Patrick Bellasi
bellasi@elet.polimi.it

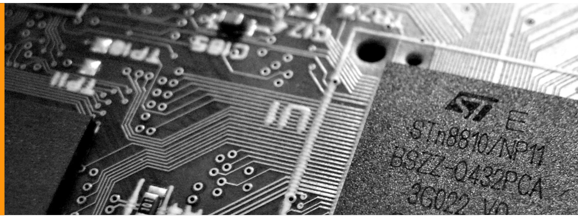
Linux Power Management Architecture

An updated review on Linux PM frameworks

Last update: 6/3/08

Patrick Bellasi bellasi@elet.polimi.it

Agenda



- Outline the architecture for advanced power management support in recent Linux kernels

  key points for effective PM and overall picture view

- Review of main subsystems
  - Advanced timekeeping framework
  - Clock framework
  - Voltage/Power control Framework
  - QoS Framework
  - CPUIdle
  - CPUFreq
  - Linux PM

  Why we need them?

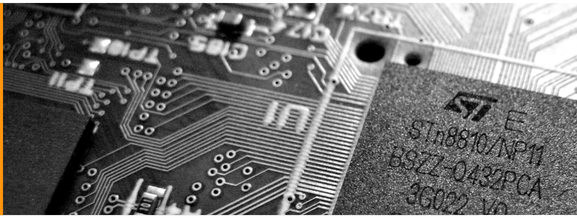
  What they do?

  How we can exploit them?

Patrick Bellasi bellasi@elet.polimi.it

Modern PM architectures

Key points for effective power management



- Exploit partial activity
    - disable parts of the system when not needed
- SW does part of the work, HW dependencies do the rest
    - exploit existing system framework
        - *track dependencies (producer-consumer)*
        - *track usage (reference counting)*
        - *system constraints assertion*
        - *notification-chains*

    Fine grained tracking and constraints

    - drivers support required
        - *aggressively get and release resources*
        - *support efficiently OFF mode (0 volts)*
            - full context loss, low-latency wakeup,
            - provide context saving-restore routines
        - *use constraints to require operational restrictions*

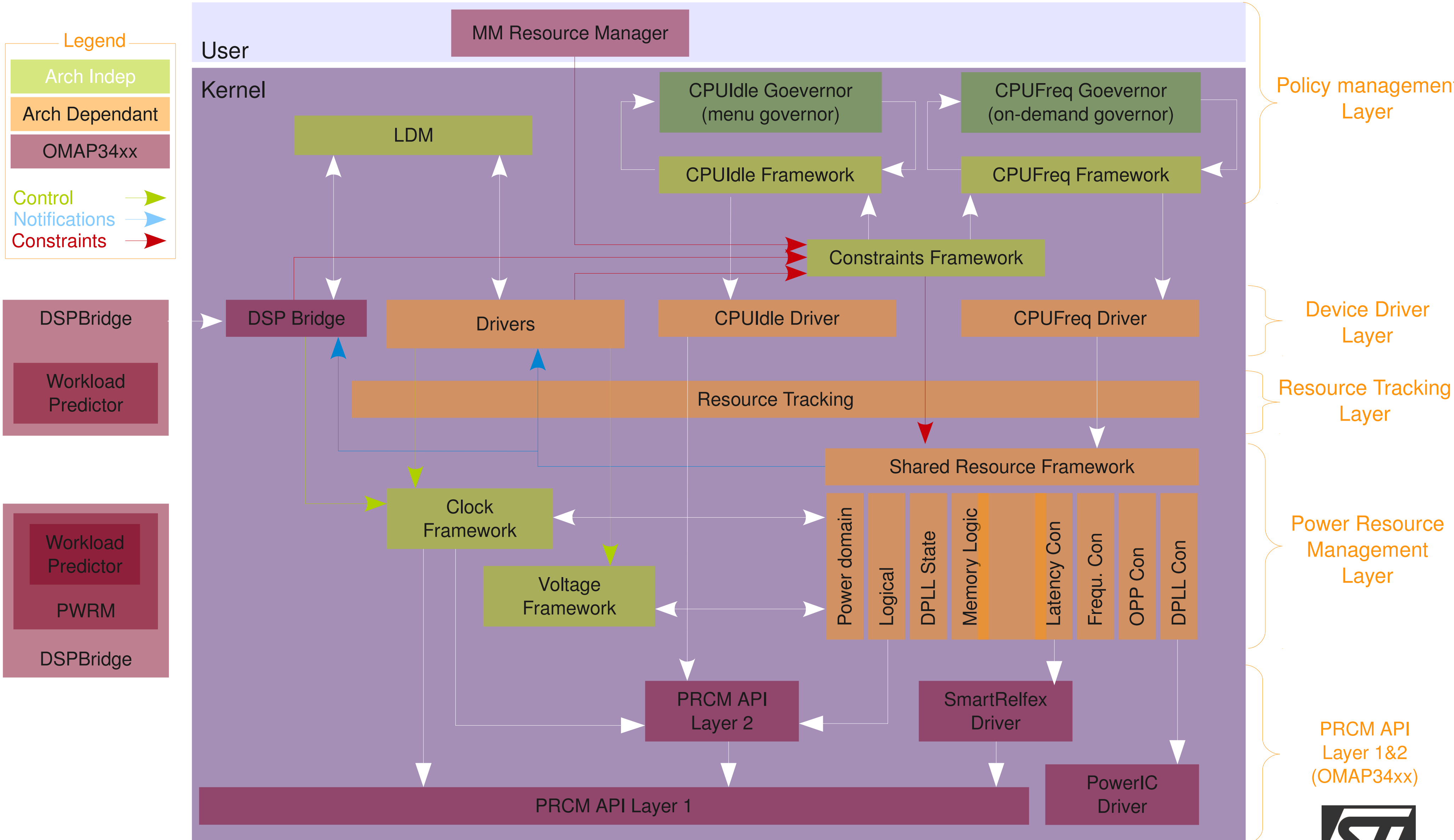
Patrick Bellasi bellasi@elet.polimi.it

- Control power sources
    - *clocks, power domains, voltage domains, memories and power IC resources*
    - *device drivers should exploit interfaces to control power resources*
- Constraints aware drivers and subsystems
- *Inactive state*
    - power saving in OS idle
        - *automatic choice between multiple C-States*
            - OFF and retention modes
        - *manual suspend/resume*
        - *system-wide sleep states*
- *Active state*
    - dynamic power management
        - *automatic choice between multiple P-States*

Patrick Bellasi bellasi@elet.polimi.it

Modern PM architectures

what we should (already) have



Linux frameworks to support PM

Introduction

- Typical mainline Linux Power Management features:
  - Platform-specific code
    - *idle-loop, timekeeping (dynamic tick) & clock tree, latency*
  - Dynamic Power Management
    - *Low-power states activation for unused devices*
    - *C-States, LDM, Suspension (RAM), Hibernation (Disk)*
  - Dynamic Voltage and Frequency Scaling
    - *Adapting device performances to application needs*
    - *P-States, OPP*
- Main focus on x86 arch
    - *custom and different PM development for SoC embedded devices*
    - *2rd Linux PM Summit was only on 2007*
    - *increasing emphasis on embedded systems*

