Linux perf

Locate performance bottleneck with low overhead

Geoffrey Papaux, geoffrey.papaux@deltaww.com

May 18, 2017

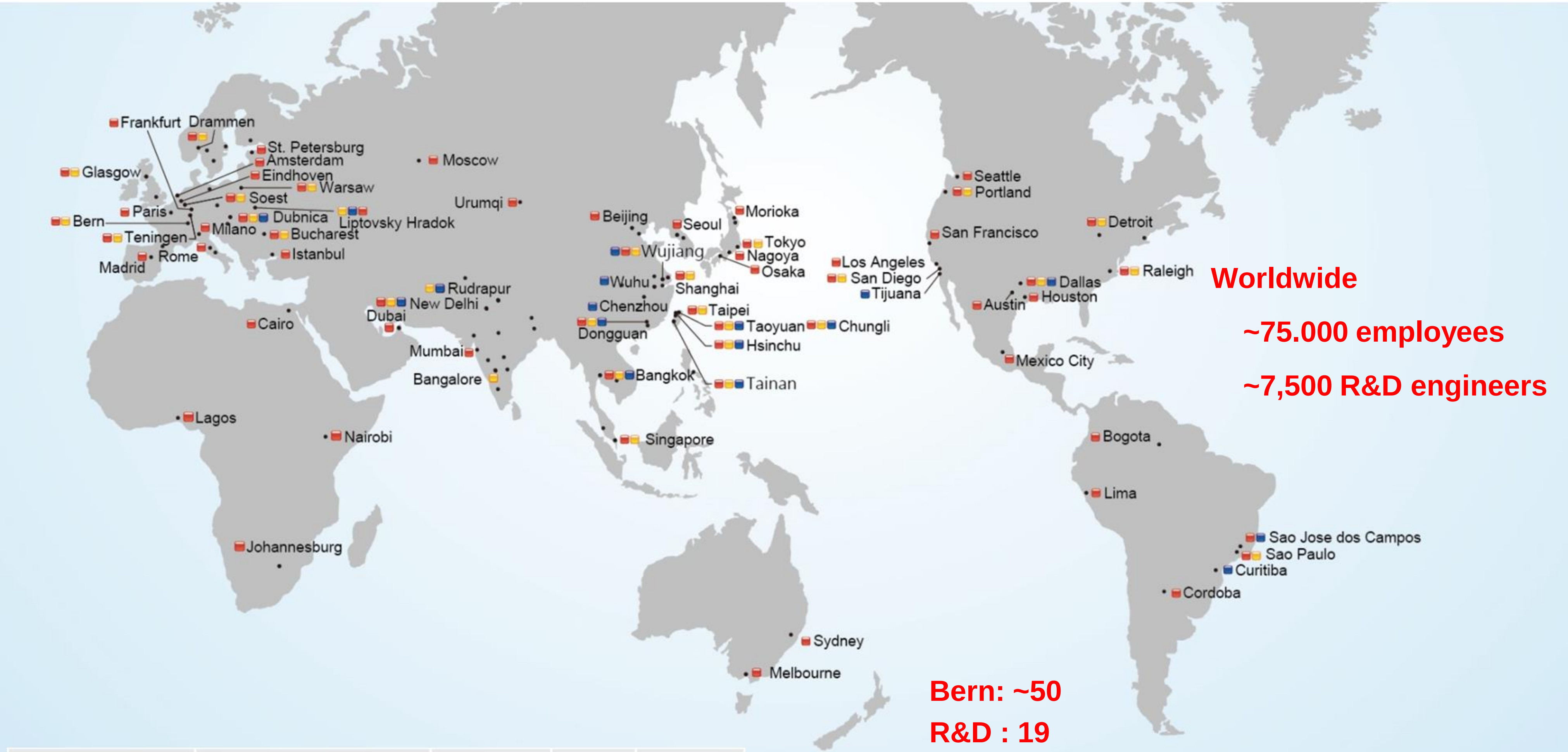
Agenda

- About
- Profiling
- Performance Counters
- Perf
  - Architecture
  - Usage
  - Advanced options & use
  - Typical issues
- Conclusion

About me

- Bachelor / Master @ HEIA-FR / HES-SO
- Embedded SW Engineer @ Delta Energy Systems
- «Experience» with perf ?
  - Bachelor Thesis: *Perf-based Profiles for GCC*
  - Master Thesis: *Virtualization on ARM*
  - Occasionally at Delta ☺

