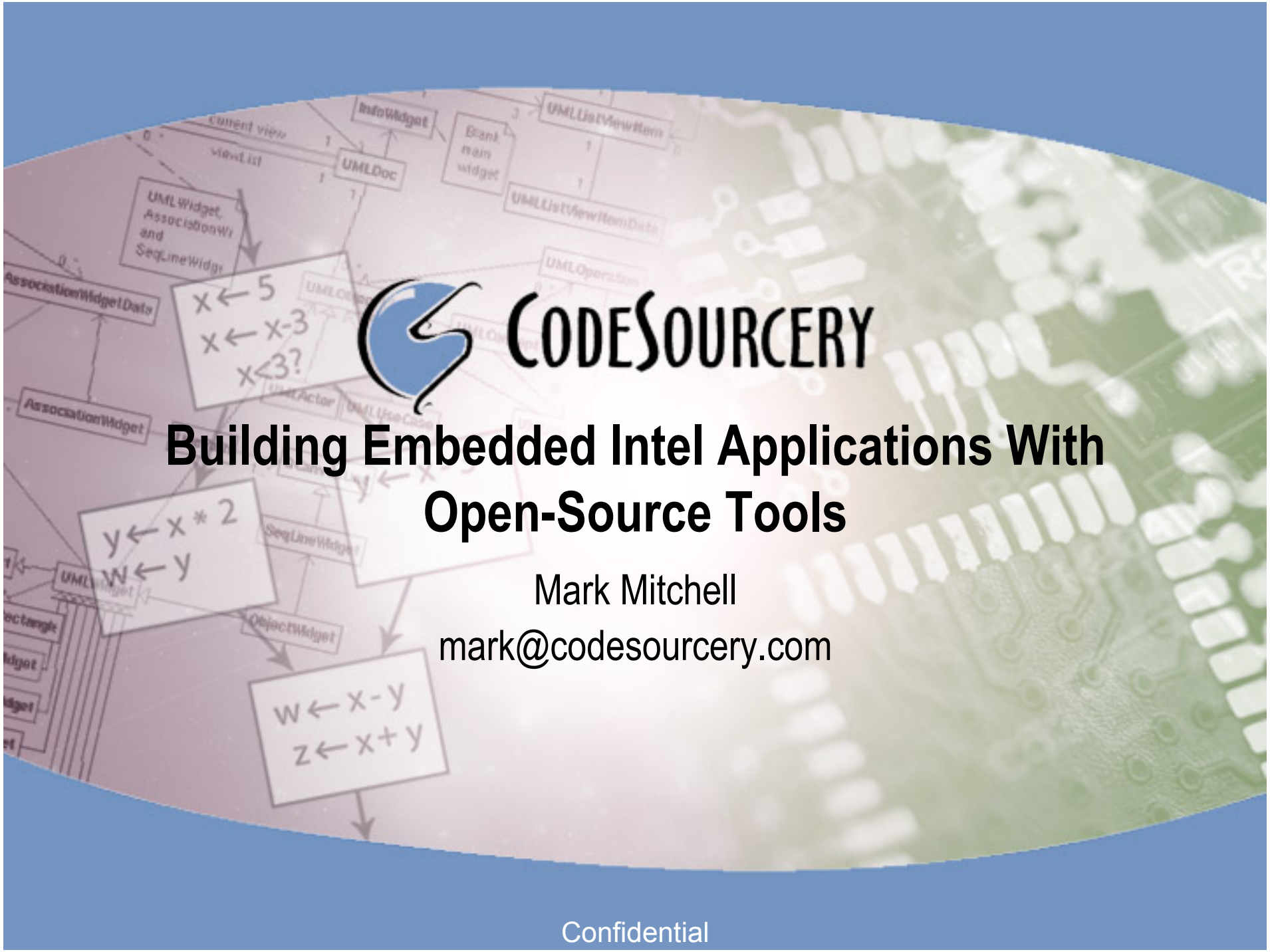
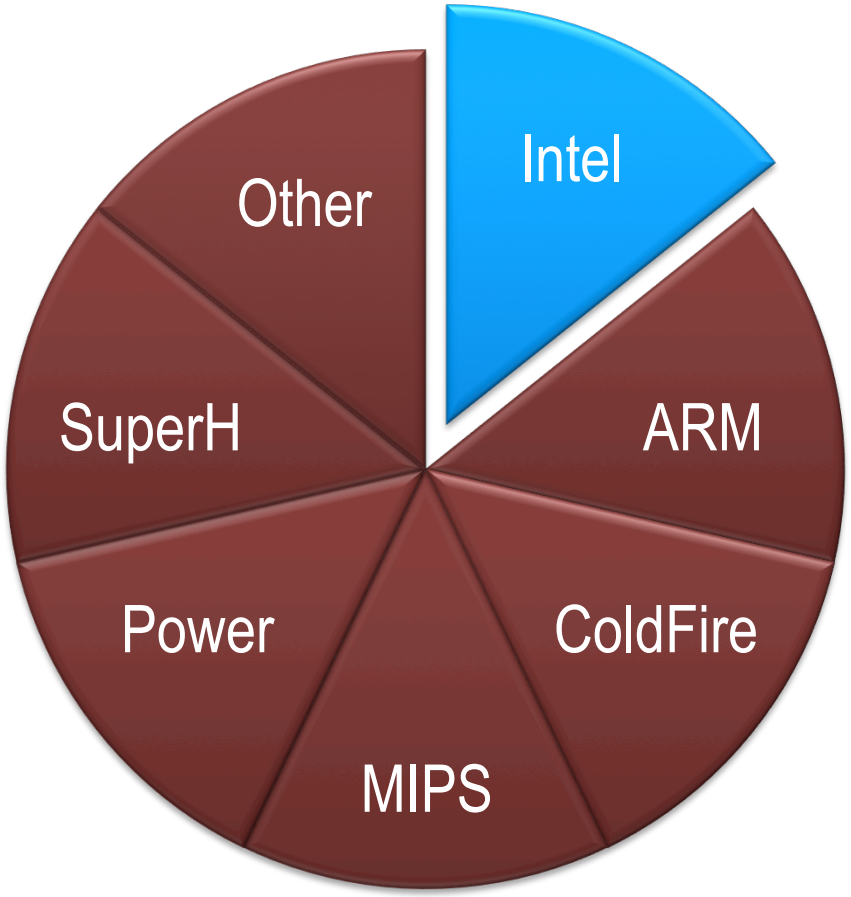