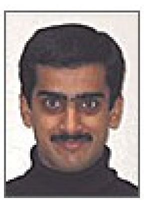
Linux frameworks to support PM

Timeline

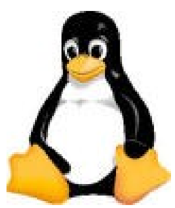
April 2008

2.6.9

CPUFreq OnDemand Governor



Venkatesh Pallipadi



John Stultz

Generic Time-of-Day

2.6.16

Hrtimers base-code

Ingo Molnar

Thomas Gleixner

2.6.21

Clock Source and Clock Event

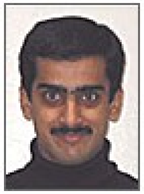
Generic Dynamic Tick

Clock framework

Dynticks

2.6.24

CPUIdle framework



Venkatesh Pallipadi

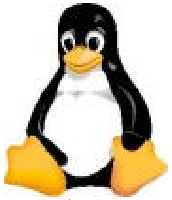
2.6.25

QoS Framework

Mark Gross

2.6.27

Regulators Framework



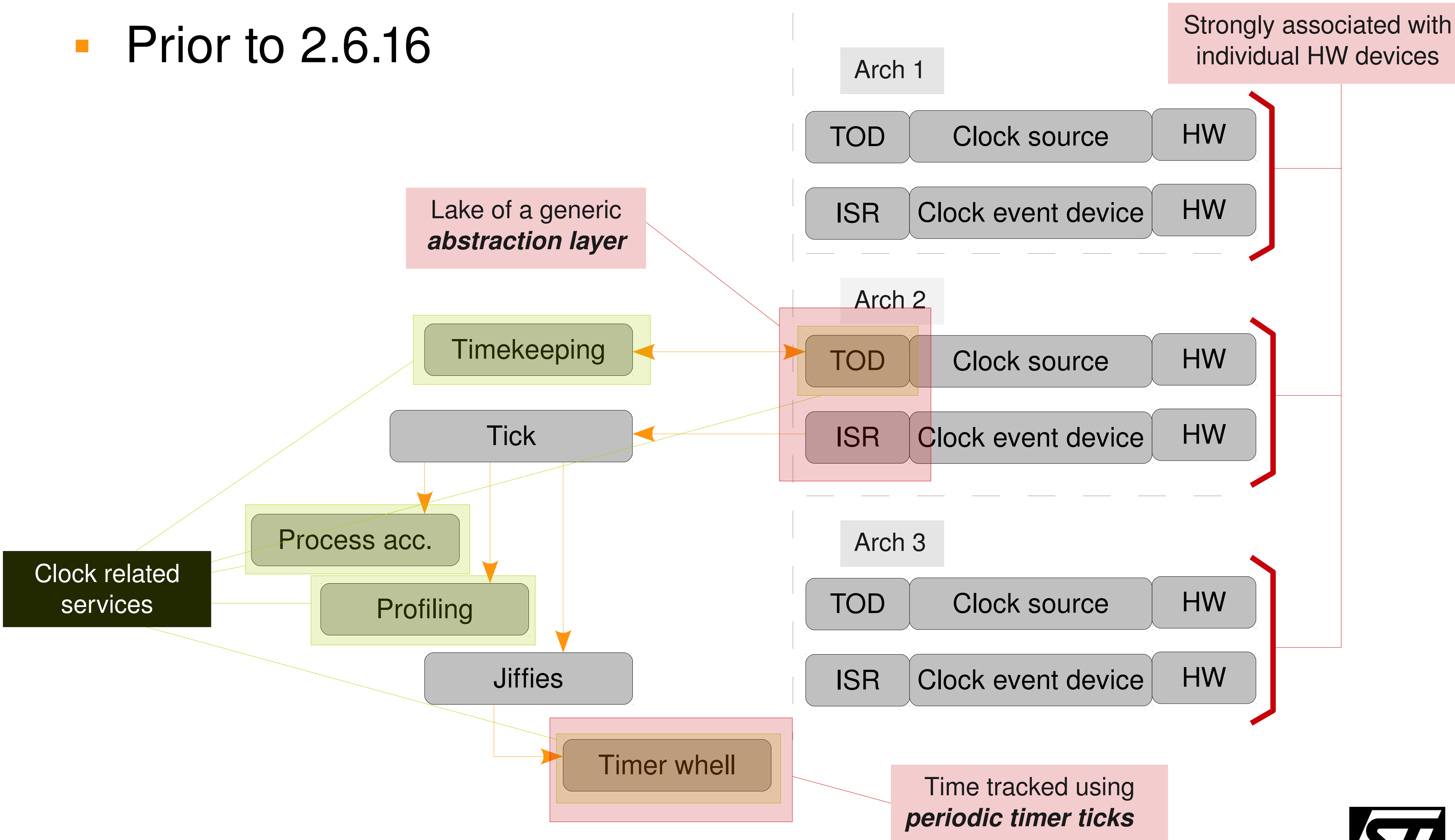
Liam Gridwood

Patrick Bellasi bellasi@elet.polimi.it

The Linux Time System

The old tick-based implementation

- Prior to 2.6.16



Patrick Bellasi bellasi@elet.polimi.it

- Required abstractions
  - Clock source (CS) management
    - provide read access to monotonically increasing time values
    - use human-oriented values (nanoseconds)
  - Clock synchronization
    - system time value and clock source drifts correction
    - use reference time sources (e.g. NTP or GPS/GSM)
  - Time-of-day representation
    - applying drifts corrections to clock source
  - Clock event device (CED) management
    - schedule next event interrupt using clock event device
    - minimize arch dependent code, allow easy run-time addition of new CED
  - Removing tick dependencies
    - better support high resolution timers and precise timer scheduling
    - by replace the CTW (Cascading Timer Wheel) mechanism

Patrick Bellasi *bellasi@elet.polimi.it*

- Why a new Timer Subsystem?

  CTW (1997) has unacceptable worst case performances

  Insertion is O(1), but cascading require interrupt disabling...

  CTW is the better solution for long-term protocol-timeout related timers:

  - expiration time within the first category
  - removed before they expire or have to be cascaded

  is based on periodic interrupt (jiffies)

