Modern Embedded Software Goes Beyond the RTOS

Embedded Online Conference

www.embeddedonlineconference.com

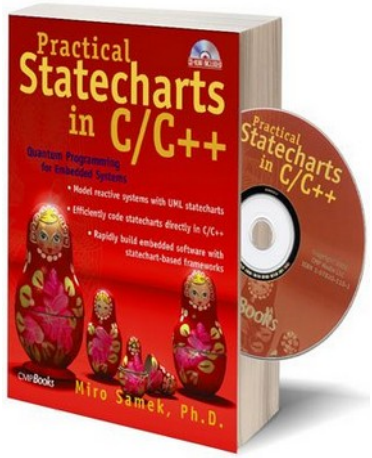
Miro Samek, Ph.D.

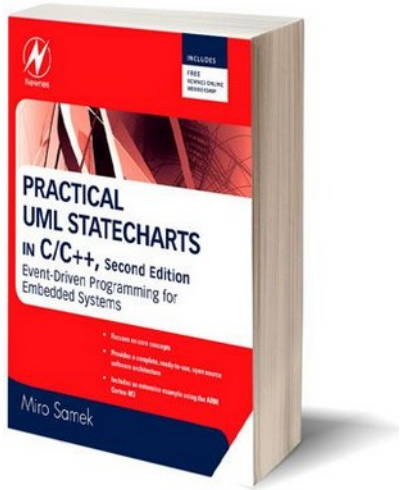
THE SPEAKER

Miro Samek, Ph.D.

CEO, Quantum® LeaPs
Modern Embedded Software

Dr. Miro Samek is an expert on programming modern real-time embedded (RTE) systems, a book author, conference speaker, and a YouTube teacher.





AGENDA

0 Challenges of Embedded Software Development

1 Traditional Sequential Programming & RTOS

2 Best Practices for Concurrency & Active Object Pattern

3 Event-Driven Paradigm Shift & RTOS vs. Framework

4 Hierarchical State Machines

5 Software Modeling & Code Generation

0

CHALLENGES OF EMBEDDED SOFTWARE DEVELOPMENT

- "Getting off the ground"
- Handling concurrency
- "Spaghetti code"


Spaghetti
Concurrency
Getting off the ground

HOBBYIST'S PERSPECTIVE

PROFESSIONAL DEVELOPER'S PERSPECTIVE

AGENDA

0 Challenges of Embedded Software Development

1 Traditional Sequential Programming & RTOS

2 Best Practices for Concurrency & Active Object Pattern

3 Event-Driven Paradigm Shift & RTOS vs. Framework

4 Hierarchical State Machines

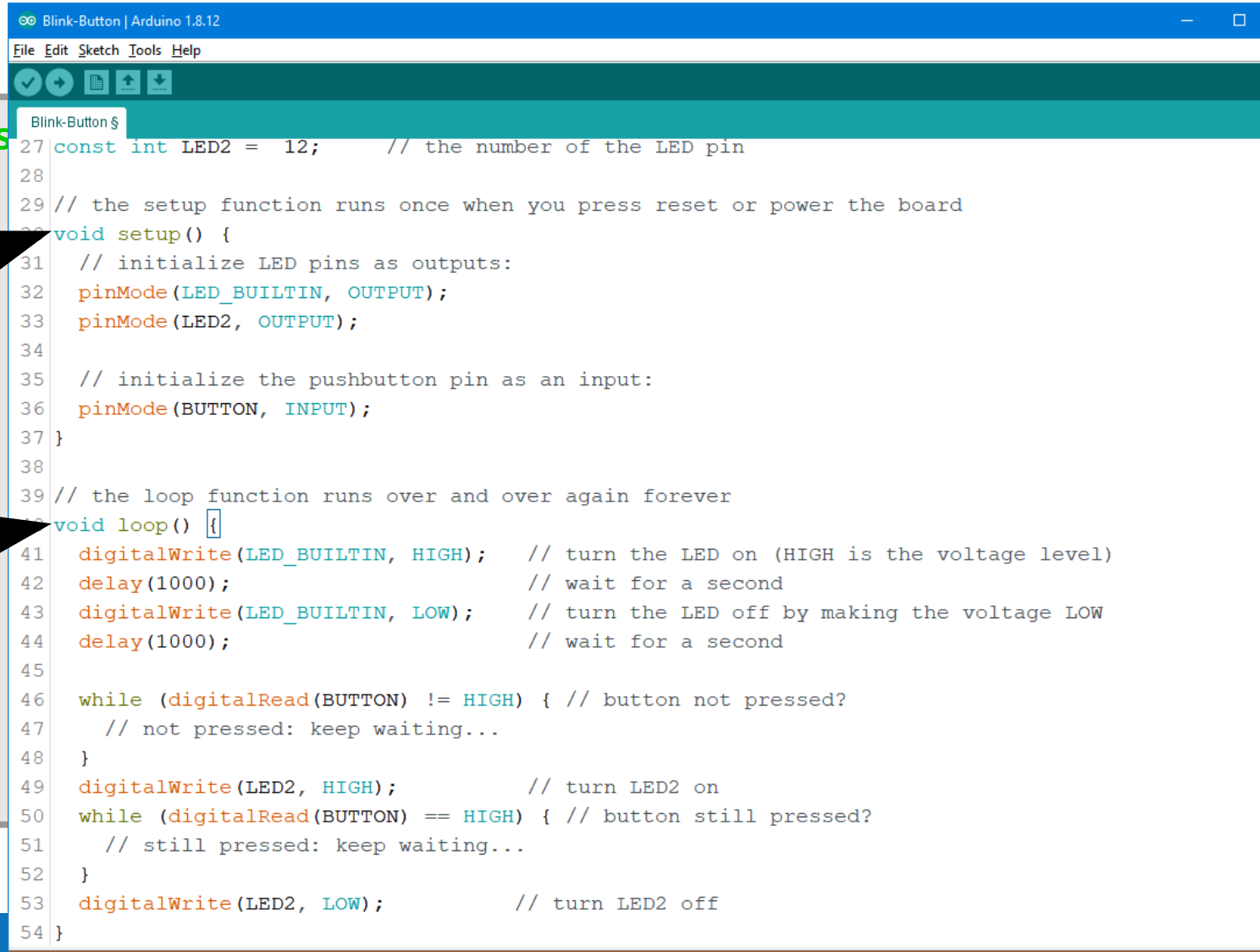
5 Software Modeling & Code Generation

"SUPERLOOP"

```
// main.cpp - Main loop for Arduino
#include <Arduino.h>
. . .
int main(void)
{
    . . .
    setup();
    . . .
    for (;;) { // for-ever
        . . .
        loop();
        . . .
    }
}
```

EXTENSIBILITY OF "SUPERLOOP"

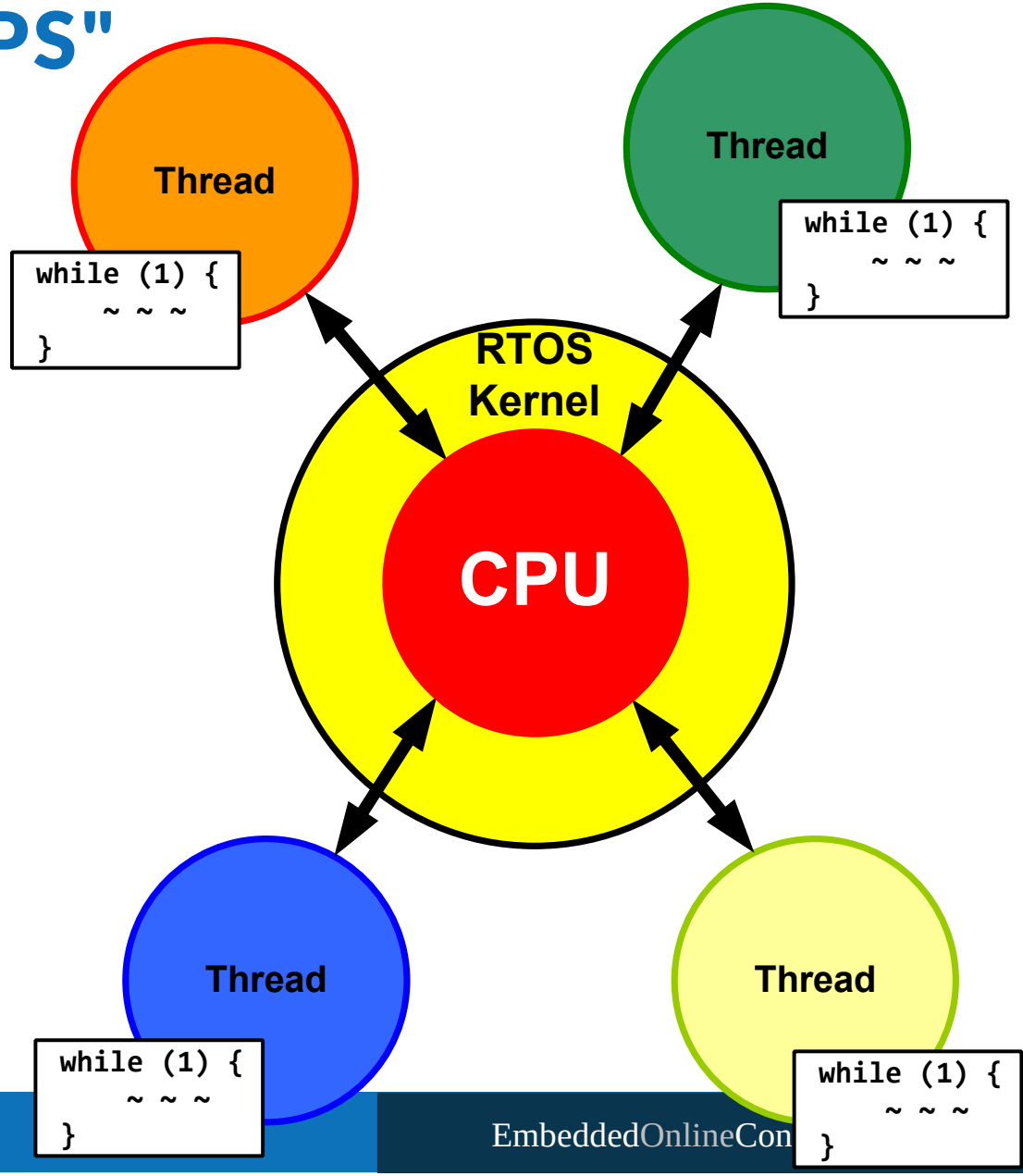
```
// main.cpp - Main loop for Arduino s
#include <Arduino.h>
. . .
int main(void)
{
    . . .
    setup();
    . . .
    for (;;) { // for-ever
        . . .
        loop();
        . . .
    }
}
```



