Distributed Systems

Inga Rüb

17 October 2018

TinyOS

TinyOS

Berkeley
UNIVERSITY OF CALIFORNIA

TinyOS

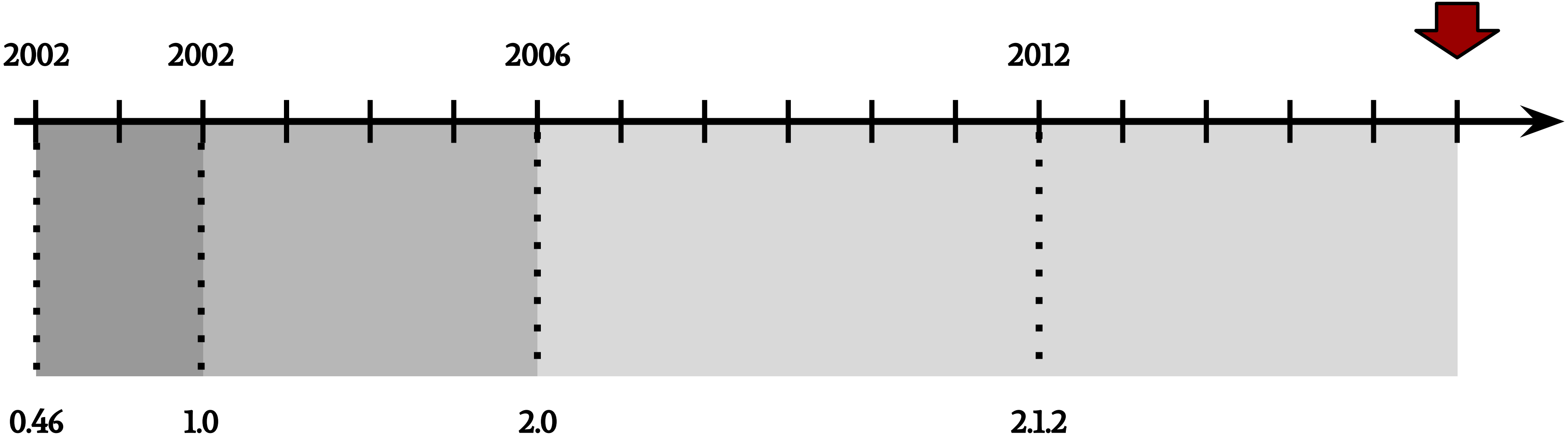
Berkeley
UNIVERSITY OF CALIFORNIA

DARPA

intel

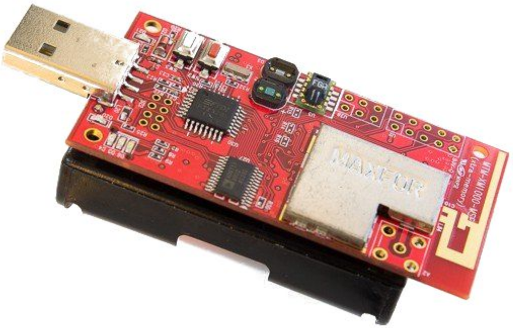
Crossbow

TinyOS - development



TinyOS - the idea

a **lightweight** operating system specifically designed for:



resource-limited

low-power

wireless

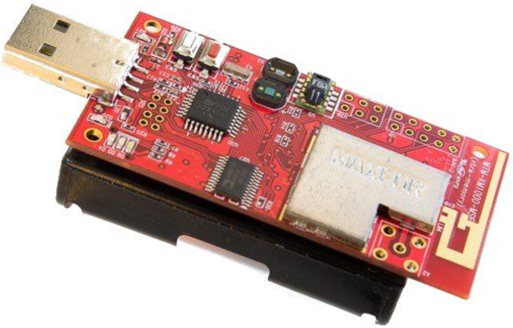
TinyOS - the idea

a **lightweight** operating system specifically designed for:

