Arkworks small field support

Using system-native types for efficient arithmetic

Benjamín Benčík, Andrew Zitek-Estrada

Charles University

EPFL

Motivation

Finite Fields are the foundation of Arkworks

- Optimizations on basic arithmetic accumulate savings at protocol level
- Ark-ff offers performance and flexibility for arbitrary sized fields

Status Quo

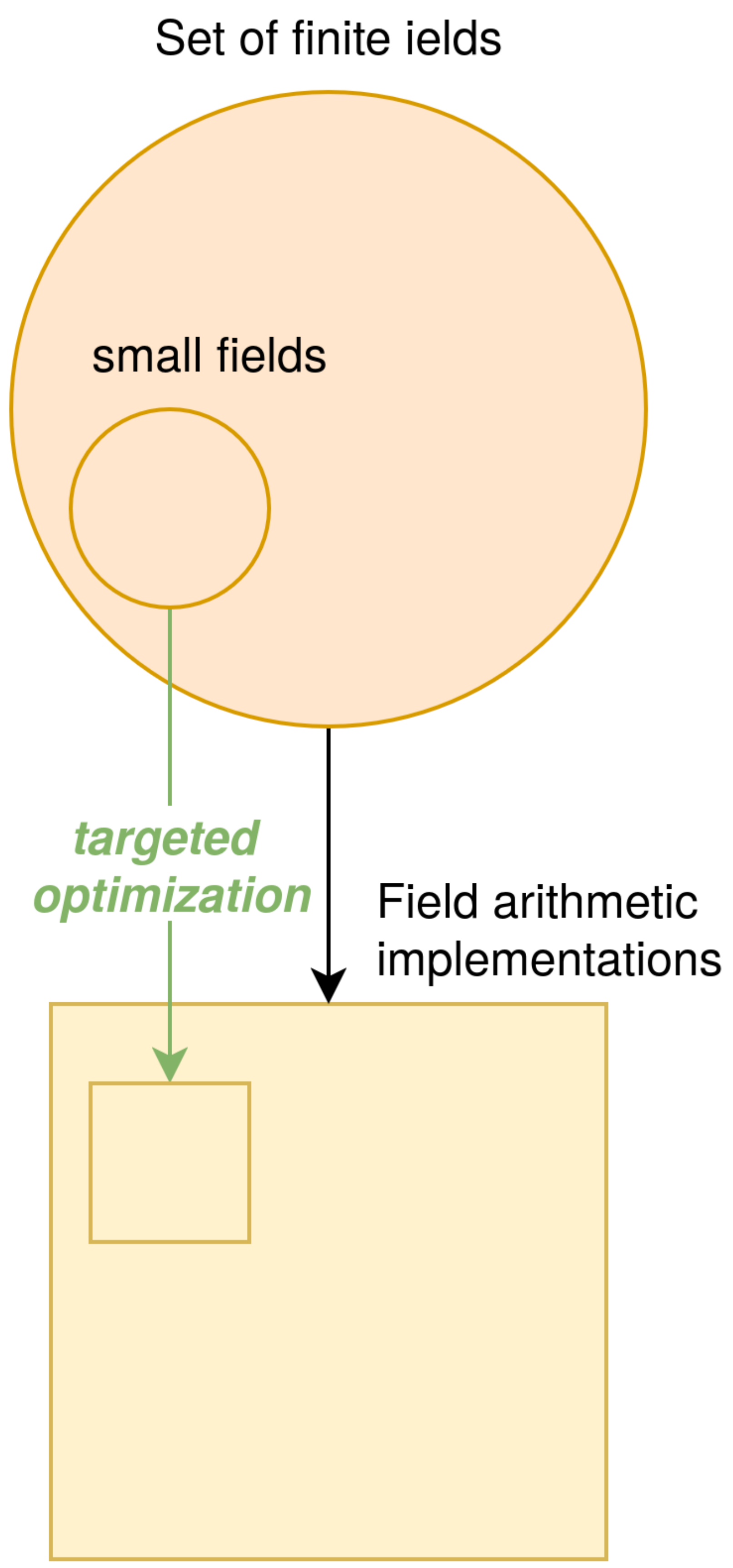
- Path toward SIMD/ vectorization and performance boost in serial

Goal

Freebie

- Rewrite arbitrary bit-length arithmetic with native types for moduli < 128 bits
- No breaking changes

