Java™ Technology
in Embedded- und Echtzeit-Systemen

thomas.hauser@esmertec.com

esmertec

Your partner for embedded Java™ technology

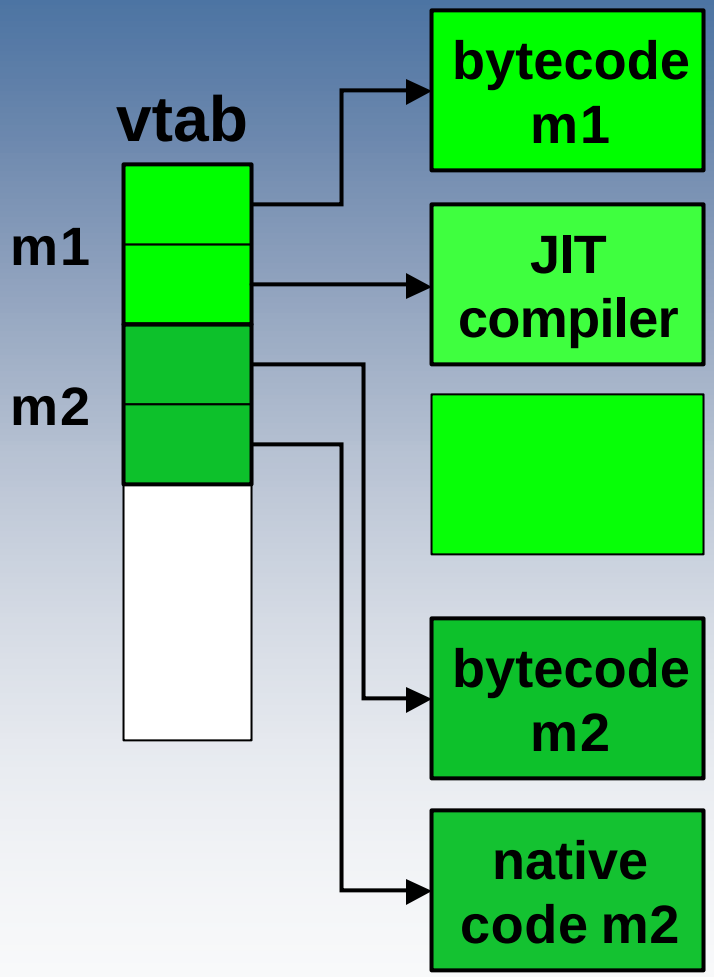
Challenges for Embedded System

- Execution speed
  - Interpreter needs resources !
- Application size
  - Cost of Chips, probability of bugs
- Garbage collection
  - Is this new ?

Execution Bytecode

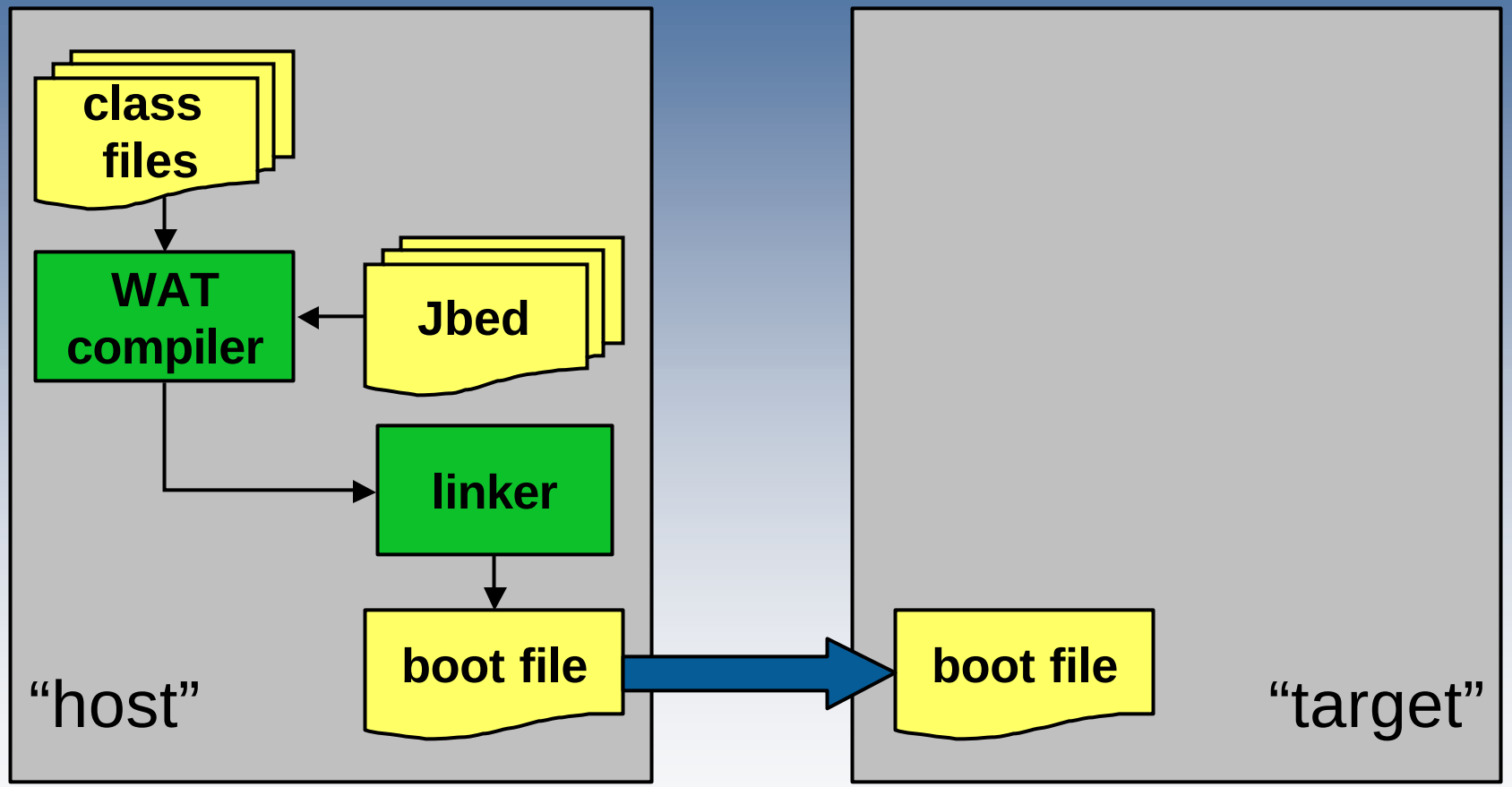
- Interpretation
  - Simple, powerful, slow
- Just-in-time compilation (JIT)
  - Complex, powerful, fast, unpredictable
- Way-ahead-of-time compilation (WAT)
  - Simple (classic compiler construction), powerful (for Java), very fast, predictable
  - But too static (dynamic code loading hard to support)

