

MOST IMPORTANT PROGRAMMING GUIDELINE:

**MAKE INTERFACES EASY TO USE
CORRECTLY AND HARD TO USE
INCORRECTLY,**

BUT ACHIEVING IT CAN BE CHALLENGING.

- Scott Meyers -
Author, Consultant
&
C++ Guru

WHO AM I AND WHAT AM I DOING HERE?

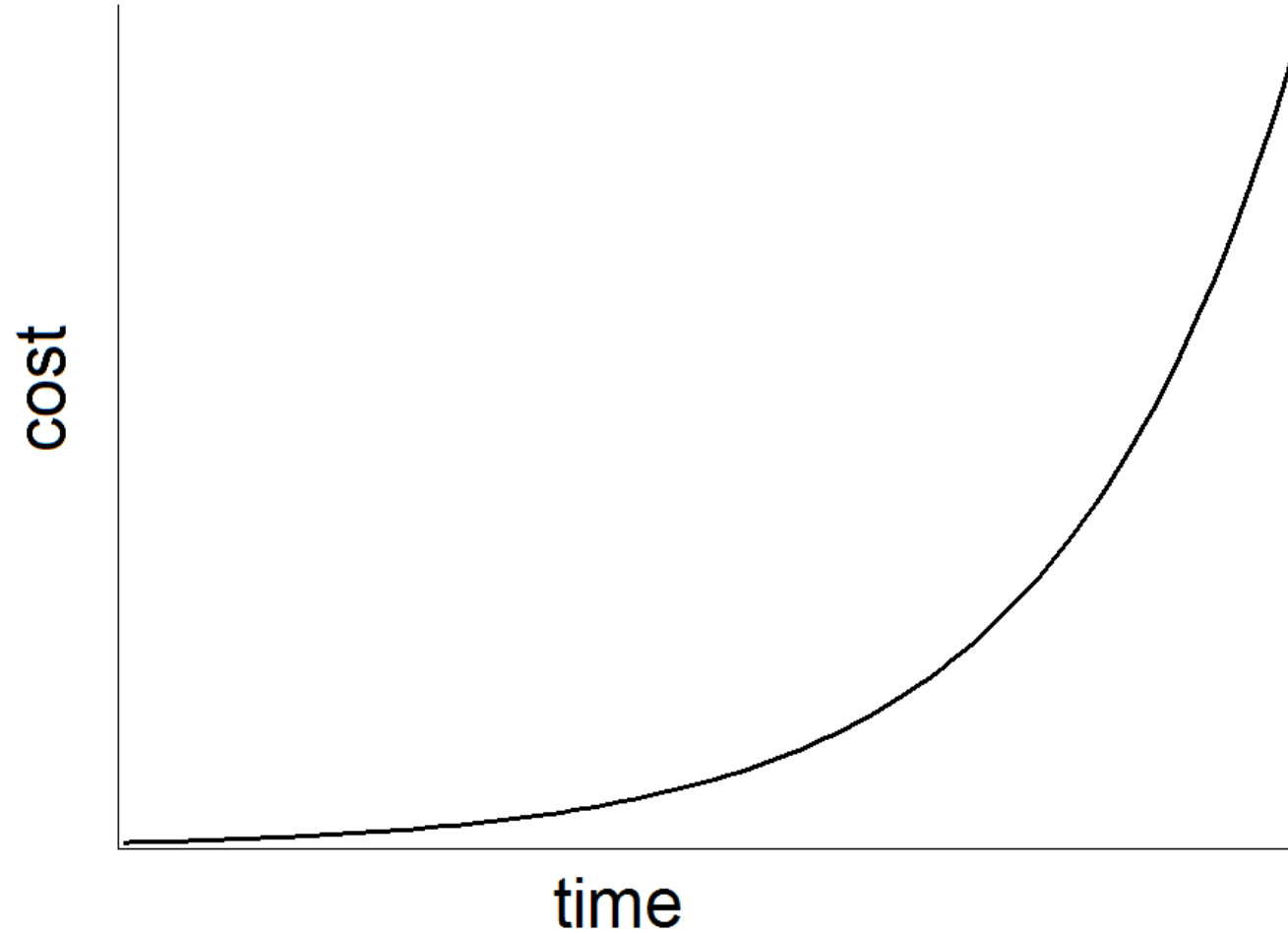
- ▶ C++ Evangelist
- ▶ Template Meta Programming Nerd
- ▶ Embedded Developer
- ▶ Kvasir.io / brigand / SG14 contributor
- ▶ Part of an awesome development team!
- ▶ Contract developer / Consultant

Immer eine zündende Idee.

EMBEDDED C++ PARADIGMS

Saving the World, One Bit-Trick at a Time

DIN 55350



CODE REVIEW

```
void f(){  
    //...  
    static int* stack = (int*)malloc(sizeof(int)*100);  
    stack[top++] = LPUART_C_ADDR_REG & C_ADDRESS_MASK >> C_ADDRESS_OFFSET;  
    LPUART_C_STAT_REG |= C_RX_FLAG_MASK;  
    //...  
}
```

CODE REVIEW

```
//deep in some header
#define LPUART_C_ADDR_REG *(volatile unsigned*) 0x40011004
#define LPUART_C_STAT_REG *(volatile unsigned*) 0x4001100C
//somewhere else in some header
#define C_ADDRESS_MASK 0x1F000000
#define C_ADDRESS_OFFSET 16
#define C_RX_FLAG_MASK (1<<7)

void f(){
//...
static int* stack = (int*)malloc(sizeof(int)*100);
stack[top++] = LPUART_C_ADDR_REG & C_ADDRESS_MASK >> C_ADDRESS_OFFSET;
LPUART_C_STAT_REG |= C_RX_FLAG_MASK;
//...
}
```