processor: **25 MHz**

512 kB flash memory

10 kB RAM



resource-limited

low-power

wireless

TinyOS - the idea

services & abstractions

TinyOS - the idea

demand for RAM

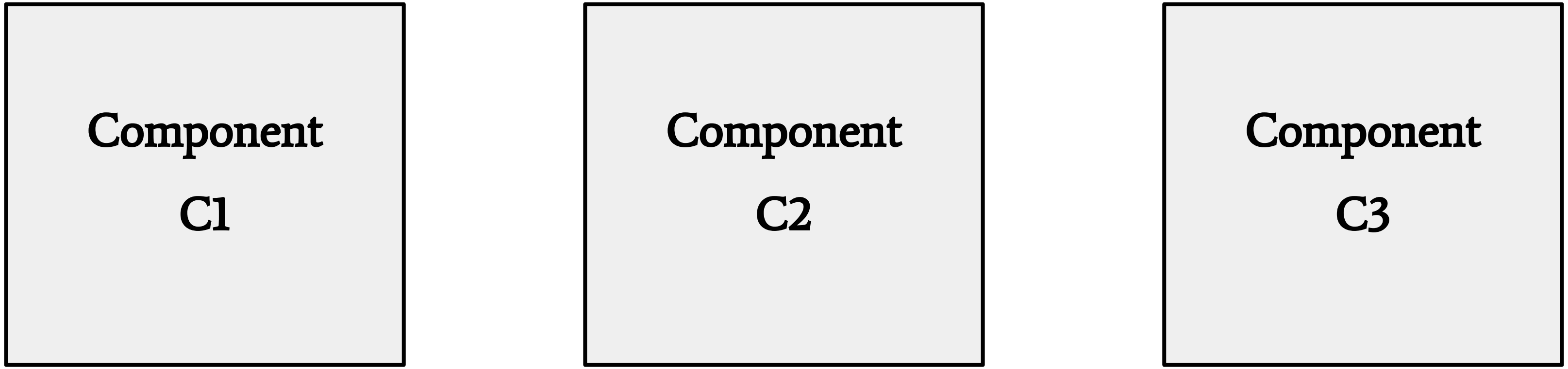
code size

services & abstractions

optimizations

debugging

A nesC application

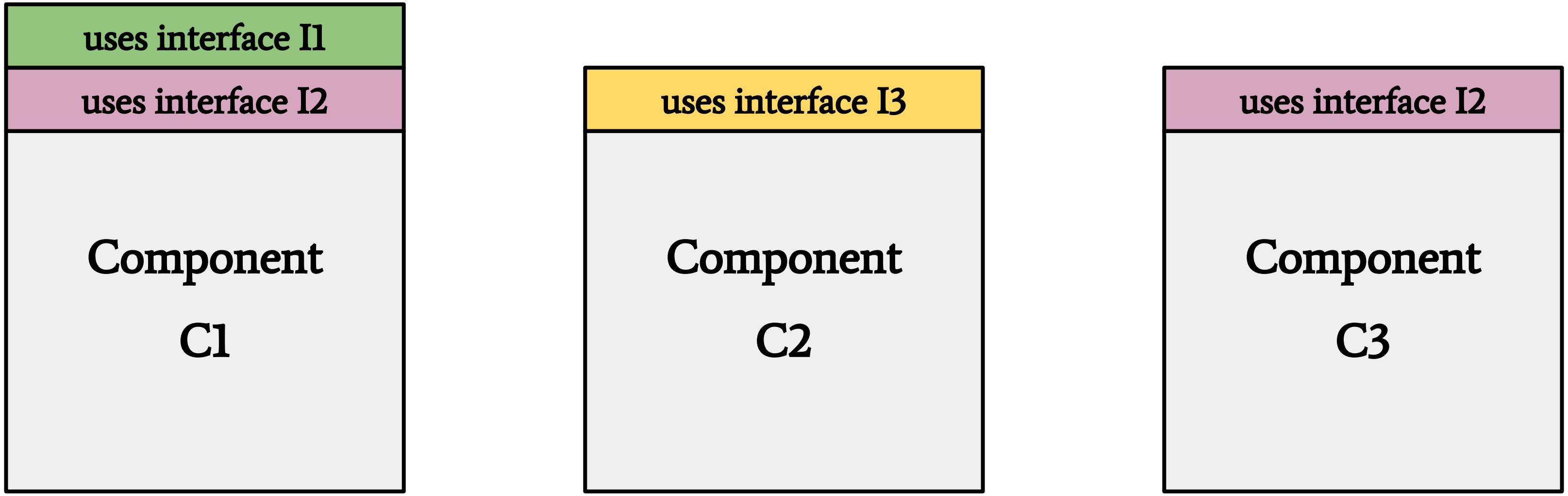


Component C1

Component C2

Component C3

A nesC application



uses interface I1

uses interface I2

Component C1

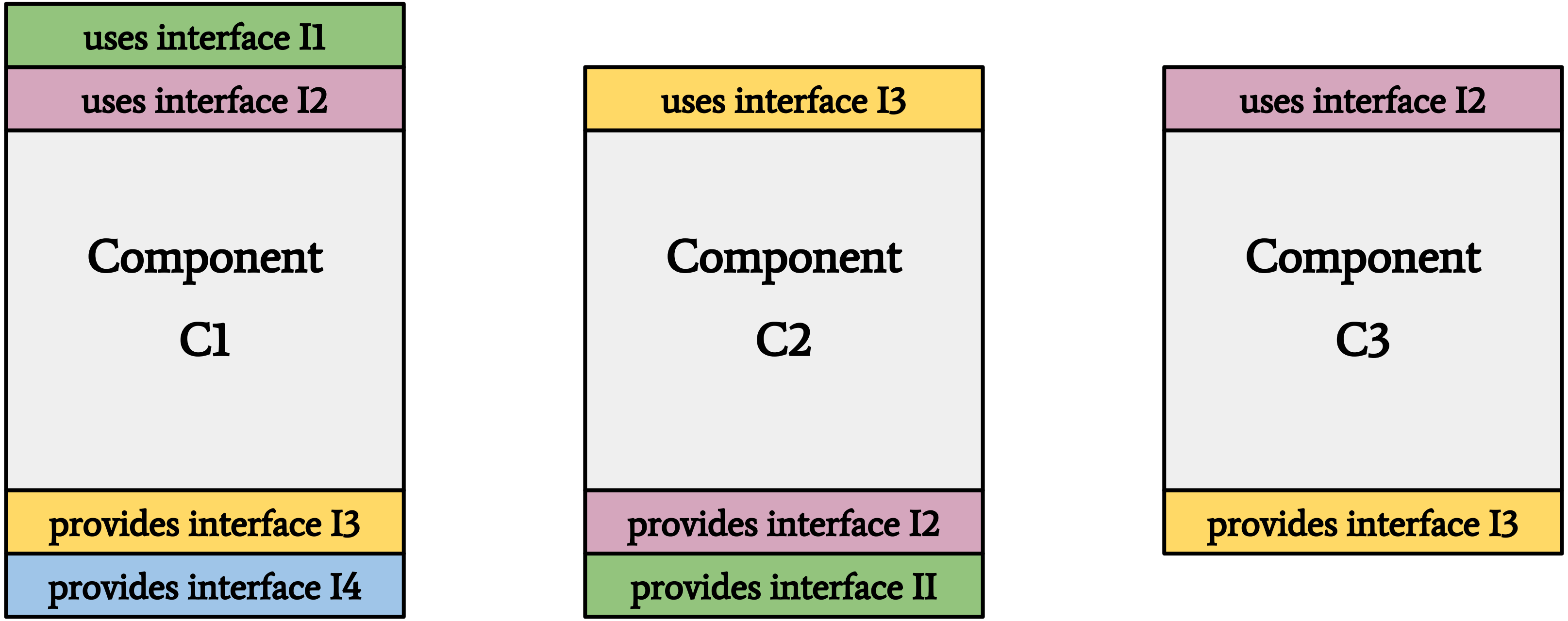
uses interface I3

Component C2

uses interface I2

Component C3

A nesC application



uses interface I1

uses interface I2

Component

C1

provides interface I3

provides interface I4

uses interface I3

Component

C2

provides interface I2

provides interface II

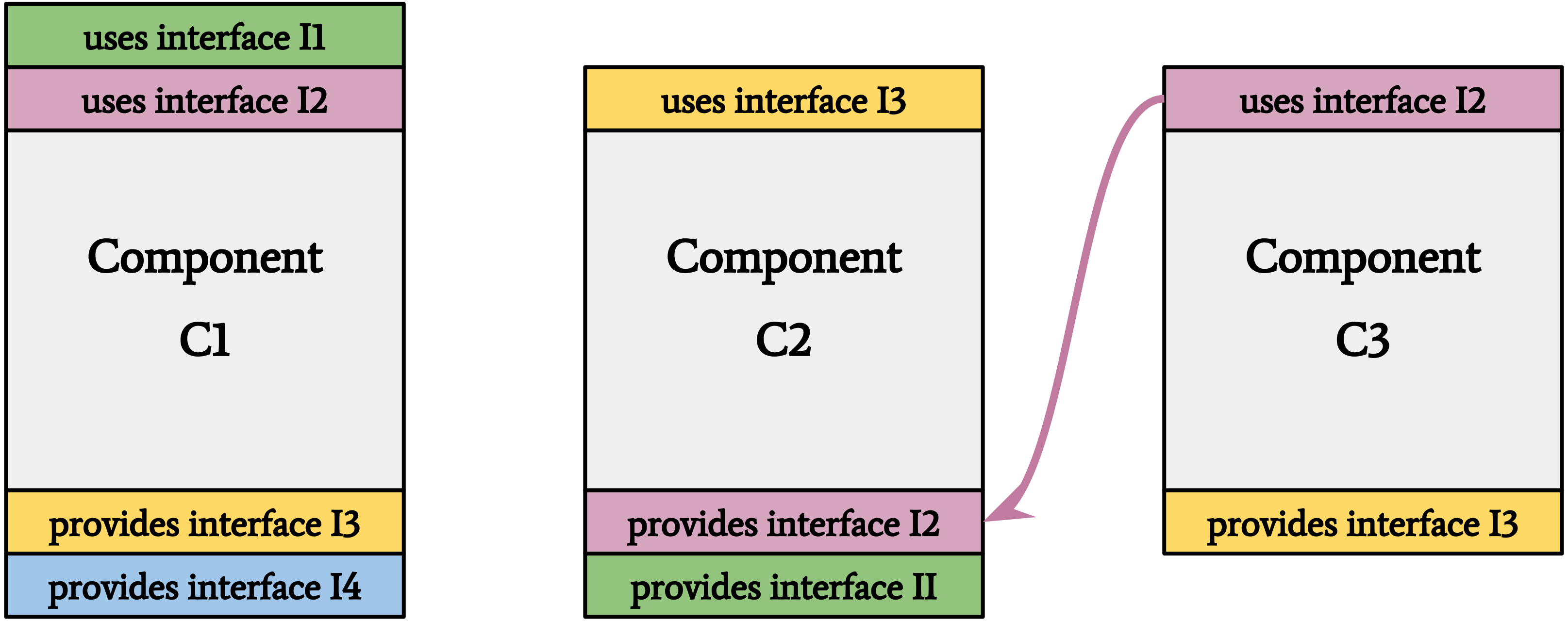
uses interface I2

Component

C3

provides interface I3

A nesC application



uses interface I1

uses interface I2

Component

C1

provides interface I3

provides interface I4

uses interface I3

Component

C2

provides interface I2

provides interface II

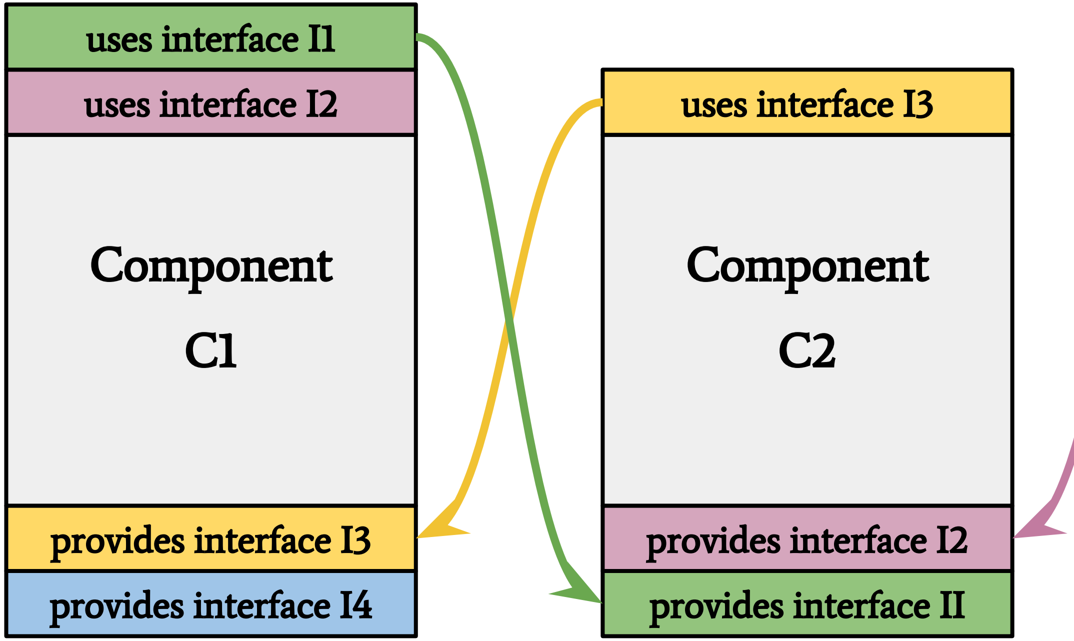
uses interface I2

Component

C3

provides interface I3

A nesC application



uses interface I1

uses interface I2

Component

C1

provides interface I3

provides interface I4

uses interface I3

Component

C2

provides interface I2

provides interface II

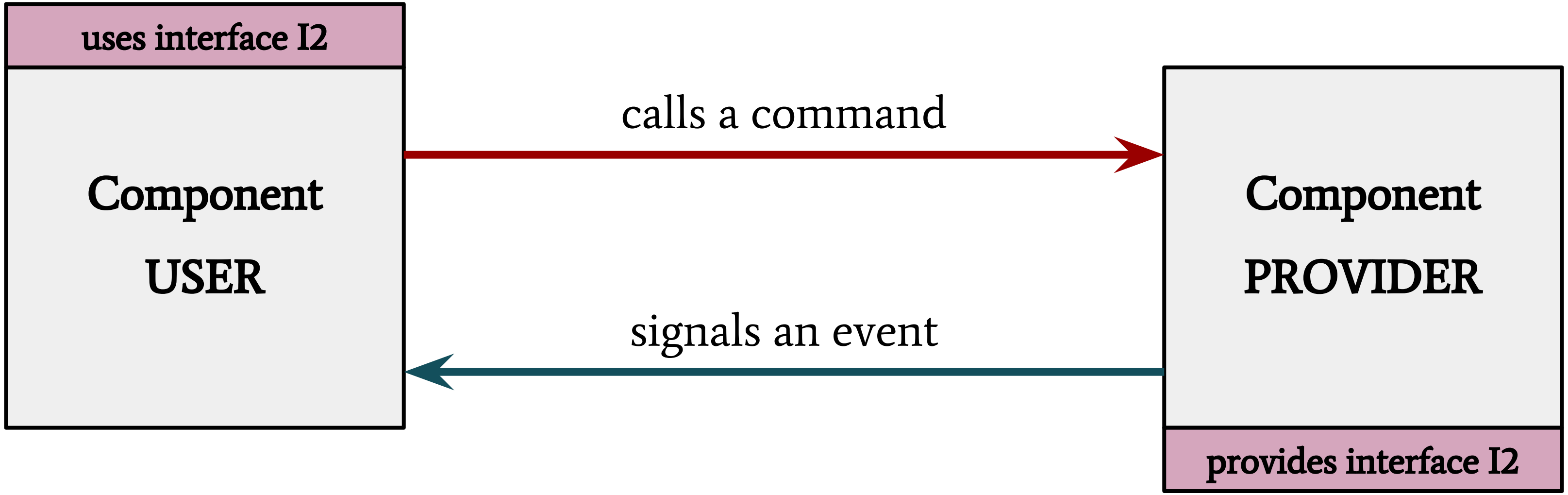
uses interface I2

Component

C3

provides interface I3

Interfaces



uses interface I2

Component

USER

calls a command

signals an event

Component

PROVIDER

provides interface I2

Interfaces - events

INTERFACE

```
/**
  * Interface that notifies
  * components when TinyOS has