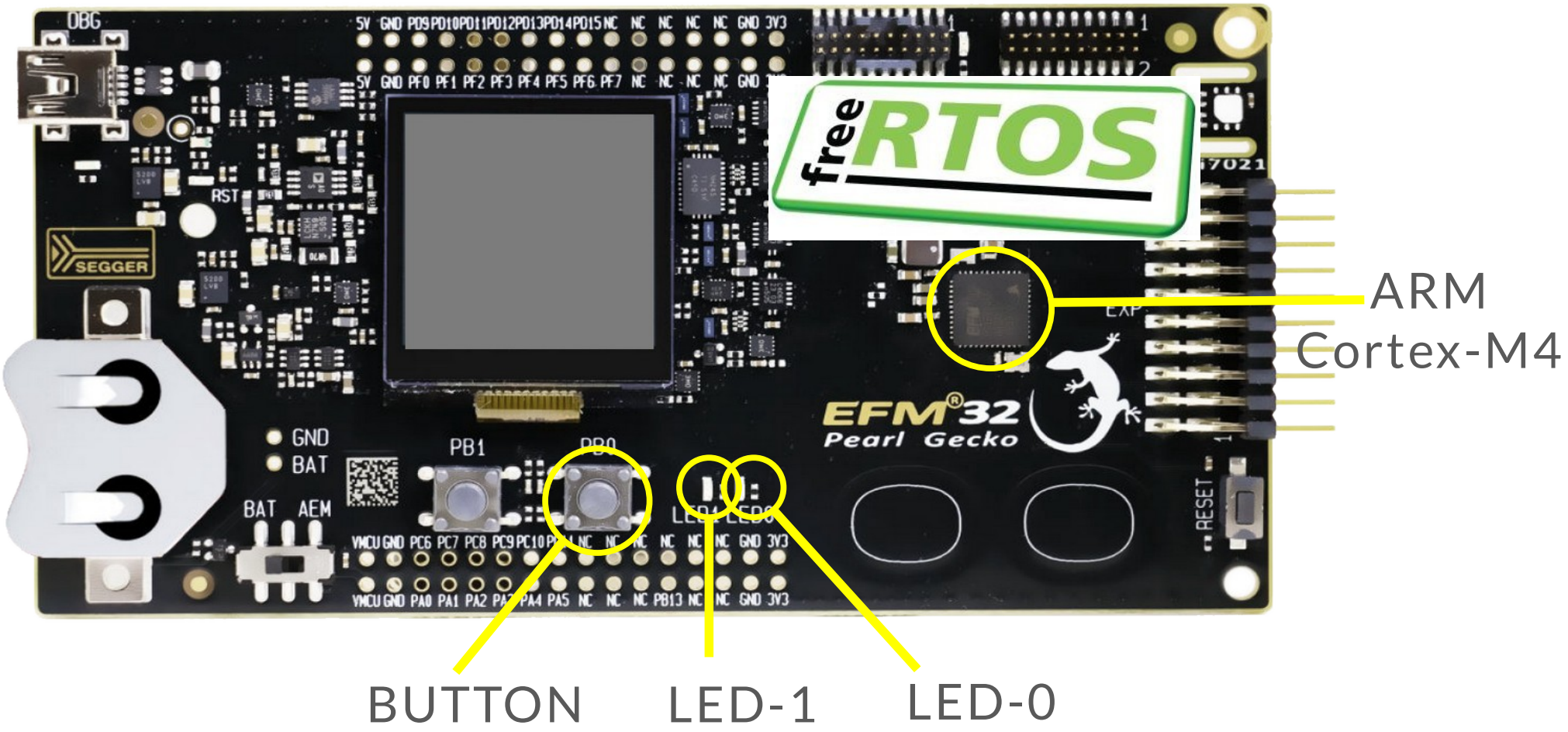
RTOS: MULTIPLE "SUPERLOOPS"

RTOS Kernel:
Software that extends the basic "superloop" architecture by allowing you to run multiple "superloops" (called **Threads** or **Tasks**) on a single CPU.

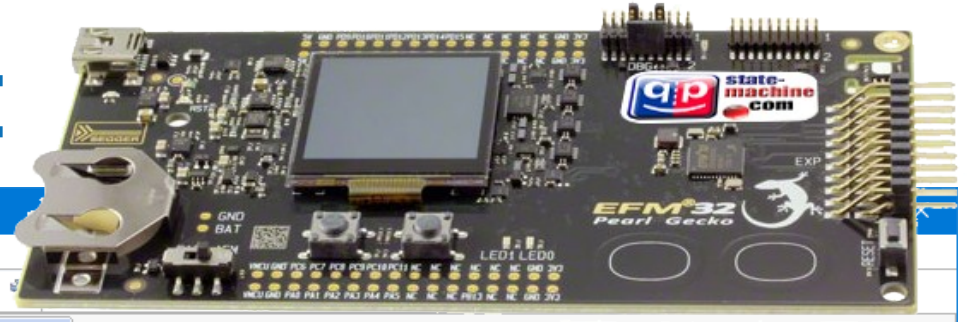
Multithreading (Multitasking):
Create an illusion that each such Thread has the whole CPU all to itself by frequently switching the CPU context from one Thread to another.



RTOS: BLINK & BUTTON EXAMPLE



RTOS: BLINK & BUTTON EXAMPLE



C:\QL-company\Screencasts-qp\EOC_Samek\code\blinky-freertos\blinky-freertos.uvprojx - µVision

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

superloop

Project

- Project: blinky-freertos
  - blinky-dbg
    - Applicatioin
      - bsp.c
      - bsp.h
      - FreeRTOSConfig.h
      - main.c
    - efm32pg1b
      - startup_efm32pg1b.s
      - system_efm32pg1b.c
      - em_cmu.c
      - em_emu.c
      - em_gpio.c
      - em_int.c
      - em_rtcc.c
      - em_system.c
    - FreeRTOS
      - tasks.c
      - list.c
      - queue.c
      - port.c

Project | Books | {} Func... | 0↓ Temp...

main.c

```
15  #include "FreeRTOS.h"
16  #include "task.h"
17  #include "semphr.h"
18  #include "bsp.h"
19
20  /* The Blinky thread ========================================================*/
21  static void blinky_thread(void *pvParameters) { /* task function */
22      (void)pvParameters; /* unused parameter(s) */
23
24      for (;;) { /* for-ever "superloop" */
25          BSP_led0_on();
26          vTaskDelay(1000 / portTICK_RATE_MS);  /* BLOCKING! */
27          BSP_led0_off();
28          vTaskDelay(1000 / portTICK_RATE_MS);  /* BLOCKING! */
29      }
30  }
31  static StaticTask_t blinky_cb; /* task control block */
32  static StackType_t blinky_stack[configMINIMAL_STACK_SIZE]; /* task stack */
33
34  /* The Button thread ========================================================*/
35  static void button_thread(void *pvParameters) { /* task function */
36      (void)pvParameters; /* unused parameter(s) */
37
38      for (;;) { /* for-ever "superloop" */
39          /* wait on the button-press semaphore (BLOCK indefinitely) */
40          xSemaphoreTake(BSP_semaPress, portMAX_DELAY); /* BLOCKING! */
41          BSP_led1_on();
42          /* wait on the button-release semaphore (BLOCK indefinitely) */
43          xSemaphoreTake(BSP_semaRelease, portMAX_DELAY); /* BLOCKING! */
44          BSP_led1_off();
45      }
46  }
47  static StaticTask_t button_cb; /* task control block */
48  static StackType_t button_stack[configMINIMAL_STACK_SIZE]; /* task stack */
49
50  static StaticSemaphore_t semaPress_cb;    /* semaphore control block */
51  static StaticSemaphore_t semaRelease_cb;  /* semaphore control block */
52  SemaphoreHandle_t BSP_semaPress;          /* global semaphore handle */
53  SemaphoreHandle_t BSP_semaRelease;        /* global semaphore handle */
54
55  /* the main function =======================================================*/
56  int main() {
```

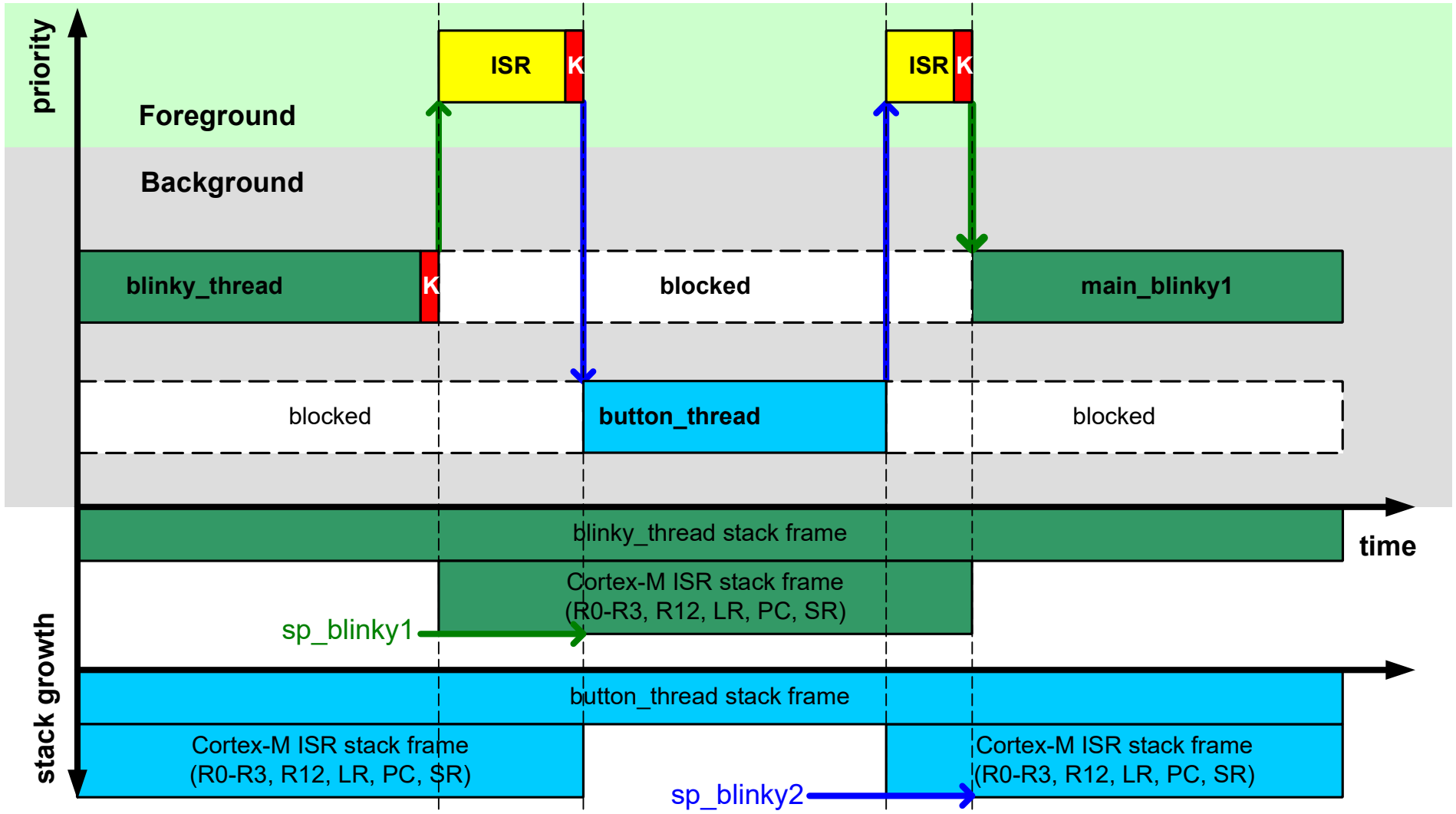
bsp.c

```
35
36  /* Hooks ====================================================================*/
37  /* Application hooks used in this project ==================================*/
38  /* NOTE: only the "FromISR" API variants are allowed in vApplicationTickHook*/
39  void vApplicationTickHook(void) {
40      BaseType_t xHigherPriorityTaskWoken = pdFALSE;
41
42      /* state of the button debouncing, see below */
43      static struct ButtonsDebouncing {
44          uint32_t depressed;
45          uint32_t previous;
46      } buttons = { 0U, 0U };
47      uint32_t current;
48      uint32_t tmp;
49
50      /* Perform the debouncing of buttons. The algorithm for debouncing
51      * adapted from the book "Embedded Systems Dictionary" by Jack Ganssle
52      * and Michael Barr, page 71.
53      */
54      current = ~GPIO->P[PB_PORT].DIN; /* read PB0 and BP1 */
55      tmp = buttons.depressed; /* save the debounced depressed buttons */
56      buttons.depressed |= (buttons.previous & current); /* set depressed */
57      buttons.depressed &= (buttons.previous | current); /* clear released */
58      buttons.previous   = current; /* update the history */
59      tmp ^= buttons.depressed;     /* changed debounced depressed */
60
61      if ((tmp & (1U << PB0_PIN)) != 0U) {  /* debounced PB0 state changed? */
62          if ((buttons.depressed & (1U << PB0_PIN)) != 0U) { /* PB0 depressed? */
63              /* signal the "button-Pressed" semaphore from ISR */
64              xSemaphoreGiveFromISR(BSP_semaPress,
65                                    &xHigherPriorityTaskWoken);
66          }
67          else { /* the button is released */
68              /* signal the "button-Released" semaphore from ISR */
69              xSemaphoreGiveFromISR(BSP_semaRelease,
70                                    &xHigherPriorityTaskWoken);
71          }
72      }
73
74      /* notify FreeRTOS to perform context switch from ISR, if needed */
75      portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
76  }
```

