

Resource Management Aspects for Sensor Network Software

Sean Walton
Eric Eide

School of Computing
University of Utah

PLOS 2007

SCHOOL OF
COMPUTING

AT THE UNIVERSITY OF UTAH



Aspect nesC

Augmenting nesC with AOP technology:

- ✓ To facilitate resource management in sensor network devices
- ✓ To experiment with AOP in embedded, componentized programming environments
- ✓ To aspectize embedded OS like TinyOS

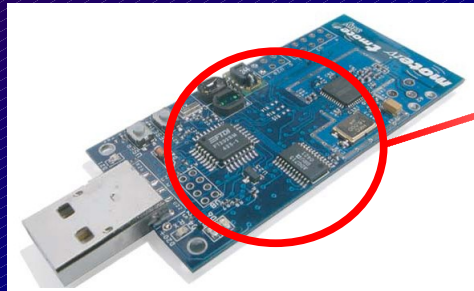
Work in progress...

Limited Resources Constrain Development

Tight microcontroller (μ C) resources

Sensor networks used for monitoring environments

Careful code development

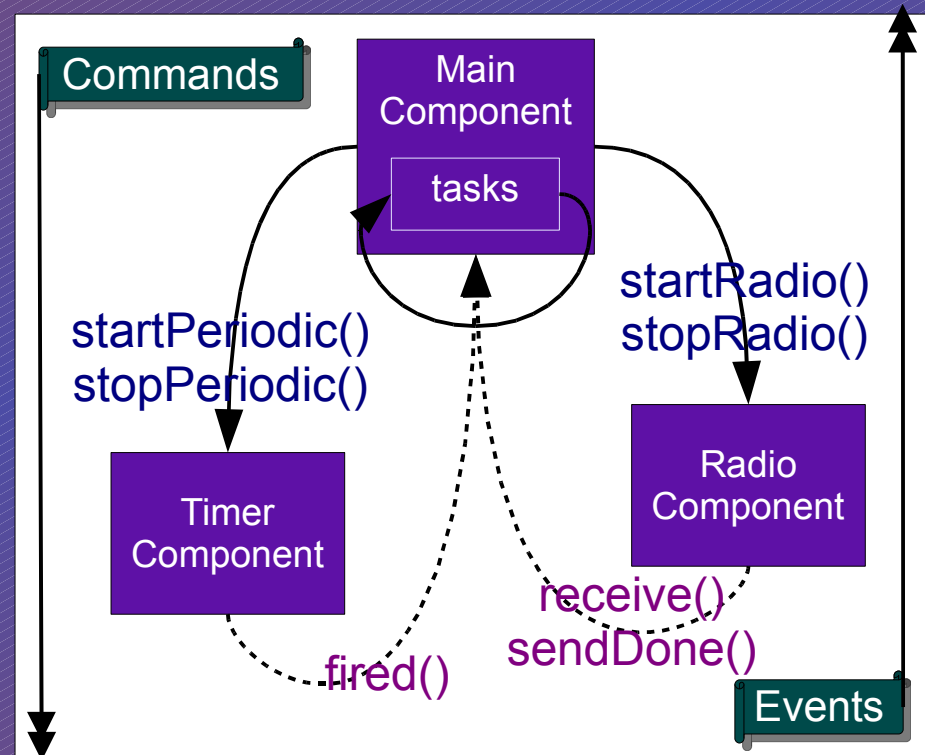
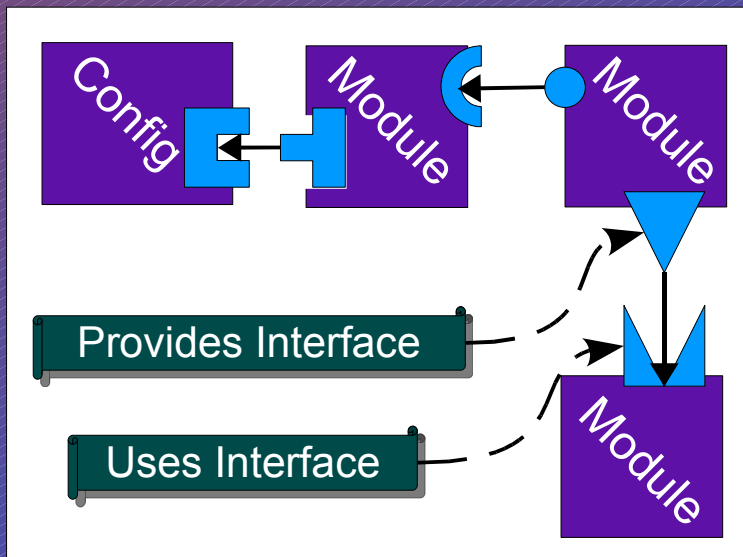


