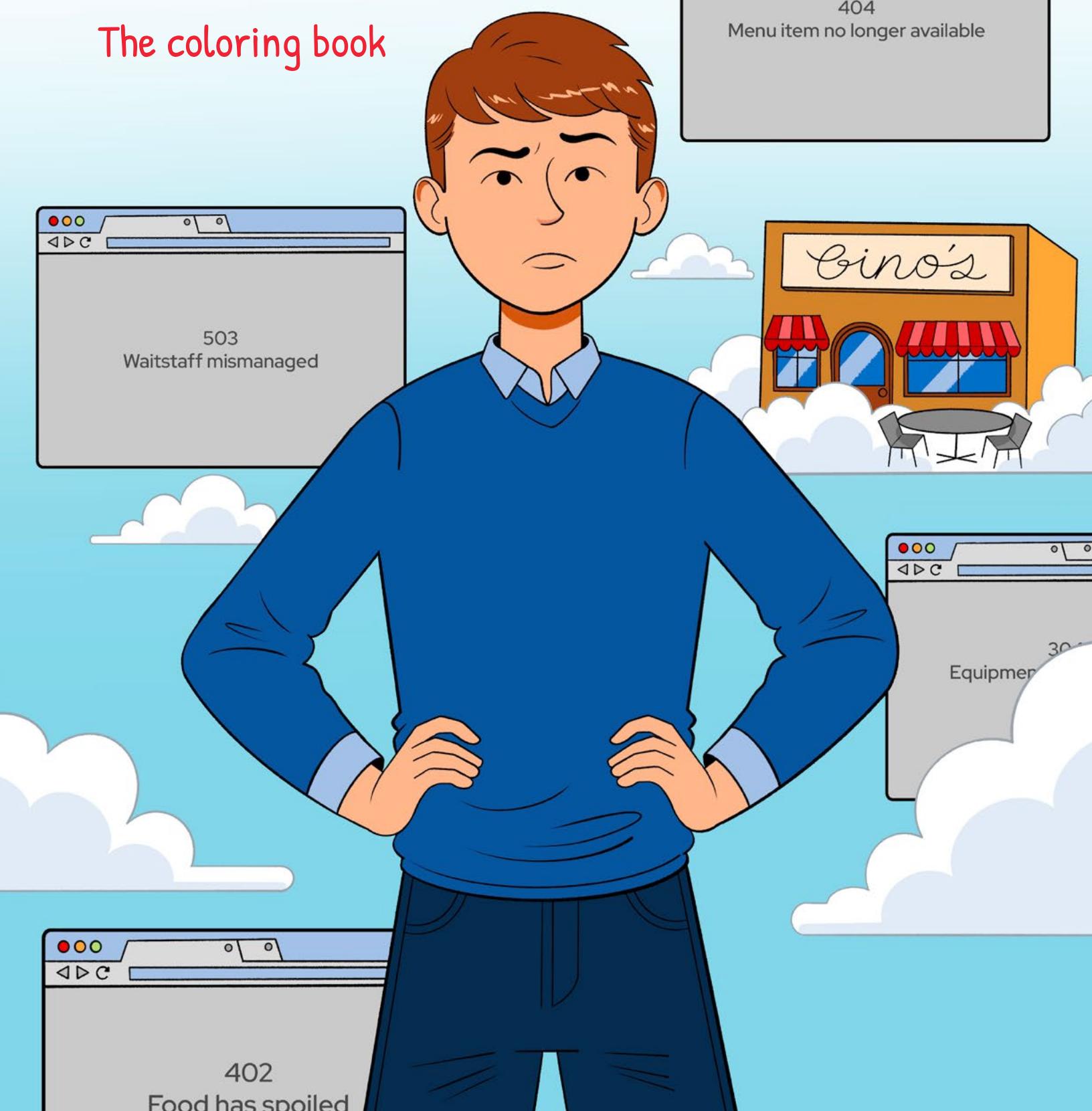


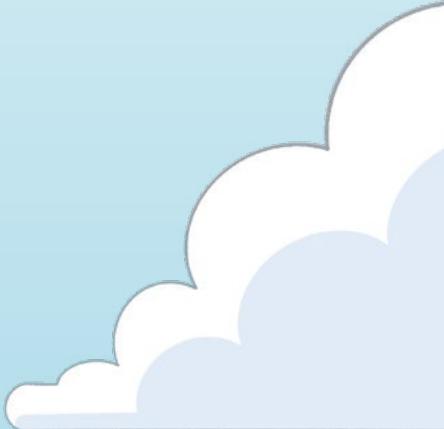
Reliability Nightmares

The coloring book



Reliability Nightmares

The coloring book



Script and storyboard by Máirín Duffy

Illustration by Wildfire

Concept and technical information by Jeremy Eder, Irit Goihman,
David Martin, and Craig Robinson



Gino's was a successful family-owned restaurant,
renowned for good food and good times.



Gino passed away and left the restaurant to his son, Leo, who quit his job to run it.
Gino had a natural instinct for running a restaurant. Leo does not.



Wrong food.



~~~~

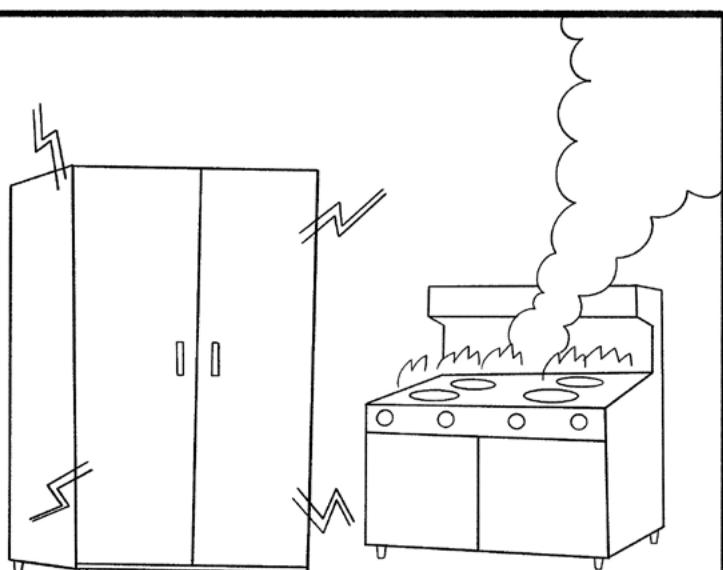
Cold food.



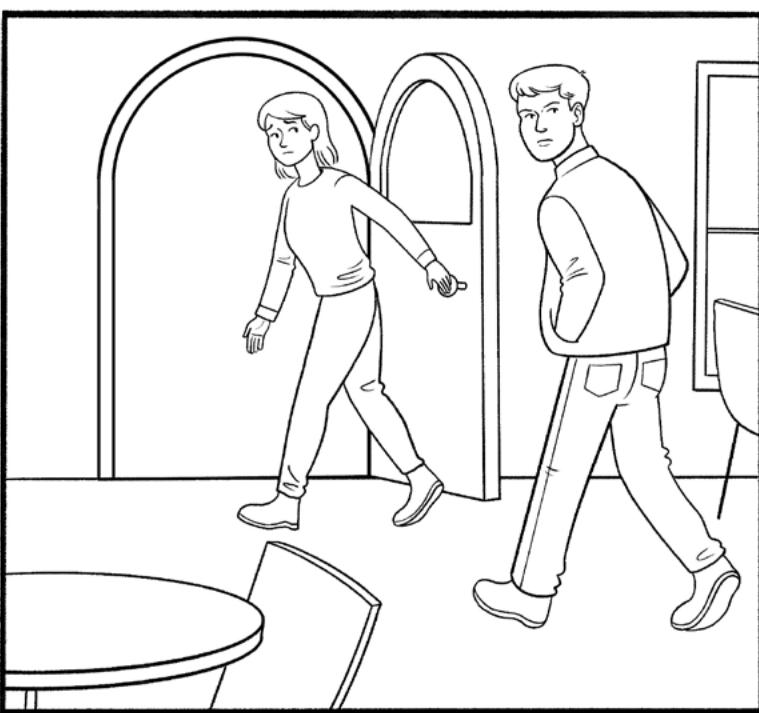
Gross food.