Build Output

```
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\tools\Keil_v5\ARM\ARMCC\Bin'
Build target 'blinky-dbg'
```

THREAD BLOCKING



RTOS BENEFITS

1. Enable composability of sequentially programmed components

Sidesteps the unresponsiveness of the simple *wait-respond* paradigm by allowing multiple "superloops" in a single program

2. More efficient CPU use

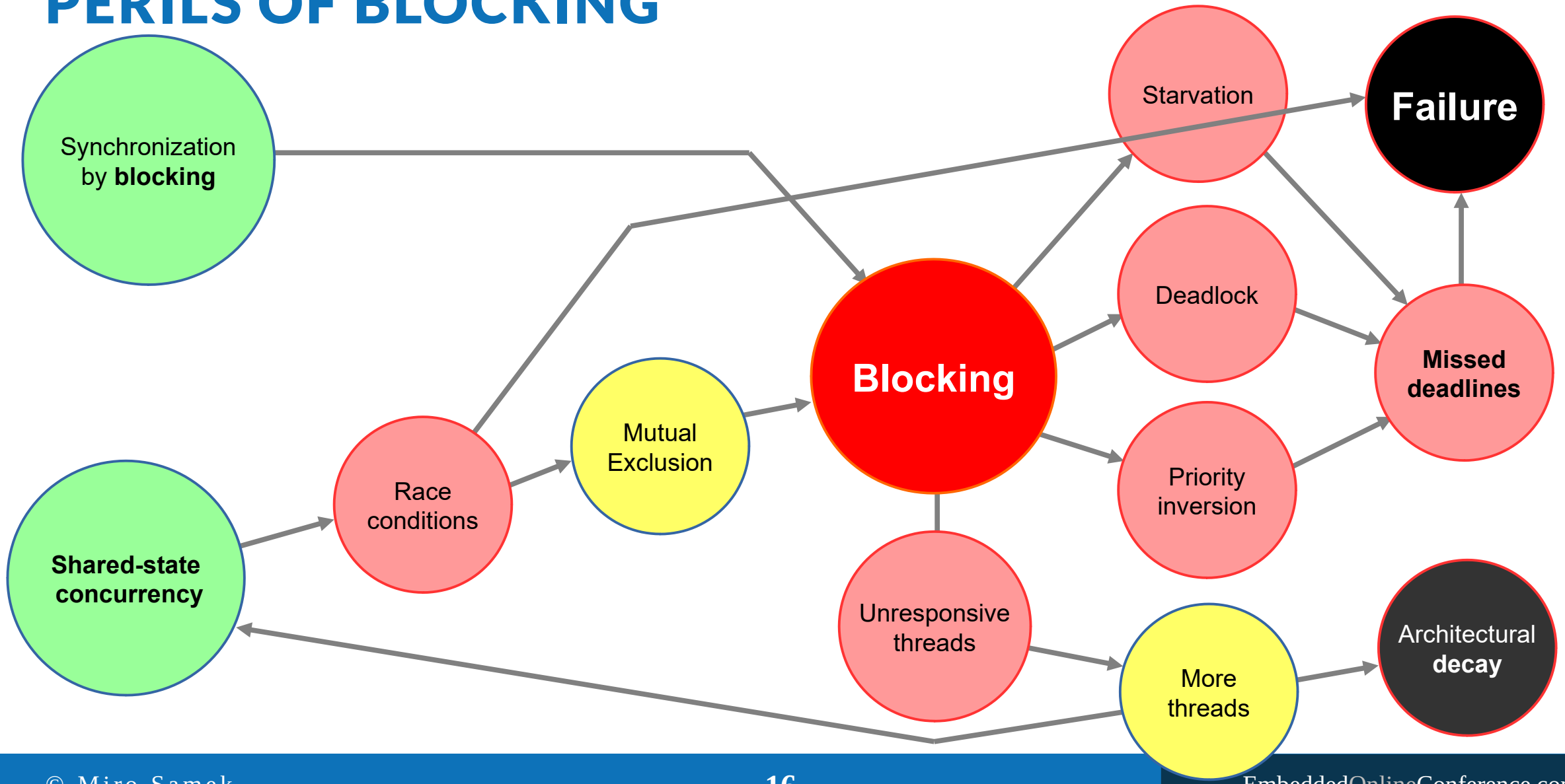
Threads that are efficiently **blocked** don't consume CPU cycles.

When all threads are blocked, idle thread can put the system into low-power mode

3. Threads can be decoupled in the time domain

Under a preemptive, priority-based scheduler, changes in low-priority threads have no impact on the timing of high-priority threads

PERILS OF BLOCKING



AGENDA

0 Challenges of Embedded Software Development

1 Traditional Sequential Programming & RTOS

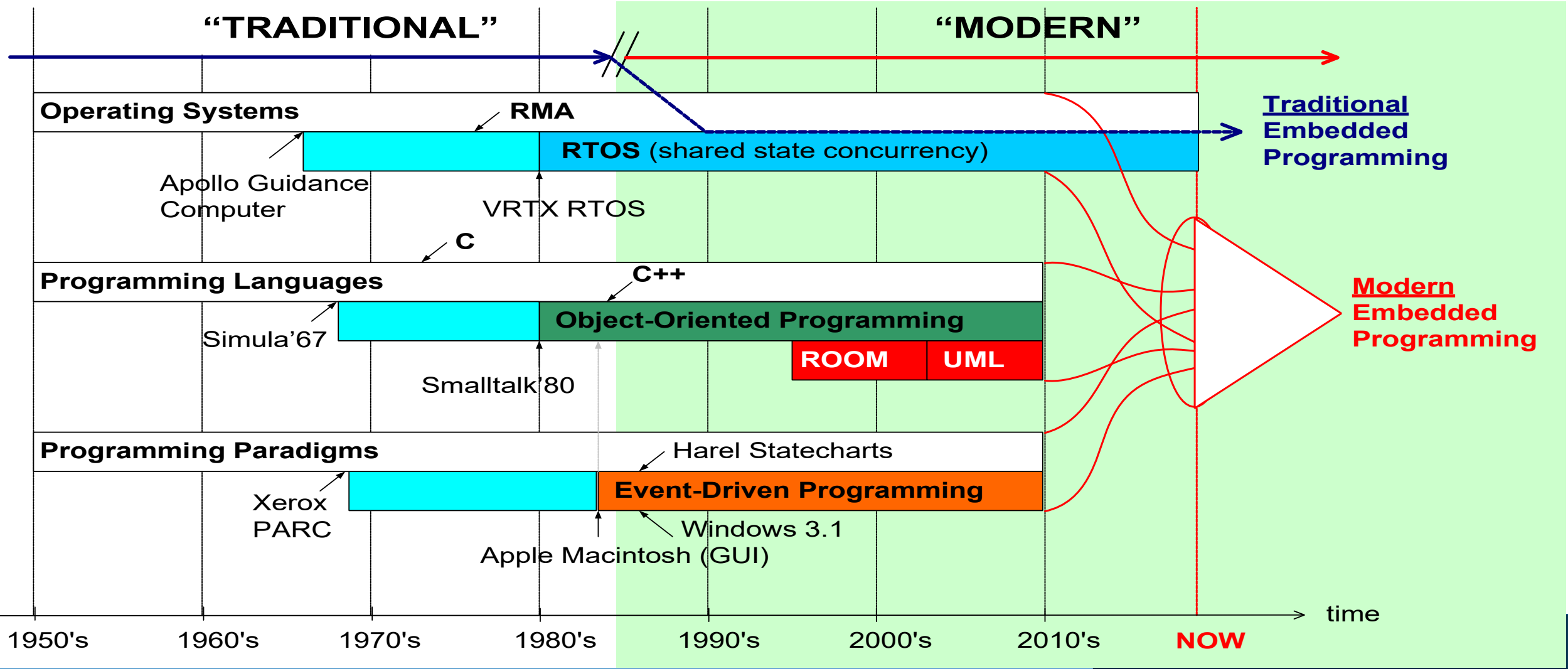
2 Best Practices for Concurrency & Active Object Pattern

