SYSTEM DESIGN

Part - 1

(INTRODUCTION)

Introduction to System Design

You may be reading this guide because you recently failed a system design interview. Or you watched a YouTube video that made system design seem like an overwhelming topic you'll never fully grasp. Or perhaps you have years of experience working in the field, yet you struggle to demonstrate your technical prowess in the brief span of a system design interview.

Do not panic!

Professional experience with distributed systems isn't needed to pass system design interviews. And even if you do have that experience, keep in mind that many talented distributed systems engineers still struggle with the system design interview format. How you perform in an interview is not a measure of your worth as a software engineer—it is a measure of your ability to do system design interviews. The two are related but not equal; being a good programmer has a surprisingly small role in passing interviews.

Anecdote

One of our experts was asked: “As an experienced engineer without any scalable systems experience, how can I go into something like a FAANG system design interview when I have never designed those systems before?” Here’s the expert’s reply:

"I worked at Facebook for five and a half years. I learned more about system design from reading the internal interviewing wiki than I ever got from working at Facebook. They've got all kinds of distributed systems knowledge there, and it’s concentrated in a handful of infrastructure teams who build really great libraries and really great backend systems. That means that the rest of us never have to think about distributed systems. We get to say, 'I'm gonna make a new data type, and if I dump a billion records in it tomorrow, it doesn't matter. The systems folks have my back.' I worked at Facebook for a really long time, but I learned almost nothing about designing systems from experience."

Remember

You can pass system design interviews even if you’ve never designed distributed systems before. If you have copied files between machines with drag-and-drop, you are halfway there. If you implemented clients or servers or have opened network connections, you’ve got this. This guide will teach you the most important 20% of information that will appear 80% of the time in system design interviews. By the end of this guide you won’t be an expert, but you’ll be well on your way to being a better engineer and a much better interview candidate.

The difference between engineering problems and design problems

In this excerpt taken from Design Your Life, written by two Stanford professors and engineers, you’ll get a better understanding of how different problems require

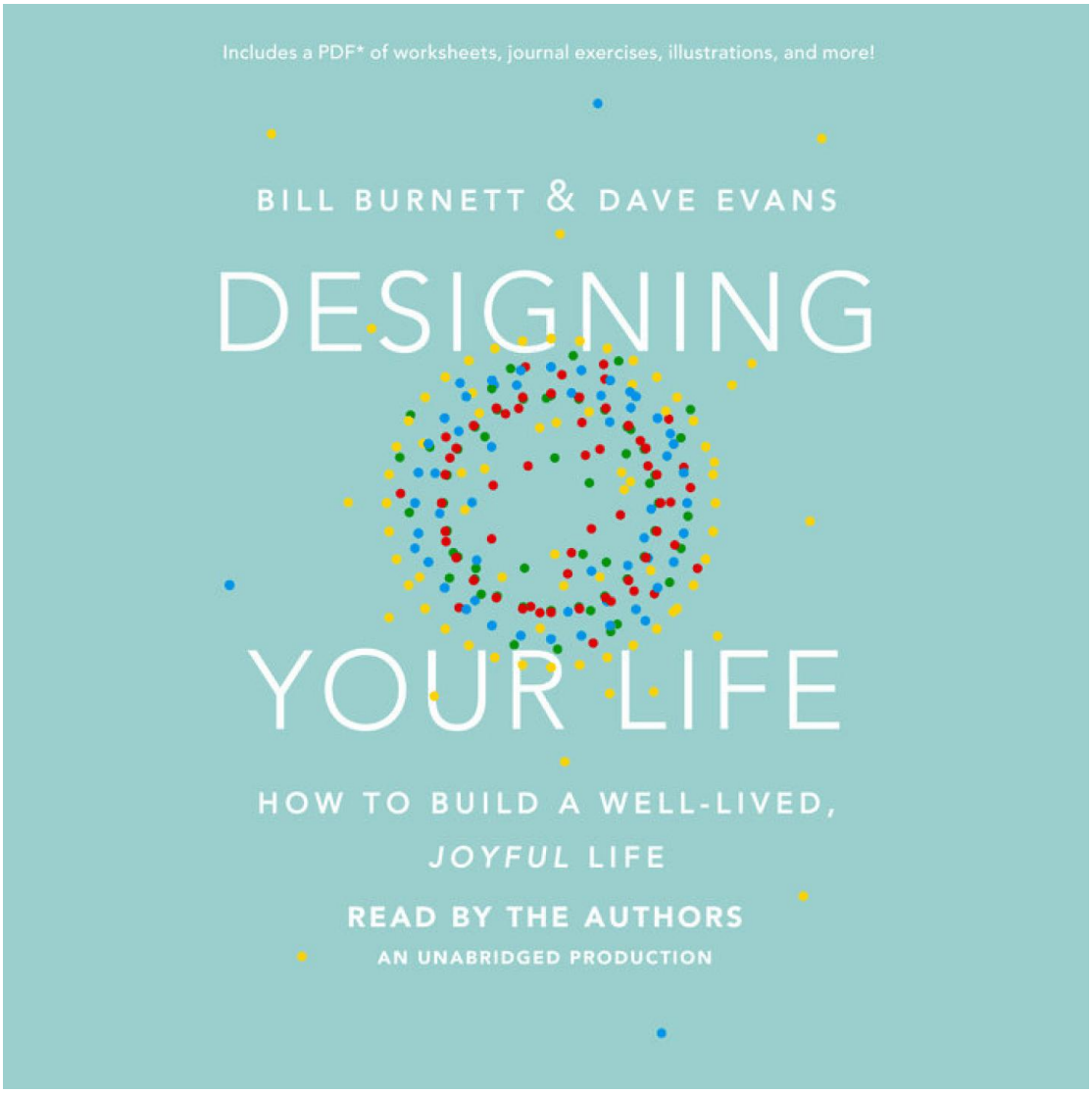
different approaches. We bolded to emphasize the parts that are most important to note.

