Диаграммы, схемы, графы

Применение средств визуализации
для документирования моделей

Описание процессов

Описание процессов – документирование процесса в свободной форме, например, простое текстовое описание пользовательских сценариев *(Use Case):*

- процесс может быть представлен в любом виде – текстовом, табличном, графическом *(в свободной нотации)*;

- *возможно,* без формальной логики и специальных обозначений и ограничений.

Нотации – формализованные литеральные/графические модели, которые используются, чтобы фиксировать процессы, анализировать поведение и оптимизировать результаты.

Моделирование процессов

= формализованная процедура, подразумевающая создание некоторой формальной модели процесса, описанной на математическом или другом формализованном языке:

- формальное представление процесса с помощью общеизвестных методологий, методов и нотаций;

- методология моделирования определяет системные основы исследования, совокупность методов и принципов построения моделей.

Назначение UML / Unified Modeling Language

Моделирование = *(контекстно)*

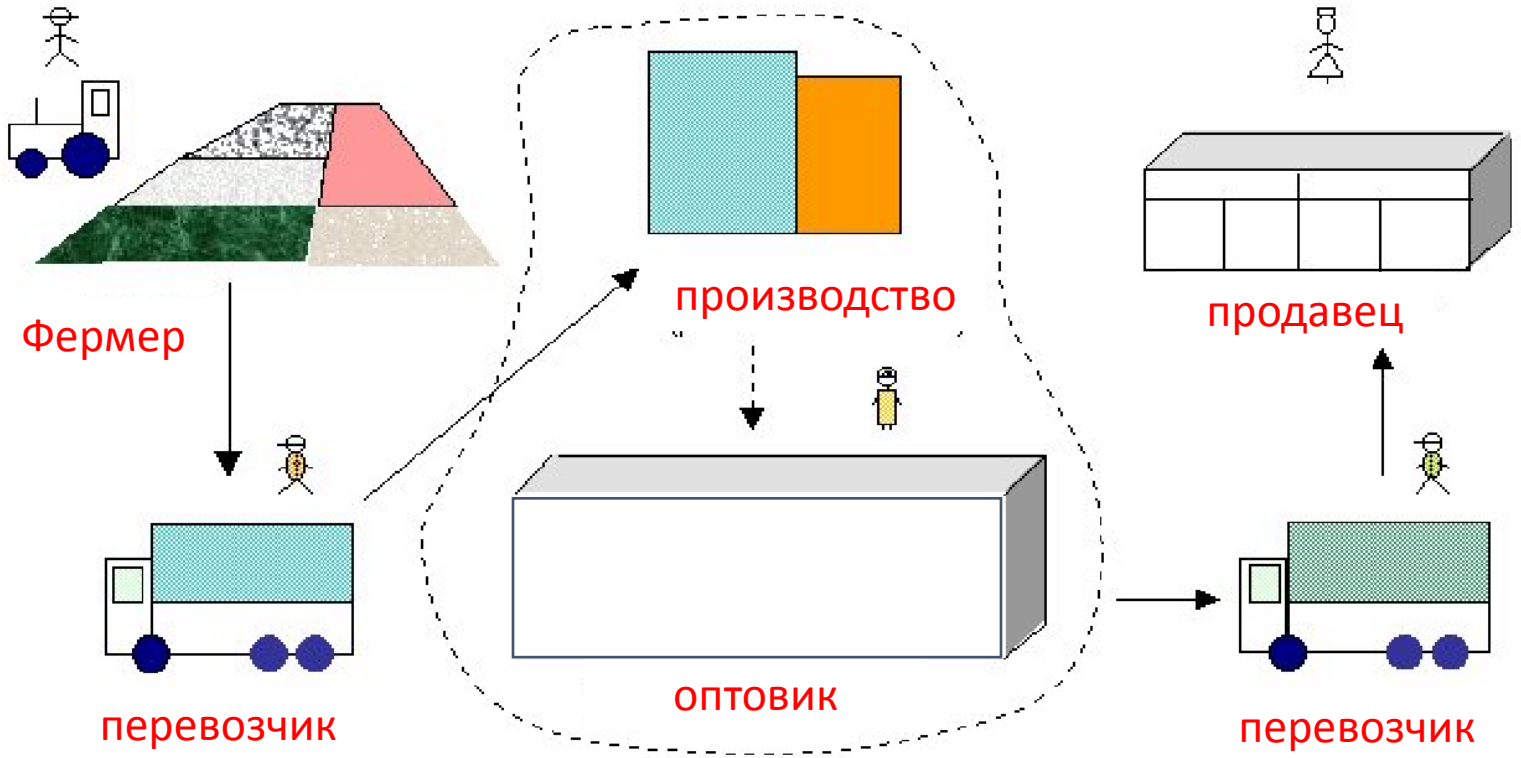
- ✓ Спецификация
  - ❖ формальная и наглядная нотация [*синтаксис*]
- ✓ Визуализация
  - ❖ для общения (исполнители, заказчики, владельцы)
- ✓ Проектирование
  - ❖ архитектура / конструирование [*семантика*]
- ✓ Документирование
  - ❖ по разным аспектам [*прагматика*]

Визуализация / инфографика

Идея: «сухой текст» хуже, чем «текст с картинками»,
а еще лучше «картинки с текстом»

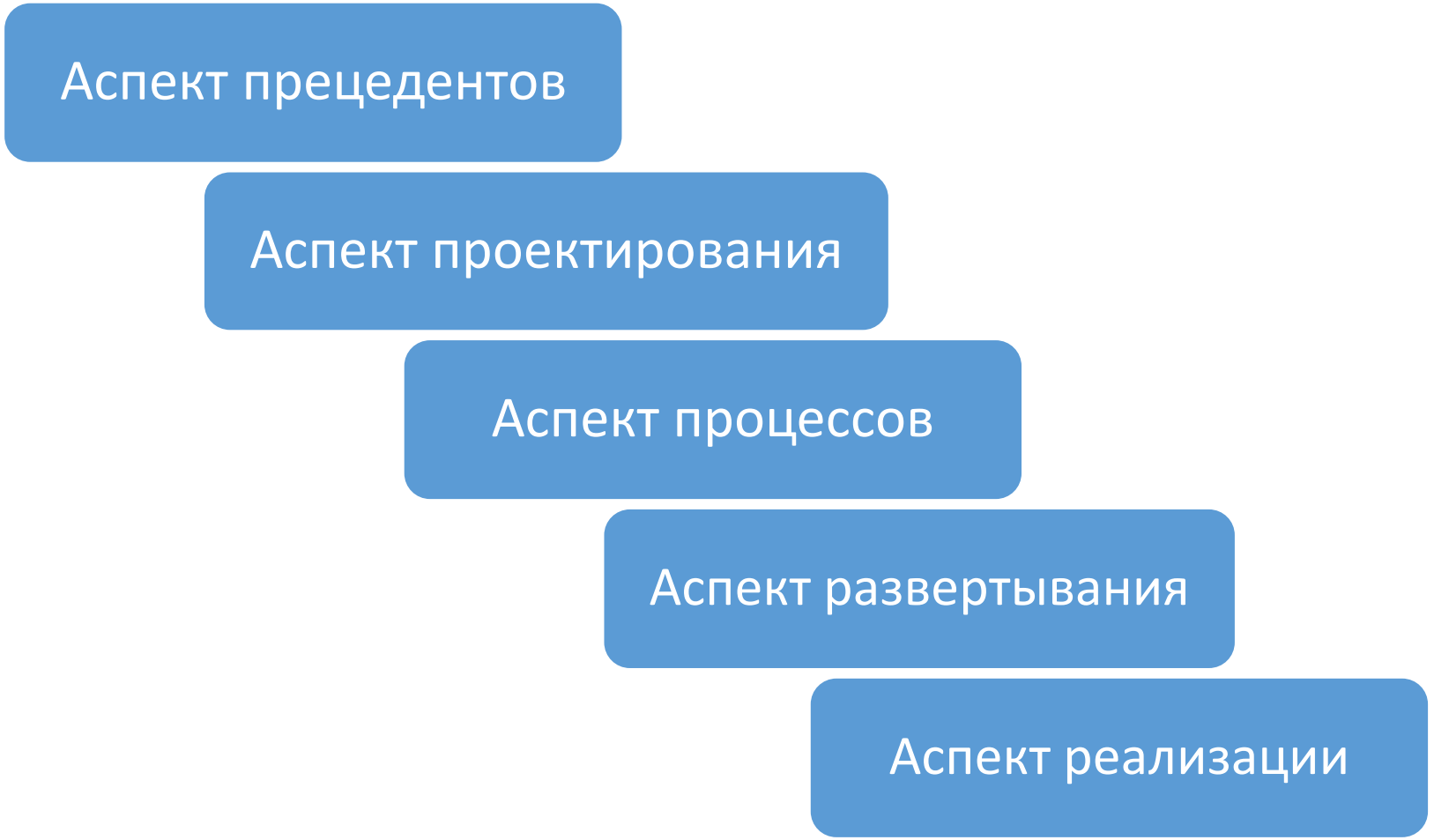
=> **UML**

➔ «комиксы»



Назначение UML / Unified Modeling Language

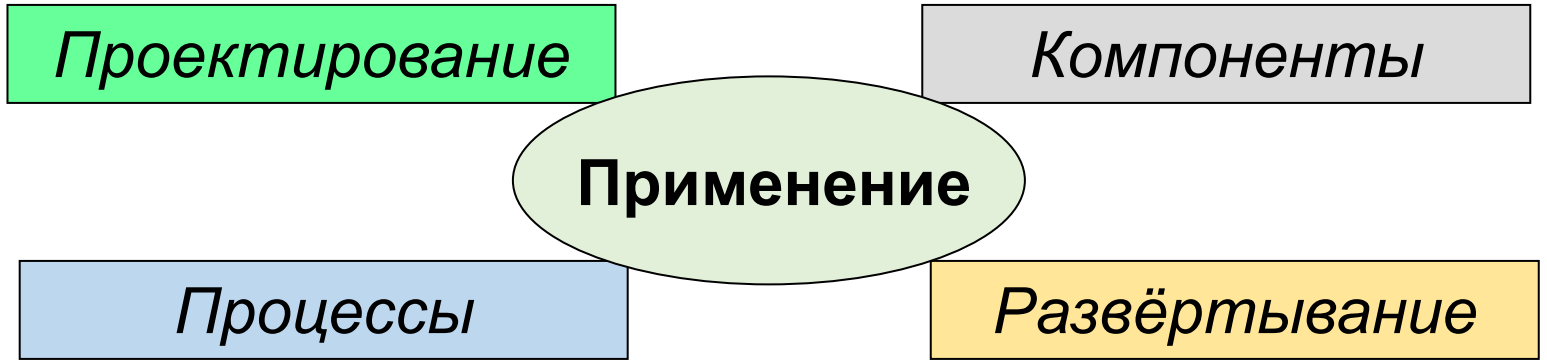
нотация UML предлагает пять представлений (аспектов) системы



Представление жизненного цикла ИС

словарь предметной области, функциональность

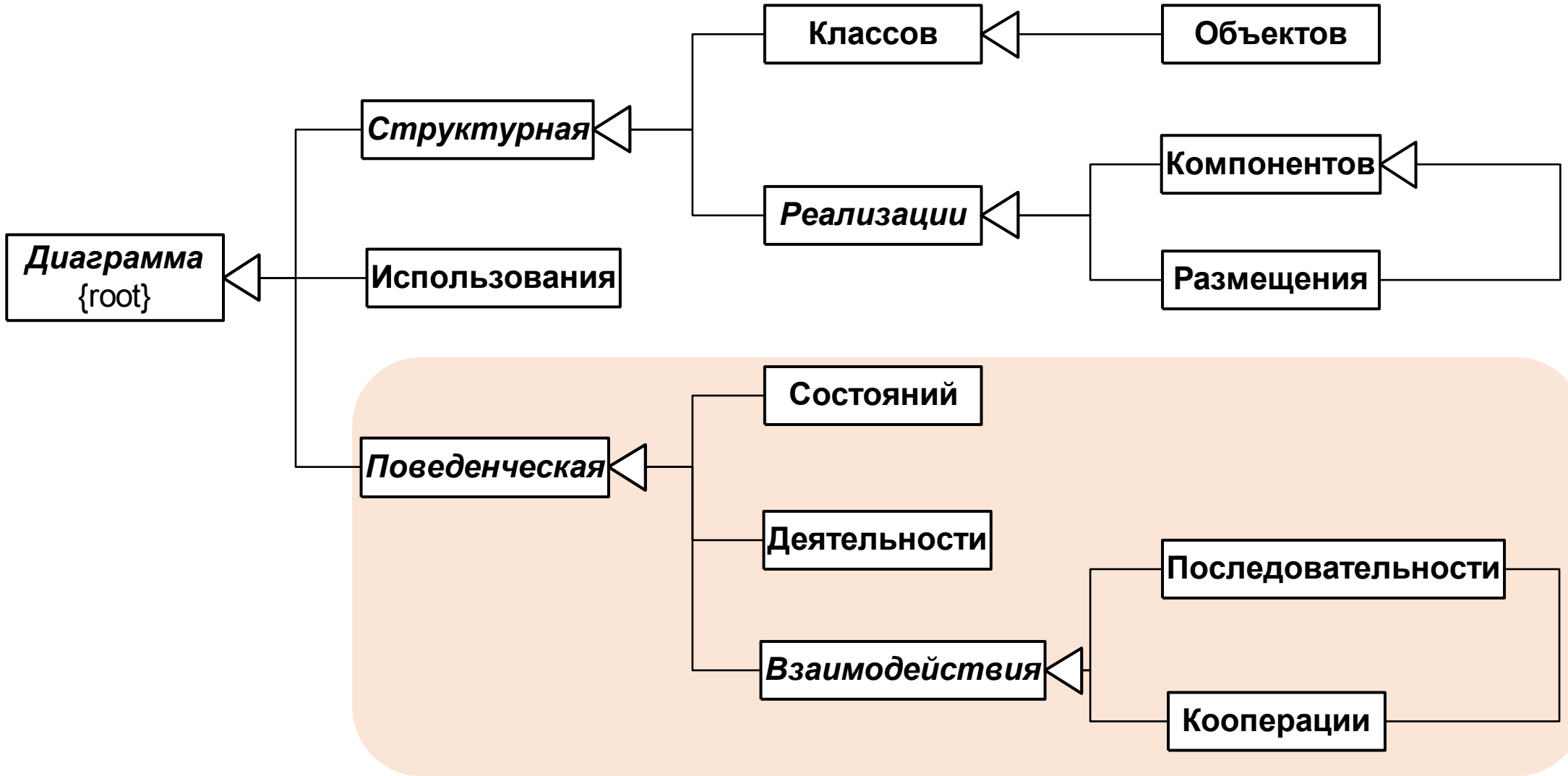
сборка системы, управление конфигурацией



деятельность, масштабируемость, пропускная способность

распределение, установка, топология системы

Типы диаграмм UML



Представление поведения

Ответ на вопрос: Как работает система ?