3 Event-Driven Paradigm Shift & RTOS vs. Framework

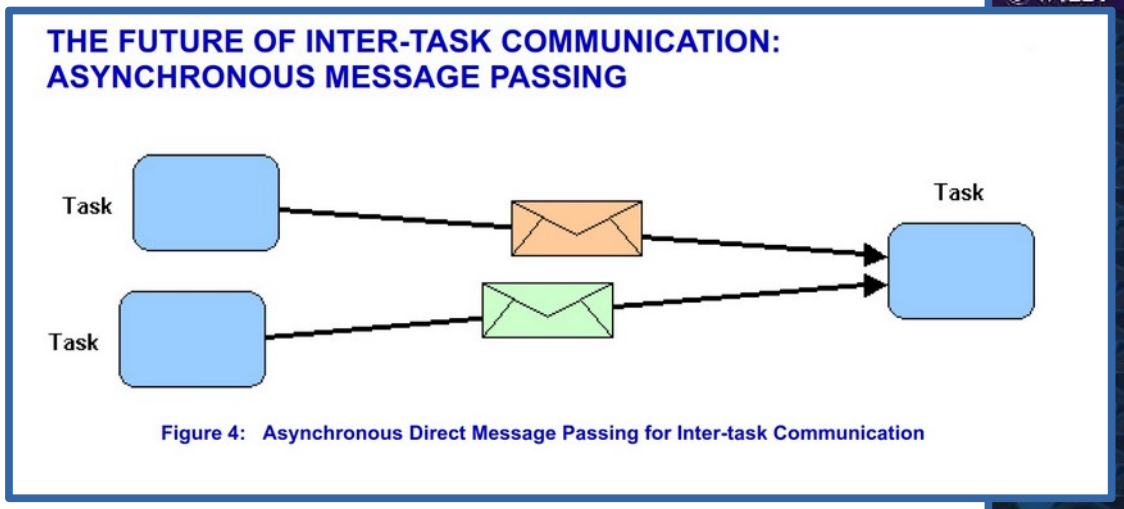
4 Hierarchical State Machines

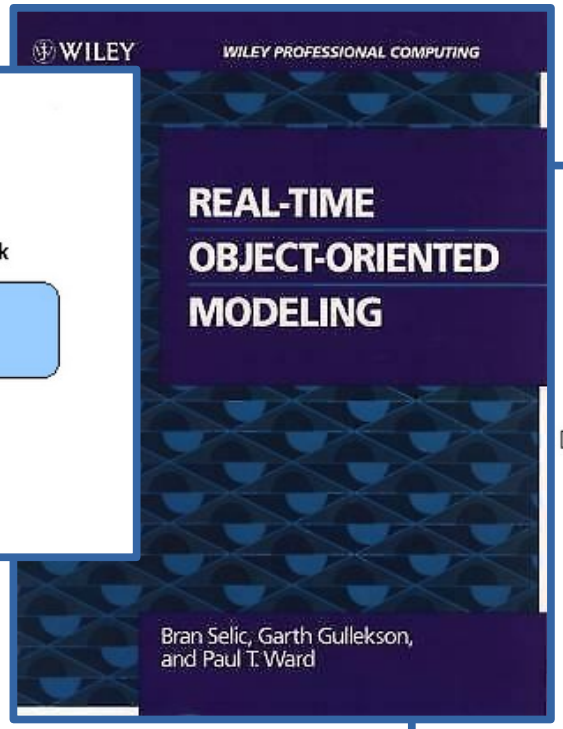
5 Software Modeling & Code Generation

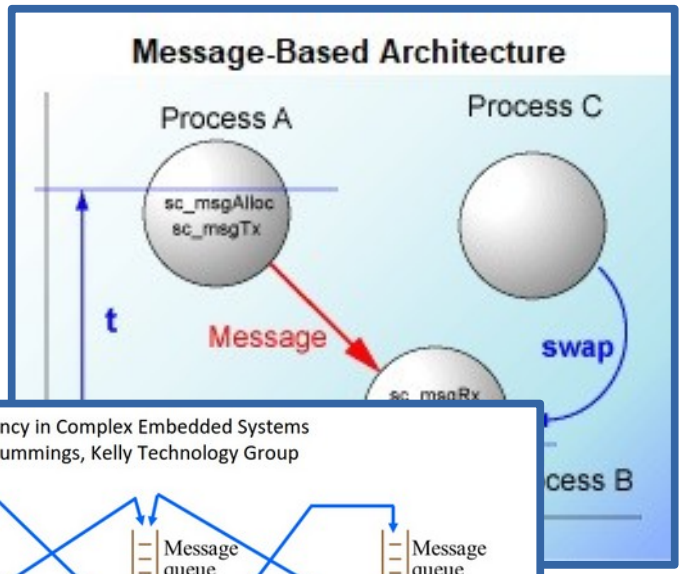
HISTORICAL PERSPECTIVE – "MODERN" vs "TRADITIONAL"

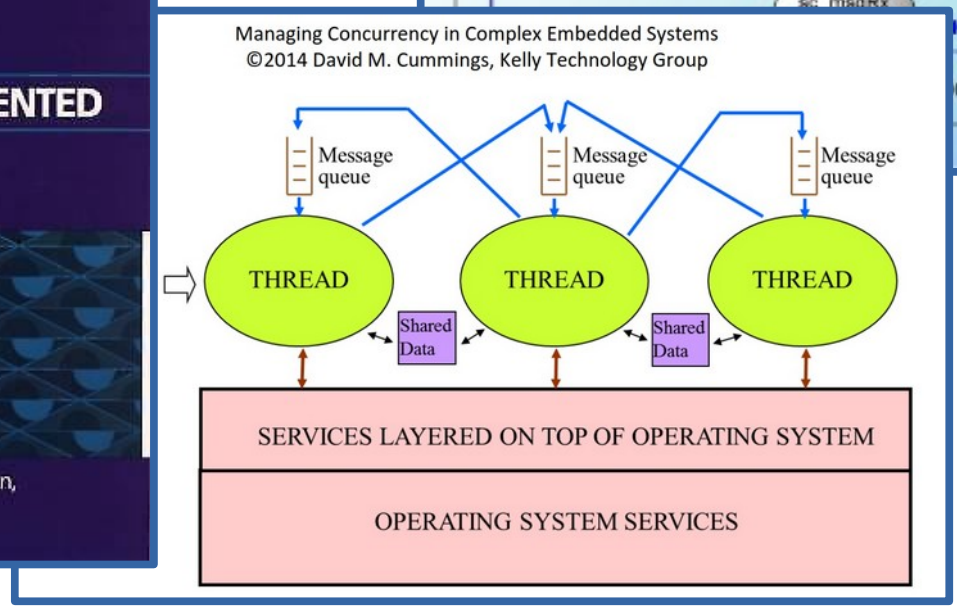


EXPERTS ON CONCURRENCY – "MODERN" APPROACHES









- Selic, Gullekson, Ward *"Real-Time Object-Oriented Modeling""*
- Kalinsky *"Messages for Intertask Communication"*
- Cummings *"Managing Concurrency in Complex Embedded Systems"*
- SCIOPTA *"Message-Based RTOS"*

BEST PRACTICES OF CONCURRENCY (*)

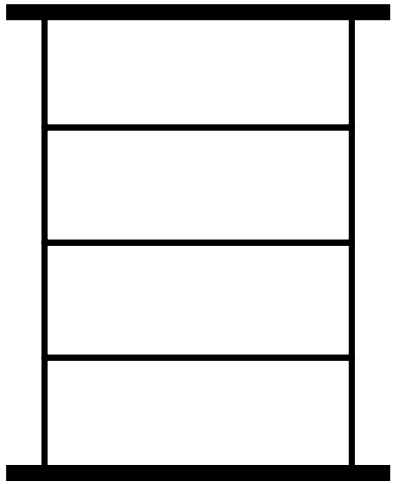
1. Keep data isolated, private and bound to threads:
   - **Don't share** data or resources among threads

2. Communicate among threads via asynchronous events:
   - Let the threads run independently **without blocking** on each other

3. Organize your threads around "message pumps"
   - Restrict **blocking** to a single place in the thread "superloop"

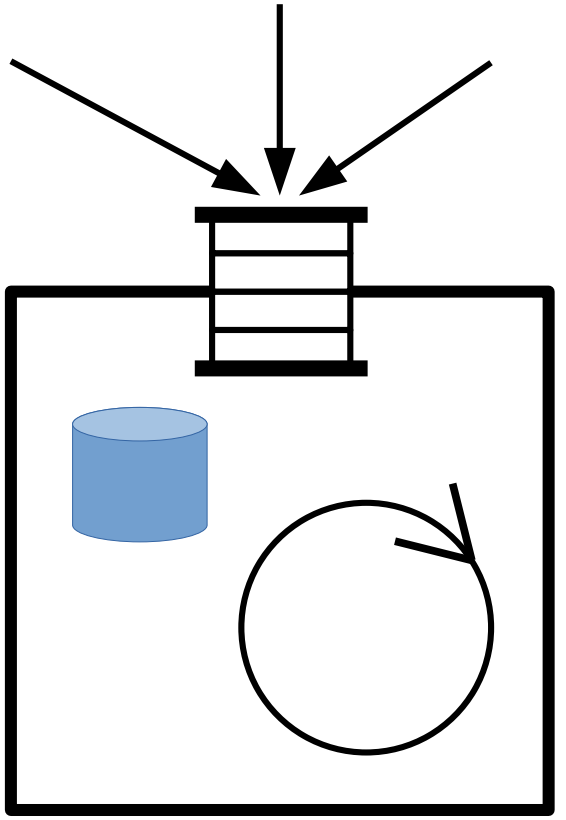
(*) **Herb Sutter** *"Prefer Using Active Objects Instead of Naked Threads"*

EVENT LOOP ("MESSAGE PUMP")



```
void thread(void *pvParameters) {  // FreeRTOS thread
    ActiveObject *me = (ActiveObject *)pvParameters;

    for (;;) {    // for-ever "superloop"
        Event e; // event object ("message")

        // wait for any event and receive it into object 'e'
        xQueueReceive(me->queue, &e, portMAX_DELAY); // BLOCKING!

        // dispatch object 'e' to the active object 'me'
        ActiveObject_dispatch(me, &e); // NO BLOCKING!
    }
}
```

"ACTIVE OBJECT" DESIGN PATTERN



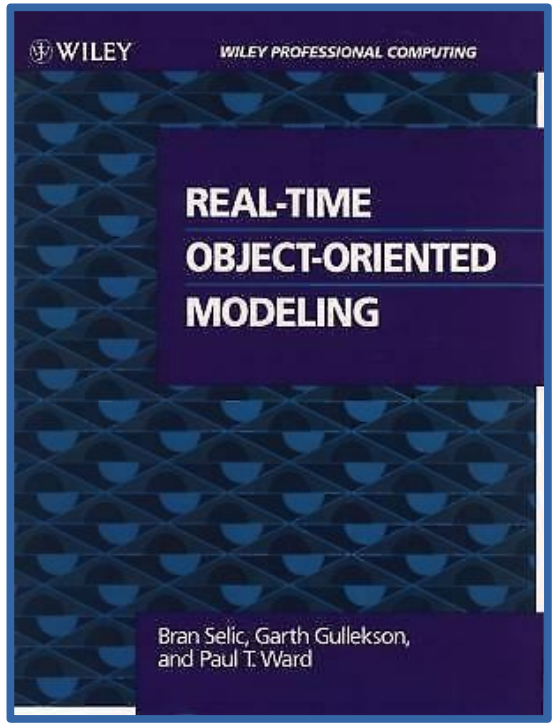
- Active Object =

```
private-data + private-thread + private-queue
```

- The only way to communicate with an AO: asynchronous event posting to its queue
- Events are processed to completion on the private thread
  - No need for mutual exclusion
- The strictest form of OOP
- Higher-level of abstraction above the "naked" RTOS

FROM "ACTORS" TO "ACTIVE OBJECTS"

- Not a novelty: can be traced back to Carl Hewitt's "Actors" (1970's)
- Real-Time Object Oriented Modeling (ROOM, 1994)
  - "ROOM Actors" for real-time embedded systems
  - Hierarchical state machines for internal behavior
- UML Active Objects / Active Classes (UML-RT "Capsules")



AGENDA

0 Challenges of Embedded Software Development

1 Traditional Sequential Programming & RTOS

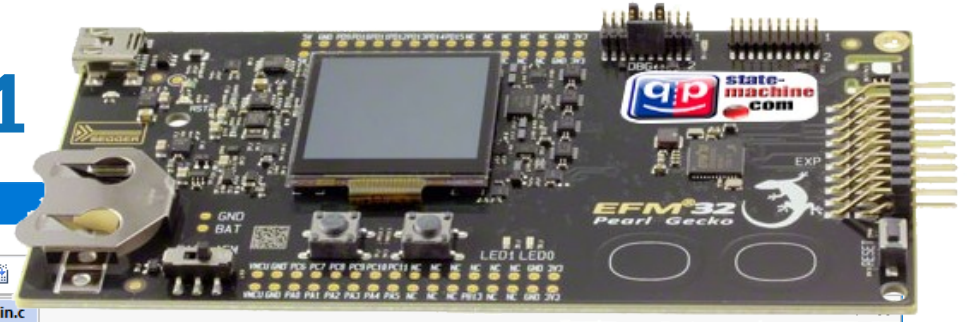
2 Best Practices for Concurrency & Active Object Pattern

3 Event-Driven Paradigm Shift & RTOS vs. Framework

4 Hierarchical State Machines

5 Software Modeling & Code Generation

ACTIVE OBJECTS WITH RTOS, STEP 1



C:\QL-company\Screencasts-qp\EOC_Samek\code\blinky1-FreeAct\blinky-freeact.uvprojx - µVision

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Project

- Project: blinky-freeact
  - blinky-dbg
    - Applicatioin
      - bsp.c
      - bsp.h
      - FreeRTOSConfig.h
      - main.c
      - FreeAct.c
      - FreeAct.h
    - FreeRTOS
    - efm32pg1b

Project Books {} Func... {} Tem...

