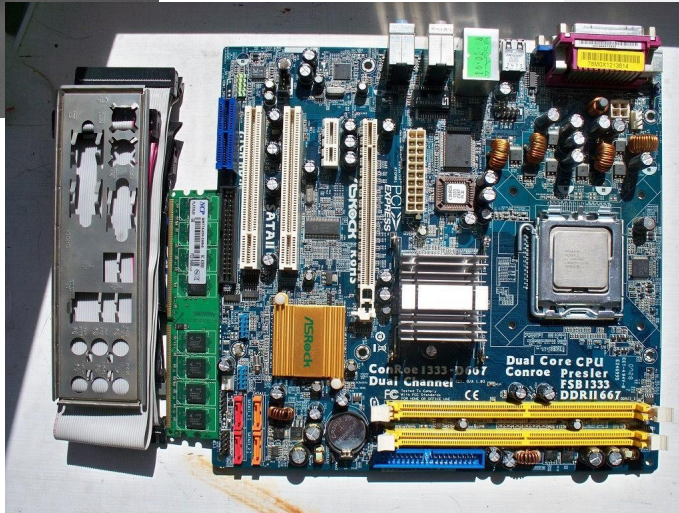
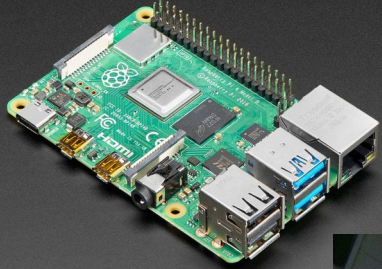


Архитектура вычислительных систем. Основные понятия



Определение архитектуры ВС (далее А.)

- А.** - концептуальная структура вычислительной машины, определяющая проведение обработки информации и включающая методы преобразования информации в данные и принципы взаимодействия технических средств и программного обеспечения.
- А.** - абстрактное представление ЭВМ, отражающее её *структурную, схемотехническую и логическую* организацию.
- А.** - множество взаимосвязанных компонент ЭВМ, включающих: *программное обеспечение (software), аппаратное обеспечение (hardware), алгоритмическое обеспечение (brainware), специальное микропрограммное обеспечение (firmware)* + система (структура), поддерживающая слаженное функционирование перечисленного.
- А.** - абстрактное многоуровневое представление физической системы с точки зрения программиста с закреплением функций за каждым уровнем и установлением интерфейса между уровнями.
- А.** вычислительного средства (в узком смысле) – совокупность его свойств и характеристик, служащих для удовлетворения потребностей пользователя.

Структура (от лат. structūra - “строение”) - внутреннее устройство чего-либо. Внутреннее устройство связано с категориями целого и его частей.

Уровень n

Виртуальная машина Мn
с машинным языком Яn

⋮

Уровень 3

Виртуальная машина М3
с машинным языком Я3

Уровень 2

Виртуальная машина М2
с машинным языком Я2

Уровень 1

Виртуальная машина М1
с машинным языком Я1

Уровень 0

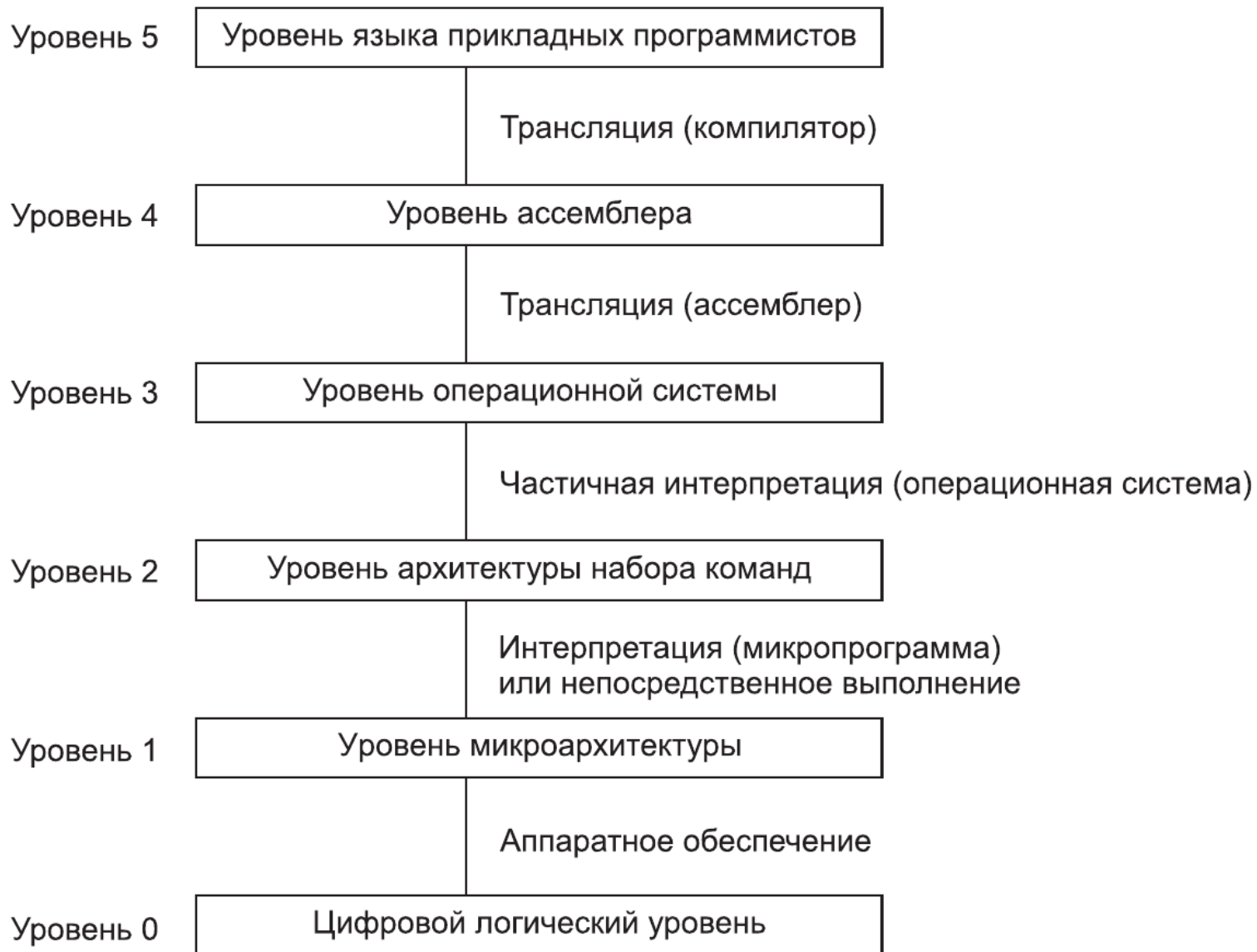
Виртуальная машина М0
с машинным языком Я0

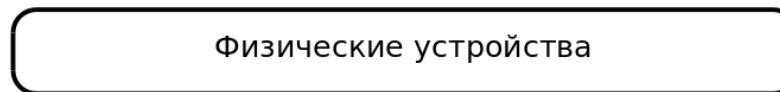
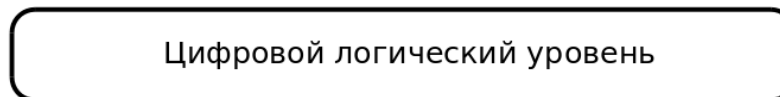
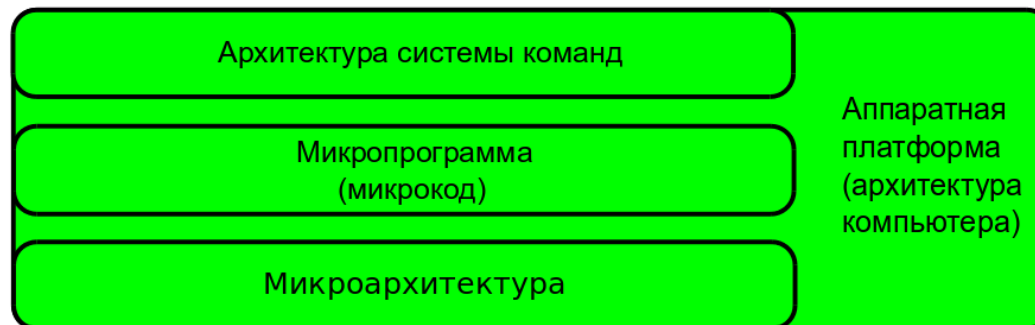
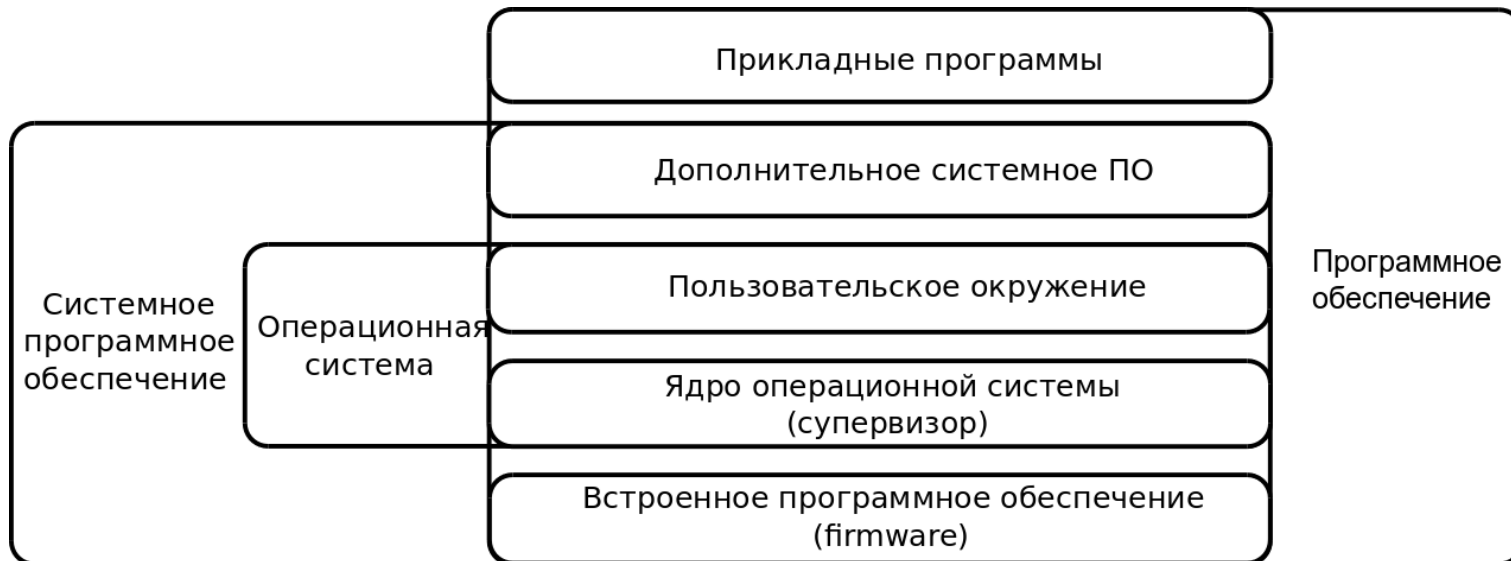
Программы на языке Яn либо интерпретируются программой-интерпретатором, работающей на машине более низкого уровня, либо транслируются на машинный язык машины более низкого уровня

