Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Save the Planet!
Go Green
using Java technology

Cory Sharp
Sentilla Corporation


sentilla
Perk
Pervasive Computing Kit
Quick S
sentilla
Perk
sentilla
sentilla
sentilla
Installation
Disc
Version 1.1
Sentilla Corporation
ghts reserved.
sentilla.com
Perk
Pervasive Computing

sentilla labs

Featured Projects

LCD Display Driver
by jason
A driver for the BPK-216 LCD display.

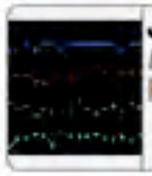
Fridge Monitor
by cmerlin
Keep Cool

Check Gmail with JCreates
by dan
Gmail, in Mote Form

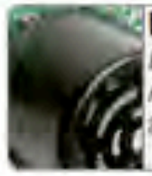
Environmental Sensors
by andrew
Environmental Sensors

Wireless Pedometer
by andrew
Count your steps and report back.

Pong
by dan
It's Pong, in Pong Form



JCreate Charting
by dan
Plot mote data with using processing.



Ultrasonic Distance Finder
by dan
A Maxbotix LV-EZ4 attached to a JCreate and visualized via Processing

Bathroom Monitor
by andrew
Know if the bathroom is in use before you walk all the way over there

Roll a dice
by andreas
Shake your mote to roll the dice...

Ever lost someone or something?
by andrew
This project shows how to use the JCreates to find each other

Another Coaster Application
by cmerlin
Hang on to your motes!

WELCOME

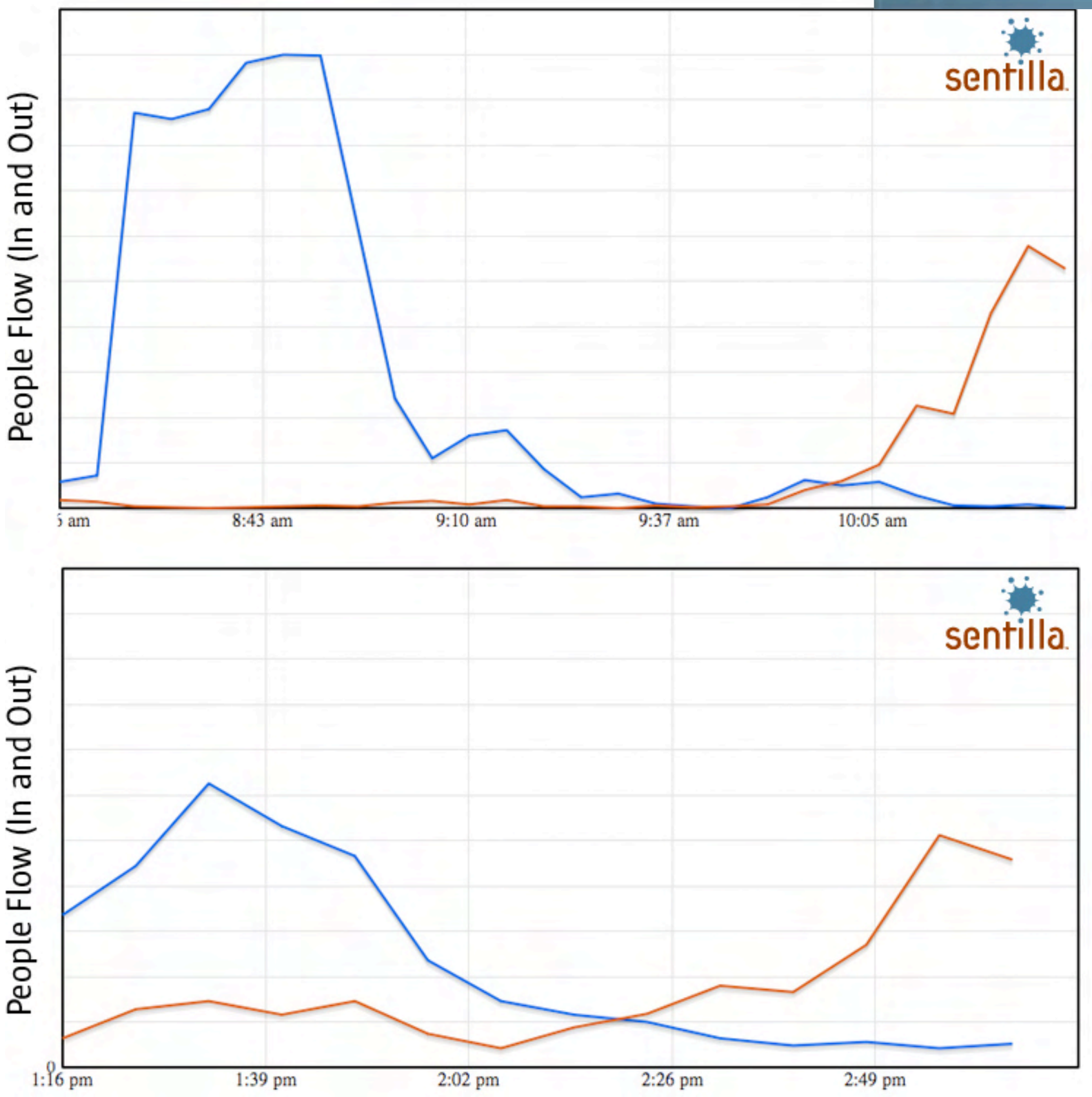
Sentilla Labs showcases the latest, coolest pervasive computing applications built by developers using Sentilla's technology. Share your ideas and be inspired by others.

PERK

Perk is Sentilla's all-in-one pervasive computing kit. Every project on Sentilla Labs was built using Perk. To start building your own application grab a **Perk** kit today.

TAGS

5V alarm boost charting dice display distance drinking energy environment fitness fridge **game** gmail graph **green** humidity java **jcreate** jtable lcd mail maxbotix notification printing **processing** random reed switch RSSI **security** sensor swing **temperature** time series Tracking



JavaOne JAVA + YOU

Sun General Session

Rich Green
Executive Vice President, Software
Sun Microsystems, Inc.

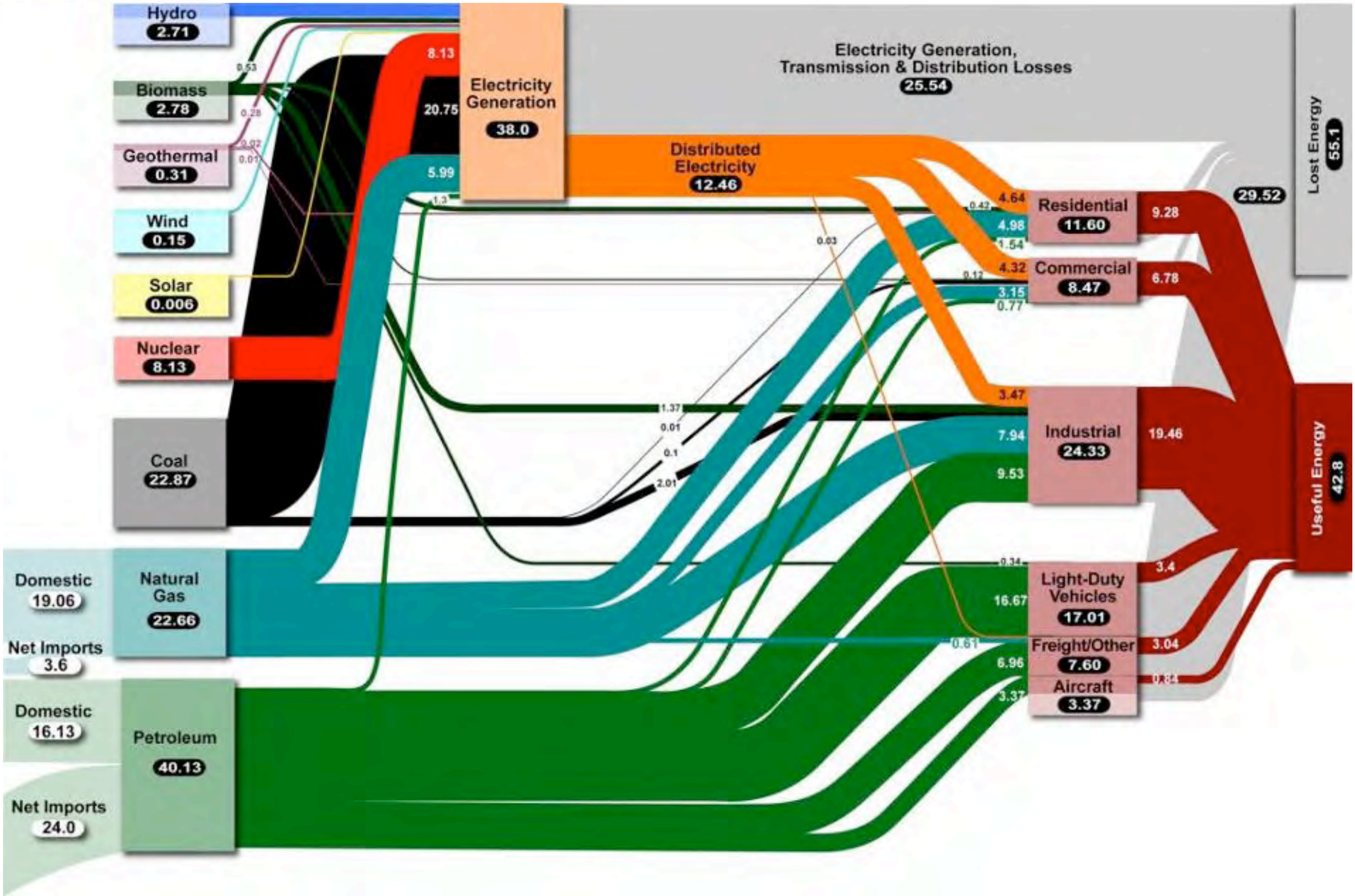
Java + YOU
Tuesday, May 6th
8:30 am - 10:30 am Halls B & C

Sun General Session

Robert Brewin
Distinguished Engineer and Chief Technology Officer, Software
Sun Microsystems, Inc.

