Domain-specific Hotspot Optimization with Scalan

Alexander Slesarenko (@avslesarenko)
Maxim Gekk (@maxgekk)
Alexey Romanov (@alexey_r)

Huawei Technologies, Russian Research Center

Jetconf.by, September 28, 2015

www.huawei.com

Дисклэймеры

- Все это эксперимент, и может не полететь в вашем конкретном случае. (Дайте знать если все же решитесь использовать и будут проблемы)

- Будьте осторожны, чтобы код *случайно* не ушел в продакшен. (Хотя у нас ушел и *пока* все *вроде* нормально)

- Все сказанное - это личное мнение докладчика и может не совпадать с мнением компании Huawei.

План

- Мотивация
- Фреймворк для спец-оптимизации
- Как оно работает
- Как использовать

Отправная точка

- Предположим мы хотим кодировать в функциональном стиле
- От этого мы ожидаем: предсказуемость, изменяемость, модульность ...
- Даже скорость и отзывчивость программы часто может быть лучше за счет естественной поддержки асинхронных паттернов (Async monad, Rx, Scalaz-streams, etc).

Условно говоря

- 80% ФП кода работает хорошо.
- 20% работает плохо, медленно

Есть **узкие места** и они требуют особого внимания

Пример 1: **M**atrix **V**ector **M**ultiplication

$$\overset{M}{\begin{pmatrix} 1.0 & 0 & 2.0 & 0 \\ 3.0 & 4.0 & 5.0 & 0 \\ 0 & 0 & 0 & 6.0 \end{pmatrix}} * \overset{V}{\begin{pmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{pmatrix}} = \overset{R}{\begin{pmatrix} 7.0 \\ 26 \\ 24 \end{pmatrix}}$$

Абстрактные типы

```
def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))
```

Создаем новый вектор по массиву координат

Достаем строки как массив векторов

Скалярное произведение

Что нам делать, если это узкое место?

Цена абстракций

Матрицы: 10000 x 10000 элементов

S_m, S_v - разреженность (% нулей)

```
def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))

val m = loadArrays("matrix.dat")
val v = loadArray("vector.dat")
val res = mvm(new DenseMatr(m), DenseVec(v))
```

```
val nRows = m.length
var res = new Array[Double](nRows)
for (i <- 0 until nRows) {
  val row = m(i)
  val nCols = row.length
  var sum:Double = 0
  for (j <- 0 until nCols)
    sum += row(j) * v(j)
  res(i) = sum
}
```

~ 40x быстрее

S_m	S_v	**dmdv**
0%	0%	11389
10%	10%	11408
50%	50%	12944
90%	90%	13093
99%	99%	13251
0%	50%	11483
50%	0%	12946
10%	90%	13907
90%	10%	14304

S_m	S_v	**dmdv**
0%	0%	309
10%	10%	311
50%	50%	310
90%	90%	307
99%	99%	307
0%	50%	308
50%	0%	310
10%	90%	311
90%	10%	311

Пример 2: Композитные приложения

Мы хотим писать такой код

Algebraic DSLs

```
def prg[F[_]](implicit I: Interacts[F], A: Auths[F]): Free[F,Unit] = {
  import I._, A._;
  for {
    uid <- ask("What's your user ID?")
    pwd <- ask("Password, please.")
    u <- login(uid, pwd)
    b <- u.map(hasPermission(_, KnowSecret)).getOrElse(Return(false))
    _ <- if (b) {
      for { _ <- tell("Hi “ + uid) } yield ()
    } else {
      for { _ <- tell("Sorry “ + uid) } yield ()
    }
  } yield ()
}

type App[A] = Coproduct[Auth, Interact, A]
def runApp = prg[App].run(AuthId or InteractId)
```

Абстракция дает свободу выбора (поменять интерпретацию)

Источник: Compositional Application Architecture With Reasonably Priced Monads (by Rúnar Bjarnason, Scala Days, Berlin, 2014)

Пример 2: Композитные приложения

Код, который нам в итоге нужен для исполнения.

```
def login: Unit = {
  println("What's your user ID?")
  val uid = readLine()
  println("Password, please.")
  val pwd = readLine()
  val ok = uid == KnowSecret && pwd == "Ghost”
  if (ok) {
    println("Hi " + uid)
  } else {
    println("Sorry " + uid)
  }
}
```

Исполняется быстрее, бинарник меньше, … одни плюсы.

Проблема: Если мы так напишем, то только так сможем исполнить.

Абстракция без сожаления

Пишем это

```
def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))
```

⇩ **??? автоматически**

Исполняем это

```
val nRows = m.length
var res = new Array[Double](nRows)
for (i <- 0 until nRows) {
  val row = m(i)
  val nCols = row.length
  var sum:Double = 0
  for (j <- 0 until nCols)
    sum += row(j) * v(j)
  res(i) = sum
}
```

План

- **Мотивация**
- **Фреймворк для спец-оптимизаций**
- **Как оно работает**
- **Как использовать**

Почему с хорошими людьми случаются неприятности

- Чтобы ускориться мы легко отказываемся сначала от ФП, а затем и от абстракций вообще
- Мы **вручную** делаем такие (обычно нехорошие) вещи:

1. Де-функционализацию
   - Убираем лямбды, делаем inlineing где надо
   - Заменяем immutable структуры на mutable
2. Специализацию
   - от общих и простых структур к спец-представлениям
   - спец-механизмы (cache, allocators, и т.д.)
3. Акселерацию
   - давайте перепишем на C++ (JNI (or Unsafe))
   - поручим часть вычислений GPU
4. Всякие трансформации
   - Массив структур в Структуру Массивов (AoS -> SoA)

В чем собственно вопрос?

Вопрос в следующем:

- Никто на самом деле не хочет менять уже написанный код
- Особенно, если код хорошо написан, выделены все абстракции и он работает
- Особенно, если сами же его написали, при этом позаботились о модульности, читаемости и модифицируемости

Альтернатива

- **Обучить компилятор** генерировать эффективный код для данного конкретного узкого места
- **Не надо трогать хороший код** ни при каких условиях

«Хороший» код

```scala
/** Matrix Vector Multiplication */

def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))
```

«Плохой» код

```scala
val nRows = m.length
var res = new Array[Double](nRows)
for (i <- 0 until nRows) {
  val row = m(i)
  val nCols = row.length
  var sum:Double = 0
  for (j <- 0 until nCols)
    sum += row(j) * v(j)
  res(i) = sum
}
```

Спец-компиляция узких мест (Hotspots)

