Understand Linux Components

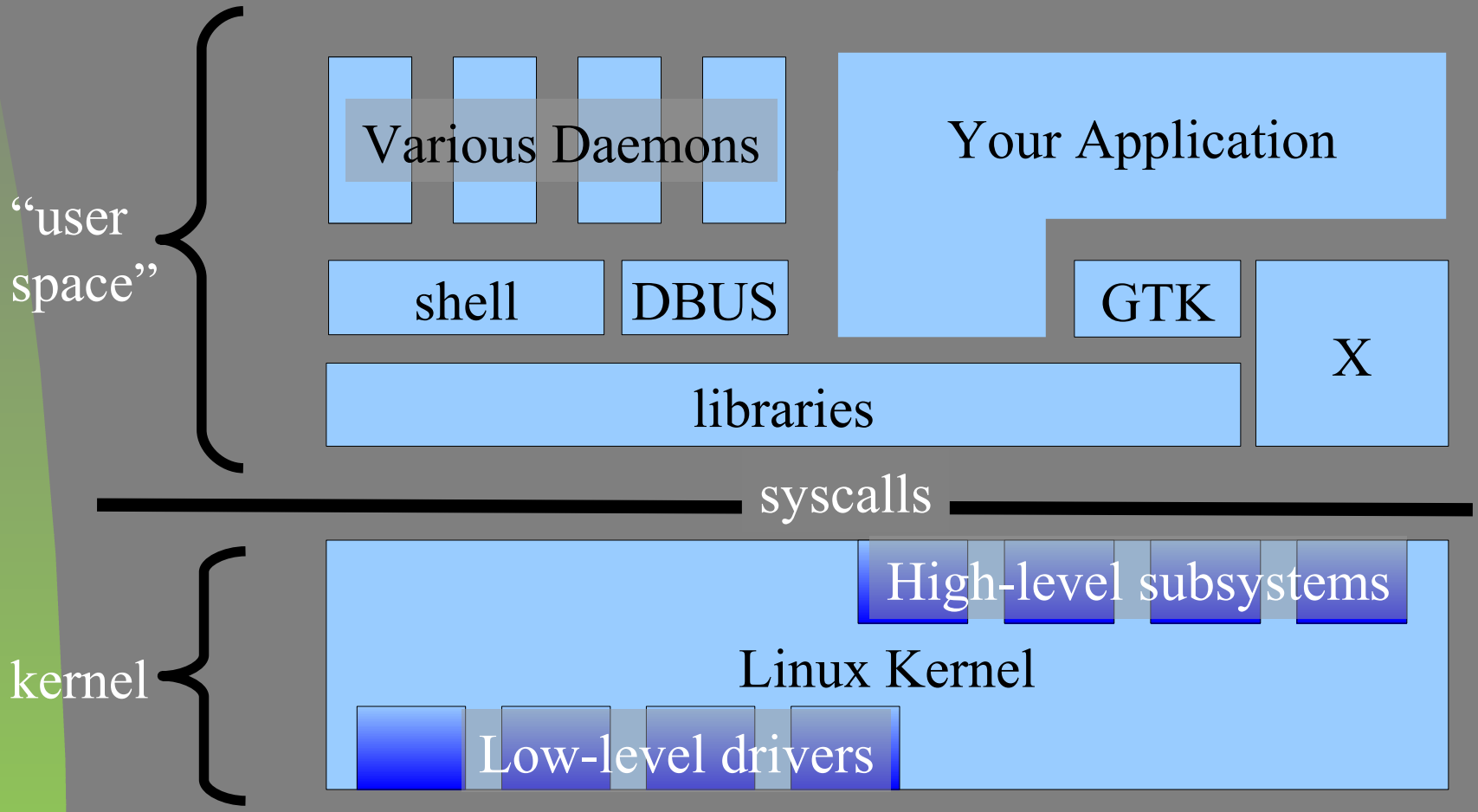
Kill Bugs, and Fix Wasteful Code

Klaas van Gend

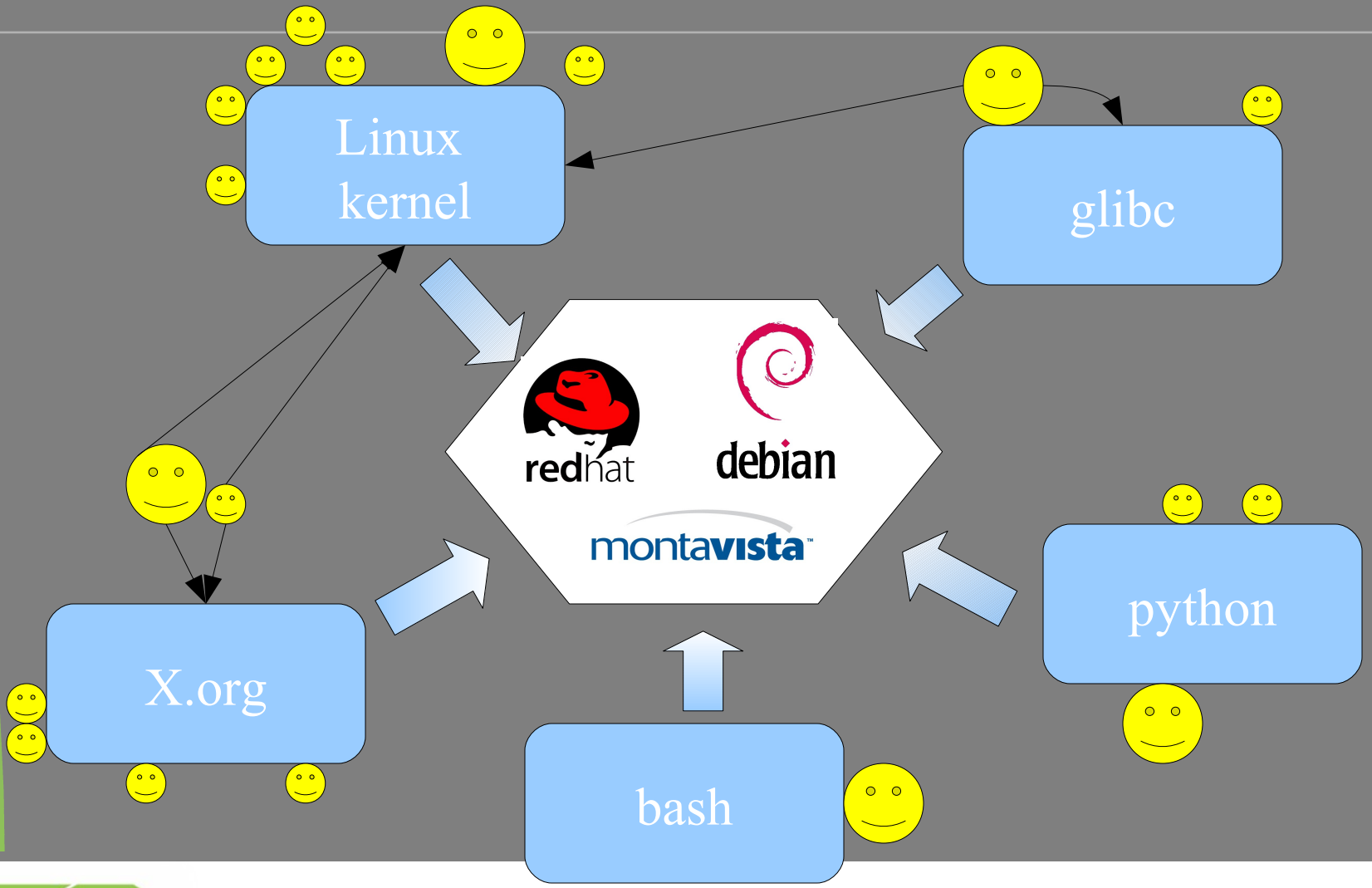
Agenda

- “Linux” is not a single building block
  - Buy, Build, Borrow
  - Deploy vs. Debug
- Demo: Use GDB
  - Lab: Kill a bug
- Power Management Blocks
  - Demo: Powertop
  - (Lab: Fix Wasteful Code)
- When have you won?

