Programming Languages and Compilers (CS 421)

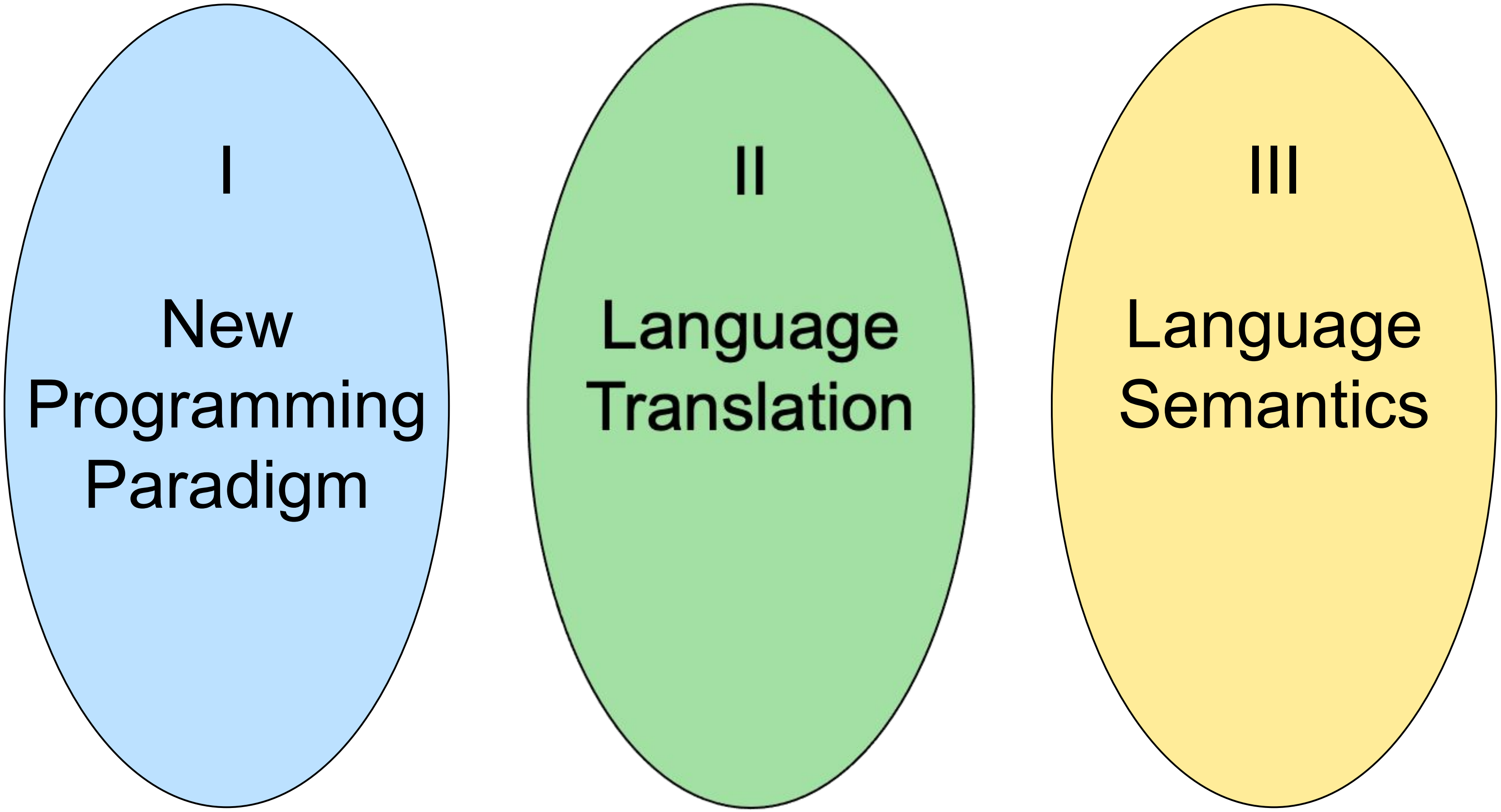
Talia Ringer (they/them)

4218 SC, UIUC

https://courses.grainger.illinois.edu/cs421/fa2023/

Based heavily on slides by Elsa Gunter, which were based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Three Main Topics of the Course



