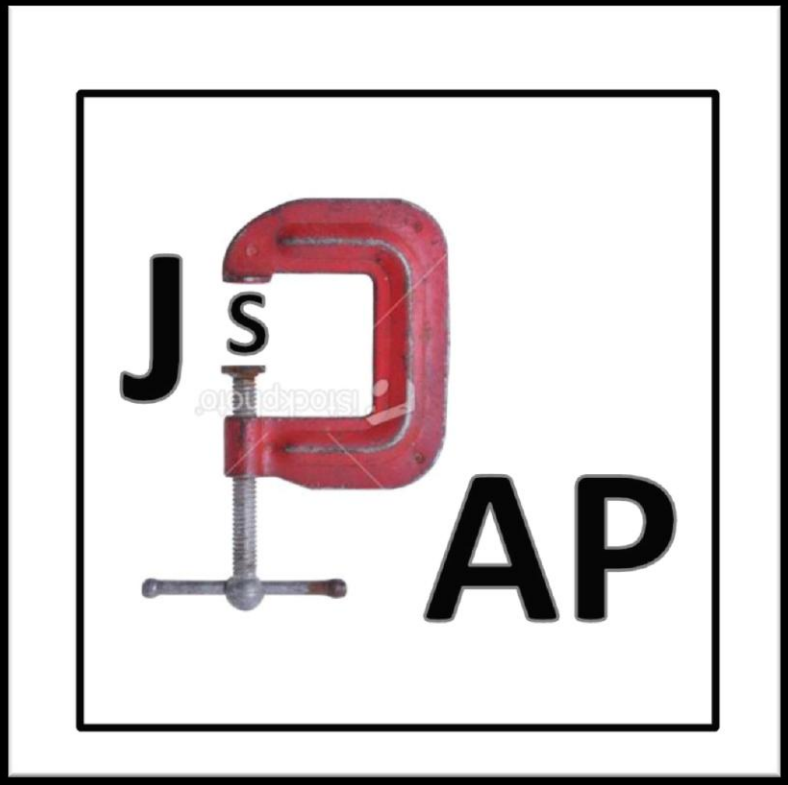


JSZap: Compressing JavaScript Code

Martin Burtscher, UT Austin

Ben Livshits & Ben Zorn, Microsoft Research

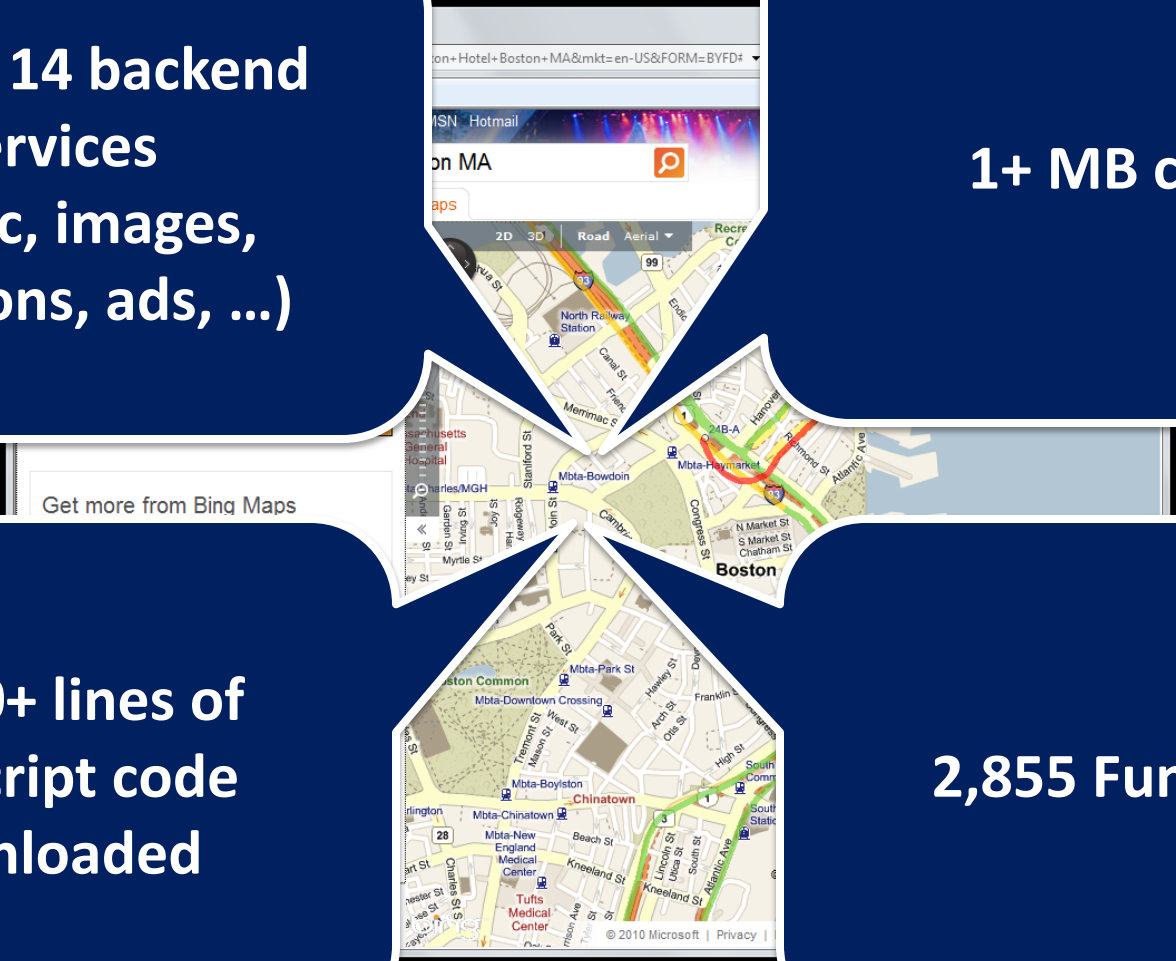
Gaurav Sinha, IIT Kanpur

RiSE Research in Software Engineering

A Web 2.0 Application Dissected

Talks to 14 backend services (traffic, images, directions, ads, …)

