



Quantum Leaps

Modern Embedded Software

Modern Embedded Software Overview of QP™ Real-Time Frameworks and QM™ Modeling Tool



Quantum Leaps
Modern Embedded Software

www.state-machine.com

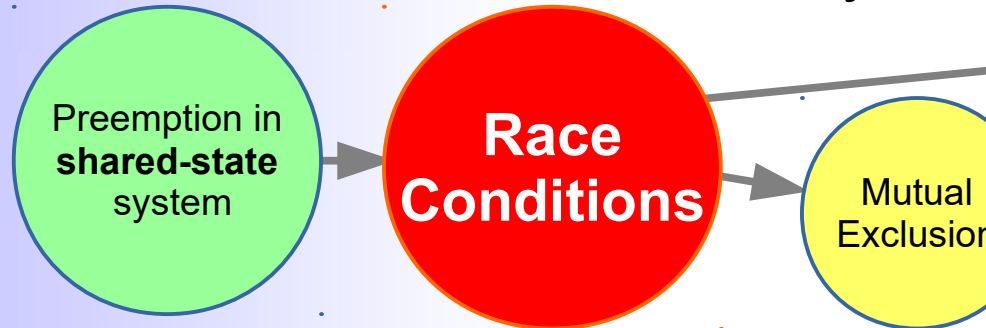


Presentation Outline

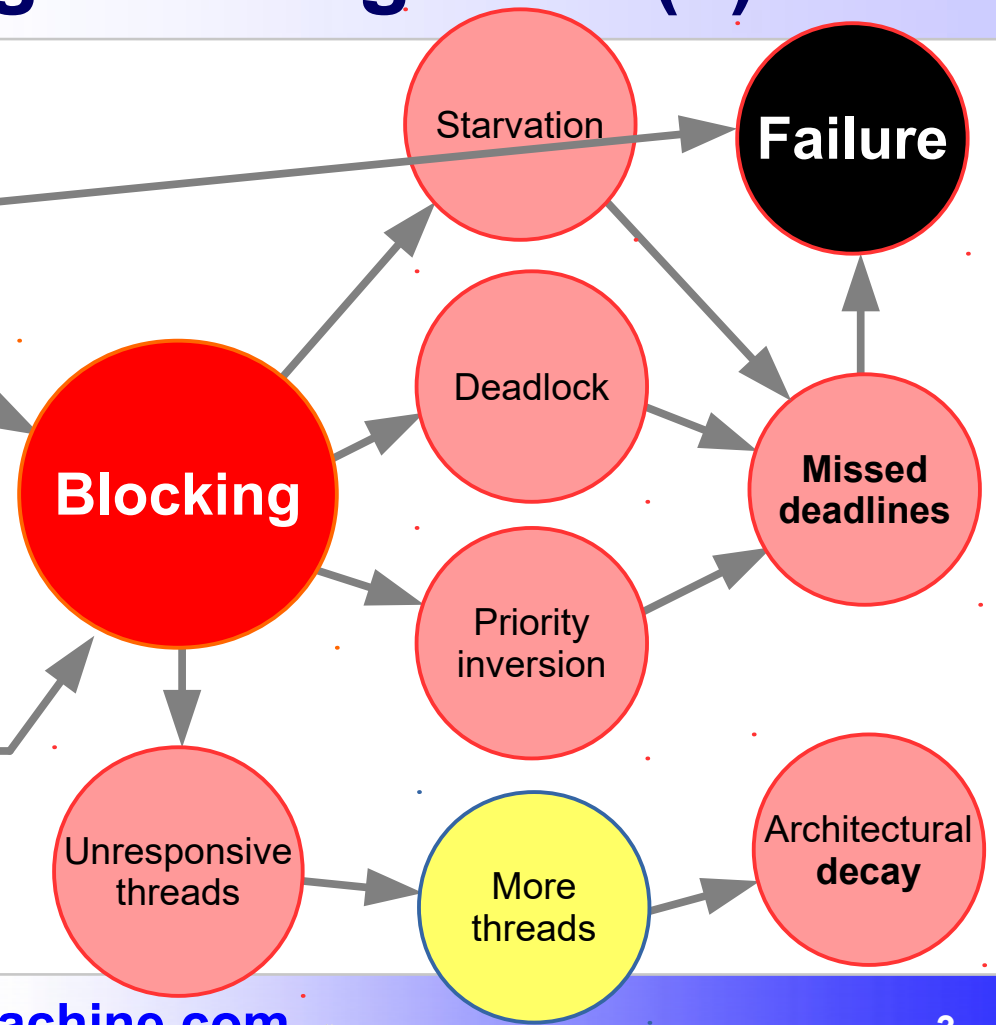
- Why is RTE programming so hard and what can we do about it?
- QP™ real-time frameworks for embedded systems
- QM™ graphical modeling and code generating tool

Why is real-time programming hard (1)?

#1: Shared-state concurrency



#2: Synchronization by blocking



What can we do about it?

Experienced developers came up with **best practices***:

- **Don't share** data or resources (e.g. peripherals) among threads
 - Keep data isolated and bound to threads (strict **encapsulation**)
- **Don't block** inside your code
 - Communicate among threads **asynchronously** via event objects
- Threads should spend their lifetime responding to **events** so their main line should consist of “message pump”
 - Encapsulated thread + “message pump” → **Active Object (Actor)**

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

Active Object (Actor) Design Pattern

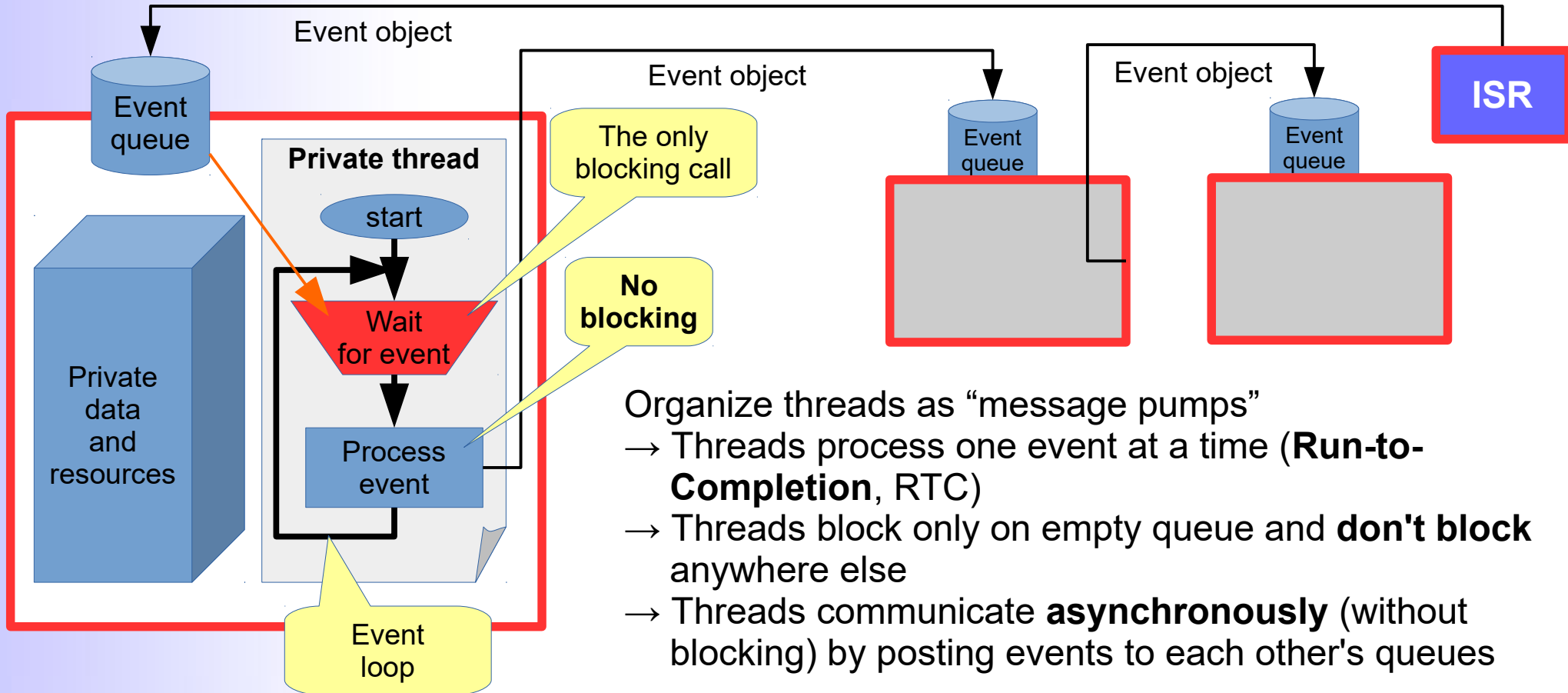
- **Active Object* (Actor*)** is an event-driven, **strictly encapsulated** software object running in its **own thread** and communicating **asynchronously** by means of **events**.
 - Not a real novelty. The concept known from 1970s, adapted to real-time in 1990s (ROOM actor), and from there into the UML (active class).
- The UML specification further proposes the UML variant of **hierarchical state machines** (UML statecharts) with which to model the *behavior* of event-driven active objects (active classes)*.
 - This addresses the “spaghetti code” problem (more about it later)

(*) Lavender, R. Greg; Schmidt, Douglas C. "Active Object"

