`Rings`: *efficient Java/Scala library for polynomial rings*

Stanislav Poslavsky

Institute for High Energy Physics NRC "Kurchatov Institute", Protvino, Russia

2018

Rings: *an overview*

- **Computational Number Theory**
  - *primes: sieving, testing, factorization*
  - *univariate polynomials over arbitrary coefficient rings: fast arithmetic, gcd, factorization etc.*
  - *Galois fields*
- **Computational Commutative Algebra**
  - *multivariate polynomials over arbitrary coefficient rings: fast arithmetic, gcd, factorization etc.*
  - *fast rational function arithmetic*
- **Computational Algebraic Geometry**
  - *Gröbner bases*
  - *Ideals in multivariate polynomial rings*
- **Programming in Scala**
  - *object-oriented and functional programming in one concise, high-level and statically typed language*

Rings: *motivation*

— Yet another program for math ?
Really ? What for ???

An incomplete list of similar software:

Closed source (proprietary)	Open source (free)
Magma, Maple, Mathematica, Fermat, ...	Singular, Macaulay2, CoCoa, Reduce, Maxima, Pari/GP, ... FLINT, NTL, FORM, ...

either quite heavyweight interactive CASs or very specialized libraries or very slow

Goals in Rings:

- **Lighweight:** *lighweight, portable, extensible and embeddable library (not a CAS)*
- **Modern:** *concise API which meets modern best programming practices*
- **Fast:** *use asymptotically fast algorithms to achieve the best performance*

In single sentence (this is ads 😎):

— Rings is the fastest library written in the most popular programming language.

Rings: *motivation (a few examples from physics)*

- **Processes at Large Hadron Collider**
  - **Phenomenology**
  - **One-loop integrals: Passarino-Veltman reduction**
  - **(Multi-)loop integrals: modern approaches**

Rings: *motivation (a few examples from physics)*

- **Processes at Large Hadron Collider**
  - **Phenomenology**
    - **Example**: *tree-level associated production of χ_b and open charm*
      - required taking about 10^4 multivariate GCDs of polynomials with $10^1 - 10^5$ terms in $\mathbb{Q}[x_1, ..., x_7]$
      - simplification of analytical results required factorization of polynomials with $10^6 - 10^7$ terms in $\mathbb{Q}[x_1, ..., x_7]$
    - **Example**: *tree-level (NLO*) production of pair of J/ψ and χ_c mesons*
      - required taking 6,145,800 multivariate GCDs of polynomials with $10^1 - 10^5$ terms in $\mathbb{Q}[x_1, ..., x_6]$
      - total time for GCDs was 11 hours (20% of total time), which is only a part of total time spent in rational function arithmetic
      - all this needed just to prepare for further numerical evaluation
  - **One-loop integrals: Passarino-Veltman reduction**
  - **(Multi-)loop integrals: modern approaches**

Rings: *motivation (a few examples from physics)*

- **Processes at Large Hadron Collider**
  - **Phenomenology**
  - **One-loop integrals: Passarino-Veltman reduction**
    - **Example**: *production of four quarks at one loop in gluon fusion*
      - requires to solve up to 86×86 linear system with symbolic coefficients from $\mathbb{Q}[x_1, ..., x_5]$
      - this is already a sort of "record" computation due to intermediate expression swell
      - advanced methods: modular techniques, rational reconstruction, Hensel lifting etc
  - **(Multi-)loop integrals: modern approaches**

Rings: *motivation (a few examples from physics)*

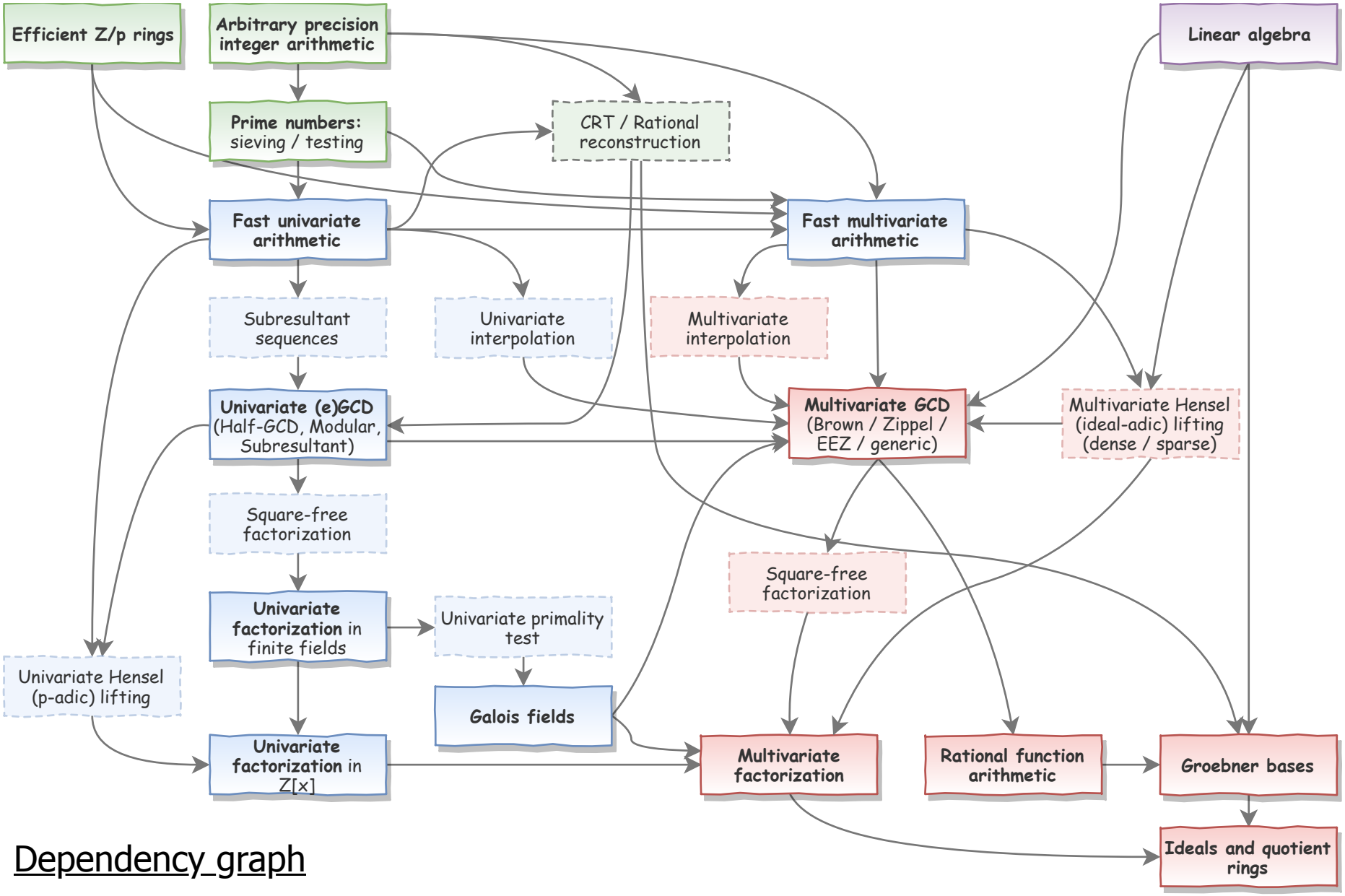
- **Processes at Large Hadron Collider**
  - **Phenomenology**
  - **One-loop integrals: Passarino-Veltman reduction**
  - **(Multi-)loop integrals: modern approaches**
    - **Keywords:**
      - Ideals and Gröbner bases
      - Effective Nullstellensatz
      - Multivariate residues
      - ...

