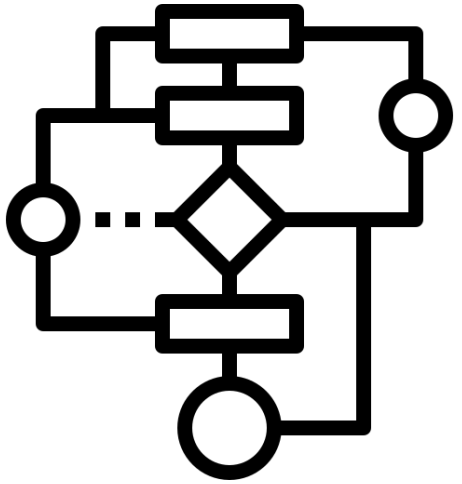


CS 310: Algorithms

Instructor: Naveed Anwar Bhatti

Who am I? Dr. Naveed Anwar Bhatti

Hometown: Islamabad

Postdoc:

2019 Senior Researcher
RISE, Stockholm, Sweden

Education:

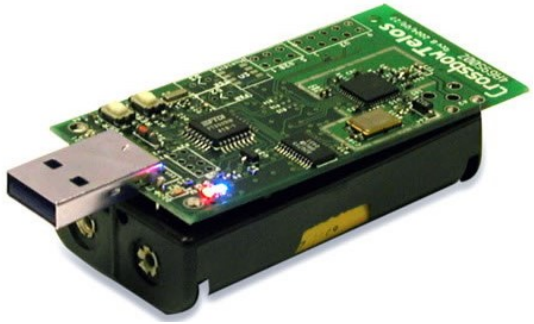
PhD 2018
Computer Science
Politecnico di Milano, Italy
System Support for Transiently Powered Embedded Systems

MS 2013
Computer Science
FAST-NUCES, Islamabad, Pakistan
Long range RFID System: Decoupling sensing and energy in sensor networks using energy transference

BS 2011
Telecom
FAST-NUCES, Islamabad, Pakistan
Internet Controlled Unmanned Ground Vehicle

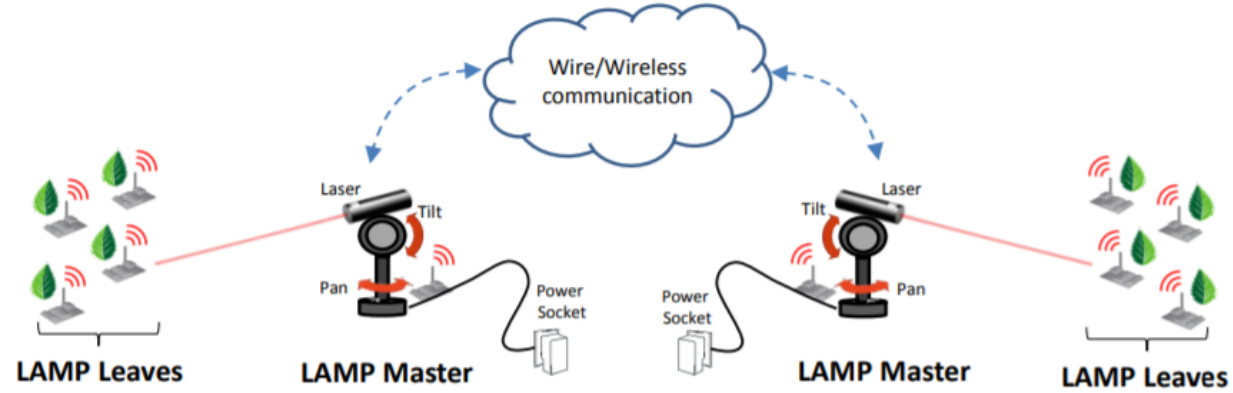
Long range RFID-like System

Laser Module



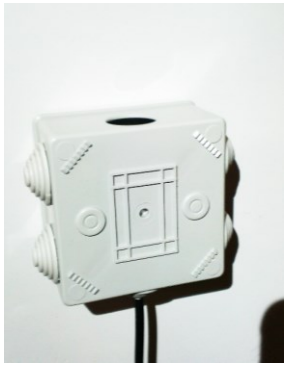
TelosB mote

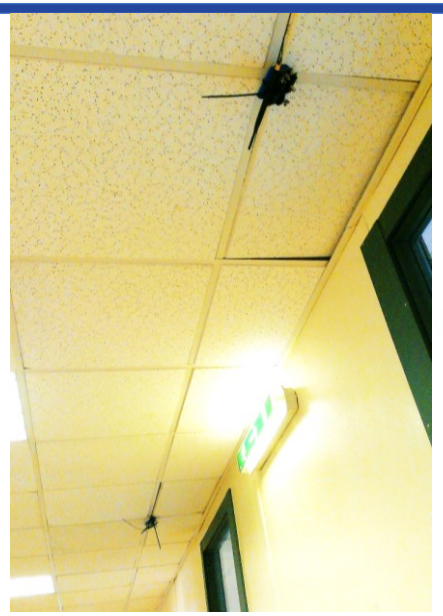
Solar Panel



Long range RFID-like System

Other Sensor Deployments





Waspmote

How to reach me?

