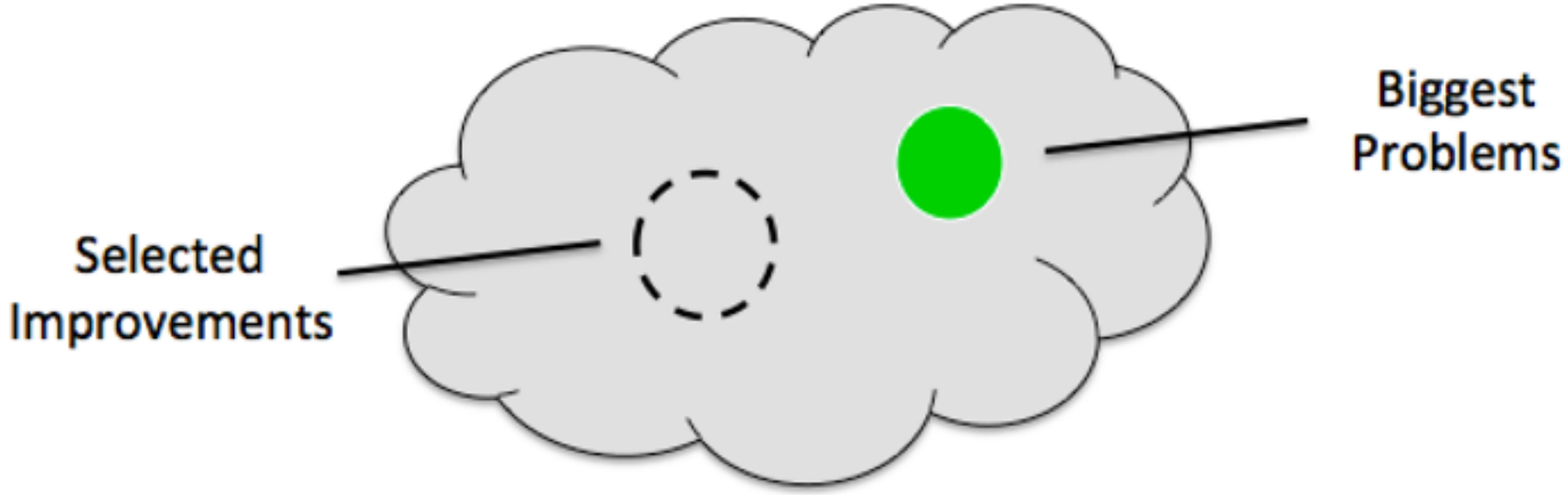
Top 5 Reasons Why

Improvement Efforts

FAIL



NEWIRON
From Vendor to Partner

 Creation Date: 10/05/2015

Janelle Klein
CTO, New Iron, Austin, TX

For the last 5 years...

Idea Flow
Learning Framework

Data-Driven Process for Software Mastery

This Talk...

Why is visibility critical to success?

(It's why improvement efforts fail)

My Story...

We were trying to do
all the **“right”** things.

We were building a

factory automation system...

We shipped to production...

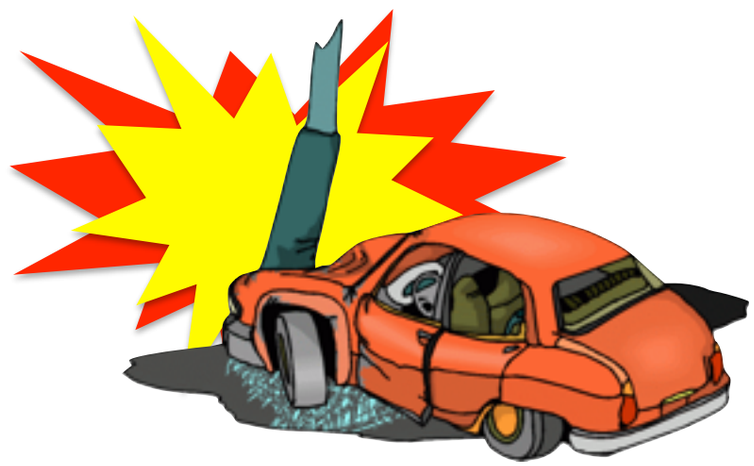
We shipped to production... **(again)**

We couldn’t reproduce the problem.

We shipped to production... **(AGAIN)**

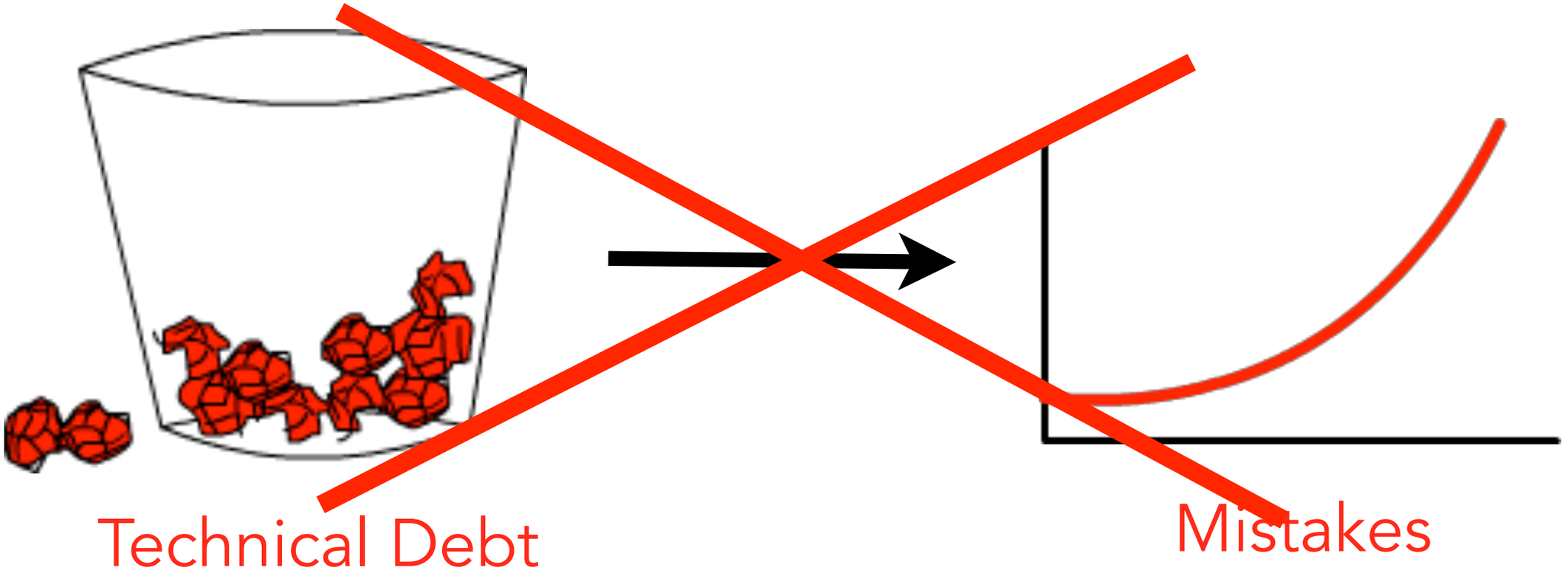
The rollback failed!

Project FAILURE

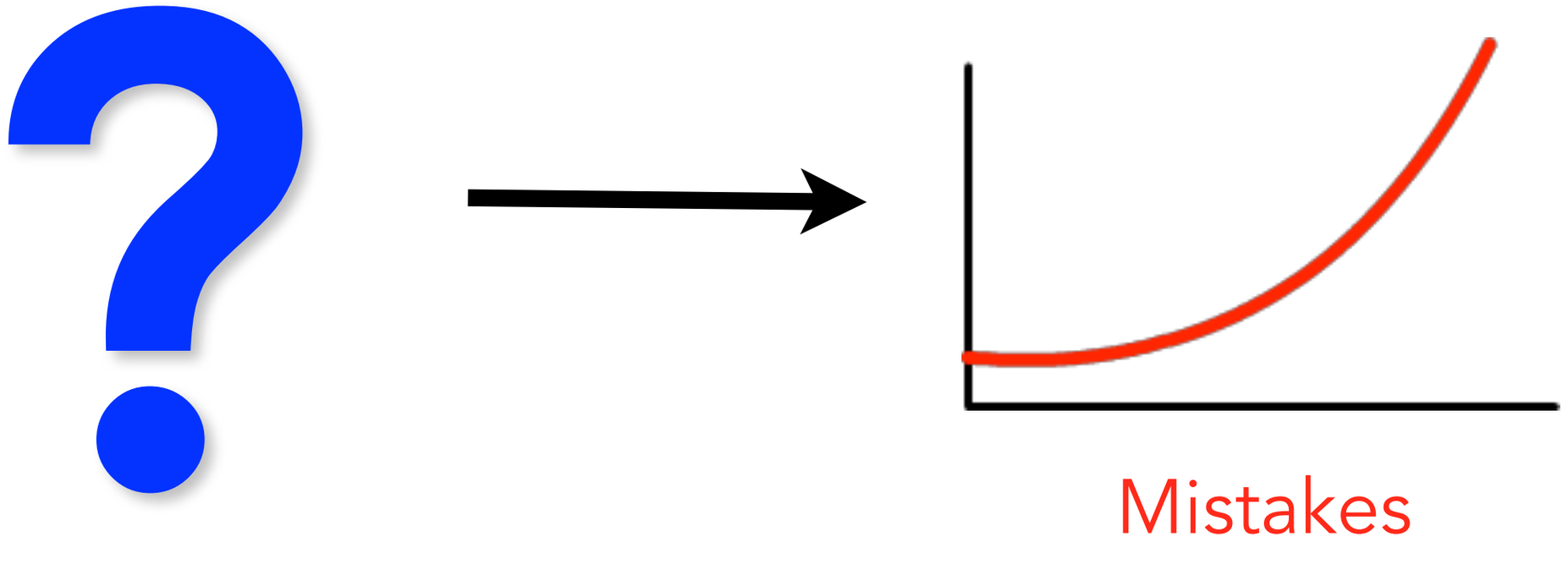


I thought the main obstacle was

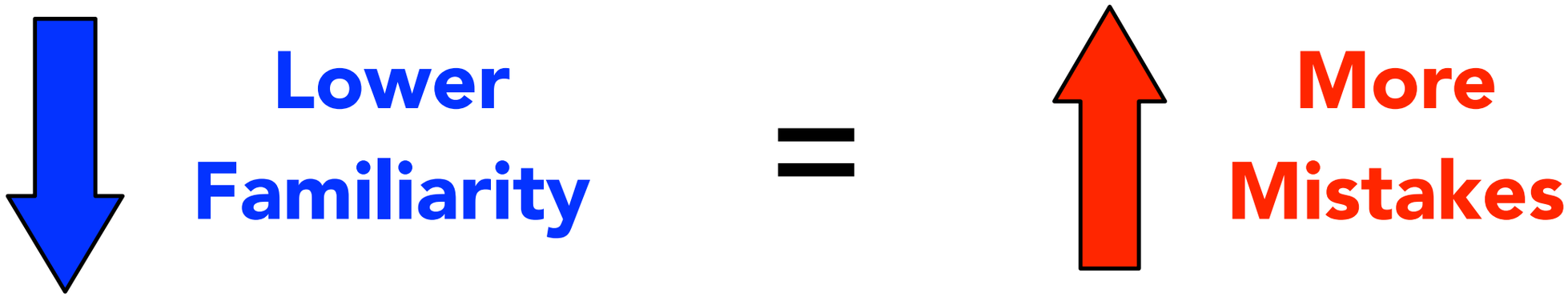
Technical Debt



Most of our mistakes were in the
most well-written parts of the code.

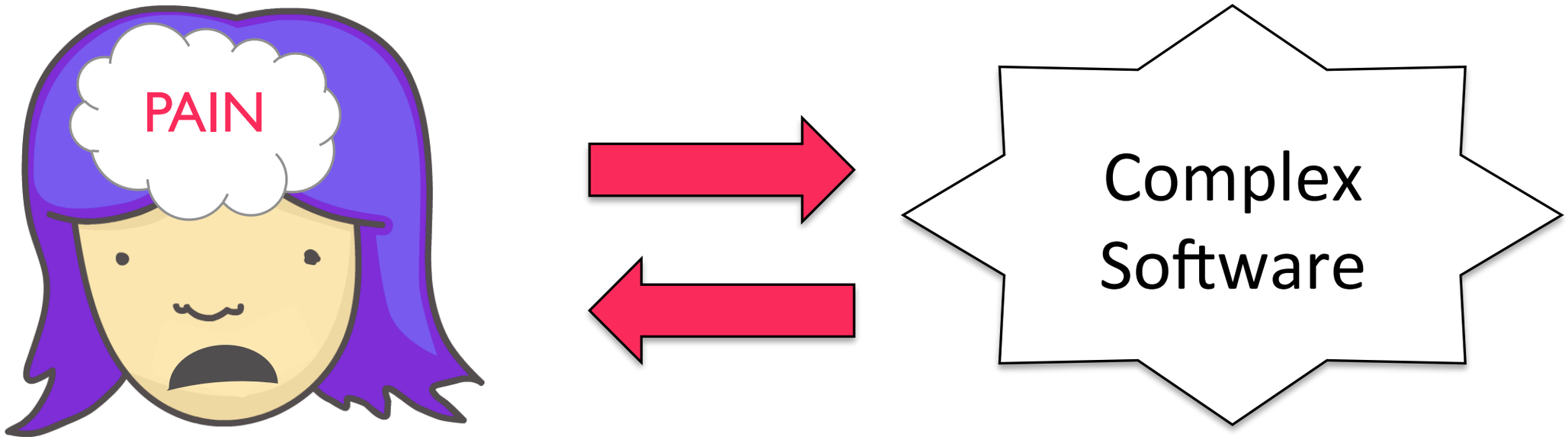


We made significantly more mistakes
in **code that we didn't write** ourselves.



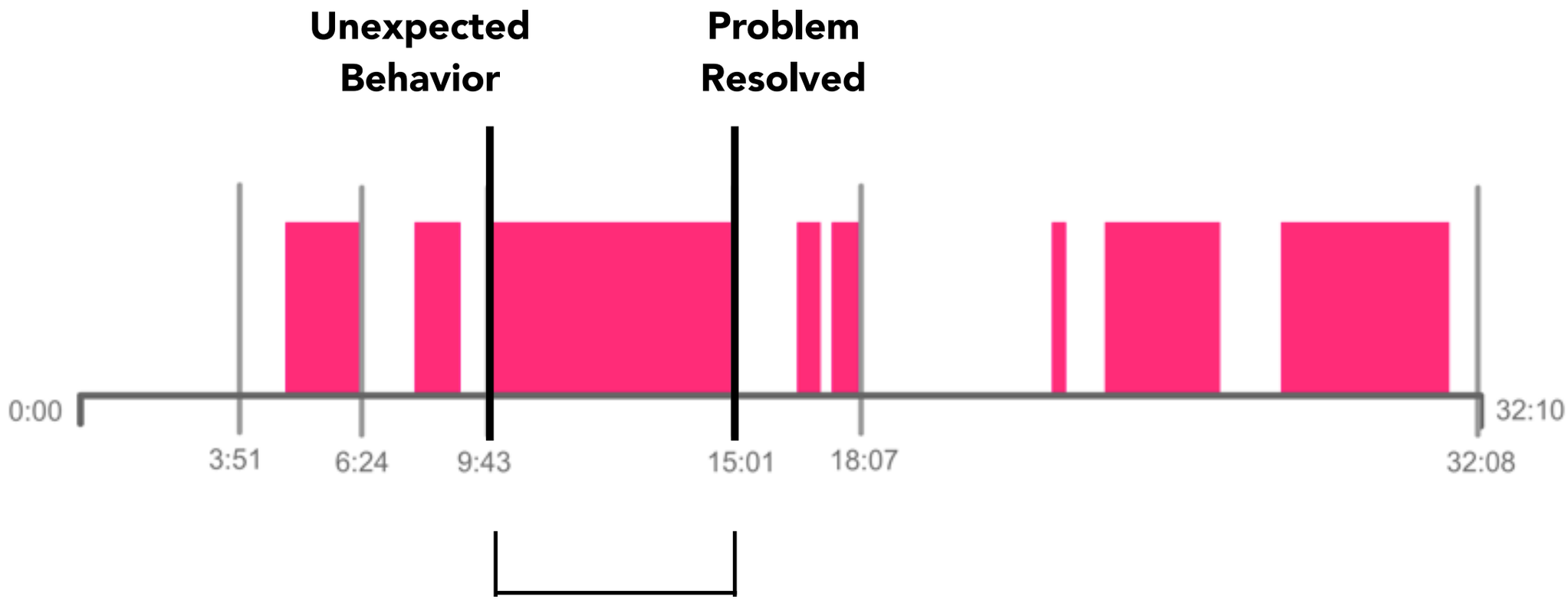
There had to be more to the story...

This is what I knew...



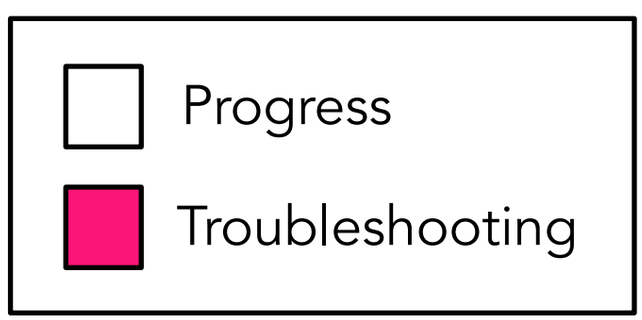
What made development ***feel*** painful?

Tracking Painful *Interaction* with the Code (Friction)

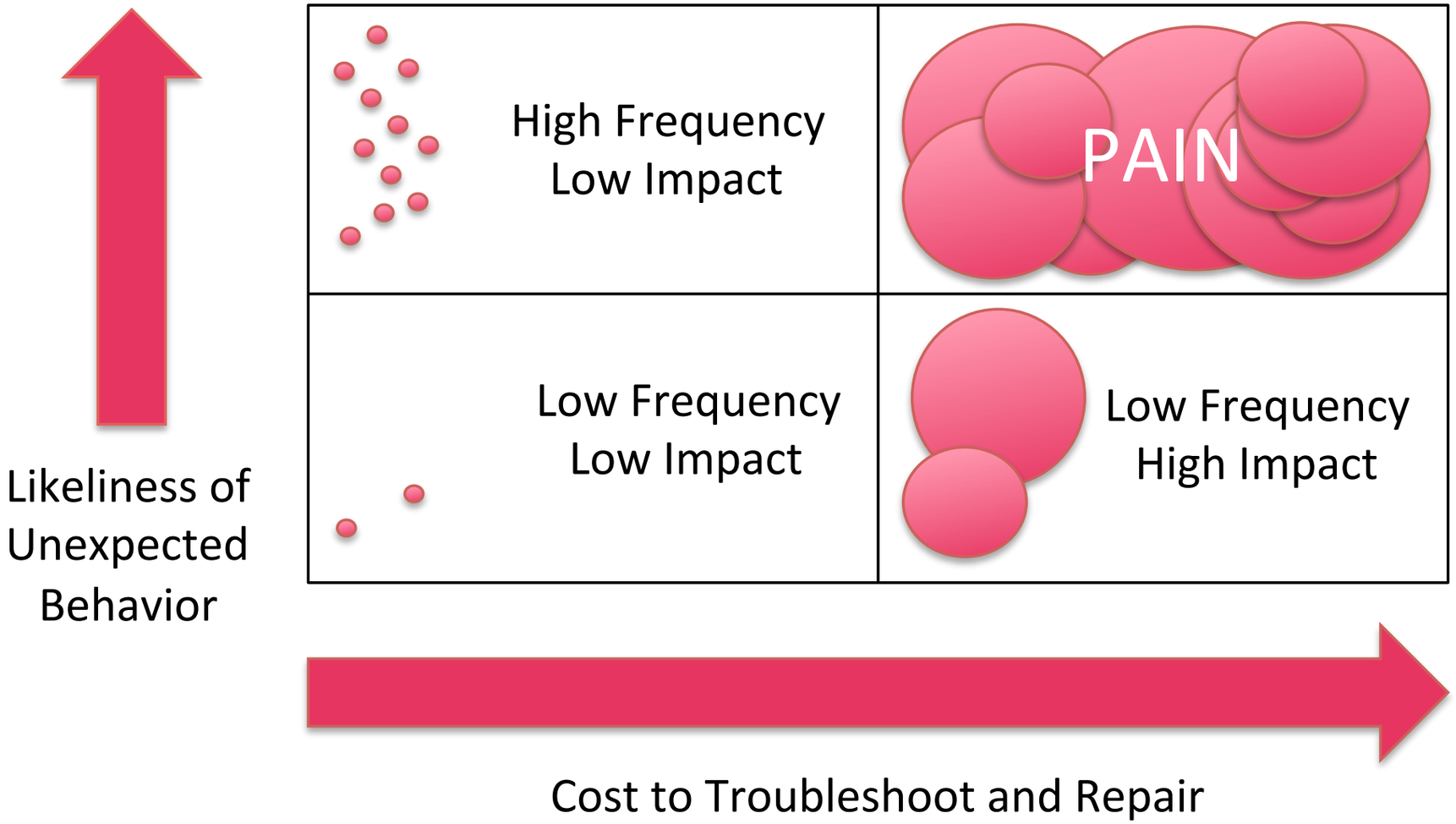


5 hours and **18 minutes** of troubleshooting...

PAINFUL



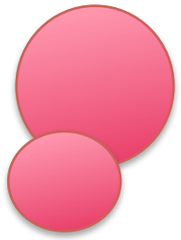
The amount of PAIN was caused by...



What causes PAIN?

What Causes Unexpected Behavior (*likeliness*)?

- Familiarity Mistakes
- Stale Memory Mistakes
- Semantic Mistakes
- Bad Input Assumption
- Tedious Change Mistakes
- Copy-Edit Mistakes
- Transposition Mistakes
- Failed Refactor Mistakes
- False Alarm



What Makes Troubleshooting Time-Consuming (*impact*)?

- Non-Deterministic Behavior
- Ambiguous Clues
- Lots of Code Changes
- Noisy Output
- Cryptic Output
- Long Execution Time
- Environment Cleanup
- Test Data Creation
- Using Debugger

Most of the pain was caused by *human factors*.

What causes PAIN?

What Causes Unexpected Behavior (*likeliness*)?

Familiarity Mistakes

Stale Memory Mistakes

Semantic Mistakes

Bad Input Assumption

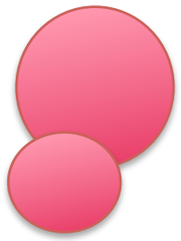
Tedious Change Mistakes

Copy-Edit Mistakes

Transposition Mistakes

Failed Refactor Mistakes

False Alarm



What Makes Troubleshooting Time-Consuming (*impact*)?

Non-Deterministic Behavior

Ambiguous Clues

Lots of Code Changes

Noisy Output

Cryptic Output

Long Execution Time

Environment Cleanup

Test Data Creation

Using Debugger

Most of the pain was caused by *human factors*.

What causes PAIN?

What Causes Unexpected Behavior (*likeliness*)?

Familiarity Mistakes

Stale Memory Mistakes

Semantic Mistakes

Bad Input Assumption

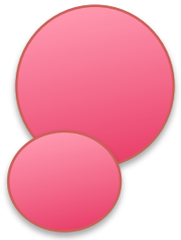
Tedious Change Mistakes

Copy-Edit Mistakes

Transposition Mistakes

Failed Refactor Mistakes

False Alarm



What Makes Troubleshooting Time-Consuming (*impact*)?

Non-Deterministic Behavior

Ambiguous Clues

Lots of Code Changes

Noisy Output

Cryptic Output

Long Execution Time

Environment Cleanup

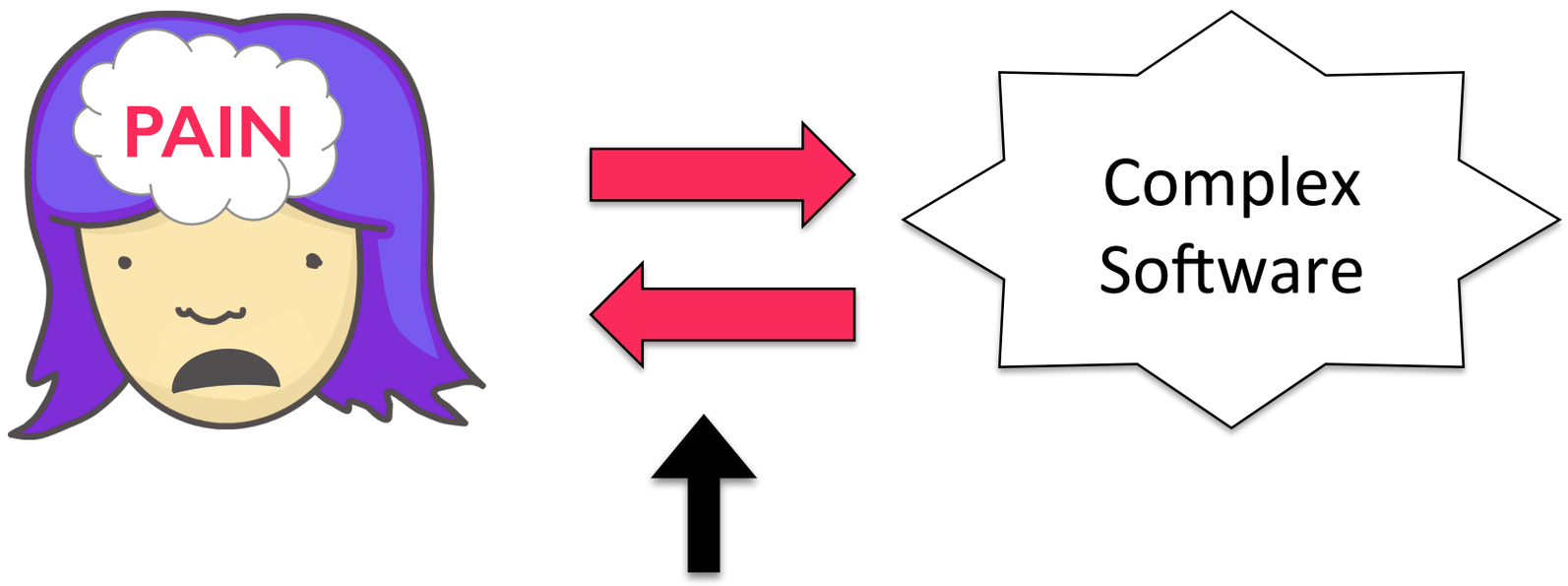
Test Data Creation

Using Debugger

Most of the pain was caused by *human factors*.

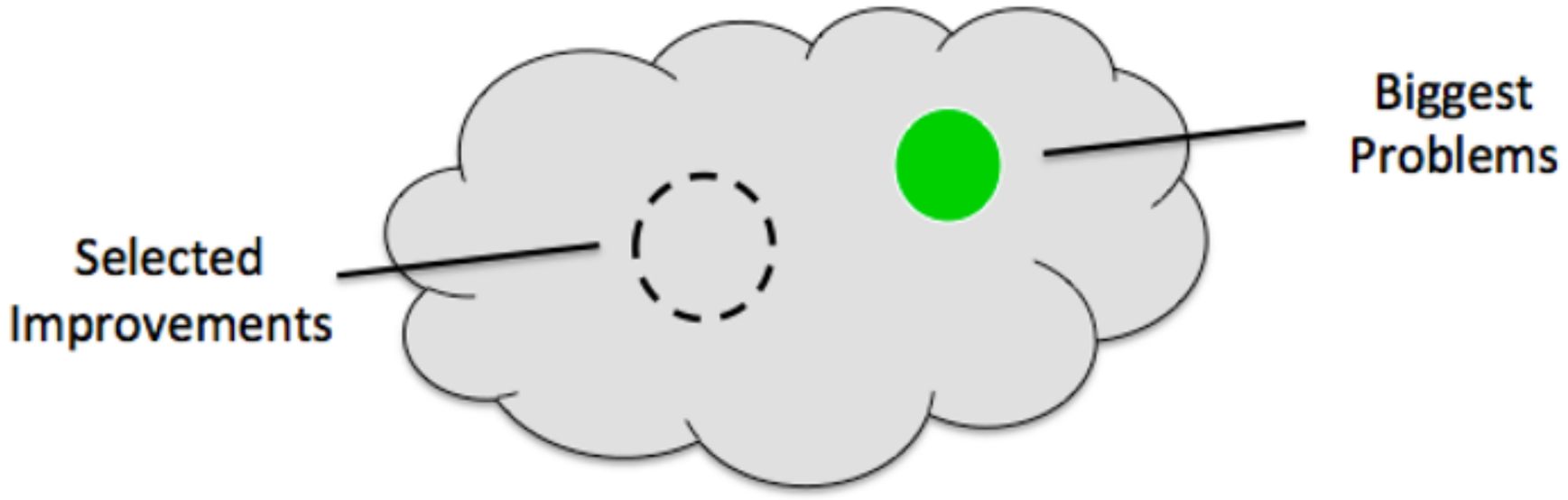
PAIN occurs during the ***process*** of understanding and extending the software

Not the Code.



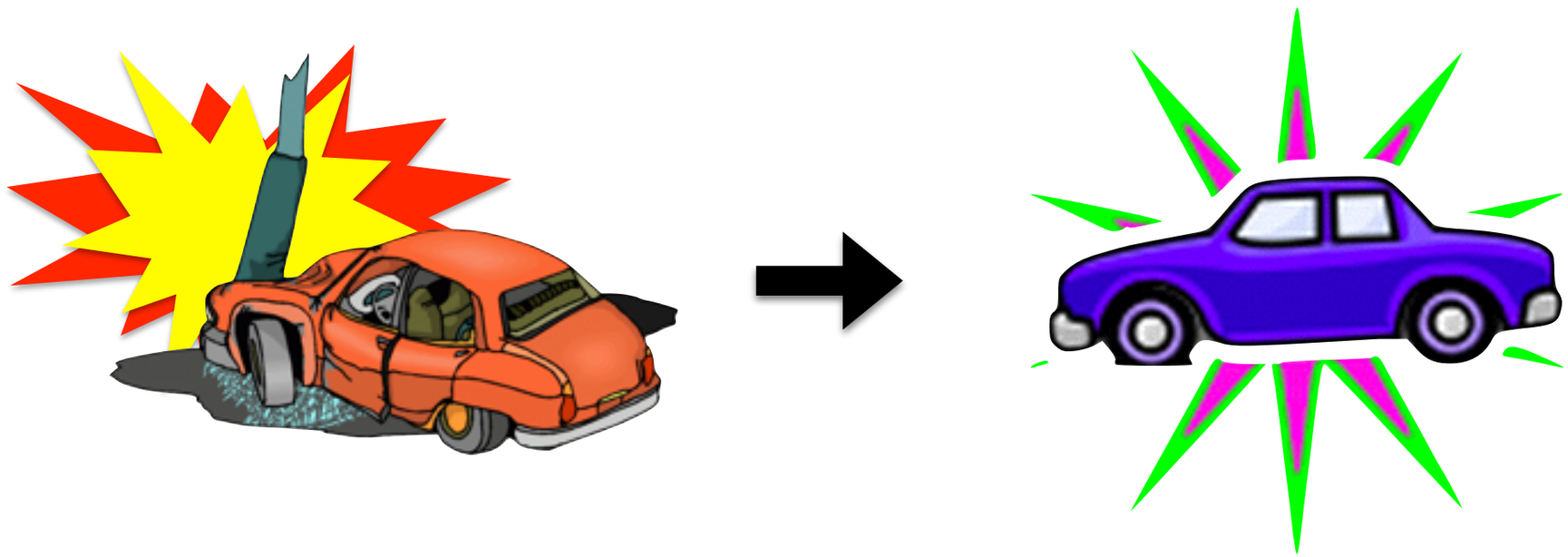
Optimize Developer Experience

We were ***trying to improve***,
but we didn't solve the right problems.



We didn't *realize*
that our improvements didn't work!

It took us **2 years** to turn the project around.



We didn’t start over.

The Same Patterns in Consulting...

PAIN

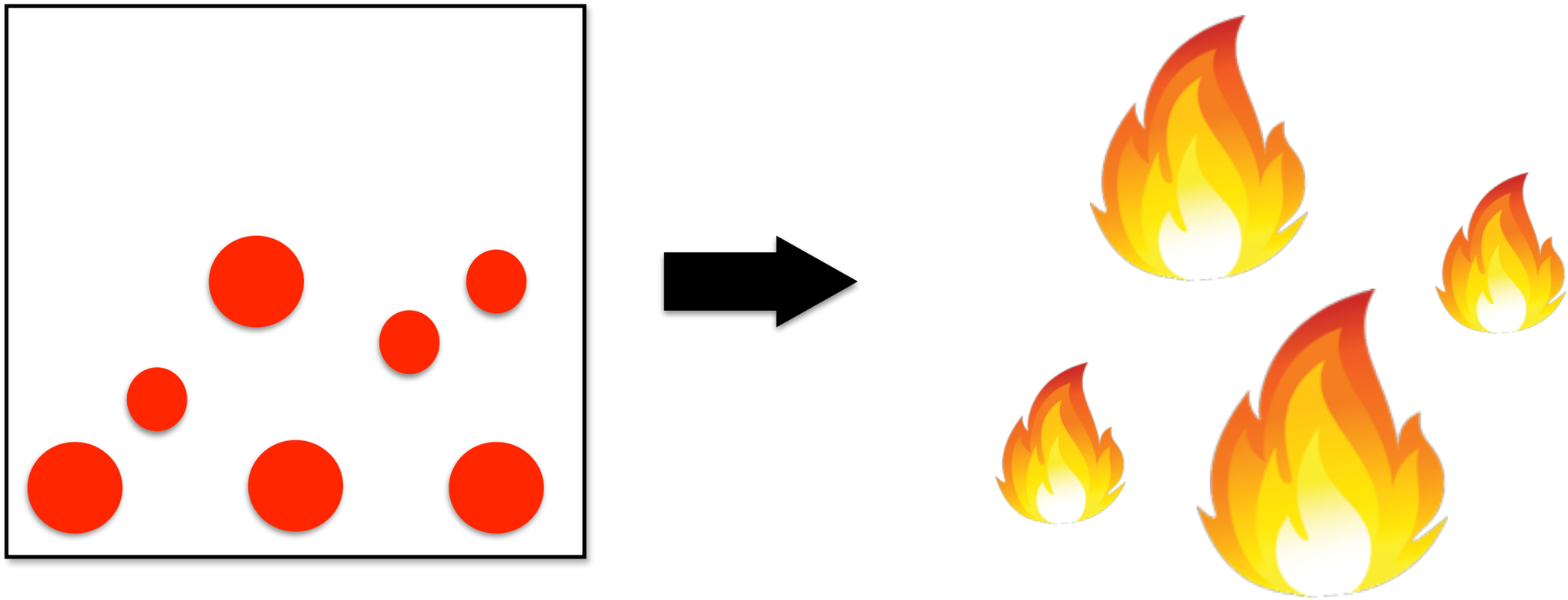
Builds were breaking

Releases were painful

Productivity slowing to a crawl

Begging for time

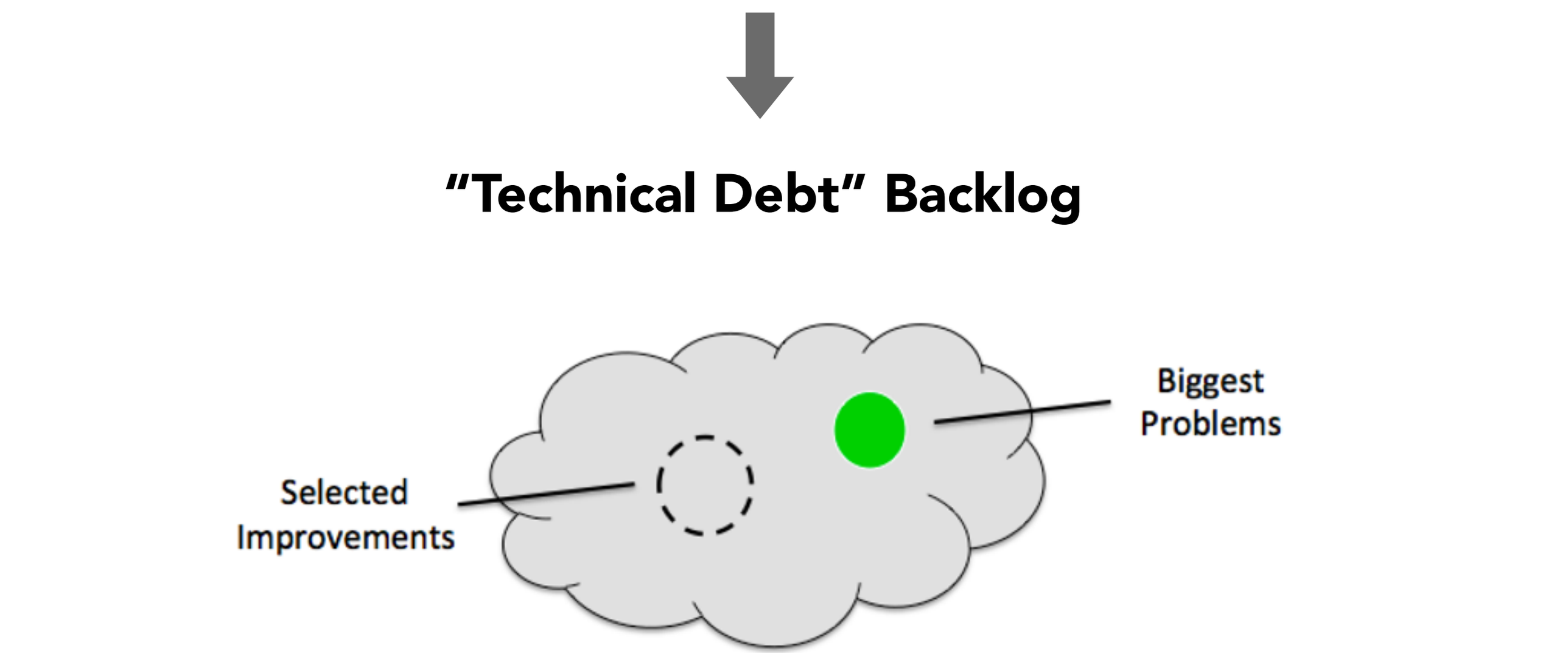
The Team Believed...



Problems Building Up
in the Code

PAIN

Brainstorming a List



"Technical Debt" Backlog

The Improvements didn't make much difference.

Why?

Top 5 Reasons Improvements Fail

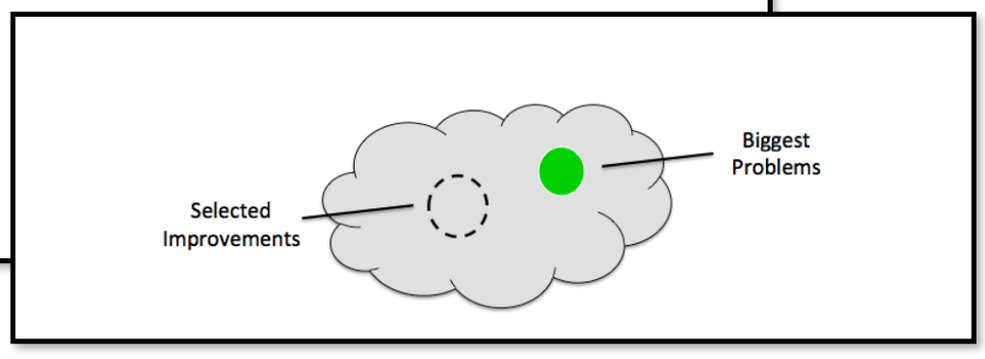
1. We assume best practices will solve the problem.

2. We assume the biggest problems will come to mind.

3. We assume that we understand the problem.

4. We assume that understanding is enough.

5. We don't make the problems visible to management.



Top 5 Reasons Improvements Fail

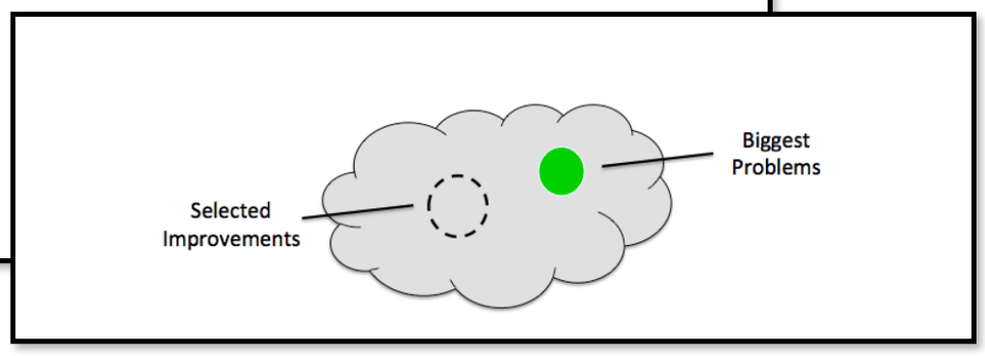
1. We assume best practices will solve the problem.

