KASPERSKY lab

ОПЫТ УПРАВЛЕНИЯ 3RD-PARTY БИБЛИОТЕКАМИ

Alexey Kutumov

Lead software engineer at Kaspersky Lab

ЧТО ТАКОЕ СТОРОННИЙ КОД

Сторонний код – это переиспользуемый программный компонент, разработанный для распространения компанией, не являющейся разработчиком платформы

Сторонний код – это переиспользуемый программный компонент, разработанный для распространения компанией, не являющейся разработчиком платформы

СПОСОБЫ ИСПОЛЬЗОВАНИЯ

- В скомпилированном виде
  - Исполняемый модуль
  - Библиотека
- В виде исходных кодов

- В C++ проектах обычно смешанный вариант

ЖИЗНЕННЫЙ ЦИКЛ СТОРОННЕГО КОДА

ЖИЗНЕННЫЙ ЦИКЛ

- Анализ
- Первоначальное внедрение
- Релиз
- Вывод из эксплуатации

ЖИЗНЕННЫЙ ЦИКЛ: АНАЛИЗ

- Решает ли поставленную задачу?
- Какую лицензию компоненты использовать?
- Где хранить исходные коды?
- Где хранить скомпилированные модули?
- В каких конфигурациях собирать?
- Требуется ли модификация компоненты?
- Как и когда обновляется компонента?
- **Прочие особенности**

ЖИЗНЕННЫЙ ЦИКЛ: ВНЕДРЕНИЕ

- Подключение к проекту
  - Настройка механизмов получения компоненты
- Тестирование
  - Запуск тестов сторонней библиотеки
  - Написание своих тестов для сторонней библиотеки
- Реализация бизнес-логики
  - Фасад для сторонней компоненты

CONAN.io

C/C++ Package Manager

KASPERSKY lab

ЖИЗНЕННЫЙ ЦИКЛ: РЕЛИЗ

- Поддержка существующего функционала
- Добавление нового функционала

- Обновление версии сторонней компоненты

ЖИЗНЕННЫЙ ЦИКЛ: РЕЛИЗ

- Поддержка существующего функционала
- Добавление нового функционала

- Обновление версии сторонней компоненты
  - Поддержка существующего функционала
  - Добавление нового функционала

ЖИЗНЕННЫЙ ЦИКЛ: ВЫВОД ИЗ ЭКСПЛУАТАЦИИ

- Окончание проекта
- Несоответствие новым требованиям
  - Несовместимость лицензии
  - Несовместимость функционала
- Высокая стоимость поддержки

СТОРОННИЙ VS СВОЙ КОД

ТРЕБОВАНИЯ

- Специфичные флаги сборки
- Различные платформы и тулчейны
- Различные конфигурации сборки
- Скорость сборки
- Воспроизводимость и контроль процесса сборки
- Время обновления версии

ДОПОЛНИТЕЛЬНЫЕ ТРЕБОВАНИЯ

- Monorepo

ДОПОЛНИТЕЛЬНЫЕ ТРЕБОВАНИЯ

- Monorepo
- GTI

ПРОБЛЕМЫ

- Долго
  - Много лишних действий
- Дорого
  - Нетривиальная поддержка
- Неконсистентно
  - Зоопарк версий
- Непредсказуемо
  - Баги и особенности систем сборки

ПРИМЕР: OPENSSL

- conanfile для openssl 1.1.1a
  - Скачивает архив + зависимости
  - Настраивает параметры сборки
  - Запускает perl Configure
  - Патчит Makefile
  - Собирает
  - Пакует

ПРИМЕР: OPENSSL

- conanfile для openssl 1.1.1a
  - Скачивает архив + зависимости
  - Настраивает параметры сборки
  - Запускает perl Configure
  - Патчит Makefile
  - Собирает
  - Пакует