Linux Building Blocks

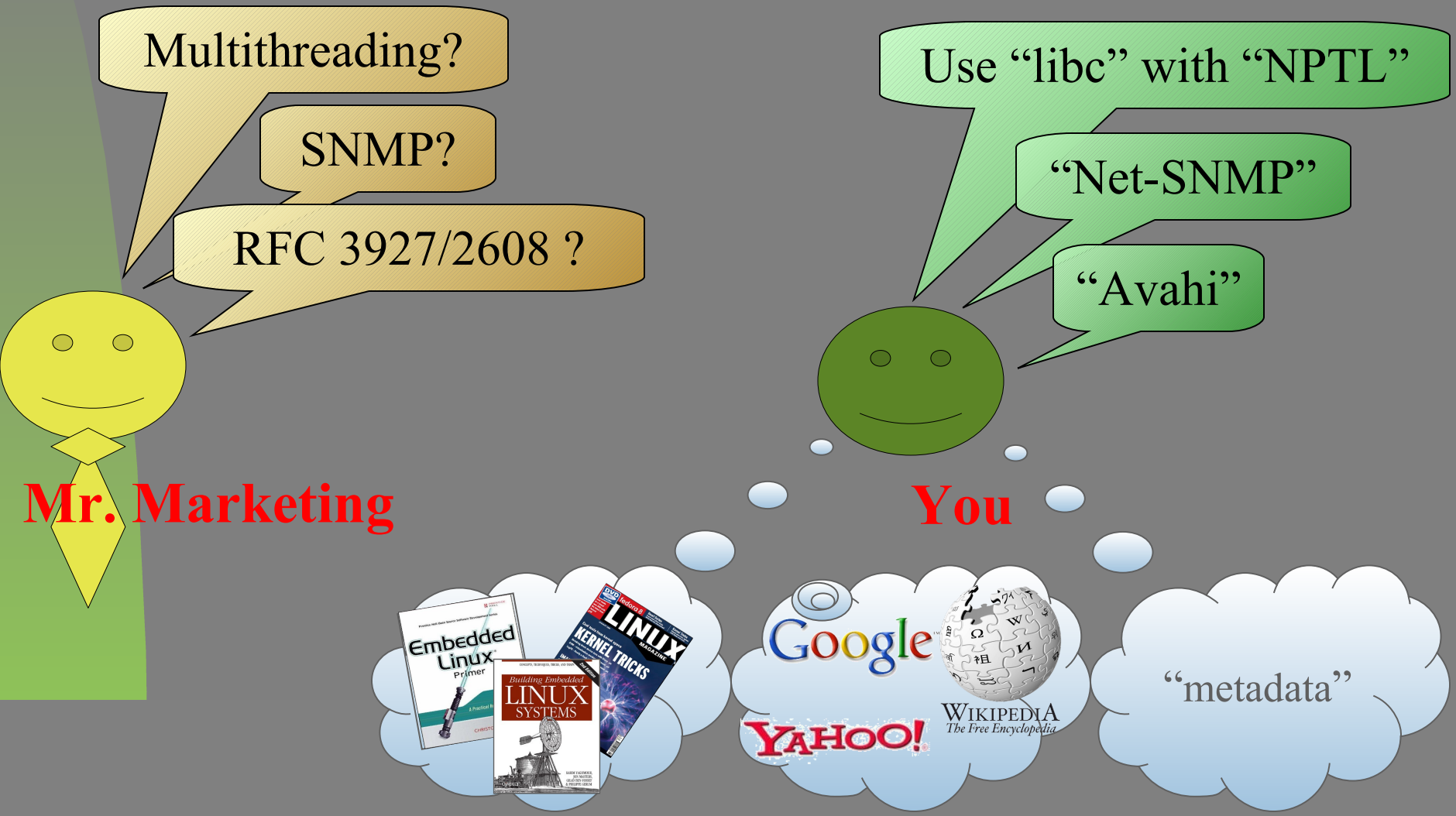


Everything is separate



Packages: Feature selections

System Designer must map requirements to packages



Build, Buy or Borrow?

What software packages to select?
From which source?

- System Software is usually more than one application
- Everything has been implemented before
- Division between kernel space and user space
  - Driver design – debugging, performance, licensing
- Software Licensing

Layers & Libraries

- Imagine you need to download a web page...

```
#include <curl/curl.h>
int main(void)
{
  CURLcode res;
  CURL * curl = curl_easy_init();

  curl_easy_setopt(curl,
     CURLOPT_URL,
     "www.mvista.com");
  res = curl_easy_perform(curl);
  curl_easy_cleanup(curl);
  return 0;
}
```

```
#include <stdlib.h>
int main(void)
{
  system("/usr/bin/wget -O - "
            "www.mvista.com");
  return 0;
}
```

- Is this all?
- Is this enough?
- Is this correct?

How to glue it together?

- Once you have decided to use package X and Y
  - How do they communicate?
- System Design
  - System startup behavior?
  - SysV or BSD-like init?
  - Daemonize or not?
  - Remote update?
  - Do I want a shell?

Debugging


cond->start >= first->end)
eturn 0;
The ranges of first and second do inters
int firstStart = first->start, firstEnd =
econd->end;
int newStart = firstStart < secondStart ?
int newEnd = firstEnd > secondEnd ? first
mpute the split-position for the res
splitPosition = splitPositionForR
he ranges overlap, we can
t contains it,

printf() / printk()

GDB - 1

- Understand code flow
- Inspect/modify variables
- Set Breakpoints
- Set Watchpoints
- View Backtraces
- Crash analysis

Demo: GDB / GDB TUI

- *-tui or Ctrl-X A to start TUI*
- 'run' / 'start' / 'step' / 'next' / 'until'
- `break <file>:<nr> if <condition>`
- `info breakpoints`
- `print x`
- `print *x`
- `display *x`
- `disable 3`

See “cheat sheet”

