JETBRAINS

Just-in-time Specification

Or Evolving Kotlin One Feature at a Time

Marat Akhin

PLSS in 30 seconds

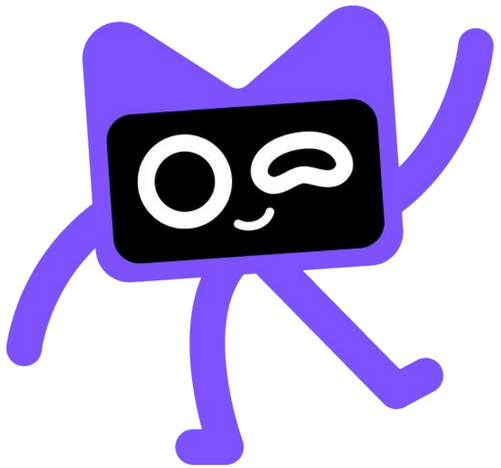
Kotlin is a multiplatform, statically typed, general-purpose programming language

- JetBrains is a single vendor
- There is only one compiler implementation

No need for standardisation or specification

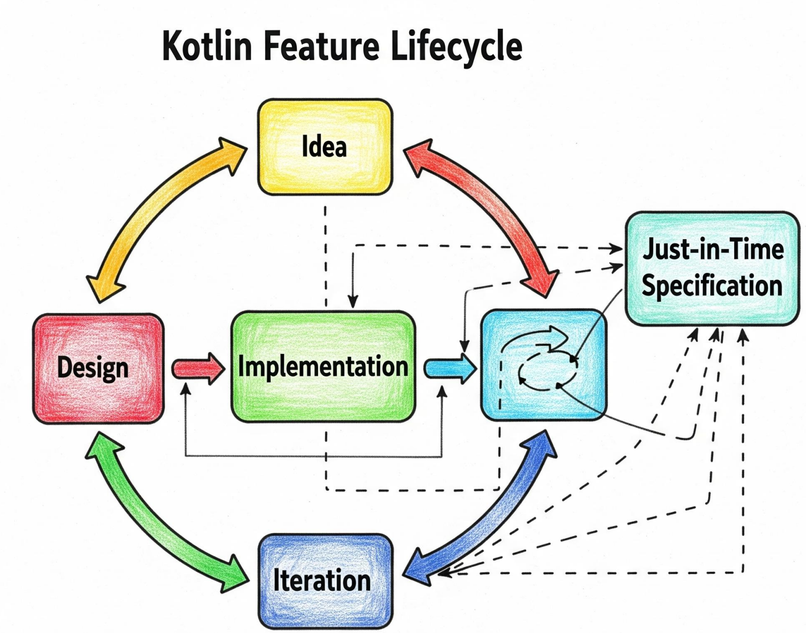
Thank you!

Marat Akhin



What are we going to talk about?

- The lifecycle of a Kotlin language feature
  - Initial idea
  - Design document
  - Iterative evolution
- Kotlin language specification
  - Just-in-time
  - Formal-just-enough
  - Partial (not total)



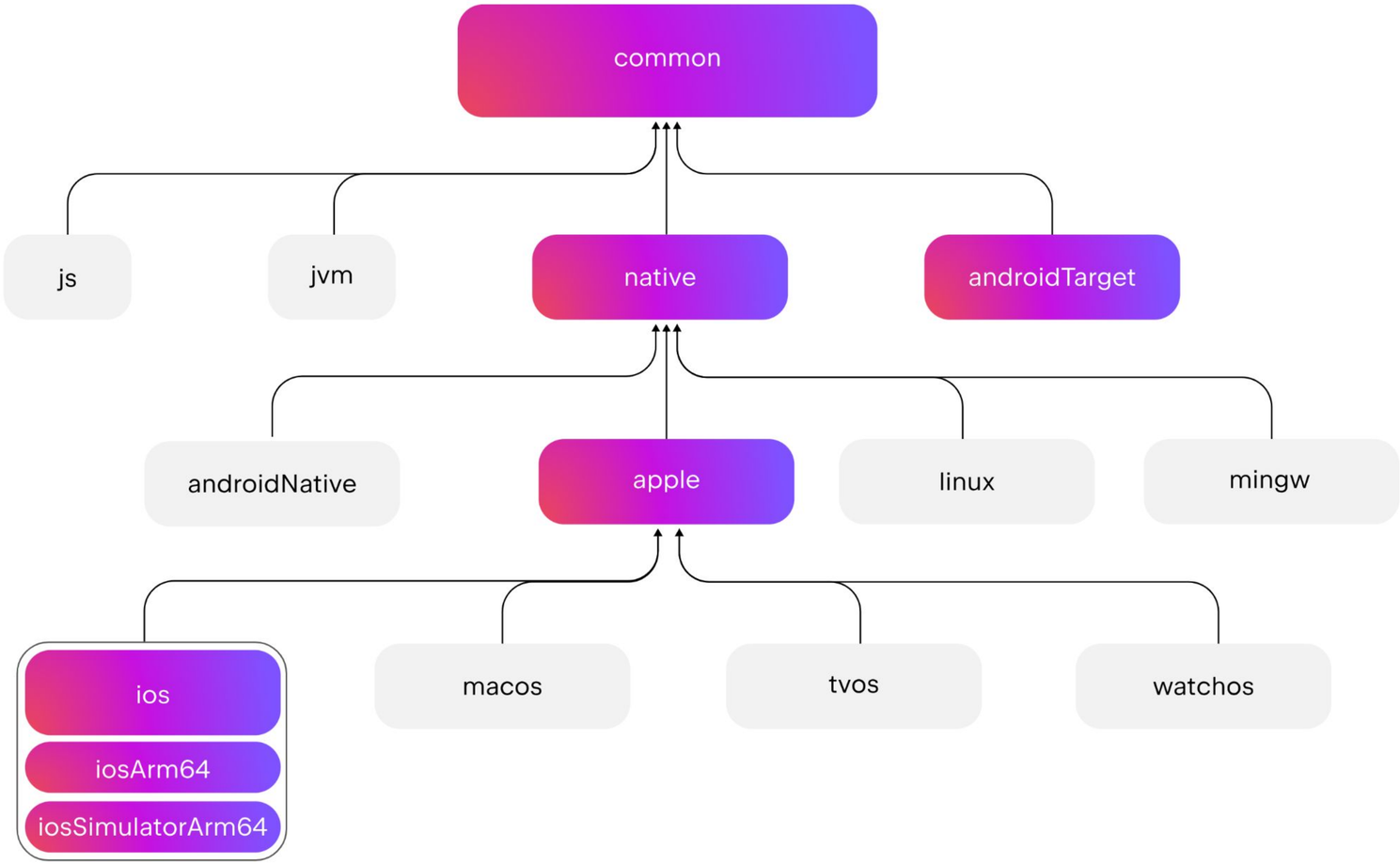
THE FOLLOWING TALK HAS BEEN
APPROVED TO ACCOMPANY

