



Московский государственный технический университет им. Н.Э. Баумана

# Кафедра «Системы обработки информации и управления» ИУ-5

## Методы анализа данных

Гапанюк Юрий Евгеньевич, к.т.н., доцент кафедры ИУ-5

# СЕМИНАР №1

## Введение в гибридные интеллектуальные информационные системы (ГИИС)

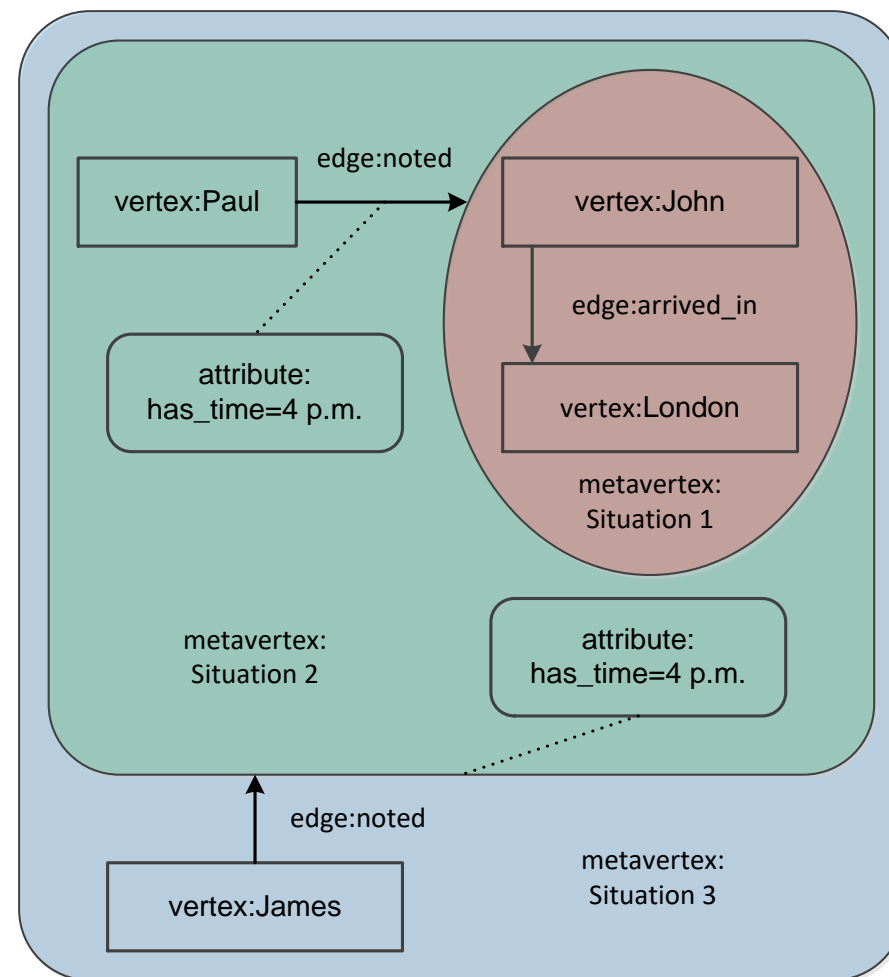
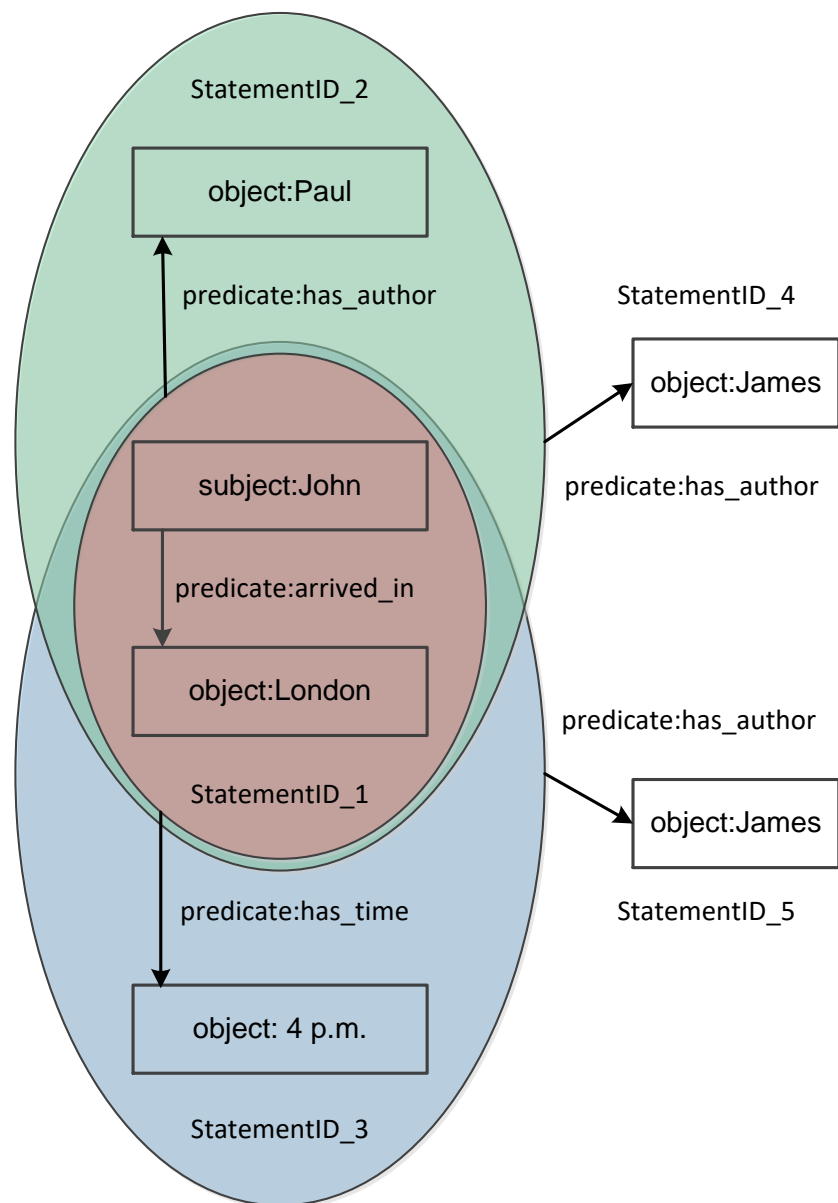
# План

1. Сравнение метаграфового подхода и RDF.
2. Основные подходы к построению хранилища на основе метаграфовой модели.
3. Разновидности метаграфовых агентов.
4. Примеры использования метаграфов и ГИИС в информационных системах.

# Сравнение метаграфового подхода и RDF

- Модель RDF широко используется для хранения знаний.
- Но проблема в том, что она плохо подходит для описания сложных вложенных контекстов.
- Рассмотрим пример описания ситуации на естественном языке: «James noted that Paul noted at 4 p.m. that John arrived in London».
- Представление ситуации в виде RDF-триплетов:
  1. *StatementID\_1 John arrived\_in London*
  2. *StatementID\_2 StatementID\_1 has\_author Paul*
  3. *StatementID\_3 StatementID\_1 has\_time “4p.m.”*
  4. *StatementID\_4 StatementID\_2 has\_author James*
  5. *StatementID\_5 StatementID\_3 has\_author James*

# Сравнение метаграфового подхода и RDF



# Сравнение метаграфового подхода и RDF

- RDF-триплет является слишком «мелким» элементом, который не позволяет описывать сложные контексты, требуется искусственно вводить вспомогательные триплеты.
- Метаграфовая модель позволяет описывать сложные контексты с использованием метавершин.

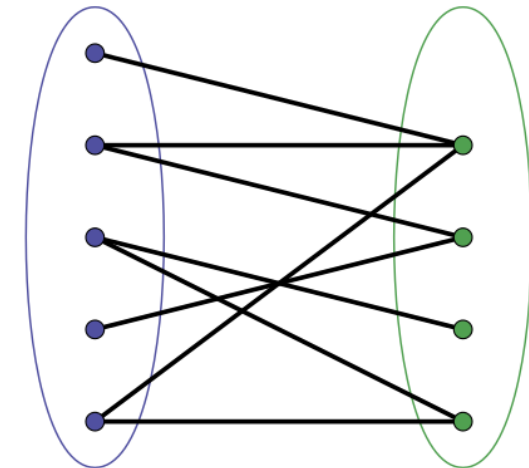
# Основные подходы к построению хранилища на основе метаграфовой модели

- Метаграфовая модель является аналогом «логической» модели СУБД и ей могут соответствовать различные «физические» модели.
- Мы рассмотрим три варианта «физической» модели:
  1. Модель на основе плоских графов.
  2. Документно-ориентированная модель.
  3. Реляционная модель.