Java-Centricity: Leveraging Java Technology at the hub of your Digital Life
Tuesday, May 6th
1:30 pm - 3:00 pm Halls B & C


Transportation
27%
Residential
18%
Commercial
20%
Industrial
34%


Hydro
2.71
Biomass
2.78
Geothermal
0.31
Wind
0.15
Solar
0.006
Nuclear
8.13
Coal
22.87
Domestic
19.06
Net Imports
3.6
Natural Gas
22.66
Domestic
16.13
Net Imports
24.0
Petroleum
40.13
0.53
0.28
0.02
0.01
8.13
20.75
5.99
1.3
Electricity Generation
38.0
Electricity Generation, Transmission & Distribution Losses
25.54
Distributed Electricity
12.46
1.37
0.01
0.1
2.01
0.03
0.42
0.12
0.34
0.61
4.64
4.98
1.54
4.32
3.15
0.77
3.47
7.94
9.53
16.67
6.96
3.37
Residential
11.60
Commercial
8.47
Industrial
24.33
Light-Duty Vehicles
17.01
Freight/Other
7.60
Aircraft
3.37
9.28
6.78
19.46
3.4
3.04
29.52
Lost Energy
55.1
Useful Energy
42.8


Residential
18%

Pacific Gas and Electric Company

WE DELIVER ENERGY.™

ELECTRIC ACCOUNT DETAIL

Service ID #: 2013989307
Rate Schedule: E1 TB Residential Service
Billing Days: 32 days

Serial	Rotating Outage Blk	Meter #	Prior Meter Read	Current Meter Read	Difference	Meter Constant	Usage
G	50	32L596	47,315	47,602	287	1	287 Kwh

Charges

08/02/2008 - 09/02/2008

Electric Charges				$33.51
Baseline Quantity	265.60000 Kwh			
Baseline Usage	265.60000 Kwh	@	$0.11559	
101-130% of Baseline	21.40000 Kwh	@	$0.13142	
Net Charges				$33.51

100+ Watts

10 Watts (!!!)

Internet

0W
100% efficient

200W
38% efficient

85W
20% efficient

only run at night

Pacific Gas and Electric Company

WE DELIVER ENERGY.™

ELECTRIC ACCOUNT DETAIL

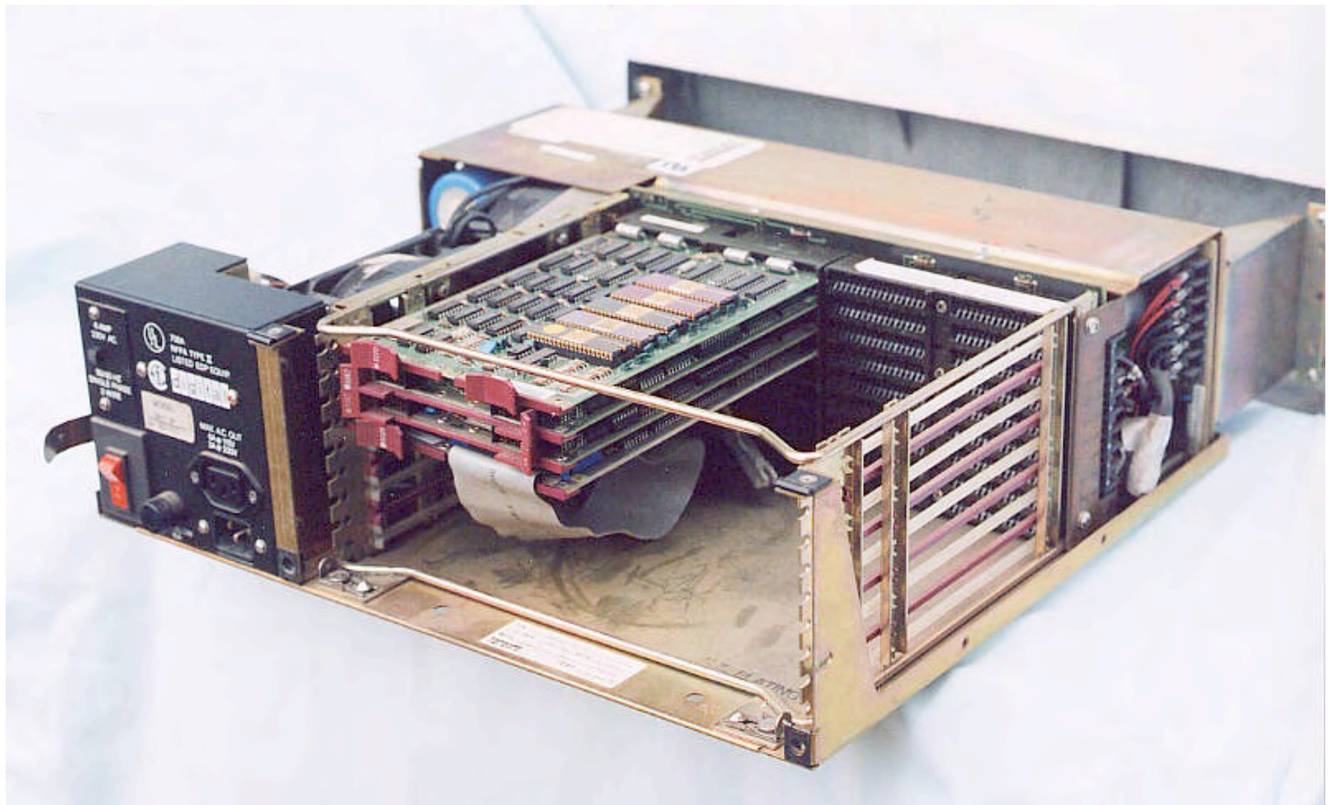
Service ID #: 2013989307
Rate Schedule: E1 TB Residential Service
Billing Days: 32 days

Serial	Rotating Outage Blk	Meter #	Prior Meter Read	Current Meter Read	Difference	Meter Constant	Usage
G	50	32L596	47,315	47,602	287	1	287 Kwh

Charges

Television	120.3 Kwh	$13.83
Refrigerator	50.2 Kwh	$5.77
Dishwasher	10.1 Kwh	$1.16
Computer	12.2 Kwh	$1.40
Clothes Dryer	35.2 Kwh	$4.05
Not Controlled	59.0 Kwh	$6.79

Evolution



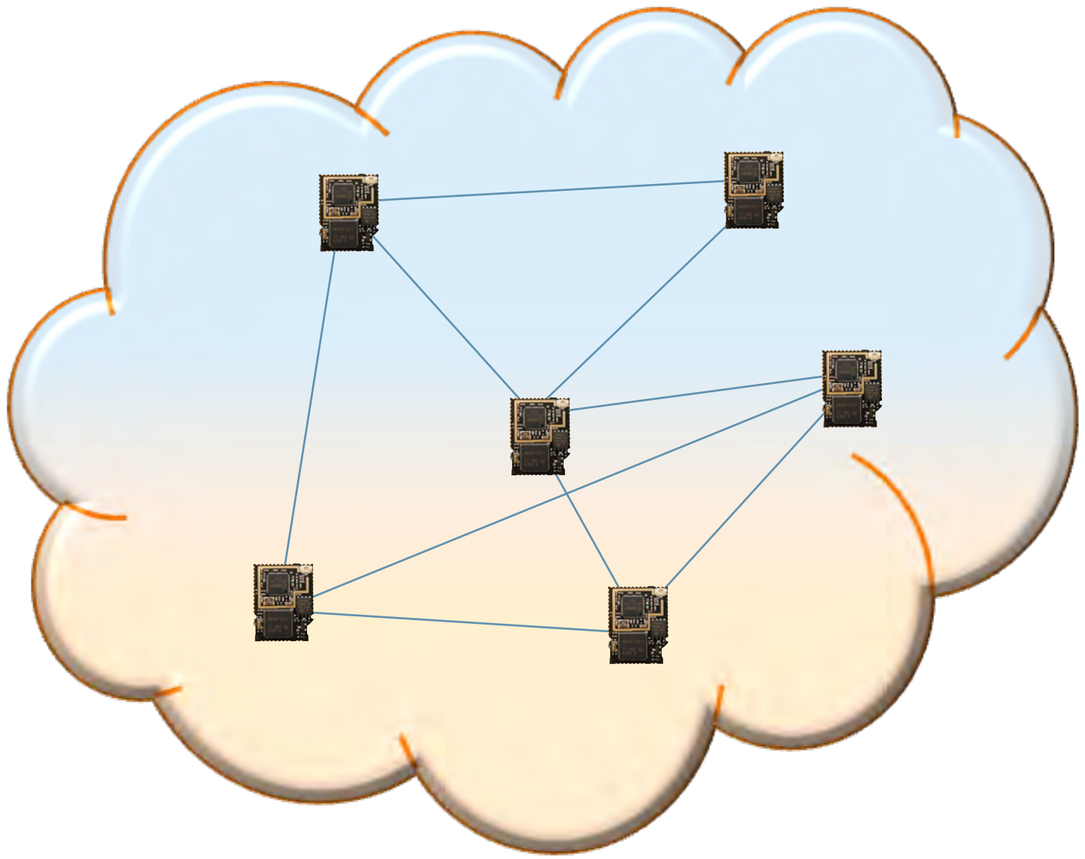
PDP-11
16-bit
64 kB
c. 1970

Mote
16-bit
48 kB
c. 2004

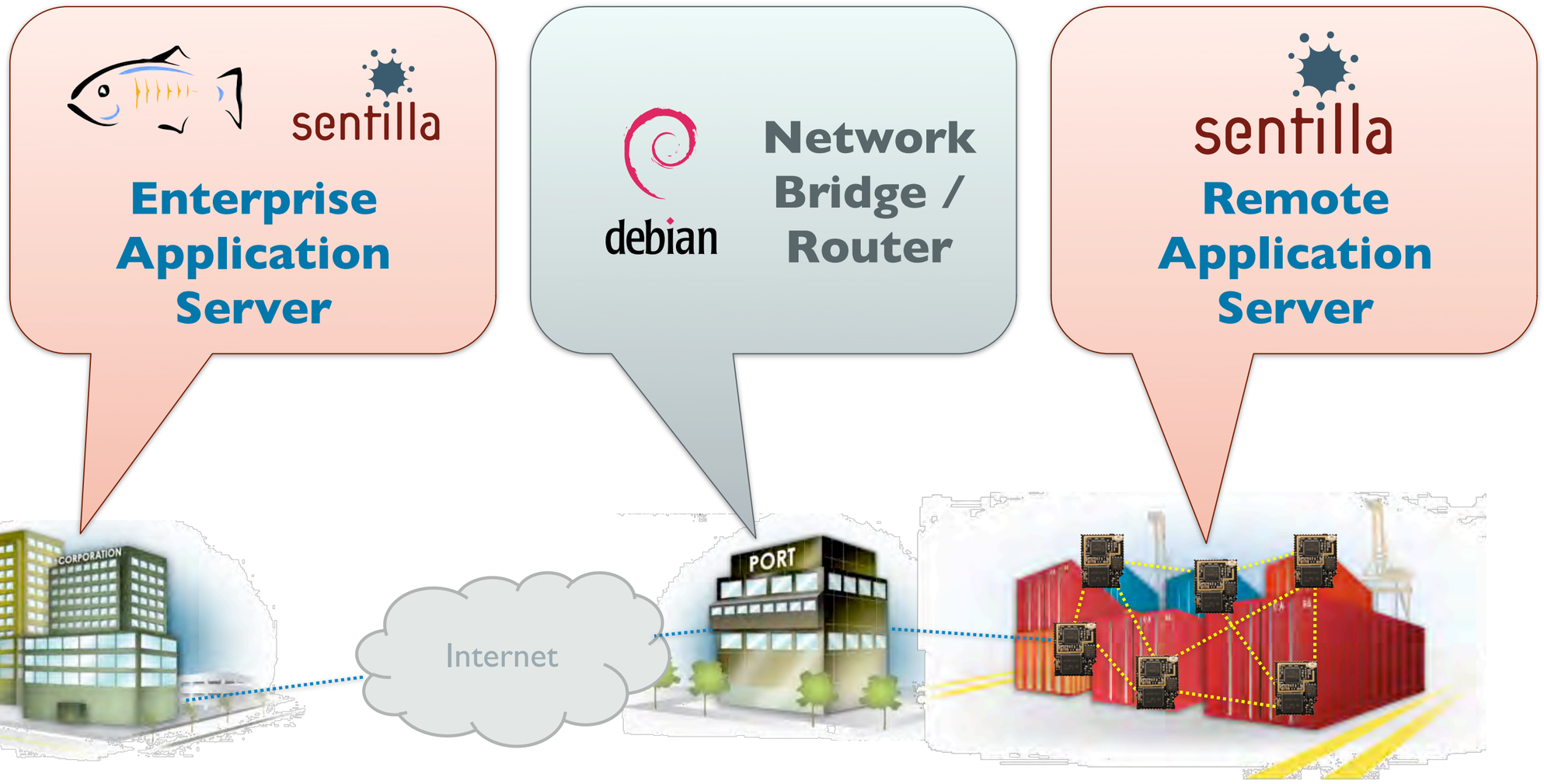
Mote
32-bit
512 kB
c. 2009

(Images to scale)