Model	Name	Bits	Flash	EEPROM	RAM	Regs
MSP430	F2001	16	1K	256	128	16
MSP430	F2011	16	2K	256	128	16
MSP430	F2121	16	4K	256	256	16
MSP430	F2131	16	8K	256	256	16
MSP430	F2252	16	16K	256	512	16
MSP430	F2272	16	32K	256	1024	16
AT	tiny13	8	1K	64	64	32
AT	tiny26	8	2K	128	128	32
AT	mega48	8	4K	256	512	32
AT	mega8	8	8K	512	1024	32
AT	mega169P	8	16K	512	1024	32
AT	mega32	8	32K	1024	2048	32
...						

Background: nesC and TinyOS

nesC is a
componentized
version of C.

nesC Static View



nesC Dynamic View

TinyOS application
comprises tasks and
events.

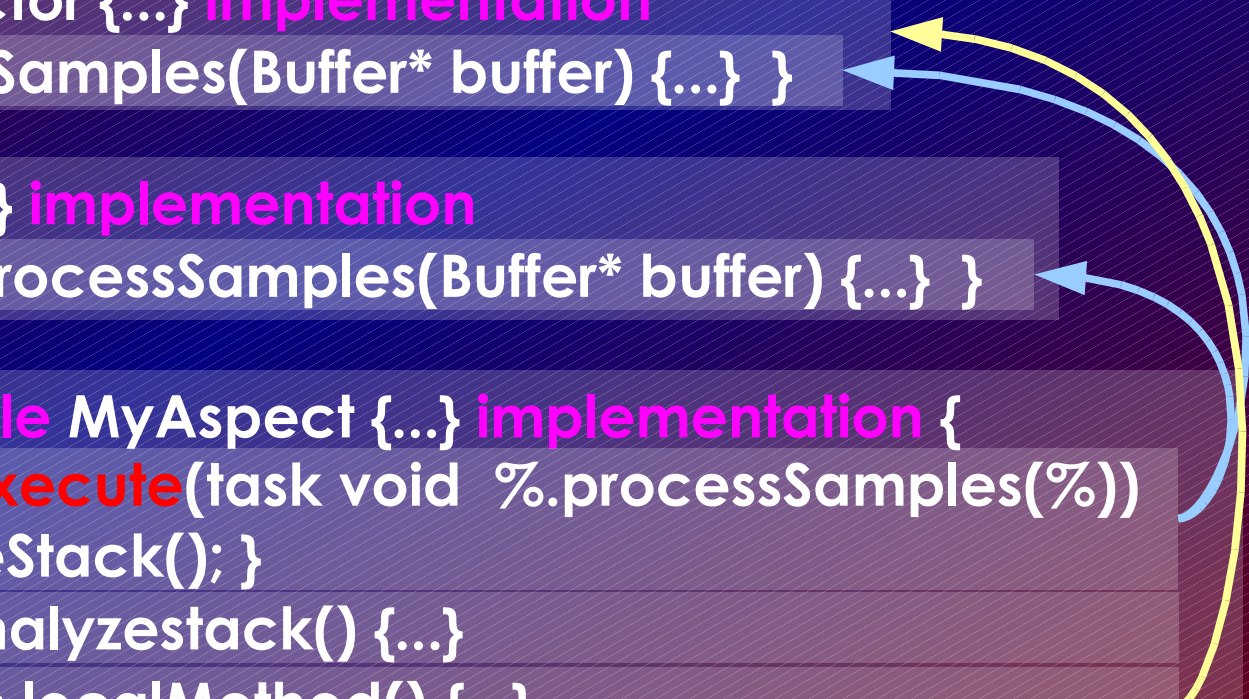
Background: Aspect-Oriented Programming

Modularizes crosscutting concerns (CCCs)

```
module TornadoDetector {...} implementation
{ task void processSamples(Buffer* buffer) {...} }
```

```
module Diagnostic {...} implementation
{ command void processSamples(Buffer* buffer) {...} }
```

