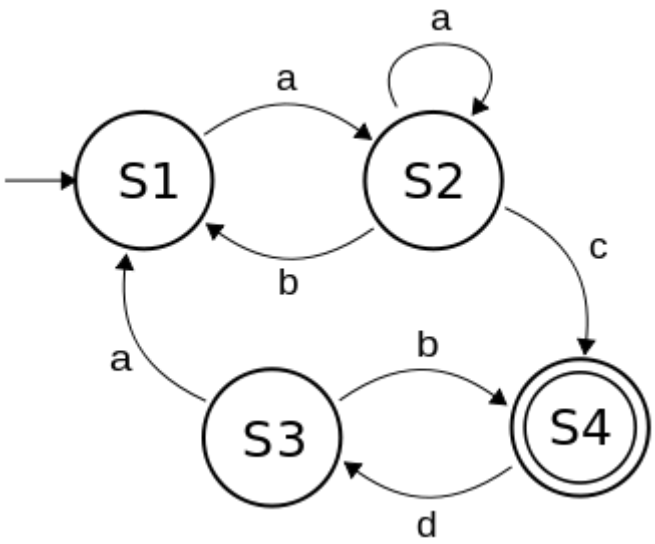
Диаграммы состояний и C++

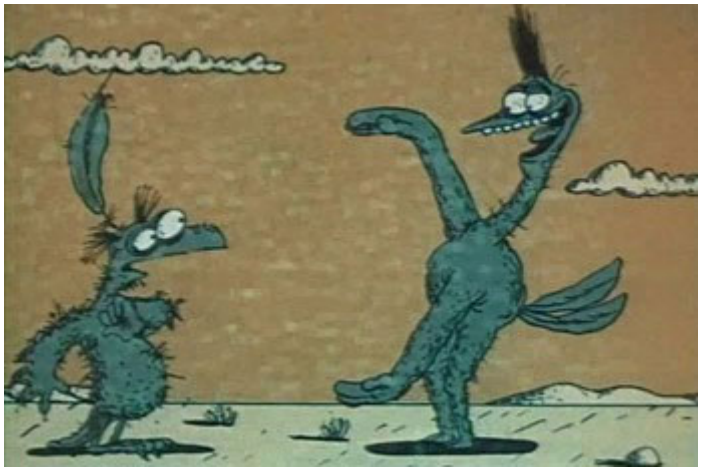
или “Как разобраться с хаосом?”



Конференция C++ CoreHard Winter 2017, Минск

Василий Вяжевич

О чем поговорим?



- Трудная задача (пример из жизни)
- Машина состояний
- Реализации на C++
- Выводы и вопросы

Наглядный пример

- Открыть ворота
- Закрыть ворота
- Запереть

Пример из жизни

```
void openTheDoor()
{
    if (startMotorForward())
        isOpening = true;
    else
        PrintError();
}
```

