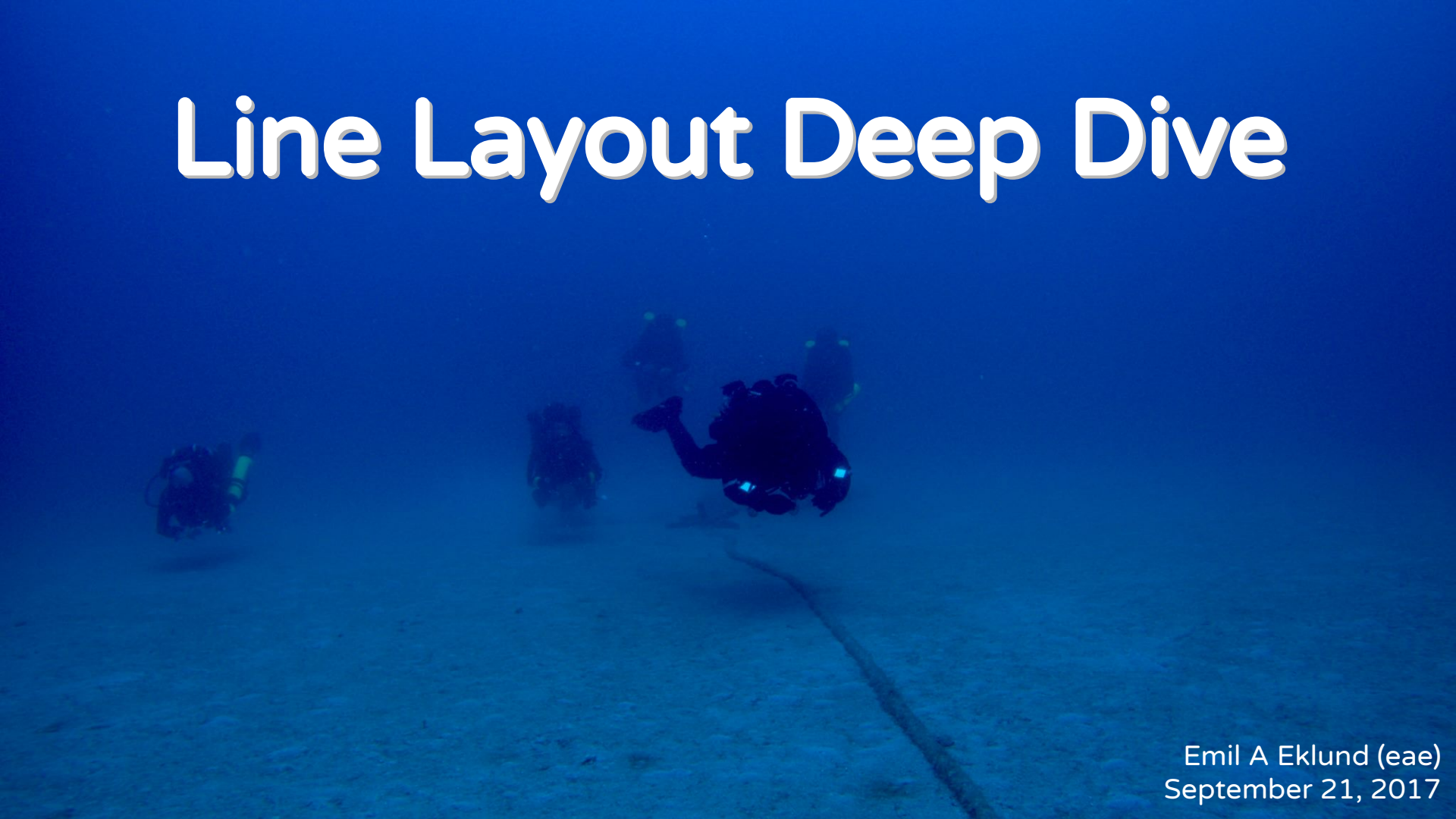
Line Layout Deep Dive



Emil A Eklund (eae)
September 21, 2017

Legacy Line Layout

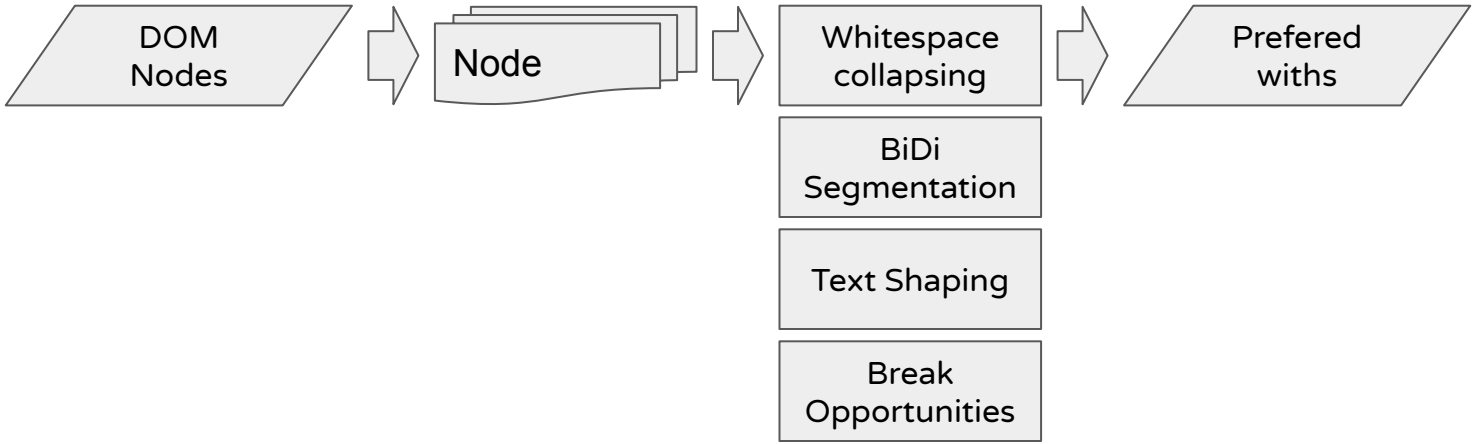
Multi-stage process, retaining little state

- Prefered width calculation
- Line breaking
- Alignment & Ellipsis
- Painting

Prefered with calculation

Iterates over nodes and processes them one by one.

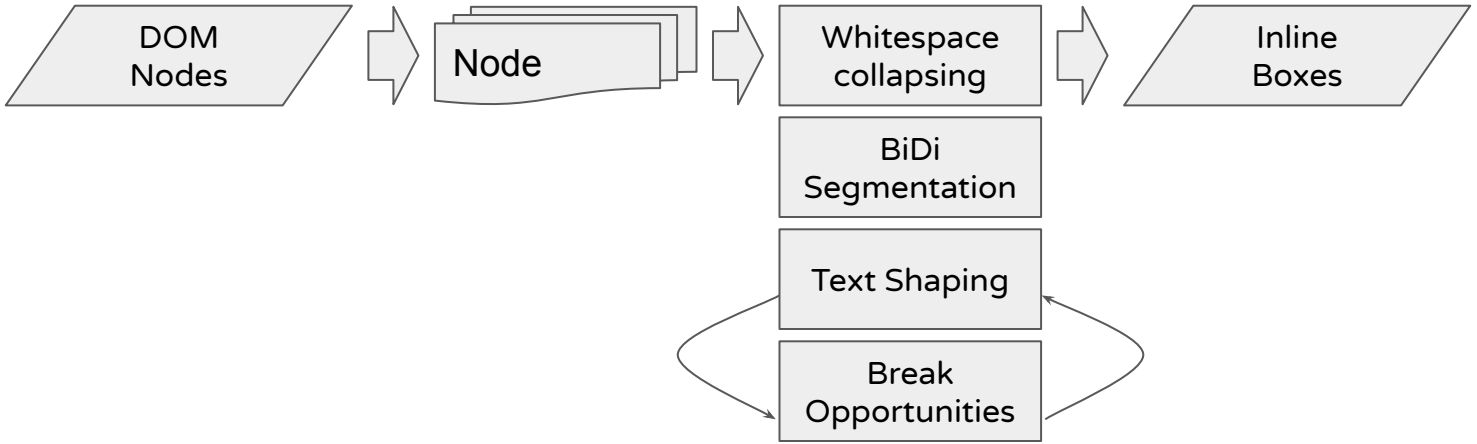
Computes minimum and maximum prefered width, stores no other information.



Line Breaking

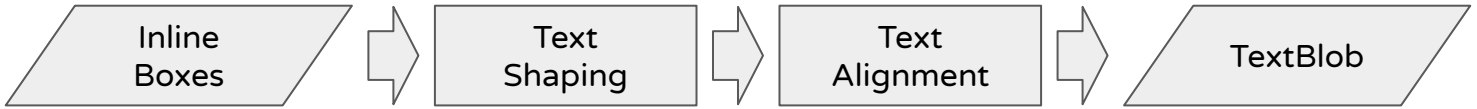
Iterates over nodes evaluating every break opportunity until available space is exceeded.

Creates structure representing line.



Painting

Iterates over all inline boxes, shapes and measures text a third time and produces a TextBlob which is then painted.



