INTRODUCTION

cs2420 | Introduction to Algorithms and Data Structures | Spring 2015

today…

-meet the teaching staff

-what is this course about?

-why should you care?

-nuts & bolts

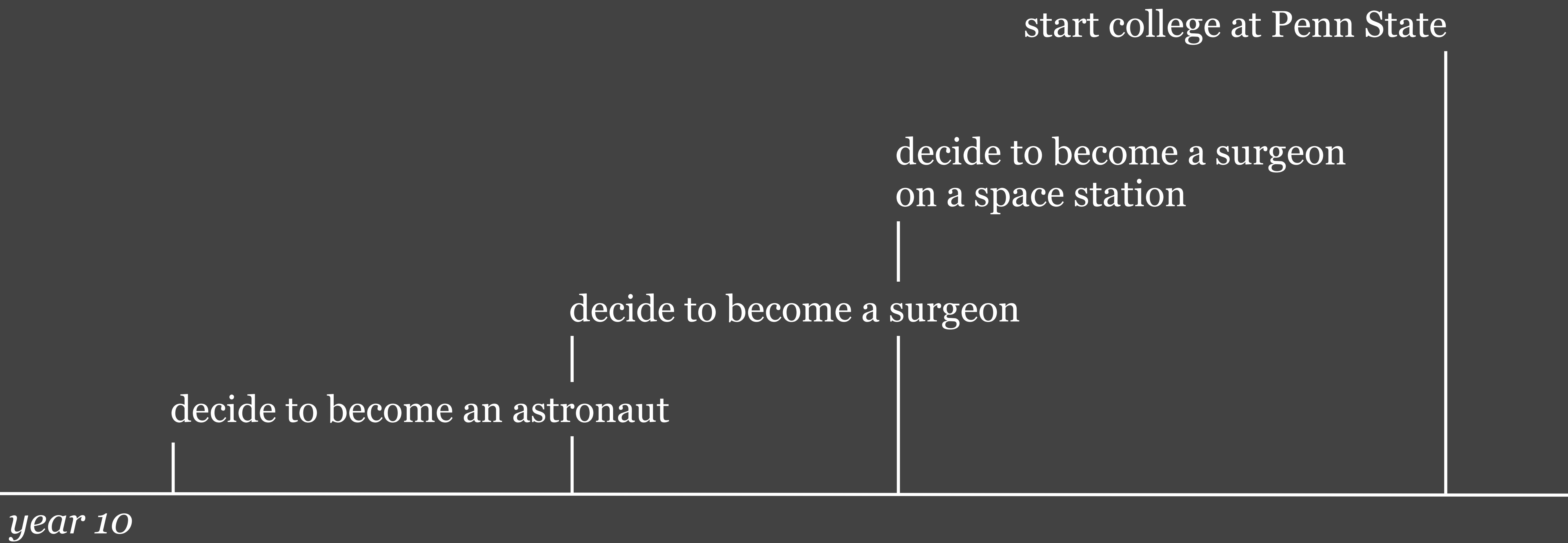
-good coding practice

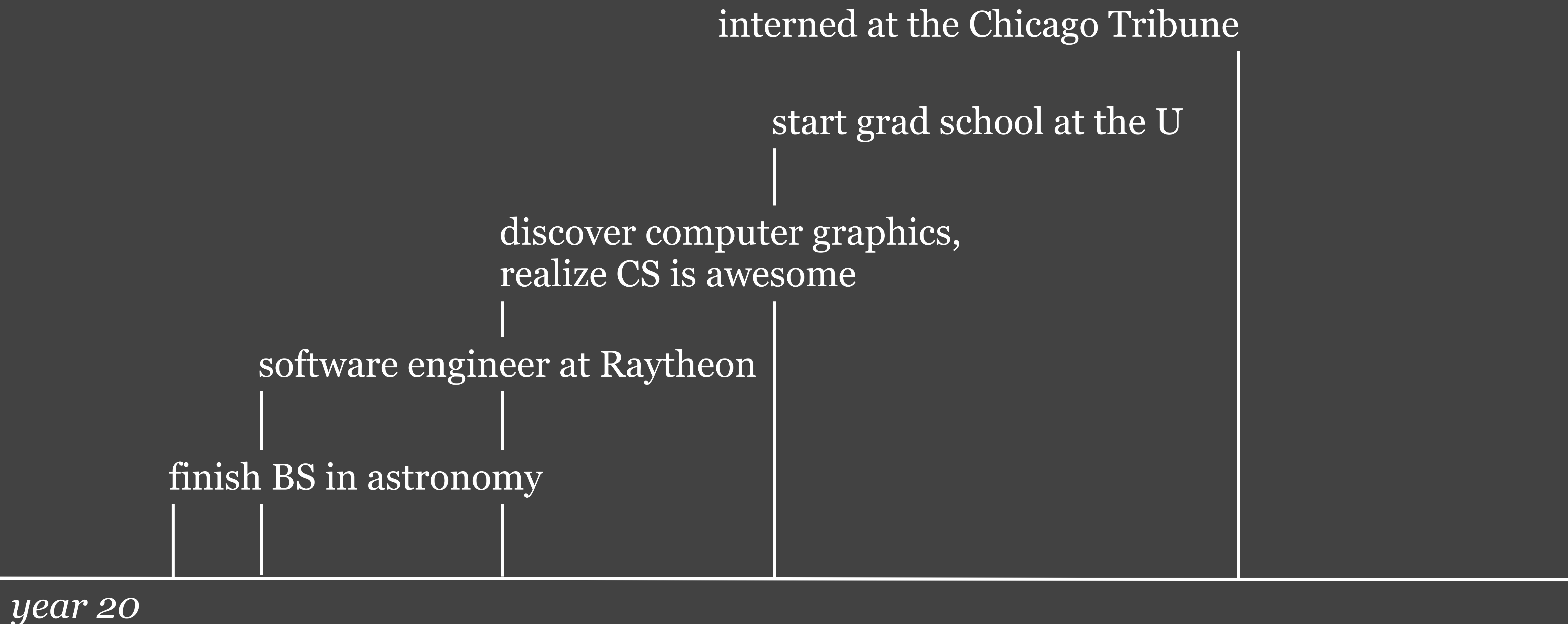
meet the teaching staff

dad buys a Commodore64

born in Martinsville, VA

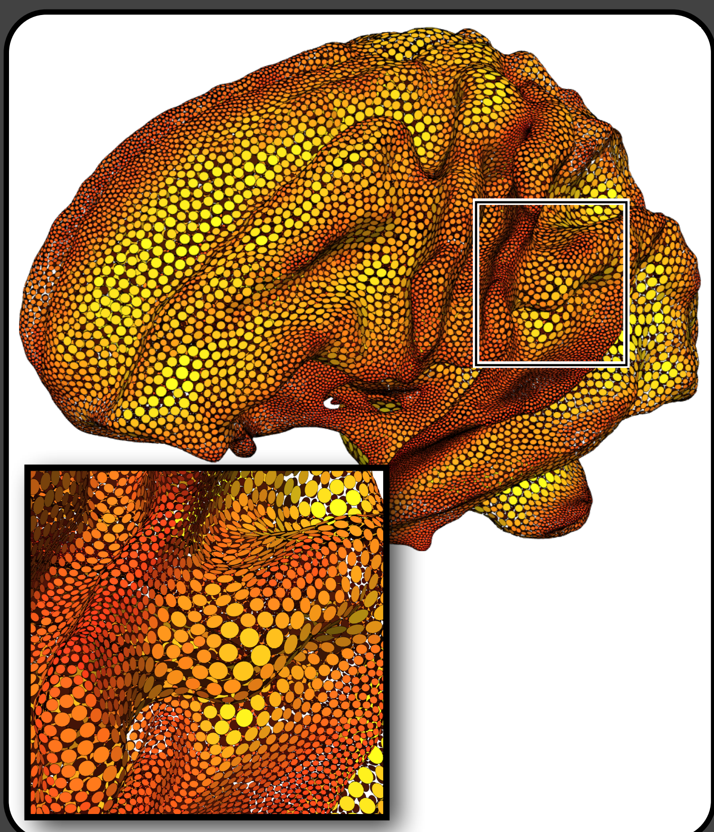
year 0


start college at Penn State
decide to become a surgeon on a space station
decide to become a surgeon
decide to become an astronaut
year 10