I

New Programming Paradigm

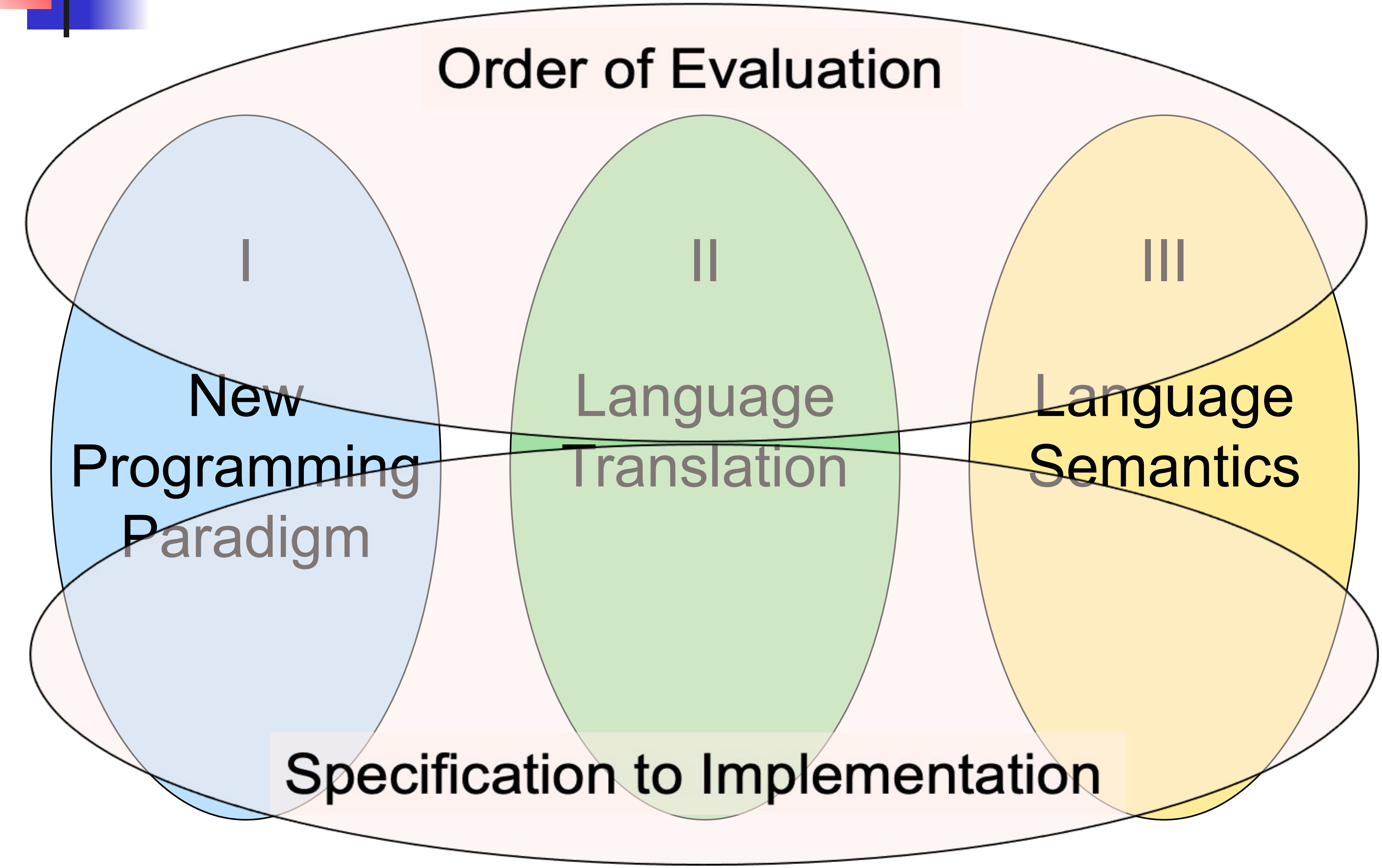
II

Language Translation

III

Language Semantics

Programming Languages & Compilers



Order of Evaluation

I

New Programming Paradigm

II

Language Translation

III

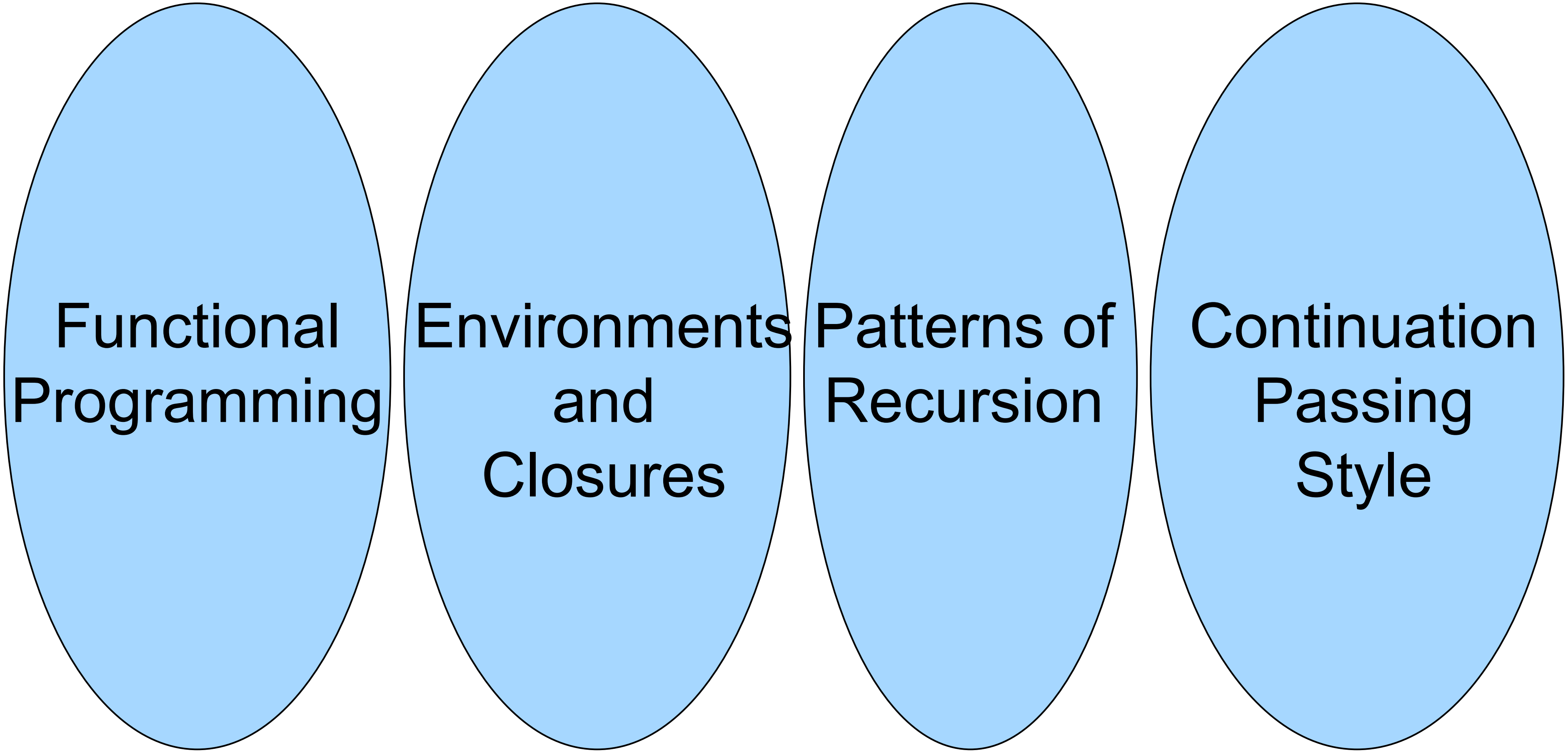
Language Semantics

Specification to Implementation

Three Main Topics

Programming Languages & Compilers

I : New Programming Paradigm



