В поисках идеального CI-пайплайна

Илья Сауленко, Авито

Root Conf

Профессиональная
конференция
по эксплуатации и devops

Кто я?

Илья Сауленко

Ведущий разработчик в Авито

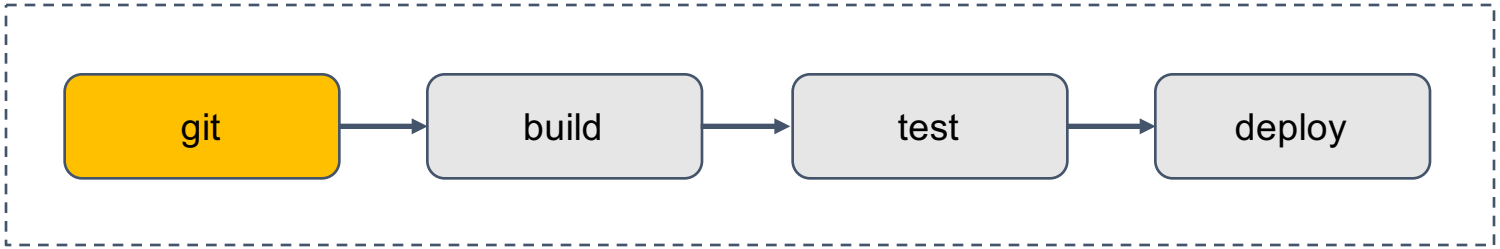
Занимаюсь платформой и Developer Experience

План доклада

1. История из жизни
2. Проблемы CI-серверов
3. Критерии современности CI-серверов
4. Идеальный пайплайн