Диаграммы -

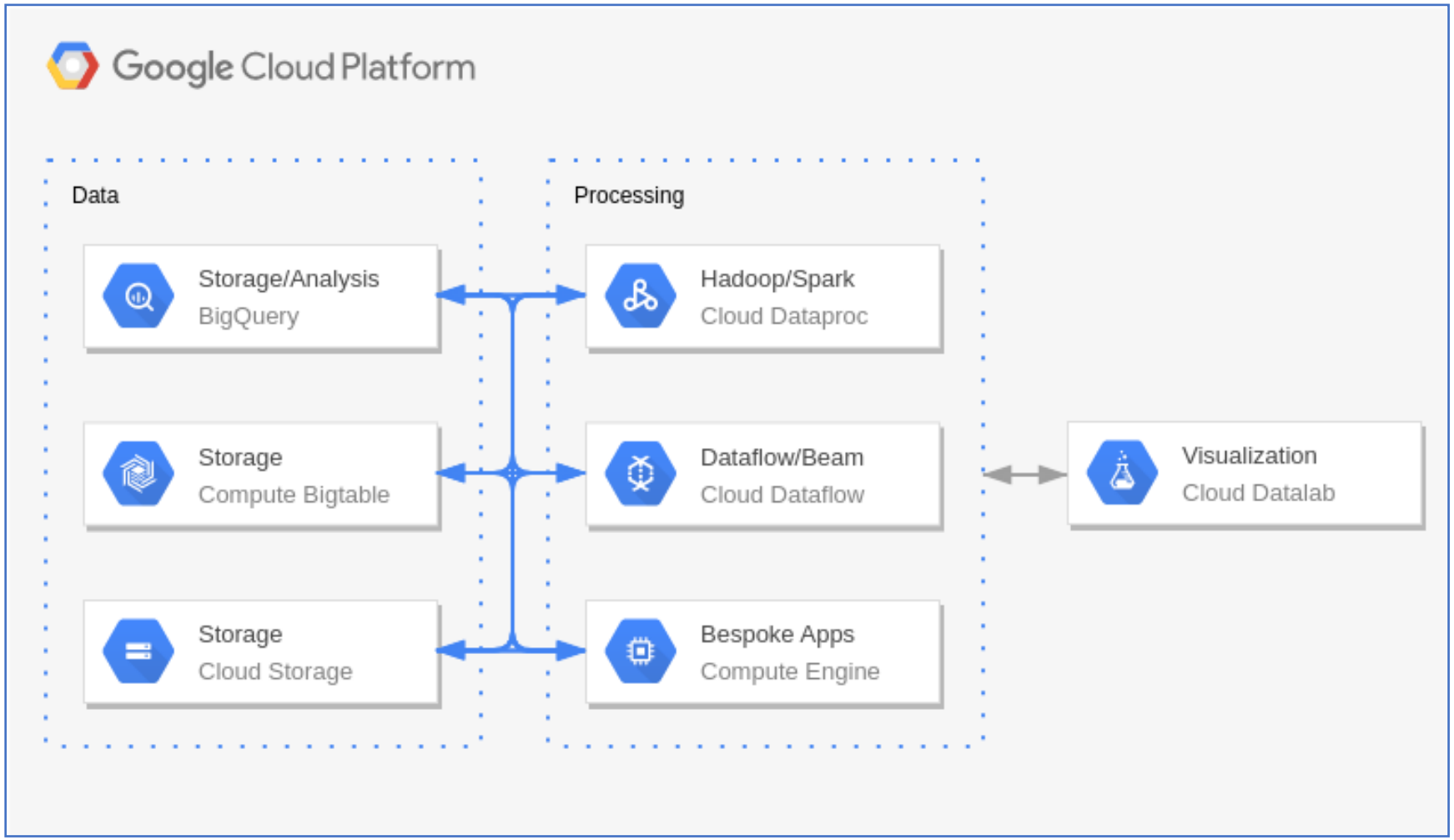
- состояний
- деятельностей
- взаимодействия
- последовательности / коммуникации

В случае необходимости элементы UML могут быть расширены и переопределены !

Облачные сервисы UML

https://online.visual-paradigm.com

https://www.smartdraw.com/uml-diagram



Практика применения диаграмм

- ✓ Простые идеи и нотация:

- применяется на всех фазах жизненного цикла

- понимают все (разработчики, заказчики, управленцы) одинаково

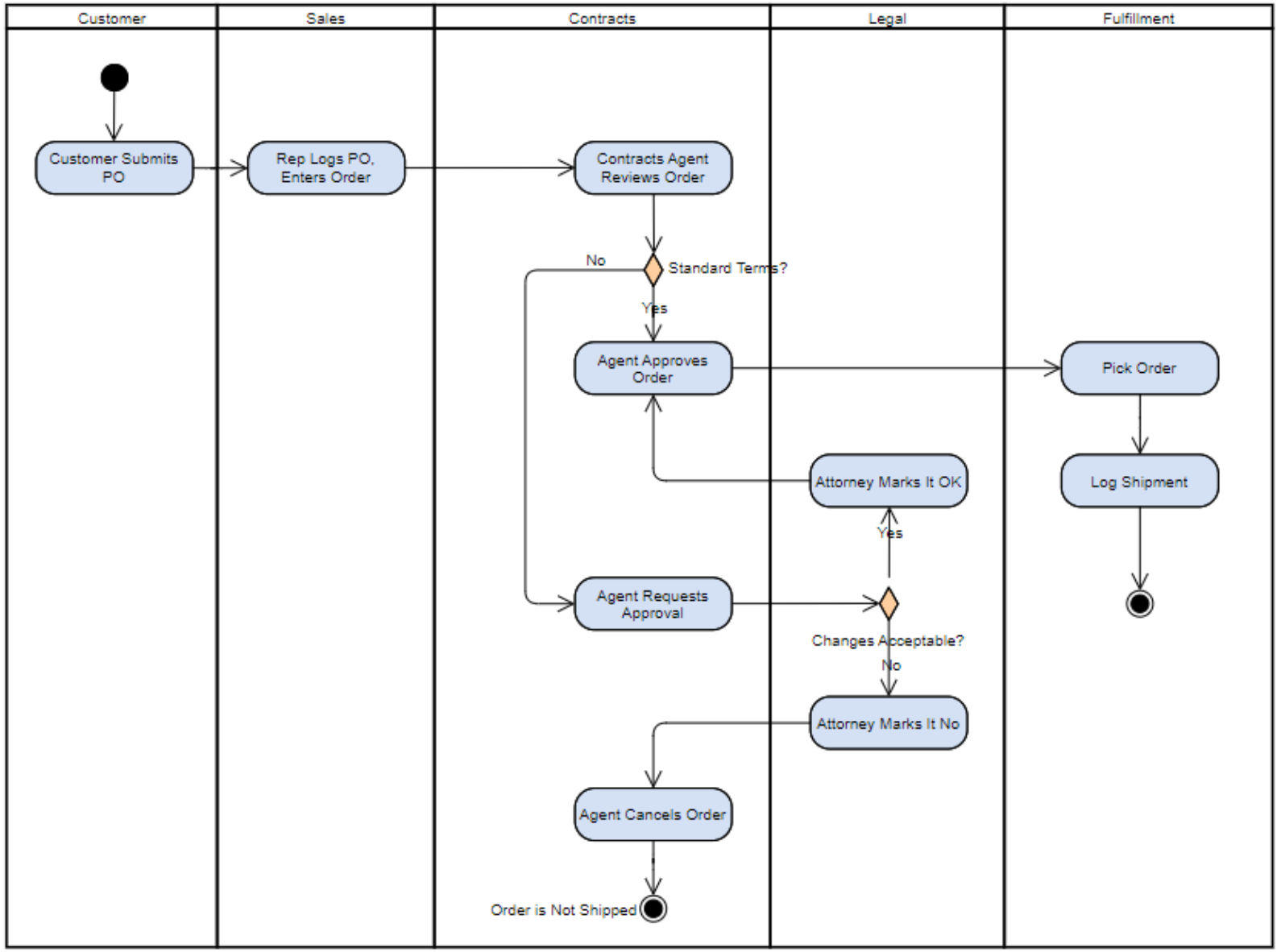
- ✓ Моделирование использования позволяет начинающему разработчику совершать меньше грубых ошибок на ранних этапах проектирования

- ❖ Опытный разработчик также может использовать и другие подходы

Диаграммы моделирования *поведения*

- Диаграммы состояний
- Диаграммы деятельностей
- Диаграммы взаимодействия
- Диаграммы коммуникации
- Диаграммы последовательности
- Моделирование параллелизма

Диаграмма деятельностей



Диаграммы деятельностей

Диаграммы деятельностей позволяют моделировать сложный жизненный цикл объекта с переходами из одного состояния (деятельности) в другое.

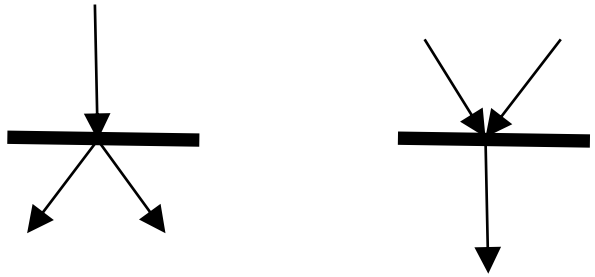
Этот вид диаграмм может быть использован для описания динамики совокупности объектов. Они применимы для детализации некоторой конкретной операции и предоставляют больше возможностей, чем классические блок-схемы.

Диаграммы деятельностей описывают переход от одной деятельности к другой (в отличие от диаграмм взаимодействия, где акцент делается на переходах потока управления от объекта к объекту).

Деятельность - это протяженное по времени составное действие.

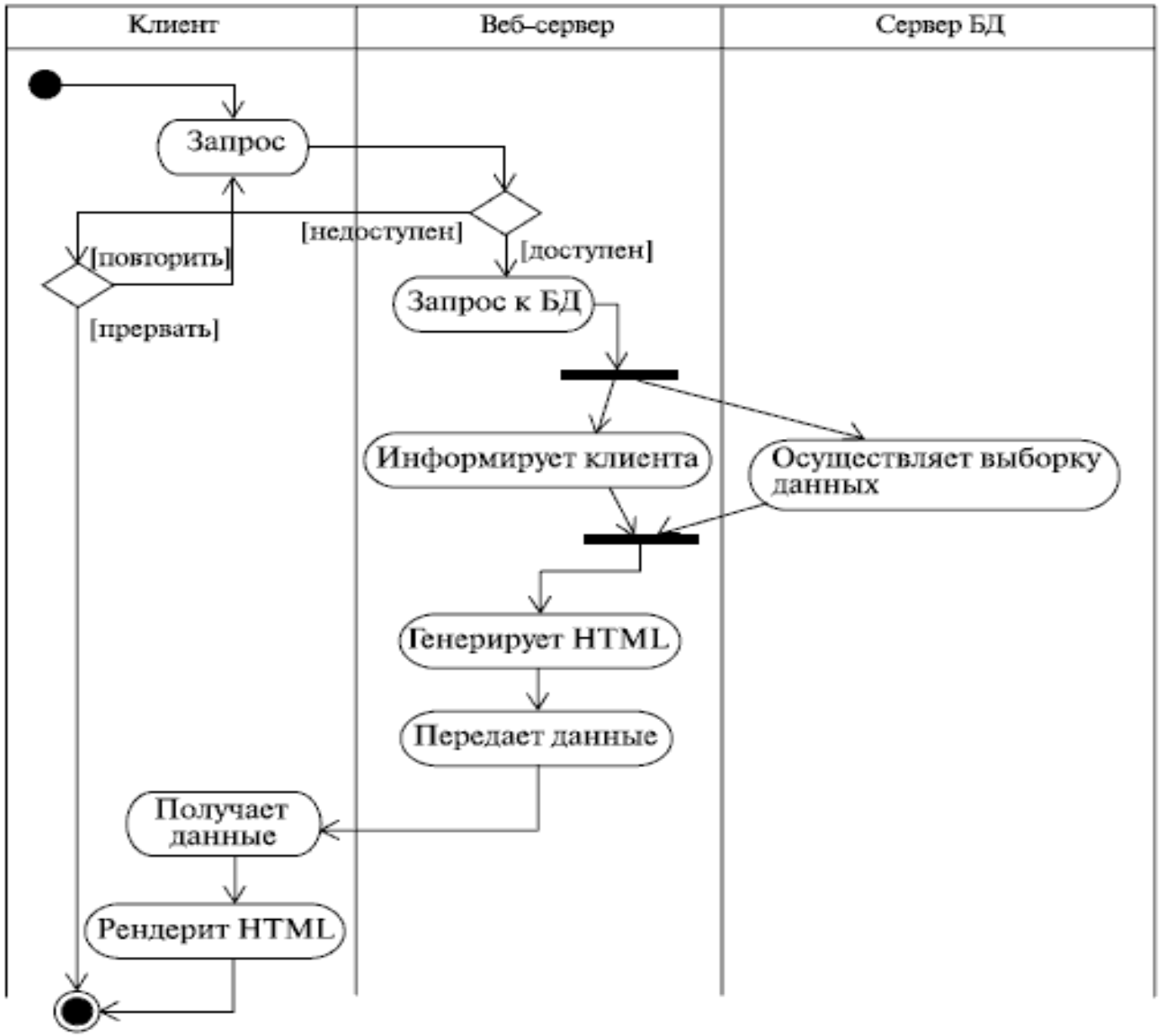
Диаграммы деятельностей

- Деятельность (activity)
- Действие (action)
- Переходы (transitions)
- Ветвления (decisions)
- Развилки и слияния (fork & join)
- Дорожки (swim lanes)
- Траектория объекта (object flow)
- Посылка и прием сигналов
- Прерывания и исключения



Отображение *распараллеливания*, а затем слияние воедино (*синхронизация*) потоков управления, т.е. *операции* выполняются *одновременно*. *Нотация* проста: несколько потоков управления сливаются в один или один *поток* разделяется на несколько.

Диаграммы деятельностей



Применение диаграмм деятельностей

«Создавая диаграммы деятельности, не забывайте, что они моделируют лишь срез некоторых динамических аспектов поведения системы.

С помощью единственной диаграммы деятельности никогда не удастся охватить все динамические аспекты системы.