The waitstaff are either  
bombarded or bored.



Food spoils due to a freezer failure,  
and a poorly maintained oven  
causes a kitchen fire.



Loyal customers were  
disappointed and started leaving.  
Word was getting around that Gino's  
was "not what it used to be."

I can't do this.  
45 years in operation!  
Will this be the last?

KNOCK  
KNOCK

Who's that at the door?

Hi! I'm celebrity chef Cookie Cache!  
You're hosting your sister's wedding  
reception! Let's get this place  
whipped into shape!!

Ugh...Mia!

What do you say?  
Want to give it a go?

Yes!

Your sister Mia called us!

...Sure, let's do it.

The first task is for Chef Cache to evaluate Gino's dining experience—  
sample the menu and check out the service.

Our specials of the day are J2EE spaghetti with artisanal memory leaks, bash'ed shellfish on bare metal, and home-cooked biz-critical ancient LAMP stack pancakes.

I'll try the LAMP stack, please!

[90 minutes pass]

... Hey, uh, Leo?

Apologies for the wait! I'll check!

Sssssorry!!  
Here you are!

Help!  
HELP!

You saved me!

Go home, get a good  
night's sleep. We'll start  
again in the morning.

## Five Challenges

There is a better way. I will teach you  
how to run Gino's as a managed service.  
You'll have five challenges, one per day.  
We'll close Gino's for the week and get  
you ready for the wedding!

## #1. Observability



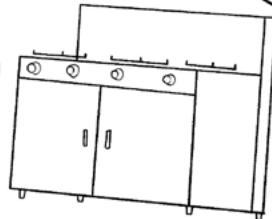
- Break down silos
- Consensus
- Enforce accountability
- Prioritize work
- Keep customers happy

The first challenge is to set up observability for the restaurant. This is foundational for improving Gino's service.

1. We need to break down the silos between each role with regular open communication.
2. We must be in consensus about the level of service we provide.
3. We enforce accountability by measuring causes of issues and working to address them.
4. We need to prioritize our work...
5. ...so we can keep our customers happy!

You need to know your equipment works, your food supplies are fresh and stocked, and that customers are served in a timely manner. You need data to achieve all of this.

How do we do it?



To accomplish this, we'll define service-level objectives for Gino's. SLOs are a set of goals you'd like to achieve that represent what your customers expect. We'll collect data to measure whether we meet our SLOs or not.



We've had a lot of dishes sent back lately because they weren't served right away and got cold. Can we set an SLO for that?



Yep! Let's set the SLO to 99% of customers will receive their food within 20 minutes of placing their order. How will we know we are meeting this SLO?



SLOs

1  $\frac{99}{100}$  IN 20 MIN

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

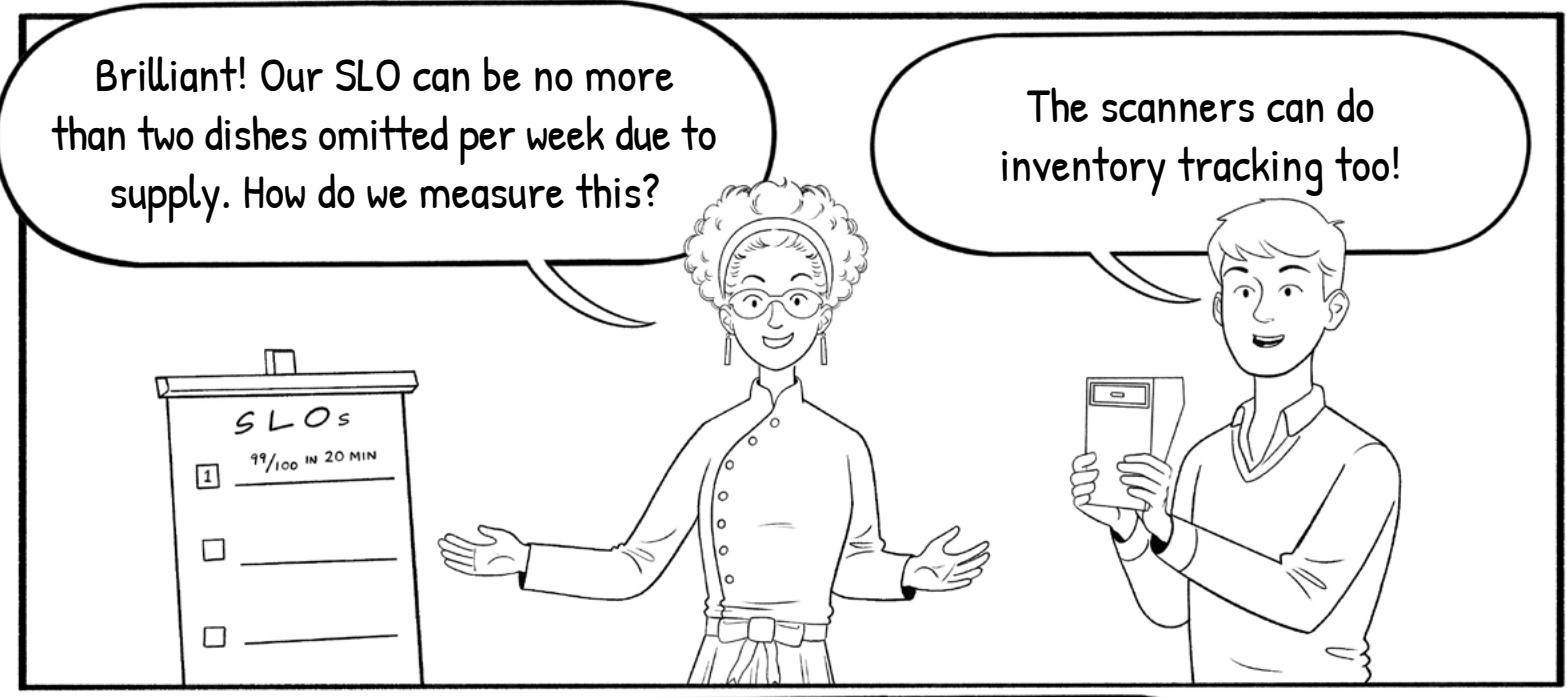
I have this new ticket scanning system...

Great, let's set it up. Chef, you scan when you get a ticket, scan when the food is ready, and waitstaff—scan when you pick the meals up to serve.

