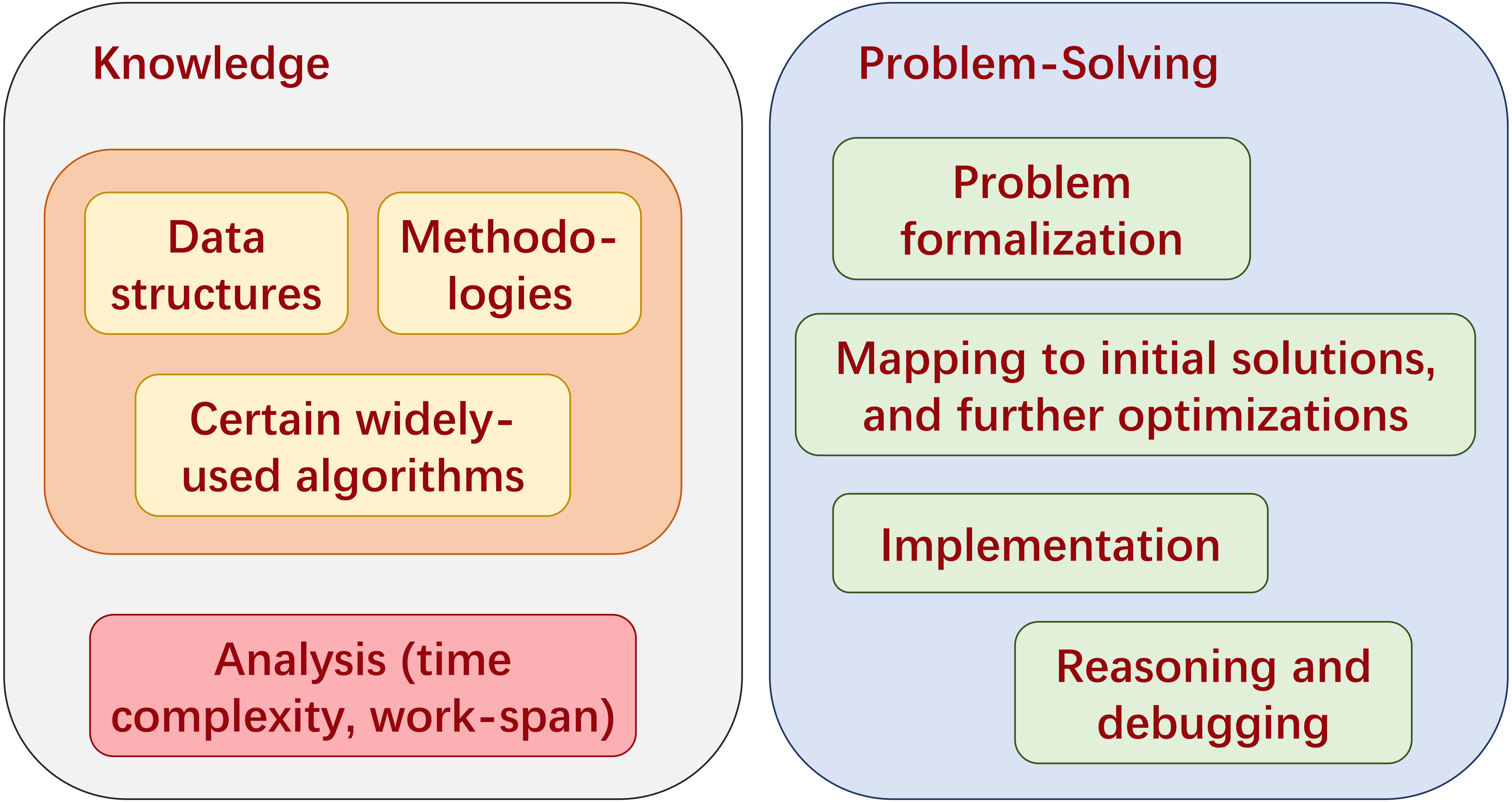
Road Ahead and Course Summary

Yan Gu

This lecture does not cover anything in CLRS

We are close to finish!

- We have done a lot of programming, some are about making your algorithm run fast, others are about implementing classic algorithms or classic problems
- Most of you have done well: about 60% have done almost all the work and head toward an A or A+
  - You deserve the grade, but hopefully the experience is more valuable


Knowledge
Data structures
Methodo-logies
Certain widely-used algorithms
Analysis (time complexity, work-span)
Problem-Solving
Problem formalization
Mapping to initial solutions, and further optimizations
Implementation
Reasoning and debugging

What is a typical interview process? (Google's software developer as an example, link)

- **Round 1: online assessment (90 minutes)**
  - Two data structures and algorithms questions that you have to complete in less than 90 minutes in total
  - You'll need to write your own test cases as you won't be provided with any
- **Round 2: Technical phone interview (1 or 2)**
  - You will solve data structure and algorithm questions
  - You'll share a Google Doc with your interviewer, write your solution directly in the document and won't have access to syntax highlighting or auto-completion like you would in a regular IDE
  - Finally, in addition to coding questions, you should also be ready to answer a few typical behavioral questions

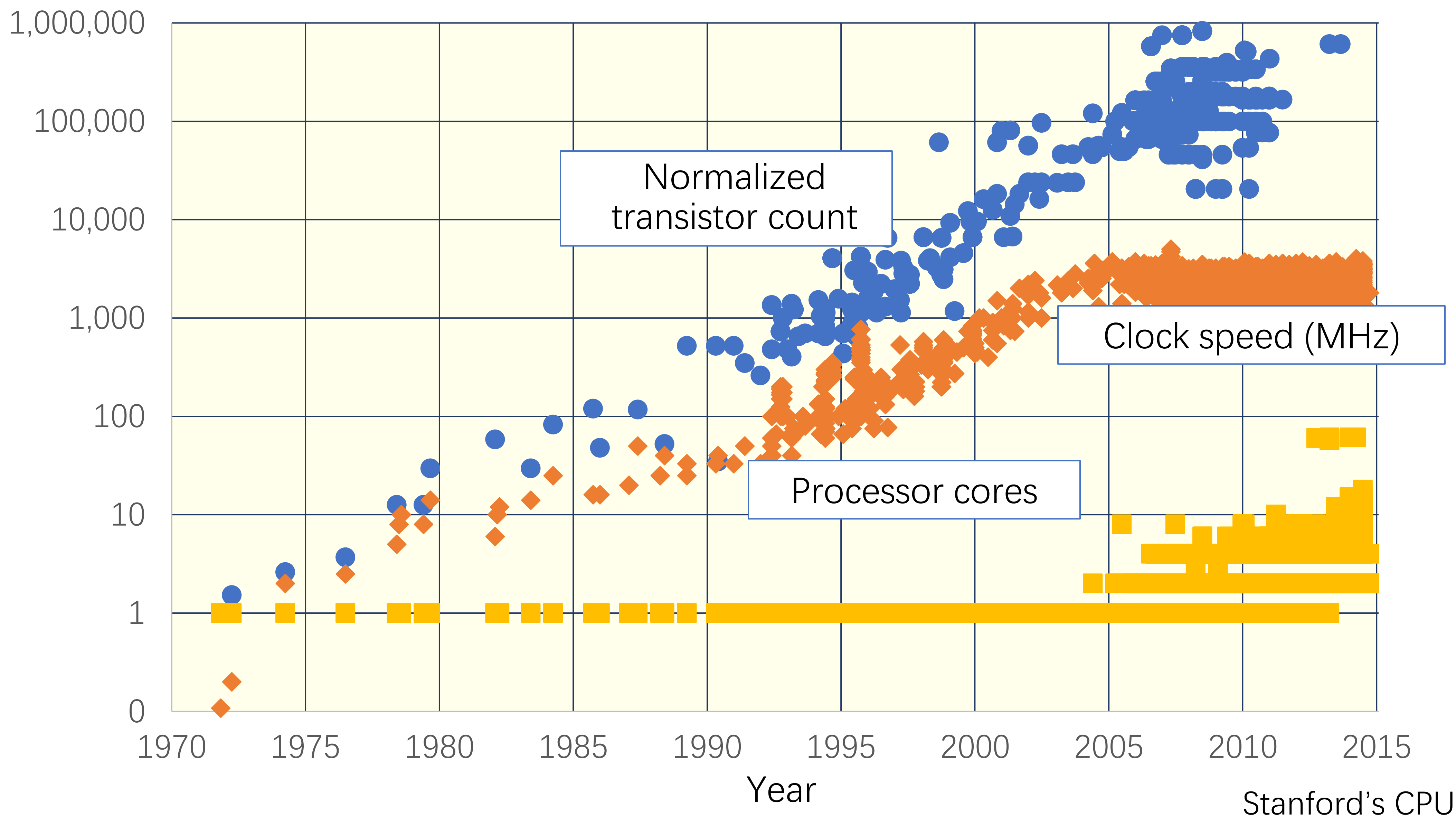
What is a typical interview process? (Google's software developer as an example)

- **Round 3: Onsite interviews**
  - Onsite interviews are the real test. You'll typically spend a full day at a Google office and do usually four interviews in total
  - You'll typically get three coding interviews with data structure and algorithm questions, and one system design interviews
  - **All candidates are expected to do extremely well in coding interviews**. If you're relatively junior (L4 or below) then the bar will be lower in your system design interviews than for mid-level or senior engineers (e.g. L5 or above)
  - You'll use a whiteboard to write your code in most onsite interviews at Google

Check again after 142:

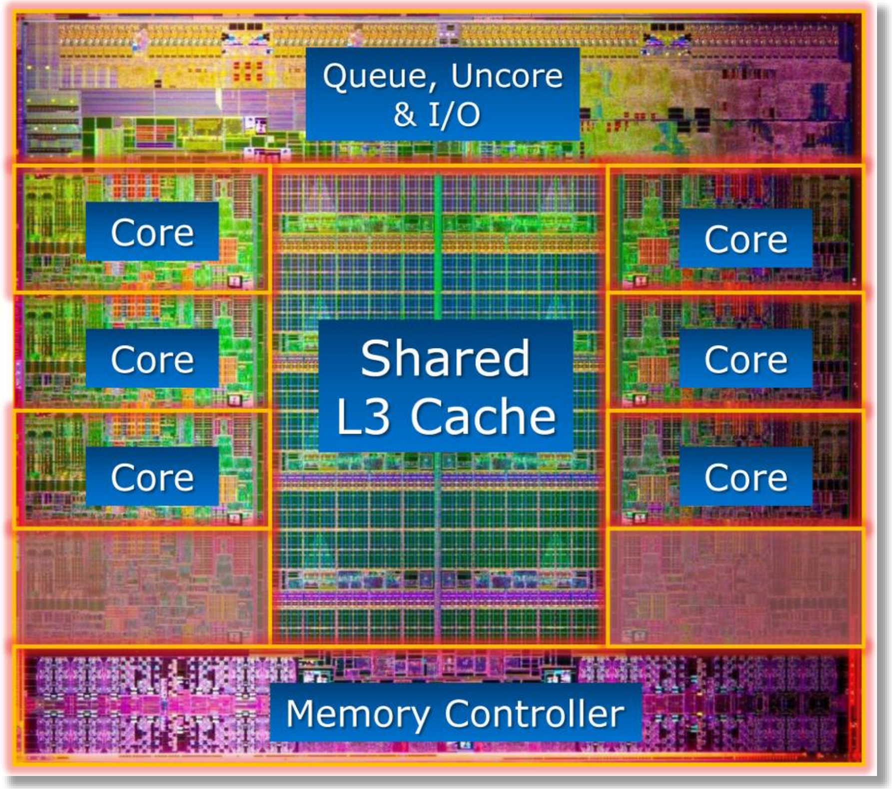
- **Can you implement the classic algorithms mentioned in 14/141?**
  - Dijkstra's/Prim's algorithm, Kruskal's algorithm, Huffman code, 0/1 knapsack, activity selection, sorting, various data structures
  - If so, put them in a code-book. Remind yourself once a while, and review them before an interview
- **Do you have a better sense of algorithmic ideas?**
  - DP in classic forms (knapsack, LCS, LIS) and variations
  - Greedy algorithms
  - Representing geometric data (points, directions, polar angles)
- **Can you implement an algorithm faster and more accurate within a time limit?**

Technology Scaling



Stanford's CPU DB [DKM12]

Vendor Solution: Multicore



Intel Core i7 3960X (Sandy Bridge E), 2011

- 6 cores / 3.3 GHz / 15-MB L3 cache

- To scale performance, processor manufacturers put many processing cores on the microprocessor chip
- Each generation of Moore's Law potentially doubles the number of cores

Check again after 142:

- **What is the motivation for parallelism?**
- **How should we consider and design a parallel algorithm?**
  - Binary fork-join model, scheduler, work and span analysis
  - Examples: reduce, scan, filter/packing, quicksort
  - Interesting ideas: computational model, race, functional algorithms/programming
- **What is our current architecture, and how to write efficient code on top of that?**
  - Computer architecture: executing instructions, and accessing memory
  - Things we don’t need to worry: ILP, hyperthreading, vectorization (mostly)
  - Things we do need to consider: multicore (parallelism), I/O-efficiency

Recall that we have candies for CS 142

- Solved hard training problems, wrote the fastest code for a problem
- Solved bonus questions for homework problems
- Attend the class and discussions, participate in discussions
- Participate in training problem analysis

CS142-Algorithm Engineering

COUPON

The candies will directly be added to your final grade

- If I can see you again on the fall quarter, you are welcome to my office and I will give you the gift

- My office: WCH 335

The candies will directly be added to your final grade

- 15 points for training
- 30 points for homework
- 5 points for midterm quiz
- Plus bonus points

- 5 points for training 4
- 25 points for final exam
- 20 points for final project
- Up to 5 points for training problem analysis

Tentative score-to-grade mapping

- A+: very top performance in the class
- A: 85%
- A-: 80%
- B+: 75%
- B: 70%
- C: 65%
- D: 60%

How about if I'm not satisfied with my current performance

- **This is an elective course, so I don't want to fail any of you**
- **This is a special time, and there are a lot of uncertainties and unexpected problems we need to handle**

- **However, this is a programming course, with 70% weight on homework and 30% on exams**
  - You need at least half of the scores for homework to pass
  - You can still work on the training and homework problems, and send it to me via email

- **Alternatively, I'm fine if you drop the course (the deadline is in two days)**

Training problem analysis

- **Prepare a 10-minute talk**
  - What the problem is
  - What your solution is and why it is correct
  - How to program and what optimization you use
  - If there are other interesting solutions
  - (You need to have solved that problem already)
- **You will get 3-5 bonus candies**
  - 3 candies by default
  - Up to 2 bonus candies for good talks

Training 1		
B	Maximum Identity Matrix	Lucas Song
C	Go Straight	
Training 2		
A	Ski	
B	Counting Candies	Osvaldo Moreno
C	Sort the train	Albert Dang
Training 3		
B	Symmetry Makes Perfect	
C	Select courses	Yuta Nakamura
D	More Office Hours	Alex Chen
Training 4		
A	Selling candies	
B	Easy sorting	
C	Share candies	

Check again after 142:

- **Can you implement the classic algorithms mentioned in 14/141?**
  - Dijkstra's/Prim's algorithm, Kruskal's algorithm, Huffman code, 0/1 knapsack, activity selection, sorting, various data structure
  - If so, put them in a code-book. Remind yourself once a while, and review them before an interview
- **Do you have a better sense of algorithmic ideas?**
  - DP in classic forms (knapsack, LCS, LIS) and variations
  - Greedy algorithms
  - Representing geometric data (points, directions, polar angles)
- **Can you implement an algorithm faster and more accurate within a time limit?**

Check again after 142:

- **Can you implement the classic algorithms mentioned in 14/141?**
  - Do you understand and be able to implement the classic algorithms in CLRS?

- **Do you have a better sense of algorithmic ideas?**
  - And use them to solve challenging problems?

- **Can you implement an algorithm faster and more accurate within a time limit?**
  - And are you able to program a sophisticated algorithm with hundreds of lines?

Check again after 142:

- **What is the motivation for parallelism?**
- **How should we consider and design a parallel algorithm?**
  - Binary fork-join model, scheduler, work and span analysis
  - Examples: reduce, scan, filter/packing, quicksort
  - Interesting ideas: computational model, functional algorithms/programming
- **What is our current architecture, and how to write efficient code on top of that?**
  - Computer architecture: executing instructions, and memory access
  - Things we don't need to worry: ILP, hyperthreading, vectorization (mostly)
  - Things we do need to consider: multicore (parallelism), I/O-efficiency

Check again after 142:

- What is the motivation for parallelism?
- How should we consider and design a parallel algorithm?
- What are the classic parallel algorithms?

- What is our current architecture, and how to write efficient code on top of that?
- How to actually engineer an efficient algorithms for modern architecture?

COMBINED BS + MS PROGRAMS

- https://student.engr.ucr.edu/BS-MS-requirements
- **More details in PDF: link, one-page overview: link**
- **Many benefits:**
  - Save time & money, automatic admission, chance to excel, preparing you better for industry or pursue a PhD
- **Disadvantage: cost you a year of time, and some tuitions**

COMBINED BS + MS PROGRAMS

- https://student.engr.ucr.edu/BS-MS-requirements

- Application requirement:

- Cumulative GPA above 3.4
- Cumulative GPA above 3.2 in all math, science, & engineering courses
- Completion of core courses within two terms of expected graduation
- Minimum grade required in each core course is at least a B-
- Minimum combined GPA for core courses is above 3.2

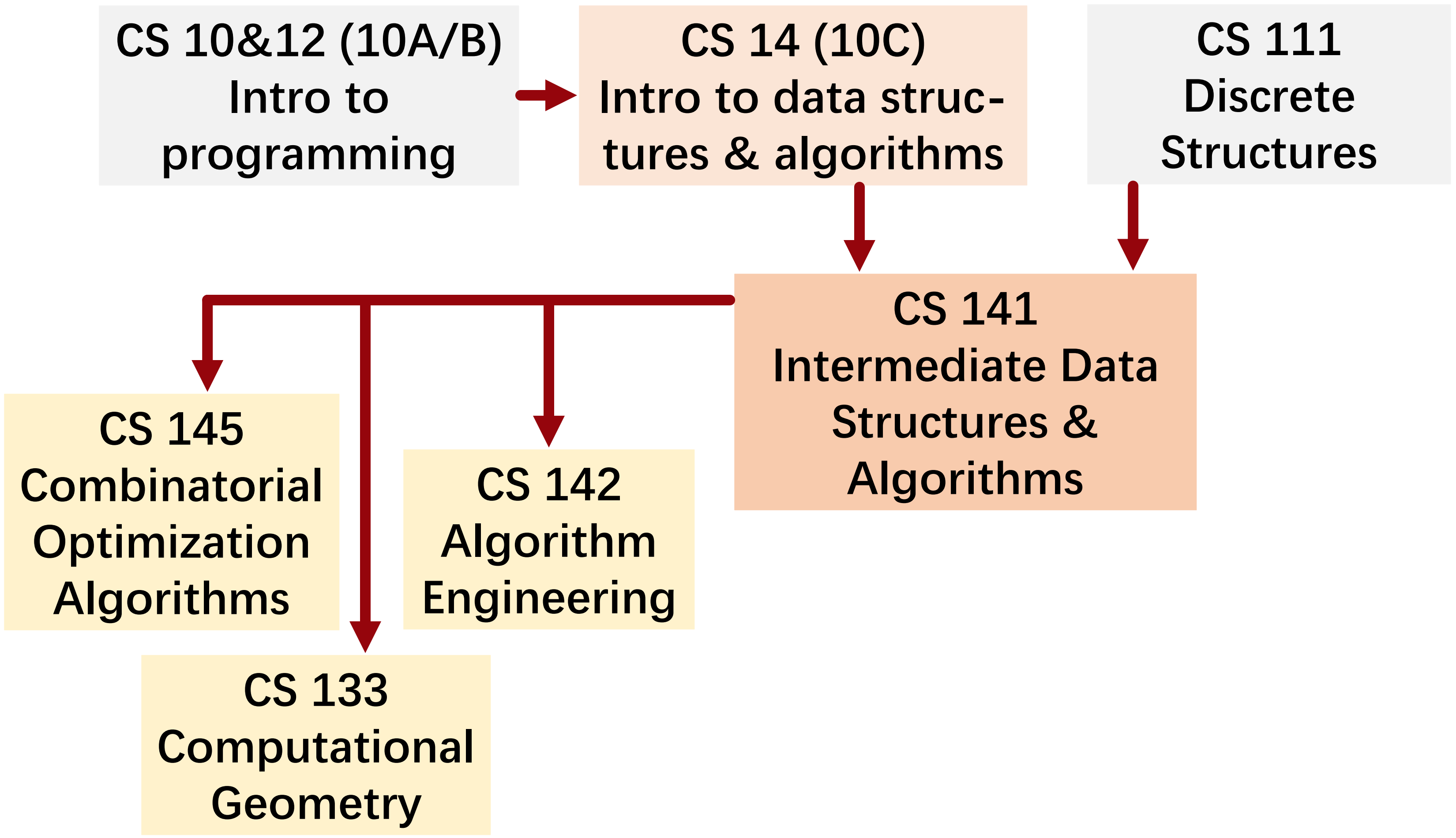
Other options

- Apply for CS master program at UCR
- Just take the courses
- Apply for the CS PhD program at UCR