Building Embedded Intel Applications With Open-Source Tools

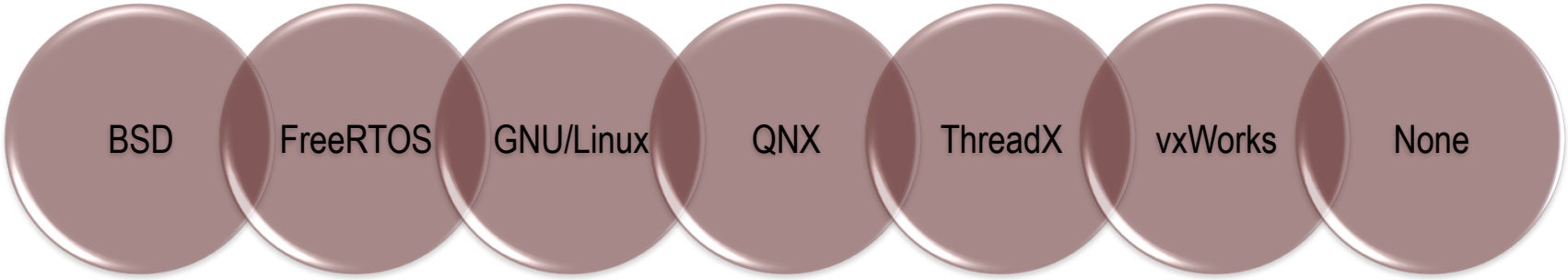
Mark Mitchell
mark@codesourcery.com

Introduction

Which Embedded Architectures Are You Using?