Email: naveed.bhatti@lums.edu.pk
Room: 9 - G23A
Office Hours: Thursday, 11:00 am - 12:00 pm
Webpage: naveedanwarbhatti.github.io

Class page and slides: LUMS LMS

Why are YOU here? - Outline

Week 1 and 2	Introduction to Algorithms, Analysis and Asymptotic growth
Week 3	Graphs
Week 4 and 5	Greedy Algorithms
Week 6 and 7	Divide and Conquer
Week 8, 9 and 10	Dynamic Programming
Week 11 and 12	Network Flow
Week 13 and 14	NP Completeness

GRADING BREAKUP AND POLICY

Assessment	Weight (%)	Related CLOs
Assignments	**10%**	CLO1- CLO4
Quizzes	**25%** (We will have N - 1 policy for quizzes. 1 quizzes will be dropped. No petition for makeup quizzes will be accepted if you have missed only 1 quiz. If you have missed more than 1 quiz for genuine reasons, you may file a petition for the 2nd and subsequent missed quizzes. However, the petition may not be accepted if there is no substantial reason.)	CLO1 - CLO3
Mid-term	**27%**	CLO1 - CLO3
Final	**38%**	CLO1 - CLO4

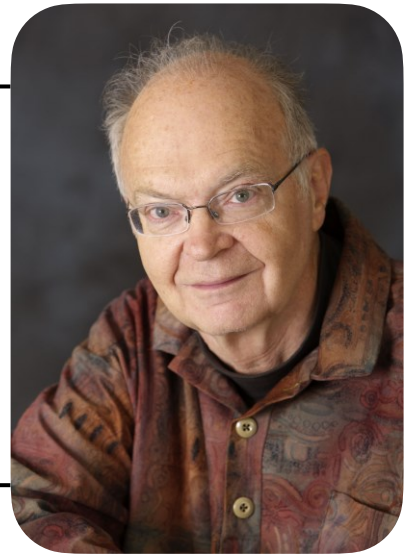
Lecture slides

- Bad experience from previous courses
- Lecture slides are **not** the course content
  - Discussions and explanations in between are equally important
  - To extract maximum benefit, take notes (like the good old days!).
  - I will selectively show/use slides

Algorithm Definition

“ An algorithm is a finite, definite, effective procedure, with some input and some output. ”

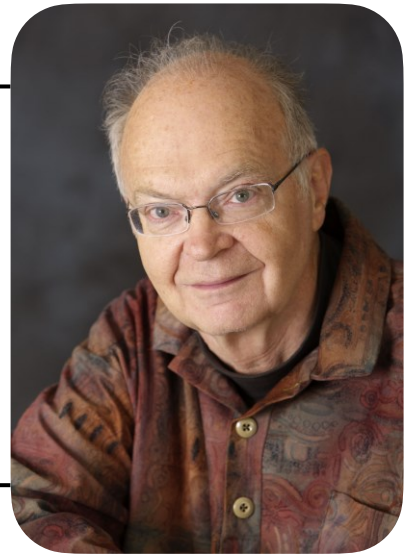
— Donald Knuth



Algorithm Definition

*" An algorithm is a **finite**, **definite**, **effective** procedure, with some input and some output. "*

— Donald Knuth



Why study algorithms?

To become a proficient programmer.

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about ***data structures*** *and their* ***relationships****."*

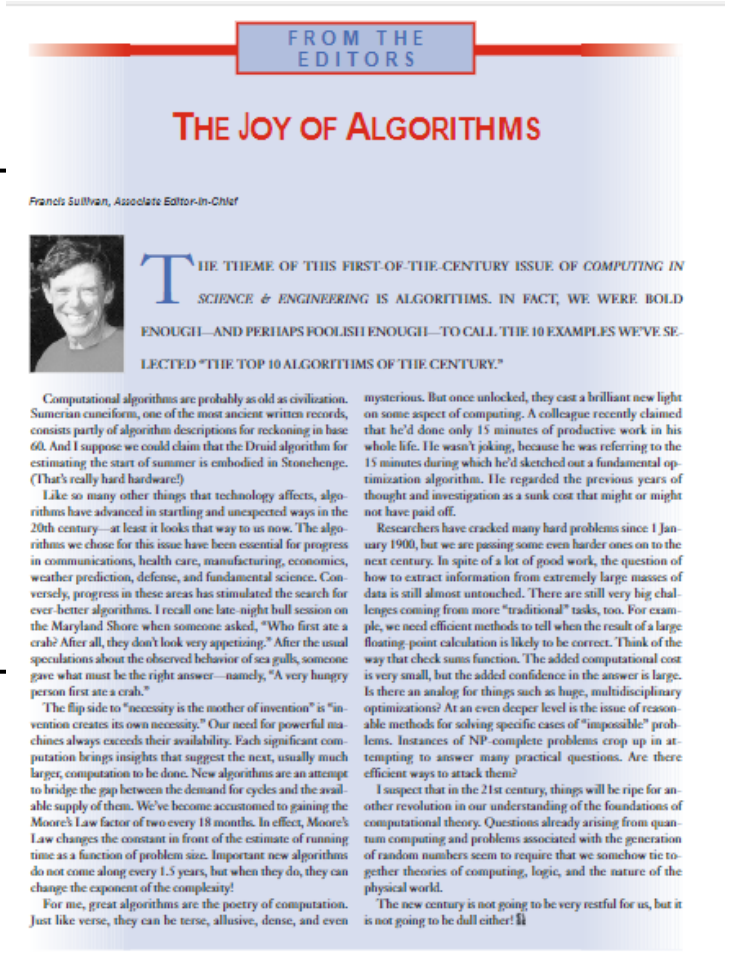
— Linus Torvalds (creator of Linux)

Why study algorithms?

