Verifying Extract Method Refactoring in Rust

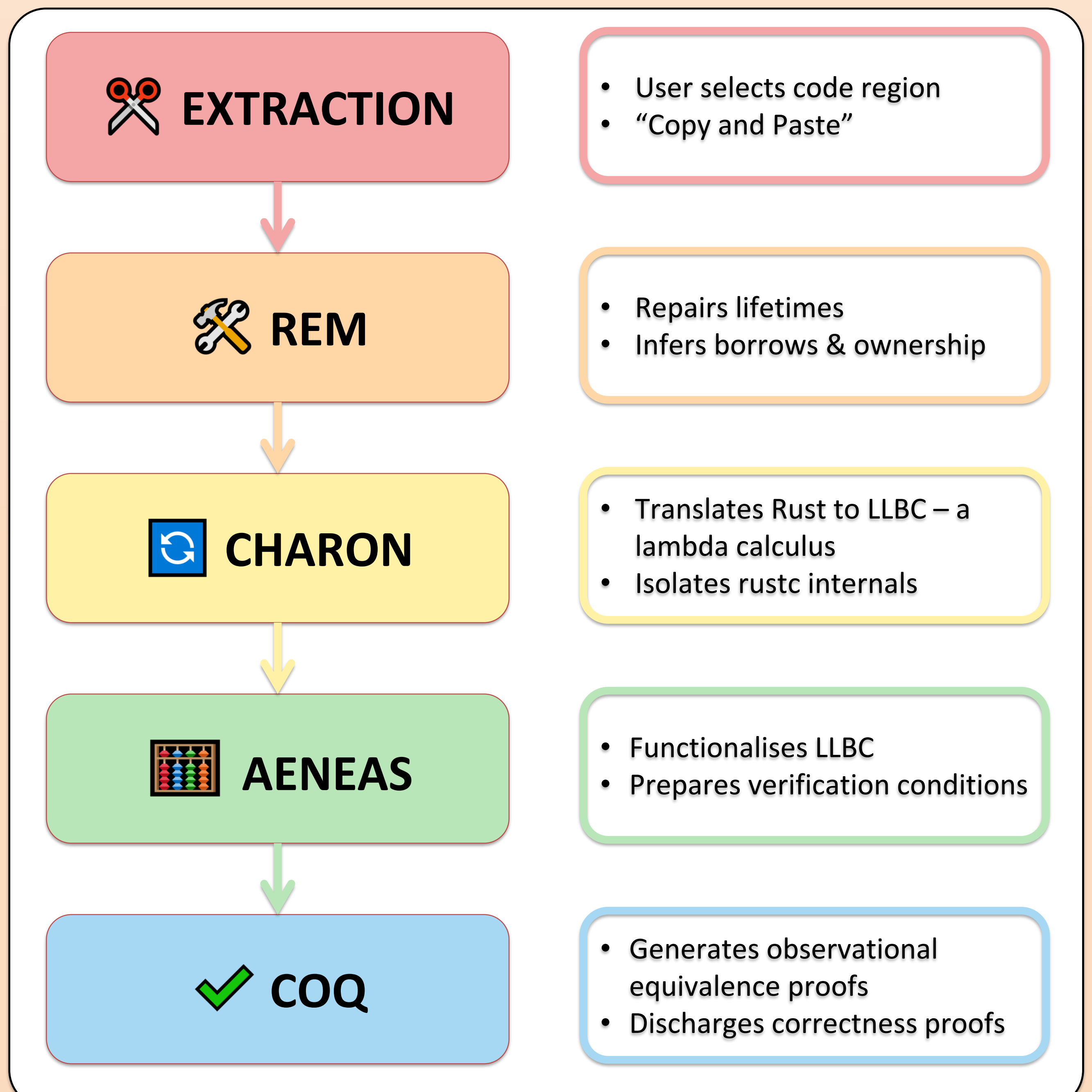
Matthew Britton, Alex Potanin, Sasha Pak

