TINYOS

Prepared By,

SHELLY SHIJU GEORGE

ASSISTANT PROFESSOR

TinyOS

TinyOS

- TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters.
- TinyOS is an embedded, component-based operating system and platform for low-power wireless devices, such as those used in wireless sensor networks (WSNs), smartdust, ubiquitous computing, personal area networks, building automation, and smart meters.
- It is written in the programming language nesC, as a set of cooperating tasks and processes.

NEED OF TinyOS

Problems with traditional OS –

- Multithreaded Architecture not useful
- Large Memory Footprint
- Does not help to conserve energy and power

Requirements for Wireless Sensor Network –

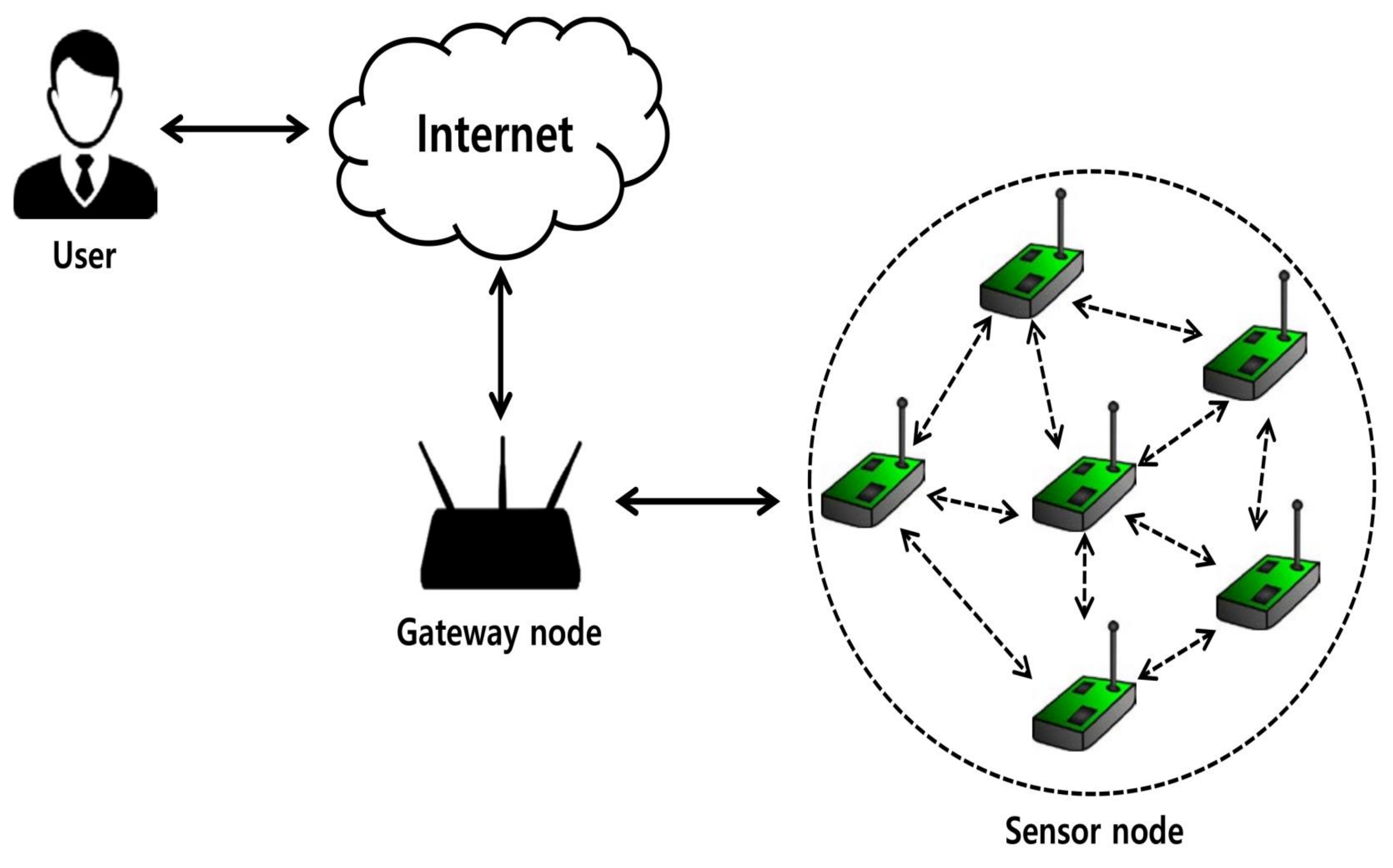
- Efficient utilization of energy and power
- Small Footprint and support diversity in design usage