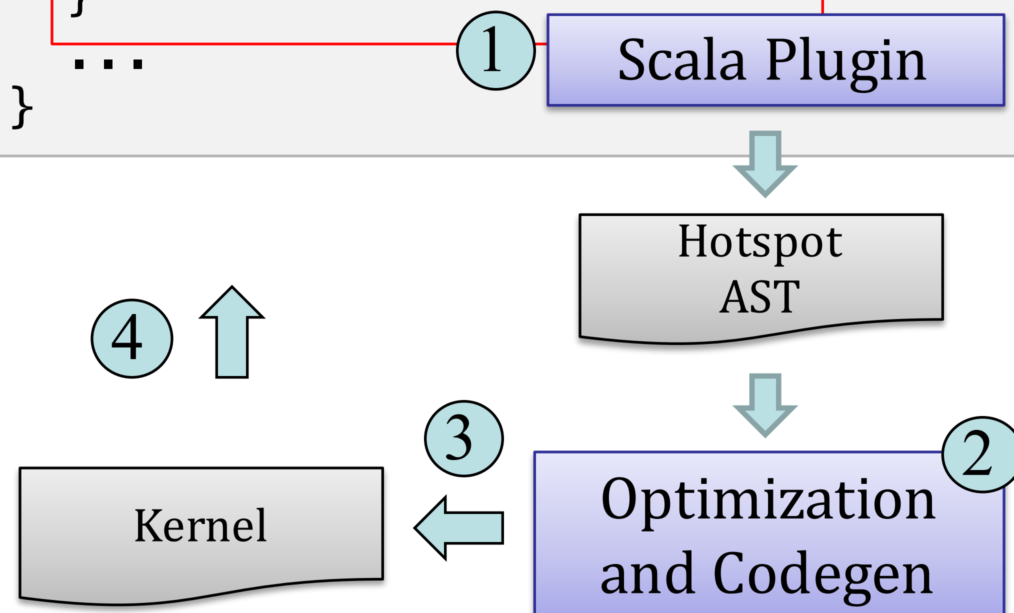
Идея: А давайте использовать свойства предметной области!

Шаги

1. **Забираем** AST фрагмента
2. **Применяем** спец-оптимизации
3. **Генерируем** «плохой», но быстрый код
4. **Заменяем** «хороший», но медленный на «плохой», но быстрый

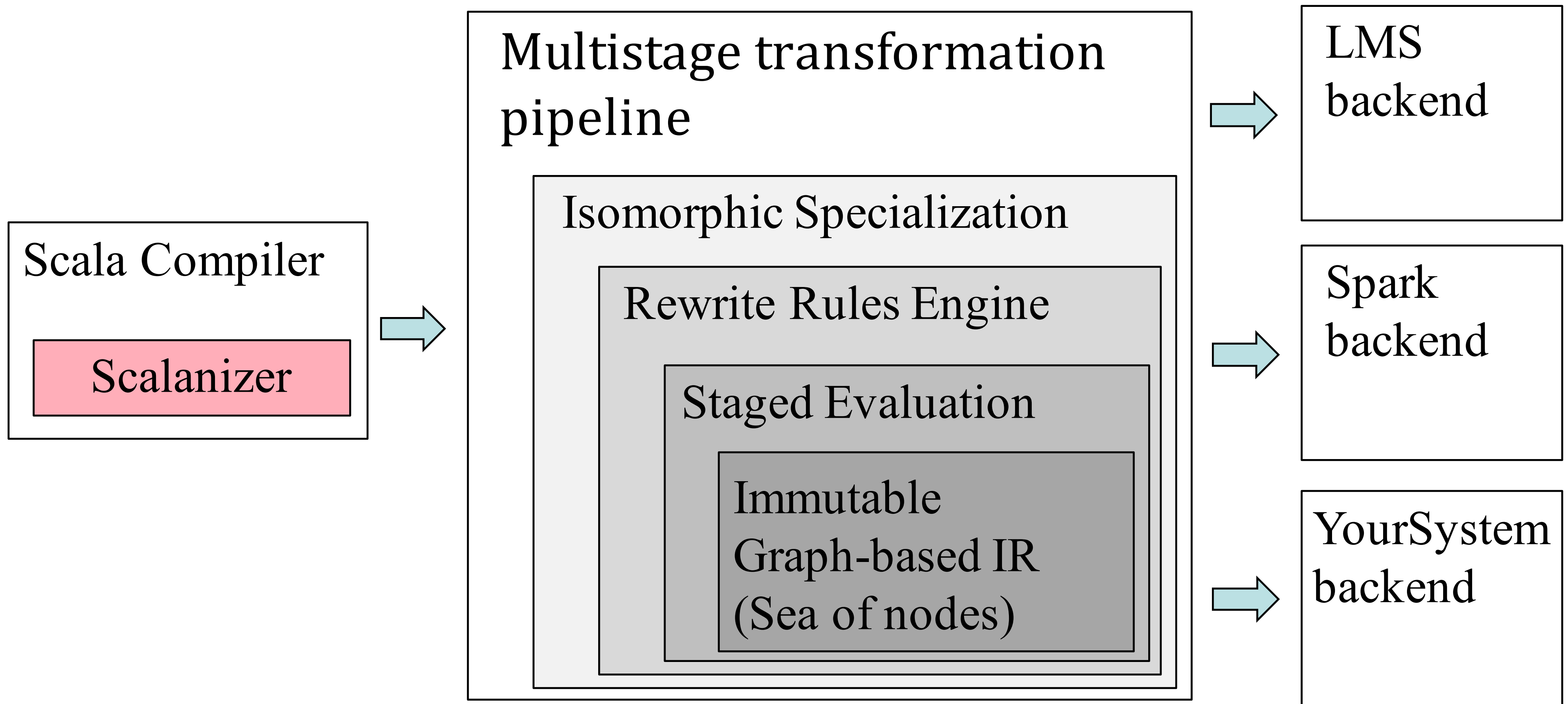
```
def mvm(m: Matrix,
        v: Vector): Vector =
  Vec(m.rows.map(r => r.dot(v)))

def main(args: Array[String]) = {
  ...
  for (i <- 0 to 1000000) {
    val v1 = mvm(m, vs(i))
          ...
  }
  ...
}
```



Дело за малым: написать оптимизаторы для всех требуемых предметных областей?

Scalan – фреймворк для спец-оптимизаторов




```
def rule1 = postulate[Int, Int, Int](
  (a, b) => b * (a /! b)  <=> a)
def rule2[A] = postulate[Array[A], A=>Boolean, Int](
  (xs, p) => array_find(xs, p).length  <=> xs.count(p))
def rule3[A] = postulate[Array[Array[A]], Array[A]](
  xss => xss.flatMap(xs => xs)  <=> xss.flatten)
```

Демонстрация

План

- **Мотивация**
- **Фреймворк для спец-оптимизации**
- **Как оно работает**
- **Как использовать**

Что такое домен в Scalan?