СТОРОННИЙ VS СВОЙ КОД

- Сторонний код

```
if(NOT WIN32)
  ./configure ${unix_args}
  make -j ${cpu_count}
  make install
else()
  msbuild lib.sln ${win_args}
endif()
```

- Свой код

```
kl_create_exe(my_app main.cpp)
```

СТОРОННИЙ VS СВОЙ КОД

- Сторонний код
  - Произвольные билд системы
  - Многообразие типов сборки
  - Нужно внедрять в процессы сборки
  - Гранулярность на уровне пакета
  - Специфика - руками
- Свой код
  - Фиксированная билд система
  - Единственный* вариант сборки
  - Интегрирован с рождения
  - Гранулярность на уровне файла
  - Специфика – в DSL

СТОРОННИЙ КАК СВОЙ КОД

- Сторонний код

```
kl_create_lib(ssl ${ssl_src})
```

- Свой код

```
kl_create_exe(my_app main.cpp)
```

САМОЕ ГЛАВНОЕ ТРЕБОВАНИЕ

Потому что это весело :)

МИГРАЦИЯ СИСТЕМЫ СБОРКИ

МИГРАЦИЯ

- Как мигрировать?
- Как верифицировать?
- Как понять что стало лучше чем было?

МИГРАЦИЯ

- Как мигрировать?
  - автоматически
- Как верифицировать?
  - Тесты, тесты и еще раз тесты
- Как понять что стало лучше чем было?
  - Уменьшилось ли время сборки?
  - Уменьшилось ли время поддержки?
  - Удобней ли стало пользоваться клиентам?

КАК МИГРИРОВАТЬ

- Анализируем оригинальную систему сборки
- Получаем граф целей для сборки
- Генерируем скрипты для сборки

АНАЛИЗ СИСТЕМЫ СБОРКИ

- configure – выясняем возможности платформы
  - Получаем build flags
- build – собираем компоненту исходя из возможностей платформы
  - `sources,` `flags -> object file`
  - `object file*,` `flags -> static library`
  - `object file*, static library*,` `flags -> shared library`

АНАЛИЗ СИСТЕМЫ СБОРКИ

- Нужно получить полный лог сборки
  - `make VERBOSE=1 V=1 --print-directories`
  - parse *.tlog (msbuild only)
  - Intercept toolchain (compiler, linker)

ПОСТРОЕНИЕ ГРАФА ЦЕЛЕЙ

- Компиляция, препроцессирование
- Сборка статической библиотеки
- Сборка динамической библиотеки

