Embedded Systems Design using UML State Machines

FASTBIT EMBEDDED BRAIN ACADEMY

www.fastbitlab.com

License

Important Links

- For the full video course please visit
  - https://www.udemy.com/course/embedded-system-design-using-uml-state-machines/
- Course repository
  - https://github.com/niekiran/EmbeddedUMLStateMachines
- Explore all FastBit EBA courses
  - http://fastbitlab.com/course1/
- For suggestions and feedback
  - contact@fastbitlab.com

Social media

- Join our Facebook private group for technical discussion
  - https://www.facebook.com/groups/fastbiteba/
- Linkedin
  - https://www.linkedin.com/company/fastbiteba/
- Facebook
  - https://www.facebook.com/fastbiteba/
- YouTube
  - https://www.youtube.com/channel/UCa1REBV9hyrzGp2mjJCagBg
- Twitter
  - https://twitter.com/fastbiteba

What is a state machine(FSM) ?

- A state machine is a software model of computation, and it comprises finite number states. Hence it is also called a Finite State Machine(FSM)
- Since states are finite, there is a finite number of transitions among the states. Transitions are triggered by the input events fed to the state machine(FSM is an event-driven system).
- A state machine also produces an output. The output produced depends on the current state of the state machine and the input events fed to the state machine.

Benefits of using state machines(FSMs)

- Used to describe situations or scenarios of your application(Modelling the life cycle of a reactive object through interconnections of states)
- FSMs are helpful to model complex applications that involve lots of decision-making, producing different outputs (actions), and processing various events.
- State machines are visualized through state machine diagrams in the form of state charts, which helps to communicate between non-developers and developers.
- Makes it easier to visualize and implement the changes to the behavior of the project
- complex application can be visualized as a collection of different states processing a fixed set of events and producing a fixed set of outputs
- Loose coupling: An application can be divided into multiple behaviors or state machines, and each unit can be tested separately or could be reused in other applications
- Easy debugging and easy code maintenance
- Scalable
- Narrow down whole application complexity to state level complexity, analyze and implement

Different types of state machines

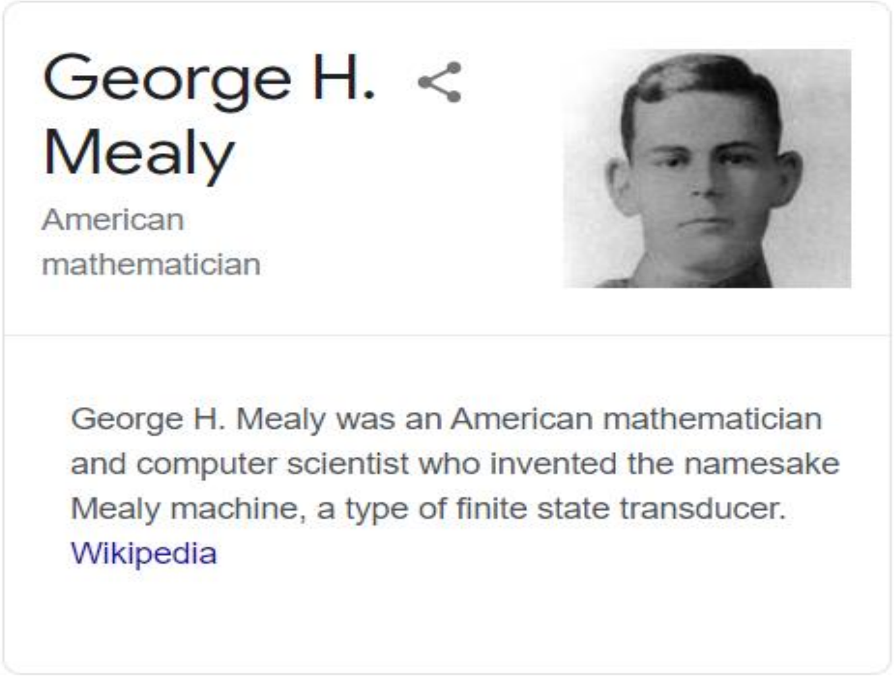
- Mealy machines
- Moore machines
- Harel state charts
- UML state machines

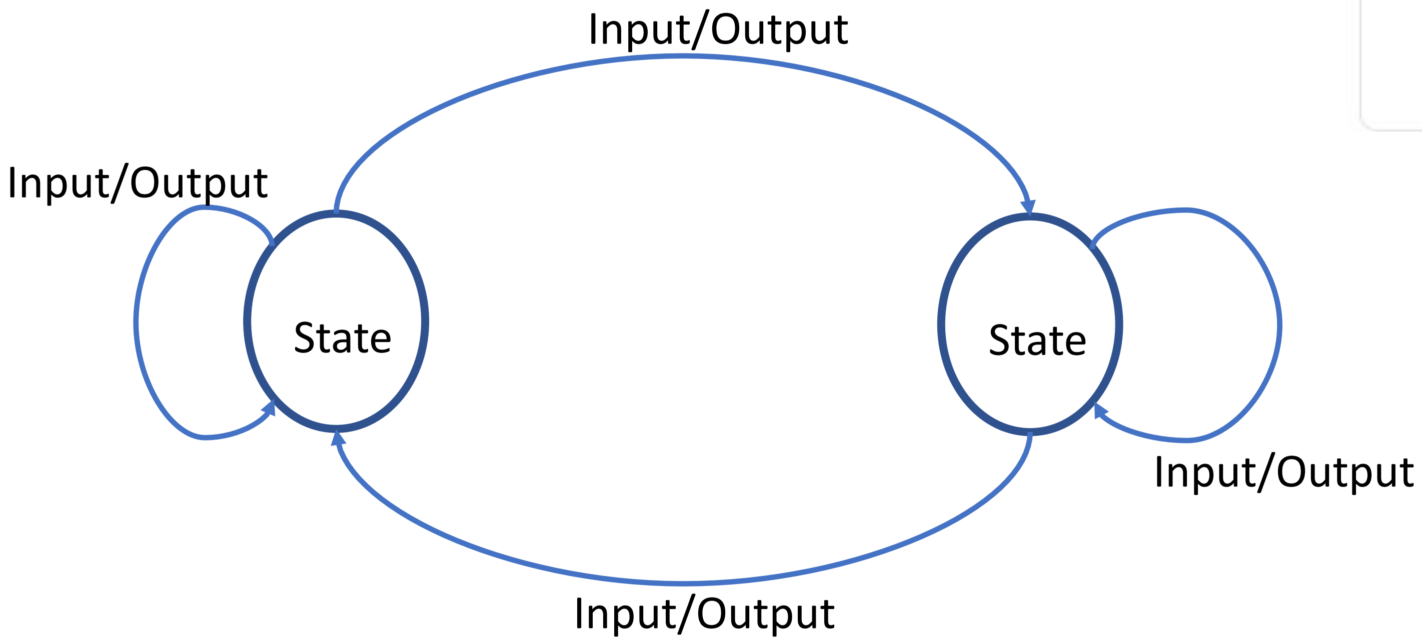
UML Modelling tool and code generator

- Rhapsody® by IBM®
- QM™ Model-based design tool by Quantum Leaps, LLC
- Visual State by IAR
- Yakindu state chart tools by Itemis AG

Mealy machine

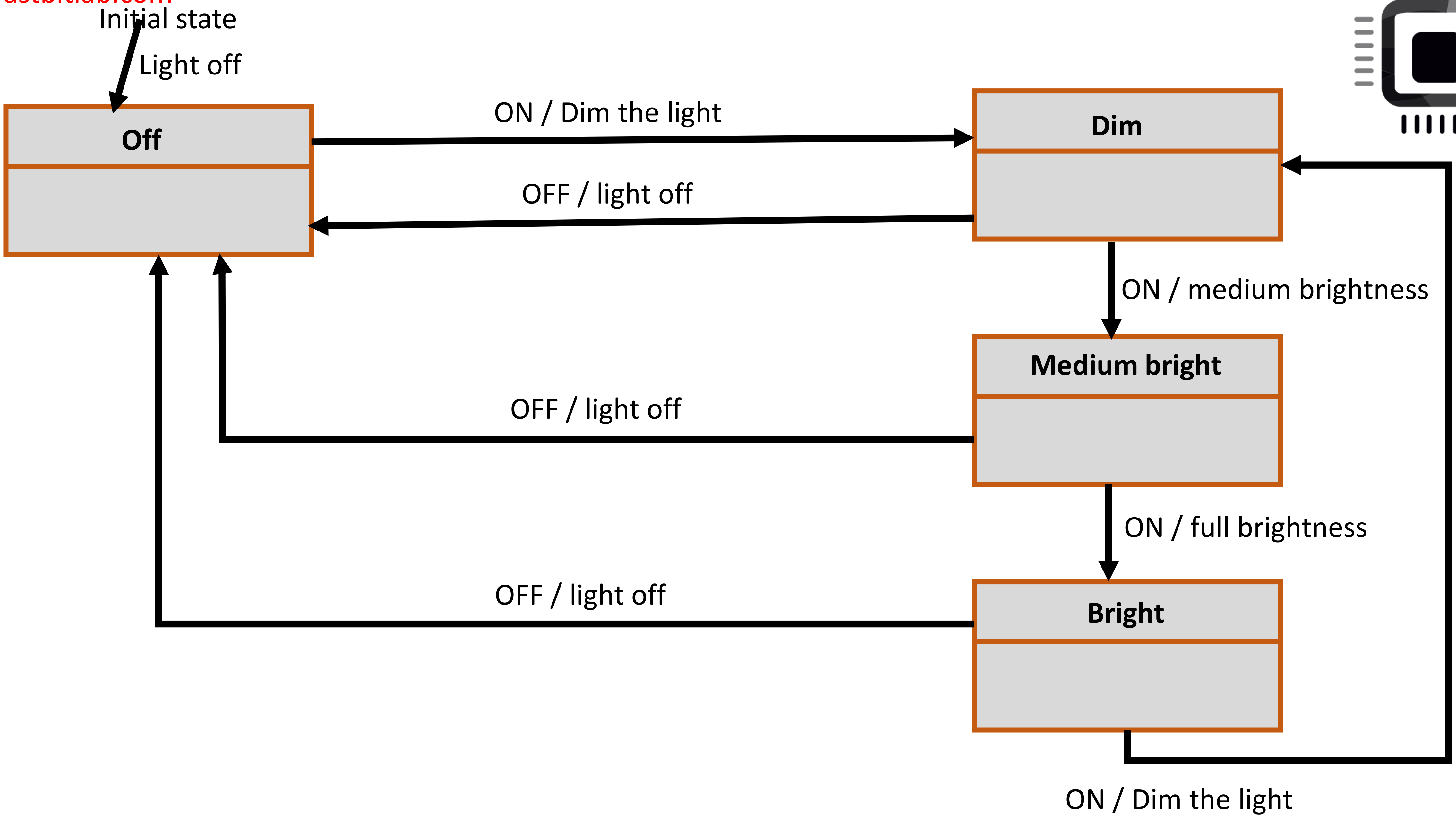
- In this machine, the output produced by the state machine depends on the input events fed to the state machine AND present active state of the state machine.
- The output is not produced inside the state





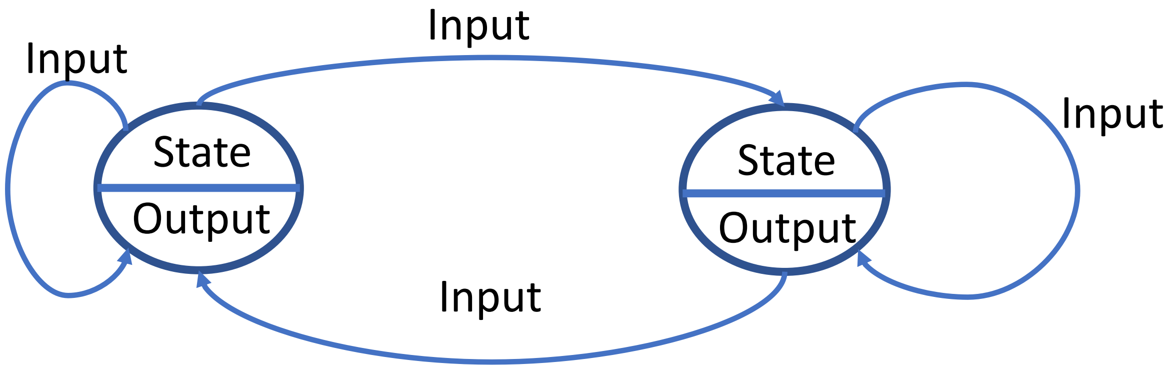
Generic Mealy model

1) Output is represented along with each input separated by '/'
2) An 'Output' is also called 'Action.'
3) In the mealy model, the 'Output' is also called 'Input action.'


Initial state
Light off
Off
ON / Dim the light
OFF / light off
Dim
ON / medium brightness
Medium bright
OFF / light off
ON / full brightness
Bright
OFF / light off
ON / Dim the light

Moore machine

- In this state machine, the output is determined only by the present active state of the state machine and not by any input events
- No output during state transition.



Generic Moore model

Edward F. Moore

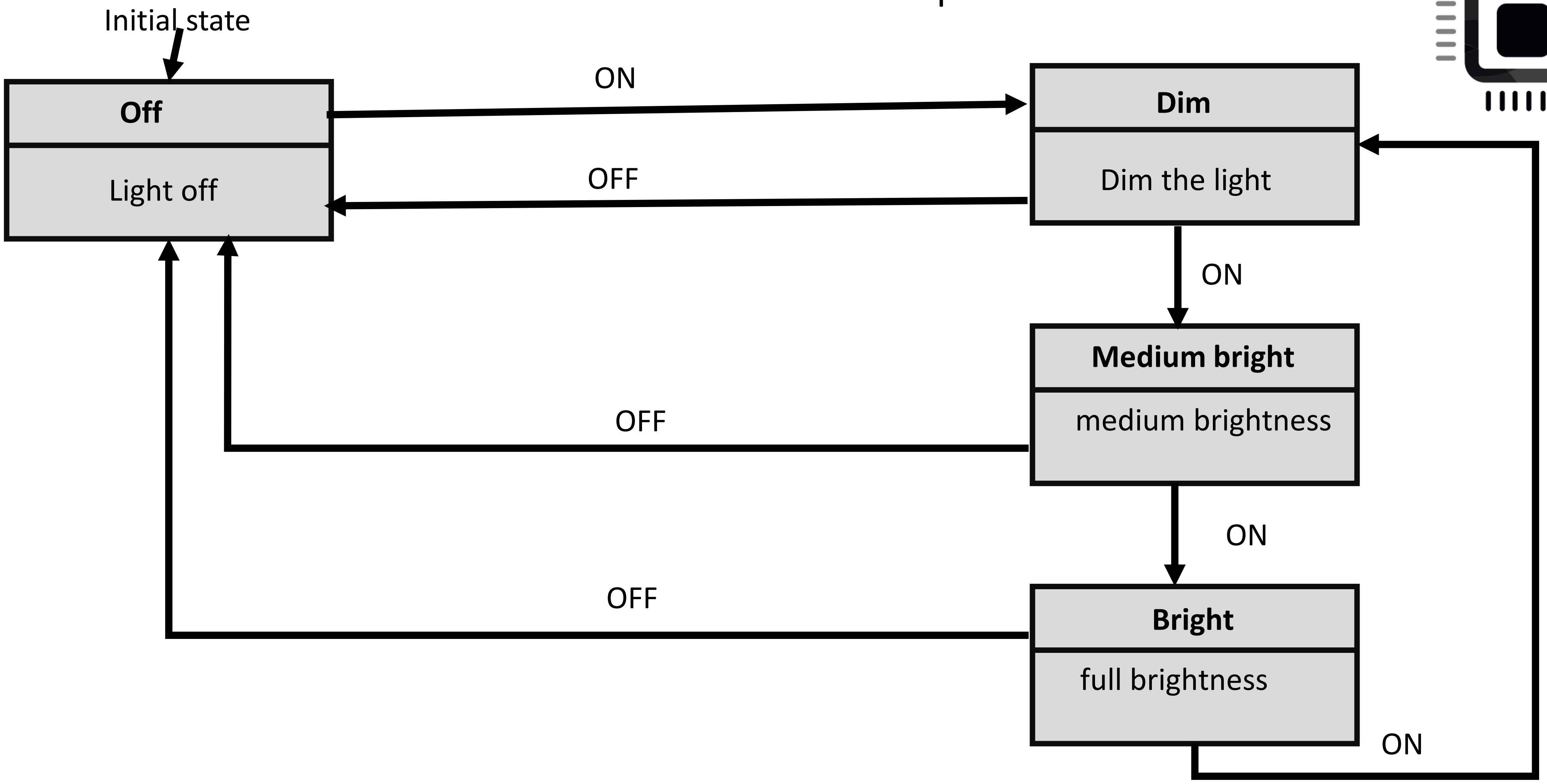
American professor of mathematics

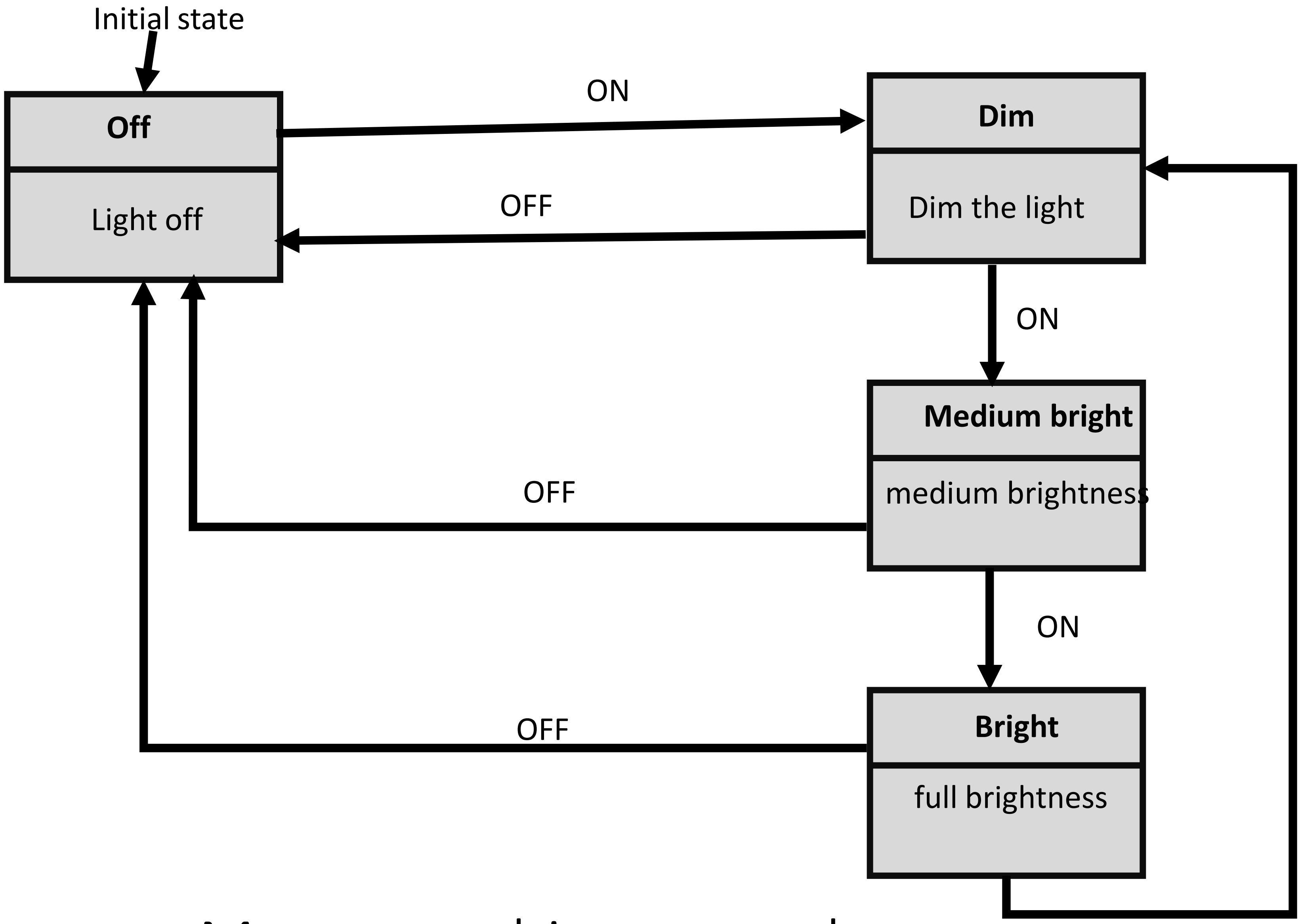
Edward Forrest Moore was an American professor of mathematics and computer science, the inventor of the Moore finite state machine, and an early pioneer of artificial life. Wikipedia

- Output is represented inside the state
- 'Output' is also called 'Action.'
- In the Moore model the 'Output' is also called 'Entry action.'

www.fastbitlab.com

Moore machine example





Input events

ON

OFF

Entry actions

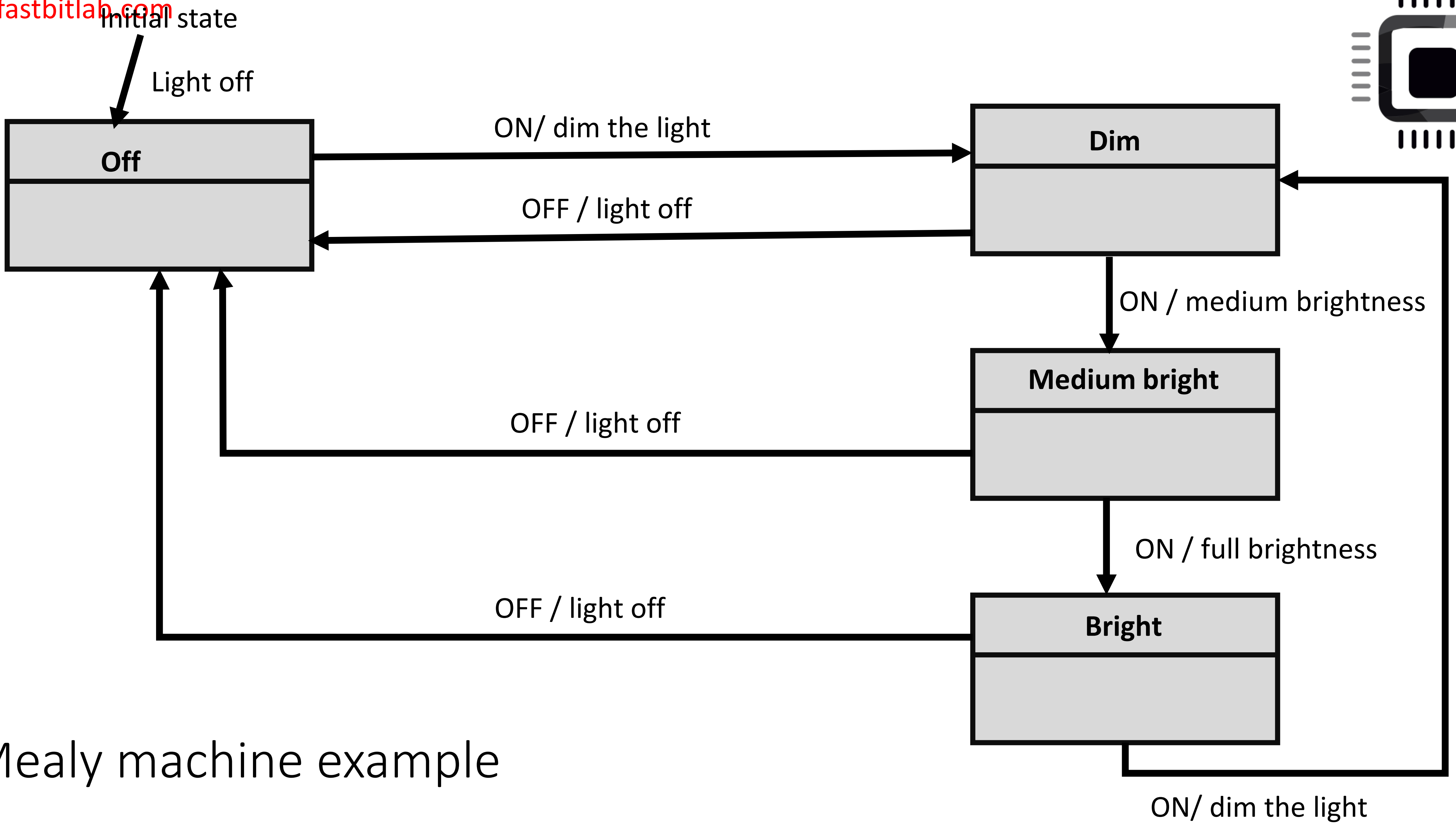
1) Light off
2) Make light dim
3) Make light medium brightness
4) Make light full brightness

Moore machine example

State transition table
Light control Moore machine

State	Entry actions (output)	Next state	
		Input events	
		OFF	ON
Off	Light off	---ignored--	Dim
Dim	Make light dim	Off	Medium
Medium	Make light medium	Off	Bright
Bright	Make light bright	Off	Dim

- When the state machine enters the ‘Off’ state, it executes the ‘Light off’ entry action and waits until an input event is received.
- If the input event received is ‘ON’, the state machine moves to the state ‘Dim.’
- If the input event received is ‘OFF’, event is ignored



Mealy machine example

State transition table
Light control Mealy machine

Present state	Next state			
	Input events			
	OFF		ON	
	Next state	input action (output)	Next state	input action (output)
Off	off	--ignored--	Dim	Make light dim
Dim	Off	Light off	Medium	Make light medium
Medium	Off	Light off	Bright	Make light bright
Bright	Off	Light off	Dim	Make light dim

When the state machine is in 'Off' state and if

- Input event 'ON' is received, then state machine produces the output 'Make light dim' and makes a transition to 'Dim' state
- Input event 'OFF' is received, the event is ignored

Harel state charts

WV

STATECHARTS: A VISUAL FORMALISM FOR COMPLEX SYSTEMS*

Communicated by A. Pnueli
Received December 1984
Revised July 1986

David HAREL

Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel

Abstract. We present a broad extension of the conventional formalism of state machines and state diagrams, that is relevant to the specification and design of complex discrete-event systems, such as multi-computer real-time systems, communication protocols and digital control units. Our diagrams, which we call *statecharts*, extend conventional state-transition diagrams with essentially three elements, dealing, respectively, with the notions of hierarchy, concurrency and communication. These transform the language of state diagrams into a highly structured and economical description language. Statecharts are thus compact and expressive—small diagrams can express complex behavior—as well as compositional and modular. When coupled with the capabilities of computerized graphics, statecharts enable viewing the description at different levels of detail, and make even very large specifications manageable and comprehensible. In fact, we intend to demonstrate here that statecharts counter many of the objections raised against conventional state

Harel statecharts = state-diagrams

+

depth

+

orthogonality

+

broadcast communication

Harel statecharts features

- Substates and superstates
- Composite states
- History states(shallow and deep)
- Orthogonality
- Communication between state machines
- Conditional transitions(guards)
- Entry and exit actions
- Activities inside a state
- Parameterized states
- Overlapping states
- Recursive statecharts

UML state machines

14 StateMachines

14.1 Summary

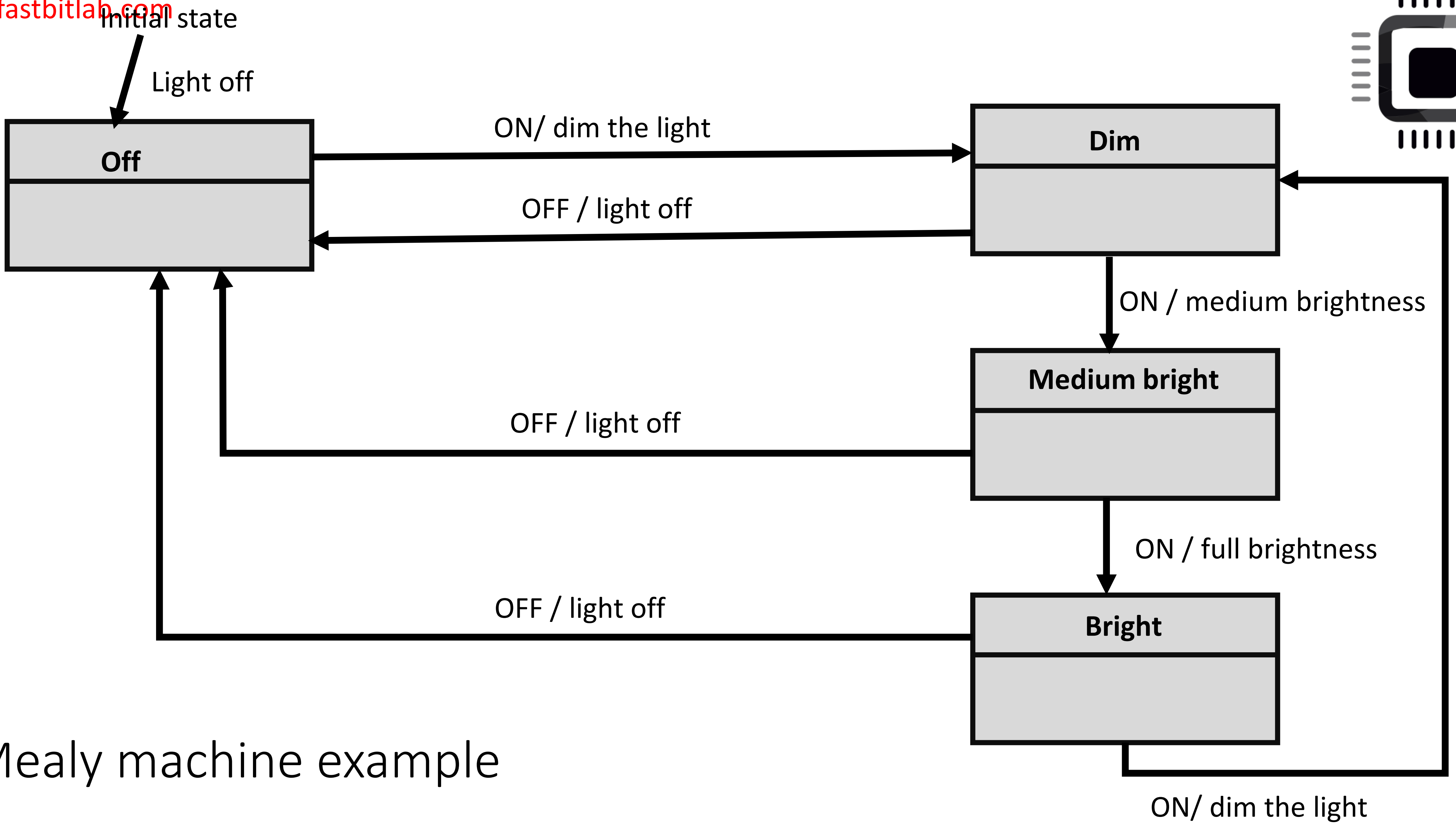
The StateMachines package defines a set of concepts that can be used for modeling discrete event-driven Behaviors using a finite state-machine formalism. In addition to expressing the Behavior of parts of a system (e.g., the Behavior of Classifier instances), state machines can also be used to express the valid interaction sequences, called *protocols*, for parts of a system. These two kinds of StateMachines are referred to as *behavior state machines* and *protocol state machines* respectively.

The specific form of finite state automata used in UML is based on an object-oriented variant of David Harel's *statecharts* formalism. (However, readers who are familiar with that formalism should note that there is a small number of semantic differences that distinguish the UML version from the original.)

[OMG® UML 2.5.1]

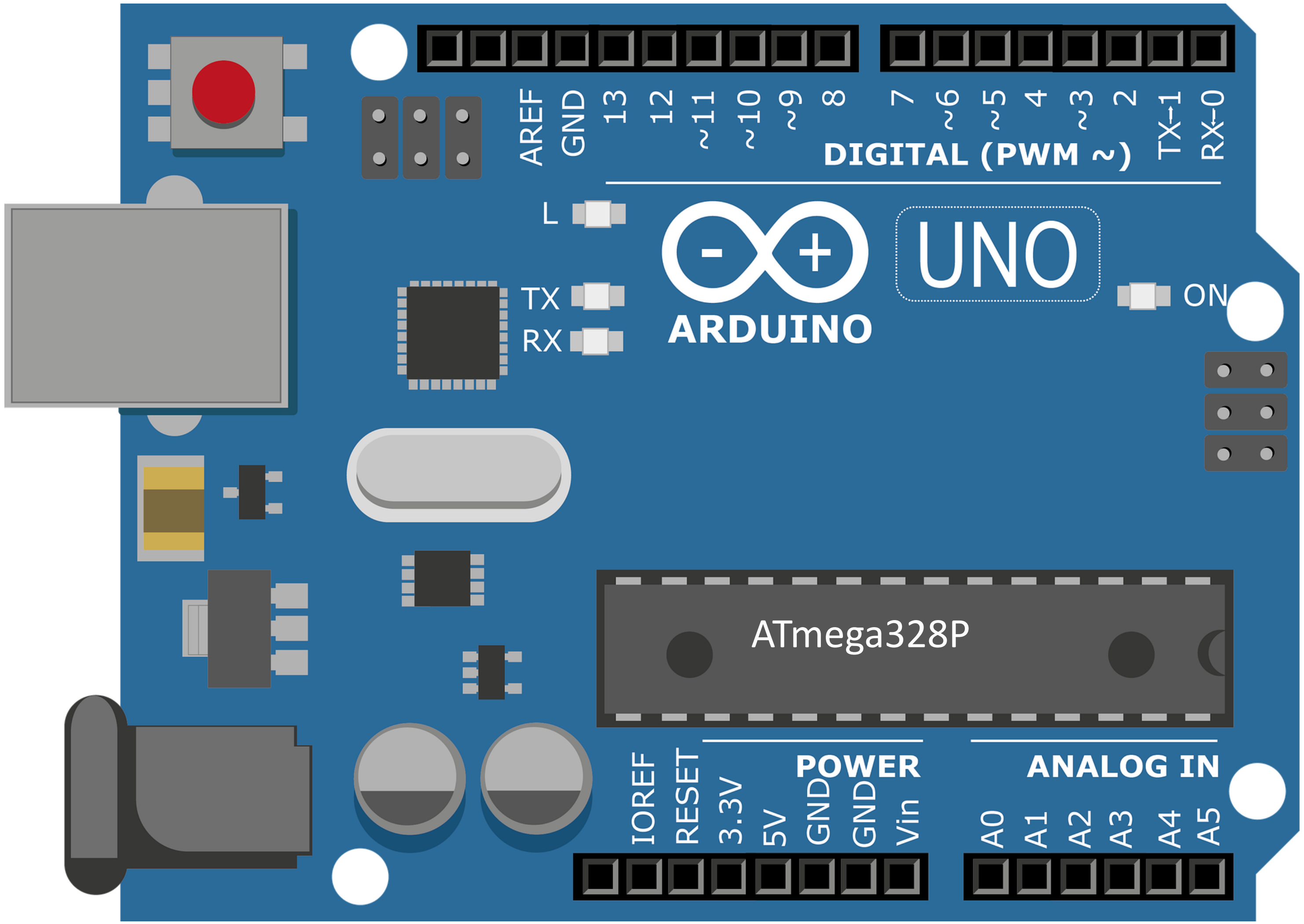
Light control Mealy machine implementation

- Arduino Uno board
- 1, 5mm LED
- Jumper wires



Mealy machine example

www.fastbitlab.com

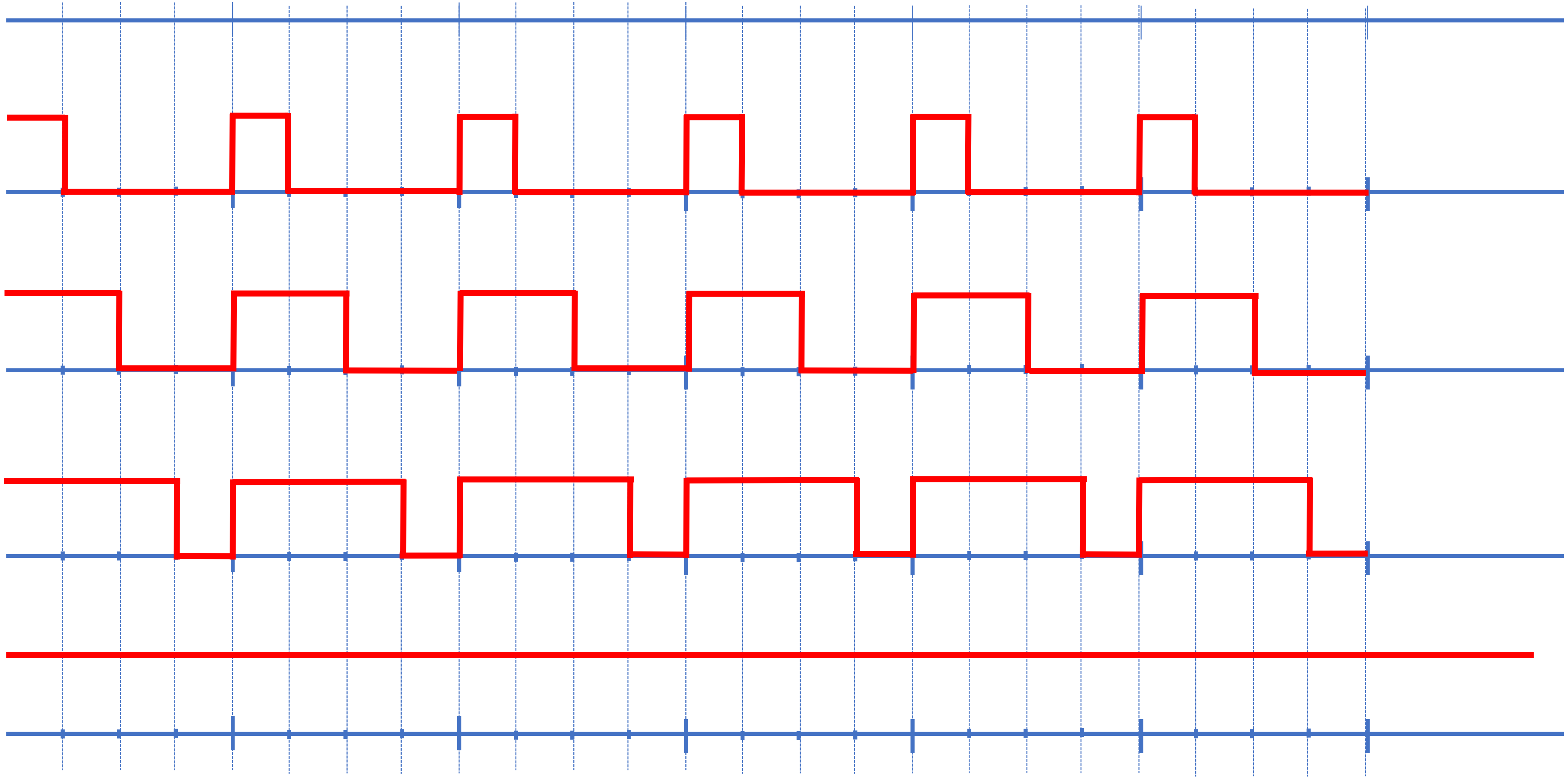


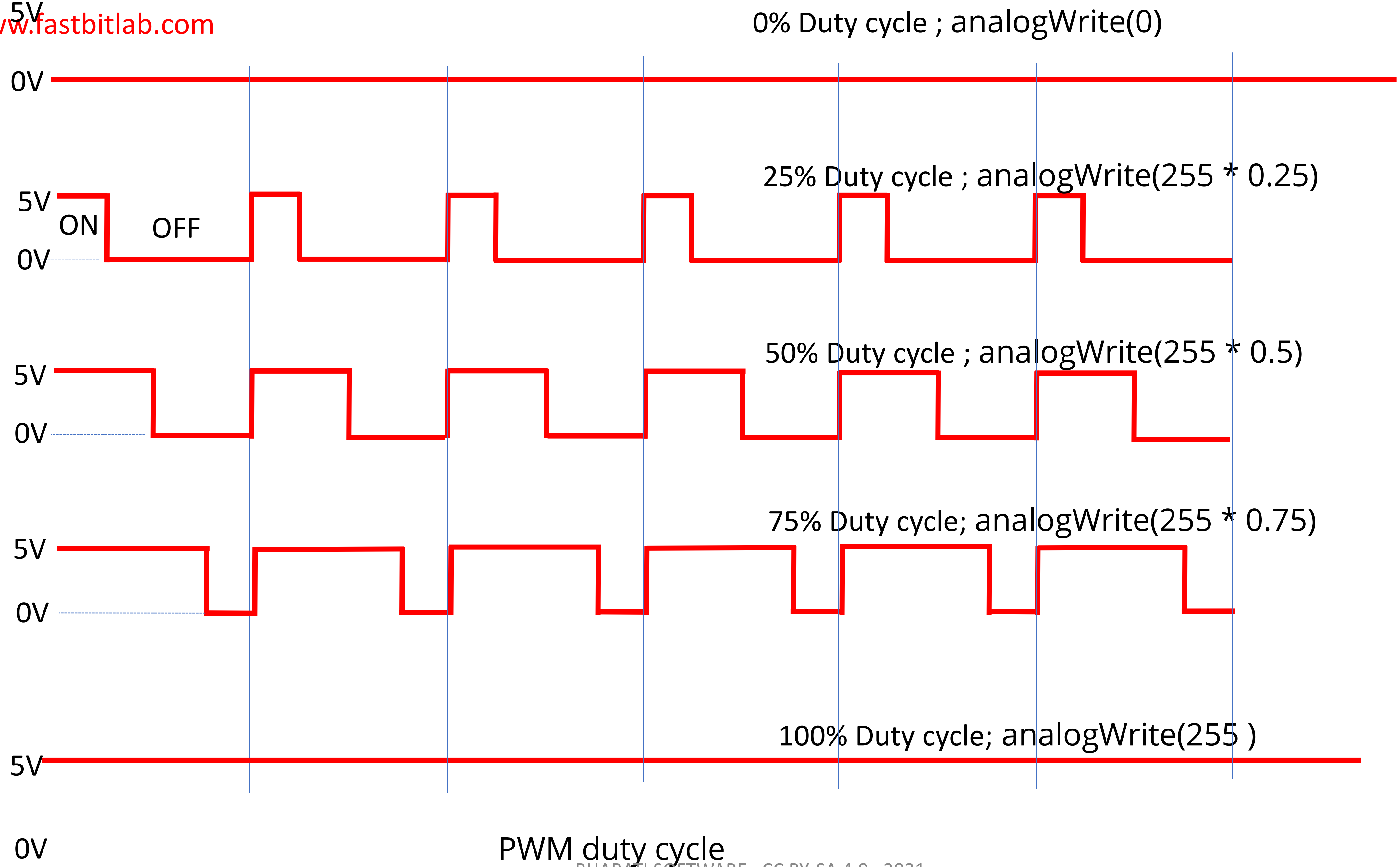
Arduino PWM pins

- Uno : 3, 5, 6, 9, 10, 11 are PWM pins
- On these pins Arduino uno can generate PWM signals
- PWM signal frequency :
  - 490Hz on 3,9,10,11
  - 980Hz on 5 and 6

