One Source to Rule Them All

Kotlin DSLs as a Single Source of Truth for Multiple Tasks

Ivan Ponomarev

Иван Пономарёв

- Java и Kotlin разработчик
- Преподаю в МФТИ

В этом докладе будут

В этом докладе будут

- Рассуждения о сильных и слабых сторонах Kotlin DSL по сравнению с другими вариантами создания DSL-ей

В этом докладе будут

- Рассуждения о сильных и слабых сторонах Kotlin DSL по сравнению с другими вариантами создания DSL-ей
- Несколько примеров задач, решаемых с помощью Kotlin DSL

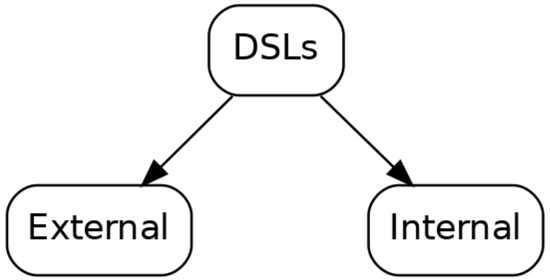
В этом докладе будут

- Рассуждения о сильных и слабых сторонах Kotlin DSL по сравнению с другими вариантами создания DSL-ей
- Несколько примеров задач, решаемых с помощью Kotlin DSL
- Некоторые продвинутые частности проектирования Kotlin DSL

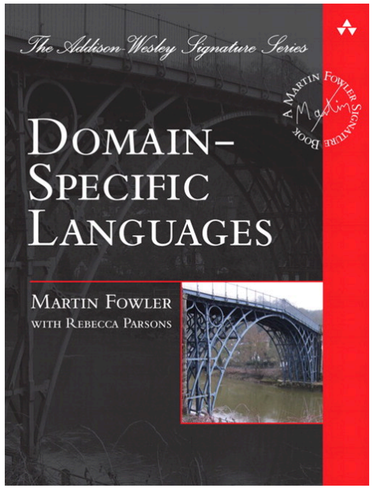
В этом докладе не будет

- Введения в создание Kotlin DSL (но будет много ссылок на соответствующие материалы)

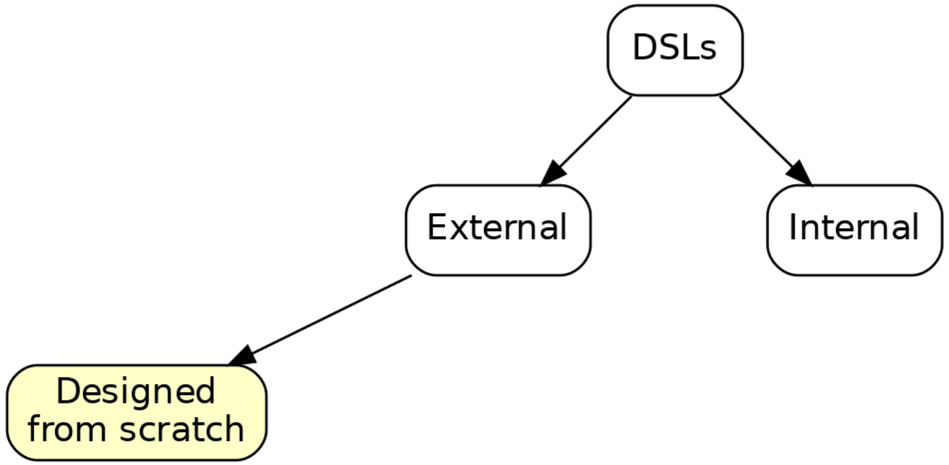
DSL внешние и внутренние



Martin Fowler, Rebecca Parsons
Domain-Specific Languages

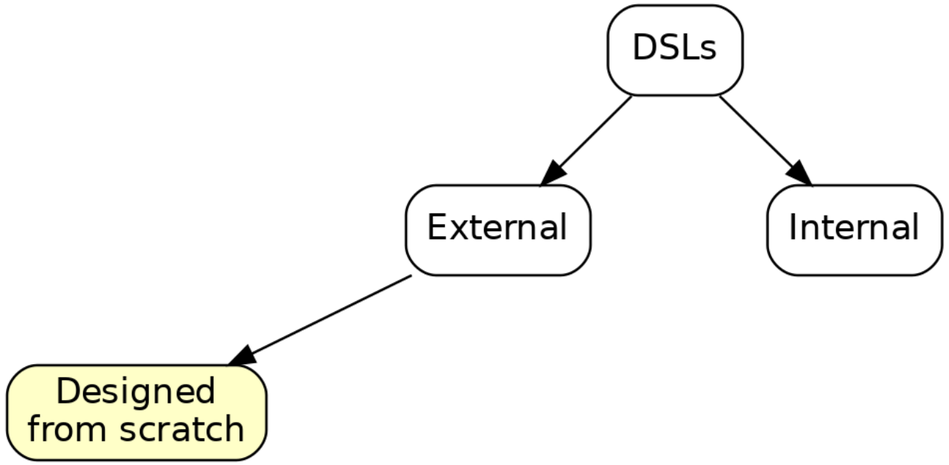


Разновидности внешних DSL



Разработанные с нуля

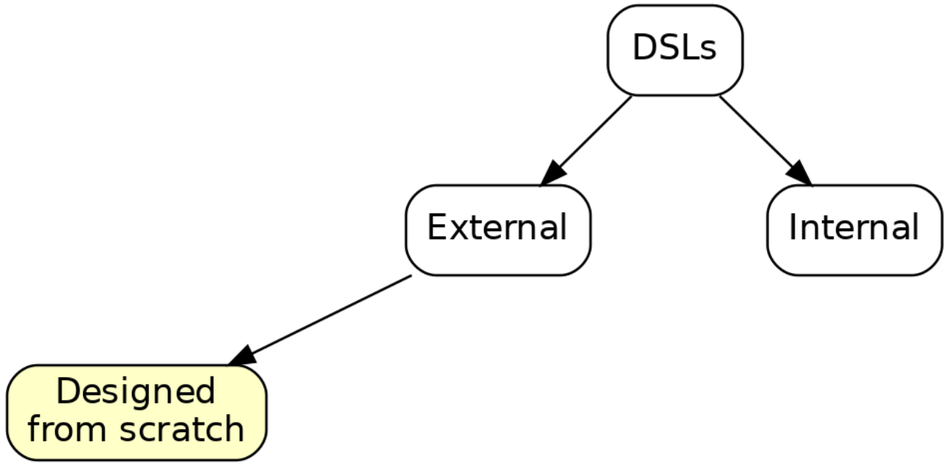
Разновидности внешних DSL



Разработанные с нуля

- DOT (GraphViz), PlantUML, gnuplot...

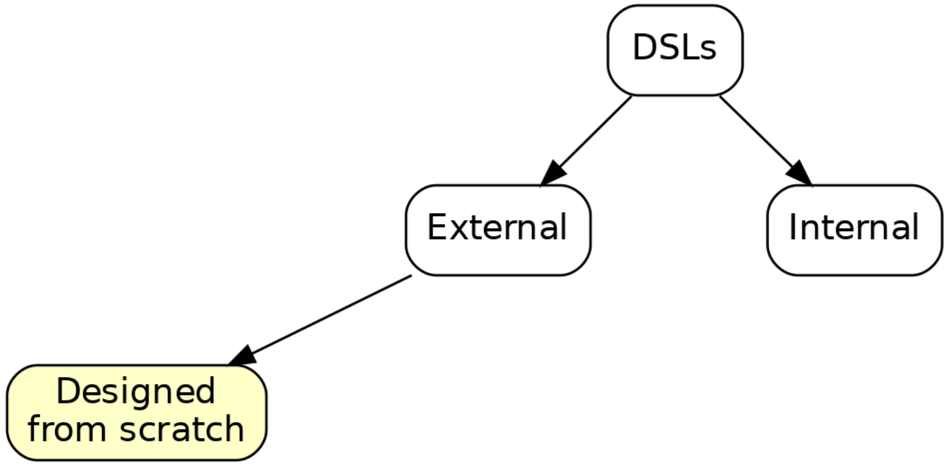
Разновидности внешних DSL



Разработанные с нуля

- DOT (GraphViz), PlantUML, gnuplot...
- HCL (Hashicorp Configuration Language)

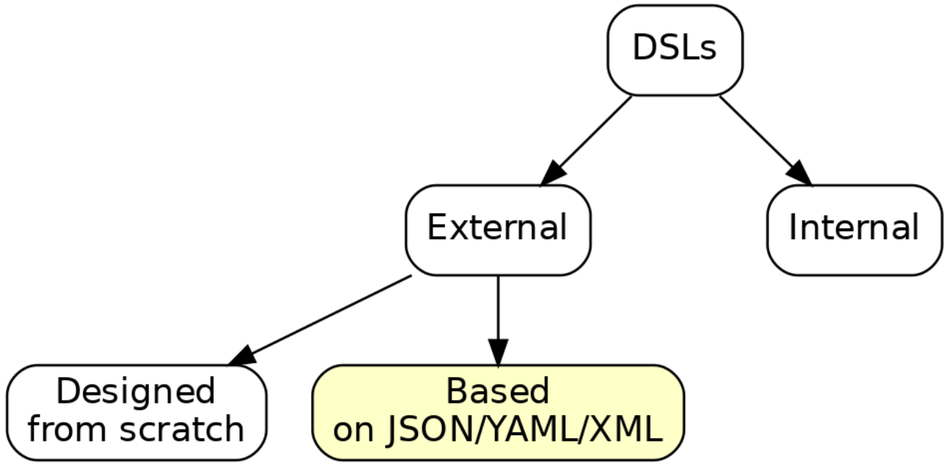
Разновидности внешних DSL



Разработанные с нуля

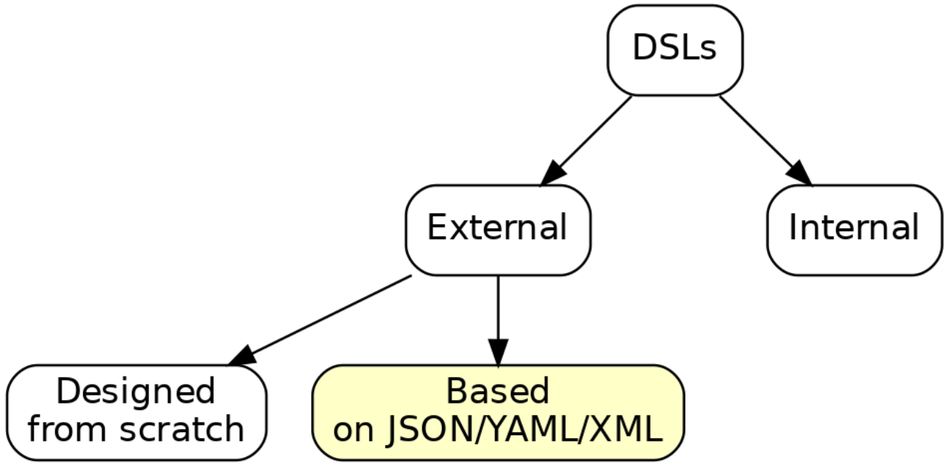
- DOT (GraphViz), PlantUML, gnuplot...
- HCL (Hashicorp Configuration Language)
- Gherkin (Cucumber framework’s language)

