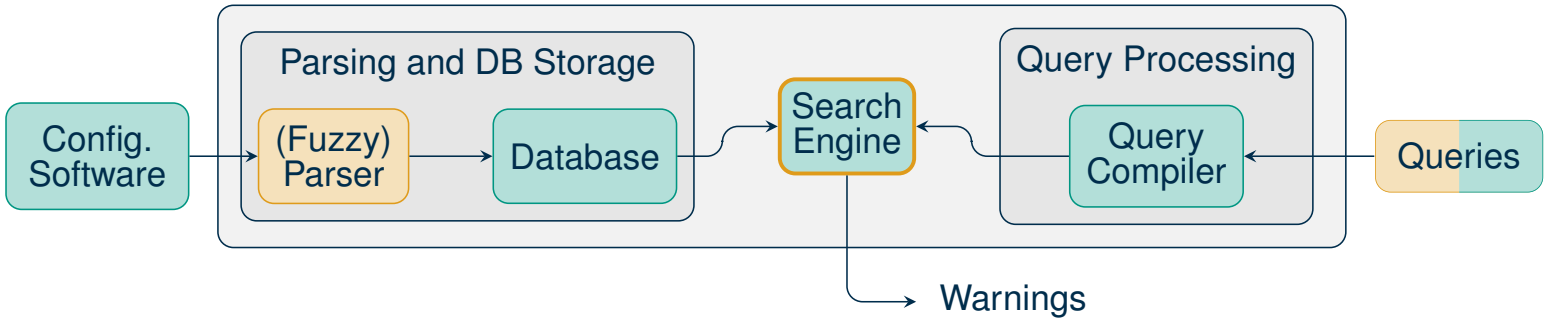
How can the **variability-aware source code representation be queried** for potentially vulnerable source code patterns?

How can these challenges be addressed for the Q-SAST tool JOERN?

Implementing Lifting by Extension

2. Parsing — Problem and Solutions

Typical structure of Q-SAST tools as described by Li et al. [Li+24]

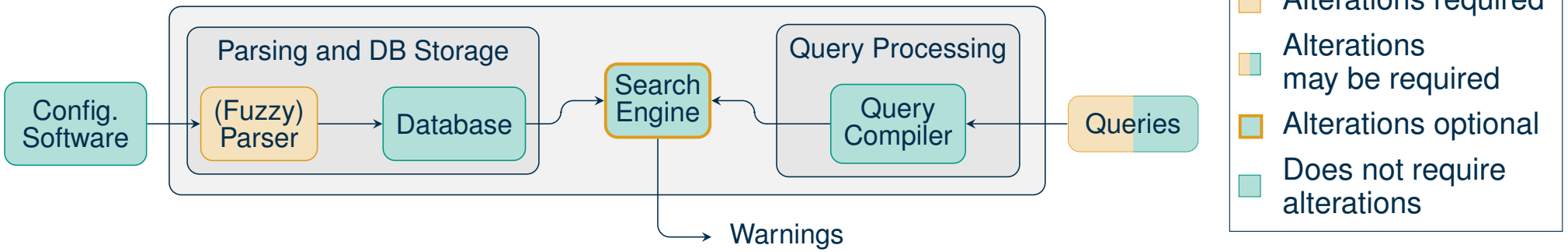


- Alterations required
- Alterations may be required
- Alterations optional
- Does not require alterations

Implementing Lifting by Extension

2. Parsing — Problem and Solutions

Typical structure of Q-SAST tools as described by Li et al. [Li+24]

