

BLAMELESS

# The Comprehensive Guide on SLIs, SLOs, and Error Budgets



# TABLE OF CONTENTS

---

The Speed/Reliability Compromise.....	4
What’s the difference between SLIs, SLOs, and SLAs?.....	6
Creating SLIs that match your users’ journey.....	8
Culture matters when adopting SLOs.....	9
Why it’s human to blame	
Learning and improving collaboratively	
Stakeholders for SLO adoption: how to get everyone on board.....	11
The resistance	
Emotional appeal	
Logical appeal	
Creating your SLOs.....	15
Roadblocks to SLO adoption	
Determining error budgets and error budget policies.....	16
Error budgeting basics	
Error budget policies	
Building a long-term process for operationalizing SLOs.....	19
Advanced SLO practices.....	20
SLOs: two case studies.....	21
How Blameless adopted SLOs.....	23
How SLOs support the SRE lifecycle.....	25
The Blameless difference: agnostic & collaboration-driven.....	26

SRE, or site reliability engineering, is one of the fastest-growing roles and rising trends shaping software teams. Job openings grew **72% YoY** for SRE positions. Organizations are adopting best practices for incident management, blameless retrospectives, automation, runbooks, and more. One of the most fundamental principles of SRE are SLOs (service level objectives), as they provide a paradigm shift in truly measuring and enforcing reliability. However, successfully implementing and driving long-term adoption of SLOs is far easier said than done, because it requires a real commitment to culture and process transformation. In this guide, we will cover everything you need to know about SLOs, SLIs, error budgets, and more.

There are major 3 benefits to these key pillars of SRE:

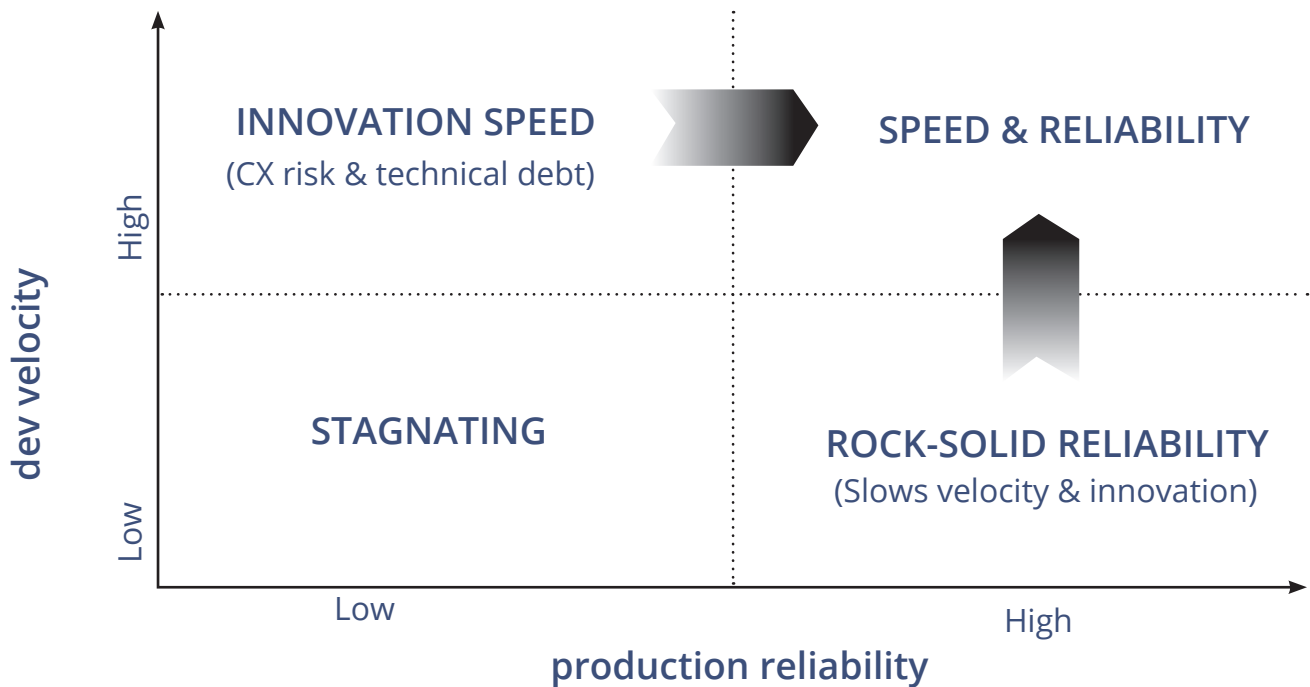
- 1 Keep customers happy:** Your customers are your No.1 priority, but metrics like churn rates and NPS scores are lagging indicators that only signal when something is wrong once it's too late. SLIs and SLOs help you proactively prevent churn and increase customer satisfaction.
- 2 Keep teams aligned:** Siloed organizations can make it difficult for everyone to get aligned on core business goals. With error budgets and error budget policies in place, teams communicate more effectively, have a common basis for decision-making, and can align priorities and incentives to encourage collaboration.
- 3 Balance reliability and innovation velocity:** SLOs and error budgets ensure that all teams, from product to SRE to dev, have a common language and can balance customer expectations for innovation as well as reliability and quality.

# The Speed-Reliability Compromise

When it comes to building and delivering modern software and applications, both enterprises and cloud-native organizations are faced with unprecedented system complexity. There's architectural complexity with some companies using hybrid architectures and different operating models. There's a desire to be on premise as well as leverage cloud infrastructure (including multi-cloud) and platform services. There's complexity in how these systems interact with each other, in the collaboration between distributed and siloed teams, and in the processes meant to govern systems and teams.

