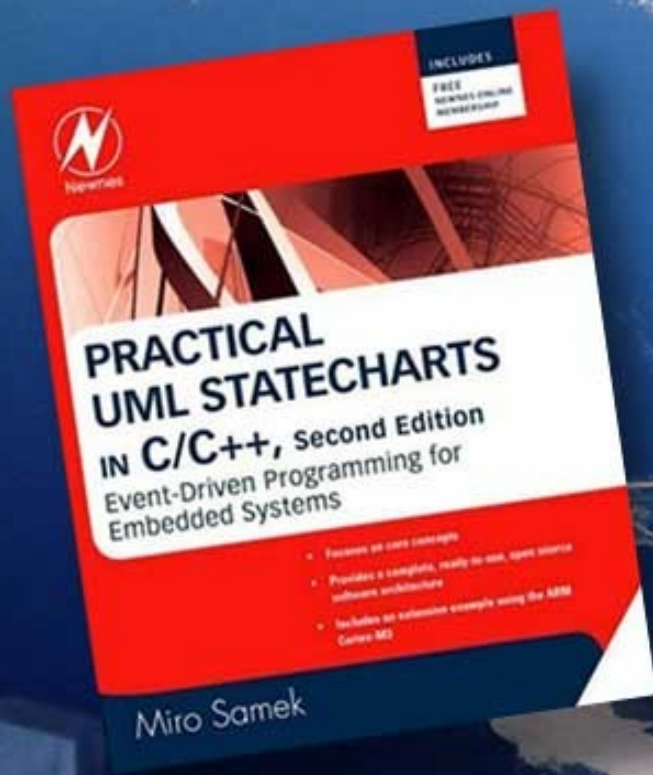


# Quick Overview of QP Frameworks and QM Tool



**Quantum<sup>®</sup>Leaps**  
innovating embedded systems