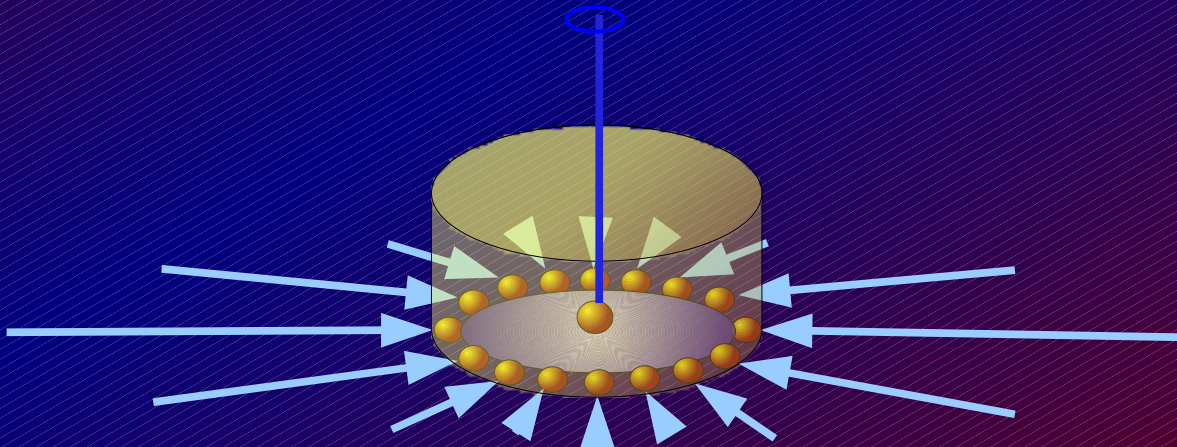
```
aspect module MyAspect {...} implementation {
  before(): execute(task void %.processSamples(%))
  { analyzeStack(); }
  inline int analyzestack() {...}
  int MyClass.localMethod() {...}
}
```



Advice can augment, modify, or replace code

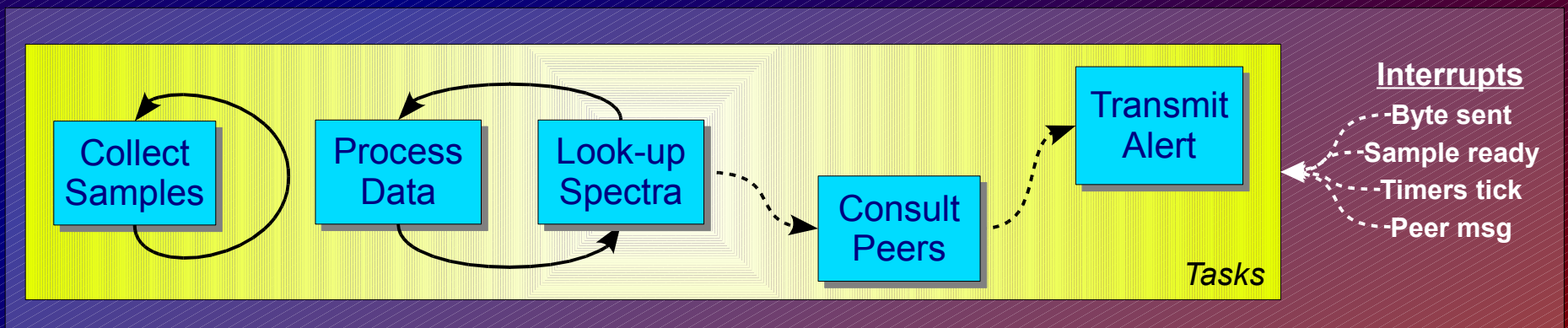
Example Model: Tornado Detector

- Collect infrasound audio samples
- Process audio data and compare with tornado signature database
- Fourier Transform (FFT) spectral processing
- Corroborate and determine vector



Programming Strains Capabilities

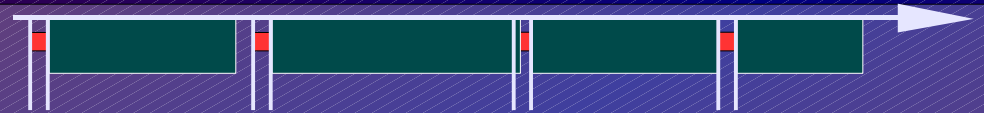
- FFTs → CPU- and stack-intensive
- Sample region (RAM) limit total sample size
- Signature database (Flash) limit sample types
- Data processing can overrun data collection



