CSCI 1377

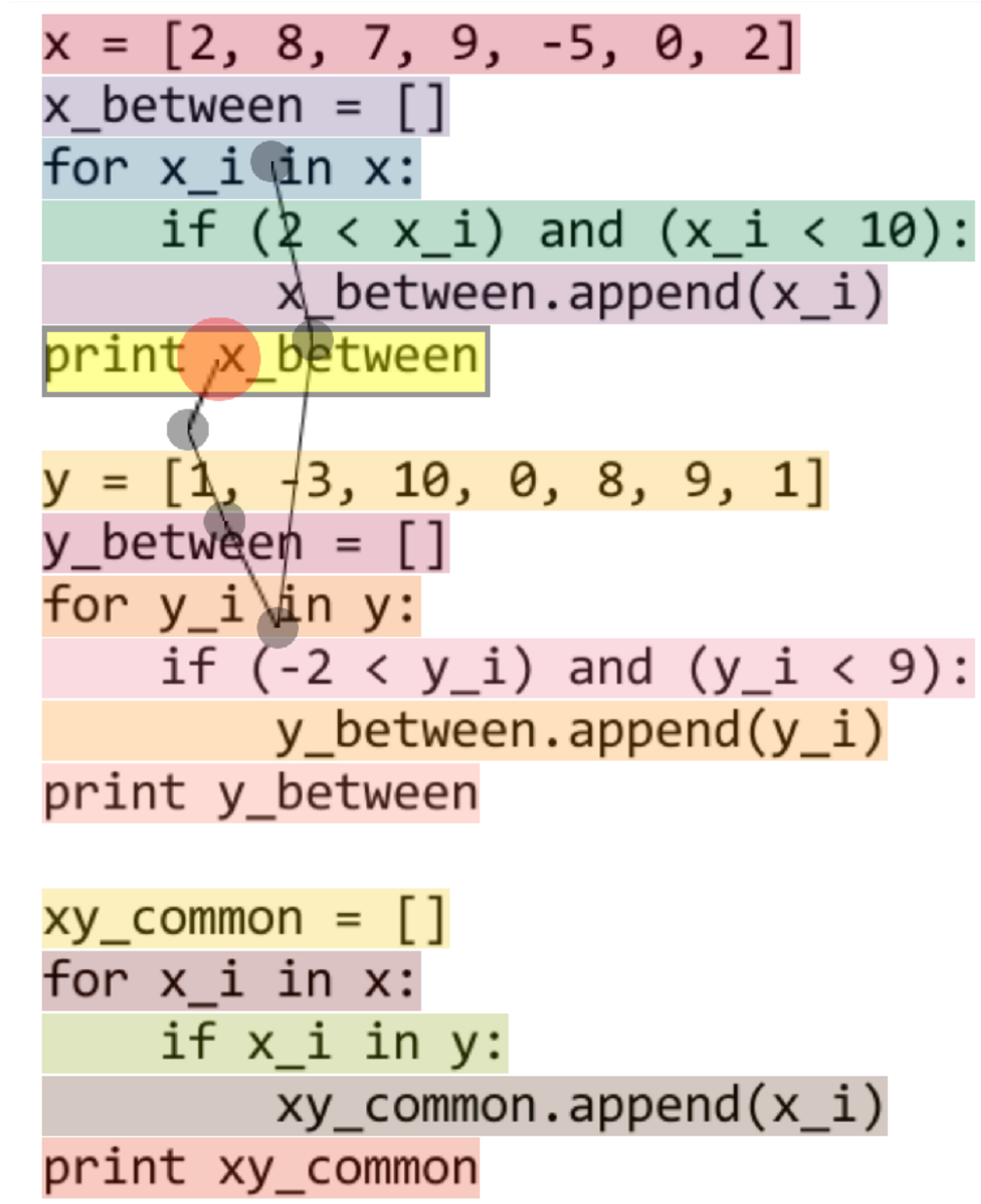
Tools for Thought

Programming III
Developer Tools

"More and more people wanted to solve problems, but were unwilling to learn octal code and manipulate bits. They wanted an easier way of getting answers out of the computer."

— Grace Hopper, on the invention of the compiler (1978)

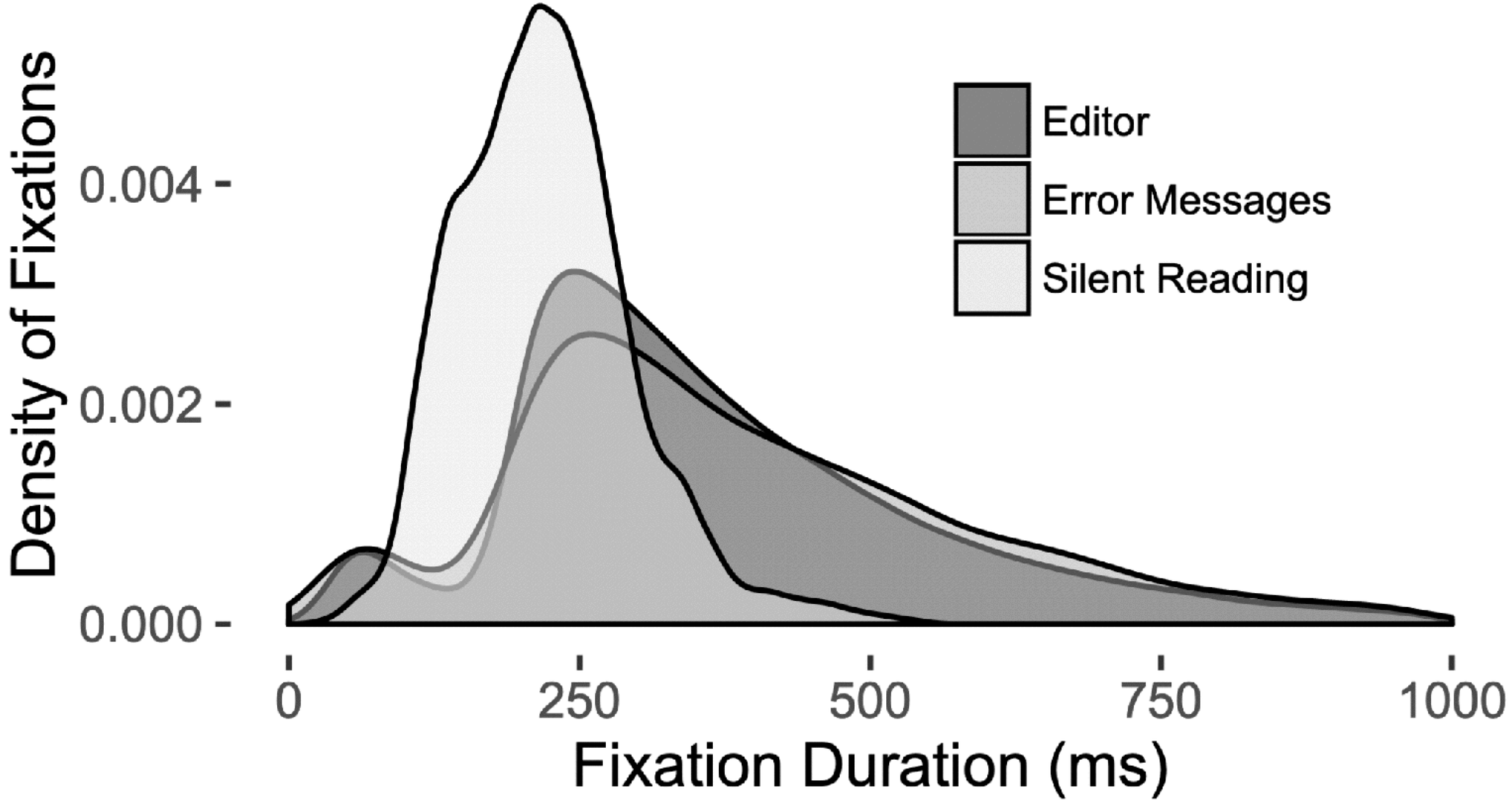
Programming languages as script



```
x = [2, 8, 7, 9, -5, 0, 2]
x_between = []
for x_i in x:
    if (2 < x_i) and (x_i < 10):
        x_between.append(x_i)
print x_between

y = [1, -3, 10, 0, 8, 9, 1]
y_between = []
for y_i in y:
    if (-2 < y_i) and (y_i < 9):
        y_between.append(y_i)
print y_between

xy_common = []
for x_i in x:
    if x_i in y:
        xy_common.append(x_i)
print xy_common
```

