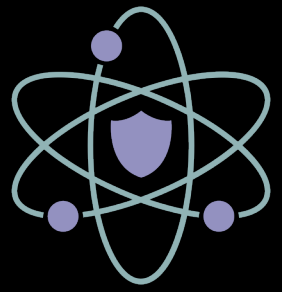
EUROPEAN CONFERENCE ON PQC MIGRATION

The Post-Quantum Cryptography Alliance (PQCA)

Driving development and adoption of open-source post-quantum cryptography

Speaker: Hanno Becker

Principal Applied Scientist, AWS



Post-Quantum Cryptography Alliance

Mission:

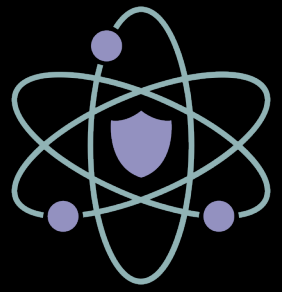
"To address cryptographic security challenges posed by quantum computing by …

- producing high-assurance software implementations of standardized algorithms, and
- supporting the continued development and standardization of new post-quantum algorithms with software for evaluation and prototyping."

https://pqca.org

https://github.com/PQCA

PQCA Overview



Post-Quantum Cryptography Alliance

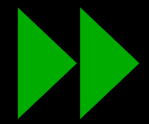
- A Linux Foundation project
- Participation open to all – join us!
- Organizations can join as members to influence budget and direction
- 24 members so far, including IBM, Google, Amazon, Cisco, NVIDIA, plus general and associate members
- All development done under permissive open source licenses (mostly MIT, Apache 2)

THE LINUX FOUNDATION

Interested to say hello and learn more?
Join our public meetings:
https://pqca.org/calendar/

PQCA Projects

Prototype → Plan → Migrate



Open Quantum Safe

Prototype implementation and integrations of PQC

https://openquantumsafe.org/

CBOMkit

Provide tools for creating and analyzing Cryptographic Bills of Materials (CBOMs)

https://github.com/cbomkit/

Post-Quantum Code Package

Production-ready implementations of standards-track PQC

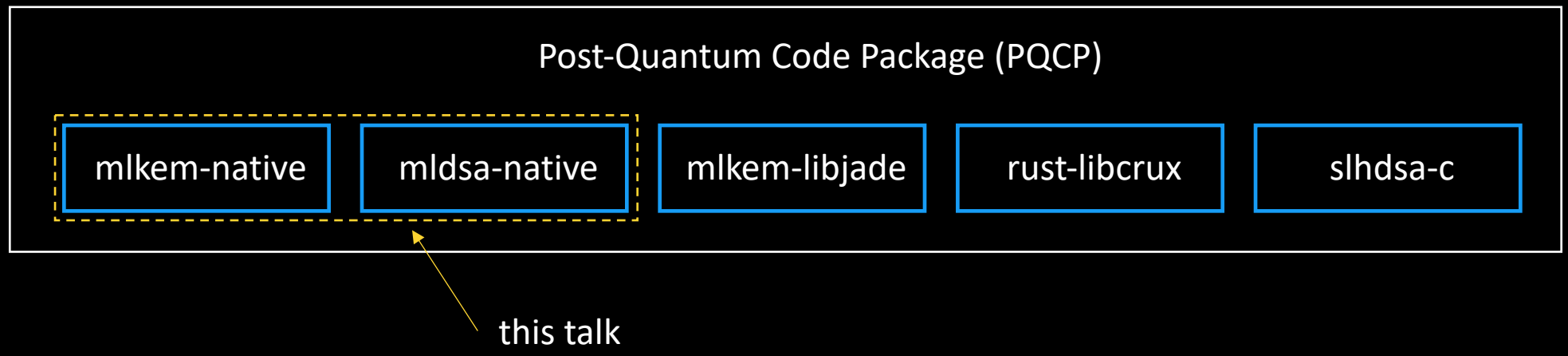
https://github.com/pq-code-package/

Post-Quantum Code Package (PQCP)

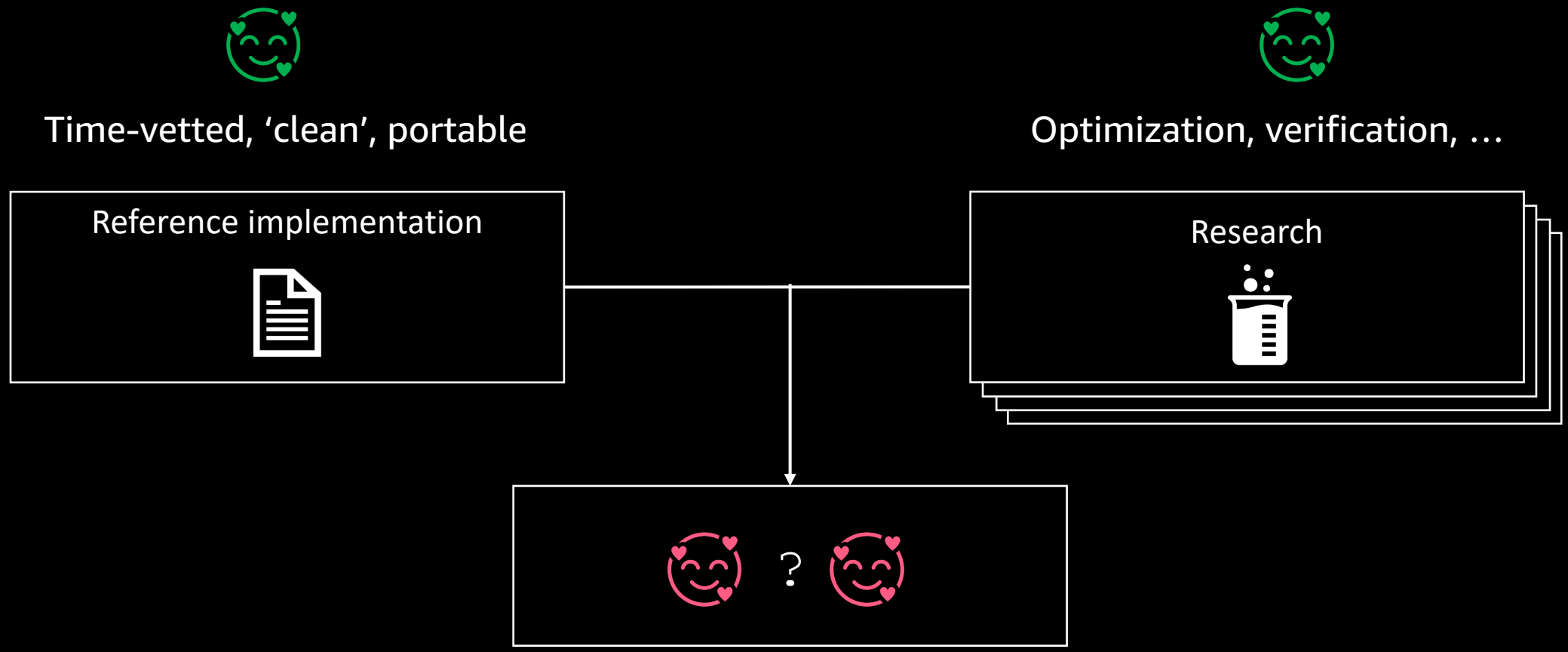
Collection of open source projects aiming to build production-ready,