  support only low-resolution time values

- Propose solution: use two category for timers

  Timeouts – low-resolution, almost always removed

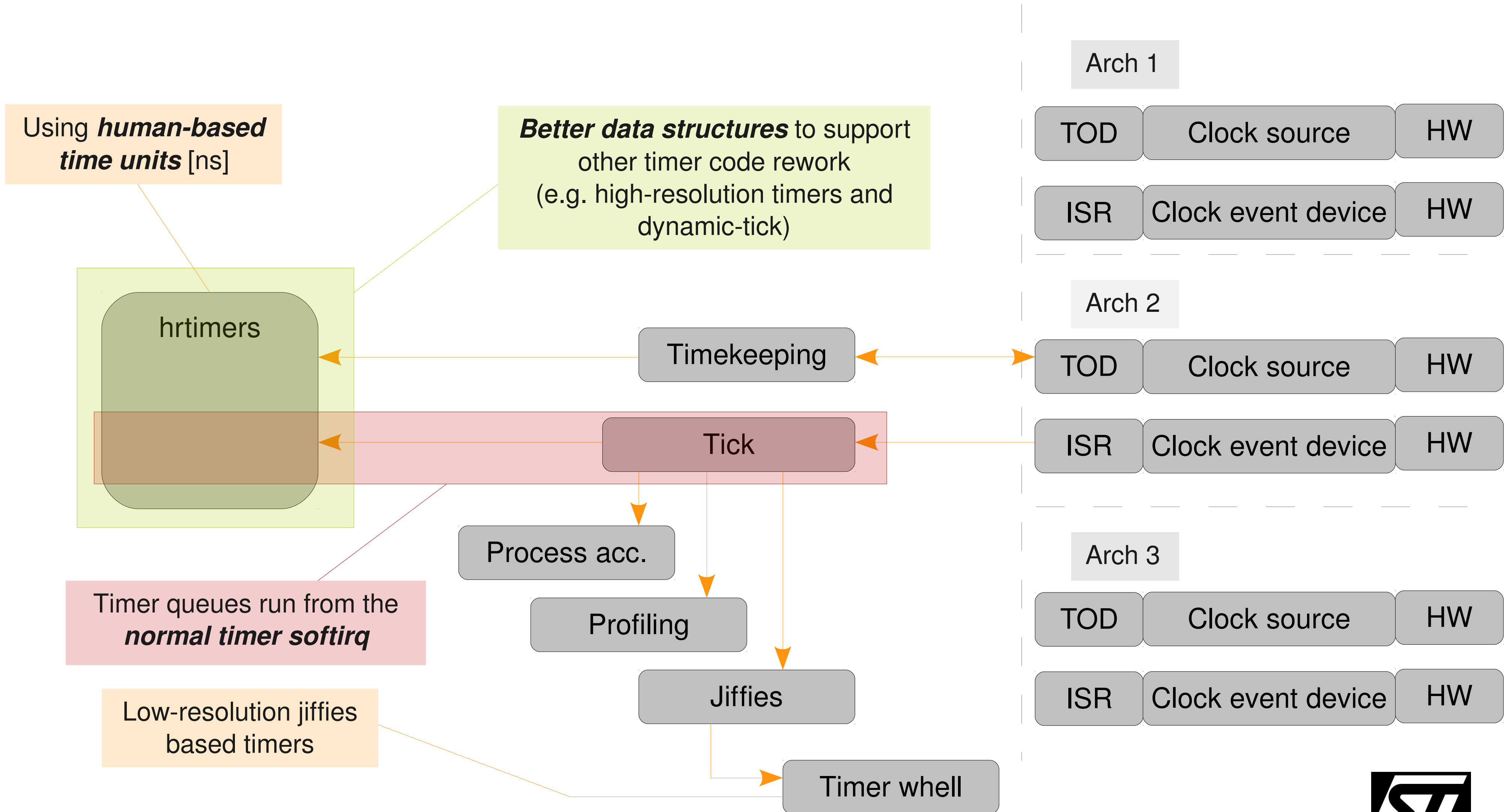
  using the existing CTW

  Timers – high-resolution, usually expire

  using the new hrtimers framework, based on human time units [ns], kept in per-CPU time sorted list implemented using red-black tree (O(log N))

Patrick Bellasi bellasi@elet.polimi.it

- Since 2.6.16



Patrick Bellasi *bellasi@elet.polimi.it*

The Linux Time System [5]

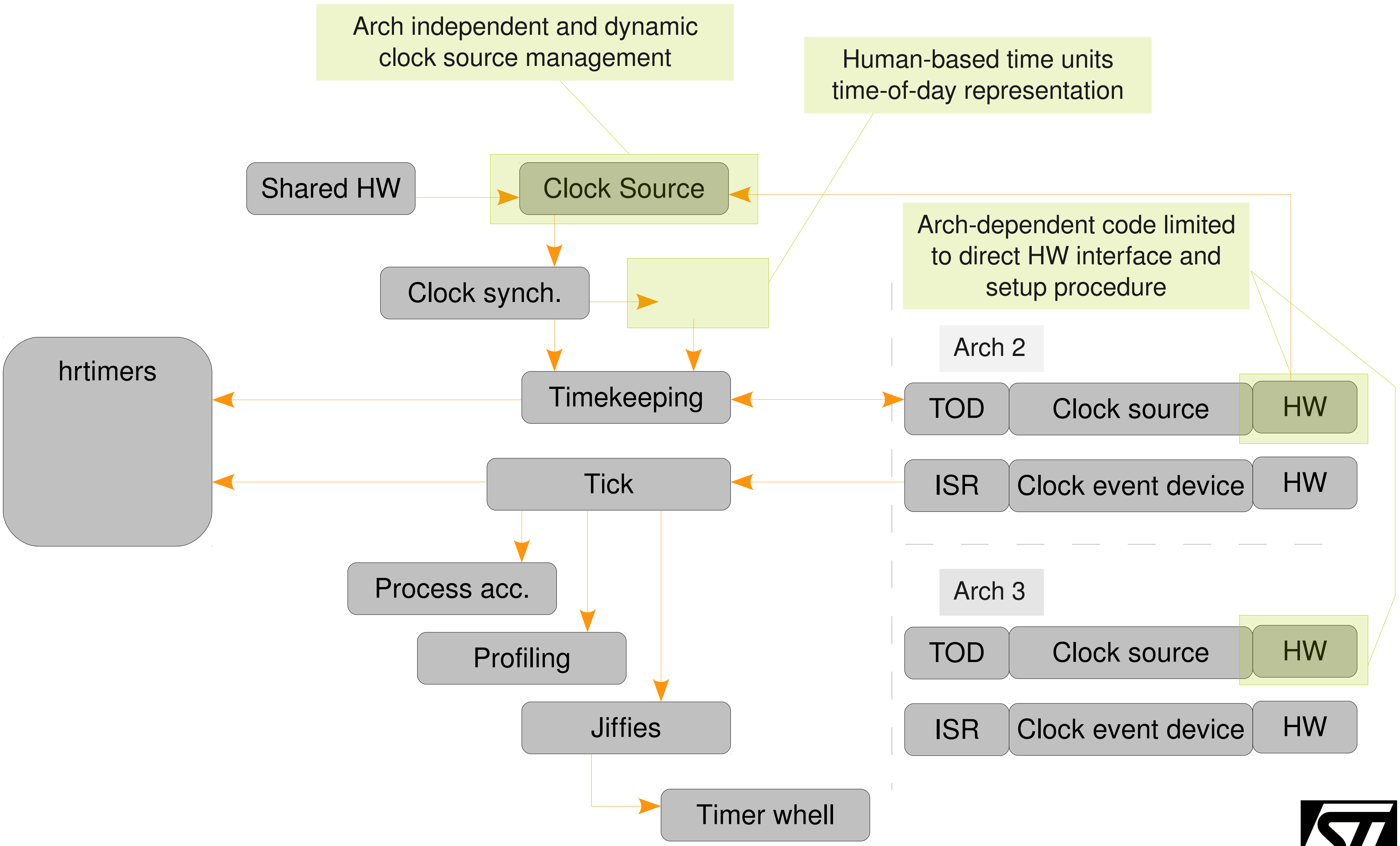
The Generic Time-of-Day (GTOD)

- Why a Generic Time of Day Implementation?
  - allow sharing of clock source code across archs
    - *move large portion of code out of arch-specific areas*
    - *limit arch-dependent code to HW interface and setup*
  - avoid interpolation based computation of human-time values
    - *compensate clock drifts as clock source offsets to get TOD*
    - *previous implementation track compensated TOD directly and derive increasing human-time values from it (error addition problem)*
- Proposed solution: a new generic TOD framework
  - presented by *John Stultz* at OLS 2005, merged in 2.6.16
    - *cleanup and simplify by arch-independent common code*
    - *use nanoseconds as fundamental time unit*
    - *modular design supporting run-time add/remove of time source*
    - *break tick-based dependency to avoid interpolation issues*

Patrick Bellasi bellasi@elet.polimi.it

The Linux Time System

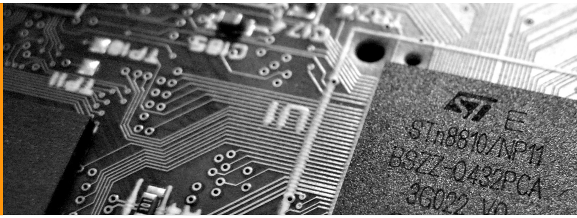
The Generic Time-of-Day (GTOD)



Patrick Bellasi bellasi@elet.polimi.it

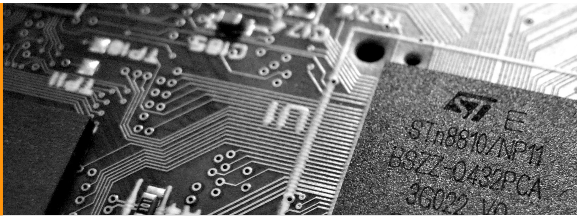
The Linux Time System [6]