Вместо этого следует использовать разные диаграммы деятельности для моделирования динамики рабочих процессов и отдельных операций»

Грэди Буч /разработчик UML/

Применение диаграмм деятельностей

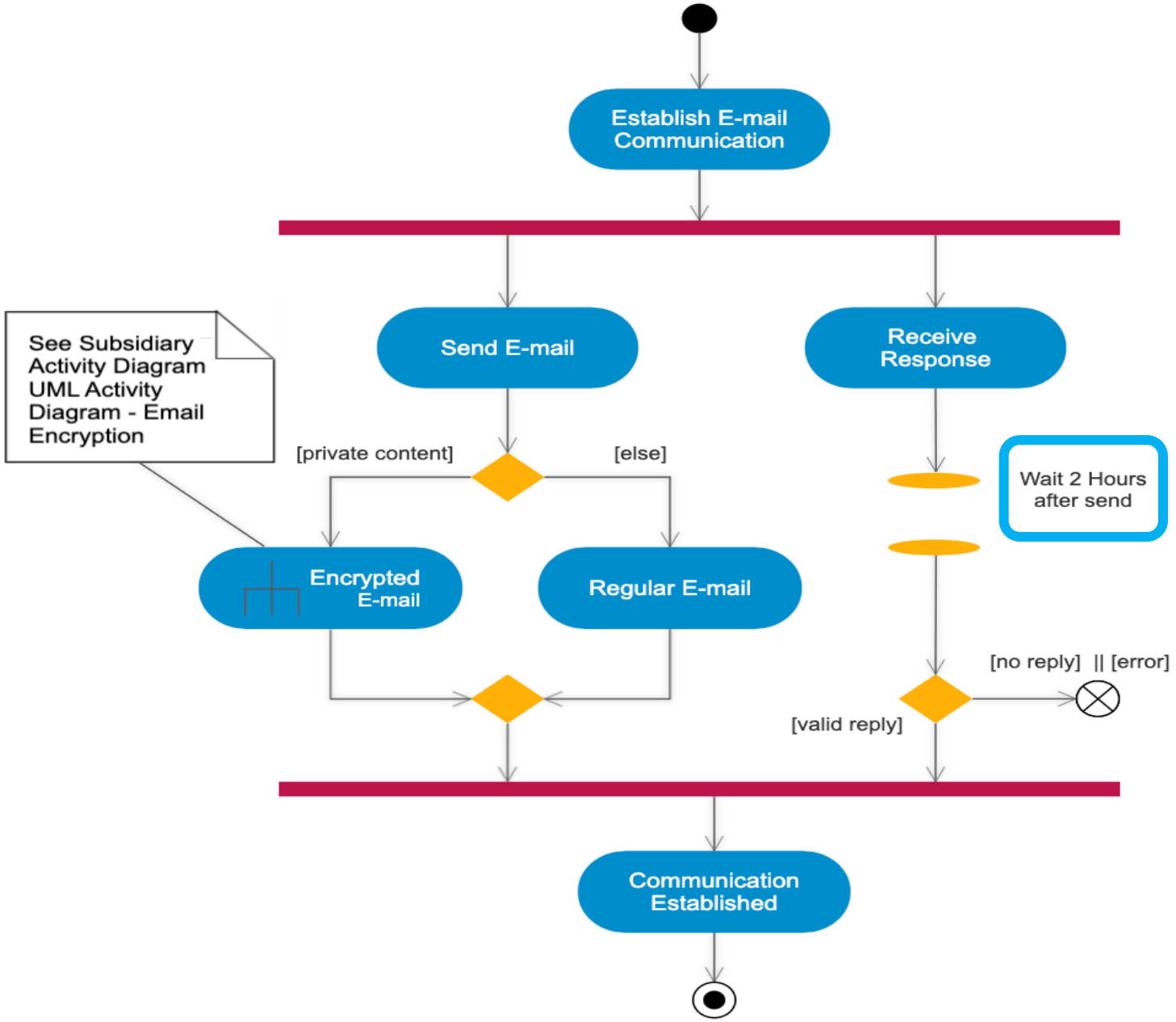
Для моделирования процессов

- Важны для деятельности с точки зрения акторов, работающих с системой, часто при описании бизнес-процессов. В случае такого использования диаграмм деятельности активно используются траектории объектов.

Для моделирования операций

- Диаграммы деятельности играют роль "продвинутых" блок-схем и применяются для *подробного* моделирования вычислений.
- При таком использовании важны конструкции принятия решения и разделения/слияния потоков управления (синхронизации).

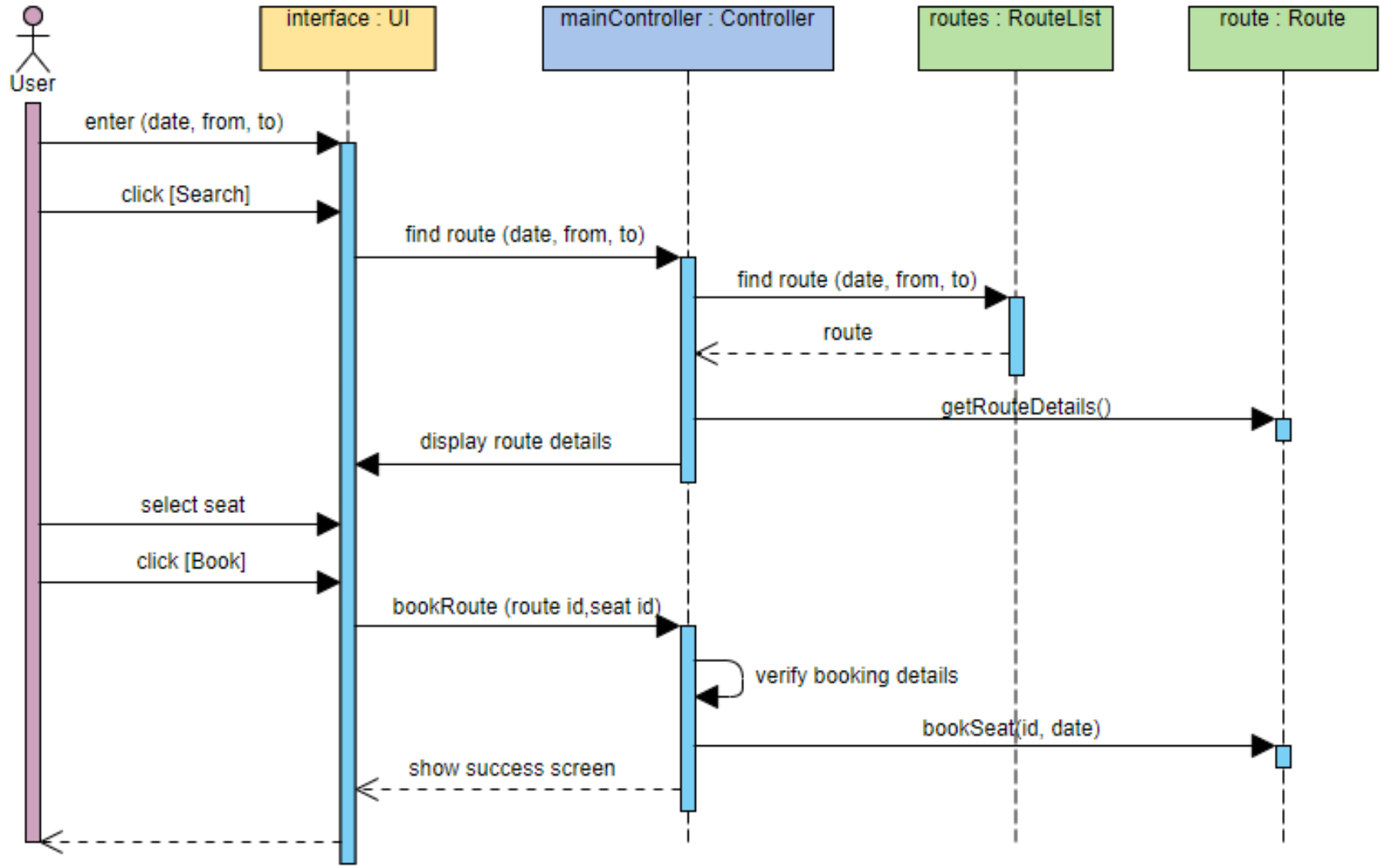
Диаграммы деятельности



Применение диаграмм деятельностей

- ✓ Диаграммы деятельностей являются вариантом диаграммы состояний.
- ✓ Диаграммы деятельности могут отображать одновременно выполняемые действия.
- ✓ На диаграммах деятельности можно использовать дорожки, распределяющие деятельности в соответствии с ролями (объектами), их выполняющими.
- ✓ Траектория объекта позволяет показать объекты, относящиеся к деятельности, и моменты переходов этих объектов из одного состояния в другое.
- ✓ Сложные деятельности можно дополнительно детализировать, разбив на действия и изобразив "диаграмму в диаграмме".
- ✓ Диаграммы деятельностей можно использовать для проектирования процессов (бизнес-процессов) или операций (действий или вычислений).

Диаграмма последовательности



Применение диаграмм взаимодействия

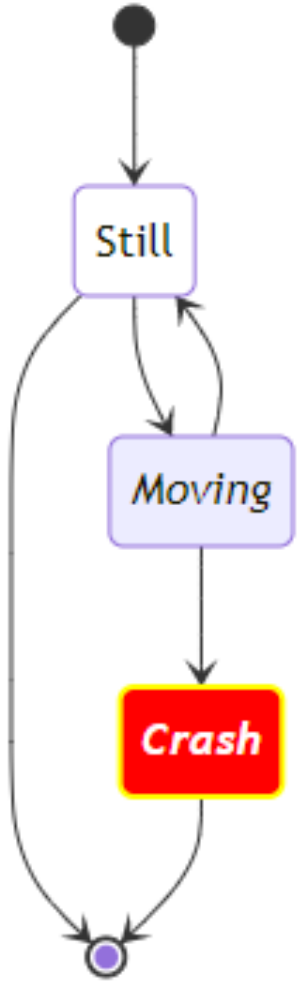
- ✓ Временная диаграмма — UML-диаграмма взаимодействия, которая фокусируется на процессах, происходящих в течение определенного периода времени.
- ✓ Это особый случай диаграммы последовательности, в которой время увеличивается слева направо.

Применение метода «конечных автоматов»

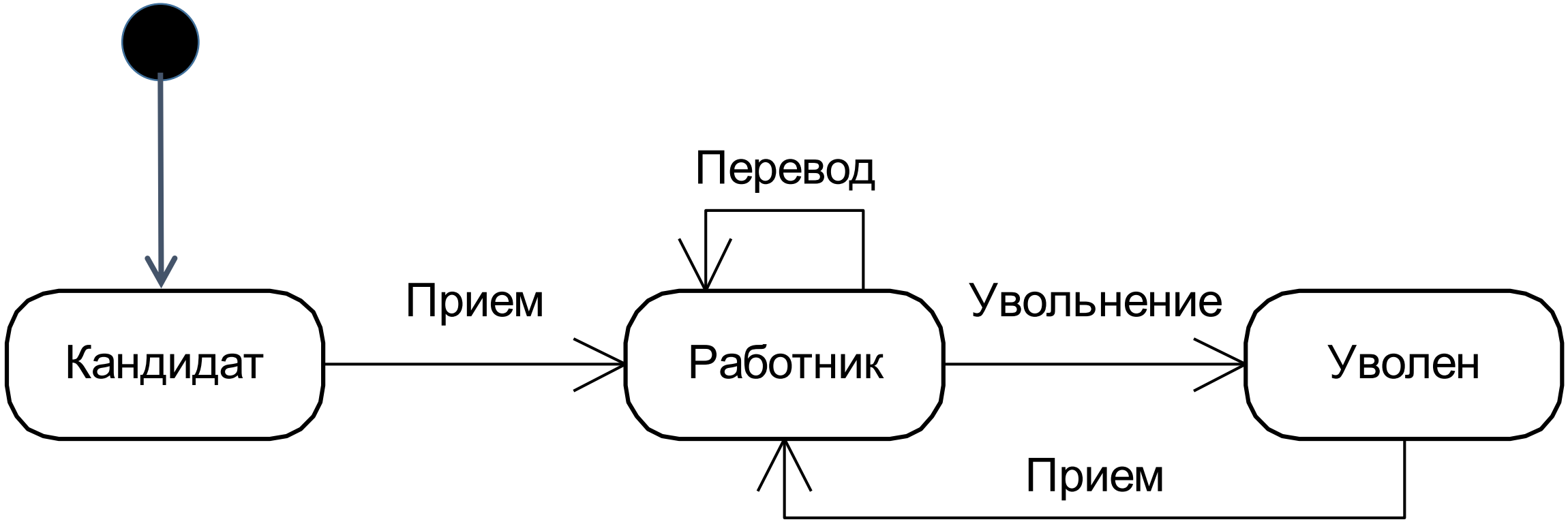
- Трансляция / интерпретация // компиляторы
- Пользовательский интерфейс
- Программы, управляемые событиями
- Моделирование жизненного цикла объекта
  - Если поведение зависит от прошлого
  - Если есть асинхронные стимулы

Применение диаграммы состояний

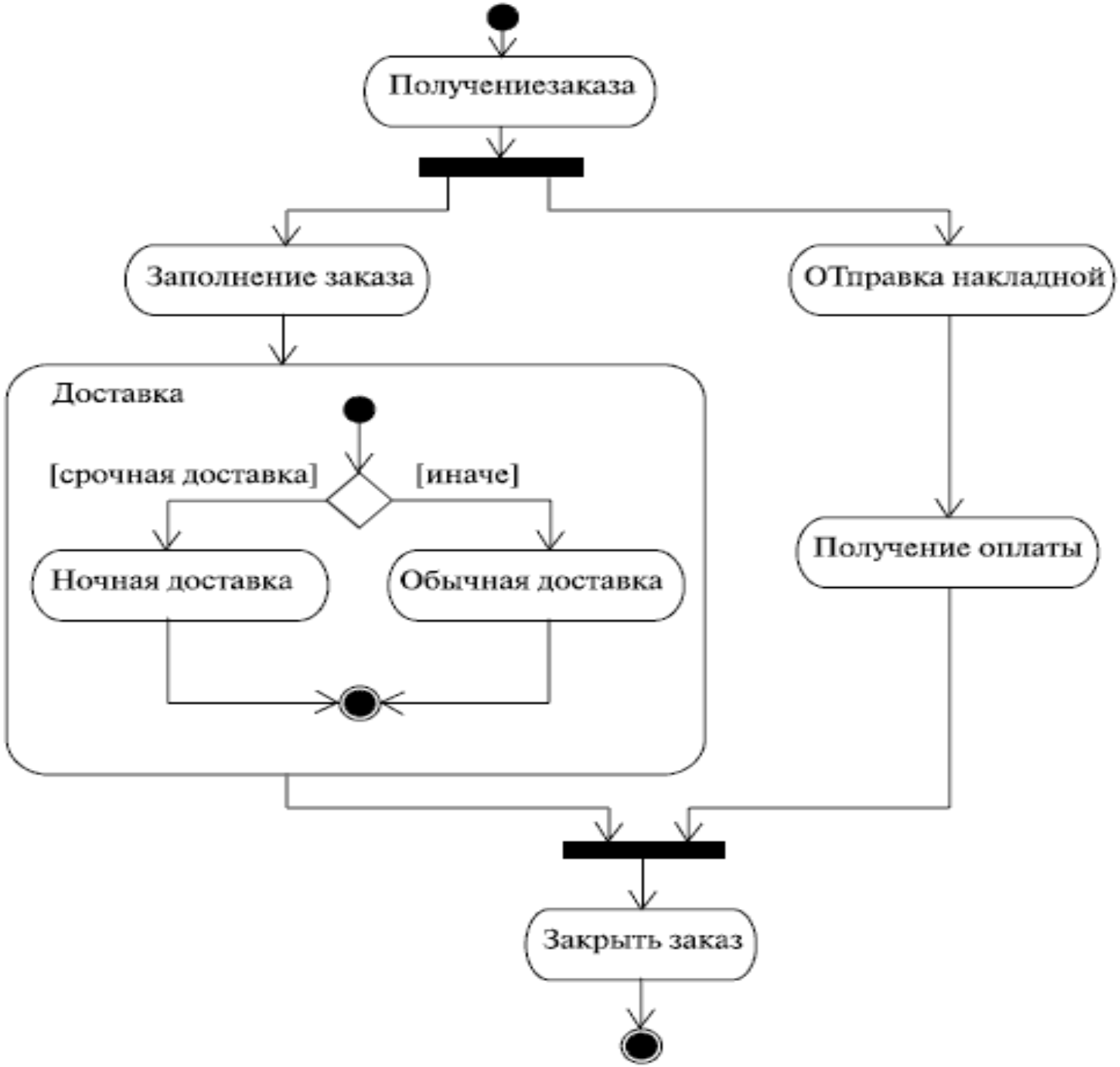
- Состояния (state)
  - Простые (simple)
  - Составные (composite)
  - Специальные (pseudo)
- Переходы (transition)
  - Начало (source)
  - Событие (event)
  - Условие (guard condition)
  - Действие (action)
  - Конец (target)



