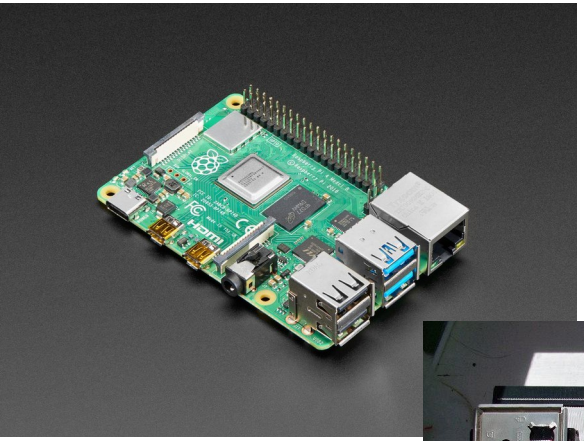
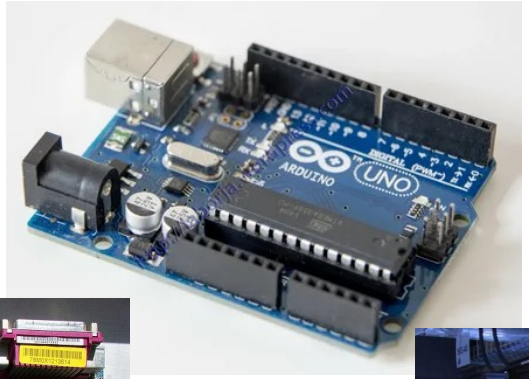
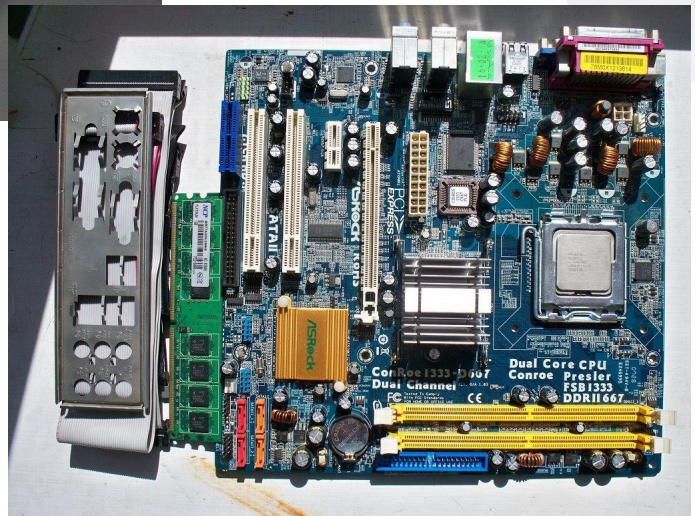
Архитектура вычислительных систем. Основные понятия







Определения архитектуры (А) ВС

Архитекту́ра компью́тера - концептуальная модель компьютерной системы, воплощённая в её компонентах, их взаимодействии между собой и с окружением, включающая также принципы её проектирования и развития. *Аспекты реализации (например, технология, применяемая при реализации памяти) не являются частью архитектуры.* [Таненбаум]

А. - абстрактное представление ВС, отражающее её *структурную, схемотехническую* и *логическую* организацию.

А. - множество взаимосвязанных компонент ВС, включающих: *программное обеспечение (software), аппаратное обеспечение (hardware), алгоритмическое обеспечение (brainware), специальное микропрограммное обеспечение (firmware)* + система (структура), поддерживающая сла́женное функционирование перечисленного.

А. - абстрактное многоу́ровневое представление физической системы с точки зрения программиста с закрепле́нием 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.**

Вычислительная система — компьютер, вычислительная машина (ВМ) в самом общем понимании

Архитектура – многогранное понятие

Критерии классификации архитектур ВС

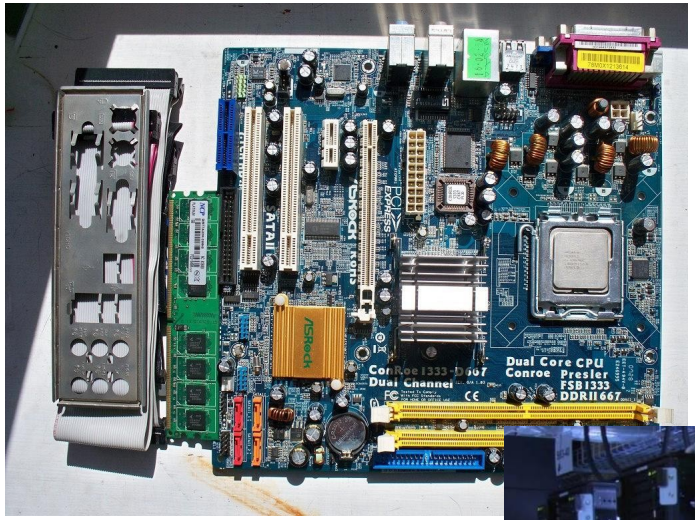
Уровень восприятия

Предметная ориентация (специализация)

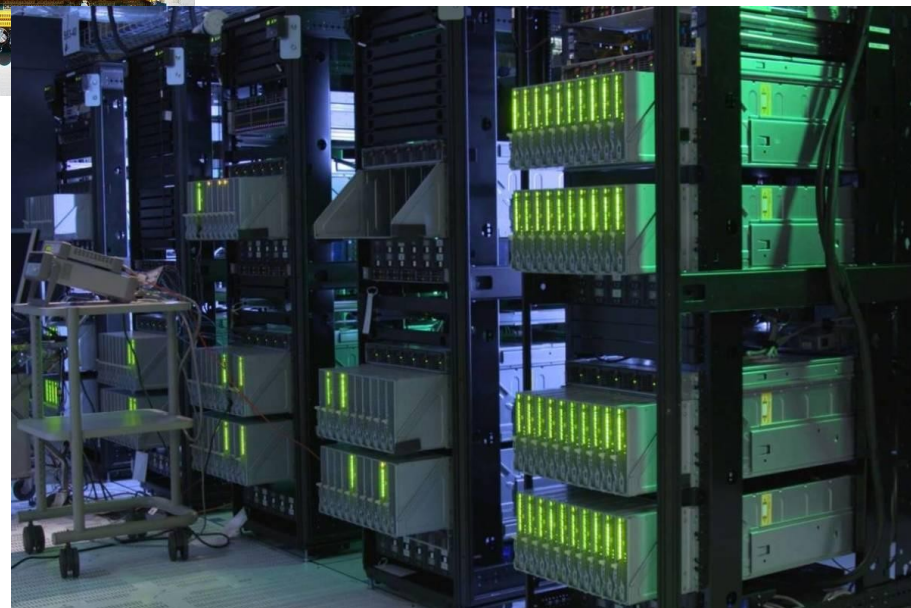
Структурная поддержка

Парадигма программирования

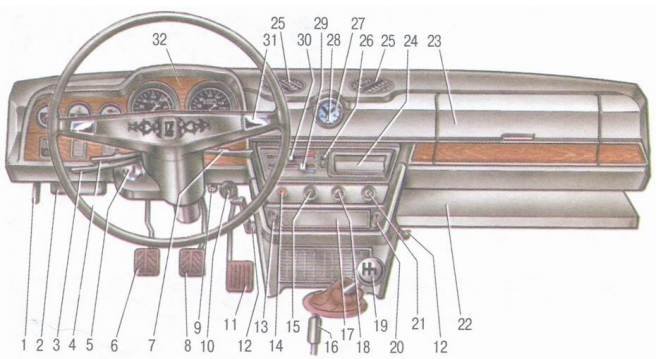
Уровень восприятия архитектуры ВС

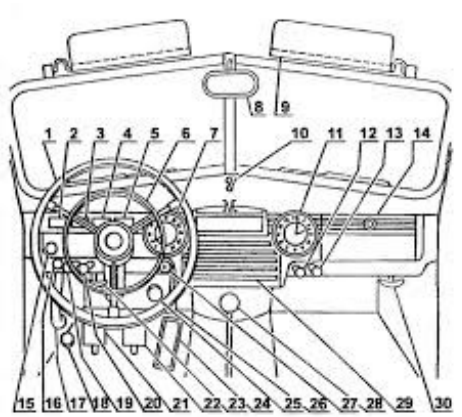


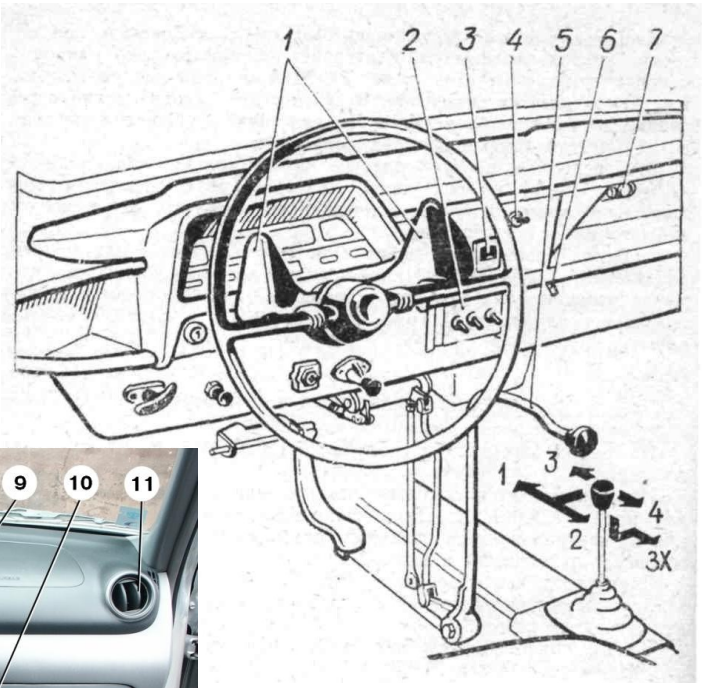
?