Разновидности внешних DSL



На основе языков разметки

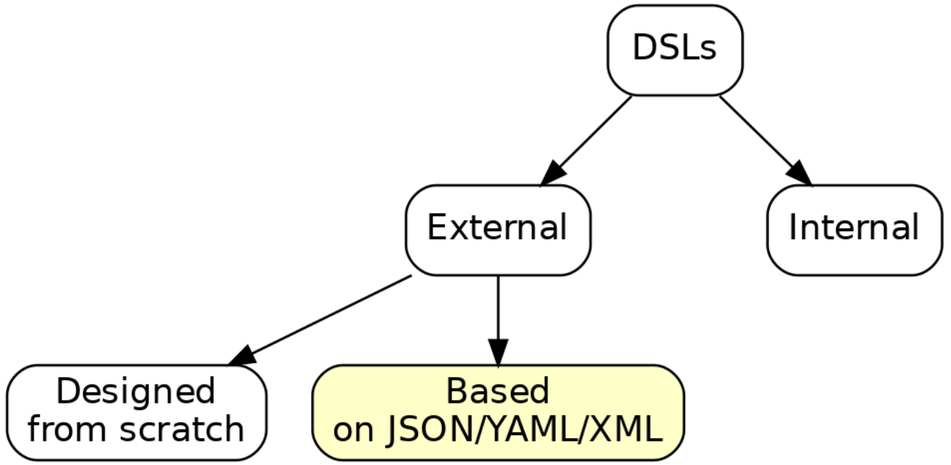
Разновидности внешних DSL



На основе языков разметки

- **YAML**: OpenAPI, Ansible, GithubActions, k8s definitions...

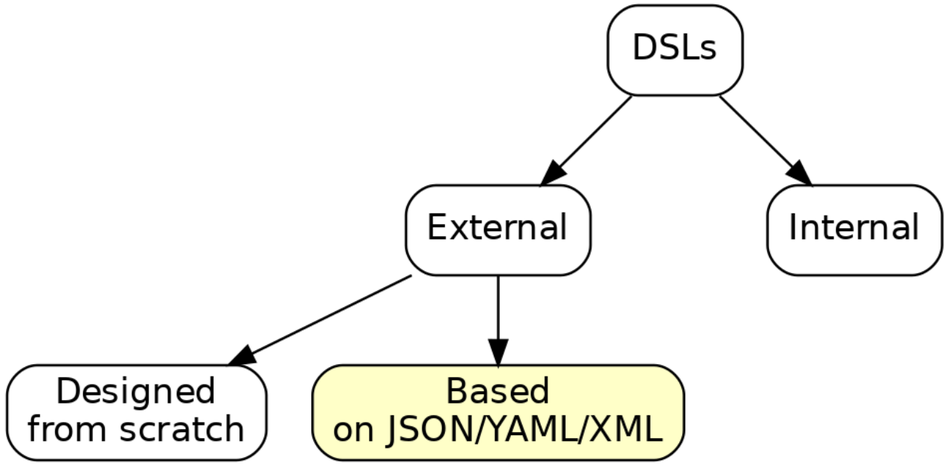
Разновидности внешних DSL



На основе языков разметки

- **YAML**: OpenAPI, Ansible, GithubActions, k8s definitions...
- **JSON**: Vue i18n files

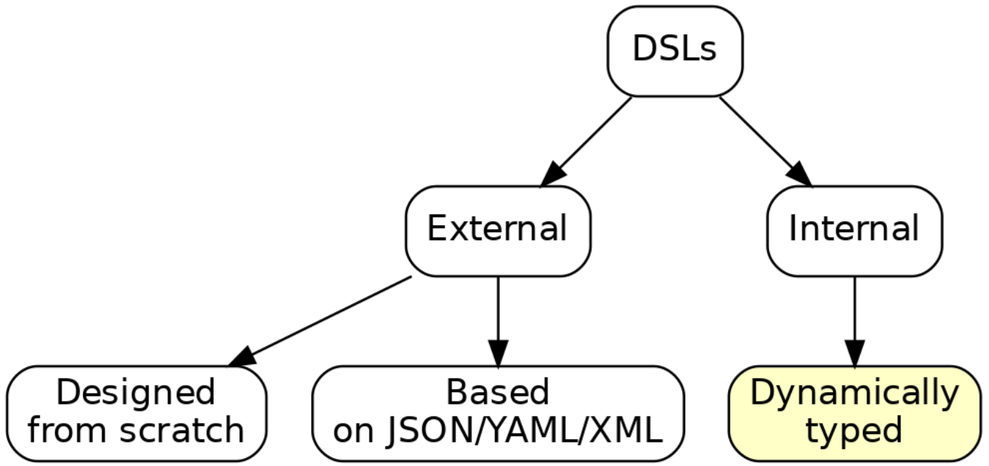
Разновидности внешних DSL



На основе языков разметки

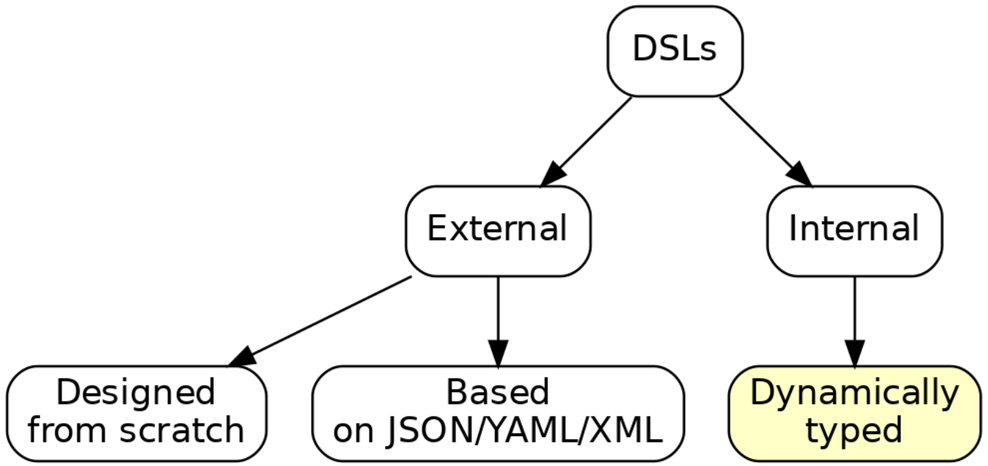
- **YAML**: OpenAPI, Ansible, GithubActions, k8s definitions...
- **JSON**: Vue i18n files
- **XML**: XSLT, XSD

Внутренние DSL



Подмножество динамически типизированного языка

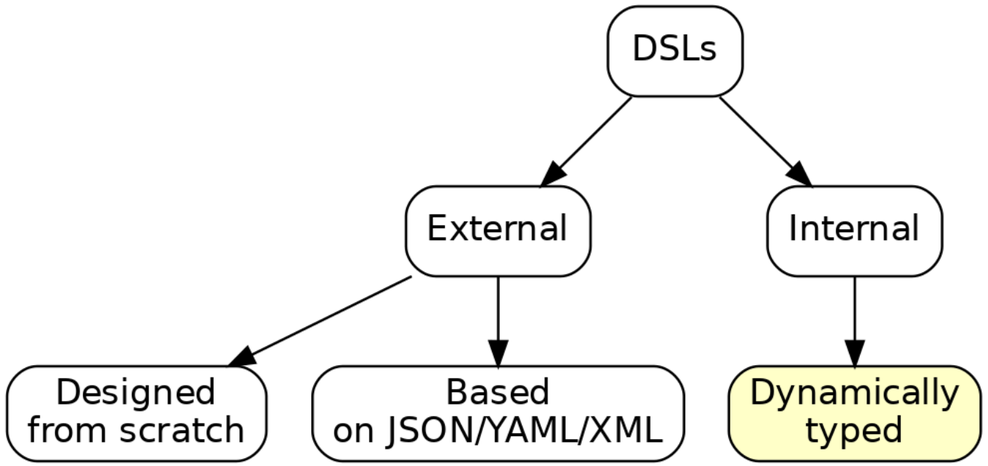
Внутренние DSL



Подмножество динамически типизированного языка

- **Lisp** (historically first): Emacs Lisp, Symbolic Mathematics etc.

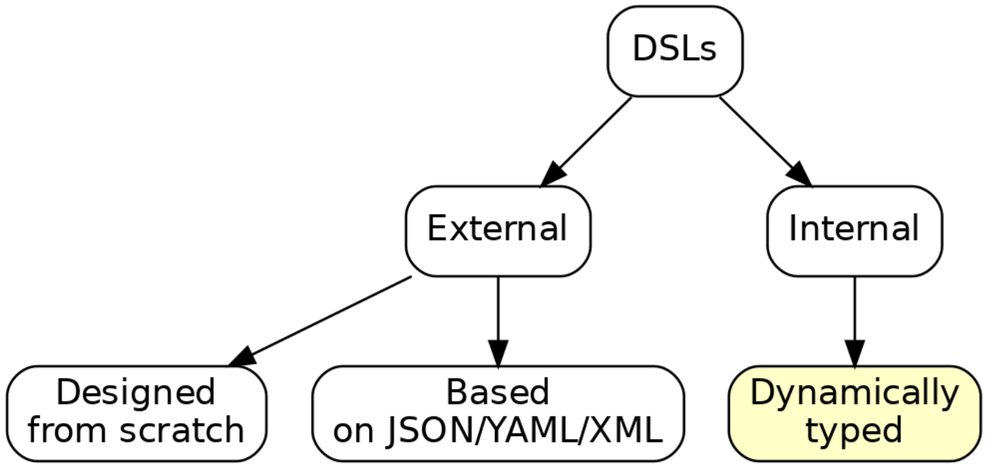
Внутренние DSL



Подмножество динамически типизированного языка

- **Lisp** (historically first): Emacs Lisp, Symbolic Mathematics etc.
- **Ruby**: Rails, RSpec, Chef...