Just in Time Compiler



- Compiles at first use
- Compile time adds to runtime
  - Only few optimizations
- Too unpredictable for hard real-time systems

Way Ahead Compiler



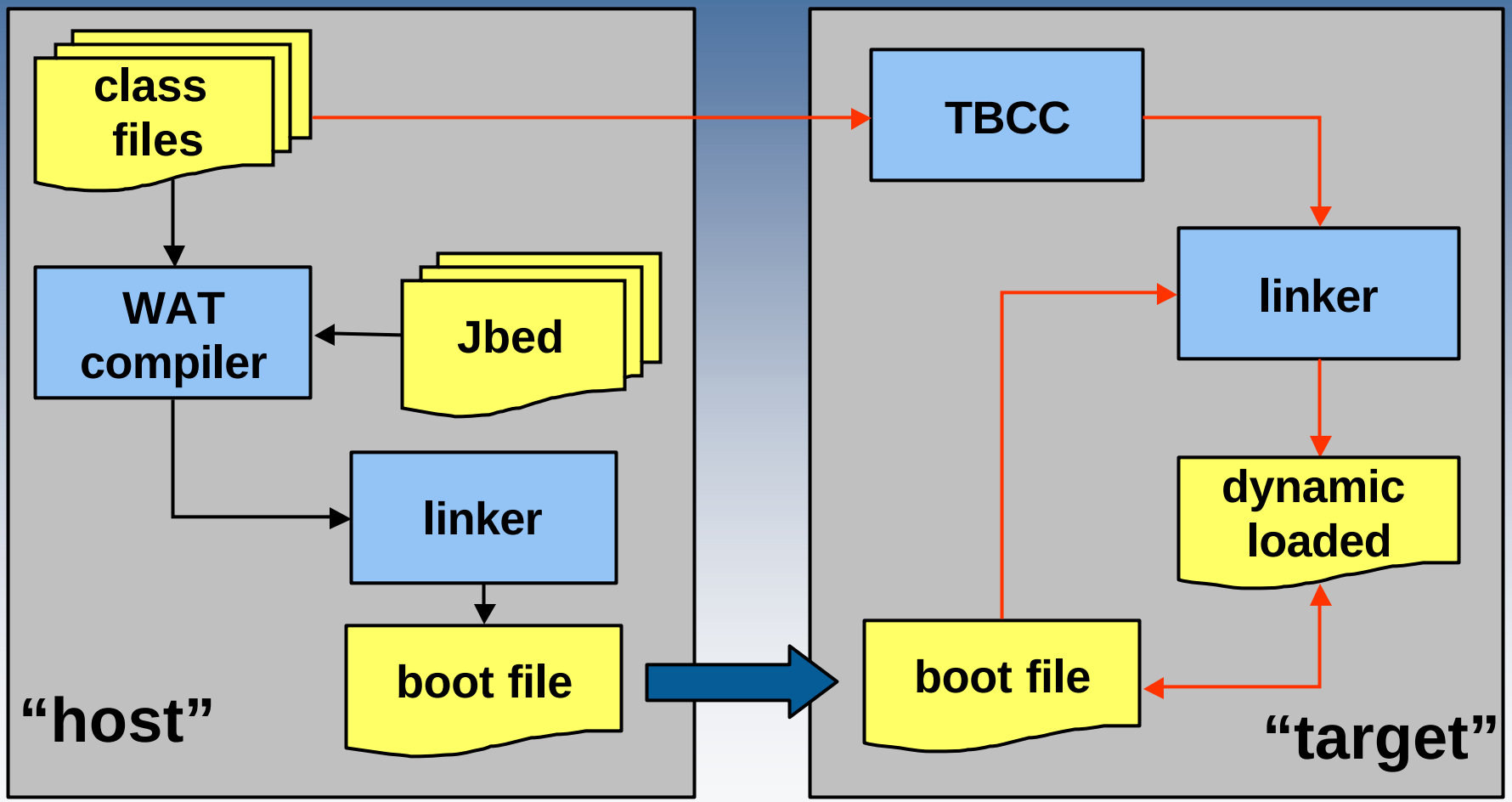
Way Ahead Compiler

- No runtime overhead
- Expensive code generation and optimization is possible

e.g. for M68k	nofilter	filter
HelloWorld (log)	204	38 kbytes
Jura (with GC)	225	60 kbytes

- Dynamic loading ?

Target Byte Code Compiler TBCC™



Dynamic Loading of Classes

```
// Dynamically load class and invoke static method within it.
    Class  hlClass  = null ;
    Method hlMethod = null ;

    try { // Dynamically load class
       hlClass = Class.forName ("com.jbed.HotLoadOne") ;

       // Retrieve static method in dynamically-loaded class
       hlMethod = hlClass.getMethod ("staticMethod",null) ;

       // Call dynamically-loaded static method
       hlMethod.invoke (null,null) ;

    } catch (Throwable e) {
          e.printStackTrace () ;
    } // Try
```

Dynamic Loaded Class

```
// real package name -> com.jbed.examples.hotload;
package com.jbed ;

public class HotLoadOne{

   // Static class method
   public static void staticMethod () {
      System.out.println ("HotLoadOne: staticMethod()") ;
   }

}  // end of class
```

Summary on Speed and Size

- **Speed : Java compiled code is comparable to C/C++**
- **Size : Java compiled and linked is comparable to C/C++**

But still Java !

