EREMEX

ИННОВАЦИОННЫЙ ПОДХОД К РАЗРАБОТКЕ ЭЛЕКТРОНИКИ

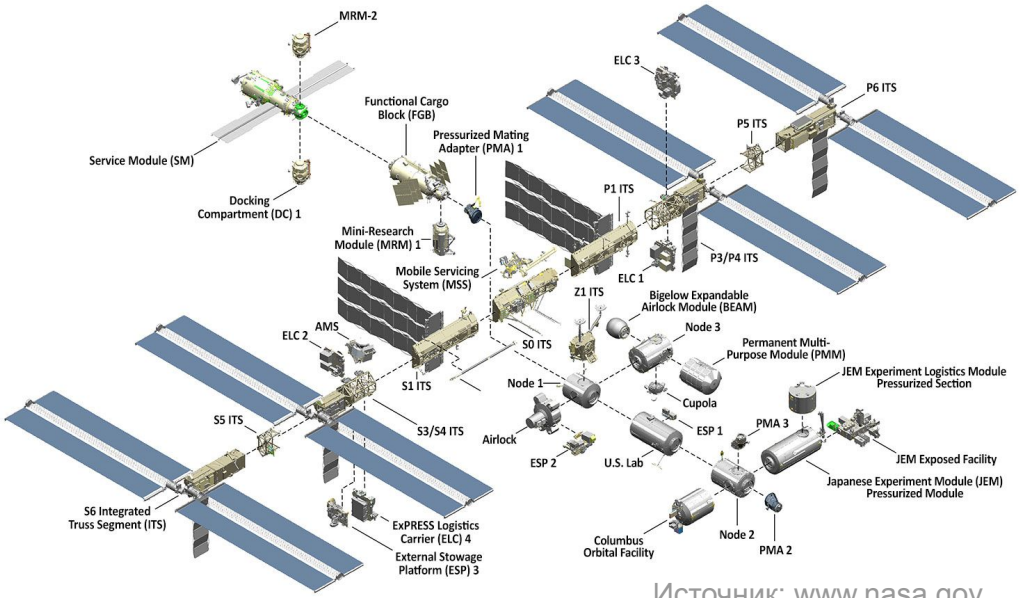
Статическое внедрение зависимостей в C/C++

