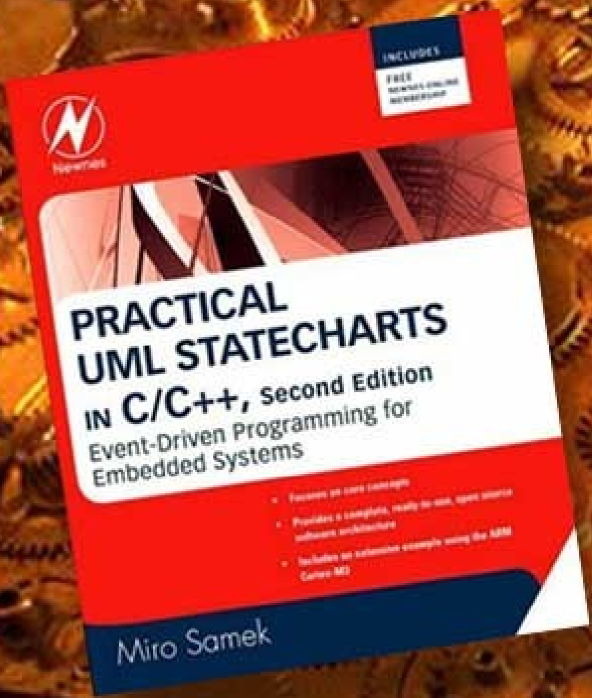
Practical UML Statecharts

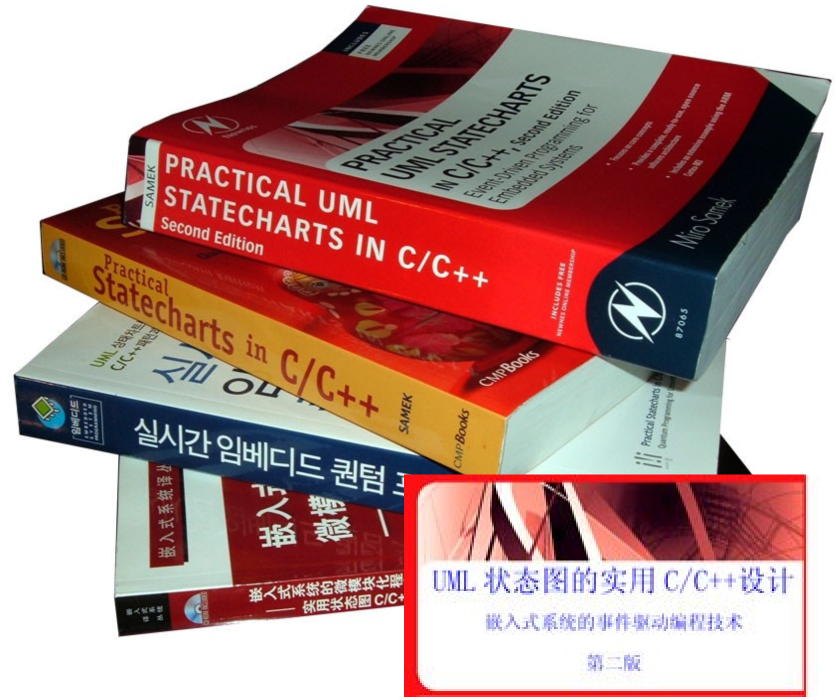
Miro Samek

miro@quantum-leaps.com



About the instructor

Dr. Miro Samek is the author of "*Practical UML Statecharts in C/C++, Second Edition: Event-Driven Programming for Embedded Systems*" (Newnes 2008), has written numerous articles for magazines, including a column for *C/C++ Users Journal*, is a regular speaker at the *Embedded Systems Conferences*, and serves on the editorial review board of the *Embedded Systems Design* magazine. For a number of years, he worked in various Silicon Valley companies as an embedded software architect and before that he worked as an embedded software engineer at GE Medical Systems (now GE Healthcare).



Contact: miro@quantum-leaps.com

Learn today. Design tomorrow.
ESD
EMBEDDED SYSTEMS DESIGN

Learn today. Design tomorrow.
ESC
Silicon Valley • May 2 - 5, 2011
McEnery Convention Center • San Jose

