FastParsers: Accelerating Parser Combinators with Macros

Eric Béguet
Manohar Jonnalagedda

Scala2014

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Parsing? In 2014?

Parsing? In 2014?


```
{
  "statuses": [
    {
      "text": "RT @dark_dimius: Martin @odersky announces
future development directions of #scala #miniboxing
#scalamacros #scalablitz #dotty http:\/\/t.co\/\u2026",
      "user": {
      },
    },
    {
       "text": "Insighful interview with Jason Zaugg
(@retronym) on #Scala, Dotty and @intellijidea. Async is
mentioned, too. Enjoy! blog.jetbraings.com/scala/...",
    }
  ],
  ...
}
```

Parsing? In 2014?

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFW
GATVITNLFSAIPYIGTNLV
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGL
TSDSDKIPFHPYYTIKDFLG
LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAIL
RSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIG
QMASILYFSIILAFLPIAGX
IENY
```

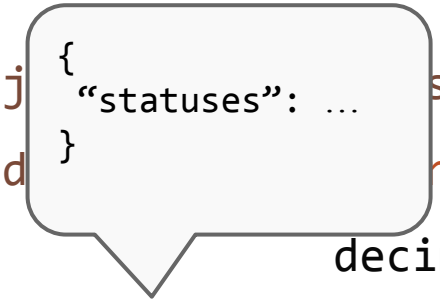
Parsing in 2014!

For Processing Structured Data

Parser Combinators are Awesome

```
object JSON extends JavaTokenParsers {
  def value: Parser[Any] = obj | arr | stringLit |
                decimalNumber | "null" | "true" | "false"
  def obj: Parser[Any] = "{" ~> repsep (member, ",") <~ "}"
  def arr: Parser[Any] = "[" ~> repsep (value , ",") <~ "]"
  def member: Parser[Any] = stringLit ~ (":" ~> value)
}
```

Parser Combinators are Awesome




```
obj                    s JavaTokenParsers {
  d                    r[Any] = obj | arr | stringLit |
                 decimalNumber | "null" | "true" | "false"
  def obj: Parser[Any] = "{" ~> repsep (member, ",") <~ "}"
  def arr: Parser[Any] = "[" ~> repsep (value , ",") <~ "]"
  def member: Parser[Any] = stringLit ~ (":" ~> value)
}
```

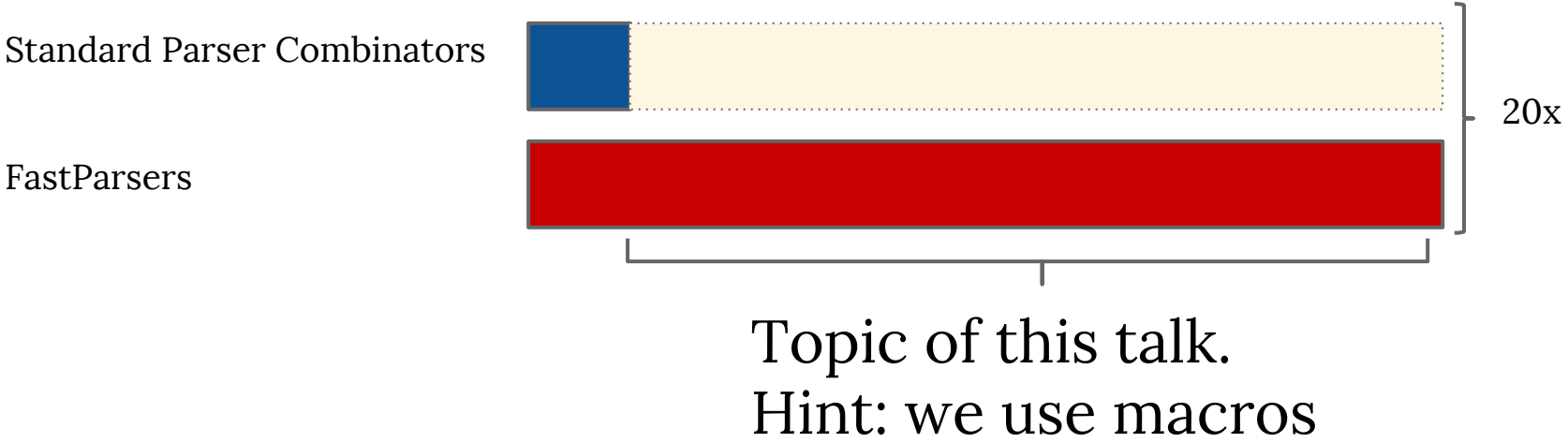
Parser Combinators are Awesome

