Java to Phantom

Трансляция байткода в двух отделениях и одном антракте

Задача

- Обеспечить migration path для Phantom OS
- Из Java+JVM
- Из Android+Dalvik
- Потенциально - из .Net/CLR

Диспозиция

- Виртуальная машина Фантом разработана без оглядки на JVM (честно!), но похожа как две капли
- Но вовсе не идентична. Действительно всё - объекты, более строгий вызов и возврат, нет статики и т.п.
- От Андроидной отличается сильно

Ещё отличия

- Всё персистентно, JIT можно делать статически.
- Статика есть в яве, значит - надо её эмулировать.
- Нет класслоадеров. Может, зря.

Что было готово

- Кодогенератор в байткод от «родного» компилятора языка Фантом - принимает на вход дерево операций, льёт на выходе фантомовские класс-файлы.
- Какое-то количество работающего кода на языке Фантом, юнит-тесты кодогенератора.

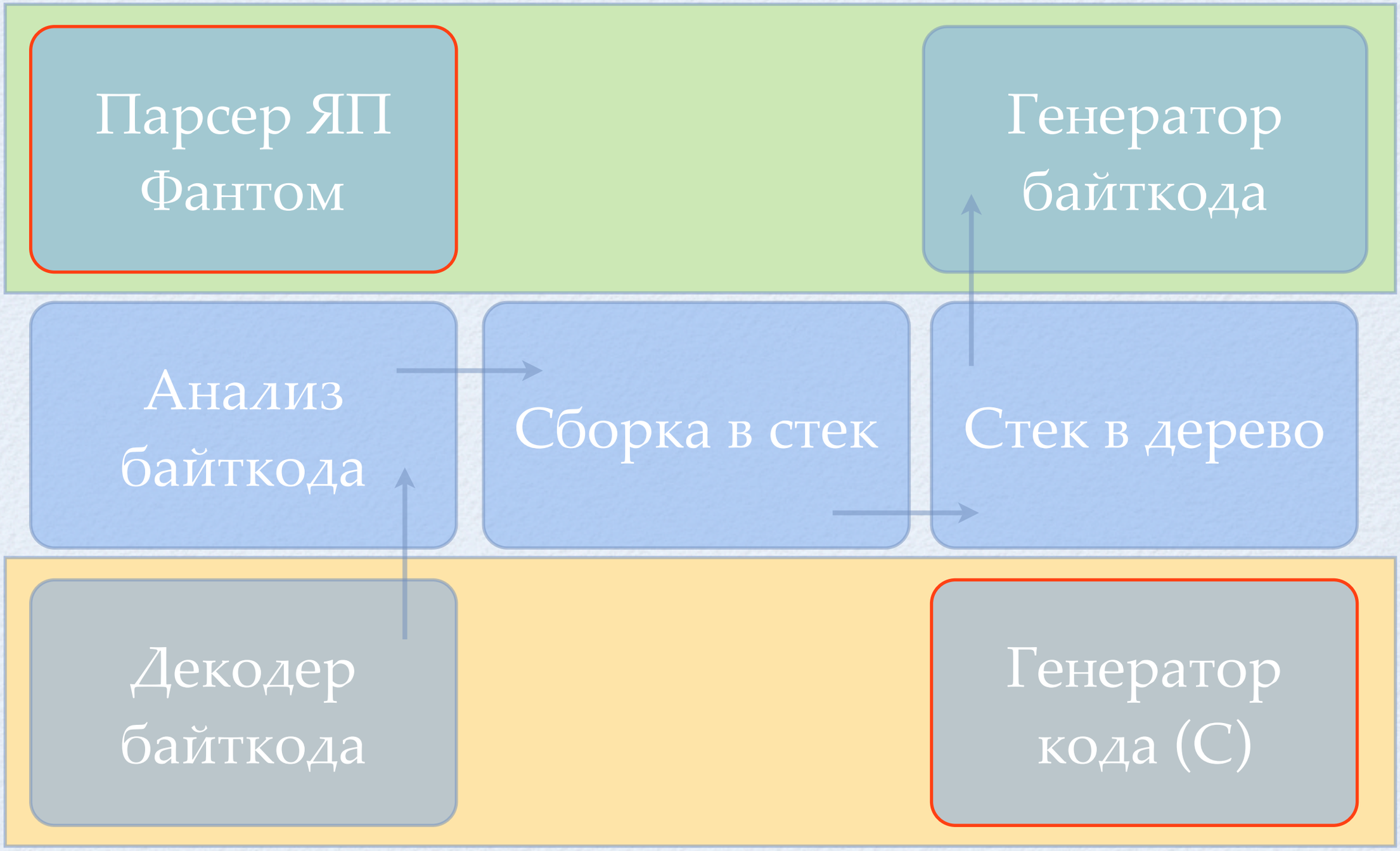
Отделение 1 - това

- Берём конвертор java-байткода в си и дорабатываем напильником.
- Война и немцы - приходится симулировать работу стек-машины, чтобы собрать байткод в дерево операций
- Распознать, лежит ли в нулевом аргументе this по байткоду тяжело.

TOBA не катит

- Знает только байткод 1.2 (не так и страшно, но более поздние-то делать надо)
- Транслятор состоит из трёх слоёв шаманства, после полугода паузы разобраться в коде очень тяжело
- Разработка потребовала огромных усилий и забуксовала

На базе tova


Парсер ЯП Фантом
Генератор байткода
Анализ байткода
Сборка в стек
Стек в дерево
Декодер байткода
Генератор кода (C)

Ещё проблемы

- Сгенерировать dup для повторного использования результата операции чрезвычайно тяжело.
- Не вполне ясны критерии сброса стека в дерево. Начало следующего выражения? Где оно?
- Если сбрасывать стек по началу перехода, выражения в линейной части будут сгенерированы в обратном порядке.