C/C++ Users Journal
Advanced Solutions for Professional Developers

Outline

Event-Driven Programming

- Hierarchical state machines
- Real-time frameworks

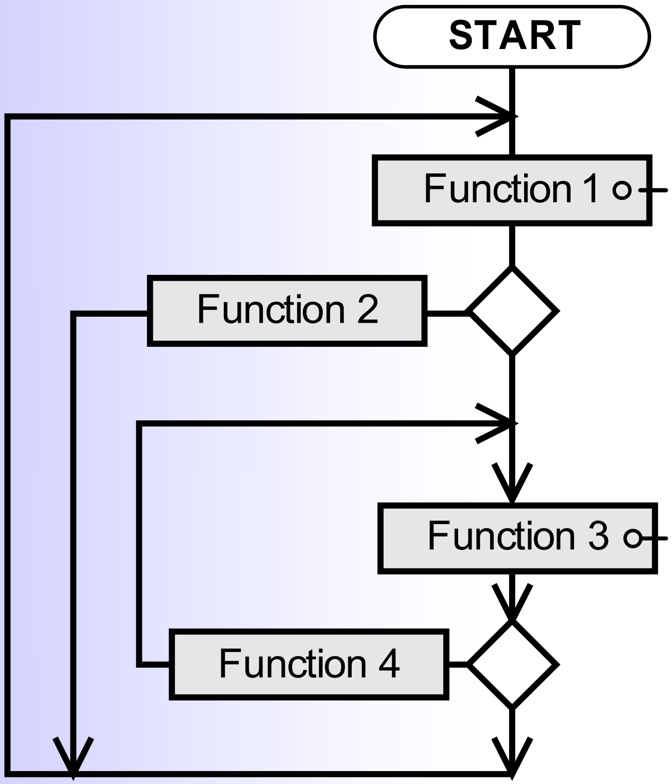
- Questions & Answers

Most computer systems are event-driven

Event-Driven System Example: Vending Machine

Traditional Sequential Program Flow

Program flow determined by sequence of instructions

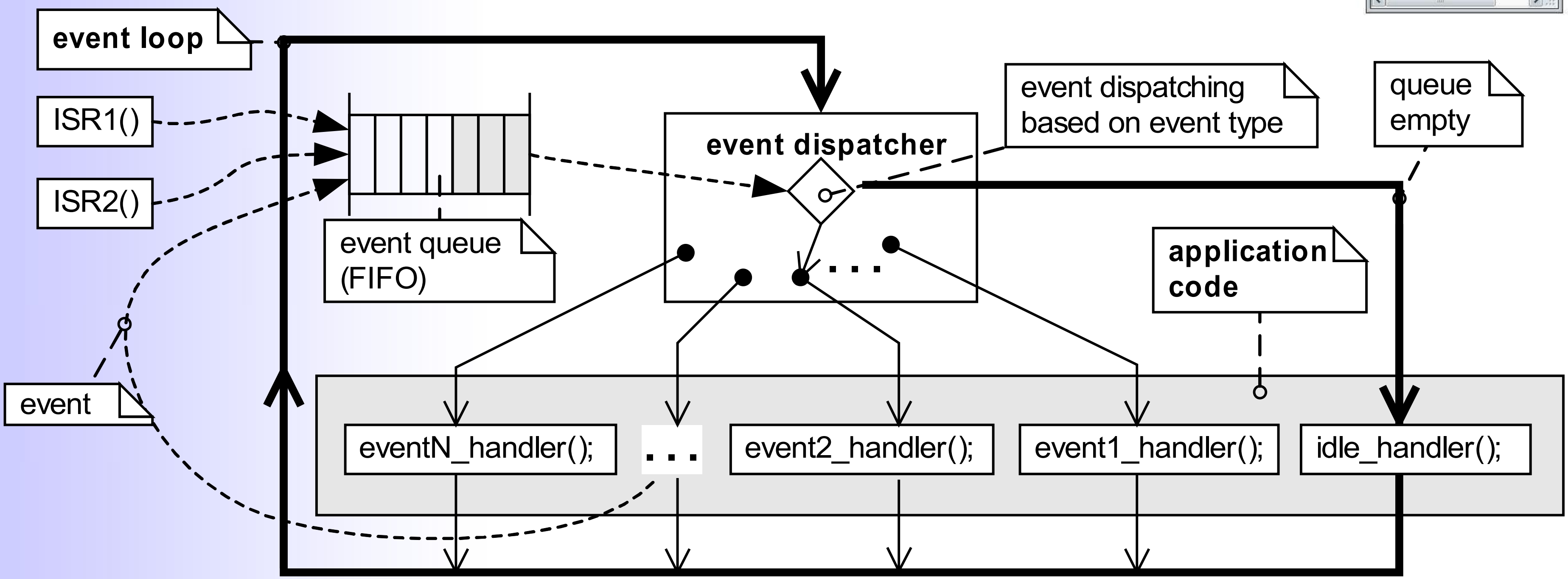



```
. . .
/* wait for Button 1 press */
while (Button1_GPIO != DEPRESSED) {
}
. . .
```

```
. . .
/* wait for Button 2 press */
OSSemPend(&Button2_Semaphore, ...);
. . .
```

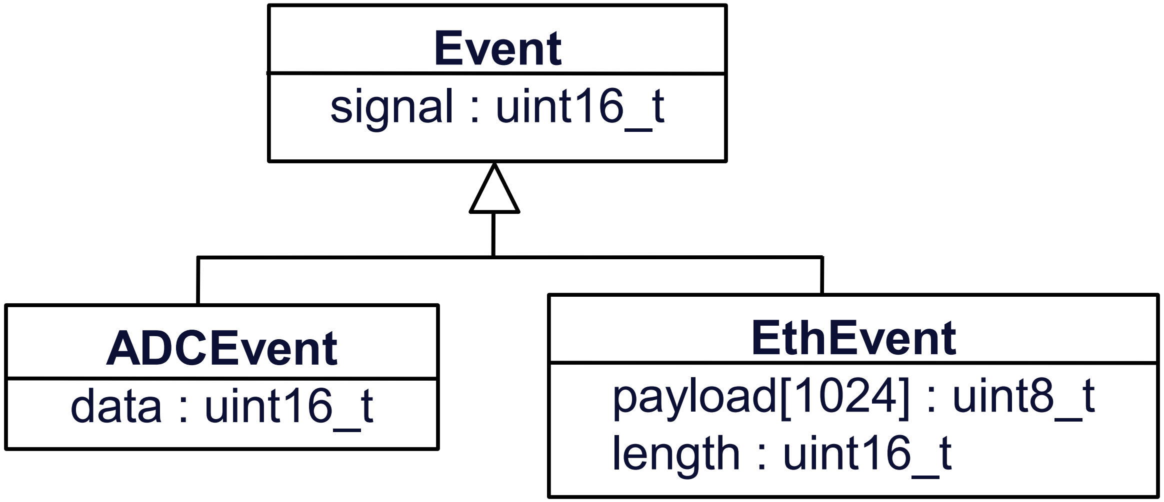
Event-Driven Program Flow

Program flow determined by order of <u>events</u>

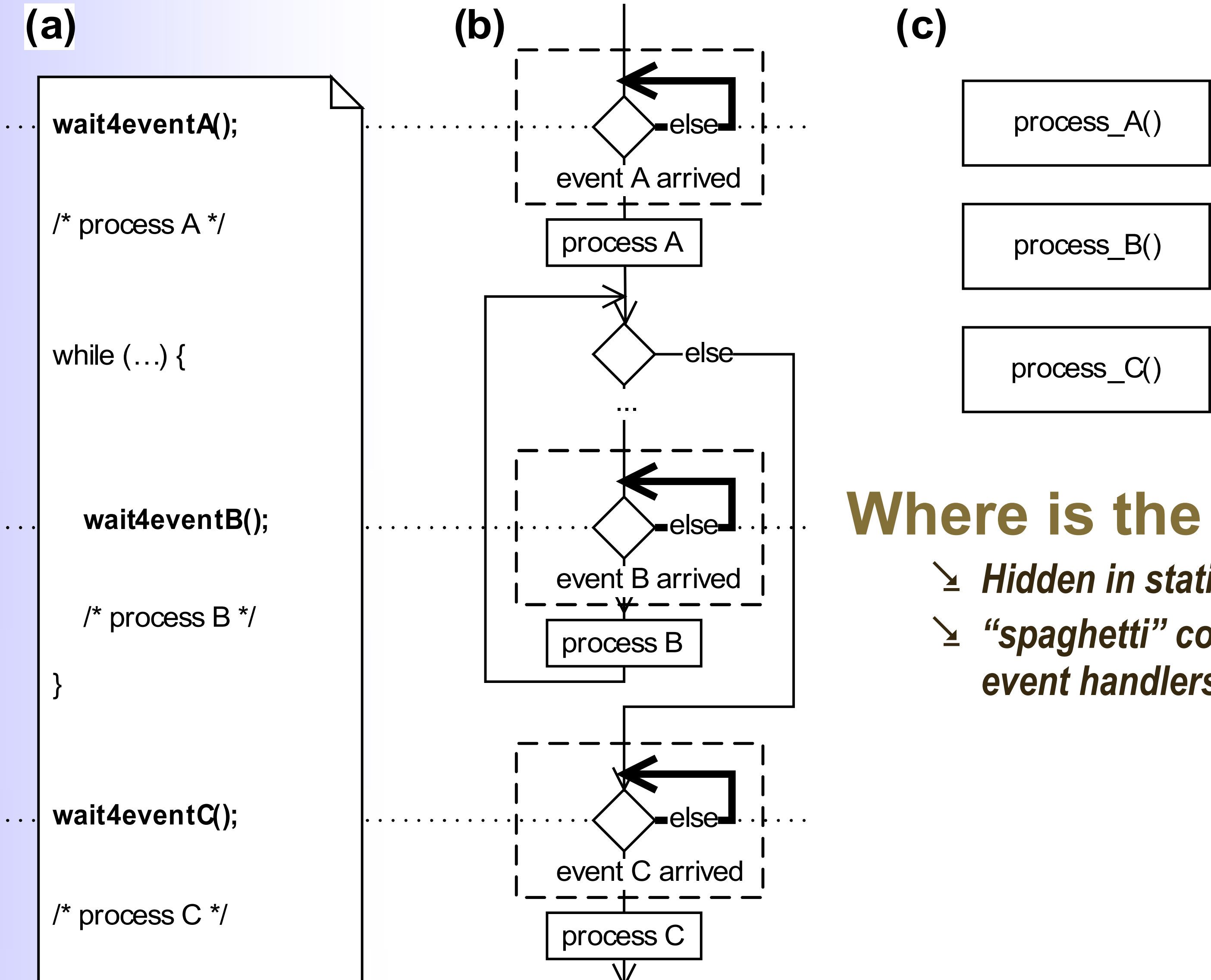


Event-driven program flow (cont'd)