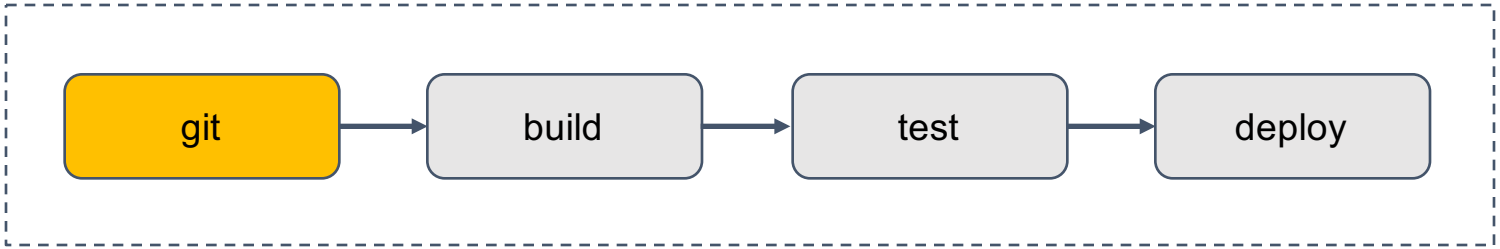
История из жизни

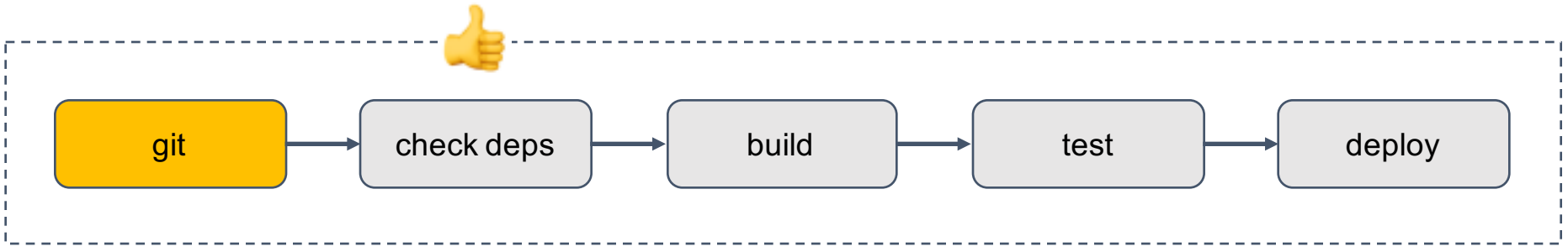
Отслеживание уязвимостей в библиотеках

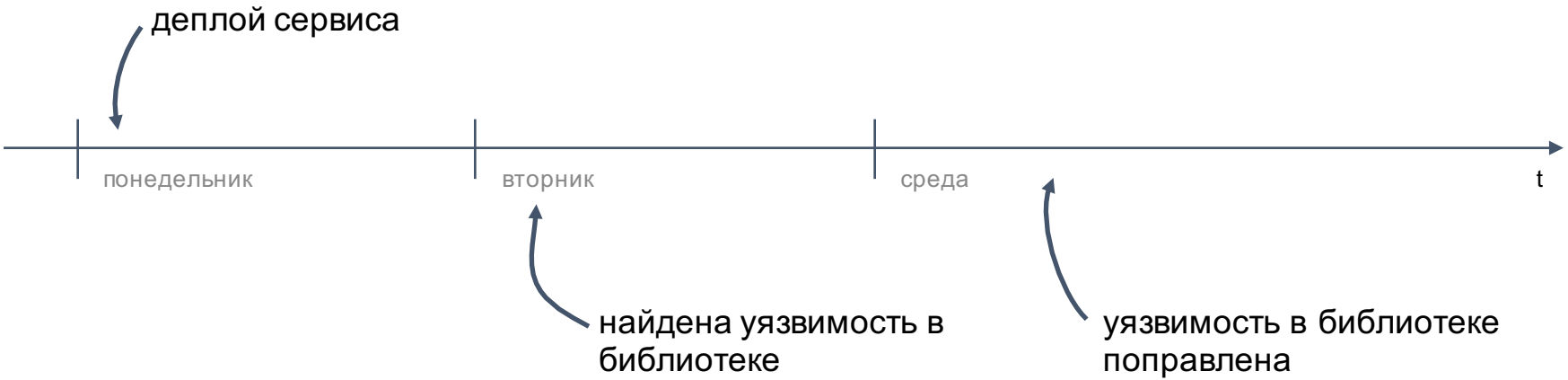


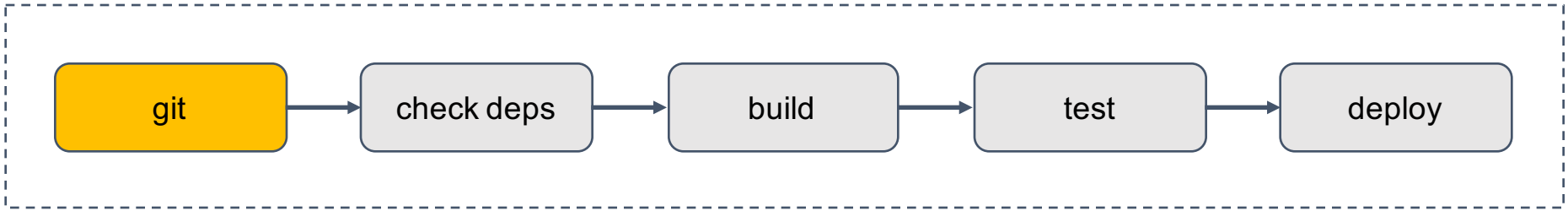
Давайте уменьшать срок жизни уязвимостей в продакшене

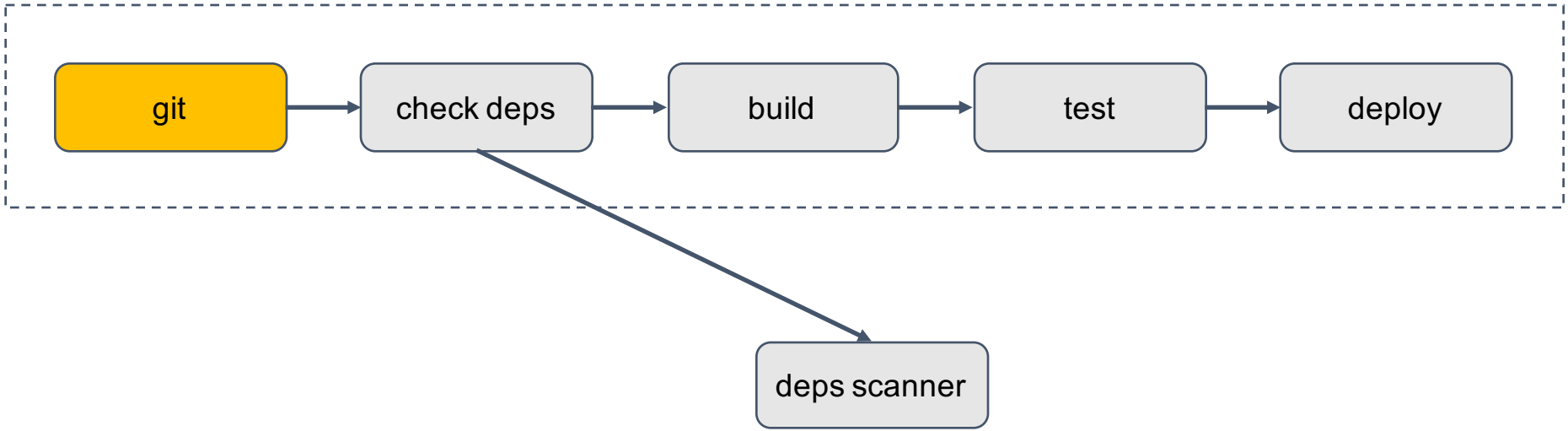
Пришли безопасники с предложением отслеживать уязвимости в используемых библиотеках