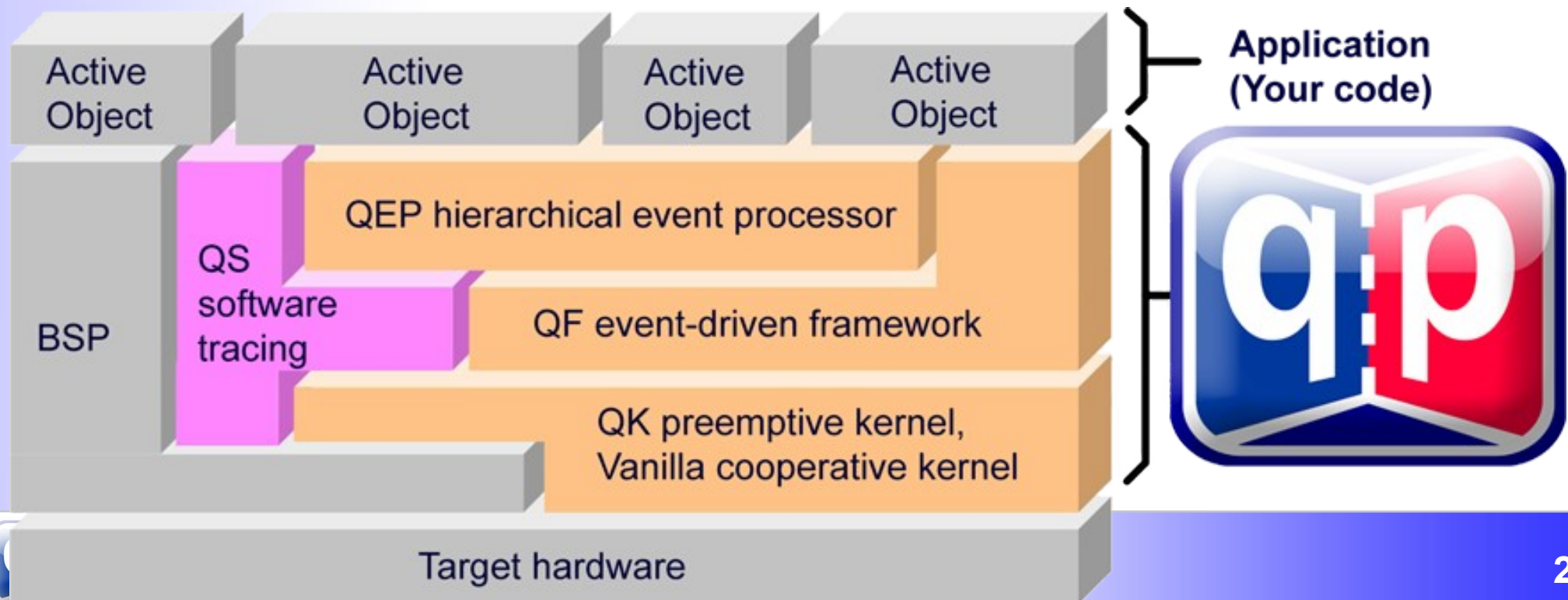
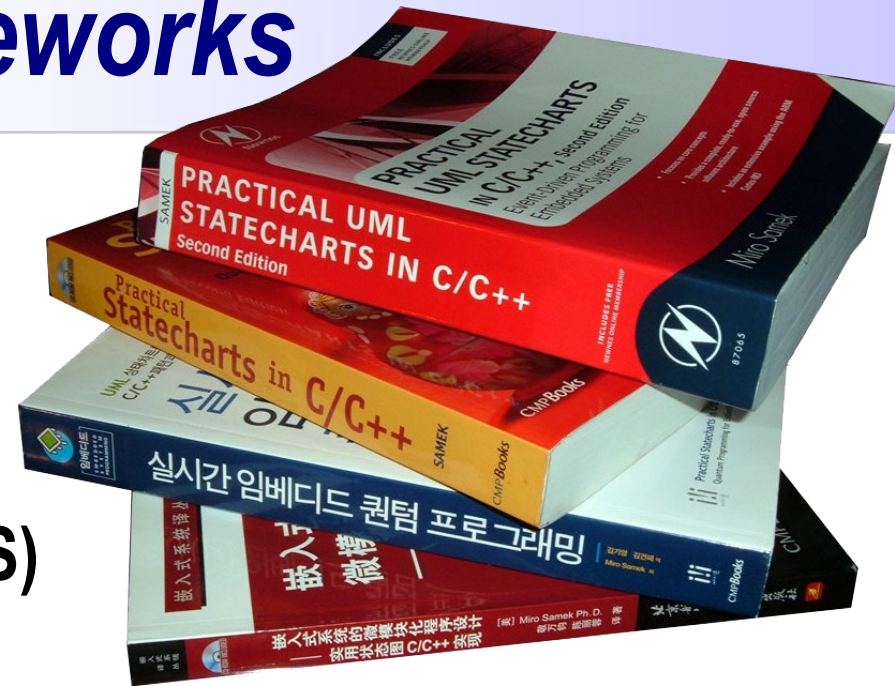
© Quantum Leaps  
state-machine.com

[info@quantum-leaps.com](mailto:info@quantum-leaps.com)