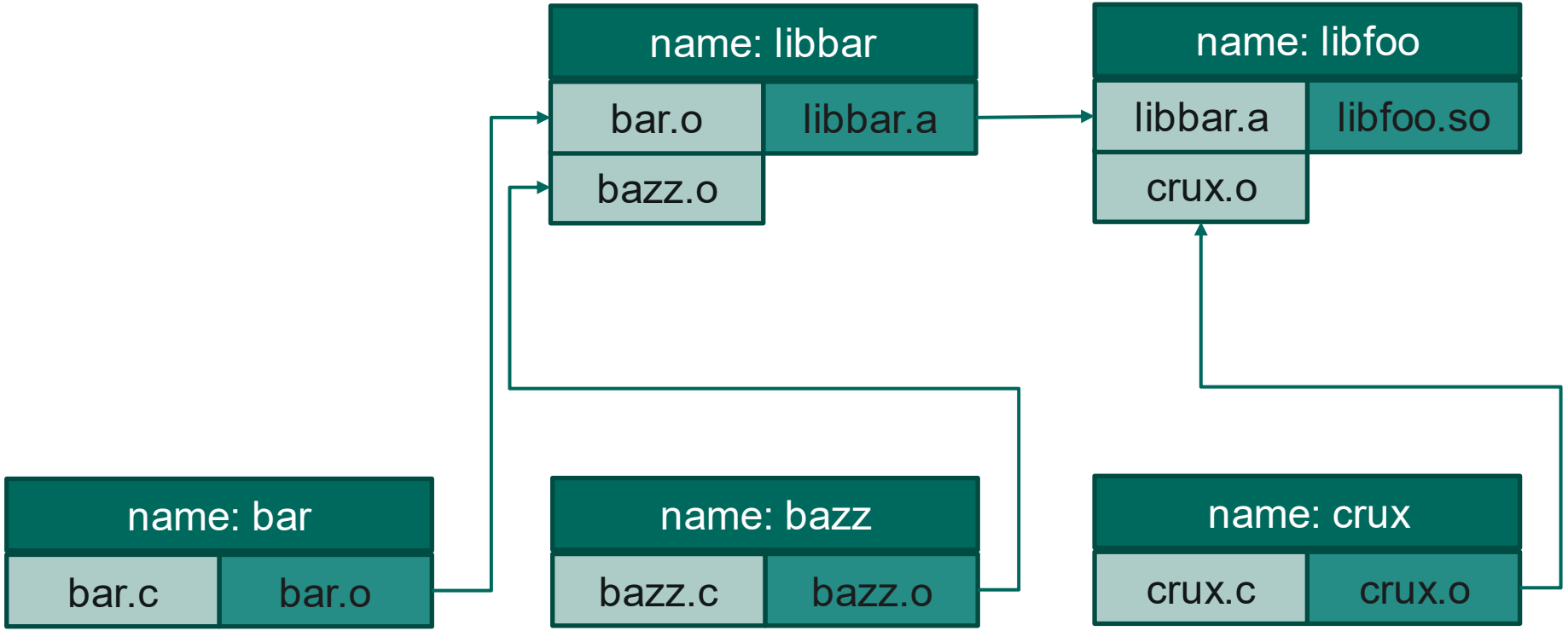
ПОСТРОЕНИЕ ГРАФА ЦЕЛЕЙ

- Компиляция, препроцессирование
  - `g++ main.cpp -c -o main.o -Iinclude -DNDEBUG`
- Сборка статической библиотеки
  - `ar cru libfoo.a src1.0 src2.0`
- Сборка динамической библиотеки
  - `ld -o libbar.so src1.0 -lfoo`

ПОСТРОЕНИЕ ГРАФА ЦЕЛЕЙ

- Получаем список объектов
  - **Type** – тип объекта (компиляция, линковка)
  - **Inputs** – входные артефакты
  - **Outputs** – результирующие артефакты
  - **Params** – параметры команды
    - Working dir
    - Preprocessor definitions
    - Include paths
    - Toolchain-specific parameters