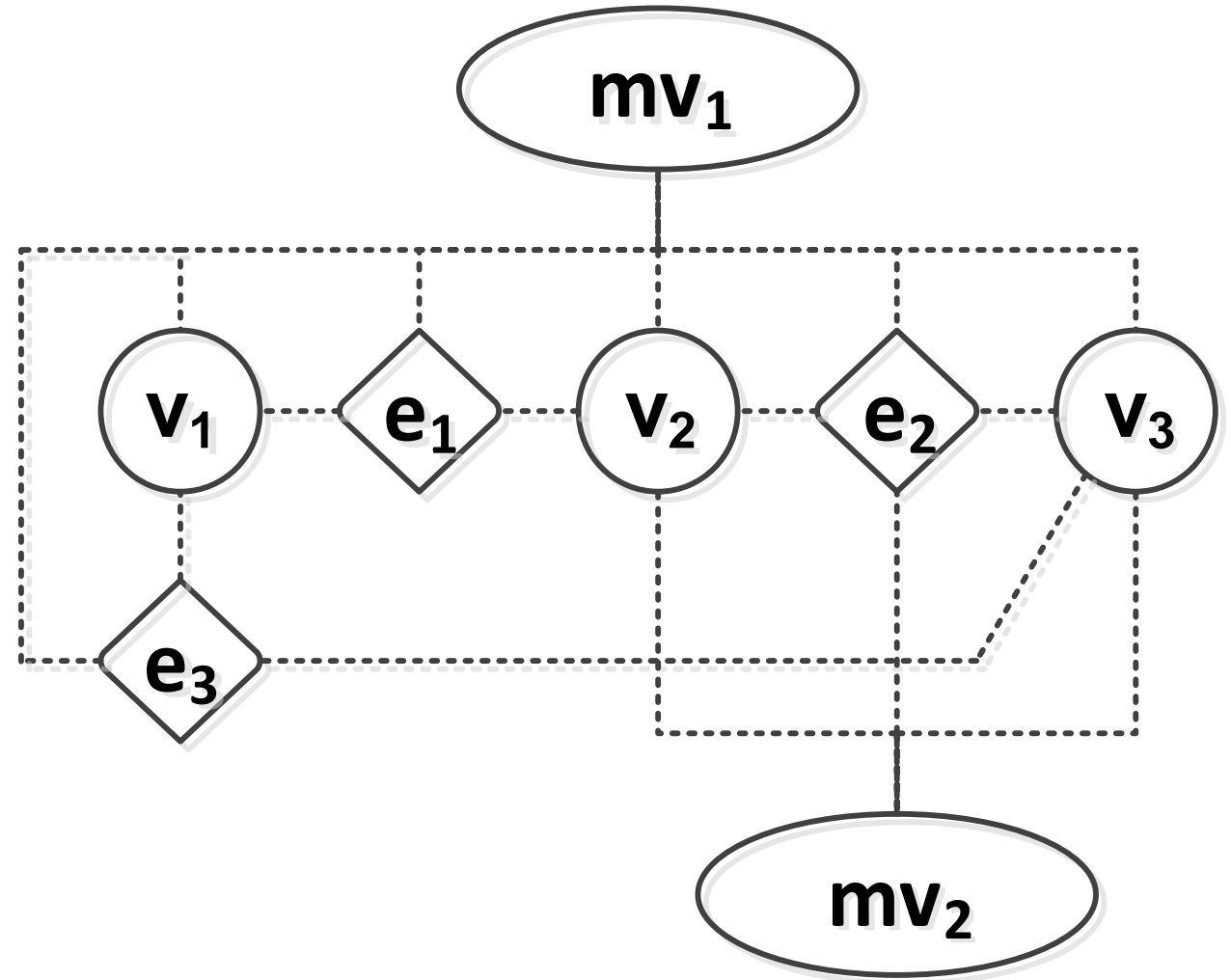
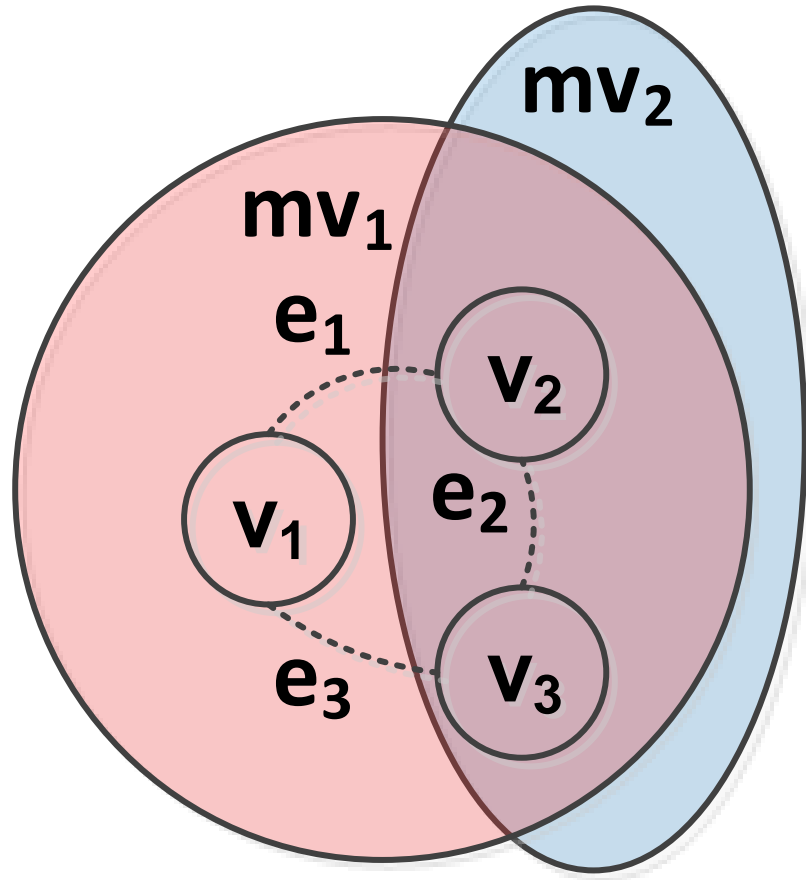
# Хранение – плоские графы

- Чтобы превратить холоническую (иерархическую) метаграфовую модель в плоскую графовую модель можно использовать подход на основе многодольных графов.
- Двудольный граф позволяет превратить вершины и ребра графа в подмножества вершин другого графа:  $FG = \langle FG^V, FG^E \rangle$ ,  $BFG = \langle BFG^{VERT}, BFG^{EDGE} \rangle$ ,  $BFG^{VERT} = \langle FG^{BV}, FG^{BE} \rangle$ ,  $FG^V \leftrightarrow FG^{BV}$ ,  $FG^E \leftrightarrow FG^{BE}$
- Для метаграфа используется трехдольный граф:
- $TFG = \langle TFG^{VERT}, TFG^{EDGE} \rangle$ ,  $TFG^{VERT} = \langle TFG^V, TFG^E, TFG^{MV} \rangle$ ,  
 $TFG^V \leftrightarrow MG^V$ ,  $TFG^E \leftrightarrow MG^E$ ,  $TFG^{MV} \leftrightarrow MG^{MV}$





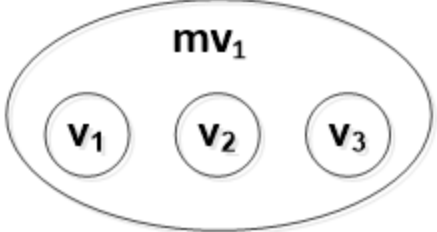


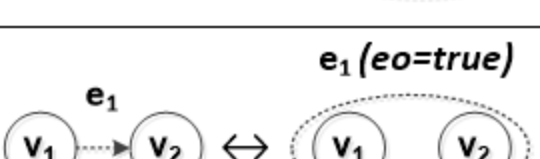
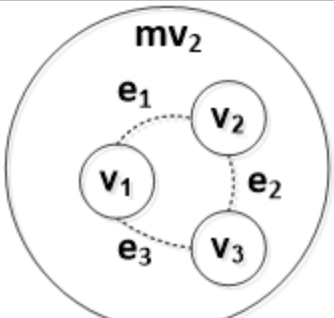
# Хранение – плоские графы (пример)

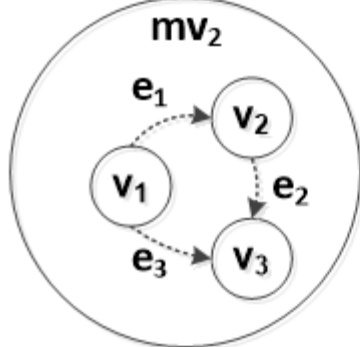
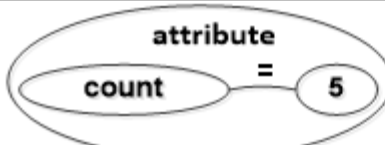
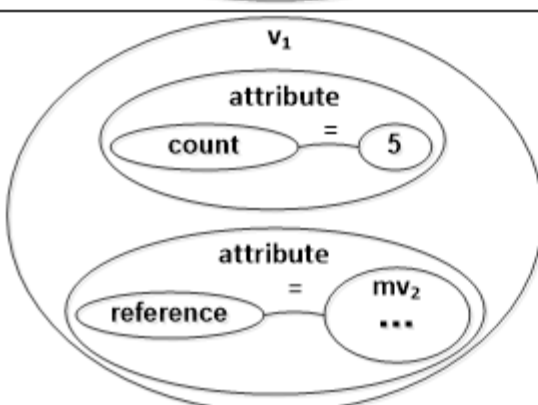


# Хранение – документы

- В качестве документоориентированного представления используется Prolog-подобное предикатное описание: *predicate(atom, \dots, key = value, \dots, predicate(\dots), \dots)*.
- Структуры, используемые в данном описании изоморфны структурам, применяемым в JSON-модели: иерархически организованные массивы и пары ключ-значение.
- Основные элементы метаграфовой модели могут быть отображены в предикатное описание:

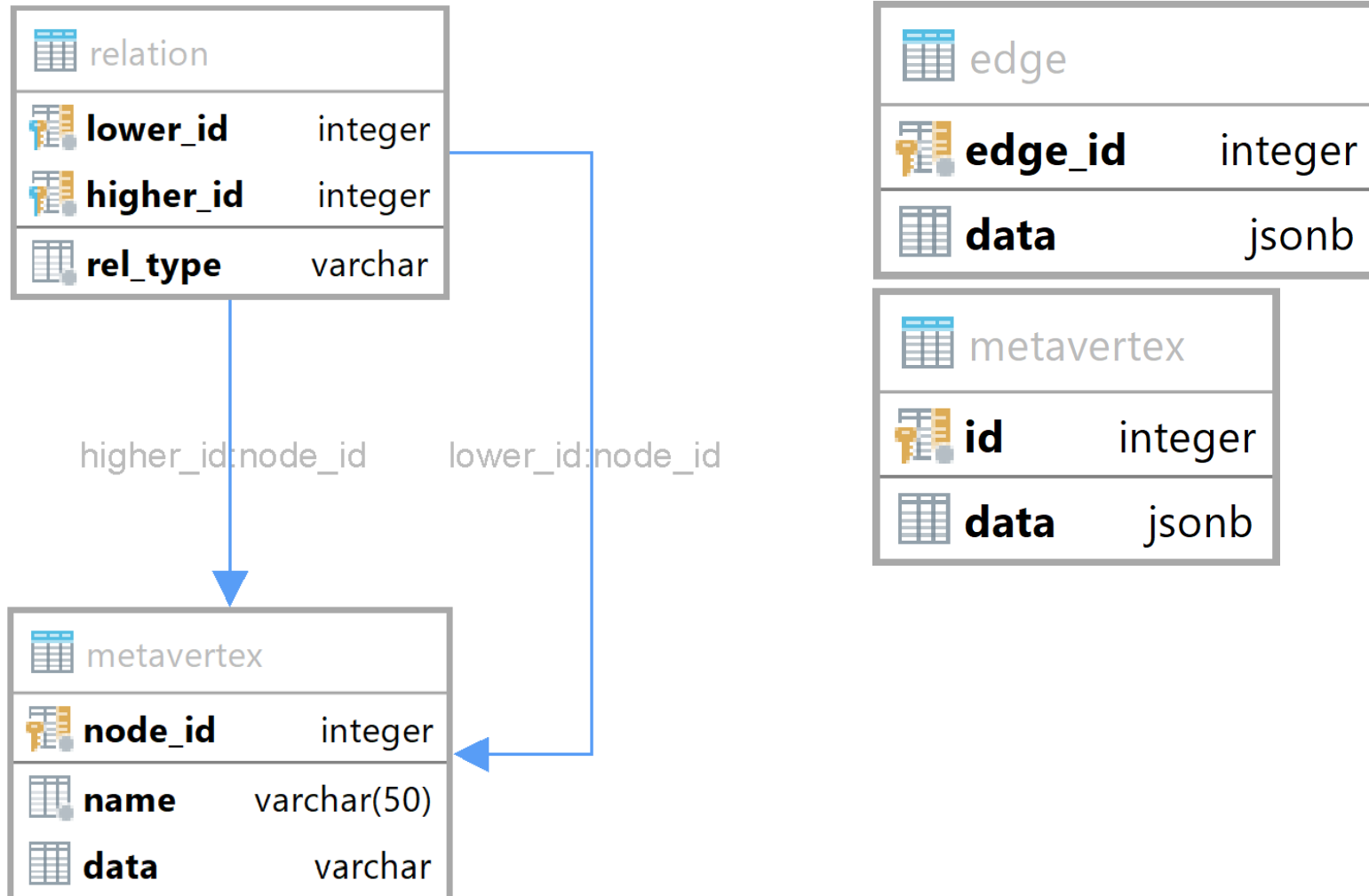
# Хранение – документы

	Metavertex(Name=mv <sub>1</sub> , v <sub>1</sub> , v <sub>2</sub> , v <sub>3</sub> )
	Edge(Name=e <sub>1</sub> , v <sub>1</sub> , v <sub>2</sub> )
	Edge(Name=e <sub>1</sub> , v <sub>1</sub> , v <sub>2</sub> , eo=false)
	1. Edge(Name=e <sub>1</sub> , v <sub>1</sub> , v <sub>2</sub> , eo=true) 2. Edge(Name=e <sub>1</sub> , v <sub>S</sub> =v <sub>1</sub> , v <sub>E</sub> =v <sub>2</sub> , eo=true)
	Metavertex(Name=mv <sub>2</sub> , v <sub>1</sub> , v <sub>2</sub> , v <sub>3</sub> , Edge (Name=e <sub>1</sub> , v <sub>1</sub> , v <sub>2</sub> ), Edge(Name=e <sub>2</sub> , v <sub>2</sub> , v <sub>3</sub> ), Edge(Name=e <sub>3</sub> , v <sub>1</sub> , v <sub>3</sub> ))

