EVENT DRIVEN PROGRAMMING

- Embedded Real Time Systems
- Ron Barker

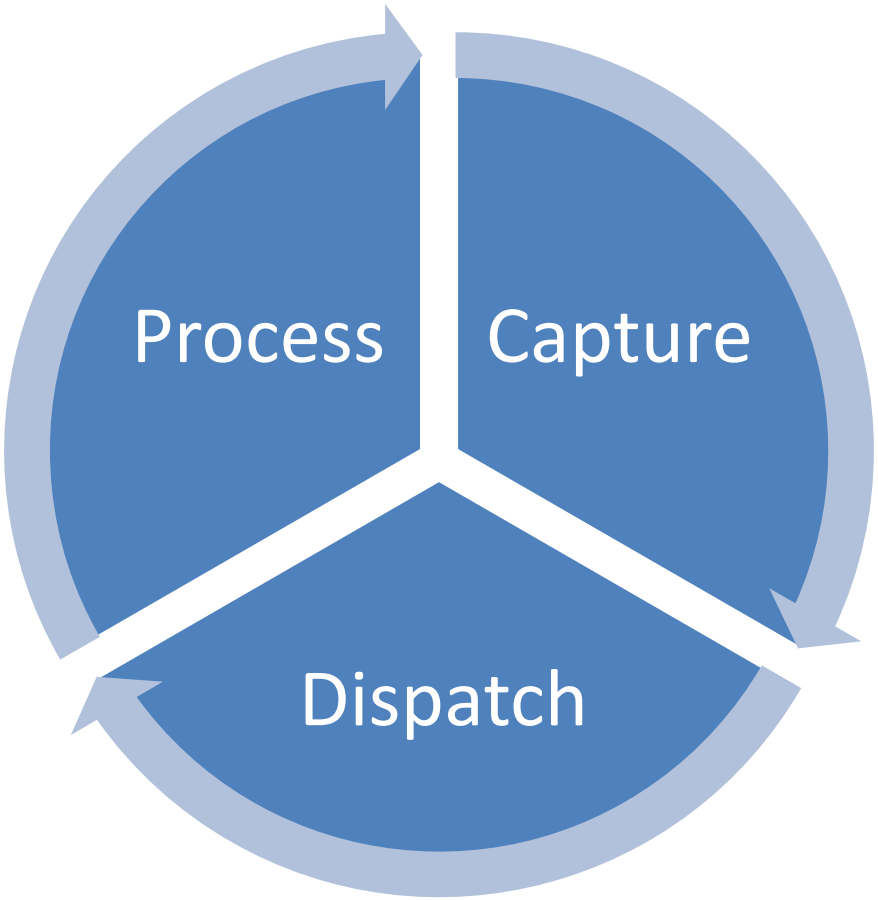
HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