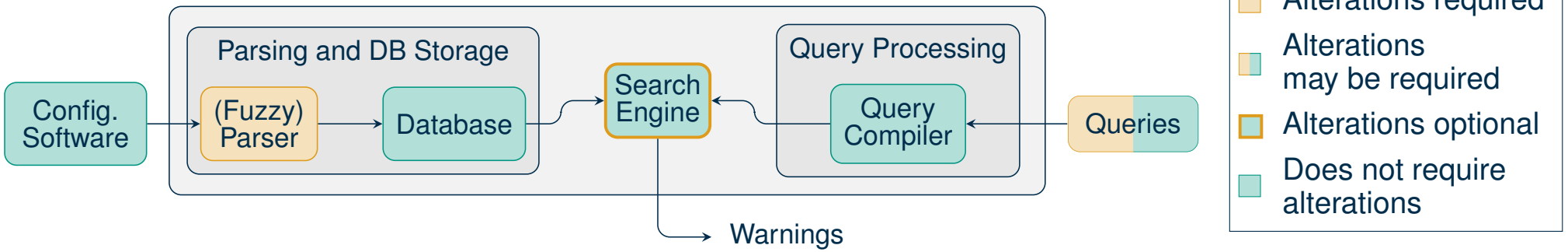


Two General Options

Implementing Lifting by Extension

2. Parsing — Problem and Solutions

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



Two General Options

1. Altering Existing Parsing Infrastructure

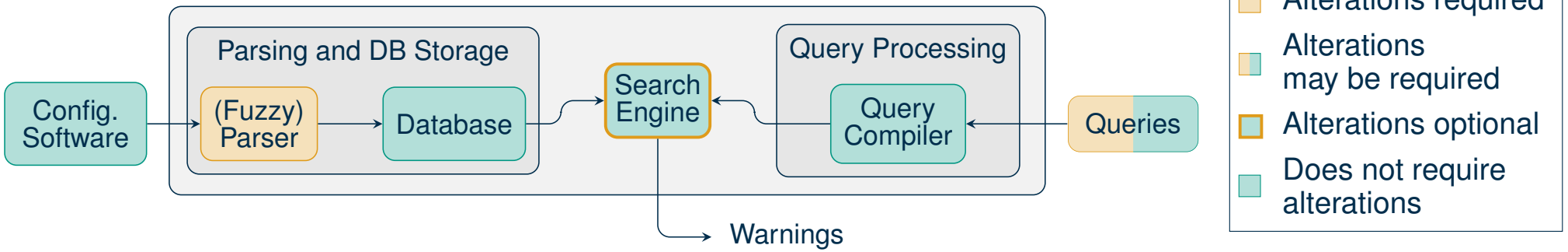
- JOERN already has a front end for parsing C into a CPG
- This front end is variability-oblivious

⇒ Can we make it variability-aware (i.e., capable of building a VA-CPG)?

Implementing Lifting by Extension

2. Parsing — Problem and Solutions

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



Two General Options

1. Altering Existing Parsing Infrastructure

- JOERN already has a front end for parsing C into a CPG
- This front end is variability-oblivious

