Working Backwards

Outcomes & Goals

James Doyle

@james2doyle

Optix

Show of *hands!*

- Freelancers? (dev, design)
- Client services? (agency, contract)
- Products?

Why this talk?

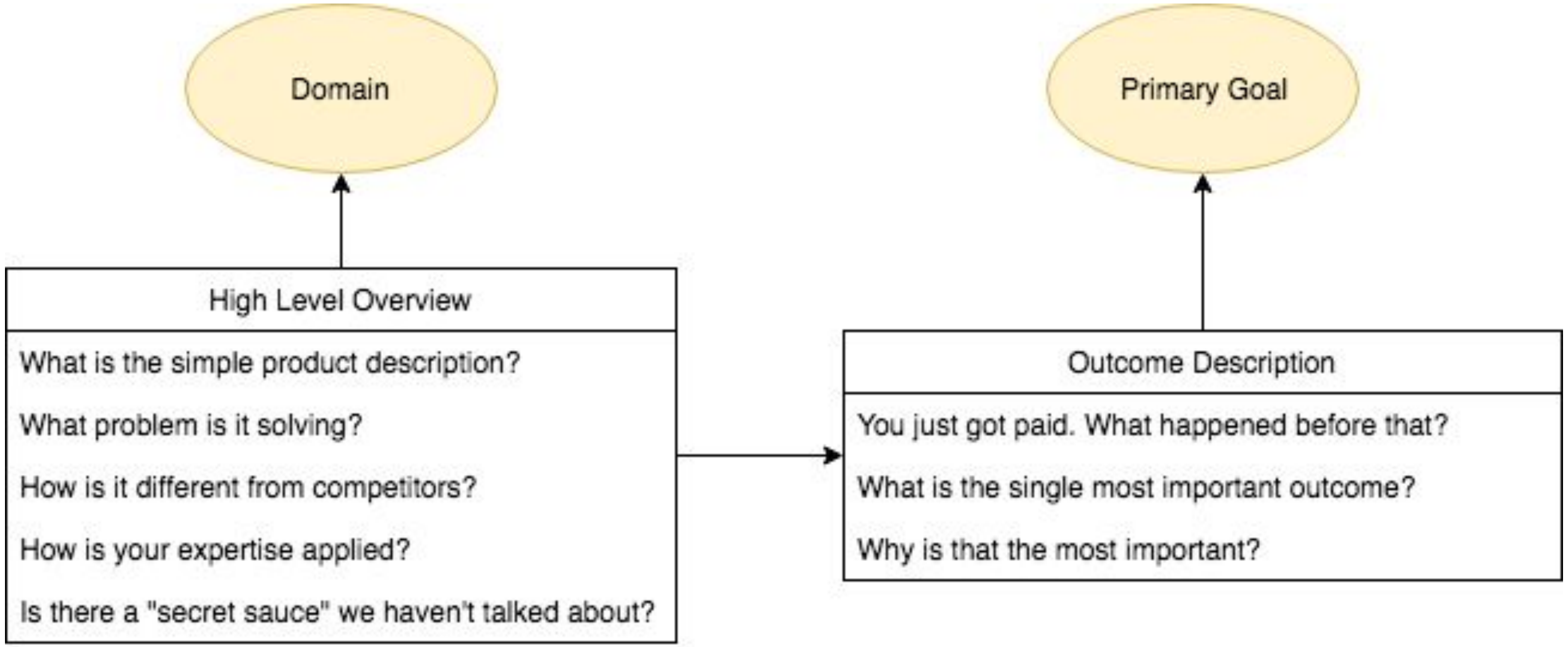
Communication between project managers, designers, and developers over how to plan a project is a mess

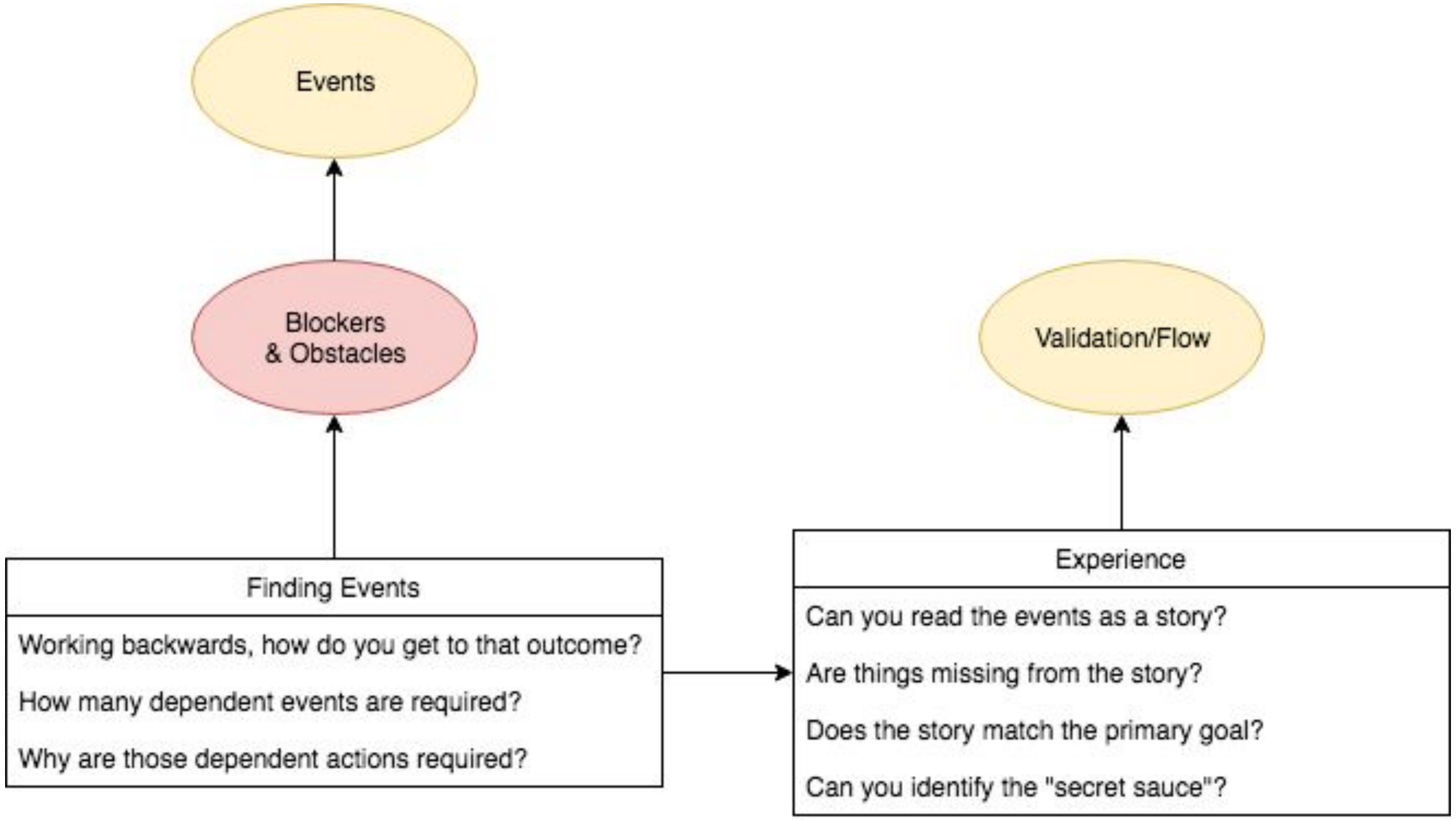
- Specs?
- Agile?
- What is the MVP?
- Are there blockers?
- Why this decision?
- What happens next?

What would a *potential process* look like?

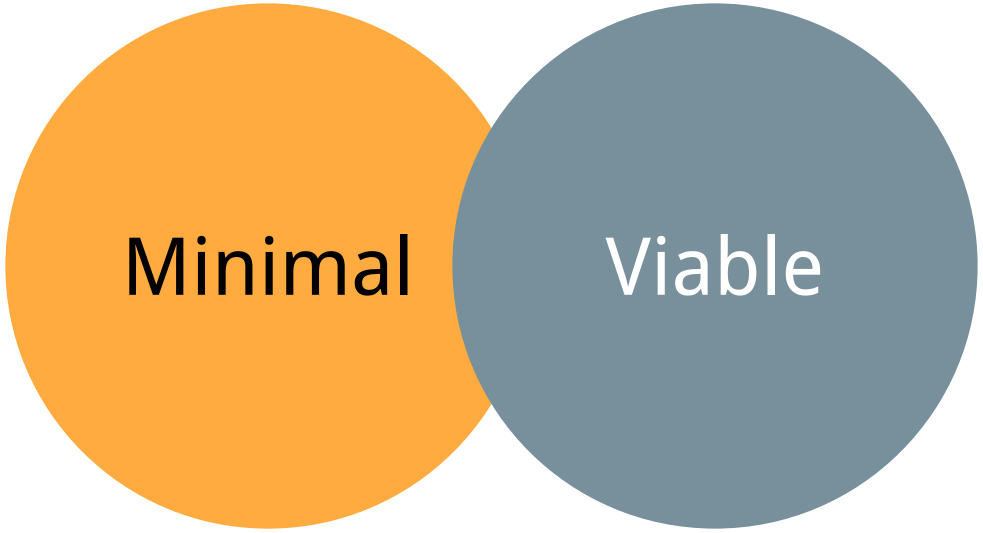
- Simple!
- Built for non-technicals
- Limited jargon
- Clear rules

