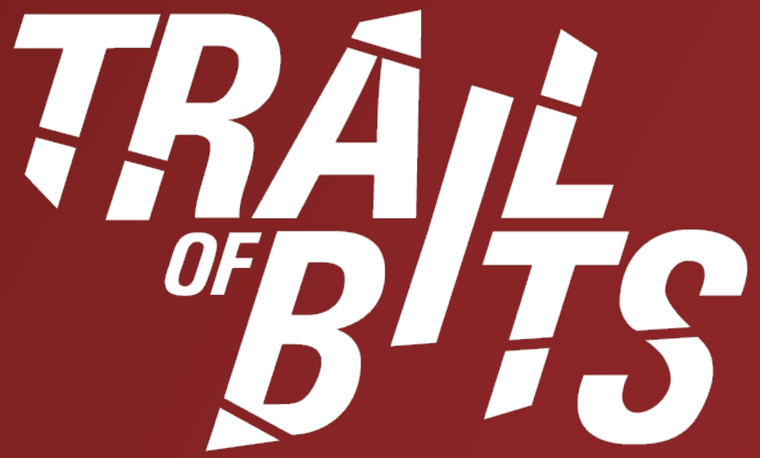


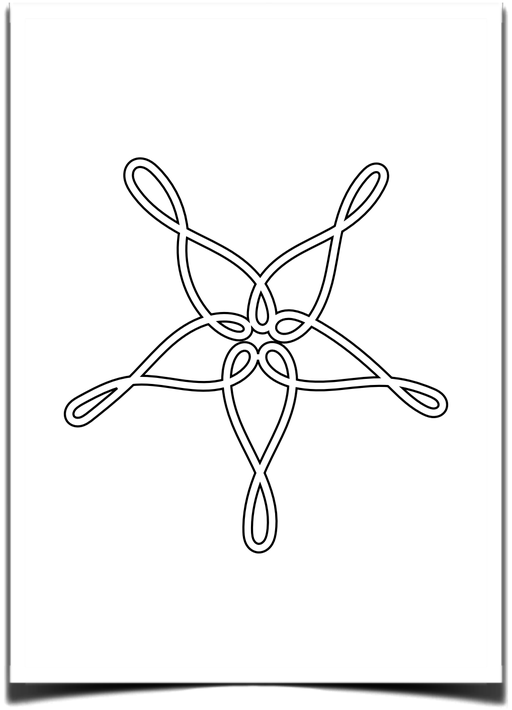
Toward Automated Grammar Extraction

via Semantic Labeling of Parser Implementations

Carson Harmon Bradford Larsen Evan Sultanik

LangSec Workshop at IEEE Security & Privacy, May 21, 2020

The Problem

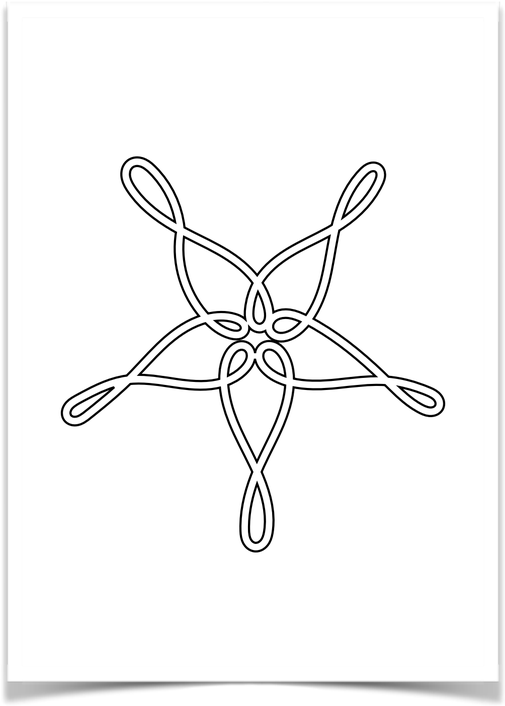


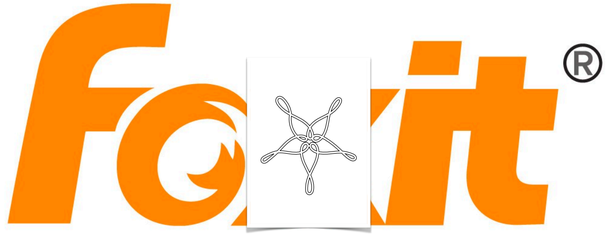
Foxit®

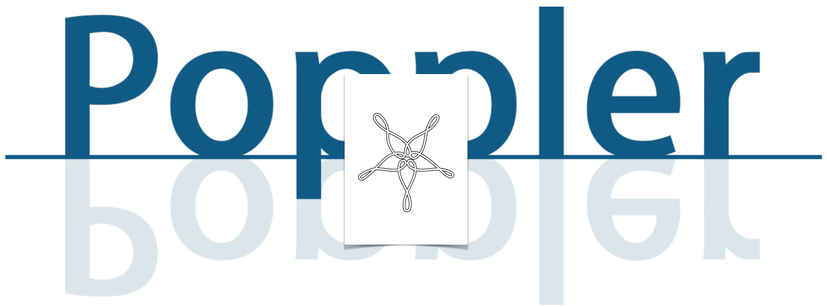
Poppler

MµPDF

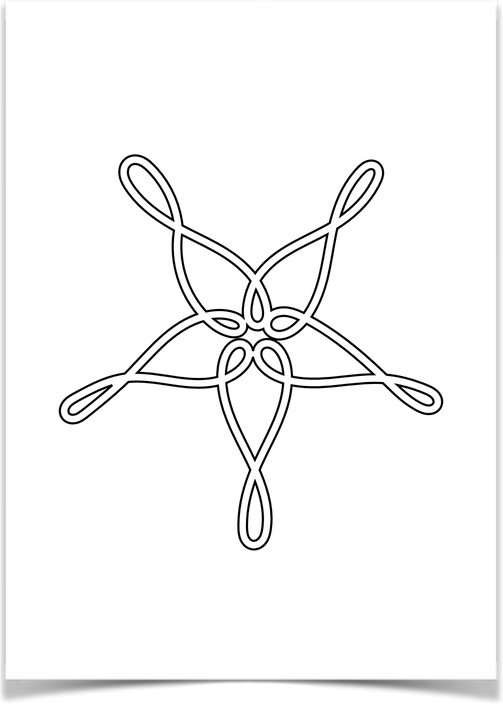
The Problem







The Problem



The Problem

Foxit®

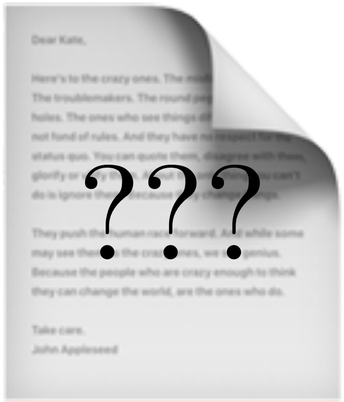
Poppler

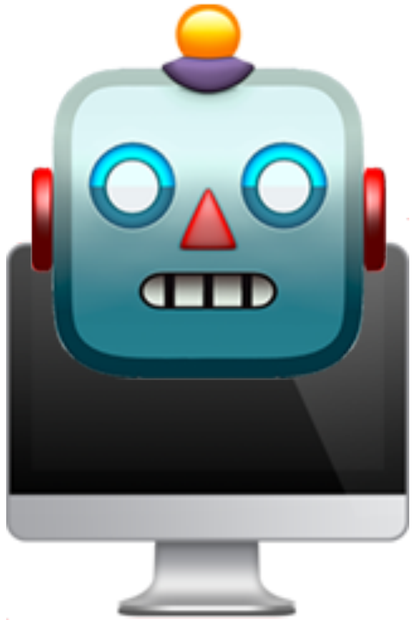
MµPDF

The Problem

High Level Goals

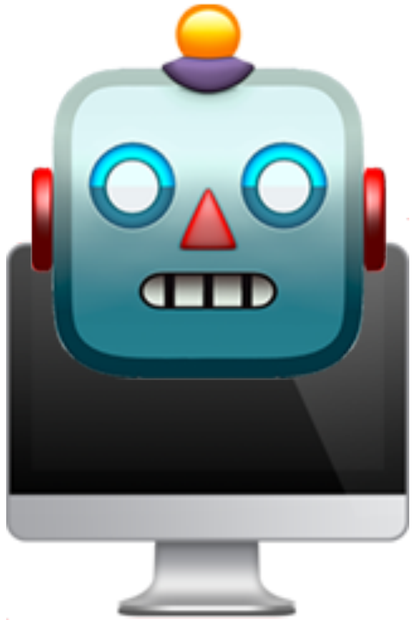
Create semantic map of the functions in a parser, which will improve grammar extraction.