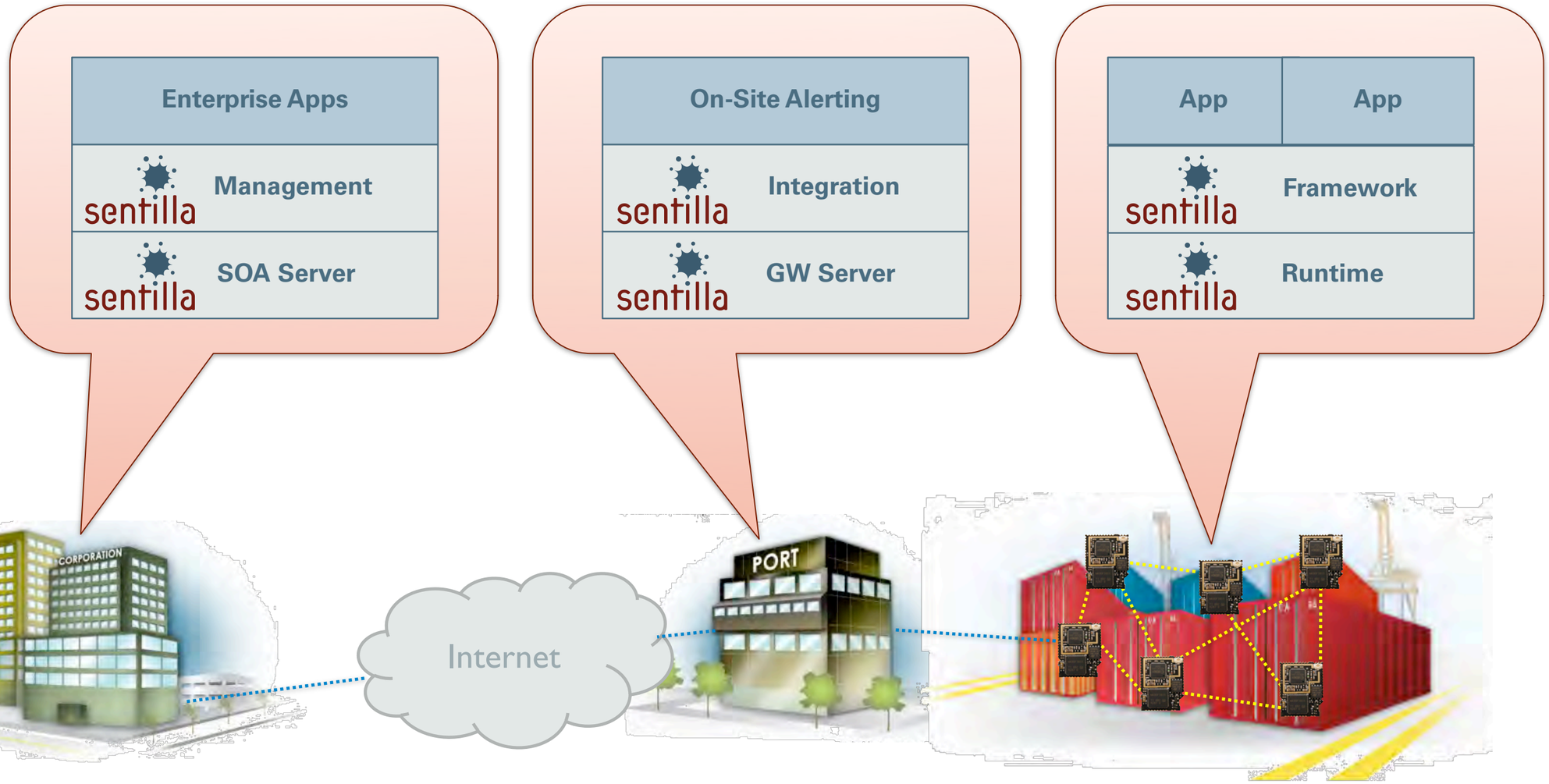
Physical Computing Cloud



Unified Framework for Applications



Sentilla Software Platform Overview

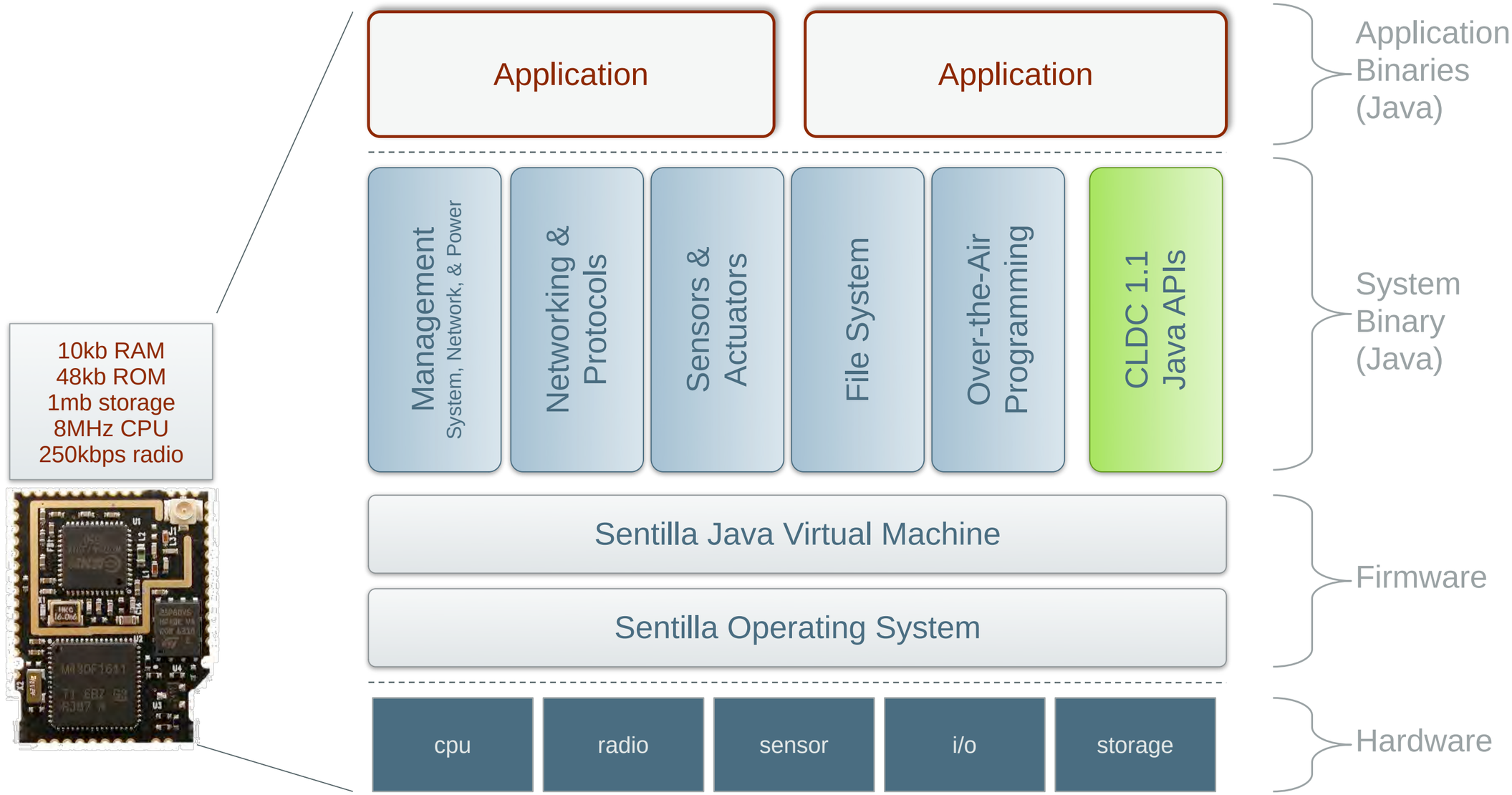


Corporate Enterprise

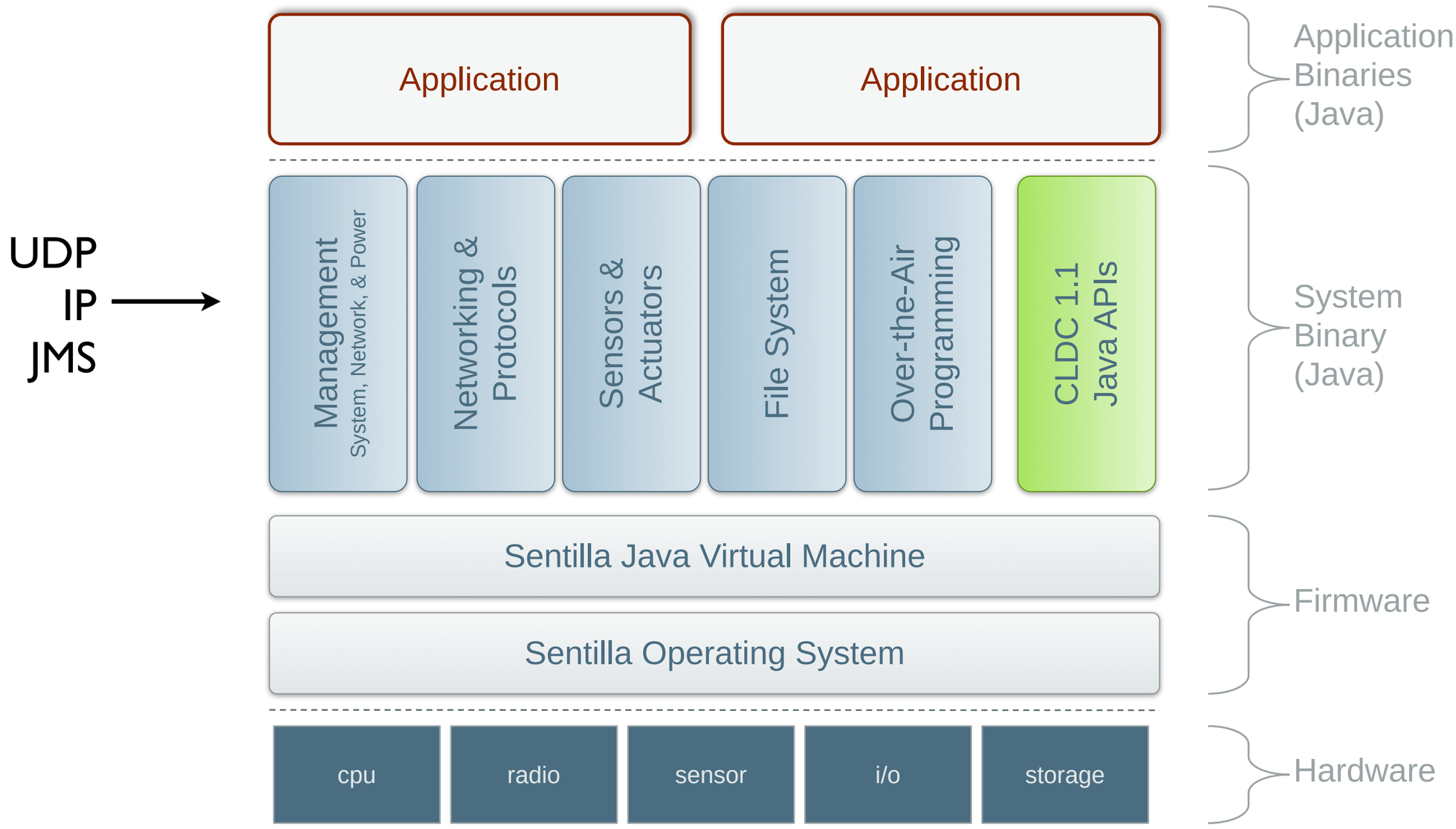
On-Site Center

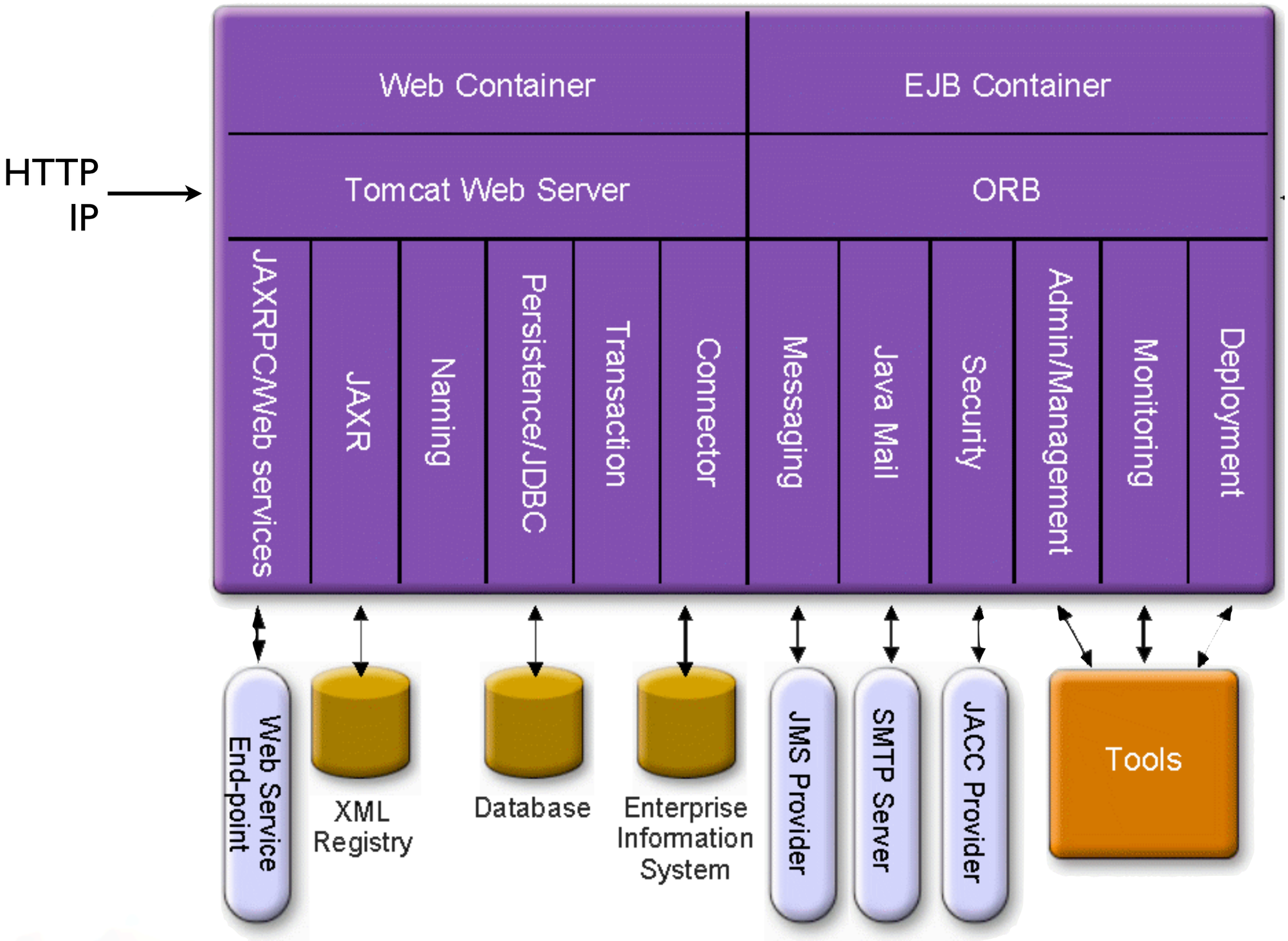
Field Deployment

Sentilla Mote Software Architecture



Sentilla Mote Software Architecture




HTTP
IP
Web Container
EJB Container
Tomcat Web Server
ORB
JAXRPC/Web services
JAXR
Naming
Persistence/JDBC
Transaction
Connector
Messaging
Java Mail
Security
Admin/Management
Monitoring
Deployment
Web Service End-point
XML Registry
Database
Enterprise Information System
JMS Provider
SMTP Server
JACC Provider
Tools

Code Comparison

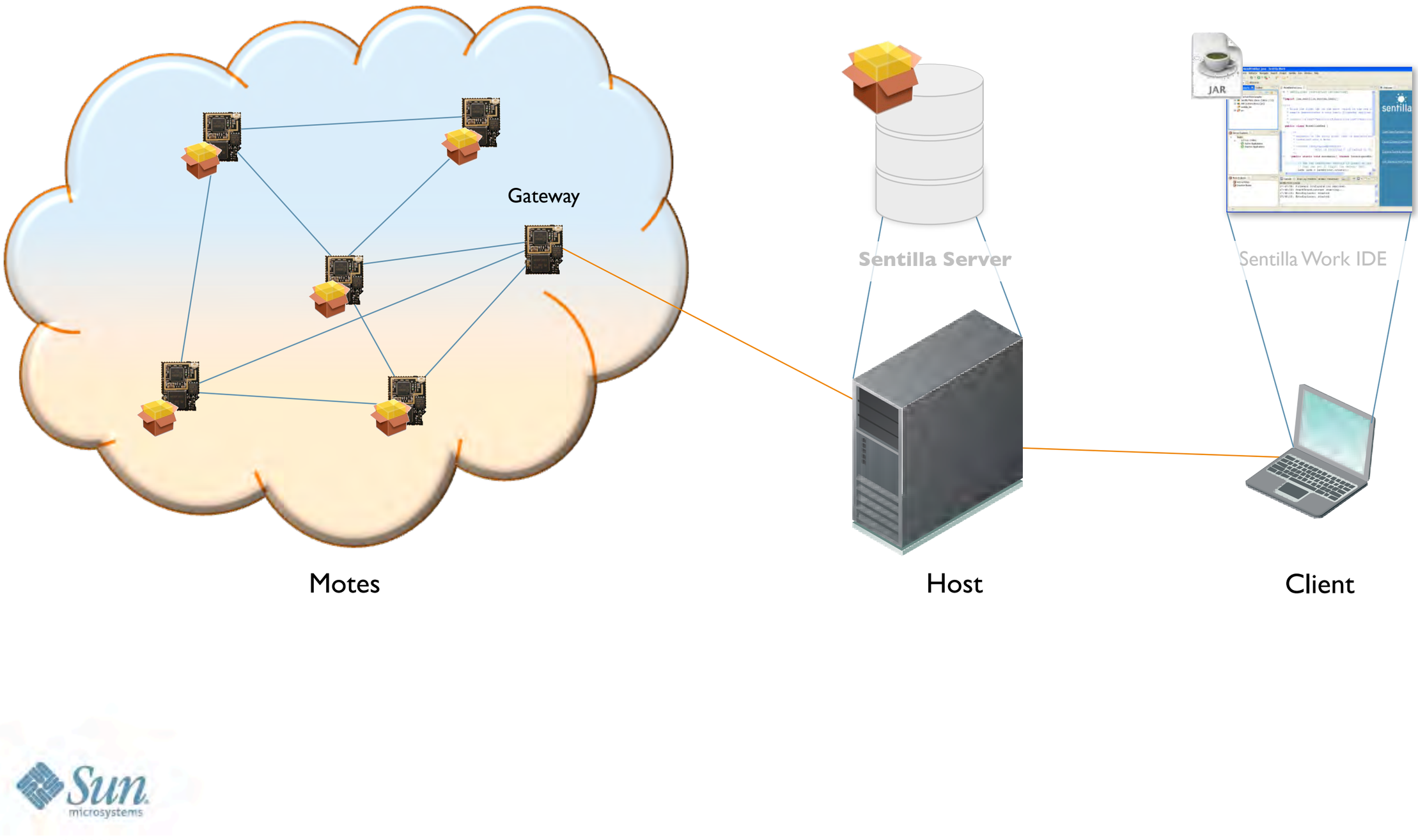
```
import com.sentilla.io.*;
import com.sentilla.system.*;
import javax.measure.*;

public class SensorApp {

  public static class SensorMessage implements Serializable {
    public long address;
    public Measurable<Temperature> value;
  }

  public static void motemain() {
    Sensor<Temperature> temp =
          SensorDriver.create("mcutemp", Temperature.class);
    Sender sender = SenderDriver.create("collect");
    SensorMessage msg = new SensorMessage();
    msg.address = PropertyDriver.open("mac64").readLong();
    while (true) {
      msg.value = temp.read();
      if (msg.value.doubleValue(SI.CELSIUS) > (double)50) {
        sender.send( msg );
      }
      try { Thread.sleep(1000); }
      catch (Exception e) { e.printStackTrace() }
    }
  }
```

```
#ifndef MULTIHOP_OSCILLOSCOPE_H
#define MULTIHOP_OSCILLOSCOPE_H
enum {
  /* Number of readings per message. If you increase this, you may have to
     increase the message_t size. */
  NREADINGS = 5,
  DEFAULT_INTERVAL = 1024,
  AM_OSCILLOSCOPE = 0x93
};
typedef nx_struct oscilloscope {
  nx_uint16_t version; /* Version of the interval. */
  nx_uint16_t interval; /* Samping period. */
  nx_uint16_t id; /* Mote id of sending mote. */
  nx_uint16_t count; /* The readings are samples count * NREADINGS onwards */
  nx_uint16_t readings[NREADINGS];
} oscilloscope_t;
#endif
configuration MultihopOscilloscopeAppC { }
implementation {
  components MainC, MultihopOscilloscopeC, LedsC, new TimerMilliC(),
    new DemoSensorC() as Sensor;
  MultihopOscilloscopeC.Boot -> MainC;
  MultihopOscilloscopeC.Timer -> TimerMilliC;
  MultihopOscilloscopeC.Read -> Sensor;