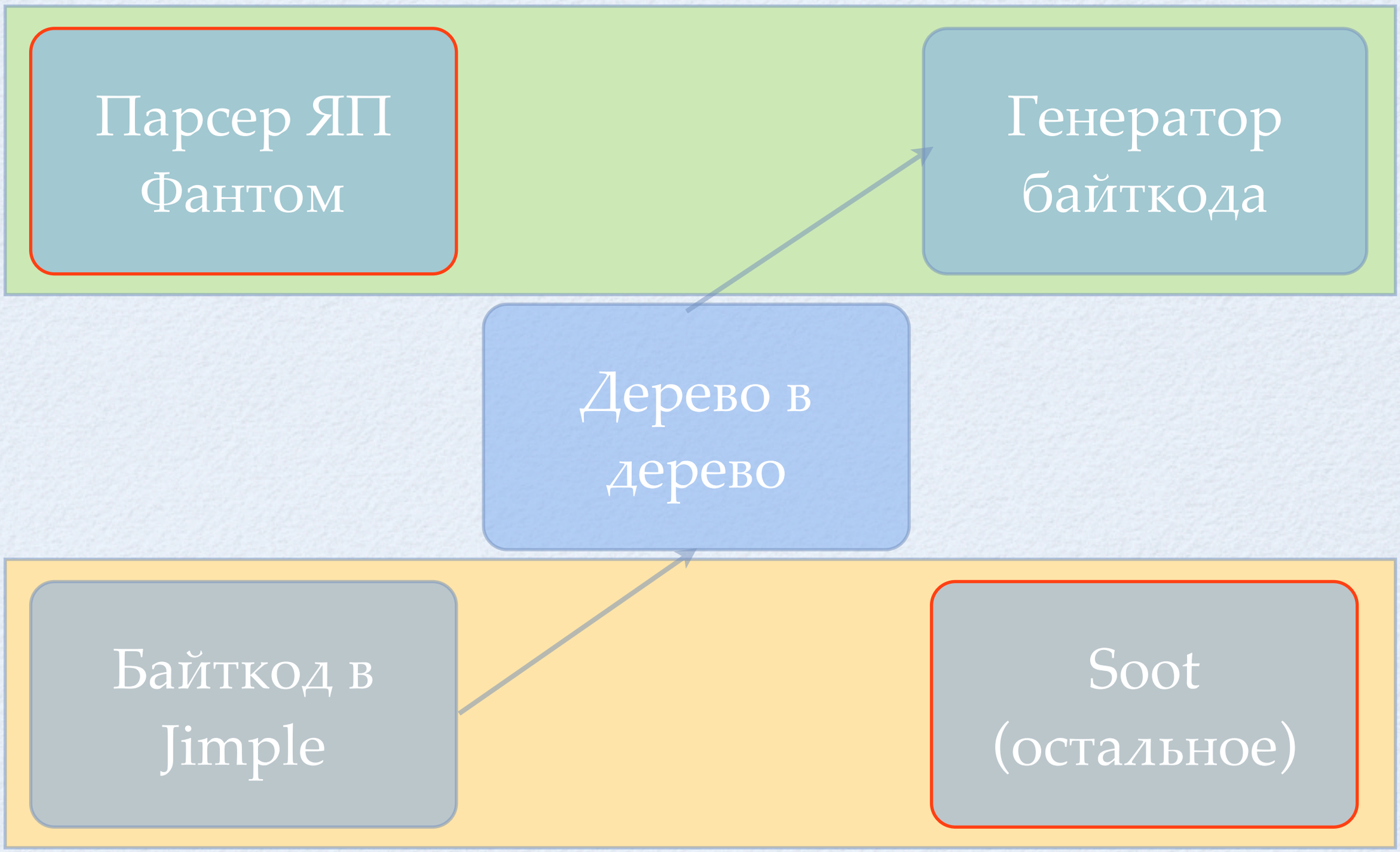
Отделение 2 - Soot

- Был найден в попытке выяснить, есть ли инфраструктура для работы с Dalvik.
- Фреймворк для анализа и трансформации байткода. Активно развивается, есть коммьюнити.
- Не документирован, но интуитивно понятен и есть обширная переписка :)

Как устроено

- Транслятор сводится к тому, чтобы взять дерево, которое сгенерировал Soot и конвертировать его в дерево, которое ест фантомовский кодогенератор
- Первый успешно отработавший прогон через неделю после начала разработки
- Множество мелочей закрывает собой Soot (this отличим от параметра, есть аннотации и т.п.)

На базе soot


Парсер ЯП Фантом
Генератор байткода
Дерево в дерево
Байткод в Jimple
Soot (остальное)

Наивная реализация

- Не используется оптимизатор Soot. Не исследовано.
- Генерация кода может быть сделана иначе - через подсистему конвертации дерева Jimple в псевдо-байткод Baf. Более оптимально?
- Впрочем, всё это - пустое на фоне JIT.

Антракт - LLVM

- Рабочего кода нет, но частичная кодогенерация работает, llvm-as не ругается.
- Есть много вопросов, но, кажется, всё разрешимо.
- Хочется добить llvm до глубокой опимизации, но тогда нужно им и линковать, а это, кажется, не получится.

Проблемы

- Стандартный фантомовский кодогенератор работает на двух стеках и гоняет int-ы через объектный стек. В JIT этого хочется избежать. Но типизация узлов дерева вычислений навскидку не позволяет.
