

Retrofitting Text Editors with Ragged Blocks

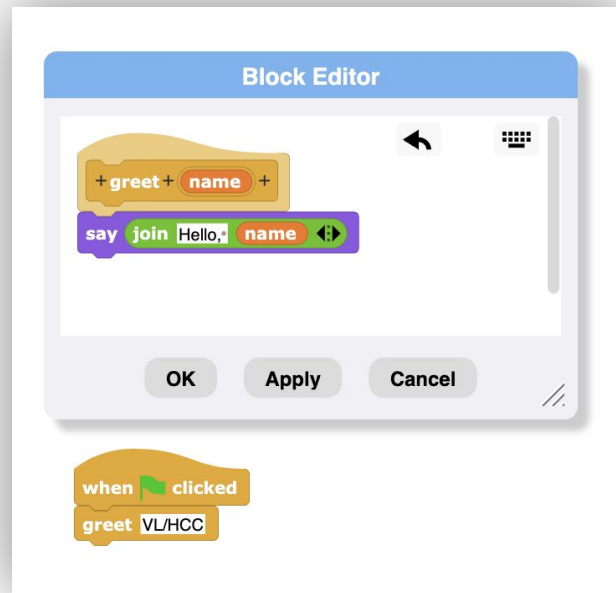
(A.K.A. paper_3796.pdf)

Sam Cohen, Jackson Slipock, and Ravi Chugh
VL/HCC 26, Paderborn, Germany

How Do We Edit Code?



Text Editors



Structure Editors

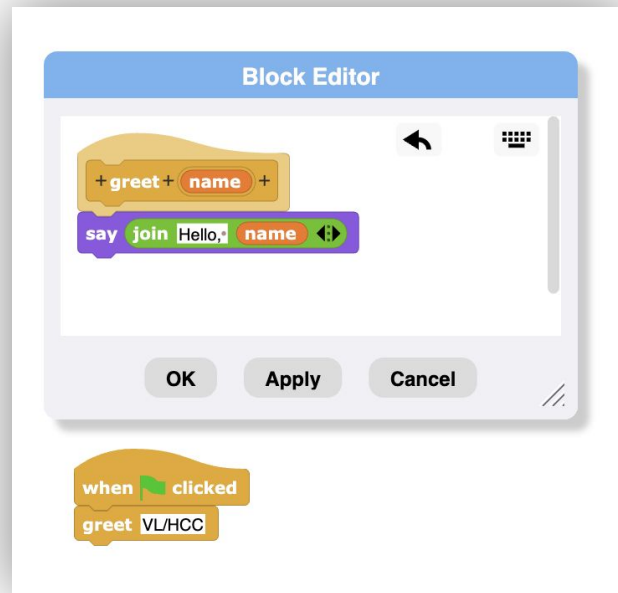
(snap.berkeley.edu)

How Do We Edit Code?



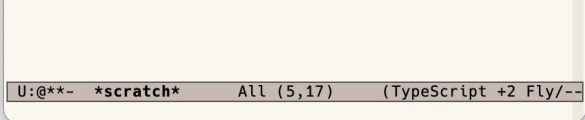
Text Editors

Goal: Blur this line

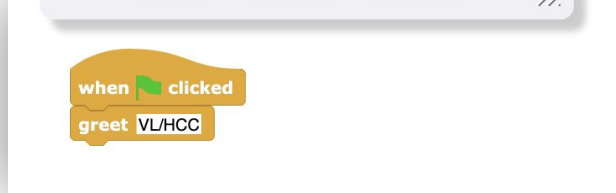


Structure Editors

(snap.berkeley.edu)



Text Editors



Structure Editors

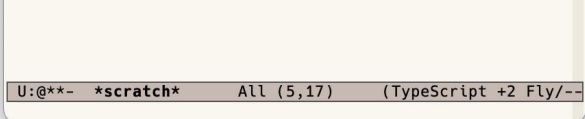
(snap.berkeley.edu)

Goal: Blur this line

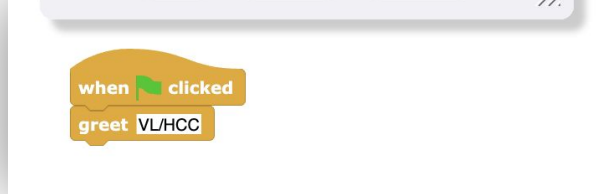


“As one student said: ‘Java is actual code, while Snap! Is something nobody will let you code in.’ This same point was made by another student who said: ‘if we actually want to program something, we wouldn’t have blocks.’”

—Weintrop & Wilensky (2015),
To Block or not to Block, That is the Question:
Students’ Perceptions of Blocks-based Programming



Text Editors



Structure Editors

(snap.berkeley.edu)

Goal: Blur this line



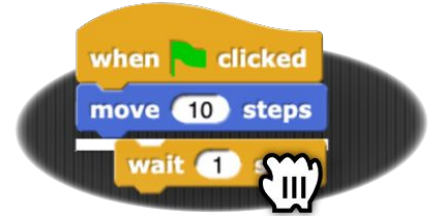
“One technological design factor that may plausibly reduce friction is the use of graphical programming interfaces (e.g. Scratch) which rely less on conventional text-based programming... However, [one participant] noted that he avoided using GUI-based coding tools because ‘block interfaces don’t feel like real coding’ ... [another participant] observed ‘I want [students] to learn how to ride a bike, because they’re gonna get a lot further on a bike than in some esoteric, you know, device that I make, and then later [they] have to relearn how to ride a bike’”