`Rings`: *motivation (a few examples from physics)*

- **Processes at Large Hadron Collider**
  - **Phenomenology**
  - **One-loop integrals: Passarino-Veltman reduction**
  - **(Multi-)loop integrals: modern approaches**

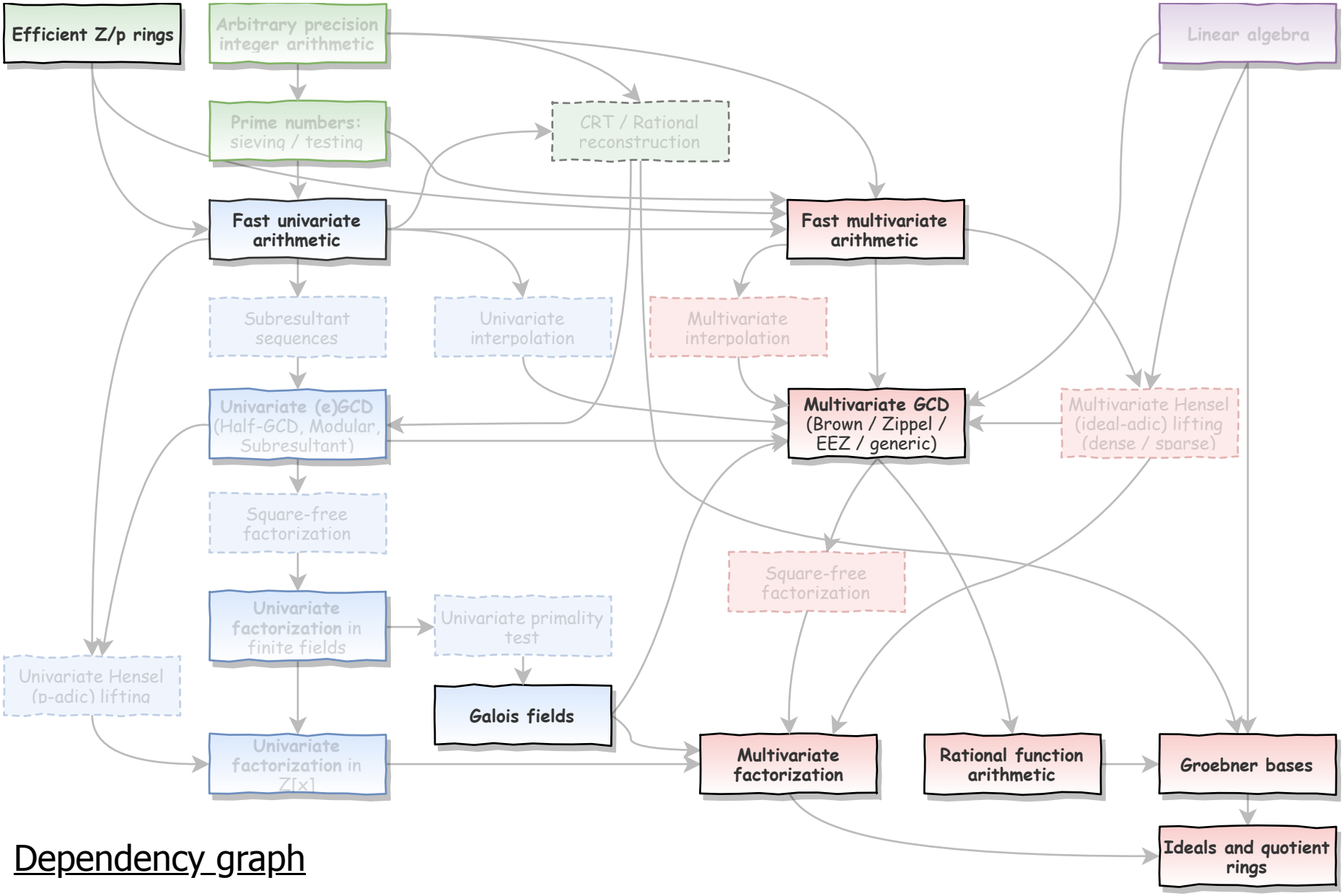
All required mathematical concepts are implemented in `Rings` *with one of the best or even unmatched achieved performance*

Rings: *implementation aspects*



Dependency graph

Rings: *implementation aspects*



Dependency graph

Rings: *design by examples*

Simple example:

```
1  implicit val ring = UnivariateRing(Q, "x") // base ring Q[x]
2  val x = ring("x")              // parse polynomial from string
3  val poly = x.pow(100) - 1      // construct polynomial programmatically
4  val factors = Factor(poly)     // factorize polynomial
5  println(factors)
```

- Explicit types are omitted for shortness, though Scala is fully statically typed

  ```
  val ring : Ring[UnivariatePolynomial[Rational[IntZ]]] = ...
  val poly : UnivariatePolynomial[Rational[IntZ]] = ...
  ```

  (types are inferred automatically at compile time if not specified explicitly)

- Trait Ring[E] implements the concept of mathematical ring and defines all basic algebraic operations over the elements of type E

  ```
  println( ring.isField )          // access ring properties
  println( ring.characteristic ) // access ring characteristic
  println( ring.cardinality )     // access ring cardinality
  ```

- The implicit brings operator overloading via type enrichment (=> continue)

Rings: *design by examples*

Meaning of implicits:

```
1  // ring of elements of type E
2  implicit val ring : Ring[E] = ...
3  val a : E = ...
4  val b : E = ...

6  val sum = a + b // compiles to ring.add(a, b)
7  val mul = a * b // compiles to ring.multiply(a, b)
8  val div = a / b // compiles to ring.divideExact(a, b)
```

Example:

```
1  val a : IntZ = Z(12)
2  val b : IntZ = Z(13)
3  assert (a * b == Z(156))   // no any implicit Ring[IntZ]

5  implicit val ring = Zp(17) // implicit Ring[IntZ]
6  assert (a * b == Z(3))     // multiplication modulo 17
```

Rings: *design by examples*

Multivariate polynomials

```
1   // base ring Q[x, y, z]
2   implicit val ring = MultivariateRing(Q, Array("x", "y", "z"))
3   // parse polynomial from strings
4   val (x, y, z) = ring("x", "y", "z")
5   // construct polynomial programmatically
6   val poly1 = (x + y + z).pow(10) - 1
7   // or again parse from string
8   val poly2 = ring("(x + y + z)^3 + 1")
9   // compute GCD
10  println(PolynomialGCD(poly1, poly2))
11  // factorize multivariate polynomial
12  println(Factor(poly1))

14  // construct some non-trivial ideal
15  implicit val ideal = Ideal(Seq(poly1 - x, poly2 - y), LEX)
16  assert ( ideal.dimension == 1 )
17  // reduce poly modulo ideal
18  assert ( poly1 %% ideal == x )
19  assert ( poly2 %% ideal == y )
```

Rings: *design by examples*

Rational function arithmetic:

```
1   // rational functions Frac(Z[x, y, z])
2  implicit val ring = Frac(MultivariateRing(Z, Array("x", "y", "z")))
3  // parse expressions from strings
4  val (x, y, z) = ring("x", "y", "z")
5  // or construct programmatically
6  val epxr1 = x / y + z.pow(2) / (x + y - 1)
7  // or import from file
8  import scala.io.Source
9  val expr2 = ring(Source.fromFile("myFile.txt").mkString)

11 val expr3 = expr1 * expr2
12 // unique factor decomposition of fraction
13 println ( ring.factor(expr3) )
```