WHERE BUGS ARE BORN:

volatile

See John Regehr, University of Utah for good papers and blogs

REORDERING RULES

```
constexpr int BUF_SIZE = 40;
```

```
volatile int buffer_ready;  
char buffer[BUF_SIZE];
```

```
void buffer_init() {  
    int i;  
    for (i = 0; i < BUF_SIZE; i++)  
        buffer[i] = 0;  
    buffer_ready = 1;  
}
```


REORDERING RULES WITH ISRs

```
//in a header file, unmodified CMSIS implementation
__STATIC_INLINE void NVIC_DisableIRQ(IRQn_Type IRQn)
{
    NVIC->ICER[0] = (uint32_t)(1UL << (((uint32_t)(int32_t)IRQn) & 0x1FUL));
}
__STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
    NVIC->ISER[0] = (uint32_t)(1UL << (((uint32_t)(int32_t)IRQn) & 0x1FUL));
}
int my_flag;
void f(){
    //in user code
    NVIC_DisableIRQ(USB0_IRQn);
    my_flag++;
    NVIC_EnableIRQ(USB0_IRQn);
}
void g(){ my_flag--; } //in USB ISR
```

REORDERING RULES WITH ISRs

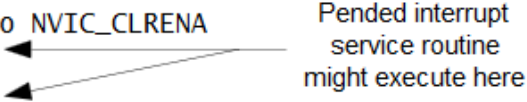
```
//in a header file, unmodified CMSIS implementation
__STATIC_INLINE void NVIC_DisableIRQ(IRQn_Type IRQn)
{
    NVIC->ICER[0] = (uint32_t)(1UL << (((uint32_t)(int32_t)IRQn) & 0x1FUL));
}
__STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
    NVIC->ISER[0] = (uint32_t)(1UL << (((uint32_t)(int32_t)IRQn) & 0x1FUL));
}
volatile int my_flag;
void f(){
    //in user code
    NVIC_DisableIRQ(USB0_IRQn);
    my_flag++;
    NVIC_EnableIRQ(USB0_IRQn);
}
void g(){ my_flag--; } //in USB ISR
```

HARDWARE SEQUENTIAL

Figure 13. Implemented interrupt disabling delay in Cortex-M processors

```

...           ; Interrupt is happening
STR R0, [R1]  ; Disable IRQ by writing to NVIC_CLRENA
...           ; Instruction N+1
...           ; Instruction N+2
...           ; Instruction N+3
    
```



Pended interrupt service routine might execute here

Architectural requirement

ARM recommends that the architectural requirement is adopted.

Depending on the requirements in your application:

- memory barriers are not required for normal NVIC programming when disabling an IRQ
- memory barriers are not required between NVIC programming and peripheral programming
- if it is necessary to ensure an interrupt will not be triggered after disabling it in the NVIC, add a [DSB](#) instruction and then an [ISB](#) instruction.

[Example 5](#) shows example code for changing an interrupt vector.

Example 5. Changing an interrupt vector

```

#define MEMORY_PTR(addr) (*((volatile unsigned long *)(addr)))
NVIC_DisableIRQ(device_IRQn); // Disable interrupt
__DSB();
__ISB();
MEMORY_PTR(SCB->VTOR+0x40+(device_IRQn<<2))=
(void) device_Handler; // Change vector to a different one
    
```

REORDERING RULES: GLOBAL INTERRUPT DISABLE

```
volatile int my_flag;  
asm ("cpsie i");  
my_flag = 1  
asm ("cpsid i");
```

```
asm volatile ("cpsie i");  
my_flag = 1  
asm volatile ("cpsid i");
```

```
asm volatile ("cpsie i:::");  
my_flag = 1  
asm volatile ("cpsid i:::");
```

REAL WORLD REORDERING RULES

```
struct USBHAL{  
    //definition of epCallback  
    bool(USBHAL::*epCallback[8 - 2])(void);  
  
    USBHAL(){  
        NVIC_DisableIRQ(USB0_IRQn);  
        // fill in callback array  
        epCallback[0] = &USBHAL::EP1_OUT_callback;  
        epCallback[1] = &USBHAL::EP1_IN_callback;  
        epCallback[2] = &USBHAL::EP2_OUT_callback;  
        epCallback[3] = &USBHAL::EP2_IN_callback;  
        epCallback[4] = &USBHAL::EP3_OUT_callback;  
        epCallback[5] = &USBHAL::EP3_IN_callback;  
        NVIC_EnableIRQ(USB0_IRQn);  
        //...  
    }  
}
```

CODE REVIEW

```
//deep in some header
#define LPUART_C_ADDR_REG *(volatile unsigned*) 0x40011004
#define LPUART_C_STAT_REG *(volatile unsigned*) 0x4001100C
//somewhere else in some header
#define C_ADDRESS_MASK 0x1F000000
#define C_ADDRESS_OFFSET 16
#define C_RX_FLAG_MASK (1<<7)

void f(){
//...
static int* stack = (int*)malloc(sizeof(int)*100);
stack[top++] = LPUART_C_ADDR_REG & C_ADDRESS_MASK >>
C_ADDRESS_OFFSET;
LPUART_C_STAT_REG |= C_RX_FLAG_MASK;
//...
}
```