For intellectual stimulation.

"For me, great algorithms are the poetry of computation. Just like verse, they can be allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing."

— Francis Sullivan



FROM THE EDITORS

THE JOY OF ALGORITHMS

Francis Sullivan, Associate Editor-in-Chief

THE THEME OF THIS FIRST-OF-THE-CENTURY ISSUE OF *COMPUTING IN SCIENCE & ENGINEERING* IS ALGORITHMS. IN FACT, WE WERE BOLD ENOUGH—AND PERHAPS FOOLISH ENOUGH—TO CALL THE 10 EXAMPLES WE'VE SELECTED "THE TOP 10 ALGORITHMS OF THE CENTURY."

Computational algorithms are probably as old as civilization. Sumerian cuneiform, one of the most ancient written records, consists partly of algorithm descriptions for reckoning in base 60. And I suppose we could claim that the Druid algorithm for estimating the start of summer is embodied in Stonehenge. (That's really hard hardware!)

Like so many other things that technology affects, algorithms have advanced in startling and unexpected ways in the 20th century—at least it looks that way to us now. The algorithms we chose for this issue have been essential for progress in communications, health care, manufacturing, economics, weather prediction, defense, and fundamental science. Conversely, progress in these areas has stimulated the search for ever-better algorithms. I recall one late-night bull session on the Maryland Shore when someone asked, "Who first ate a crab? After all, they don't look very appetizing." After the usual speculations about the observed behavior of sea gulls, someone gave what must be the right answer—namely, "A very hungry person first ate a crab."

The flip side to "necessity is the mother of invention" is "invention creates its own necessity." Our need for powerful machines always exceeds their availability. Each significant computation brings insights that suggest the next, usually much larger, computation to be done. New algorithms are an attempt to bridge the gap between the demand for cycles and the available supply of them. We've become accustomed to gaining the Moore's Law factor of two every 18 months. In effect, Moore's Law changes the constant in front of the estimate of running time as a function of problem size. Important new algorithms do not come along every 1.5 years, but when they do, they can change the exponent of the complexity!

For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing. A colleague recently claimed that he'd done only 15 minutes of productive work in his whole life. He wasn't joking, because he was referring to the 15 minutes during which he'd sketched out a fundamental optimization algorithm. He regarded the previous years of thought and investigation as a sunk cost that might or might not have paid off.

Researchers have cracked many hard problems since 1 January 1900, but we are passing some even harder ones on to the next century. In spite of a lot of good work, the question of how to extract information from extremely large masses of data is still almost untouched. There are still very big challenges coming from more "traditional" tasks, too. For example, we need efficient methods to tell when the result of a large floating-point calculation is likely to be correct. Think of the way that check sums function. The added computational cost is very small, but the added confidence in the answer is large. Is there an analog for things such as huge, multidisciplinary optimizations? At an even deeper level is the issue of reasonable methods for solving specific cases of "impossible" problems. Instances of NP-complete problems crop up in attempting to answer many practical questions. Are there efficient ways to attack them?

I suspect that in the 21st century, things will be ripe for another revolution in our understanding of the foundations of computational theory. Questions already arising from quantum computing and problems associated with the generation of random numbers seem to require that we somehow tie together theories of computing, logic, and the nature of the physical world.

The new century is not going to be very restful for us, but it is not going to be dull either!

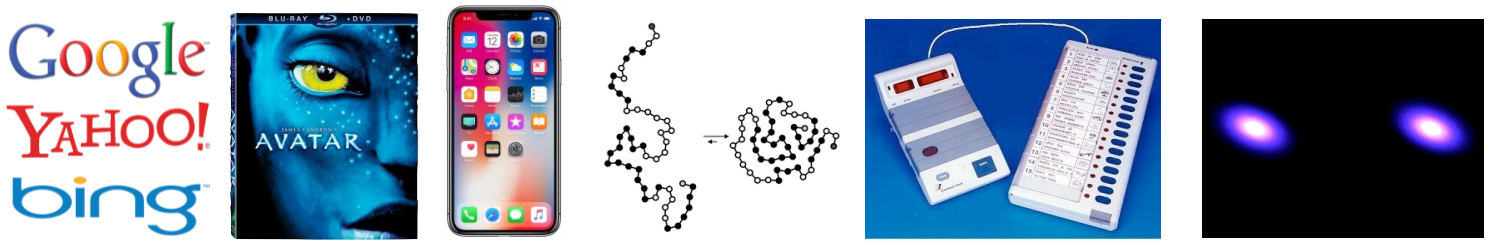
Why study algorithms?

For fun and profit.

Why study algorithms?

