FFW™

Talk nerdy to me

Translating technical jargon into plain speak for the rest of us

My name's Leigh

Welcome to my session. We're all friends here.

I'm a marketer at FFW

Job title: Senior Marketing Communications Manager

What it really means: "Hey, I need some words written. Help!"

I used to work at the Drupal Association

It was my job to tell stories about Drupal and the human impact of the software

I'm here to help

If you get stuck, you can ping me on twitter: @leigh_can

FFW™

“What should I expect from this session?”

- What’s communication?
- What are communication barriers and how do we reduce them?
- How do we understand our audience better?
- How do we clarify our language?
- Cats

Let's get started

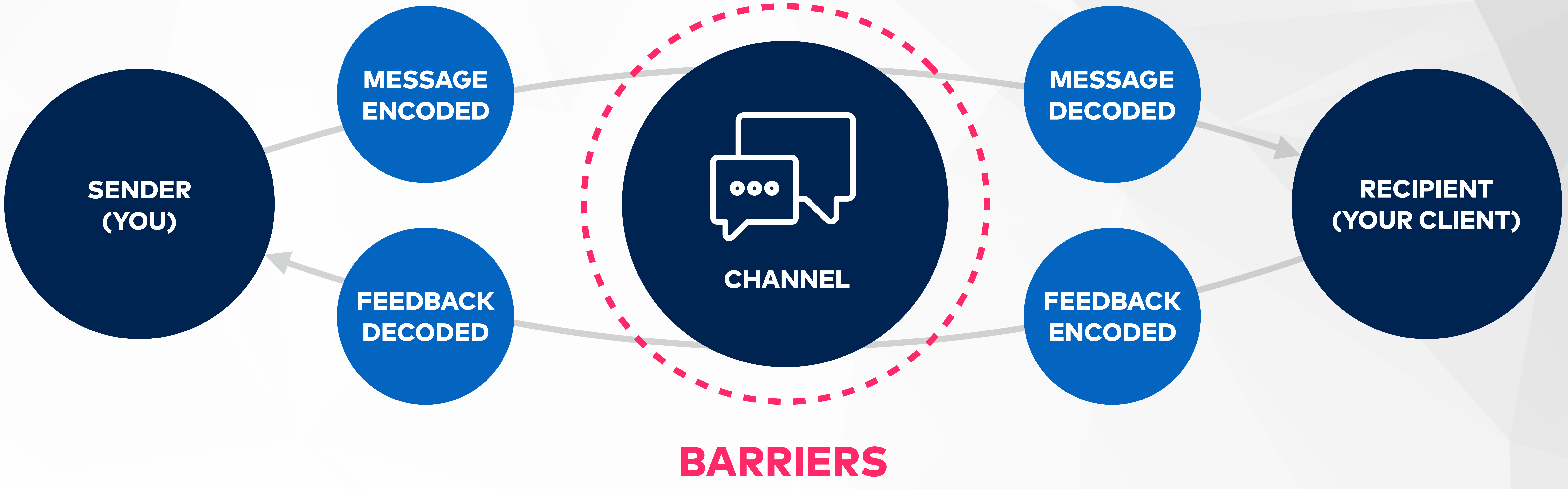
Communication

What is it, how do we do it, and how can we get better at it?