	Metavertex(Name=mv <sub>2</sub> , v <sub>1</sub> , v <sub>2</sub> , v <sub>3</sub> , Edge(Name=e <sub>1</sub> , v <sub>S</sub> =v <sub>1</sub> , v <sub>E</sub> =v <sub>2</sub> , eo=true), Edge(Name=e <sub>2</sub> , v <sub>S</sub> =v <sub>2</sub> , v <sub>E</sub> =v <sub>3</sub> , eo=true), Edge(Name=e <sub>3</sub> , v <sub>S</sub> =v <sub>1</sub> , v <sub>E</sub> =v <sub>3</sub> , eo=true))
	Attribute(count, 5)
	Vertex(Name=v <sub>1</sub> , Attribute(count, 5), Attribute(reference, mv <sub>2</sub> ))

# Хранение – реляционная модель

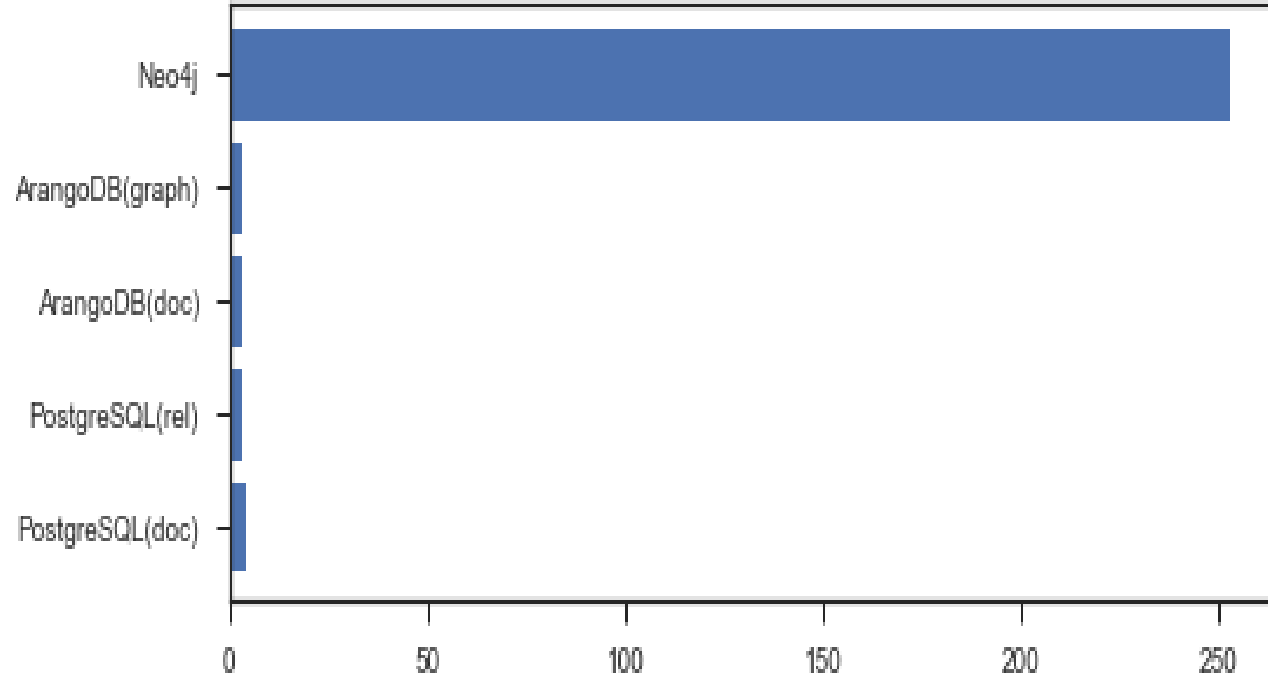
- Использование чистого реляционного подхода или документно-ориентированного хранилища, встроенного в реляционную СУБД.



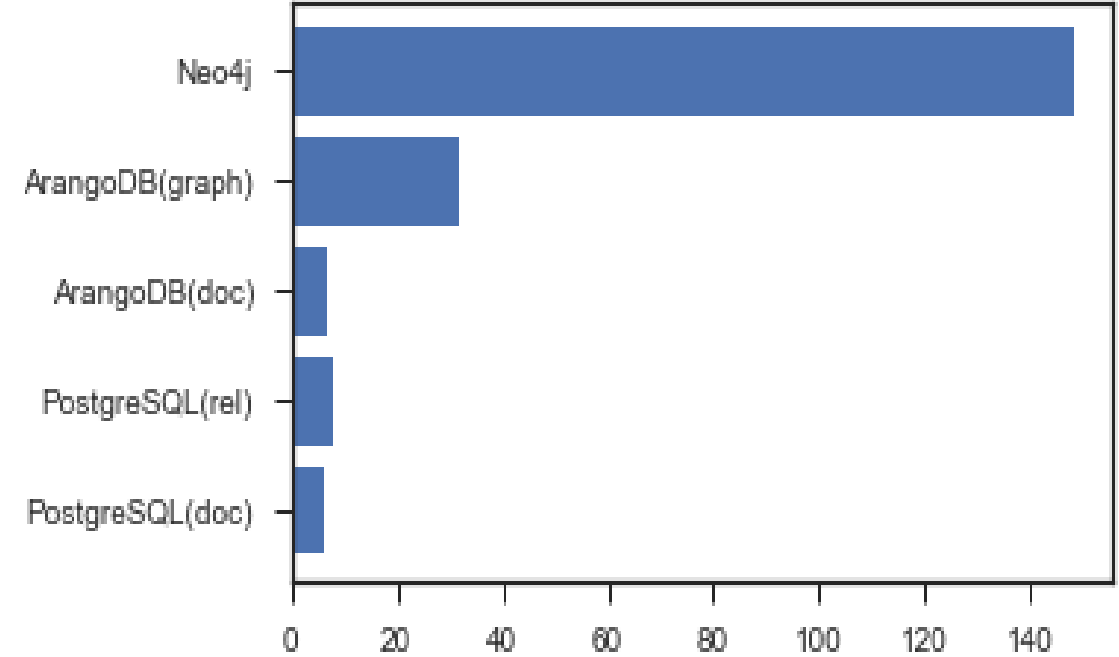
# Эксперименты

- Эксперименты проводились для следующих подходов:
  - Neo4j – использование СУБД Neo4j с плоской графовой моделью;
  - ArangoDB(graph) – использование СУБД ArangoDB с плоской графовой моделью;
  - ArangoDB(doc) – использование СУБД ArangoDB с документно-ориентированной моделью;
  - PostgreSQL(rel) – использование СУБД PostgreSQL (классическая реляционная модель);
  - PostgreSQL(doc) – использование документно-ориентированного хранилища на основе СУБД PostgreSQL.
- Характеристики сервера: Intel Xeon E7-4830 2.2 GHz, 4 Gb RAM, 1 Tb SSD, OS Ubuntu 16.04.
- Время в миллисекундах, меньшее значение лучше.