```
obj                    s JavaTokenParsers {
  d                    r[Any] = obj | arr | stringLit |
                 decimalNumber | "null" | "true" | "false"
  def obj: Parser[Any] = "{" ~> repsep (member, ",") <~ "}"
  def arr: Parser[Any] = "[" ~> repsep (value , ",") <~ "]"
  def membe                stringLit ~ (":" ~> value)
}
```

```
{
 "statuses": …
}
```

```
[
 {"text" : …},
 {"text" : …}
]
```

Parser combinators are slow



Goals

- Make Scala's parser combinators faster
- Stay as close as possible to the current interface
- Benchmarks

Parser Combinators are slow

```
object JSON extends JavaTokenParsers {
  def value: Parser[Any] = obj | arr | stringLit |
                decimalNumber | "null" | "true" | "false"
  def obj: Parser[Any] = "{" ~> repsep (member, ",") <~ "}"
  def arr: Parser[Any] = "[" ~> repsep (value , ",") <~ "]"
  def member: Parser[Any] = stringLit ~ (":" ~> value)
}
```

Parser Combinators are slow

```
class Parser[T] extends
(Input => ParseResult[T])
{ ... }
```

```
object JSO                    rsers {
  def value: Parser[Any] = obj | arr | stringLit |
                 decimalNumber | "null" | "true" | "false"
  def obj: Parser[Any] = "{" ~> repsep (member, ",") <~ "}"
  def arr: Parser[Any] = "[" ~> repsep (value , ",") <~ "]"
  def member: Parser[Any] = stringLit ~ (":" ~> value)
}
```

Parser Combinators are slow

```
class Parser[T] extends
(Input => ParseResult[T])
{ ... }
```

```
object JSO                    rsers {
  def value: Parser[Any] = obj | arr | stringLit |
                 decimalNumber | "null" | "true" | "false"
  def obj: Parser[Any] = "{" ~> repsep (member, ",") <~ "}"
  def arr: Parser[Any] = "[" ~> repsep (value , ",") <~ "]"
  def member: Parser[Any] = stringLit ~ (":" ~> value)
}
```

```
def ~[U](that: Parser[U])
= new Parser[(T,U)] {
    def apply(i: Input) =
...
  }
```

Parser Combinators are slow

```
class Parser[T] extends
(Input => ParseResult[T])
{ ... }
```

```
object JSO                                  rsers {
  def value: Parser[Any] = obj | arr | stringLit |
                 decimalNumber | "n
  def obj: Parser[Any] = "{" ~>                                  }"
  def arr: Parser[Any] = "[" ~>                                  ]"
  def member: Parser[Any] = str
}
```

```
def | (that: Parser[T]) = new
Parser[T] {
  def apply(in: Input) = {
    val tmp = this(in)
    if(tmp.isEmpty) that(in)
    else tmp
  }
}
```

```
    def apply(i: Input) =
  ...
  }
```

Parser Combinators are slow

```
object JSON extends JavaTokenParsers {
  def value: Parser[Any] = obj | arr | stringLit |
                decimalNumber | "null" | "true" | "false"
  def obj: Parser[Any] = "{" ~> repsep (member , ",") <~ "}"
  def arr: Parser[Any] = "[" ~> repsep (value , ",") <~ "]"
  def member: Parser[Any] = stringLit ":" ~> value)
}
```

Composition is
(mostly) static

Example (1)

Whitebox macro

