День 3

- Поговорим о том, как сделать SDK API: поймем как проектируются SDK и научимся предполагать их структуру
- Поработаем с SDK на примере Ansible: рассмотрим реализацию вышеназванных принципов, написав модуль для Ansible
- Расслабимся и поговорим о взаимодействии в команде: разберемся как жить с тем, что программирование только часть работы
- Подведем итоги

Что думаете насчет софт-скилов и фреймворков работы?

1. Это рычаги давления менеджмента на инженеров
2. Это работает по принципу Парето: 20% пользы и 80% пыли в глаза
3. Это работает по принципу Парето: 80% пользы и 20% пыли в глаза
4. Это полезные метрики и фреймворки, при условии, что менеджмент понимает как их использовать
5. Это работает на инженеров, экранируя их от менеджмента

Python API

гибко и расширяемо

СПЕРМ

На что делать упор?

- **Системный анализ** - выделение доменов функциональности
- **Создание промежуточных интерфейсов** для доступа к низкоуровневой логике
- **Унификация интерфейсов** для доступа к условной функциональности

На что делать упор?

- **Снижать зацепление** - минимизировать связи между отдельными модулями программы. В идеале - единый интерфейс со стороны Core и реализующие его модули

- **Поддерживать в необходимой и достаточной силе связность**.
  Важно не делать связность ради связности, если контекст того не требует,
  но и не отказывать себе в увеличении связности в неограниченных пределах
  по необходимости

- **Строго следить за вертикальной иерархией зависимостей**. Не допускать зависимостей верхнеуровневого от низкоуровневого. В идеале иметь четкую иерархию, приемлемо делать зависимости “через слой”