- Fractions are always reduced to a common denominator and GCD is cancelled automatically;

`Rings`: *design by examples*

Built-in ring	**Description**
`Z`	ring of integers
`Q`	field of rationals
`Zp(p)`	integers modulo `p`
`GF(p,q)`	finite field with cardinality p^q
`Frac(R)`	field of fractions over Euclidean ring R
`UnivariateRing(R, x)`	univariate ring $R[x]$
`MultivariateRing(R, vars)`	multivariate ring $R[x_1, x_2, \dots]$
`QuotientRing(R, ideal)`	multivariate quotient ring $R[x_1, x_2, \dots]/I$

Rings: *design by examples*

Diophantine equations: solve $\sum f_i s_i = gcd(f_1, \ldots, f_N)$ for given f_i and unknown s_i:

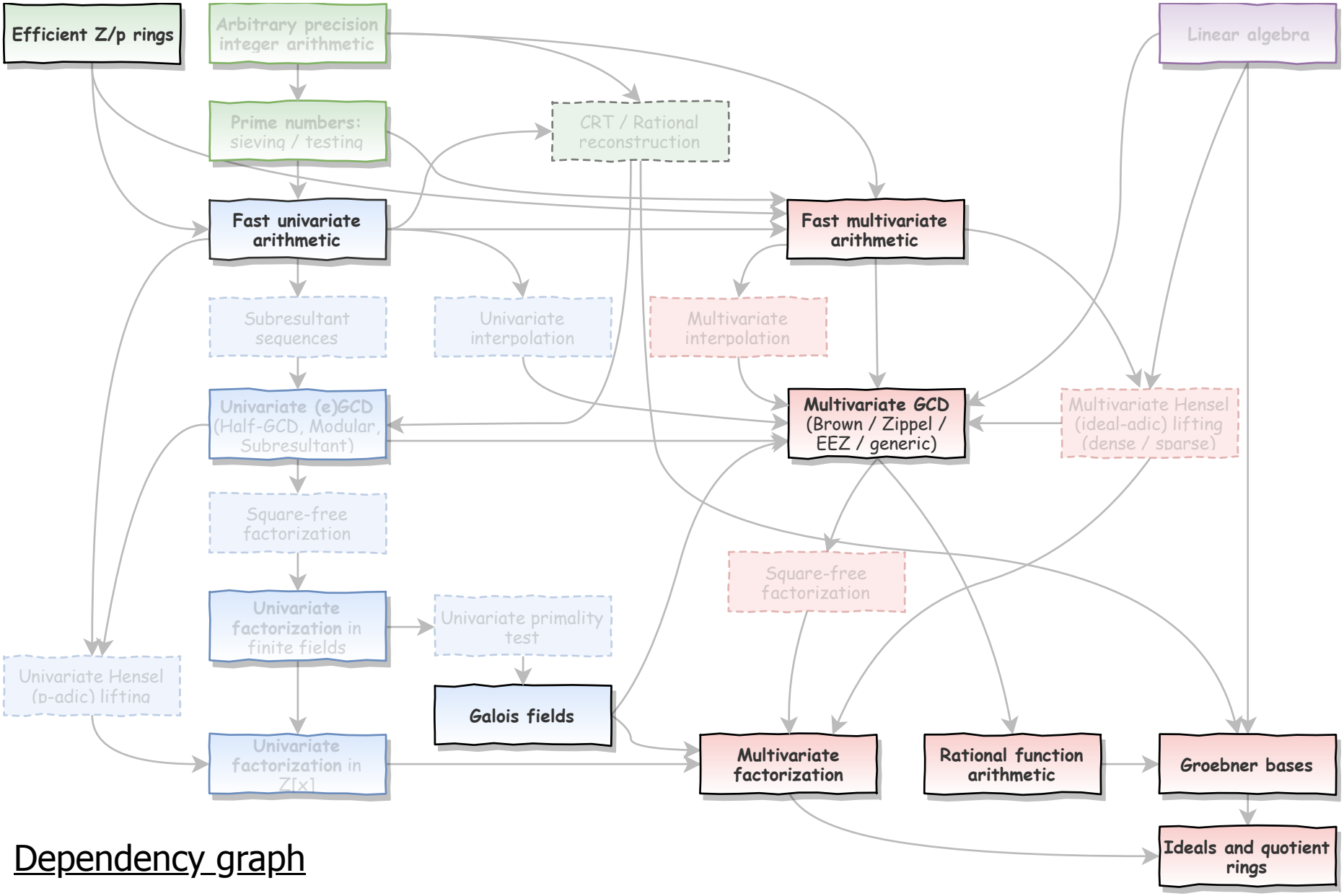
```
1  def solveDiophantine[E](fi: Seq[E])(implicit ring: Ring[E]) =
2    fi.foldLeft((ring(0), Seq.empty[E])) { case ((gcd, seq), f) =>
3      val xgcd = ring.extendedGCD(gcd, f)
4      (xgcd(0), seq.map(_ * xgcd(1)) :+ xgcd(2))
5    }
```

Diophantine equations in $Frac(GF(17^3)[x, y, z])[W]$**:**

```
1  // Galois field GF(17, 3)
2  implicit val gf    = GF(17, 3, "t")
3  // Rational functions in x, y, z over GF(17, 3)
4  implicit val fracs = Frac(MultivariateRing(gf, Array("x", "y", "z")))
5  // univariate ring Frac(GF(17, 3)[x,y,z])[W]
6  implicit val ring  = UnivariateRing(fracs, "W")

8  val f1 = ring("1 + t^2 + x/y - W^2") // parse elements from strings
9  val f2 = ring("1 + W + W^3/(t - x)") // parse elements from strings
10 val f3 = ring("t^2 - x  - W^4")      // parse elements from strings
11 // do the job
12 val solve = solveDiophantine(Seq(f1, f2, f3))
```

Rings: *implementation aspects*



Dependency graph

`Rings`: *modular arithmetic with machine numbers*

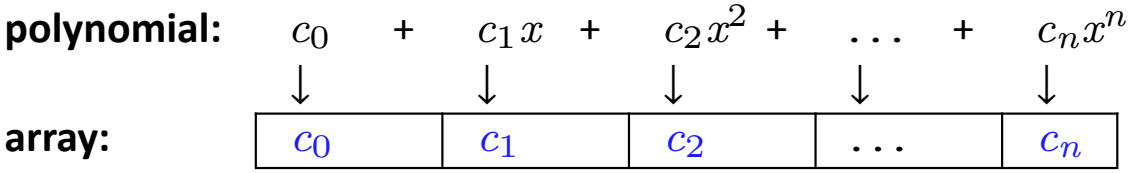
- *Arithmetic in $\mathbb{Z}_p$ with word-sized p ($p < 2^{64}$) lies in the basis of the most part of fundamental algorithms and directly affects performance of all computations*

- $N \mod p \equiv N - \lfloor N/p \rfloor$ — this is how remainder is computed by the CPU
- Integer division (`DIV`) is one of the most inneficient CPU instructions:
  - *it has 20-80 times worth throughput than e.g.* `MUL` *(for Intel Skylake)*
  - *it breaks CPU pipelining*
- **The hack** (*Barret reduction; see Hacker's delight*):
  - Compute *once* the $magic = \lfloor 2^m/p \rfloor$ for sufficiently large m
  - Then $\lfloor N/p \rfloor = (N \times magic)/2^m$ which is one `MUL` and one `SHIFT`
- **Another hack**:
  - $(a \times b)_{\mathbb{Z}_p} = (a \times b) \mod p$ if a and b are less than 2^{32} (fast)
  - else, the Montogomery multiplication is used