High Level Goals

Create semantic map of the functions in a parser, which will improve grammar extraction.



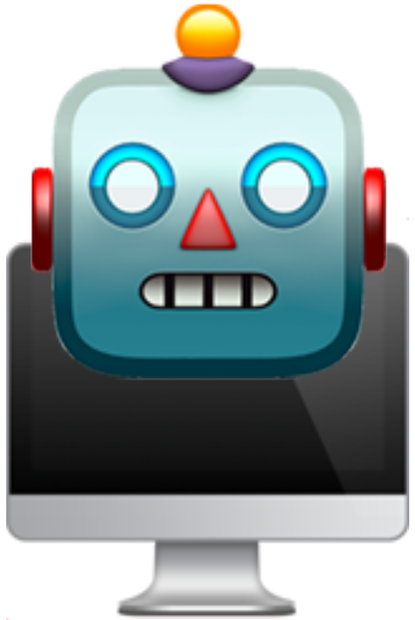
```
parser_function1

↳byte 0, 10, 50

  Object Stream


parser_function2

↳byte 10, 74

  Xref


parser_function3

↳byte 20

  JFIF
```

High Level Goals

Create semantic map of the functions in a parser, which will improve grammar extraction.

Ultimate Goal: Automatically extract a minimal grammar specifying the files accepted by a parser

Hypothesis: The majority of the potential for maliciousness and schizophrenia will exist in the symmetric difference of the grammars accepted by a format's parser implementations



```
parser_function1
↳byte 0, 10, 50
  Object Stream

parser_function2
↳byte 10, 74
  Xref

parser_function3
↳byte 20
  JFIF
```

Approach

Semantic Ground Truth

Label the *Type Composition Hierarchy* of the input files

Instrumentation

Use *universal taint analysis* to track *all* input bytes through the execution of a parser

Associative Labeling

Merge the results of the first two steps to produce a labeling of which functions operate on which types

Detect backtracking

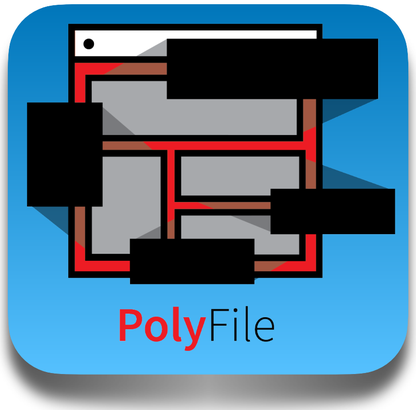
Detect error handling

Differential analysis

Approach

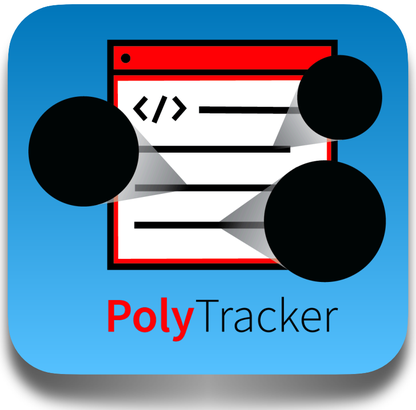
Semantic Ground Truth

Label the *Type Composition Hierarchy* of the input files



Instrumentation

Use *universal taint analysis* to track *all* input bytes through the execution of a parser



Associative Labeling

Merge the results of the first two steps to produce a labeling of which functions operate on which types

Detect backtracking

Detect error handling

Differential analysis

Approach

Sema

Label

Hiera

Grammar Extraction

(future work)

Instrumentation

universal taint analysis

all input bytes

ough the execution of a

parser

Associative Labeling

Merge the results of the first two steps to produce a labeling of which functions operate on which types

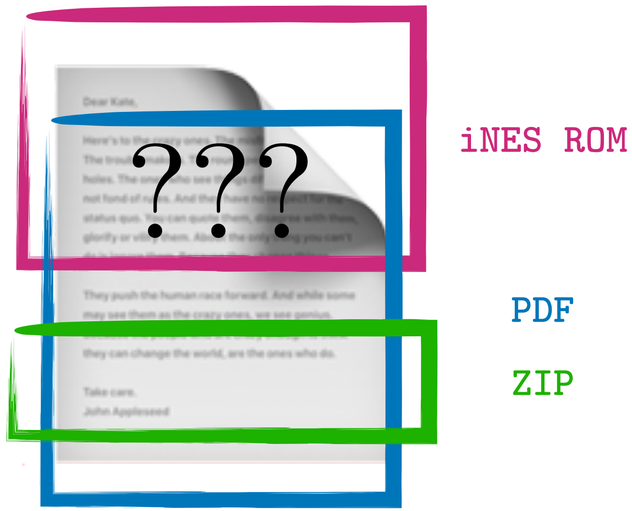
Detect backtracking

Detect error handling

Differential analysis

Prior Work: Semantic Labeling

Polyglot-Aware File Identification



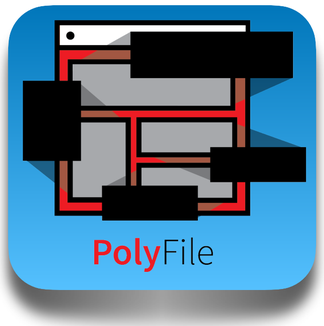
Resilient Parsing

Modify parsers for best effort

Instrument to track input byte offsets

Label regions of the input

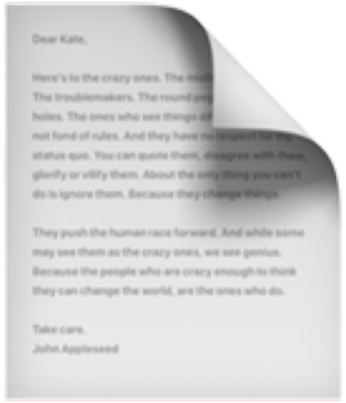
Produce ground truth



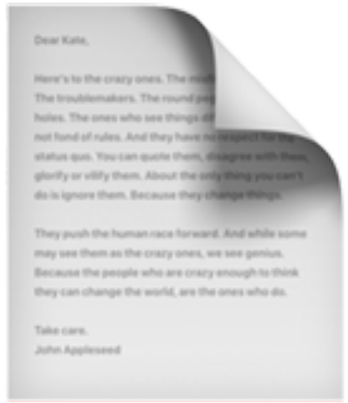
Syntax Tree

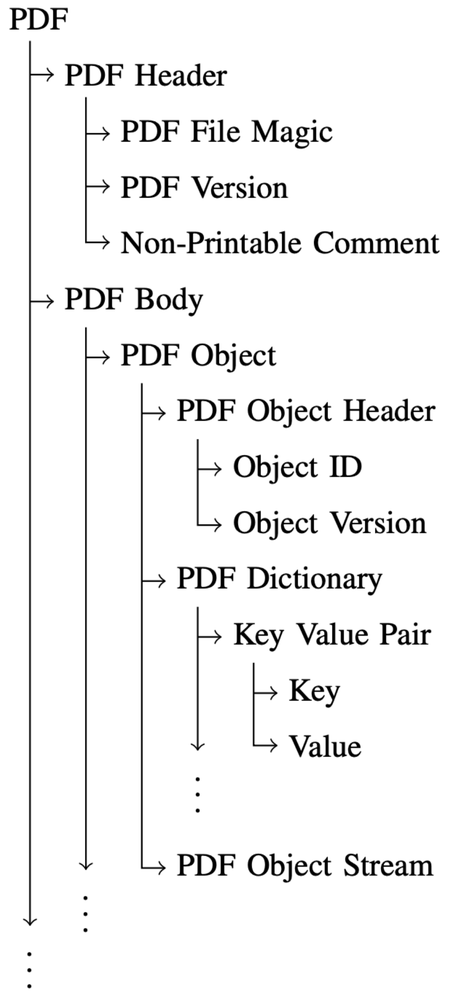
```
iNES [0x0→0x12220]
↳ Magic [0x0→0x3]
   Header [0x4→0xF]
   ⋮
   PRG [0xC210→0x1020F]
   CHR [0x10210→0x12220]
PDF [0x10→0x2EF72F]
↳ Magic [0x10→0x1E]
   Object 1.0 [0x1F→0x12221]
   ↳ Dictionary [0x2A→0x3E]
      Stream [0x3F→0x12219]
      ↳ JFIF Image [0x46→0x1220F]
         ↳ JPEG Segment […]
            ↳ Magic […]
               Marker […]
⋮
```

