WeTransfer

Disclaimer

WeTransfer will not look into your transfer or files, unless this is necessary and in line with our Privacy & Cookie Statement, for instance to provide you with (technical) support on your request or if we are by law obliged to do so

Is it a TIFF?

MRI of your spine?

RAW file from your camera?

JPEG file with EXIF tag containing a thumbnail?

At a certain point "just download" was not enough

You want people to be able to present their work and you want to offer choices: what to download if you are pressed for time.

Help | About us | Plus

Coast to coast roadtrip

13 items · 128 MB · No delete date

Download all

DSCF5624.jpg
2.7 MB · jpg

6:37 — 14:28

Voice recording 14
2.7 MB · jpg

DSCF5625.cr2
12.7 MB · cr2

Could Tesla Power Its Electric Truck With Solar Panels?

Tesla just keeps making cool things. On the top of the list is its newest addition to the lineup, an all-electric semitruck. Oh, that might sound like a dumb idea-but I don't think so. Just consider how much stuff is shipped back and forth across the country.

wired.com

Coast to coast roadtrip

13 items · 128 MB · No delete date

Hey G, some ideas for Twigs' new music video, coming at you from the I-95. Let me know what you... more

13 items
Preview open

Download all

What do we need to know?

Without downloading it to the server in full?

— File format
— Resolution for images
— Composition for archives
— Duration for video/audio
— Page count for PDFs

Gimme thumbnails

Before loading an image into your image processing library you want at least to know how large it is going to be.

Reminder: with 95% of image processing libraries your RAM use will be at the very least this (assuming 8-bit color channels and RGBA):

```
width_px * height_px * 4
```

So an unompressed buffer for a 512x512 image is 1048576 bytes.

File processing can be ugly

- CPU-intensive
- IO-intensive
- Relies on a multitude of libraries, libraries might contain exploits
- Large dependencies

We can't download the entire file and run detection on it

EC2 instances have slow, network-connected disks. These disks fill up and get slower at burst workload. We also pay a ton for the traffic from S3 and back.

File format detection

- Airbnb binary_alert and YARA
- exiftool
- magic_bytes (JS) https://github.com/sindresorhus/file-type
- magic_bytes (.rb) https://github.com/julik/magic_bytes

File properties detection

- FastImage - https://github.com/sdsykes/fastimage
- Dimensions - https://github.com/sstephenson/dimensions
- imagesize - https://rubygems.org/gems/imagesize/
- exiftool 😂