“There’s a difference between design problems and engineering problems... **Engineering is a good approach to solving a problem when you can get a great deal of data and you’re sure there is one best solution.** Bill [one of the authors] worked on the problem of engineering the hinges on Apple’s first laptops, and the solution he and his team came up with made those laptops some of the most reliable on the market. The solution required many prototypes and lots and lots of testing, similar to the design process, but the goal of creating hinges that would last five years (or opening and closing ten thousand times) was fixed, and his team tested many different mechanical solutions until they met their goal. **Once this goal was met, the solution could be reproduced millions of times. It was a good engineering problem.**

“Compare this with the problem of designing the first laptop that had a ‘built in mouse’. Because Apple’s computers relied on the mouse to do almost everything, building a laptop that required you to be wired up to a regular mouse was unacceptable. **This was a design problem. There was no precedent to design toward, there was no fixed or predetermined outcome;** there were plenty of ideas floating around the lab, and a number of different designs were tested, but nothing was working. Then along came an engineer named Jon Krakower. Jon had been tinkering around with miniaturized trackballs, and had the crazy idea to push the keyboard to the back of the unit, leaving just enough room to squeeze in this tiny pointing device. This turned out to be the big breakthrough everyone had been looking for, and has been part of the signature look of Apple laptops ever since.

“When you have a desired outcome (a truly portable laptop computer) but no clear solution in sight, that’s when you brainstorm, try crazy stuff, improvise, and keep ‘building your way forward’ until you come up with something that works. You know it when you see it. A great design comes together in a way that can’t be solved

with equations and spreadsheets and data analysis. It has a look and feel all of its own - a beautiful aesthetic that speaks to you."