```
val jsonParser = FastParser {
  def value: Parser[Any] = obj | arr | stringLit |
                decimalNumber | "null" | "true" | "false"
  def obj: Parser[Any] = "{" ~> repsep (member, ",") <~ "}"
  def arr: Parser[Any] = "[" ~> repsep (value , ",") <~ "]"
  def member: Parser[Any] = stringLit ~ (":" ~> value)
}
```

Example (2): Generated code

Generated parser object

```
val jsonParser = new FinalFastParser {
  def value(input: String, offset: Int = 0): ParseResult[Any]

     @saveAST(obj | arr | .. | "true" | "false") = ..
  def obj(input: String, offset: Int = 0): ParseResult [Any]

     @saveAST("{" ~> repsep (member, ",") <~ "}") = ..
  def arr(input: String, offset: Int = 0): ParseResult [Any]

     @saveAST("[" ~> repsep (value , ",") <~ "]") = ..
  def member(input: String, offset: Int = 0): ParseResult [Any]

     @saveAST(stringLit ~ (":" ~> value )) = ..
}
```

Example (2): Generated code

Generated parser object

```
val jsonParser = new FinalFastParser {
  def value(input: String, offset: Int = 0): ParseResult[Any]

    @saveAST(obj | arr | .. | "true" | "false") = ..
  def obj(input: String, offset: Int = 0): ParseResult [Any]

    @saveAST("{                                  ,") <~ "}") = ..
  def arr(input:                                  0): ParseResult [Any]

    @saveAST("[                                  ,") <~ "]") = ..
  def member(input: String, offset: Int = 0): ParseResult [Any]

    @saveAST(stringLit ~ (":" ~> value )) = ..
}
```

Specialized for String inputs

Example (3) : usage

```
val cnt = "{\"firstName \": \"John\" , \"age\": 25}"
jsonParser.value (cnt) match {
    case Success ( result ) =>
        println ("success : " + result )
    case Failure ( error ) =>
        println (" failure : " + error )
}
```

3 transformation phases

1. Rule transformation
2. Rule rewriting
3. Putting it all together

Rule transformation (1)

Inlining

```
def rule1 = ‘a’ ~ ‘b’
def rule2 = ‘x’ | rule1

def rule3 = ‘y’ ~ rule4
def rule4 = rule3 | ‘x’
```

rule2 is *inlined* to:

```
def rule2 = ‘x’ | (‘a’ ~ ‘b’)
```

Rule transformation (2)

External calls

```
val parser1 = FastParser {
    def rule1 = ’a ’ ~ ’b ’
    def rule2 = ’d ’ ~ rule1
}
val parser2 = FastParser {
    def rule1 = ’c ’ ~ parser1.rule2
}
```

```
def rule1(..) @saveAST(‘a’ ~ ‘b’)
def rule2(..) @saveAST(‘d’ ~ ‘rule1’)
```

Rule rewriting (1)

```
def rule = ‘a’ ~ ‘b’
```

Transform rule into imperative code

```
def expand(tree : c . Tree , rs : ResultsStruct): c.Tree = tree match {
    case q"$lhs ~[$_] $rhs" =>
        q"""
         ${expand(lhs, rs)}
         if (success) {
             ${expand(rhs, rs)}
         }
        """
}
```

Rule rewriting (1)

```
def rule = ‘a’ ~ ‘b’
```

Transform rule into imperative code

```
def expand(tree : c . Tree , rs : ResultsStruct): c.Tree = tree match {
    case q"$lhs ~[$_] $rhs" =>
      q"""
       ${expand(lhs, rs)}
       if (success) {
           ${expand(rhs, rs)}
       }
      """
}
```

Quasiquotes

Quasiquotes

Rule rewriting (2)