Программы на языке Я2 либо интерпретируются программами-интерпретаторами, работающими на машине М1 или М0, либо транслируются на Я1 или Я0

Программы на языке Я1 либо интерпретируются программой-интерпретатором, работающей на машине М0, либо транслируются на Я0

Программы Я0 непосредственно выполняются электронными схемами





Детализация уровней

- Цифровой логический уровень
- Уровень микроархитектуры
- Уровень архитектуры набора команд
- Уровень операционной системы
- Уровень ассемблера
- Уровень промежуточных языков (команд) – (LLVM, IL, JavaVM...)
- Уровень языков ориентированных на системное программирования (C, C++, Rust...)
- Уровень универсальных языков прикладного программирования (Java, Kotlin, C#, Python, Go, JS...)
- Уровень предметно-ориентированных (проблемно-ориентированных) языков программирования (Norma, ANTLR, GPSS, Prolog...)

Компьютеры предназначены для программирования, а архитектурные уровни выделяются для повышения эффективности процесса разработки ПО для преодоления семантического разрыва между предметной областью и реальным физическим исполнителем.

От предметной области к архитектуре ВС

Разработка больших программ – многоступенчатый процесс, в ходе которого осуществляются как ручные трансформации неформальных моделей решаемой задачи в формализованные представления, так и их последующая автоматическая трансформация с использованием различных систем программирования.

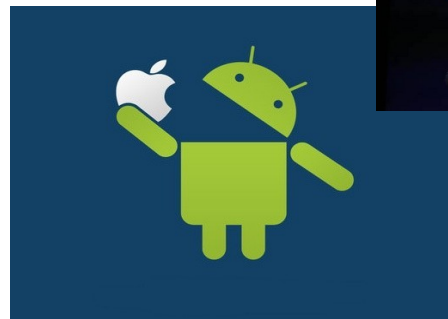
Предметная область
Архитектура ВС
Инструменты для разработки ПО



Влияние особенностей архитектур вычислительных систем («железо», ОС, ЯП)

- время выполнения программы
- особенности программирования
- особенности оборудования

...



Помимо увязки моделей предметной области с архитектурой ВС, встают проблемы, определяемые спецификой инструментов, используемых для создания программ.

Можно выделить **технологическое направление**, напрямую не связанное с предметной областью.

В его рамках формулируются требования:

- к средствам, обеспечивающим написание программ,
- к средствам проектирования, определяющим переход от моделей предметной области к программам.

Огромную роль на разработку ПО оказывает необходимость соответствия заданным **критериям качества.**

Ряд критериев вытекает из особенностей предметной области. Другие обуславливаются архитектурой ВС и другими причинами.

Вместе они характеризуют комплекс проблем, преодолеть который пытаются разработчики программного обеспечения.

Критерии качества программного обеспечения

Корректность (правильность). Обеспечивает правильную обработку на правильных данных.

Устойчивость. "Эlegantное" завершение обработки ошибок.

Расширяемость. Может легко адаптироваться к изменяющимся требованиям.

Многократность использования. Может использоваться и в других системах, а не только в той, для которой было создано.

Совместимость. Может легко использоваться с другим программным обеспечением.

Эффективность. Эффективное использование времени, компьютерной памяти, дискового пространства и т. д.

Переносимость (мобильность). Можно легко перенести на другие аппаратные и программные средства.

Критерии качества программного обеспечения

Верифицируемость. Простота проверки, легкость разработки тестов при обнаружении ошибок, легкость обнаружения мест, где программа потерпела неудачу, и т. д.

Поддержка целостности. Защищает себя от неправильного обращения и неправильного употребления.

Легкость использования. Для пользователя и для будущих программистов

Невозможно сопоставить важность указанных характеристик, так как все они должны учитываться при разработке программного обеспечения.

Расстановка приоритетов осуществляется в зависимости от целей процесса разработки ПО

Цели и задачи процесса разработки

Разработка ПО - многоэтапный процесс, зависящий от многих факторов, например:

- **видов решаемых, задач**, определяющих содержание создаваемых программ;
- **методологий**, задающих особенности организационного и технического проведения основных этапов разработки ПО;
- **методов и парадигм программирования**, обуславливающих стили кодирования и **архитектуры виртуальных машин (ВС)**;
- **системных программных средств**, предоставляющие логические ресурсы для использования ПО.
- **аппаратных средств**, предоставляющие физические ресурсы для использования ПО.

Цель процесса разработки – создание программы, обеспечивающей решение поставленной задачи некоторым исполнителем (**вычислительной системой с целевой архитектурой**) в соответствии с заданными критериями качества.

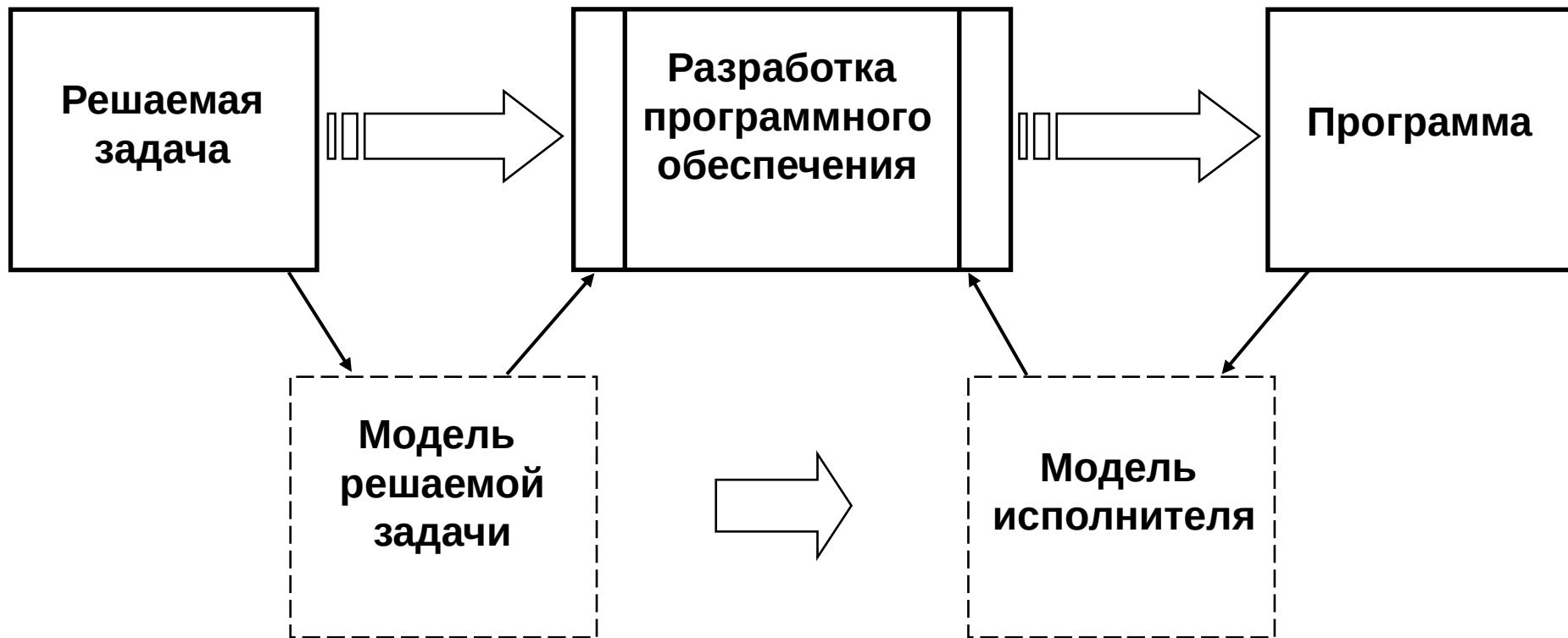
Решаемая задача описывается совокупностью формальных и эмпирических (неформальных) **моделей**, определяющих как протекающие **процессы**, так и используемые при этом **данные**.