The Clock Event Device Abstraction



- Why a Clock Event Device Abstraction?
  - provide an abstraction level for timekeeping and time-related activities
    - *substantial reduction in arch-dependent code*
    - *support either periodic or individual programmable events*
    - *build the base for a generic dynamic tick implementation*
- Proposed solution: by Thomas Gleixner, merged in 2.6.21
  - *Registration interface: run-time configuration of clock event device*
    - property based definition (e.g. capabilities, max/min delta, mult/shift)
    - support generic interrupt setup code (i.e. call-back based handlers)
    - support for power-management
  - *Event distribution interface: bind clock events related services to clock event sources*
    - classical time services (e.g. process accounting, profiling, periodic tick)
    - next event interrupt (e.g. high-resolution timers, dynamic tick)

Patrick Bellasi *bellasi@elet.polimi.it*



The Linux Time System [6]

High Resolution Timers Support

- Why high-resolution timers?
  - exploit brand-new time subsystems reworks (GTOD, hrtimers)
    - *require just arch-independent code by modifying hrtimers framework*
  - accurate delivery of high-resolution timers
- Proposed solution:
  - presented by Thomas Gleixner at OLS 2006
    - *switch to high-resolution mode late in the boot process*
      - exploiting clock source and clock event device dynamic registration
      - ensure kernel compatibility on non high-resolution platforms
    - *support next event scheduling and next event interrupt handler*
      - softirq based execution of hrtimer queues, independent from tick bound
      - callbacks from next-event handler (with some limitations, e.g. for nanosleep)
      - notification to clock event distribution code about periodic interval expiration

Patrick Bellasi *bellasi@elet.polimi.it*

The Linux Time System [7]

The Dynamic Tick Support

- Why a Tick-less Kernel?
  - processors are quite good about saving power when idle
    - *HW has support to reduce leakage on idle states*
  - avoid processor wakeups when nothing is happening
    - *by turning off the period timer tick when in idle state*
    - *looking at the timer queue to see when the next timer expires*
      - CPUIdle can exploit this information
      - in the absence of other events (e.g. hardware interrupts), the system will sleep until the nearest timer is due
    - *enable the periodic tick once the processor goes out of the idle state*
- *Proposed solution: merged in 2.6.21*
    - *room for further development (i.e. full tickless systems)*
      - *time slice is controlled by the scheduler, variable frequency profiling, and a complete removal of jiffies in the future*

The Linux Time System [8]

The Dynamic Tick Support

- Nomadik status

  support for disabling the timer tick during idle was first merged in 2.6.6 (2004, arch-s390)

  we are using an old interface, not the new one provided by clock source devices, but...

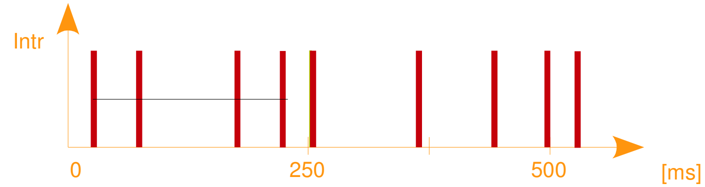
- Russell King (2008-05-11, 21:55:59 GMT):

  [ARM] Remove obsolete and unused ARM dyntick support

  dyntick is superseded by the clocksource/clockevent infrastructure, using the NO_HZ configuration option. No one implements dyntick on ARM anymore, so it's pointless keeping it around. Remove dyntick support.

Patrick Bellasi bellasi@elet.polimi.it

- Why deferrable timers?
  - better exploit dynamic-tick, by allowing to sleep longer
    - *not all timers has to run as soon as the requested period has expired*
    - *non-critical timeouts can run some fraction of a second later*
      - i.e. when the processor wakes up for other reasons

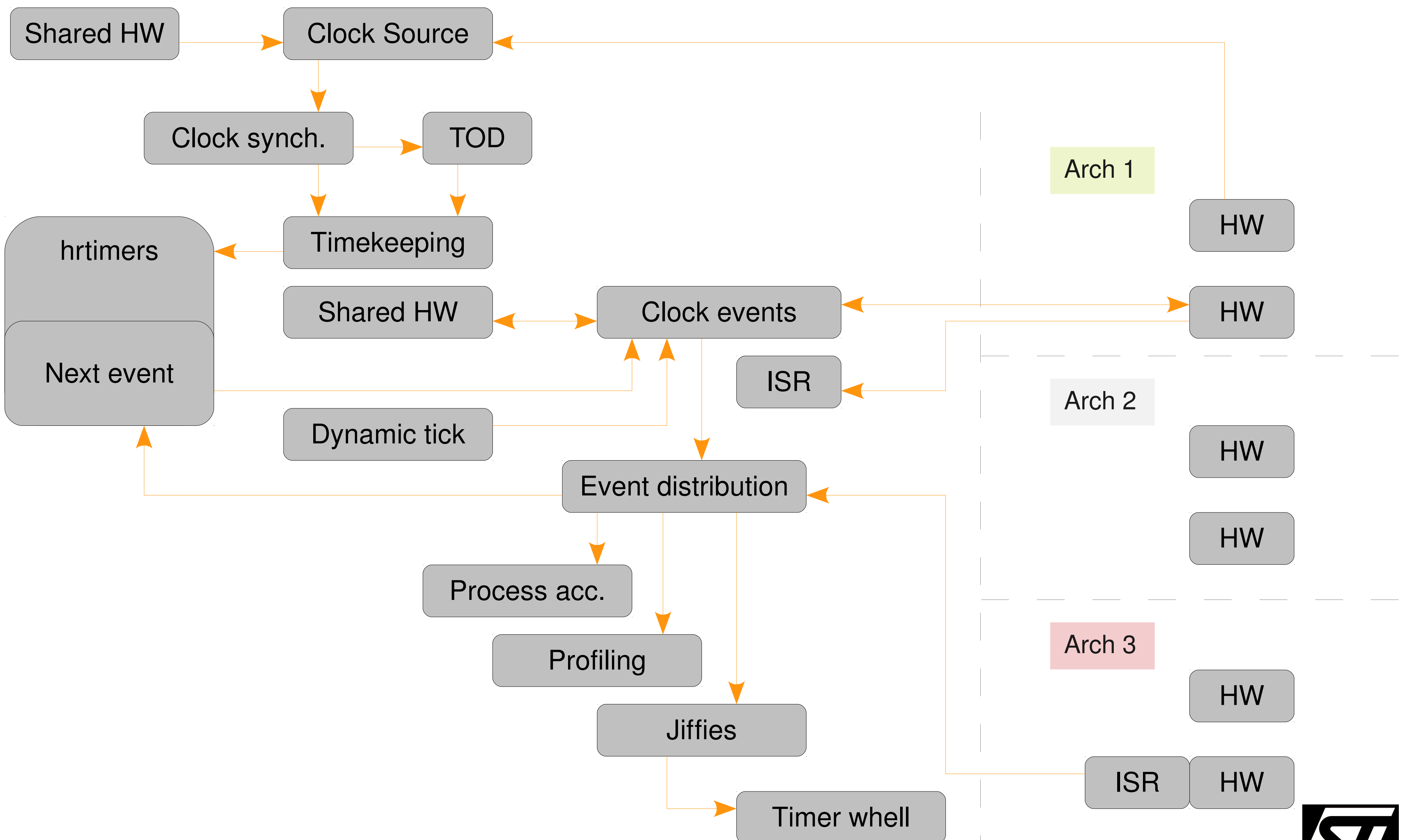


- Proposed solution: new function added to the internal kernel API
  - *init_timer_deferrable(struct timer_list *timer)*
  - *timers defined as deferrable are recognized by the kernel*
  - *will not be considered when the kernel makes its "when should the next timer interrupt be?" decision*
  - *timer_list, workqueue,*