Internet. Web search, packet routing, distributed file sharing, ...
Biology. Human genome project, protein folding, …
Computers. Circuit layout, databases, caching, networking, compilers, …
Computer graphics. Movies, video games, virtual reality, …
Security. Cell phones, e-commerce, voting machines, …
Multimedia. MP3, JPG, DivX, HDTV, face recognition, …
Social networks. Recommendations, news feeds, advertisements, …
Physics. Particle collision simulation, n-body simulation, …

⋮



Slide credit: Kevin Wayne. Theory of Algorithms (COS423)
www.cs.princeton.edu/courses/archive/spring18/cos423

A First Problem: **Stable Matching**

- Nicely illustrates many of the **themes we will be emphasizing**

- Target Applications:
  - Selecting teaching assistants (TA) for courses – TA-course matching problem
  - Matching employers to applicants for job hiring
  - College admission – matching students to colleges
  - Content delivery networks – assigning users to web servers

A First Problem: **Stable Matching**

- Nicely illustrates many of the **themes we will be emphasizing**

- Target Applications:
  - **Selecting teaching assistants (TA) for courses – TA-course matching problem**
  - Matching employers to applicants for job hiring
  - College admission – matching students to colleges
  - Content delivery networks – assigning users to web servers

Stability Matching – Formulating the Problem

TA-course matching problem

- Suppose:
  - ***n*** courses are being offered by the CS Department in the current semester
  - Due to budget constraints **only one TA can be assigned to each course** – *n* TAs for *n* courses
  - Each course instructor ranks all the TA applicants in the order of his/her preference (**preference list of the course instructor**)
  - Each TA applicant ranks all the courses in the order of his/her preference (**preference list of TA applicant**)
- Issues
  - An applicant accepting TAship offer for a course may later quit it for a different course that is ranked higher in his/her preference list
  - A course instructor may withdraw an offer accepted by an applicant to hire another TA who is ranked higher in the instructor's preference list
  - Unstable matching

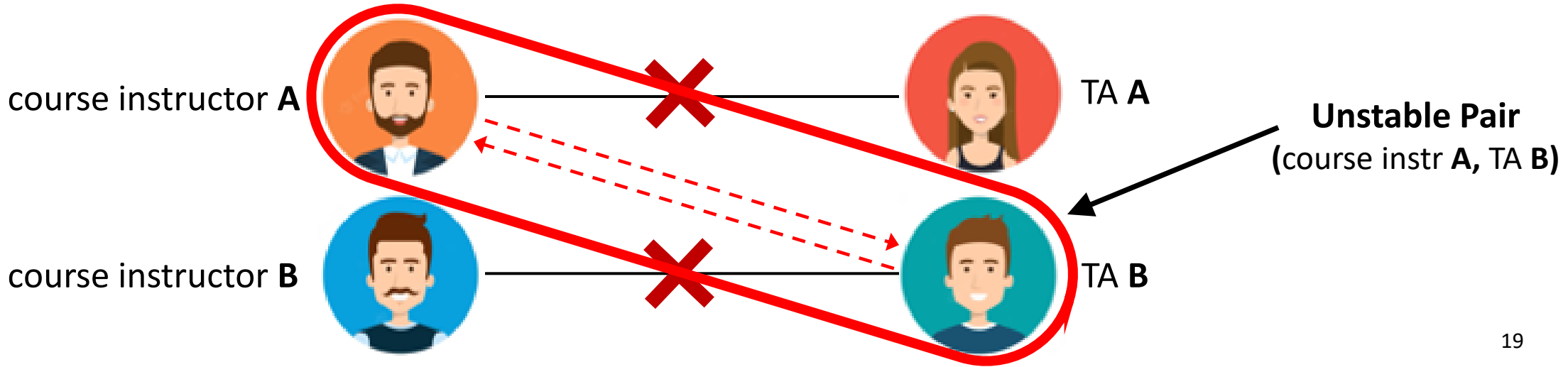
TA-course matching problem

Goal. Given a set of preferences among course instructors and TA applicants, design a self-reinforcing TA selection process

---

Unstable pair. Course instructor ***c*** and TA applicant ***a*** form an unstable pair if both:

- ***c*** prefers ***a*** to an already selected TA
- ***a*** prefers ***c*** to the already assigned course instructor



TA-course matching problem

Goal. Given a set of preferences among course instructors and TA applicants, design a self-reinforcing TA selection process

---

Unstable pair. Course instructor ***c*** and TA applicant ***a*** form an unstable pair if both:

- ***c*** prefers ***a*** to an already selected TA
- ***a*** prefers ***c*** to the already assigned course instructor

---

Stable assignment. Assignment with **no unstable pairs**.