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

(*) OMG Unified Modeling Language TM (OMG UML) Superstructure, formal/2011-08-06

Active Object pattern with conventional RTOS



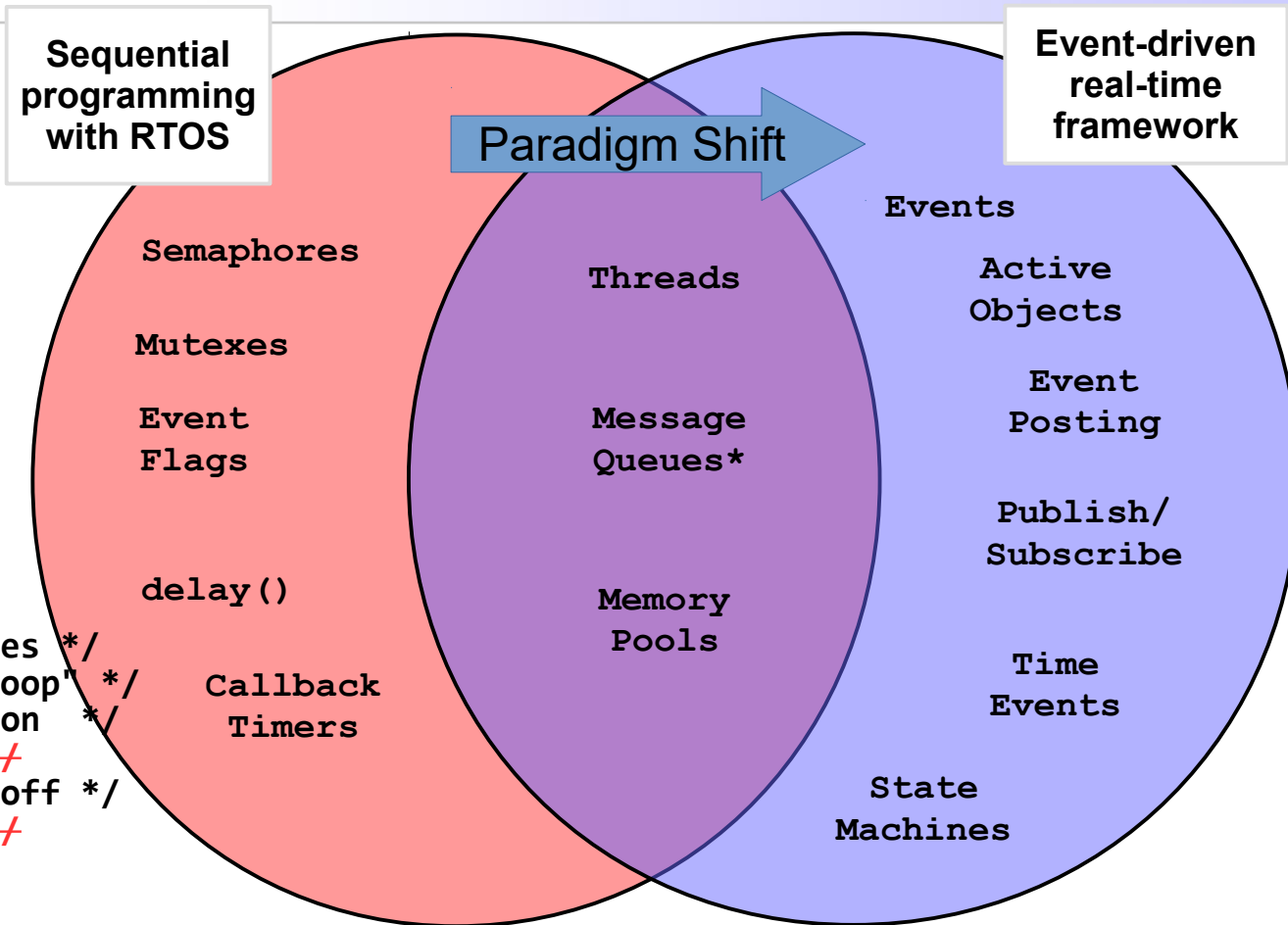
A Better Way: Real-Time Framework

- Implement the Active Object design pattern as a **framework**
 - The best way to capture an **architecture** and make it **reusable**
 - Raises the *level of abstraction* (directly linked to productivity)
- **Inversion of control**
 - The main difference between a framework and a toolkit (e.g., RTOS)
 - The main way to *automate* and *enforce* the best practices (**safer** design)
 - The main way to hide the difficult aspects from application (**safer** design)
 - The main way to bring *conceptual integrity* to the application
 - The main way to bring *consistency* among applications (product lines)

Paradigm Shift: Sequential → Event-Driven

- No blocking
 - Most RTOS mechanisms!
- No sharing
 - Use events with parameters instead
- No sequential code

```
/* this "Blinky" code no longer flies */  
while (1) { /* RTOS task or "superloop" */  
    BSP_ledOn(); /* turn the LED on */  
    OS_delay(500); /* blocking!!! */  
    BSP_ledOff(); /* turn the LED off */  
    OS_delay(500); /* blocking!!! */  
}
```



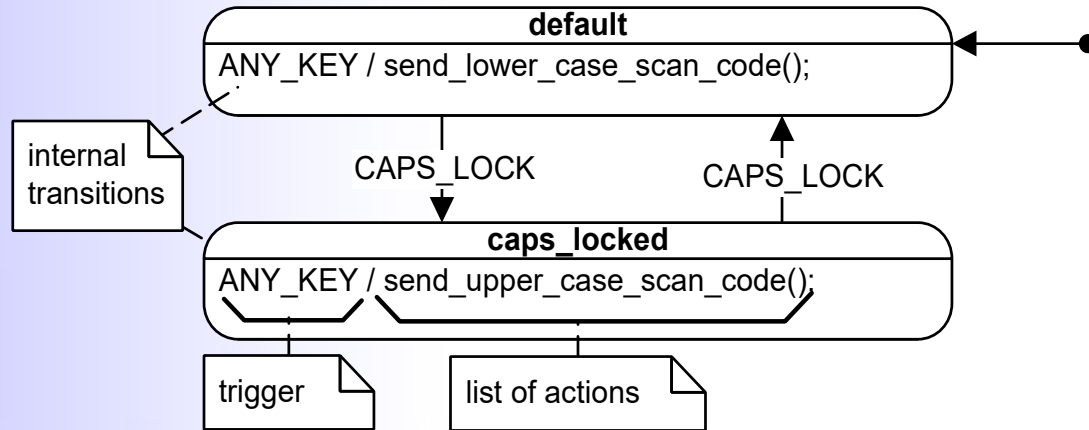
Why is event-driven programming hard (2)?

- Responding to events leads to “spaghetti code”
 - The response depends on both: the event type and the **internal state** of the system
 - State of the system (history) is represented *ad hoc* as multitude of flags and variables
 - Convoluted, deeply nested IF-THEN-ELSE-SWITCH logic based on complex expressions
 - **spaghetti code**

```
1 void operator_click(int Index) {
2   char TempReadout[READOUT_LEN];
3   strcpy(TempReadout, Readout);
4   if (strcmp>LastInput, "NUMS") == 0) {
5     NumOps++;
6   }
7   switch (NumOps) {
8     case 0:
9       if ((Operator[Index].Caption[0] == '-')
10          && (strcmp>LastInput, "NEG") == 0)
11         {
12           strcpy(&Readout[1], &Readout[0]);
13           Readout[0] = '-';
14           LastInput = "NEG";
15         }
16         break;
17     case 1:
18       Opt1 = Readout;
19       if ((Operator[Index].Caption[0] == '-')
20          && (strcmp>LastInput, "NUMS") != 0)
21          && (OpFlag != '='))
22         {
23           Readout[0] = '-';
24           LastInput = "NEG";
25         }
26         break;
27     case 2:
28       Op2 = TempReadout;
29       switch (OpFlag) {
30         case '+':
31           Op1 = CDb1(Op1) + CDb1(Op2);
32           break;
33         case '-':
34           Op1 = CDb1(Op1) - CDb1(Op2);
35           break;
36         case '*':
37           Op1 = CDb1(Op1) * CDb1(Op2);
38           break;
39         case '/':
40           if (Op2 == 0) {
41             Error("Division by zero");
42           }
43           Op1 = CDb1(Op1) / CDb1(Op2);
44           break;
45       }
46       Readout = Op1;
47       NumOps = 1;
48       break;
49   }
50   if (strcmp>LastInput, "NEG") == 0) {
51     strcpy>LastInput, "OPS";
52     OpFlag = Operator[Index].Caption;
53   }
54 }
```

What can we do about it?

- Finite State Machines—the best known “spaghetti reducers”
 - “State” captures only the relevant aspects of the system's history
 - Natural fit for event-driven programming, where the code cannot block and must return to the event-loop after each event)
 - Context stored in a single state-variable instead of the whole call stack

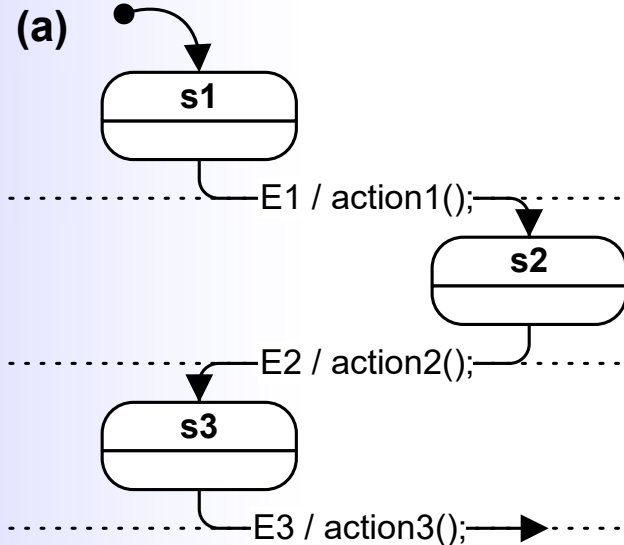


Paradigm Shift: Sequential → Event-Driven (2)

State Machines are **not** Flowcharts (!)

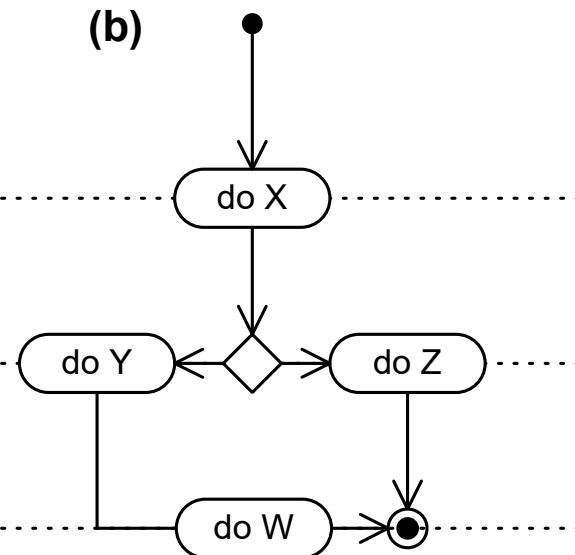
Statechart (event-driven)

- represents all states of a system
- driven by explicit **events**
- processing happens on arcs (transitions)
- no notion of “progression”



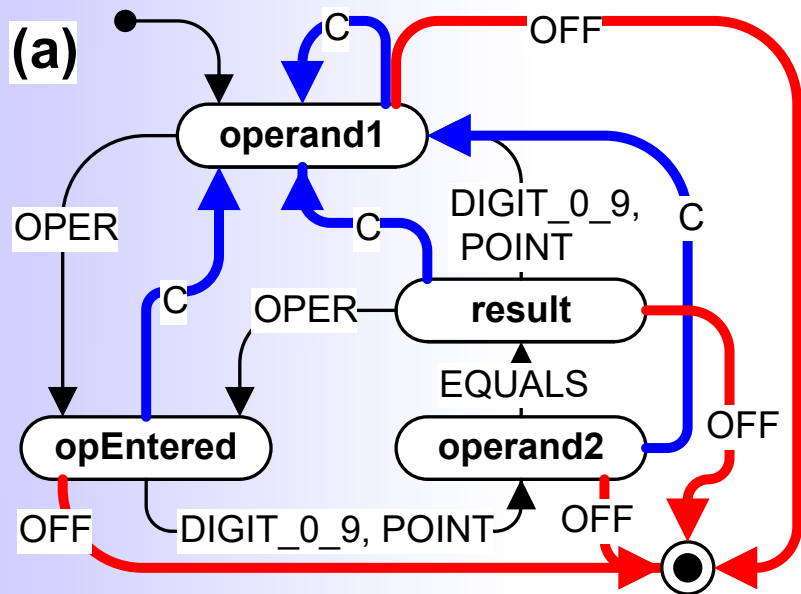
Flowchart (sequential)