G KOTLIN
FEATURE DESIGN

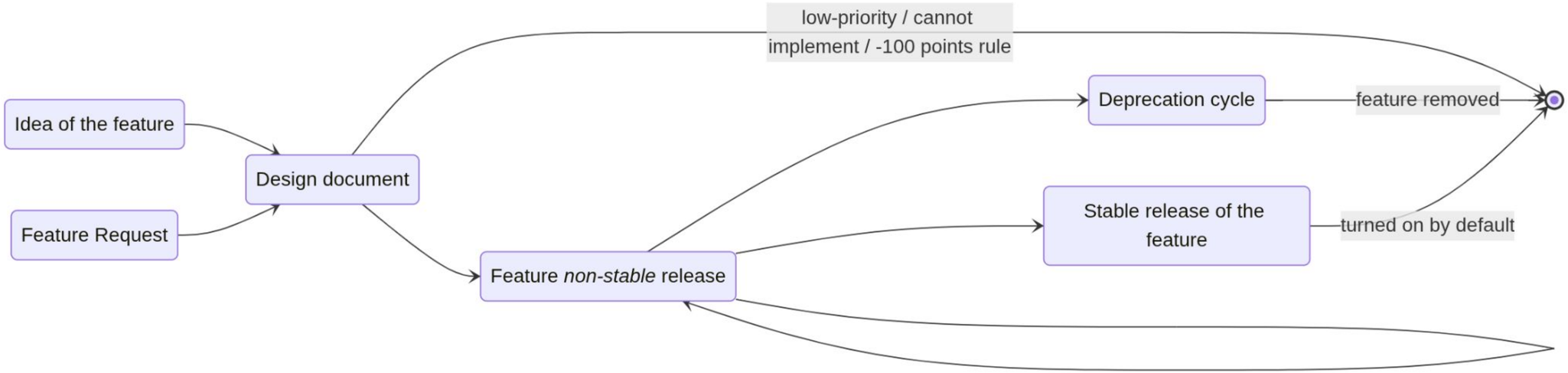
ALL AGES ADMITTED

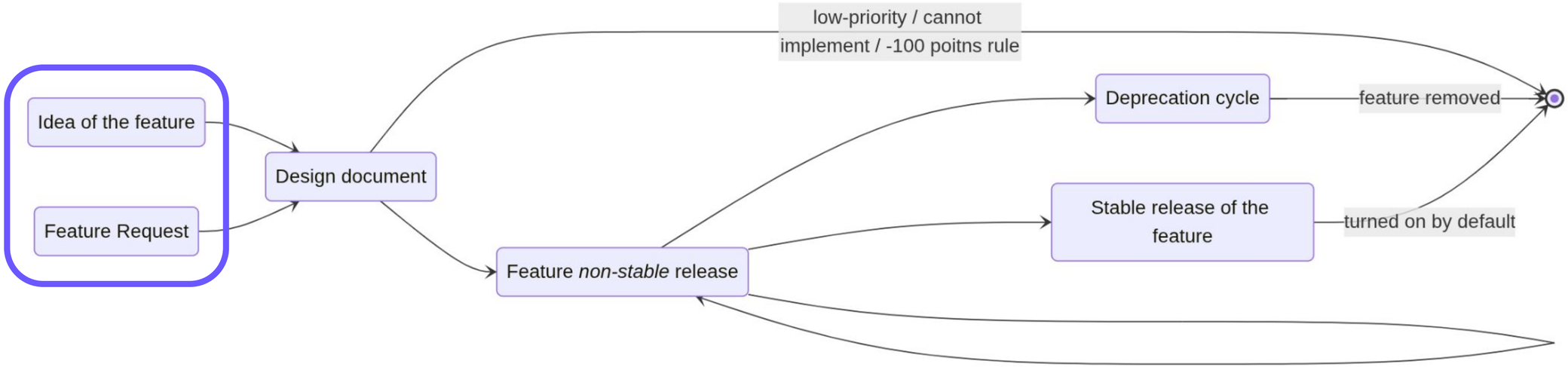
MPA

Kotlin in 30 seconds



Life of a feature


Idea of the feature
Feature Request
Design document
low-priority / cannot implement / -100 points rule
Feature *non-stable* release
Deprecation cycle
feature removed
Stable release of the feature
turned on by default


Idea of the feature
Feature Request
Design document
low-priority / cannot implement / -100 poitns rule
Feature non-stable release
Deprecation cycle
feature removed
Stable release of the feature
turned on by default

Feature “seed”

Feature request

- External
- Usually comes from the Kotlin users
  - EAP champions
  - Regular developers

An **“existing”** problem and its **possible** solution

Idea of a feature

- Internal
- Usually comes from the Kotlin team
  - Features from other languages
  - Results of SotA PL research

A **possible** problem and its **“existing”** solution

A good feature...

...is about an **existing** problem and its **existing** solution

- Sometimes problems are not actually problems

- Sometimes solutions are not actually feasible (for Kotlin)

A good feature...

...is about an **existing** problem and its **existing** solution

- Sometimes problems don’t have (good) solutions

- Sometimes solutions do not target actual problems (yet)

A problem exists…

…if there is **a lot of code** struggling with it

- We begin with **GitHub search**
  - Index what GitHub indexes
  - Quick and easy and good-enough initial approximation
- We continue with **custom Kotlin compiler builds**
  - Index a curated set of user projects
  - Can do non-trivial code analyses
- We used to have in-house code search tools
  - Indexes a configurable set of GitHub projects
  - Can do very complicated code analyses

Imprecise

Precise enough

Too precise

A problem exists…

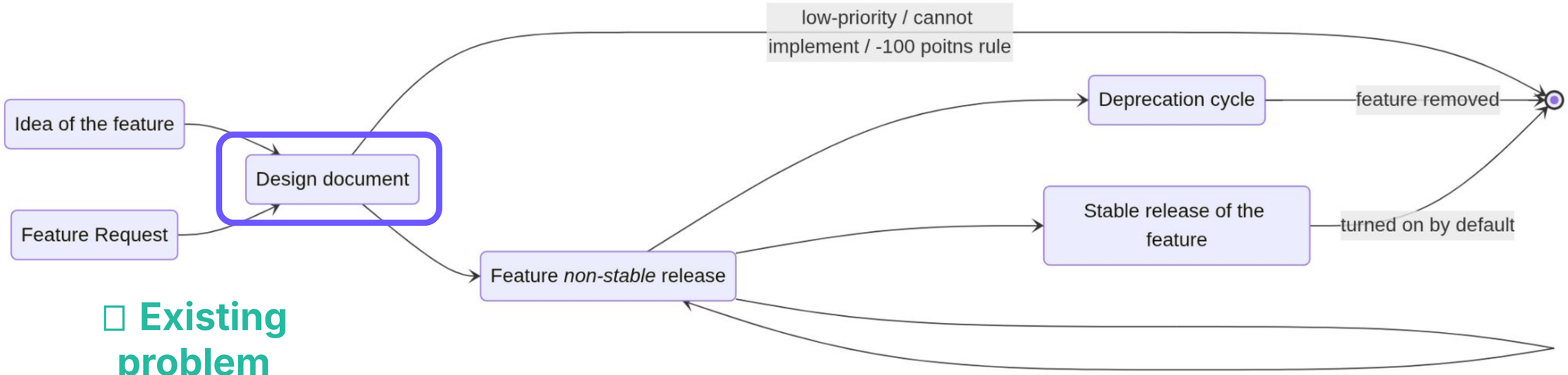
…if there is **a lot of code** struggling with it

- We begin with GitHub search
  - Indexes what GitHub indexes
  - Quick and easy and good-enough initial approximation
- We continue with custom Kotlin compiler builds
  - Indexes a curated set of user projects
  - Can do very complicated code analyses
- We **used to have** in-house code search tools (but not anymore)
  - Index a configurable set of GitHub projects
  - Can do very complicated code analyses

Imprecise

Precise enough

Too precise



□ Existing problem

Kotlin Evolution and Enhancement Process

- https://github.com/Kotlin/KEEP
  - Draft design notes
  - In-progress feature proposals and their discussions
  - An archive of accepted and rejected proposals
- Every KEEP must have a driver from the Kotlin team
  - It is hard to go in-depth enough
  - It is hard to not be too selfish
  - It is hard to write a good KEEP for free
- For people outside Kotlin we use other channels
  - Issue tracker (https://youtrack.jetbrains.com/issues/KT)
  - Kotlin language Slack (https://kotlinlang.slack.com/#language-design)

Kotlin Evolution and Enhancement Process

- https://github.com/Kotlin/KEEP
  - Draft design notes
  - In-progress feature proposals and their discussions
  - An archive of accepted and rejected proposals
- Every KEEP **must have** a **driver from the Kotlin team**
  - Or it is hard to go in-depth enough
  - Or it is hard to not be too selfish
  - Or it is hard to write a good KEEP for free
- For people outside Kotlin we use other channels
  - Issue tracker (https://youtrack.jetbrains.com/issues/KT)
  - Kotlin language Slack (https://kotlinlang.slack.com/#language-design)

Kotlin Evolution and Enhancement Process

- https://github.com/Kotlin/KEEP
  - Draft design notes
  - In-progress feature proposals and their discussions
  - An archive of accepted and rejected proposals
- Every KEEP must have a driver from the Kotlin team
  - It is hard to go in-depth enough
  - It is hard to not be too selfish
  - It is hard to write a good KEEP for free
- For people outside Kotlin we use other channels
  - Issue tracker (https://youtrack.jetbrains.com/issues/KT)
  - Kotlin language Slack (https://kotlinlang.slack.com → #language-design)

The anatomy of a KEEP

Outline

Filter headings

The anatomy of a KEEP

- **Motivation** behind the feature
- **Use-cases** that the feature is intended to cover

Problem

- Feature design from the **developer experience** perspective
- In-depth design from the **implementation** perspective

Solution

- **Problems** and **risks** that the feature may bring
- **Open questions** for the discussion

Risks

Is it actually a problem?