- Иерархия классов и интерфейсов с операциями
- Важна семантическая информация, а не синтаксис

Примеры доменов

```
trait Vec[T] {    // abstract vector
  def length: Int
  def coords: Array[T]
  def dot(vec: Vec[T]): T
}

trait Matr[T] {   // abstract matrix
  def rows: Array[Vec[T]]
  def * (vec: Vec[T]): Vec[T] = {
    val vs: Array[Vec[T]] = rows
    Vec(vs.map(v => v.dot(vec)))
  }
}

def Vec[T](coords: Array[T]): Vec[T]
```

Цель:

В каждом узком месте собрать **артефакты из доменов,** от которых он зависит

Артефакты из доменов

- Кроме AST фрагмента Scalanizer должен достать реализации всех абстрактных типов (ADT), которые в нем использованы
- Реализуются ADT наследованием соответствующих интерфейсов

Пример реализации ADT

```
class DenseVec[T](val coords: Array[T])
  extends Vec[T] {
  def length = coords.length
  def dot(vec: Vec[T]) = vec match {
    case dv: DenseVec[T] =>
      sum(coords |*| dv.coords)
    case sv: SparseVec[T] =>
      sum(sv.values |*| coords(sv.indices))
  }
}

class SparseVec[T](
      val indices: Array[Int],
      val values: Array[T],
      val length: Int) extends Vec[T] {
  ...
}

class DenseMatr[T](
      val rows: Array[Vec[T]]) extends Matr[T]{
  ...
}
```

Техники поддержанные в Scalan

1. Де-функционализацию
   - ✔ Убираем лямбды, делаем inlineing где надо
   - Заменяем immutable структуры на mutable
2. Специализацию
   - ✔ от общих и простых структур к спец-представлениям
   - спец-механизмы (cache, allocators, и т.д.)
3. Акселерацию
   - ✔ давайте перепишем на C++ (JNI (or Unsafe))
   - поручим часть вычислений GPU
4. Всякие трансформации
   - ✔ Массив структур в Структуру Массивов (AoS -> SoA)
   - ✔ rewrite rules
   - ✔ flattening (for Nested Data Parallelism)

Центральная идея в основе Scalan

```
def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))
```

Domain A
(Linear Algebra)

Автоматическая меж-доменная трансляция

```
def dmdv(
  m: Array[Array[Double]],
  v: Array[Double]): Array[Double] =
    m.map(row => sum(row |*| v))
```

Domain B
(Arrays)

Вопросы:

1. Работает для всех программ из A или только некоторых?
2. Кросс доменные программы?
3. Многоуровневые трансляции (A -> B -> C)?

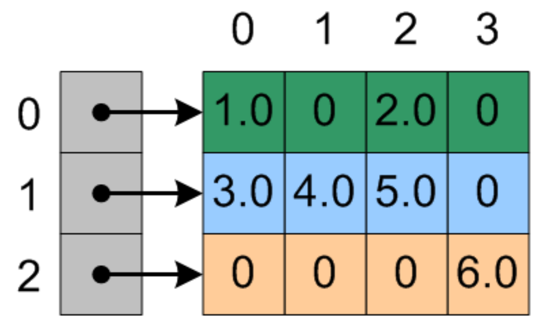
Optimizing Compiler

Runtime
(JVM,C++)

Проблема: Выбор лучшего представления

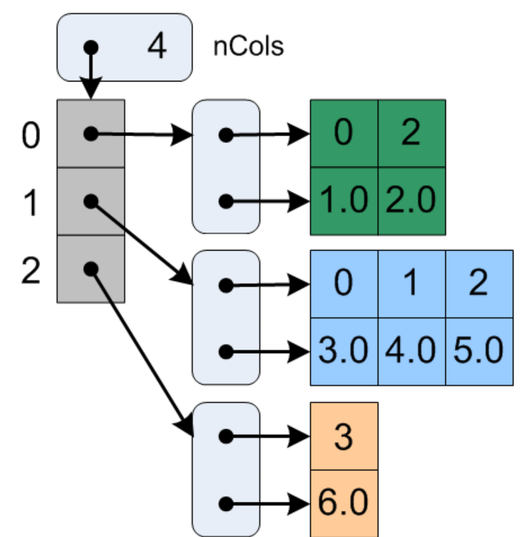
```
def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))
```

Dense Matrix

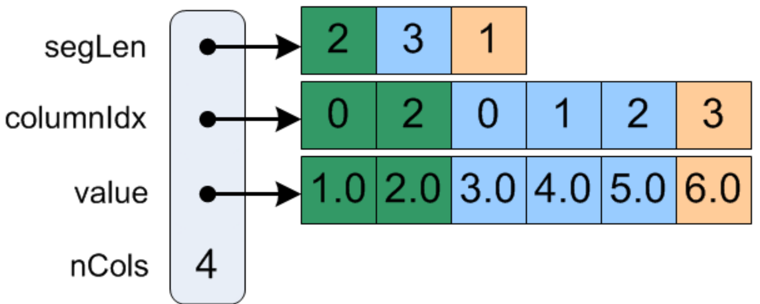


Matrix

Sparse Matrix



Flat Sparse Matrix



Почему это важно?

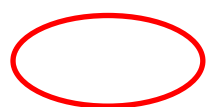
Возмем матрицы $10^4 x 10^4$ разной разреженности

S_m – разреженность матрицы (% нулей)

S_v - разреженность вектора

S_m	S_v	**dmdv**	**dmsv**	**smdv**	**smsv**
0%	0%	309	354	366	760
10%	10%	311	323	332	1002
50%	50%	310	202	187	924
90%	90%	307	104	42	172
99%	99%	307	18	8	18
0%	50%	308	198	373	1134
50%	0%	310	359	187	986
10%	90%	311	118	335	497
90%	10%	311	323	42	345

Время выполнения в миллисекундах


Лучший вариант для данной разреженности

Как догадаться, что это плохой выбор?

Почему это важно (2): умножение матриц

```
def mmm(A: Matr, B: Matr]): Matr =
  Matr.fromColumns(B.columns.map(c => mvm(A, c)))
```

исходный формат данных

A: 10000x1000 B:1000x1000 Execution time in seconds

Sparseness		Scala Kernels				Best Static		Best Dynamic	
A	B	dmdm	dmsm	smdm	**smsm**	Time	Ratio	Time	Ratio
0,01	0,01	**15,55**	16,39	19,84	32,12	15,55	2,26	14,25	2,25
0,01	0,50	15,70	**12,16**	19,47	32,36	12,16	3,21	10,20	3,17
0,01	0,99	16,65	**1,55**	23,08	25,23	1,55	18,42	1,40	18,02
0,50	0,01	**16,87**	18,19	17,64	84,86	16,87	5,96	14,24	5,96
0,50	0,50	16,31	**12,68**	13,87	73,37	12,68	7,20	10,21	7,19
0,50	0,99	15,96	**1,68**	13,87	10,34	1,68	7,60	1,38	7,49
0,99	0,01	15,61	16,27	**0,55**	19,94	0,55	36,50	0,56	35,61
0,99	0,50	15,68	12,12	**0,54**	11,17	0,54	20,59	0,54	20,69
0,99	0,99	15,64	1,39	**0,54**	1,26	0,54	2,32	0,54	2,33
				Total /	**Avg:**	**62,12**	**11,56**	**53,32**	**11,41**