PolyFile Ground Truth



PolyFile Ground Truth





Prior Work: Parser Instrumentation

LLVM

Operate on LLVM/IR

Can work with all open source parsers

Eventually support closed-source binaries by lifting to LLVM (*e.g.*, with McSEMA or Remill)

Instrumentation

Shadow memory inspired by the Data Flow Sanitizer (dfsan)

Negligible CPU overhead

$O(n)$ memory overhead, where n is the number of instructions executed by the parser

Taint Tracking

Novel datastructure for efficiently storing taint labels

dfsan status quo:

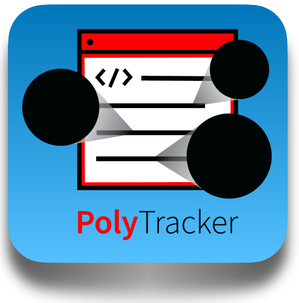
$\boldsymbol{\Theta}(1)$ lookups

$\boldsymbol{\Theta}(n^2)$ storage

PolyTracker:

$O(\log\ n)$ lookups

$O(n)$ storage



PolyTracker Instrumentation


```
{
  "ensure_solid_xref": [
    2276587,
    2276588
  ],
  "fmt_obj": [
    2465223,
    2465224,
    2465225,
    2465226,
    2465227,
    2465228,
    2465240,
    2465241,
    2465242,
    2465243,
    2465244,
    2465245,
    2465246,
    2465258,
    2465259,
    2465260,
    2465261,
    2465262
  ]
}
```

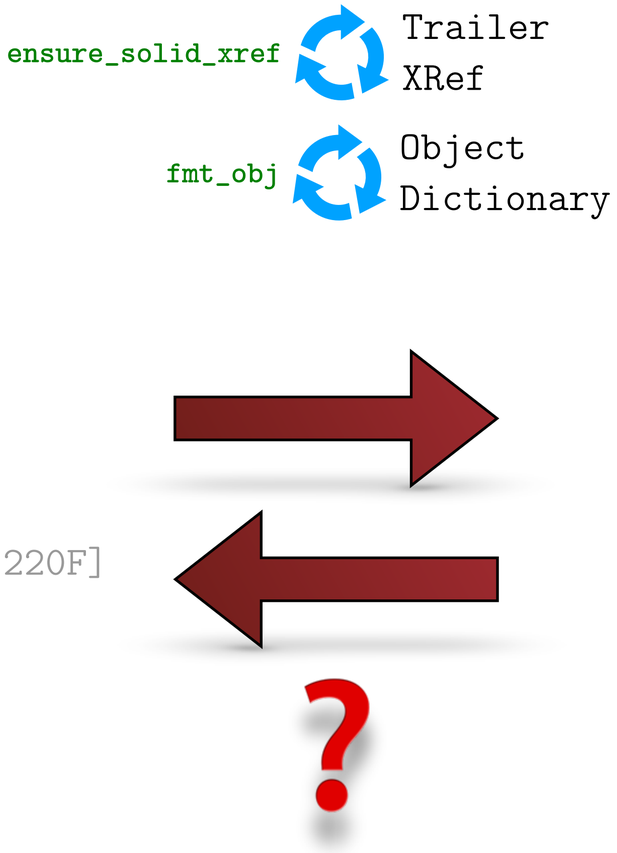
PolyTracker Instrumentation

```
iNES [0x0→0x12220]
↳ Magic [0x0→0x3]
   Header [0x4→0xF]
   ⋮
   PRG [0xC210→0x1020F]
   CHR [0x10210→0x12220]
PDF [0x10→0x2EF72F]
↳ Magic [0x10→0x1E]
   Object 1.0 [0x1F→0x12221]
   ↳ Dictionary [0x2A→0x3E]
      Stream [0x3F→0x12219]
      ↳ JFIF Image [0x46→0x1220F]
         ↳ JPEG Segment […]
            ↳ Magic […]
               Marker […]
⋮
```


```
{
  "ensure_solid_xref": [
    2276587,
    2276588
  ],
  "fmt_obj": [
    2465223,
    2465224,
    2465225,
    2465226,
    2465227,
    2465228,
    2465240,
    2465241,
    2465242,
    2465243,
    2465244,
    2465245,
    2465246,
    2465258,
    2465259,
    2465260,
    2465261,
    2465262
  ]
}
```

PolyTracker Instrumentation

```
iNES [0x0→0x12220]
↳ Magic [0x0→0x3]
   Header [0x4→0xF]
   ⋮
   PRG [0xC210→0x1020F]
   CHR [0x10210→0x12220]
PDF [0x10→0x2EF72F]
↳ Magic [0x10→0x1E]
   Object 1.0 [0x1F→0x12221]
   ↳ Dictionary [0x2A→0x3E]
      Stream [0x3F→0x12219]
      ↳ JFIF Image [0x46→0x1220F]
         ↳ JPEG Segment […]
            ↳ Magic […]
               Marker […]
⋮
```



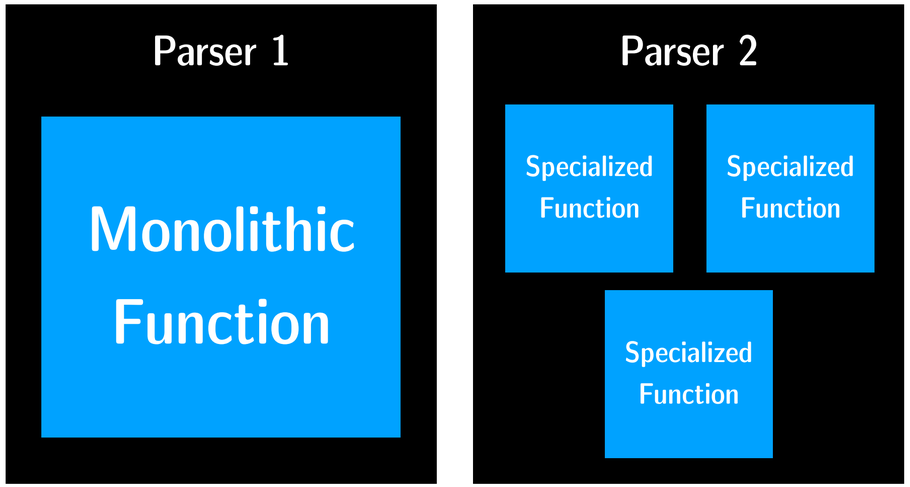

```
{
  "ensure_solid_xref": [
    2276587,
    2276588
  ],
  "fmt_obj": [
    2465223,
    2465224,
    2465225,
    2465226,
    2465227,
    2465228,
    2465240,
    2465241,
    2465242,
    2465243,
    2465244,
    2465245,
    2465246,
    2465258,
    2465259,
    2465260,
    2465261,
    2465262
  ]
}
```

The Challenge of Associative Labeling

How can we associate types in the file format
to the set of functions most specialized in operating on that type?