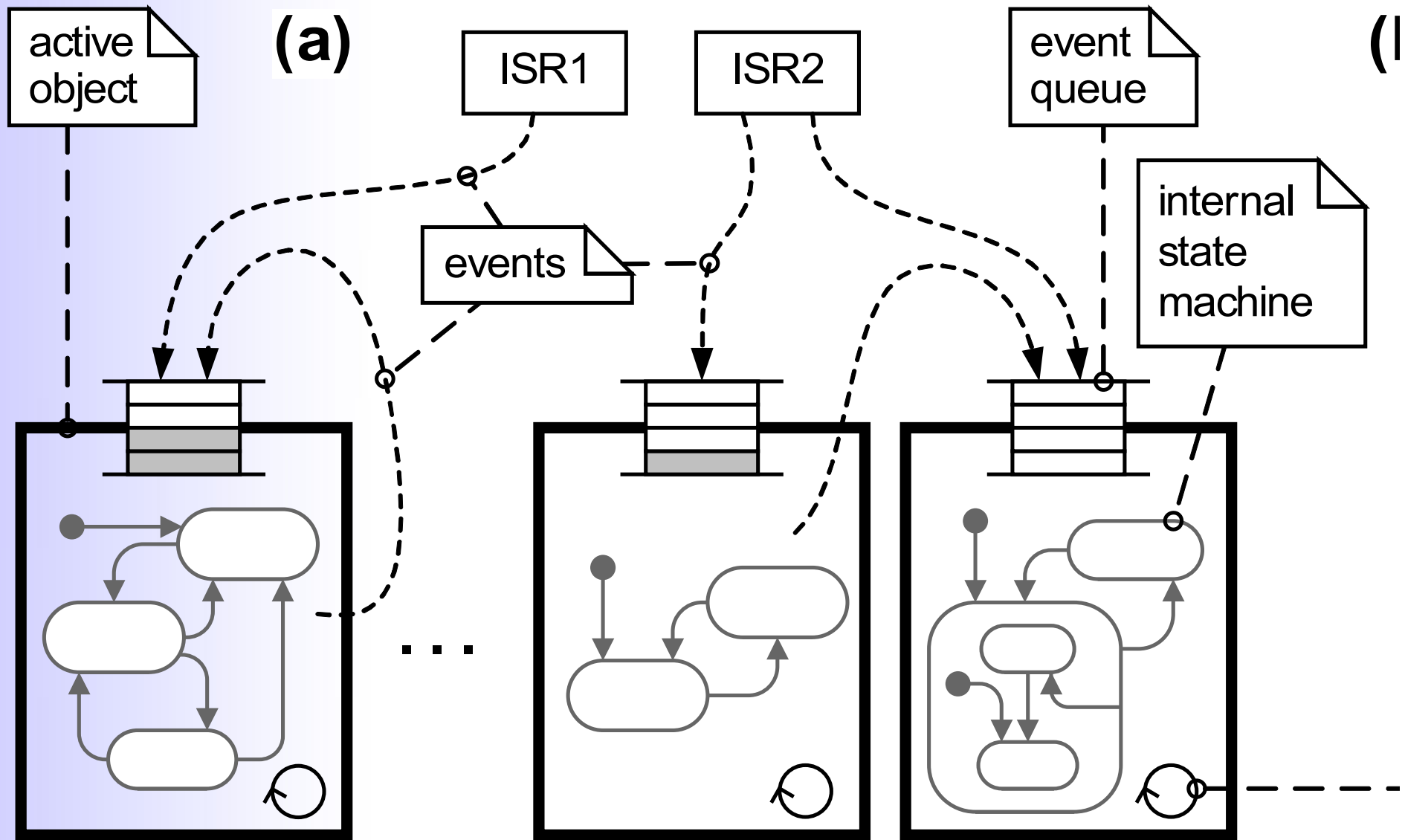
# QP state machine frameworks

## QP combines:

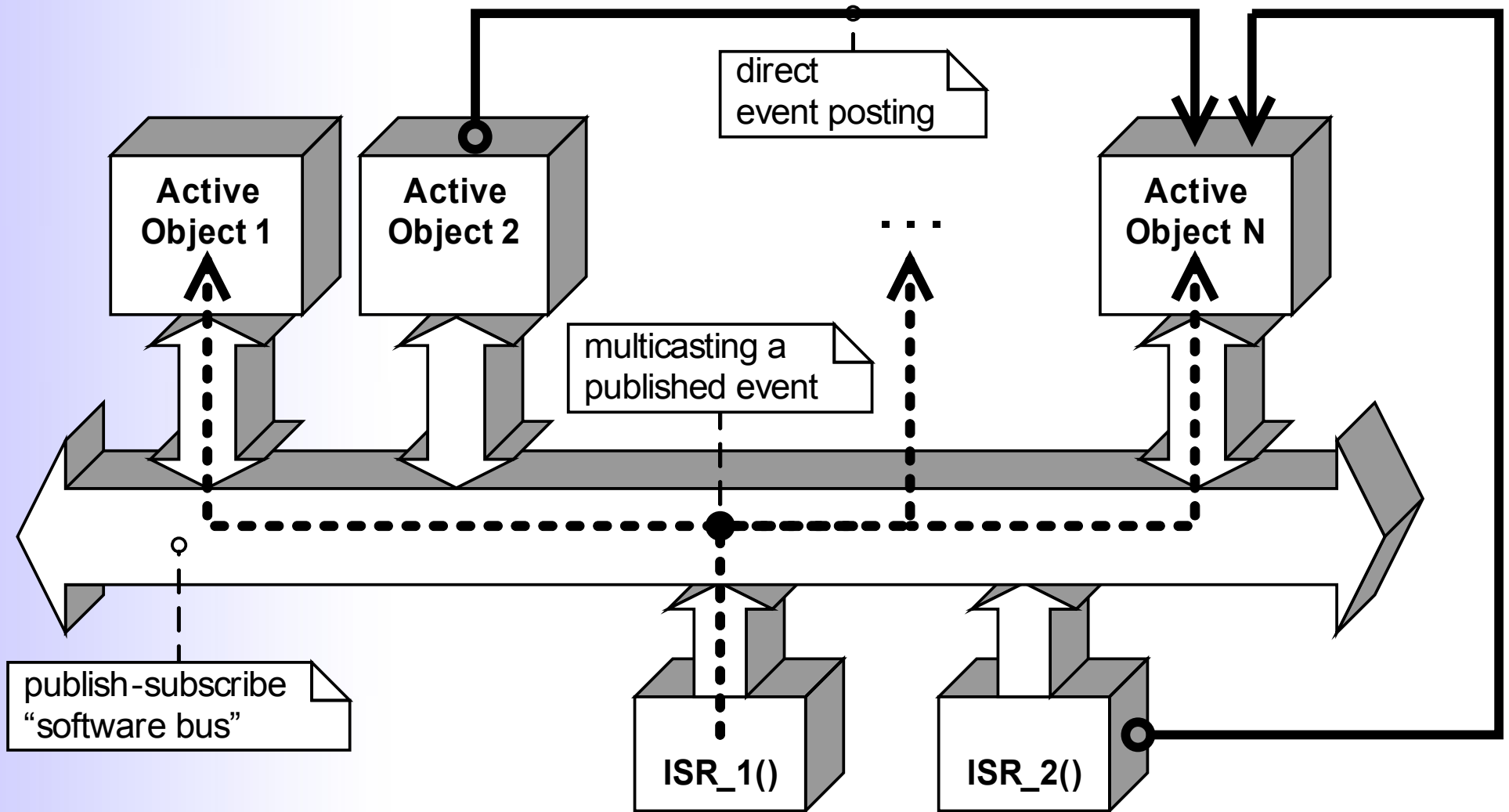
- Hierarchical state machines (QEP)
- Event-driven framework (QF)
- Real-time kernels (QK/Vanilla/RTOS)
- Software tracing (QS)