How do you extract the right context from the volumes of data to make meaningful business decisions?

Expectations from your users around new features, reliability, availability, security, and quality are increasing exponentially. However, organizations typically must make tradeoffs and aren't set up to deliver on all these vectors. We call this the **speed-reliability compromise**.



On one axis, there's the desire and need to have rock-solid reliability. But the challenge is, if that's what you're optimizing for, you're likely not innovating to your full potential. On the other hand, you also can't spend all your time pushing on just feature delivery without regard to stability, as you'll rapidly accrue risk and technical debt.

Finding the balance is at the center of delivering great customer experiences, and critical to organizational survival as the dominance of digital increases the stakes of competition. To attain this balance, you'll want to focus on:



**Proactive remediation:** Working within error budgets can eliminate unnecessary risks and give your teams a threshold for reliability when pushing new features. By working to remain within your error budget and monitoring depletion over a rolling window, you're less likely to have incidents that will lead to unhappy customers or potential churn.



**Accountability:** With SLOs in place, there will be shared ownership between dev, ops, and product. All stakeholders will have the same incentive—customer happiness. When the incentive is the same, there's fewer points of contention between teams.



**Focused action:** Blameless Staff SRE Amy Tobey uses this analogy: "You have climatology, which is the study of atmospheric and weather patterns over time. Then you have meteorology which is the day-to-day measure. Real-time monitoring is like meteorology and SLOs are like climatology because they are an early warning indicator that there is a shift about to happen." When you have this early indication, it allows you to focus and prioritize your efforts in the right area.

Achieving balance between innovation and reliability is more difficult than ever due to rising complexity, but simultaneously more important than ever as customer expectations only rise. For this guide, we'll use XYZ Corp, a fictional company as an example of an organization striving to break the speed-reliability compromise. Here's a quick snapshot of XYZ Corp:

- ▶ **Industry:** Transportation SaaS
- ▶ **Product:** Ridesharing platform
- ▶ **Business Goals:** Rapidly out-innovate competition and increase ridership by x % across the globe, while protecting platform stability and keeping user churn to < x %
- ▶ **SRE maturity level:** Mid— implemented SRE best practices concerning incident resolution and incident retrospectives, but working to adopt SLOs

XYZ Corp is beginning a reliability journey with us, and we'll walk you through how this organization operationalizes SLIs, SLOs, and error budgets. First, let's define what these concepts mean.

# What's the difference between SLIs, SLOs, and SLAs?

Below are the definitions for each of these terms, as well as a brief description (we'll go over each more in-depth later). Definitions are according to the [Google SRE Handbook](#).

**SLI:**

“ A carefully defined quantitative measure of some aspect of the level of service that is provided. ”

SLIs are a quantitative measure, typically provided through your APM platform. Traditionally, these refer to either latency or availability, which are defined as response times, including queue/wait time, in milliseconds. A collection of SLIs, or composite SLIs, are a group of SLIs attributed to a larger SLO. These indicators are points on a digital user journey that contribute to customer experience and satisfaction.

When a developer sets up SLIs measuring their service, they do them in two stages:

- 1 SLIs that will directly impact the customer.
- 2 SLIs that directly influence the health and the availability or the latency and performance of certain services.

Once you have SLIs set up, you connect them to your SLOs, which are targets against your SLI.

**SLO:**

“ A target value or range of values for a service level that is measured by an SLI. A natural structure for SLOs is thus  $SLI \leq \text{target}$ , or  $\text{lower bound} \leq SLI \leq \text{upper bound}$ . ”

Service level objectives become the common language for cross-functional teams to set guardrails and incentives to drive high levels of service reliability. Today, many companies operate in a constantly reactive mode. They're reacting to NPS scores, churn, or incidents. This is an expensive, unsustainable use of time, and resources, let alone the potentially irrecoverable damage to customer satisfaction and the business. SLOs give you the objective language and measure of how to prioritize reliability work for proactive service health.

## SLAs:

“ An explicit or implicit contract with your users that includes consequences of meeting (or missing) the SLOs they contain. ”

Service level agreements are set by the business rather than engineers, SREs, or ops. When anything happens to an SLO, typically your SLA will kick in; they're the actions that are taken when your SLO fails and often result in financial or contractual consequences.

## Key SRE Metrics: SLIs, SLOs, SLAs

Empower software teams to better understand customer experience

Service Level Indicators (SLIs)	Service Level Objectives (SLOs)	Service Level Agreements (SLAs)
<p>A quantitative measure of some aspect of the service level. Typically:</p> <ul style="list-style-type: none"><li>• <b>Latency or Availability</b>- Response time, including queue/wait time, in milliseconds</li><li>• <b>Error Rates</b>- Error rate, in requests/sec</li><li>• <b>Rate</b>- Request rate, in requests/sec</li><li>• <b>Utilization</b>- How busy is the resource or system</li></ul>	<p>A target value or range of values for a service level that is measured by an SLI (or multiple SLIs):</p> <ul style="list-style-type: none"><li>• <math>SLI \leq \text{Target}</math></li><li>• <math>\text{Lower bound} \leq SLI \leq \text{upper bound}</math></li></ul>	<p>Contract with customers that includes consequences for meeting (or missing) the SLOs contained in agreements</p> <p>Essentially, it is an explicit or external-facing consequence of not meeting the SLO.</p>

SLA violations are one of the reasons that XYZ Corp is looking to adopt SLOs. XYZ Corp breaches its SLA for availability almost every month. As it onboards more customers with SLAs, these expenses can grow if it doesn't meet its performance guarantees.