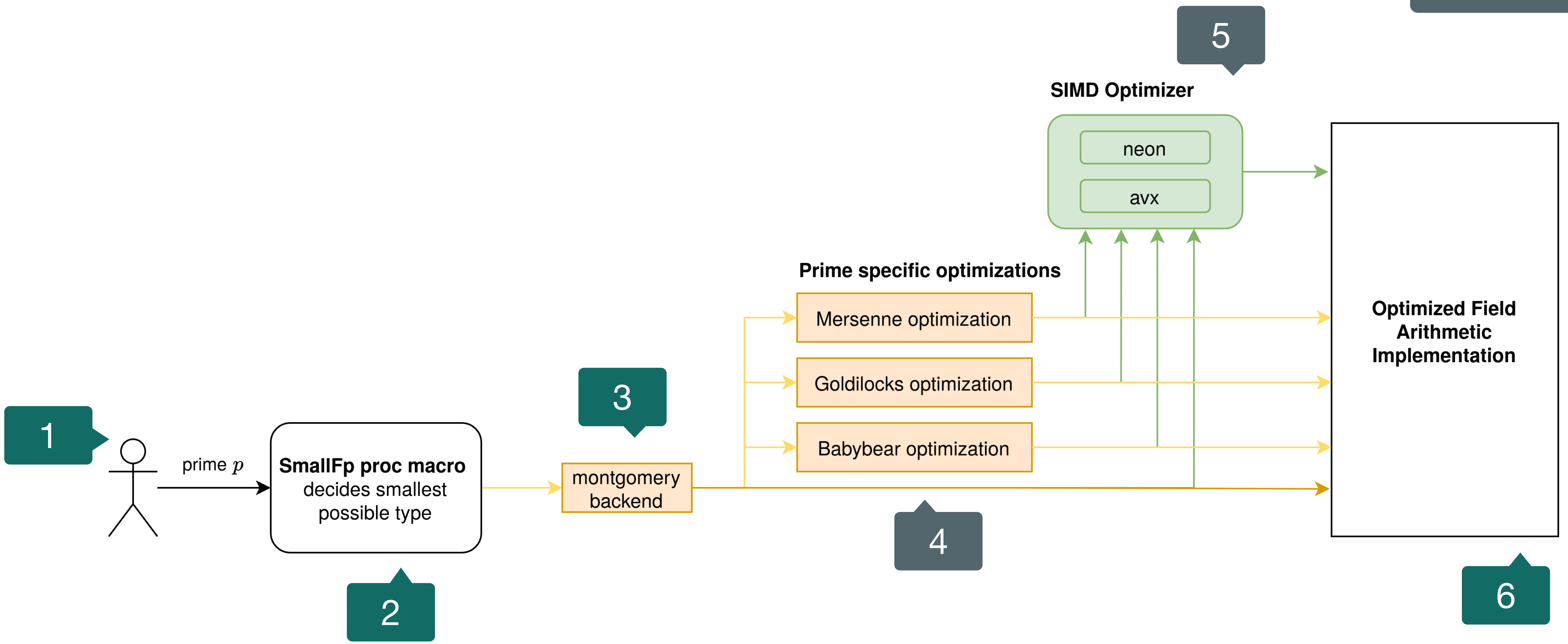
🎉



Roadmap

Overview of project scope

4 and 5 future work needed by this effort



Crates: ff-macros | ff | test-curves

Instantiation

Familiar flow for user

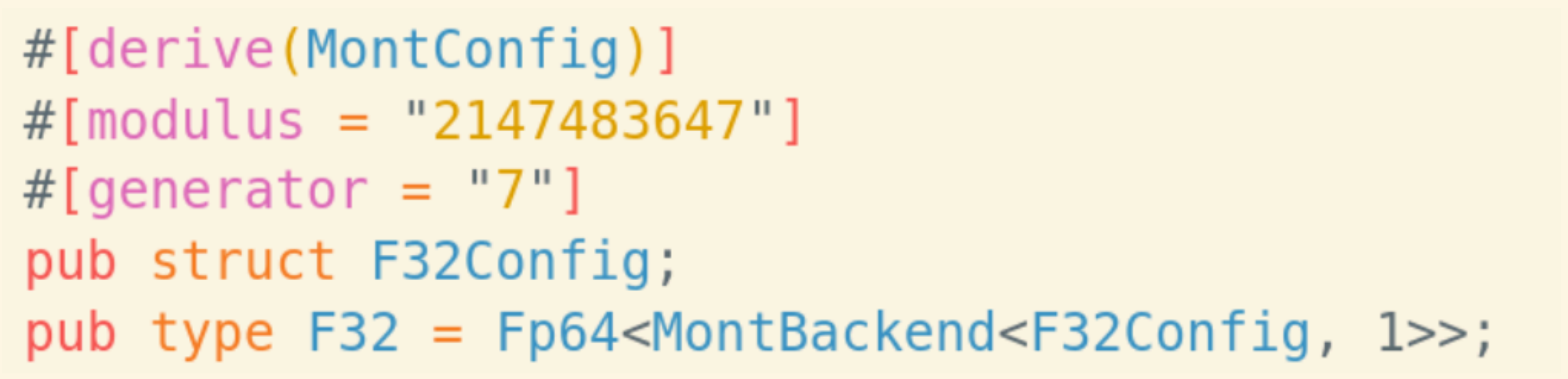
- Supply same config to new macro

- Use the generated type as usual 🎉

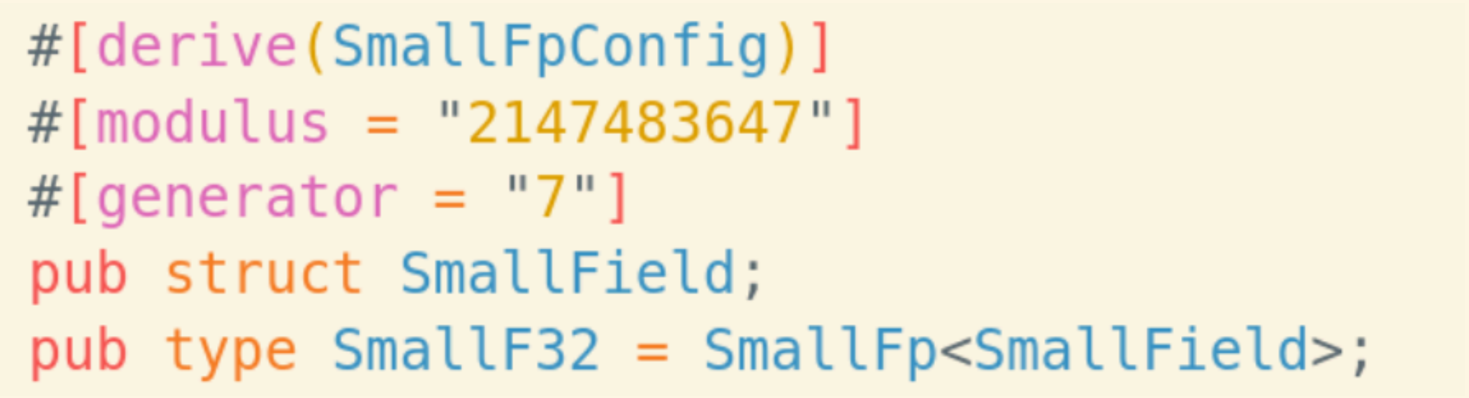
Existing macro and new macro are orthogonal

- Instead of BigInt, SmallFp macro uses u8, u16, u32, u64 or u128

Existing



```rust
#[derive(MontConfig)]
#[modulus = "2147483647"]
#[generator = "7"]
pub struct F32Config;
pub type F32 = Fp64<MontBackend<F32Config, 1>>;
```