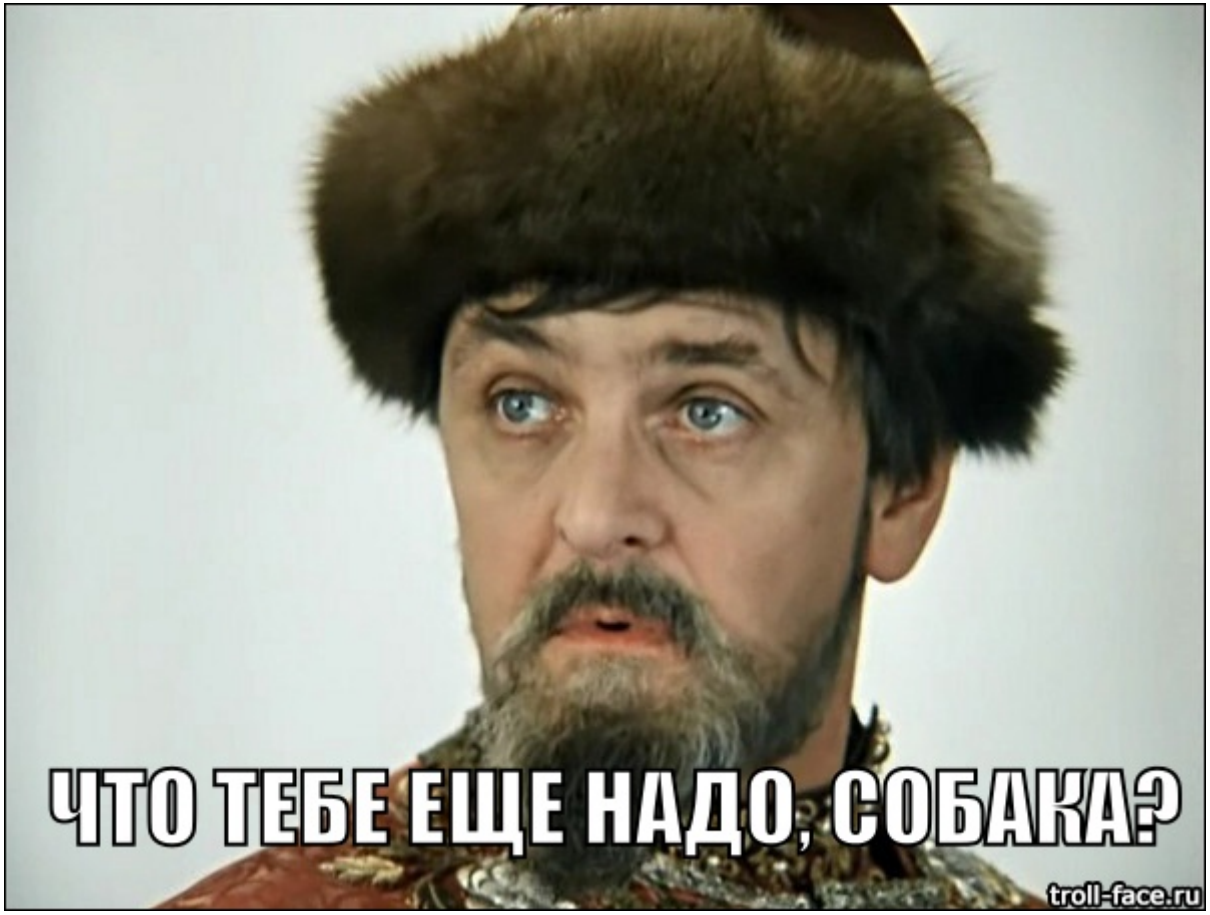
Через XX-цать недель

```
void openTheDoor()
{
    while (!isOpened && !fail && !close) {
        if (isOpened || isOpening) {
           return;
        } else if ((closed && !locked && !isOpening) ||
                   (!closed && !opened && !isOpening)) {
            if (poweredOn && startMotorForward())
                isOpening = true;
        } else if (locked && closed) {
            displayLockedMessage(); return;
        }
        Sleep(100);
    }
    if (close)
       closeTheDoor();
}
```

А что если...[1)]

А что если...

...нужно “чуть-чуть” изменить структуру алгоритма?

