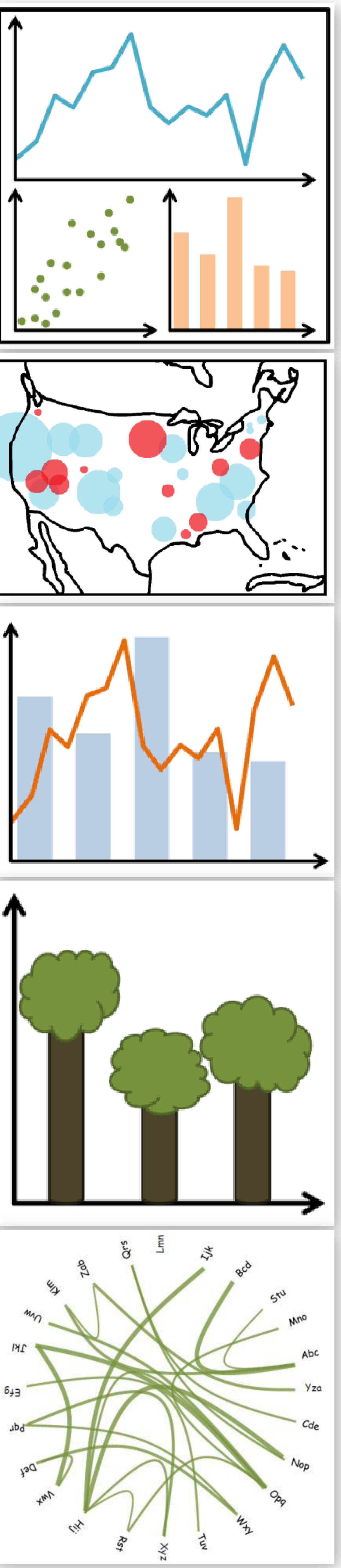


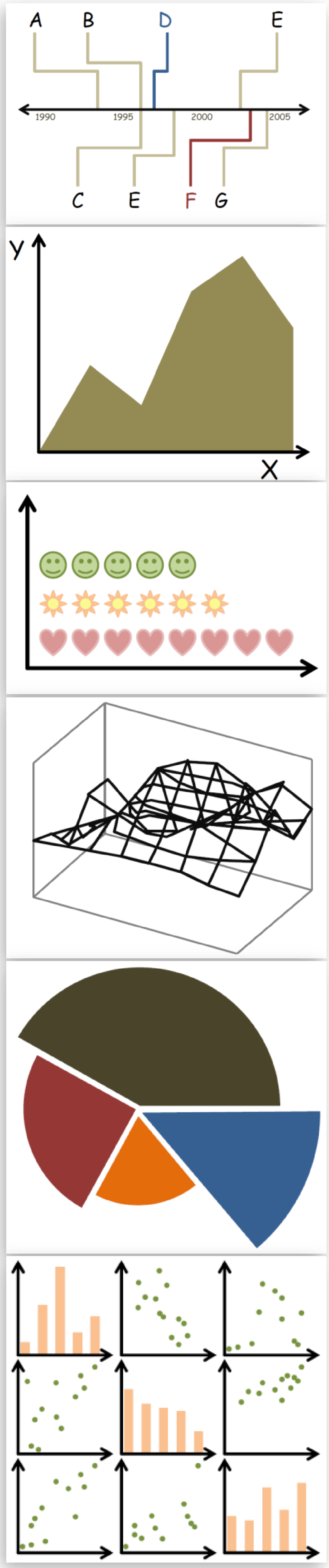
Lecture 3: JS Development, Projects

CS 7250

Spring 2021

Prof. Cody Dunne

Northeastern University



Slides and inspiration from Michelle Borkin, Krzysztof Gajos, Hanspeter Pfister, Miriah Meyer, Jonathan Schwabish, and David Sprague

Checking In

Staff Introductions

Cody Dunne
Assistant Professor
Instructor

Girik Malik
TA

David Saffo
TA

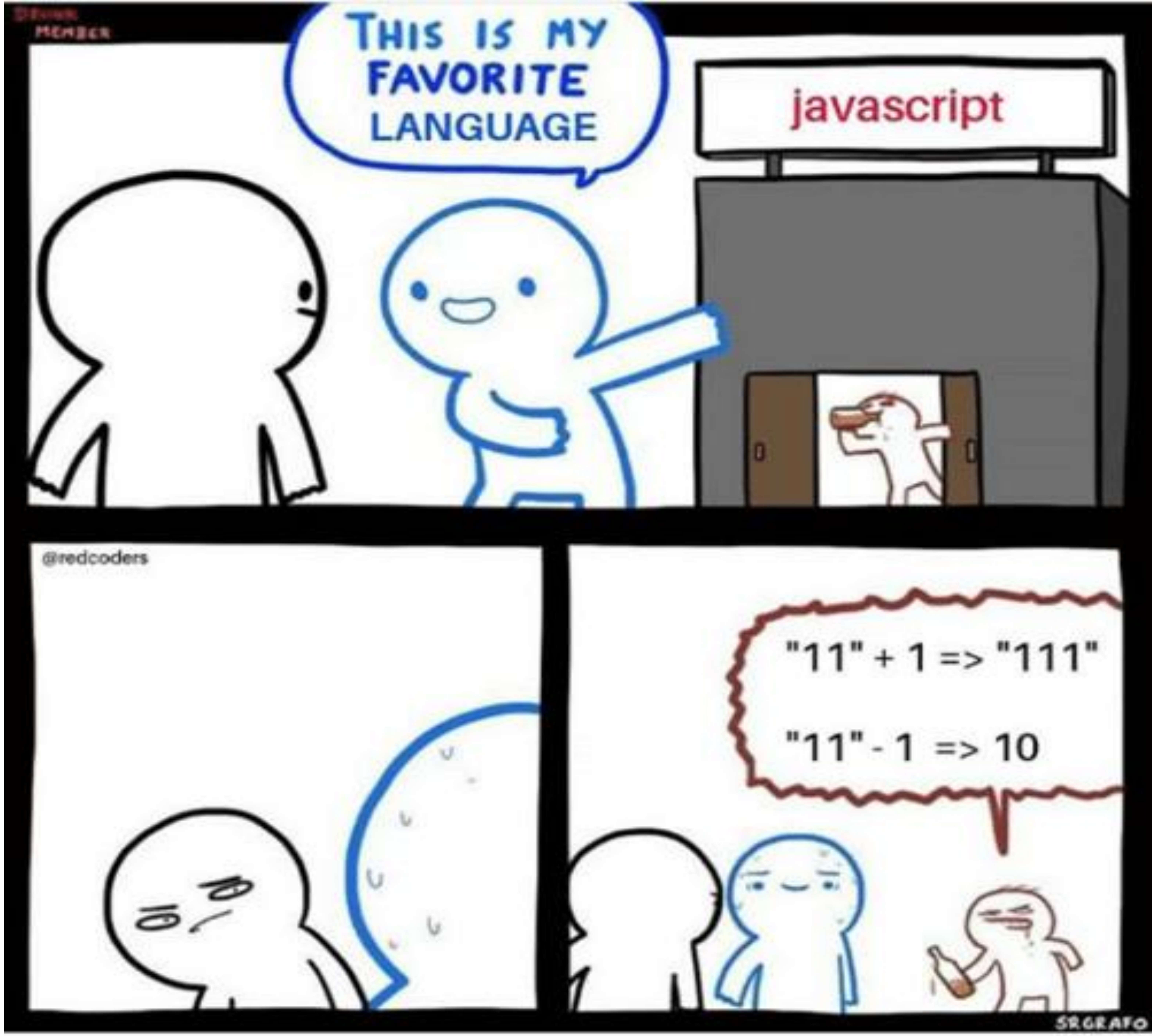
Sara Di Bartolomeo
TA

Sophia Gunzberg
Service-Learning TA

Previously, on CS 7250…

JavaScript Development

Slides and inspiration from Sara Di Bartolomeo


MEMBER
THIS IS MY FAVORITE LANGUAGE
javascript
@redcoders
"11" + 1 => "111"
"11" - 1 => 10
SRGRAFO

JavaScript is **bad**

JavaScript is **good**

- You can change the appearance and behavior of everything that you see in a webpage
- Extremely easy to make other people access your work
- You can write good code if you know how

Starting a Project

index.html

python3 -m http.server
(or py, python… whatever
your python 3 is called)

Browser open on
127.0.0.1:8000

Running your code → loading page in the browser

Starting a Project

index.html

You can open index.html directly from the browser without having a server running, but **you will encounter problems with CORS**

```
python3 -m http.server
```

Run this in the root folder of your project

Browser open on
127.0.0.1:8000

IF YOU OPEN INDEX.HTML USING FILE://,
NSTRUCTO
YOU'RE GONNA HAVE A BAD TIME

Image credit: South Park

Starting a Project

index.html

You can open index.html directly from the browser without having a server running, but **you will encounter problems with CORS**

```
python3 -m http.server
```

Run this in the root folder of your project

Browser open on
127.0.0.1:8000

Starting a Project

index.html

python3 -m http.server

Browser open on
127.0.0.1:8000

style.css

script.js

Editor recommendations

All of them are pretty light, very customizable and ready out of the box

VS Code https://code.visualstudio.com/ (by Microsoft)
- some additional features like autocompletion are built in
- runs on electron (very customizable but heavier than necessary on resources)

Sublime https://www.sublimetext.com/
- lightweight but you can obtain everything you need through plugins
- the only one in this list that is not open source

Atom https://atom.io/ (by Github)
- runs on electron too

Brackets http://brackets.io/ (by Adobe)
- runs on electron too

Notepad++ https://notepad-plus-plus.org/
- Windows on C++

Not ready out of the box:
Vim
- only recommended if you want to spend a good chunk of time configuring it and learning new shortcuts.

Where do I put my script?

Where do I put my script in an HTML page?

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>title</title>
    </head>
    <body>
        <div>content…</div>
        <div>content…</div>
    </body>
</html>
```

http://htmlshell.com/

Ways to run a script

Inline

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>title</title>
    </head>
    <body>
        <div>content…</div>
        <div>content…</div>
        <script>
            … your code ...
        </script>
    </body>
</html>
```

- does NOT scale
- will make you very confused when your code becomes longer
- only good for fast prototyping

From another file

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>title</title>
        <script src="./main.js"></script>
    </head>
    <body>
        <div>content…</div>
        <div>content…</div>
    </body>
</html>
```

- much better, can add as many files as you want and divide your code effectively

From another file (better)

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>title</title>   </head>
    <body>
        <div>content…</div>
        <div>content…</div>
        <script src="./main.js"></script>
    </body>
</html>
```

- scripts at the end avoid need for dealing with async, defer, or onload event handlers

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>title</title>
    </head>
    <body>
        <div>content…</div>
        <div>content…</div>
    </body>
</html>
```

Head (document metadata)

Body (content)

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>title</title>
        <script src="./main1.js"></script>
        <script src="./main2.js"></script>
    </head>
    <body>
        <div>content…</div>
        <script src="./main3.js"></script>
        <div>content…</div>
        <script src="./main4.js"></script>
    </body>
</html>
```

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8">
		<title>title</title>
		<script src="./main1.js"></script>
		<script src="./main2.js"></script>
	</head>
	<body>
		<div>content…</div>
		<script src="./main3.js"></script>
		<div>content…</div>
		<script src="./main4.js"></script>
	</body>
</html>
```

In head:

- Executed before everything else
- Can be used to make sure that some resources are accessible before everything else is loaded
- Can’t access DOM objects (because they have not been created yet) unless forced to wait
- Loading of this script is blocking towards the loading of the rest of the resources and scripts

In body:

- Executed after some content and before some other content
- Only useful for very small, localized scripts

End of body:

- Able to access every DOM element created in body
- Executed after everything else, won’t block loading of the body

Workarounds to keep in mind if you have issues with flow control:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>title</title>
        <script src="./main1.js"></script>
        <script src="./main2.js"></script>
    </head>
    <body>
        <div>content…</div>
        <script src="./main3.js"></script>
        <div>content…</div>
        <script src="./main4.js"></script>
    </body>
</html>
```

Option 1:

```
document.addEventListener(
   'DOMContentLoaded', function() {/*fun code to run*/}
)
```

Use this as a starting point to wait for all content to have loaded in the DOM regardless of where you position your script

The event **DOMContentLoaded** is automatically dispatched by the browser as soon as all the resources are loaded.

Option 2:
Build system / task runner tool set up to do flow control (out of the scope of this class, Google if you want to know more)

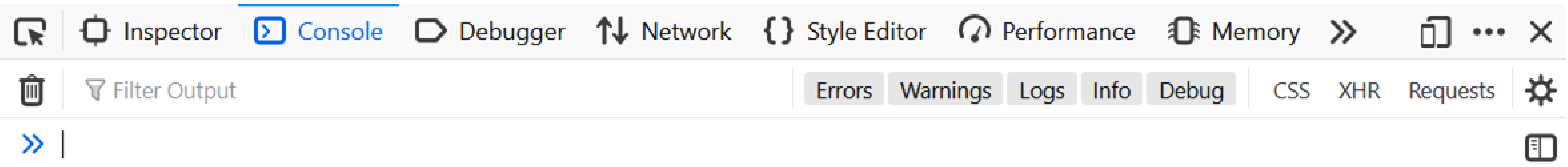
Using the browser console

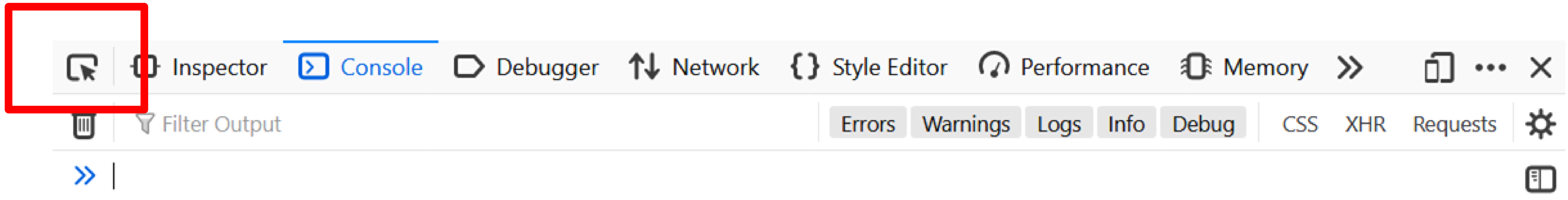
Open the browser console

Ctrl+shift+k on Firefox

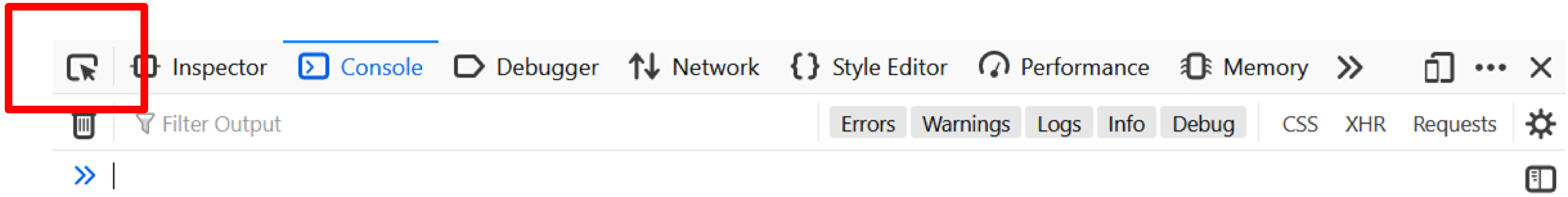
Ctrl+shift+j on Chrome

Or click anywhere on the page with your right click and select “Inspect Element” then click “Console” in the menu





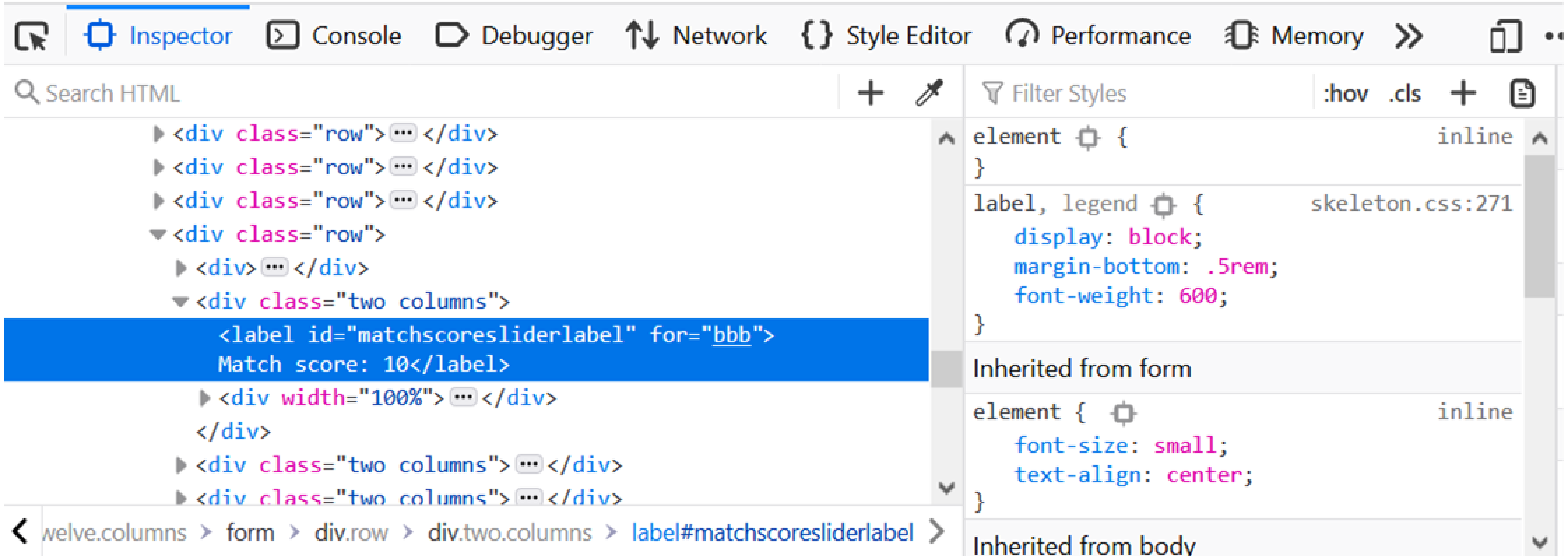
Will allow you to select any element in the page and see its properties, position in the DOM, etc.