- Prevents TAs quitting the assigned courses
- Prevent instructor withdrawing TAship offer

TA course matching problem - Example

Input. A set ***C*** of *n* course instructors and a set ***A*** of *n* TA applicants

- Each course instructor $c \in C$ ranks TA applicants
- Each TA applicant $a \in A$ ranks courses

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list

Matching

- Def. A matching M is a set of ordered pairs $c - a$ with $c \in C$ and $a \in A$ such that:
  - Each course instructor $c \in C$ appears **in at most one pair** of M
  - Each TA applicant $a \in A$ appears **in at most one pair** of M

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list

M = {(CS100 – Charlie)}

Matching

- Def. A matching M is a set of ordered pairs $c - a$ with $c \in C$ and $a \in A$ such that:
  - Each course instructor $c \in C$ appears **in at most one pair** of M
  - Each TA applicant $a \in A$ appears **in at most one pair** of M

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list

M = {(CS100 – Charlie)}
M = {(CS100 – Charlie), (CS200 – Bob)}

Perfect matching

Def. A matching M is perfect if $|M| = |C| = |A| = n$

- Each course instructor $c \in C$ appears **in one pair** of M
- Each TA applicant $a \in A$ appears **in one pair** of M

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list

A perfect matching M = {(CS100 – Charlie), (CS200 – Bob), (CS300 – Alice)}

Unstable pair

- Def. Given a perfect matching M, course instructor $\boldsymbol{c}$ and a TA applicant $\boldsymbol{a}$ form an unstable pair if both
  - c prefers a to the assigned TA
  - a prefers c to the assigned course

M = {(CS100 – Charlie), (CS200 – Bob), (CS300 – Alice)}

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list

Can you find any unstable pair?

Unstable pair

- Def. Given a perfect matching M, course instructor $\boldsymbol{c}$ and a TA applicant $\boldsymbol{a}$ form an unstable pair if both
  - c prefers a to the assigned TA
  - a prefers c to the assigned course

M = {(CS100 – Charlie), (CS200 – Bob), (CS300 – Alice)}

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list

(CS100 – Alice) are unstable pair

Unstable pair – Through Bipartite Graph

- Def. Given a perfect matching M, course instructor c and a TA applicant a form an unstable pair if both
  - c prefers a to the assigned TA
  - a prefers c to the assigned course

