Chapter 6. Leading at Scale

Written by Ben Collins-Sussman

Edited by Riona MacNamara

In Chapter 5, we talked about what it means to go from being an "individual contributor" to being an explicit leader of a team. It's a natural progression to go from leading one team to leading a set of related teams, and this chapter talks about how to be effective as you continue along the path of engineering leadership.

As your role evolves, all the best practices still apply. You're still a "servant leader"; you're just serving a larger group. That said, the scope of problems you're solving becomes larger and more abstract. You're gradually forced to become "higher level." That is, you're less and less able to get into the technical or engineering details of things, and you're being pushed to go "broad" rather than "deep." At every step, this process is frustrating: you mourn the loss of these details, and you come to realize that your prior engineering expertise is becoming less and less relevant to your job. Instead, your effectiveness depends more than ever on your *general* technical intuition and ability to galvanize engineers to move in good directions.

The process is often demoralizing—until one day you notice that you're actually having much more impact as a leader than you ever had as an individual contributor. It's a satisfying but bittersweet realization.

So, assuming that we understand the basics of leadership, what it does it take to scale yourself into a *really good* leader? That's what we talk about here, using what we call "the three Always of leadership": Always Be Deciding, Always Be Leaving, Always Be Scaling.

Always Be Deciding

Managing a team of teams means making ever more decisions at ever-higher levels. Your job becomes more about high-level strategy rather than how to solve any specific engineering task. At this level, most of the decisions you'll make are about finding the correct set of trade-offs.

The Parable of the Airplane

Lindsay Jones is a friend of ours who is a professional theatrical sound designer and composer. He spends his life flying around the United States, hopping from production to production, and he's full of crazy (and true) stories about air travel. Here's one of our favorite stories:

It's 6 a.m., we're all boarded on the plane and ready to go. The captain comes on the PA system and explains to us that, somehow, someone has overfilled the fuel tank by 10,000 gallons. Now, I've flown on planes for a long time, and I didn't know that such a thing was possible. I mean, if I overfill my car by a gallon, I'm gonna have gas all over my shoes, right?

Well, so anyway, the captain then says that we have two options: we can either wait for the truck to come suck the fuel back out of the plane, which is going to take over an hour, or twenty people have to get off the plane right now to even out the weight.

No one moves.

*Now, there's this guy across the aisle from me in first class, and he is absolutely livid. He reminds me of Frank Burns on M*A*S*H; he's just super indignant and sputtering everywhere, demanding to know who's responsible. It's an amazing showcase, it's like he's Margaret Dumont in the Marx Brothers movies.*

So, he grabs his wallet and pulls out this massive wad of cash! And he's like "I cannot be late for this meeting!! I will give $40 to any person who gets off this plane right now!"

Sure enough, people take him up on it. He gives out $40 to 20 people (which is $800 in cash, by the way!) and they all leave.

So, now we're all set and we head out to the runway, and the captain comes back on the PA again. The plane's computer has stopped working. No one knows why. Now we gotta get towed back to the gate.

Frank Burns is apoplectic. I mean, seriously, I thought he was gonna have a stroke. He's cursing and screaming. Everyone else is just looking at each other.

We get back to the gate and this guy is demanding another flight. They offer to book him on the 9:30, which is too late. He's like, "Isn't there another flight before 9:30?"

The gate agent is like, "Well, there was another flight at 8, but it's all full now. They're closing the doors now."

And he's like, "Full?! Whaddya mean it's full? There's not one open seat on that plane?!?!?!"

The gate agent is like, "No sir, that plane was wide open until 20 passengers showed up out of nowhere and took all the seats. They were the happiest passengers I've ever seen, they were laughing all the way down the jet bridge."

It was a very quiet ride on the 9:30 flight.

This story is, of course, about trade-offs. Although most of this book focuses on various technical trade-offs in engineering systems, it turns out that trade-offs also apply to human behaviors. As a leader, you need to make decisions about what your teams should do each week. Sometimes the trade-offs are obvious ("if we work on this project, it delays that other one…"); sometimes the trade-offs have unforeseeable consequences that can come back to bite you, as in the preceding story.

At the highest level, your job as a leader—either of a single team or a larger organization—is to guide people toward solving difficult, ambiguous problems. By *ambiguous*, we mean that the problem has no obvious solution and might even be unsolvable. Either way, the problem needs to be explored, navigated, and (hopefully) wrestled into a state in which it's under control. If writing code is analogous to chopping down trees, your job as a leader is to "see the forest through the trees" and find a workable path through that forest, directing engineers toward the important trees. There are three main steps to this process. First, you need to identify the *blinders*; next, you need to identify the *trade-offs*; and then you need to *decide* and iterate on a solution.