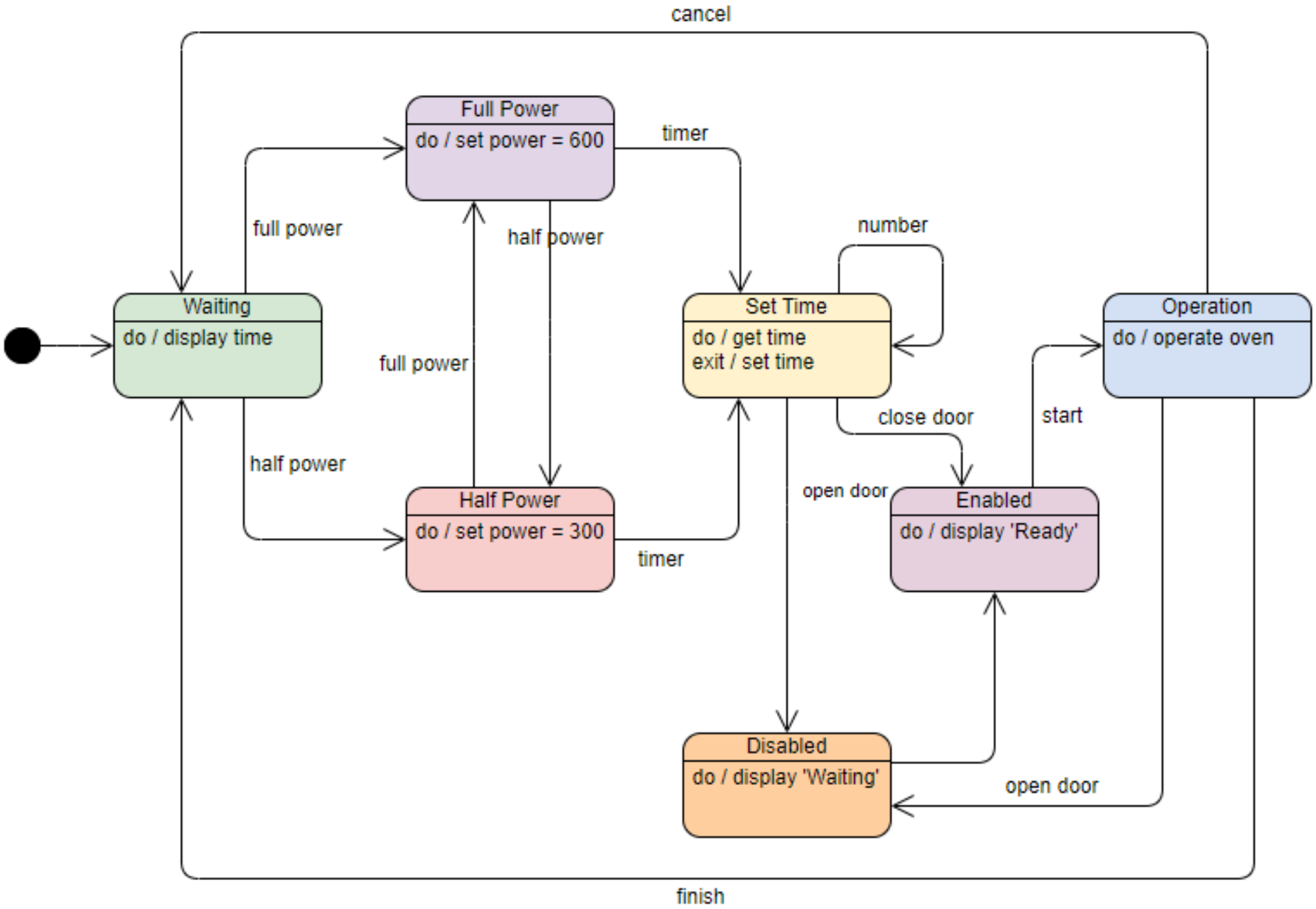
Пример диаграммы состояний сотрудника



Пример диаграммы состояний заказа



Пример диаграммы состояний СВЧ-печки



Сложные диаграммы состояний

Простое состояние — это состояние, не имеющее подструктуры.

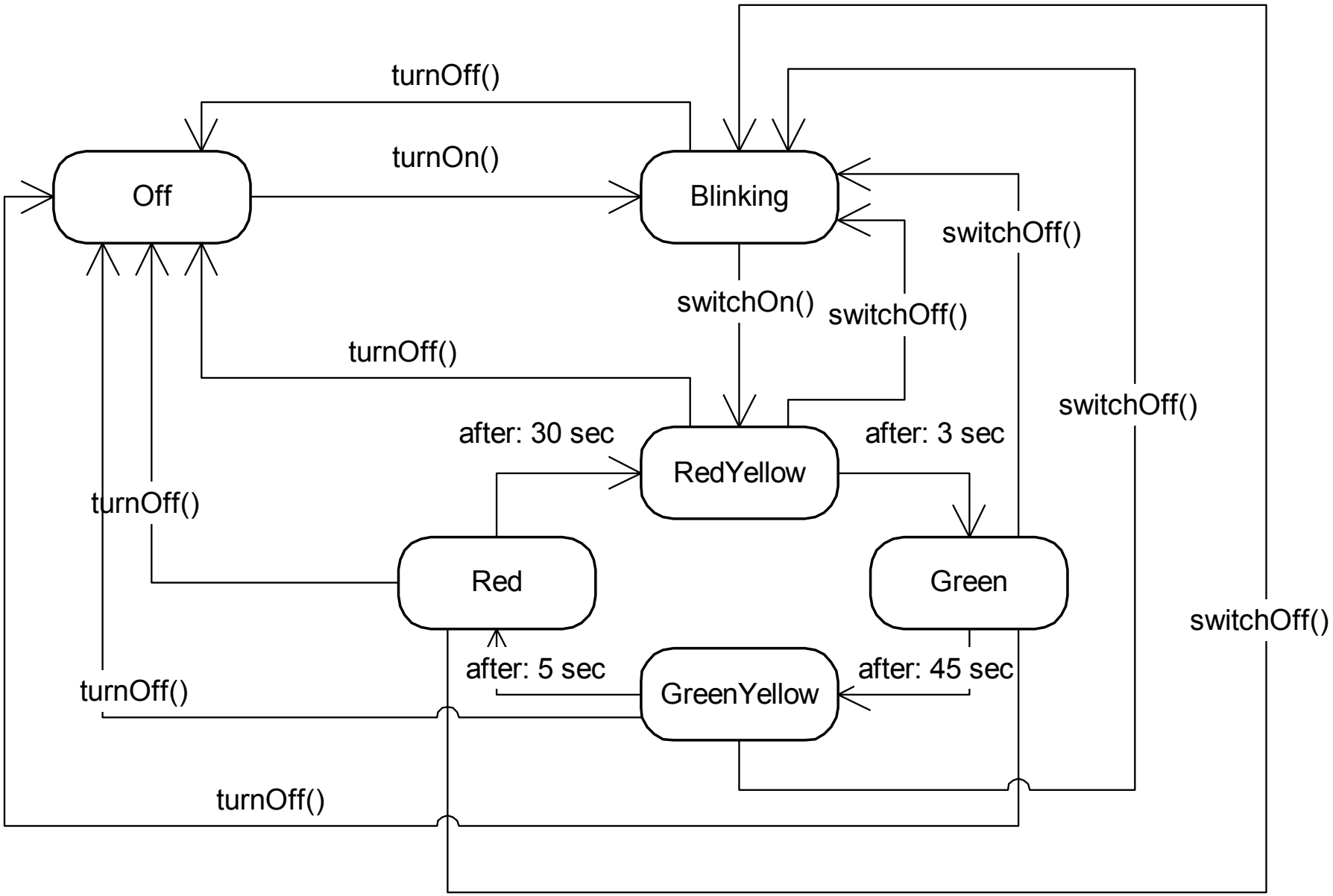
Состояние, которое имеет под-состояния (вложенные состояния), называется составным состоянием.

Под-состояния могут быть вложены на любом уровне. Вложенный конечный автомат может иметь не более одного начального и одного конечного состояния.

Под-состояния используются для упрощения сложных плоских автоматов состояний, показывая, что некоторые состояния возможны только в определенном контексте (окружающее состояние).

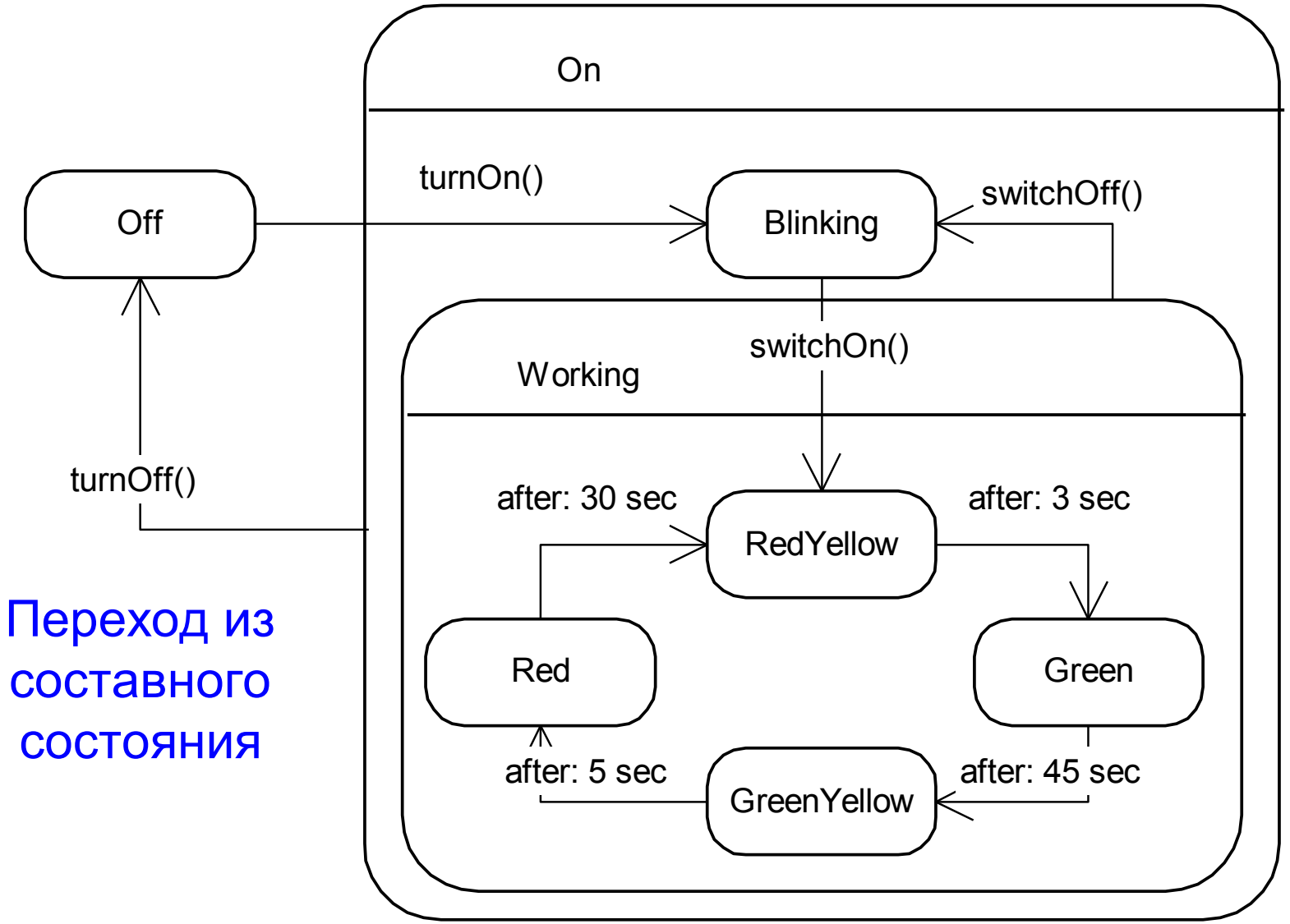
Связанные состояния могут быть сгруппированы в одно составное состояние. Вложение состояний внутрь других необходимо, если действие включает параллельные под-действия.

Пример диаграммы состояний светофора (1)



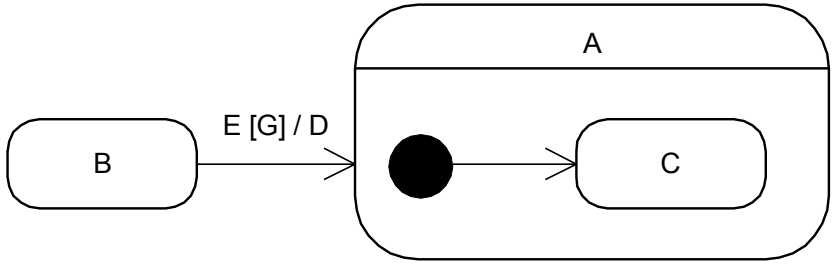
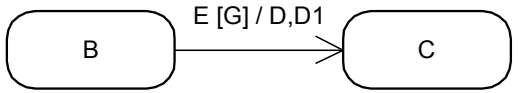
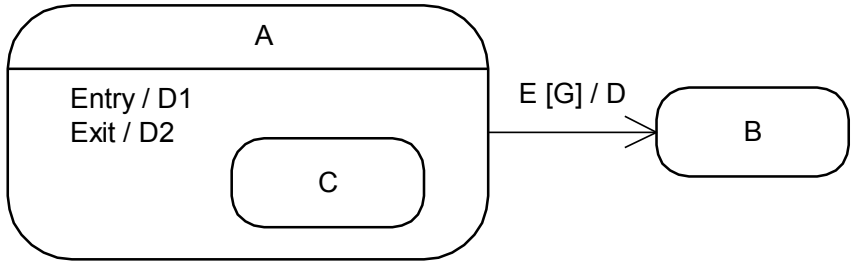
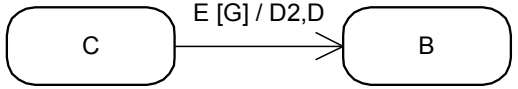
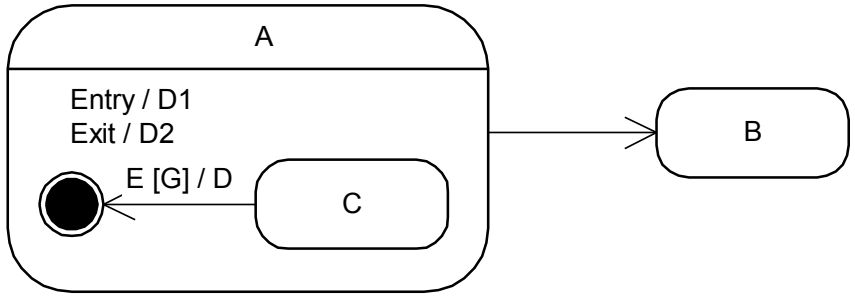
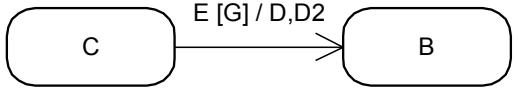
Пример диаграммы состояний светофора (2)