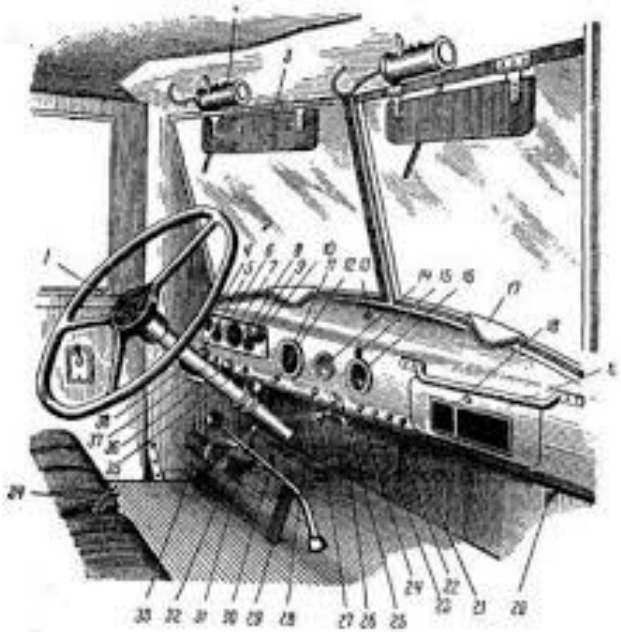


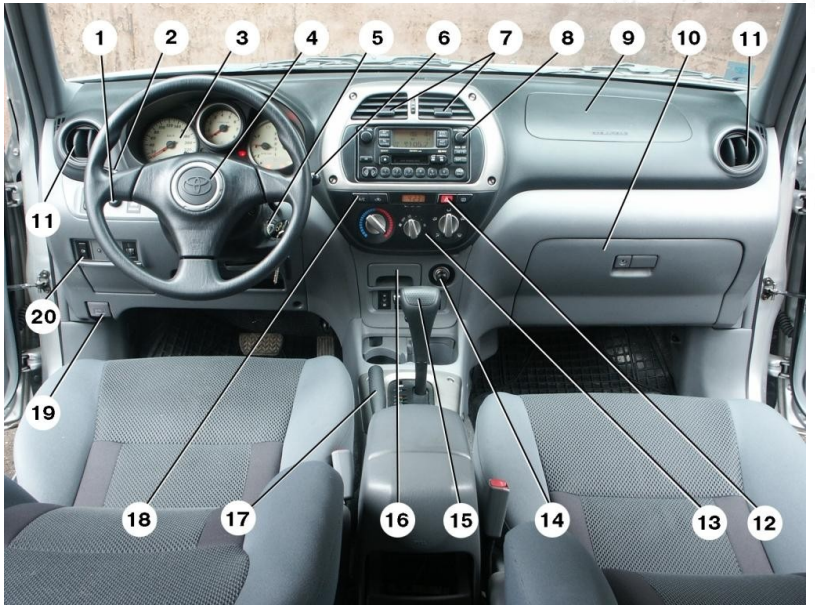
Уровень управления автомобилем



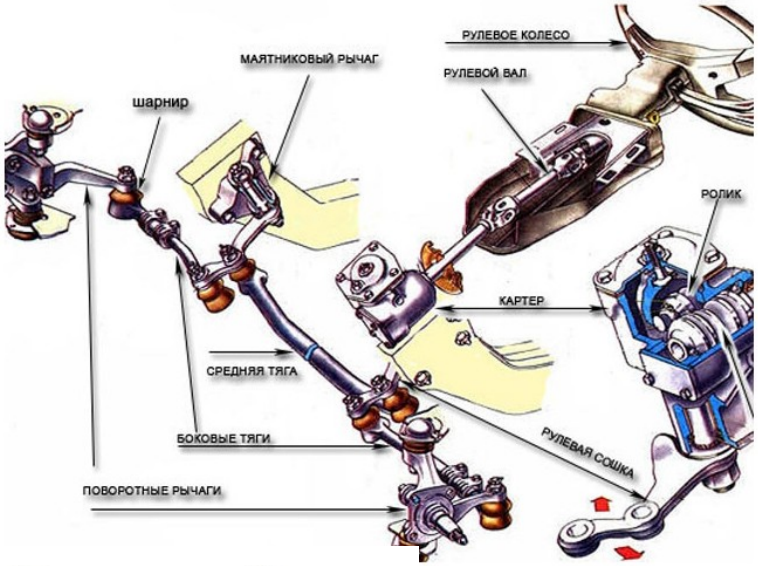


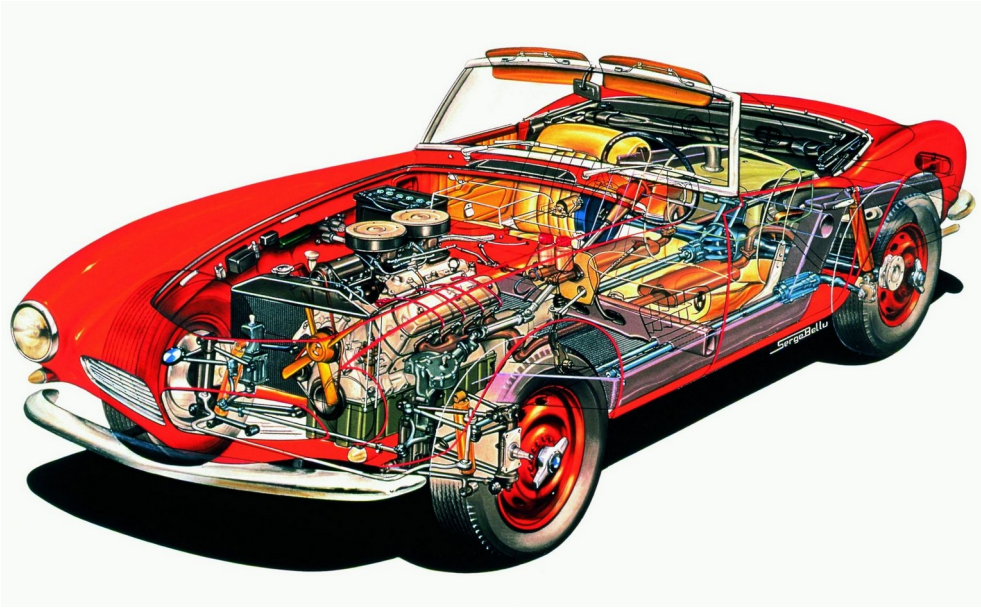


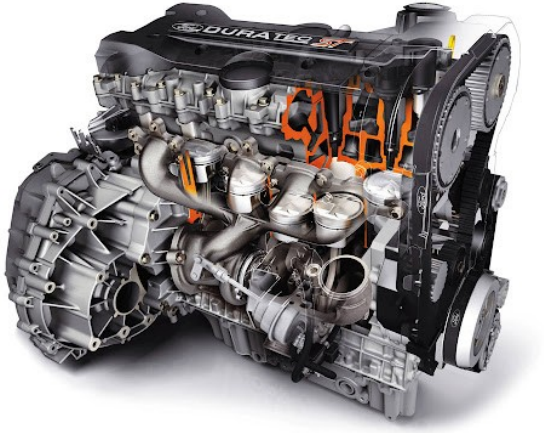




Уровени конструкции автомобиля и его узлов

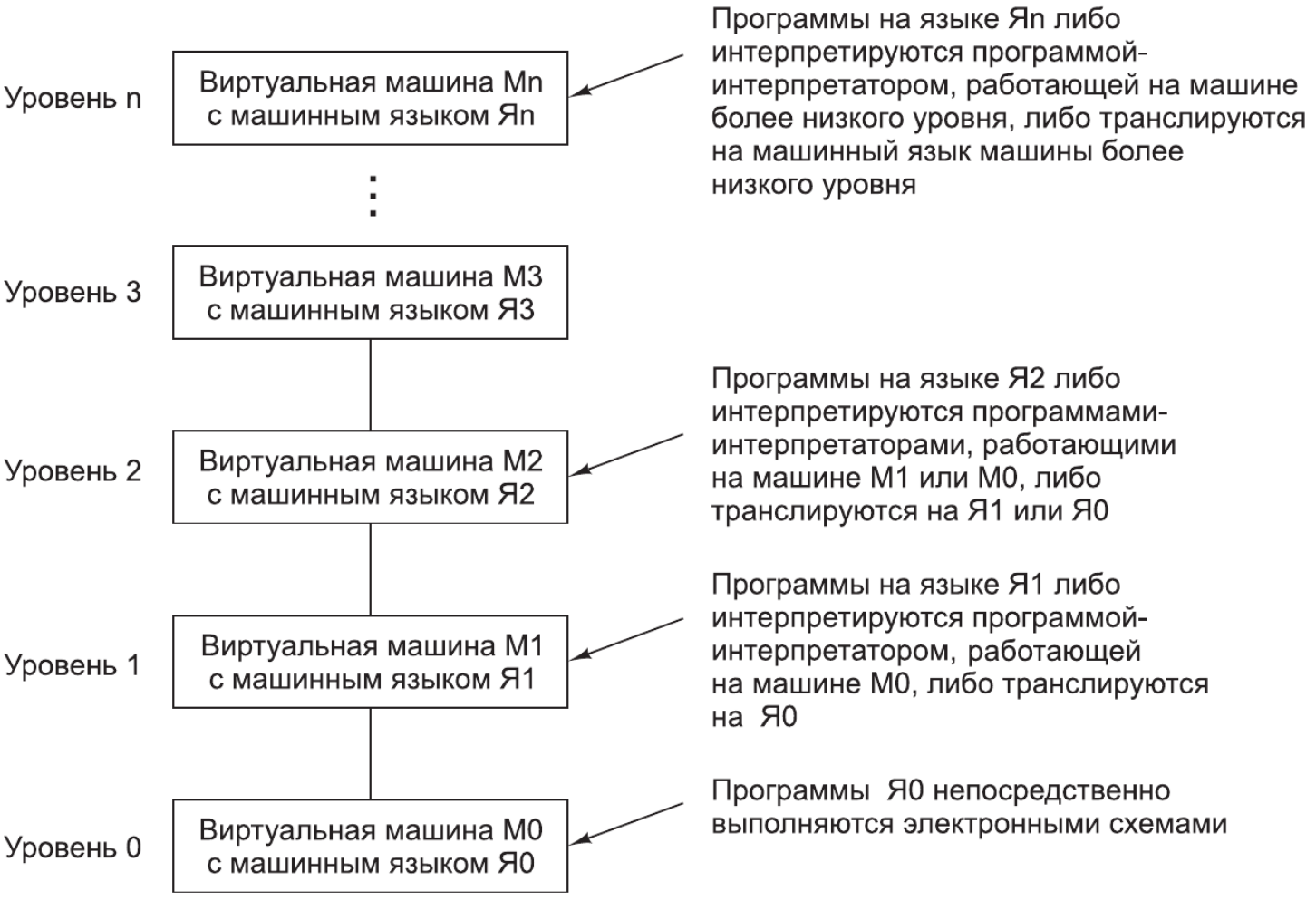






Уровень удобства использования (юзабилити) автомобиля

