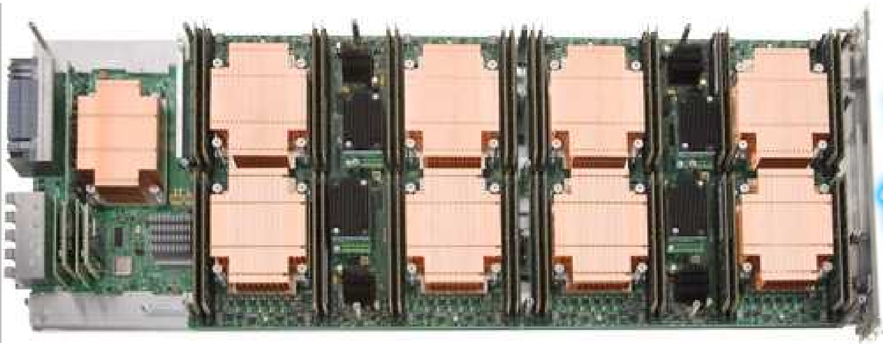
SAND2014-19796C

Exceptional service in the national interest

Sandia National Laboratories



A Prototype of a Power API Framework

Speaker: David DeBonis

Power API Team: James H. Laros III (Lead), Kevin Pedretti, Suzanne M. Kelly, Micheal Levenhagen, David DeBonis, Stephen Olivier, Ryan E. Grant

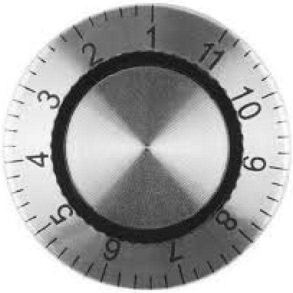
U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Images courtesy of: Cray Inc.
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2014-15182PE

Motivation and Foundation

- Extreme Scale Power / Energy Aware Computing
  - Within power budgets
  - Turning it up to “eleven” is not maintainable
  - Power capping will be the norm

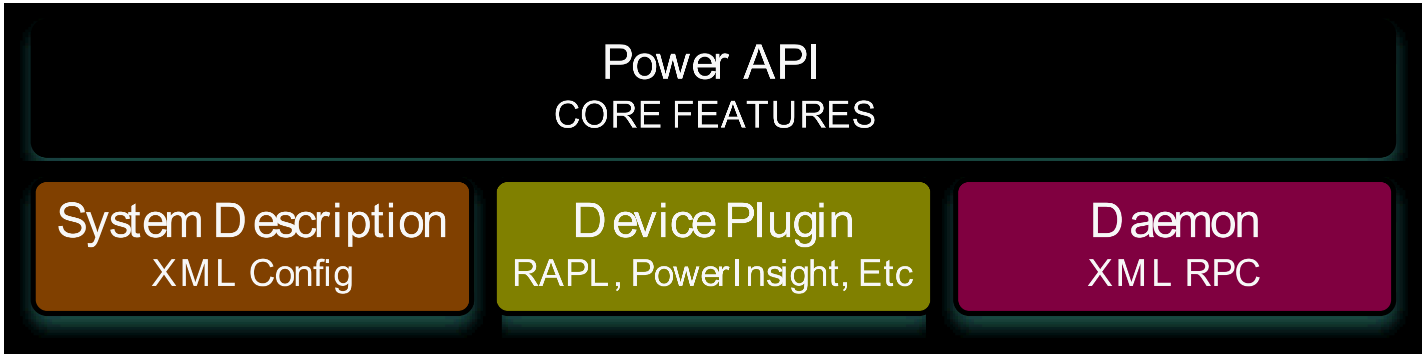


- Power API Specification
  - Effort at SNL to influence future systems procurement
  - Facilitate power-aware and energy-efficient computing at scale
  - Vendor and platform neutral
  - Unified measurement and control interface
  - Traversable hierarchy

Objectives

- Power API Prototyping Effort
  - Solidifying the Power API specifications
  - Enable communication between vendors and community
  - Explore implementation concepts
  - DISCLAIMER: NOT meant to be a production implementation