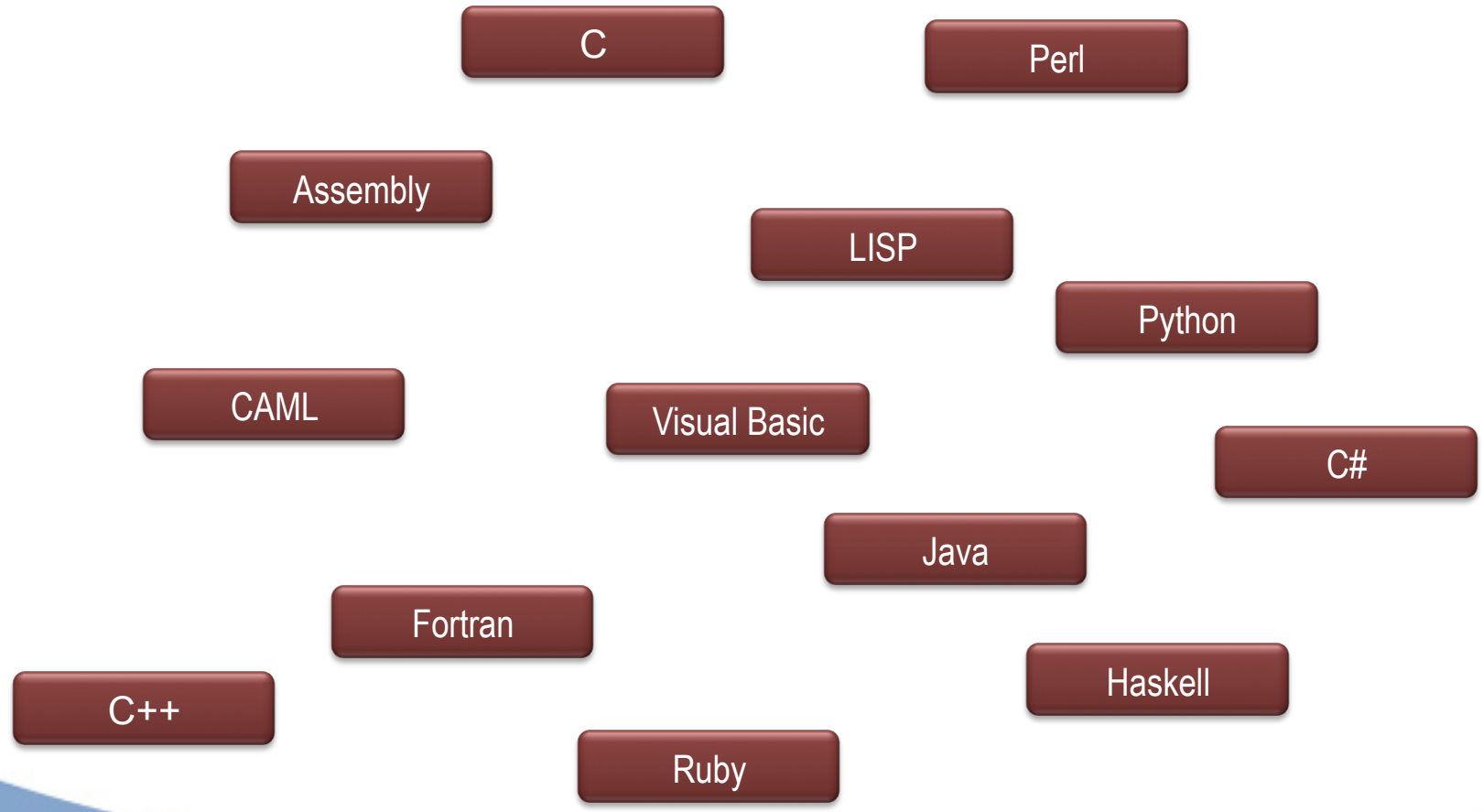


Which Embedded Operating Systems?



Embedded Systems Are Different

Fewer Programming Languages



Fewer Programming Languages

C

Assembly

C++

Different Goals

Minimize power usage

- Battery lifetime is critical for portable devices
- Performance is often about getting back to sleep
- Even fixed devices often have strict power requirements
- Heat generation is a function of power consumption

Minimize footprint

- RAM is expensive
- Flash is very expensive
- Disks? What disks would those be?

Meet real-time requirements

- Algorithms must have predictable worst-case performance
- Code must be interruptible

Weird Hardware Stuff

Memory maps

- Program code must go here …
- … while data must go there …
- … and peripherals are over here …

Peripherals

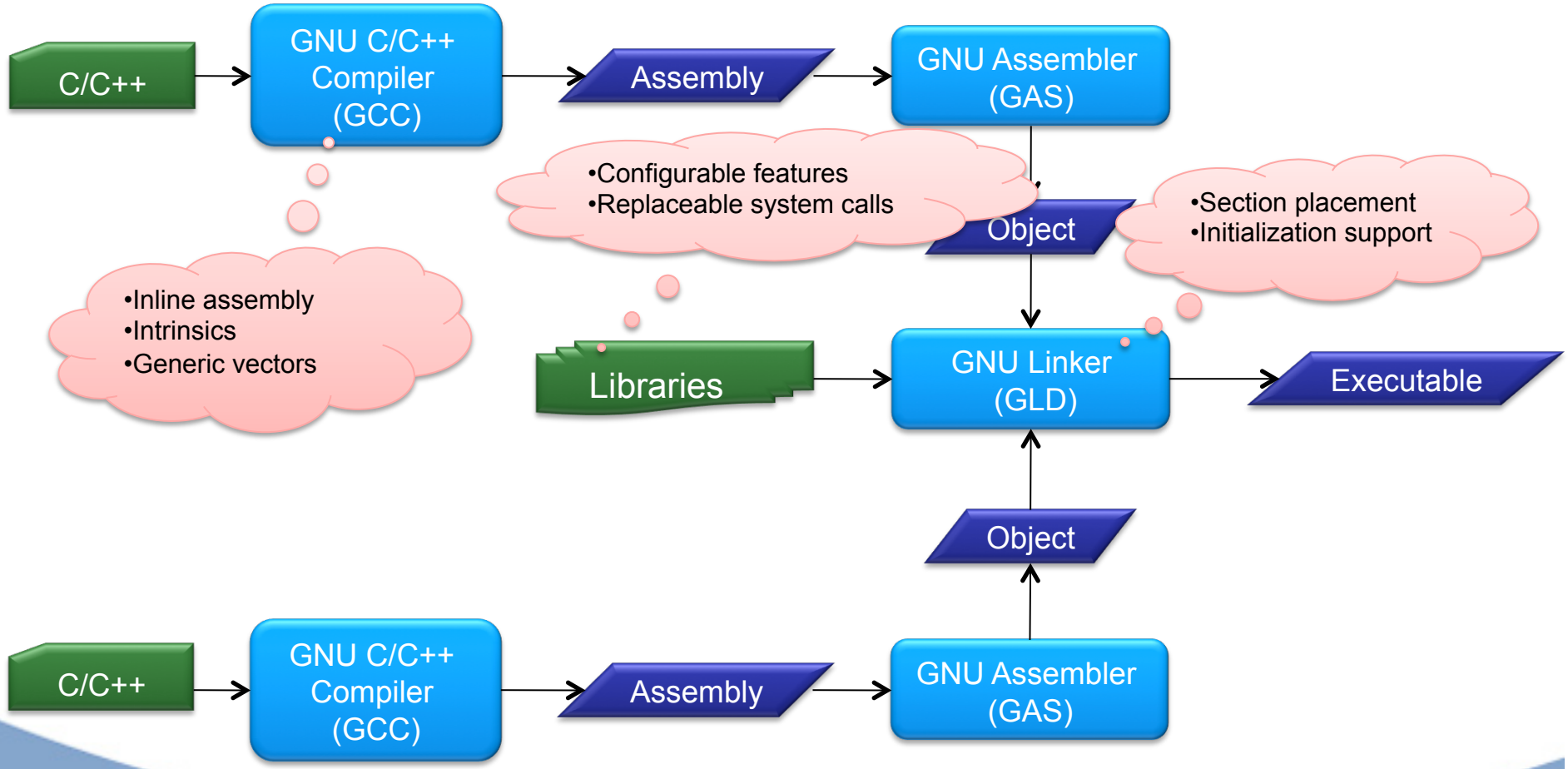
- Analog inputs and outputs
- Real-time requirements
- Fault-tolerance requirements