⇒ Can we make it variability-aware (i.e., capable of building a VA-CPG)?

2. Reusing Variability-Aware Parsing Solutions

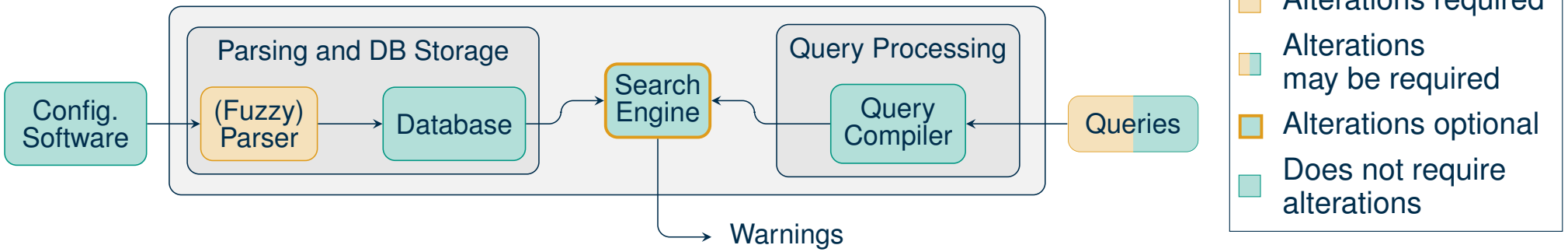
- Parsing variable C into a VA-AST has been its own area of research, e.g.:
  - TYPECHEF [Ken+10]
  - SUPERC [GG12]

⇒ Can we build on existing variability-aware parsing solutions?

Implementing Lifting by Extension

2. Parsing — Problem and Solutions

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



Two General Options

1. Altering Existing Parsing Infrastructure

- JOERN already has a front end for parsing C into a CPG
- This front end is variability-oblivious

⇒ Can we make it variability-aware (i.e., capable of building a VA-CPG)?

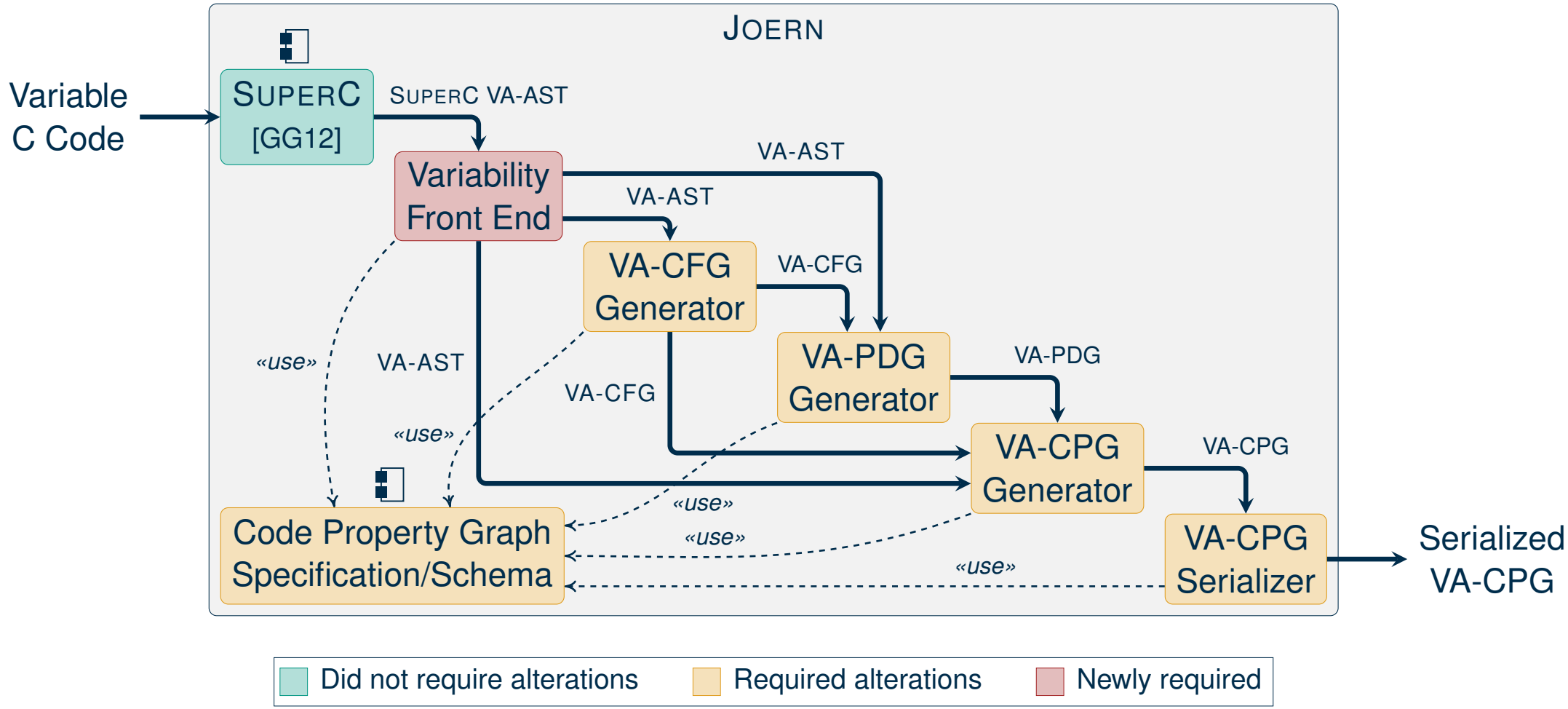
2. Reusing Variability-Aware Parsing Solutions