- Power API Prototype Framework



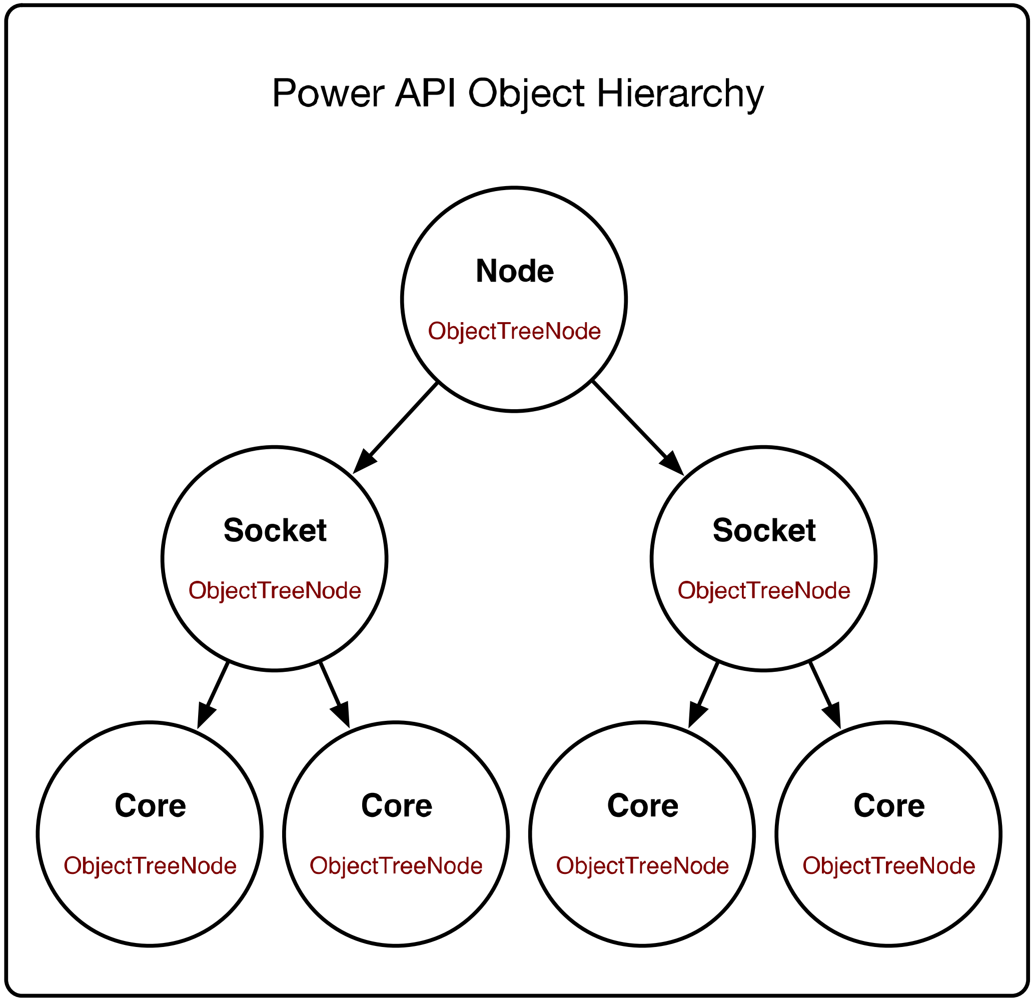
Power API
CORE FEATURES

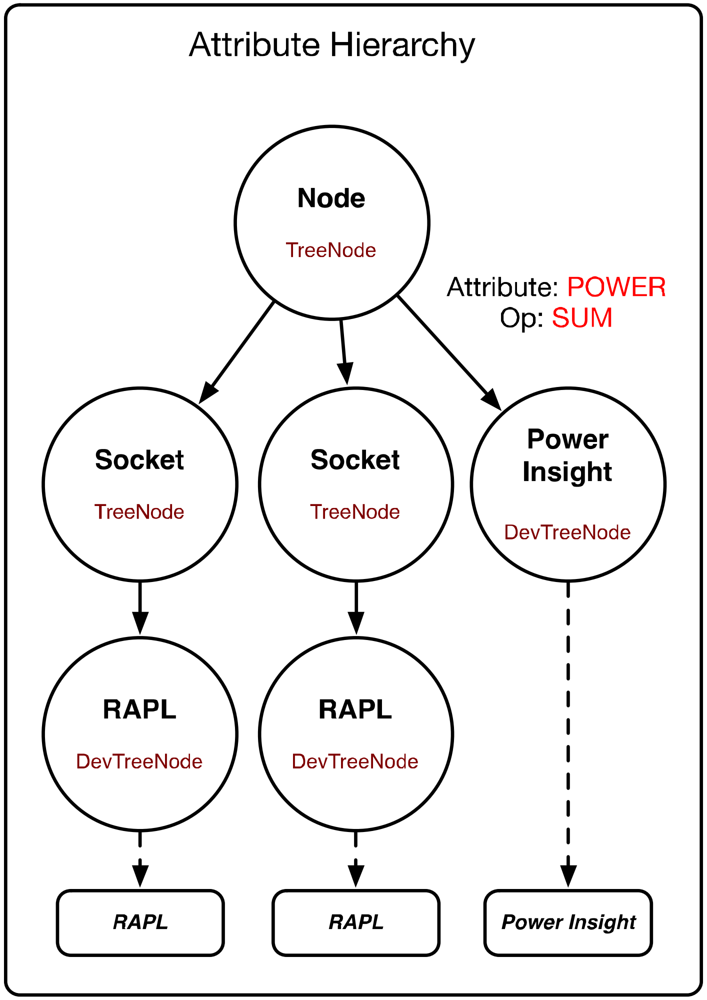
System Description
XML Config

Device Plugin
RAPL, PowerInsight, Etc

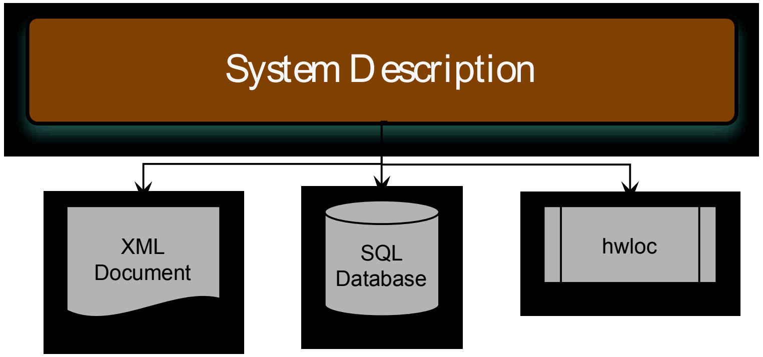
Daemon
XML RPC

Prototype Framework



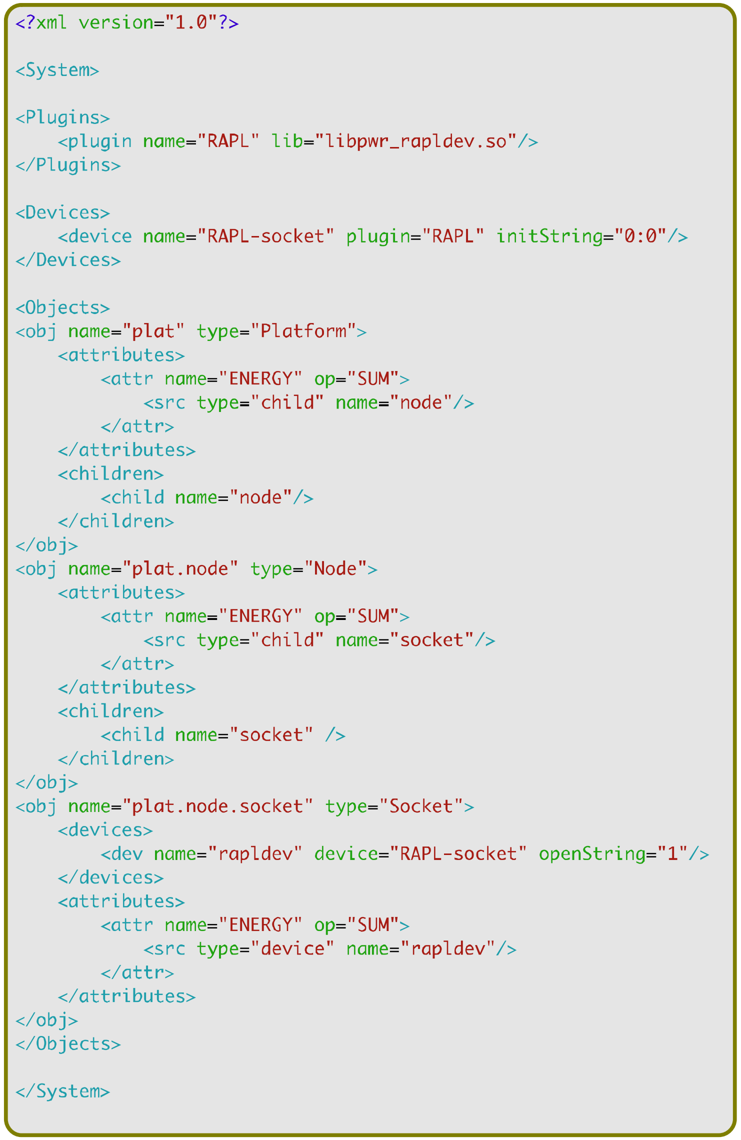