Модель задачи – совокупность специализированных моделей, описывающих различные аспекты решаемой задачи, отражаемые в разрабатываемой программе.

Специализированная модель – модель, предназначенная для описания определенных параметров рассматриваемого явления. Используется для акцентирования внимания на частных характеристиках.

Разрабатываемая программа должна обеспечивать выполнение функций, необходимых для решения задачи, в соответствующем **исполнителе (вычислительной системе)**, специфика которого отражается в его **модели**.

Модель исполнителя – совокупность специализированных моделей, описывающих организацию и поведение вычислительной системы, осуществляющей выполнение программы (**архитектура ВС**).



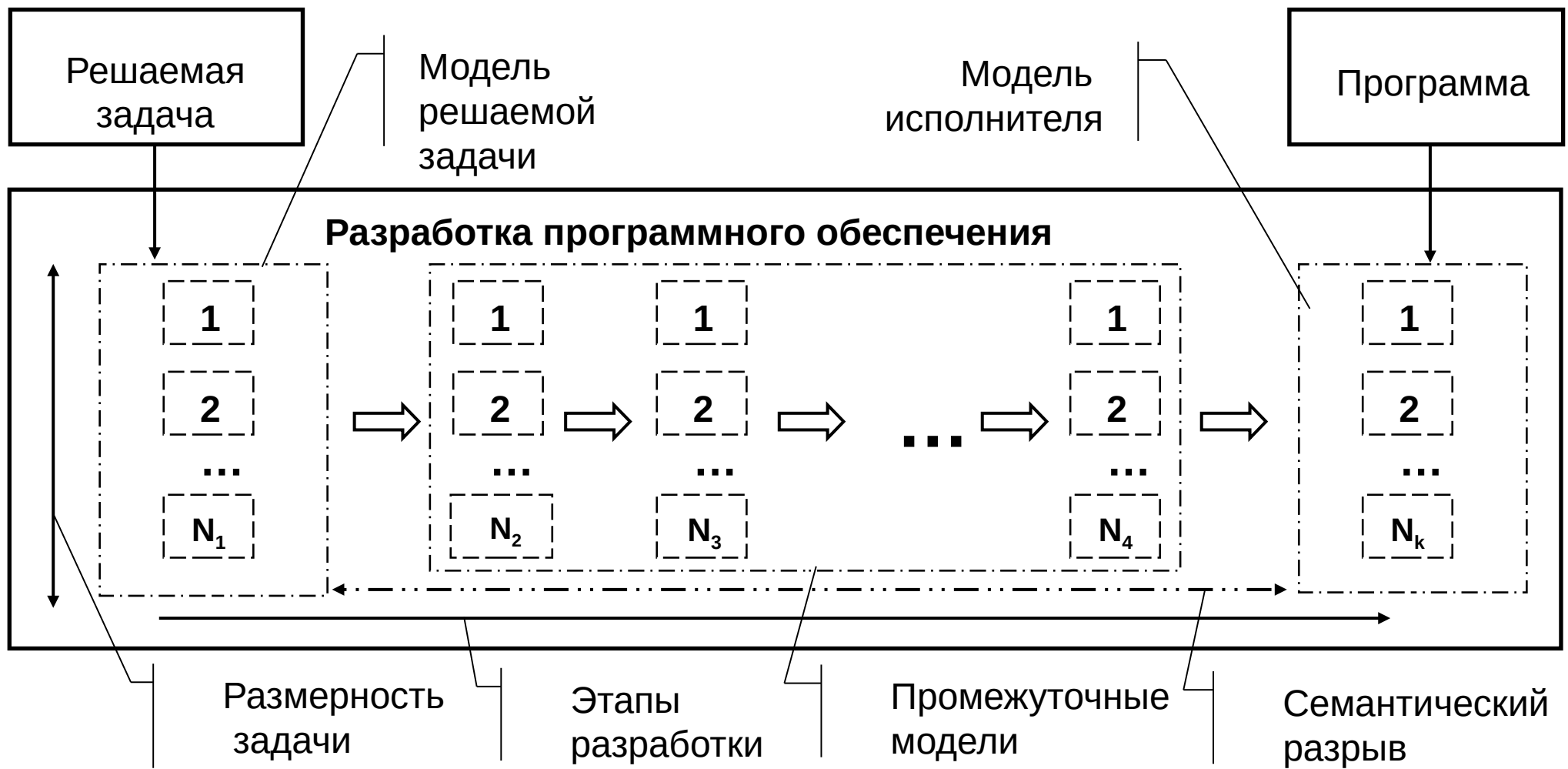
Создаваемая программа является отображением модели решаемой задачи на модель исполнителя

Трудоемкость программирования, с одной стороны, определяется **количеством специализированных моделей**, описывающих задачу, их размером, семантическим **отличием от специализированных моделей исполнителя**.

С другой стороны она зависит от **характеристик исполнителя**, задающего требования к уровню абстракции разрабатываемой программы и ее приближенностью к **архитектуре реальной вычислительной системы**.

Существует **семантического разрыв** между моделями задач и моделями исполнителей: **объекты и операции, которыми разработчик манипулирует при описании задачи, не совпадают с объектами и операциями, используемыми при построении программы**.

Преодоление семантического разрыва осуществляется использованием **методических и технических приемов**, повышающих эффективность процесса разработки ПО.



Влияние сложности задачи на процесс преобразования исходных моделей на модели исполнителя

Методические приемы

Методические приемы ориентированы на формализацию представления моделей и методов перехода между ними.

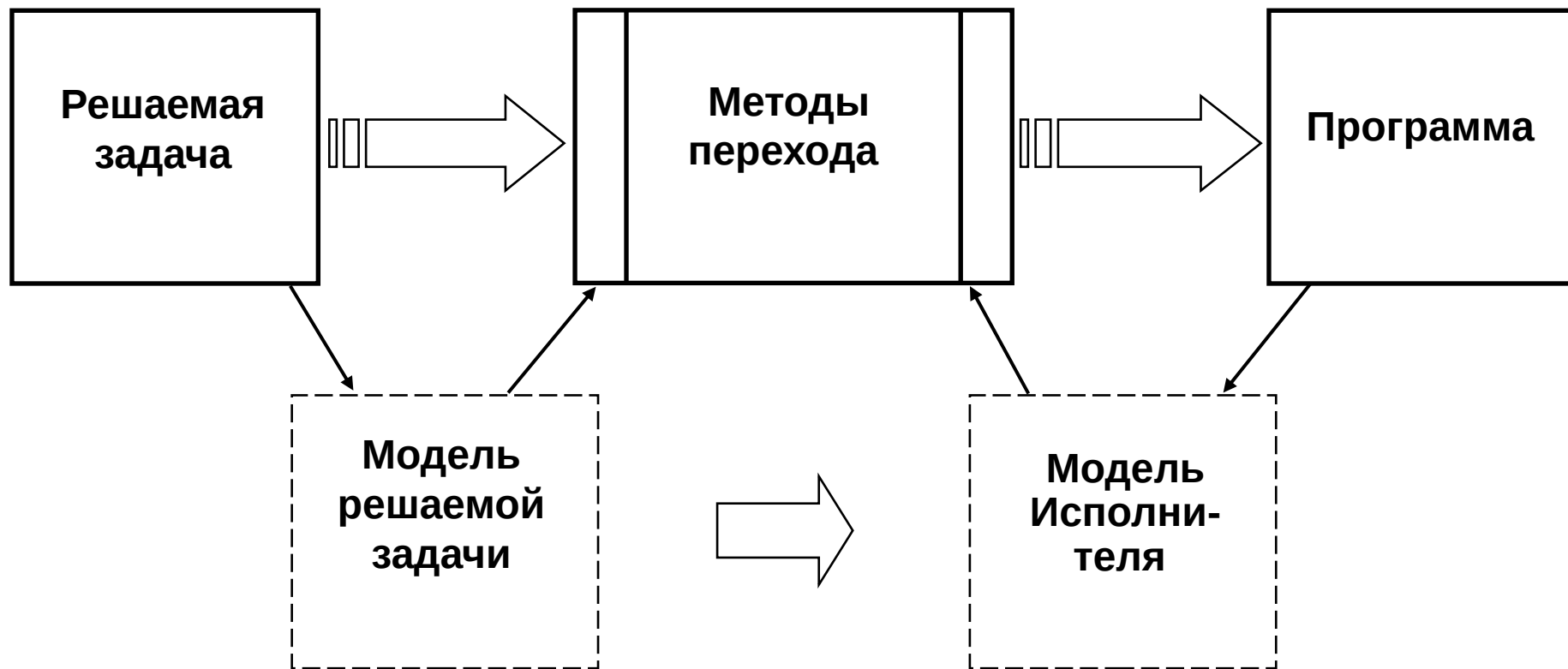
Позволяют ускорить процесс разработки следующими способами:

- **формализацией на уровне предметных областей;**
- **созданием методик разработки программного обеспечения.**

Формализация предметной области

Формализация предметной области заключается в построении ее модели и разработке методов преобразования в **модель исполнителя (в архитектуру ВС)**.