—McNutt et al. (2025), Slowness Politics and Joy:
Values That Guide Technology Choices in
Creative Coding Classrooms

Prior Approaches to “Blurring the Line”

```
2 function main(  
3     move(10);  
4     wait|  
5
```

Text Editors



Structure Editors

Prior Approaches to “Blurring the Line”

```
2 function main(  
3     move(10);  
4     wait|  
5
```

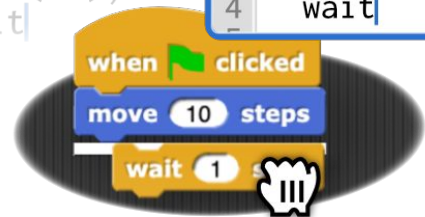
Text Editors

```
2 function main(  
3     move(10);  
4     wait|  
5
```

```
2 function main(  
3     move(10);  
4     wait|
```

Greenfoot¹

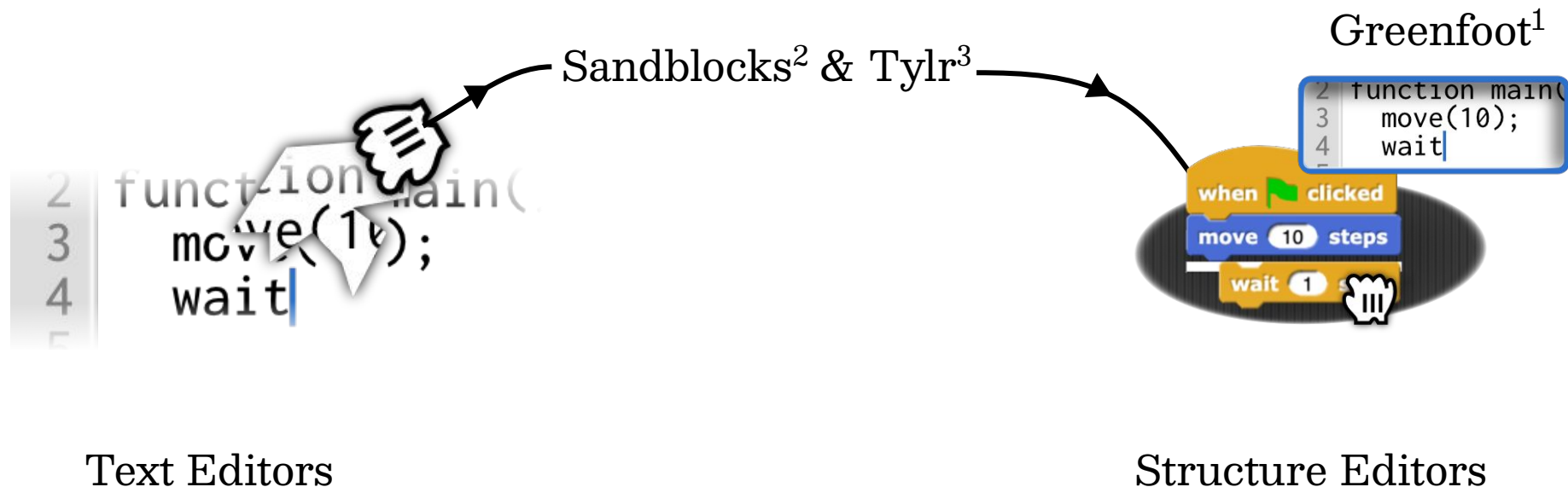
```
2 function main(  
3     move(10);  
4     wait|
```



Structure Editors

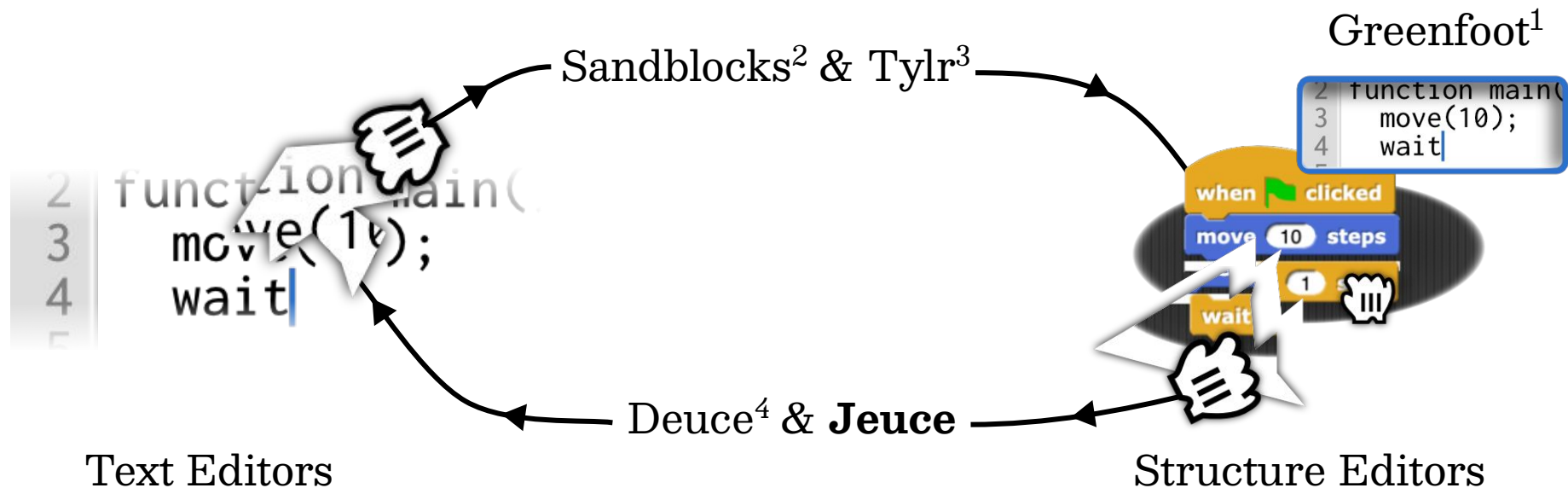
¹Kölling et al. (2015)