# Эксперименты

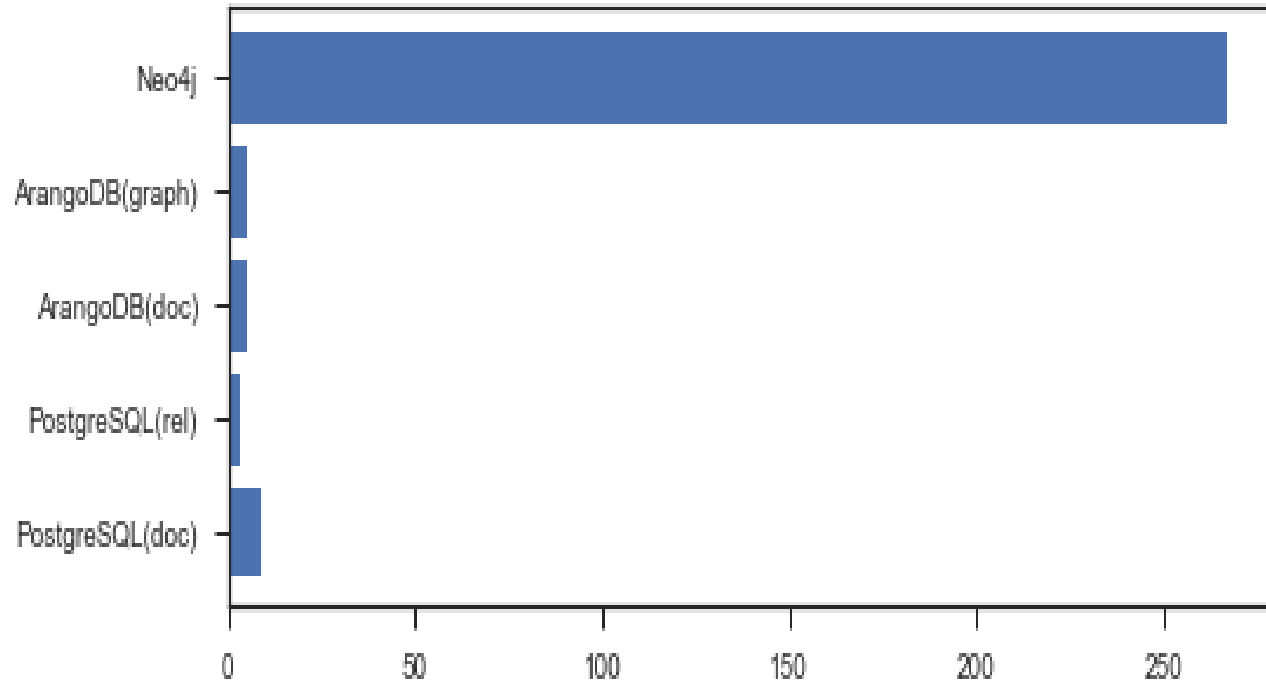


Добавление вершины в метаграф

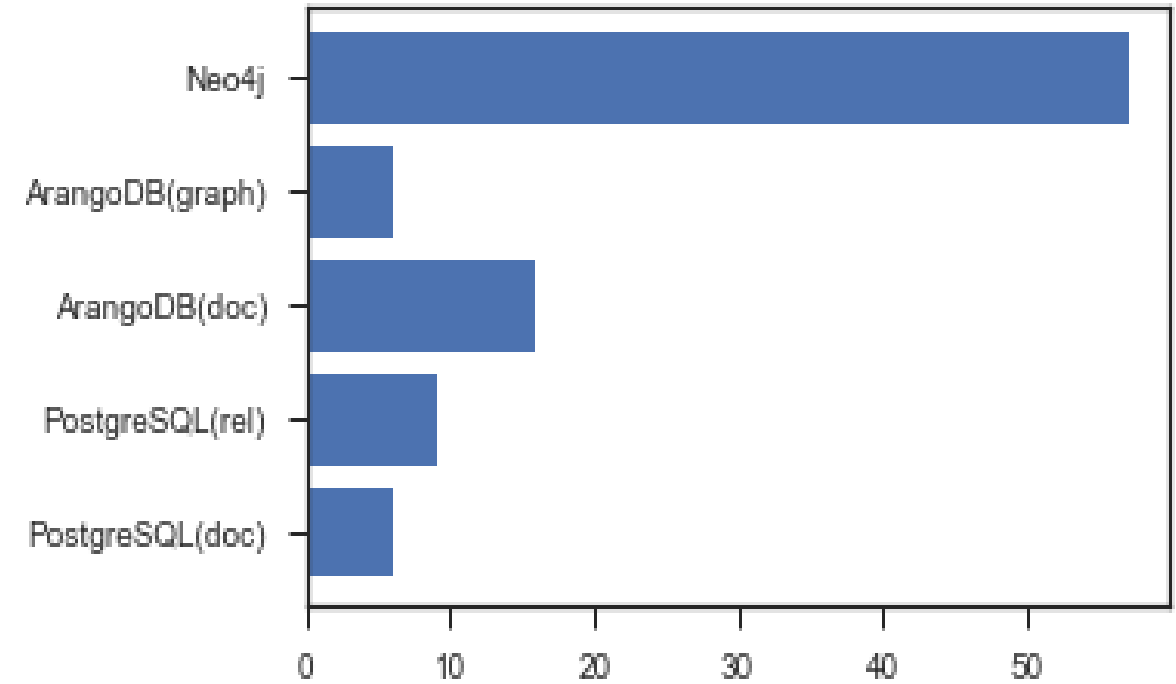


Добавление ребра в метаграф

# Эксперименты



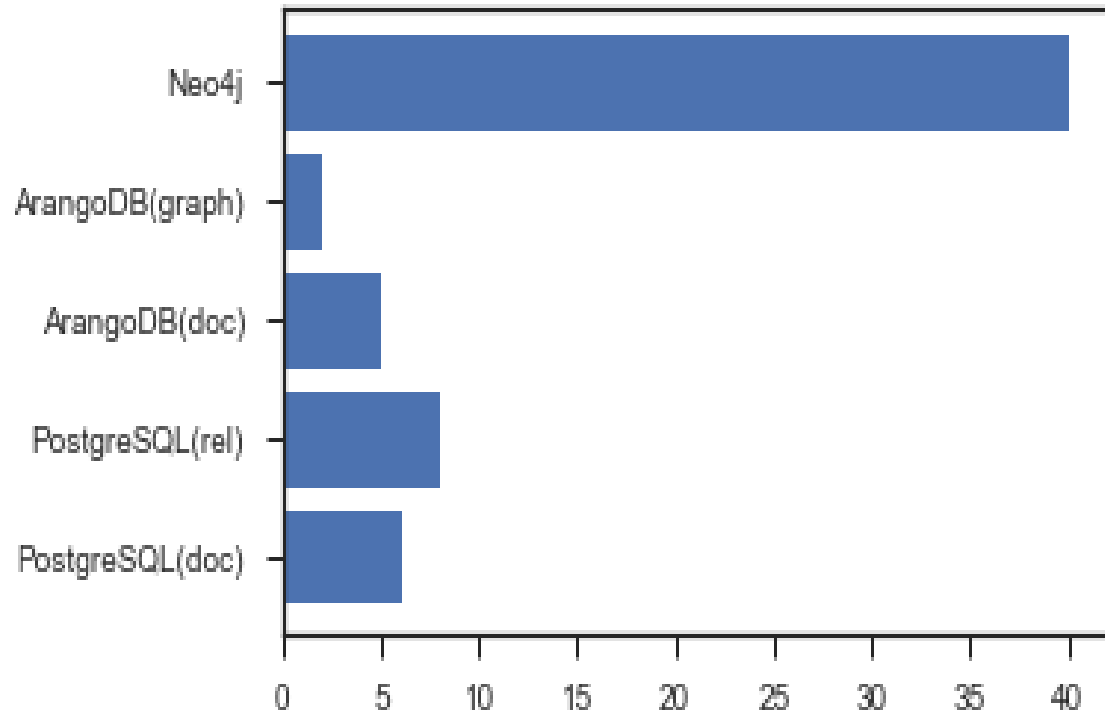
Редактирование вершины в метаграфе



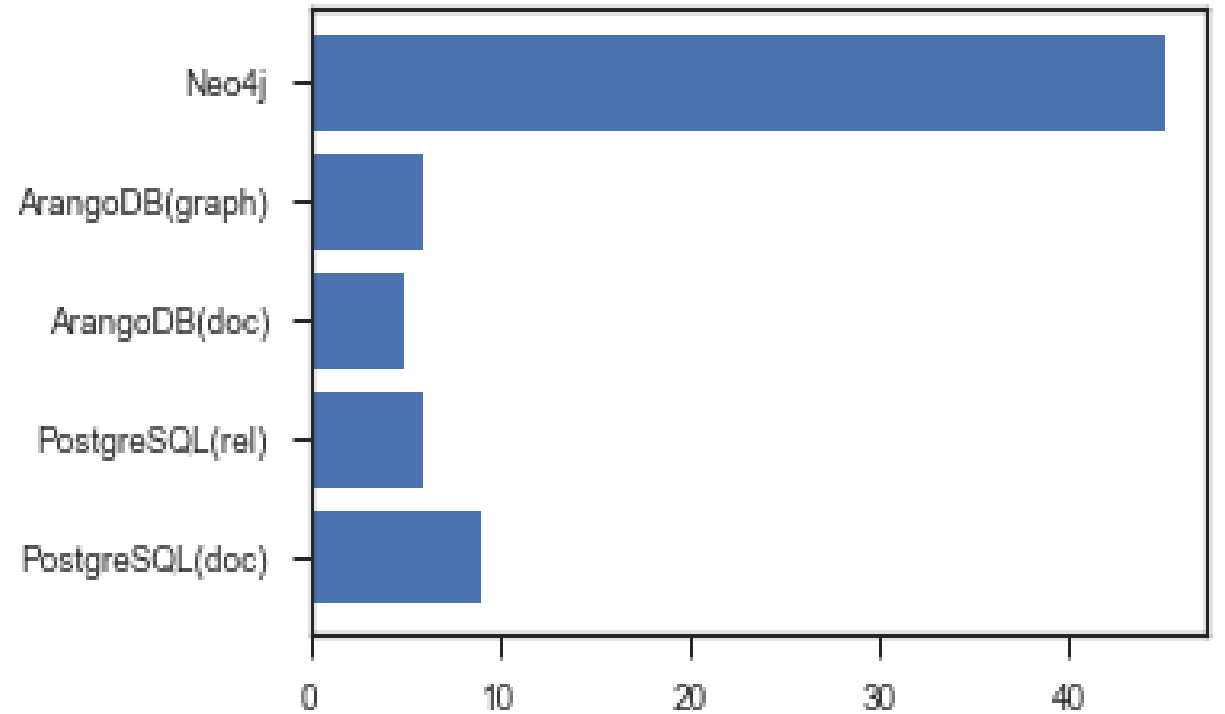
Удаление ребра из метаграфа



# Эксперименты

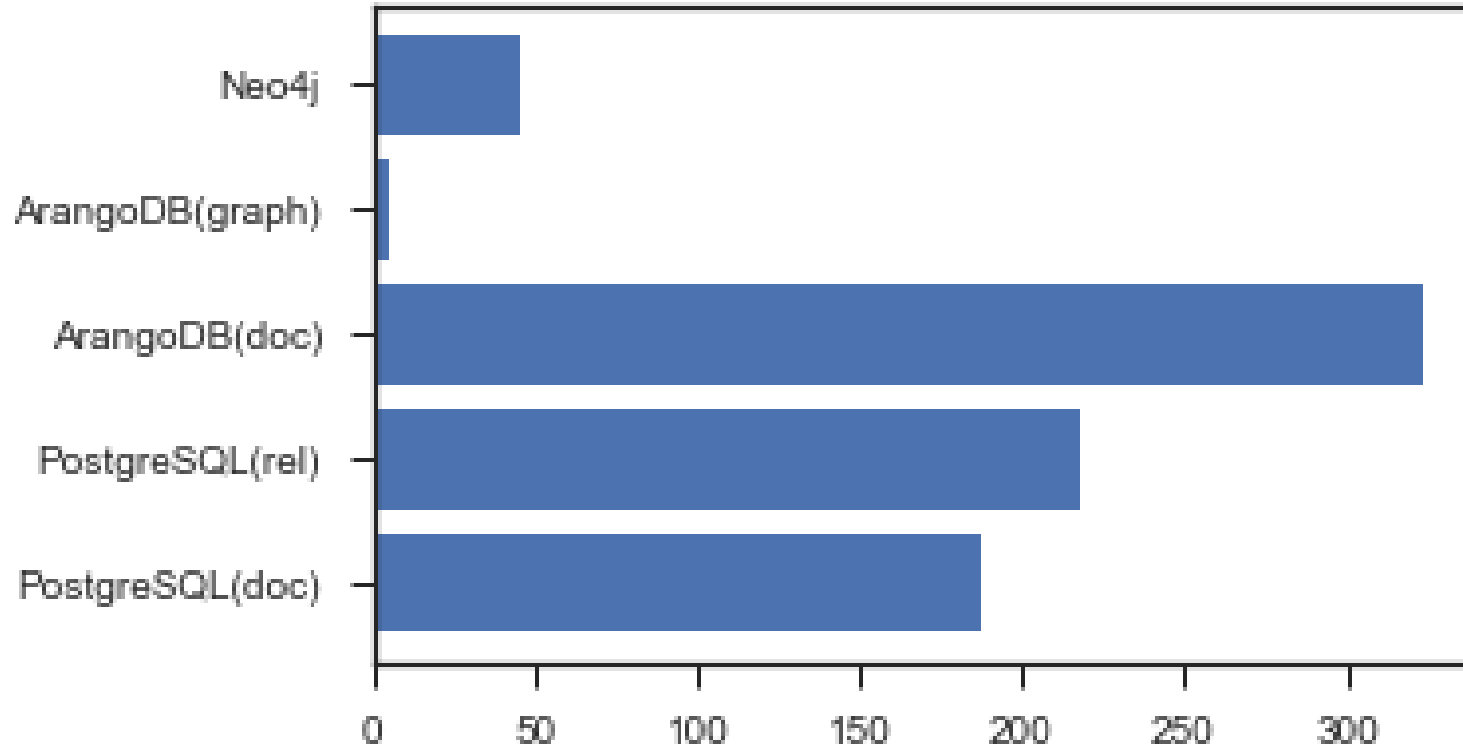


Добавление вершины в метавершину



Удаление вершины из метавершины

# Эксперименты



Выборка иерархических данных из 100 связанных метавершин

# Выводы (по результатам экспериментов с хранением метаграфовой модели)

- В случае добавления, обновления и удаления данных наиболее эффективным вариантом является использование PostgreSQL. Но в случае выборки иерархических данных графовые СУБД показывают значительно лучшие результаты.
- Для графовых СУБД (как для ArangoDB так и для Neo4j) время выполнения запросов на получение иерархических данных сопоставимо с временем выполнения запросов на добавление, обновление и удаление данных.
- В PostgreSQL добавление, обновление и удаление данных в целом производятся быстрее, чем в графовых СУБД или за сопоставимое время. Но при этом время выполнения запросов на получение иерархических данных во много раз больше времени добавления, обновления и удаления данных.
- В целом наиболее производительным вариантом является использование СУБД ArangoDB с плоской графовой моделью.

# Холоническая МАС для реализации активности

- Метаграф является пассивной структурой данных. Как реализовать активность при моделировании сложной сети с эмерджентностью?
- Как выполнить рассмотренные требования 1, 2, 3?
- Для реализации требования 1 предлагается использовать два вида агентов: агент-функцию и метаграфовый агент.
- Для реализации требования 2 предлагается использовать контейнерный агент.
- Для реализации требования 3 предлагается использовать динамический метаграфовый агент.
- Рассмотрим данные виды агентов более подробно.

# Агент-функция

- Агент-функция:

$$ag^F = \langle MG_{IN}, MG_{OUT}, AST \rangle,$$

