Вступление

План:

- Краткое описание архитектуры `Cloud-Haskell`
- Текущий статус
- Расширение Static Pointers

Распределенное программирование покрывает много систем

- от слабосвязанные систем (SOA)
  - REST API
  - HTTP EndPoints
  - Расширяемые протоколы JSON/XML
- до сильно связаных систем
  - RPC
  - протоколы передачи сообщений
  - фиксированные схемы общения
  - формат сообщений является внутренним

Что тут может дать Haskell?

- Поддержка библиотек (wai,http-client, aeson, cloud haskell, . . .)
- Легкая возможность подключения внешних библиотек (CCI, zmq, . . .)
- Оптимизирующий компилятор
- Достаточно мощная система типов
- Язык поддерживающий высокоуровневые абстракции

В чем задачи Cloud Haskell

- Слоган: Erlang в Haskell
- Идея: управление кластером "в целом"
  (расширение идеи программирования многопроцессорных систем)
- Запуск исполняемого файла, который может работать
  с распределенными ресурсами как с локальными
- Реализуется библиотекой без расширений компилятора (почти)

Модель исполнения

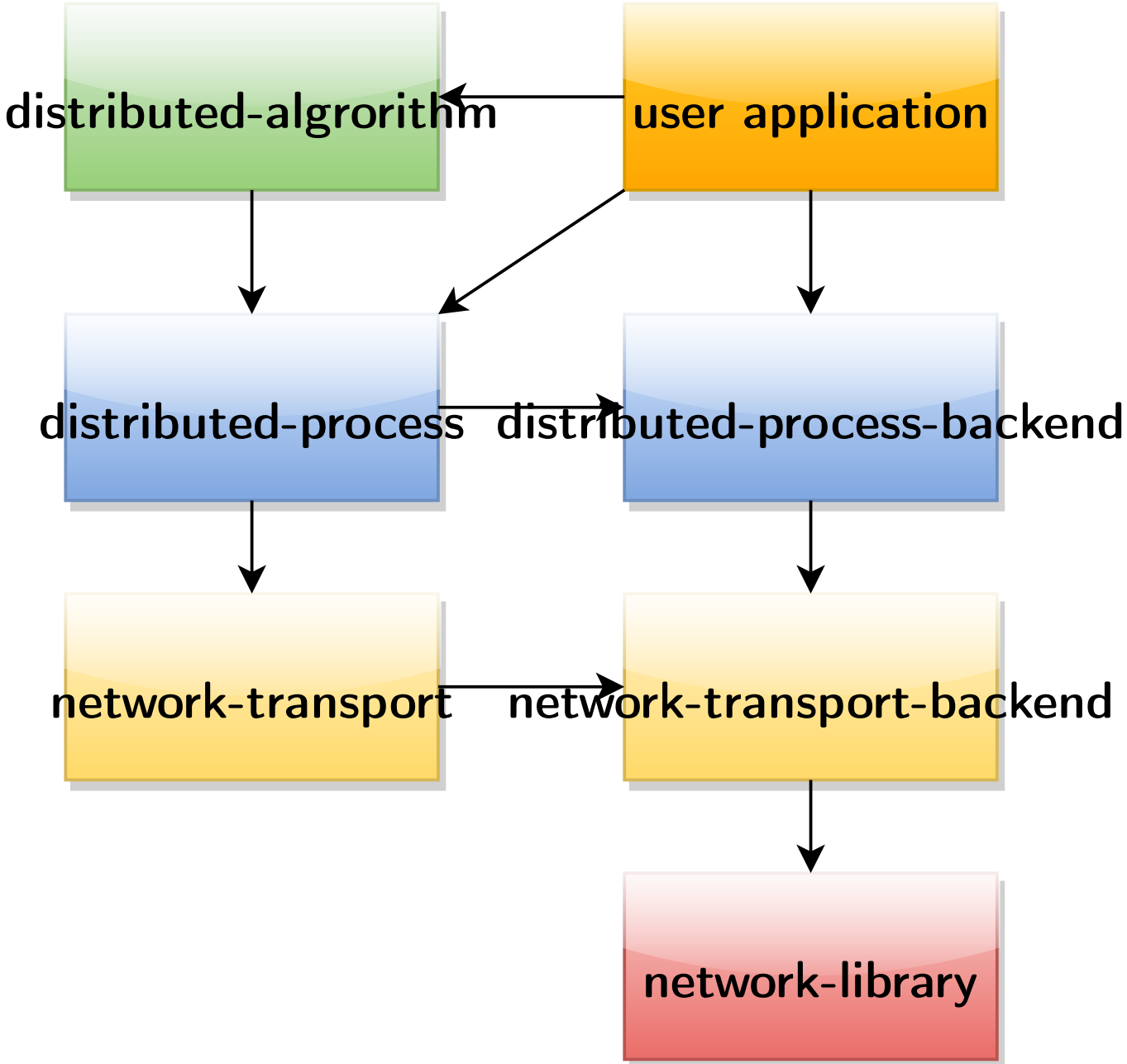
- Используется Actor model
  - явная конкурентность
  - использование легковесных процессов
  - наличие только локального состояния
  - взаимодействие путем обмена сообщениями
- Любую программу можно представить в виде Actor модели
- Cloud Haskell не единственная Haskell библиотека использующая модель актёров