Five “Whys” rule

- Why the users think it is a problem?
- Why the users want to write code differently?
- Why are existing solutions not enough any more?
- ...
- Do a lot of users care about the problem?
- Is there no easy workaround?
- Is it a general problem not specific to a narrow domain?

Is it actually a problem?

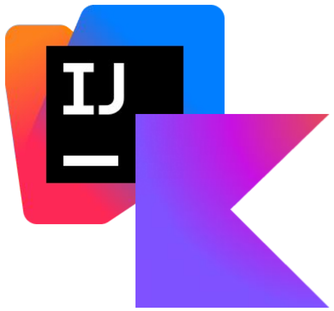
Minus 100 points rule

- Why the users think it is a problem?
- Why the users want to write code differently?
- Why are existing solutions not enough any more?
- ...
- Do a lot of users care about the problem?
- Is there no easy workaround?
- Is it a general problem not specific to a narrow domain?

Is it a Kotlin enough problem?

Minus 100 points rule

- Can we not implement the solution as a part of Kotlin standard library / kotlinx.*?
- Cannot it be a Kotlin compiler plugin?
- Cannot it be solved via IDE support?



Is there a solution?

- What do we want to add to the language?
  - Is it known in other languages?
  - Is it known in PL research?
- Can we implement what we want?
  - Is it feasible with the current compiler architecture?

- *Does it help **reading the code**?*

- *Is the mental model **easy to explain**?*

Is there a solution?

Minus 100 points rule

- What do we want to add to the language?
  - Is it known in other languages?
  - Is it known in PL research?
- Can we implement what we want?
  - Is it feasible with the current compiler architecture?

- *Does it help **reading the code**?*

- *Is the mental model **easy to explain**?*

What are the risks?

Minus 100 points rule

- Does the feature preserve backwards compatibility?
  - Binary compatibility
  - Source compatibility
- How does the new feature interact with existing features?
  - Does it create new puzzlers?
  - Does it change the meaning of existing code?

- *How do we teach and migrate the users to the new feature?*

What are the risks?

Minus 100 points rule

- Does the feature preserve backwards compatibility?
  - Binary compatibility
  - Source compatibility
- How does the new feature interact with existing features?
  - Does it create new puzzlers?
  - Does it change the meaning of existing code?

- *How do we **teach** and **migrate** the users to the new feature?*

External vs internal

- Motivation behind the feature
- Use-cases that the feature is intended to cover

Does it change the **meaning of existing code**?

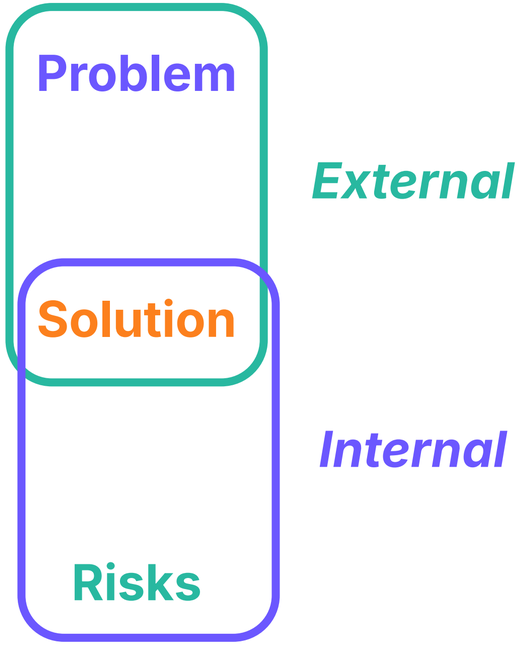
leveloper experience perspective

esign from the impleme erspective

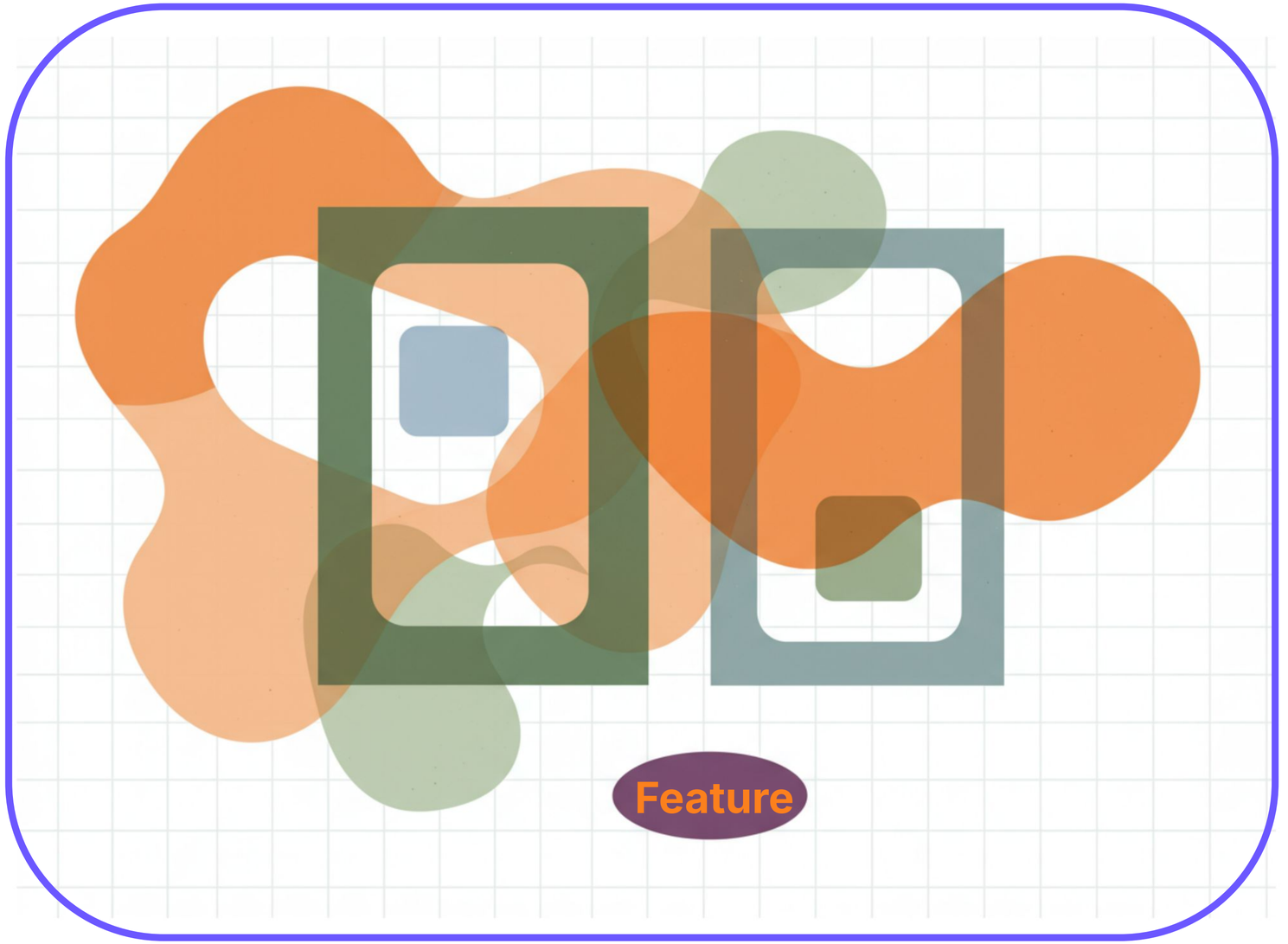
Does the feature preserve **backwards compatibility**?