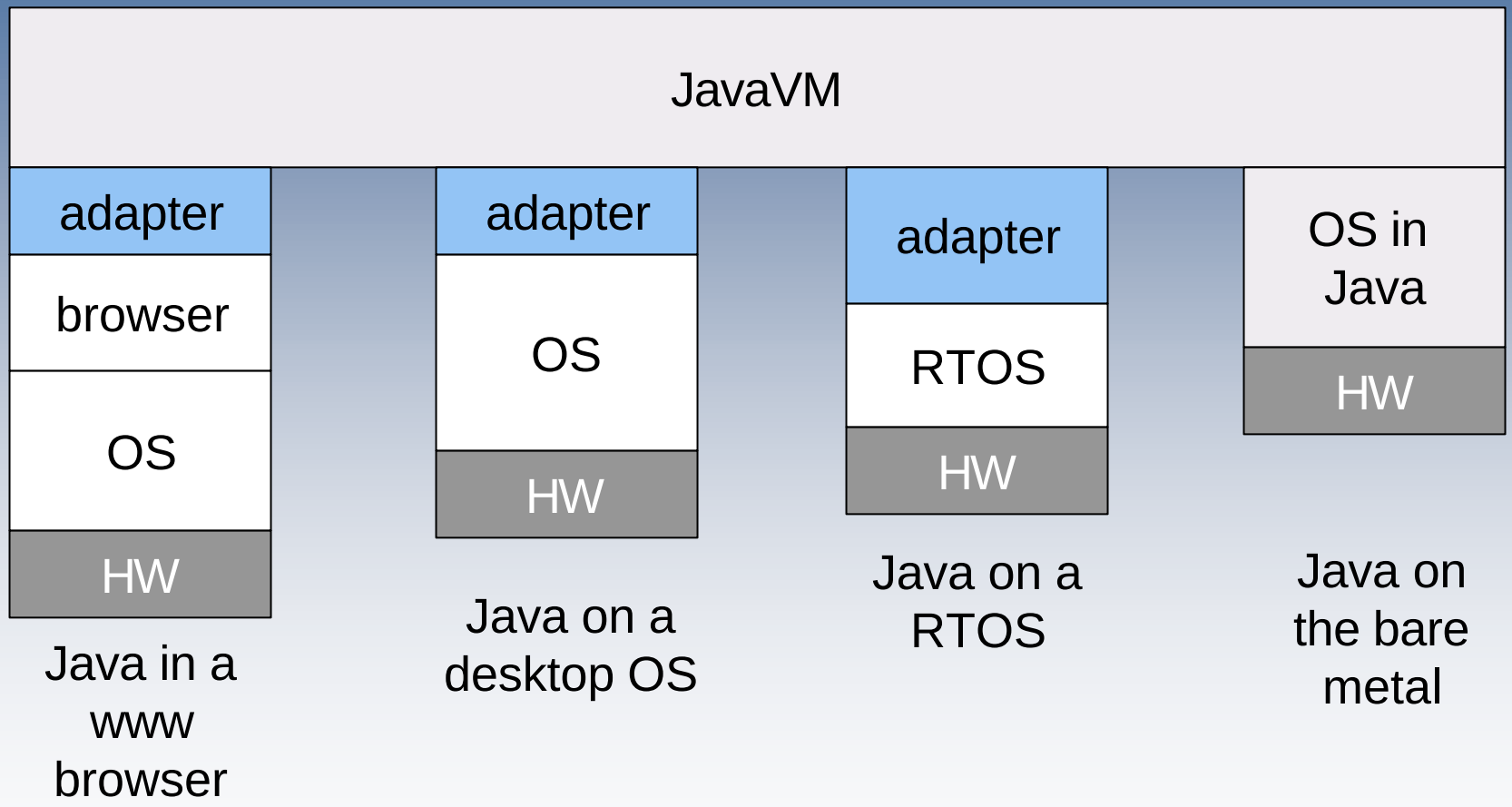
Still possible to gain more space and speed ?

Java runtime System

- Java uses the thread concept
  - parallel execution units within one process
  - using same process resources and address range
- “Standard” Java threads as defined by

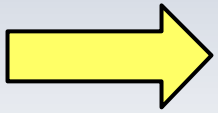
  `java.lang.Thread` and `java.lang.Runnable`
- Priority based, preemptive multitasking
  - 10 Priorities
- Standard Java monitors (`synchronize`)
- Jbed adds priority inheritance

VM Implementations



Jbed

- Written in Java
- Provides Java API
- Combines Java Virtual Machine
  and Operating System



Jbed RTOS Package

always compiles never interprets

Embedded System

....and pure Java Technology

How about .....

Driver‘s in Java

```
public static void show (boolean on) {
    // read LED memory mapped register
    int bits = Unsafe.getInt(PORTC);

    if (on) {
         bits &= 0xFFFFFFFD;
    } else {
         bits |= 0x00000002; }

    // write LED memory mapped register
    Unsafe.putInt(PORTC, bits);
}
```

Java and Real-time ?

- **Soft Real-time**

  Button pressed
  Light may react in 100 –500 ms

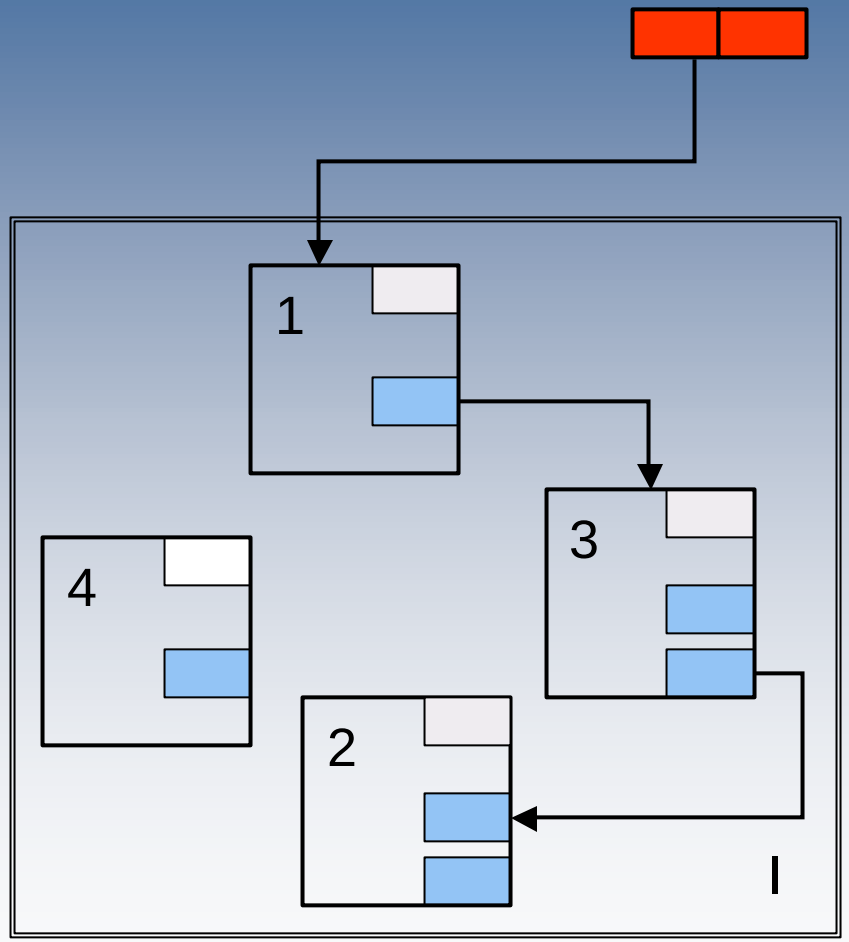
- **Hard Real-time**

  UART ISR (no FIFO, 9600, 8, 1, n)
  duration 1uS
  it must end till 1 ms (Deadline)