Prior Approaches to “Blurring the Line”



¹Kölling et al. (2015) ²Beckmann et al. (2023) ³Moon et al. (2023)

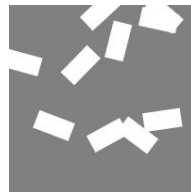
Prior Approaches to “Blurring the Line”



¹Kölling et al. (2015) ²Beckmann et al. (2023) ³Moon et al. (2023) ⁴Hempel et al. (2018)

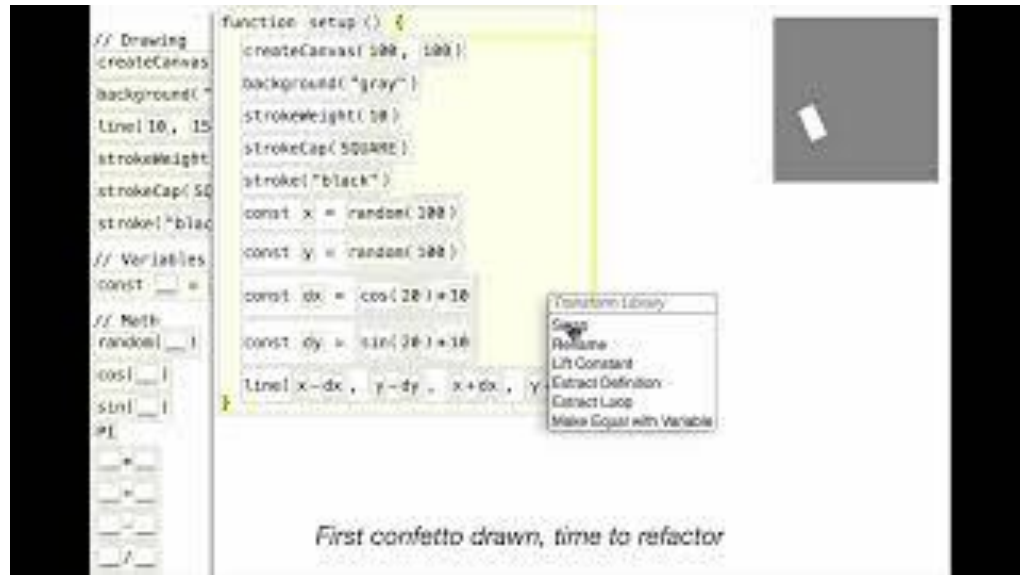
Demo

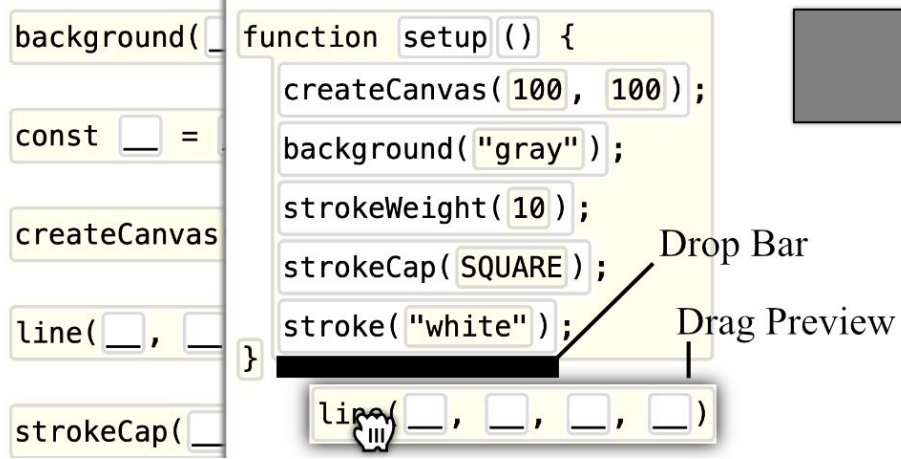
Draw confetti using p5.js



Demo

Draw confetti using p5.js





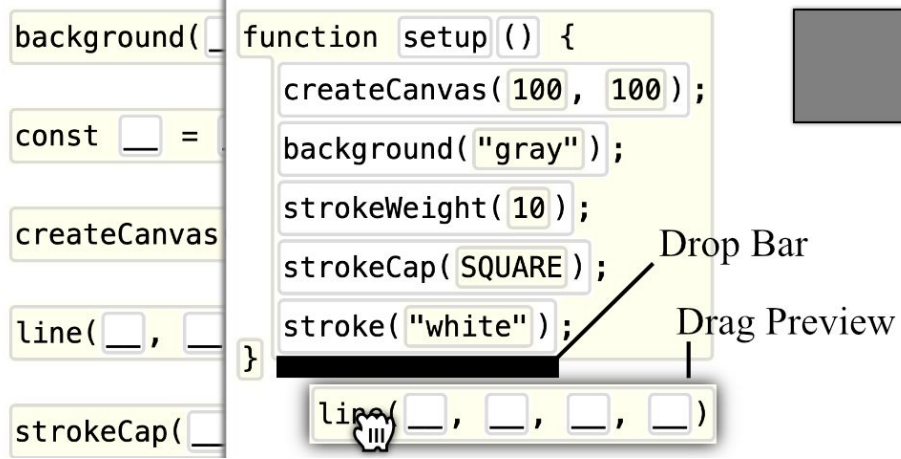
Drag and Drop Editing^{1,2,3,4}

¹Scratch (Resnick et al., 2009)