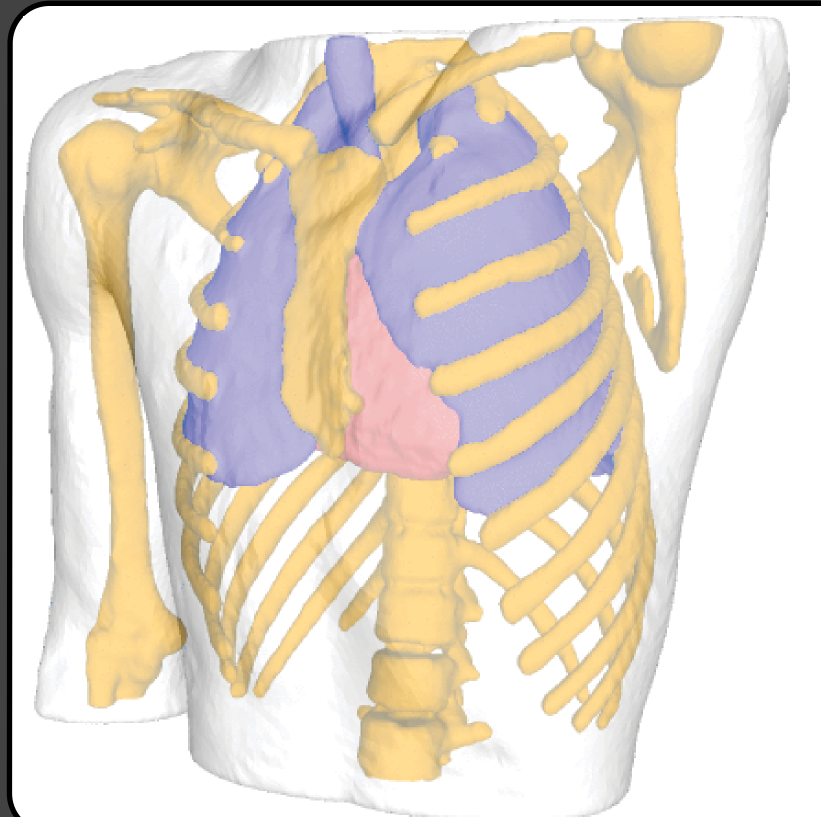

interned at the Chicago Tribune
start grad school at the U
discover computer graphics,
realize CS is awesome
software engineer at Raytheon
finish BS in astronomy
year 20

finish PhD in computer science

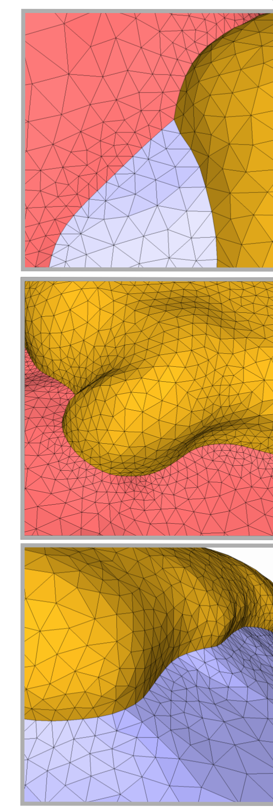
year 30



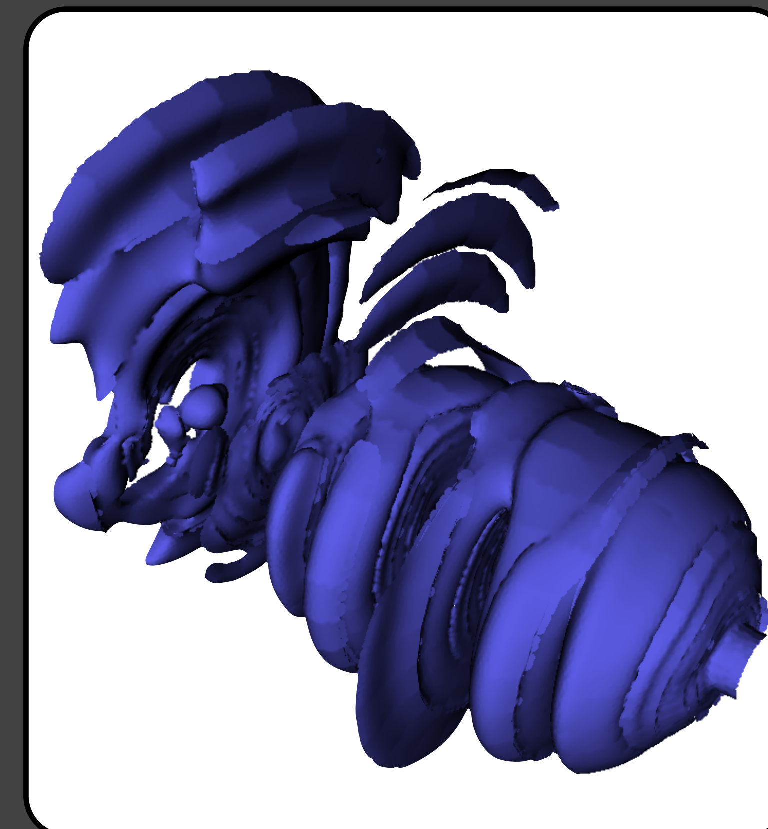
Robust Particle Systems for Curvature Dependent Sampling of Implicit Surfaces.
M. Meyer et al., SMI 2005.



Topology, Accuracy, and Quality of Isosurface Meshes Using Dynamic Particles.
M. Meyer et al., Vis 2007.



Particle-based Sampling and Meshing of Multimaterial Volumes.
M. Meyer et al., Vis 2008.