booted.
  */

interface Boot {
  event void booted();
}
```

whip6-pub/nesc/whip6/api/sys/boot/Boot.nc

Interfaces - events

INTERFACE

```
/**
  * Interface that notifies
  * components when TinyOS has
booted.
  */

interface Boot {
  event void booted();
}
```

PROVIDER

```
...
signal Boot.booted();
...
```

whip6-pub/nesc/whip6/api/sys/boot/Boot.nc

Interfaces - events

INTERFACE

```
/**
  * Interface that notifies
  * components when TinyOS has
booted.
  */

interface Boot {
  event void booted();
}
```

PROVIDER

```
...
signal Boot.booted();
...
```

USER

```
event void Boot.booted() {
...
}
```

whip6-pub/nesc/whip6/api/sys/boot/Boot.nc

Interfaces - commands

INTERFACE

```
/**
  * Interface that provides
  * functionality of a led.
  */

interface Led {
  command void on();
  command void off();
  command void set(bool on);
  command bool isOn();
  command void toggle();
}
```

`whip6-pub/nesc/whip6/api/diagnostic/Led.nc`

Interfaces - commands

INTERFACE

```
/**
  * Interface that provides
  * functionality of a led.
  */

interface Led {
  command void on();
  command void off();
  command void set(bool on);
  command bool isOn();
  command void toggle();
}
```

PROVIDER

```
command void Led.set(bool on) {
    ...

}
```

whip6-pub/nesc/whip6/api/diagnostic/Led.nc

Interfaces - commands

INTERFACE

```
/**
  * Interface that provides
  * functionality of a led.
  */

interface Led {
  command void on();
  command void off();
  command void set(bool on);
  command bool isOn();
  command void toggle();
}
```

PROVIDER

```
command void Led.set(bool on) {
    ...

}
```

USER

```
...
call Led.set(true);
...
```