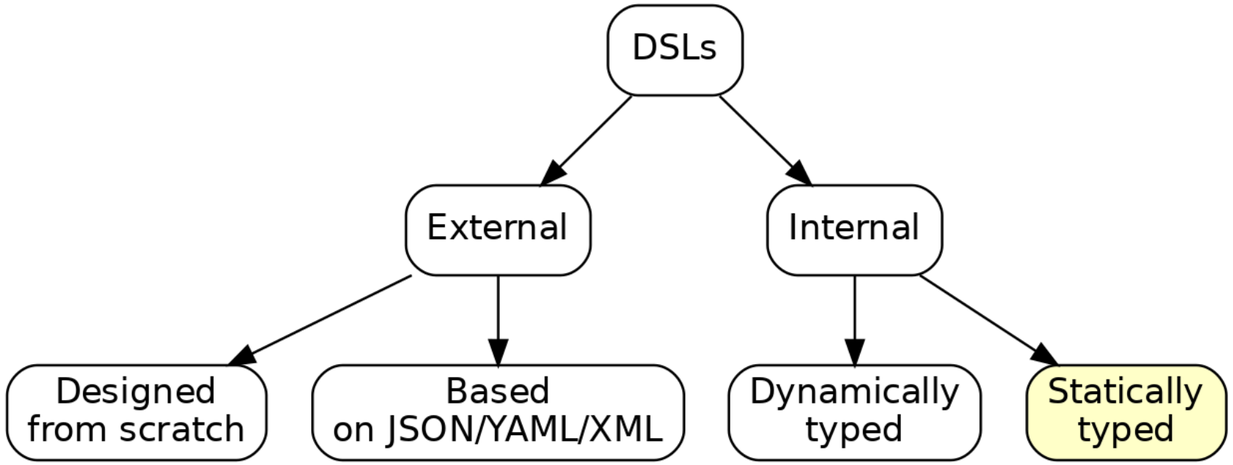
Внутренние DSL



Подмножество динамически типизированного языка

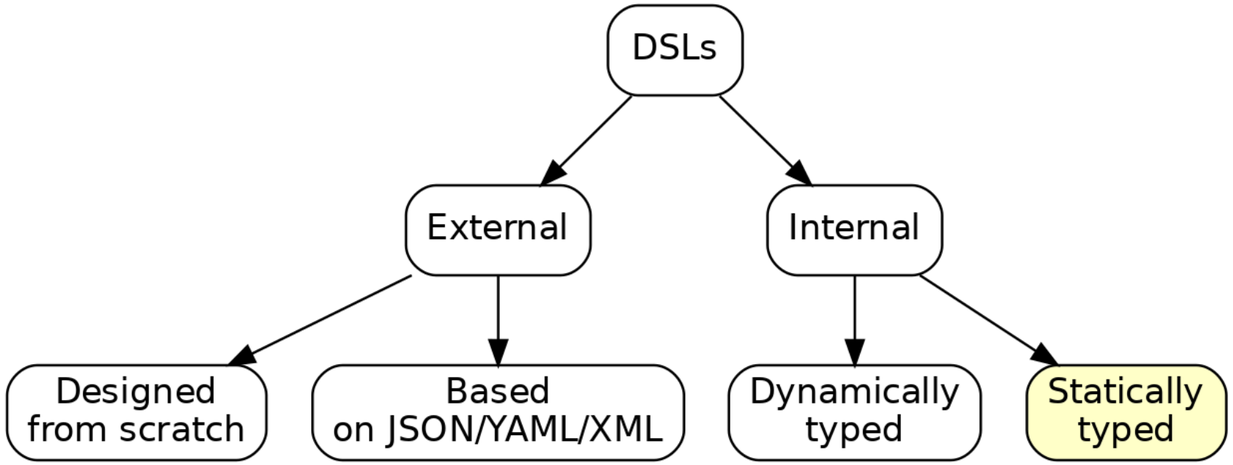
- **Lisp** (historically first): Emacs Lisp, Symbolic Mathematics etc.
- **Ruby**: Rails, RSpec, Chef...
- **Groovy**: Spock, Ratpack, Grails, Gradle, Jenkinsfile...

Внутренние DSL



Подмножество статически типизированного языка

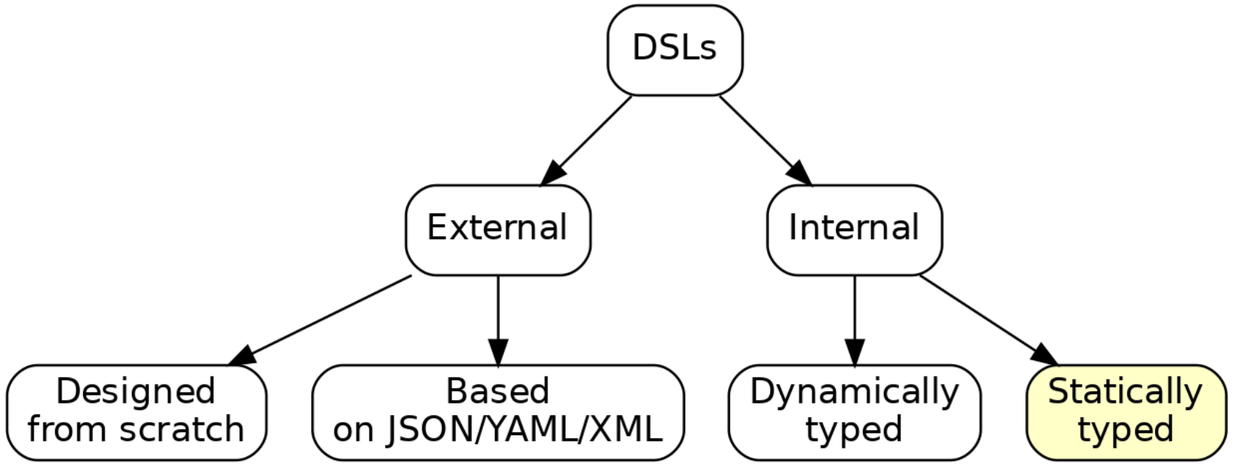
Внутренние DSL



Подмножество статически типизированного языка

- **Scala**: Scalatest, Akka HTTP...