Functional Programming

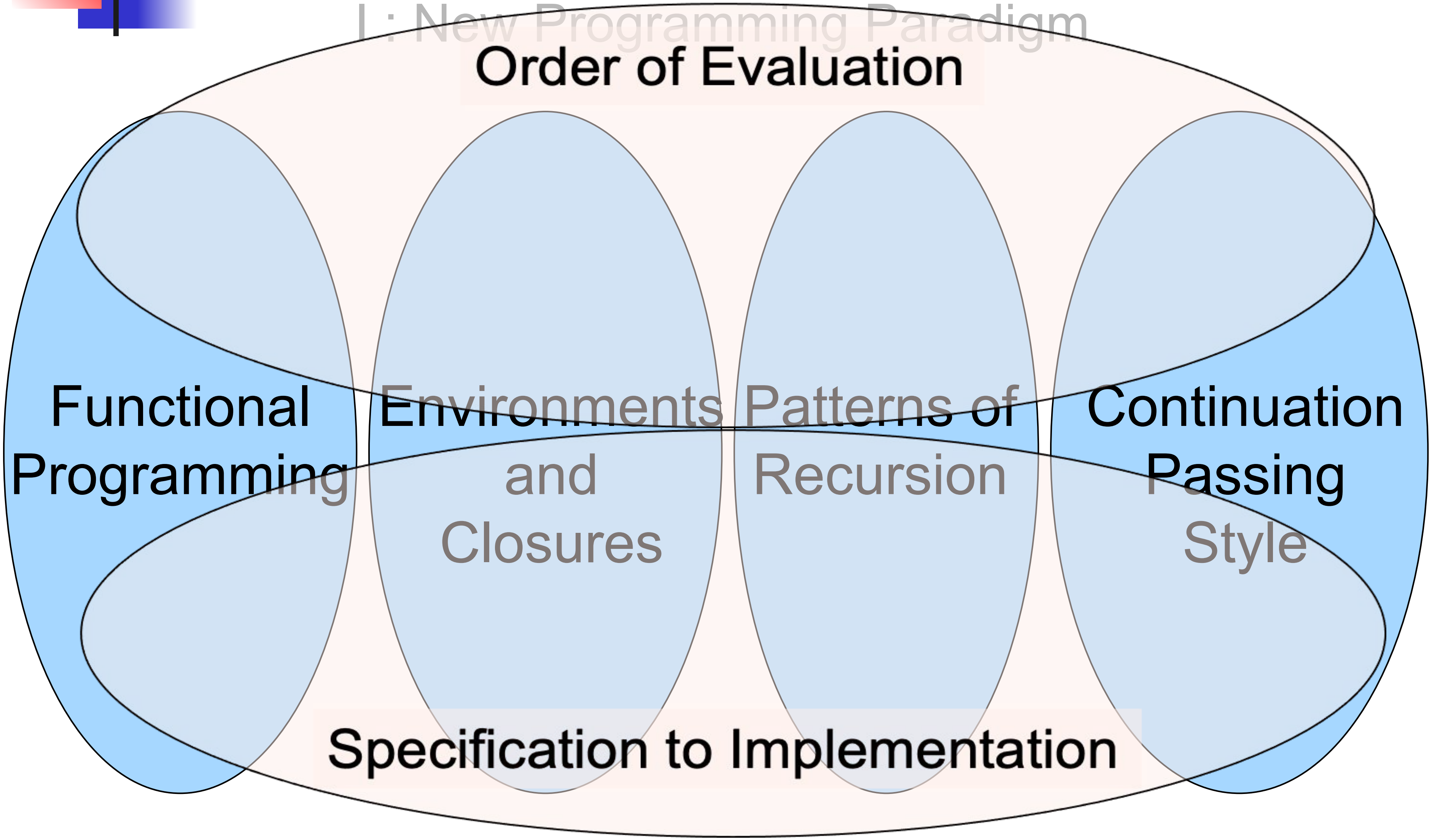
Environments and Closures

Patterns of Recursion

Continuation Passing Style

Three Main Topics

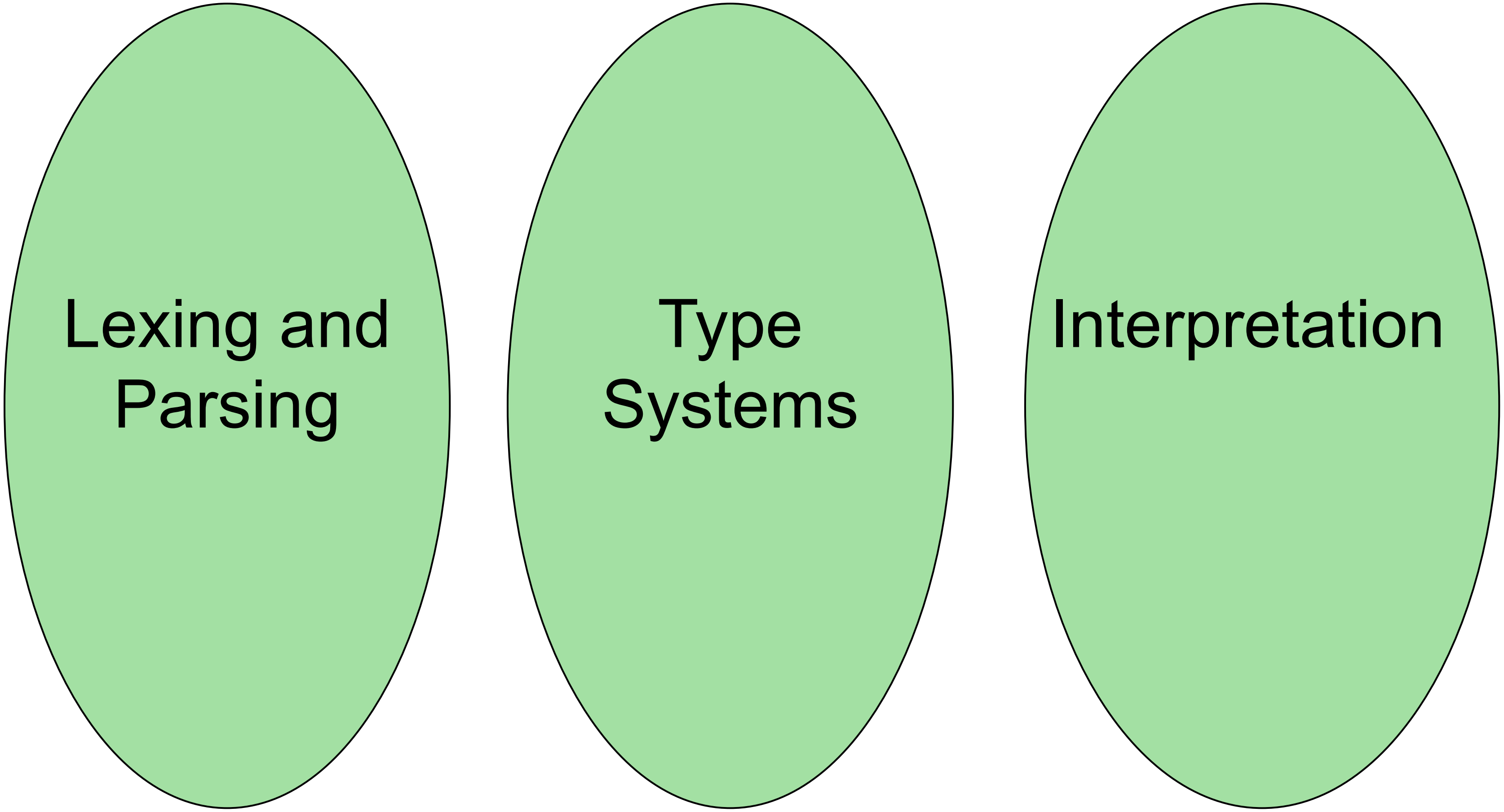
Programming Languages & Compilers


I : New Programming Paradigm
Order of Evaluation
Functional Programming
Environments and Closures
Patterns of Recursion
Continuation Passing Style
Specification to Implementation