whip6-pub/nesc/whip6/api/diagnostic/Led.nc

Interfaces - commands

INTERFACE

```
/**
  * Interface that provides
  * functionality of a led.
  */

interface Led {
  command void on();
  command void off();
  command void set(bool on);
  command bool isOn();
  command void toggle();
}
```

PROVIDER

```
command void Led.set(bool on) {
   if (on) { call Led.on(); }
   else { call Led.off(); }
}
```

USER

```
...
call Led.set(true);
...
```

whip6-pub/nesc/whip6/api/diagnostic/Led.nc

Interfaces - let’s design one

```
interface ReadNow {






}
```

Interfaces - let’s design one

```
interface ReadNow {
  /**
   * Reads a value.
   *
   * @return the value
   */
  command uint8_t read();
}
```

Interfaces - let’s design one

```
interface ReadNow {
  /**
   * Initiates a read of a value.
   *
   * @return SUCCESS if a readDone() event will eventually come back.
   */
  command error_t read();

  /**
   * Signals the completion of the read().
   *
   * @param result SUCCESS if the read() was successful
   * @param val the value that has been read
   */
  event void readDone(error_t result, uint8_t val);
}
```

a split-phase interface

Interfaces - let’s design one

```
interface ReadNow<val_t> {
  /**
   * Initiates a read of a value.
   *
   * @return SUCCESS if a readDone() event will eventually come back.
   */
  command error_t read();

  /**
   * Signals the completion of the read().
   *
   * @param result SUCCESS if the read() was successful
   * @param val the value that has been read
   */
  event void readDone(error_t result, val_t val);
}
```

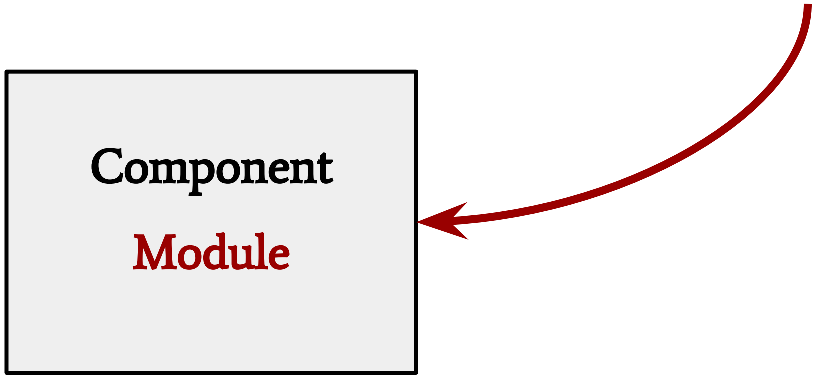
a generic interface

Components

There are two kinds of components:

Components

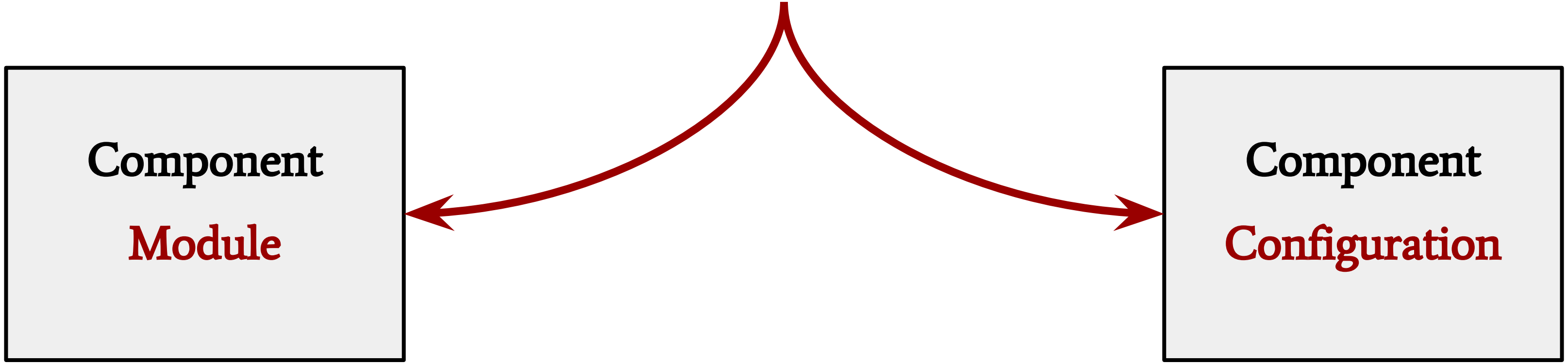
There are two kinds of components:



implements interfaces

Components

There are two kinds of components:



Component

Module

Component

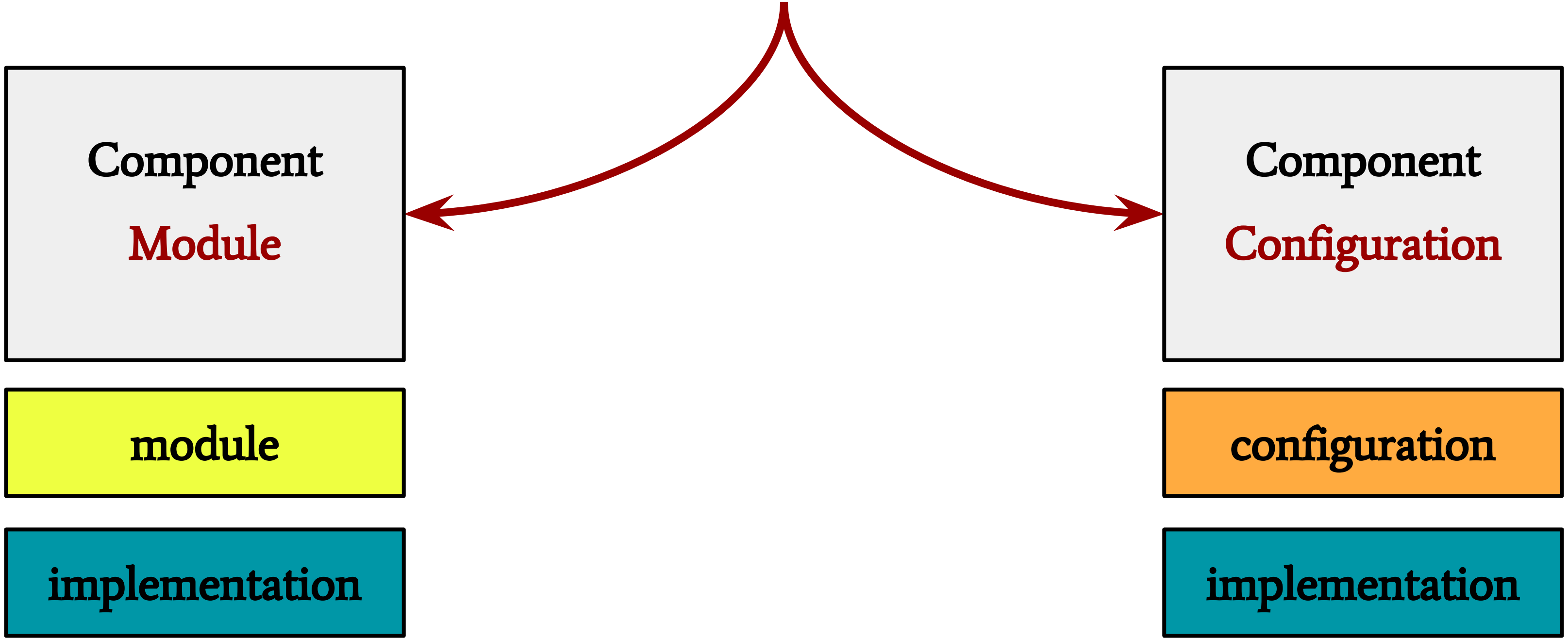
Configuration

implements interfaces

wires components
with each other

Components

All components consist of two parts:



Component

Module

module

implementation

Component

Configuration

configuration

implementation

Module (a component)

```
module PowerupC {
    uses interface Boot;
    uses interface Leds;
}

implementation {
    event void Boot.booted() {
        call Leds.led0On();
    }
}
```

Configuration (a component)

```
configuration PowerupAppC {
    uses {
        interface Boot;
    }
}
implementation {
    components PowerupC, LedC;
    PowerupC.Boot = Boot;
    PowerupC.Leds -> LedsC.Leds;
}
```

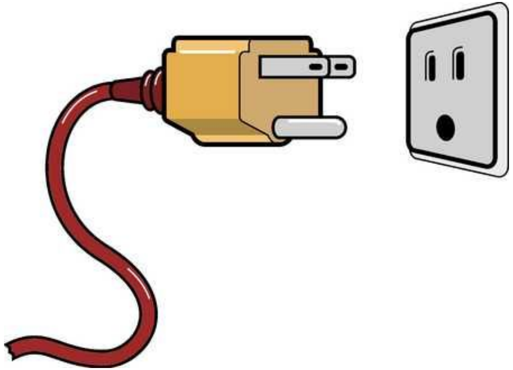
Configuration (a component)

```
configuration PowerupAppC {
    uses {
        interface Boot;
    }
}
implementation {
    components PowerupC, LedC;
    PowerupC.Boot = Boot;
    PowerupC.Leds -> LedsC.Leds;
}
```

```
configuration PowerupAppC {

}


implementation {
    components MainC, PowerupC;
    MainC.Boot <- PowerupC.Boot;
    components LedC;
    PowerupC.Leds -> LedsC.Leds;
}
```

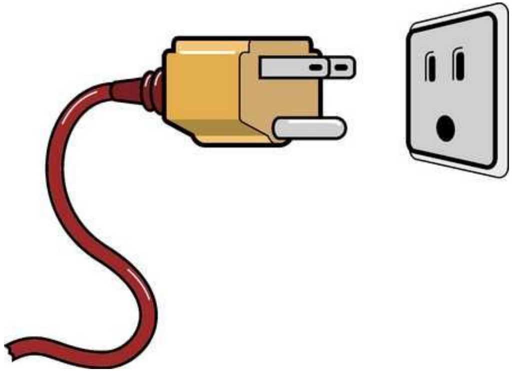
Wirings

```
    user  ->  provider

provider  <-  user
```