Singapore
SPLASH 2025

Motivation & Problem

- **Refactoring is Essential.** Developers spend much of their time improving existing codebases, and refactoring helps reduce technical debt, improve readability, and simplify testing.
- **Extract Method matters.** Splitting large methods into smaller, named functions is one of the most common and effective refactorings. In garbage collected languages (Java, Python), this is trivial.
- **Rust is Different.** Ownership, borrowing and lifetimes – Rust's safety guarantees – make naïve extraction nearly impossible.
  - Moving values breaks ownership
  - Borrows (`&T`, `&mut T`) can easily become invalid
  - Lifetimes may need explicit annotations the compiler won't infer automatically
  - Non-local control flow (`return`, `break`) doesn't trivially transfer to a new function
- **The gap.** Existing IDE tools (IntelliJ's Rust Plugin, Rust Analyzer) handle only simple extractions. They often fail on asynchronous code, generics, macros or complex lifetimes. Developers are often left with uncompilable or subtly incorrect code.
- **Why this matters.** In high-assurance domains, a refactoring that might silently change semantics is unacceptable. Even in Rust, compilation success ≠ semantic equivalence.

Approach & Pipeline



EXTRACTION
- User selects code region
- "Copy and Paste"

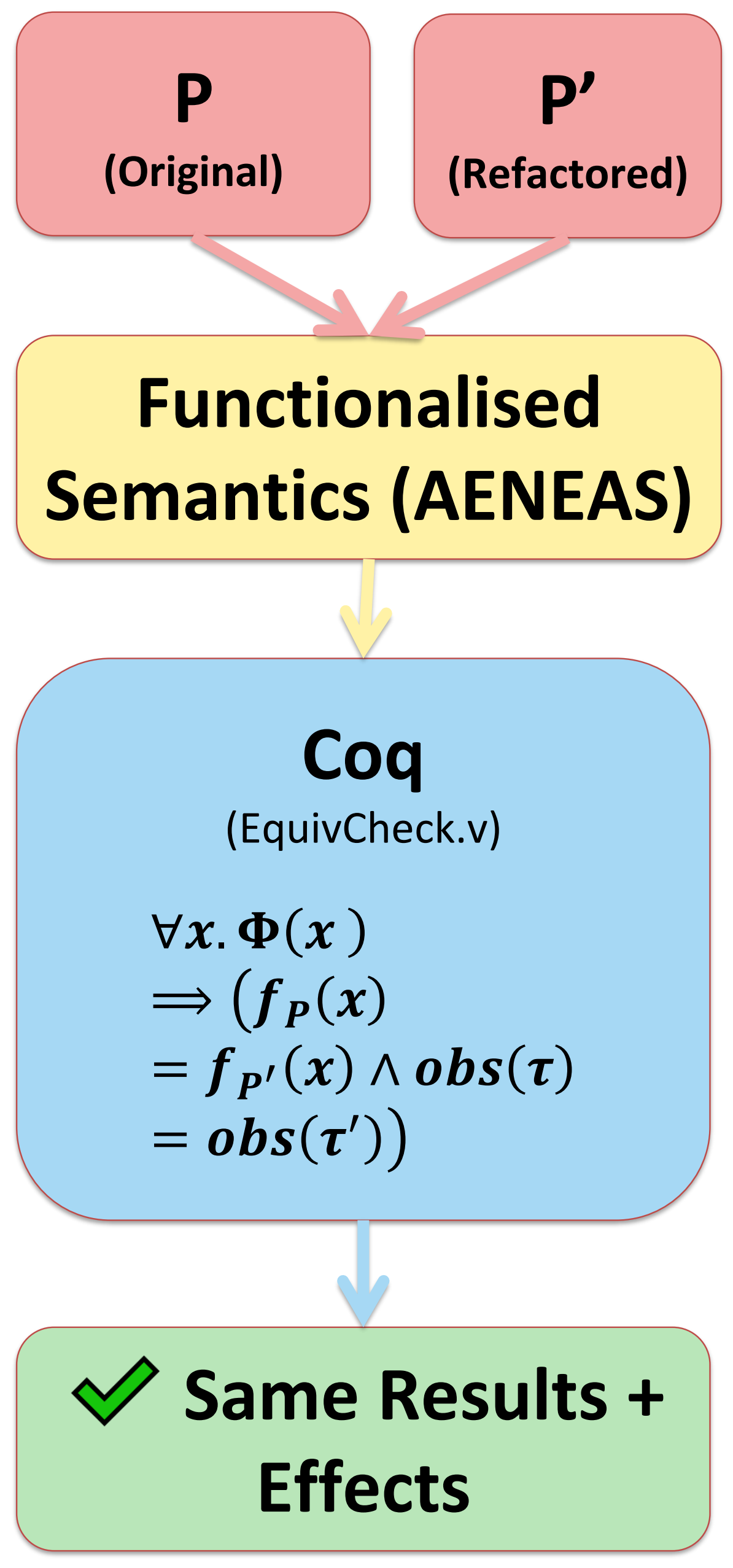
REM
- Repairs lifetimes
- Infers borrows & ownership