Research opportunities

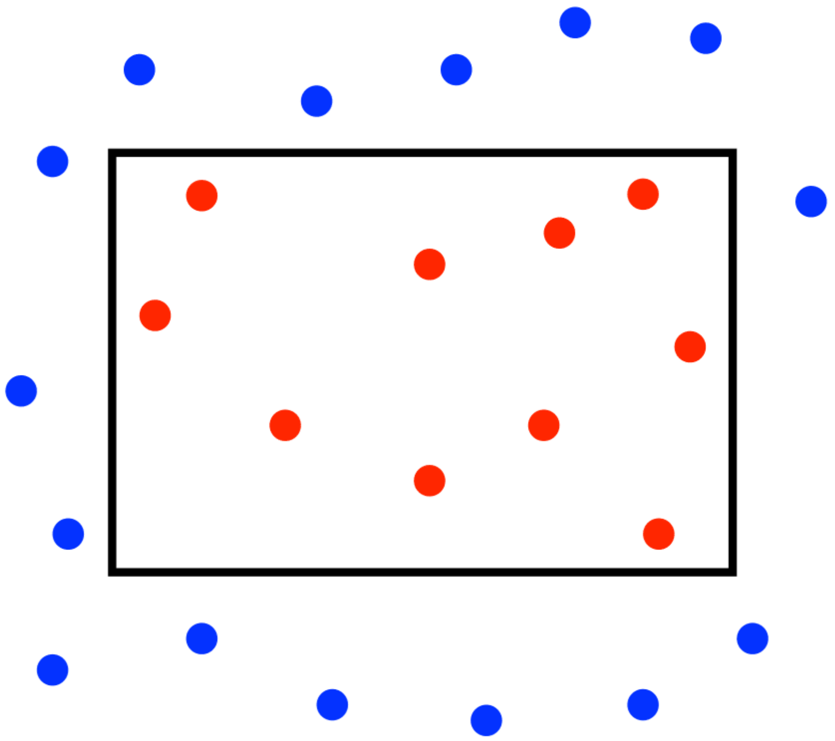
- Amey Bhangale (complexity), Marek Chrobak (approximate algorithms), Silas Richelson (cryptography)
- Yan Gu, Yihan Sun (efficient (parallel) algorithms)
- Vagelis Hristidis, Vassilis Tsotras (databases)
- Evangelos Papalexakis (data mining algorithms)
- Ahmed Eldawy (distributed algorithms), Amr Magdy (GIS algorithms)
- Rajiv Gupta, Zhijia Zhao (parallel software)
- Craig Schroeder, Tamar Shinar (graphics and scientific computing)

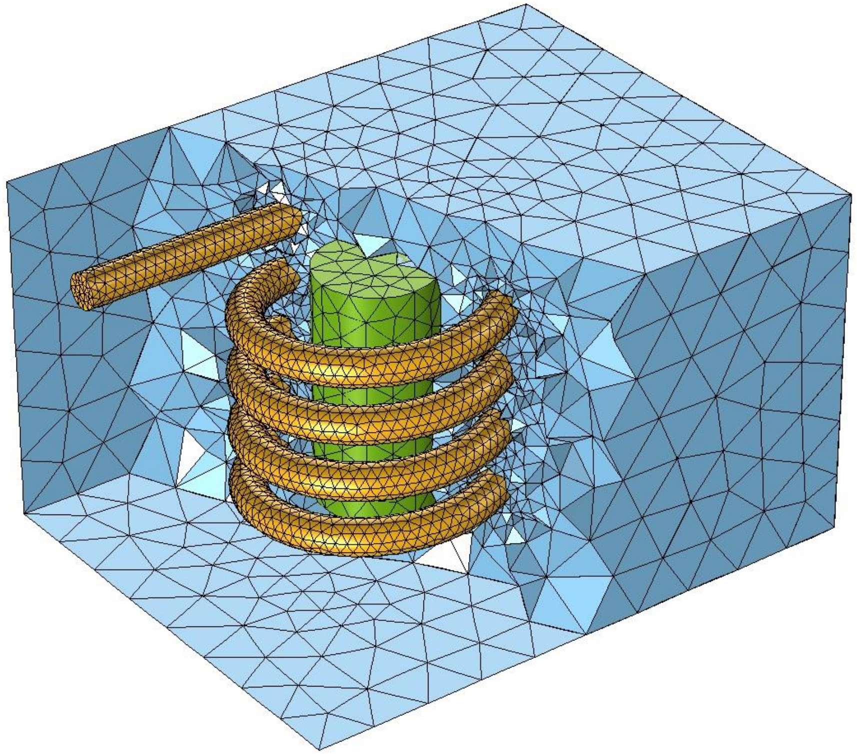
- Usually during summertime, but can continue during the quarters
- Can discuss more privately


CS 10&12 (10A/B)
Intro to programming
CS 14 (10C)
Intro to data struc-tures & algorithms
CS 111
Discrete Structures
CS 141
Intermediate Data Structures & Algorithms
CS 145
Combinatorial Optimization Algorithms
CS 142
Algorithm Engineering
CS 133
Computational Geometry

CS 133 Computational Geometry

- Last offering: Spring 2020 (no offering in AY 2020-21)
- By: Ahmed Eldawy





Why studying geometry?

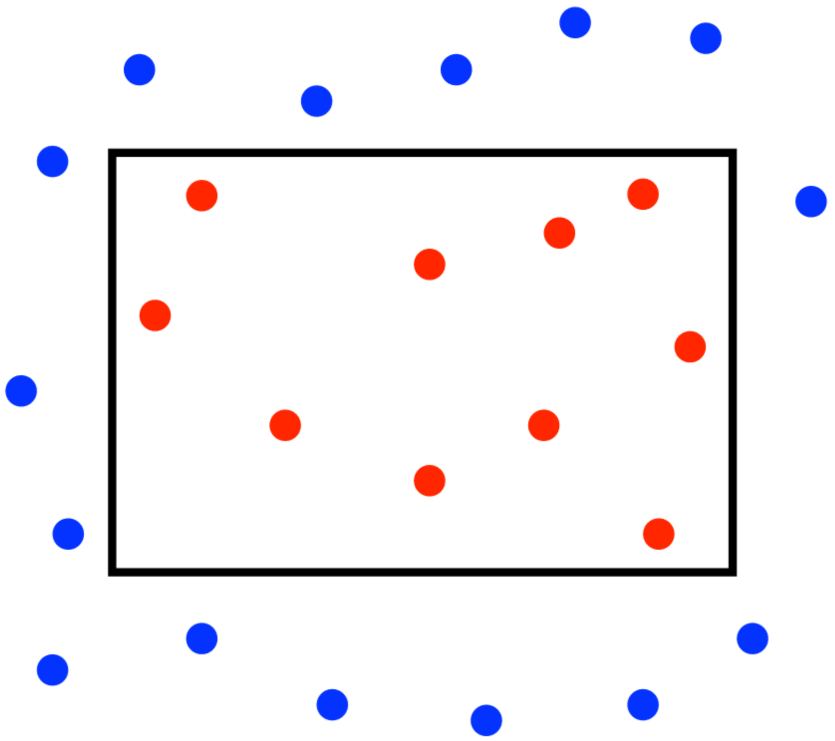
- **Consider the data that a computer works on**
- **Except for very special cases (like a sequence), they usually fall into two categories:**
- **Objects and their relations**
  - Usually abstracted as graphs, and we have learned many graph algorithms
- **Objects with their positions**
  - Abstracted as geometric problems
  - Used further in areas such as databases, computer graphics, data mining, machine learning

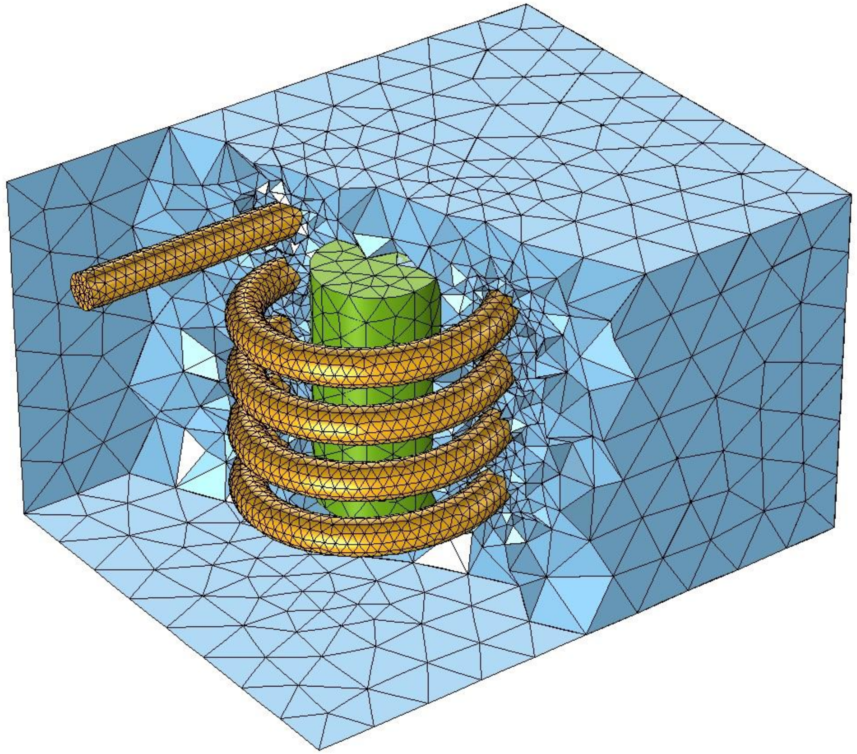
What are covered in CS 133 Computational Geometry?