Disadvantages of GDB

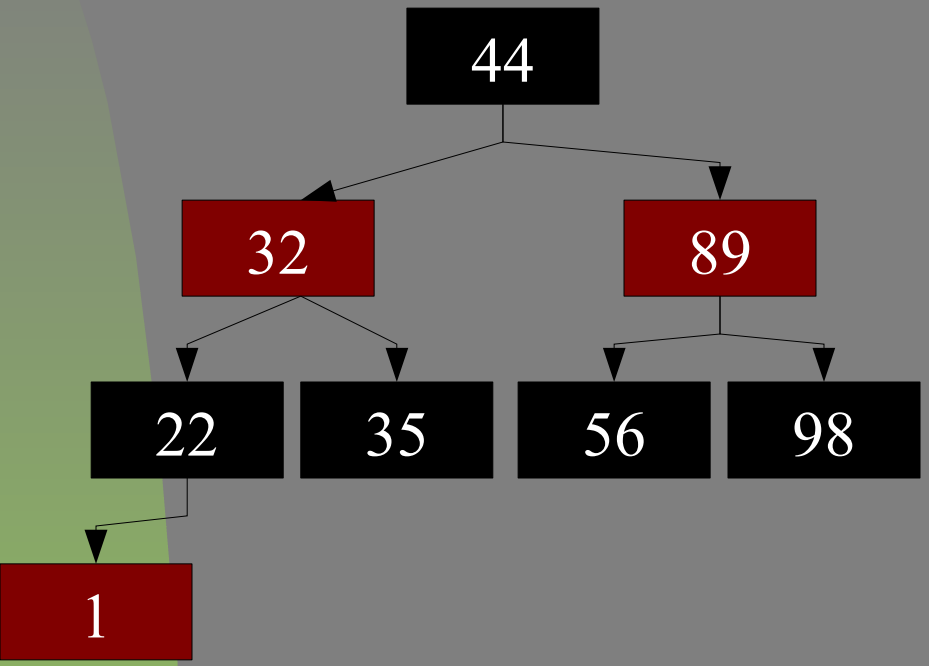
- *stdout* mixes with command view
- Breakpoints halt the whole process
- Breakpoints halt the process only
- Watchpoints are expensive
- GDB is big
- GDB requires symbol info

} gdbserver

- Not graphical

} DDD or Eclipse

Demo: Red/Black tree



R/B tree

- always balanced
- $\log_2(n)+1$ levels deep
- Fast insert O(log N)
- Fast search O(log N)

- But... is the implementation bad?

Let's check that (demo)

Lab assignment: preparation

- Set the date of your board to today:

```
date  -s 033003152009
```

Lab assignment: Red/Black

- Single step through the code
- Look carefully at variables
- Find where the balancing fails
- Fix it
- What is the maximum level now?
- Bonus: draw the tree

Steps:

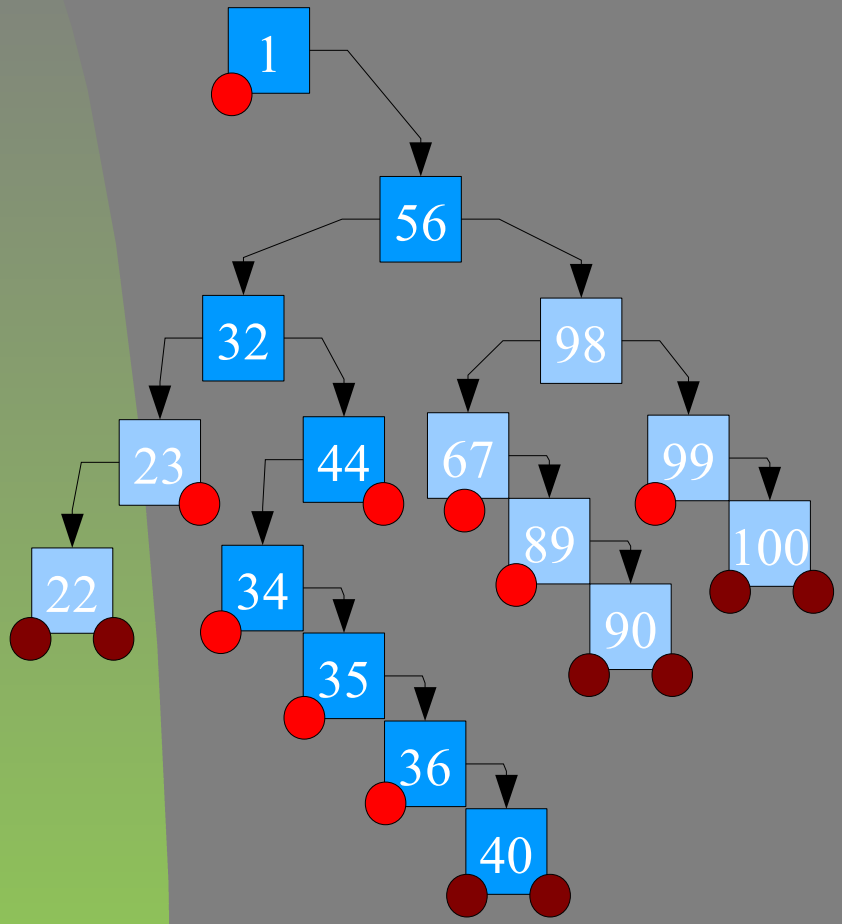
```
cd /home/reblack
gdb ./redblack
  set args myfile
  start
  step step step next
```