```
if (inputpos < inputsize && input(inputpos) == 'a'){
	result1 = 'a'
	inputpos += 1
	success = true
} else
	//error code
if (success){
	if (inputpos < inputsize && input(inputpos) == 'b'){
		result2 = 'b'
		inputpos += 1
		success = true
	} else
		//error code
}
```

Putting it all together

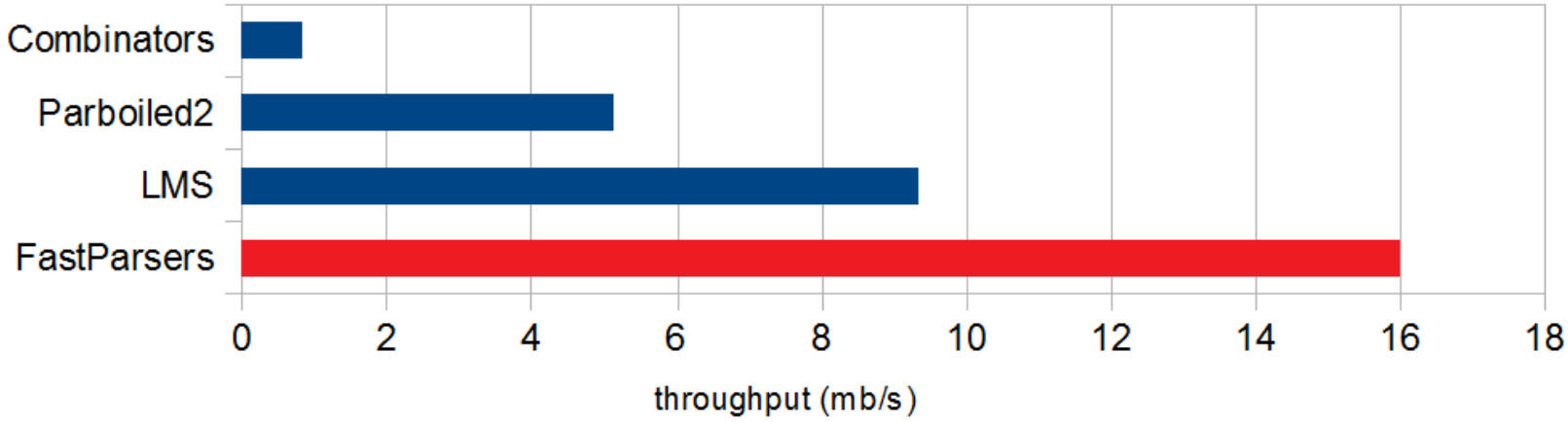
- transform rule into “real” method
- save the original AST
- group all methods and create the parser object

```
new FinalFastParser {
 def rule1: Parser [(Char, Char)];
 def rule1(input: Array[Char], offset: Int): ParseResult[(Char, Char)] @saveAST(’a’ ~ ’b’)
 def rule2: Parser[( Char, (Char, Char))];
 def rule2(input: Array[Char], offset: Int): ParseResult[(Char, (Char, Char))]
     @saveAST(’a’ ~ rule1)
}
```

Optimizing String handling