Will allow you to select any element in the page and see its properties, position in the DOM, etc.

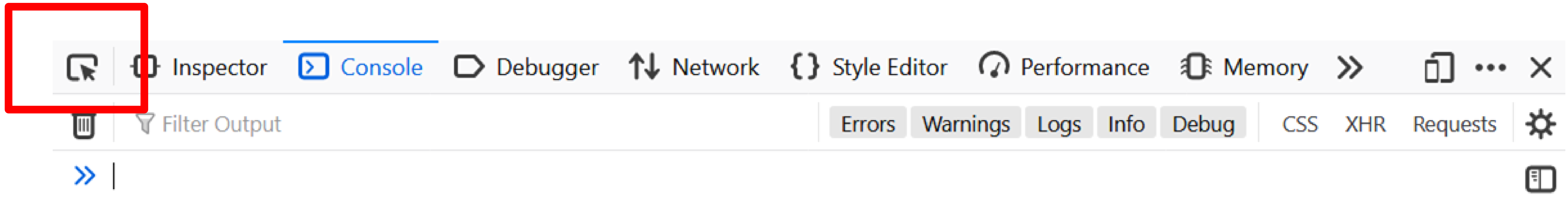
CSS associated to selected element

Selected element in the DOM

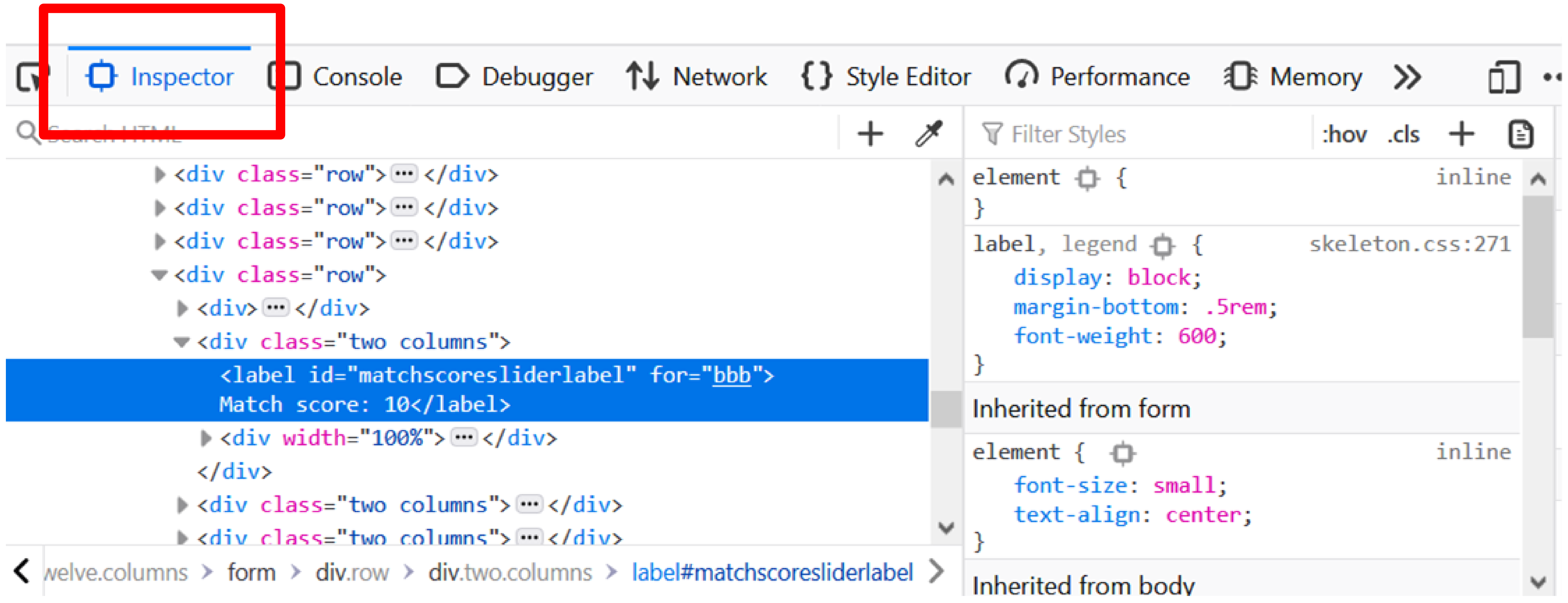


```
  <div class="row">…</div>
  <div class="row">…</div>
  <div class="row">…</div>
  <div class="row">
    <div>…</div>
    <div class="two columns">
      <label id="matchscoresliderlabel" for="bbb">
      Match score: 10</label>
      <div width="100%">…</div>
    </div>
    <div class="two columns">…</div>
    <div class="two columns">…</div>
```

```
element {                                  inline
}
label, legend {                 skeleton.css:271
  display: block;
  margin-bottom: .5rem;
  font-weight: 600;
}
Inherited from form
element {                                  inline
  font-size: small;
  text-align: center;
}
Inherited from body
```



Will allow you to select any element in the page and see its properties, position in the DOM, etc.

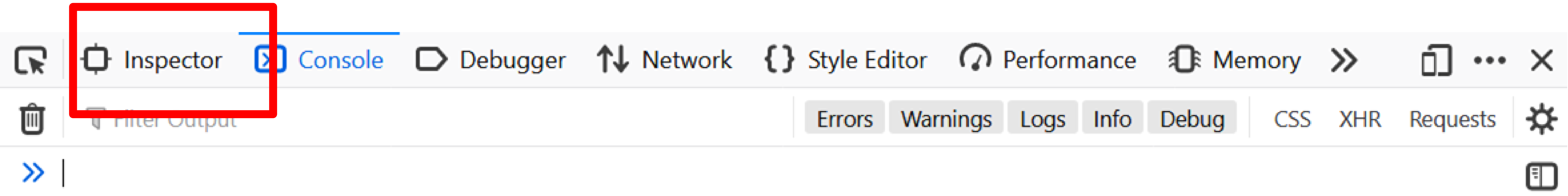


```
▶ <div class="row"> ··· </div>
▶ <div class="row"> ··· </div>
▶ <div class="row"> ··· </div>
▼ <div class="row">
  ▶ <div> ··· </div>
  ▼ <div class="two columns">
      <label id="matchscoresliderlabel" for="bbb">
      Match score: 10</label>
    ▶ <div width="100%"> ··· </div>
    </div>
  ▶ <div class="two columns"> ··· </div>
  ▶ <div class="two columns"> ··· </div>
```

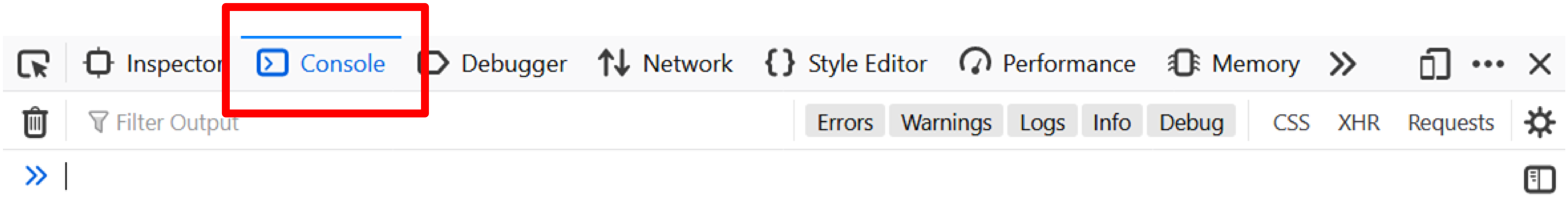
```
element {                                   inline
}
label, legend {                   skeleton.css:271
    display: block;
    margin-bottom: .5rem;
    font-weight: 600;
}
Inherited from form
element {                                   inline
    font-size: small;
    text-align: center;
}
Inherited from body
```

Will allow you to answer questions such as:

- What is the id of this element that I am seeing?
- Is this element in the correct position in the DOM?
- What events are associated to this element?
- How would this element look like if I make it red without having to re-run the whole page?

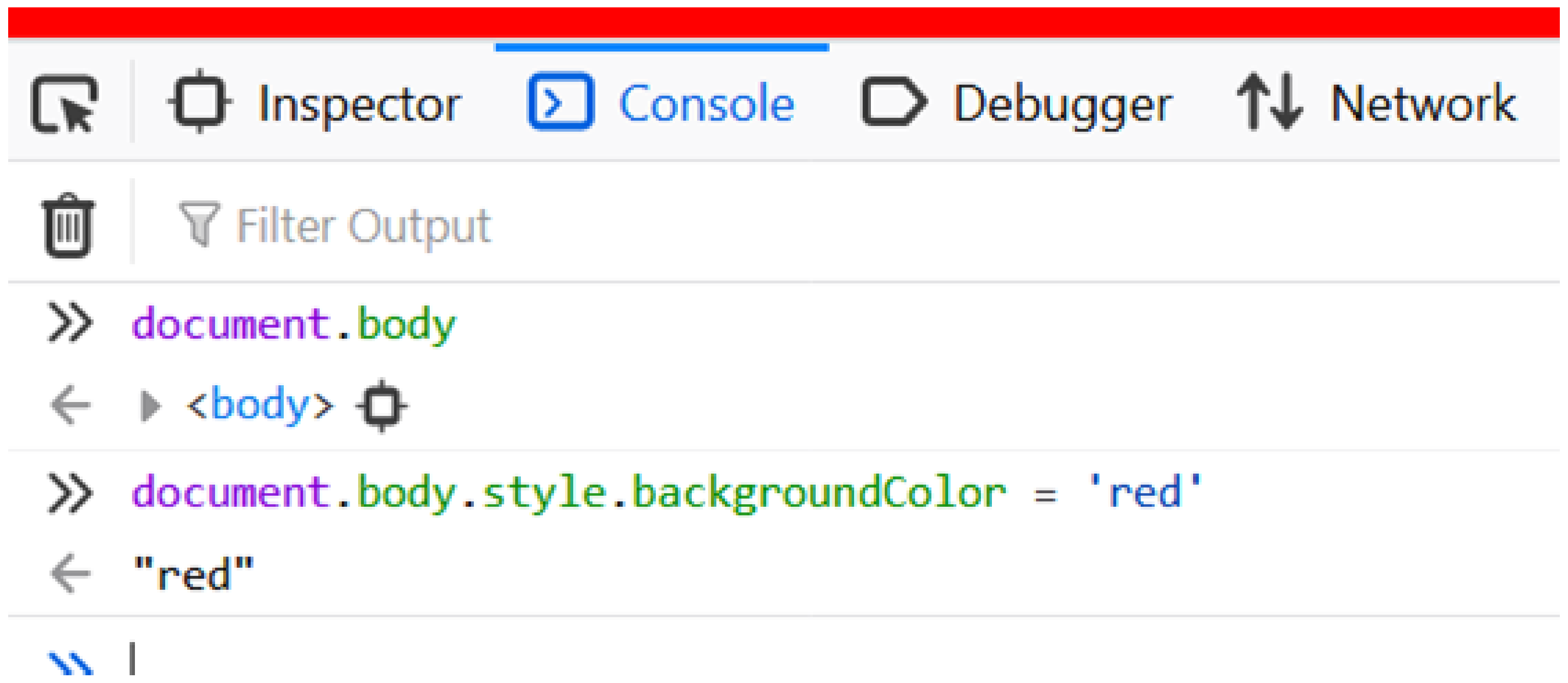


Shows the structure of the page plus CSS style associated with it



Shows print output and **errors**
Can run scripts after page is loaded

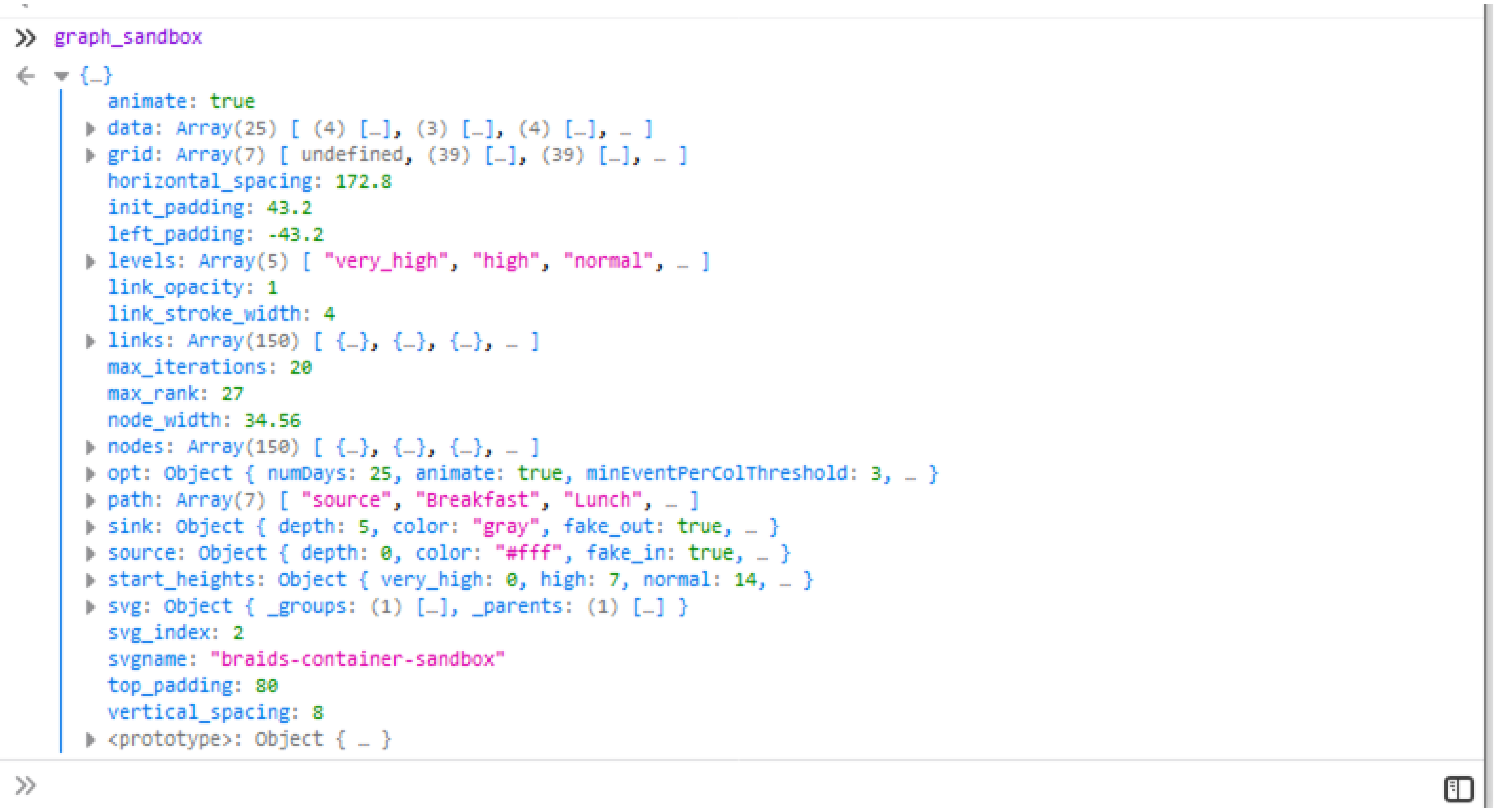
example:



```
>> document.body
<- <body>
>> document.body.style.backgroundColor = 'red'
<- "red"
```

Everything is an object

And everything can be printed in the console
If you **print an object in the browser console**, you can **navigate the fields of the object** and the functions associated with it



```
» graph_sandbox
← ▼ {…}
      animate: true
    ▶ data: Array(25) [ (4) […], (3) […], (4) […], … ]
    ▶ grid: Array(7) [ undefined, (39) […], (39) […], … ]
      horizontal_spacing: 172.8
      init_padding: 43.2
      left_padding: -43.2
    ▶ levels: Array(5) [ "very_high", "high", "normal", … ]
      link_opacity: 1
      link_stroke_width: 4
    ▶ links: Array(150) [ {…}, {…}, {…}, … ]
      max_iterations: 20
      max_rank: 27
      node_width: 34.56
    ▶ nodes: Array(150) [ {…}, {…}, {…}, … ]
    ▶ opt: Object { numDays: 25, animate: true, minEventPerColThreshold: 3, … }
    ▶ path: Array(7) [ "source", "Breakfast", "Lunch", … ]
    ▶ sink: Object { depth: 5, color: "gray", fake_out: true, … }
    ▶ source: Object { depth: 0, color: "#fff", fake_in: true, … }
    ▶ start_heights: Object { very_high: 0, high: 7, normal: 14, … }
    ▶ svg: Object { _groups: (1) […], _parents: (1) […] }
      svg_index: 2
      svgname: "braids-container-sandbox"
      top_padding: 80
      vertical_spacing: 8
    ▶ <prototype>: Object { … }
»
```

Note: you can access any DOM element too as JavaScript objects

Callbacks and events

Callbacks and events

“Event-driven architecture”: the flow of a program is defined by **events**.