youtu.be/xMrenh1Va1I

Reading code is not like reading prose



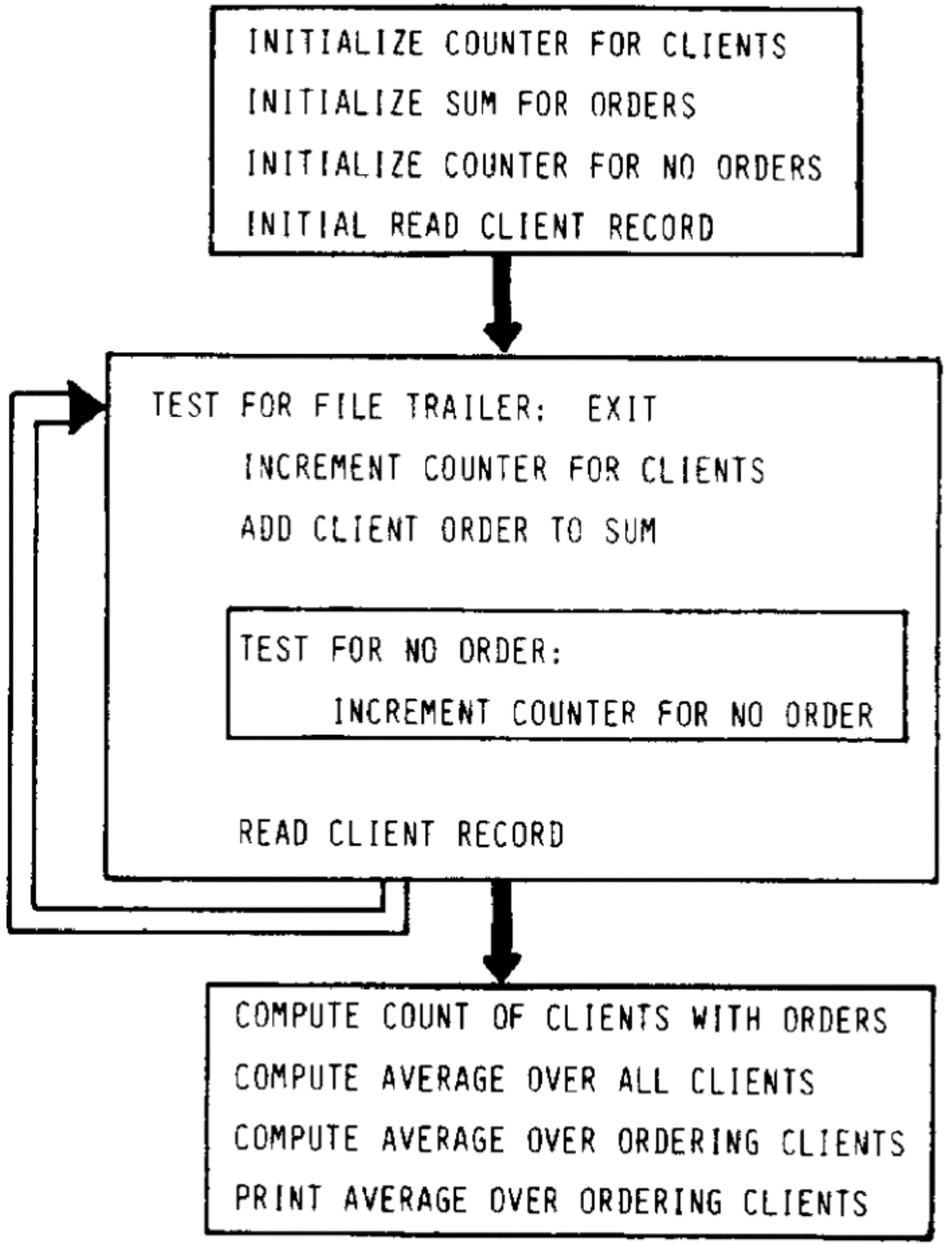
Barik et al. “Do Developers Read Compiler Error Messages?” 2017

How would a programmer mentally represent this code?

```
count_clients = 0
total_orders = 0
inactive_clients = 0
order_rec = read(order_file)
while order_rec.id != 999999:
  sum_orders()
active_clients =
  count_clients - inactive_clients
client_avg =
  total_orders / count_clients
active_avg =
  total_orders / active_clients
print(active_avg)

fn sum_orders():
  count_clients += 1
  total_orders += order_rec.quant
  if order_rec.quant == 0:
    inactive_clients += 1
  order_rec = read(order_file)
```

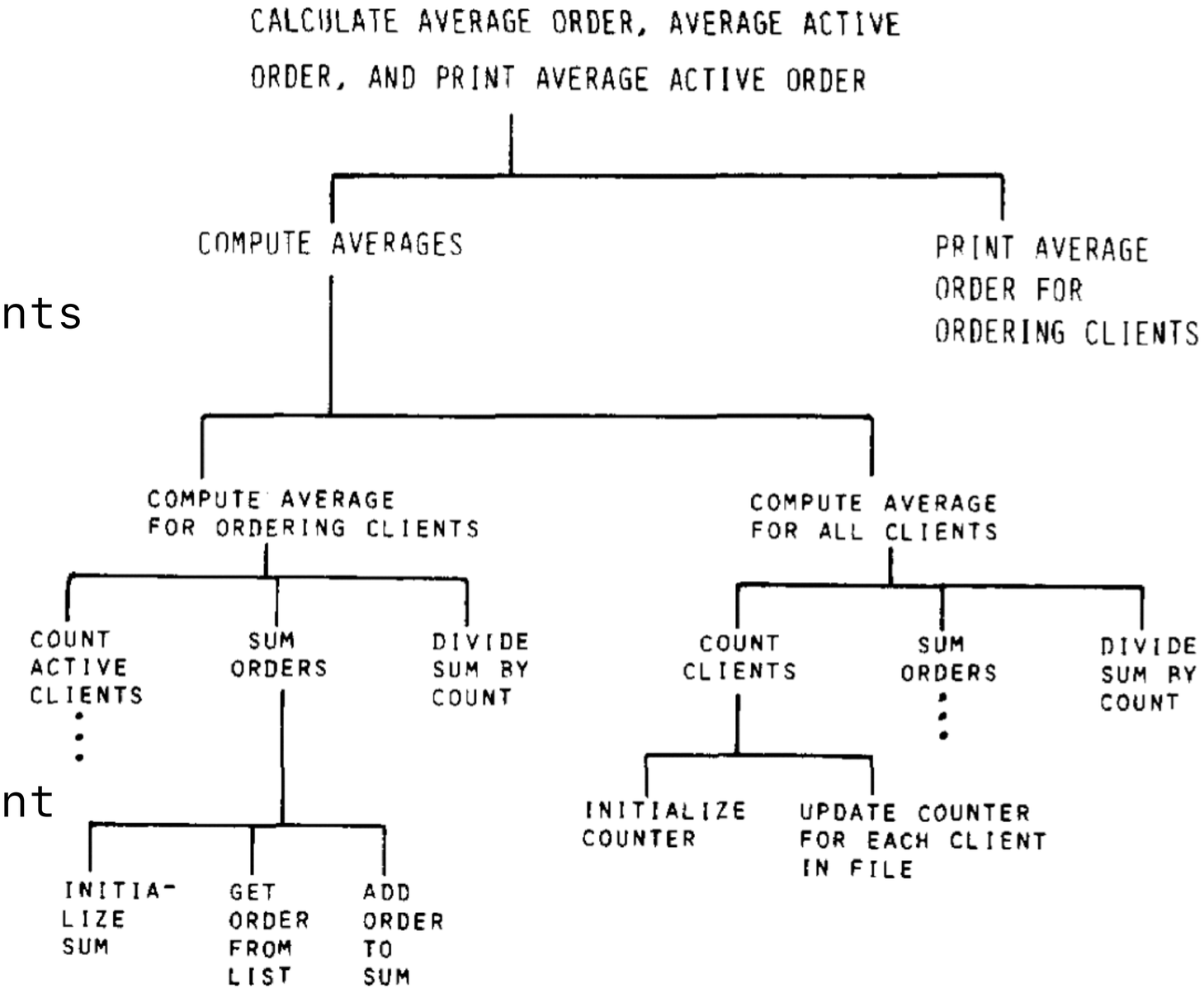
```
count_clients = 0
total_orders = 0
inactive_clients = 0
order_rec = read(order_file)
while order_rec.id != 999999:
  sum_orders()
active_clients =
  count_clients - inactive_clients
client_avg =
  total_orders / count_clients
active_avg =
  total_orders / active_clients
print(active_avg)

fn sum_orders():
  count_clients += 1
  total_orders += order_rec.quant
  if order_rec.quant == 0:
    inactive_clients += 1
  order_rec = read(order_file)
```

Control-flow representation

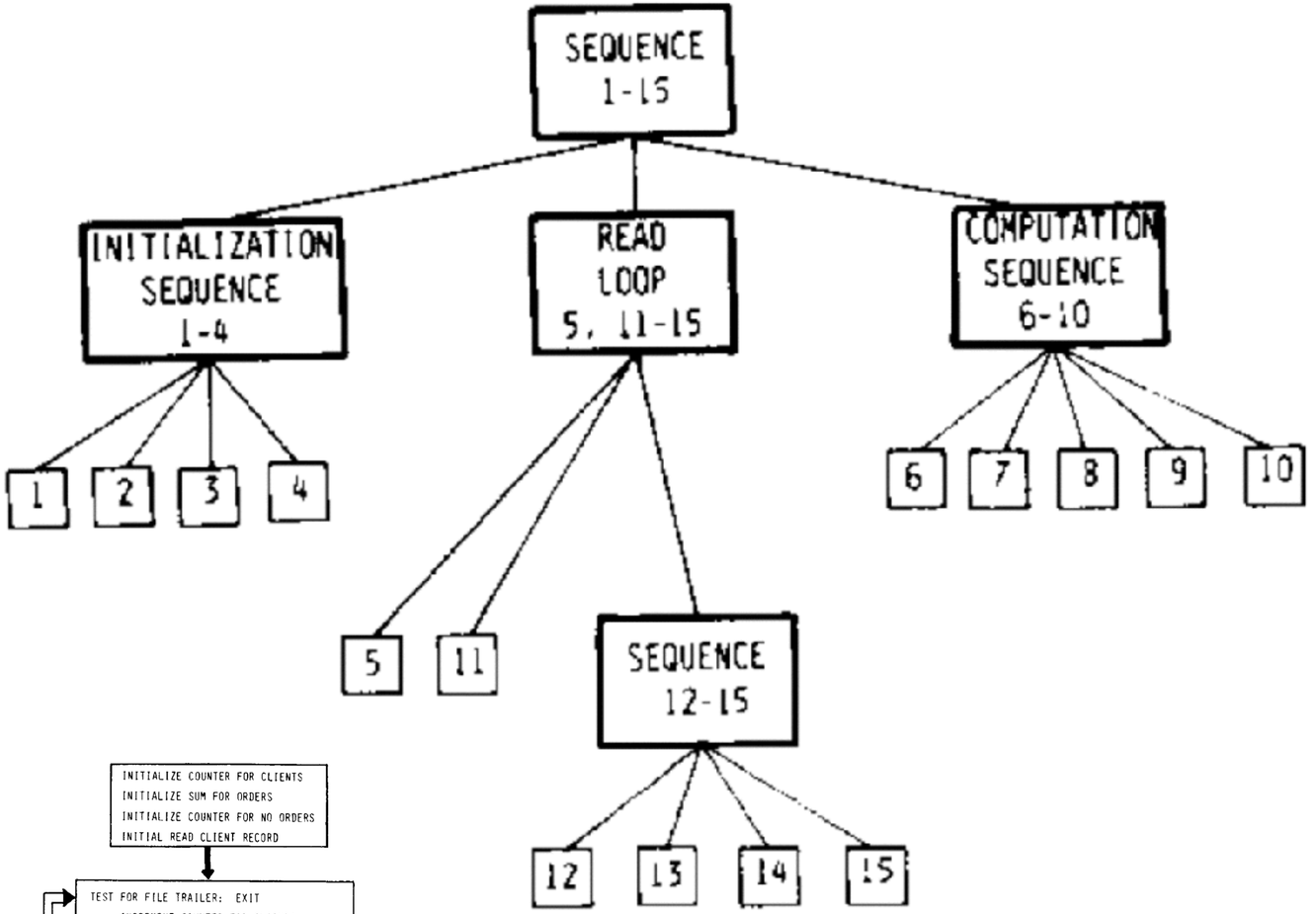


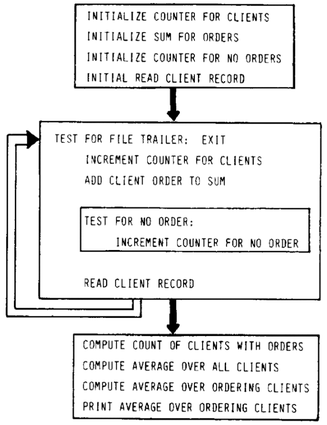
```
count_clients = 0
total_orders = 0
inactive_clients = 0
order_rec = read(order_file)
while order_rec.id != 999999:
  sum_orders()
active_clients =
  count_clients - inactive_clients
client_avg =
  total_orders / count_clients
active_avg =
  total_orders / active_clients
print(active_avg)

fn sum_orders():
  count_clients += 1
  total_orders += order_rec.quant
  if order_rec.quant == 0:
    inactive_clients += 1
  order_rec = read(order_file)
```