2. We assume the biggest problems will come to mind.

3. We assume that we understand the problem.

4. We assume that understanding is enough.

5. We don't make the problems visible to management.



Here's What My Team Did.

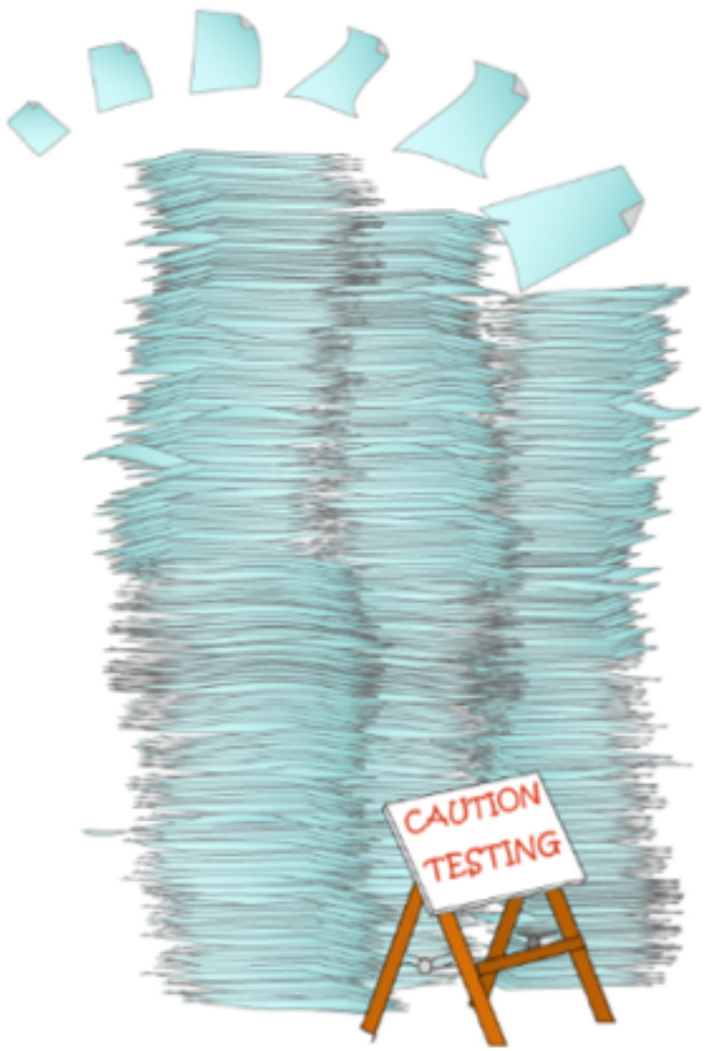
Our regression testing took 3-4 weeks…

Let's *automate* the tests!

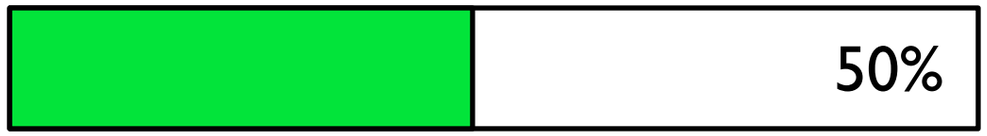
Here's What My Team Did.

Percent Automated:

Here's What My Team Did.



Percent Automated:

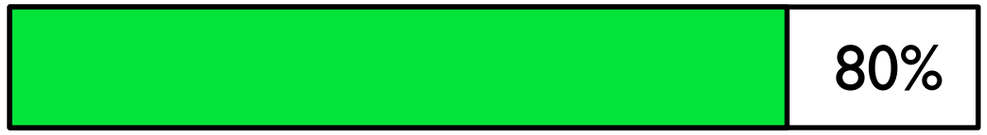


Maintenance
started getting **PAINFUL**…

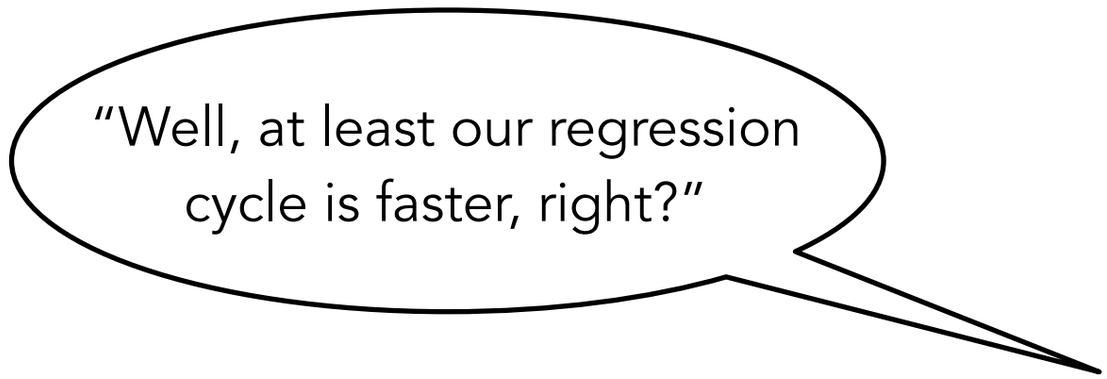
Let's *refactor* the tests!

Here's What My Team Did.

Percent Automated:



It was still really **PAINFUL**…



Our regression cycle *still* took 3-4 weeks!

Here's What My Team Did.

We deleted *everything*...

The automation wasn't solving the problem.

How did we solve the quality problem?

What are the *risks* in this release?

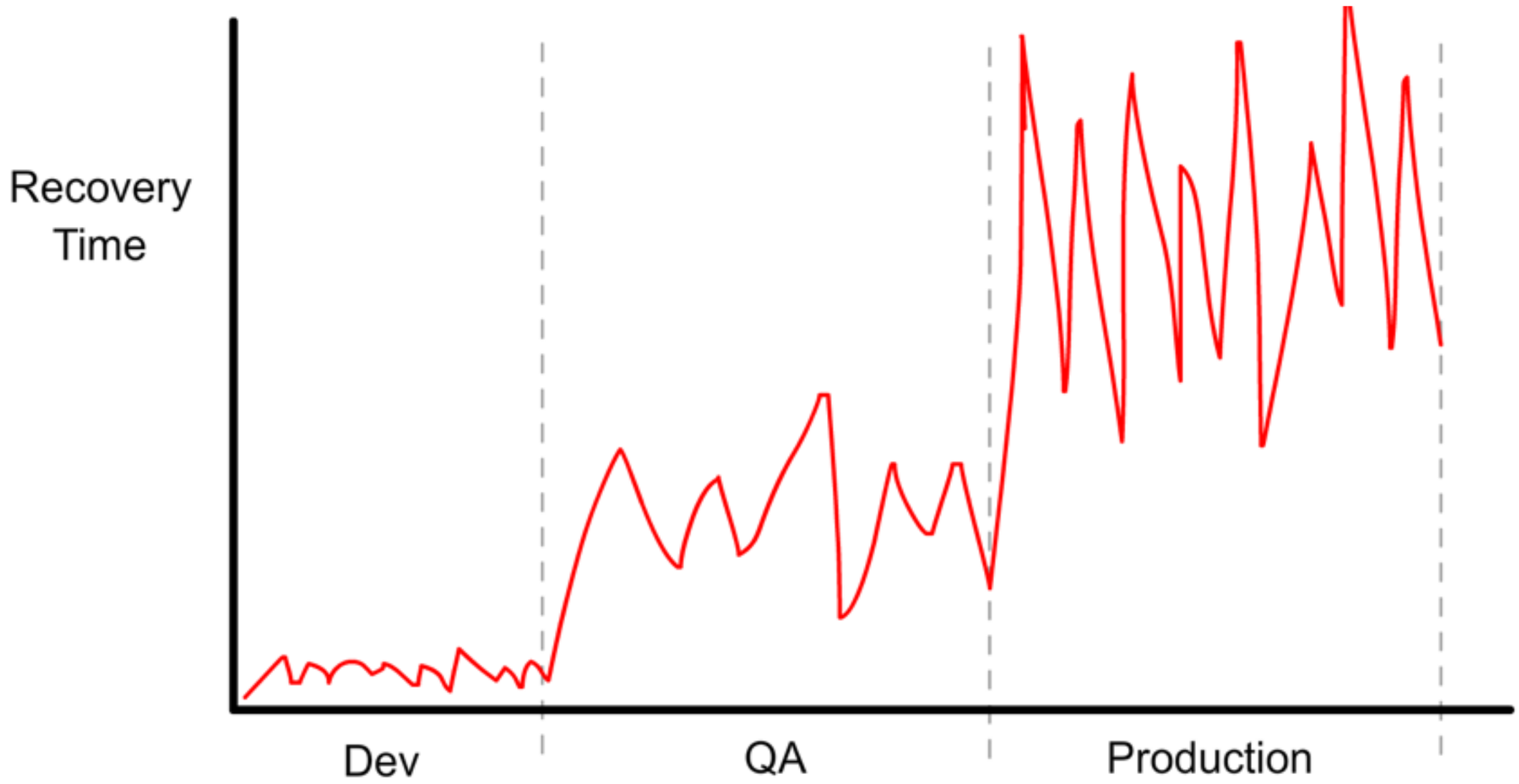
How can we *mitigate* the risks?

Handful of
manual tests

+

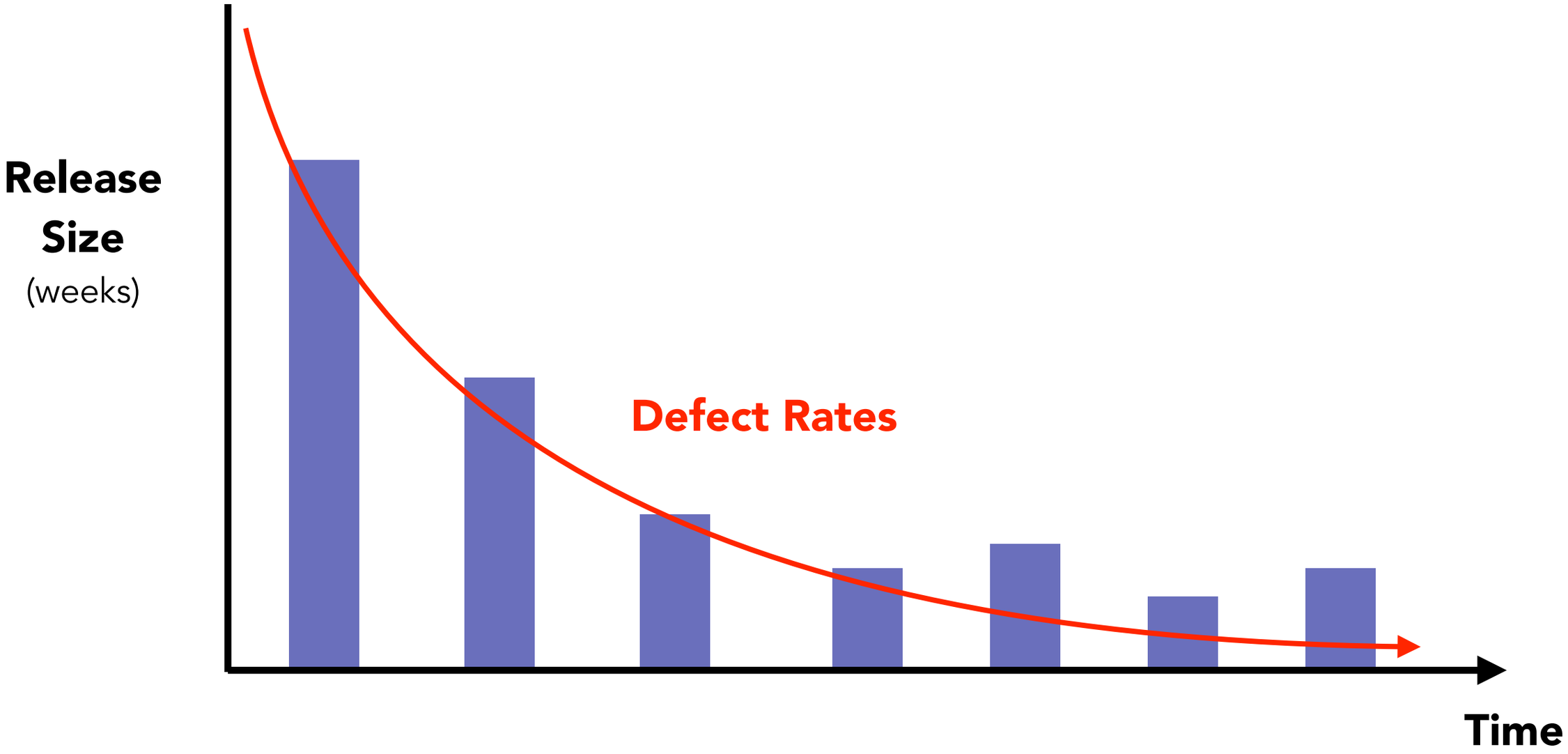
Custom
Tools

Rushing at the Deadline *Escalates* Costs



Switched to Fixed-scope “as small as possible” releases

We Regained Control Over Quality
By ***Dropping*** the Deadlines



Sometimes Best Practices are the *cause* of the problem

What target are we aiming for?

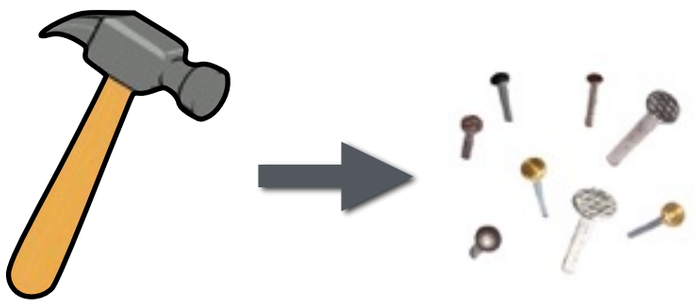
What does “better” really mean?

What does "better" really mean?

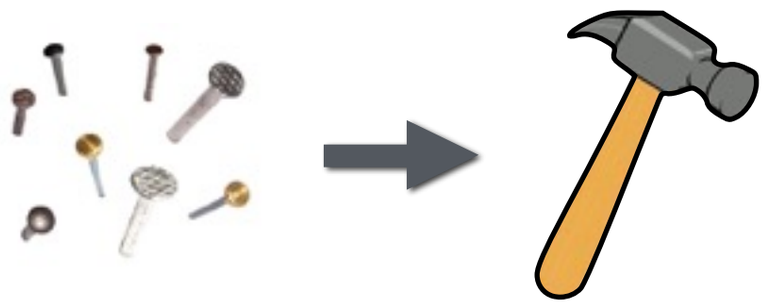
"Better"
following best practices
(solution-focused)

"Better"
solving the problems
(problem-focused)

Start with the Solution



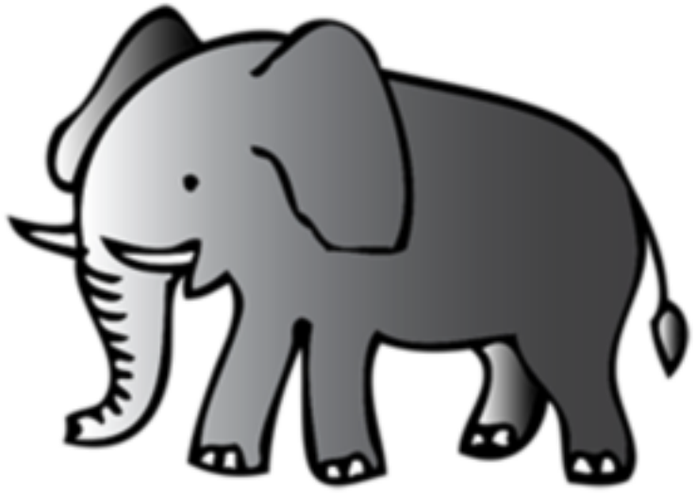
Start with the Problem



Solution-Oriented "Problems"

"What's the biggest problem we need to solve?"

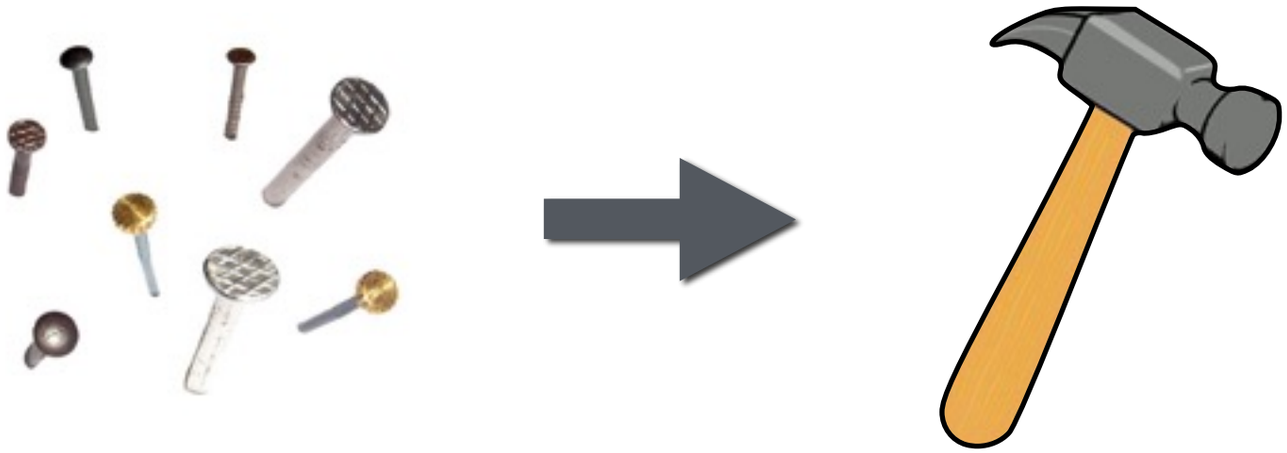
"The problem is we don't have enough test automation!"



Our biggest problem

Broken Target

Lesson Learned: Focus on the Nails



“What *problem* am I trying to solve?”

Top 5 Reasons Improvements Fail

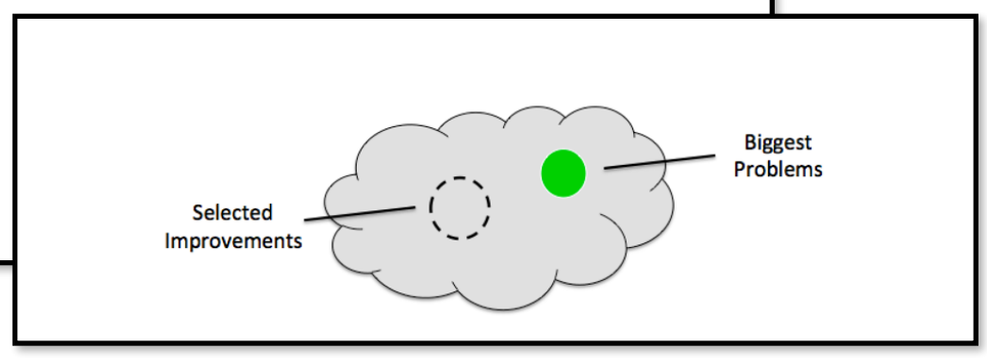
BROKEN TARGET

2. We assume the biggest problems will come to mind.

3. We assume that we understand the problem.

4. We assume that understanding is enough.

5. We don't make the problems visible to management.



Top 5 Reasons Improvements Fail

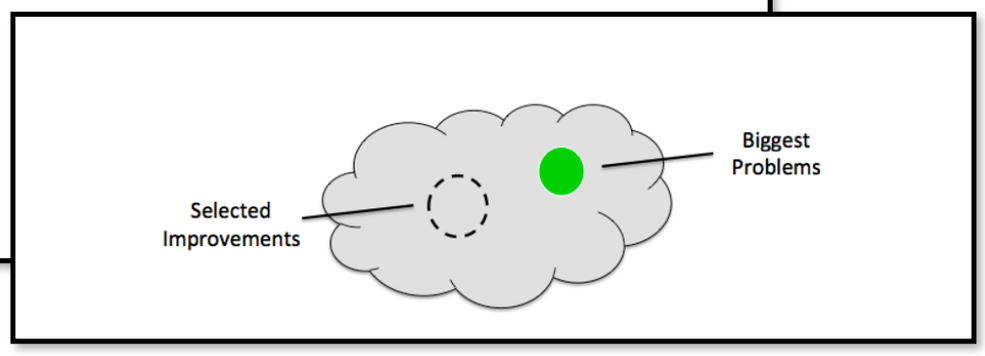
BROKEN TARGET

2. We assume the biggest problems will come to mind.

3. We assume that we understand the problem.

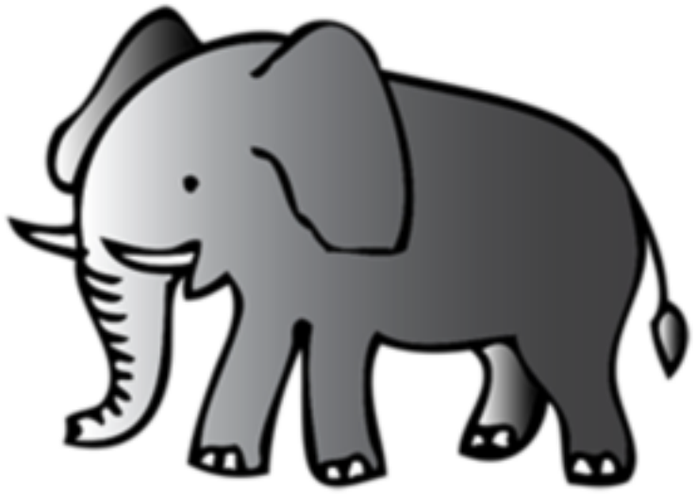
4. We assume that understanding is enough.

5. We don't make the problems visible to management.



Here's what my team did.

"What's the biggest opportunity for improvement?"

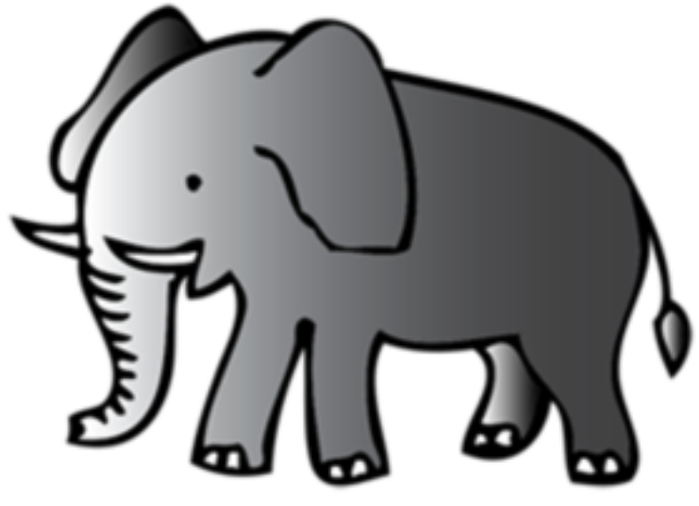


Our biggest problem

Here's what my team did.

"What's the biggest opportunity for improvement?"

"The awful email template engine code!"

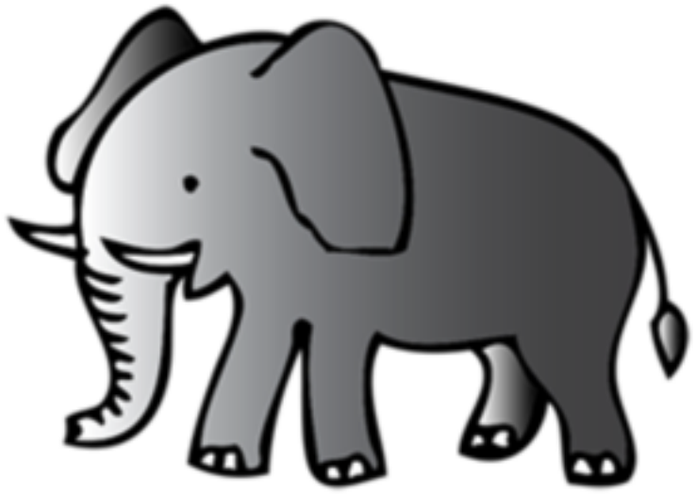


Our biggest problem

Here's what my team did.

"What's the biggest opportunity for improvement?"

"Fill in missing unit tests!"

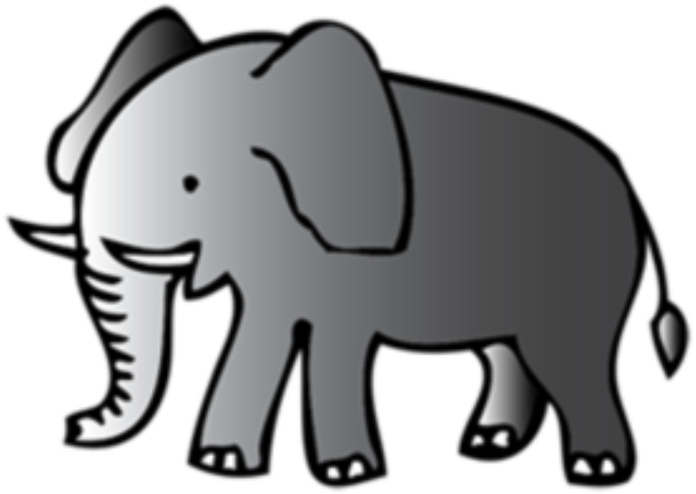


Our biggest problem

Here's what my team did.

"What's the biggest opportunity for improvement?"

"I know how to improve database performance!"

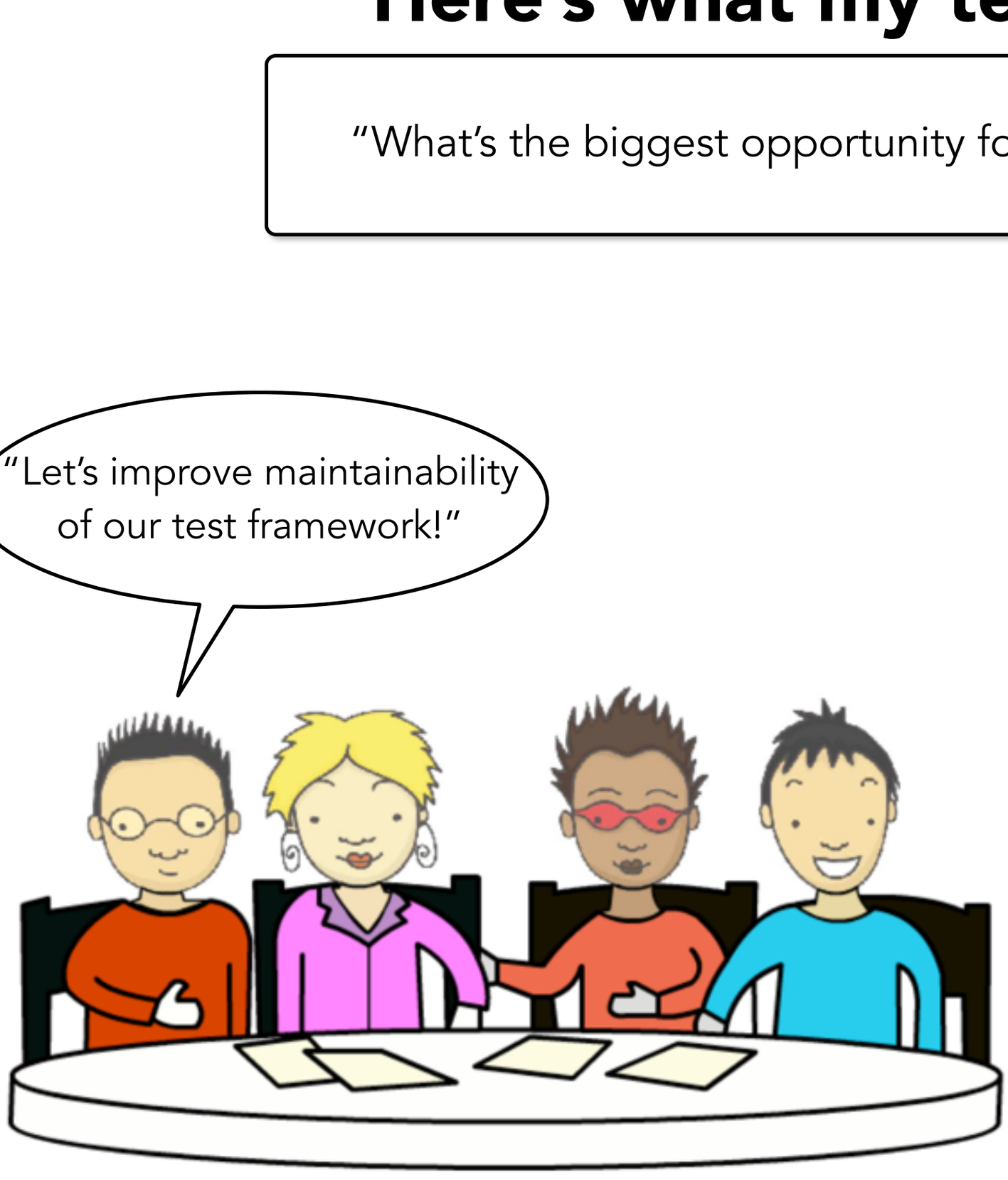


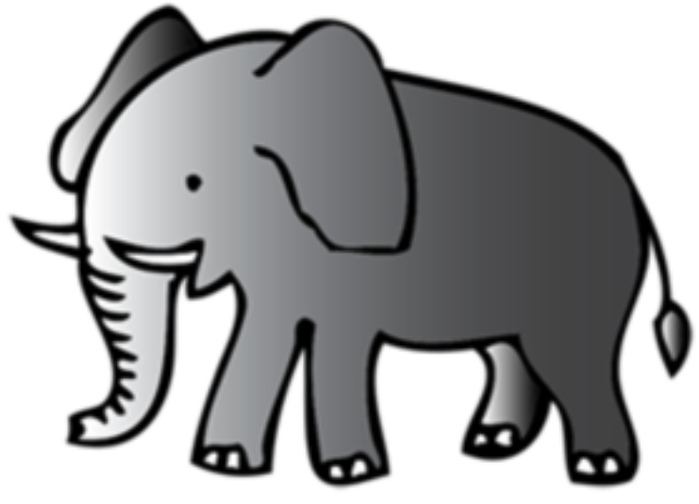
Our biggest problem

Here's what my team did.

"What's the biggest opportunity for improvement?"

"Let's improve maintainability of our test framework!"



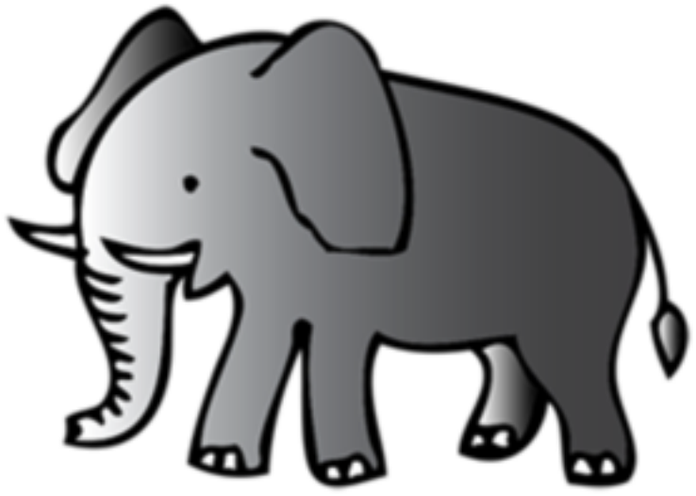


Our biggest problem

Here's what my team did.

"What's the biggest opportunity for improvement?"

Just because a problem **comes to mind**,
doesn't mean it's an important problem to solve.



Our biggest problem

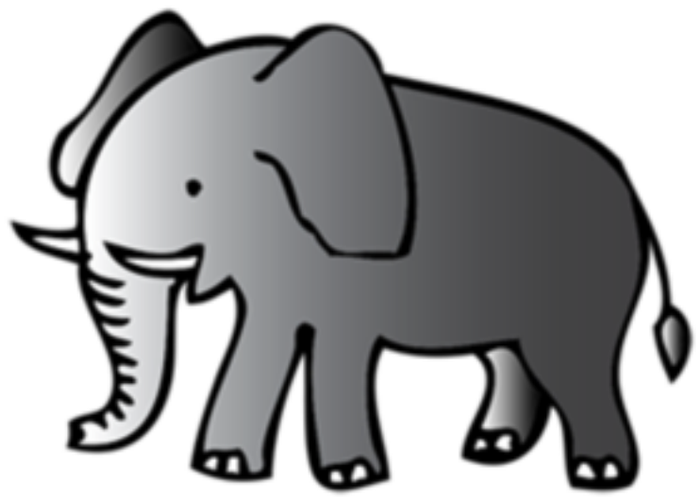
Here's what my team did.

"What's the biggest opportunity for improvement?"

What do I feel the most ***intensely*** about?

Daniel Kahneman

Thinking Fast and Slow



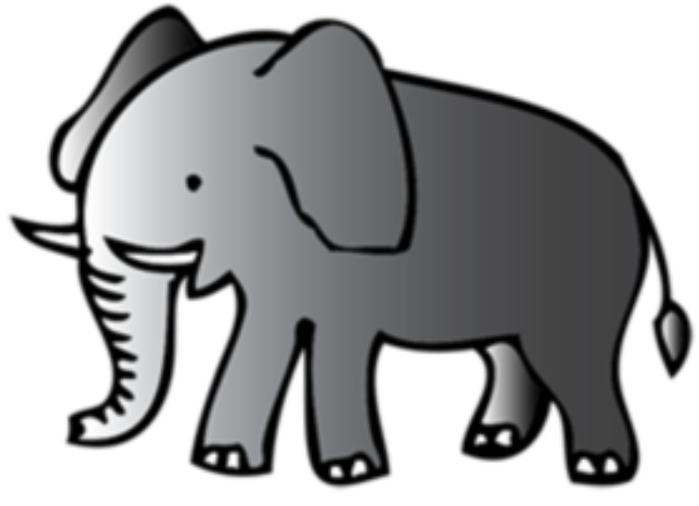
Our biggest problem

Here's what my team did.

"What's the biggest opportunity for improvement?"

Recency Bias

"The awful email template engine code!"



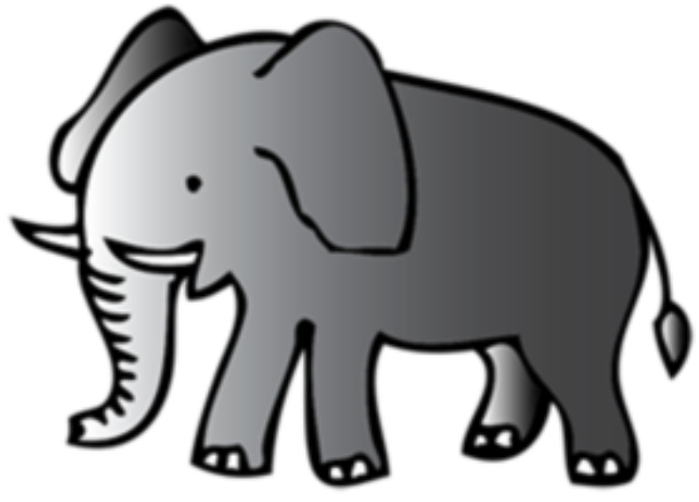
Our biggest problem

Here's what my team did.

"What's the biggest opportunity for improvement?"

Guilt Bias

"Fill in missing
unit tests!"



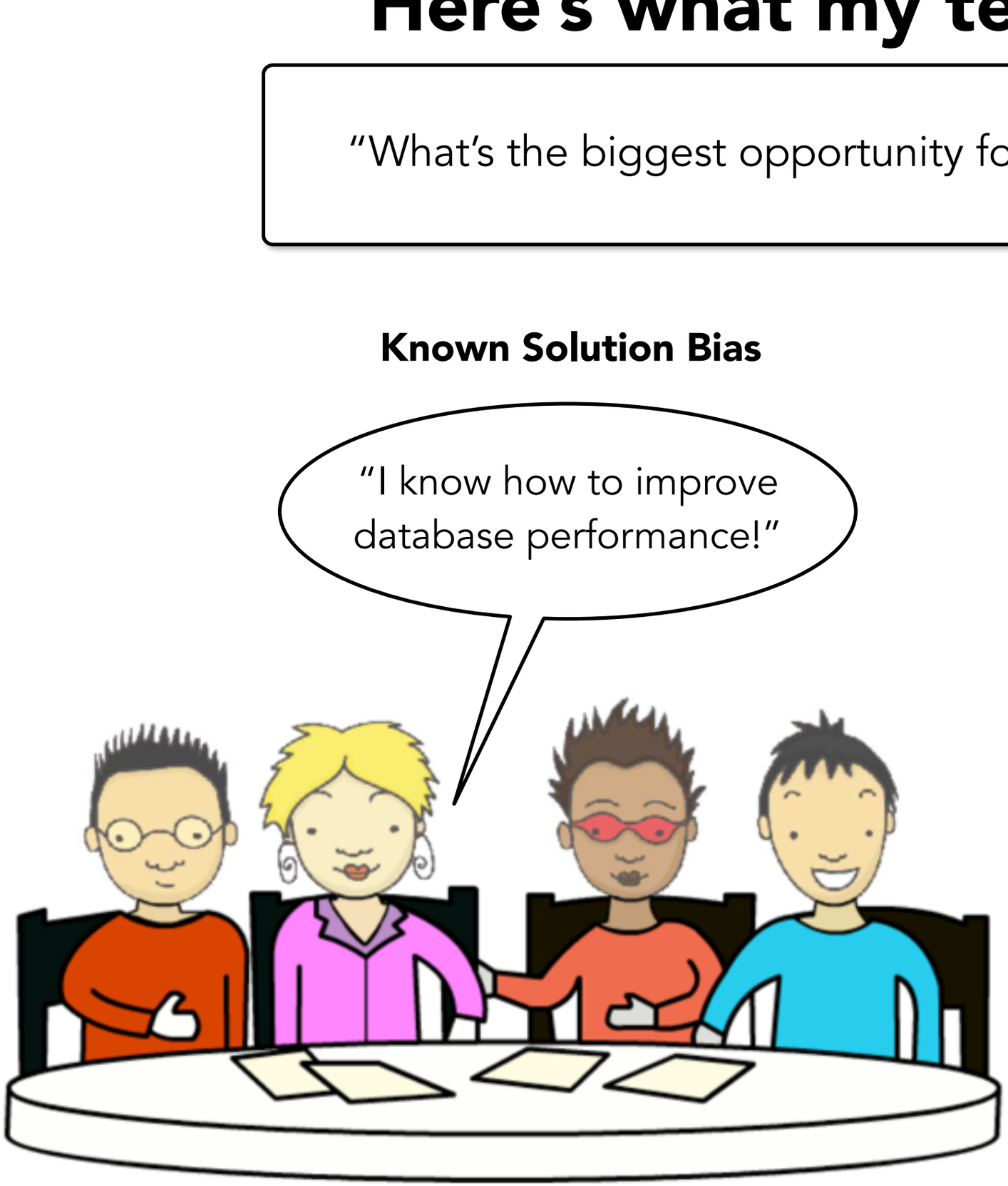
Our biggest problem

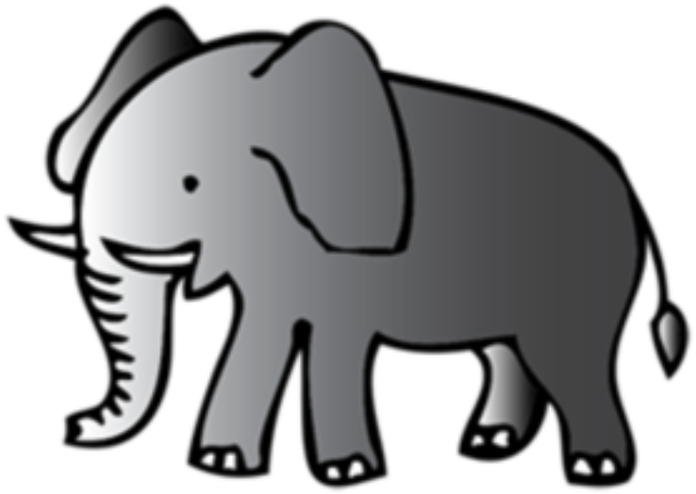
Here's what my team did.