Events can be generated by the user or by the browser. Examples of events that you will want to use a callback for:

- user interacts with an element
- loading of a resource is completed
- browser window is resized
- request to some API is returned

…

Callbacks and events

Most of the events that you will use are already defined by the browser.

Examples:

- **mouseover**: cursor enters the bounding box of a specified element
- **mouseout**: cursor exits the bounding box of a specified element
- **onClick**: user clicks on specified element
- **onWindowResize**: browser window is resized
- **onDocumentReady**: all resources in document are loaded

You can also define and dispatch your own events

Callbacks and events

Adding an event listener to an item:

a callback

```
item.on(‘mouseover’, function(){
    console.log(‘hello’);
})
```

Events are usually managed using callbacks.

Callbacks are nameless functions that are executed after a condition is verified.

Callbacks and events

Adding an event listener to an item:

a callback

```
item.on(‘mouseover’, function(){
    console.log(‘hello’);
})
```

≈

```
item.on(‘mouseover’, () => {
    console.log(‘hello’);
})
```

Events are usually managed using callbacks.

Callbacks are nameless functions that are executed after a condition is verified.

Callbacks and events

Callbacks are not only for events:

```
myArray = [1, 2, 3, 4, 5, 6]
result = myArray.filter( function(a) => {
    return a%2==0
})
// returns [2, 4, 6]
```

In this case, we use a callback to filter an array, keeping only even numbers

Callbacks and events

Similar to **lambdas** in python

JS

```
myArray = [1, 2, 3, 4, 5, 6]
result = myArray.filter(function(a) ⇒ {
    return a%2==0
})
// returns [2, 4, 6]
```

Python

```
myArray = [1, 2, 3, 4, 5, 6]
result = list(filter(lambda a: (a%2 == 0),
myArray))
// returns [2, 4, 6]
```

Ways to declare a variable

x = 5;	Global (or error in strict mode)
var x = 5, y = 6, z = 7;	Global
let x = 5;	Scope of the variable is constrained to the scope in which it has been declared.
const x = 5;	Scope limited, x has to be constant.

Recommended to generally use **let** and **const** instead of **var**

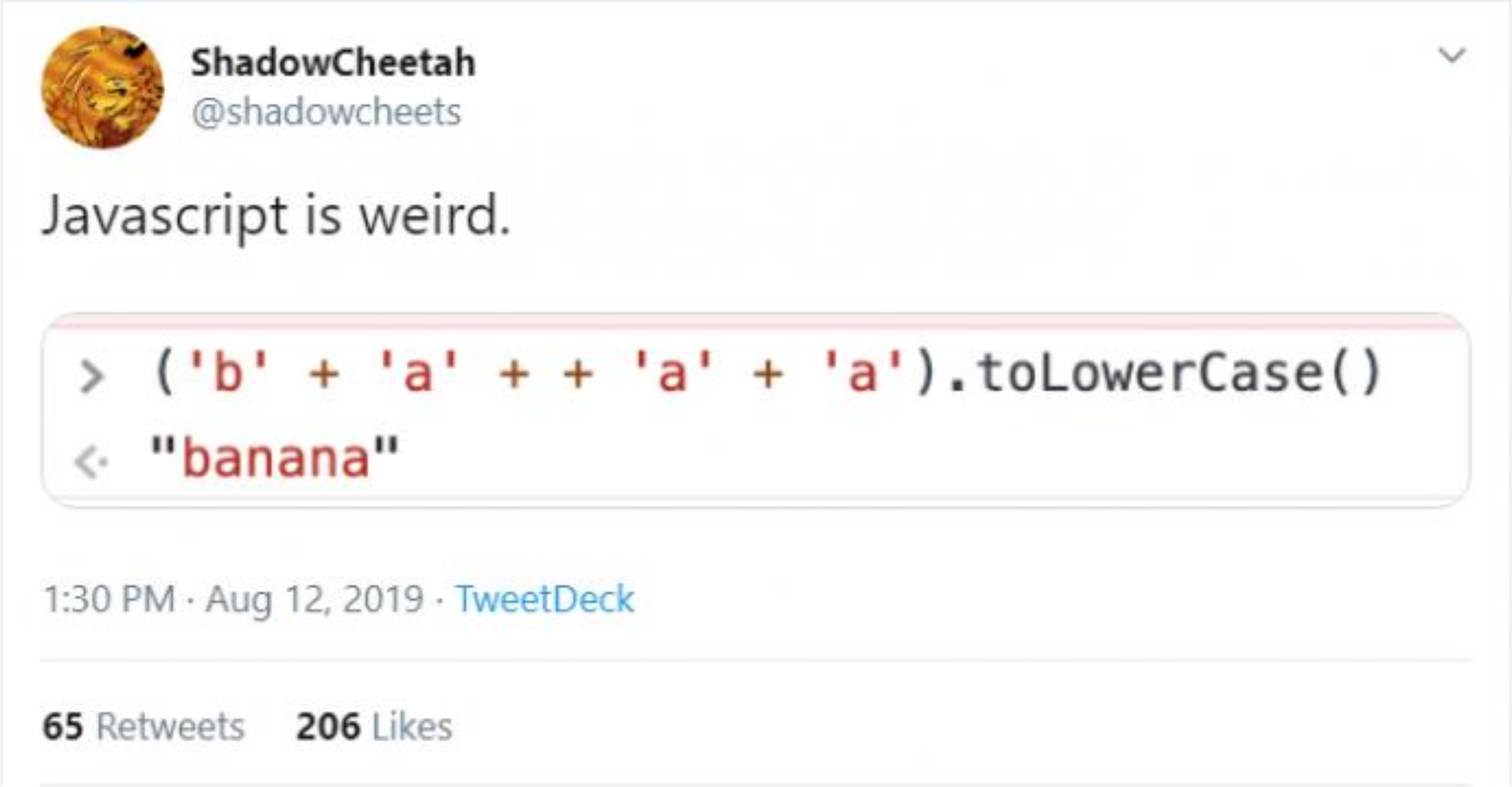
```
if (true) {
        var foo = 5;
}

console.log(foo);   // 5
```

```
if (true) {
        let foo = 5;
}

console.log(foo);   // undefined
```

Always be aware of the data type that you are dealing with



ShadowCheetah
@shadowcheets

Javascript is weird.

```
> ('b' + 'a' + + 'a' + 'a').toLowerCase()
<· "banana"
```

1:30 PM · Aug 12, 2019 · TweetDeck

65 Retweets **206** Likes

https://github.com/denysdovhan/wtfjs

Ways to declare a function

```
name("Ted");
```

Function declaration

```
function name (params) {
        ...
}
```

Function expression

```
let name = function (params) {
        ...
}
```

Arrow function

```
let name = (params) => {
        ...
}
```

All of these will have *almost* the same effect

In arrow function: this, arguments from outer function; no constructor; implicit return

Hoisting: a function will be positioned at the top of the scope and made available at any point of its own scope even before its own declaration

Arrow functions will let you write a lot of fun oneliners:

```
// custom sorting function
[3, 1, 2, 4].sort((a, b) => a < b)
→ [1, 2, 3, 4]

// custom filtering function
[1, 2, 3, 4].filter(a => a%2 == 0)
→ [2, 4]

// sum of all elements in an array
[1, 2, 3, 4].reduce((a, b) => a + b, 0)
→ 10

// sort then filter then sum
[3, 1, 2, 4].sort((a, b) => a < b).filter(a => a%2 == 0).reduce((a, b) => a + b, 0)
→ 6
```

Style guides

Google style guide: https://google.github.io/styleguide/javascriptguide.xml

Airbnb: https://github.com/airbnb/javascript

Standardjs: https://standardjs.com/#the-rules

Idiomatic: https://github.com/rwaldron/idiomatic.js

Linting

Linters force you to write code following some pre-established policies.

Jslint: http://www.jslint.com/

jshint: https://jshint.com/ started as a fork of jslint, customizable

prettier: https://prettier.io/ customizable

Automated code review

one of many tools to check issues in your code:

https://www.codacy.com/

(example)
https://app.codacy.com/app/picorana/sparqling/files?bid=7480002

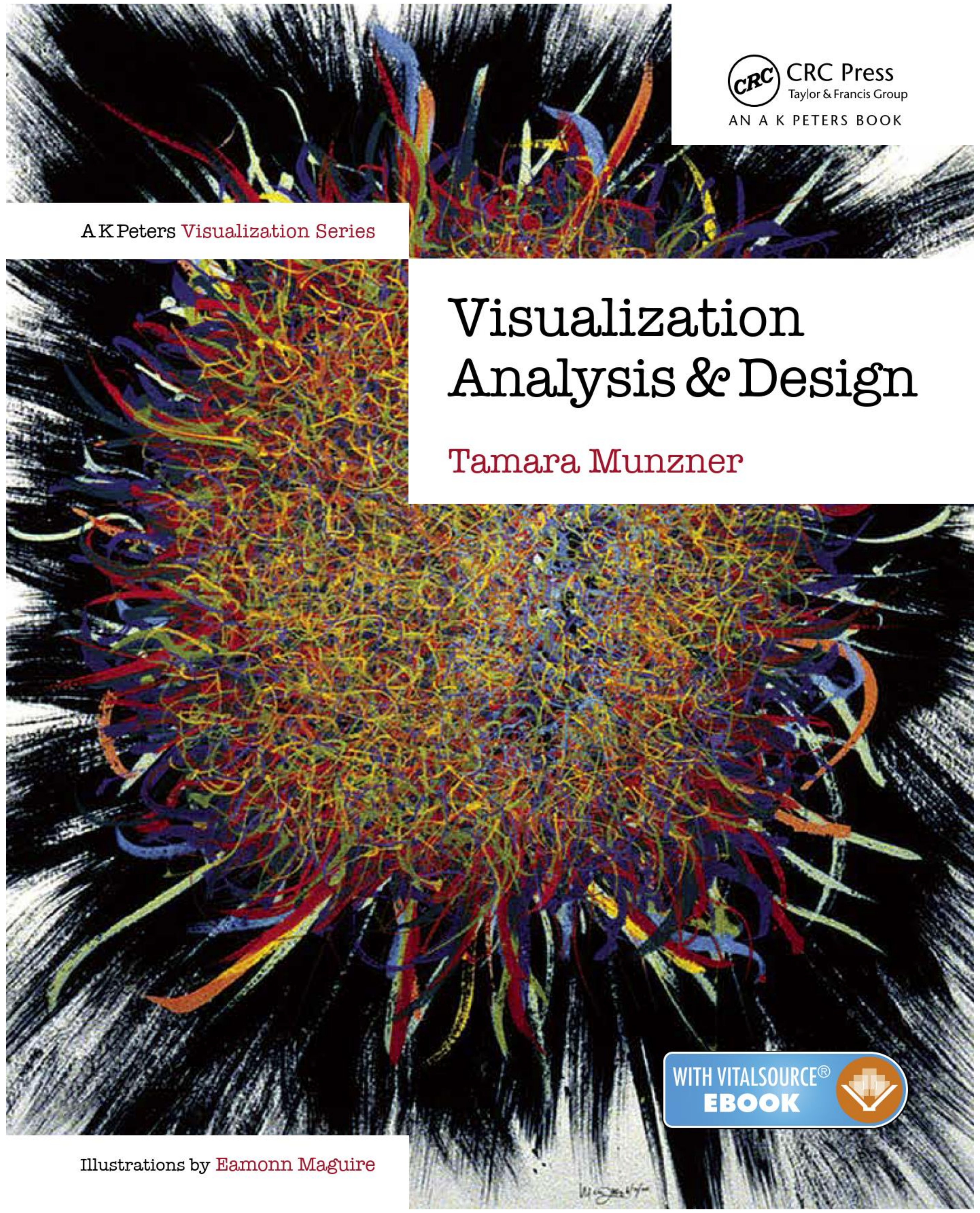
In-Class Programming — JavaScript

~25 min total

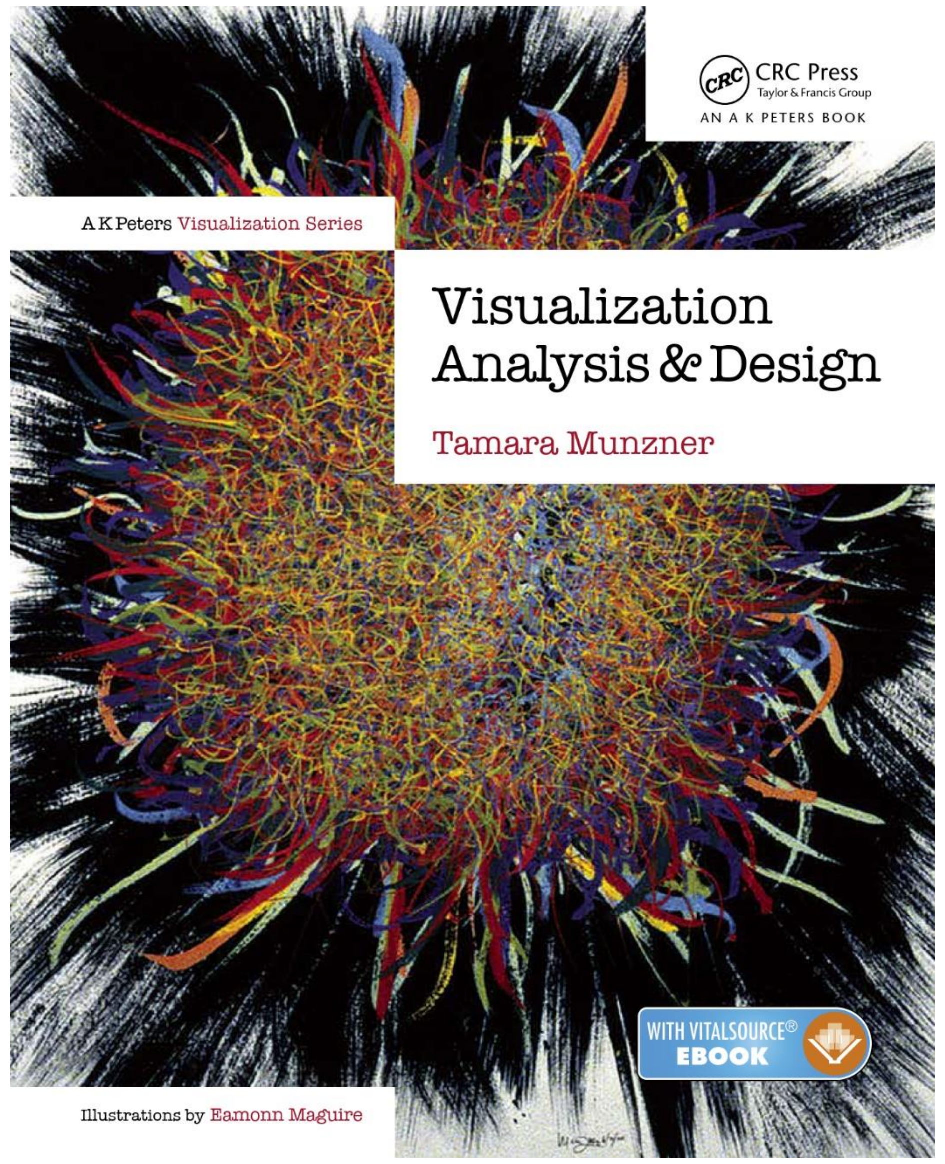
The Nested Model for Visualization Development