Functional representation

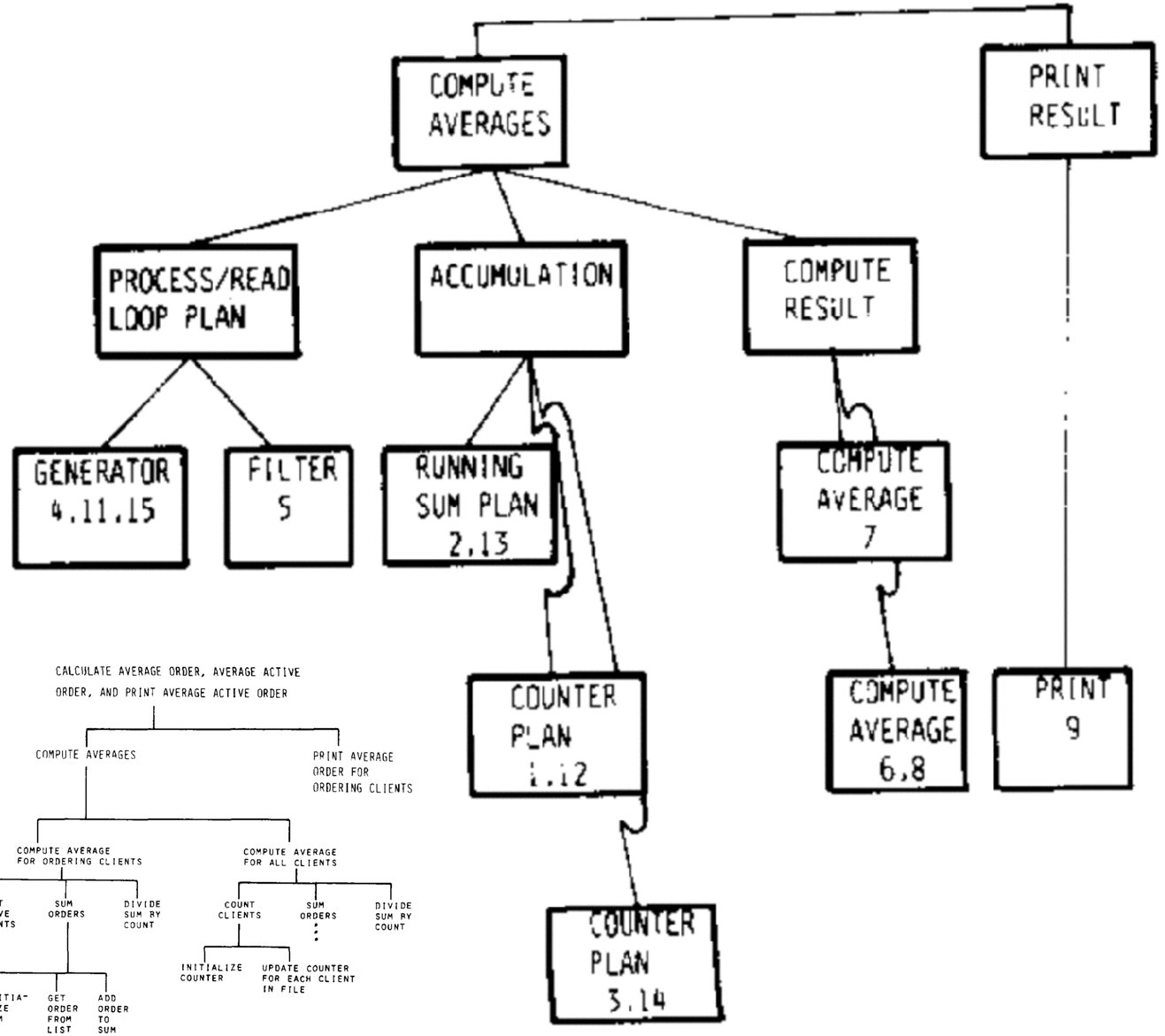


```
count_clients = 0
total_orders = 0
inactive_clients = 0
order_rec = read(order_file)
while order_rec.id != 999999:
  sum_orders()
active_clients =
  count_clients - inactive_clients
client_avg =
  total_orders / count_clients
active_avg =
  total_orders / active_clients
print(active_avg)

fn sum_orders():
  count_clients += 1
  total_orders += order_rec.quant
  if order_rec.quant == 0:
    inactive_clients += 1
  order_rec = read(order_file)
```





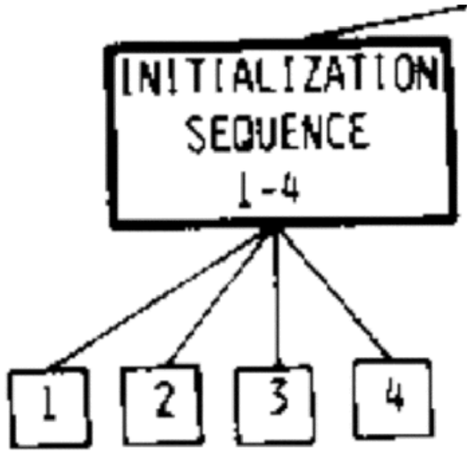
```
count_clients = 0
total_orders = 0
inactive_clients = 0
order_rec = read(order_file)
while order_rec.id != 999999:
  sum_orders()
active_clients =
  count_clients - inactive_clients
client_avg =
  total_orders / count_clients
active_avg =
  total_orders / active_clients
print(active_avg)

fn sum_orders():
  count_clients += 1
  total_orders += order_rec.quant
  if order_rec.quant == 0:
    inactive_clients += 1
  order_rec = read(order_file)
```



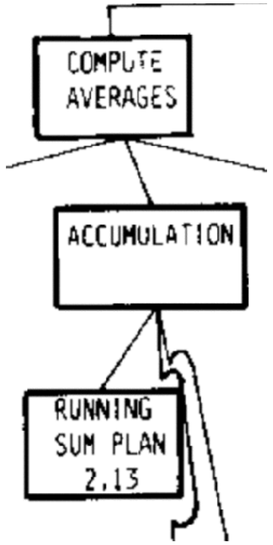
Representational hypotheses are tested with primes

```
 2: total_orders = 0

 4: order_rec = read(order_file)

13: total_orders += order_rec.quant
```

Control flow



Functional



Participant 1:

```
total_orders = 0
```

Y/N?

```
order_rec =
  read(order_file)
```

Y/N?

Participant 2:

```
total_orders = 0
```

Y/N?

```
total_orders +=
  order_rec.quant
```

Y/N?

Participants more likely had a control-based mental representation of the program

- Control-primed targets were 105ms faster for recognition than function-primed targets
- This gap grew to 219ms for upper quartile participants (based on comprehension questions)
- Later experiments suggested that representation shifted to functional given more comprehension time and a deeper task

Nancy Pennington. “Stimulus structures and mental representations in expert comprehension of computer programs” 1987

Programming languages as notation

Non-programmers find some programming names unintuitive

	Programmer	Non-programmer
Booleans	boolean	condition
Strings	string	characters
Modulus	x = 7 % 2	x = remainder of 7 / 2
Inequality	x != y	x unequal y
Arrays	value x[]	value series x

Stefik and Siebert. “An Empirical Investigation into Programming Language Syntax” (2013)

Dimensions of syntactic variation

Objective-C

```
NSMutableDictionary *mutableDictionary =
  [NSMutableDictionary dictionaryWithCapacity:2];
[mutableDictionary setObject:@"value1" forKey:@"key1"];
[mutableDictionary setObject:@"value2" forKey:@"key2"];
[mutableDictionary removeObjectForKey:@"key1"];
```

Rust

```
let mut map = HashMap::with_capacity(2);
map.insert("key1", "value1");
map.insert("key2", "value2");
map.remove("key1");
```

Scheme

```
(define ht (make-hash-table))
(hash-table-set! ht "key1" "value1")
(hash-table-set! ht "key2" "value2")
(hash-table-delete! ht "key1")
```

Postfix vs. prefix vs. infix

Positional vs. named

Implicit vs. explicit

Uniform vs. specialized

Dimensions of syntactic variation

Conditionals

```
if x < 0:
   y = -x
else:
   y = x
```

```
y = -x if x < 0 else x
```

```
y = x < 0 ? -x : x;
```

```
y = if x < 0 then -x else x
```

Lambdas

```
inc = 1
```

Python
```
lambda n: n + inc
```

Haskell
```
\n -> n + inc
```

Ruby
```
->(n) { n + inc }
```

Javascript
```
n => n + inc
```

APL
```
{ω+inc}
```

Smalltalk
```
[:n | n+inc]
```

Objective-C
```
^(int n){return n+inc;}
```

Rust
```
move |n| n + inc
```

C++
```
[=](int n) {return n+inc;}
```

“You can see us avoiding blowing the [strangeness] budget in Rust with many of our syntactic choices. We chose to stick with curly braces, for example, because one of our major target audiences, systems programmers, is currently using a curly brace language. Instead, we spend this strangeness budget on our major, core feature: ownership and borrowing.”

https://steveklabnik.com/writing/the-language-strangeness-budget/

Goto makes for tough reading

"Our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible."