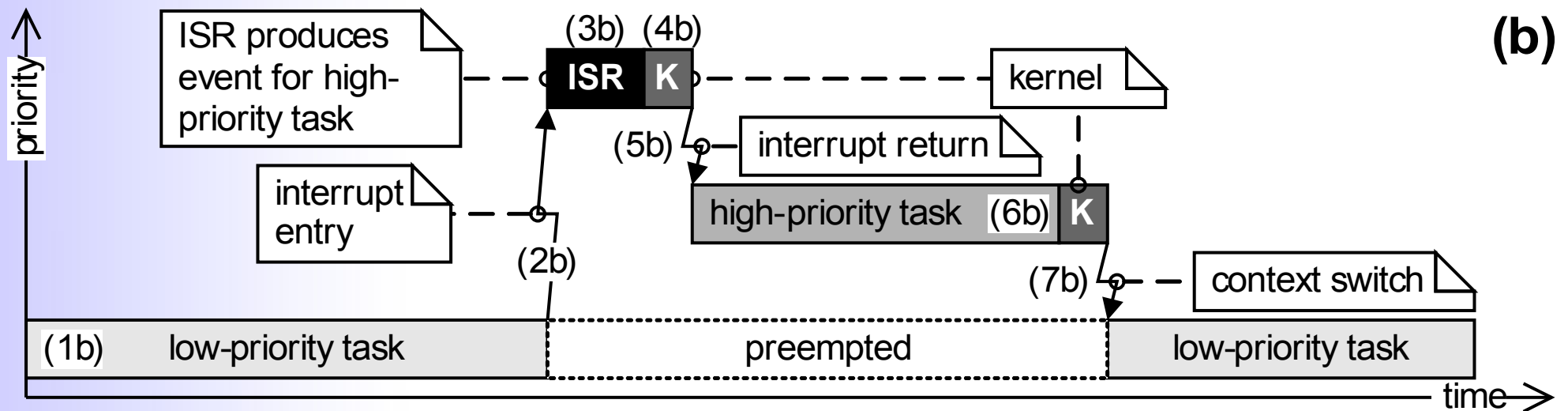
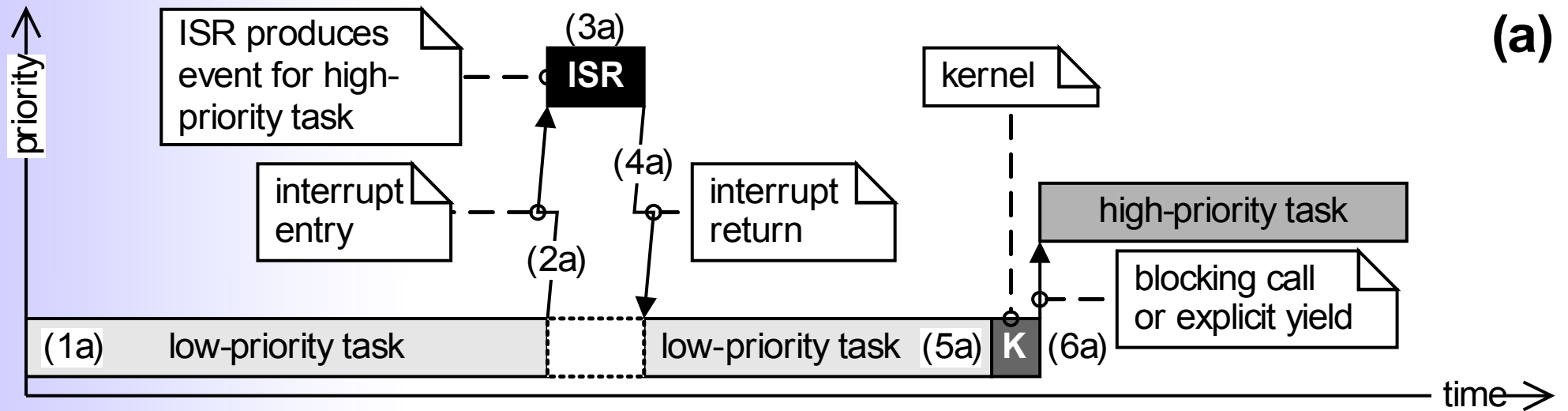
# Active Object Computing Model