Legacy Line Layout Recap

- Retains little state.
- A lot of code duplication.
- Multiple implementations that must be kept in sync.
- Requires aggressive caching for decent performance.

NG Line Layout

Proper multi-stage, two-phase pipeline.

Intermediate data is retained across layouts.

Phase 1: Per “paragraph”

Phase 2: Per line

Phase 1: “Paragraph” Processing

The first phase takes DOM nodes and generates a list of Inline Items for the entire paragraph (root inline block).

The inlines items are persistent across layouts and are only recreated on DOM changes.

Allows the most expensive parts of line-layout to be done once and reused for each line and subsequent layout passes.

Text processing

- White-space collapsing
  - Normalize and collapse white-space.
  - Create mapping between DOM offsets and collapsed/normalized offsets
- Text transformation
  - Capitalize, lowercase, uppercase, full-width.

Segmentation & Shaping

Both segmentation and shaping are done on the entire content of the paragraph rather than node-by-node as in legacy.

- Faster BiDi segmentation.
  - Using up-to-date ICU.
- Faster run segmentation.
  - Fonts, scripts, emoji, etc.
- Faster, context-aware, and higher quality shaping.
  - Allows shaping across node boundaries!

Inline Items

NGInlineItem

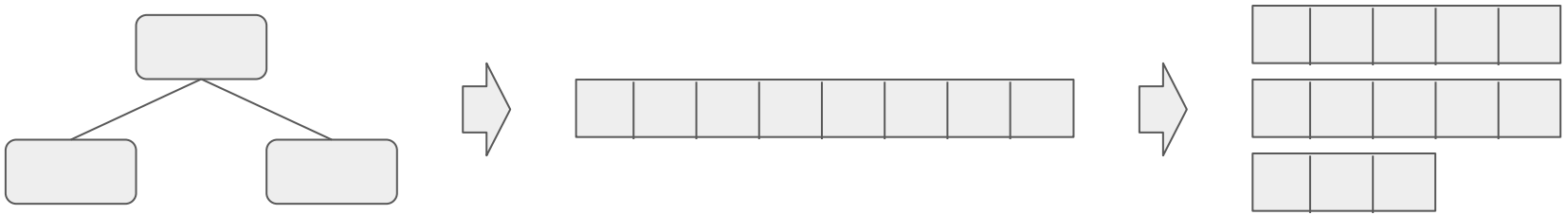
- Type
- Start/End Offset
- Script
- BiDi Level
- Orientation
- Shape Results
- Computed Style

Memory Model

Phase 1 turns DOM node tree to into a flat list of inline items per paragraph.

Phase 2 operates on the flattened list and generates lines.

A flat list model simplifies processing and is significantly faster.



Phase 2: Line Processing

Takes the inline items generated by phase 2 and generates fragments.

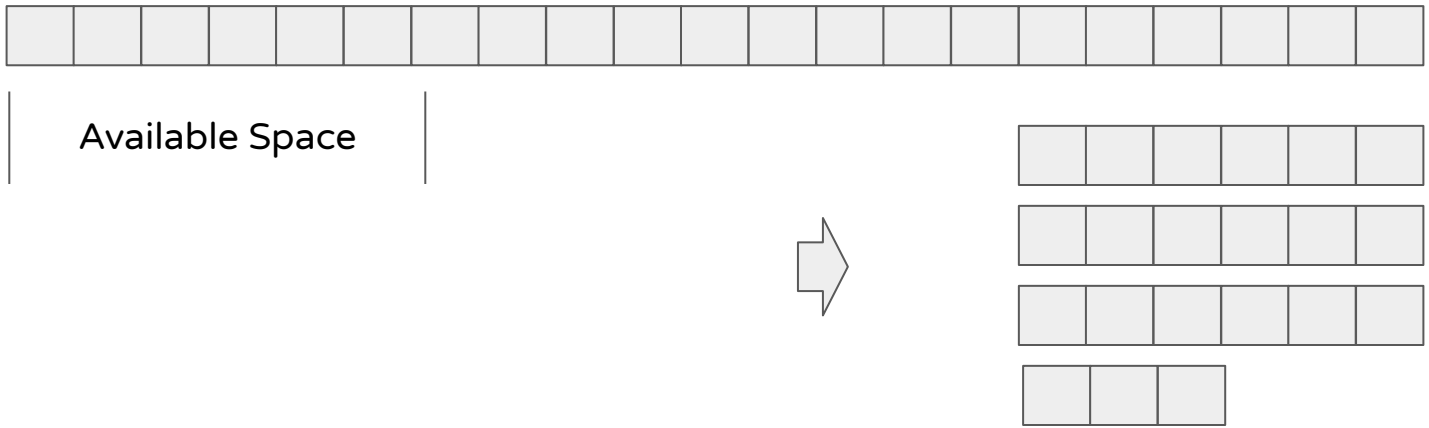
Line breaking operates on shape results and is significantly faster than before, even for the initial layout. For subsequent layouts it's almost free.