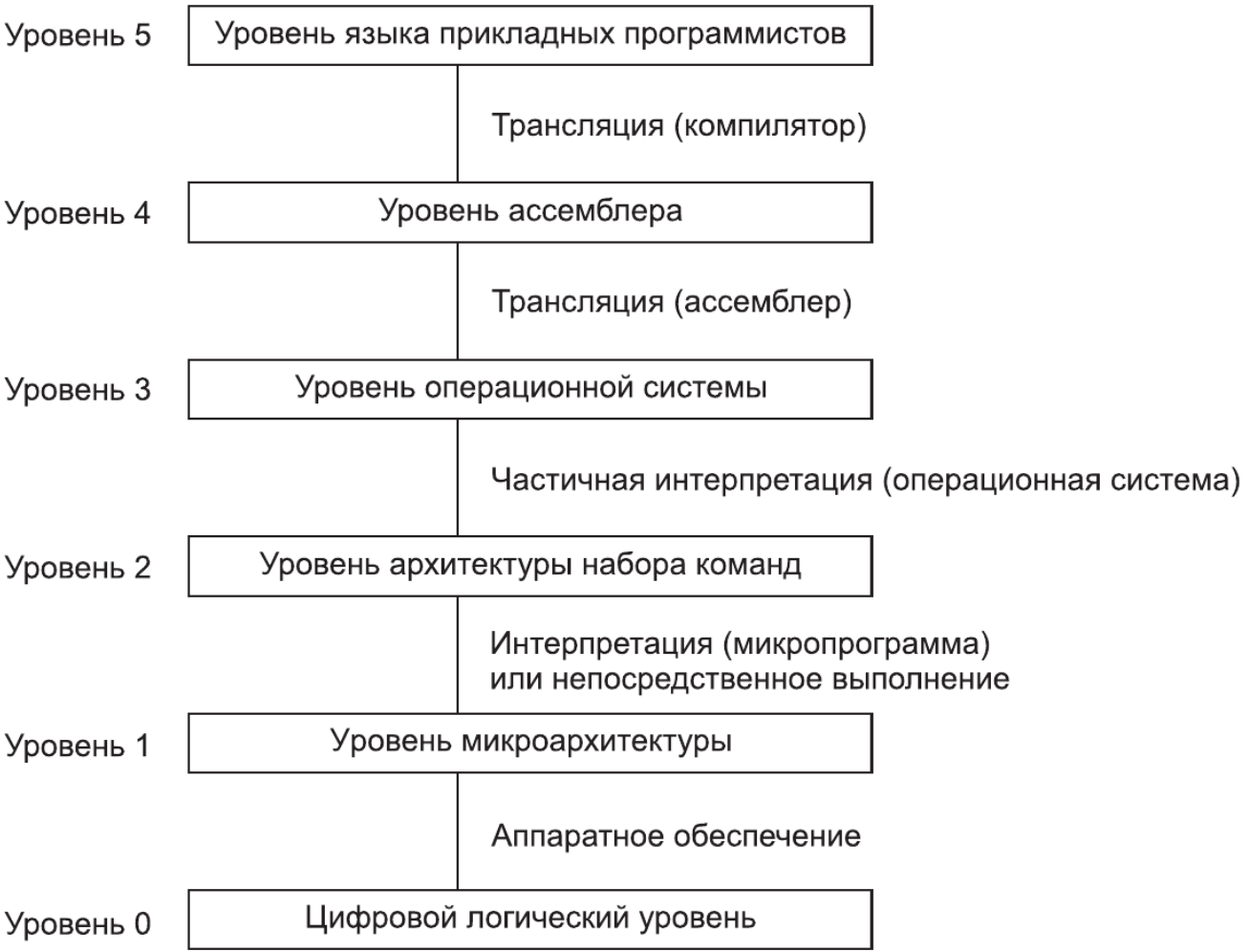
Многоуровневая организация архитектур ВС



[Таненбаум]

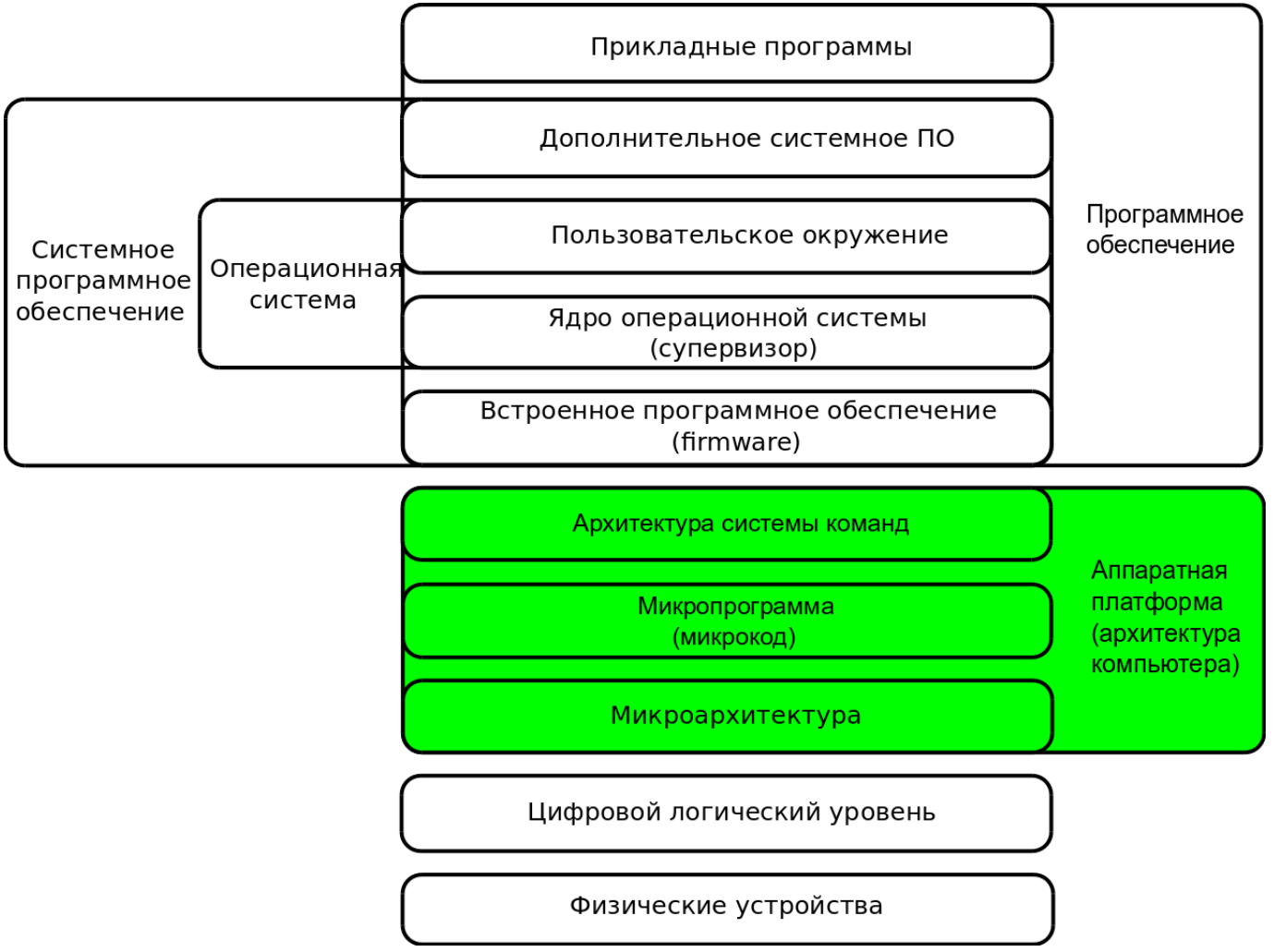
Компьютеры предназначены для программирования, а архитектурные уровни выделяются для повышения эффективности процесса разработки ПО, для преодоления семантического разрыва между предметной областью и реальным исполнителем, предоставляющи архитектуру на уровне системы команд.