high-assurance implementations of standards-track post-quantum cryptography

https://github.com/pq-code-package/

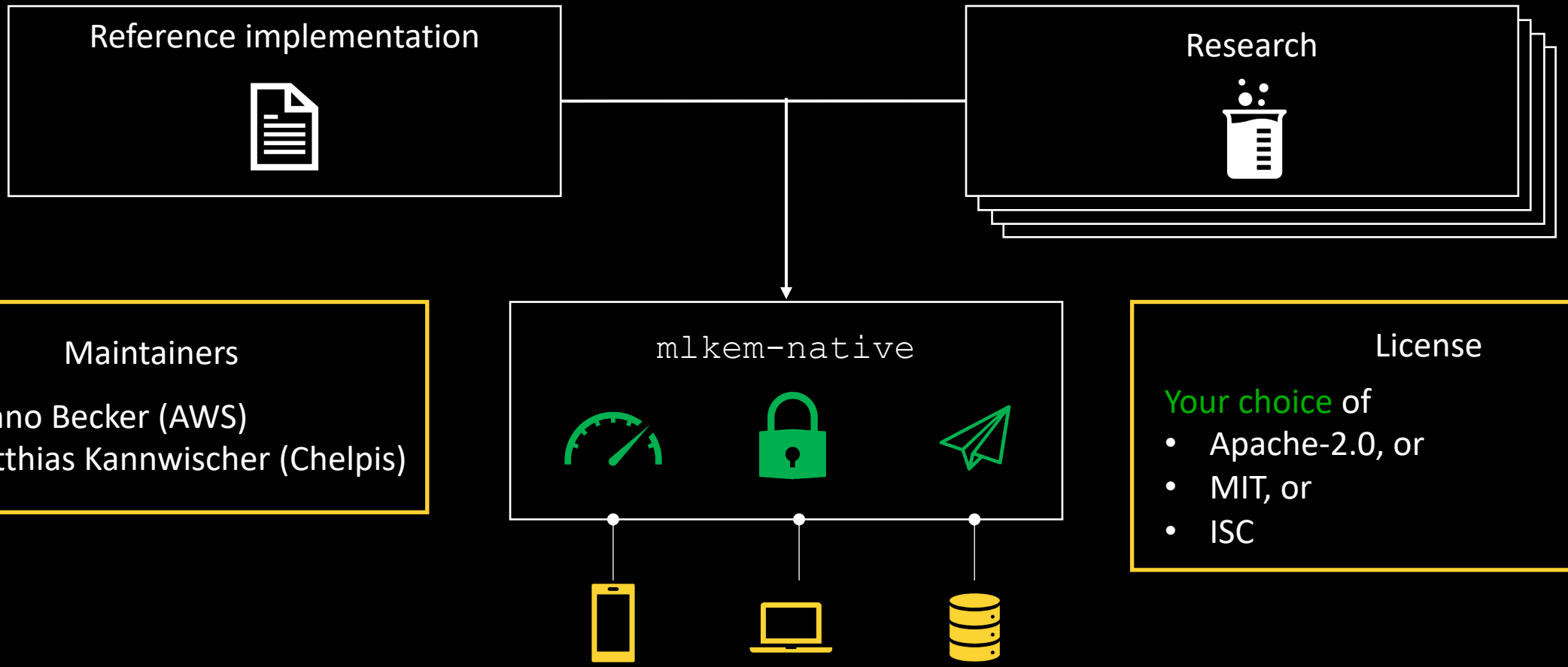


Reference vs. Research



Meet: mlkem-native

https://github.com/pq-code-package/mlkem-native



“Production ready” Cryptographic Engineering

Protects customer’s privacy

Maximizes portability & maintainability

Minimizes cryptographic tax

Why use mlkem-native?