Дмитрий Алексеев, программист ООО “Эремекс”
alekseev.d@prosoft.ru

WWW.EREMEX.RU

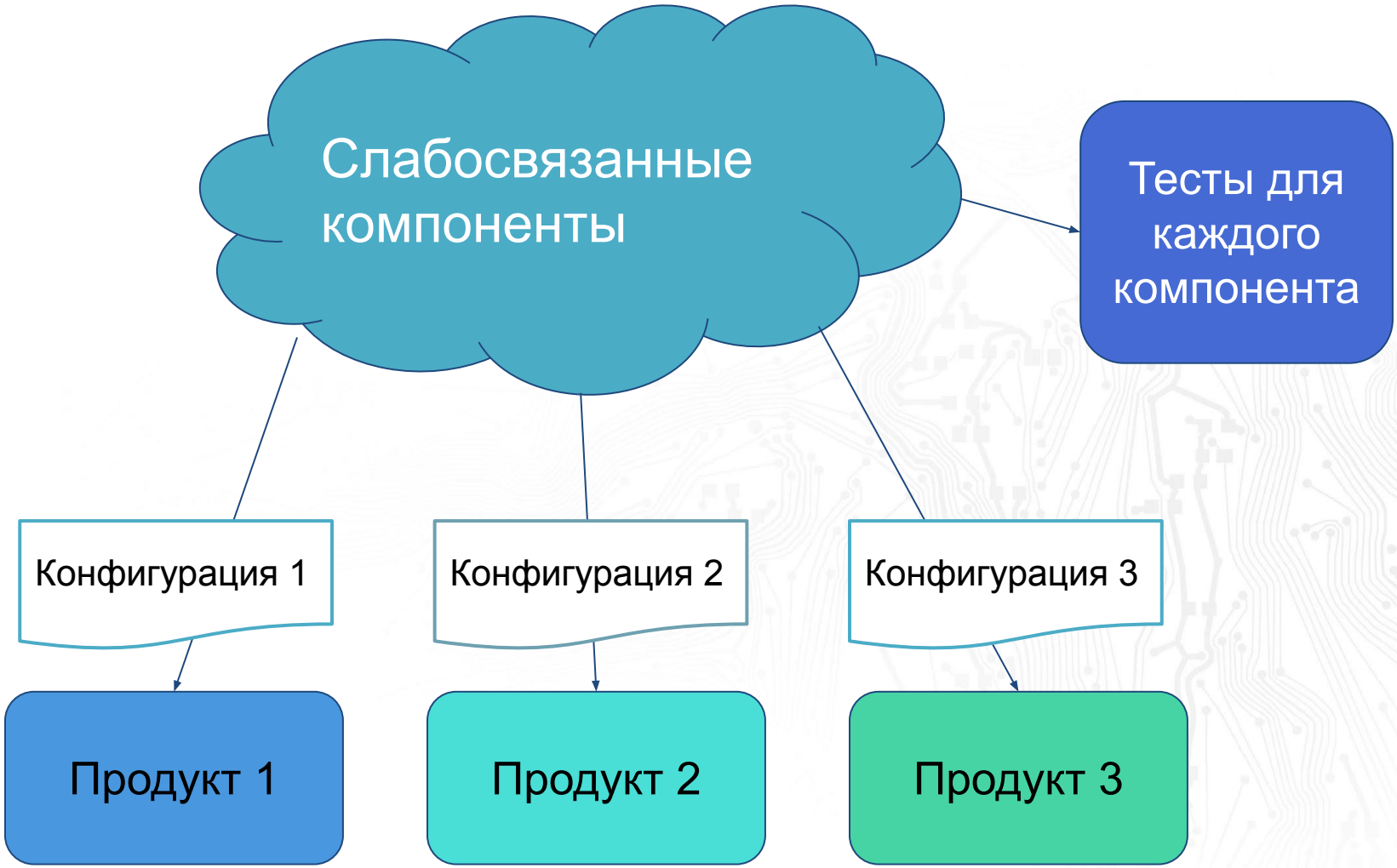
Требования больших проектов

- Модульность
  - Разделение по функциональности
  - Упрощение интеграции
- Слабая связность
- Повторное использование кода
- Конфигурируемость
- Независимая параллельная разработка



Источник: www.nasa.gov

Требования больших проектов



Компоненты в ПО

- Компонент - часть программы, взаимодействующая с другими компонентами через определённый интерфейс

- Примеры:
  - Модули Pascal (static)
  - MEF .NET (runtime)

Проблемы использования компонентов в C/C++

- Нет концепции компонента
- Нет сопоставления исходника заголовочному файлу
- Децентрализованное управление конфигурацией с помощью макросов
- Сложно использовать части одних проектов в других:
  - Кросс-зависимости
  - Согласование исходников и системы сборки вручную

Проблемы использования компонентов в C/C++

- Компоненты в бинарном виде (библиотеки)
  - Влекут накладные расходы времени выполнения
  - Усложняют внесение изменений
  - Требуют сопровождения сборки

- Компоненты в исходных текстах требуют:
  - Указание информации о принадлежности файла компоненту
  - Обработчик информации о компонентах

Концепция метаданных

- Метаданные - декларативная информация о коде
- Варианты расположения метаданных для описания компонентов:
  - В отдельных файлах
  - В исходных текстах

FX-DJ: Описание метаданных

- Записываются в тексте программы внутри макроса FX_METADATA(...)
- При компиляции этот макрос не используется
- Формат описания - словарь, подмножество YAML
- Каждый файл имеет метку принадлежности к компоненту:

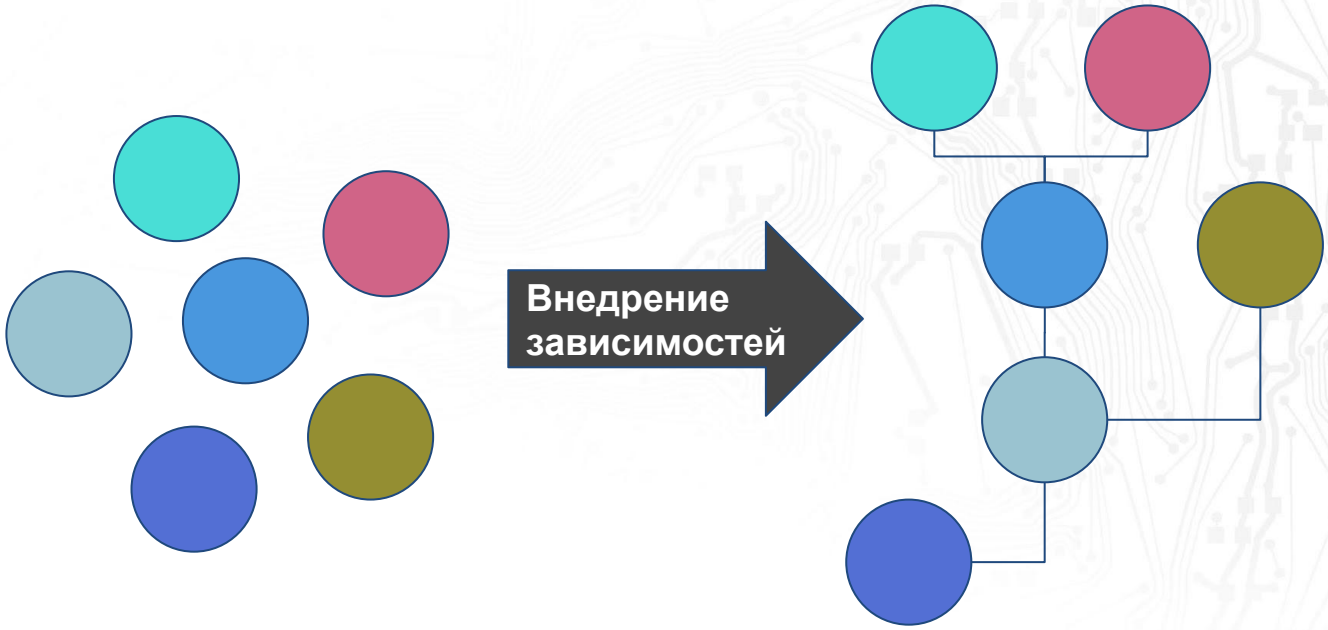
```
FX_METADATA(({ interface: [IF_NAME, COMP_NAME ]))
FX_METADATA(({ implementation: [IF_NAME, COMP_NAME}))
```

- Для включения компонента используется макрос:

```
#include FX_INTERFACE(IF_NAME)
```