"What's the biggest opportunity for improvement?"

Known Solution Bias

"I know how to improve database performance!"





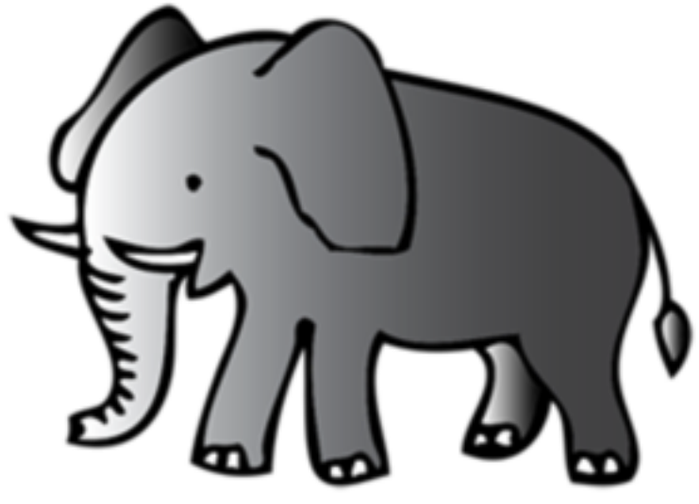
Our biggest problem

Here's what my team did.

"What's the biggest opportunity for improvement?"

Sunk Cost Bias

"Let's improve maintainability of our test framework!"

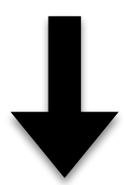


Our biggest problem

Our Perception of Time is WAY OFF

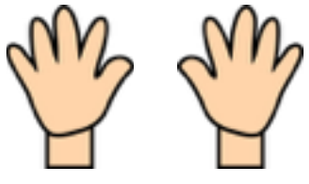
waiting - time goes slow

Execute



Experiment Time

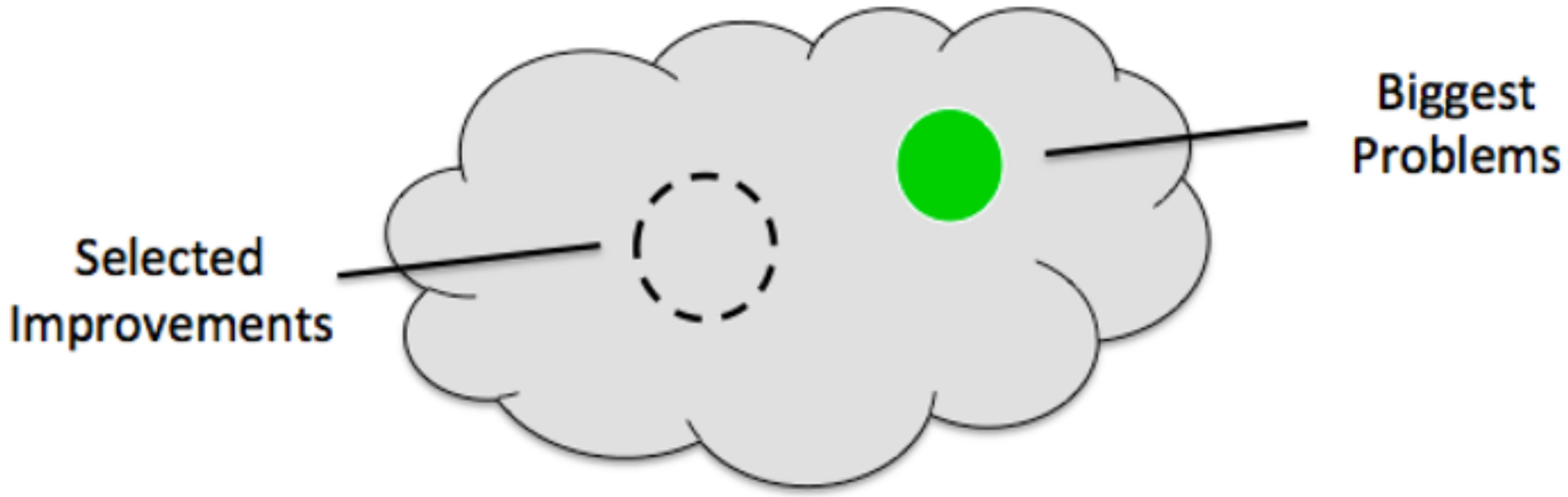
Setup Experiment



Analyze Results &
Decide Next Experiment

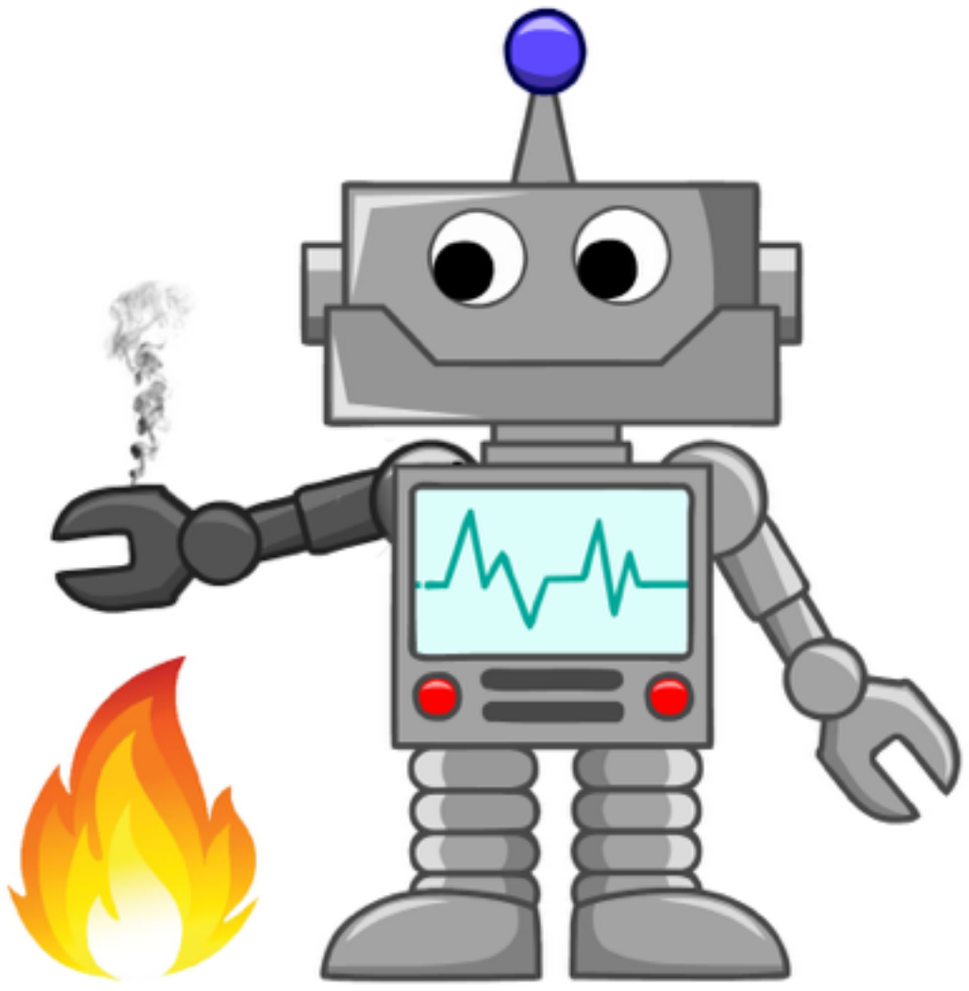
doing - time zooms by

Relying on Gut Feel:

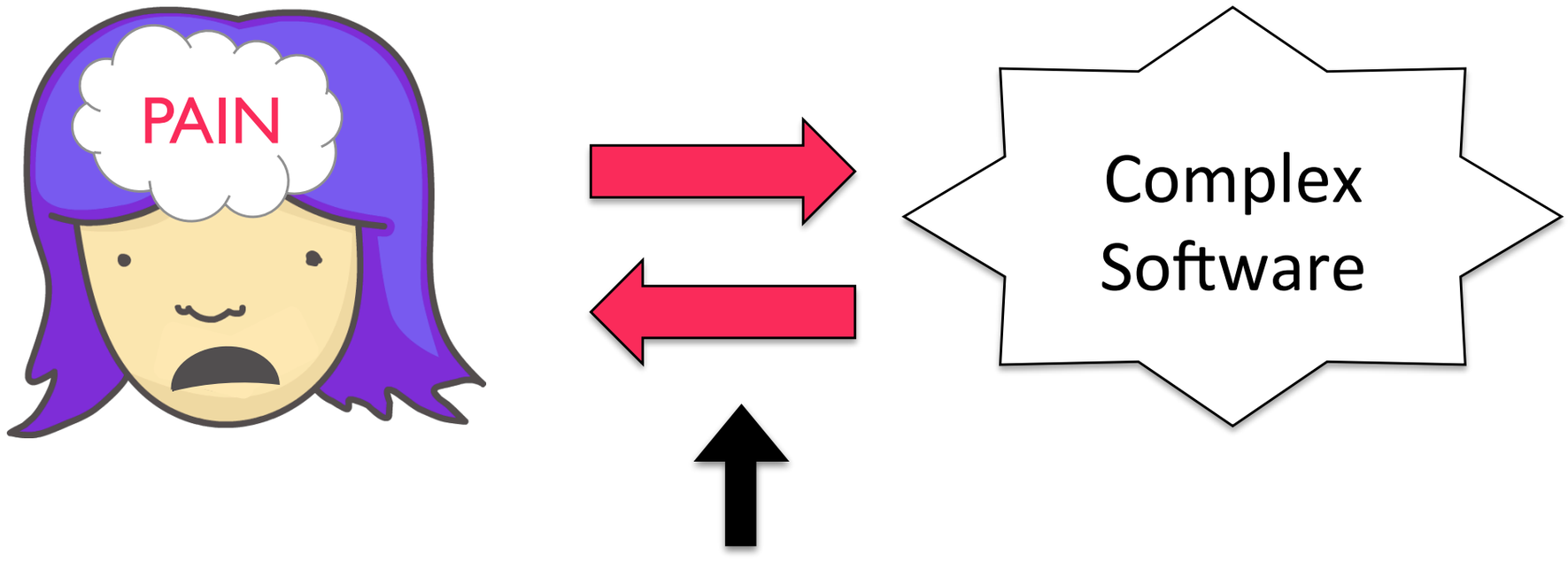


Our gut feel is way off!!!

Install a Pain Sensor



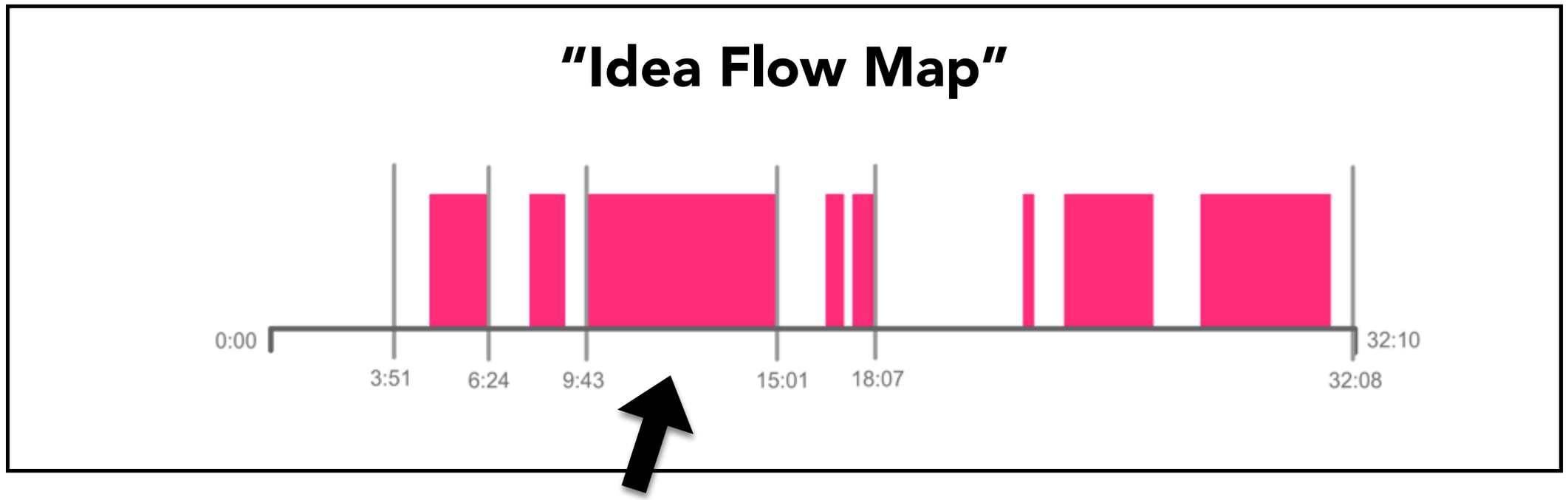
PAIN occurs during the ***process*** of understanding and extending the software



“Idea Flow”

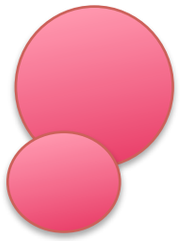
“Idea Flow” is the flow of ideas between the developer and the software

Visualize “Friction” in Developer Experience

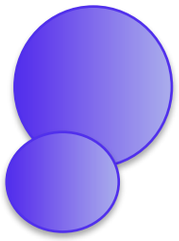


This is “Friction” in Idea Flow

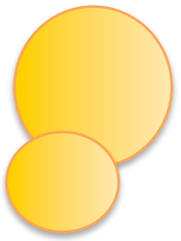
Three Types of Friction



Troubleshooting

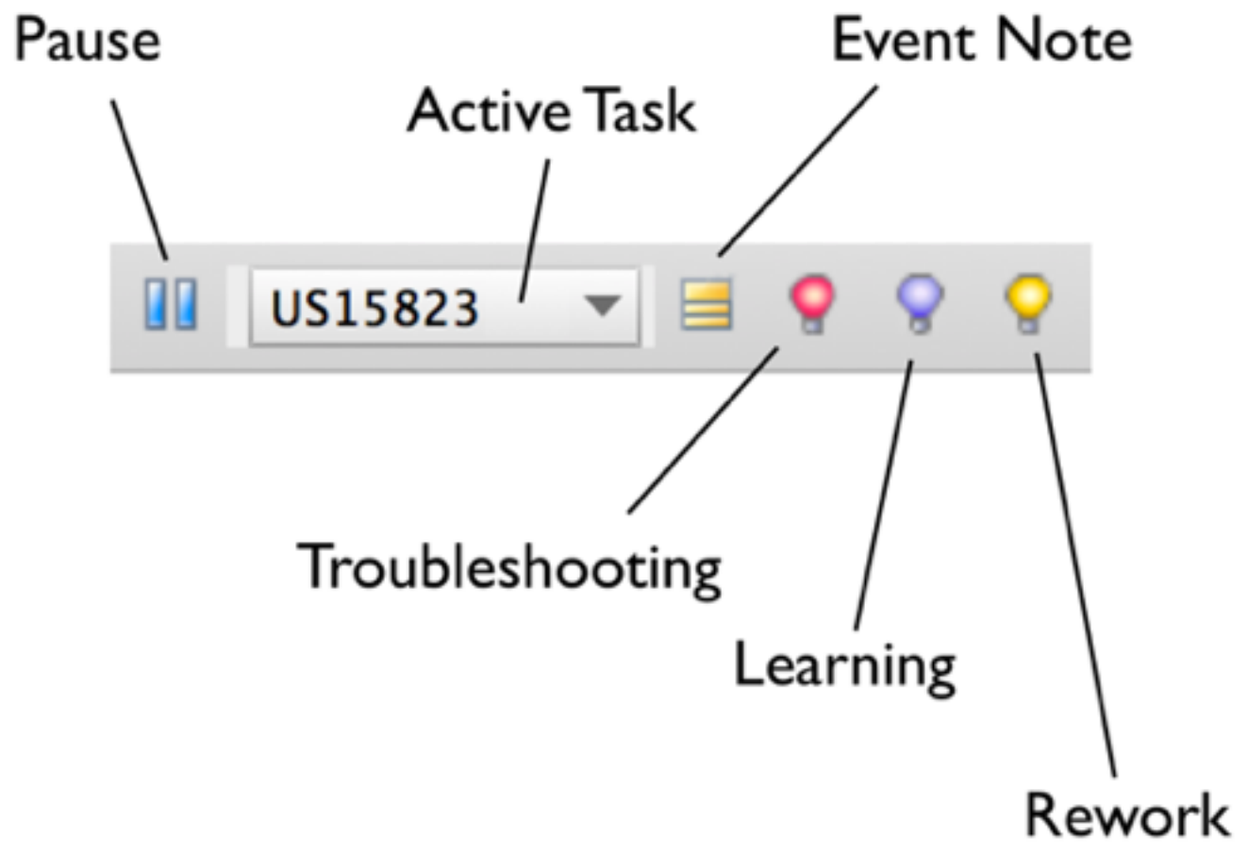


Learning



Rework

Idea Flow Mapping Tools

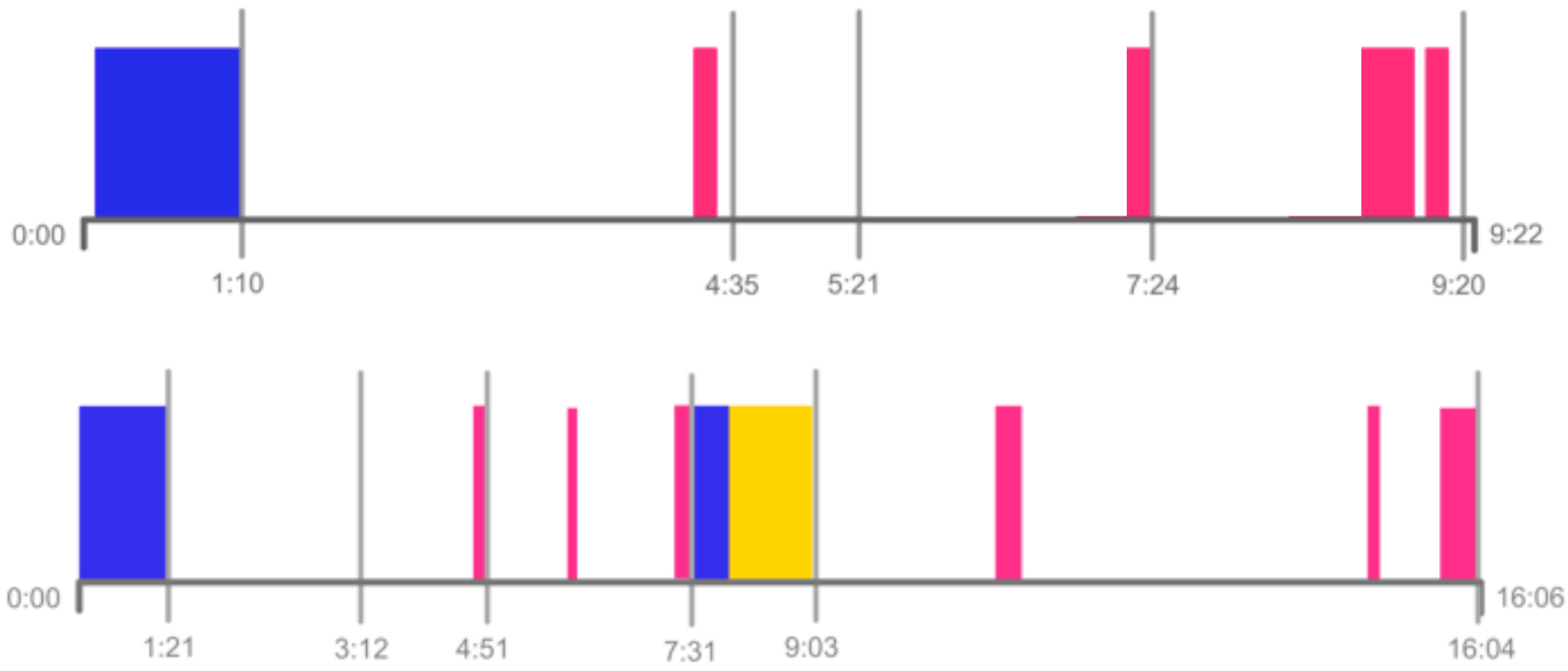


(Open Source, Supported GA ~April 2016)

github.com/ideaflow/tools

Case Study 1:

Healthy project (10 months) with a focus on reducing troubleshooting time.



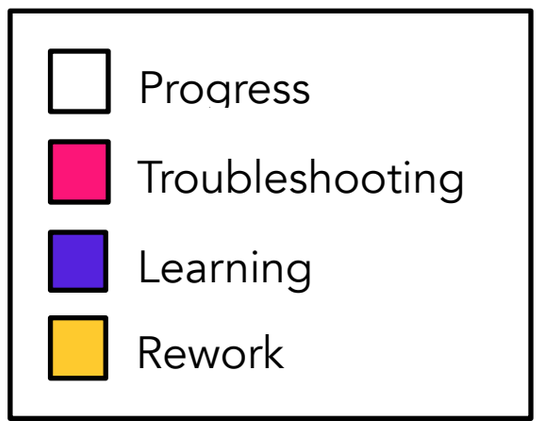
10-20% friction

Progress
Troubleshooting
Learning
Rework

Case Study 2:

18 months after a micro-services and Continuous Delivery rewrite

40-60% friction



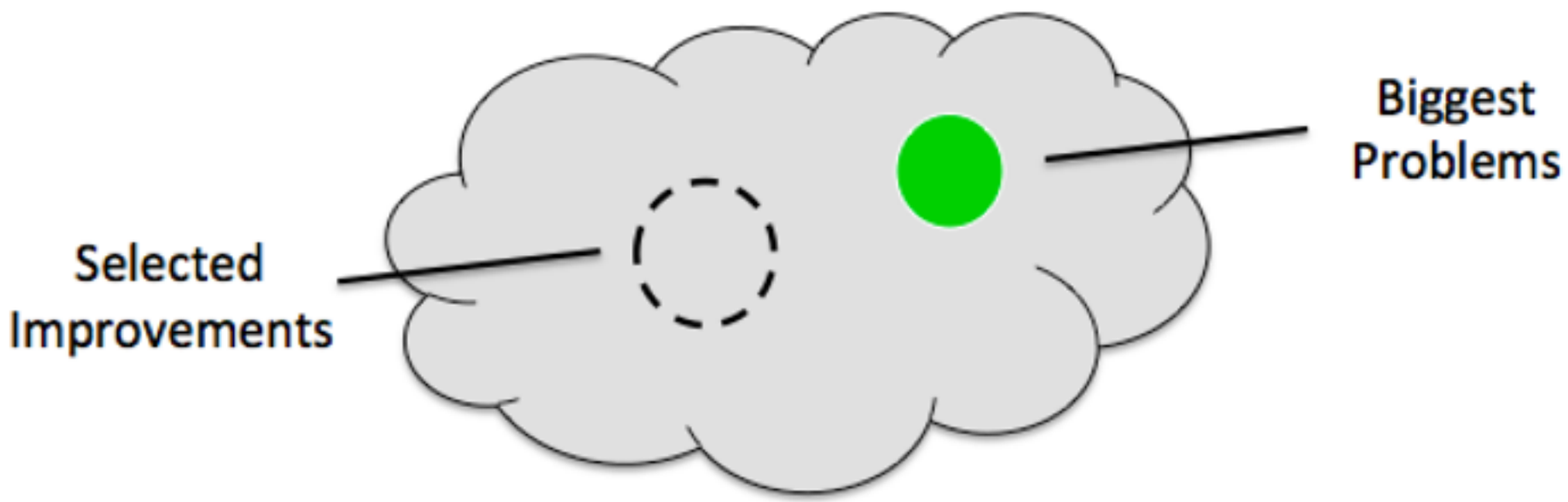
Case Study 2:

The Team's Focus: Reducing technical debt
in the micro-services code.

The Biggest Problem: Integration problems across teams
leading to environment down time.

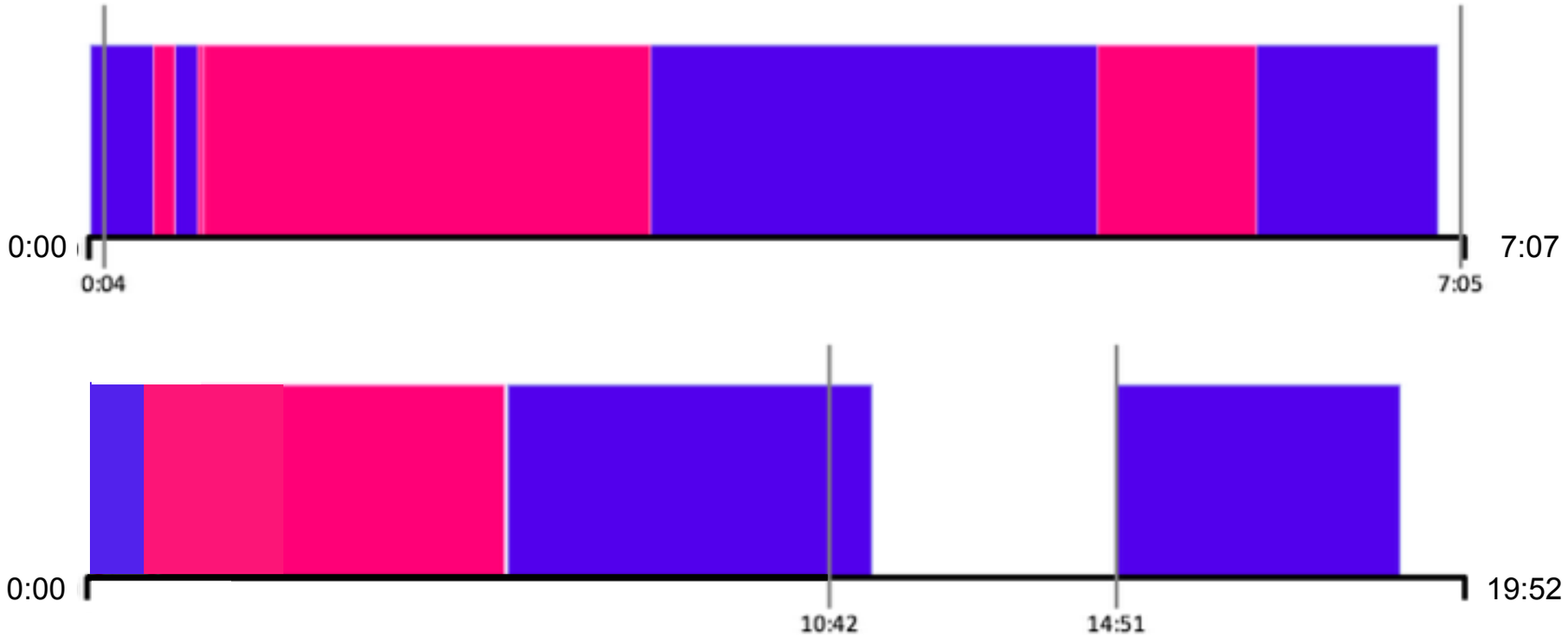
Cost:

1000 hours/month!!

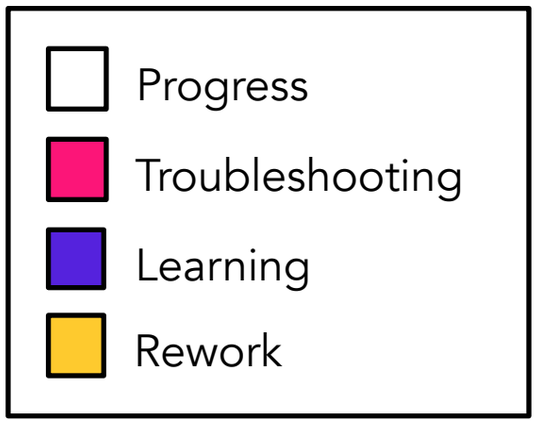


Case Study 3:

12 year old project inherited after all the original developers had left.



60-90% friction



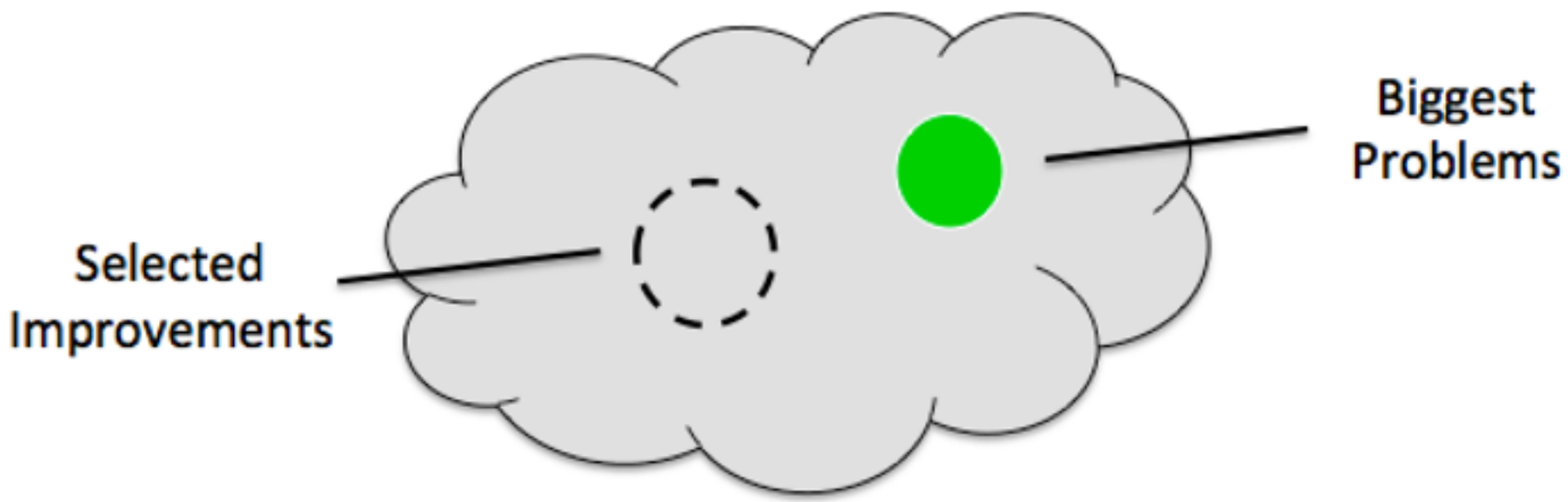
Case Study 3:

The Team's Focus: Increasing
Unit Test Coverage by 5%

The Biggest Problem: Generating test data
for running reporting engine experiments.

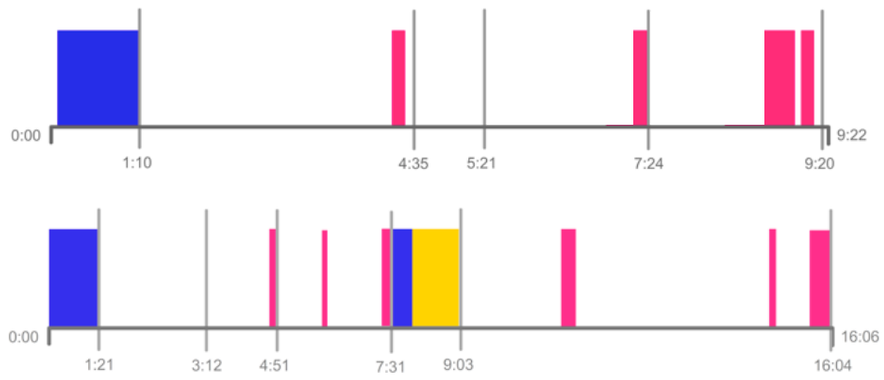
Cost:

700 hours/month!!



Why measure friction?

Case Study 3



1 day

2 days

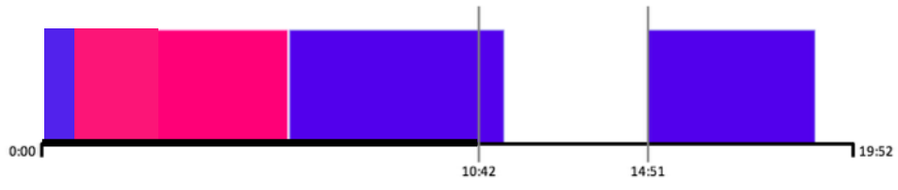
Case Study 1

1 day

3 days

Case Study 2

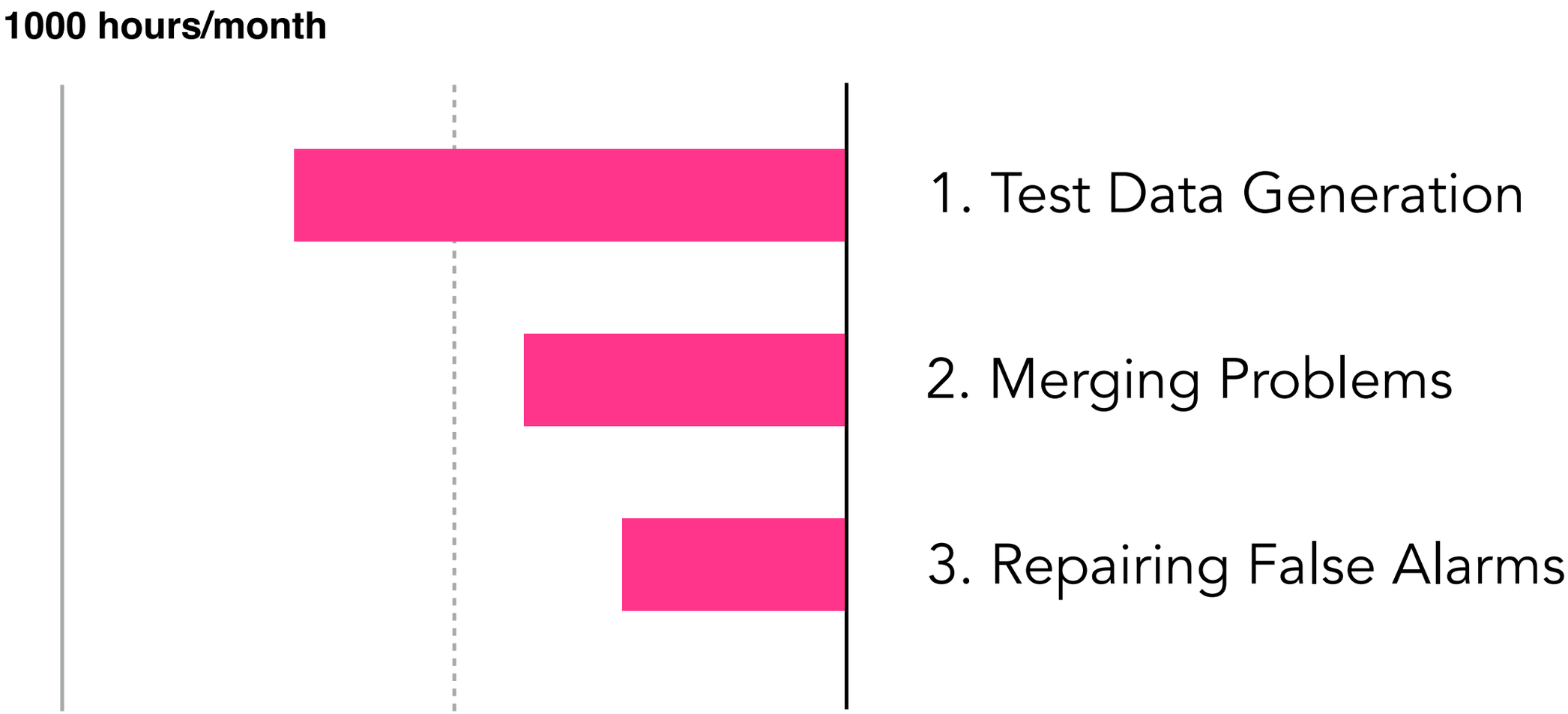
1 day



3 days

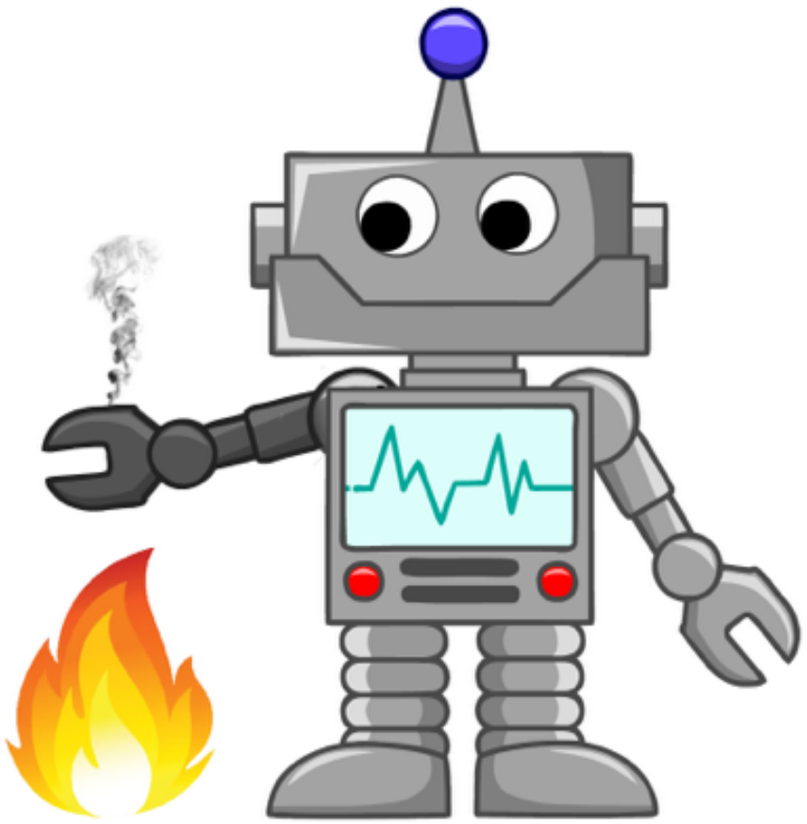
We **can’t see these effects** by measuring velocity or task lead-time.

Visibility lets us add up the distributed pain.



Broken Visibility

Lesson Learned: Install a Pain Sensor



"How do I know the *right improvements* are on the list?"