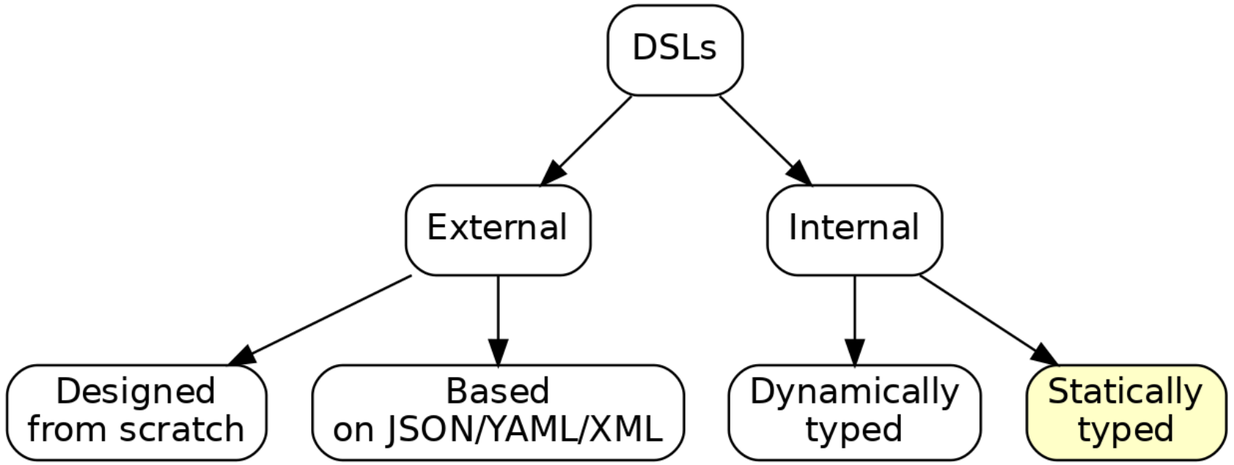
Внутренние DSL



Подмножество статически типизированного языка

- **Scala**: Scalatest, Akka HTTP...
- **Haskell**: Parsec

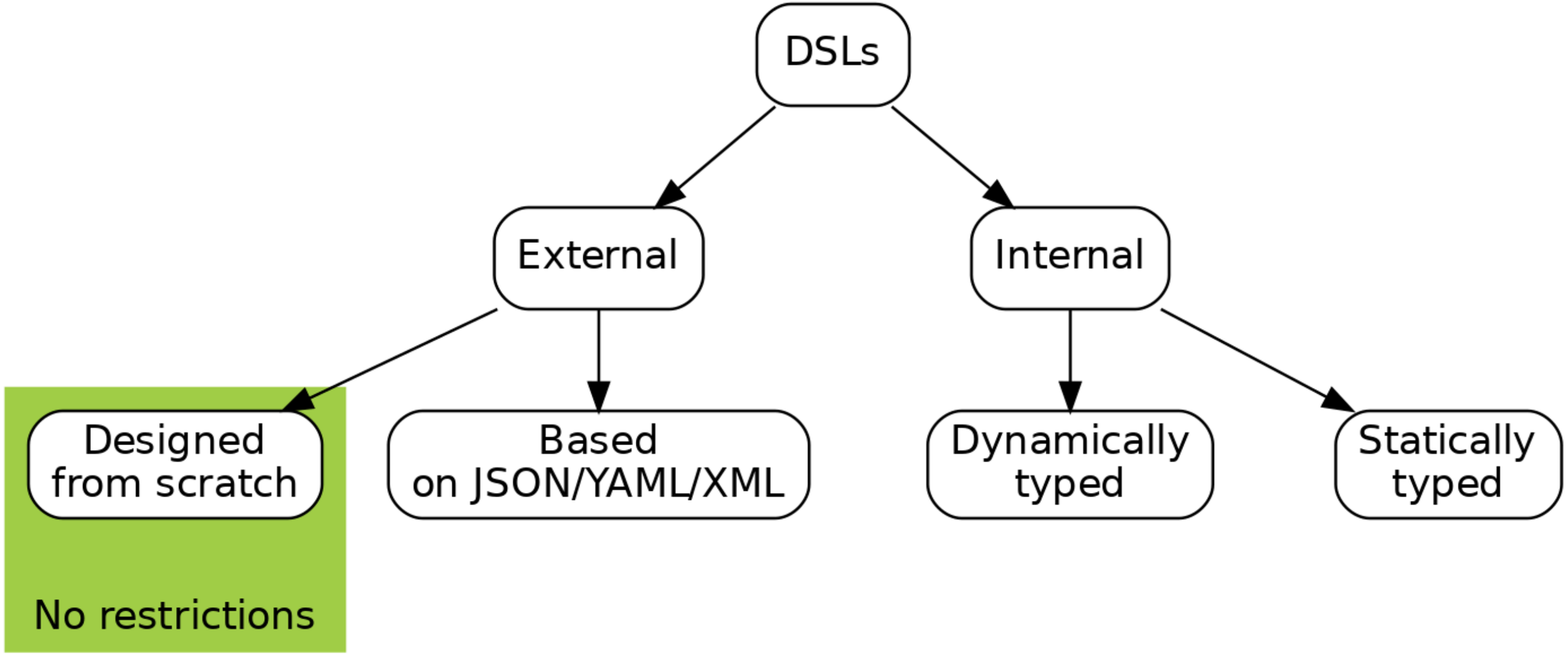
Внутренние DSL



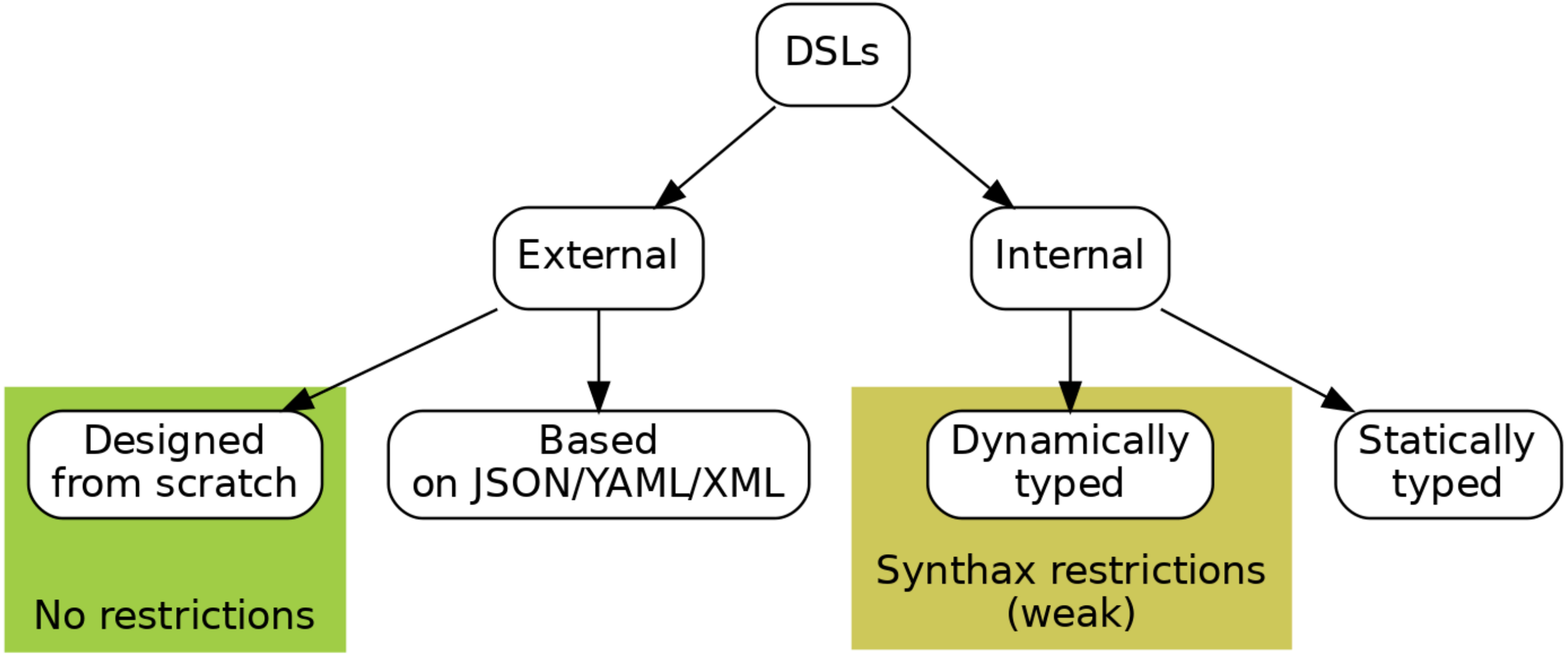
Подмножество статически типизированного языка

- **Scala**: Scalatest, Akka HTTP...
- **Haskell**: Parsec
- **Kotlin**: Kotlinx.html, Ktor, Gradle,...