Observations

Raw mapping is not necessarily injective:



A parser's functional implementation will rarely be isomorphic to the type hierarchy or syntax tree of the input file

There will rarely be a perfect bijection between the types and functions

Information Entropy

Idea: Use information entropy to measure function specialization

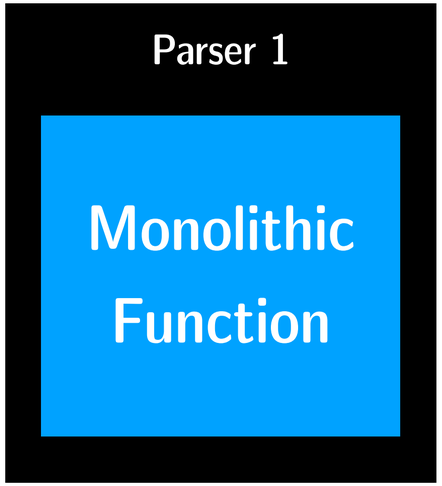
- For each type, collect the functions that operate on that type
- Calculate $P(t,\ f)$ = the probability that a specific type occurs within a function
- Calculate the "genericism" of a function $G : F \to \mathbb{R}$

$$G(f \in F) \mapsto -\sum_{t \in T} P(t, f) \log_2 P(t, f).$$

- Use G to sort the functions associated with a type, discarding all but the smallest (most specialized) standard deviation

Problem: Code is Too Monolithic

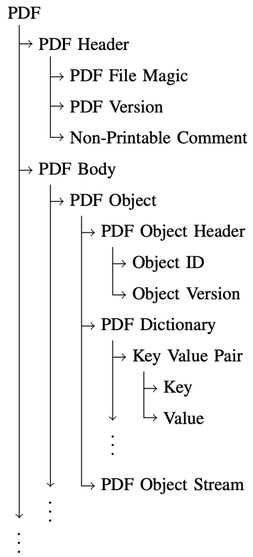
The parser has a single function responsible for parsing multiple types

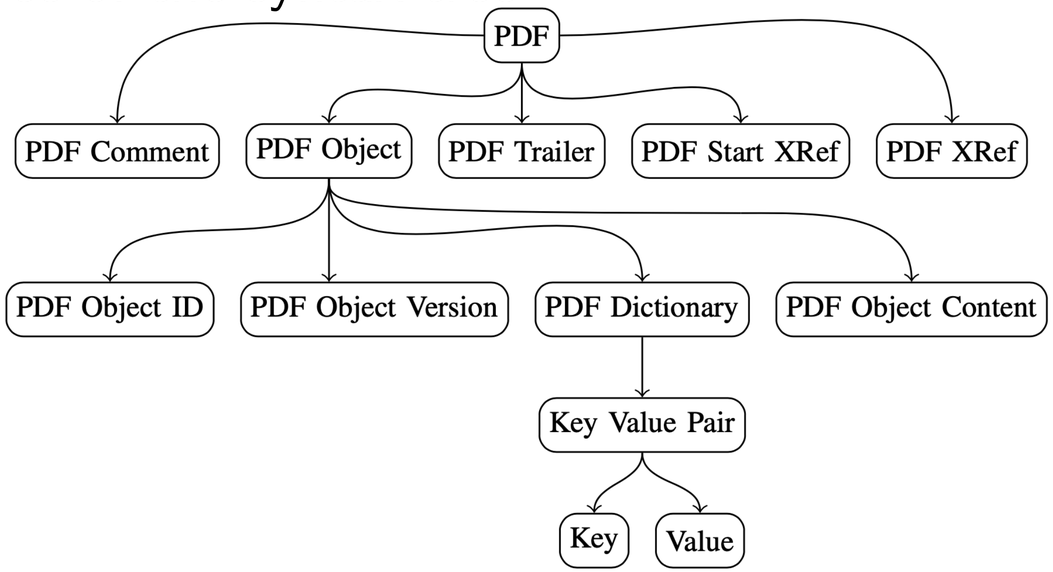


Problem: Code is Too Monolithic

The parser has a single function responsible for parsing multiple types

- Calculate the *dominator tree* of the syntax tree

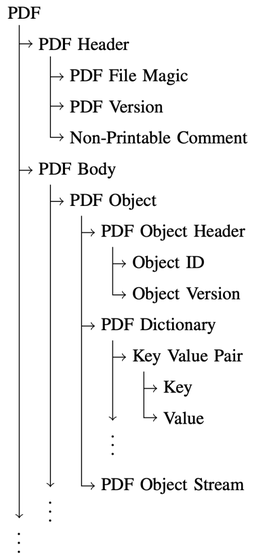


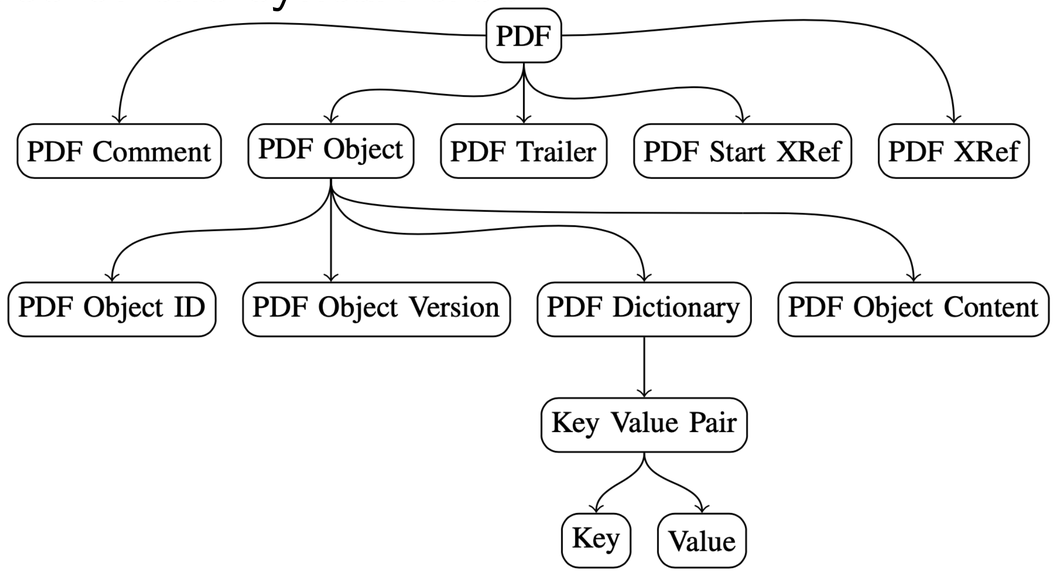


Problem: Code is Too Monolithic

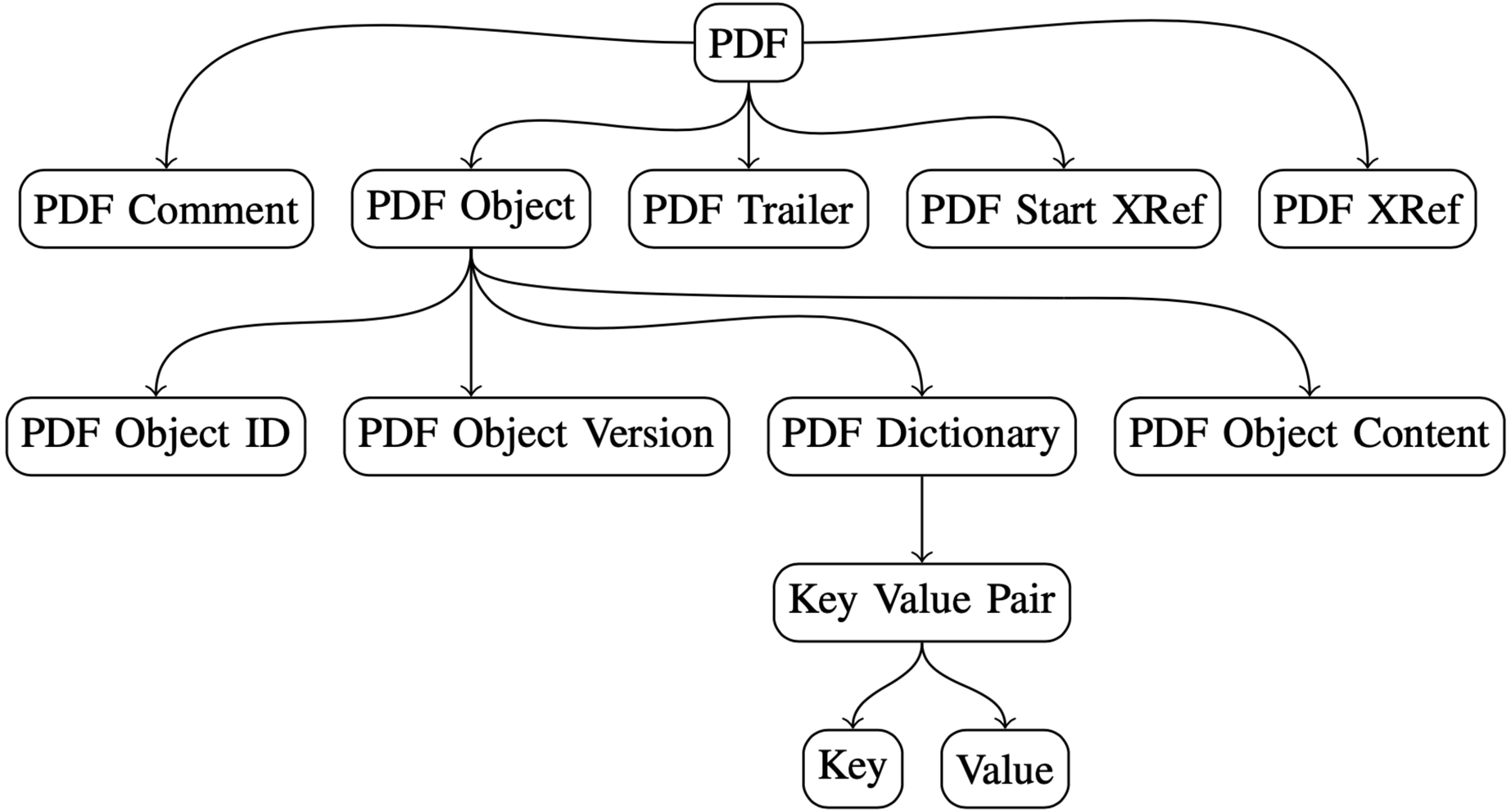
The parser has a single function responsible for parsing multiple types