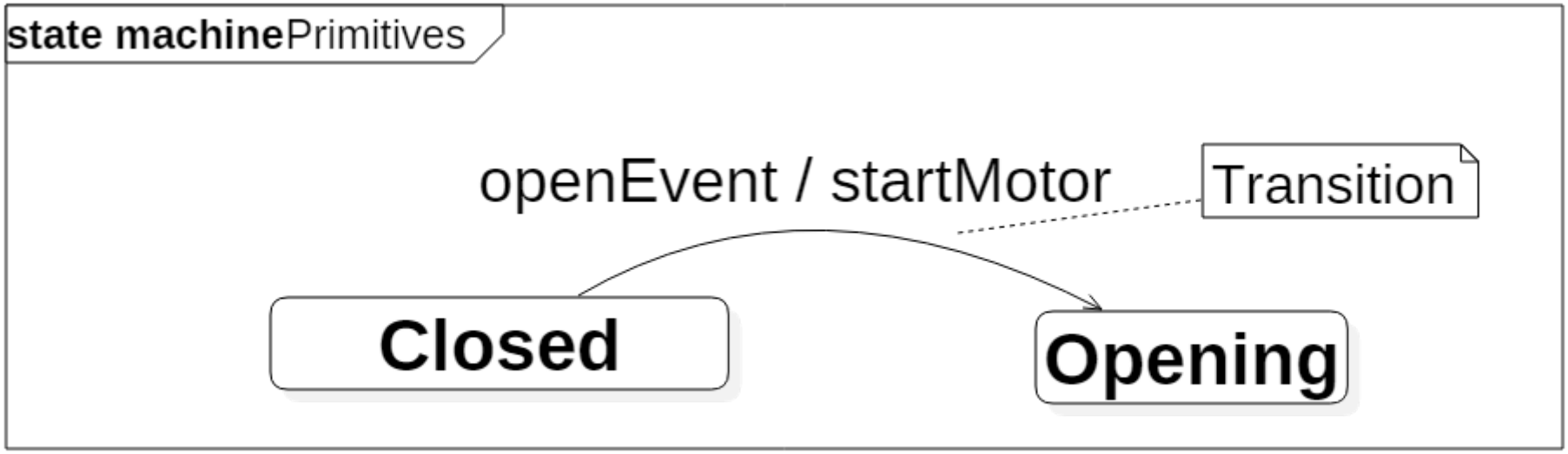


Будьте честными!

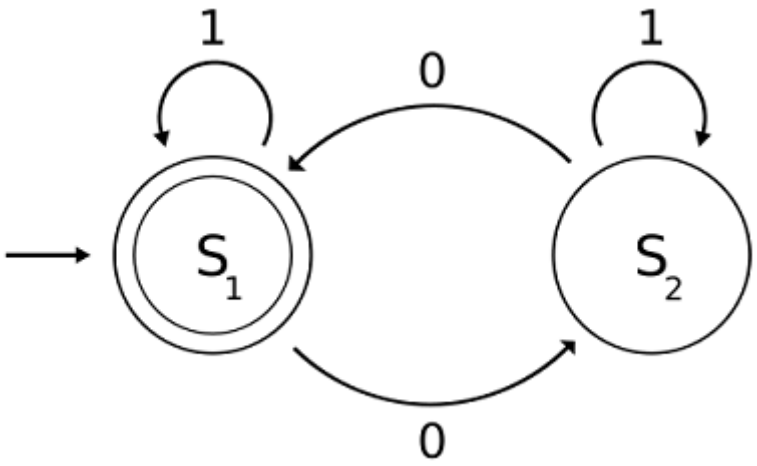
- Часто ли вы делаете блок схемы алгоритмов?
- Можете ли вы легко объяснить как работает Ваша программа?

А какая альтернатива?

```
StateClosed::react(const Event<Open>& event)
{
    if (startMotorForward())
        transit<StateOpening>();
    else
        transit<StateFail>(&Door::DisplayMotorError(), event);
}
```