- Parsing variable C into a VA-AST has been its own area of research, e.g.:
  - TYPECHEF [Ken+10]
  - SUPERC [GG12]

⇒ Can we build on existing variability-aware parsing solutions?

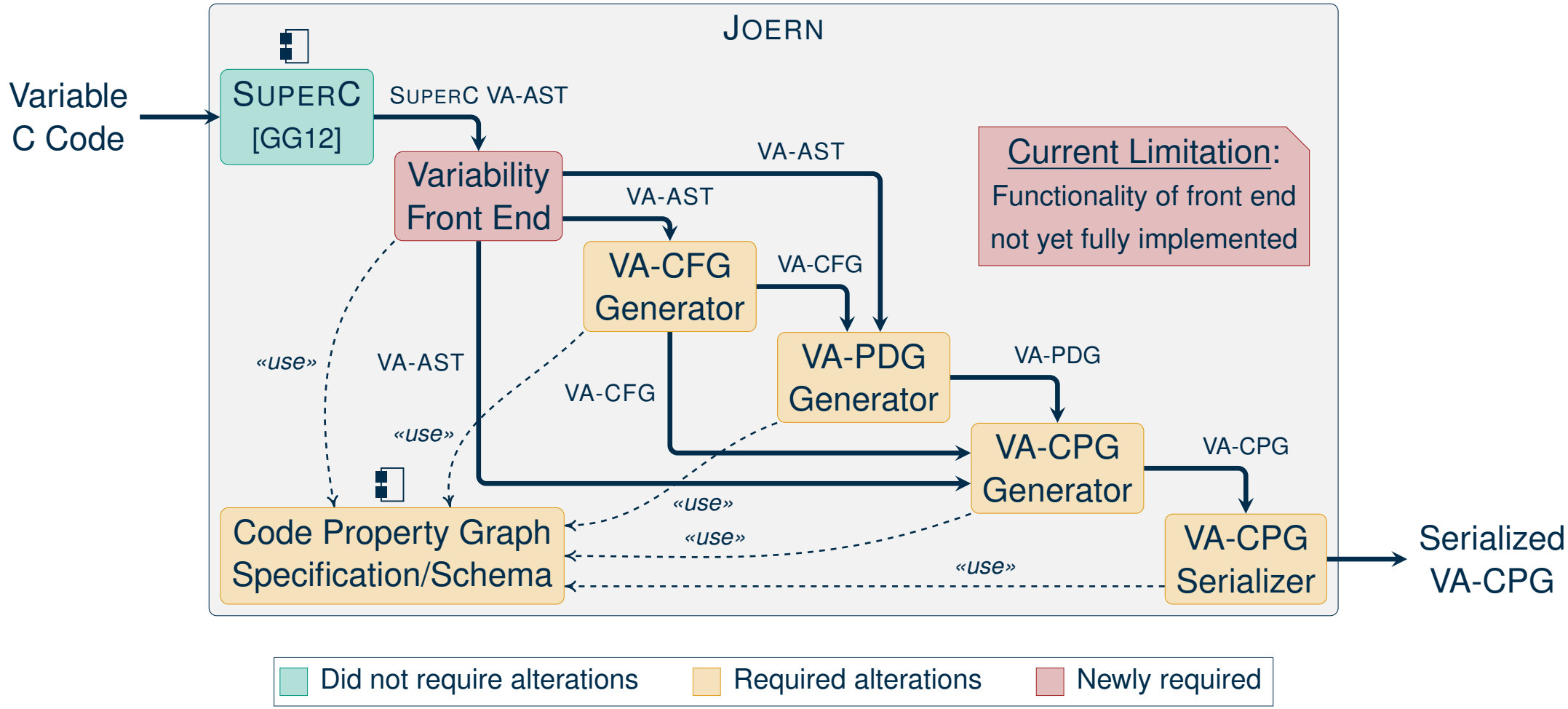
Implementing Lifting by Extension

2. Parsing — Implementation



Implementing Lifting by Extension

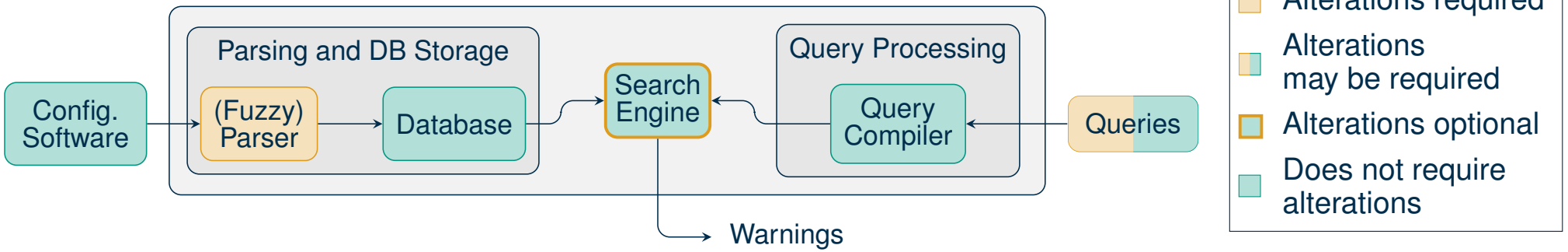
2. Parsing — Implementation



Implementing Lifting by Extension

Challenges

Typical structure of Q-SAST tools as described by Li et al. [Li+24]



1. Var.-Aware Data Structure

How can **variability** be **integrated into** the **source code representation** employed by the chosen Q-SAST tool?

2. Parsing

How can the **variability-aware source code representation be constructed** from variable source code?

3. Querying

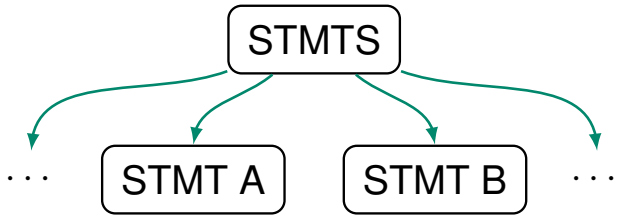
How can the **variability-aware source code representation be queried** for potentially vulnerable source code patterns?

How can these challenges be addressed for the Q-SAST tool JOERN?

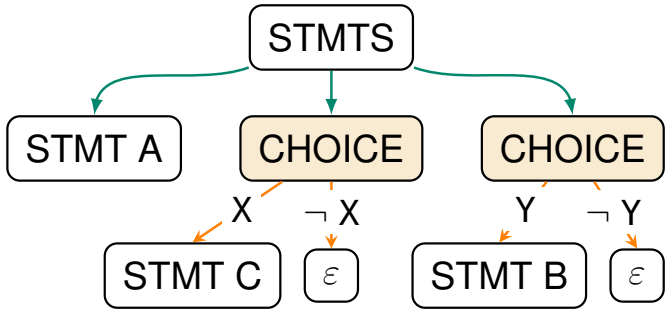
Implementing Lifting by Extension

3. Querying — Potential Problems

A potentially dangerous source code pattern:[a]



The pattern occurring in a variability-aware AST:

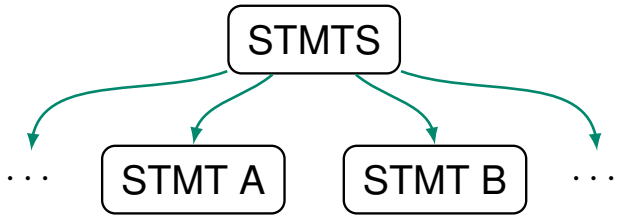


[a] Without loss of generality, we focus on an AST pattern.

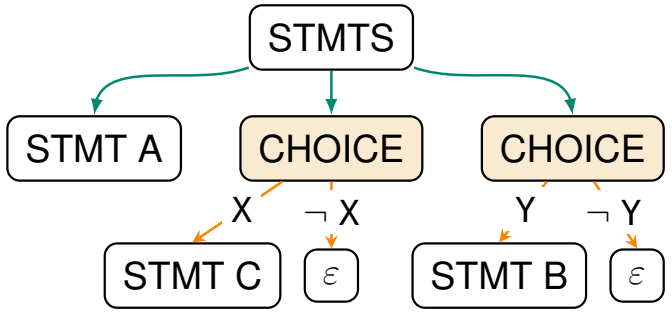
Implementing Lifting by Extension

3. Querying — Potential Problems

A potentially dangerous source code pattern:[a]



The pattern occurring in a variability-aware AST:



[a] Without loss of generality, we focus on an AST pattern.

Variability-Oblivious Queries

- Do not take variability into account
- \+ Easy to specify (existing queries can be reused)
- − Can lead to false negatives due to variability

Variability-Aware Queries

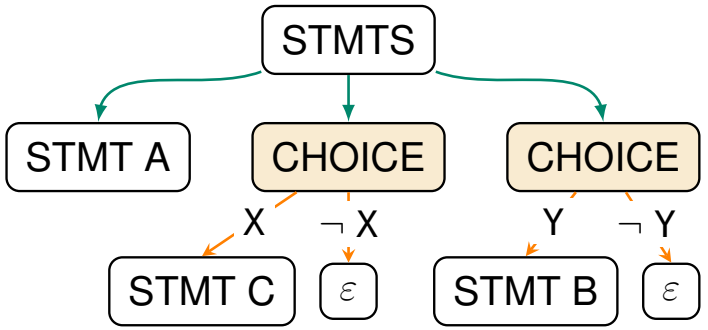
- Take variability into account
- \+ Can avoid false negatives
- − Harder to specify (user has to be aware of variability)

Implementing Lifting by Extension

3. Querying — Possible Solutions

Adjusting the Queries

- Some queries are not affected by variability
  ⇝ `(cpg.method("(?i)gets").callIn).l`
- Others might need to be made variability-aware



Implementing Lifting by Extension

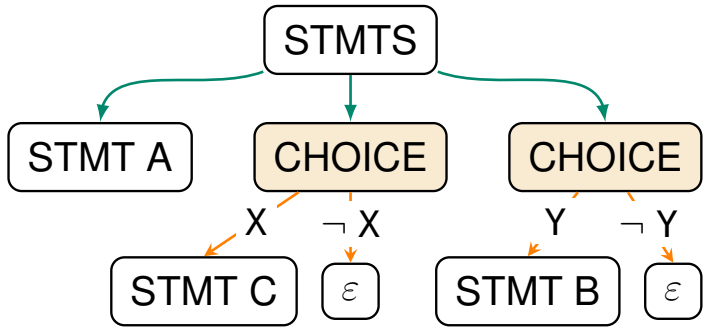
3. Querying — Possible Solutions

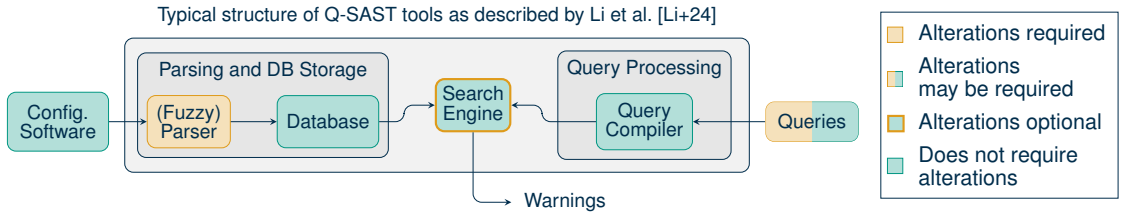
Adjusting the Queries

- Some queries are not affected by variability
  ⇝ `(cpg.method("(?i)gets").callIn).l`
- Others might need to be made variability-aware

Adjusting the Search Engine

- Make search engine aware of variability (e.g., via choice nodes)
- Would avoid having to adjust certain queries





Implementing Lifting by Extension

3. Querying — Possible Solutions

Adjusting the Queries

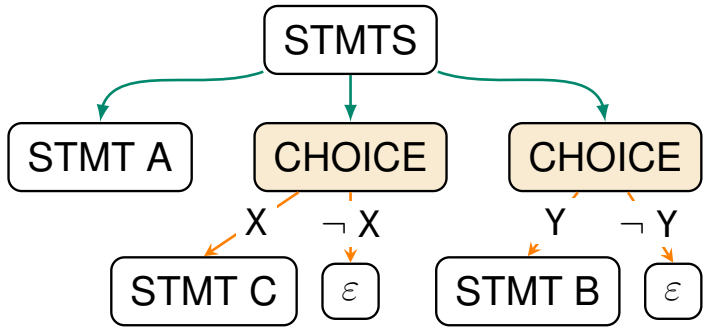
- Some queries are not affected by variability
  ⇝ `(cpg.method("(?i)gets").callIn).l`
- Others might need to be made variability-aware

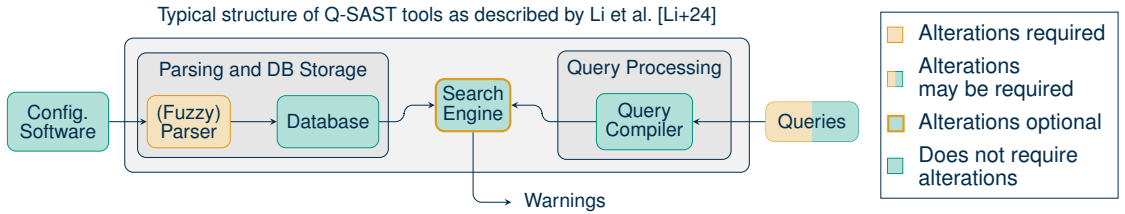
Adjusting the Search Engine

- Make search engine aware of variability (e.g., via choice nodes)
- Would avoid having to adjust certain queries