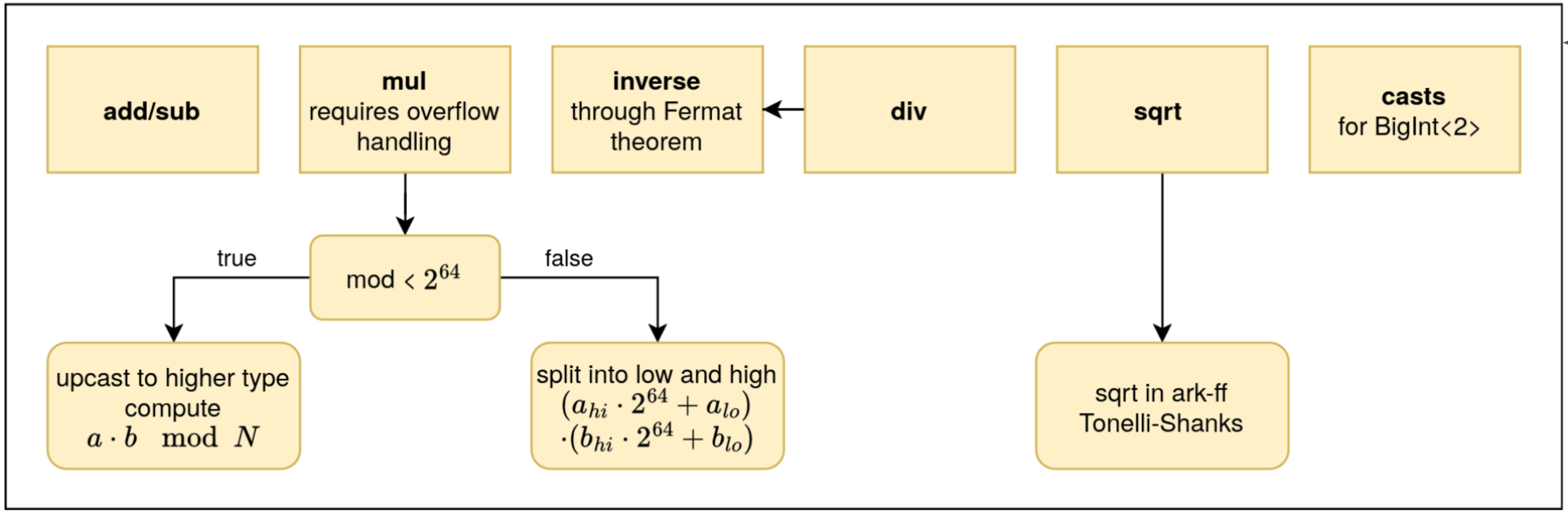
```rust
#[derive(SmallFpConfig)]
#[modulus = "2147483647"]
#[generator = "7"]
pub struct SmallField;
pub type SmallF32 = SmallFp<SmallField>;
```

New

Macro

Macro generates the backend functions

Macro implementations



add/sub

mul requires overflow handling

inverse through Fermat theorem

div

sqrt

casts for BigInt<2>

mod < 2^{64}

true

false

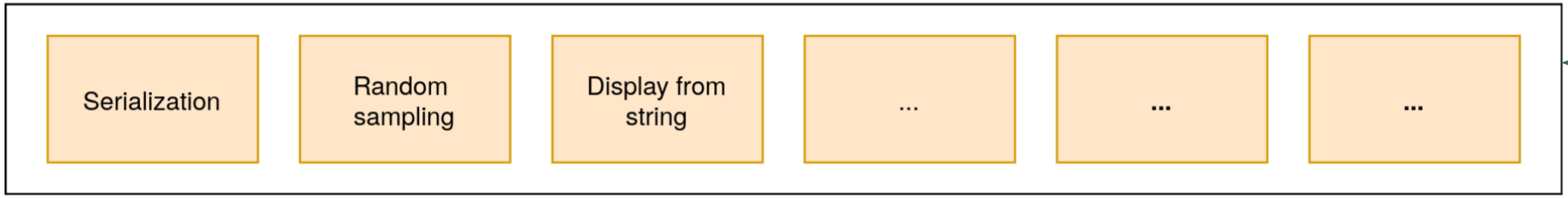
upcast to higher type compute $a \cdot b \mod N$

split into low and high $(a_{hi} \cdot 2^{64} + a_{lo}) \cdot (b_{hi} \cdot 2^{64} + b_{lo})$

sqrt in ark-ff Tonelli-Shanks

Generate by macro

Default trait implementations (backend agnostic)



Serialization

Random sampling

Display from string

...