FreeAct.c | FreeAct.h

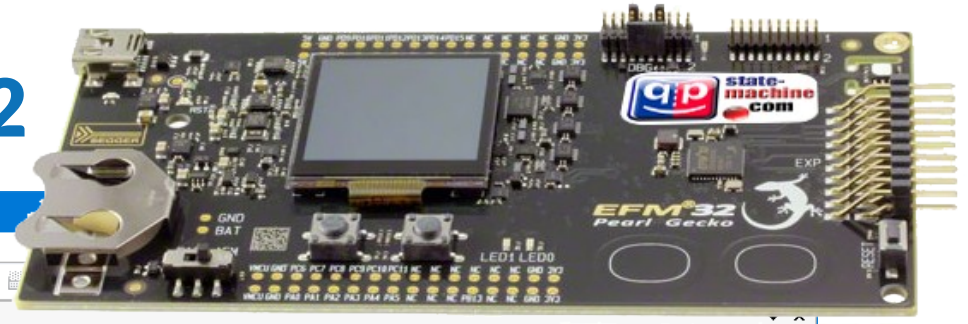
```
56
57  /*--------------------------------------------------------------------------*/
58  /* Actvie Object facilities... */
59
60  typedef struct Active Active; /* forward declaration */
61
62  typedef void (*DispatchHandler)(Active * const me, Event const * const e);
63
64  /* Active Object base class */
65  struct Active {
66      TaskHandle_t thread;     /* private thread */
67      StaticTask_t thread_cb;  /* thread control-block (FreeRTOS static alloc) */
68
69      QueueHandle_t queue;     /* private message queue */
70      StaticQueue_t queue_cb;  /* queue control-block (FreeRTOS static alloc) */
71
72      DispatchHandler dispatch; /* pointer to the dispatch() function */
73
74      /* active object data added in subclasses of Active */
75  };
76
77  void Active_ctor(Active * const me, DispatchHandler dispatch);
78  void Active_start(Active * const me,
79                    uint8_t prio,
80                    Event **queueSto,
81                    uint32_t queueLen,
82                    StackType_t *stackSto,
83                    uint32_t stackDepth);
84  void Active_post(Active * const me, Event const * const e);
85  void Active_postFromISR(Active * const me, Event const * const e,
86                          BaseType_t *pxHigherPriorityTaskWoken);
87
88  /*--------------------------------------------------------------------------*/
89  /* Time Event facilities... */
90
91  /* Time Event class */
92  typedef struct {
93      Event super;       /* inherit Event */
94      Active *act;       /* the AO that requested this TimeEvent */
95      uint32_t timeout;  /* timeout counter; 0 means not armed */
```

main.c

```
59
60  /* The Button AO ============================================================*/
61  typedef struct {
62      Active super; /* inherit Active base class */
63      /* add private data for the AO... */
64  } Button;
65
66  static void Button_dispatch(Button * const me, Event const * const e) {
67      switch (e->sig) {
68          case BUTTON_PRESSED_SIG: {
69              BSP_led1_on();
70              break;
71          }
72          case BUTTON_RELEASED_SIG: {
73              BSP_led1_off();
74              break;
75          }
76          default: {
77              break;
78          }
79      }
80  }
81  void Button_ctor(Button * const me) {
82      Active_ctor(&me->super, (DispatchHandler)&Button_dispatch);
83  }
84  static StackType_t button_stack[configMINIMAL_STACK_SIZE]; /* task stack */
85  static Event *button_queue[10];
86  static Button button;
87  Active *AO_button = &button.super;
88
89  /* the main function =======================================================*/
90  int main() {
91
92      BSP_init(); /* initialize the BSP */
93
94      /* create and start the Blinky AO */
95      Blinky_ctor(&l_blinky);
96      Active_start(AO_blinky,
97                   1U,
98                   blinky_queue,
```

Build Output

ACTIVE OBJECTS WITH RTOS, STEP 2



C:\QL-company\Screencasts-qp\EOC_Samek\code\blinky1-FreeAct\blinky-freeact.uvprojx - µVision

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Project

- Project: blinky-freeact
  - blinky-dbg
    - Applicatioin
      - bsp.c
      - bsp.h
      - FreeRTOSConfig.h
      - main.c
      - FreeAct.c
      - FreeAct.h
    - FreeRTOS
    - efm32pg1b

Project Books {} Func... {} Tem...

FreeAct.c | FreeAct.h

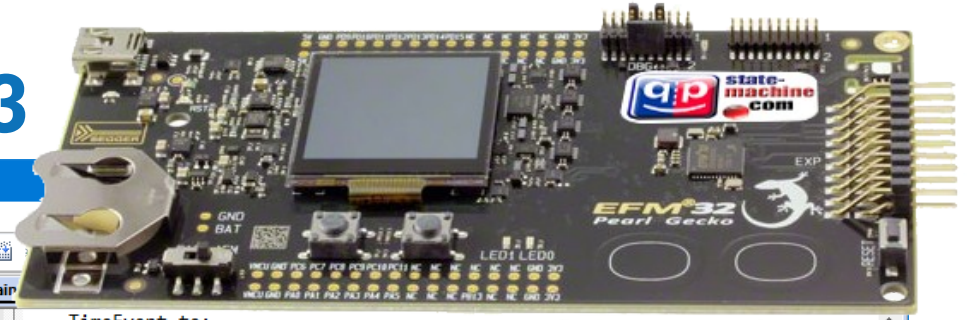
```
69      QueueHandle_t queue;     /* private message queue */
70      StaticQueue_t queue_cb;  /* queue control-block (FreeRTOS static alloc) */
71
72      DispatchHandler dispatch; /* pointer to the dispatch() function */
73
74      /* active object data added in subclasses of Active */
75  };
76
77  void Active_ctor(Active * const me, DispatchHandler dispatch);
78  void Active_start(Active * const me,
79                    uint8_t prio,
80                    Event **queueSto,
81                    uint32_t queueLen,
82                    StackType_t *stackSto,
83                    uint32_t stackDepth);
84  void Active_post(Active * const me, Event const * const e);
85  void Active_postFromISR(Active * const me, Event const * const e,
86                          BaseType_t *pxHigherPriorityTaskWoken);
87
88  /*--------------------------------------------------------------------------*/
89  /* Time Event facilities... */
90
91  /* Time Event class */
92  typedef struct {
93      Event super;        /* inherit Event */
94      Active *act;        /* the AO that requested this TimeEvent */
95      uint32_t timeout;   /* timeout counter; 0 means not armed */
96      uint32_t interval; /* interval for periodic TimeEvent, 0 means one-shot */
97  } TimeEvent;
98
99  void TimeEvent_ctor(TimeEvent * const me, Signal sig, Active *act);
100 void TimeEvent_arm(TimeEvent * const me, uint32_t timeout, uint32_t interval);
101 void TimeEvent_disarm(TimeEvent * const me);
102
103 /* static (i.e., class-wide) operation */
104 void TimeEvent_tickFromISR(BaseType_t *pxHigherPriorityTaskWoken);
105
106 #endif /* FREE_ACT_H */
107
```

main.c

```
23      TimeEvent te;
24      bool isLedOn;
25  } Blinky;
26
27  static void Blinky_dispatch(Blinky * const me,
28                              Event const * const e)
29  {
30      switch (e->sig) {
31          case INIT_SIG: /* intentionally fall through... */
32          case TIMEOUT_SIG: {
33              if (!me->isLedOn) { /* LED not on */
34                  BSP_led0_on();
35                  me->isLedOn = true;
36                  TimeEvent_arm(&me->te, (200 / portTICK_RATE_MS), 0U);
37              }
38              else {  /* LED is on */
39                  BSP_led0_off();
40                  me->isLedOn = false;
41                  TimeEvent_arm(&me->te, (800 / portTICK_RATE_MS), 0U);
42              }
43              break;
44          }
45          default: {
46              break;
47          }
48      }
49  }
50  void Blinky_ctor(Blinky * const me) {
51      Active_ctor(&me->super, (DispatchHandler)&Blinky_dispatch);
52      TimeEvent_ctor(&me->te, TIMEOUT_SIG, &me->super);
53      me->isLedOn = false;
54  }
55  static StackType_t blinky_stack[configMINIMAL_STACK_SIZE]; /* task stack */
56  static Event *blinky_queue[10];
57  static Blinky l_blinky;
58  Active *AO_blinky = &l_blinky.super;
59
60  /* The Button AO ============================================================*/
61  typedef struct {
62      Active super; /* inherit Active base class */
```

Build Output

ACTIVE OBJECTS WITH RTOS, STEP 3



C:\QL-company\Screencasts-qp\EOC_Samek\code\blinky1-FreeAct\blinky-freeact.uvprojx - µVision

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Blinky

Project

- Project: blinky-freeact
  - blinky-dbg
    - Applicatioin
      - bsp.c
      - bsp.h
      - FreeRTOSConfig.h
      - main.c
      - FreeAct.c
      - FreeAct.h
    - FreeRTOS
    - efm32pg1b

Project | Books | {} Func... | {} Tem...