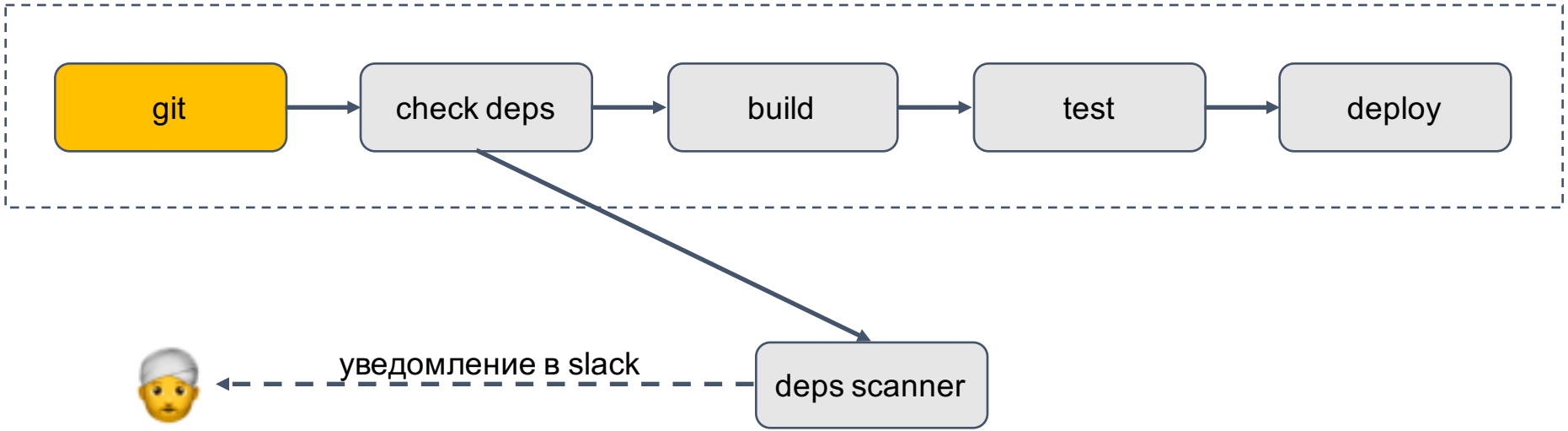


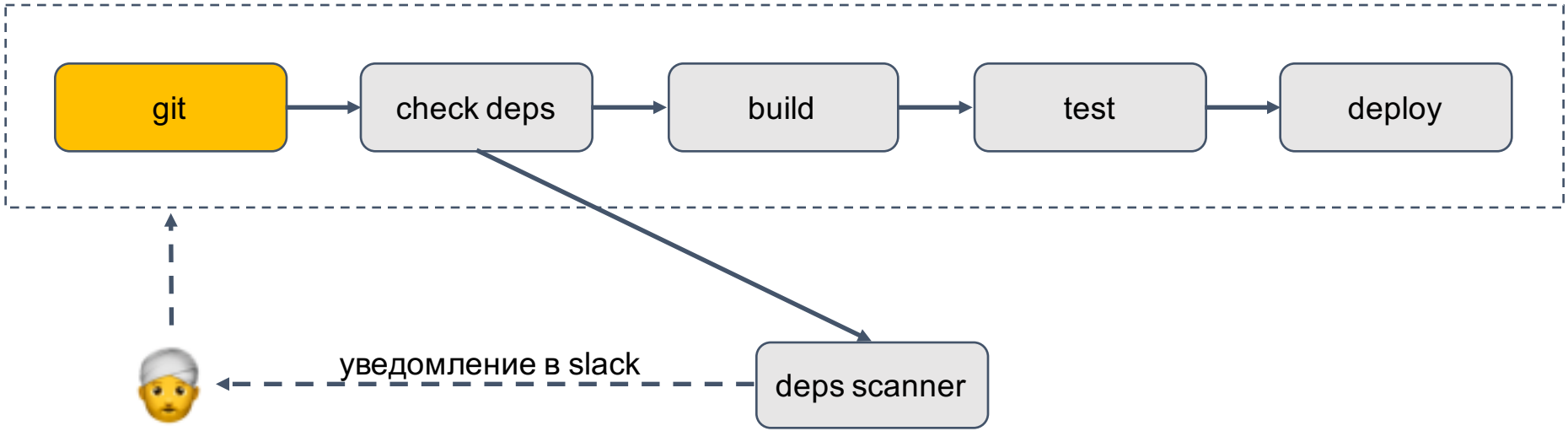


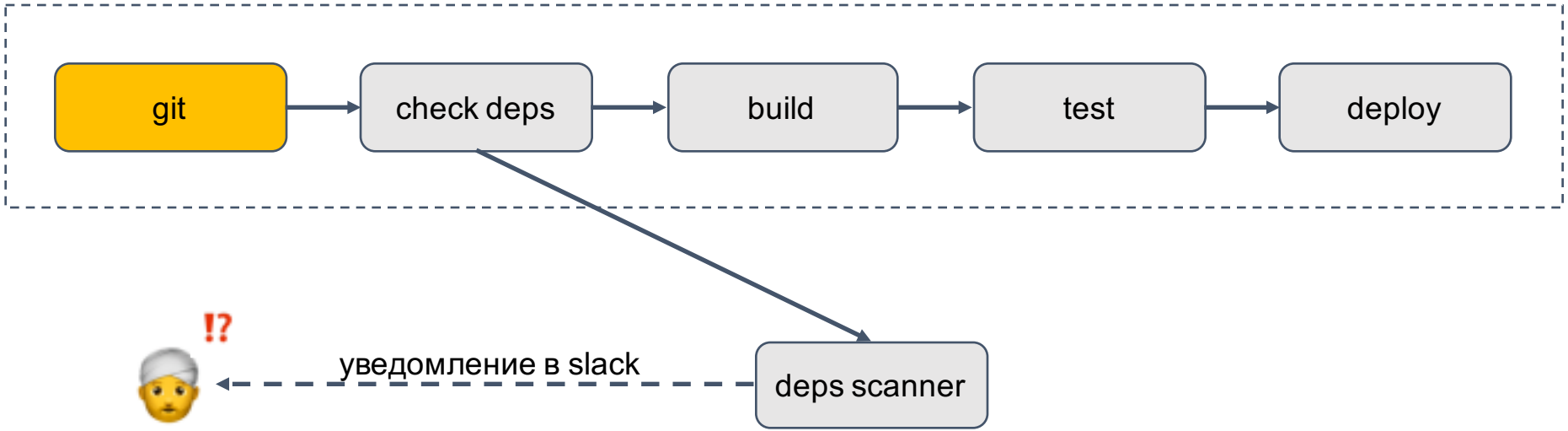

деплой сервиса
понедельник
вторник
среда
t
найдена уязвимость в библиотеке
уязвимость в библиотеке поправлена