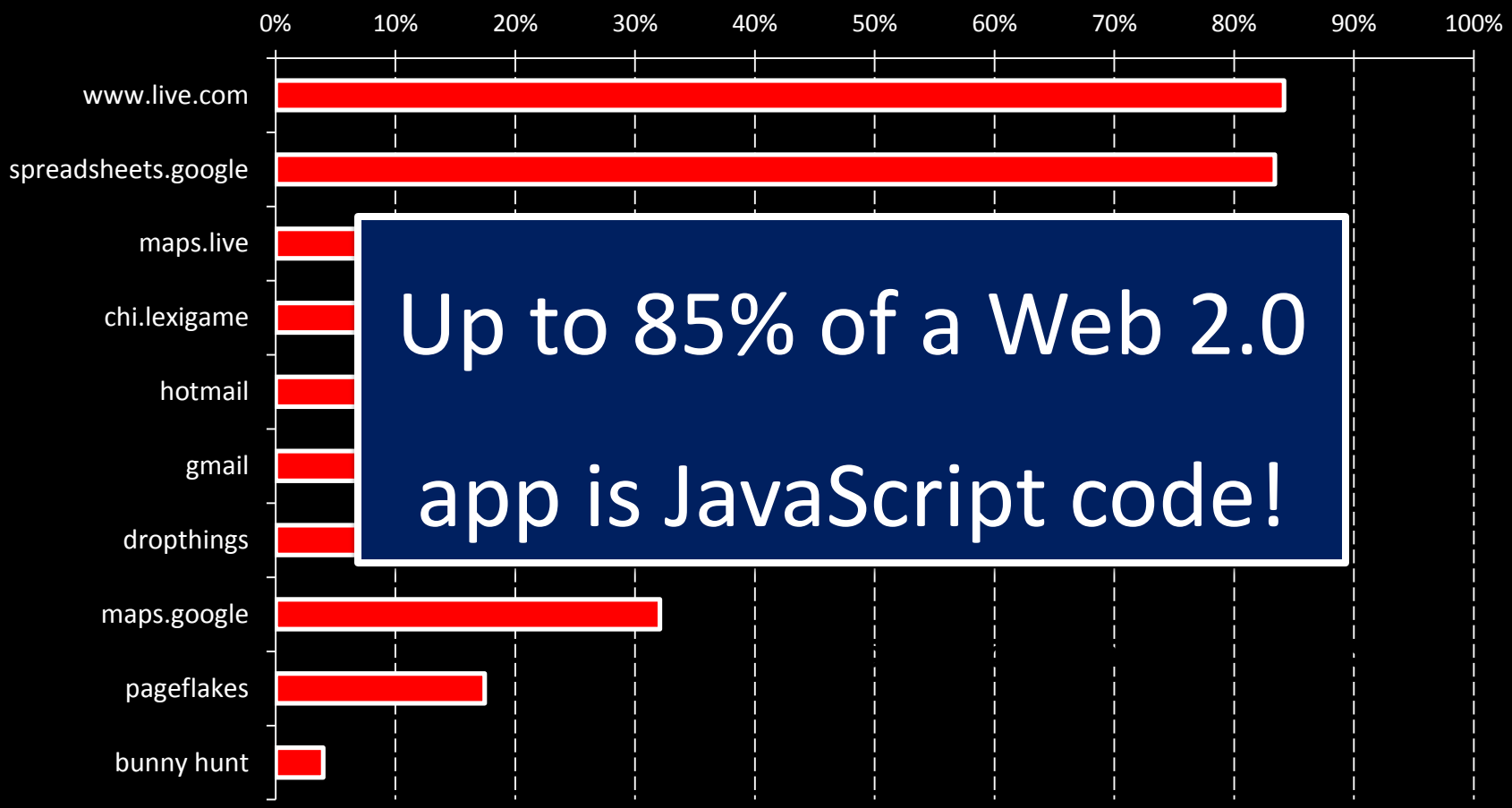
1+ MB code



70,000+ lines of JavaScript code downloaded

2,855 Functions

Lots of JavaScript being Transmitted

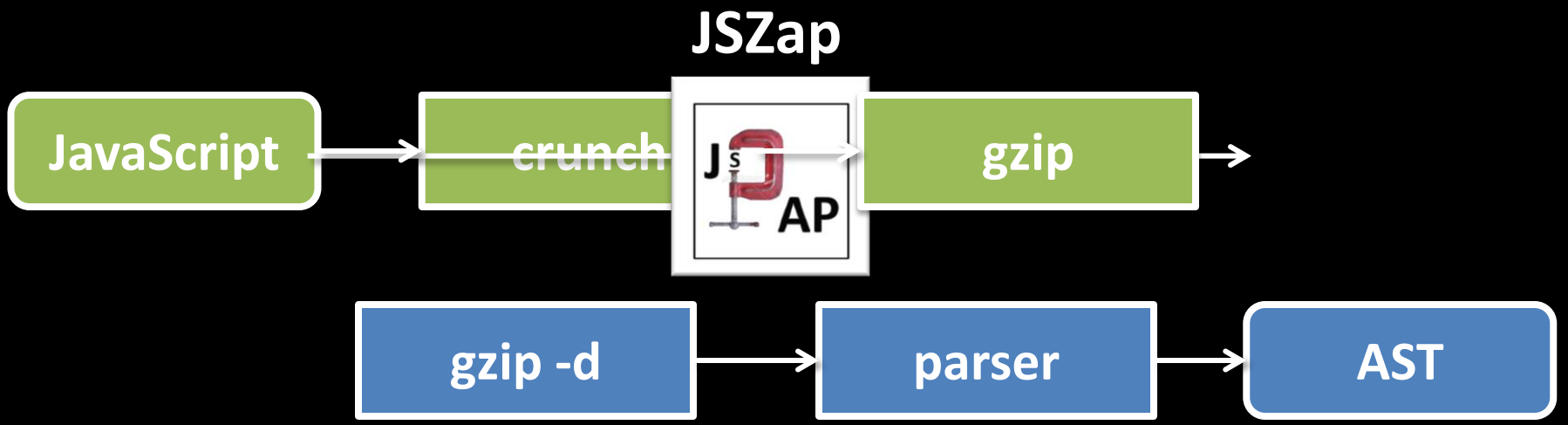


AJAX: Tension Headaches

Move code to client for responsiveness

Execution can't start without the code

JavaScript on the Wire



JSZap Approach

- Represent JavaScript as AST instead of source
- Serialize the compressed AST
- Decompress directly into AST on client
- Use gzip as 2nd-level (de-)compressor

Benefits of AST-based Compression

Reduced Latency

- Compression: less to transmit
- ASTs are blasted directly into the browser

Reduced Network Bandwidth

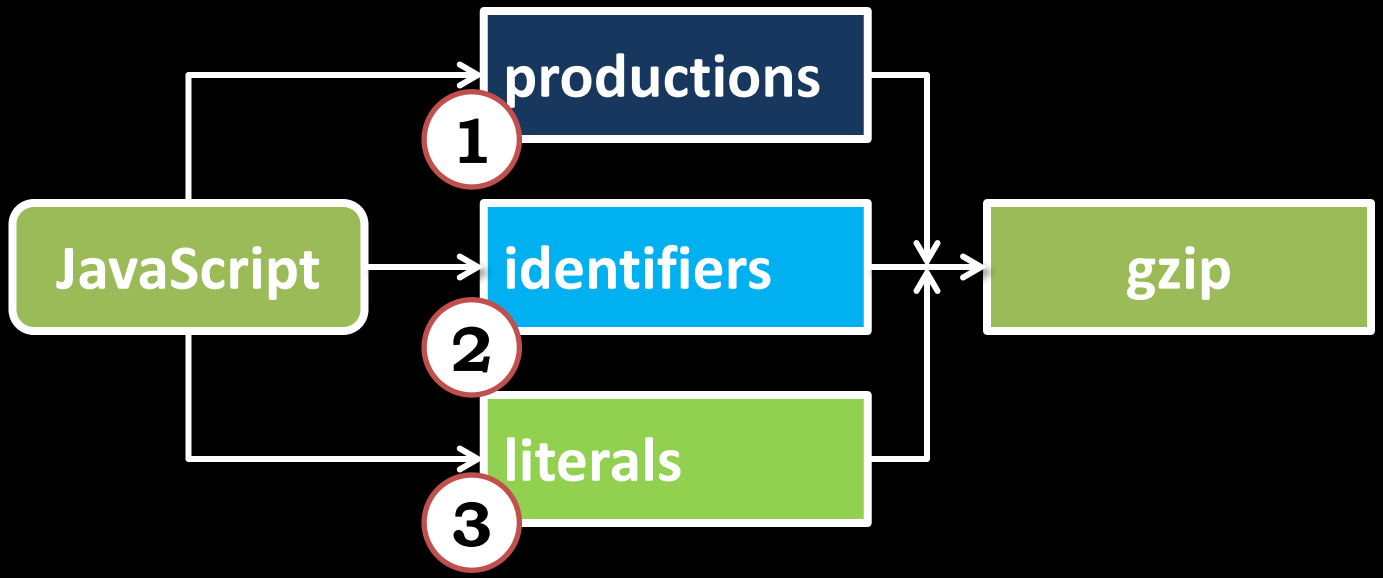
- Reduces mobile charges
- Reduces operator network costs: better for servers

Correctness, Security, and other Benefits

- Ensures well-formedness of code
- Can use to check language subsets easily (AdSafe)
- Caching incremental updates
- Unblocking HTML parser