NEED OF TinyOS (CONTINUED)

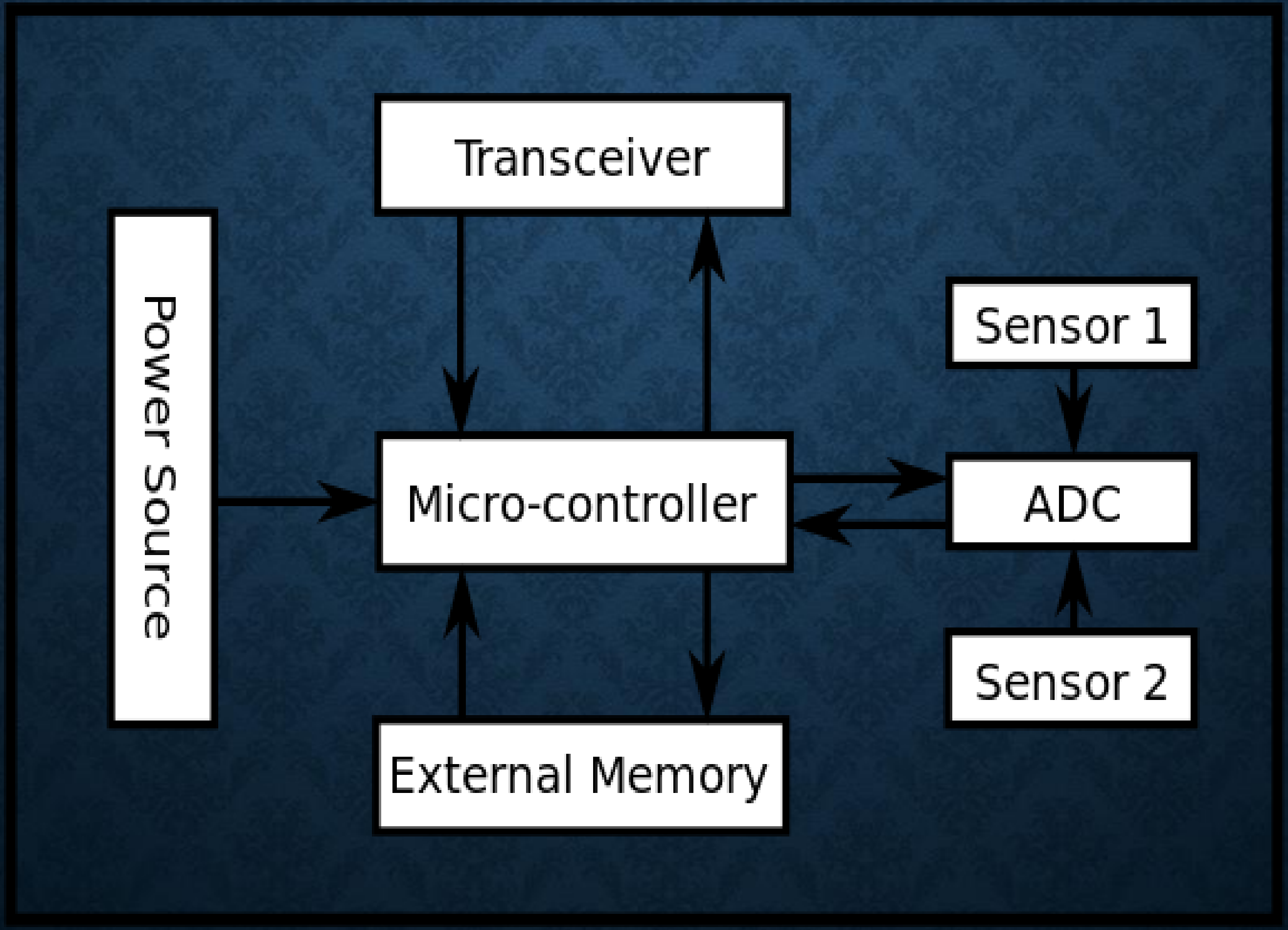
WSN (Wireless Sensor Network) –

- It mainly use broadcast communication. Wireless Sensing + Data Networking
- Consist of sensor networks which have – Low power, limited power, energy constrained due to small size.
- Large number of heterogeneous sensor node devices spread over a large field.