²Snap! (Harvey et al., 2013)

³Blockly (Google, 2013)

⁴MakeCode (Ball et al., 2019)



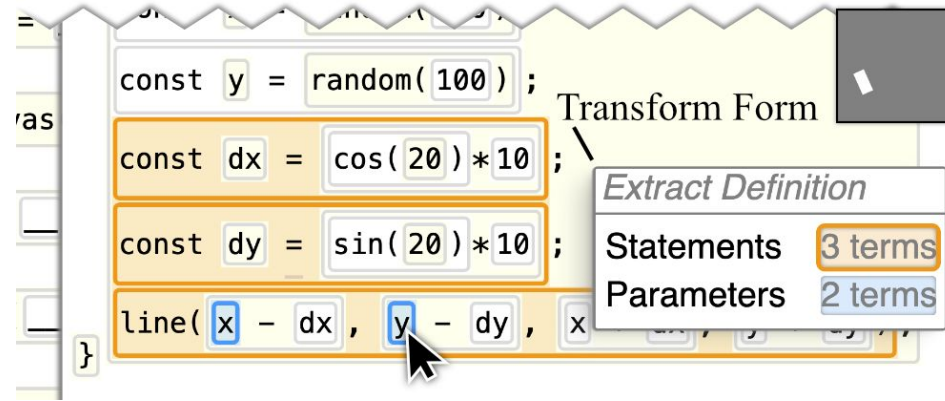
Drag and Drop Editing^{1,2,3,4}

¹Scratch (Resnick et al., 2009)

²Snap! (Harvey et al., 2013)

³Blockly (Google, 2013)

⁴MakeCode (Ball et al., 2019)



Menu-Based Transforms⁵ & Semantics of Transforms^{6,7}

⁵Deuce (Hempel et al., 2018)

⁶Hazelnut (Omar et al., 2017)

⁷Pantograph (Prinz et al., 2025)

Implementation

*Layout
Algorithm*

```
pairs =  
[ (i, j)  
  for i in range(0, 10)  
    for j in range(0, 10)  
      if i != j  
  ]
```

Boxes



Layout Algorithm

```
pairs =  
[ (i, j)  
  for i in range(0, 10)  
  for j in range(0, 10)  
  if i != j ]
```

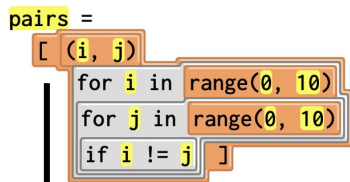
S-Blocks¹

```
pairs =  
[ (i, j)  
  for i in range(0, 10)  
    for j in range(0, 10)  
      if i != j ]
```

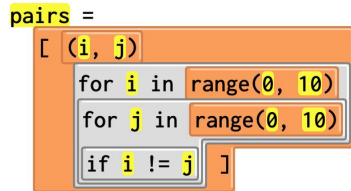
Boxes

¹Cohen & Chugh (OOPSLA, 2025)