FFW

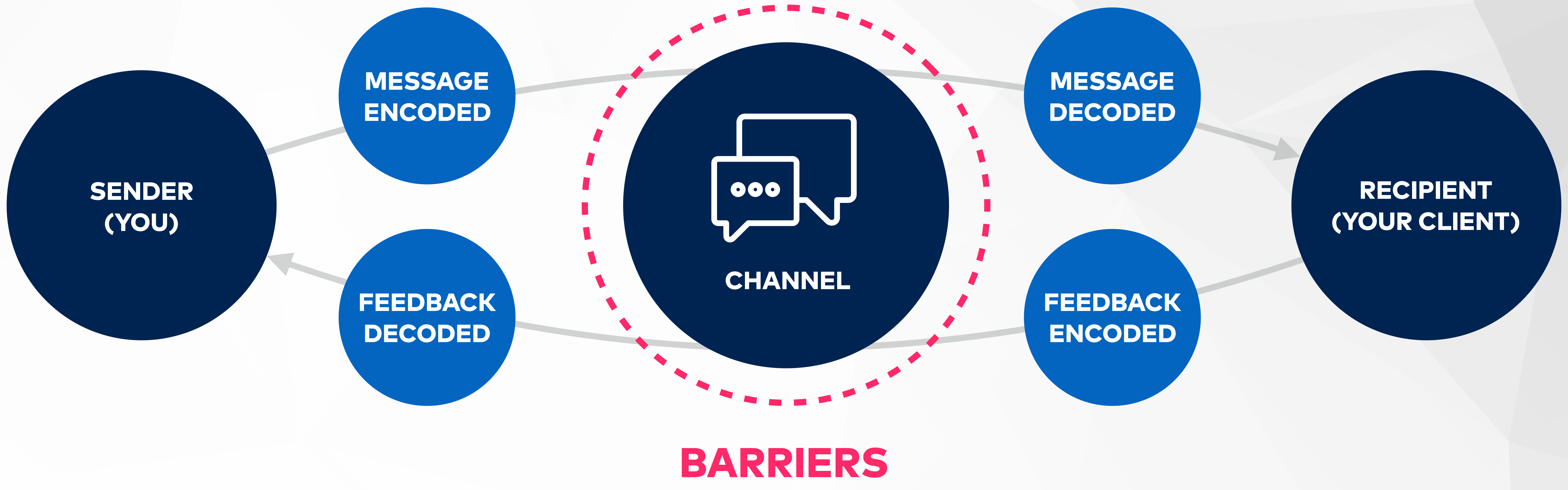
Communication 101

Communication is the act of transferring information from one source to another. Unfortunately, it's a really complicated process.



Barriers

" You keep using that word. I do not think it means what you think it means."



Clear communication means...

- Using the right **words**
- With the right **people**
- In the right **medium**
- At the right **time**
- For the right **response**

Reducing barriers

How do you identify barriers and navigate them?