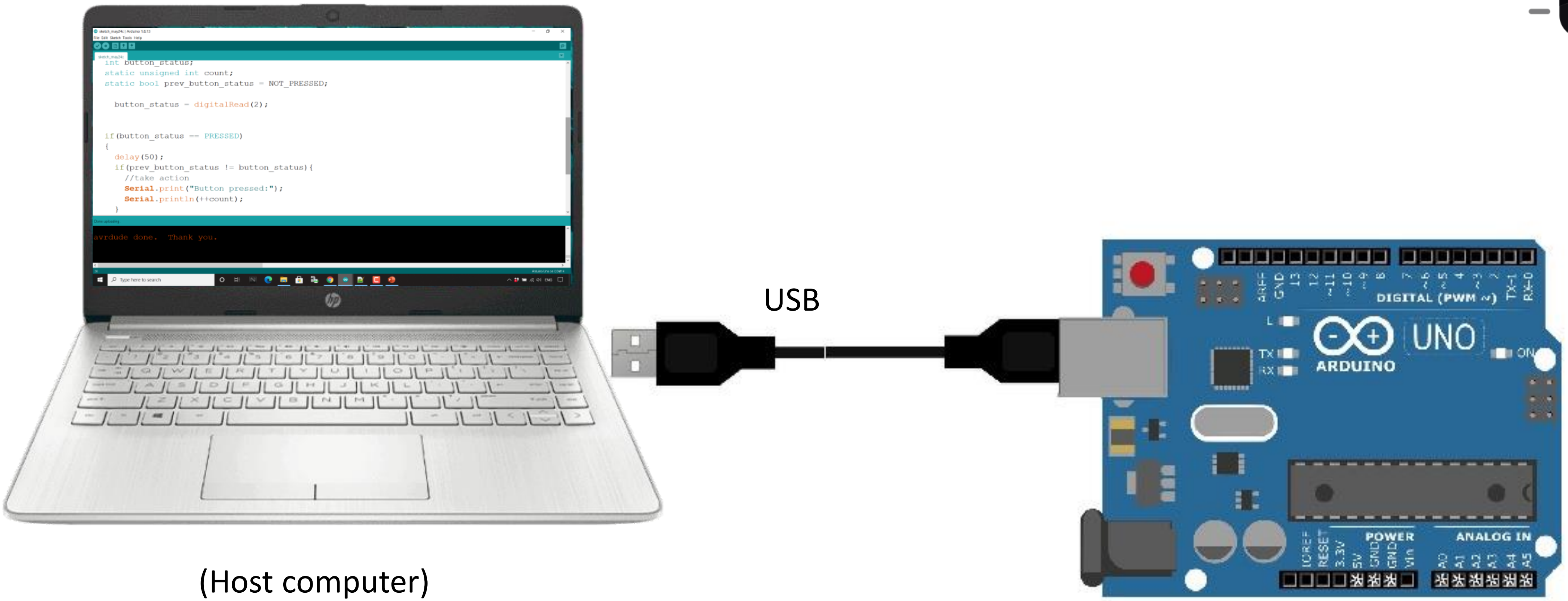
Changing intensity of an LED

1. Connect the LED to any of the PWM pins
2. Change the duty cycle of the PWM wave using the Arduino API analogWrite()
3. Use the Arduino serial interface to send ON, OFF events from HOST





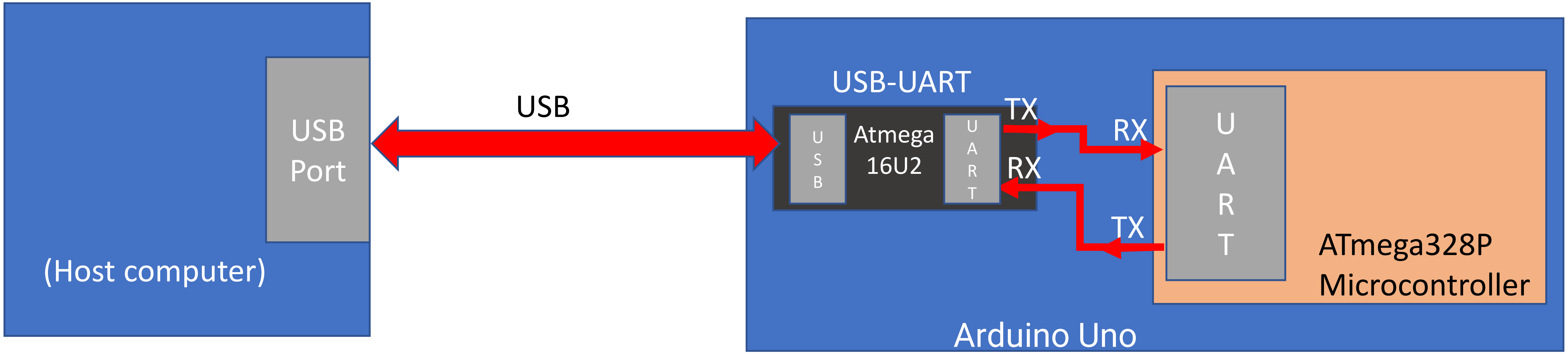
PWM duty cycle



(Host computer)

(Target)

Arduino serial(UART) communication with HOST



Arduino serial(UART) communication with HOST

Enumeration(enum) in C

- enum in C/C++ is a user-defined data type.
- By using enum, you can define your own data type.
- Provides some clarity to your program
- To use enum, enumerator-list must be known in advance (a finite list set of named integer constant values a data type can take on)

Example

I want to create data type to represent the day of a week.

```
enum Day_of_week
{
  SUNDAY,
  MONDAY,
  TUESDAY
};

enum Day_of_week whatIsToday;
whatIsToday = MONDAY;
```

```c
typedef enum{ /* tag less */
    MEETING,
    CODING,
    TRAVELLING = 20,
    READING,
    EATING
}activity_t; /* Typedef alias */
```

The enumerated values are in the set { 0, 1, 20, 21,22 }.

Constraints

2 The expression that defines the value of an enumeration constant shall be an integer constant expression that has a value representable as an `int`.

Semantics

3 The identifiers in an enumerator list are declared as constants that have type `int` and may appear wherever such are permitted.[107] An enumerator with `=` defines its enumeration constant as the value of the constant expression. If the first enumerator has no `=`, the value of its enumeration constant is 0. Each subsequent enumerator with no `=` defines its enumeration constant as the value of the constant expression obtained by adding 1 to the value of the previous enumeration constant. (The use of enumerators with `=` may produce enumeration constants with values that duplicate other values in the same enumeration.) The enumerators of an enumeration are also known as its members.

4 Each enumerated type shall be compatible with `char`, a signed integer type, or an unsigned integer type. The choice of type is implementation-defined,[108] but shall be capable of representing the values of all the members of the enumeration. The enumerated type is incomplete until after the `}` that terminates the list of enumerator declarations.

Exercise 2 : Productivity Timer(ProTimer)

The application that tracks your productive time

Project requirements

- + button → Increment time (minutes should increase)
- – button → Decrement time (minutes should decrease)
- S/P button → Start/pause the countdown; show STAT
- When the countdown is paused, time can be modified.
- Press the + and – button simultaneously to abort the running timer
- Application must beep 20 times when it returns to IDLE mode
- When the application is in IDLE mode, pressing the S/P button should show the STAT for 1 sec and auto return to IDLE mode.

State

What is a state?
How do you arrive at fixing a state?

state

What is a state?
State represents a distinct stage of an object's lifecycle.

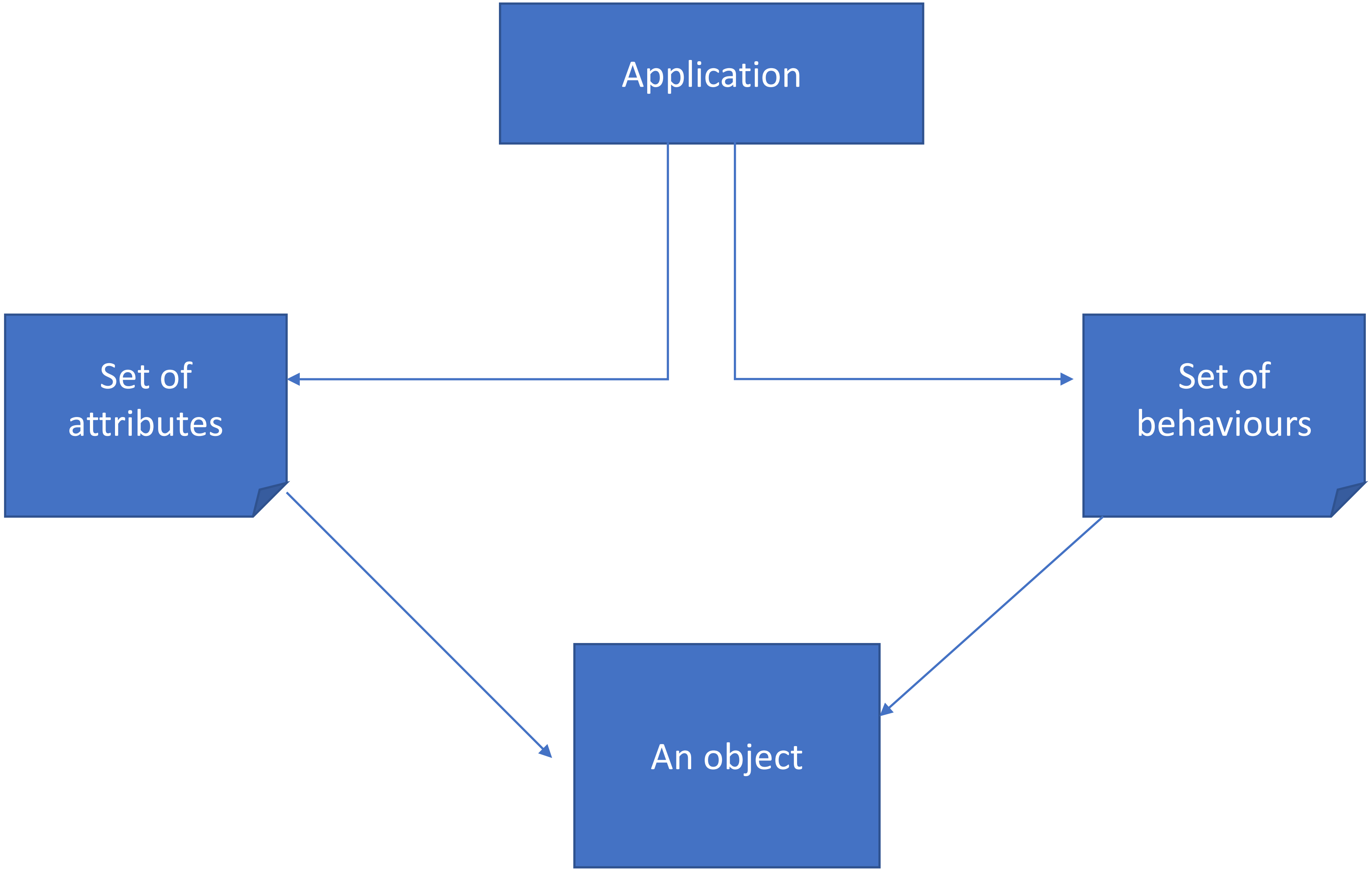
A State models a situation in the execution of a StateMachine Behavior during which some invariant condition holds. In most cases, this condition is not explicitly defined but is implied, usually through the name associated with the State (OMG® UML 2.5.1)

How do you arrive at fixing a state?
Map different scenarios through which an object lifecycle passes into number of states

Different scenarios of Protimer application

- IDLE
- TIME-SET
- COUNTDOWN
- PAUSE
- STAT

These are the different scenarios through which the protimer application object life cycle passes through


Application
Set of attributes
Set of behaviours
An object

State

How to create a state?

1. Draw a round-cornered rectangle
2. Create a horizontal name compartment
3. Give a name that is unique within the state machine diagram

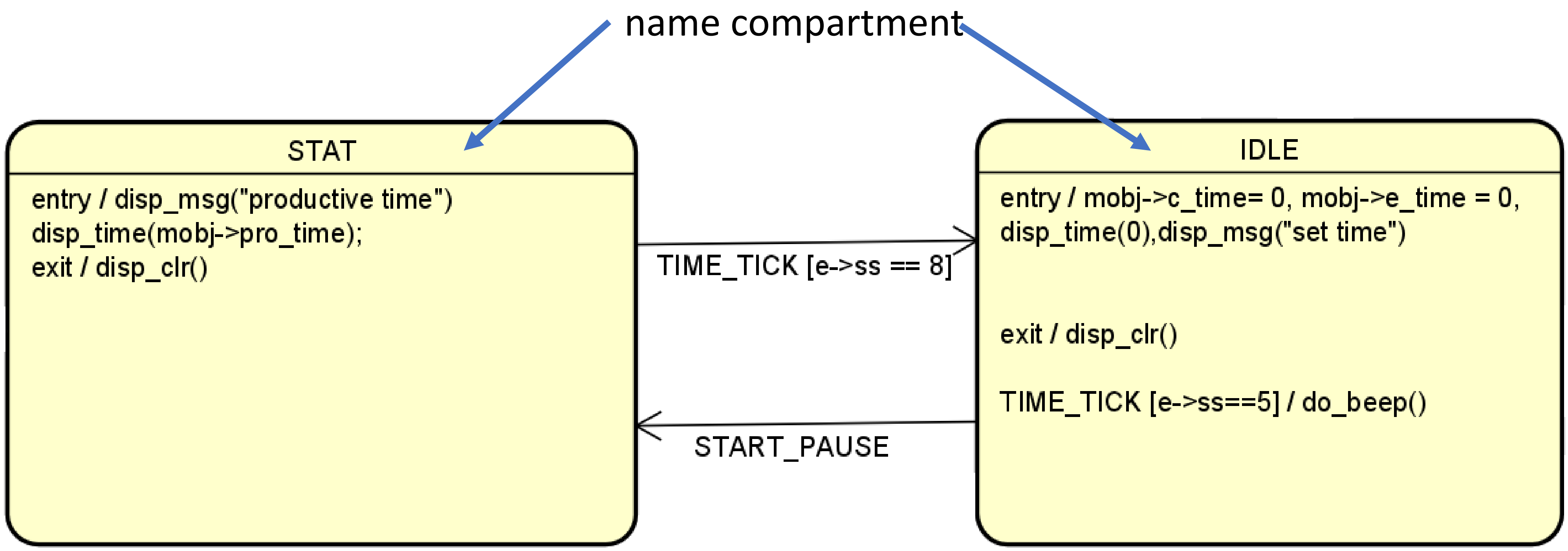
Types of states in UML

- Simple State
- Composite State
- Submachine State

Simple state

If a state doesn’t have any substates, transitions, regions, submachines then it’s a simple state

Simple state with name compartment



A simple state

A simple state

This compartment holds the name of the State as a string.

Internal activities compartment

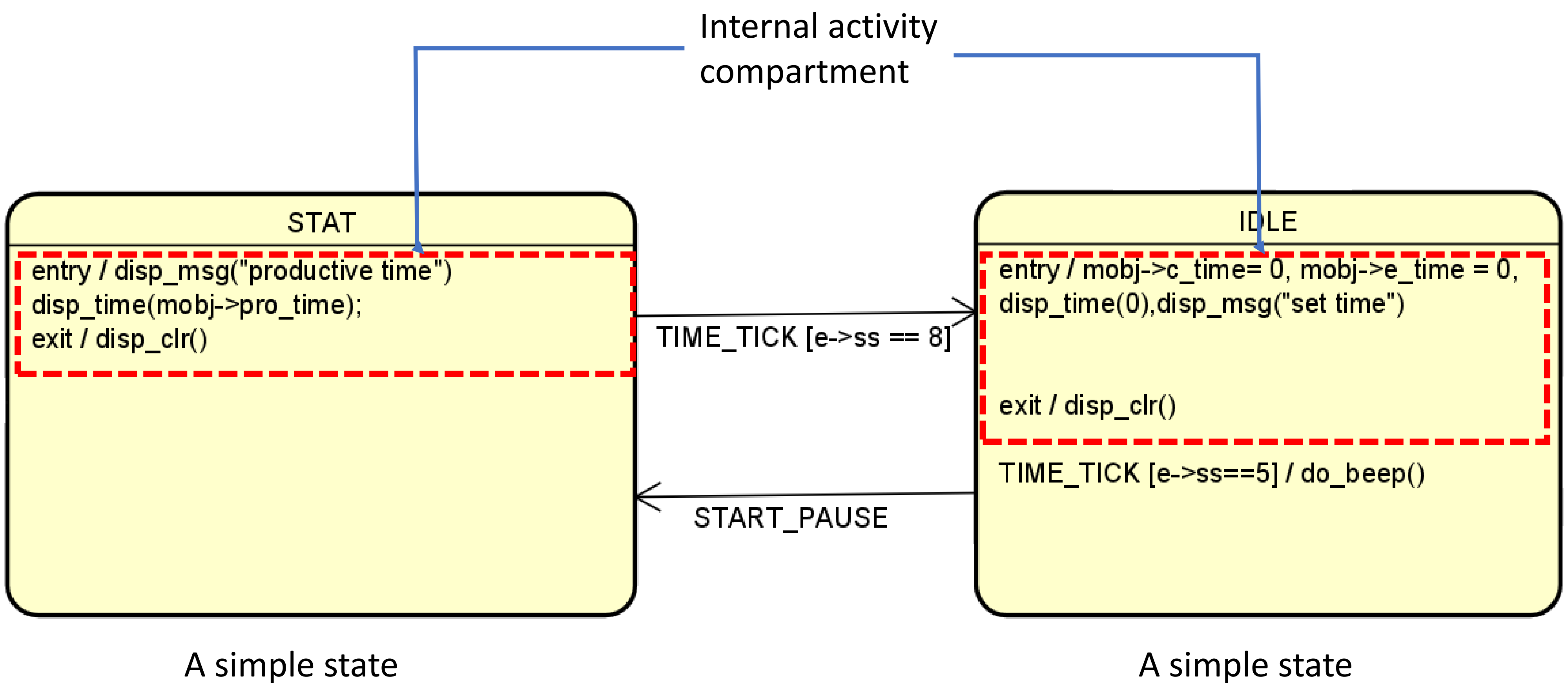
- This compartment holds a list of internal behaviors associated with a state
- Each entry has the following format:
  - *<behavior-type-label>['/' <behavior-expression>]*
- Example :

  'entry,' 'exit', 'do' are internal activities labels or keywords defined in the UML . Do not use these keywords to represent events in the state machine diagram

Internal activity labels

- These labels identify the circumstances under which the behaviors specified by the ‘behavior-expression’ is executed.
- *entry*: Behavior identified by *<behavior-expression>* will be executed upon entry to the state. Use the *entry* keyword if the state has entry action.
- *exit*: Behavior identified by *<behavior -expression>* will be executed upon exit from the state. Use the *exit* keyword if a state has exit action
- *do*: Behavior identified by *<behavior-expression>* will be executed as long as the object is in the state or until the computation specified by the expression is completed. This represents ongoing behavior. Use the *do* keyword only if a state has *do* action

www.fastbitlab.com

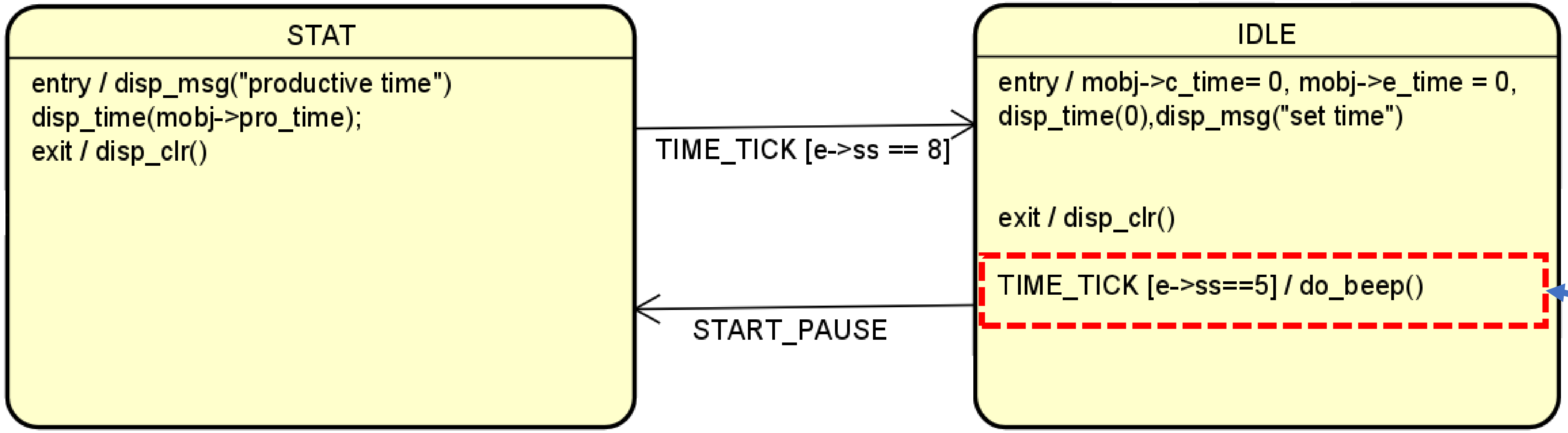


Internal transition

- Each item has the following syntax

  {<trigger>} ['[' <guard>']'] [/<behavior-expression>]*

- If the event occurrence matches the ‘trigger’ and ‘guard’ of the internal transition evaluates to be TRUE, then behavior identified by < *behavior-expression*> will be executed without exiting or re-entering the state in which it is defined.



A simple state

A simple state

Internal transition compartment

Composite state

What is a composite state?

A state which has substates. A composite State contains at least one region

1) Simple composite state
2) Orthogonal state

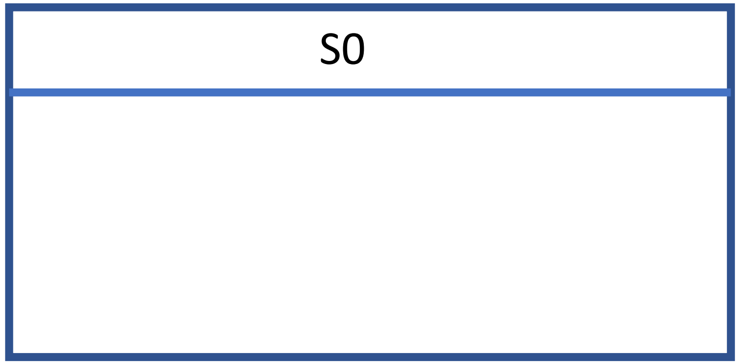
A simple composite state has exactly one region.

- By using composite states, you can express state hierarchies.
- It makes statecharts more comprehensible by reducing the number of transitions between sates

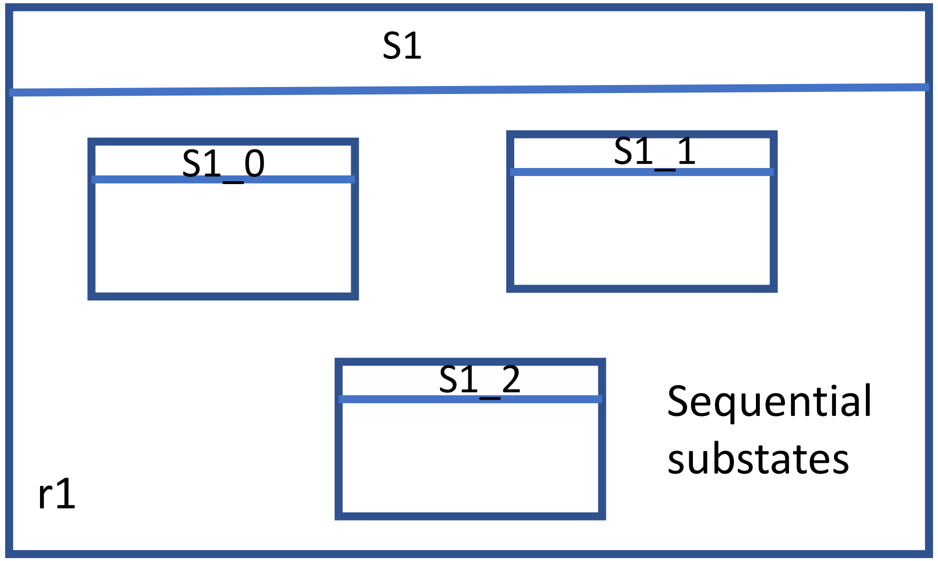
www.fastbitlab.com

Example

Any state enclosed within a region of a composite state is called a substate of that composite state [OMG® UML 2.5.1]

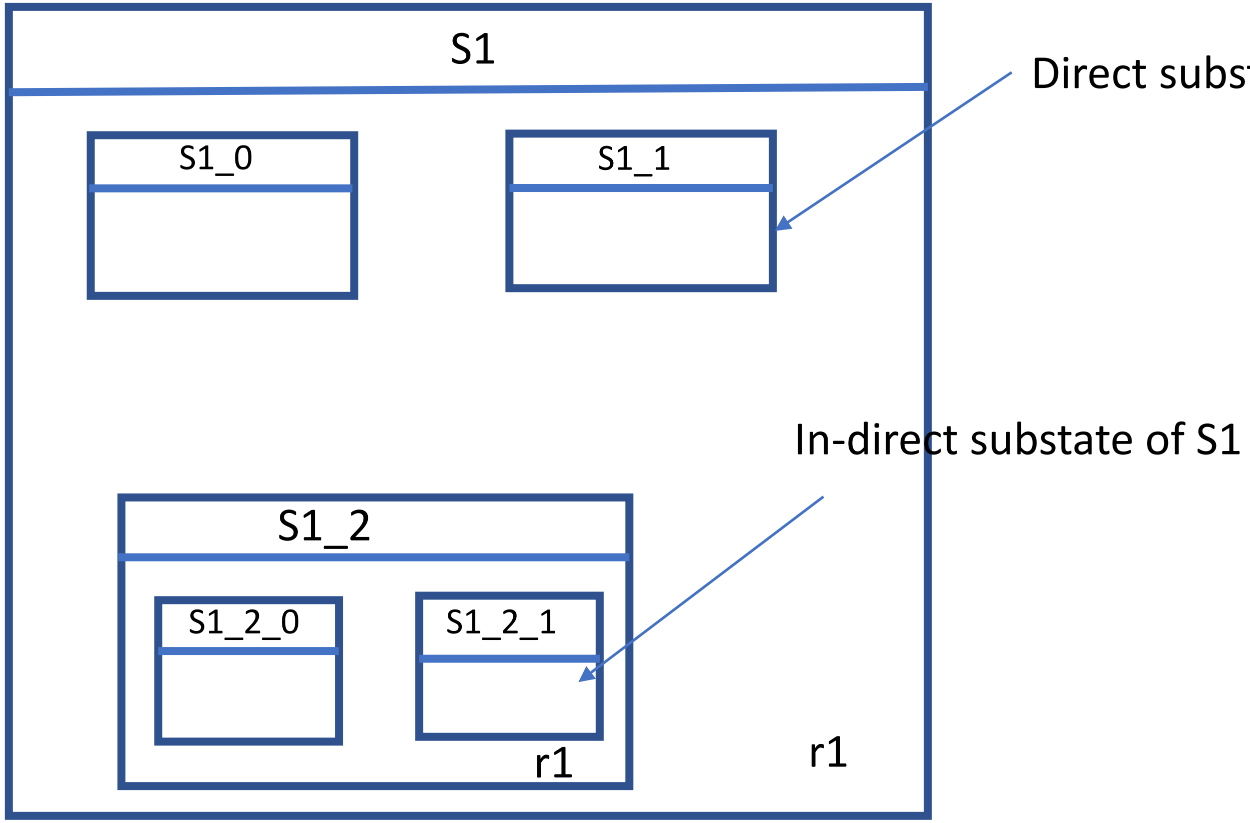


This is a state(simple state)
No regions
No sub states



S1 is a composite state
S1 has 1 region
S1_x are sub states of S1
S1 is a superstate of S1_x
S1_x are simple states

Example



A simple composite state

S1 is a composite state
S1 has 1 region
S1_x are sub states of S1
S1 is a superstate of S1_x
S1_0 ,S1_1,S1_2_0,S1_2_1 are simple states
S1_2 is a composite state
S1_2 is superstate of S1_2_0 and S1_2_1
S1_2 has 1 region

Orthogonal state

- A special case of a composite state.
- A composite state with multiple regions

www.fastbitlab.com

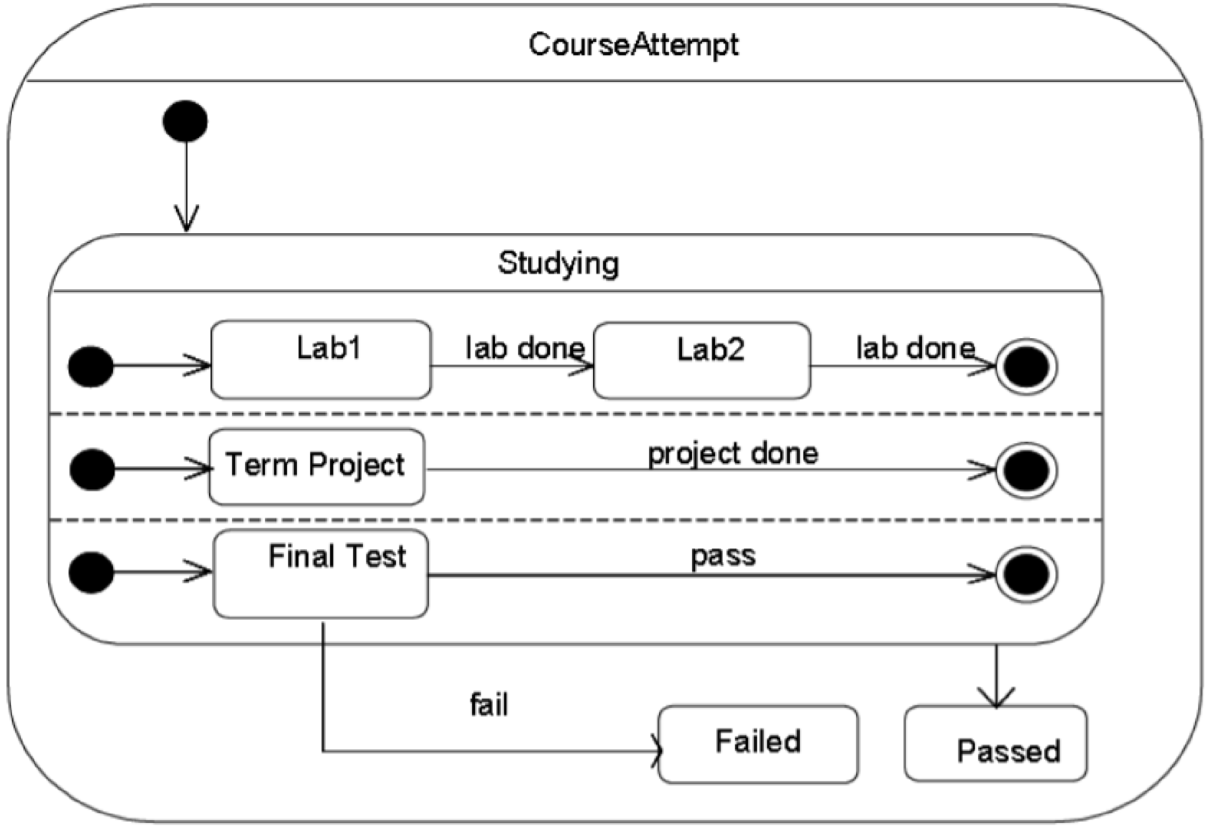


Figure 14.9 Composite State with Regions

[OMG® UML 2.5.1]

Studying is a orthogonal state (composite state having 3 regions) having concurrent substates(not sequential)

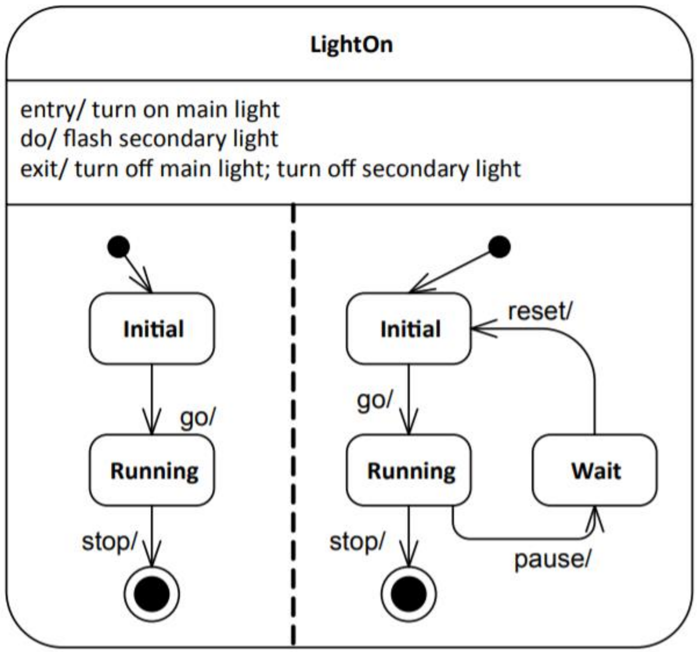


Figure 14.10 Composite State with two Regions and entry, exit, and do Behaviors

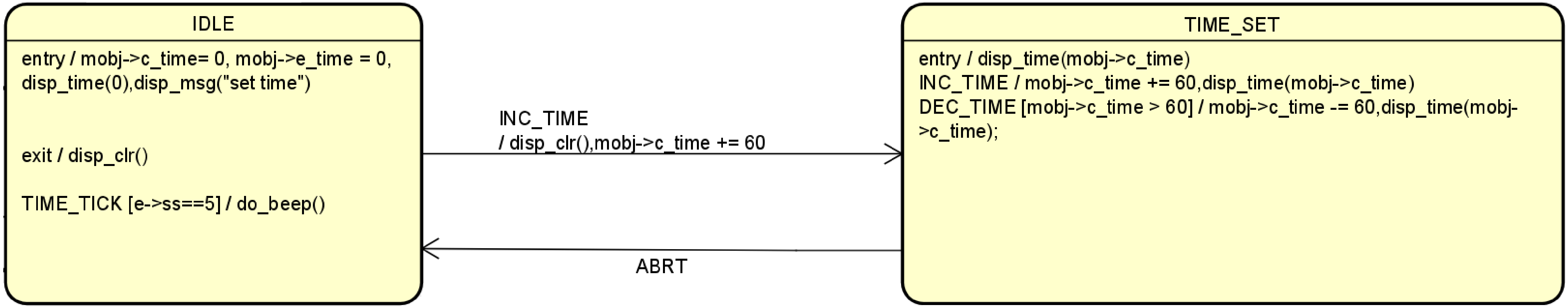
Types of transitions

- External
- Local
- Internal

www.fastbitlab.com

Types of transitions

- External : In external transition the source state is exited due incident of a trigger , the optional action associated with the transition is executed followed by execution of exit action of the state .



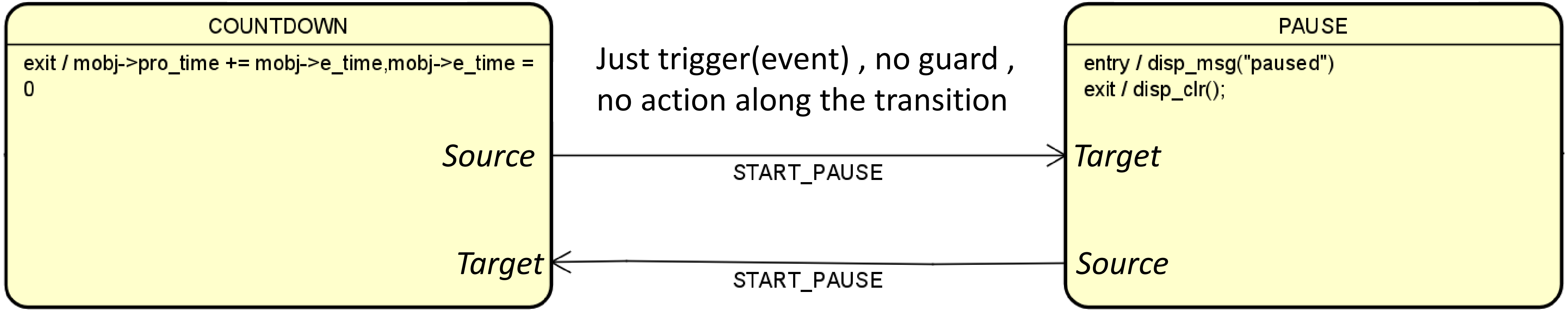
External transition

- In external transition, the source state is exited due incident of a trigger, the optional action associated with the transition is executed followed by execution of exit action of the state
- An external transition signifies a change of state or an object's situation in the object's life cycle.
- When the state is changed, now the object is ready to process a new set of events and execute a new set of actions.
- Transitions are denoted by lines with arrowheads leading from a source state to a target state

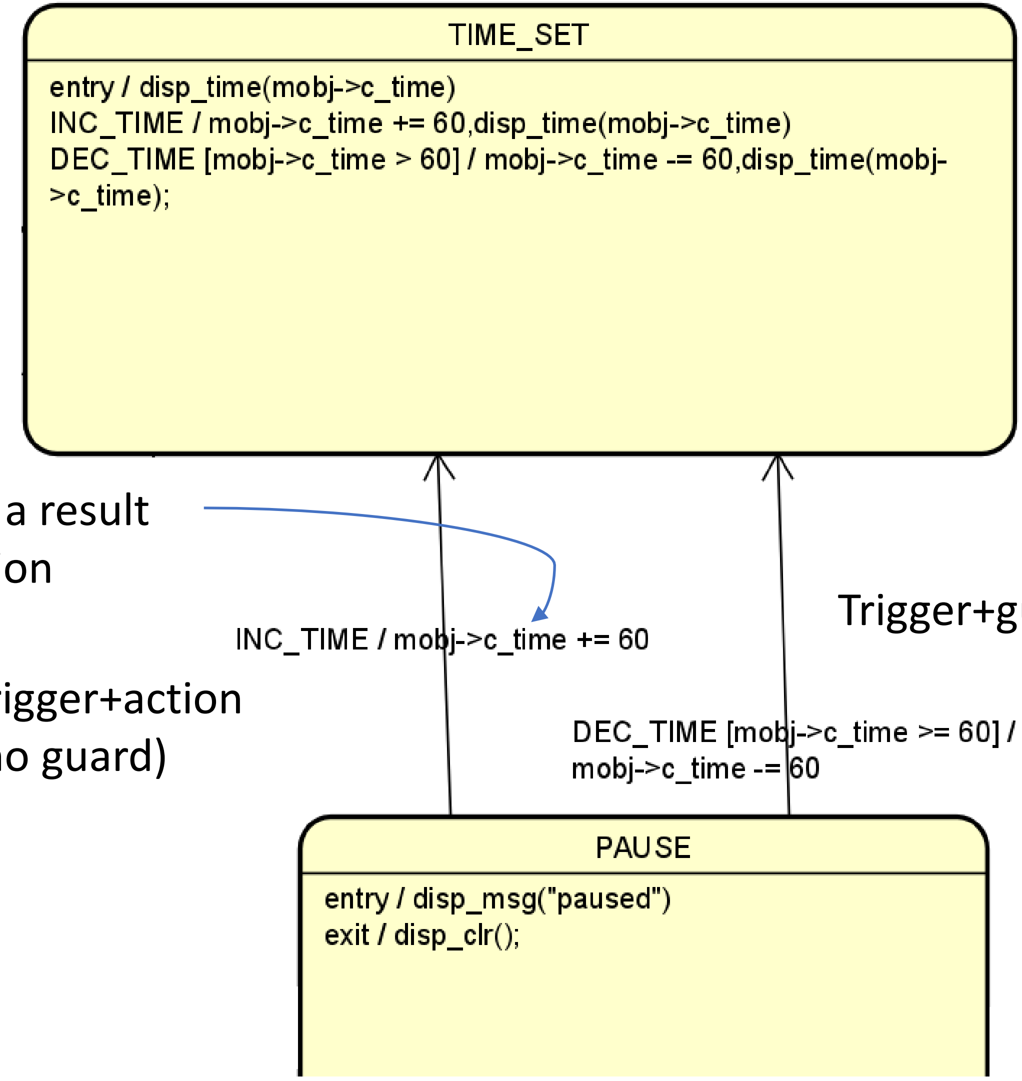
Transition

- Transition syntax :
- {<trigger>}* ['[' <guard>']'] [/<behavior-expression>]
- {<trigger>}[guard]/action
- event[guard]/action