Identify the Blinders

When you first approach a problem, you'll often discover that a group of people has already been wrestling with it for years. These folks have been steeped in the problem for so long that they're wearing "blinders"—that is, they're no longer able to see the forest. They make a bunch of assumptions about the problem (or solution) without realizing it. "This is how we've always done it," they'll say, having lost the ability to consider the status quo critically. Sometimes, you'll discover bizarre coping mechanisms or rationalizations that have evolved to justify the status quo. This is where you —with fresh eyes—have a great advantage. You can see these blinders, ask

questions, and then consider new strategies. (Of course, being unfamiliar with the problem isn't a requirement for good leadership, but it's often an advantage.)

Identify the Key Trade-Offs

By definition, important and ambiguous problems do *not* have magic "silver bullet" solutions. There's no answer that works forever in all situations. There is only the *best answer for the moment*, and it almost certainly involves making trade-offs in one direction or another. It's your job to call out the trade-offs, explain them to everyone, and then help decide how to balance them.

Decide, Then Iterate

After you understand the trade-offs and how they work, you're empowered. You can use this information to make the best decision for this particular month. Next month, you might need to reevaluate and rebalance the trade-offs again; it's an iterative process. This is what we mean when we say *Always Be Deciding*.

There's a risk here. If you don't frame your process as continuous rebalancing of trade-offs, your teams are likely to fall into the trap of searching for the perfect solution, which can then lead to what some call "analysis paralysis." You need to make your teams comfortable with iteration. One way of doing this is to lower the stakes and calm nerves by explaining: "We're going to try this decision and see how it goes. Next month, we can undo the change or make a different decision." This keeps folks flexible and in a state of learning from their choices.

CASE STUDY: ADDRESSING THE "LATENCY" OF WEB SEARCH

In managing a team of teams, there's a natural tendency to move away from a single product and to instead own a whole "class" of products, or perhaps a broader problem that crosses products. A good example of this at Google has to do with our oldest product, Web Search.

For years, thousands of Google engineers have worked on the general problem of making search results better—improving the "quality" of the results page. But it turns out that this quest for quality has a side effect: it gradually makes the product slower. Once upon a time, Google's search results were not much more than a page of 10 blue links, each representing a relevant website. Over the past decade, however, thousands of tiny changes to improve "quality" have resulted in ever-richer results: images, videos, boxes with Wikipedia facts, even interactive UI elements. This means the servers need to do much more work to generate information: more bytes are being sent over the wire; the client (usually a phone) is being asked to render ever-more-complex HTML and data. Even though the speed of networks and computers have markedly increased over a decade, the speed of the search page has become slower and slower: its *latency* has increased. This might not seem like a big deal, but the latency of a product has a direct effect (in aggregate) on users' engagement and how often they use it. Even increases in rendering time as small as 10 ms matter. Latency creeps up slowly. This is not the fault of a specific engineering team, but rather represents a long, collective poisoning of the commons. At some point, the overall latency of Web Search grows until its effect begins to cancel out the improvements in user engagement that came from the improvements to the "quality" of the results.

A number of leaders struggled with this issue over the years but failed to address the problem systematically. The blinders everyone wore assumed that the only way to deal with latency was to declare a latency "code yellow"[1] every two or three years, during which everyone dropped everything to optimize code and speed up the product. Although this strategy would work temporarily, the latency would begin creeping up again just a month or two later, and soon return to its prior levels.

So what changed? At some point, we took a step back, identified the blinders, and did a full reevaluation of the trade-offs. It turns out that the pursuit of "quality" has not one, but *two* different costs. The first cost is to the user: more quality usually means more data being sent out, which means more

latency. The second cost is to Google: more quality means doing more work to generate the data, which costs more CPU time in our servers—what we call "serving capacity." Although leadership had often trodden carefully around the trade-off between quality and capacity, it had never treated latency as a full citizen in the calculus. As the old joke goes, "Good, Fast, Cheap—pick two." A simple way to depict the trade-offs is to draw a triangle of tension between Good (Quality), Fast (Latency), and Cheap (Capacity), as illustrated in Figure 6-1.