System Configuration Description



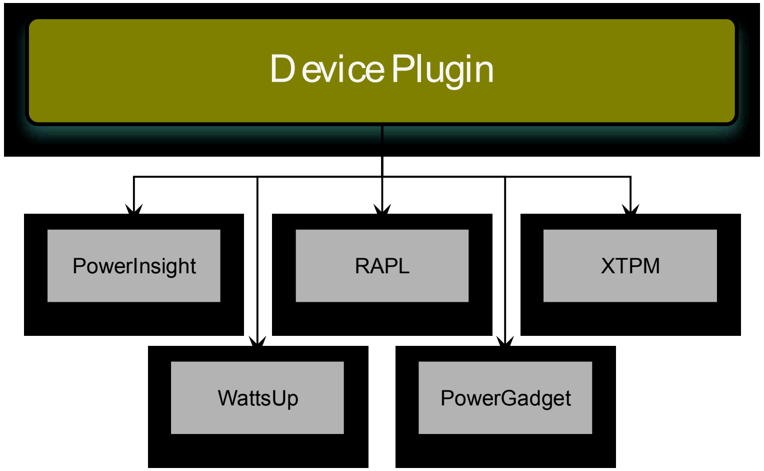
- Dynamically configurable system
  - Not tied to specific system
  - Definable or discoverable
  - Representation neutral

- XML based configuration file
  - Hierarchy and relationships
  - Plugins and mappings
  - Operation on attributes