Wirings

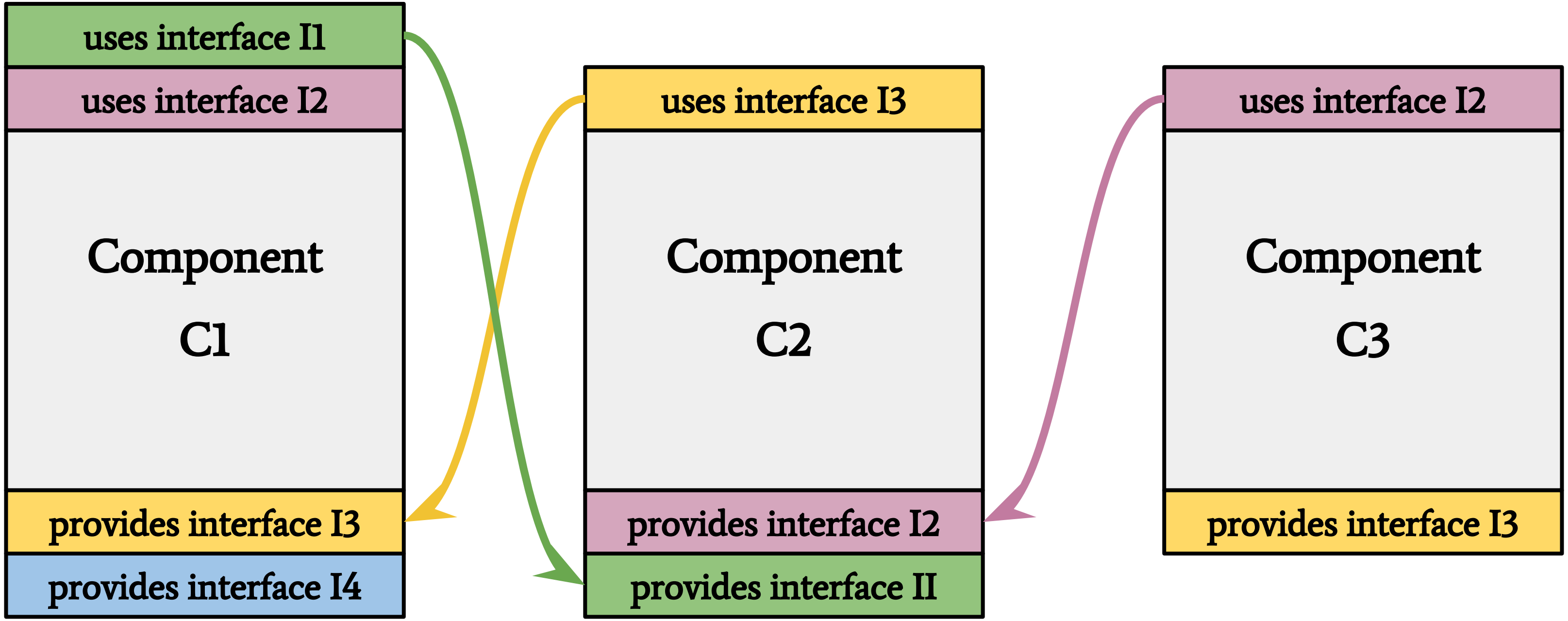
```
    user  ->  provider

provider  <-  user
```



```
interface I of component C1  =  interface I of component C2
```