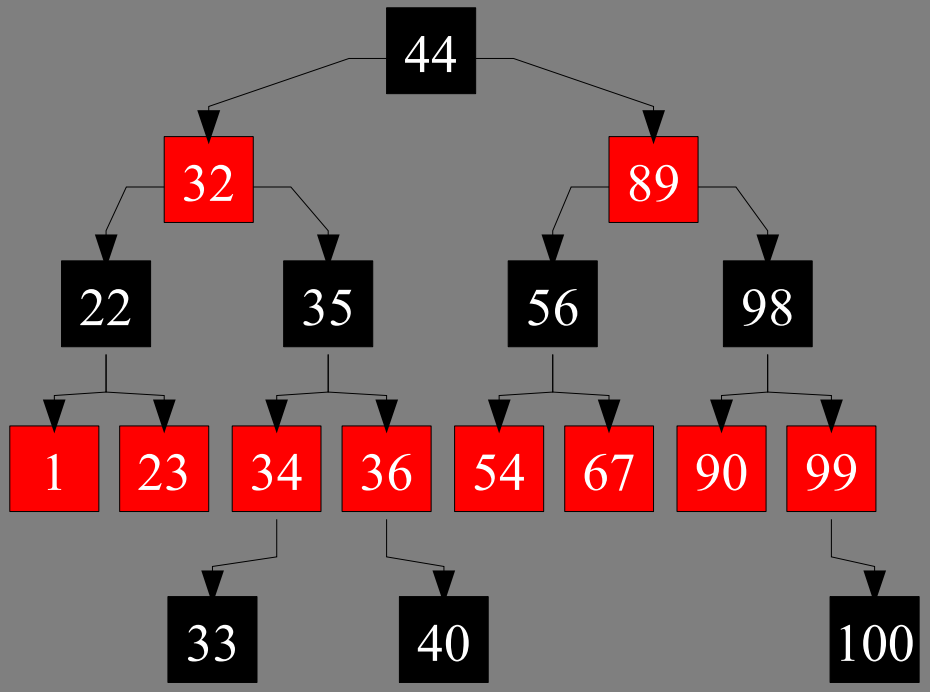
`runheadless()`
loads & prints the file

`KVGLoadFile()`
loads the file into the tree

edit file: `vi` or `nano`

Demo: the lab assignment





Power Management Building Blocks

Power Management 101

Power management:

is a **system level** design goal...

not a software level design goal.

How we save power

Two big ways:

- **Turn stuff off**
  - Clock trees, caches, displays, radios, USB, memory, anything you can get your hands on.
- **Clock stuff down and power it at a lower voltage**
  - $P = CV^2f$ in CMOS
  - Switching capacitance
  - Voltage (which also relates to frequency)
  - Frequency

Five Step Homework Assignment

1. Enumerate system devices
2. Determine degrees of power management freedom for each device
3. Identify constraints
4. Identify product use cases
5. Define power management policies

From the user's perspective…

how you turn *on*

is as important as what you turn *off!*

Define Power Management Policies

Before we get into that we'd better learn what we can control and how we can control it

- Saving power while the CPU is *active*
  - Voltage and frequency scaling of the CPU using “cpufreq”
  - Power Management-aware Drivers
- Saving power while the CPU is *inactive*
  - Idle scaling
  - Dynamic tick
  - Deferrable timers
  - Mitigate wakeups using “PowerTop” and system tuning

Stitching it together

- CPUfreq
  - Create a processor driver: √
  - Define operating points: √
  - Modify *standard drivers* to respond to CPUfreq notifications: √
  - Select and configure the governor
- Power management aware device drivers
  - Implementing power management in a device driver
  - Handling CPUfreq notifications in a device driver
  - Suspend/Resume hooks
  - Clock framework

√ = Implemented for you by MontaVista

The processor driver

Best to consult the kernel source code:

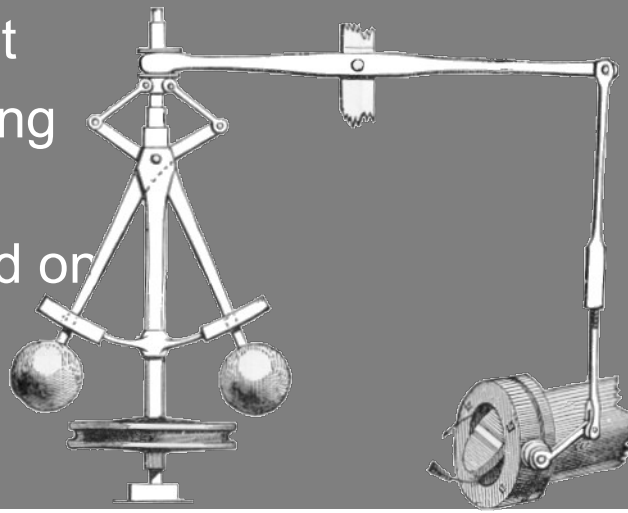
`Documentation/cpu-freq/cpu-drivers.txt`

- `cpufreq_driver.name`
- `cpufreq_driver.owner`
- `cpufreq_driver.init`
- `cpufreq_driver.verify`
- `cpufreq_driver.[setpolicy|target]`
- `cpufreq_driver.exit`
- `cpufreq_driver.resume`
- `cpufreq_driver.attr`

The Governor

Task: Decide how and when to change operating points.