WSN



TYPICAL ARCHITECTURE OF SENSOR NODE



OPERATING SYSTEM FOR WSN

- TinyOS
- ContikiOS
- MANTIS
- SOS
- Nano-RK

WHAT IS TinyOS?

- TinyOS is a free open source operating system.
- Designed for WSN
- TinyOS began as a collaboration between University of California, Berkeley and Intel Research
- An embedded operating system written in nesC language,
- It features a component based architecture.

TinyOS DESIGN MODELS

Component-based model (Modularity)

- Simple functions are incorporated in components with clean interfaces.
- Complex functions can be implemented by complex components.

TinyOS DESIGN MODELS (CONTINUED)

Event-based Model

- Interact with outside by events (no command shell)
- There are two kinds of events for TinyOS –

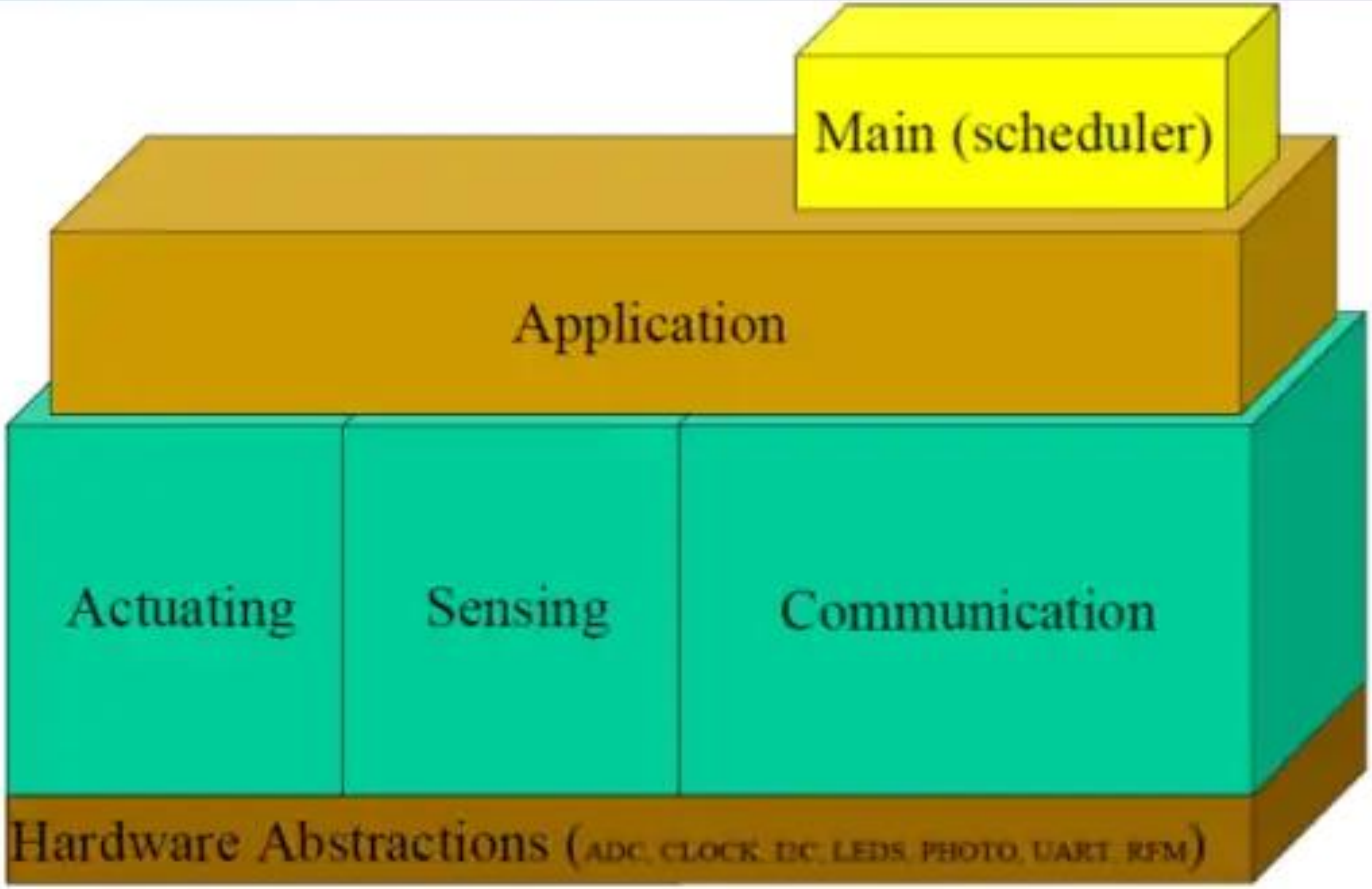
External events – Clock events and message events