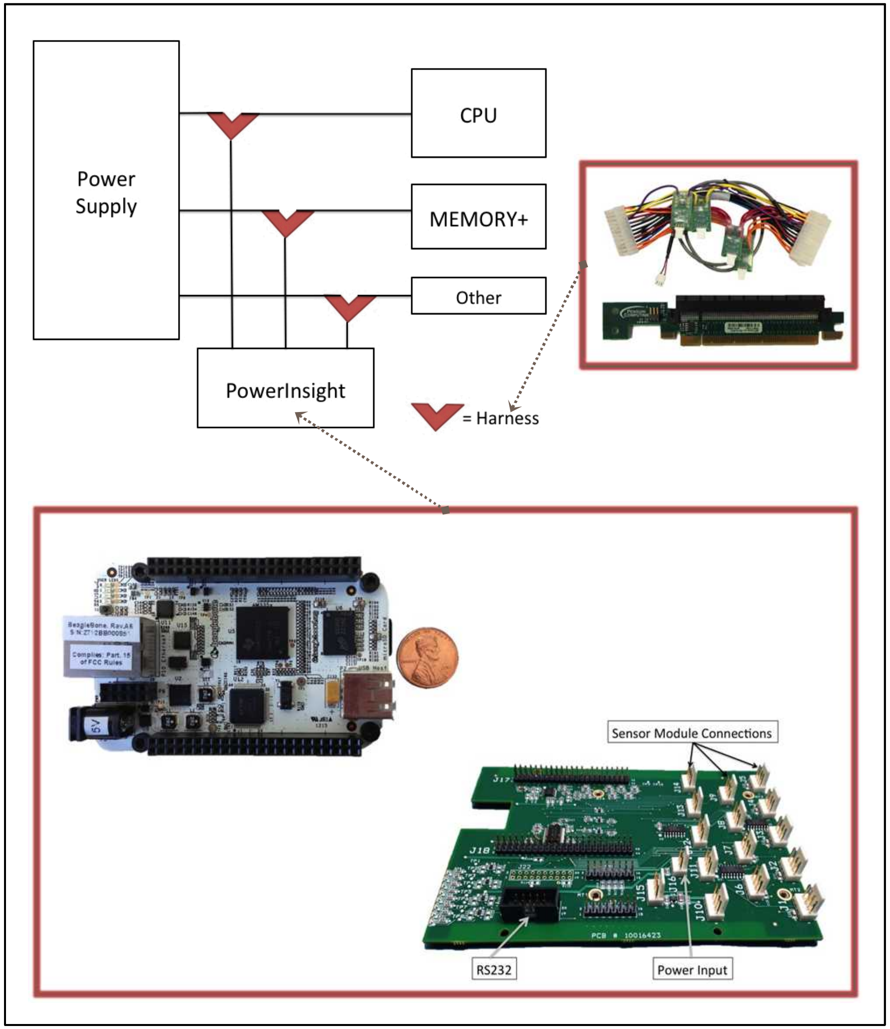


```xml
<?xml version="1.0"?>

<System>

<Plugins>
    <plugin name="RAPL" lib="libpwr_rapldev.so"/>
</Plugins>

<Devices>
    <device name="RAPL-socket" plugin="RAPL" initString="0:0"/>
</Devices>

<Objects>
<obj name="plat" type="Platform">
    <attributes>
        <attr name="ENERGY" op="SUM">
            <src type="child" name="node"/>
        </attr>
    </attributes>
    <children>
        <child name="node"/>
    </children>
</obj>
<obj name="plat.node" type="Node">
    <attributes>
        <attr name="ENERGY" op="SUM">
            <src type="child" name="socket"/>
        </attr>
    </attributes>
    <children>
        <child name="socket" />
    </children>
</obj>
<obj name="plat.node.socket" type="Socket">
    <devices>
        <dev name="rapldev" device="RAPL-socket" openString="1"/>
    </devices>
    <attributes>
        <attr name="ENERGY" op="SUM">
            <src type="device" name="rapldev"/>
        </attr>
    </attributes>
</obj>
</Objects>

</System>
```