- Four options provided:
  - **performance:**
    statically set highest power operating point
  - **powersave:**
    statically set lowest power operating point
  - **userspace:** permit any application running as `root` to set the operating point
  - **ondemand:** set the operating point based on current CPU usage



More on the *ondemand* Governor

- Works by altering the operating point to *minimize idle time*.
- Lots of control knobs:
  - `sampling_rate`
  - `sampling_rate_max`
  - `sampling_rate_min`
  - `up_threshold`
  - `powersave_bias`
  - `ignore_nice_load`

Making Drivers Power Management Aware

Three areas to focus on:

- Wise power management: minimizing power usage of the driver in regular operations
  - Staying “off” between close() and open()
  - Staying “off” if the transceiver/PHY indicates no connection
  - Gating off unused clocks
  - Switching off unused power
  - Using lower voltages
- System sleep: Preparing the driver to respond to system wide low-power sleep requests
- Responding to cpufreq notifications

System Wide Sleep

- Create a **`struct platform_driver`** in your driver
- Register the platform driver
- Implement driver specific suspend and resume functions
- Use `/sys/power/state` as a test interface

```
#include <linux/platform_device.h>

static struct platform_driver sample_driver = {
  .suspend  = sample_driver_suspend,
  .resume   = sample_driver_resume,
  .driver   = {
    .owner = THIS_MODULE,
    .name  = "sample_driver",
  },
};
```

cpufreq Notifications

Your driver can register with cpufreq to get notified of power events:

- `CPUFREQ_PRECHANGE`: sent immediately before a new operating point is set
- `CPUFREQ_POSTCHANGE`: sent immediately after a new operating point is set
- `CPUFREQ_RESUMECHANGE`: sent if the cpufreq subsystem determines that an operating point was changed during system suspend

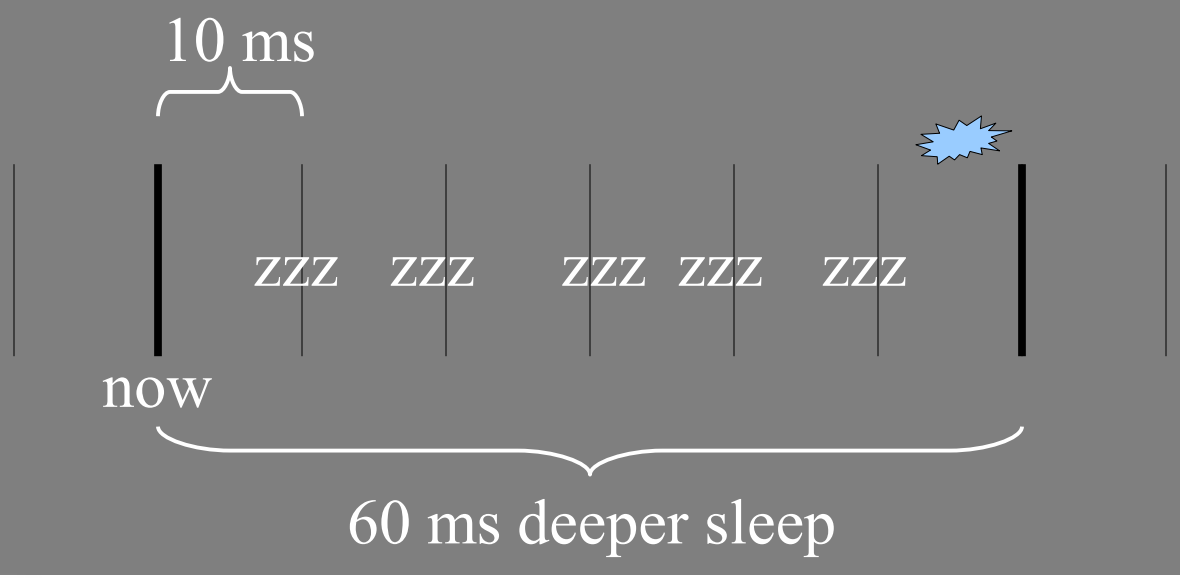
Define Power Management Policies

Before we get into that we'd better learn what we can control and how we can control it

- ✔ Saving power while the CPU is *active*
  - ✔ Voltage and frequency scaling of the CPU using cpufreq
  - ✔ Power Management-aware Drivers
- **Saving power while the CPU is *inactive***
  - Idle scaling
  - Dynamic tick
  - Deferrable timers
  - Mitigate wakeups using PowerTop and system tuning

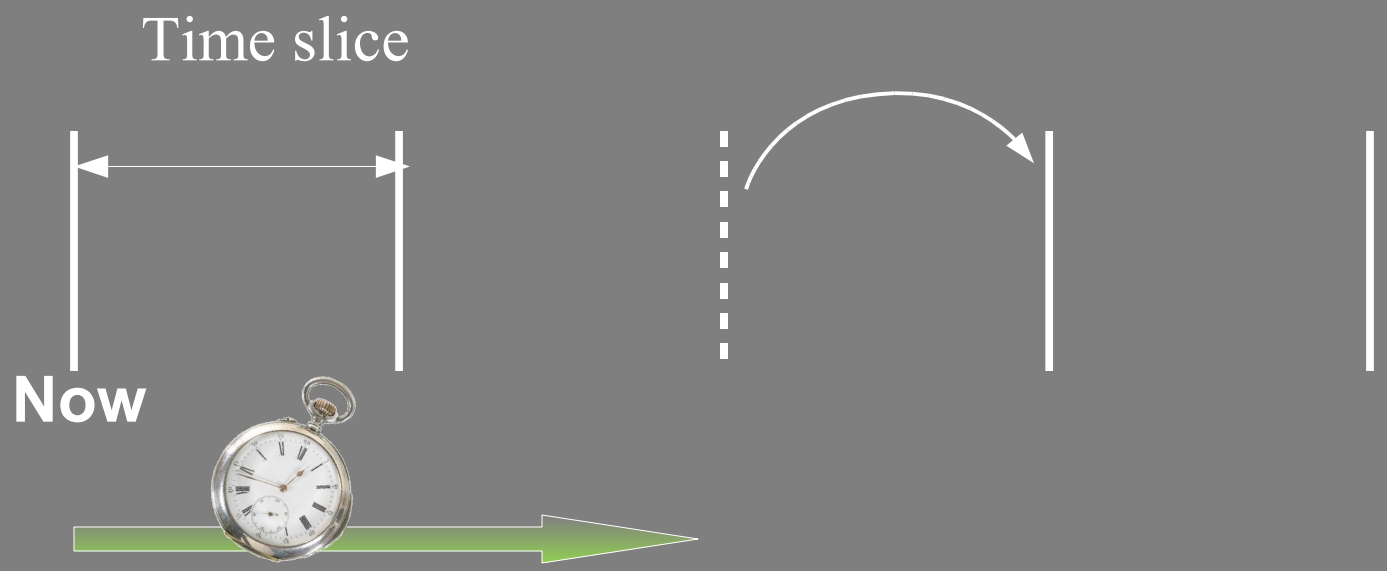
Saving power during idle

- **Idle Scaling**: Reduce power consumption during idle periods
  - If you've done cpufreq well, you've got the job done already!
- **Dynamic Tick**: coalescing ticks to avoid unnecessary wakeups



Deferrable Timers

- API so that drivers can notify the kernel that the timer wakeup is needed but that the precise time of the wakeup is flexible.
- Use `init_timer_deferrable()`
- Example usage: flashing LED that indicates an email has arrived