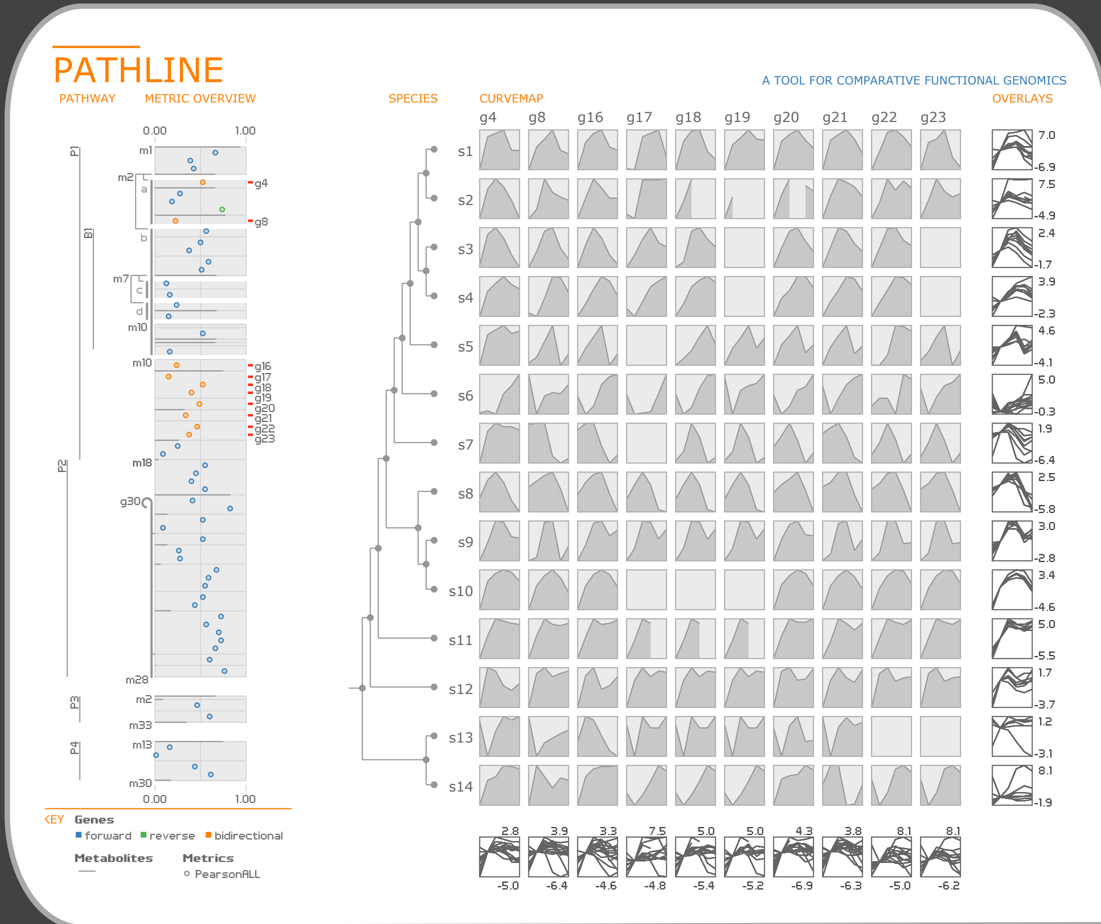
Particle Systems for Efficient and Accurate High-Order Finite Element Visualization
M. Meyer et al., TVCG 2006.