Составное состояние

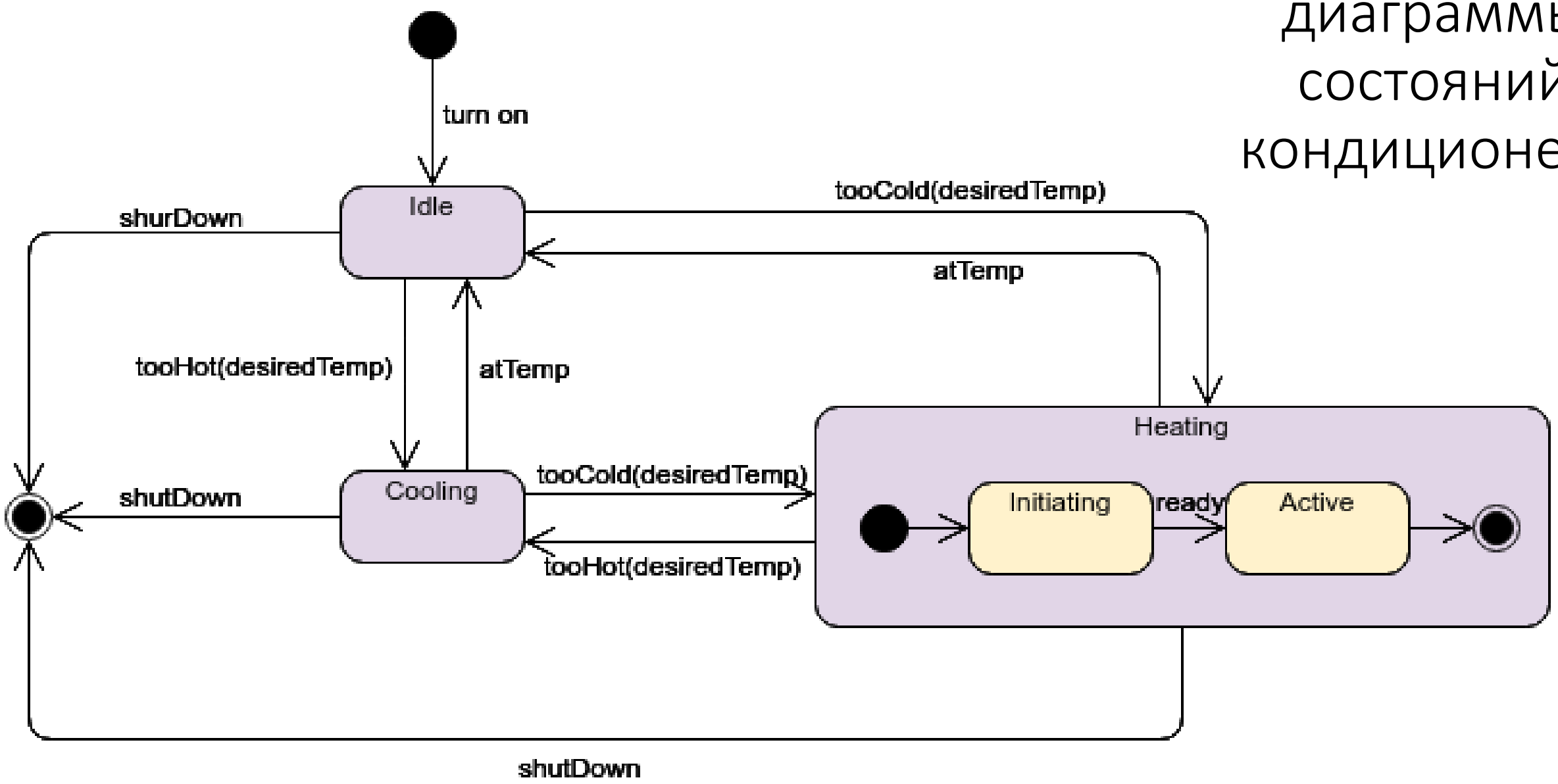


Переход из составного состояния

Переходы между составными состояниями

Вид перехода	Диаграмма перехода	Эквивалентная диаграмма
Переход в составное состояние		
Переход из составного состояния		
Переход по завершении		

Пример диаграммы состояний кондиционера



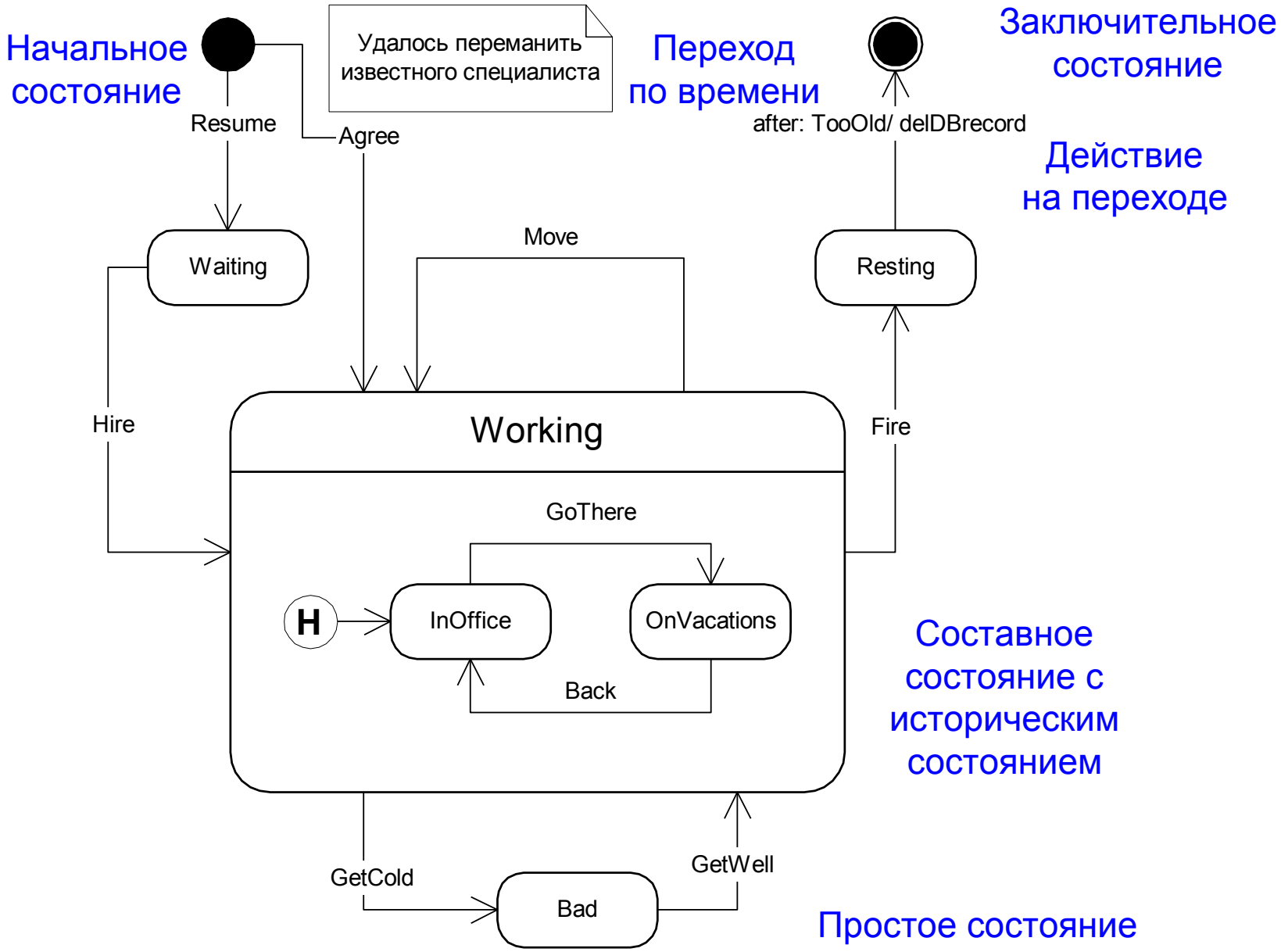
Сложные диаграммы состояний

Состояния на диаграммах конечного автомата могут быть историческими (History).

По умолчанию когда переход входит в составное состояние, действие вложенного конечного автомата начинается снова с начального состояния (если переход не нацелен непосредственно на под-состояние).

Историческое состояние позволяет автомату повторно войти в последнее под-состояние, которое было активным перед выходом из составного состояния.

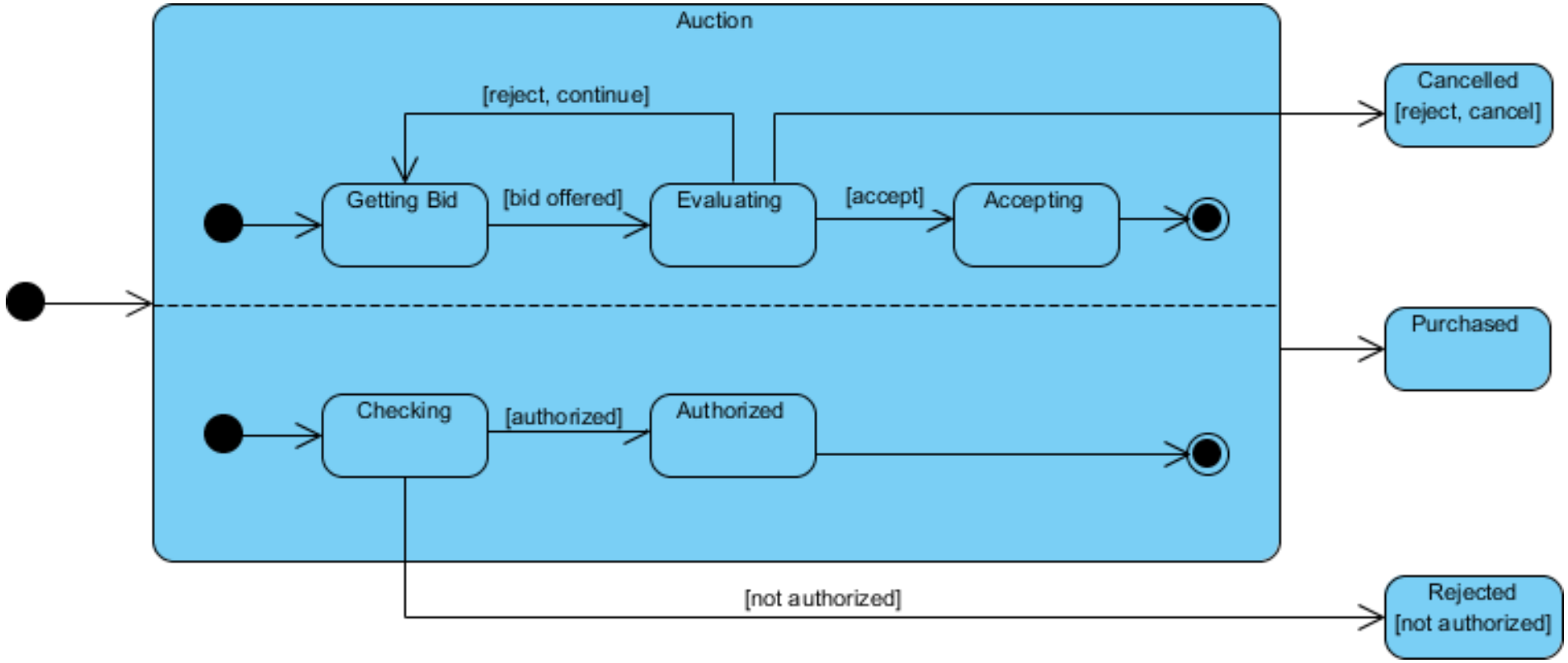
Пример диаграммы состояний сотрудника (2)



Составные состояния

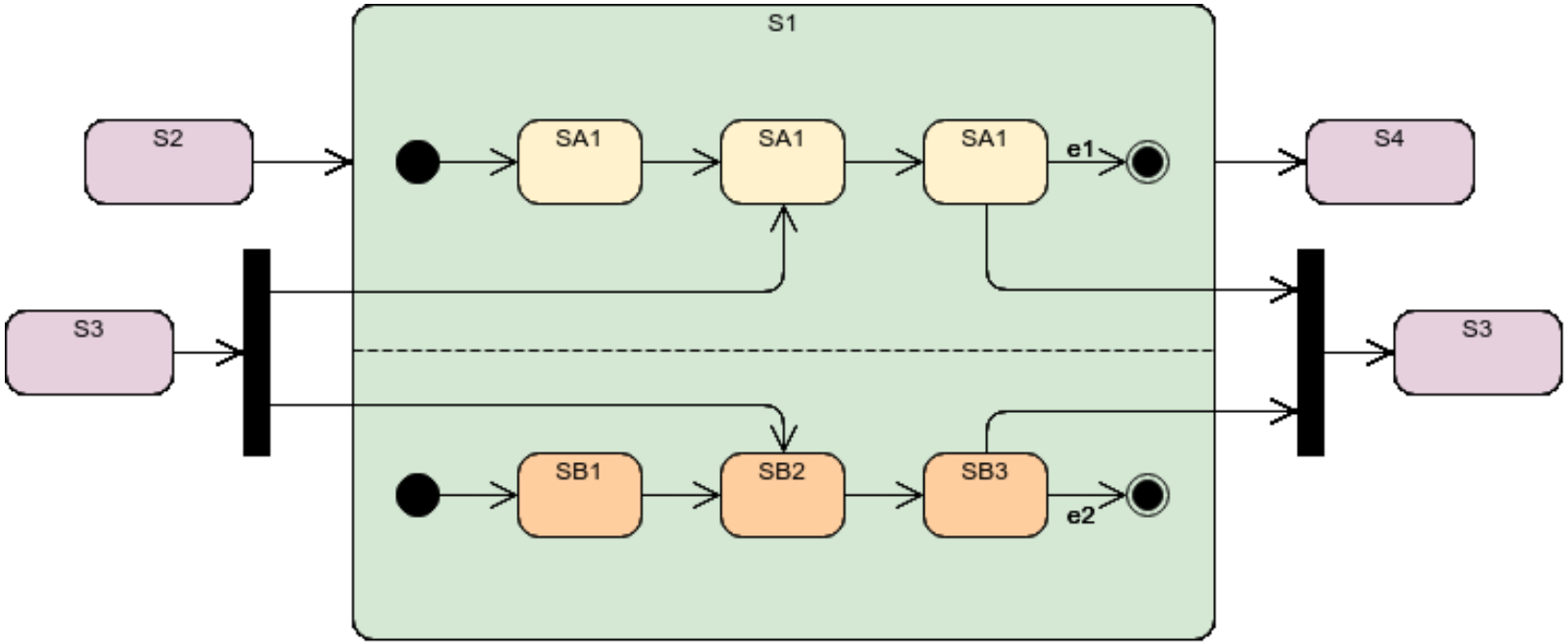
Вложение состояний внутрь других необходимо, когда действие включает **параллельные** под-действия.