- Modular arithmetic in `Rings` is 3-5 times faster than with native CPU instructions and especially fast in $\mathbb{Z}_p$ rings with $p < 2^{32}$

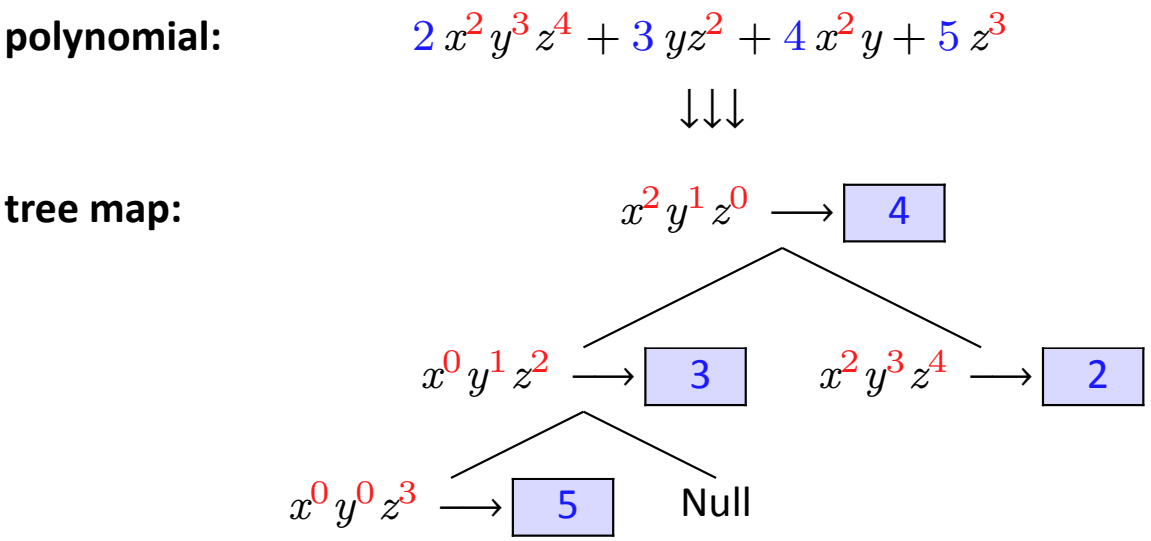
Rings: *polynomials*

- **Univariate polynomials are always dense:**



  - *native arrays are used to store univariate polynomials*

- **Multivariate polynomials are sparse:**



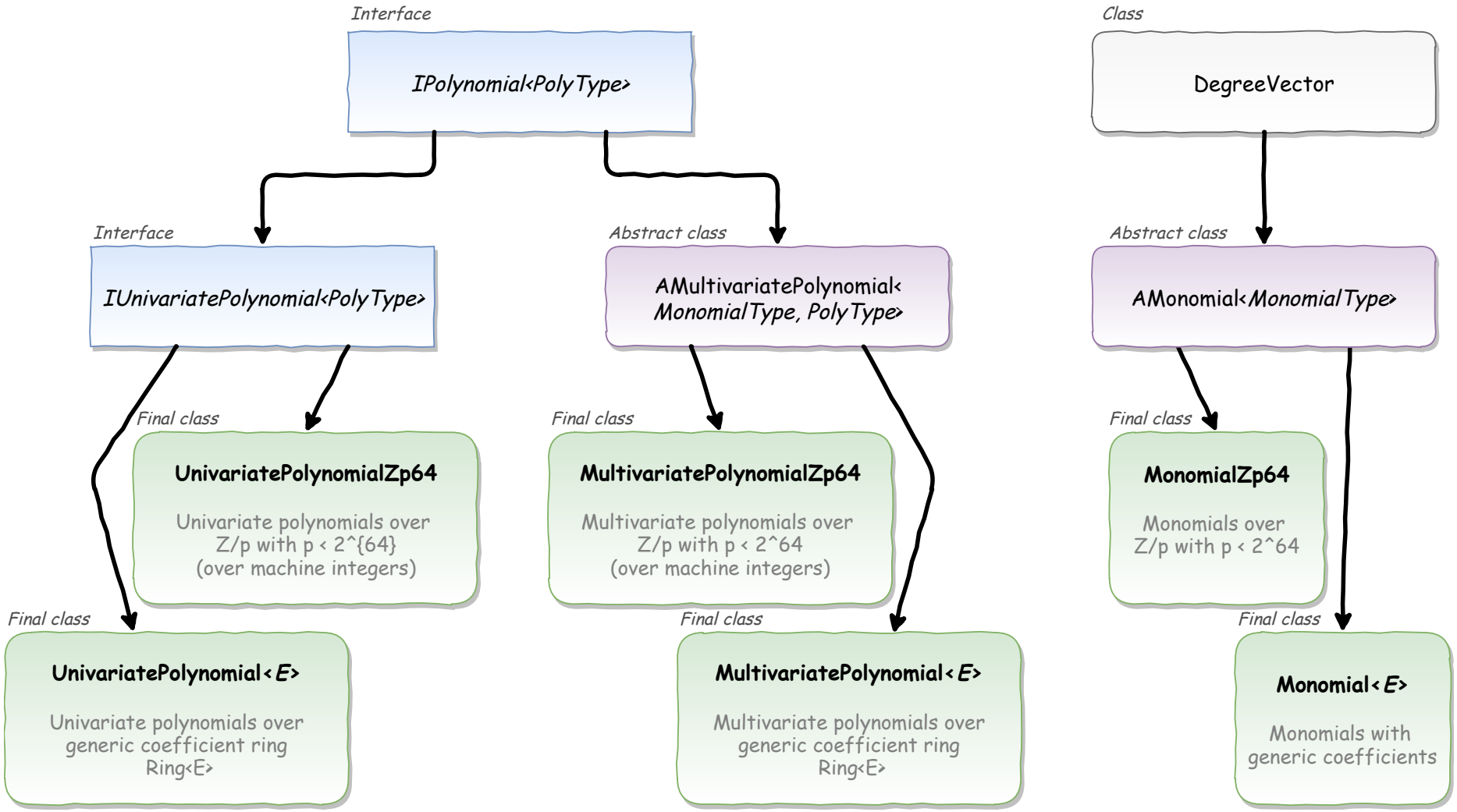
  - *red-black tree map is used to store multivariate terms*

`Rings`: *polynomials*

- Polynomials over $\mathbb{Z}_p$ with $p < 2^{64}$ (machine numbers) have separate implementations
  - `E[] data` — *generic array for univariate polynomials over generic rings (with elements of reference type E)*
  - `long[] data` — *native array for univariate polynomials over $\mathbb{Z}_p$ with $p < 2^{64}$ (machine words)*

- Motivation:
  - $\mathbb{Z}_p$ with $p < 2^{64}$ already has separate implementation
  - more specific and optimized algorithms
  - avoid inefficient generics with primitive types in `Java` (however, e.g. in `C`/`C++` one would have to do the same, like in `NTL`)