- Problems and risks that the feature may bring
- Open questions for the discussion

Is the mental model ***easy to explain****?*



It’s all about dependencies



Internal

It’s all about ~~dependencies~~ the language

- The **common language** to standardize language features
  - One could talk about “function default arguments”
  - Another could argue about “optional parameters”
  - But they are actually discussing exactly the same thing

If you standardize enough language features and their dependencies, you get a **language specification**

Just-in-time specification

Kotlin language specification

- https://kotlinlang.org/spec/
- 310 pages of formal-just-enough specification of Kotlin/Core
  - Platform-agnostic parts of Kotlin language
  - Covers the most important parts of the syntax and semantics
  - Specifies things which should work the same way on all platforms*
    - Type system
    - Overload resolution
    - Type inference
    - ...

**terms and conditions apply*

A little bit of history

Specification-as-technical-doc

Specification rework

2010	2016	2018	2020	2024
Kotlin created	**Kotlin 1.0 released**	**Kotlin 1.3 released** Re-implementation of the type system aka New Inference Re-implementation of the frontend aka K2	**Kotlin 1.4 released** ✅ **Re-implementation of the type system aka New Inference**	**Kotlin 2.0 released** ✅ **Re-implementation of the frontend aka K2**

Language Specification

- Informal workings of the language
- Language reference

...

- **Language specification**

...

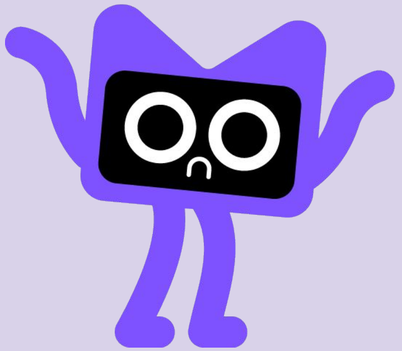
- Formal / executable language specification

...

- Compiler implementation

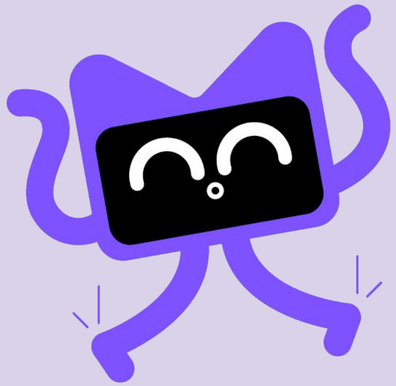
When do you need a formal specification?

- When you have **more than one compiler** for your language
  - They need to agree on how things should work
  - They need to do this independently from each other
  - They need more than just a common language, they need a common **executable behavior**
- When the **cost** of a breaking change **is too high**
  - Mission-critical systems
  - Safety-critical systems



When do you need a specification?

- When you need to **talk a lot** about language features, syntax and semantics
  - You are re-implementing large parts of the compiler
  - You need to support (new) tooling for the language
- When your language is **complicated**
  - It has a very “smart” compiler
  - It does a lot of things implicitly



When do you write a specification?

Before you write the implementation, duh!

- “Ahead-of-time” specification when everything is defined first and implemented second

- Does not really work
  - ☑️ If you already have an implementation
  - ☑️ If you have to “catch up” to your dependencies fast
  - ☑️ If backwards compatibility is important

Just-in-time specification

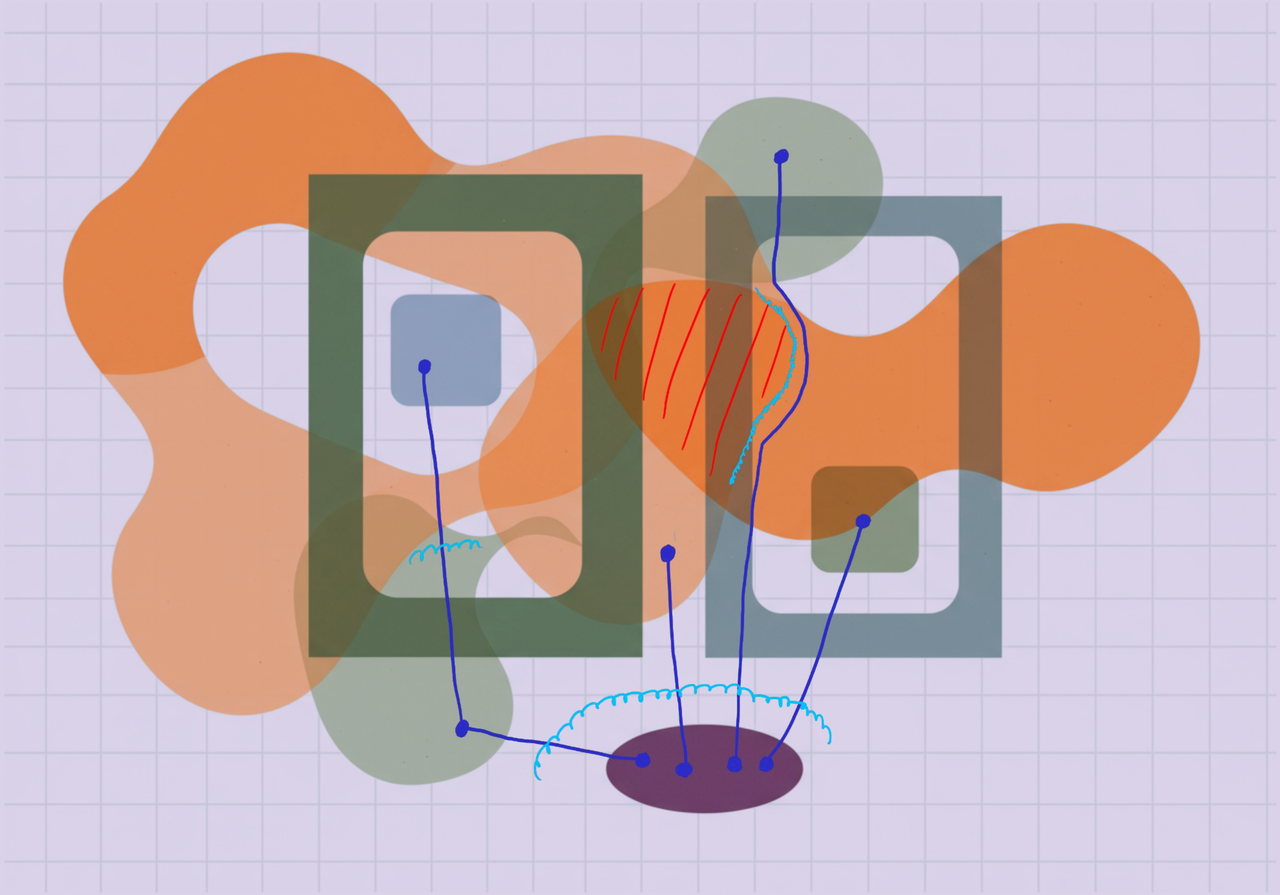
Specify only the parts needed to talk about the new feature

or

Specify the parts needed to compare the old and the new implementation

or

<insert your reason for procrastination here>



Just-in-time specification

- It is pragmatic
  - You do not specify things which you never talk about
  - Or which you never implement
- It is smaller
  - People can reasonably fit it into their heads
  - The focus is on the core behavior
- It is faster / cheaper to write
  - Because it's pragmatic and smaller
  - And you can always skip writing it