postdoc at Harvard University

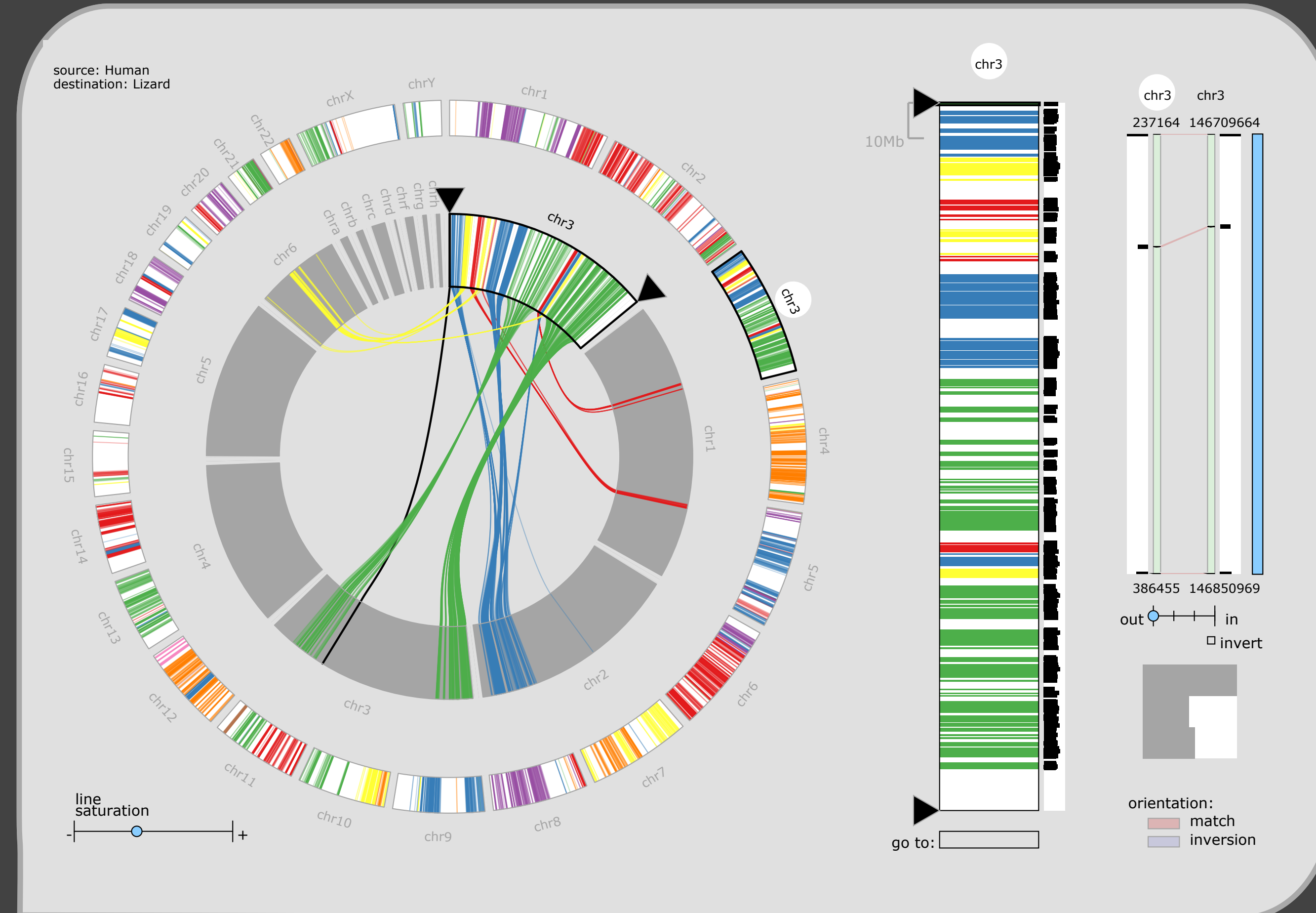
finish PhD in computer science

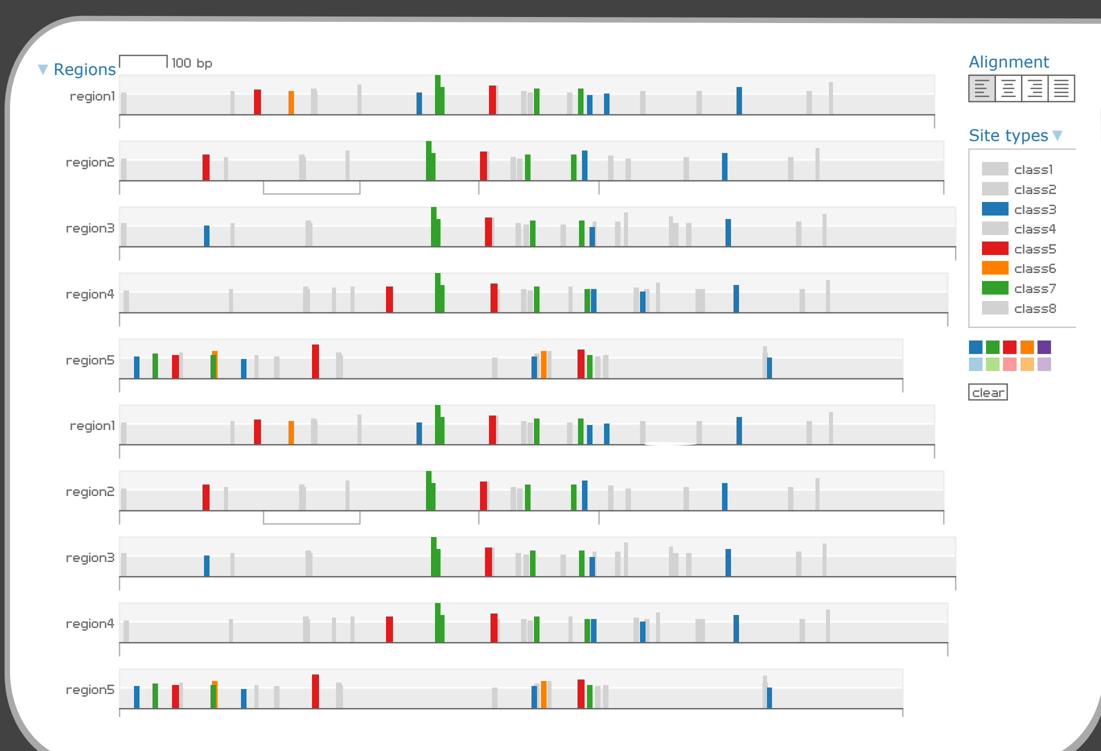
year 30

M. Meyer et al., EuroVis 2010. **Pathline**

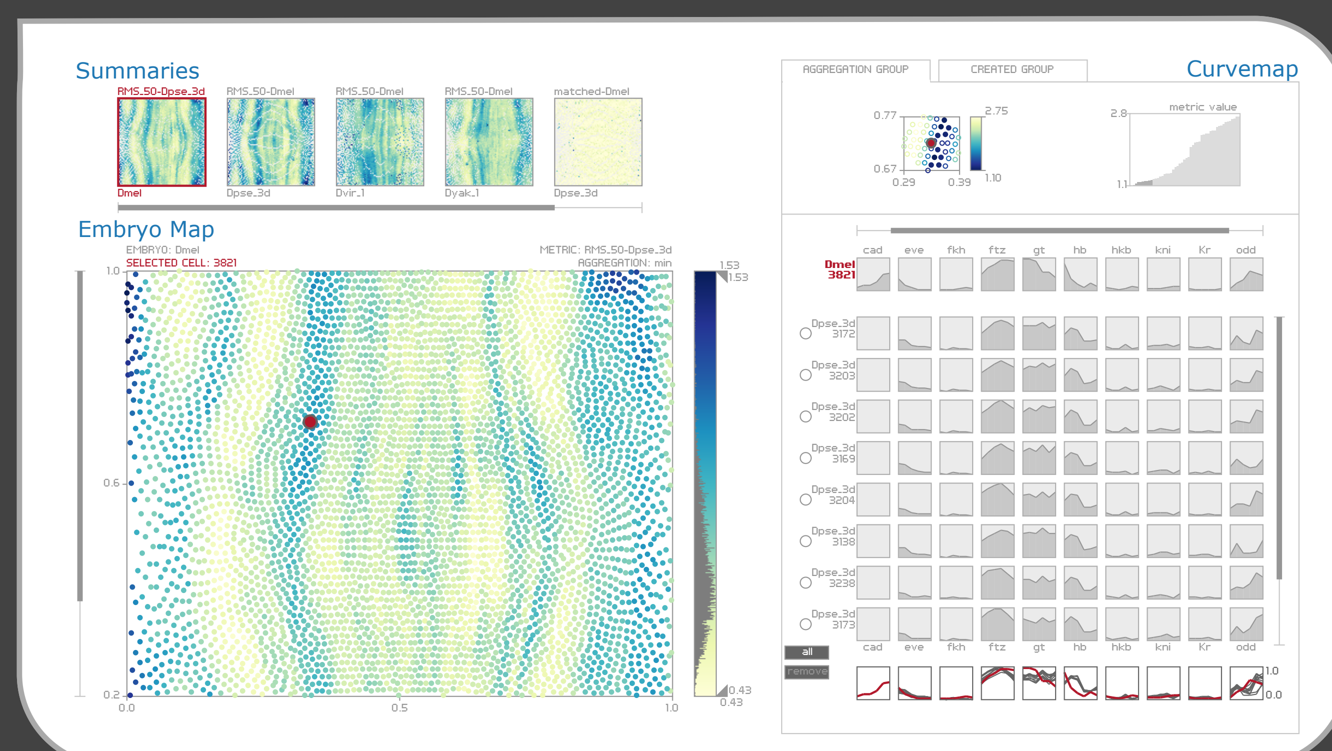


MizBee *M. Meyer et al., InfoVis 2009.*





InSite



MulteeSum *M. Meyer et al., InfoVis 2010.*

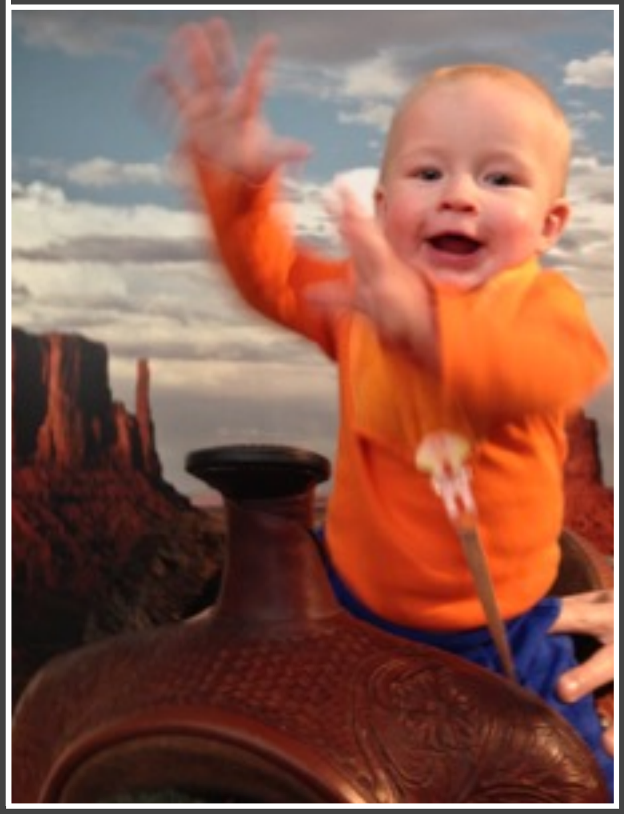
assistant professor at the U in
School of Computing and SCI

WE ARE HERE

postdoc at Harvard University

finish PhD in computer science

year 30



Visualization Design Lab

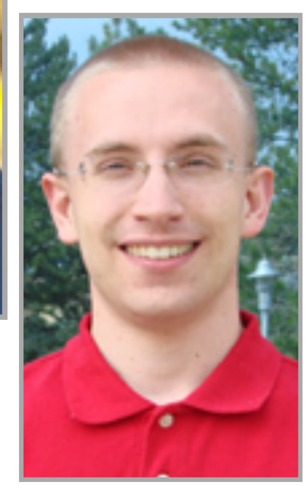
@ the University of Utah

Miriah Meyer
assistant professor

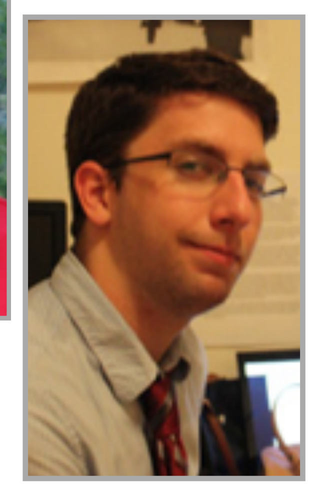
Alex Bigelow, PhD student

Ethan Kerzner, PhD student

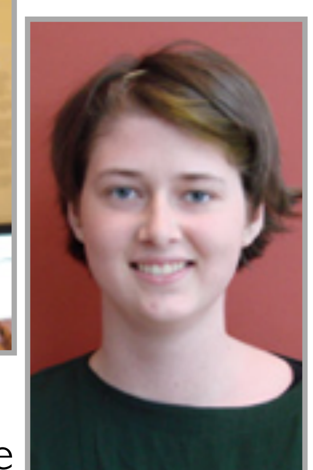
Nina McCurdy, PhD student



Sean McKenna, PhD student



Sam Quinan, PhD student



Zella Urquhart, undergraduate

Kris Zygmunt, PhD student

technique-driven what ***should*** we build?