Rings: *polynomials*



Rings: *polynomials*

	Univariate (n is polynomial degree)	**Multivariate** (n is polynomial size)
Addition/Subtraction	$O(n)$	$O(n \log n)$
Multiplication	$O\left(n^{\log_2 3}\right)$ via Karatsuba method (with lots of heuristic to reduce the constant)	$O\left(nm \log(n) \log(m)\right)$ via plain method (Kronecker trick is used to significantly reduce the constant)
Division	$O\left(n^{\log_2 3}\right)$ via Newton's iteration (with lots of heuristic to reduce the constant)	$O\left(nm \log(n) \log(m)\right)$ via plain method
Evaluation	$O\left(n\right)$ via Horner method	$O\left(n \log(d)\right)$ via plain method with caching or via recursive Horner scheme

`Rings`: *polynomial GCD*

- **Univariate (e)GCD:**
  - `Rings` switches between Euclidean GCD, Half-GCD and Brown's GCD (for coefficient rings with characteristic zero)
- **Multvariate GCD:**
  - for sparse inputs `Rings` uses Zippel's algorithm based on linear algebra
  - for relatively dense polynomials `Rings` uses Enhanced Extended Zassenhaus (EEZ) approach based on multivariate (ideal-adic) Hensel lifting
  - when the coefficient ring has very small cardinality `Rings` uses a version of Kaltofen-Monagan generic GCD algrotithm
  - for coefficient rings of characteristic zero, modular algrotihm (Zippel-like for sparse or Brown-like with EEZ for dense inputs) is used
  - *all these contain tons of heuristic (code for algorithms spans more than 5,000 l.o.c.)*

Rings: *polynomial GCD*

Benchmarks:

- Generate three polynomials a, b and g at random and compute $gcd(ag, bg)$ (non-trivial) and $gcd(ag + 1, bg)$ (trivial)
- Terms of polynomials are generated independently
- Two ways to generate exponents inside terms:
  - *Uniform exponents* (uniform distribution):
    choose each exponent independently in range $\text{exp}_{\text{min}} \leq \text{exp}_i < \text{exp}_{\text{max}}$; the total degree will be $N_{\text{vars}}\text{exp}_{\text{min}} \leq \text{exp}_{\text{tot}} < N_{\text{vars}}\text{exp}_{\text{max}}$
    Example ($\text{exp}_{\text{min}} = 0$, $\text{exp}_{\text{max}} = 10$):
    $$\cdots + x^5 y^2 z^8 + x^3 y^8 z^6 + \ldots$$
  - *Sharp exponents* (multinomial distribution):
    choose the total degree exp_{tot}, then for the first variable $0 \leq \text{exp}_1 \leq \text{exp}_{\text{tot}}$, for the second variable $0 \leq \text{exp}_2 \leq (\text{exp}_{\text{tot}} - \text{exp}_1)$ and so on
    Example ($\text{exp}_{\text{tot}} = 10$):
    $$\cdots + x^7 y^2 z^1 + x^0 y^8 z^2 + \ldots$$

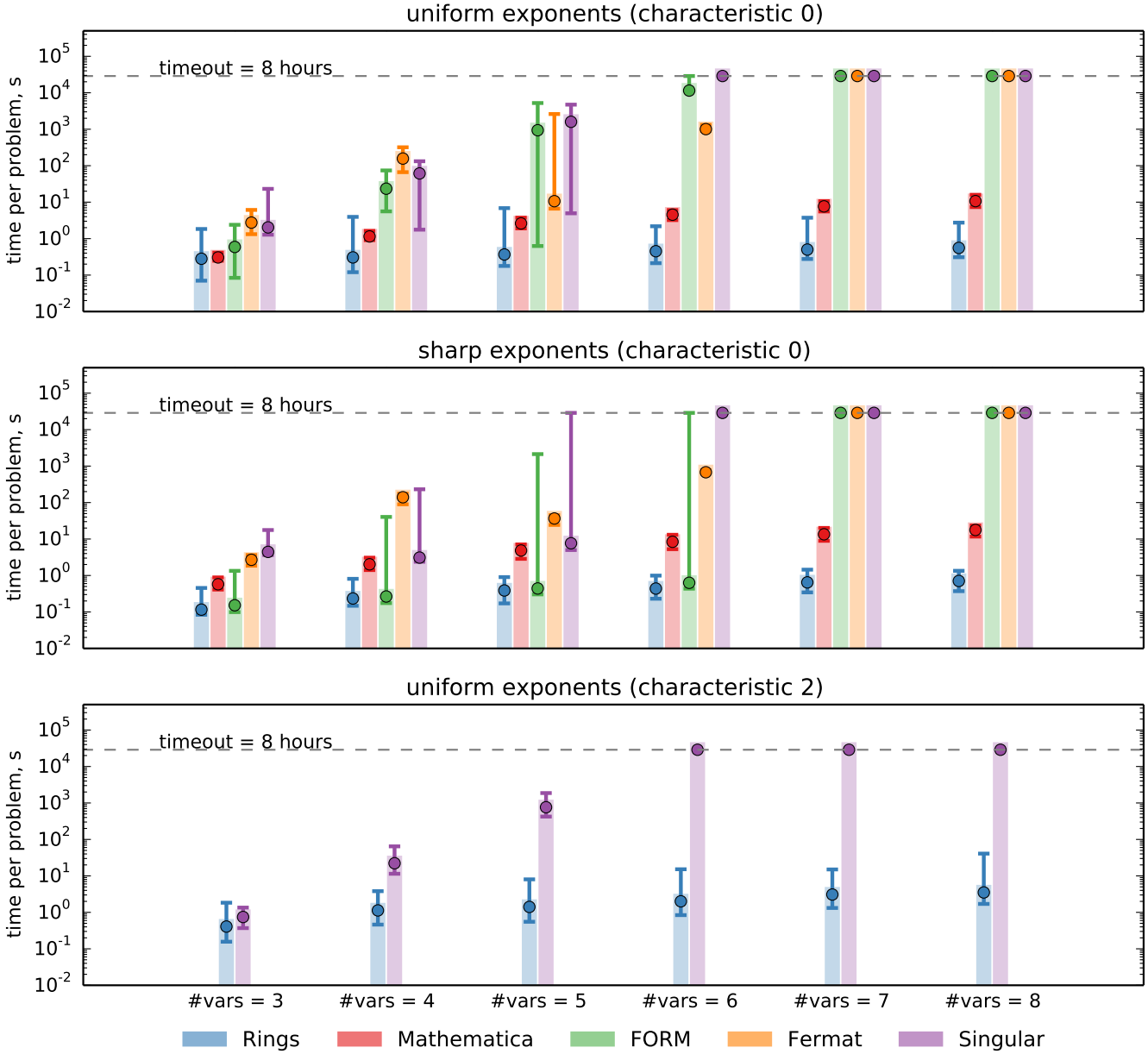
Rings: *polynomial GCD*

Params (a,b,g):