(обведено) Лучший вариант для данной разреженности

Ручное решение - специализация и профилировка

```
def mvm(m: Matr, v: Vec): Vec = {
  Vec(m.rows.map(r => r.dot(v)))
}
```

- Абстрактный домен
- Конкретный домен

1) специализируем

```
def dmdv(m: Array[Array[Double]], v: Array[Double]): Array[Double] =
    m.map(row => sum(row |*| v))

def dmsv(m: Array[Array[Double]], v: (Array[Int], Array[Double], Int)): Array[Double] = {
  val (indices, values, _) = v
  m.map(row => sum(row(indices) |*| values))}

def smsv(m: Array[(Array[Int],Array[Double],Int)],
         v: (Array[Int],Array[Double],Int)): Array[Double] = {
  val (indices, values, _) = v
  m.map((is,(vs,_)) => dotProductSV(is, vs, indices, values))}

def smdv(m: Array[(Array[Int], (Array[Double], Int))], v: Array[Double]): Array[Double] =
  m.map(r => {
    val (indices, values, _) = r
    sum(values |*| v(indices))
  })
```

Implementation-specific primitives:
- map
- sum
- |*|
- dotProductSV

2) запускаем
3) замеряем скорость
4) строим табличку

Как специализировать, вручную?

О какой специализации идет речь?

Немного математики…

Частичные вычисления (Partial Evaluation)	$P(x,y),\ x = const$ $\mathbf{P'}(y)$ for all y: $\mathbf{P'}(y) = P(x,y)$
Supercompilation	$P(x),\ x \in D$ $\mathbf{P'}(x) \wedge x \in D' \subset D$ for all $x \in D'$: $\mathbf{P'}(x) = P(x)$
✔ Isomorphic Specialization	Следующий слайд

Изоморфная специализация

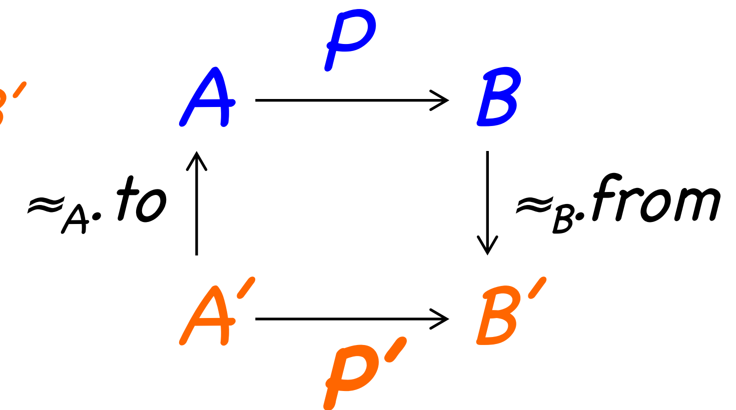
✔ Isomorphic Specialization

P: *A* -> *B*, *A* $\approx_A$ *A′*, *B* $\approx_B$ *B′*

P′: *A′* -> *B′*

такая что диаграмма коммутирует

При этом P′ использует только типы и операции конкретного домена P



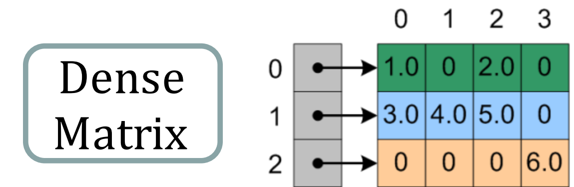
■ абстрактный домен

■ конкретный домен

- Изоморфная специализация – это *автоматическая* трансформация программы
- Реализована как множество правил переписывания (rewrite rules)
- Хорошо комбинируется с другими правилами и дает синергетический эффект

Применяем к MVM

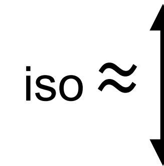
- абстрактный домен
- конкретный домен

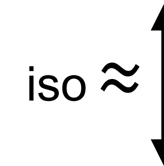


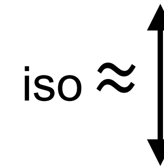
P: A -> B

P′: A′ -> B′

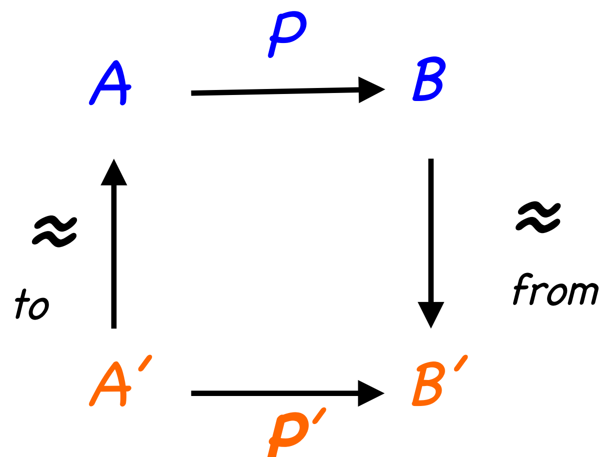
mvm: (Matr, Vec) ⟶ Vec

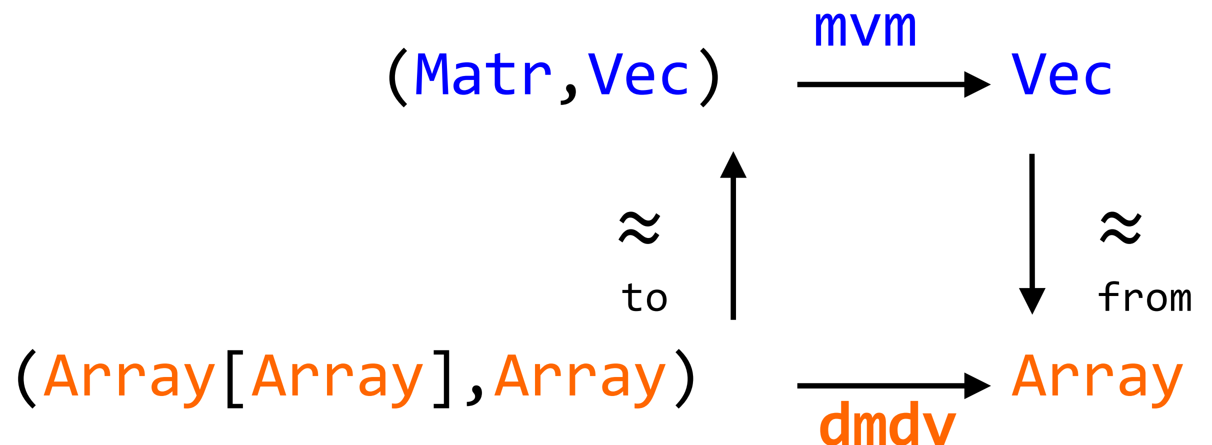






dmdv: (Array[Array], Array) ⟶ Array

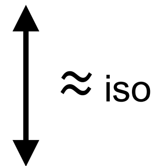




“Специализируемые по построению” абстракции

Vectors Domain

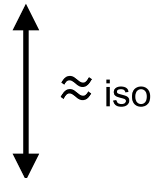
```
Array[T]
```




```
case class DenseVec[T](coords: Array[T])
  extends Vec[T]
```

```
trait Vec[T] {
  def length: Int
  def coords: Array[T]
  def dot(vec: Vec[T]): T
}
```