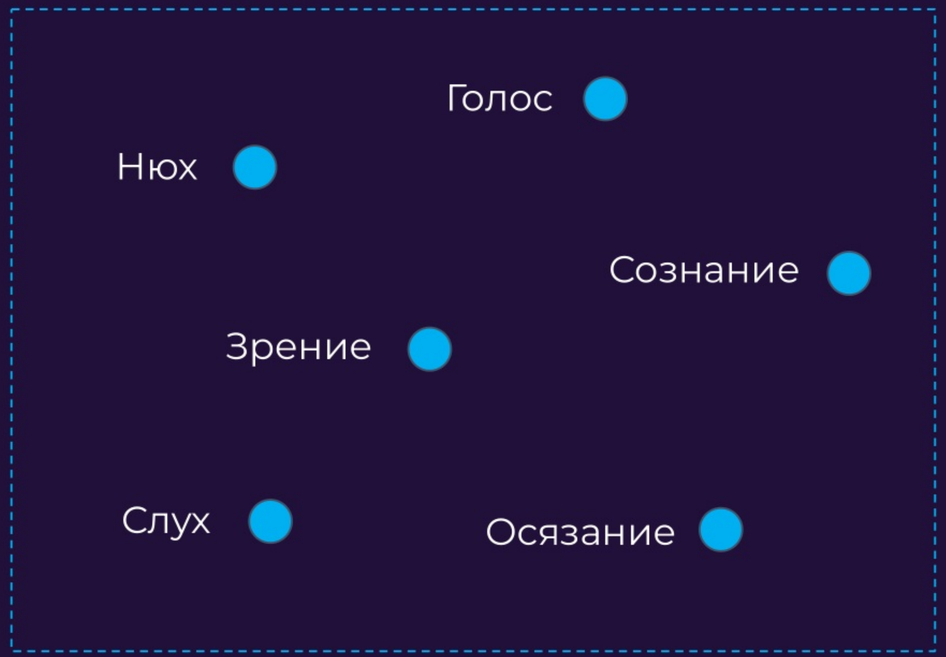
Задача

Проектируем кота

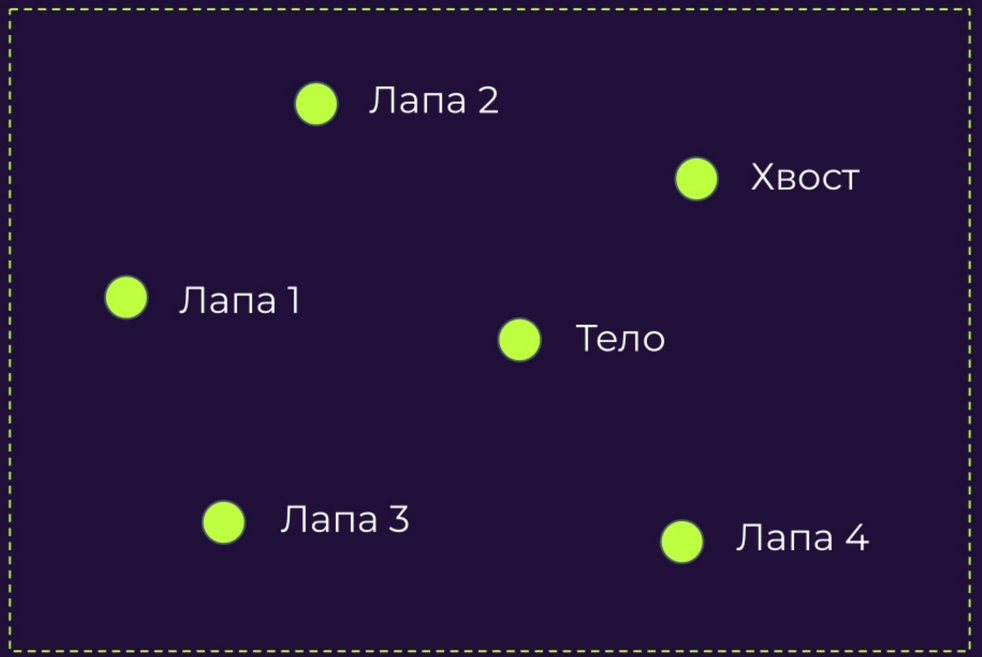
- Кот должен уметь ходить (последовательно перемещать 1-3 и 2-4 пары лап) и бегать (перемещать 1-2, 3-4 пары лап)
- При движении допускается использование хвоста и поворотов тела
- Кот должен использовать сигналы от органов чувств при принятии решений
- Кот должен использовать наиболее полную информацию при принятии решений