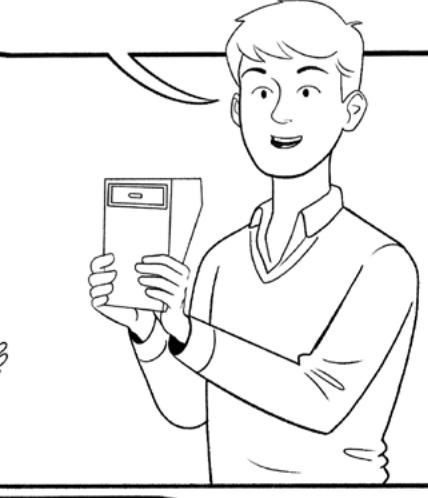




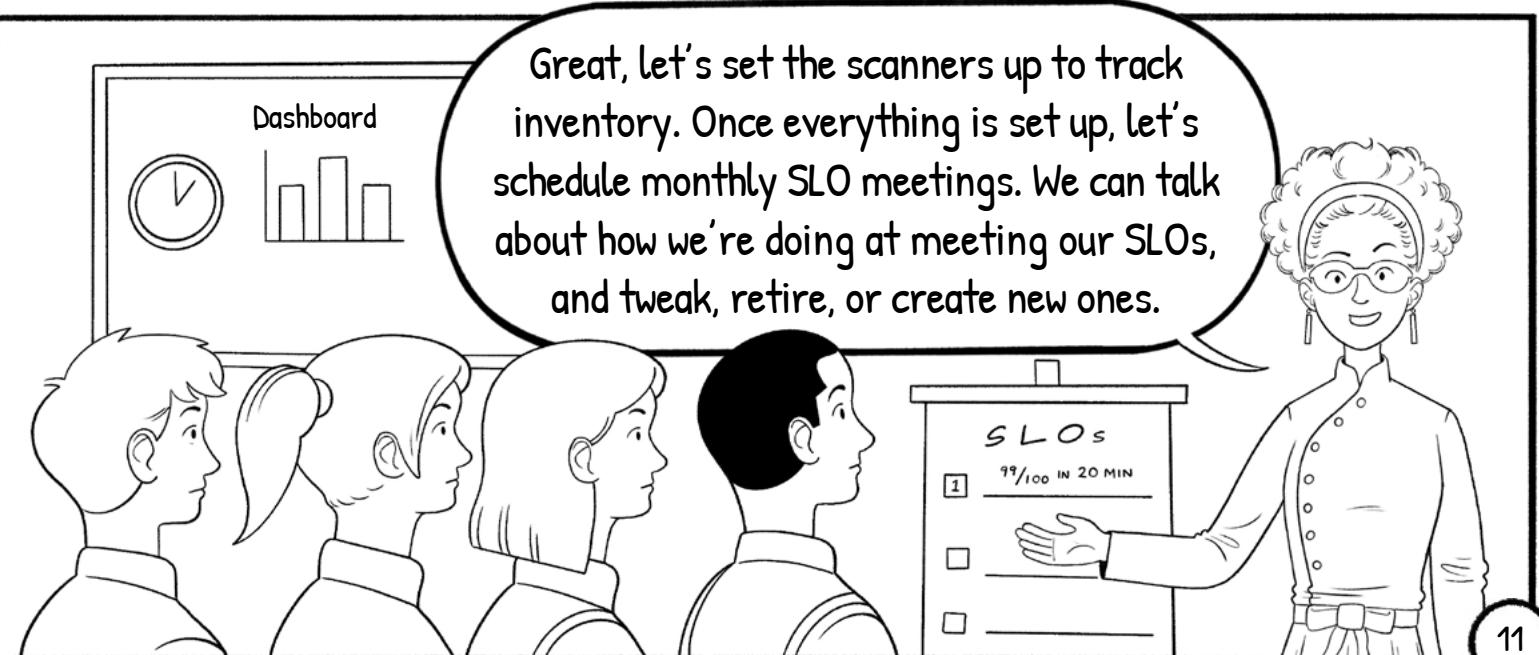
How about supplies? So we don't have to tell customers we're out of their favorite dishes quite so much?



Brilliant! Our SLO can be no more than two dishes omitted per week due to supply. How do we measure this?

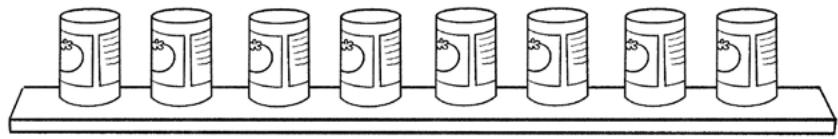


The scanners can do inventory tracking too!



Great, let's set the scanners up to track inventory. Once everything is set up, let's schedule monthly SLO meetings. We can talk about how we're doing at meeting our SLOs, and tweak, retire, or create new ones.

## #2. Safety and self-healing



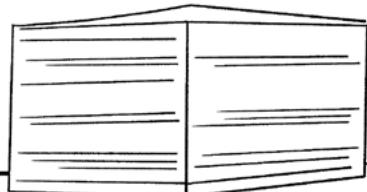
Great work on instrumenting for observability! Today's challenge is safety and self-healing. We'll know when things go wrong with the observability we set up yesterday. For issues that can be addressed through automation, we can self-heal!



So instead of simply tracking inventory—we automatically order more when low?



Yes! If Chef spends less time with ordering forms, that's more time to spend on customer experience. You can self-heal inventory issues.





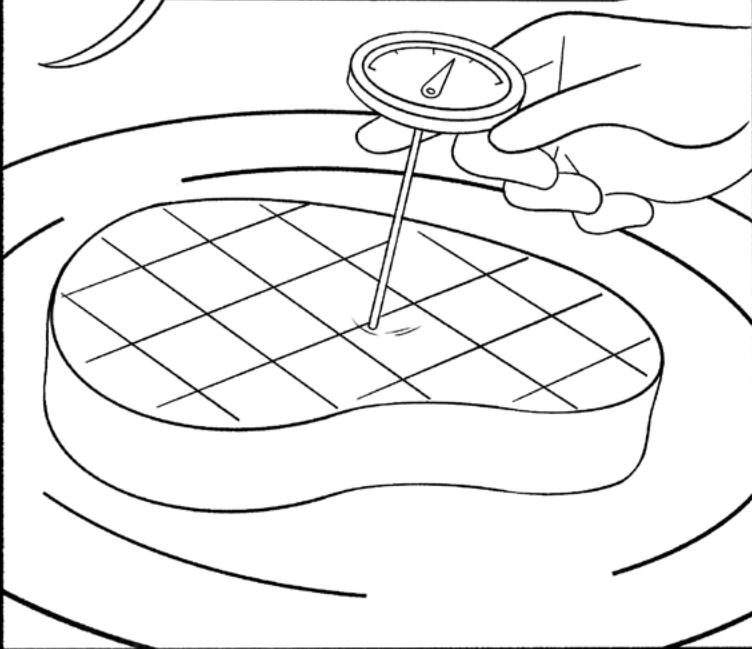
What about when it's less clear there's a problem? I had to bring 15 orders back last week because the customers disagreed about the steak doneness.

We can't be perfect on subjective things. Let's create an SLO—1/100 customers will send their steaks back. We can set an error budget for doneness. For a medium steak what would your range be?

Medium rare to medium-well done.  
Not well done or rare.

That means that of 100 steaks ordered, only one will be out of range. If we get more than one out of range, it is time to do something about it ASAP. Let's call that an 'incident'.

So measure it! Measure before you send them out. Time them and use a thermometer—don't just go by your gut—until you meet the SLO.



And if you hit two steak returns in one day, you can self-heal by using thermometers on every steak afterwards to make sure no more get returned and so you can recalibrate yourself.

Hmm...maybe something is wrong with the stove?