When the current state of the object is COUNTDOWN and if the event START_PAUSE is received then object transitions to state PAUSE (updates its state to PAUSE)



www.fastbitlab.com

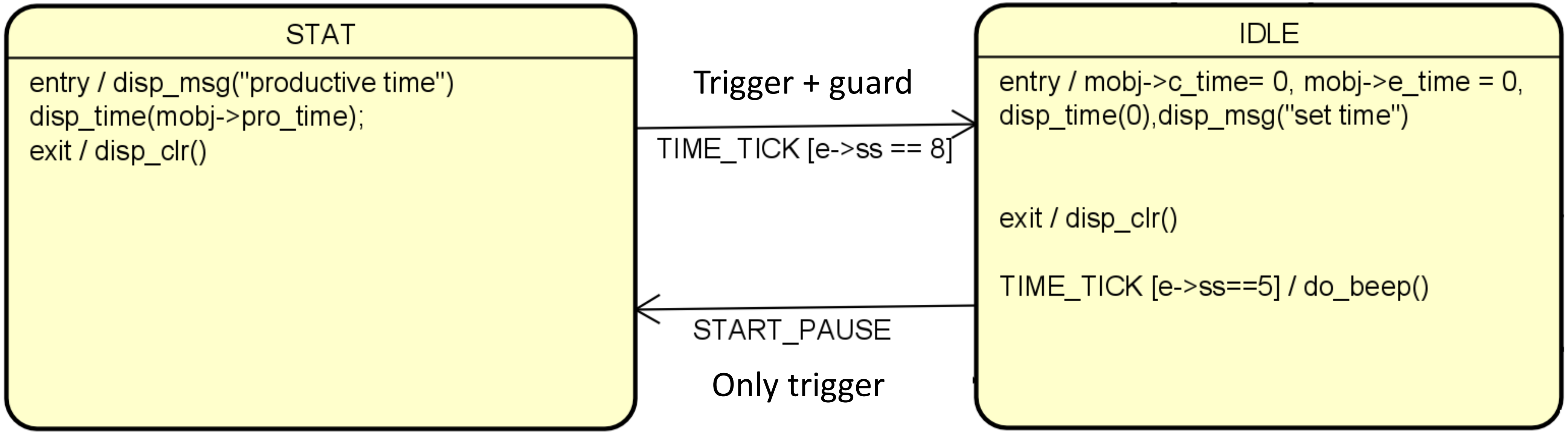


Action as a result of transition

Trigger+action (no guard)

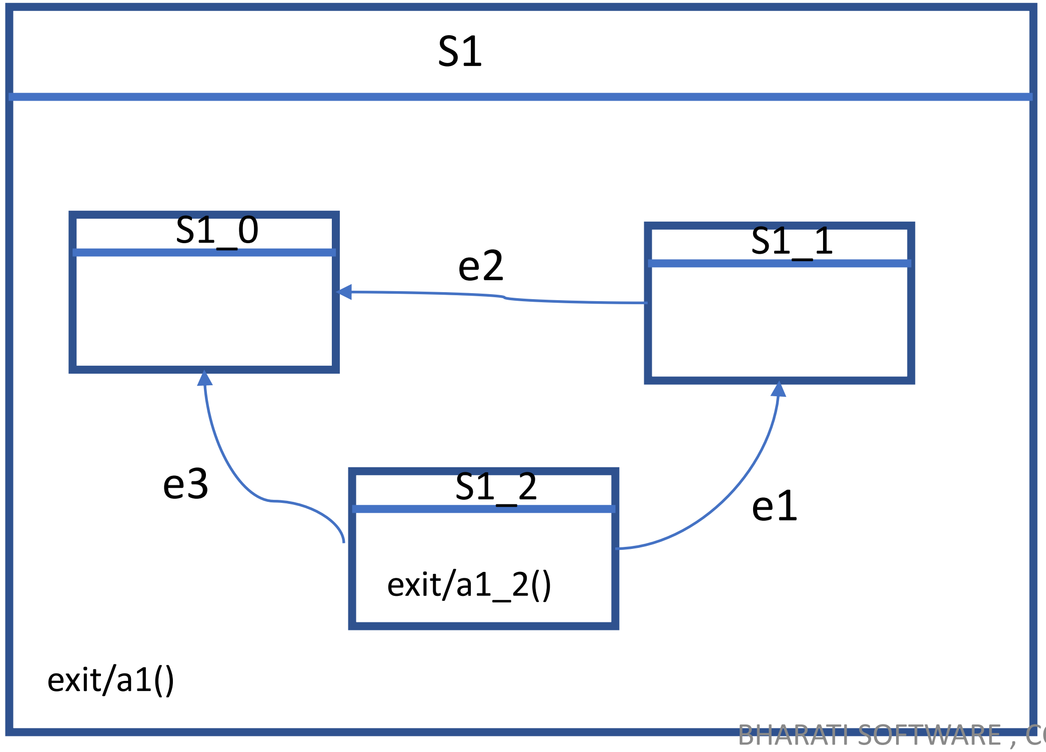
Trigger+guard+action

Guard is a boolean expression and must evaluate to TRUE for transition to fire


STAT
entry / disp_msg("productive time")
disp_time(mobj->pro_time);
exit / disp_clr()
Trigger + guard
TIME_TICK [e->ss == 8]
START_PAUSE
Only trigger
IDLE
entry / mobj->c_time= 0, mobj->e_time = 0,
disp_time(0),disp_msg("set time")
exit / disp_clr()
TIME_TICK [e->ss==5] / do_beep()

Types of transitions

- Local: local is the opposite of external, meaning that the Transition does not exit its containing state (and, hence, the exit Behavior of the containing State will not be executed). However, for local Transitions, the target Vertex must be different from its source Vertex. A local Transition can only exist within a composite State. [OMG® UML 2.5.1]



Set of Triggers

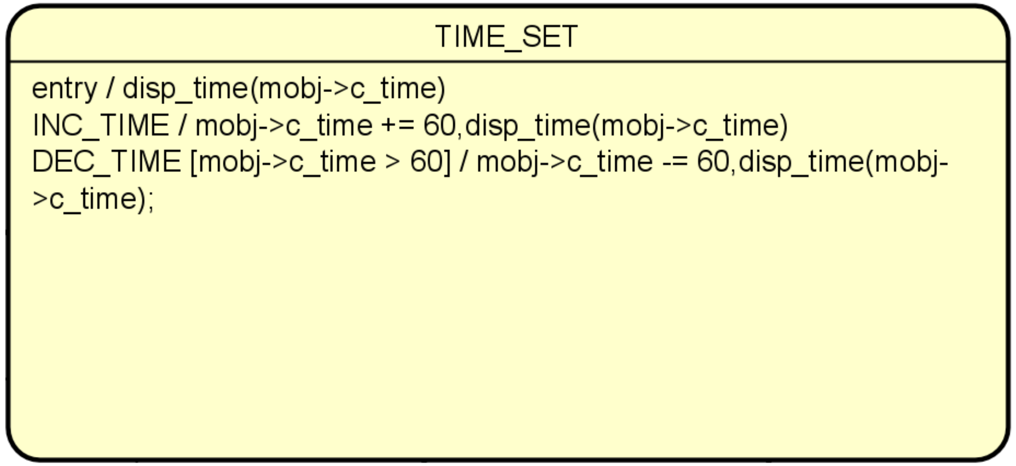
A Transition may own a set of Triggers, each of which specifies an Event whose occurrence, when dispatched, may trigger traversal of the Transition. A Transition trigger is said to be *enabled* if the dispatched Event occurrence matches its Event type. When multiple triggers are defined for a Transition, they are logically disjunctive, that is, if *any* of them are enabled, the Transition will be triggered.

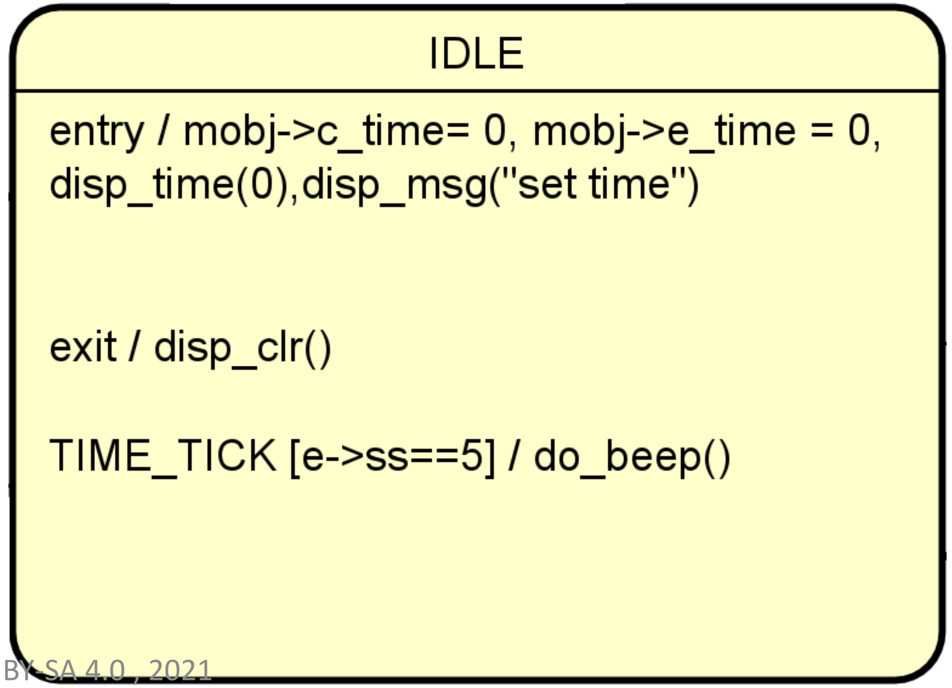
[OMG® UML 2.5.1]

e1,e2,e3[a > 0] / action_1()

Types of transitions

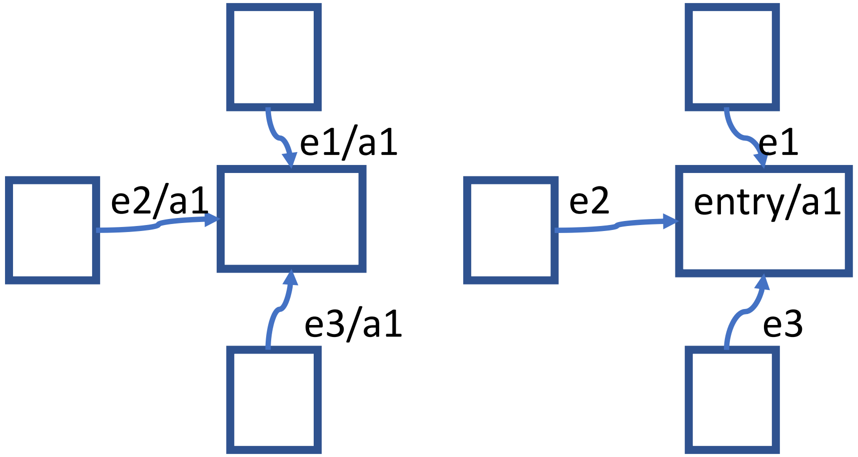
Internal : internal is a special case of a local Transition that is a self-transition (i.e., with the same source and target States), such that the State is never exited (and, thus, not re-entered), which means that no exit or entry Behaviors are executed when this Transition is executed. This kind of Transition can only be defined if the source Vertex is a State [OMG® UML 2.5.1]



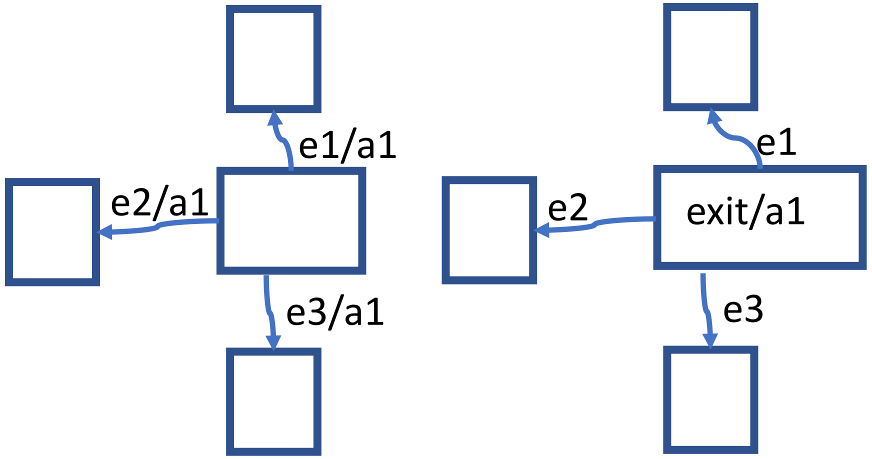


Few points to remember

- doActivity behavior commences execution when the state is entered only after the state entry behavior has completed.
- entry , exit , do , cannot be associated with any transitions



If all transitions into a state perform the same action, push the common action inside the state to make it an entry action.



If all transitions leaving a state perform the same action, push the common action inside the state to make it an exit action.

Events(Trigger)

What is an event?

Events are incidents or a stimulus by which a state machine can be triggered; Incidents are abstracted as events

In state machine events cause transitions(external or internal)

Incidents in the operation of a microwave oven

1)Opening door; Heater off , lights on

2)Closing door; lights off

3)Set timer;   manage time

4)Start ; heater ON

Events

An event usually has 2 components

1) Signal

2) one or more associated values or parameters (optional)

event

sig: INC_TIME

parameters:
--

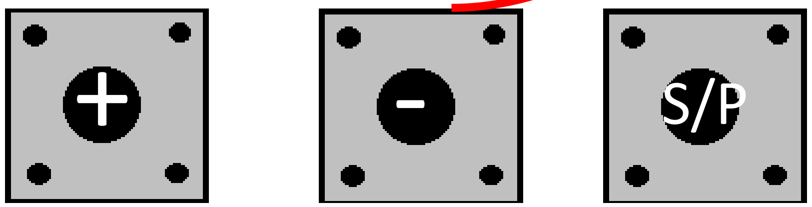
This event signifies by its signal attribute that the user has pressed a button which increases the time

The signal doesn’t require associated parameters.

event

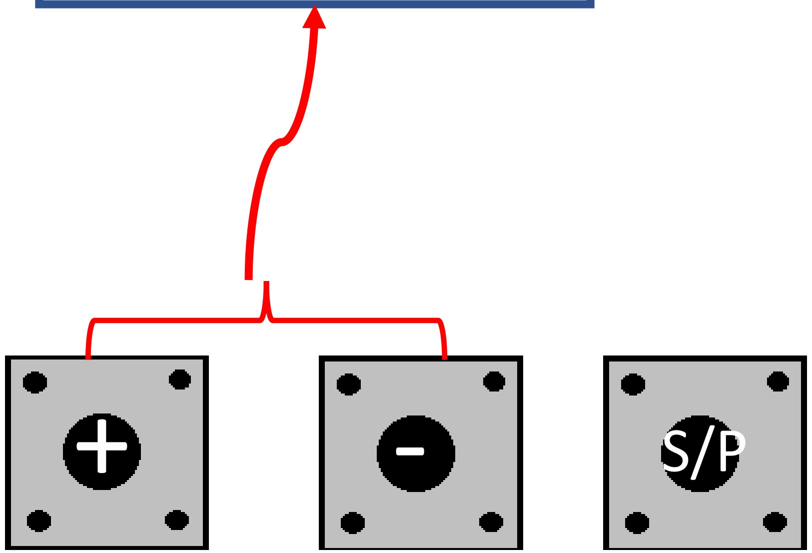
sig: DEC_TIME

parameters:
--



event

sig: TIME_CHANGE
parameters: dir Direction_t




```
enum
{
   UP,
   DOWN
} Direction_t
```

- This event signifies by its signal attribute that the user has pressed a button that changes time.
- The signal has an associated parameter that encodes whether the user has pressed a button that increases time(UP) or decreases time(DOWN)

event

sig: DIGI_0_9

parameters:
digit   uint8_t

www.fastbitlab.com

event

sig: OPER
parameters: type uint8_t

Exercise 2 : Productivity Timer(ProTimer)

The application that tracks your productive working time

Project requirements

- + button → Increment time (minutes should increase)
- – button → Decrement time (minutes should decrease)
- S/P button → Start/pause the countdown; show STAT
- When the countdown is paused, time can be modified.
- Press the + and – button simultaneously to abort the running timer
- Application must beep 20 times when it returns to IDLE mode
- When the application is in IDLE mode, pressing the S/P button should show the STAT for 1 sec and auto return to IDLE mode.

States

Map different situations of your project into states

1) IDLE
2) TIME SET
3) PAUSE
4) COUNTDOWN
5) STAT

Events

User activity	Event generated : SIGNAL	Parameters	note
Press ‘+’ button	INC_TIME	none	This event gets posted to the state machine whenever the user presses the + button
Press - button	DEC_TIME	none	This event gets posted to the state machine whenever the user presses the - button
Press S/P button	START_PAUSE		This event gets posted to the state machine whenever the user presses the S/P button
Press + and – button together	ABRT		This event gets posted to the state machine whenever the user presses the + and – buttons together
	TIME_TICK	ss (sub second)	This event is system generated for every 100ms ss parameter value can vary between 1 to 10 1 → 100ms

Data structures

- Define various signals of the application using enum
- Define various states of the application using enum
- Define the main application structure
- Define structures to represent events

Signals and States

- Add this to “main.h”

```
/* Signals of the application*/
typedef enum{
    INC_TIME,
    DEC_TIME,
    TIME_TICK,
    START_PAUSE,
    ABRT,
/* Internal activity signals */
    ENTRY,
    EXIT
}protimer_signal_t;
```

```
/* Various States of the application*/
typedef enum{
    IDLE,
    TIME_SET,
    COUNTDOWN,
    PAUSE,
    STAT
}protimer_state_t;
```

Main application structure

```c
/* Main application structure */
typedef struct {
    uint32_t curr_time;
    uint32_t elapsed_time;
    uint32_t pro_time;
    protimer_state_t active_state;
}protimer_t;
```

Events

```
/* For user generated events */
typedef struct{
    uint8_t sig;
}protimer_user_event_t;
```

```
/* For tick event */
typedef struct{
    uint8_t sig;
    uint8_t ss;
}protimer_tick_event_t;
```

Events

```c
/*Generic(Super) event structure */
typedef struct{
    uint8_t sig;
}event_t;

/* For user generated events */
typedef struct{
    event_t super;
}protimer_user_event_t;

/* For tick event */
typedef struct{
    event_t super;
    uint8_t ss;
}protimer_tick_event_t;
```

Derived from

Implementation of state machine

- Nested switch approach
- State table approach
- State handler approach

Extended state variables

curr_time : uint32_t

elapsed_time : uint32_t

pro_time : uint32_t

```
/* the main application structure */
typedef struct protimer_tag
{
  uint32_t curr_time;
  uint32_t elapsed_time;
  uint32_t pro_time;
   .
   .
   .
}protimer_t;
```

Data structure of events

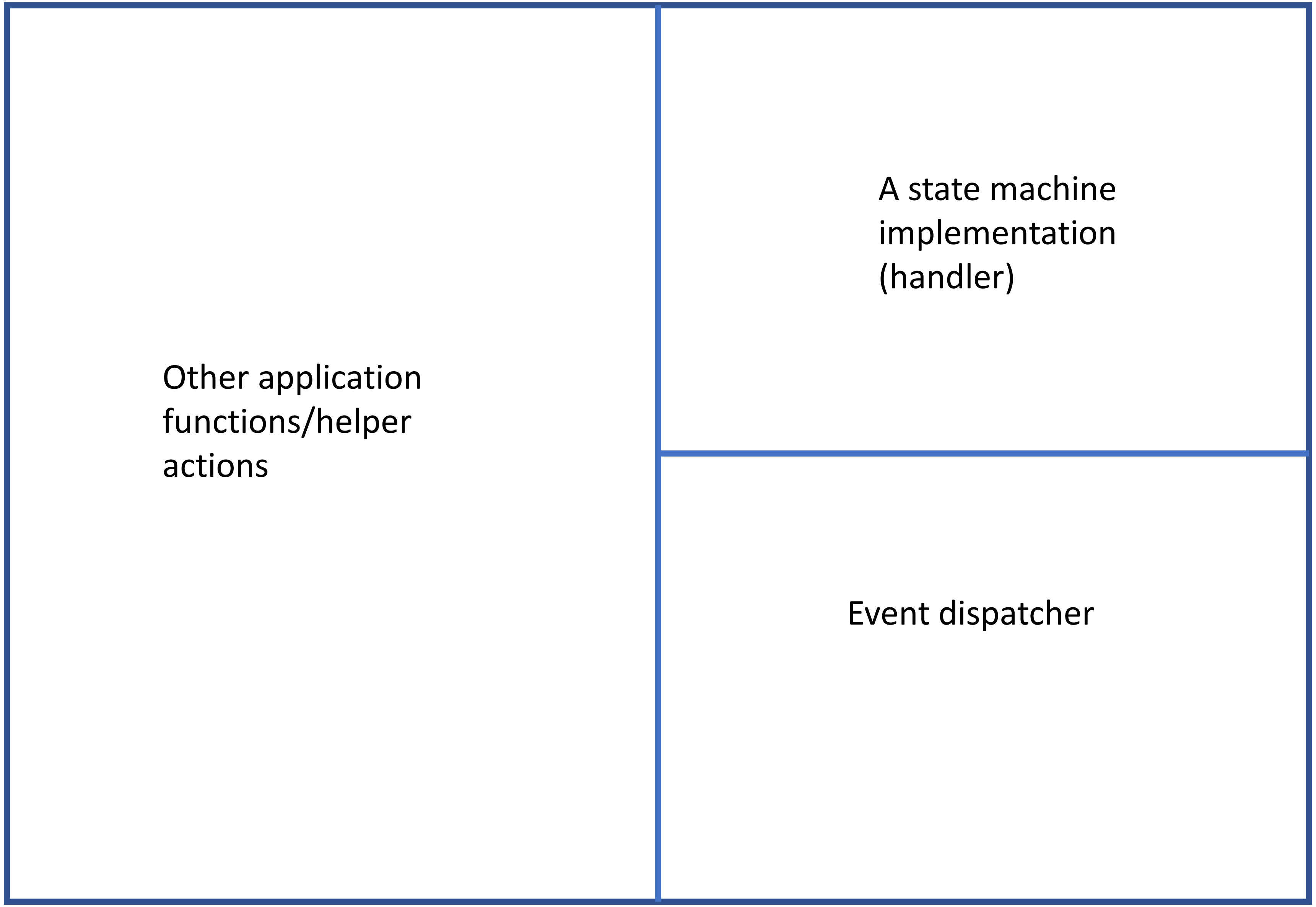
```
typedef struct{
  uint8_t sig;
}ptimer_user_event_t;

typedef struct{
  uint8_t sig;
  uint8_t ss; /*sub-second*/
}ptimer_tick_event_t;
```

```
enum ptimer_signal
{
   INC_TIME,
   DEC_TIME,
   TIME_TICK,
   START_PAUSE,
   ABRT,
   /*internal activity signals*/
   ENTRY,
   EXIT
};
```

STEP1

- Create new Arduino project
- Save the project as ptimer (this also creates ptimer.ino source file)
- create new application C header file ptimer.h
- Add all so far discussed data structures to ptimer.h

Application



STEP2 : State machine handler

- It is just a C function
- It receives events and takes actions
- It also operates on the extended state variables

Generic event structure

```
/*
 * Generic event structure to hold\
 * signal part of the event
 */
typedef struct event_tag{
  uint8_t sig;
}event_t;
```

Implementation of state machine handler

- Nested switch
- State table approach
- State handler approach

Nested switch

```
event_status ptimer_state_machine(ptimer_t *mobj, event_t *e)
{
  switch(mobj->active_state)
  {
    case IDLE:{
        return IDLE_state_handler(mobj,e);
    }
    case TIME_SET:{
      return TIME_SET_state_handler(mobj,e);
    }
    .
    .
    .
    .
  }

  return EV_IGNORED;
}
```

Event handling status

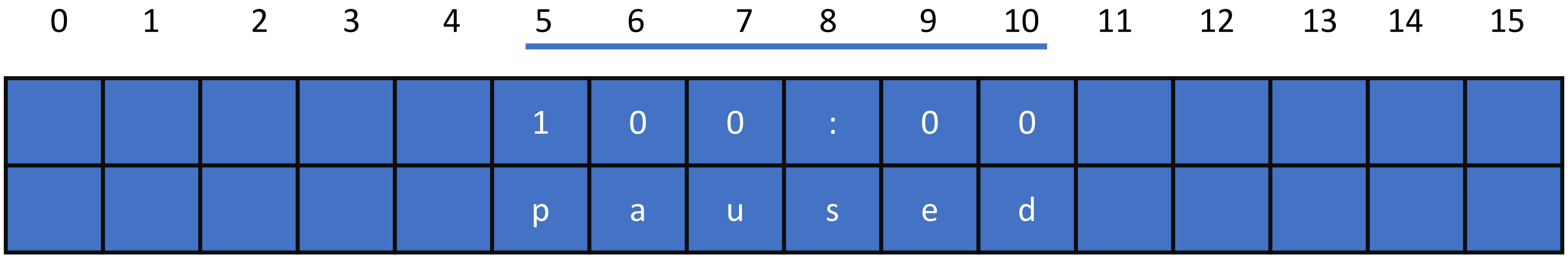
```
enum  event_status
{
   EV_HANDLED   ,
   EV_IGNORED   ,
   EV_TRANSITION
};
```

TODO STEP2:

- Create new Arduino source file ptimer_sm
- Implement state machine handler discussed

	column 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	column 15
row 0	s	e	t			0	0	0	:	0	0					
row 1	t	i	m	e												

16x2 LCD



16x2 LCD

STAT

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
					1	0	0	:	0	0					
	P	r	o	d	u	c	t	l	v	e	t	l	m	e	

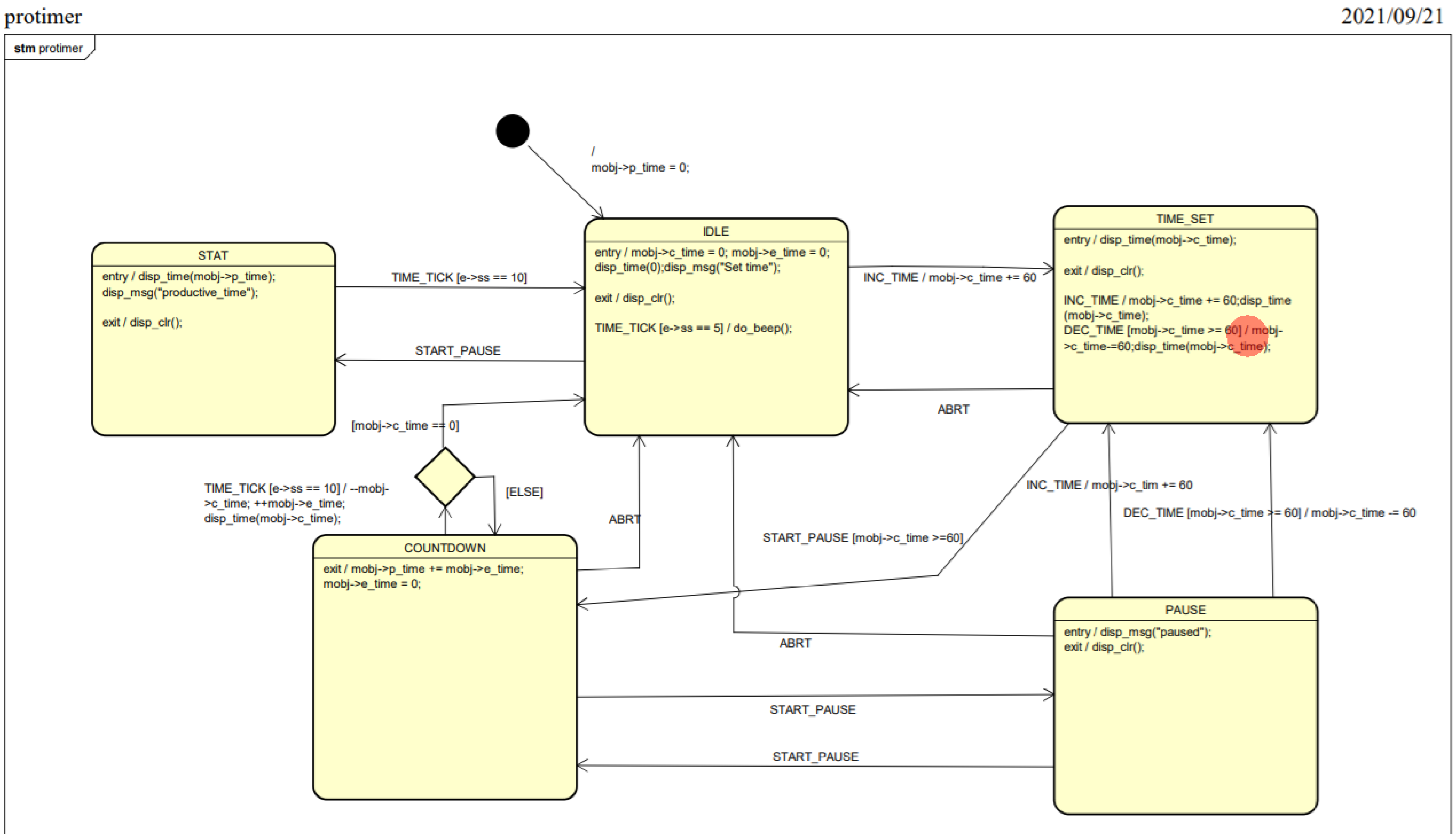
16x2 LCD

State table approach

state	Event						
	INC_MIN	DEC_MIN	START_PAUSE	ABRT	TIME_TICK	ENTRY	EXIT
INIT	Init_inc_min() NS : TIME_SET	x	Init_start_pause() NS: STAT	x	Init_time_tick()	Init_entry()	init_exit( )
TIME_SET	Time_set_inc _min()	Time_set_Dec _min	Time_set_start_pau se() NS: COUNTDOWN	Time_set_ab rt() NS:INIT	x	Time_set_ent ry()	x
COUNTDOWN	X	X	Countdown_start_p ause() NS:PAUSE	Countdown_ abrt() NS:INIT	Countdown_time _tick()	x	Countdo wn_Exit( )
PAUSE	Pause_inc_mi n()	Pause_Dec_mi n()	Pause_start_pause() NS:COUNTDOWN	Pause_abrt() NS:INIT	x	Pause_entry( )	Pause_E xit()
STAT	X	X	x	x	Stat_time_tick()	Stat_entry()	Stat_exit ()

www.fastbitlab.com

state entry and state exit Behaviors



Pseudo states

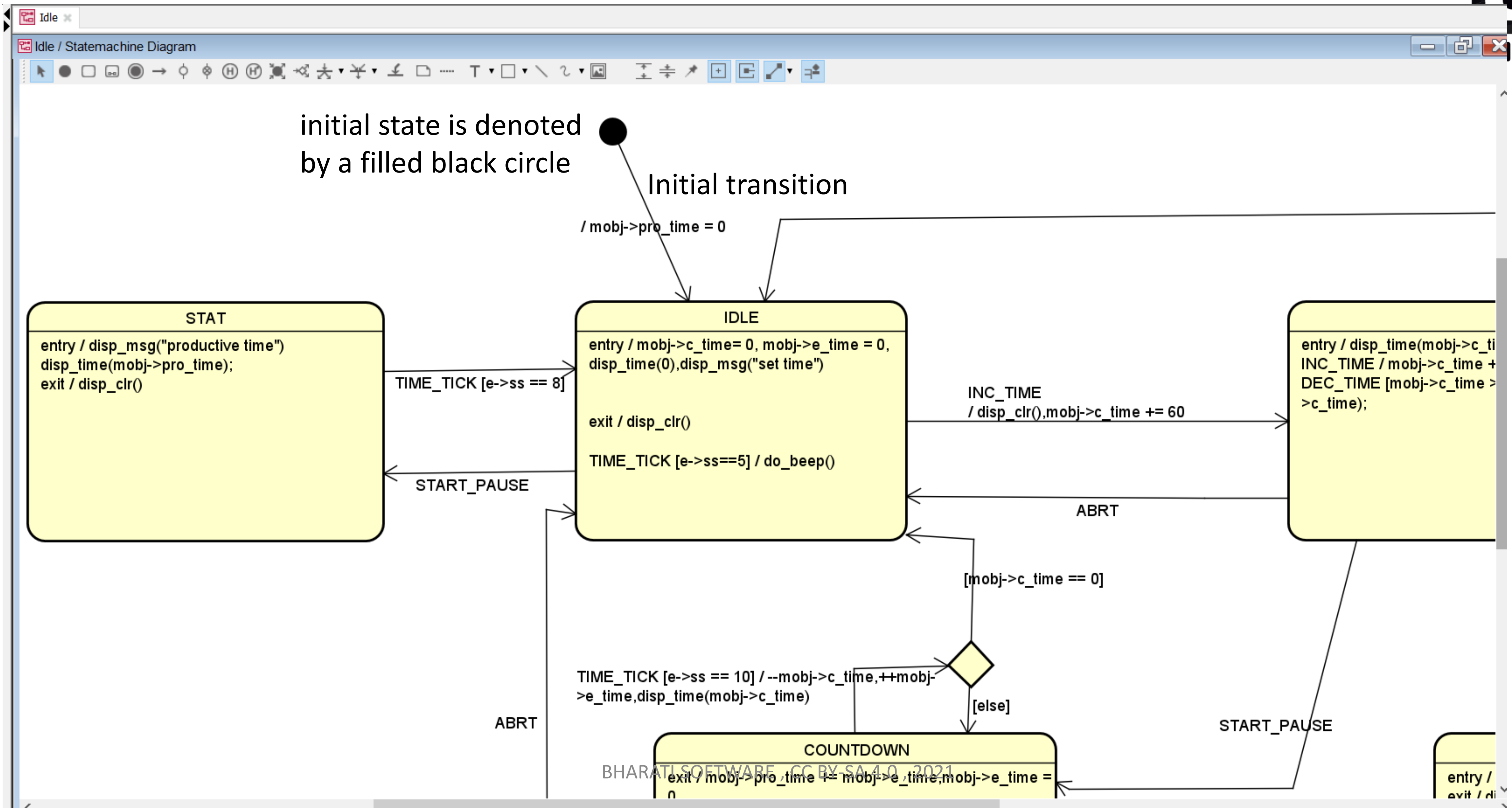
- Initial
- Choice
- Join
- Deep history
- Shallow history

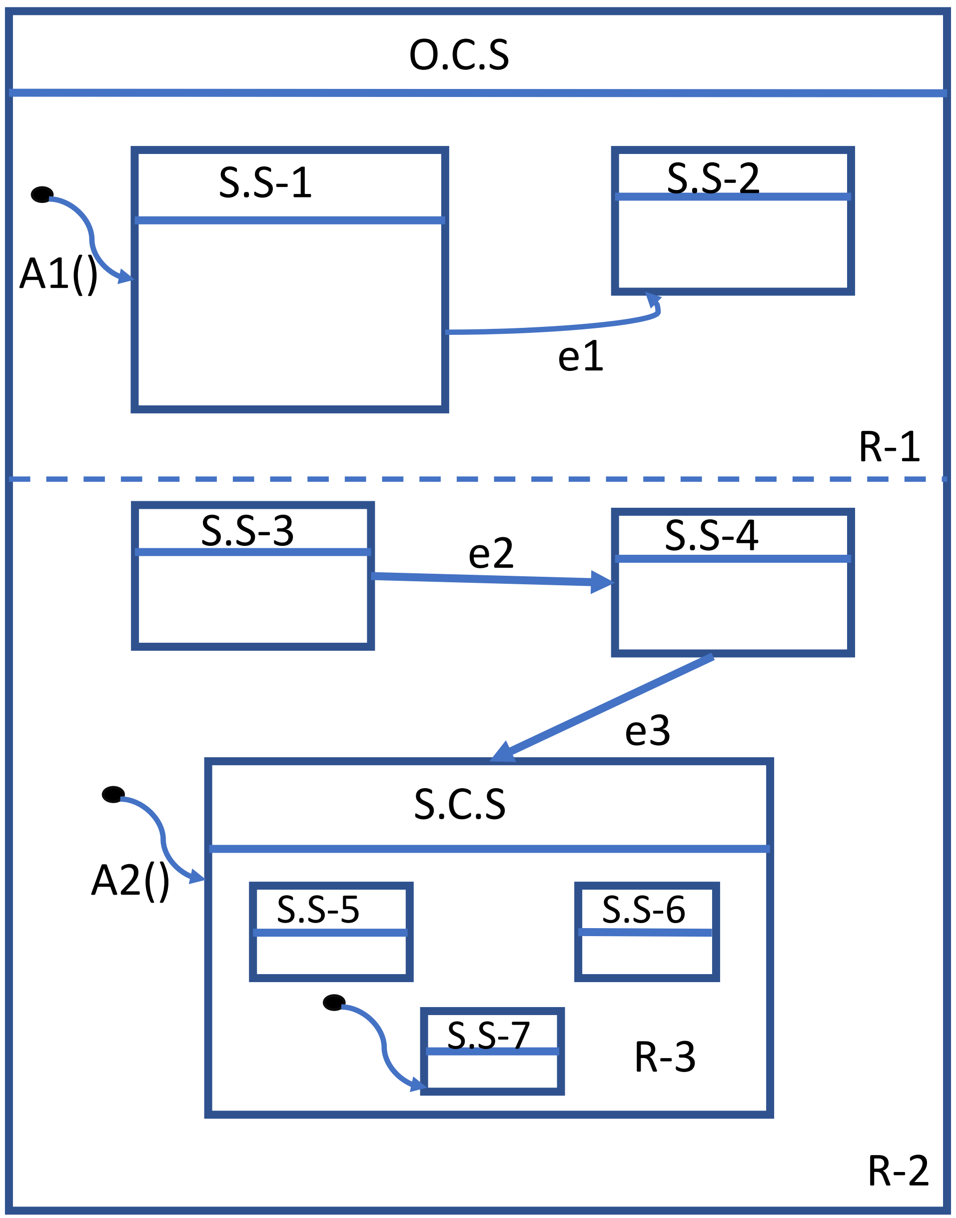
Refer [OMG® UML 2.5.1] for complete list of pseudo states

[OMG® UML 2.5.1]

initial - An **initial** Pseudostate represents a starting point for a Region; that is, it is the point from which execution of its contained behavior commences when the Region is entered via default activation. It is the source for at most one Transition, which may have an associated effect Behavior, but not an associated trigger or guard. There can be at most one **initial** Vertex in a Region.

Pseudo state : Initial

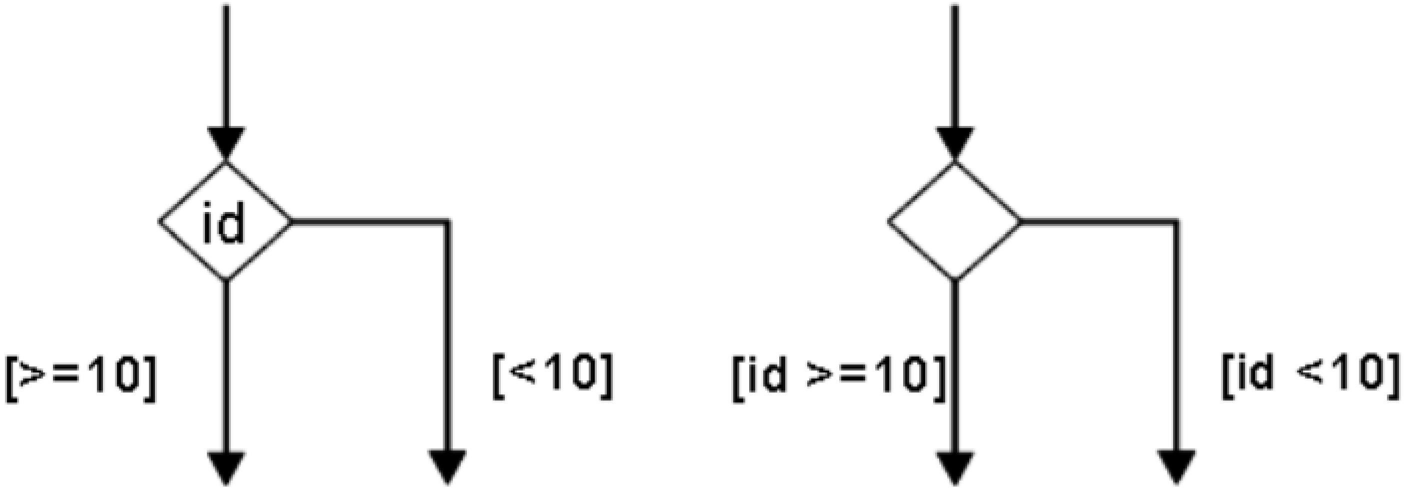



O.C.S
S.S-1
A1()
S.S-2
e1
R-1
S.S-3
e2
S.S-4
e3
S.C.S
A2()
S.S-5
S.S-6
S.S-7
R-3
R-2

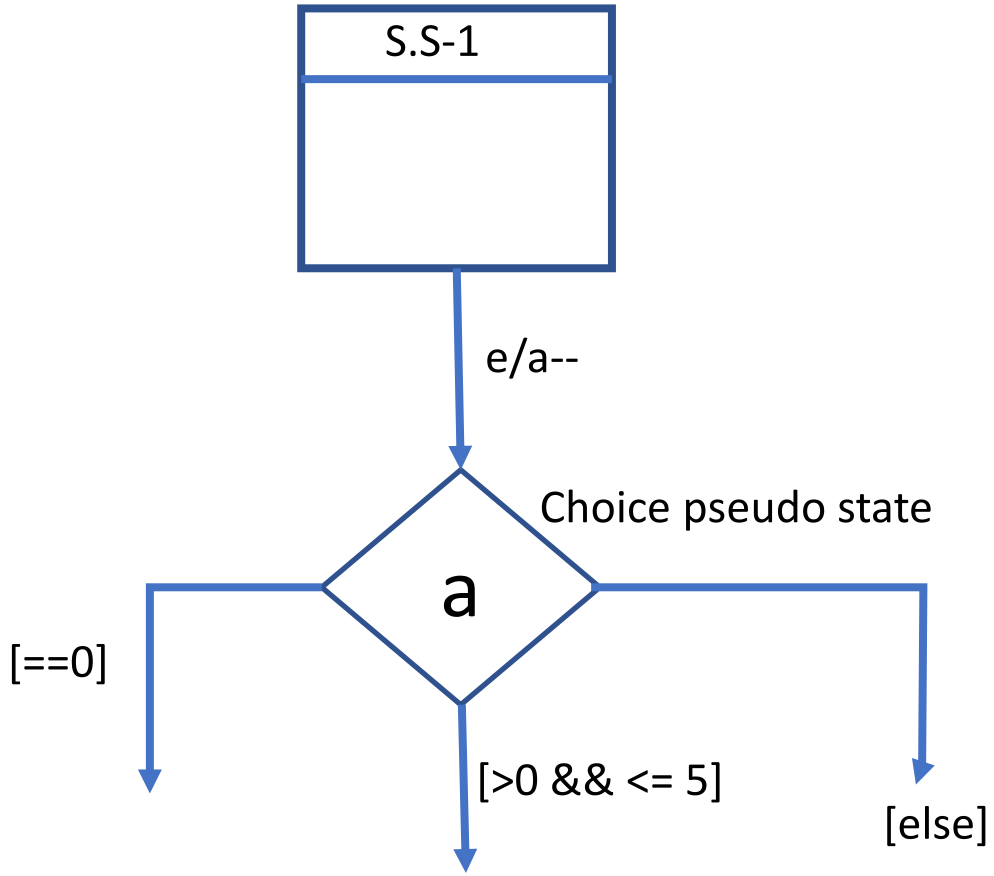
- In our example , the initial transition is associated with a transition to IDLE state.
- When we create the application object, manually we should call a function which takes care of executing the initial transition action and set the initial state as IDLE before processing any events .