Portability & Maintainability

- C90; minimal dependency; tested against many 16/32/64-bit compilers
- Uniform frontend/backend architecture
- Documentation of relation to FIPS 203 and reference implementation

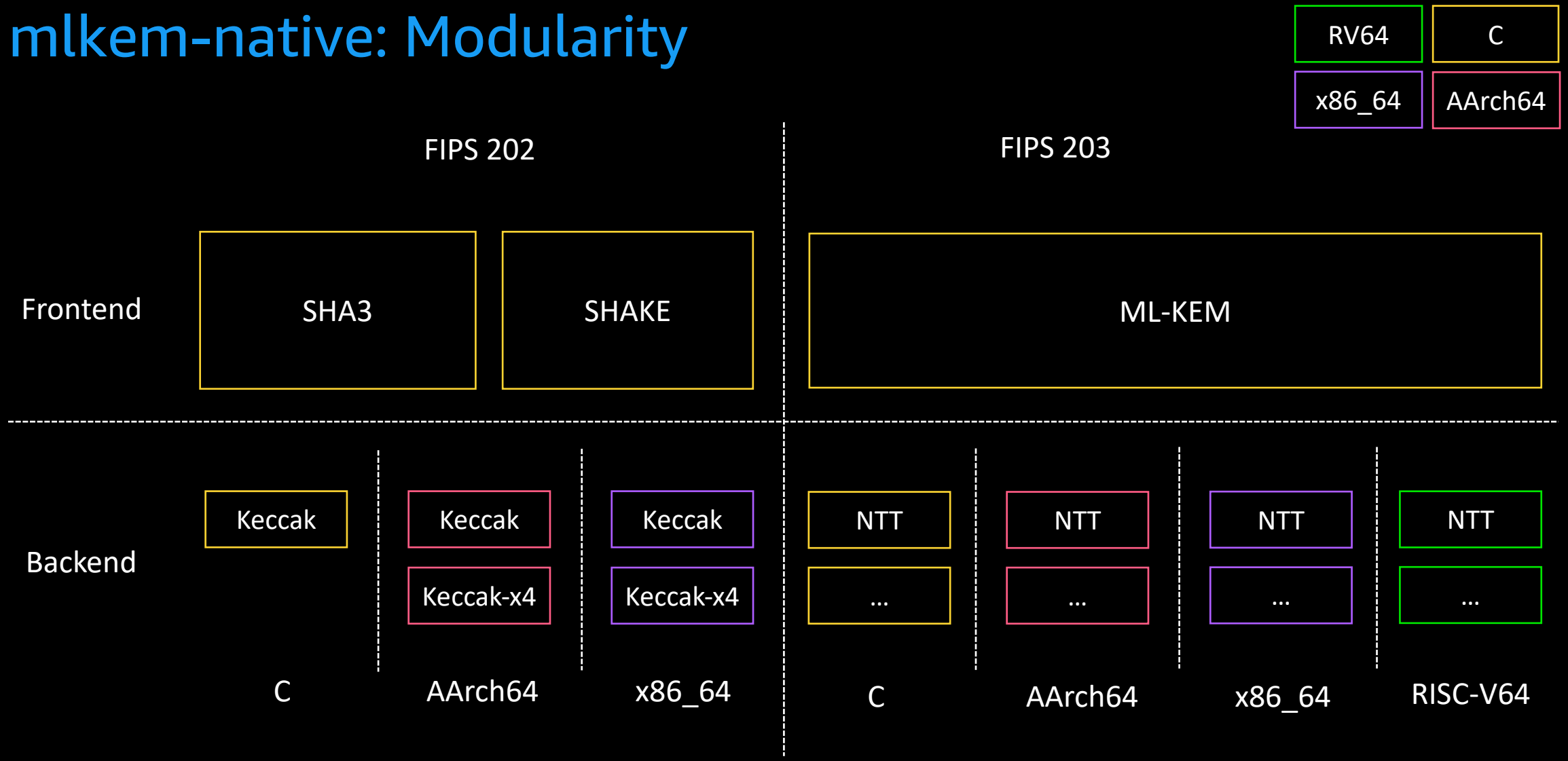
Assurance

- Memory-safe and type-safe: Statically verified using model checking (CBMC)
- Functional correctness of assembly verified at binary level (HOL-Light)
- Extensive dynamic constant-time testing (valgrind)