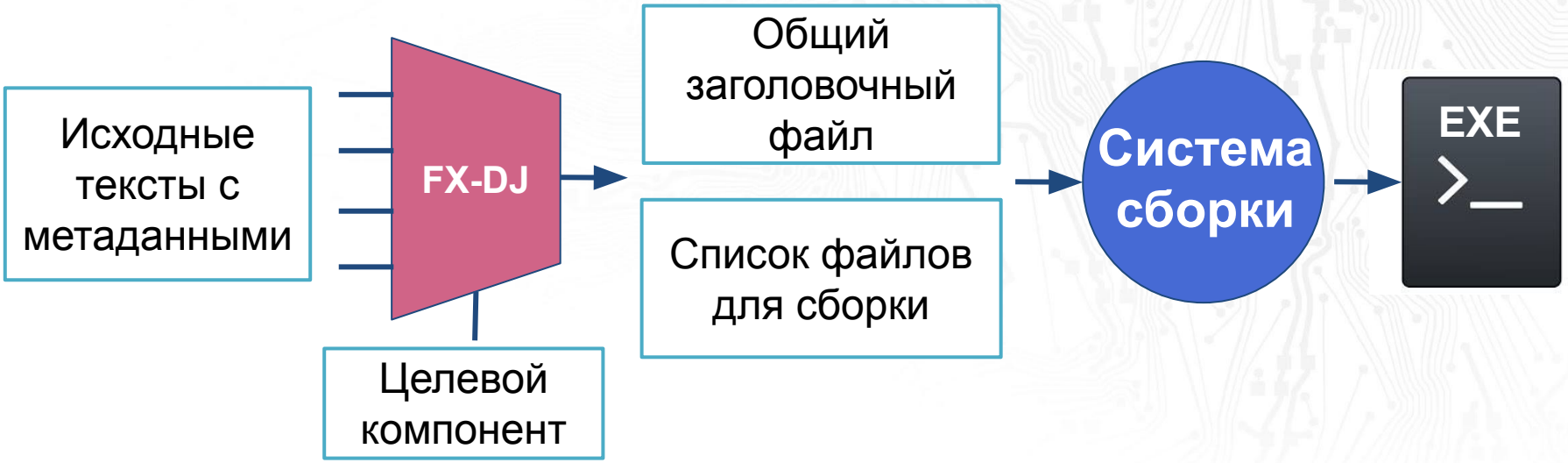
FX-DJ: Обработка метаданных

- Скрипт Python, ~ 700 LoC
- Включение в сборку компонентов, а не файлов
- Построение графа зависимостей по исходникам
- Внедрение зависимостей



FX-DJ: Сборка

- Общий заголовочный файл сборки:
  - Генерируется инструментом FX-DJ
  - Отображает имена файлов на имена компонентов
  - Принудительно включается во все компилируемые файлы



Описание интерфейса и реализации

ketchup.h

```
FX_METADATA(({ interface: [FLAVOR, KETCHUP] }))
```

ketchup.c

```
#include FX_INTERFACE(FLAVOR)
                                  <interface> <implementation>
FX_METADATA(({ implementation: [FLAVOR, KETCHUP] }))

int put_flavor(flavor_t* obj)
{
     obj = ketchup_alloc();
}
```

sausage.h

```
FX_METADATA(({ interface: [FILLING, SAUSAGE] }))
```

sausage.c

```
#include FX_INTERFACE(FILLING)

FX_METADATA(({ implementation: [FILLING, SAUSAGE] }))

int put_filling(filling_t* obj)
{
     obj = sausage;
}
```

Включение интерфейса

hot_dog.h

```
FX_METADATA(({ interface: [HOT_DOG, VER1] }))
```

hot_dog.c

```
#include FX_INTERFACE(FLAVOR)
#include FX_INTERFACE(FILLING)
#include FX_INTERFACE(HOT_DOG)

FX_METADATA(({ implementation: [HOT_DOG, VER1] }))

hotdog_t hdog;

int main(void)
{
     hot_dog_cut_bread();
     put_filling(&hdog->fill);
     put_flavor(&hdog->flav);
     return 0;
}
```

Генерация общего хедера и списка файлов для сборки

```
> fx-dj.py -p c:\src_path -t HOT_DOG:VER1 -c common.h -o input.txt
```

common.h

```
#define FX_METADATA(x)
#define __FX_INTERFACE(i) i
#define FX_INTERFACE(i) __FX_INTERFACE(I_##i)

#define I_HOT_DOG_VER1 "c:\src_path\hotdog.h"
#define I_FLAVOR_KETCHUP "c:\src_path\ketchup.h"
#define I_FILLING_SAUSAGE "c:\src_path\sausage.h"
#define I_HOT_DOG I_HOT_DOG_VER1
#define I_FLAVOR I_FLAVOR_KETCHUP
#define I_FILLING I_FILLING_SAUSAGE
```