- Events are first-class objects
- Events are processed asynchronously
- Events are processed in Run-to-Completion (RTC) fashion
- Events are queued



Challenges of event-driven programming



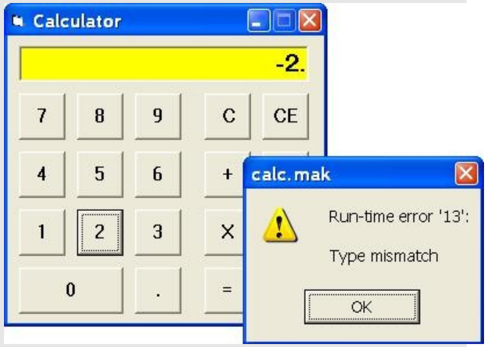
Where is the structure?

- *Hidden in static variables*
- *"spaghetti" code inside event handlers*

Event-action paradigm—spaghetti code

Bunch of flags and variables

```
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
        Case 1
        Op1 = Readout
        If Operator(Index).Caption ="-" And LastInput <> "NUMS"
           And OpFlag <> "=" Then
            Readout = "-"
            LastInput = "NEG"
        End If
```



Complex conditional code based on the flags and variables

Outline

- Event-driven programming

Hierarchical state machines

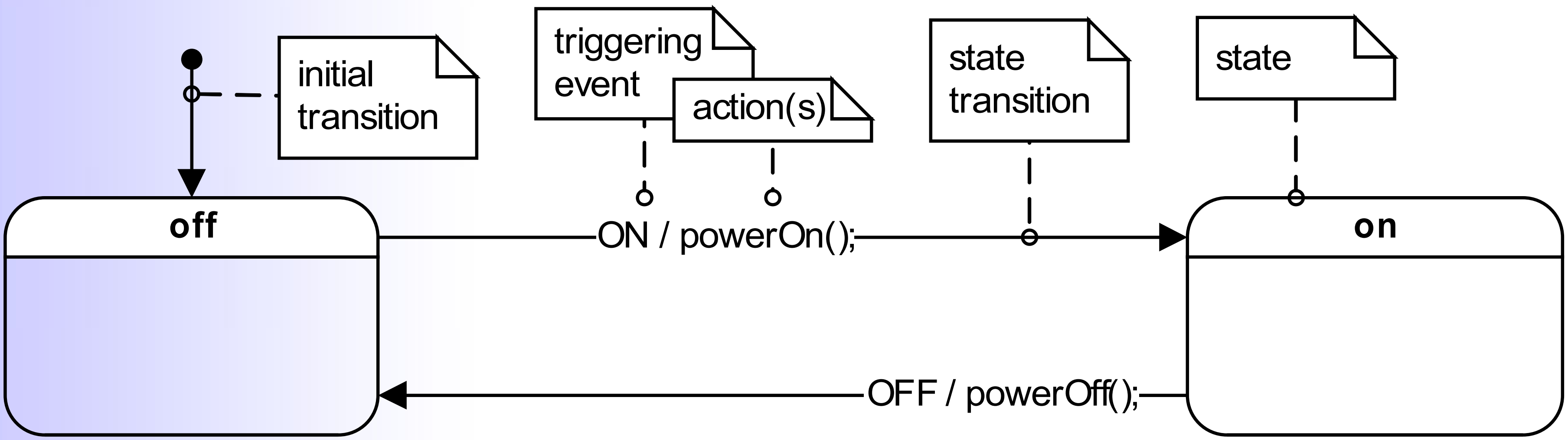
- Real-time frameworks

- Q & A

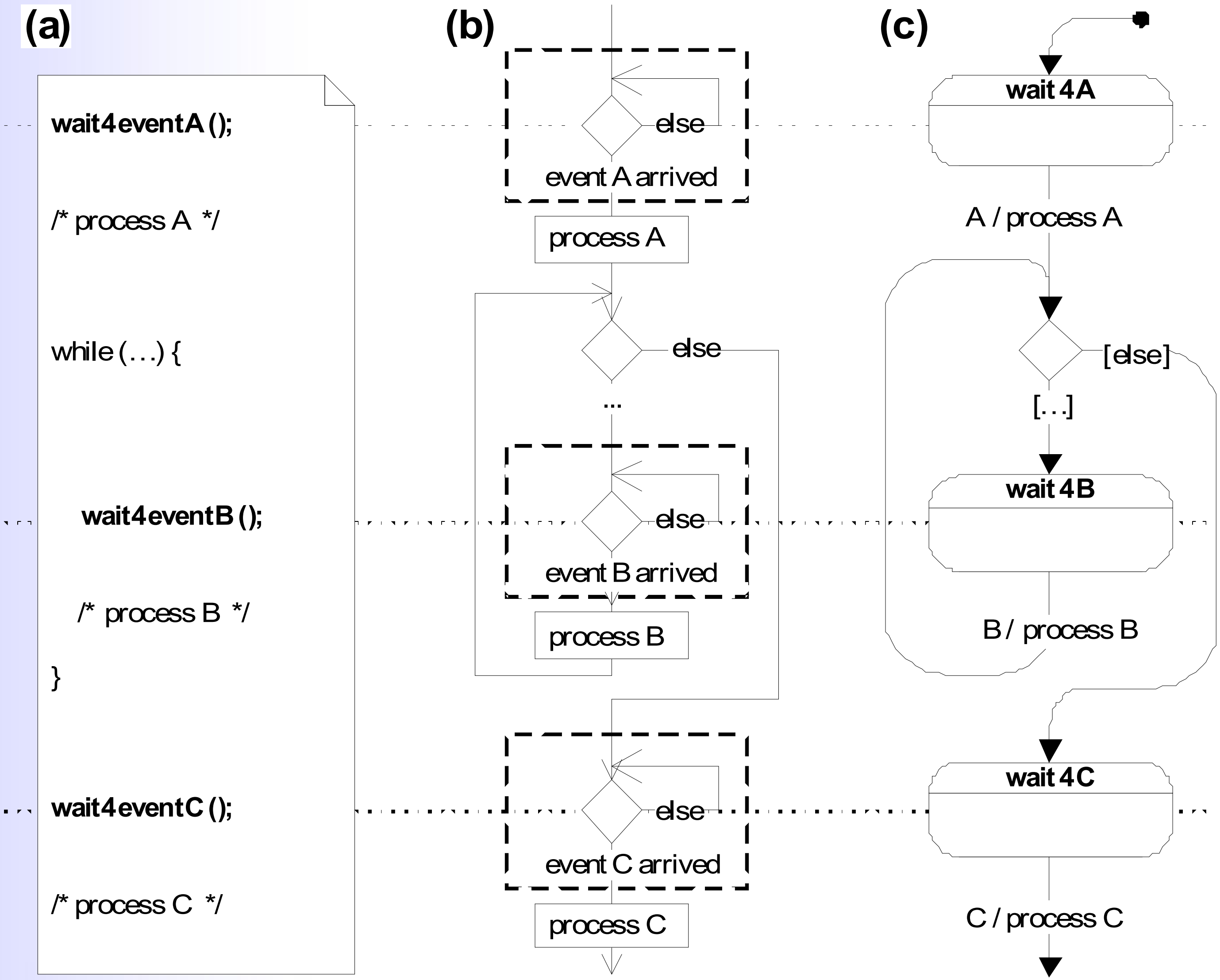
UML state machines (statecharts)