Garbage Collection

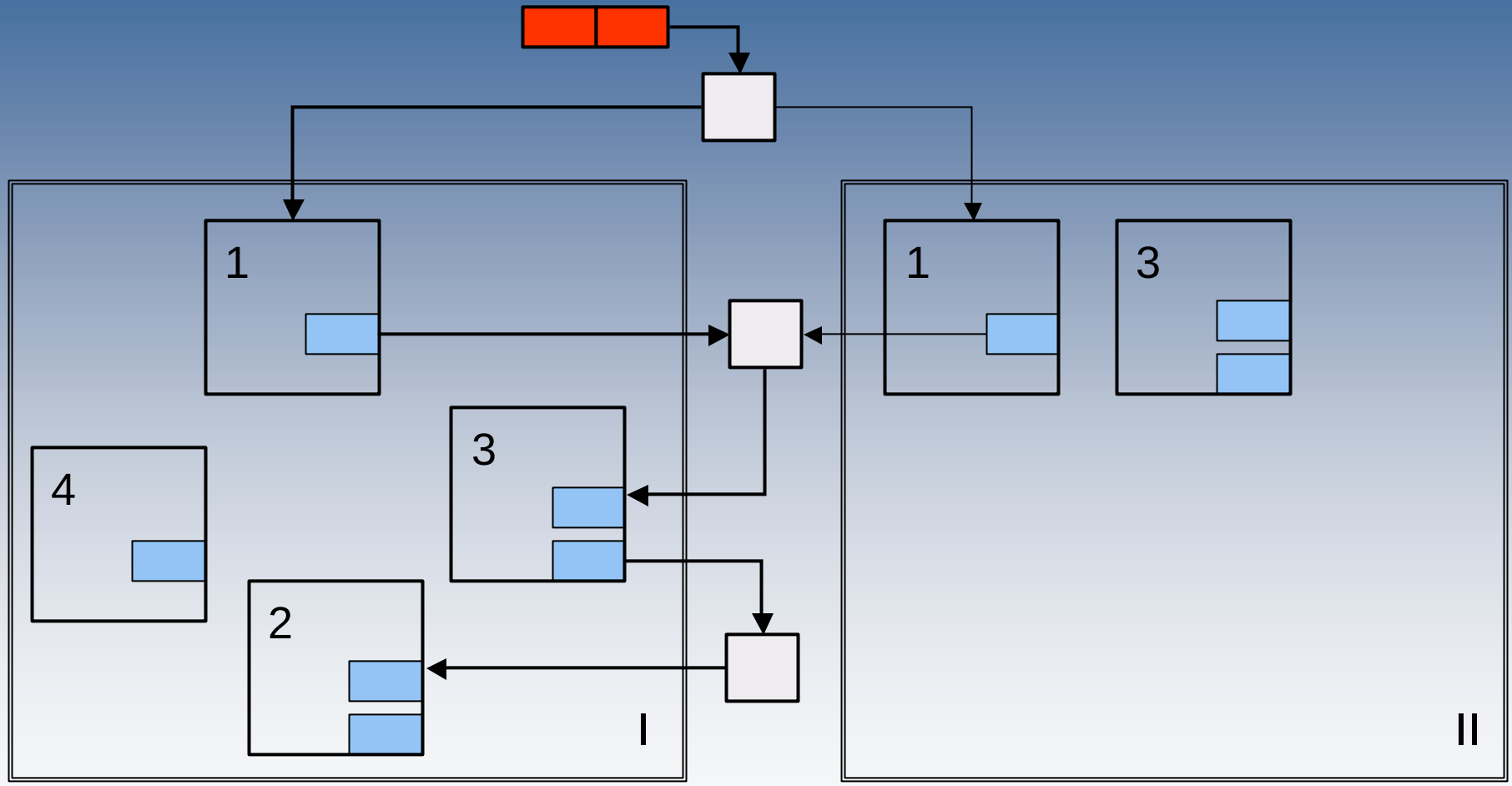
- **Objects/arrays are allocated on the heap**
- **Two classes of algorithms**
  - Copying garbage collection / mark and sweep
- **Blocking / non blocking**
  - Real-time requires non blocking
- **Precise / imprecise**
  - In Java, all pointers are known

Mark and Sweep Garbage Collection



- 80 % are small objects
- No copy of large object (arrays)
- Garbage Collector is a thread in the system
- Declare large Objects static