Choice

A choice Pseudostate is shown as a diamond-shaped symbol



It has a single incoming transition and two or more outgoing transitions



[OMG® UML 2.5.1]

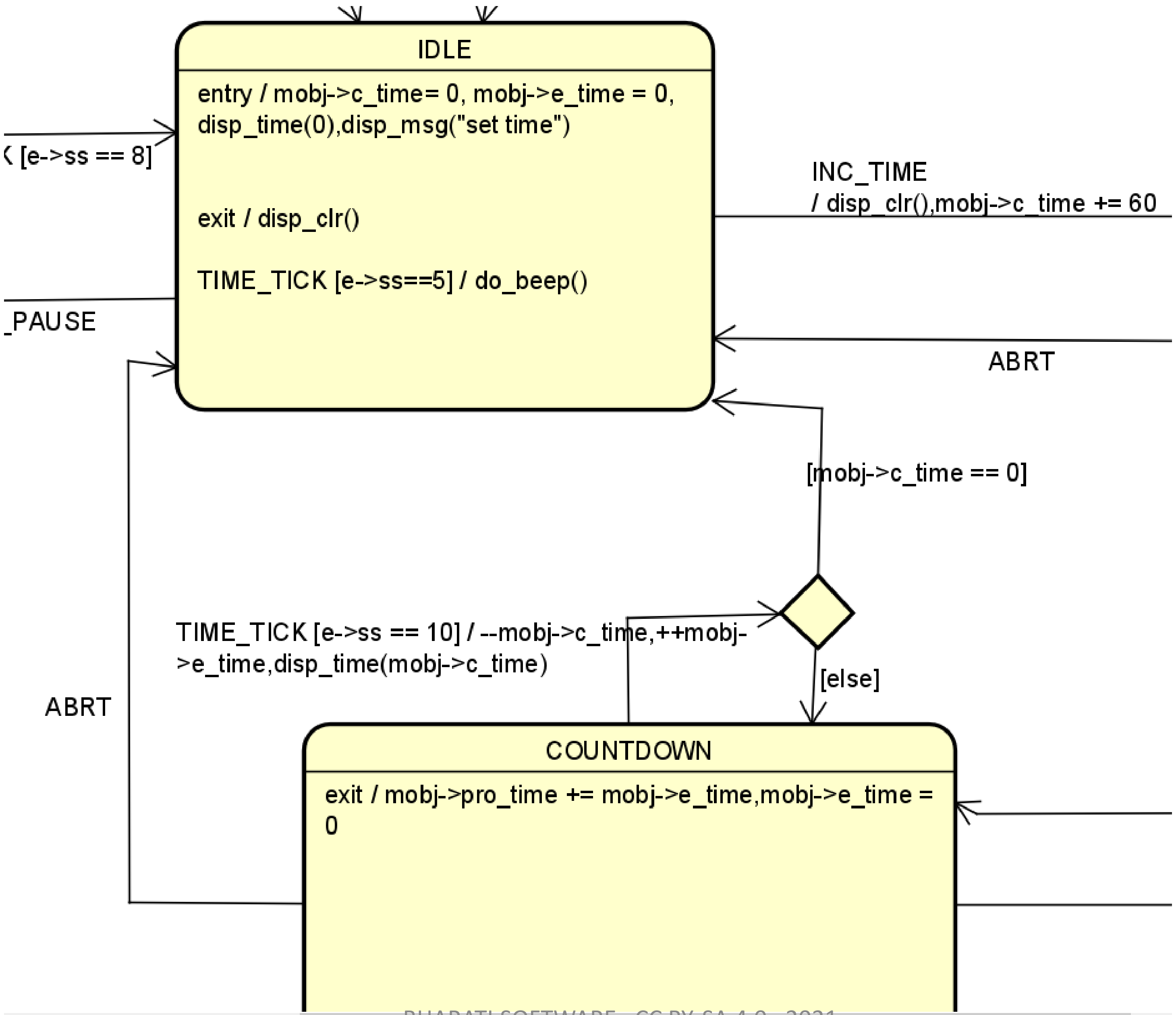
choice is used to realize a dynamic conditional branch. It allows splitting of compound transitions into multiple alternative paths such that the decision on which path to take may depend on the results of Behavior executions performed in the same compound transition prior to reaching the choice point

If none of the guards evaluates to true, then the model is considered ill formed

If more than one guard evaluates to true, one of the corresponding Transitions is selected

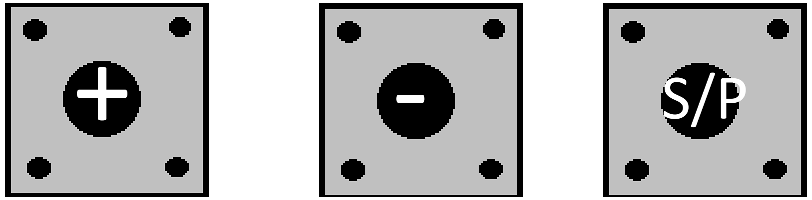
it is recommended to define one outgoing Transition with the predefined “else” guard for every choice Pseudostate

www.fastbitlab.com



Arduino and Button connections

Component	Arduino Pin number
Button 1 (INC_TIME)	2
Button 2 (DEC_TIME)	3
Button 3 (START/PAUSE)	4
Buzzer	12



1   2   3

Button pad truth table

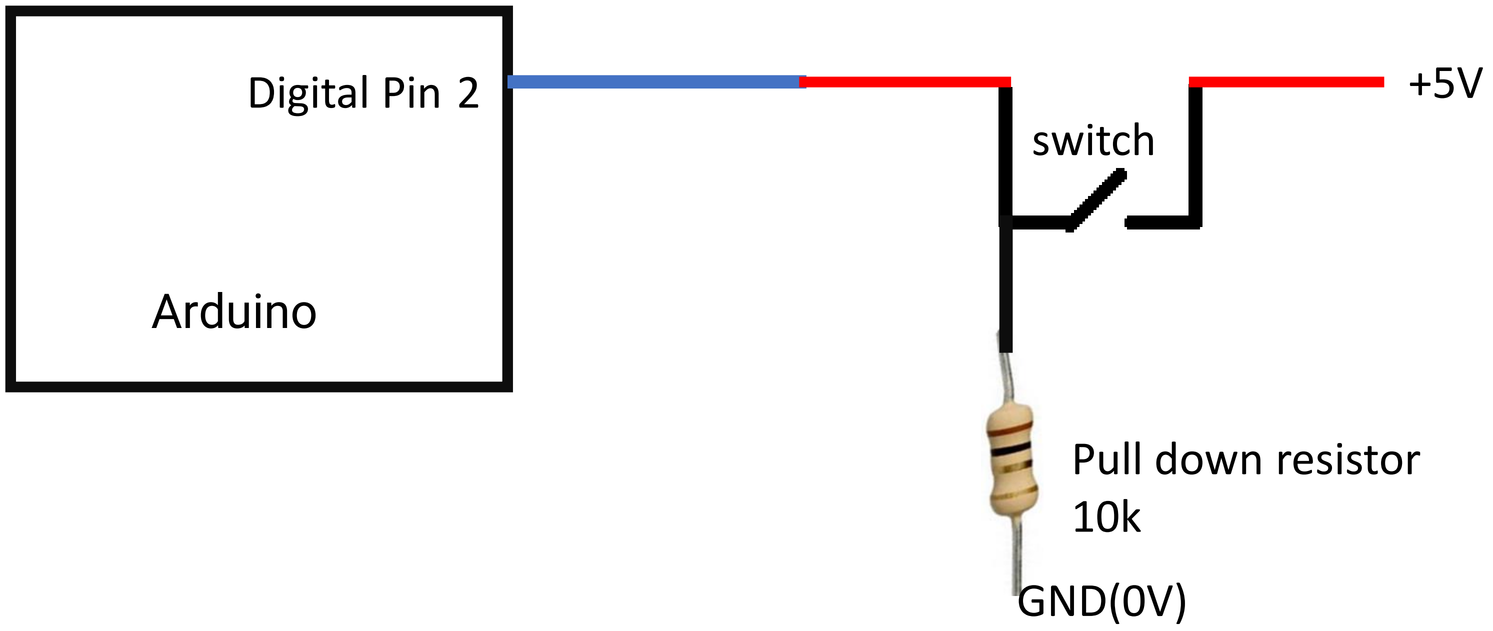
B1	B2	B3	Value	Signal
0	0	1	1	START_PAUSE
0	1	0	2	DEC_TIME
1	0	0	4	INC_TIME
1	1	0	6	ABRT
Other values				XX

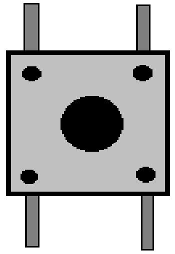
Arduino and 16x2 LCD connection

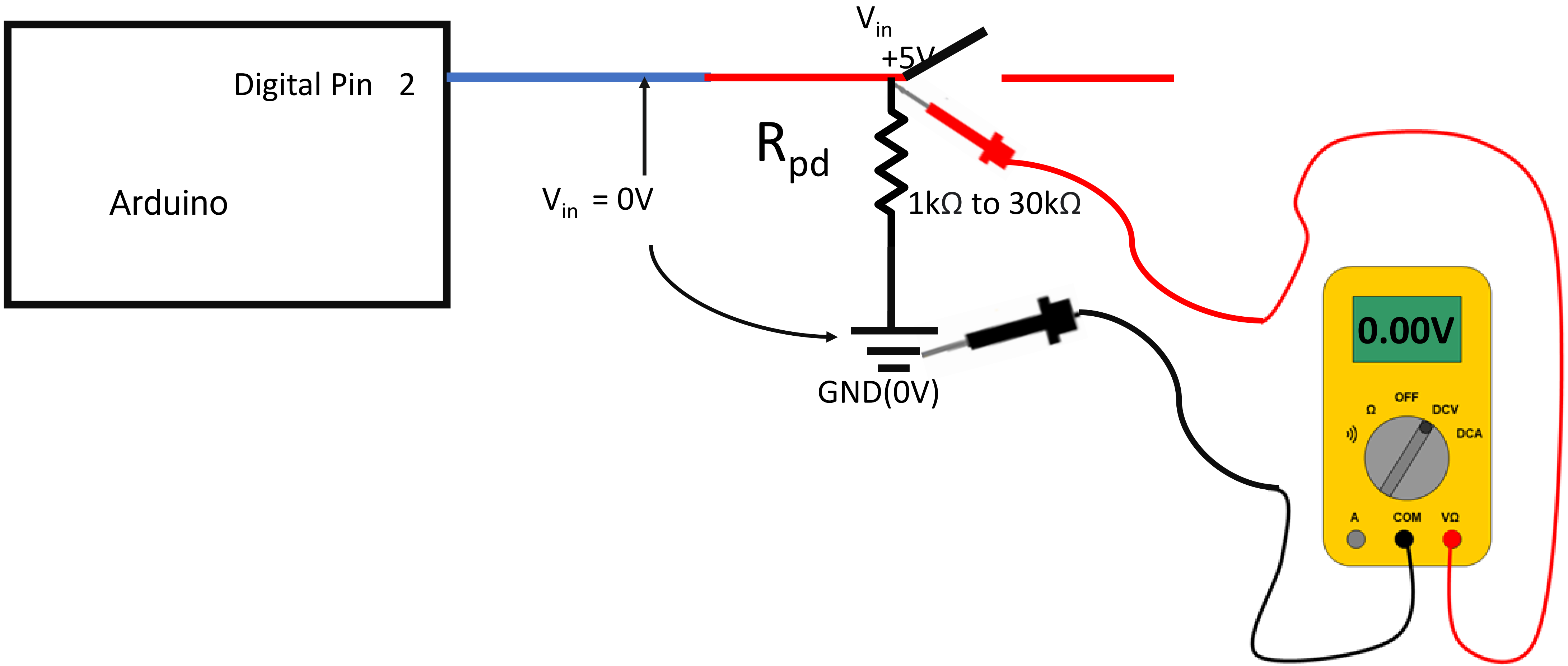
LCD pins	Arduino pin number
LCD_RS	5
LCD_RW	6
LCD_EN	7
LCD_D4	8
LCD_D5	9
LCD_D6	10
LCD_D7	11

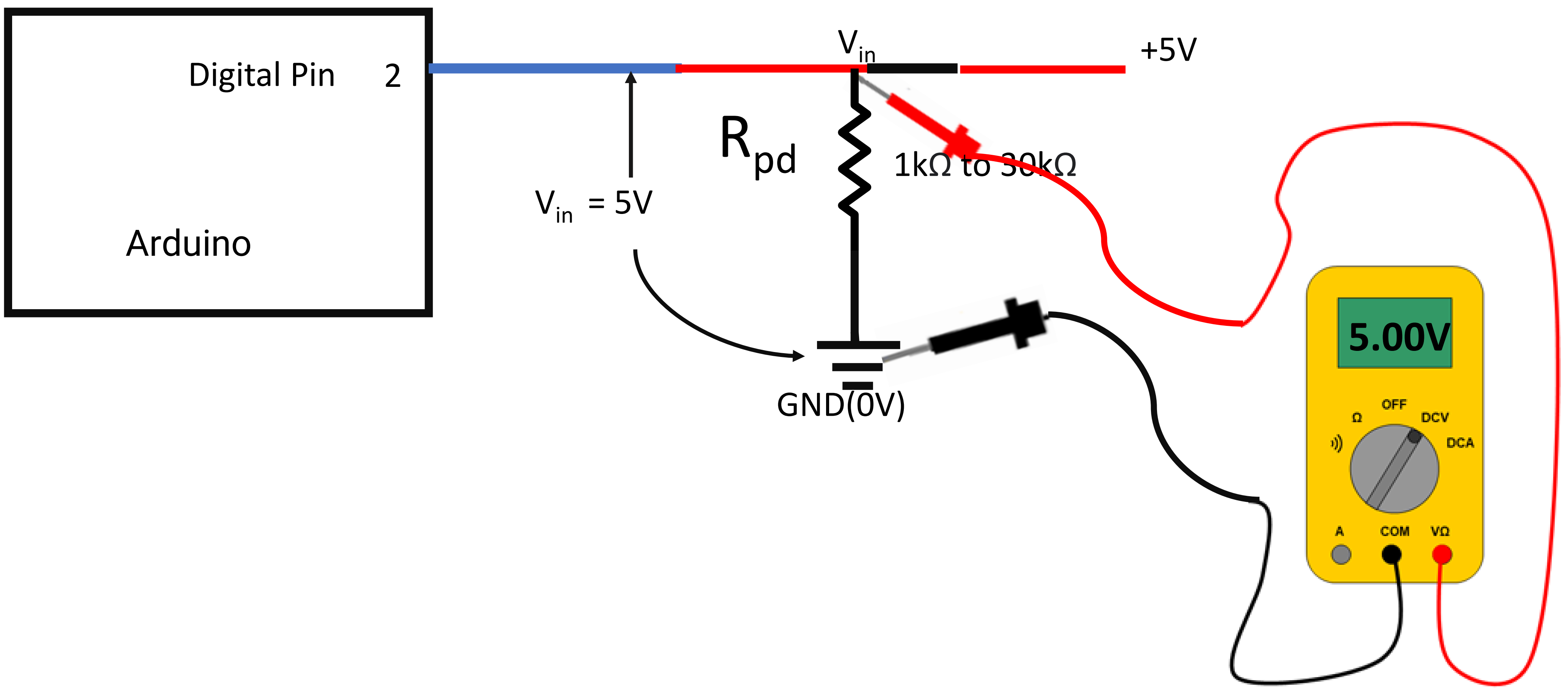
www.fastbitlab.com

Push button interfacing to Arduino using Pull down resistor

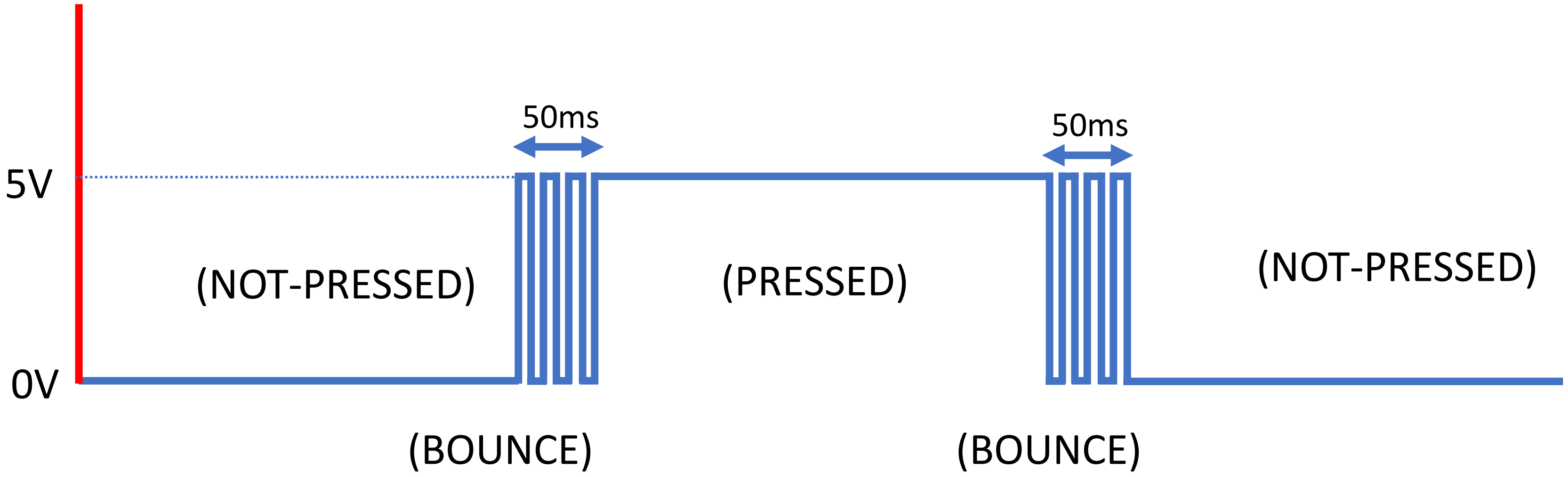


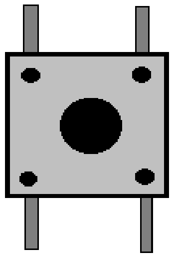



Digital Pin 2
Arduino
V_{in}
+5V
R_{pd}
V_{in} = 0V
1kΩ to 30kΩ
GND(0V)
0.00V
OFF
Ω
DCV
DCA
A
COM
VΩ


Digital Pin 2
Arduino
V_{in} = 5V
R_{pd}
V_{in}
+5V
1kΩ to 30kΩ
GND(0V)
5.00V
OFF
Ω
DCV
DCA
A
COM
VΩ

Button Bouncing





Implementation of state machine

- Nested switch approach
- State table approach
- State handler approach [1]

[1] These approaches were originally described in the book 'Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems [SECOND EDITION] by Miro Samek '

Function pointers in ‘C’

State handler approach

```
event_status state_handler_1(protimer_t *mobj,event_t *e){
    //Switch between different signals


    return EVENT_IGNORED;
}
```

```
event_status state_handler_2(protimer_t *mobj,event_t *e){
    //Switch between different signals


    return EVENT_IGNORED;
}
```

```
protimer_state_t active_state;
```

(Function pointer)

```
event_status state_handler_3(protimer_t *mobj,event_t *e){
    //Switch between different signals


    return EVENT_IGNORED;
}
```

```
event_status state_handler_4(protimer_t *mobj,event_t *e){
    //Switch between different signals


    return EVENT_IGNORED;
}
```

State table approach

State table

State	Event						
	INC_TIME	DEC_TIME	TIME_TICK	START_PAUSE	ABRT	ENTRY	EXIT
IDLE	IDLE_Inc_time() NextState : TIME_SET	NULL	IDLE_Time_tick()	IDLE_Start_pause() N.S: STAT	NULL	IDLE_Entry()	IDLE_Exit()
TIME_SET	TIME_SET_Inc_time()	TIME_SET_Dec_time()	NULL	TIME_SET_Start_pause() NextState: COUNTDOWN	TIME_SET_abrt() NextState: IDLE	TIME_SET_Entr y()	TIME_SET_Exit()
COUNT DOWN	NULL	NULL	COUNTDOWN_Time NextState: IDL	COUNTDOWN_Start_pau se() NextState : PAUSE	COUNTDOWN_abrt() NextState: ABRT	NULL	COUNTDOWN_E xit()
PAUSE	PAUSE_Inc_time() NextState: TIME_SET	PAUSE_Dec_time() NextState: TIME_SET	NULL	PAUSE_Start_pause() NextState: COUNTDOWN	PAUSE_abrt() NextState:IDLE	PAUSE_Entry()	PAUSE_Exit()
STAT	NULL	NULL	STAT_Time_tick() NextState: IDLE	NULL	NULL	STAT_Entry()	STAT_Exit()

Table of handlers

2-Dimensional array in 'C'

Indexing in x dimension

```
uint8_t scores[3]= {10,20,30};
```

```
uint8_t scores[2][3]= { {10,20,30}, {33,22,45} };
```

Indexing in dimension-x

```
uint8_t scores[2][3]= { {10,20,30},
                        {33,22,45}
                      };
```

row col

Row = 2
Col = 3

Indexing in dimension-y

Signifies a data representation in the form of table which has 2 rows and 3 columns

```
                         (0,0) (0,1) (0,2)
uint8_t scores[2][3]= { {10,20,30},          row-0
                        {33,22,45}           Row-1
                      };  (1,0) (1,1) (1,2)
```

Marksheet of Test-1 as per roll numbers

Roll No. \ Subject	Math(0)	Physics(1)	Chemistry(2)	Economics(3)
0	34	56	77	99
1	67	34	89	44
2	78	33	30	67
3	98	45	21	43

```
uint8_t marksheet_of_test1[16] = {34,56,77,99,67,34,89.......21,43};
```

What’s the mark scored by roll number 1 in Chemistry ?

```
uint8_t mark = marksheet_of_test1[1 * 4 + 2];
```

www.fastbitlab.com

Marksheet of Test-1 as per roll numbers

Subject / Roll No.	Math(0)	Physics(1)	Chemistry(2)	Economics(3)
0	34	56	77	99
1	67	34	89	44
2	78	33	30	67
3	98	45	21	43

```
uint8_t marksheet_of_test1[4][4] = { {34,56,77,99},
                                     {67,34,89,44},
                                     {78,33,30,67},
                                     {98,45,21,43}
                                   };
```

Subject

Marks

Table of marks

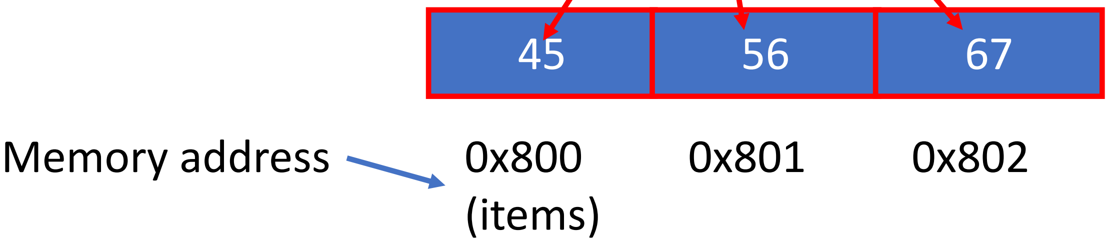
Roll number

What's the mark scored by roll number 2 in Chemistry ?

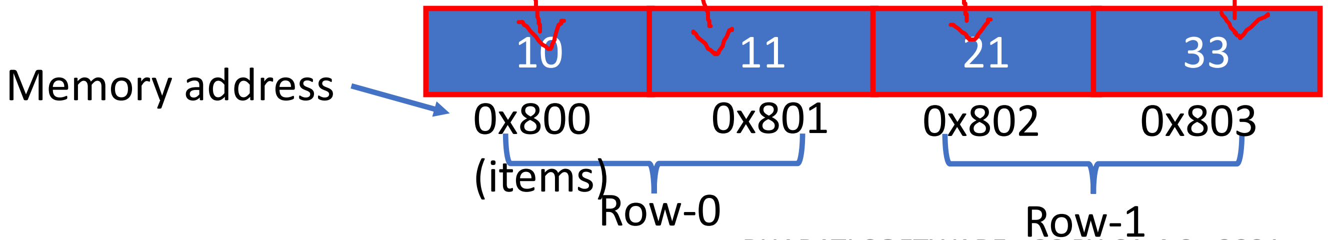
```
uint8_t mark =
marksheet_of_test1[2][2];
```

Storage of 2D array in memory

```
uint8_t items[3] = {45,56,67};
```




```
uint8_t items[2][2]  = { {10,11},
                         {21,33}
                       };
```



www.fastbitlab.com

Protimer State table

State	Event						
	INC_TIME	DEC_TIME	TIME_TICK	START_PAUSE	ABRT	ENTRY	EXIT
IDLE	IDLE_Inc_time() NextState : TIME_SET	NULL	IDLE_Time_tick()	IDLE_Start_pause() N.S: STAT	NULL	IDLE_Entry()	IDLE_Exit()
TIME_SET	TIME_SET_Inc_time()	TIME_SET_Dec_time()	NULL	TIME_SET_Start_pause() NextState: COUNTDOWN	TIME_SET_abrt() NextState: IDLE	TIME_SET_Entr y()	TIME_SET_Exit()
COUNT DOWN	NULL	NULL	COUNTDOWN_Time NextState: IDL	COUNTDOWN_Start_pau se() NextState : PAUSE	COUNTDOWN_abrt() NextState: ABRT	NULL	COUNTDOWN_E xit()
PAUSE	PAUSE_Inc_time() NextState: TIME_SET	PAUSE_Dec_time() NextState: TIME_SET	NULL	PAUSE_Start_pause() NextState: COUNTDOWN	PAUSE_abrt() NextState:IDLE	PAUSE_Entry()	PAUSE_Exit()
STAT	NULL	NULL	STAT_Time_tick() NextState: IDLE	NULL	NULL	STAT_Entry()	STAT_Exit()

Table of handlers

	column 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	column 15
row 0				H	h	:	m	m	:	s	s	.	s	_	a	m
row 1																

16x2 LCD

	column 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	column 15
row 0					A	L	A	R	M	!	!	!				
row 1				H	h	:	m	m	:	s	s	.	s	_	a	m

16x2 LCD

Notify alarm

	column 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	column 15
row 0				H	h	:	m	m	:	s	s	.	s	_	a	m
row 1				H	h	:	m	m	:	s	s	.	s	_	a	m

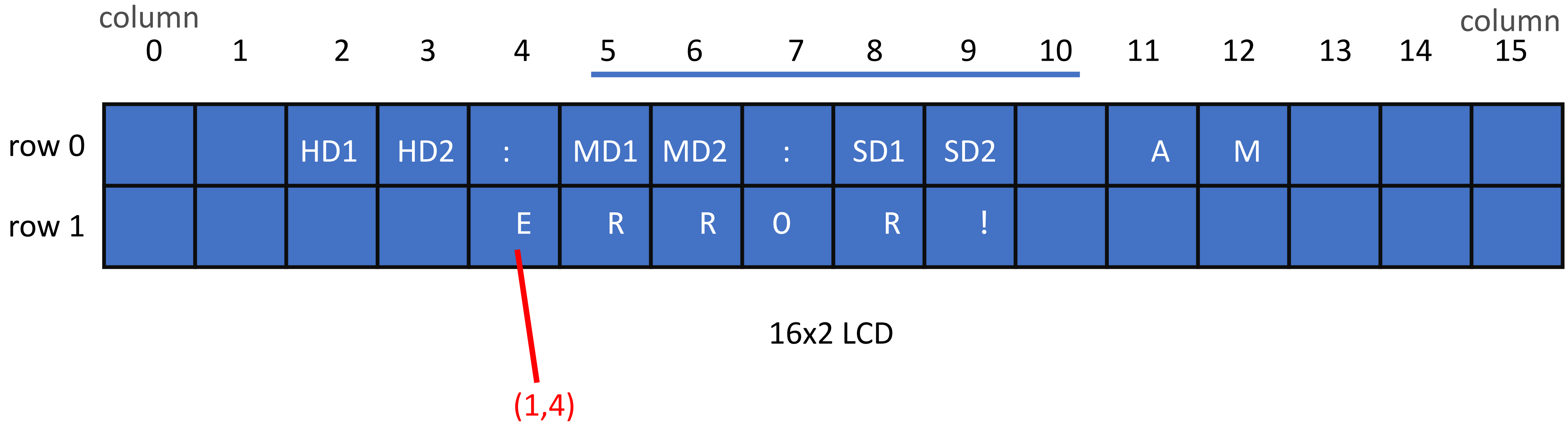
16x2 LCD

Alarm setting

	column 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	column 15
row 0					A	L	A	R	M		O	F	F			
row 1				H	h	:	m	m	:	s	s	.	s	_	a	m

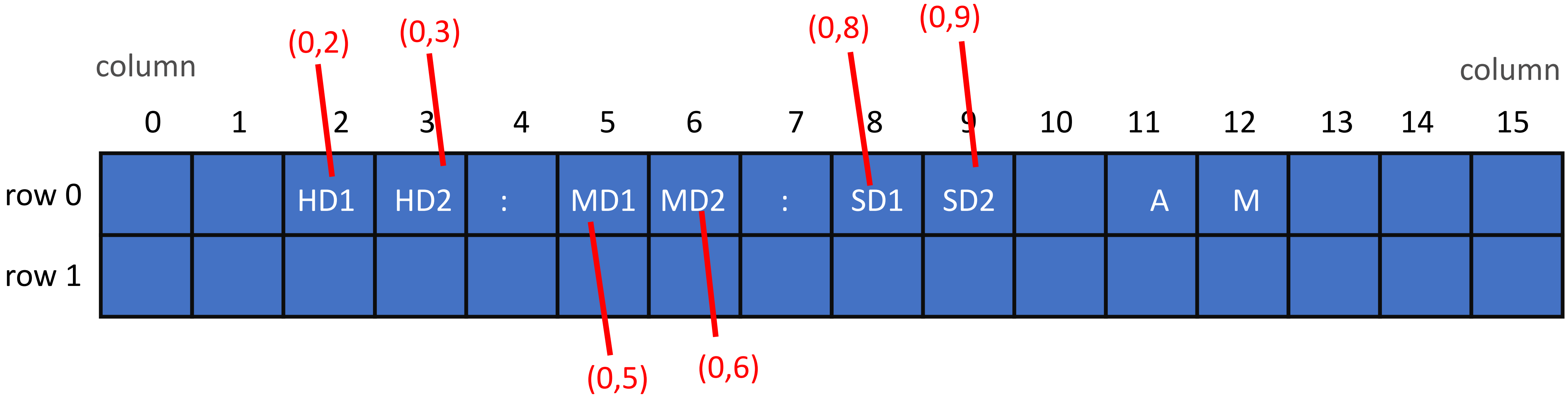
16x2 LCD

Alarm status



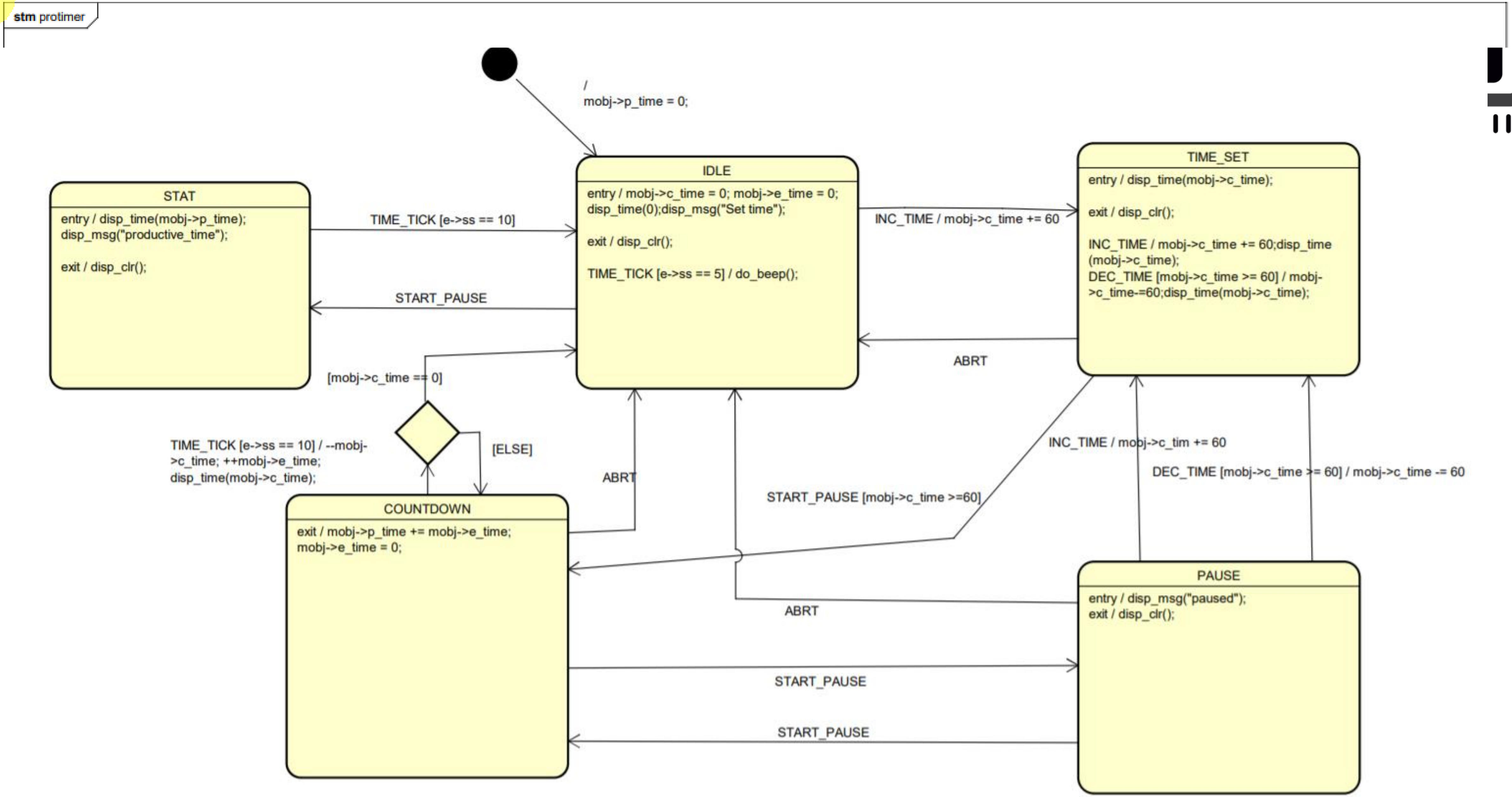
SETTING errors

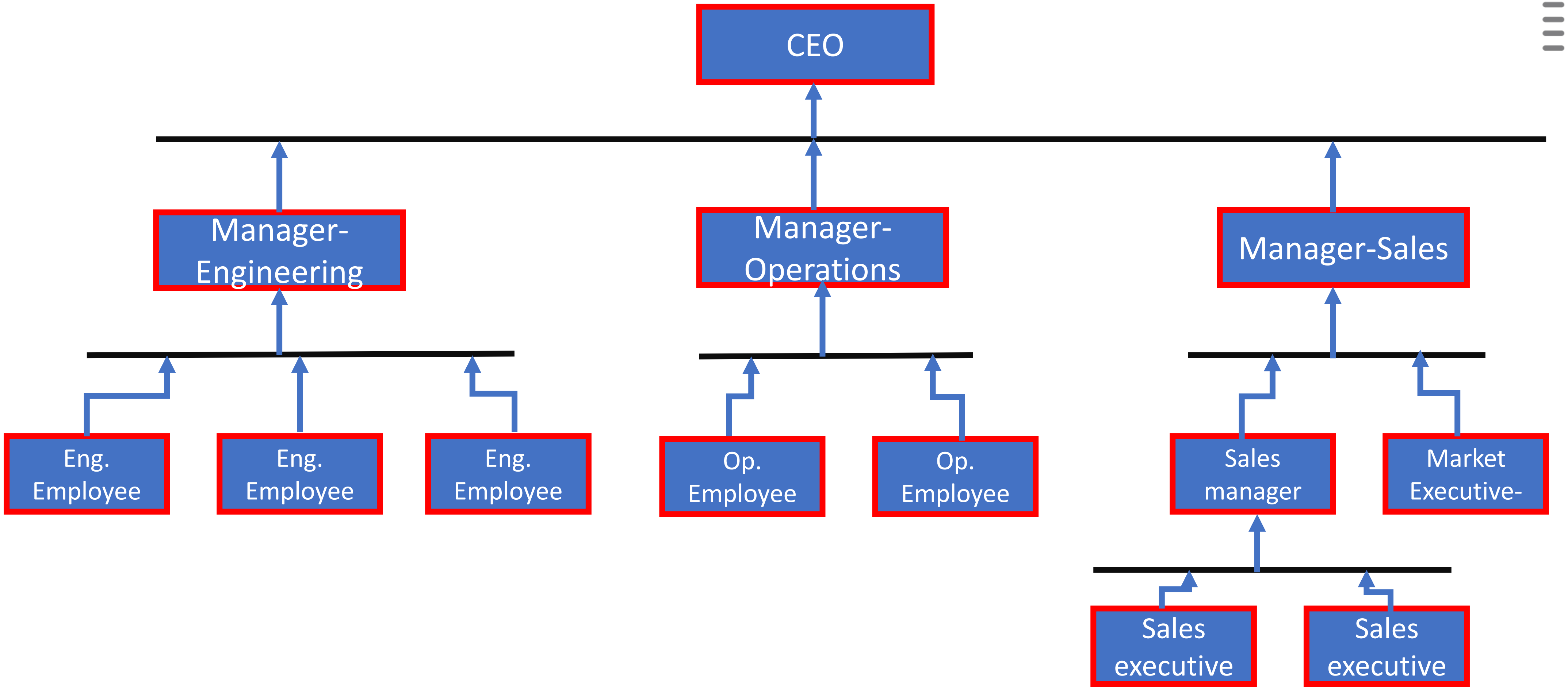
www.fastbitlab.com



16x2 LCD

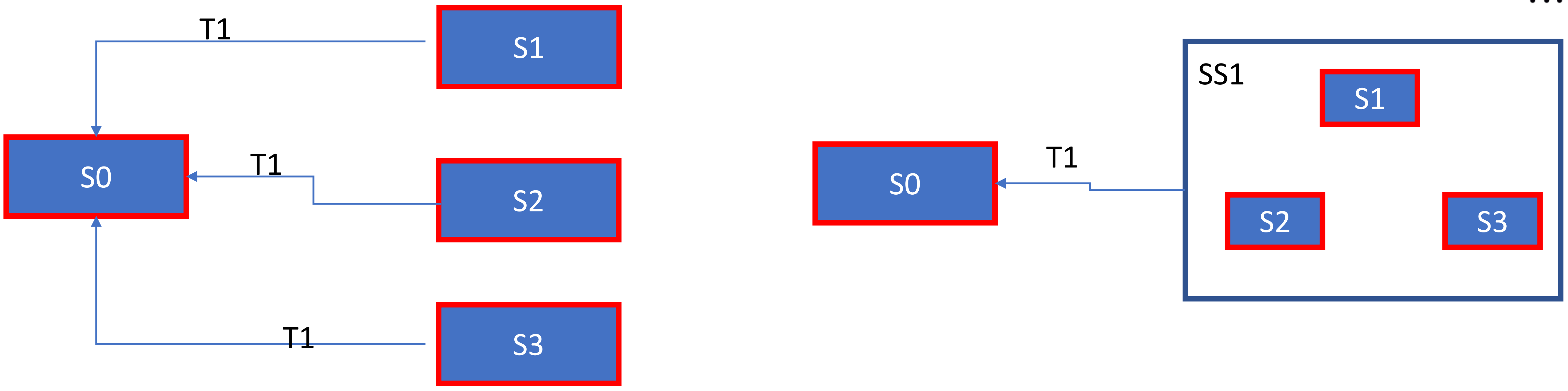
HSMs(Hierarchical State Machines)

