

How to get a job in CS

Putting your Software Construction skills to work

CS-214 - 16 Dec 2024

Clément Pit-Claudel

This lecture is going to be HIRE 🔥
— Anonymous CS214 student

Quick announcements

Congrat on webapps!

Checkoffs Tue and Fri: [check!](#)

Exam format

Exercises + mini-labs

Purposefully too long!

Final exam dry-run

After Wed lecture

Only chance to test setup!

One exercise + one mini-lab

Detailed poll

will be out on Wednesday

⇒ must fill to access dry run ⇐

EPFL poll

Out in January

This week:

Getting a CS job

Learning objectives:

- 1. Describe the job search and application process**
- 2. Get ahead on building your profile and portfolio**

- Motivation: Why now?
- Building a profile
 - Networking
 - Portfolio
- Making opportunities
 - Activating your network
 - Reading job ads
- Applying
 - CV & top CV sins
 - Cover letters
 - Recommendations
- Passing interviews

Why do I have to worry about this now?

— *CS214 student who would prefer to talk about futures*

Reasons to plan ahead

- Looking for internships before your MS
- Applying for AE positions or semester projects
- Doing your MS abroad
- Founding a startup
- Training takes a while
- Building a profile and a network also takes a long time

The job search timeline

0. Build skills, a profile, and a network

⇒ From now to the end of your career

1. Search for opportunities

⇒ A few months

2. Apply and get invited to interview

⇒ A few weeks; aim for 15-30% hit rate

3. Interview and get an offer

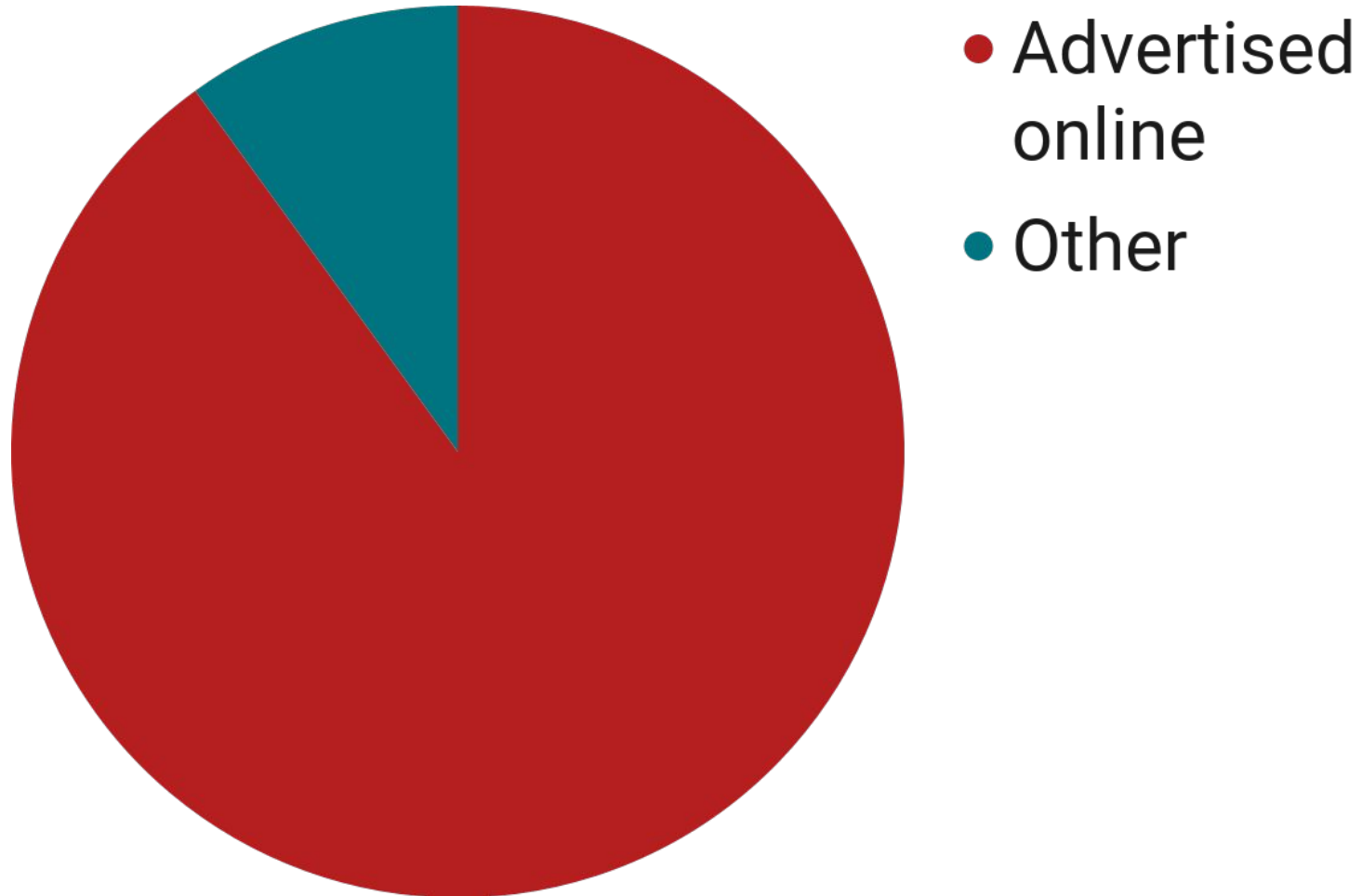
⇒ A few days to a week of interviews

⇒ 1-2 weeks to get an offer; may ghost if negative

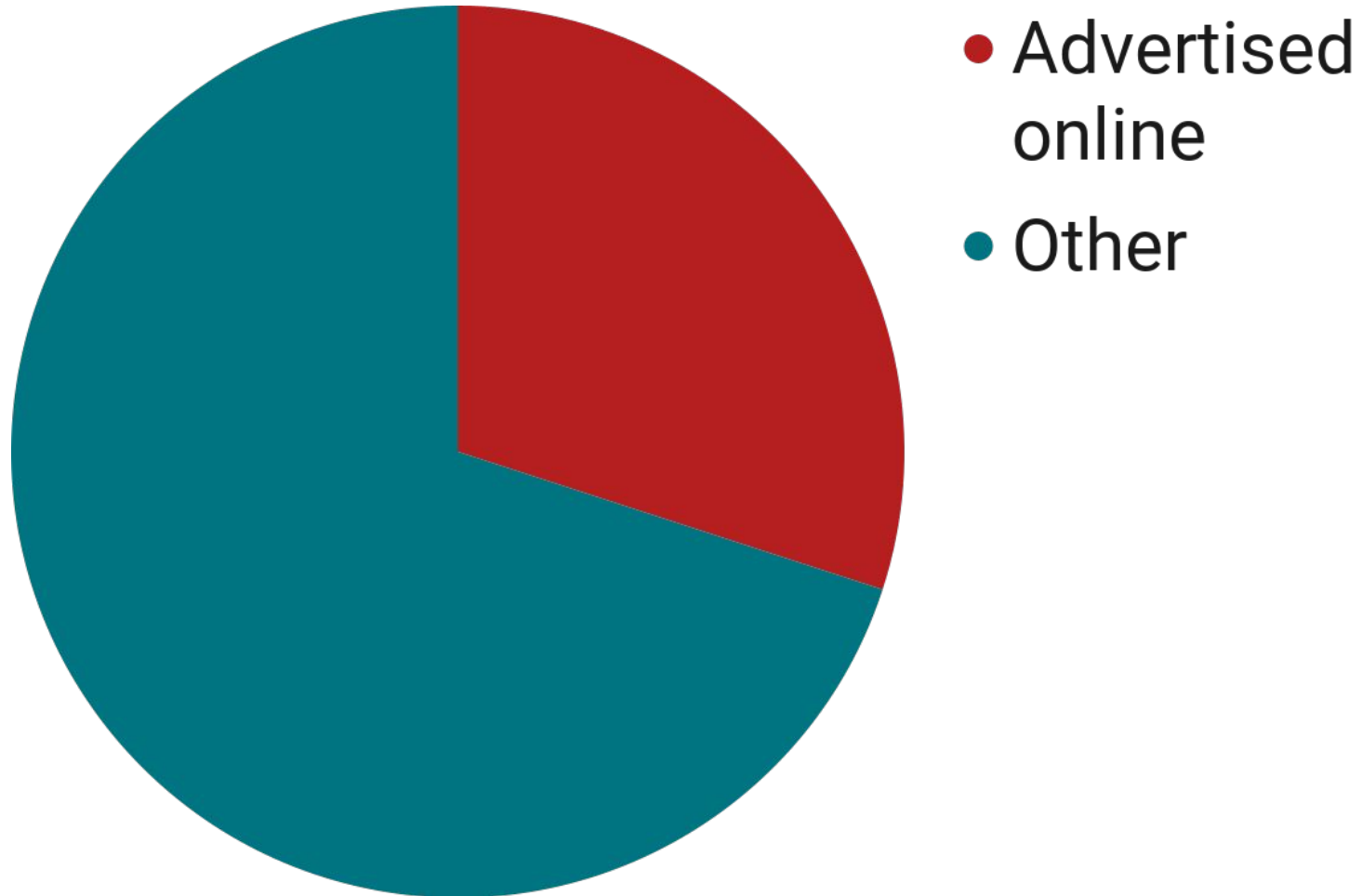
Part 1

Finding and making opportunities

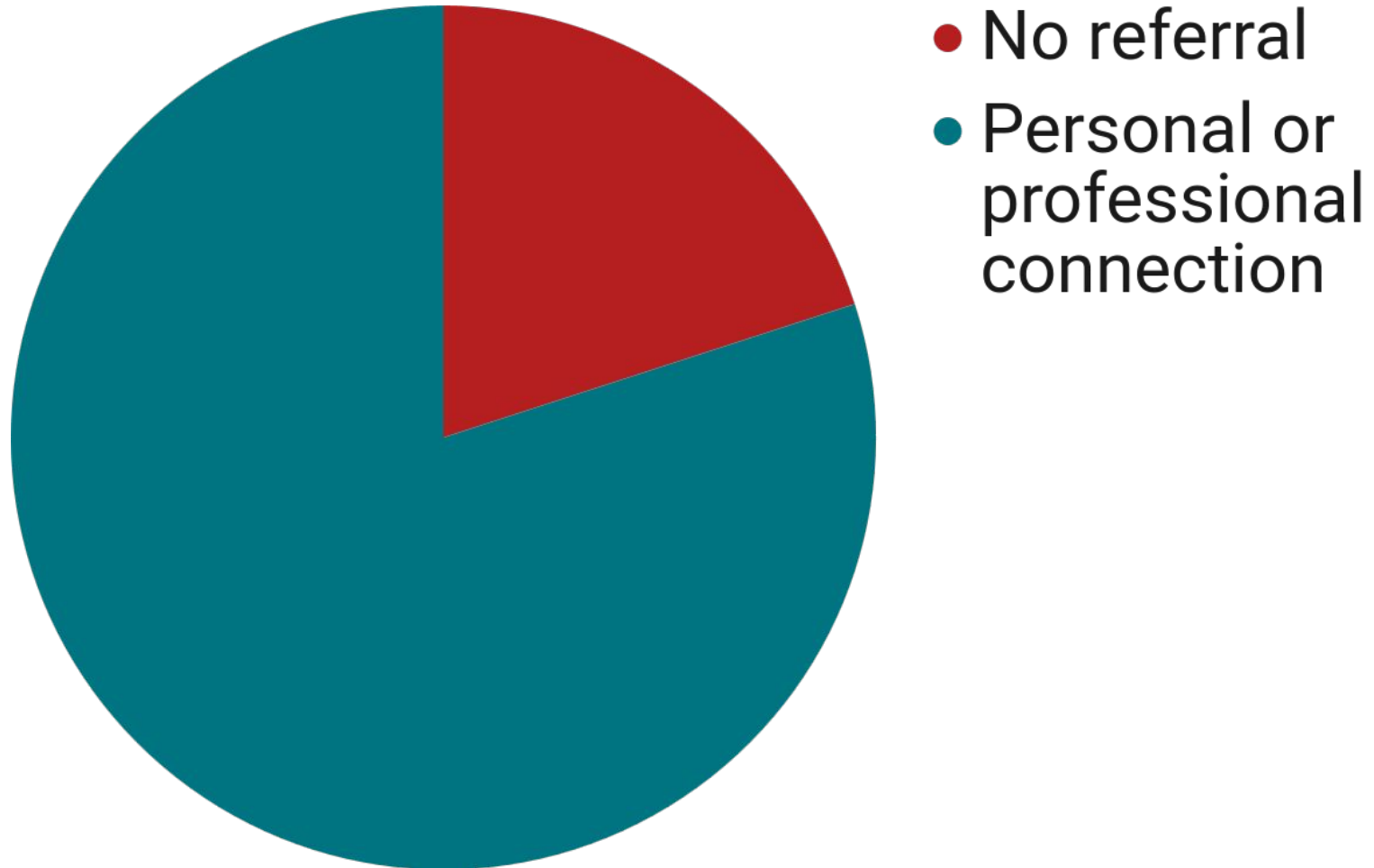
The typical vision of the job market



The reality of the job market



How positions are actually filled



Do referrals matter? Why?