NPS scores indicate lower customer satisfaction over the last two quarters, but unfortunately they are a lagging indicator with respect to customers that have already begun to churn. The team met to discuss what needs to be done. The first step to this is breaking down the company's SLIs.

# Creating SLIs that match your users' journey

---

It's crucial to create **SLIs that directly affect customer happiness**, so we refer to our user journeys for insight and inspiration. It's important to note that a single SLI cannot capture the entire user journey alone. A typical user of your service might care about the latency of the site's response, the availability of key functions, and the liveness of data they're accessing. Their happiness with the service during this journey depends on all three, but there's no way to monitor them as one. In order for your SLO to be a functional objective, your SLI must be a singular metric captured by the service's monitorable data.

At the same time, creating SLIs for every possible metric is just as troublesome. As the Google SRE Handbook states, "You shouldn't use every metric you can track in your monitoring system as an SLI; an understanding of what your users want from the system will inform the judicious selection of a few indicators. Choosing too many indicators makes it hard to pay the right level of attention to the indicators that matter." There are nearly endless subsets of metrics you can consolidate into your SLI. Understanding the perspective of users can help you choose.

For example: Let's say that a user's channel involves making a dozen requests to the same service component – like clicking through many pages of search results. Separately, these requests return faster than the SLO set for them, maybe under a second, and a user looking at just one or two pages will be satisfied with this speed. However, if your user journey involves looking through twenty pages, the annoyance of nearly a second wait repeated twenty times could be intolerable. Only with both relevant monitoring data and broader perspectives could you discover this point of user frustration.



Finding these pain points along the user journey could lead to a radical redesign of the service as a whole. Additionally, it opens up a path to solutions deep in the backend and helps determine priorities for development. In our example above, you could either redesign the catalog to avoid the need to look through twenty pages, or you could optimize the components serving those pages until the total delay for twenty pages is still acceptable.

XYZ Corp knows it needs to examine availability and set SLOs for it; the CSM team hears the customer complaints. However, after looking at their user's journey, the team determines that while the individual pages of each tab on the receipts feature don't load slowly individually, when someone needs to skim through multiple pages, it becomes tedious. So the team also decides to create an SLO for latency as well.

Now that XYZ Corp has determined the points of the user journey most critical to overall user happiness, the team can begin looking at building SLOs. However, there are some important things to know before diving in.



## Culture matters when adopting SLOs

Small to large, if you can only do the 20%, the 20% matters. You don't need 100 SLOs, or a number of SREs to match Google or Netflix. Start with 2-3 SLOs, iterate, and work your way up. The [Twitter SRE team](#) was able to start small with its SLO adoption, evangelizing SLO usefulness and implementing SLOs slowly at first to demonstrate success. This team-led growth is an important aspect of the cultural changes required for SLO adoption.

## Why It's Human to Blame

In 2005, a [major explosion](#) at BP's refinery in Texas City killed 15 people and injured 180 others.

The vice president publicly blamed their staff, stating that "if [our people] followed the start-up procedures, we wouldn't have had this accident." However, when analyses found the explosion to be "years in the making" due to substandard equipment and inadequate safety procedures, the blame was then placed on management officials for choosing to operate under flawed conditions. The safety conditions did not improve after this incident, and more incidents occurred after the initial explosion, resulting in over \$100 million in fines, dozens of lawsuits, and payouts to victims of up to \$1.6 billion. A decade later, fires continue to ravage refineries on a weekly basis and data shows that at least [58 people have died](#) in U.S. refineries since 2005.

What does this story tell us about the nature of blame? For one, it highlights our tendency as a species to focus on human fallibility as the reason for failure. This is a common theme prevalent across industries, particularly in aviation safety and healthcare. In fact, the U.S. National Transportation Safety Board lists "people" as the probable cause in [96% of accidents](#).

Psychologists term this the [Fundamental Attribution Error](#) — the belief that individuals, not situations, cause error. For example, if an incident takes a site down and we discover during the post-mortem that the engineer in charge could have prevented the incident, we tend to judge the engineer as sloppy or neglectful instead of considering other situational factors that could have contributed to the crash (such as an understaffed team or a lack of standardization of development practices).

While it is a deeply unsettling thought, it also speaks to a wider culture of blame. We need to overcome this in order to successfully adopt SLOs and keep them operational.

## Learning and improving collaboratively




[Failure will happen](#), incidents will occur, and SLOs will be breached. These things may be difficult to face, but part of adopting SRE is to acknowledge that they are the norm. Systems are made by humans, and humans are imperfect. What's important is learning from these failures and celebrating the opportunity to grow.

Fear is an innovation killer, but failure is an innovation inspiration. Creating safety and trust within your organization is key to fully realizing and unleashing your team’s potential. Work together to make your SLO adoption process better. This will take time, and all hands on deck. Speaking of all hands on deck, let’s talk about the stakeholders you’ll need for implementation.

# Stakeholders for SLO adoption: how to get everyone on board

Organizations comprise people with competing priorities. Operations doesn’t want to push features if that means the pager buzzing all weekend. Developers don’t want to halt releases because operations say so. Sales and marketing want new features to sell and promote. So, how can you prioritize all these needs? SLOs can help with this, too.

## Why SLOs: Align Competing Priorities Around Customer Experience

	 <b>Product &amp; Eng</b>	 <b>Operations</b>	 <b>Business</b>
<b>Focus</b>	Prioritizing & Building Functionality	Minimizing Business Risk	Protecting Customer Experience and \$\$\$
<b>Metrics</b>	Feature Velocity CSAT & Churn	Availability, Latency, Throughput, etc.	SLA Adherence Customer Impact
<b>SLO Value</b>	Data-driven way to focus scarce engineering resources	Help dev teams increase velocity without sacrificing quality	Show leading instead of lagging indicators of CX to Board, key stakeholders

## SLOs can help three core stakeholders in the following ways:



**Product and engineering:** SLOs provide a data-driven way to focus resources. With the do-more-with-less mentality, teams must pick the most crucial features to spend engineering time on. SLOs help align those priorities using metrics like CSAT and churn.