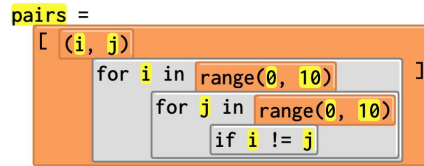
Layout Algorithm



Ragged Blocks²

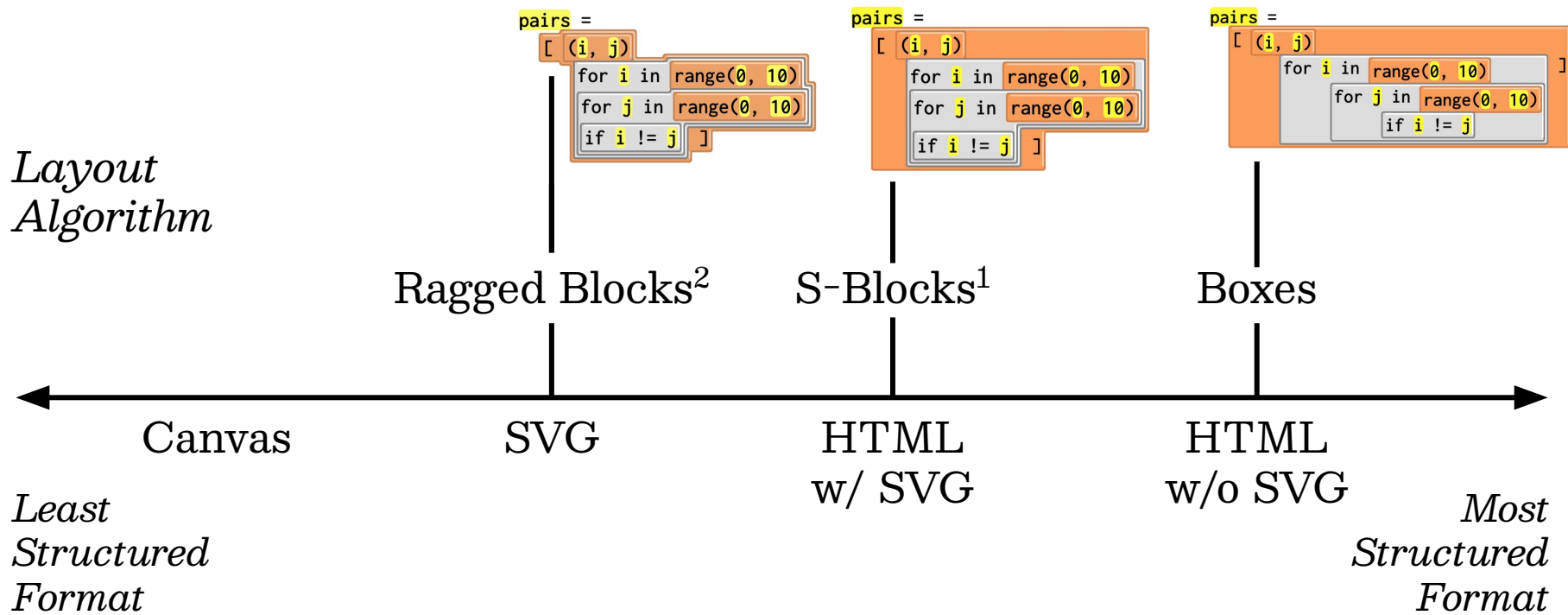


S-Blocks¹

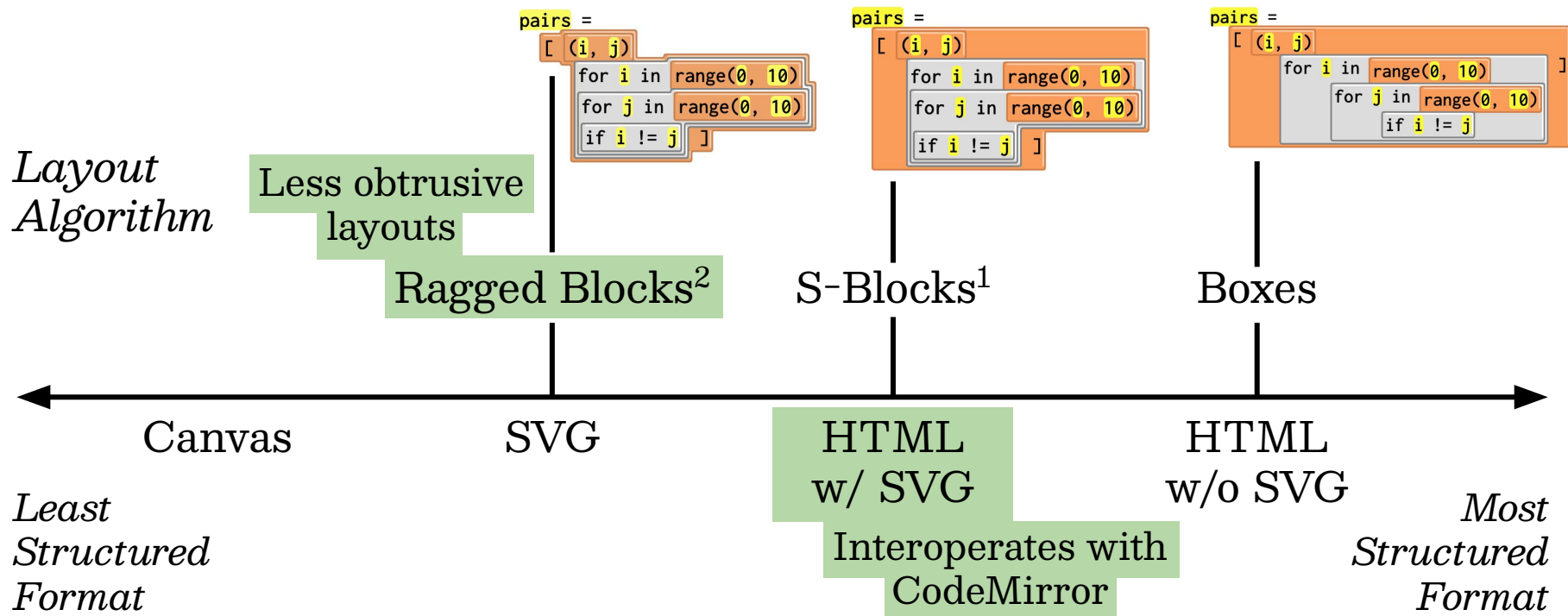


Boxes