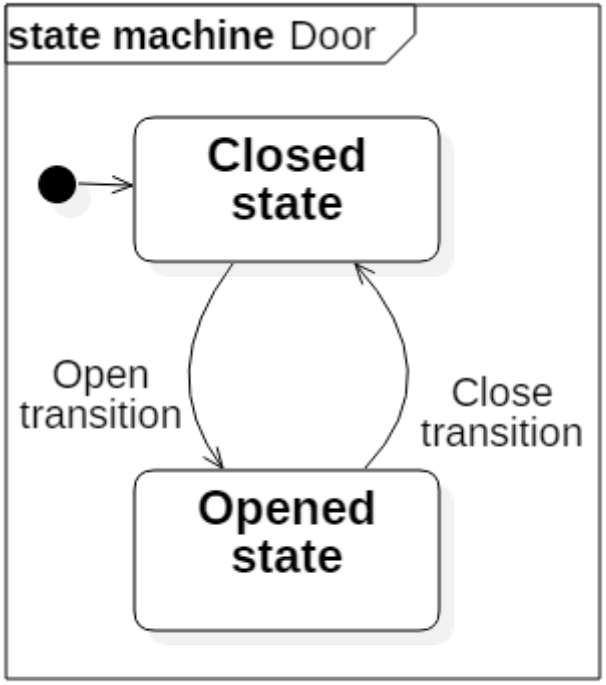


Терминология



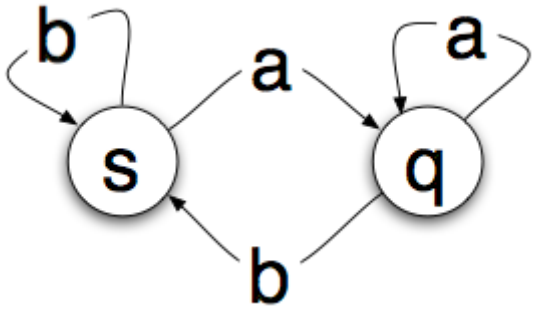
- Машина состояний
- Конечный автомат
- Диаграмма состояний
- Finite State Machine (FSM)
- Statechart

Что такое Машина состояния?



- Математическая модель, абстракция
- Модель или подход к проектированию

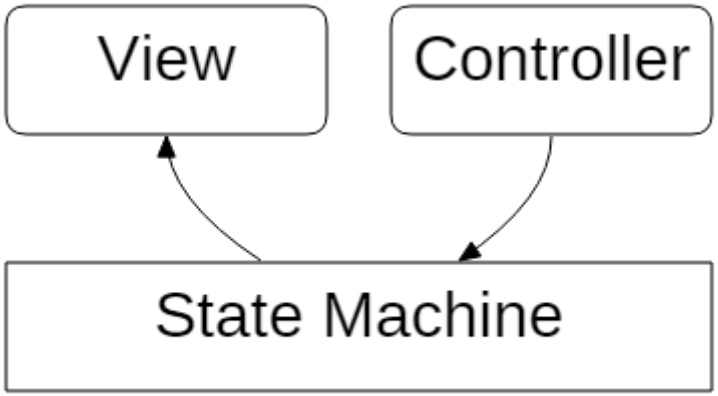
Реализации для C++



- Open source
- Qt (Core / QML)
- Boost
- Custom

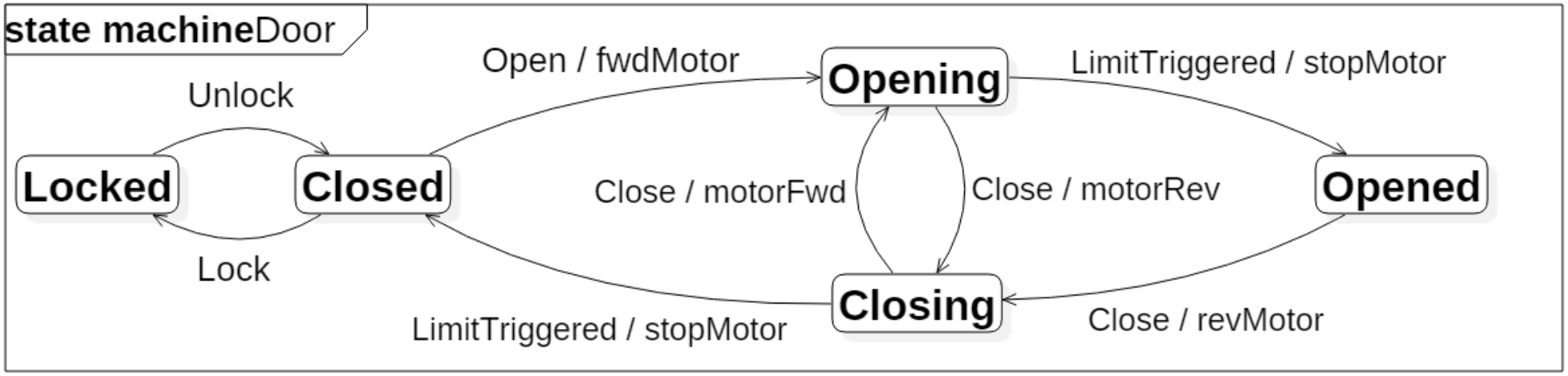
Свойства машин состояний

- Автономность
- Детерминированность состояний
- Основана на событиях (во времени)
- Конечность или цикличность