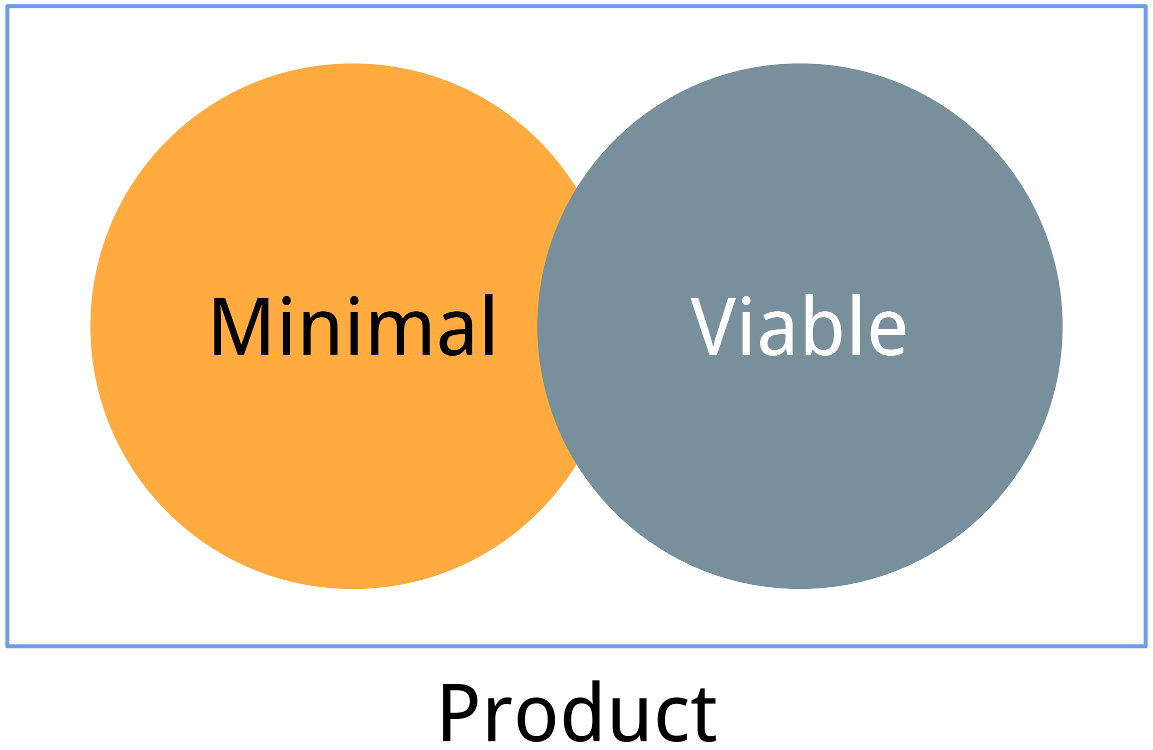
My *process*

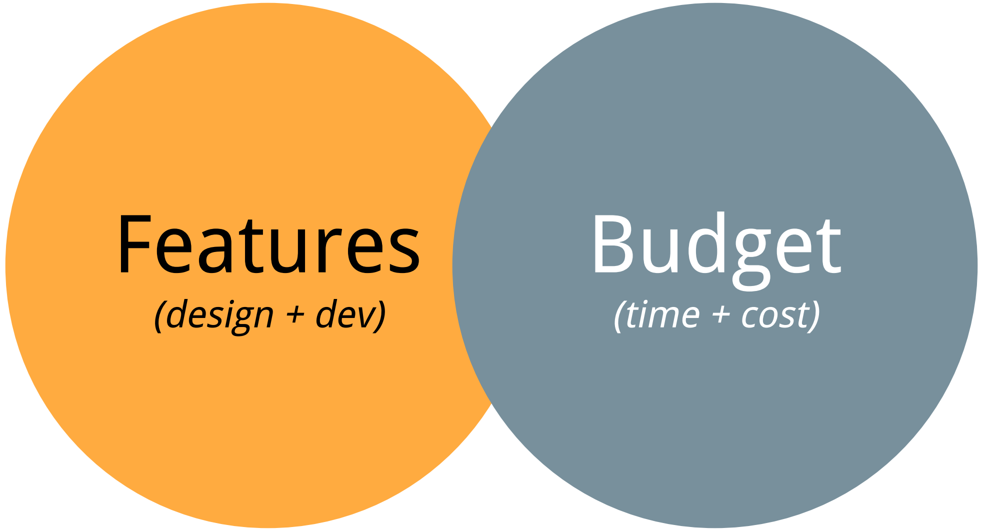

Domain
Primary Goal
High Level Overview
What is the simple product description?
What problem is it solving?
How is it different from competitors?
How is your expertise applied?
Is there a "secret sauce" we haven't talked about?
Outcome Description
You just got paid. What happened before that?
What is the single most important outcome?
Why is that the most important?

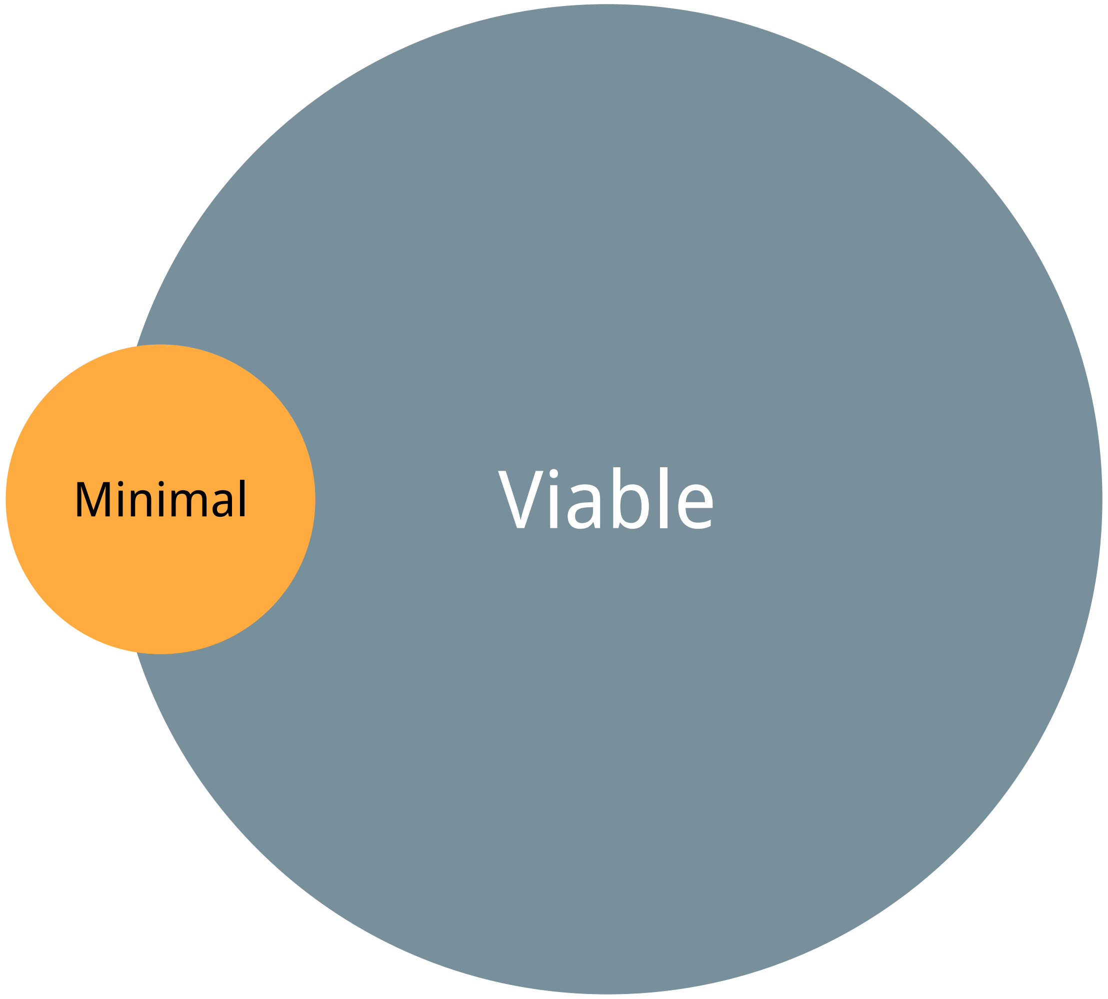

Events
Blockers & Obstacles
Validation/Flow
Finding Events
Working backwards, how do you get to that outcome?
How many dependent events are required?
Why are those dependent actions required?
Experience
Can you read the events as a story?
Are things missing from the story?
Does the story match the primary goal?
Can you identify the "secret sauce"?