Internal events – triggered by external events

FEATURES OF TinyOS

- Completely non-blocking
- Programs are built out of software components.
- Tasks are non-preemptive and run in FIFO order.
- TinyOS code is statically linked

Structure of TinyOS


Main (scheduler)
Application
Actuating
Sensing
Communication
Hardware Abstractions (ADC, CLOCK, I2C, LEDS, PHOTO, UART, RFM)

TinyOS AS A SOLUTION

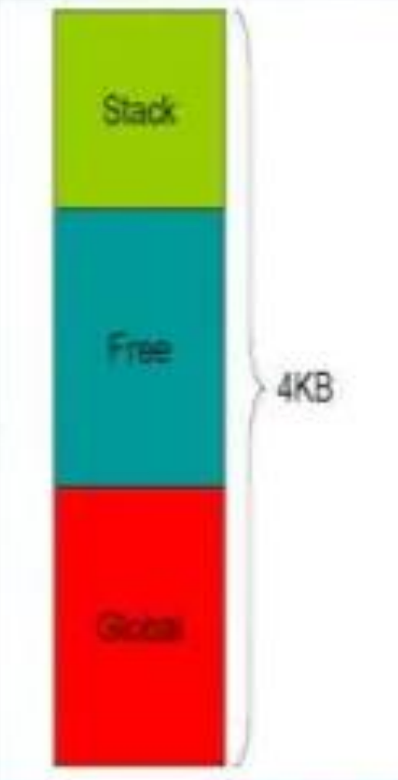
- Component based architecture allows frequent changes while still keeping the size of code minimum.
- Event based execution model means no user/kernel boundary and hence supports high concurrency.
- It is power efficient as it makes the sensors sleep as soon as possible.
- Has small footprint as it uses a non-preemptable FIFO task scheduling.

TinyOS MODELS

- Data Model
- Thread Model
- Programming Model
- Component Model
- Network Model

Data Memory Model

- **Static Memory Allocation**
  - No Heaps or any other dynamic structures used.
  - Memory requirements determined at compile time.
  - *This increases the runtime efficiency.*
- **Global variables**
  - Allocated on per frame basis.
- **Local Variables**
  - Saved on the stack
  - Defined in the function/method



THREAD MODEL

Power-Aware Two-levels Scheduling

- Long running tasks and interrupt events
- Sleep unless tasks in queue, wakeup on event

Tasks

- Time-flexible, background jobs
- Atomic with respect to other tasks
- Can be preempted by events

Events

- Time-critical, shorter duration
- Last-in first-out semantic (no priority)
- Can post tasks for deferred execution

PROGRAMMING MODEL

Separation construction/composition

Construction of Modules

- Modules implementation similar to C coding
- Programs are built out of components
- Each component specifies an interface
- Interfaces are "hooks" for wiring components