Пример диаграммы моделирует аукцион с 2 одновременными под-состояниями: обработка заявки и авторизация лимита платежа. Конечный автомат реализует в начале **разветвление** на 2 отдельных потока. Каждое под-состояние имеет состояние выхода, обозначающее конец потока. Выход из составного состояния происходит, когда оба под-состояния вышли (завершились) .



Ортогональные состояния

Анализ иерархической декомпозиции состояний может включать применение операции «исключающее ИЛИ» к любому заданному состоянию. Например, если система находится в сверхсостоянии «Включено», она также находится либо в подсостоянии «операнд1» ИЛИ в подсостоянии «операнд2» ИЛИ в подсостоянии «opEntered» ИЛИ в подсостоянии «результат».
Диаграммы состояний UML содержат дополнительное И-разложение. Такое разложение означает, что составное состояние может содержать две или более ортогональные области (здесь «ортогональные» означает совместимые и независимые). Нахождение в таком составном состоянии означает пребывание во всех его ортогональных областях одновременно*!*



Ортогональные состояния

Замечание: если ортогональные области полностью независимы друг от друга, их совокупная сложность просто **аддитивна**. Это означает, что количество независимых состояний, необходимых для моделирования системы, представляет собой просто сумму $k + l + m + ...$

где $k, l, m, ...$ обозначают количество ИЛИ-состояний в каждой ортогональной области.

Но в общем случае взаимной зависимости это приводит к мультипликативной сложности, т.е. в общем случае необходимое количество состояний будет равно **произведению** $k \times l \times m \times ....$

Несмотря на то, что ортогональные регионы предполагают независимость выполнения (обеспечивая параллелизм), спецификация UML не требует, чтобы каждому ортогональному региону был назначен отдельный поток выполнения.

Фактически, ортогональные области выполняются в одном потоке.

Спецификация UML требует, чтобы разработчик не полагался на какой-либо конкретный порядок отправки экземпляров событий в соответствующие ортогональные области.

События

События:

- Внешние – передаются между системой и действующими лицами
- Внутренние – передаются между объектами внутри системы

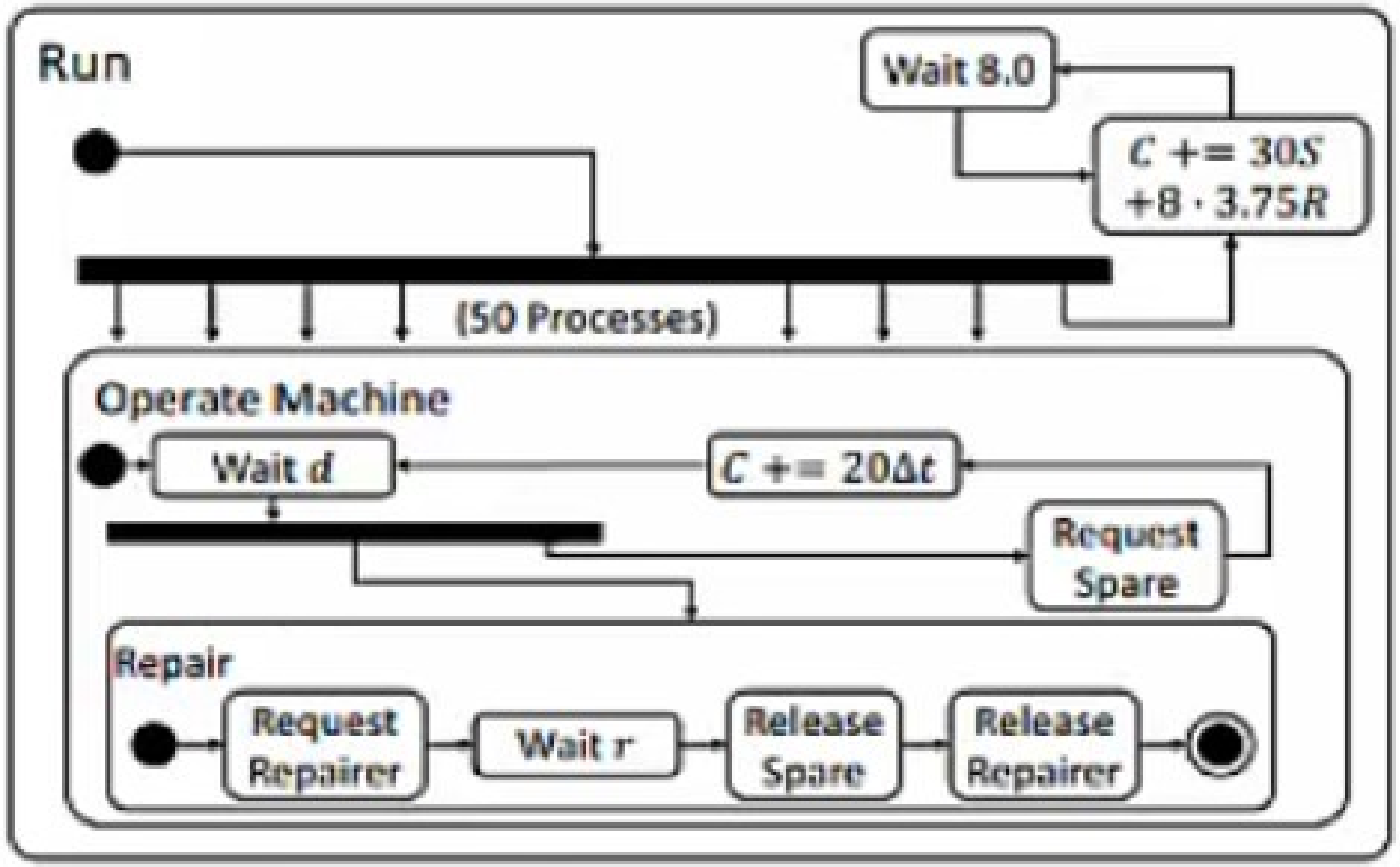
Типы событий UML:

- Вызов (Call event)
- Изменение (Change event)
- Таймер (Time event)
- Сигнал (Signal event)

Диаграммы деятельности

Спецификация для производственной системы

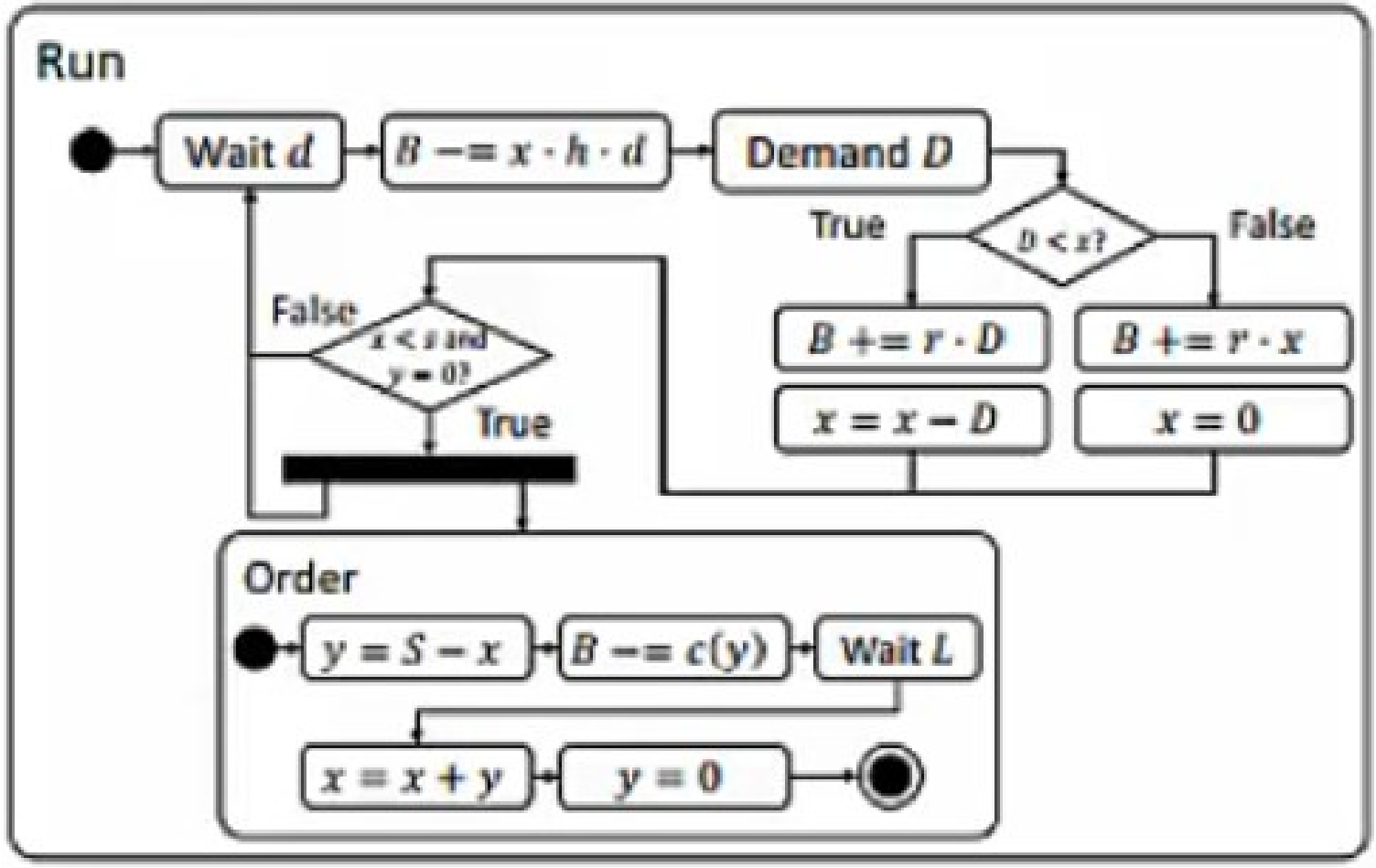
N = 50 станков работают 8 час/день
и 5 дней в неделю
Поломки случайны в интервале
от 132 до 182 ч (_d_)
Если есть запчасти ремонт занимает
от 4 до 10 ч (_r_)
Ремонтник (_R_) получает 3,75 уе в час
Можно купить S запчастей
на 30 уе в день
Потери от простоя станка при
отсутствии запчастей равен 20 уе в час



Диаграммы деятельности

Учет товаров на складе

Цена продажи ед.товара *r=100*
Период прихода клиентов *d =(expovar(la=5))*
Клиент покупает *D=(uniform(1,4))* штук
Дозаказ на склад:
если запас *x< s,* то дозаказ *y=S-x*
Стоимость *c(y) =50*y* при заказе *y*
Время поставки *L=2* дня
Цена хранения *h=2* за штуку в день



Диаграммы состояний

Диаграмма состояний UML предлагает специальный механизм для ***отсрочки*** событий в состояниях (Event deferral).

В каждом состоянии можно включить пункт [список событий]/ defer

Если происходит событие в списке отложенных событий текущего состояния, событие будет сохранено (отложено) для будущей обработки до тех пор, пока не будет введено состояние, в котором это событие не указано в списке отложенных событий.

При входе в такое состояние конечный автомат UML автоматически вызывает любые сохранённые события, а затем либо использует, либо отбрасывает эти события.

В суперсостоянии может быть определён переход для события, которое отложено под-состоянием.

Под-состояние имеет приоритет над над-состоянием, т.е. событие будет отложено, а переход в суперсостояние не будет выполнен.

Mermaid - *Инструментарий для схем описаний*

https://mermaid.js.org/

Mermaid.js интегрирована в GitLab, GitHub.
Есть плагины для множества других сервисов.
Mermaid помогает быстрее и эффективнее оформлять схемы и графики для документации в виде описаний генерации диаграмм.

Блок-схемы состоят из узлов в виде геометрических фигур и стрелок, соединяющих узлы. Блок-схема создается с помощью ключевого слова **flowchart** и аббревиатуры для указания направления (*TD, LR*).
В узел можно поместить любой текст, если после имени указать текст в квадратных скобках.
Mermaid позволяет создавать динамические блок-схемы.

Mermaid - *Инструментарий для схем описаний*

Форма узла задается символами вокруг текста

```
flowchart LR
  node1[Форма 1]
  node2(Форма 2)
  node3([Форма 3])
  node4[[Форма 4]]
  node5[(Форма 5)]
  node6((Форма 6))
  node14(((форма 14)))
```

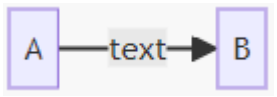
```
flowchart TD
  node7>Форма 7]
  node8{Форма 8}
  node9{{Форма 9}}
  node10[/Форма 10/]
  node11[\Форма 11\]
  node12[/Форма 12\]
  node13[\Форма 13/]
```

Mermaid - *Инструментарий для схем описаний*

Узлы в блок-схеме соединяются с помощью стрелок и линий.

A - -> B

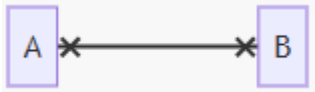
A - - text - -> B



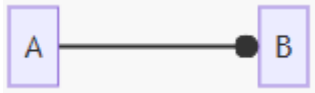
A < - -> B

A - - -x B

A x- - -x B



A - - - o B

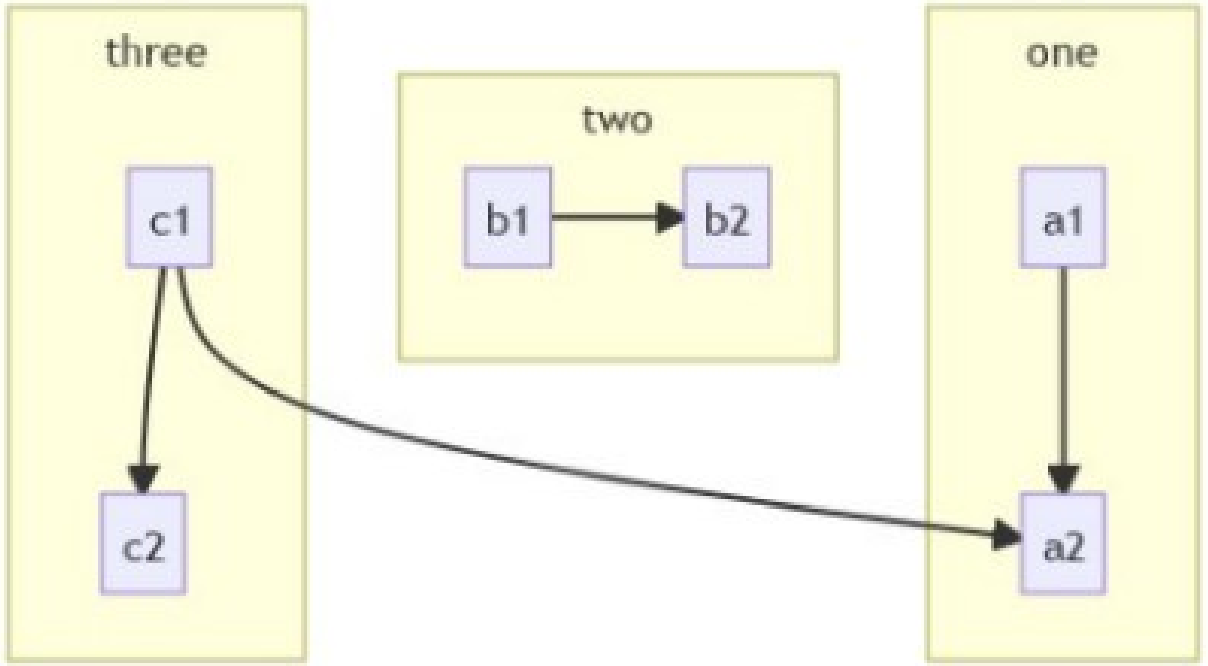


Можно размещать текст на стрелках и связях узлов.