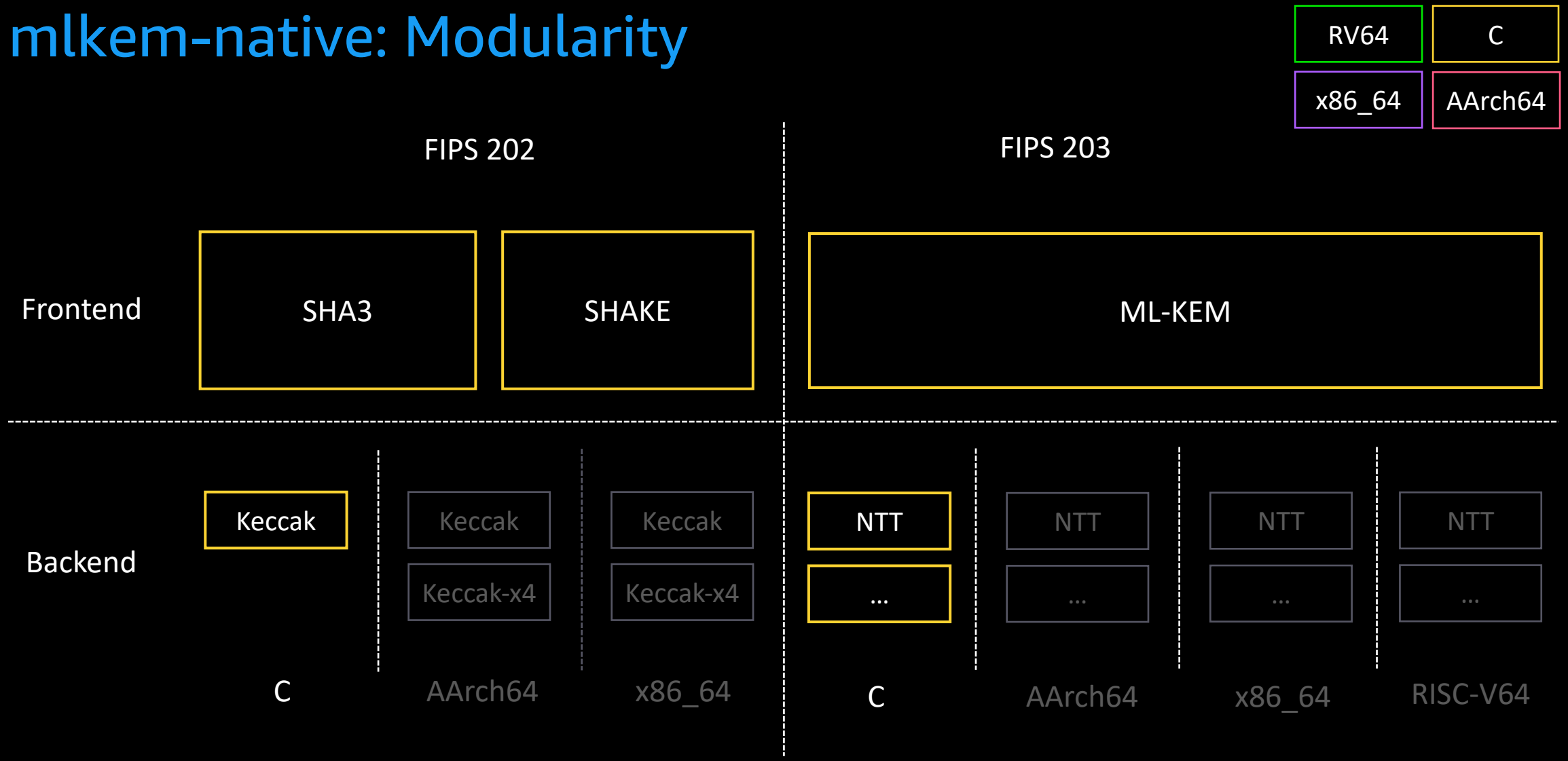
Performance

- Assembly/intrinsics backends for AArch64, x86_64, RISCV-64

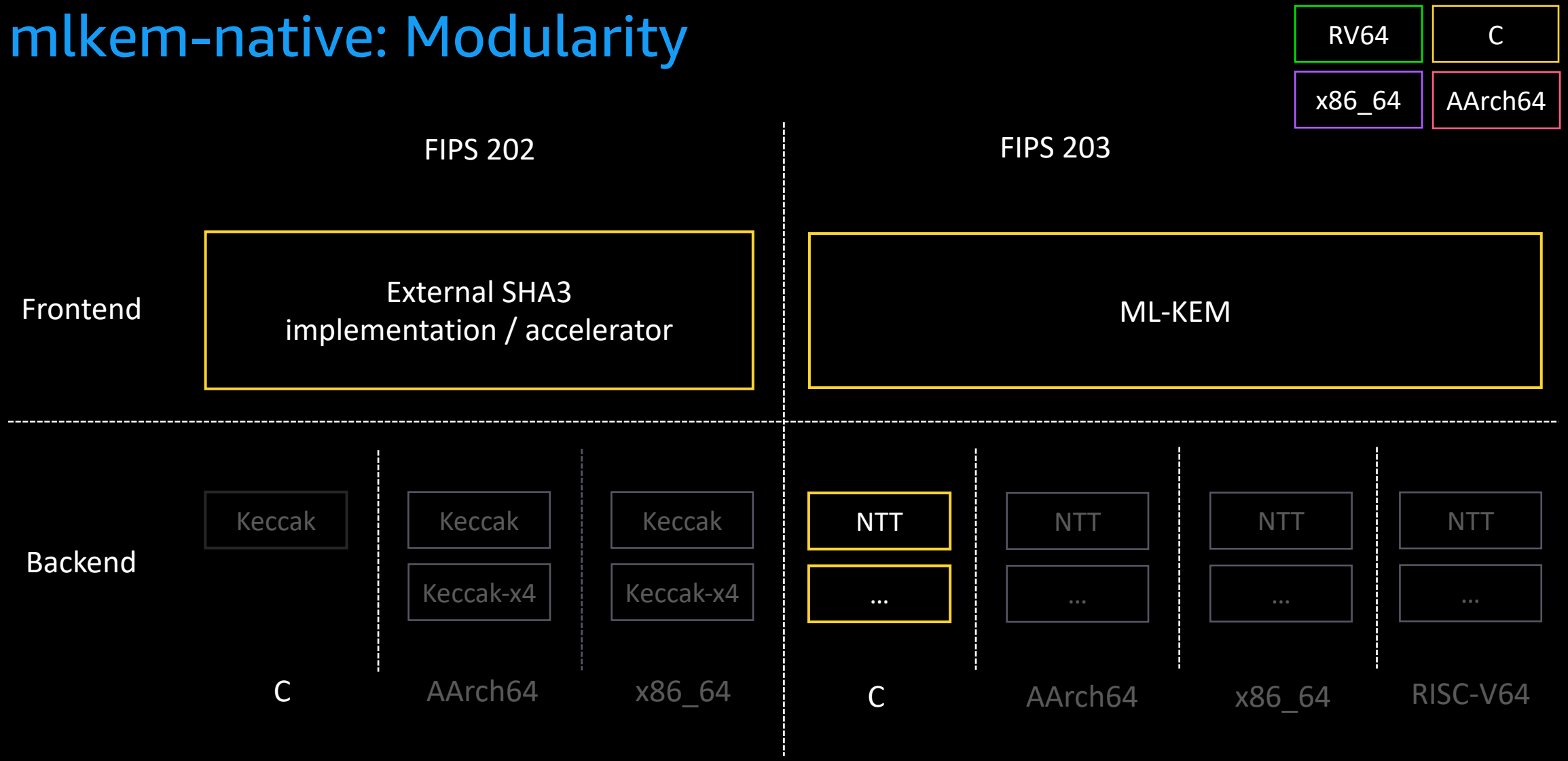
mlkem-native: Modularity



mlkem-native: Modularity



mlkem-native: Modularity

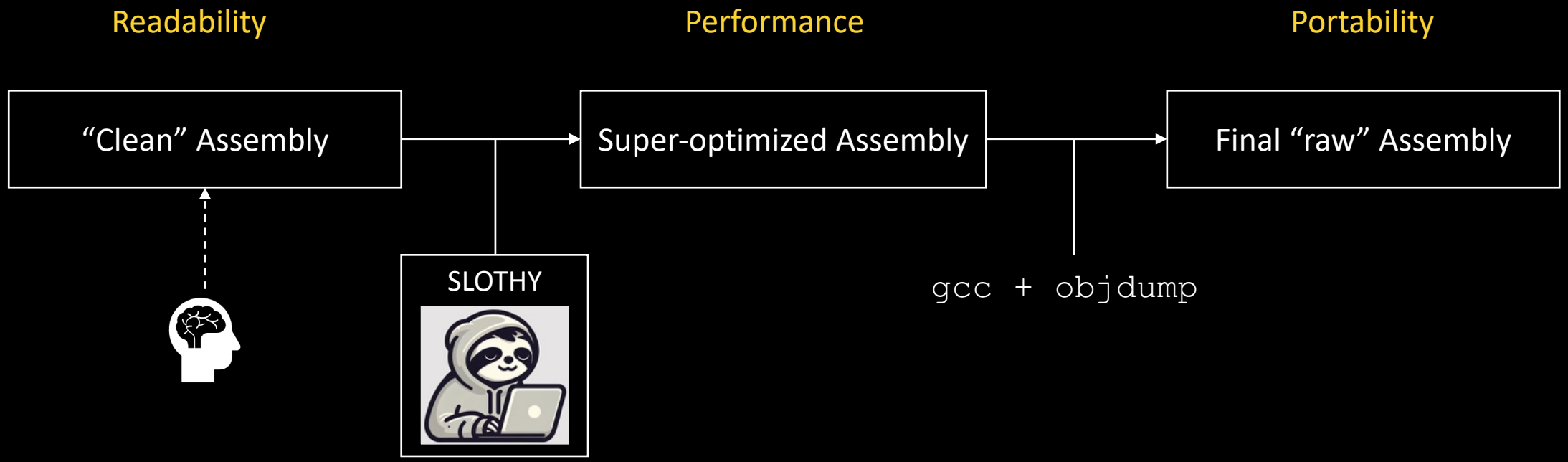


mlkem-native: Assembly

Handwritten assembly is great! 

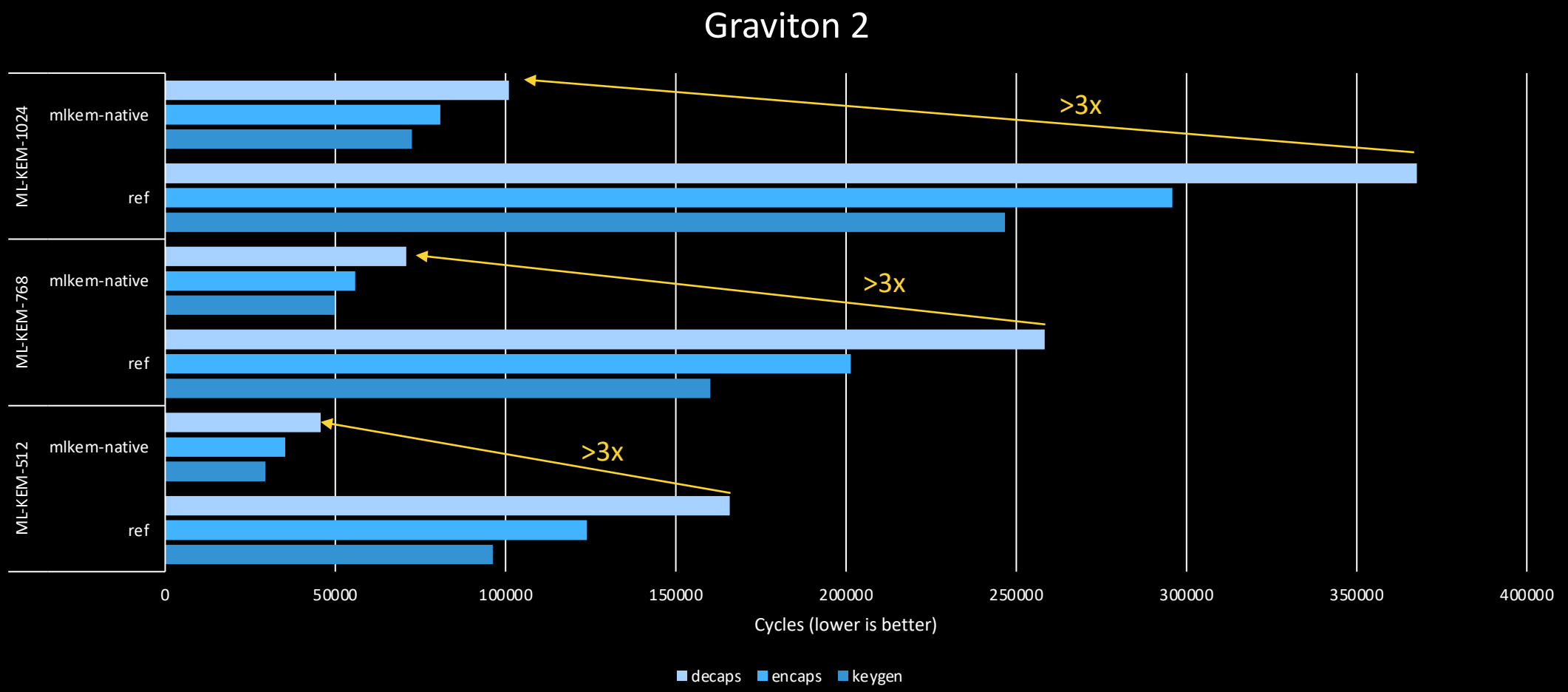
… until you need to maintain it 

mlkem-native: Assembly lifecycle



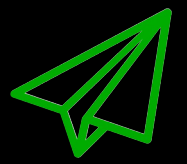
https://github.com/slothy-optimizer/slothy/

mlkem-native: Performance before/after optimization



“Production ready” Cryptographic Engineering

Maximizes portability & maintainability



- C90
- Automate low-level performance optimization from “clean” assembly
- Uniform frontend/backend architecture