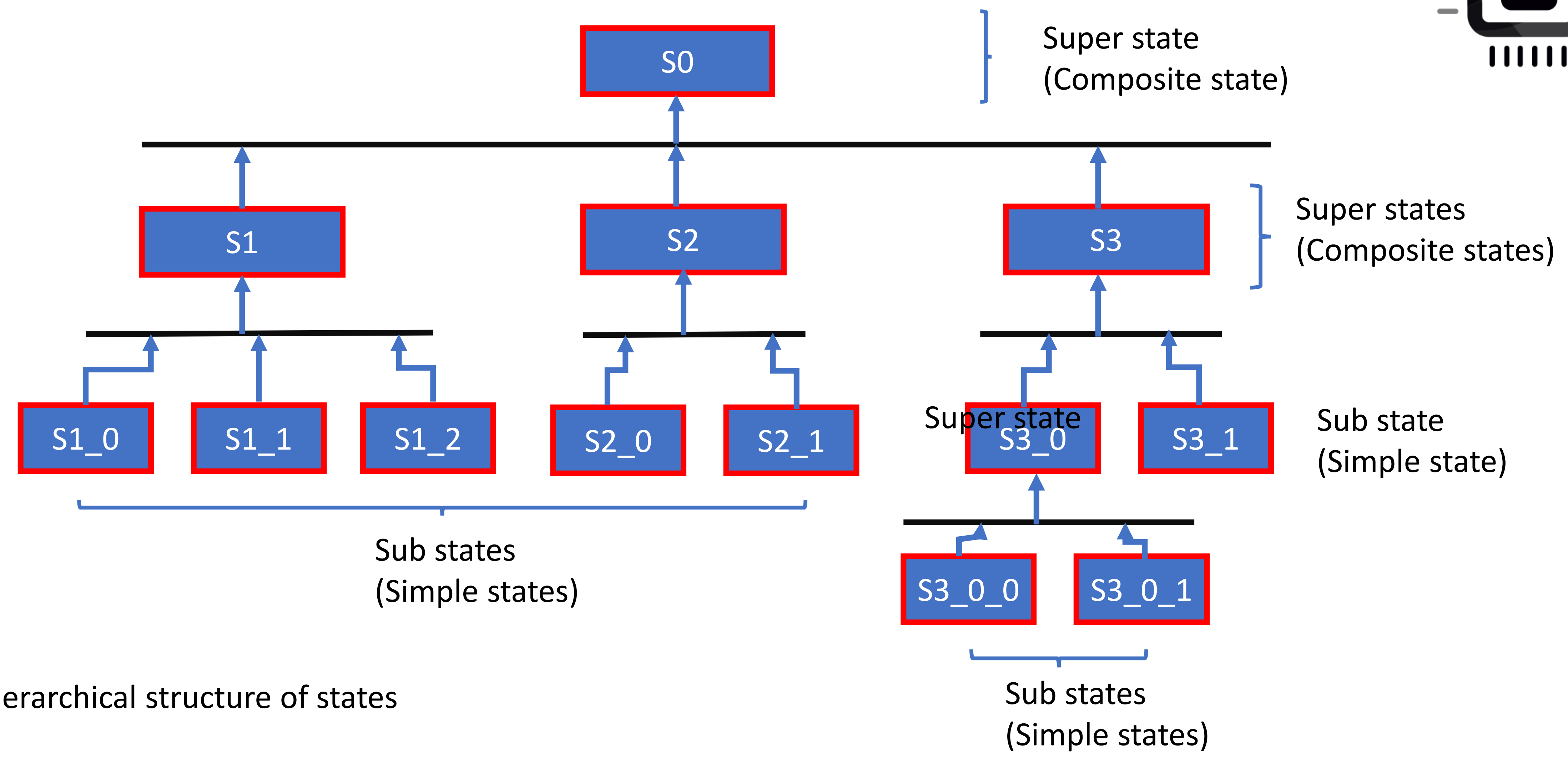

stm protimer
/
mobj->p_time = 0;
IDLE
entry / mobj->c_time = 0; mobj->e_time = 0;
disp_time(0);disp_msg("Set time");
exit / disp_clr();
TIME_TICK [e->ss == 5] / do_beep();
STAT
entry / disp_time(mobj->p_time);
disp_msg("productive_time");
exit / disp_clr();
TIME_SET
entry / disp_time(mobj->c_time);
exit / disp_clr();
INC_TIME / mobj->c_time += 60;disp_time
(mobj->c_time);
DEC_TIME [mobj->c_time >= 60] / mobj-
>c_time-=60;disp_time(mobj->c_time);
TIME_TICK [e->ss == 10]
START_PAUSE
INC_TIME / mobj->c_time += 60
ABRT
[mobj->c_time == 0]
[ELSE]
TIME_TICK [e->ss == 10] / --mobj-
>c_time; ++mobj->e_time;
disp_time(mobj->c_time);
ABRT
INC_TIME / mobj->c_tim += 60
DEC_TIME [mobj->c_time >= 60] / mobj->c_time -= 60
START_PAUSE [mobj->c_time >=60]
COUNTDOWN
exit / mobj->p_time += mobj->e_time;
mobj->e_time = 0;
ABRT
PAUSE
entry / disp_msg("paused");
exit / disp_clr();
START_PAUSE
START_PAUSE



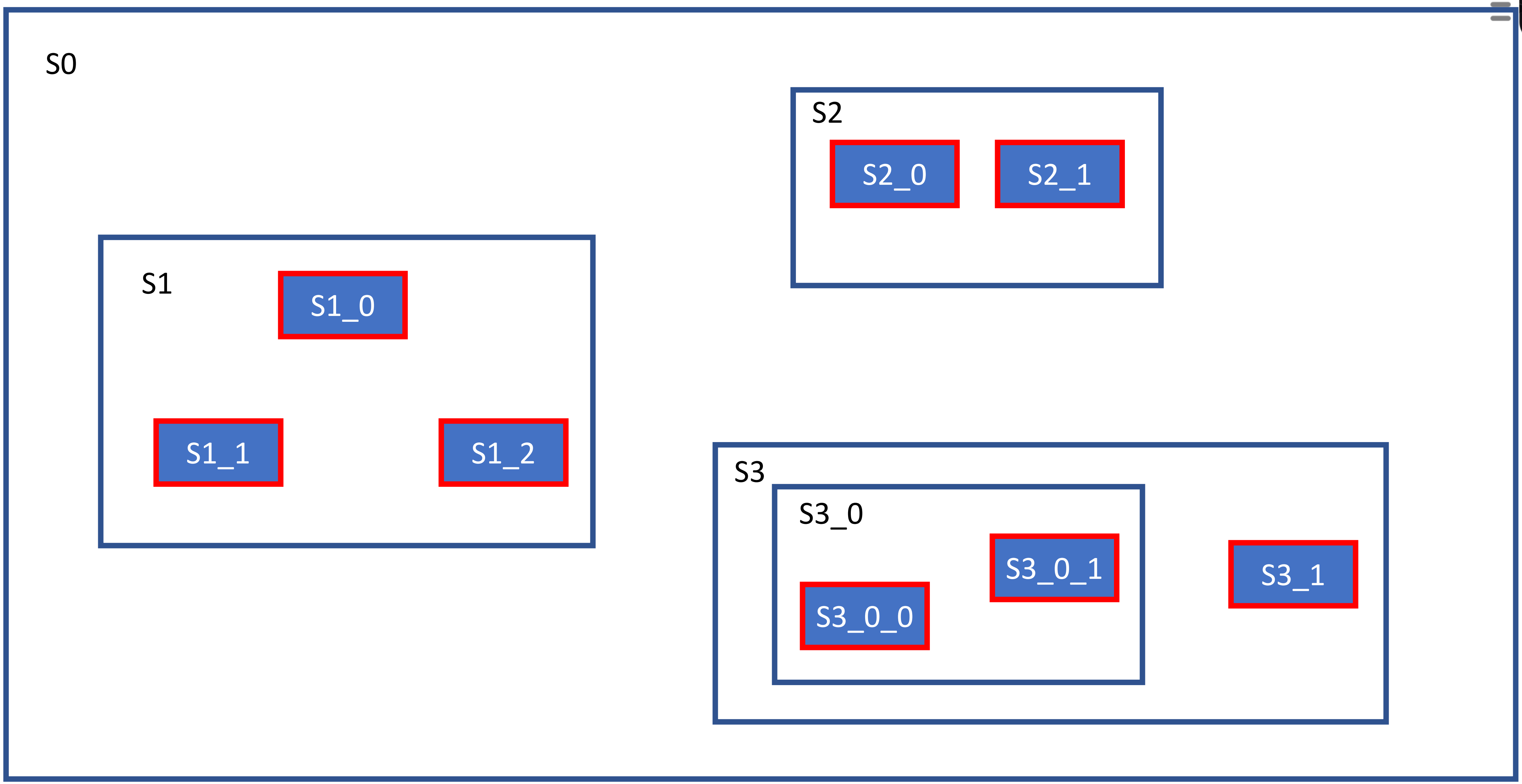
Corporate hierarchy

www.fastbitlab.com





Hierarchical structure of states


S0
S2
S2_0
S2_1
S1
S1_0
S1_1
S1_2
S3
S3_0
S3_0_1
S3_0_0
S3_1

Cons of flat state machine

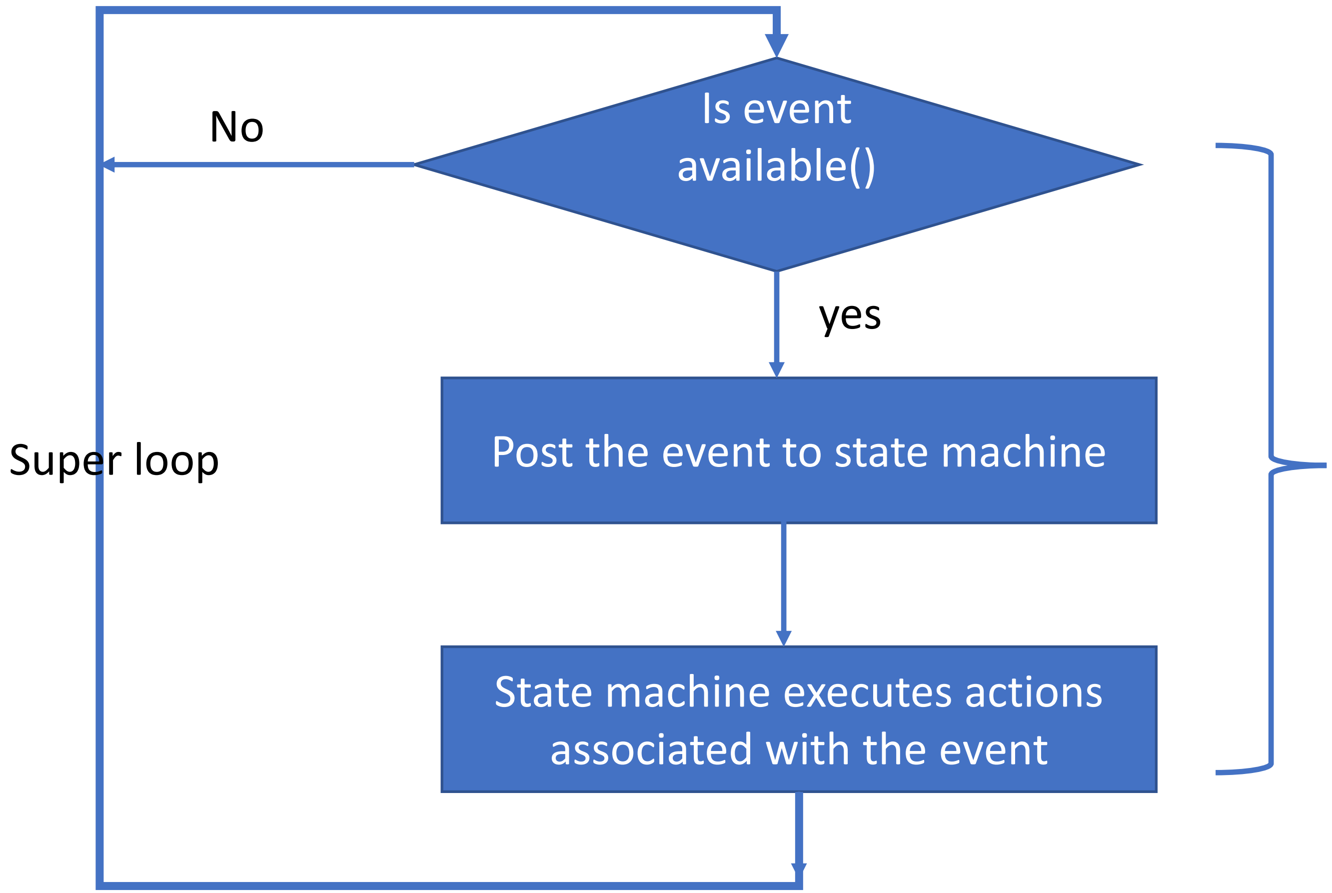
- As the number of states increases, the number of transitions tends to increase, which increases the complexity
- Hard to draw, visualize, trace and troubleshoot
- Chances of committing more mistakes by missing transitions
- Code repetitions (not efficient code reusability)

Run-to-completion(RTC) paradigm

[OMG® UML 2.5.1]

- Run-to-completion means that, in the absence of exceptions or asynchronous destruction of the StateMachine execution, a pending Event occurrence is dispatched only after the processing of the previous occurrence is completed and a stable state configuration has been reached. That is, an Event occurrence will never be dispatched while the StateMachine execution is busy processing the previous one. This behavioral paradigm was chosen to avoid complications arising from concurrency conflicts that may arise when a StateMachine tries to respond to multiple concurrent or overlapping events.
- The UML state machines follow run-to-completion paradigm

www.fastbitlab.com



An RTC step

Run-to-completion: Processing of the current event must finish first, before processing the next event

Do not put any blocking code (e.g. delay()) anywhere here, violating the rtc paradigm.

The event-driven + rtc paradigm provides better power efficiency. When the events are absent, you can utilize the MCU low power modes to put the MCU to sleep and make MCU active only during the execution of an rtc step.

HSM event processor

[1]QP™ real-time embedded frameworks (RTEFs) and QM™ modeling tool by Quantum Leaps, LLC

Explore more:
https://www.state-machine.com/
https://www.state-machine.com/products/qp
https://www.state-machine.com/licensing

Miro Samek
Founder and CEO, Quantum Leaps, LLC
https://www.state-machine.com/

[1] QP™ framework, QM™ tool, QTools™ are registered trademarks of Quantum Leaps, LLC

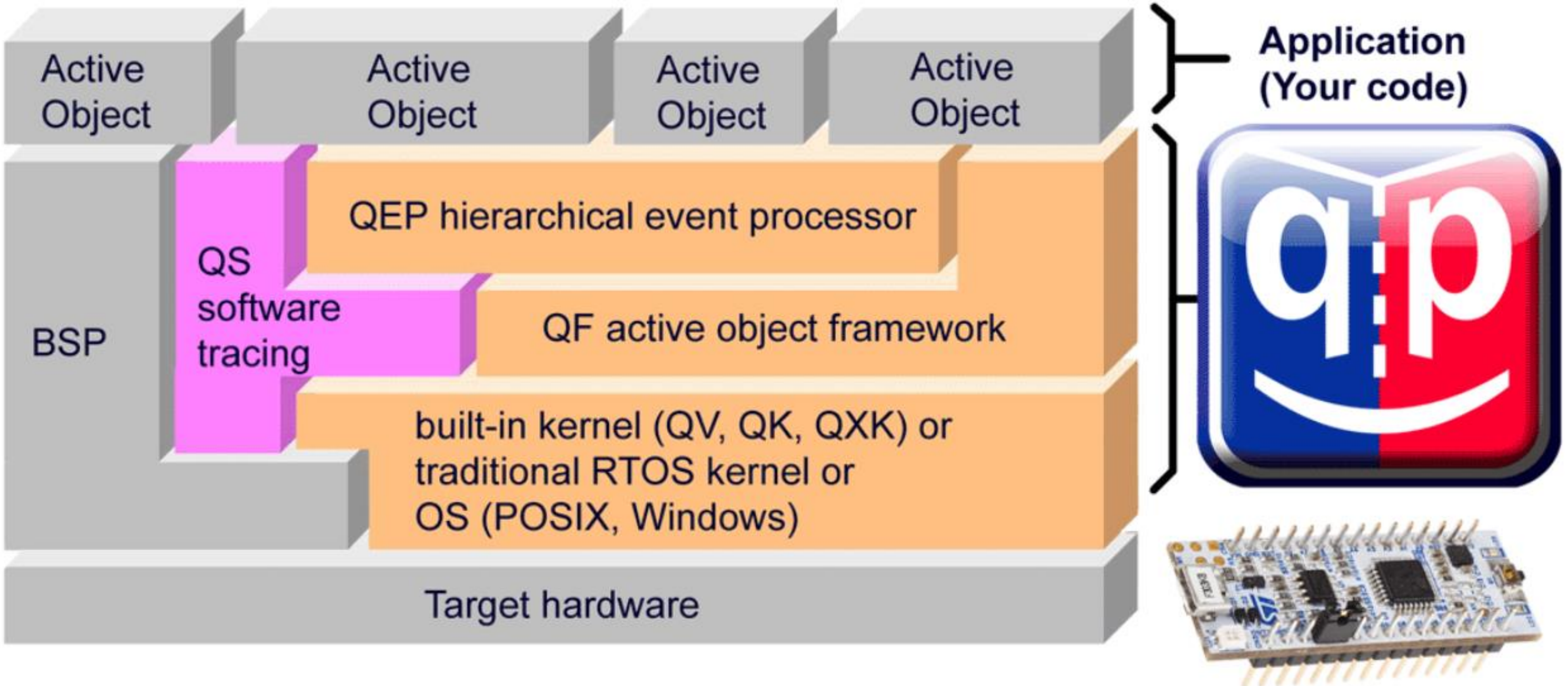
QP™ real-time embedded frameworks (RTEFs) and QM™ modeling tool by Quantum Leaps, LLC

- Run to completion and Event-driven architecture
- Support for nested HSMs(UML statecharts)
- Model-based design using QM™ tool (QP™ Modeler)
- Automatic code generation (converting UML statecharts into traceable c/c++ code)
- Support for Active objects (Actors)
- Various Kernel support
  - QV (simple cooperative)
  - QK (Preemptive Run-To-Completion (Non-Blocking) Kernel)
  - QXK (Preemptive Dual-Mode (Run-to-Completion/Blocking) RTOS Kernel)
- QP/Spy™ Software Tracing tool
- QUTest™ Unit Testing Harness
- Dual licensing (Open and closed source licensing)
- The QP/C and QP-nano frameworks comply with most of the MISRA-C:2004 rules while the QP/C++ framework complies with most of the MISRA-C++:2008 rules

QP™ Real-Time Embedded Frameworks (RTEFs)

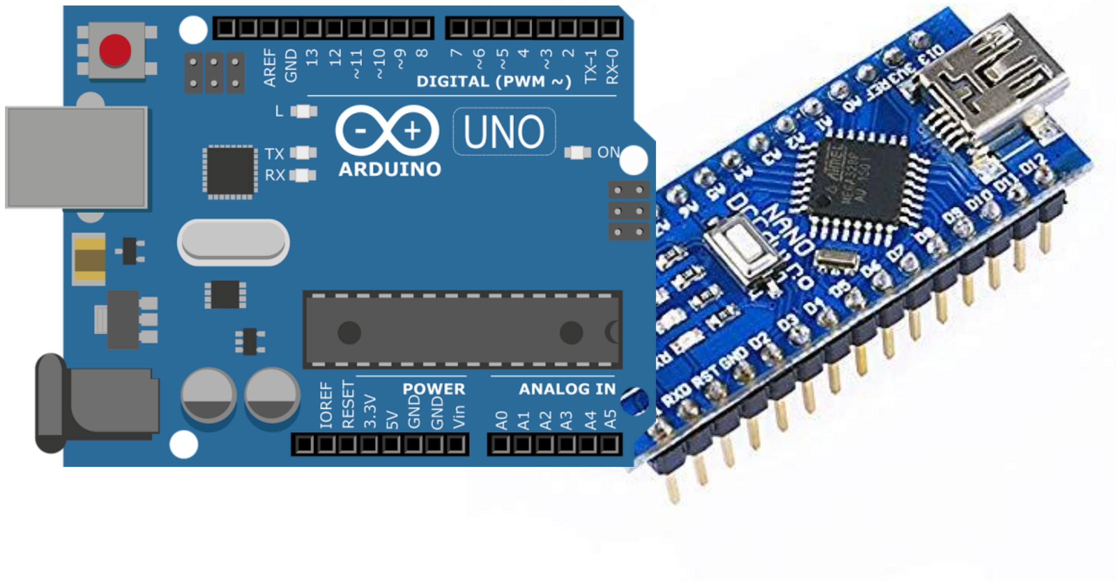
QP framework type	Note
QP/C™	Quantum Platform in C. The framework supports UML based FSM and nested HSM implementation in ‘C’
QP/C++™	Quantum Platform in C++. The framework supports UML based FSM and nested HSM implementation in ‘C++’
QP-nano™	Quantum Platform Nano. The framework is also based on ‘C’ and it is intended for low-end 8/16-bit microcontrollers such as AVRmega, MSP430, or 8051 with very limited RAM on board (less then 1KB).

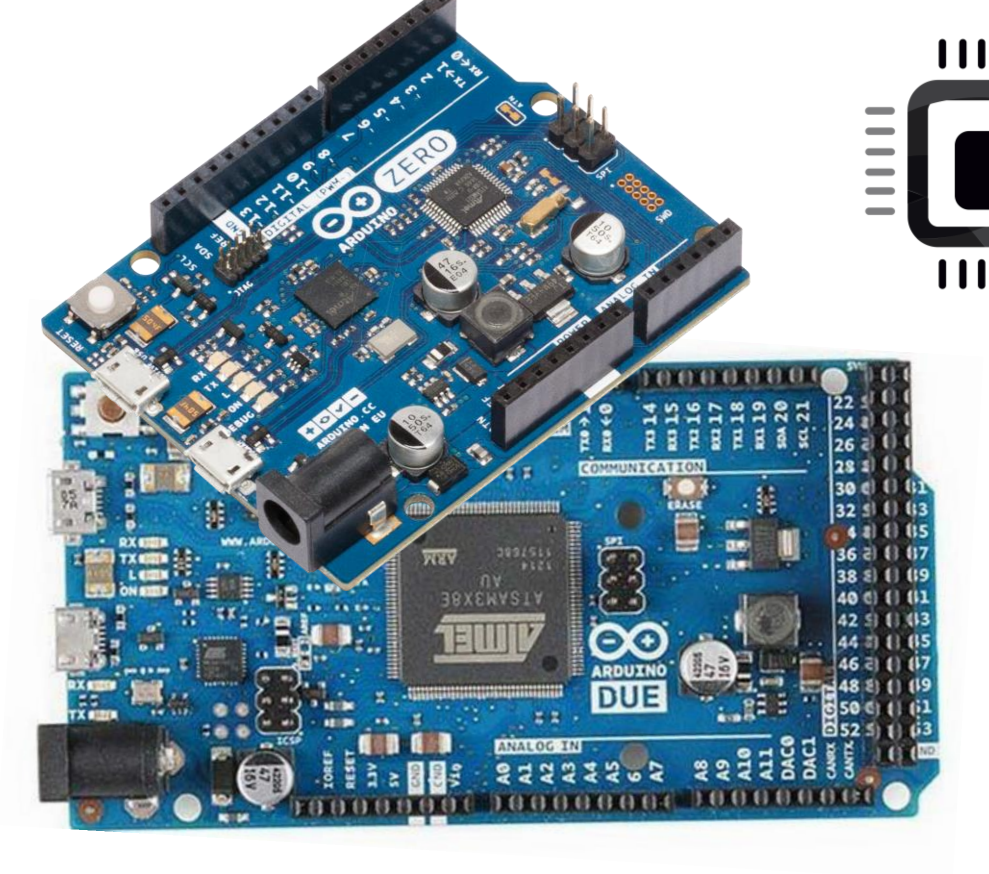
QP/C™, QP/C++™, QP-nano™ are registered trademarks of Quantum Leaps, LLC



Block diagram showing the components of the QP™ framework and their relationship to the hardware and the application

Source : https://www.state-machine.com/products/qp





AVR based Arduinos
QP-Nano Arduino library
QP-nano framework support will be discontinued

ARM based arduinos
QP/C++ Arduino library

Explore more : https://www.state-machine.com/arduino

Downloads

Download and install QM™ tool

https://www.state-machine.com/products/qm

Download QP Arduino library

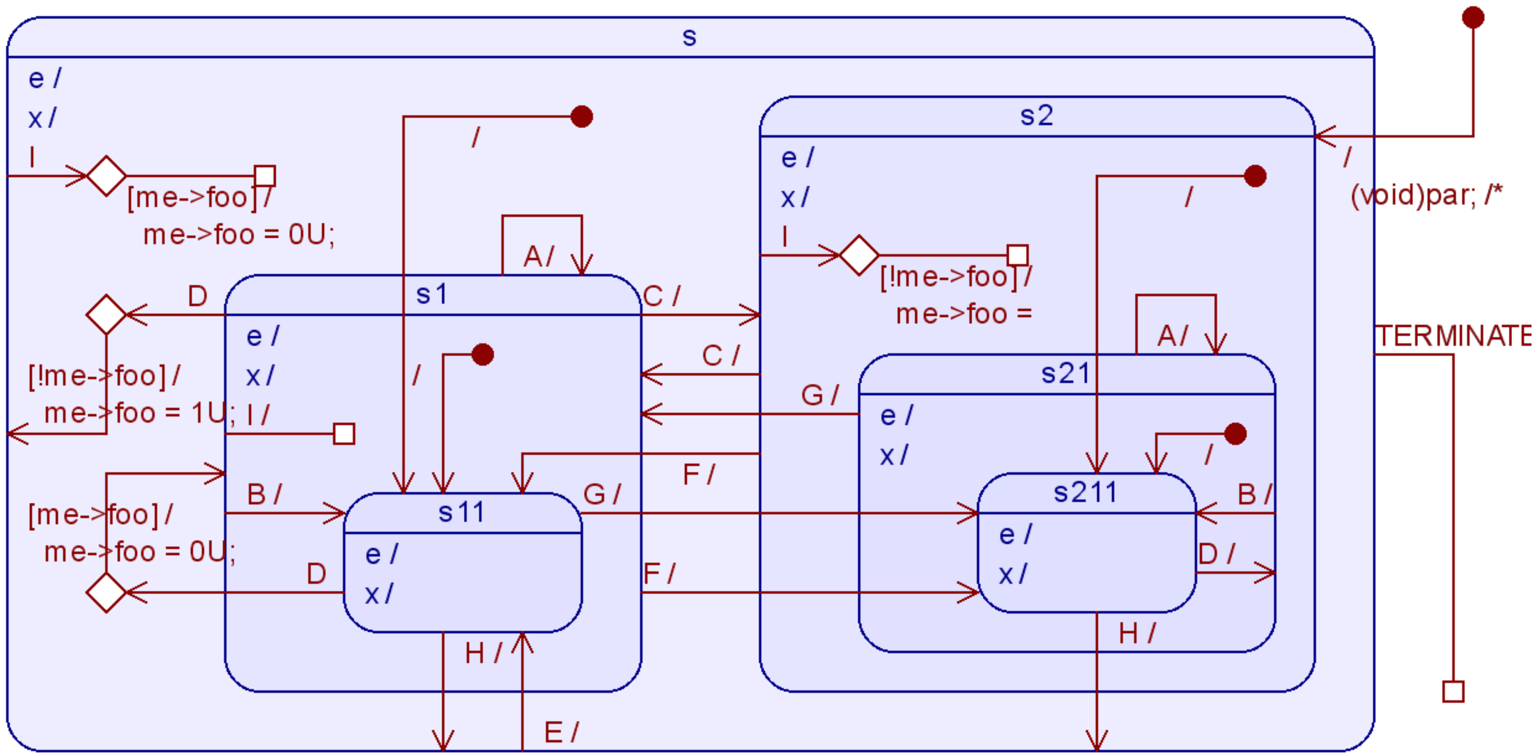
https://www.state-machine.com/arduino

QP-Nano API reference

https://www.state-machine.com/qpn/api.html

www.fastbitlab.com

Transition execution sequence and Event propagation



Location : <installation drive>\ qp\qpc\examples\workstation\qhsmtst

Adding files in QM tool

- QHSMTest_SM.c
  - State handler function declarations
  - State handler functions definitions
  - private variables and functions definitions
- QHSMTest_SM.h
  - Keep here something you want to share with other files
  - Event definitions(enum of events)
  - Function declarations of exposed functions

Code-Generation Directives
https://www.state-machine.com/qm/ce_directive.html

Summary

- Created QHSMTest_SM.cpp and QHSMTest_SM.h
- QHSMTest_SM.cpp
  - Structure declaration
  - State handlers declarations
  - State handlers definitions

  File scope
- QHSMTest_SM.h
  - Events declarations(enums)
  - Keep any exposed declarations of variables or functions

QP-Nano APIs

Refer : https://www.state-machine.com/qpn/api.html

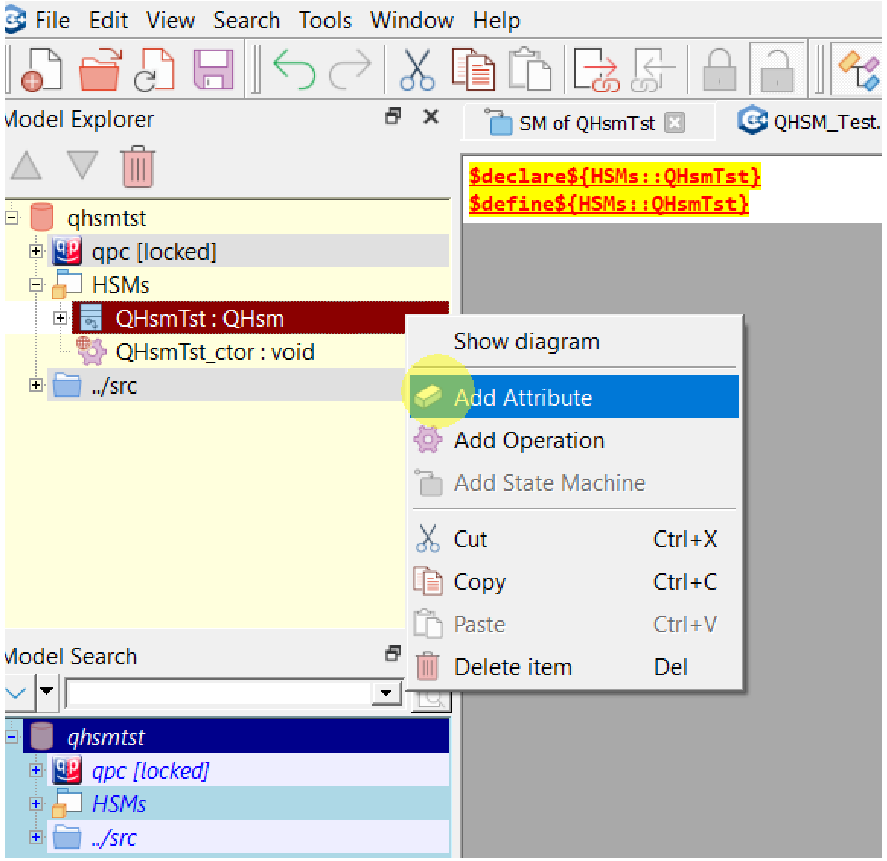
Adding a Class attribute

1)Static class attribute

- Static class attribute exists in one copy only, regardless of the number of class/structure instances.
- In ‘C++’, no matter how many instances(objects) of the class you create, there will be only one copy of static class attribute. Static attribute is used to share common information among different instances of the class

2)Non-static class attribute

- Non-static class attributes correspond to data members of a struct/class, present in each instance of the struct/class

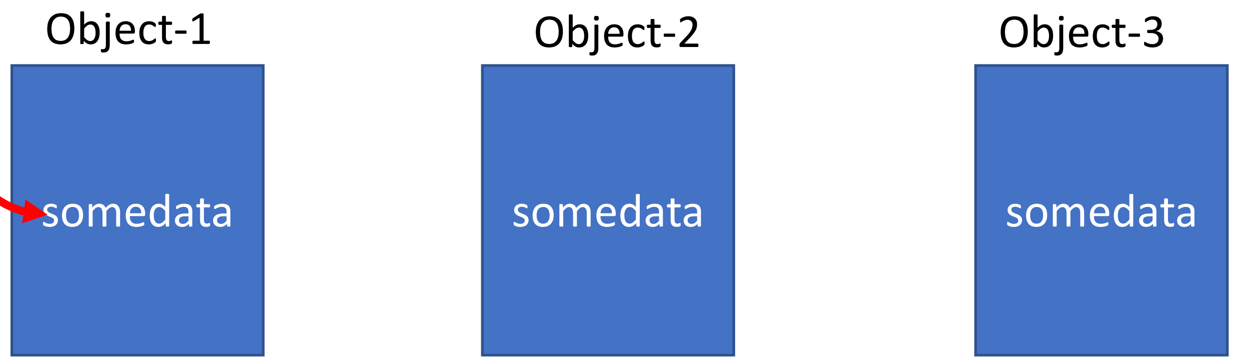


Source : https://www.state-machine.com/qm/bm_attr.html

```
class SomeClassName{
    private:
        /*non-static  attribute */
        int somedata;

        /*static attribute declaration */
        /*one copy for all the objects of this class */
        static int some_static_data;

    public:
        /*non-static method*/
        void setdata(int data){

        }

        /*static method*/
        static void static_function(int data){

        }
};

/* definition of static attribute of class 'SomeClassName' */
int SomeClassName::some_static_data = 0;

int main(void){
    return 0;
}
```

www.fastbitlab.com

Each object contains its own copy of 'non static' data



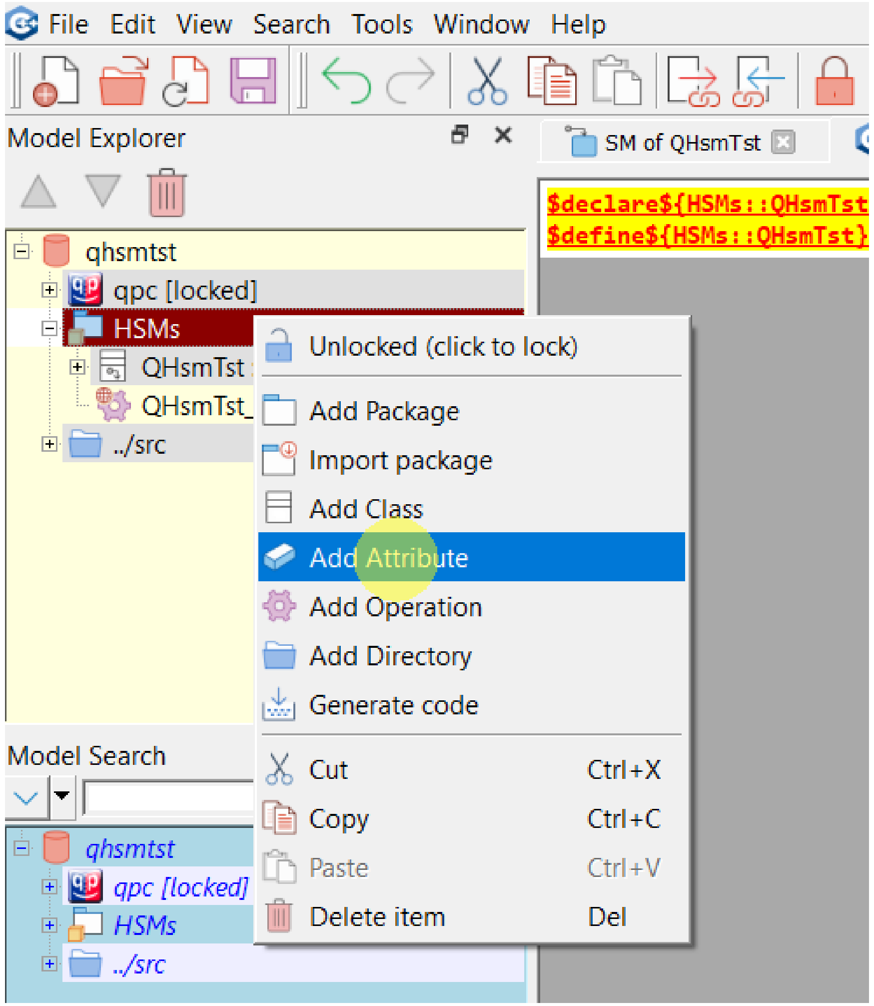
Objects of class 'SomeClassName'

int some_static_data

One copy for all the objects of the class 'SomeClassName'

Adding a free attribute

Refer : https://www.state-machine.com/qm/bm_attr.html



Adding a Class operation

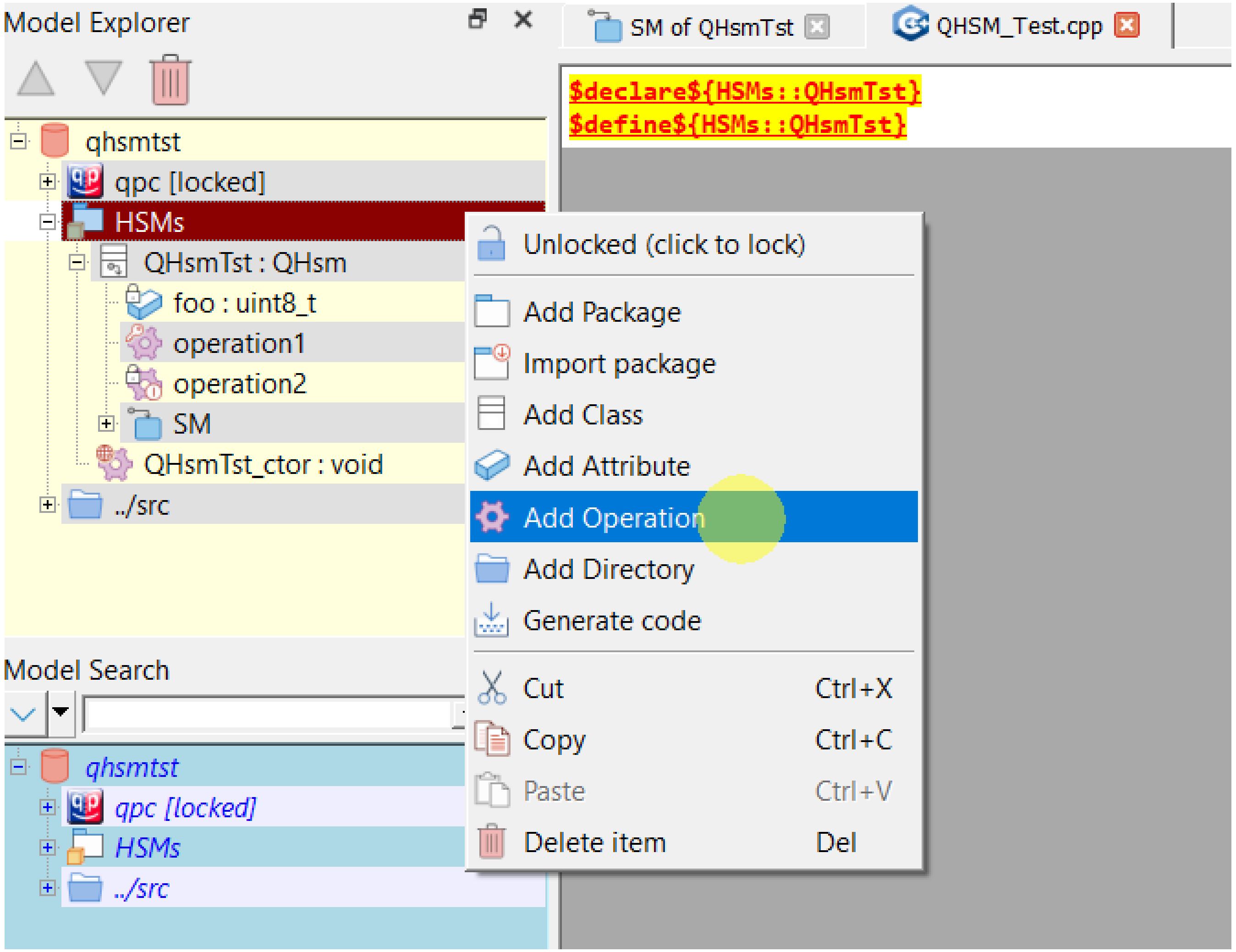
An operation added to a Class is called Class operation

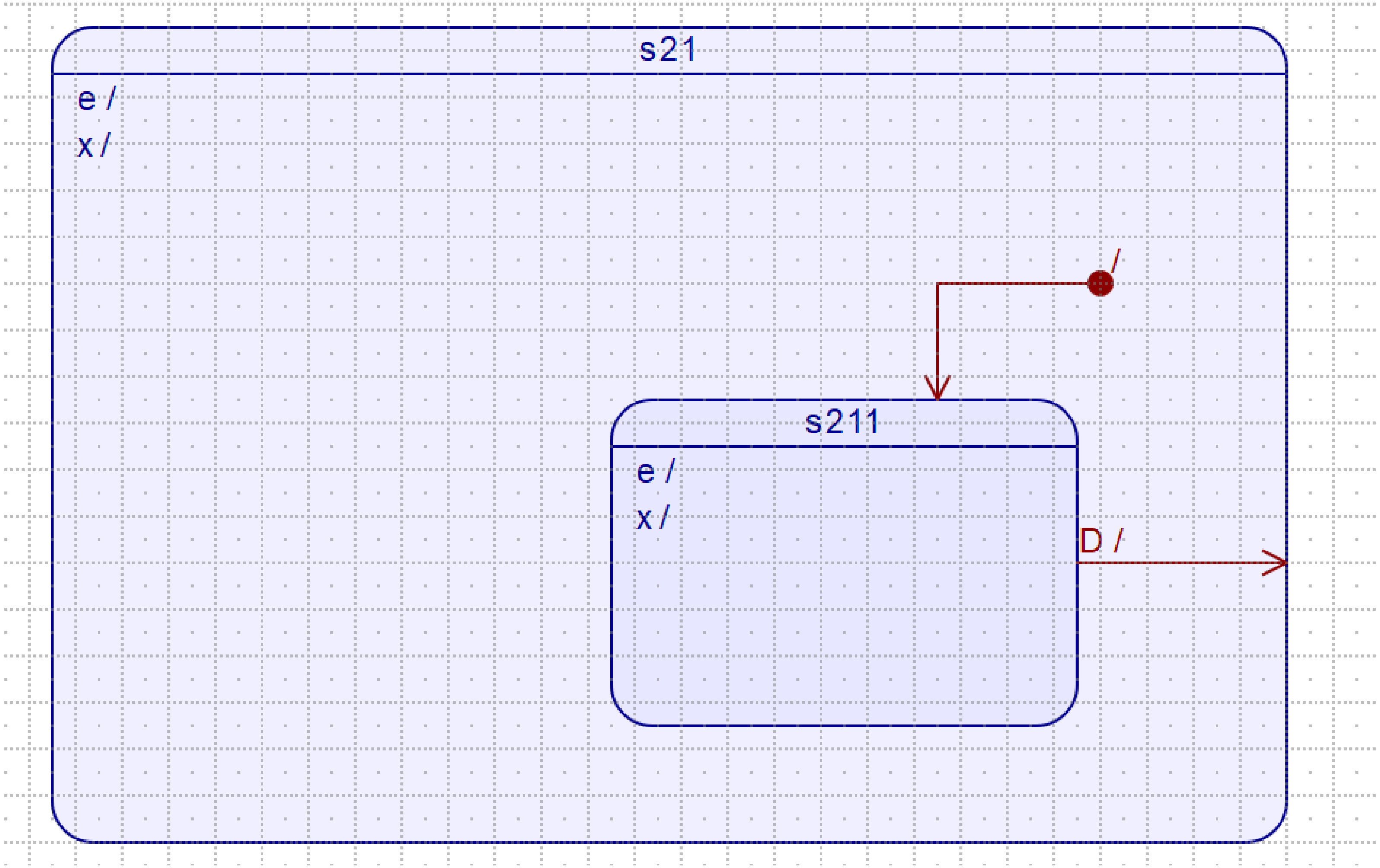
1. Static Class operation
   - In ‘C’, functions that don’t have access to the ‘me’ pointer(pointer of the instance of the main application structure)
2. Non-static Class operation
   - In C, a non-static class operation corresponds to a function that (by a coding convention) takes the "me" pointer to the struct with the class attribute. This is a function that can access instance of a main application structure(via ‘me’ pointer)
3. Class constructor
   - A function of the class/structure used to initialize the instance of a structure. The return type must be void.