Complex debug cycle

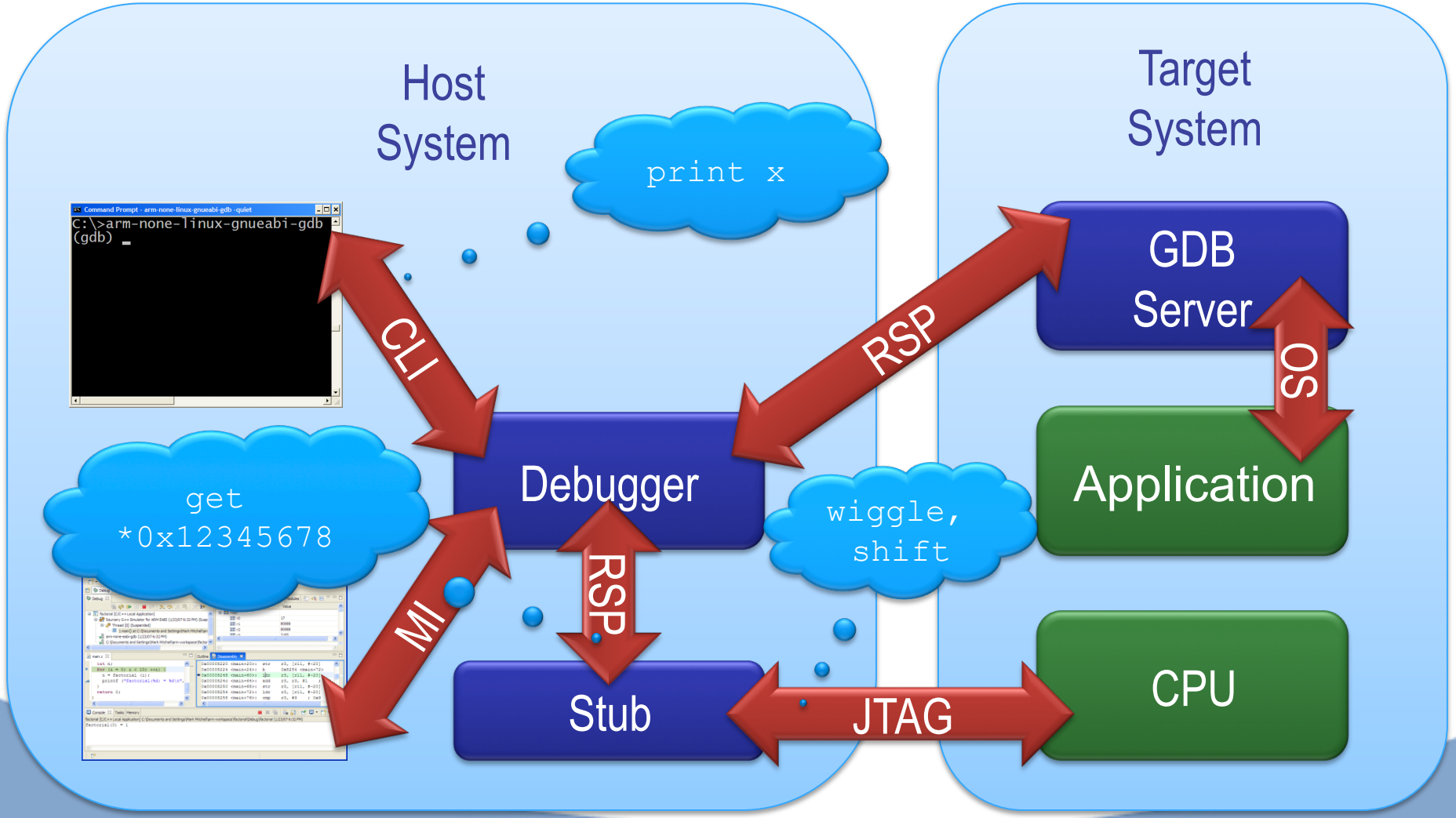
- Editing the program requires reflashing the system
- Debugging requires connecting a JTAG probe to the system
- Debugging the application often changes how it behaves