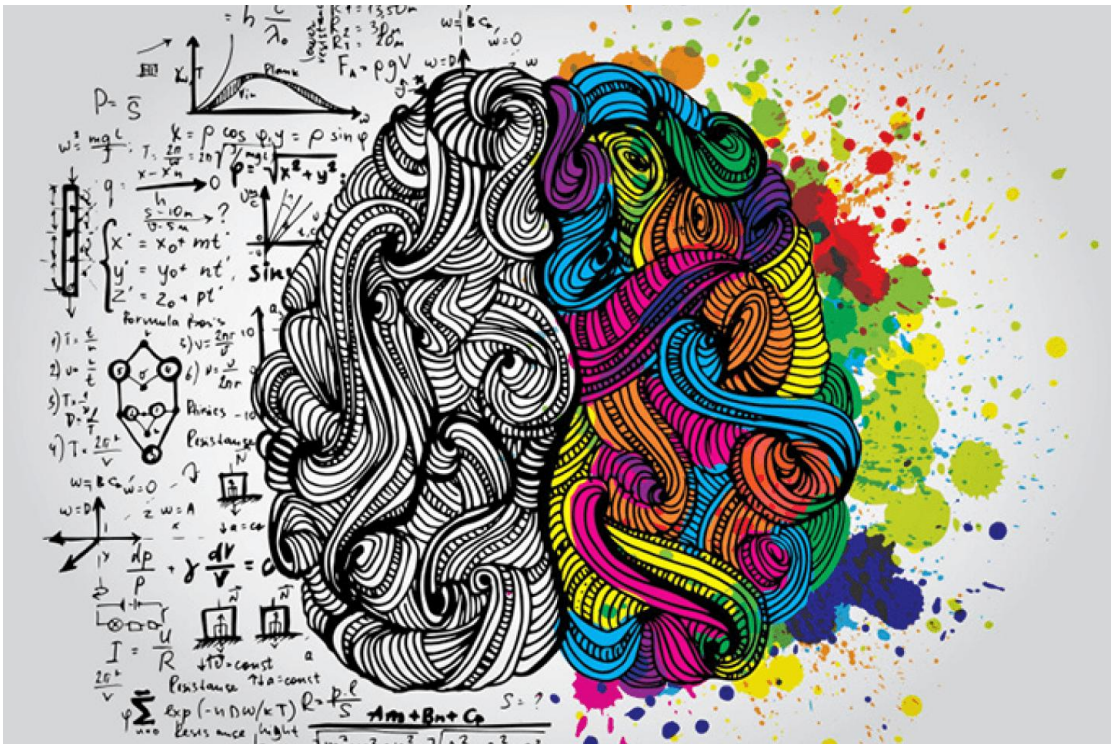
This is one reason engineers new to system design can bomb their first couple of system design interviews spectacularly: They approach a design problem as if it's an engineering problem. There is not a single "best" solution to a system design problem. There are no predetermined outcomes. The less code you write in a system design interview, the better.

To succeed in a system design interview, you want to collaborate with your interviewer, try crazy stuff, and try more crazy stuff until the design "feels right."

How approaching a system design interview is different than a coding interview

The difference between coding and system design is the difference between retrieving and creating.

Instead of finding (or “retrieving”) a solution, you are creating a solution. In this way, coding is akin to a science, while system design is more like an art.



Here’s another way to think about it. You aren’t solving a problem—you’re creating a map to help someone else find the solution. Instead of coloring inside some lines, you’ll need to draw the lines for someone else to color in. In a system design interview, there are no correct answers—though there are certainly incorrect ones—so there is nothing to solve. Instead, you’ll ask questions, make stuff, and explain how and why the stuff you made was reasonable.

Don’t think like a coder. Think like a Tech Lead.

During the interview, you’ll spend an hour playing the role of a Tech Lead, so just pretend that the interviewer is a junior engineer who will be implementing your design. Juniors will have lots of questions, and since you’re the Tech Lead, you want to welcome these questions.

Anecdote

“Pretend it’s 1999, so a lot of the stuff we have access to today simply doesn’t exist. You and a group of your schoolmates are in your garage, hoping to make something. You're the most senior one there. You will design it and your friends will code it up, and the thing is: the Minimum Viable Product has to be completed by tomorrow. So, there’s no time to prep and no need to worry about the intricacies of system architecture that you don’t know. Just answer this: How would you design this system so your friends could code it up today, right now? It doesn’t have to be pretty. It doesn’t have to be complicated. It doesn’t have to impress anyone. It just has to get done.”

What you do is important, but how you communicate is even more important.

The value of communication in coding interviews is controversial (source). But without communication in a system design interview, nothing happens. By necessity, it’s more conversational in nature than a coding interview and will have more back and forth between interviewer and candidate.

With that said, sometimes you’ll have an interviewer who is cold or not very collaborative. Dealing with these interviewers requires practice. The more senior you become, the more important it is to learn how to adjust your communication style to match your audience. We recommend completing mock interviews with a variety of interviewers to help you become a seasoned, fearless veteran of system design interviews.