exactly the same interface

A nesC application



uses interface I1

uses interface I2

Component

C1

provides interface I3

provides interface I4

uses interface I3

Component

C2

provides interface I2

provides interface II

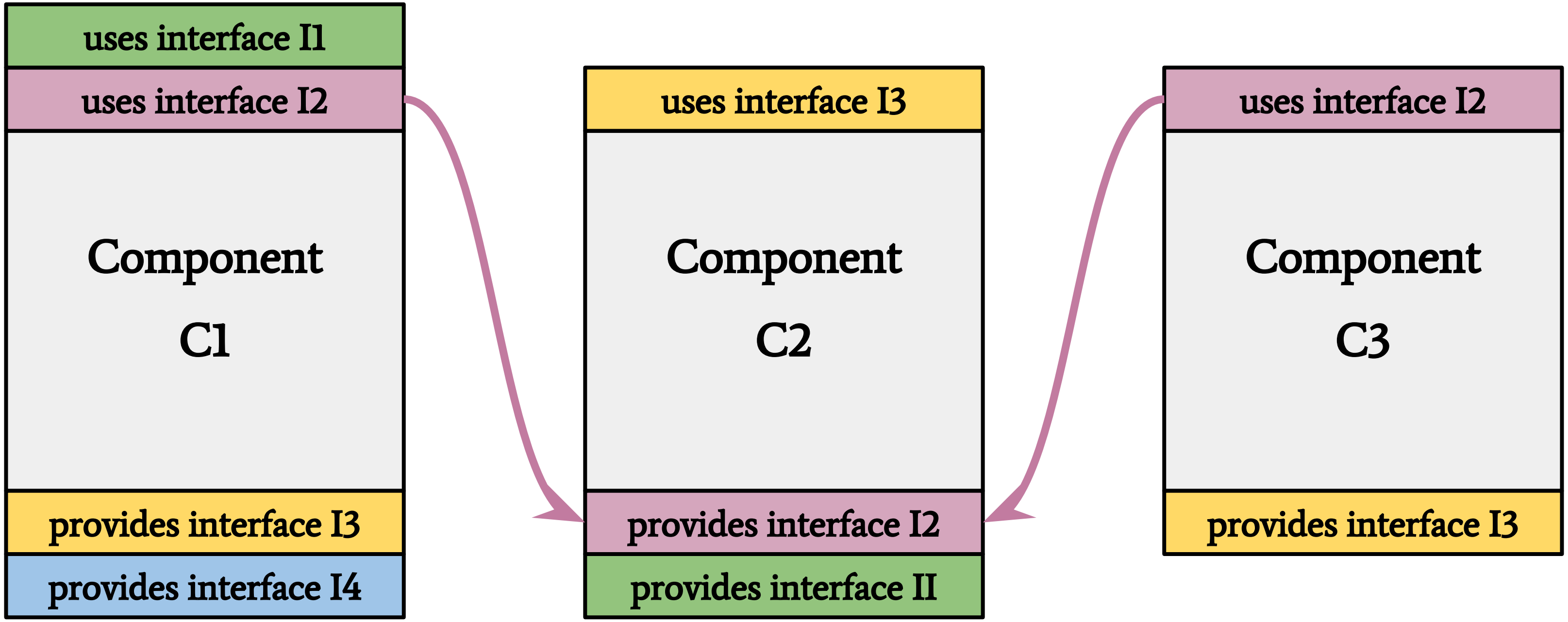
uses interface I2

Component

C3

provides interface I3

A nesC application



uses interface I1

uses interface I2

Component

C1

provides interface I3

provides interface I4

uses interface I3

Component

C2

provides interface I2

provides interface II

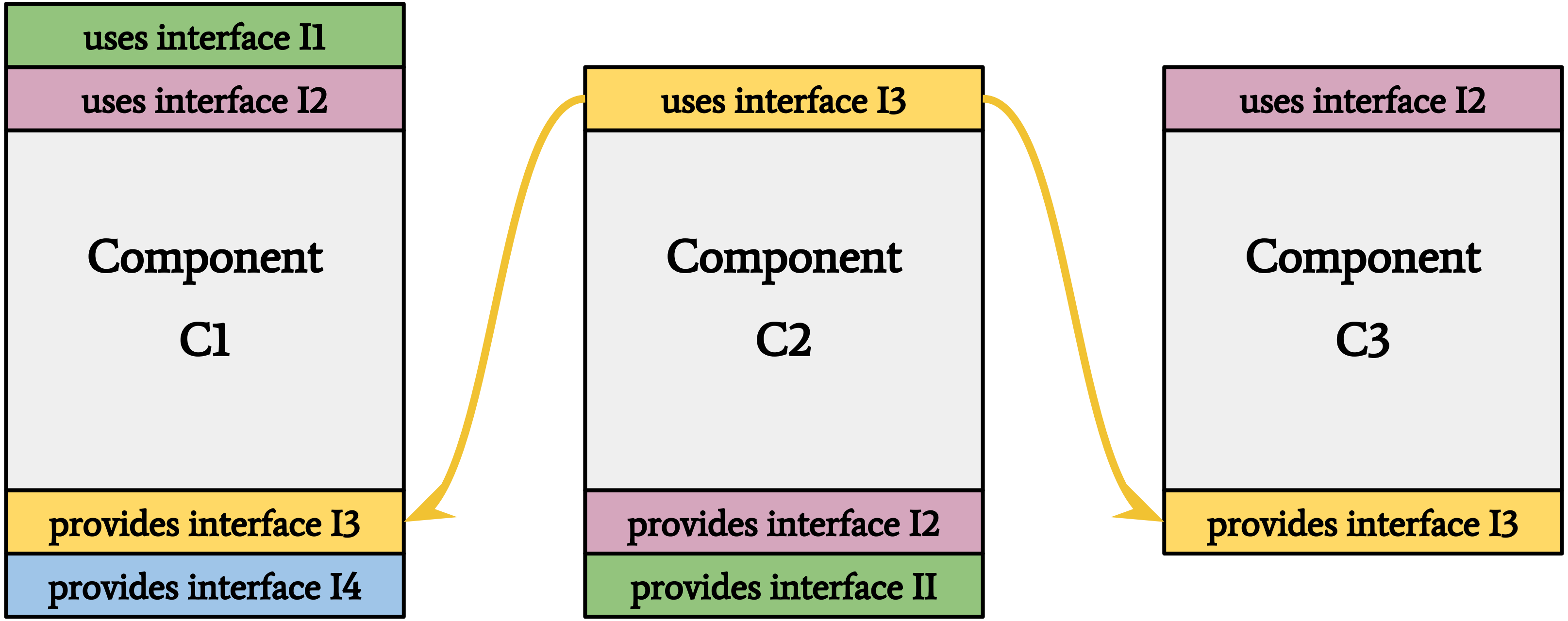
uses interface I2

Component

C3

provides interface I3

A nesC application



uses interface I1

uses interface I2

Component

C1

provides interface I3

provides interface I4

uses interface I3

Component

C2

provides interface I2

provides interface II

uses interface I2

Component

C3

provides interface I3

multiple wirings

Same interfaces, different wirings

```
module LedsP {
    provides {
        interface Init;
        interface Leds;
    }
    uses {
        interface GeneralIO as Led0;
        interface GeneralIO as Led1;
        interface GeneralIO as Led2;
    }
}
```

Same interfaces, different wirings

```
module Leds {
    provides {
        interface Led as Led0;
        interface Led as Led1;
        interface Led as Led2;
    }
    ...
}
```

Same interfaces, different wirings

```
module Leds {
    provides {
        interface Led as Led0;
        interface Led as Led1;
        interface Led as Led2;
    }
    ...
}
```

```
module Leds {
    provides {
        interface Led[uint8_t]
    }
    ...
    command bool Led.isOn[uint8_t]() { ... }
}
```

a parameterized interface

From singletons to generic components

```
generic configuration TimerMilliC() {
    provides interface Timer<TMilli>;
} implementation { … }
```

From singletons to generic components

```
generic configuration TimerMilliC() {
    provides interface Timer<TMilli>;
} implementation { … }

configuration BlinkAppC {}
    implementation {
        components MainC, BlinkC, LedsC;
        components new TimerMilliC() as Timer0;
        components new TimerMilliC() as Timer1;
        components new TimerMilliC() as Timer2;
        /* Wirings below */
} implementation { … }
```

From singletons to generic components

```
generic configuration TimerMilliC() {
    provides interface Timer<TMilli>;
} implementation { … }

configuration BlinkAppC {}
    implementation {
        components MainC, BlinkC, LedsC;
        components new TimerMilliC() as Timer0;
        components new TimerMilliC() as Timer1;
        components new TimerMilliC() as Timer2;
        /* Wirings below */
} implementation { … }

generic module QueueC(typedef queue_t, uint8_t queueSize) {
    provides interface Queue<queue_t>;
} implementation { … }
```

Module variables

Module variables

All of them are private.

Module variables

All of them are private.

They can be accessed only via interfaces.

Module variables

All of them are private.

They can be accessed only via interfaces.

```
module CountingGetC {
    provides interface Get<uint8_t>;
}

implementation {
    uint8_t count;
    command uint8_t Get.get() {
        return count++;
    }
}
```

Data shared between modules

- ★ Avoid passing pointers to memory as parameters.
- ★ Make sure only one module owns the memory at a time.