Kinds of barriers

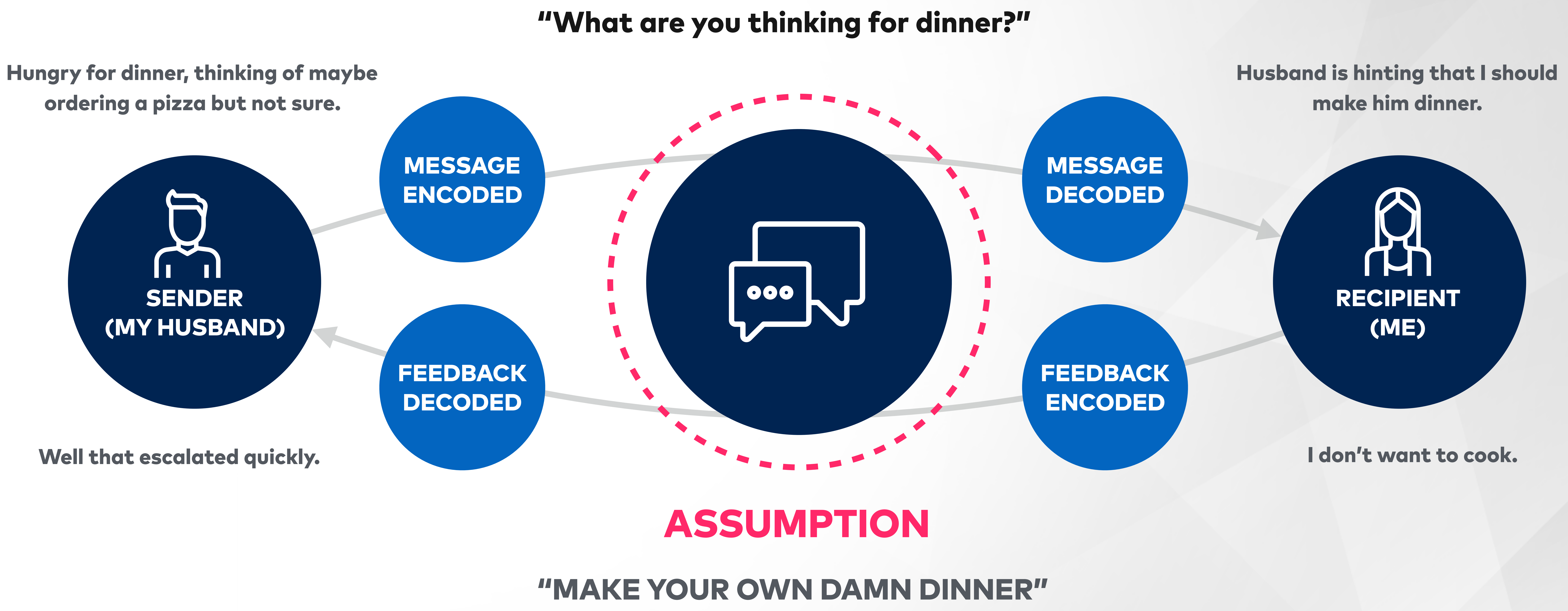
Encoding/decoding problems

- Incomprehensible jargon
- Lack of a shared knowledge base
- Linguistic or vocabulary barriers
- Different values or morals
- Different communication styles
- Inattention

Issues with the medium

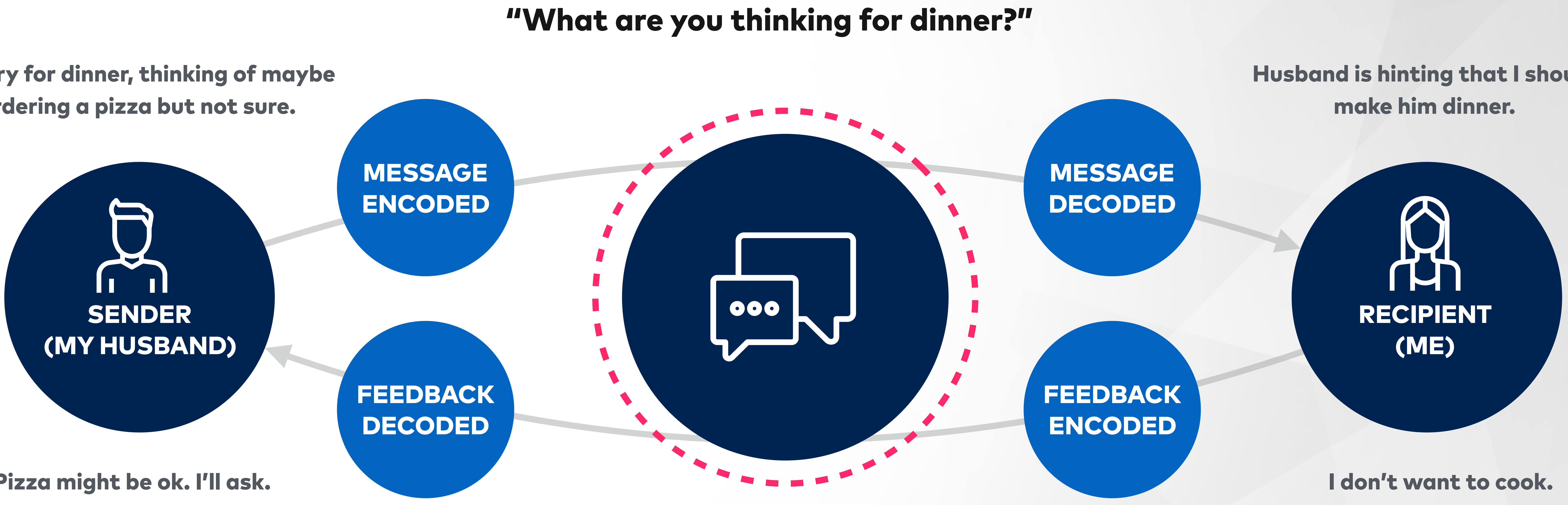
- Hearing, vision, or other disabilities
- Difficulties operating the medium
- Physical impairments around the medium (eg a call dropping or breaking up)
- Inherent flaws in the medium