Многоуровневая организация архитектур ВС



[Таненбаум]

Многоуровневая организация архитектур ВС



[Википедия]

Детализация архитектурных уровней

- Цифровой логическиий 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 языков (Norma, ANTLR, GPSS, Prolog, Cmake...)

Мотивы понимания многоуровневости

Многие классы задач требуют понимания и использования разнообразных по уровню компьютерных архитектур.

Часто в компаниях перебрасывают программистов с одного проекта на другой. При этом изменяются характеристики языковых и инструментальных средств, определяющих специфику уровней архитектуры.

Некоторые направления предметной области изменяются за счет автоматизации, отсутствия спроса и по другим причинам. Это ведет к необходимости переориентации на другие задачи, зачастую меняющие уровень используемой компьютерной архитектуры.

Незнание методов и подходов других архитектурных уровней зачастую ведет к неэффективному решению поставленной задачи за счет неправильно выбора инструментов, которые соответствуют текущим знаниям разработчика, но не соответствуют архитектурному уровню решаемой задачи.

- Несоответствие вниз
- Несоответствие вверх

Пример. Проявление многоуровневости в C++

Прикладной уровень => библиотеки
- Стандартная библиотека (универсальность)
- Boost (универсальность)
- SFML (специализация)
- SDL (специализация)

Системный уровень => (уровень данного языка)
- Генераторы компиляторов (специализация)
- Встроенный ассемблер (уровень команд)

Параллелизм (специализация)
- Pthread (системный уровень)
- OpenMP(прикладной уровень)
- MPI (прикладной уровень)

Специализация

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
Инструменты для разработки ПО
Архитектура ВС

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 архитектуровой ВС и другими причиннами.

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

Критерии качества программного обеспечения

Корректность (правильность). Обеспечивает правильную обработку на правильных данных.
Устойчивость. "Элегантное" 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.

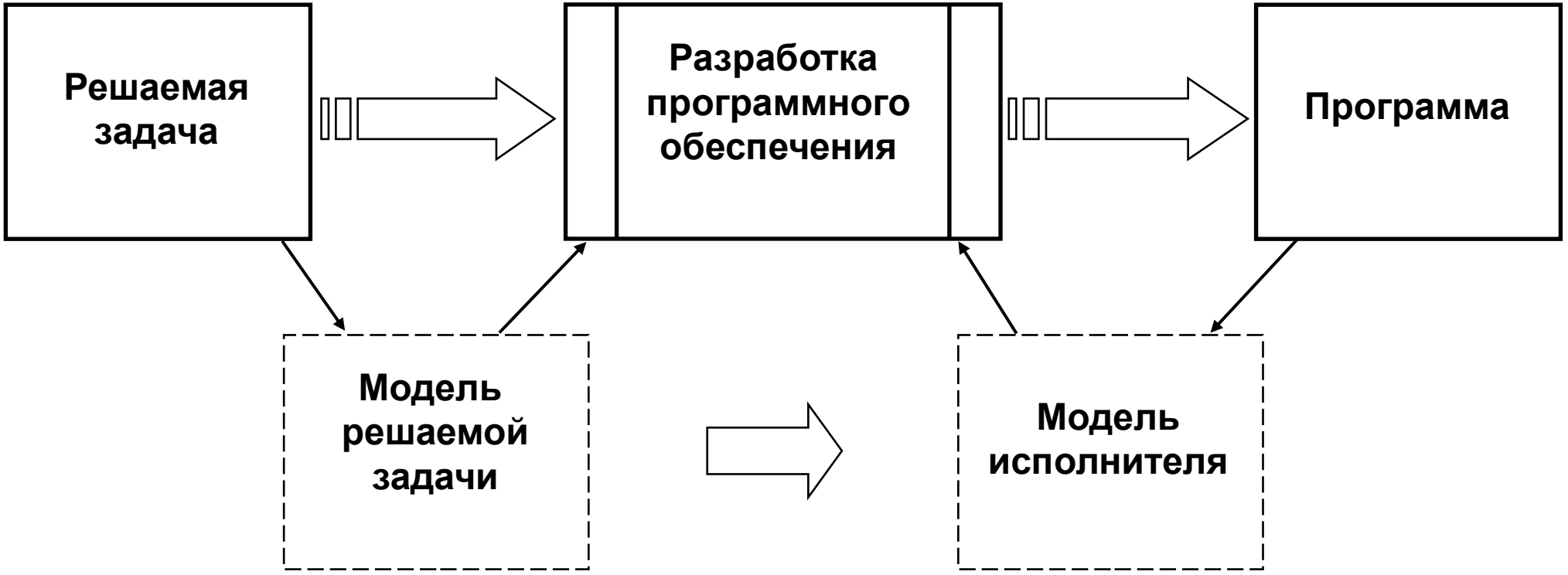
Решаемая задача описывается совокупностью формальных и эмпирических (неформальных) **моделей**, определяющих как протекающие **процессы**, так и используемые при этом **данные**.

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**).

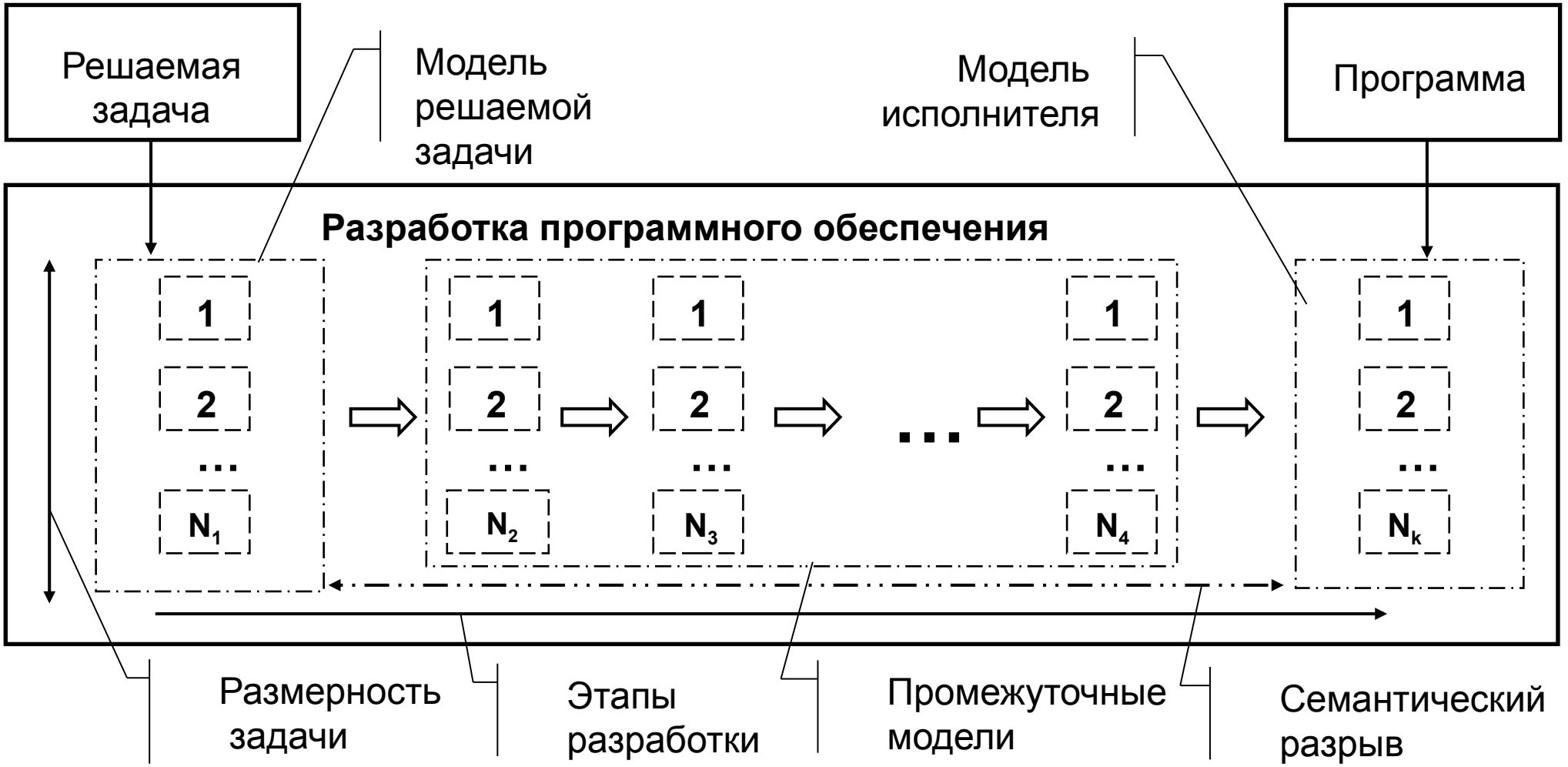


Создаваемая программа является отображением модели решаемой задачи на модель исполнителя

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.

