# Swift Fox

## Programming sensor networks for fun and profit

Marcin Szczodrak[1] Vasileios P. Kemerlis[1]
Xuan Linh Vu[2] Yiwei Gu[2]

Programing Languages and Translators (PLT)
Computer Science Department
Columbia University

[1]{*msz*, *vpk*}@cs.columbia.edu
[2]{*xv2103*, *yg2181*}@cs.columbia.edu

04/26/2010

# Outline

1. Introduction (Marcin Szczodrak)
   - Overview
   - Problem statement
   - Swift Fox language
2. Language internals (Vasileios Kemerlis)
   - Language constructs
   - Syntactic structures
3. Compiler details (Yiwei Gu)
   - Compiler architecture
   - Development tools
   - Testing tools
4. Conclusion (Xuan Linh Vu)
   - Lessons learned
   - Why Swift Fox

# Outline

1. Introduction (Marcin Szczodrak)
   - Overview
   - Problem statement
   - Swift Fox language
2. Language internals (Vasileios Kemerlis)
   - Language constructs
   - Syntactic structures
3. Compiler details (Yiwei Gu)
   - Compiler architecture
   - Development tools
   - Testing tools
4. Conclusion (Xuan Linh Vu)
   - Lessons learned
   - Why Swift Fox

# Wireless Sensor Networks

## What are they?

- wireless ad-hoc networks

**multipurpose sensor nodes (*motes*)**

- small
- low-cost, low-power
- self-organizing capabilities

- ultimately at the size of a grain of sand (*smart dust*)


Figure: Tiny mote (courtesy of MIT Technology Review)

# Wireless Sensor Networks

## Why bother?

- WSNs are pervasive
  - military (battlefield surveillance, reconnaissance)
  - environment (pollution monitoring, chemical detection)
  - home automation (“smart home”)
  - commercial domain

**but...**

- no standardized system facilities
- absence of high-level abstractions
- “single” image implementations

# Swift Fox

## Bringing back the fun in WSN programming

- simple, event-driven language for describing reconfiguration policies for WSNs

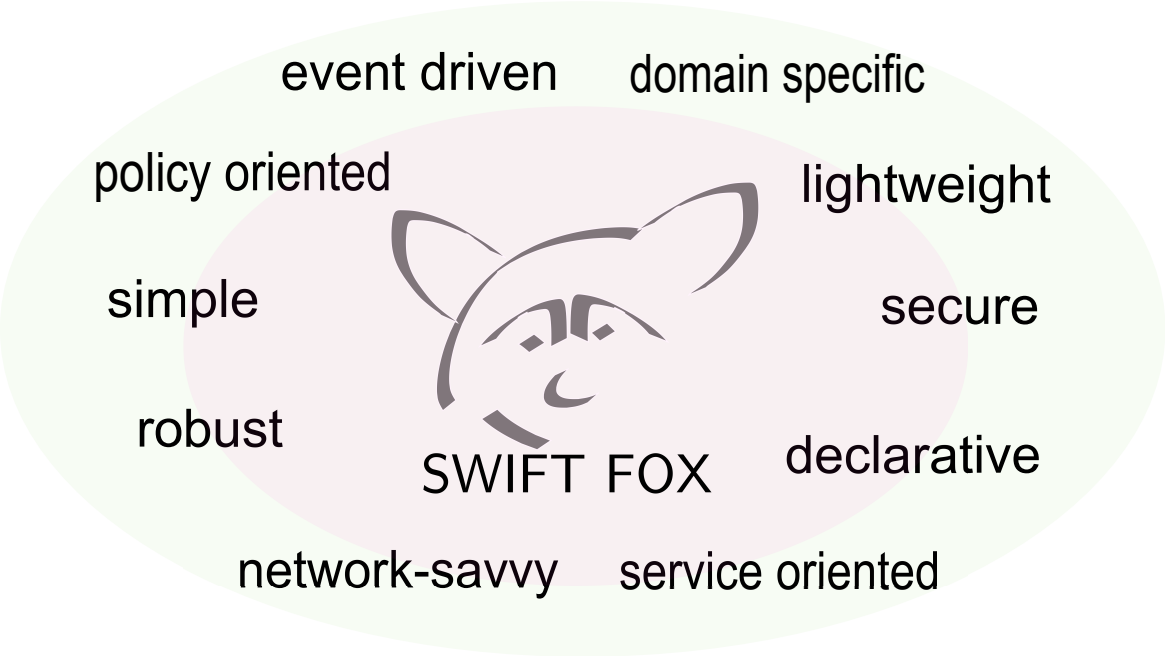


Figure: Buzzwords for Swift Fox

# Swift Fox
## Distinctive characteristics

- simple, simple, simple
- enables code/logic re-use
- releases the programmer from the burden of dealing with WSN OS internals, event handling, data scatter/gather, network and routing protocol details