Bad communication


"What are you thinking for dinner?"
Hungry for dinner, thinking of maybe ordering a pizza but not sure.
Husband is hinting that I should make him dinner.
MESSAGE ENCODED
MESSAGE DECODED
SENDER (MY HUSBAND)
RECIPIENT (ME)
FEEDBACK DECODED
FEEDBACK ENCODED
Well that escalated quickly.
I don't want to cook.
ASSUMPTION
"MAKE YOUR OWN DAMN DINNER"

FFW™

Better communication

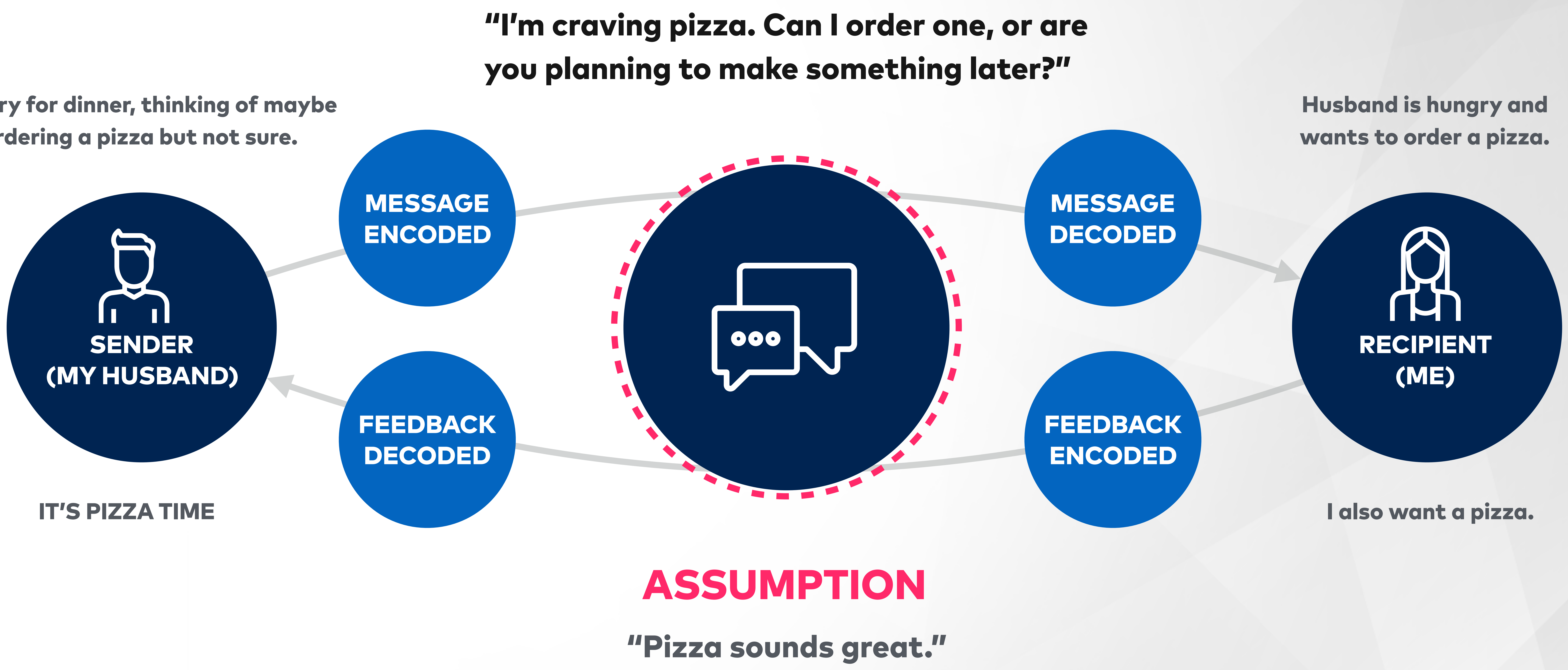

"What are you thinking for dinner?"
Hungry for dinner, thinking of maybe ordering a pizza but not sure.
Husband is hinting that I should make him dinner.
MESSAGE ENCODED
MESSAGE DECODED
SENDER (MY HUSBAND)
RECIPIENT (ME)
FEEDBACK DECODED
FEEDBACK ENCODED
Pizza might be ok. I'll ask.
I don't want to cook.

ASSUMPTION

"I hadn't given it much thought, but I'd prefer not to cook tonight. Why? Do you have something in mind?"

FFW

Best communication


"I'm craving pizza. Can I order one, or are you planning to make something later?"
Hungry for dinner, thinking of maybe ordering a pizza but not sure.
Husband is hungry and wants to order a pizza.
MESSAGE ENCODED
MESSAGE DECODED
SENDER (MY HUSBAND)
RECIPIENT (ME)
FEEDBACK DECODED
FEEDBACK ENCODED
IT'S PIZZA TIME
I also want a pizza.
ASSUMPTION
"Pizza sounds great."

FFW™

Know your audience

Even if you don’t want to

FFW™

- “How can I phrase this* in terms this person will understand?”
- “How can I talk about this in a way that will make <u>them</u> care?”

* without being condescending

Understand who you're talking to:

- What are they doing now?
- What are they trying to do?
- What's keeping them from doing the thing?
- What do they value?
- What makes them successful?
- What makes them fail?

Find your audience's barriers:

- Do this person and I have a similar cultural or professional background?
- Does this person speak my native or work language?
- Is this the best medium for the person I am communicating with?
- Is this the best medium for the message I need to share?