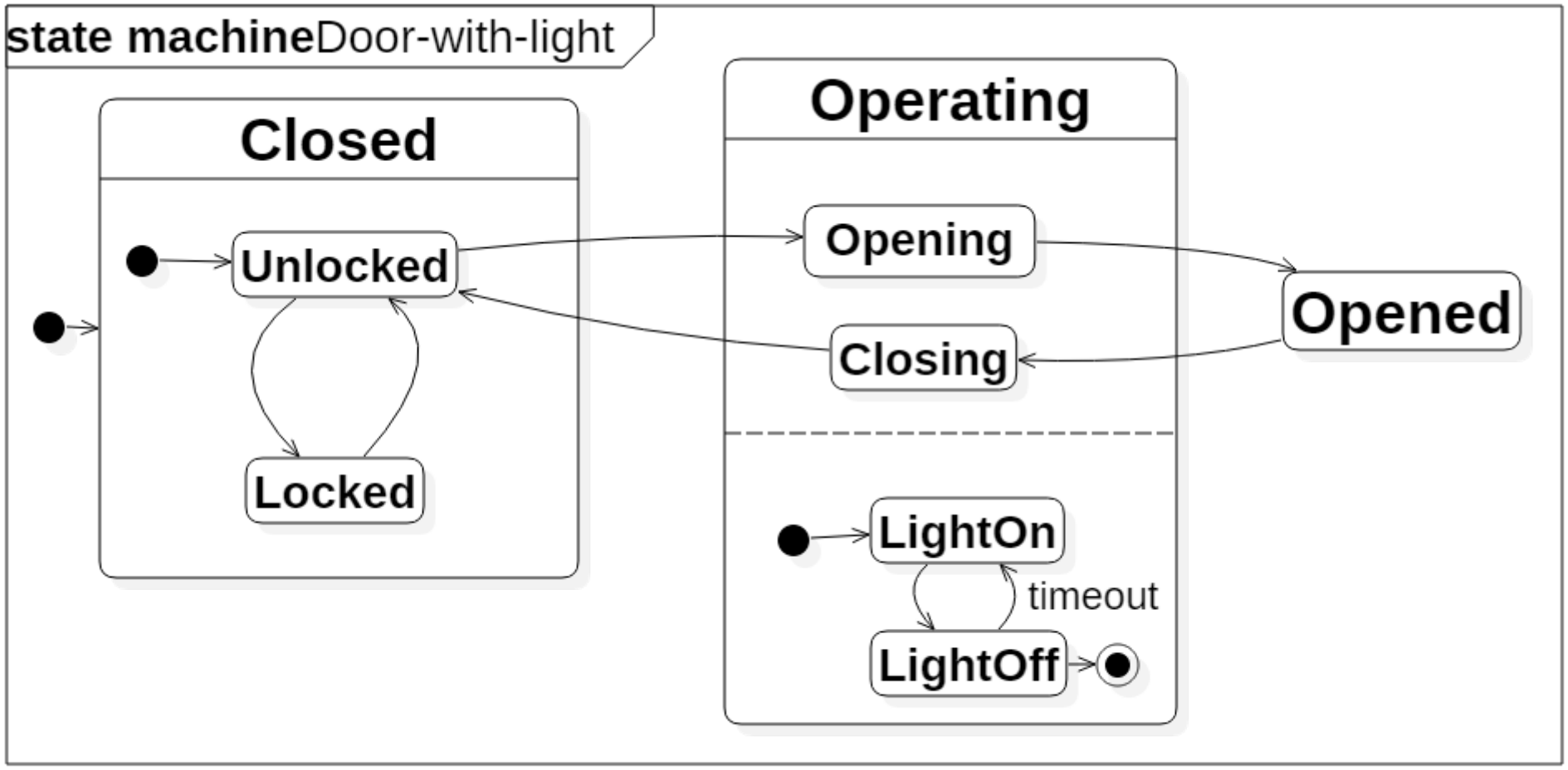


Примитивы

- Состояния (States)
- Переходы (Transitions)
- События (Events)
- Действия (Actions)



Иерархии и ортогональные состояния



Практическая польза

- Проектирование и документирование
- Помогает избежать сложного ветвления
- Определенность последовательности событий
- “Ограничение свободы”

Примеры применения

- Протоколы обмена
- Системы самообслуживания
- Системы автоматики, управления
- Интерактивные игры