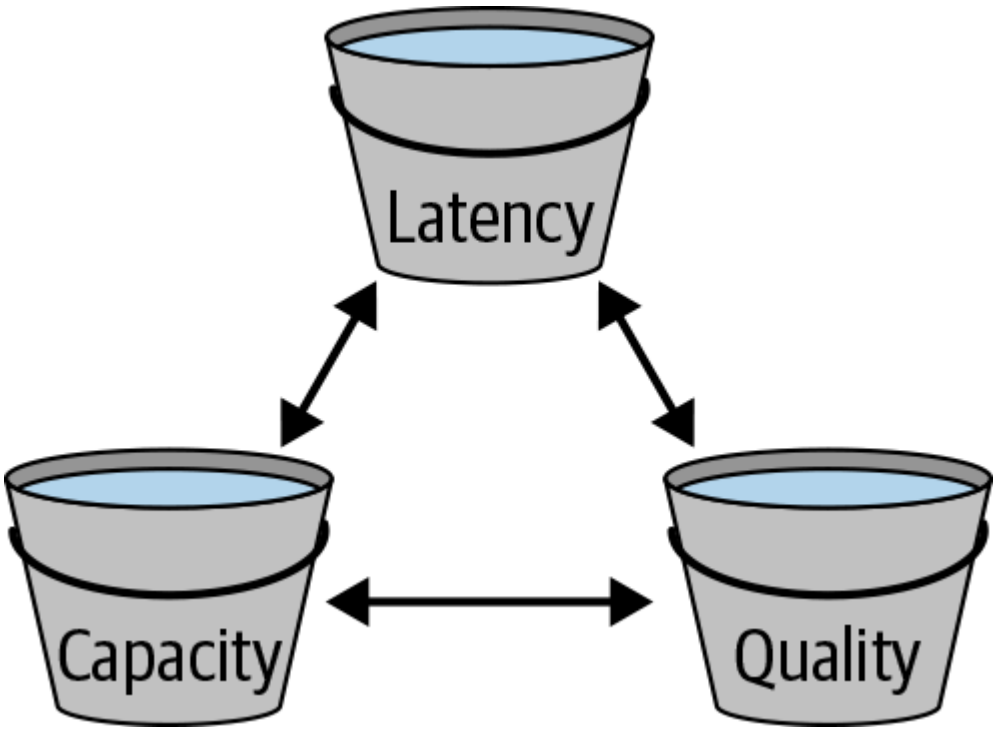

Figure 6-1. Trade-offs within Web Search; pick two!

That's exactly what was happening here. It's easy to improve any one of these traits by deliberately harming at least one of the other two. For example, you can improve quality by putting more data on the search results page—but doing so will hurt capacity and latency. You can also do a direct trade-off between latency and capacity by changing the traffic load on your serving cluster. If you send more queries to the cluster, you get increased capacity in the sense that you get better utilization of the CPUs—more bang for your hardware buck. But higher load increases resource contention within a computer, making the average latency of a query worse. If you deliberately decrease a cluster's traffic (run it "cooler"), you have less serving capacity overall, but each query becomes faster.

The main point here is that this insight—a better understanding of *all* the trade-offs—allowed us to start experimenting with new ways of balancing. Instead of treating latency as an unavoidable and accidental side effect, we

could now treat it as a first-class goal along with our other goals. This led to new strategies for us. For example, our data scientists were able to measure exactly how much latency hurt user engagement. This allowed them to construct a metric that pitted quality-driven improvements to short-term user engagement against latency-driven damage to long-term user engagement. This approach allows us to make more data-driven decisions about product changes. For example, if a small change improves quality but also hurts latency, we can quantitatively decide whether the change is worth launching or not. We are *always deciding* whether our quality, latency, and capacity changes are in balance, and iterating on our decisions every month.

Always Be Leaving

At face value, *Always Be Leaving* sounds like terrible advice. Why would a good leader be trying to leave? In fact, this is a famous quote from Bharat Mediratta, a former Google engineering director. What he meant was that it's not just your job to solve an ambiguous problem, but to get your organization to solve it *by itself*, without you present. If you can do that, it frees you up to move to a new problem (or new organization), leaving a trail of self-sufficient success in your wake.

The antipattern here, of course, is a situation in which you've set yourself up to be a single point of failure (SPOF). As we noted earlier in this book, Googlers have a term for that, the bus factor: *the number of people that need to get hit by a bus before your project is completely doomed.*

Of course, the "bus" here is just a metaphor. People become sick; they switch teams or companies; they move away. As a litmus test, think about a difficult problem that your team is making good progress on. Now imagine that you, the leader, disappear. Does your team keep going? Does it continue to be successful? Here's an even simpler test: think about the last vacation you took that was at least a week long. Did you keep checking your work email? (Most leaders do.) Ask yourself *why*. Will things fall apart if you don't pay attention? If so, you have very likely made yourself an SPOF. You need to fix that.