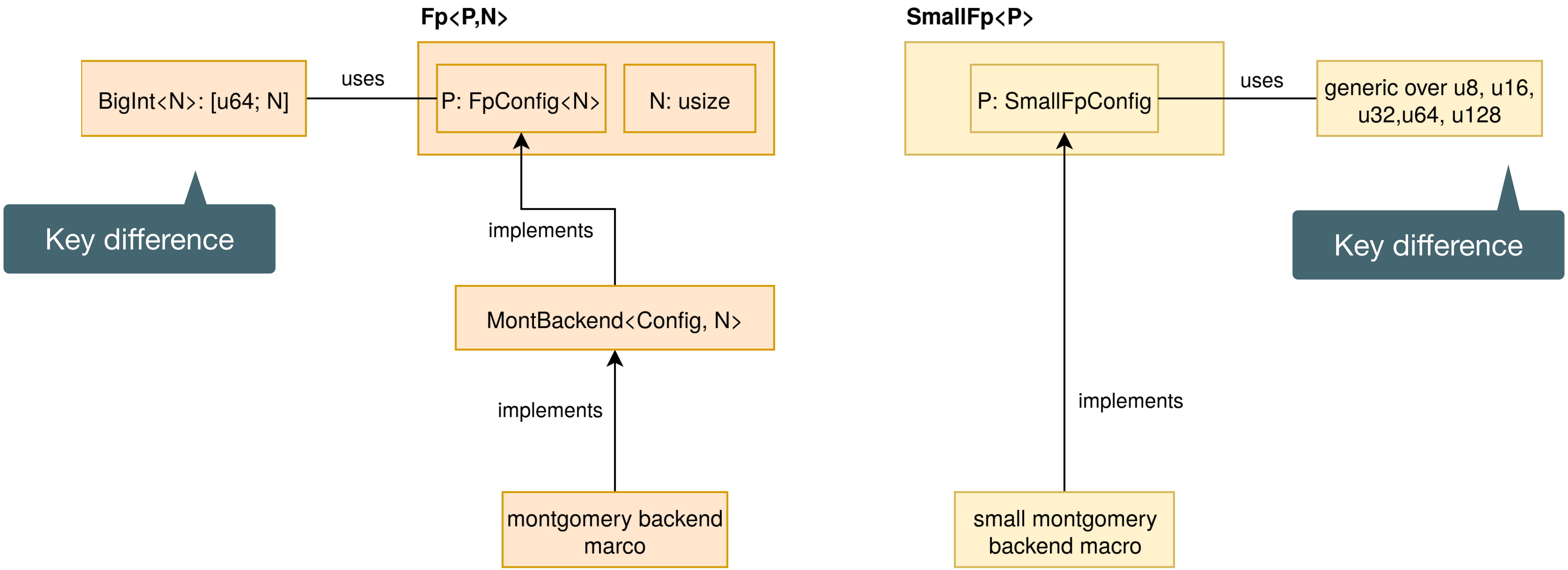
...

...

Same for all inputs

Trait

Trait implementation is filled in with the backend

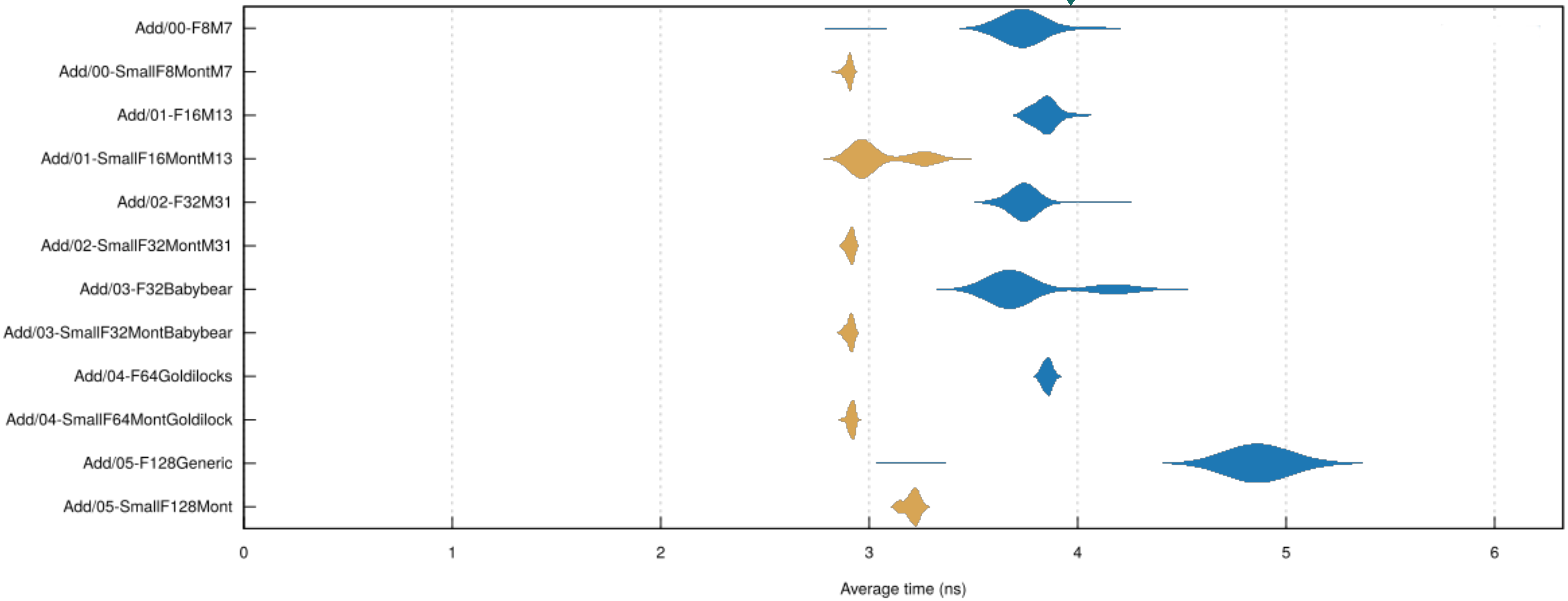


Results

Benchmarks (serial)