Used for your Projects

TEXTBOOK



Additional “recommended” books as resources in syllabus


CRC Press
Taylor & Francis Group
AN A K PETERS BOOK
A K Peters Visualization Series
Visualization Analysis & Design
Tamara Munzner
WITH VITALSOURCE® EBOOK
Illustrations by Eamonn Maguire

“Nested Model”

Domain situation
Observe target users using existing tools

Example

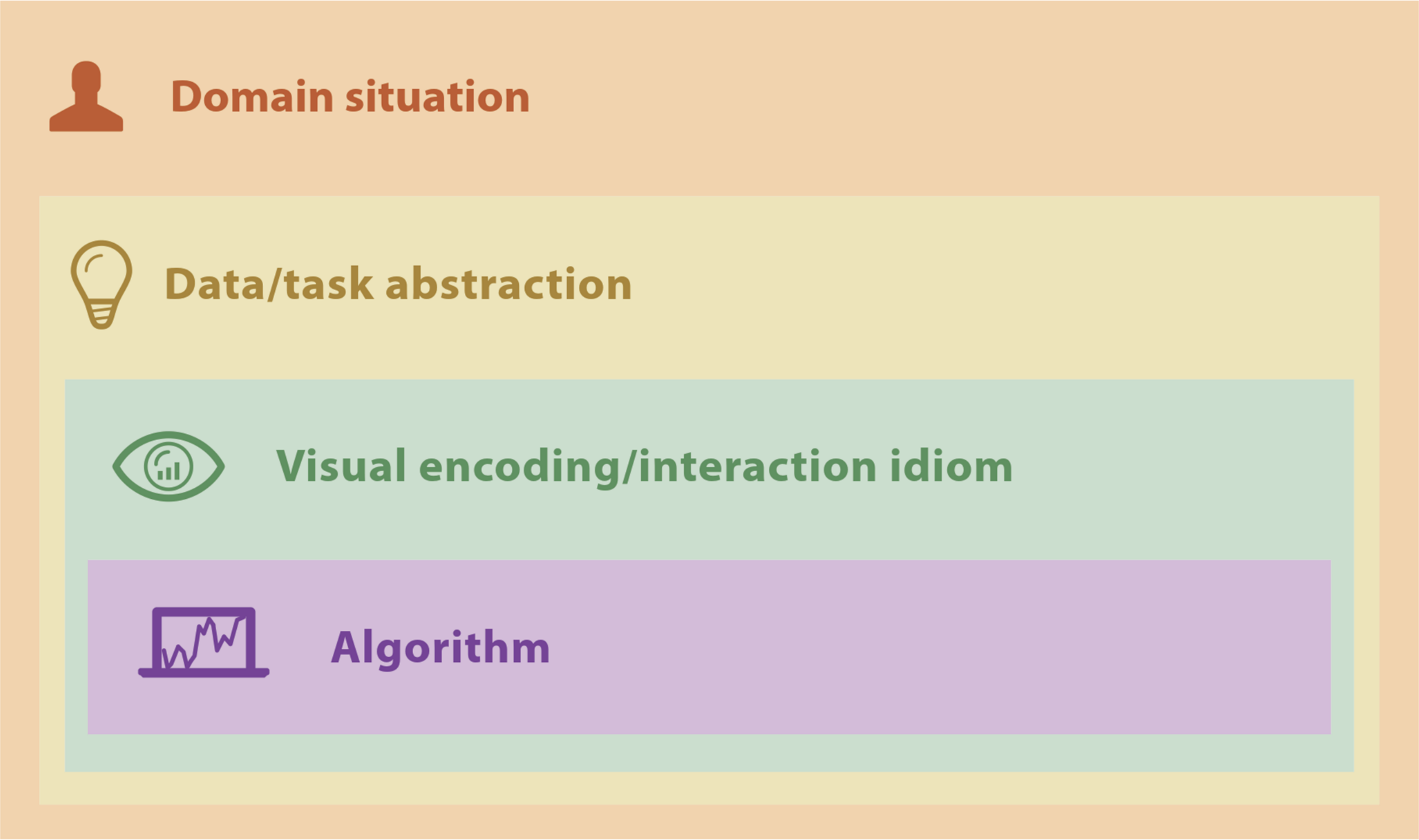
FAA (aviation)

What is the busiest time of day at Logan Airport?

Map vs. Scatter Plot vs. Bar

[Tamara Munzner]

Nested Model



Domain situation

Data/task abstraction

Visual encoding/interaction idiom

Algorithm

Nested Model

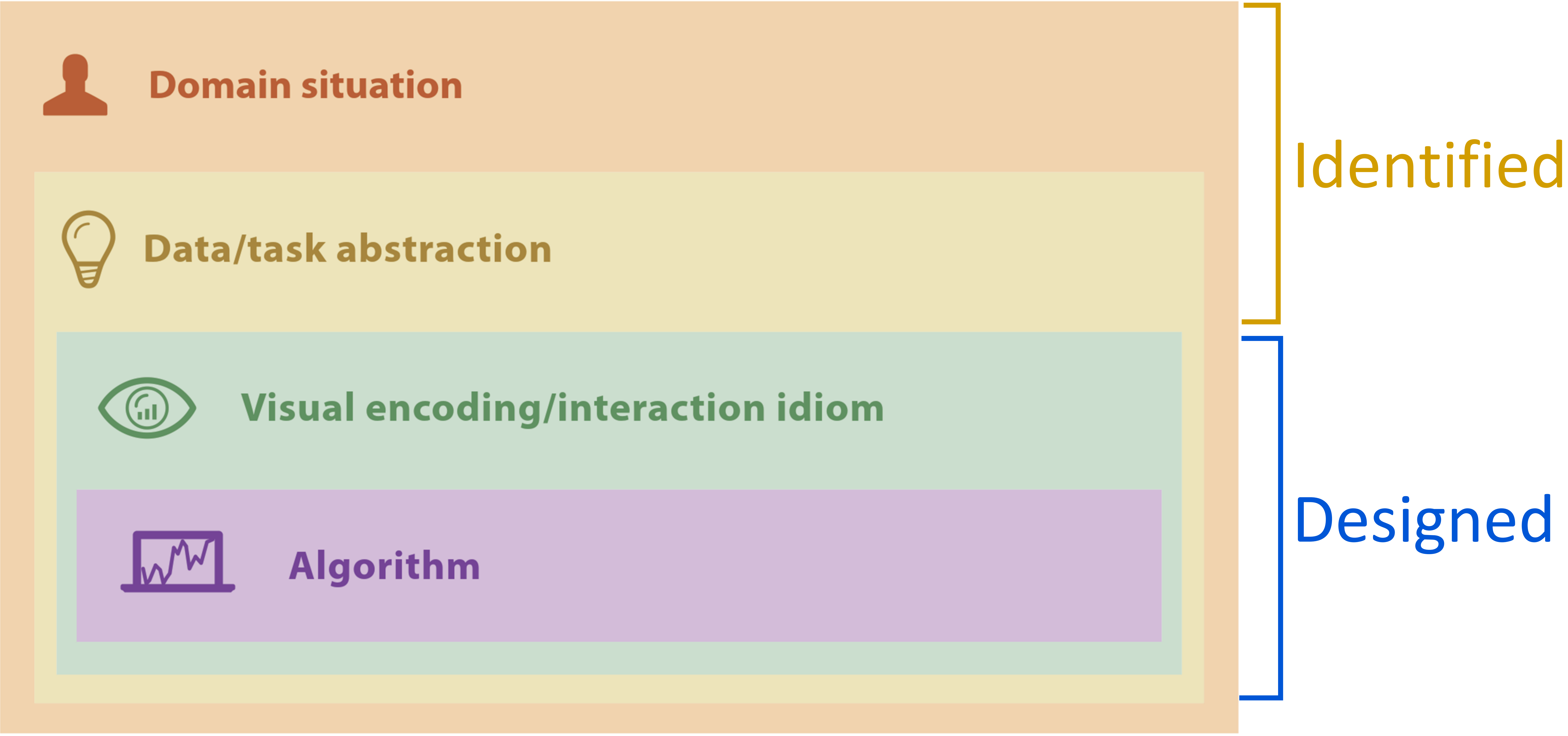
Human-centered design

Designer understands user

Abstract domain tasks

Visualization design

Implementation



Domain situation

Data/task abstraction

Visual encoding/interaction idiom

Algorithm

Identified

Designed

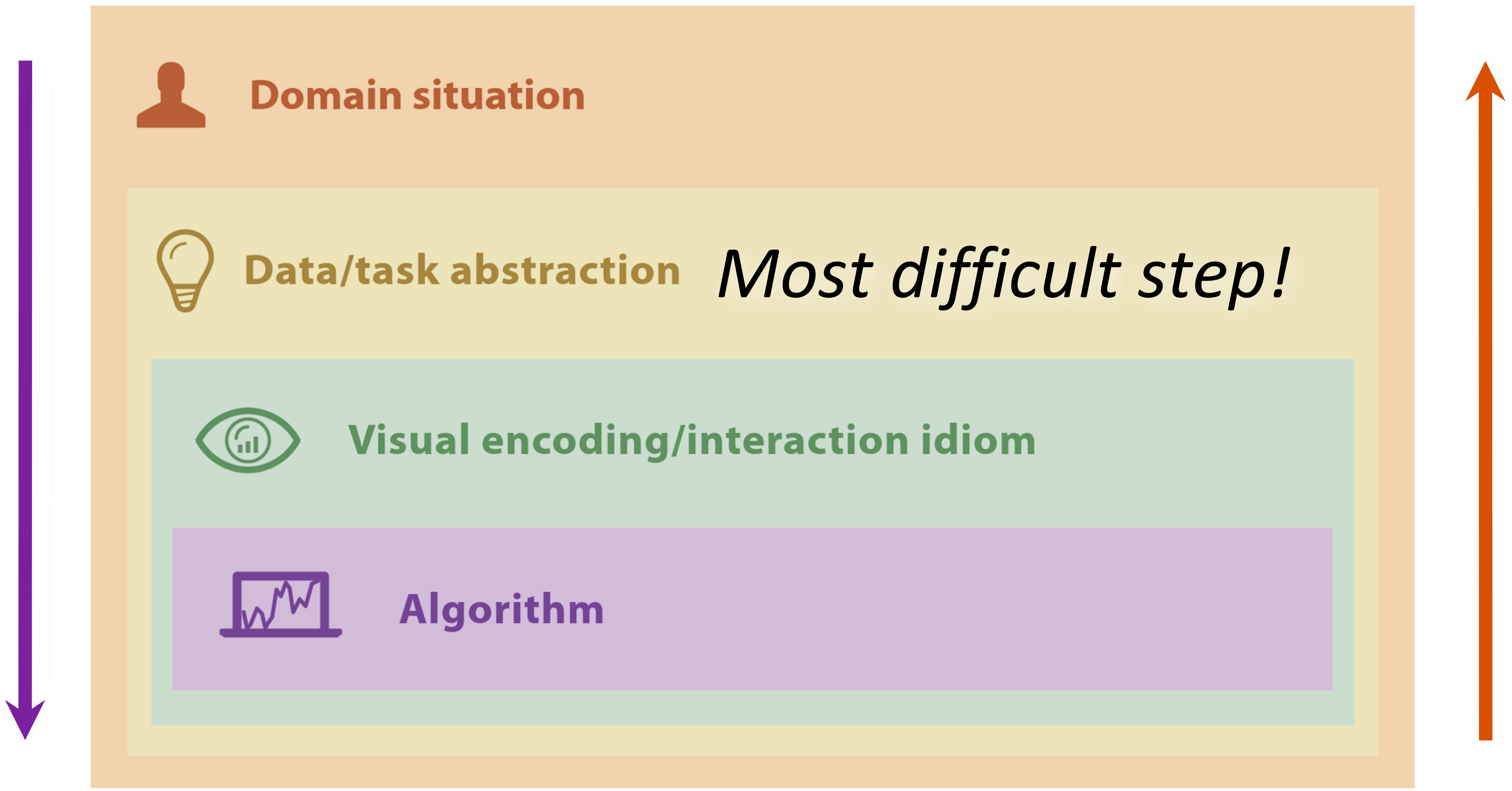
Nested Model

Design Study

TOP-DOWN
“problem-driven”

Technique

BOTTOM-UP
“technique-driven”



Domain situation

Data/task abstraction

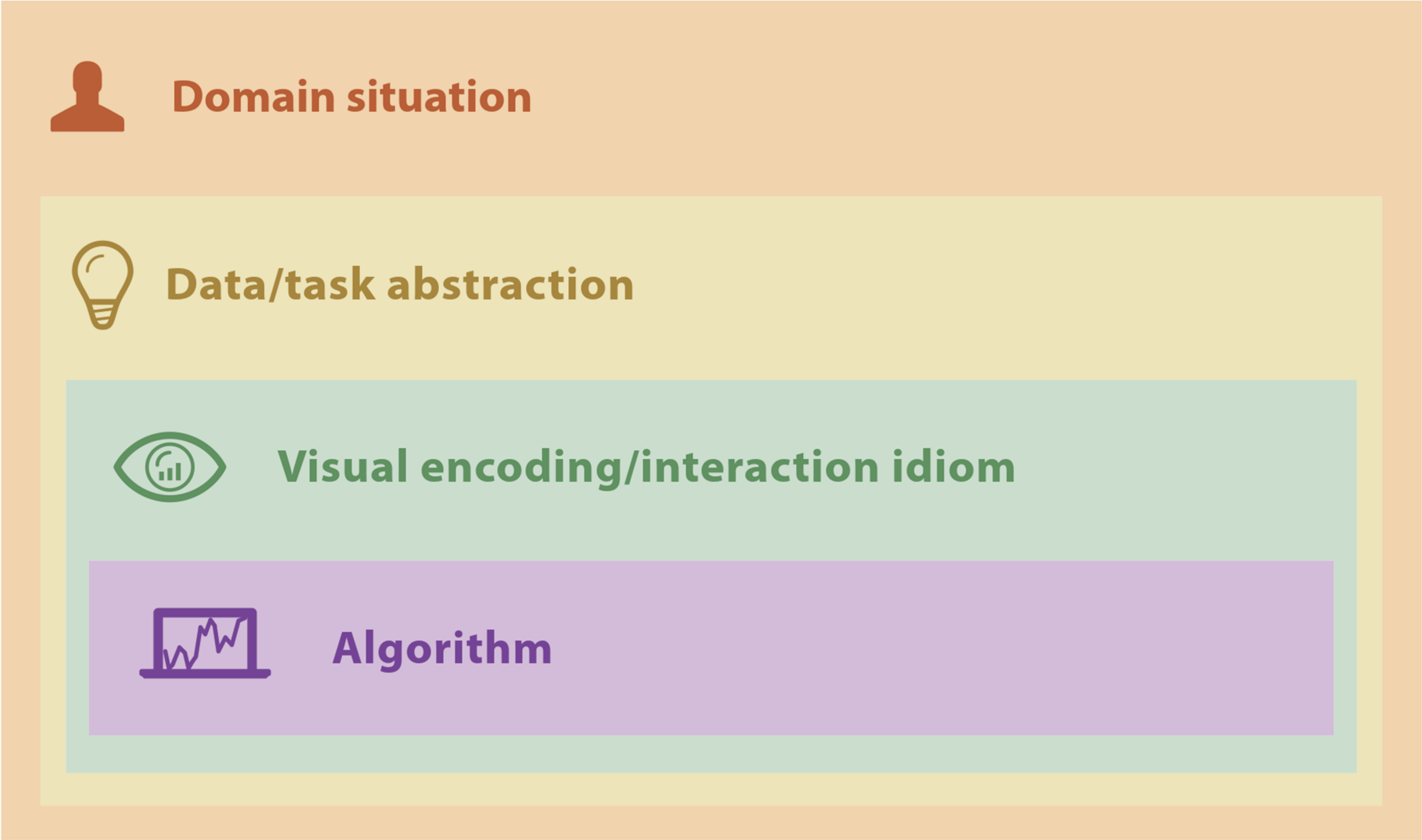
Most difficult step!