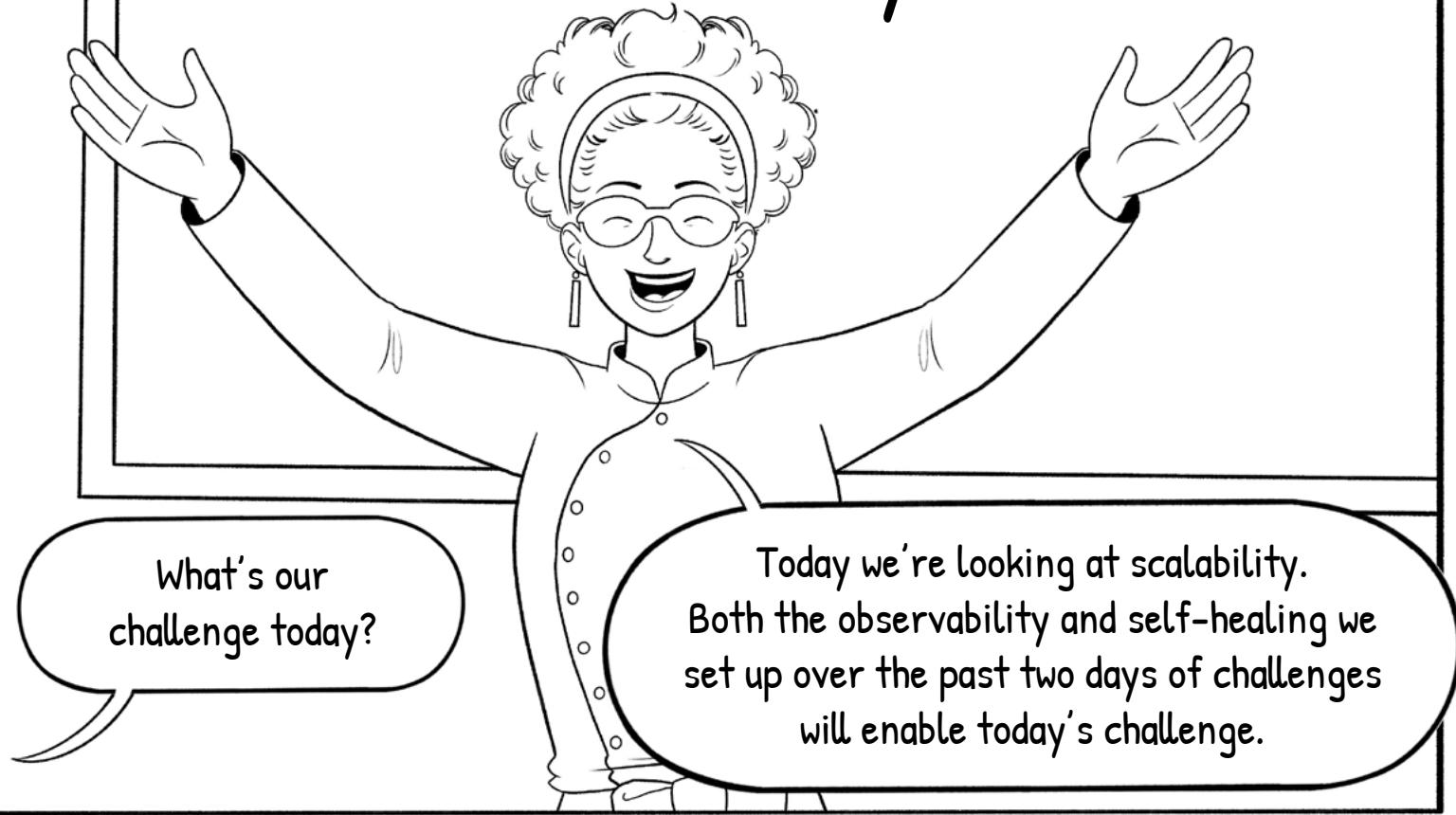


## #2. Safety and self

1. steak

So let's list out all of these kinds of potential issues and set error budgets for them. Then we can track them each day on our dashboard!

# #3. Scalability



We had an online coupon for 15% off that went viral last week. We got flooded with takeout orders—600 orders between 5-7 PM. We couldn't figure out how to shut off online orders...

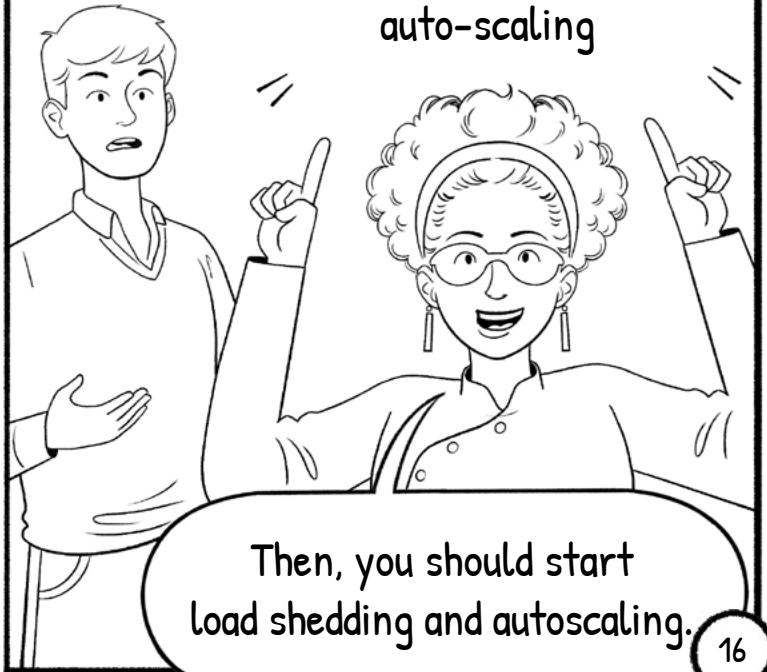


Let's address this. First, let's add the number of active online takeout orders to the dashboard. How many can you handle per hour?



So we set up an alert when we have 10+ takeout orders.  
Then what?

load shedding  
+  
auto-scaling

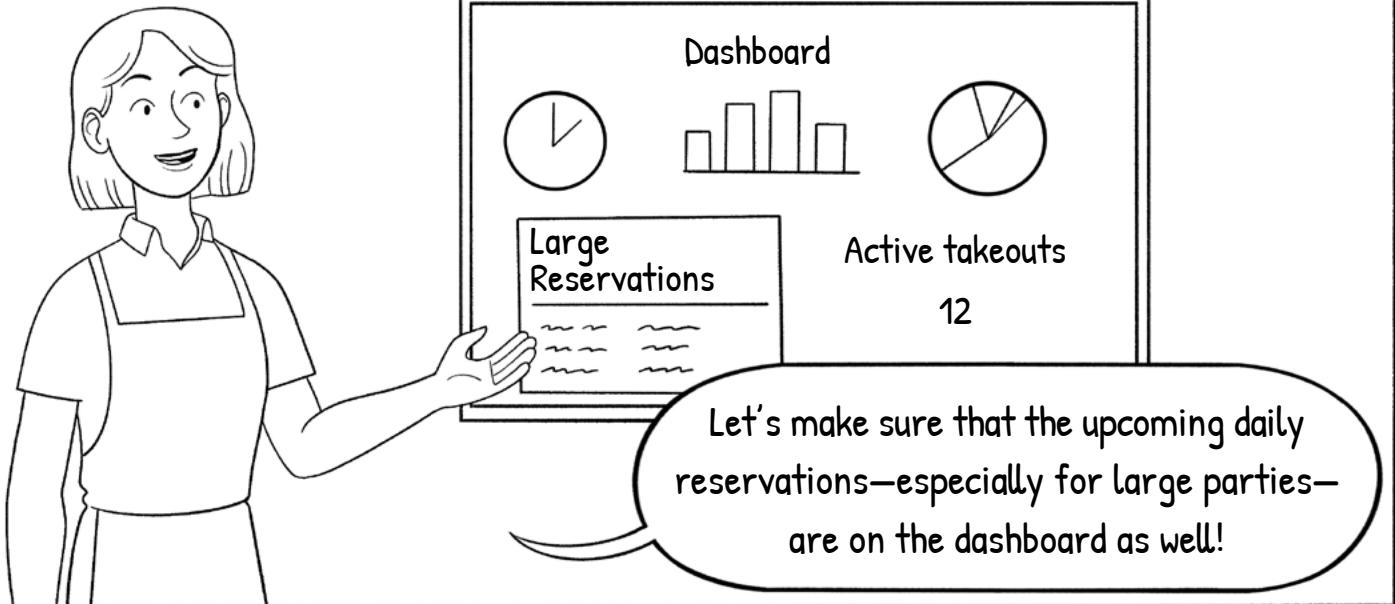


Load shedding is when you drop some tasks purposely to work through your queue. To load shed here, you could add a 20 minute wait to new dine-in guests. You can also pad out time estimates online to buy time to work thru the queue.