exiftool

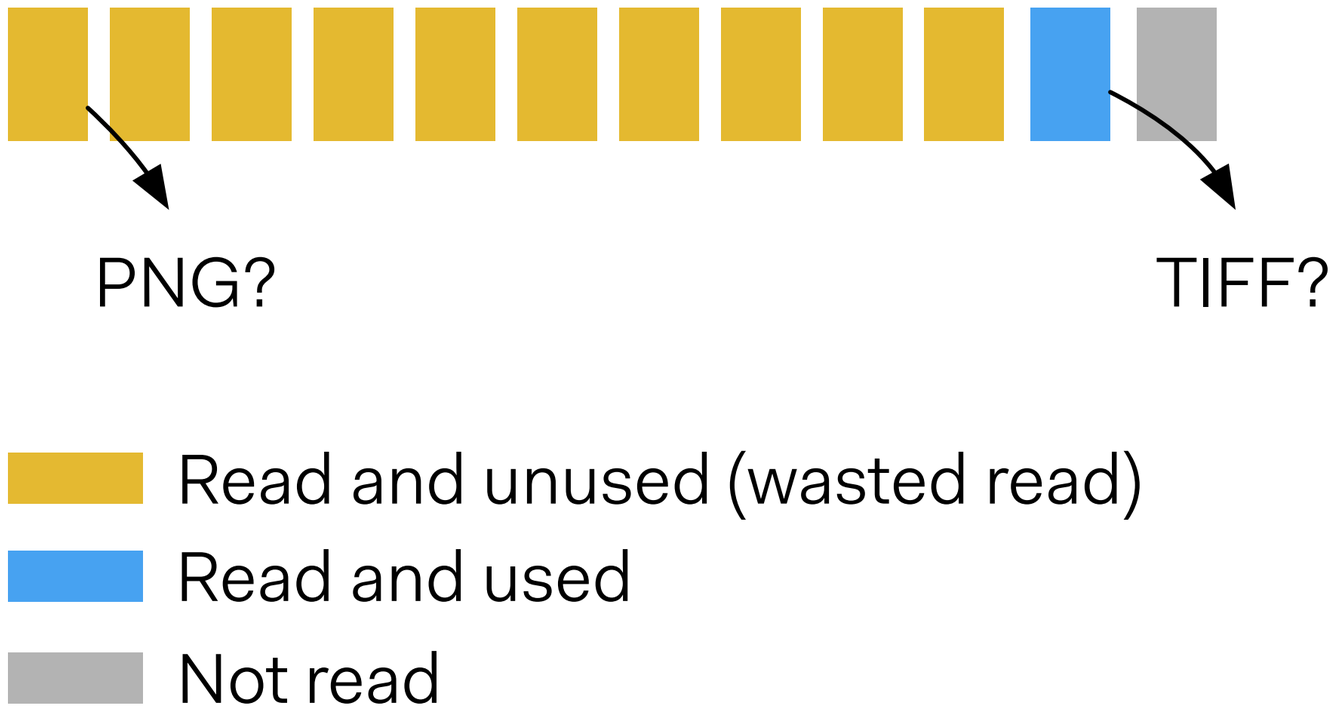
Completely amazing but assumes local files. Perl 🌶️

But if you want to know how to do this amazingly well, and HTTP reads are not required - use efixtool. **Really.** It is that good.

Being clever, but in a wrong way for the use case

Let's have a state machine that reads the file starting at the beginning, Switch to the proper format once we have conclusive evidence