Three Main Topics

Programming Languages & Compilers

II : Language Translation



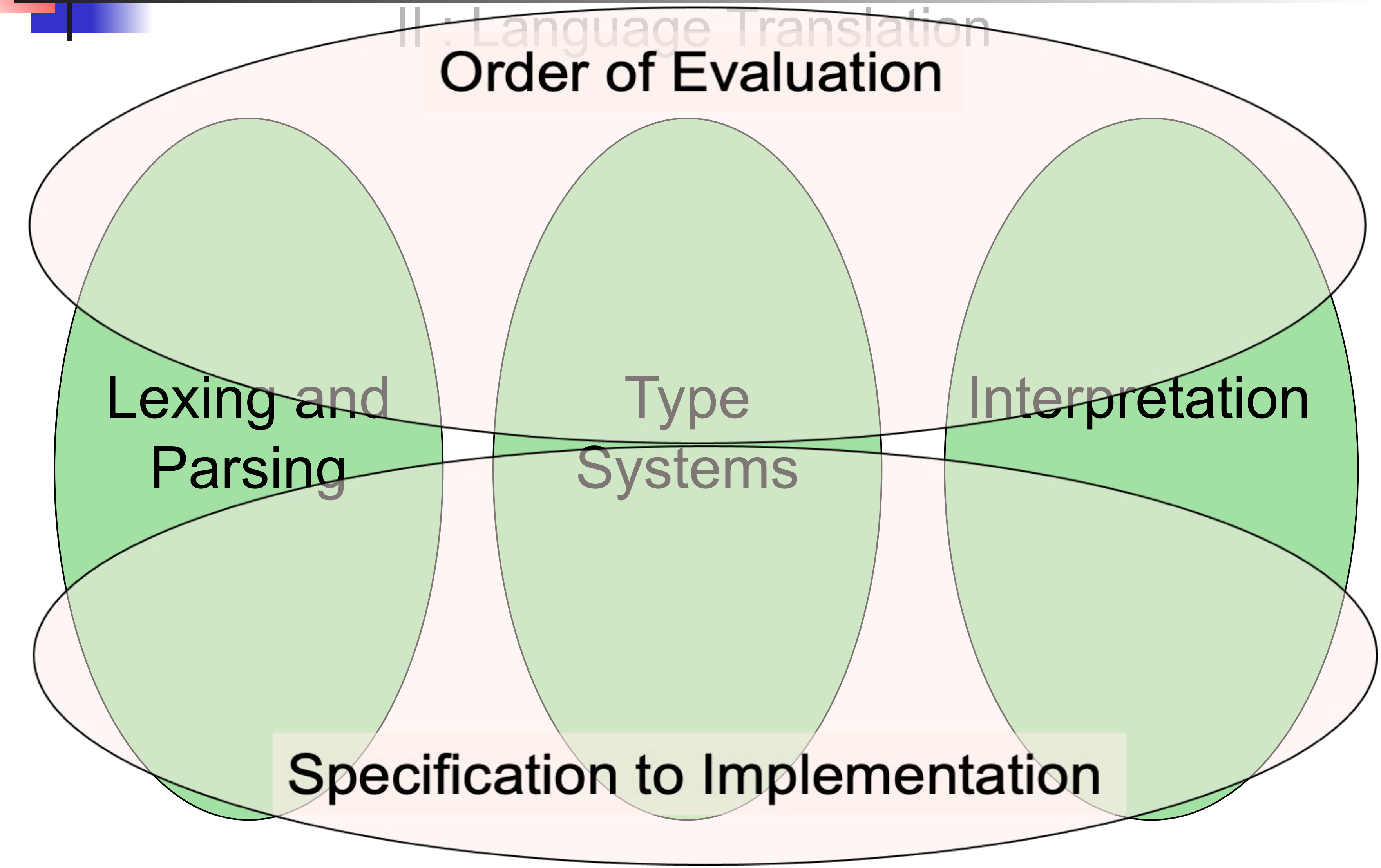
Lexing and Parsing

Type Systems

Interpretation

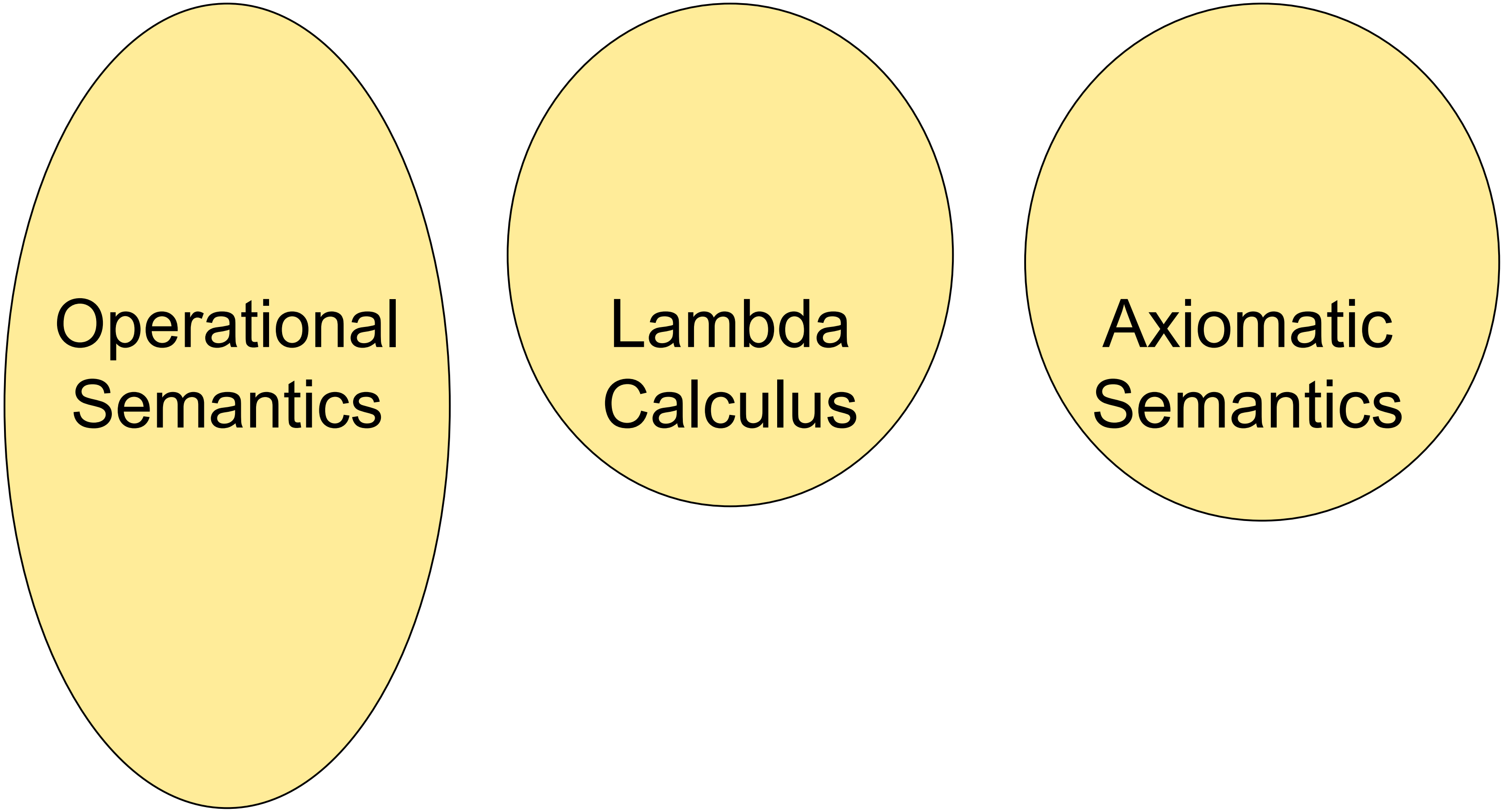
Three Main Topics

Programming Languages & Compilers



Three Main Topics

III : Language Semantics



Operational Semantics

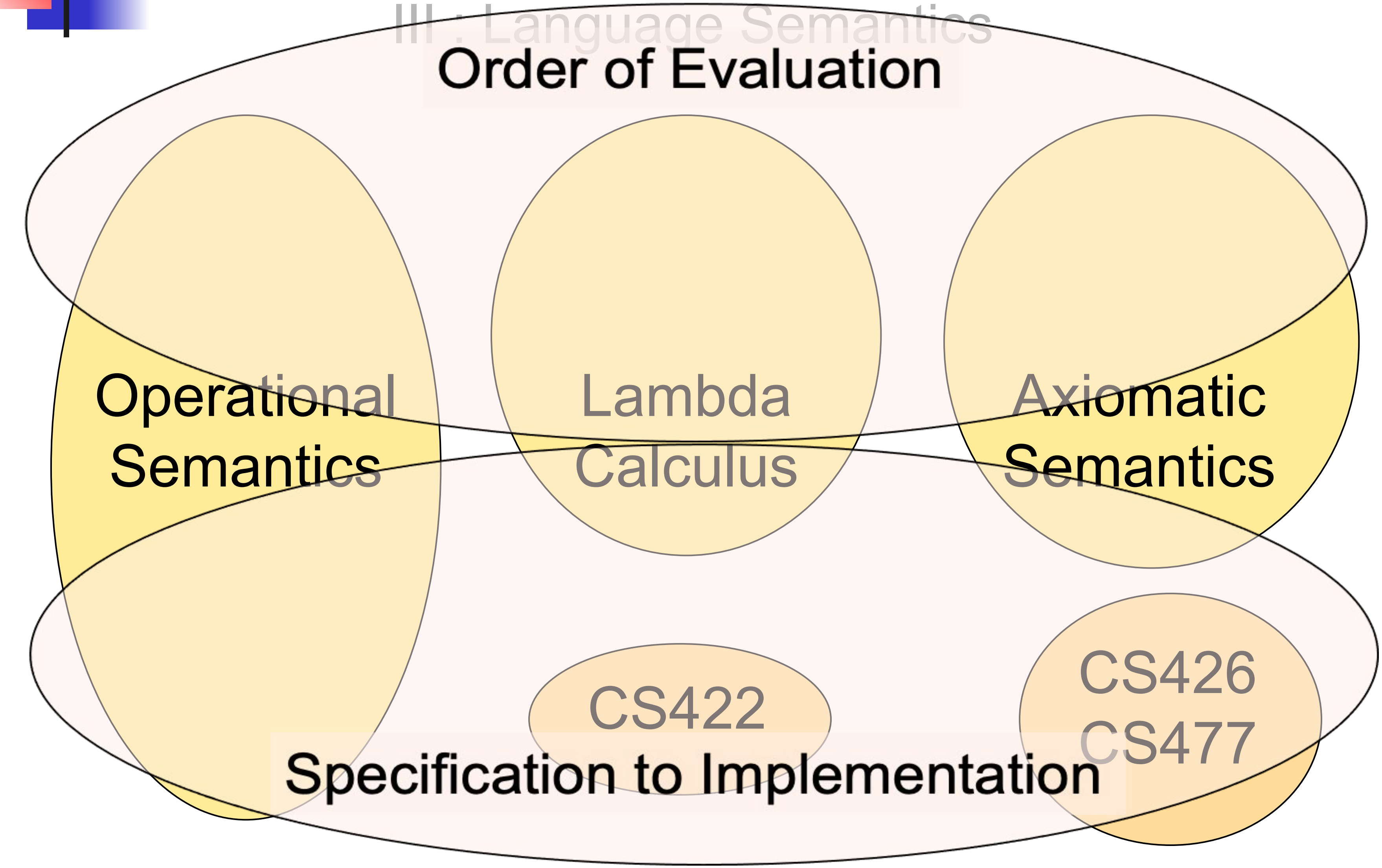
Lambda Calculus

Axiomatic Semantics

Three Main Topics

Programming Languages & Compilers

III : Language Semantics



Order of Evaluation

Operational Semantics

Lambda Calculus

Axiomatic Semantics

CS422

CS426
CS477

Specification to Implementation

Three Main Topics

Course Objectives

- New programming paradigm
  - Functional programming
  - Environments and Closures
  - Patterns of Recursion
  - Continuation Passing Style
- Language translation
  - Lexing and parsing
  - Type systems
  - Interpretation
- Language semantics
  - Lambda Calculus
  - Operational Semantics
  - Axiomatic Semantics

Three Main Topics

Course Logistics

Contact Information - Talia Ringer