- Referrals matter a lot
 - Companies are overwhelmed with applications
 - Interviews are very costly
 - CVs can be fabricated, candidates lie
- You need a **meaningful** referral
 - Putting your name into a random system:
Pointless
 - Connecting you to a team looking for your profile:
Priceless

Making opportunities

1. Be visible (see next section)
Keep a public and active profile
2. Look for open positions, even only vaguely related.
Job posting websites, EPFL internship site
3. Look for relevant companies (skills-needs alignment)
Don't waste time applying to irrelevant jobs!
4. Put the word out! Talk to friends, colleagues, relations
Ask for tips, recommendations, experiences
5. Reach out to contacts — leverage your network!
Caveat: Power of network $\propto 1 / \text{rarity}$

Reading job ads

- Check overall relevance / skills
- Don't obsess over specific details



Sebastián Ramírez
@tiangolo

Follow



I saw a job post the other day. 

It required 4+ years of experience in FastAPI. 

I couldn't apply as I only have 1.5+ years of experience since I created that thing. 



The Register®



SOFTWARE

192 

**IBM job ad calls for
12 years'
experience with
Kubernetes – which
is six years old**



DO NOT SPAM



- Do not send mass emails
- Do not write to 15 people at once
- Do not resend an email 4 times
- Do not sound desperate



APPLY SMART



- **Do not apply to 50 job postings**

Exercise

Map your network

Part 2

Applying

Building up your CV

- One page
- Clearly structured
- One-sentence objective / intro
- Projects, portfolio should have links
- Important info should stand out
- *You* should stand out
- Good form matters
- Content matters much more

An example

Good? Bad?

Devops

Implemented streamlined pipeline based on state-of-the-art open source libraries to produce a clean, robust, scalable system with fast response times. Deployed system and ensured reliability and user satisfaction.

It's bad. The following is much better:

Foo Corp (Subsidiary of Blah), Junior DevOps engineer

Replaced a bespoke sales-monitoring system with a grafana-based dashboard, reducing maintenance costs by 0.3 FTEs and shrinking report-generation time from 4 hours to 2 minutes.

CV sins: A checklist

- Technobabble
 - Be understandable
- Vague terms, fluff
 - Be specific, have numbers
- Unclear responsibilities
 - Say what your role you had
- Unclear contributions
 - Say what you did
- Missing names / institutions
 - Highlight relevant places
- Missing keywords
 - Don't get filtered out
- Too short
 - Have relevant experience
- Too long, pointless padding
 - Stop boasting, stay relevant
- Method over results
 - Say what, not just how
- Not following country norms
 - No photo in most countries!
- Bad fonts, layout, grammar
 - Look good
- Lies and exaggerations
 - You'll get caught. It'll suck.

Exercise

Write a CV bullet point

Cover letters

- One page
- Have a story about yourself
 - How you got where you are
 - Why you want to be there next ← Customize this part
- Sins
 - Pointless/generic blather (Modern version: ChatGPT)
 - Vacuous / unsupported statements
 - “I am highly motivated”
 - “I am convinced your company is a great match”

Don't bullshit: 2023 version



You

I'm applying for a paid position as an undergraduate teaching assistant in a Software Construction course, please help me write a cover letter email to the teacher to get the job.