Straight-ahead scanning



Where being clever hurts even more

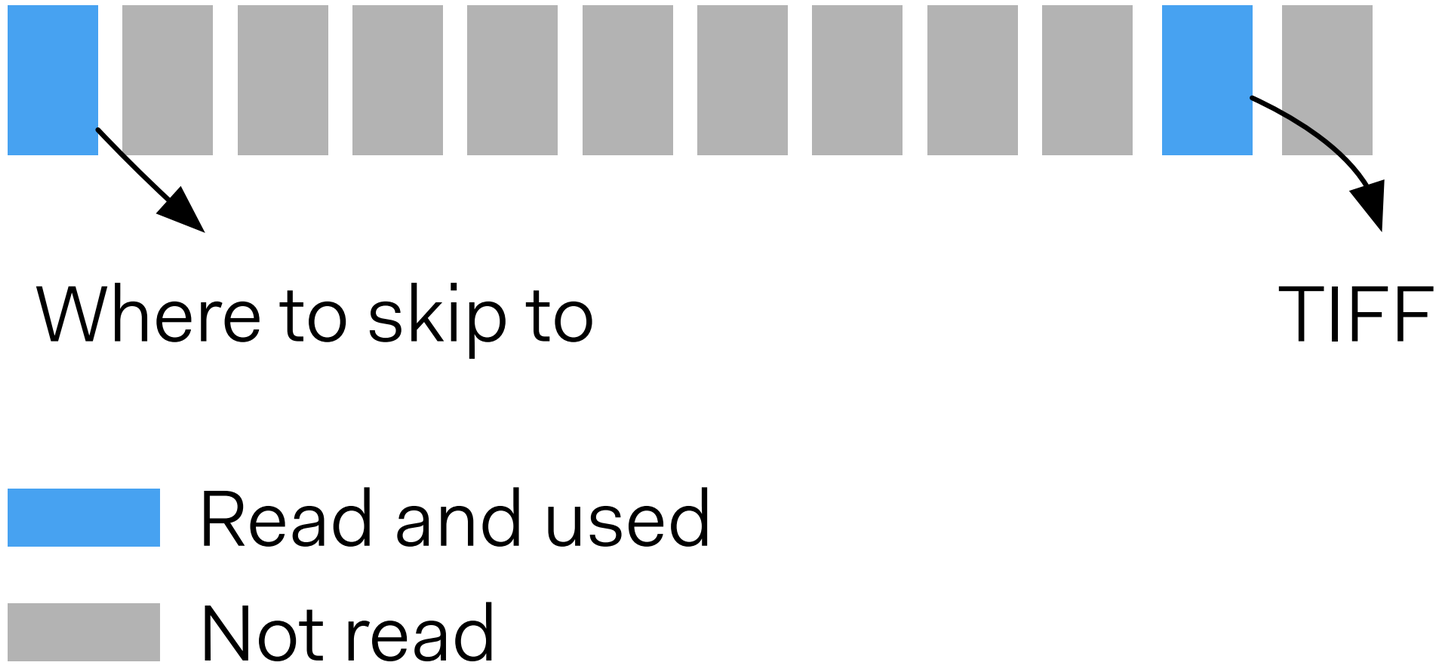
Scanning all the formats is in the same body of code. Integrating support for a different format gets ever harder the more formats you have.

```ruby
def check_type(img_top)
  if img_top =~ /^GIF8[7,9]a/                                then Type::GIF
  elsif img_top[0, 8] == "\x89PNG\x0d\x0a\x1a\x0a" then Type::PNG
  elsif img_top[0, 2] == "\xFF\xD8"                          then Type::JPEG
  elsif img_top[0, 2] == 'BM'                                then Type::BMP
  elsif img_top =~ /^P[1-7]/                                 then Type::PPM
  elsif img_top =~ /\#define\s+\S+\s+\d+/                    then Type::XBM
  elsif img_top[0, 4] == "MM\x00\x2a"                        then Type::TIFF
  elsif img_top[0, 4] == "II\x2a\x00"                        then Type::TIFF
  elsif img_top =~ /\/\* XPM \*\//                           then Type::XPM
  elsif img_top[0, 4] == "8BPS"                              then Type::PSD
  elsif img_top[1, 2] == "WS"                                then Type::SWF
  elsif img_top[0] == 10                                     then Type::PCX
  else Type::OTHER
  end
end
```

And cleverer still

To scan ahead FastImage uses a fiber-wrapped HTTP request, to permit it piecewise reads.

Less clever, but more useful - assume random access



Random access via HTTP is totally possible

If your server supports `Range:` request headers you already have it. All cloud storage providers support that header, as do most CDNs.

```
GET /img.jpg
Range: bytes 0-14

Content-Length: 13
Content-Range: bytes 0-14/657899
```

So, what do we want?

- Random-access via HTTP using `Range:` and support for local files
- One isolated module per supported file format - not a page-long case statement
- Some protection from crafted malicious payloads

format_parser

Random access everywhere

What is a Ruby IO?

```
methods_from_ancestors = IO.ancestors[1..-1].inject([])  do |s, anc|
  s + anc.public_instance_methods
end

(IO.public_instance_methods - methods_from_ancestors).length #=> 96
```

To implement an IO stub we would need to implement 96 methods 😱

Let's define the **smallest possible subset of IO for our use:**

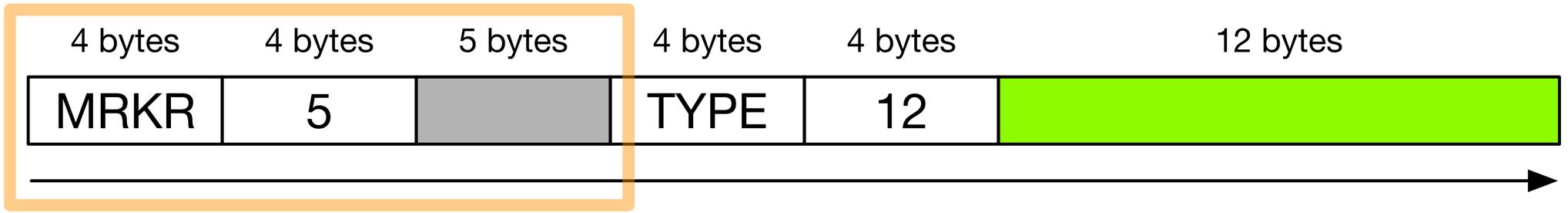
```
io_ish.size #=> Integer
io_ish.seek(absolute_positive_offset_bytes) #=> void
io_ish.pos #=> Integer
io_ish.read(positive_n_bytes_from_current_pos) #=> String<Binary> | nil
```

We permit any given parser to only use these methods, and nothing else.