- Is this message structured in the right way for this medium?

Example client scenario:

1. Jordan Anderson runs a nonprofit living history museum outside the city of Austin, Texas

2. It's a primarily volunteer-based operation

3. Its main sources of revenue are from field trips and special events

4. He and his office staff are not particularly computer savvy

5. They have a low operating budget

6. You're building him a new site, and during the planning phase **he says:**

"I spend a lot of time sharing information about bookings to people. There's nowhere on the website for visitors to see if dates are blocked out, and I have to call all of my volunteers individually to ask them if they can work events or field trips. Some days I spend so much time coordinating events that I'm not able to get important maintenance done around the property. **It's a really big problem."**

Jordan needs a better way of sharing information about events.

What is he doing now?

- Running a nonprofit property, managing volunteers, managing event booking

What is he trying to do?

- Keep the lights on and the doors open, AND keep the property safe and clean

What's keeping him from doing the thing?

- Spending too much time on communication tasks

What does he value?

- Information dissemination
- Things that save him time in dealing with inquiries and volunteers
- Simplification of repetitive tasks

What makes him successful?

- Doing his job instead of answering the same question 1,000 times

What makes him fail?

- Losing money or time on repetitive communication tasks
- Paying through the nose for this new website
- Getting stuck with a site that he doesn't know how to use

Find Jordan's barriers:

Do Jordan and I have a similar cultural or professional background?

- Cultural, yes. Professional. no.

Does Jordan speak my native or work language?

- Native, yes. Work, no.

Is this the best medium for Jordan?

- He seems to respond better to verbal conversations than written ones.