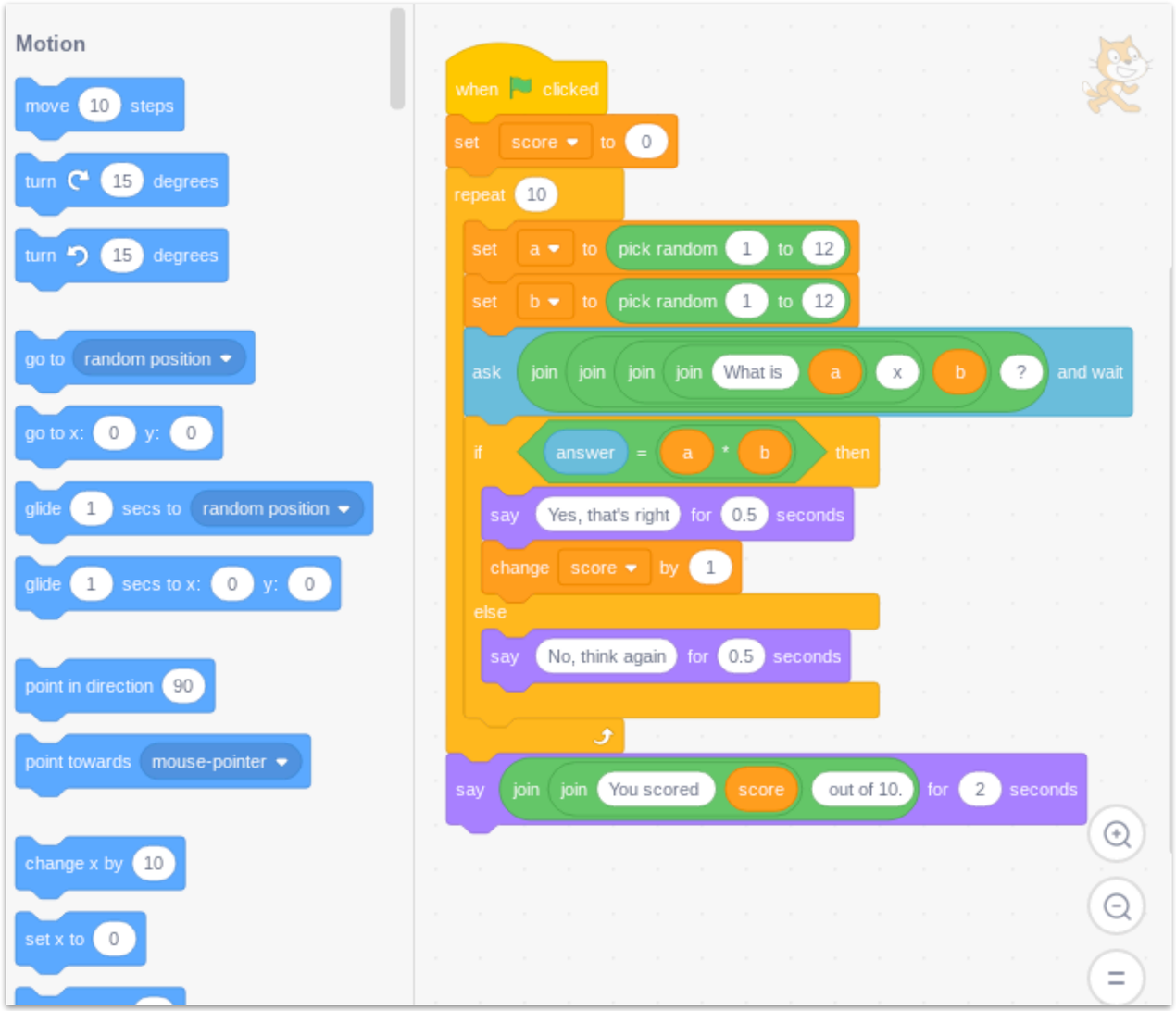
```
if x > 0 {
   if y == 1 {
      print("A")
   } else {
      print("B")
   }
   print("C")
} else {
   print("D");
}
print("E");
```

```
   if x <= 0 goto D
   if y != 1 goto B
   print("A")
   goto C
B: print("B")
C: print("C")
   goto E
D: print("D")
E: print("E")
```

Edsger Dijkstra. "Go To Statement Considered Harmful" 1968

Structure editors prevent syntax errors

Scratch



Hazel

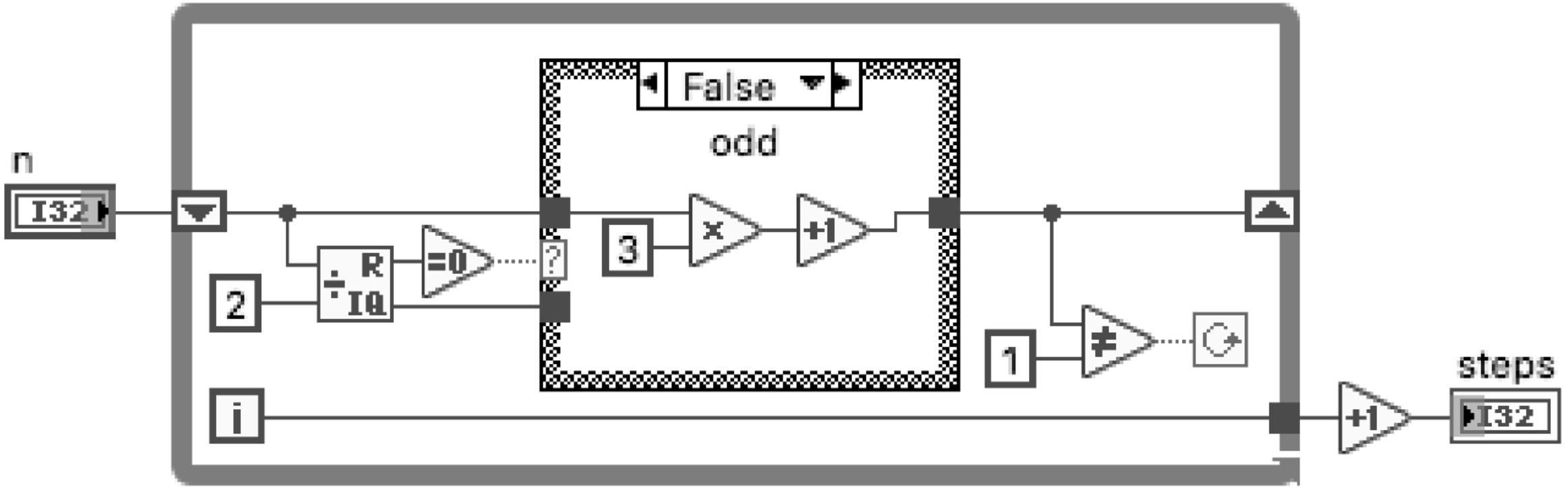
```
let incr = fun (x : Int) -> x+1 in
let x = incr(○) in
x * 2
```

```
≡   (○ + 1) * 2
```

`EXP` / Expression hole — Fillable by any expression of type `Int`

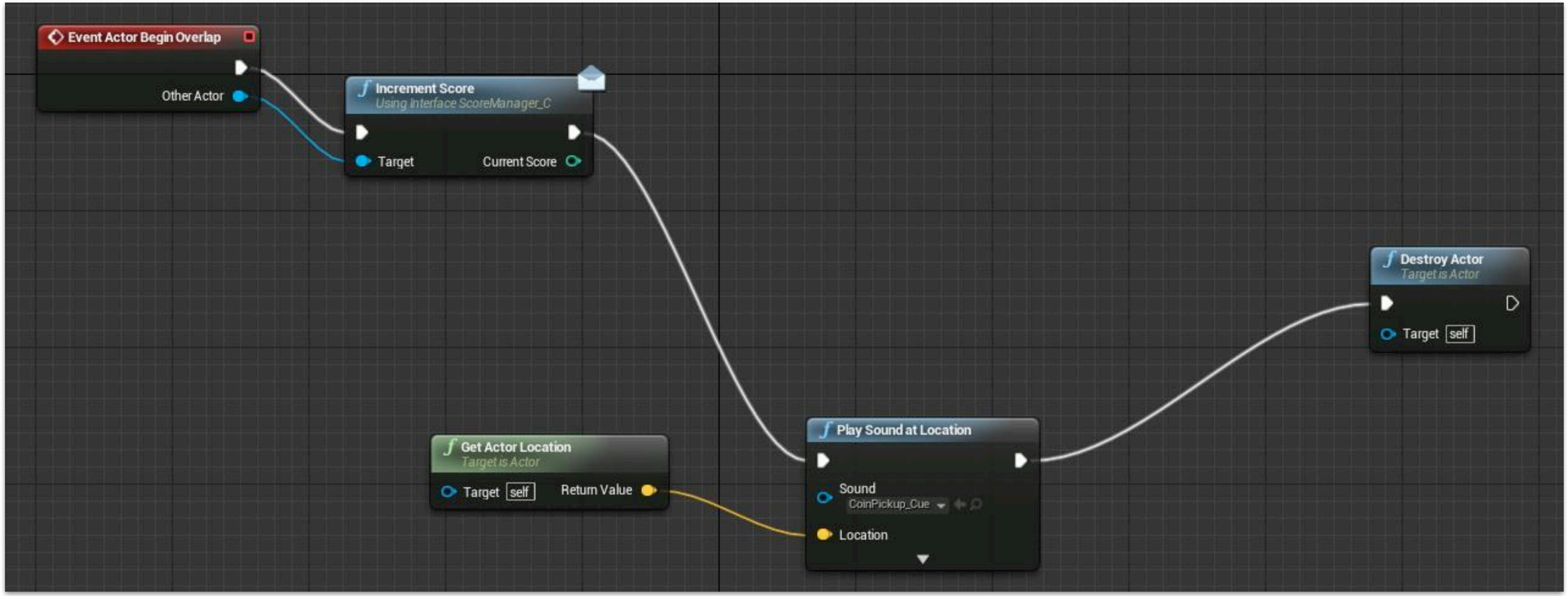
```
type DailySighting = (
  day=Int, fox=Bool, deer=Bool, rabbit=Bool,
  bear=Bool, owl=Bool, raccoon=Bool, wolf=Bool, moose=Bool) in
let sightings : [DailySighting] = [
  (1, ☑, ☐, ☑, ☐, ☐, ☑, ☐, ☐),
  (2, ☐, ☑, ☑, ☐, ☑, ☐, ☐, ☐),
  (3, ☑, ☐, ☐, ☐, ☑, ☐, ☐, ☐),
  (4, ☐, ☐, ☑, ☐, ☐, ☐, ☐, ☐),
  (5, ☑, ☐, ☐, ☐, ☐, ☑, ☐, ☐),
  (6, ☐, ☑, ☐, ☑, ☐, ☐, ☐, ☑),
  (7, ☐, ☐, ☑, ☐, ☐, ☐, ☑, ☐),
  (8, ☑, ☐, ☑, ☐, ☐, ☐, ☐, ☐),
  (9, ☐, ☑, ☐, ☐, ☑, ☐, ☐, ☐)
```