  MultihopOscilloscopeC.Leds -> LedsC;
  components CollectionC as Collector,  // Collection layer
    ActiveMessageC,                     // AM layer
    new CollectionSenderC(AM_OSCILLOSCOPE), // Sends multihop RF
    SerialActiveMessageC,               // Serial messaging
    new SerialAMSenderC(AM_OSCILLOSCOPE);  // Sends to the serial port
  MultihopOscilloscopeC.RadioControl -> ActiveMessageC;
  MultihopOscilloscopeC.SerialControl -> SerialActiveMessageC;
  MultihopOscilloscopeC.RoutingControl -> Collector;
  MultihopOscilloscopeC.Send -> CollectionSenderC;
  MultihopOscilloscopeC.SerialSend -> SerialAMSenderC.AMSend;
  MultihopOscilloscopeC.Snoop -> Collector.Snoop[AM_OSCILLOSCOPE];
  MultihopOscilloscopeC.Receive -> Collector.Receive[AM_OSCILLOSCOPE];
  MultihopOscilloscopeC.RootControl -> Collector;
  components new PoolC(message_t, 10) as UARTMessagePoolP,
    new QueueC(message_t*, 10) as UARTQueueP;
  MultihopOscilloscopeC.UARTMessagePool -> UARTMessagePoolP;
  MultihopOscilloscopeC.UARTQueue -> UARTQueueP;
#include "Timer.h"
#include "MultihopOscilloscope.h"
module MultihopOscilloscopeC {
  uses {
    // Interfaces for initialization:
    interface Boot;
    interface SplitControl as RadioControl;
    interface SplitControl as SerialControl;
    interface StdControl as RoutingControl;

    // Interfaces for communication, multihop and serial:
    interface Send;
    interface Receive as Snoop;
    interface Receive;
    interface AMSend as SerialSend;
    interface CollectionPacket;
    interface RootControl;
    interface Queue<message_t*> as UARTQueue;
    interface Pool<message_t> as UARTMessagePool;
    // Miscalleny:
    interface Timer<TMilli>;
    interface Read<uint16_t>;
    interface Leds;
  }
}
implementation {
  task void uartSendTask();
  static void startTimer();
  static void fatal_problem();
  static void report_problem();
  static void report_sent();
  static void report_received();
  uint8_t uartlen;
  message_t sendbuf;
  message_t uartbuf;
  bool sendbusy=FALSE, uartbusy=FALSE;
  oscilloscope_t local;
  uint8_t reading; /* 0 to NREADINGS */
  bool suppress_count_change;
  event void Boot.booted() {
    local.interval = DEFAULT_INTERVAL;
    local.id = TOS_NODE_ID;
    local.version = 0;
    // Beginning our initialization phases:
    if (call RadioControl.start() != SUCCESS)
      fatal_problem();
    if (call RoutingControl.start() != SUCCESS)
      fatal_problem();
```