State machine

- **Event-action paradigm applied locally within each state**



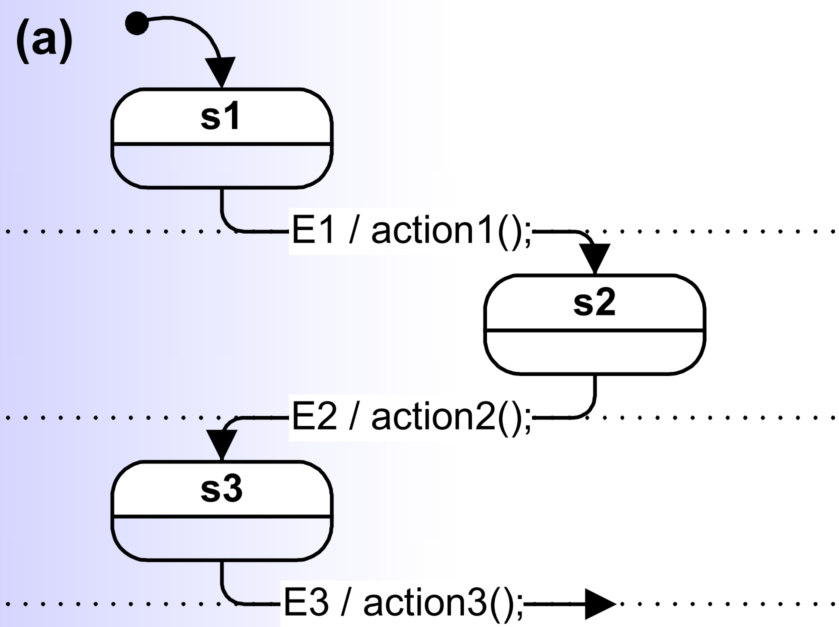
Recovering the structure with a state machine

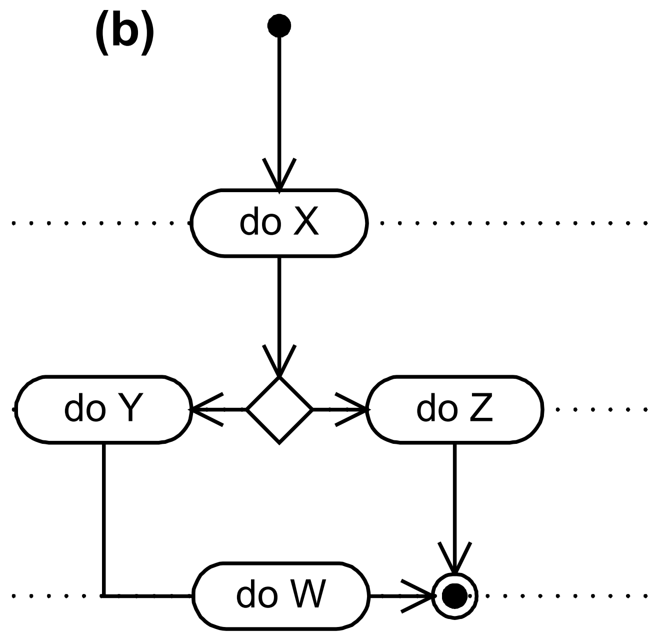


Statecharts vs. Flowcharts

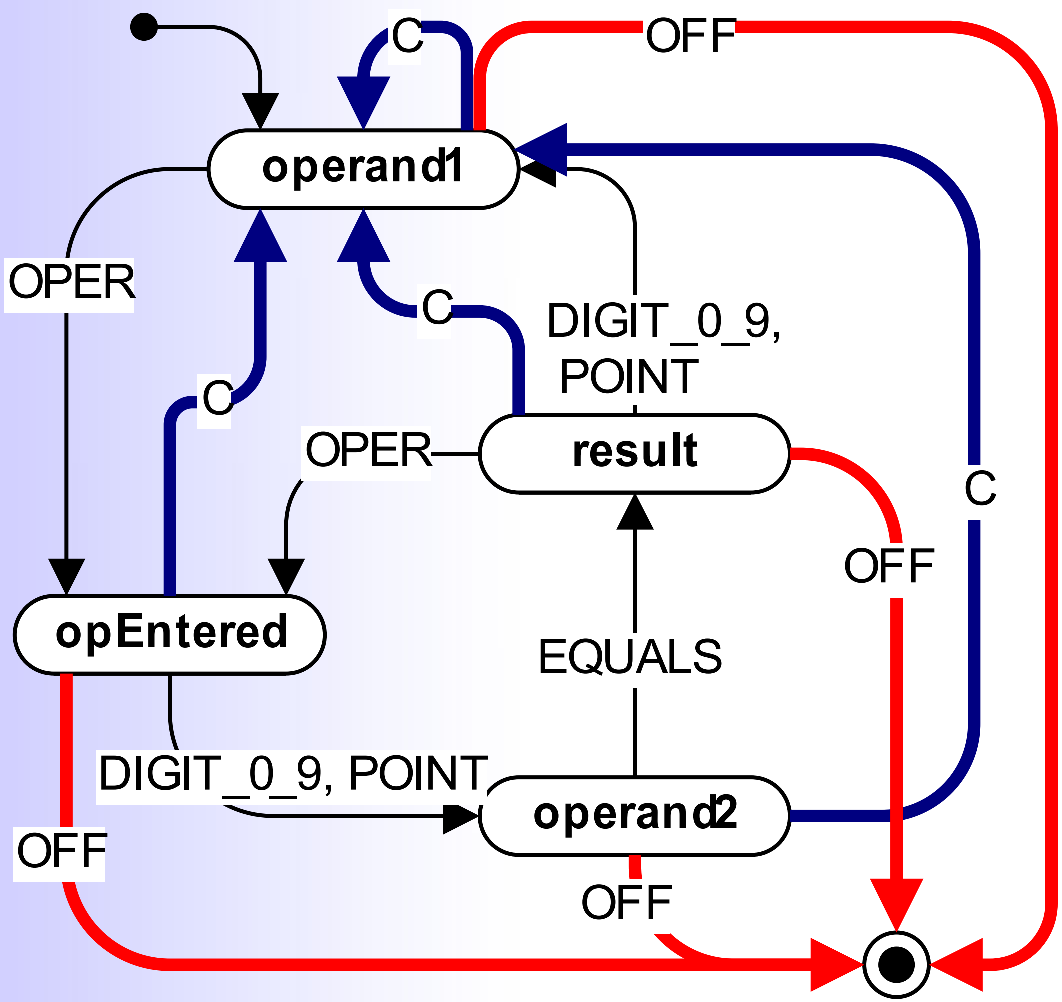
Completely distinct: different use of CPU!