- represents stages of processing in a system
- gets from node to node upon completion
- processing happens in nodes
- progresses from start to finish

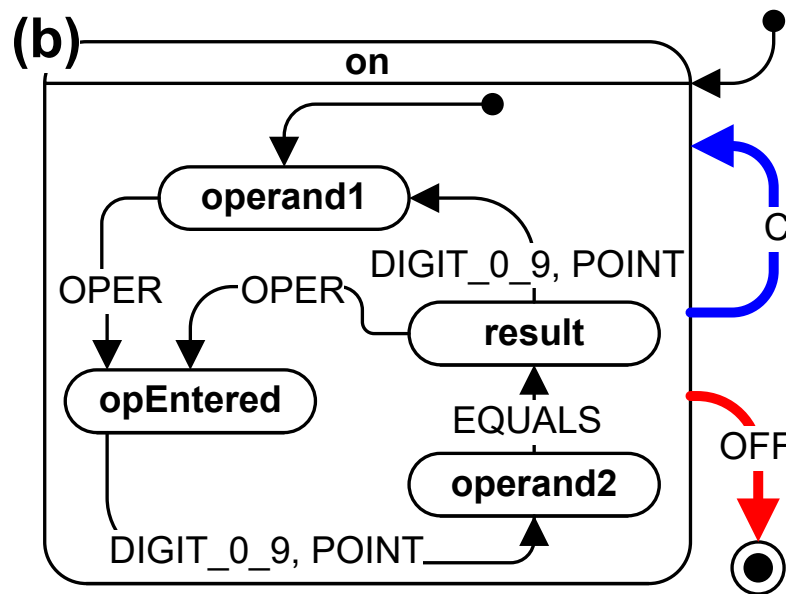


Hierarchical State Machines

Traditional FSMs “explode”
due to **repetitions**



State hierarchy eliminates repetitions
→ programming-by-difference



Presentation Outline

- Why is RTE programming so hard and what can we do about it?
- QP™ real-time frameworks for embedded systems
- QM™ graphical modeling and code generating tool

QP™ Real-Time Frameworks



- Family of frameworks for deeply embedded real-time systems: QP/C, QP/C++, QP-nano
 - Combines Active Object pattern with Hierarchical State Machines, which beautifully complement each other
 - Many advanced features yet lightweight (smaller than RTOS kernel)
- Good fit for systems with **functional safety** requirements
 - Sound, component-based **architecture** *safer* than “naked” RTOS
 - Provides means of designing applications based on **state machines** and **documented** as UML state diagrams (recommended by safety standards)
 - **Traceable** implementation in MISRA-compliant C or C++

Who is using QP™?

professional



open source

QP™ has been licensed by companies large and small in diverse industries.

- Consumer electronics
- Medical devices
- Defense
- Industrial controls
- Communication & IoT
- Robotics
- Semiconductor IP
- ... (see online)

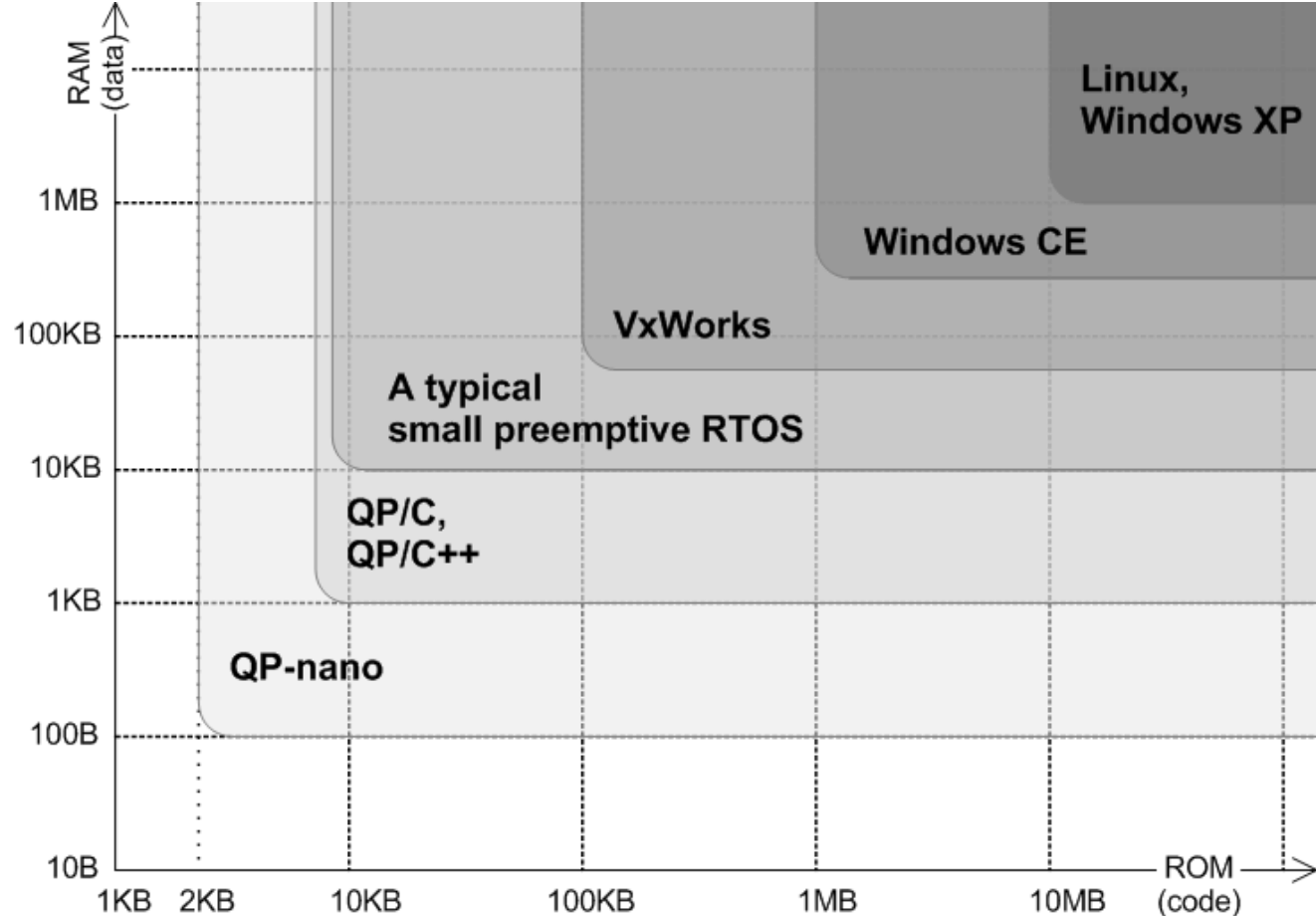
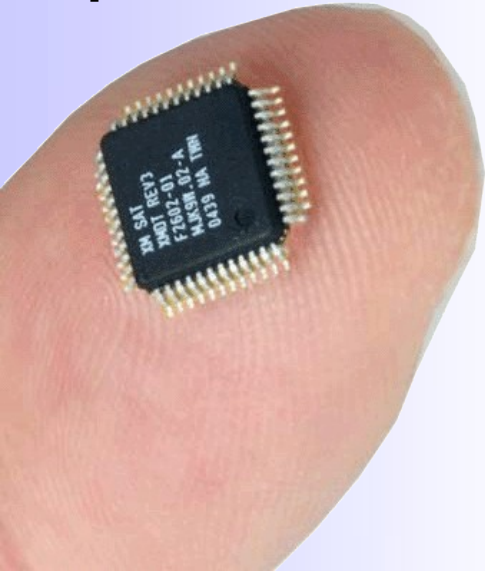
ABB ABB Switzerland	AC PROPULSION AC Propulsion	Acuity Brands Acuity Brands	Alti Alti Digital Monitoring	Amway Amway	dynamic Dynamic Controls (repeat customer)	EATON Eaton	Electrolux Electrolux Sweden	EMERSON Emerson Network Power	EMERSON Emerson Process Management	SIEMENS Siemens Schweiz (repeat customer)	TRW TRW Poland
AMX AMX (repeat customer)	ANALOG DEVICES Analog Devices	APC American Power Conversion	ARRIS ARRIS (Motorola Mobility)	Artesis Artesis Turkey	G-Tech G-Tech New Zealand	GATSO GATSOmeter (repeat customer)	GE GE Consumer & Industrial	GENERAL DYNAMICS General Dynamics	HACH Hach (multiple divisions)	SunTech Medical SunTech Medical	WHOOOP Whoop
BIO-RAD Bio-Rad	BOSCH Bosch	BRAND HYDRAULICS Brand Hydraulics	CAMERON Cameron	Cardinal Health Cardinal Health	HAEMONETICS Haemonetics	hansen Hansen Medical	HEX HEX Microsystems	HITACHI Hitachi	Honeywell Honeywell (multiple divisions)	TOMRA Tomra	Starkey Starkey Laboratories
CATERPILLAR Caterpillar Trimble	Corindus Corindus Vascular Robotics	CORNING Corning	CUBIC Cubic Defense	CUBIC Cubic Transportation	ホジザキ電機 Hoshizaki Group	HsinPlas Hsin-Chin Machinery Co.	IBM IBM (Lenovo)	Imagination Imagination Technologies	INFIMED Infimed	VERATHON Verathon	tellabs Tellabs (multiple divisions)
CYBERCOM GROUP Cybercom Group	DDD DDD Diagnostic	digitalSTROM DigitalSTROM	DRS DRS Tactical Systems	DMG MORI DMG MORI	INSITU Insitu (repeat customer)	intel Intel	iRhythm iRhythm	JBL JBL	JOHN DEERE John Deere Forestry	ST. JUDE MEDICAL St. Jude Medical	TYMPHANY Tymphany
Kodak alaris Kodak-Alaris (repeat customer)	L3 communications L3 MAS	LINK Link	Land Transport & Technology LTA Singapore	lytx Lytx	sonova Phonak Sonova	PHYSIO CONTROL Physio Control	proteus Proteus	QinetiQ QinetiQ (for NASA)	Qwizdom Qwizdom	TATA TATA Technologies	wirelessSEISMIC Wireless Seismic
Wantowoc Wantowoc	Medtronic Medtronic (various divisions)	METTLER TOLEDO Mettler Toledo	SMSC Microchip SMSC	micronics Micronics	Raytheon Anschutz Raytheon-Anschutz	Resodyn Resodyn	RINGLY Ringly	rittmeier Rittmeyer	Roche Roche	TOSHIBA Toshiba (repeat customer)	STRATOS Stratos
MOOG Moog Medical	MOOG FERNAU Moog Fernau (repeat customer)	MOTOROLA SOLUTIONS Motorola Solutions	MOTOROLA Motorola (various divisions)	cerHi Certi	Rockwell Collins Rockwell Collins	RÖHDE & SCHWARZ Rohde Schwarz	SAIC Saic	Sandia National Laboratories Sandia National Laboratories	Schneider Electric Schneider Electric	VISURAY Visuray	tm4 TM4 Electrodynamic Systems
NANO MR NanoMR (DNA Electronics)	NASA NASA	NETGEAR Netgear	nielsen Nielsen (Abtizon)	NOV NOV	SEA SEA Schless-Systeme	SECOM Secom Japan	SECURITON Securiton	Sound Sound Environmental Products	STEP STEP Electric Shanghai	STANLEY Stanley Healthcare	SAFRAN SAFRAN Vectronix
Naval Research Laboratory Naval Research Laboratory	NTREPID Ntrepid	OCEANEERING Oceaneering	PATTON Patton Inrap	PHILIPS Philips Healthcare (repeat customer)	SIEMENS Siemens Schweiz (repeat customer)	ST. JUDE MEDICAL St. Jude Medical	STANLEY Stanley Healthcare	Starkey Starkey Laboratories	STRATOS Stratos	TeleCommunication Systems TeleCommunication Systems	Whirlpool Whirlpool Sweden