- Office: 4218 SC
- Office hours:
  - Mondays 330 PM - 430 PM
  - Also by appointment (Calendly)
- Email: tringer@illinois.edu
- they/them

Relationship to CS421D Sections

- **Same**
  - Lecture schedule
  - Assignments
  - Shared pool of TAs
  - Most policies
- **Different**
  - Professor
  - Lecture style
  - Grading policy & extra credit

Relationship to CS421D Sections

- **Same**
  - Lecture schedule
  - Assignments
  - Shared pool of TAs
  - Most policies
- **Different**
  - Professor
  - **Lecture style**
  - Grading policy & extra credit

CS421D videos and slides will be online, too, if you'd like a different perspective on the same material.

Relationship to CS421D Sections

- **Same**
  - Lecture schedule
  - Assignments
  - **Shared pool of TAs**
  - Most policies
- **Different**
  - Professor
  - Lecture style
  - Grading policy & **extra credit**

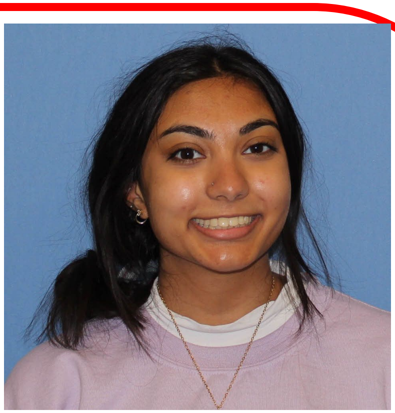
Please don’t ask other sections’ TAs for help with extra credit unique to our sections (will be explicit when relevant).