Open-Source Tools

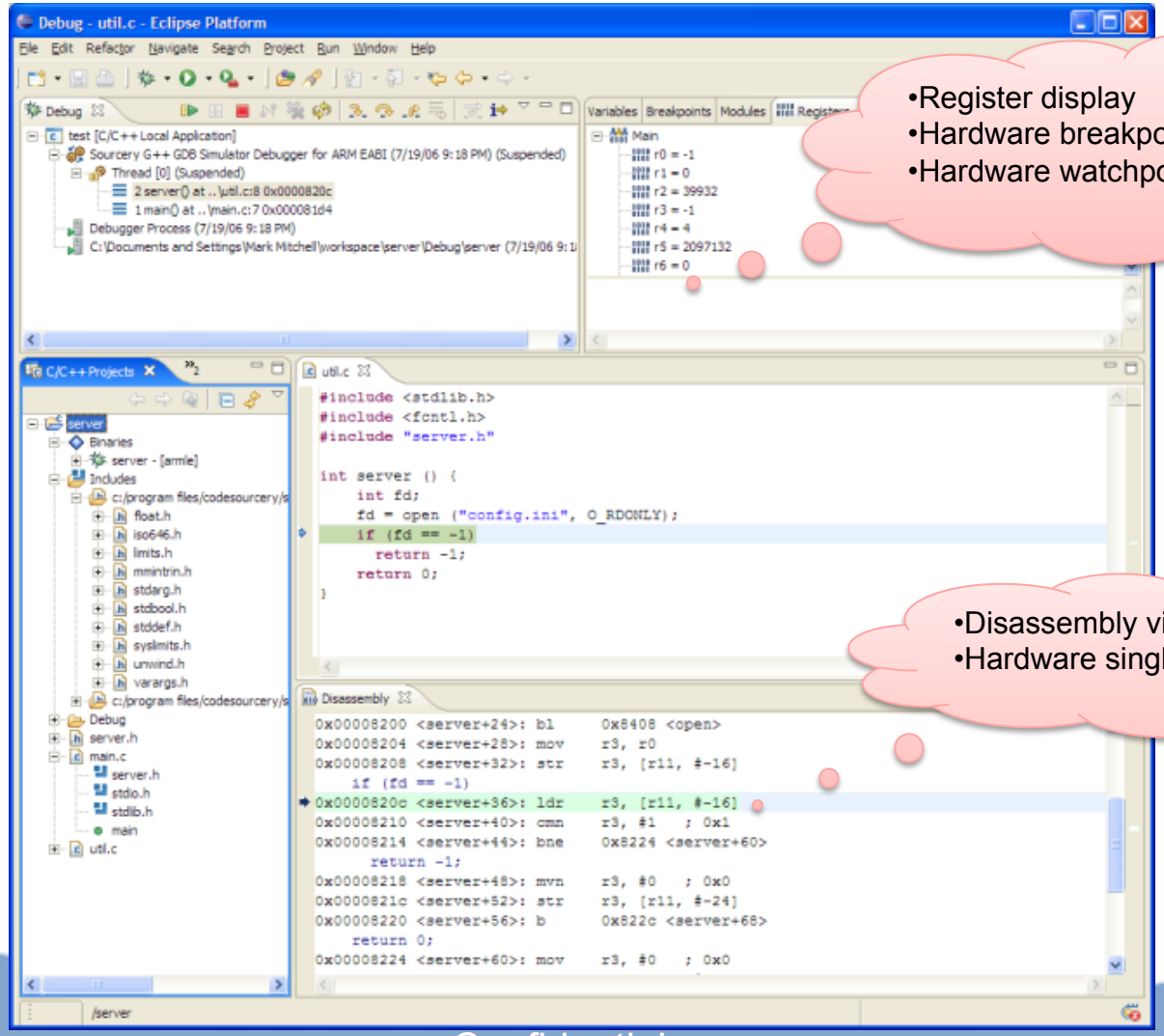
GNU Toolchain



GNU Debug Architecture



Eclipse



- Register display
- Hardware breakpoints
- Hardware watchpoints

- Disassembly view
- Hardware single-step

Analysis Tools

oprofile

- System-wide profiler
  - Kernel driver
  - Daemon for collecting data
  - Post-processing tools
- Leverages Intel hardware performance counters
  - Low overhead (1%-8%)

valgrind

- Debugging tool
  - Memory bugs
  - Threading bugs
  - Pluggable interface for building new tools
- Dynamically modifies running programs
  - Inserts instrumentation code
  - Collects data as program runs

Advantages of Open-Source Tools

Portability across architectures

- Tools work on non-Intel architectures too
- Easier to leverage investment in skills or software

Improvements from many sources

- Silicon companies
- Software developers
- University researchers

Great for research!