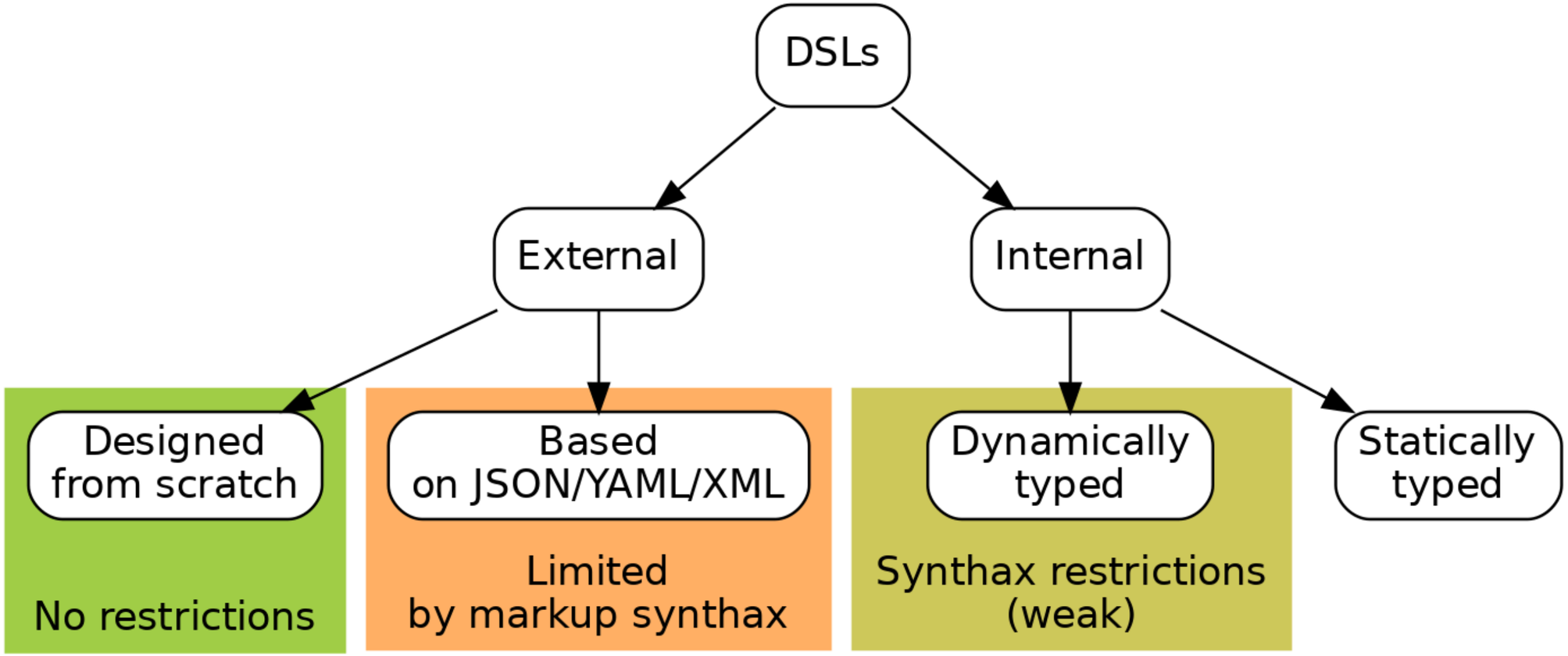
Ограничения



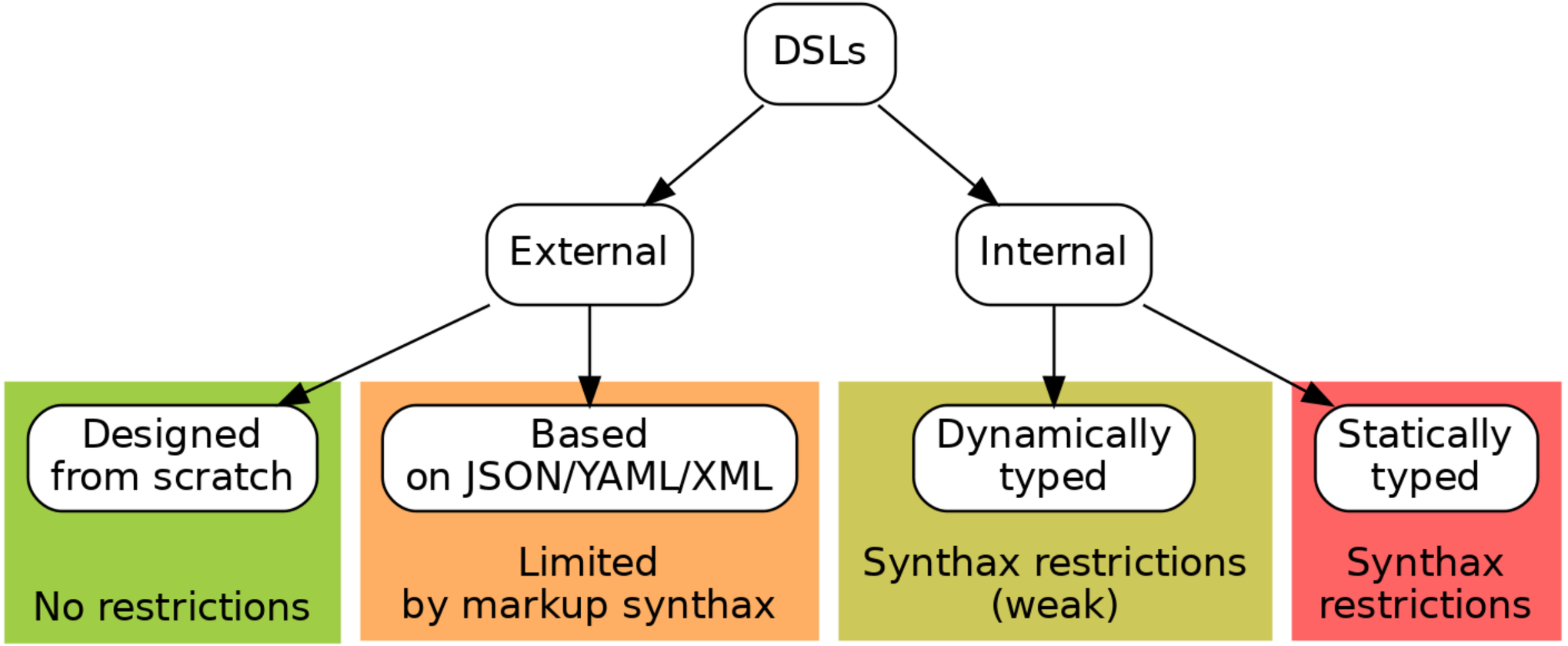
Ограничения



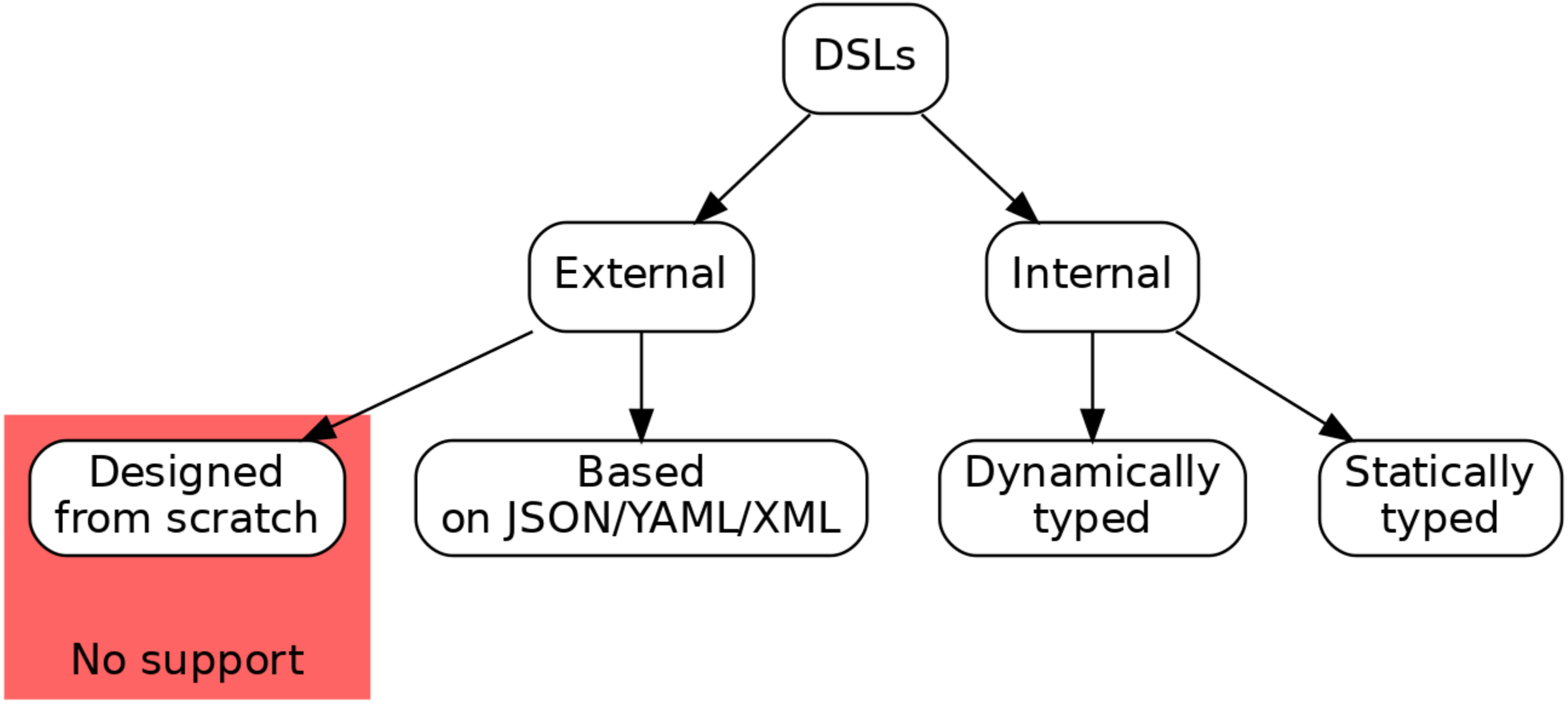
Ограничения



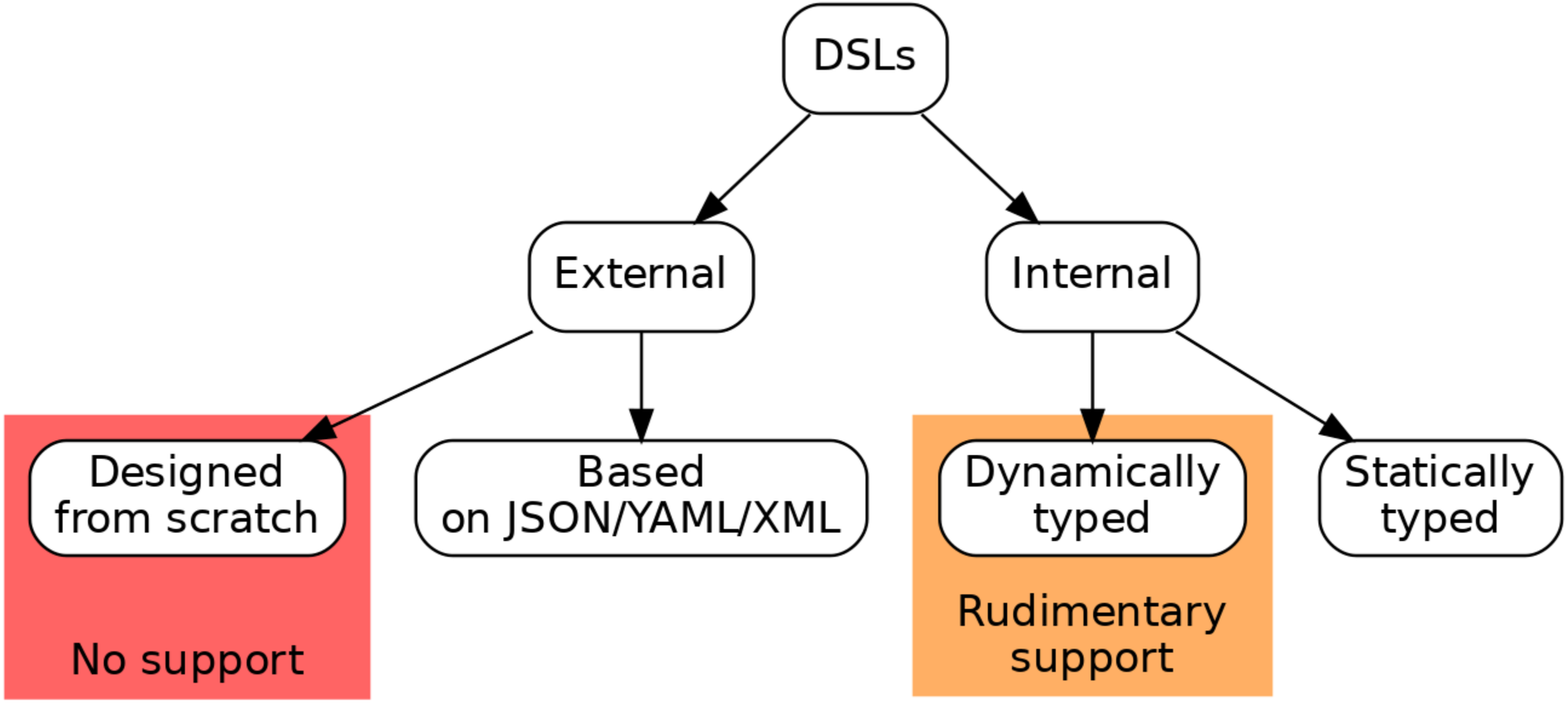
Ограничения



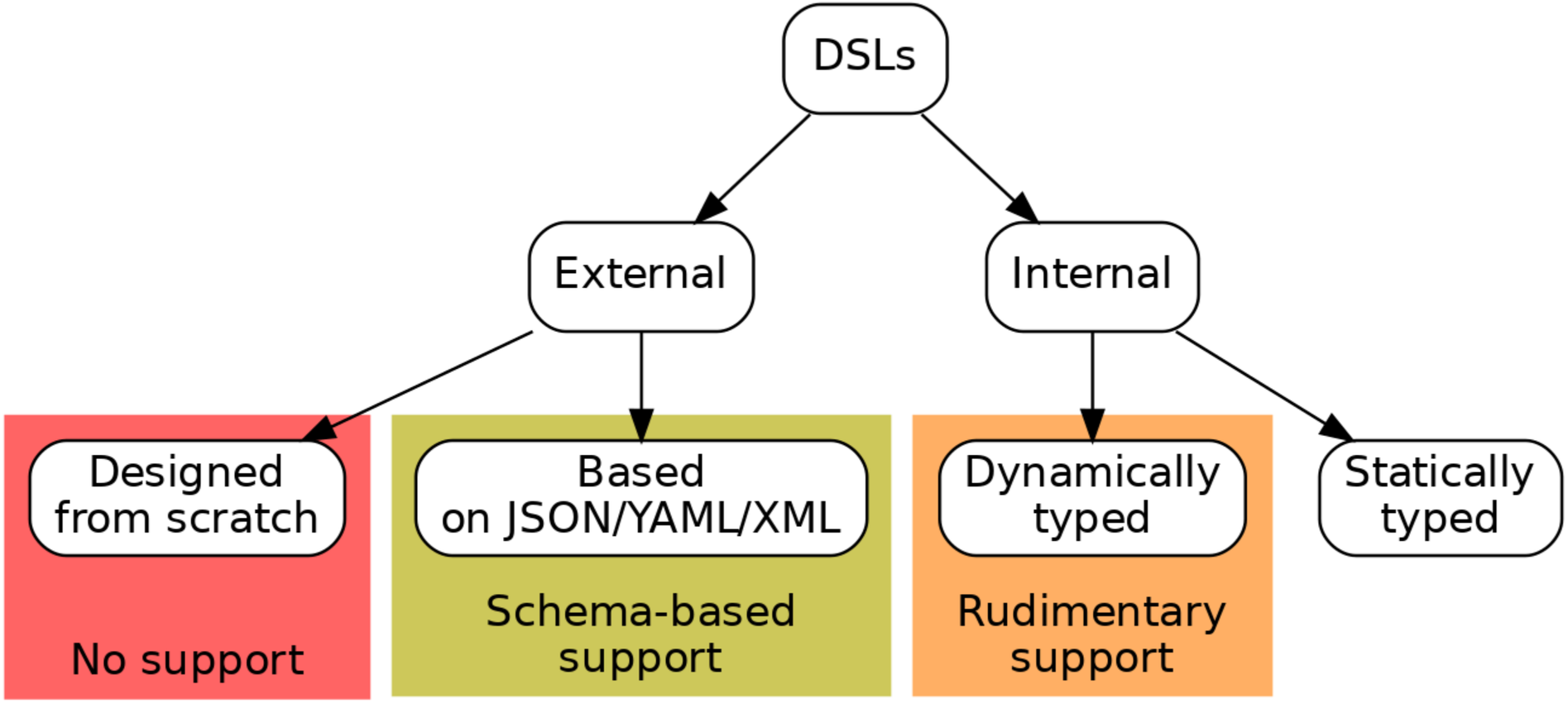
Поддержка IDE



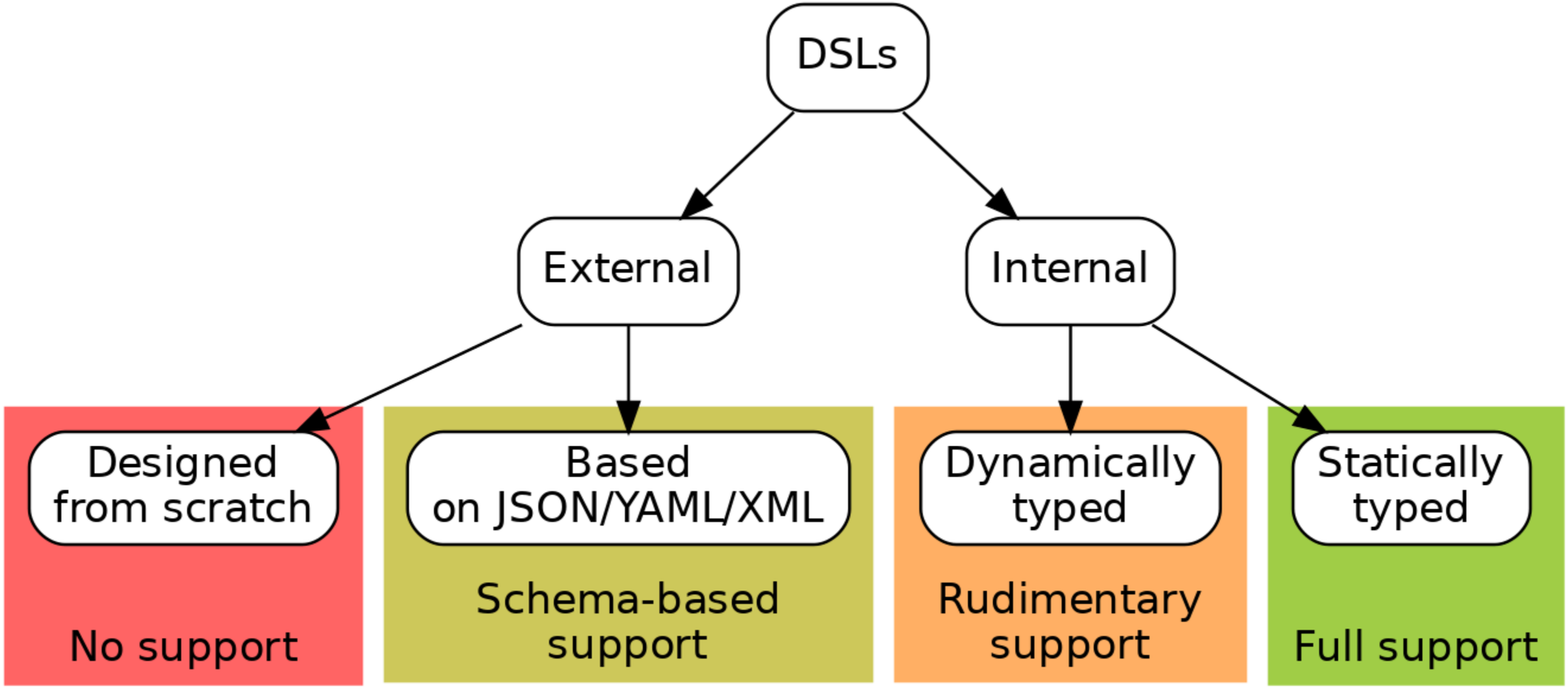
Поддержка IDE



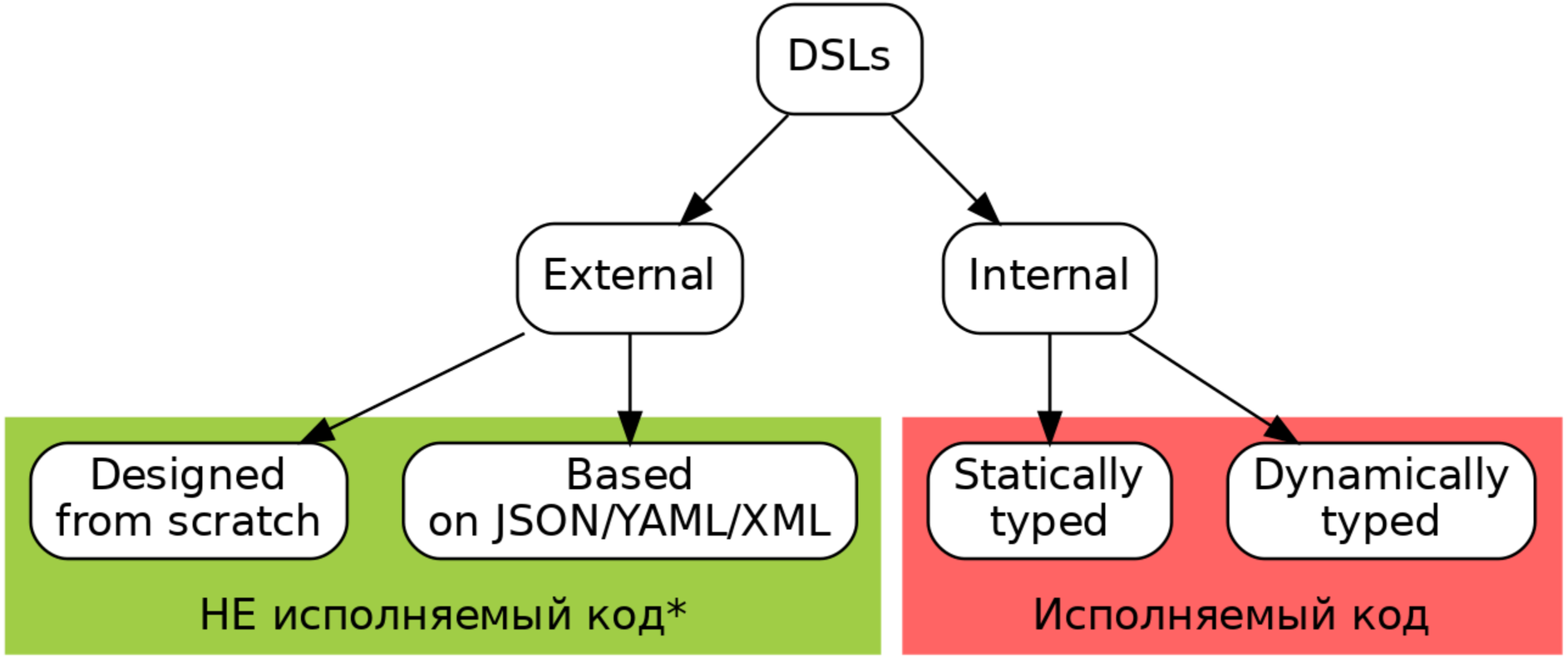
Поддержка IDE



Поддержка IDE



Соображения безопасности

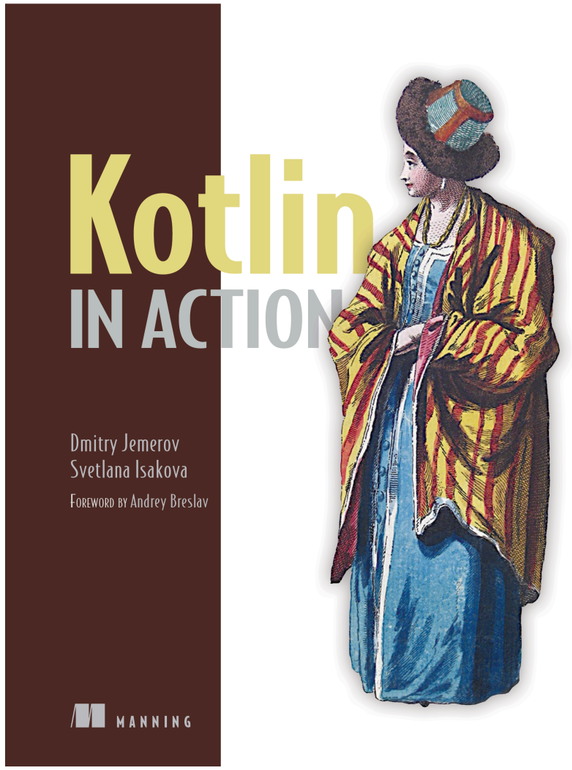


* **НЕ** означает, что вы в полной безопасности, см. напр. "Billion_laughs_attack" или доклады Сергея Васильева на Heisenbug и Joker про уязвимости XML-парсинга.