Source : https://www.state-machine.com/qm/bm_oper-class.html

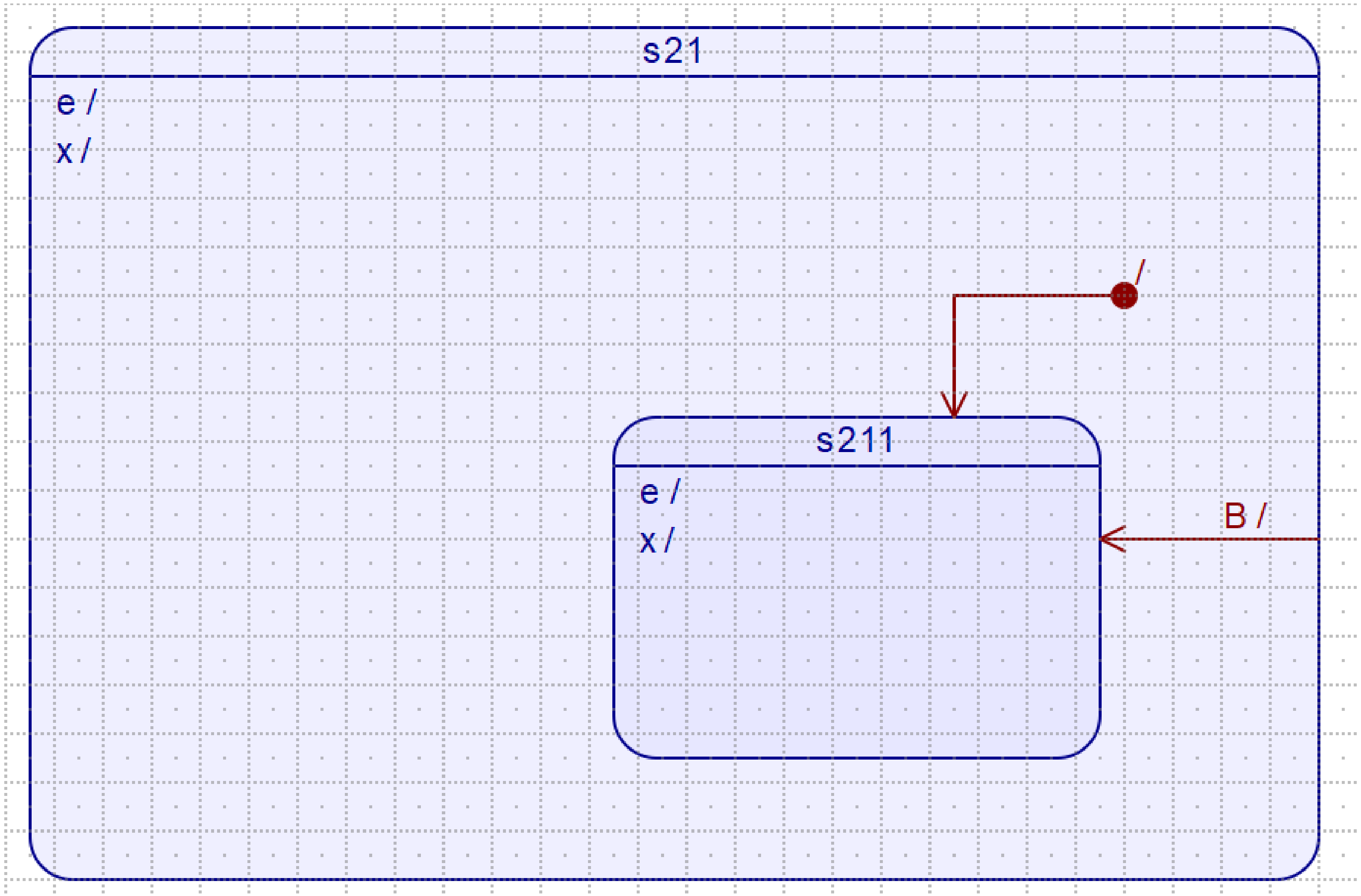
```
class SomeClassName{
    private:
        int somedata; /*non-static  attribute */
        /*static attribute . one copy for all the objects of this class */
        static int some_static_data;
    public:
        /*non-static operation or method*/
        void setdata(int data){
            somedata = data;
            this->somedata = data; //'this' pointer is available
        }
        /*static operation or  method*/
        static void static_function(int data){
            /*error: 'this' is unavailable for static member functions*/
            this->somedata = data;
            somedata = data; //error.
            some_static_data = data; //OK
        }
};
```

Adding a free operation

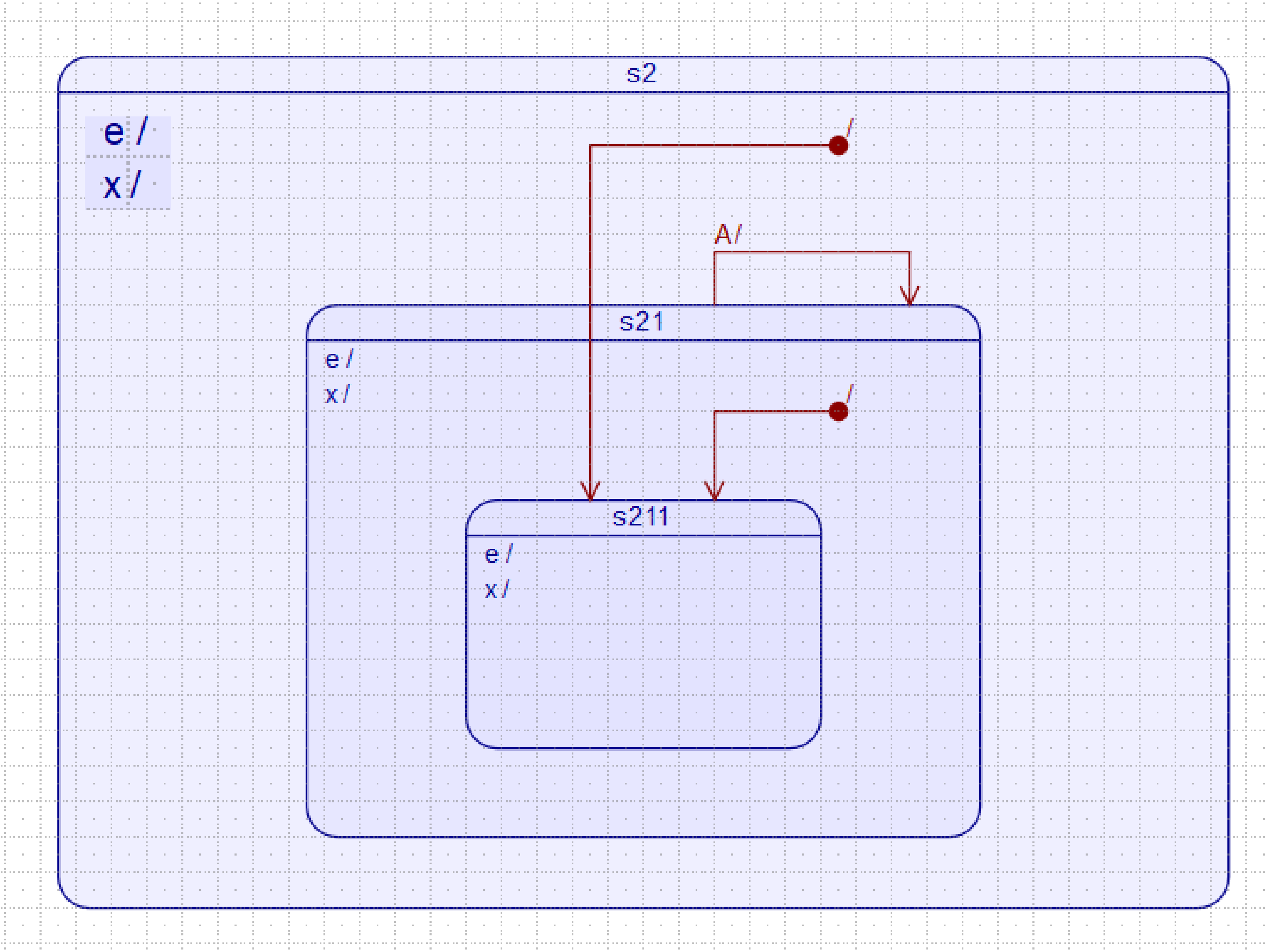




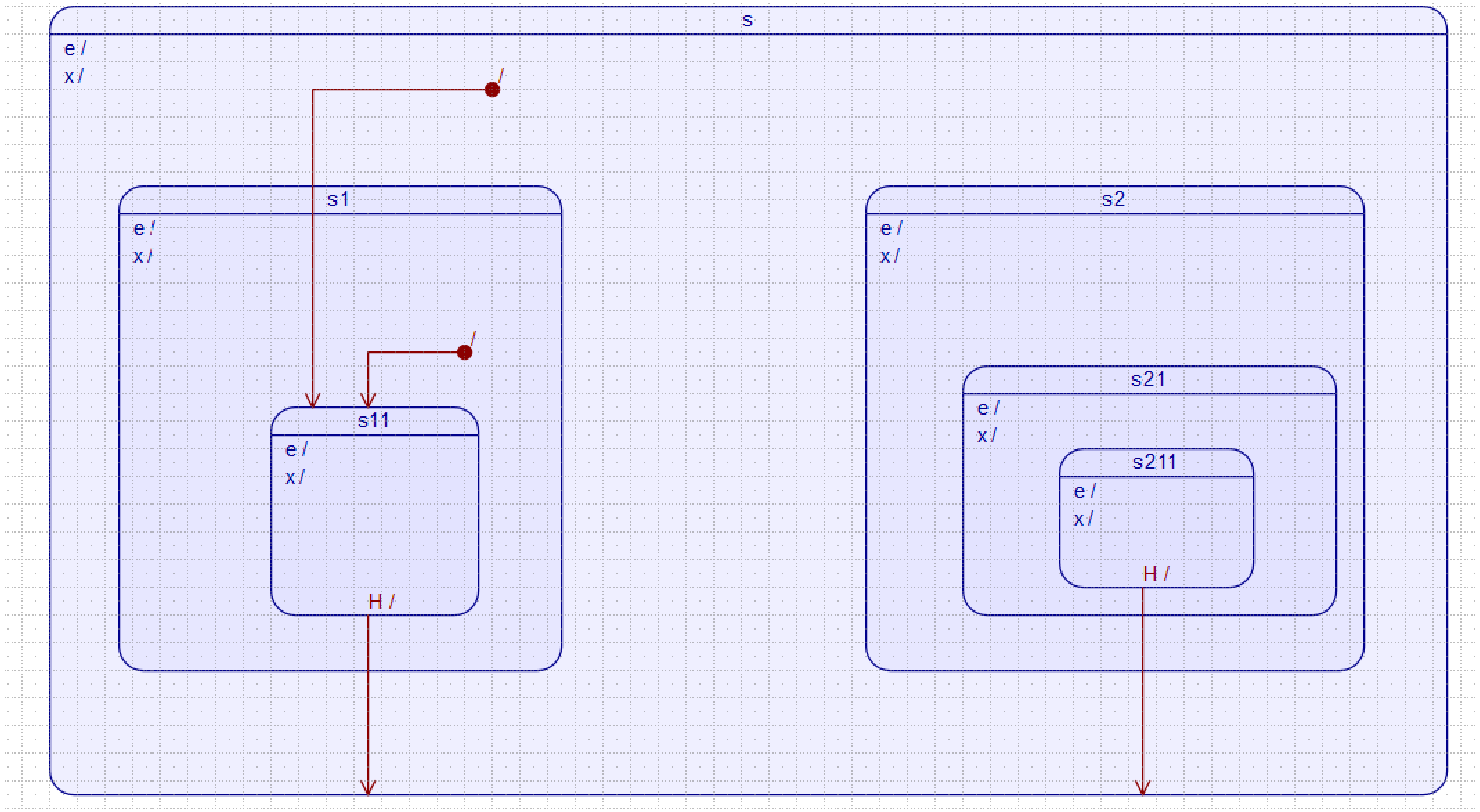
D→s211-D;s211-EXIT;s21-INIT;s211-ENTRY;



B→s21-B;s211-EXIT;s211-ENTRY;



A→s21-A;s211-EXIT;s21-EXIT;s21-ENTRY;s21-INIT;s211-ENTRY;

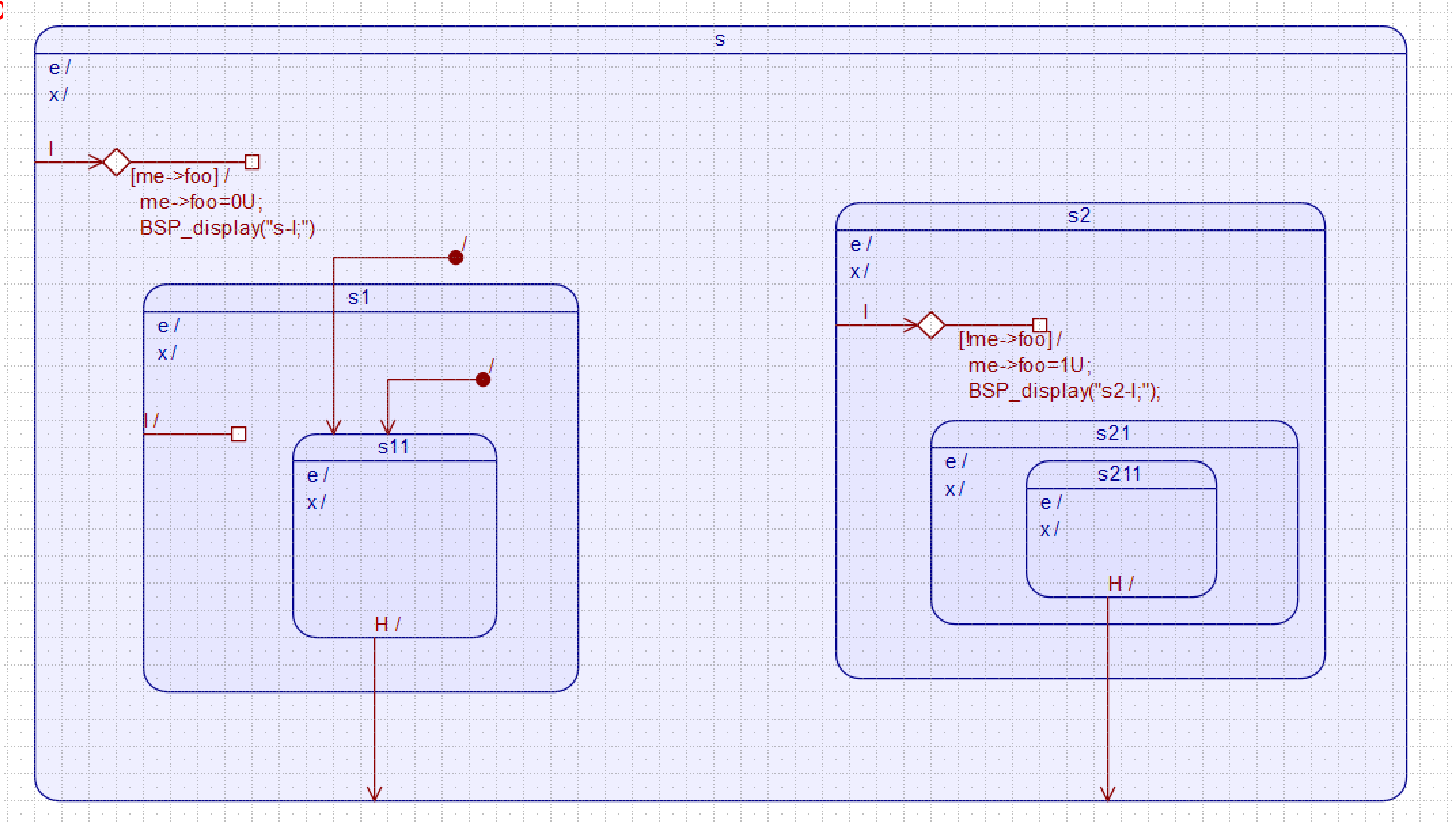


State ➔ S211

H→s211-H;s211-EXIT;s21-EXIT;s2-EXIT;s-INIT;s1-ENTRY;s11-ENTRY;

State ➔ S11

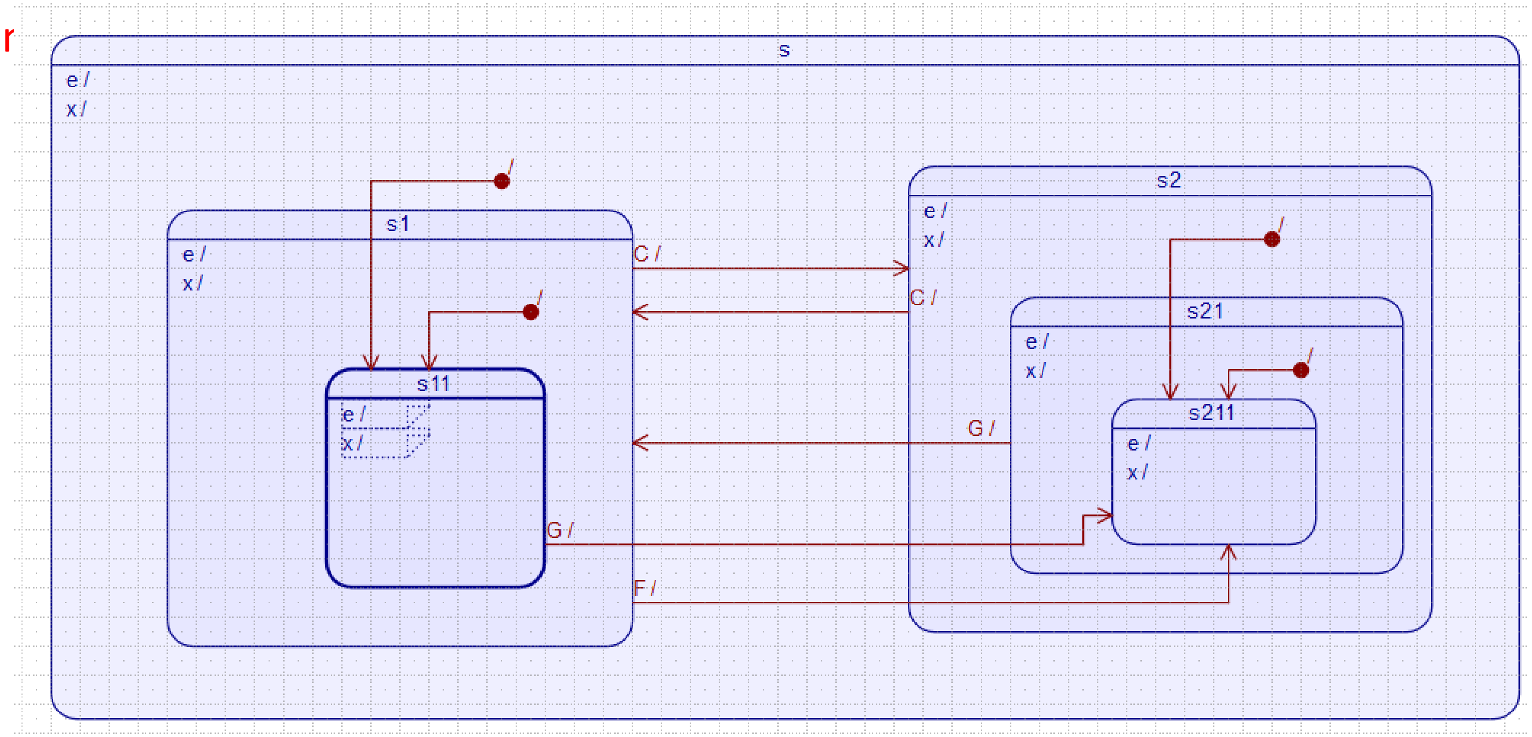
H→s11-H;s11-EXIT;s1-EXIT;s-INIT;s1-ENTRY;s11-ENTRY;



State → s211
I→s2-I;
I→s-I

State → s11
I→s1-I

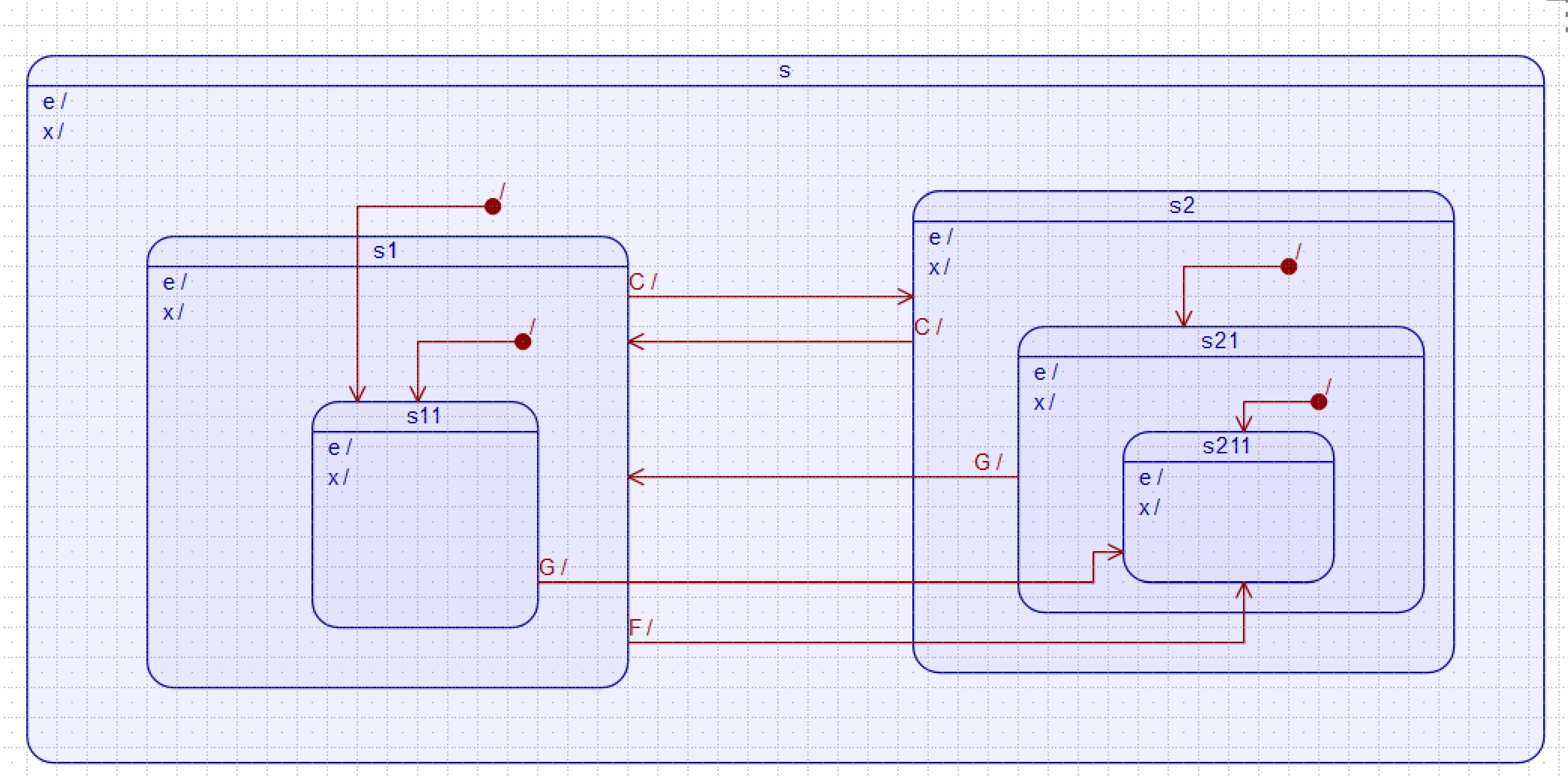
www.fastbitlab.cor

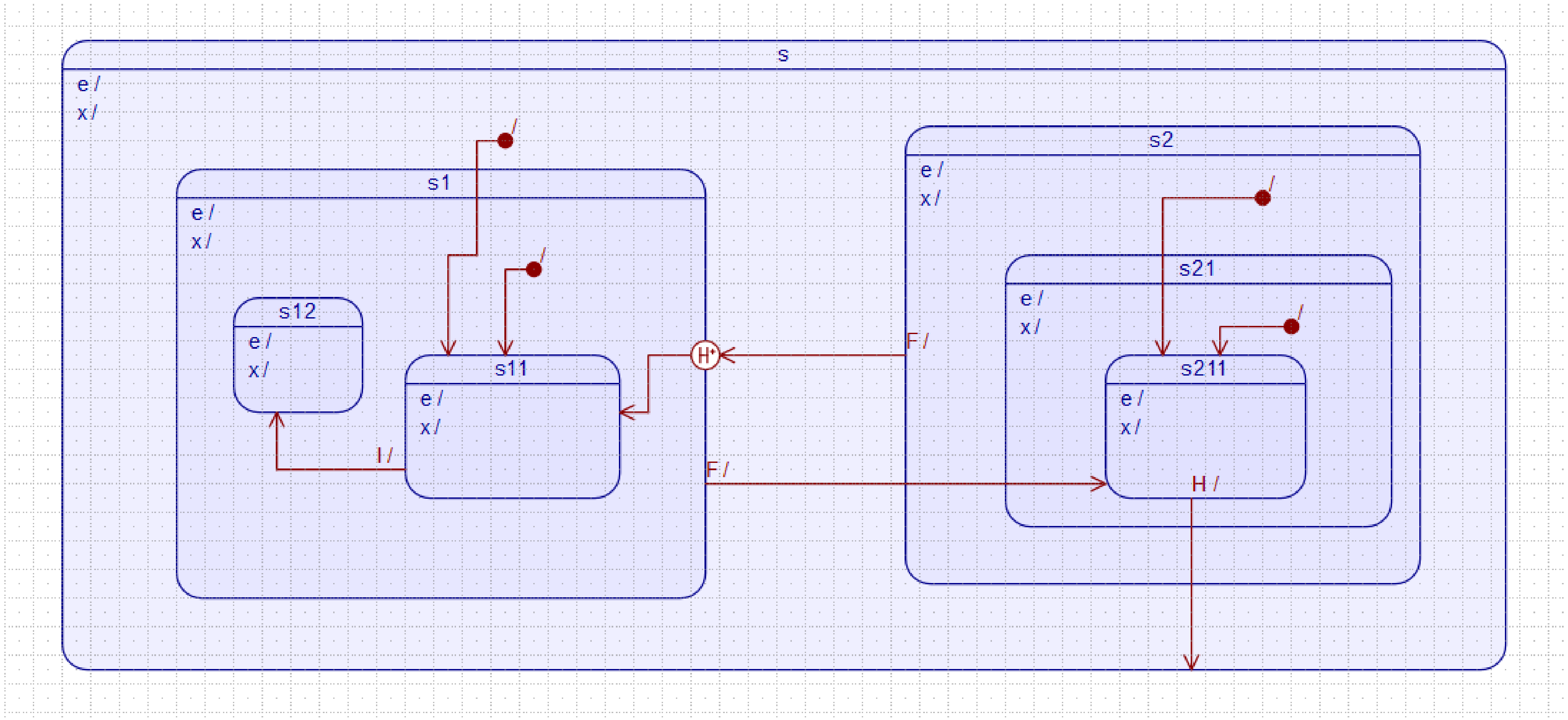


State ➔s211
C→s2-C;s211-EXIT;s21-EXIT;s2-EXIT;s1-ENTRY;s1-INIT;s11-ENTRY;

State ➔ s11
C→s1-C;s11-EXIT;s1-EXIT;s2-ENTRY;s2-INIT;s21-ENTRY;s211-ENTRY;

State ➔ s211
G→s21-G;s211-EXIT;s21-EXIT;s2-EXIT;s1-ENTRY;s1-INIT;s11-ENTRY;

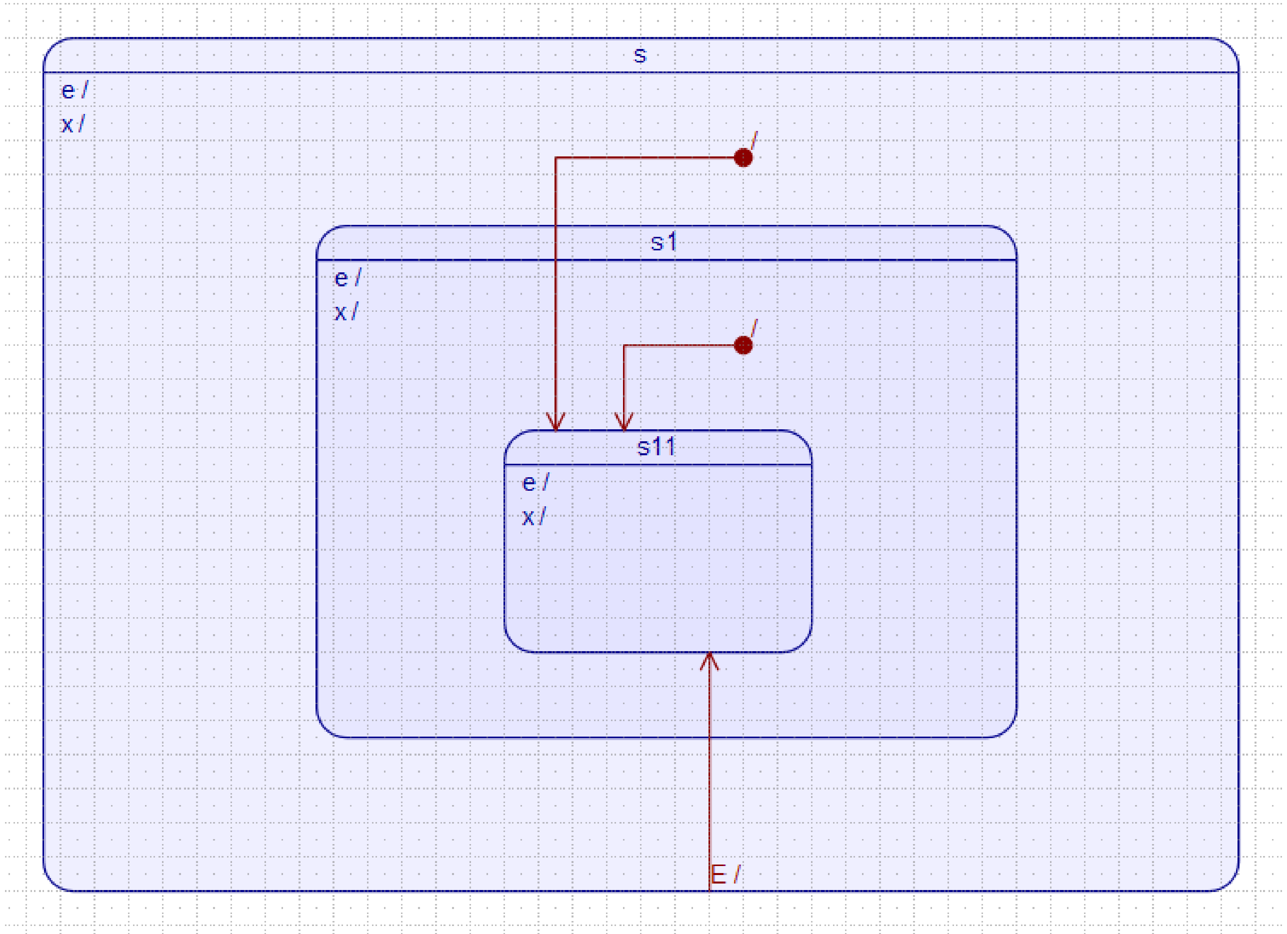

s
e /
x /
s1
e /
x /
s11
e /
x /
C /
C /
G /
G /
F /
s2
e /
x /
s21
e /
x /
s211
e /
x /



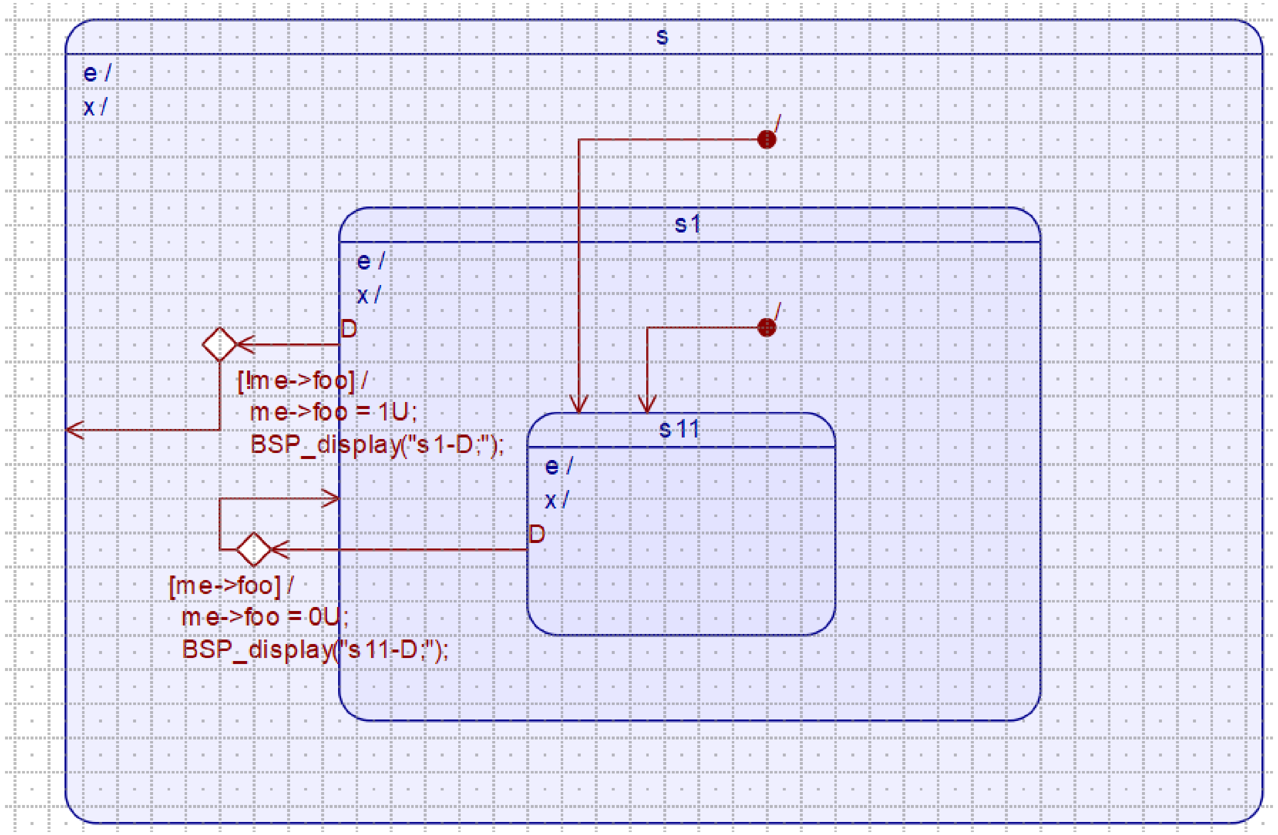
State ➔ s211
F→s2-F;s211-EXIT;s21-EXIT;s2-EXIT;s1-ENTRY;s12-ENTRY; (If History)

State ➔ s211
F→s2-F;s211-EXIT;s21-EXIT;s2-EXIT;s1-ENTRY;s11-ENTRY; (If No History)

www.fastbitlab.com



E→s-E;s11-EXIT;s1-EXIT;s1-ENTRY;s11-ENTRY;



State→s11, me->foo = 0
D→s1-D;s11-EXIT;s1-EXIT;s-INIT;s1-ENTRY;s11-ENTRY;
me->foo = 1;

State→s11, me->foo = 1
D→s11-D;s11-EXIT;s1-INIT;s11-ENTRY;

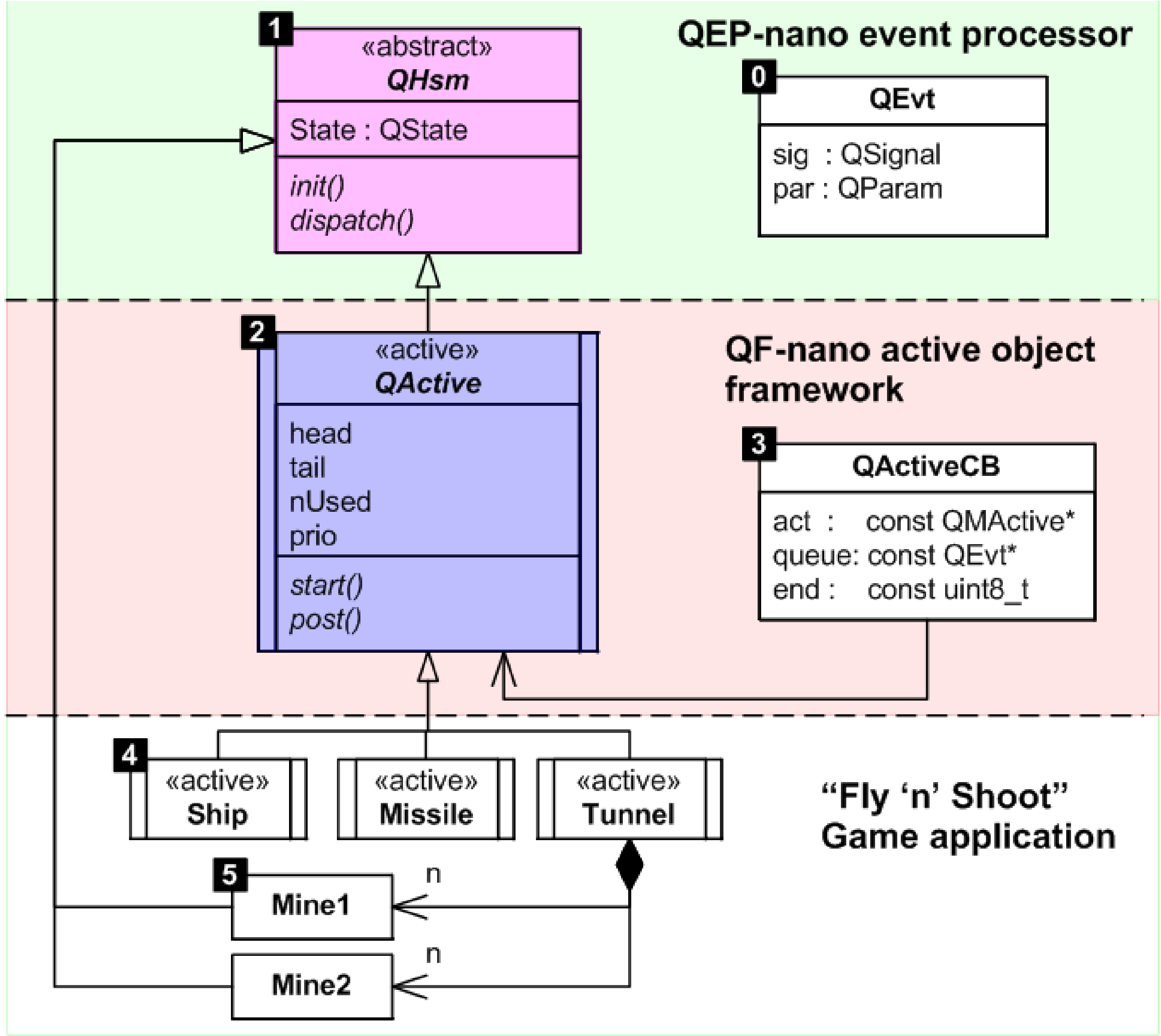
14.2.3.8.2 High-level (group) Transitions

Transitions whose source Vertex is a composite States are called *high-level* or *group* Transitions. If they are **external**, group Transitions result in the exiting of all substates of the composite State, executing any defined exit Behaviors starting with the innermost States in the active state configuration. In case of **local** Transitions, the exit Behaviors of the source State and the entry Behaviors of the target State will be executed, but not those of the containing State.