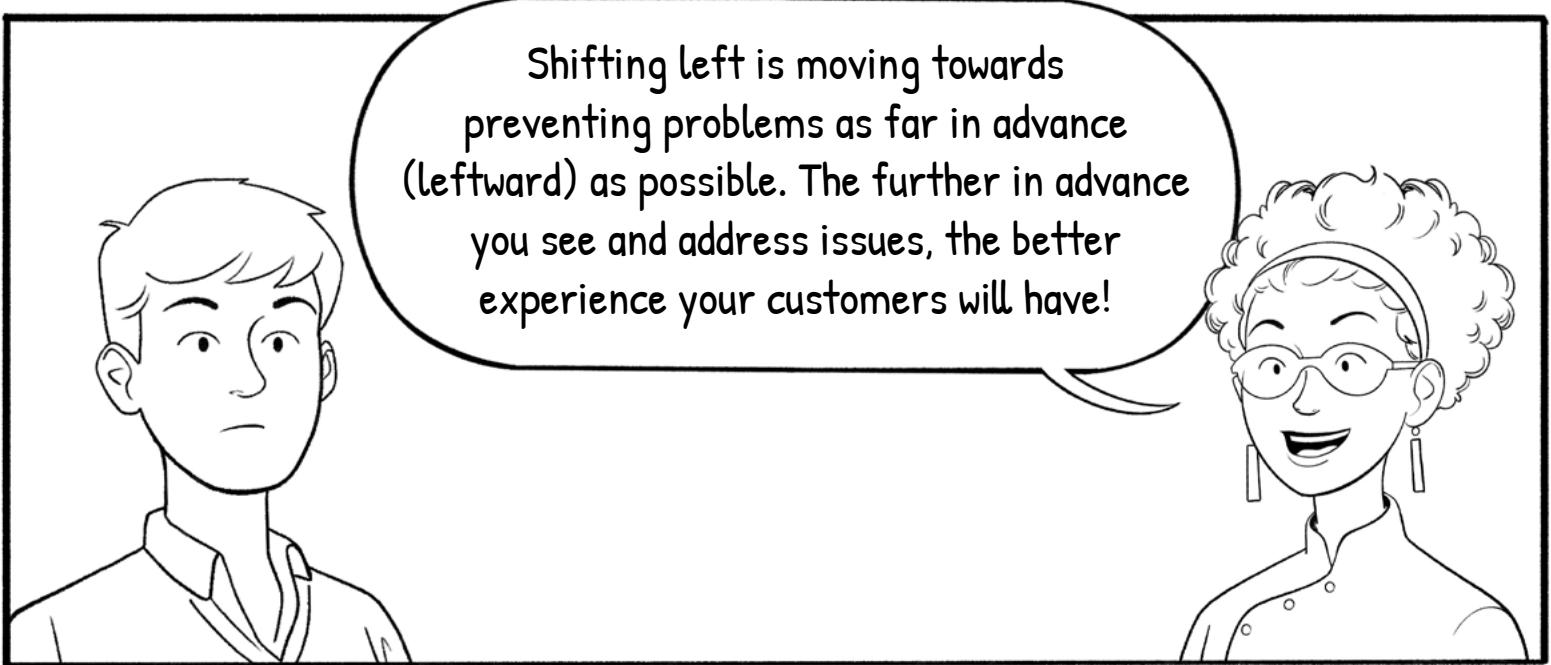
Autoscaling is when you horizontally scale out your capacity on demand. Say you have a large party reservation coming when takeout orders are high—you could call in extra staff to meet the demand if load shedding can't free up enough resources.



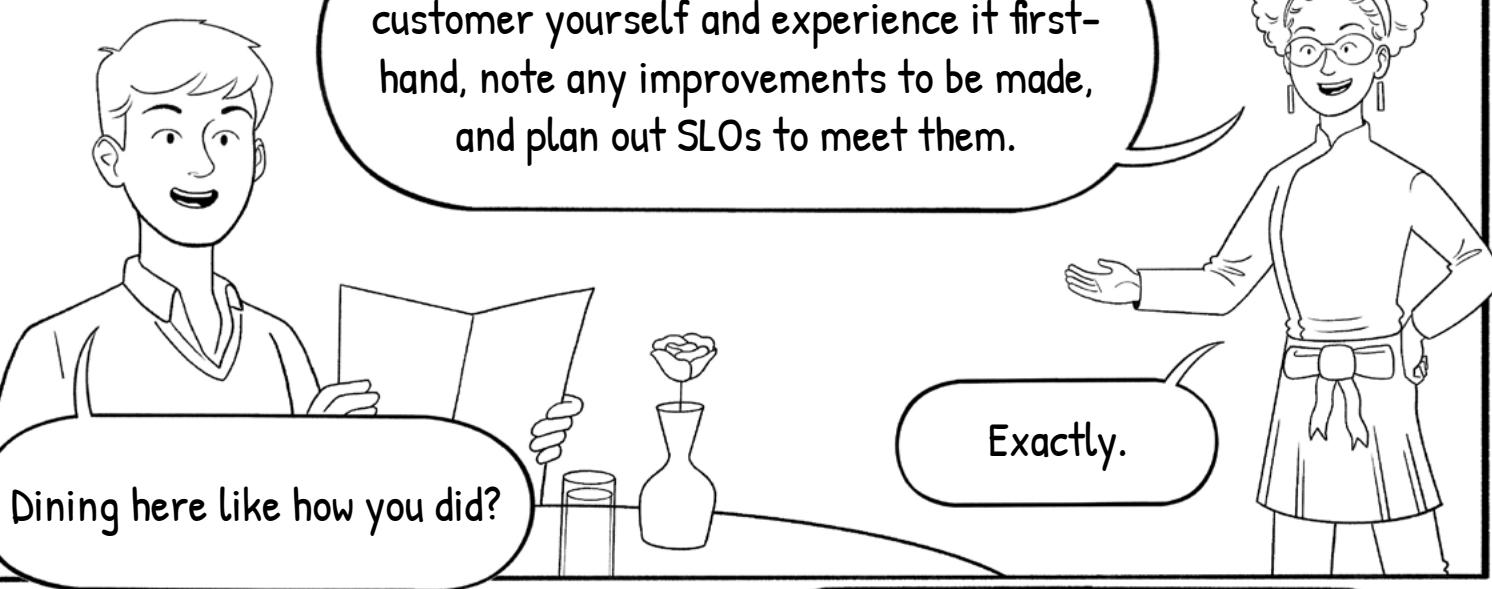


## #4. Shifting left





Shifting left is moving towards preventing problems as far in advance (leftward) as possible. The further in advance you see and address issues, the better experience your customers will have!



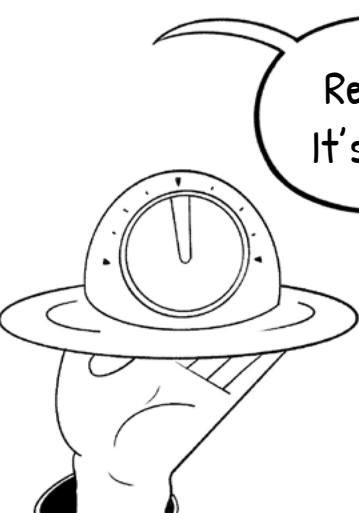


That's a great idea.  
I didn't think of that!