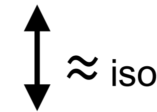
```
case class SparseVec[T](
  indices: Array[Int],
  values: Array[T],
  length: Int) extends Vec[T]
```




```
(Array[Int], Array[T], Int)
```

Matrix Domain

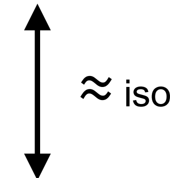
```
Array[Vec[T]]
```




```
case class DenseMatr[T](
  rows: Array[Vec[T]])
  extends Vec[T]
```

```
trait Matr[T] {
  def rows: Array[Vec[T]]
}
```

```
case class SparseMatr[T](
  rows: Array[(Array[Int], Array[T])],
  nCol: Int) extends Vec[T]
```




```
(Array[(Array[Int], Array[T])], Int)
```

Изо-специализация в конвейере компилятора

	0	1	2	3
0	1.0	0	2.0	0
1	3.0	4.0	5.0	0
2	0	0	0	6.0

Dense Matrix $\approx_{iso}$ ⟷ `Array[Array[T]]`

```
def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))
```

Изо-специализация в функциональные операции над массивами

```
def dmdv(m: Array[Array[Double]], v: Array[Double]): Array[Double] =
  m.map(row => sum(row |*| v))
```

LMS (Lightweight Modular Staging)

Компиляция операций над массивами, слияние циклов, удаление промежуточных структур

```
def dmdv(m: Array[Array[Double]],
         v: Array[Double]): Array[Double] = {
  val nRows = m.length
  var res = new Array[Double](nRows)
  for (i <- 0 until nRows) {
    val row = m(i)
    val nCols = row.length
    var sum:Double = 0
    for (j <- 0 until nCols)
      sum += row(j) * v(j)
    res(i) = sum
  }
}
```

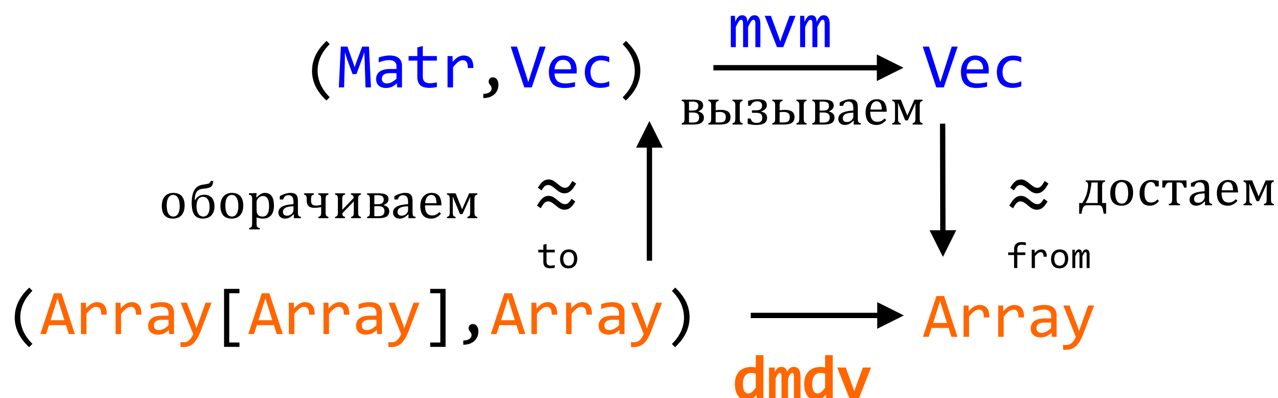
Шаблон Обернуть-Вызвать-Извлечь

```
def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))

def dmdv(m: Array[Array[Double]],
         v: Array[Double]): Array[Double]= {
  // wrap
  val dm = new DenseMatr(
               m.map(r => new DenseVec(r)))
  val dv = new DenseVec(v)

  val vres: Vec = mvm(dm, dv) // apply

  vres.coords    // unwrap
}
```



На абстрактном уровне мы *не хотим знать* как Matr[T] и Vec[T] реализованы

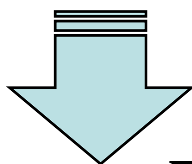
Когда мы хотим применить абстрактный код с конкретными реализациями мы делаем 3 шага:

1) **Оборачиваем** данные
2) **Вызываем** метод
3) **Разворачиваем** данные

Получается задача изо-специализации

Изо-специализация при стадийном вычислении (staged evaluation)

```
def dmdv_S(m: Array[Array[Double]],
           v: Array[Double]) = SE[{
  val dm = new DenseMatr(
                m.map(r => new DenseVec(r)))
  val dv = new DenseVec(v)
  val vres = mvm(dm, dv)
  vres.coords
}]
```



Staged Evaluation(SE) со специализирующими правилами переписывания

```
def dmdv_S(m: Array[Array[Double]],
         v: Array[Double]) =
    m.map(row => sum(row |*| v))
```

Isomorphic Specialization by Staged Evaluation

$$SE[\![_]\!] : Term \to List\ \partial\mathcal{E} \to Dag \to Dag\langle Addr\rangle$$

$$\begin{array}{lll}
SE[\![v']\!]\ (d\,\overline{v}\Box e'\overline{e} :: \mathcal{R})\ \Delta & \mapsto & SE[\![e']\!](d\,\overline{v}v'\Box\overline{e} :: \mathcal{R})\Delta \\
SE[\![d\ e_0\overline{e}]\!]\ \mathcal{R}\ \Delta & \mapsto & SE[\![e_0]\!]\ (d\ \Box\overline{e} :: \mathcal{R})\ \Delta \\
SE[\![l]\!]\ \mathcal{R}\ \Delta & \mapsto & SE[\![v]\!]\ \mathcal{R}\ \Delta' \text{ where } \Delta'\langle v\rangle = \Delta \leftarrow l \\
SE[\![\textbf{case}\ e\ \textbf{of}\ \{p_i \to e_i\}]\!]\ \mathcal{R}\ \Delta & \mapsto & SE[\![e]\!]\ (\textbf{case}\ \Box\ \textbf{of}\ \{p_i \to e_i\} :: \mathcal{R})\ \Delta \\
SE[\![e_1\ e_2]\!]\ \mathcal{R}\ \Delta & \mapsto & SE[\![e_1]\!]\ (\Box\ e_2 :: \mathcal{R})\ \Delta \\
SE[\![v']\!]\ (d\,\overline{v}\Box :: \mathcal{R})\ \Delta & \mapsto & SE[\![v]\!]\ \mathcal{R}\ \Delta' \text{ where } \Delta'\langle v\rangle = \Delta \leftarrow d\,\overline{v}v'
\end{array}$$