Smallest module for an imaginary file format

```
# Implements support for .yolo files which always start with "yolo"
class StartsWithYolo
  def call(io_ish)
    io_ish.seek(0)
    if io_ish.read(4) == "yolo"
      return Result.new(format: "yolo", nature: :binary)
    end
  end
end
```

Imaginary TLV (Tag-Length-Value) file format



Parser for the imaginary TLV format

```
# Parse .big files which have chunks that are 4 bytes long, followed
# by the length of the chunk body
class BigBinaryParser
  def call(io_ish)
    io_ish.seek(0)
    # IO#read may return less bytes than we requested if the file ends
    # before that amount of bytes. If that happens we need to stop.
    while chunk_name_and_length = io_ish.read(8) && chunk_name_and_length == 8
      chunk_type, length = chunk_name_and_length.unpack("A4N")
      if chunk_type === "TYPE"
        tag_value = io_ish.read(length)
        return Result.new(format: "bigbinary", binary_type: tag_value)
      else
        io_ish.seek(io_ish.pos + length) # Skip to next chunk
      end
    end
  end
end
```

TLV-encoded file formats

- PNG
- AIFF
- WAV
- TIFF and derivatives (sometimes, mostly)
- MPEG4 / QuickTime / M4A

Almost TLV-encoded file formats

— JPEG

File formats from the 7th hell

— PDF

What is a useful result?

```
result = FormatParser.parse(io)
if result && result.nature == :image
  return [result.display_width_px, result.display_height_px]
end
```

```
result.sample_rate # For audio
result.entries # For ZIP archives
result.intrinsics # EXIF data, ID3 tags...
```

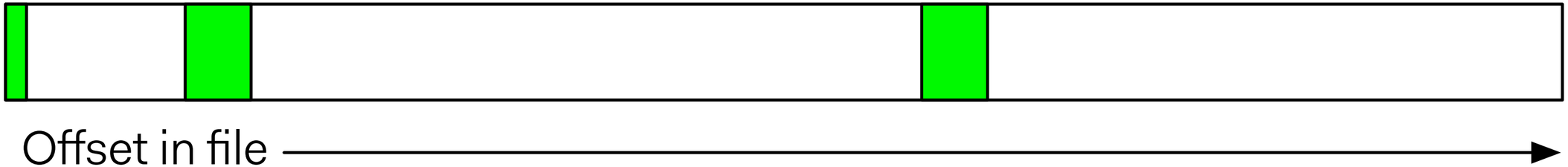
Applying parsers

```
defined_parsers.each do |parser|
  if result = parser.(file_or_remote_io_or_io_ish)
    return result
  end
end
```