There are no optimal solutions in system design interviews.

There’s no “right” way to design a system. If two experts designed the same system, you would see two different designs, beautiful and aesthetic in their own way and both as “correct” as the other (and with the accompanying justifications to support them).

Tip

In part 2, two experts will show you how they would each design the same system, providing you with a practical lesson on this topic.

Lean towards your strengths.

Anecdote

“If you want to be a great interview candidate you’ve gotta know… It’s your responsibility to leave breadcrumbs for the interviewer to go where you want them to go. That way you have them walk you down the road where you are at your best. And then the Jedi mind trick is to get them to think it was their idea to get you there.”

What it’s like to walk into a system design interview

When beginning an interview, try to imagine what the interviewer is looking for. What are their goals for the session? How can you help them achieve those goals in a way that persuades them that you’ll be a strong hire?

Put simply, the interviewer's goal is to find enough data to hire you. Given the limited time available to them, an interviewer has to try to get enough positive signal about your ability so they can justify giving you a “hire” rating. In one hour you have to show your interviewer that you understand the fundamentals of a system (end to end). You also should be able to name and explain (at least at a high level) each part of the system, describe the tradeoffs you make, and find a solution.

Remember

The best way to accomplish this is to imagine that you're explaining a design doc to a group of more junior engineers. They will ask you questions about your decisions and want to know what you're trying to solve. Anticipating these questions and your responses will set you up for success in the interview.

What your interviewer looks for, and what they don't

With this basic model in mind, let's consider the main elements that system design interviewers look for, and the elements that don't matter.

What your interviewer wants to see

1. a broad, base-level understanding of system design fundamentals.
2. back-and-forth about problem constraints and parameters.
3. well-reasoned, qualified decisions based on engineering trade-offs.
4. the unique direction your experience and decisions take them.
5. a holistic view of a system and its users.

What your interviewer is not looking for

1. deep expertise in the given problem domain.
2. assumptions about the prompt.
3. specific answers with ironclad certainty.
4. a predefined path from the beginning to end of the problem.
5. strictly technical considerations.

You do not need to display deep expertise in the given problem domain. Interviewers want to see that you have a broad, base-level understanding of system design fundamentals.

Your interviewer will expect you to have knowledge of a wide range of basic topics, but they won't expect you to be an expert in any of them. For instance, you should understand the difference between SQL and NoSQL databases, their broad performance characteristics, and the types of applications each might be useful for (which we'll teach you later in this guide). But you would not need to know how the internals of either type of database work at any kind of detailed level.

In spite of this, you still might be asked to design those internals! Keep in mind, though, that your answer doesn't need to be optimal or reflect real-world implementations. For example, if an interviewer asks you to design a database/SQL query engine, they're not trying to discern if you're familiar with the academic literature on query engines or discover how much time you've spent working on database internals.

Instead, they want to see how you would approach the problem based on what you do know, starting from first principles and collaborating with them. Your answer will probably not be anywhere near optimal, and that's OK! The interviewer will focus on the process, not the result.

Interviewers want to engage you in a back-and-forth conversation about problem constraints and parameters, so avoid making assumptions about the prompt.

Initial prompts to system design problems tend to be intentionally light on detail. Many candidates make a mistake by extrapolating details from the initial prompt and crafting a solution based on those assumptions.

For example, imagine that the interviewer instructs you to design a "photo sharing service" with some minimally defined capabilities. This may cause some candidates to imagine that they're rebuilding Instagram and start designing around the

assumption that all images will be relatively small, not examined closely, and that extensive compression to save storage and bandwidth is acceptable.

But the interviewer didn't tell you to rebuild Instagram, so you'll need to keep in mind that there are many different types of photo sharing services. The interviewer may have had in mind something like Imgur or Photobucket, sites that cater more to basic image hosting for the web. Or they could be thinking about something like Flickr or 500px, services built for photographers to show off their work in high resolution.

So how do you figure out what type of service the interviewer wants you to build? Ask them! A basic prompt leaves room for you to start a conversation with your interviewer about the system you're designing—what type of users does it serve, what type of traffic can it expect, what limits will it have? Demonstrating that you can think critically about the parameters of your service is the first step in any system design interview.