$$\begin{array}{lll}
SE[\![v']\!](\textbf{case}\ \Box\ \textbf{of}\ \{p_i\ \overline{x_i} \to e_i\} :: \mathcal{R})\ \Delta_0 & \mapsto & SE[\![v]\!]\ \mathcal{R}\ \Delta' \text{ where} \\
 & & \quad (\Delta_i\langle\overline{\alpha_i}\rangle \ = \Delta_{i-1} \leftarrow \overline{x_i})^n;\ \Delta'_0 = \Delta_n \\
 & & \quad (\Delta'_i\langle\beta_i\rangle \ = SE[\![[\overline{\alpha_i}/\overline{x_i}]e_i]\!]\ \epsilon\ \Delta'_{i-1})^n \\
 & & \quad \Delta'\langle v\rangle \ = \Delta'_n \leftarrow \textbf{case}\ v'\ \textbf{of}\ \{p_i\ \alpha^{n_i} \to \beta_i\}
\end{array}$$

$$\begin{array}{lll}
SE[\![v']\!](\Box\ e_2 :: \mathcal{R})\ \Delta & \mapsto & SE[\![e_2]\!]\ (v'\ \Box :: \mathcal{R})\ \Delta \\
SE[\![v_2]\!](v_1\ \Box :: \mathcal{R})\ \Delta & \mapsto & SE[\![v]\!]\ \mathcal{R}\ \Delta \text{ where } \Delta'\langle v\rangle = \Delta \leftarrow app\ v_1 v_2
\end{array}$$

$$\begin{array}{lll}
SE[\![\lambda x.e]\!]\mathcal{R}\ \Delta & \mapsto & SE[\![v]\!]\ \mathcal{R}\ \Delta' \text{ where} \\
 & & \quad \Delta_1\langle\alpha\rangle \ = \Delta \leftarrow x \\
 & & \quad \Delta_2\langle\beta\rangle \ = SE[\![[\alpha/x]e]\!]\ \epsilon\ \Delta_1 \\
 & & \quad \Delta'\langle v\rangle \ = \Delta_2 \leftarrow \lambda\alpha.\beta
\end{array}$$

$$\begin{array}{lll}
SE[\![e.\mathtt{f}]\!]\ \mathcal{R}\ \Delta & \mapsto & SE[\![e]\!]\ (\Box.\mathtt{f} :: \mathcal{R})\ \Delta \\
SE[\![v']\!](\Box.\mathtt{f} :: \mathcal{R})\ \Delta & \mapsto & SE[\![v]\!]\ \mathcal{R}\ \Delta' \text{ where } \Delta'\langle v\rangle = \Delta \leftarrow v'.\mathtt{f}
\end{array}$$

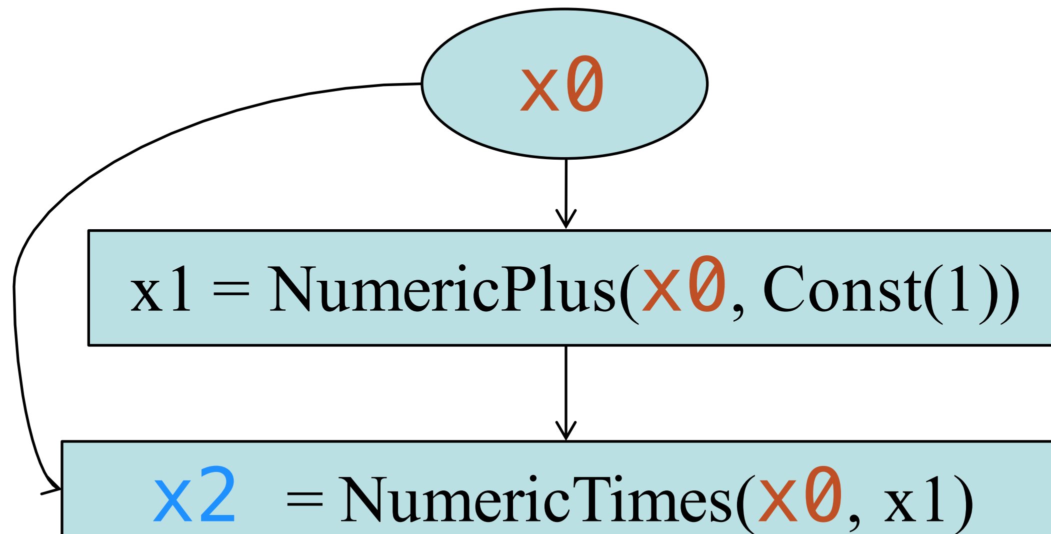
$$\begin{array}{lll}
SE[\![v']\!]((v_0 : \sigma).m(\overline{w}\Box) :: \mathcal{R})\ \Delta & \mapsto & SE[\![v]\!]\ \mathcal{R}\ \Delta' \text{ where} \\
 & & \quad \Delta_1\langle\mu\rangle \ = \Delta \leftarrow \sigma.m \\
 & & \quad \Delta'\langle v\rangle \ = \Delta_1 \leftarrow mcall(v_0, \mu, \overline{w} :: v')
\end{array}$$

$$SE[\![v]\!]\ \epsilon\ \Delta \quad \mapsto \quad \Delta\langle v\rangle$$

Figure 16: Staged Evaluation (call-by-value)

Графовое представление (Graph-based IR)

```
var x0: Int
val x2 = x0 * (x0 + 1)
```



Стадийные вызовы методов на графах

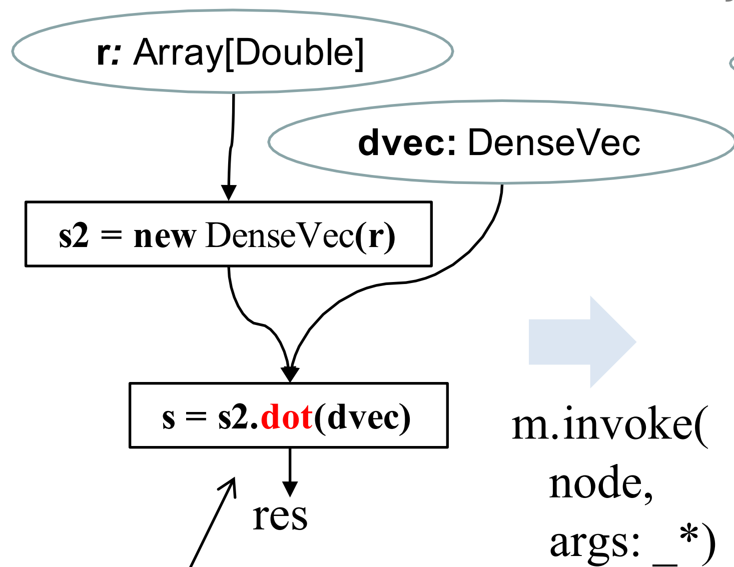
Идея: использовать узлы внутреннего представления как обычные объекты.