- **Statechart: on the arrows**
  - ↘ *Otherwise CPU idle*
- **Flowchart: in the boxes**
  - ↘ *CPU never idle*

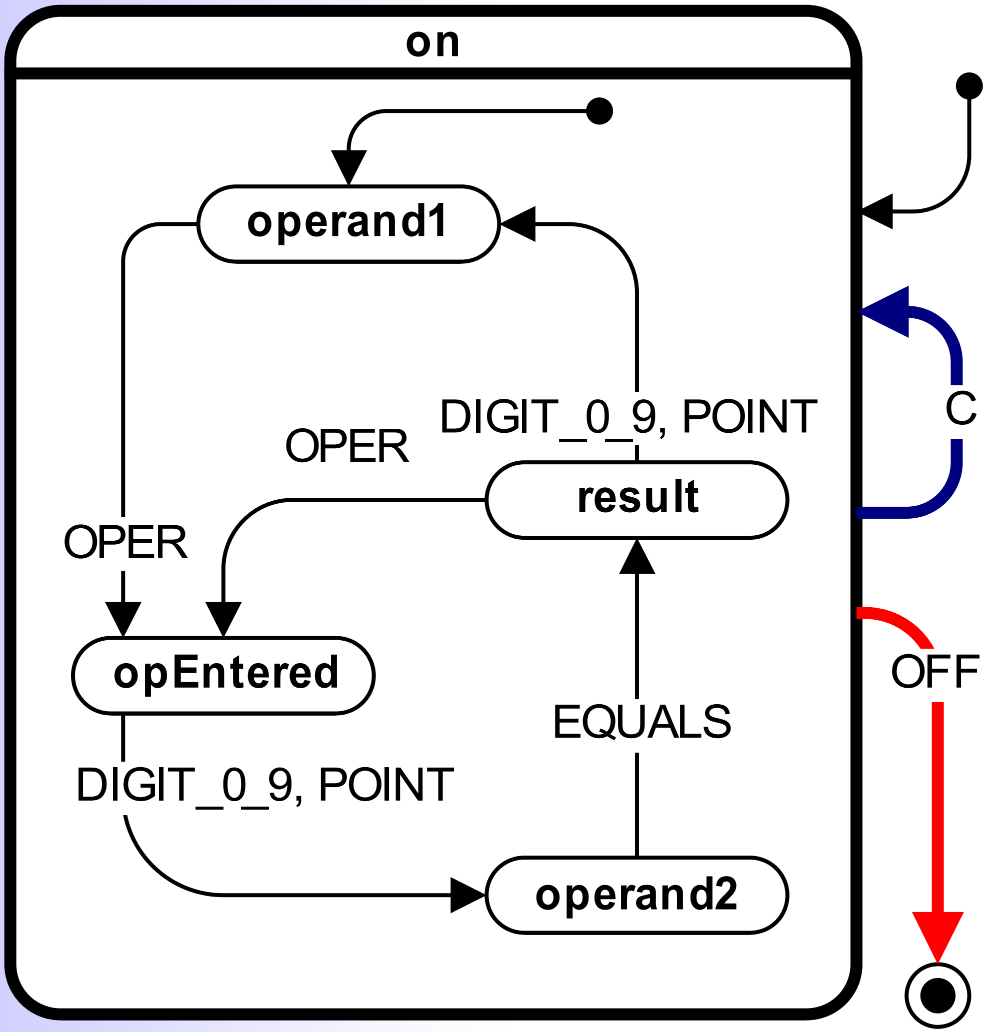




State-transition explosion



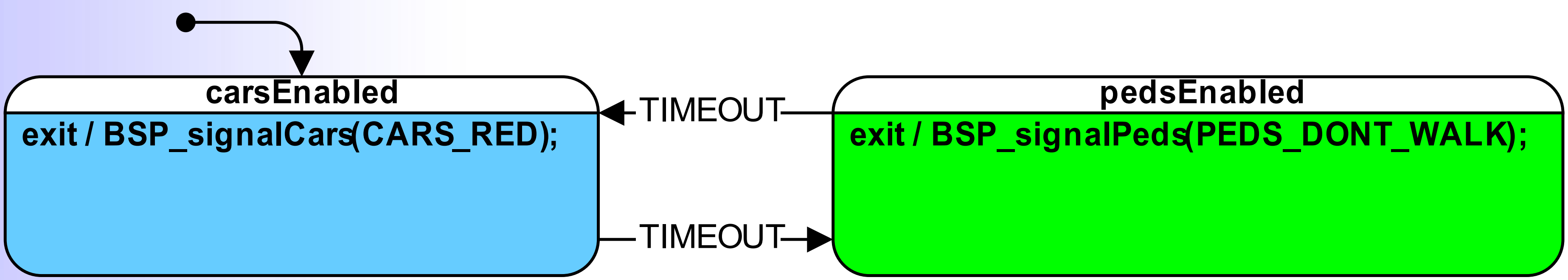
Reuse of behavior through state nesting



- **Programming by difference**
  - ↘ *Behavioral inheritance*

State entry and exit actions

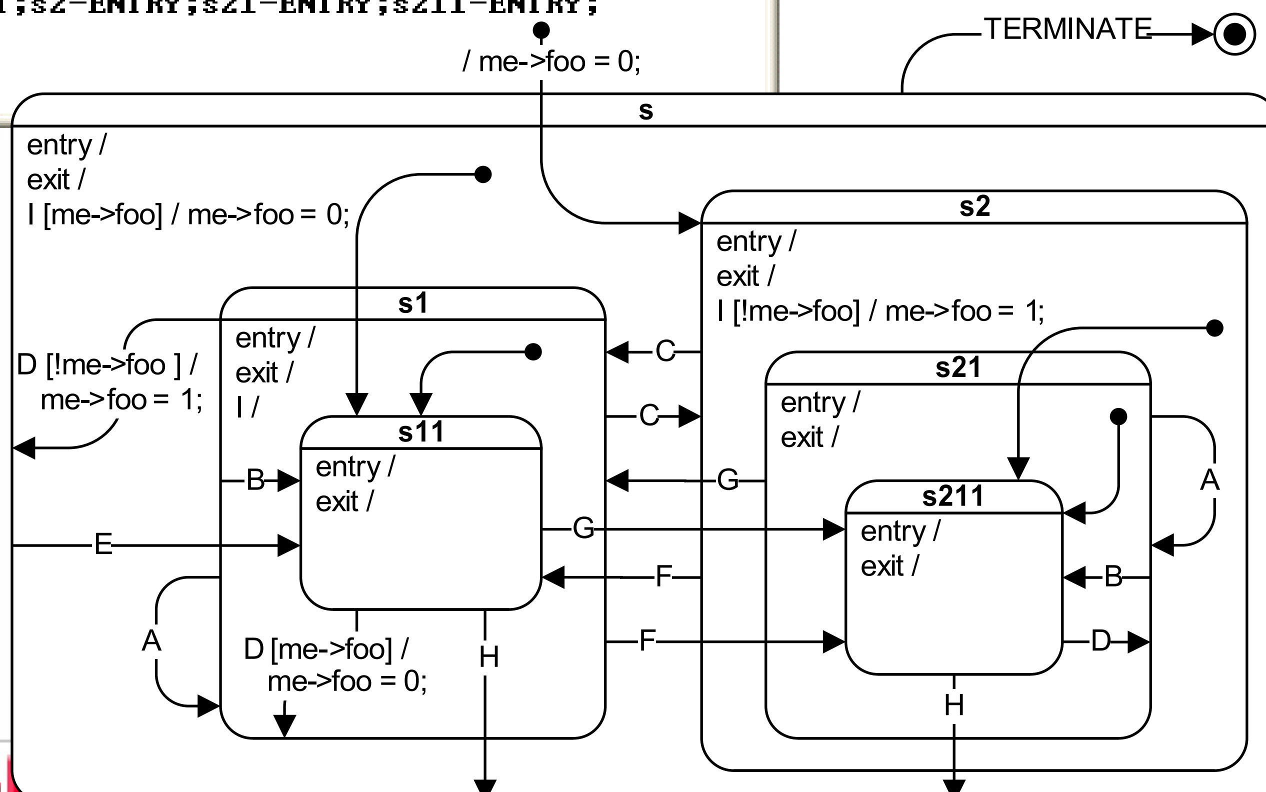
- Guaranteed initialization and cleanup
- Superstates are entered before substates
  - *like class constructors in OOP*
- Superstates are exited after substates
  - *like class destructors in OOP*



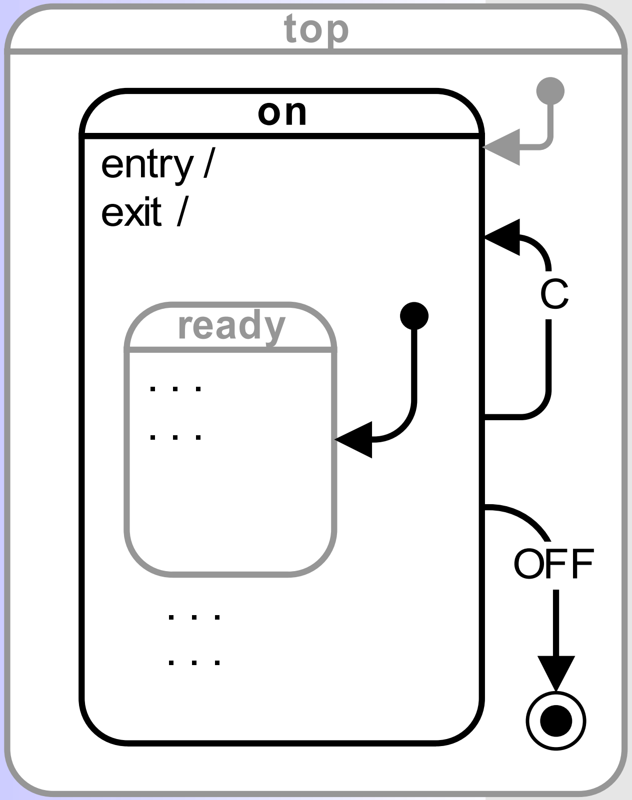
UML state machine semantics—QHsmTst example

Command Prompt - dbg\qhsmtst

```
     QHsmTst example, built on Sep 25 2007 at 09:11:31,
     QEP/C: 3.4.01.
     Press ESC to quit...
(1)  top-INIT;s-ENTRY;s2-ENTRY;s2-INIT;s21-ENTRY;s211-ENTRY;
(2)  >G: s21-G;s211-EXIT;s21-EXIT;s2-EXIT;s1-ENTRY;s1-INIT;s11-ENTRY;
(3)  >I: s1-I;
(4)  >A: s1-A;s11-EXIT;s1-EXIT;s1-ENTRY;s1-INIT;s11-ENTRY;
(5)  >D: s1-D;s11-EXIT;s1-EXIT;s-INIT;s1-ENTRY;s11-ENTRY;
(6)  >D: s11-D;s11-EXIT;s1-INIT;s11-ENTRY;