Addition

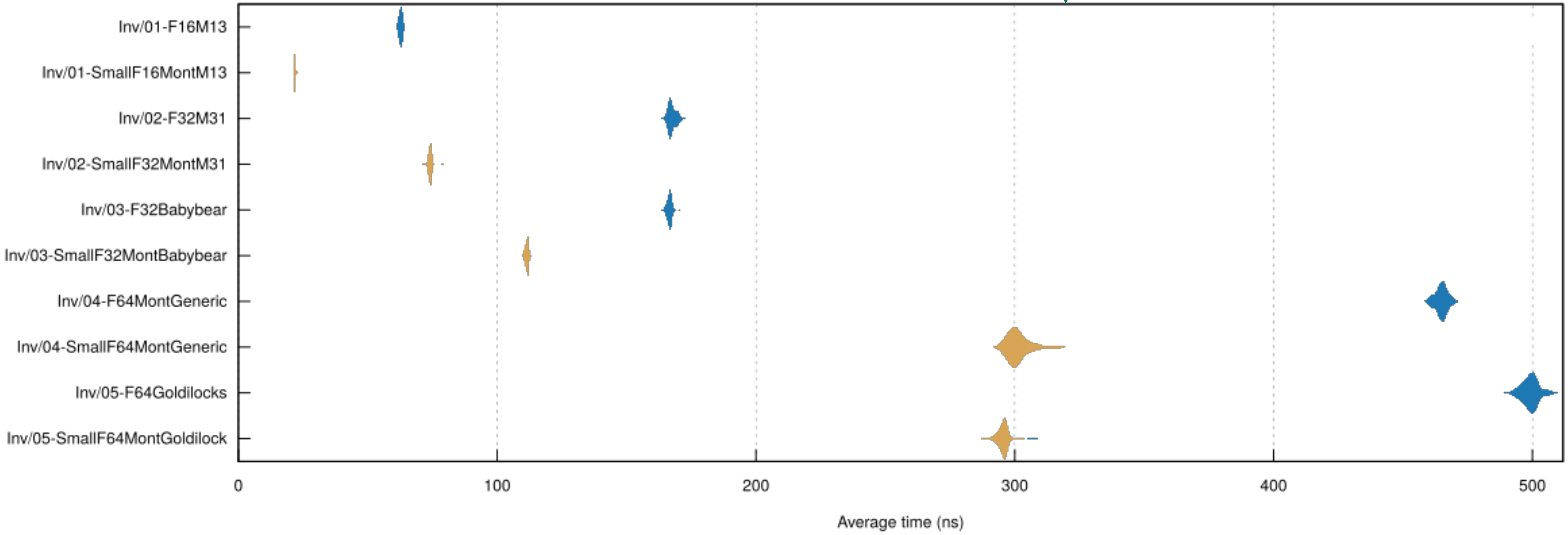
~ 20-35% improvement for all tested fields



Benchmarks (serial)

Inverse

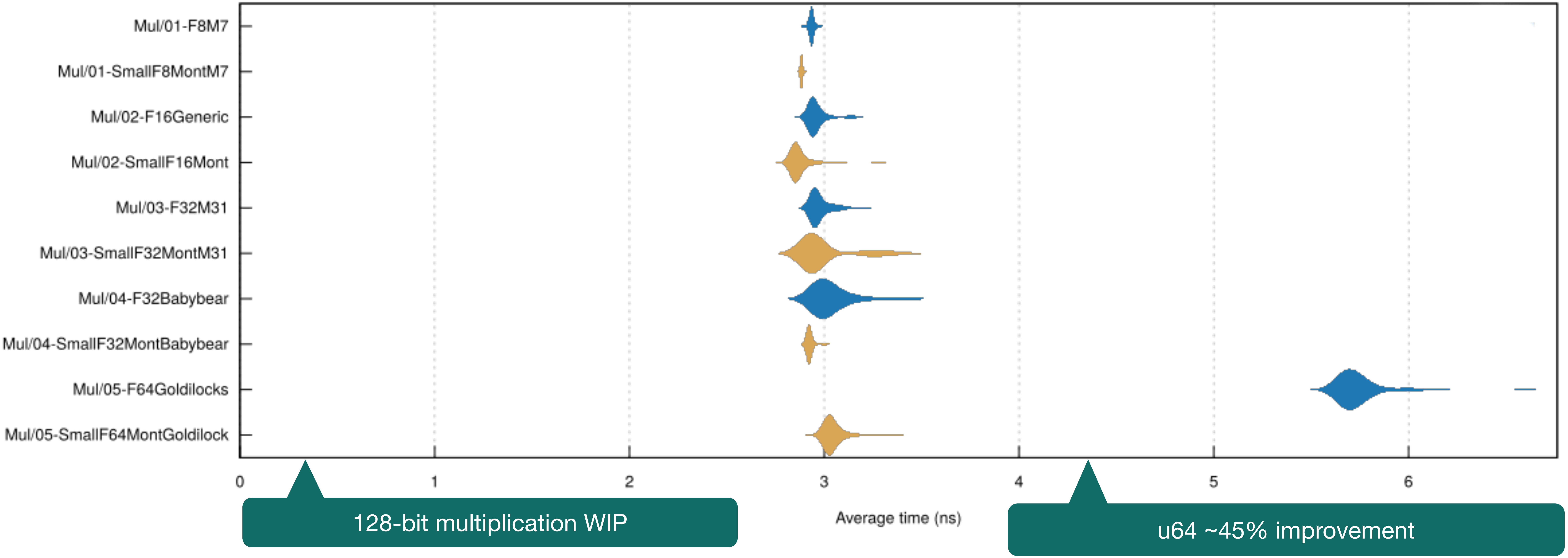
~ 35-60% improvement for all tested fields



Benchmarks (serial)