Your Mission: Build a "Self-Driving" Team

Coming back to Bharat's quote: being a successful leader means building an organization that is able to solve the difficult problem by itself. That

organization needs to have a strong set of leaders, healthy engineering processes, and a positive, self-perpetuating culture that persists over time. Yes, this is difficult; but it gets back to the fact that leading a team of teams is often more about organizing *people* rather than being a technical wizard. Again, there are three main parts to constructing this sort of self-sufficient group: dividing the problem space, delegating subproblems, and iterating as needed.

Dividing the Problem Space

Challenging problems are usually composed of difficult subproblems. If you're leading a team of teams, an obvious choice is to put a team in charge of each subproblem. The risk, however, is that the subproblems can change over time, and rigid team boundaries won't be able to notice or adapt to this fact. If you're able, consider an organizational structure that is looser—one in which subteams can change size, individuals can migrate between subteams, and the problems assigned to subteams can morph over time. This involves walking a fine line between "too rigid" and "too vague." On the one hand, you want your subteams to have a clear sense of problem, purpose, and steady accomplishment; on the other hand, people need the freedom to change direction and try new things in response to a changing environment.

Example: Subdividing the "latency problem" of Google Search

When approaching the problem of Search latency, we realized that the problem could, at a minimum, be subdivided into two general spaces: work that addressed the *symptoms* of latency, and different work that addressed the *causes* of latency. It was obvious that we needed to staff many projects to optimize our codebase for speed, but focusing *only* on speed wouldn't be enough. There were still thousands of engineers increasing the complexity and "quality" of search results, undoing the speed improvements as quickly as they landed, so we also needed people to focus on a parallel problem space of preventing latency in the first place. We discovered gaps in our metrics, in our latency analysis tools, and in our developer education and documentation. By assigning different teams to work on latency causes and symptoms at the same time, we were able to systematically control latency over the long term. (Also, notice how these teams owned the *problems*, not specific solutions!)

Delegating subproblems to leaders

It's essentially a cliché for management books to talk about "delegation," but there's a reason for that: delegation is *really difficult* to learn. It goes against all our instincts for efficiency and achievement. That difficulty is the reason for the adage, "If you want something done right, do it yourself."

That said, if you agree that your mission is to build a self-driving organization, the main mechanism of teaching is through delegation. You must build a set of self-sufficient leaders, and delegation is absolutely the most effective way to train them. You give them an assignment, let them fail, and then try again and try again. Silicon Valley has well-known mantras about "failing fast and iterating." That philosophy doesn't just apply to engineering design, but to human learning as well.

As a leader, your plate is constantly filling up with important tasks that need to be done. Most of these tasks are things that are fairly easy for you do. Suppose that you're working diligently through your inbox, responding to problems, and then you decide to put 20 minutes aside to fix a longstanding and nagging issue. But before you carry out the task, be mindful and stop yourself. Ask this critical question: *Am I really the only one who can do this work?*

Sure, it might be most *efficient* for you to do it, but then you're failing to train your leaders. You're not building a self-sufficient organization. Unless the task is truly time sensitive and on fire, bite the bullet and assign the work to someone else—presumably someone who you know can do it but will probably take much longer to finish. Coach them on the work if need be. You need to create opportunities for your leaders to grow; they need to learn to "level up" and do this work themselves so that you're no longer in the critical path.

The corollary here is that you need to be mindful of your own purpose as a leader of leaders. If you find yourself deep in the weeds, you're doing a disservice to your organization. When you get to work each day, ask yourself a different critical question: *What can I do that* nobody *else on my team can do?*

There are a number of good answers. For example, you can protect your teams from organizational politics; you can give them encouragement; you can make

sure everyone is treating one another well, creating a culture of humility, trust, and respect. It's also important to "manage up," making sure your management chain understands what your group is doing and staying connected to the company at large. But often the most common and important answer to this question is: "I can see the forest through the trees." In other words, you can *define a high-level strategy.* Your strategy needs to cover not just overall technical direction, but an organizational strategy as well. You're building a blueprint for how the ambiguous problem is solved and how your organization can manage the problem over time. You're continuously mapping out the forest, and then assigning the tree-cutting to others.

Adjusting and iterating

Let's assume that you've now reached the point at which you've built a self-sustaining machine. You're no longer an SPOF. Congratulations! What do you do now?

Before answering, note that you have actually liberated yourself—you now have the freedom to "Always Be Leaving." This could be the freedom to tackle a new, adjacent problem, or perhaps you could even move yourself to a whole new department and problem space, making room for the careers of the leaders you've trained. This is a great way of avoiding personal burnout.