QP™ Framework Family Features

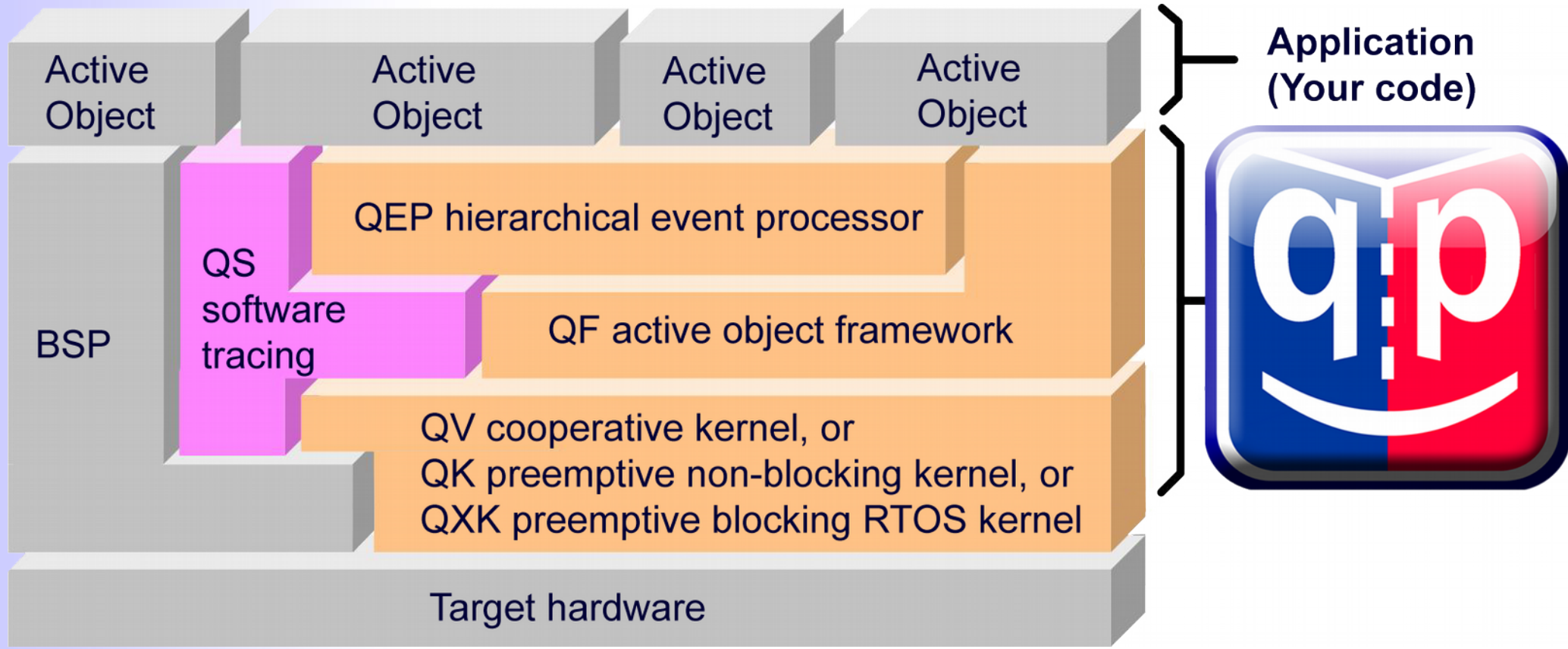
Feature	QP/C	QP/C++	QP-nano
Code (ROM) / Data (RAM) footprint	4KB / 1KB	5KB / 1KB	2KB / 0.5KB
Maximum number of active objects	64	64	8
Hierarchical state machines	✓	✓	✓
Events with arbitrary parameters	✓	✓	32-bits
Event pools and automatic event recycling	✓	✓	✗
Direct event posting	✓	✓	✓
Publish-Subscribe	✓	✓	✗
Event deferral	✓	✓	✗
Number of time events per active object	unlimited	unlimited	1
Software tracing support (Q-SPY)	✓	✓	✗
Cooperative QV kernel	✓	✓	✓
Preemptive, non-blocking QK kernel	✓	✓	✓
Preemptive, blocking kernel (QXK)	✓	✓	✗
Portable to 3 rd -party RTOS	✓	✓	✗

QP™ vs. RTOS Memory Footprint

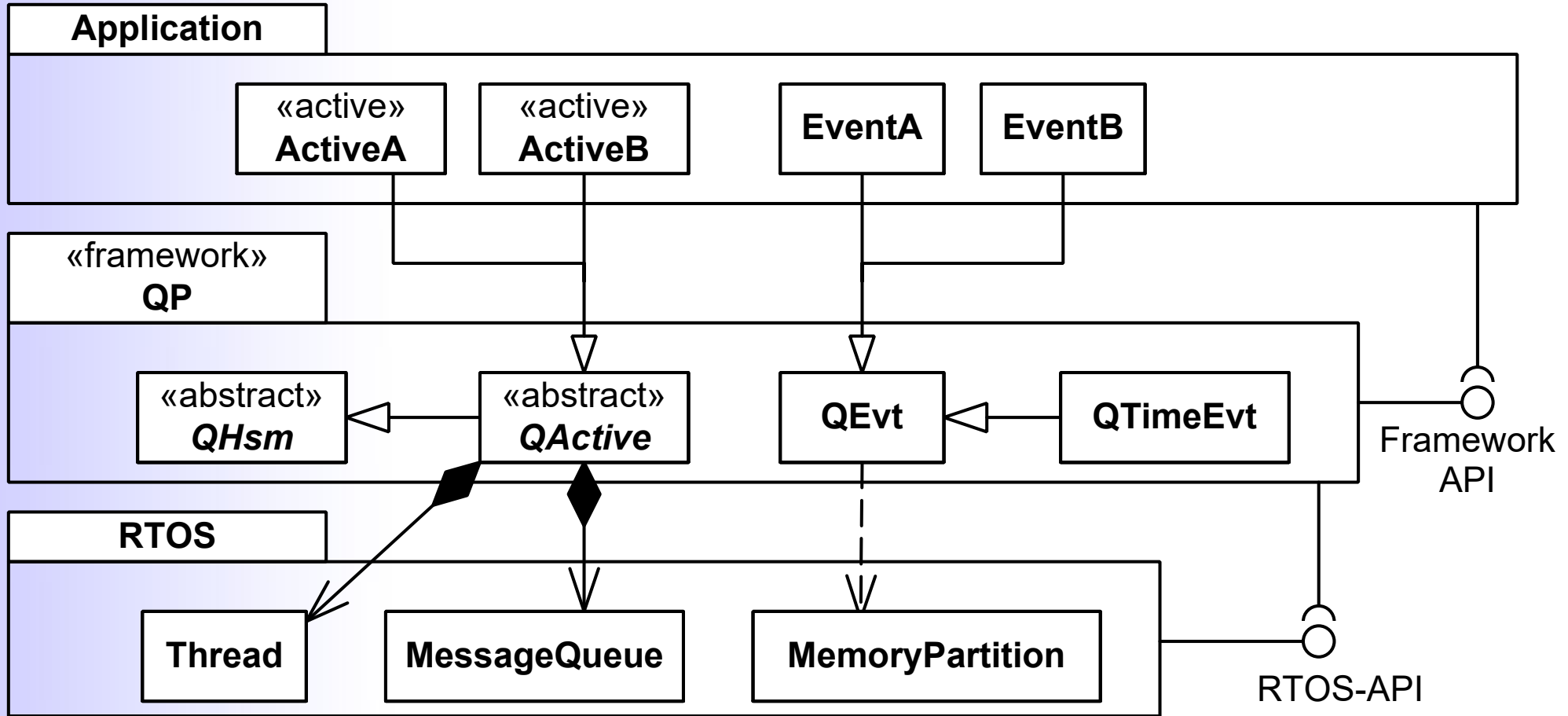
QP frameworks fit into smaller RAM, because event-driven programming style uses much less **stack space**