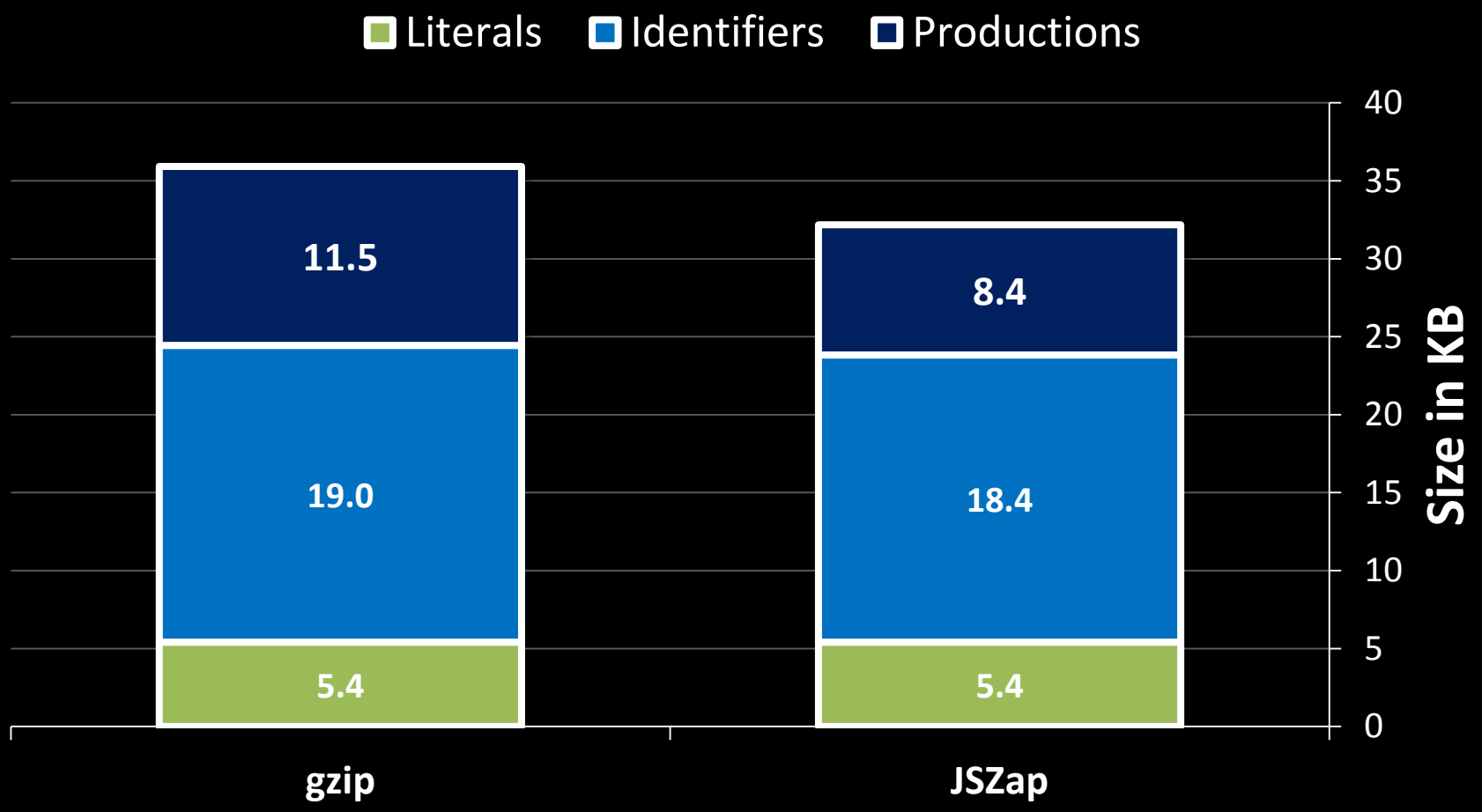
JSZap Compression

JSZap Compression

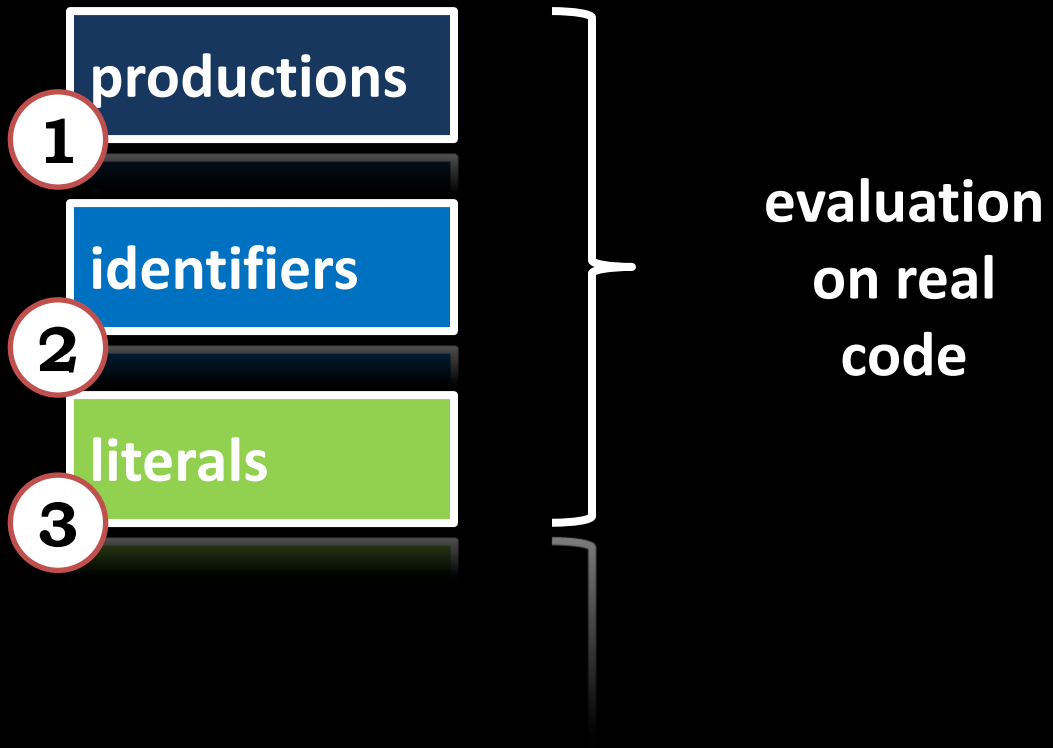



GZIP is a formidable opponent
THE COLBERT REPORT

JSZap vs. GZIP



Talk Outline