Extended resolution algorithm

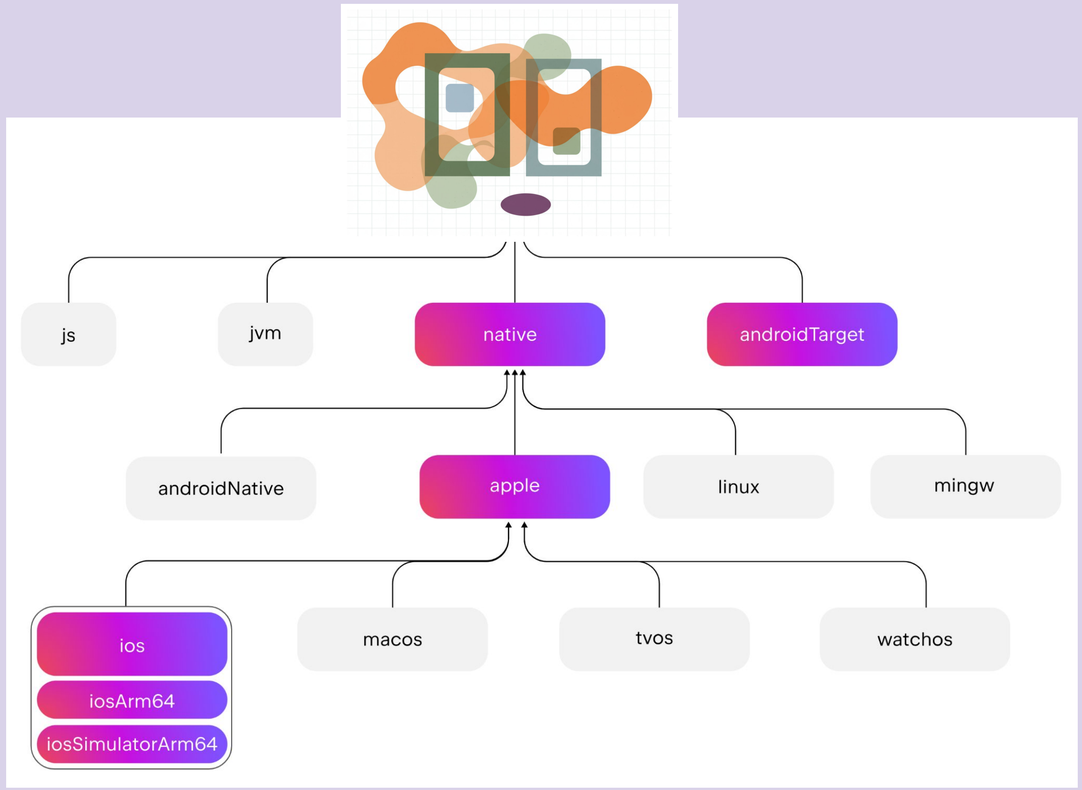
§7.3 *(declaration with context parameters)*: The context parameters declared for a callable are available in the same way as "regular" value parameters in the body of the function. Both value and context parameters are introduced in the same scope, there is no shadowing between them (remember that parameters names must be unique across both value and context parameters),

§7.4 *(applicability, lambdas)*: Building the constraint system is modified for lambda arguments. Compared with the Kotlin specification, the type of the parameter U_m is replaced with *nocontext*(U_m), where *nocontext* removes the initial `context` block from the function type.

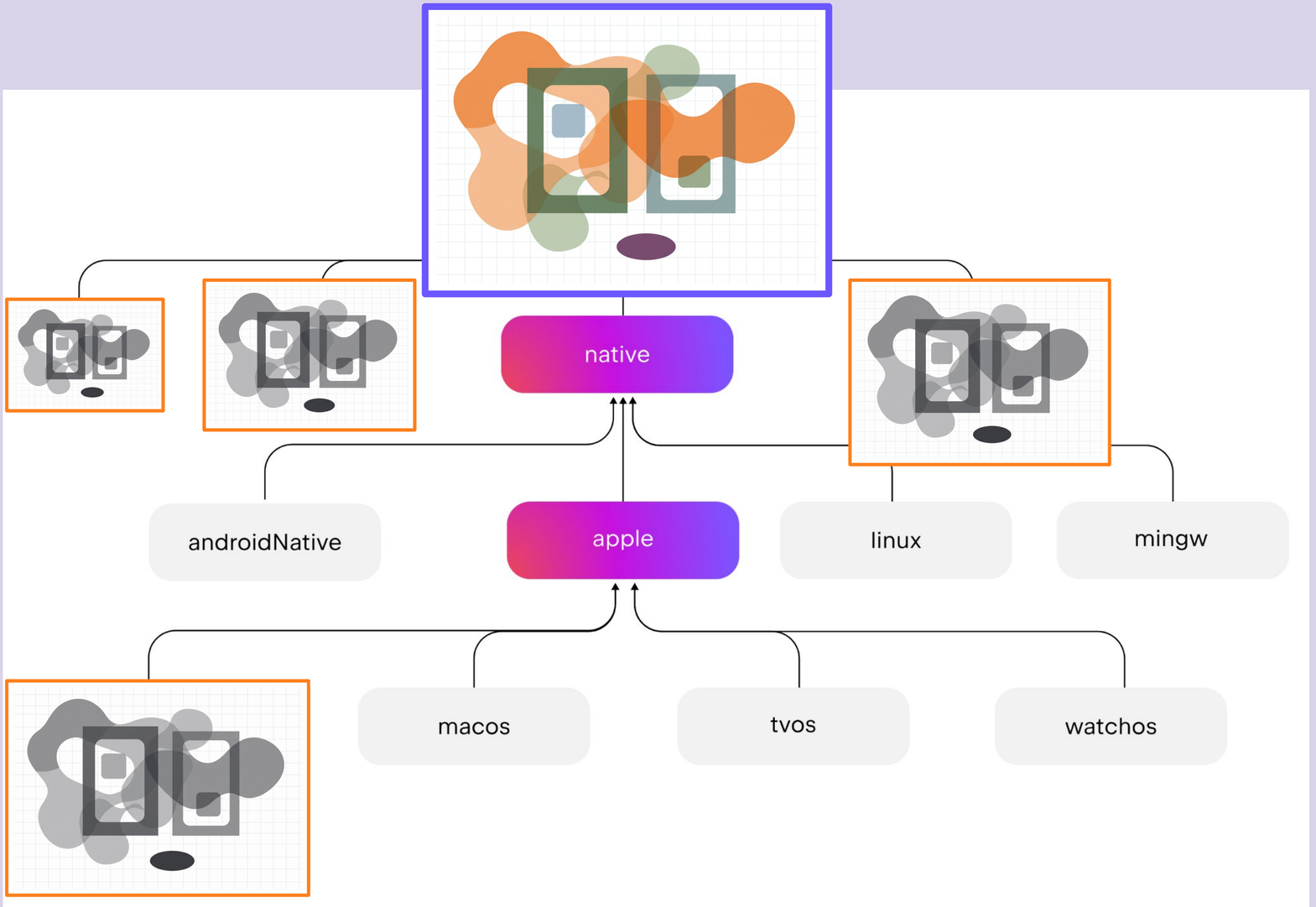
§7.5 *(applicability, context resolution)*: After the first phase of function applicability -- checking the type constraint problem -- an additional **context resolution** phase is inserted. For each potentially applicable callable, for each context parameter, we traverse the tower of scopes looking for **exactly one** default receiver or context parameter with a compatible type.