Patrick Bellasi bellasi@elet.polimi.it

Linux Timekeeping Subsystem

Complete support for RT and PM



The Clock Framework

centralized control for all clock related functionalities

- Define an (abstract) API for all clock related functionalities
  - *dependencies (clock tree), rates (get/set), status (enable/disable)*
- Since 2.6.16
  - standard open source solution
    - *Initially added for ARM, now multiple architectures use it*
      - In 2.6.25: ARM (OMAP, PXA, SA1100, AT91, ...), Avr32, PowerPC, ...
    - *recent proposals on LKML for a generic clock API implementation*
- Used by device drivers
  - reference counting
  - re-entrant code
- May support layered structure
  - platform generic layer and board specific low-level

Patrick Bellasi bellasi@elet.polimi.it

Power/Voltage Framework

track device's power dependencies

- provide a framework analogues to the clock framework but for power/voltage regulators control
  - voltage regulators dependency tracking
  - satisfy client devices needs depending on their state
    - *optimize generators usage and efficiency*
- no support in mainline, but different patch-set available
  - ARM patch by Nokia (no too much general)
  - Platform independent framework by Wolfson Microelectronics
- Nomadik status (2.6.24)
    - *n*k15: limited platform power domains*
    - *n*k30: much more opportunities*

Patrick Bellasi bellasi@elet.polimi.it

Power/Voltage Framework

Wolfson Microelectronics - *Linux Voltage and Current Regulator Framework* [2]

- Announced on LKML in Feb 2008, presented on ELC 2008

  Standard and generic kernel interfaces to dynamically control voltage regulators and current sinks

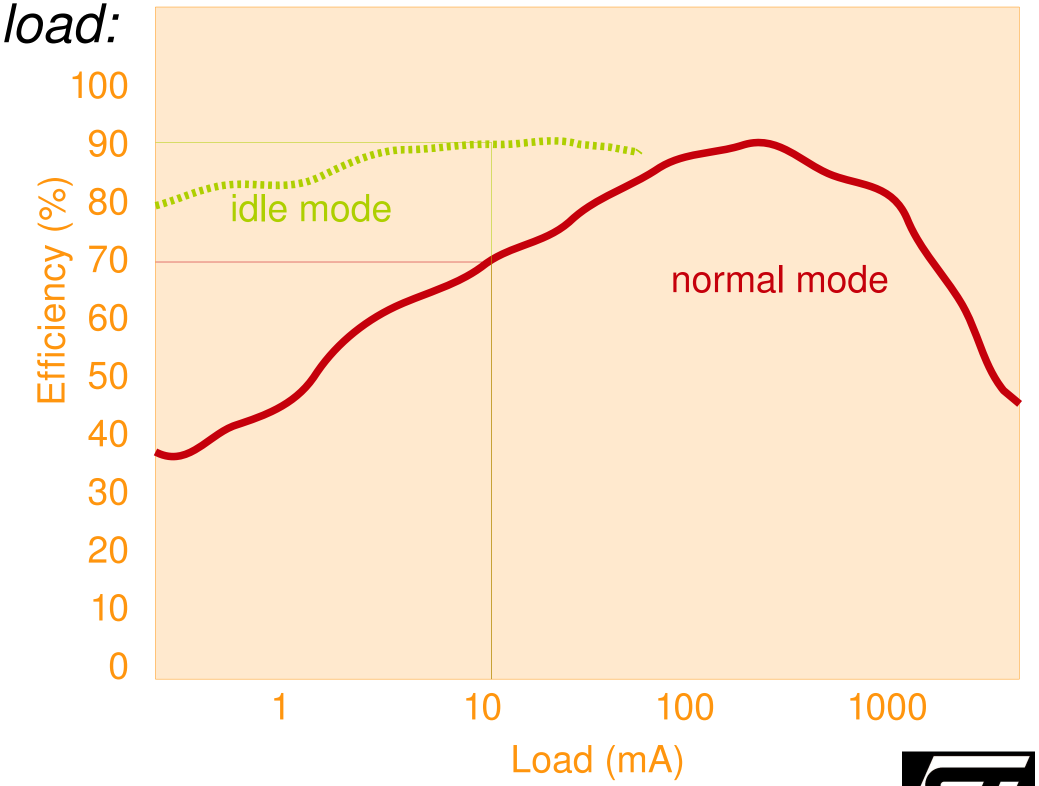
- Better exploit regulators dynamics

  e.g. consumer with 10mA load:

  70% @ Normal =~ 13mA

  90% @ Sleep =~ 11mA

  Saving ~2mA



Patrick Bellasi bellasi@elet.polimi.it

- Some real world examples:
  - *CPUFreq:* voltage control matching operating frequency
  - *CPUIdle:* voltage control matching idle state
  - *LCD back lighting:* control via white LED current reduction
  - *Audio:* analog supply control, components control
    - FM-Tuner, Speaker Amplifier when using Headphone
  - *NAND & NOR:* idle power control
    - *e.g. R/W=35mA, Erase=40mA, Erase+R/W=55mA, Idle=1mA*

Patrick Bellasi *bellasi@elet.polimi.it*

Power/Voltage Framework

Wolfson Microelectronics - *Linux Voltage and Current Regulator Framework* [2]

- Four separate interfaces
  - *Regulator driver API*
    - Allows regulator drivers to register and provide operations to the core
    - Notifier call chain for propagating regulator events to clients
  - *Consumer (Client) driver API*
    - complete control over their supply voltage and current
      - - static clients: only enable/disable support
      - - dynamic clients: voltage/current limit control
    - notify client V/I requirements to regulator driver
  - *Platform API*
    - creation of voltage/current domains (with constraints) for each regulator
    - creation of a regulator tree
  - *Userspace API*
    - exports a lot of useful voltage/current/opmode data to userspace via sysfs
    - help monitor device power consumption and status

The Latency Framework

Explicit system-wide latency-expectation infrastructure

- *Latency is the elapsing time between service request and service delivery*
- *Could influence application behaviors*
  - *e.g. audiocodec DMA buffer refill*
- *Power saving actions could influence overall system latency*
  - *we should consider their impact on overall system latency*
- *Need for a system framework that trace instantaneous latency allowed*

Patrick Bellasi bellasi@elet.polimi.it

The Latency Framework

Explicit system-wide latency-expectation infrastructure

- include/linux/latency.h

The system can tune its operations to the minimum latency requirements in effect at the moment

multiple drivers can announce their maximum accepted latency

drivers know device operating modes at any given moment and the corresponding expected system response time

collect and summarize these expectations globally

cumulated result can be used by power management and similar users

to make decisions that have trade-offs

Patrick Bellasi bellasi@elet.polimi.it

The Latency Framework

Explicit system-wide latency-expectation infrastructure

- Since 2.6.19, an interface where drivers can:
  - *announce the maximum latency [us] that they can deal with*
  - *modify this latency*
  - *give up their constraint*
  - *a function where the code that decides on power saving strategy can query the current global desired maximum*
  - *a notifier chain allow interested subsystems know when the maximum acceptable latency has changed*

  Currently used by the generic cpuidle driver on x86 arch
- Nomadik status (2.6.24)

  not exploited: because cpuidle is not correctly used?!

  could be interesting for some streaming devices

  Audio, Camera, WiFi, Bluetooth, CLCD, ...