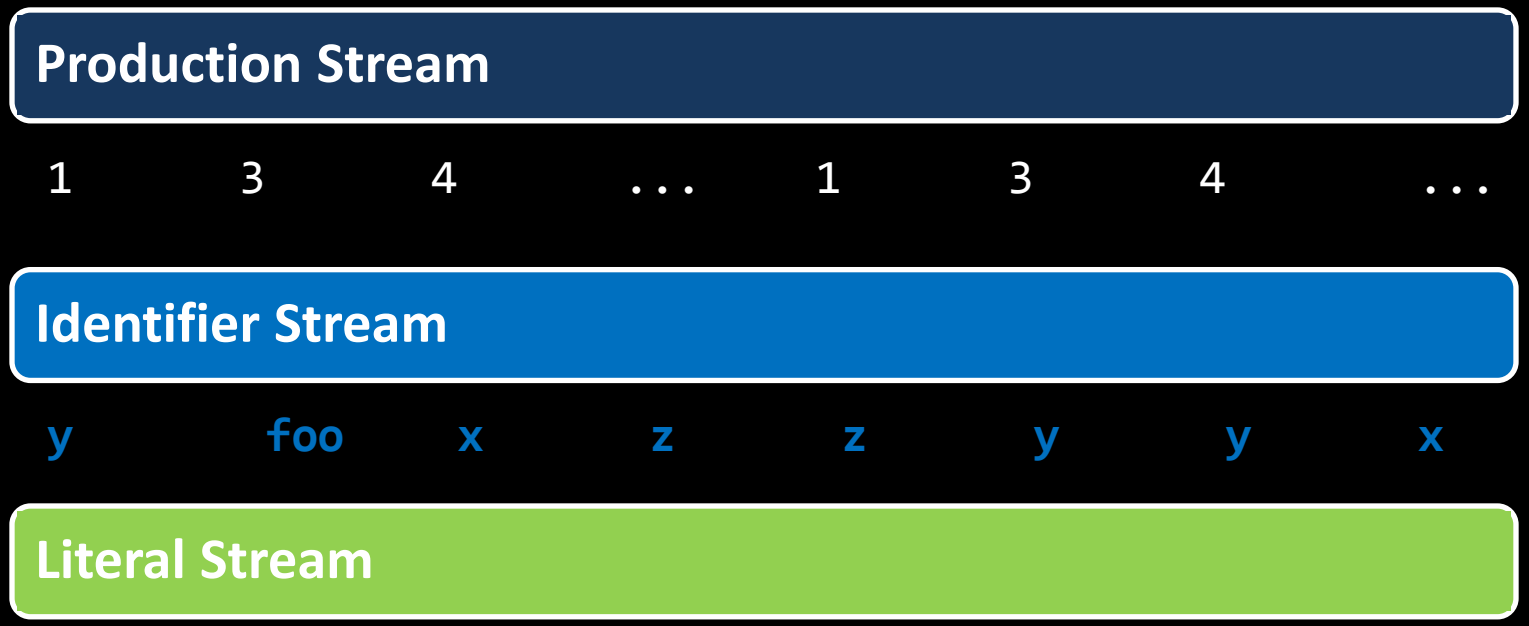
Background: ASTs

Expression

a * b + c

A Simple Javascript Example

```
var y = 2;
function foo () {
        var  x = "jscrunch";
        var  z = 3;
        z = y + y;
}
x = "jszap";
```

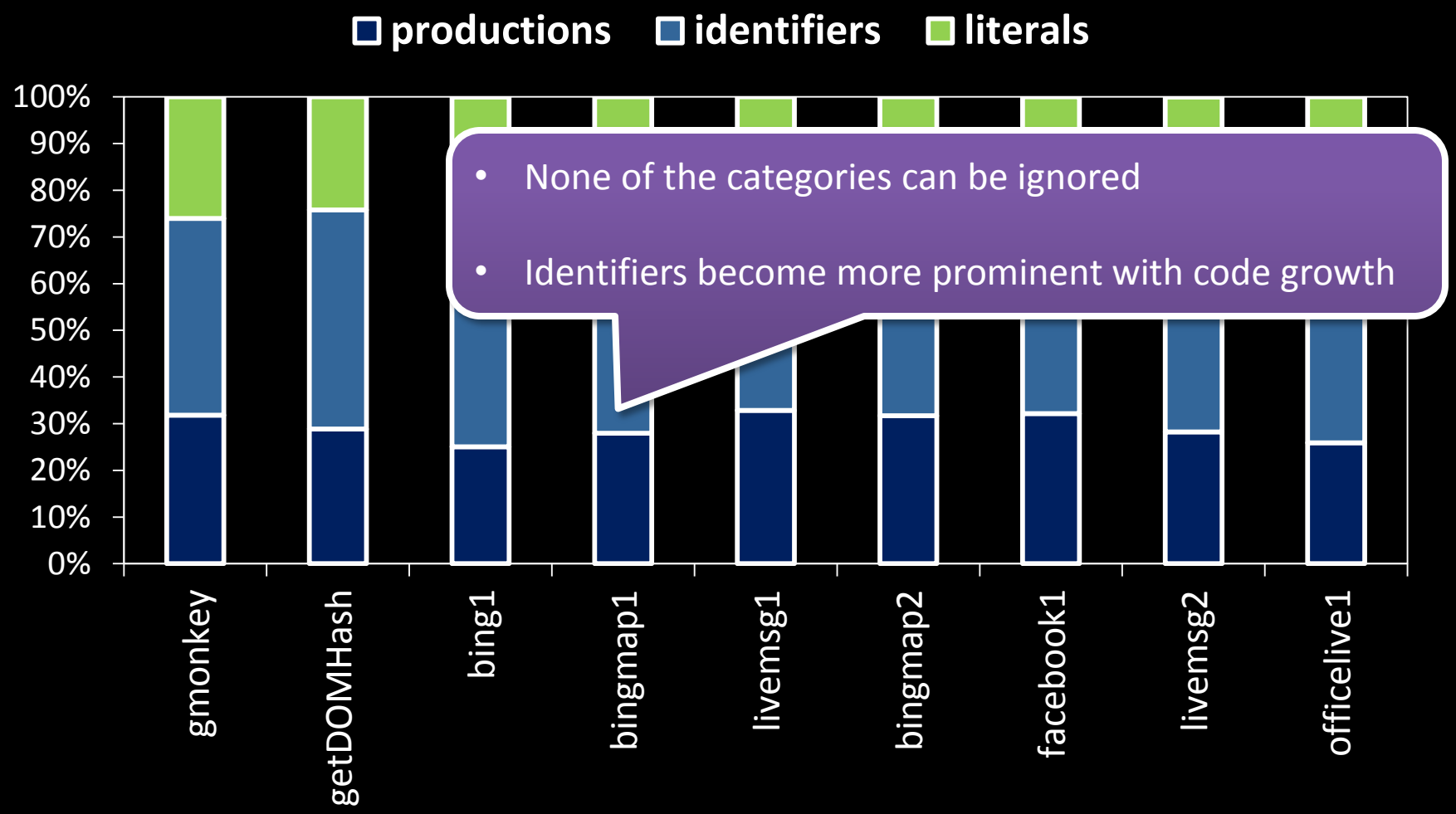


Benchmarking JSZap

Benchmark name	Source lines	Source bytes
gmonkey	922	17,382
getDOMHash	1,136	25,467
bing1	3,758	77,891
bingmap1	3,473	80,066
livemsg1	5,307	93,982
bingmap2	9,726	113,393
facebook1	5,886	141,469
livemsg2	7,139	156,282
officelive1	22,016	668,051

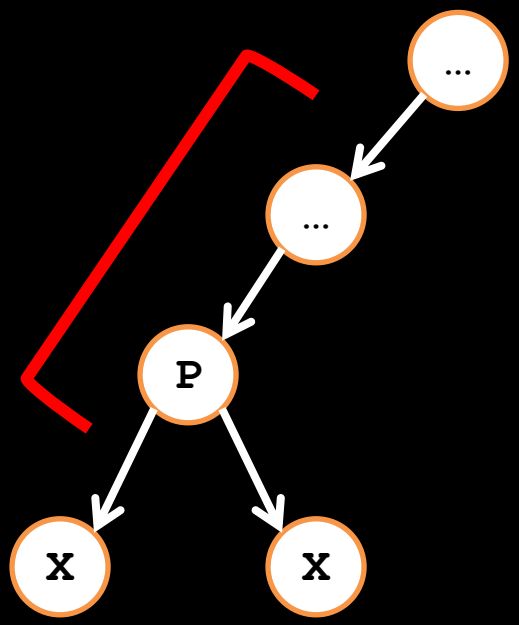
- JavaScript files up to 22K LOC
- Variety of app types
- Both hand-generated, and machine-generated
- gzipped everything

Components of JavaScript Source



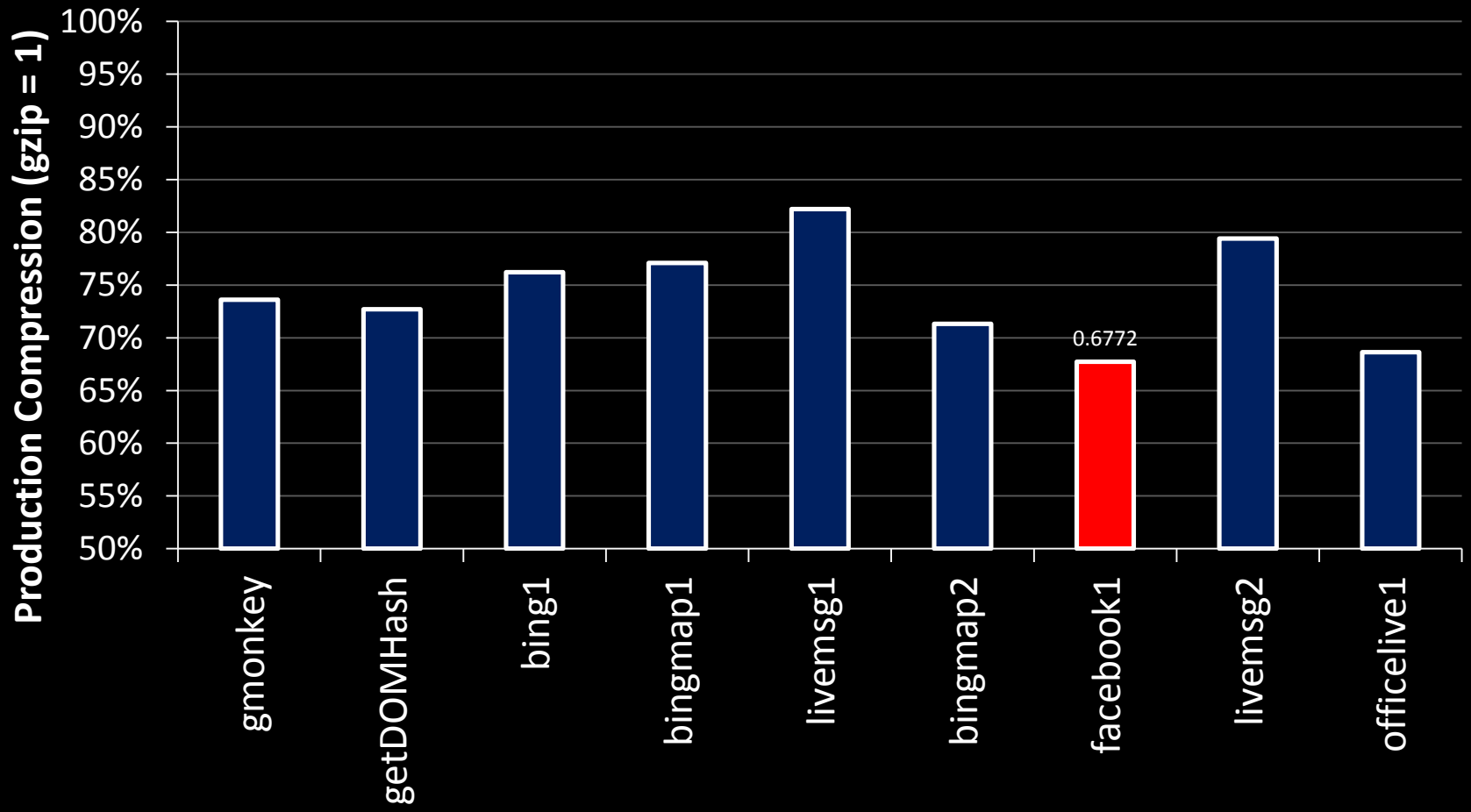
Compressing the Production Stream

- Frequency-based production renaming
- Differential encoding: 26 and 57 => 2 and 3
- Chain rule: eliminate predictable productions
- Tree-based prediction-by-partial-match



- Tree context used to build a predictor
- Provides the next likely child node given context *C* and child position *p*
- Arithmetic coding: more likely=shorter IDs
- See paper for details

Production Compression with PPMC

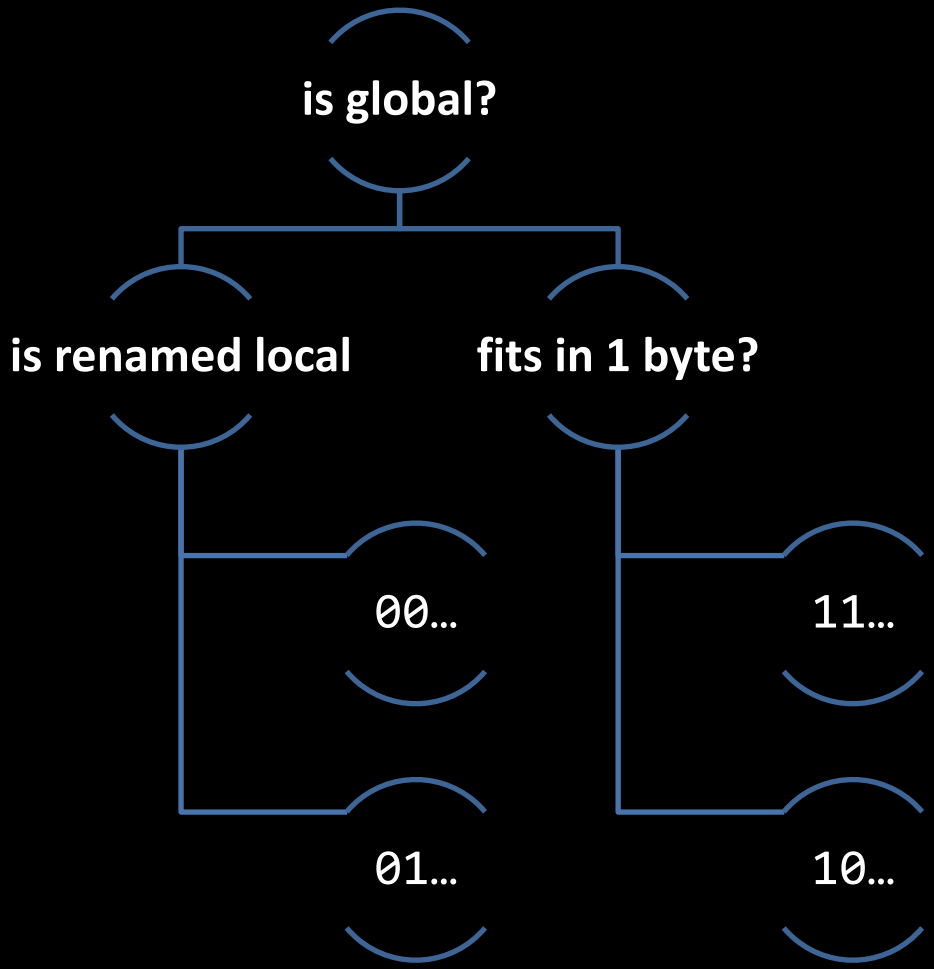


Compressing the Identifier Stream

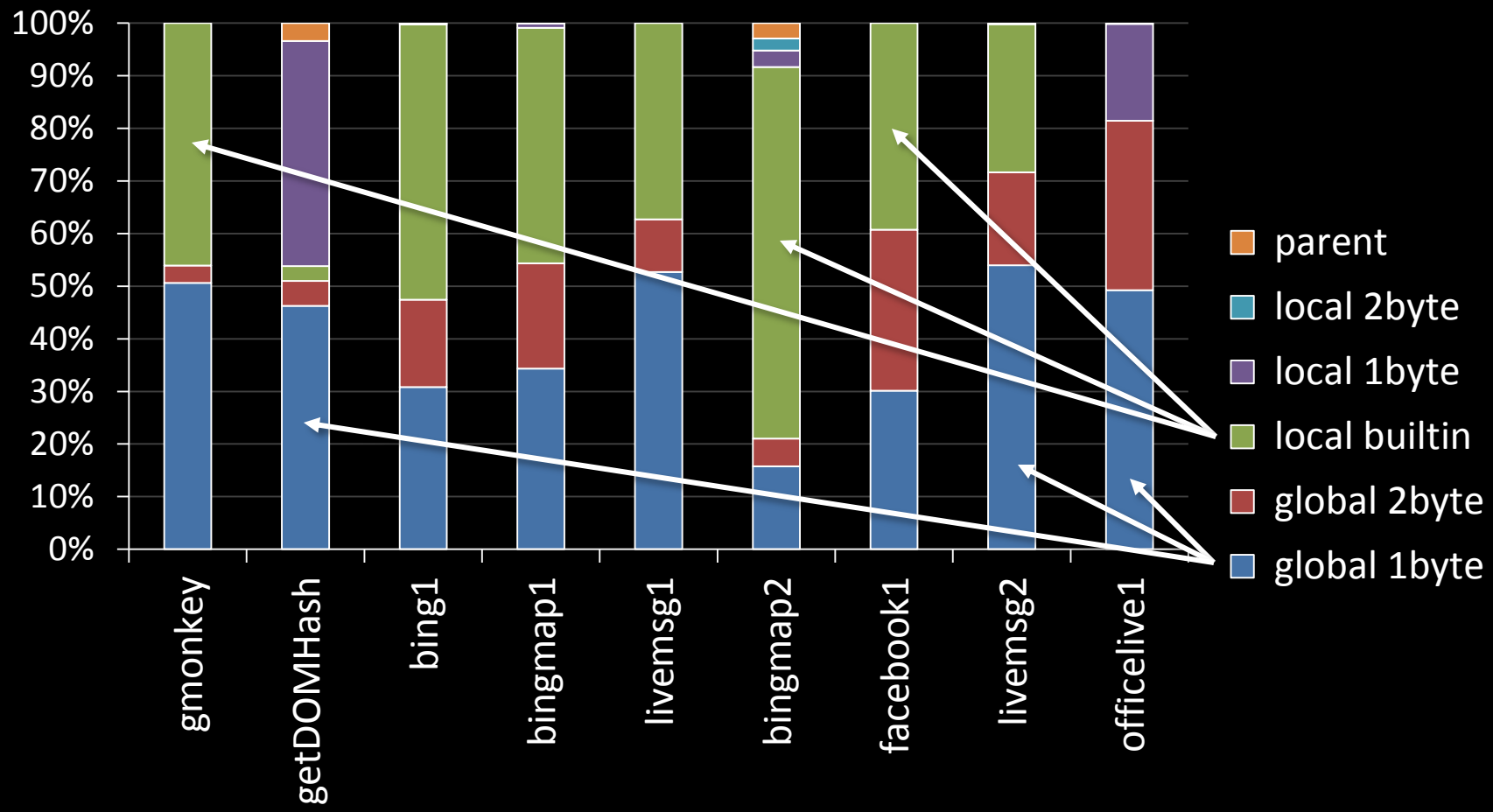
- Symbol tables instead of identifier stream:
  - Compress redundancy: offset into table
  - Global or local symbol tables
  - Use variable-length encoding

- Other techniques:
  - Sort symbols by frequency
  - Rename local variables

Variable-length Encoding for Identifiers



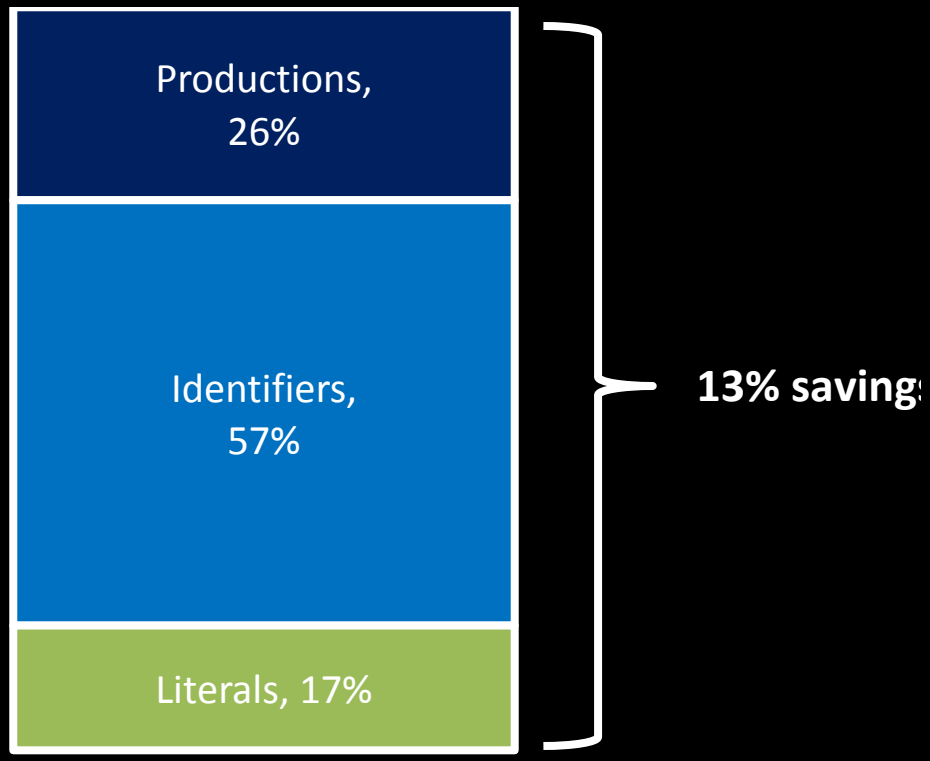
Variable-Length Identifier Encoding



Symbol Tables: Effectiveness

Compressing Literals

- Symbol tables
- Grouping literals by type
- Pre-fixes and post-fixes
- These techniques result in 5-10% savings compared to gzip


Productions,
26%
Identifiers,
57%
Literals, 17%
13% saving

Summary and Conclusions

- JSZap: AST-based compression for JavaScript
- Propose a range of techniques for compressing
  - Productions
  - Identifiers
  - Literals
- Preliminary results are encouraging: 10% savings over gzip
- Future focus
  - Latency measurements
  - Browser integration


Security (AdSafe)
Well-formedness
?
AST
representation
Unblocking HTML parser
Compression with JSZap
Caching and incremental updates

Questions?