**Operations:** Operations wants to limit business risk. SLOs help communication with developers to achieve both innovation velocity and quality. This team will look at SLO metrics like availability, latency, and throughput.



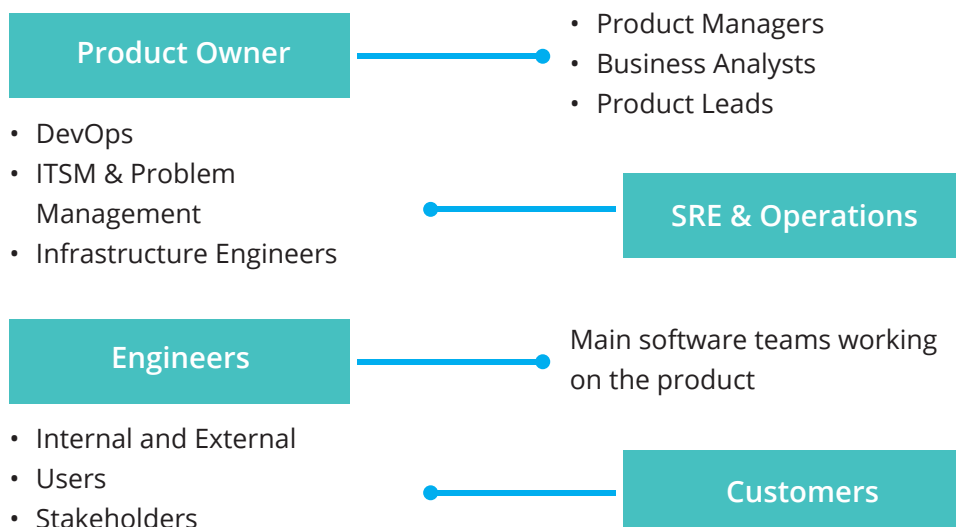
**Business:** These efforts are important, but if the business can't attract and keep customers, it won't last. SLOs help this team by providing leading indicators of customer happiness.

With SLOs as a unifying tool, organizations can make decisions that take all needs into account. It's important to identify all the key stakeholders for SLOs early in this process.

SLOs can generally start at the grassroots level. However, it requires a lot of orchestration between a number of different teams for long-term operationalization. This requires buy-in from people at all levels and within all teams to make SLOs successful within an organization.

## Key Stakeholders in Defining SLOs

Designing SLO is a collaborative process requiring input from multiple stakeholders



Initiatives like SLOs can initially be difficult to get buy-in for, just as any investment in a new process may be hard to justify in context of long to-do lists and urgent priorities such as new product delivery. In order to get engineering leadership on board, you'll need to **communicate the business impact of SLOs** and use empathy to overcome objections and resistance. Three of the biggest incentives to adopting SLOs are:

- ▶ **Reduced cognitive and manual toil:** No more digging through metrics or shooting in the dark to understand where to prioritize limited engineering resources. It's easy to be data rich but information poor, but SLOs ensure you have the right level of context for effective decision-making.
- ▶ **Ability to surface technical debt:** Technical debt directly related to reliability, or the lack thereof, can often be swept under the rug. SLOs give you insight to where technical debt is accumulating at unsustainable rates. On the flip side, if you aren't consuming your whole error budget, you can push harder on innovation. SLOs also provide a common language to discuss important efforts in refactoring and tech debt management.
- ▶ **Getting ahead of incidents:** Your customers' experience depends on the stability of your platform and the quality of your product. With SLOs, your teams will spend less time on reactive work, as they'll not only be better positioned to manage tech debt but also better anticipate threats to the customer experience.



Once you've outlined some of the most important business impacts of SLOs, it's important to frame an argument to support this. SLOs can be difficult to implement, and results are not immediate. You'll need to anticipate the resistance leadership might feel and combat it with emotional and logical appeals.

## The resistance

What this will come down to is company priorities. C-level executives and leadership might not see the link between business performance and reliability, as often, incentives are aligned toward new product innovation. Therefore, it may be difficult to convince them that SLOs should be a company-level priority.

## Emotional appeal

Here, we lean heavily on customer impact. Leadership cares about whether or not customers are satisfied with their service. Satisfied customers cultivate pride, while dissatisfied customers create fear.

Additionally, there is a significant financial aspect involved. Without SLOs, organizations would have direct customer impact via SLA losses. That can be very expensive and hurtful to the brand and customer trust. If the reliability issues are too disruptive to overlook, customers will begin to churn. The data you can collect from the cost of downtime can indicate how reliability affects your brand value.

**To prove to leadership that SLOs are crucial, you can do two things:**

- 1 Quantify the cost of downtime (e.g. measure SLA losses) and estimate a bottom line for reliability impact.
- 2 Show them your organization's NPS (or net promoter score) for an indicator of brand and customer sentiment, alongside a detailed customer satisfaction survey, to correlate the score with reliability.

Once you've laid down this emotional appeal, you can move on to the logic.

## Logical appeal

The first logical appeal you can present is the need for a competitive advantage. When you share similar services as your competition, you look like a less viable option when a competitor is able to respond to, recover from, and prevent incidents better than you. SLOs are an important lever to understand your product and customer experience, so you can be the best and stay ahead of the competition. Show your leadership the metrics on the SLOs and explain how they are set to optimize performance of most important paths in the user's journey. Also consider bringing to the table the amount of data and access points in the cloud, and the number of services the company depends on. This can demonstrate the need for a system that can adapt to the complexity of cloud-native and microservice environments, and distributed systems.