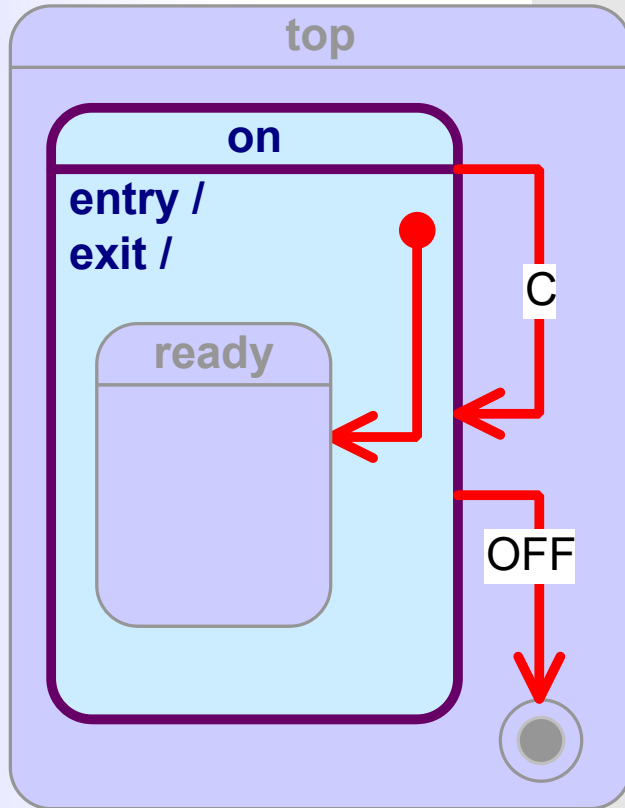
QP™ Sub-Components



QP™ Package and Class View

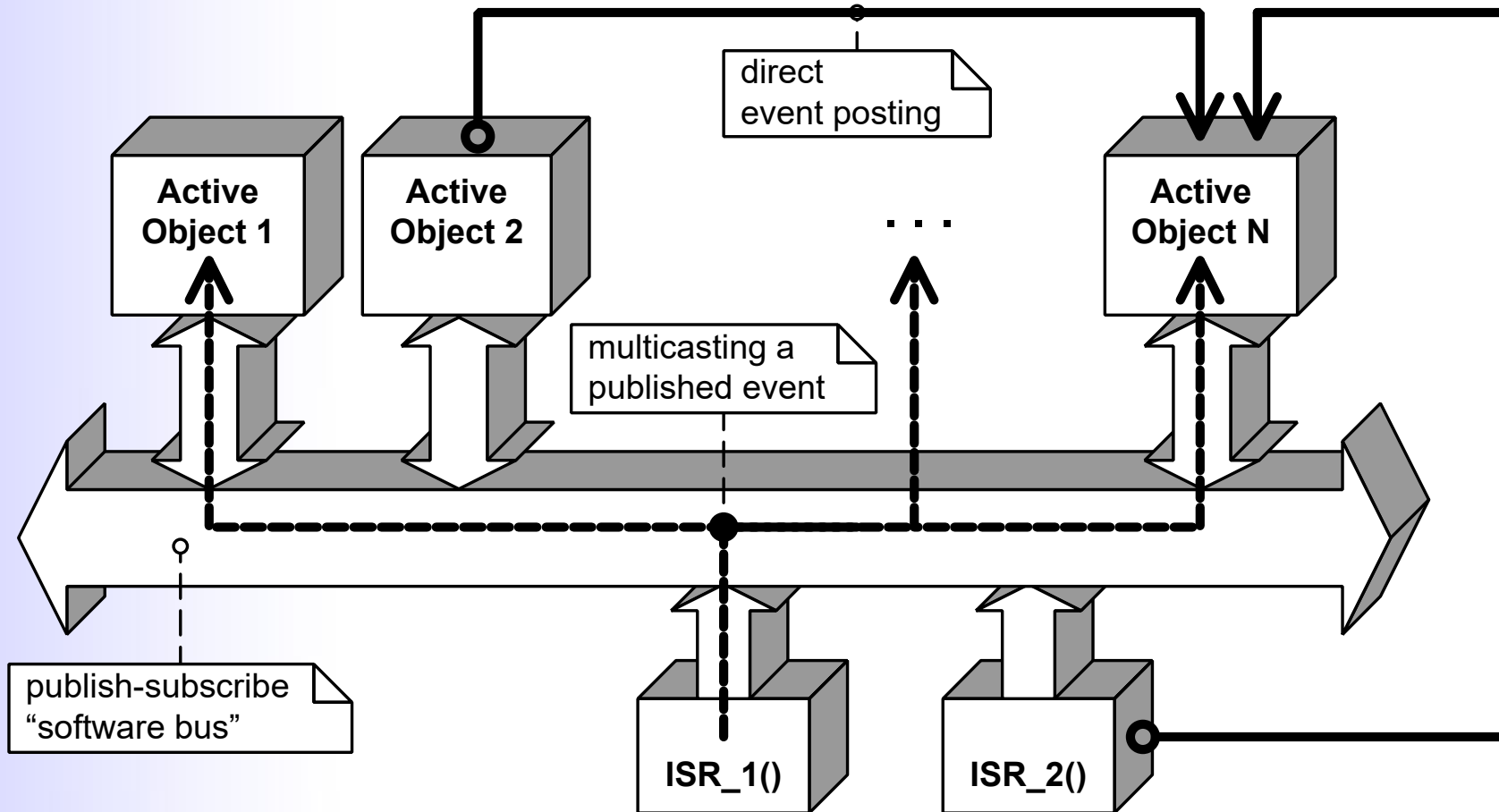


QEP Hierarchical Event Processor

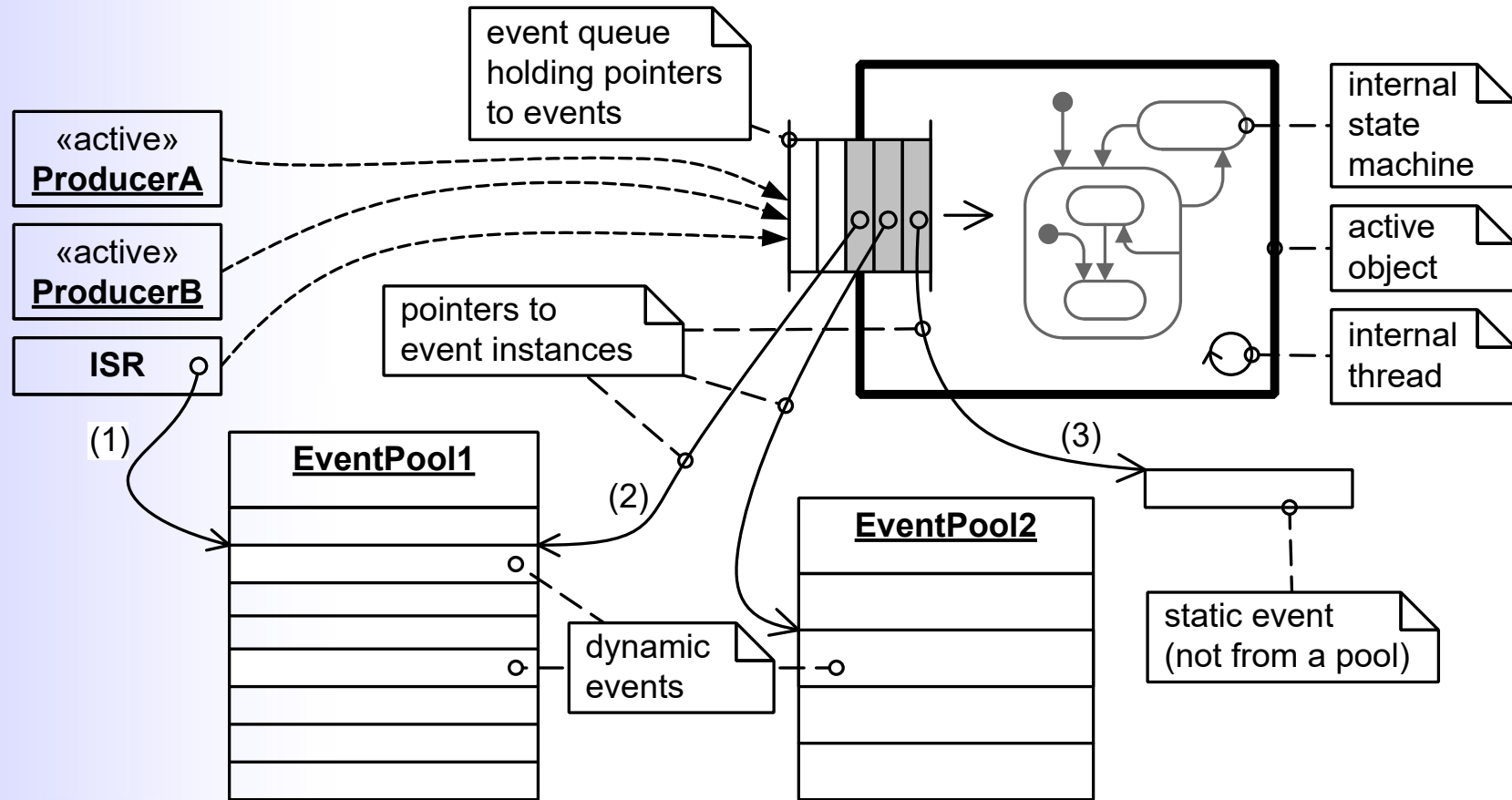


```
QState Calc_on(Calc * const me, QEvt const *e) {
    QState status;
    switch (e->sig) {
        case Q_ENTRY_SIG:    /* entry action */
            BSP_message("on-ENTRY");
            status = Q_HANDLED();
            break;
        case Q_EXIT_SIG:    /* exit action */
            BSP_message("on-EXIT");
            status = Q_HANDLED();
            break;
        case Q_INIT_SIG:    /* initial transition */
            BSP_message("on-INIT");
            status = Q_TRAN(&Calc_ready);
            break;
        case C_SIG:         /* state transition */
            BSP_clear();    /* clear the display */
            status = Q_TRAN(&Calc_on);
            break;
        case OFF_SIG:      /* state transition */
            status = Q_TRAN(&Calc_final);
            break;
        default:
            status = Q_SUPER(&QHsm_top); /* superstate */
            break;
    }
    return status;
}
```