Preprocessing Queries

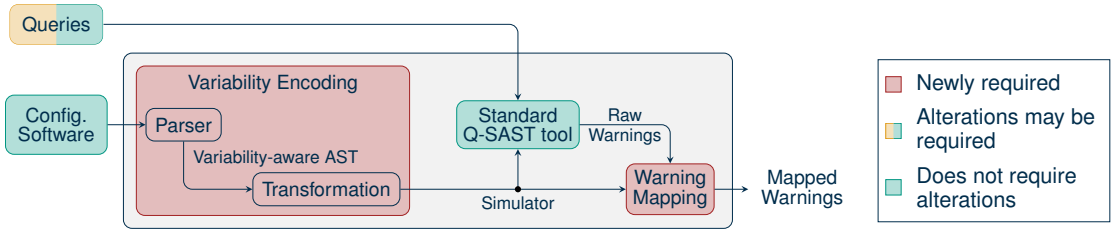
- Adjusting queries and / or search engine is tedious
- Can we make queries variability-aware by preprocessing them?





Summary

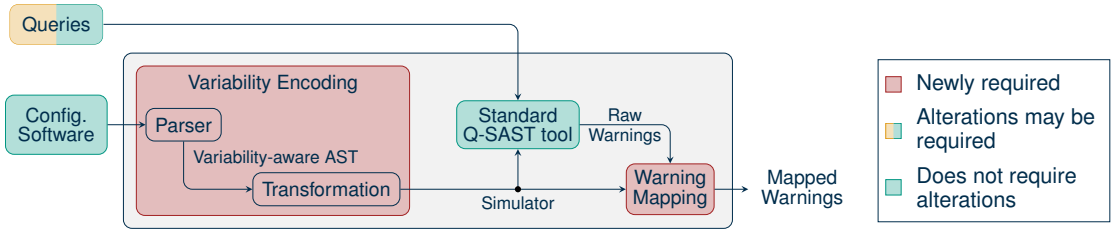
Lifting by Simulation



- ⊕ Low implementation effort and independent of tool
- ⊖ Additional analysis effort and potential imprecision

Summary

Lifting by Simulation



- ⊕ Low implementation effort and independent of tool
- ⊖ Additional analysis effort and potential imprecision

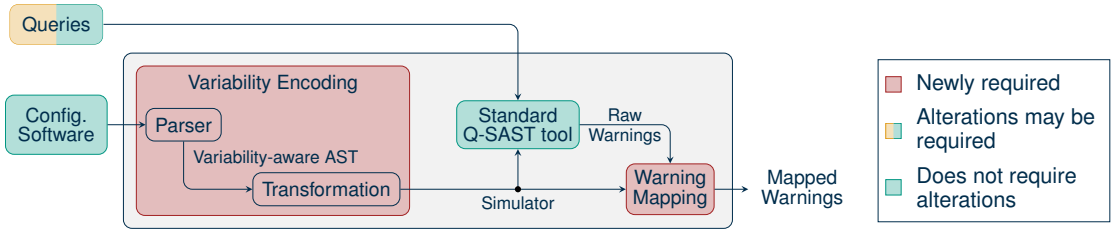
System	t	Share Type 1	Share Type 2	Share Type 3	Total
axTLS	2, 3	95.28% (1010)	0.94% (10)	3.77% (40)	1060
Toybox	2, 3	78.83% (5770)	14.75% (1080)	6.42% (470)	7320
BusyBox	2	34.59% (4991)	59.82% (8632)	5.60% (808)	14431
	3	34.76% (5019)	59.83% (8639)	5.41% (781)	14439

Type 1: Both PB and FB Type 2: Only PB Type 3: Only FB

Main reason for Type 2: **Limitations of variability encoding**

Summary

Lifting by Simulation



⊕ Low implementation effort and independent of tool

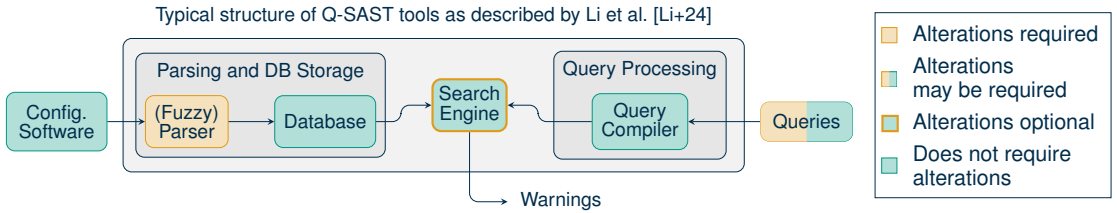
⊖ Additional analysis effort and potential imprecision

System	t	Share Type 1	Share Type 2	Share Type 3	Total
AXTLS	2, 3	95.28% (1010)	0.94% (10)	3.77% (40)	1060
TOYBOX	2, 3	78.83% (5770)	14.75% (1080)	6.42% (470)	7320
BUSYBOX	2	34.59% (4991)	59.82% (8632)	5.60% (808)	14431
	3	34.76% (5019)	59.83% (8639)	5.41% (781)	14439