The second logical appeal is that you always want to spend more time on planned versus unplanned work, and empower teams to focus on efforts that are more strategic and business-differentiating. SLOs and the use of error budgets help us move from a reactive mode (knowing that incidents will occur but not where and why), to a proactive mode of anticipating areas of risk and failure. Error budgets with negotiated terms between the business and engineering teams allow teams to automatically respond in the right way, by standardizing actions and protocols. This saves money, time, and resources.

## Creating your SLOs



XYZ Corp has its SLIs determined, the culture part down, and the whole team on board; now the team just needs to actually set up the SLOs.

XYZ corp is looking at availability of the site, measuring minutes down for customers per month. Based on traffic levels, customer usage, and NPS scores, the team has determined that its customers are likely to be happy with 99.5% availability. On the other hand, customer satisfaction and usage doesn't seem to increase during months where uptime is greater than 99.9%. This means that there's no reason to optimize at this point for higher than a 99.5% uptime metric. Easy enough, right?

However, the latency issue is slightly more complicated as 2 SLIs are used to set this. The first is load time per page, the second is the amount of requests pushed to load balancing servers. The SLO will need to incorporate both to be an accurate target value.

## Roadblocks to SLO adoption

XYZ Corp was able to set SLOs without any major roadblocks, but that's not the norm. Many teams will face issues during their SLO journey. We've identified 3 of the most common issues and how to avoid them or address them as they occur.

**Siloed data:** Data lives across multiple observability vendors and sources, making visibility a challenge. Many of us work on a single product. An engineer might use Splunk or Prometheus or a host of other observability tools, an SRE might use service mesh. The goal with SLOs is to try to tie all of that data together and provide you insights into exactly how your application is performing to customer satisfaction. This could require you to collaborate with other teams to create a service architecture map which denotes which observability and other vendors each service or team uses, to centralize important context.

**Lack of process & alignment:** As we discussed above, when set up in a silo without a collaborative process, SLOs often fail. They end up being just a metric sitting in a dashboard. Aligning all stakeholders prior to setting SLOs is the best way to ensure that the process is streamlined and that adoption follows through.

**Treating it as one and done:** SLOs will need to be continuously re-validated until they're truly actionable and quantify the customer experience. When you create SLOs, there needs to be a process to operationalize them, whether it's a weekly meeting or something scheduled into your sprint life cycle. As your product grows and changes, so will your users' needs and expectations. You'll need to adapt and adjust your SLOs accordingly.

## Determining error budgets and error budget policies



Once your SLOs are set, you need to know what to do with them. If they're just metrics that you're occasionally paged for, they'll quickly become obsolete. One way to make sure your SLOs stay relevant is by determining error budgets and error budget policies.



## Error budgeting basics

An error budget is the percentage of remaining wiggle room you have in terms of your SLO. Generally, you'll institute a rolling window versus historical purview into your data. This keeps that SLO fresh and constantly moving forward as something that you can monitor. Error budget can be shown as the below calculation:

## Deeper Dive into Error Budgets

Understanding when to prioritize reliability over feature development

*Error budgets enable teams to innovate while maintaining reliability.*

*For every SLO, there is an error budget.*

*As a team's error budget becomes depleted, an error budget policy codifies how a team will respond.*

$$SLI = \left[ \frac{\text{Good events}}{\text{Valid events}} \right] \times 100\%$$

**SLO**  
99.9%

is equal to.....

**Error Budget**  
.1%

$$.001 \times \frac{30 \text{ days}}{\text{Month}} \times \frac{24 \text{ Hours}}{\text{Day}} \times \frac{60 \text{ Mins}}{\text{Hours}} = \frac{43 \text{ Mins}}{\text{Months}}$$

## Error budget policies

It's not enough to know what your error budget is; you also need to know what you'll do in the event of error budget violations. You can do this through an error budget policy, which determines alerting thresholds and actions to take to ensure that error budget depletion is being addressed accordingly. It will also denote things like escalation policies as well as the point at which SRE or ops should hand the pager back to the developer if reliability standards are not being met.

## Alerting

Alert fatigue, or pager fatigue, is something that can drastically reduce even the most seasoned team's ability to respond to incidents. This is the effect of receiving too many alerts, either because there are simply too many incidents occurring, or because your monitoring is picking up on insignificant issues and notifying you for things that do not require your attention (also known as alert noise). This can lower your team's cognitive ability and capacity, making incident response a slow, difficult process. It can also lead your team to ignore crucial alerts, resulting in major incidents going unresolved or unnoticed until it's too late.

You'll want to make sure that your alerting isn't letting you know every time a small portion of your error budget is being eaten; after all, this will happen consistently through the rolling window. Instead, you'll want to make sure that alerts are meaningful to you and your team, and that they are indicative of actions you need to take. That's why many teams care more about getting notified on error budget burndown rate over a specific time interval, compared to the depletion percentages themselves (i.e. 25% vs. 50% vs. 75%). One way to determine if action needs to be taken for error budget burn is to write in stipulations, which could look something like this: if error budget % burned  $\leq$  % of rolling window elapsed, no alerting is necessary. After all, a 90% burn for error budget isn't really too concerning if you only have 3 hours left in your window and no code pushes.