CHARON
- Translates Rust to LLBC – a lambda calculus
- Isolates rustc internals

AENEAS
- Functionalises LLBC
- Prepares verification conditions

COQ
- Generates observational equivalence proofs
- Discharges correctness proofs

Proof Mechanics & Obligations



$$\forall x.\, \Phi(x) \Longrightarrow \big(f_P(x) = f_{P'}(x) \wedge obs(\tau) = obs(\tau')\big)$$

- **Equivalence Obligation:** For all inputs satisfying preconditions, extracted code and original return the same result and effects
- **Proof Generation:** REMV auto-writes `EquivCheck.v` Coq file (imports + lemma skeleton)
- **Automation:** Coq discharges proof via `reflexivity` after unfolding both sides
- **Guarantee:** Trusted code without the need for user annotations

Preliminary Results

We evaluated REM on a curated set of extraction sites adapted from the `rust-analyzer` test suite, selected to span diverse language features (loops, control flow, comments, etc.). Each site was automatically transformed and verified for observational equivalence. All 10/10 cases discharged successfully in Coq, with average verification time ≈ 2 s—fast enough for interactive IDE use. These results show REMV's ability to scale beyond toy examples, and highlight opportunities for larger-scale evaluation on real-world crates.

ID	Focus	Extracted Signature	LOC(P→P')	Equiv
0	break loop	n: i32 → Option<i32>	11 → 18	✓
1	Break with value	() → Option<i32>	13 → 19	✓
2	Comments in block	() → i32	9 → 10	✓
3	Extract from nested loop	() -> i32	9 → 12	✓
4	Extract from trait impl	&self -> i32	11 → 16	✓
5	Extract mutable reference	y: &mut Foo	14 → 17	✓
6	Extract return statement	() → u32	5 → 8	✓
7	Extract mutable method call	mut n: i32	12 → 15	✓
8	No arguments if let else	() → i32	5 → 8	✓
9	Try option with return	() → Option<i32>	12 → 16	✓

From Prototype to Production

- **Standalone CLI:** working tool outside of the research harness
- **Language Coverage:** already supports async/await, generics, macros, etc.
- **IDE integration:** VSCode extension provides proof – of – concept with live extract → fix → verify cycle.
- **Performance:** ~2s proof cycles → feels interactive
- **Robustness:** automated repair loop makes it work on real crates

"Not just theory – A usable developer tool"

Future Work

- **Scaling:** whole-crate coverage via caching + incremental re-verification.
- **New features:** unsafe code, concurrency, richer ownership patterns.
- **Diagnostics:** friendlier failure messages when proofs don't go through
- **Large-scale evaluation:** across multiple community crates
- **Stronger guarantees:** beyond observational equivalence!

"Towards real-world scale"

GitHub

VSCode

DOI