- Possible to change the tools

CodeSourcery & Intel

Activities for Intel CPUs

Performance Optimization

- Instruction selection
- Instruction scheduling

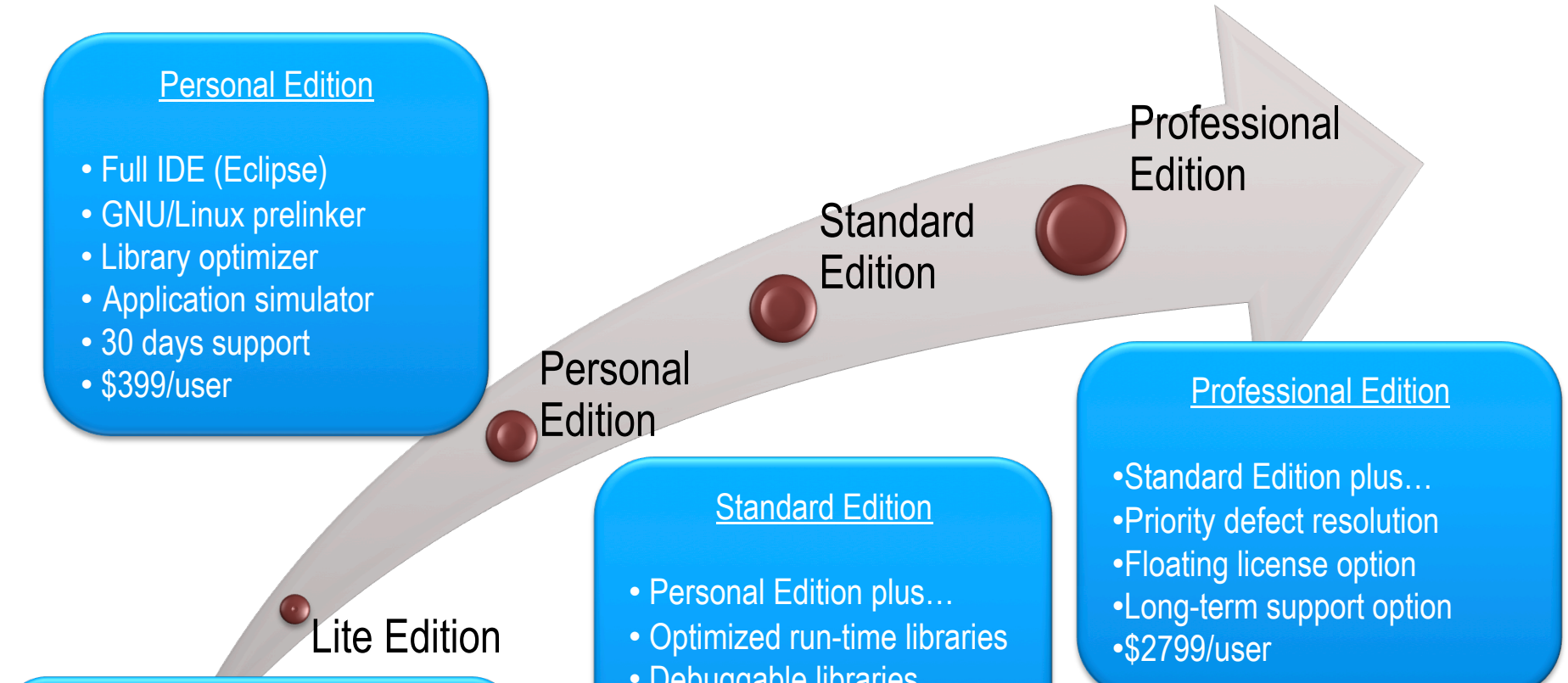
Embedded Functionality

- “Bare-metal” toolchains
- JTAG debug for Atom

Regular High-Quality Releases

- For GNU/Linux and bare-metal/RTOS platforms
- Zero-cost command-line tools
- Commercial packages available

Sourcery G++ Editions



Lite Edition

- Core command-line tools
- No support
- Zero-cost solution

Personal Edition

- Full IDE (Eclipse)
- GNU/Linux prelinker
- Library optimizer
- Application simulator
- 30 days support
- $399/user

Standard Edition

- Personal Edition plus…
- Optimized run-time libraries
- Debuggable libraries
- Unlimited support
- $1599/user

Professional Edition

- Standard Edition plus…
- Priority defect resolution
- Floating license option
- Long-term support option
- $2799/user

Future Directions I: Optimization

Optimization Opportunities

Traditional optimizations

- Loop optimizations
- Instruction scheduling
- SIMD auto-vectorization

Link-time optimization

- Overcome limitations of separate compilation
- Propagate link-time constants
- Inline across modules
- Align data on cache lines