```
class InputWindow[Input](val in: Input, val start: Int, val end: Int){
    override def equals(x: Any) = x match {
        case s : InputWindow[Input] =>
            s.in == in &&
            s.start == start &&
            s.end == end
        case _ => super.equals(x)
    }
}
```

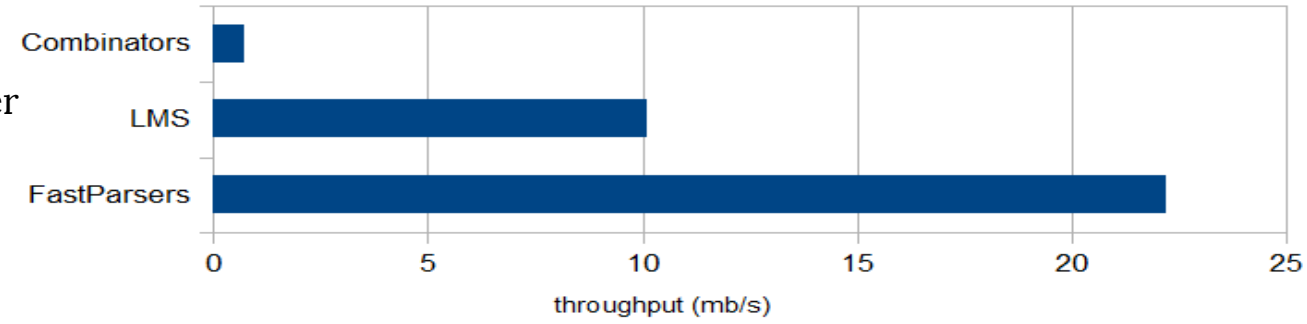
Results: JSON benchmark



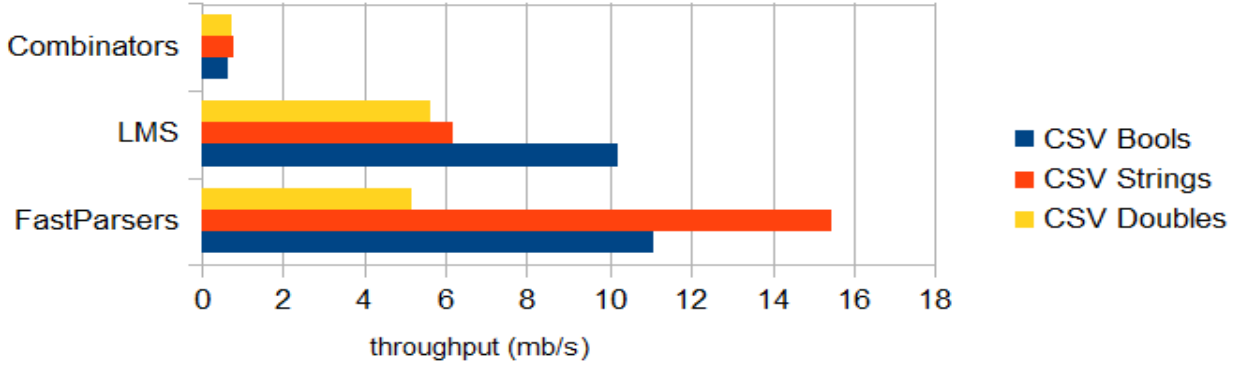
- 20 times faster than Scala's parser combinators
- 3 times faster than Parboiled2
- 2 times faster than LMS version

Results: Benchmarks

HTTP parser



CSV parser



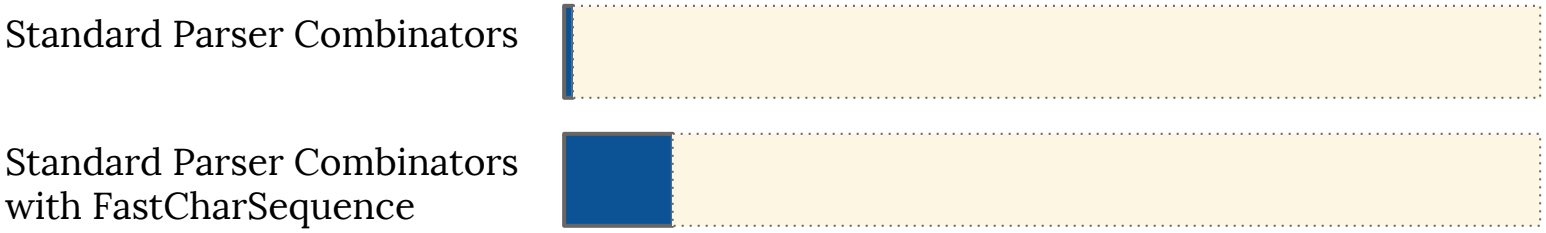
Key performance impactors

Standard Parser Combinators

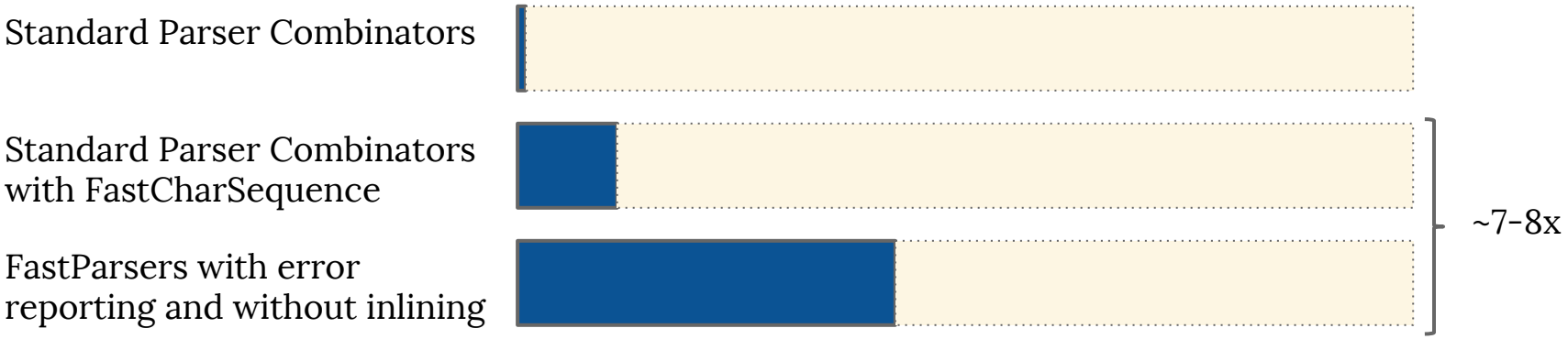
Beware!

- String.substring is in linear time (>= Java 1.6).
- Parsers on Strings are inefficient.
- Need to use a FastCharSequence which mimics original behaviour of substring.

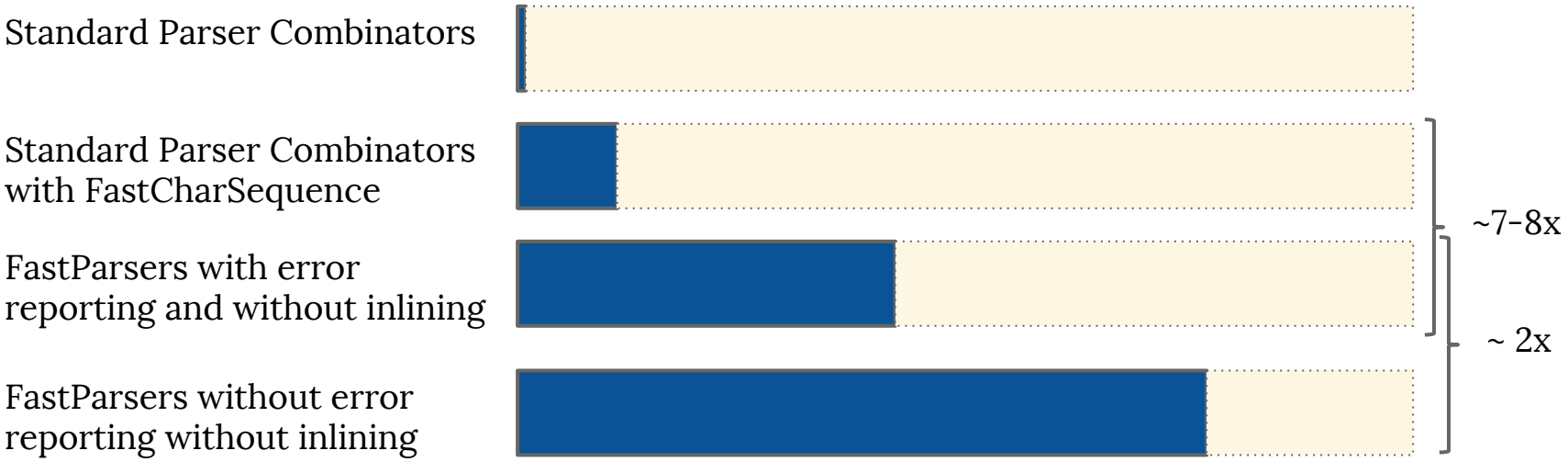
Key performance impactors



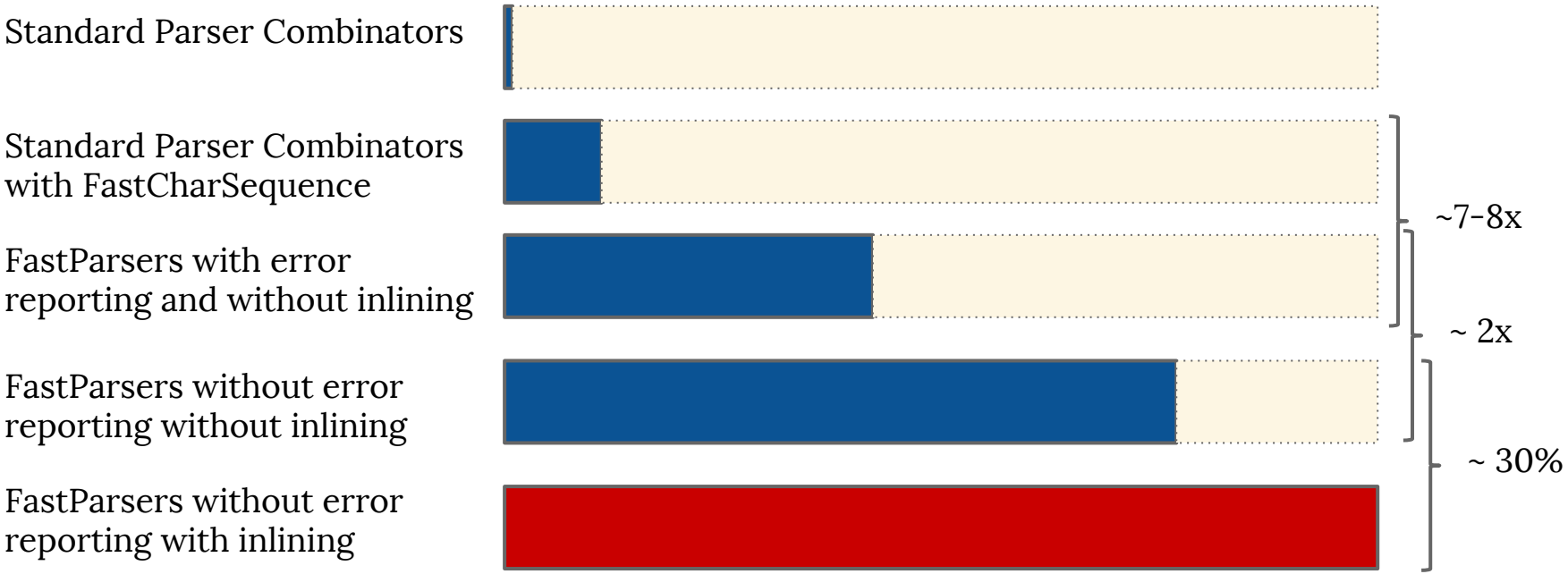
Key performance impactors



Key performance impactors



Key performance impactors



Current limitations

- Separating usage and implementation
  - Separate projects for both
  - Extending optimized combinator from the user side

More in the paper on ...

- Higher order combinators (flatMap)
- Rules with parameters
- Comparison of staging and macro implementations
- Modular input and error types

Future work

- Better error handling
- Profiling
- Generate other kinds of parsers (LALR, GLL..)
- Incremental parsing

Thank you !

Questions ?

Related work

- LMS
- Parboiled2
- Scheme and LALR table generation

The flatmap combinator

```
FastParser {
    def rule = number flatMap { n => {println("stuff"); take(n)} }
}
```

```
lhs.flatMap { params => { body; ret }}
lhs.flatMap { case a => {..; ret1}; case b => {..;
ret2} }
```

- expand lhs
- expand ret
- apply the result of lhs the the rhs function

Rules with parameters

```
FastParser {
     def rule3(p: Parser[List[Char]], y: Int): Parser[Any] = ’a’ ~ p ~ rule4(y)
     def rule4(x: Int): Parser[Any] = rule3(rep(’c’ , x , x), x + 1) | ’b’
}
def rule3(input: String, p: (String, Int) => ParseResult[List[Char]], y: Int, offset:
Int = 0) = ..
```

`rep(’c’, x, x)` has to be expanded into a separated rule =>

```
FastParser {
     def rule3(p: Parser[List[Char]] , y: Int): Parser[Any] = ’a’ ~ p ~ rule4(y)
     def rule4(x: Int): Parser[Any] = rule3(anonymous$1$, x + 1) | ’b’
     def anonymous$1$(x: Int) = rep(’c’ , x, x)
}
```