The simple answer to "what now?" is to *direct* this machine and keep it healthy. But unless there's a crisis, you should use a gentle touch. The book *Debugging Teams*[2] has a parable about making mindful adjustments:

> *There's a story about a Master of all things mechanical who had long since retired. His former company was having a problem that no one could fix, so they called in the Master to see if he could help find the problem. The Master examined the machine, listened to it, and eventually pulled out a worn piece of chalk and made a small X on the side of the machine. He informed the technician that there was a loose wire that needed repair at that very spot. The technician opened the machine and tightened the loose wire, thus fixing the problem. When the Master's invoice arrived for $10,000, the irate CEO wrote back demanding a breakdown for this ridiculously high charge for a simple chalk mark! The Master responded with another invoice, showing a $1 cost for the chalk to make the mark, and $9,999 for knowing where to put it.*
>
> *To us, this is a story about wisdom: that a single, carefully considered adjustment can have gigantic effects. We use this technique when managing people. We imagine our team as flying around in a great blimp, headed slowly and surely in a certain direction. Instead of micromanaging and trying to make continuous course corrections, we spend most of the week carefully watching and listening. At the end of the week we make a small chalk mark in a precise location on the blimp, then give a small but critical "tap" to adjust the course.*

This is what good management is about: 95% observation and listening, and 5% making critical adjustments in just the right place. Listen to your leaders and skip-reports. Talk to your customers, and remember that often (especially if your team builds engineering infrastructure), your "customers" are not end users out in the world, but your coworkers. Customers' happiness requires just as much intense listening as your reports' happiness. What's working and what isn't? Is this self-driving blimp headed in the proper direction? Your direction should be iterative, but thoughtful and minimal, making the minimum adjustments necessary to correct course. If you regress into micromanagement, you risk becoming an SPOF again! "Always Be Leaving" is a call to *macro*management.

Take care in anchoring a team's identity

A common mistake is to put a team in charge of a specific product rather than a general problem. A product is a *solution* to a problem. The life expectancy of solutions can be short, and products can be replaced by better solutions.

However, a *problem*—if chosen well—can be evergreen. Anchoring a team identity to a specific solution ("We are the team that manages the Git repositories") can lead to all sorts of angst over time. What if a large percentage of your engineers want to switch to a new version control system? The team is likely to "dig in," defend its solution, and resist change, even if this is not the best path for the organization. The team clings to its blinders, because the solution has become part of the team's identity and self-worth. If the team instead owns the *problem* (e.g., "We are the team that provides version control to the company"), it is freed up to experiment with different solutions over time.

Always Be Scaling

A lot of leadership books talk about "scaling" in the context of learning to "maximize your impact"—strategies to grow your team and influence. We're not going to discuss those things here beyond what we've already mentioned. It's probably obvious that building a self-driving organization with strong leaders is already a great recipe for growth and success.

Instead, we're going to discuss team scaling from a *defensive* and personal point of view rather than an offensive one. As a leader, *your most precious resource is your limited pool of time, attention, and energy.* If you aggressively build out your teams' responsibilities and power without learning to protect your personal sanity in the process, the scaling is doomed to fail. And so we're going to talk about how to effectively scale *yourself* through this process.

The Cycle of Success

When a team tackles a difficult problem, there's a standard pattern that emerges, a particular cycle. It looks like this:

Analysis

: First, you receive the problem and start to wrestle with it. You identify the blinders, find all the trade-offs, and build consensus about how to manage them.

Struggle

You start moving on the work, whether or not your team thinks it's ready. You prepare for failures, retries, and iteration. At this point, your job is mostly about herding cats. Encourage your leaders and experts on the ground to form opinions and then listen carefully and devise an overall strategy, even if you have to "fake it" at first.[3]

Traction

Eventually your team begins to figure things out. You're making smarter decisions, and real progress is made. Morale improves. You're iterating on trade-offs, and the organization is beginning to drive itself around the problem. Nice job!

Reward

Something unexpected happens. Your manager takes you aside and congratulates you on your success. You discover your reward isn't just a pat on the back, but a *whole new problem* to tackle. That's right: the reward for success is more work…and more responsibility! Often, it's a problem that is similar or adjacent to the first one, but equally difficult.

So now you're in a pickle. You've been given a new problem, but (usually) not more people. Somehow you need to solve *both* problems now, which likely means that the original problem still needs to be managed with *half* as many people in *half* the time. You need the other half of your people to tackle the new work! We refer to this final step as the *compression stage*: you're taking everything you've been doing and compressing it down to half the size.