Модель предметной области объединяет совокупность специализированных моделей предназначенных для описания определенного класса решаемых задач, что обеспечивают унификацию решения сверху. Дальнейший переход к модели исполнителя обычно осуществляется по выработанным методам или алгоритмам



Общий механизм перехода с использованием разработанных методов

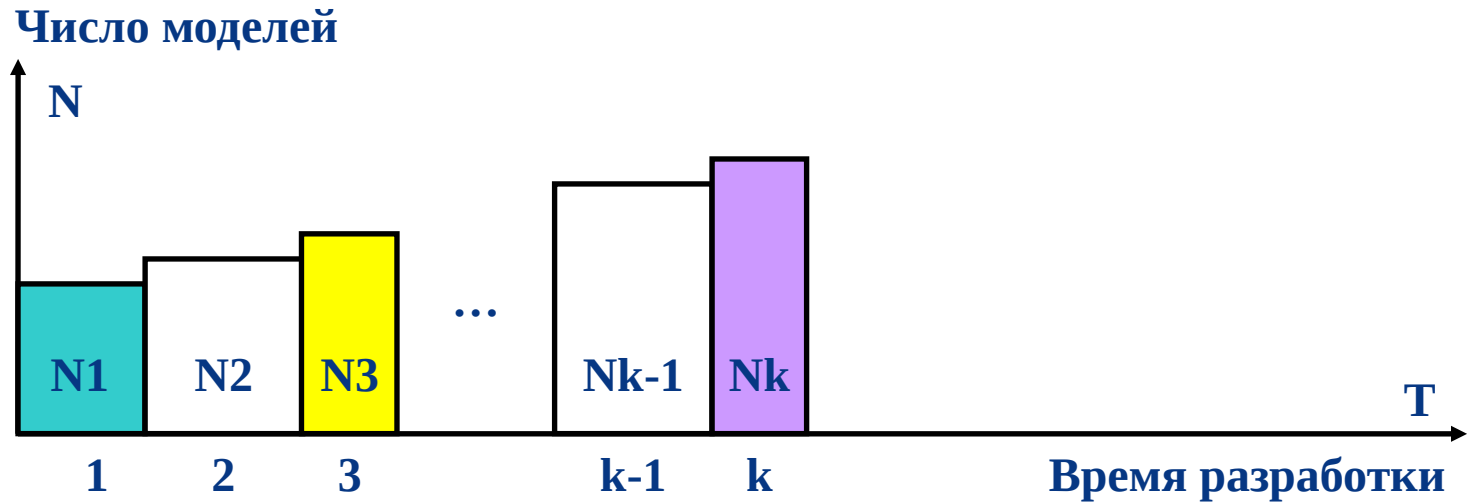
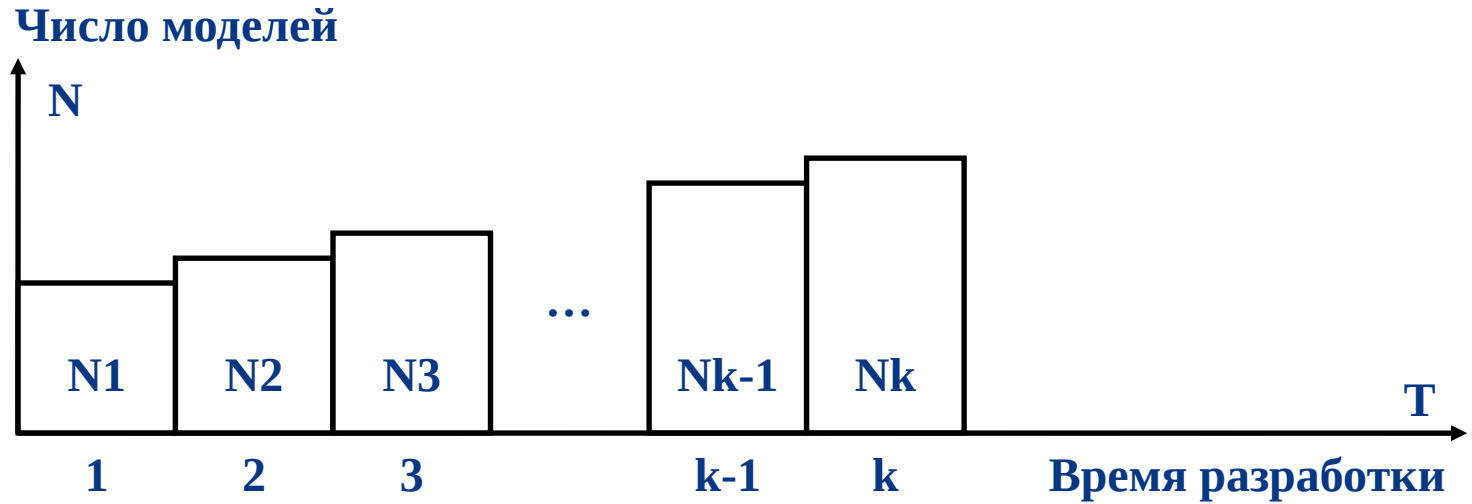
Примеры использования:

- 1. Разработка компиляторов.** Модель предметной области описывается с использованием формальных грамматик, эквивалентность между различными грамматиками и автоматами позволяет перейти к распознавателям путем использования наработанных методов их программной реализации.
- 2. Разработка программных систем на основе теории автоматов.** Последующая программная реализация автоматов является хорошо отработанным формальным приемом с применением различных технологий.
- 3. Нейронные сети.** Отработанные теоретические подходы, формализованные в различные наборы инструментов (Tensorflow, Keras, Theano...)

...

Что это дает →

(Отсутствует непосредственная связь с архитектурой ВС. Необходимы дополнительные шаги к архитектуре ВС. Эти шаги должны учитывать целевую архитектуру (модель исполнителя))



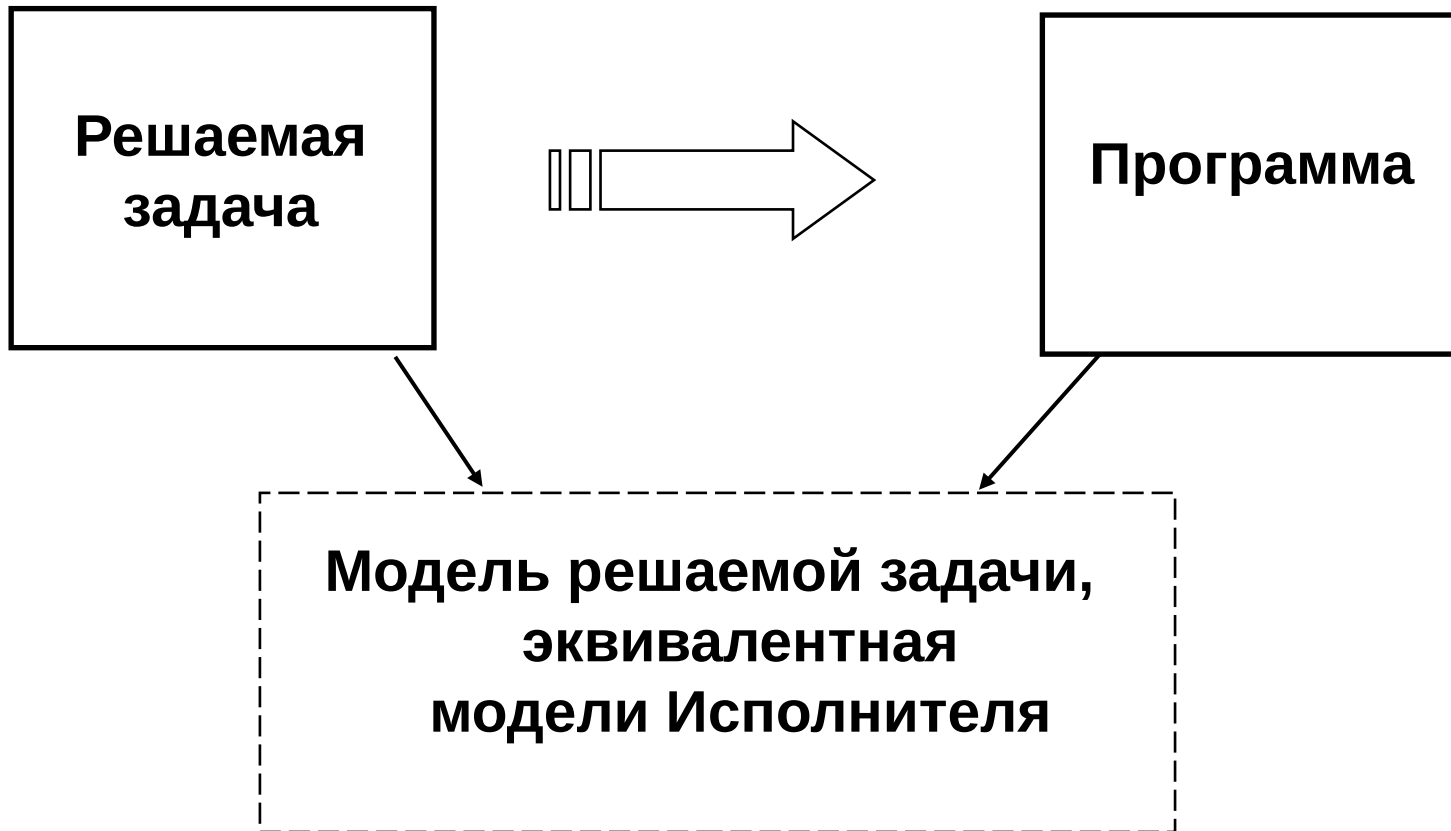
Предметно-ориентированные архитектуры

Разработка алгоритмов преобразования одних моделей в другие позволяет автоматизировать процесс и обеспечить представление исходной задачи в виде формализованных данных или программы на **специализированном (проблемно-ориентированном) языке программирования**.