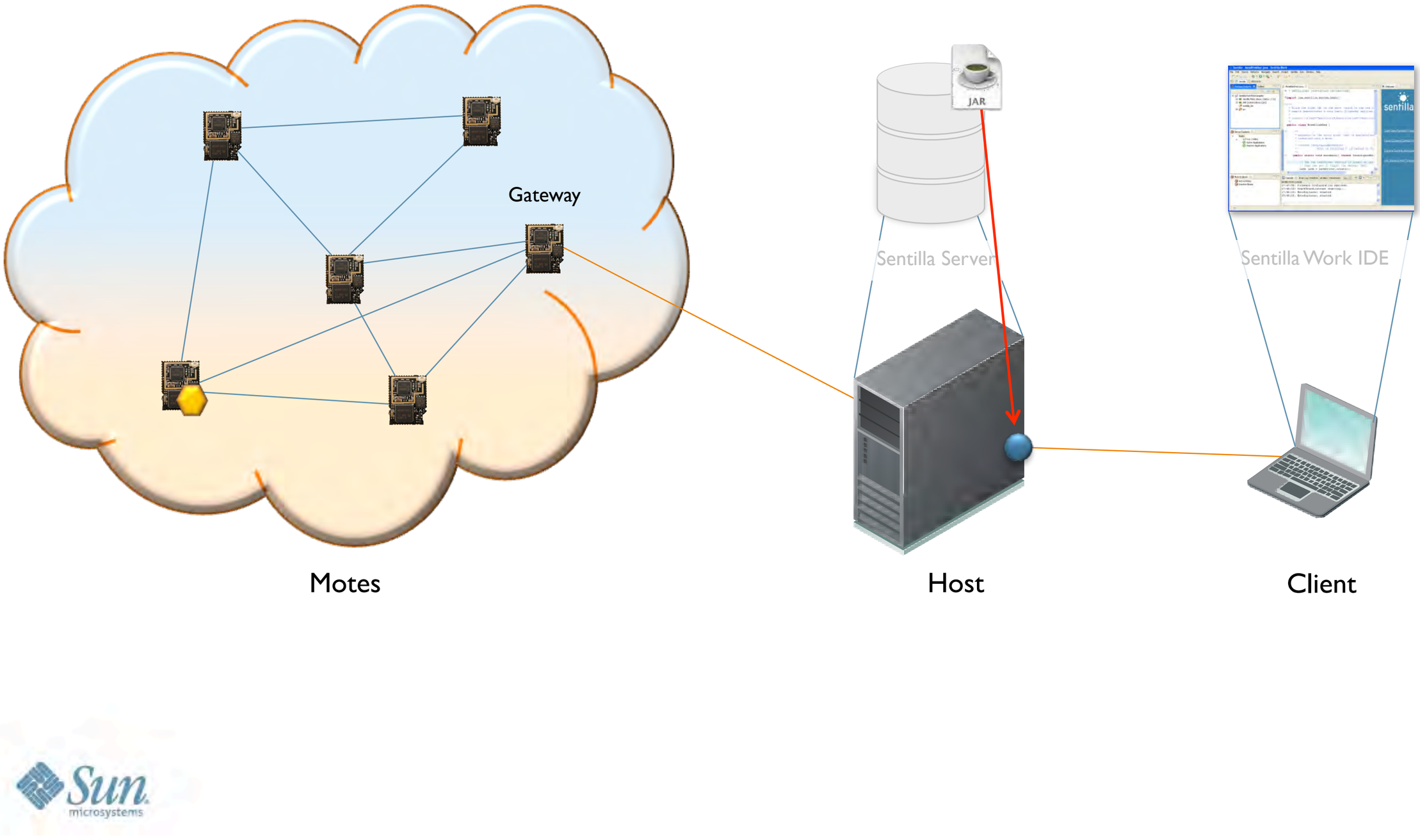
```
event void RadioControl.startDone(error_t error) {
  if (error != SUCCESS)
    fatal_problem();
  if (sizeof(local) > call Send.maxPayloadLength())
    fatal_problem();
  if (call SerialControl.start() != SUCCESS)
    fatal_problem();
}
}
 event void SerialControl.startDone(error_t error) {
  if (error != SUCCESS)
    fatal_problem();
  // This is how to set yourself as a root to the collection layer:
  if (local.id % 500 == 0)
    call RootControl.setRoot();
  startTimer();
}
static void startTimer() {
  if (call Timer.isRunning()) call Timer.stop();
  call Timer.startPeriodic(local.interval);
  reading = 0;
}
event void RadioControl.stopDone(error_t error) { }
event void SerialControl.stopDone(error_t error) { }
event message_t*
Receive.receive(message_t* msg, void *payload, uint8_t len) {
  oscilloscope_t* in = (oscilloscope_t*)payload;
  oscilloscope_t out;
  if (uartbusy == FALSE) {
    out = (oscilloscope_t*)call SerialSend.getPayload(&uartbuf,
    if (len != sizeof(oscilloscope_t)) {
return msg;
    }
    else {
memcpy(&out, payload, sizeof(oscilloscope_t));
    }
    uartlen = sizeof(oscilloscope_t);
    call SerialSend.send(
    uartbusy = TRUE;
  }
  else
    // The UART is busy; queue up messages and send them when the
    // UART becomes free.
    message_t *newmsg = call UARTMessagePool.get();
    if (newmsg == NULL) {
      // This should never happen with the current size of queue space.
      report_problem();
      return msg;
    }
    //Prepare message to be sent over the uart
    out = (oscilloscope_t*)call SerialSend.getPayload(newmsg);
    memcpy(out, payload, sizeof(oscilloscope_t));
    if (call UARTQueue.enqueue(newmsg) != SUCCESS) {
      call UARTMessagePool.put(newmsg);
      fatal_problem();
      return msg;
    }
  }
  return msg;
}
task void uartSendTask() {
  if (call SerialSend.send(0xffff, &uartbuf, uartlen) != SUCCESS) {
    report_problem();
  } else {
    uartbusy = TRUE;
  }
}
event void SerialSend.sendDone(message_t *msg, error_t error) {
  uartbusy = FALSE;
  if (call UARTQueue.empty() == FALSE) {
    message_t *queuemsg = call UARTQueue.dequeue();
    if (queuemsg == NULL) {
      fatal_problem();
      return;
    }
    memcpy(&uartbuf, queuemsg, sizeof(message_t));
    if (call UARTMessagePool.put(queuemsg) != SUCCESS) {
      fatal_problem();
      return;
    }
    post uartSendTask();
  }
}
event message_t*
Snoop.receive(message_t* msg, void* payload, uint8_t len) {
  oscilloscope_t *omsg = payload;
  report_received();
  // If we receive a newer version, update our interval.
  if (omsg->version > local.version) {
    local.version = omsg->version;
    local.interval = omsg->interval;
    startTimer();
  }
```

```
// If we hear from a future count, jump ahead but suppress our own
  // change.
  if (omsg->count > local.count) {
    local.count = omsg->count;
    suppress_count_change = TRUE;
  }
  return msg;
}
event void Timer.fired() {
  if (reading == NREADINGS) {
    if (!sendbusy) {
      oscilloscope_t *o = (oscilloscope_t *)call Send.getPayload(&sendbuf);
      memcpy(o, &local, sizeof(local));
      if (call Send.send(&sendbuf, sizeof(local)) == SUCCESS)
        sendbusy = TRUE;
      else
        report_problem();
    }

    reading = 0;
    /* Part of cheap "time" sync: increment our count if we didn't
       jump ahead. */
    if (!suppress_count_change)
      local.count++;
    suppress_count_change = FALSE;
  }
  if (call Read.read() != SUCCESS)
    report_problem();
}
event void Send.sendDone(message_t* msg, error_t error) {
  if (error == SUCCESS)
    report_sent();
  else
    report_problem();
  sendbusy = FALSE;
}
event void Read.readDone(error_t result, uint16_t data) {
  if (result != SUCCESS) {
    data = 0xffff;
    report_problem();
  }
  local.readings[reading++] = data;
}
// Use LEDs to report various status issues.
static void fatal_problem() {
  call Leds.led0On();
  call Leds.led1On();
  call Leds.led2On();
  call Timer.stop();
}
static void report_problem() { call Leds.led0Toggle(); }
static void report_sent() { call Leds.led1Toggle(); }
static void report_received() { call Leds.led2Toggle(); }
```