Реализации

	cust.	Qt[2)]	MSM[3)]	StCh [4)]	NSF[5)]
Static	+	-	+	+	-
Hierarchy	-	+	-	+	-
Table	-	+	+	-	-
History	-	+	-	+	+
Ortogonal	-	+	-	+	+
Serialize	+	+	+	+	+

Cust. - реализация через оператор switch/case

StCh - Boost.Statechart

NSF - UML North State Framework

switch/case

```
static int currentState = CLOSED;

switch (currentState)
{
    case CLOSED:
        if (event == OPEN)
            currentState = OPENED;
        else if (event == LOCK)
            currentState = LOCKED;
        break;
    case OPENED:
        if (event == CLOSE)
            currentState = CLOSED;
        break;
    case LOCKED:
        ...
}
```

Qt State Machine Framework

```
QStateMachine Door;

QState *stateOpened = new QState();
QState *stateClosed = new QState();
QState *stateLocked = new QState();

stateOpened->addTransition(ctrl, SIGNAL(close()), stateClosed);
stateClosed->addTransition(ctrl, SIGNAL(open()), stateOpened);
stateLocked->addTransition(ctrl, SIGNAL(lockTrigger()), stateClosed);
stateClosed->addTransition(ctrl, SIGNAL(lockTrigger()), stateLocked);
```

Boost.Statechart

```
struct Open : sc::event<Open> {};
struct Close : sc::event<Close> {};
struct LockTrigger : sc::event<LockTrigger> {};

struct Opened : sc::simple_state<Opened, Door> {
    typedef sc::transition<Close, Closed> reactions;
};

struct Closed : sc::simple_state<Closed, Door> {
    typedef mpl::list<sc::transition<Open, Opened>
                      sc::transition<LockTrigger, Locked>
                     > reactions;
};

struct Locked : sc::simple_state<Locked, Door> {
    typedef sc::transition<LockTrigger, Closed> reactions;
};
```

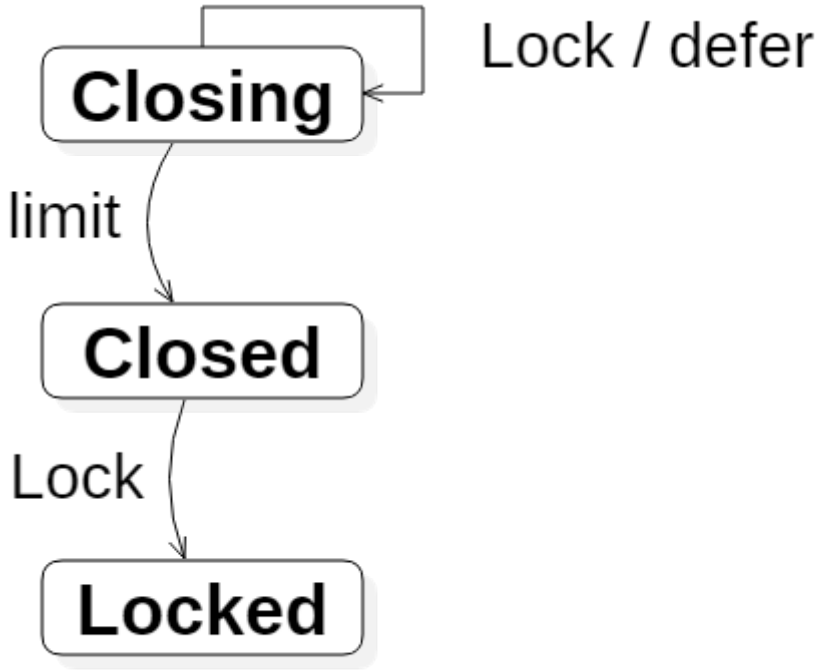
Boost Meta State Machine

```
struct transition_table : mpl::vector<
//     Start     Event         Target      Action
//   +---------+-------------+-----------+--------------------+
a_row< Closed  , open        ,  Opened   , &door_::motorFwd  >,
a_row< Closed  , lock        ,  Locked   , &door_::lock      >,
a_row< Locked  , lock        ,  Closed   , &door_::unlock    >,
a_row< Opened  , close       ,  Closed   , &door_::motorBack >,
//   +---------+-------------+-----------+--------------------+
> {};
```