# Event delivery mechanisms

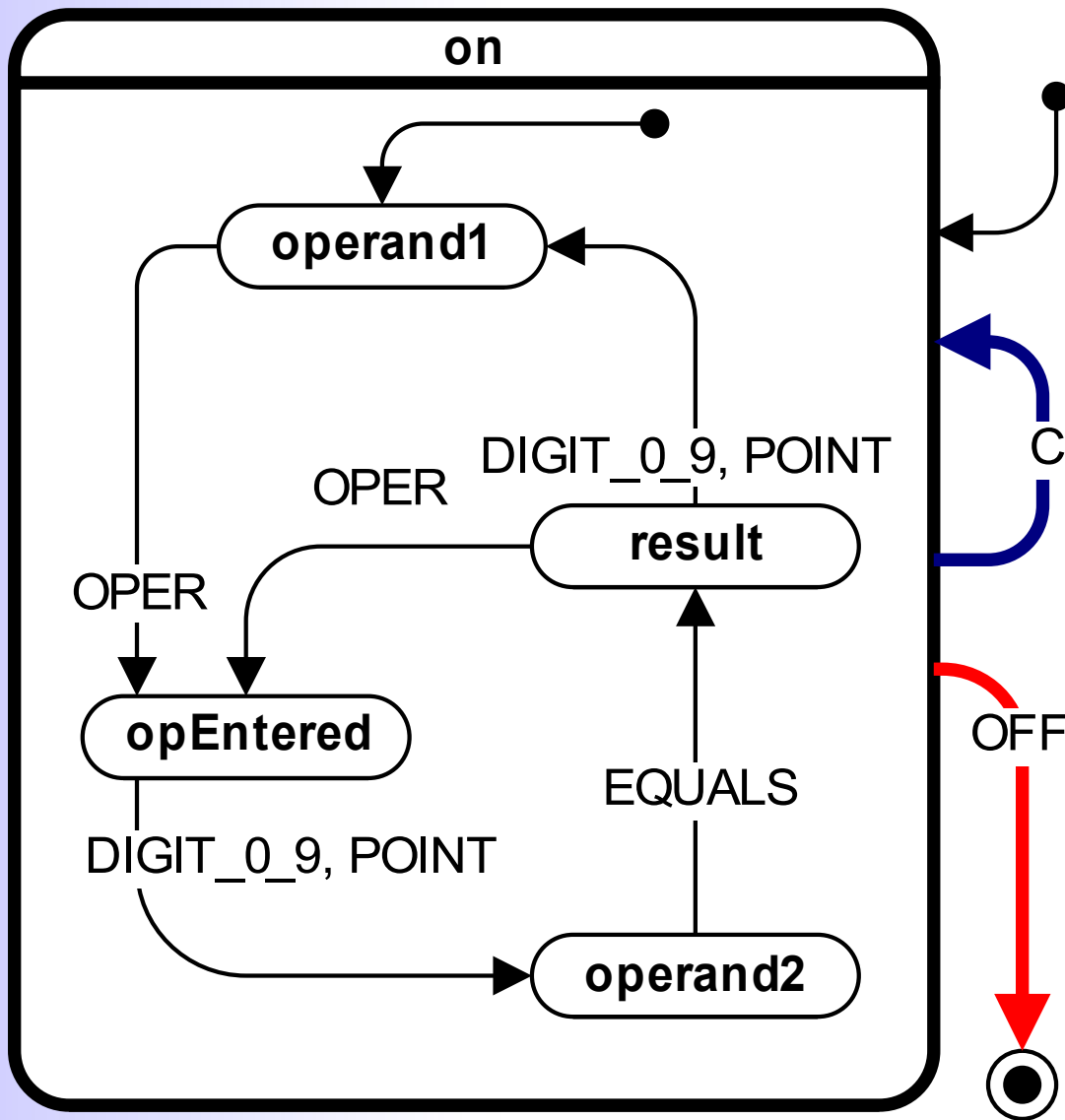


# Execution Profiles of Vanilla and QK Kernels





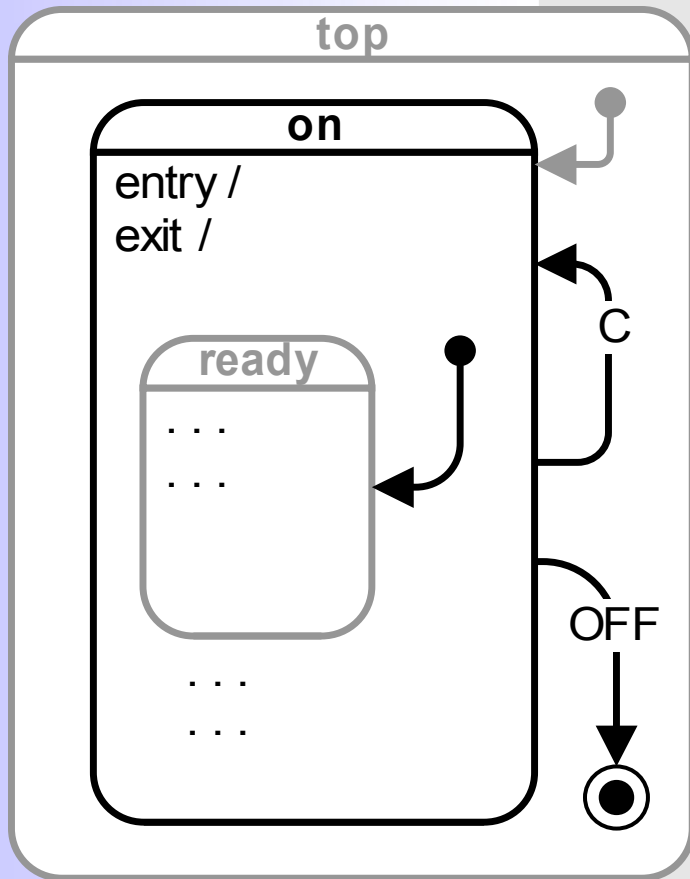
# Reuse of behavior through state nesting