Node-link syntax permits “point-free” code and diagrammatic reasoning

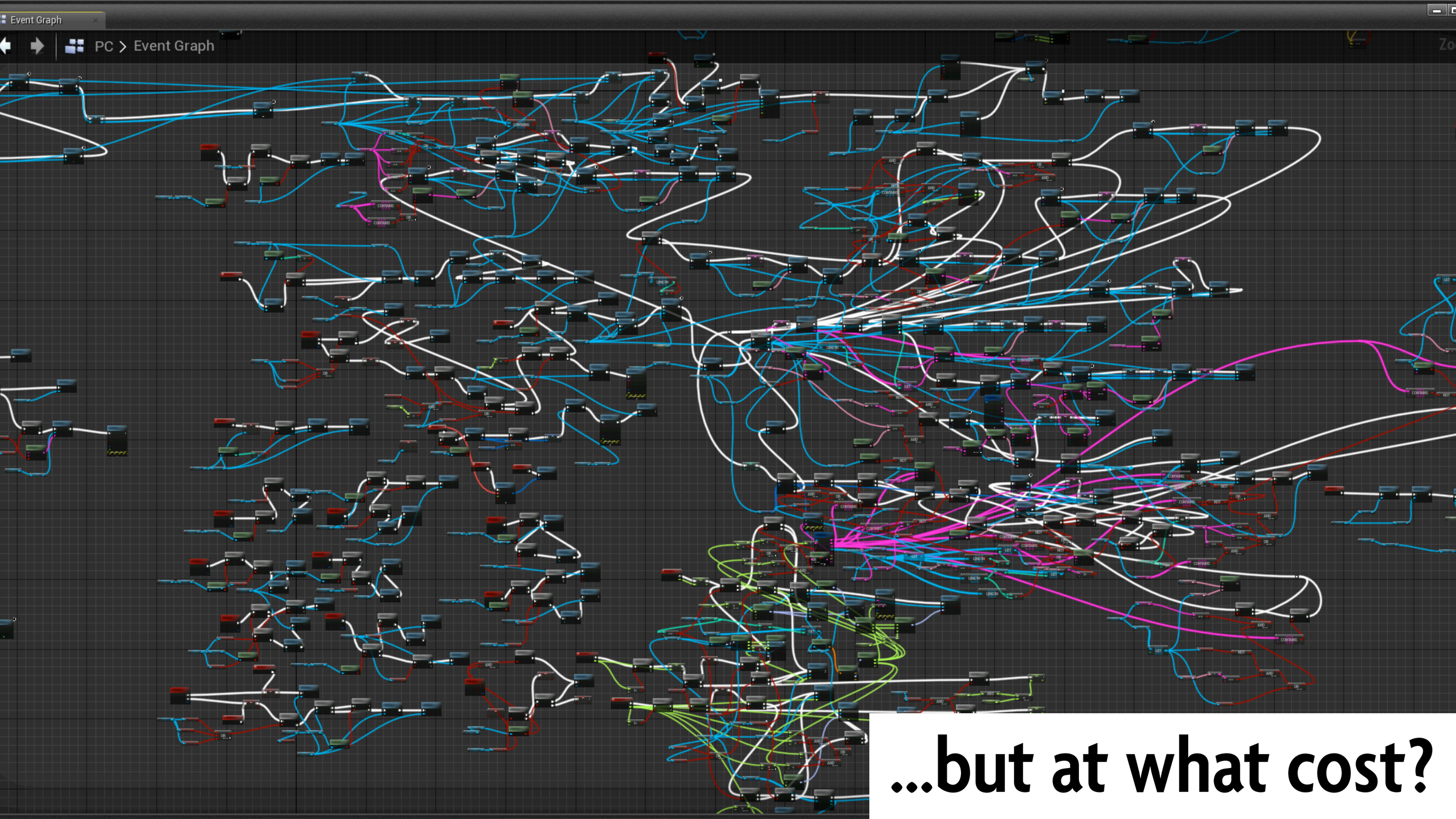


Collatz conjecture implemented in LabView

Node-link syntax permits “point-free” code and diagrammatic reasoning



Implementing a collectible in Unreal Engine 4 Blueprints


Event Graph
PC > Event Graph
...but at what cost?

IDEs as hypertext reading augmentation systems

Are code editors and IDEs different any more?

- Once upon a time, Eclipse and Visual Studio were quite different than Vim and Emacs
- Language Server Protocol (2016) standardized IDE features over the last decade
- Today, you have autocomplete in Vim and Emacs keybindings in VSCode

Nodes

Content

Finding

Links

Navigation

Let's do a hypertext analysis of IDEs!