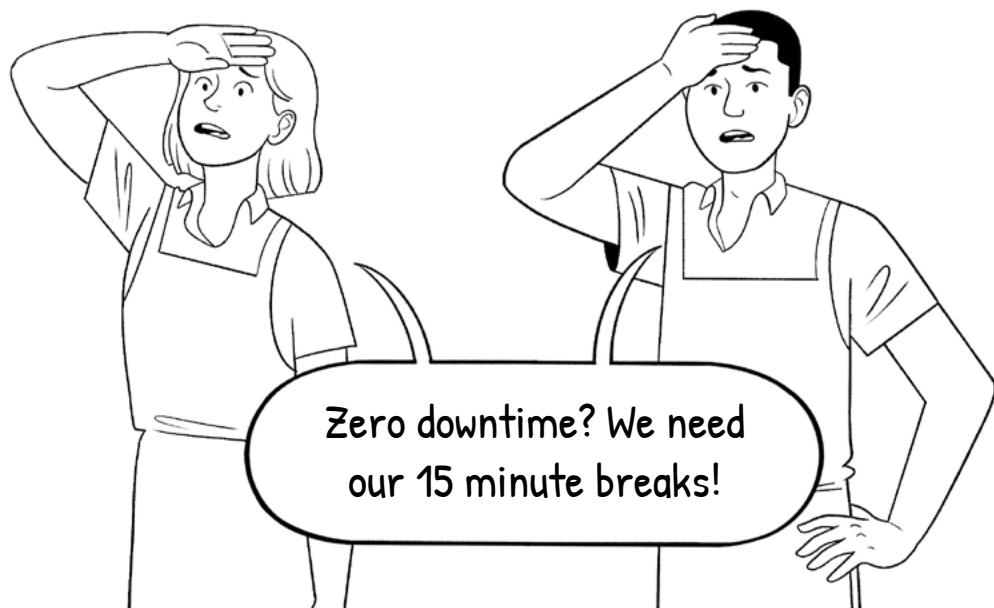
So let's prep a service for Leo  
and Chef—I'll be the stand-in chef,  
and I'll work off your menu.  
Let's see how it goes!



## #5. Zero downtime



Ready for today's challenge?  
It's...zero downtime upgrades!





Of course, of course! You need breaks to do your best work—we want to prevent burnout and help you do your best work!



Zero downtime upgrades are more about seamless transitions during needed changes. For example, waitstaff shift changes—instead of scheduling Annie to leave at the beginning of Thomas's shift...

...schedule their shifts to overlap, and slowly transfer guests from the departing waitstaff to the incoming waitstaff.

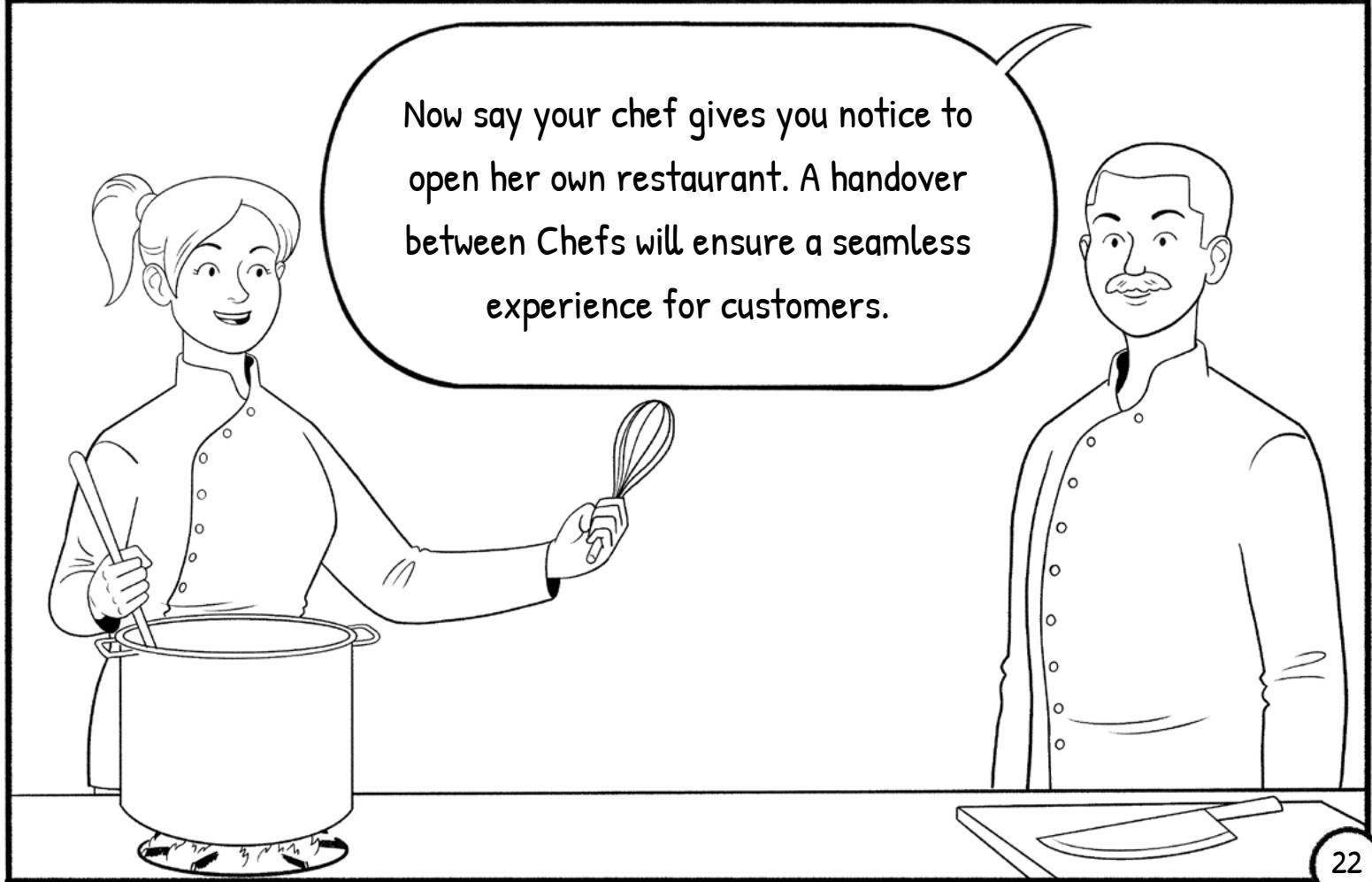


Once all guests are transferred, Annie can go home, confident she won't be called back.

That would be great! We're always so rushed between shifts, sometimes we neglect a customer waiting on a drink they ordered before our shift, for example!



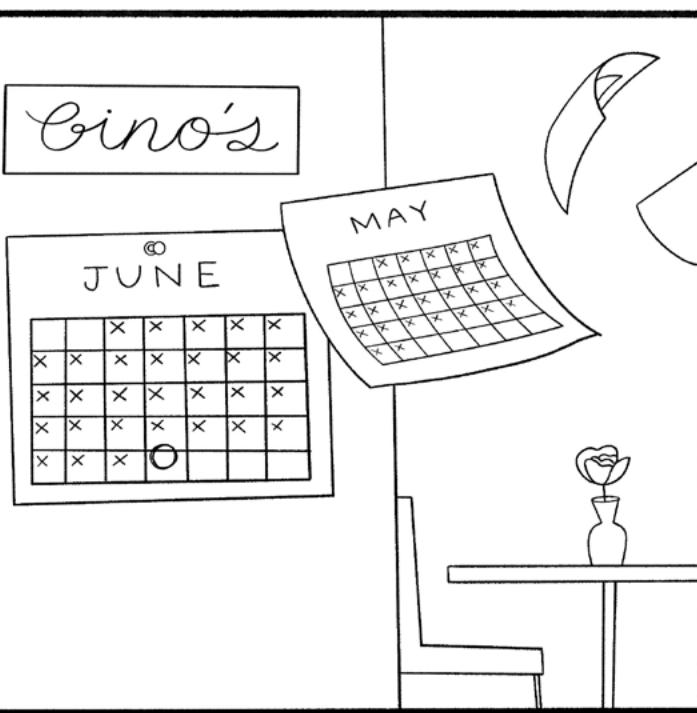
Now say your chef gives you notice to open her own restaurant. A handover between Chefs will ensure a seamless experience for customers.