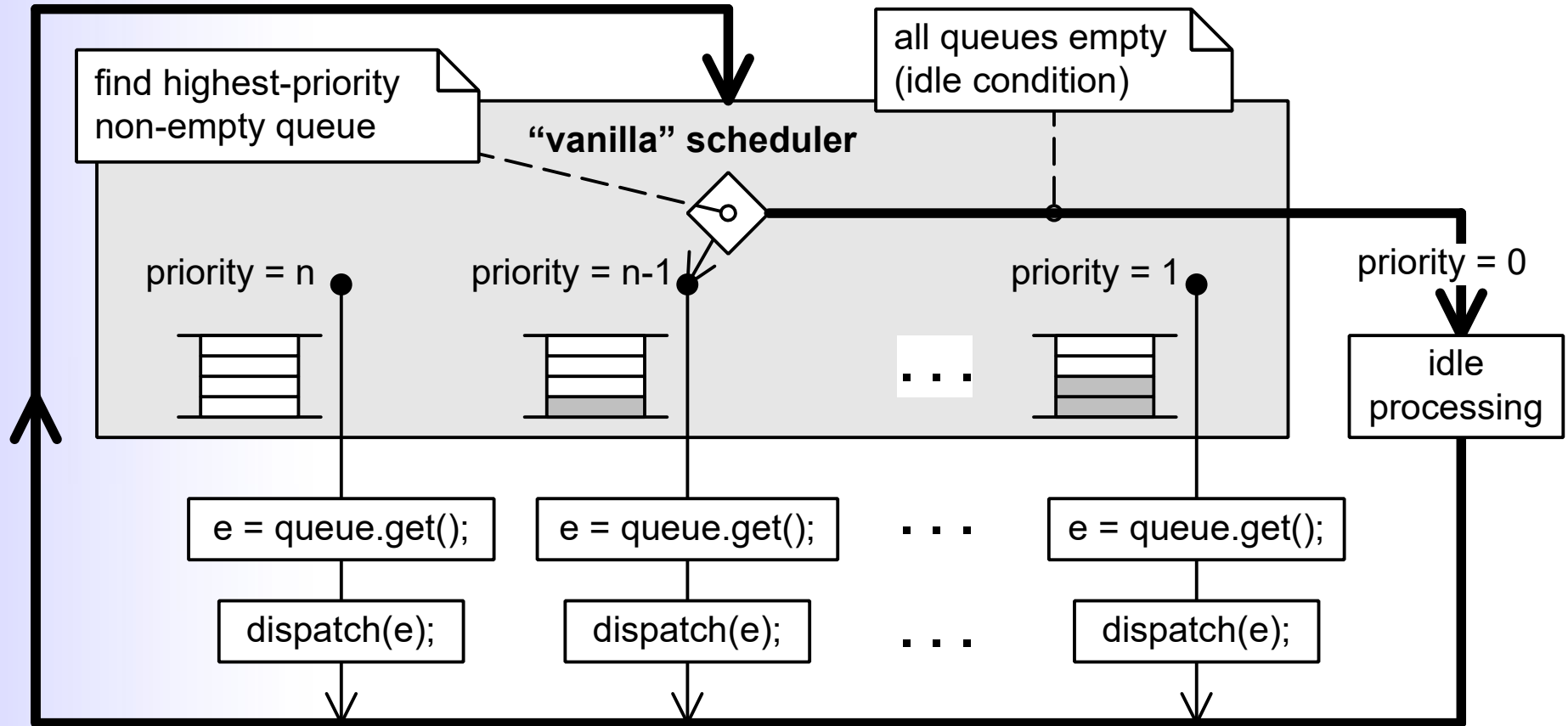
QF Framework – “Software Bus”



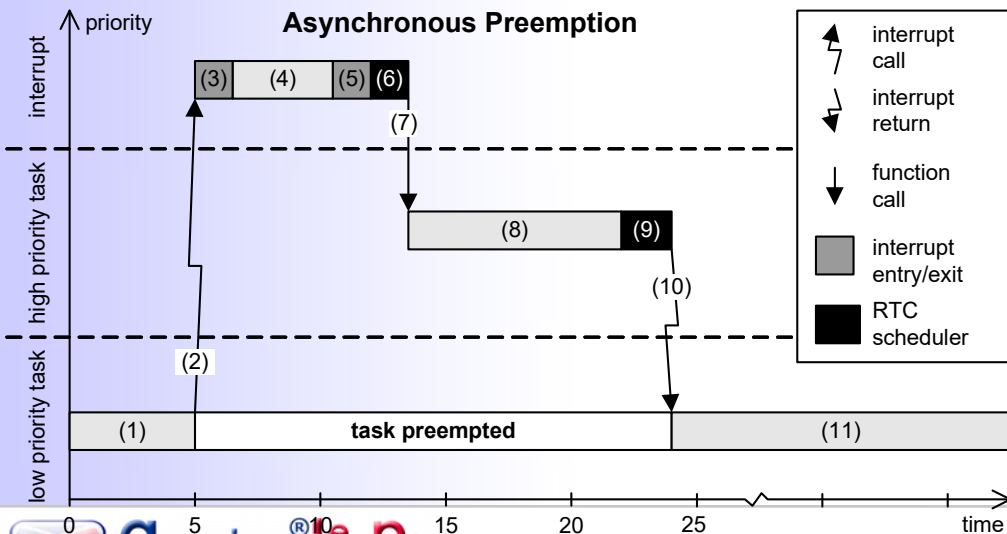
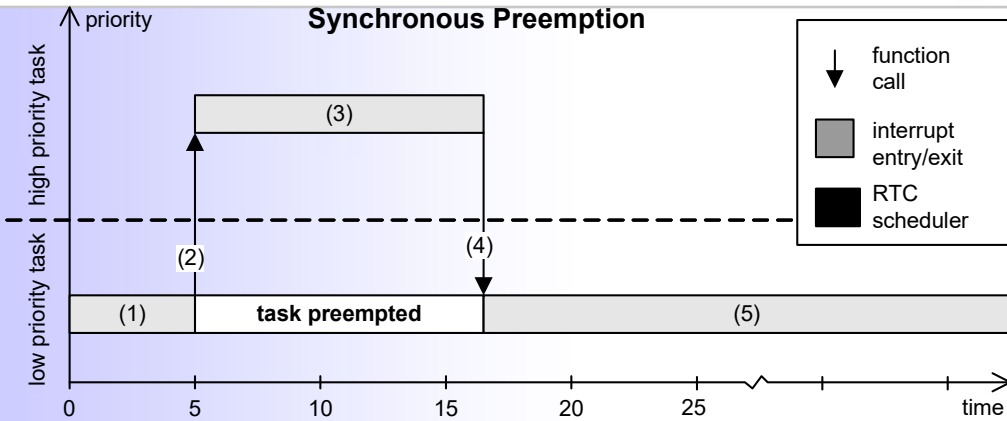
QF Framework – “Zero Copy” Event Delivery



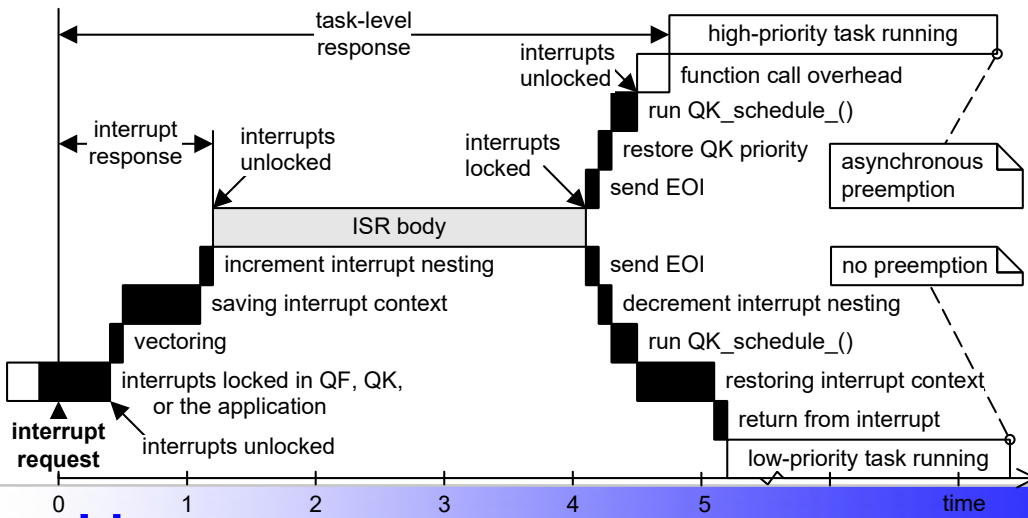
QV™ Cooperative Kernel



QK™ Preemptive, Non-Blocking Kernel

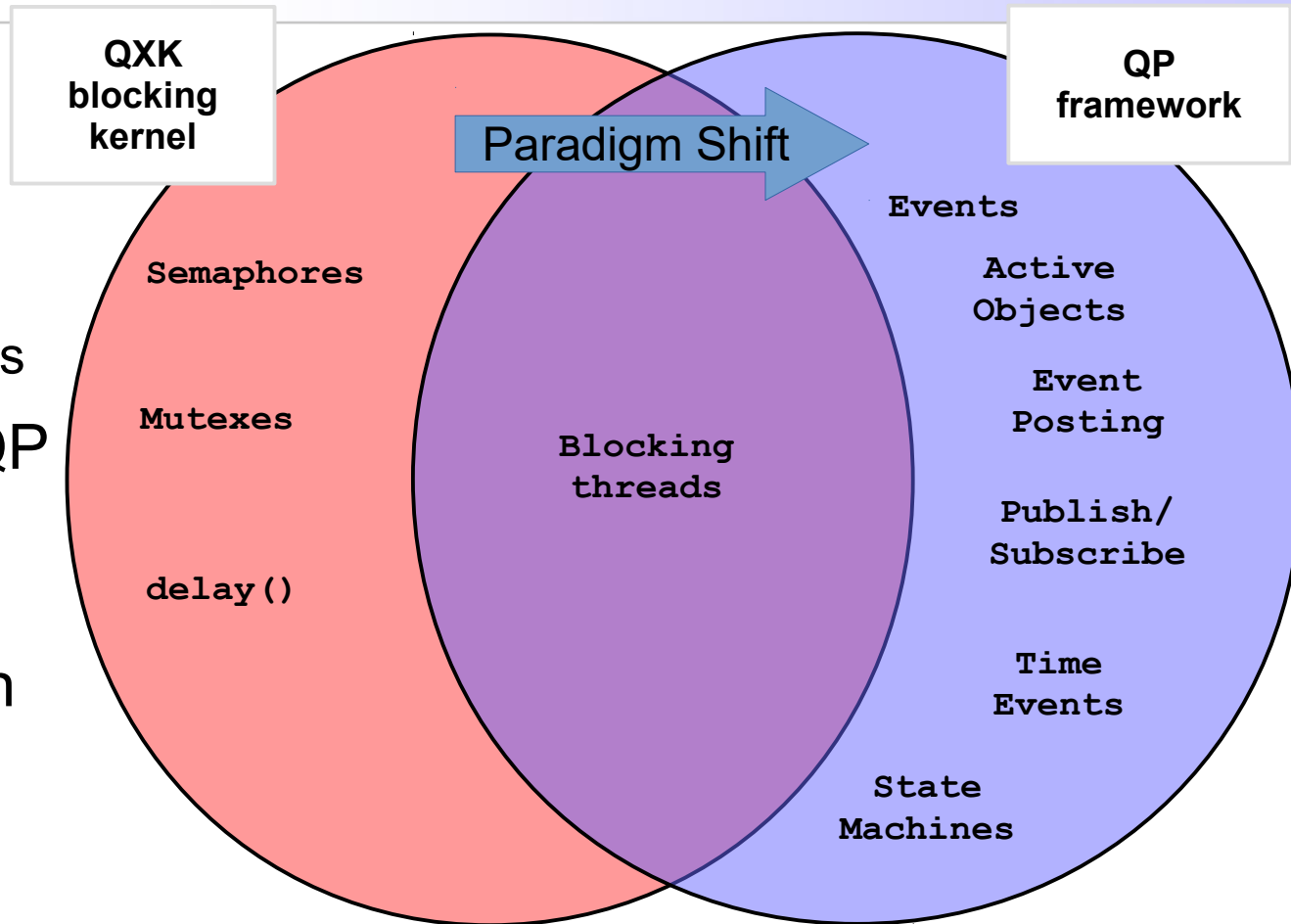


- Preemptive priority-based kernel
 - Meets all requirements of Rate Monotonic Analysis (RMA)
 - Run-to-Completion Kernel
- **Cannot block in-line**
- **Single stack operation (like ISRs)**