- Calculate the *dominator tree* of the syntax tree

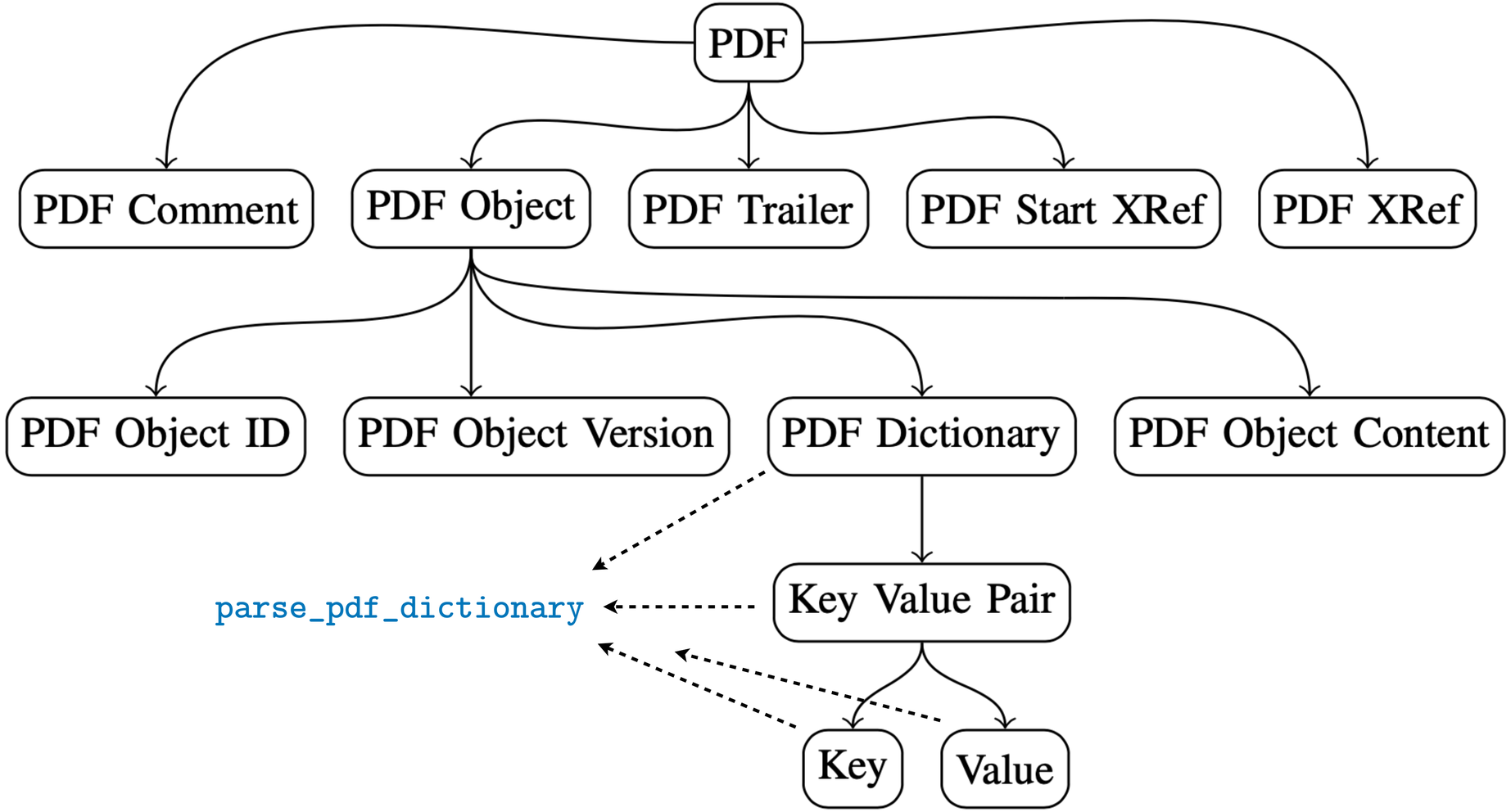




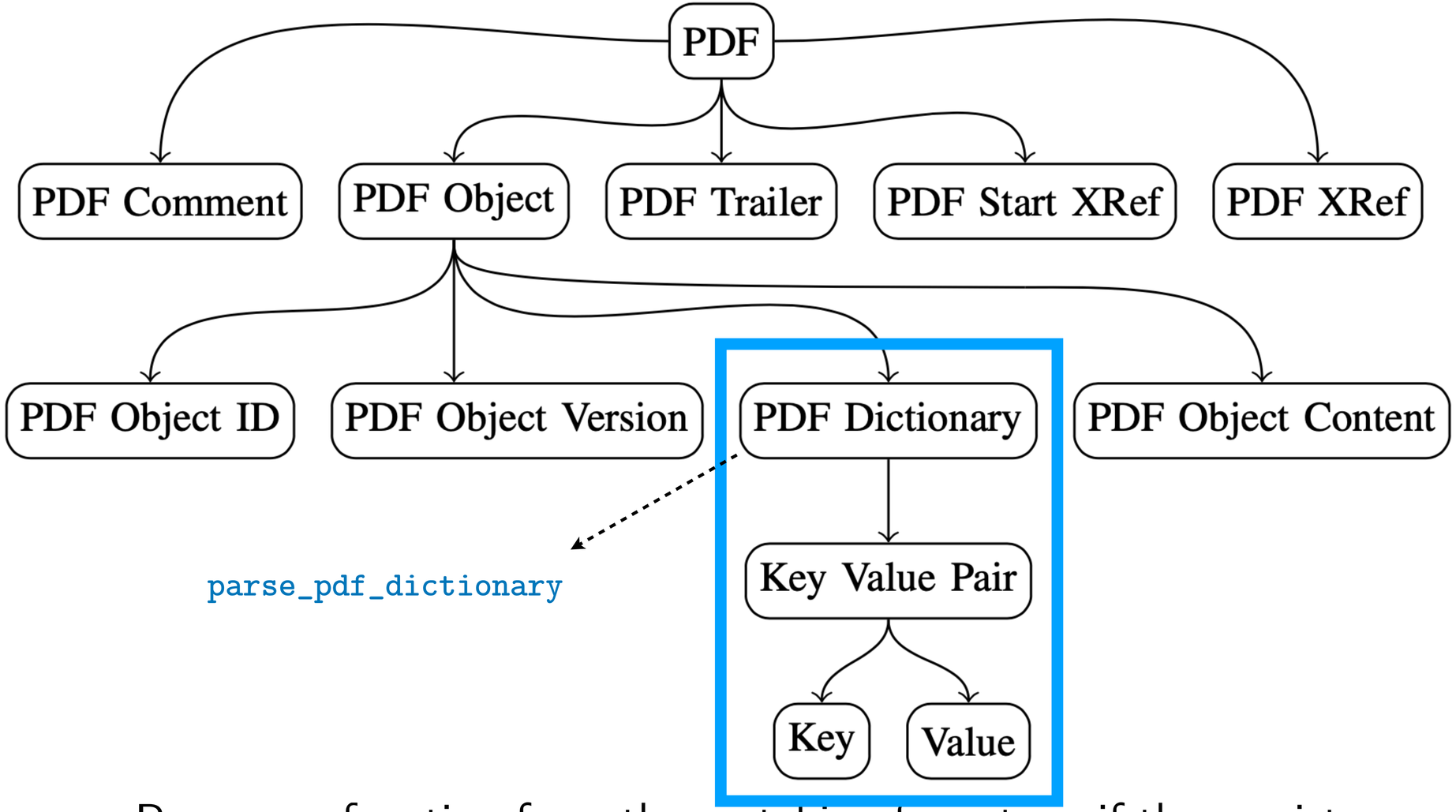
- Remove a function from the matching for a type if there exists an ancestor of the type in the dominator tree that maps to the same function