So really, the cycle of success is more of a spiral (see Figure 6-2). Over months and years, your organization is scaling by tackling new problems and then figuring out how to compress them so that it can take on new, parallel struggles. If you're lucky, you're allowed to hire more people as you go. More often than not, though, your hiring doesn't keep pace with the scaling. Larry Page, one of Google's founders, would probably refer to this spiral as "uncomfortably exciting."

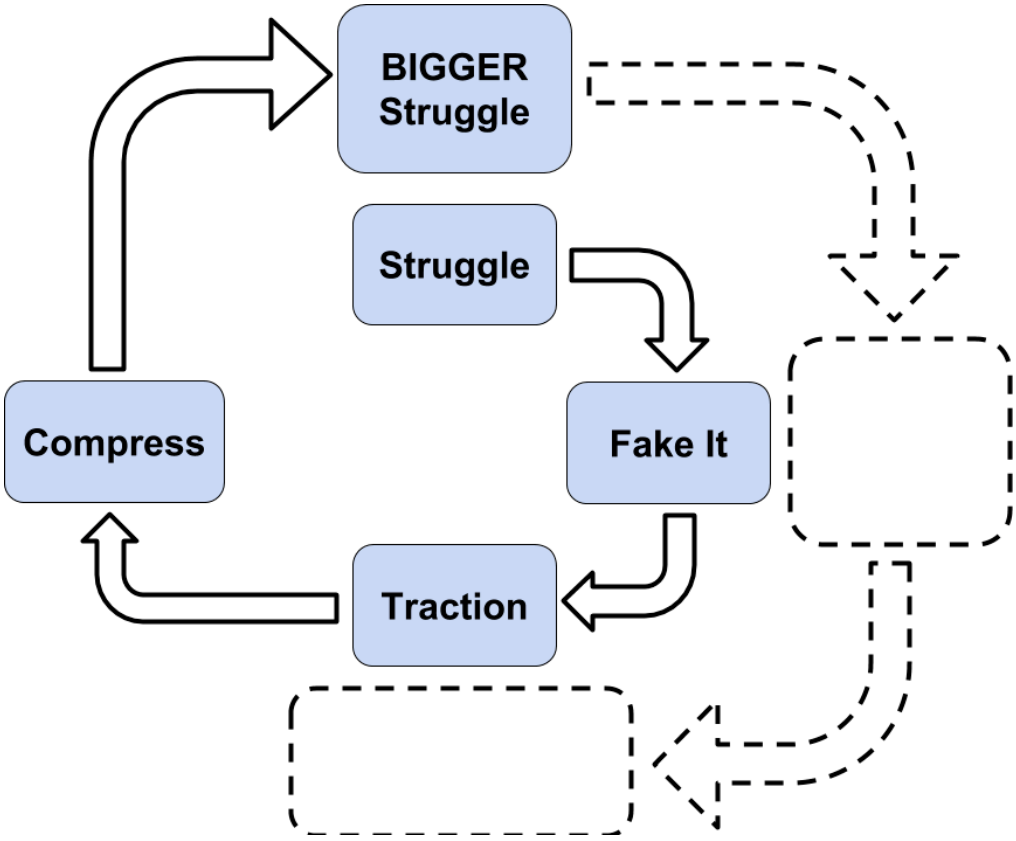

Figure 6-2. The spiral of success

The spiral of success is a conundrum—it's something that's difficult to manage, and yet it's the main paradigm for scaling a team of teams. The act of compressing a problem isn't just about figuring out how to maximize your team's efficiency, but also about learning to scale your *own* time and attention to match the new breadth of responsibility.

Important Versus Urgent

Think back to a time when you weren't yet a leader, but still a carefree individual contributor. If you used to be a programmer, your life was likely calmer and more panic-free. You had a list of work to do, and each day you'd methodically work down your list, writing code and debugging problems. Prioritizing, planning, and executing your work was straightforward.

As you moved into leadership, though, you might have noticed that your main mode of work became less predictable and more about firefighting. That is, your job became less *proactive* and more *reactive.* The higher up in leadership you go, the more escalations you receive. You are the "finally" clause in a long list of code blocks! All of your means of communication—email, chat rooms, meetings—begin to feel like a Denial-of-Service attack against your time and attention. In fact, if you're not mindful, you end up spending 100% of your time in reactive mode. People are throwing balls at you, and you're

frantically jumping from one ball to the next, trying not to let any of them hit the ground.