ChatGPT

Subject: Application for Undergraduate Teaching Assistant Position in Software Construction

Dear Professor [Last Name],

I hope this email finds you well. My name is [Your Name], and I am writing to **express my enthusiastic interest** in the Undergraduate Teaching Assistant position for the Software Construction course. I am currently [mention your academic year] **majoring in [Your Major]** at [Your University], and **I am confident that my academic background, passion for software development, and strong interpersonal skills make me an excellent candidate for this role.**

Throughout my coursework, I have gained a solid foundation in software development principles, including [mention specific skills or programming languages relevant to the course]. I am particularly drawn to the Software Construction course **because of its emphasis on practical application and real-world problem-solving**, which aligns perfectly with my **academic and professional aspirations. I am eager to contribute to the learning experience of my peers by sharing my knowledge and assisting them in mastering the key concepts covered in the course.**

My previous experiences, including [mention any relevant experiences such as internships, projects, or coursework], **have equipped me with effective communication and collaboration skills.** These skills, combined with my **ability to break down complex concepts into understandable components**, will enable me to support students in **comprehending challenging topics** and completing their assignments successfully.

As an aspiring educator, I am excited about the opportunity to work closely with you and the students to create a positive and engaging learning environment. **I am confident** that my dedication, technical expertise, and commitment to fostering a collaborative atmosphere will contribute to the success of the Software Construction course.

Thank you for considering my application. **I would be thrilled to further discuss how my skills and experiences align with the goals of the course during an interview.** I am available at your convenience and can be reached via email at [Your Email Address] or **by phone at [Your Phone Number].**

I appreciate your time and consideration and look forward to the possibility of contributing to the success of the Software Construction course.

Sincerely,

[Your Full Name]

[Your University ID/Class]

[Your Contact Information]

Don't bullshit: 2024 version



You

I'm applying for a paid position as an undergraduate teaching assistant in a Software Construction course, please help me write a cover letter email to the teacher to get the job.



ChatGPT

Subject: Application for Undergraduate Teaching Assistant Position in Software Construction

Dear Professor [Last Name],

I hope this email finds you well. My name is [Your Name], and I am writing to **express my enthusiastic interest** in the Undergraduate Teaching Assistant position for the Software Construction course. I am currently [mention your academic year] **majoring in [Your Major]** at [Your University], and **I am confident that my academic background, passion for software development, and strong interpersonal skills make me an excellent candidate for this role.**

Throughout my coursework, I have gained a solid foundation in software development principles, including [mention specific skills or programming languages relevant to the course]. I am particularly drawn to the Software Construction course **because of its emphasis on practical application and real-world problem-solving,** which aligns perfectly with my **academic and professional aspirations.** **I am eager to contribute to the learning experience of my peers by sharing my knowledge and assisting them in mastering the key concepts covered in the course.**

My previous experiences, including [mention any relevant experiences such as internships, projects, or coursework], **have equipped me with effective communication and collaboration skills.** These skills, combined with my **ability to break down complex concepts into understandable components,** will enable me to support students in **comprehending challenging topics** and completing their assignments successfully.

As an aspiring educator, I am excited about the opportunity to work closely with you and the students to create a positive and engaging learning environment. **I am confident** that my dedication, technical expertise, and commitment to fostering a collaborative atmosphere will contribute to the success of the Software Construction course.

Thank you for considering my application. **I would be thrilled to further discuss how my skills and experiences align with the goals of the course during an interview.** I am available at your convenience and can be reached via email at [Your Email Address] or **by phone at [Your Phone Number].**

I appreciate your time and consideration and look forward to the possibility of contributing to the success of the Software Construction course.

Sincerely,

[Your Full Name]

[Your University ID/Class]

[Your Contact Information]

Getting recommendation letters

- Chose relevant people
- Make sure we know your name
- Give us something to write about
- Ask whether we can provide a *great* recommendation letter

Part 0

Building your profile and your network

Your profile: A roadmap for the next 3.5 years

Show independence and initiative, stand out, **have fun**. Some ideas:

- Free software contributions
Contributing is harder than building from scratch. Many projects have onboarding guides.
- Personal projects, hackathons, junior enterprise
Most students stick with just coursework — **Webapp fits!**
- Semester projects in labs
A must if you want to do R&D or a startup
- Paid student assistantships
Great for your network
- S4S, Fréquence banane, IT for a club
Springboard to get an assistantship
- Volunteer / nonprofit work
Make it relevant
- Language training
English, then French, then German
- Course projects and assignments
Only if they are good

Your network: A roadmap for the next 3.5 years

At school

- Attend events
Talks, career fairs, hackathons, conferences
- Find group hobbies
Hiking, gaming, music, anything
- Have a study group
Bonus: it helps with grades
- Get involved in mentoring
Seek it and give it: you may eventually depend on your mentees
- Get involved in committees
Serving your community always pays off



Networking is mostly one-on-one.
Introverts can be great at networking.