Jordan needs a way to streamline his communication process so that people making inquiries about site availability can see online that the venue is booked, and so that volunteers will get notifications that they're needed onsite.

He doesn't have a lot of money to spend on the project, and he needs a solution that will save him time (and doesn't require a lot of technical expertise to manage).

How are you going to solve this problem?

- Phrase your answer in terms Jordan will understand
- Talk about this in a way that will make him care

We can build you a solution that...

- ✗ Uses an API to connect your internal calendar with your site
- ✗ Strips personally identifying data out of the calendar on the frontend
- ✗ Uses Workflows to identify which Fields have been selected in the Event Content Type, and then based on that we'll set up Rules to notify different users based on their Role on the backend.

- ✓ Whenever you make a calendar event, you can select whether it's a tour or a special event
- ✓ Depending on what kind of event it is, different people will be notified
- ✓ We'll also make sure that all of your availability information gets displayed on your website so that users can see that the venue is booked, but not by who
- ✓ Your volunteers who like to work events will get email notifications about events, and your volunteers who like to work tours will get email notifications about tours
- ✓ This is an easy thing to do, and is within your budget.

Simple
Straightforward
Precise
Awesome

To find clarity, run a workshop with your inner 4-year-old

"Ok, but why?"

"What's that mean?"

Hey, you did a really brilliant thing!
Congratulations, you brilliant person!

Hold up. Nobody's as excited about this as you are.

Why is that?

What’s the point?

Finding the point: be a 4-year-old.

Ask "Why" until you have the most basic possible answer

Variations on **"Why"** questions:

- ✓ What does the thing do?
- ✓ Why did you do the thing?

Avoid going down the **"How"** rabbit hole unless asked outright:

- ✗ How does the thing work?
- ✗ How did you come up with the idea?

Avoid technical language. Find a concrete reason for doing what you did, rather than an abstract one.

An example from real life - a livestream site for a massive event:

The most difficult feature of the site by far was a cache invalidation solution that we built.

We rewrote a time-based cache expiration so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a very good cache-hit ratio. We had these very long cache lifetimes, but when a page changed, we'd use an API provided by our hosting provider and our CDN to explicitly remove the documents from cache.

Then, to maintain referential integrity, we added a chron job that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

gifbin.com

Find the most important info:

The most difficult feature of the site by far was a cache invalidation solution that we built.

We rewrote a time-based cache expiration so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a very good cache-hit ratio. We had these very long cache lifetimes, but when a page changed, we'd use an API provided by our hosting provider and our CDN to explicitly remove the documents from cache.

Then, to maintain referential integrity, we added a chron job that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a time-based cache expiration so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a very good cache-hit ratio. We had these very long cache lifetimes, but when a page changed, we'd use an API provided by our hosting provider and our CDN to explicitly remove the documents from cache.

Then, to maintain referential integrity, we added a chron job that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a very good cache-hit ratio. We had these very long cache lifetimes, but when a page changed, we'd use an API provided by our hosting provider and our CDN to explicitly remove the documents from cache.

Then, to maintain referential integrity, we added a chron job that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a **very good cache-hit ratio**. We had these very long cache lifetimes, but when a page changed, we'd use an API provided by our hosting provider and our CDN to explicitly remove the documents from cache.

Then, to maintain referential integrity, we added a chron job that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a **very good cache-hit ratio**. We had these very long cache lifetimes, but **when a page changed**, we'd use an API provided by our hosting provider and our CDN to explicitly remove the documents from cache.

Then, to maintain referential integrity, we added a chron job that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a **very good cache-hit ratio**. We had these very long cache lifetimes, but **when a page changed**, we'd use an API provided by our hosting provider and our CDN to explicitly **remove the documents from cache**.

Then, to maintain referential integrity, we added a chron job that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a **very good cache-hit ratio**. We had these very long cache lifetimes, but **when a page changed**, we'd use an API provided by our hosting provider and our CDN to explicitly **remove the documents from cache**.