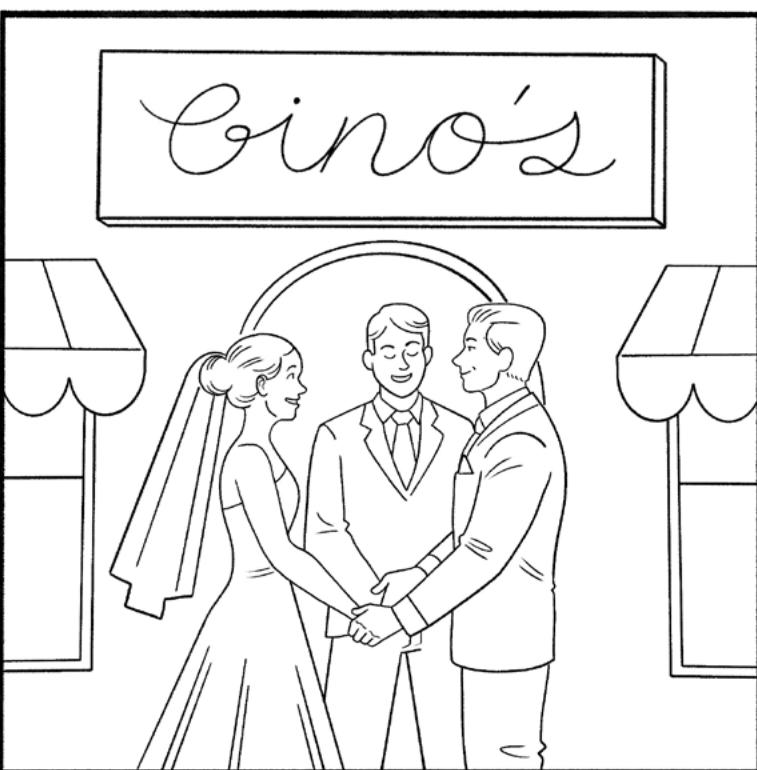
Great ideas—except for  
the thought of Chef leaving us.  
Let's work on adjusting our  
staffing schedules accordingly!



Chef Cookie Cache left Gino's  
at the end of day five.

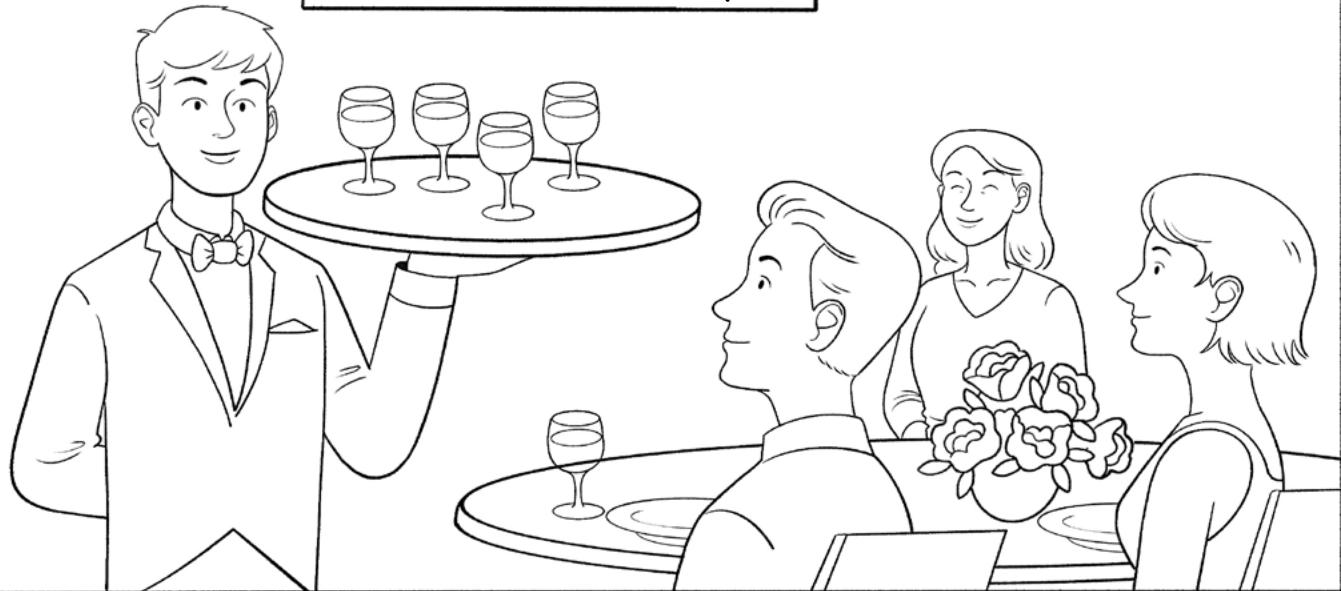


In the months leading up to his  
sister's wedding reception, Leo put  
into practice the lessons from the  
five managed service challenges...



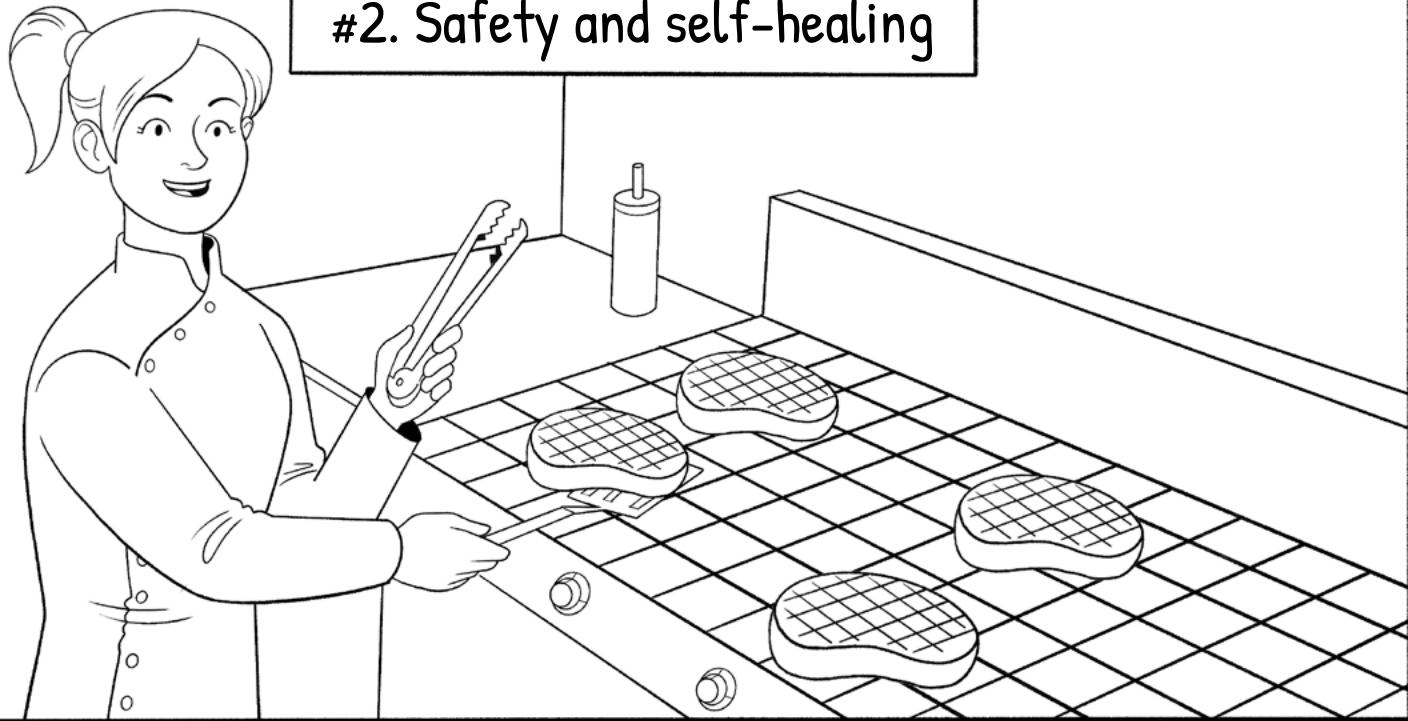
...and the wedding reception at Gino's  
went off without a hitch. Each lesson  
helped avert issues.