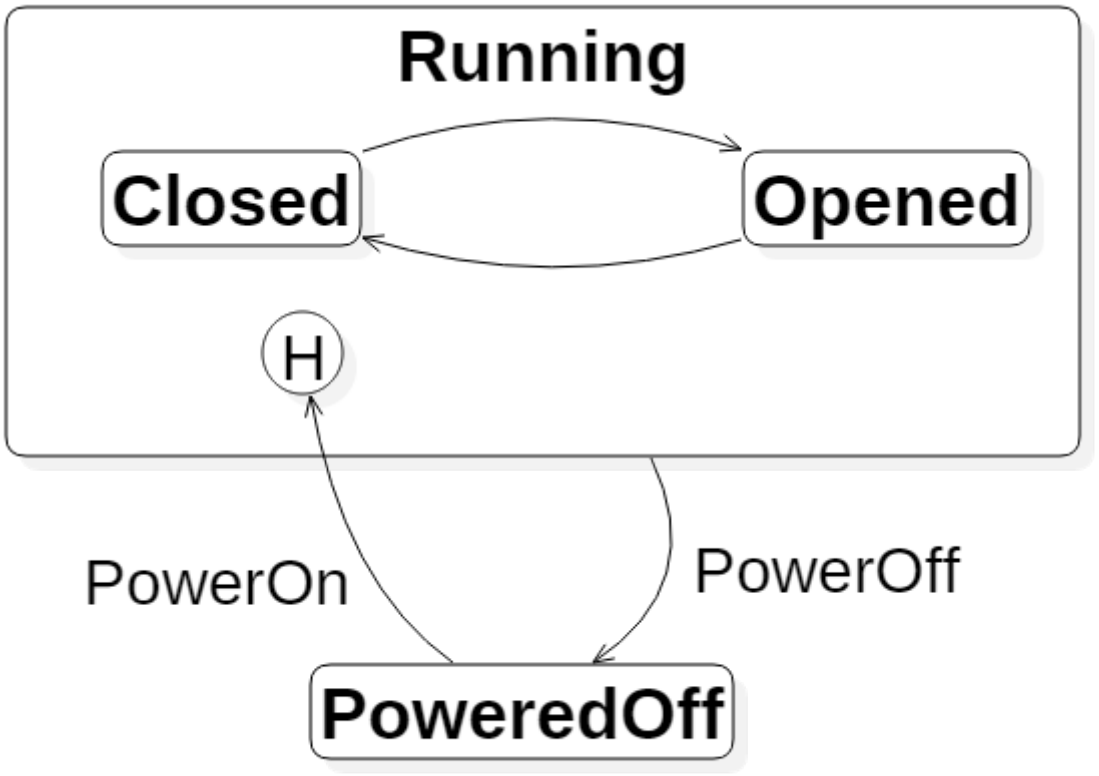
Бонусы от реализаций

- Отсроченные события (deferred events)
- История состояний (History)
- Сериализация машин состояний
- Контроль переходов состояний
- Тестирование (Unit testing)