Problems with JIT specification

- It requires discipline
  - One needs to say they need X to be specified
  - Otherwise X will never be specified
- It misses things
  - You can forget to update
  - You can forget to synchronize
- It is not enough
  - When you have many external dependencies (e.g., target platforms)
  - When you are doing something completely new



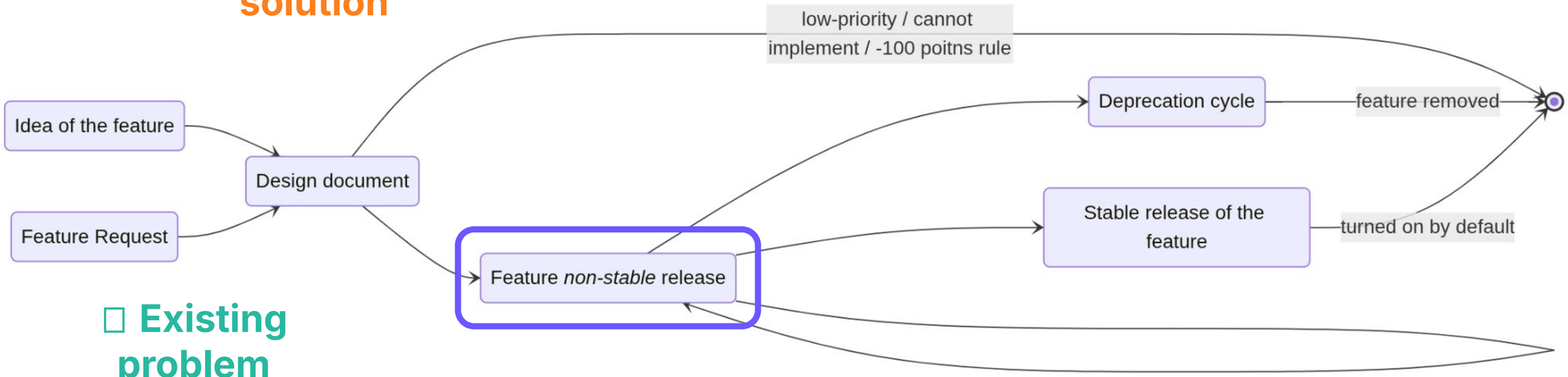
Going forward



Life of a feature

 JIT specification

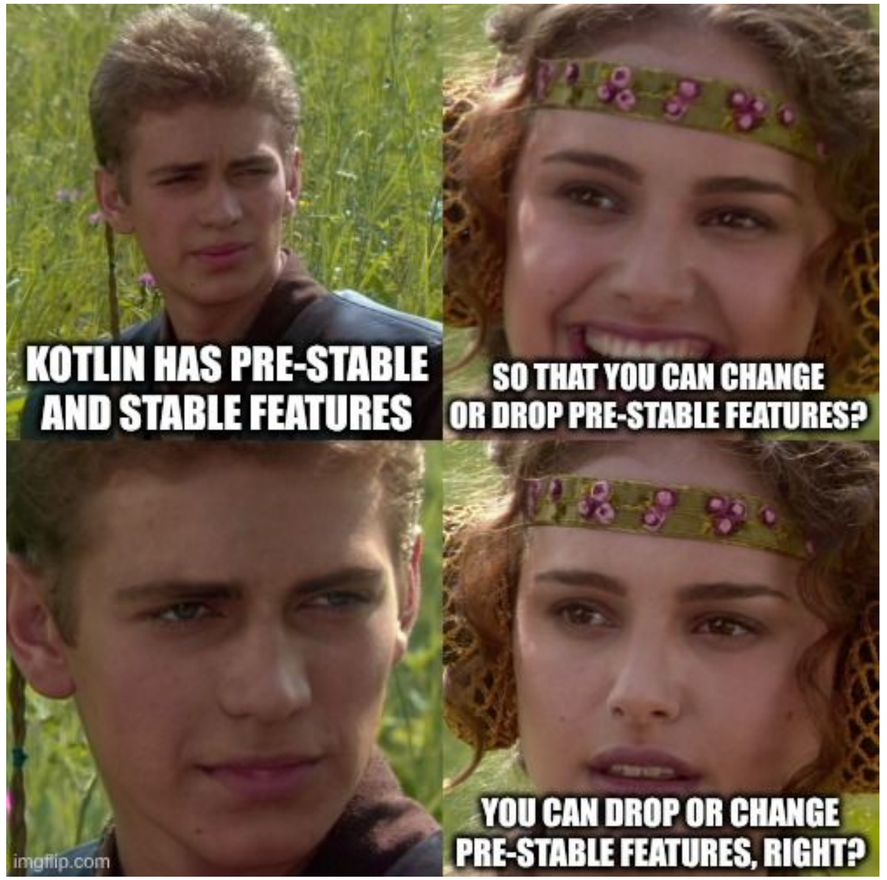
 Existing solution



 Existing problem

Feature kinds

- **Experimental** means "try it only in toy projects"
- **Alpha** means "use at your own risk, expect migration issues"
- **Beta** means "you can use it, we'll do our best to minimize migration issues for you"
- **Stable** means "use it even in most conservative scenarios"



Hyrum’s law

- If there is a Kotlin feature under two compiler flags and three hidden internal-only annotations, people will find it and use it
- And they will depend on it in their code

Only after Kotlin 2.0 we really began changing
or dropping pre-stable features

When to drop a feature?

Is it a good feature?

- The feature should be useful to us
  - Battle-test it in the Kotlin compiler / kotlinx.* libraries / Ktor / …
  - Try it in the IntelliJ code base
- The feature should be useful to our EAP champions
- The feature should be useful to our users

People write less code

People write easier to read code

People write less error prone code

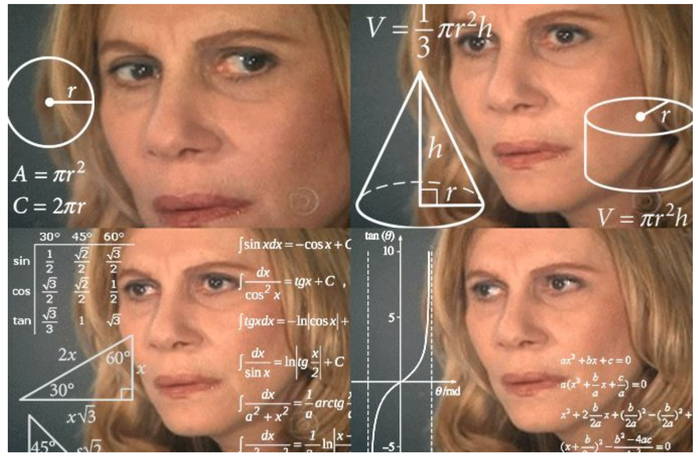
Is it a bad feature?

- The feature creates (unexpected) problems
- The feature breaks other features
- The feature does not solve the original problem

People have to think about new corner cases

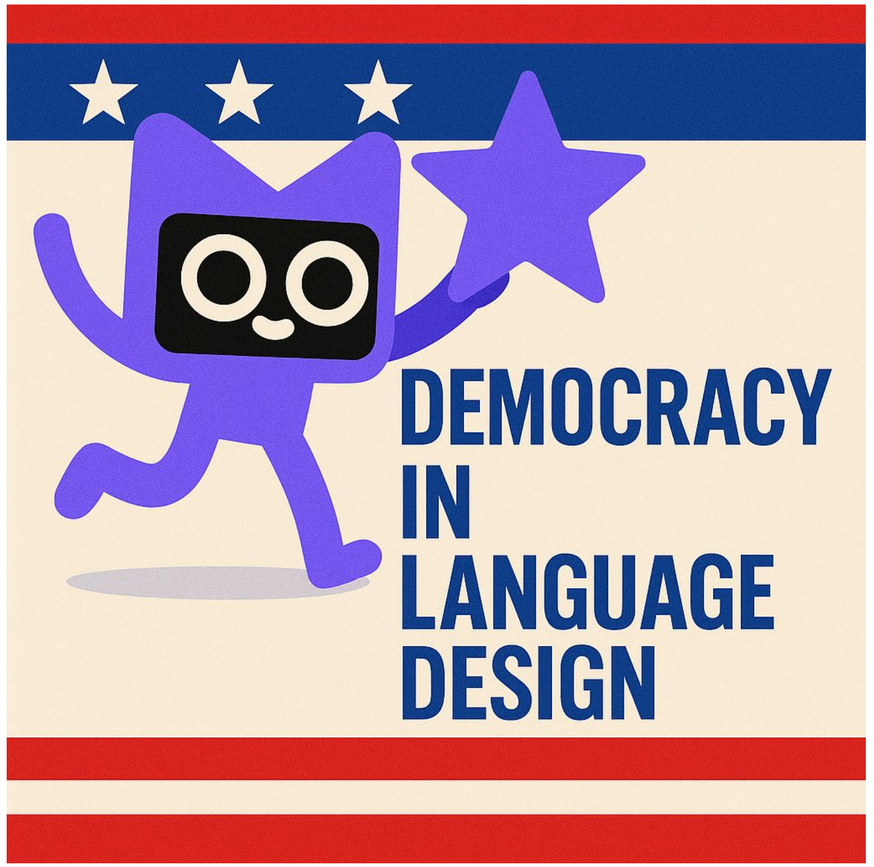
People write unintuitive code

People are not using the feature (enough)



What happens if there’s a tie?