Course TAs - Our Sections

Paul Krogmeier

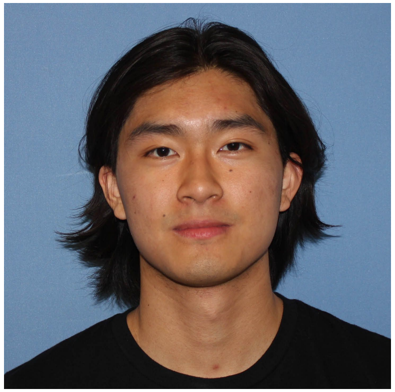
Course TAs - Other Sections

Shaurya Gomber



Deeya Bansal

Benjamin Darnell

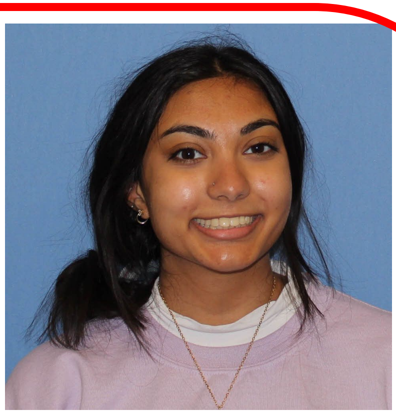


Alan Yao

Course TAs - All

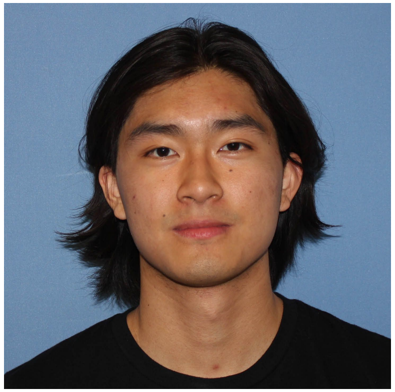
Paul Krogmeier

Shaurya Gomber



Deeya Bansal

Benjamin Darnell



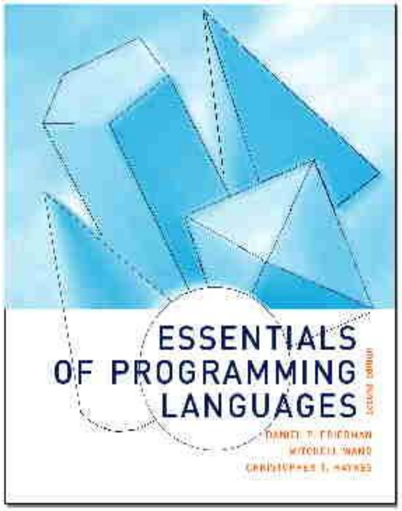
Alan Yao

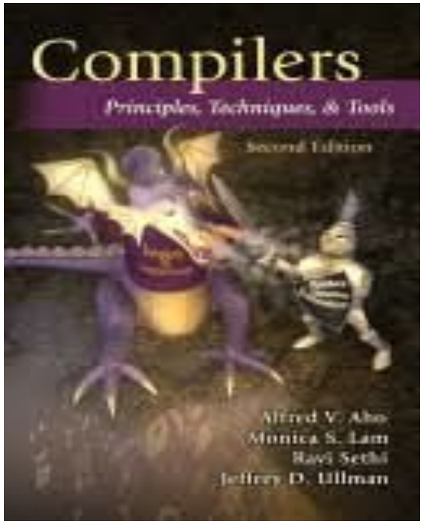
Course Website

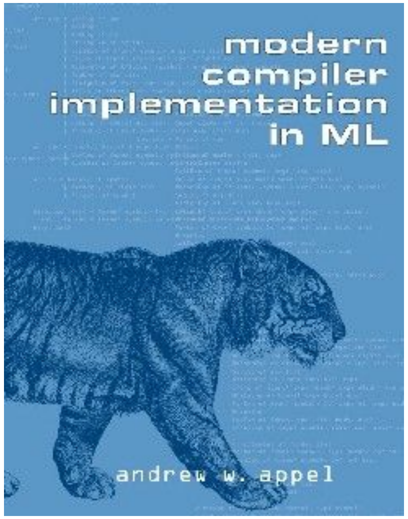
- https://courses.grainger.illinois.edu/cs421/fa2023/
  - I am Prof. Ringer; your section is CU/CG
- **Main page** - summary of news items
- **Class forum** - link to Piazza
- **Policy** - rules governing course
- **Lectures** - syllabus and slides
- **MPs** - information about assignments
- **Exams** – Syllabi and review material for exams
- **Unit Projects** - for 4 credit students
- **Resources** - tools and helpful info
- **FAQ**

Some Course References

- No required textbook
- Some suggested references







Some Course References

- No required textbook.
- Pictures of the books on previous slide
- **Essentials of Programming Languages** (2nd Edition) by Daniel P. Friedman, Mitchell Wand and Christopher T. Haynes, MIT Press 2001.
- **Compilers: Principles, Techniques, and Tools**, (also known as "The Dragon Book"); by Aho, Sethi, and Ullman. Published by Addison-Wesley. ISBN: 0-201-10088-6.
- **Modern Compiler Implementation in ML** by Andrew W. Appel, Cambridge University Press 1998
- Additional ones for Ocaml given separately

Course Grading

- **Assignments** 20%
  - Web Assignments (WA) (~10%)
  - MPs (in Ocaml) (~10%)
  - All WAs and MPs Submitted in **PrairieLearn**
  - Late submission:
    - 48 hours, unless otherwise specified
    - capped at 80% of total

Course Grading

- **Assignments 20%**
  - Web Assignments (WA) (~10%)
  - MPs (in Ocaml) (~10%)
  - All WAs and MPs Submitted in **PrairieLearn**
  - Late submission:
    - 48 hours, unless otherwise specified
    - capped at 80% of total

Weighed more heavily than in CS421D sections—please do these!

Course Grading

- **Four quizzes**, in class - 10%
- **3 Midterms**, CBTF - 15% each
  - Midterm 1: 9/14 - 9/16
  - Midterm 2: 10/12 - 10/14
  - Midterm 3: 11/9 - 11/11
  - Be around for these dates!
- **Final**: 25%
  - Tuesday, 12/12, 8:00 AM - 11:00 AM
- Percentages are approximate

Course Grading

- **Four quizzes**, in class - 10%
- **3 Midterms**, CBTF - 15% each
  - Midterm 1: 9/14 - 9/16
  - Midterm 2: 10/12 - 10/14
  - Midterm 3: 11/9 - 11/11
  - Be around for these dates!
- **Final**: **25%**
  - Tuesday, 12/12, 8:00 AM - 11:00 AM
- Percentages are approximate

Weighed less heavily than in CS421D, to make a bit less stressful, hopefully.