Visual encoding/interaction idiom

Algorithm

Nested Model

Mistakes propagate through model!



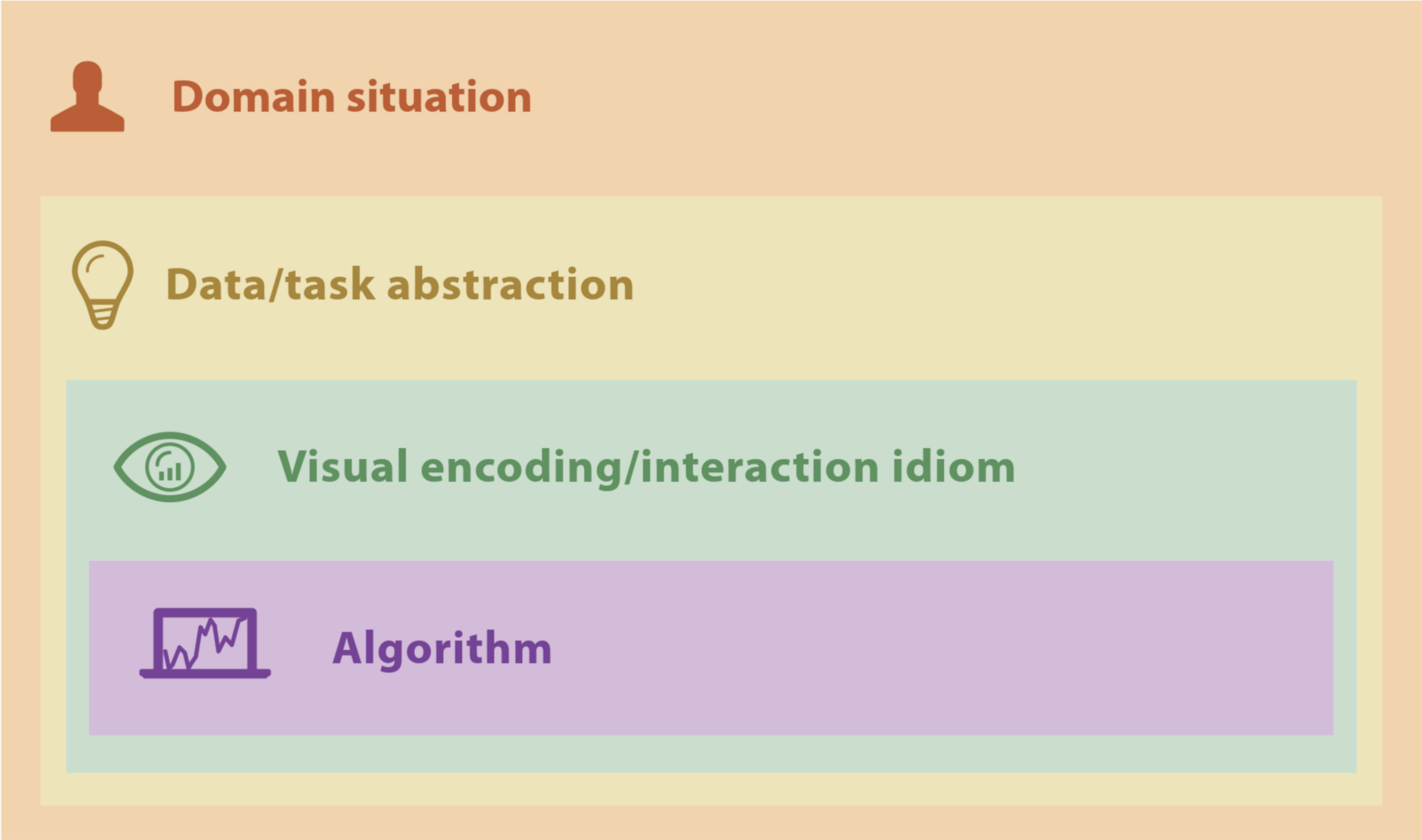
Domain situation

Data/task abstraction

Visual encoding/interaction idiom

Algorithm

Threats to Validity



Domain situation

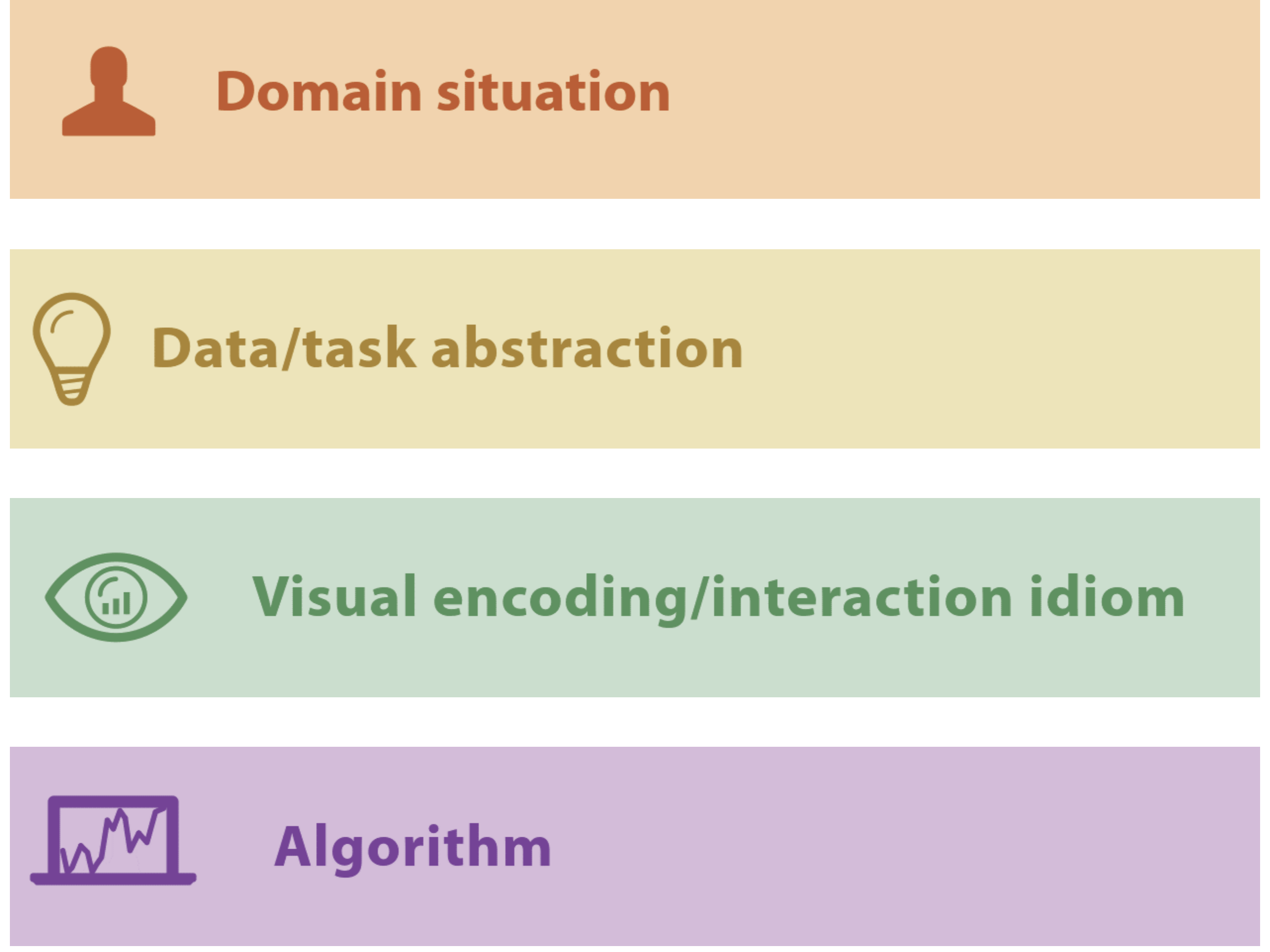
Data/task abstraction

Visual encoding/interaction idiom

Algorithm

Threats to Validity

✓ Final Project validation



Domain situation

Data/task abstraction

Visual encoding/interaction idiom

Algorithm

Final project follow-up

PROJECTS

(Using the nested model via *design study "lite" methodology*)

https://northeastern.instructure.com/courses/63405/pages/project-overview

EXPERIENTIAL LEARNING PROJECTS

Why are we doing experiential learning?

Design Study "Lite" Methodology (Borkin et al. 2017, Syeda et al. (2020))

- Design studies are a growing and valuable research area.
- Real-world data visualization experience.
- Visualization for exploration and communication.
- A more realistic experience of creating visualizations, and doing work in general.
- Teaches design, interview, evaluation, communication, and feedback techniques difficult to replicate in a classroom.
- Higher-stakes deliverables.
- Professional development.
- Make a positive impact in the community.
- Publication?

Experiential Learning Projects

What are the challenges?

- Real-world data is messy and difficult to gather and process.
- Partners may not have clear goals and expectations.
- There is communication and scheduling overhead, inc. for teaching staff to differentiate assignment grading if necessary.
- Project areas may be too predefined.
- Project areas may be too ambiguous.
- May not actually make a meaningful impact.
- Reduces time for white-room technical education.
- More ambiguous expectations and grading challenges.
- Possible variation in student workload.
- Students may not know they are signing up for Service-Learning in advance (common problem with our registrar).

Experiential Learning Projects

Who to blame for getting you into this?

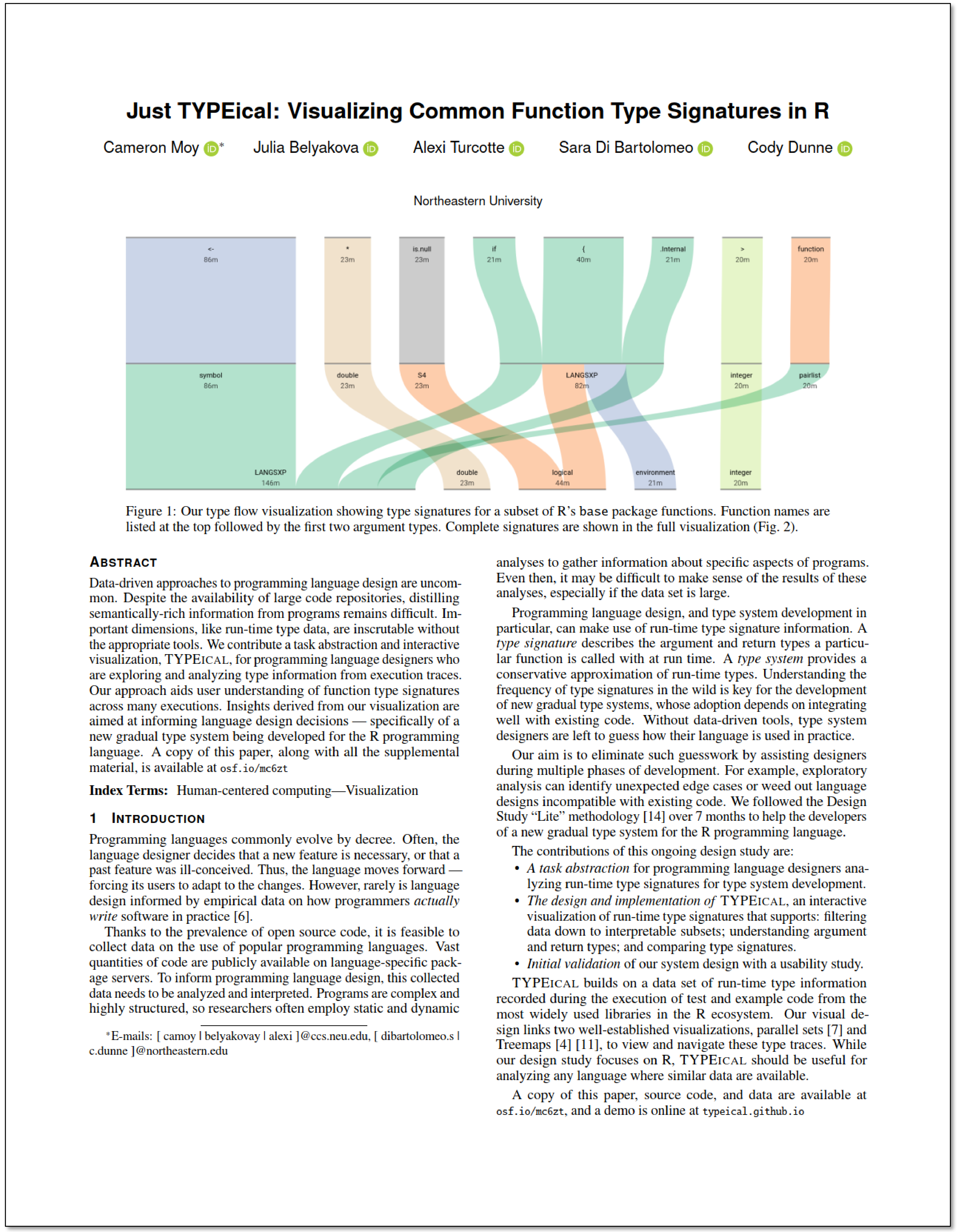
Michelle Borkin

Cody Dunne

Examples of Successful Course Projects

(Albeit with different requirements per course)

Project Example — Just TYPEical



Just TYPEical: Visualizing Common Function Type Signatures in R

Cameron Moy* Julia Belyakova Alexi Turcotte Sara Di Bartolomeo Cody Dunne

Northeastern University

Figure 1: Our type flow visualization showing type signatures for a subset of R's base package functions. Function names are listed at the top followed by the first two argument types. Complete signatures are shown in the full visualization (Fig. 2).

Abstract

Data-driven approaches to programming language design are uncommon. Despite the availability of large code repositories, distilling semantically-rich information from programs remains difficult. Important dimensions, like run-time type data, are inscrutable without the appropriate tools. We contribute a task abstraction and interactive visualization, TYPEical, for programming language designers who are exploring and analyzing type information from execution traces. Our approach aids user understanding of function type signatures across many executions. Insights derived from our visualization are aimed at informing language design decisions — specifically of a new gradual type system being developed for the R programming language. A copy of this paper, along with all the supplemental material, is available at `osf.io/mc6zt`

Index Terms: Human-centered computing—Visualization

1 Introduction

Programming languages commonly evolve by decree. Often, the language designer decides that a new feature is necessary, or that a past feature was ill-conceived. Thus, the language moves forward — forcing its users to adapt to the changes. However, rarely is language design informed by empirical data on how programmers *actually write* software in practice [6].

Thanks to the prevalence of open source code, it is feasible to collect data on the use of popular programming languages. Vast quantities of code are publicly available on language-specific package servers. To inform programming language design, this collected data needs to be analyzed and interpreted. Programs are complex and highly structured, so researchers often employ static and dynamic analyses to gather information about specific aspects of programs. Even then, it may be difficult to make sense of the results of these analyses, especially if the data set is large.

Programming language design, and type system development in particular, can make use of run-time type signature information. A *type signature* describes the argument and return types a particular function is called with at run time. A *type system* provides a conservative approximation of run-time types. Understanding the frequency of type signatures in the wild is key for the development of new gradual type systems, whose adoption depends on integrating well with existing code. Without data-driven tools, type system designers are left to guess how their language is used in practice.

Our aim is to eliminate such guesswork by assisting designers during multiple phases of development. For example, exploratory analysis can identify unexpected edge cases or weed out language designs incompatible with existing code. We followed the Design Study "Lite" methodology [14] over 7 months to help the developers of a new gradual type system for the R programming language.

The contributions of this ongoing design study are:

- *A task abstraction* for programming language designers analyzing run-time type signatures for type system development.
- *The design and implementation of* TYPEical, an interactive visualization of run-time type signatures that supports: filtering data down to interpretable subsets; understanding argument and return types; and comparing type signatures.
- *Initial validation* of our system design with a usability study.