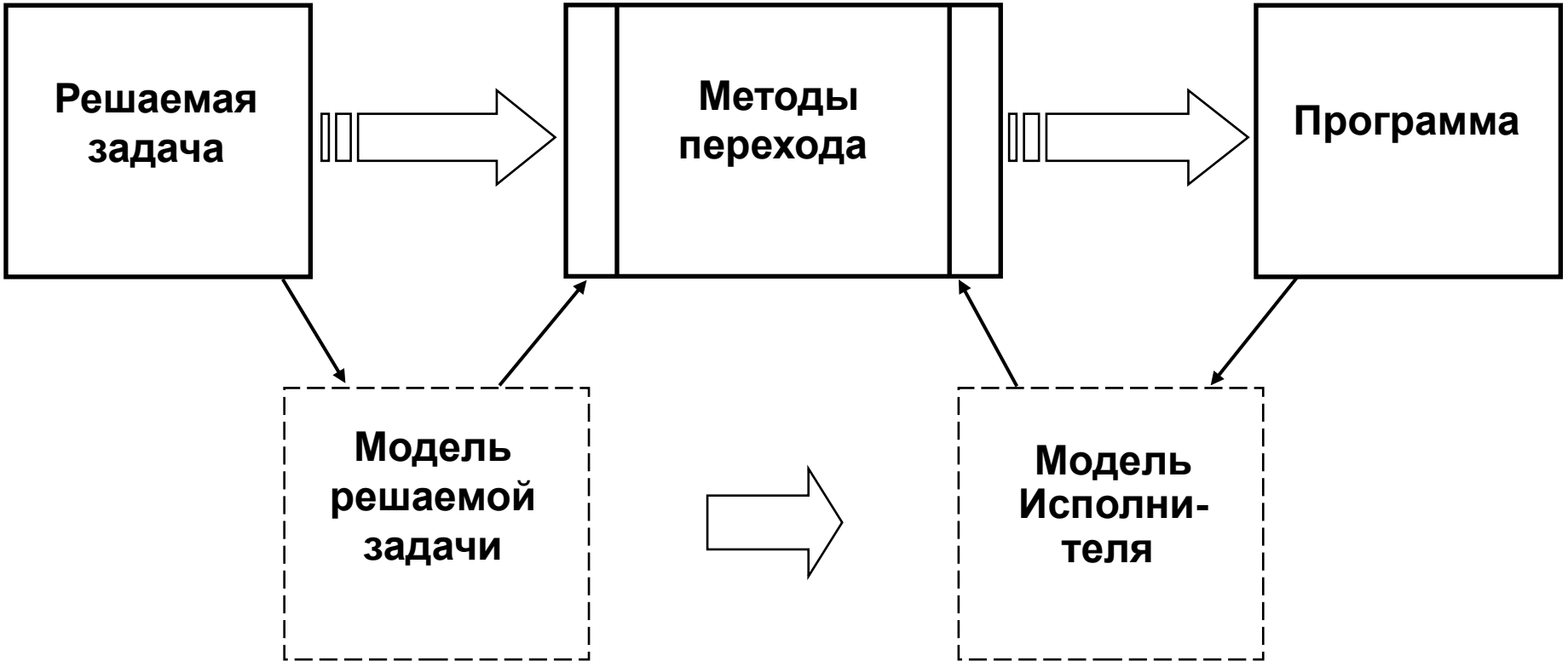
Позволяют ускорить процесс разработки следующими способами:

- **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.

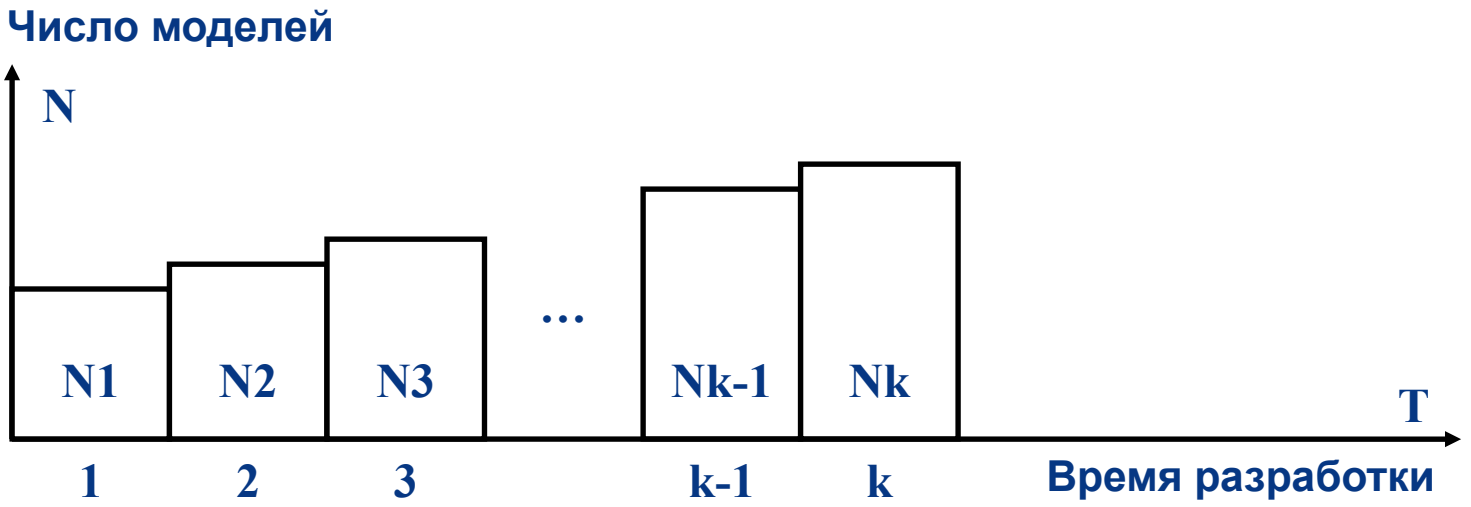
2. Разработка программных систем на основе теории автоматов. Последующая программная реализация автоматов является хорошо отработанным формальным приемом с применением различных технологий.

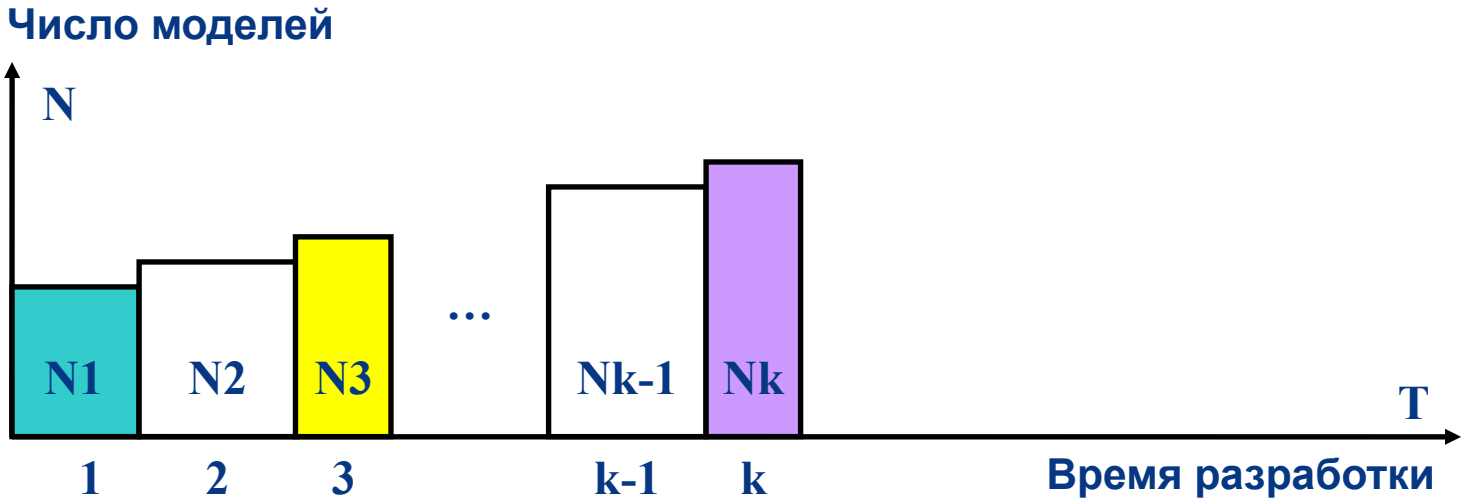
3. Нейроннные сети. Отработанные теоретические подходы, формализованные в различные наборы инструментов (Tensorflow, Keras, Theano…)

...