Viewing

Annotation

Versioning

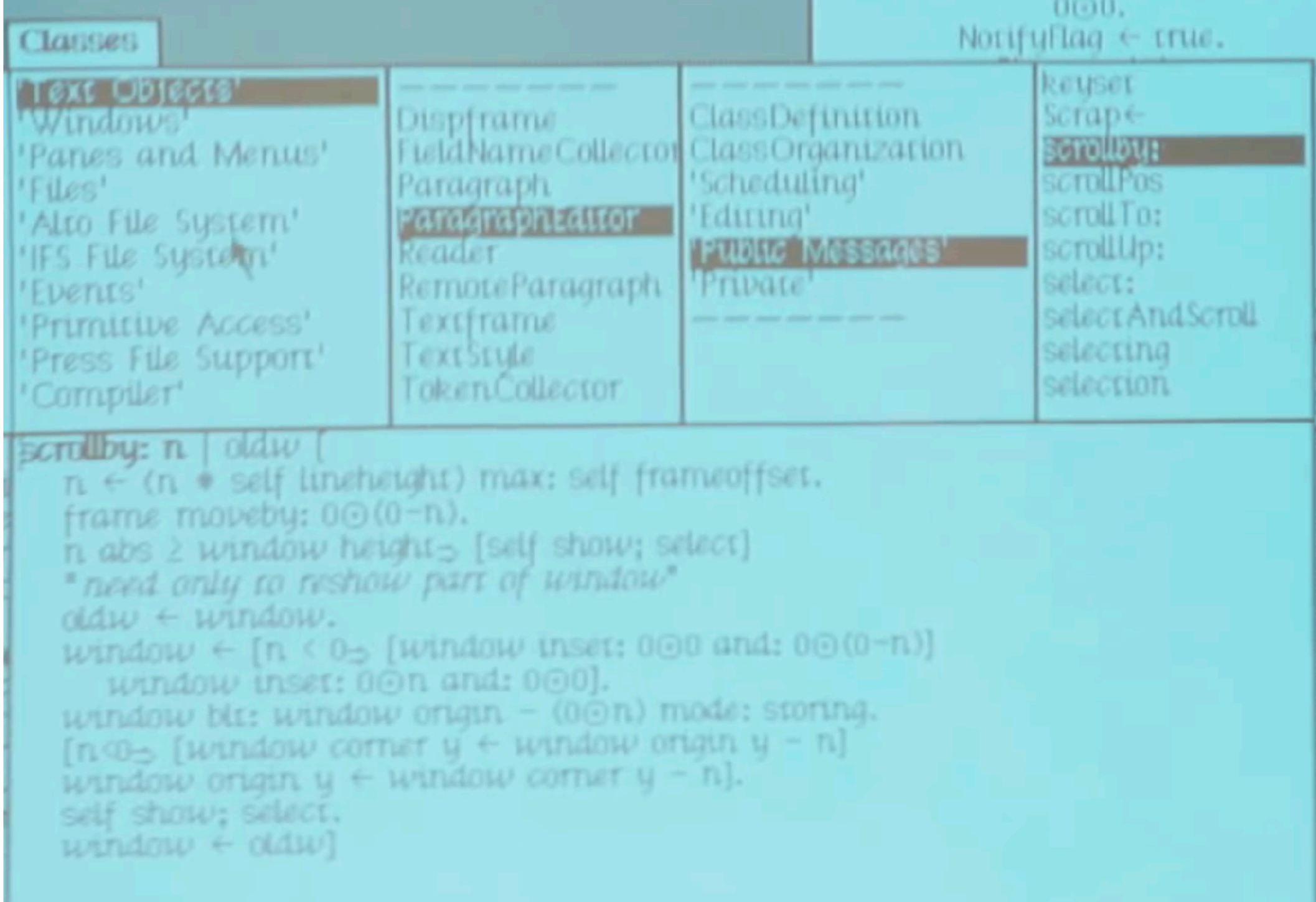
Authoring

Collaboration

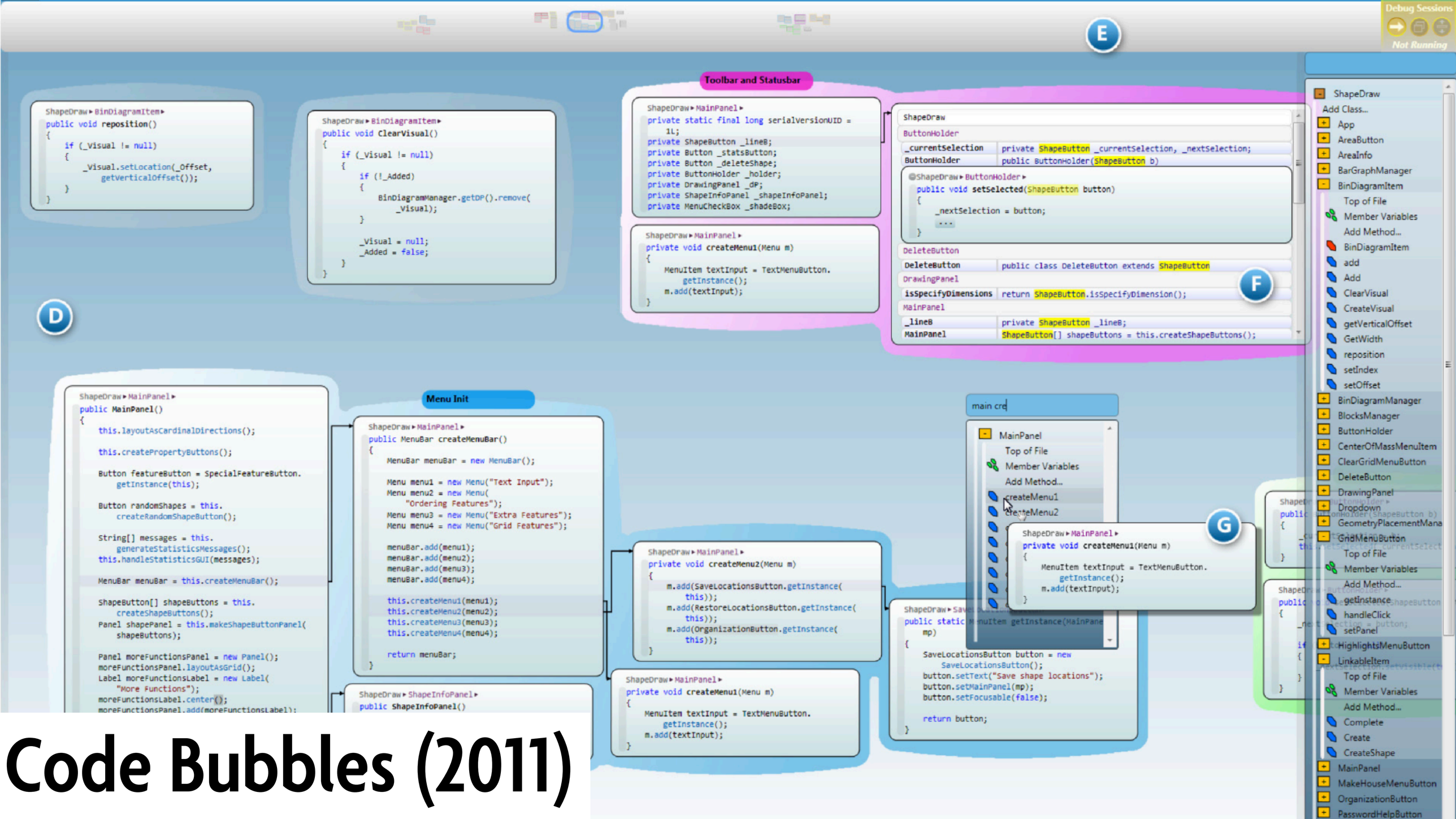
Let's do a hypertext analysis of IDEs!

- Nodes: files
- Links: variable/class references
- Content: source code
- Navigation:
  - Jump to definition
  - File explorer
- Finding
  - Find all references
  - Fulltext search / grep
- Viewing: windows
  - Managed by IDE or OS
- Authoring: typing + LLMs
- Collaboration: VCS
- Annotation: comments
  - Jump to definition
  - File explorer
- Versioning: commits

Smalltalk pioneered the object-oriented IDE in the 1970s



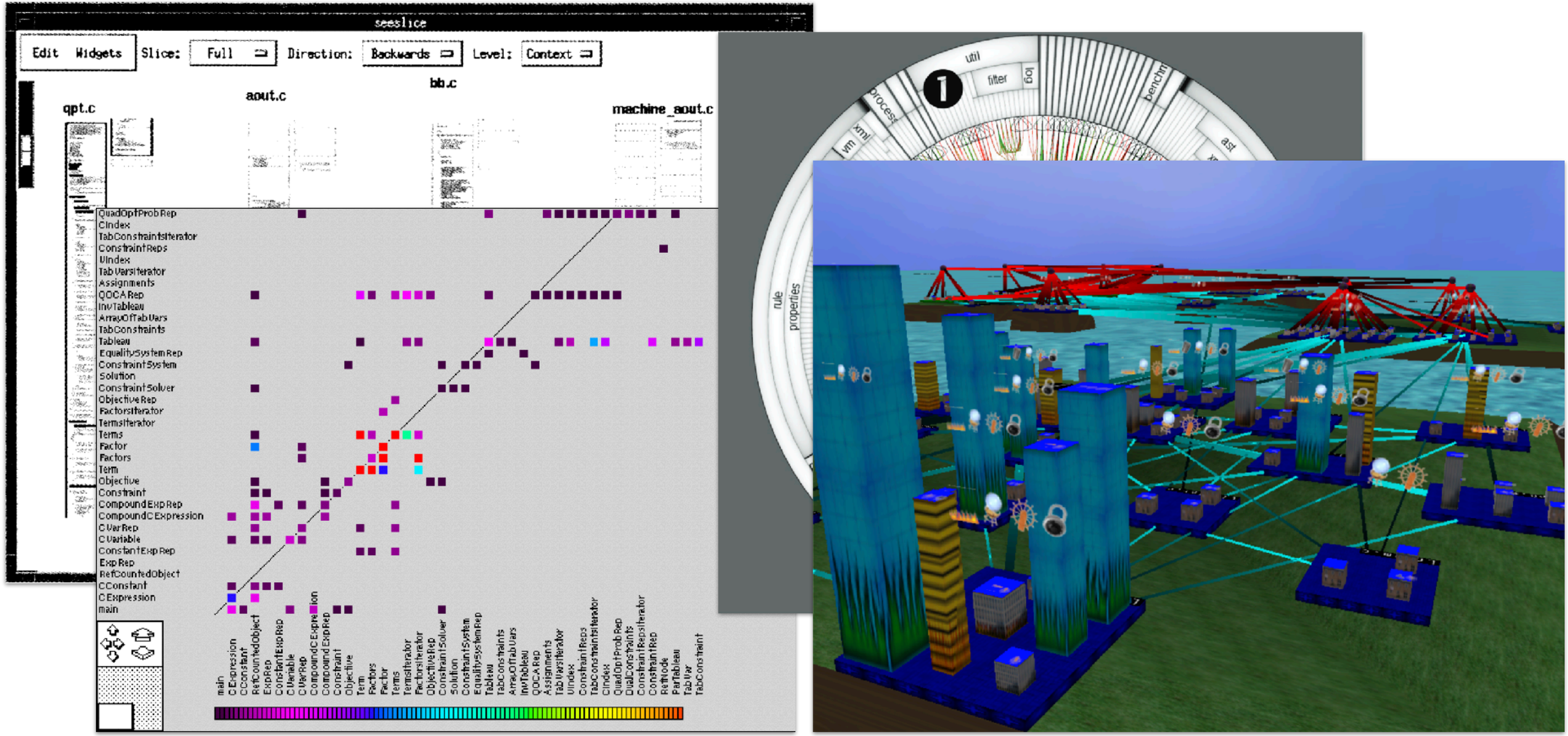
[Glamorous Toolkit demo]



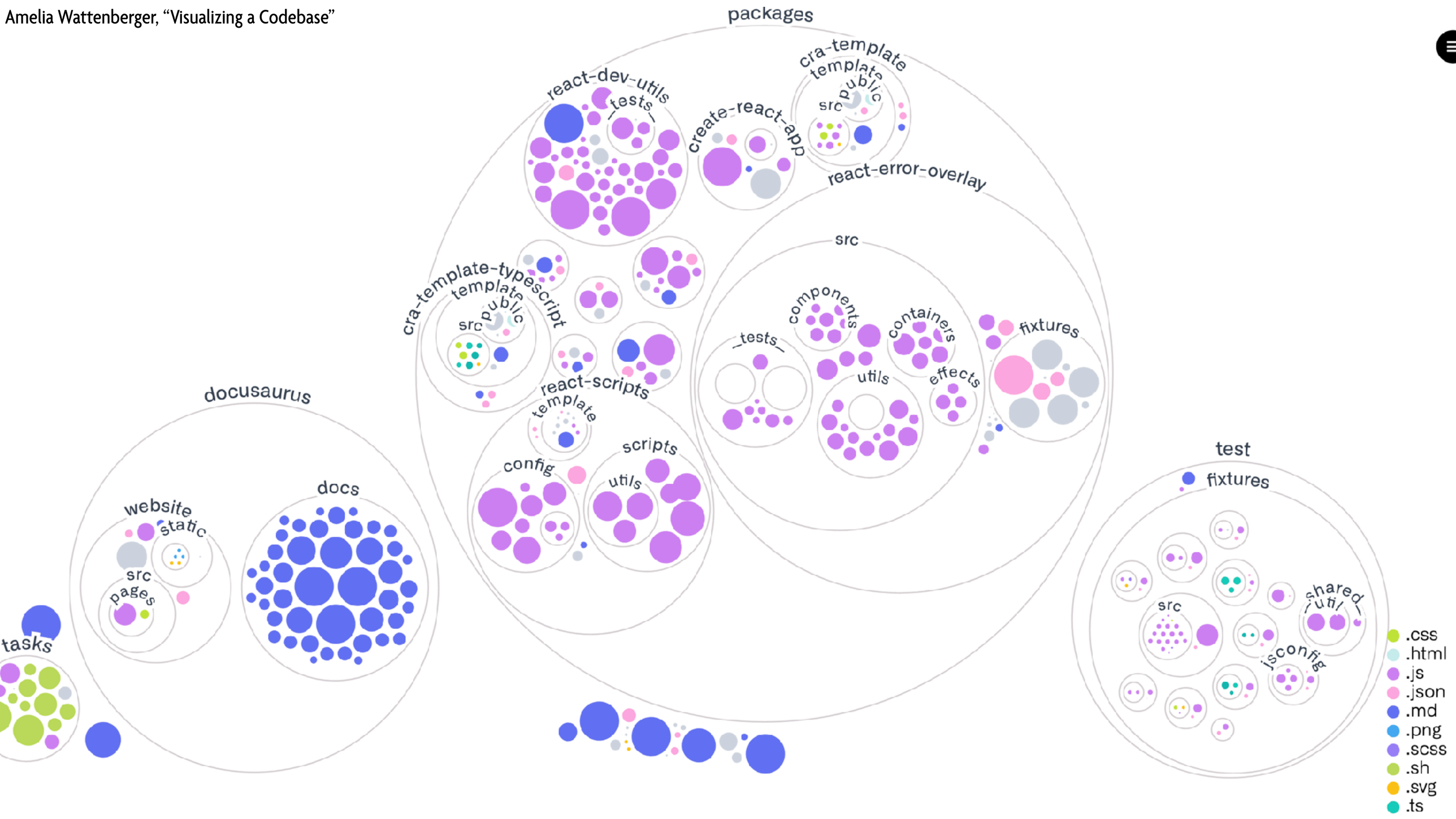
Code Bubbles (2011)

Devtools as domain-specific visualization systems

Many software visualizations have been tried



Amelia Wattenberger, “Visualizing a Codebase”