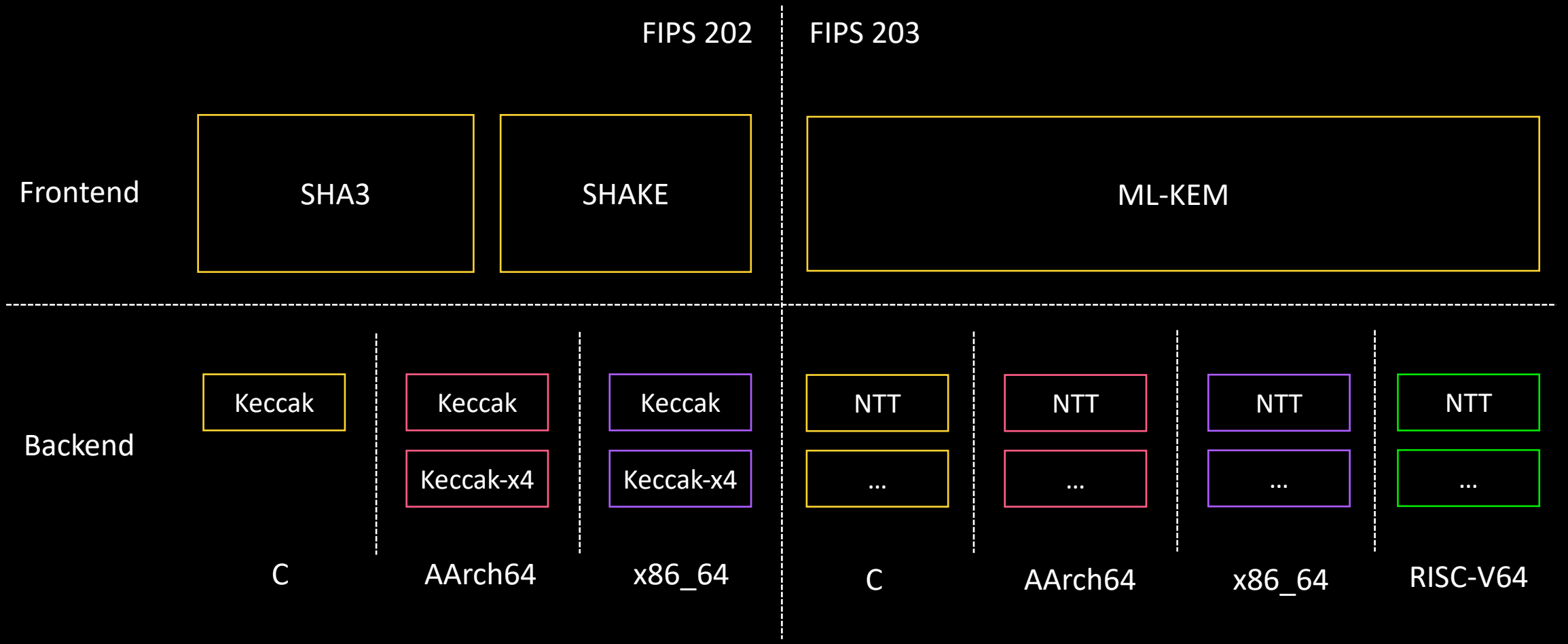
Minimizes cryptographic tax

Optimization without verification is difficult to trust and may not be adopted

Protects customer’s privacy

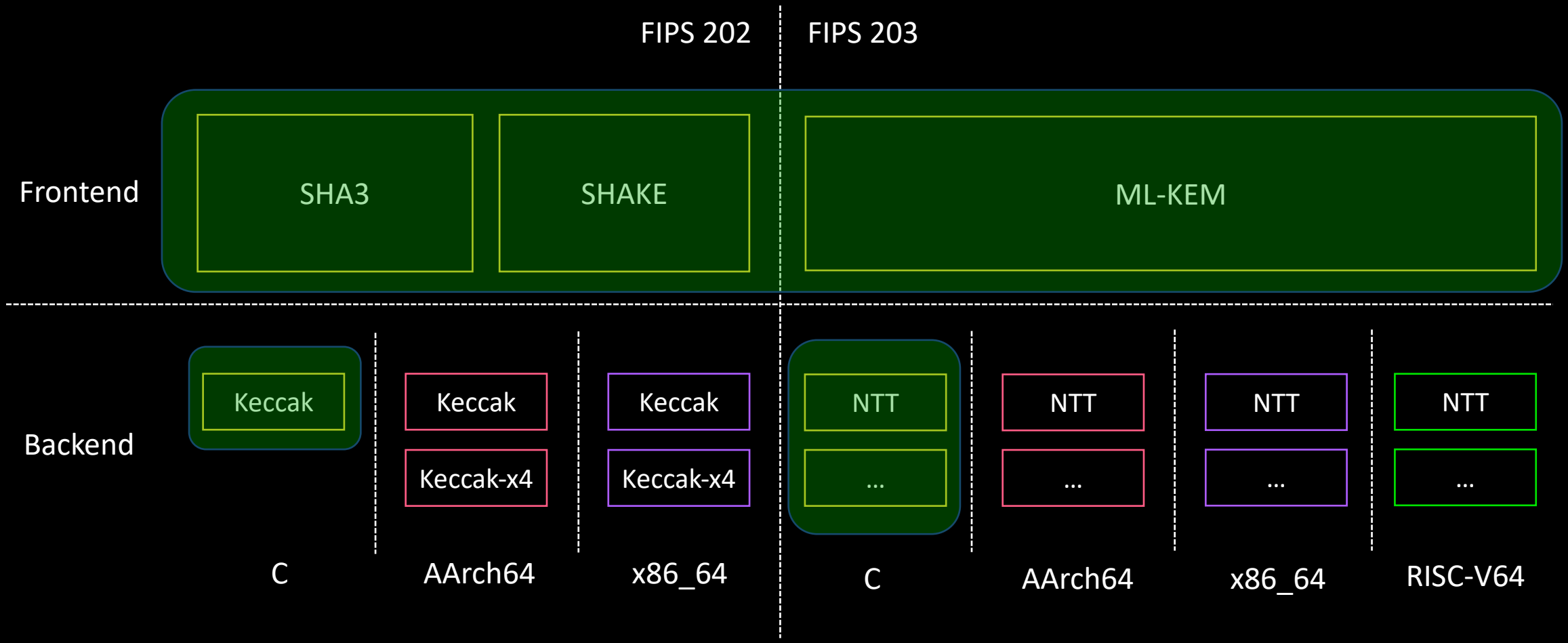
It’s (too) hard to write safe and secure C. Why trust your’s?