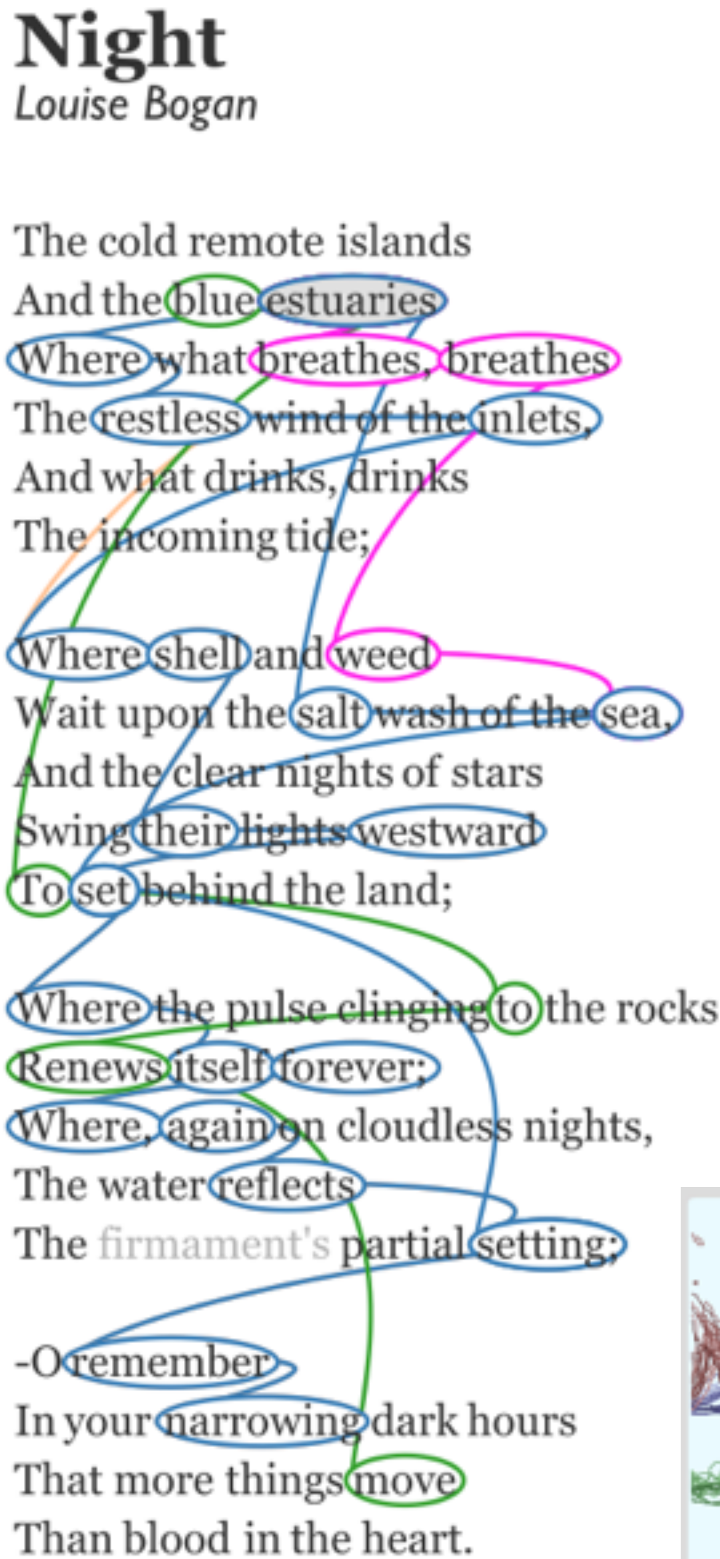
what ***can*** we build? **problem-driven**