- Based on the last offering by Ahmed Eldawy in S20
- Search problems
- Spatial indexing
- Intersection problems (Intersection of lines, rectangles, and polygons)
- Convex Hull
- Dealunay Triangulation

CS 133 Computational Geometry

- Last offering: Spring 2020 (no offering AY 2020-21)
- By: Ahmed Eldawy





CS 145 Combinatorial Optimization Algorithms

- Last offering: Fall 2020
- By: Silas Richelson

What are covered in CS 145 Combinatorial Optimization?

- Based on the last offering by Silas Richelson in F20

- Sorting (1 lecture)
- Stable Marriage (1 lecture)
- Network flow (1 lecture)
- Linear programming (6 lectures)
- Some random topics about complexity and randomized/approximate algorithms (~6 lectures)

Linear Programming

$$\text{Min} \quad \sum_{j=1}^{n} c_j x_j \qquad \text{(objective function)}$$

subject to:

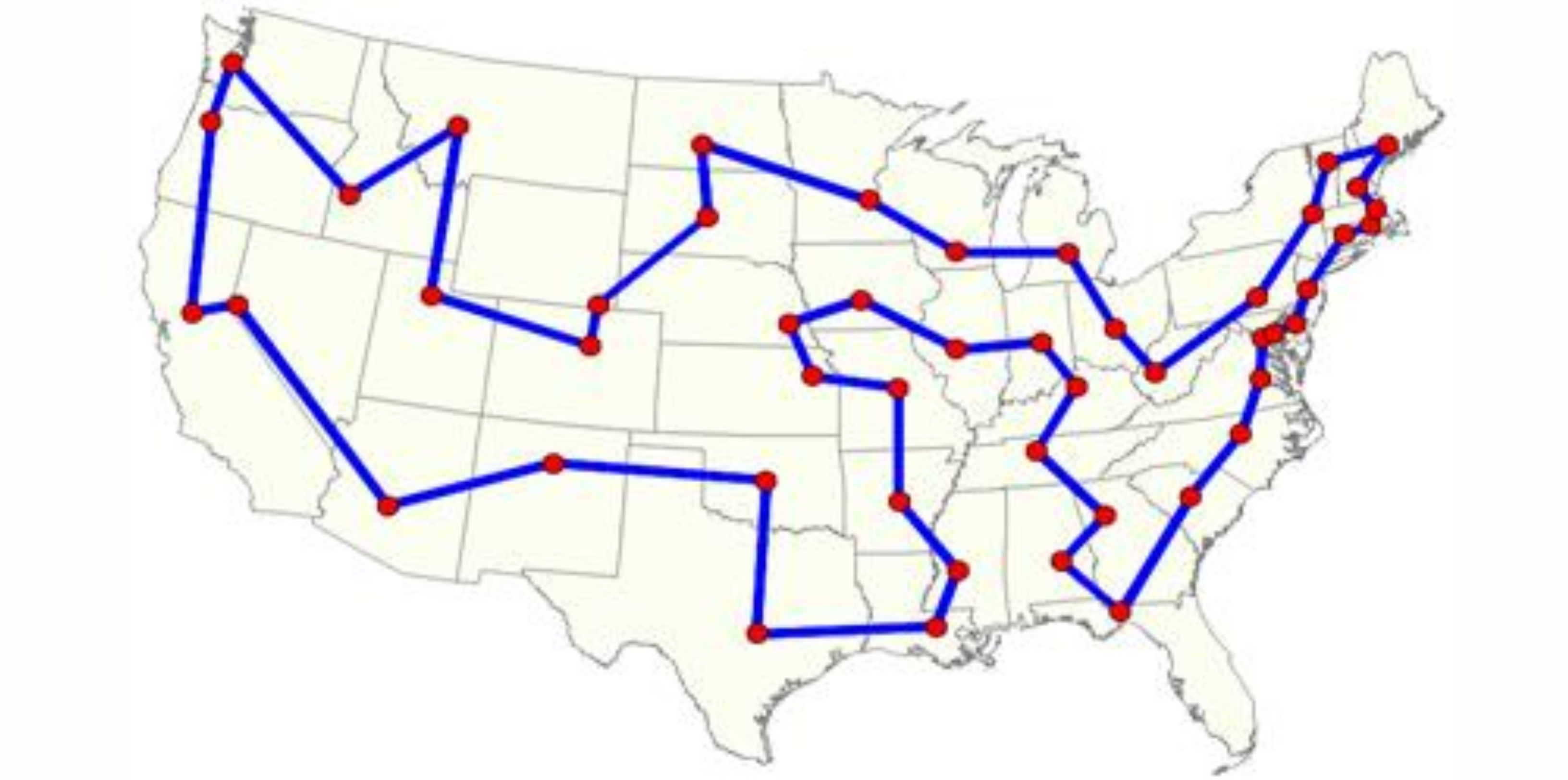
$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1 \ldots m \qquad \text{(constraints)}$$

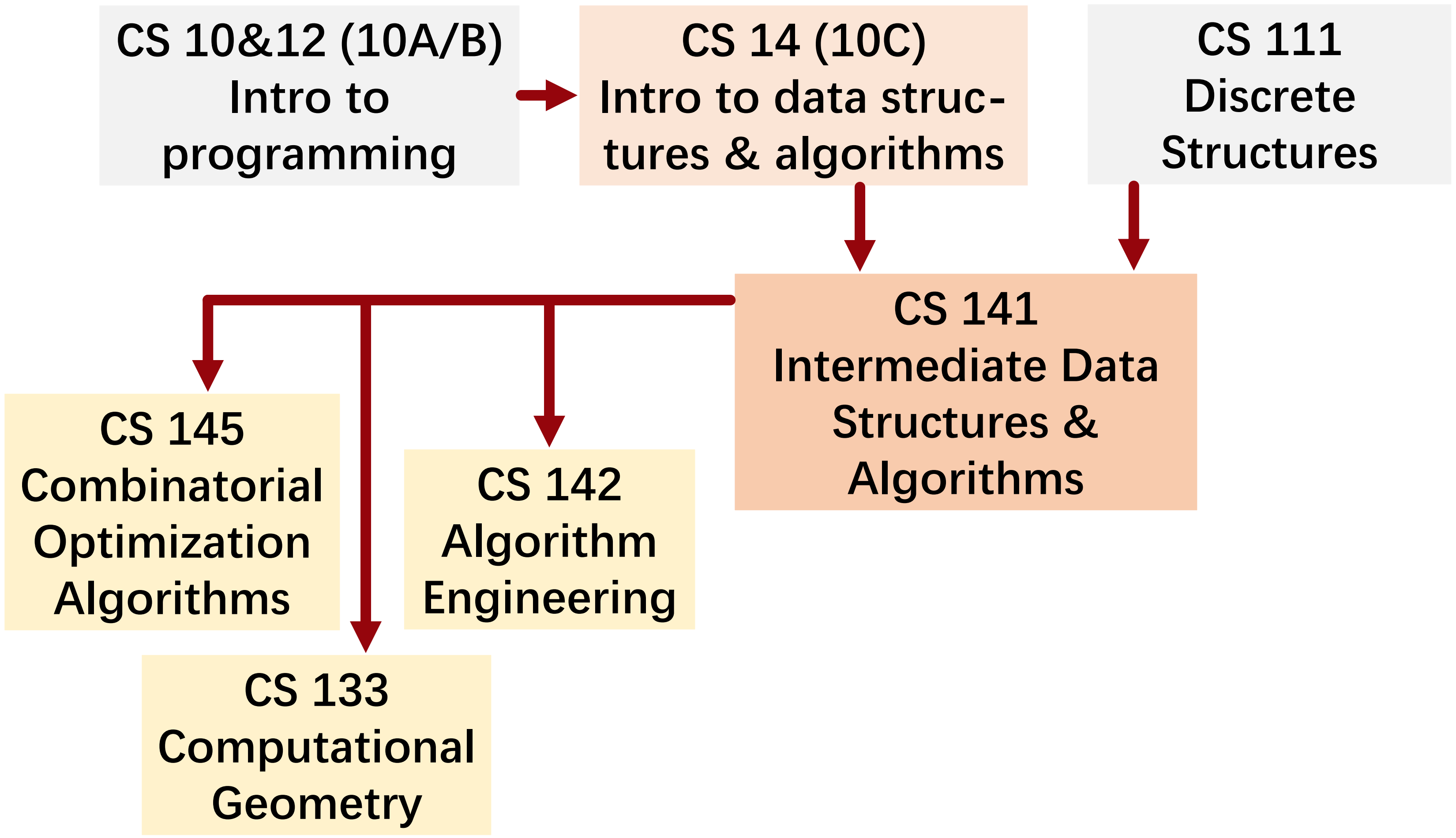
$$x_j \geq 0, \qquad j = 1 \ldots n \quad \text{(non-negativity constraints)}$$