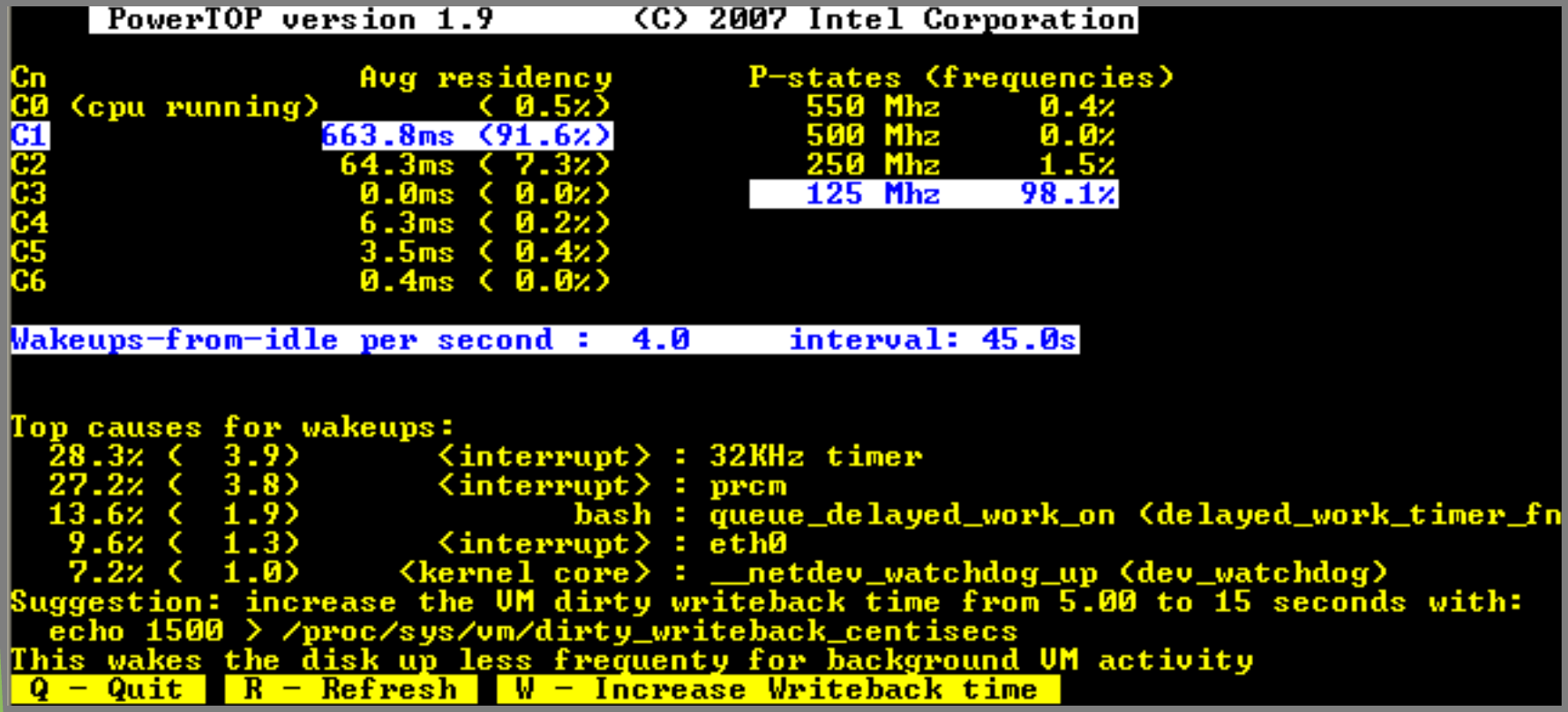


Are we finished now?

Let's assume you finished your kernel stuff…

Question: What about applications and other stuff you didn't write???

Demo: hunt power waste using PowerTop



Recommended Reading

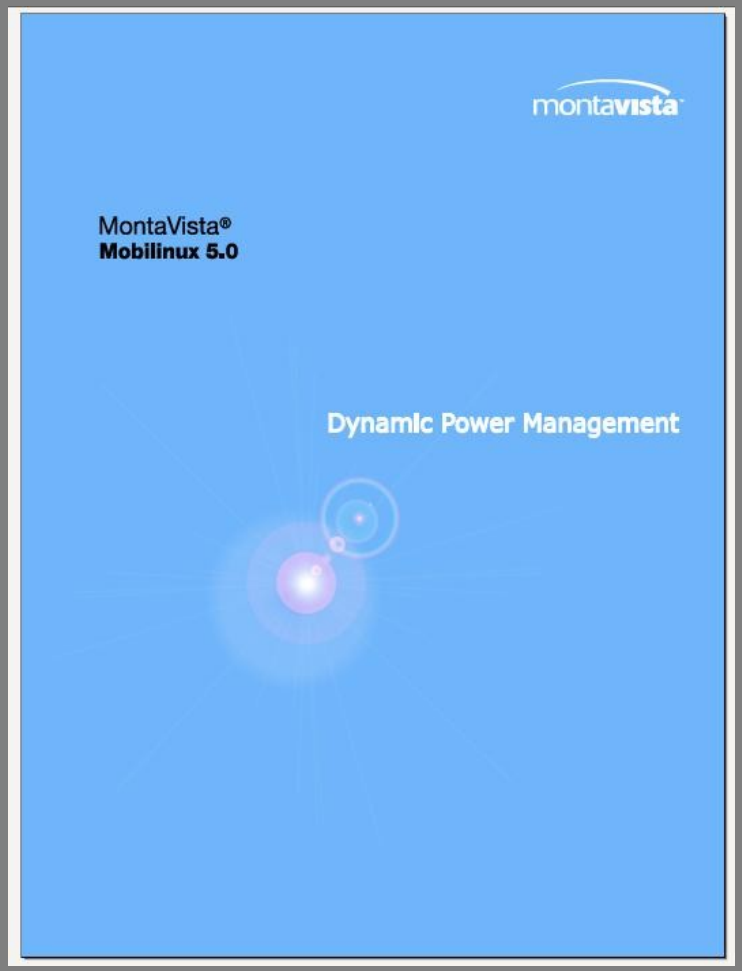
- **Benchmarking of Dynamic Power Management Systems**

  Frank Dols

  CELF Embedded Linux Conference 2007, Santa Clara

Linked from http://www.mvista.com/power

Free for the asking...



http://www.mvista.com/power

montavista™

Lab: let's hunt!

A daemon runs in `/drop`

- files put in there will be SHA1-checksummed

The daemon is using the (fixed) redblack code

- But there are two things wrong...
- What are they?

The sha_daemon problems

```
for(;;)
{
select(0, NULL,
NULL,
    NULL, &tv);
…
Getthetime();
}
```

Select()

- Wakes up every time tick...

Getthetime()

- Doesn't exactly work as advertised, either :-)

Summary

- Linux system design isn't trivial
  - Many building blocks
  - Even more blocks when you want to debug
- Debugging is useful
- Power management is not trivial either
  - But doable as many Linux cell phones prove
  - Use `powertop`
  - Toy with Governors
- Don't forget to have fun!

Klaas.van.Gend@mvista.com