Top 5 Reasons Improvements Fail

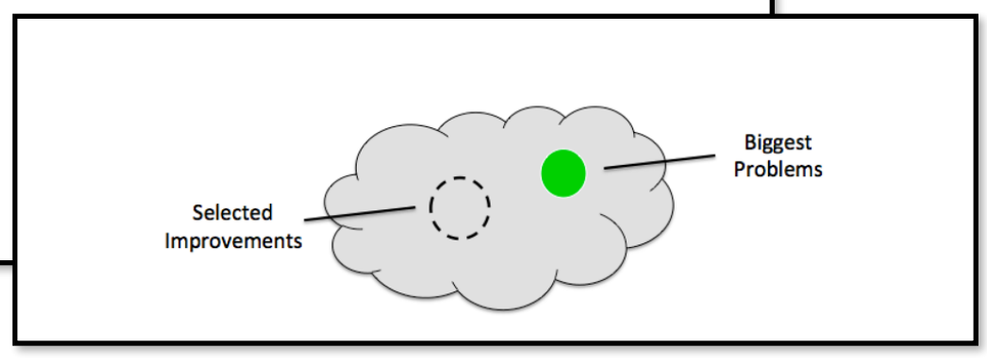
BROKEN TARGET

BROKEN VISIBILITY

3. We assume that we understand the problem.

4. We assume that understanding is enough.

5. We don't make the problems visible to management.



Top 5 Reasons Improvements Fail

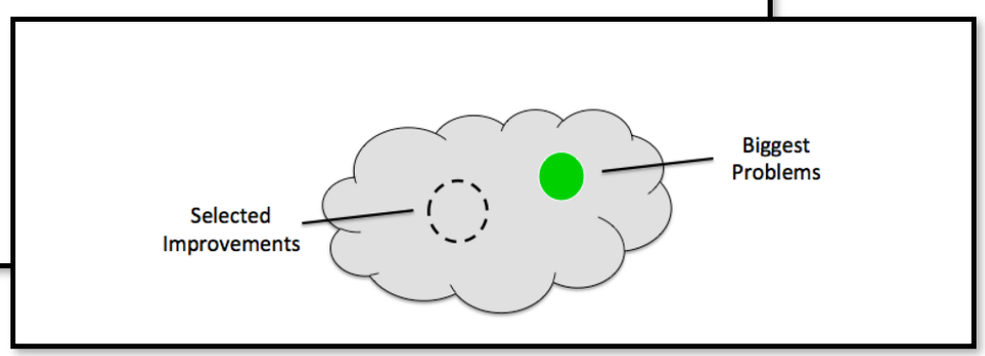
BROKEN TARGET

BROKEN VISIBILITY

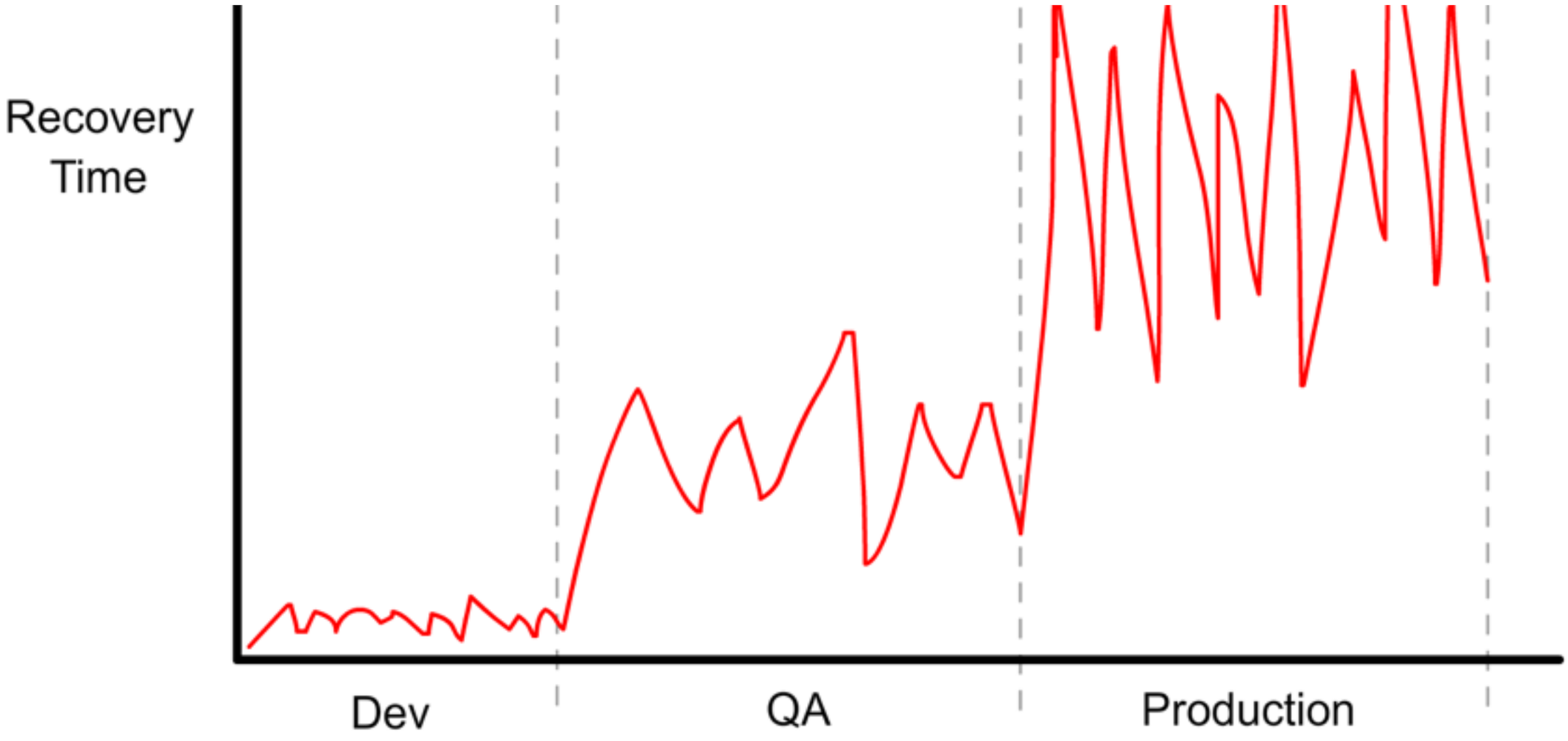
3. We assume that we understand the problem.

4. We assume that understanding is enough.

5. We don't make the problems visible to management.

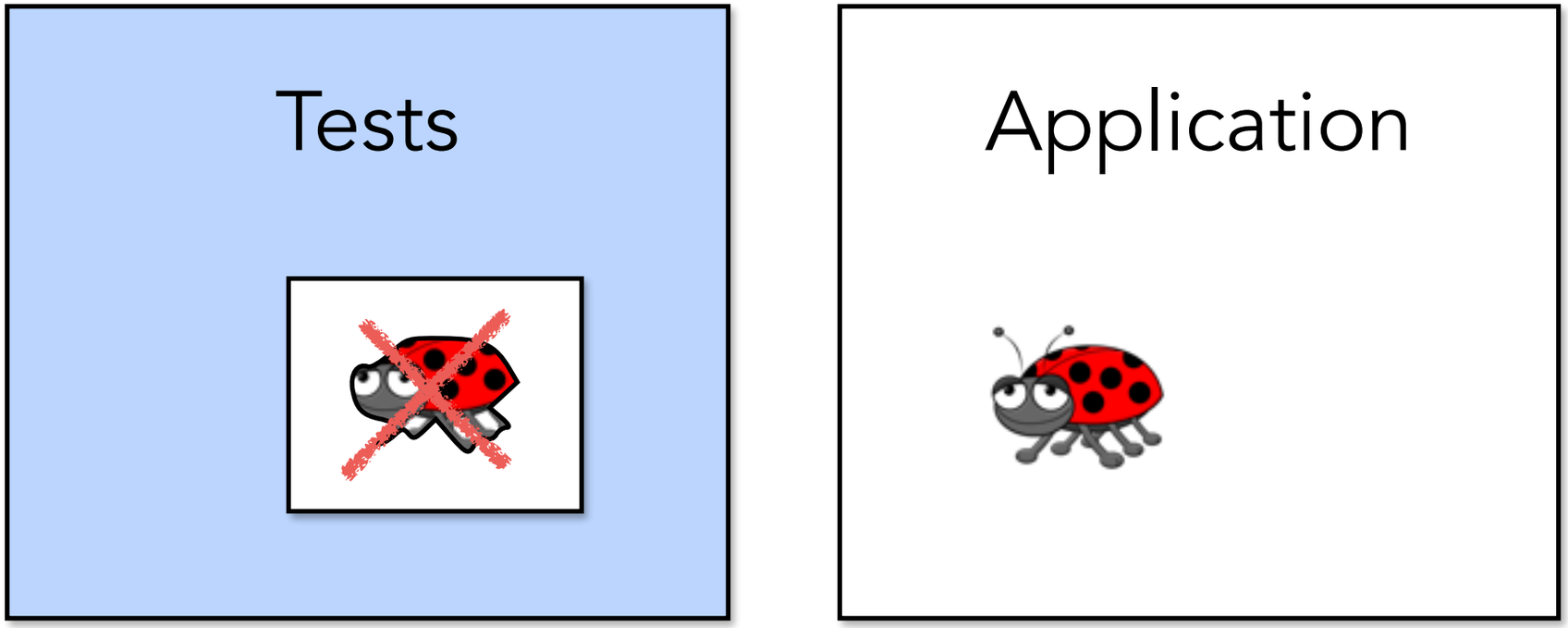


The Deadline Effect

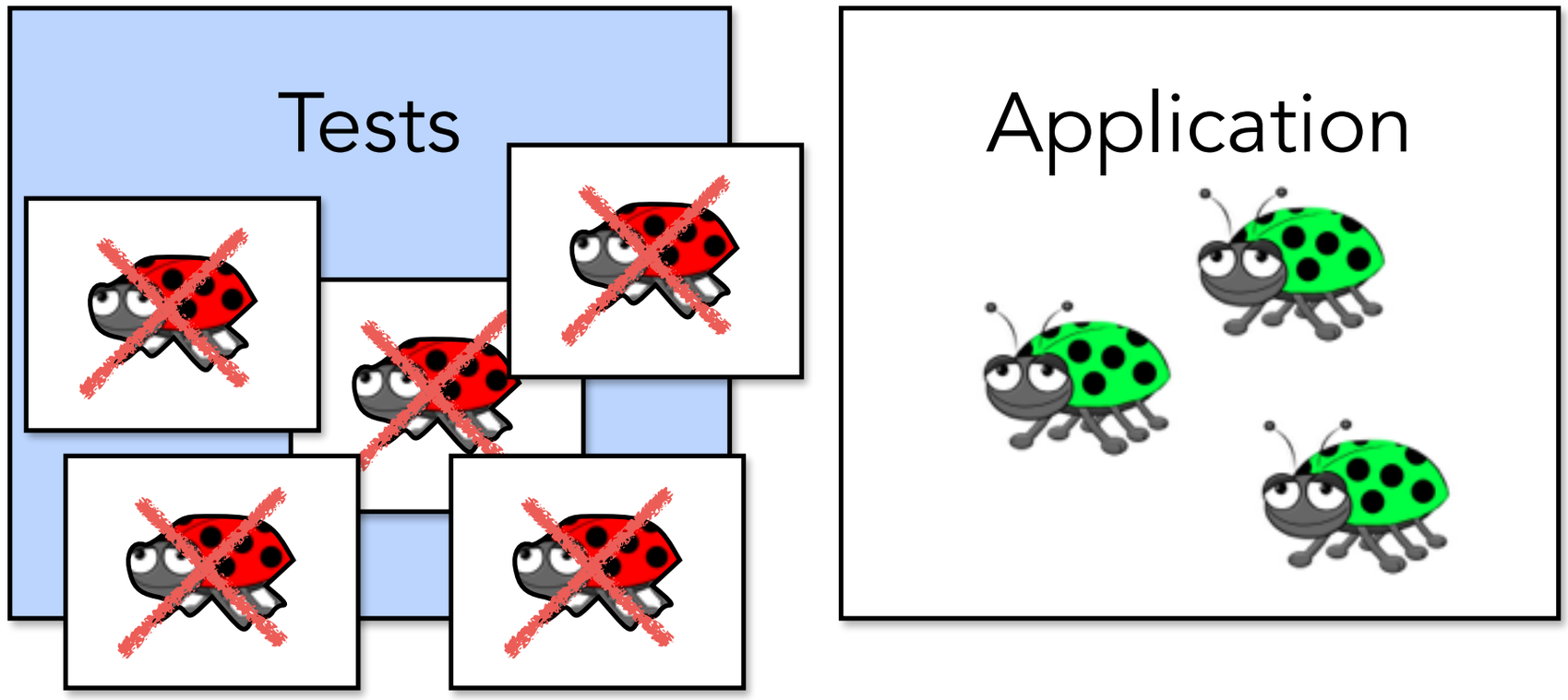


Conclusion: Developers need to detect mistakes earlier.

Our Test Suite

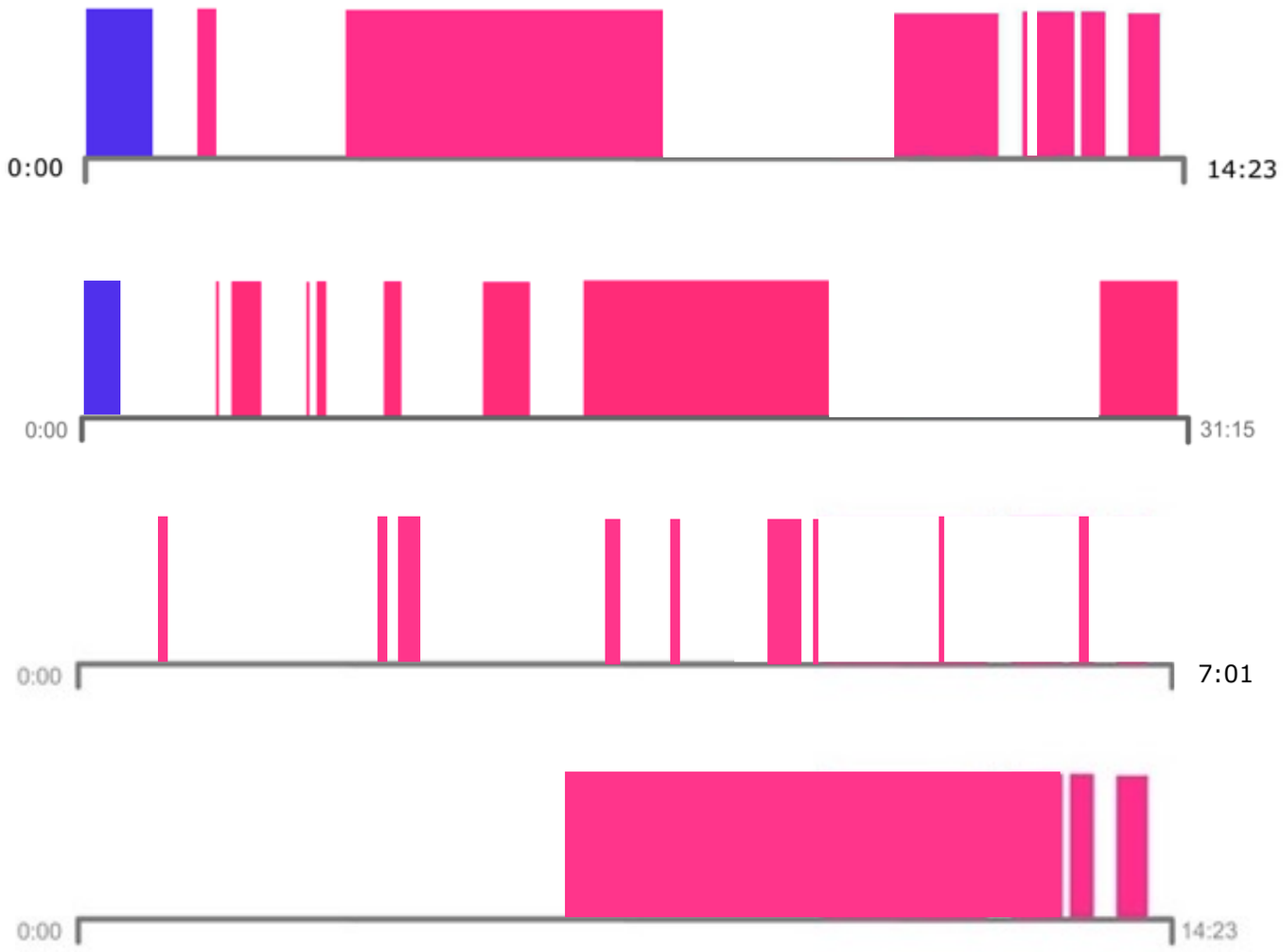


Our Test Suite



“What are we supposed to do?!”

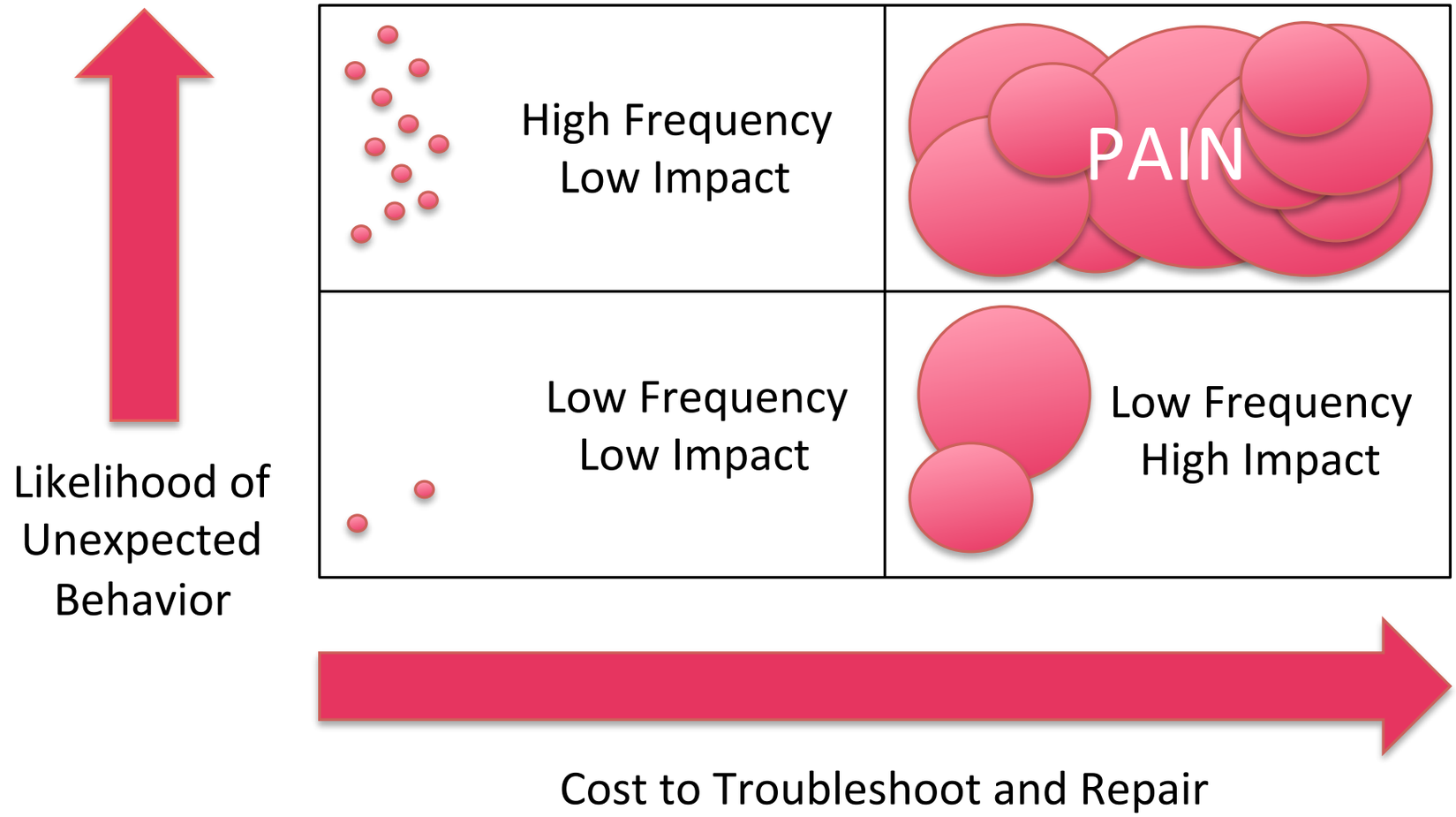
Stop Making Generalizations.



What caused each *specific* problem?

What are the *patterns* of cause and effect?

Breakdown the Dimensions of Risk…



Breakdown the Problem Patterns

What Causes Unexpected Behavior (*likeliness*)?

Familiarity Mistakes

Stale Memory Mistakes

Semantic Mistakes

Bad Input Assumption

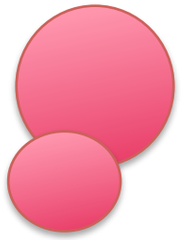
Tedious Change Mistakes

Copy-Edit Mistakes

Transposition Mistakes

Failed Refactor Mistakes

False Alarm



What Makes Troubleshooting Time-Consuming (*impact*)?

Non-Deterministic Behavior

Ambiguous Clues

Lots of Code Changes

Noisy Output

Cryptic Output

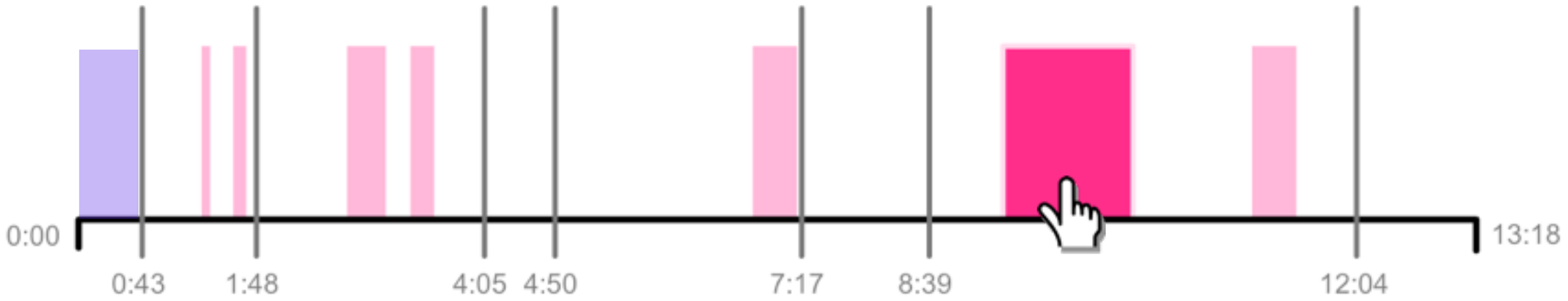
Long Execution Time

Environment Cleanup

Test Data Creation

Using Debugger

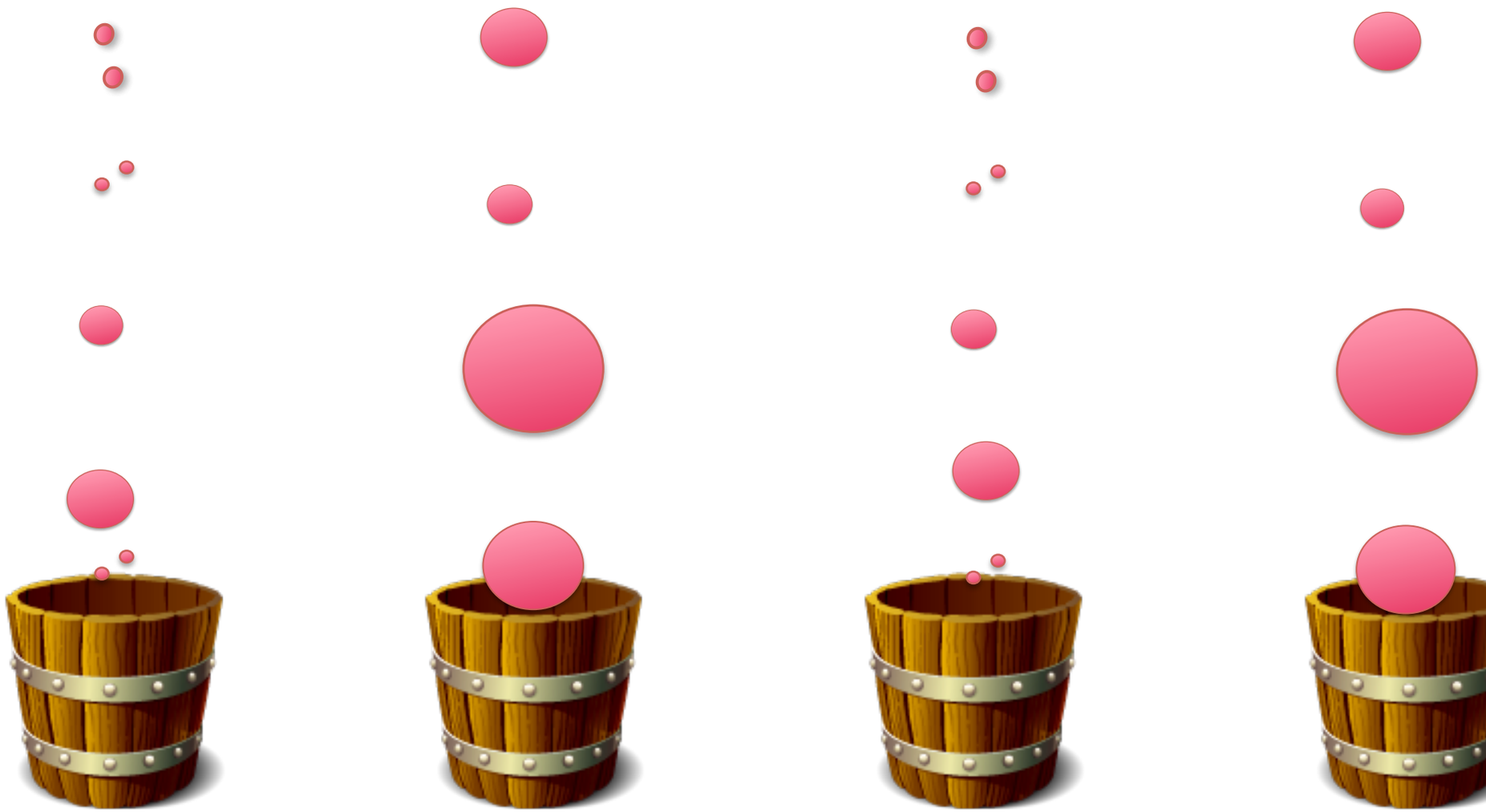
Tag the Examples by Problem Type



Conflict: "Why isn't the token showing up in the URL?"
Resolution: "Typo in the email template."
Duration: 1h 08m 05s

Tags: #problemtag

Sort the Examples into Buckets



Experiment Pain

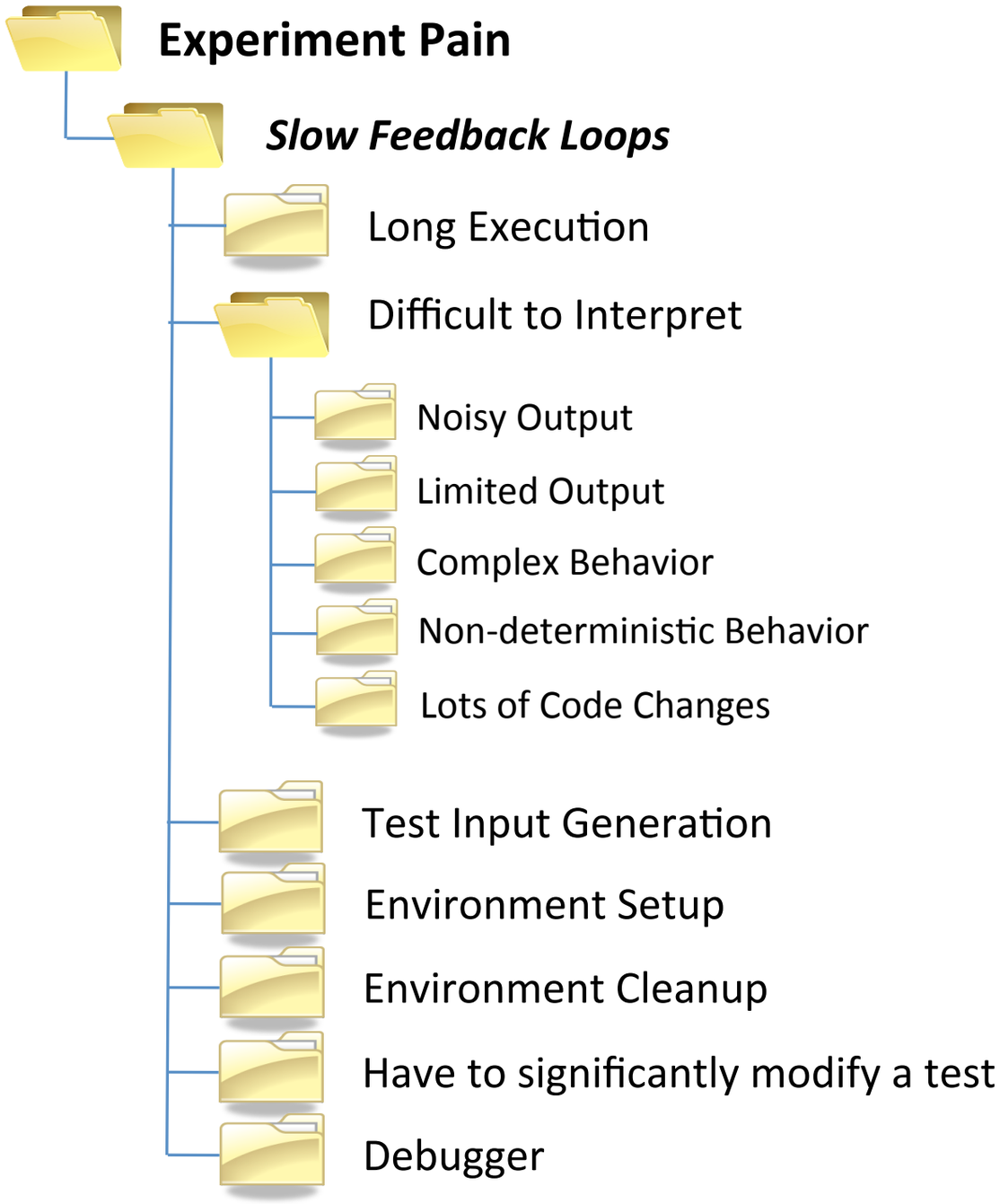
Alarm Pain

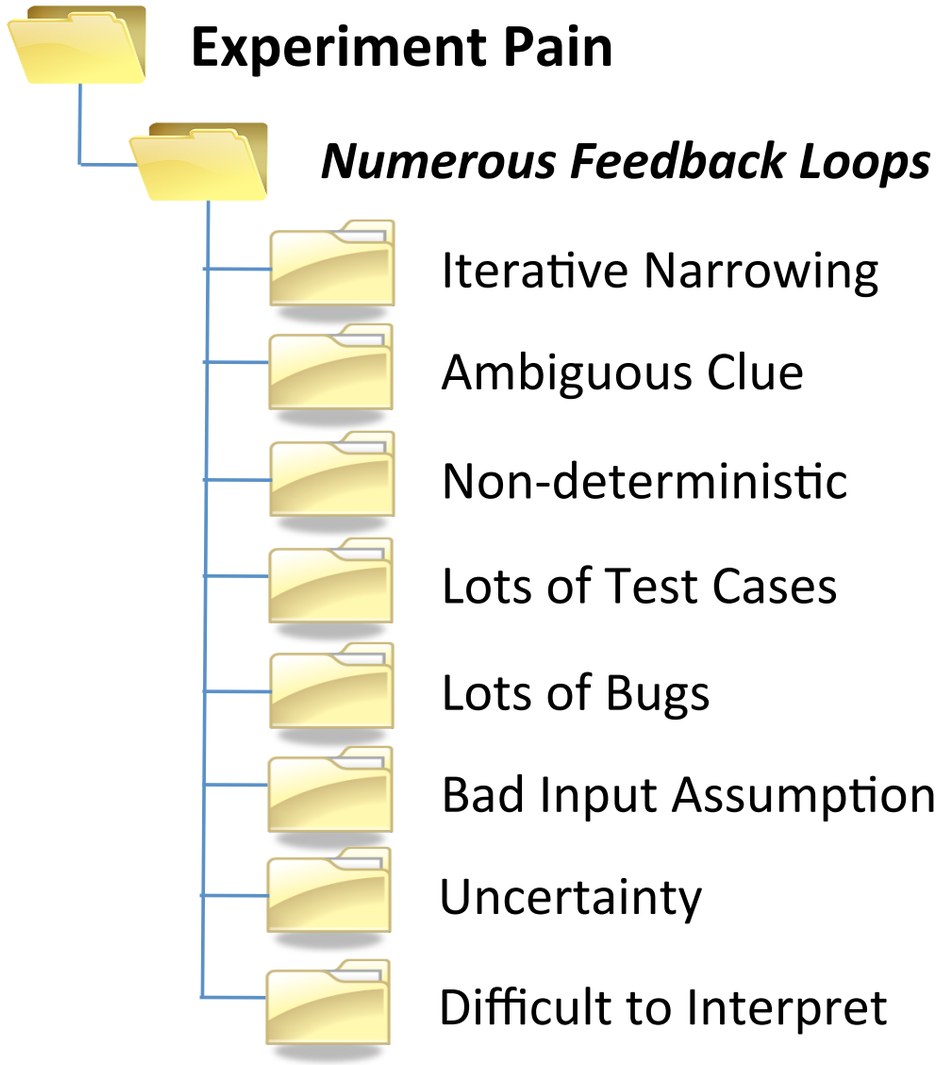
Collaboration Pain

Cognitive Pain

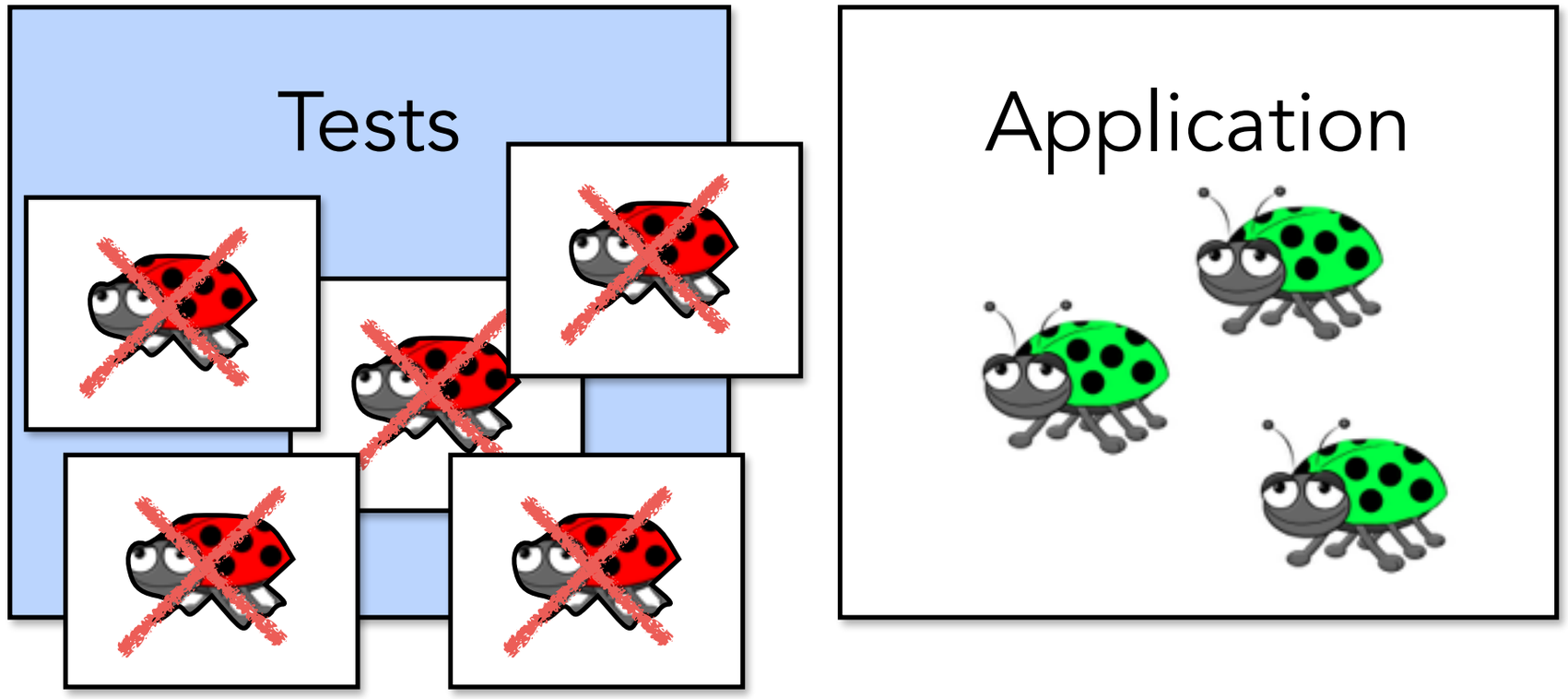
Break It Down, Down, Down

Drill Down ⟶ Find Patterns ⟶ More Buckets



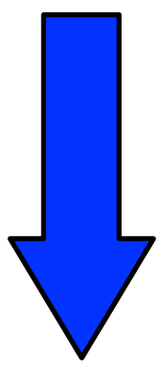


Guess what we found?



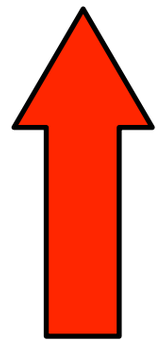
Remember this?

We made significantly more mistakes
in **code that we didn't write** ourselves.



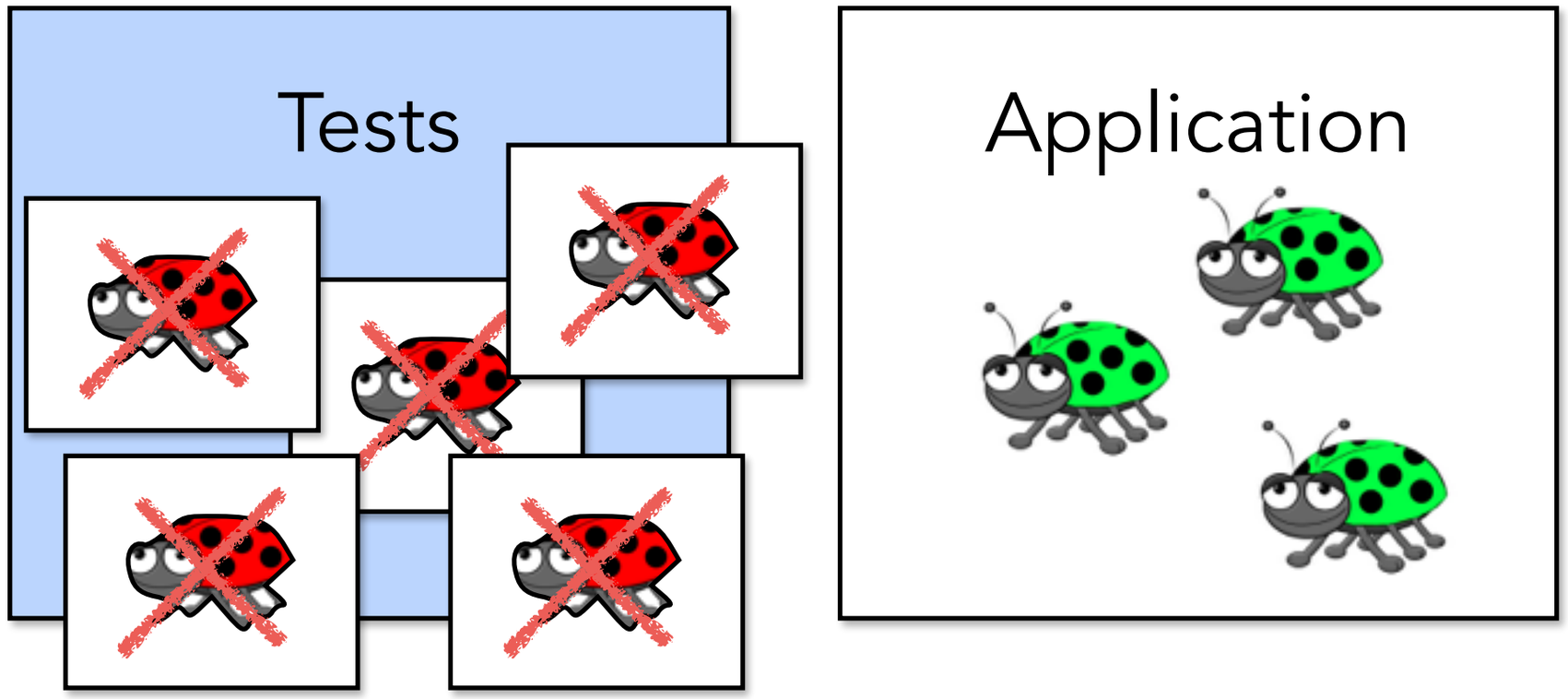
Lower Familiarity

=



More Mistakes

The bugs were in our *conceptual models,* not the code.