Code-Generation Directives

- Code-Generation Directives
- https://www.state-machine.com/qm/ce_directive.html

www.fastbitlab.com

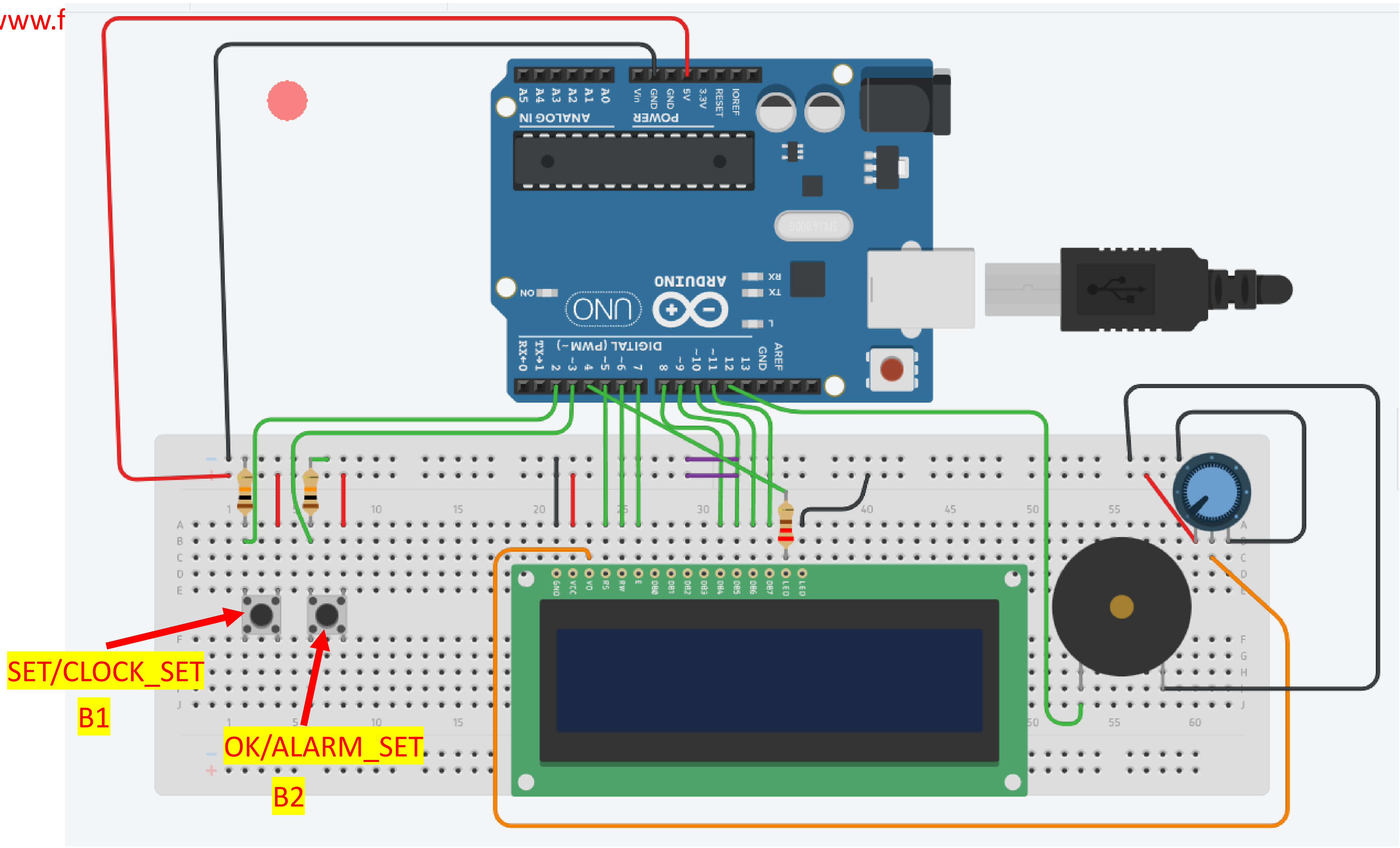


https://www.state-machine.com/qpn/struct.html

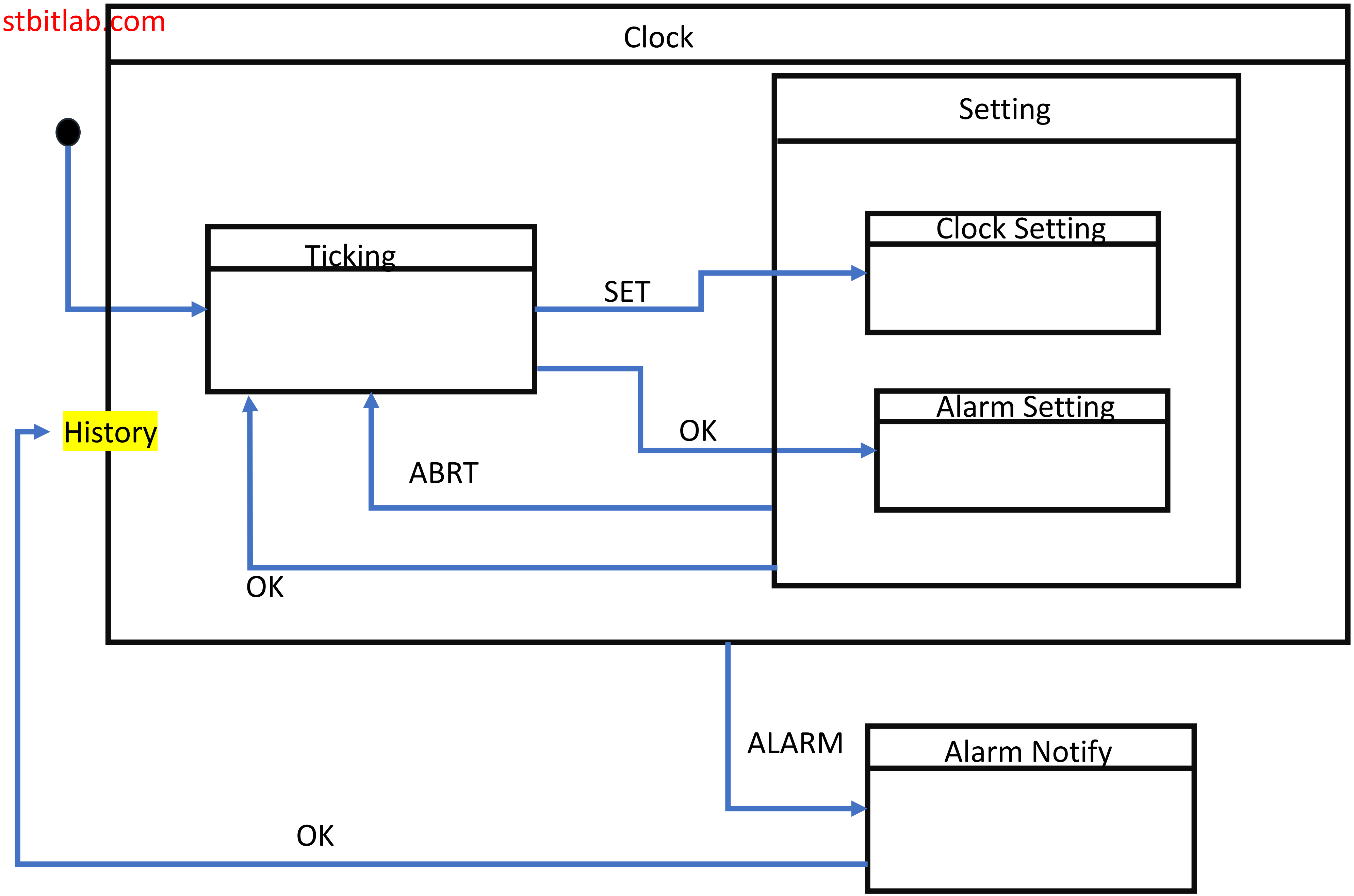
Exercise 007 : ClockAlarm

- Implement a real time clock using software
- Requirements
  - Show current time of the day
  - Clock setting
  - Alarm setting
  - Alarm notification
  - Show Date, month, year, day of the week [TODO]
  - Date setting[TODO]

www.f



Demo


Clock
Setting
Clock Setting
Ticking
SET
Alarm Setting
OK
History
ABRT
OK
ALARM
Alarm Notify
OK

States

Main application structure

QHSM

Derived from

```
/*Main application Structure*/
typedef struct {

    /*Updated by the timer ISR every 100ms*/
    uint32_t curr_time;
    /*Holds time information during clock/alarm setting*/
    uint32_t temp_time;
    /*User configured Alarm time*/
    uint32_t alarm_time;
    /*Alarm on/off status*/
    uint8_t  alarm_status;
    /*Time mode: 24H or 12H */
    uint8_t  time_mode;



} Clock_Alarm;
```

Signals

SET

OK

ABRT

ALARM

TICK (for every 25ms)

TODO

1) Create a new project 007ClockAlarm in VS code
2) Copy and paste lcd.c and lcd. h files attached with the lecture in VS code project folder ‘Src’

```
uint32_t curr_time;
```

Stores the time in number of 100 milliseconds
curr_time = 1 ; //100ms →00:00:00.1
curr_time = 9 ; //900ms →00:00:00.9
curr_time = 10 ; //1000ms →00:00:01.0
curr_time = 605; //60.5 seconds → 00:01:00.5
Note: This variable holds the time in 24H format

```
uint32_t alarm_time;
```

Stores the time in number of seconds
Note : This variable holds the time in 24H format

```
uint32_t temp_time;
```

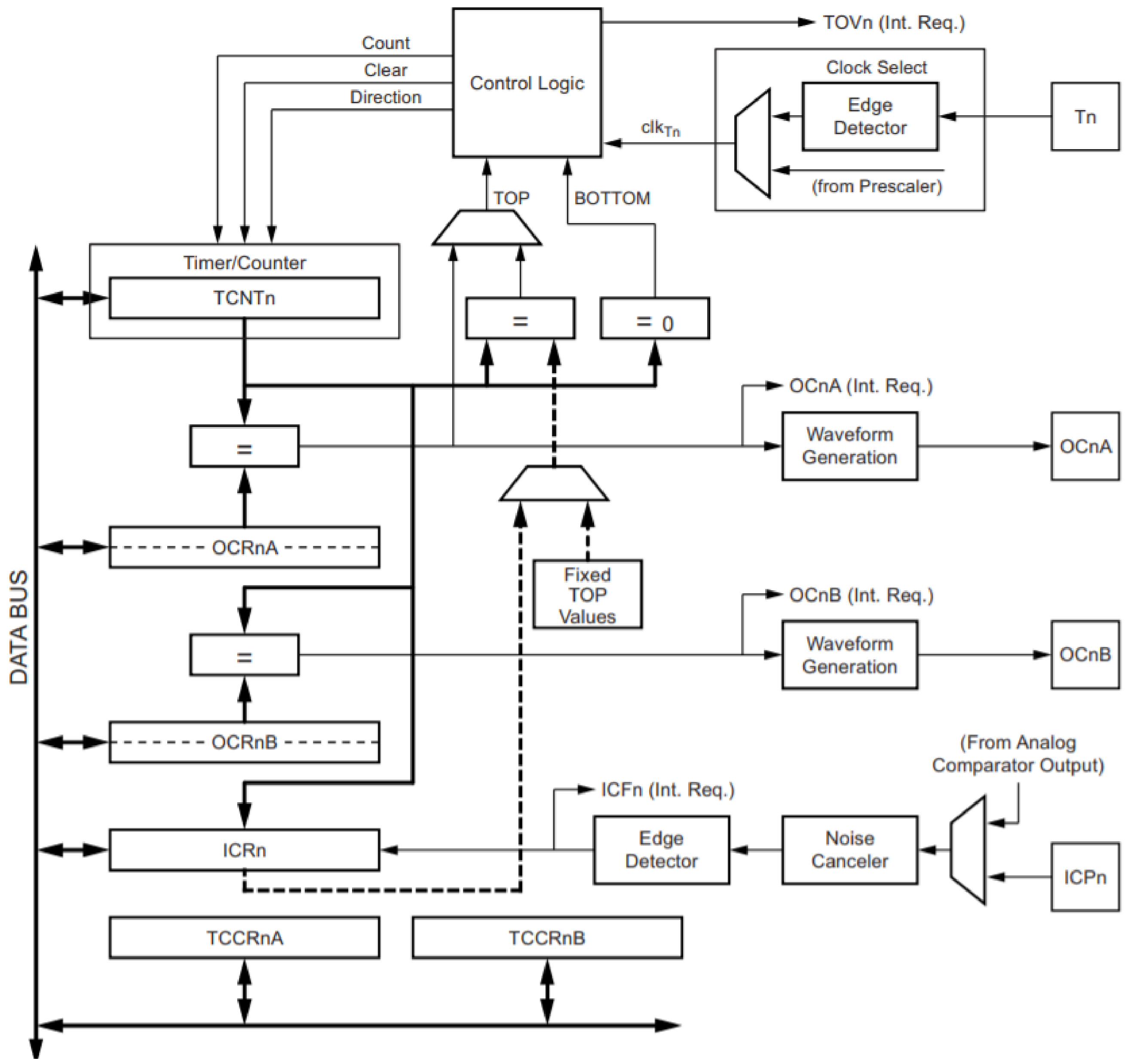
Stores the time in number of seconds
Note: Depending on the value of the 'time_mode' variable, this variable will hold time in 24H for 12H format

www.fastbitlab.com

ATmega328P Timer

16-bit Timer/Counter1

Figure 15-1. 16-bit Timer/Counter Block Diagram[1]



15.7 Output Compare Units

The 16-bit comparator continuously compares TCNT1 with the output compare register (OCR1x). If TCNT equals OCR1x the comparator signals a match. A match will set the output compare flag (OCF1x) at the next timer clock cycle. If enabled (OCIE1x = 1), the output compare flag generates an output compare interrupt. The OCF1x flag is automatically cleared when the interrupt is executed. Alternatively the OCF1x flag can be cleared by software by writing a logical one to its I/O bit location. The waveform generator uses the match signal to generate an output according to operating mode set by the waveform generation mode (WGM13:0) bits and compare output mode (COM1x1:0) bits. The TOP and BOTTOM signals are used by the waveform generator for handling the special cases of the extreme values in some modes of operation (see Section 15.9 "Modes of Operation" on page 100).

Source : Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Timer modes

- Refer : *15.9 Modes of Operation* in the datasheet

Timer/Counter clock (clk_{T1})

- In Arduino Uno the ATmega328P MCU is clocked by external 16MHz resonator
- $f_{CLK\_I/O}$ =16Mhz
- Timer1 count clock (clk_{T1}) = $f_{CLK\_I/O}$ / Prescaler

Configure Timer/counter1 Prescaler

15.11.2 TCCR1B – Timer/Counter1 Control Register B

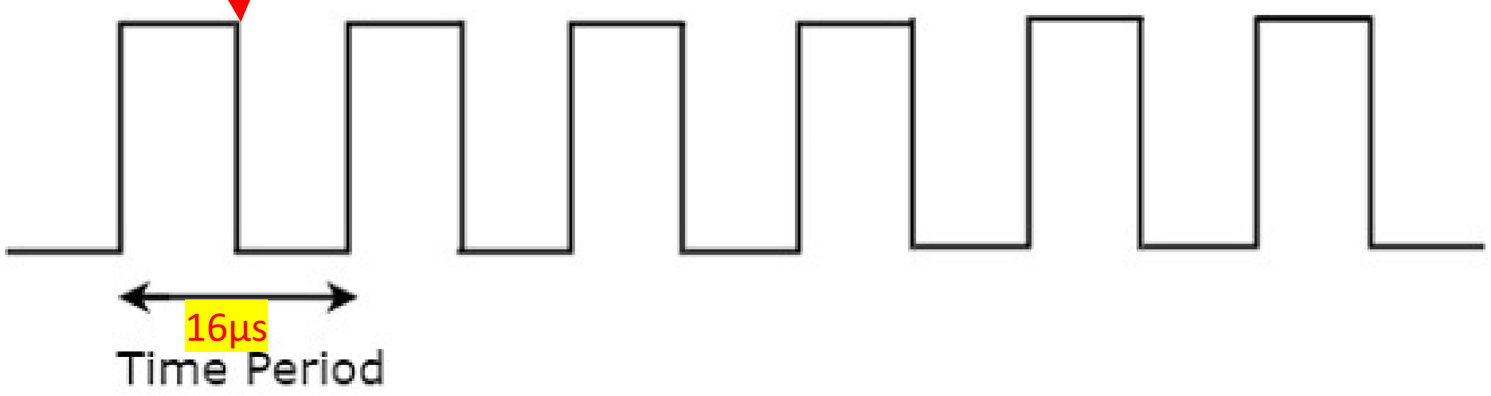
Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Timer/Counter clock (clk_{T1})

- In Arduino Uno the ATmega328P MCU is clocked by external 16MHz resonator
- $f_{CLK\_I/O}$ =16Mhz
- Timer1 count clock (clk_{T1}) = $f_{CLK\_I/O}$ / 256 ➔ 62.5KHz



Output Compare match value calculation

- Tick resolution = $1/ clk_{T1}$ = 1/62.5KHz = 16µs (Prescaler = 256)

- 16µs → 1 counter tick
- 100ms → ?

$$\frac{100ms}{16\mu s} = 6250$$

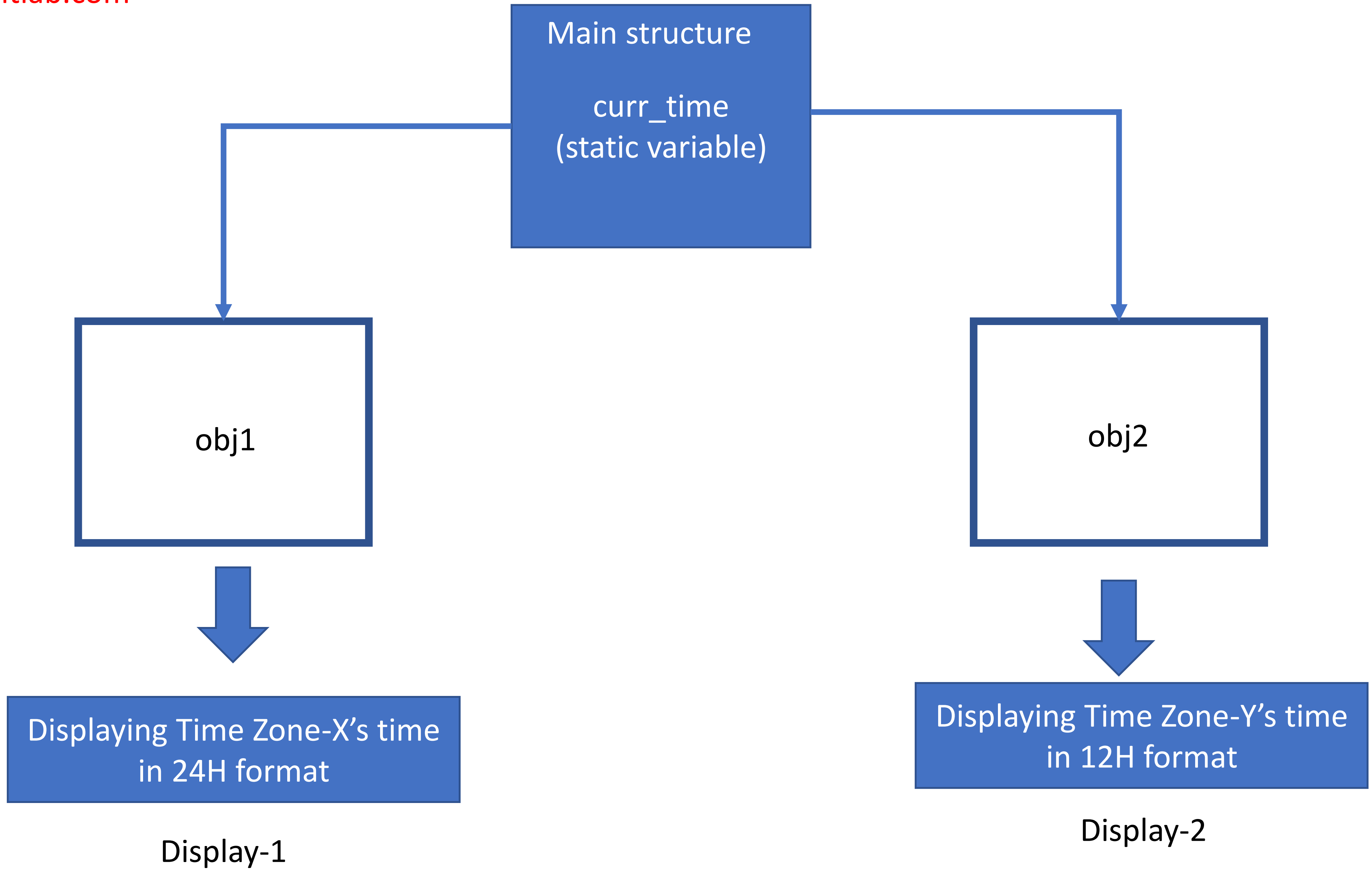
Output Compare match value = 6250-1

$$\frac{f_{CLK\_I/O} \times \text{Required delay}}{\text{Prescaler}} - 1$$

ISR

Define the TIMER1 COMPA ISR in ClockAlarm_SM.cpp

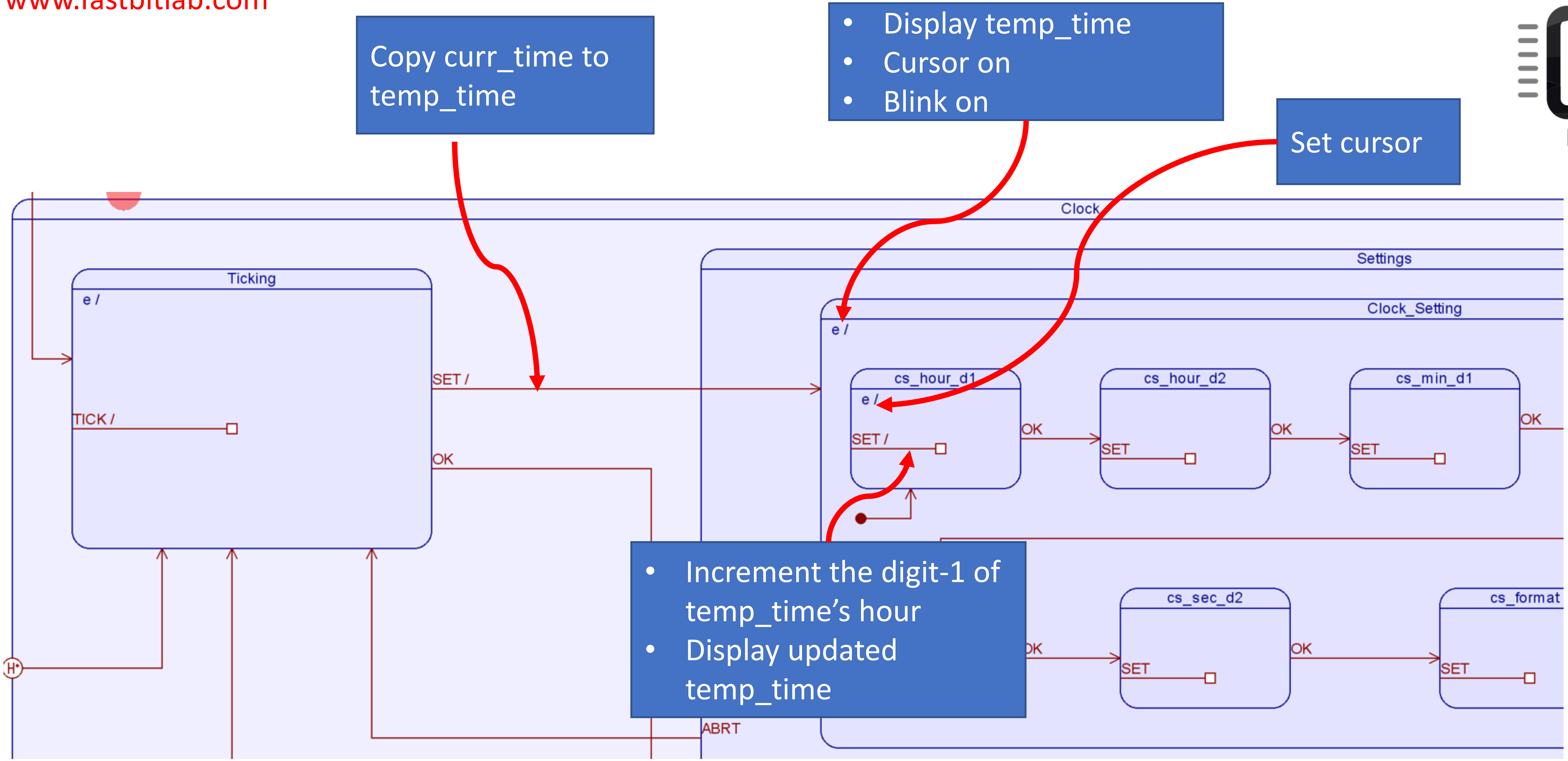
```
ISR(TIMER1_COMPA_vect){

}
```


Main structure
curr_time
(static variable)
obj1
obj2
Displaying Time Zone-X's time
in 24H format
Display-1
Displaying Time Zone-Y's time
in 12H format
Display-2

TODO

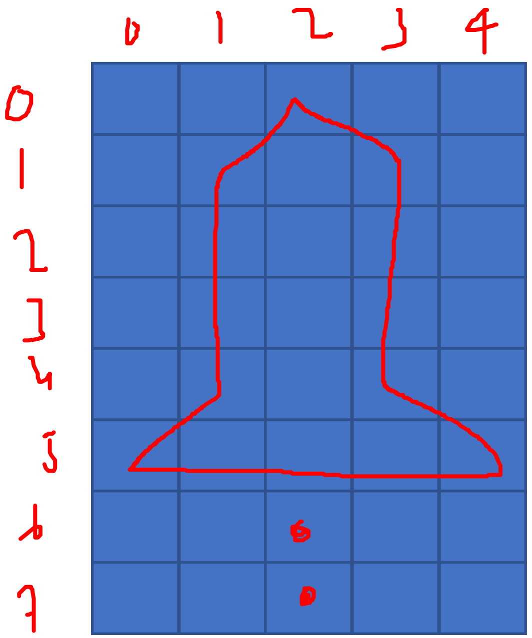
1. Add Timer1 setup code in main.cpp
2. Add a static class operation "update_curr_time" and increment the curr_time variable
3. Implement the ISR in ClockAlarm_SM.cpp and inside the ISR call the above class operation
4. Call the main application structure constructor from main.cpp
5. Add the initial transition action
    - Initialize the curr_time to 10:10:10
    - Initialize the alarm_time to 8:00:00
    - time_mode to 12H

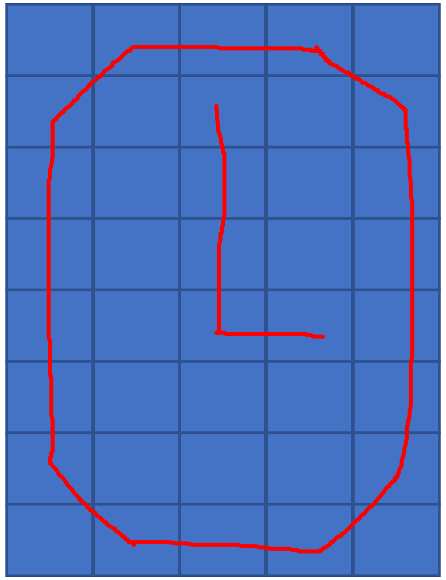
www.fastbitlab.com



www.fastbitlab.com

5 X 8



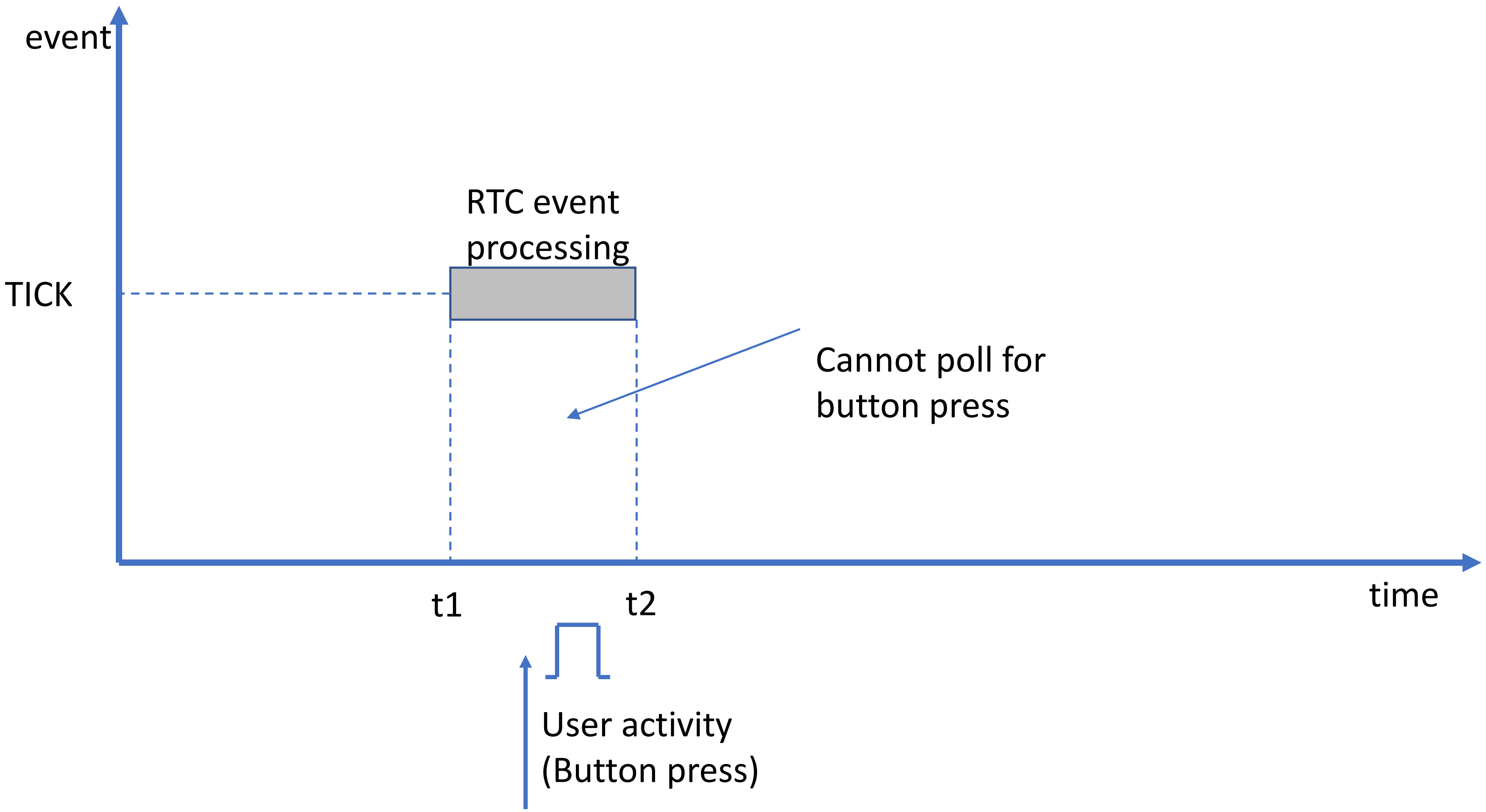


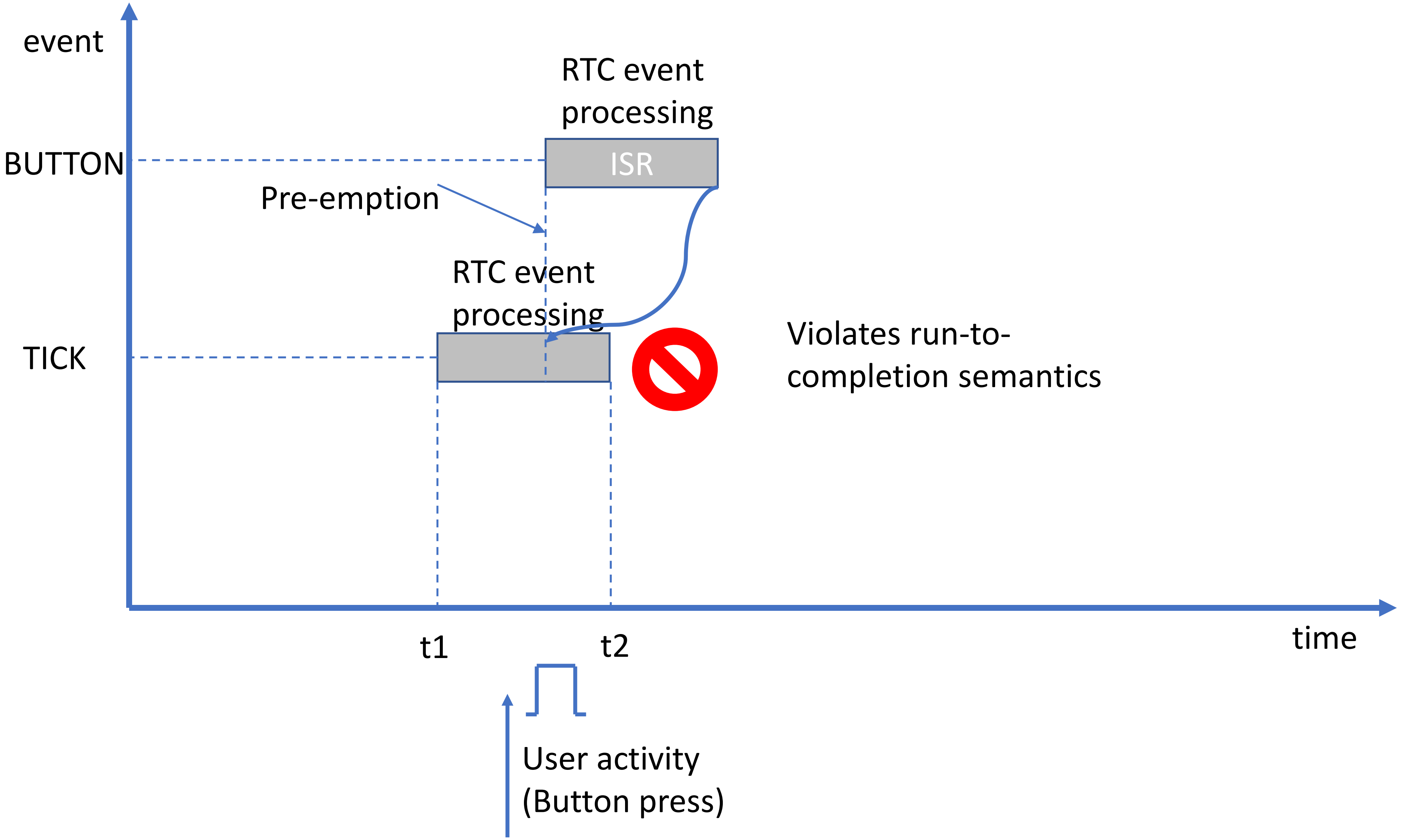
www.fastbit

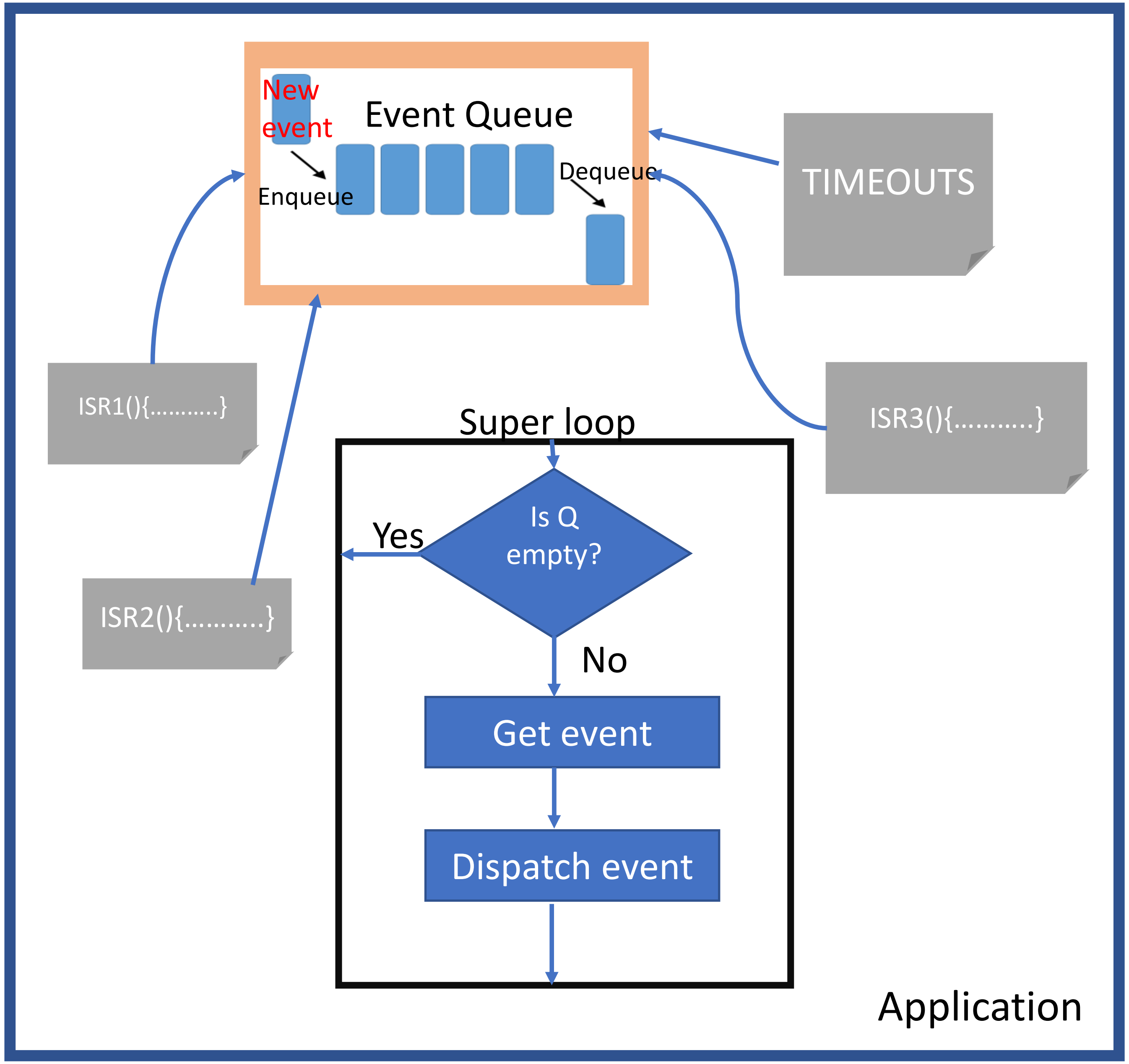
```
class SomeClassName{
    private:
        int somedata; /*non-static  attribute */
        /*static attribute . one copy for all the objects of this class */
        static int some_static_data;
    public:
        /*non-static operation or method*/
        void setdata(int data){
            somedata = data;
            this->somedata = data; //'this' pointer is available
        }
        /*static operation or  method*/
        static void static_function(int data){
            /*error: 'this' is unavailable for static member functions*/
            //this->somedata = data;
            // somedata = data; //error.
           some_static_data = data; //OK
        }
};

int main(void){
    SomeClassName Class1; //Instace of SomeClassName
    Class1.somedata = 10; //Error
    return 0;
}
```

Active objects

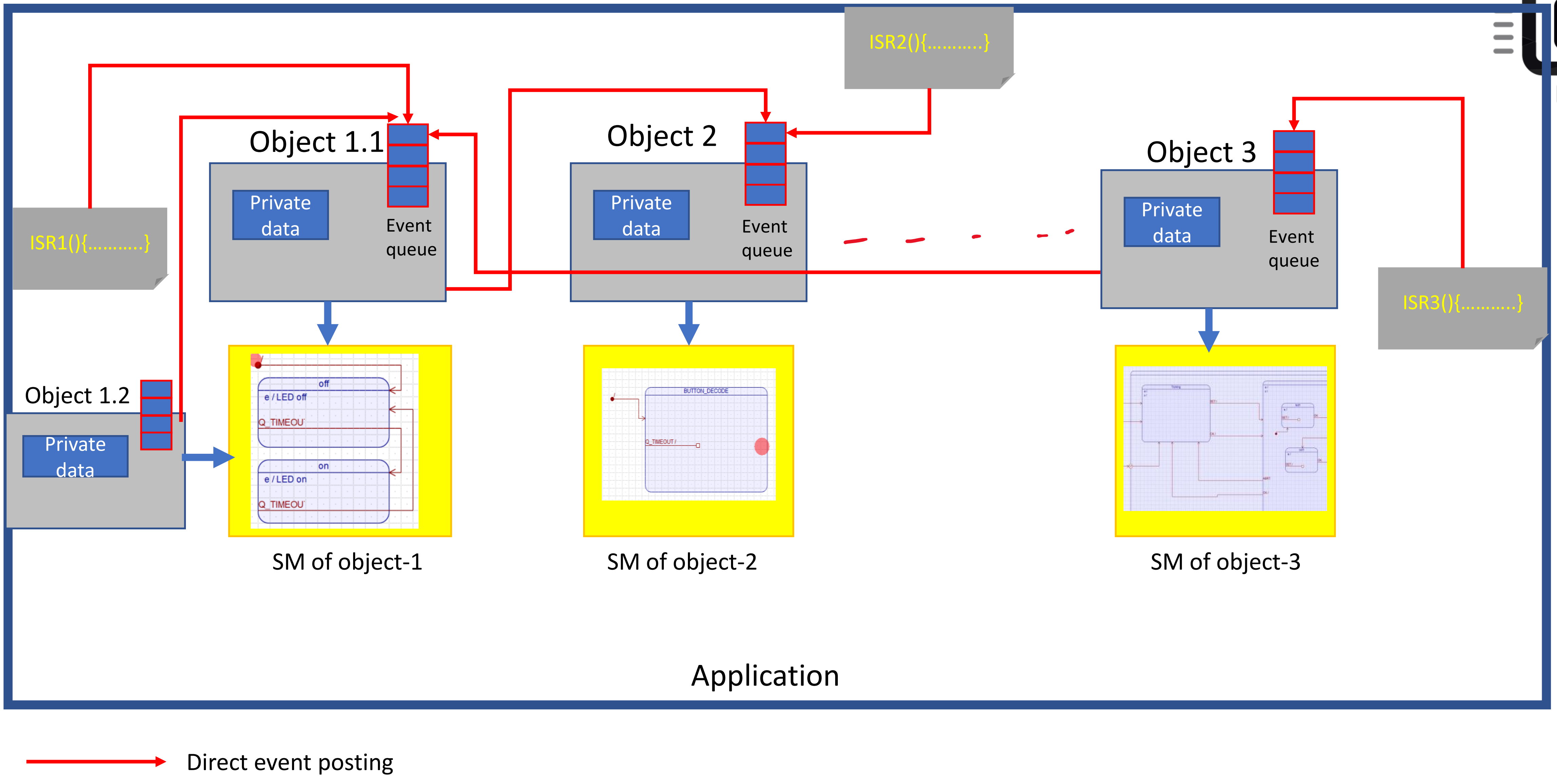
www.fastbitlab.com




event
RTC event
processing
BUTTON
ISR
Pre-emption
RTC event
processing
TICK
Violates run-to-
completion semantics
t1
t2
time
User activity
(Button press)


New
event
Event Queue
Enqueue
Dequeue
TIMEOUTS
ISR1(){..........}
ISR2(){..........}
ISR3(){..........}
Super loop
Yes
Is Q
empty?
No
Get event
Dispatch event
Application

www.fastbitlab.com

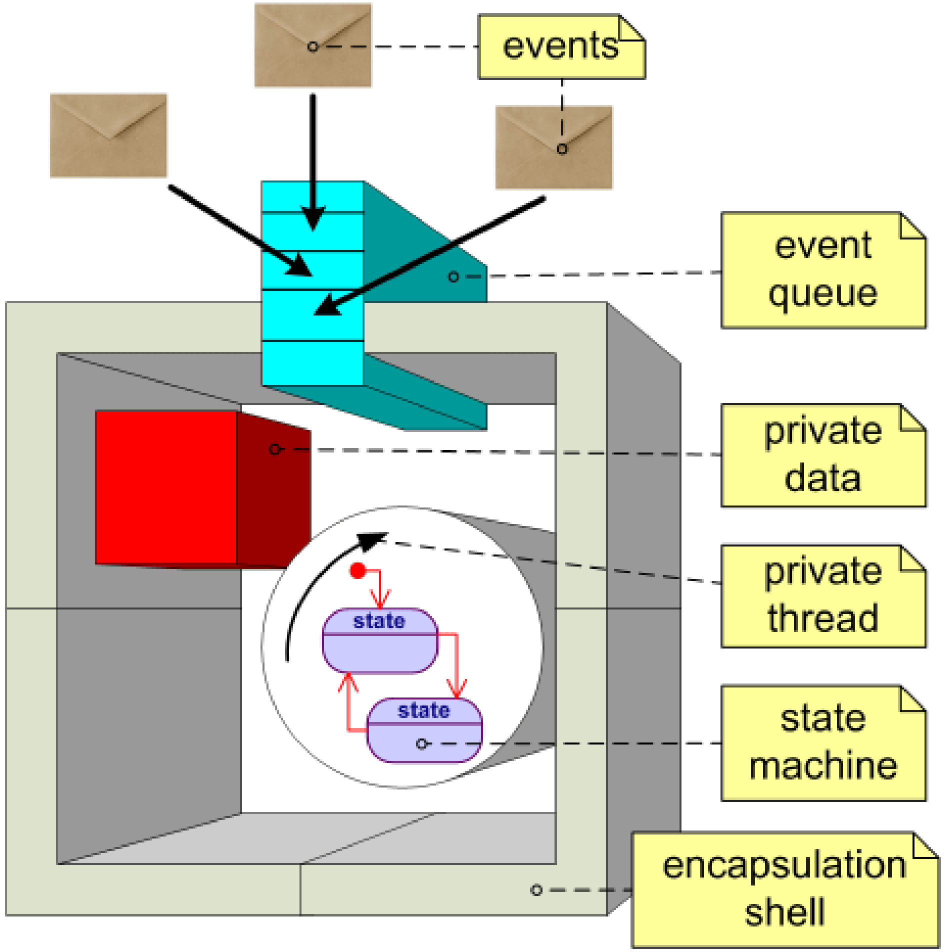


Active objects

[OMG® UML 2.5.1]

An active object is an object that, as a direct consequence of its creation, commences to execute its *classifierBehavior*, and does not cease until either the complete Behavior is executed or the object is terminated by some external object. (This is sometimes referred to as “the object having its own thread of control.”) The points at which an active object responds to communications from other objects is determined solely by the Behavior of the active object and not by the invoking object. If the *classifierBehavior* of an active object completes, the object is terminated.