Copying Garbage Collection



Conventional RTOS Schedulers

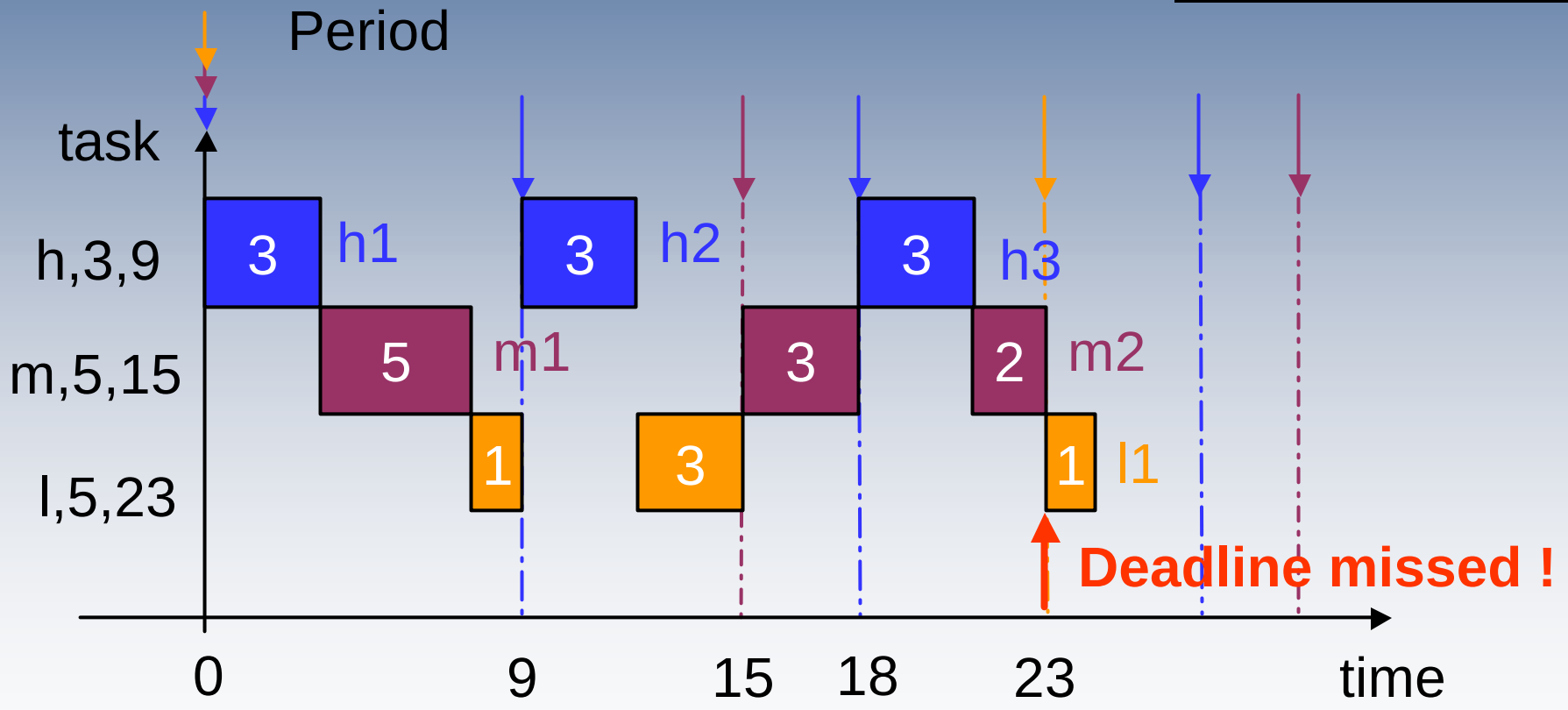
- Event-driven systems
- Priority based, preemptive scheduling
  - Round-Robin scheduling
  - Highest priority first, run to completion
- Problems
  - Translate time constraints into priorities
  - How to guarantee a deadline
  - How to add a new component