Interviewers are not looking for specific answers with ironclad certainty. They want to see well-reasoned, qualified decisions based on engineering trade-offs.

Be very careful any time you find yourself responding immediately to a prompt in a system design interview. Even aspects of your design that seem insignificant need at least cursory consideration. Let's use IDs as an example.

A candidate will often start a discussion of a data model with a statement like, "I'll use auto incrementing IDs," or "I'll use GUID here" as kind of a default approach to assigning IDs to data. In many applications, however, the type of ID you assign to your data has practical consequences.

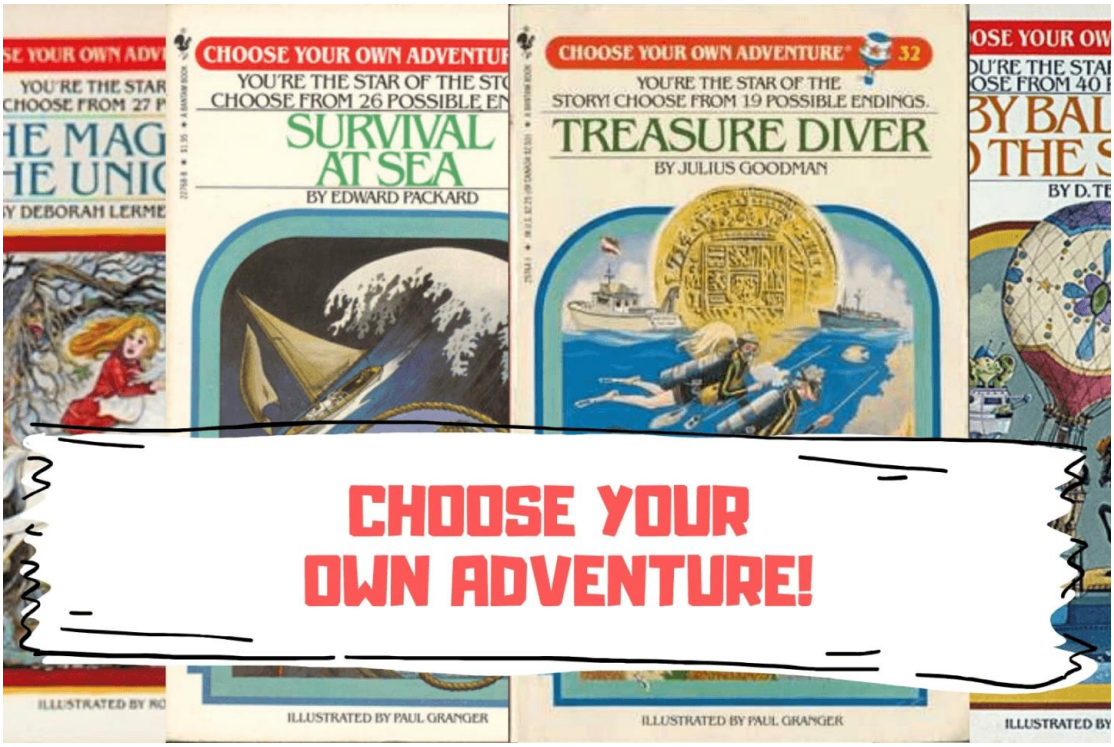
Is this ID going to be exposed to users? If so, how long does it need to be to avoid collisions? If we auto-increment it, are we worried about the visibility that will give third parties into our traffic patterns or the possibilities of users guessing the IDs to each others' data? If it's intended to be shared, is it convenient to type? If you print it

on a business card or a flier, does it contain characters that you could confuse for each other (e.g., “1” and “l”, “0” and “O”)?

You don't need to hold an inquiry for every minor detail, but always be sure to give some justification for the decisions you make and let your interviewer know how your decisions would change in different circumstances. System design problems don't have a single definitive answer, so interviewers just want to see that you can justify your answers.

Interviewers are not looking for a predefined path from the beginning to end of the problem. They want to see the unique direction your experience and decisions take them.

Coding problems usually have an expected path. Typically you'll begin with an obvious but inefficient solution, and then the interviewer will prompt you for a series of improvements. Those improvements lead you to increasingly efficient solutions until you finally arrive at the optimal implementation.