- Remove a function from the matching for a type if there exists an ancestor of the type in the dominator tree that maps to the same function



- Remove a function from the matching for a type if there exists an ancestor of the type in the dominator tree that maps to the same function

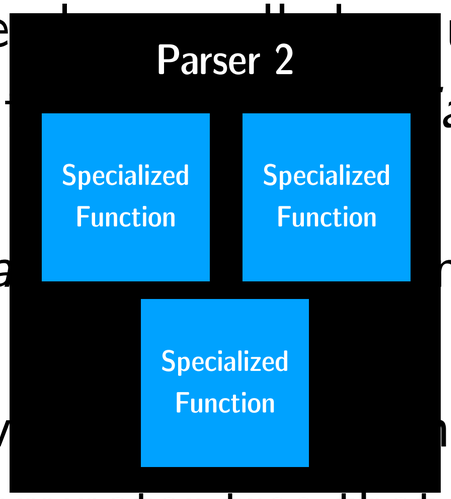


- Remove a function from the matching for a type if there exists an ancestor of the type in the dominator tree that maps to the same function

Problem: Code is Too Cohesive

The parser has many, tightly coupled functions collectively responsible for parsing a single type

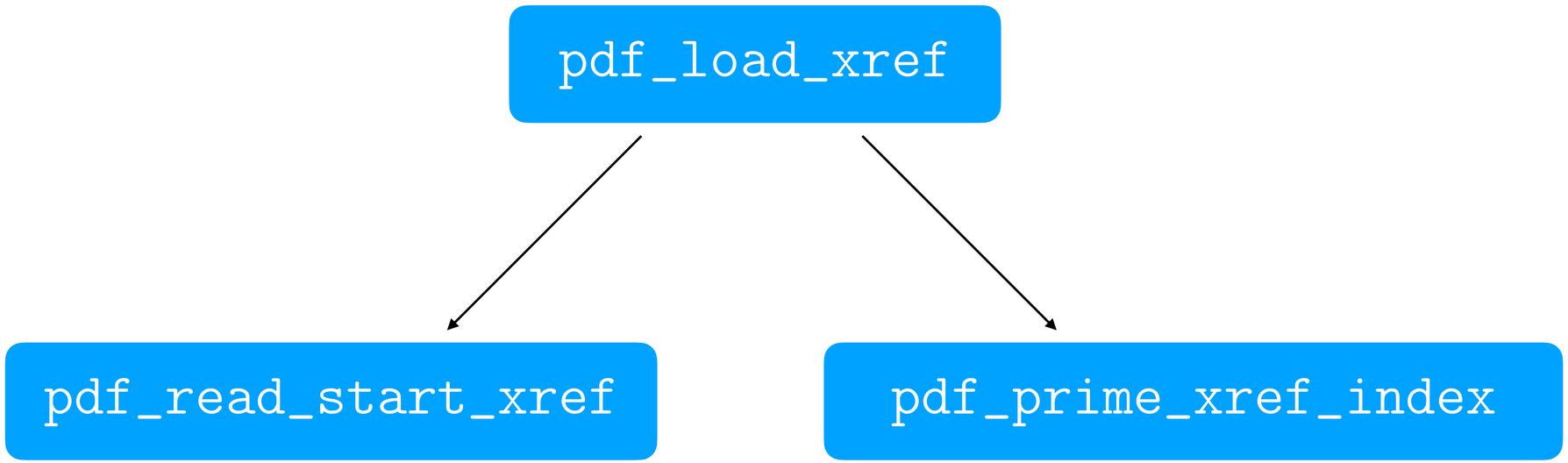
- If those functions are [illegible] uentially, then we ideally only want the single [illegible] *ates* the sequence
- Calculate the *domina* [illegible] ntime control flow graph
- For each type, remov [illegible] the matching that have an ancestor in the dominator tree that is *also* in the matching

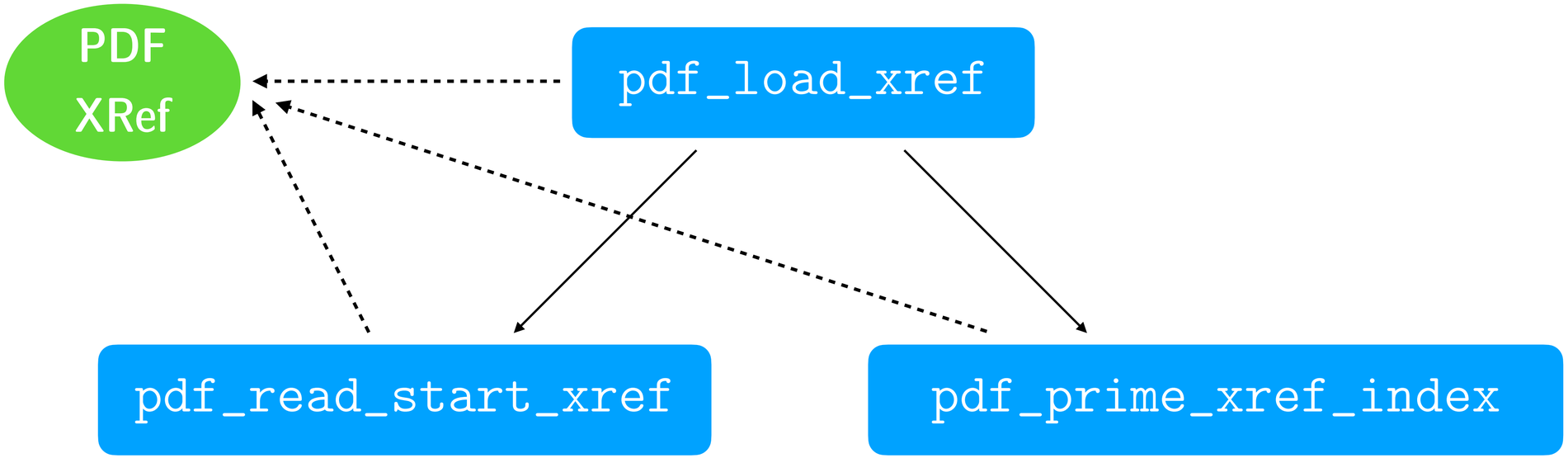


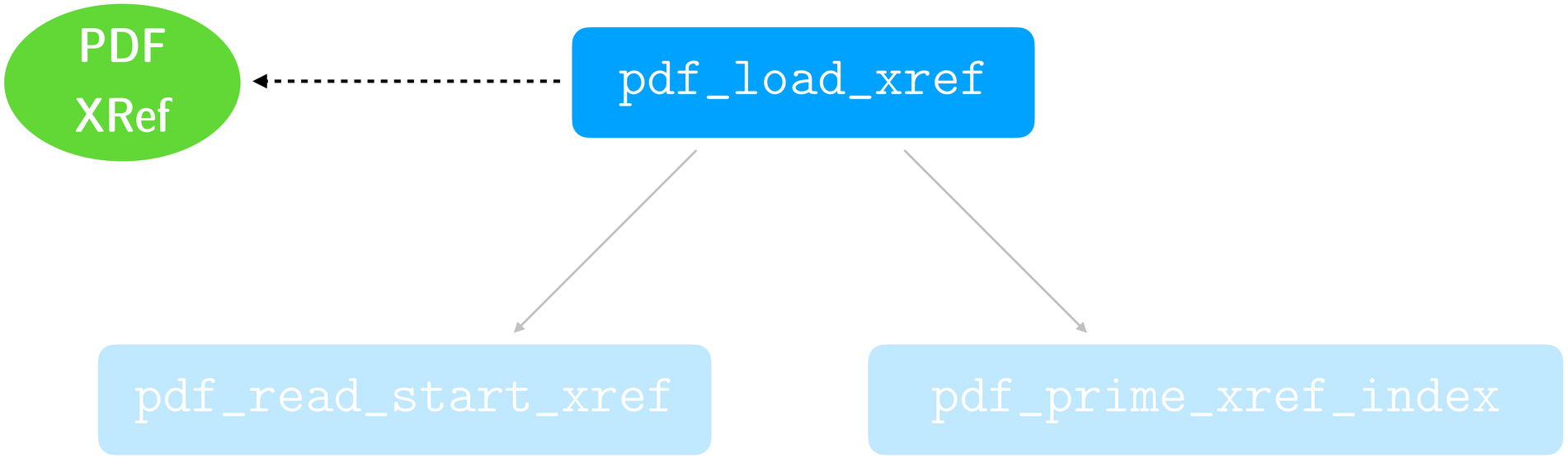
Problem: Code is Too Cohesive

The parser has many, tightly coupled functions collectively responsible for parsing a single type

