

SS : A Slide Description Language with Dependence-Scheduled Execution

Yuchi Yamaguchi
(Institute of Science Tokyo)



Key Features

- We propose **User-Defined, No Repeated Passes, Declarative** document-wide components through Dependence-Scheduled Execution.
- We also support **bidirectional synchronization** between **Source** ↔ **WYSIWYG editor**



Programmable Slide Authoring

- Programmable Slide Authoring is a way to **write slides as programs**.

- 👍 **Provides abstraction ability that makes components reusable**
- 👍 **Version control friendly**

- 😞 **Hard to use**
(Your code may contain bugs)
- 😞 **Not suited for precise alignment**

ss Mitigates These Problems

Dependence-Scheduled Execution

- Some components depend on objects across the slides (e.g., tables of contents and figure numbers).

Example: ToC in slideshow [1]

```
1 #lang slideshow
2 (require racket/string)
3
4 (define titles '())
5 (define (section title)
6   (set! titles (append titles (list title))))
7   (slide #:title title (t title)))
8
9 (slide #:title "Contents"
10  (t (string-join titles " / ")))
11 (section "Algorithm")
12 (section "Results")
```

Users have to manage dependencies carefully to create such components correctly. (part of 😞 **Hard to use**)

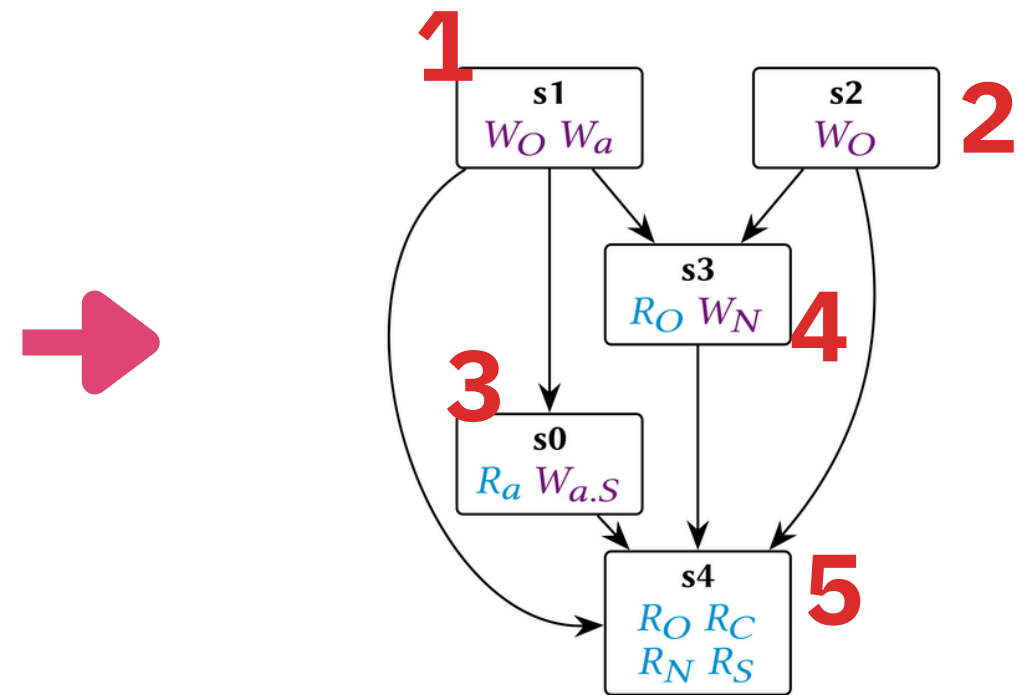
Missing Dependence!

Our approach:

Dependence-Scheduled Execution Model

- The compiler infers read/write objects for each statement.
- Reorder statements in topological order.

```
let content = "hello!"
;; ^dep?
;; DependencyQuery:
;;   write Variable(page:sample, content)
;;   ...
let t = text!(content)
;; ^dep?
;; DependencyQuery:
;;   read Variable(page:sample, content)
;;   write Objects(body)
;;   ...
```



No Repeated Pass!