#terms = 40
#bits = 32
exp_{min} = 0
exp_{max} = 30

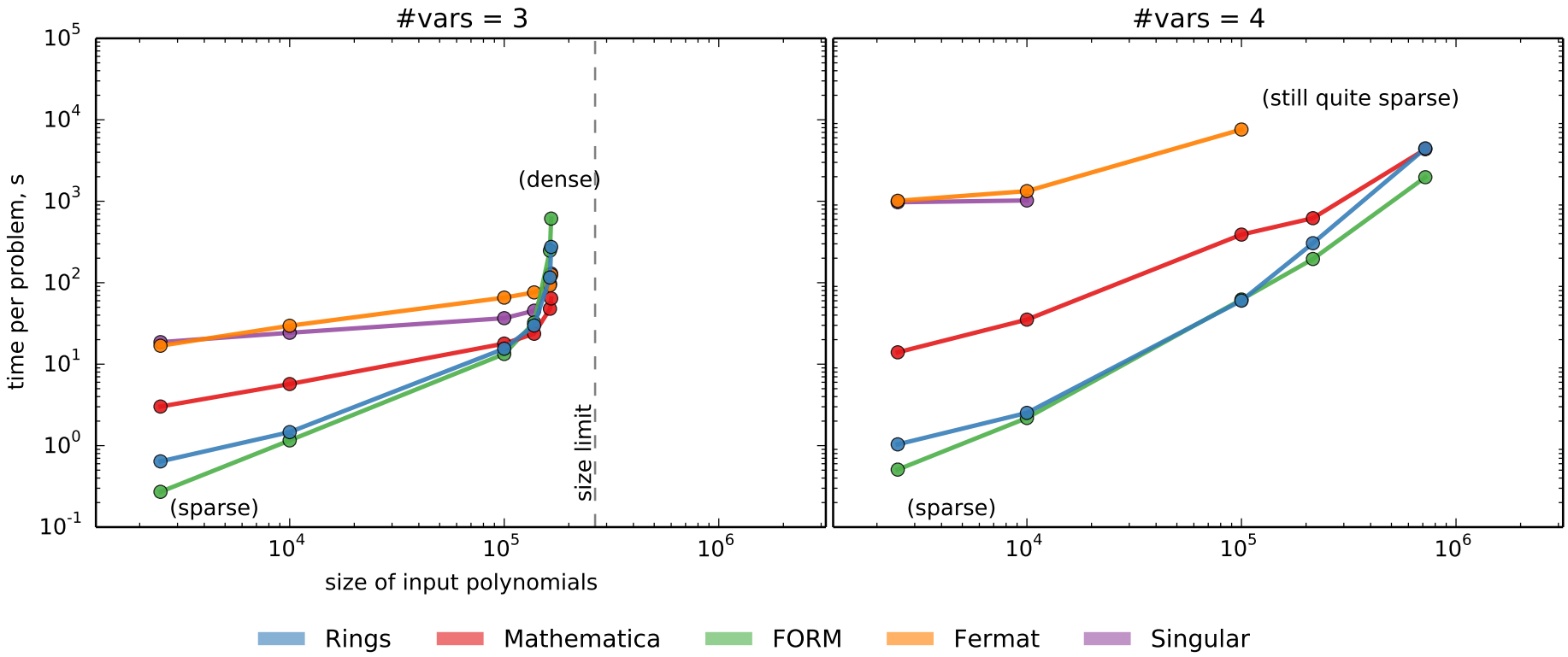
#terms = 40
#bits = 32
exp_{tot} = 50

#terms = 40
#bits = 1
exp_{min} = 0
exp_{max} = 30



Rings: *polynomial GCD*

“Record” problems:



Params (a,b,g):

exp_{tot} = 50 / #bits = 128 / #terms = 50, 100, 500, 1000, 5000

Rings: *polynomial GCD*

Dense input:

$$a = (1 + 3x_1 + 5x_2 + 7x_3 + 9x_4 + 11x_5 + 13x_6 + 15x_7)^7 - 1$$
$$b = (1 - 3x_1 - 5x_2 - 7x_3 + 9x_4 - 11x_5 - 13x_6 + 15x_7)^7 + 1$$
$$g = (1 + 3x_1 + 5x_2 + 7x_3 + 9x_4 + 11x_5 + 13x_6 - 15x_7)^7 + 3$$

Problem	Cf. ring	Rings	Mathematica	FORM	Fermat	Singular
$gcd(ag, bg)$	$\mathbb{Z}$	104s	115s	148s	1759s	141s
$gcd(ag, bg+1)$	$\mathbb{Z}$	0.4s	2s	0.3s	0.1s	0.4s
$gcd(ag, bg)$	$\mathbb{Z}_{524287}$	25s	33s	N/A	147s	46s
$gcd(ag, bg+1)$	$\mathbb{Z}_{524287}$	0.5s	2s	N/A	0.2s	0.2s

Rings: *polynomial GCD*

Dense input:

$$a = (1 + 3x_1 + 5x_2 + 7x_3 + 9x_4 + 11x_5 + 13x_6 + 15x_7)^7 - 1$$
$$b = (1 - 3x_1 - 5x_2 - 7x_3 + 9x_4 - 11x_5 - 13x_6 + 15x_7)^7 + 1$$
$$g = (1 + 3x_1 + 5x_2 + 7x_3 + 9x_4 + 11x_5 + 13x_6 - 15x_7)^7 + 3$$

Problem	Cf. ring	Rings	Mathematica	FORM	Fermat	Singular
$gcd(ag, bg)$	$\mathbb{Z}$	104s	115s	148s	1759s	141s
$gcd(ag, bg+1)$	$\mathbb{Z}$	0.4s	2s	0.3s	0.1s	0.4s
$gcd(ag, bg)$	$\mathbb{Z}_{524287}$	25s	33s	N/A	147s	46s
$gcd(ag, bg+1)$	$\mathbb{Z}_{524287}$	0.5s	2s	N/A	0.2s	0.2s

- GCD performance on trivial input is very important (since e.g. most part of GCDs computed in rational function arithmetic are trivial)

Rings: *polynomial GCD*

Dense input:

$$a = (1 + 3x_1 + 5x_2 + 7x_3 + 9x_4 + 11x_5 + 13x_6 + 15x_7)^7 - 1$$
$$b = (1 - 3x_1 - 5x_2 - 7x_3 + 9x_4 - 11x_5 - 13x_6 + 15x_7)^7 + 1$$
$$g = (1 + 3x_1 + 5x_2 + 7x_3 + 9x_4 + 11x_5 + 13x_6 - 15x_7)^7 + 3$$

Problem	Cf. ring	Rings	Mathematica	FORM	Fermat	Singular
$gcd(ag, bg)$	$\mathbb{Z}$	104s	115s	148s	1759s	141s
$gcd(ag, bg+1)$	$\mathbb{Z}$	0.4s	2s	0.3s	0.1s	0.4s
$gcd(ag, bg)$	$\mathbb{Z}_{524287}$	25s	33s	N/A	147s	46s
$gcd(ag, bg+1)$	$\mathbb{Z}_{524287}$	0.5s	2s	N/A	0.2s	0.2s

- GCD performance on trivial input is very important (since e.g. most part of GCDs computed in rational function arithmetic are trivial)
- one have to make a trade-off between performance on non-trivial and trivial inputs