However, if burn is occurring faster than time elapsing, you'll need to know what to do. Who needs to be notified? At what point does feature work need to be halted to work on reliability? Who should own the product and be on-call for it at this point? Answers to questions like these should be baked into your error budget policy. [Google produced an example](#) of what this document looks like. It contains information on:

- ▶ Service overview
- ▶ Policy goals
- ▶ Policy non-goals
- ▶ SLO miss, outage, and escalation policies
- ▶ Any necessary background information

## Handing back the pager

In the example policy above, Google reminds us, “This policy is not intended to serve as a punishment for missing SLOs. Halting change is undesirable; this policy gives teams permission to focus exclusively on reliability when data indicates that reliability is more important than other product features.” If a certain level of reliability is not being met and the product is unable to remain within the error budget over a determined period of time, SRE or operations can hand back the pager to the developers.

This is not a punishment; it’s simply a way to keep dev, SREs and ops all on the same page, and help shift quality left into the software lifecycle by incentivizing developer accountability. Quality matters. Developers are held to task for their code. If it’s not up to par, feature work will halt, reliability work will take center stage, and the pager will be handed over to those who write the code. This also helps protect SRE and ops from experiencing pager fatigue or spending all their time on reactive work. Error budget policies are an efficient way to keep everyone aligned on what matters most, which is their SLOs and ultimately customer happiness.

# Building a long-term process for operationalizing SLOs

The process that goes into creating SLOs, especially the people aspect, is extremely critical for consistency and ability to scale it across your entire organization. To operationalize SLOs, you’ll need to remember a few key things:

- ▶ You’re not going to get it right the first time and that’s okay: You need to have an iterative mindset to get the correct SLOs, thresholds, and teams in place. Patience and persistence are important.
- ▶ Review your SLOs on a weekly or bi-weekly cadence: Many of you probably have internal operational review meetings where you look at your key reliability metrics such as the number of incidents, incident retrospective completion, follow-up action item status, and customer-reported issues. In that meeting, one of the key things you’ll want to make time for reviewing is your SLO dashboard.
- ▶ Review critical upcoming initiatives collaboratively: Determine if any planned updates or pushes are likely to exceed your error budget and plan sprints accordingly to prevent this. Are you shipping as safely as possible? Attendees in this meeting should be from product, SRE, the core component/service engineering teams, and other stakeholders.

Once you've got these basics down, you can begin to expand your SLO practices.

## Advanced SLO practices

Here are some additional, more advanced SLO practices that you can start using once you've found success with the basics:

- ▶ **Composite SLOs:** Combine two or more SLOs from multiple different services to represent an end-to-end product view of overall reliability performance. This could include an SLO containing both availability and latency thresholds, something XYZ Corp might want to investigate in the future.
- ▶ **Treating SLO violations as incidents:** How do you effectively treat a violation as an incident, and thread that into your incident management process? When we violate our SLO, we are acutely affecting our users and customers. Those issues must be treated as incidents, so be sure to define the right [severity levels](#) for SLO breaches.
- ▶ **Giving back error budget:** You may have maintenance windows or services that must be unavailable in certain time periods. That may be perfectly normal, and will automatically consume the error budget. You can give that error budget back, but make sure you document the reason why this was consumed.
- ▶ **Correlating changes to SLO:** SLOs are not like diamonds; they're not going to be there forever. Ask yourself, "Are these still valid?" Your organization, your teams, and your product are constantly evolving and changing. Why should your SLOs be static?

Maybe you'll be ready to take these advanced steps in a few months. Maybe it will take a few years. No organization's SLO journey looks the same. Below we have a case study about how a ride sharing organization's SLO adoption progressed, as well as an insight on how we implemented SLOs at Blameless.

Additionally, you can look at this aggregated [list of resources](#) under the SLO section of our Industry Leader Insights page.