target specific domain problems

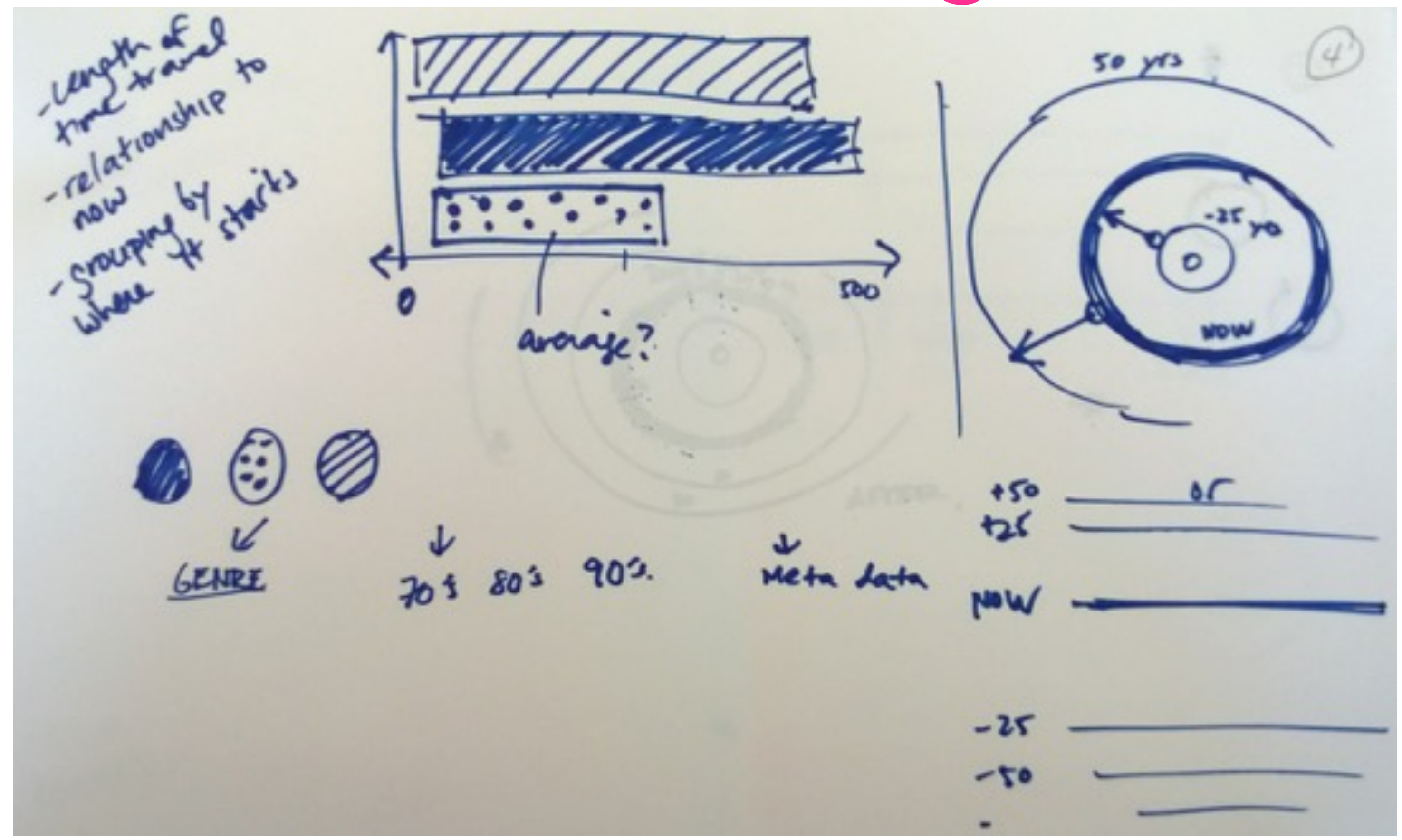
close collaboration with domain experts

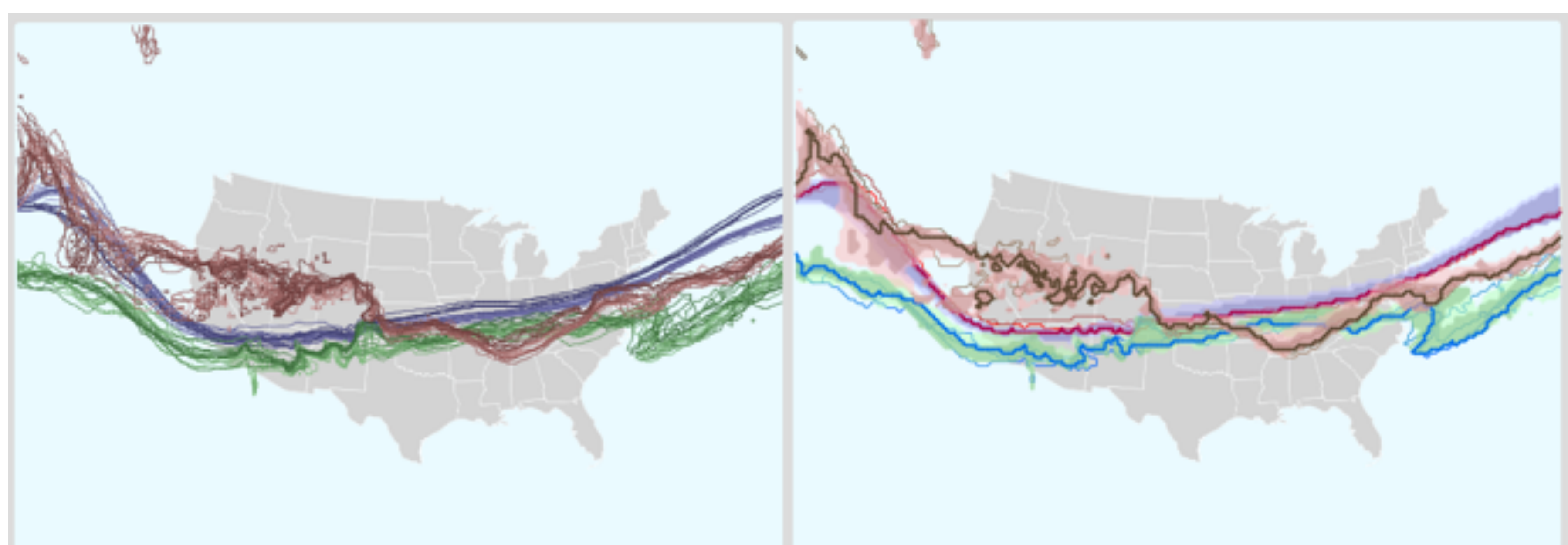
rapid, iterative prototyping

refine visualizations through user feedback poetry visualization



visualization design tools





visualizing weather forecasts

what is this course about?


cost
shows
operator
set
reference
returns
recursive
element
integer
types
instance
null
hash
vertex
last
recursion
defined
lines
analysis
position
collection
operation
linear
true
double
iterator
objects
methods
case
queue
nodes
heap
binary
left
array
object
implement
log
tree
trees
however
part
event
sort
note
code
time
implementation
size
stack
collections
random
interface
main
generic
see
void
contains
node
value
method
line
int
private
function
run
calls
statement
test
show
items
type
java
average
write
api
string
algorithm
search
example
section
elements
access
sequence
input
arraylist
new
classes
insert
operations
suppose
item
routine
program
empty
make
order
first
static
basic
theorem
algorithms
root
running
add
list
data
simple
large
linked
right
many
current
call
following
remove
sorting
boolean
single
result
may
sum
base
find
path
need
graph
file
key
table
insertion
priority
child
final
next

fundamentals of coding

-how to analyze your algorithms
  -improve efficiency
  -make good coding choices

-recursion
  def. Recursive loop: See “recursive loop”.

-basic sorting algorithms
  -one of most studied operations in CS

-elementary data structures
  -provide mechanism for what we can do with data

why should you care?

why do(n't) algoritms matter?

-many different ways to solve a problem

  -one method may take 1ms longer per item….

  -computers operate on LARGE numbers of items

    -millions

    -billions

    -… or more

$1x10^{12}$ `*(minuscule amount of time) = large amount of time`

-this matters, but not as much as **algorithmic complexity**

N

- we refer to unspecified integer quantities as **N**
  - **N** is the problem size
    - *sorting an array of* **N** *numbers*
    - *searching for an item in a set of* **N** *items*
    - *inserting an item into a set of* **N** *items*

- amount of work done for these operations usually depends on **N**
  - work required is a **function** of **N**

why DO algorithms matter?

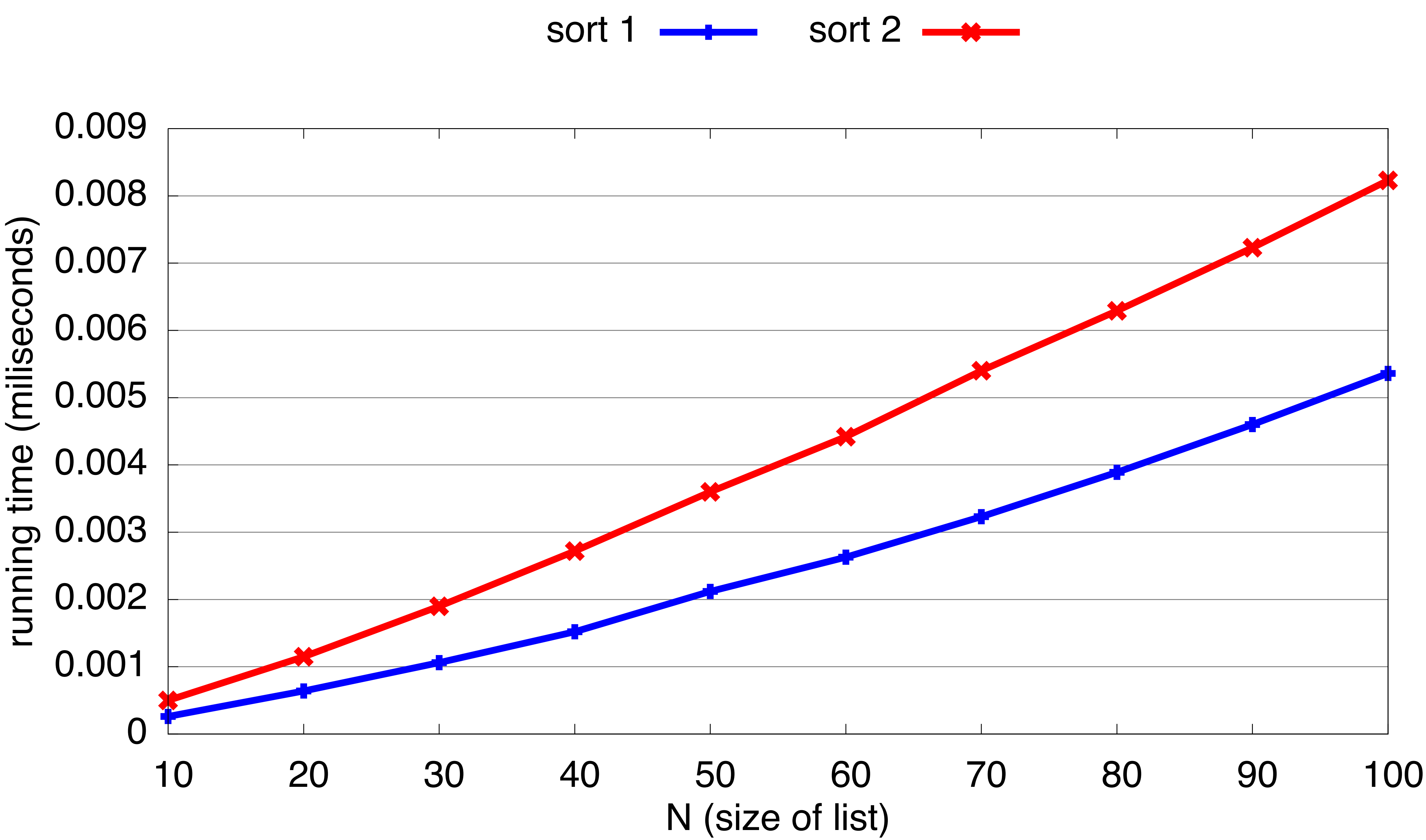
-algorithms don't always require **N** steps for **N** items
  -could be linear, quadratic, logarithmic, …
  -called the **complexity** of an algorithm