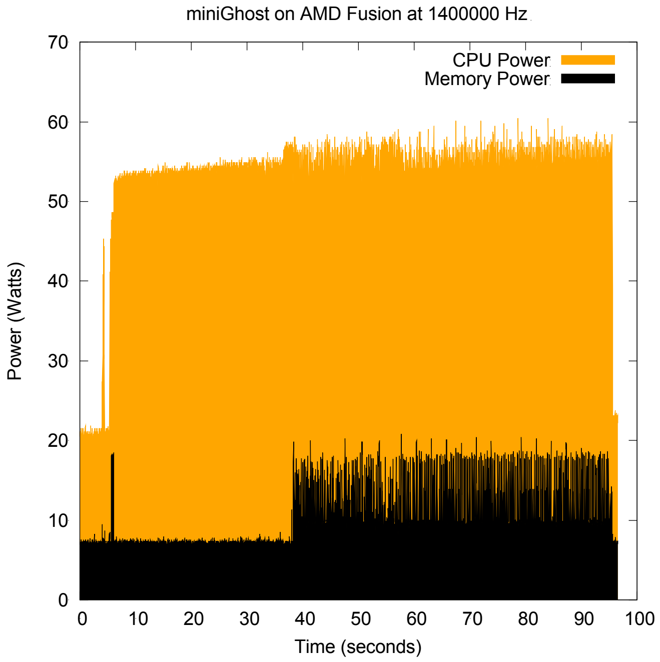
Plugin Capabilities

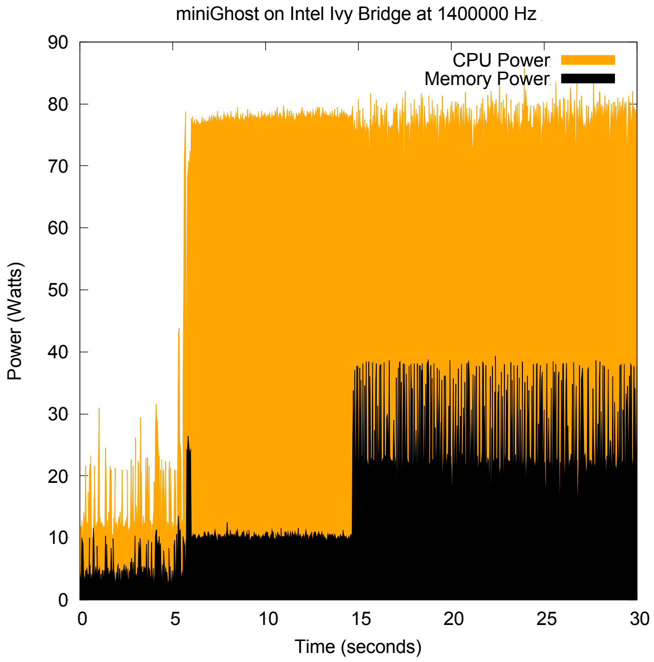


- Well-defined interface
  - POSIX-like semantics
- Penguin PowerInsight
  - Fine-grain per component
  - Out-of-band collection
  - Extended capabilities
    - offloaded calculations
    - embedded analysis