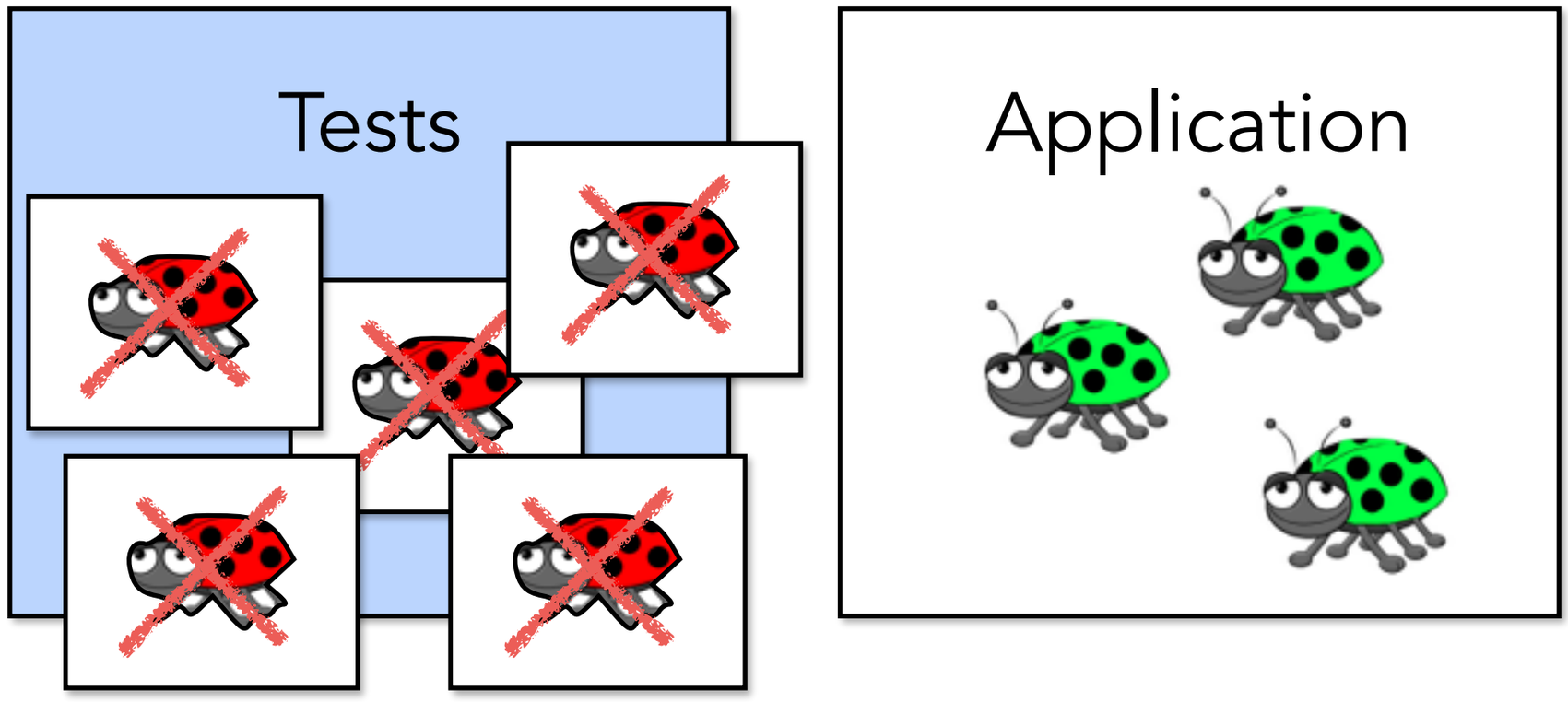


We built a

Data Dictionary

Broken Clarity

Lesson Learned: Stop Making Generalizations



"What are the specific patterns of cause and effect?"

Top 5 Reasons Improvements Fail

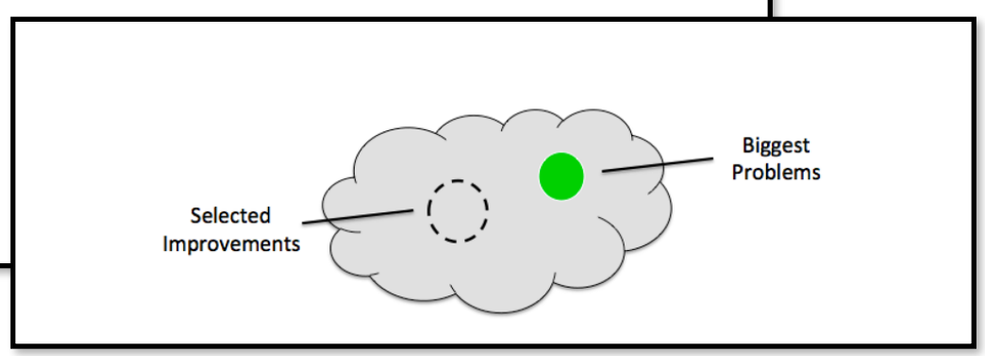
BROKEN TARGET

BROKEN VISIBILITY

BROKEN CLARITY

4. We assume that understanding is enough.

5. We don’t make the problems visible to management.



Top 5 Reasons Improvements Fail

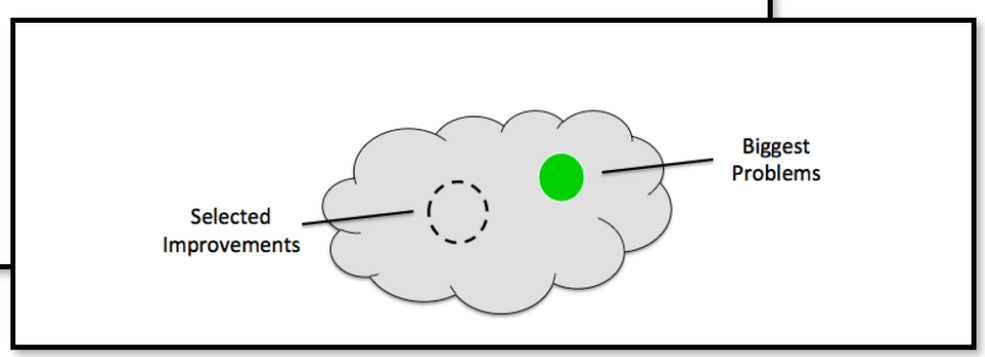
BROKEN TARGET

BROKEN VISIBILITY

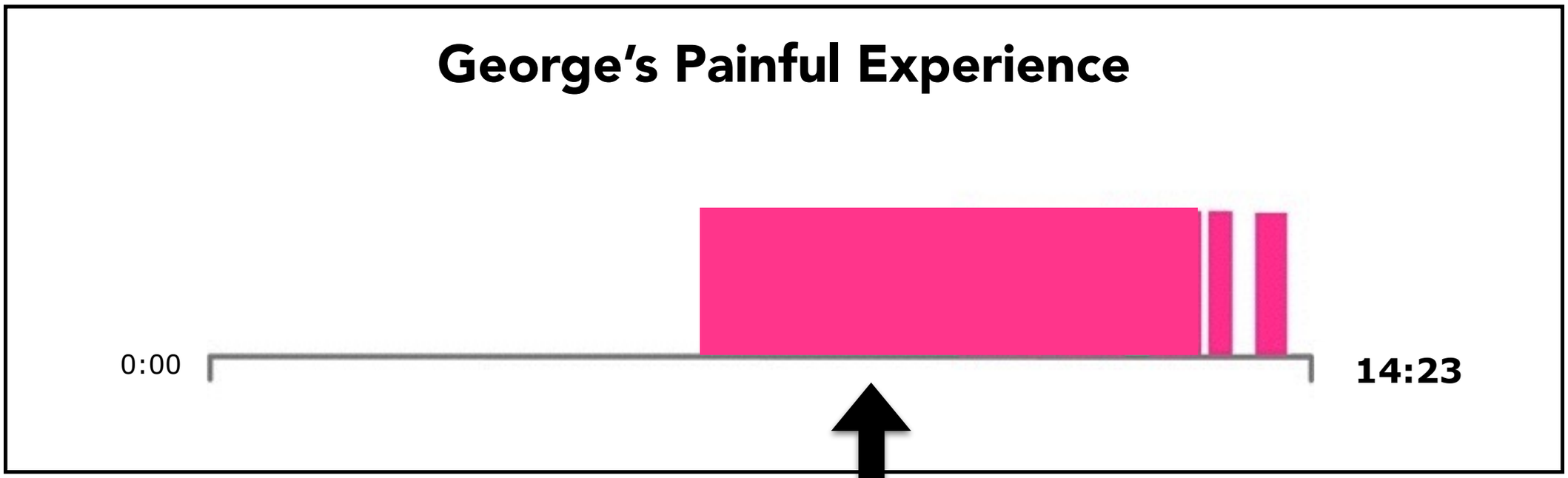
BROKEN CLARITY

4. We assume that understanding is enough.

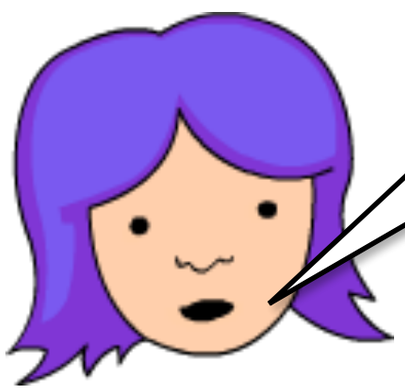
5. We don't make the problems visible to management.

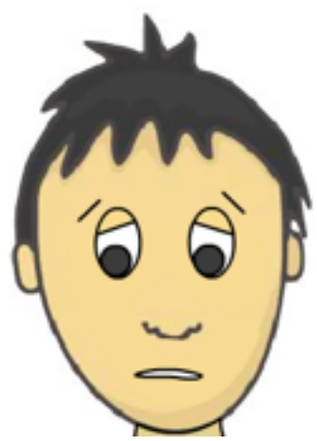


The *Experience* Review



What made **troubleshooting** take so long?



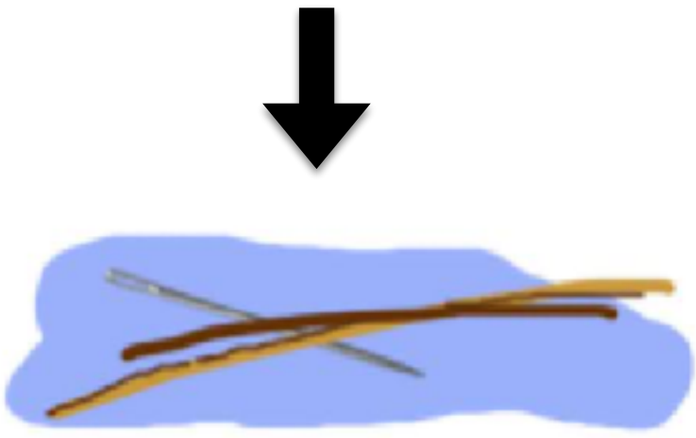


Teach the Missing Concept

The Haystack Principle

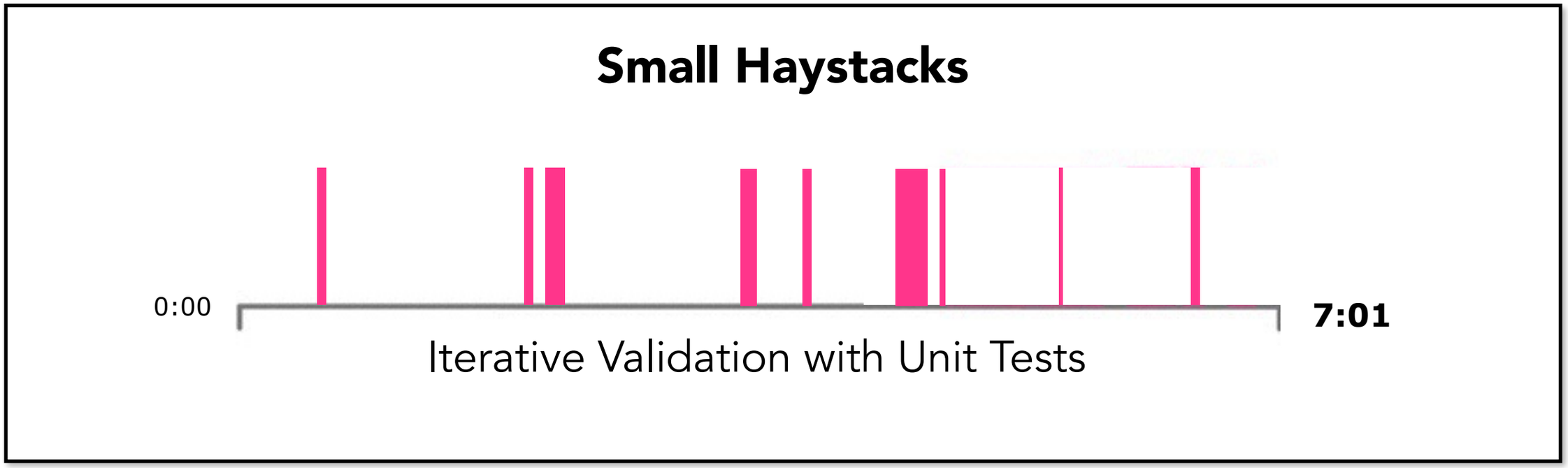
Lots of unvalidated changes

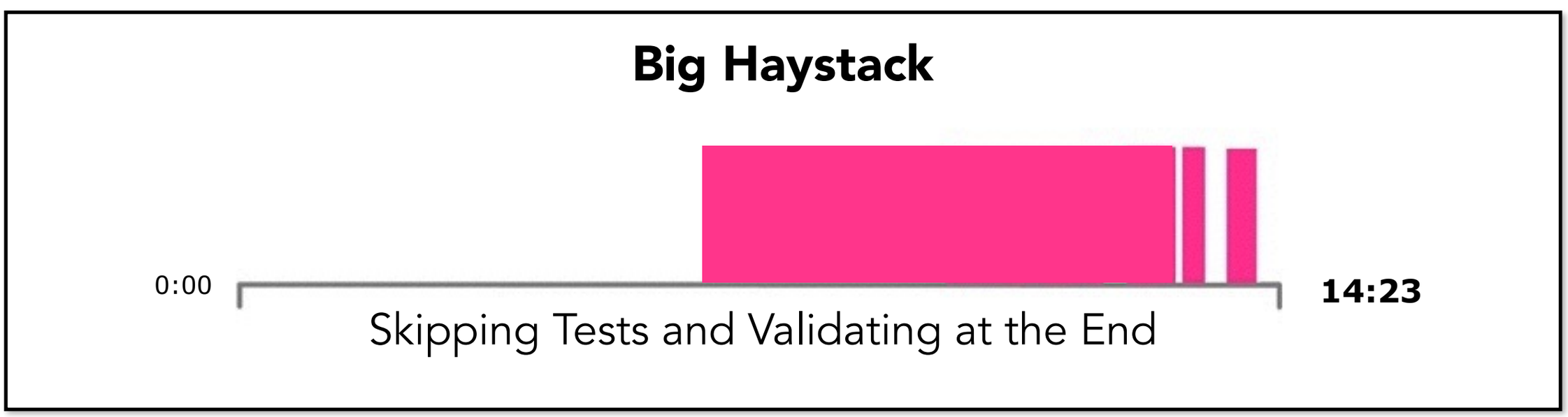
Easier to find the needle.



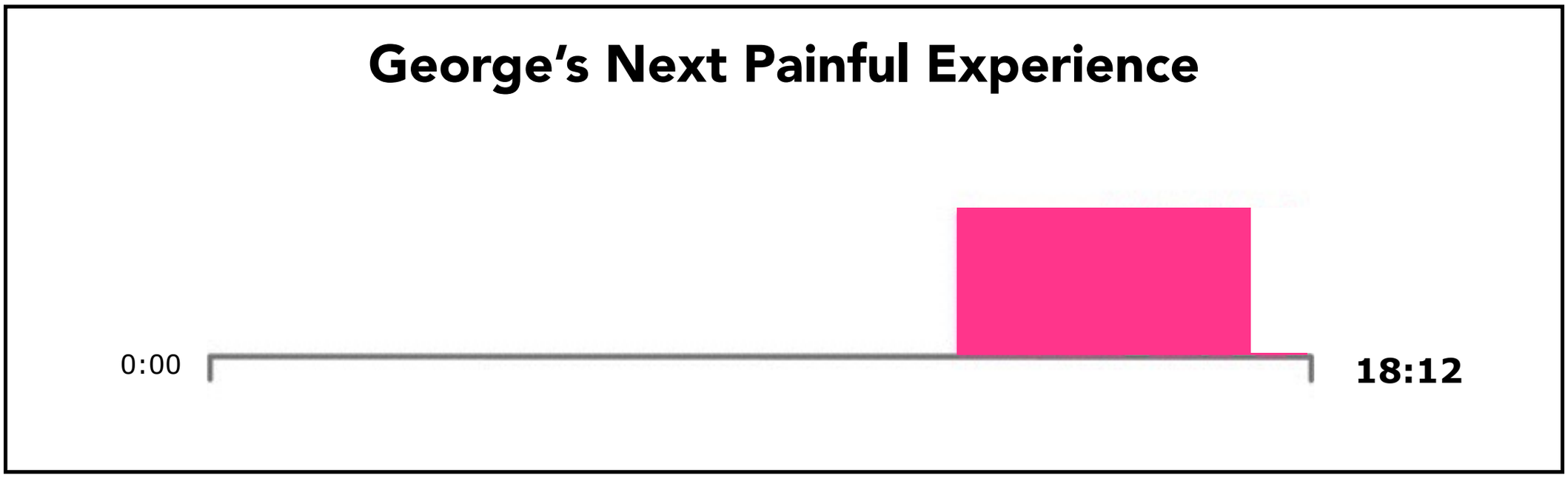
Optimize for small manageable haystacks.

The Haystack Principle

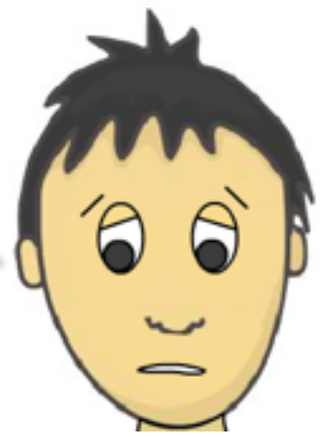




The Developer kept *repeating* the same mistakes.



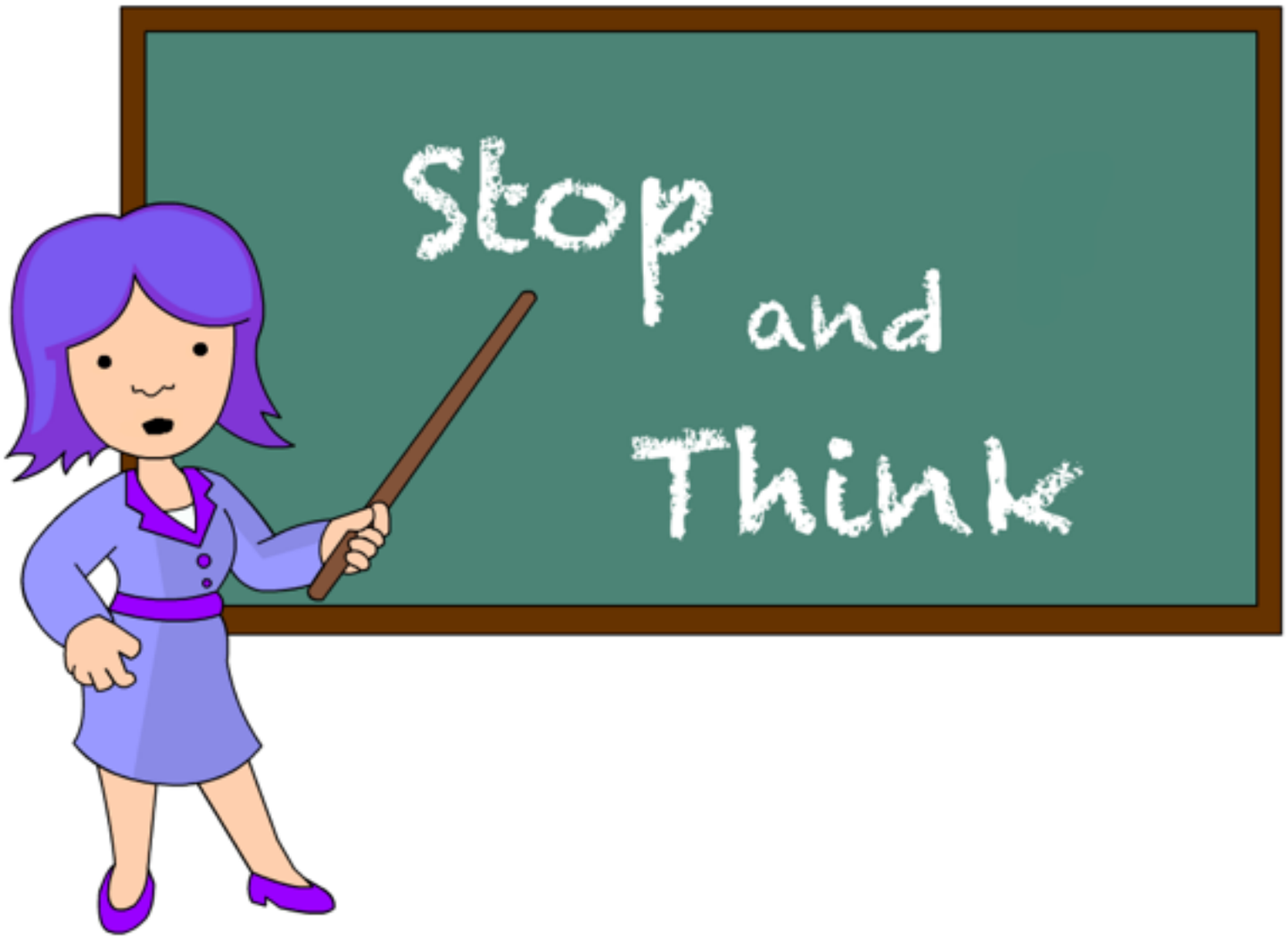
I know I made a big haystack,
I'll do better next time.



This is hindsight bias, it's not that easy!

The Challenge

We tend to make ***auto-pilot decisions*** and do what's familiar

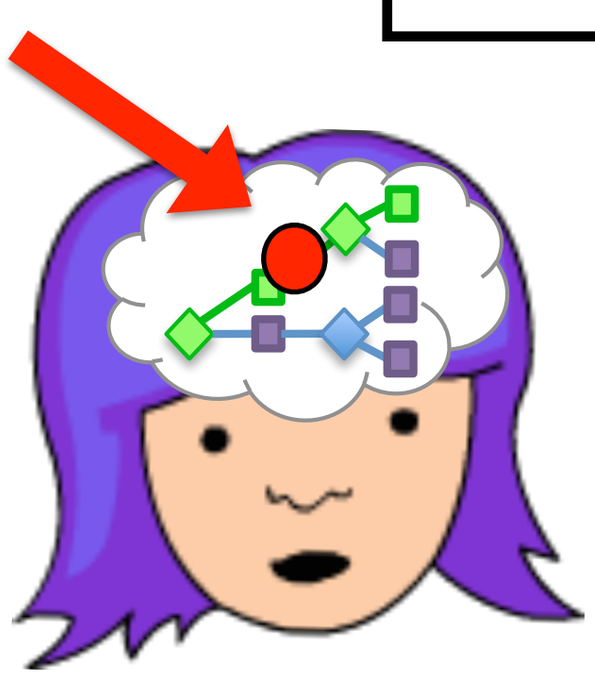


How do we break *decision habits*?

Changing Decision Habits

Stop and Think!

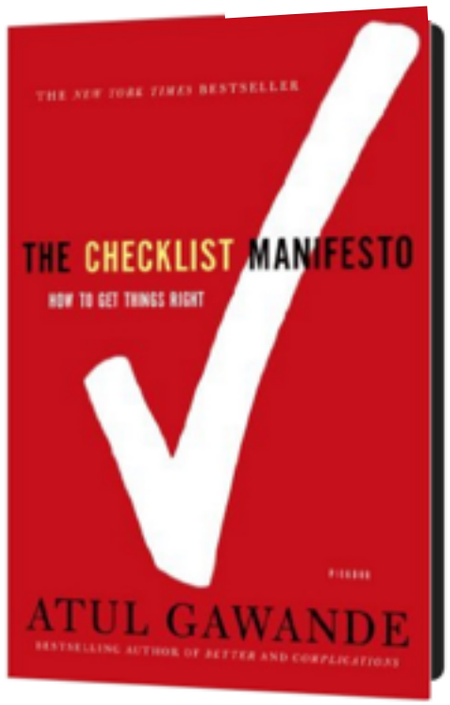
Breakpoint



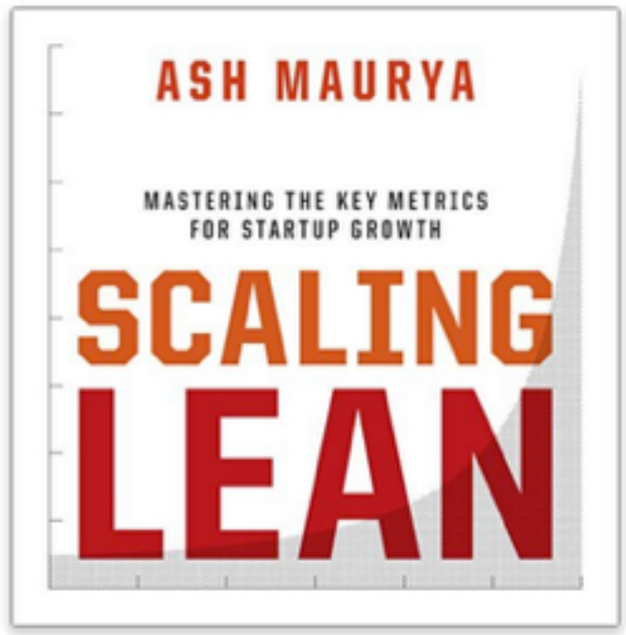
```
Decision decideSomething( input1, input2 ) {
    //logical evaluation of inputs
    return decision
}
```

We have to **predict** the big haystack during the decision.

Change Decision Habits with *Explicit Predictions*



+



The Checklist Manifesto

Atul Gawande

Scaling Lean (Lean Startup)

Ash Maurya

Let's Make a Checklist!

I want to avoid this...

"What **question** could I ask my future self to recognize similar risks in the future?"

"In what **situation** would I ask the question?"

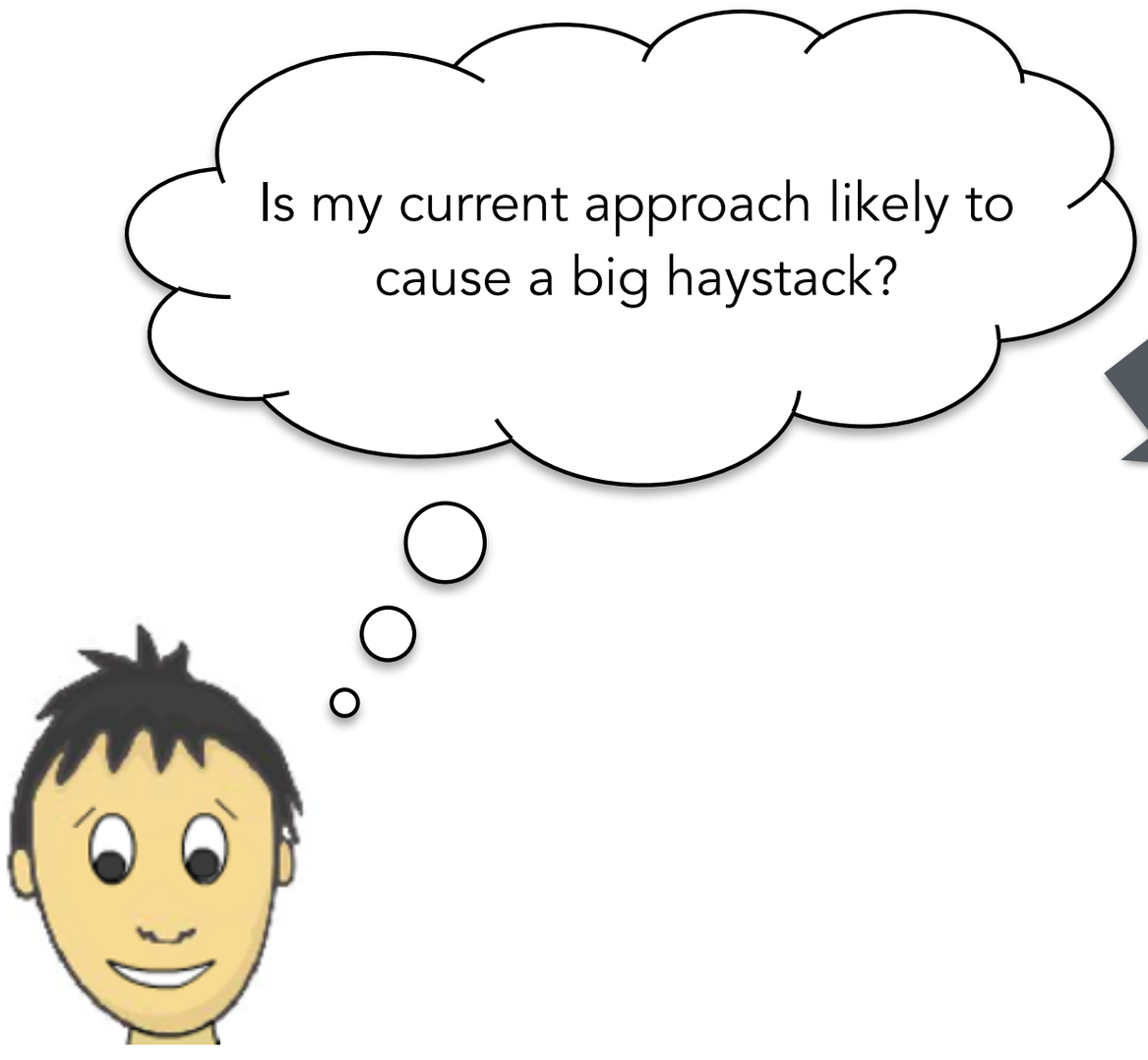
Thinking Checklist

Is my current approach likely to cause a big haystack?

Situation: start of subtask

Strategy Experiments

Stop and Think:

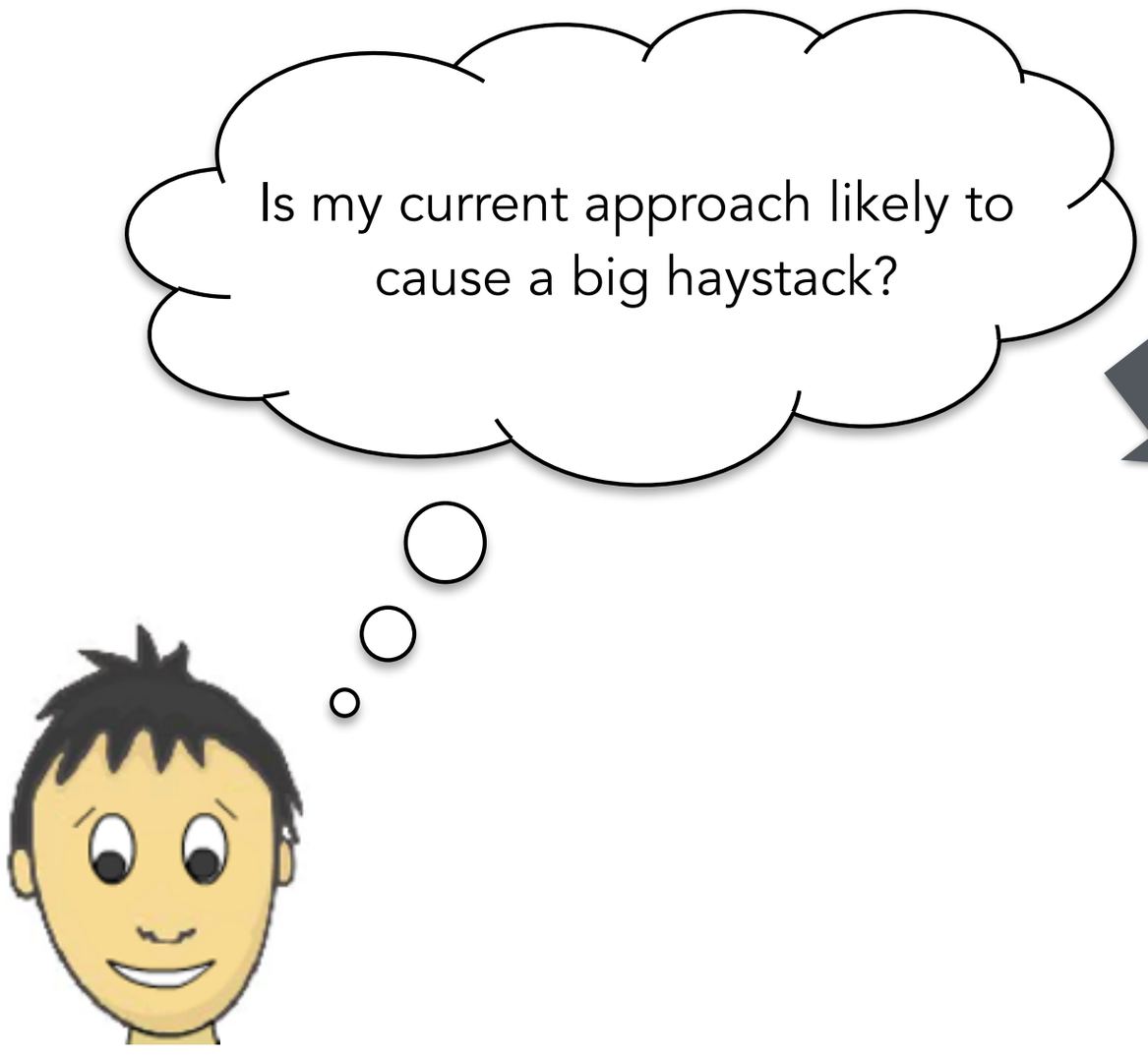


Predict: *Small* haystack

Strategy Experiments

Stop and Think:

False Prediction

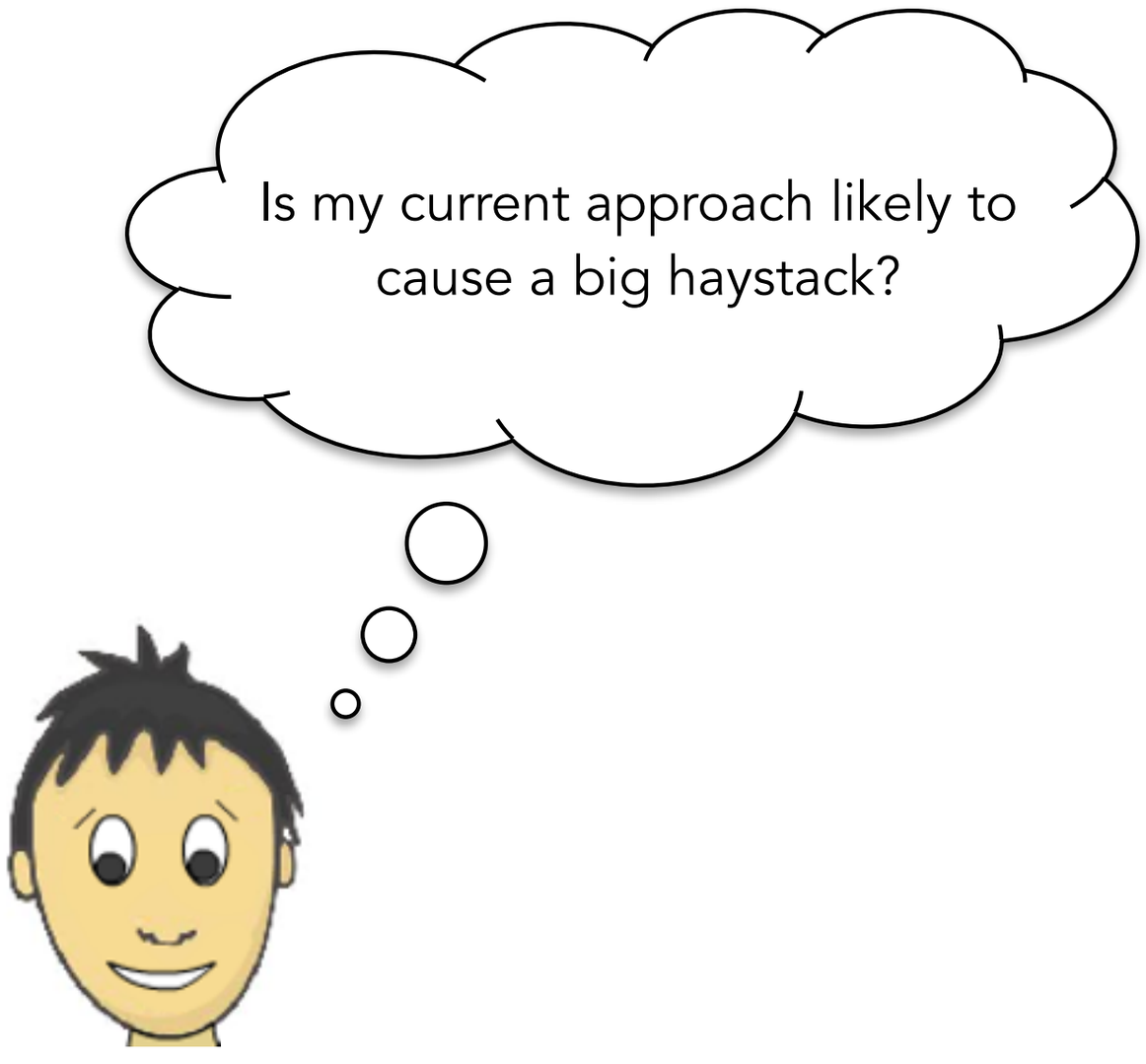


Predict: *Small* haystack

Strategy Experiments

Stop and Think:

False Prediction



Start of Subtask

Q: Is my current approach likely to cause a big haystack?