Фактически это означает слияние моделей задачи и исполнителя.

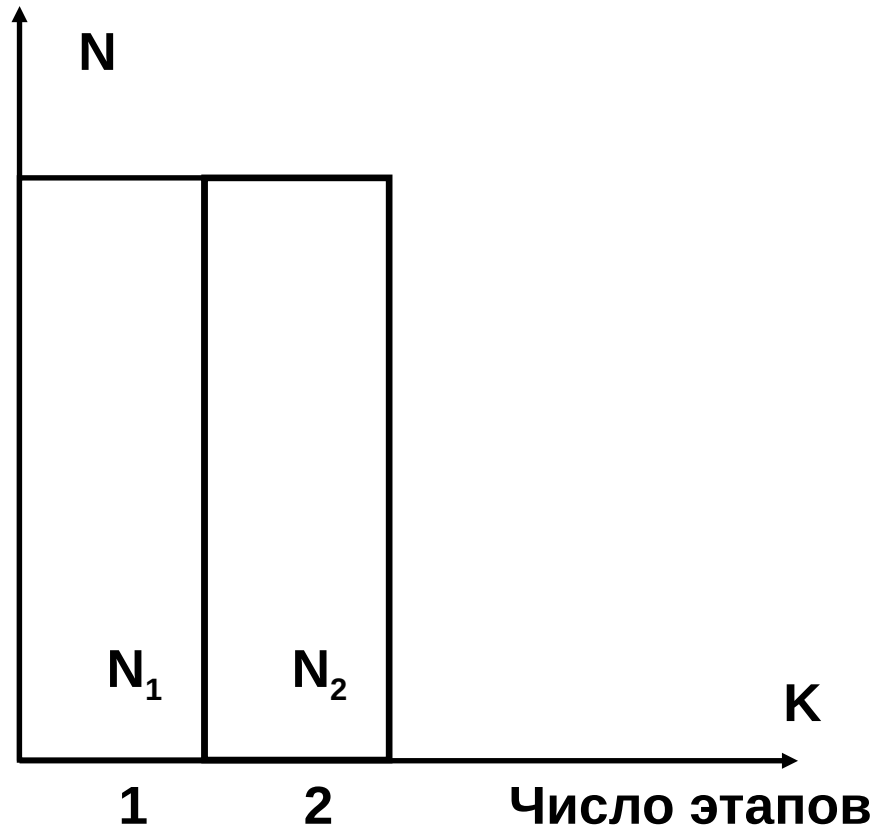
Примеры:

- Языки для работы с базами данных (SQL...)
- Языки имитационного моделирования (GPSS...)
- Языки, для управления промышленным оборудованием (LD, FBD...)
- Генераторы компиляторов по описанию синтаксиса языка программирования (ANTLR...)
- Языки описания аппаратуры (VHDL, Verilog...)



Слияние моделей при замене методов перехода системой программирования (переход к предметно-ориентированной архитектуре)

Число моделей



Использование специализированных языков
(идеальный вариант перехода к предметно-ориентированной архитектуре)

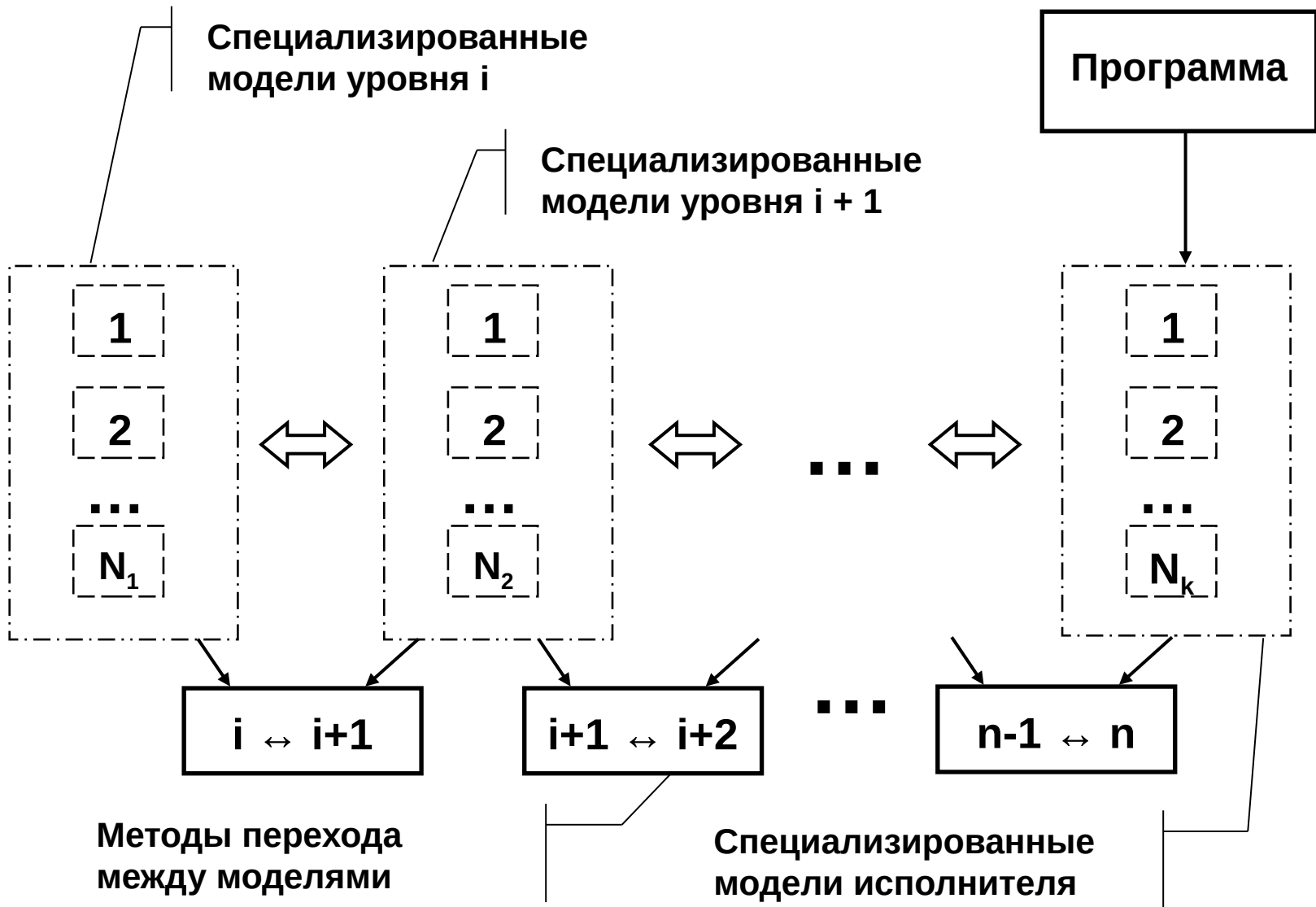
Методики разработки ПО

Напрямую не связаны с целевой архитектурой ВС

Ориентированы на **формализацию взаимосвязей** между моделями конкретных исполнителей и моделями, используемыми на предшествующих этапах разработки (например, при анализе и проектировании).