A lot of books have discussed this problem. The management author Stephen Covey is famous for talking about the idea of distinguishing between things that are *important* versus things that are *urgent.* In fact, it was US President Dwight D. Eisenhower who popularized this idea in a famous 1954 quote:

> *I have two kinds of problems, the urgent and the important. The urgent are not important, and the important are never urgent.*

This tension is one of the biggest dangers to your effectiveness as a leader. If you let yourself slip into pure reactive mode (which happens almost automatically), you spend every moment of your life on *urgent* things, but almost none of those things are *important* in the big picture. Remember that your job as a leader is to do things that *only you can do*, like mapping a path through the forest. Building that meta-strategy is incredibly important, but almost never urgent. It's always easier to respond to that next urgent email.

So how can you force yourself to work mostly on important things, rather than urgent things? Here are a few key techniques:

Delegate

Many of the urgent things you see can be delegated back to other leaders in your organization. You might feel guilty if it's a trivial task; or you might worry that handing off an issue is inefficient because it might take those other leaders longer to fix. But it's good training for them, and it frees up your time to work on important things that only you can do.

Schedule dedicated time

Regularly block out two hours or more to sit quietly and work *only* on important-but-not-urgent things—things like team strategy, career paths for your leaders, or how you plan to collaborate with neighboring teams.

Find a tracking system that works

There are dozens of systems for tracking and prioritizing work. Some are software based (e.g., specific "to-do" tools), some are pen-and-paper based (the "Bullet Journal" method), and some systems are agnostic to implementation. In this last category, David Allen's book,

Getting Things Done, is quite popular among engineering managers; it's an abstract algorithm for working through tasks and maintaining a prized "inbox zero." The point here is to *try* these different systems and determine what works for you. Some of them will click with you and some will not, but you definitely need to find something more effective than tiny Post-It notes decorating your computer screen.

Learn to Drop Balls

There's one more key technique for managing your time, and on the surface it sounds radical. For many, it contradicts years of engineering instinct. As an engineer, you pay attention to detail; you make lists, you check things off lists, you're precise, and you finish what you start. That's why it feels so good to close bugs in a bug tracker, or whittle your email down to inbox zero. But as a leader of leaders, your time and attention are under constant attack. No matter how much you try to avoid it, you end up dropping balls on the floor—there are just too many of them being thrown at you. It's overwhelming, and you probably feel guilty about this all the time.

So, at this point, let's step back and take a frank look at the situation. If dropping some number of balls is inevitable, isn't it better to drop certain balls *deliberately* rather than *accidentally*? At least then you have some semblance of control.

Here's a great way to do that.

Marie Kondo is an organizational consultant and the author of the extremely popular book *The Life-Changing Magic of Tidying Up*. Her philosophy is about effectively decluttering all of the junk from your house, but it works for abstract clutter as well.

Think of your physical possessions as living in three piles. About 20% of your things are just useless—things that you literally never touch anymore, and all very easy to throw away. About 60% of your things are somewhat interesting; they vary in importance to you, and you sometimes use them, sometimes not. And then about 20% of your possessions are exceedingly important: these are the things you use *all* the time, that have deep emotional meaning, or, in Ms. Kondo's words, spark deep "joy" just holding them. The thesis of her book is that most people declutter their lives incorrectly: they spend time tossing the bottom 20% in the garbage, but the remaining 80% still feels too cluttered.

She argues that the *true* work of decluttering is about identifying the top 20%, not the bottom 20%. If you can identify only the critical things, you should then toss out the other 80%. It sounds extreme, but it's quite effective. It is greatly freeing to declutter so radically.

It turns out that you can also apply this philosophy to your inbox or task list—the barrage of balls being thrown at you. Divide your pile of balls into three groups: the bottom 20% are probably neither urgent nor important and very easy to delete or ignore. There's a middle 60%, which might contain some bits of urgency or importance, but it's a mixed bag. At the top, there's 20% of things that are absolutely, critically important.

And so now, as you work through your tasks, do *not* try to tackle the top 80%—you'll still end up overwhelmed and mostly working on urgent-but-not-important tasks. Instead, mindfully identify the balls that strictly fall in the top 20%—critical things that *only you can do*—and focus strictly on them. Give yourself explicit permission to drop the other 80%.

It might feel terrible to do so at first, but as you deliberately drop so many balls, you'll discover two amazing things. First, even if you don't delegate that middle 60% of tasks, your subleaders often notice and pick them up automatically. Second, if something in that middle bucket is truly critical, it ends up coming back to you anyway, eventually migrating up into the top 20%. You simply need to *trust* that things below your top-20% threshold will either be taken care of or evolve appropriately. Meanwhile, because you're focusing only on the critically important things, you're able to scale your time and attention to cover your group's ever-growing responsibilities.