- где  $ag^F$  – агент-функция;  $MG_{IN}$  – метаграф, который выполняет роль входного параметра агента-функции;  $MG_{OUT}$  – метаграф, который выполняет роль выходного параметра агента-функции;  $AST$  – абстрактное синтаксическое дерево агента-функции, которое может быть представлено в виде метаграфа.

# Метаграфовый агент (определение)

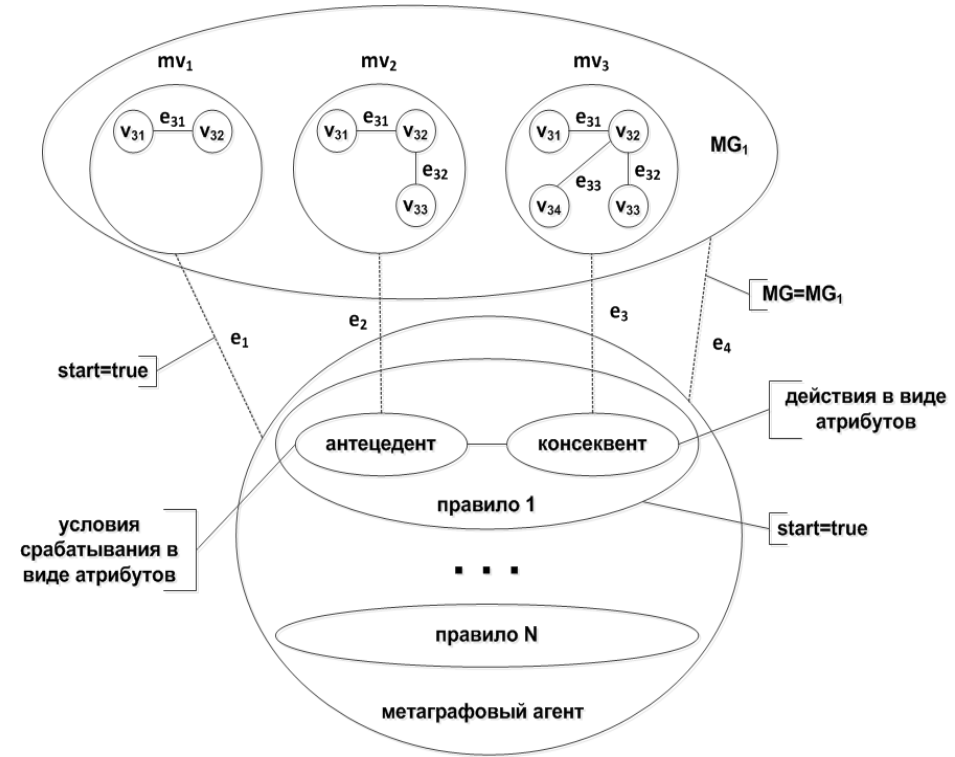
- Метаграфовый агент:

$$ag^M = \langle MG, R, AG^{ST}, \{ag_i^M\} \rangle, R = \{r_j\},$$

- где  $ag^M$  – метаграфовый агент;  $MG$  – метаграф, на основе которого выполняются правила агента;  $R$  – набор правил (множество правил  $r_j$ );  $AG^{ST}$  – стартовое условие выполнения агента (фрагмент метаграфа, который используется для стартовой проверки правил, или стартовое правило).
- При этом агент  $ag^M$  содержит множество вложенных агентов  $ag_i^M$  что соответствует принципам организации холонической многоагентной системы. Агент верхнего уровня может активизировать агентов нижнего уровня для решения подзадач.
- Структура правила метаграфового агента:

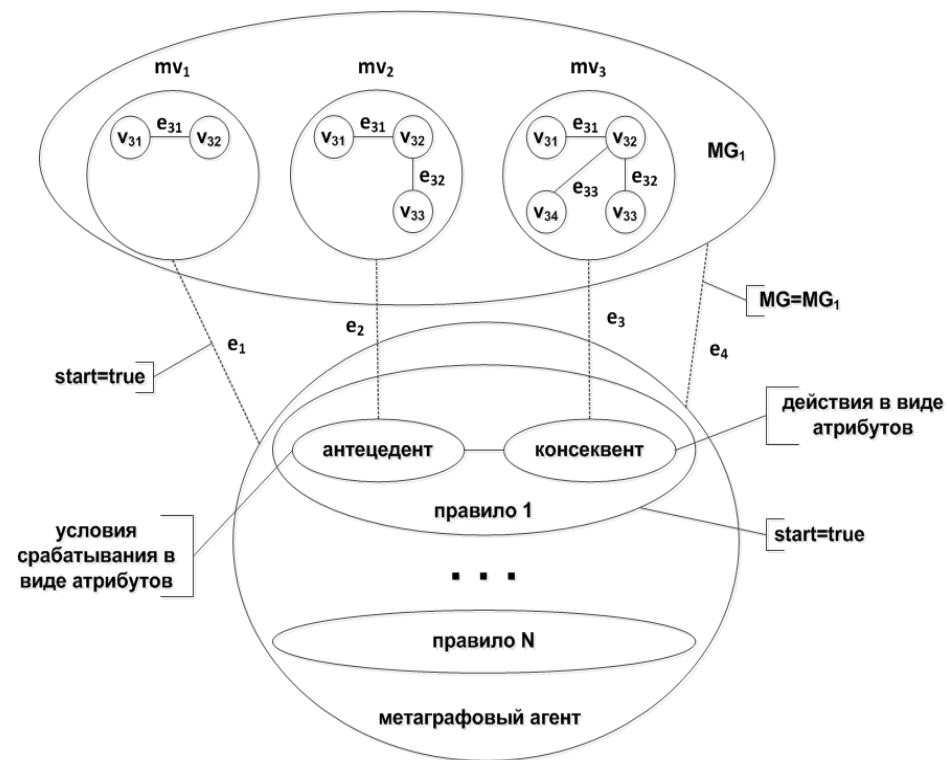
$$r_i : MG_j \rightarrow OP^{MG},$$

- где  $r_i$  – правило;  $MG_j$  – фрагмент метаграфа, на основе которого выполняется правило;  $OP^{MG}$  – множество действий, выполняемых над метаграфом.
- Антецедементом правила является фрагмент метаграфа, консеквентом правила является множество действий, выполняемых над метаграфом.



# Метаграфовый агент (правила)

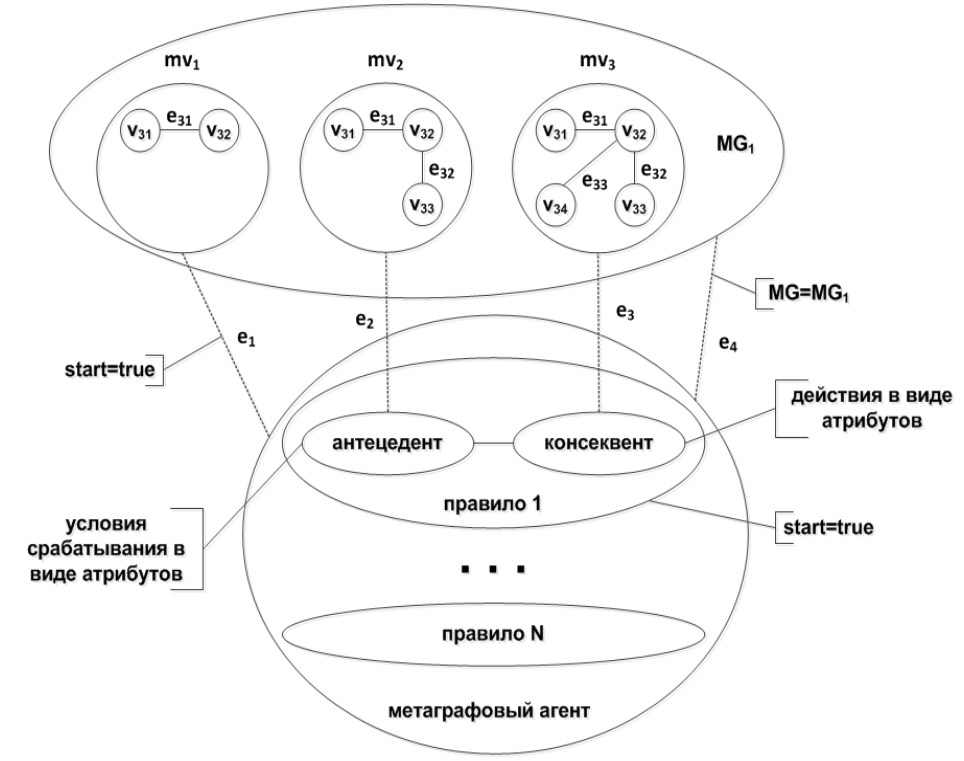
- Правила метаграфового агента можно разделить на замкнутые и разомкнутые.
- Разомкнутые правила не меняют в правой части правила фрагмент метаграфа, относящийся к левой части правила. Можно разделить входной и выходной фрагменты метаграфа. Данные правила являются аналогом шаблона, который порождает выходной метаграф на основе входного.
- Замкнутые правила меняют в правой части правила фрагмент метаграфа, относящийся к левой части правила. Изменение метаграфа в правой части правил заставляет срабатывать левые части других правил. Но при этом некорректно разработанные замкнутые правила могут привести к заикливанию метаграфового агента.
- Таким образом, метаграфовый агент позволяет генерировать один метаграф на основе другого (с использованием разомкнутых правил) или модифицировать метаграф (с использованием замкнутых правил).