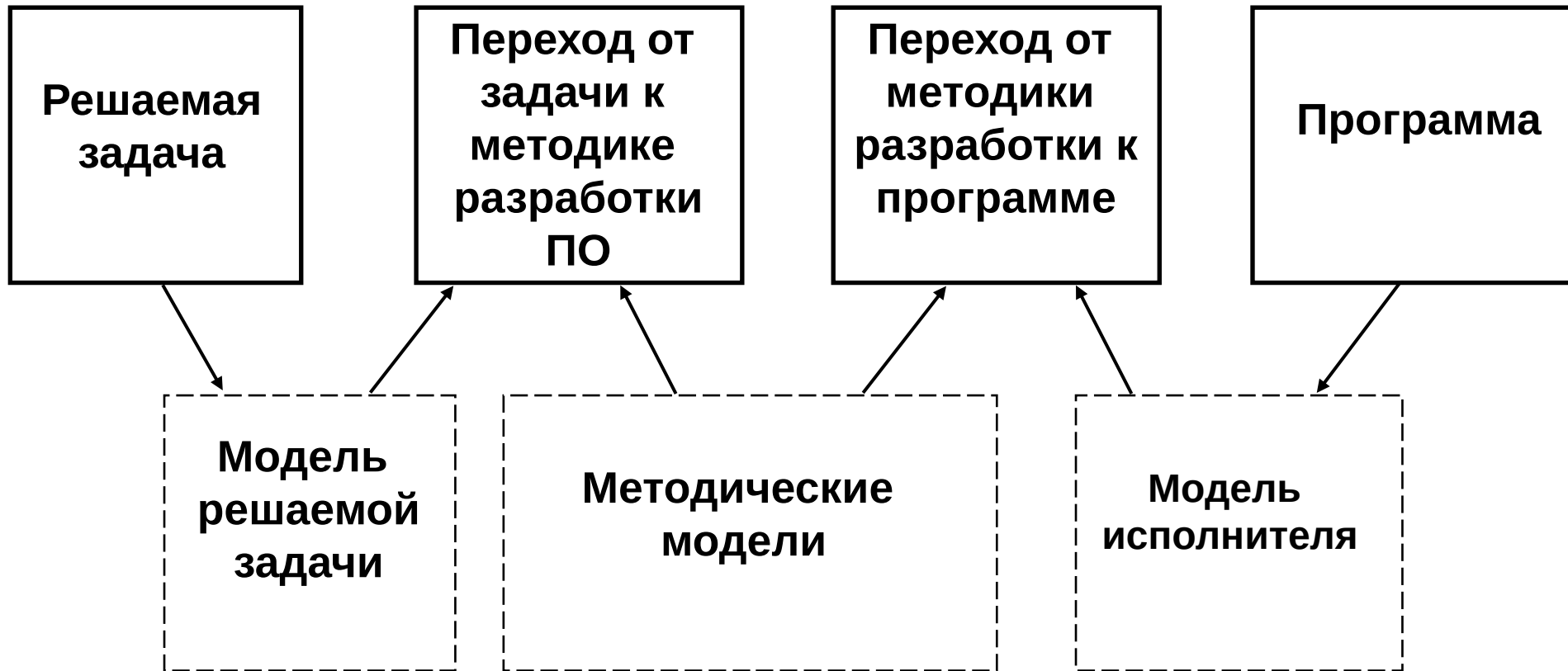
Изначальная **ориентация Исполнителя на универсальность вычислений** обычно предполагает применение методик для широкого класса задач, **не связывая их непосредственно с предметными областями.**

Поддерживают процесс преобразования **как от модели задачи к модели исполнителя, так и в обратном направлении.**



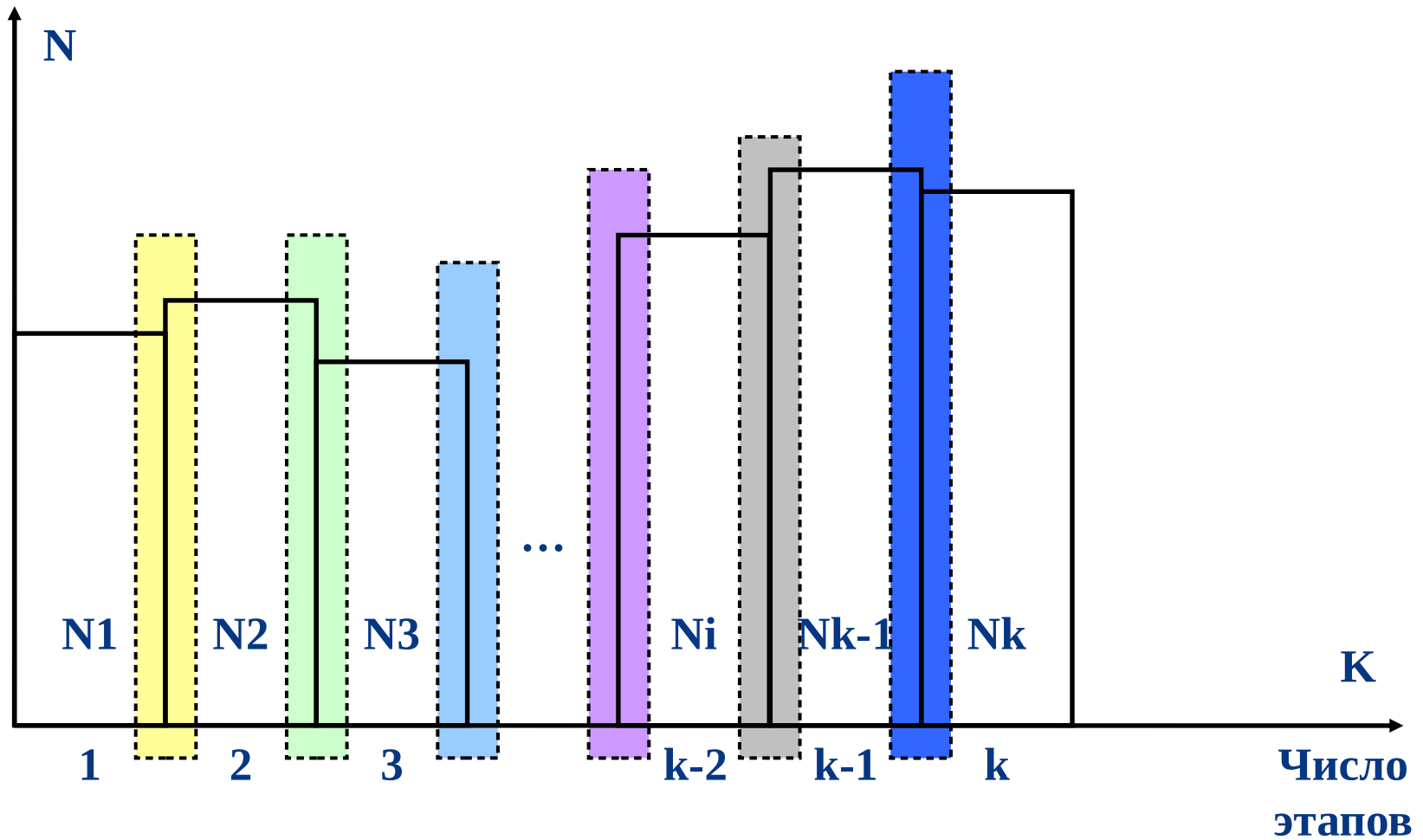
Примеры методик:

- **объектно-ориентированный подход**, используемый в составе объектно-ориентированной методологии;
- **методы структурного анализа и проектирования**, применяемые при разработке информационных систем;
- **методы быстрой разработки приложений**, ориентированные на построение программ от моделей, определяющих взаимодействие системы с пользователем;
- ...



Совместное использование методов формализации предметной области и методик разработки

Число моделей



Применение методик (методических моделей)

Резюме:

Методики широко используются при разработке больших программных систем, так как повышают эффективность процесса проектирования за счет предоставления достаточно простых и ясных подходов, выработанных на основе эмпирического опыта и теоретических исследований.

Их использование, в сочетании с инструментальной поддержкой, обеспечивает сокращение семантического разрыва между моделями различных задач и исполнителем (архитектурой ВС).

Технические приемы

Технические приемы обеспечивают создание инструментальных средств, поддерживающих различные аспекты разработки ПО для архитектур ВС заданного уровня.

Можно выделить:

- средства поддержки методических приемов;
- вспомогательные средства;
- системы программирования.

Поддержка методических приемов

Обеспечивает компьютерное представление и анализ разрабатываемых моделей, преобразование построенных моделей в другие модели, автоматизацию процесса построения требуемой документации, а также ведения организационной деятельности в ходе разработки ПО.

Примеры:

- CASE-средства (Computer Aided Software Engineering)
- Системы поддержки документации
- ...

Непосредственно с архитектурами ВС не связаны

Вспомогательные средства

Предназначены для повышения эффективности процессов, не связанных с непосредственной разработкой структуры программы, но, в то же время, сильно влияющих на качество и время разработки.

Примеры:

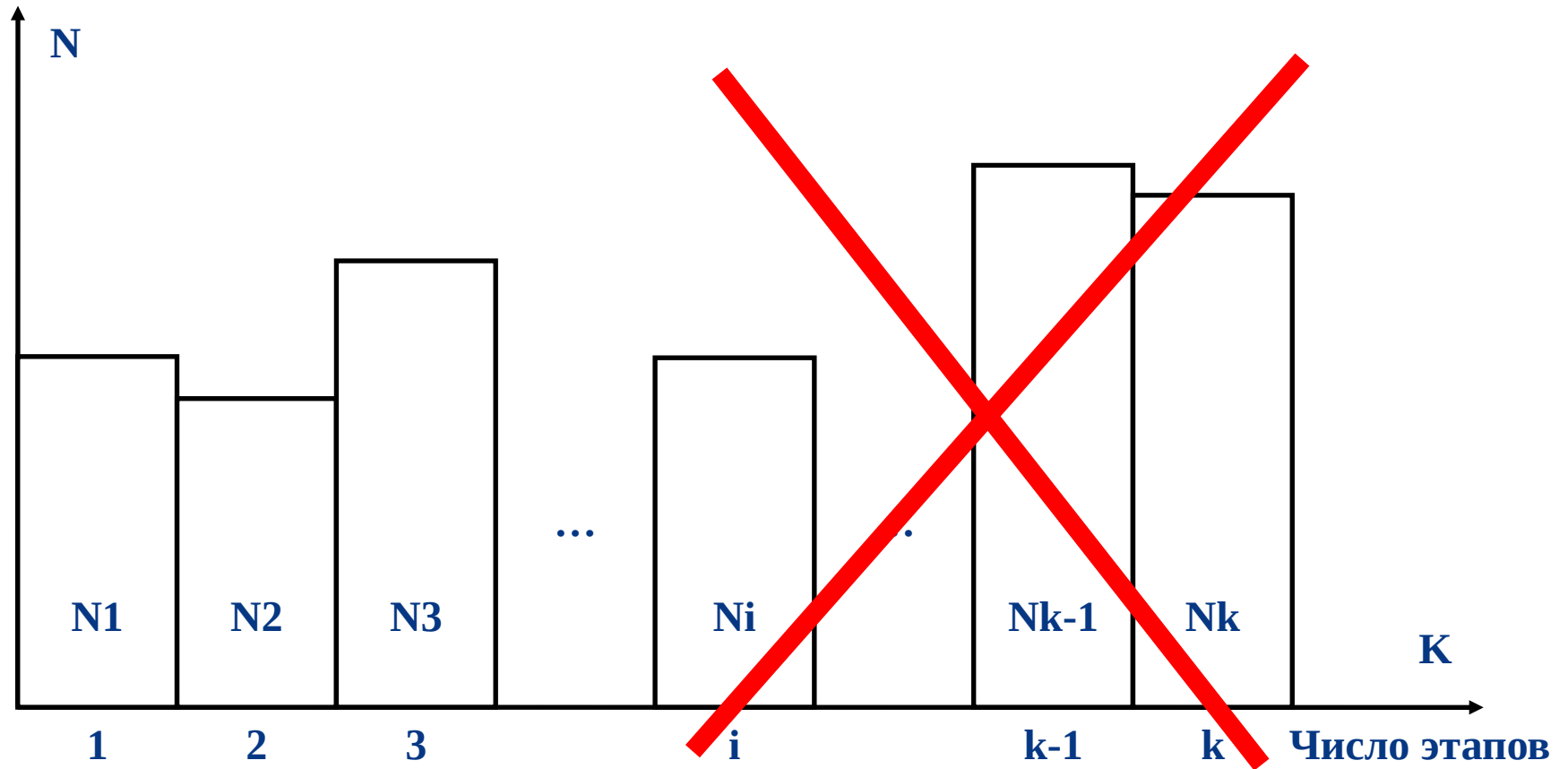
- средства отладки
- средства тестирования
- средства профилирования
- ...