Target Example Resources

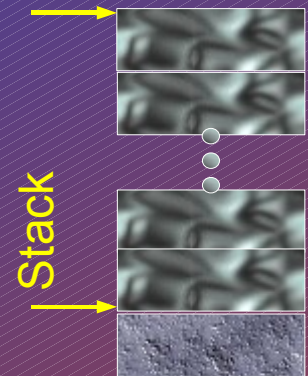
Real-time deadlines

- Event-/interrupt-driven processing requires on-time completion



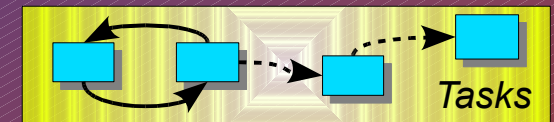
Limited stack space

- μC have fixed-position/small execution stacks



API-managed resources

- Management crosses application/OS boundaries.

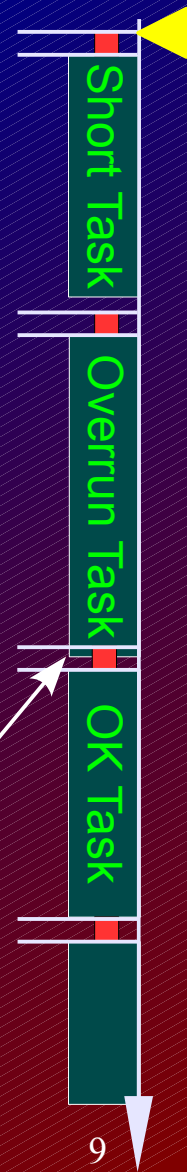


Real-Time Deadlines Advice

- TinyOS tasks are run-to-completion
- Atomic sections can present problems

```
aspect module WatchRealtimeDeadline {  
} implementation {  
  advice after(): task_deadline_reached() {  
    //--- Change FFT accuracy, reducing calculations  
    //---or- Change data-sample duration  
  }  
  advice before(): task_completion() {  
    //--- If ends before deadline  
    //--- Increase analysis accuracy  
    //--- If accuracy tuning was visited before  
    //--- Revert accuracy  
  }  
}
```