input.txt

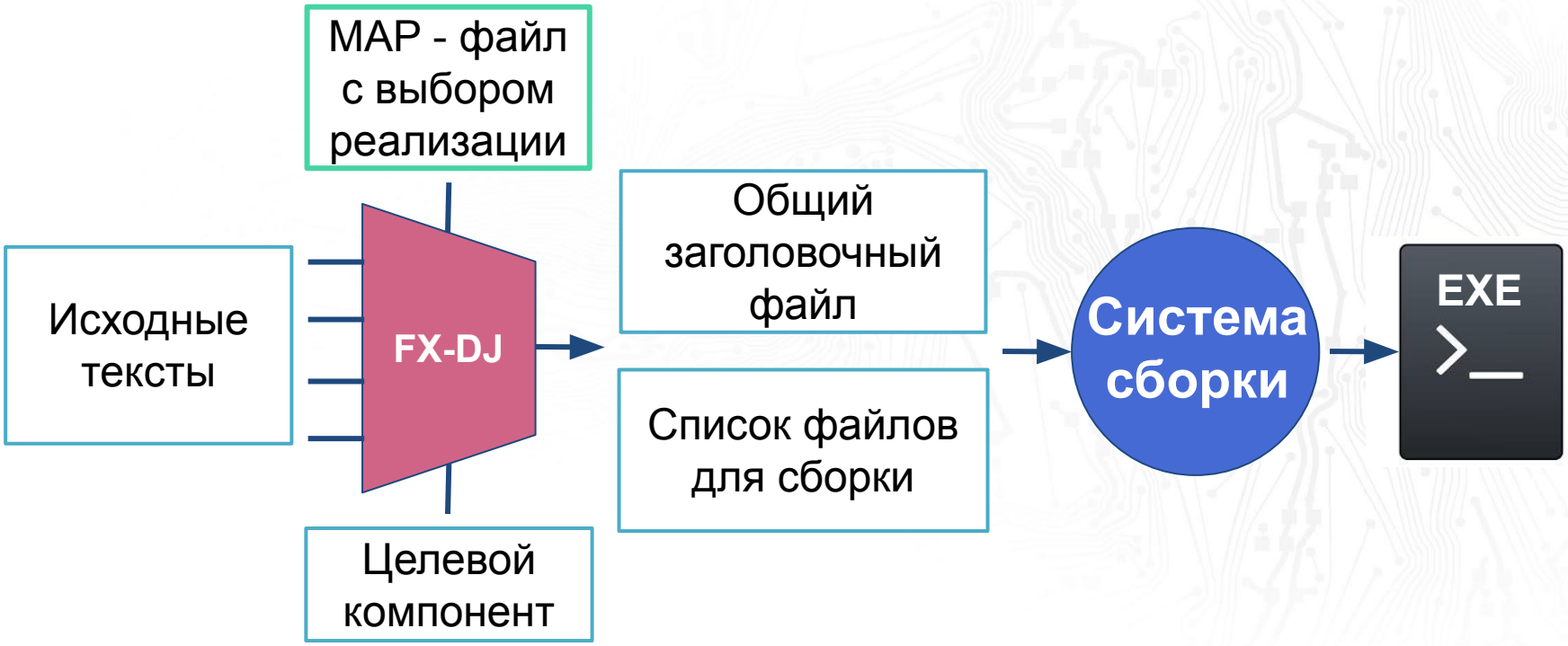
```
c:\src_path\hot_dog.c
c:\src_path\ketchup.c
c:\src_path\sausage.c
```

Извлечение зависимостей

- Обработка препроцессором исходных текстов начиная с целевого компонента
- Импорты из общего заголовочного файла
- Поиск меток `interface` в выводе препроцессора

Внедрение зависимостей

- Наличие более одной реализации интерфейса
- Выбор реализации с помощью внешней информации
- Выполняется во время сборки