REVIEW
MODELING WITH QM – BLINKYQM

Real Time Frameworks

EVENT CAPTURE

Real Time Event Driven Cycle

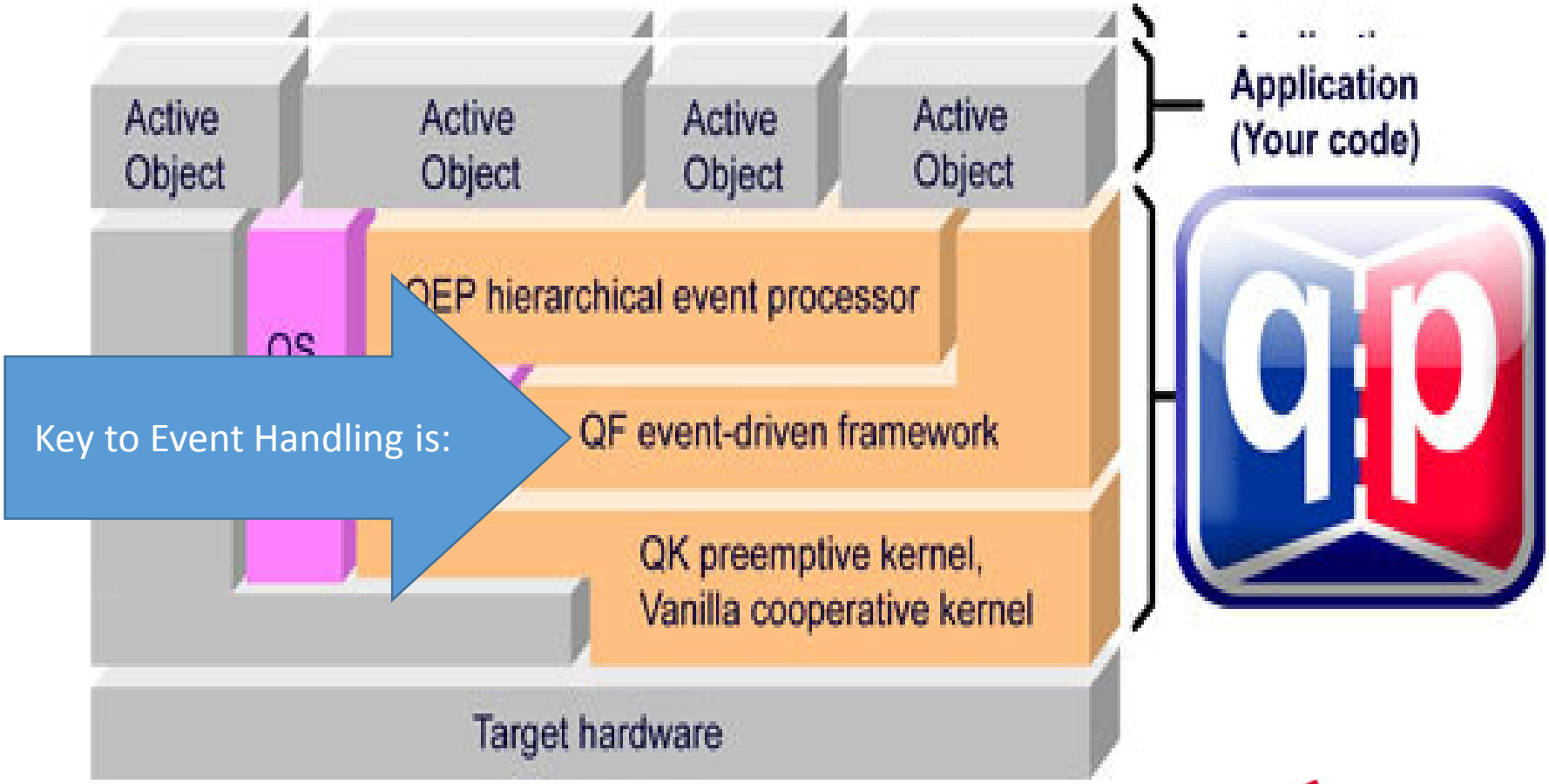


HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Event Dispatch Review

- Inversion of Control Execution Model
  - control resides in the event manager
  - application handler is called only when event arrives and runs to completion
- Based on A Generic Event Handler
- Enables N events to be processed in parallel
- Question: How best to communicate events?

QF Framework



Framework Events Management

- Quantum manages Events via Queues

- Event Queues Attributes
  - Simple Async Comm
  - Rely on Scheduler Support
  - De-coupled from Dispatch
  - De-coupled from Process
  - Dynamic Protection Levels

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Event Queues Basics

- Event Queues == RTOS
  - Queues Need Consumers
  - Consumers == TASKS
  - Tasks require Scheduling
  - Scheduling means Time Management

Real Time Frameworks

TIME MANAGEMENT

Frameworks depend on Clocking

- Clocking Mean Interrupts
  - Framework Heartbeat
  - Intrinsic Resource
  - Source of Time Events

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Interrupts and Frameworks

- Interrupts are required for scheduling
- How does Scheduler know when an interrupt occurs?