TYPEical builds on a data set of run-time type information recorded during the execution of test and example code from the most widely used libraries in the R ecosystem. Our visual design links two well-established visualizations, parallel sets [7] and Treemaps [4] [11], to view and navigate these type traces. While our design study focuses on R, TYPEical should be useful for analyzing any language where similar data are available.

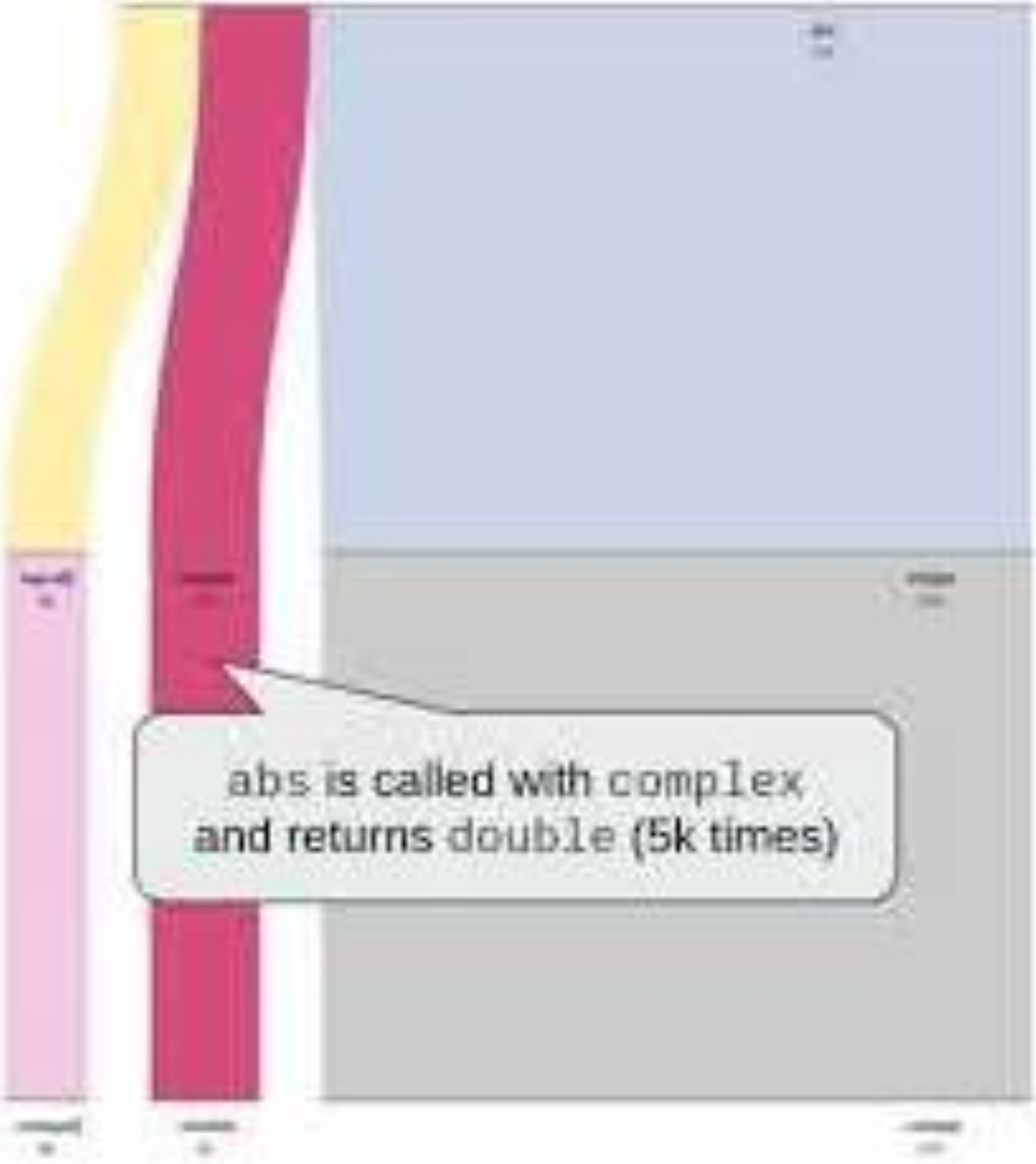
A copy of this paper, source code, and data are available at `osf.io/mc6zt`, and a demo is online at `typeical.github.io`

*E-mails: [camoy | belyakovay | alexi]@ccs.neu.edu, [dibartolomeo.s | c.dunne]@northeastern.edu

CS 7250 Spring 2020:
Information Visualization:
Theory and Applications

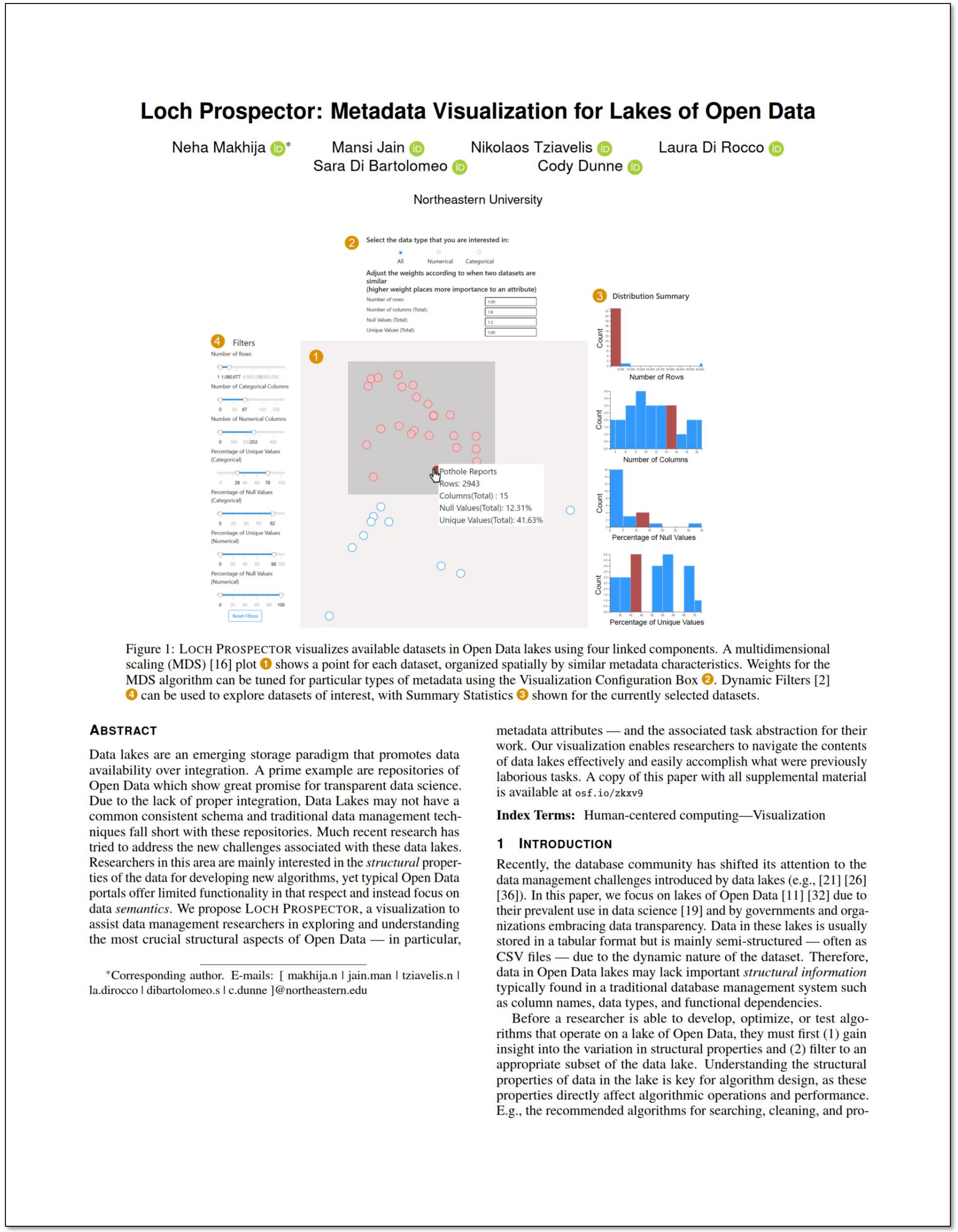
Website Moy et al. VIS 2020

TYPEical: A tool for programming language designers



`abs` is called with `complex` and returns `double` (5k times)

PROJECT EXAMPLE — LOCH PROSPECTOR



Loch Prospector: Metadata Visualization for Lakes of Open Data

Neha Makhija*, Mansi Jain, Nikolaos Tziavelis, Laura Di Rocco, Sara Di Bartolomeo, Cody Dunne

Northeastern University

2
Select the data type that you are interested in:
All
Numerical
Categorical
Adjust the weights according to when two datasets are similar
(higher weight places more importance to an attribute)
Number of rows:
Number of columns (Total):
Null Values (Total):
Unique Values (Total):
3
Distribution Summary
Count
Number of Rows
Number of Columns
Percentage of Null Values
Percentage of Unique Values
4
Filters
Number of Rows
Number of Categorical Columns
Number of Numerical Columns
Percentage of Unique Values (Categorical)
Percentage of Null Values (Categorical)
Percentage of Unique Values (Numerical)
Percentage of Null Values (Numerical)
Reset Filters
1
Pothole Reports
Rows: 2943
Columns(Total) : 15
Null Values(Total): 12.31%
Unique Values(Total): 41.63%

Figure 1: LOCH PROSPECTOR visualizes available datasets in Open Data lakes using four linked components. A multidimensional scaling (MDS) [16] plot 1 shows a point for each dataset, organized spatially by similar metadata characteristics. Weights for the MDS algorithm can be tuned for particular types of metadata using the Visualization Configuration Box 2. Dynamic Filters [2] 4 can be used to explore datasets of interest, with Summary Statistics 3 shown for the currently selected datasets.

ABSTRACT

Data lakes are an emerging storage paradigm that promotes data availability over integration. A prime example are repositories of Open Data which show great promise for transparent data science. Due to the lack of proper integration, Data Lakes may not have a common consistent schema and traditional data management techniques fall short with these repositories. Much recent research has tried to address the new challenges associated with these data lakes. Researchers in this area are mainly interested in the *structural* properties of the data for developing new algorithms, yet typical Open Data portals offer limited functionality in that respect and instead focus on data *semantics*. We propose LOCH PROSPECTOR, a visualization to assist data management researchers in exploring and understanding the most crucial structural aspects of Open Data — in particular, metadata attributes — and the associated task abstraction for their work. Our visualization enables researchers to navigate the contents of data lakes effectively and easily accomplish what were previously laborious tasks. A copy of this paper with all supplemental material is available at `osf.io/zkxv9`

Index Terms: Human-centered computing—Visualization

1 INTRODUCTION

Recently, the database community has shifted its attention to the data management challenges introduced by data lakes (e.g., [21] [26] [36]). In this paper, we focus on lakes of Open Data [11] [32] due to their prevalent use in data science [19] and by governments and organizations embracing data transparency. Data in these lakes is usually stored in a tabular format but is mainly semi-structured — often as CSV files — due to the dynamic nature of the dataset. Therefore, data in Open Data lakes may lack important *structural information* typically found in a traditional database management system such as column names, data types, and functional dependencies.

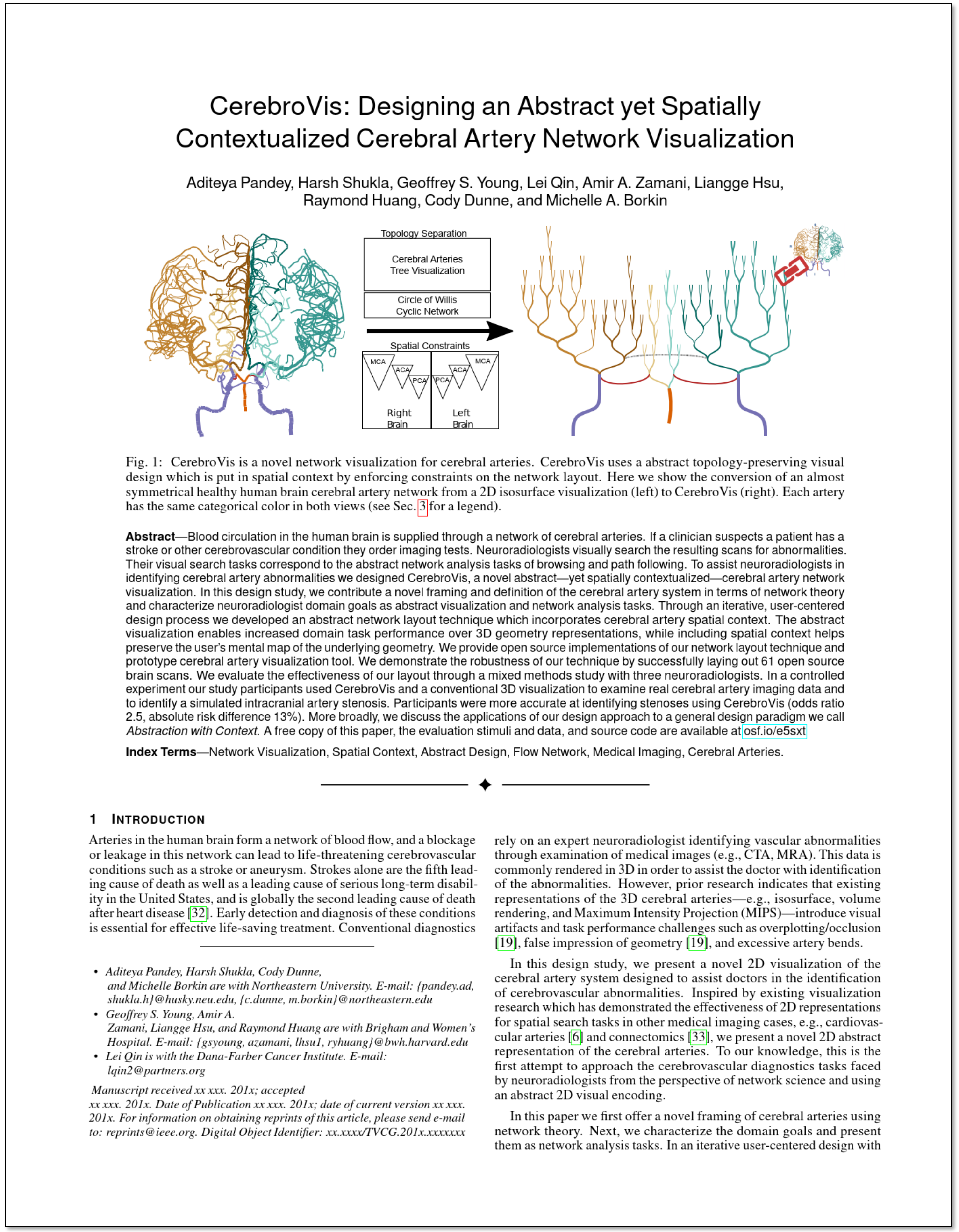
Before a researcher is able to develop, optimize, or test algorithms that operate on a lake of Open Data, they must first (1) gain insight into the variation in structural properties and (2) filter to an appropriate subset of the data lake. Understanding the structural properties of data in the lake is key for algorithm design, as these properties directly affect algorithmic operations and performance. E.g., the recommended algorithms for searching, cleaning, and pro-

*Corresponding author. E-mails: [makhija.n | jain.man | tziavelis.n | la.dirocco | dibartolomeo.s | c.dunne]@northeastern.edu

CS 7250 SPRING 2020:
INFORMATION VISUALIZATION:
THEORY AND APPLICATIONS

Website Makhija et al. VIS 2020

Project Example — CerebroVis



CerebroVis: Designing an Abstract yet Spatially Contextualized Cerebral Artery Network Visualization

Aditeya Pandey, Harsh Shukla, Geoffrey S. Young, Lei Qin, Amir A. Zamani, Liangge Hsu,
Raymond Huang, Cody Dunne, and Michelle A. Borkin

Fig. 1: CerebroVis is a novel network visualization for cerebral arteries. CerebroVis uses a abstract topology-preserving visual design which is put in spatial context by enforcing constraints on the network layout. Here we show the conversion of an almost symmetrical healthy human brain cerebral artery network from a 2D isosurface visualization (left) to CerebroVis (right). Each artery has the same categorical color in both views (see Sec. 3 for a legend).