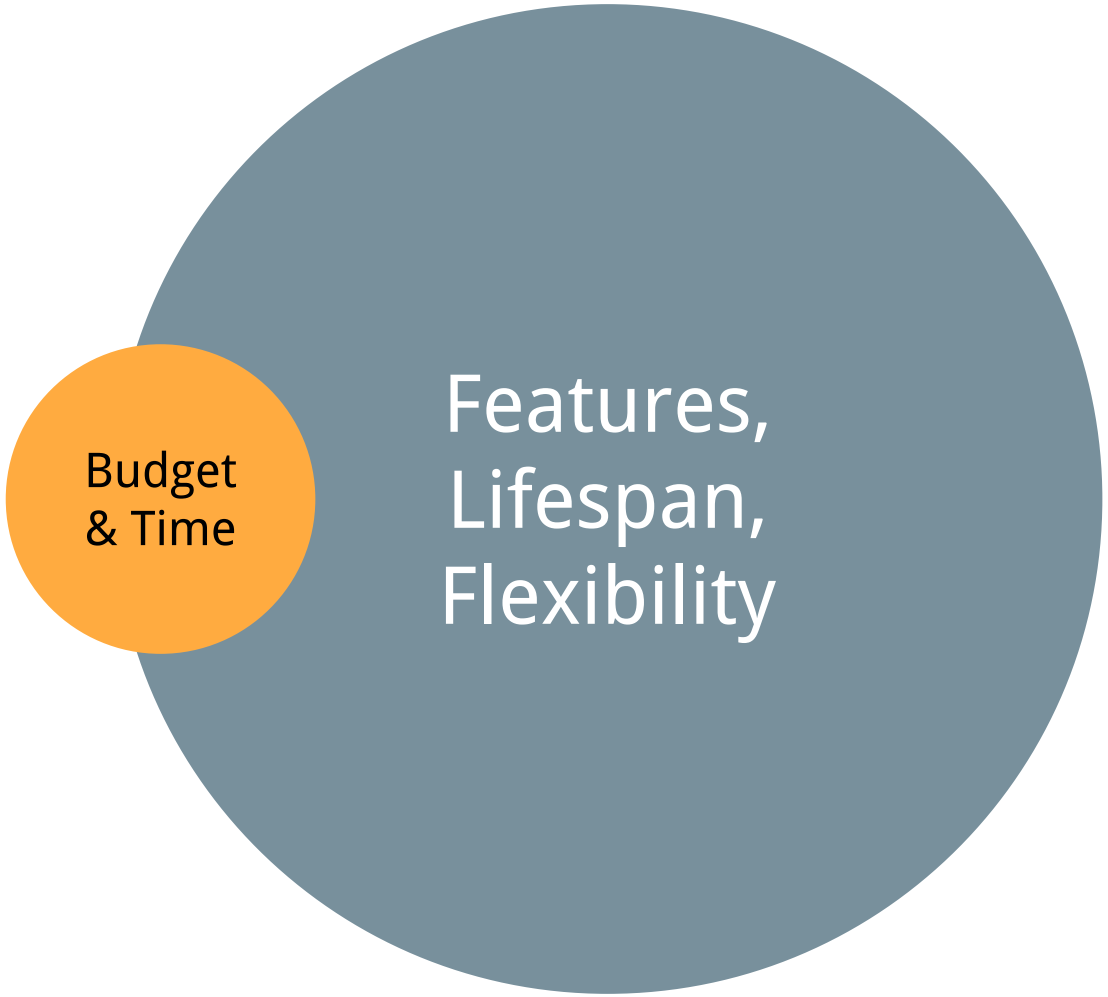
What is an MVP *really?*

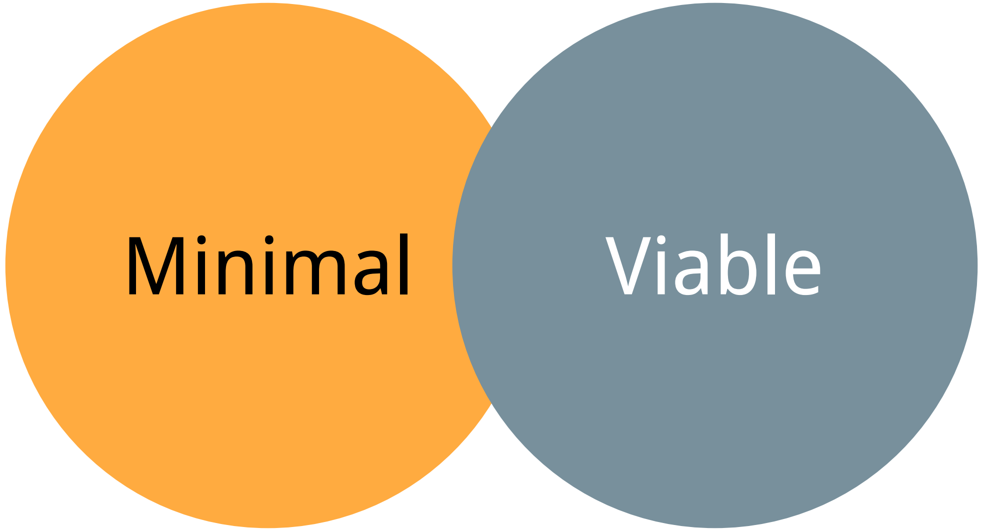

Minimal
Viable

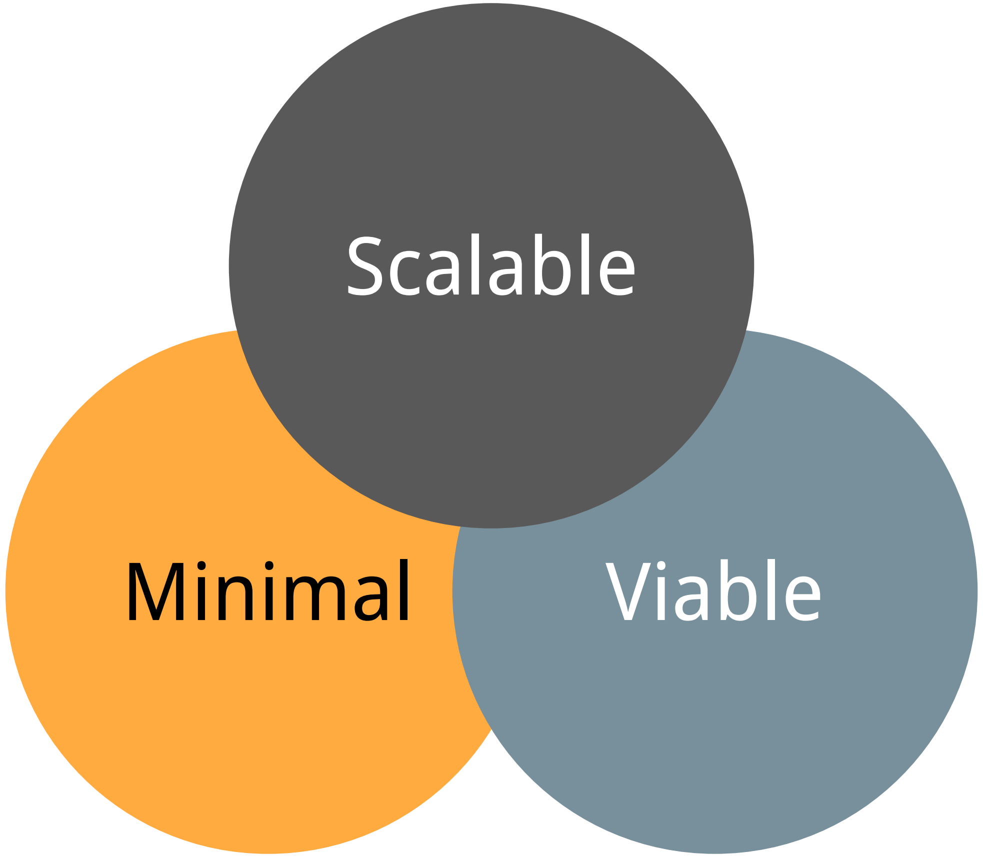

Minimal
Viable
Product


Features
(design + dev)
Budget
(time + cost)


Minimal
Viable


Budget & Time
Features, Lifespan, Flexibility


Minimal
Viable


Scalable
Minimal
Viable

What’s wrong with design/development through *iteration?*

A Real World Reference

Your eyes

- Everything is upside down
- Requires eyelids
- Requires ample light
- Limited visual spectrum
- Fixed directional positioning
- Single focus point
- Limited distance (close & far)

How did they get that way?

Slow iteration over millions of years with *no rewrites*, *only adaptations*

The eye was developed to detect ambient light, not see long distances and become an emotional indicator

If we designed eyes based on the required job (*outcome*), would they be better?

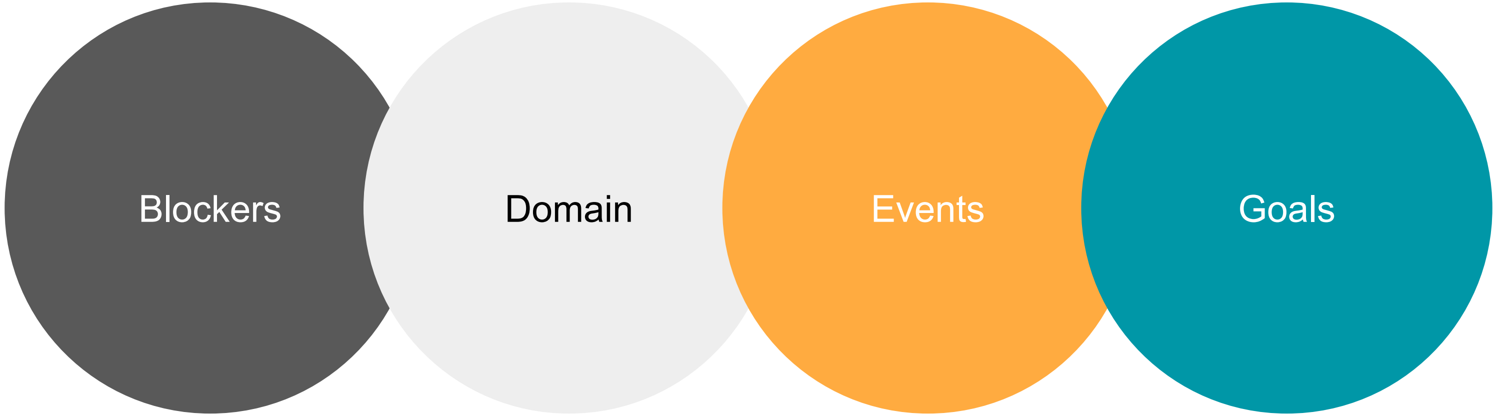
How do we design a better *MVP?*

Understanding over *Experimenting*

Knowing is better than *Assuming*

Understanding is fundamental to delivery

How do we improve
our *understanding?*


Blockers
Domain
Events
Goals

Blockers

Dependencies that are beyond your field of influence that can inhibit your progress

- Time and budget
- Unknowns
- 3rd Party tools/services
- Payment gateways
- Certifications (PCI, *GDPR*)
- etc.

Domain

Defines a set of common requirements, terminology, and functionality

- User
- Account
- Profile
- Identity
- Credentials
- etc.

Events

Actions, processes, and triggers that occur within the system

- User signed in
- Payment deposited
- Team created
- Comment deleted
- Error occurred
- etc.

Goals

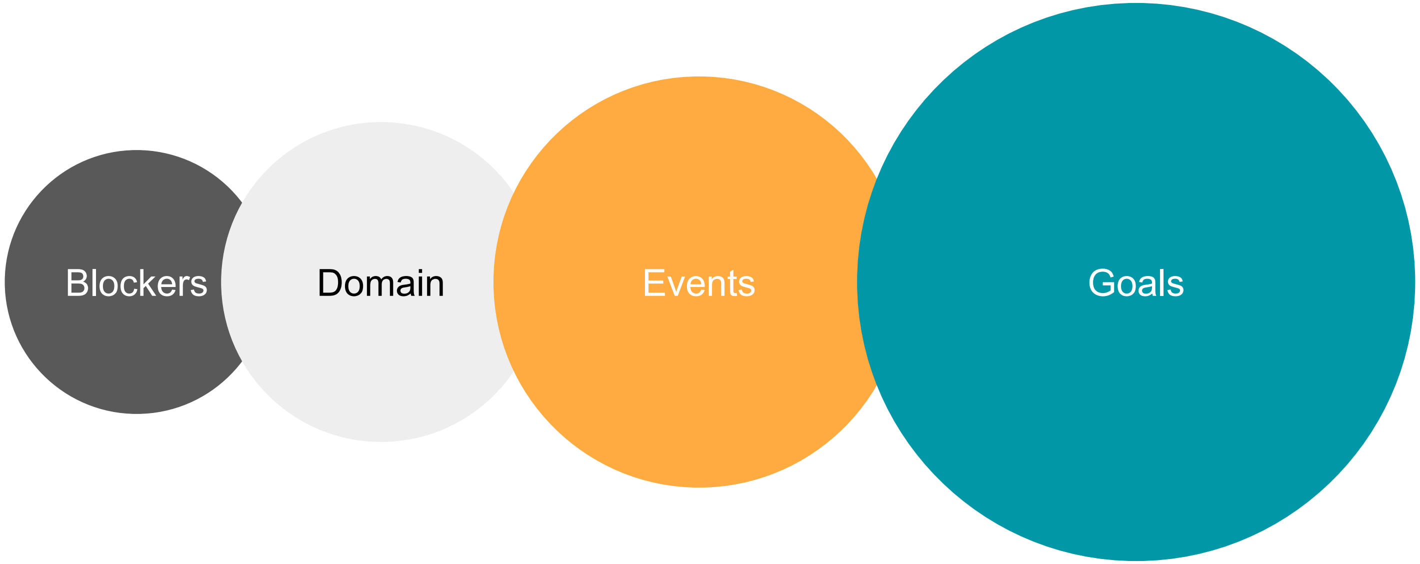
Targets and actions that will indicate success or failure within the defined parameters

- User sign up
- Adding credit card
- Upload an image
- etc.

Validation

The ability to tell a story with the information gathered through the previous steps.

- Tell a story
- Know if something is missing
- Identify blockers
- etc.


Blockers
Domain
Events
Goals

Build an *understanding* by *working backwards* from your goals

How Do You Set A *Goal?*

Focus On The End Result

Don’t think about what you did to get there, just assume you are already at the finish line.

Focus On The Happy Path

Don’t focus on what can go wrong, just assume nothing has and nothing ever will.

What do *you* care about?

- User Acquisition
- Branding & Aesthetics
- Features & Functionality
- Content & Communication

But what is the *primary goal?*

- Buy your products?
- Share your content?
- Subscribe to your platform/service?
- Contact you for services?

But what is the *primary goal* for *my product/service?*

What is the single most important outcome?

Ok, how about a hint?

MAKE $$$!!!!!

You just got paid.

What happened before that?