Выделение доменов функциональности

Модуль принятия решений

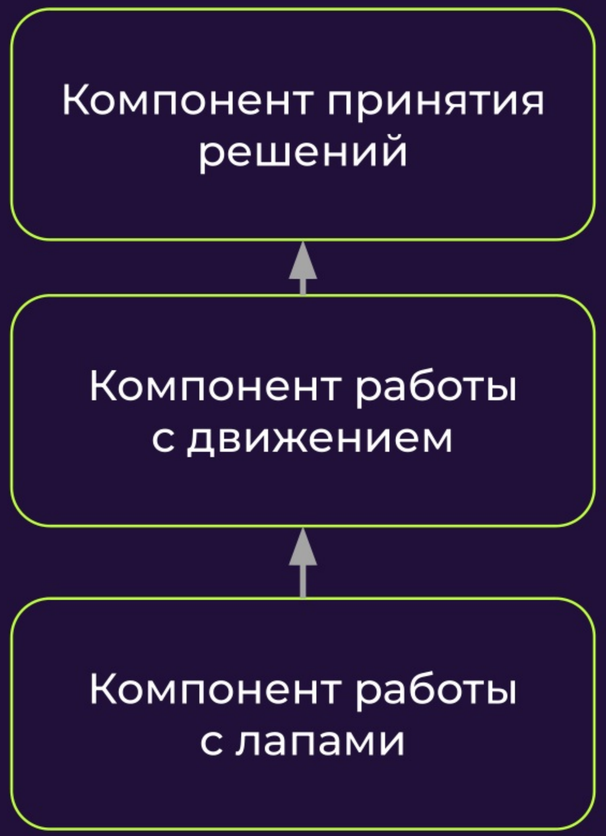


Модуль передвижения



Вертикальная иерархия

Идеально



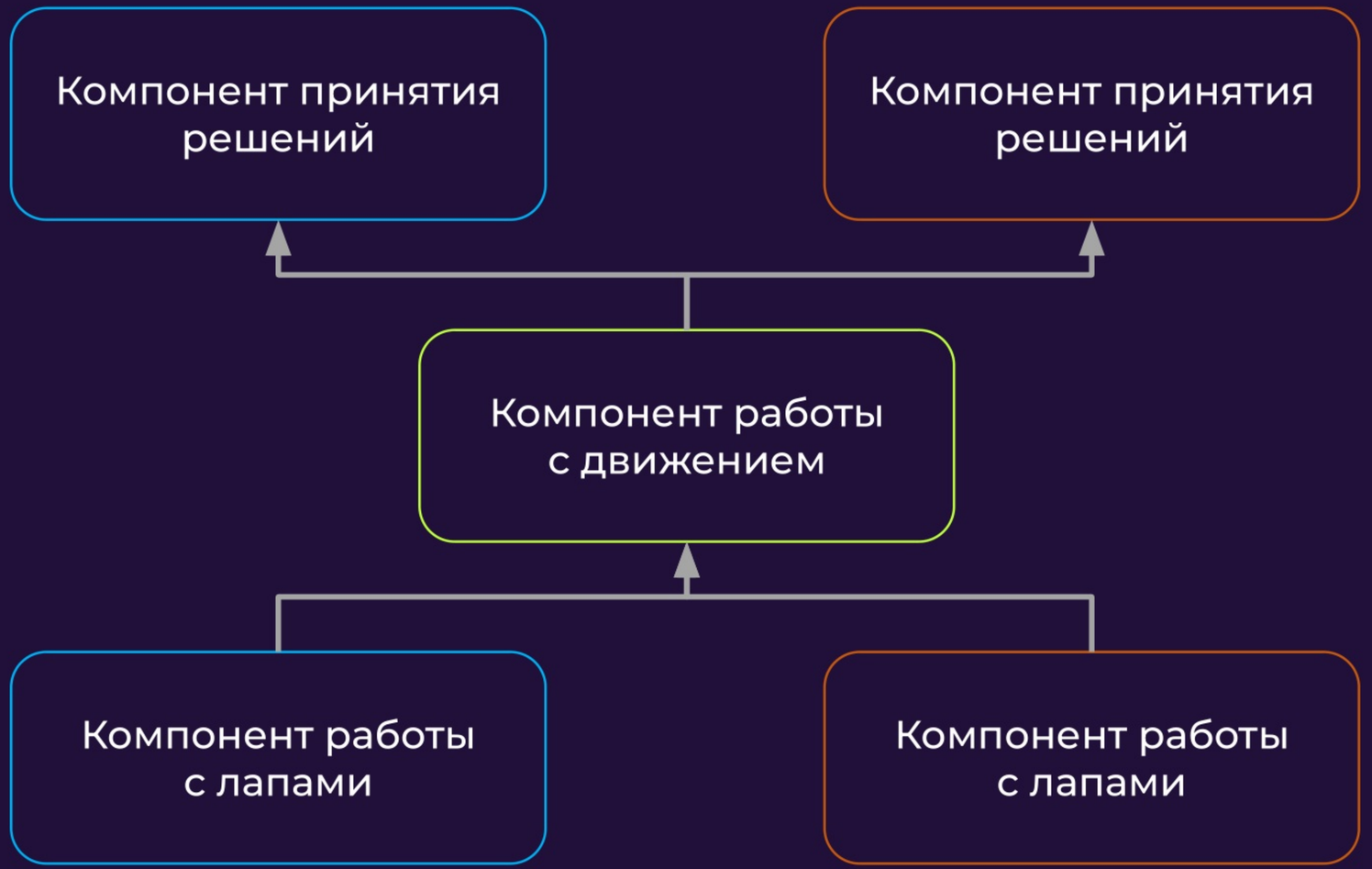
Хуже, но все еще можно

Плохо

Идеи, почему второй вариант хуже, чем первый?