Protecting Your Energy

We've talked about protecting your time and attention—but your personal energy is the other piece of the equation. All of this scaling is simply exhausting. In an environment like this, how do you stay charged and optimistic?

Part of the answer is that over time, as you grow older, your overall stamina builds up. Early in your career, working eight hours a day in an office can feel like a shock; you come home tired and dazed. But just like training for a marathon, your brain and body build up larger reserves of stamina over time.

The other key part of the answer is that leaders gradually learn to *manage* their energy more intelligently. It's something they learn to pay constant attention to. Typically, this means being aware of how much energy you have at any given moment, and making deliberate choices to "recharge" yourself at specific moments, in specific ways. Here are some great examples of mindful energy management:

Take real vacations

A weekend is not a vacation. It takes at least three days to "forget" about your work; it takes at least a week to actually feel refreshed. But if you check your work email or chats, you *ruin* the recharge. A flood of worry comes back into your mind, and all of the benefit of psychological distancing dissipates. The vacation recharges only if you are truly disciplined about disconnecting.[4] And, of course, this is possible only if you've built a self-driving organization.

Make it trivial to disconnect

When you disconnect, leave your work laptop at the office. If you have work communications on your phone, remove them. For example, if your company uses G Suite (Gmail, Google Calendar, etc.), a great trick is to install these apps in a "work profile" on your phone. This causes a second set of work-badged apps to appear on your phone. For example, you'll now have two Gmail apps: one for personal email, one for work email. On an Android phone, you can then press a single button to disable the entire work profile at once. All the work apps gray out, as if they were uninstalled, and you can't "accidentally" check work messages until you re-enable the work profile.

Take real weekends, too

A weekend isn't as effective as a vacation, but it still has some rejuvenating power. Again, this recharge works only if you disconnect from work communications. Try truly signing out on Friday night, spend the weekend doing things you love, and then sign in again on Monday morning when you're back in the office.

Take breaks during the day

Your brain operates in natural 90-minute cycles.[5] Use the opportunity to get up and walk around the office, or spend 10 minutes walking outside. Tiny breaks like this are only tiny recharges, but they can make a tremendous difference in your stress levels and how you feel over the next two hours of work.

Give yourself permission to take a mental health day

Sometimes, for no reason, you just have a bad day. You might have slept well, eaten well, exercised—and yet you are still in a terrible mood anyway. If you're a leader, this is an awful thing. Your bad mood sets the tone for everyone around you, and it can lead to terrible decisions (emails you shouldn't have sent, overly harsh judgements, etc.). If you find yourself in this situation, just turn around and go home, declaring a sick day. Better to get nothing done that day than to do active damage.

In the end, managing your energy is just as important as managing your time. If you learn to master these things, you'll be ready to tackle the broader cycle of scaling responsibility and building a self-sufficient team.

Conclusion

Successful leaders naturally take on more responsibility as they progress (and that's a good and natural thing). Unless they effectively come up with techniques to properly make decisions quickly, delegate when needed, and manage their increased responsibility, they might end up feeling overwhelmed. Being an effective leader doesn't mean that you need to make perfect decisions, do everything yourself, or work twice as hard. Instead, strive to always be deciding, always be leaving, and always be scaling.

TL;DRs

- Always Be Deciding: Ambiguous problems have no magic answer; they're all about finding the right *trade-offs* of the moment, and iterating.
- Always Be Leaving: Your job, as a leader, is to build an organization that automatically solves a class of ambiguous problems—over *time*—without you needing to be present.
- Always Be Scaling: Success generates more responsibility over time, and you must proactively manage the *scaling* of this work in order to protect your scarce resources of personal time, attention, and energy.

1 "Code yellow" is Google's term for "emergency hackathon to fix a critical problem." Affected teams are expected to suspend all work and focus 100% attention on the problem until the state of emergency is declared over.

2 Brian W. Fitzpatrick and Ben Collins-Sussman, *Debugging Teams: Better Productivity through Collaboration* (Boston: O'Reilly, 2016).

3 It's easy for imposter syndrome to kick in at this point. One technique for fighting the feeling that you don't know what you're doing is to simply pretend that *some* expert out there knows exactly what to do, and that they're simply on vacation and you're temporarily subbing in for them. It's a great way to remove the personal stakes and give yourself permission to fail and learn.

4 You need to plan ahead and build around the assumption that your work simply won't get done during vacation. Working hard (or smart) just before and after your vacation mitigates this issue.

5 You can read more about BRAC at *https://en.wikipedia.org/wiki/Basic_rest-activity_cycle*.