Состояние: При стадийном вычислении mvm мы находимся перед вызовом

```
val r: Array[Double]
val dvec: DenseVec
new DenseVec(r).dot(dvec)
```

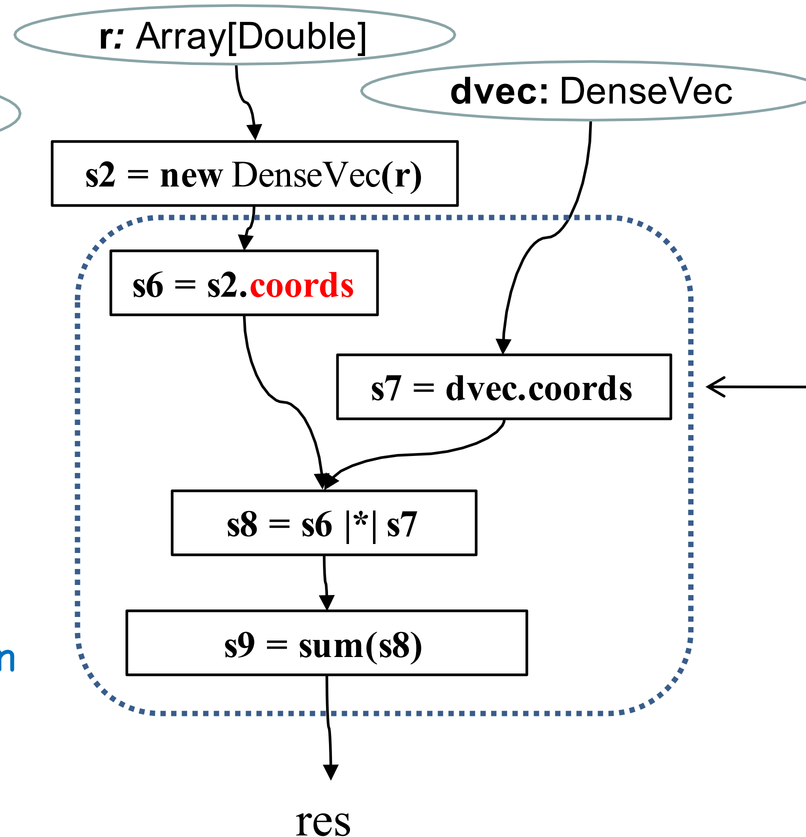
```
def dot(vec: Vec[T]) = vec match {
    case dv: DenseVec[T] =>
      sum(coords |*| dv.coords)
    case sv: SparseVec[T] =>
      sum(sv.values |*| arr(sv.indices))
}
```

Метод вызывается через механизм вызова виртуальных методов



m.invoke(
node,
args: _*)

Java Reflection API



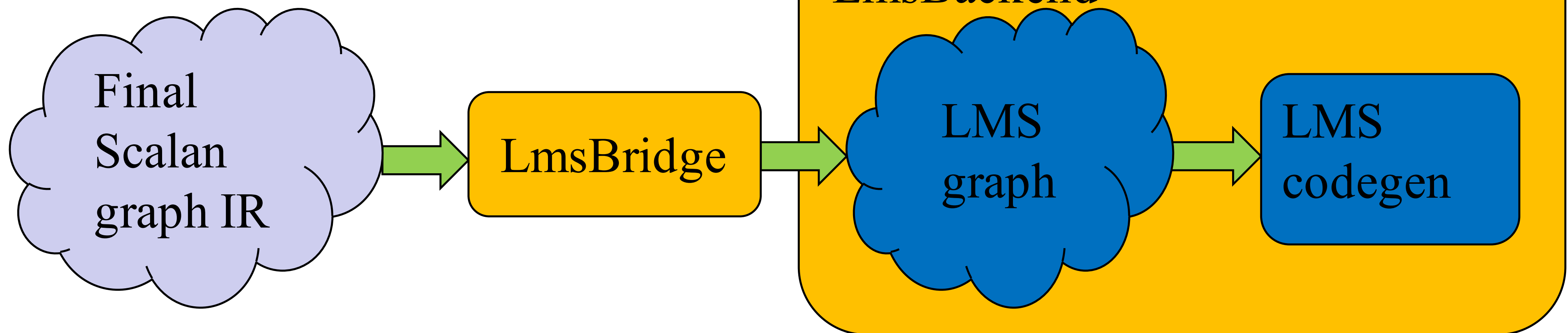
Вызов возможен, потому что s2 ссылается на экземпляр объекта (который одновременно узел графа).

Вызов НЕ возможен и поэтому откладывается в виде узла. Если же вызов возможен, то узел заменяется на подграф.

Компиляция по умолчанию (LMS backend)

```
def dmdv_S(m: Array[Array[Double]], v: Array[Double]) =
    m.map(row => sum(row |*| v))
```

s91:(Array[Array[Double]], Array[Double])
s93:Array[Array[Double]] = s91._1
s94:Array[Double] = s91._2
s95:Array[Double]
s97:Array[(Double, Double)] = ArrayZip(s94, s95)
s98:Array[Double] = ArrayMap(s97, s87)
s96:Array[Double] => Double = s95 => s99 = ArrayReduce(s98, Monoid[Double](+, s71, s82))
s92:((Array[Array[Double]], Array[Double])) => Array[Double] = s91 => s100 = ArrayMap(s93, s96)
s86:(Double, Double)
s88:Double = s86._1
s89:Double = s86._2
s87:((Double, Double)) => Double = s86 => s90 = s88 * s89
s81:(Double, Double)
s83:Double = s81._1
s84:Double = s81._2
s82:((Double, Double)) => Double = s81 => s85 = s83 + s84
s71:Double = Const(0.0)



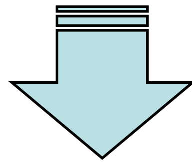
Program Generation and Embedded Compilers in Scala. (http://scala-lms.github.io)

Сгенерированный код

«Хороший» код

```
/** Matrix Vector Multiplication */

def mvm(m: Matr, v: Vec): Vec =
  Vec(m.rows.map(r => r.dot(v)))
```