- Working backwards, how do you get to that outcome?
- How many dependent actions are required?
- Why are those dependent actions required?

A Simple Example

A Social Network

Finterest

Finterest

Finally, a social network for fish

Social Network

Users can post content, follow each other, tag each other, join groups, etc.

- Users
- Groups
- Posts
- Follow
- Tag

Social Network

We make money selling ads on the platform. Our revenue relies on users interacting with the ads.

- Ads
- Interactions

Build an *understanding* by *working backwards* from your goals

You just got paid.

What happened before that?

Hold up! Let’s talk about *good event design!*

Good Event Design

We want events that can translated into verifiable actions for QA, design, code, and testing

- Simple
- Past tense
- Include a domain
- Actionable
- Chainable
- Reusable
- Flexible

BAD!

Some examples of bad events

Bad Event Design

Bad events usually have no actionable outcome, describe a state, and cannot be chained

- Describes the state
- Passive
- Assumes UI
- Out of your control

The user sees a success message

✔ Past tense

✔ Include a domain

✔ Chainable

✘ Simple

✘ Actionable

✘ Reusable

✘ Flexible

User was notified

✔ Past tense

✔ Include a domain

✔ Chainable

✔ Simple

✔ Actionable

✔ Reusable

✔ Flexible

User was notified

User triggered undo

User was notified

The user clicked on the submit form button

✔ Past tense

✔ Include a domain

✔ Actionable

✔ Chainable

✘ Flexible

✘ Simple

✘ Reusable

User submitted form

✔ Past tense

✔ Include a domain

✔ Actionable

✔ Chainable

✔ Flexible

✔ Simple

✔ Reusable

User submitted form

Use was notified

User submitted form

Use was notified

Give me *events!*

MAKE $$$!!!!!

You just got paid.

What happened before that?

Primary Goal

?
?
?
Payment deposited

Now we *extract* some information

Domains & Events

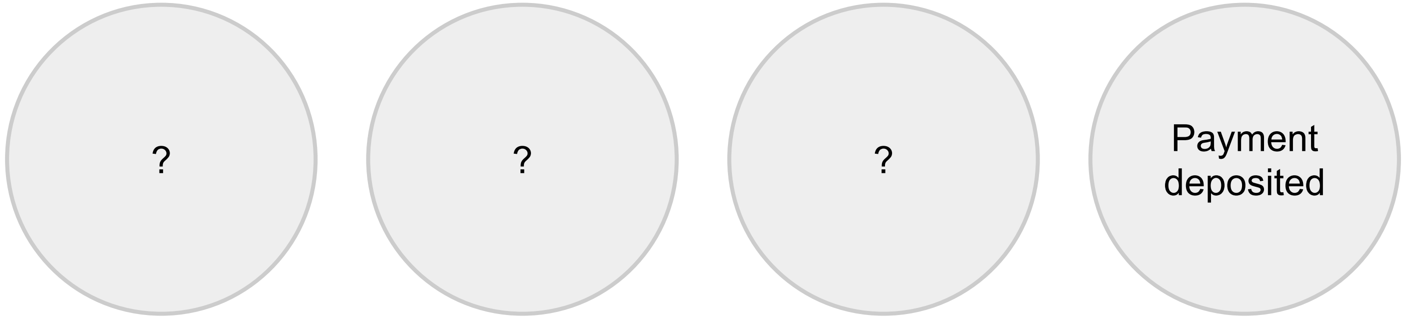
Domains:

- Payments

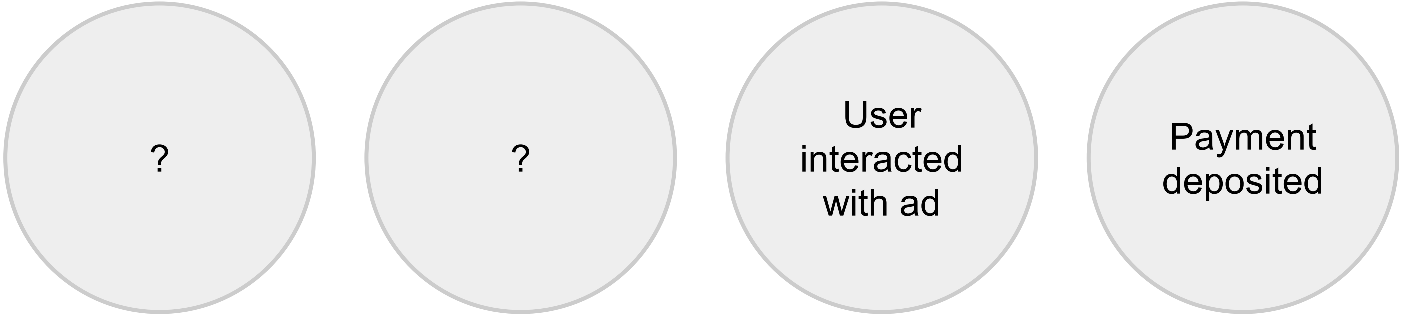
Events:

- Payment deposited

Primary Goal



Primary Goal



Domains & Events

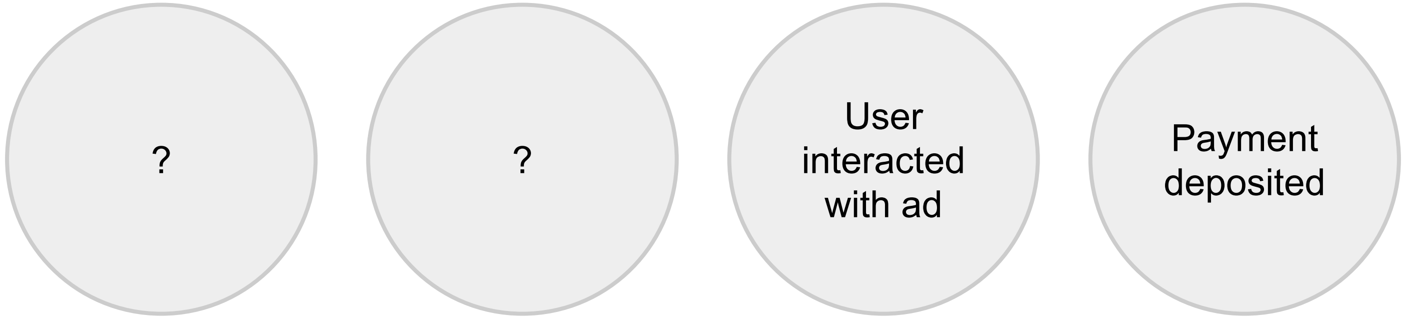
Domains:

- Users
- Ads
- Payments

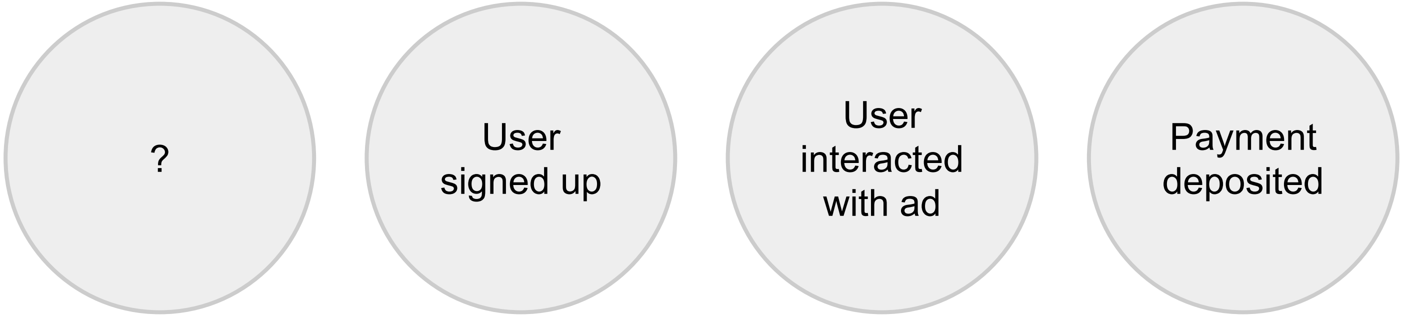
Events:

- User interacted with ad
- Payment deposited

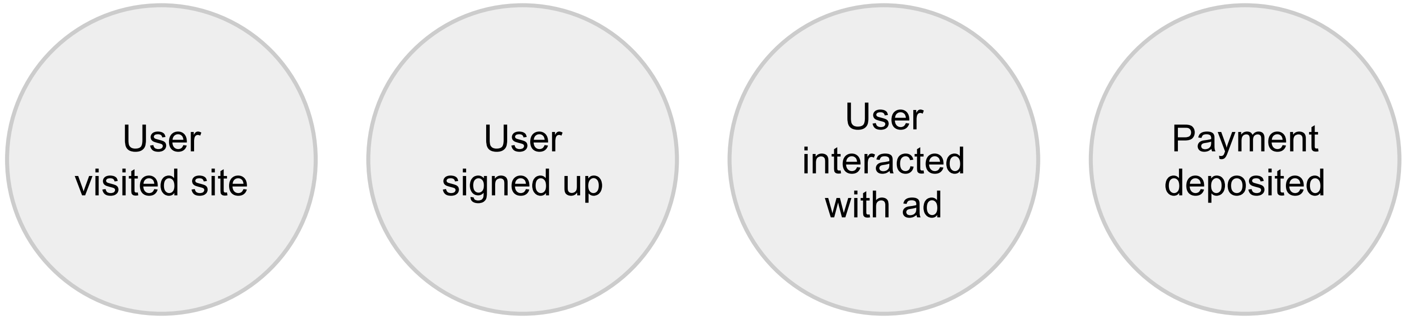
Primary Goal



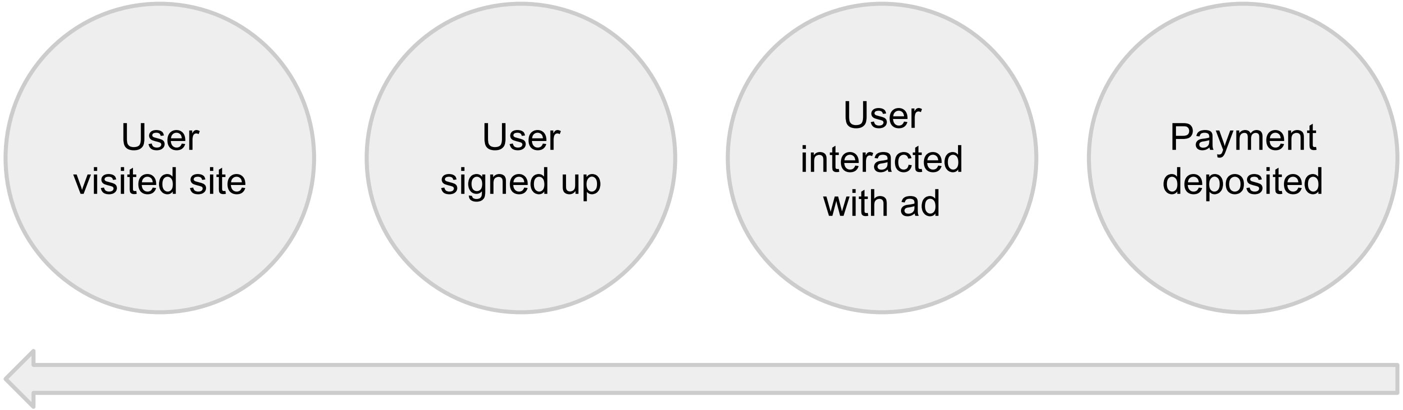
Primary Goal



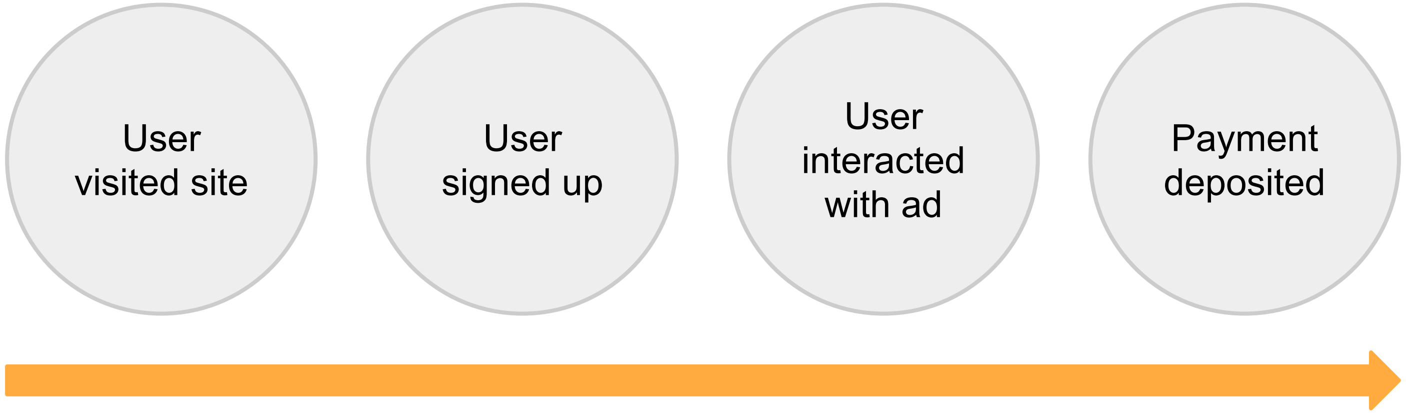
Primary Goal



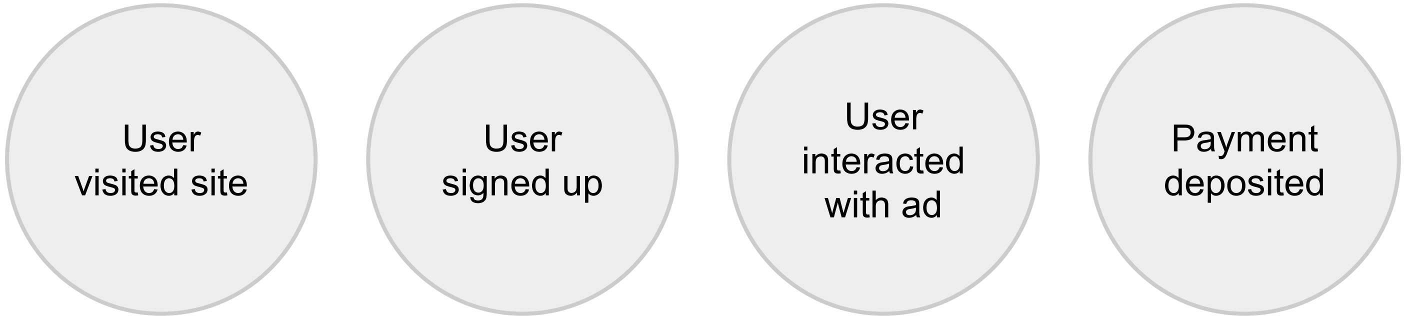
Primary Goal



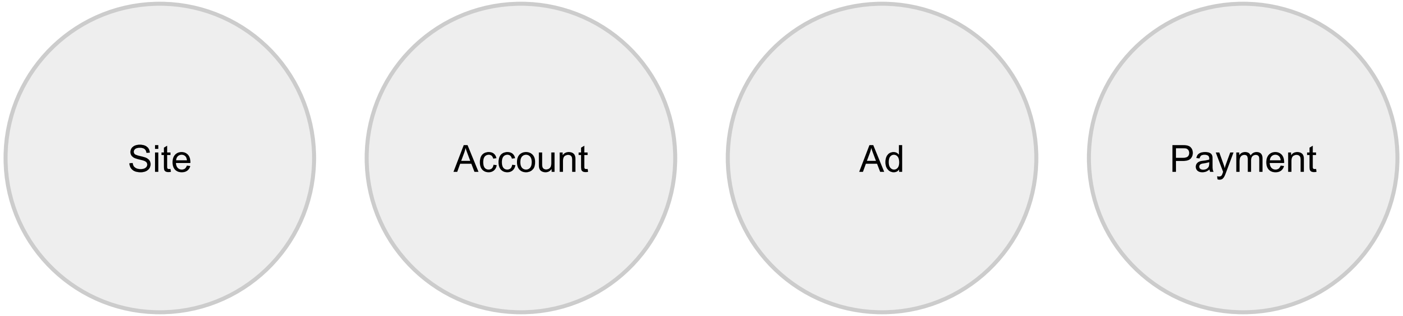
Primary Goal



Primary Goal



Primary Goal



Domains & Events

Domains:

- Users
- Ads
- Accounts
- Payments
- Site

Events:

- User visited site
- User signed up
- User interacted with ad
- Payment deposited

Another Example

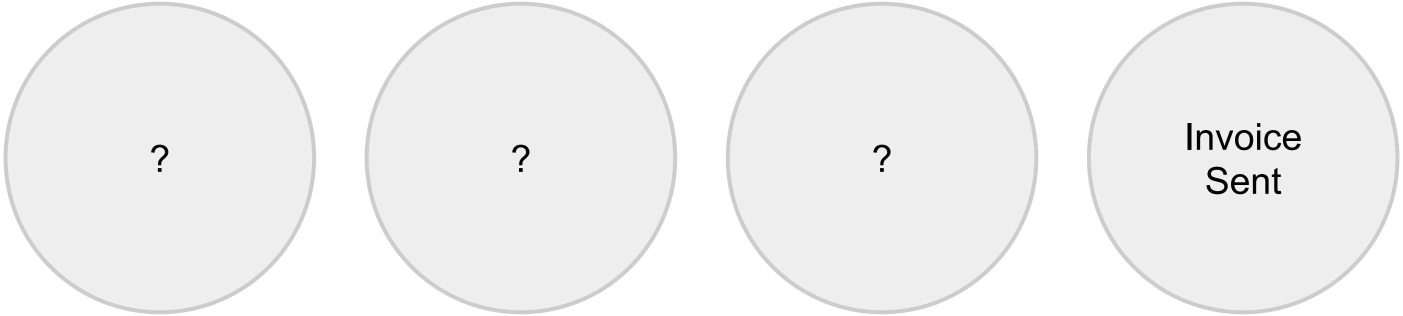
How about applying this
to *a single feature?*