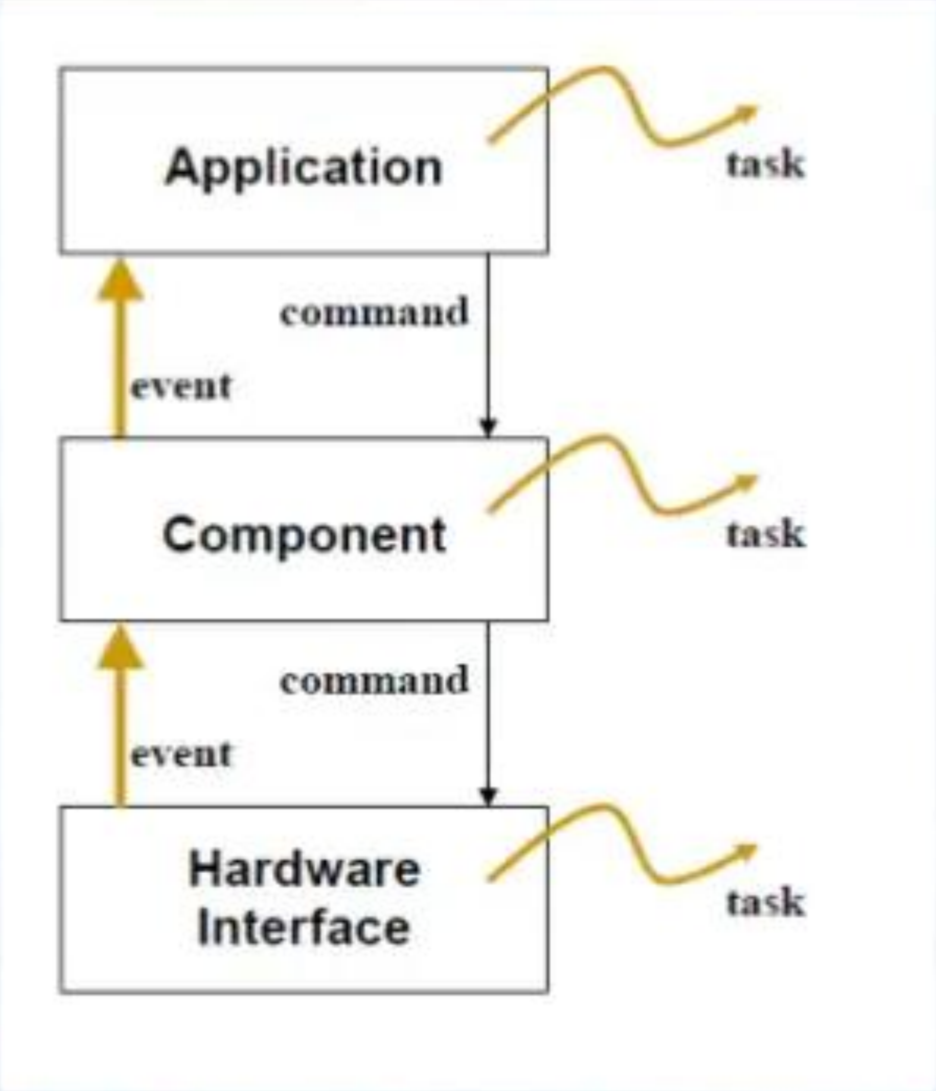
Composition of Configurations

- Components are statically wired together
- Increases programming efficiency (code reuse) an runtime efficiency

COMPONENT MODEL

- Components should use and provide bidirectional interfaces.
- Components should call and implement commands and signal and handle events.
- Components must handle events of used interfaces and also provide interfaces that must implement commands.