Your network: A roadmap for the next 3.5 years

Online

- Have a LinkedIn
That's where recruiters are
- Keep in touch with friends
From high school, clubs, summer camps, military service, ...
- Join mailing lists, discord, IRC
But be careful with contributions under your name. Stay professional.
- Publish tools and code
Not just for your portfolio: also helps meet people
- Blog, follow social media
Get a literal pressbook

Exercise during the break

Talk to three people you vaguely recognize but don't know

Part 3

Interviewing

Common parts of an interview structure

Most interviews are hybrid: online + in person.

1. Phone screen

Basic questions to check if you're a reasonable fit: background, algorithms

2. Behavioral interview

Questions on past experience and hypothetical situations

3. Engineering interview

Estimation, system design, tech quiz

4. Coding interview

Algorithms on paper or coding

5. HR interview

Typically short, towards end of circuit

Some examples from Switzerland (company redacted)

Each company does it a bit differently:

- Online screen
+ 3 x 45 minutes of behavioral, engineering, coding
- Phone screen
+ 4 x 30 minutes of coding and engineering
- Phone screen (rapid-fire)
+ 1h HR
+ 3h { engineering + code review }

General interview tips

- Be engaged and excited

Know why you want to be there

Have a pitch — interviewer may not have read your CV

- Make a good impression

Be on time

Dress adequately — in many cases this means no suit / tie

- Come in informed

Read up on the company — culture, principles, practices... and interview process!

Don't ask uncomfortable / illegal questions

- Don't waste opportunities

Pick your questions carefully — ask strategic questions

- Don't put yourself in a bad situation

Don't drink (too much) — get sparkling water

Don't ask for a smoke break

Don't eat dumb food — no lobster please

Tell them ahead of time if you need accommodations

Part 3.1

Behavioral interviews

Behavioral interviews — **Examples**

- Tell me about a missed deadline and how you approached it.
- Have you had to deal with conflict with work colleagues? If so, how did you resolve it?
- What's the worst mistake you've made at work?
- What is your top-one achievement in your previous job?
- What's the most interesting bug you've ever fixed, and how did you approach it?

Behavioral interviews — **Technique: STAR**

Tell a story from your own experience

Situation:

Lay the groundwork for the story, describe an issue, give details

Task:

Describe what your responsibility was in that moment

Action:

Describe the action you took to move forward

Result:

Explain what results your action led to

Also applicable to CVs!

Behavioral interviews — **Tips**

- **Stand out**

Have interesting stories — collect them over time!

Don't tell us about missed homework deadlines — please.

- **Be memorable**

Chose a diverse set of examples. Don't use the same job for every question.

- **Be positive, enthusiastic, and humble**

Don't complain or mock the questions!

Don't criticize your previous colleagues, workplace, etc.

- **Be concrete**

Give numbers, facts, measurable outcomes

Behavioral interviews — **How to prepare**

- **Collect stories**

Write down small summaries of interesting situations that you encounter.

- **Seek responsibility**

Get involved at school and work.

- **Practice the STAR method**

Can also be used on your CV

- **Train with friends**

Use the questions given above

- **Learn to think on your feet & react to unexpected situations**

Do improv

Join a D&D group?

Exercise: STAR

- **What is the hardest tech problem that you've had to deal with?**
- **Have you ever had to resolve a disagreement about a technical topic between two coworkers?**

Part 3.2

Engineering interviews

Conceptual, Quick-fire, System-design

Conceptual engineering interviews

Intended to get you to think — precise answer irrelevant
Falling out of favor

- How many cow farmers are there in Appenzell?
- What has more calories? A smartphone battery or a cookie?
- How much would the Matterhorn weight if it were made entirely of Toblerone candy?
- What does it mean if the probability of rain tomorrow is 60%?

Break the problem down, make assumptions and guesses, explain your thinking, use back-of-the-envelope calculations and fermi estimates.