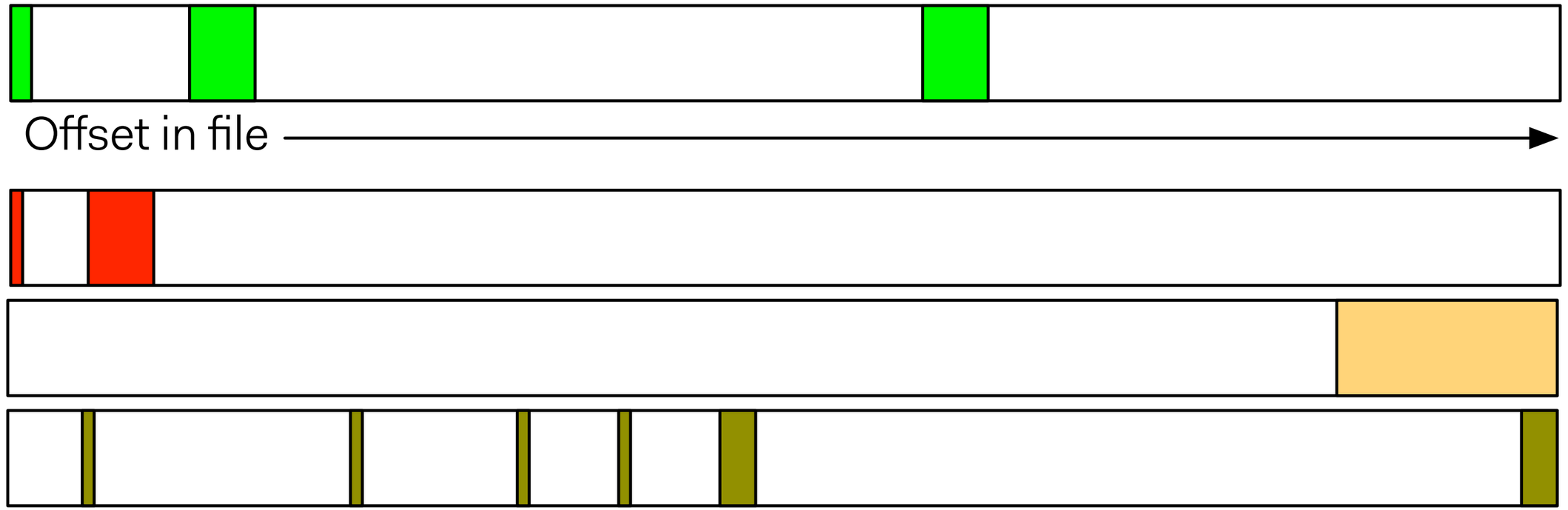
Dealing with small reads

Every parser will need just a few crucial bytes, but **where** they will be in the fille might be different. Doing an HTTP request for each `read()` would be prohibitive.