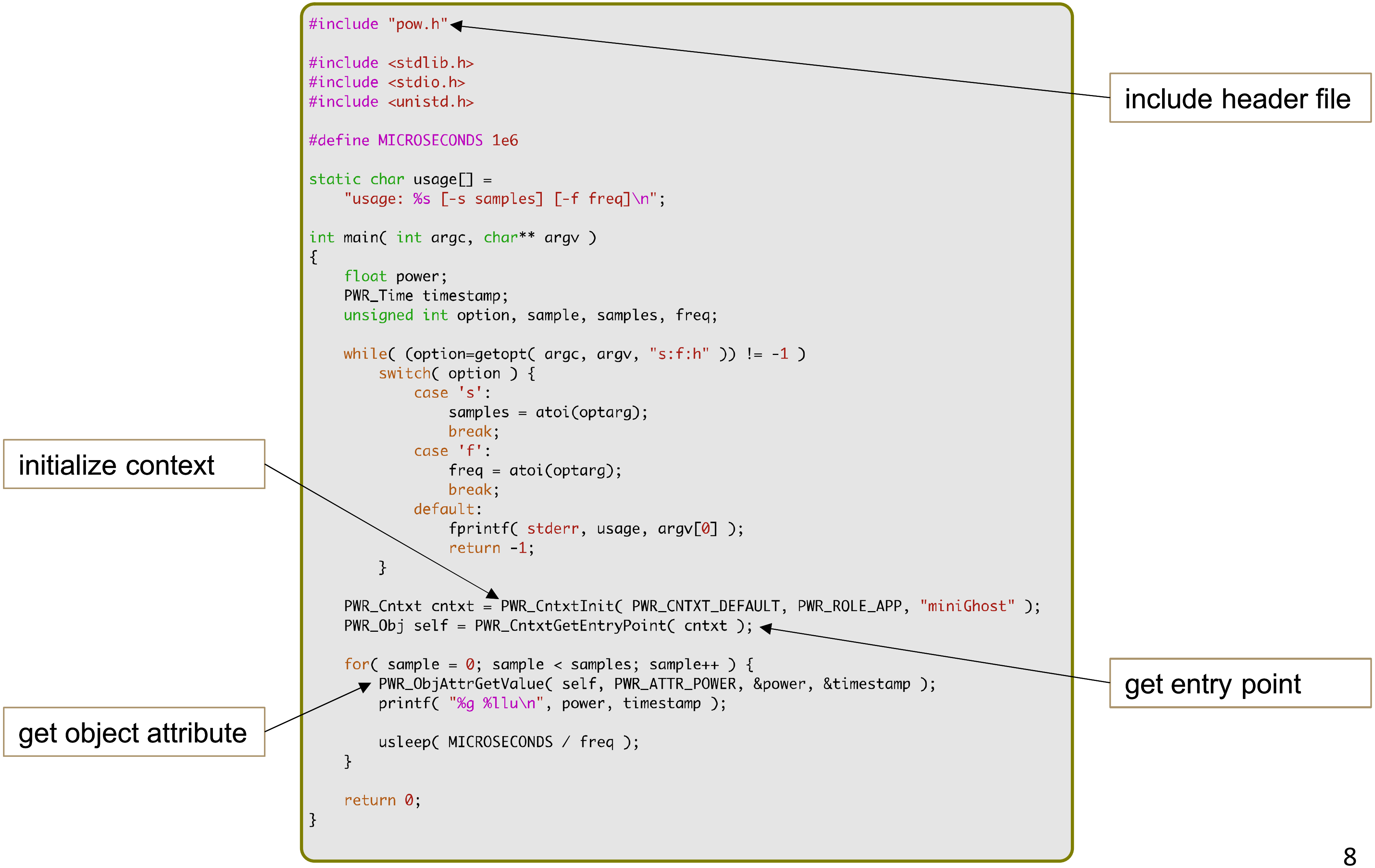
Study - Application Characterization





- Energy and power characteristics at various CPU frequencies
  - Frequency scaling using the Mantevo miniApps
  - Performed on Advanced Architecture Test Beds
  - Architectures: AMD APU, Intel Ivy Bridge, Haswell, Phi, Power8, ARM
  - Plugins: PowerInsight, WattsUp, XTPM, RAPL, CPU

Multi-Architecture Study Simplified



```c
#include "pow.h"

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define MICROSECONDS 1e6

static char usage[] =
    "usage: %s [-s samples] [-f freq]\n";

int main( int argc, char** argv )
{
    float power;
    PWR_Time timestamp;
    unsigned int option, sample, samples, freq;

    while( (option=getopt( argc, argv, "s:f:h" )) != -1 )
        switch( option ) {
            case 's':
                samples = atoi(optarg);
                break;
            case 'f':
                freq = atoi(optarg);
                break;
            default:
                fprintf( stderr, usage, argv[0] );
                return -1;
        }

    PWR_Cntxt cntxt = PWR_CntxtInit( PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "miniGhost" );
    PWR_Obj self = PWR_CntxtGetEntryPoint( cntxt );

    for( sample = 0; sample < samples; sample++ ) {
        PWR_ObjAttrGetValue( self, PWR_ATTR_POWER, &power, &timestamp );
        printf( "%g %llu\n", power, timestamp );

        usleep( MICROSECONDS / freq );