Hard Real-time
Example Application

Task	Duration	Period	Priority
h	3	9	High
m	5	15	Medium
l	5	23	Low
Total	13		

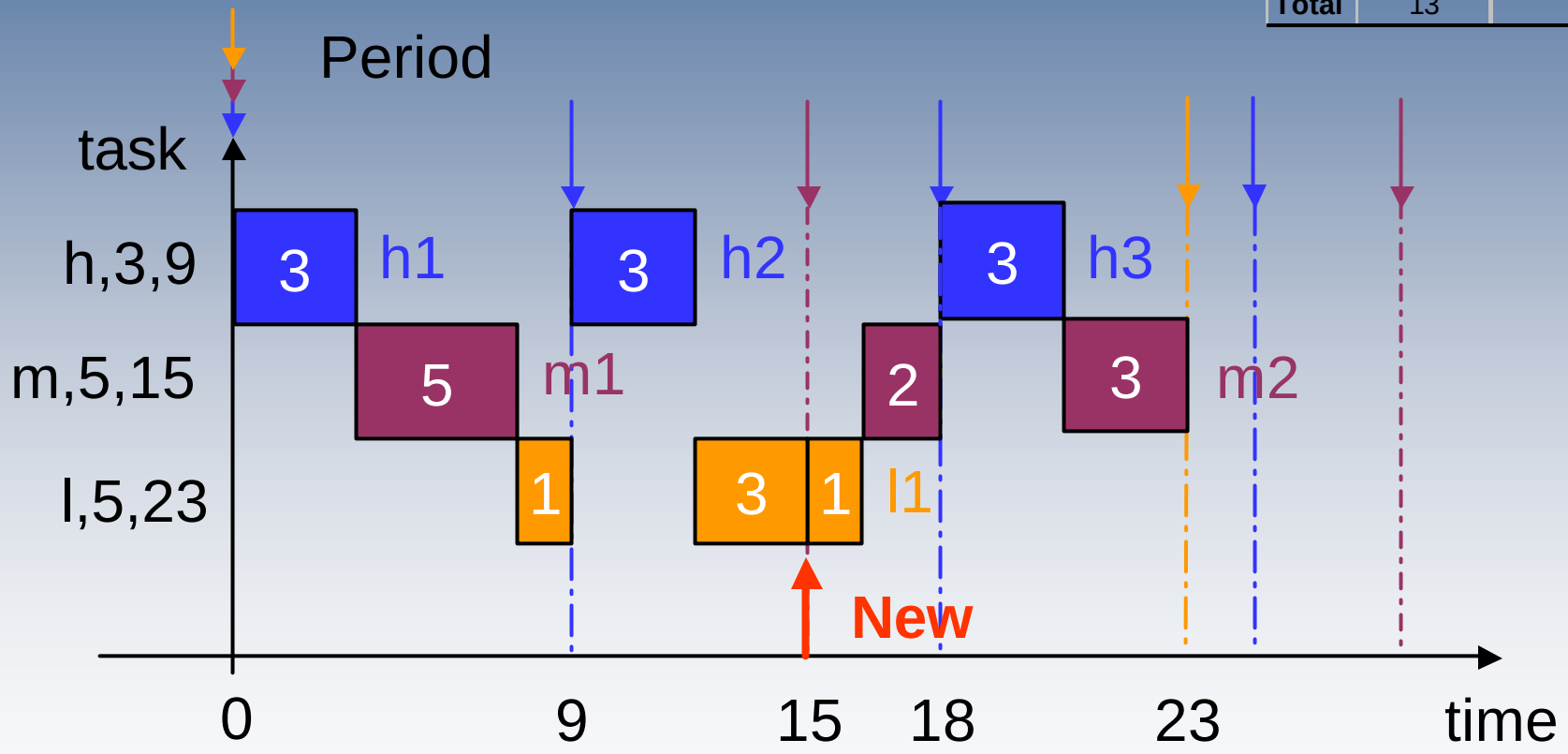
Priority Based Scheduler

Task	Duration	Period	Priority
h	3	9	High
m	5	15	Medium
l	5	23	Low
Total	13		



Earliest Deadline First Scheduler

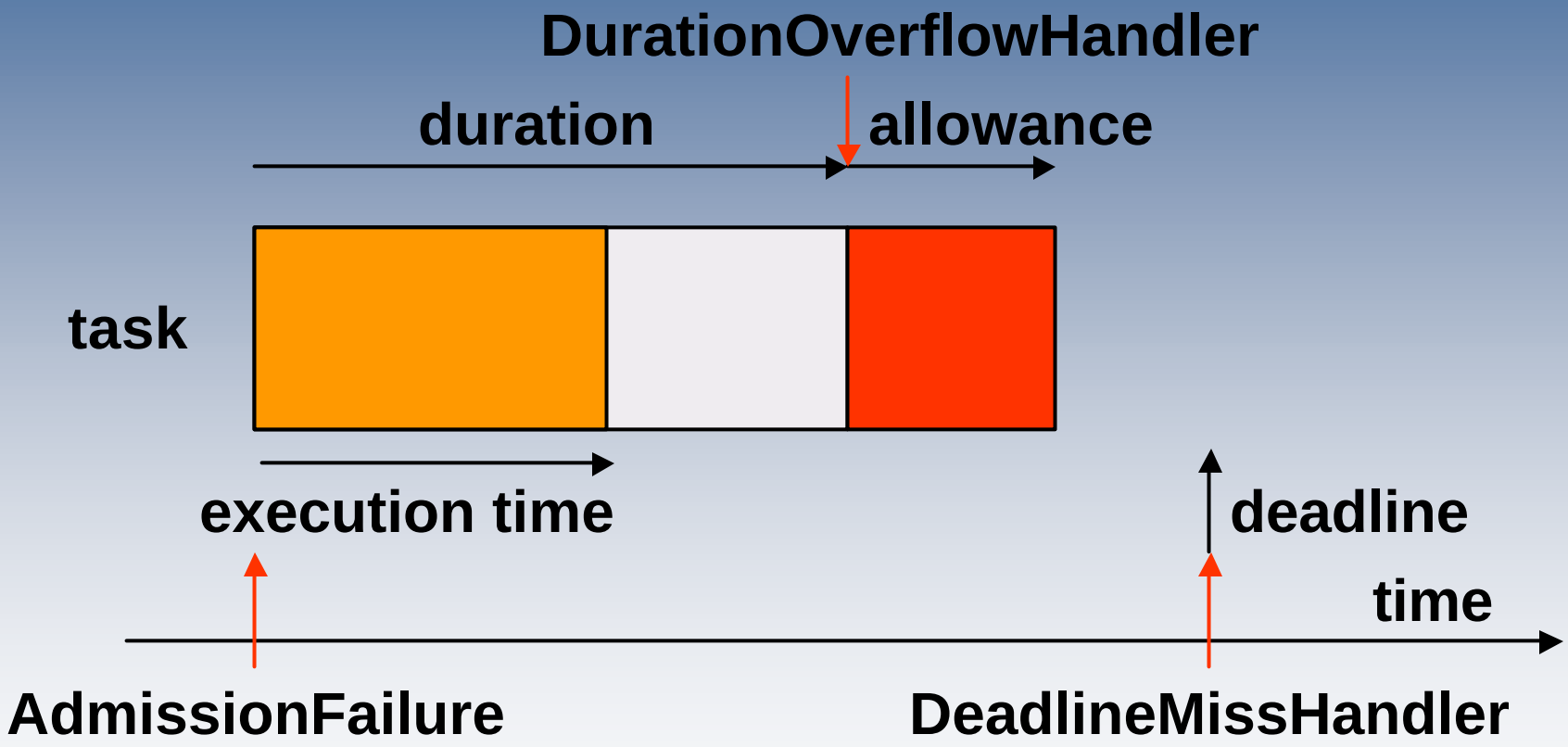
Task	Duration	Period	Priority
h	3	9	High
m	5	15	Medium
l	5	23	Low
Total	13		