Позволяют провести анализ особенностей выполнения программы на целевой архитектуре ВС

Системы программирования

Системы программирования – инструментальные средства, поддерживающие разработку программ для **заданного виртуального исполнителя (для целевой архитектуры ВС)** и их последующее автоматическое преобразование в программы **реального исполнителя (реальной архитектуры ВС)**.

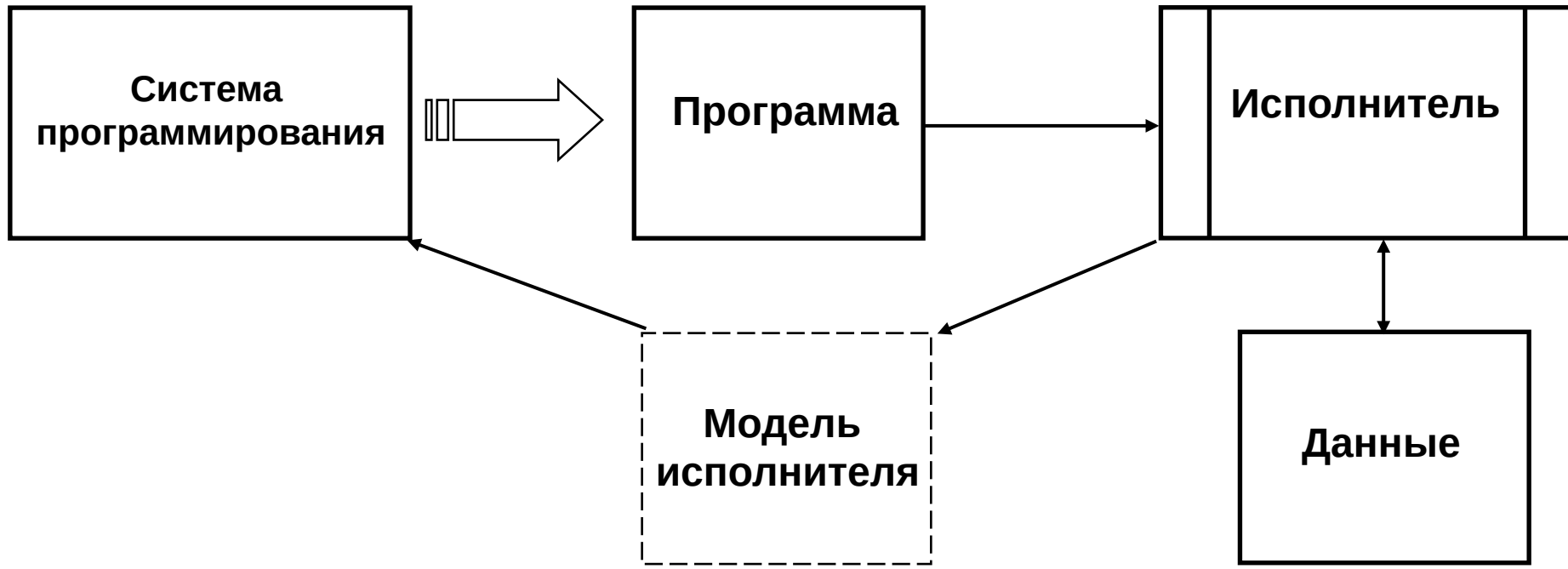
Число моделей



Системы программирования «убирают» часть моделей из процесса разработки, виртуализируя тем самым архитектуру ВС

Варианты выполнения:

1. Непосредственная интерпретация, полностью скрывающая процесс реального выполнения, который может оказаться намного сложнее.



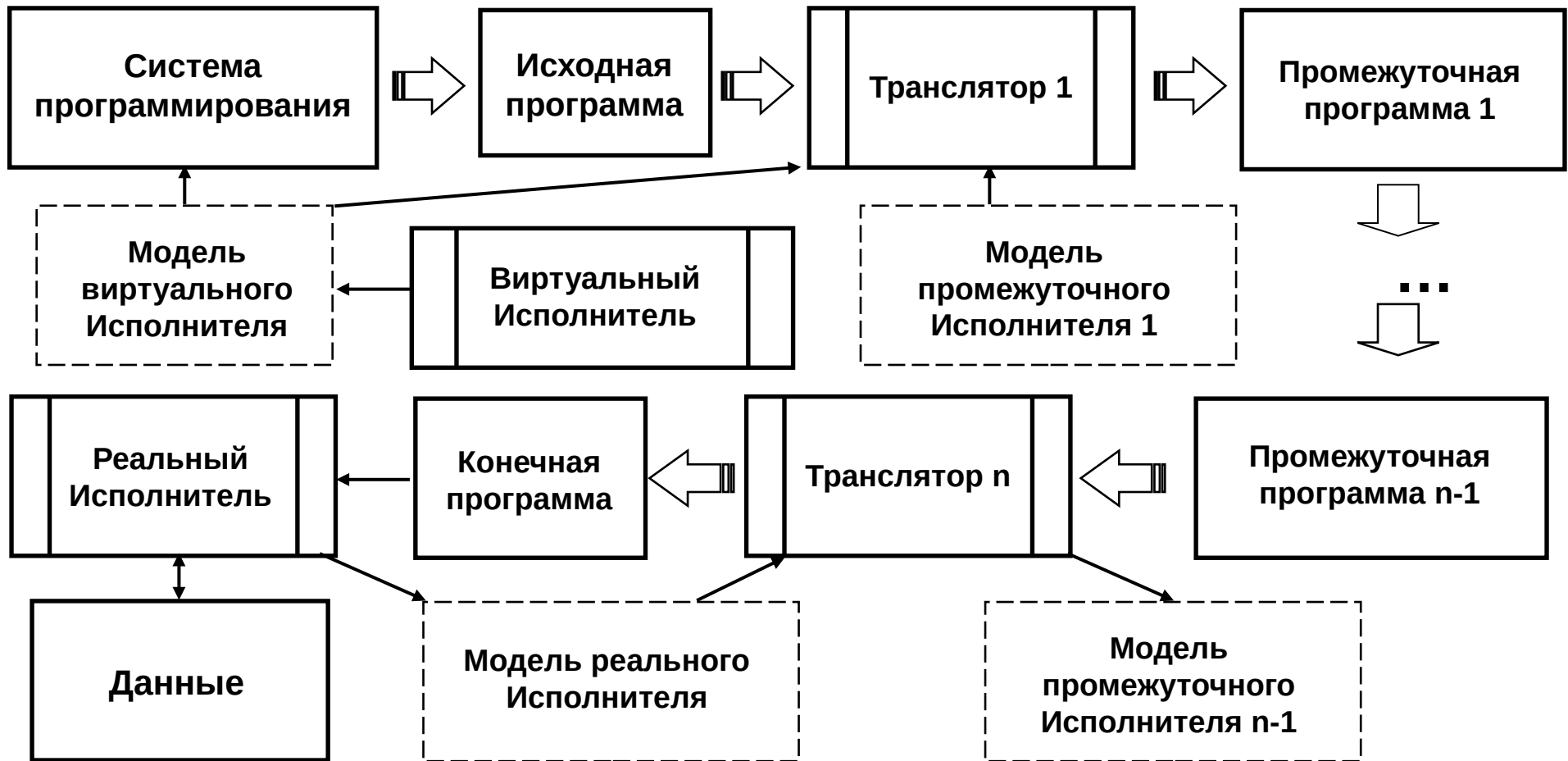
Непосредственное использование исполнителем программы, написанной с применением системы программирования

Варианты выполнения:

2. Автоматическим и поэтапным преобразованием программы, написанной для виртуальной машины, в программу конечного (реального или виртуального) исполнителя, что позволяет разрабатывать программы для систем, не допускающих непосредственное выполнение **(системы компиляции)**.