Промежуточные выводы

- Kotlin DSLs относятся к классу внутренних DSL на основе статических языков программирования
- У этого есть как преимущества, так и недостатки
- Выбор за вами!

Как нам построить Kotlin DSL?

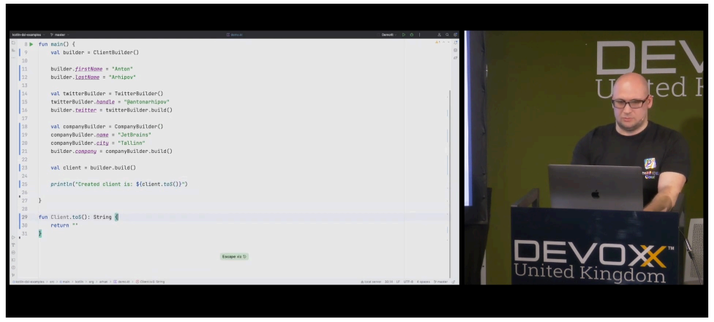


Дмитрий Жемеров, Светлана Исакова
Kotlin in Action
(Manning готовит второе издание)

Как нам построить Kotlin DSL?

- **Иван Осипов** Kotlin DSL: from Theory to Practice
  https://www.jmix.io/cuba-blog/kotlin-dsl-from-theory-to-practice/
- JPoint 2018/Heisenbug 2018:
  https://www.youtube.com/watch?v=q_UM1EY2S5g

Как нам построить Kotlin DSL?