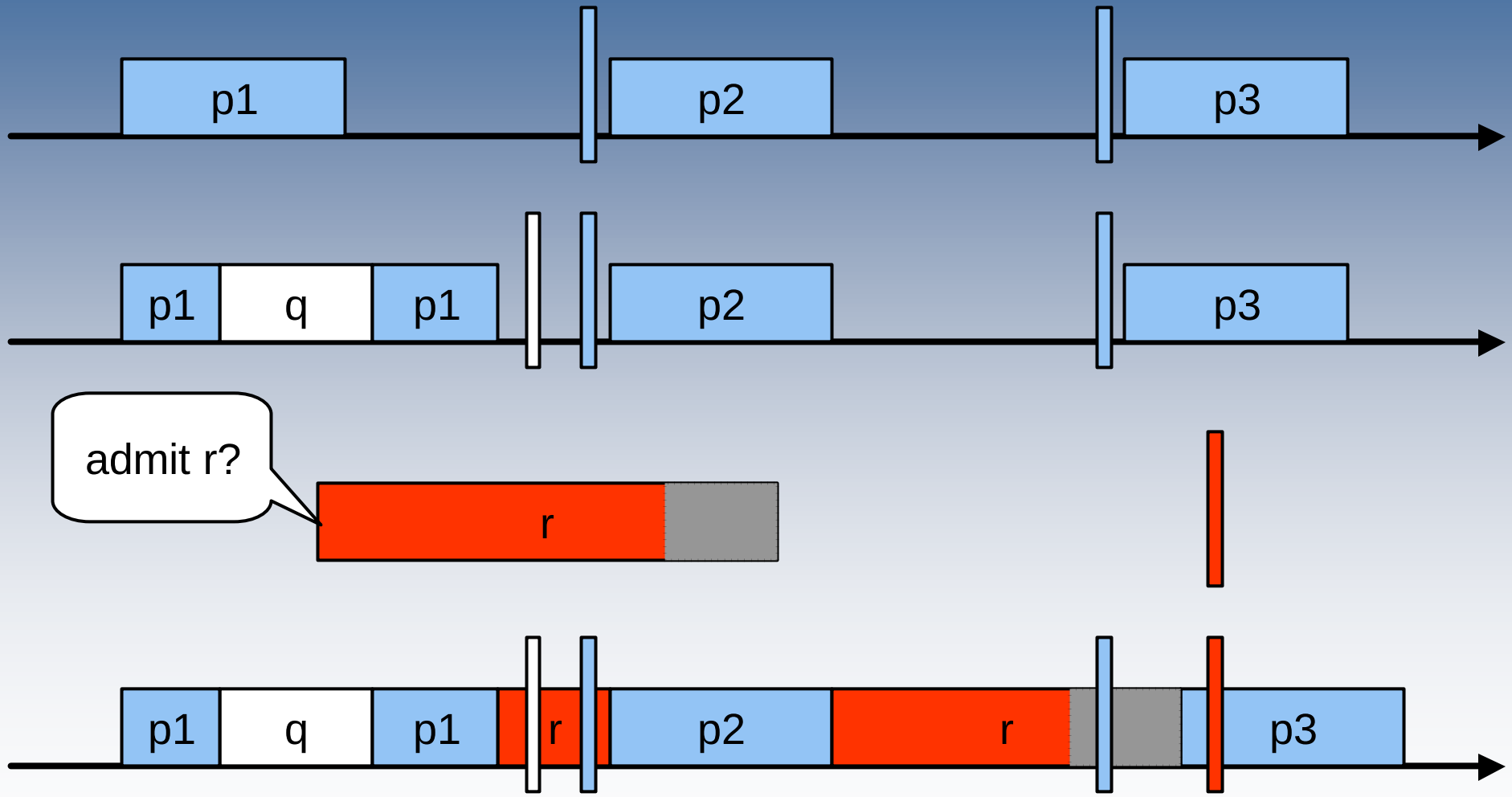
Earliest Deadline First Scheduler

- **Scheduler runs the task with the closest deadline**
- **Advantages**
  - Admission testing
  - Guaranteed deadlines for all tasks (even if created at runtime)
  - Safely add new tasks and components
- **Interrupts can be handled like tasks**
  - Need to know max. frequency

Earliest Deadline First Scheduler



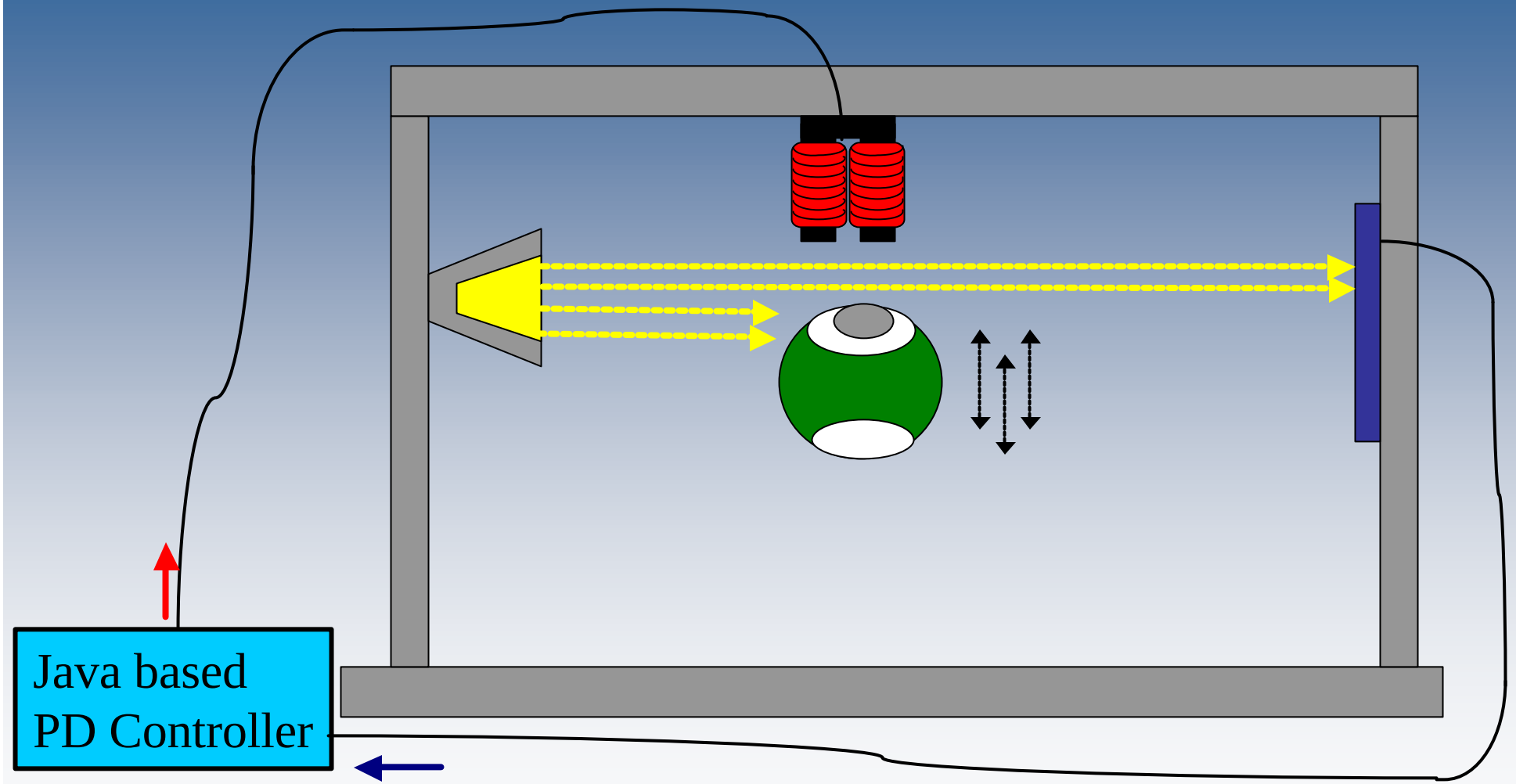
Add new tasks



Scheduling Caveats

- If tasks synchronize using blocking primitives, admission testing cannot guarantee deadlines
  - **`DeadlineMissHandler`**
- A task can not get more runtime than requested upfront
  - **`DurationOverflowHandler`**
- Real-time and processor load are limited to 90%
- Resolution of 1 system tick for period and delay

Real Example : Floating Ball



PD Controller

```
class PDController extends Task {
  ...

  public final void run () {
    isPos = - sensor.read();    // analog sensor
    isSpeed = (isPos-oldPos) * CLOCK_INVERSE;
    oldPos = isPos;
    outForce = kp * (sollPos-isPos)
             - kd * isSpeed;
    currentCoil.write(outForce); // analog
                                 // actuator
  }
} // end of class
```

Installing the PD Controller

```
try {
  RealtimeEvent event = new PeriodicTimer(1000);

  // duration: 500, allowance: 0,
  // deadline: 1000, period: event
  controller = new PDController(500,0,1000,event);

  controller.start();

  System.out.println("controller installed");
} catch (AdmissionFailure e) {
  System.out.pintln(“admission test failed”);
  ...
}
```