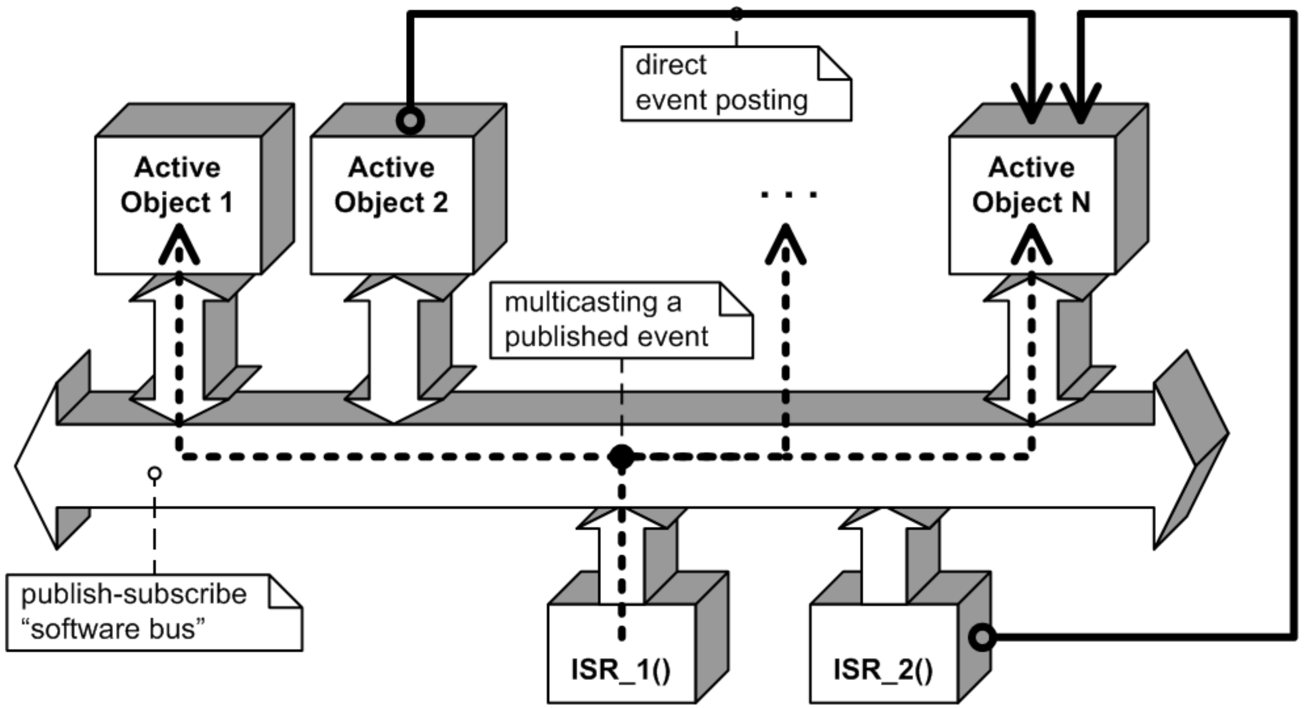
QP framework active object design paradigm



Source : https://www.state-machine.com/active-object

Event posting

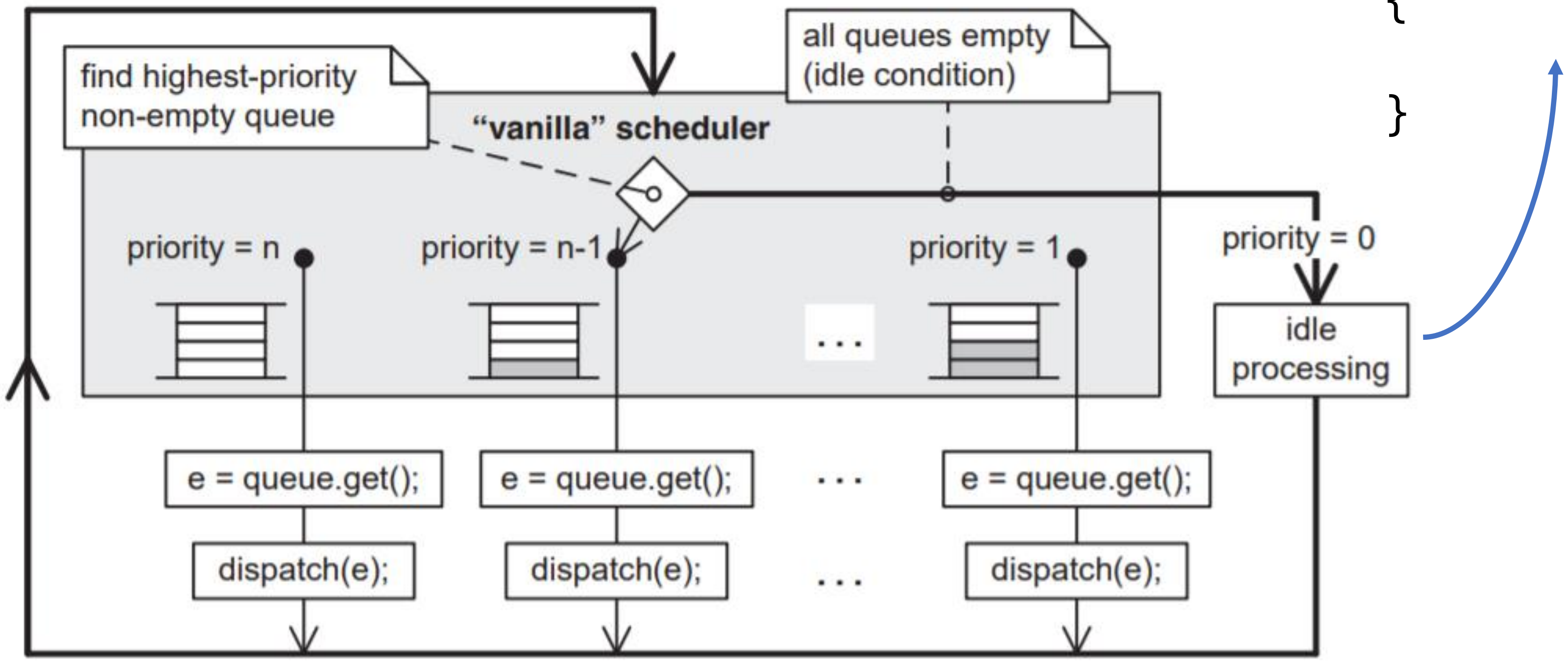
1) Direct event posting: event producer to event queue of the active objects
2) Publish and subscribe



Source : https://www.state-machine.com/active-object

- All events are delivered asynchronously
- Each active object processes events in run-to-completion (RTC) fashion

[1]




```
void QV_onIdle(void)
{

}
```

QF cooperative vanilla kernel scheduling
events to different AOs based on priority

[1] Source: *'Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems [SECOND EDITION] by Miro Samek '*

This PC > Documents > Arduino > libraries > qpn_avr > src

Name	Date modified	Type	Size
qassert	04-03-2020 00:07	C Header Source F...	13 KB
qepn	09-03-2020 06:34	C Source File	17 KB
qepn	16-06-2021 20:55	C Header Source F...	16 KB
qfn	09-03-2020 06:43	C Source File	16 KB
qfn	09-03-2020 06:48	C Header Source F...	16 KB
qfn_port	16-06-2021 19:47	C Header Source F...	4 KB
qpn	09-03-2020 06:49	C Header Source F...	4 KB
qpn_conf	28-03-2020 19:53	C Header Source F...	2 KB
qstamp	01-09-2017 03:38	C Source File	1 KB
qvn	09-03-2020 06:41	C Source File	6 KB
qvn	31-12-2019 04:52	C Header Source F...	3 KB

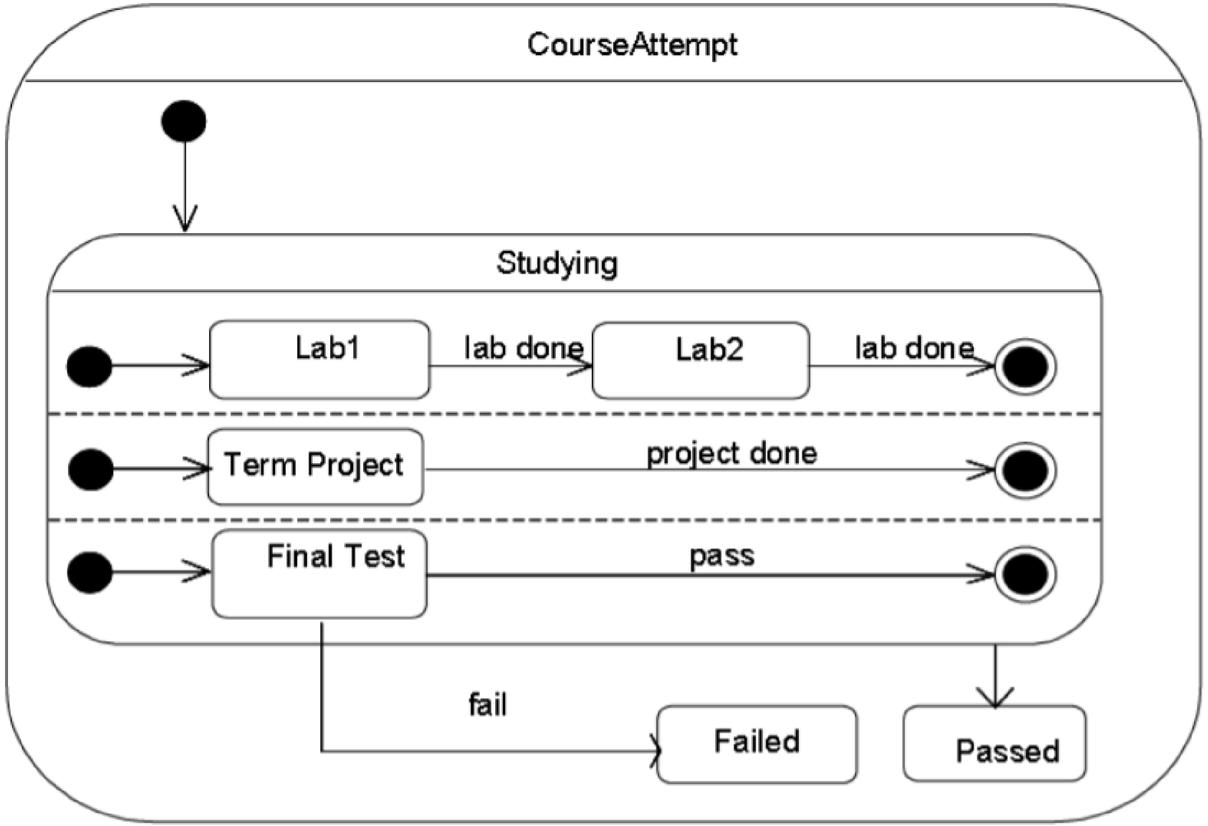


Figure 14.9 Composite State with Regions

Studying is a orthogonal state (composite state having 3 regions) having concurrent substates(not sequential)

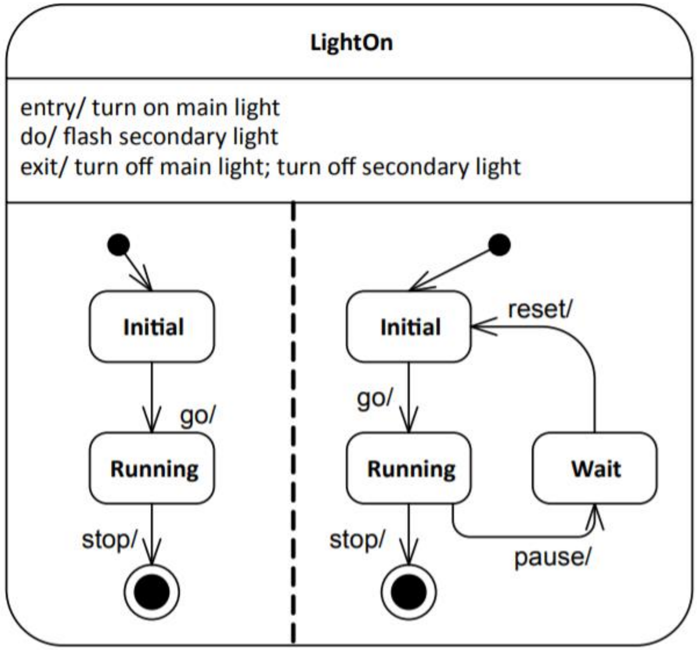
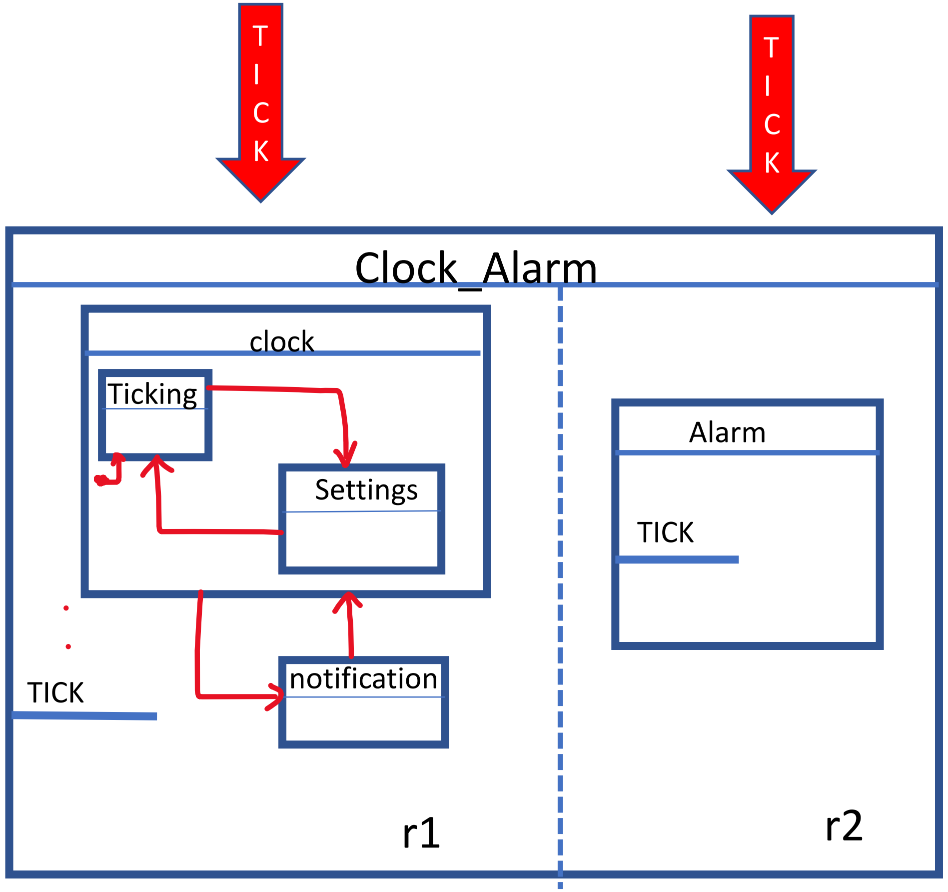


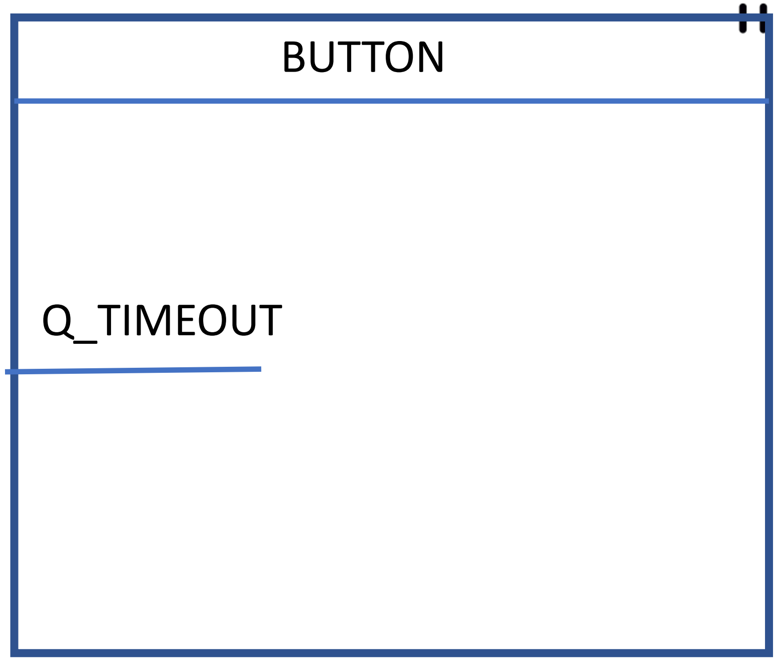
Figure 14.10 Composite State with two Regions and entry, exit, and do Behaviors

[OMG® UML 2.5.1]

www.fastbitlab.com

When there is need to achieve Independent functionalities, it may result in orthogonal regions





Clock_Alarm is an orthogonal state(composite state having 2 regions) and both the regions execute concurrently (r1 AND r2 decomposition)

SM of AO1

SM2

SM of AO2

Orthogonal regions

- QP framework does not support orthogonal regions directly
- Use orthogonal component state pattern which emulates orthogonal regions using composition(Strong aggregation) of classes

Orthogonal component state pattern

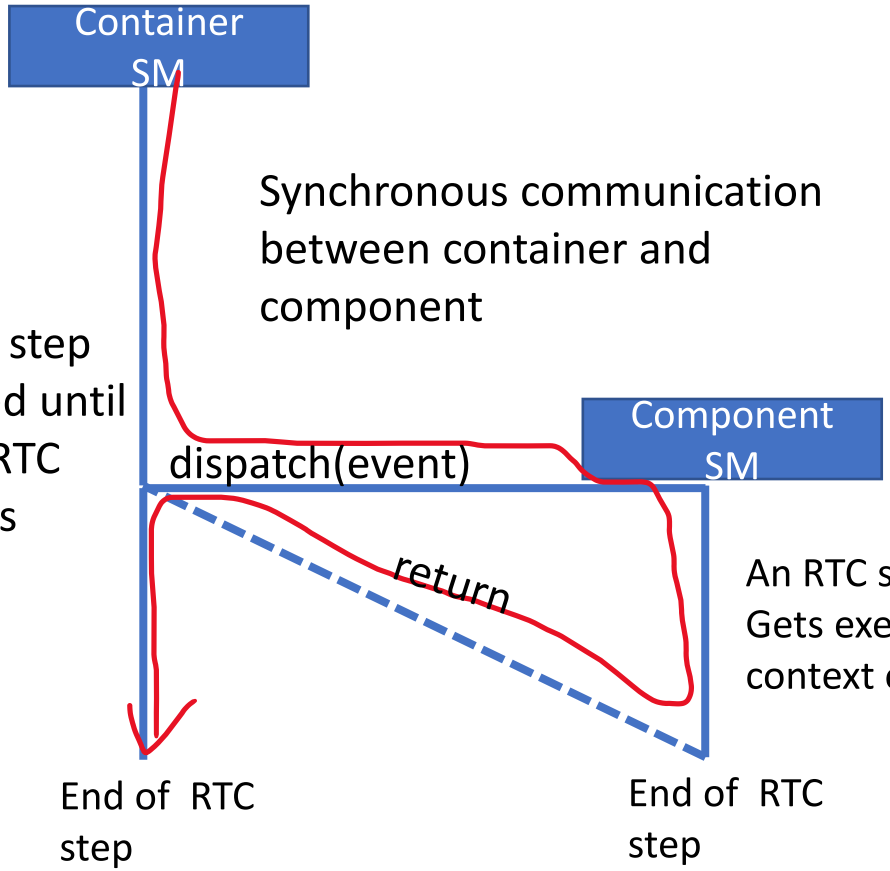
- Using state machines as components
- Identify the independent components in your project and use the technique 'composition' of classes
- For example, in our Clock Alarm exercise, Alarm could be independent exhibiting its own behaviour(State machine) and data (attributes such as alarm_time, alarm_mode, alarm_status)
- Alarm component can be reused with other containers of the application

Orthogonal component state pattern

- Communication
  - container to component
  - Component to container

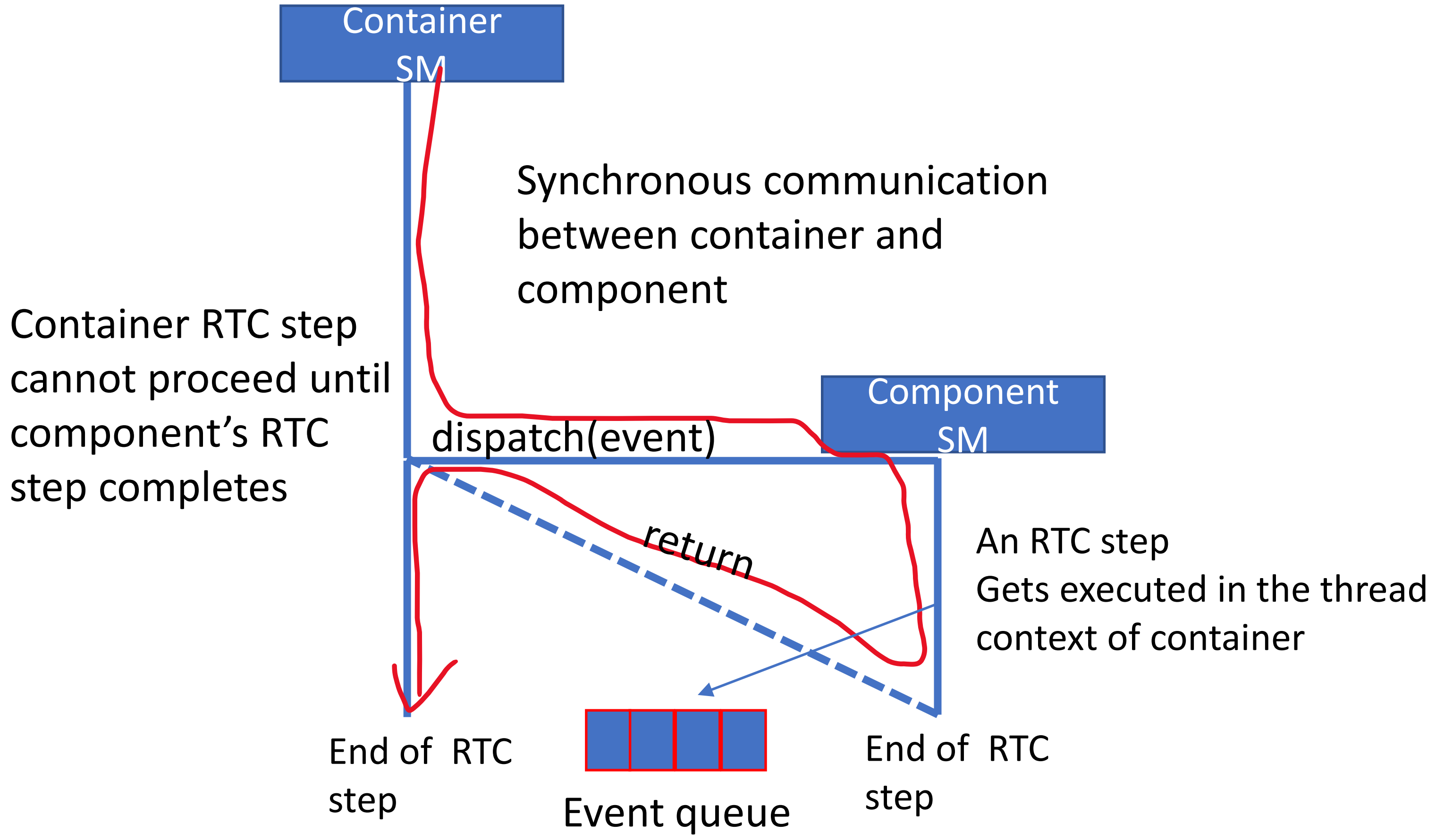
Synchronous communication between container and component

Container RTC step cannot proceed until component's RTC step completes

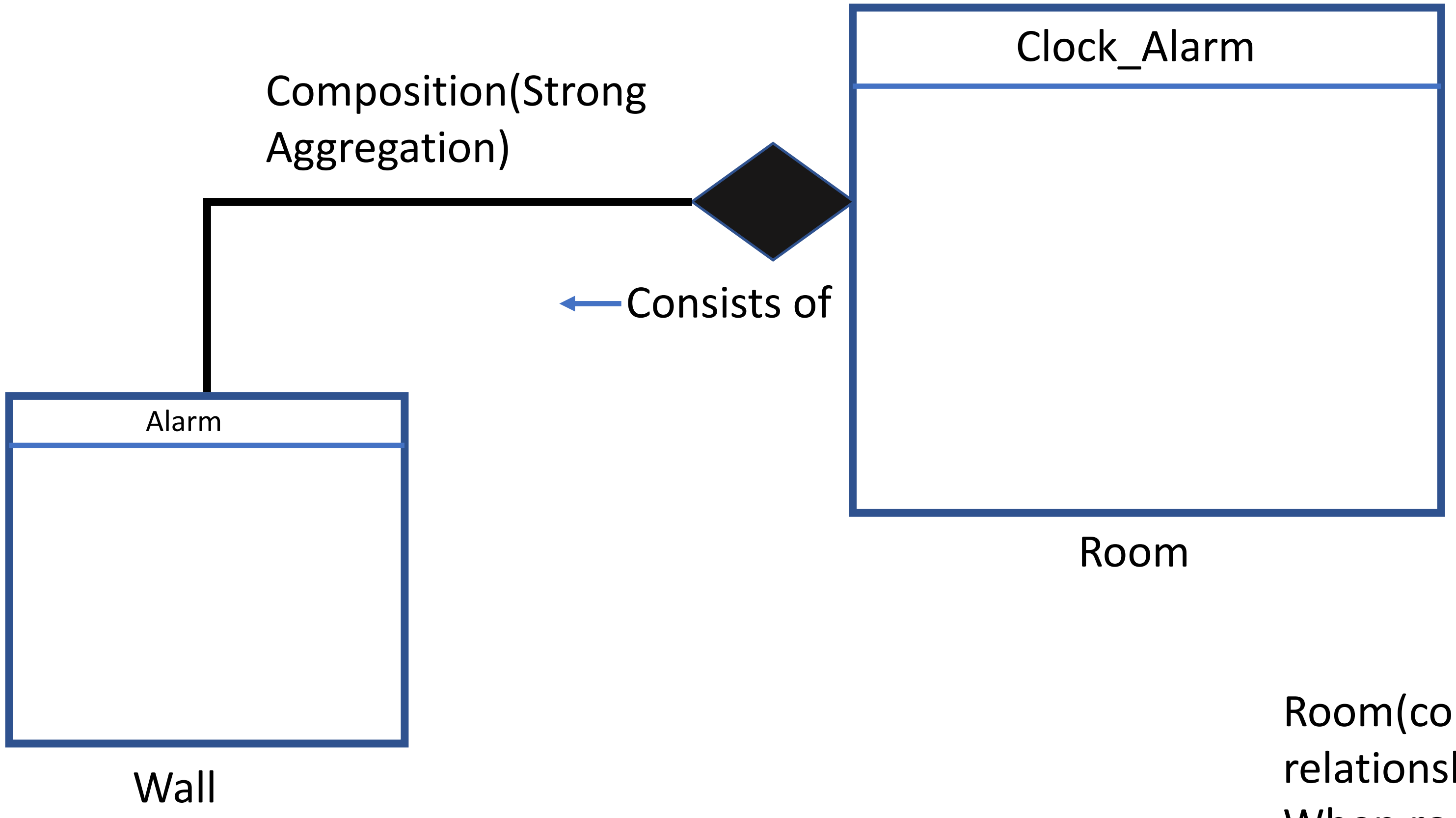


An RTC step
Gets executed in the thread context of container

container to component

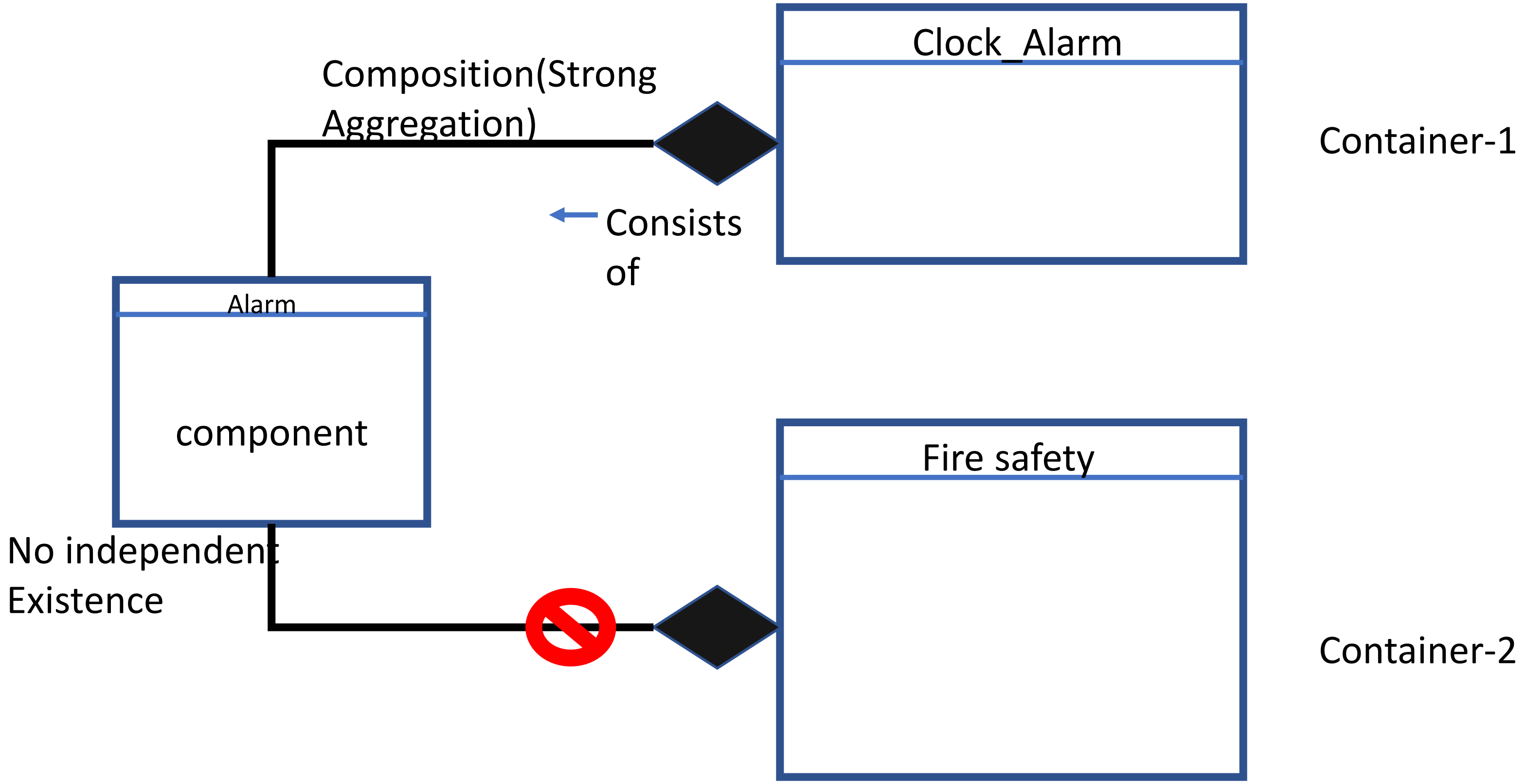


Component to container

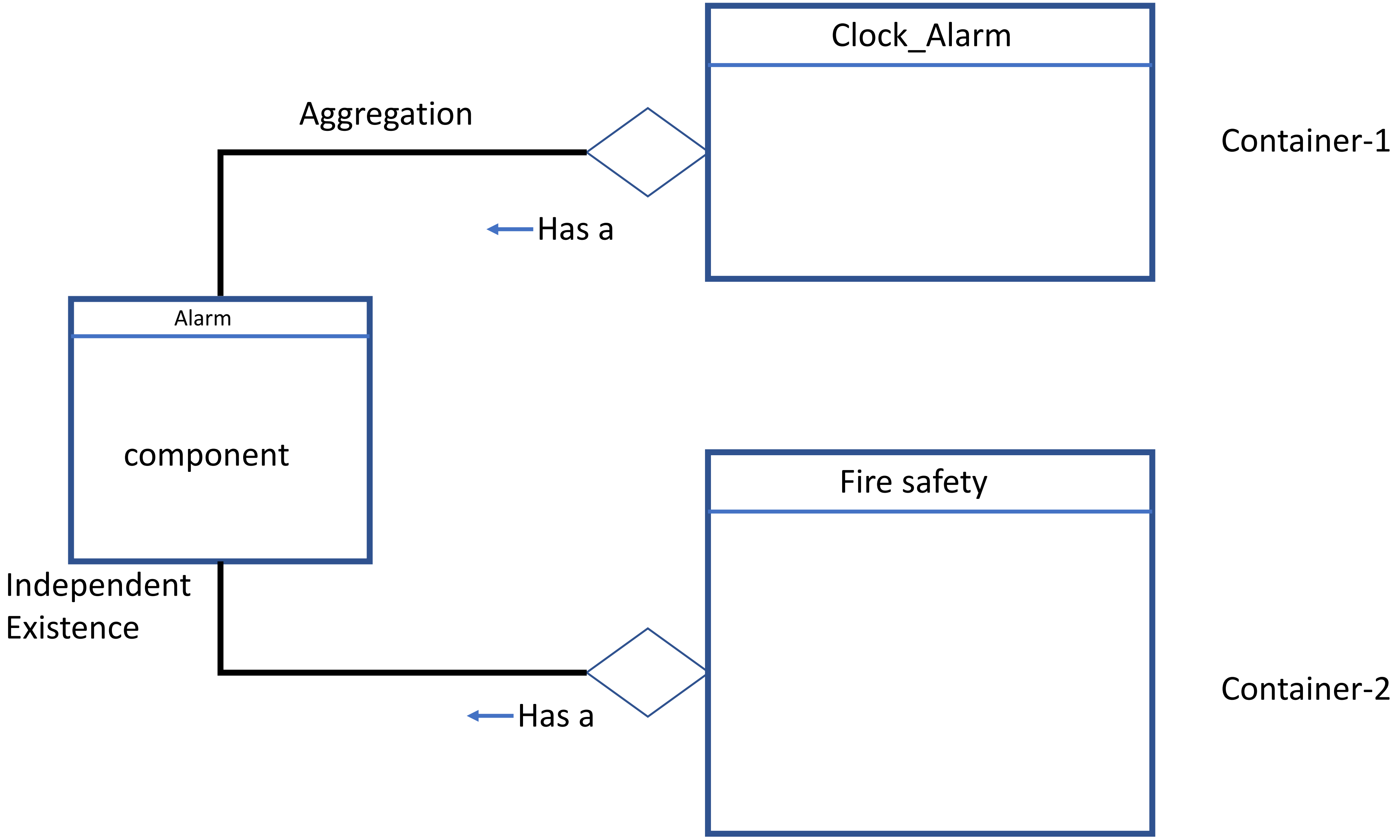


Room(container)-wall(component) relationship
When room dies , wall also dies
Room 'consist of' wall.

Strong aggregation means when the parent(container) dies child(component) also dies.
Parent manages memory for the child.

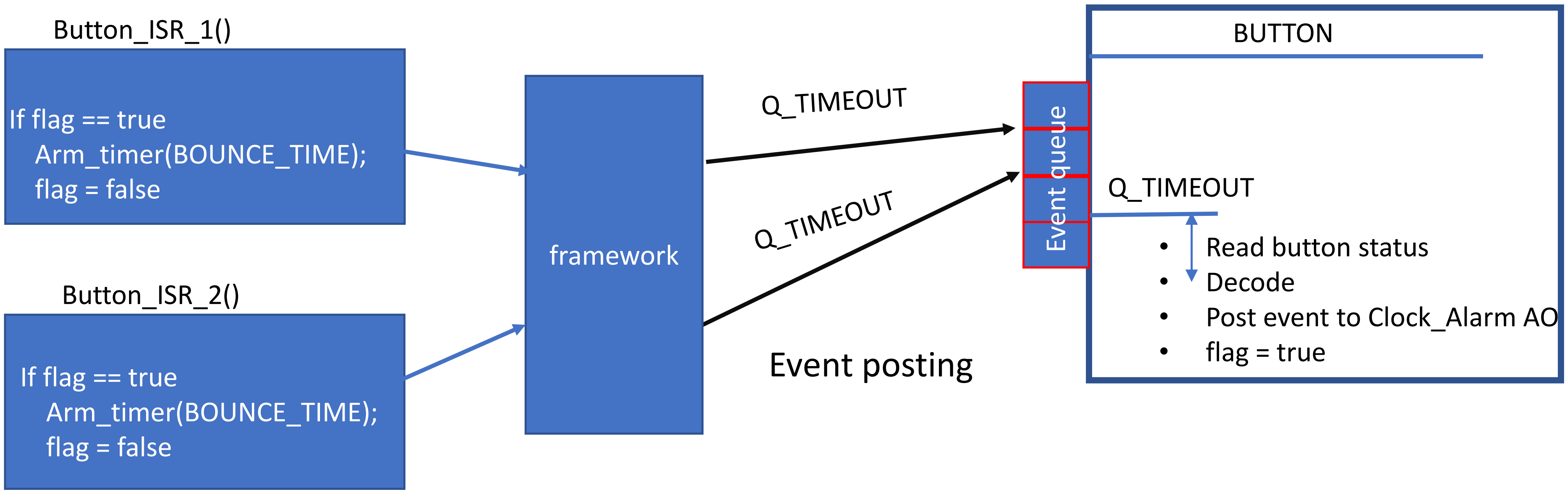


- Clock Alarm is an exclusive container of the Alarm component.
- The alarm component has no independent existence. Its existence only inside the Clock_Alarm
- When a container gets created, all components are also gets created


Clock_Alarm
Aggregation
Container-1
Has a
Alarm
component
Independent
Existence
Fire safety
Container-2
Has a

```c
/*.${HSMs::Clock_Alarm} ...................................................*/
typedef struct Clock_Alarm {
/* protected: */
    QActive super;

/* private: */
    uint32_t temp_time;
    uint8_t time_mode;
    uint8_t temp_format;
    uint8_t temp_digit;
    uint8_t curr_setting;
    Alarm alarm;

/* private state histories */
    QStateHandler hist_Clock;
} Clock_Alarm;
```

www.fastbitlab.com



QF (Active Object Framework)

- Create event queues(array of QEvt) for the active objects
- Create and initialize Active object control block (QActiveCB)
- Call QF_init(maxActive) to initialize the Active object framework of QP-nano
  - This also calls the init function of underlying kernel
- Call QF_run()
  - This also calls the scheduler of the kernel and never returns
- Call QF_tickXISR() from your application's tick ISR