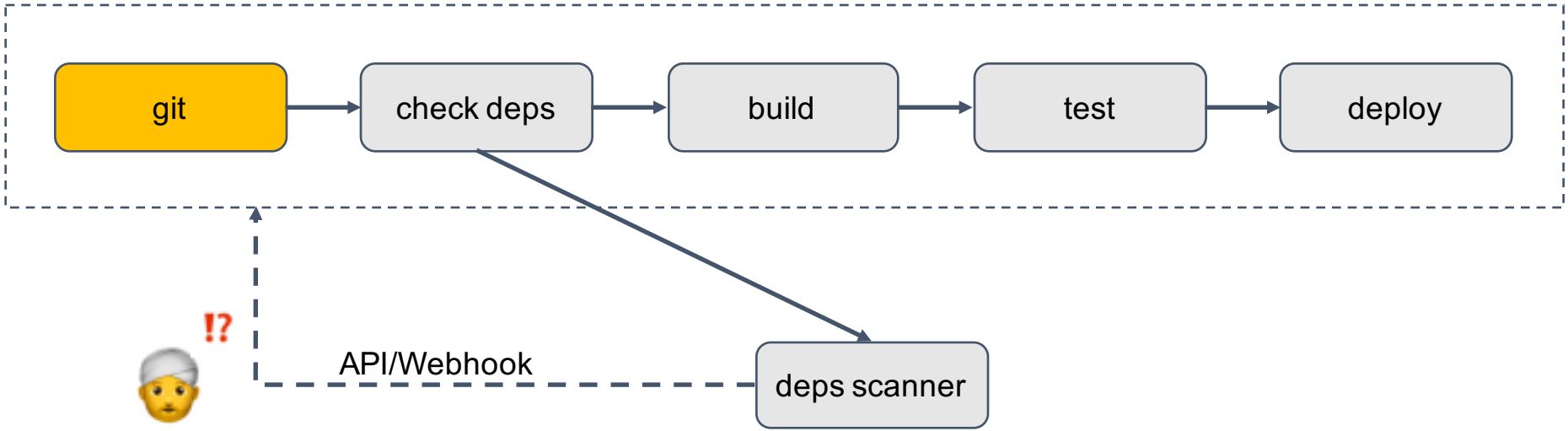




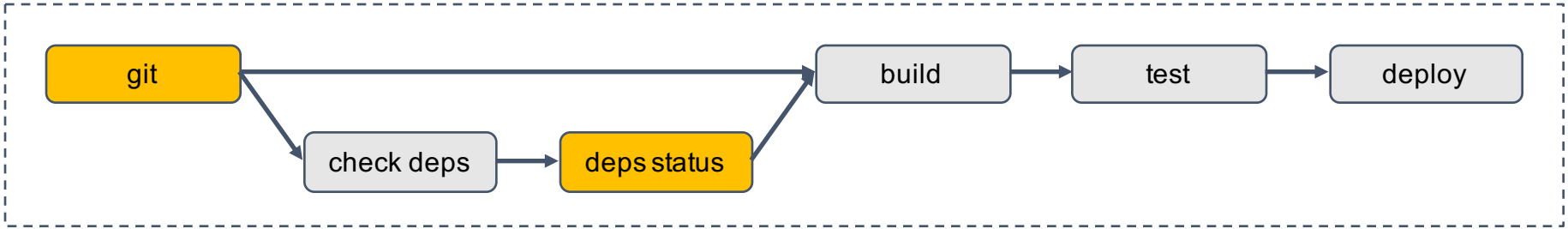








Можно ли сделать более явно?



Чего не хватает?

Наш источник данных — не система контроля версий

Результат проверки нужно куда-то сохранять

Нужно триггерить сборку, когда уязвимость в библиотеке исправили

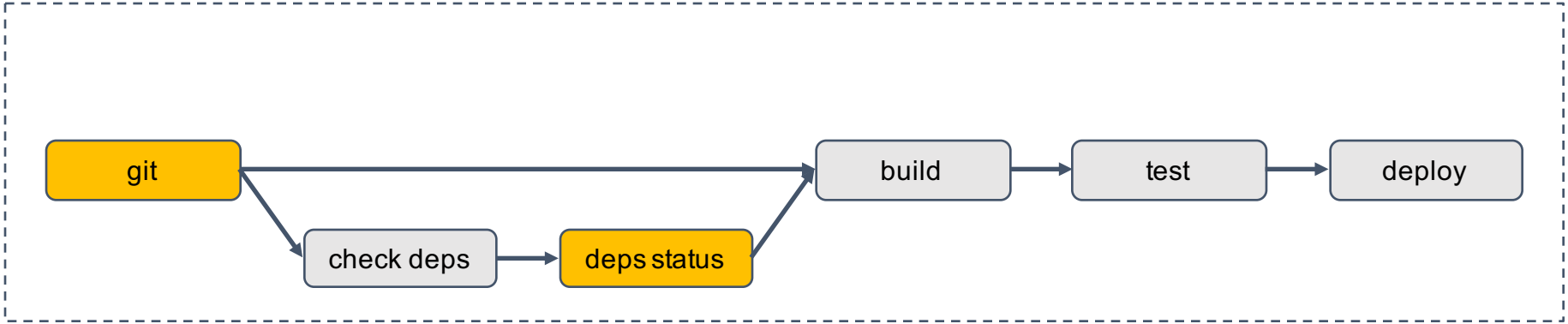
Собираем проблемы

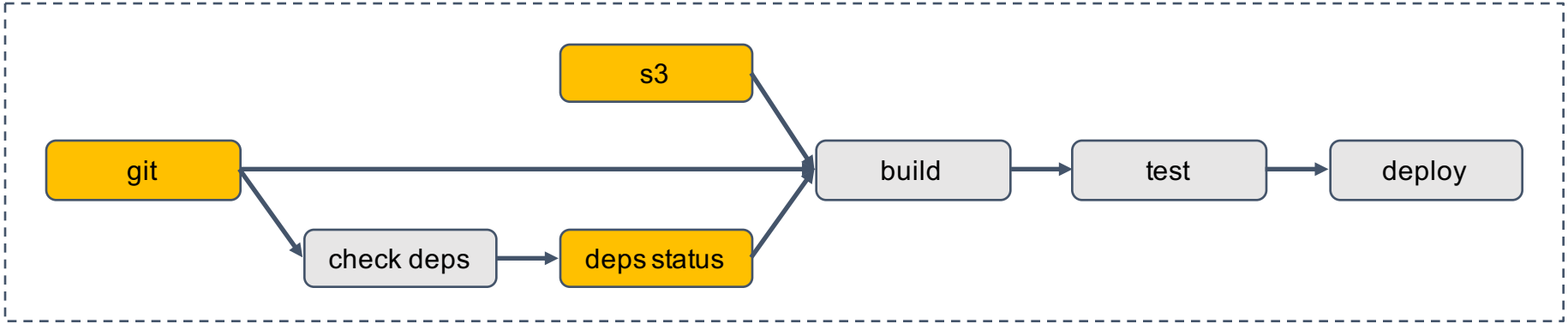
Как CI сервер нас ограничивает?

Расширяемость источников данных

Источник данных — система контроля версий или артефакт другого билда

Собираем сервис рекомендаций. Для сборки нужна натренированная модель, которая лежит в S3. Как быть?