Aka there are several ~~bad~~ good feature designs at any point in the process

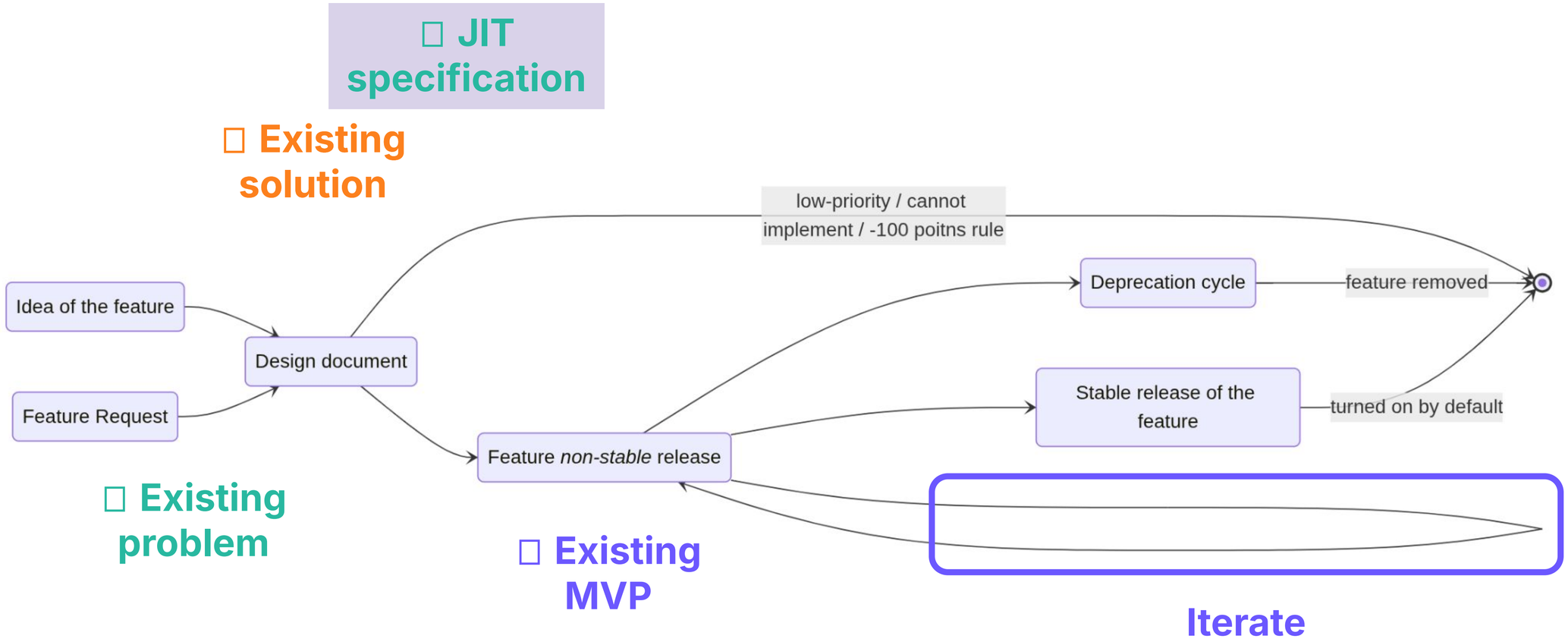


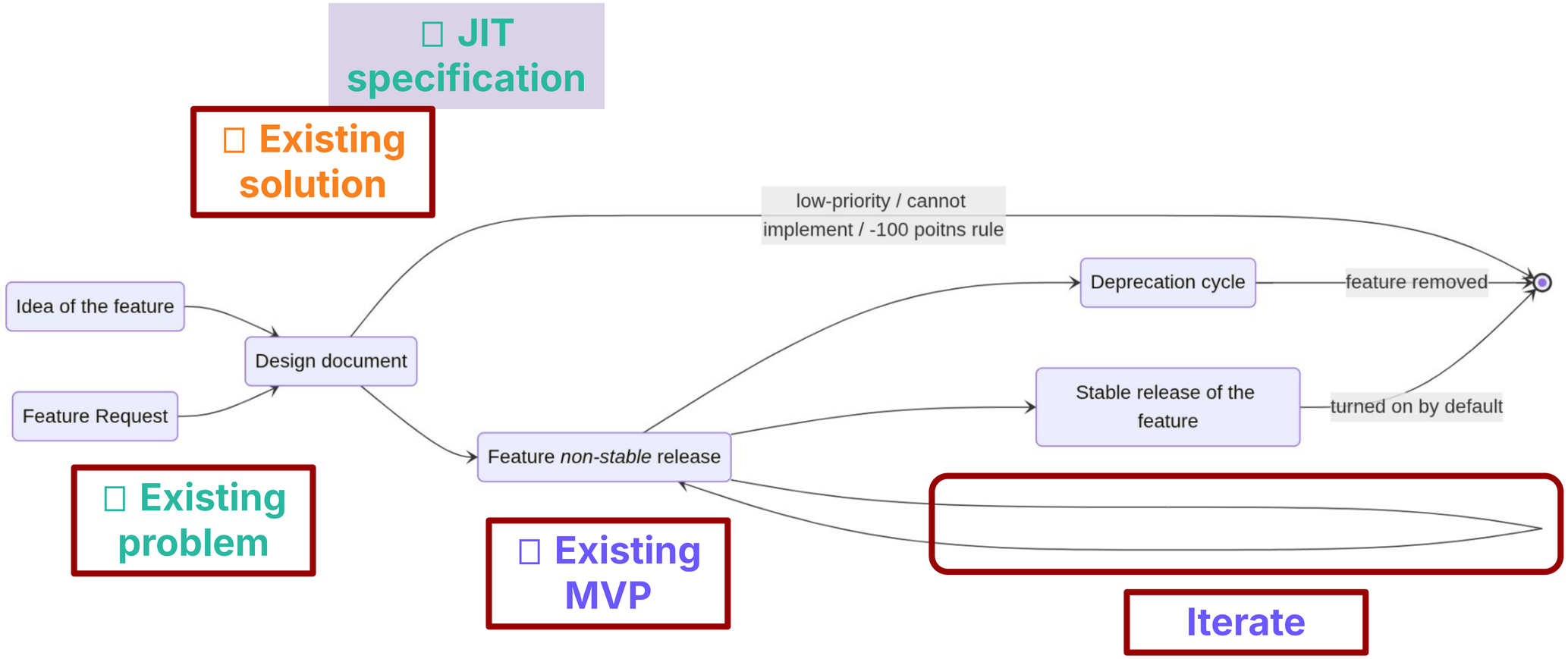
What happens if there’s a tie?