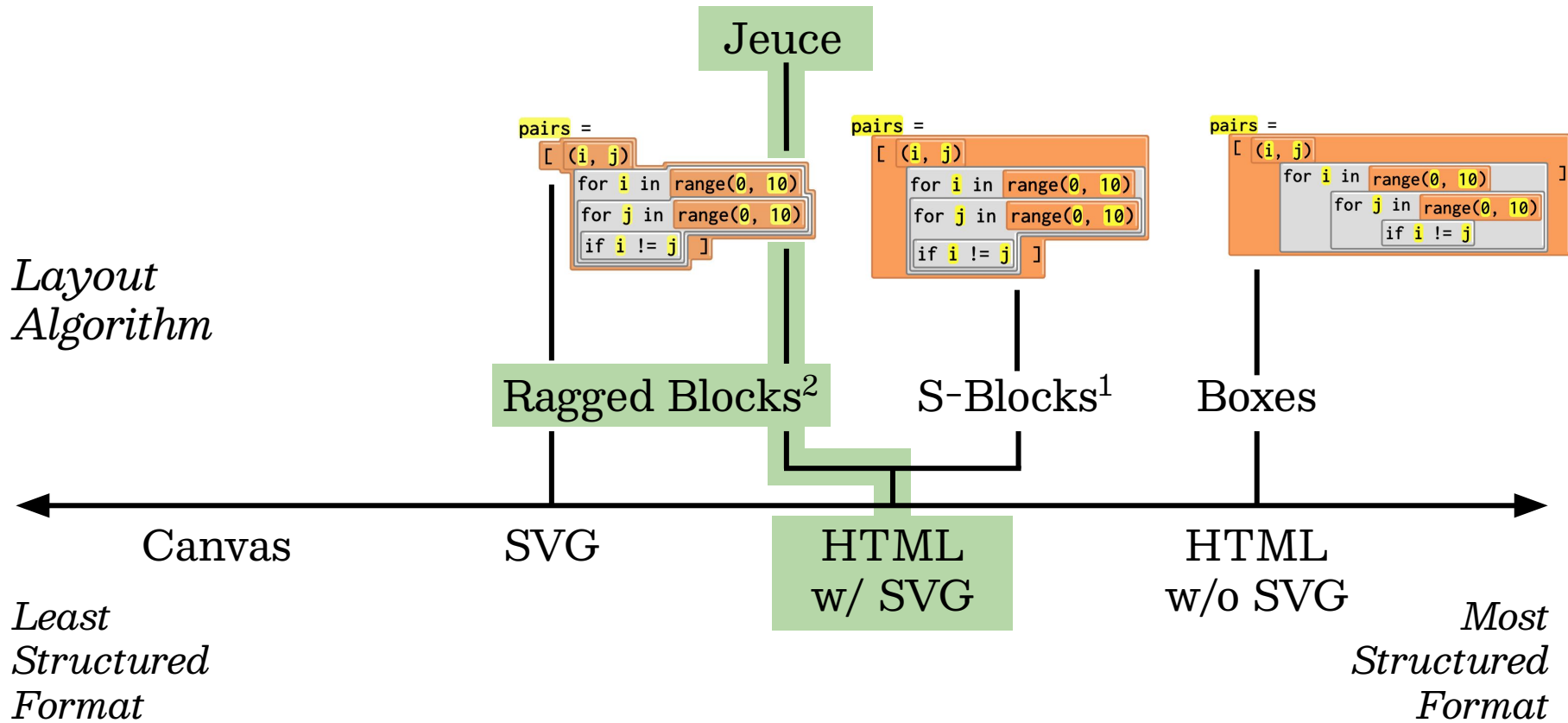
¹Cohen & Chugh (OOPSLA, 2025) ²Cohen & Chugh (UIST, 2025)



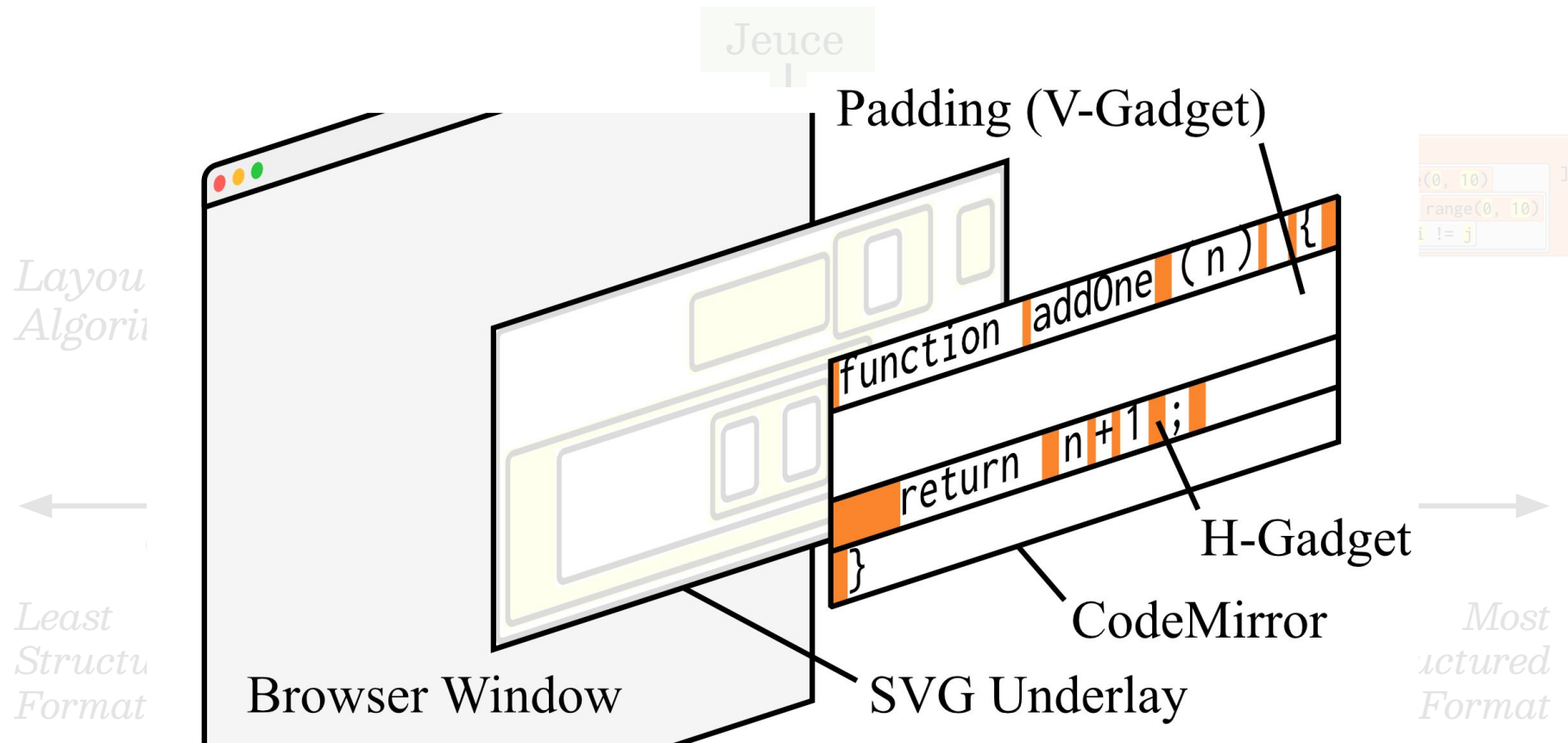
¹Cohen & Chugh (OOPSLA, 2025) ²Cohen & Chugh (UIST, 2025)