Course Grading

- **4 credit students have a course project**, **25%**
  - The rest of the grade will add to **75%**
  - See policy webpage for details

Course Grading

- **Creative opportunities**, **extra credit**, ~1% each
  - This section only
  - Hope to spark enthusiasm and reduce stress
  - More details coming soon

Course Grading

- **Creative opportunities**, **extra credit**, ~1% each
  - This section only
  - Hope to spark enthusiasm and reduce stress
  - More details coming soon

If you are in CS421D and hand these in, you will not get credit, even if you plead ignorance. This is an experiment I'm doing for this section!

Course Assignments – WA & MP

- **You may discuss** assignments & solutions with others.
- **You may work in groups**, but:
  - You must **list members with whom you worked**

    if you share solutions or detailed solution outlines.
  - Each student must write up and turn in **their own solution** separately. (**No direct copy-paste** – type i yourself from your understanding.)
- **Cite any sources appropriately**.
  - Note: University policy on plagiarism still holds — cite your sources if not the sole author of your solution.
  - Do not have to cite course notes or me.

Accommodations

- All professors and TAs **must comply** with DRES accommodations!
  - It is illegal not to do this in the US, and if any of your course staff for any course refuse, you can escalate to DRES and/or CS CARES
- The system is **not completely just**, so:
  - I can help you get started with DRES if you do not have official accommodations, but need them
  - Please tell me if there are ways I can make the class more accessible
  - You can always reach out if you are going through something or need help

Accommodations

- All professors and TAs **must comply** with DRES accommodations!
  - It is illegal not to do this in the US, and if any of your course staff for any course refuse, you can escalate to DRES and/or CS CARES
- The system is **not completely just**, so:
  - I can help you get started with DRES if you do not have official accommodations, but need them
  - Please tell me if there are ways I can make the class more accessible
  - You can always reach out if you are going through something or need help

Unofficial Communication

- I know students like to make **Discord servers** for these courses. That's fine, but:
  - Bullying on Discord is real and unacceptable
  - If you use any server name or description that is in any way affiliated with the university, department, or course, even if it unofficial, you are responsible for upholding the **CS Code of Conduct**
  - The same holds for other unofficial class forums if they are branded as such
  - Please talk to CS CARES or to me (I am part of CS CARES, anyways) if this is not being upheld

Questions so far?

OCaml

OCaml

- Locally:
  - Will use OCaml inside VSCode inside PrairieLearn problems this semester
- Globally:
  - Main OCaml home: http://ocaml.org
  - To install OCaml on your computer see: http://ocaml.org/docs/install.html
  - To try on the web: https://try.ocamlpro.com
  - More notes on this later

References for OCaml

- Supplemental texts (not required):
  - The Objective Caml system release 4.05, by Xavier Leroy, online manual
  - Introduction to the Objective Caml Programming Language, by Jason Hickey
  - Developing Applications With Objective Caml, by Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano, on O’Reilly
    - Available online from course resources

Features of OCaml

- Higher order applicative language
- Call-by-value parameter passing
- Modern syntax
- Parametric polymorphism
  - Also known as structural polymorphism
- Automatic garbage collection
- User-defined algebraic data types

Ways of Writing OCaml

- In your favorite editor (good for large projects)
- In an interactive session (good for class)

Ways of Writing OCaml

- In your favorite editor (good for large projects)
- **In an interactive session (good for class)**

Session in OCaml

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop *)
  2 + 3;;      (* Expression *)
-  : int = 5
# 3 < 2;;      (* Expression *)
- : bool = false
```

Session in OCaml

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop *)
  2 + 3;;     (* Expression *)
-  : int = 5
# 3 < 2;;     (* Expression *)
- : bool = false
```

Session in OCaml

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop *)
  2 + 3;;     (* Expression *)
-  : int = 5
# 3 < 2;;     (* Expression *)
- : bool = false
```

Session in OCaml

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop *)
   2 + 3;;      (* Expression *)
-   : int = 5
# 3 < 2;;      (* Expression *)
- : bool = false
```

Expressions in OCaml

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop *)
   2 + 3;;     (* Expression *)
-  : int = 5
# 3 < 2;;     (* Expression *)
- : bool = false
```

Expressions in OCaml

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop *)
  2 + 3;;      (* Expression *)
-   : int = 5
# 3 < 2;;      (* Expression *)
- : bool = false
```

Expressions in OCaml

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop *)
  2 + 3;;      (* Expression *)
-  : int = 5
# 3 < 2;;      (* Expression *)
- : bool = false
```

Expressions in OCaml

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop *)
   2 + 3;;      (* Expression *)
-   : int = 5
# 3 < 2;;      (* Expression *)
- : bool = false
```

Sequencing Expressions

```
# print_string "Bye\n";;
Bye
- : unit = ()
# 25;;
- : int = 25
# (print_string "Bye\n"; 25);;
Bye
- : int = 25
```

Declarations

```
# let x = 2 + 3;;     (* declaration *)
val x : int = 5
# let test = 3 < 2;; (* declaration *)
val test : bool = false
```

Declarations

```
# let x = 2 + 3;;      (* declaration *)
val x : int = 5
# let test = 3 < 2;; (* declaration *)
val test : bool = false
```

Sequencing of Declarations

```
# let a = 1
  let b = a + 4;;
val a : int = 1
val b : int = 5
```

Booleans (aka Truth Values)

```
# true;;
- : bool = true
# false;;
- : bool = false
```

Boolean Combinators

```
# 3 > 1 && 4 > 6;;
- : bool = false
# 3 > 1 || 4 > 6;;
- : bool = true
# not (4 > 6);;
- : bool = true
# if b > a then 25 else 0;;
- : int = 25
```

Booleans and Short-Circuit Evaluation

```
# (print_string "Hi\n"; 3 > 1) || 4 > 6;;
Hi
- : bool = true
# 3 > 1 || (print_string "Bye\n"; 4 > 6);;
- : bool = true
```

Notes About Floats: No Overloaded Operators

```
# 1 + 0;;
- : int = 1
# 1.35 + 0.23;;  (* Wrong type of addition *)
Characters 0-4:
  1.35 + 0.23;;  (* Wrong type of addition *)
  ^^^^
Error: This expression has type float but an
    expression was expected of type
      int
# 1.35 +. 0.23;;
- : float = 1.58
```

Notes About Floats: No Overloaded Operators

```
# 1 + 0;;
- : int = 1
# 1.35 + 0.23;;  (* Wrong type of addition *)
Characters 0-4:
  1.35 + 0.23;;  (* Wrong type of addition *)
  ^^^^
Error: This expression has type float but an
    expression was expected of type
       int
# 1.35 +. 0.23;;
- : float = 1.58
```

Notes About Floats: No Overloaded Operators

```
# 1 + 0;;
- : int = 1
# 1.35 + 0.23;;  (* Wrong type of addition *)
Characters 0-4:
  1.35 + 0.23;;  (* Wrong type of addition *)
  ^^^^
Error: This expression has type float but an
     expression was expected of type
         int
# 1.35 +. 0.23;;
- : float = 1.58
```