- Предположительно, связывание придётся вынести в рантайм - теряем глубокий инлайн и подстановку констант через границу вызова.

Линкер

- Вместо всех вызовов вставляем вызов прокси-функций. Прокси-функция содержит константы с именем класса и метода.
- Фиксируем состояние стека, уходим в линкер. Линкер подменяет вызов прокси на вызов реальной функции, затем откручивает стек и прыгает на вход реальной функции.

Синхронизация

- Снапшот можно делать только в момент, когда регистры сохранены в персистентной памяти. Надо поймать треды и поставить их на паузу.
  - volatile native_t *catch_point; *catch_point = 0;
  - hal_page_control(... catch_point ... page_readonly ...);
  - Ловим их в pagefault, записываем стейт процессора в персистентный объект.

Статус

- Есть живой байткод
- Нет long/float/double
- Неэффективные массивы
- Готовим юнит-тесты
- Тест не проходит - расширяем покрытие языка

Фантом в целом

- Главное, чего нет - это, как раз, тулчейна для прикладной разработки. Его и делаем.
- Есть ещё много ошибок, недоделок и просто глупостей, но уже нет зияющих дыр в стенах и полу.
- Процесс начинает выглядеть сходящимся. :)

Спасибо! Вопросы?

- Дмитрий Завалишин, dz@dz.ru
- Digital Zone, http://dz.ru

- Присоединяйтесь к проекту Фантом - это open source, и это невероятно интересно.
- Google code, phantomuserland (там и ядро, не верьте названию репозитория)