M = {(CS100 – Charlie), (CS200 – Bob), (CS300 – Alice)}

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list

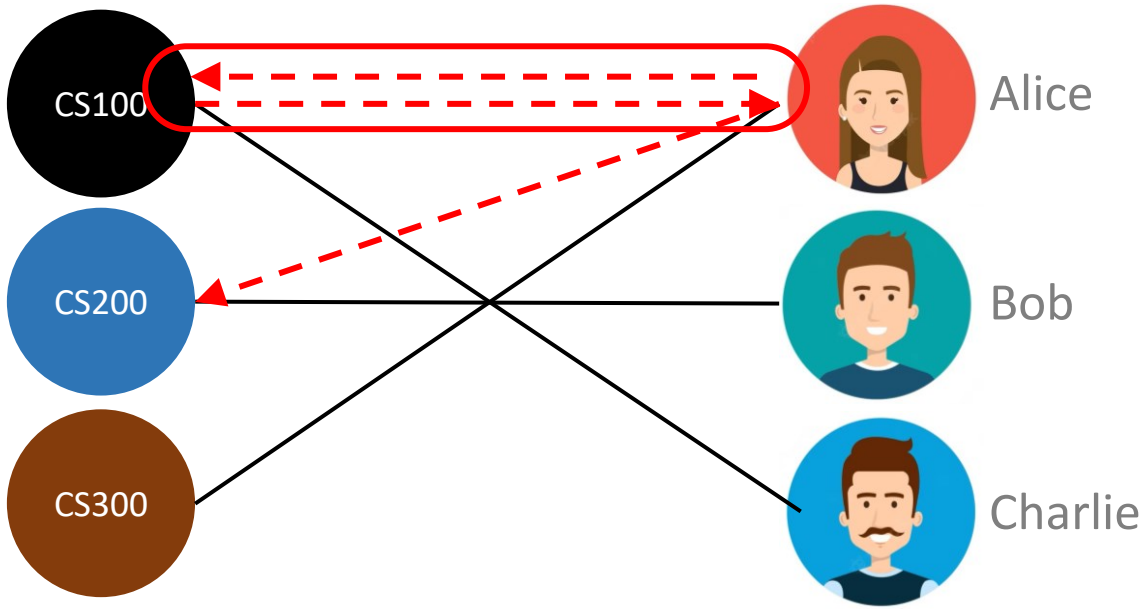
(CS100 – Alice) are unstable pair

Unstable pair – Through Bipartite Graph

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

M = {(CS100 – Charlie), (CS200 – Bob), (CS300 – Alice)}



(CS100 – Alice) are unstable pair

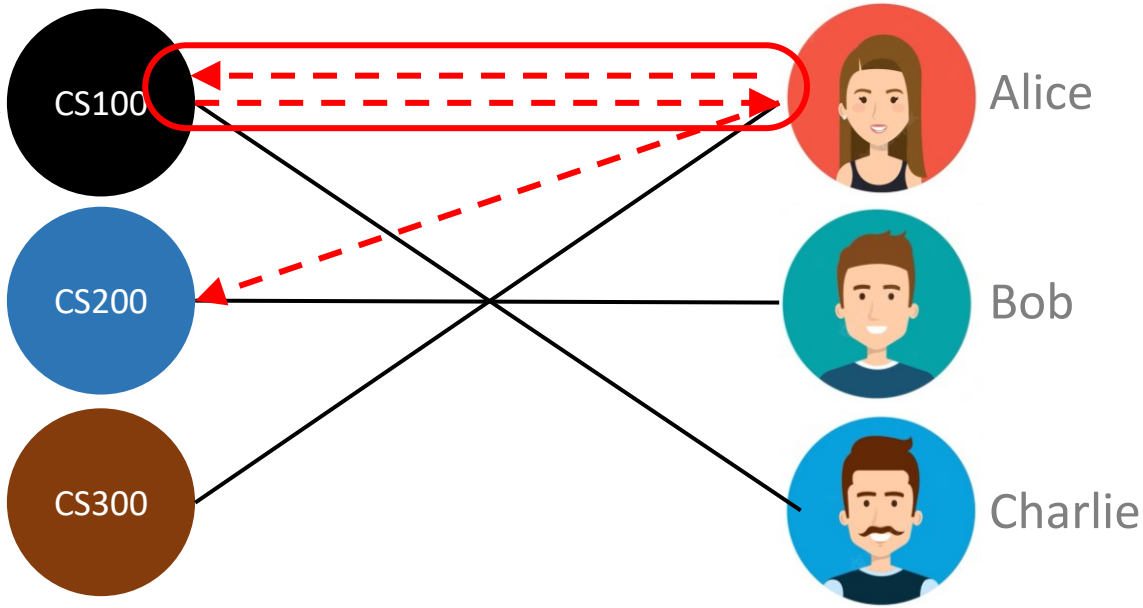
Unstable pair – How to resolve?

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

M = {(CS100 – Charlie), (CS200 – Bob), (CS300 – Alice)}

an unstable pair $c - a$ could each improve by joint action



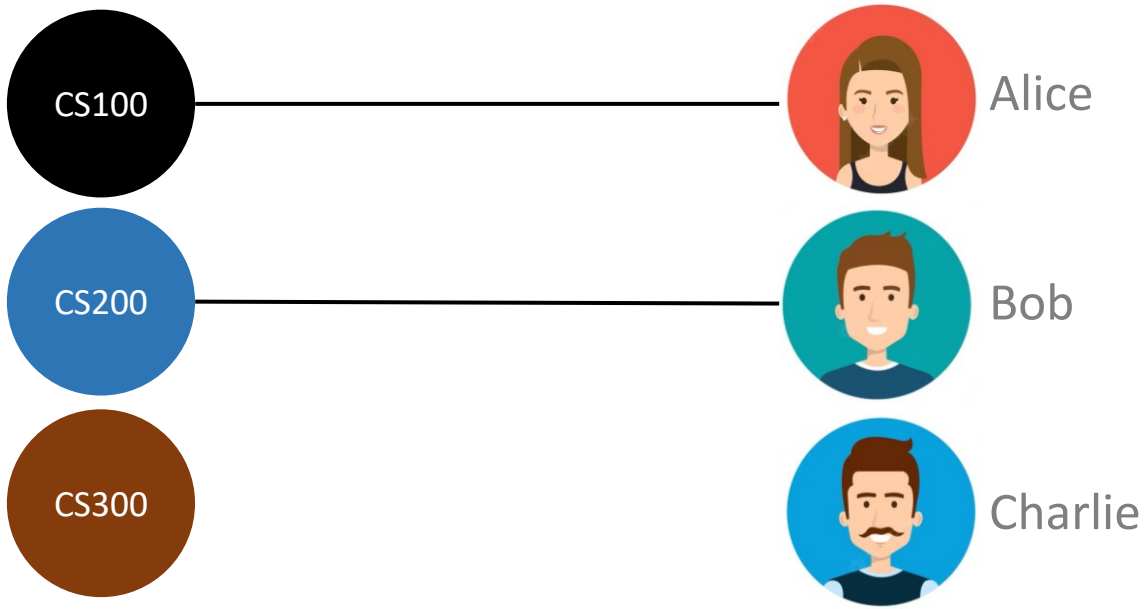
(CS100 – Alice) are unstable pair

Unstable pair – How to resolve?