- **Anton Arhipov**
  Kotlin DSL in under an hour
  https://www.youtube.com/watch?v=0DJqr4FZ6f0

Возможности языка Kotlin для построения DSL

Tool	DSL syntax	General syntax

Extension functions	`mylist.first();`
	`/* there isn't first() metho`
	`in mylist collection*/`
		`ListUtlis.first(mylist)`

Infix functions	`1 to "one"`	`1.to("one")`
Operators overloading	`collection += element`	`collection.add(element)`
Type aliases	`typealias Point = Pair`	Creating empty inheritors classes and other duct tapes

Kotlin language features for DSL building (continued)

Tool	DSL syntax	General syntax
get/set methods convention	`map["key"] = "value"`	`map.put("key", "value")`

Destructuring declaration	`val (x, y) = Point(0, 0)`	`val p = Point(0, 0)`
		`val x = p.first`
		`val y = p.second`

Lambda out of parentheses	`list.forEach { ... }`	`list.forEach({...})`
Lambda with receiver (главный инструмент)	`Person().apply { name = "John" }`	N/A
Context control	`@DslMarker`	N/A

Demo time!

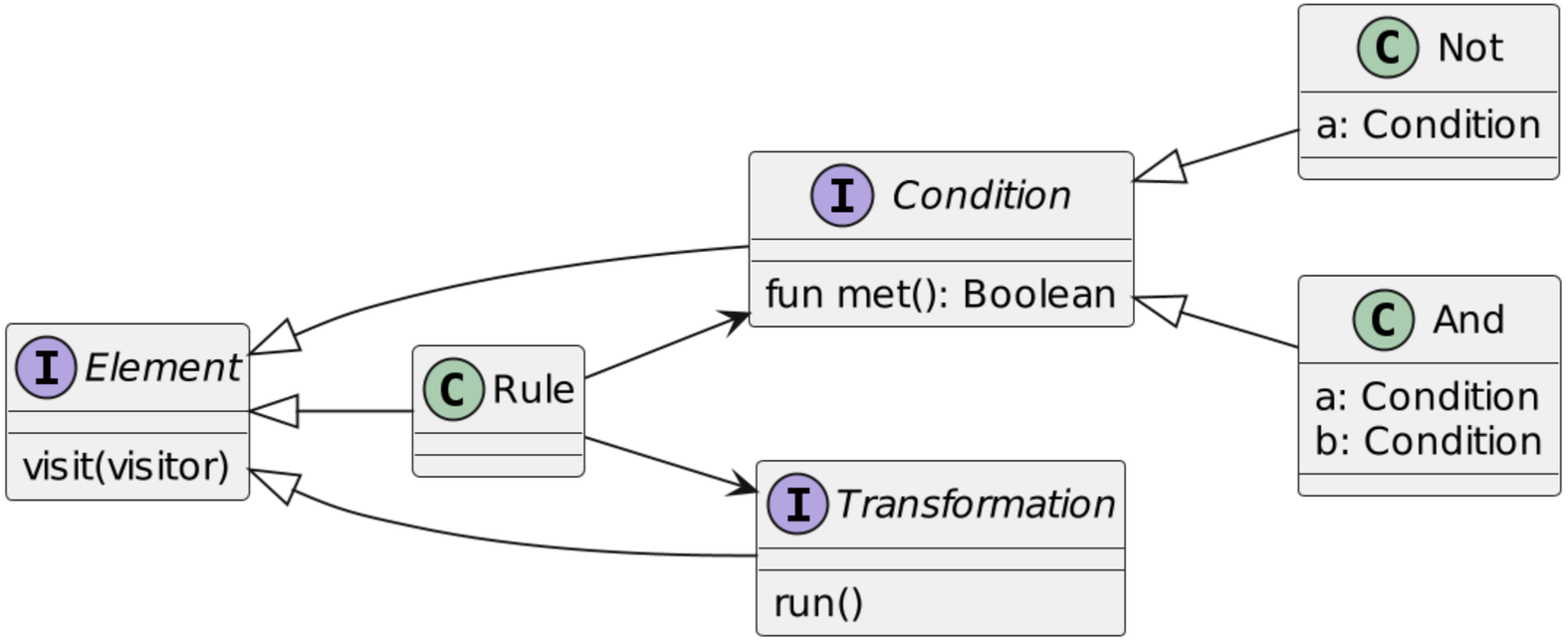
- Наша предметная область:
  - Условия
  - Трансформации
  - Правила
- Если выполняется набор условий — запускается нужная трансформация

Стартовое состояние: императивный код

```
fun main() {
    if (conditionOneMet() && conditionTwoMet()) {
        runTransformationA()
    } else if ((ConditionIII.met() || ConditionIV.met()) && conditionOneMet()) {
        runTransformationB()
    } else if (conditionTwoMet()) {
        runTransformationC()
    }
}
```

- пошаговая отладка!
- тестирование с оглядкой на coverage (хотя подмена `conditionXmet()` и `runTransformationX()` моками может представлять трудность)
- код разрастается и быстро становится трудно читаемым/поддерживаемым

Объектная модель



Паттерн "Strategy", декларативный код

```
private fun rules(): List<Rule> = listOf(
    Rule(ConditionII, TransformationC),
    Rule(Not(ConditionIV), TransformationB),
    Rule(And(ConditionI, ConditionII), TransformationA),
    Rule(Or(And(ConditionIII, ConditionIV), ConditionI), TransformationB)
)


fun main() {
    rules()
        .firstOrNull { it.condition.met() }
        ?.transformation?.run()
}
```

- Такое можно написать и на Java, но на Kotlin получилось компактнее из-за отсутствия `new`.
- Визуально связь правил и трансформаций лучше воспринимается
- Пошаговая отладка превратилась в ад

Kotlin DSL во всей красе

```
val rules: List<Rule> =
    // @formatter:off
    rules {
        ConditionI and ConditionIV                invokes TransformationA
        ConditionII                               invokes TransformationC
        not(ConditionIV)                          invokes TransformationB
        (ConditionI and not(ConditionIII))        invokes TransformationA
        (ConditionIII
                and ConditionIV
                or ConditionI)                    invokes TransformationB
    }
   // @formatter:on
```

- Параметр метода `rules` — лямбда с ресивером
- `and`, `or`, `not`, `invokes` — инфиксные функции-расширения

Выполнение

```
fun main() {
    rules.firstOrNull { it.condition.met() }?.transformation?.run()
}
```

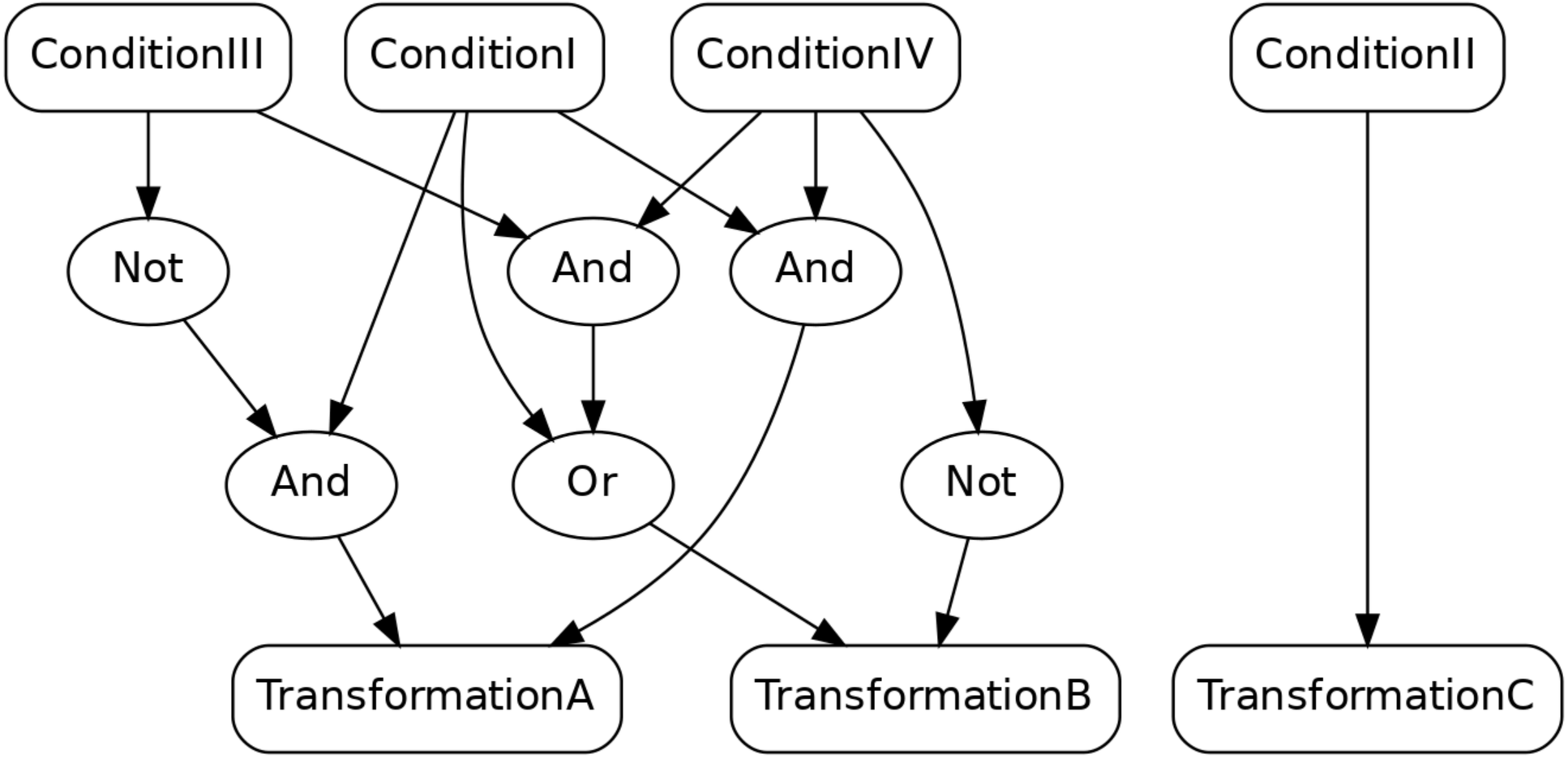
- Достоинство: весь код в одну строчку
- Недостаток: пошаговая отладка правил невозможна (но это можно компенсировать, см. далее)