System design problems, on the other hand, resemble a Choose Your Own Adventure book rather than a linear novel. A complex system contains a multitude of sub-components, each one of which could serve as a design problem on its own. After you've sketched the overall layout of your system, an interviewer may decide to keep your focus on the big picture or dive into a deeper examination of one particular component.

The path your interview takes will be steered by your interviewer, but they're likely to take cues from the sub-problems in which you display interest or aptitude. In some cases they may explicitly ask you which part of the problem you'd prefer to focus on.

Even if you're not choosing directly, you can still influence an interview's direction. As you talk your way through a solution, it’s OK to specifically note the parts that you have experience in and explain when you're making educated guesses. Your interviewer won't expect you to know everything, but giving them a better idea of what you do know will help them steer the interview in ways that reveal your strengths and problem-solving ability.

Interviewers seek a holistic view of a system and its users.

When faced with a choice in a design interview, it's easy to focus on the technical details, but remember that computer systems serve human users, so you’ll want to anchor your technical decisions to the user experience they enable.

Suppose, for instance, that the image sharing service you're designing will require users to log in before uploading an image. In technical terms, you might want to avoid login to keep the database schema simpler, or you could introduce login to gather better metrics. An anonymous experience may be best for a public image-hosting site intended for quick turnaround and low interaction, while a logged-in experience offers the possibility of community features like commenting and sharing, personalized metrics, and the ability to restrict an upload to authorized

viewers. You may want to take either approach or even both, allowing a limited anonymous experience with extra features for logged-in users.

The important thing is to discuss the possible approaches and their consequences for the user experience with your interviewer before making a decision. You can never go wrong by making the end user the driving force in your design.

Green Flags, Red Flags, and other signposts

Think of red and green flags as signposts you can use to orient yourself in the interview. Green flags indicate that things are going well, that you're engaging with the interviewer and making a positive impression. Red flags warn you that you may be going astray and should try to get the interview back on track.

Red Flag #1: You believe that to pass a system design interview, you should just "play the game, keep talking, and make sure nobody explodes."

Following this quote's advice has steered many interviewees in the wrong direction. There is no game, and talking for the sake of talking is one way to hang yourself with the rope the interviewer gives you. Also, if the goal is to not explode, well, you're wasting your and your interviewer's time.

Green Flag #1: You communicate honestly about what you know and what you don't.

As we mentioned earlier, this guide will teach you the basic information that you'll be asked about in 80% of system design interviews. Although these are great odds, you still may encounter a scenario that's beyond your level of understanding. If this happens to you, don't worry! Just engage in an honest dialogue with your interviewer, explaining when you lack certain knowledge or have gaps in your understanding. When you do have a sense of how to proceed, but you're uncertain, you should communicate from first principles. Later in this guide, we will explain how to overcome that uncertainty and still score points with your interviewer.

Red Flag #2: You find yourself pushing against interviewer feedback.

Keep in mind that your interviewers use the same problems over and over again, and they frequently see candidates make the same mistakes. If they try to divert you from a course of action, it's likely because they've seen others flounder when using the same approach. You may be the one candidate in a hundred who finds a unique and better solution—we've had this happen before!—but carefully consider the odds before proceeding with a solution against the interviewer's advice.

With that said, there is an art to pushing back against your interviewer when the situation calls for it, and later in this guide we'll teach you how and when to employ this strategy .

Green Flag #2: The interview feels like collaboration between you and the interviewer.

When the interviewer offers feedback, you integrate it into your design. You ask probing questions and receive useful answers about the system you're designing, its users, and its traffic. Try to establish a tone as if you were working through a

problem with a coworker rather than proving yourself to an interviewer. In the real world, when you're assigned a project, you'll have to ask a variety of people several questions to ensure that you fully understand the problem before making decisions. That's what interviewers want to see.

Red Flag #3: You skip over questions and ignore interviewer prompts, trying to move the interview ahead without addressing their concerns.

It's OK to not know things—no one will have every answer—but it's better to admit that to your interviewer than to avoid the questions altogether. Your interviewer may be able to offer you a hint or help you reason about alternatives if they know you're struggling, but if you skip right ahead you'll miss the opportunity to provide them with any positive signal from that portion.

Green Flag #3: Your role determines who should drive the focus and pace of the interview.