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

M = {(CS100 – Alice), (CS200 – Bob)}

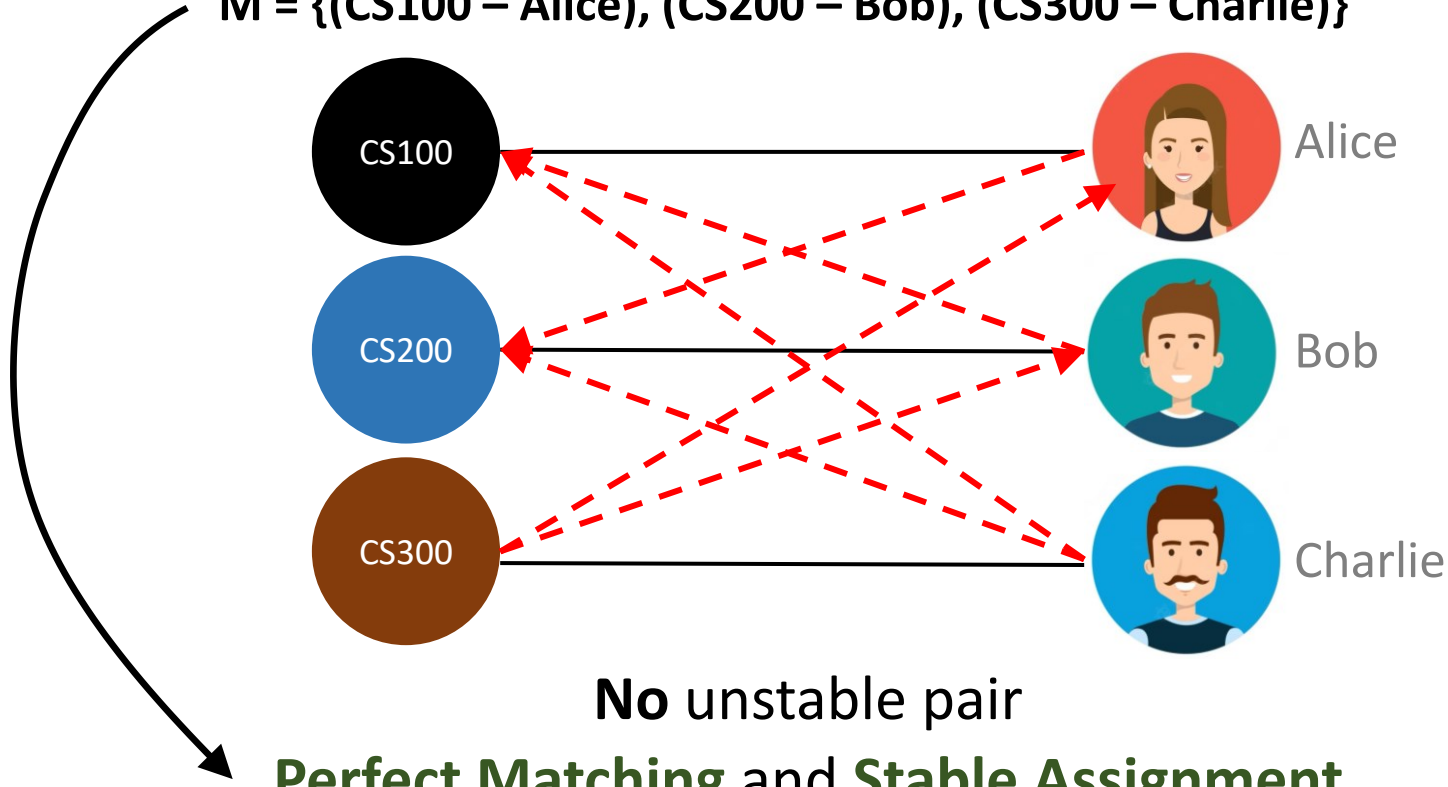


Unstable pair – How to resolve?

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

M = {(CS100 – Alice), (CS200 – Bob), (CS300 – Charlie)}



No unstable pair

Perfect Matching and **Stable Assignment**

Stable matching – Live Poll 1

Which pair(s) is unstable in the matching M = {(CS100 – Alice), (CS200 – Charlie), (CS300 – Bob)}

A. (CS100 - Bob)

B. (CS200 – Alice)

C. (CS200 – Bob)

D. None of the above

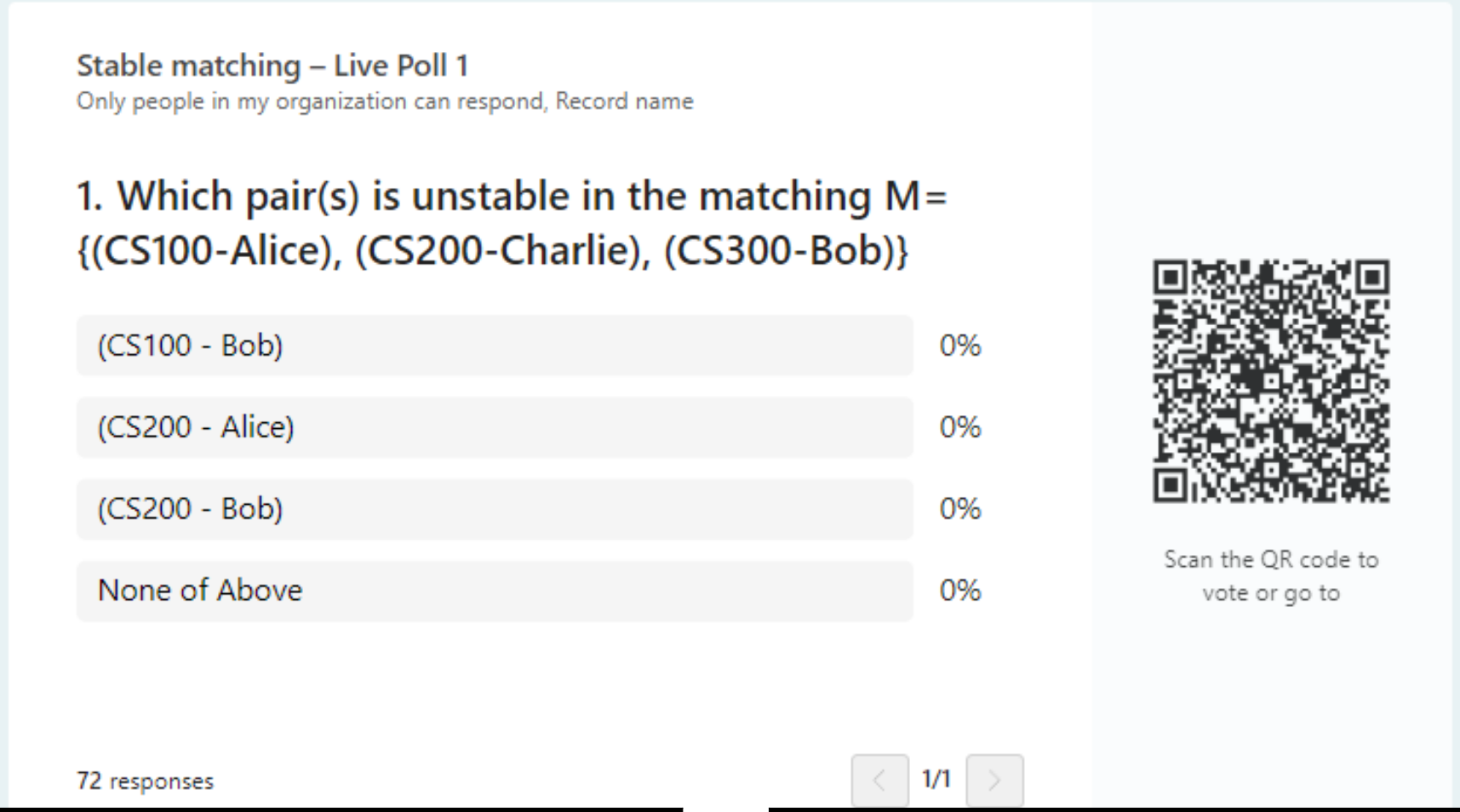
Scan the QR code to vote or go to https://forms.office.com/r/mwS1SuBX0B