BIT FIELDS

34.3.2 LPUART Status Register (LPUARTx_STAT)

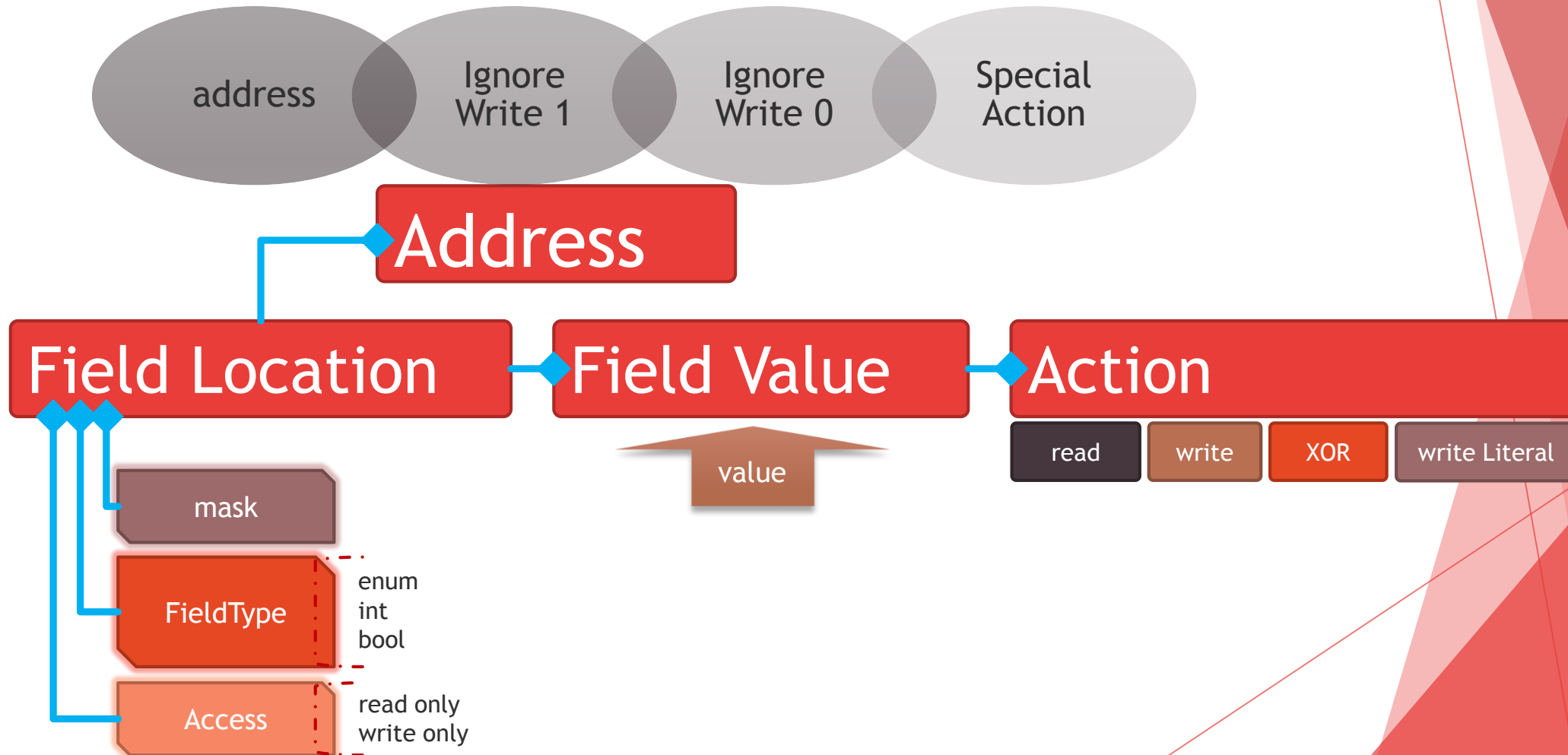
Address: Base address + 4h offset

[illegible]

#BITFIELDPROBLEMS

- ▶ Mask-register applicability errors
- ▶ Special register functionality:
 - ▶ Clear on read
 - ▶ Set to clear
 - ▶ Reserved bits
 - ▶ FIFO
- ▶ Field offset errors
- ▶ Optimization (volatile problems)
- ▶ ISR collisions

MENTAL MODEL OF REGISTERS



CONSTEXPR TYPE-ENCODED TAG

```
template<typename Location, typename Action, int Value>  
struct Action{};
```

```
template<unsigned Address, unsigned Mask, typename TType = int /* ... */>  
struct FieldLocation{  
    using DataType = TType;  
};
```

```
constexpr FieldLocation<0x1234, (1 << 4)> myBits{};
```

CONSTEXPR METAFUNCTION

//turn a bit location into an action which sets the corresponding bit

```
template<typename T>
constexpr Action<T, WriteTag, 1> set(T) {
    return{};
}
```

```
template<typename T>
constexpr typename MakeClearAction<T, WriteTag, 1>::type clear(T)
{
    return{};
}
```

```
template<typename... T>
void apply(T...) { /*lazy magic*/ }
```

USER CODE EXAMPLE

//user code

```
apply(set(Uart1::cfgEnable));  
apply(write(Uart1::cfgStopBValC::one,  
Uart1::cfgDataBValC::nine));
```

```
if(apply(read(cfgEnable))) == true){}
```

```
auto temp = apply(read(Uart1::cfgStopB, Uart1::cfgDataB,  
Uart1::prescale));  
if(temp == Uart1::cfgDataBValC::nine){}  
else if(temp[Uart1::prescale] == 100){}
```

//or C++17

```
const auto[s,d,p] = apply(read(Uart1::cfgStopB,  
Uart1::cfgDataB,  
    Uart1::prescale));  
if (d == Uart1::cfgDataBValC::nine) {}  
else if (p == 100) {}
```