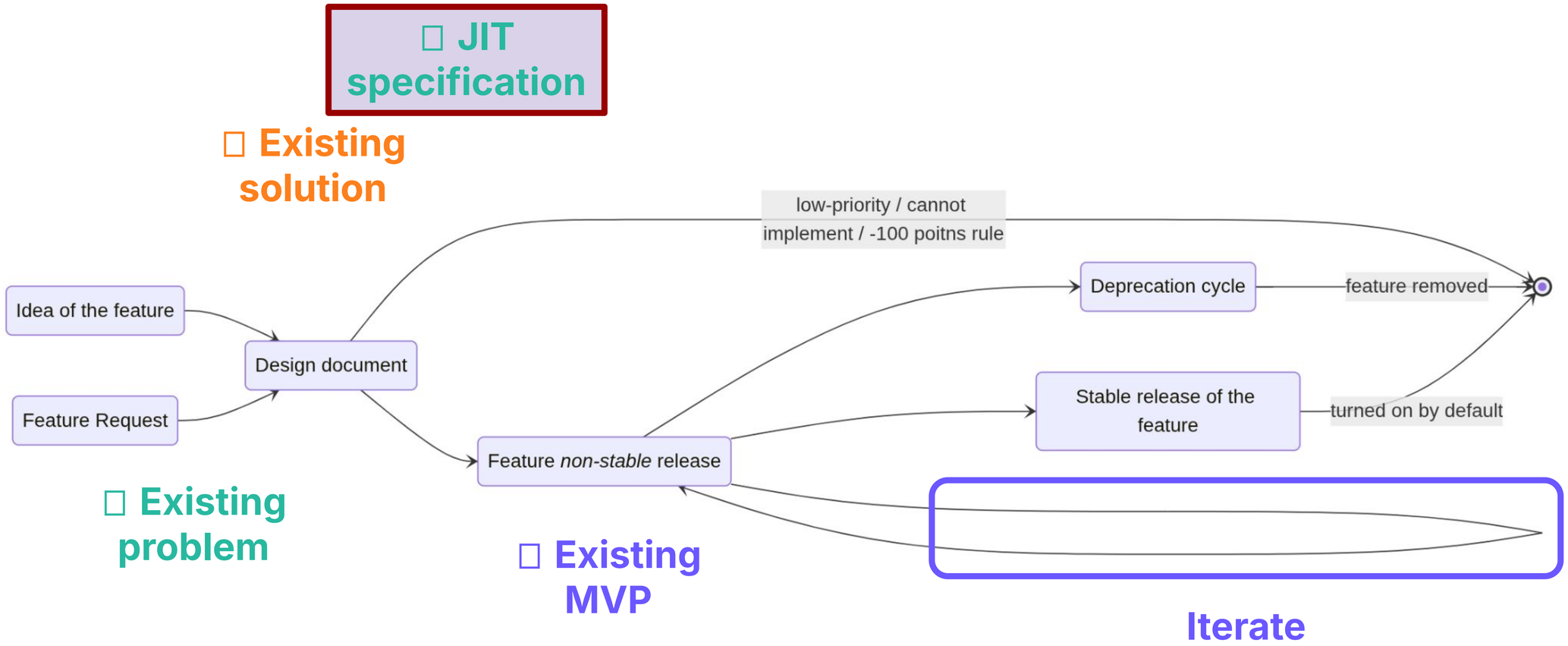
We have a lead language designer for a reason!

- We didn’t find a good framework to objectively compare different feature designs
- Democracy had the same success rate as “just pick one and move forward”

If there is a tie, **Mikhail has the final say**

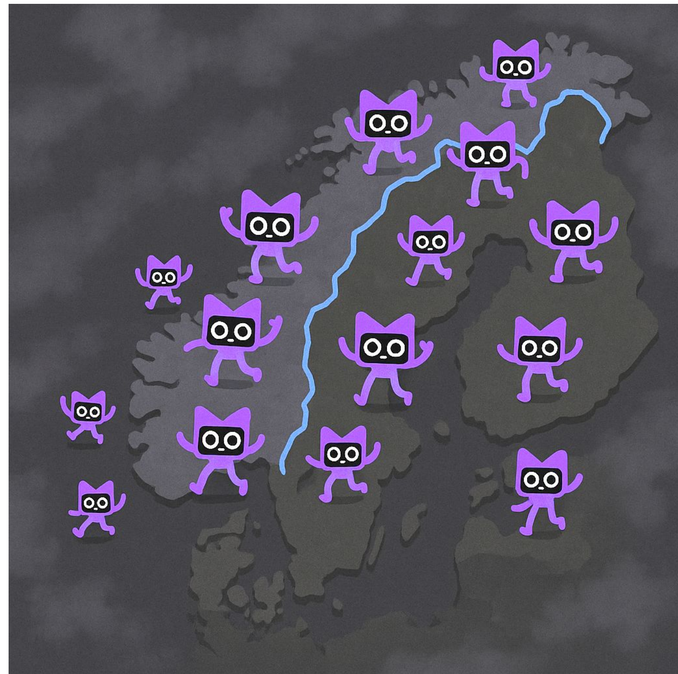

JIT specification
Existing solution
low-priority / cannot implement / -100 poitns rule
Idea of the feature
Design document
Feature Request
Deprecation cycle
feature removed
Stable release of the feature
turned on by default
Feature non-stable release
Existing problem
Existing MVP
Iterate


JIT specification
Existing solution
low-priority / cannot implement / -100 poitns rule
Idea of the feature
Design document
Feature Request
Deprecation cycle
feature removed
Stable release of the feature
turned on by default
Feature non-stable release
Existing problem
Existing MVP
Iterate


JIT specification
Existing solution
low-priority / cannot implement / -100 poitns rule
Idea of the feature
Design document
Feature Request
Deprecation cycle
feature removed
Stable release of the feature
turned on by default
Feature non-stable release
Existing problem
Existing MVP
Iterate

Let’s share problems!

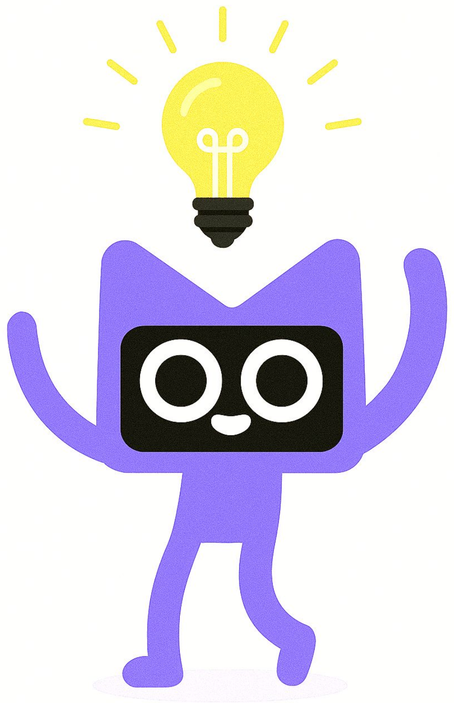
- What are the problems you consider “the next big thing” in PL design?
  - Concurrency and safety (data race freedom?)
  - Resource management (linear types?)
  - Better immutability (mutable value semantics?)
- How do you collect the user pain points?
  - To choose what to work on next
  - To choose what to drop from the language



Let’s share solutions!

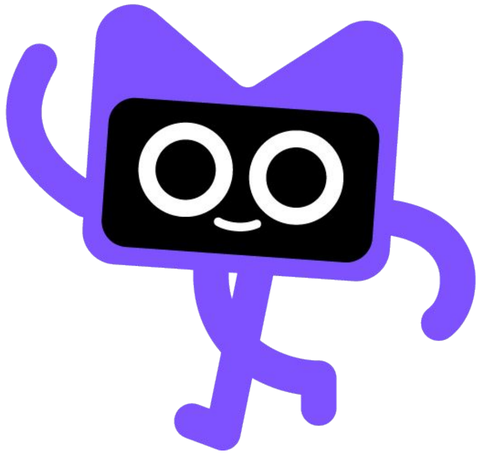
Let’s collaborate!

- What are your sources of inspiration?
  - POPL? PLDI? **ECOOP**?
  - **PLSS**?
- Do you “peek” at what other languages do?
  - How do you check if the features “fit”?
  - How do you talk about features **between** languages?



Kotlin
ahead-of-time
specification?

Let's share problems?



Let's share solutions?

Would you like to talk more?

marat.akhin@jetbrains.com
mikhail.zarechenskiy@jetbrains.com