# SLOs: case studies

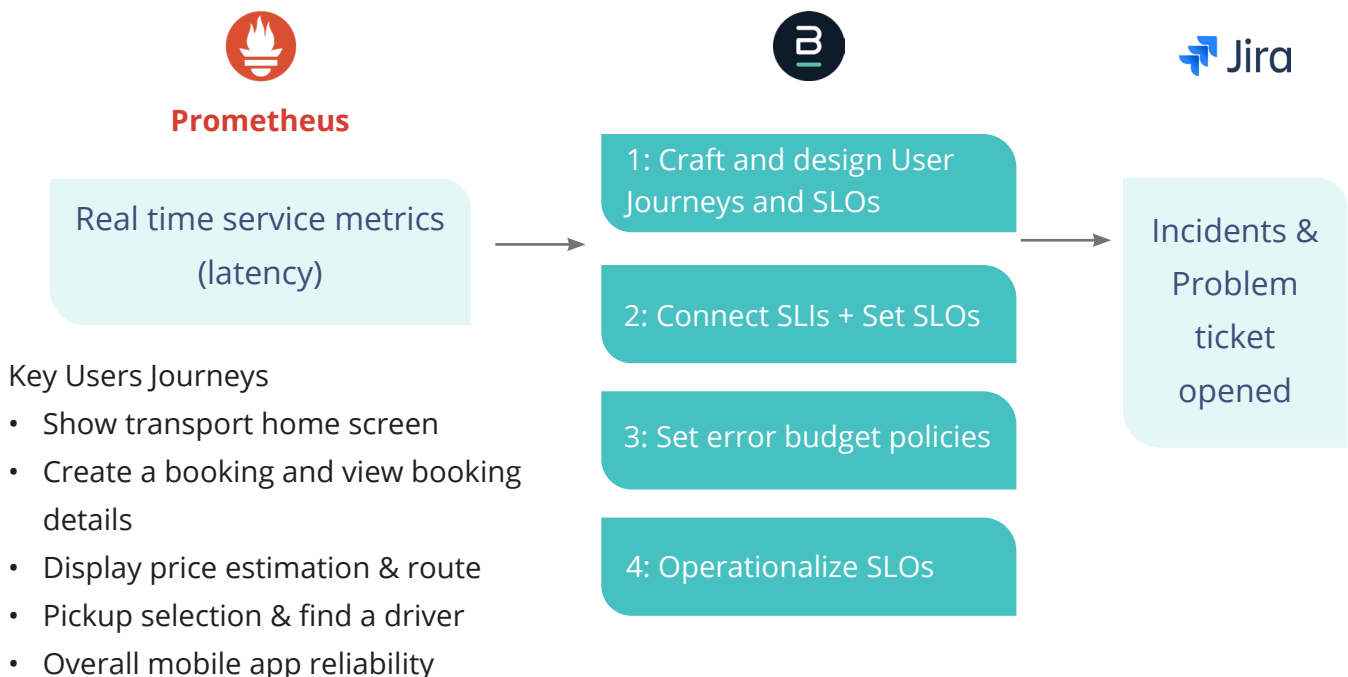


One of our customers — a global ride hailing company just like XYZ Corp — came to us with a problem: their transportation homescreen was failing to delight their users. If they weren't able to improve their home screen, or improved too slowly, it immediately affected their revenue. We worked with them to help fix this issue.

First, we helped connect all their SLIs through their services, helping them define the most crucial places through the user's journey where improvements could be made. Next we set SLOs, providing a range of acceptable behaviors for these points of the user journey. Then we worked with them to set error budget policies and understand exactly what to measure and what actions to take once error budgets have been exhausted. With this, they were able to prioritize product improvements and fix those problem spots users encountered. Below is a diagram showing the process of SLO implementation:

## Global Ride Hailing Company Stages of SLO Implementation

Using SLOs and Error Budget policies to optimize experience for critical user journeys



## The benefits of SLO according to Twitter

Organizations like [Twitter](#) also benefited from setting SLOs. While the Twitter engineering team had laid a very strong foundation around observability and reliability, it took several important breakthroughs before SLOs began achieving broader adoption within the organization and the journey continues. Team is still early in its adoption of SLOs, yet they've already seen the immense potential and value of SLOs in several ways.

### From a 'distributed service zoo' to a shared language

Twitter has hundreds, if not thousands, of services, making its infrastructure a complex beast to understand. The current members of the Twitter Command Center (TCC) have been around long enough where they generally know what most of the services are and how services 'snap together'. However, they know that eventually they will reach a point where that becomes impossible, where no one individual can grok how it all works. By investing in SLOs now to help guide discussions, the goal is that by the time they reach that point of un-knowable complexity, they will have set themselves up to manage service metrics programmatically.

### The right amount of context

Context is the key. Dashboards can easily have hundreds of charts which translate into thousands of metrics. Teams might have tens or hundreds of alerts on their services across multiple data centers. These dashboards, metrics, and alerts are helpful for those running those services, but they're very high context, and information overload for anyone else.

SLOs create the ability to have more directed conversations with shared context. Instead of looking at a hundred pictures of a dashboard, the team can align on the four or five things that matter. If any of those are not green, others can understand that something's not right without having to know anything else about the service.

### Dynamic load balancing and load shedding

By making SLOs a first class entity, services can speak it at the programming level, beyond just measuring it. This enables the team to make systematic improvements using SLOs as a building block. For example, the team is exploring whether back pressure in Finagle can instead be SLO-based.

With Finagle, services can programmatically detect when they are under load (typically with second class signals such as CPU), and then signal to redirect traffic to another instance. Instead of relying on second class signals to implement back pressure, a service can directly know if it's trending towards an SLO violation in order to signal back pressure and reduce load on itself.

## Graceful degradation

One of the Twitter team's goals for SLO is in gracefully degrading services during large-scale events to ensure that core functionality is always available. Rather than an all-or-nothing failure mode, the team aims to gracefully degrade services by stripping away peripheral features while maintaining core functionality.

The Twitter team is interested in utilizing SLOs to implement a selective circuit breaker pattern to improve overall system reliability. Service owners can decide what upstream services are necessary for core functionality, and which are only necessary for add-ons or bells and whistles. An upstream service that is not very important to one service could be critical for others. A consuming service can implement a circuit breaker to detect and stop sending traffic to services experiencing high error rates.

Blameless has our own success story with SLO adoption.



# Blameless: our own SLO journey

SLO adoption is a long process, and positive results are often a delayed benefit. You can look to [other companies](#) for inspiration; the ones who stick with the program and see SLOs through to operationalization reap the rewards. For an example, we'll share [our own story](#) about how we implemented SLOs.

In full candor and transparency, our journey began with failure. One major reason for this failure was that we were tracking NPS and churn as key business metrics. These are lagging indicators, not leading indicators. By the time we noticed a customer was unhappy, it was already too late. We needed something beyond NPS and churn to describe to us how our customers were feeling, so we began the exercise of setting up SLOs as a company.

We started defining our SLOs and user journeys so that everybody, even at the highest levels within the leadership team, could agree to and understand why we set certain KPIs and monitor certain metrics. We conveyed the importance of SLOs as a strong and early indicator of our users' happiness.

## Key changes due to this new way of thinking included:

- ▶ Setting up our own SLOs with Blameless using integrations like Prometheus, and watching our dashboard
- ▶ Weekly operational review meetings where we looked at key user journeys, the associated SLOs, and the SLO statuses.
- ▶ Setting error budget policies for those SLOs and tracking them.
- ▶ Getting buy-in from respective component owners to commit to changing their sprints if we violated our error budget.
- ▶ Mandating that any regression from customer expectations would be considered a high severity incident requiring immediate attention.