Patrick Bellasi bellasi@elet.polimi.it

The Latency Framework

Explicit system-wide latency-expectation infrastructure

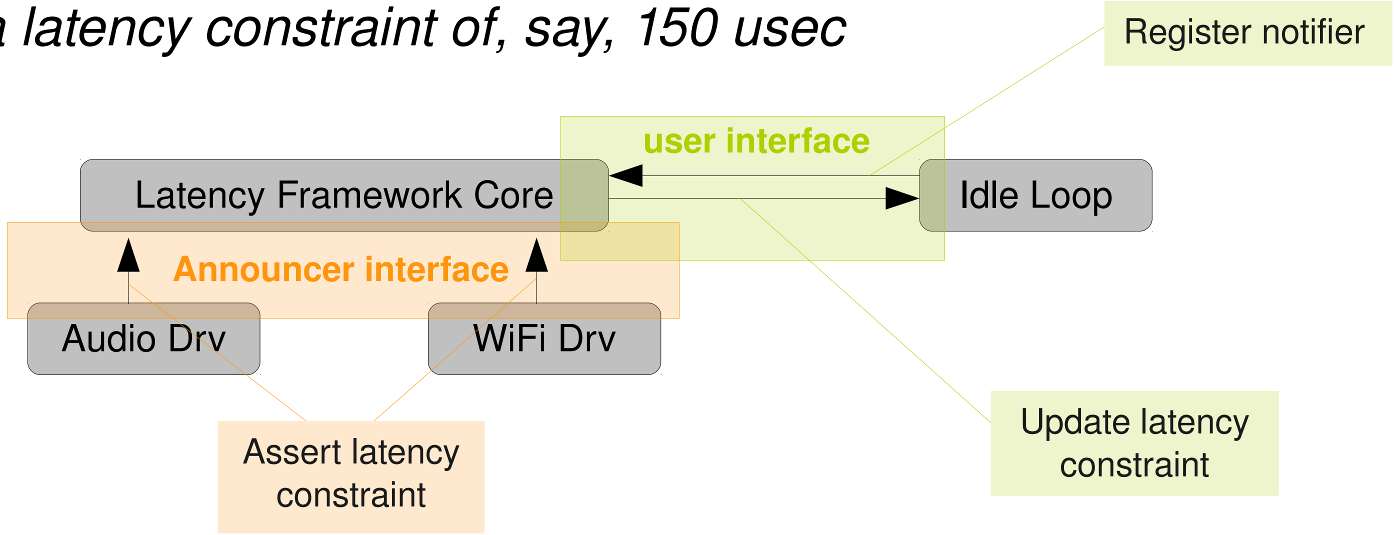
- Example:
  - *user:* idle loop code

    higher C-state saves more power, but has a higher exit latency

    idle loop code use these informations to make a good decision which C-state to use
  - *announcer:* audio driver

    knowns it will get an interrupt when the hardware has 200 usec of samples left in the DMA buffer

    set a latency constraint of, say, 150 usec



Patrick Bellasi bellasi@elet.polimi.it

LatencyTOP

measuring and fixing Linux latency

- tool for software developers (both kernel and userspace),
  - identifying where in the system latency is happening
  - what kind of operation/action is causing the latency to happen
  - both at system level or on a per process level
- focuses on the cases
  - the applications want to run and execute useful code, but there's some resource that's not currently available (and the kernel then blocks the process)
- Kernel patch needed
  - keep track of what *high level* operation it is doing
  - limited number of annotations
  - output on procfs or using a ncurses based GUI

Patrick Bellasi bellasi@elet.polimi.it

LatencyTOP

measuring and fixing Linux latency

Here's some output of LatencyTOP, collected for a make -j4 of a kernel on a quad core system

Cause	Maximum [msec]	Average [msec]
process fork	1097.7	2.5
Reading from file	1097.0	0.1
updating atime	850.4	60.1
Locking buffer head	433.1	94.3
Writing to file	381.8	0.6
Synchronous bufferhead read	318.5	16.3
Waiting for buffer IO	298.8	7.8

Cause	Maximum [msec]	Average [msec]
Writing to file	814.9	0.8
Reading from file	441.1	0.1
Waiting for buffer IO	419.0	3.4
Locking buffer head	360.5	75.7
Unlinking file	292.7	5.9
EXT3 (jbd) do_get_write_access	274.0	36.0
Filename lookup	260.0	0.5

Patrick Bellasi *bellasi@elet.polimi.it*

Quality of Service Framework [4]

the latency framework reloaded

- linux/pm_qos_params.h

  Kernel infrastructure to implement a coordination mechanism to facilitate communication among devices (drivers), users (applications) and system (power manager)

  devices have specific *power management capabilities*
    - *devices talk in terms of latencies, time outs, throughput*
    - *drivers could better address power management*
    - *expose power management mechanisms*

  applications have specific *performances needs*
- Since 2.6.25

  work-on-progress, used for iwl4965 WiFi driver on x86

Patrick Bellasi *bellasi@elet.polimi.it*

- Provide infrastructure for the registration of:
  - Dependents: register their QoS needs
    - e.g.
  - Watchers: keep track of the current QoS needs of the system
    - e.g. cpuidle, wifi drivers, ...
- 3 basic classes of QoS parameter
  - latency [us], timeout [us], throughput [kbs]
    - platform customizable constraints set
      - Interrupt latency, power domain latency, frequency/OPP
  - maintain lists of pm_qos requests and aggregated requirements
  - kernel notification tree for each parameter
- User mode interface for request QoS
  - using simple char device node

The CPUIdle Framework [1]

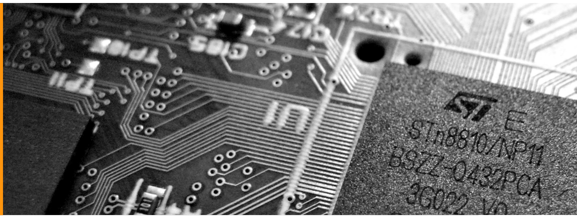
Do nothing, efficiently...

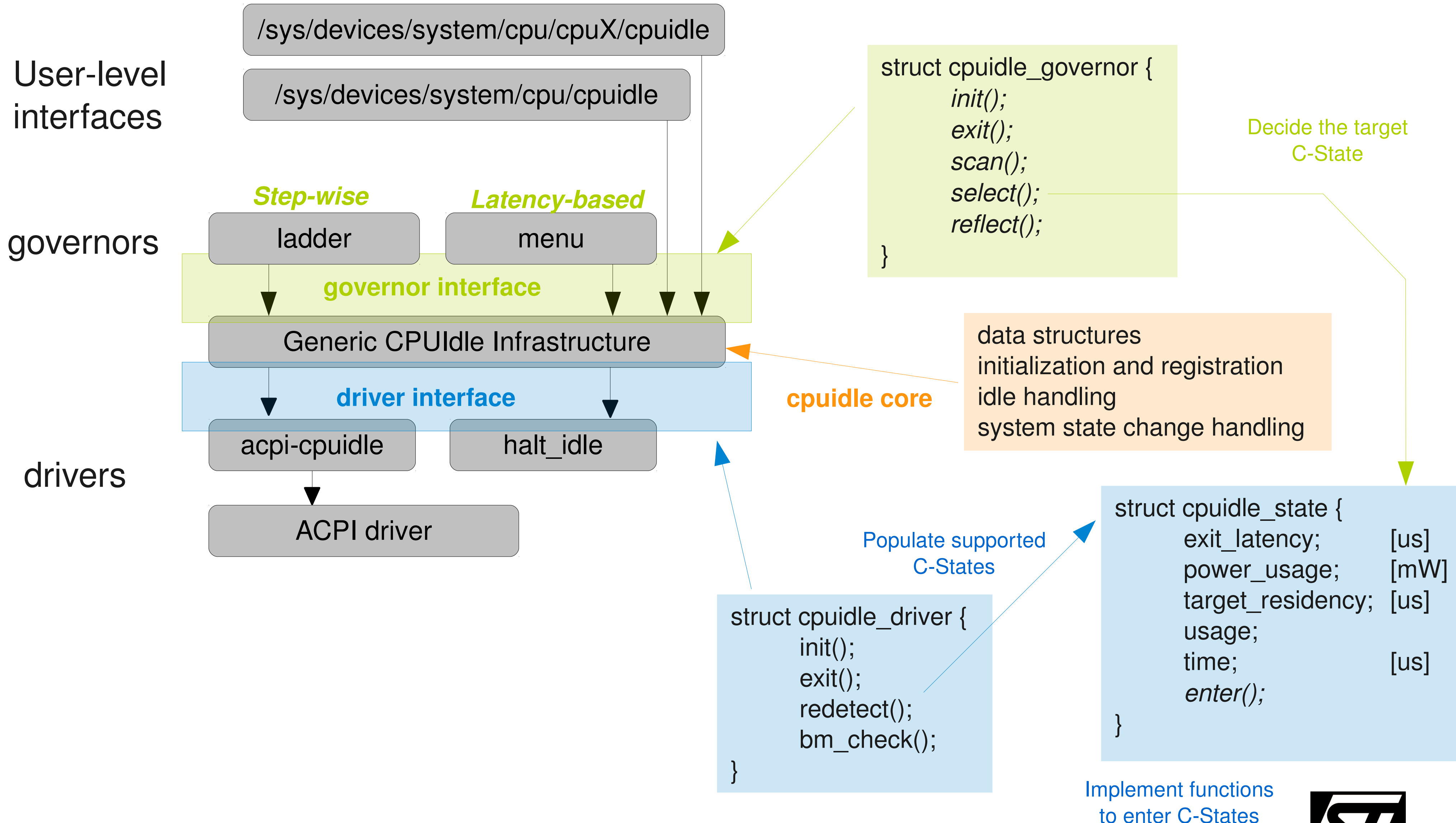
- Generic processor *idle power management* framework
  - *support for multiple idle states with different characteristics*
    - power consumptions, state preservation, state constraints, entry/exit latency
- Support trade-off efficiently between application expectations and idle state power saving
- Clean interface
  - *abstract between idle-driver and idle-governor*
    - separate arch-specific idle driver (mechanisms) from arch-independent power management governors (policies)
  - *provide a convenient user-space interface*
- Since 2.6.24
  - *X86 - ACPI, for OMAP mapping to ACPI states*