Framework / RTOS Clocks

- Basic Principle is:
  - Signaling Mechanism from
  - ISR -> Scheduler Code is required
- All Frameworks / RTOS have a special interface
  - Nucleuls+ NU_Tick()
  - FreeeRTOS –
  - QM – QM_Tick
- This interface must be integrated into HW Timer ISR

Framework Time Management

- QF manages time via Time Events
  - Time Events: Instances of a Class:QTimeEvt
  - QTimeEvt Structure:

```
typedef struct QTimeEvtTag {
  QEvt super;
  struct QTimeEvtTag * volatile prev;
  struct QTimeEvtTag * volatile next;
  void * volatile act;
  QTimeEvtCtr volatile ctr;
    /** the internal down-counter of the time event. The down-counter
    * is decremented by 1 in every QF_tickX_() invocation. The time event
    * fires (gets posted or published) when the down-counter reaches zero.
    */
   QTimeEvtCtr interval;
} QTimeEvt;
```

THE Consumer / Task

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

QF Time Event Interfaces

Initialisation – 1 each for each Time event

```
QTimeEvt_ctor(&BlinkyTimer.super, BLINKY_TIME_SIG);
```

Periodic time event

```
QTimeEvt_postEvery(&BlinkyTimer.super, (QActive *)me, 5);
```

One-Shot time event

```
QTimeEvt_postIn( );
```

Disarm any

```
QTimeEvt_disarm();
```

Rearm any

```
QTimeEvt_rearm();
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

BLINKY AGAIN
BLINKY , TIMER ISR + QF_TICK

Blinky + Framework

- Update Blinky UML with ISR driver System Tick
- Refactor TICK Event as QTimeEvt Event

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Event Queue Troubles?

- Blinky Events in a Framework Event Queue,

  Something go wrong ??

- Is this modeling creating a mess?

- Where are the problems??

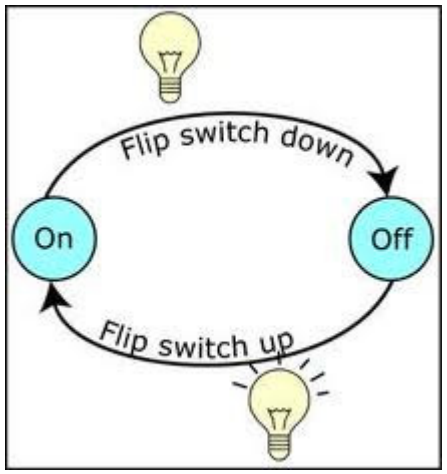
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Real Time Frameworks

EVENT PROCESSING

Bringing it all together

1. State Machine



2.Event Queue

3. Thread of Excution
Or Task

=

Active Object

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Active Object Definition

- An Active Object combines
  - A State Machine Object
  - An Event Object Process (Event Queue)
  - Thread of Execution with unique priority
- The State Machine is the basic data object
- Event Queues ->
  - Answer to How ISR Events communicate to event processor
- Execution Thread provides execution control

QF Active Object Implementation

- QF provides Super Class Active
- Active objects in QF are encapsulated tasks
  - embedding a state machine (FSM or HSM)
  - and an event queue for
    - asynchronous communicate
    - by sending and receiving events.
- Within an active object, events are processed sequentially in a run-to-completion (RTC)
- QF encapsulates all the details of thread-safe event exchange and queuing.

QF Active Objects Interfaces

Initialisation – 1 each for each AO

```
QActive_ctor(&me->super, (QStateHandler)&BlinkyAO_initial);
```

Creat Task

```
QActive_start((QActive *)&l_blinkyAO, 1, l_ADEvtQSto,
Q_DIM(l_ADEvtQPoolSto),   (void *)0, 0, (QEvt *)0);
```

QF Active Event Queue Interfaces

Dynamic Event Pools

Initialisation – Only needed for dynamic Events

```
QF_poolInit(l_ADEvtQPoolSto,sizeof(l_ADEvtQPoolSto),sizeof(l_ADEvtQPoolSto[0]));
```

```
Q_NEW(BlinkyADEvt, AD_SIG);
```

Direct Event Posting

```
QF_psInit(l_subscrSto, Q_DIM(l_subscrSto));
```

```
QF_publish((QEvent *)evt);
```

```
QF_Active_subscribe(QActive *)me, (Qevent) *);
```

Queued Post Event

```
QActive_postFIFO((QActive *)me, &LcdDisplayEvt);
```

```
QActive_postLIFO((QActive *)me, &LcdDisplayEvt);
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN