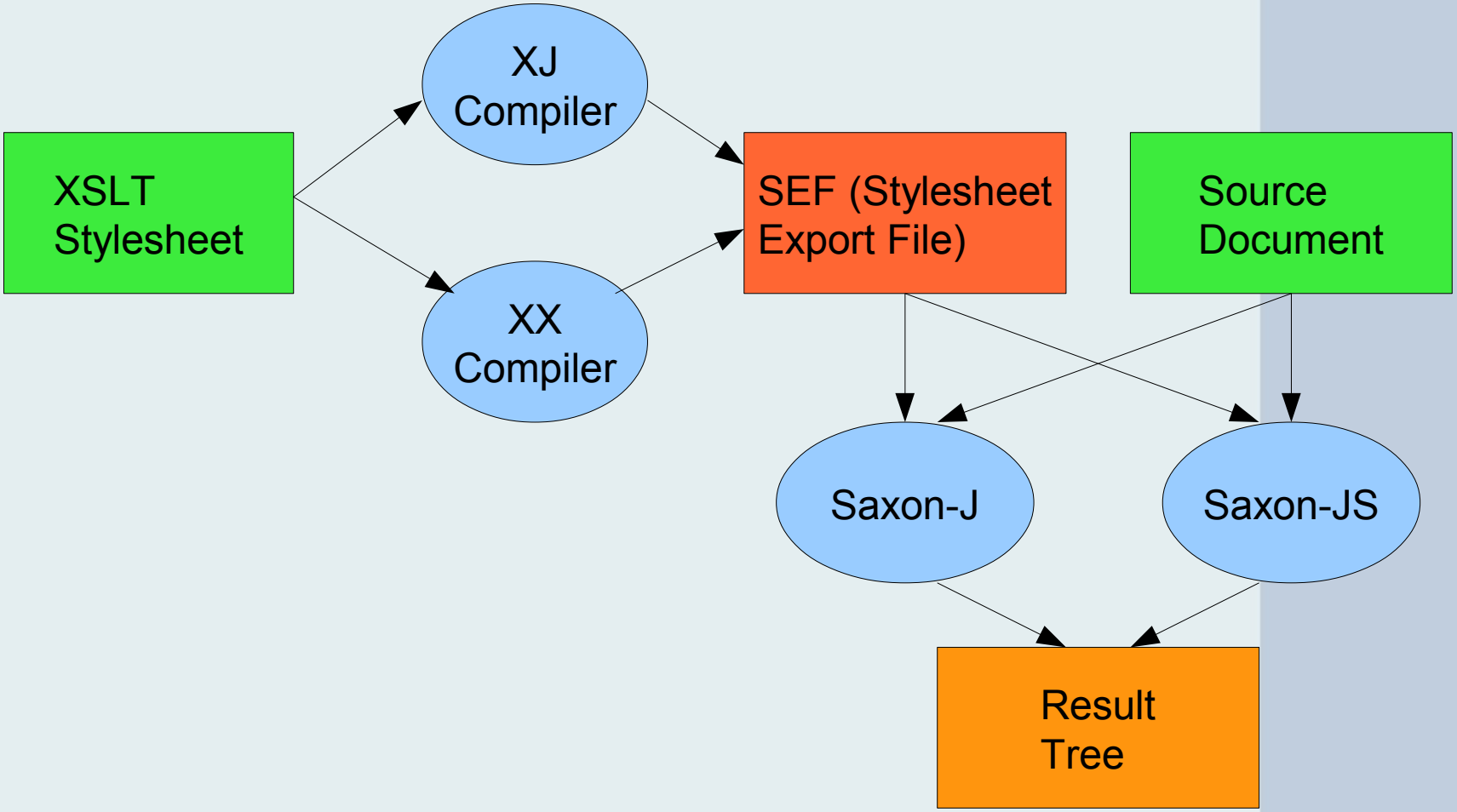
SAXONICA.COM
XSLT AND XQUERY PROCESSING

An XSLT compiler written in XSLT
Can it Perform?

Michael Kay, Saxonica
John Lumley, Saxonica+jwL
XML Prague 2019

The XX Compiler



What the Compilers Do

- Static Preprocessing
  - include/import, use-when, shadow attributes
- Structural Validation and normalization
- XPath Parsing
- Resolving component references
- Type Checking
  - inject code for run-time checks and conversions
- Optimization
- Streamability Analysis

The Compiler as Transformer...

At a first level of approximation,

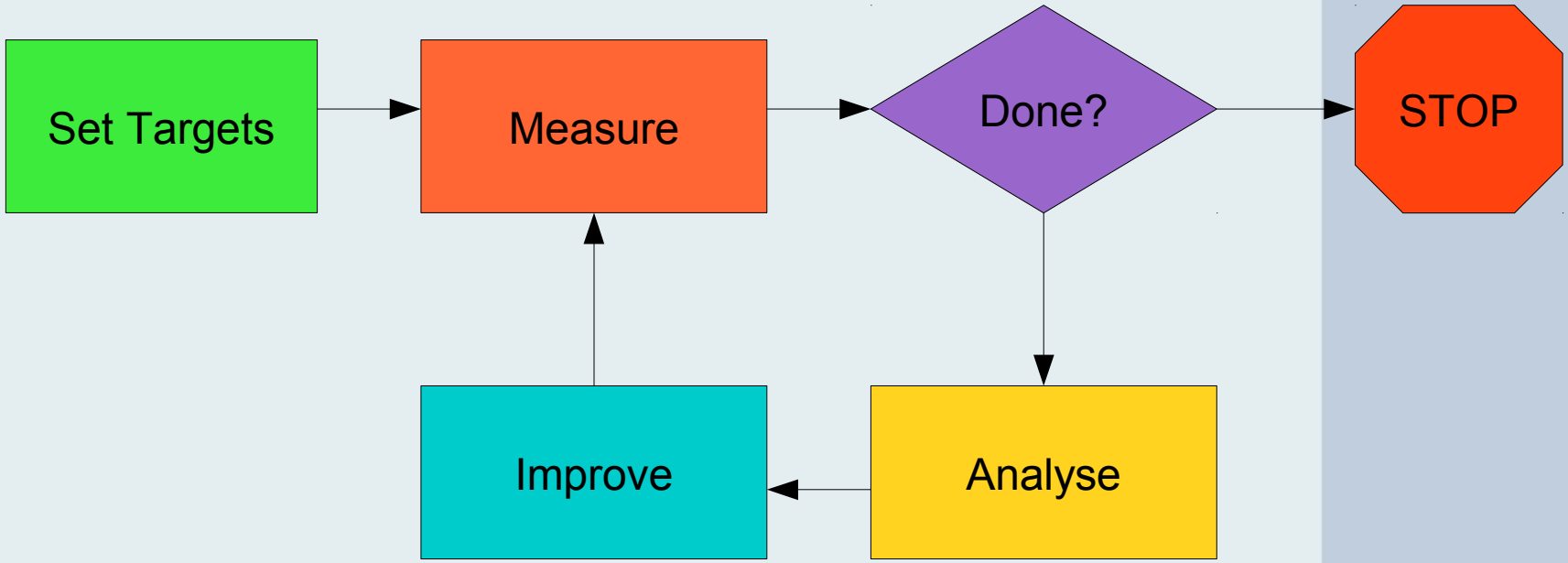
The XSLT Compiler is a multiphase tree-to-tree transformation operating in 6 passes

So it makes sense to write it in XSLT...

XJ / XX Differences

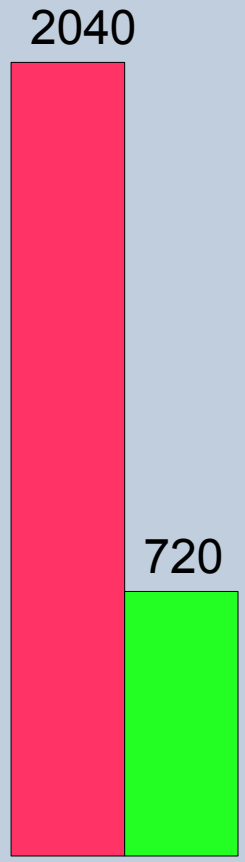
- XJ uses an in-memory tree of Java objects
  - mutable
  - decorated with arbitrary object values
- XX uses an XDM tree
  - immutable
  - decorated with string-valued attributes

Performance Engineering Process



Targets for XX

- Task:
  - Use the compiler to compile itself
- Baseline:
  - Time taken by XJ: 240ms
  - Time taken by XX on Java: 2040ms
  - Time taken by XX on Chrome: 90s
- Target:
  - XX on Java: 720ms
  - XX on Node.js: 3s



Measurement Techniques

- Transform -t -repeat:50
  - "whole task" measure, no breakdown
- Transform -TP:profile.html
  - breakdown by templates/functions
- Java-level profiling
- "Subtractive measurement"
  - stop doing X, see the difference
- "Additive measurement"
  - do X twice, see the difference

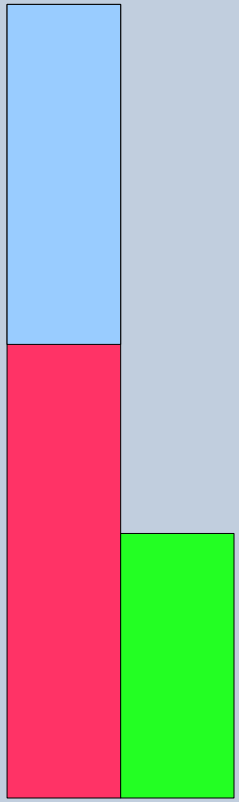
Measurement Repeatability

- Eliminate Java warm-up time
- Cut out background tasks
- Use a consistent configuration
- Let the CPU cool down
- Plug into mains power!

- Use the "best" numbers
  - external factors will never speed things up!
- Use counters rather than timing

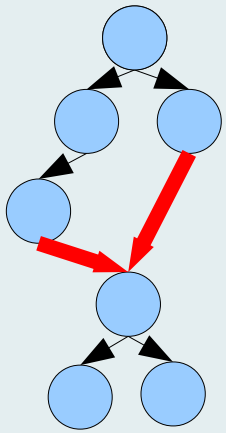
Phase 1 Improvement

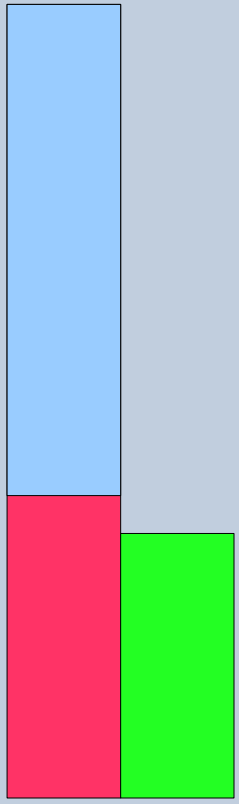
- XPath Parsing
  - XX calls a Java/JS XPath parser
  - Needs to supply static context for each expression
- Eager evaluation of the fixed parts
  - global functions and variables
- Lazy evaluation of the variable parts
  - namespace context, local variables
- Progress: 2040ms ⇛ 1280ms



Phase 3: Tree copying revisited

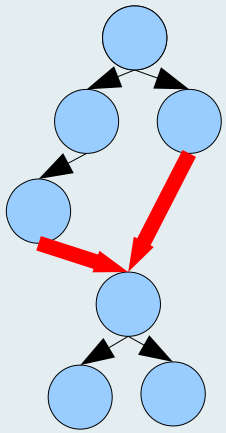
- XML Prague 2018: Efficient tree copying by avoiding parent pointers
  - implemented in Saxon 9.9
  - no performance benefits ☹
  - the reason: NAMESPACES
- do less copying
- use copy-namespaces=no
- internals: tunnel parameters
- Progress: 1120ms => 825ms

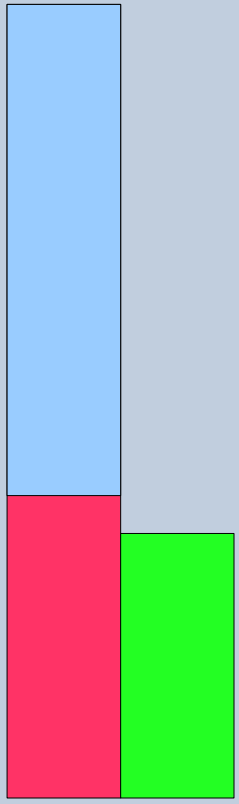




Phase 3: Tree copying revisited

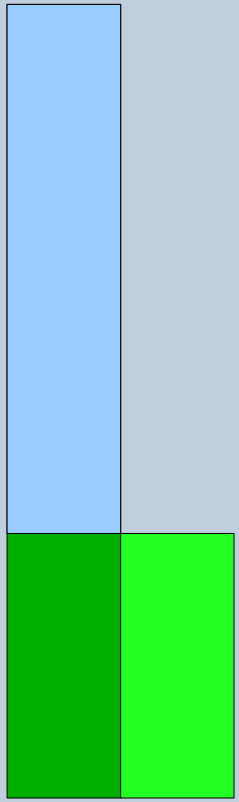
- XML Prague 2018: Efficient tree copying by avoiding parent pointers
  - implemented in Saxon 9.9
  - no performance benefits ☹
  - the reason: NAMESPACES
- do less copying
- use copy-namespaces=no
- internals: tunnel parameters
- Progress: 1120ms ⇛ 825ms





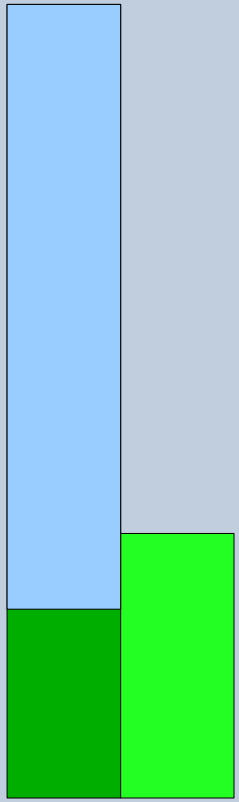
Phase 4: Algorithmic Improvements

- New algorithm for computing import precedence "on the fly"
  - avoids post-order tree traversal
- New algorithm for encoding SequenceTypes as strings
  - for fast parsing and fast comparison
- Progress: 825ms ⇒ 725ms



Phase 5: Coup de grâce

- XPath processing in parallel
  - xsl:for-each saxon:threads="8"
- Progress: 725ms ⇛ 550ms



So what about Node.js?

- Ask us at

Observations and Conclusions

- It can be done
- Debugging complex stylesheets is hard
  - as a spin-off, we improved diagnostics
- Measuring small improvements is difficult
- There's often a hot-spot that gives a big improvement for small effort
- When there isn't, you have to be prepared to make radical changes