FreeAct.c | FreeAct.h

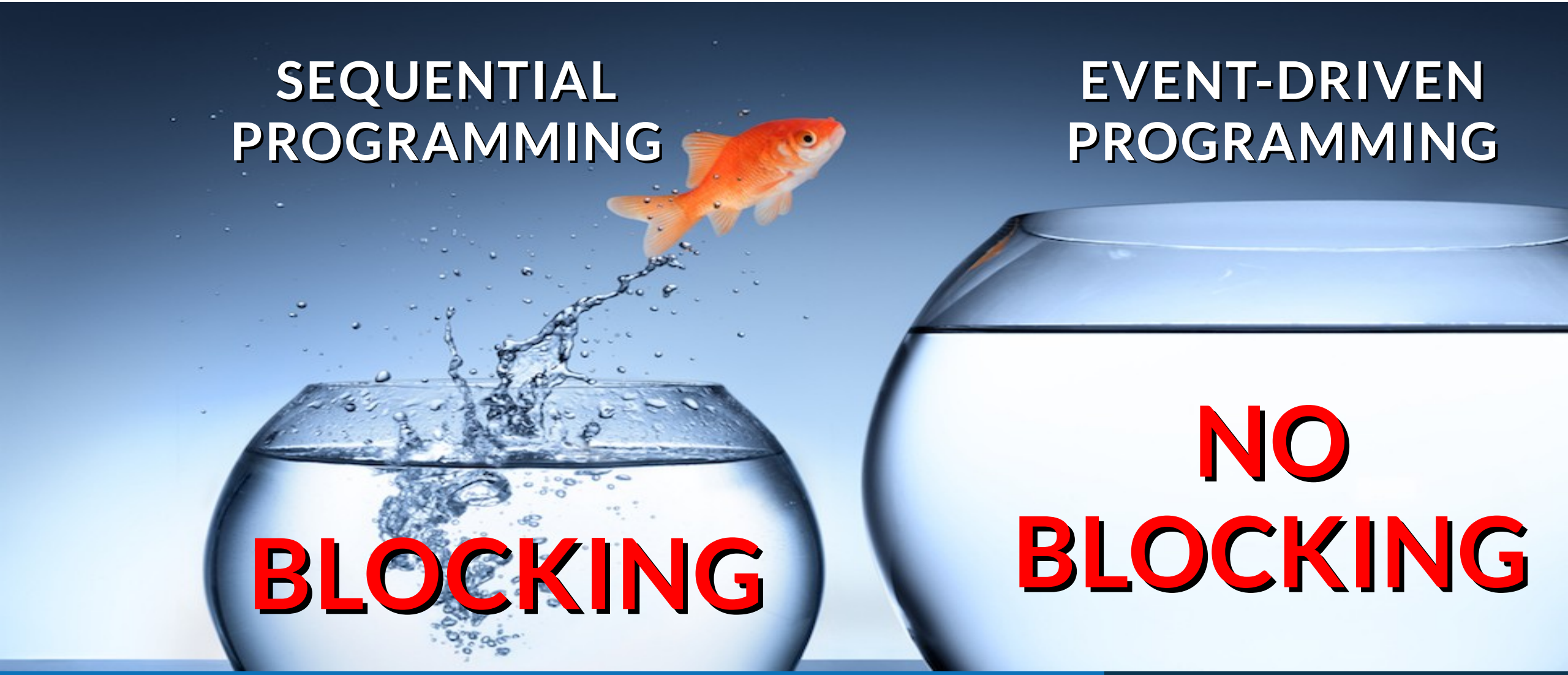
```
69      QueueHandle_t queue;     /* private message queue */
70      StaticQueue_t queue_cb;  /* queue control-block (FreeRTOS static alloc) */
71
72      DispatchHandler dispatch; /* pointer to the dispatch() function */
73
74      /* active object data added in subclasses of Active */
75  };
76
77  void Active_ctor(Active * const me, DispatchHandler dispatch);
78  void Active_start(Active * const me,
79                    uint8_t prio,
80                    Event **queueSto,
81                    uint32_t queueLen,
82                    StackType_t *stackSto,
83                    uint32_t stackDepth);
84  void Active_post(Active * const me, Event const * const e);
85  void Active_postFromISR(Active * const me, Event const * const e,
86                          BaseType_t *pxHigherPriorityTaskWoken);
87
88  /*--------------------------------------------------------------------------*/
89  /* Time Event facilities... */
90
91  /* Time Event class */
92  typedef struct {
93      Event super;       /* inherit Event */
94      Active *act;       /* the AO that requested this TimeEvent */
95      uint32_t timeout;  /* timeout counter; 0 means not armed */
96      uint32_t interval; /* interval for periodic TimeEvent, 0 means one-shot */
97  } TimeEvent;
98
99  void TimeEvent_ctor(TimeEvent * const me, Signal sig, Active *act);
100 void TimeEvent_arm(TimeEvent * const me, uint32_t timeout, uint32_t interval);
101 void TimeEvent_disarm(TimeEvent * const me);
102
103 /* static (i.e., class-wide) operation */
104 void TimeEvent_tickFromISR(BaseType_t *pxHigherPriorityTaskWoken);
105
106 #endif /* FREE_ACT_H */
107
```

main

```
23      TimeEvent te;
24      bool isLedOn;
25  } Blinky;
26
27  static void Blinky_dispatch(Blinky * const me,
28                              Event const * const e)
29  {
30      switch (e->sig) {
31          case INIT_SIG: /* intentionally fall through... */
32          case TIMEOUT_SIG: {
33              if (!me->isLedOn) { /* LED not on */
34                  BSP_led0_on();
35                  me->isLedOn = true;
36                  TimeEvent_arm(&me->te, (200 / portTICK_RATE_MS), 0U);
37              }
38              else {  /* LED is on */
39                  BSP_led0_off();
40                  me->isLedOn = false;
41                  TimeEvent_arm(&me->te, (800 / portTICK_RATE_MS), 0U);
42              }
43              break;
44          }
45          default: {
46              break;
47          }
48      }
49  }
50  void Blinky_ctor(Blinky * const me) {
51      Active_ctor(&me->super, (DispatchHandler)&Blinky_dispatch);
52      TimeEvent_ctor(&me->te, TIMEOUT_SIG, &me->super);
53      me->isLedOn = false;
54  }
55  static StackType_t blinky_stack[configMINIMAL_STACK_SIZE]; /* task stack */
56  static Event *blinky_queue[10];
57  static Blinky l_blinky;
58  Active *AO_blinky = &l_blinky.super;
59
60  /* The Button AO ============================================================*/
61  typedef struct {
62      Active super; /* inherit Active base class */
```

Build Output

SEQUENTIAL → EVENT-DRIVEN PARADIGM SHIFT

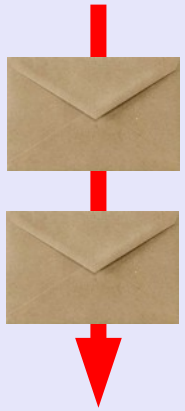


SEQUENTIAL → EVENT-DRIVEN PARADIGM SHIFT

"Shifting the paradigm didn't work. Time for Plan B...shifting the blame."

PROS AND CONS OF SEQUENTIAL PROGRAMMING

```
BSP_led0_on();
vTaskDelay(...);
BSP_led0_off();
vTaskDelay(...);
```



Hard-coded sequence of events

- 👍 Easy to handle the **context** between events
- 👎 Unresponsive to new events
- 👎 Hard to handle multiple event sequences
- 👎 Requires RTOS for composability

Problem:

Most real-life systems must handle MULTIPLE event sequences

PROS AND CONS OF EVENT-DRIVEN PROGRAMMING

```
switch (e->sig) {
    case INIT_SIG:
        . . .
        break;
    case TIMEOUT_SIG:
        . . .
        break;
    case BUTTON_PRESSED_SIG:
        . . .
        break;
    case BUTTON_RELEASED_SIG:
        . . .
        break;
}
```

Any sequence of events

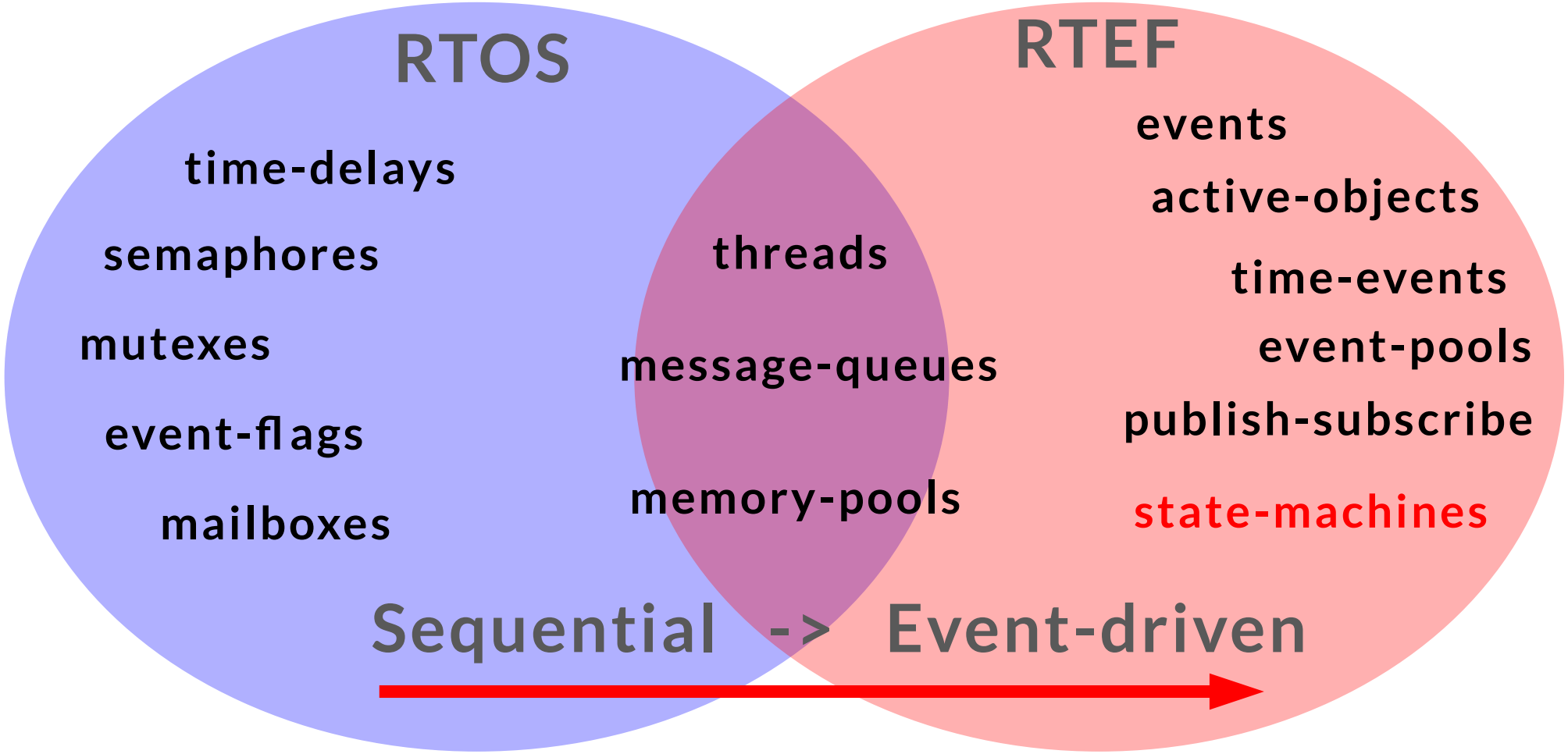
Easy to handle multiple sequences of events