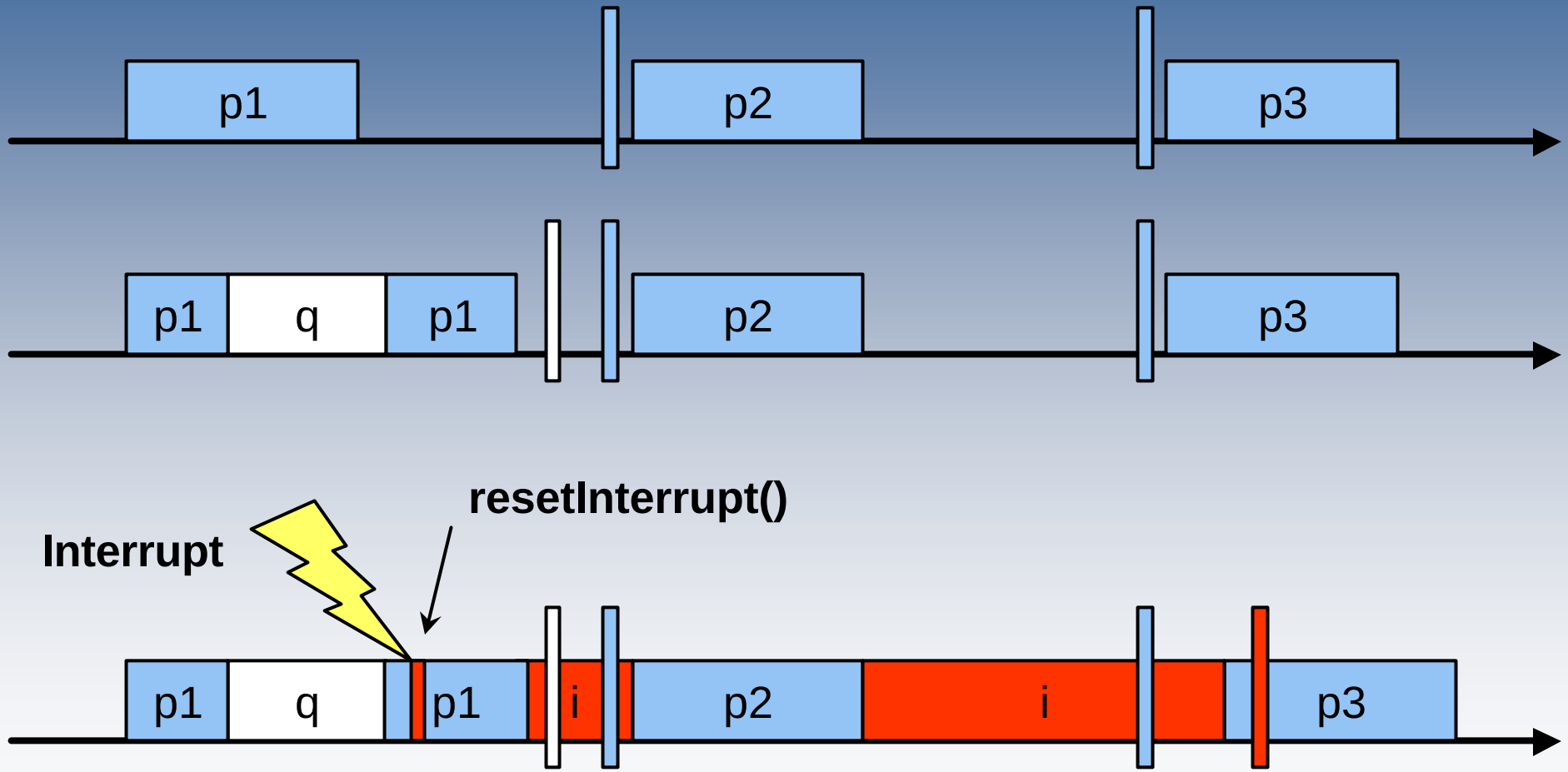
Task Classes

- **Oneshot** Executes once
- **Periodic** Executes at regular intervals
- **User** Executes when a task signals an event
- **Interrupt** Executes when HW signals an event

Interrupt Handling

- **Interrupt task**
  - Specify max. frequency
  - First reset interrupt
  - Then start interrupt task
  - Overwrite the reset method
- **Interrupt handler (no task)**
  - Outside of scheduler control
  - Specify max. load

Interrupt



Example: Interrupt Service Event

```
static final class ServiceEvent
  extends InterruptEvent
{
  int VECTOR = 3; // hw addr for interrupt
  ServiceEvent(long minPeriod) {
    super(minPeriod, VECTOR);
  }

  public void resetInterrupt() {
    // interrupt reset
    Unsafe.putInt(PORTC
                  ,Unsafe.getInt(PORTC) | 0x02);
  }
}
```

Example: Interrupt Task

```
static final class HandlerTask
  implements Runnable
{
  static int count = 0;

  public final void run() {
    // yellow led
    Unsafe.putInt(PORTC
                  ,Unsafe.getInt(PORTC) | 0x04);
    count++;
  }
}
```

Example: Installing the Interrupt Task

```
int MIN_PERIOD = 100000;  // us
...
try {
  ServiceEvent sEvent = new ServiceEvent(MIN_PERIOD)
  Runnable handler = new HandlerTask();
  Task i = new Task(handler, 200, 0, 500, sEvent);
  i.start();
  ...
} catch (AdmissionFailure e) {
  ...
}
```

Example: Installing an Interupt Handler

```
static void handler() {
  Unsafe.option(Unsafe.noFrame);
  Inline.saveIrqState();
  // yellow led & interrupt reset
  Unsafe.putInt(PORTC, Unsafe.getInt(PORTC) | 0x06);
  count++;
  Inlines.restoreState();
}

...
System.setHandler(0x300
                  ,Unsafe.staticAddr("handler"));
...
```

Synchronization

- Between Threads
  - All standard Java mechanisms
- Between Tasks
  - All standard Java mechanisms
- Between Threads and Tasks
  - Start a Task from a Thread
  - Signal a Thread from a Task (`wait()/notify()`)
  - Do not use `synchronize`
    - System can throw `IllegalSynchronizationException`
  - Use non-blocking communication primitives

The two Worlds

- **The “standard” Java threads world**
  - Always called *thread* in Jbed
- **The world of real-time tasks**
  - Always called *task* in Jbed

Conclusion

- It is possible to use Java for hard real-time embedded systems
- EDF scheduling eases the development of the hard real-time part
- Having two schedulers is possible and powerful
- Be careful with synchronization

Still time for questions

Please

Real-Time Specification for Java

- Add/exchange scheduling policies
  - Default: preemptive priority, FIFO
  - Min. 28 priorities
- Prevent/bound eligibility inversion
  - Default: priority inheritance
- New types of memory (`ScopedMemory`, `ImmortalMemory`)
  - Threads not affected by the GC
- Abstractions for asynchronous event handling
- Asynchronous transfer of control
- `PhysicalMemory` for direct memory access