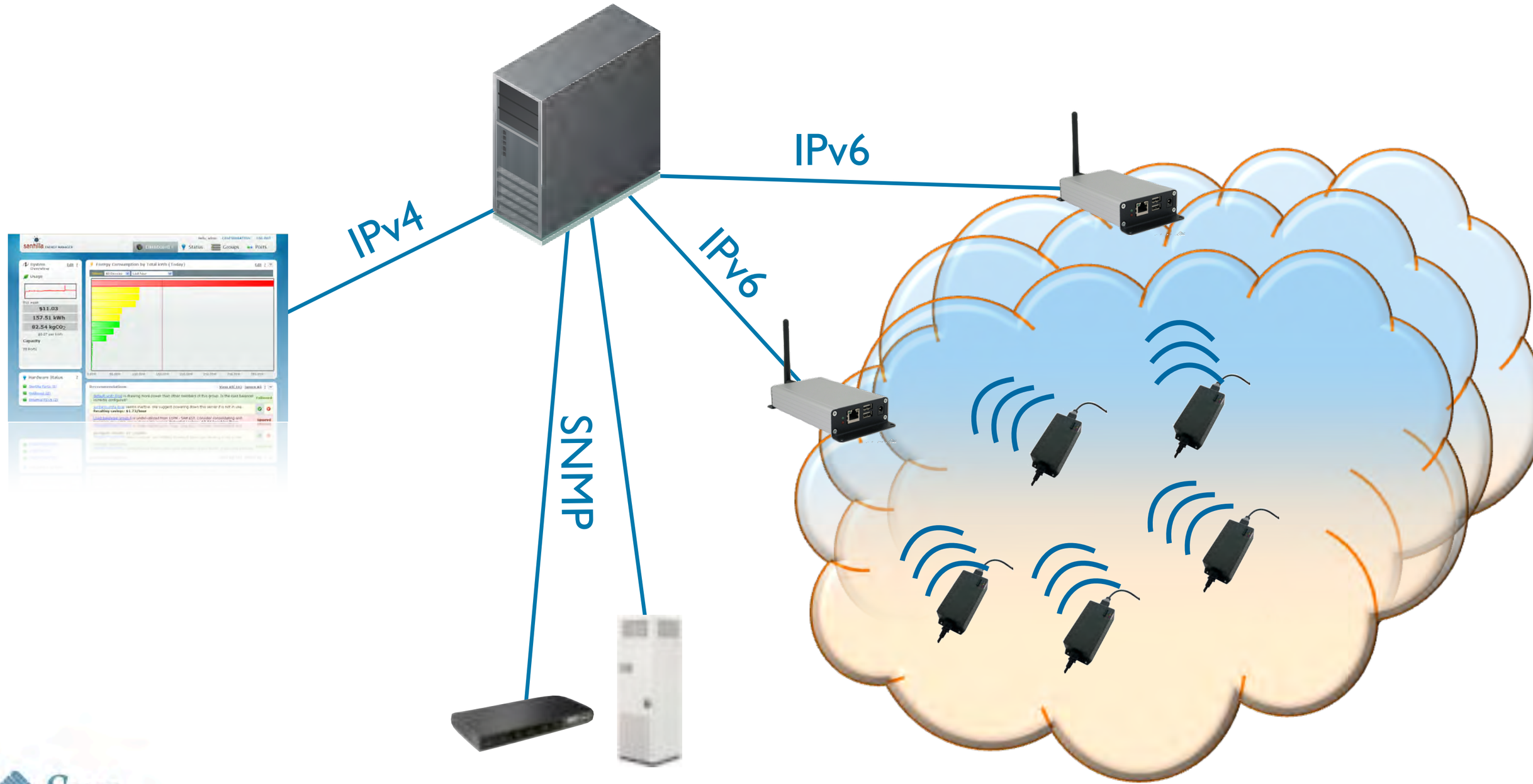
Sentilla Application Distribution



Sentilla Application Distribution



Architecture for Energy Management



Low Power IP: 6lowpan

```
sender = (DatagramConnection) Connector.open("udp://[ff02::1]:" + ENERGY_PORT);
receiver = (DatagramConnection) Connector.open("udp://:" + ENERGY_COMMAND_PORT);
data = (Datagram) sender.newDatagram(100);
rdata = (Datagram) receiver.newDatagram(100);

data.reset();
data.setLength(100);
emsg = new EnergyMessage();
```

```
emsg.clear();
emsg.moteid = moteid;
emsg.VRMS = meter.VRMS.read();
emsg.IRMS = meter.IRMS.read();
emsg.LA_ENERGY = meter.LA_ENERGY.read();
emsg.LVA_ENERGY = meter.LVA_ENERGY.read();
emsg.seqno++;
sender.send(emsg.encode(data));

rdata.reset();
receiver.receive(rdata);
this.moteid = rdata.readLong();
this.seqno = rdata.readLong();
this.command = rdata.readByte();
this.param = rdata.readInt();
```

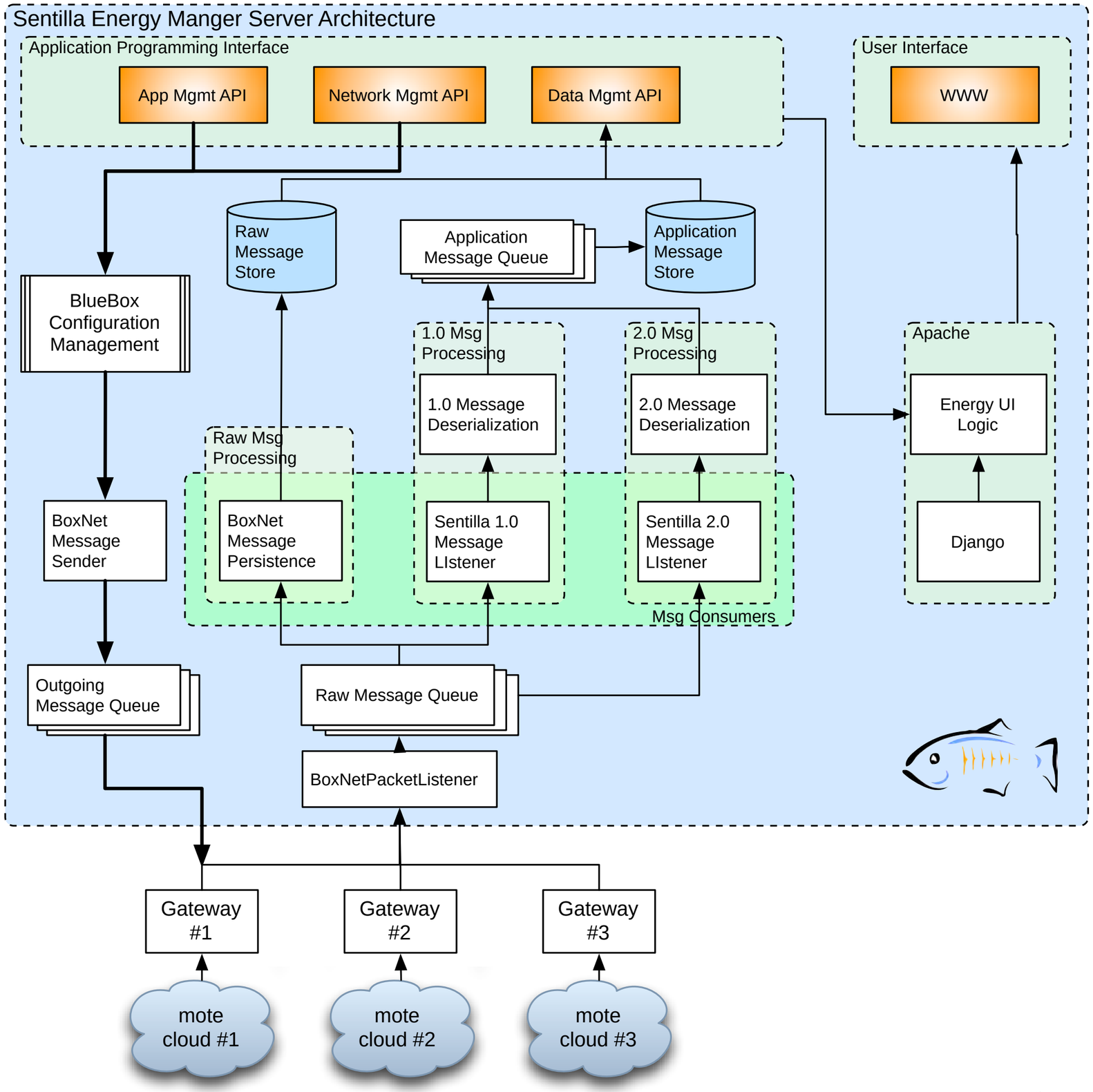
Intelligent Energy Measurement

```
switch (dispatch) {
    case Message.CAL_MSG:
     setCalibration();
       break;

    case Message.COMMAND_MSG:
     cmdmsg.decode(rdata);
       doSwitch(cmgmsg.command);
       break;
    default:
          break;
    }
```

```
switch(cmdmsg.command)
{
case CommandMessage.CMD_SET_MODE:
 Mode = (  byte)cmdmsg.param;
 demo_stoptime = System.currentTimeMillis() * 1000 * 60;
   break;
case CommandMessage.CMD_GET_CALIBRATION:
 sender.block();
 calmsg.send(sender,data);
   break;
case CommandMessage.CMD_RESET:
 meter.reset((  byte)0);
   break;
case CommandMessage.CMD_SET_REPORT_RATE:
 PollingPeriod = cmdmsg.param;
 PropertyDriver.open(  "P").writeInt(PollingPeriod);
   break;
case CommandMessage.CMD_GET_REPORT_RATE:
 cmdmsg.param = PollingPeriod;
 cmdmsg.encode(data);
  sender.block();
 cmdmsg.send(sender,data);
   break;
case CommandMessage.CMD_REINIT_ADE:
 initADE();
   break;
}
```