Responsive to all events

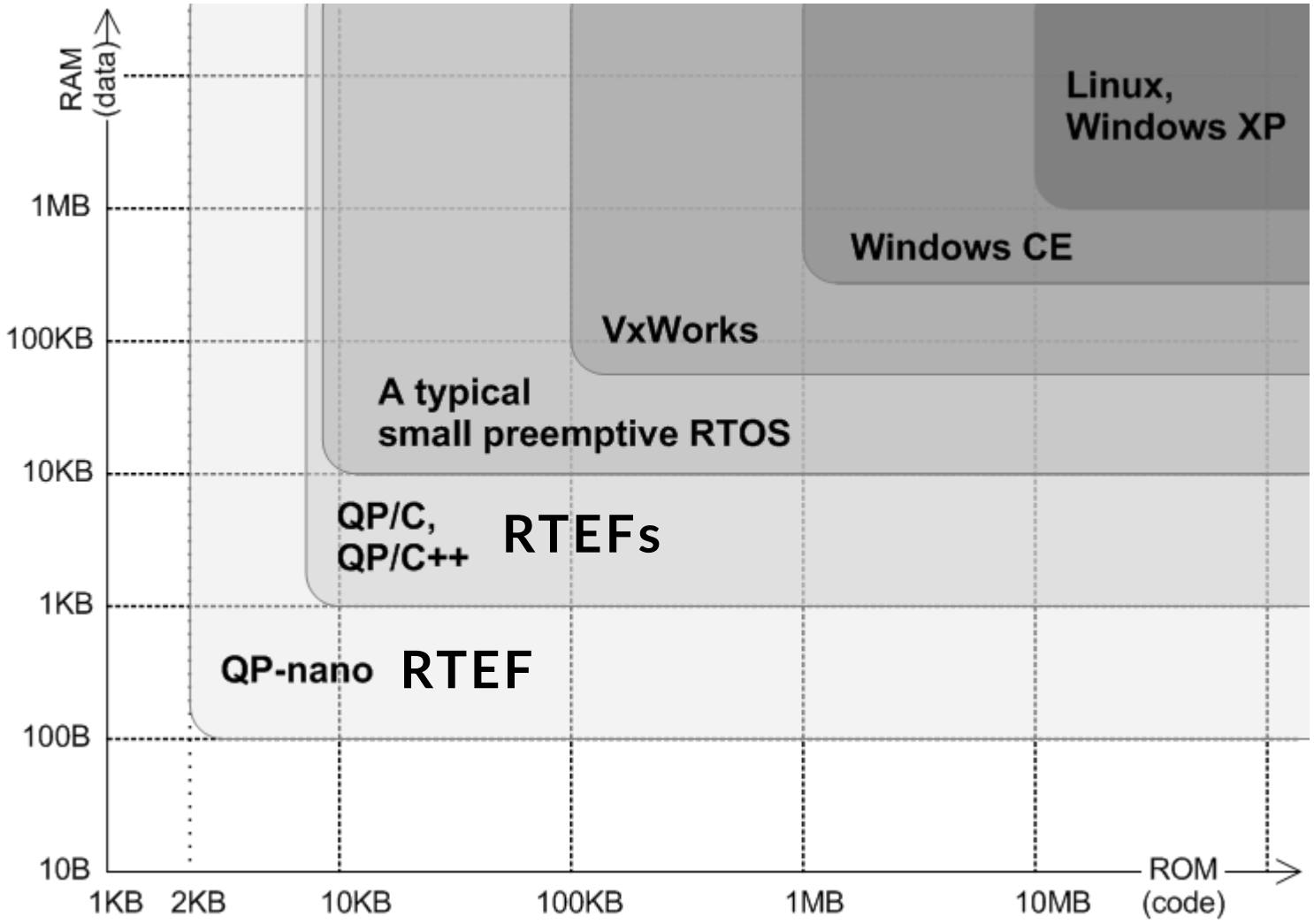
Hard to handle the **context** between events

Requires event-driven infrastructure (framework)

REAL-TIME OPERATING SYSTEM (RTOS) vs. REAL-TIME EMBEDDED FRAMEWORK (RTEF)



MEMORY FOOTPRINTS of RTOS vs. RTEF



AGENDA

0 Challenges of Embedded Software Development

1 Traditional Sequential Programming & RTOS

2 Best Practices for Concurrency & Active Object Pattern

3 Event-Driven Paradigm Shift & RTOS vs. Framework

4 Hierarchical State Machines

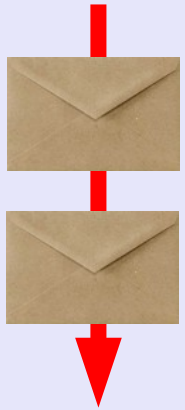
5 Software Modeling & Code Generation

THE CHALLENGE OF EVENT-DRIVEN PROGRAMMING

Proper reaction to an event depends not just on the **event signal** , but also on the **history of past events**.

Sequential

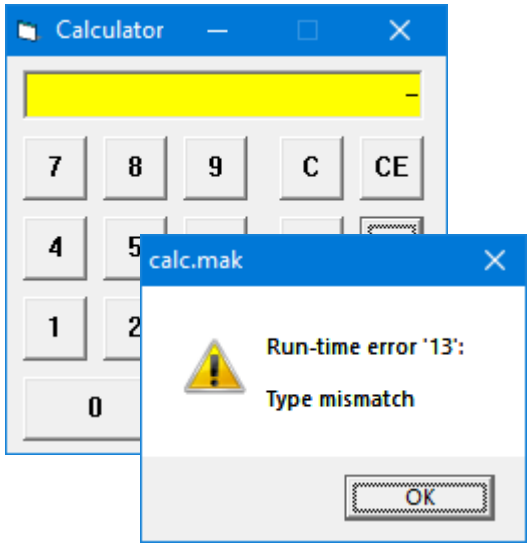
```
for (;;) {
    BSP_led0_on();
    vTaskDelay(...);
    BSP_led0_off();
    vTaskDelay(...);

}
```



Event-driven

```
switch (e->sig) {
    case TIMEOUT_SIG: {
        if (!me->isLedOn) {
            BSP_led0_on();
            me->isLedOn = true;
            TimeEvent_arm(...);
        }
        else {  /* LED is on */
            BSP_led0_off();
            me->isLedOn = false;
            TimeEvent_arm(...);
        }
        break;
    }
```

VISUAL BASIC CALCULATOR "SPAGHETTI CODE" EXAMPLE




```
Dim Op1, Op2                    ' Previously input operand.
Dim DecimalFlag As Integer      ' Decimal point present yet?
Dim NumOps As Integer           ' Number of operands.
Dim LastInput                   ' Indicate type of last keypress event.
Dim OpFlag                      ' Indicate pending operation.
Dim TempReadout
...
Private Sub Operator_Click(Index As Integer)
    TempReadout = Readout
    If LastInput = "NUMS" Then
        NumOps = NumOps + 1
    End If
    Select Case NumOps
        Case 0
        If Operator(Index).Caption = "-" And LastInput <> "NEG" Then
            Readout = "-" & Readout
            LastInput = "NEG"
        End If ...
```

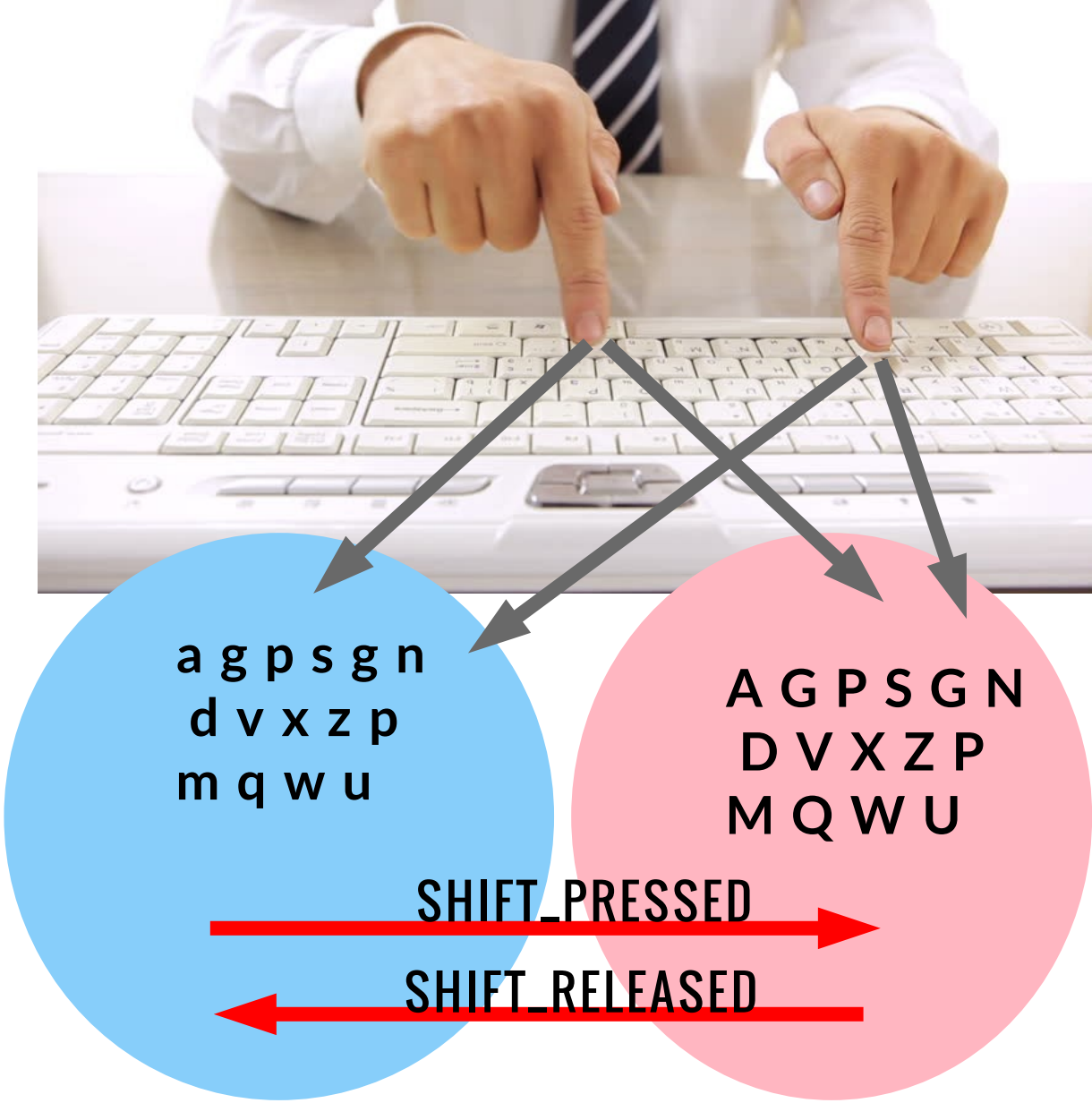
Problem :

Improvised way of recording event history with various flags and variables… Too much of disorganized & redundant information!

THE RELEVANT HISTORY AND SYSTEM "STATE"

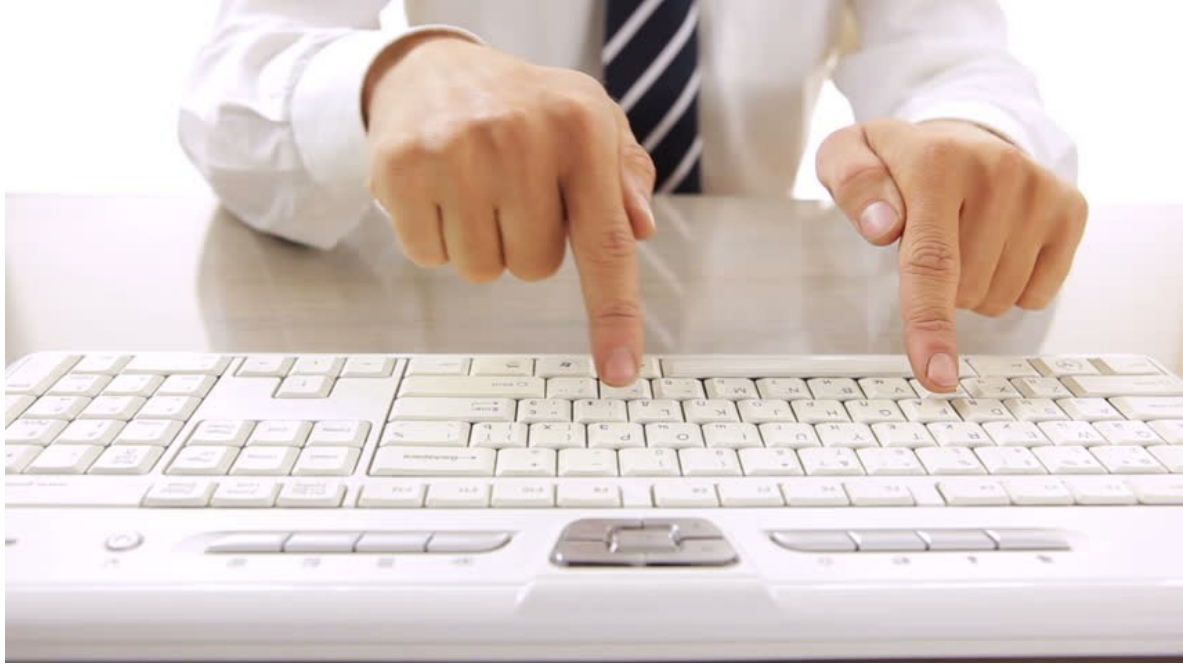
- "Relevant history" is only what influences the future behavior
- Events don't contribute equally to the relevant history