Delta Worldwide



Worldwide

~75.000 employees

~7,500 R&D engineers

	Asia-Pacific (China)	Americas	EMEA	Total
🟥 Sales Offices	**96 (52)**	**20**	**37**	**153**
🟦 Plant Sites	**34 (22)**	**4**	**2**	**40**
🟨 R&D Centers	**43 (23)**	**7**	**11**	**61**

Bern: ~50

R&D : 19

Software Engineer: 9

Hardware Engineer : 2

Test Engineer: 2

Support : 3

DELTA

Business Categories

Power Electronics

- Embedded Power Supplies
- Mobile Power Supplies
- Industrial and Medical Power Supplies
- Fans and Thermal Management
- Electronic Components for ICT Equipment

Energy Management

- Industrial Automation
- Telecom Power Systems Delta Bern
- UPS & Datacenter Infrastructure
- Automotive Electronics & EV Charging
- Renewable Energy
- Energy Storage Systems

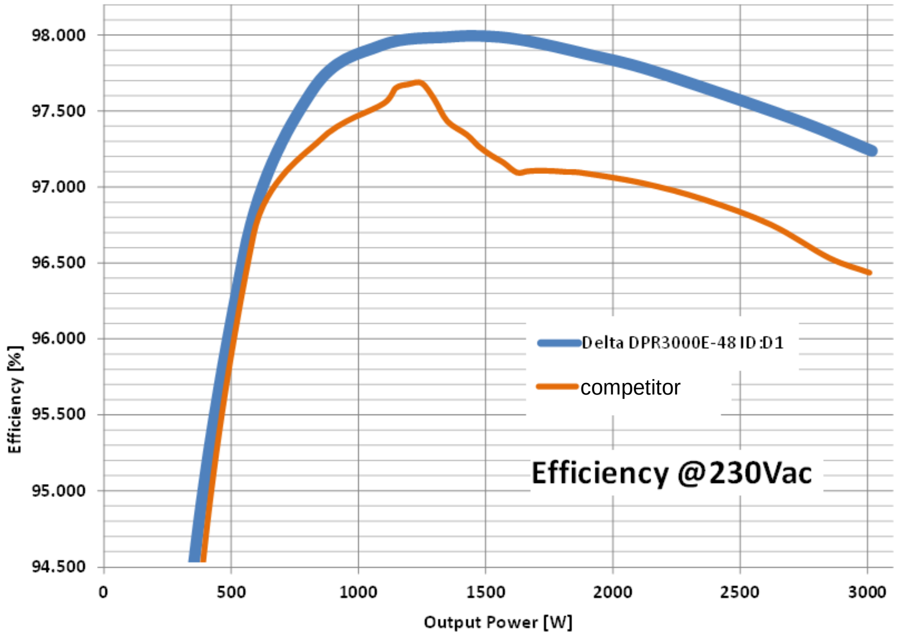
Smart Green Life

- Networking Systems
- Display & Visualization
- LED Lighting
- Healthcare Devices
- Innergie
- vivitek

R&D Bern: Telecom Power Systems

- 4th Gen Power System Controller ORION
- Color 2” Touch screen
- ~100’000 units per year
- Software continuously enhanced
  - > 40 software version
  - > 700’000 lines of code
  - > 2000 configuration parameters
  - > 200 web pages
- > 50 different power converter or extension modules

- 48V DC Rectifier 3000W
- **98% efficiency**
- 56.8 W/inch^3 power density



Profiling ?

When and why do I need it ?

HELP! My app is slow

- Slower than before
  - after adding a new functionality
  - after upgrading an external library
  - with a special kind of input / workload
  - after changing the hardware platform
- Needs to run faster
  - to satisfy timing constraints
  - to use less resources (save cloud costs)

- **=> I want to locate the bottleneck!**

Profiling

- Art of collecting and analysing data
  - *dynamic* analysis (vs *static* analysis)
- Locate **hot spots**
  - where is the execution time spent ?
  - how often is this function called and from where ?
- Resolution
  - function level
  - instruction level
- Two categories: instrumentation or sampling

Instrumentation

- The monitored program is modified
  - at source code level (`clock_gettime()`)
  - at compile time (gprof)
  - at execution time (valgrind)

(+) provides exact measurements

(-) high overhead (>= 30% for gprof)

(-) need special binary / compilation (not for valgrind)

(-) program behavior is altered

Sampling

Or **statistical profiling**

- The program runs unmodified
- A tool records the program state at regular intervals
  - by probing call stack
  - by using hardware capabilities

(+) very low overhead (1-2%)

(+) no special binaries needed

(+) behavior is unaltered

(-) not 100% accurate (but usually enough)

Profiling is not straightforward

- Today's architectures are complex
- Bottleneck is not only about instructions executed
  - memory access ? cache L1, L2, L3 ?
  - branch prediction ?
  - Instruction pipeline ?
- And not only hardware
  - context switch ?
  - CPU migration ?

Hardware to the rescue

Performance counters

- CPU registers counting hardware events
  - CPU Cycles
  - Branch-misses / Cache-misses
  - and many more
- Supported by Intel, AMD, ARM, PowerPC, ...
- Architecture specific
- Often referenced as
  - *PMU (Performance Monitoring Unit)*
  - *PMC (Performance Monitoring Counters)*

perf

THE performance analysis tool for Linux.

perf provides rich generalized abstractions over hardware specific capabilities

https://perf.wiki.kernel.org/index.php/Main_Page

perf reads counters

- **Hardware** counters from PMU, but not only.

- Lot of **software** counters:
  - Context-switches
  - CPU Migrations
  - Kernel tracepoints
  - Page faults
  - ...

perf architecture

- perf is a *framework*
- **Back-end** must be enabled in your kernel config:

```
  -> General setup
      [*] Kernel Performance Events And Counters
```

- **Front-end** is in Linux Kernel source tree
  - In directory `tools/perf`
- Wait. User-space application in Kernel Source ?
  - => tightly coupled to Linux kernel ABI
  - => require exact version matching

Getting perf

- From your distribution

```
apt-get install linux-tools-common linux-tools-generic
pacman -S perf
dnf install perf
```

- As a Yocto package

```
IMAGE_INSTALL = "perf"
```

- Compile from source

```
make -C tools/perf
```

Using perf

perf tool

- git-like interface
- Usage via subcommands

```
perf <subcommand>
```

- List all subcommands

```
perf
```

- Getting help about a subcommand

```
perf help <subcommand>
```

perf subcommands

Command	Description
`list`	list of available events
`stat`	collect events while running a command
`record`	record events in perf.data file
`report`	analyse a perf.data file
`script`	scripting interface for processing perf.data file
`top`	performance counters in real-time
`bench`	micro-benchmarking framework
`...`	…

- 25+ commands

perf

- List available commands

```
$ perf
 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate      Read perf.data (created by perf record) and display annotated code
   archive       Create archive with object files with build-ids found in perf.data
   bench         General framework for benchmark suites
   buildid-cache Manage build-id cache.
   buildid-list  List the buildids in a perf.data file
   config        Get and set variables in a configuration file.
   data          Data file related processing
   diff          Read perf.data files and display the differential profile
   evlist        List the event names in a perf.data file
   ftrace        simple wrapper for kernel's ftrace functionality
 [...]
```

perf list

- List available events (hardware and software)

```
$ perf list
List of pre-defined events (to be used in -e):

  branch-instructions OR branches        [Hardware event]
  branch-misses                          [Hardware event]
  cache-misses                           [Hardware event]
  cache-references                       [Hardware event]
  cpu-cycles OR cycles                   [Hardware event]
  instructions                           [Hardware event]

  alignment-faults                       [Software event]
  context-switches OR cs                 [Software event]
  [...]
```

perf stat

- Collect counter statistics while running a command

```
$ perf stat <command>

 Performance counter stats for '<command>':

      7934.312649      task-clock:u (msec)    #    1.000 CPUs utilized
                0      context-switches:u     #    0.000 K/sec
                0      cpu-migrations:u       #    0.000 K/sec
              106      page-faults:u          #    0.013 K/sec
   35,688,627,718      cycles:u               #    4.498 GHz
   92,920,415,499      instructions:u         #    2.60  insn per cycle
   14,406,550,955      branches:u             # 1815.728 M/sec
      258,189,358      branch-misses:u        #    1.79% of all branches

      7.934377499 seconds time elapsed
```

Try again with -d

perf top

- Display event counters in real time (top-like)

```
$ perf top

Samples: 482  of event 'cycles', Event count (approx.): 4563253774
Overhead  Shared Object      Symbol
  43.84%  solver             [.] checkRow
  16.96%  solver             [.] checkSquare
  16.73%  solver             [.] checkColumn
   8.50%  solver             [.] placeNum
   2.57%  [kernel]           [k] format_decode
   2.05%  solver             [.] goBack
   1.81%  solver             [.] solveSudoku
   0.51%  perf               [.] rb_next
[...]
```

perf record

- Collect profiling data
- Data written in `perf.data` file
- Run an application and record

```
$ perf record <command>

[ perf record: Woken up 5 times to write data ]
[ perf record: Captured and wrote 1.244 MB perf.data (32133 samples) ]
```

- Collect system-wide

```
$ perf record -a
```

perf report

- Read and analyse `perf.data` previously recorded

```
$ perf report -n --stdio

# Samples: 32K of event 'cycles:u'
# Event count (approx.): 35983350419
#
# Overhead       Samples  Command  Shared Object      Symbol
# ........  ............  .......  ................   ..................
#
    40.69%         13073  solver   solver             [.] checkRow
    24.49%          7869  solver   solver             [.] placeNum
    17.71%          5689  solver   solver             [.] checkSquare
     9.40%          3021  solver   solver             [.] checkColumn
     4.16%          1337  solver   solver             [.] goBack
     3.54%          1136  solver   solver             [.] solveSudoku
```

Interactive analysis with

```
$ perf report
```

Advanced options

- Scope
  - `-a` system-wide collection (all CPUs)
  - `-p <pid>` from a running process
  - `-t <tid>` from a running thread
- Collect specific event(s)
  - `-e <event>`
- Collect at a given frequency (Hz)
  - `-F <freq>`
- Record call graph
  - `-g`

Advanced usage (1)

- «Cross-analysis»
  - Collect on embedded system
  - Analyse perf.data from another machine

```
$ perf report --objdump=<path> --symfs=<path-to-debug>
```

- Filter user / kernel

```
$ perf record -e cycles:u,cache-misses:k -a
```

- View report by source lines

```
$ perf report -s srcline
```

- Memory access analysis

```
$ perf mem [record|report]
```

Advanced usage (2): scripting

- Built-in scripts

```
$ perf script –l
```

- Generate script

```
$ perf script -g python
$ python perf-script.py
```

- Example: record failed syscalls

```
$ perf script record failed-syscalls
$ perf script report failed-syscalls
```

Advanced usage (3)

- Dynamic tracing

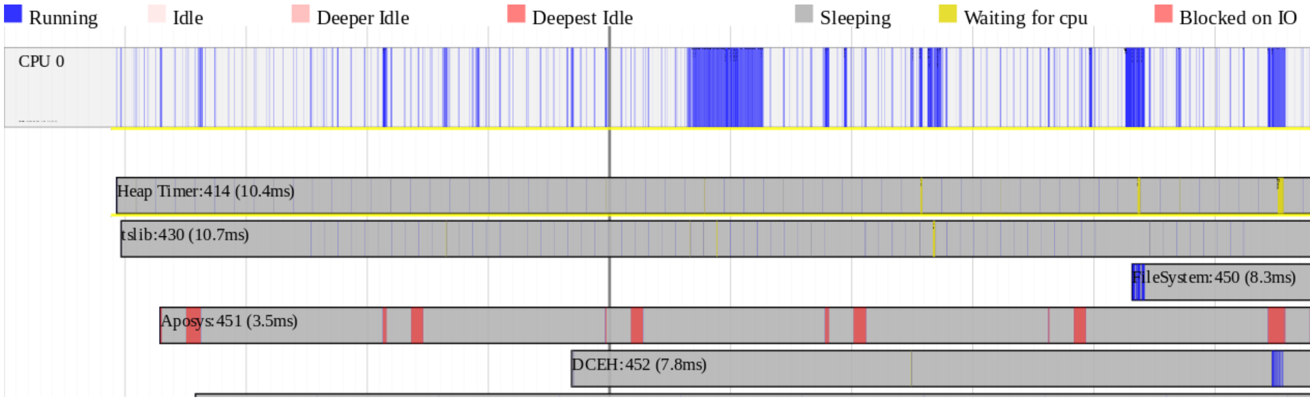
```
$ perf probe /lib/libc.so.6 malloc
$ perf record -g -e probe_libc:malloc –aR
$ perf report
```

- Scheduler

```
$ perf sched record
$ perf report
```

- «Timechart»:

```
$ perf timechart
```



Accuracy

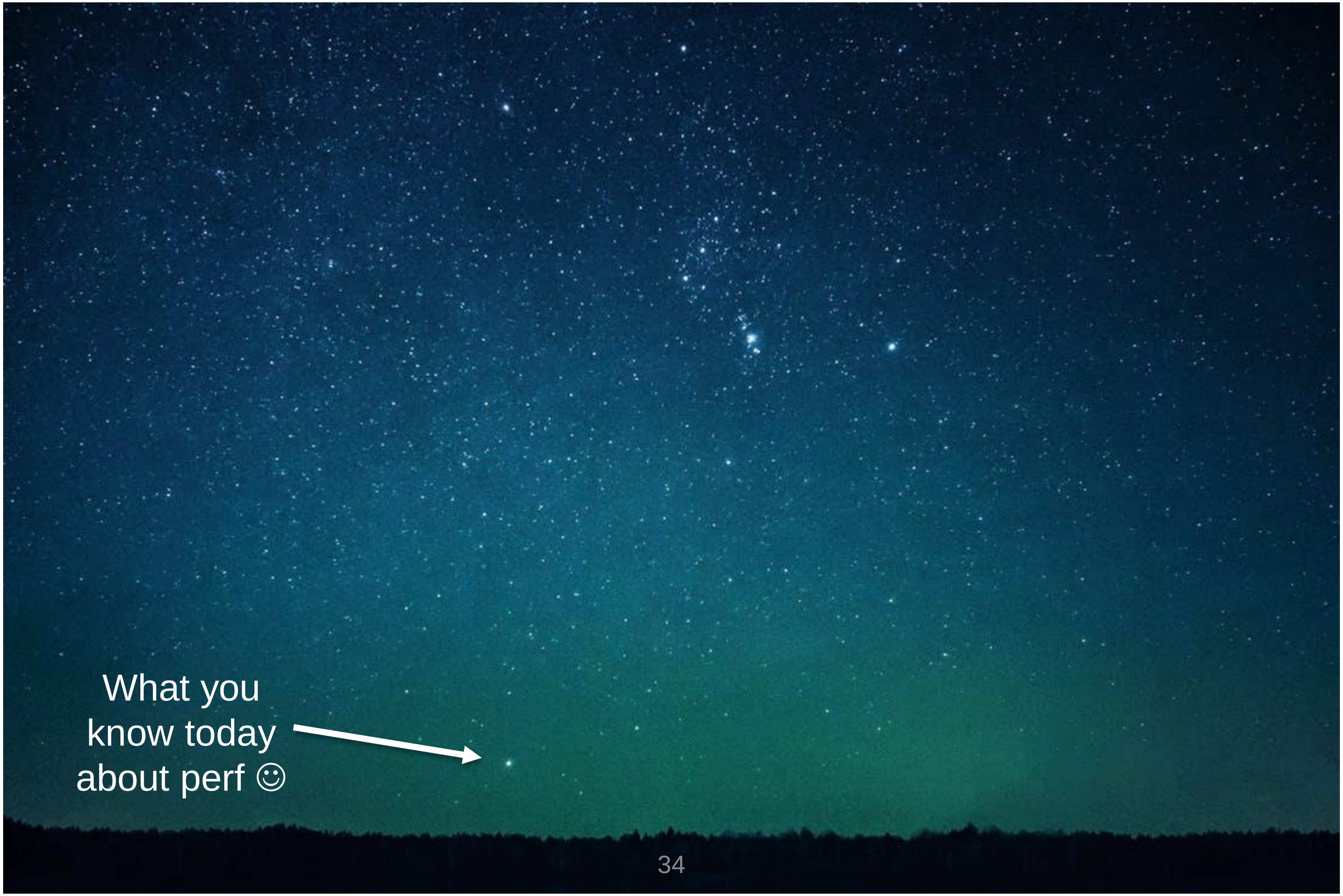
> *The Performance Monitors provide approximately accurate count information. To keep the implementation and validation cost low, a reasonable degree of inaccuracy in the counts is acceptable. ARM does not define a reasonable degree of inaccuracy.*

ARM Architecture Reference Manual (ARMv7-A)

Typical issues

- Stack not working (=> `-fno-omit-frame-pointer`)
- Debug symbols not available
- No PMU support (typical `<not supported>` message)
  - virtual machine / cloud
  - not available on your hardware
  - not implemented for your target
  - sometimes just not enabled in device tree
- Sudo is your friend, even for perf list

Conclusion



What you
know today
about perf ☺

Conclusion

- perf is a powerful, low overhead profiling tool
- Actively developed https://kernelnewbies.org/LinuxChanges
  - Getting new functionality with every kernel release
  - Dedicated section in change log
  - Listed in *Prominent Features* in 4.11, 4.10, 4.7, 4.4
- Available for Linux > 2.6.31
- ! Understand what you are measuring
- Make sure it is working before making any conclusion

Documentation

- Perf wiki: https://perf.wiki.kernel.org
- Kernel documentation: tools/perf/Documentation/
- Architecture reference manuals
  - INTEL: http://www.intel.com/Assets/PDF/manual/253669.pdf
  - ARM: http://support.amd.com/us/Processor_TechDocs/31116.pdf
- Brendan Gregg, Netflix Performance Engineer http://www.brendangregg.com/perf.html

Thanks for your attention

Smarter. Greener. Together.

To learn more about Delta, please visit www.deltaww.com.