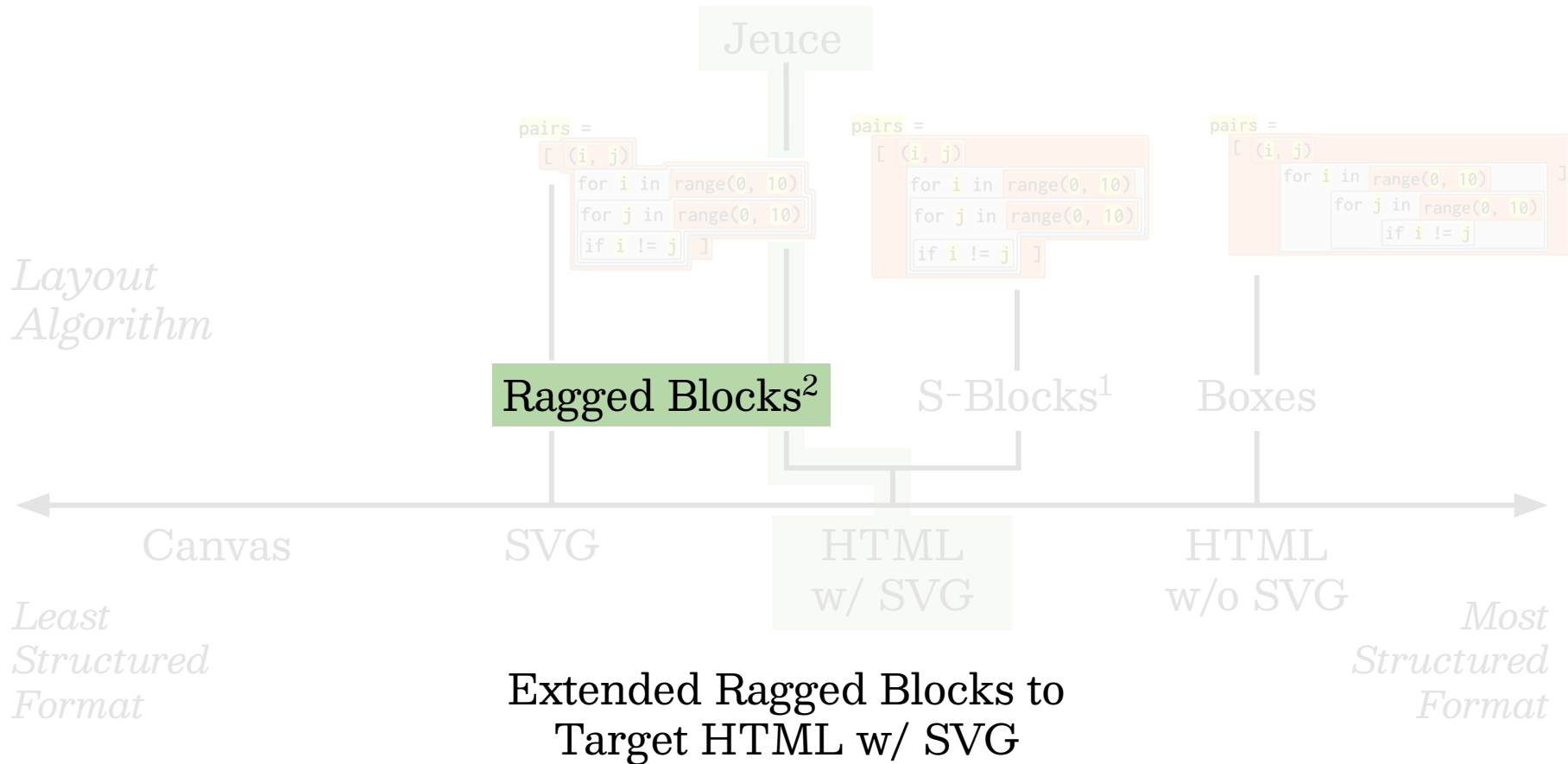


¹Cohen & Chugh (OOPSLA, 2025) ²Cohen & Chugh (UIST, 2025)



¹Cohen & Chugh (OOPSLA, 2025) ²Cohen & Chugh (UIST, 2025)





¹Cohen & Chugh (OOPSLA, 2025) ²Cohen & Chugh (UIST, 2025)