	длина1	длина2
Нормальная	- - -	- - - -
Невидимая	~ ~ ~	~ ~ ~ ~
Нормальная стрелка	- - ->	- - - ->
Толстая	= = =	= = = =
Толстая стрелка	= = =>	= = = =>
Точечная	- . -	- . . . -
Точечная стрелка	- . ->	- . . . ->
2х-сторонняя толстая стрелка	<= = =>	<= = = =>

Mermaid - *Инструментарий для схем описаний*

Если на одной схеме необходимо показать взаимосвязанную работу двух алгоритмов, то для этих целей подойдут подсхемы, позволяющие визуально разделять несколько сущностей на одном рисунке.




```
flowchart TD
    c1--> a2
    subgraph one
      a1--> a2
    end
    subgraph two
      b1--> b2
    end
    subgraph three
      c1--> c2
    end
```

Mermaid - *Инструментарий для схем описаний*

Узлам можно задавать альтернативные цвета. При этом для каждого отдельно взятого узла можно задать свой цвет.
Для этого после описания блок-схемы пишется ключевое слово **style**, после указать имя узла, потом описать цветовую схему. Цветовая схема задается с помощью тегов:
fill - заливка; *stroke* - цвет границы; *stroke-width* - толщина границы;
color - цвет текста; *stroke-dasharray* - пунктирная граница.


```
flowchart LR
  id1(Start) - -> id2(Stop)
style id1 fill:green, stroke:#333, stroke-width:4px;
style id2 fill:red, stroke:#333, stroke-width:4px, color:#fff, stroke-dasharray: 12 5;
classDef class1 fill:green, stroke:#333, stroke-width:4px;
classDef class2 fill:red, stroke:#333, stroke-width:4px, color:#fff, stroke-dasharray: 12 5;
```

Mermaid - *для диаграммы классов*

UML-диаграммы используются для схематичного отображения классов и их связей в коде. Здесь можно создавать диаграммы классов с помощью ключевого слова classDiagram. Каждый экземпляр класса содержит в себе три составляющие: в верхней части располагается название класса и описание; в центральной части находятся атрибуты класса, которые указываются со строчной буквы и выравниваются по левому краю; в нижней части размещены методы класса.

```
classDiagram
class BankAccount
BankAccount : +String owner
BankAccount : +Bigdecimal balance
BankAccount : +deposit(amount)
BankAccount : +withdrawl(amount)
```

Или

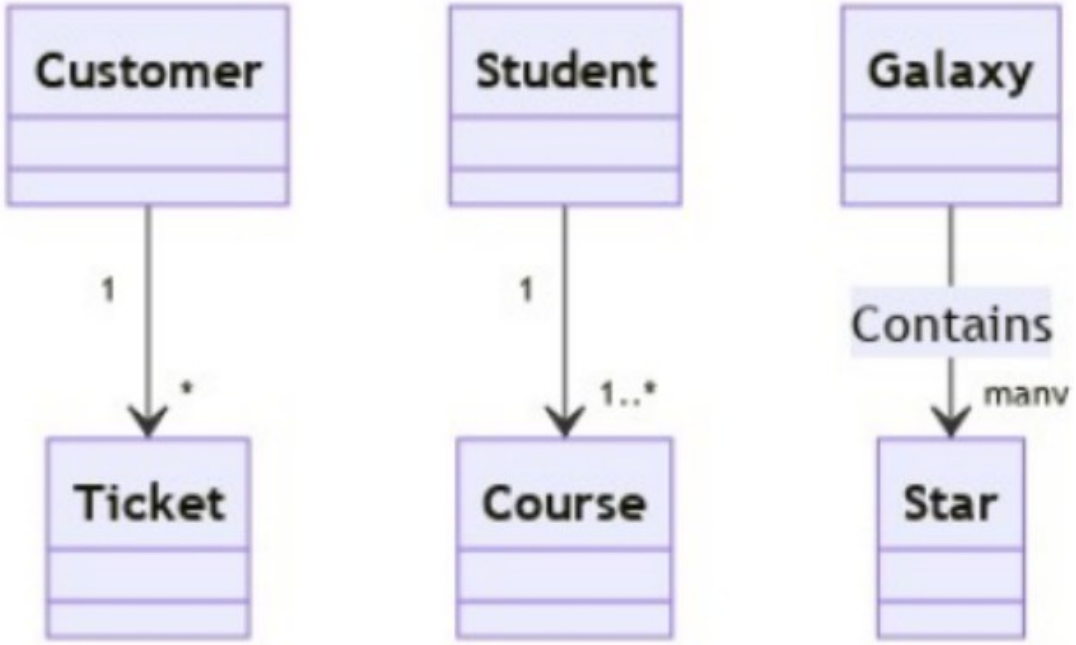
```
classDiagram
class BankAccount {
+String owner
+BigDecimal balance
+deposit(amount)
+withdrawl(amount)  }
```

Mermaid - *для диаграммы классов*

Отношения между классами задаются с помощью различных видов стрелок.
При желании можно явно указать вид связи в виде текста.
Двусторонние отношения задаются с помощью дублированной стрелки, направленной в обе стороны. Двусторонние отношения возможны для следующих видов взаимоотношений:

<|Наследование * Композиция o Агрегация
> Ассоциация < Ассоциация |> Реализация

```
classDiagram
 Customer "1" --> "*" Ticket
 Student "1" --> "1..*" Course
 Galaxy --> "many" Star: Contains
```

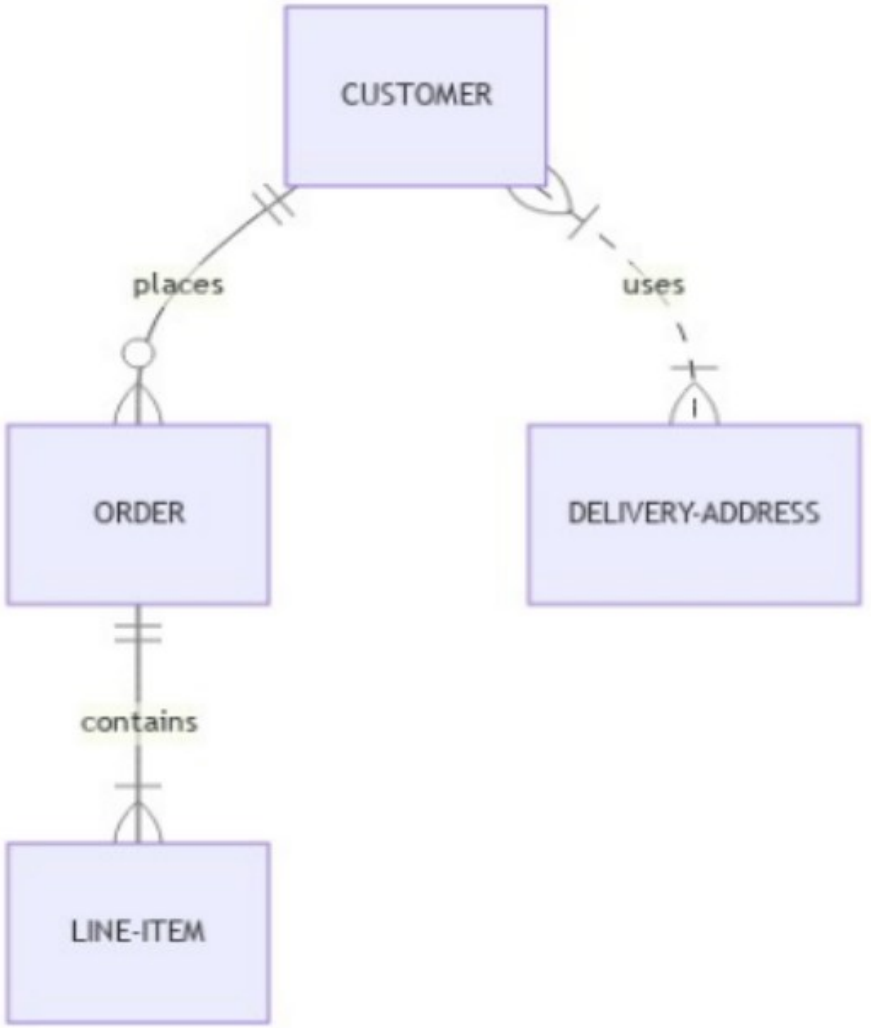


Mermaid - *для модели данных*

Модель данных, описывающая взаимосвязанные вещи, представляющие интерес в определенной области знаний, используется при высокоуровневом проектировании баз данных. Задается с помощью ключевого слова

erDiagram

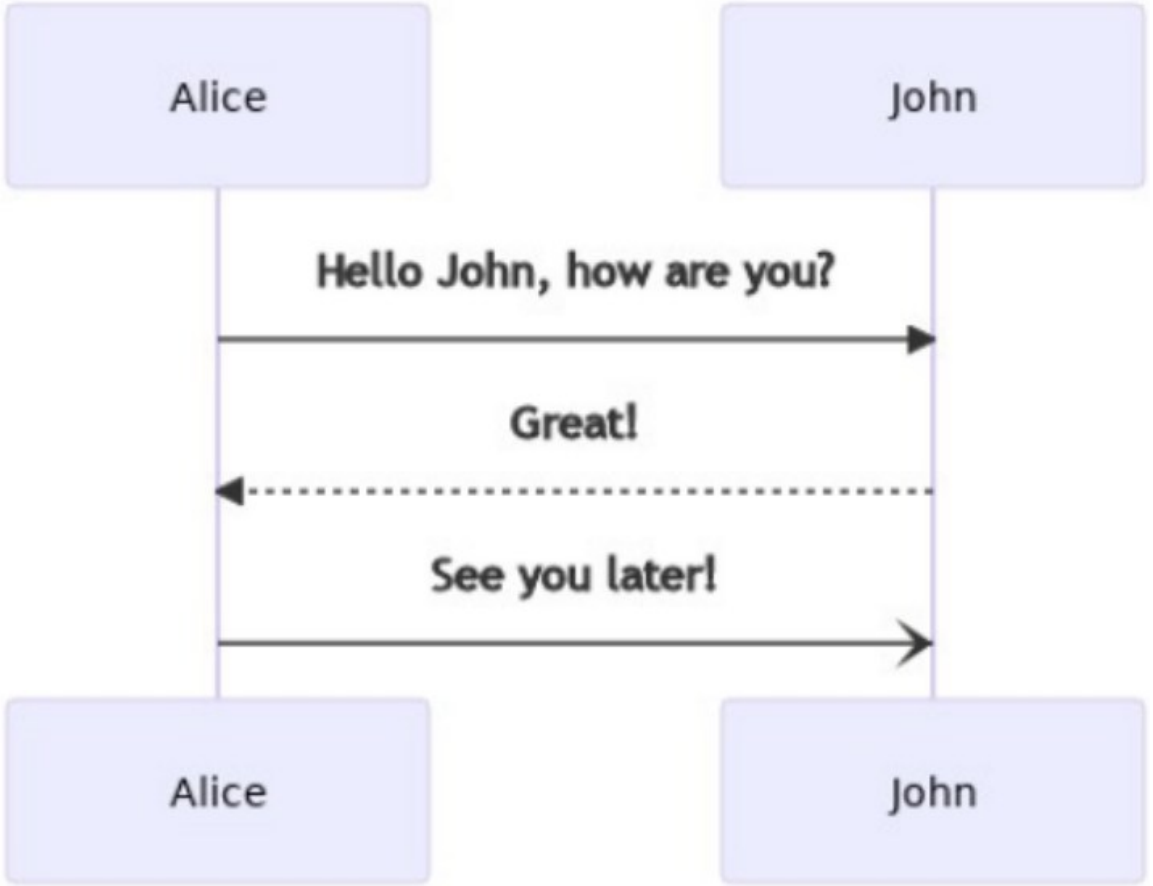
```
erDiagram
 CUSTOMER ||--o{ ORDER : places
 ORDER ||--|{ LINE-ITEM : contains
 CUSTOMER }|..|{ DELIVERY-ADDRESS : uses
```



Mermaid - *для диаграмм последовательности*

На диаграммах последовательности отображается жизненный цикл одного объекта или нескольких, а также их взаимодействие. Диаграммы последовательности состоят из *акторов* в виде прямоугольников и взаимодействий в виде вертикальных «линий жизни». В Mermaid диаграммы последовательности задаются с помощью ключевого слова sequenceDiagram

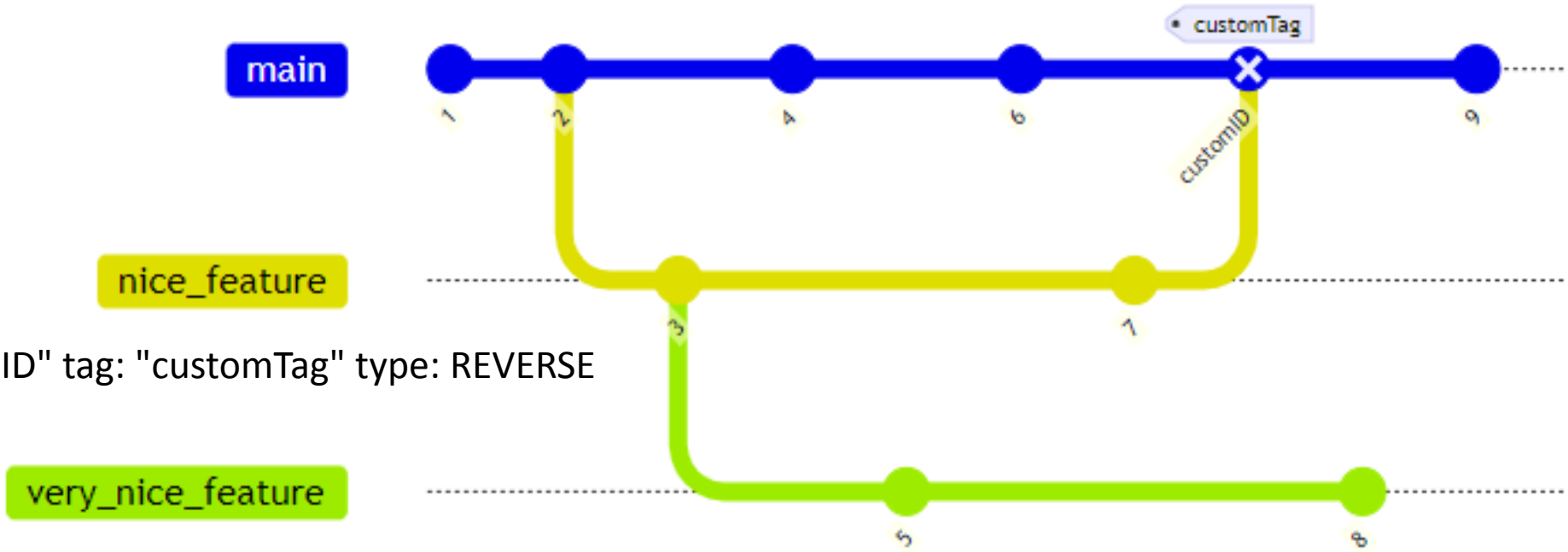
```
sequenceDiagram
 Alice->>John: Hello John, how are you?
 John-->>Alice: Great!
 Alice-)John: See you later!
```