Внедрение зависимостей: несколько реализаций интерфейса

dj.map

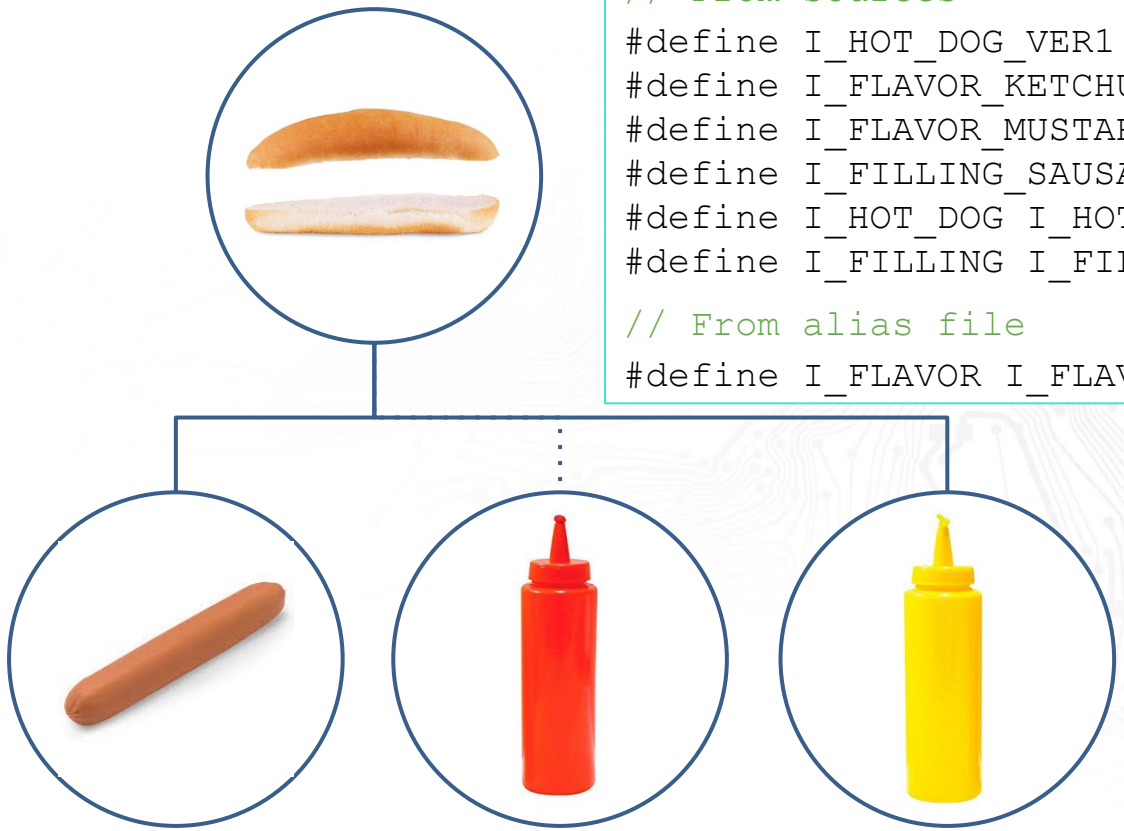
```
FLAVOR = MUSTARD
```

common.h

```
#define FX_METADATA(x)
#define __FX_INTERFACE(i) i
#define FX_INTERFACE(i)
__FX_INTERFACE(I_##i)

// From sources
#define I_HOT_DOG_VER1 "hotdog.h"
#define I_FLAVOR_KETCHUP "ketchup.h"
#define I_FLAVOR_MUSTARD "mustard.h"
#define I_FILLING_SAUSAGE "sausage.h"
#define I_HOT_DOG I_HOT_DOG_VER1
#define I_FILLING I_FILLING_SAUSAGE

// From alias file
#define I_FLAVOR I_FLAVOR_MUSTARD
```



Применение метаданных в FX-RTOS

- Ядро ОСРВ как набор заменяемых компонентов
- Всего ~ 300 компонентов
- Совместимость со всеми популярными системами сборки
- Масштабирование ядра от 16-битных микроконтроллеров до многоядерных процессоров с MMU
- В ядре для MPU используется 90% кода для MCU
- Специальные конфигурации
  - low-latency
  - secure
  - tiny

Ссылки

Спецификация формата FX_METADATA

https://github.com/Eremex/metadata_spec

FX-DJ

- Извлечение метаданных
- Внедрение зависимостей

https://github.com/Eremex/fx-dj

FX-MGR

- Более “продвинутая” (и старая) реализация на C#
- Внедрение зависимостей
- Аспекты, опции, конструкторы и т.д.

https://github.com/Eremex/fx-mgr