Notes About Floats: No Overloaded Operators

```
# 1 + 0;;
- : int = 1
# 1.35 + 0.23;;  (* Wrong type of addition *)
Characters 0-4:
  1.35 + 0.23;;  (* Wrong type of addition *)
  ^^^^
Error: This expression has type float but an
    expression was expected of type
        int
# 1.35 +. 0.23;;
- : float = 1.58
```

Notes About Floats: No Implicit Coercion

```
# 1.0 * 2;; (* No Implicit Coercion *)
Characters 0-3:
  1.0 * 2;; (* No Implicit Coercion *)
  ^^^
Error: This expression has type float but an
     expression was expected of type
         int
```

Functions

```
# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
```

Functions

```
# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
```

Functions

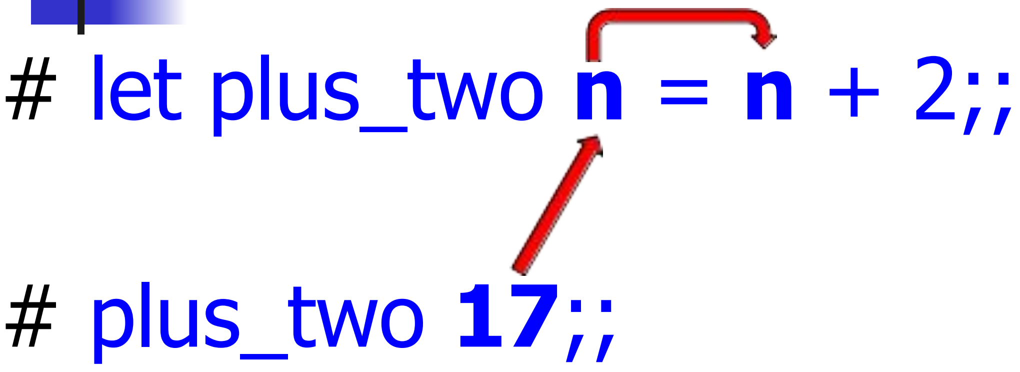
```
# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
```

Functions

```
# let plus_two n = n + 2;;


# plus_two 17;;
- : int = 19
```

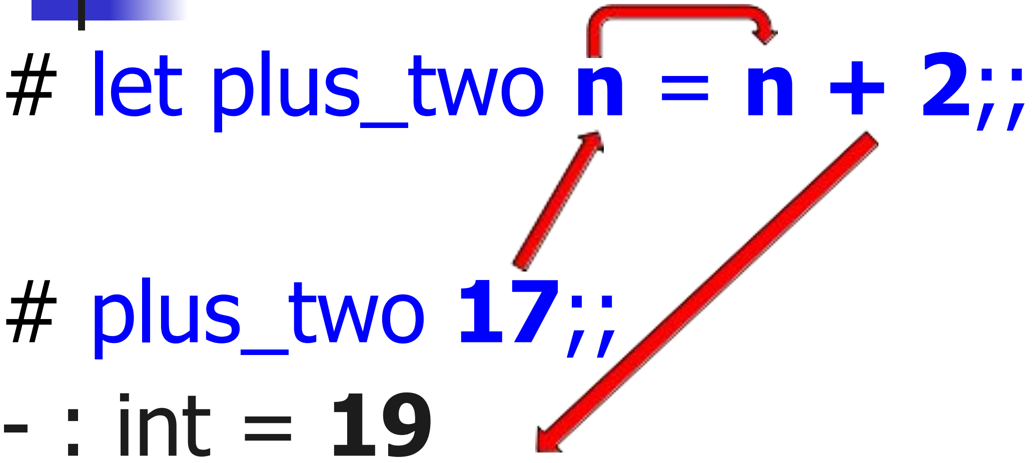
Functions



```
# let plus_two n = n + 2;;


# plus_two 17;;
- : int = 19
```

Functions



```
# let plus_two n = n + 2;;


# plus_two 17;;
- : int = 19
```

Functions

```
# let plus_two n = n + 2;;

# plus_two 17;;
```

Anonymous Functions are **Fun**

```
# fun n -> n + 2;;

# (fun n -> n + 2) 17;;
```

Anonymous Functions are **Fun**

```
# let plus_two = fun n -> n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
```

Functions

```
# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
```

Functions with More Arguments

```
# let add_three x y z = x + y + z;;
val add_three : int -> int -> int -> int = <fun>
# let t = add_three 6 3 2;;
val t : int = 11
# let f = add_three 6;;
val f : int -> int -> int = <fun>
# let add_three =
   fun x -> (fun y -> (fun z -> x + y + z));;
val add_three : int -> int -> int -> int = <fun>
```

Functions with More Arguments

```
# let add_three x y z = x + y + z;;
val add_three : int -> int -> int -> int = <fun>
# let t = add_three 6 3 2;;
val t : int = 11
# let f = add_three 6;;
val f : int -> int -> int = <fun>
# let add_three =
   fun x -> (fun y -> (fun z -> x + y + z));;
val add_three : int -> int -> int -> int = <fun>
```

OCaml

Functions with More Arguments

```
# let add_three x y z = x + y + z;;
val add_three : int -> int -> int -> int = <fun>
# let t = add_three 6 3 2;;
val t : int = 11
# let f = add_three 6;;
val f : int -> int -> int = <fun>
# let add_three =
   fun x -> (fun y -> (fun z -> x + y + z));;
val add_three : int -> int -> int -> int = <fun>
```

Functions with More Arguments

```
# let add_three x y z = x + y + z;;
val add_three : int -> int -> int -> int = <fun>
# let t = add_three 6 3 2;;
val t : int = 11
# let f = add_three 6;;
val f : int -> int -> int = <fun>
# let add_three =
   fun x -> (fun y -> (fun z -> x + y + z));;
val add_three : int -> int -> int -> int = <fun>
```

Ways of Writing OCaml

- **In your favorite editor (good for large projects)**
- In an interactive session (good for class)

Ways of Writing OCaml

- **In VSCode (MP1 on PrairieLearn)**
- In an interactive session (good for class)

REPL versus Files

```
# let x = 2 + 3;;
val x : int = 5

# let a = 1
  let b = a + 4;;
val a : int = 1
val b : int = 5
```

REPL versus **Files**

```
let x = 2 + 3
```

```
# let a = 1
  let b = a + 4;;
val a : int = 1
val b : int = 5
```

REPL versus **Files**

```
let (x : int) = 2 + 3
```

```
# let a = 1
  let b = a + 4;;
val a : int = 1
val b : int = 5
```

REPL versus Files

```
# let a = 1
  let b = a + 4;;
val a : int = 1
val b : int = 5
```

REPL versus **Files**

```
let a = 1
let b = a + 4
```

REPL versus **Files**

```
(* different meaning, but more common *)
   let a = 1 in
   let b = a + 4
```

Questions so far?

Bonus

OCaml in the Wild

More next class!

More Next Class

Start MP1!