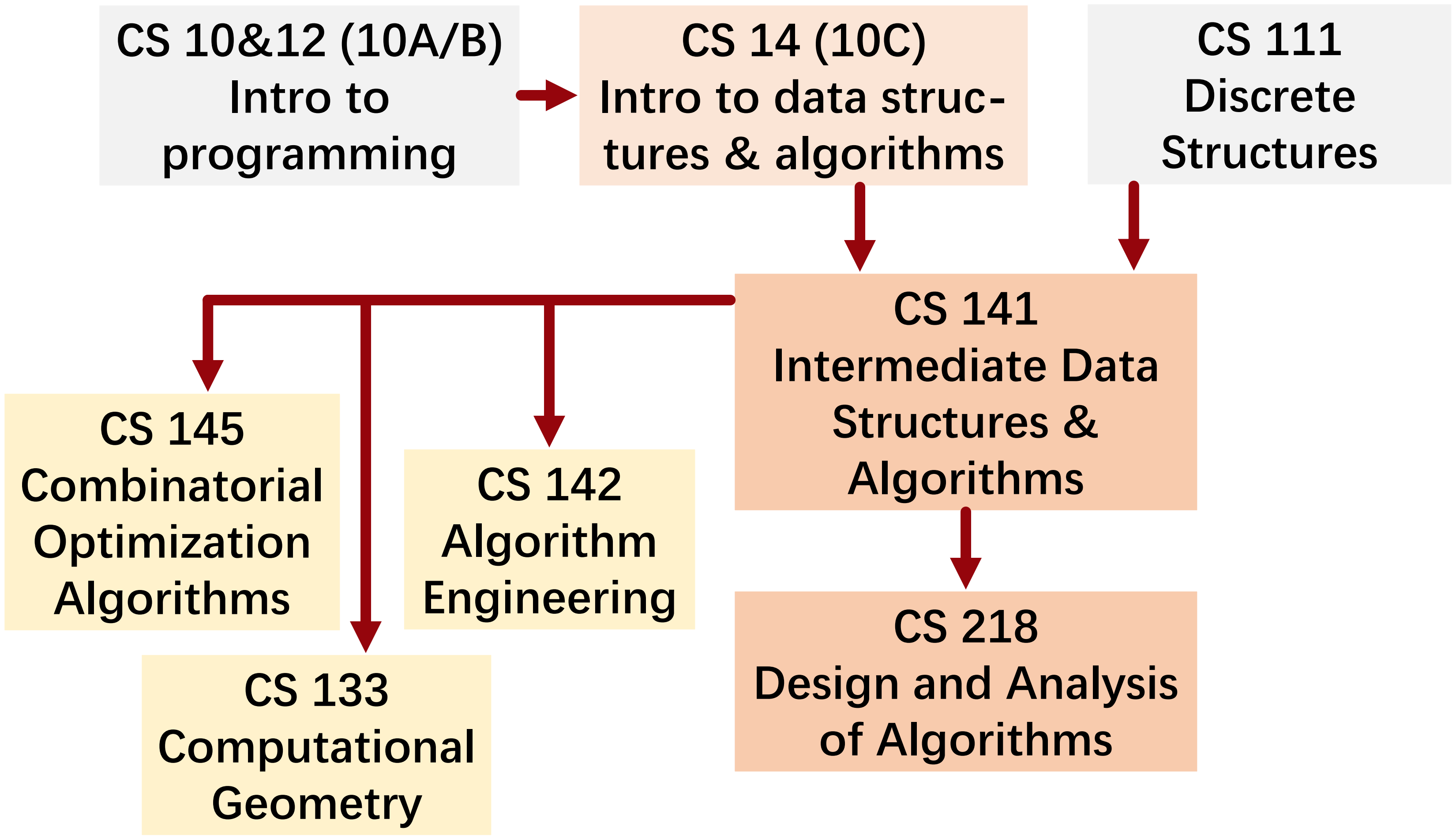
- **A generalization for many problems such as knapsack problems, network flow**
- **Most graph problems can be formalized as LP problems**

CS 145 Combinatorial Optimization Algorithms

- Last offering: Fall 2020
- By: Silas Richelson

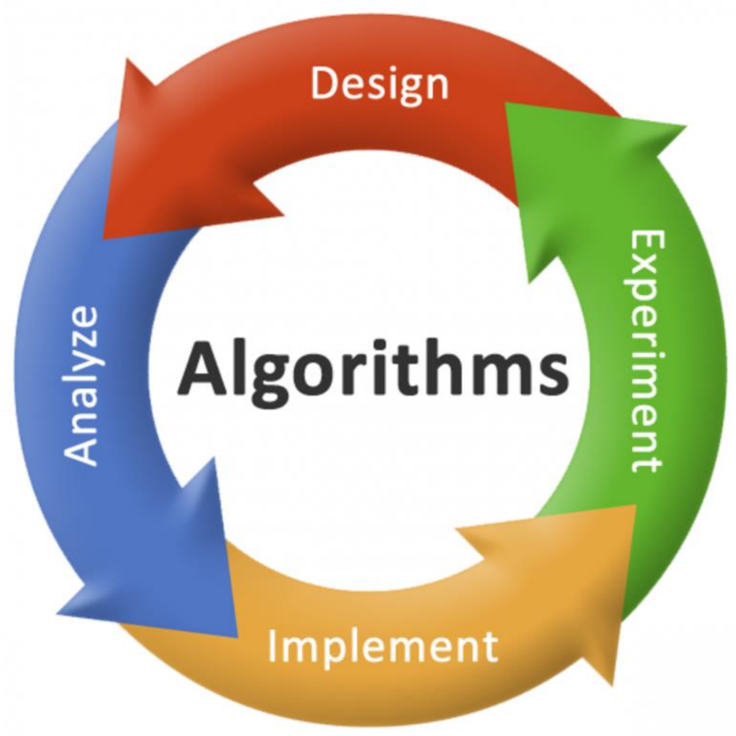



CS 10&12 (10A/B)
Intro to programming
CS 14 (10C)
Intro to data struc-tures & algorithms
CS 111
Discrete Structures
CS 141
Intermediate Data Structures & Algorithms
CS 145
Combinatorial Optimization Algorithms
CS 142
Algorithm Engineering
CS 133
Computational Geometry


CS 10&12 (10A/B)
Intro to programming
CS 14 (10C)
Intro to data structures & algorithms
CS 111
Discrete Structures
CS 141
Intermediate Data Structures & Algorithms
CS 145
Combinatorial Optimization Algorithms
CS 142
Algorithm Engineering
CS 133
Computational Geometry
CS 218
Design and Analysis of Algorithms

CS 218 Design And Analysis Of Algorithms

- Next offerings: Winter 2021 by Yihan, Spring 2021 by Yan
- Tier-1 graduate course



What are covered in CS 218 Design And Analysis Of Algorithms? (Winter 2021 version)

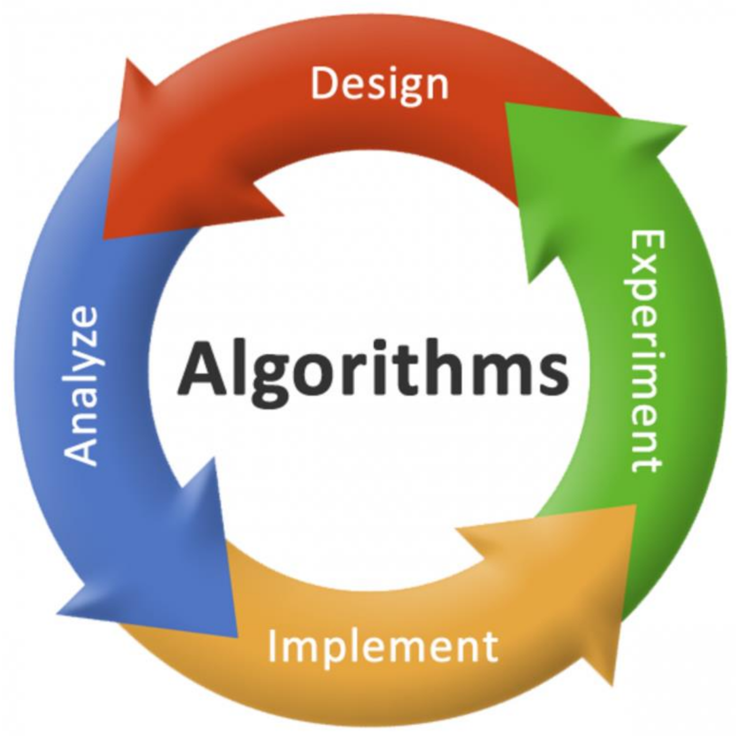
- **Divide-and-conquer (1 lecture)**
- **Greedy (1 lecture)**
- **Data structures (3 lectures)**
    - Winning trees, augmented trees, union-find, range trees
- **Dynamic programming (4 lectures)**
    - DP implementations, DP for games, DP on trees, DP with optimizations
- **Graph algorithms (3 lectures)**
    - Review of 141 algorithms with optimizations, topological sort, matching
- **Randomized algorithms and amortized analysis (2 lectures)**

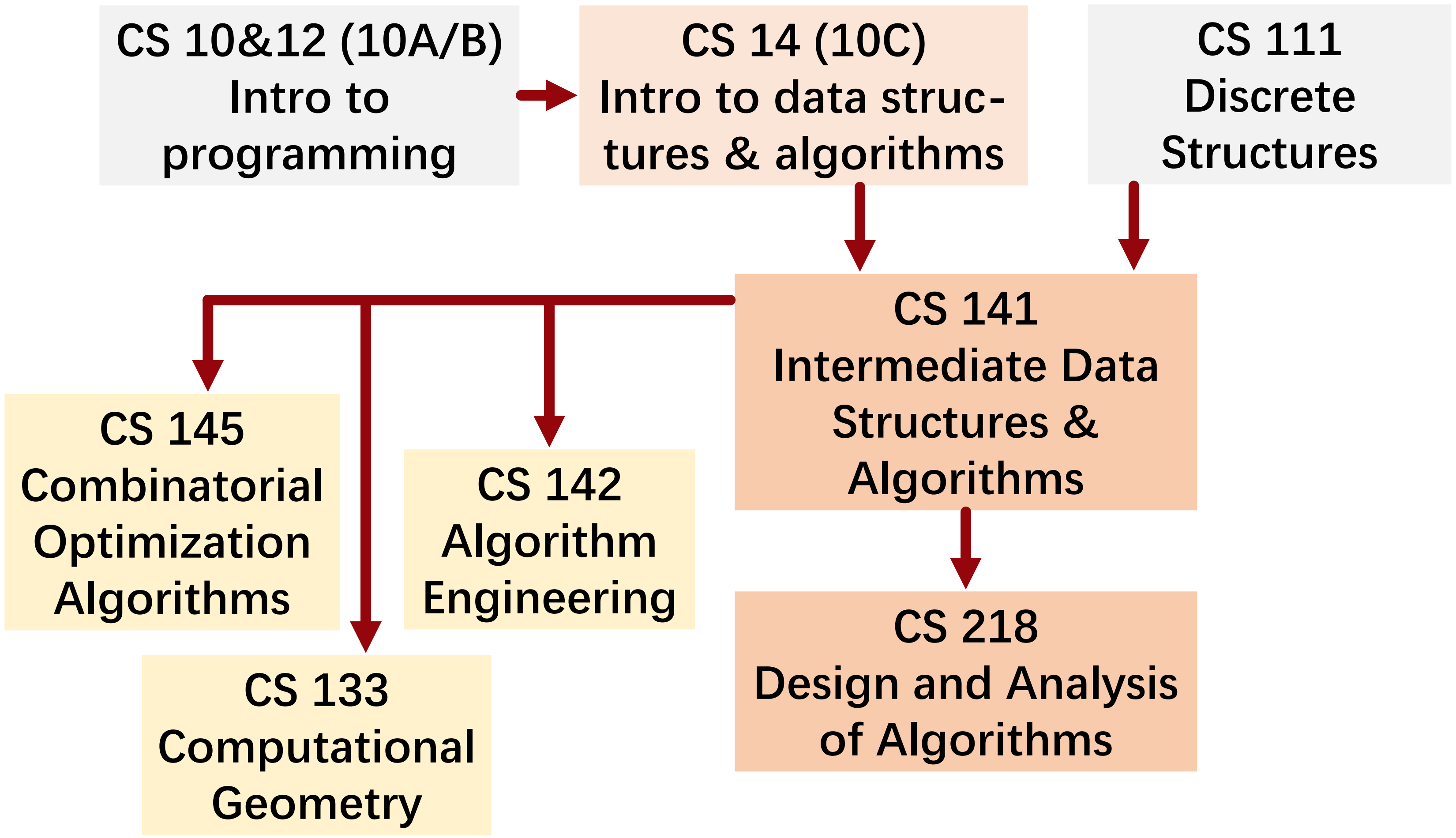
Our versions of algorithm courses make an emphasis on problem-solving

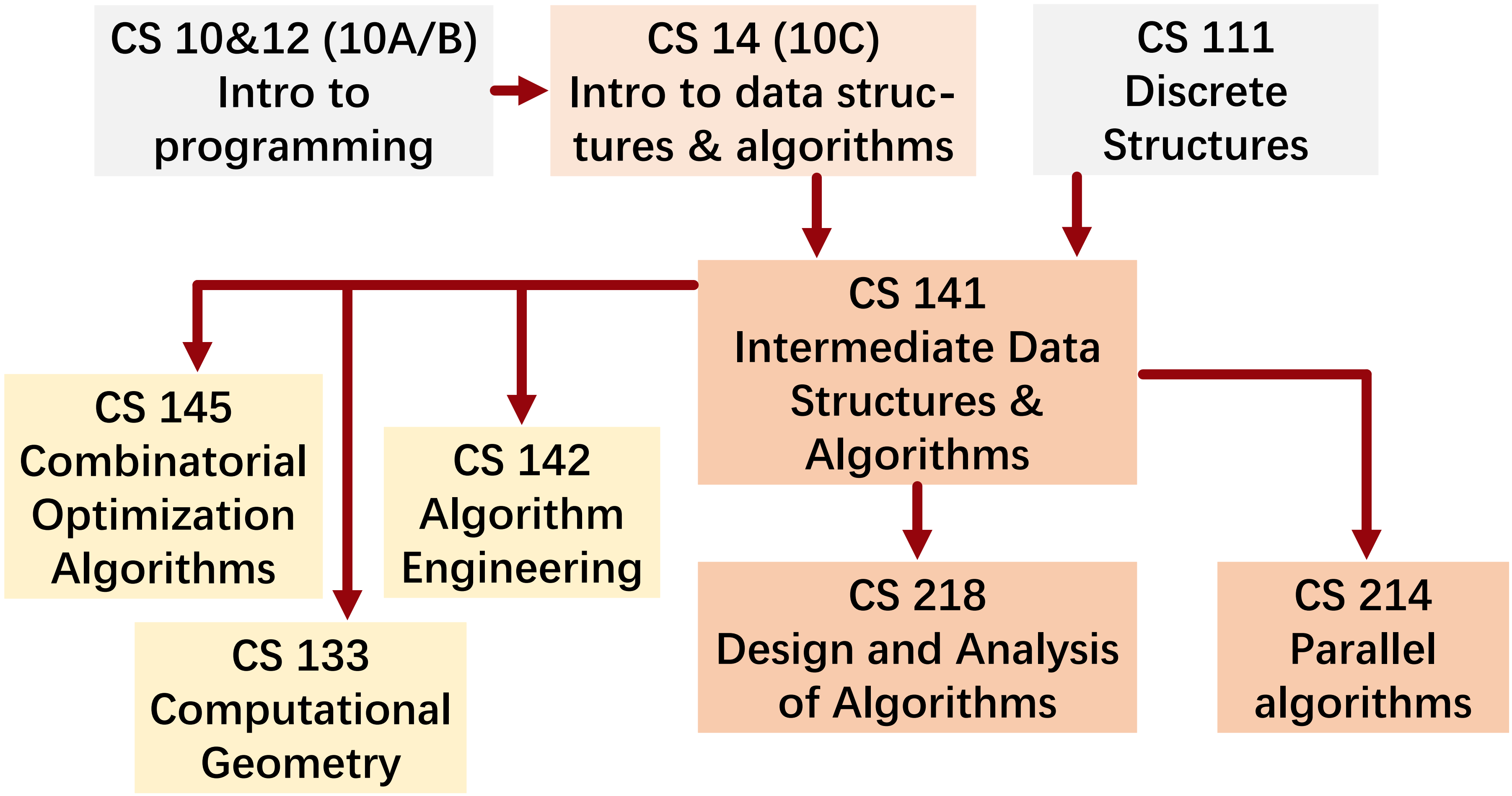
- (Winter 2021 version)
- Solving 15 problems with programming (similar to 142 programming assignments), 3 problems in every two weeks (25%)
- Written homework (20%)
- Midterm (25%) and Final (30%)
- (Optional proof track, lots of hard problems you can try to solve for bonus points)

CS 218 Design And Analysis Of Algorithms

- Next offerings: Winter 2021 by Yihan, Spring 2021 by Yan
- Tier-1 graduate course



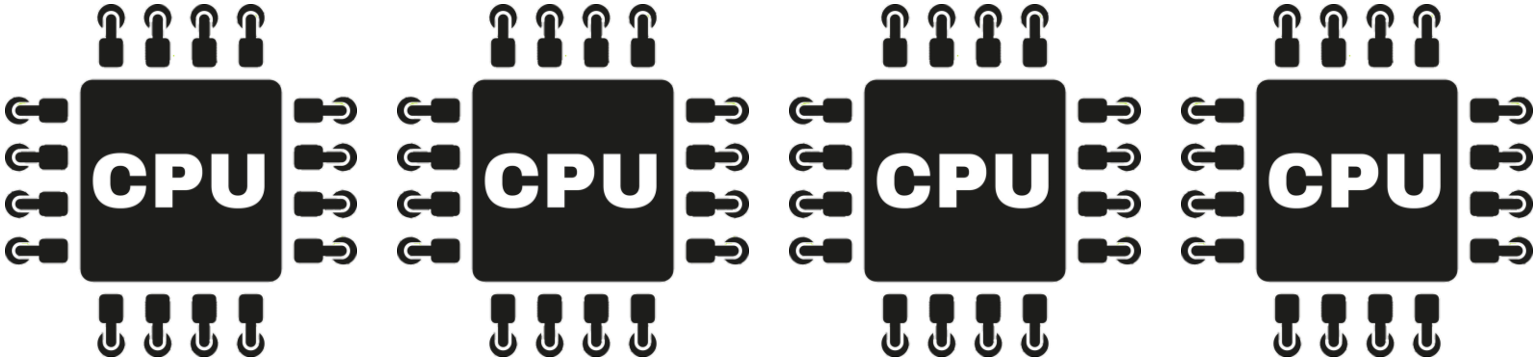

CS 10&12 (10A/B)
Intro to programming
CS 14 (10C)
Intro to data struc-tures & algorithms
CS 111
Discrete Structures
CS 141
Intermediate Data Structures & Algorithms
CS 145
Combinatorial Optimization Algorithms
CS 142
Algorithm Engineering
CS 133
Computational Geometry
CS 218
Design and Analysis of Algorithms

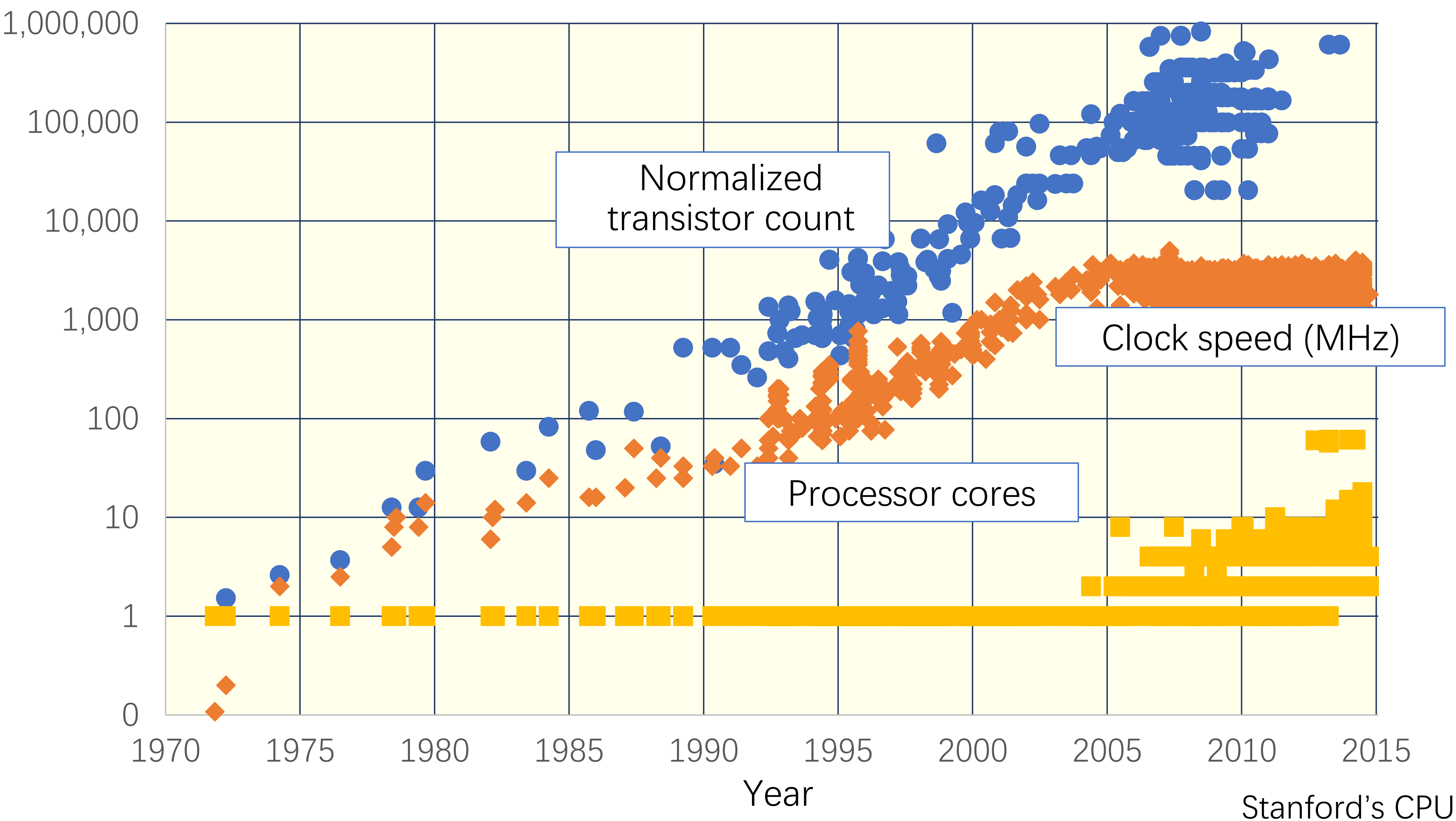


CS 10&12 (10A/B)
Intro to programming
CS 14 (10C)
Intro to data structures & algorithms
CS 111
Discrete Structures
CS 141
Intermediate Data Structures & Algorithms
CS 145
Combinatorial Optimization Algorithms
CS 142
Algorithm Engineering
CS 133
Computational Geometry
CS 218
Design and Analysis of Algorithms
CS 214
Parallel algorithms

CS 214 Parallel algorithms

- Next offering: Spring 2021 by Yihan Sun
- Tier-1 graduate course





Stanford's CPU DB [DKM12]

Powerful machines

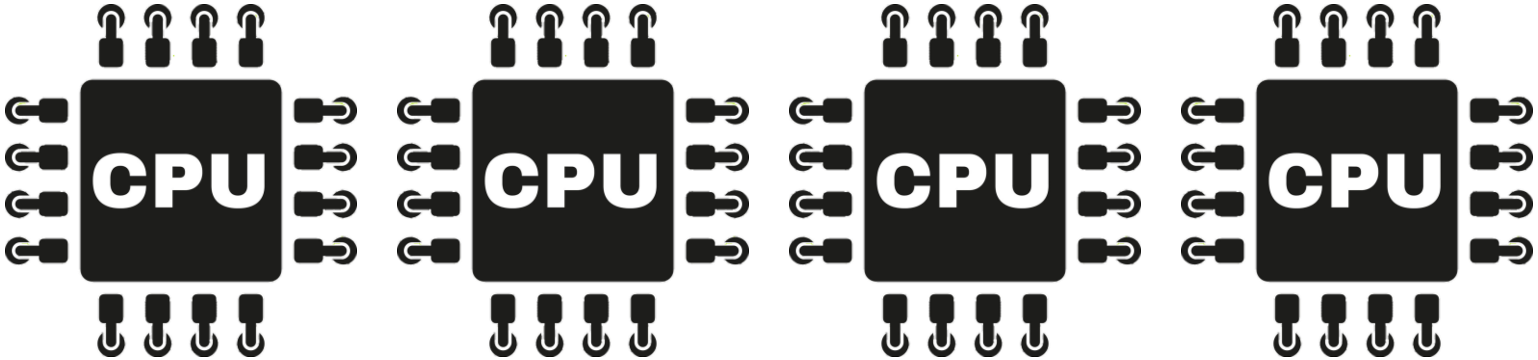
- 96 cores, 192 hyperthreads, 1.5 TB main memory
- No one in this world can make Dijkstra or Bellman-Ford 100x faster sequentially, but it is not too hard when we have this many of cores
- Every key component in a system or software is or will be run in parallel
- Learning parallel programming $\neq$ writing fast parallel code

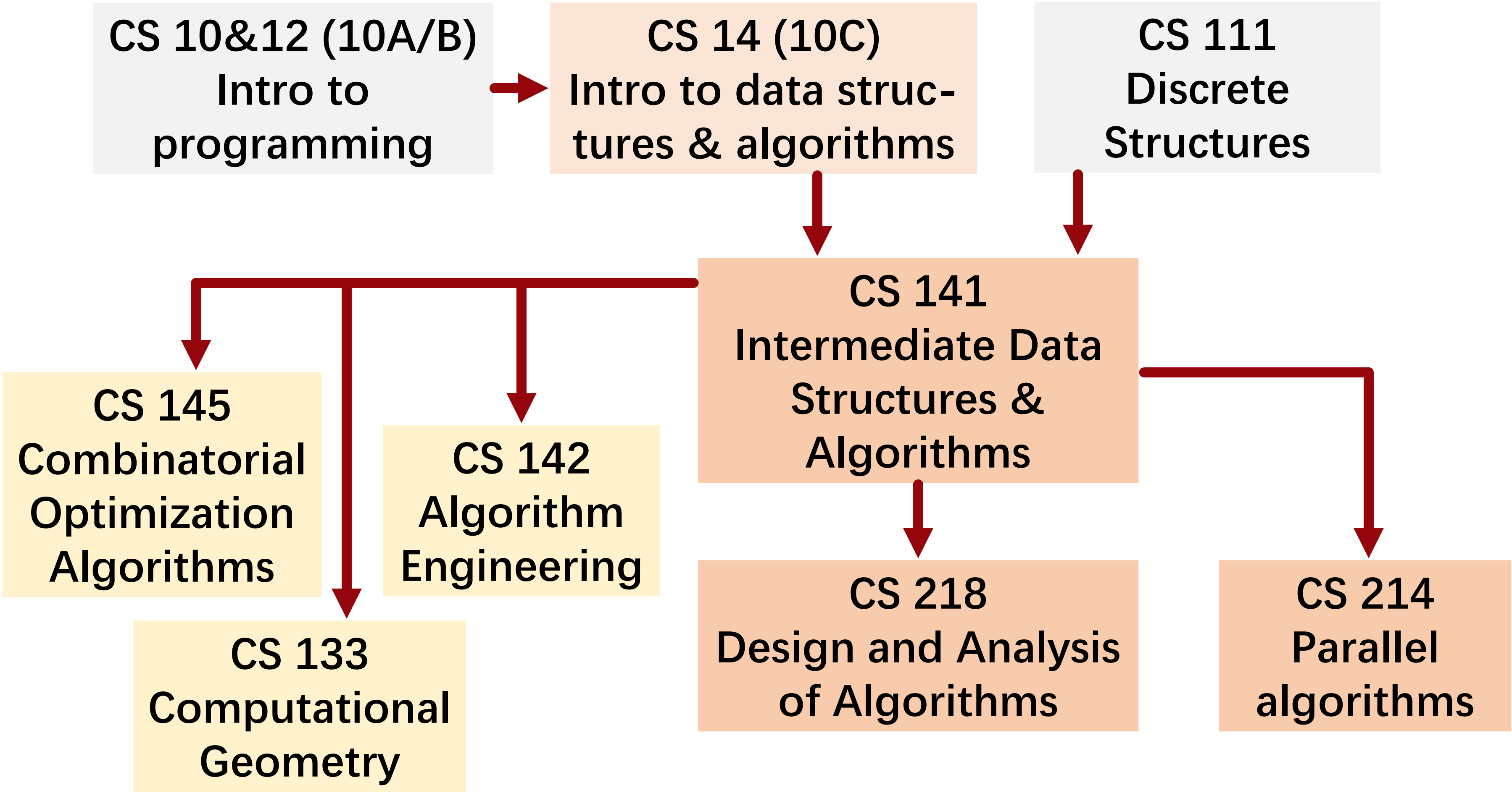
What are covered in CS 214 Parallel Algorithms?

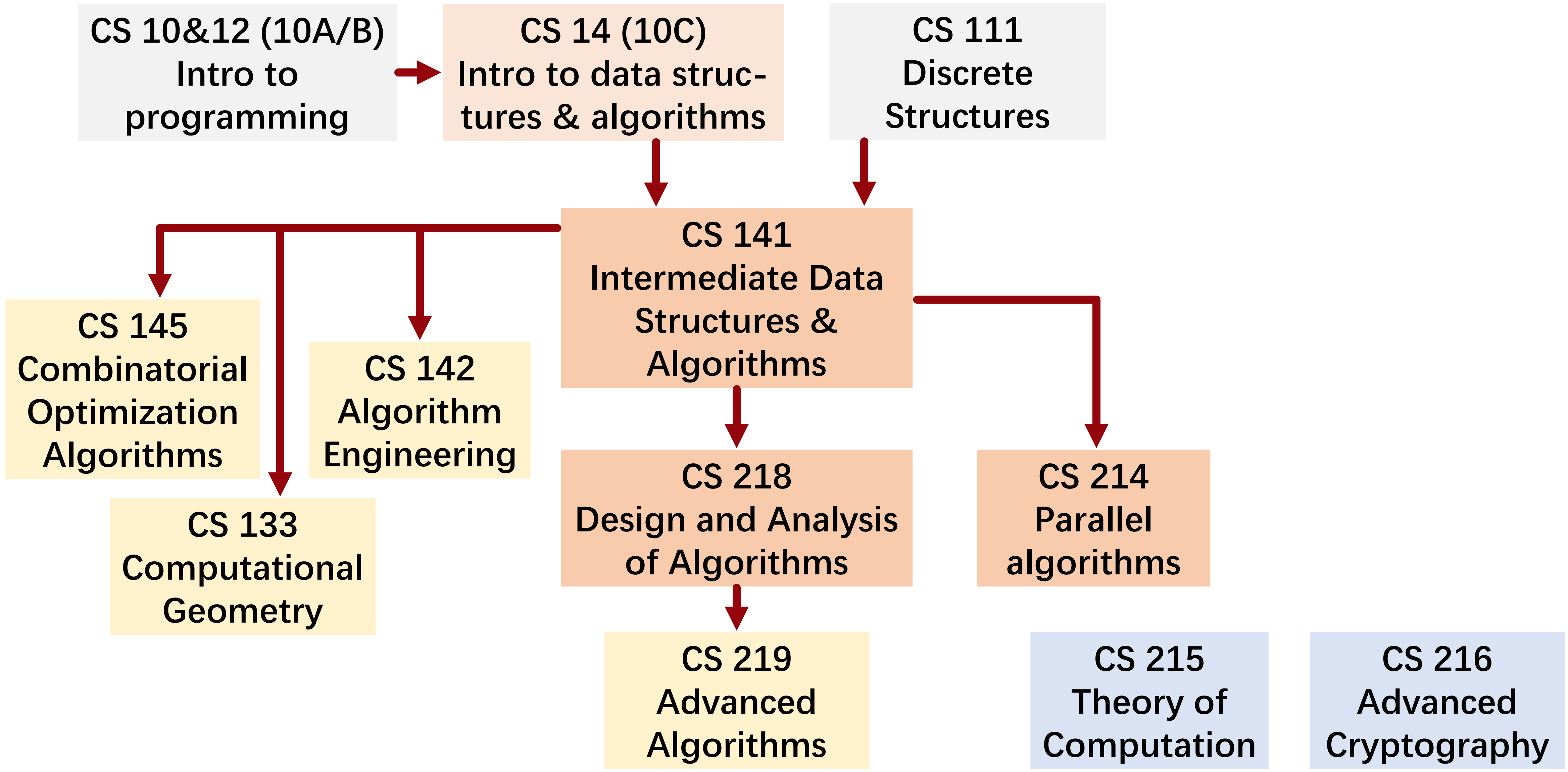
- Computational models
- Divide-and-conquer and recurrences
- Sorting
- Graph algorithms
- Data structures
- Locality and I/O-efficiency
- Scheduling

CS 214 Parallel algorithms

- Next offering: Spring 2021 by Yihan Sun
- Tier-1 graduate course




CS 10&12 (10A/B)
Intro to
programming
CS 14 (10C)
Intro to data struc-
tures & algorithms
CS 111
Discrete
Structures
CS 141
Intermediate Data
Structures &
Algorithms
CS 145
Combinatorial
Optimization
Algorithms
CS 142
Algorithm
Engineering
CS 133
Computational
Geometry
CS 218
Design and Analysis
of Algorithms
CS 214
Parallel
algorithms


CS 10&12 (10A/B) Intro to programming
CS 14 (10C) Intro to data struc-tures & algorithms
CS 111 Discrete Structures
CS 141 Intermediate Data Structures & Algorithms
CS 145 Combinatorial Optimization Algorithms
CS 142 Algorithm Engineering
CS 133 Computational Geometry
CS 218 Design and Analysis of Algorithms
CS 214 Parallel algorithms
CS 219 Advanced Algorithms
CS 215 Theory of Computation
CS 216 Advanced Cryptography

CS 219 Advanced algorithms

- **Tier-2 graduate course**

- **If I'm going to teach it AY 21-22, I would probably cover:**
  - Randomization and sampling
  - High probability analysis
  - Advanced graph techniques: sparsification, low-stretch embedding, spectrum graph theory
  - Range and nearest neighbor search (in low and high dimension)
  - streaming/online algorithms

CS 215 Theory of Computation

- Next offerings: Winter 2021 by Amey Bhangale
- Tier-2 graduate course

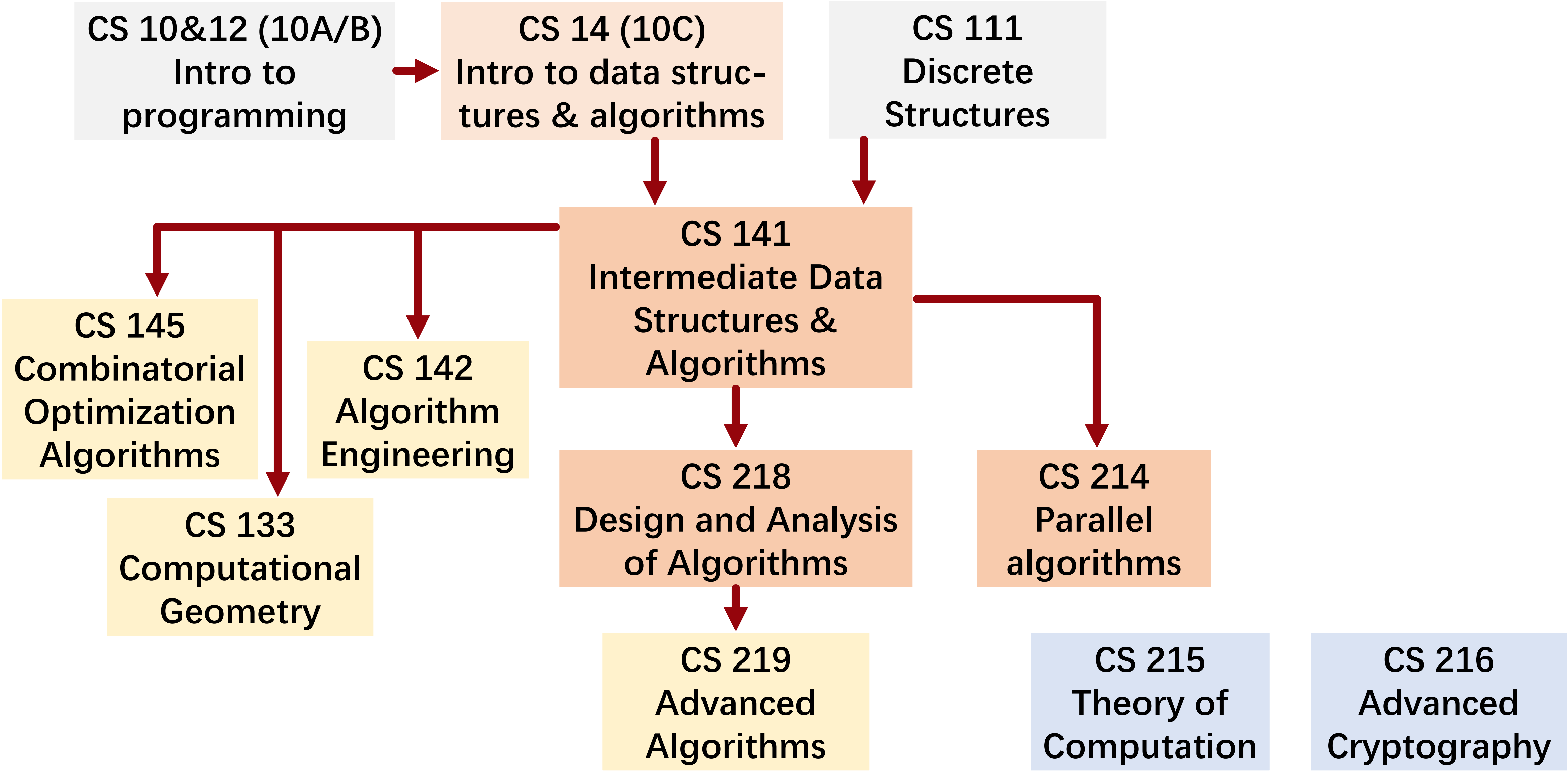
- List of topics:

Turing machines and computability theory
Decidability
Undecidability and the Halting Problem
Reducibility
PCP and the mapping reducibility
Time complexity classes
The famous classes P and NP
NP-completeness
More examples of NP-complete problems
Space complexity and Savitch's Theorem
PSPACE and PSPACE-completeness
Class NL

CS 216 Advanced Cryptography

- **Next offerings: Winter 2021 by Silas Richelson**
- **Tier-2 graduate course**

- **List of topics include formal models of security and applications:**
  - Public key encryption
  - Digital signatures
  - Secure protocols
  - Etc.


CS 10&12 (10A/B)
Intro to
programming
CS 14 (10C)
Intro to data struc-
tures & algorithms
CS 111
Discrete
Structures
CS 141
Intermediate Data
Structures &
Algorithms
CS 145
Combinatorial
Optimization
Algorithms
CS 142
Algorithm
Engineering
CS 133
Computational
Geometry
CS 218
Design and Analysis
of Algorithms
CS 214
Parallel
algorithms
CS 219
Advanced
Algorithms
CS 215
Theory of
Computation
CS 216
Advanced
Cryptography

CS 142 is not just an end, but it can also be a start for your algorithm journey

- **UCR offers 8 related algorithm/theory courses after CS 141**
  - 4 are new this year (142, 214, 216, 219)
- **The problem-solving ability is valuable for a variety of purposes**
  - Yihan and I will provide you further opportunities for training in existing courses: 142 (W21), 218 (W21, S21), and 214 (S21)
- **Potential future paths:**
  - Get sufficient training before finding a job (right after graduation)
  - Combined BS + MS Programs at UCR (cover both width and depth)
  - Research or attend another graduate school
- **More questions, arrange 1-on-1 meeting with me:** https://docs.google.com/spreadsheets/d/1zaHLyGvIEtokIaJwXZjBf014_Tw6S_qLZ9snsDA5bis/edit#gid=0