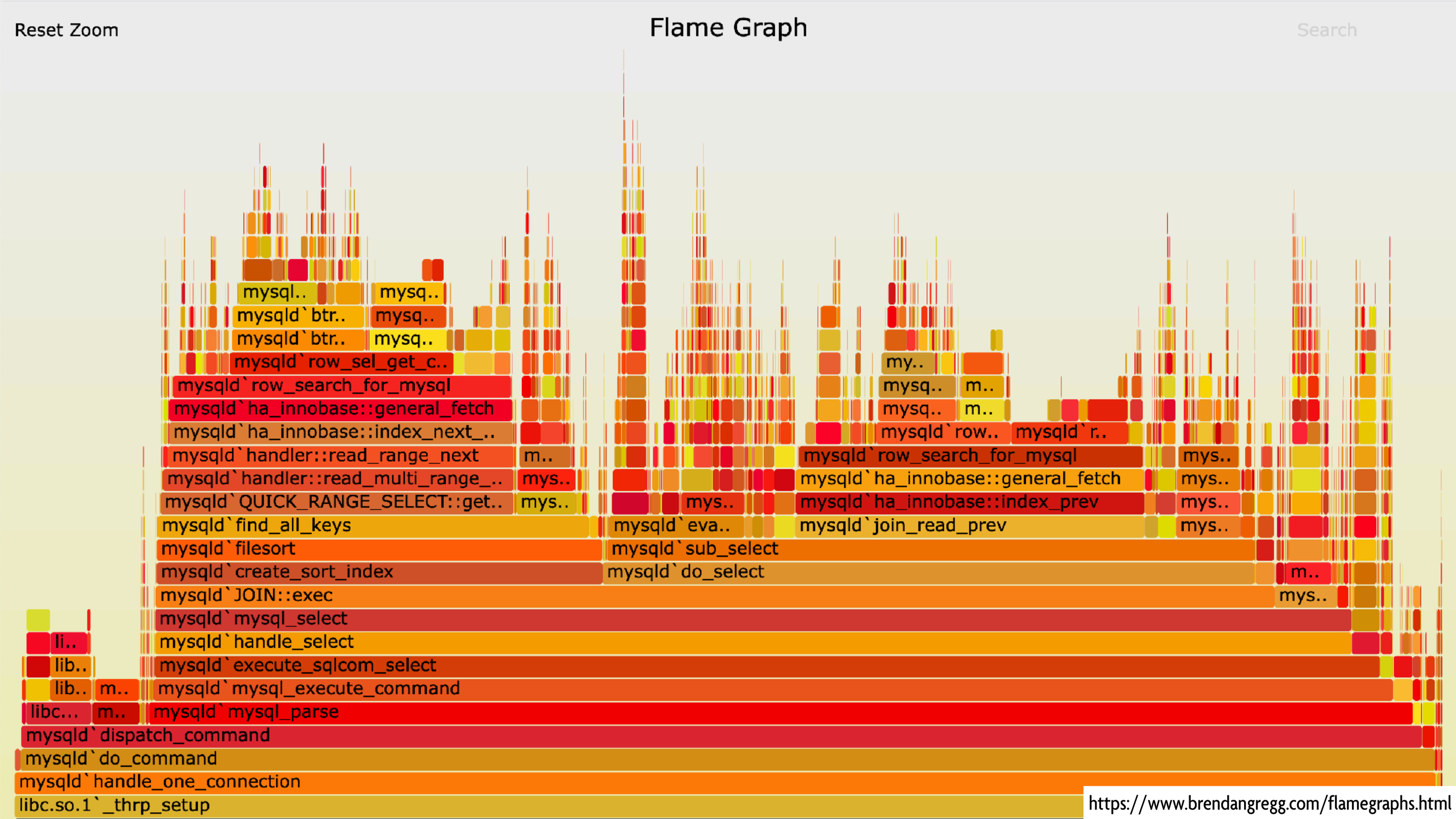


“None of today's mainstream programming environments feature visualization. None of the standard programming tools used today uses visualization in anything but the crudest way. The various experiments with visualization, even when they appeared in production environments, have failed; successor environments did not feel it was worth reproducing visualization tools. Programmers don't use visualization tools and don't seem to want to use them. […]

The visualization systems that have been developed address “generic” understanding problems. They look at the program structure from the generic view of the class hierarchy or the call hierarchy. They look at dynamic understanding from the generic view of what basic blocks are being executed, what resources are being used, what is being allocated, or what states the threads are in. But the reality of software understanding is that programmers ask specific questions, not generic ones. […]

Generic tools provide generic answers.”

— Steve Reiss, “The Paradox of Software Visualization” (2005)


Reset Zoom
Flame Graph
Search
mysql..
mysq..
mysqld`btr..
mysq..
mysqld`btr..
mysq..
mysqld`row_sel_get_c..
my..
mysqld`row_search_for_mysql
mysq..
m..
mysqld`ha_innobase::general_fetch
mysq..
m..
mysqld`ha_innobase::index_next_..
mysqld`row..
mysqld`r..
mysqld`handler::read_range_next
m..
mysqld`row_search_for_mysql
mys..
mysqld`handler::read_multi_range_..
mys..
mysqld`ha_innobase::general_fetch
mys..
mysqld`QUICK_RANGE_SELECT::get..
mys..
mys..
mysqld`ha_innobase::index_prev
mys..
mysqld`find_all_keys
mysqld`eva..
mysqld`join_read_prev
mys..
mysqld`filesort
mysqld`sub_select
mysqld`create_sort_index
mysqld`do_select
m..
mysqld`JOIN::exec
mys..
mysqld`mysql_select
li..
mysqld`handle_select
lib..
mysqld`execute_sqlcom_select
lib..
m..
mysqld`mysql_execute_command
libc...
m..
mysqld`mysql_parse
mysqld`dispatch_command
mysqld`do_command
mysqld`handle_one_connection
libc.so.1`_thrp_setup
https://www.brendangregg.com/flamegraphs.html

Current Selection

Area Selection

CPU by thread | CPU by process

Flat | Tree

Process Name	Thread Name	Count COUNT 3154	CPU Time ↓ SUM 573.4ms	CPU Time % SUM 100.00%	CPU Time AVG 181.8µs
com.android.systemui	RenderThread	256	98.01ms	17.09%	382.9µs
com.android.vending	RenderThread	182	69.05ms	12.04%	379.4µs
/system/bin/traced_probes	traced_probes	26	52.96ms	9.24%	2.037ms
/vendor/bin/hw/android.hardware.graphics.composer@2.1-service	HwBinder:642_1	61	33.89ms	5.91%	555.6µs
com.google.android.googlequicksearchbox:search	HeapTaskDaemon	12	30.54ms	5.33%	2.545ms
/system/bin/surfaceflinger	surfaceflinger	74	30.12ms	5.25%	407.1µs
com.android.systemui	ndroid.systemui	85	23.49ms	4.10%	276.3µs
irq/760-synaptl	irq/760-synaptl	340	20.76ms	3.62%	61.05µs