mlkem-native: Verification



mlkem-native: Verification

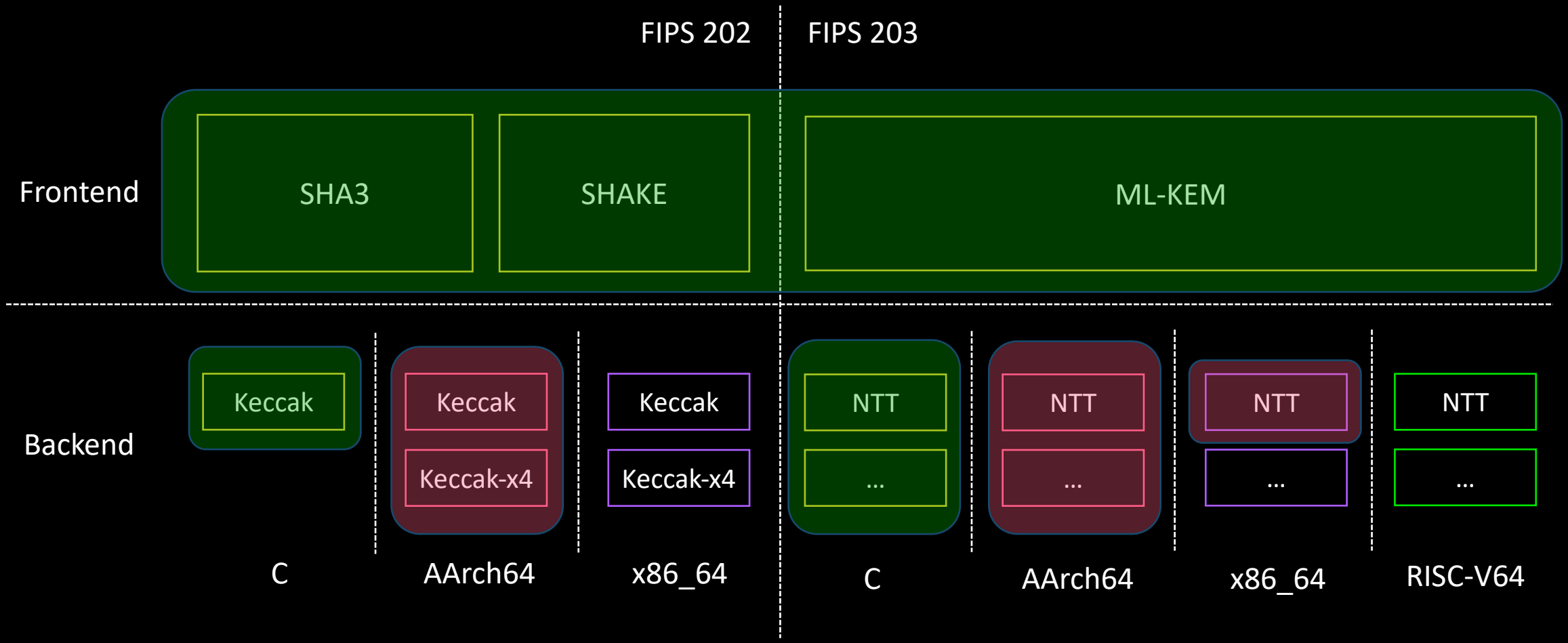
Verified: Memory-safe, type-safe



mlkem-native: Verification

Verified: Memory-safe, type-safe

Verified: Functionally correct



mlkem-native: C Bounded Model Checker (CBMC)

What:

Memory-safety, Type-safety

How:

In-source contracts, invariants

Coverage:

All C code in C/AArch64 backend

When:

Runs in CI on every change

mlkem-native: HOL-Light & s2n-bignum

What:

Functional correctness of object code

How:

Manual proof, HOL-Light + s2n-bignum

Coverage:

- All AArch64 assembly
- Some x86 assembly (in progress)

When:

Runs in CI on every change. ~1.5h

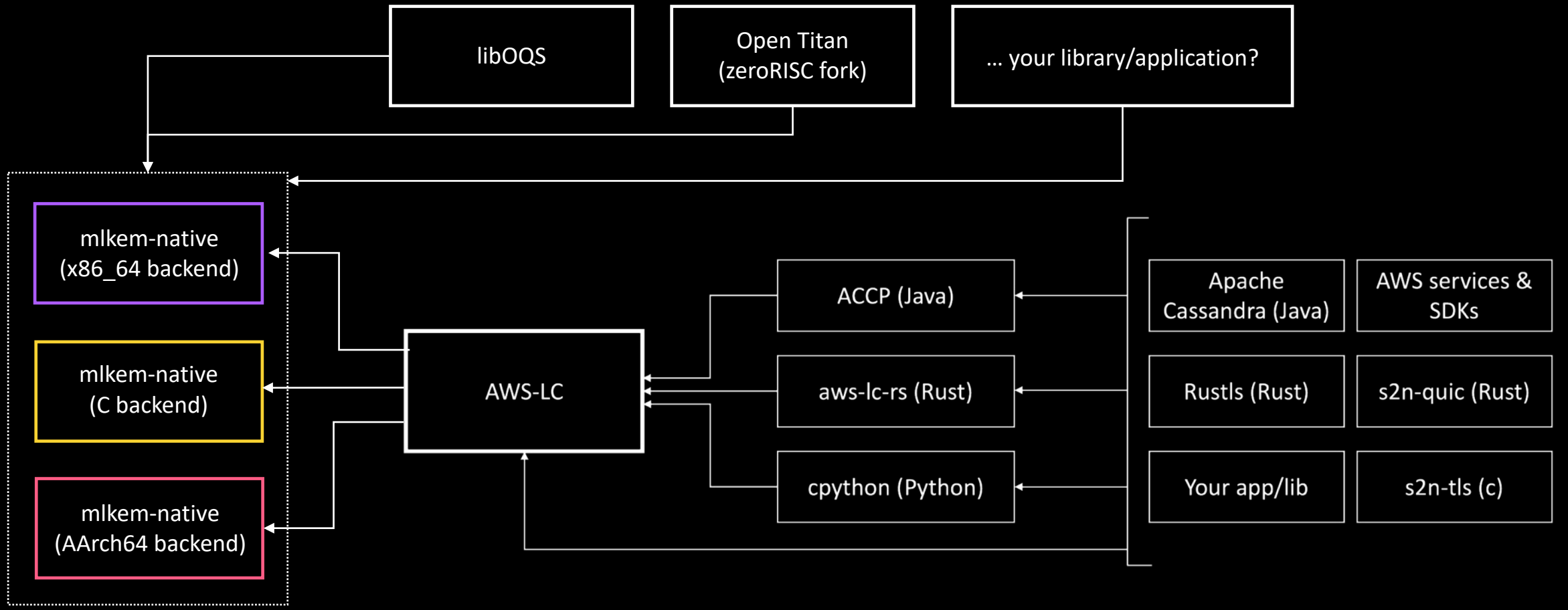
Side-channel attacks

All code in mlkem-native is written with constant-time in mind:

- No secret-dependent branches or memory accesses
- No variable-time instructions
- Value-barriers preventing harmful compiler optimizations

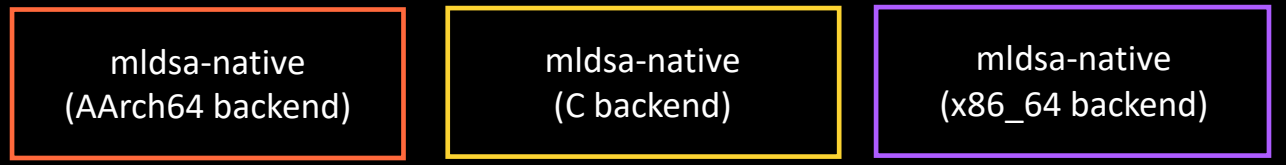
Use `valgrind` to test for violations (based on code annotations)

Applications



mldsa-native

Doing it all over



mldsa-native v1-alpha released two weeks ago!

https://github.com/pq-code-package/mldsa-native

Why (not) use mlkem/mldsa-native?

Portability & Maintainability

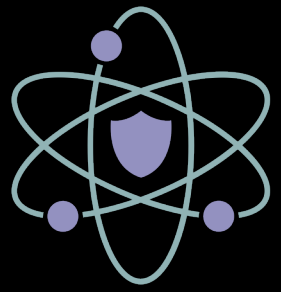
Assurance

Performance

If you need ML-KEM/ML-DSA, consider using mlkem/mldsa-native and their verified s2n-bignum assembly kernels.

Like mlkem/mldsa-native, but they don’t quite fit your project yet? Please let us know
Let’s collaborate, not duplicate efforts!

Summary



Post-Quantum Cryptography Alliance

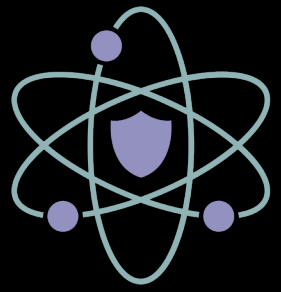
The PQCA is a Linux Foundation project that

- Develops open source software aiding the transition to PQC
- Covers all phases of the transition: Prototyping, Planning, Migration
- Is open to all!

mlkem-native and mldsa-native

- Open-source projects in the Post-Quantum Code Package under the PQCA
- Production-ready, highly portable ML-KEM/ML-DSA implementations with formal safety and correctness guarantees and low maintenance cost

Get Involved!



Post-Quantum Cryptography Alliance

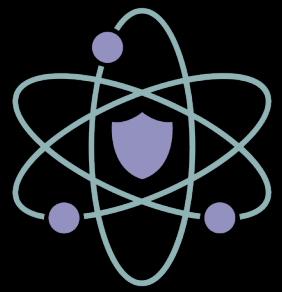
If you are interested, reach out to us or join one of our open meetings!

https://pqca.org/get-involved/

https://pqca.org/calendar/

https://github.com/pq-code-package/mlkem-native

https://github.com/pq-code-package/mldsa-native



Post-Quantum Cryptography Alliance

Thank you!

Questions?