Server Architecture

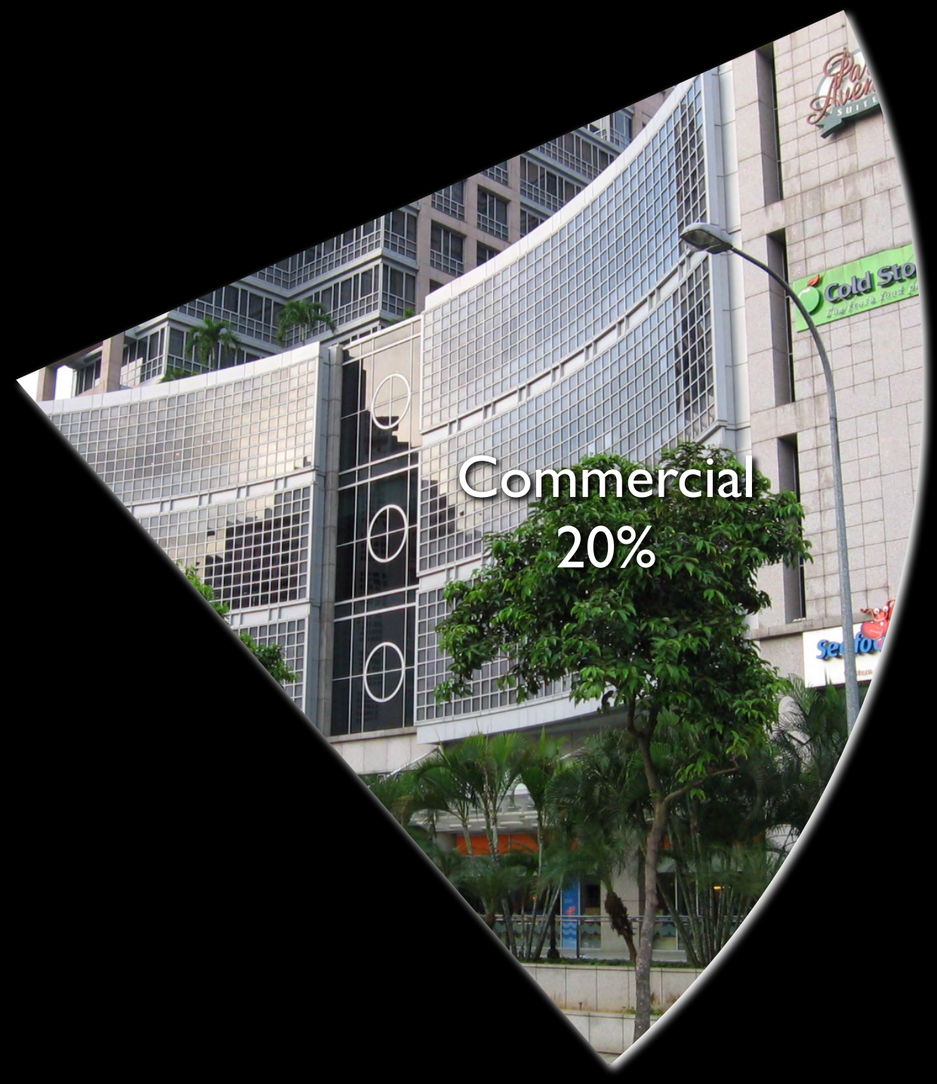
- Glassfish
- Message Queue
- Postgres
- Django/Python
- Apache



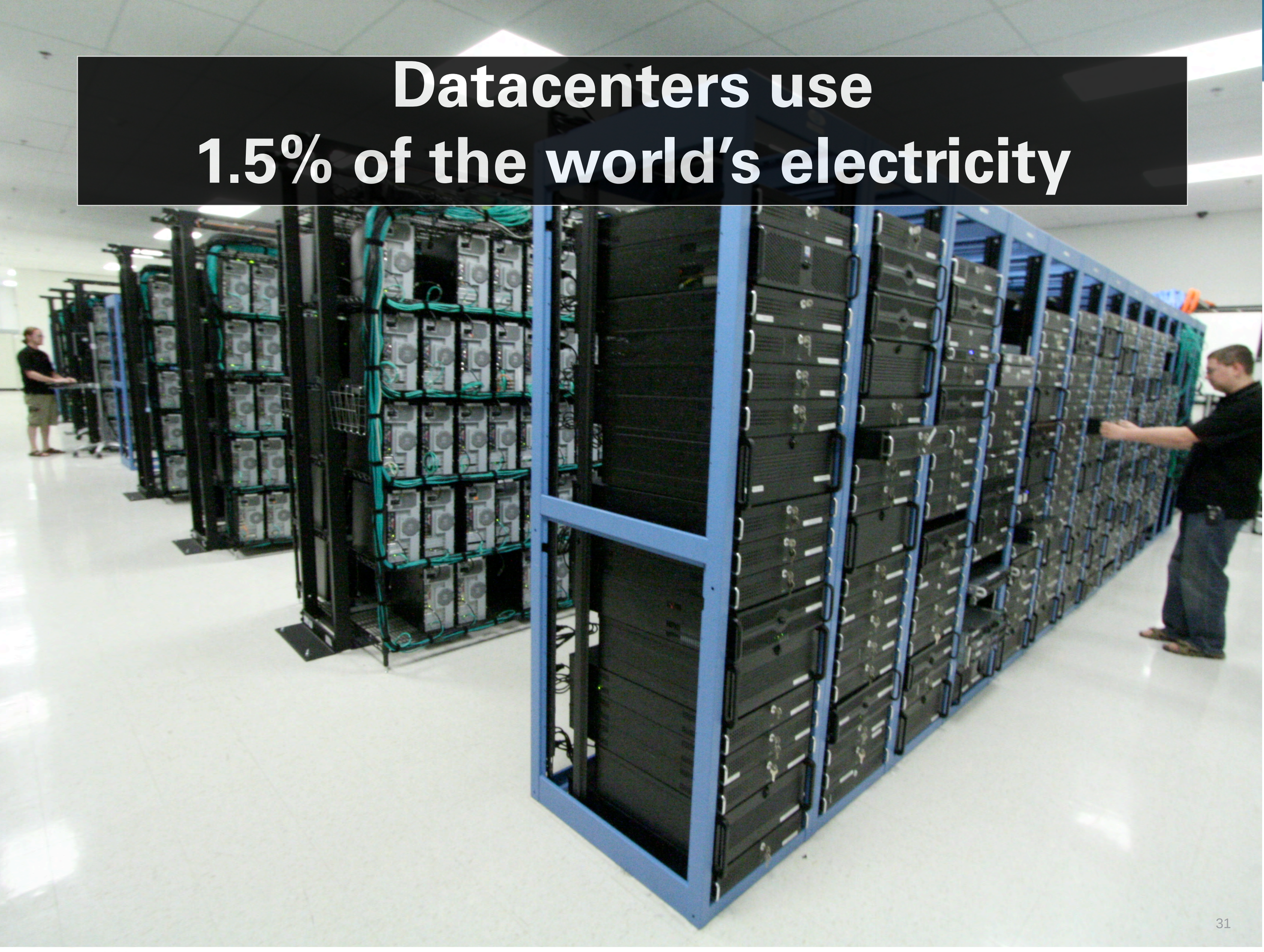
Building an Energy Analysis EJB

```
package com.sentilla.bluebox.reclets.alerts;

import com.sentilla.bluebox.alerts.CurrentAlert;
import com.sentilla.bluebox.energy.entity.EnergyState;
import com.sentilla.bluebox.recommendation.interfaces.Reclet;
import java.io.Serializable;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@MessageDriven(mappedName = "EnergyManagerInputTopic", activationConfig = {
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
    @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "Durable"),
    @ActivationConfigProperty(propertyName = "clientId", propertyValue = "AlertBean"),
    @ActivationConfigProperty(propertyName = "subscriptionName", propertyValue = "AlertBean")
})
```

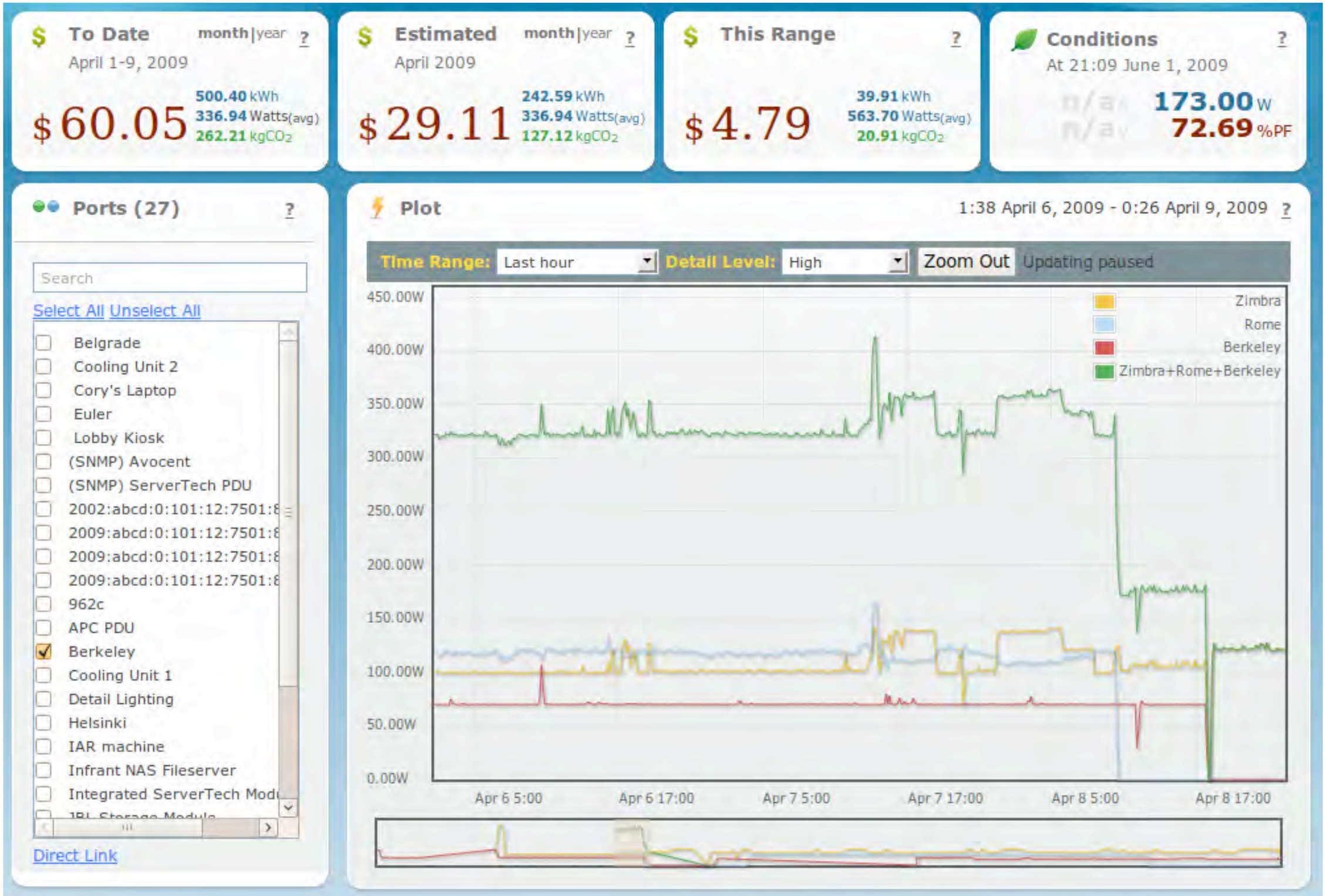
```
public class SampleCurrentAlert implements MessageListener, Reclet {
    private static int THRESHOLD = 140;
    @PersistenceContext(unitName = "SampleCurrentAlertPU")
    private EntityManager em;

    public void onMessage(Message message) {
        if (message instanceof ObjectMessage) {
            Serializable objmsg = null;
            try { objmsg = ((ObjectMessage)message).getObject();
            } catch (JMSException ex) {
                ex.printStackTrace();
            }

            if (objmsg instanceof EnergyState) {
                EnergyState emsg = (EnergyState) objmsg;
                if (emsg.getIRMS() > THRESHOLD) {
                    CurrentAlert alert = new CurrentAlert();
                    alert.setNodeId(emsg.getMoteId());
                    alert.setMessage("Max current threshold exceeded");
                    em.persist(alert);
                }
            }
        }
    }
}
```