Sending Invoices

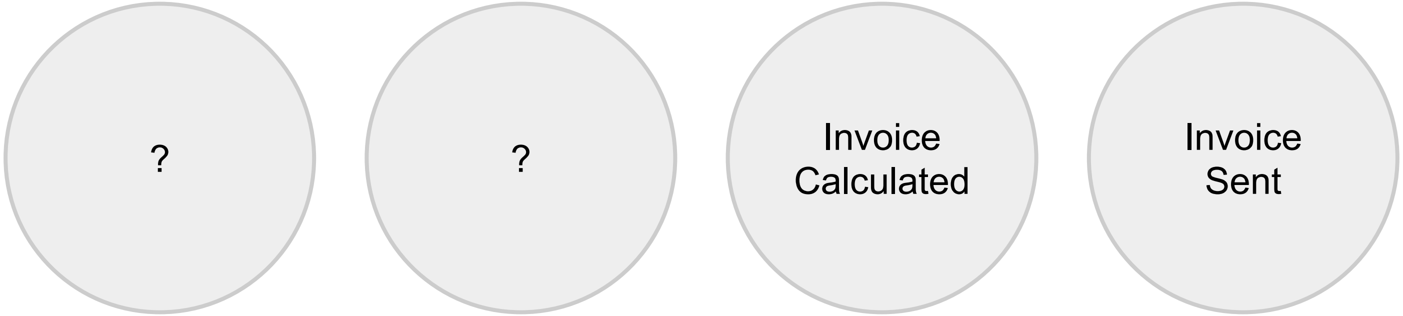
~~You just got paid.~~

~~*What happened before that?*~~

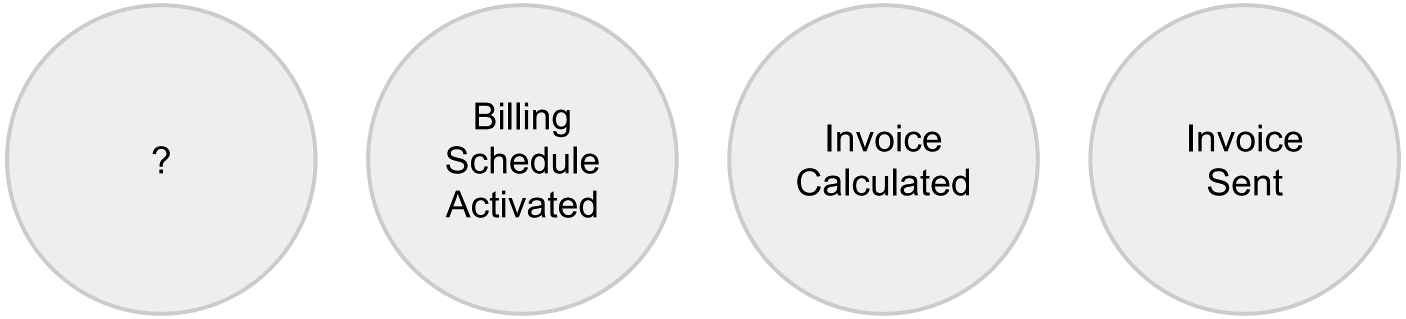
Sending Invoices



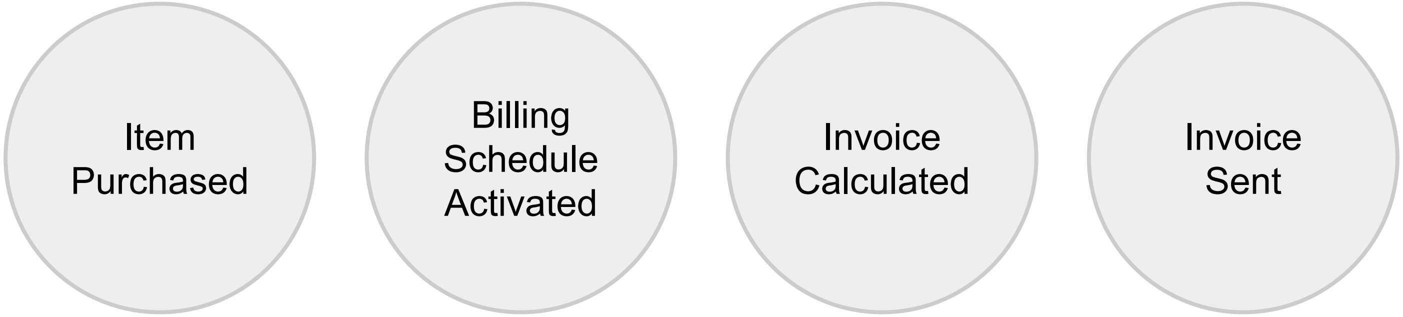
Sending Invoices



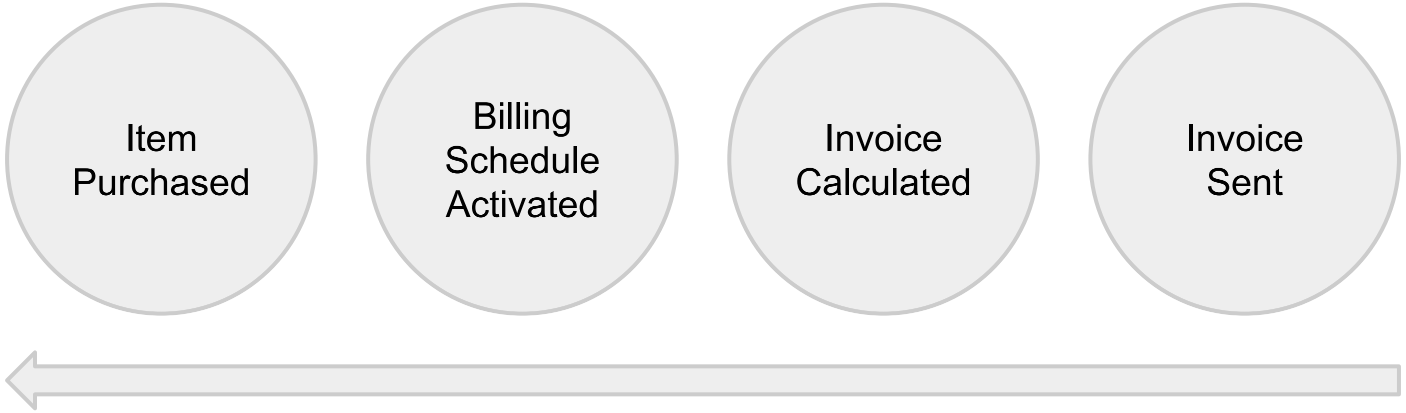
Sending Invoices



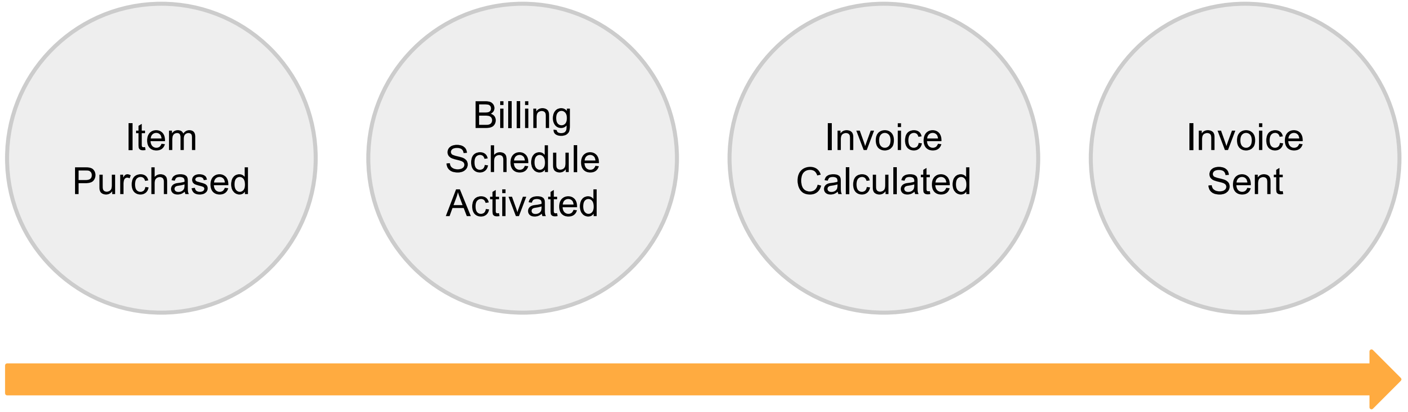
Sending Invoices



Sending Invoices



Sending Invoices



Domains & Events

Domains:

- Invoice
- Items
- Billing Schedule
- Invoice Calculation

Events:

- Item Purchased
- Billing Schedule Activated
- Invoice Calculated
- Invoice Sent

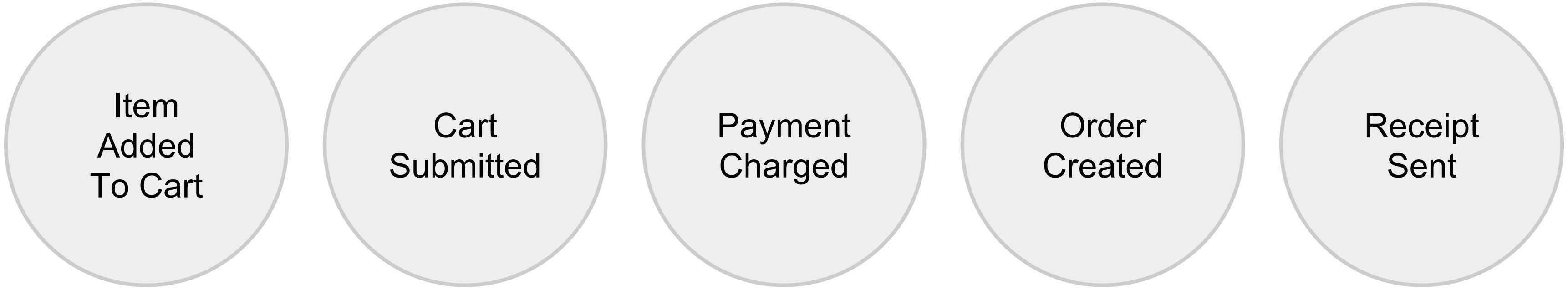
Example: The Unhappy Path

Unhappy Path

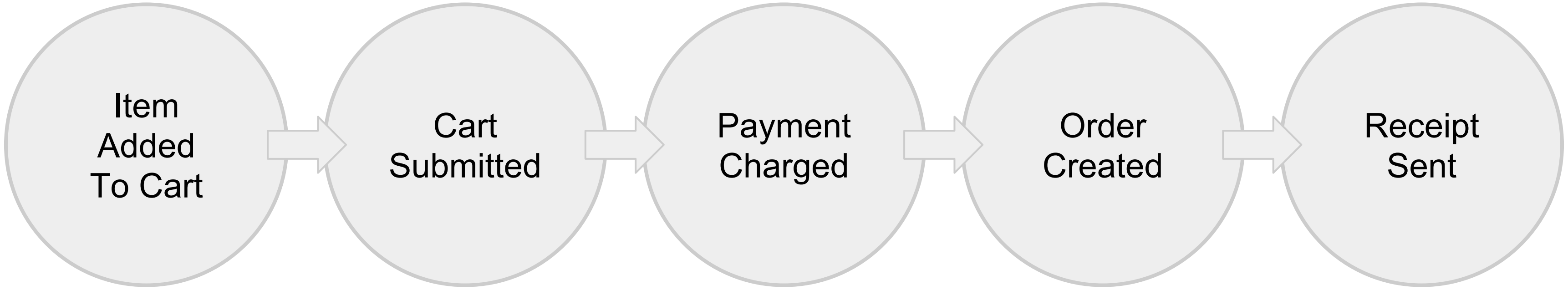
The path where a user, system, UI, or service experiences errors or problems during operation

- Payment failures
- Server errors
- Missing data
- Validation problems
- Etc.