Abstract—Blood circulation in the human brain is supplied through a network of cerebral arteries. If a clinician suspects a patient has a stroke or other cerebrovascular condition they order imaging tests. Neuroradiologists visually search the resulting scans for abnormalities. Their visual search tasks correspond to the abstract network analysis tasks of browsing and path following. To assist neuroradiologists in identifying cerebral artery abnormalities we designed CerebroVis, a novel abstract—yet spatially contextualized—cerebral artery network visualization. In this design study, we contribute a novel framing and definition of the cerebral artery system in terms of network theory and characterize neuroradiologist domain goals as abstract visualization and network analysis tasks. Through an iterative, user-centered design process we developed an abstract network layout technique which incorporates cerebral artery spatial context. The abstract visualization enables increased domain task performance over 3D geometry representations, while including spatial context helps preserve the user's mental map of the underlying geometry. We provide open source implementations of our network layout technique and prototype cerebral artery visualization tool. We demonstrate the robustness of our technique by successfully laying out 61 open source brain scans. We evaluate the effectiveness of our layout through a mixed methods study with three neuroradiologists. In a controlled experiment our study participants used CerebroVis and a conventional 3D visualization to examine real cerebral artery imaging data and to identify a simulated intracranial artery stenosis. Participants were more accurate at identifying stenoses using CerebroVis (odds ratio 2.5, absolute risk difference 13%). More broadly, we discuss the applications of our design approach to a general design paradigm we call *Abstraction with Context*. A free copy of this paper, the evaluation stimuli and data, and source code are available at osf.io/e5sxt

Index Terms—Network Visualization, Spatial Context, Abstract Design, Flow Network, Medical Imaging, Cerebral Arteries.

1 Introduction

Arteries in the human brain form a network of blood flow, and a blockage or leakage in this network can lead to life-threatening cerebrovascular conditions such as a stroke or aneurysm. Strokes alone are the fifth leading cause of death as well as a leading cause of serious long-term disability in the United States, and is globally the second leading cause of death after heart disease [32]. Early detection and diagnosis of these conditions is essential for effective life-saving treatment. Conventional diagnostics rely on an expert neuroradiologist identifying vascular abnormalities through examination of medical images (e.g., CTA, MRA). This data is commonly rendered in 3D in order to assist the doctor with identification of the abnormalities. However, prior research indicates that existing representations of the 3D cerebral arteries—e.g., isosurface, volume rendering, and Maximum Intensity Projection (MIPS)—introduce visual artifacts and task performance challenges such as overplotting/occlusion [19], false impression of geometry [19], and excessive artery bends.

In this design study, we present a novel 2D visualization of the cerebral artery system designed to assist doctors in the identification of cerebrovascular abnormalities. Inspired by existing visualization research which has demonstrated the effectiveness of 2D representations for spatial search tasks in other medical imaging cases, e.g., cardiovascular arteries [6] and connectomics [33], we present a novel 2D abstract representation of the cerebral arteries. To our knowledge, this is the first attempt to approach the cerebrovascular diagnostics tasks faced by neuroradiologists from the perspective of network science and using an abstract 2D visual encoding.

In this paper we first offer a novel framing of cerebral arteries using network theory. Next, we characterize the domain goals and present them as network analysis tasks. In an iterative user-centered design with

- *Aditeya Pandey, Harsh Shukla, Cody Dunne, and Michelle Borkin are with Northeastern University. E-mail: {pandey.ad, shukla.h}@husky.neu.edu, {c.dunne, m.borkin}@northeastern.edu*
- *Geoffrey S. Young, Amir A. Zamani, Liangge Hsu, and Raymond Huang are with Brigham and Women's Hospital. E-mail: {gsyoung, azamani, lhsu1, ryhuang}@bwh.harvard.edu*
- *Lei Qin is with the Dana-Farber Cancer Institute. E-mail: lqin2@partners.org*

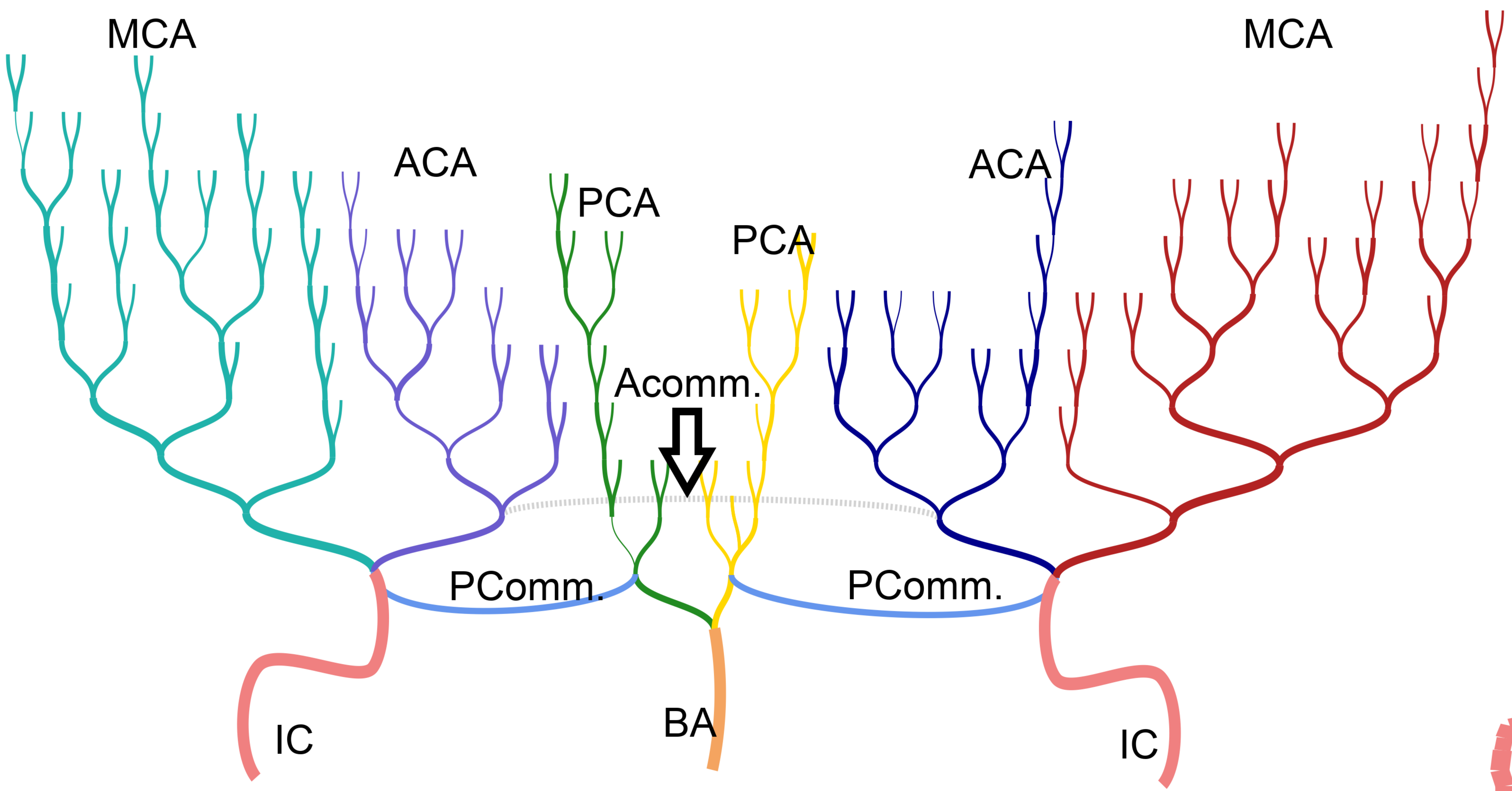
Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

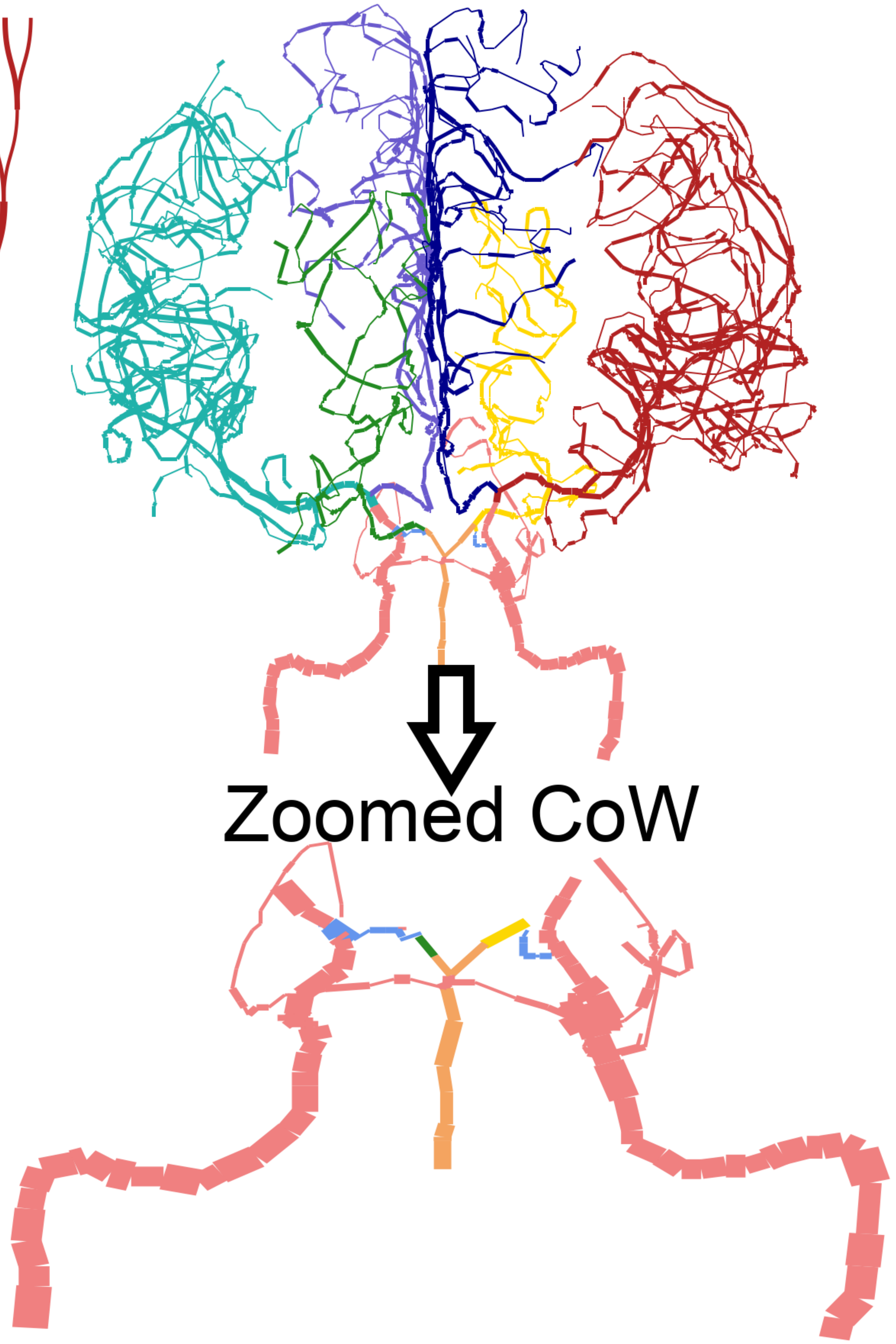
CS 7260 Fall 2017:
Visualization for
Network Science

Pandey et al. VIS 2019

Isosurface Projection
CerebroVis
Left - MCA
Left - ACA
Left - PCA
Right - ACA
Right - MCA

Project Example — CerebroVis



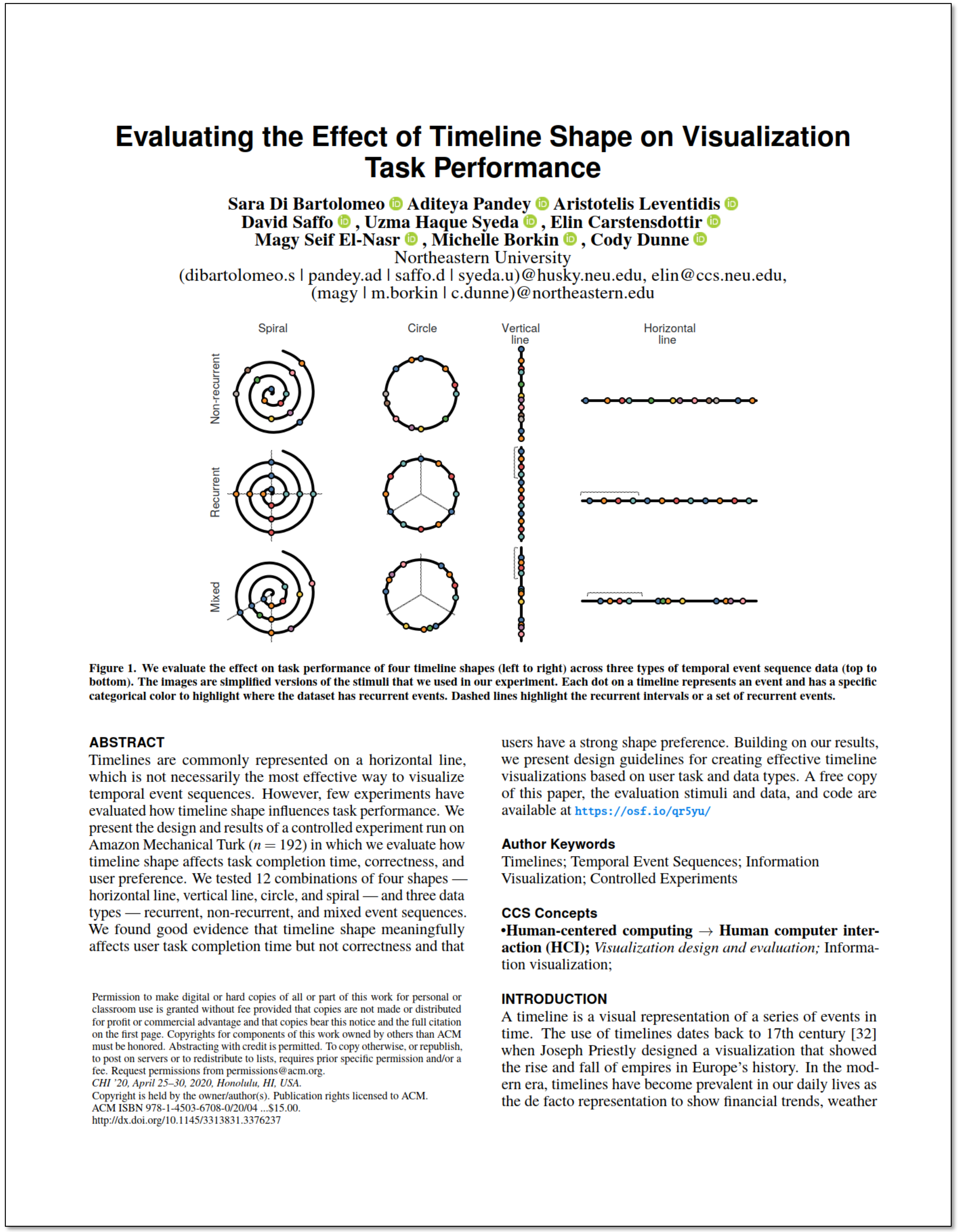


Pandey et al. VIS 2019

Example of a Successful *Differentiated* Course Project

(Requires prior instructor approval to waive / alter requirements)

Project Example — Diversiform Timelines



Evaluating the Effect of Timeline Shape on Visualization Task Performance

Sara Di Bartolomeo , Aditeya Pandey , Aristotelis Leventidis
David Saffo , Uzma Haque Syeda , Elin Carstensdottir
Magy Seif El-Nasr , Michelle Borkin , Cody Dunne
Northeastern University
(dibartolomeo.s | pandey.ad | saffo.d | syeda.u)@husky.neu.edu, elin@ccs.neu.edu,
(magy | m.borkin | c.dunne)@northeastern.edu

Figure 1. We evaluate the effect on task performance of four timeline shapes (left to right) across three types of temporal event sequence data (top to bottom). The images are simplified versions of the stimuli that we used in our experiment. Each dot on a timeline represents an event and has a specific categorical color to highlight where the dataset has recurrent events. Dashed lines highlight the recurrent intervals or a set of recurrent events.