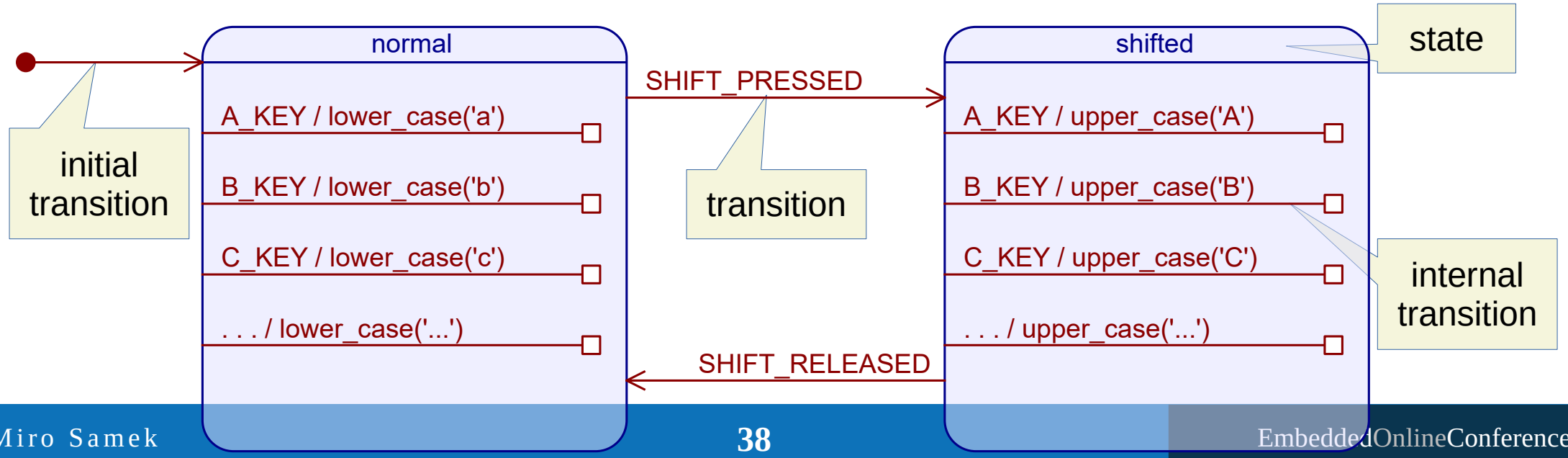
"State" of a system is an equivalence class of past histories of a system, all of which are equivalent in the sense that the future behavior of the system given any of these past histories will be identical.



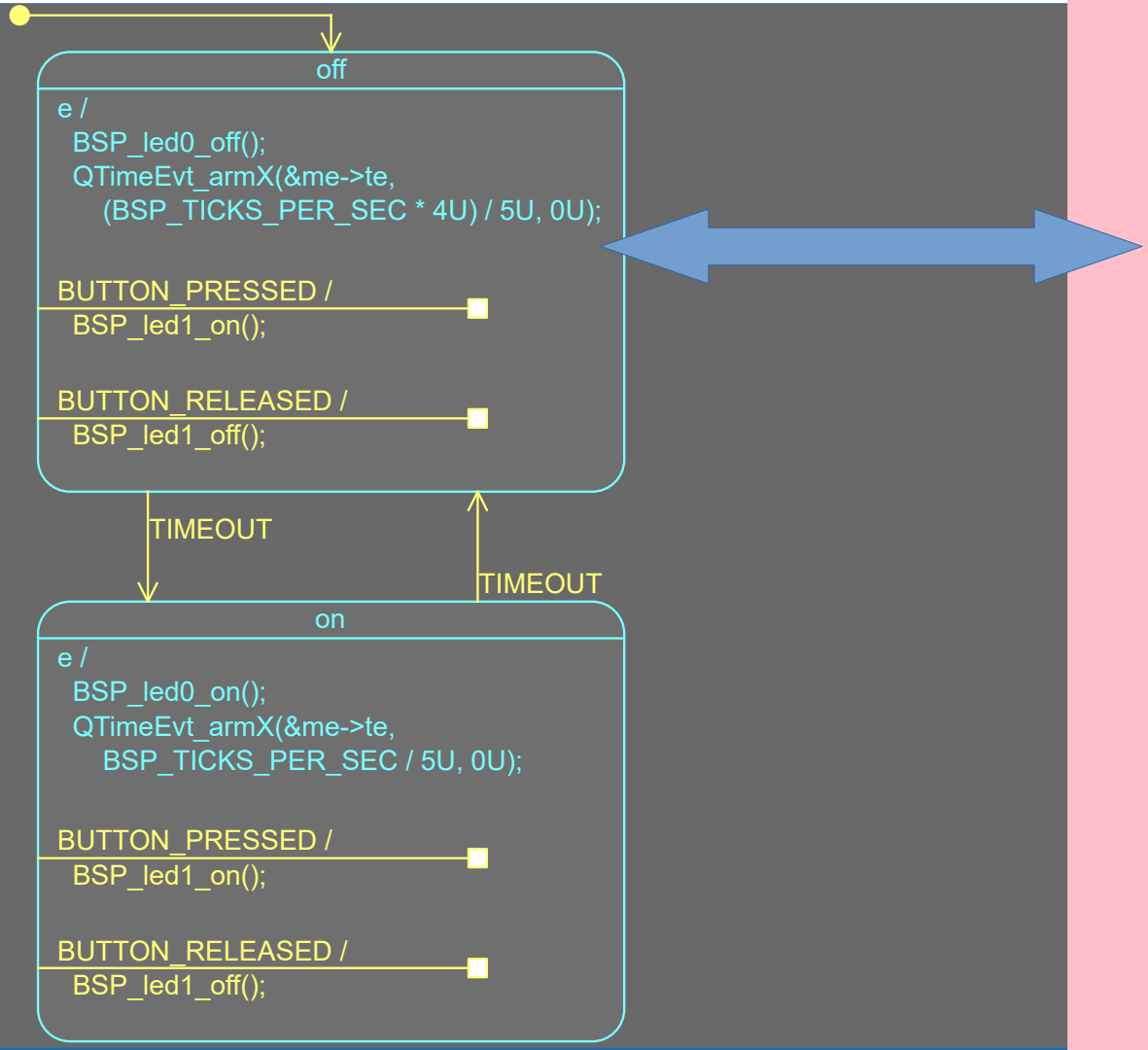
STATE MACHINE

"State Machine" is a set of all states (equivalence class of past histories of a system) plus the rules for changing from one state to another (state transitions)



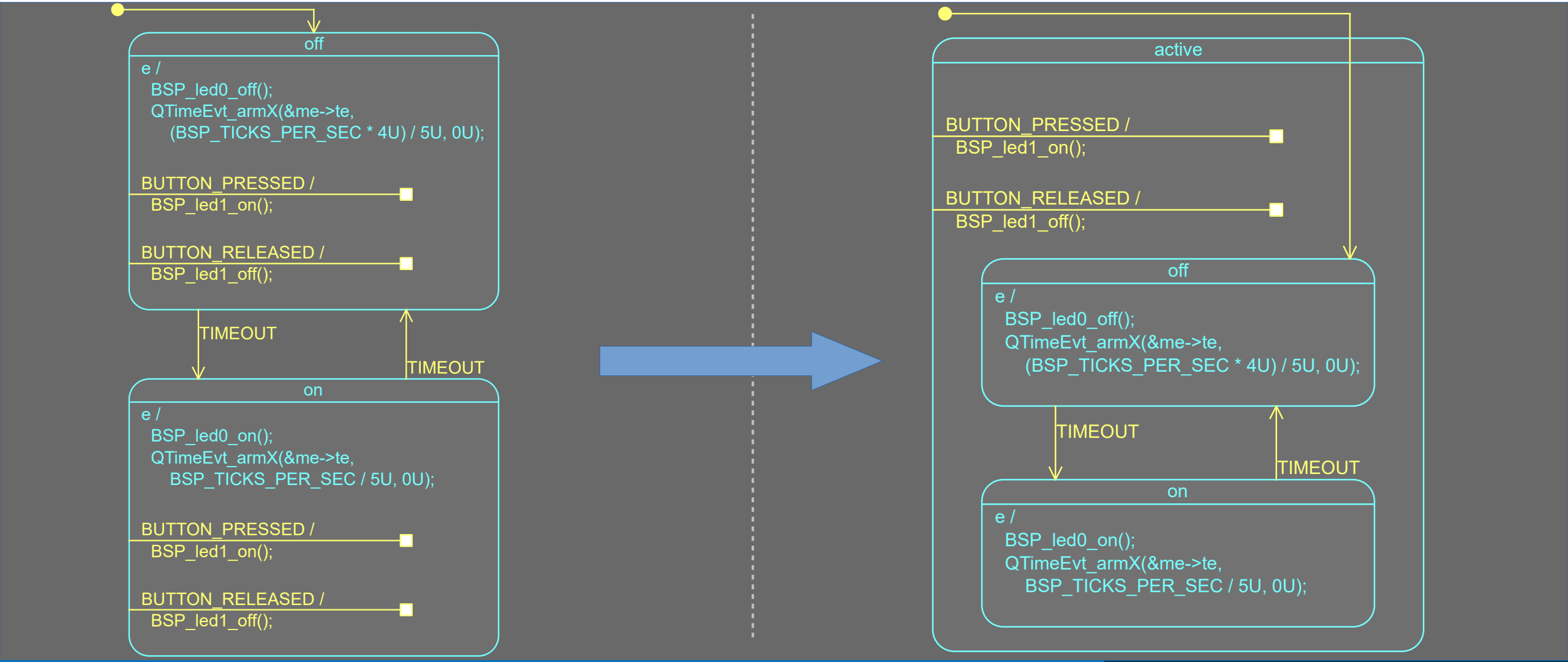


STATE MACHINE EXAMPLE




```
static QState BlinkyButton_off(BlinkyButton * const me,
                               QEvt const * const e)
{
    QState status;
    switch (e->sig) {
        case Q_ENTRY_SIG: {
            BSP_led0_off();
            QTimeEvt_armX(&me→te,
                (BSP_TICKS_PER_SEC * 4U) / 5U, 0U);
            status = Q_HANDLED();
            break;
        }
        case TIMEOUT_SIG: {
            status = Q_TRAN(&BlinkyButton_on);
            break;
        }
        case BUTTON_PRESSED_SIG: {
            BSP_led1_on();
            status = Q_HANDLED();
            break;
        }
        case BUTTON_RELEASED_SIG: {
            BSP_led1_off();
            status = Q_HANDLED();
            break;
        }
        default: {
            status = Q_SUPER(&QHsm_top);
            break;
        }
    }
    return status;
}
```

HIERARCHICAL STATE MACHINE EXAMPLE



SUMMARY

- Superloop" and RTOS are not the only games in town
- Experts on concurrency are weary of blocking and instead of a "naked" RTOS use the event-driven Active Object design pattern
- Modern hierarchical state machine are ideal for specifying the behavior of active objects, without creating "spaghetti code"
- Active objects and state machines raise the level of abstraction and provide the right abstractions for applying graphical modeling and automatic code generation

Questions, comments?  miro@state-machine.com

THANK YOU

Embedded Online Conference

www.embeddedonlineconference.com