(7)  >C: s1-C;s11-EXIT;s1-EXIT;s2-ENTRY;s2-INIT;s21-ENTRY;s211-ENTRY;
(8)  >E: s-E;s211-EXIT;s21-EXIT;s2-EXIT;s1-ENTRY;s11-ENTRY;
(9)  >E: s-E;s11-EXIT;s1-EXIT;s1-ENTRY;s11-ENTRY;
(10) >G: s11-G;s11-EXIT;s1-EXIT;s2-ENTRY;s21-ENTRY;s211-ENTRY;
(11) >I: s2-I;
(12) >I: s-I;
(13) >←: Bye, Bye!
```



Coding a HSM in QP/C++




```
QState Calc::on(Calc *me, QEvent const *e) {
    switch (e->sig) {
        case Q_ENTRY_SIG: {    // entry action
            BSP_message("on-ENTRY");
            return Q_HANDLED();
        }
        case Q_EXIT_SIG: {     // exit action
            BSP_message("on-EXIT");
            return Q_HANDLED();
        }
        case Q_INIT_SIG: {     // initial transition
            BSP_message("on-INIT");
            return Q_TRAN(&Calc::ready);
        }
        case C_SIG: {          // state transition
            BSP_clear();       // clear the display
            return Q_TRAN(&Calc::on);
        }
        case OFF_SIG: {        // state transition
            return Q_TRAN(&Calc::final);
        }
    }
    return Q_SUPER(&QHsm::top); // superstate
}
```