REAL WORLD LIBRARY CODE

```
namespace Usb0Istat{    ///
```

REAL WORLD LIBRARY CODE 2

```
namespace Usb0Inten{    ///
```

EXTENTION FILE ENUM CORRECTION

```
".PORTA" : {  
  "register" : {  
    "PCR0" : {  
      "field" : {  
        "MUX" : {  
          "enum" : {  
            "000" : {".rename" : "disabled"},  
            "001" : {".rename" : "gpio" },  
            "010" : {".rename" : "reserved"},  
            "011" : {".rename" : "tpm0ch5"},  
            "100" : {".rename" : "reserved"},  
            "101" : {".rename" : "reserved"},  
            "110" : {".rename" : "reserved"},  
            "111" : {".rename" : "swdClk"}  
          }  
        }  
      }  
    }  
  }  
}
```

EXTENTION FILE ENUM CORRECTION

```
"kvasir" : {  
  "io" : {  
    "default" : { "type" : "group", "ports" : "A-E",  
                  "peripheral" : "GPIO%s"},  
    "output" : {"value" : 1, "register" : "PDDR"},  
    "input" : {"value" : 0, "register" : "PDDR"},  
    "set" : {"value" : 1, "register" : "PSOR"},  
    "clear" : {"value" : 1, "register" : "PCOR"},  
    "toggle" : {"value" : 1, "register" : "PTOR"},  
    "read" : {"register" : "PDIR"}  
  }  
}
```


GPIO IN KVASIR

```
namespace Io = Kvasir::Io;
// **** pin locations for FRDM-KL27Z board ****
constexpr auto ledr = makePinLocation(Io::portB, Io::pin18);
constexpr auto ledb = makePinLocation(Io::portA, Io::pin13);
constexpr auto ledg = makePinLocation(Io::portB, Io::pin19);
constexpr auto adc_input = makePinLocation(Io::portB, Io::pin2);

// **** pure laziness ****
namespace k = Kvasir;
constexpr auto pinAssignment = list(makeOutput(ledr, ledb, ledg),
    makeInput(adc_input),
    write(k::PortaPcr13::MuxValC::gpio,
        k::PortbPcr18::MuxValC::gpio,
        k::PortbPcr19::MuxValC::gpio,
        k::PortbPcr2::MuxValC::adc0Se12,
        k::PortbPcr3::MuxValC::gpio));

apply(pinAssignment/*probably other lists of init stuff here*/);
```

NEW PIN DEFINITION SYNTAX

```
namespace Io = Kvasir::Io::port;  
// **** pin locations for FRDM-KL27Z board ****  
constexpr auto ledr = Io::B,18_c;  
constexpr auto ledr = Io::B[18_c];  
constexpr auto ledb = 0xA'13_pin;  
constexpr auto ledg = 2.19_pin;  
constexpr auto adc_input = Io::B::pin2;
```

PORT MANIPULATION IN KVASIR

```
constexpr auto lcdPort = makePort(  
    1.1_pin, 1.2_pin, 1.3_pin, 1.4_pin, 1.5_pin,  
    1.6_pin, 1.9_pin, 1.10_pin);  
  
apply(write(lcdPort, 10));  
apply(write(lcdPort, 10_c));  
  
struct PrintToLcdFinished {};  
  
auto myLcd = LCD::make(myQueue, lcdPort, LCD::rst = 2.7_pin,  
    LCD::rw = 2.8_pin, LCD::onWritten<PrintToLcdFinished>);  
  
myLcd.put('c');
```

PORT MANIPULATION 2

```
struct PrintToLcdFinished {};  
auto queue = Queue::make(  
    list(PrintToLcdFinished{}, LCD::onTimerEvent),  
    time::make(time::hw = timer0));  
  
auto myLcd = LCD::make(myQueue, lcdPort, LCD::rst = 2.7_pin, LCD::rw = 2.8_pin,  
    LCD::onWritten<PrintToLcdFinished>);  
  
auto str = lazyf("hello {} world{}", strf("formatted"), charf('!'));  
auto r = range(str);  
  
void callback(PrintToLcdFinished&) { if (r) { r = copy(r, myLcd.tx); } }  
void postTimeEv(LCD::OnTimerEvent& e) { LSM::processEvent(myLcd, e); }  
  
while (1) {  
    using namespace Queue;  
    processEv(queue, list(  
        callback<PrintToLcdFinished, callback>{},  
        callback<LCD::OnTimerEvent, postTimeEv>{}));  
}
```

ATOMIC / ISOLATED

```
using namespace Uart1;  
  
apply(atomic(write(cfgDataBValC::eight, CfgStopBValC::one)));  
  
apply(isolated(write(cfgDataBValC::eight, CfgStopBValC::one)));
```

ATOMIC TRICKS

- ▶ Bit Manipulation Engine
- ▶ Bit Banding
- ▶ Blind write is possible if all bits between two byte boundaries are either written to or have an ignored value
- ▶ Idrex strex, the CAS of cortex
- ▶ Dedicated toggle register

UNIT TESTING

//in library default is exec

```
template <typename T, typename U>  
struct ExecuteSeam : Detail::RegisterExec<T>{};
```

//user can override using specialization

```
namespace Kvasir::Register
```

```
{
```

```
    template <typename T>
```

```
    struct ExecuteSeam<T, ::Kvasir::Tag::User> : RecordActions<T>
```

```
    {
```

```
    };
```

```
}
```

CODE REVIEW

```
//deep in some header
#define LPUART_C_ADDR_REG *(volatile unsigned*) 0x40011004
#define LPUART_C_STAT_REG *(volatile unsigned*) 0x4001100C
//somewhere else in some header
#define C_ADDRESS_MASK 0x1F000000
#define C_ADDRESS_OFFSET 16
#define C_RX_FLAG_MASK (1<<7)

void f(){
//...
static int* stack = (int*)malloc(sizeof(int)*100);
stack[top++] = LPUART_C_ADDR_REG & C_ADDRESS_MASK >> C_ADDRESS_OFFSET;
LPUART_C_STAT_REG |= C_RX_FLAG_MASK;
//...
}
```