Then, **to maintain referential integrity**, we added a chron job that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a **very good cache-hit ratio**. We had these very long cache lifetimes, but **when a page changed**, we'd use an API provided by our hosting provider and our CDN to explicitly **remove the documents from cache**.

Then, **to maintain referential integrity**, we **added a chron job** that would pull the Drupal database every minute to check and see if there were any videos that experienced a live-state change in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a **very good cache-hit ratio**. We had these very long cache lifetimes, but **when a page changed**, we'd use an API provided by our hosting provider and our CDN to explicitly **remove the documents from cache**.

Then, **to maintain referential integrity**, we **added a chron job** that would pull the Drupal database every minute to check and see if there were any videos that experienced a **live-state change** in the current minute. If there was, it would look up the URLs of the live-state features, and send a purge to the hosting providers.

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a **very good cache-hit ratio**. We had these very long cache lifetimes, but **when a page changed**, we'd use an API provided by our hosting provider and our CDN to explicitly **remove the documents from cache**.

Then, **to maintain referential integrity**, we **added a chron job** that would pull the Drupal database every minute to check and see if there were any videos that experienced a **live-state change** in the current minute. If there was, it would look up the URLs of the live-state features, and **send a purge to the hosting providers.**

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.

Find the most important info:

The most difficult feature of the site by far was a **cache invalidation** solution that we built.

We rewrote a **time-based cache expiration** so that pages were cached for a very long time — the shortest span was 30 minutes, and the longest would be 48 hours, which would give us a **very good cache-hit ratio**. We had these very long cache lifetimes, but **when a page changed**, we'd use an API provided by our hosting provider and our CDN to explicitly **remove the documents from cache**.

Then, **to maintain referential integrity**, we **added a chron job** that would pull the Drupal database every minute to check and see if there were any videos that experienced a **live-state change** in the current minute. If there was, it would look up the URLs of the live-state features, and **send a purge to the hosting providers.**

So those three things: pages that purge when they save, pages that purge referential data, and minute-pulling — combined, **those give you a site that, to the end-user and to the editorial team, looks like it's updating in real time.**

The simplified version:

- We built a cache invalidation solution
- We rewrote a time-based cache expiration
- This gave us a very good cache-hit ratio
- When a page changed, we'd remove the documents from cache
- To maintain referential integrity, we added a chron job
- When the job found changes, it would look up the URLs and purge them.

Those three things created a site that, to the end-user and to the editorial team, looks like it's updating in real time.

... getting closer, but still not quite there.

What's the point?

Let's break this down:

We built a cache invalidation solution

- **WHAT'S THAT:** We built a solution to make the site feel like it was updating in real-time.
- **WHY:** Because it was a big streaming event, so we needed to have updates as they happened.

We rewrote a time-based cache expiration

- **WHAT'S THAT:** We made the pages update themselves as infrequently as possible.
- **WHY:** It's a big site and we couldn't update every single page constantly.

This gave us a very good cache-hit ratio

- **WHAT'S THAT:** It determines how fast your site loads for people.

Having a breakdown, part two:

When a page changed, we'd remove the documents from cache

- **WHAT'S THAT:** When someone edited or changed a page, we'd pull that page out of where it was stored on the site's server, and THEN...

To maintain referential integrity, we added a chron job. When the job found changes, it would look up the URLs and purge them.

- **WHAT'S THAT:** To keep from accidentally breaking the site, we had a special program that would update or swap out any old pages with new ones instead.

The simple version:

- Because it was a large streaming event and the site needed to reflect changes in real-time,
- We built a solution to make the pages update instantly.
- We were working with a big site, and having every page update constantly would have slowed it down.
- So we made the pages update themselves as infrequently as possible.
- When someone edited or changed a page, we'd pull that page out of where it was stored on the site's server.
- Then, to keep from accidentally breaking everything, a special program that we made would update or swap out any old pages with new ones instead.