Расширяемость источников данных

Решение: плагин на родном языке сервера

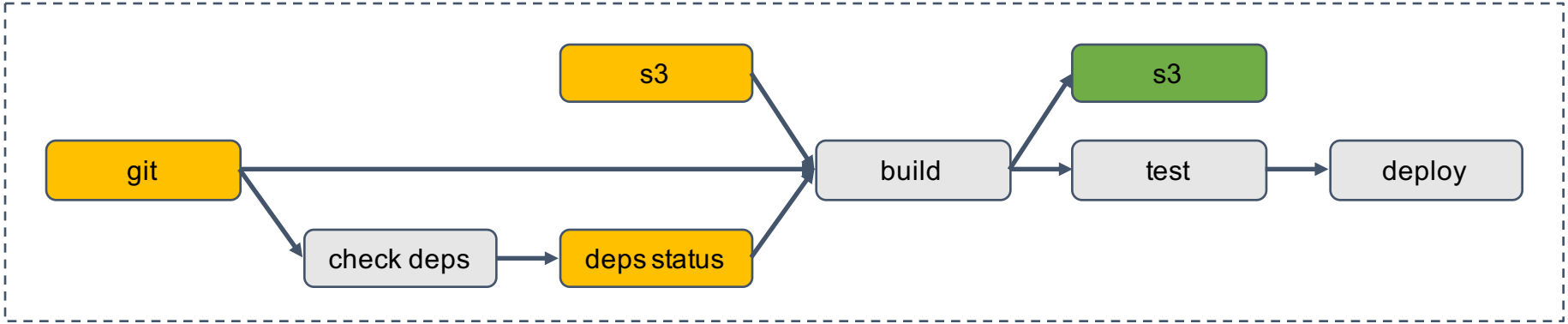
Jenkins и TeamCity позволяют писать плагины на Java

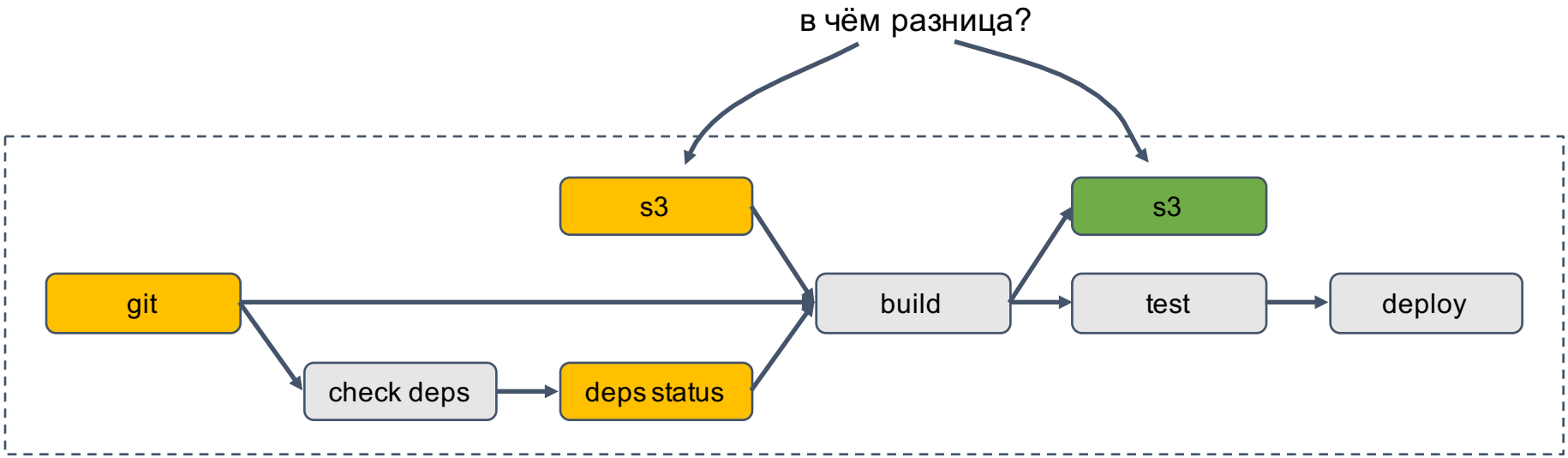
Не горим желанием писать на Java

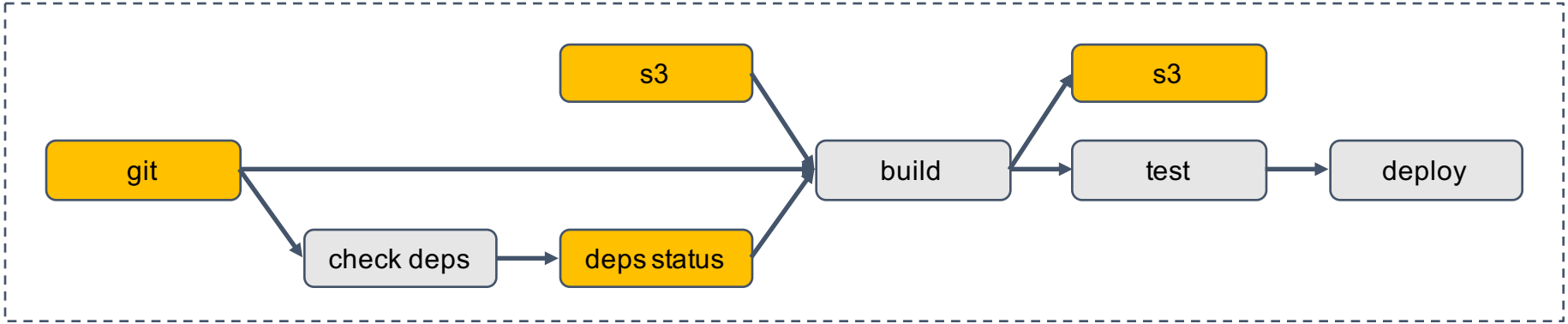
Расширяемость хранилищ артефактов

git pull автоматически, а git push — вручную

Если тренируем модель в шаге сборки — артефакт нужно залить в S3







Расширяемость хранилищ артефактов

Есть ли принципиальная разница между источниками и хранилищами?

Расширяемость хранилищ артефактов

Решение: объединить источники и артефакты в одну сущность?

Было бы круто, но нет

Расширяемость плагинов вообще

Язык реализации плагина зависит от языка сервера

Точки расширения зависят от реализации сервера

Историческая справка

Что такое “Continuous Integration”?

Kent Beck и Ron Jeffries популяризировали термин в рамках методологии Extreme Programming

Не обязательно автоматизировать всё

Что такое “Continuous Integration”?

Список действий

Периодически выполняется

Своевременное получение фидбэка

Каждое действие от чего-то зависит и приносит какой-то результат

Действие == функция?