Гибкость

В основе дизайна Cloud Haskell стояла легкая возможность адаптации библиотеки к разным условиям.

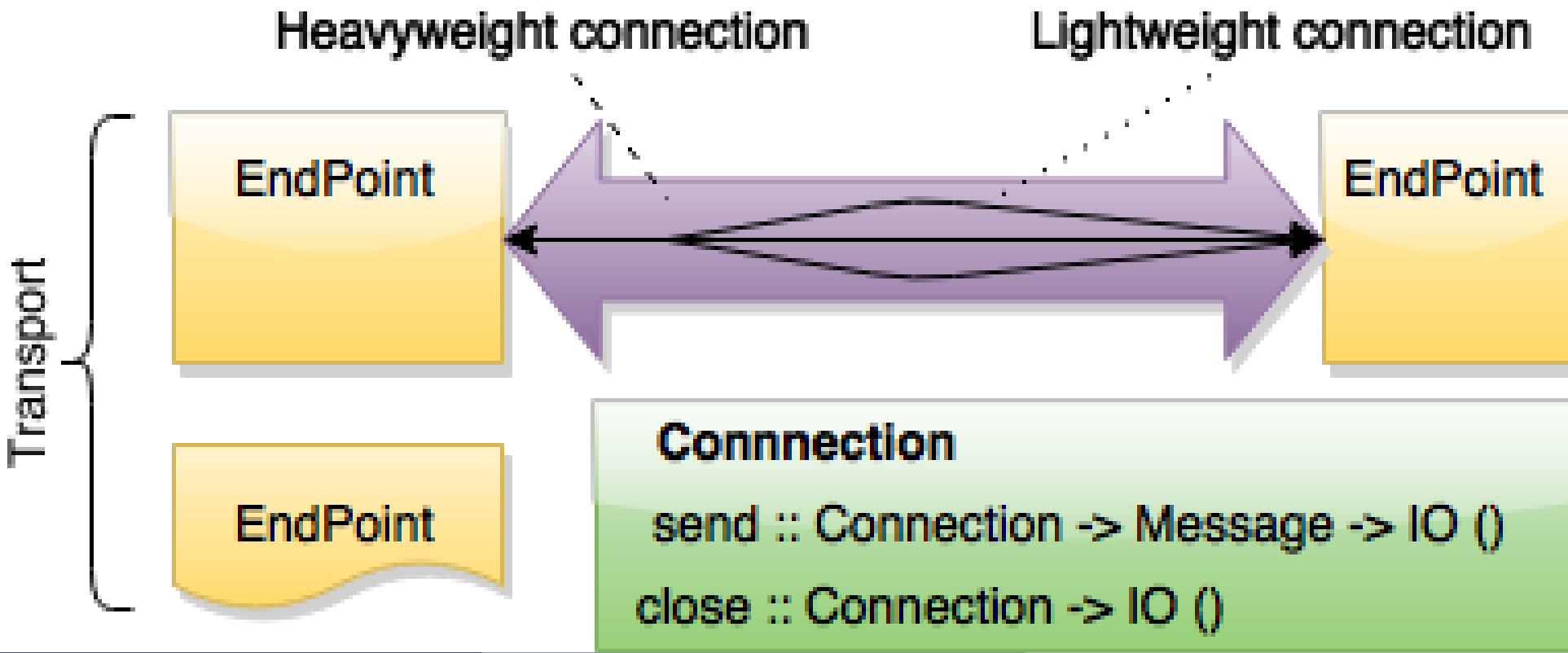
- различные реализации транспорта (железо и протоколы) (TCP, non-IP сети, unix-pipes, shared memory)
- различные способы устанавки исполняемых файлы и инициализировать окружения (scp/ssh, менеджер задач (azure), API облака, загрузка контейнеров)
- различные способы конфигурации исполняемых файлов (переменные окружения, параметры запуска, настройки в менеджере)
- различные способы поиска узлов в сети (динамический поиск в сети, использование master узла, список всех известных узлов)

Архитектура



Network Transport - 1

- API ориентировано на отправку сообщений
  `type Message = [ByteString]`
- легковесные однонаправленные каналы
- управление свойствами соединения (Reliable/Unriable/Ordered/Unordered)



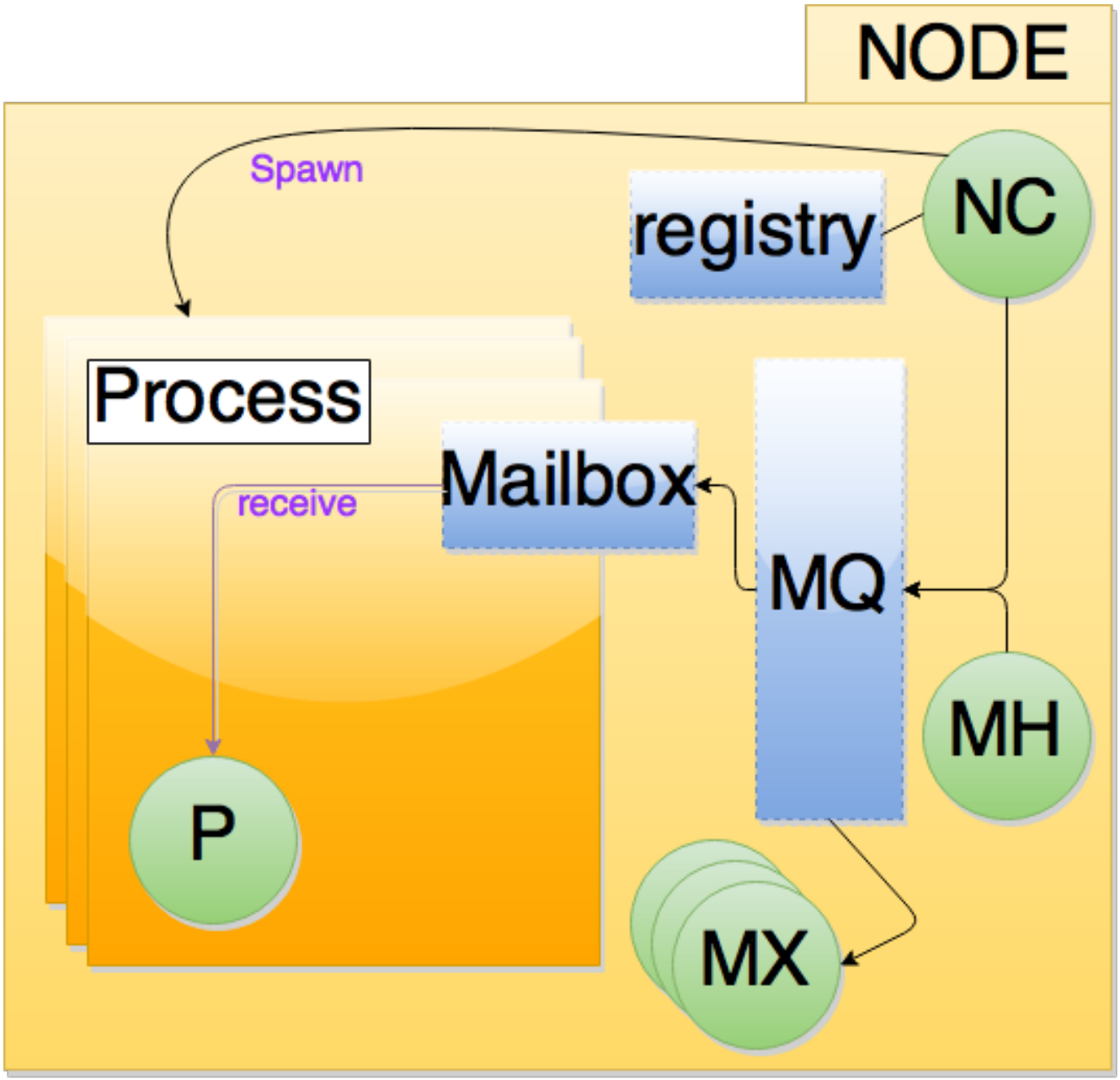
Network Transport - 2

- API (упрощенное)
  - ```
    Transport { newEndPoint :: IO EndPoint
              , transportClose :: IO ()
              }
    ```
  - ```
    EndPoint { receive :: EndPoint -> IO Event
             , endPointClose :: IO ()
             , address :: EndPointAddress
             , connect :: EndPointAddress -> IO Connection
             }
    ```
  - ```
    Connection { send :: Message -> IO ()
               , close :: IO ()
               }
    ```
- Существующие реализации
  - TCP (стабильное)
  - CCI (стабильное)
  - zeromq (экспериментальное)
  - in-memory (экспериментальное)
  - p2p (???)
    ```

Distributed Process - 1

- Процессы (тред Haskell)
- Легковесное соединения для каждой пары общающихся процессов
- Message Handler (MH) – поток распределяющий сообщения, события
- Node Controller (NC) – запуск, связывание процессов, мониторинг, управление реестром
- Node Agent (Logger, пользовательские агенты).

distributed-process layer



API

- управление процессами
  - data ProcessId
  - data Process a
  - newLocalNode :: Transport → IO LocalNode
  - forkProcess :: LocalNode → Process () → IO ProcessID
  - spawn :: NodeId → Closure (Process ()) → Process ProcessId
  - spawnLocal :: Process a → ProcessId
- отправка сообщений
  - send :: Serializable a ⇒ ProcessId → a → Process ()
  - expect :: Serializable a ⇒ Process a

Каналы в Cloud Haskell

- неконтролируемая отправка сообщений может приводить к проблемам разного рода
  - возможность memory leak, нужно периодически очищать очередь от неизвестных сообщений
  - нет проверки системой типов
- Каналы
  - Позволяют типизировать сообщения
  - Позволяют не тегировать передаваемые данные
  - API
    - ★ newChan :: Serializable a ⇒ Process (SendPort a, ReceivePort a)
    - ★ (Functor ReceivePort, Applicative ReceivePort, Monad ReceivePort, Serializable SendPort)
    - ★ sendChan :: Serializable a ⇒ SendPort a → a → Process ()
    - ★ receiveChan :: Serializable a ⇒ ReceivePort a → Process a
    - ★ mergePortsBiased, mergePortsRR

Сериализация и передача функций - 1

- В haskell в отличии от VM-языков сериализация не является свойством среды
- Не все значения могут быть сериализованы
- class (Binary a, Typeable a) $\Rightarrow$ Serializable a
- С сериализаций функций все сложнее, особенно если в функции есть свободные переменные
- spawn :: Closure (Process a) -> Process ProcessId

Сериализация и передача функций - 2

- Разрешаем сериализацию только "статических функций нету свободных переменных или переменные определены в toplevel.
- data Static a
- Такие функции можно применять к сериализуемым значением и результат будет сериализуем
- data Closure a

Сериализация и передача функций - 3

- Remote Table
  - data Closure a = Closure (Static (ByteString -> a)) ByteString
  - data Static a = Static StaticLabel
  - data StaticLabel = StaticLabel String | StaticApply StaticLabel StaticLabel
  - type RemoteTable = Map StaticLabel $\rightarrow$ Dynamic
- Проблемы
  - Легко приводит к ошибкам
  - Не типобезопасно
  - Антимодулярно

Static Pointers - 1

Towards Haskell in the Cloud, Jeff Epstein, Andrew P. Black, Simon Peyton-Jones

- расширение `-XStaticPointers`
- конструктор: ключевое слово static
- выражение должно быть замкнуто
- $\Gamma a : A$, static expr $\rightarrow$ StaticPtr a
- deRefStaticPtr :: StaticPtr a $\rightarrow$ a

Static Pointers - 2

- type StaticKey = Fingerprint – уникальный ключ для выражения под static
- data Fingerprint = Fingerprint Word64 Word64
- staticKey :: StaticPtr a → StaticKey
- unsafeLookupStaticPtr :: StaticKey → Maybe (StaticPtr a)
- в GHC 7.12:
  lookupStaticPtr :: StaticKey → (∀ a . Typeable a ⇒ StaticPtr a → b) → Maybe (StaticPtr b)

Static Pointers - 3

- Для каждого вхождения `static` генерируется top-level выражение и `StaticKey`,`StaticPtr`, `StaticPtrInfo` (пакет, модуль, внутренее имя, локация в коде)
- `data StaticPtr a = StaticPtr StaticKey StaticPtrInfo a`
- `static StgWord64 key[2]`
- `static void hs_hpc_init_Main(void) __attribute__((constructor));`
- при инициализации модуля создается/или дополняется глобальная хеш таблица `StaticKey` $\rightarrow$ `StaticPtr`

Static Closure - 1

- URI: https://github.com/tweag/distributed-closure
- Находится в разработке
- ```
  data Closure a where
    StaticPtr :: !(StaticPtr a) -> Closure a
    Encoded :: !ByteString -> Closure ByteString
    Ap :: !(Closure (a -> b)) -> !(Closure a) -> Closure b
  ```
- существуют и другие подходы к сериализации, см `https://ghc.haskell.org/trac/ghc/blog/simonpj/StaticPointers`
- `Value :: (StaticPtr (ByteString a)) -> ByteString -> Closure a`
  ```

Static Closure - 2

- (квази?)-аппликативный функтор
- cpure :: Serializable a $\Rightarrow$ a $\rightarrow$ Closure a
- $<*>$ :: Closure (a $\rightarrow$ b) $\rightarrow$ Closure a $\rightarrow$ Closure b
- closure :: StaticPointer a $\rightarrow$ Closure a
- Пример:

```
fac :: StaticPtr (Int -> Int)
fac = static (\x -> x * unstatic (fac <*> cpure (x-1)))

fac50on :: NodeId -> Process Int
fac50on = call node (fac <*> pure 50)
```

- необходимо быть осторожным при вызове кода удаленно
- требуется минимальная поддержка среды выполнения
- C-H предоставляет основу для построения более высокоуровневых систем
  - построение data-parallelism
  - MapReduce, распределенный NestedDataParallelism, DAG из преобразований (Spark)