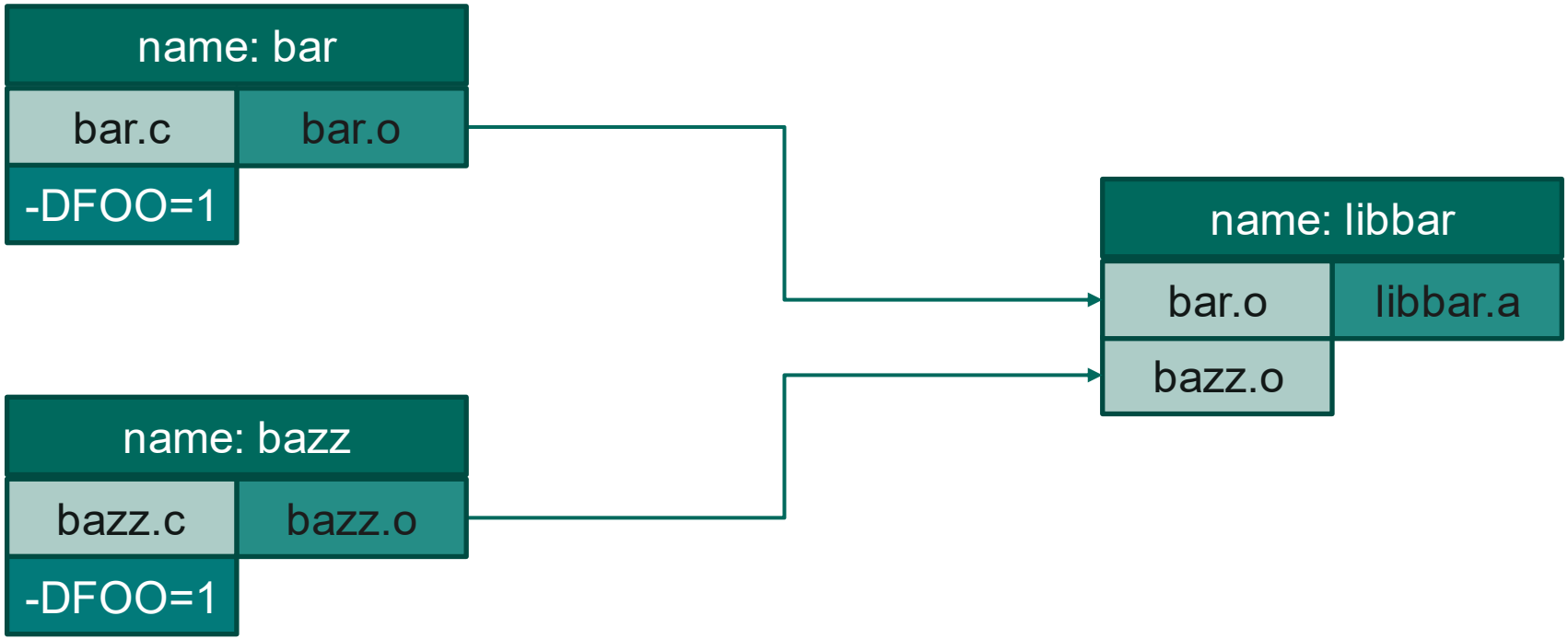
ПОСТРОЕНИЕ ГРАФА ЦЕЛЕЙ



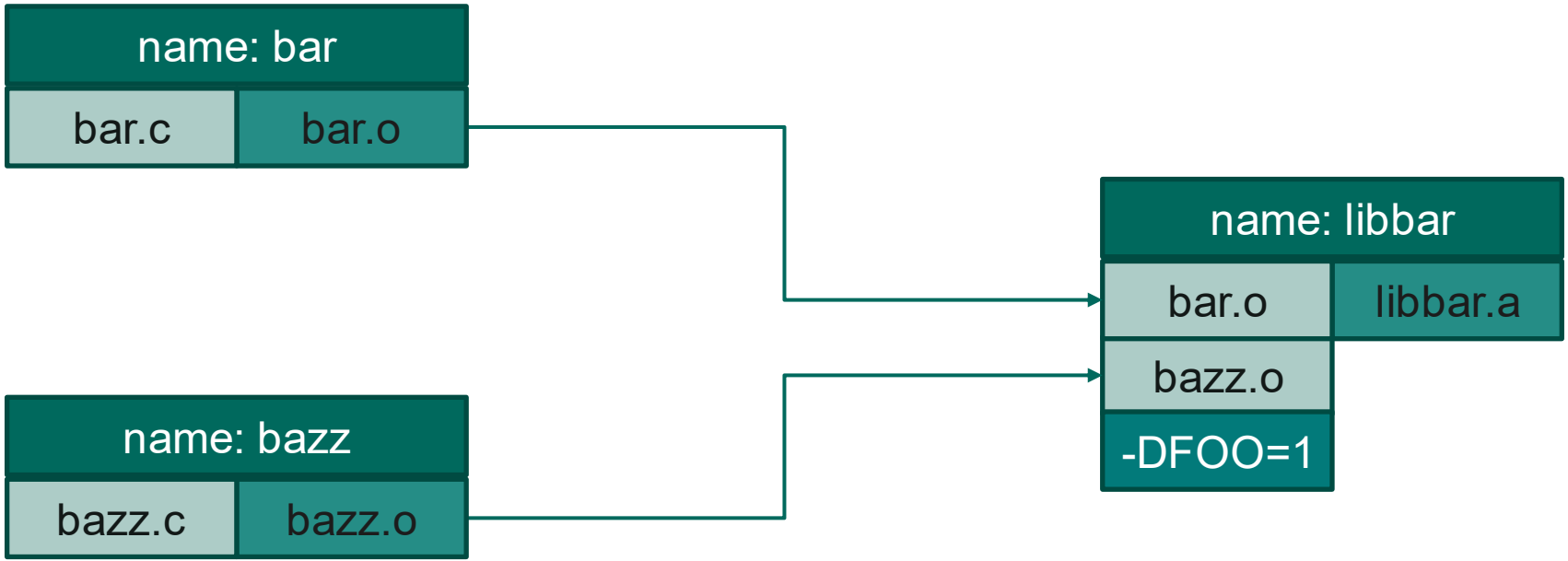
ОПТИМИЗАЦИЯ ГРАФА ЦЕЛЕЙ

- Поднимаем общие опции наверх
  - Пробегаем по всем «детям» и собираем одинаковые опции
  - Убираем их у «детей» и ставим «родителю»

ОПТИМИЗАЦИЯ ГРАФА ЦЕЛЕЙ



ОПТИМИЗАЦИЯ ГРАФА ЦЕЛЕЙ



ОПТИМИЗАЦИЯ ГРАФА ЦЕЛЕЙ

- Прячем дефолтные опции тулчейна

```
cl /c /ID:\src\shared /ZI /nologo /W3 /WX- /Od /Oy- /D WIN32
/D _DEBUG /D _CONSOLE /D _UNICODE /D UNICODE /D FOO=1 /Gm
/EHsc /RTC1 /MTd /GS /fp:precise /Zc:wchar_t /Zc:forScope
/Zc:inline /Fo"DEBUG\\" /Fd"DEBUG\VC140.PDB" /Gd /TP
/analyze- foo.cpp
```

ОПТИМИЗАЦИЯ ГРАФА ЦЕЛЕЙ

- Прячем дефолтные опции тулчейна

```
kl_create_exe(my_app main.cpp)

kl_create_exe(my_lib STATIC lib.cpp)
```

ОПТИМИЗАЦИЯ ГРАФА ЦЕЛЕЙ

- Прячем дефолтные опции тулчейна

```