Ограничения системы плагинов

Решение: функции вместо плагинов

Шаг сборки как функция от входных данных и параметров:

- artifacts = build(inputs, parameters)
- plugin(parameters)
- …

resources = f(resources, parameters)

Ограничения системы плагинов

Решение: контейнеры как функции

Drone и Concourse запускают контейнеры из произвольных Docker-образов

Передача параметров:

- в Drone — переменные окружения DRONE_* и PLUGIN_*
- в Concourse — файлы в директориях и JSON в stdin

Ограничения системы плагинов

Решение: контейнеры для работы с источниками данных/артефактами

Concourse запускает Docker-контейнер, монтирует туда директории и ожидает, что в нём есть три приложения:

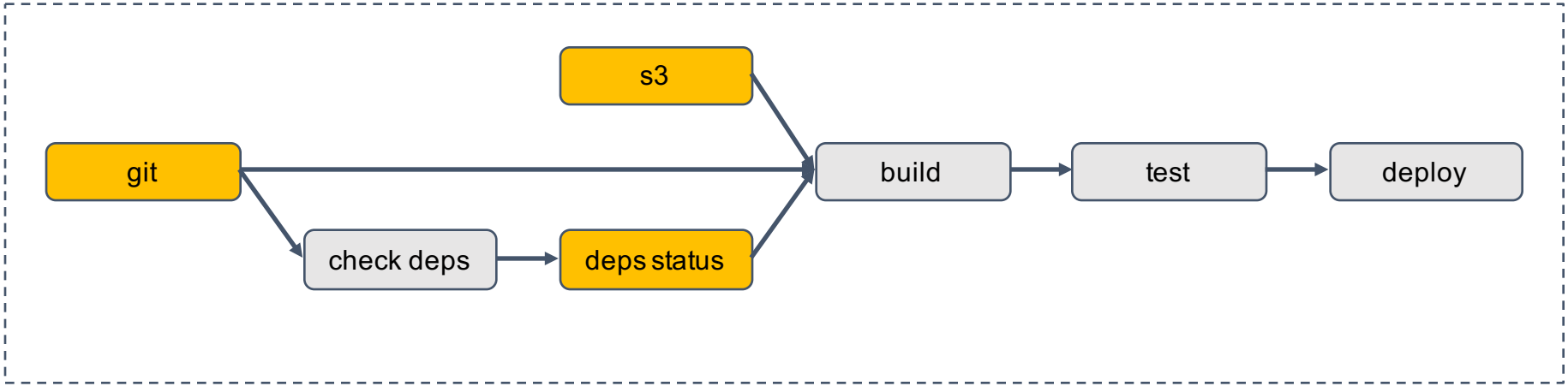
- /check
- /in
- /out

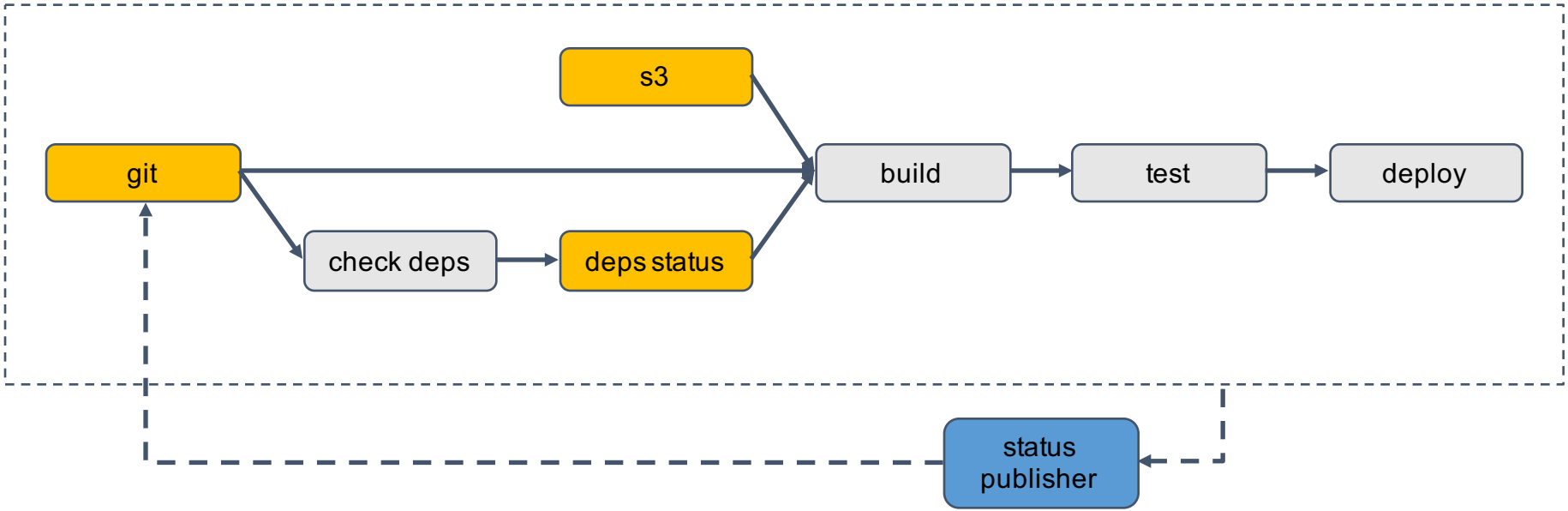
Ок, обратно к проблемам!