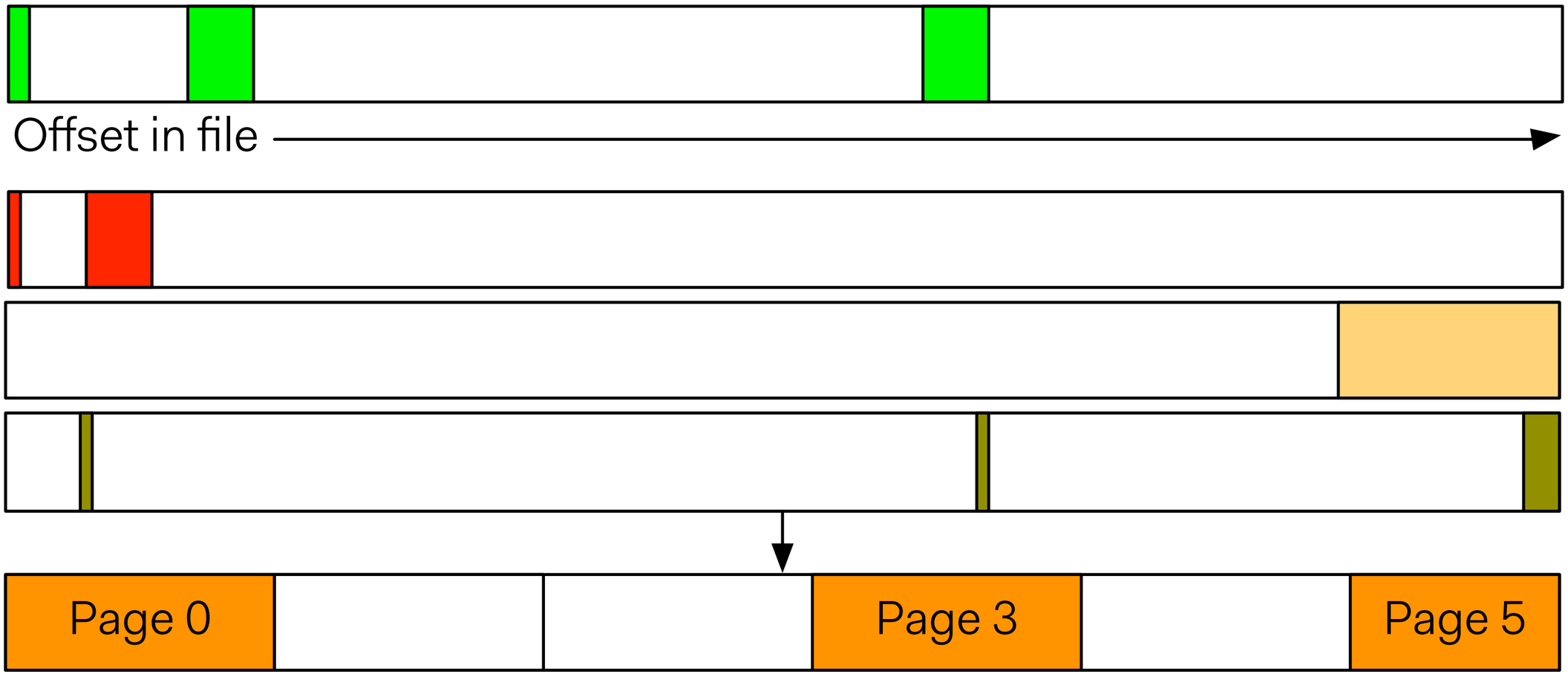
With 1 parser



With multiple parsers



With a shared page cache



In practice most parsers will at least need N bytes at the start of the file. Sometimes also N bytes at the tail of the file.

Exploits

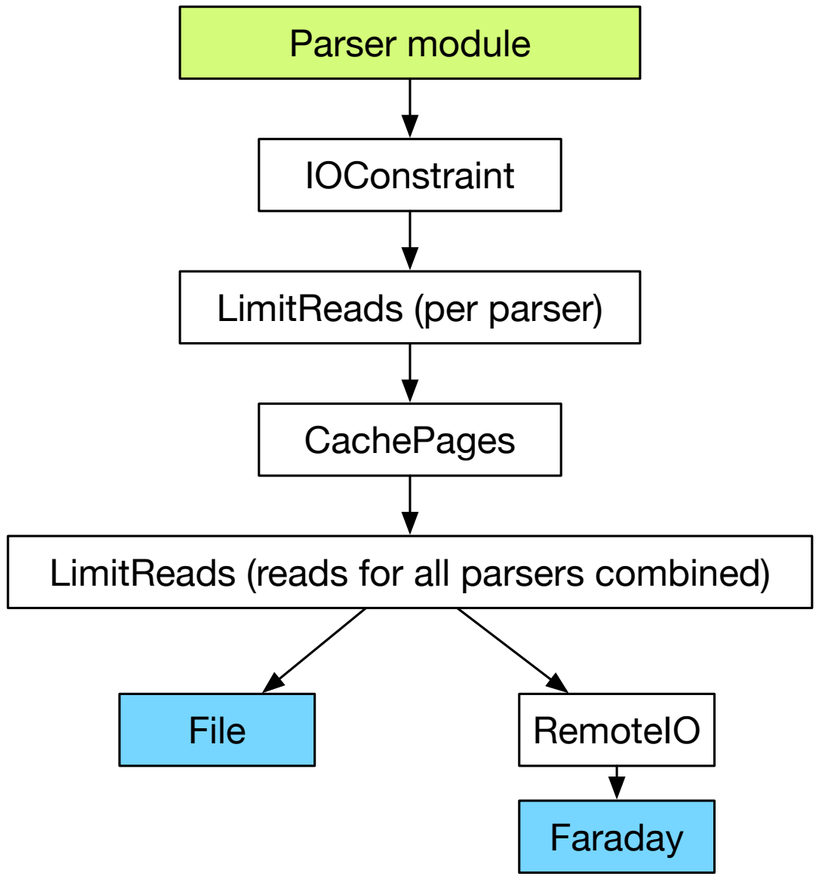
- Cause a parser to perform reads indefinitely
- Cause a parser to read too much data into memory at once
- Cause a parser to return to the same offset and read it over and over
- Cause a parser to go to a negative offset