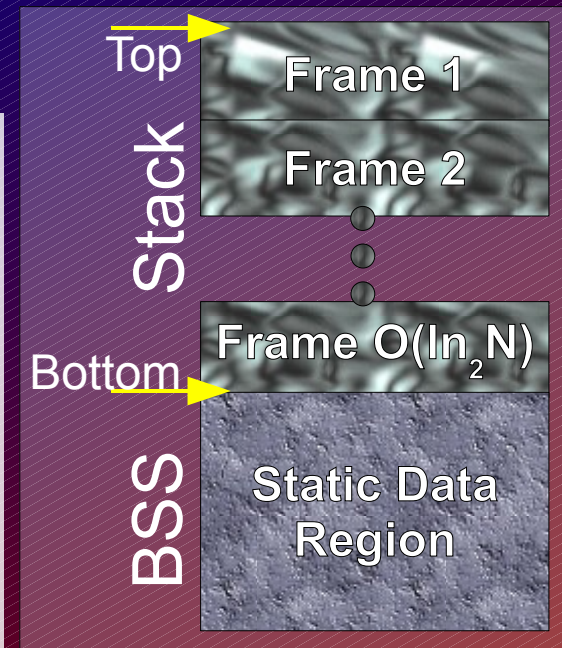
Overrun
event



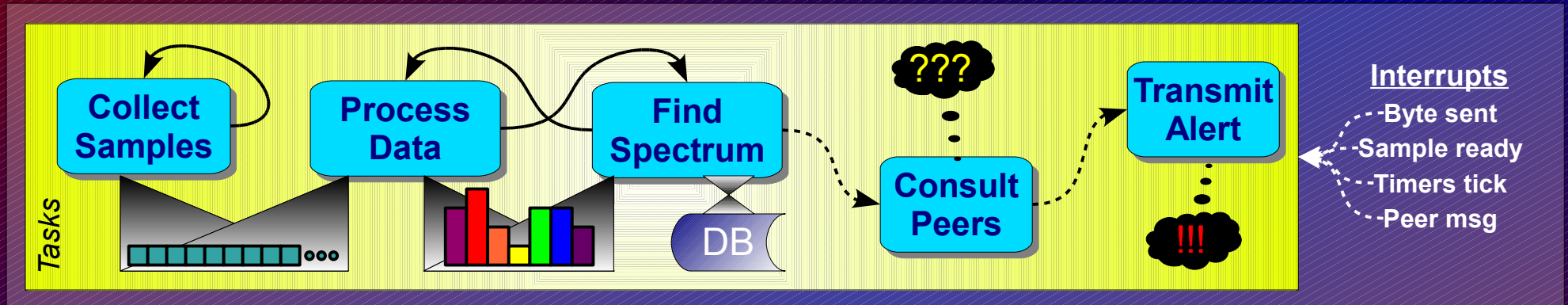
Stack Overflow Advice

- Sample size determines stack usage ($O(\ln_2 N)$)
- A diagnostic mode switch may overflow stack
- Overflow response can use intermediate results
- Overflow *prevention* is the target

```
aspect module PreventStackOverflow {  
} implementation {  
    advice inline after(): stack_overflow_imminent() &&  
        withintask(task void process_data(void)) {  
        //--- Report the current results of processing  
        //--- Stop task and release buffers  
    }  
}
```



Limited Resource Advice



- More data sampled → more accurate the results.
- Buffer hand-off (dangling pointers).
- Memory pool availability changes over time.
- Sending a message requires memory.
- Message sending happens infrequently.

Advice Options

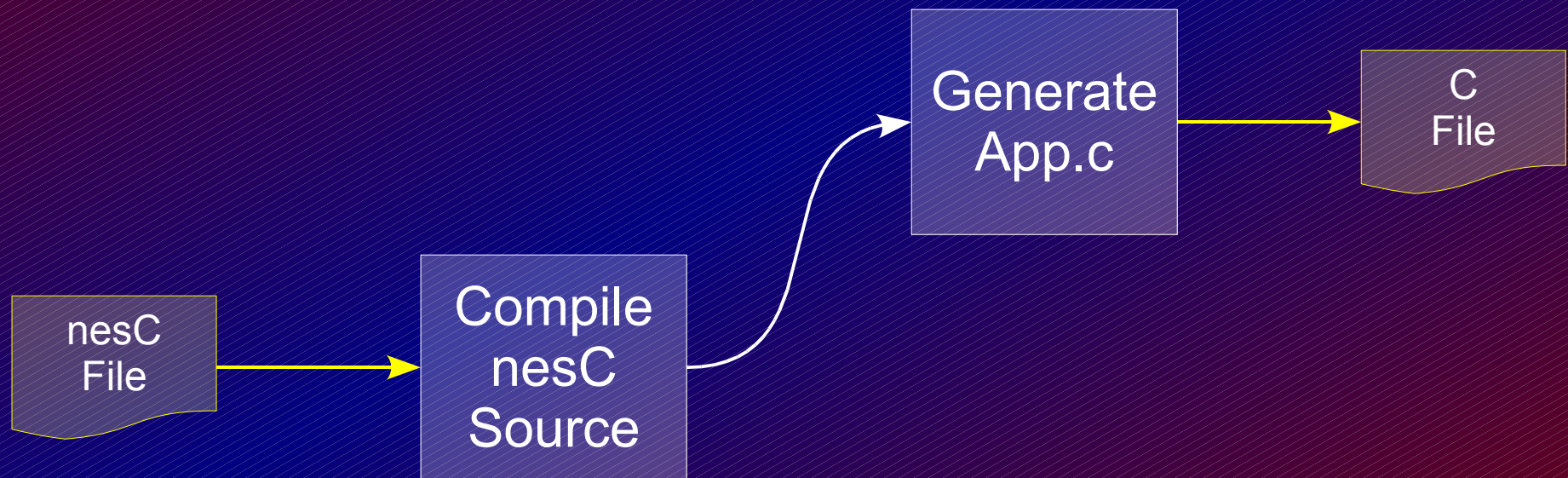
What to do with system state when interrupting program flow?