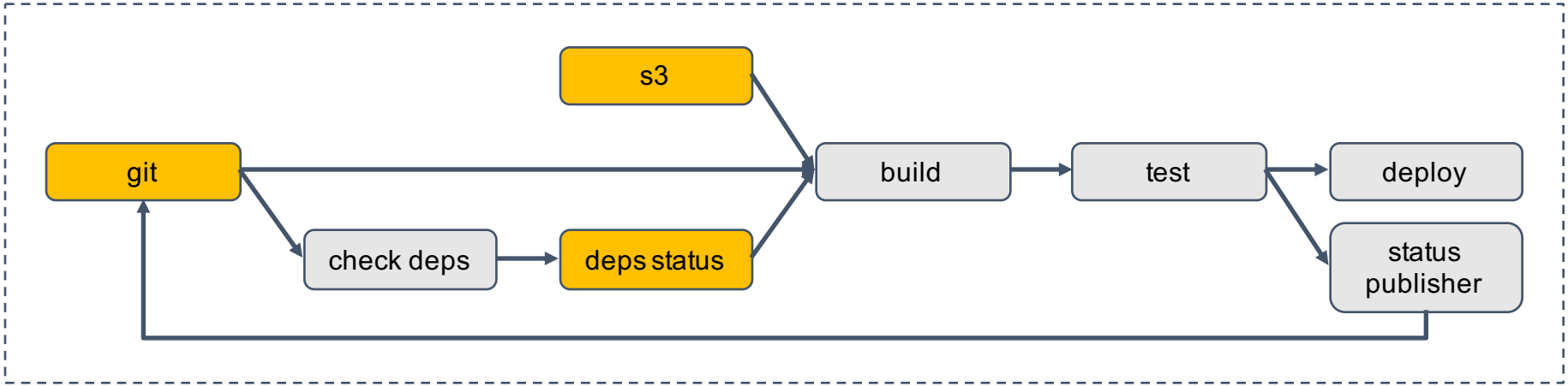
Отсутствие полной картины

Плагины, добавляющие функционал, действуют неочевидно

Не все зависимости явно видны







Отсутствие полной картины

Решение: плагины не нужны, любое действие — job или шаг

Шаг для отправки статуса билда в API Bitbucket Server

Шаг для отправки уведомления в Slack

Хрупкие билд-агенты

Долгоиграющие mutable агенты

Конфигурация привычными способами — puppet/chef/etc

Агенты слабо связаны с приложениями, которые они собирают

Хрупкие билд-агенты

Решение: запуск сборки в контейнерах

CircleCI позволяет указать Docker-образ для job’а

Concourse и Drone — отдельный Docker-образ для каждого шага

Хрупкие билд-агенты

Решение: Infrastructure as Code

CI-сервер, билд-агенты и всё, на чём они запускаются тоже катить через CI

Программирование в textarea

Удобно для быстрого старта

Создает проблемы с расширяемостью

Сложно воспроизвести локально и протестировать

Программирование в textarea

Решение: скрипты сборки рядом с кодом (или в общей репе)

Можно запускать локально, тестировать и ревьюить

Меньше зависимость от сервера

Масштабирование пайплайнов

Сервисов >100 и все нужно собирать

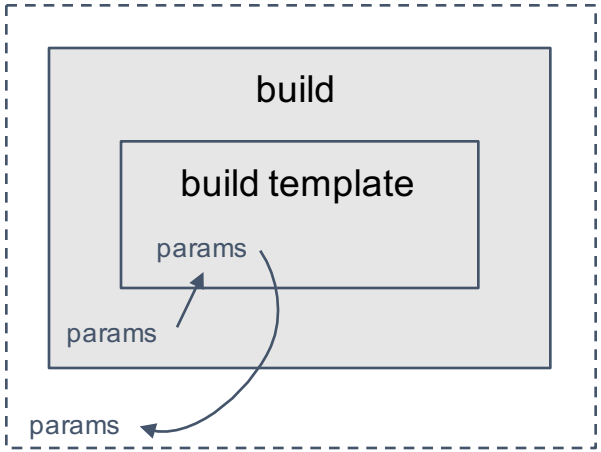
Наследуем билд от шаблона

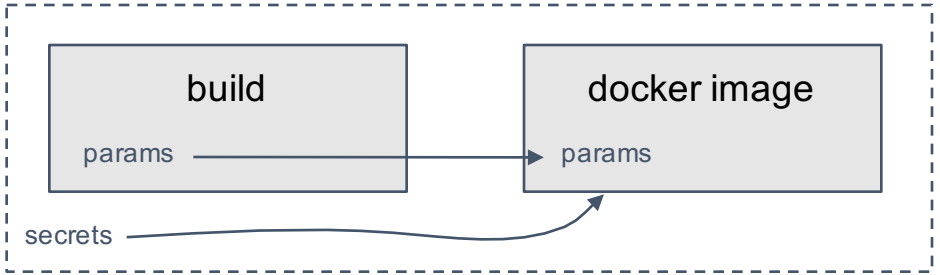
Флажками конфигурируем поведение шаблона

Флажки могут переопределяться на разных уровнях наследования

Надеемся на чудо :)

Наследование vs Композиция





Масштабирование пайплайнов

Решение: композиция вместо наследования

В TeamCity есть meta runners

Concourse всё делает через шаги, а пайплайн может создавать другие пайплайны

Формируем критерии

Как должен выглядеть современный CI-сервер?

Критерии

Расширяемость

Локальная воспроизводимость шагов

Иммутабельность билд-агентов

Прозрачность процессов

Идеальный пайплайн

Какие еще процессы разработки не визуализированы?