Mermaid - *для схем git-потоков*

```
gitGraph LR:
 commit id: "1"
 commit id: "2"
 branch nice_feature
 checkout nice_feature
 commit id: "3"
 checkout main
 commit id: "4"
 checkout nice_feature
 branch very_nice_feature
 checkout very_nice_feature
 commit id: "5"
 checkout main
 commit id: "6"
 checkout nice_feature
 commit id: "7"
 checkout main
 merge nice_feature id: "customID" tag: "customTag" type: REVERSE
 checkout very_nice_feature
 commit id: "8"
 checkout main
 commit id: "9"
```

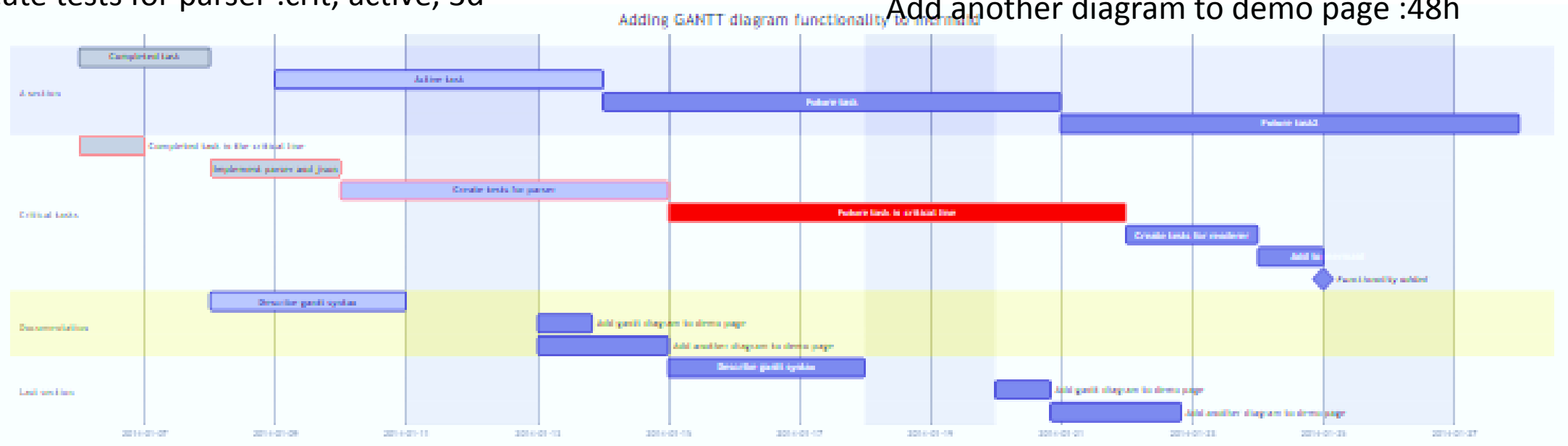
Mermaid может отображать диаграммы Git. Подобные диаграммы полезны командам разработчиков, поскольку они позволяют поделиться своими вариантами ветвления Git. Это упрощает визуализацию того, как работает поток git.



Mermaid - *для сетевых графиков*

gantt

```
dateFormat YYYY-MM-DD
title Adding GANTT diagram functionality
section A section
Completed task :done, des1, 2022-01-06,2022-01-08
Active task :active, des2, 2022-01-09, 3d
Future task : des3, after des2, 5d
Future task2 : des4, after des3, 5d
section Critical tasks
Completed task in the critical line :crit, done, 2022-01-06,24h
Implement parser and jison :crit, done, after des1, 2d
Create tests for parser :crit, active, 3d
Future task in critical line :crit, 5d
Create tests for renderer :2d
Add to mermaid :until isadded
Functionality added :milestone, isadded, 2022-01-25, 0d
section Documentation
Describe gantt syntax :active, a1, after des1, 3d
Add gantt diagram to demo page :after a1 , 20h
Add another diagram to demo page :doc1, after a1 , 48h
section Last section
Describe gantt syntax :after doc1, 3d
Add gantt diagram to demo page :20h
Add another diagram to demo page :48h
```



Mermaid - *для интерактивных схем*

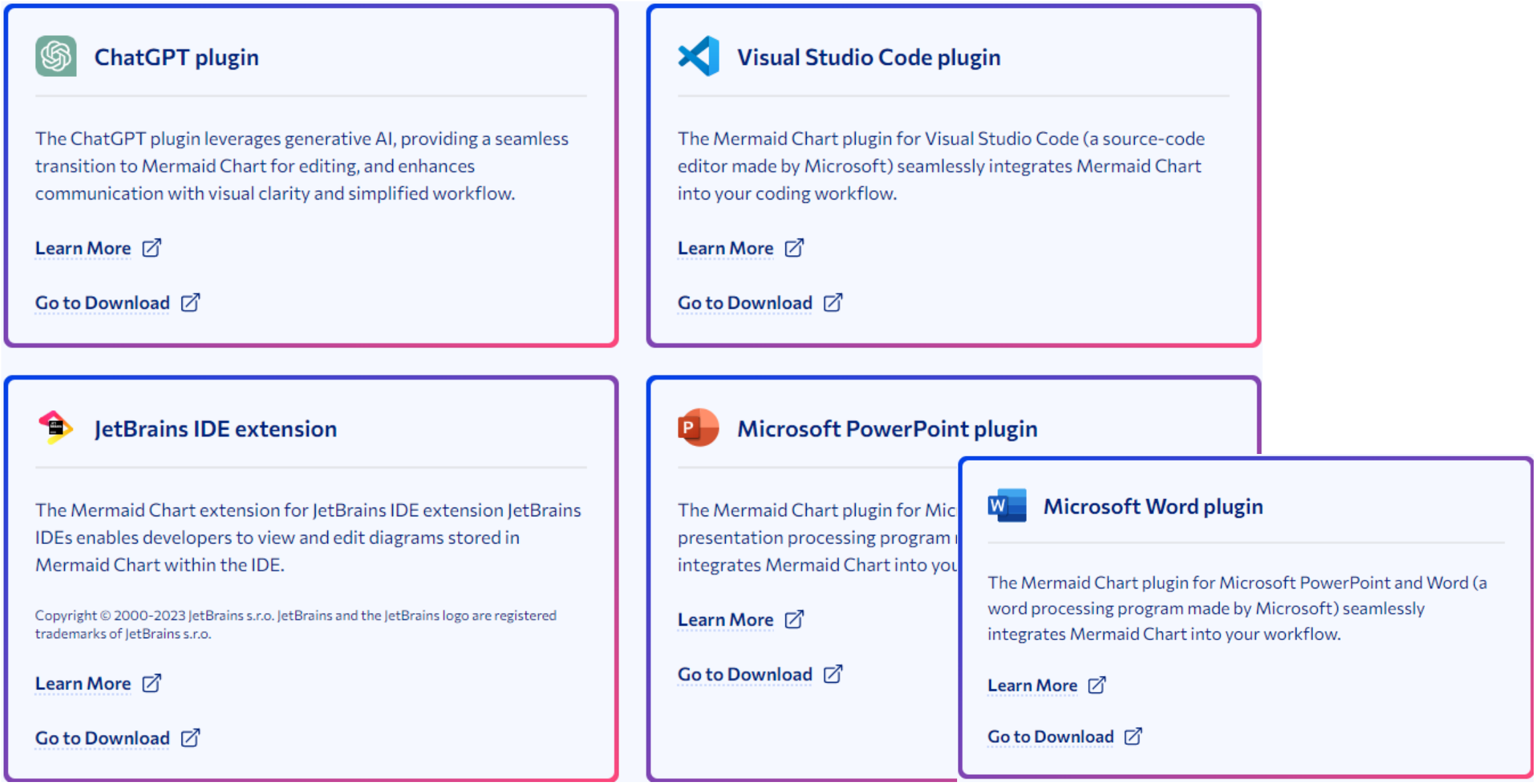
Узел можно сделать кнопкой, открывающей страницу в браузере. Для этого надо указать ссылку в описании узла. Ссылку необходимо экранировать кавычками.

```
flowchart TD
%% создаем узел
   A --> B
%% Комментарии задаются двумя знаками процента
%% Описываем действия для клика по узлу
click A "https://www.bmstu.ru" _blank
```

Можно настроить инициализацию и умолчания:

```
%%{init: { 'logLevel': 'debug', 'theme': 'default',
'themeVariables': { 'commitLabelColor': '#ff0000',
'commitLabelBackground': '#00ff00' } } }%%
```

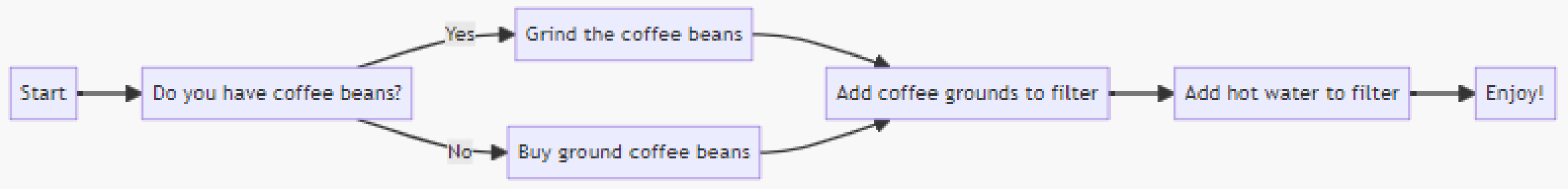
Mermaid – плагины



Mermaid - графы

Процесс приготовления кофе:

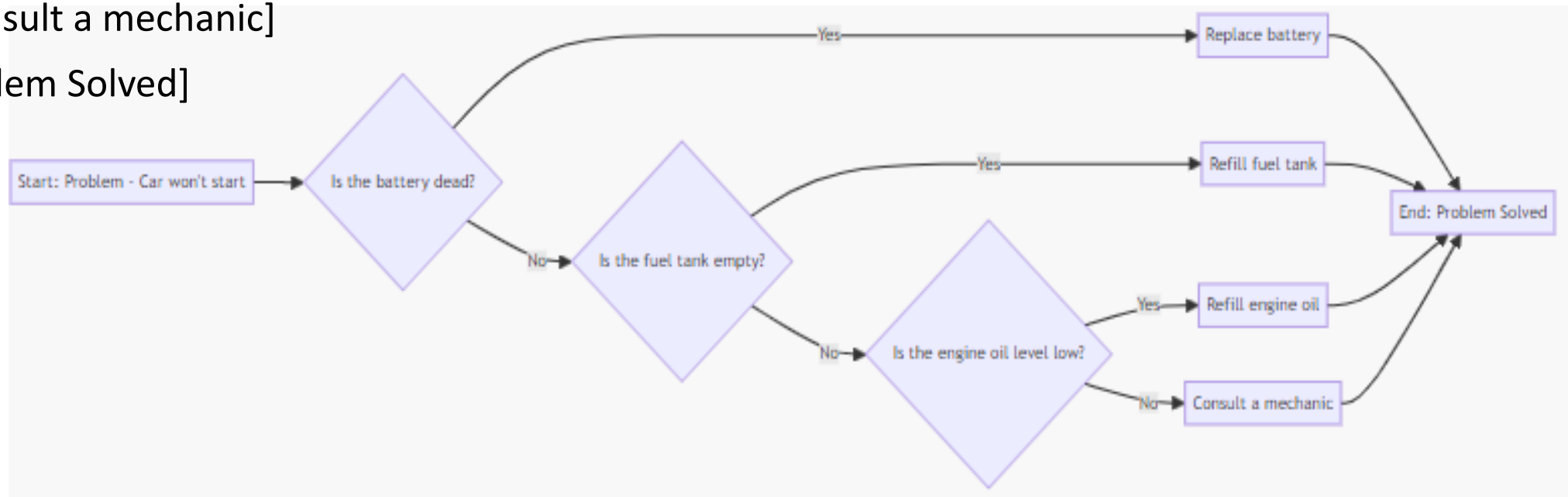
```
flowchart LR
    A[Start]-->B[Do you have coffee beans?]
    B-->|Yes|C[Grind the coffee beans]
    B-->|No|D[Buy ground coffee beans]
    C-->E[Add coffee grounds to filter]
    D-->E
    E-->F[Add hot water to filter]
    F-->G[Enjoy!]
```



Mermaid - графы

Процесс поиска неисправности автомобиля

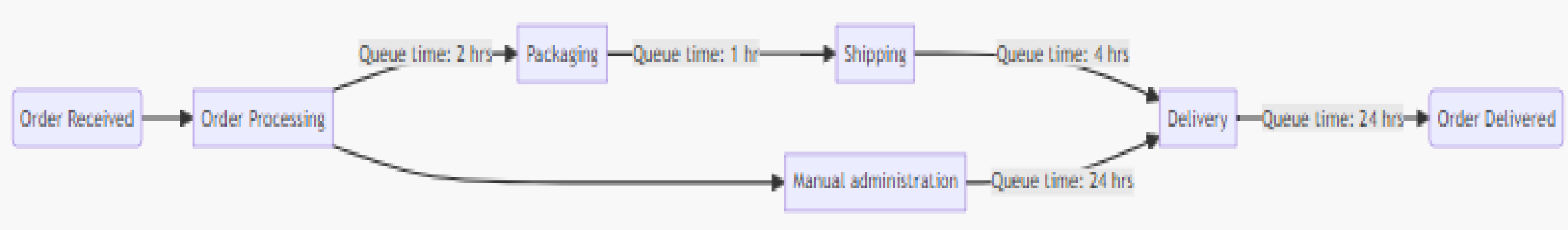
```
flowchart LR
    A[Start: Problem - Car won't start] --> B{Is the battery dead?}
    B -- Yes --> C[Replace battery]
    B -- No --> D{Is the fuel tank empty?}
    D -- Yes --> E[Refill fuel tank]
    D -- No --> F{Is the engine oil level low?}
    F -- Yes --> G[Refill engine oil]
    F -- No --> H[Consult a mechanic]
    C --> I[End: Problem Solved]
    E --> I
    G --> I
    H --> I
```



Mermaid - графы

Процесс доставки заказа

```
flowchart LR
    A(Order Received) --> B[Order Processing]
    B -->|Queue time: 2 hrs| C[Packaging]
    C -->|Queue time: 1 hr| D[Shipping]
    D -->|Queue time: 4 hrs| E[Delivery]
    E -->|Queue time: 24 hrs| F(Order Delivered)
    B --> G[Manual administration]--Queue time: 24 hrs--> E
```



Плагин Mermaid Chart для ChatGPT

Плагин для чата ChatGPT для создания диаграмм Mermaid.

С помощью простых инструкций, таких как:

“Create a State diagram showing the states of a stereo”

“Draw a timeline of World War Two”

можно создавать текстовые диаграммы в ходе чата.

После создания диаграммы ее можно доработать в Mermaid Chart.

Использование возможностей генеративного искусственного интеллекта и обширной базы знаний ChatGPT этот плагин предоставляет хорошую точку начала в синтаксисе диаграмм.

Плагин извлекает соответствующую информацию из Интернета для заполнения содержимого диаграмм без необходимости программирования или дизайна.