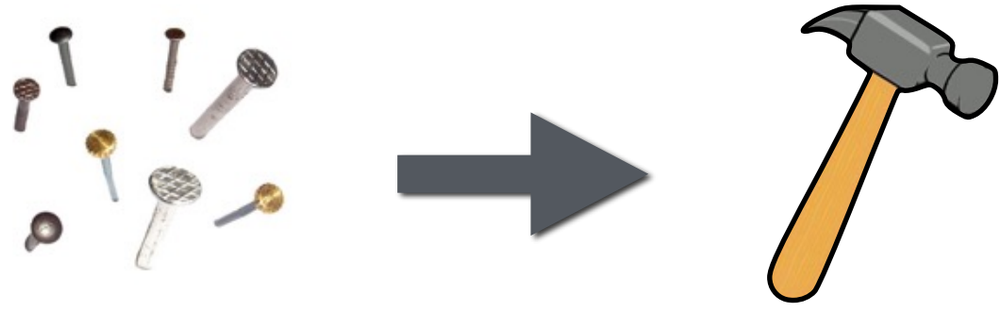
High-Risk Situations

1. Unraveling sweater
2. Integration-heavy change
3. High state variation
4. Minimum scope is big

Haystack Decisions

Situation Types ("see")	Strategy Types ("do")
1. UnravelingSweater	1. DependencyAnalysis
2. HeavyIntegrationLogic	2. IncrementalIntegrationTest
3. HighStateVariation	3. DataDrivenTest
4. CoupledExternalDependencies	4. IsolateHardToTestCode

See **Do**

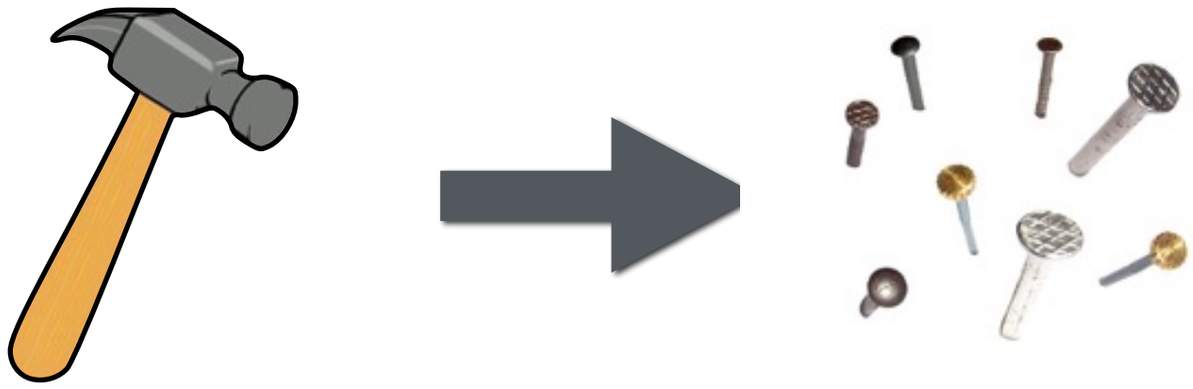


Codify What Works

Best Practices are Backwards

```
def bestPractices = new HashMap<SolutionPattern, ProblemPattern>()
```

Do



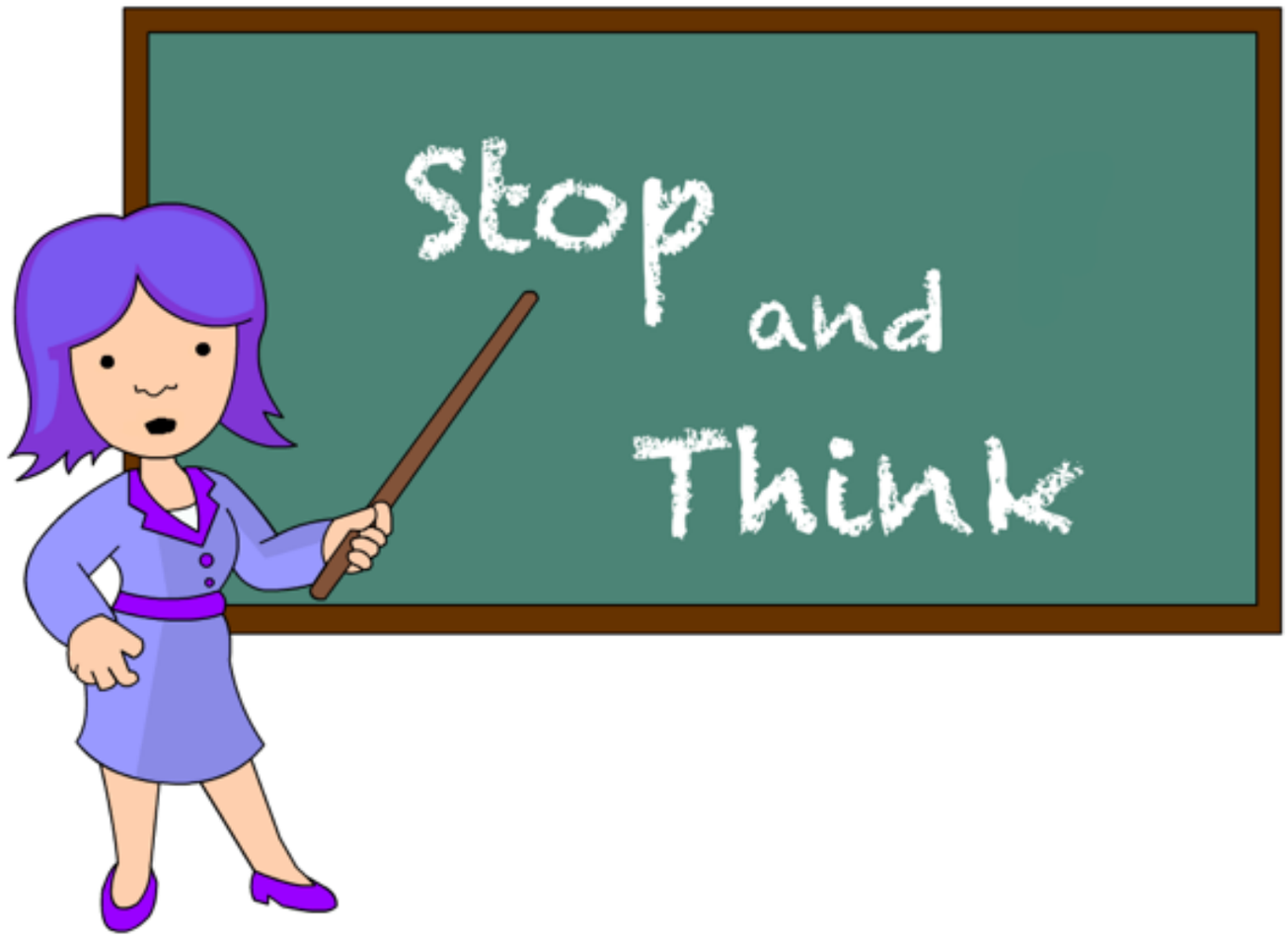
See

We need a new data structure!!

We don’t encounter solutions**, we encounter problems.**

Broken Awareness

Lesson Learned: Stop and Think



"How does my decision-making need to change?"

Top 5 Reasons Improvements Fail

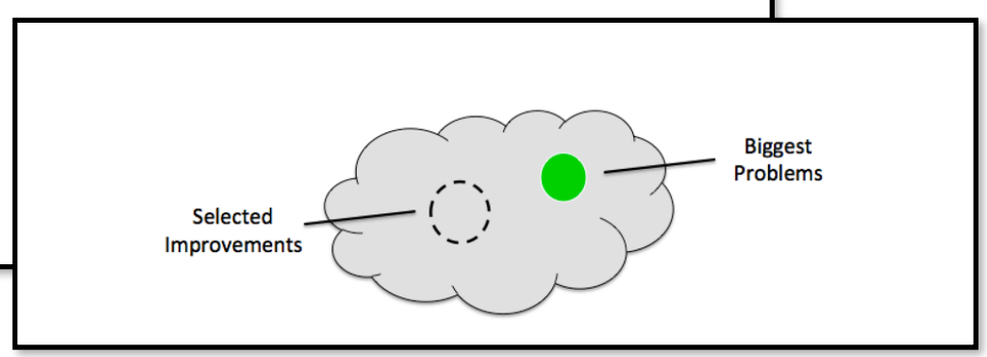
BROKEN TARGET

BROKEN VISIBILITY

BROKEN CLARITY

BROKEN AWARENESS

5. We don’t make the problems visible to management.



Top 5 Reasons Improvements Fail

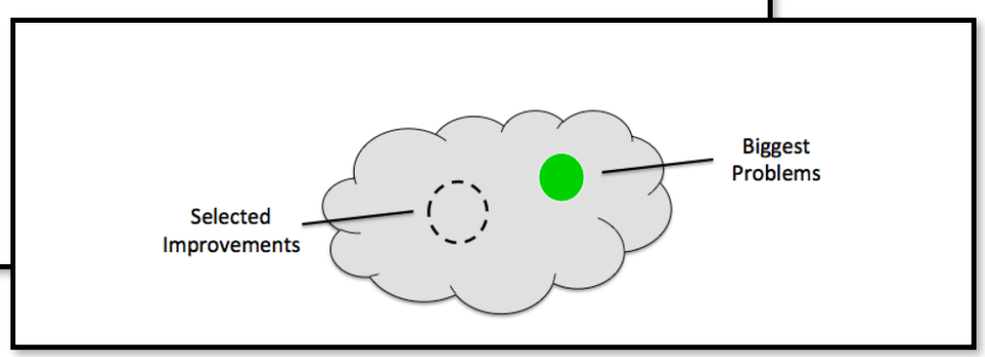
BROKEN TARGET

BROKEN VISIBILITY

BROKEN CLARITY

BROKEN AWARENESS

5. We don’t make the problems visible to management.




I WANT IT NOW!

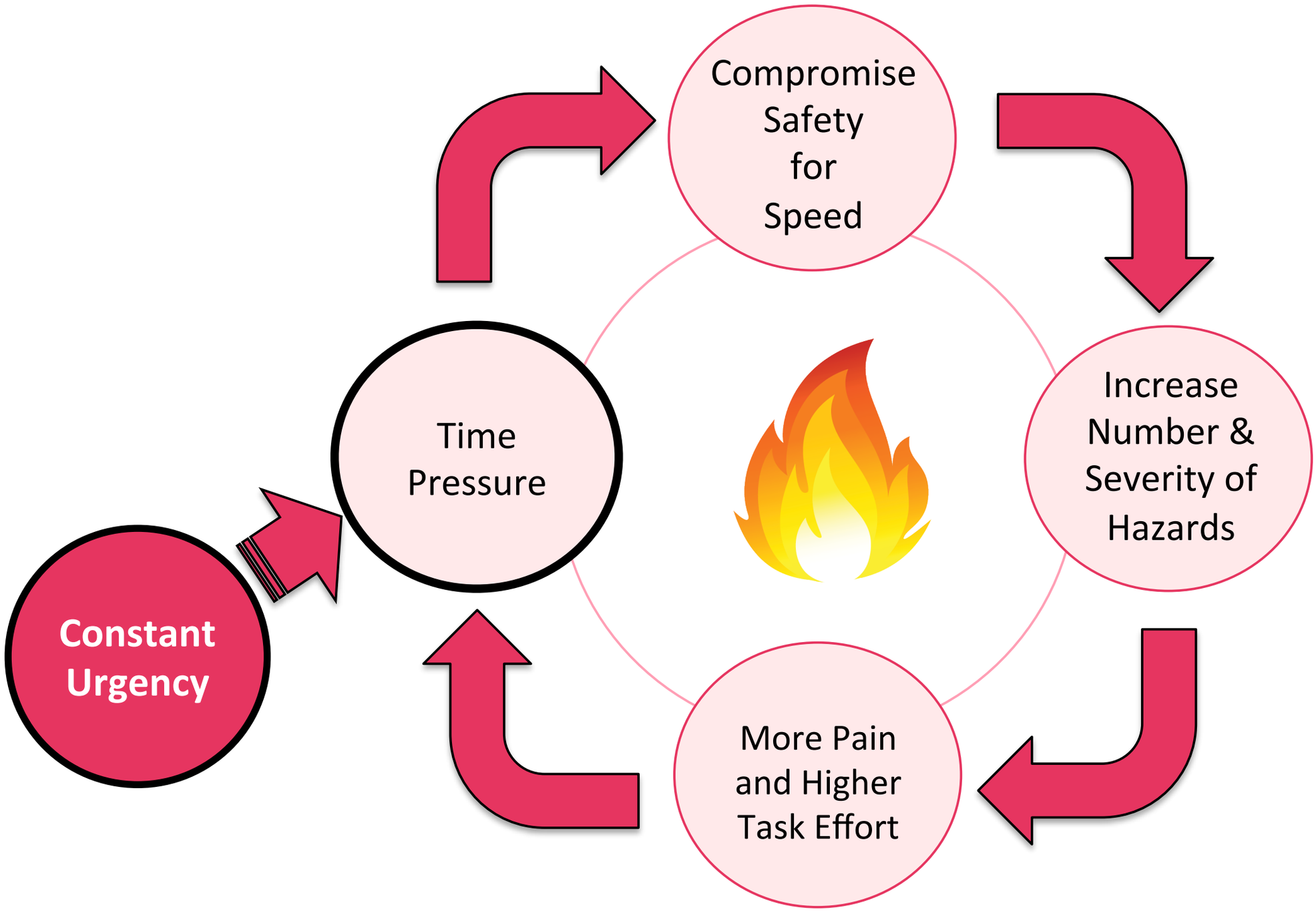
Constant
Urgency

↓

Compromise
Safety for Speed

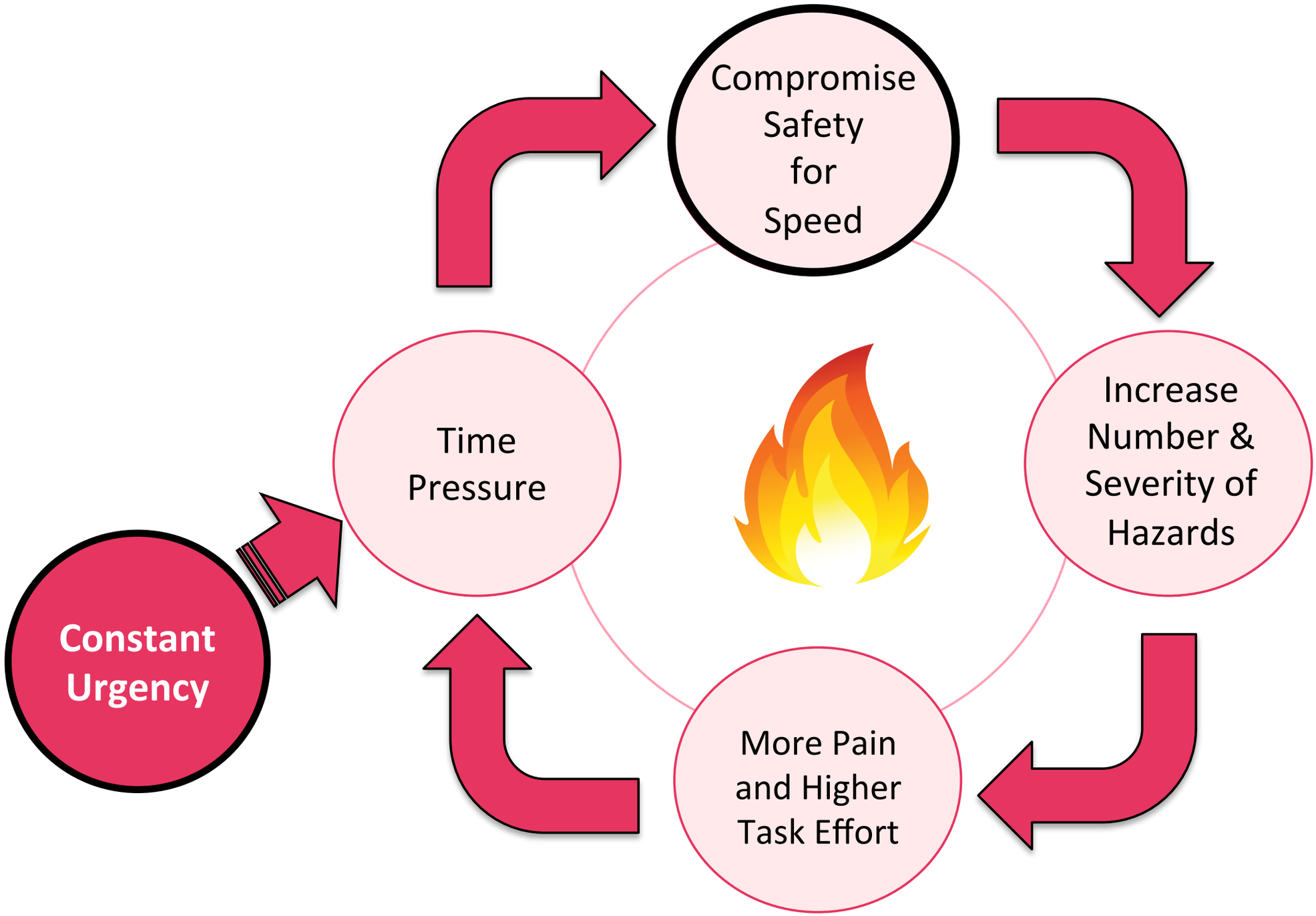
Cycle of Chaos

High-Risk Decision Habits



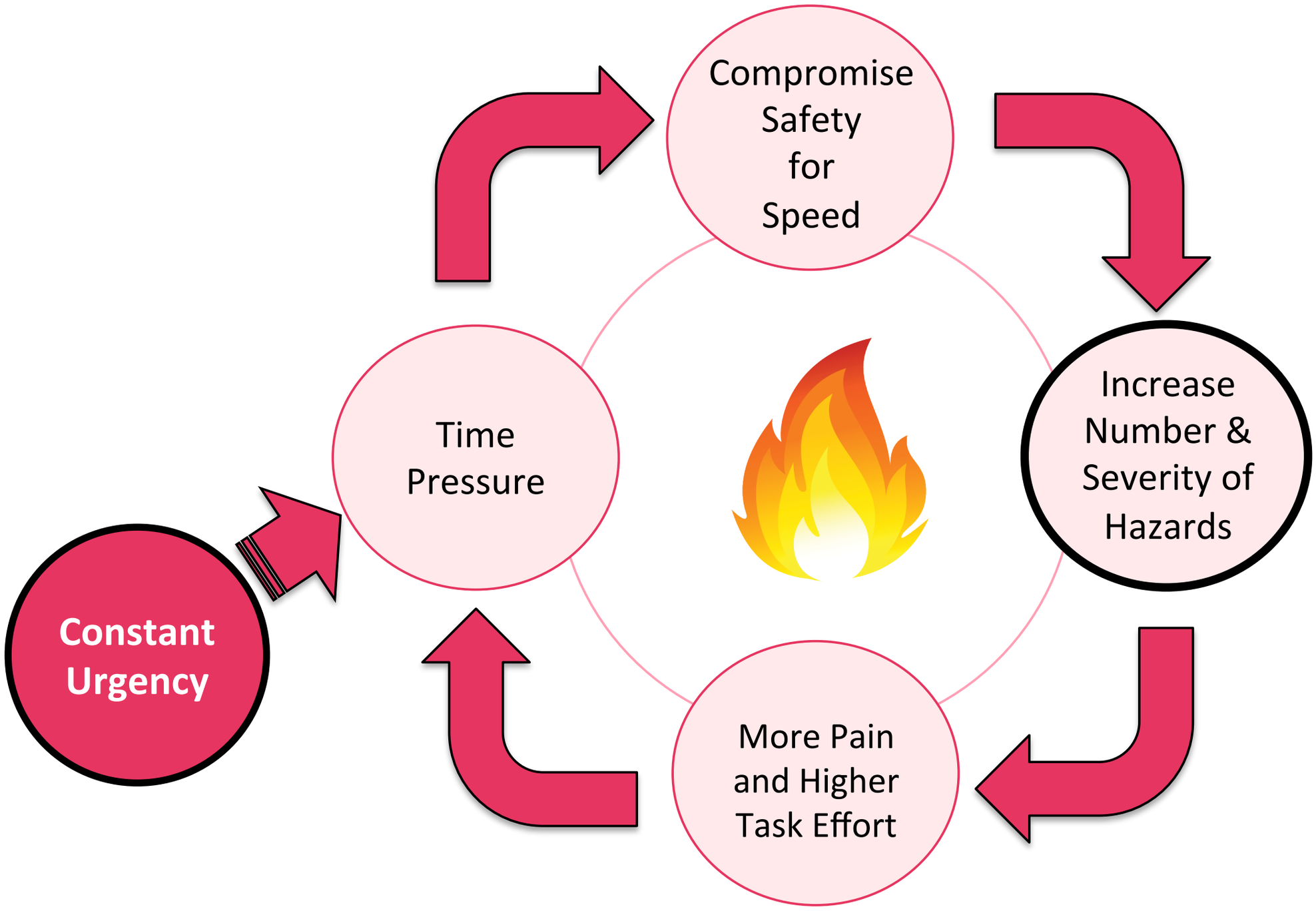
Cycle of Chaos

High-Risk Decision Habits



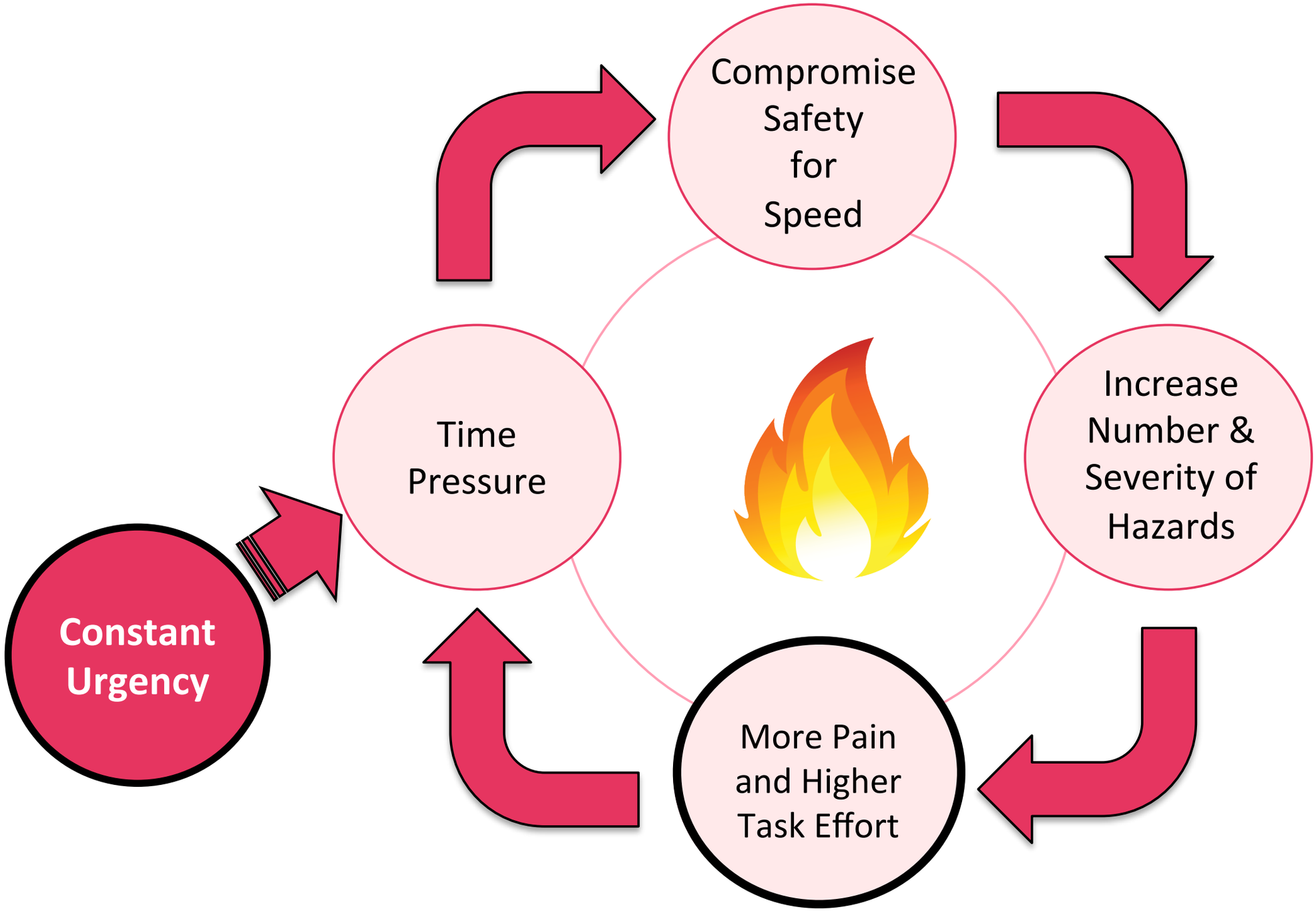
Cycle of Chaos

High-Risk Decision Habits



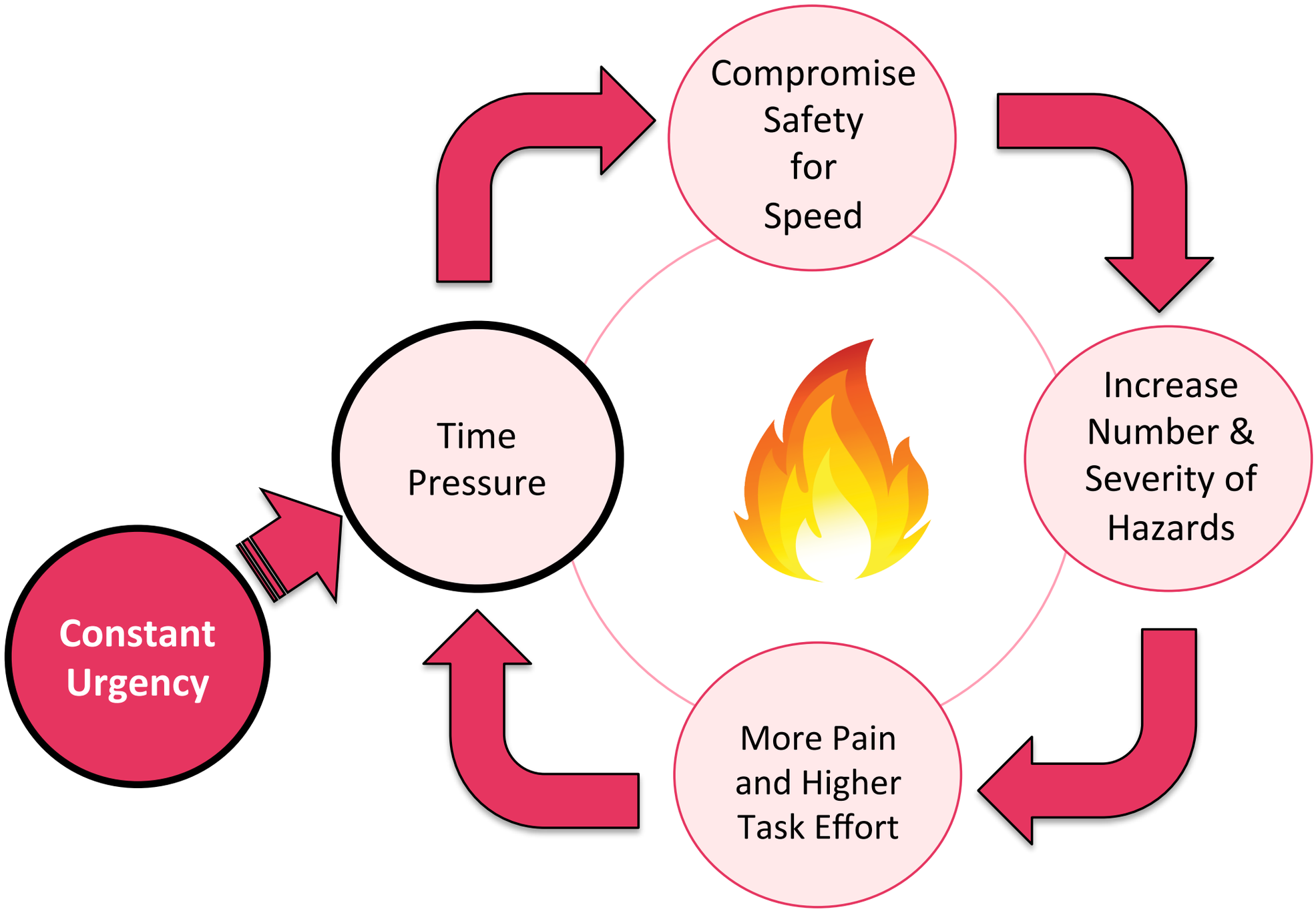
Cycle of Chaos

High-Risk Decision Habits

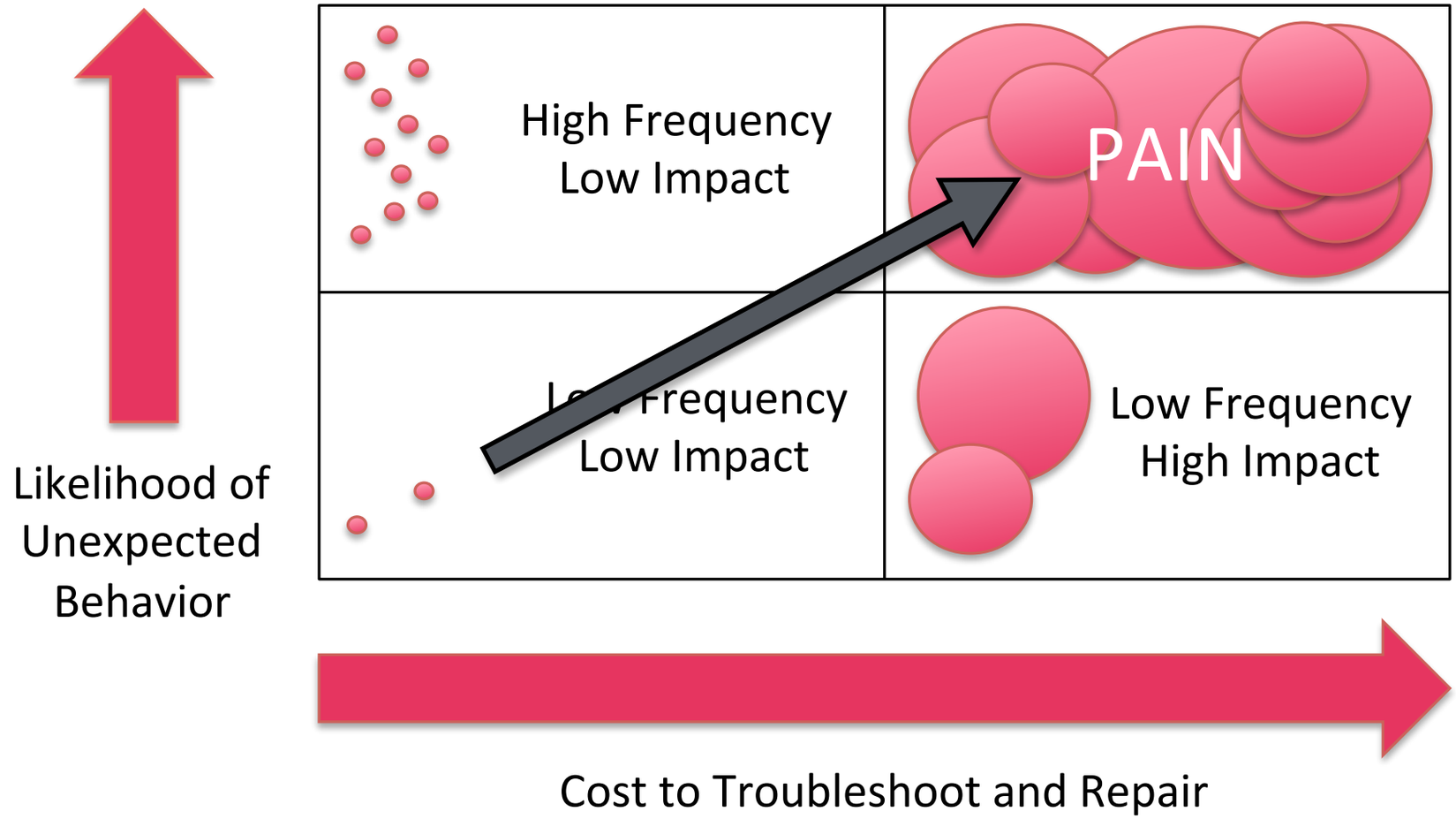


Cycle of Chaos

High-Risk Decision Habits

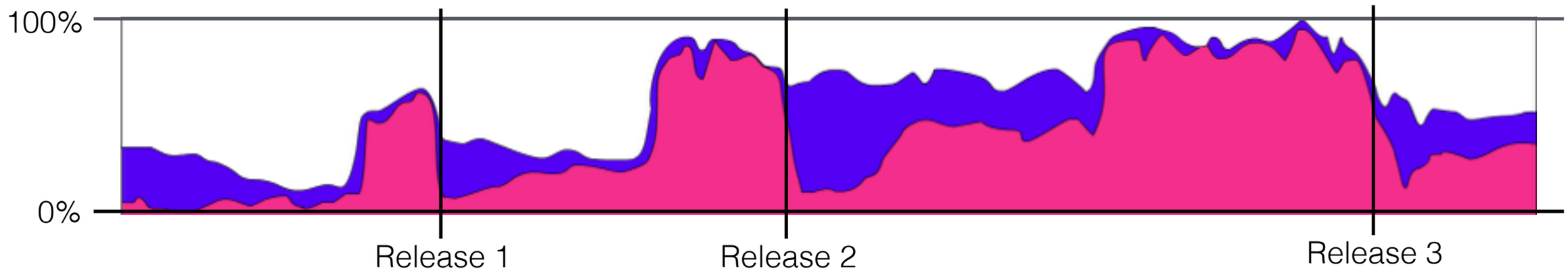


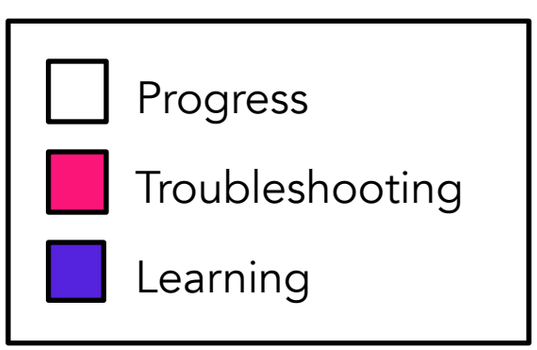
Risk Factors Multiply



Long-Term Trends

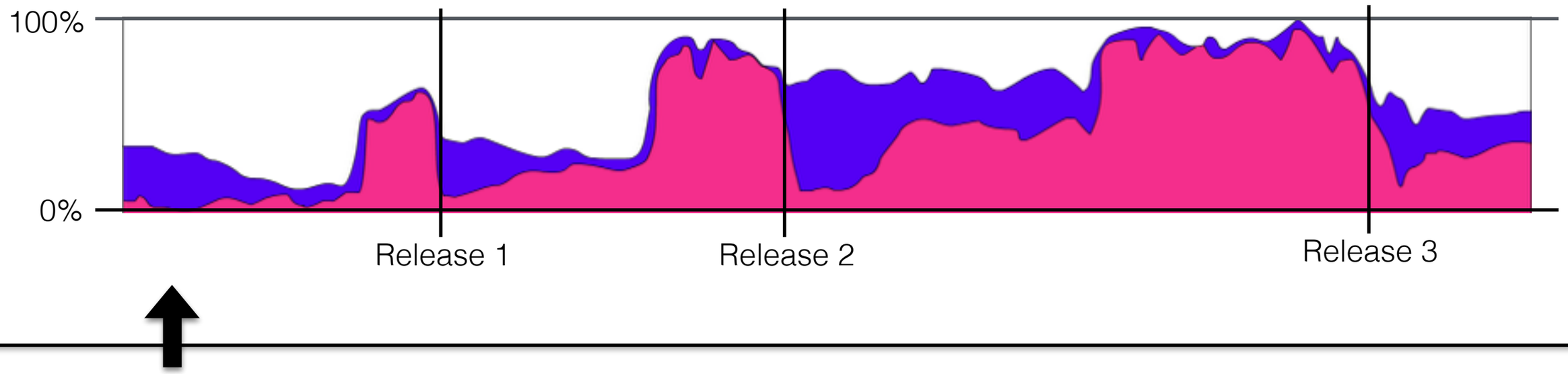
Percentage Capacity spent on Troubleshooting (red) and Learning (blue)



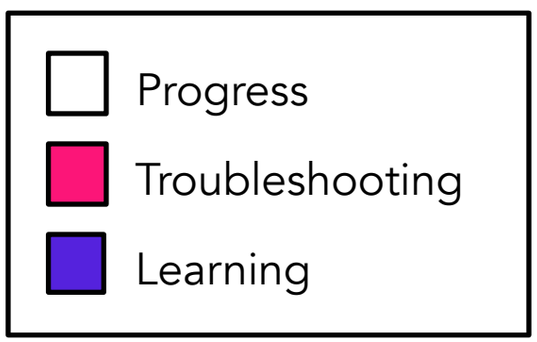


Long-Term Trends

Percentage Capacity spent on Troubleshooting (red) and Learning (blue)

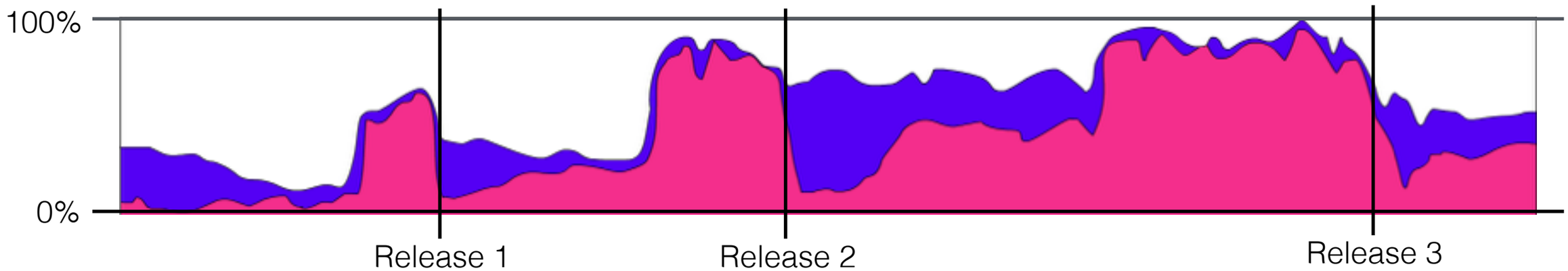


Learning is front-loaded

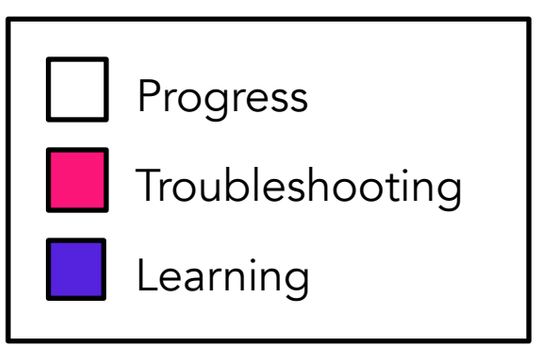


Long-Term Trends

Percentage Capacity spent on Troubleshooting (red) and Learning (blue)

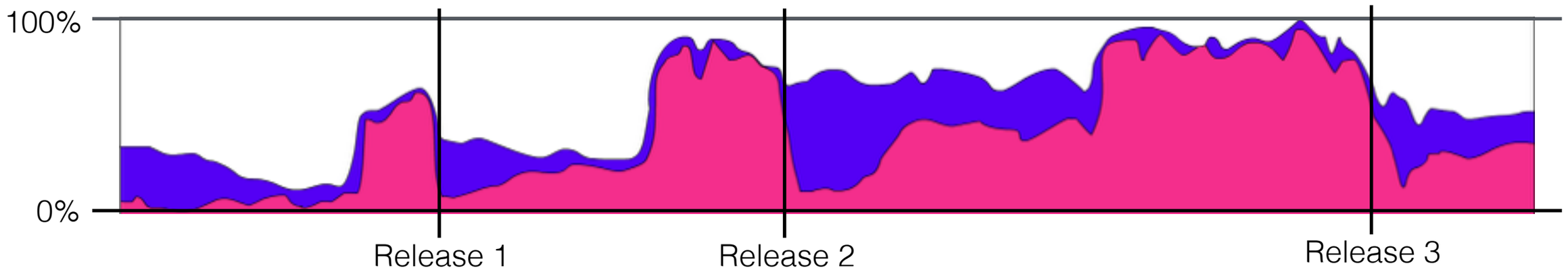


Validation is deferred

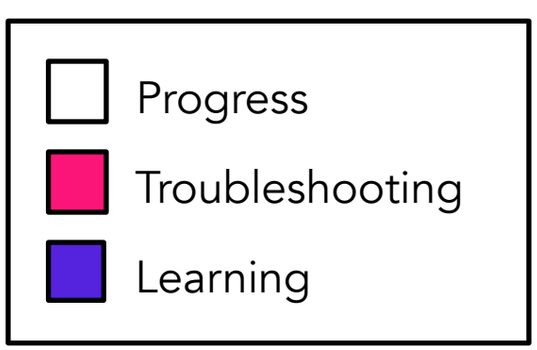


Long-Term Trends

Percentage Capacity spent on Troubleshooting (red) and Learning (blue)

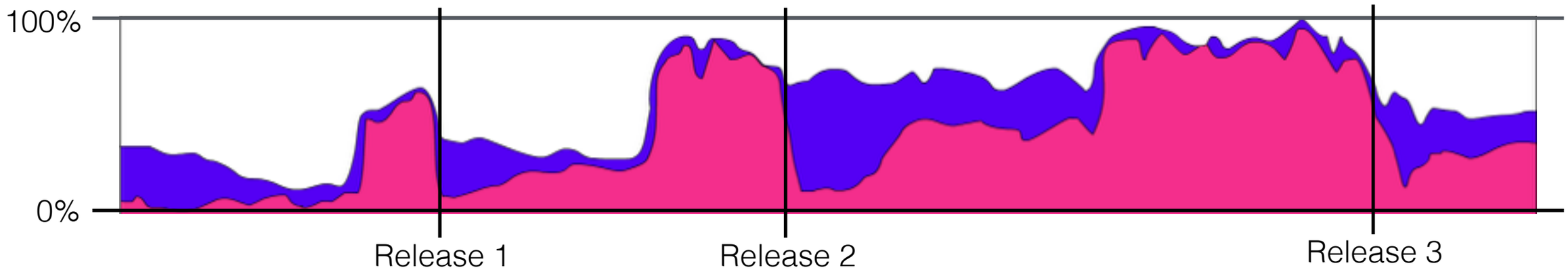


Baseline friction keeps rising

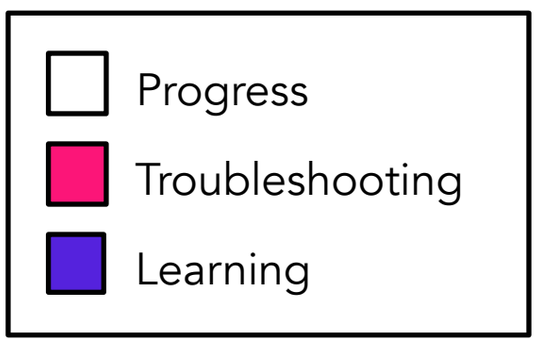


Long-Term Trends

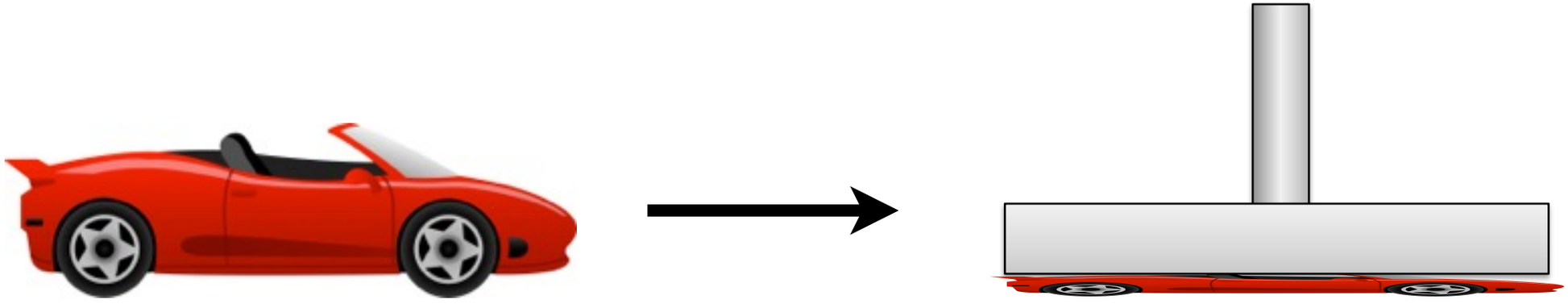
Percentage Capacity spent on Troubleshooting (red) and Learning (blue)