Что это дает →

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





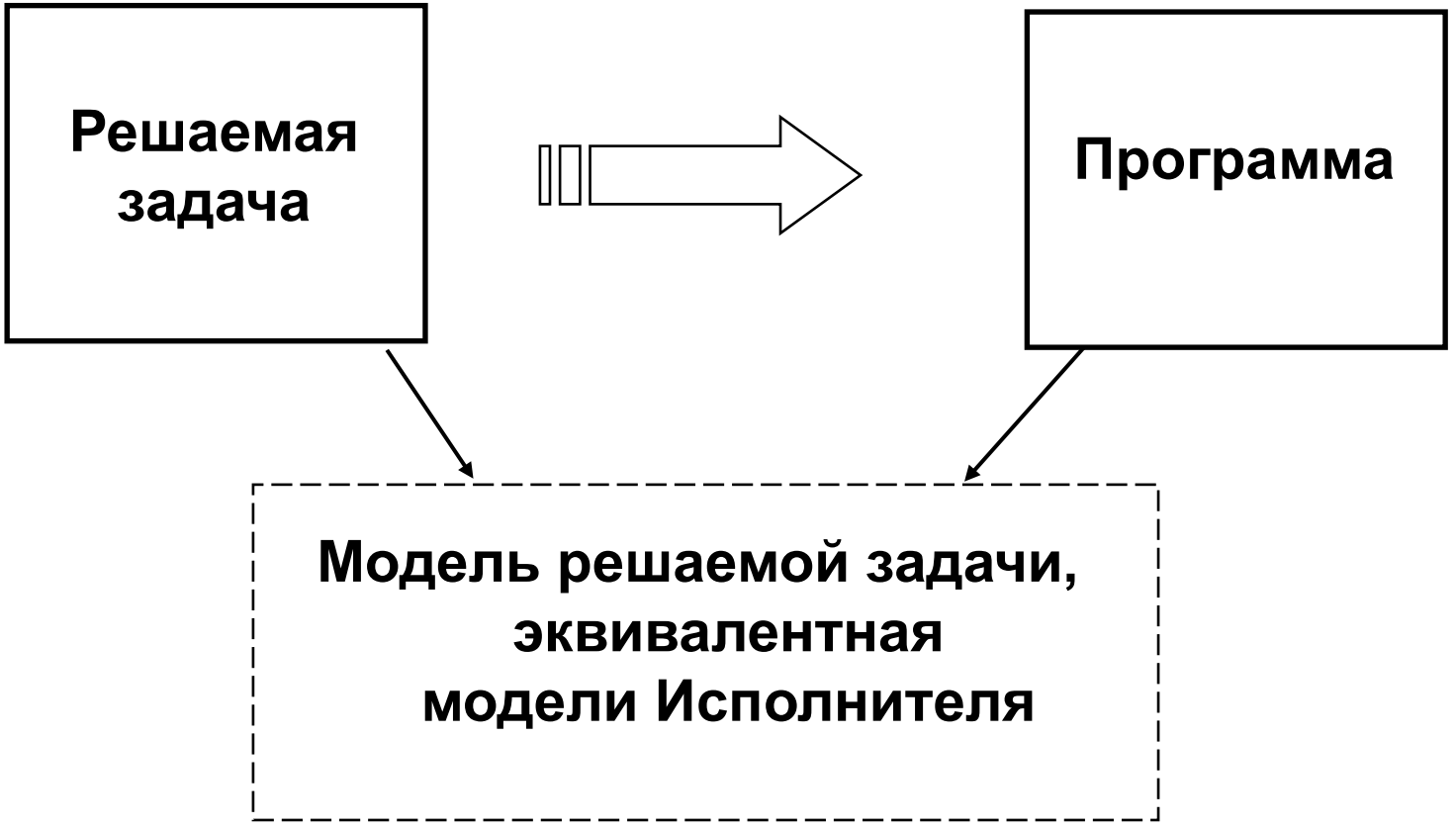
Предметно-ориентированные архитектуры

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**.

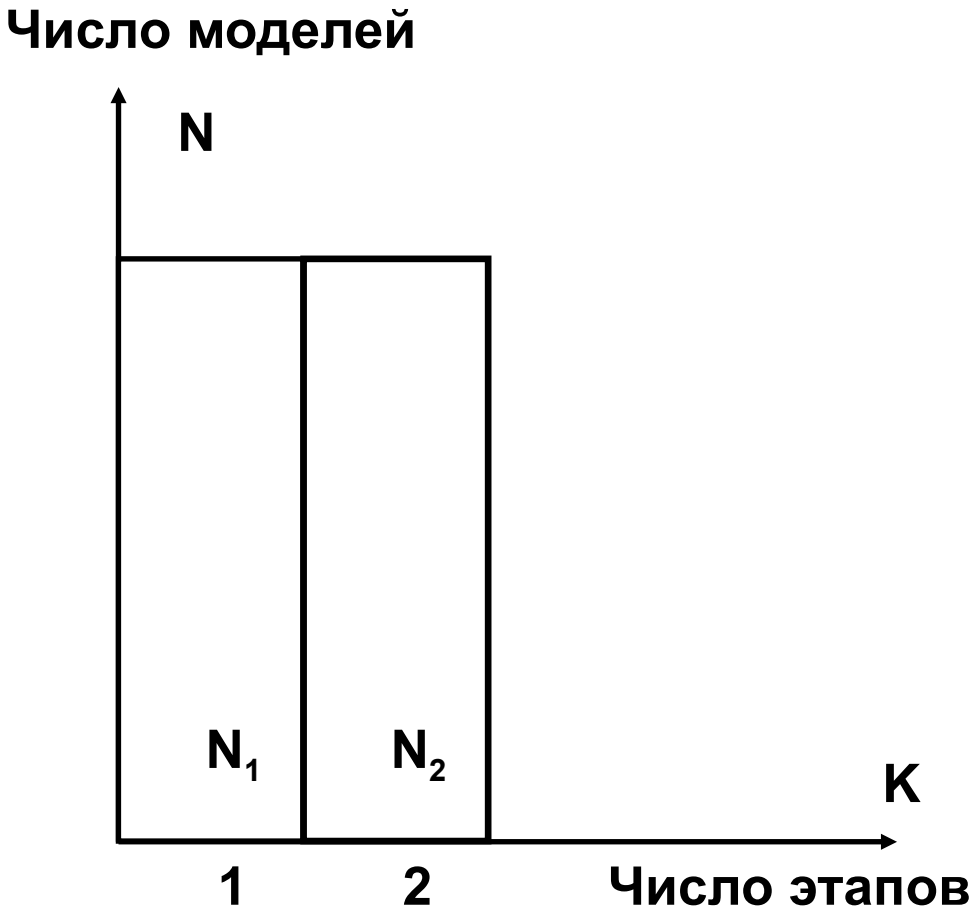
Фактически это означает слияние моделей задачи и исполнителя.

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)



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

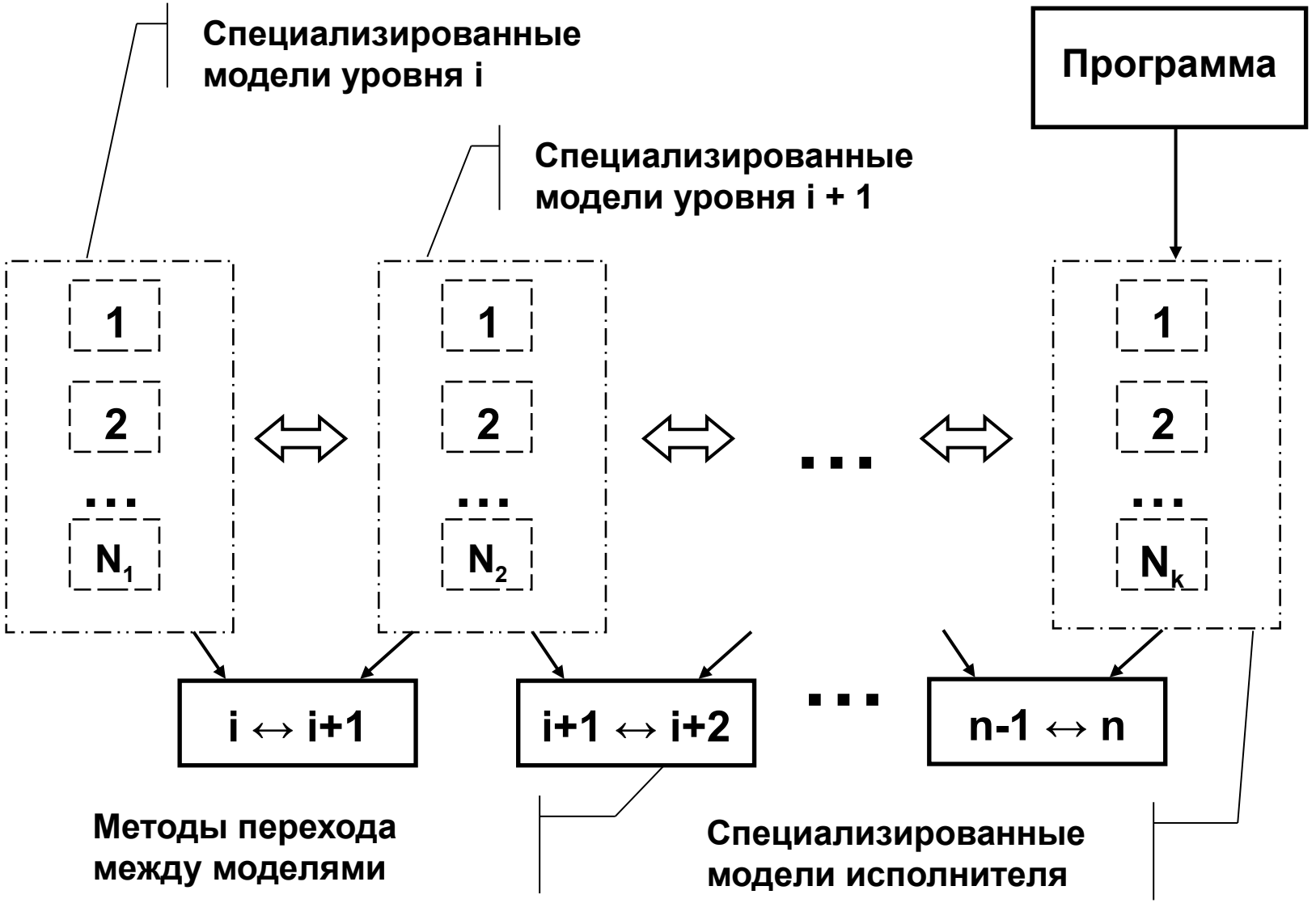
Методики разработки ПО

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**.