**first** programming language for WSN applications

- solve the “problem” and avoid plumbing

# Outline

1. Introduction (Marcin Szczodrak)
   - Overview
   - Problem statement
   - Swift Fox language
2. Language internals (Vasileios Kemerlis)
   - Language constructs
   - Syntactic structures
3. Compiler details (Yiwei Gu)
   - Compiler architecture
   - Development tools
   - Testing tools
4. Conclusion (Xuan Linh Vu)
   - Lessons learned
   - Why Swift Fox

# Swift Fox
## Essential constructs

- **configuration**: *binding* between an application and a network protocol
- **event-condition**: *predicate* that becomes true when a specific sensor reading satisfies a condition
- **policy**: *transition* specification between different configuration, upon event-conditions

### Example

- **configuration** *too-cold* {Send-Temp CTP}
- **event-condition** *cold-day* {Temperature < 70F}
- **from any goto** *too-cold* **when** *cold-day*

```
# define configurations
configuration sleep-day {nothing CTP}
configuration sleep-night {nothing CTP}
configuration too-cold {Send-Temp CTP}

# define time passing events
event-condition day {Timer = 16hr}
event-condition night {Timer = 8hr}

# define temperature sensing events
event-condition cold-day {Temperature < 70F }
event-condition cold-night {Temperature < 60F }

# reconfiguration policies
from any goto sleep-day when day
from any goto sleep-night when night
from sleep-day goto too-cold when cold-day
from sleep-night goto too-cold when cold-night
from too-cold goto sleep-day when not cold-day
from too-cold goto sleep-night when not cold-night

# and finally, the initial configuration
start sleep-day
```

# Swift Fox
## Example tree

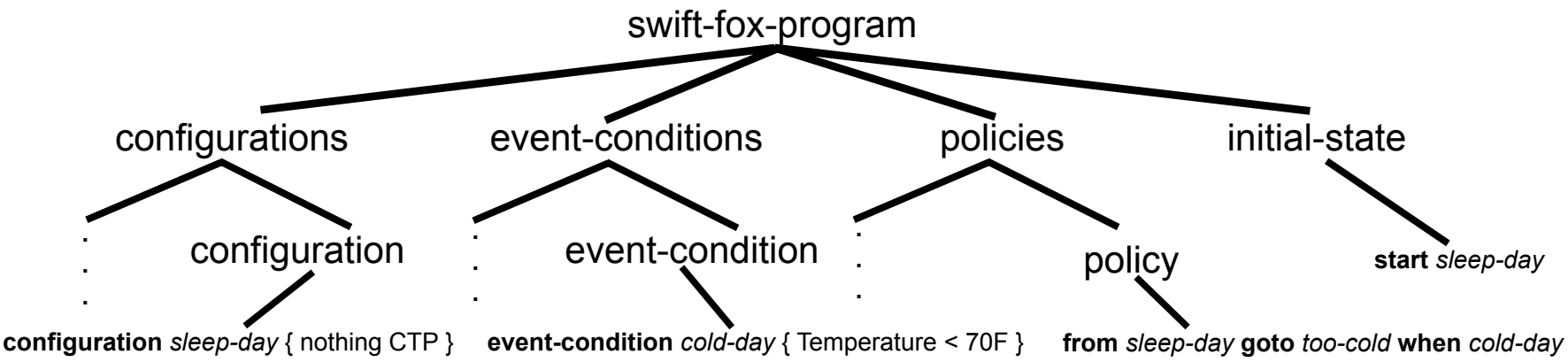


Figure: AST for the previous code snippet

# Outline

1. Introduction (Marcin Szczodrak)
   - Overview
   - Problem statement
   - Swift Fox language
2. Language internals (Vasileios Kemerlis)
   - Language constructs
   - Syntactic structures
3. Compiler details (Yiwei Gu)
   - Compiler architecture
   - Development tools
   - Testing tools
4. Conclusion (Xuan Linh Vu)
   - Lessons learned
   - Why Swift Fox