-**N**^{2} is much MUCH bigger than **N**
  -what if **N** == 1 million?

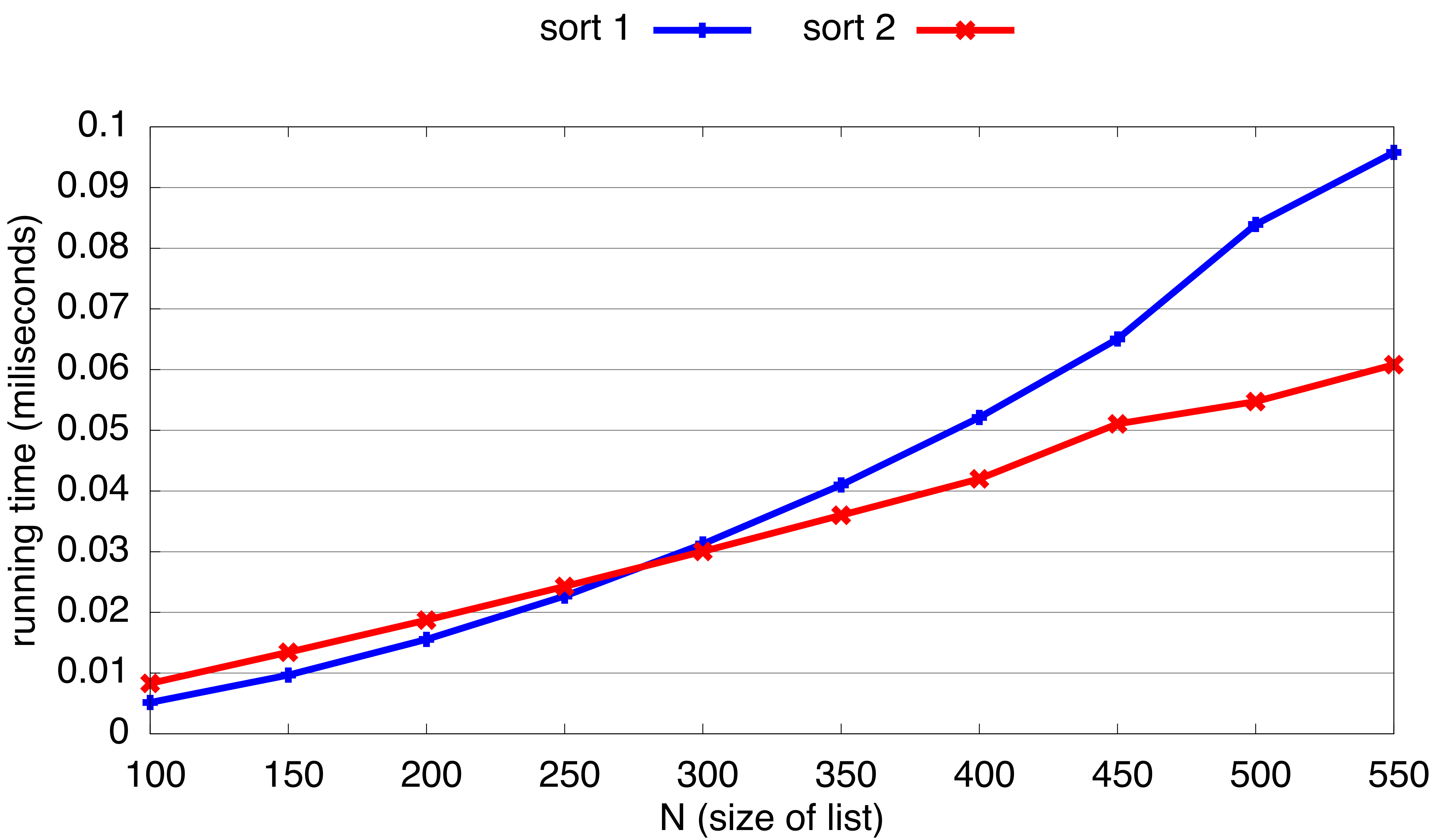
-we only care about large **N**

`sort1` **versus** `sort2`

small N


sort 1
sort 2
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
running time (miliseconds)
10
20
30
40
50
60
70
80
90
100
N (size of list)