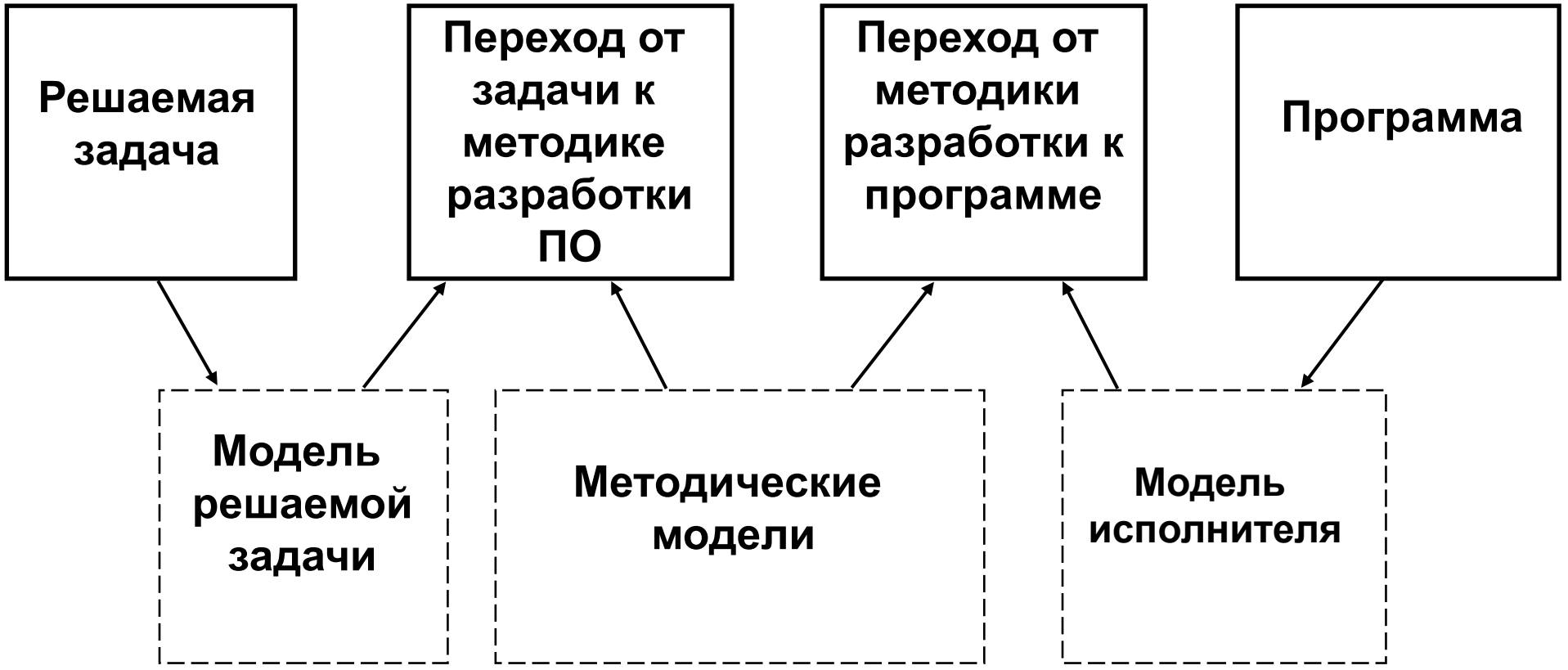
Поддерживают процесс преобразования **как от модели задачи к модели исполнителя,так и в обратном направлении**.


Специализированные модели уровня i
Специализированные модели уровня i + 1
Программа
1
2
...
N1
1
2
...
N2
...
1
2
...
Nk
i ↔ i+1
i+1 ↔ i+2
...
n-1 ↔ n
Методы перехода между моделями
Специализированные модели исполнителя

Примеры методик:

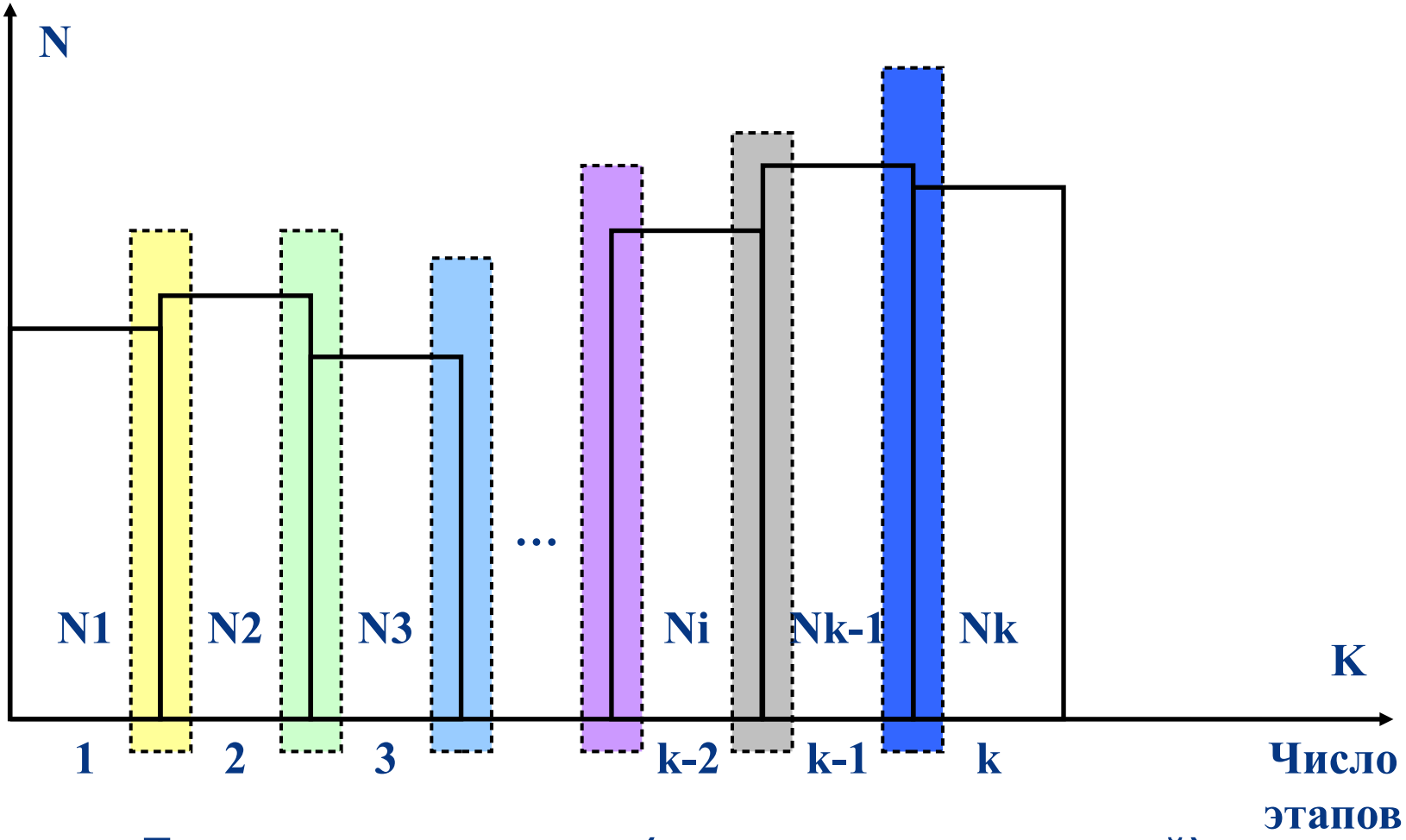
- **объектно-ориентированный подход**, используемый в составе объектно-ориентированной методологии;
- **методы структурного анализа и проектирования**, применяемые при разработке информационных систем;
- **методы быстрой разработки приложений**, ориентированные на построение программ от моделей, определяющих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