Визуализация: транспиляция в DOT

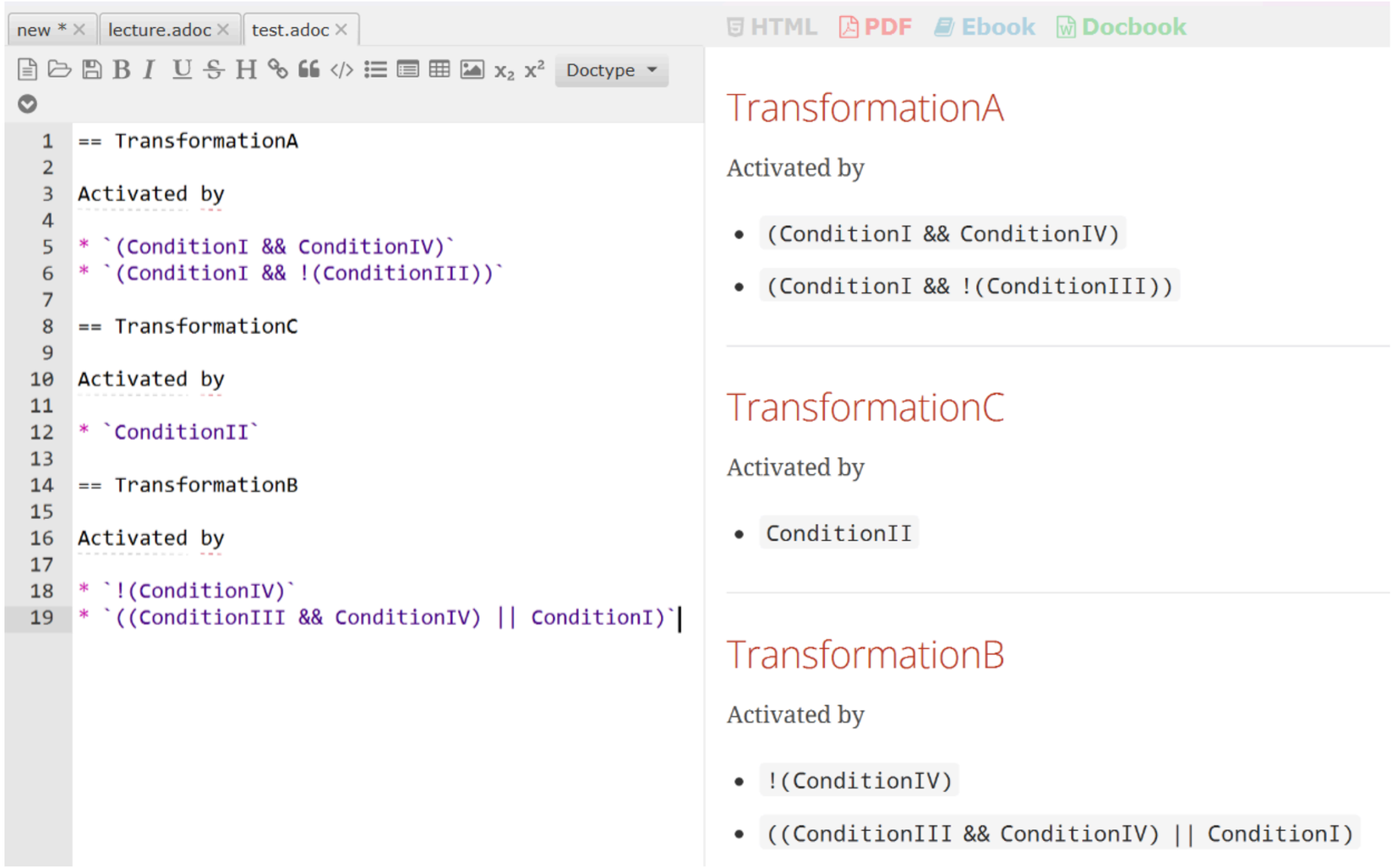
```
for (rule in rules) {
    rule.visit(::visitor)
}

class Rule(...) : Element {
    override fun visit(visitor: (Element) -> Unit) {
        condition.visit(visitor)
        transformation.visit(visitor)
        visitor.invoke(this)
    }
}
```

Хитросплетения правил и трансформаций



Документирование: транспиляция в Asciidoctor



DocOps и автоматизация документирования

DocOps: Шоссе к актуальной документации

Up-to-date doc highway

Николай Поташников
Руководитель проектов, IT-архитектор
КУРС-ИТ

consulting@yandex.ru @nmpotashnikov

HEISENBUG

1

DSL-конструкции языка Kotlin и архитектура как код

Николай Поташников
Руководитель проектов, IT-архитектор
КУРС-ИТ

consulting@yandex.ru @nmpotashnikov

JPoint 2023

1

Сериализация

- "Почти бесплатное" представление DSL в виде JSON/YAML/XML средствами, например, FasterXML Jackson.
- Идеально для построения WebUI с формами для показа/редактирования настроек (projectional editors)

```
- condition:
    And:
      a:
        ConditionI: {}
      b:
        ConditionIV: {}
  transformation:
    TransformationA: {}
```

Тестирование на сложные ограничения

Код, порождающий 2^N комбинаций множеств выполняющихся условий,
где N — число субклассов `BasicCondition`

```
private val conditions = BasicCondition::class.sealedSubclasses
fun outcomes(): Sequence<Set<ConditionClass>> = sequence {
    for (i in 0L until (1L shl conditions.size)) { // тут возникает 2^N
        val activeConditions = mutableSetOf<ConditionClass>()
        for (j in 0 until conditions.size) {
            if ((i and (1L shl j)) != 0L) {
                activeConditions.add(conditions[j])
            }
        }
        yield(activeConditions)
    }
}
```

Проверка модели на непротиворечивость

- "Не существует недостижимых правил"
- "При фиксированном условии, каждое из правил определенного класса достижимо"
- ... и т. д. — всё зависит от вашей задачи

Тестируем саму модель, заданную в DSL, а не результат её интерпретации!

Один исходник может быть использован для

1. Выполнения правил
2. Генерации документации
3. Визуализации
4. Валидации
5. Сериализации ("бесплатная" JSON/YAML/XML-версия нашего Kotlin DSL)

Продвинутые вопросы DSL-строения

Досадное ограничение Kotlin

Groovy Gradle DSL:

```
implementation 'com.acme:example:1.0'
```

Kotlin Gradle DSL:

```
implementation ("com.acme:example:1.0")
```

Инфиксные функции не работают на `this`

```
val jpoint = javaConference {
    //Без круглых скобок нельзя (в отличие от Groovy)
    talk("Пишем приложение на Ktor") deliveredBy {
        speaker("Александр Нозик")
        speaker("Глеб Королькевич")
    }

    talk("One source to rule them all: Kotlin DSL") deliveredBy {
        speaker("Иван Пономарев")
    } withExperts {
        speaker("Андрей Кулешов")
    }
}
```

Инфиксные функции не работают на `this`: обходной манёвр

```
val jpoint = javaConference {
    //Всё без скобок (но и без осмысленного имени метода)
    + "Пишем приложение на Ktor" deliveredBy {
        + "Александр Нозик"
        + "Глеб Королькевич"
    }

    + "One source to rule them all: Kotlin DSL") deliveredBy {
        + "Иван Пономарев"
    } withExperts {
        + "Андрей Кулешов"
    }
}
```

@DslMarker

По смыслу стуктуры DSL нам бы такого не хотелось, но лямбда с ресивером это не запрещает:

```
javaConference { //this: ConferenceBuilder
    talk ("Talk 1") deliveredBy {
        //this: ConferenceBuilder.SpeakersBuilder,
        //но также доступны методы из ConferenceBuilder
        talk (...) // ??!!!
    }
}
```

@DslMarker

```
@DslMarker
annotation class MeetupDsl

@MeetupDsl
class MeetupBuilder { ... }

@MeetupDsl
class SpeakersBuilder { ... }
```

- Также обещана (но не задокументирована) расширенная поддержка со стороны IDE, поэтому имеет смысл размечать DSL-биледеры с помощью `@DslMarker` в любом случае.

Кросс-ссылки

```
private val jpoint = javaConference {

    val ip = Speaker("Иван Пономарев", "N/A")

    talk("One source to rule them all: Kotlin DSL") deliveredBy {
        + ip
    } withExperts {
        + Speaker("Андрей Кулешов", "Huawei")
    }

    talk("Kotlin Script: для кого, зачем и как") deliveredBy {
        + Speaker("Анатолий Нечай-Гумен", "Банк «Центр-инвест»")
    } withExperts {
        + ip
    }

}
```

- `val ip = …` выглядит как императивный код, но ничего тут поделать нельзя
- В Groovy тут гораздо больше возможностей

Захват имени переменной при помощи делегирования

```
//ConferenceBuilder
val speakers = mutableMapOf<String, Speaker>()
//Функция возвращает делегат свойства только для чтения
fun speaker(name: String, company: String): ReadOnlyProperty<Nothing?, Speaker> =
    ReadOnlyProperty { _, property ->
        //property.name содержит имя переменной
        speakers.computeIfAbsent(property.name) { Speaker(it, name, company) }
    }
```

Захват имени переменной при помощи делегирования

```
//Сам язык будет гарантировать нам уникальность идентификаторов
val an by speaker("Александр Нозик", "МФТИ")
val gk by speaker("Глеб Королькевич", "Хоум Банк")

talk("Пишем приложение на Ktor") deliveredBy {
    +an
    +gk
}
```

Вдохновляющие примеры

- Gradle Kotlin DSL
- Ktor Framework: https://ktor.io/ (а кстати на этом JPoint есть воркшоп на тему Ktor!)
- Exposed (an ORM for Kotlin): https://github.com/JetBrains/Exposed?tab=readme-ov-file#examples

Выводы

Выводы

- DSL в сочетании с дизайн-паттернами представляет собой мощный инструмент для решения множества задач.

Выводы

- DSL в сочетании с дизайн-паттернами представляет собой мощный инструмент для решения множества задач.
- Создавать DSL в Kotlin не страшно. Прямо сегодня вы можете улучшить части существующих внутренних API, сделав их «DSL-подобными».

Выводы

- DSL в сочетании с дизайн-паттернами представляет собой мощный инструмент для решения множества задач.
- Создавать DSL в Kotlin не страшно. Прямо сегодня вы можете улучшить части существующих внутренних API, сделав их «DSL-подобными».
- Внутренние DSL Kotlin — не единственный способ реализации DSL, со своими сильными и слабыми сторонами, но определенно не самый худший во многих сценариях.

Спасибо за внимание!

Код и слайды доступны GitHub
https://github.com/inponomarev/dsl-talk

ivan@synthesized.io
@inponomarev