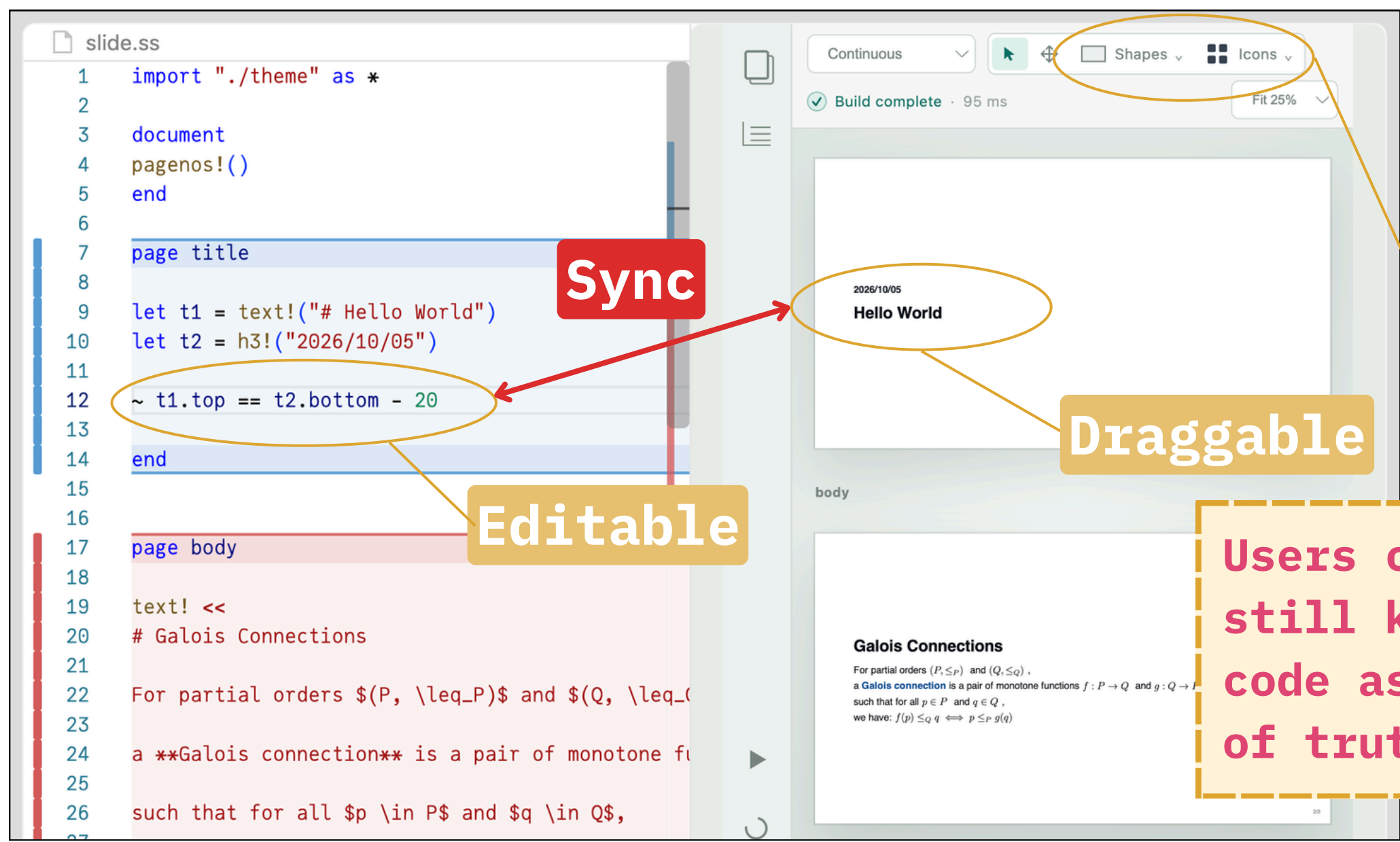
Users can write document-wide components everywhere (= Declarative)

WYSIWYG Integration

- In ss, users position slide objects by writing layout constraints between objects (e.g., `~ obj1.top == obj2.bottom - 20`)
- Constraints can express most layouts easily, but make fine adjustments difficult. (part of 😞 **Not suited for precise alignment**)

Our approach:

Bidirectional Sync Constraints ↔ WYSIWYG Editor like Sketch-n-Sketch [2]



Example of ss (proposed language)

```
1 import std:themes/default as *
2 import tablelib as tl
3
4 fn! section_title(title: String) -> Object
5   return head("Fibonacci: " ++ title)
6 end
7
8 const sample_program: String =
9   "fib(n) = n < 2 ? n : fib(n-1) + fib(n-2)"
10
11 page toc
12   text!(join(doc_objs(docctx(), "title"),
13     " / ", (title: Object) |-> content(title)))
14 end
15
16
17 page algo
18   section_title!("Implementations")
19   text!(sample_program)
20 end
21
22 page result
23   section_title!("Benchmarks")
24   text!(sample_program)
25   tl.show_table!("bench_result.csv")
26 end
```

Module System

Function

Query Component

Reuse Component

Reuse Shared Value

Data-Driven Content

Implementation & Performance

- We implemented a compiler for ss in Zig.
- We evaluated its performance on a 57-page slide deck.
- Our compiler tries to maximize parallelism and supports incremental builds to improve performance.

Phase (ms)	Initial (1 job)	Initial (8 jobs)	Incremental (Text Edit)	Incremental (Position Edit)
Analyze	40	40	33	—
Evaluate document	44	43	26	—
Prepare pages	5	5	5	—
Solve layouts	633	550	24	2
Build Render IR	153	166	10	—
Generate / Update HTML	153	154	5	8
Total	1,049	978	148	15

Compilation time targeting HTML (v0.8.3, Intel Core i7-11700K)
Medians: 5 initial runs, 30 incremental updates via LS

Comparison with Other Approaches

- slideshow** [1] has the same goal, but it is an internal DSL and retains its host language's properties, including **syntax & execution model**.
- LaTeX and Typst** have similar goals. However, they resolve document-wide information through **repeated passes**, which can complicate processing and increase compilation time.

Future Work

- Build a central package registry for reuseness over users.
- Design better syntax so that most cases can be written easily

[1] Robert Bruce Findler and Matthew Flatt., (ICFP 2004)

[2] Brian Hempel, Justin Lubin, and Ravi Chugh., (UIST 2019)