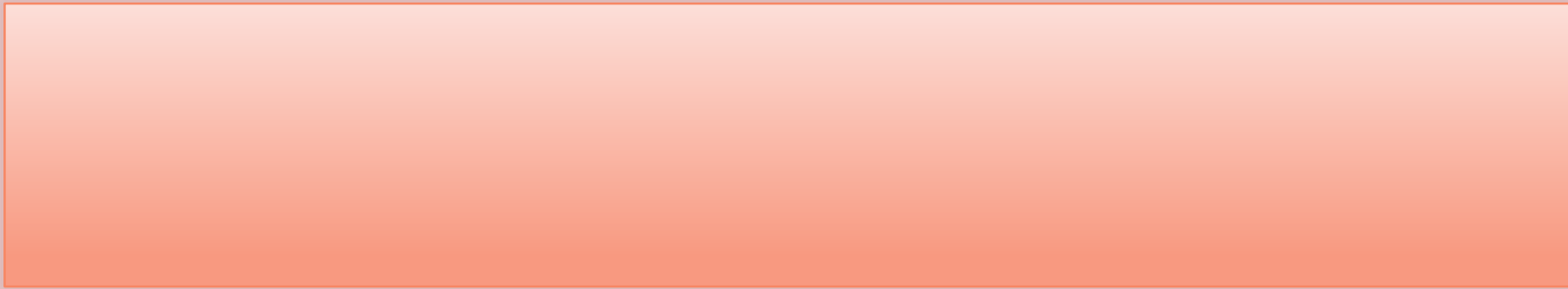

MEALTIME POLICY INSTANTIATED

CODE REVIEW

```
//deep in some header
#define LPUART_C_ADDR_REG *(volatile unsigned*) 0x40011004
#define LPUART_C_STAT_REG *(volatile unsigned*) 0x4001100C
//somewhere else in some header
#define C_ADDRESS_MASK 0x1F000000
#define C_ADDRESS_OFFSET 16
#define C_RX_FLAG_MASK (1<<7)

void f(){
//...
static int* stack = (int*)malloc(sizeof(int)*100);
stack[top++] = LPUART_C_ADDR_REG & C_ADDRESS_MASK >> C_ADDRESS_OFFSET;
LPUART_C_STAT_REG |= C_RX_FLAG_MASK;
//...
}
```

HEAP FRAGMENTATION



HEAP FRAGMENTATION



Stack

The diagram consists of a horizontal bar divided into two sections. The left section is a dark gray rectangle labeled 'Stack'. The right section is a larger, light orange rectangle representing the heap memory area.

HEAP FRAGMENTATION



SINGLETON

- ▶ Generally used as a global instance
- ▶ Violate the single responsibility principle
- ▶ Inherently cause code to be tightly coupled
- ▶ Carry around state for the lifetime of the application

MOCABLE SINGELTON

```
template<typename T, typename U>  
struct wrapped_static_seam { using type = T; };
```

```
template<typename T, typename U>  
struct wrapped_static { static T val_; using type = T; };
```

```
template<typename T, typename U>  
T wrapped_static<T, U>::val_;
```

```
template<typename T, typename U>  
auto operator*(const wrapped_static<T, U> t)->  
    typename wrapped_static_seam<T, U>::type& {  
    return wrapped_static<typename wrapped_static_seam<T,  
        U>::type,U>::val_;  
}
```

MOCABLE SINGLETON USER CODE

```
//user header
struct MyIntTag{};
constexpr wrapped_static<int, MyIntTag> myInt{};

///user unit test
struct int_moc {
    int val_;
    int_moc& operator=(int i) {
        val_ = i;
        return *this;
    };
};
template<>
struct wrapped_static_seam<int, MyIntTag> {
    using type = int_moc; //replace type with another allows mocing
};
```


