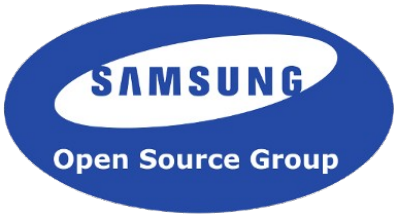


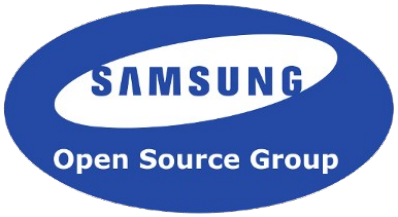
Internet of Smaller Things

More of Less is More

Jon A. Cruz
Samsung Open Source Group
jonc@osg.samsung.com

Why IoT?

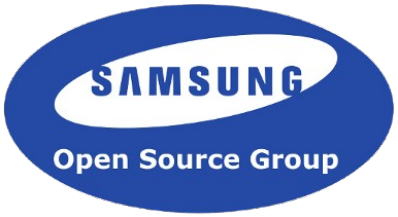
Why IoT?



- Simple answer:
  - It's coming in everywhere
  - Tipping point has been passed
  - Details left to other sources

Why IoTivity?

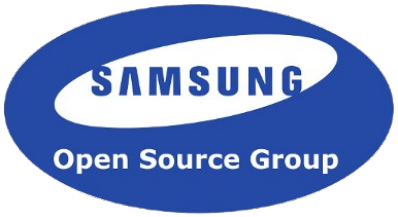
Why IoTivity?



- Open
  - Open Source
  - Open Standard
  - Open Governance
- Implementing OIC spec
- Licensing clarity
- Vendor agnostic

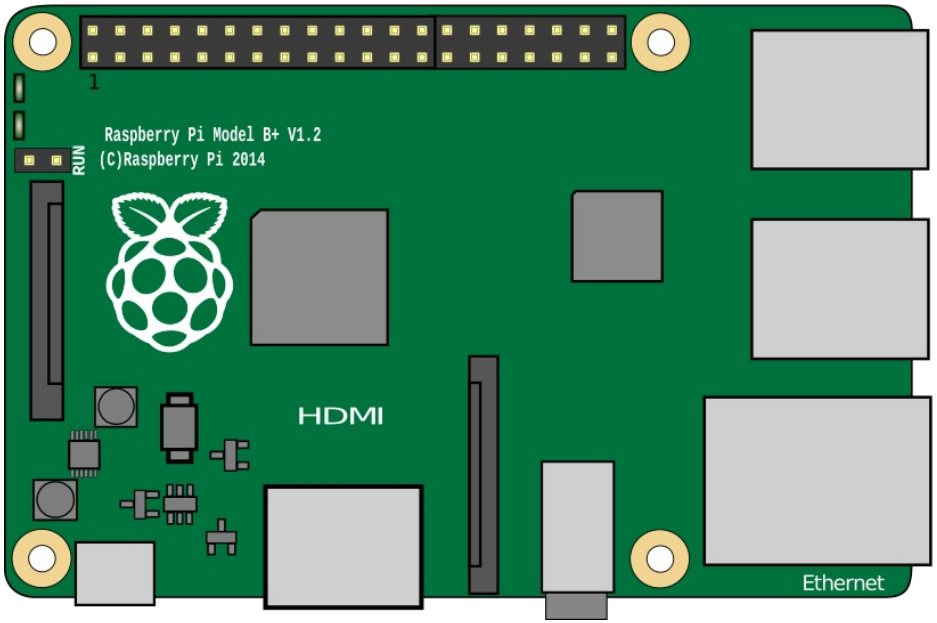
'Smaller' ?

What is 'Smaller' ?



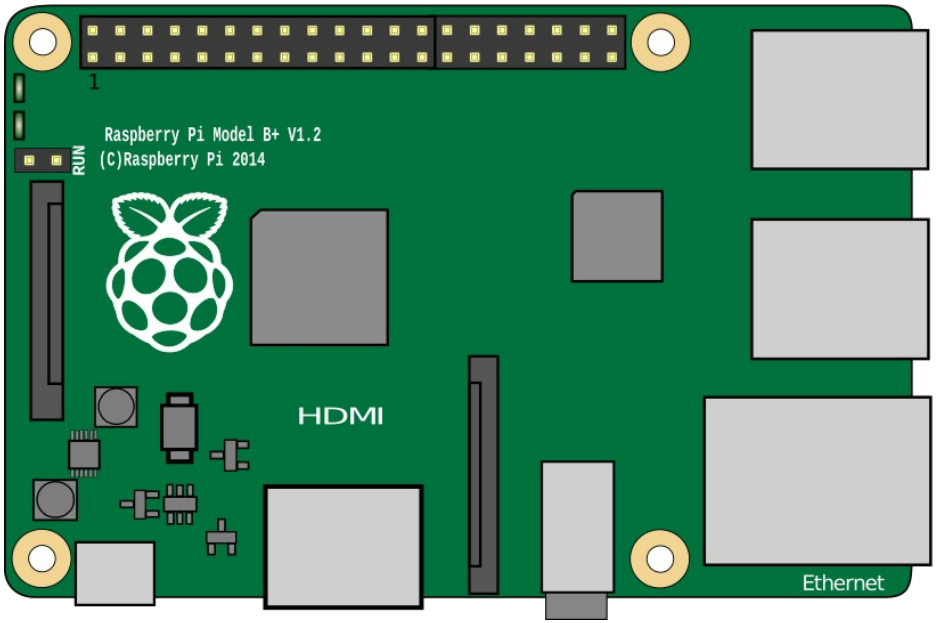
- Size
- Specs
- Data-sets
- Applications
  - Aka 'apps'
- But not impact

Smaller Size



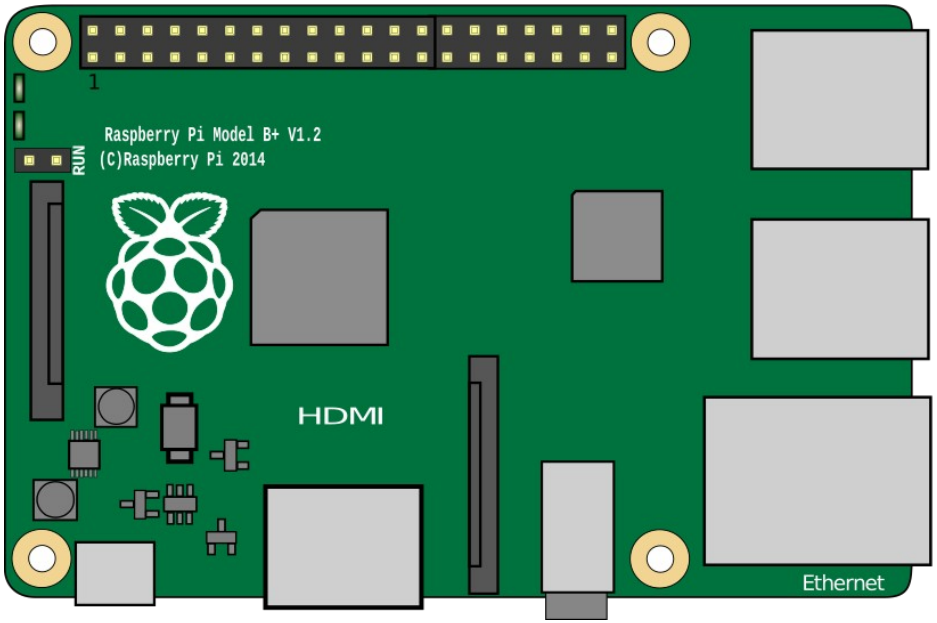
- 85.60 mm × 56.5 mm
- Not small enough

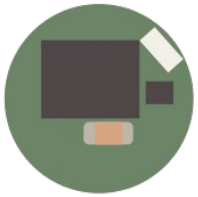
Smaller Size



- 35.5 mm × 25.0 mm
- Not small enough

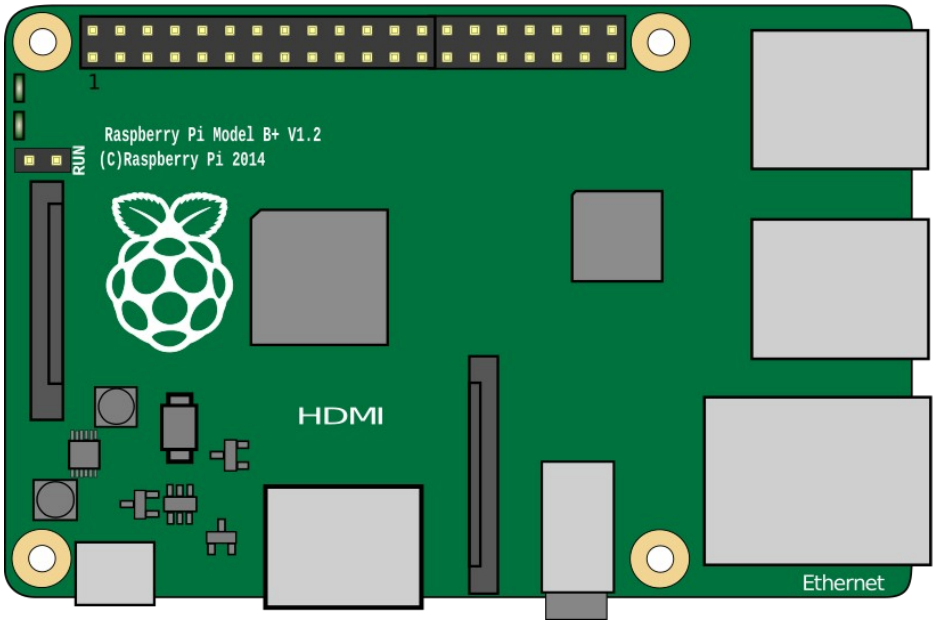
Smaller Size





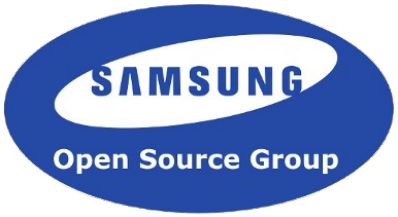
- 18 mm x 18 mm
- Not small enough

Smaller Size



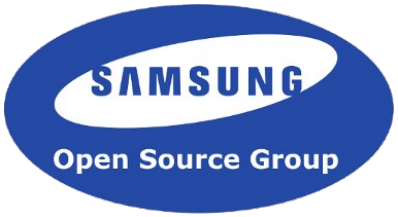
- 12 mm x 12 mm
- Getting there

Smaller Specs



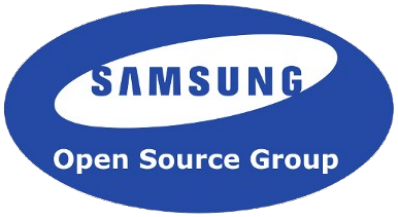
- Limited processor
  - 32-bit or even 8-bit
  - Often microcontroller
- Limited RAM
  - 1 GB, 1 MB or even down to 8 KB
- Limited storage
  - Flash
  - 4 GB, 4 MB or even down to 256 KB

Smaller Specs



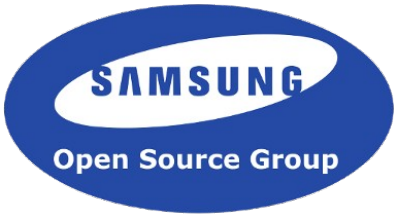
	RAM	Flash
Raspberry Pi 2	1 GB	4 GB
Edison	1 GB	4 GB
Artik-1	1 MB	4 MB
Curie	80 KB	384 KB
Arduino Due	96 KB	512 KB
Arduino Mega2560	8 KB	256 KB

Smaller Specs (unified)



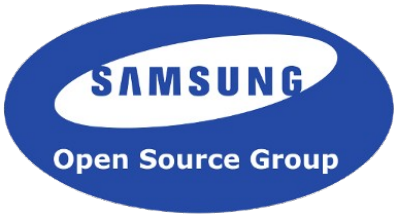
	RAM	Flash
Raspberry Pi 2	1024.000 MB	4096.000 MB
Edison	1024.000 MB	4096.000 MB
Artik-1	1.000 MB	4.000 MB
Curie	0.078 MB	0.375 MB
Arduino Due	0.094 MB	0.500 MB
Arduino Mega2560	0.008 MB	0.250 MB

Smaller Data-sets



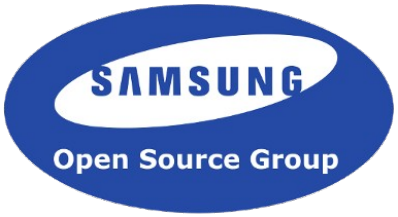
- Issues with data
  - Storage
  - Bandwidth
  - Connectivity
- Usefulness
- Aggregated is not smaller
- Privacy
  - Gets into business concerns, legal issues, etc.

Smaller Applications



- 'Apps' or programs
- Space limits complexity
- 'Targeted' hardware
  - Less general
  - More task-oriented

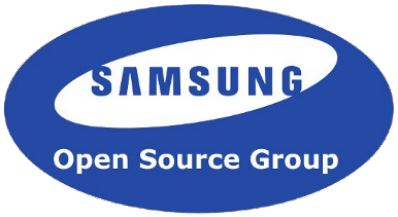
Smaller Devices Increasing



- Originally shrinking PCs
  - Leveraged existing Internet experience
- Now growing micro-controllers/modules
  - More contributions from 'Makers'
- Less is more
- More of less is more
  - Lots of smaller devices
  - Self-organizing

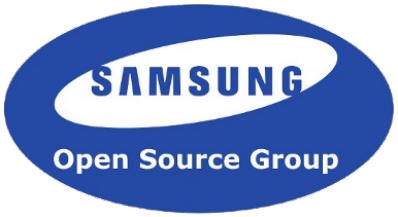
IoTivity Approach

IoTivity Approach



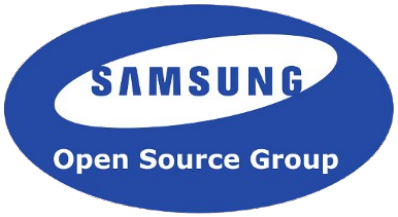
- Single, unified code-base
- Multi-platform
- Stacks and Layers

Single, unified code-base



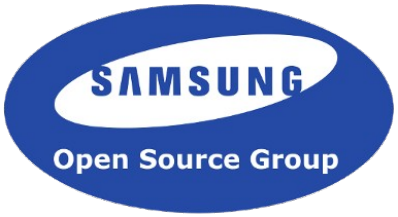
- Cross-platform
- Multiple tool chains
- Disparate platforms
- Edge cases to cover more

Multi-platform



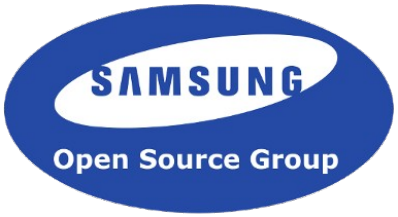
- Current
  - Linux
  - Arduino
  - Android
  - Tizen
- In-progress
  - OS X
  - iOS
  - MS Windows

Stacks and Layers



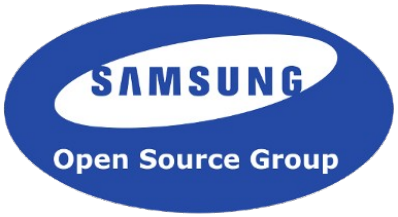
- Built upon each other
  - Core functionality by taking a subset of the code
  - Simplifies verification
  - Easier to correct and bug-fix
- Other solutions have separate code-bases
  - Many problems from this approach

Core is Common



- Implemented in C
- Some optional features
  - More features in higher layers

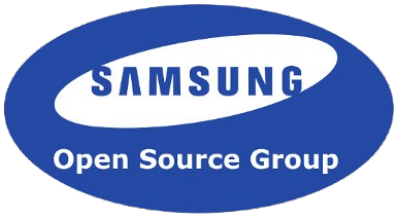
Layers on Top



- C++ layer adds functionality to the core
  - Ease of development
- Java wrapper layer
  - Same core library executing on Android
  - Exposed in Android friendly manner
  - Possible to implement in pure Java later

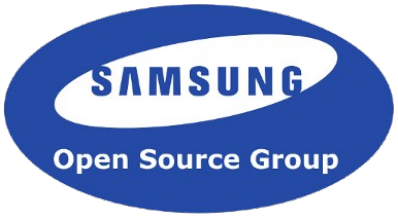
Classes of Platforms

Classes of Platforms



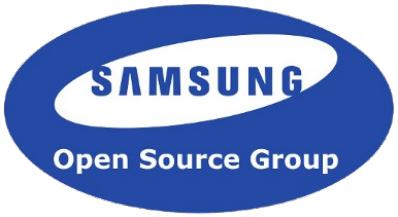
- Focus on common code
- Certain exceptions
- Groups of platforms
  - Linux
  - OS X
  - Tizen & Android
  - Arduino

Differences



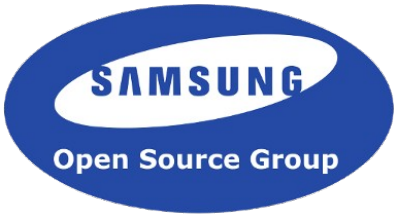
- Linux
- UNIX
- Tizen & Android
- Arduino

Linux



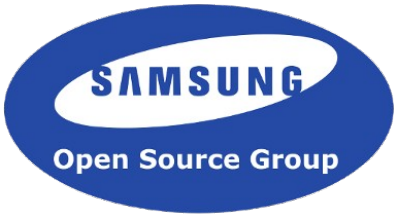
- Default target/assumptions
- Older and newer distros targeted

UNIX



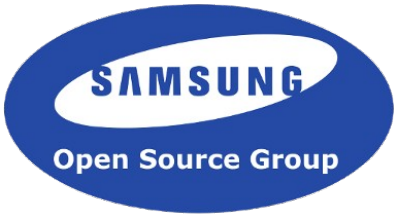
- BSD and OS X differ in certain ways
  - e.g. clock_gettime()
  - Generally covered by POSIX info

Tizen & Android



- Different in similar ways
- Both stripped down for devices
  - Network code is the most common divergence

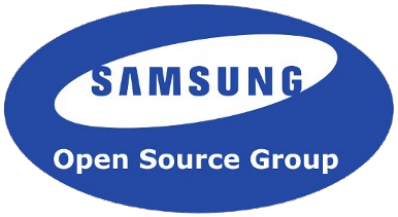
Arduino



- Represents device/micro-controller world
- Unique challenges
  - Logging can be especially troublesome
- Helps keep other platforms clean

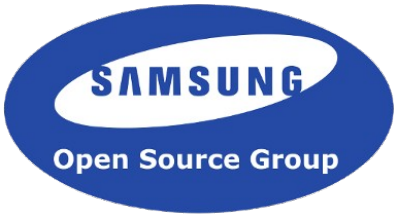
Build System

Build System



- Implemented as a 'build', not just scripted
  - Setup targets, dependencies, etc.
  - Allow the system to do the heavy lifting
- Mainly SCons
  - Subs out to other systems
    - Android
    - Tizen
- Set dependencies correctly

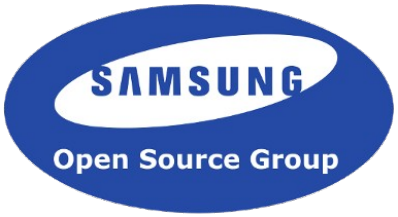
Cross-compile



- Build for different platform than the current
- Allows one system to check many
  - e.g. all devs can validate changes didn't break other platforms
- Improves developer habits and code quality

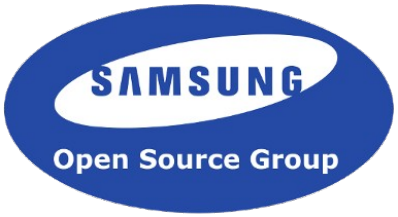
Architectural Issues

Split between C & C++ stacks



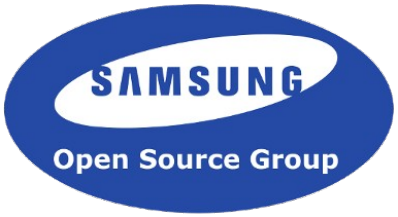
- Simpler/required functionality in C
- Complexity and helpers in C++
- C++ stack build on top of C
- Java wrapping C/C++

Focus on Protocol/Messaging



- Initially CoAP
  - Abstracted and potentially replaceable
- Abstracted stack from transport
  - Higher levels avoid transport and protocol details
  - Lower levels implement individual transports

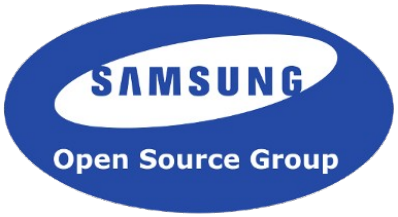
Threading



- Threading not supported on all platforms
- Details have been abstracted
  - Internal functions and structs for threads, mutexes, etc.
- Work in progress
  - Moving threading out of lower levels and into higher
  - Tuning and synchronization optimization

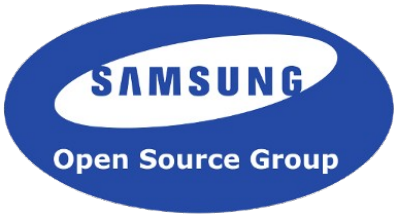
Coding Design & Practices

Coding Design & Practices



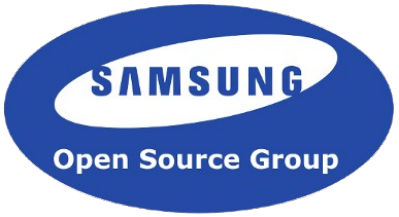
- As cross-platform as possible
- Minimize code duplication
  - #ifdef for certain functionality
    - Focus on feature not platform/OS
  - Files excluded from other situations
    - Platform dependent code
    - Specific transports
    - Had to reduce file duplication

Platform Specifics



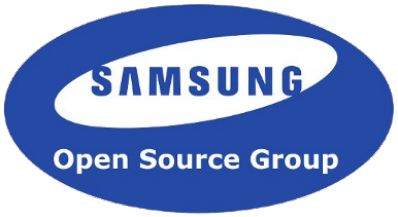
- Languages in use
- Compiler versions
  - Bugs
- Tools available
- APIs/functions available
  - Tizen/Android networking
- Unique aspects

Arduino



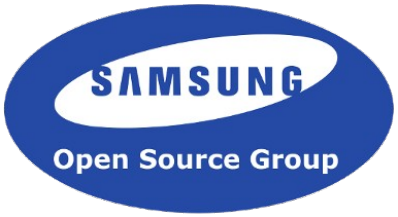
- Standard tool-chain
- Minimal capabilities
- Constants/strings in flash
  - PROGMEM
    - Needs explicit calls to access
  - F()
    - Wraps string constants
    - IoTivity added to logging macros

C Practices



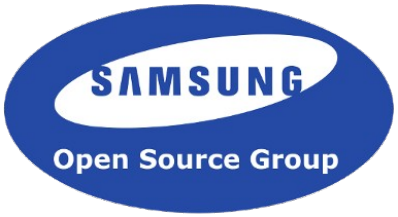
- Minimize memory use
  - Flash/program size
  - Runtime RAM
- Minimize duplicated code
- Follow standard functions
  - Look at POSIX + extensions
  - Use 'man' liberally

C Practices



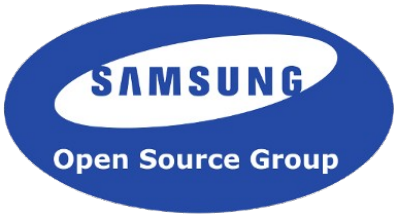
- Be explicit on target spec
  - C99
    - -std=c99
  - Avoid compiler extensions when possible
    - minimize -std=gnu99

C Practices



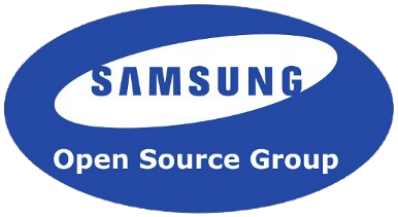
- Enable and watch warnings
  - Know which ways to get desired warnings
    - For gcc, -Wall -Wextra
  - Do not ignore warnings
  - Use fatal warnings when possible
- Fix, don't suppress!!!

C Practices



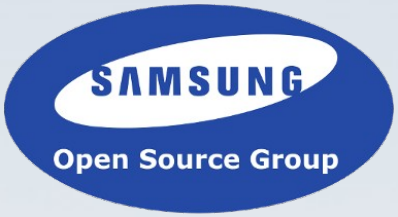
- Use reentrant functions
  - e.g. strtok_r() instead of strtok()
- Pass by reference
  - Watch for structs passed by value
  - Prefer refs/pointers
- Use 'const'
  - Especially on pointer parameters
  - Easy to remove, but hard to add later

C Practices



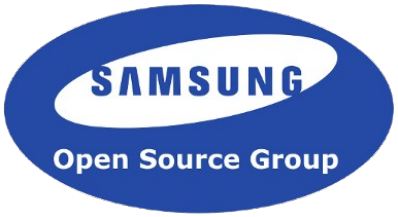
- Use 'static'
  - Limit visibility and impact of file-local globals
- Avoid use of globals
  - Eliminate when possible
  - Replace with context pointers
  - Leverage reentrancy
    - Follow practices of POSIX foo_r() functions

Questions?



Thank You!

References



- Credits
  - Images
    - Original Raspberry Pi drawing CC BY-SA 4.0 Efa2
      - https://commons.wikimedia.org/wiki/File:Raspberry_Pi_B%2B_rev_1.2.svg