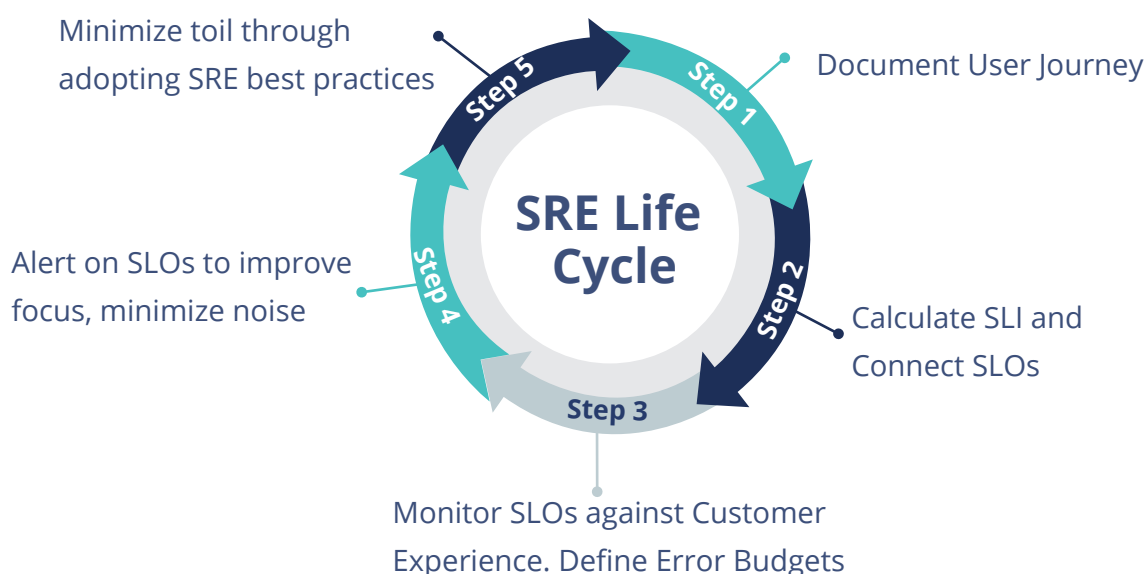
After a while, the Board could see that we were putting our money where our mouth is in prioritizing customer experience. Through SLOs, we were able to detect potentially customer-impacting issues before they became customer facing incidents. Most importantly, SLOs became a shared contextual framework we could leverage with our Board Members. Just like NPS or ARR, the SLO became a metric that everybody understands.



We found success with SLOs, as did our ride hailing customer, and other organizations such as Twitter and Evernote, though we all implemented them for slightly different reasons. This is because SLOs are dynamic. SLOs can change company by company, product by product, and even day by day. The key is that SLOs work to support your SRE lifecycle and improve both customer and engineering team satisfaction as a whole.

## How SLOs support the SRE lifecycle

### Components of SLOs



We talked about the components of an SLO, but how does that fit into the bigger SRE picture? The focus of SRE is to ensure user happiness. In this way, SLOs are the focal point of any SRE practice. SLOs are the motivation for documenting the user journey. User journeys are a bucket for collecting key paths within your system that a user will frequently interact with (i.e. a daily basis). Something that integral to the performance of your application is essential to monitor. When we look at metric thresholds and calculate SLIs and SLOs, we connect everything together and we tie it back to the user journey.

When we monitor SLOs, our system begins to aggregate the data, and ties that back into error budgets which determines the allowable amount of system downtime or latency over a specific timeframe. These are a guideline for what metrics we should maintain in order to have positive customer experiences. Once SLOs and error budgets are established, we define alerting and

actionability. SLOs need to have some action taken upon them in the event of a breach or near-breach. Error budget policies don't need to be set in stone, but they govern the steps that should be taken in order to remedy the situation, linking SLO best practices to your incident management process.

Those learnings from the triggered incident are discovered through your incident retrospective process and are baked back into your alerting, user journeys, and SLOs, making your system even more resilient through a process of continuous learning.

## The Blameless difference: flexible & collaboration-driven

---

Blameless has a vendor-agnostic approach to ingesting data from multiple observability vendors, to provide that maximum visibility and flexibility when you're setting up your SLOs. We integrate with Datadog, New Relic, Prometheus, Pingdom, and API. We also tie in with infra provisioning and change events, alerting, chatops, and a whole host of tools so you can simplify complexity across the software lifecycle. This agnostic approach enables you to drive process repeatability and consistency across any tools you're using today, or may be using tomorrow.

This encourages collaboration, as teams have shared context and understanding regardless of what tools they prefer to use day-to-day.

If you want to learn more about Blameless SLOs, feel free to reach out to us [here](#) for a demo. If you enjoyed this guide and want to read more from us, check out these related resources:

- ▶ [SLO Adoption at Twitter](#)
- ▶ [What Are Service-Level Objectives? Lessons Learned](#)
- ▶ [Garrett Plasky Shares How SLOs Transformed Evernote](#)
- ▶ [Webinar: Implementing SLOs](#)

**Blameless** is the first end-to-end Site Reliability Engineering platform, trusted by leading teams such as Home Depot, Mercari, and Citrix. By operationalizing SRE best practices through unified context, workflow automation, and emphasis on learning, Blameless helps software teams embrace a culture of resilience. With integrated service level objectives, incident resolution automation, toil-free learning, reliability insights, and more, teams are empowered to optimize innovation velocity without sacrificing reliability. Headquartered in San Mateo, California, Blameless is backed by Lightspeed Venture Partners and Accel.