Data shared between modules

- Avoid passing pointers to memory as parameters.
- Make sure only one module owns the memory at a time.

```
interface Send {
    command error_t send(message_t* msg, uint8_t len);
    event void sendDone(message_t* msg, error_t error);
}
```

Data shared between modules

- Avoid passing pointers to memory as parameters.
- Make sure only one module owns the memory at a time.

```
interface Send {
    command error_t send(message_t* msg, uint8_t len);
    event void sendDone(message_t* msg, error_t error);
}

interface Receive {
    event message_t* receive(message_t* msg, void* payload, uint8_t len);
}
```

Data shared between modules

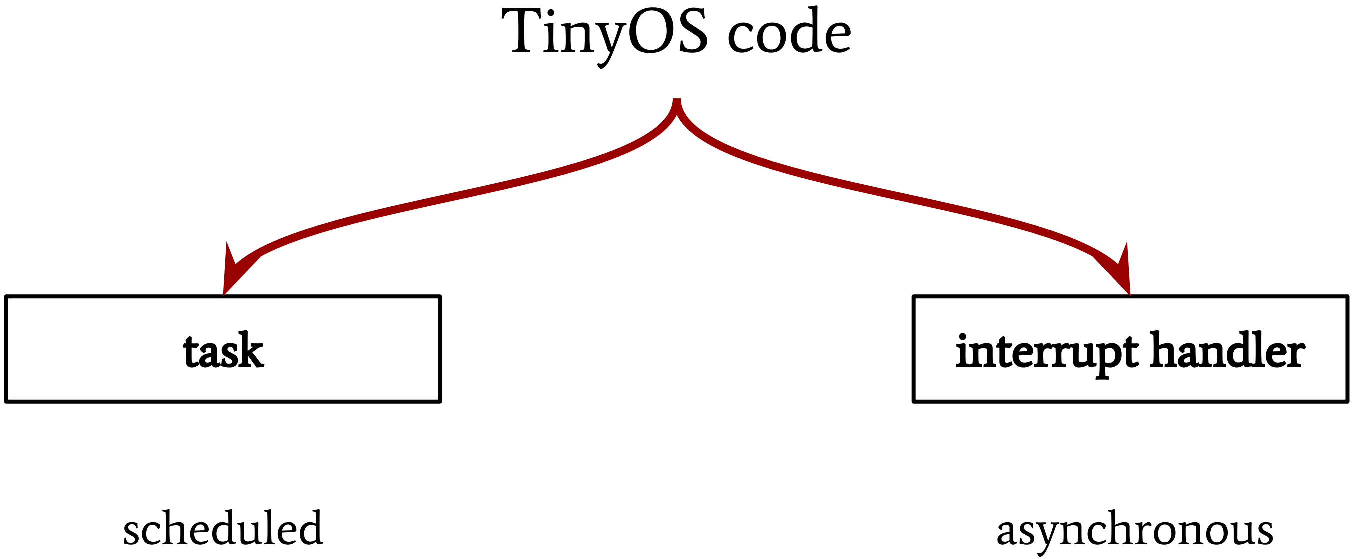
★ Avoid passing pointers to memory as parameters.

★ Make sure only one module owns the memory at a time.

```
interface Send {
    command error_t send(message_t* msg, uint8_t len);
    event void sendDone(message_t* msg, error_t error);
}


interface Receive {
    event message_t* receive(message_t* msg, void* payload, uint8_t len);
}
```

buffers are returned to original owners

The execution model



TinyOS code

task

interrupt handler

scheduled

asynchronous

Tasks

a simple deferred computation mechanism

return value of tasks is always void

tasks take no parameters

Tasks

a simple deferred computation mechanism

return value of tasks is always void

tasks take no parameters

```
task void setupTask() {
    // task code
}
```

Tasks

a simple deferred computation mechanism

return value of tasks is always void

tasks take no parameters

```
task void setupTask() {
    // task code
}
```

```
event void Boot.booted() {
    call Timer.startPeriodic(1024);
    post setupTask();
}
```

★ Use to offload computations within event handlers.

Tasks

★ Use to call commands indirectly from within event handlers.

```
event void Read.readDone(error_t err, uint16_t val) {
    buffer[index] = val;
    index++;
    if (index < BUFFER_SIZE) {
        call Read.read();        // put instead: post doRead();
    }
}
```

Why?

Interrupt handlers

allowed to preempt tasks

Interrupt handlers

allowed to preempt tasks

require keyword “`async`” for code they use

Interrupt handlers

allowed to preempt tasks

require keyword “`async`” for code they use

introduce the need for “`atomic`” mechanism

Interrupt handlers

```
bool state;

async command bool toggle () {
    if (state == 0) {
        state = 1;
        return 1;
    }
    if (state == 1) {
        state = 0;
        return 0;
    }
}
```

Interrupt handlers

```
bool state;

async command bool toggle () {
    if (state == 0) {
        state = 1;
        return 1;
    }
    if (state == 1) {
        state = 0;
        return 0;
    }
}
```

```
bool state;

async command bool toggle () {
    atomic {
        if (state == 0) {
            state = 1;
            return 1;
        }
        if (state == 1) {
            state = 0;
            return 0;
        }
    }
}
```

“`atomic`” disables interrupts

TinyOS - features

TinyOS - features

The component model - static wiring.

TinyOS - features

The component model - static wiring.

Commands and signals as a way to interact.

TinyOS - features

The component model - static wiring.

Commands and signals as a way to interact.

No threads - just a single stack.

TinyOS - features

The component model - static wiring.

Commands and signals as a way to interact.

No threads - just a single stack.

The execution model - tasks and handlers.

Constant data

★ Use enums to declare constant values.

★ Do not use enums as types for variables.

```
enum {
    SUCCESS   = 0,
    FAIL      = 1,      // Generic condition: backwards compatible
    ESIZE     = 2,      // Parameter passed in was too big.
    ECANCEL   = 3,      // Operation cancelled by a call.
    EOFF      = 4,      // Subsystem is not active
    EBUSY     = 5,      // The underlying system is busy; retry later
    EINVAL    = 6,      // An invalid parameter was passed
    ERETRY    = 7,      // A rare and transient failure: can retry
    ERESERVE  = 8,      // Reservation required before usage
    EALREADY  = 9,      // The device state you are requesting is already set
};

typedef uint8_t error_t;
```

Useful resources

Philip Levis, David Gay: *TinyOS Programming*

wiki TinyOS: http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Overview

TEPs: http://tinyos.stanford.edu/tinyos-wiki/index.php/TEPs

for vim: http://www.vim.org/scripts/script.php?script_id=1847