With only 3 methods to implement, we can use proxies to track method calls.

```
def read(n)
  @reads ||= 0
  @reads += 1
  if @reads > MAXIMUM_READ_CALLS_PER_PARSER
    raise "The parser was derailed and is not allowed to proceed"
  end
  @underlying_io_ish.read(n)
end
```

```ruby
def read(n)
  @bytes_read ||= 0
  @bytes_read += n
  if @bytes_read > MAXIMUM_BYTES_PERMITTED_PER_PARSER
    raise "The parser wanted to read more data than we are comfortable with"
  end
  @underlying_io_ish.read(n)
end
```

IO stack

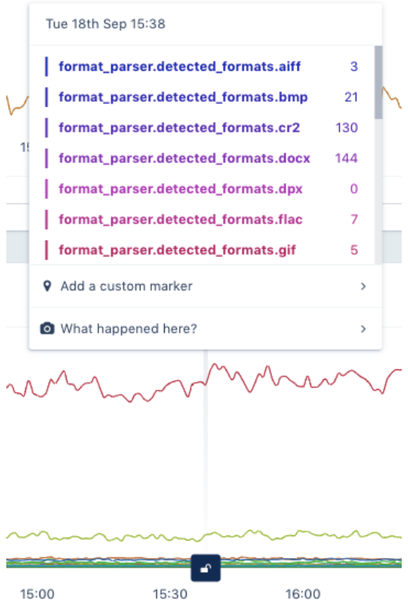

Parser module
IOConstraint
LimitReads (per parser)
CachePages
LimitReads (reads for all parsers combined)
File
RemoteIO
Faraday

Instrumentation is a given

— https://github.com/WeTransfer/measurometer

Instrumentation is a given

— https://github.com/WeTransfer/measurometer



format_parser.parser.GIFParser	732	3,86 sec	
format_parser.Care.read_page	710	3,86 sec	
format_parser.RemoteIO.read	697	3,86 sec	
http.patron	237	3,86 sec	
format_parser.parser.JPEGParser	2.456	775 µs	
format_parser.EXIFParser.exif_from_tiff_io	2.121	614 µs	

Well-factored modules: good for assesment

We can have a candidate make a module for format_parser, which shows us:

— How you collaborate on an existing codebase
— How you honor the existing style
— How your tests are
— How you communicate in a PR

1 Open ✓ 86 Closed Author Labels Projects Milestones Reviews Assignee Sort

Add :priority when registering parsers and use it when parsing 3
#132 by julik was merged on 17 Aug • Approved

Reset the PDF parser to the utmost basics ✓ 1
#129 by grdw was merged on 18 Jun

Fixes bytes_used_by_frames variable calculation ✓ 1
#127 by krists was merged on 18 Jun

Remove uneeded guard when ID3Tag tries to find an unknown genre ✓ 4
#125 by alex-quiterio was merged on 18 Jun

Finish work on Ogg support ✗ 1
#124 by julik was merged on 14 Jun

Improve performance of the PDF parser ✓ 31
#123 by grdw was closed on 18 Jun • Changes requested

Improve performance of the PDF parser ✓ 1
#122 by grdw was closed on 8 Jun

JPEG: Only accept the SOI marker at start of file ✓
#117 by julik was merged on 10 May • Approved

😍 Used libraries

— https://github.com/remvee/exifr - EXIF for TIFF, CR2, JPEG

— https://github.com/krists/id3tag - ID3 tags in MP3

Summary

- Detecting files reliably is hard, but not impossible
- `format_parser` is on par with FastImage in terms of speed, and ordrers of magintude faster when file information is at the tail of the file
- We think format_parser turned out good and you should use it in your projects

And yes…

It's a TIFF!

Thank you!

@julikt
https://github.com/WeTransfer/format_parser