Profile-directed feedback

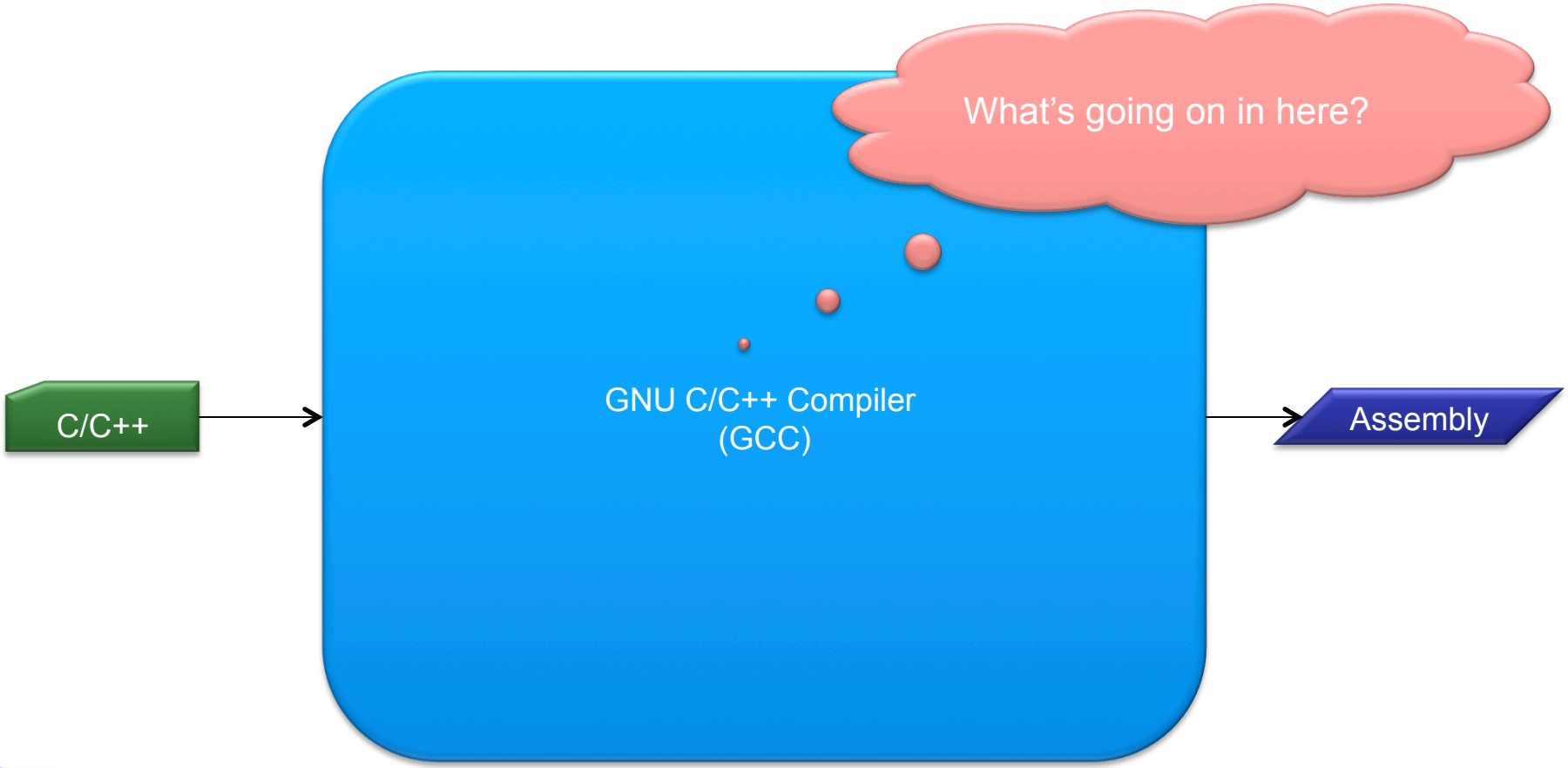
- Learn from program execution
- Optimize hot code for speed; cold code for space
- Layout program images to maximize cache performance
- Optimize for expected data values

Power optimization

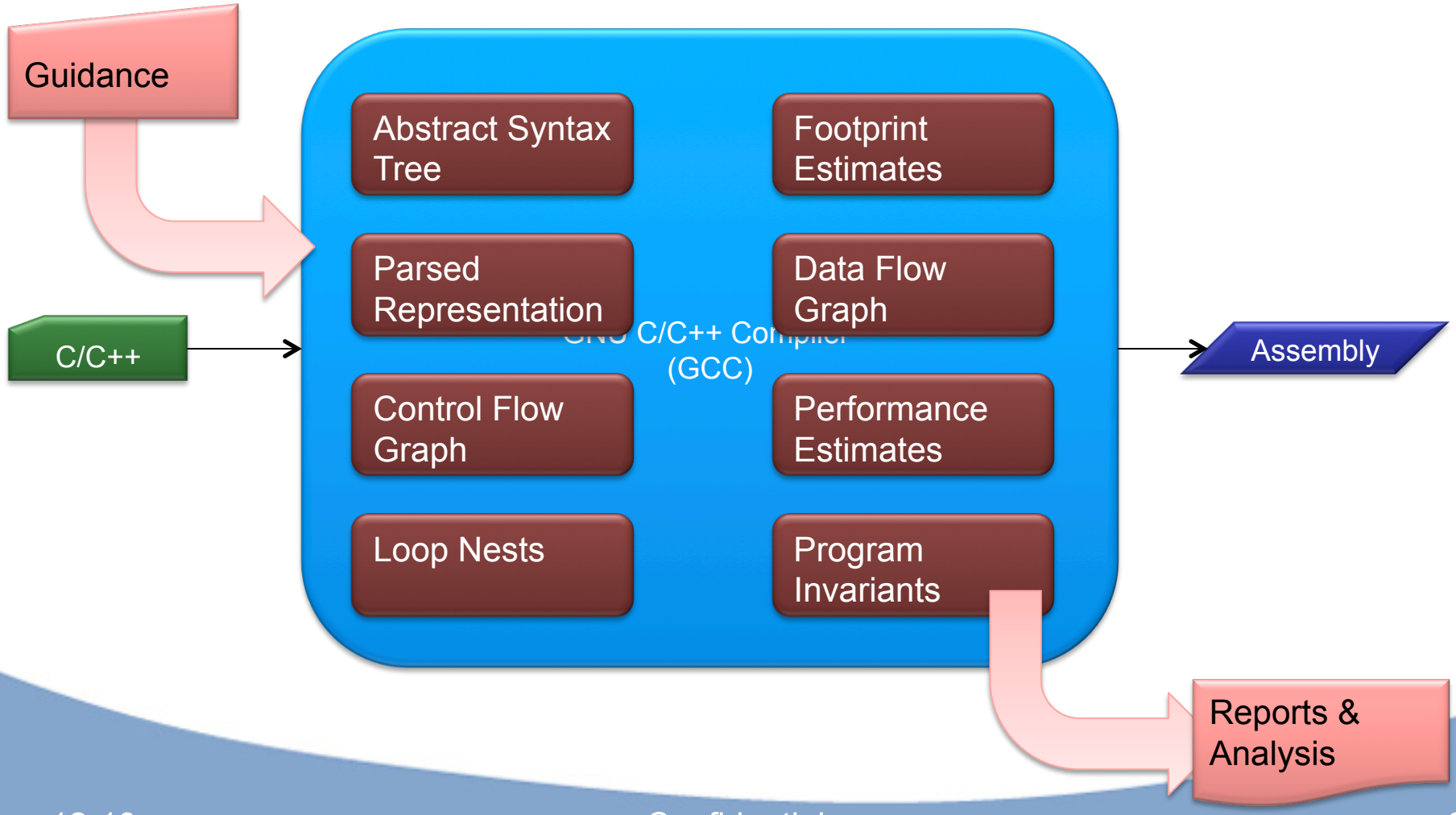
- GNU tools are blissfully unaware of power impact
- Choose low-power instructions
- Provide expected power consumption information