- If those functions are always called sequentially, then we ideally only want the single function that *initiates* the sequence
- Calculate the *dominator tree* of the runtime control flow graph
- For each type, remove any functions in the matching that have an ancestor in the dominator tree that is *also* in the matching


pdf_load_xref
pdf_read_start_xref
pdf_prime_xref_index

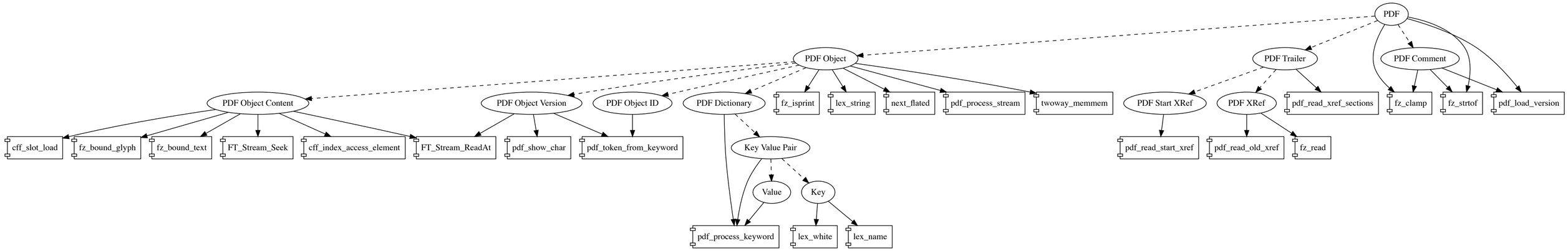

PDF
XRef
pdf_load_xref
pdf_read_start_xref
pdf_prime_xref_index


PDF
XRef
pdf_load_xref
pdf_read_start_xref
pdf_prime_xref_index

Results

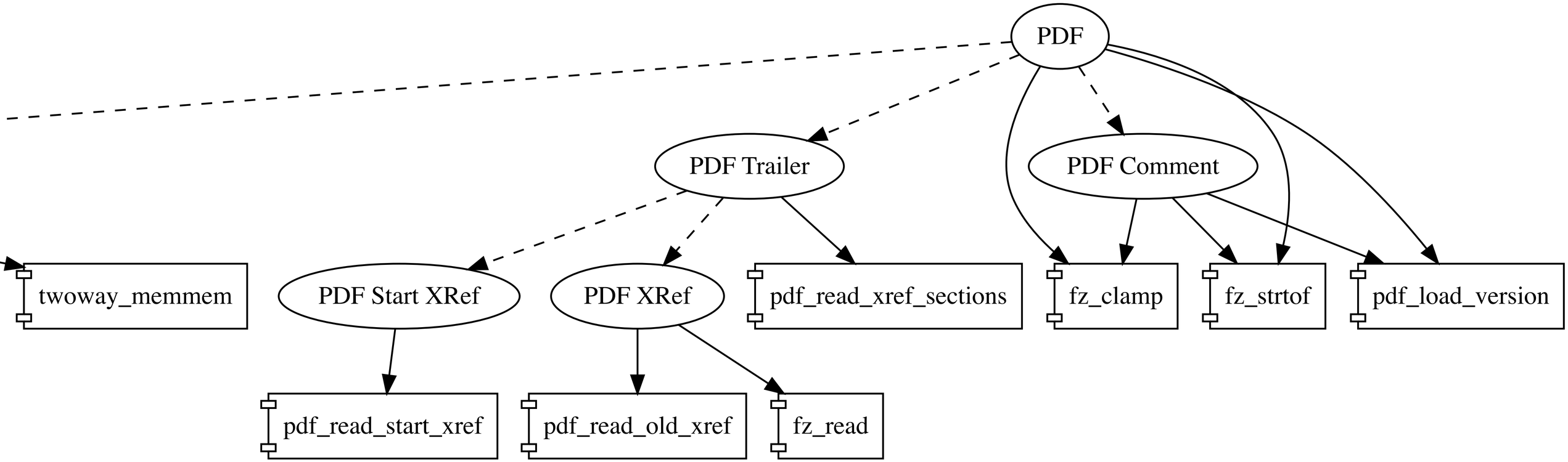
- Runs in $O(|F|n \log |T|)$ time
  - $F =$ # functions in the parser
  - $T =$ # types (or production rules) in the grammar
  - $n =$ # bytes in the input file
- Mappings for various parsers and file formats
- Implementation in the `polymerge` application distributed with PolyFile:
  - `pip3 install polyfile`

Results: MuPDF



(Generated from a single parse of a single PDF)

Results: MuPDF



(Generated from a single parse of a single PDF)

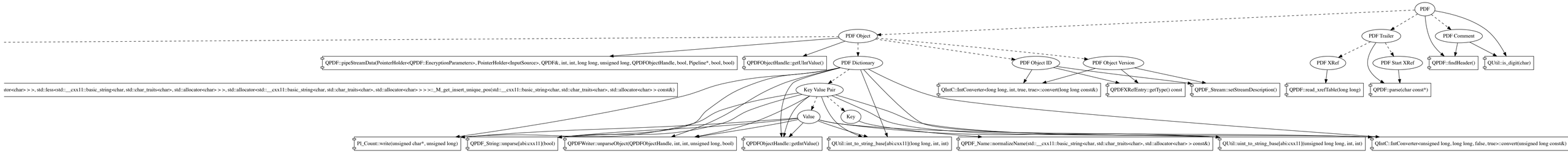
Results: QPDF

(Generated from a single parse of a single PDF)

Results: QPDF

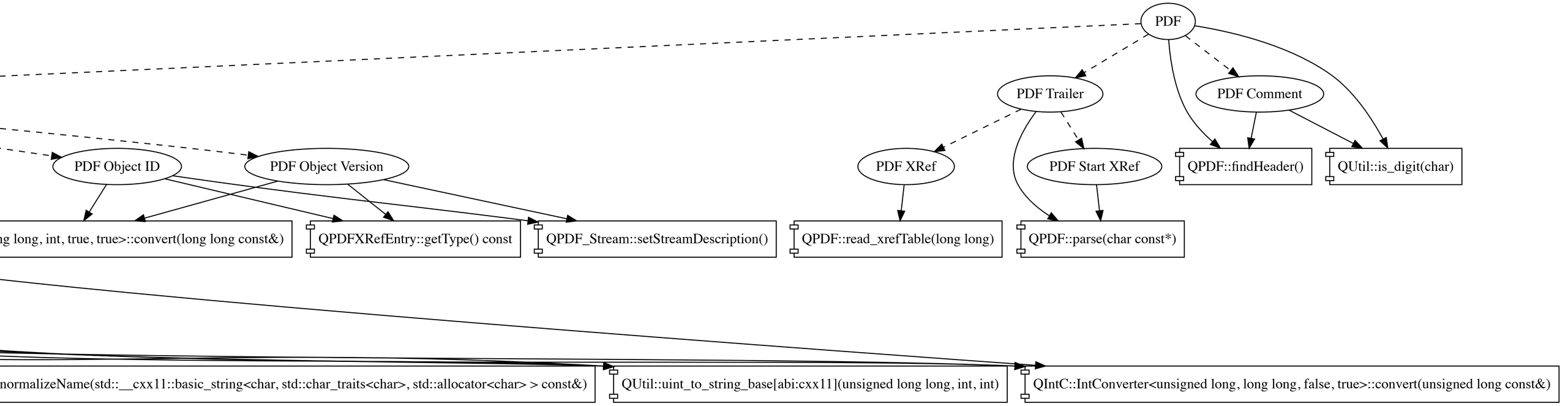
(Generated from a single parse of a single PDF)