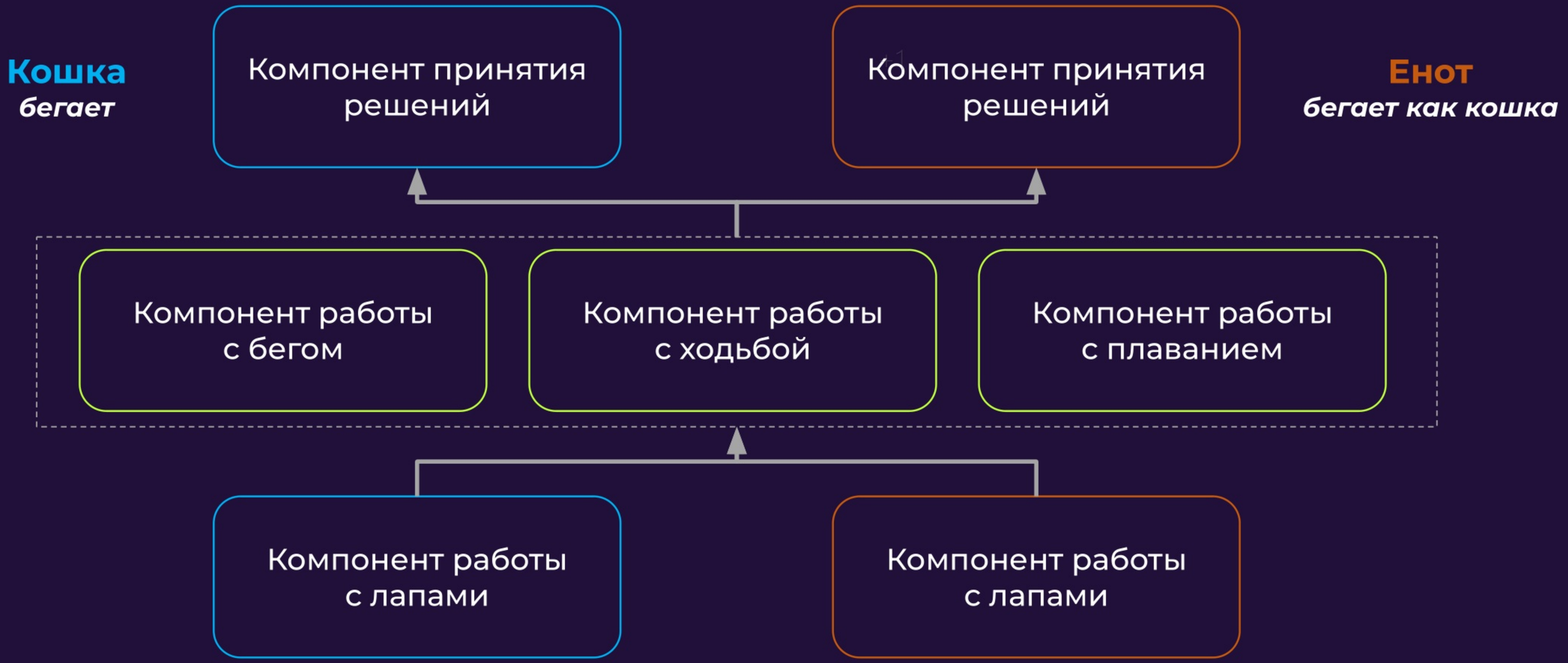
Промежуточные интерфейсы / контроллеры

Кошка
бегает

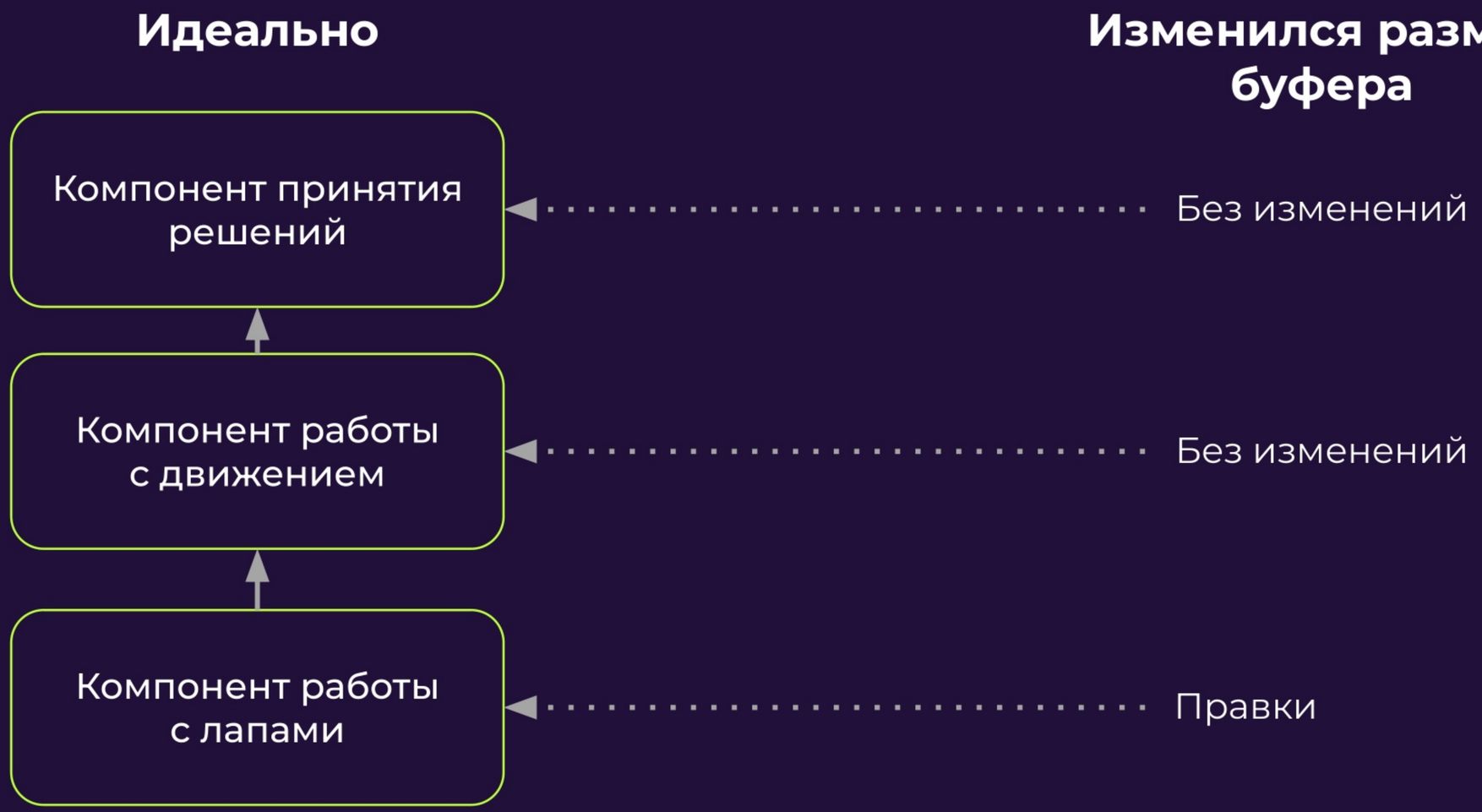


Енот
бегает как кошка

Унификация интерфейсов



Вертикальная иерархия



Этапы доработки

Модификация внутренней логики компонента работы с сетью

Функциональное тестирование внутренней логики компонента работы с сетью

...