Multiplication

fields that fit into u32 faster 3-5%



128-bit multiplication WIP

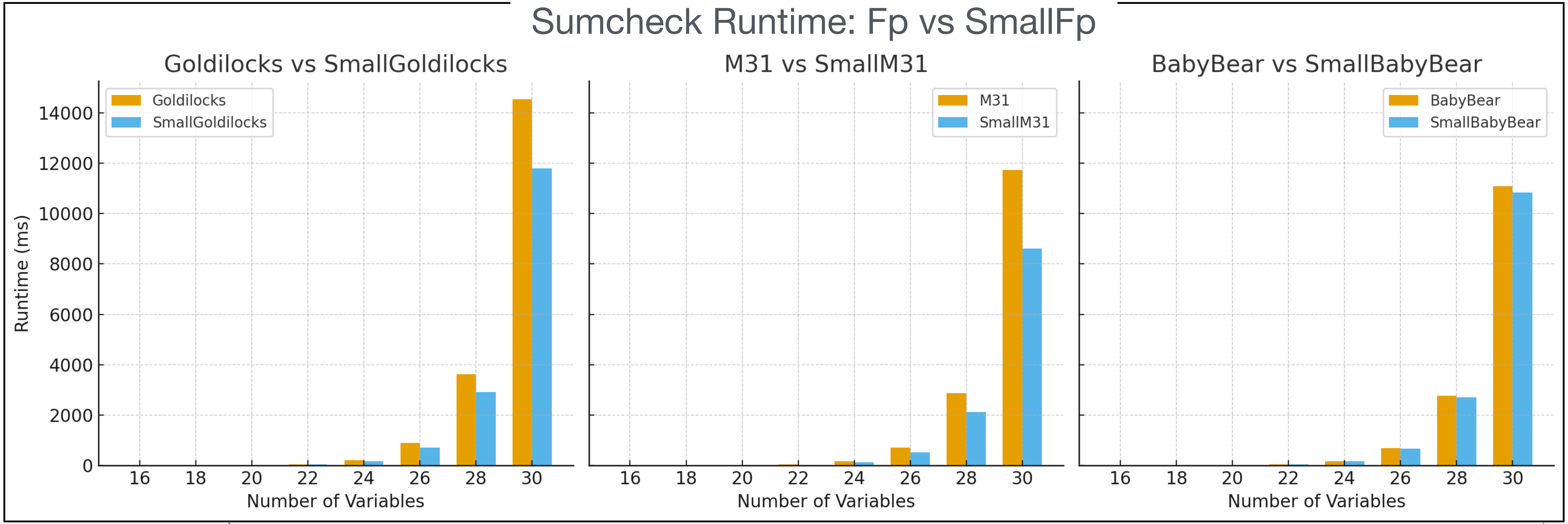
u64 ~45% improvement

Benchmarks (serial)