Commercial
20%

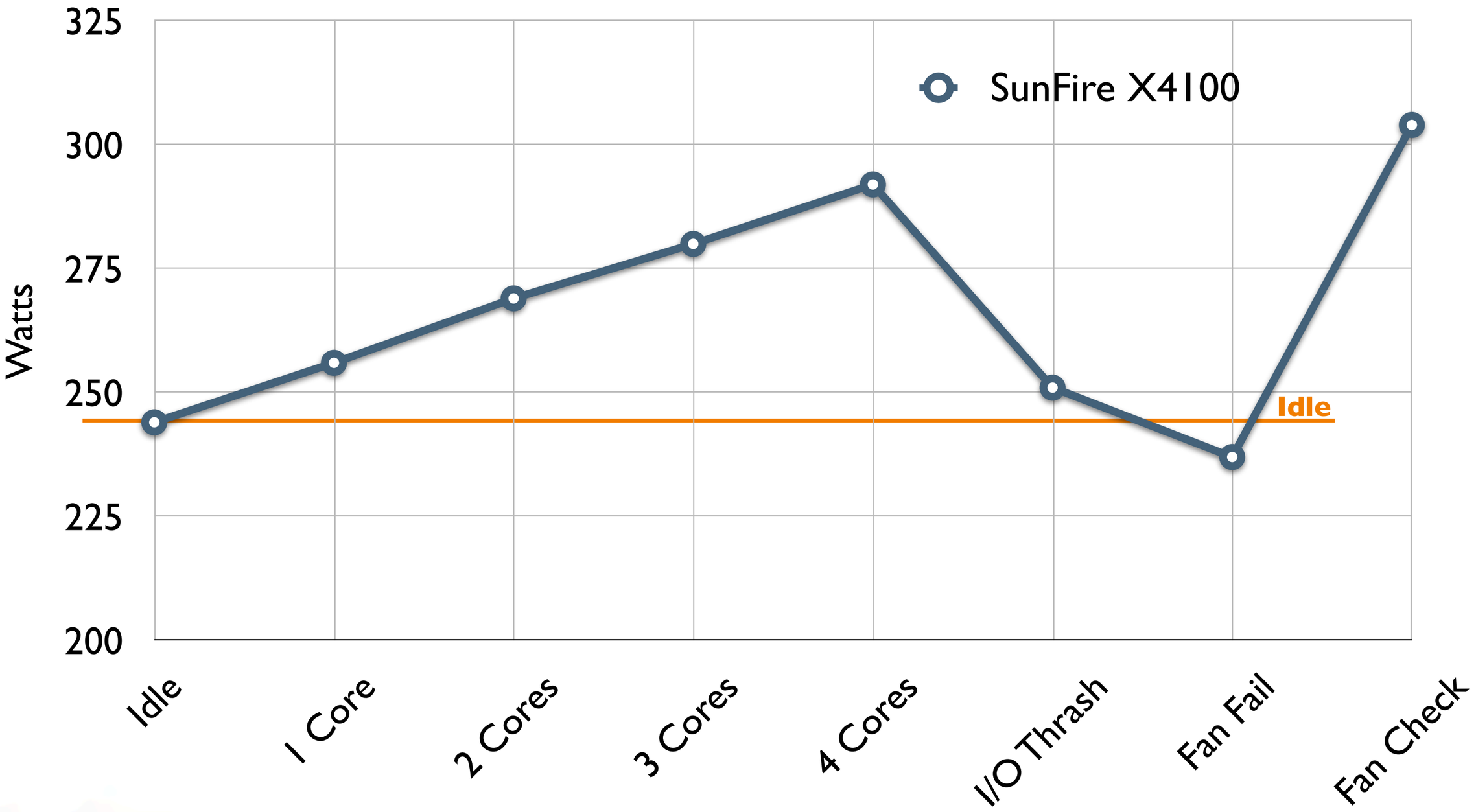
Datacenters use 1.5% of the world's electricity



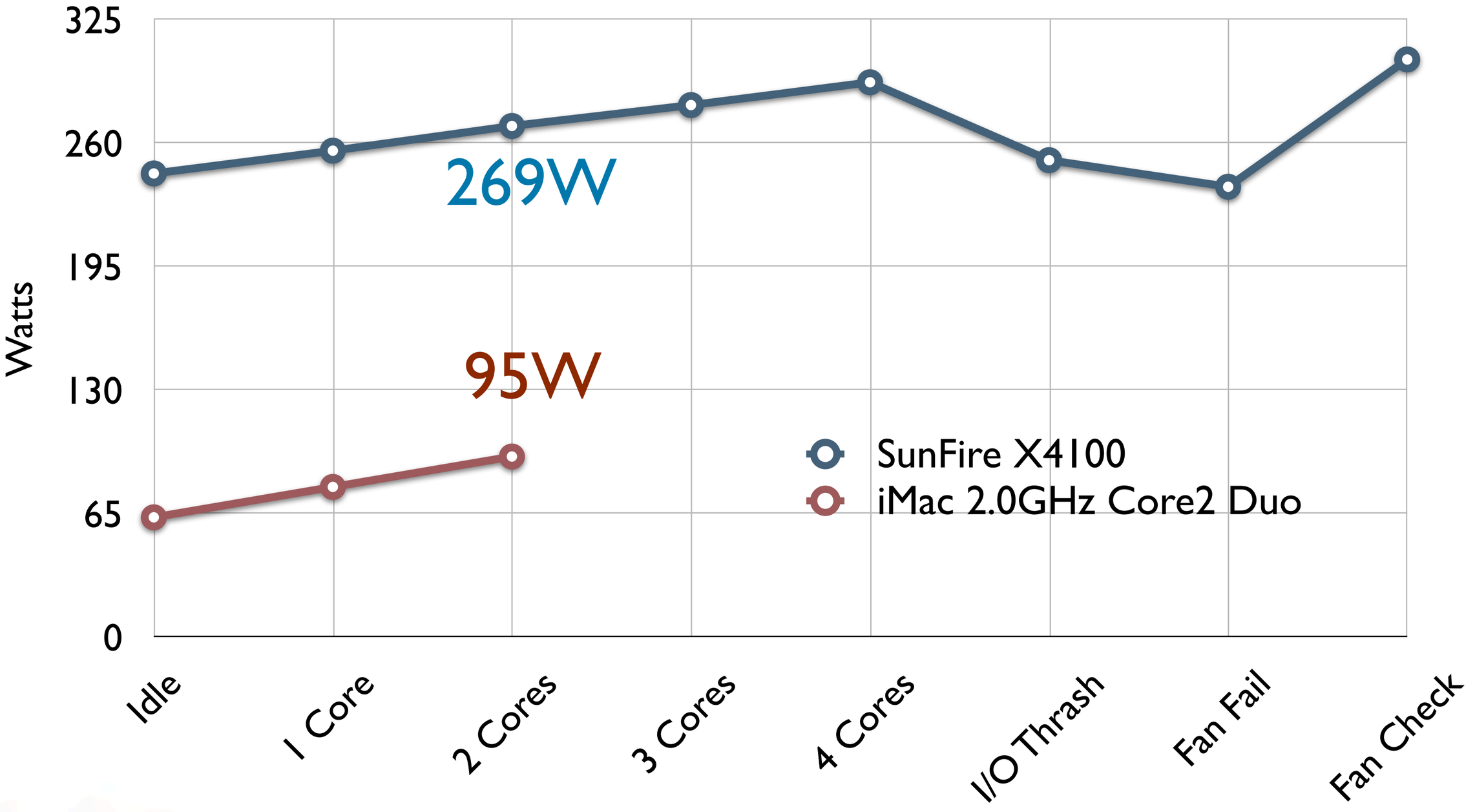
Demo


$ To Date month|year ?
April 1-9, 2009
$60.05
500.40 kWh
336.94 Watts(avg)
262.21 kgCO2
$ Estimated month|year ?
April 2009
$29.11
242.59 kWh
336.94 Watts(avg)
127.12 kgCO2
$ This Range ?
$4.79
39.91 kWh
563.70 Watts(avg)
20.91 kgCO2
Conditions ?
At 21:09 June 1, 2009
n/a
173.00 W
n/a
72.69 %PF
Ports (27) ?
Search
Select All Unselect All
Belgrade
Cooling Unit 2
Cory's Laptop
Euler
Lobby Kiosk
(SNMP) Avocent
(SNMP) ServerTech PDU
2002:abcd:0:101:12:7501:8
2009:abcd:0:101:12:7501:8
2009:abcd:0:101:12:7501:8
2009:abcd:0:101:12:7501:8
962c
APC PDU
Berkeley
Cooling Unit 1
Detail Lighting
Helsinki
IAR machine
Infrant NAS Fileserver
Integrated ServerTech Modu
Direct Link
Plot
1:38 April 6, 2009 - 0:26 April 9, 2009 ?
Time Range: Last hour
Detail Level: High
Zoom Out
Updating paused
450.00W
400.00W
350.00W
300.00W
250.00W
200.00W
150.00W
100.00W
50.00W
0.00W
Zimbra
Rome
Berkeley
Zimbra+Rome+Berkeley
Apr 6 5:00
Apr 6 17:00
Apr 7 5:00
Apr 7 17:00
Apr 8 5:00
Apr 8 17:00

Real-time Energy Profiling



Energy Comparison



Floating Point Comparison

	Idle Watts	Active Watts	SPECfp2006	Watt/fp
SunFire	244	269	14.7	18.3
iMac	63	95	8.8	10.8

SPECfp2006 is relative to a Sun UltraSparc II system at 296MHz

Pacific Gas and Electric Company

WE DELIVER ENERGY.™

ELECTRIC ACCOUNT DETAIL

Service ID #: 2013989307
Rate Schedule: E1 TB Residential Service
Billing Days: 32 days

Serial	Rotating Outage Blk	Meter #	Prior Meter Read	Current Meter Read	Difference	Meter Constant	Usage
G	50	32L596	47,315	47,602	287	1	287 Kwh

Charges

Television	120.3 Kwh	$13.83
Refrigerator	50.2 Kwh	$5.77
Dishwasher	10.1 Kwh	$1.16
Computer	12.2 Kwh	$1.40
Clothes Dryer	35.2 Kwh	$4.05
Not Controlled	59.0 Kwh	$6.79

The Internet is Lonely
(and so is the thermostat)

570 million things on the Internet

10+ trillion things in the world

0.005% of things on the Internet

JavaOne℠ Thank You

Cory Sharp
cory@sentilla.com

Sentilla Corporation
201 Marshall Street
Redwood City, CA 94063
650.241.0220