Python 3.11
known limitations

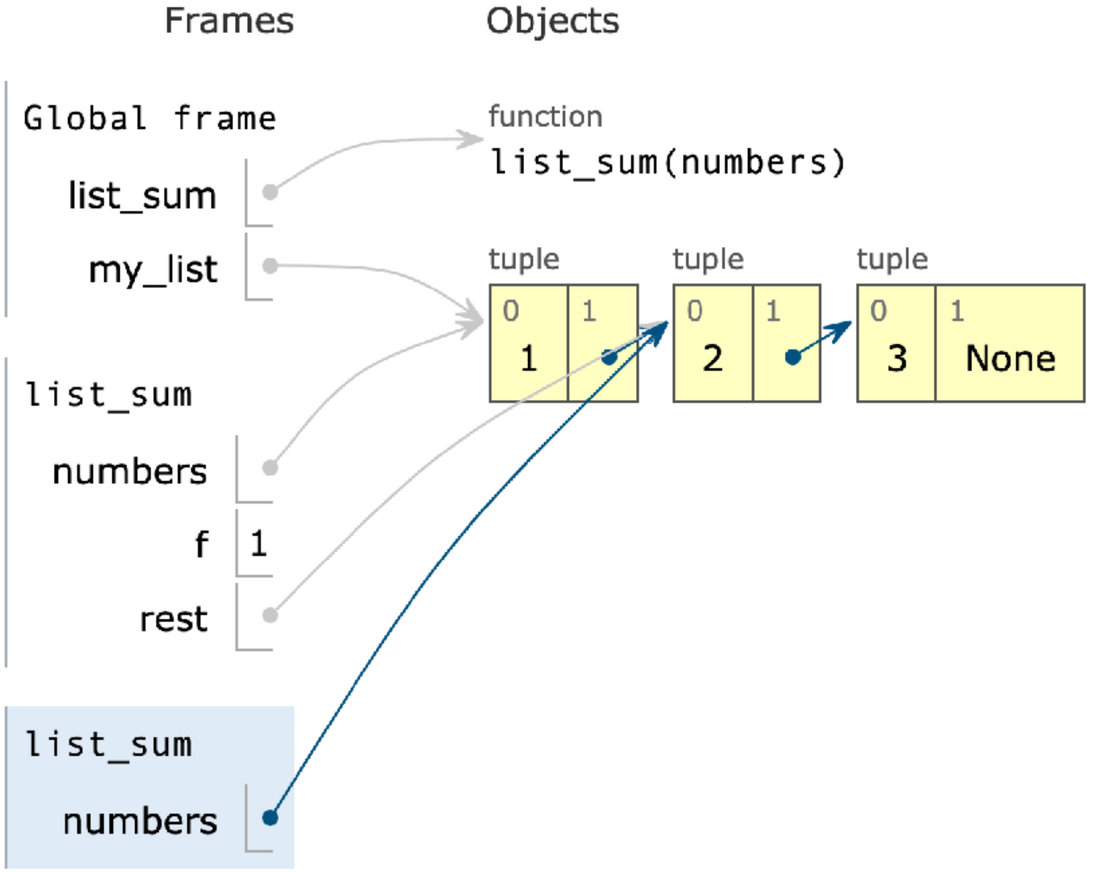
```
1  def list_sum(numbers):
2      if not numbers:
3          return 0
4      else:
5          (f, rest) = numbers
6          return f + list_sum(rest)
7
8  my_list = (1, (2, (3, None)))
9  total = list_sum(my_list)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 10 of 22



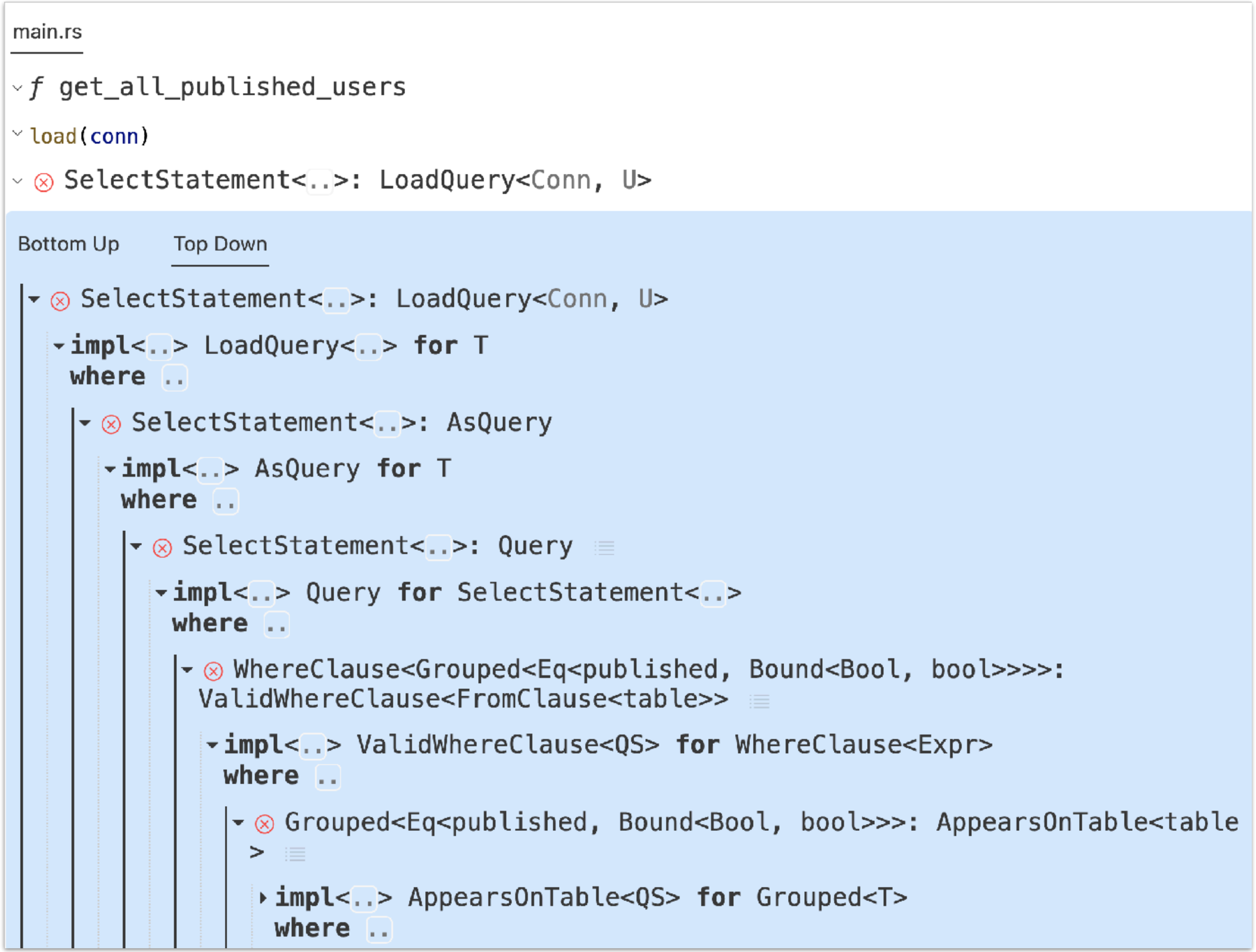
```
let mut v: Vec<i32> = vec![1, 2, 3];

let num: &i32 = &v[2];

println!("v[2] = {}", *num);

v.push(4);
```

v	↥	+R	+W	+O

v	→	R	~~W~~	~~O~~
num	↥	+R	–	+O
*num	↥	+R	–	–

v	↻	R	+W	+O
num	↴	~~R~~	–	~~O~~
*num	↴	~~R~~	–	–

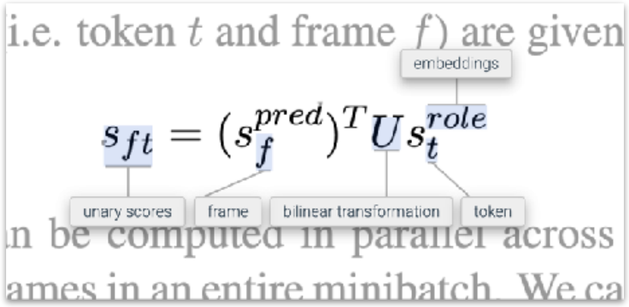
v	↴	~~R~~	~~W~~	~~O~~


main.rs
ƒ get_all_published_users
load(conn)
⊗ SelectStatement<..>: LoadQuery<Conn, U>
Bottom Up
Top Down
⊗ SelectStatement<..>: LoadQuery<Conn, U>
impl<..> LoadQuery<..> for T
where ..
⊗ SelectStatement<..>: AsQuery
impl<..> AsQuery for T
where ..
⊗ SelectStatement<..>: Query
impl<..> Query for SelectStatement<..>
where ..
⊗ WhereClause<Grouped<Eq<published, Bound<Bool, bool>>>>:
ValidWhereClause<FromClause<table>>
impl<..> ValidWhereClause<QS> for WhereClause<Expr>
where ..
⊗ Grouped<Eq<published, Bound<Bool, bool>>>: AppearsOnTable<table
>
impl<..> AppearsOnTable<QS> for Grouped<T>
where ..

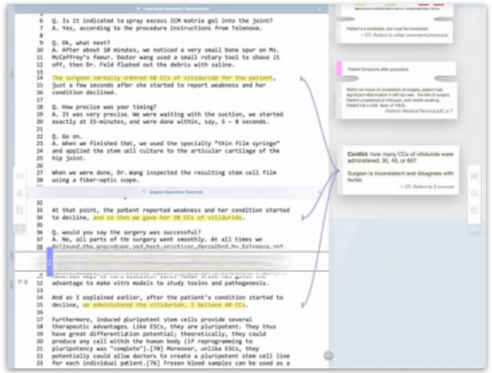
Course recap

I hope that you have a broader view on the present and future of TfT

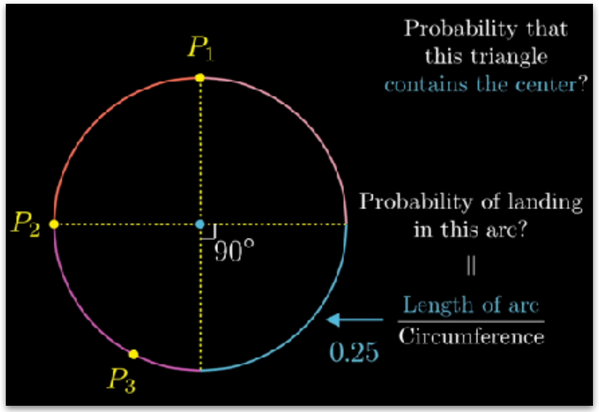
Flashcards



Reading

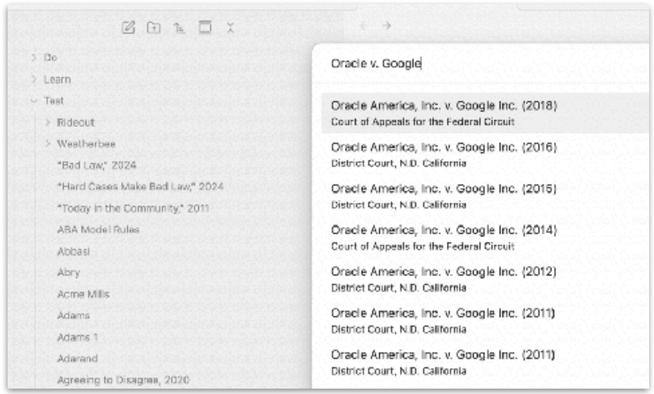


Note-taking

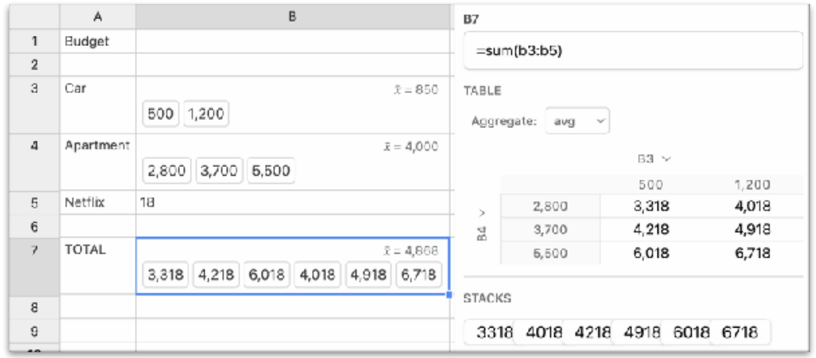


Technical communication

Data visualization



Information management



Programming

I hope that you understand how to think systematically and scientifically about TfT

- **Capabilities**: how do the brain's primitive components work, and how do they support higher-level cognition?
- **Tasks**: what is the shape of knowledge work? How do people decompose tasks into manageable pieces?
- **Representations**: what makes one form of information more effective than another at facilitating a cognitive task?

“I think that AI will be able to do math at a superhuman level in my lifetime, and that there is a good chance this will happen relatively soon. […]

As a result, I may be among the last generation of PhDs who are able to prove a difficult original mathematical result—where, by difficult, I mean one which requires introducing a new and different way of thinking about a given problem—purely using their own intellectual efforts.”

— A CS prof last month

(I hope you know now that)

Everything is a technology!!!

language, scripts, alphabets, poetry, paper, whitespace, books (codices), margins, headings, indexes, printers, math notation, card catalogs, filing cabinets, maps, graphs, arrows, diagrams, animations...

Let's hear from you!

- What is one system, idea, assignment, etc. in this course that resonated with you?
  - i.e., it was memorable, interesting, thought-provoking
- How did you feel about the balance of focus on history, psychology, and software? The balance of breadth vs. depth? Level of effort put into the course?
- What is one thing you would change about the course?
  - Stuff we're already working on: labs, rubrics