Преобразование может быть многоступенчатым, что может оказаться целесообразным при использовании нескольких промежуточных моделей исполнителей.

Проводимые промежуточные преобразования могут быть **скрыты от программиста**. Однако **знание дополнительной информации о конечном исполнителе позволяет повысить эффективность процессов отладки и выполнения программы**.



Многоэтапное преобразование модели исполнителя, поддерживаемого системой программирования, на модель исполнителя, реально осуществляющего вычисления

Список источников для изучения дисциплины (1)

◆◆ Основная литература

1. Э.Таненбаум. Архитектура компьютера. 6-е изд. —СПб.: Изд. Питер, 2017. – 816 с.: с ил. + CD-ROM. (2013-2018)
2. Randal E.Bryant, David R.O’Hallaron. Computer systems: a programmer’s perspective. 2-nd ed. — Prentice Hall, 2010. – 1043 p. (есть русск. перевод: Рэндал Э.Брайант, Дэвид Р.О’Халларон. Компьютерные системы. — СПб.: БХВ-Петербург, 2005. – 1104 с.)
3. Бройдо В.Л., Ильина О.П.. Архитектура ЭВМ и систем. —СПб.: Изд. Питер, 2009. –720 с.: с ил.
4. Максимов Н.В., Партыка Т.Л., Попов И.И. Архитектура ЭВМ и вычислительных систем. —М.: Изд. Форум, 2010. – 512 с., ил.

◆◆ Дополнительная литература

◆ По данной теме

1. Легалов А.И. О разработке программного обеспечения. Модельный взгляд. 2018. – <http://softcraft.ru/notes/devproc/>

◆ Общие вопросы, принципы работы процессоров, внешние устройства ЭВМ

1. Жмакин А.П. Архитектура ЭВМ. —СПб.: Изд. БХВ-Петербург, 2010. – 352 с.: ил. + CD-ROM.
2. Колдаев В.Д., Lupin С.А. Архитектура ЭВМ. —М.: Изд. ФОРУМ - ИНФРА-М, 2009. - 384 с.: ил.
3. Федорков Е.Д, Паринов А.В. Архитектура ЭВМ и систем. –Воронеж: Изд. ВГТУ, 2005. – 308 с.
4. Буза М.К. Архитектура компьютеров. —Минск: Изд. ООО “НОВОЕ ЗНАНИЕ”, 2006. – 559 с.: ил.
5. Корнеев В.В. Вычислительные системы. —М.: Гелиос АРВ, 2004. – 512 с., ил.
6. Хорошевский В.Г. Архитектура вычислительных систем. —М.: Изд. МГТУ имени Н.Э.Баумана, 2005. – 512 с.: с ил.

Список источников для изучения дисциплины (2)

♦ Многопроцессорные и многомашинные вычислительные системы, параллельные вычисления

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. —СПб.: БХВ-Петербург, 2004. – 608 с.
2. Алабужев А.А. Архитектура параллельных ЭВМ. —Пермь: 2007. –89 с.: ил.
3. Гергель В.П. Теория и практика параллельных вычислений. —М.: Интернет-Университет Информационных технологи; БИНОМ. Университет знаний, 2007. –423 с.: ил., табл.
4. Корнеев В.В. Вычислительные системы. –М.: Гелиос АРВ. 2004, –512 с: ил.
5. Богачев К.Ю. Основы параллельного программирования. –М.: —М.: Интернет-Университет Информационных технологий; БИНОМ. Университет знаний, 2010. –344 с.
6. Сущенко С.П. Архитектура вычислительных систем. –Томск: Изд. СКК-пресс, 2006. – 198 с.
7. Шпаковский Г.И. Организация параллельных ЭВМ и суперскалярных процессоров. // Учеб. пособие. -Минск.: Белгосуниверситет, 1996. -296 с.
8. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI. -Минск.: БГУ, 2002. - 325 с.
9. Андрианов А.Н., Бугеря А.Б., Ефимкин К.Н., Задыхайло И.Б. FORTRAN. Описание языка. Рабочий стандарт. // Препринт ИПМ им.М.В.Келдыша РАН, № 120, 1995, -50 с.

Список источников для изучения дисциплины (3)

♦ Реконфигурируемые вычислительные системы

1. Дорнопуло А.И. и др. Высокопроизводительные реконфигурируемые вычислительные системы. // Журнал “Суперкомпьютеры”, —М.: 2010, № 3, с.45-47.
2. Корнеев В.В. Архитектура вычислительных систем с программируемой структурой. —Новосибирск: Наука, 1985. —166 с.

♦ Технология CUDA (вычисления на графических картах фирмы NVIDIA)

1. Боресков А.В., Харламов А.А., Марковский Н.Д. и др. Параллельные вычисления на GPU. Архитектура и программная модель CUDA. Учебное пособие. — М.: Изд. Московского университета, 2012. — 336 с.
2. Дж.Сандерс, Э.Кенрот. Технология CUDA в примерах: введение в программирование графических процессоров (пер. с англ.). — М.: ДМК Пресс, 2011. — 232 с.
3. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. — М.: ДМК Пресс, 2011. — 232 с.

♦ Нейросети и нейрокомпьютеры

1. Злобин В.К., Ручкин В.Н. Нейросети и нейрокомпьютеры. Учебное пособие. — СПб.: БХВ-Петербург, 2011. — 256 с.
2. Николенко С.И., Тулупьев А.Л. Самообучающиеся системы. — М.: МЦНМО, 2009. — 288 с. [стр. 81–117].
3. Яхъяева Г.Э. Нечёткие множества и нейронные сети. Учебное пособие. — М.: БИНОМ, 2008. — 316 с.
4. Злобин В.К., Ручкин В.Н. Нейросети и нейрокомпьютеры. Учебное пособие. — СПб.: БХВ-Петербург, 2011. — 256 с.
5. Комарцова Д.Г., Максимов А.В. Нейрокомпьютеры. Учебное пособие для ВУЗ'ов. — М.: Изд. МГТУ им. Н.Э.Баумана, 2004. — 400 с.
6. Головкин В.А. Нейронные сети: обучение, организация и применение. Учебное пособие для ВУЗ'ов. — М.: ИПРЖР, 2001. — 256 с.

Список источников для изучения дисциплины (4)

◆ Неклассические архитектуры, потоковые (DATA-FLOW) вычислители

1. Барский А.Б. Параллельные информационные технологии. —М.: Интернет-Университет Информационных технологий; БИНОМ. Университет знаний, 2007. —503 с.
2. Нетрадиционная система остаточных классов и ее основоположник И.Я.Акушский. [Электронный ресурс] — Режим доступа: <http://lib2.znate.ru/docs/index-337416.html>, свободный.
3. Стерлинг Томас. Многоточие Стерлинга. // Журнал “Суперкомпьютеры”, —М.: 2010, № 3, с.17-20.
4. Баканов В.М. Параллелизация обработки данных на вычислителях потоковой (Dataflow) архитектуры. // Журнал “Суперкомпьютеры”, —М.: 2011, № 5, с.54-58.
5. Баканов В.М. Управление динамикой вычислений в процессорах потоковой архитектуры для различных типов алгоритмов. // Журнал “Программная инженерия”, —М.: 2015, № 9, с.20-24.

◆ Ресурсы сети InterNet

1. Информационно-аналитические материалы по параллельным вычислениям. [Электронный ресурс] — Режим доступа: <http://parallel.ru>, доступ свободный.
2. Журнал “Суперкомпьютеры”. [Электронный ресурс] — Режим доступа: <http://supercomputers.ru>, свободный.
3. Архитектура вычислительных систем. [Электронный ресурс] — Режим доступа: <http://studdi.ru/lecture/avs>, свободный.
4. NVIDIA: архитектура графических карт и технология CUDA. [Электронный ресурс] — Режим доступа: <http://www.nvidia.ru>, свободный.

Информация для самостоятельной работы

□ Ключевые слова для поиска в сети InterNet: архитектура вычислительных систем; классификации архитектуры компьютеров и систем М.Флинна, Р.Хокни, Т.Фенга, В.Хендлера, Л.Шнайдера, Д.Скилликорна; языки высокого и сверхвысокого уровня; язык ассемблера; формат машинных команд;

□ Некоторые известные (и популярные!) системы поиска информации в Сети: (больше – [здесь](#)):`

Яндекс

Google

 Rambler

YANOO!
ПО-РУССКИ

Вопросы для обсуждения на семинаре

- 1-1. Определение и место дисциплины АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ в программной инженерии.
- 1-2. Многоуровневость архитектурных решений. Отличительные черты уровней.
- 1-3. Связь уровней архитектур с методами и парадигмами программирования. Отображение парадигм программирования в архитектурах различного уровня.
- 1-4. Место архитектур ВС в процессе разработки программного обеспечения.
- 1-5. Связь формализации предметной области с конкретной архитектурой.
- 1-6. Связь между методиками разработки ПО и архитектурами ВС.
- 1-7. Вспомогательные технические средства. Их помощь в повышении эффективности программного обеспечения.
- 1-8. Роль систем программирования в виртуализации архитектур ВС.