Outline

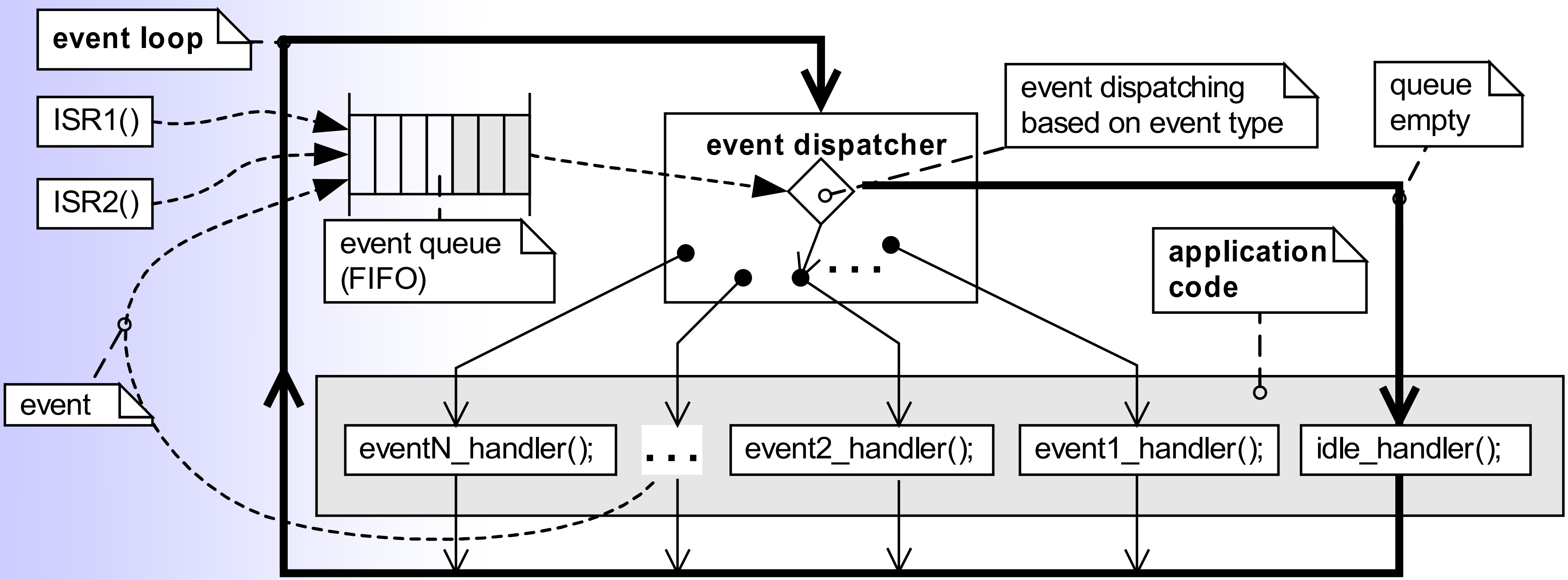
- Event-driven programming
- Hierarchical state machines

Real-time frameworks

- Q & A

Problems with the simple event-loop

- Single event queue prevents prioritization of work
- Event dispatcher is incompatible with state machines



Vertical vs. Horizontal Slicing

Slicing by event-signal destroys the notion of state

Event-handlers — Events →, States ↓

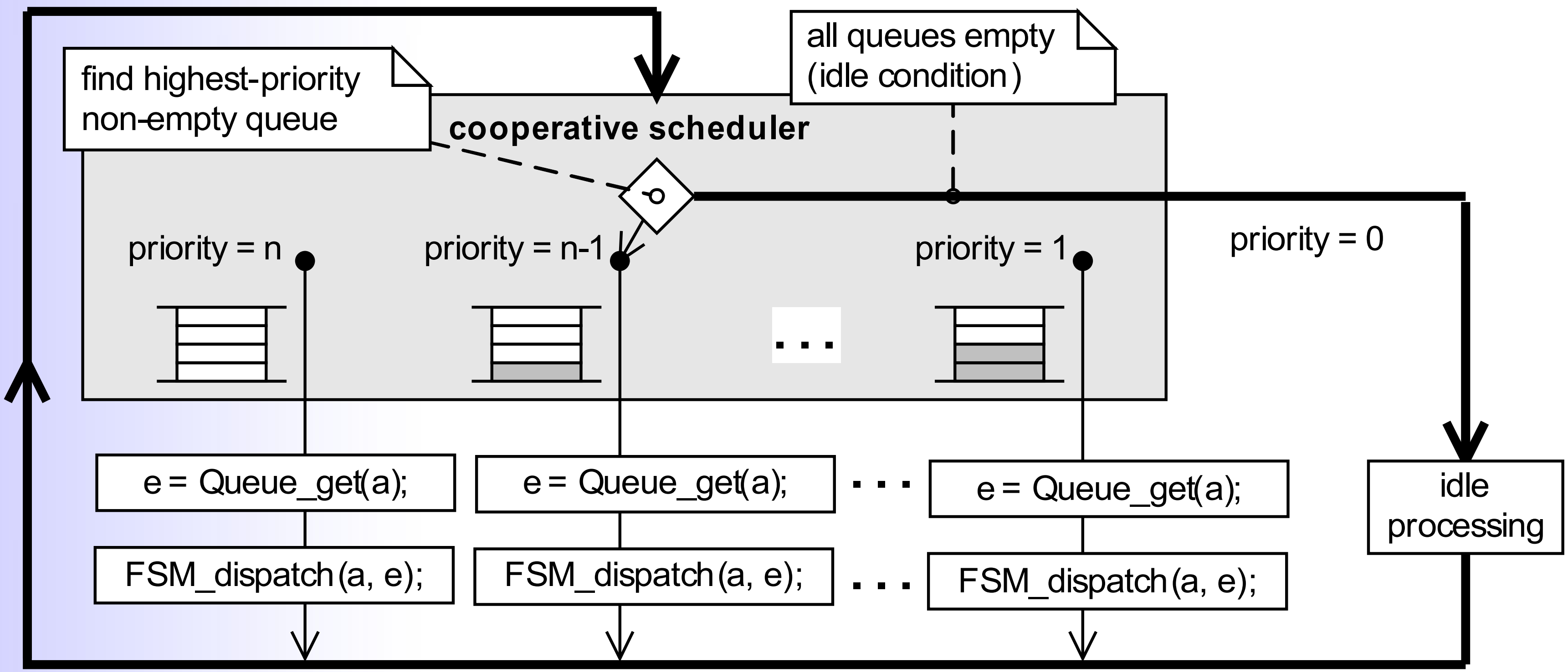
States	ON_SIG	OFF_SIG
"off"	{ powerOn (), "on" }	
"on"		{ powerOff (), "off" }