# Метаграфовый агент (самоотображаемость)

- Особенностью метаграфового агента является то, что его структура может быть представлена в виде фрагмента метаграфа. Это соответствует принципу самоотображаемости (англ. homoiconicity) в языках программирования. Самоотображаемость – это способность языка программирования анализировать программу на этом языке как структуру данных этого языка.
- Структура агента может быть изменена как данные с помощью правил агентов верхнего уровня.
- Метаграфовый агент представлен в виде метавершины метаграфа. В соответствии с определением он связан с метаграфом  $MG_1$ , на основе которого выполняются правила агента. Данная связь показана с помощью ребра  $e_4$ .
- Метаграфовый агент содержит множество вложенных метавершин, соответствующих правилам (правило 1 – правило N). В данном примере с антецедентом правила связана метавершина данных  $mv_2$ , что показано ребром  $e_2$ , а с консеквентом правила связана метавершина данных  $mv_3$ , что показано ребром  $e_3$ . Условия срабатывания задаются в виде атрибутов соответствующих вершин.
- Стартовое условие выполнения агента задается с помощью атрибута «start=true». Если стартовое условие задается в виде стартового правила, то данным атрибутом помечается метавершина соответствующего правила, в данном примере это правило 1. Если стартовое условие задается в виде стартового фрагмента метаграфа, который используется для стартовой проверки правил, то атрибутом «start=true» помечается ребро, которое связывает стартовый фрагмент метаграфа с метавершиной агента, в данном примере это ребро  $e_1$ .



# Контейнерный агент

- Контейнерный агент:

$$ag^C = MG, v_i \equiv ag_i, v_i \in V, mv_i \equiv ag_i, mv_i \in MV,$$

- где  $ag^C$  – контейнерный агент;  $MG$  – метаграф;  $v_i$  – вершина метаграфа;  $ag_i$  – агент;  $V$  – множество вершин метаграфа;  $mv_i$  – метавершина метаграфа;  $MV$  – множество метавершин метаграфа.
- Контейнерный агент, представляет собой метаграф, вершины и метавершины которого являются агентами.

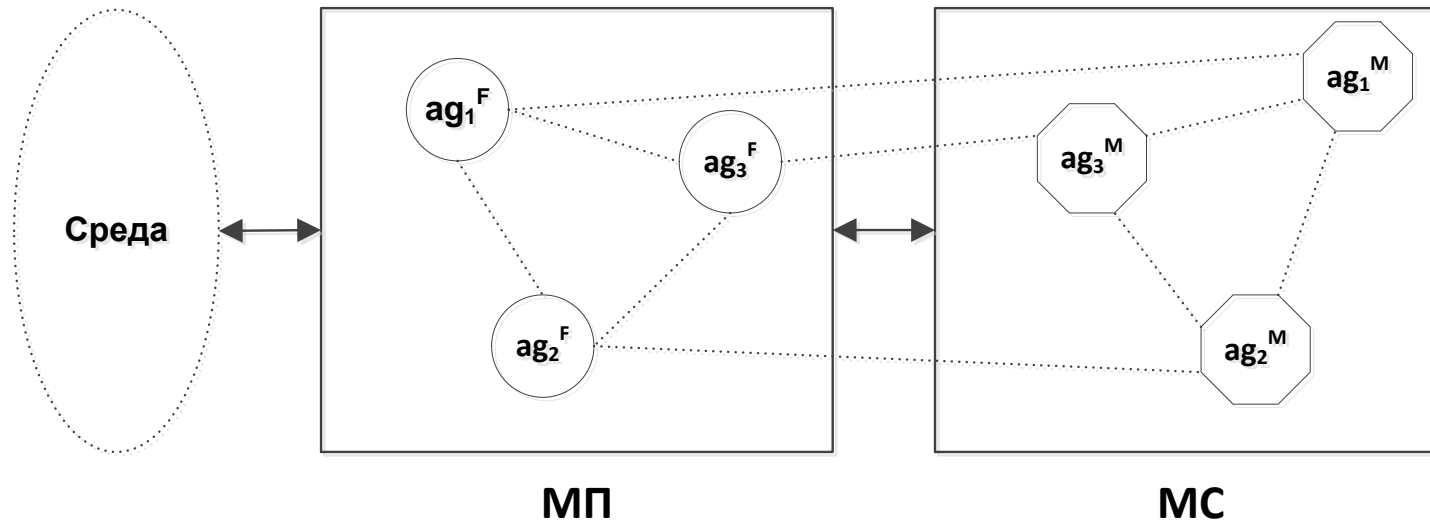
# Динамический метаграфовый агент

- Динамический метаграфовый агент:

$$ag^{MD} = \langle (ag^C \cup ag^{MD}), R, AG^{ST} \rangle, R = \{r_j\},$$

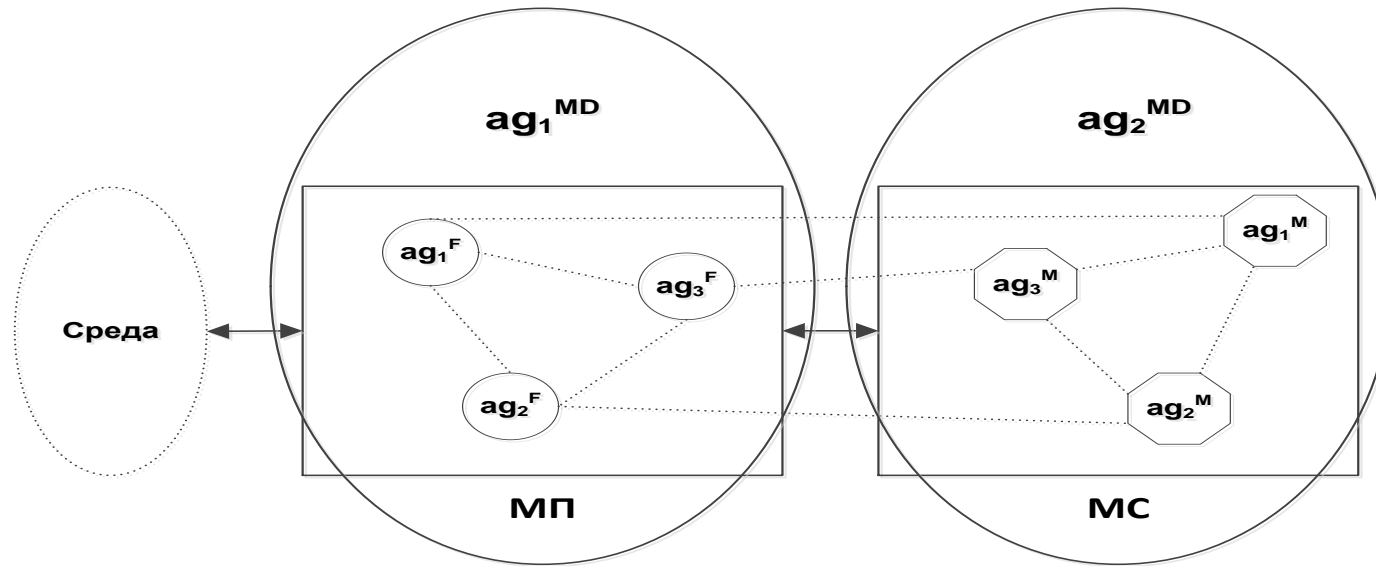
- где  $ag^{MD}$  – динамический метаграфовый агент;  $ag^C$  – контейнерный агент, на метаграфе которого выполняются правила агента;  $R$  – набор правил (множество правил  $r_j$ );  $AG^{ST}$  – стартовое условие выполнения агента (фрагмент метаграфа, который используется для стартовой проверки правил, или стартовое правило).
- Правила обработки динамического метаграфового агента выполняются не на метаграфе данных и знаний, а на метаграфе агентов для заданного контейнерного агента.
- По определению контейнерный агент включает все рассмотренные ранее виды агентов: агенты-функции и метаграфовые агенты. Поэтому динамический метаграфовый агент может изменять все виды агентов.
- Определение данного агента использует тот факт, что в предлагаемой модели все агенты являются метаграфами, поэтому любые элементы структуры агентов доступны для обработки агентами верхнего уровня. Эта особенность является аналогом свойства «самоотображаемости» в традиционных языках программирования.
- Отметим, что данное определение является рекурсивным. Динамические метаграфовые агенты первого уровня могут обрабатывать статические контейнерные агенты, метаграфовые агенты второго уровня могут обрабатывать метаграфовые агенты первого уровня и так далее. По мере необходимости систему можно надстраивать требуемыми уровнями динамики.
- В зависимости от условий динамический метаграфовый агент может решать следующие задачи:
  - первичное развертывание, создание, системы агентов более низкого уровня;
  - изменение системы нижнего уровня (изменение внутренней структуры агентов, изменение связей между агентами, удаление агентов).

# Пример 1. Статическая структура ГИИС



- На рисунке представлена система, МП которой является нейронной сетью, а МС построен на основе обработки правил.
- Агенты-нейроны показаны в виде окружностей, а метаграфовые агенты обработки данных в виде восьмиугольников. МП и МС выполняют роль контейнерных агентов. В данном примере используются одноуровневые контейнеры, однако, возможно использование произвольной вложенности контейнеров.
- Отметим, что вся система холонических агентов представляет собой метаграф, при этом каждый агент также является метаграфом.

# Пример 2. Динамическая структура ГИИС



- По сравнению с предыдущим рисунком добавились два динамических метаграфовых агента.
- Агент, отвечающий за МП, может изменять структуру самоорганизующейся нейронной сети. Или может применять эволюционные методы для оптимизации конфигурации нейронной сети.
- Агент, отвечающий за МС, может изменять связи между агентами для решения задачи автоматизированного планирования. Или может применять эволюционные методы для оптимизации конфигурации агентов.
- На рисунке показаны только динамические метаграфовые агенты первого уровня, однако, количество таких уровней не ограничено. Над показанными динамическими метаграфовыми агентами могут быть надстроены динамические метаграфовые агенты более высоких уровней.