CODE REVIEW

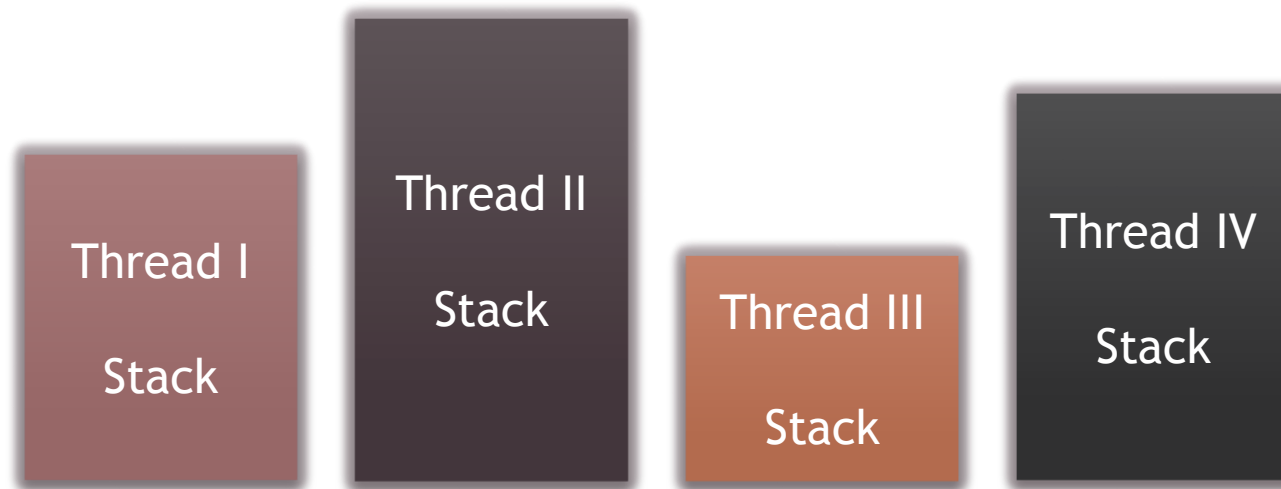
```
//deep in some header
#define LPUART_C_ADDR_REG *(volatile unsigned*) 0x40011004
#define LPUART_C_STAT_REG *(volatile unsigned*) 0x4001100C
//somewhere else in some header
#define C_ADDRESS_MASK 0x1F000000
#define C_ADDRESS_OFFSET 16
#define C_RX_FLAG_MASK (1<<7)

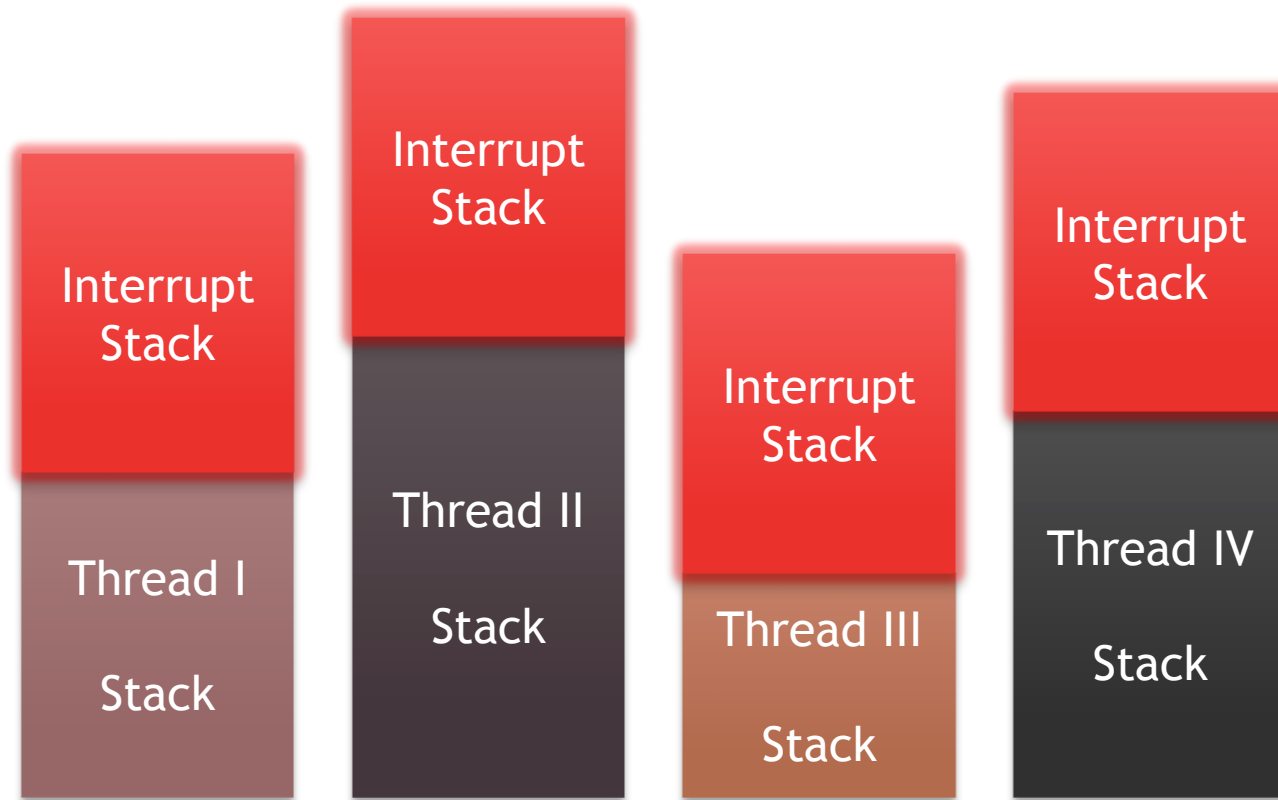
void f(){
//...
static int* stack = (int*)malloc(sizeof(int)*100);
stack[top++] = LPUART_C_ADDR_REG & C_ADDRESS_MASK >> C_ADDRESS_OFFSET;
LPUART_C_STAT_REG |= C_RX_FLAG_MASK;
//...
}
```

MAXIMUM STACK DEPTH

- ▶ Most use “guess and pray”
- ▶ Absint can calculate this
- ▶ Python script can calculate this
- ▶ Coding style can confuse tools
- ▶ Best trick is to grow into unused address space

MAXIMUM STACK DEPTH





RTC vs. SCHEDULER

- ▶ Deterministic
- ▶ Efficient
- ▶ Different coding style
- ▶ Long running processes a problem
- ▶ Far far fewer race conditions
- ▶ Clear when sleep/deep sleep possible
- ▶ Single stack allows for tricks

RTC vs. SCHEDULER

If you cannot use Linux, use a dead simple event loop. Complexity kills.