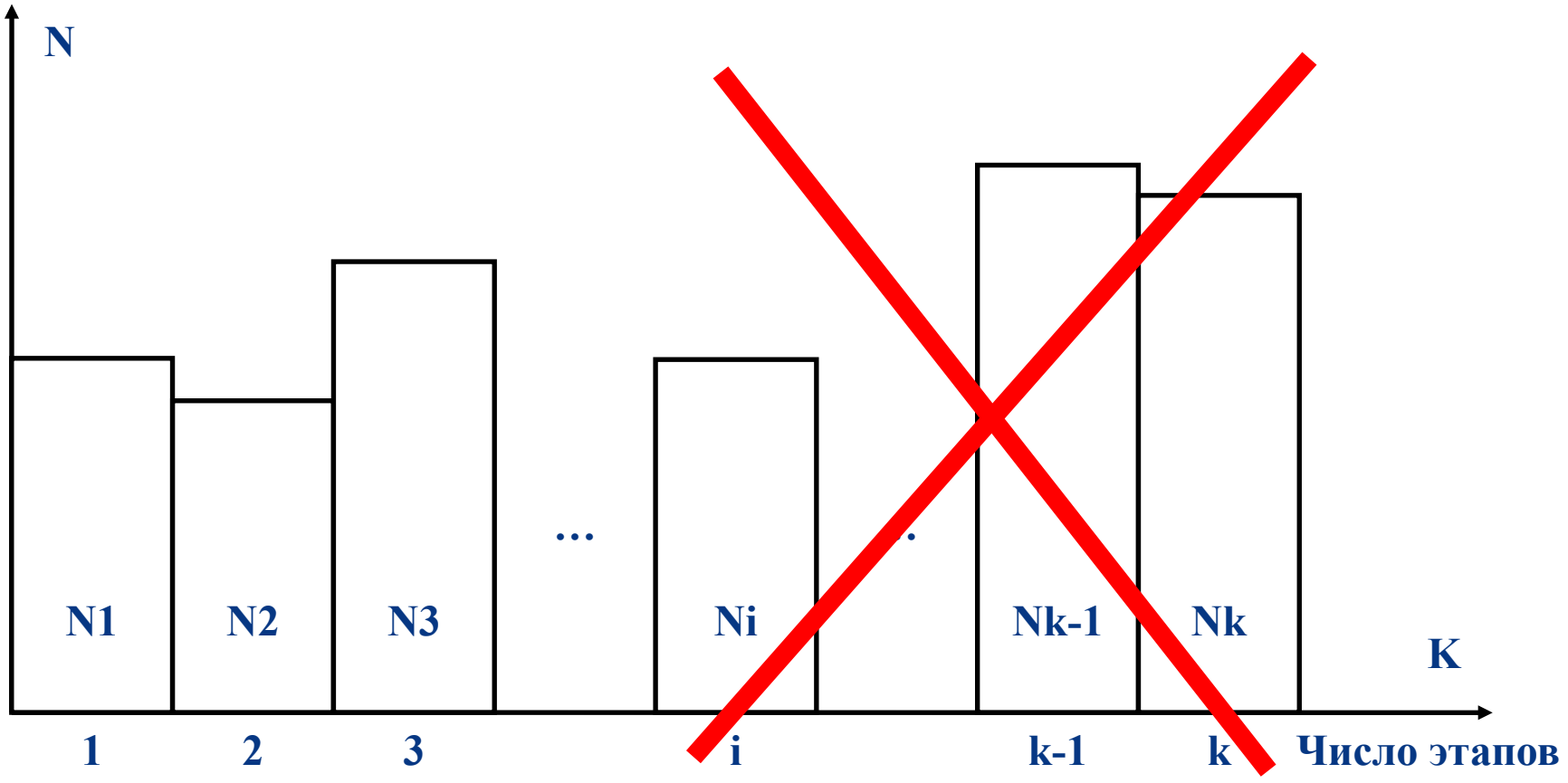
- средства отладки
- средства тестирования
- средства профилирования
- …

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***).

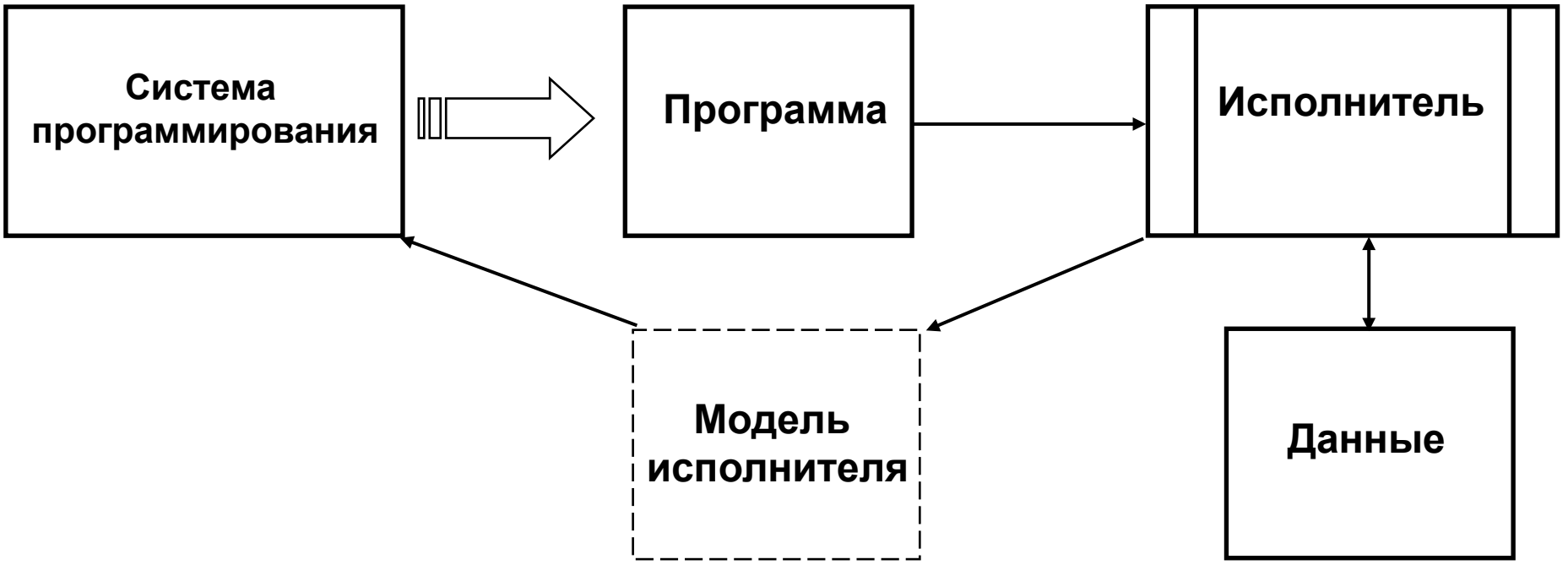
Число моделей



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.



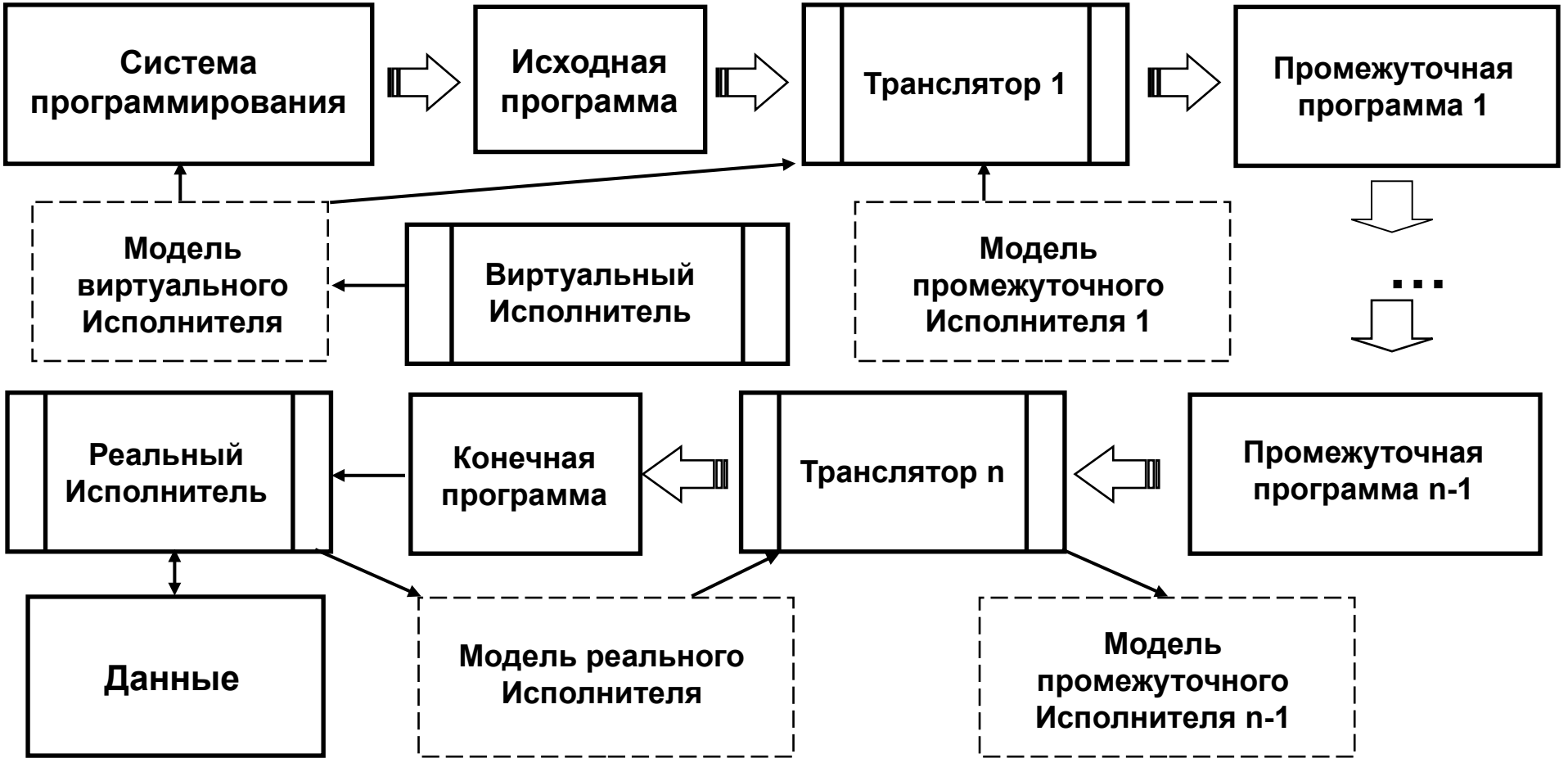
Непосредственное использование исполнителем программы, написанной с применением системы программирования

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 информации по данной теме

1. [Таненбаум] Э.Таненбаум. Архитектура компьютера. 6-е изд. — СПб.: Изд. Питер, 2017. – 816 с.: с ил. + CD-ROM. *(2013-2018)*

2. [Википедия] Архитектура компьютера. Статья в википедии - https://ru.wikipedia.org/wiki/Архитектура_компьютера

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