Even though the site wasn't actually refreshing every single page constantly, it still felt like everything was being updated in real time.

Client says: **"Wow! How'd you make the site go so fast?"**

Identify Barriers:

- **Does this person have a similar background to me?**
  Similar upbringing, different professions
- **Does this person speak my language?**
  Almost definitely won't understand the technical language
- **Is this the best medium for the person I am communicating with?**
  Client's attention tends to drift, so I need to keep my verbal responses short
- **Is this the best medium for the message I need to share?**
  Unless you've got someone taking notes, verbally is a bad way of explaining cache invalidation
- **Is this message structured in the right way for this medium?**
  A 5-minute soliloquy on cache invalidation will probably make their eyes glaze over

“Wow! How’d you make the site go so fast?”

“We actually gave every page a really really long shelf life. Most pages only updated on their own every few hours, or even every few days. But we built a special program on the site that was constantly checking for changes on pages. Whenever a page was updated, that program would go in, pull the old page, and update or change the information.”

“That’s so cool! How did you do that?”

“Do you know what a cache is?”

“Nope!”

Lucky you, you get to repeat that whole long process you just went through.

Let’s do it again.

An example from real life - an education platform:

We were having trouble with the underlying system architecture.

Our client was serving up 300 GB of digital assets, like design files, videos, photos, and whatnot. 90,000 students were uploading what they were creating day-to-day and the system wasn't able to keep up.

To solve it, we separated Drupal from that file content by building an abstracted file system using an Amazon Web Server.

Let's break this down:

- We were having trouble with the underlying system architecture.
- The site was was serving up 300 GB of digital assets, while 90,000 students were uploading what they were creating day-to-day
- The system wasn’t able to keep up.
- To solve it, we separated Drupal from that file content by building an abstracted file system using an Amazon Web Server.

… getting closer, but still not quite there.

What's that? Why? What's that? Why?

We were having trouble with the underlying system architecture

- **WHAT'S THAT:** The site wasn't built right for the job it was being asked to do.

The site was was serving up 300 GB of digital assets, while 90,000 students were uploading what they were creating day-to-day.

- **WHAT'S THAT:** It was being asked to serve a LOT of content. Like, a LOT.

The system wasn't able to keep up.

- **WHY:** The files were too big to store in the site itself, so we had to find another place to put them.

What’s that? Why? What’s that? Why?

To solve it, we separated Drupal from the file content and stored it on AWS.

- **WHAT’S THAT:** The files were too big to store in the site itself, so we had to find another place to put them.

- **BUT WHY:** It’s basically the difference between keeping all your papers in a pile on your desk, or keeping them in a filing cabinet.

“How’d you get the site to stop crashing?”

“We were storing files wrong and the site couldn’t keep up. So, we pulled the files out of the site and put them in their own special system. Instead of rummaging around through all of the page and site content every time it needs a file, the site accesses the special system, which pulls the file and sends it over to the site.”

“That’s so cool! How did you do that?”

“Do you know what Amazon Web Services is?”

“Nope!”

“It’s like a filing cabinet. Sort of.”

What’s the point?

Yes, it's a long process. But once you get the hang of it, it becomes second nature.

- When we communicate clearly, we build better relationships.
- We can't account for every potential barrier in interpersonal communication.
- But if you try, people will usually do their best to meet you halfway.

Clarity and kindness make everything better.

Any questions?

If you've brought tricky problems of your own, now's the time to share with the class

FFW™

Resources

Sideways Dictionary: https://sidewaysdictionary.com/
(A dictionary of analogies explaining tech. Hooray!)

Internet Linguistics: https://gretchenmcculloch.com/
(An excellent starting point for exploring how we communicate online)

Communication Skills Toolkit: https://www.skillsyouneed.com/ips/communication-skills.html

Thank you!

FFW™

FFWagency.com