**Hierarchical state machines don't explode**

- Because they reuse behavior

# Manual 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
}
```

# Automatic coding a HSM with QM

The screenshot displays the QM (Qt Modeler) software interface for developing a Hierarchical State Machine (HSM) for a Pelican. The main window shows the statechart of the Pelican, which is a hierarchical state machine with several states and transitions.

**Statechart of Pelican:**

- operational** (Superstate)
  - entry / CARS\_RED; PEDS\_DONT\_WALK
  - TERMINATE / OFF
  - carsEnabled** (Substate)
    - exit /
    - carsGreen** (Substate)
      - entry / CARS\_GREEN
      - exit /
      - carsGreenNoPed** (Substate)
        - entry / PEDS\_WAITING
        - TIMEOUT
      - carsGreenInt** (Substate)
        - entry / PEDS\_WAITING
      - carsGreenPedWait** (Substate)
        - entry / TIMEOUT
      - carsYellow** (Substate)
        - entry / CARS\_YELLOW
        - exit /
        - TIMEOUT
  - pedsEnabled** (Substate)
    - exit /
    - pedsWalk** (Substate)
      - entry / PEDS\_WALK
      - exit /
      - TIMEOUT
    - pedsFlash** (Substate)
      - entry /
      - exit /
      - TIMEOUT
      - Decision diamond:  $[me \rightarrow flashCtr \neq 0] / \neg me \rightarrow flashCtr$ 
        - [else]
      - Decision diamond:  $[(me \rightarrow flashCtr \& 1) == 0] / BSP\_signalPeds(PEDS\_BLANK);$ 
        - [else] / BSP\\_signalPeds(PEDS\\_DONT\\_WALK)

**Property Editor: State**

name: pedsFlash  
superstate: pedsEnabled  
entry:

```
BSP_showState("pedsFlash");  
QTimeEvt_postEvery(&me->timeout,  
me->flashCtr = PEDS_FLASH_NUM*2 +
```

exit:

```
QTimeEvt_disarm(&me->timeout);
```

**Bird's Eye View**

The Bird's Eye View shows a zoomed-out overview of the entire statechart, highlighting the hierarchical structure and the current state being edited.

# Main benefits of QP

## Higher level of abstraction than RTOS (productivity)

- You work at the level of events and state machines, not the raw RTOS
- Your design is more structured, maintainable, testable, and robust



## Safer programming model

- You don't need to use semaphores, and other such troublesome RTOS mechanisms

## More efficient in time and space

- You use less CPU (no blocking) and less RAM (less stack space)
- QP is smaller in RAM use and code size (ROM) than any RTOS!

# Who is using QP?



# Supported processors (bare metal)

QP is very portable to 8-, 16-, 32-bit MCUs

- ARM7/9, ARM Cortex
- ColdFire
- Nios II
- M16C/R8C/M32C, H8
- MSP430
- AVRmega, AVRXmega
- TMS320C28
- PsoC
- others upon request...



# Supported operating systems

## QP can work with traditional OS/RTOS

- Linux/embedded Linux
- QNX, Integrity (POSIX)
- Windows/WindowsCE
- VxWorks
- ThreadX
- eCos
- uC/OS-II
- FreeRTOS.org
- others...



# Supported middleware

QP can work with middleware directly (bare metal)

- LwIP embedded TCP/IP stack
- emWin / uC/GUI embedded GUI
- others upon request...



# QM graphical modeling tool

QP is supported by the free graphical QM modeling tool

- Best-in-class state machine diagramming
- Automatic code generation based on QP

The screenshot displays the QM graphical modeling tool interface. On the left, a file explorer shows a project structure with folders like 'Statechart' and 'active'. The main workspace shows a statechart for 'Mine2' with states like 'unused', 'planted', 'exploding', 'active', 'demo', and 'screen\_saver'. A large, stylized 'qm' logo is overlaid on the statechart. On the right, a 'Property Editor State' window shows C++ code for the 'screen\_saver\_show' state, including initialization and transition logic. A 'Log' window at the bottom right shows the output of code generation, indicating that 6 files were generated and processed successfully.



# Summary

**QP is a lightweight event-driven, state machine framework**

- Higher level of abstraction (productivity) and safer than a raw RTOS
- Excellent documentation (books, application notes, articles, manuals)

**QP has been used by hundreds of companies worldwide**

- Markets: consumer, industrial, communication, medical, defense

**QP is supported by the free QM graphical modeling tool**