Scalan + LMS

«Плохой» код

```
def dmdv(m: Array[Array[Double]], v: Array[Double]): Array[Double] = {
    val nRows = m.length
    var res = new Array[Double](nRows)
    for (i <- 0 until nRows) {
        val row = m(i)
        val nCols = row.length
        var sum: Double = 0
        for (j <- 0 until nCols) {
            sum += row(j) * v(j)
        }
        res(i) = sum
    }
    res
}
```

План

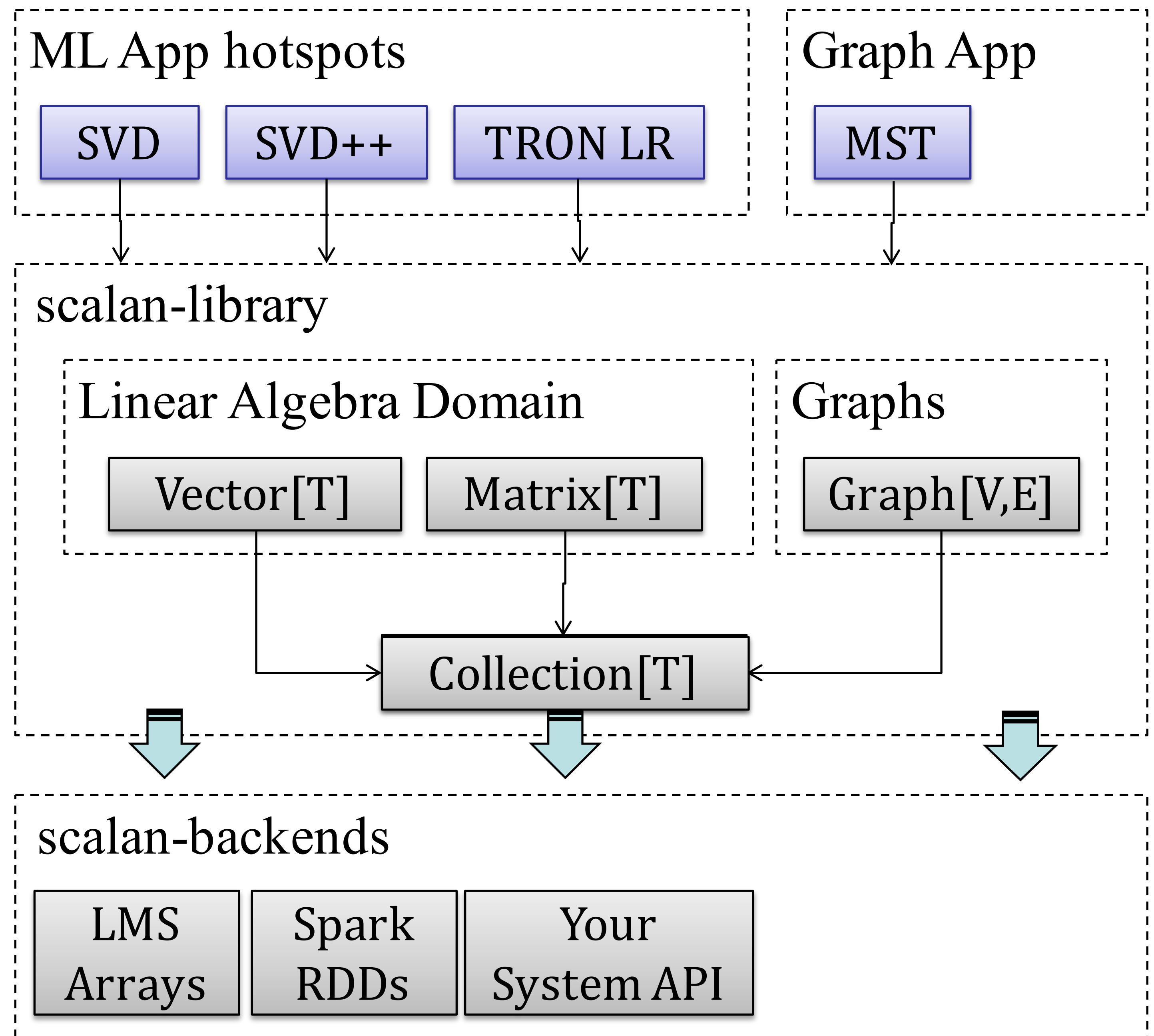
- **Мотивация**
- **Фреймворк для спец-оптимизации**
- **Как оно работает**
- **Как использовать**

Как мы используем Scalan: Архитектура приложения

Пишем алгоритмы используя абстракции оформленные как библиотеки на Scala. Места вызовов – отмечаем как хотспоты.

Реализуем абстракции через API нижнего уровня и запаковываем в «умные» библиотеки (smart library)

Системы нижнего уровня могут быть «умными» (LMS,Spark), но не обязательно



На чем испытывали

Алгоритмические ядра реализованы на Scalan и затем специализированы в 2 разные системы нижнего

Kernel		**Backends**	
Domain	**Name**	**LMS Arrays**	**Spark RDDs**
Machine Learning	Logistic Regression, Trust Region Newton Method (150 LoC)	✔	✔
Linear Algebra	Matrix Vector Multiplication (1 LoC)	✔	✔
	Matrix-Matrix Multiplication (3 LoC)	✔	✔
	SVD (100 LoC)	✔	✔
	SVD++(300 LoC)	✔	✔
Graphs	Minimum Spanning Tree (20 LoC)	✔	
Optimization	TabuSearch (100 LoC)	✔	✔
Functional Programming	Monad combinators (sequence, traverse, etc)	✔	
	Monads: State, Reader, Free, FreeState	✔	
	Compositional Application Architecture	✔	

SVD++ реализация для Spark

Cluster
1 master: 4 x 12 Intel(R) Xeon(R) CPU E7-4850 v2 @ 2.30GHz
4 workers: 12 Intel(R) Xeon(R) CPU E5-2609 v3 @ 1.90GHz
1000Base-T Ethernet
Spark 1.3.1 + GraphX
Dataset (http://www.netflixprize.com)
Users: 95313, Items: 3651

"Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model" by Yehuda Koren

Component	LoC
Collections	140
Matrices	180
SVD++	220
Generated Spark	1100

Cluster configuration	GraphX		Scalan	Performance gain
	50 iterations sqrtErr = 1.32	100 iteration sqrtErr = 1.23	sqrtErr = 1.05	
local[4]	307 sec	643 sec	183 sec	1.67 / 3.51
local[8]	242 sec	540 sec	117 sec	2.06 / 4.61
local[16]	206 sec	520 sec	80 sec	2.57 / 6.5
4-nodes cluster	430 sec	848 sec	120 sec	3.58 / 7.06

Get involved

1. Check out project at **github.com/scalan**

2. Ask questions on Google Group
   https://groups.google.com/d/forum/scalan

3. Follow us on twitter *@avslesarenko, @maxgekk, @alexey_r*

Thank you

github.com/scalan