Unhappy Path

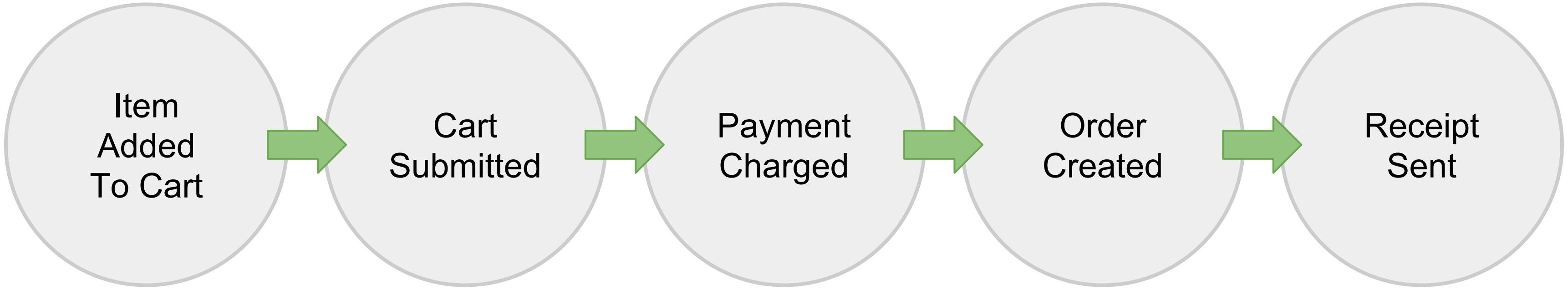


Unhappy Path

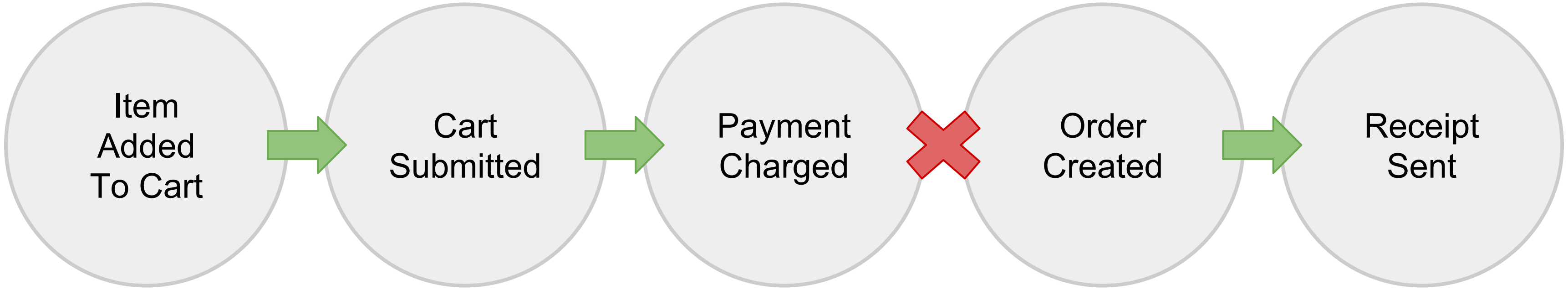


Every step assumes a *successful step* before it

Unhappy Path

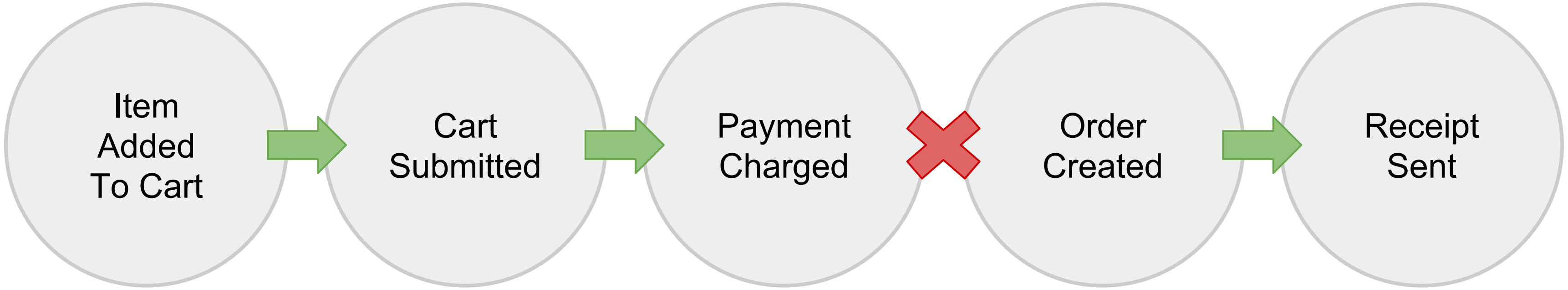


Unhappy Path

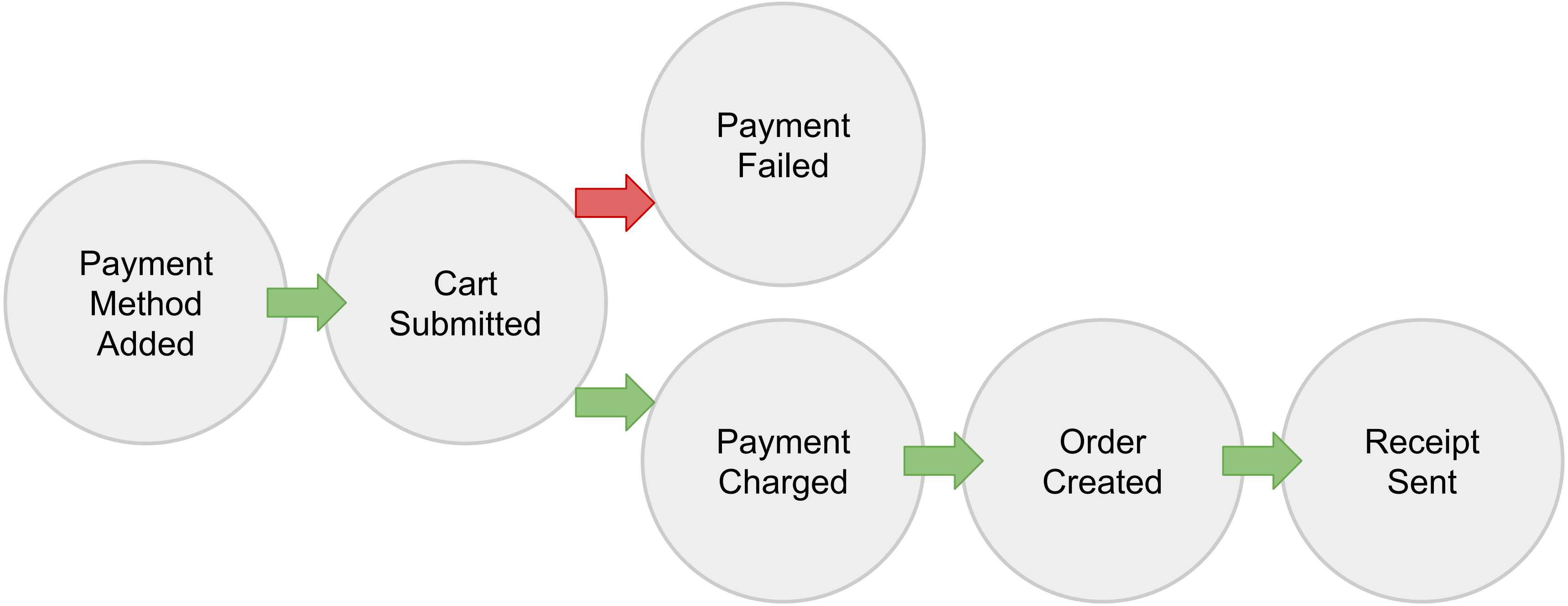


An *unhappy path* is just an *unsuccessful event*

Unhappy Path

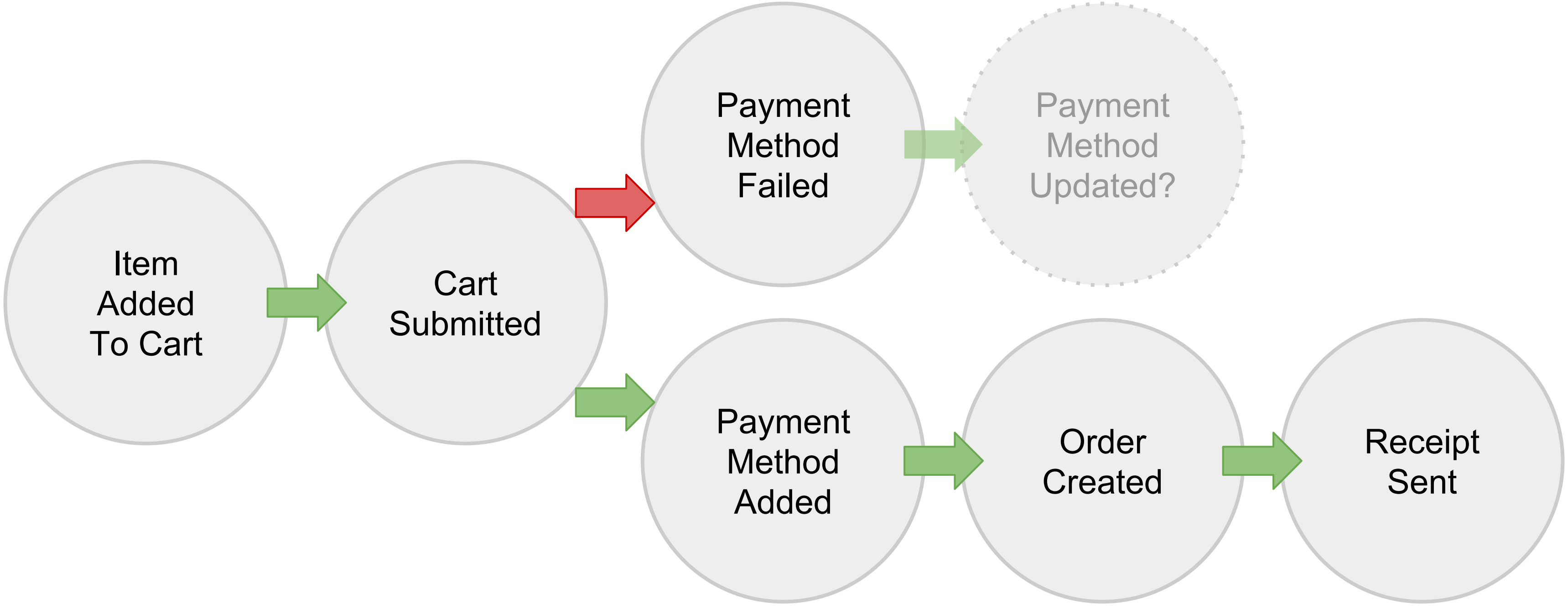


Unhappy Path

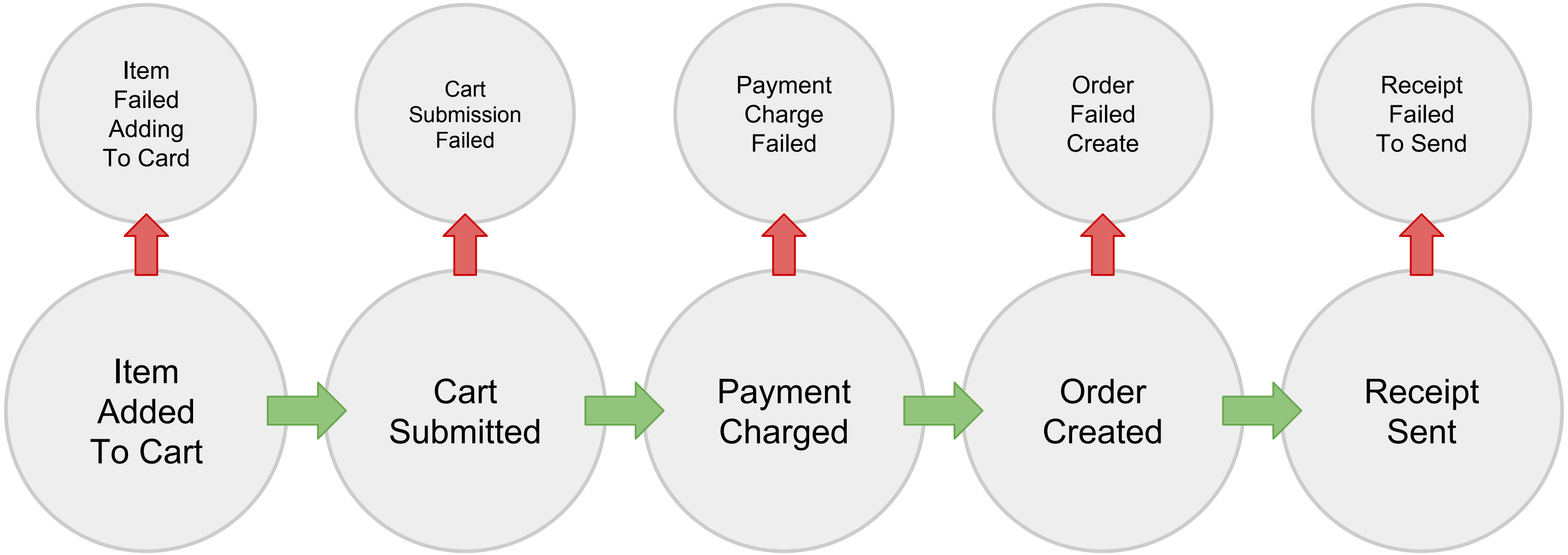


An *unhappy path* should try
to find a way back
to the *happy path*

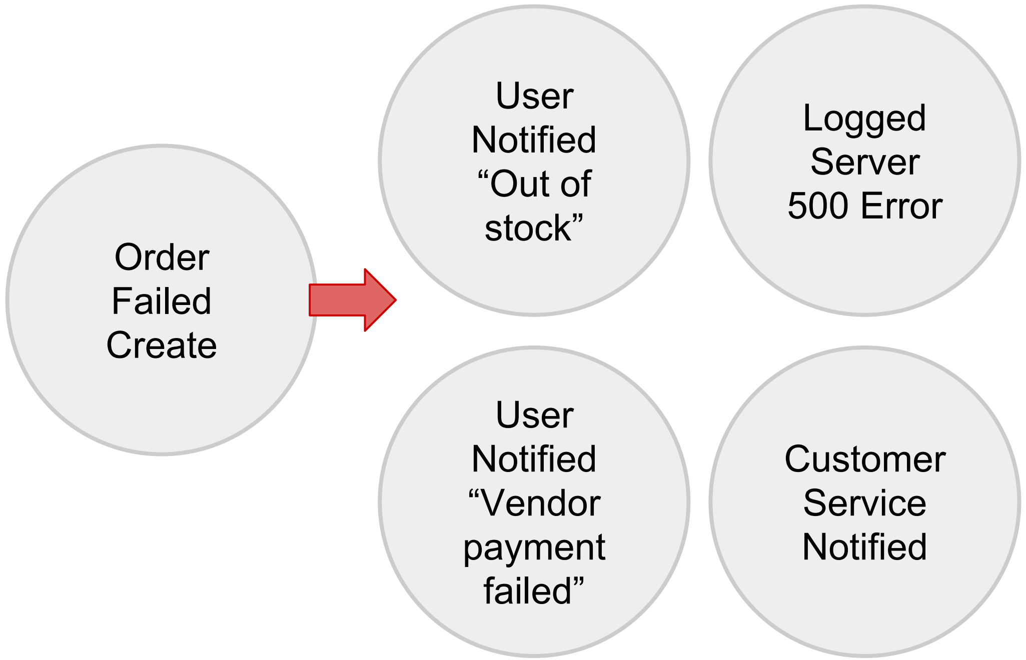
Unhappy Path



Unhappy Path



Unhappy Path


Order Failed Create
User Notified “Out of stock”
Logged Server 500 Error
User Notified “Vendor payment failed”
Customer Service Notified

Applying This Process

Consider “Event Types”

By splitting events into types, you can start to figure out a UI and even infrastructure

- Notifications
- Redirects
- Errors
- Logging
- Etc.

Try “Given, When, Then”

Given, an ad is visible

When, a user clicks on an ad

Then, debit the ad budget

Given, the user is logged in

When, the cart is submitted

Then, create an order

Given: the user is logged in, the user has a valid payment method

When: the cart is submitted, the payment succeeds

Then: create an order, send a receipt, notify the vendor, decrement the inventory

In Summation

Know Your Goals

Decide what everyone agrees is the most important task for the entire project to accomplish