State-handlers — Events →, States ↓

States	ON_SIG	OFF_SIG
"off"	{ powerOn (), "on" }	
"on"		{ powerOff (), "off" }

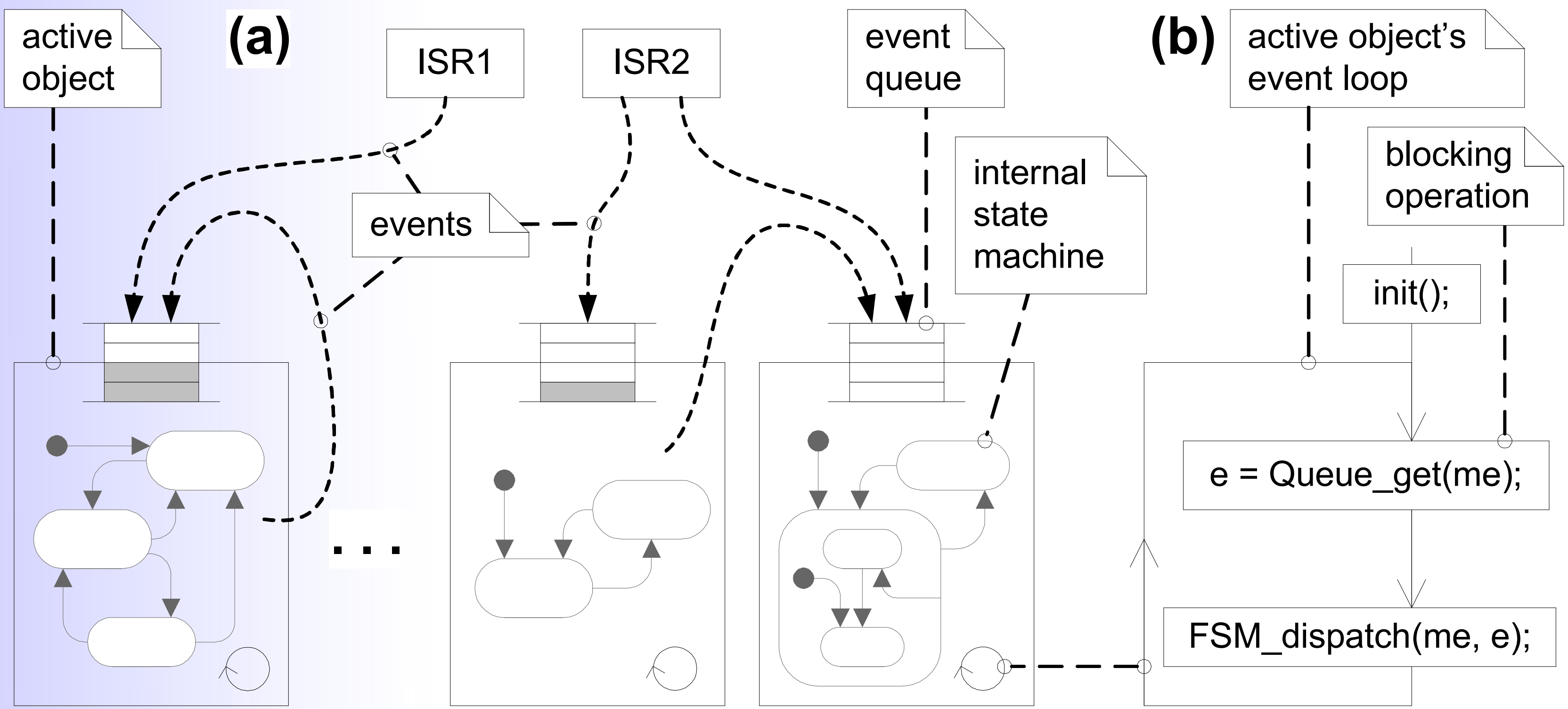
State machine framework based on cooperative kernel

- Use multiple priority queues bound to state machines
- Don't sort events based on the signal (vertical slicing)

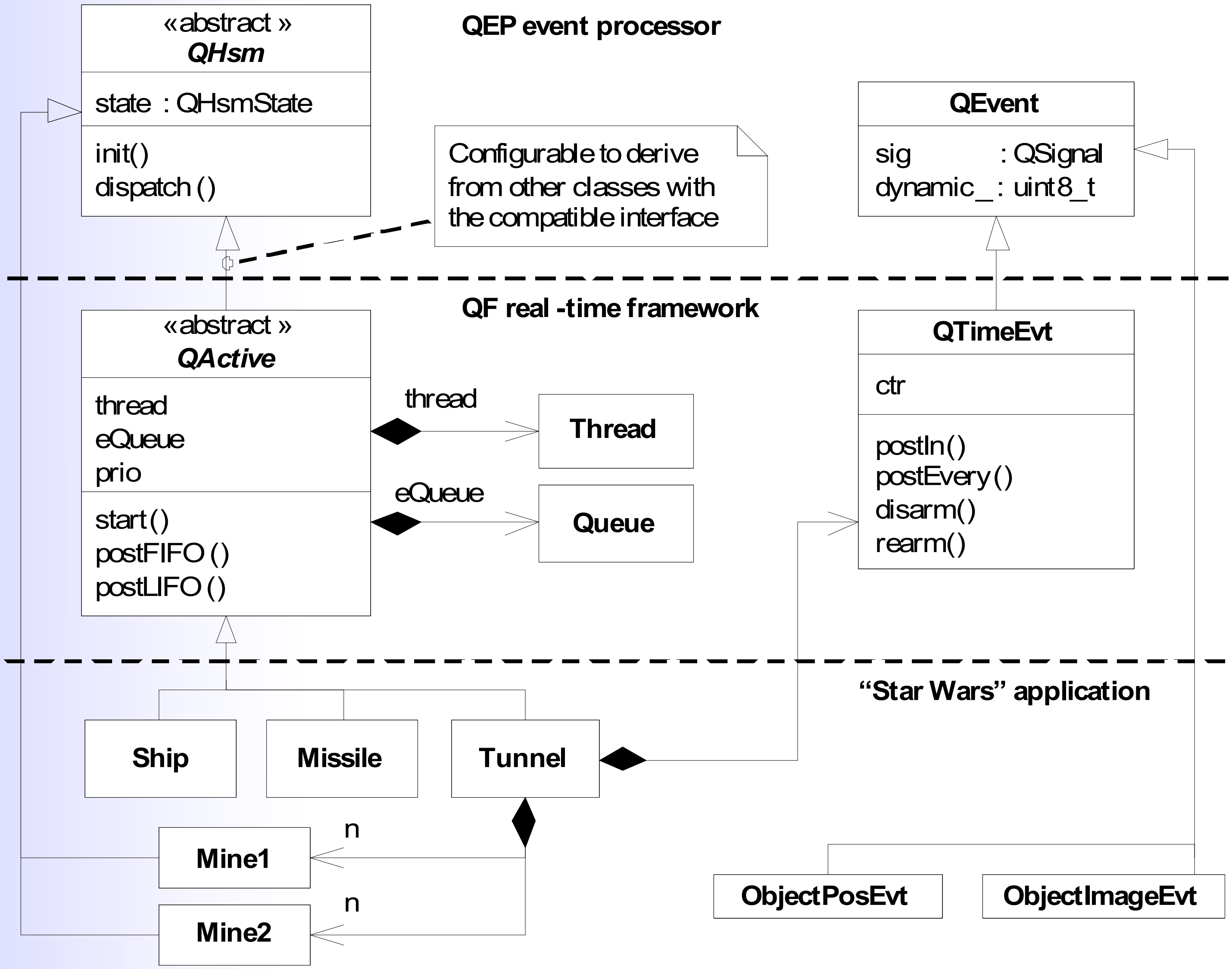


State machine framework based on preemptive kernel

- RTC does not mean that state machines cannot preempt each other
- Each state machine executes in its own thread of control
  - ↘ *(State Machine + Event Queue + Thread) = Active Object*



Minimal active object framework (QP)



Summary

State machines complement imperative languages (C, C++, Java, C#, etc.)

State machines "explode" without state hierarchy

State machines are impractical without a framework

Once you try an event-driven, state machine framework you will not want to go back to "spaghetti" and raw RTOS/OS

Welcome to the 21 century!

Outline

- Event-driven programming
- Hierarchical state machines
- Real-time frameworks

Questions & Answers