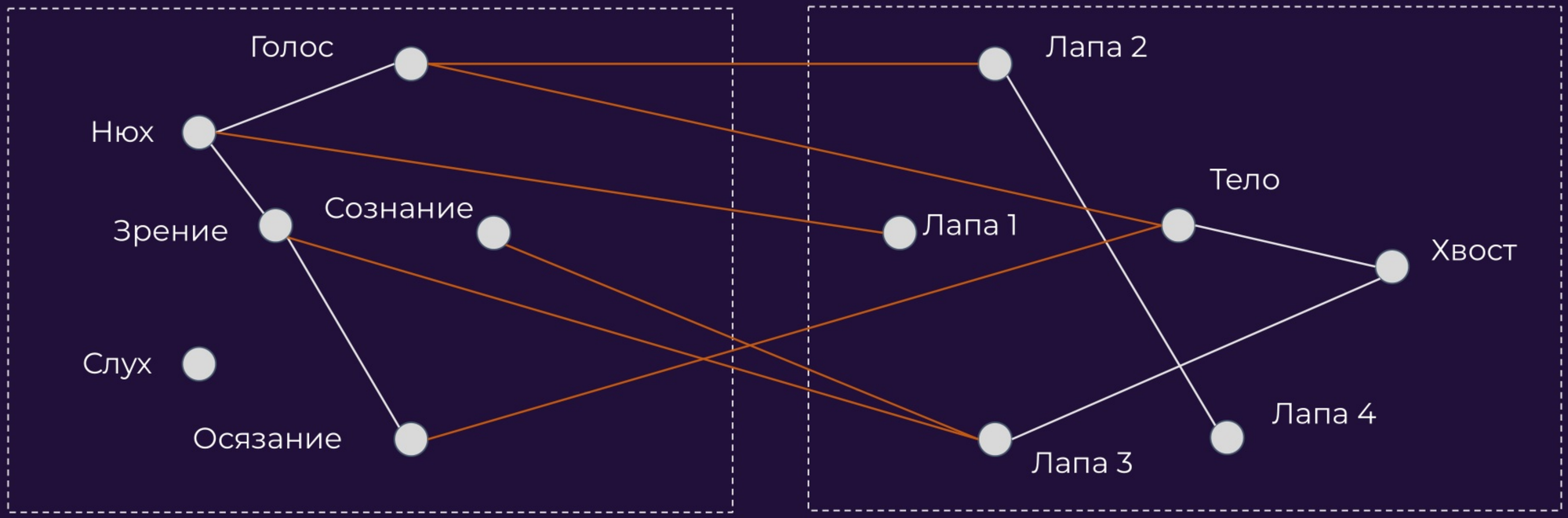
PROFIT

Вертикальная иерархия

Этапы доработки

- Модификация внутренней логики компонента работы с сетью
- Функциональное тестирование внутренней логики компонента работы с сетью
- Регрессионное тестирование внутренней логики компонента работы с протоколом
- Модификация внутренней логики компонента работы с протоколом
- Функциональное тестирование внутренней логики компонента работы с протоколом
- Регрессионное тестирование логики компонента работы с протоколом

Сильное зацепление и слабая связность



Баги:

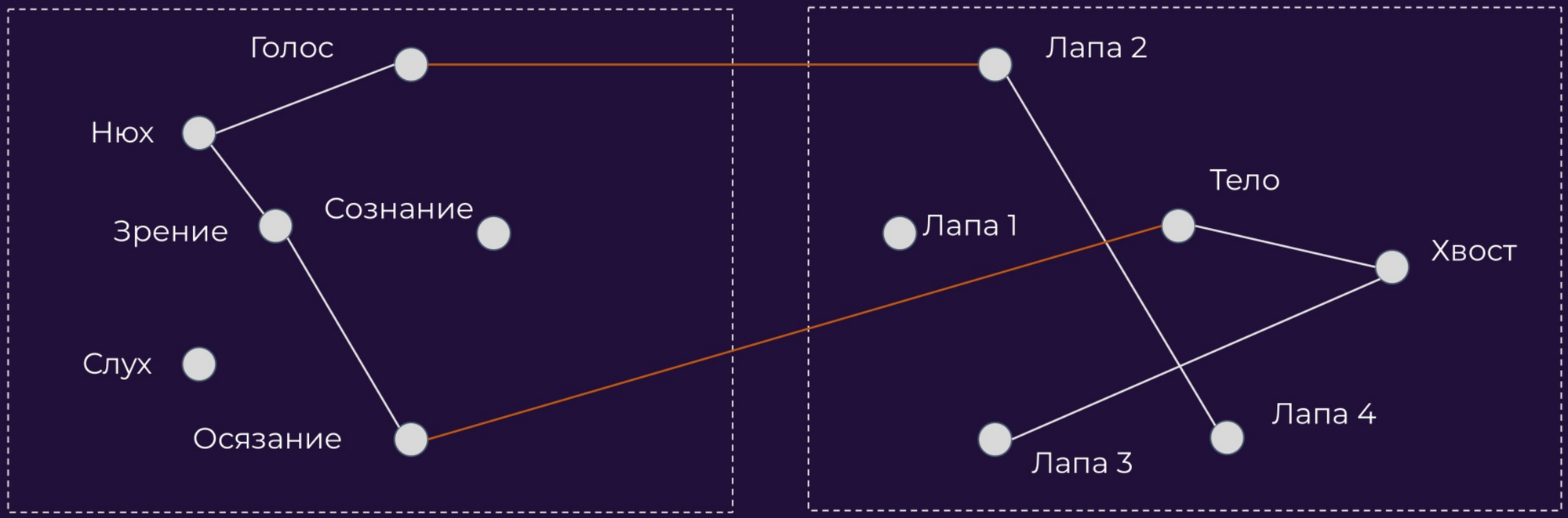
Синхронно двигаются только одна пара лап. С другим модулем восприятия кот не может бегать.

Если кот заметит опасность, то двигается только третья лапа. Чтобы двигались все, он вынужден кричать и воспринимать опасность по запаху.

Если двигается третья лапа - непроизвольно двигается хвост и тело.