`Rings`: *polynomial factorization*

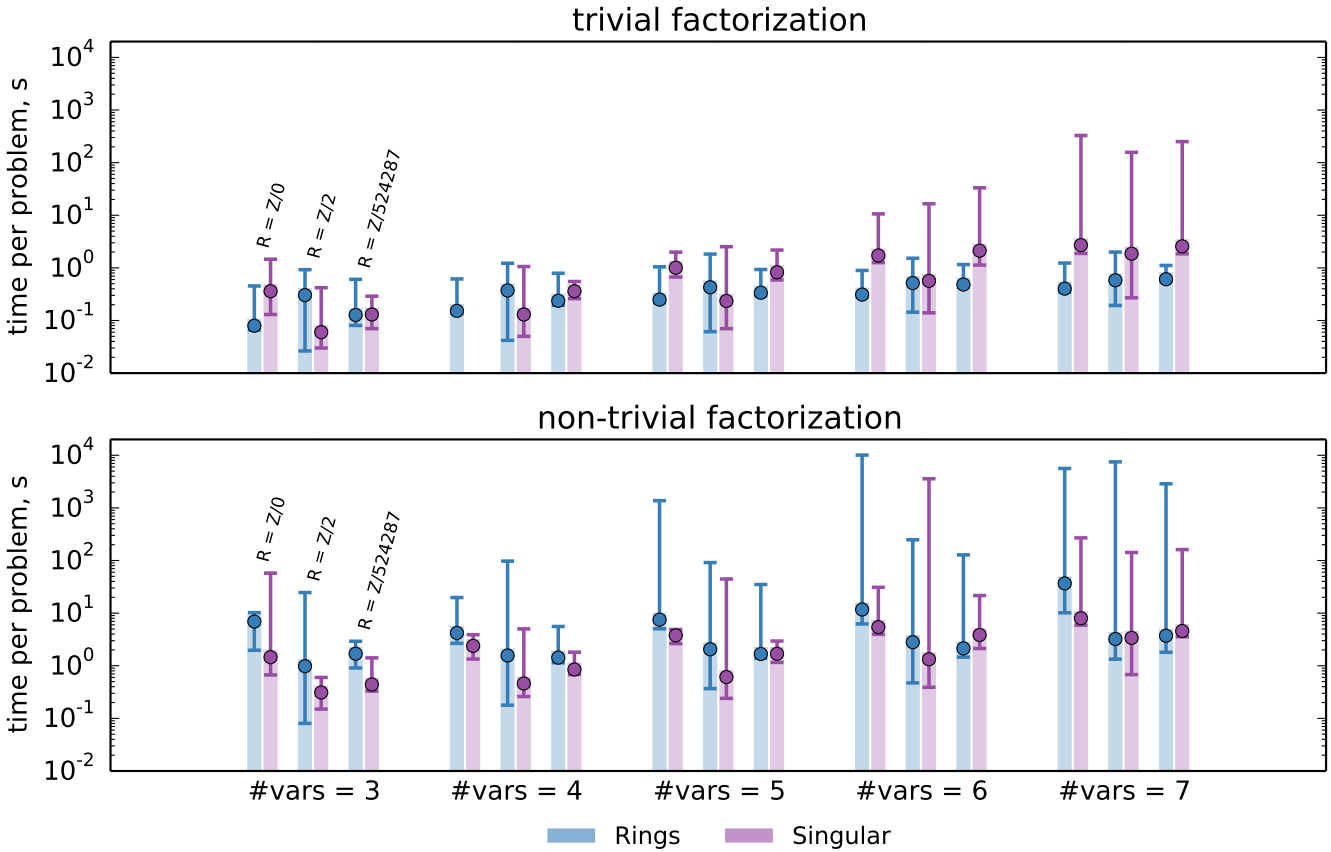
- **Univariate factorization:**
  - `Rings` switches between Cantor-Zassenhaus and Shoup's baby-step-giant-step algorithms for polynomials over finite fields
  - p-adic Hensel lifting is used to compute factorization over $\mathbb{Z}$ (resp. $\mathbb{Q}$)
- **Multvariate factorization:**
  - for bivariate polynomials Bernardin's algorithm is used
  - Kaltofen's algorithm is used in all other cases
  - ideal-adic Hensel lifting switches between sparse (based on linear algebra) and dense (based on Bernardin's algorithm)
  - *all these contain tons of heuristic*

Rings: *polynomial factorization*

Benchmark: generate three polynomials a, b and c at random and compute $factor(abc)$ (non-trivial) and $factor(abc+1)$ (trivial)

Params:

#factors = 3
#terms = 20
exp_{min} = 0
exp_{max} = 30

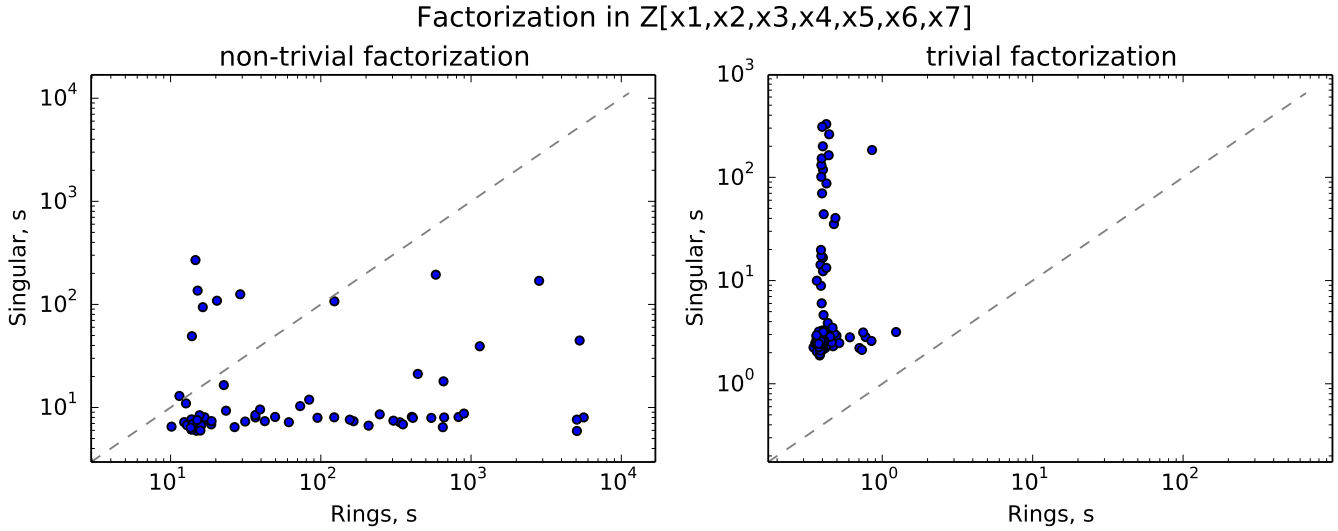


Rings: *polynomial factorization*

Benchmark: generate three polynomials a, b and c at random and compute $factor(abc)$ (non-trivial) and $factor(abc+1)$ (trivial)

Params:

#factors = 3
#terms = 20
exp_{min} = 0
exp_{max} = 30



Rings: *polynomial factorization*

Dense input:

$$p_1 = (1 + 3x_1 + 5x_2 + 7x_3 + 9x_4 + 11x_5 + 13x_6 + 15x_7)^{15} - 1$$

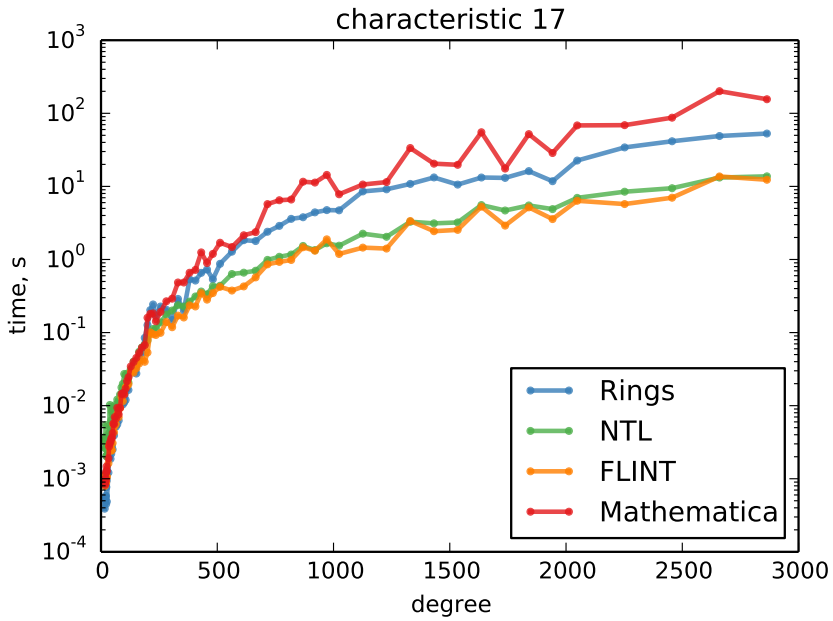
$$p_2 = -1 + (1 + 3x_1x_2 + 5x_2x_3 + 7x_3x_4 + 9x_4x_5 + 11x_5x_6 + 13x_6x_7 + 15x_7x_1)^3$$
$$\times(1 + 3x_1x_3 + 5x_2x_4 + 7x_3x_5 + 9x_6x_5 + 11x_7x_6 + 13x_6x_1 + 15x_7x_2)^3$$
$$\times(1 + 3x_1x_4 + 5x_2x_5 + 7x_3x_6 + 9x_6x_7 + 11x_7x_1 + 13x_6x_2 + 15x_7x_3)^3$$

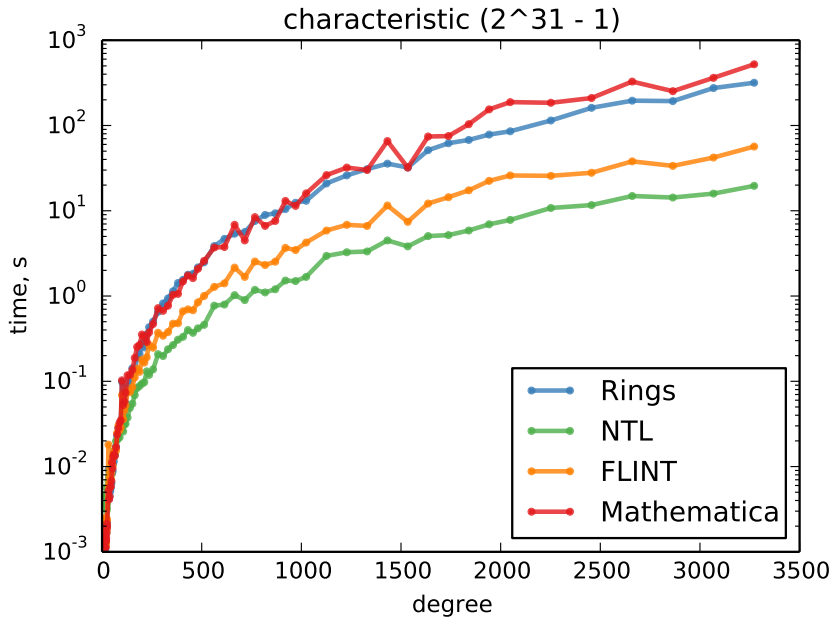
Problem	Cf. ring	Rings	Singular	Mathematica
$factor(p_1)$	$\mathbb{Z}$	55s	20s	271s
$factor(p_1)$	$\mathbb{Z}_2$	0.25s	> 1h	N/A
$factor(p_1)$	$\mathbb{Z}_{524287}$	28s	109s	N/A
$factor(p_2)$	$\mathbb{Z}$	23s	12s	206s
$factor(p_2)$	$\mathbb{Z}_2$	6s	3s	N/A
$factor(p_2)$	$\mathbb{Z}_{524287}$	26s	9s	N/A

Rings: *polynomial factorization*

Univariate input:

$$p_{\deg}[x] = 1 + \sum_{i=1}^{i \le \deg} i \times x^i$$





- This benchmark covers almost all aspects of univariate arithmetic in finite fields

`Rings`: *Gröbner bases*

- **Note**: `Rings` is not optimized for computing Gröbner bases for "challenging" problems yet (like those arise in post-quantum cryptography)
- Gröbner bases for graded orders for polynomials over finite fields computed with Faugere's F4 algorithm (hardly based on fast sparse linear algebra)
- In other cases `Rings` may switch between Buchberger algorithm (with different selection strategies), Hilbert-driven methods or modular algorithms
- Again, many heuristics applied

Problem	Cf. ring	`Rings`	`Mathematica`	`Singular`
cyclic-7	$\mathbb{Z}_{1000003}$	3s	26s	N/A
cyclic-8	$\mathbb{Z}_{1000003}$	51s	897s	39s
cyclic-9	$\mathbb{Z}_{1000003}$	14603s	∞	8523s
katsura-7	$\mathbb{Z}_{1000003}$	0.5s	2.4s	0.1s
katsura-8	$\mathbb{Z}_{1000003}$	2s	24s	1s
katsura-9	$\mathbb{Z}_{1000003}$	2s	22s	1s
katsura-10	$\mathbb{Z}_{1000003}$	9s	216s	9s
katsura-11	$\mathbb{Z}_{1000003}$	54s	2295s	65s
katsura-12	$\mathbb{Z}_{1000003}$	363s	28234s	677s
katsura-7	$\mathbb{Z}$	5s	4s	1.2s
katsura-8	$\mathbb{Z}$	39s	27s	10s
katsura-9	$\mathbb{Z}$	40s	29s	10s
katsura-10	$\mathbb{Z}$	1045s	251s	124s

`Rings`: *final technical aspects*

- **90,929** lines of code: **83,018** (`.java`) **+ 7,911** (`.scala`)
- Licensed under **Apache 2.0**
- Sources at **GitHub**:

  https://github.com/PoslavskySV/rings
- Comprehensive documentation at **RTD**:

  https://rings.readthedocs.io
- Covered by **tens of thousands of tests**:

  http://circleci.com/gh/PoslavskySV/rings (CI)
- Interactive **REPL**:

```
sh> brew install PoslavskySV/rings/rings.repl
```

Rings: *conclusions*

- **Computational Number Theory**
  - *primes: sieving, testing, factorization*
  - *univariate polynomials over arbitrary coefficient rings: fast arithmetic, gcd, factorization etc.*
  - *Galois fields*
- **Computational Commutative Algebra**
  - *multivariate polynomials over arbitrary coefficient rings: fast arithmetic, gcd, factorization etc.*
  - *fast rational function arithmetic*
- **Computational Algebraic Geometry**
  - *Gröbner bases*
  - *Ideals in multivariate polynomial rings*
- **Programming in Scala**
  - *object-oriented and functional programming in one concise, high-level and statically typed language*

Rings: *conclusions*

- **Computational Number Theory**
  - *primes: sieving, testing, factorization*
  - ***univariate polynomials over arbitrary coefficient rings: fast arithmetic, gcd, factorization etc.***
  - *Galois fields*
- **Computational Commutative Algebra**
  - ***multivariate polynomials over arbitrary coefficient rings: fast arithmetic, gcd, factorization etc.***
  - *fast rational function arithmetic*
- **Computational Algebraic Geometry**
  - ***Gröbner bases***
  - *Ideals in multivariate polynomial rings*
- **Programming in Scala**
  - ***object-oriented and functional programming in one concise, high-level and statically typed language***

With one of the best or even unmatched performance