Quick-fire engineering interviews — Examples

Intended to gauge your scientific and engineering fundamentals

- How do you insert an element in a linked list?
- How do you check if a linked list has a cycle?
- What is a promise and what do I use it for?
- Why is it unsafe for generic arrays to be covariant?
- What is a type class and why would I prefer that over an interface?
- What is the point of unit tests if you already have integration tests?
- What is more efficient, a tail-recursive function or a loop?
- Why would you use exceptions over the `Result` monad?
- What is the problem with rebasing after pushing to upstream?
- What is the complexity of quicksort when using linked lists?
- What is the difference between `# :::` and `++` on lazy lists?

Quick-fire engineering interviews — **Technique**

- Don't make things up
It's OK to say you don't know.
Offer something related if you can.
- Ask clarification as needed
Typically should be needed often.
- Prepare ahead of time
Review CS-214 debriefs, for example.

System design engineering interviews — Examples

Intended to gauge your real-world experience / streetwise

- Design a task-list app based on given framework (**webapp!**)
 - How many clients would this support?
 - How would you stress-test this?
 - What is missing to make it realistic / robust?
 - How would you distribute this across multiple servers to scale?
- Analyze the safety / longevity / maintainability of this particular design
- Design the architecture for a distributed key-value store with (some specific parameters)

System design engineering interviews — **More**

Sometimes about your previous experience instead:

- Have you ever developed a webapp?
 - Which framework did you use, if any?
 - How did you architect it (draw diagram)
 - How did you come up with this design?
 - How did you test it?
 - How long did it take to develop?
 - What bugs did you have and how could you have prevented them?

System design engineering interviews — Technique

- **Communicate throughout**
Explain what you're doing at all times.
- **Ask lots of questions**
Make sure you're designing for the customer.
But don't fish for answers.
- **Restate the problem as the interview evolves**
Confirm that you're on the same page.
- **Make guesses / assumptions and check them**
Communicate with your interviewer.
- **Offer alternatives**
Mentioning an option even if you don't explore it is still valuable.
Clarify trade-offs and design choices.
- **Draw and write**
Much easier on a whiteboard — use it!

System design and quick-fire — **How to prepare**

- **Be reflective**

Think back after every grade, lab, completed project.

Don't settle for superficial understanding.

Ask for feedback and give feedback.

- **Learn from others**

Never be the smartest person in the room.

Get involved in projects with experts.

Read blogs.

Program in pairs.

- **Learn from the pros**

Read problem solutions, lab scaffolds, source code.

Read and criticize Ed, StackOverflow questions.

- **Train yourself at answering questions**

Contribute on Ed, get involved in education

Make money from it: private lessons for gymnase students, assistantships

Exercise (in exercise session)

Answer the quick-fire questions listed above

Part 3.3

Coding interviews

Coding interviews — Intro

- Typically 20 minutes to 1 hour
- Free or not to pick language
- Typically one of
 - Algorithms design
 - Programming puzzle
 - Code review (performance, correctness)
- Almost never in an IDE / executed

Examples: Almost all of our on-computer exercises + lots online

Coding interviews — **How to prepare**

- Pay attention in BA4 algorithms class

Know the classics

- Use interview websites, books

Practice, practice, practice

- Participate in programming challenges

E.g. ICPC, advent of code

- Be reflective

Give and seek feedback

- Sign up to be an assistant for a class

Best way to get unexpected questions and have to think fast

- Network!

Set up a question bank

Use the Google drive!

Coding interviews — Tips

- Communicate, ask questions, speak while coding

Describe your thinking.

Speak while coding..

- Take the time to write correct code

Don't jump right in!

Remember tests, docs, and specs

- Be comfortable with debugging

You'll need to think fast when you're told your code is wrong

- Get enough sleep

Matters enormously

- Don't cheat, don't lie

Reputations travel fast

Some useful resources

- Books

- McDowell, *Cracking the coding interview*
- Bentley, *Programming pearls*
- Okasaki, *Purely functional data structures*

- Online resources

- [ETHZ CV guide](#)
- [The Workplace Stack Exchange](#)
- [Advent of code](#) ([Scala solutions](#)), [Project Euler](#)
- [ICPC](#)

- Parting words

- Be generous to your peers
- Help each other

Exercise (in exercise session)

Attempt a few interview puzzles

Demo

A coding interview

Demo disclaimers

- Max is a personal friend (See? Networking!)
- Not here on behalf of his job
- The interview problem was provided by Pr. Kunčák
I have not studied it, but I know roughly what it is about.

Thank you!