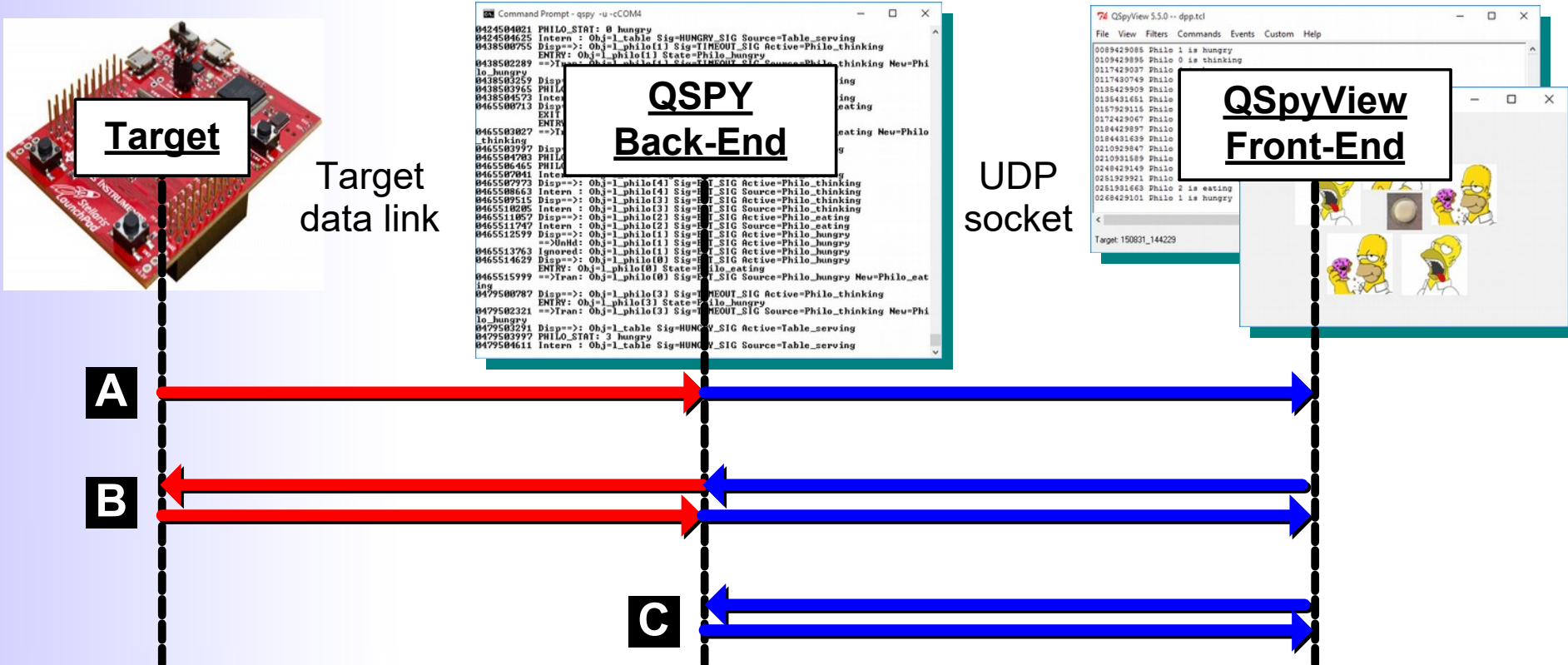
QXK™ Preemptive, Blocking Kernel

- A “bridge” to legacy software & middleware in sequential paradigm → Sequential threads can coexist with event-driven AOs
- Tightly integrated with QP (reuse of event queues, time events, etc.)
- More efficient way to run QP apps than any 3rd-party RTOS.



QS/QSPY™ Software Tracing System

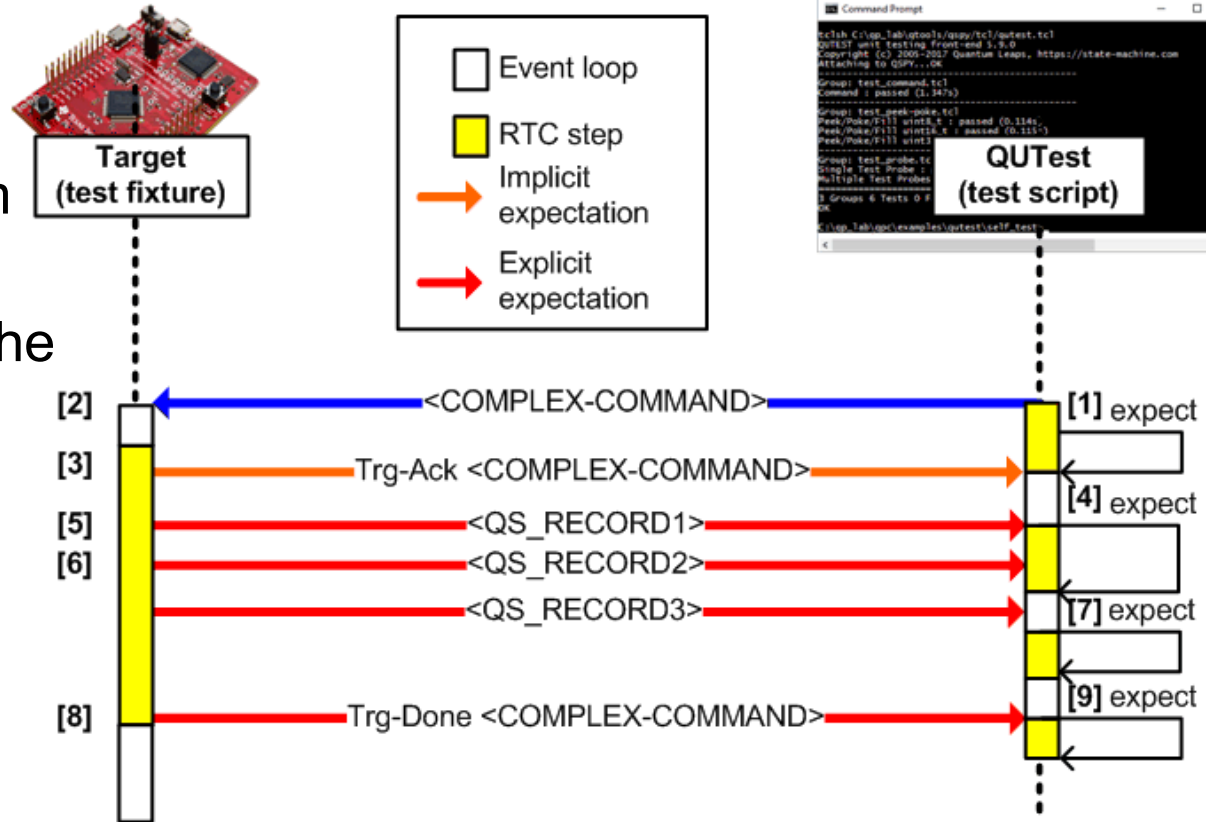
- You need to observe system **live**, not stopped in a debugger



QUTest™ Unit Testing Harness

Specifically designed for **TDD** of deeply embedded software

- Separates CUT execution from checking the test assertions
- Small, reusable test fixture in the Target (C or C++ code)
- Driving the tests and checking correctness on the Host
- Python and Tcl test scripting
- Specifically suitable for **event-driven** systems (simplifies “mocking”)



QSpyView™ Front-End

- Customizable (scripted) Front-End for **monitoring** and **control** of embedded Targets
 - Remote User Interface
 - Graphic display of Target status
 - Dynamic interaction with Target
 - Remote resetting the Target

The screenshot displays the QSpyView GUI Front-End. On the left, a 'Command window running QSPY' shows a terminal window with the following output:

```
cmd: Command Prompt - qspy -u -cCOM4
2981504673 PHILO_STAT: 0 thinking
2981506441 PHILO_STAT: 1 eating
2981507
2981507 7% QSpyView 5.5.0 -- dpp.tcl
2981509 File View Filters Commands Ev
2981509
2981510 2799006463 Philo 4 is eating
2981511 2808504013 Philo 0 is hungry
2981512 2822504769 Philo 1 is thinkin
2981513 2822506537 Philo 2 is eating
2981514 2847003979 Philo 3 is hungry
2981514 2870003887 Philo 1 is hungry
2981515 2896004719 Philo 4 is thinkin
2981516 2896006487 Philo 0 is eating
2981516 2899004713 Philo 2 is thinking
2996500 2899006481 Philo 3 is eating
2996500 2935503963 Philo 4 is hungry
2996503 2968503989 Philo 2 is hungry
2996504 2981504673 Philo 0 is thinking
2996504 2981506441 Philo 1 is eating
2996504 2996504739 Philo 3 is thinking
2996506 2996506507 Philo 4 is eating
2996507
2996508
2996509
2996510 Target: 150831_144229 Tx [6] Rx [1] Err [0] Rx [2188] Err [0] [x] scroll
2996511 2996511907 Intern : Obj=1_philo[0] Sig=EAT_SIG Source=Philo_thinking
2996511909 Disp==>: Obj=1_philo[2] Sig=EAT_SIG Active=Philo_hungry
2996511910 ==>UnHd: Obj=1_philo[2] Sig=EAT_SIG Active=Philo_hungry
2996513073 Ignored: Obj=1_philo[2] Sig=EAT_SIG Active=Philo_hungry
2996513925 Disp==>: Obj=1_philo[1] Sig=EAT_SIG Active=Philo_eating
2996514615 Intern : Obj=1_philo[1] Sig=EAT_SIG Source=Philo_eating
2996515481 Disp==>: Obj=1_philo[0] Sig=EAT_SIG Active=Philo_thinking
2996516171 Intern : Obj=1_philo[0] Sig=EAT_SIG Source=Philo_thinking
```

On the right, a 'Canvas' window displays a graphical representation of the target application. It features five Homer Simpson characters in various states: 'thinking Philosopher', 'eating Philosopher', 'hungry Philosopher', and a 'button to pause/resume granting the forks'. A callout box indicates that the 'Canvas window of QSpyView customized for the DPP application' is communicating with QSPY via a UDP socket.