ABSTRACT

Timelines are commonly represented on a horizontal line, which is not necessarily the most effective way to visualize temporal event sequences. However, few experiments have evaluated how timeline shape influences task performance. We present the design and results of a controlled experiment run on Amazon Mechanical Turk ($n = 192$) in which we evaluate how timeline shape affects task completion time, correctness, and user preference. We tested 12 combinations of four shapes — horizontal line, vertical line, circle, and spiral — and three data types — recurrent, non-recurrent, and mixed event sequences. We found good evidence that timeline shape meaningfully affects user task completion time but not correctness and that users have a strong shape preference. Building on our results, we present design guidelines for creating effective timeline visualizations based on user task and data types. A free copy of this paper, the evaluation stimuli and data, and code are available at `https://osf.io/qr5yu/`

Author Keywords
Timelines; Temporal Event Sequences; Information Visualization; Controlled Experiments

CCS Concepts
•**Human-centered computing** → **Human computer interaction (HCI);** *Visualization design and evaluation;* Information visualization;

INTRODUCTION

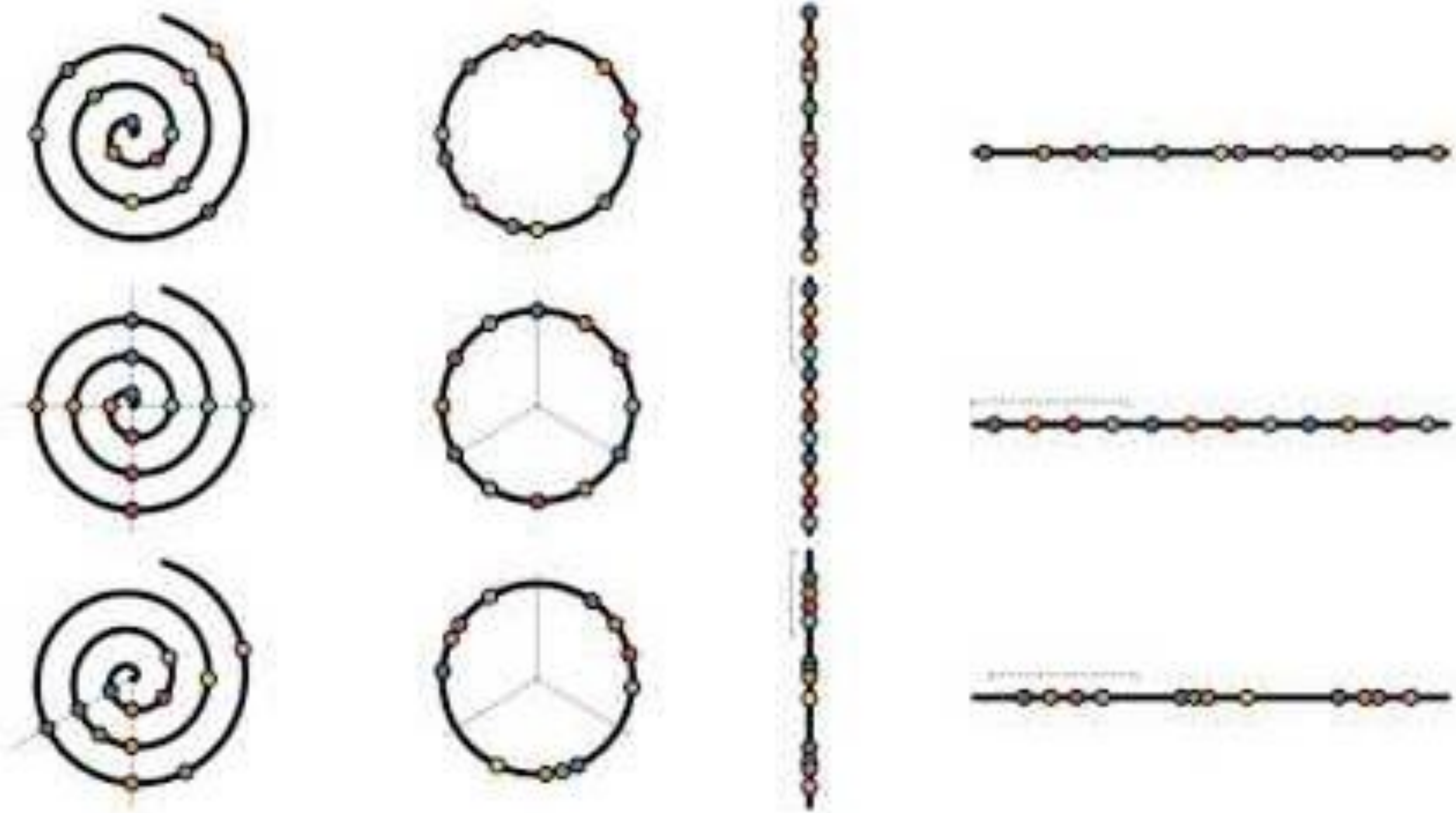
A timeline is a visual representation of a series of events in time. The use of timelines dates back to 17th century [32] when Joseph Priestly designed a visualization that showed the rise and fall of empires in Europe's history. In the modern era, timelines have become prevalent in our daily lives as the de facto representation to show financial trends, weather

CHI '20, April 25–30, 2020, Honolulu, HI, USA.

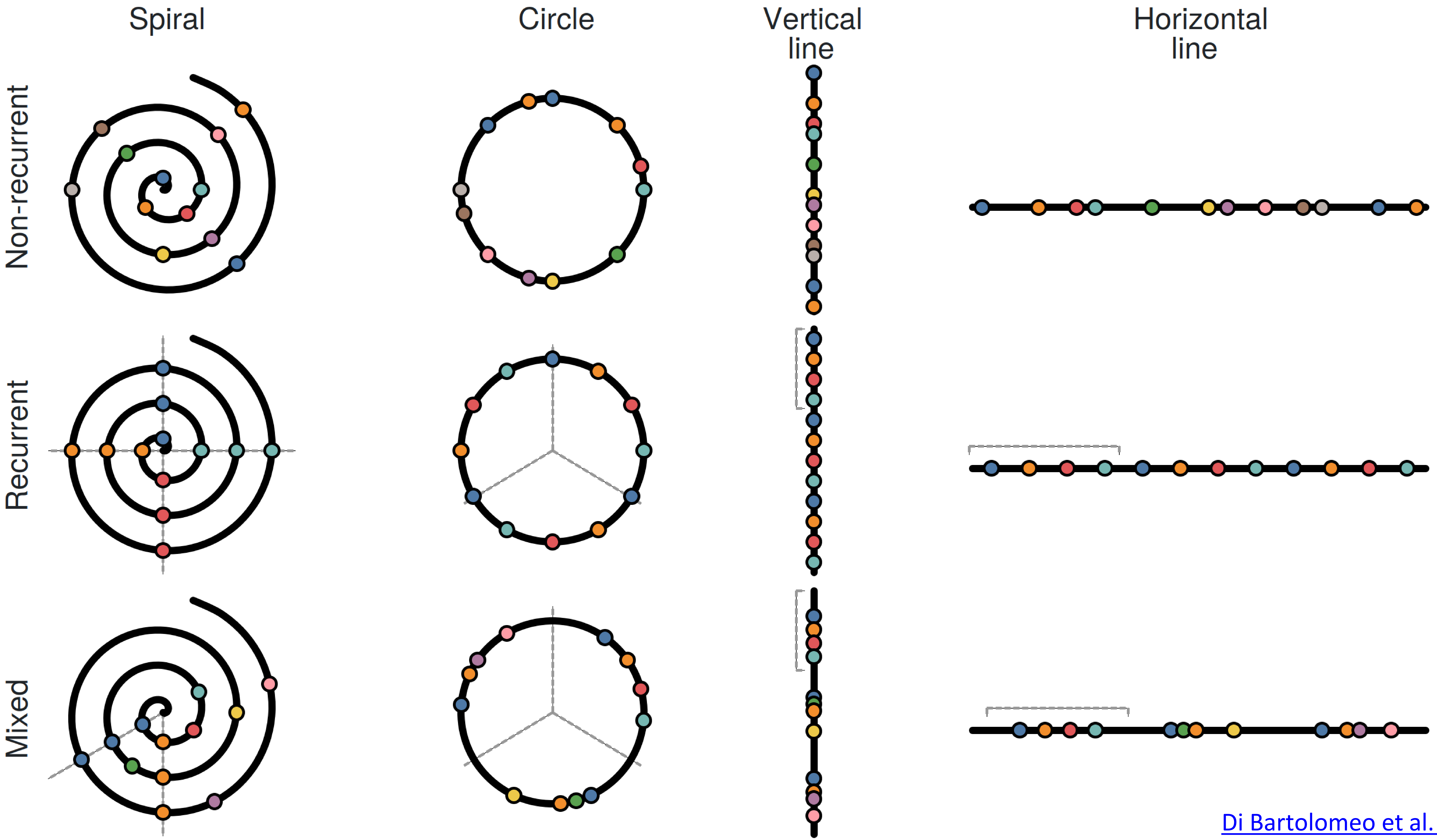
ACM ISBN 978-1-4503-6708-0/20/04 ...$15.00.
http://dx.doi.org/10.1145/3313831.3376237

CS 7340 Fall 2018:
Theory and Methods in Human Computer Interaction

[Di Bartolomeo et al. CHI 2020](#)



Project Example — Diversiform Timelines



[Di Bartolomeo et al. CHI 2020](#)

Project Ideas:

VIS + X

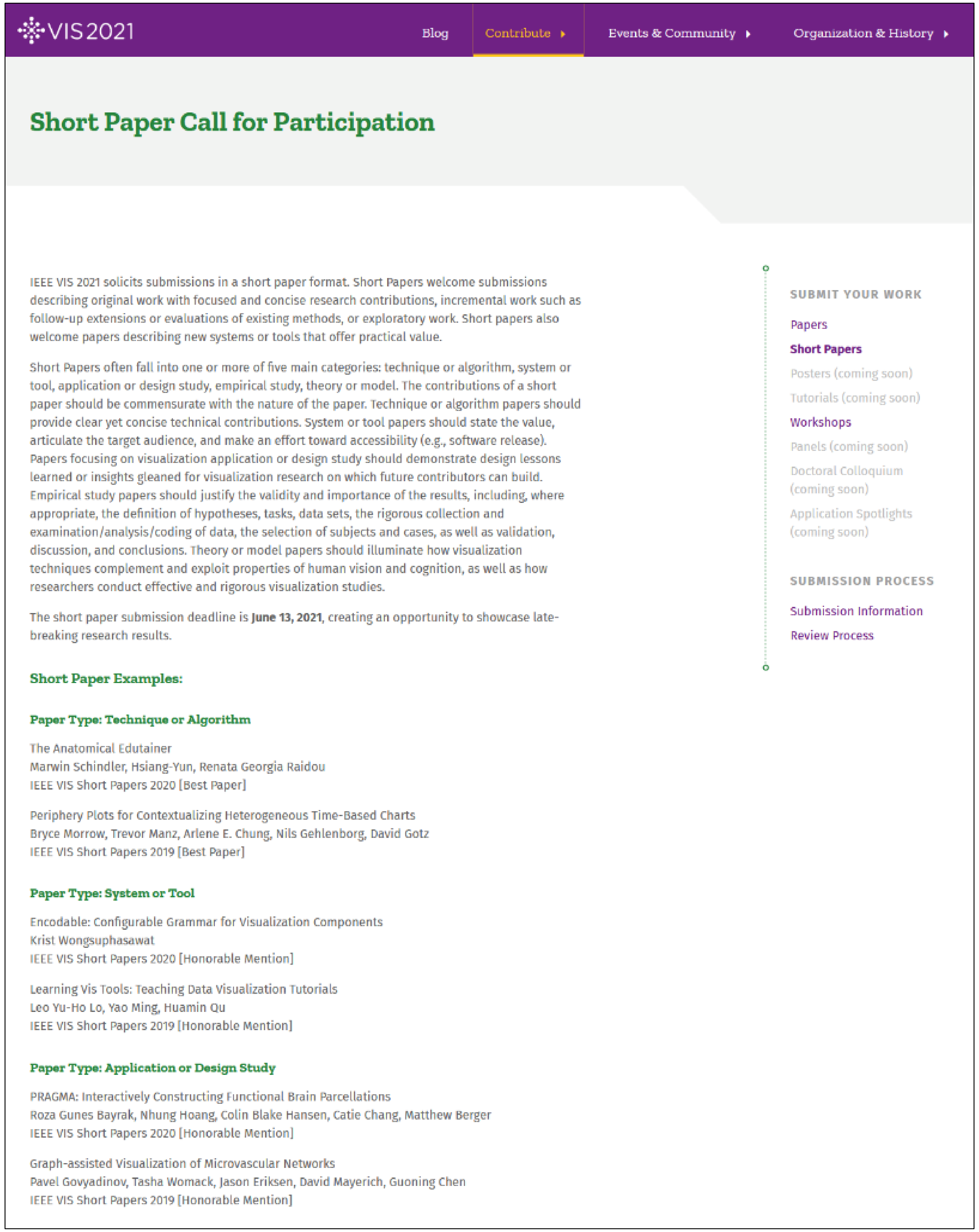
Where X = a CS subfield (ML | SEC | NLP | HCC | GAM | NS | SYS | …)

OR

Where X = a domain application (health, energy, transportation, astronomy, crime…)

Potential Venue:
IEEE VIS 2021 Short Papers

Deadline 2021-06-13



Short Paper Call for Participation

IEEE VIS 2021 solicits submissions in a short paper format. Short Papers welcome submissions describing original work with focused and concise research contributions, incremental work such as follow-up extensions or evaluations of existing methods, or exploratory work. Short papers also welcome papers describing new systems or tools that offer practical value.

Short Papers often fall into one or more of five main categories: technique or algorithm, system or tool, application or design study, empirical study, theory or model. The contributions of a short paper should be commensurate with the nature of the paper. Technique or algorithm papers should provide clear yet concise technical contributions. System or tool papers should state the value, articulate the target audience, and make an effort toward accessibility (e.g., software release). Papers focusing on visualization application or design study should demonstrate design lessons learned or insights gleaned for visualization research on which future contributors can build. Empirical study papers should justify the validity and importance of the results, including, where appropriate, the definition of hypotheses, tasks, data sets, the rigorous collection and examination/analysis/coding of data, the selection of subjects and cases, as well as validation, discussion, and conclusions. Theory or model papers should illuminate how visualization techniques complement and exploit properties of human vision and cognition, as well as how researchers conduct effective and rigorous visualization studies.

The short paper submission deadline is **June 13, 2021**, creating an opportunity to showcase late-breaking research results.

Short Paper Examples:

Paper Type: Technique or Algorithm

The Anatomical Edutainer
Marwin Schindler, Hsiang-Yun, Renata Georgia Raidou
IEEE VIS Short Papers 2020 [Best Paper]

Periphery Plots for Contextualizing Heterogeneous Time-Based Charts
Bryce Morrow, Trevor Manz, Arlene E. Chung, Nils Gehlenborg, David Gotz
IEEE VIS Short Papers 2019 [Best Paper]

Paper Type: System or Tool

Encodable: Configurable Grammar for Visualization Components
Krist Wongsuphasawat
IEEE VIS Short Papers 2020 [Honorable Mention]

Learning Vis Tools: Teaching Data Visualization Tutorials
Leo Yu-Ho Lo, Yao Ming, Huamin Qu
IEEE VIS Short Papers 2019 [Honorable Mention]

Paper Type: Application or Design Study

PRAGMA: Interactively Constructing Functional Brain Parcellations
Roza Gunes Bayrak, Nhung Hoang, Colin Blake Hansen, Catie Chang, Matthew Berger
IEEE VIS Short Papers 2020 [Honorable Mention]

Graph-assisted Visualization of Microvascular Networks
Pavel Govyadinov, Tasha Womack, Jason Eriksen, David Mayerich, Guoning Chen
IEEE VIS Short Papers 2019 [Honorable Mention]

VIS 2021 Short Paper CFP

PROJECTS

In-class project pitches: F 2021-02-12

What questions do you have for me?

Upcoming Assignments & Communication

https://northeastern.instructure.com/courses/63405/assignments/syllabus

Look at the upcoming assignments and deadlines regularly!

- Textbook, Readings, & Reading Quizzes — Variable days
- In-Class Activities — 11:59pm same day as class
  - This F: Lecture & in-class activity on Tableau
  - Next F: Lecture & in-class activity on D3
- Assignments & Projects— Generally due R 11:59pm
  - This R (2 days): Assignment 2 due
  - Next R (9 days): Assignments 3a, 3b due
  - Next-Next R (16 days): Project 1 (pitches) due
- Project Overview

Everyday Required Supplies:

- 5+ colors of pen/pencil
- White paper
- Laptop and charger

Use Canvas Discussions for general questions, email the TAs/S-LTA/instructor for questions specific to you: codydunne-and-tas@ccs.neu.edu. Include links!