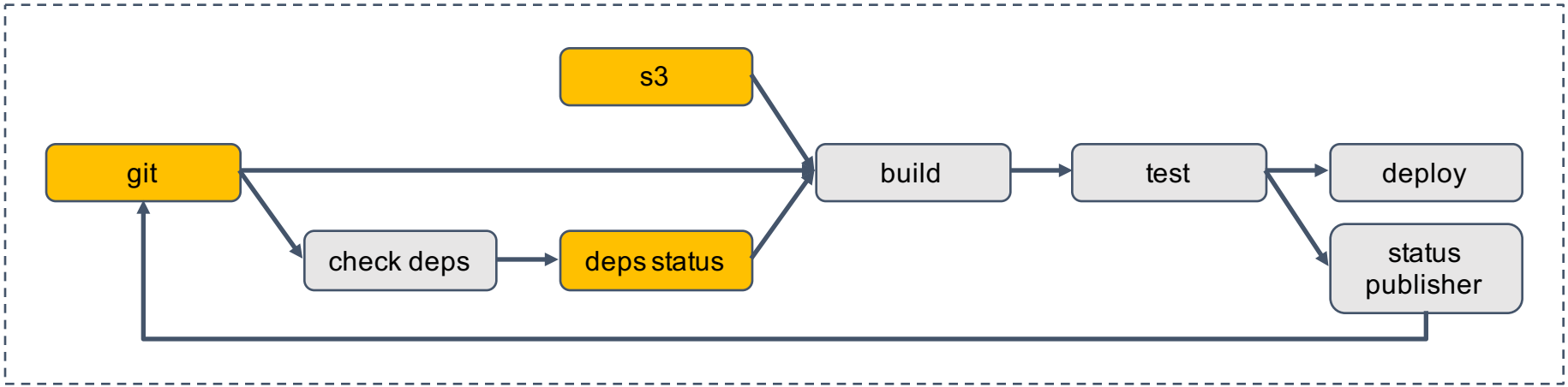
Процессы с обратной связью

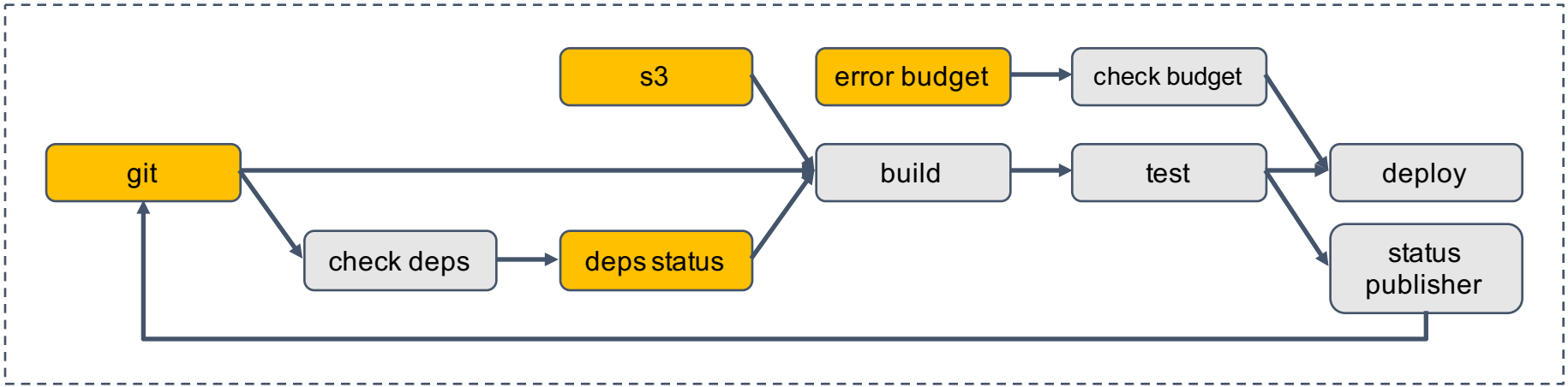
Результаты A/B-тестов как ресурс

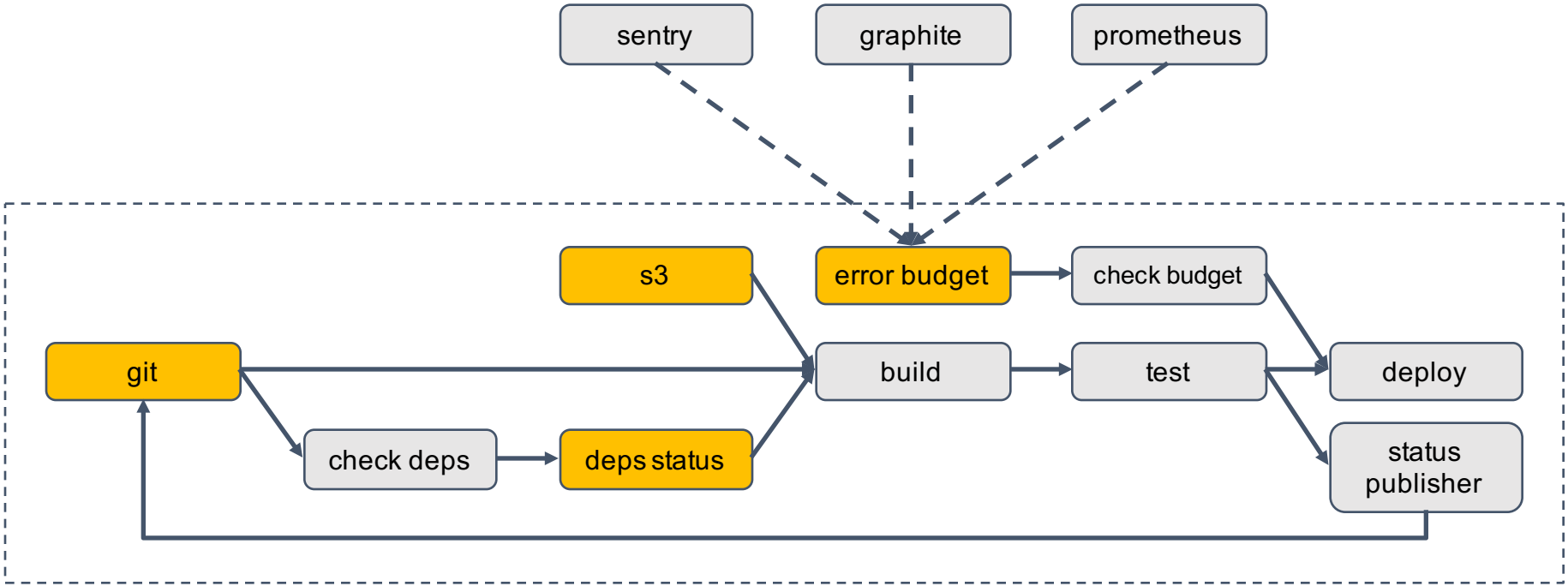
Результаты нагрузочного тестирования как ресурс

Статус canary deployment как ресурс

Error budget как ресурс







И даже продуктовая разработка

Релизы в app-сторы как шаг пайплайна

Версия дизайн-системы как ресурс

Пользовательские истории от аналитиков как ресурс

Ссылки

https://drone.io

https://concourse-ci.org

https://github.com/lambci/lambci

https://github.com/JetBrains/teamcity-s3-artifact-storage-plugin

https://greenkeeper.io

https://www.amazon.com/Extreme-Programming-Explained-Embrace-Change/dp/0201616416/

Спасибо! Вопросы?

Илья Сауленко, Авито

@mynameiswhm

hi@mynameiswhm.ru