Unpredictable work stops fitting in the timebox

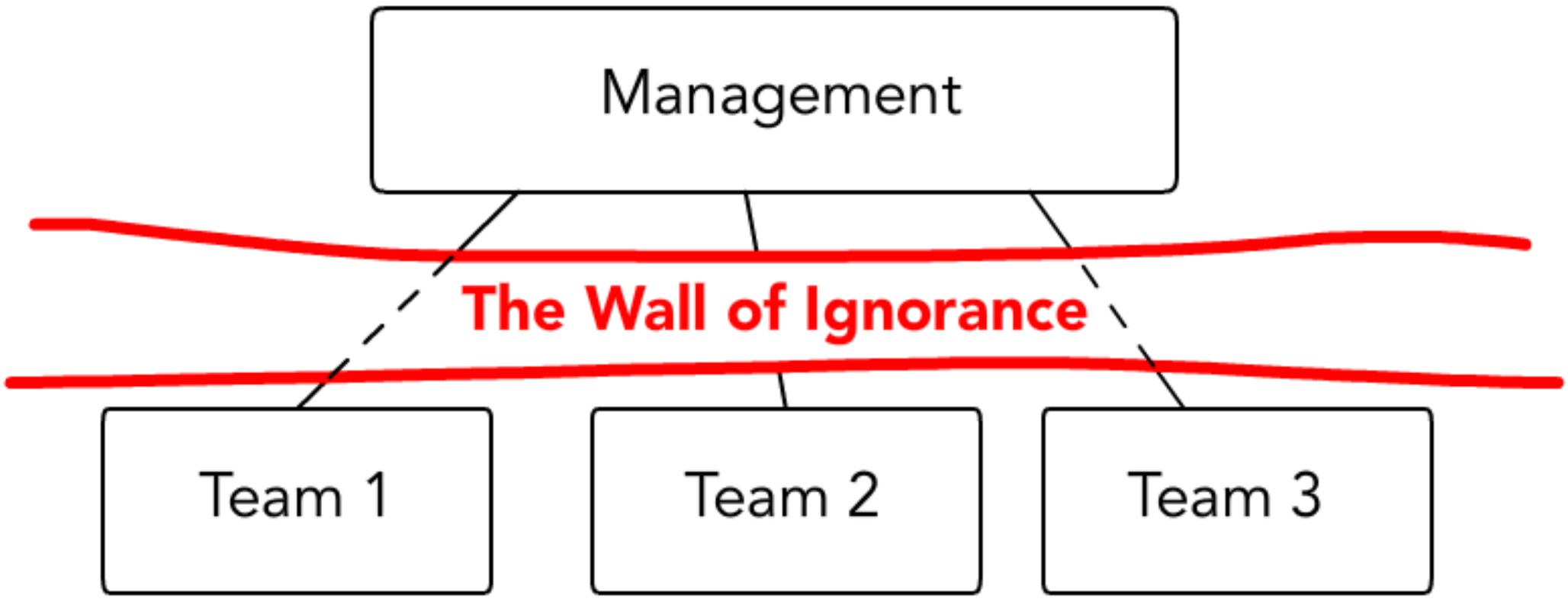


The Effects of Business Pressure



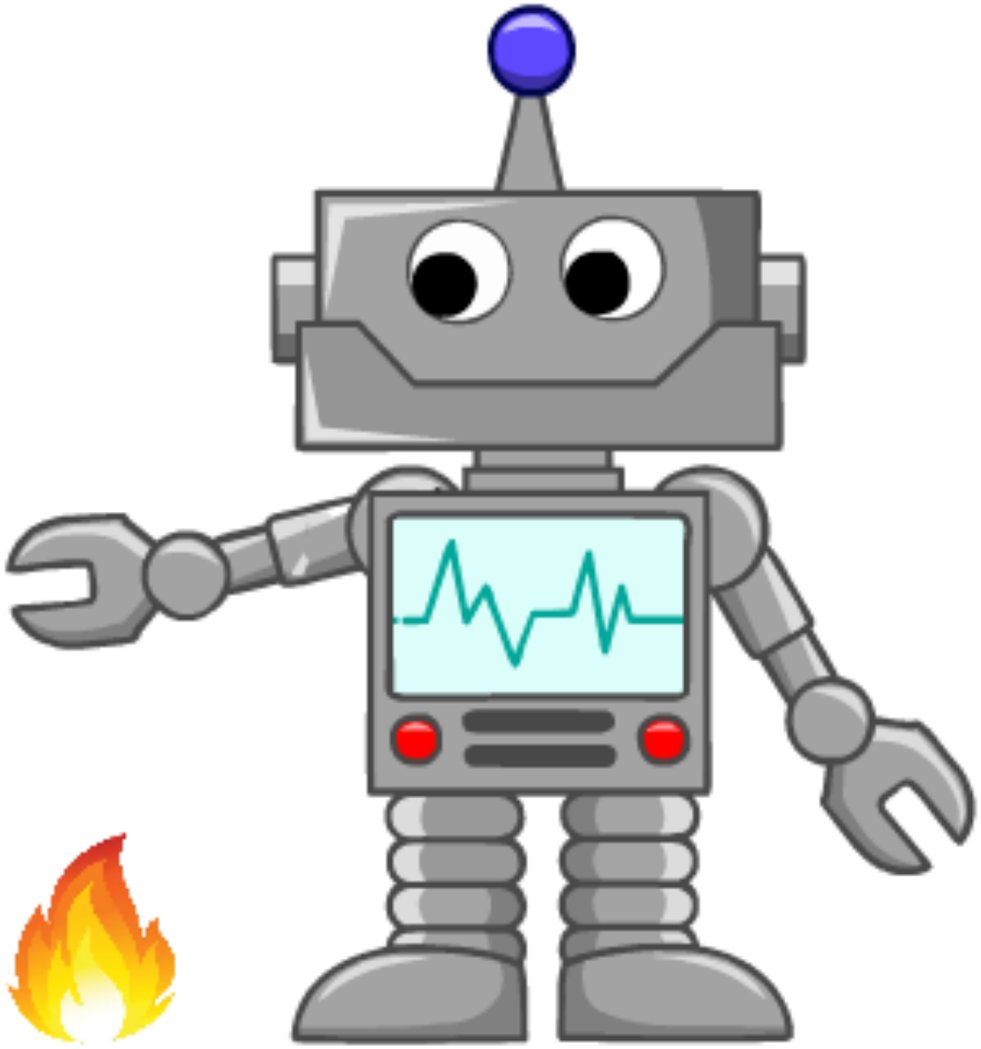
The Solution?

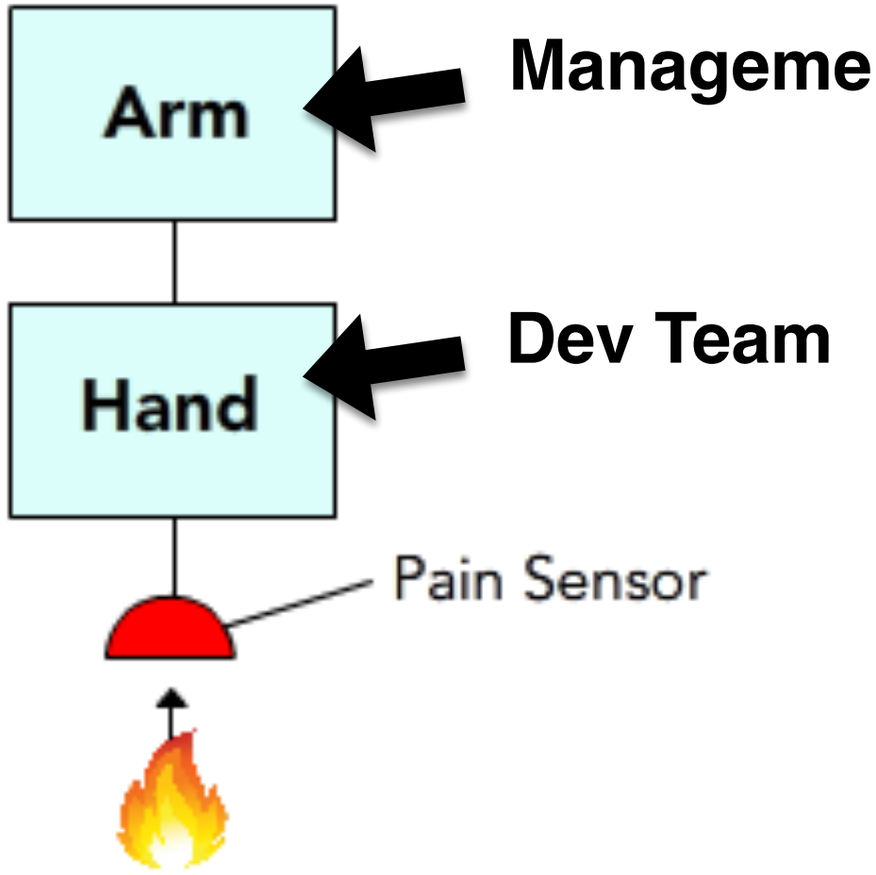
We have to solve this problem:



The Challenge: Decision-Making is Distributed.

What if our organization was a robot?

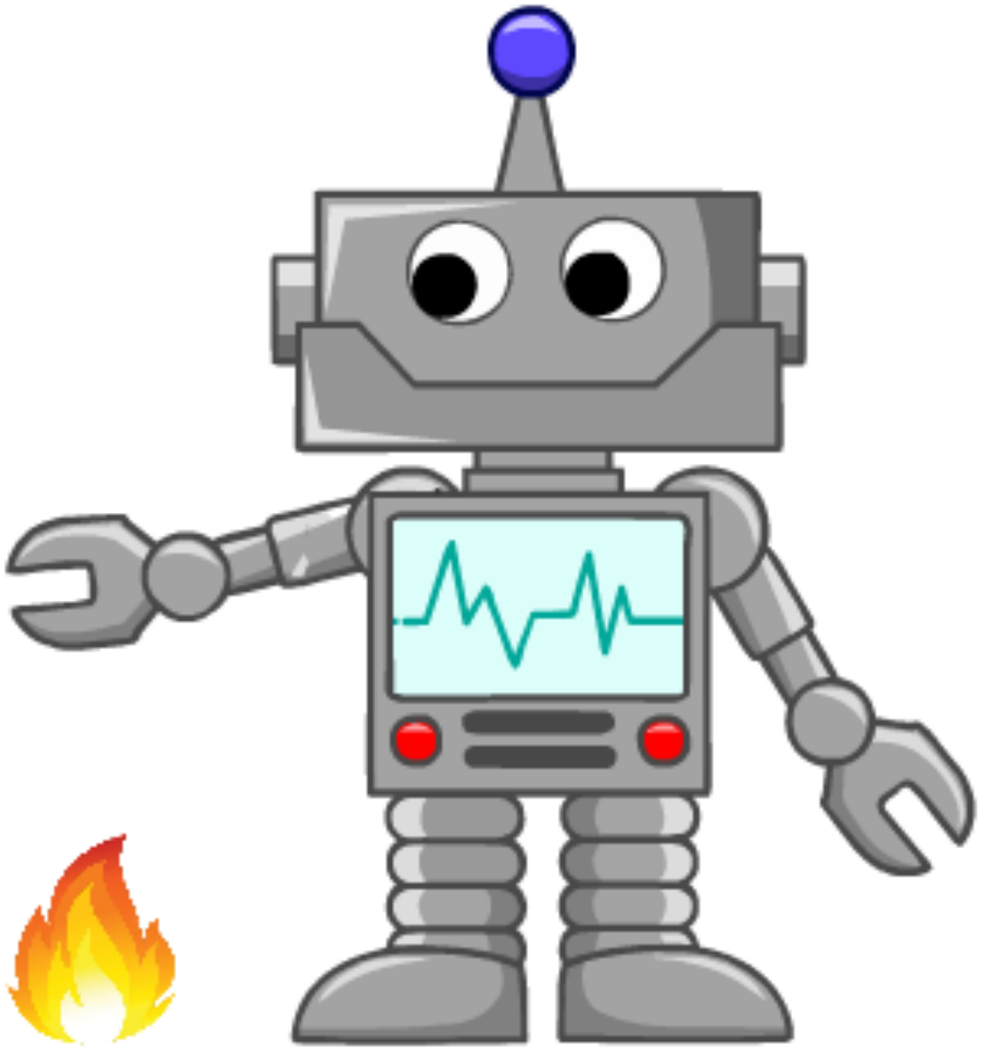


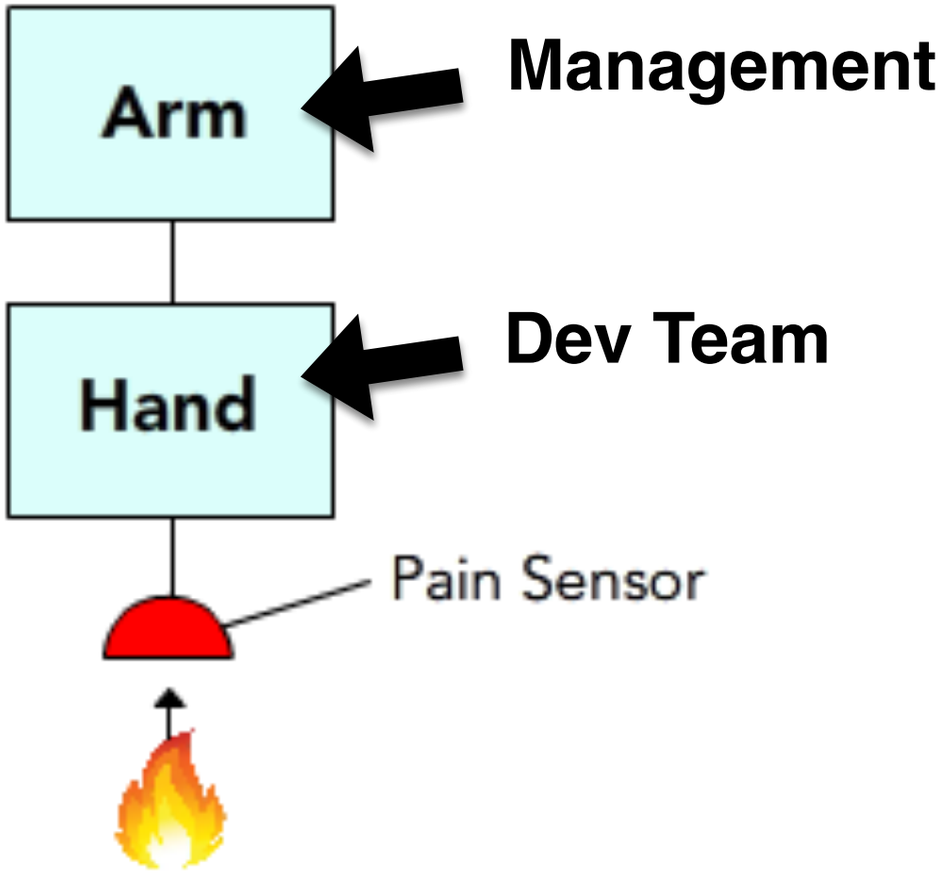


Fire x1

Nothing's happening...

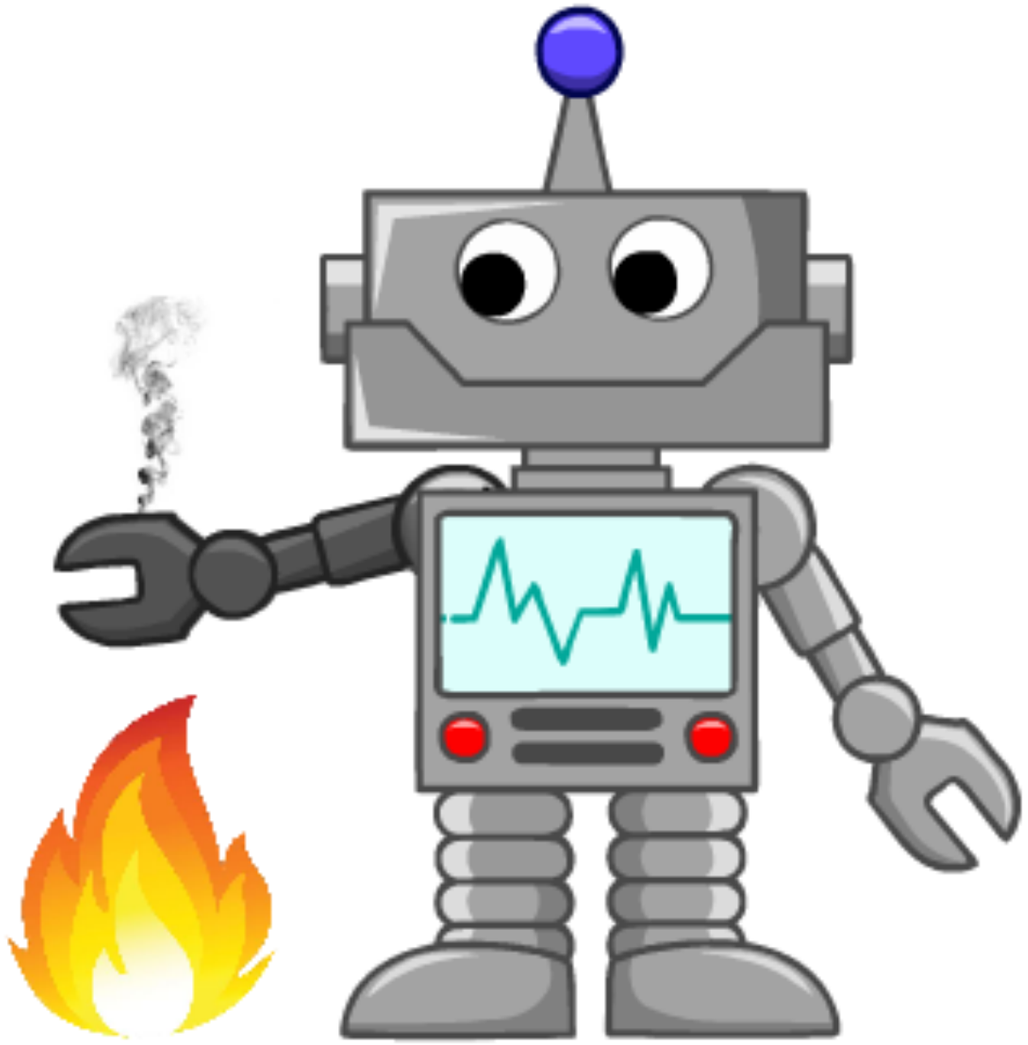
What if our organization was a robot?

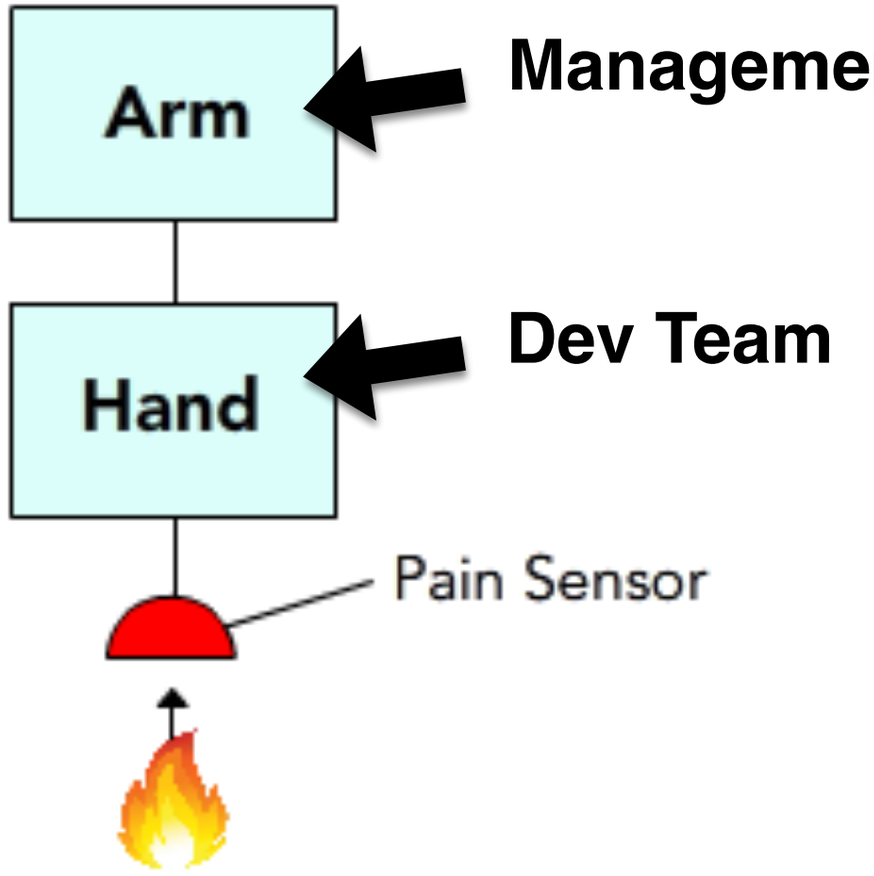




Fire x10

What if our organization was a robot?

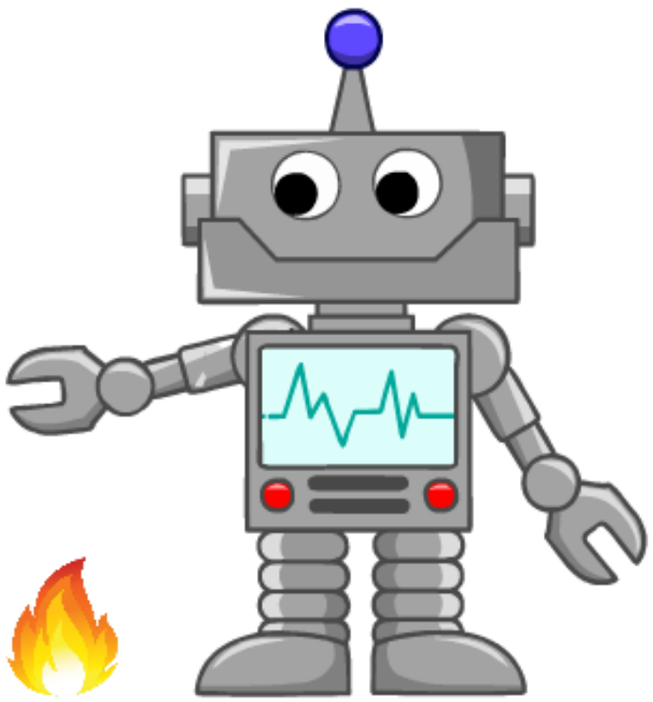


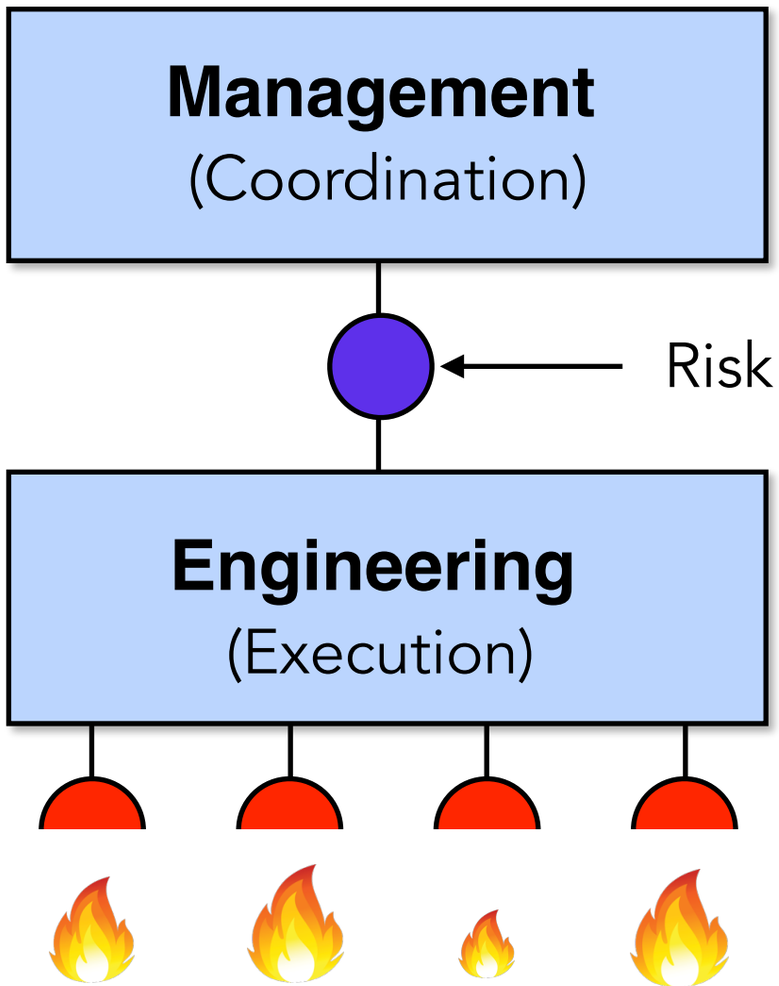


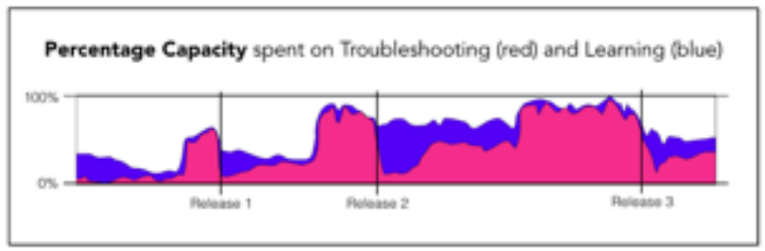
Fire x10

If the feedback loop is broken, **we burn**.

How do we repair the broken feedback loop?

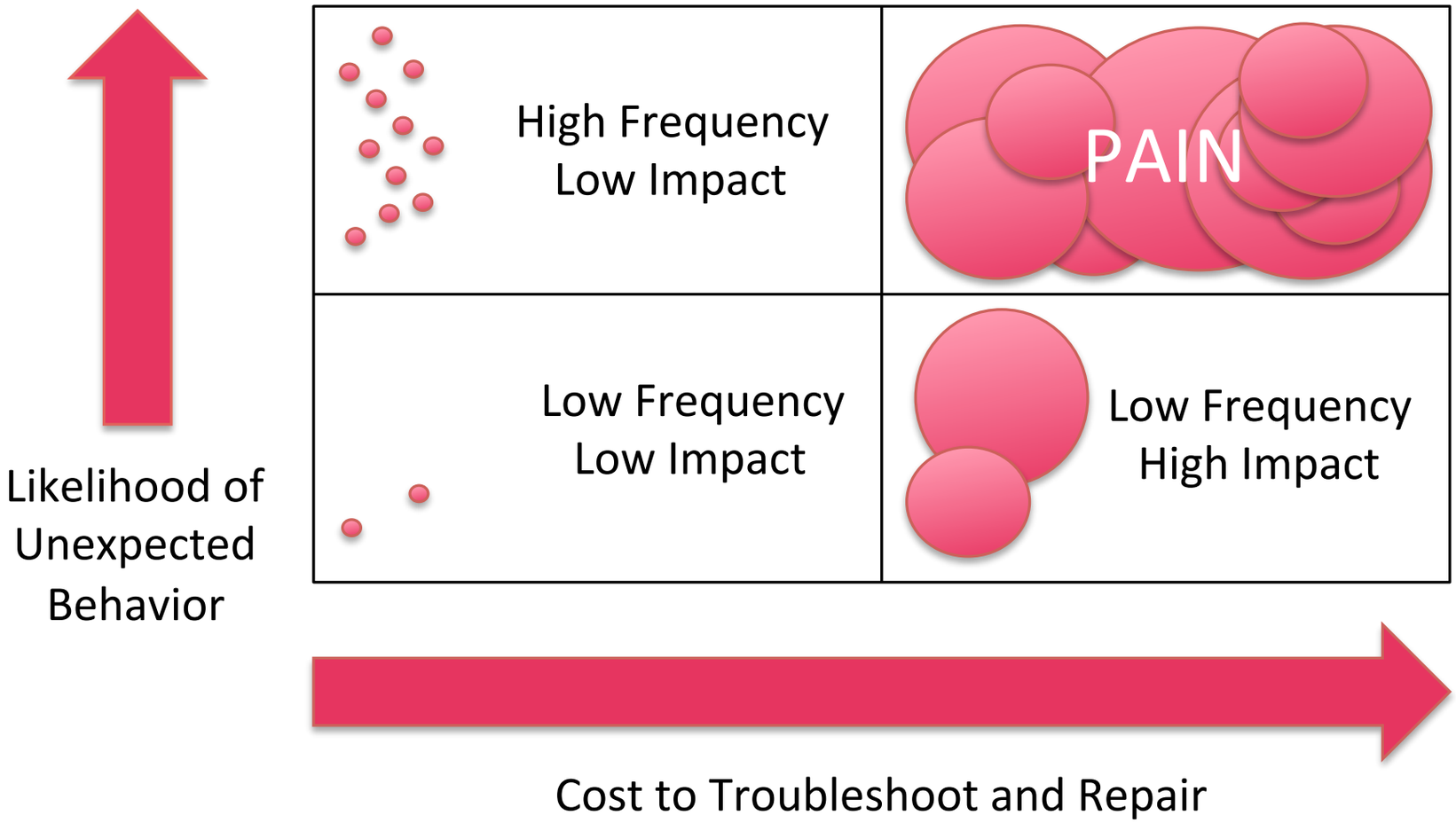




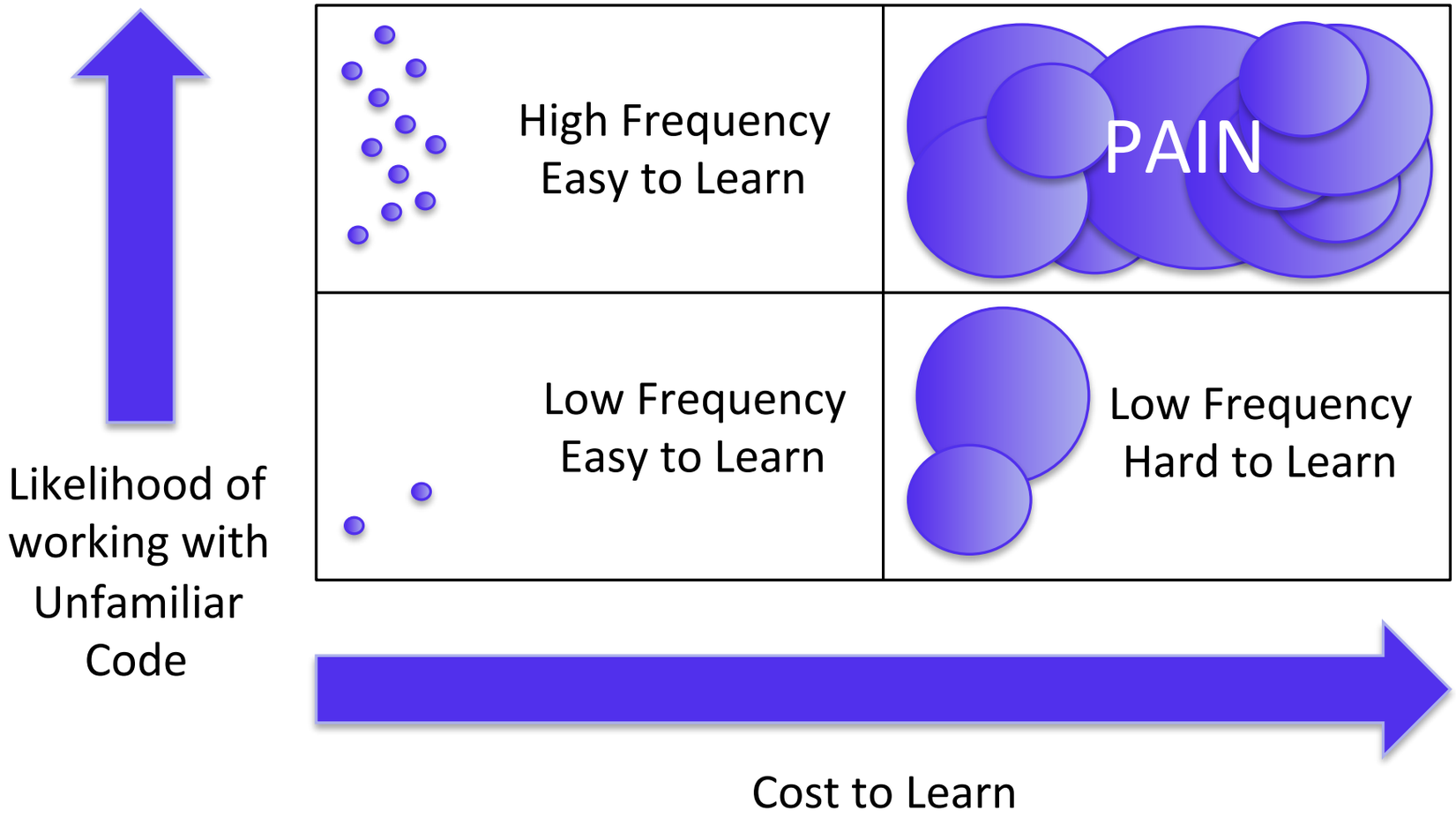


Risk is the *bridge language*.

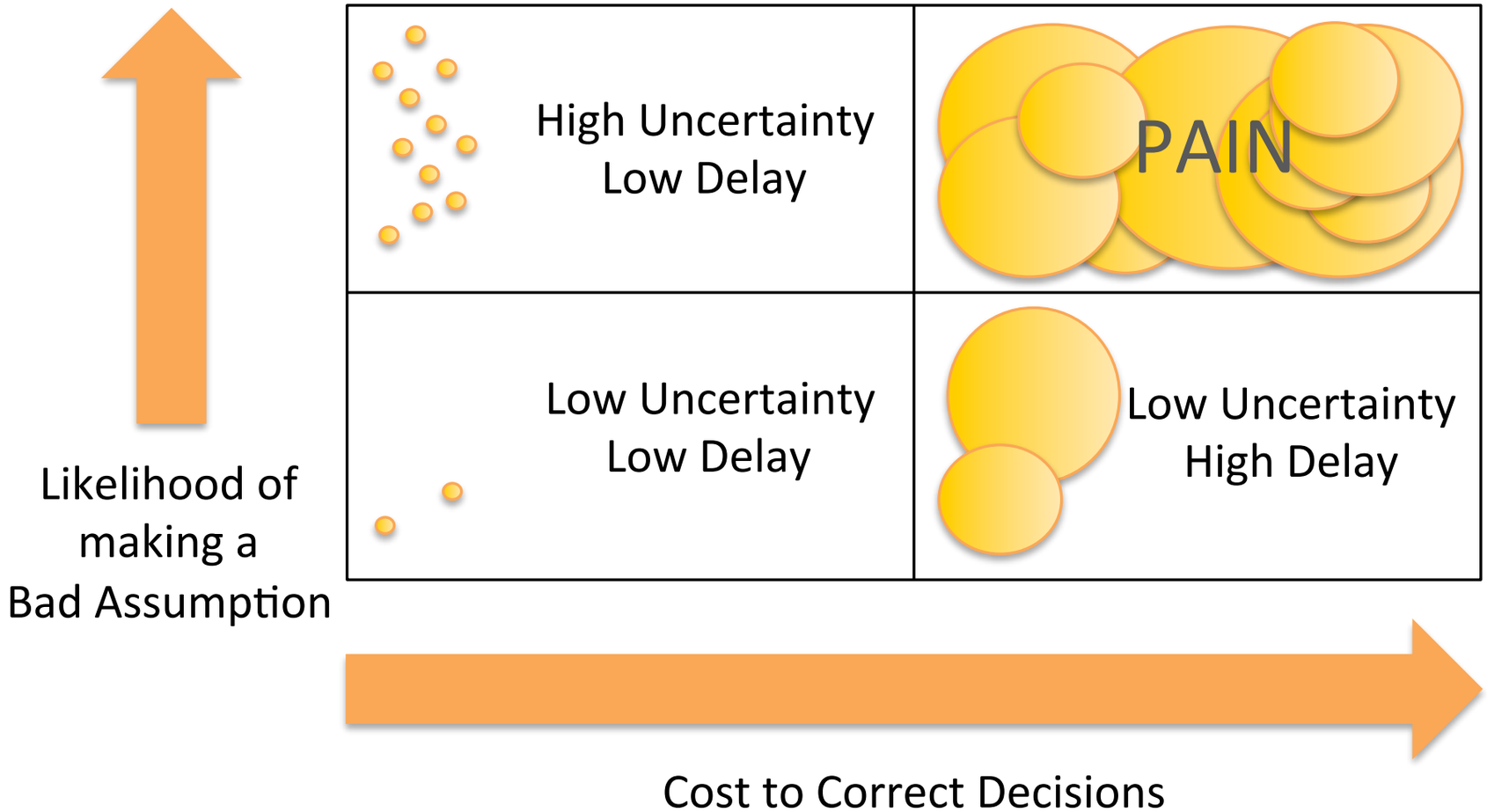
Quality Risk (Troubleshooting)



Familiarity Risk (Learning)



Assumption Risk (Rework)



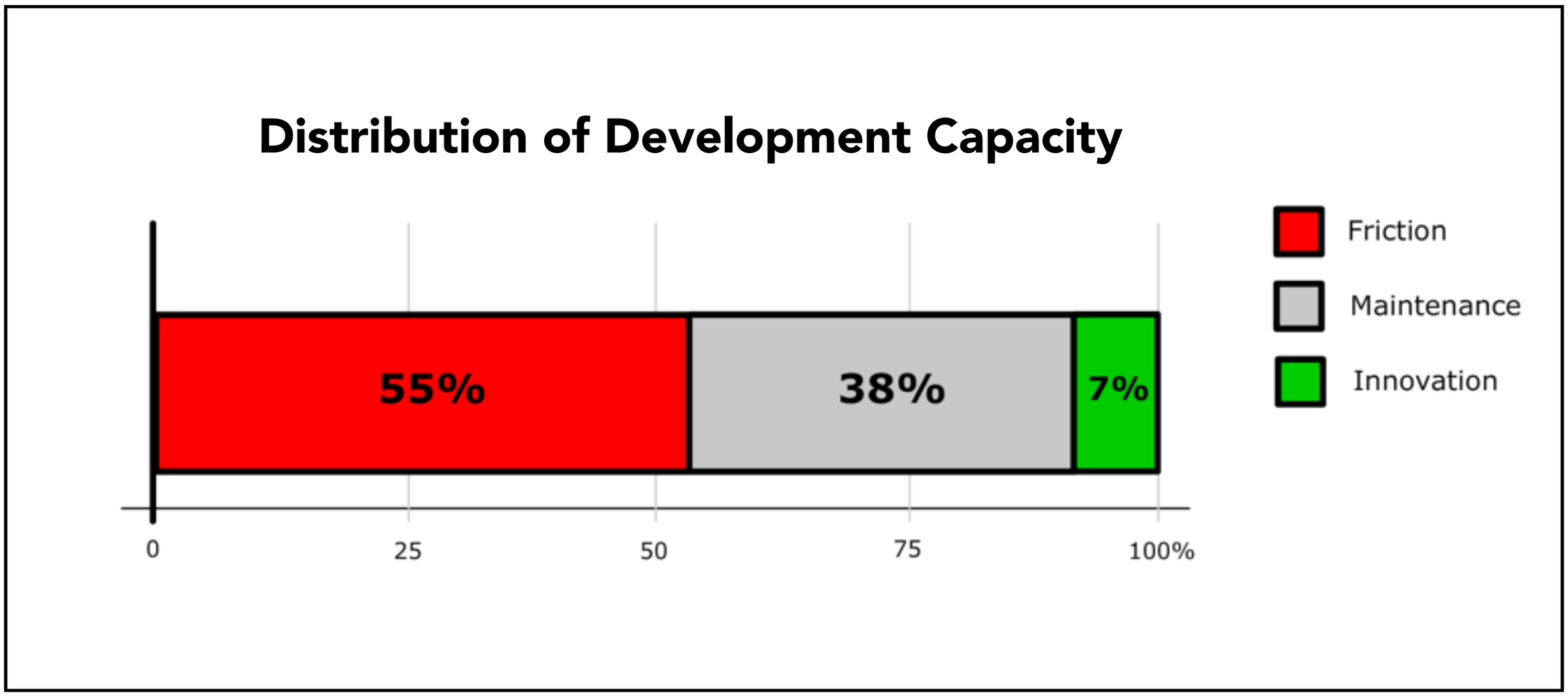
Explain Problems in Terms of Risk (Gambling)

Save 40 hours in direct costs
(leave the toy on the stairs)

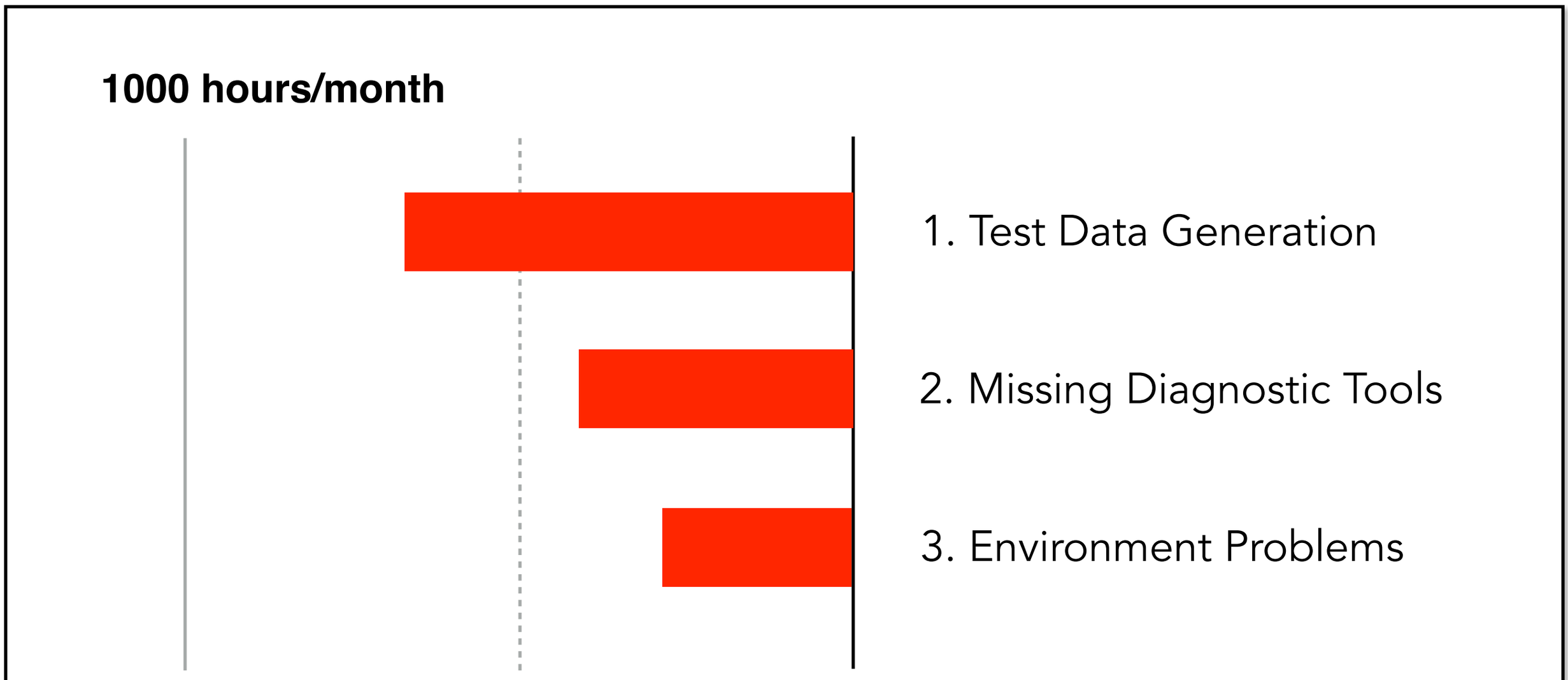
Increase chances of losing 1000 hours by 20%
(tripping and falling)

Decisions that save a few hours → **Side-effects** that cost several hours

Over the long-term, probability wins.