-- Niklas Hauser
(RTOS programmer at ARM)

NAMED PARAMETERS

```
namespace p2 = parameter2;  
constexpr auto length = p2::make_tag<1>(4);  
constexpr p2::tag<2, convertible<int>, integral_constant<int, 9>> height{};
```

```
struct DepthMaker{  
    int operator()() {return 42;}  
};
```

```
constexpr p2::tag<3, p2::is<int>, DepthMaker> depth{};
```

```
template<typename T>  
struct is_magic_thing : false_type {};  
template<typename...T>  
struct is_magic_thing<list<magic, T...>> : true_type {};
```

```
constexpr p2::tag<3, is_magic_thing<brigand::_1>, magic_maker> magic{};
```

NAMED PARAMETERS

```
template<typename...Ts>
void draw(Ts&&...args) {
    //make tuple uses D style syntax, compile time values as first arg list
    //and runtime in the second arg list first arg list must contain all
    //name tags in the desired positional order second list must be the args
    //passed by the user
    auto t = p2::make_tuple(length,height,depth)(std::forward<Ts>(args)...);
    auto h = t[height]; //input parameter indexing is trivial
    auto& l = t[length];
    l = 99;
    auto l2 = t[length];
    auto d = t[depth];
}
```


POLICY BASED FACTORIES

```
namespace clk = System::clock;
using namespace clk::tags;
auto clockMaster = clk::make(source = clk::ext, //external
    exFreq = 12'000'000_c, //12 mhz
    inFreq = 48'000'000_c); //48 mhz
auto myTimer = timeServer::make(clockMaster, //compile time clock settings
    hw = timer1::peripheral, //which timer do we use
    memory::policy::singleton); //how do we get our RAM
auto myUart = uart::make(bustiming, baud = 9600_c, autobaudCapability);
auto myPool = memory::pool::make(packetSize = 100_c, poolSize = 100_c);
auto myIODevice = IIODevice::make(hw = myUart,
    timer = myTimer, //data must be sent in chunks if larger than the
    //output buffer so we need a callback timer
    queue = memory::queue::intrusive::fifo::make(myPool, 1_c));

copy("hello world", myIODevice::tx);
```

REAL WORLD USB CODE

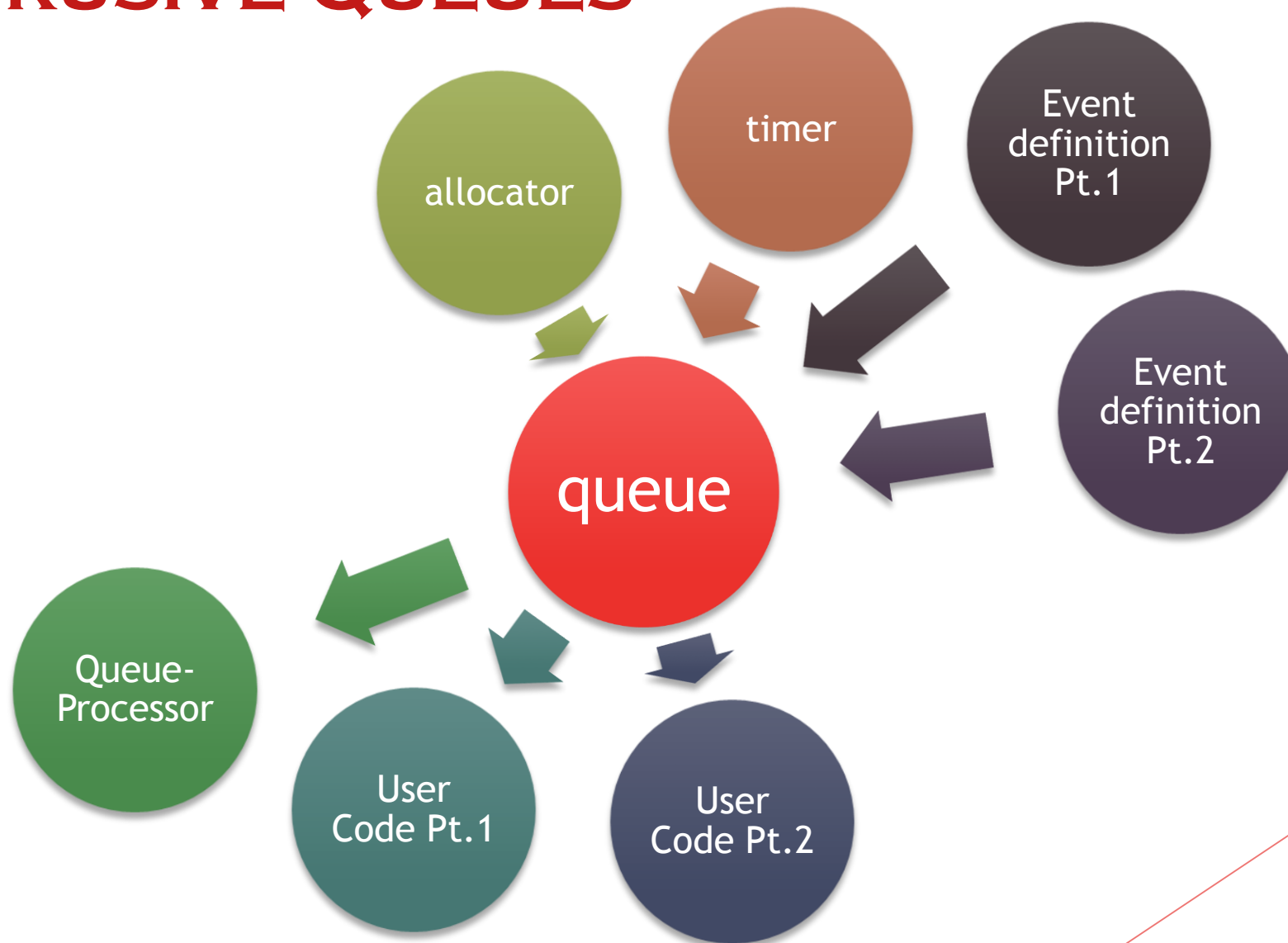
```
int MyBuffPolicy::size_{0};
uint8_t MyBuffPolicy::buf_[8];
struct MyCdcSettings : Kvasir::Usb::Cdc::DefaultSettings{
    using BufferPolicy = MyBuffPolicy;
};
struct MyDeviceSettings : Kvasir::Usb::DefaultDeviceSettings{
    using MemoryPolicy = MyMemoryPolicy;
    using StringDescriptors = MyStringDescriptors;
    static constexpr uint16_t vid = 0x1F00;
    static constexpr uint16_t pid = 0x2012;
};
using Device = Kvasir::Usb::Device<MyDeviceSettings, MyCdcSettings>;
using Hal = Kvasir::Usb::Hal<Device>;

extern "C" void USB0_IRQHandler() {
    Hal::isr();
}
```