	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list



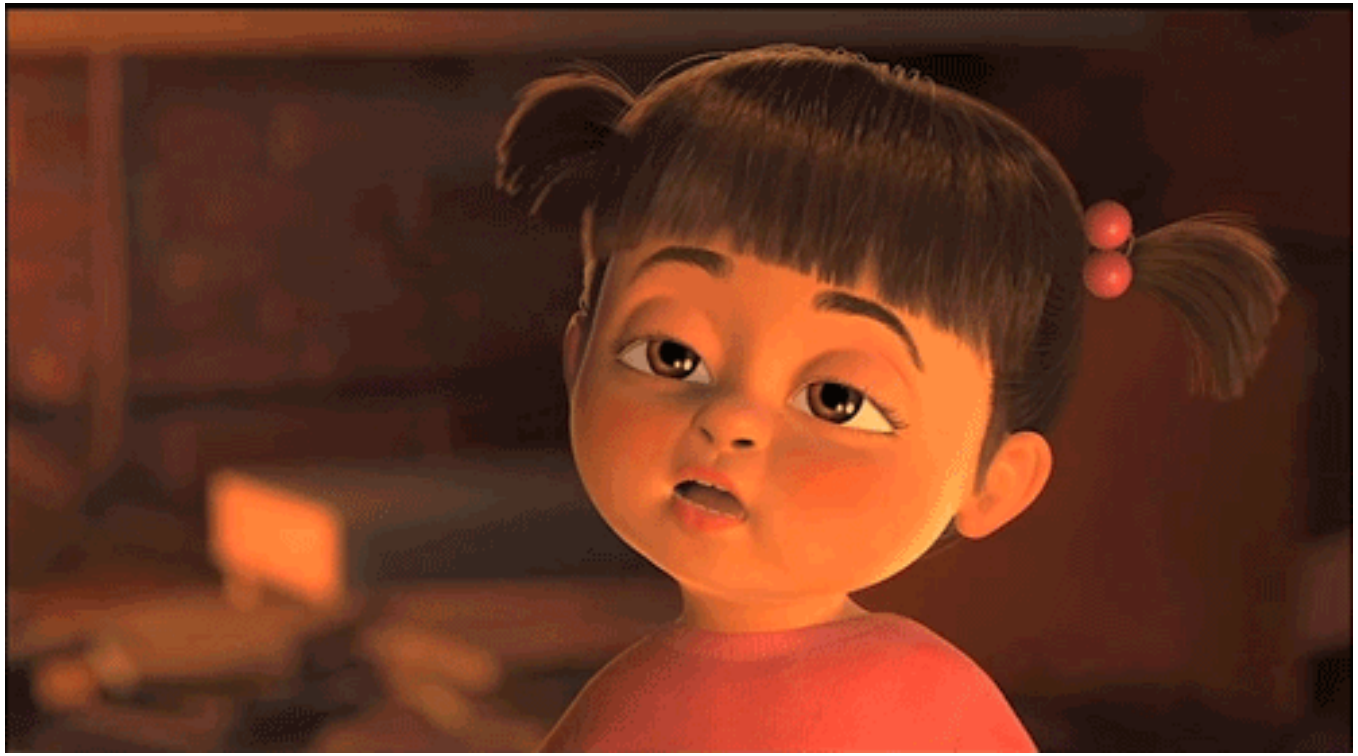
	1st	2nd	3rd
CS 100	Alice	Bob	Charlie
CS 200	Bob	Alice	Charlie
CS 300	Alice	Bob	Charlie

Course instructors' preference list

	1st	2nd	3rd
Alice	CS 200	CS 100	CS 300
Bob	CS 100	CS 200	CS 300
Charlie	CS 100	CS 200	CS 300

TA applicants' preference list

Thanks a lot



If you are taking a Nap, **wake up**.....Lecture Over