If you're looking for a mid-level position or below, your interviewer should determine the direction and speed of the interview. Given an initial overview of your design, they may ask you for clarification on some aspects of it. They may ask you to produce a more detailed design for one or more components. And they may also change the requirements and ask how you could adapt your solution to accommodate this new view of the world. Wherever they take the interview, follow along and focus on the areas they direct you to.

If you're applying for a senior role (or above), it's a good sign if you direct more of the interview. In junior system design interviews, the interviewer expects to drive the interview, but as you reach senior levels the expectation shifts to the interviewee.

Anecdote from a seasoned interviewer

Being overly confident and talking too much might count against a mid-level candidate. Some interviewers (especially off-script ones) love giving candidates more rope to hang themselves with, and then they ask specific questions that focus on what the candidate struggles with.

If your goal is to maximize a mid-level offer, not improve your "average passing rate" (i.e., if you are comfortable sacrificing some senior-plus chances to increase your mid-level chances), then you might be better off consciously "giving control away" to your interviewer.

Simply put, at the above-senior level an awkward pause will be held against you—that's basically guaranteed. But at mid-level, most of your attempts to fill in an awkward pause may hurt you more than keeping silent.

Another way to think of it: when you are not leading the conversation, you signal that you're not really far above mid-level. (But if you are comfortable at mid-level, this is not a downside!)

The saying, 'Better to remain silent and be thought a fool than to speak out and remove all doubt' can be true for mid-level interviews but not for seniors or above-senior."

Red Flag #4: You leave long stretches (several minutes) of silence multiple times throughout the interview.

If you're struggling to provide an answer, give yourself a little bit of time to come up with something. If you're truly stuck, however, you should ask your interviewer for help. They can't tell that you're at an impasse unless you tell them, and you may waste valuable interview time while they debate whether it's been long enough to interrupt you.

Green Flag #4: You take time to collect your thoughts and refine solutions before offering them up out loud/on the board.

An interview doesn't need to be a continuous stream of consciousness, and it never hurts to sanity check your ideas before verbalizing them.

Tip

In Part 4 of this guide, we'll teach you how to get unstuck and exactly what to say when you're stuck.

A few more signposts

A common failure point occurs when candidates don't make decisions

Often, candidates will say things like: "we could use this type of DB, or this other, or that other, and these are some pros and cons…" and then they move on to another

component. It's a good practice to talk about benefits and tradeoffs, but then you have to make a decision. In the real world you have to make decisions—the same thing applies to the interview. If the interviewer challenges you with some questions, it's totally fine to change your mind and alter the component (if you think there are better choices).

Don't say

We could use this type of DB, or this other, or that other, and these are some pros and cons…

Do say

"We could use this type of DB, or this other, or that other, and these are some pros and cons… And based on all these tradeoffs, I'll use THAT type of DB."

Interviewers want to identify "impostors": people who just learned a few words and try to pass the interview.

Don't say things because you think you're supposed to say them. This often occurs when candidates name specific brands of technologies (e.g., "Kafka" or "Cassandra"). Not being familiar with specific databases or other components is fine. Be smart and don't say brand names just for the sake of saying them.

Don't say

I'm going to use Cassandra..." unless you are VERY familiar with that, because the next question will be: "Why Cassandra and not some_other_db?

Do say

I'm going to use a NoSQL db because of [insert brief rationale].

Don't say

I will use Kafka…" unless you're prepared to explain how Kafka works. Don't say "I will use Kafka" unless you are prepared to talk about other types of queues, because they may ask you: "Oh, Kafka, interesting choice. Why that instead of [some other queue]?

Do say

I will use a queue because of [insert brief rationale].

Remember

Say the generic name of the component, not the brand name unless you are very familiar with it. Don't say Kafka. Instead, say "a queue.

You finished Part 1! We hope you gained a basic understanding of the system design interview and learned some tips and tricks you can use to excel in it. As you continue reading Parts 2-4, these "glimmers" of understanding will become more and more the default system of your interviewing skills.

In Part 2 we'll teach you the 15 fundamental system design concepts. You'll also get to watch our long form video of two system design experts designing the same system. They're tasked with designing interviewing.io, which in this challenge is actually three systems in one: "Design google docs, a remote compiler, and a recording service… in 30 minutes or less."