medium N

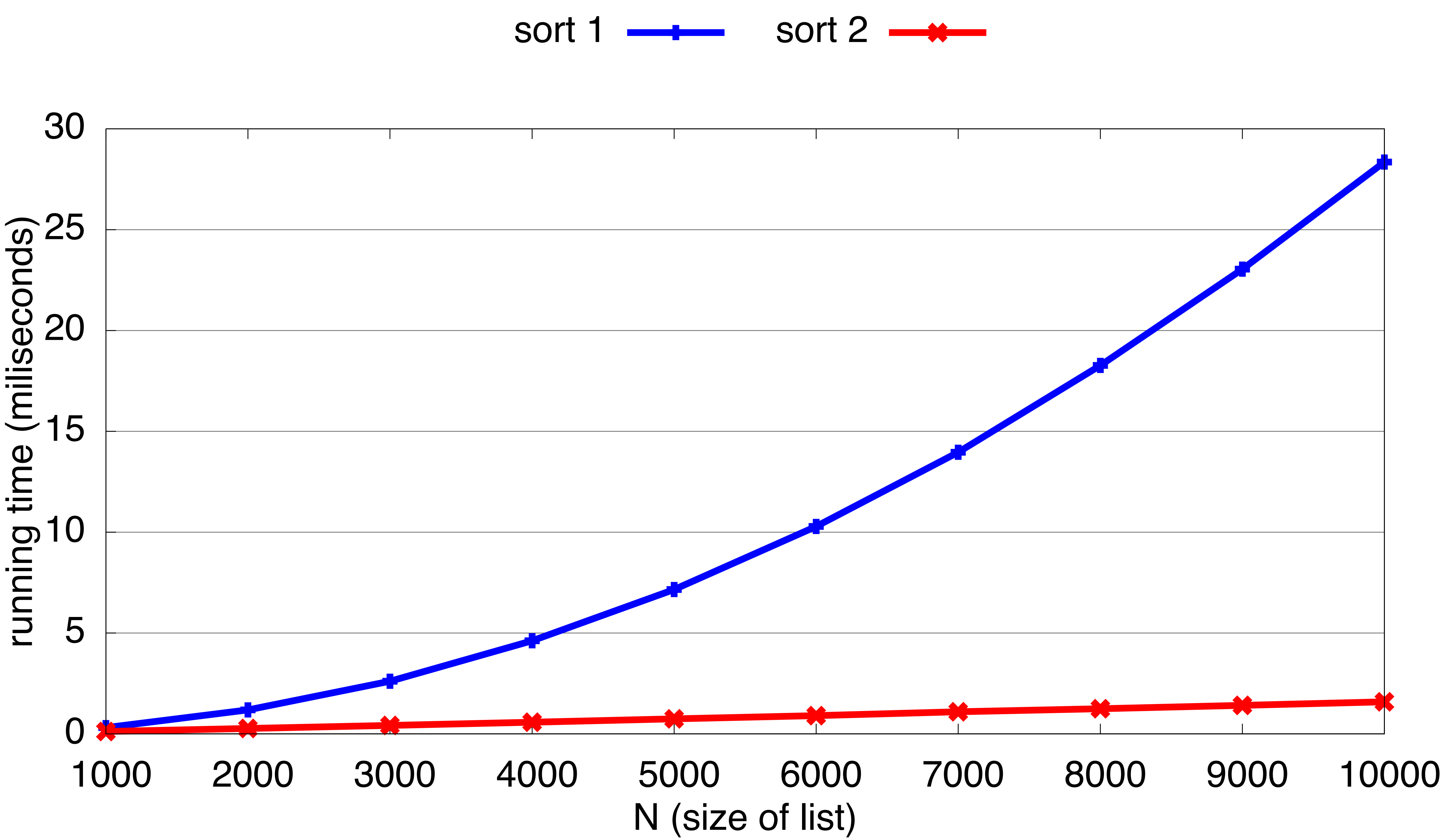


sort 1
sort 2
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
running time (miliseconds)
100
150
200
250
300
350
400
450
500
550
N (size of list)

large N

TAKE AWAY:

AS N BECOMES LARGE COMPLEXITY MATTERS!



sort1

```
void sort1(int[] arr) {
  for(int i = 0; i < arr.length-1; i++) {
    int j, minIndex;
    for(j = i+1, minIndex = i; j < arr.length; j++)
      if(arr[j] < arr[minIndex])
        minIndex = j;
    swap(arr, i, minIndex);
  }
}
```

best case: O(**N**^{2})

average case: O(**N**^{2})

worst case: O(**N**^{2})

sort2

```
void sort2(int[] arr, int beg, int end) {
  if (end > beg + 1) {
    int piv = arr[beg], l = beg + 1, r = end;
    while (l < r) {
      if (arr[l] <= piv)
        l++;
      else
        swap(arr, l, --r);
    }
    swap(arr, --l, beg);
    sort2(arr, beg, l);
    sort2(arr, r, end);
  }
}
```

best case: O(**N** log **N**)

average case: O(**N** log **N**)

worst case: O(**N**^{2})

complexity matters…

the difference between 1ms and 30ms doesn't matter if you run the algorithm once…

… but this is rarely the case in computing

~30ms/frame for all algorithms in a game

~1 billion Google searches per day, every day

data structures & algorithms matter

-for large N, the difference between $O(N \log N)$ and $O(N^2)$ is HUGE!

-is running time the only measure of efficiency?

-transitioning from **cs1410** to **cs2420**

cs1410: correct algorithms (or just code?) to solve problems

cs2420: correct algorithms analyzed for efficiency; advanced structures for intuitive organization of data

nuts & bolts

eng.utah.edu

CS2420 | Introduction to Algorithms and Data Structures | Spring 2015

INSTRUCTOR: Miriah Meyer

LECTURES: T/Th 10:45am-12:05pm
PLACE: L101 WEB

LABS: M various times
PLACE: 3225 MEB

OFFICE HRS: T 1-3pm, WEB 4887

SCHEDULE | TAs | SYLLABUS | LECTURES:LABS:ASSIGNMENTS

This course provides an introduction to tools found throughout computer science -- basic algorithms and data structures that lend themselves naturally to computational problem solving, as well as the problem of engineering computational efficiency in to programs. Students will gain an understanding of classical algorithms (including sorting, searching, tree and graph traversal), and data structures (including linked-lists, trees, graphs, hash tables, and heaps), and an introduction to parallel computing. Students will complete extensive programming assignments that require the implementation and testing of these concepts.

SCHEDULE

WEEK	DATE	TOPIC	EXAMS
1	1/13 & 1/15	Introduction & Java review	
2	1/20 & 1/22	OO & generic programming	
3	1/27 & 1/29	Algorithm analysis & data structures	
4	2/3 & 2/5	Basic sorting	
5	2/10 & 2/12	Recursive sorting	
6	2/17 & 2/19	Linked lists	Midterm 1 on Tues
7	2/24 & 2/26	Stacks & queues	

- programming homework
  - one assignment per week
  - must be done with a partner, except this week
  - all programming to be done in Java 7

- exams
  - two midterms, held during class time
  - one final

- labs
  - held most Mondays
  - practice with class topics, 1-on-1 help from TA's
  - **required**: they count towards your final grade!

-assignments

-grades

-student-to-student discussion forum

-announcements

eclipse

-help sessions this Friday

- -9:40am | 10:45am | 11:50am
- -MEB 3225

- -getting started with Eclipse
- -Java refresher
- -assignment posted by Thursday night
    - *-this will not count towards your grade*

good coding practice

the nature of programming

-requires more time than you think

  -more time consuming than 4-credit hours may imply

-when is a program done?

  -when it compiles?

-can the time required to code and debug a program be reduced?

  -YES! by practicing good software engineering

phases of software development

-requirements gathering

-read and understand assignment specs, ask questions

-planning | design | analysis

-outline how to solve a problem, determine algorithms, write pseudocode

-construction

-write code, debug **syntactic** errors

-testing

-test thoroughly to find **semantic** errors and boundary cases

-maintance

using SE in assignments

-careful planning and coding can save hours of debugging

-learn from your mistakes: anticipate errors
  -misspellings, typos, off-by-one errors

-thorough, organized testing will detect more errors

-pay attention to the way you design, code, debug, test — habits form quickly!

testing

-white-box

-test with knowledge of the program's inner-workings — from the programmer's perspective

-unit testing, boundary analysis

-black-box

-test only with knowledge of the program's interface — from the user's perspective

-stress testing

-test-first model

-write acceptance tests before writing any code

good coding style

-benefits the programmer and all other readers of the program

-components:
- -descriptive names (variables, methods, classes)
- -clear expressions, straightforward control flow
- -consistency, conventions, and language idioms
- **-comments!**

-well-written code is often smaller, has fewer errors, and is easier to extend and modify

SE in cs2420

-start practicing good coding style for its own rewards, not just credit

-try applying SE to each assignment

- -learn from development process on previous assignments
- -make necessary improvements on future assignments

-cs3500 (Software Practice 1) will cover SE principles more thoroughly

this week…

- reading
  - chapters 1 & 2

- homework
  - proficiency exam *(do not hand in)*
  - student survey *(due Thursday, Jan 15th at 5pm)*

- no lab
  - optional help sessions on Friday