Efficient Sumcheck

Drop-in replacement no code changes

27% improvement



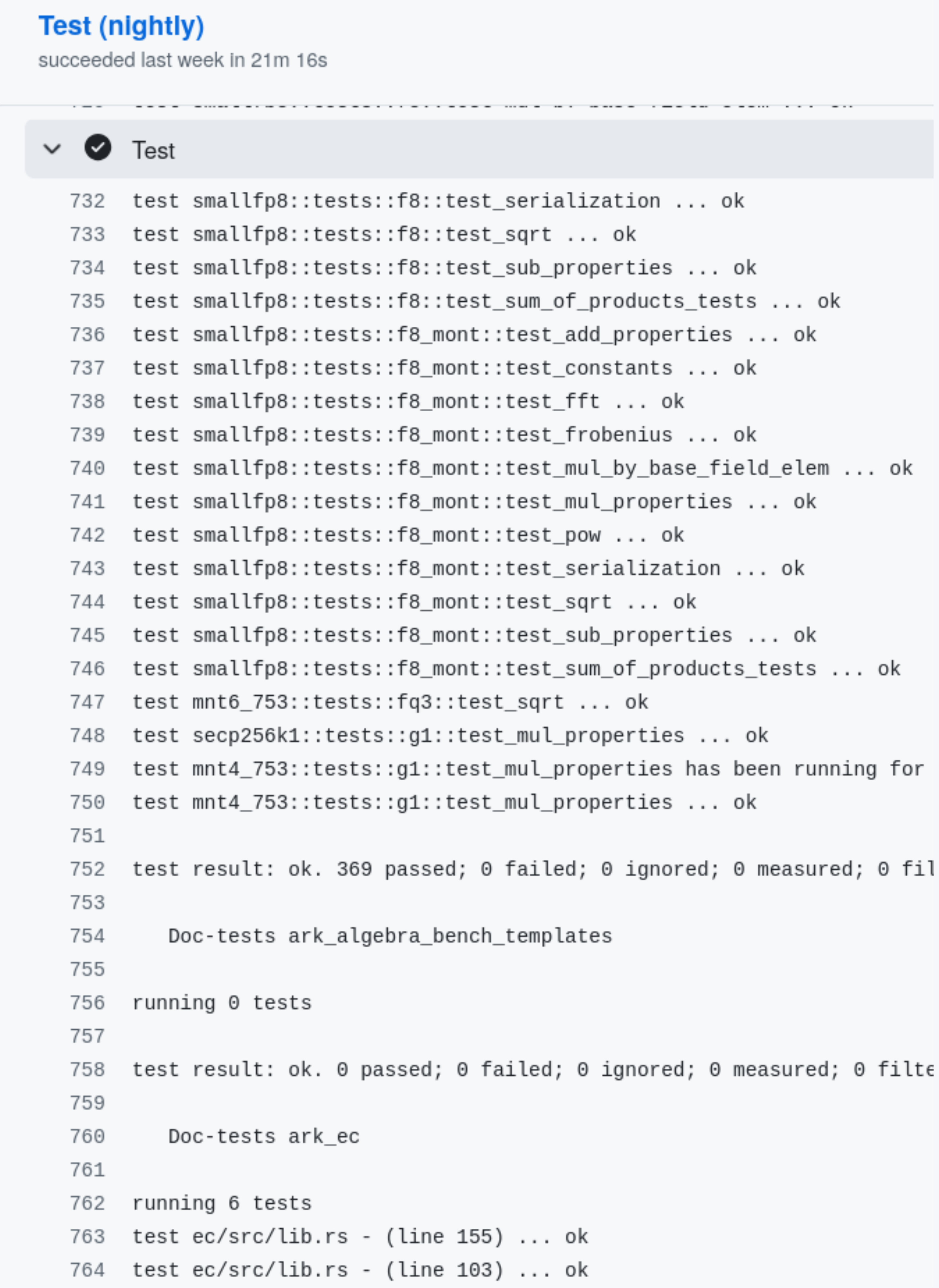
19% improvement

2% improvement

Integration

PR contains tests and benches

- Trait in crate **ark-ff**
- Macro in crate **ff-macros**
- Sample fields added to crate **test-curves**



```
test smallfp8::tests::f8::test_serialization ... ok
test smallfp8::tests::f8::test_sqrt ... ok
test smallfp8::tests::f8::test_sub_properties ... ok
test smallfp8::tests::f8::test_sum_of_products_tests ... ok
test smallfp8::tests::f8_mont::test_add_properties ... ok
test smallfp8::tests::f8_mont::test_constants ... ok
test smallfp8::tests::f8_mont::test_fft ... ok
test smallfp8::tests::f8_mont::test_frobenius ... ok
test smallfp8::tests::f8_mont::test_mul_by_base_field_elem ... ok
test smallfp8::tests::f8_mont::test_mul_properties ... ok
test smallfp8::tests::f8_mont::test_pow ... ok
test smallfp8::tests::f8_mont::test_serialization ... ok
test smallfp8::tests::f8_mont::test_sqrt ... ok
test smallfp8::tests::f8_mont::test_sub_properties ... ok
test smallfp8::tests::f8_mont::test_sum_of_products_tests ... ok
test mnt6_753::tests::fq3::test_sqrt ... ok
test secp256k1::tests::g1::test_mul_properties ... ok
test mnt4_753::tests::g1::test_mul_properties has been running for
test mnt4_753::tests::g1::test_mul_properties ... ok

test result: ok. 369 passed; 0 failed; 0 ignored; 0 measured; 0 fil

   Doc-tests ark_algebra_bench_templates

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filte

   Doc-tests ark_ec

running 6 tests
test ec/src/lib.rs - (line 155) ... ok
test ec/src/lib.rs - (line 103) ... ok
```

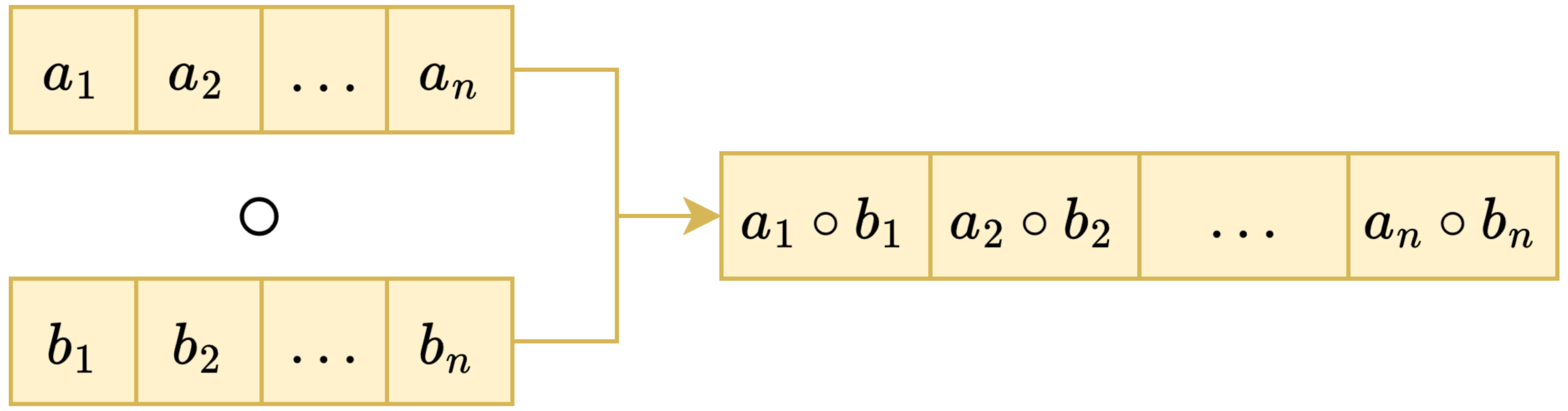
Future Work

1. Prime specific optimizations: Mersenne, Goldilocks, Babybear

2. Vectorized operations

Base + Extension field arithmetic!