- Fixing up algorithmic state (e.g., locks)
- Increasing limited resources
- Unwrapping jumps to a known state (e.g., setjmp())
- Aborting stops a task and resets state

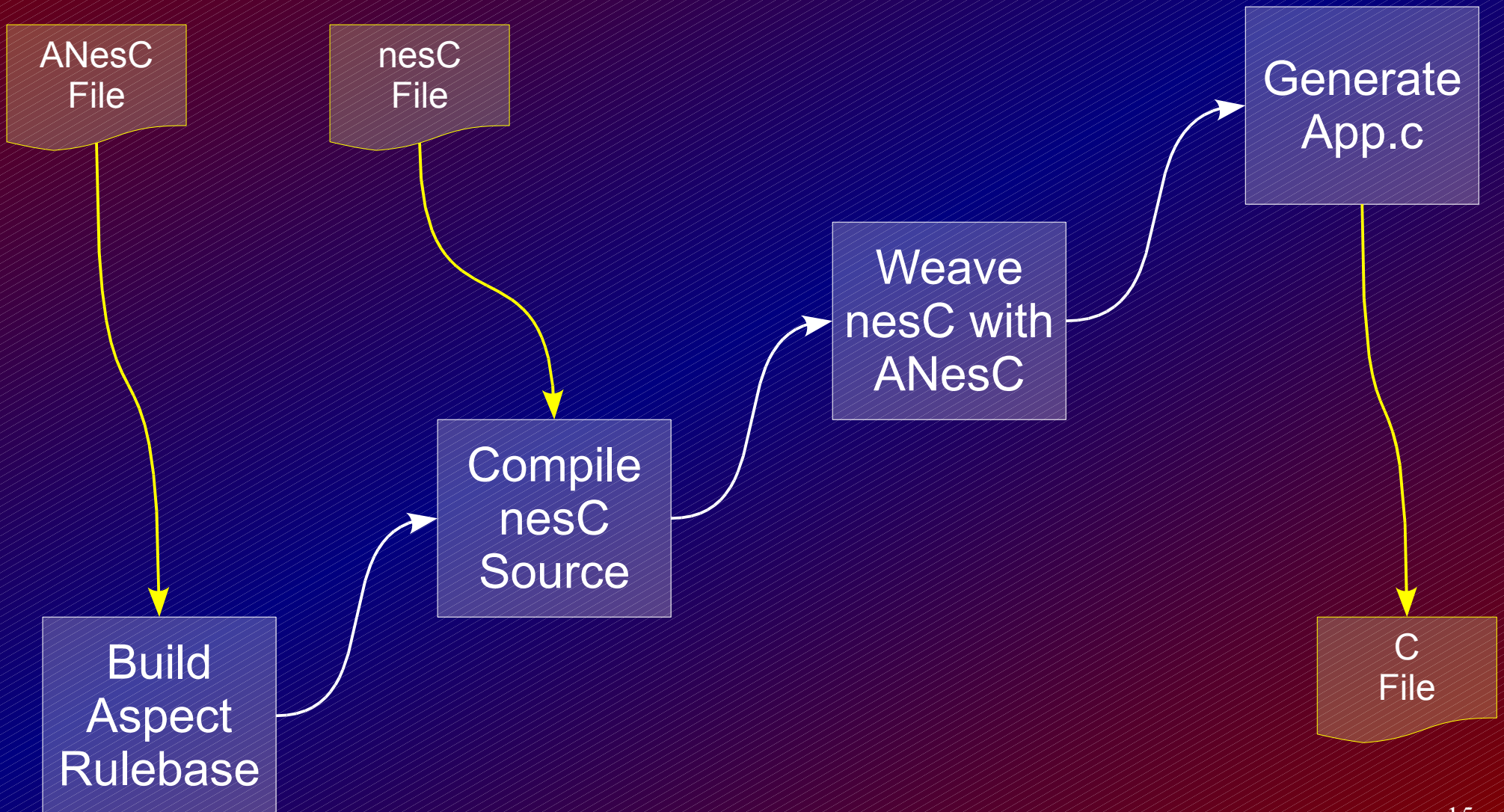
Integrating with nesC

- Extended nesC syntax
- Standard AOP
- Embedded AOP
- Minimal code footprint

nesC Processing



Aspect nesC Processing



Summary

Extending nesC with AOP capabilities to:

- Facilitate resource management in sensor networks
- Capture nesC's tasks, commands, etc.
- Reconfigure component wiring
- Used two problems in a tornado example model
- Showed ANesC's integration with nesC