Patrick Bellasi bellasi@elet.polimi.it

The CPUIdle Framework [1]

Do nothing, efficiently...





Patrick Bellasi bellasi@elet.polimi.it

The CPUIdle Framework [1]

The OMAP43xx implementation

- The following C states are supported in Cpuidle driver:

```
C0   - System executing code          0        0
C1   - MPU WFI + Core active         20       30
C2   - MPU CSWR + Core active       100      300
C3   - MPU OFF + Core active       3300     4000
C4   - MPU CSWR + Core CSWR       10000    12000
C5   - MPU OFF + Core CSWR        11500    15000
C6   - MPU OFF + Core OFF         20000   300000
```

- Menu governor takes the following into account to decide the target sleep state:
  - Next timer expiry in the system
  - Comparing target residency with available sleep time
  - Comparing exit latency with system wide latency constraints
  - Checking for activity in the core domain
- Dynamic tick based on support in kernel

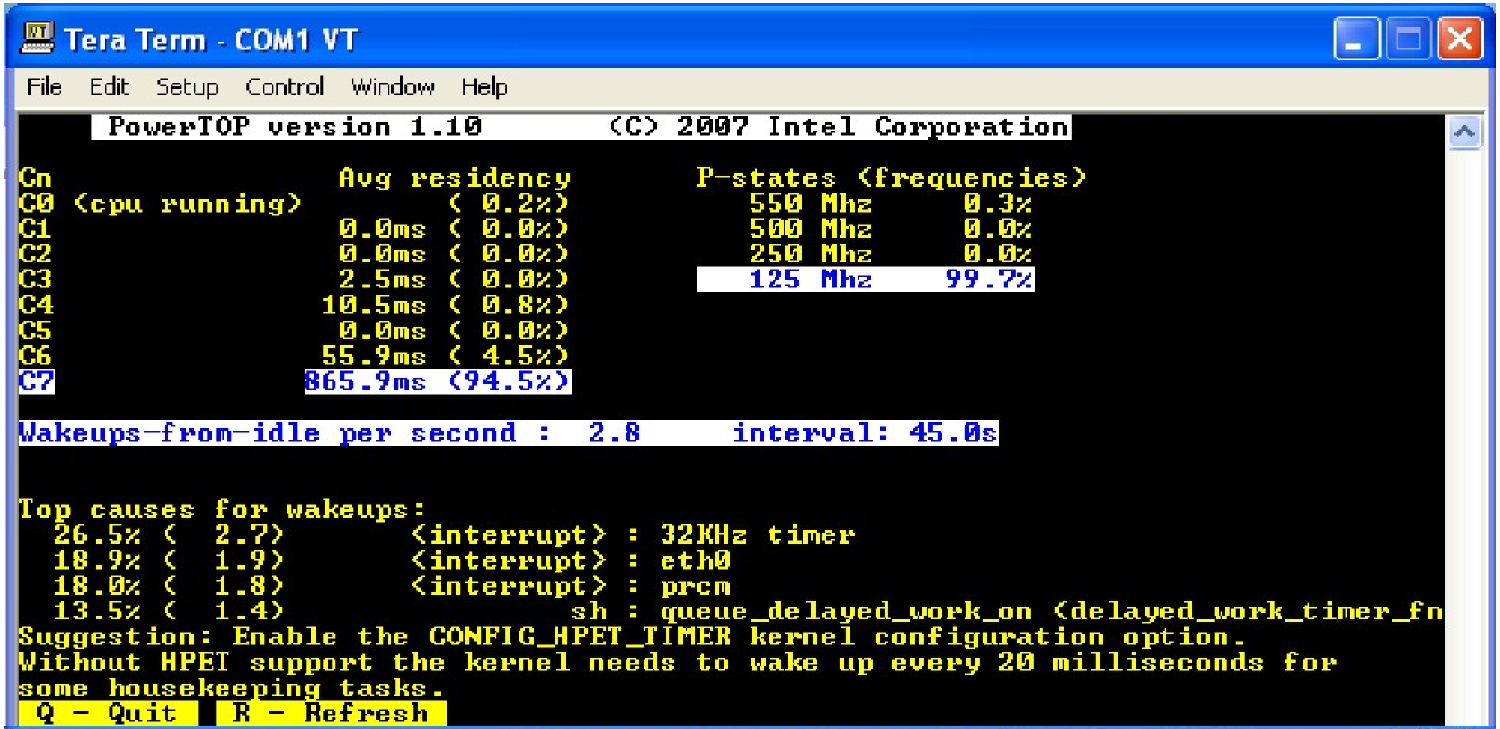
Patrick Bellasi bellasi@elet.polimi.it

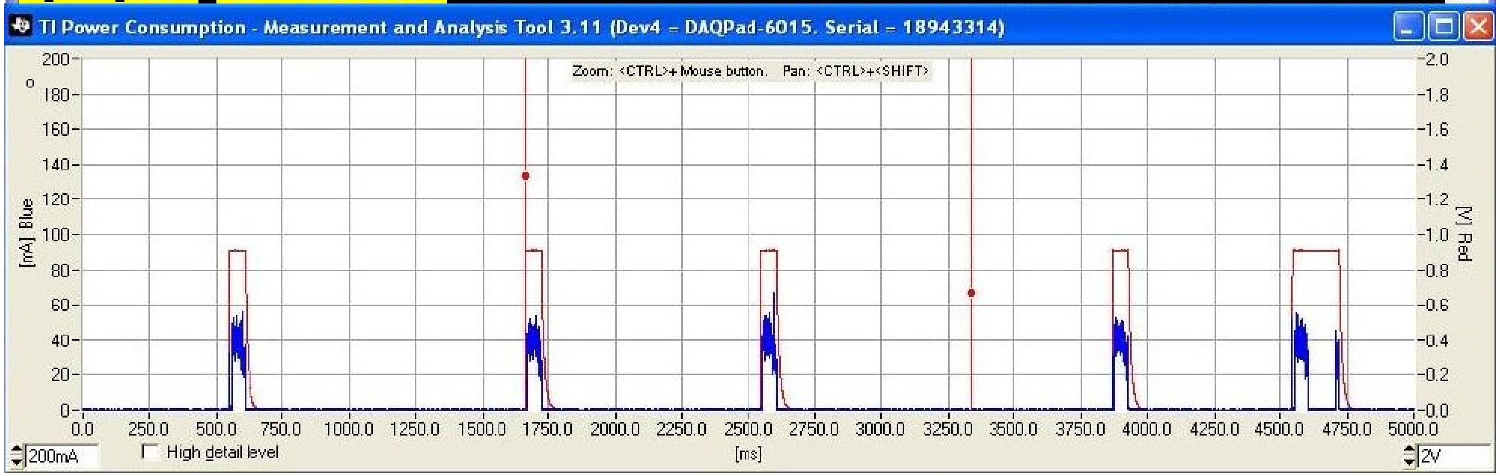
- identifying what is causing system wake-ups
  - *what kind of operation/action prevent long-time permanence in low-power consumption states?*
  - *support software developers, both kernel-space and user-space*
- main features
  - *show how well your system is using various HW power-saving features*
    - both C- and P-States statistics are collected
  - *show you the culprit software components that are preventing optimal usage of your HW power savings*
    - userspace should prefere deferrable timers
  - *provide you with tuning suggestions to achieve low power consumption*
- support for CPUIdle since v1.10
  - *previsou version used the ACPI interface*
  - *a patch make it possible to collect statistics on both C- and P-States*