- Reduce noise
- Align the team
- Align the business

Understand > Experiment

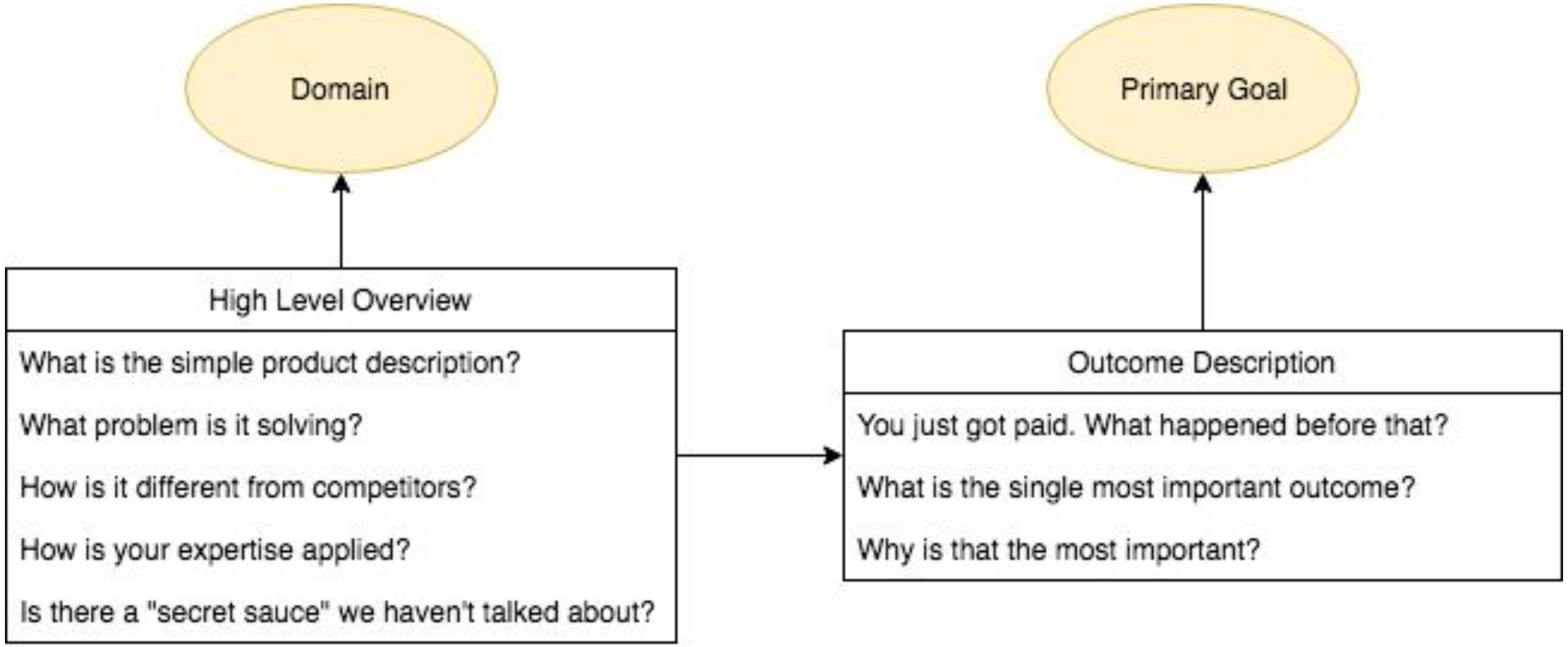
Knowing how you want things to work before the project is developed and designed will save you rewrites

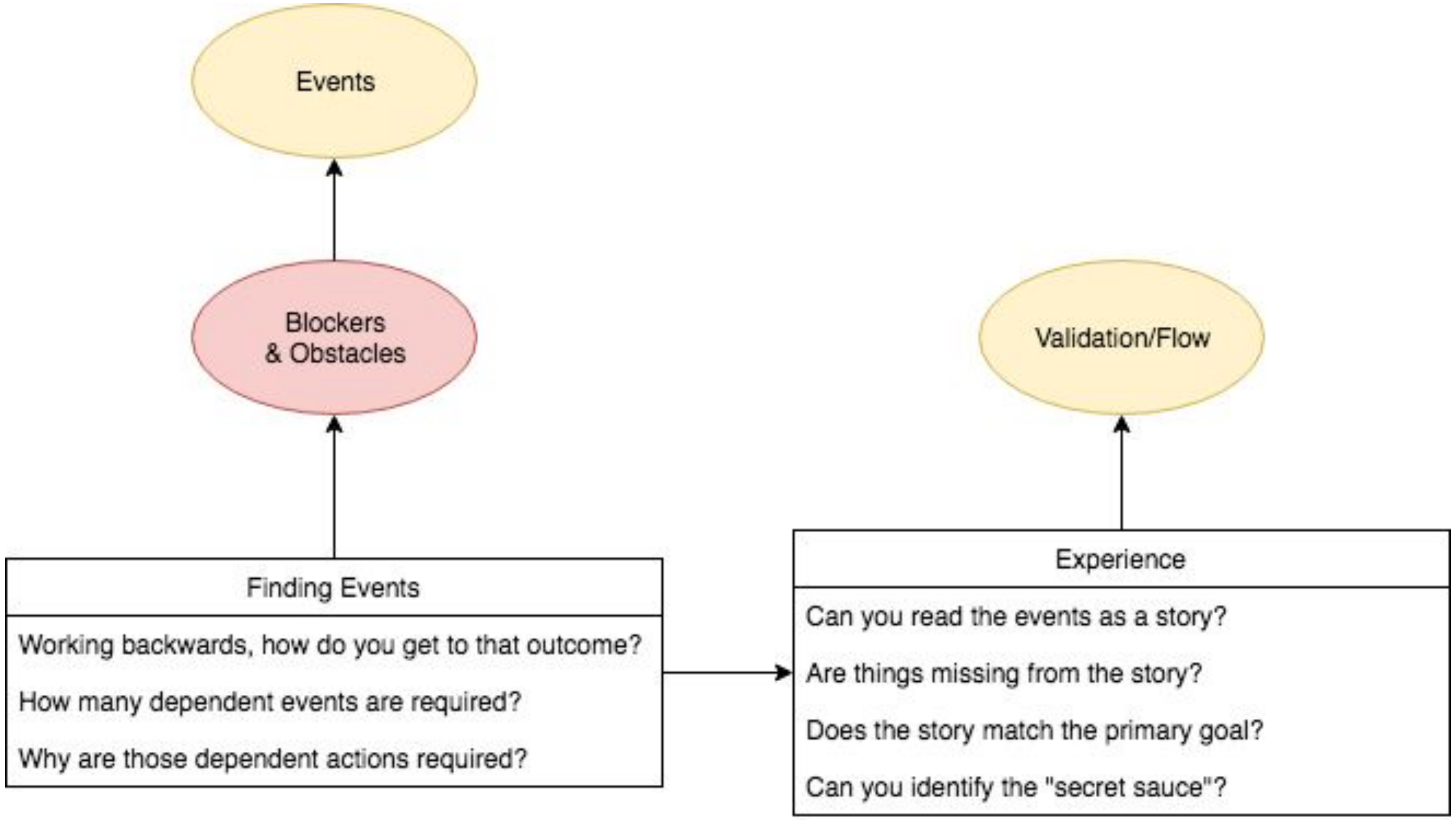
- Planning is cheap
- Avoid rewrites
- Acknowledge your sacrifices

Use Event Design

Thinking in events forces flexibility, applies to frontend and backend development, makes it easy to write tests and tasks

- Only design/develop required features
- Minimal features
- Maximum viability
- Easily translates to code, tests, and tasks


Domain
Primary Goal
High Level Overview
What is the simple product description?
What problem is it solving?
How is it different from competitors?
How is your expertise applied?
Is there a "secret sauce" we haven't talked about?
Outcome Description
You just got paid. What happened before that?
What is the single most important outcome?
Why is that the most important?


Events
Blockers & Obstacles
Validation/Flow
Finding Events
Working backwards, how do you get to that outcome?
How many dependent events are required?
Why are those dependent actions required?
Experience
Can you read the events as a story?
Are things missing from the story?
Does the story match the primary goal?
Can you identify the "secret sauce"?

Easy Right?

That’s it!

Thank You

James Doyle
@james2doyle