Type 1: Both PB and FB Type 2: Only PB Type 3: Only FB

Main reason for Type 2: **Limitations of variability encoding**

Lifting by Extension

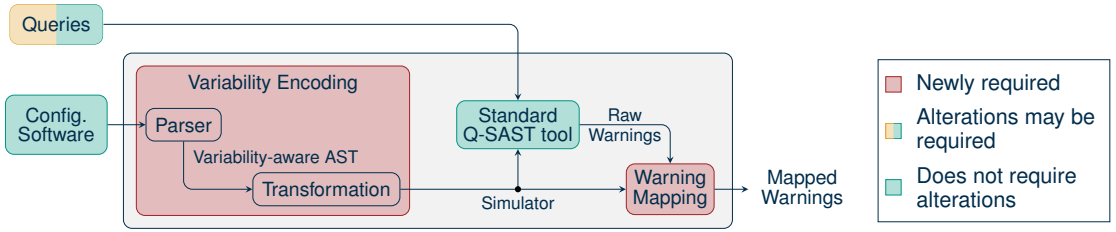


⊕ Precision and Performance?

⊖ Tool-specific and implementation labor-intensive

Summary

Lifting by Simulation



⊕ Low implementation effort and independent of tool

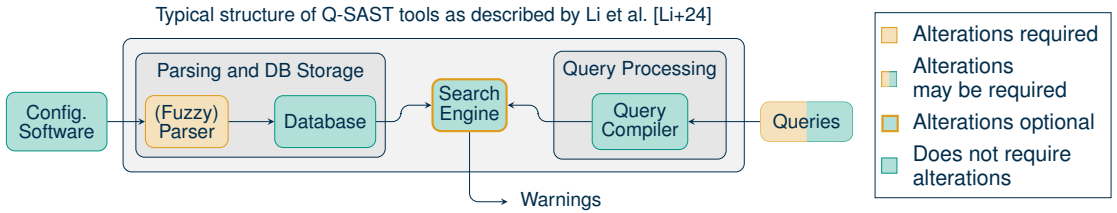
⊖ Additional analysis effort and potential imprecision

System	t	Share Type 1	Share Type 2	Share Type 3	Total
AXTLS	2, 3	95.28% (1010)	0.94% (10)	3.77% (40)	1060
TOYBOX	2, 3	78.83% (5770)	14.75% (1080)	6.42% (470)	7320
BUSYBOX	2	34.59% (4991)	59.82% (8632)	5.60% (808)	14431
	3	34.76% (5019)	59.83% (8639)	5.41% (781)	14439

Type 1: Both PB and FB Type 2: Only PB Type 3: Only FB

Main reason for Type 2: **Limitations of variability encoding**

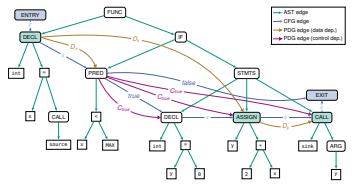
Lifting by Extension



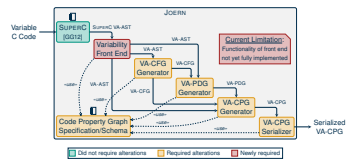
⊕ Precision and Performance?

⊖ Tool-specific and implementation labor-intensive

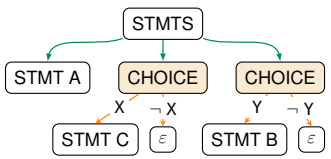
1. Var.-Aware Data Structure



2. Parsing



3. Querying



References I

[24a] *Joern - The Bug Hunter's Workbench*. Website. June 2024. URL: https://joern.io/ (visited on 06/02/2024).

[24b] *Joern Query Database*. Website. 2024. URL: https://queries.joern.io/ (visited on 09/12/2024).

[Ape+13] Sven Apel et al. "Strategies for Product-Line Verification: Case Studies and Experiments". In: *2013 35th International Conference on Software Engineering*. ICSE '13. San Francisco, CA, USA: IEEE, May 2013, pp. 482–491. ISBN: 978-1-4673-3076-3. DOI: 10.1109/ICSE.2013.6606594. URL: http://ieeexplore.ieee.org/document/6606594/ (visited on 04/25/2024).

[Bäc+25] Tim Bächle et al. "Investigating the Effects of T-Wise Interaction Sampling for Vulnerability Discovery in Highly-Configurable Software Systems". In: *Proceedings of the 29th ACM International Systems and Software Product Line Conference*. SPLC '25. A Coruña, Spain: ACM, Aug. 2025. DOI: 10.1145/3744915.3748462.

References II

[Fer+16] Gabriel Ferreira et al. “Do #ifdefs Influence the Occurrence of Vulnerabilities? An Empirical Study of the Linux Kernel”. In: *Proceedings of the 20th International Systems and Software Product Line Conference*. SPLC ’16. Beijing, China: ACM, Sept. 2016, pp. 65–73. ISBN: 978-1-4503-4050-2. DOI: 10.1145/2934466.2934467. URL: https://dl.acm.org/doi/10.1145/2934466.2934467 (visited on 04/09/2025).