## #1. Observability



The dashboard pointed out when three reception tables were in danger of not meeting a 20 min food wait SLO—Leo was able to step in and help the waitstaff get the food to the guests on time.

## #2. Safety and self-healing



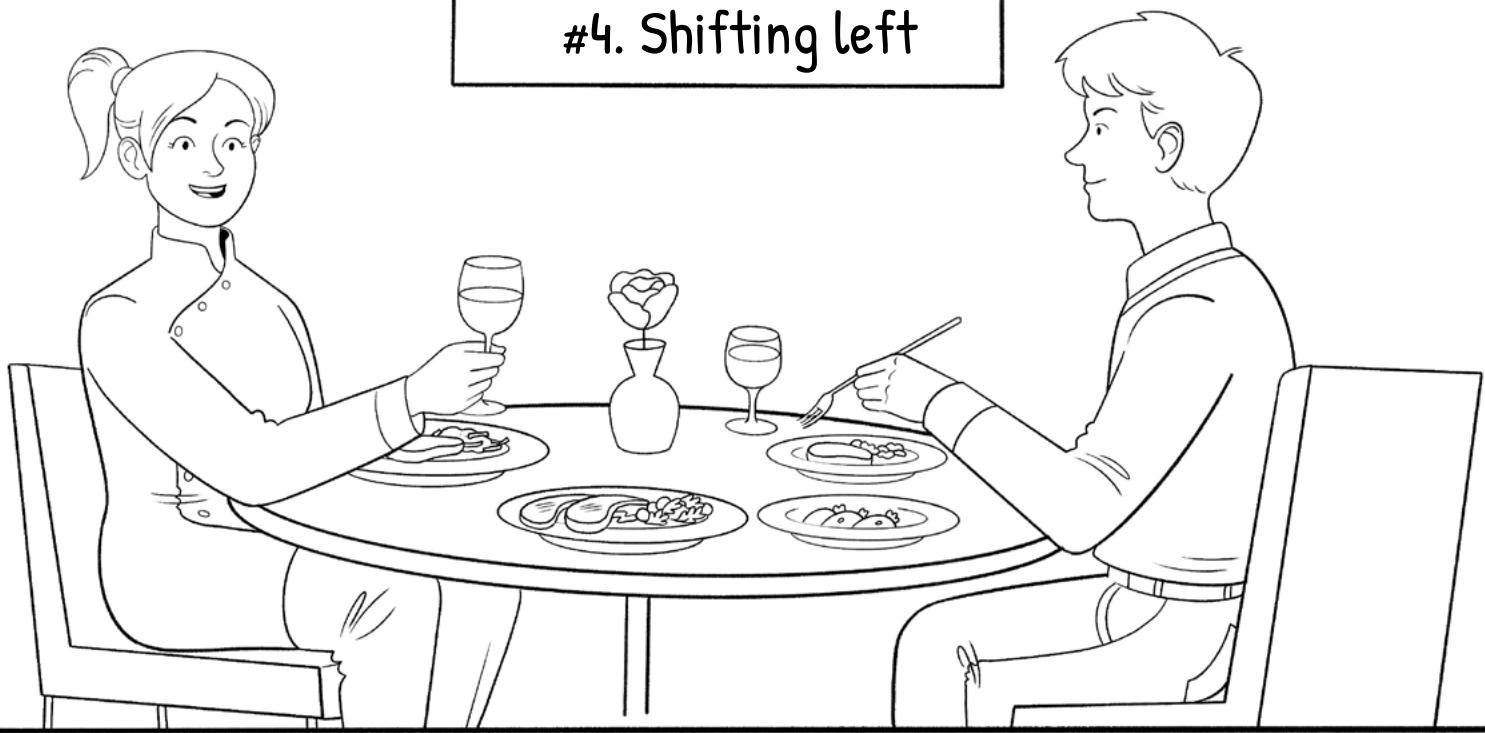
Out of 200 guests, four sent their steaks back—and only one was outside of the error budget for doneness. Chef's steak SLO was met for the event!

### #3. Scalability



When the dance floor opened up, the bar was overwhelmed with drink orders. Leo took waitstaff off of dessert service (everyone was dancing anyway) to help auto-scale the bar. Once the bar quieted down, those waitstaff continued serving dessert.

### #4. Shifting left



Leo and Chef did a dry-run of the reception menu and improved the wine pairing and dessert options based on the experience—both were a big hit at the event!

## #5. Zero downtime



The reception ran six hours—well past midnight. From monthly staff meetings and regular communication, Leo knew Annie had to leave at 11. Leo started taking over her guests and by 11 she was all set to go home.



Mia, Leo's sister, was thrilled, and the entire family had a blast! Gino's, the family restaurant, was saved, and became a much greater success. Leo's dad would have been proud!

## Observability

1..... 2..... 3..... 4..... 5

Never tried to get metrics out of it

Regularly enable new metrics for SRE team working closely with service development and product teams

## Safety and self-healing

1..... 2..... 3..... 4..... 5

My codebase OOM kills routinely, oops.

My codebase checks for common preventable failures and avoids them.

## Scalability

1..... 2..... 3..... 4..... 5

I have no idea of the capacity limits of my codebase in production.

My codebase's default deployment includes capacity-limit based auto-scaling and load shedding.

## Shifting left

1..... 2..... 3..... 4..... 5

I've never run my codebase in Production before.

I work in a DevOps team that is regularly involved in production deployment and operations, and am directly responsible for its reliability.

## Zero downtime upgrades

1..... 2..... 3..... 4..... 5

Upgrading my codebase requires a weeks-long consulting engagement.

My codebase can gracefully handoff connections to a new upgraded version of my application and take those clients back in case of failure.

Check your score on the next page!

So what's the operability of your codebase? For each of the five challenges...





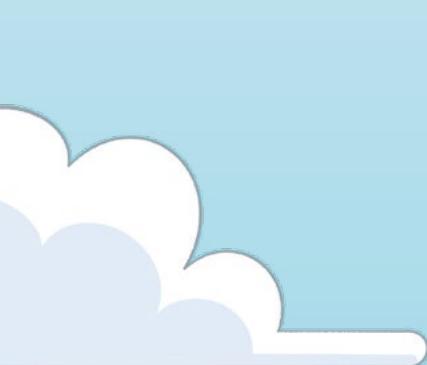
How many points did you get?  
Add them up, let's see  
where you are!

0-10 ..... beginner  
11-20 ..... advanced  
21+ ..... expert



# **Stable, Secure, Performant, and Boring**

[red.ht/sre-coloring-book](http://red.ht/sre-coloring-book)



Sponsored by Red Hat