APPENDIX A
ALGORITHM DESCRIPTION

In § III, we informally described changes to the layout algorithm proposed in prior work [7]—specifically, algorithm L1_P—which accomplish our goal of rendering ragged blocks underneath an HTML-based text editor. In this appendix, we make these changes more precise, describing a minimal version of the algorithm implemented in our Jeuce prototype, using the same language as L1_P whenever possible. The primary difference is that we make the visual separation between fragments concrete through the insertion of h- and v-gadgets. Algorithm L1_P is not immediately suited to this task because, though it finds the position of the fragments in the document, it also forgets the document structure in doing so, making it difficult to associate the space between each fragment with an h-gadget, and with each line a v-gadget.²

We solve this problem by retaining some essential document structure (the location of newlines) throughout. In L1_P, a function called `layout` is used to process a layout tree into a list of regions, each one representing a line. These lines are then merged into a single *region* using the merge function. This final merge is problematic because it forgets the extent of the input lines. In the algorithm below, this final merge is implemented by `positionLines`, whose output is roughly a nested list, rather than a single list (i.e., a *region*). The output of `positionLines` is only *roughly* a list because it must produce not only the positioned lines, but the v-gadgets between them. To represent fragments or lines separated by gadgets, we define a gadget-list (*gList*), which is a list where each element is separated by a gadget, in our case represented by a number *n*.

The structure of this algorithm is thus slightly different from L1_P; rather than producing a list of lines through a single pass of the layout tree, then finding each line's vertical position, the algorithm here performs one additional pass. We first flatten the layout tree into a list of lines, then find the horizontal position of each fragment in each line, then find the vertical position of each line. (This alternate approach is discussed briefly in the prior work, since retaining line-by-line structure is a prerequisite for managing constraints between fragments on different lines [7, § A], or adding *glue* between fragments [7, § B]. Incidentally, these changes are only sketched in the prior work, whereas here we give a complete description of the “structure preserving” variant of L1_P.)

In light of these additional passes, we introduce the following notation:

$x, x_i, \dots$	Fragments with a known size
$\dot{x}, \dot{x}_i, \dots$	Fragments with a known size and horizontal position
$\ddot{x}, \ddot{x}_i, \dots$	Fragments with a known size, horizontal position, and vertical position
$n, n_i, \dots$	Integers

```
e(0, 10)
range(0, 10)
i != j
```

Most
structured
Format

Layout
Algorithm

Least
Structured
Format

Future Work

Jeuce with Style Sheets

```
main =  
  getContents >>=  
    print  
  . length  
  . filter (not . isPrefix0f)  
  . lines
```

in **red term** expected
[Char] -> Bool to match Bool

lint warning

type error

Type error style sheet

Jeuce with Style Sheets

```
main =
  getContents >=>
    print
    . length
    . filter (not . isPrefix0f)
    . lines
```

`. filter (not . isPrefix0f`

`. lines`

Chains of operators should all read
left to right or **right to left**.

switch to Operator Pipeline Lint style sheet

lint warning

Correcting the type
error, then switching
to linting style sheet

Jeuce with Style Sheets

```
main =  
  getContents >>=  
  print  
  . length  
  . filter (not . isPrefixOf "  
  . lines
```

```
  . filter (not . isPrefixOf "--")  
  . lines
```

Lint error style sheet

in red term expected
[Char] -> Bool

Chains of operators should
left to right or right to left
switch to Operator Pipeline Lint s

```
main =  
  getContents >>=  
  print  
  . length  
  . filter (not . isPrefixOf "--")  
  . lines
```

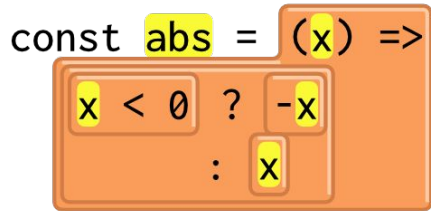
Chains of operators should all read
left to right or right to left.

lint warning

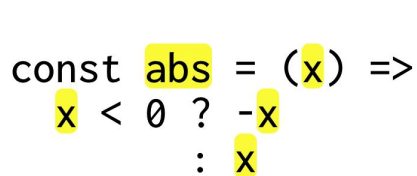
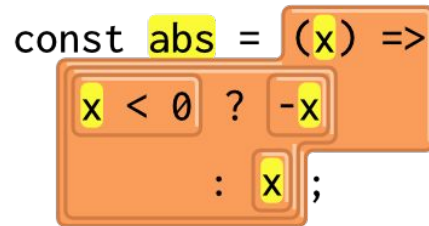
Jeuce with *Animated* Style Sheets

```
main =  
  getContents >>=  
    print  
    . length  
    . filter $ not . isPrefixOf "--"  
    . lines
```

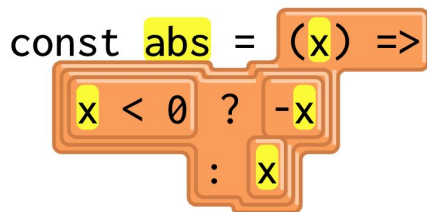

Extra Slides



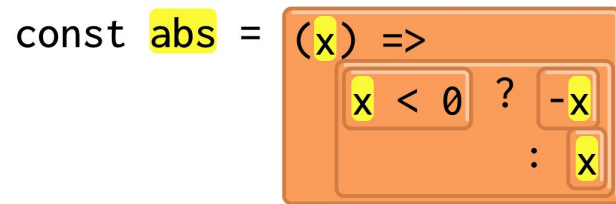
Smooth Rocks



Unstyled
Text



Rocks



S-Blocks

Blocks