TinyOS Basic Constructs



TinyOS Basic Constructs

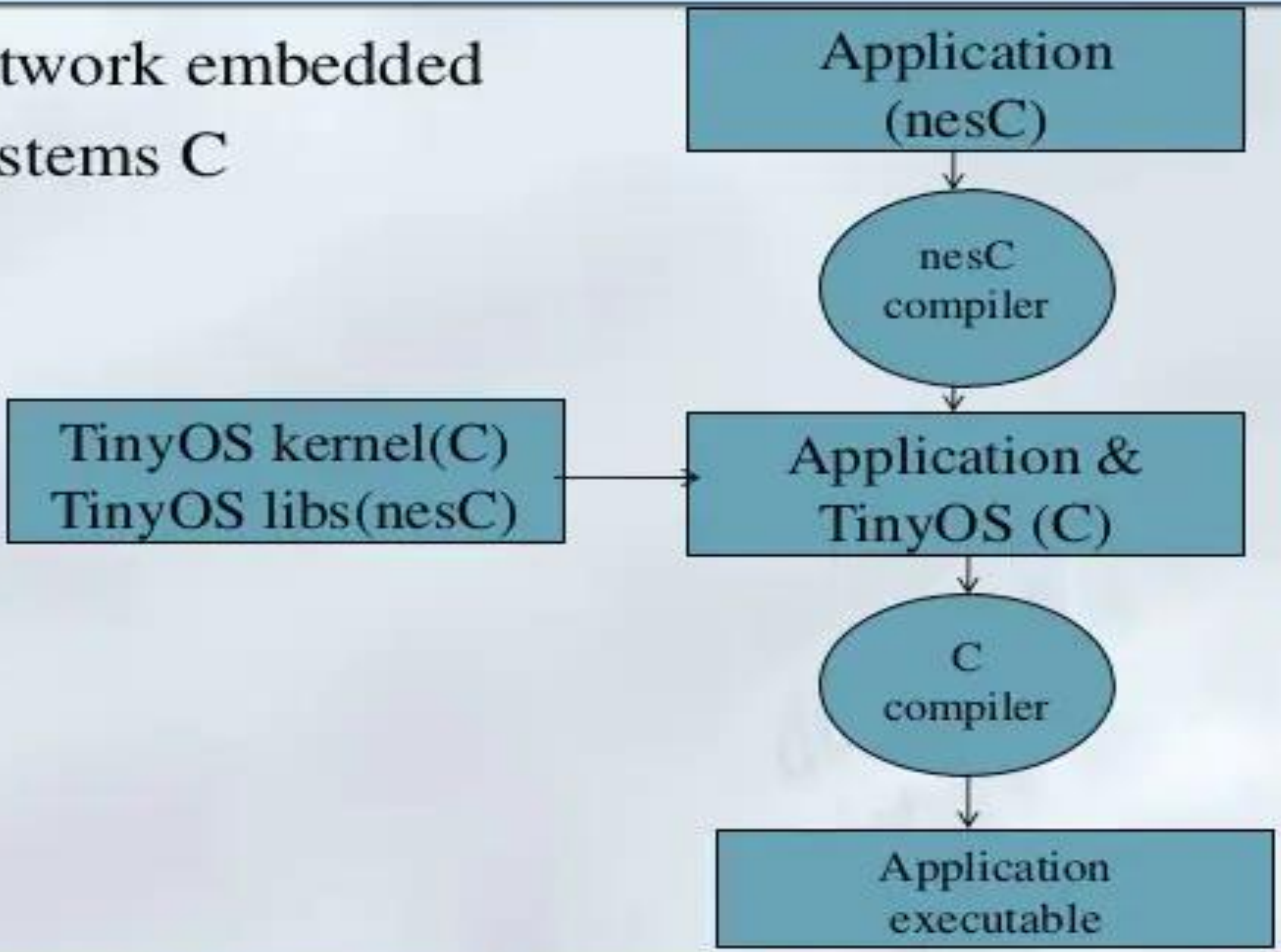
- **Commands**
  - Cause actions to be initiated
- **Events**
  - Small amount of processing to be done in a timely manner
  - E.g. timer, ADC interrupts
  - Notify that action has occurred.
  - Can interrupt longer running tasks

TinyOS Basic Constructs

- **Tasks**
  - Background Computation
  - Not time critical
  - Larger amount of processing. E.g. : computing the average of a set of readings in an array
  - Run to completion with respect to other tasks.Only need a single stack.

The nesC Language

- nesC – network embedded systems C



Application (nesC)

nesC compiler

TinyOS kernel(C)
TinyOS libs(nesC)

Application & TinyOS (C)

C compiler

Application executable

THE nesC LANGUAGE

- An extension to the C programming language, embody the concepts and execution model of TinyOS.
- Filename extension .nc

Static language

- No dynamic memory (malloc)
- No function pointers
- No heap
- Includes task FIFO scheduler
- Designed to encourage code reuse.

THE nesC LANGUAGE

- nesC (pronounced "NES-see") is a component-based, event-driven programming language used to build applications for the TinyOS platform.
- TinyOS is an operating environment designed to run on embedded devices used in distributed wireless sensor networks.
- nesC is built as an extension to the C programming language with components "wired" together to run applications on TinyOS.
- The name nesC is an abbreviation of "network embedded systems C".