cl /c /ID:\src\shared /ZI /nologo /W3 /WX- /Od /Oy- /D WIN32
/D _DEBUG /D _CONSOLE /D _UNICODE /D UNICODE /D FOO=1 /Gm
/EHsc /RTC1 /MTd /GS /fp:precise /Zc:wchar_t /Zc:forScope
/Zc:inline /Fo"DEBUG\\" /Fd"DEBUG\VC140.PDB" /Gd /TP
/analyze- foo.cpp
```

ОПТИМИЗАЦИЯ ГРАФА ЦЕЛЕЙ

- Прячем дефолтные опции тулчейна

```
cl     /ID:\src\shared
                                                    /D FOO=1

            foo.cpp
```

СЛИЯНИЕ ГРАФОВ ДЛЯ РАЗНЫХ ПЛАТФОРМ

- Генерируем граф для каждой платформы
  - Сливаем одинаковые узлы в один
  - Различные узлы остаются как есть

ВРЕМЯ СБОРКИ OPENSSL

- Как сторонний код
  - cmake generate: 6 секунд
  - cmake build: 764 секунды
  - total time: 13 минут
- Как свой код
  - cmake generate: 52 секунды
  - cmake build: 111 секунд
  - total time: 3 минуты

LET'S TALK?

alexey.kutumov@gmail.com

KASPERSKY lab