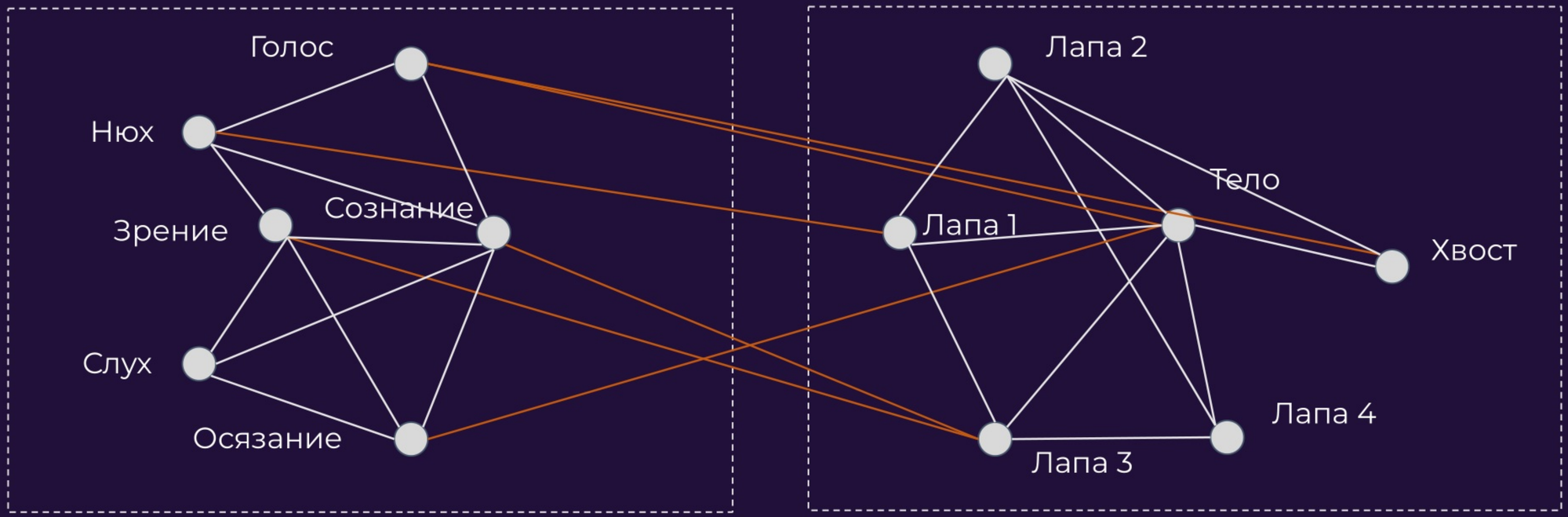
Кот глухой.

Слабое зацепление и слабая связность



Баги:
Одна лапа парализована.
Другая может двигаться только совместно с хвостом.
Кот все еще глухой.
Если кот ничего не чувствует он вынужден кричать, чтобы идти. Третью из лап при этом тоже парализует.