Results: QPDF



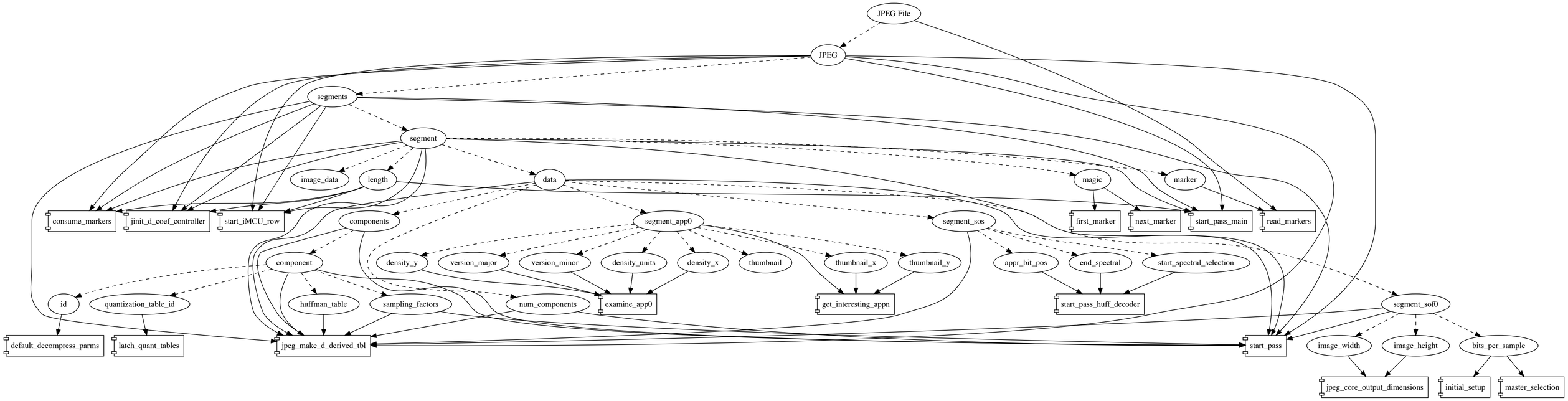
(Generated from a single parse of a single PDF)

Results: QPDF



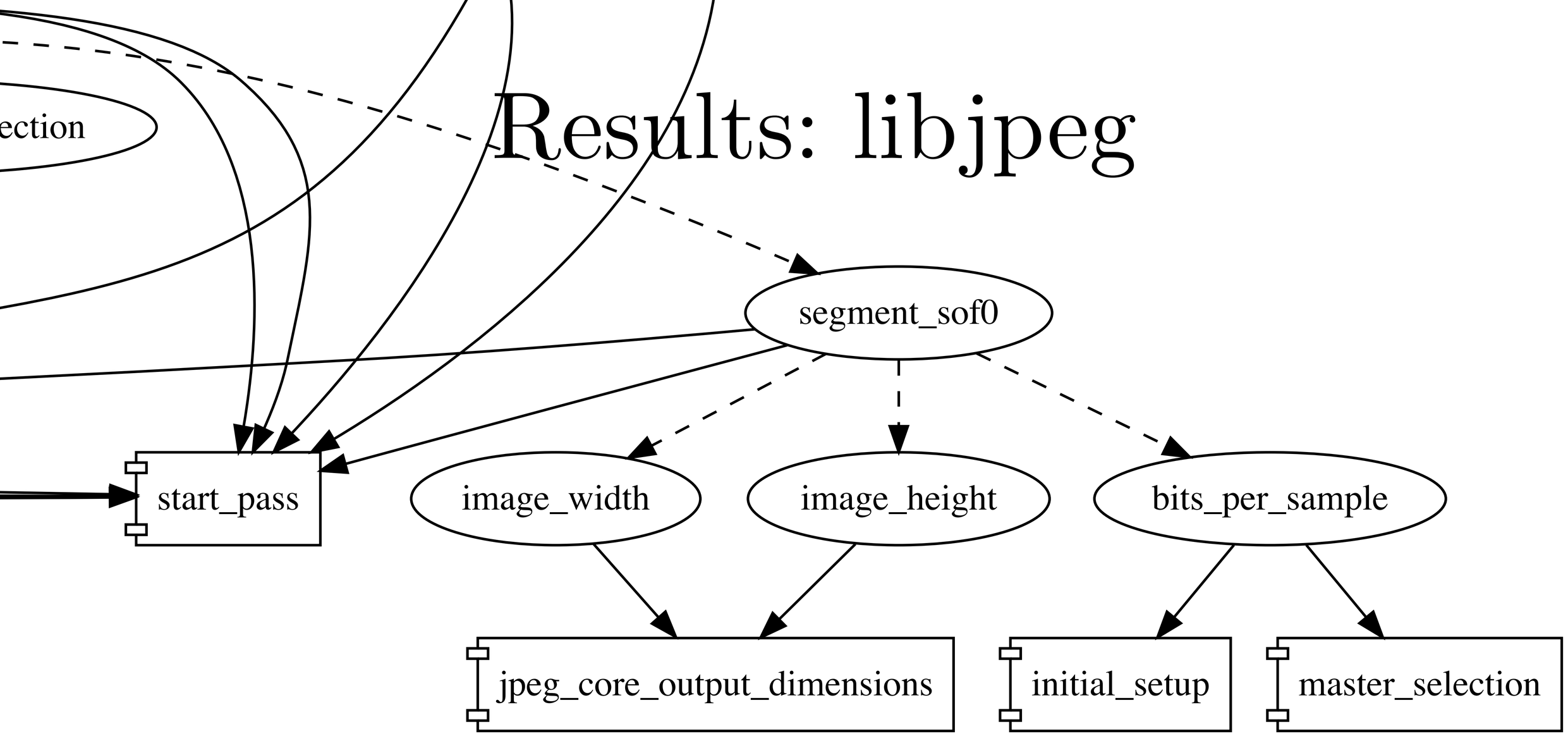
(Generated from a single parse of a single PDF)

Results: libjpeg



(Generated from a single parse of a single JPEG)

Results: libjpeg



(Generated from a single parse of a single JPEG)

Next Step: Grammar Extraction

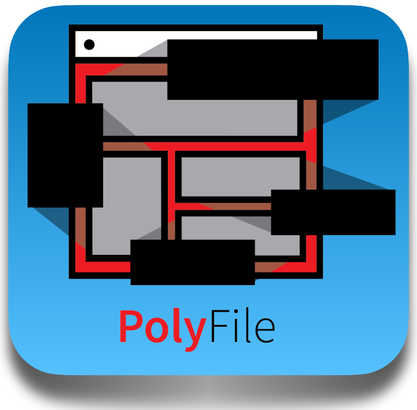
- **AUTOGRAM:** (Zeller, *et al.*, 2016) Uses data-flow analysis
  - No type information other than what can be inferred from native types in the code
  - Can be improved with our type mapping from the associative labeling
- **Mimid:** (Zeller, *et al.*, 2019) Uses static control-flow analysis
  - Can also be improved by our type mapping
  - Needs to infer indirect control-flow that we can definitively observe with our runtime instrumentation
- We can observe control-flow events like backtracking *and* infer types at the same time

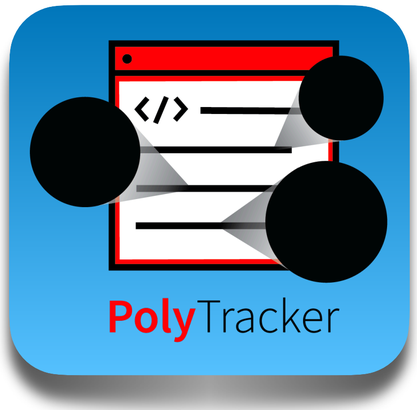
Future Directions

- **Differential Analysis of Parsers**
  - Use graph matching to map the functions of one parser to another
  - Automatically identify feature differences
- **Differential Analysis over a Corpus of Files**
  - Not all files exercise will exercise all functionality of a parser
  - Combine the output of multiple files (including intentionally malformed files) to maximize coverage
- **Type Hierarchy Learning**
  - If there is no ground truth, *learn* the type hierarchy from the data structures of the parser

Conclusions

- Introduced new technique for semantically labeling types operated on by parsers
- Works with a single run of a parser on a single file
- Next step: integrate with grammar extraction
- Tools are currently available:
  - `https://github.com/trailofbits/polyfile`
  - `https://github.com/trailofbits/polytracker`





Thanks!

Contact Info

Carson Harmon	`@reyeetengineer`
Brad Larsen	`@BradLarsen`
Evan Sultanik	`@ESultanik`

https://github.com/trailofbits/polyfile

https://github.com/trailofbits/polytracker