BiDi reordering and alignment done line-by-line.

Line Breaking

Uses paragraph level shape information to identify break suitable break opportunities based on available space.

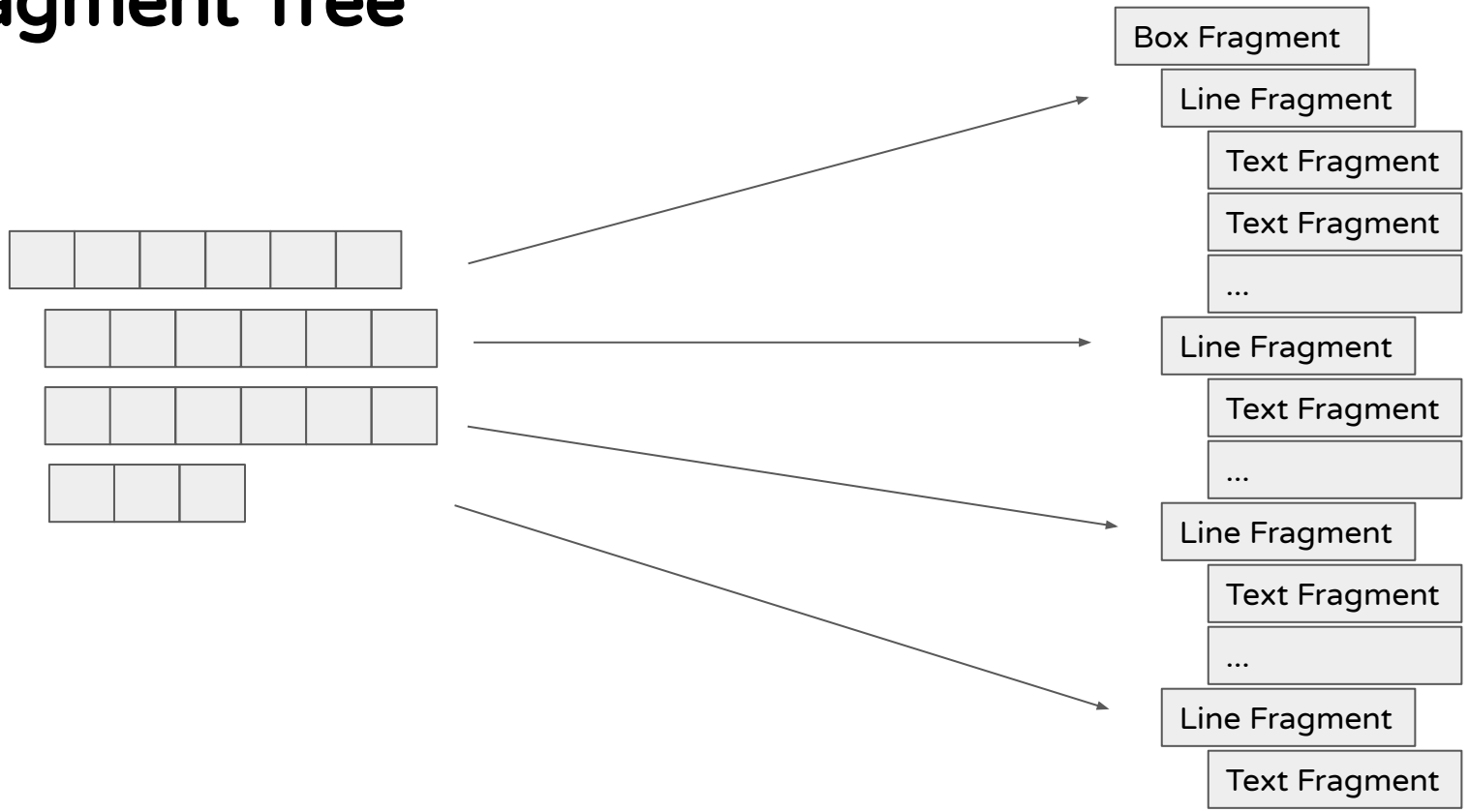
Re-uses shape results for line creation.



BiDi Reordering & Alignment

- Uses up-to-date ICU directly for BiDi reordering.
  - Legacy layout uses an out-of-date highly specialized fork.
- Much simplified text-/vertical-align implementation
  - Using a fraction of the code.
  - Significantly faster.

Fragment Tree




Thank you!