# Swift Fox
## Compiler block diagram

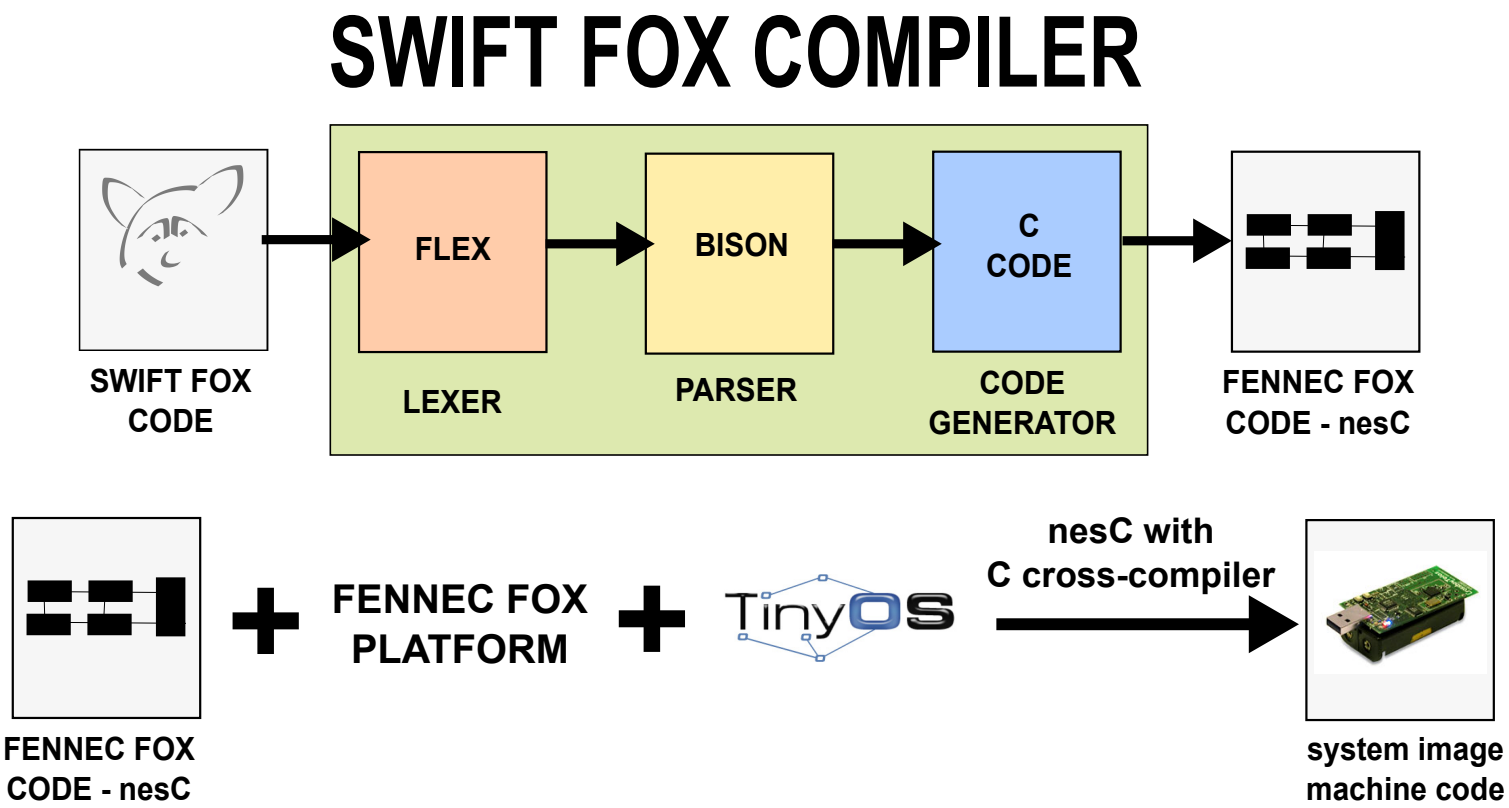


Figure: Swift Fox block diagram

# Swift Fox
## Development tools

### development

- Lex (flex), YACC (bison)
- nesC, TinyOS
- make

### management & documentation

- Trac (web-management and bug-tracking)
- Subversion (revision control)
- LaTeX

# Swift Fox
## Testing procedure

- assume correctness of the front-end generators and execution environment (*e.g.,* Lex, YACC, nesC, TinyOS, Fennec Fox)
- combination of *unit* and *regression* testing
- separate regression testing suites for the lexer, parser, and code generator

# Outline

1. Introduction (Marcin Szczodrak)
   - Overview
   - Problem statement
   - Swift Fox language
2. Language internals (Vasileios Kemerlis)
   - Language constructs
   - Syntactic structures
3. Compiler details (Yiwei Gu)
   - Compiler architecture
   - Development tools
   - Testing tools
4. Conclusion (Xuan Linh Vu)
   - Lessons learned
   - Why Swift Fox

# Swift Fox
## What we learned

- testing is important
- keep it simple, add features steadily
- documentation helps!
- project management is hard

# Why Swift Fox?

- first language (of that kind) out there
- simpler than coding in *nesC*
- solve the “problem” and avoid plumbing

**Try it! (*coming soon...*)**

`http://nslvm2.cs.columbia.edu`