Future Directions II: Analysis

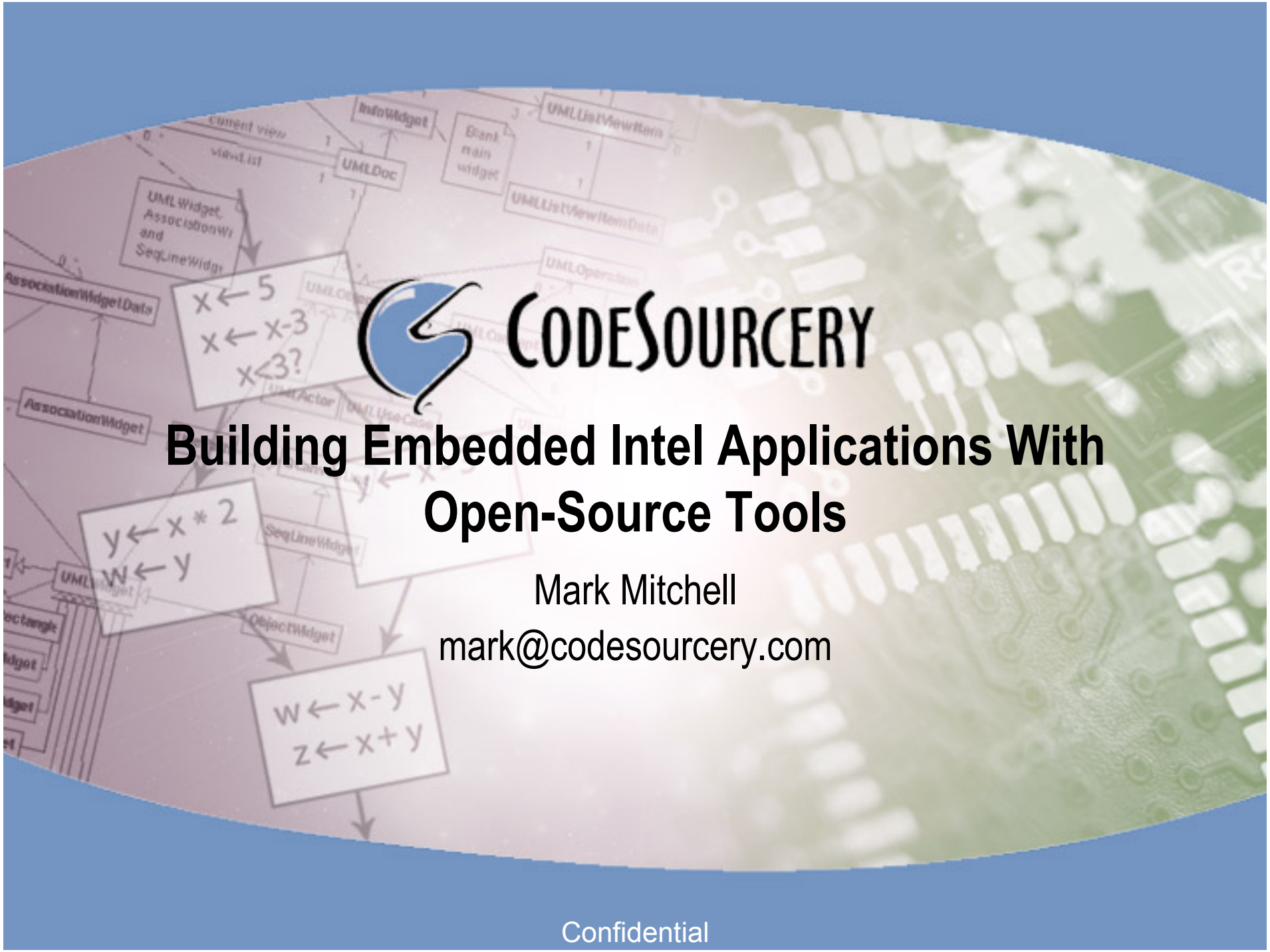
Compilers Are Black Boxes



Compilers Should Be White Boxes



Questions



Building Embedded Intel Applications With Open-Source Tools

Mark Mitchell
mark@codesourcery.com