Pairwise operation for $a, b \in F_p^n$



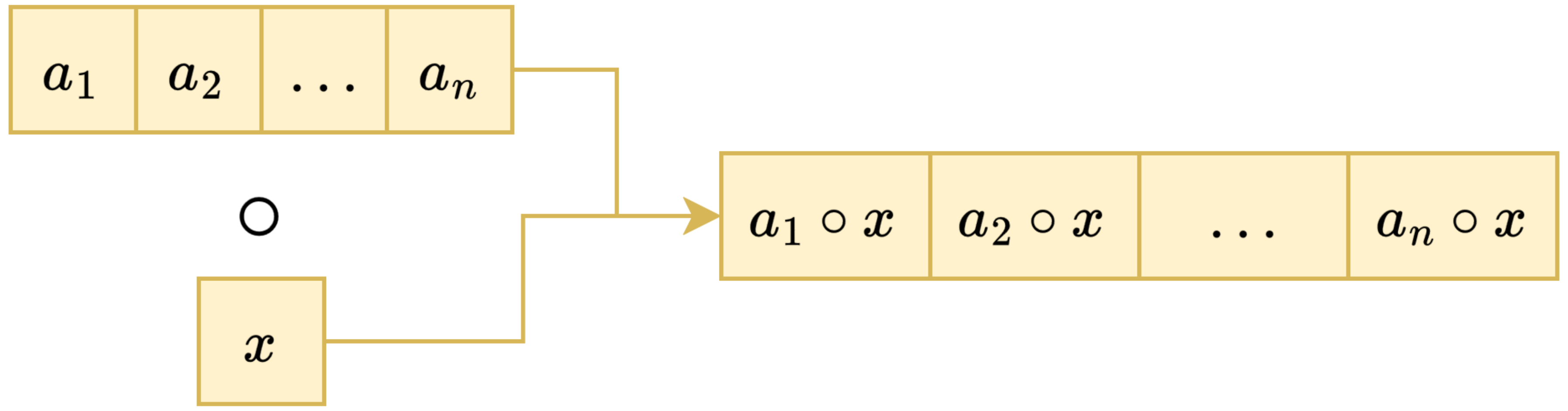
a_1 | a_2 | $\dots$ | a_n

$\circ$

b_1 | b_2 | $\dots$ | b_n

$a_1 \circ b_1$ | $a_2 \circ b_2$ | $\dots$ | $a_n \circ b_n$

Scalar operation for $a \in F_p^n, x \in \mathbb{N}$



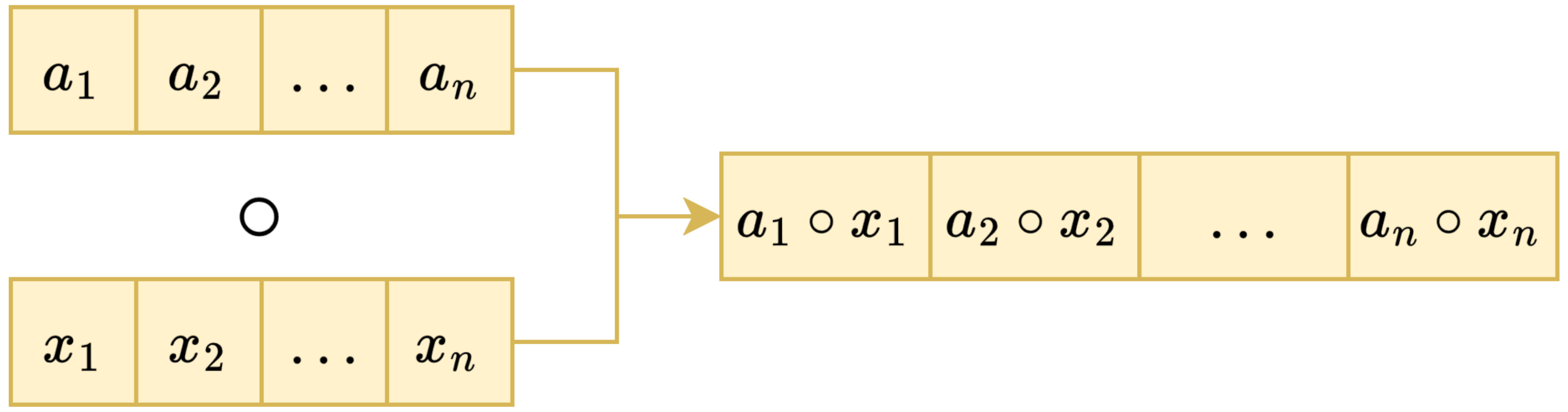
a_1 | a_2 | $\dots$ | a_n

$\circ$

x

$a_1 \circ x$ | $a_2 \circ x$ | $\dots$ | $a_n \circ x$

Multiscalar operation for $a \in F_p^n, x \in \mathbb{N}$



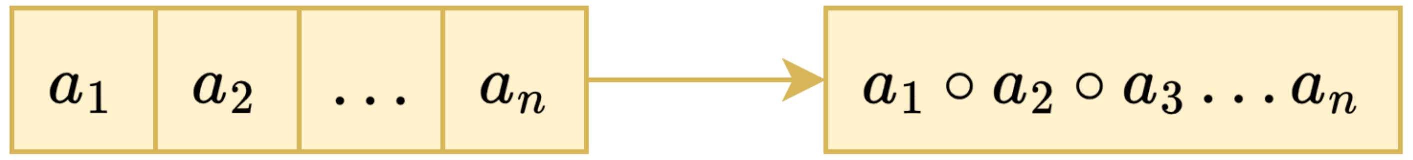
a_1 | a_2 | $\dots$ | a_n

$\circ$

x_1 | x_2 | $\dots$ | x_n

$a_1 \circ x_1$ | $a_2 \circ x_2$ | $\dots$ | $a_n \circ x_n$

Reduction for $a \in F_p$

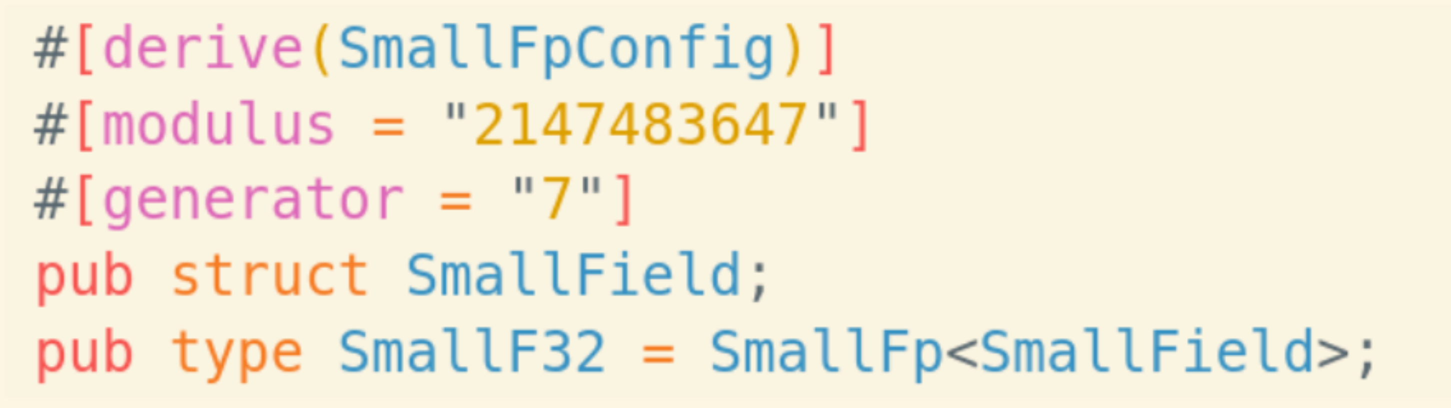


a_1 | a_2 | $\dots$ | a_n

$a_1 \circ a_2 \circ a_3 \dots a_n$

Summary

Recap Arkworks Small Fields



```
#[derive(SmallFpConfig)]
#[modulus = "2147483647"]
#[generator = "7"]
pub struct SmallField;
pub type SmallF32 = SmallFp<SmallField>;
```

New

- SmallFp and its macro are a drop in replacement that implement Field
- Requires no new code and contains no breaking changes
- High-level protocols expect up to 30% serial-runtime improvement for moduli < 128 bits

Bonus side-effect 🚀

- Clear path exists toward vectorization/ SIMD optimizations

Goal achieved 🎯