    }

    return 0;
}
```

include header file

initialize context

get entry point

get object attribute

Future Directions (in the works)

- Power API Core support
  - complete the statistics interface
  - complete the meta-data interface
- Additional Power API prototype plugins
- Python bindings
  - objects and attributes
- Power API Prototype at scale
  - distribution and aggregation

Acknowledgements

Power API Team: James H. Laros III (Lead), Kevin Pedretti, Suzanne M. Kelly, Micheal Levenhagen, David DeBonis, Stephen Olivier, Ryan E. Grant

- Power API
  - http://powerapi.sandia.gov
  - SC14 BoF – Power API for HPC (Nov. 19th from 12:15 – 1:15pm)
  - SC14 Poster – A Power API for the HPC Community
- Penguin Computing
  - http://penguincomputing.com
- Advanced Architecture Test Beds
  - http://www.sandia.gov/asc/computational_systems/HAAPS.html

This work was funded through the Computational Systems and Software Environment sub-program of the Advanced Simulation and Computing Program funded by the National Nuclear Security Administration

Advanced Architecture Test Beds

Host Name	Nodes	CPU	Accelerator / Co-Processor
Volta	56	Dual Socket Intel Xeon E5-2695 v2 (Ivy Bridge) 2.4 GHz 12-core	N / A
Compton	42	Dual Socket Intel Xeon E5-2670 (Sandy Bridge) 2.6 GHz 8-core	Intel Xeon Phi (2x) 1.1 GHz 57-core
Teller	104	AMD A10-5800K (Piledriver) 3.8GHz 4-core	Radeon HD-7660D 800MHz 384-core