Make the Problems *Visible* to your Manager



Save time by skipping data tools (~80 hours)

→

Side-effects of Troubleshooting time (~700 hours/month)

Take Responsibility and Ask for Support

3-Month Improvement Trial

Dedicated resources (1 or 2 developers)

Dev team uses data to identify biggest problems and prioritizes with management

Dev team shares data & progress each month

"Will you **help us** turn this project around?"

Broken Focus

Lesson Learned: Make the Decision to Lead

“How can I provide better visibility for management?”

Top 5 Reasons Improvements Fail

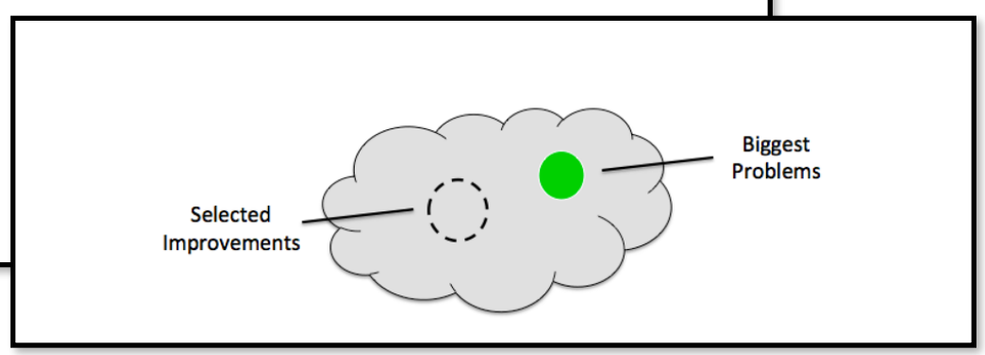
BROKEN TARGET

BROKEN VISIBILITY

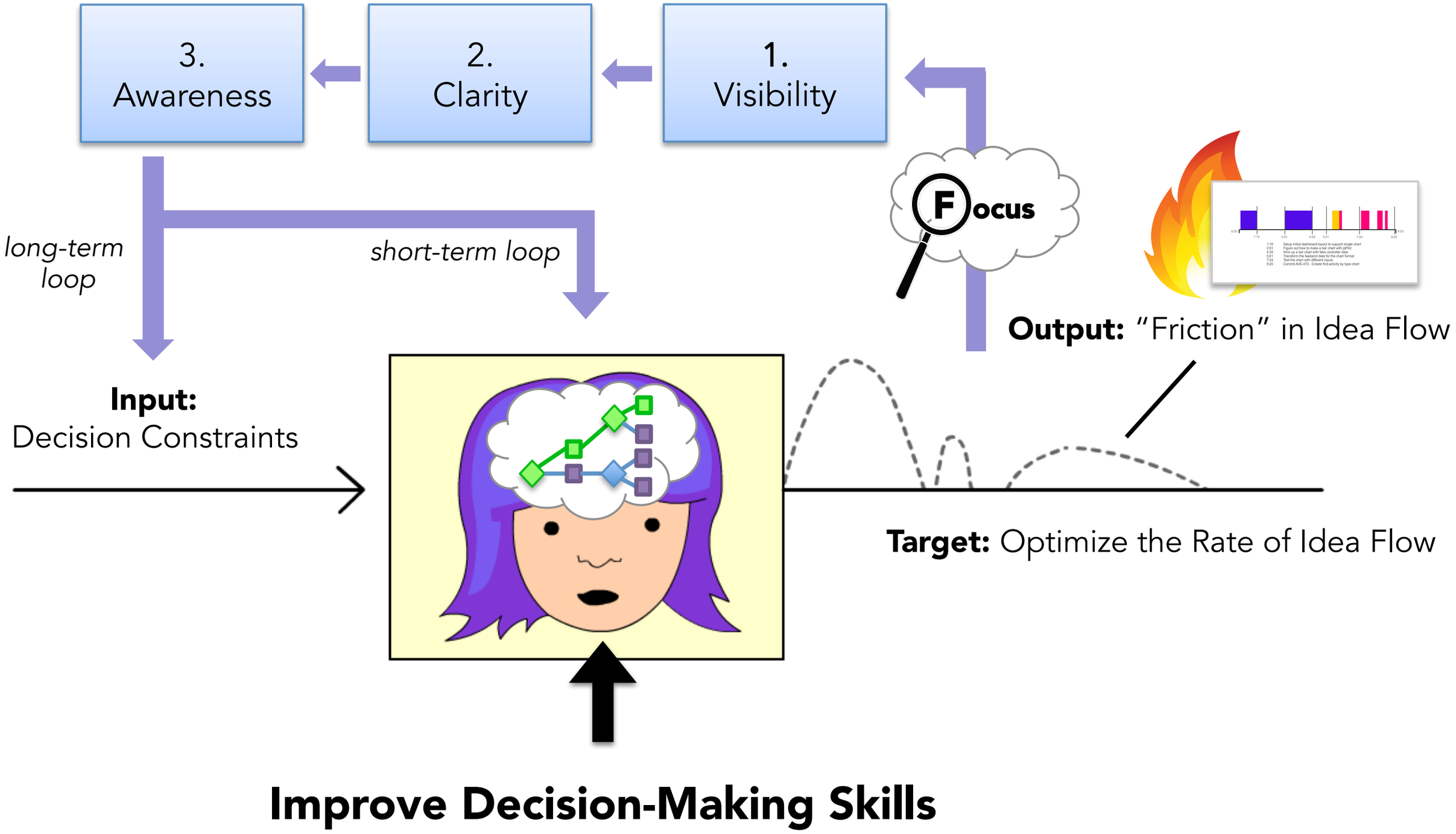
BROKEN CLARITY

BROKEN AWARENESS

BROKEN FOCUS

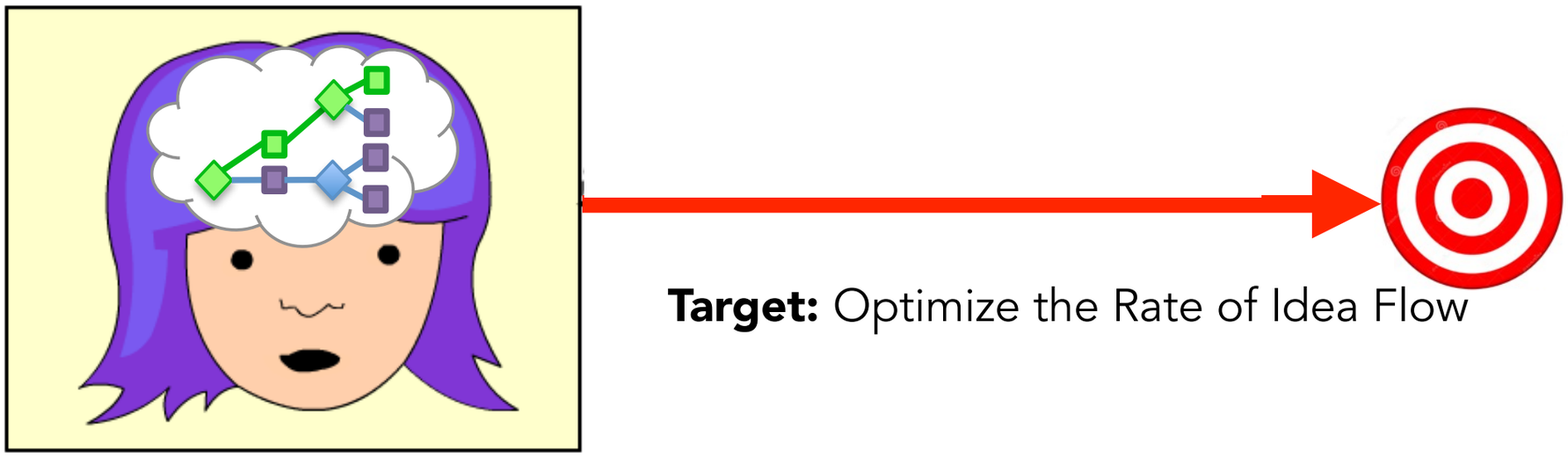


Idea Flow Learning Framework



Improve Decision-Making Skills

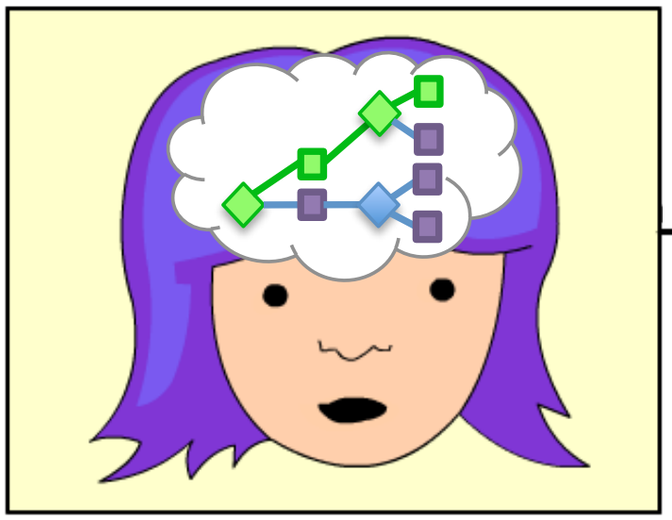
Idea Flow Learning Framework



Target - The ***direction*** of “better”

Idea Flow Learning Framework

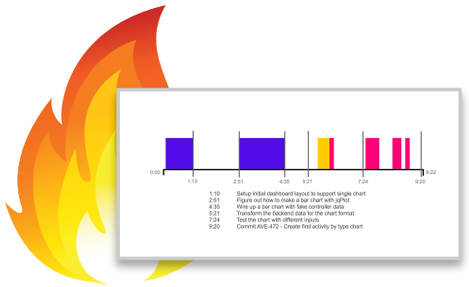
Input:
Decision Constraints



Target: Optimize the Rate of Idea Flow

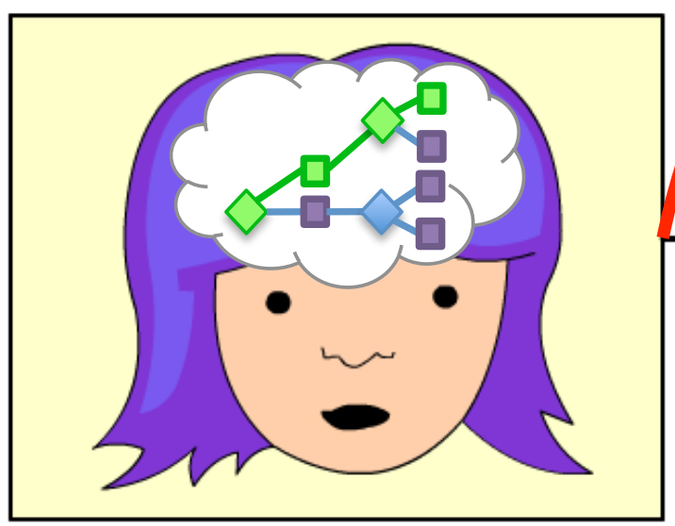
Input - The ***constraints*** that limit our short-term choices…

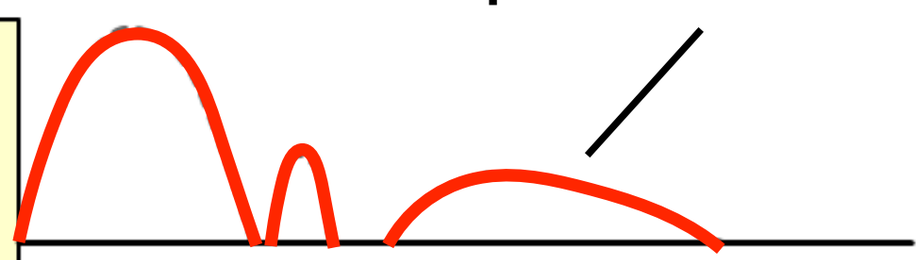
Idea Flow Learning Framework



Output: "Friction" in Idea Flow

Input:
Decision Constraints

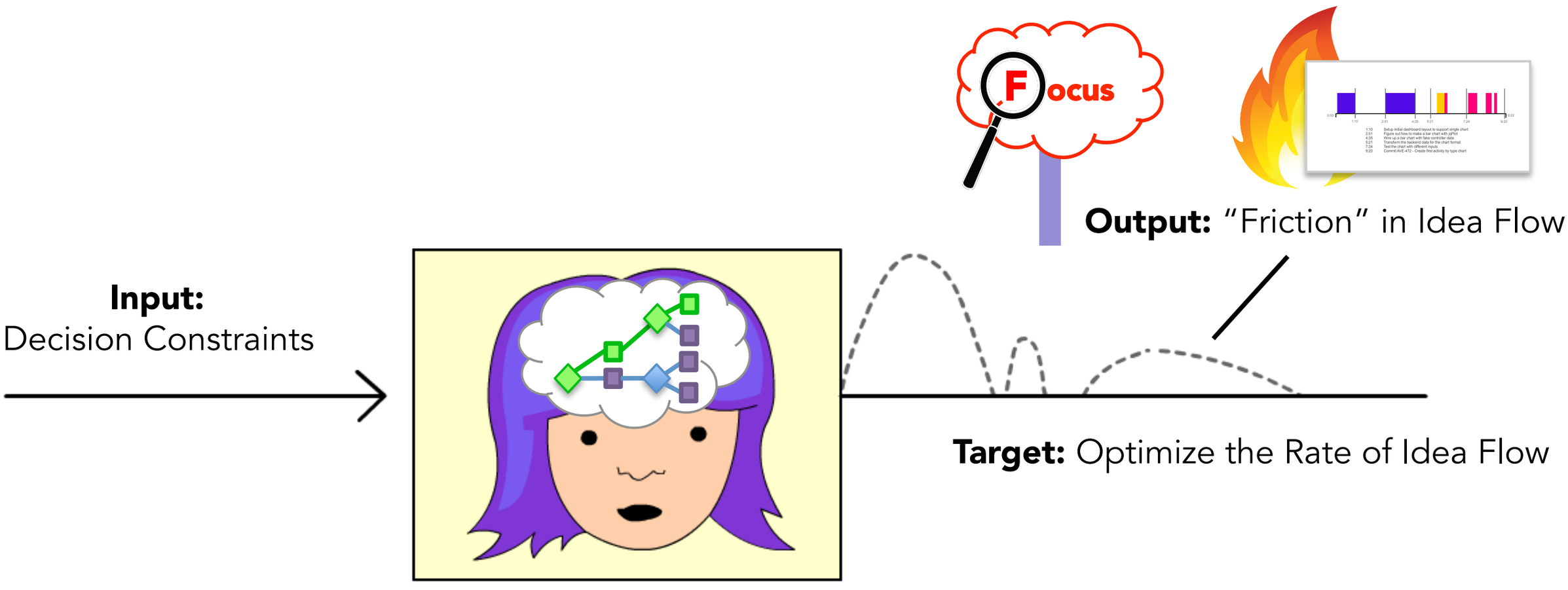




Target: Optimize the Rate of Idea Flow

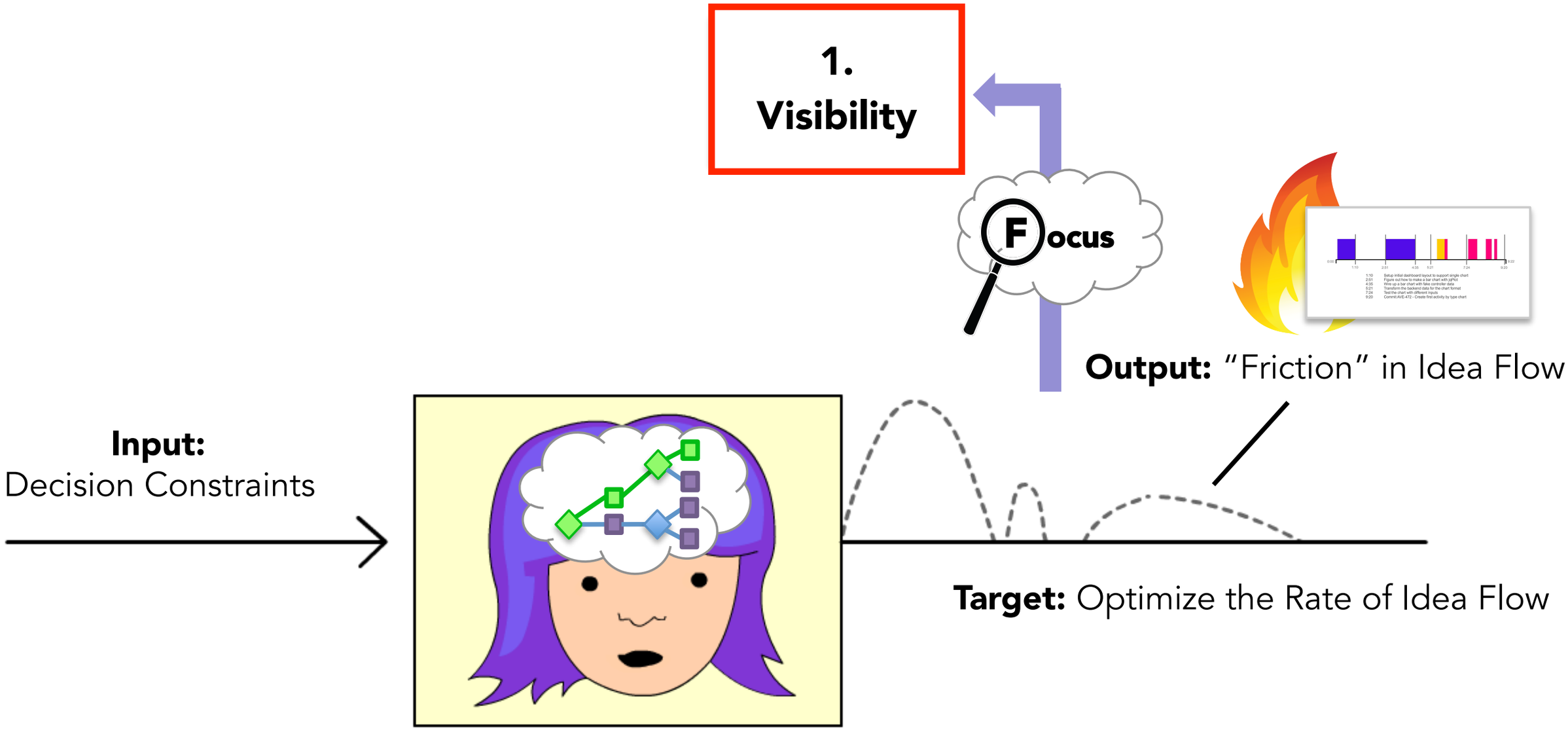
Output - The ***pain signal*** we're trying to improve

Idea Flow Learning Framework



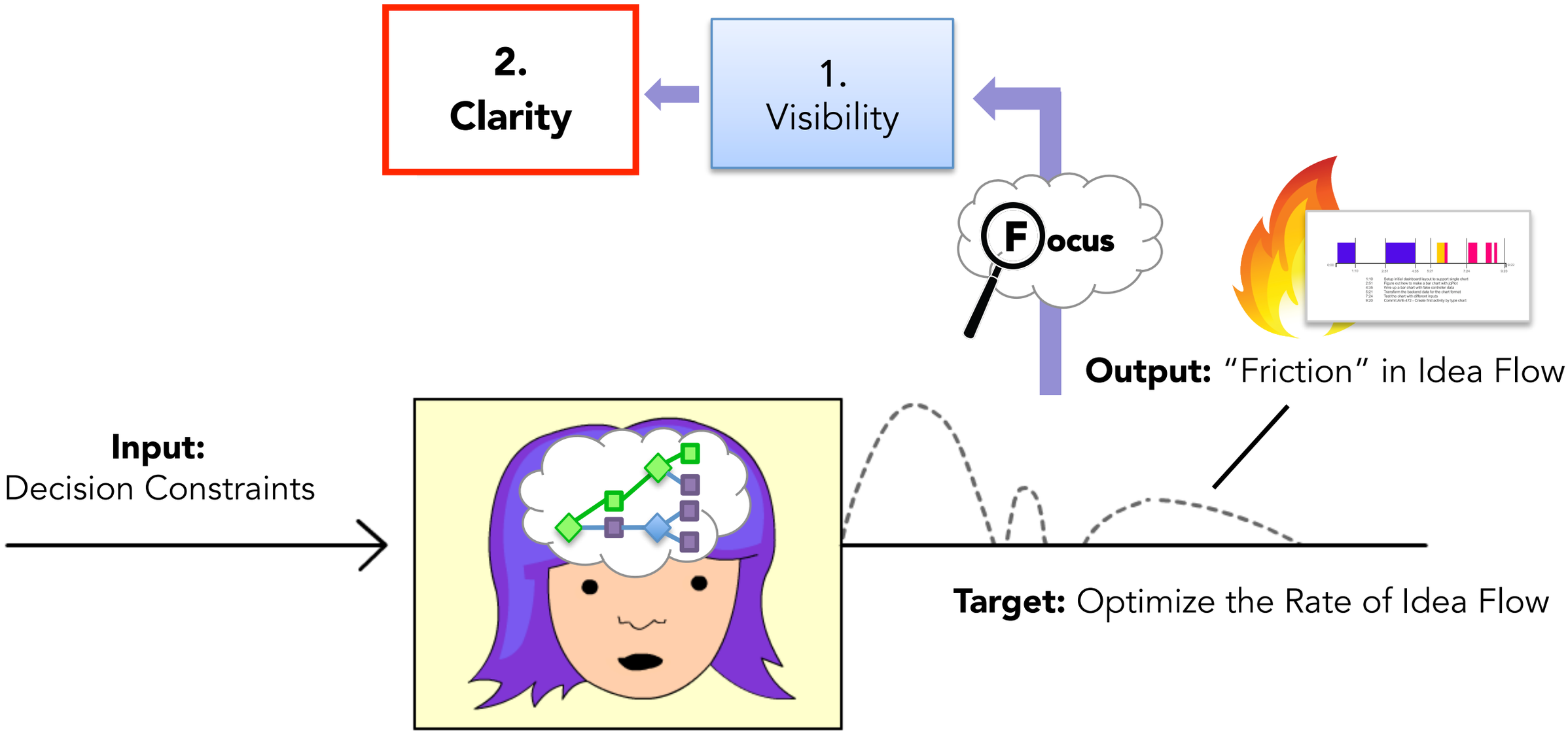
Focus on the ***biggest pain***…

Idea Flow Learning Framework



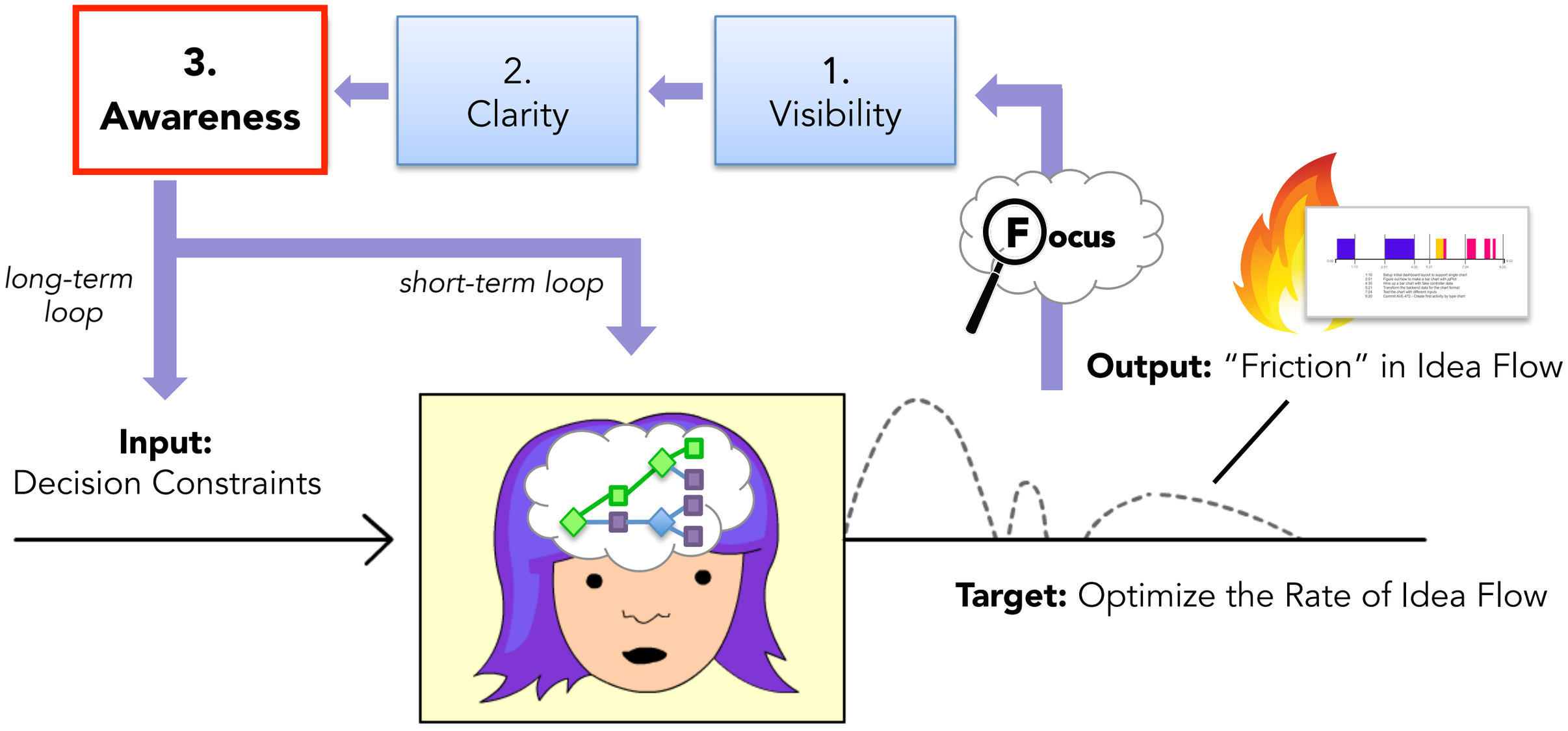
1. Visibility - *Identify* the specific patterns.

Idea Flow Learning Framework



2. Clarity - ***Understand*** cause and effect.

Idea Flow Learning Framework



3. Awareness - Learn strategies to ***avoid*** the pain.

Repair the *Broken* Feedback Loop!

Target: Focus on the Nails.

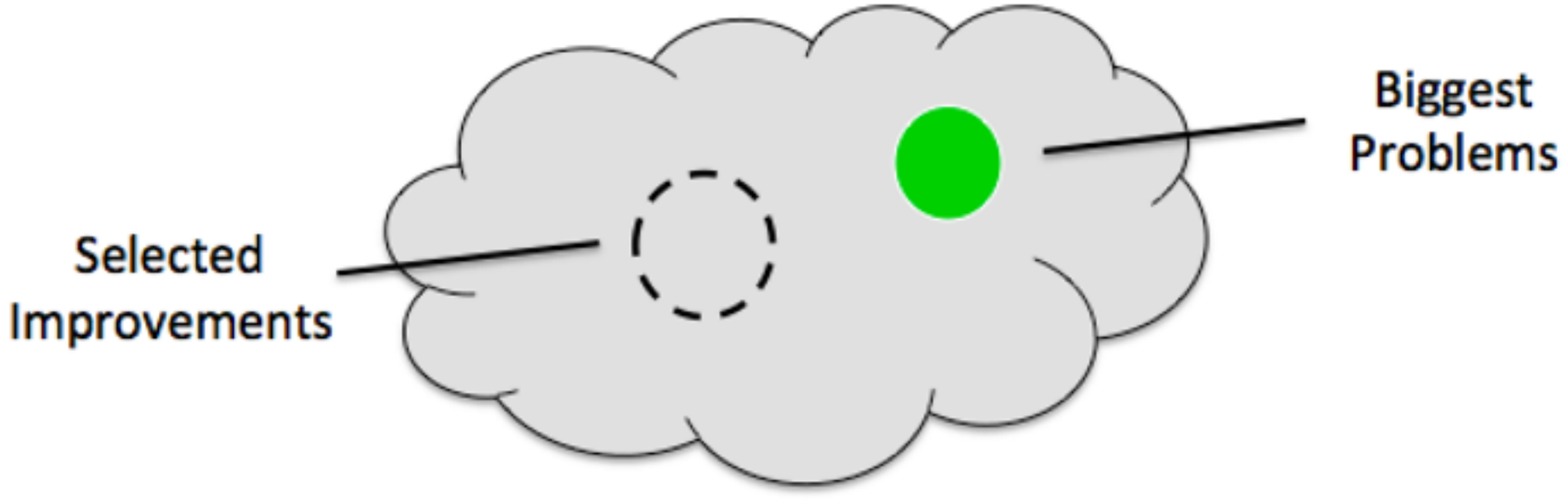
Visibility: Install a Pain Sensor.

Clarity: Stop making generalizations.

Awareness: Stop and think.

Focus: Make the decision to lead.

The hard part isn't solving the problems,
it's ***identifying*** **the right problems to solve**.



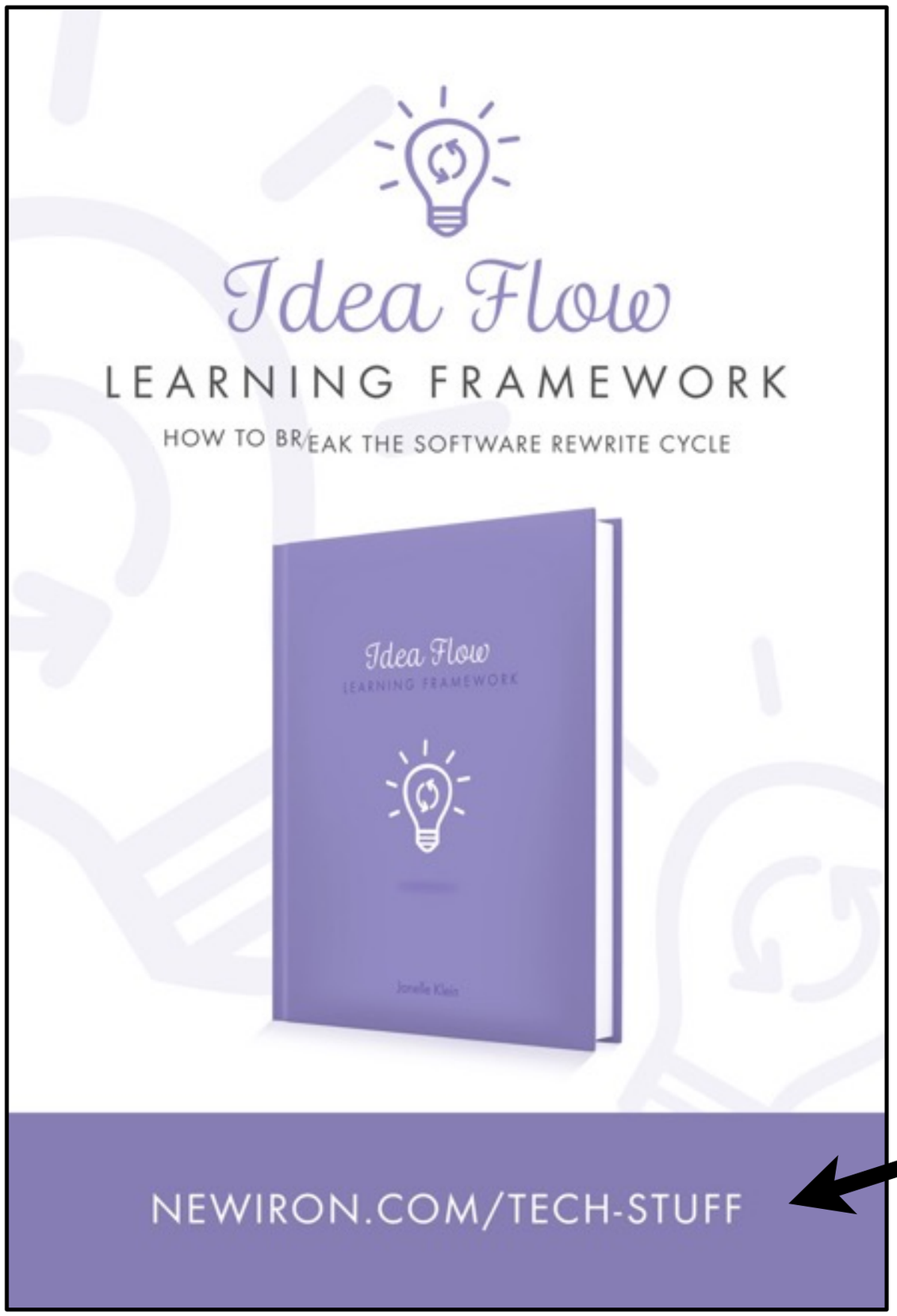
This Talk...

Why is visibility critical to success?

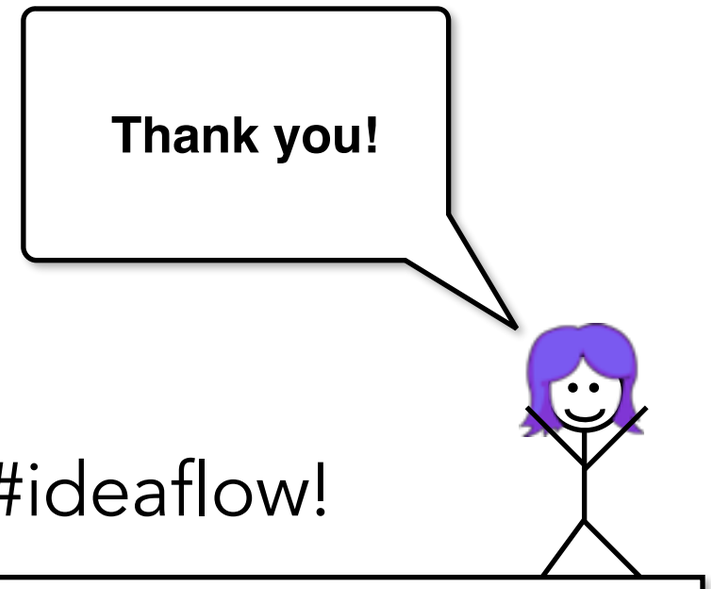
Visibility ***enables*** the feedback loop.

Free Online Training
and Community Support:

Idea Flow
Learning Framework



IFM Tools available at:
github.com/ideaflow/tools



Free e-book if you sign up before publish day! (Feb 2016)

janelle@newiron.com

@janellekz

2016 Tour Schedule

2/15-17: DevNexus 2016 - Atlanta, GA

4/4-7: ArchConf 2016 - San Diego, CA

4/29-5/1: NFJS - Columbus, OH

5/20-22: NFJS - Dallas, TX

6/8-10: NFJS - San Diego, CA

7/8-10: NFJS - Austin, TX

7/19-22: UberConf 2016 - Denver, CO

8/19-20: NFJS - Raleigh, NC

9/16-18: NFJS - Atlanta, GA

9/30-10/2: NFJS - Boston, MA

10/14-16: NFJS - Chicago, IL

10/21-22: NFJS - Minneapolis, MN

10/28 - 30: Seattle, WA

11/4-6: Reston, VA