Сильное зацепление и сильная связность

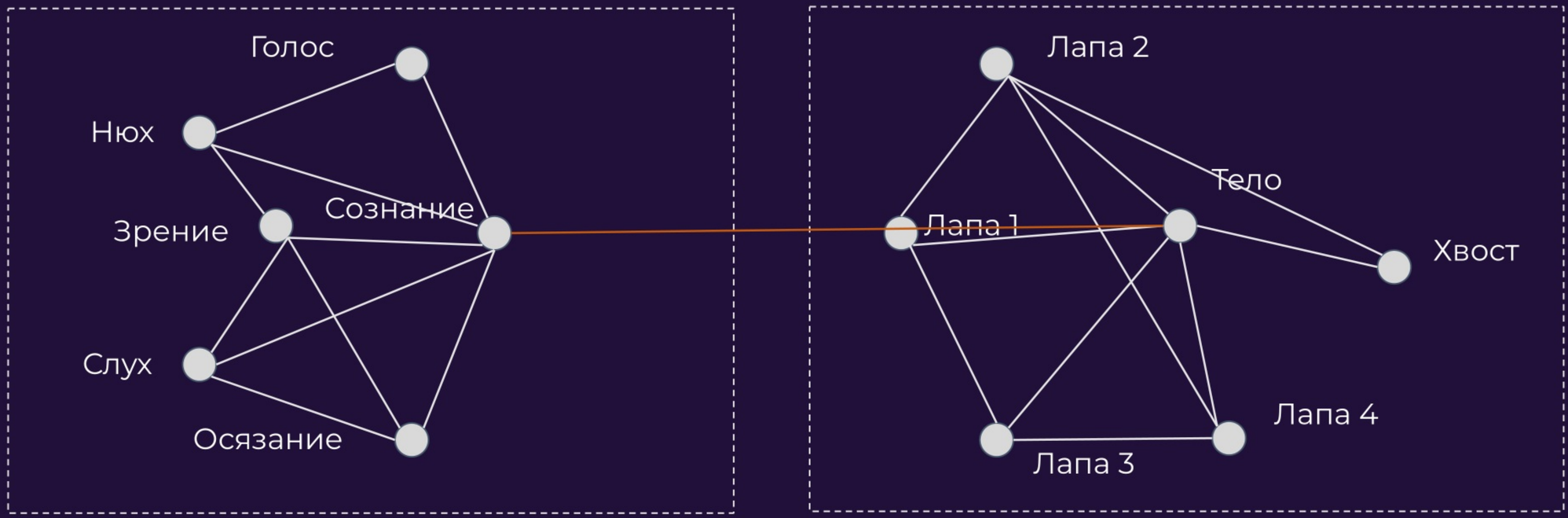


Баги:

Если кот кричит - кот движется и бьет хвостом.

При ярком свете кот хромает.

Слабое зацепление и сильная связность



Баги:

Вторая лапа сильно влияет на движение хвостом. Фича “Коту не нравится, когда трогают вторую лапу”

Паттерны GRASP позволяют избежать большую часть типовых архитектурных ошибок

+1

Паттерны(правила) GRASP

- Информационный эксперт - принцип единственной ответственности компонента
- Создатель - порождает объекты исходя из вводных данных. Вносит инверсию управления
- Контроллер и Перенаправление - Добавляет промежуточные интерфейсы
- Слабое зацепление и сильная связность - выделяет домены предметной области
- Полиморфизм - абстрагирует использование от зацепления
- Устойчивость к изменениям - append-only расширение интерфейсов или создание новых с дублированием старых

А теперь посмотрим,
как это выглядит на
примере Ansible Python SDK

Приходилось ли писать под Ansible?

1. Не приходилось
2. Конфигурации (роли, плейбуки, etc.)
3. Скриптовые расширения (передача скриптов в готовые модули/плагины)
4. Модули
5. Плагины

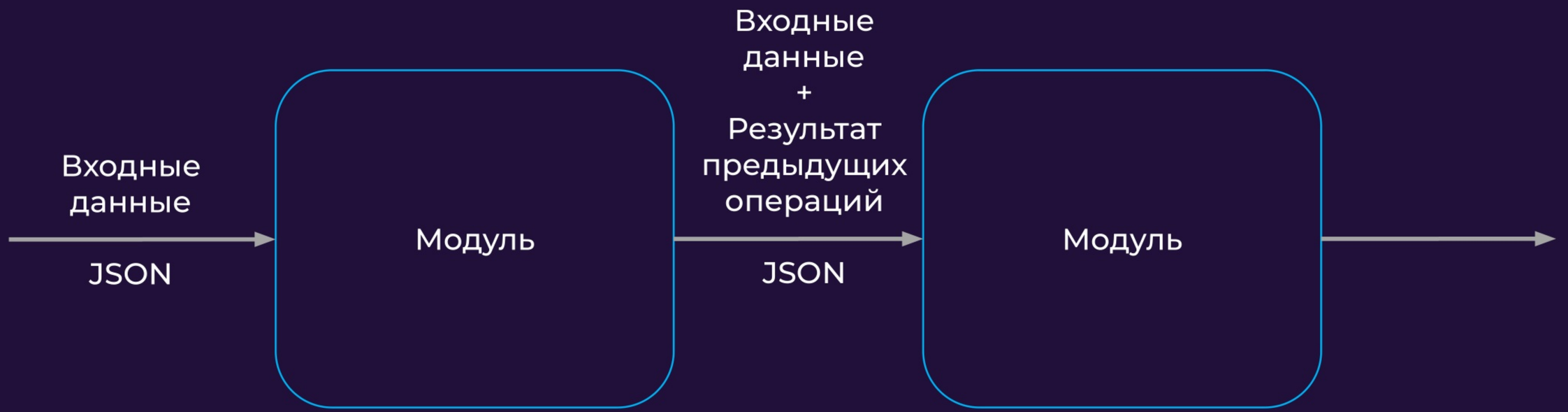
IoC в Ansible

Нужен ли модуль/плагин?

Да, если

- Необходимой функциональности нет в ansible galaxy
- Необходимой функциональности нет на github
- Необходимая функциональность является цепочкой действий (плейбук) и вы не можете обосновать необходимость вынесения такой функциональности в модуль

Взаимодействие модулей?



Доступные параметры