Design by Contract (DbC)

- The QP's error-handling policy is based on DbC
- Preconditions / Postconditions / Invariants / General Assertions
 - DbC built-into the framework
 - Designed to catch problems in the *application*
 - No way of ignoring errors (enforcement of rules)
 - Provides redundancy and self-monitoring for safety-critical applications
- Example QP policies enforced by DbC
 - Event delivery guarantee (event pools and queues can't overflow)
 - Arming / disarming / re-arming of time events
 - System initialization, starting active objects

Presentation Outline

- Why is RTE programming so hard and what can we do about it?
- QP™ real-time frameworks for embedded systems
- QM™ graphical modeling and code generating tool

QM™ Modeling Tool

- Modeling and code-generation tool for QP™ frameworks
 - Adds graphical state machine modeling to QP™
 - QP™ frameworks provide an excellent target for automatic code generation



QM™ Design Philosophy

- “Low ceremony”, code centric tool (no PIM, PSM, action-languages,...)
→ Not appropriate if you need these features (80% of benefits for 20% of costs)
- Optimized for C and C++, (no attempts to support other languages)
- Optimized for QP™ (no attempts to support other frameworks)
- Forward-engineering only (no attempts at “round-trip engineering”)
- Capture *logical design* (packages, classes, state machines)
- Capture *physical design* (directories and files generated on disk)
- Minimize “*fighting the tool*” while drawing diagrams and generating code
- Capable of invoking external tools, such as compilers, flash-downloaders...
- **Freeware**



Logical Design (Packages/Classes/Statecharts)

The screenshot displays the QM 3.3.0 software interface. The main window shows a statechart for 'SM of QMsmTst' with states s, s1, s11, s2, s21, and s211. Transitions are labeled with events and actions like 'e / x /', '[me->m_foo] / me->m_foo = 0U;', 'A /', 'C /', 'G /', 'F /', 'B /', 'D /', 'H /', and 'E /'. A 'TERMINATE' action is also present.

The left sidebar shows the Model Explorer with a tree structure: SMs > QMsmTst: QMsm > m_foo: bool > QMsmTst > SM > ->s2 > s > ->s11 > I > [me->m_foo] > E > TERMINATE > s1 > ->s11 > I > D > A > B > F > C.

The bottom-left 'Bird's Eye View' shows a smaller version of the statechart.

The bottom-center code editor shows the following C++ code:

```
namespace QMSMTST {  
enum QMsmTstSignals {  
    A_SIG = QP::Q_USER_SIG,  
    B_SIG,  
    C_SIG.  
};  
static QMsmTst l_msmst; // the only instance of th  
// global-scope definitions -----  
QP::QMsm * const the_msm = &l_msmst; // the opaque  
#define(SMs::QMsmTst)
```

The bottom-right 'Property Editor' shows the configuration for state s2:

- State: State
- name: s2
- superstate: s
- documentation: [empty]
- entry: [empty]
- BSP_display("s2-ENTRY;");
- exit: [empty]
- BSP_display("s2-EXIT;");

The bottom Log Console shows the following output:

```
INFO> Code generation started (04:07:17.976 pm)  
INFO> Entire model: C:\qp\qpcpp\test\win32\qmsmtst\qmsmtst.qm  
INFO> Code generation ended (time elapsed 0.000s)  
INFO> 0 file(s) generated, 2 file(s) processed, 0 error(s), and 0 warning(s)
```

Physical Design (Directories / Files)

Model Explorer

- dpp
 - qpc [locked]
 - Events
 - AOs
 - dpp.h
 - philo.c
 - table.c
 - main.c

```
#include "qp"
#include "dpp"
#include "bsp"
Q_DEFINE_THIS_FILE
/* Active object
$declare(AOS)
$define(AOS:Missile_ctor)
$define(AOS:Missile)
/* Local object
static table
/* Global scope
Active = const AO_Missile = (QActive
```

Computer > DATA (D:) >

Organize Open Burn

Name	Type
dbg	File folder
rel	File folder
settings	File folder
spy	File folder
bsp.c	C File
bsp.h	H File
dpp.h	H File
dpp.qm	QM Model File
dpp-qk.dep	DEP File
dpp-qk.ewd	EWD File
dpp-qk.ewp	EWP File
dpp-qk.eww	IAR IDE Workspac
dpp-qk.icf	ICF File
gpio.h	H File
main.c	C File
philo.c	C File
README.txt	TXT File
rom.h	H File
startup_tm4c.c	C File
sysctl.h	H File
table.c	C File

File Generated on Disk

win32\mingw\game-gui\game.qm - [missile.c]

ols Window Help

```
#include "qp_port.h"
#include "bsp.h"
#include "game.h"
Q_DEFINE_THIS_FILE
/* local objects
$declare(AOS:Missile)
static Missile l_missile; /* the sole
Missile instance */
/* Public-scope objects
QActive = const AO_Missile = (QActive
Missile)
/* Active object definition
$define(AOS:Missile_ctor)
$define(AOS:Missile)
```

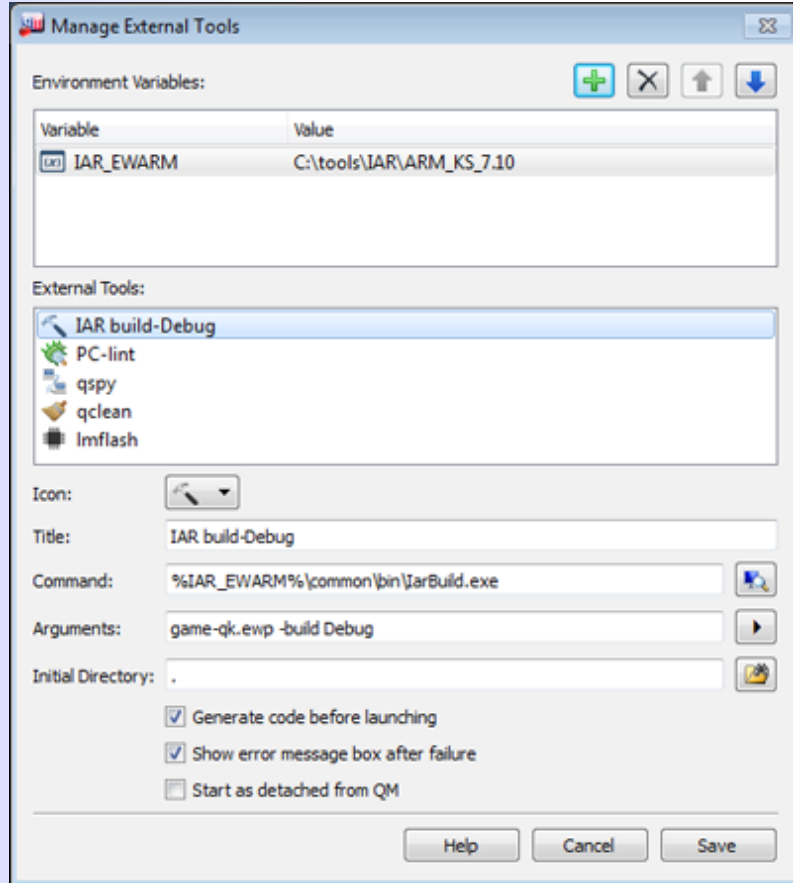
Expanded \$declare() Code-Generation Directive

```
Model: game.qm
File: ../Missile.c
This code has been generated by QM tool (see state-machine.com/qm).
DO NOT EDIT THIS FILE MANUALLY. All your changes will be lost.
This program is open source software: you can redistribute it and/or
modify it under the terms of the GNU General Public License as published
by the Free Software Foundation.
This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
for more details.
.....
#include "qp_port.h"
#include "bsp.h"
#include "game.h"
Q_DEFINE_THIS_FILE
/* local objects
$declare(AOS:Missile)
static Missile l_missile; /* the sole
Missile instance */
/* Public-scope objects
QActive = const AO_Missile = (QActive
Missile)
/* Active object definition
$define(AOS:Missile_ctor)
$define(AOS:Missile)
.....
static QState Missile_initial(Missile * const me, QEvt const * const e);
static QState Missile_armed (Missile * const me, QEvt const * const e);
static QMstate const Missile_armed_s = {
(QMstate const *)0, /* superstate (cop) */
Q_STATE_CAST(&Missile_armed),
Q_ACTION_CAST(0), /* no entry action */
Q_ACTION_CAST(0), /* no initial tran */
};
static QState Missile_exploding (Missile * const me, QEvt const * const e);
static QMstate const Missile_exploding_s = {
(QMstate const *)0, /* superstate (cop) */
Q_STATE_CAST(&Missile_exploding_e),
Q_ACTION_CAST(&Missile_exploding_e),
Q_ACTION_CAST(0), /* no initial tran */
};
static Missile l_missile; /* the sole instance of the Missile active object */
/* Public-scope objects
QActive = const AO_Missile = (QActive *)&l_missile; /* opaque pointer */
/* Active object definition
$define(AOS:Missile_ctor)
$define(AOS:Missile)
.....
void Missile_ctor(void) {
Missile me = &l_missile;
QActive_ctor(&me->super, Q_STATE_CAST(&Missile_initial));
}
$AOS:Missile_ctor
$AOS:Missile_ctor
.....
static QMstate Missile_armed_s = {
(QMstate const *)0, /* superstate (cop) */
Q_ACTION_CAST(0) /* zero terminator */
};
```

Expanded \$define() Code-Generation Directives

File Template in QM

Extending QM™ with Command-Line Tools



```
Log Console

{{{ External tool "build-Debug"

INFO> Code generation started (07:21:17.862 am)
INFO> Entire model: D:\qp\qpc\examples\arm-cm\qk\iar\game-qk_ek-1m3s81
INFO> Code generation ended (time elapsed 0.031s)
INFO> 0 file(s) generated, 7 file(s) processed, 0 error(s), and 0 warn:

%IAR_EWARM%\common\bin\IarBuild.exe game-qk.ewp -build Debug

      IAR Command Line Build Utility V7.0.3.3119
      Copyright 2002-2014 IAR Systems AB.

Building configuration: game-qk - Debug
Updating build tree...

13 file(s) deleted.
Updating build tree...
bsp.c
display96x16x1.c
main.c
mine1.c
mine2.c
missile.c
ship.c
startup_1m3s.c
system_1m3s.c
tunnel.c
Linking
game-qk.out

Total number of errors: 0
Total number of warnings: 0

}}}
```

Welcome to the 21st Century!

- Experts avoid shared-state concurrency and blocking
- Experts use the event-driven Active Object design pattern
- Experts use hierarchical state machines instead of “spaghetti code”
- Event-driven active objects and state machines require a paradigm shift from sequential to event-driven programming
- QP™ real-time frameworks provide a very lightweight, reusable architecture based on the AO pattern and hierarchical state machines for deeply embedded systems, such as single-chip MCUs
- QM™ modeling tool eliminates manual coding of your HSMs