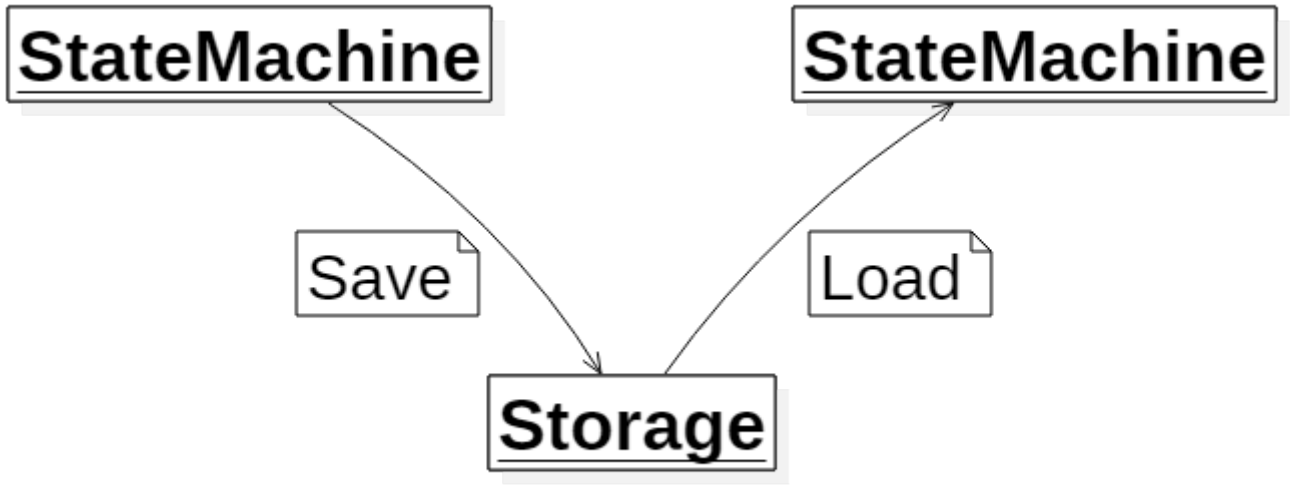
Отсроченные события



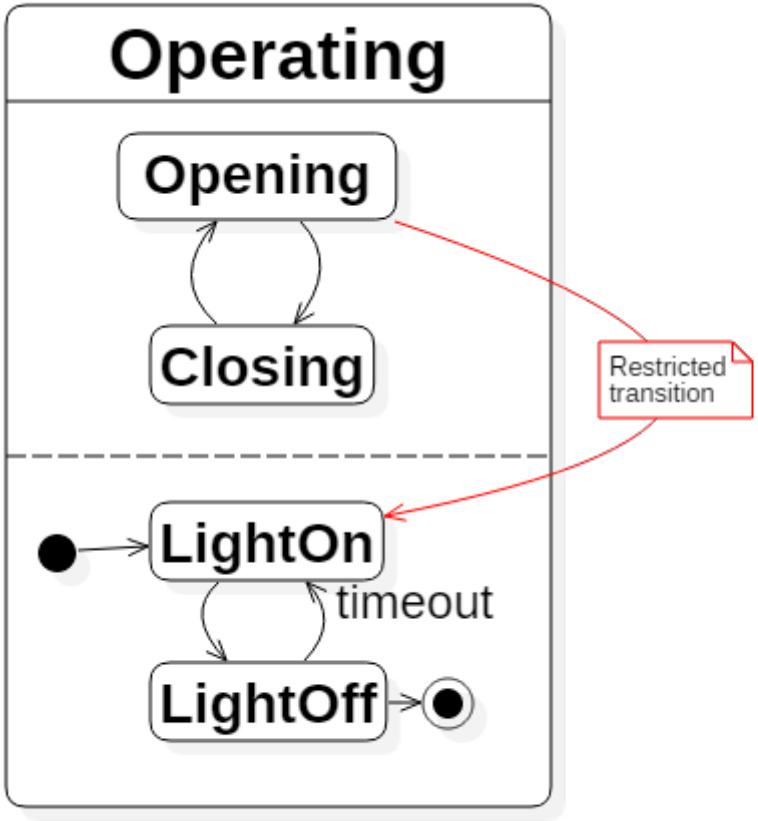
История состояний



Сериализация



Контроль переходов



Заключение

- Зачем? - Формализация процесса разработки, проектирование и документирование.
- Для чего? - Алгоритмы основанные на событиях.
- Почему FSM? - Если не знаете ничего лучше... 

Вопросы?

Тема: Диаграммы состояний и C++.

Докладчик: Василий Вяжевич

Компания: Klika Tech,
http://klika-tech.com/,
vviazhevich@klika-tech.com

[1] Внимание! Все названия переменных и алгоритмы вымышленные, любые совпадения случайны!
[2] http://doc.qt.io/qt-5/statemachine.html
[3] http://www.boost.org/doc/libs/1_63_0/libs/msm/doc/HTML/index.html
[4] http://www.boost.org/doc/libs/1_63_0/libs/statechart/doc/index.html
[5] http://northstatesoftware.github.io/NorthStateFramework-cpp/