[GG12] Paul Gazzillo and Robert Grimm. “SuperC: Parsing All of C by Taming the Preprocessor”. In: *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’12. Beijing, China: ACM, June 2012, pp. 323–334. ISBN: 978-1-4503-1205-9. DOI: 10.1145/2254064.2254103. URL: https://dl.acm.org/doi/10.1145/2254064.2254103 (visited on 04/19/2024).

[Ios+17] Alexandru Florin Iosif-Lazar et al. “Effective Analysis of C Programs by Rewriting Variability”. In: *The Art, Science, and Engineering of Programming* 1.1 (Jan. 2017), p. 1. ISSN: 2473-7321. DOI: 10.22152/programming-journal.org/2017/1/1. URL: http://programming-journal.org/2017/1/1 (visited on 04/19/2024).

References III

[Ken+10] Andy Kenner et al. “TypeChef: Toward Type Checking #ifdef Variability in C”. In: *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development*. FOSD ’10. Eindhoven, Netherlands: ACM, Oct. 2010, pp. 25–32. ISBN: 978-1-4503-0208-1. DOI: 10.1145/1868688.1868693. URL: https://dl.acm.org/doi/10.1145/1868688.1868693 (visited on 04/16/2024).

[Li+24] Zongjie Li et al. “Evaluating C/C++ Vulnerability Detectability of Query-Based Static Application Security Testing Tools”. In: *IEEE Transactions on Dependable and Secure Computing* 21.5 (2024), pp. 1–18. ISSN: 1545-5971, 1941-0018, 2160-9209. DOI: 10.1109/TDSC.2024.3354789. URL: https://ieeexplore.ieee.org/document/10400834/ (visited on 05/08/2024).

[Lie+13] Jörg Liebig et al. “Scalable Analysis of Variable Software”. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE ’13. Saint, Petersburg Russia: ACM, Aug. 2013, pp. 81–91. ISBN: 978-1-4503-2237-9. DOI: 10.1145/2491411.2491437. URL: https://dl.acm.org/doi/10.1145/2491411.2491437 (visited on 04/28/2024).

References IV

[Pat+22] Zachary Patterson et al. “SugarC: Scalable Desugaring of Real-World Preprocessor Usage into Pure C”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. Pittsburgh, Pennsylvania: ACM, May 2022, pp. 2056–2067. ISBN: 978-1-4503-9221-1. DOI: 10.1145/3510003.3512763. URL: https://dl.acm.org/doi/10.1145/3510003.3512763 (visited on 05/15/2024).

[Pat23] Zachery Patterson. “Toward Applying Variability-Oblivious Static Analyses to Software Product Lines”. PhD thesis. Dallas: The University of Texas at Dallas, Dec. 2023. URL: https://utd-ir.tdl.org/items/1623bed4-684c-44e7-94c2-f20cfeb7c976.

[Sch+22] Philipp Dominik Schubert et al. “Static Data-Flow Analysis for Software Product Lines in C: Revoking the Preprocessor’s Special Role”. In: *Automated Software Engineering* 29.35 (May 2022). ISSN: 0928-8910, 1573-7535. DOI: 10.1007/s10515-022-00333-1. URL: https://link.springer.com/10.1007/s10515-022-00333-1 (visited on 04/18/2024).

References V

[Thü+12] Thomas Thüm et al. “Family-Based Deductive Verification of Software Product Lines”. In: *Proceedings of the 11th International Conference on Generative Programming and Component Engineering*. GPCE ’12. Dresden, Germany: ACM, Sept. 2012, pp. 11–20. ISBN: 978-1-4503-1129-8. DOI: 10.1145/2371401.2371404. URL: https://dl.acm.org/doi/10.1145/2371401.2371404 (visited on 06/10/2024).

[Thü+14] Thomas Thüm et al. “A Classification and Survey of Analysis Strategies for Software Product Lines”. In: *ACM Computing Surveys* 47.1 (July 2014), pp. 1–45. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/2580950. URL: https://dl.acm.org/doi/10.1145/2580950 (visited on 04/16/2024).

[von+16] Alexander von Rhein et al. “Variability Encoding: From Compile-Time to Load-Time Variability”. In: *Journal of Logical and Algebraic Methods in Programming* 85.1 (Jan. 2016), pp. 125–145. ISSN: 23522208. DOI: 10.1016/j.jlamp.2015.06.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S2352220815000577 (visited on 04/25/2024).

References VI

[von+18] Alexander von Rhein et al. “Variability-Aware Static Analysis at Scale: An Empirical Study”. In: *ACM Transactions on Software Engineering and Methodology* 27.4 (Oct. 2018), pp. 1–33. ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3280986. URL: https://dl.acm.org/doi/10.1145/3280986 (visited on 04/15/2024).

[Yam+14] Fabian Yamaguchi et al. “Modeling and Discovering Vulnerabilities with Code Property Graphs”. In: *2014 IEEE Symposium on Security and Privacy*. S&P ’14. Berkeley, CA, USA: IEEE, May 2014, pp. 590–604. ISBN: 978-1-4799-4686-0. DOI: 10.1109/SP.2014.44. URL: http://ieeexplore.ieee.org/document/6956589/ (visited on 04/15/2024).