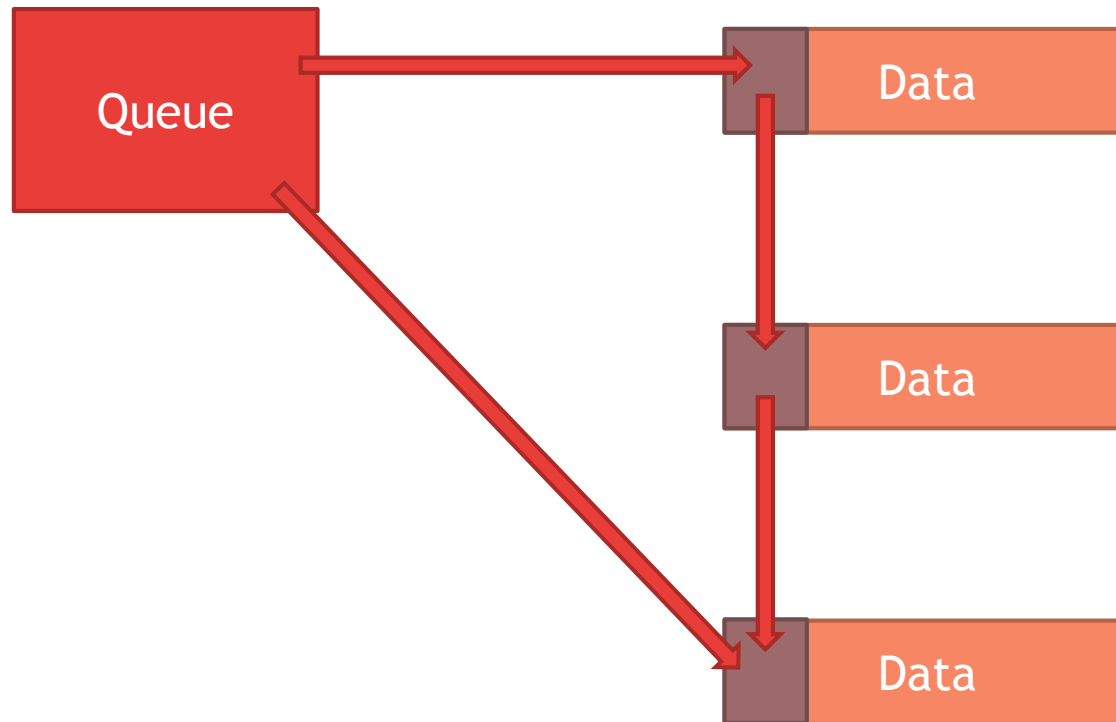
FLEXIBLE USB STACK

- ▶ Supports multiple device classes
- ▶ Supports any memory allocation policy
- ▶ Fully interrupt driven
- ▶ Fully statically linked
- ▶ Deep sleep persistent memory placement possible
- ▶ Simple minimal HAL interface
- ▶ Movable
- ▶ ~3x smaller than mbed, better RAM footprint

INTRUSIVE QUEUES



GLOBAL PRIORITY QUEUE



LAMBDA STATE MACHINE

- ▶ Event based coding style can be hard
- ▶ Protothreads are essentially a state machine
- ▶ Boost.MSM-lite is super powerful
- ▶ Most real SMs have long transition chains

LAMBDA STATE MACHINE 2

- ▶ Most heap based allocations follow SM patterns
- ▶ No fragmentation penalty in SMs
- ▶ Whole pools can be data members of parent states
- ▶ Certain states can be deep sleep capable allocating memory accordingly
- ▶ Multicore with message passing possible
- ▶ SM nesting for orthogonal regions possible

LSM CODE

```
transition("Uninitialized"_s, "Initialized"_s,  
    guard = event == init, reset,  
    write(Config::port, 2_c), toggleEnable, resetWait,  
    //function set  
    write(Config::port, Config::functionSetH), toggleEnable,  
    write(Config::port, Config::functionSetL), toggleEnable, resetWait,  
    //turn off display  
    write(Config::port, 0_c), toggleEnable,  
    write(Config::port, 8_c), toggleEnable, resetWait,  
    //clear display  
    write(Config::port, 0_c), toggleEnable,  
    write(Config::port, 1_c), toggleEnable, resetWait,  
    //entry mode  
    write(Config::port, Config::entryModeH), toggleEnable,  
    write(Config::port, Config::entryModeL), toggleEnable, resetWait,  
    //turn on display  
    write(Config::port, 0_c), toggleEnable,  
    write(Config::port, 8_c), toggleEnable, resetWait  
)
```


LSM CODE 2

```
auto enableWait = chain(
    [](auto& context) { context.enableWait(); },
    guard = event == timer
);

auto resetWait = chain(
    [](auto& context) { context.resetWait(); },
    guard = event == timer
);

auto toggleEnable = chain(
    enableWait,
    set(Config::enable),
    enableWait,
    clear(Config::enable),
    enableWait
);
```

QUESTIONS?



Odin Holmes

holmes@auto-intern.de