# Пример. Моделирование работы нейронной сети с использованием метаграфового подхода

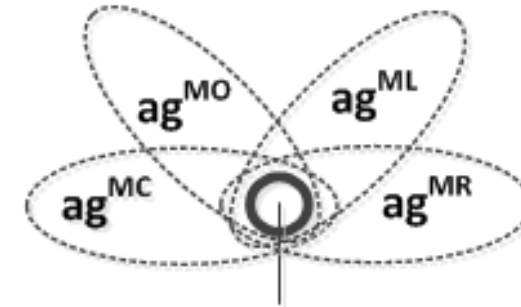
- На следующих 4 слайдах показан пример моделирования нейронной сети с использованием метаграфового подхода.
- Слайд А – Общая схема функционирования нейронной сети на основе метаграфовых агентов. Данная схема не рассматривает детально метаграфовое представление структуры отдельных нейронов, что рассматривается на слайдах Б, В, Г.
- Слайд Б – Описание функционирования персептрона на основе метаграфового подхода. Персептрон является базовым элементом для построения простых нейронных сетей.
- Слайды В,Г – Варианты описания глубокой нейронной сети с использованием различных стратегий регуляризации.
- Исследования в области метаграфового описания нейронных сетей продолжаются.

# А) Описание нейронной сети с помощью метаграфовых агентов

- Представление нейронной сети в виде метаграфа может быть реализовано с помощью метаграфовых агентов.
- С использованием метаграфовых агентов может быть смоделирована работа нейросети в различных режимах.
- В примере используются следующие динамические метаграфовые агенты:

1.  $ag^{MC}$  – агент создания нейросети; 2.  $ag^{MO}$  – агент изменения нейросети;
3.  $ag^{ML}$  – агент обучения нейросети; 4.  $ag^{MR}$  – агент запуска нейросети.

- Агент создания нейросети ( $ag^{MC}$ ) реализует правила создания начальной топологии нейросети. Данный агент содержит как правила создания отдельных нейронов, так и правила соединения нейронов в нейросеть, в частности создает AST агентов-функций, моделирующих отдельные нейроны.
- Агент изменения нейросети ( $ag^{MO}$ ) содержит правила изменения топологии сети в процессе работы. Это особенно важно для сетей с изменяемой топологией, таких как SOINN.
- Агент обучения нейросети ( $ag^{ML}$ ) реализует один из алгоритмов обучения. При этом в результате обучения измененные значения весов записываются в метаграфовое представление нейросети. Возможна реализация нескольких алгоритмов обучения с использованием различных наборов правил для агента  $ag^{ML}$ .
- Агент запуска нейросети ( $ag^{MR}$ ) реализует запуск и работу обученной нейросети в штатном режиме.
- Отметим, что агенты могут работать как независимо, так и совместно. Например, при обучении сети SOINN агент  $ag^{ML}$  может вызывать правила агента  $ag^{MO}$  для изменения топологии в процессе обучения.
- Каждый агент на основе заложенных в него правил фактически реализует специфическую программную «машину». Использование метаграфового подхода позволяет реализовать принцип «мультимашинности», когда несколько агентов с различными целями реализуют различные действия на одной и той же структуре данных.

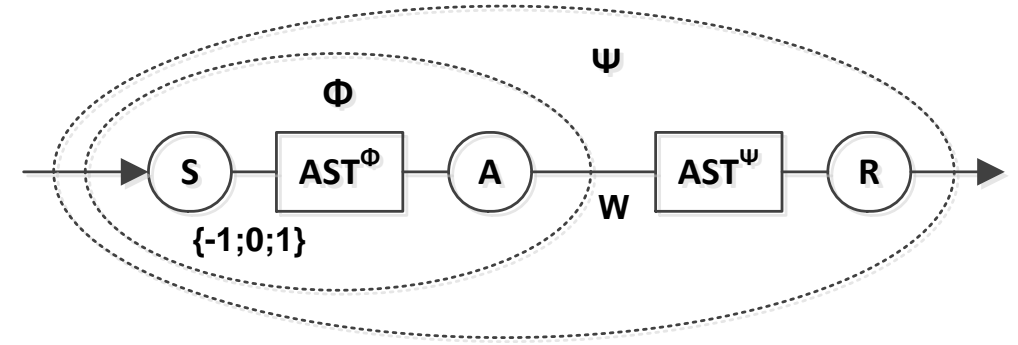


**Представление нейронной  
сети в виде метаграфа**



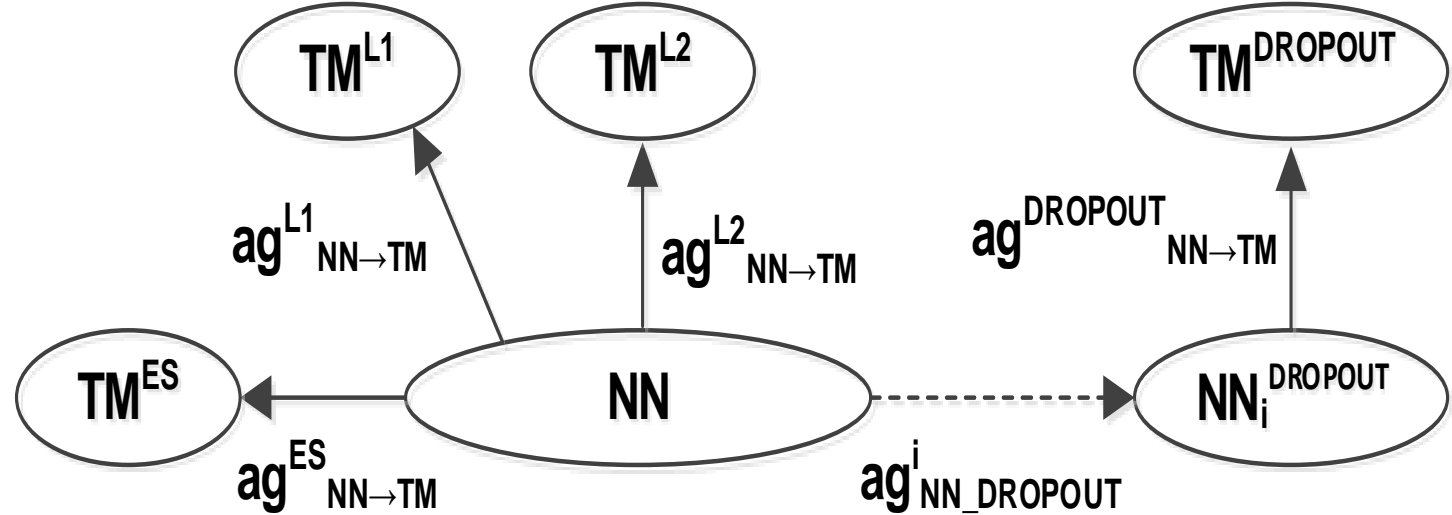
## Б) Метаграфовое представление персептрона

- В соответствии с моделью Розенблатта, персептрон состоит из трех уровней: S, A и R.
- Слой сенсоров (S) представляет собой набор входных сигналов. Ассоциативный слой (A) включает набор промежуточных элементов, которые активизируются, если одновременно активизируется некоторый набор (образ) входных сигналов. Сумматор (R) активизируется, если одновременно активизируется некоторый набор A-элементов.
- В соответствии с обозначениями, принятыми в работе М. Минского и С. Пайперта, значение сигнала на A-элементе может быть представлено в виде предиката  $\phi(S)$  а значение сигнала на сумматоре в виде предиката  $\psi(A, W)$ , где  $W$  – вектор весов. Под предикатом здесь понимается функция, принимающая только два значения «0» и «1».
- В зависимости от конкретного вида персептрона вид предикатов  $\phi(S)$  и  $\psi(A, W)$  может быть различным. Как правило, с помощью предиката  $\phi(S)$  проверяется, что суммарный входной сигнал от сенсоров не превышает некоторый порог. Также с помощью предиката  $\psi(A, W)$  проверяется, что взвешенная сумма от A-элементов не превышает некоторого порога.
- В нашем случае конкретный вид предикатов не важен, важно то, что предикаты являются обычными функциональными зависимостями, которые на программном уровне могут быть представлены в виде абстрактного синтаксического дерева и следовательно могут быть смоделированы агентами-функциями  $\varphi^F = \langle S, A, AST^\phi \rangle$ ,  $\psi^F = \langle \langle \{\varphi^F\}, W \rangle, R, AST^\psi \rangle$ . Структура агентов-функций представлена на рисунке.
- Описание функций может содержать различные параметры, например пороги, но мы предполагаем, что эти параметры входят в описание абстрактного синтаксического дерева и могут быть переписаны с использованием метаграфовых агентов верхнего уровня.



# В) Описание регуляризации с помощью метаграфов - 1

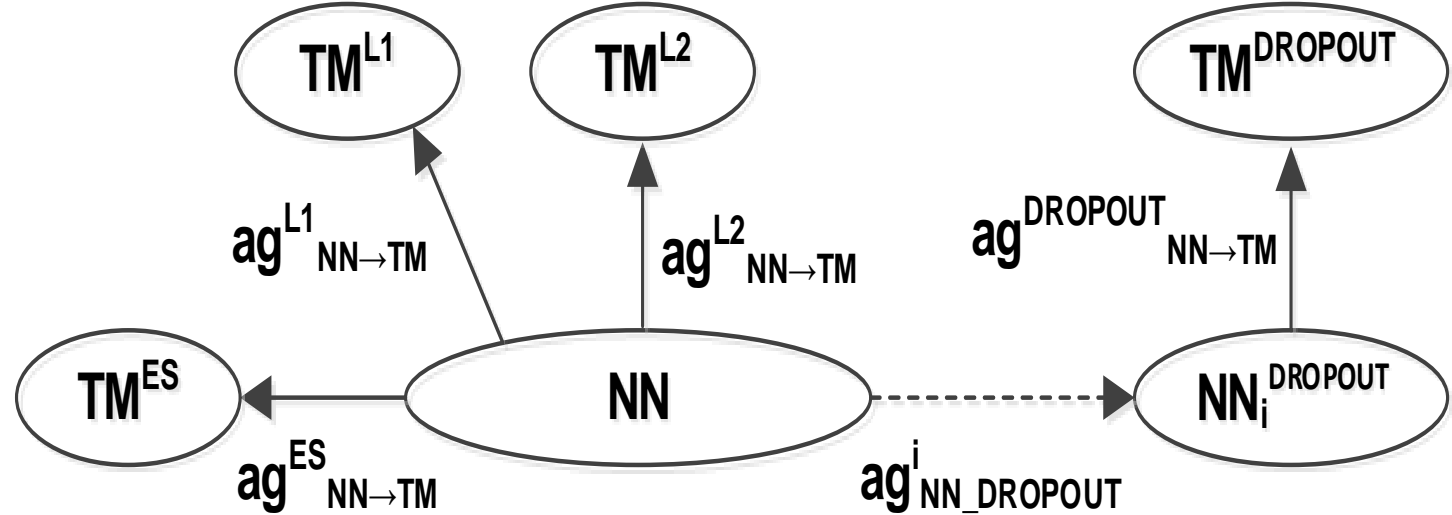
- Как и в случае отдельного персептрона, глубокая нейронная сеть может быть создана с использованием метаграфовых агентов. В зависимости от целей моделирования, нейрон может быть представлен в виде простой вершины или сложной метавершины, содержащей детализированное описание нейрона.
- На шаге создания нейросети создается структура “Neural Network” (NN), которая является плоским графом из нейронов, соединенных связями. Нейрон может быть описан как персептрон или аналогичным образом. Но при этом внутреннее представление нейрона может быть раскрыто в форме метавершины. Также в форме метавершины может быть представлен каждый уровень нейросети.



- В режиме обучения создается структура “Training Metagraph” (TM), изоморфная NN. Для каждой вершины-нейрона в NN создается метавершина в TM, связи сохраняются. Для создания TM на основе NN используется агент-функция  $ag_{NN \rightarrow TM}$ .
- TM является активным метаграфом, в котором с метаграфом данных связан метаграфовый агент  $ag_{TM}$ , отвечающий за обучение сети. Результаты обучения сохраняются в TM.
- Поскольку агенты могут быть представлены в виде метаграфов, то агент  $ag_{TM}$  создается с помощью агента  $ag_{NN \rightarrow TM}$ .

# Г) Описание регуляризации с помощью метаграфов - 2

- Агент  $ag_{TM}$  может использовать различные стратегии регуляризации.
- Для NN может быть создано несколько структур TM, использующих различные стратегии регуляризации: L1, L2, ES (Early Stopping), dropout (обозначены верхними индексами).
- В случае использования стратегий L1, L2, ES не требуется изменения структуры графа нейросети в процессе обучения. Соответствующие агенты являются агентами-функциями.



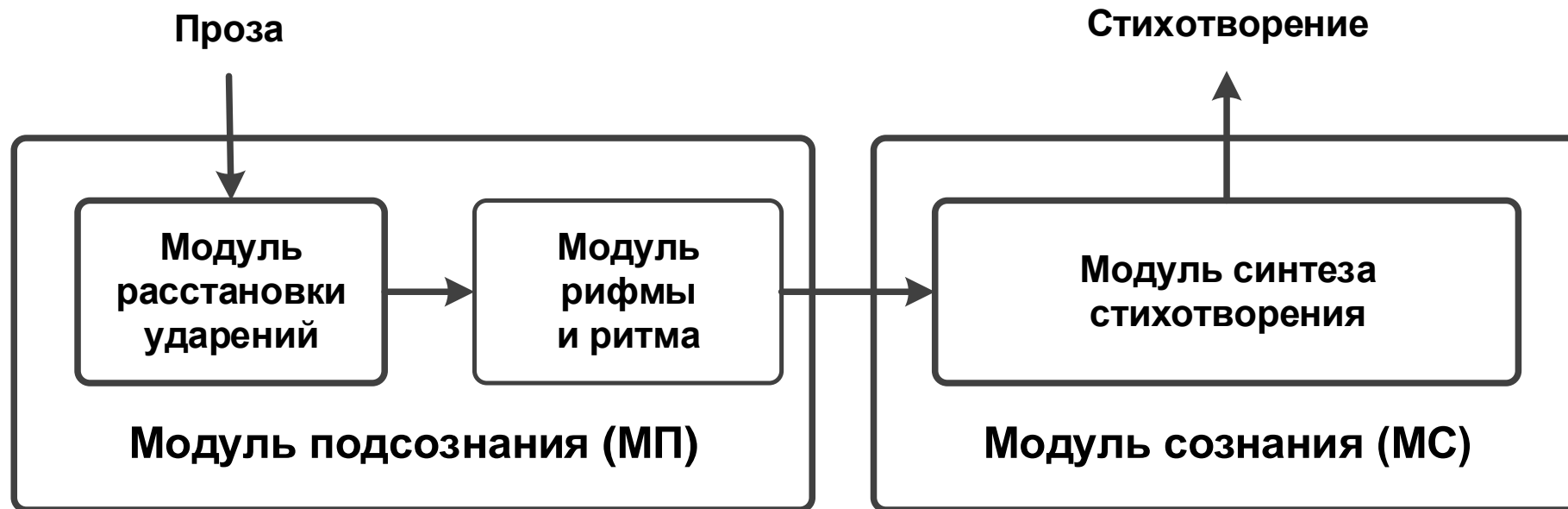
- В случае использования стратегии dropout структура нейросети должна быть изменена. При использовании данной стратегии сначала применяется метаграфовый агент  $ag_{NN\_DROPOUT}$  (в отличие от агентов-функций показан пунктирной стрелкой) который на основе правил модифицирует NN и формирует новую структуру NN<sup>DROPOUT</sup> (индекс  $i$  показывает, что может быть сформировано несколько структур для различных вариантов dropout). Далее для NN<sup>DROPOUT</sup> создается соответствующая структура TM<sup>DROPOUT</sup> используемая для обучения сети.
- Альтернативой данному подходу является встраивание правил в агент  $ag_{TM}$ , которые будут использовать dropout уже в TM-структуре без необходимости модификации исходной структуры NN.
- Таким образом:
  - Нейронная сеть может быть представлена в виде метаграфа данных.
  - С использованием агентов-функций можно трансформировать структуру нейросети.
  - С использованием метаграфовых агентов можно динамически изменять структуру нейросети, моделируя различные алгоритмы обучения.

# Гибридная интеллектуальная информационная система для генерации стихотворений

*Maria Taran, Georgiy Revunkov, Yuriy Gapanyuk. The Hybrid Intelligent Information System for Poems Generation. NEUROINFORMATICS 2019: Advances in Neural Computation, Machine Learning, and Cognitive Research III. pp 78-86.*

**Подсознание – нейронная сеть**

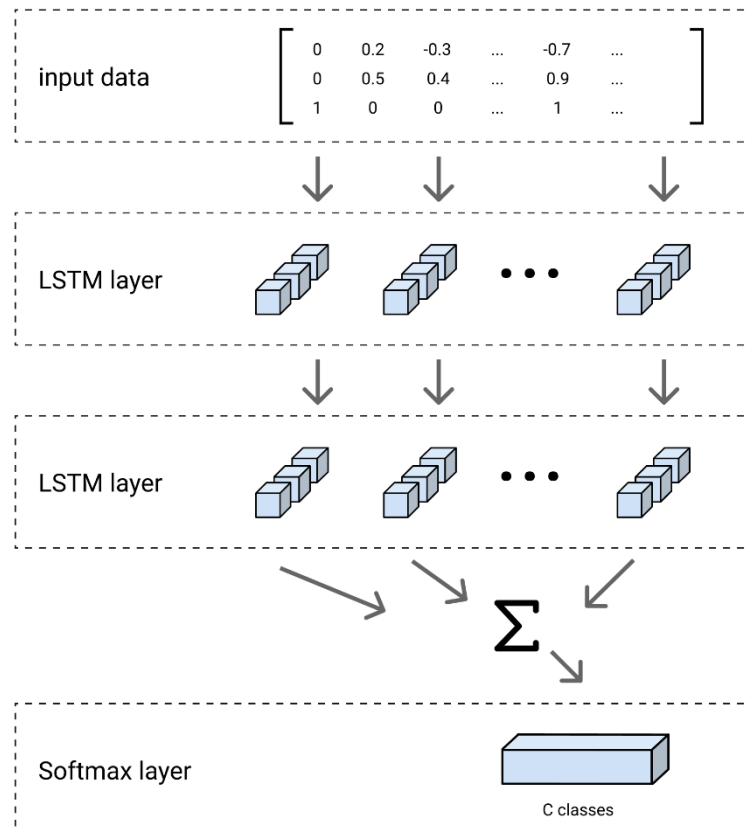
**Сознание – система на правилах**



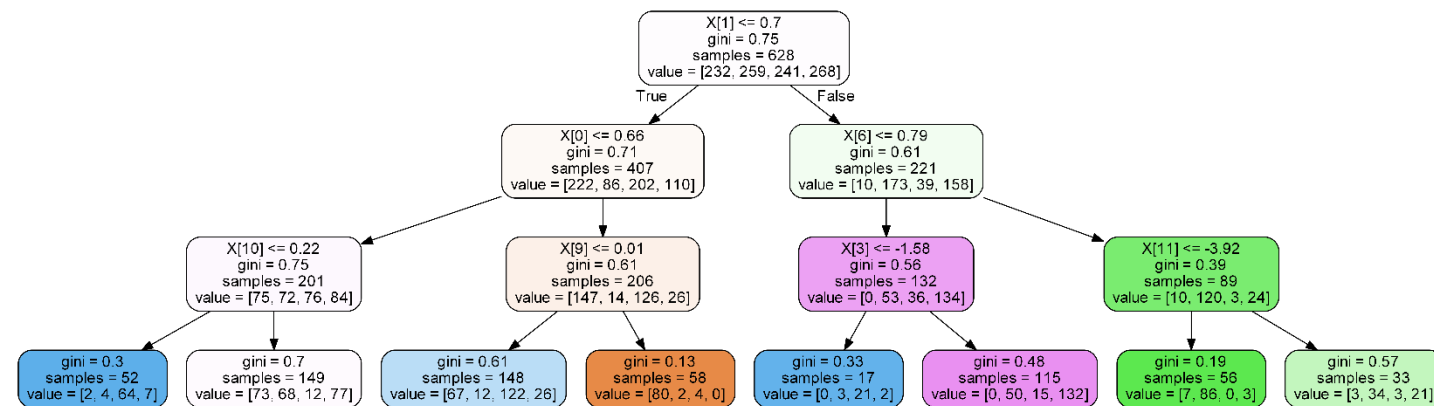
# Гибридная интеллектуальная информационная система для классификации музыки

*Aleksandr Stikharnyi, Alexey Orekhov, Ark Andreev, Yuriy Gapanyuk. The Hybrid Intelligent Information System for Music Classification. NEUROINFORMATICS 2019: Advances in Neural Computation, Machine Learning, and Cognitive Research III. pp 71-77.*

Подсознание – нейронная сеть LSTM



Сознание – решающее дерево



# Реализация ГИИС

- Классическое понимание

- В качестве методов обработки данных «подсознания» хорошо подходят методы, основанные на машинном обучении, нейронных сетях, нечеткой логике, в том числе и комбинированные нейронечеткие методы. Эти методы основаны на векторном (матричном, тензорном) представлении признаков и их обработке.
- В качестве методов обработки данных «сознания» используются онтологии и правила, экспертные системы.

- Современное понимание

- Методы, основанные на векторном представлении все активнее используются для обработки данных «сознания». Пример – Graph Neural Network (GNN).
- Использование векторного представления можно рассматривать как **«схему глубинной интеграции»** сознания и подсознания.
- Концепции «сознания» и «подсознания» ГИИС не исчезают, но могут быть реализованы унифицированным образом на основе векторного представления признаков.