Patrick Bellasi bellasi@elet.polimi.it

PowerTOP

example session running on OMAP3430



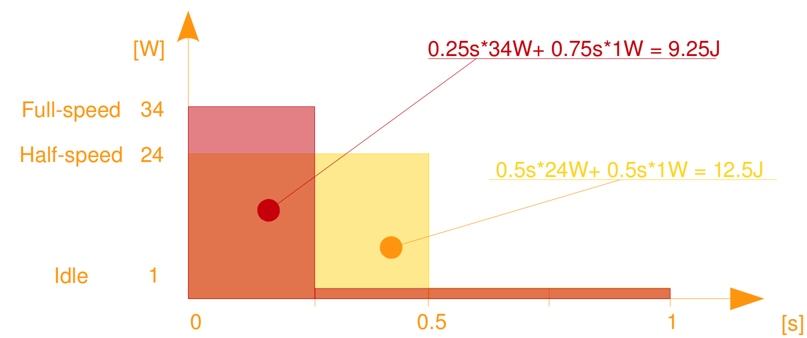


Patrick Bellasi bellasi@elet.polimi.it

The CPUFreq Framework [9]

CPU active power consumption optimization

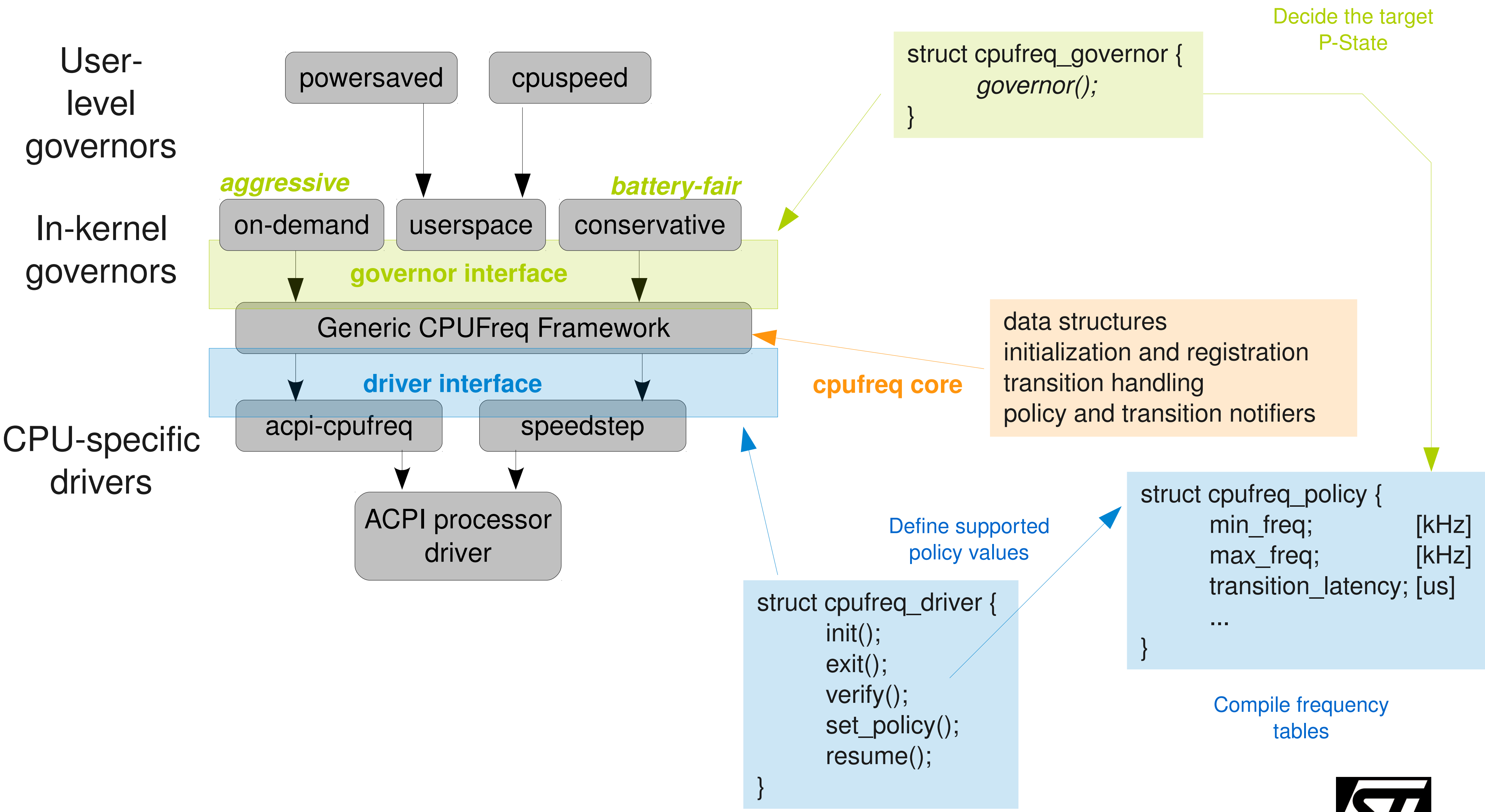
- Generic processor *active power management* framework
  - *support for multiple performance states with different characteristics*
    - power consumptions
- Clean interface
  - *abstract between driver and governor*
    - separate arch-specific cpu driver (mechanisms) from arch-independent power management governors (policies)
  - *provide a convenient user-space interface*
- Since 2.6.9
  - *on-demand governor*
    - scaling based on load
    - exploit race-to-idle
  - *more recently added support for conservative governor*
    - implementing a better battery-fair policy



Patrick Bellasi *bellasi@elet.polimi.it*

CPUFreq

optimized CPU power consumptions



Patrick Bellasi bellasi@elet.polimi.it

Linux PM

device runtime suspend/resume support

- Every driver should implement suspend/resume calls
  - register to the LDM (Linux Device Model)
  - release clocks and save context in suspend call and restore these when resume is called
  - drivers which have already released their clocks and have saved their context need not do anything in their suspend call
- Drivers should support OFF modes
  - all registers in the power domain are reset when the power domain goes to OFF
  - OFF mode could introduce considerable latency for wakeup
  - the system can enter chip off through two paths:
    - *Idle loop*
    - *Suspend/Resume*

Patrick Bellasi *bellasi@elet.polimi.it*

- Device drivers responsibilities
  - aggressively manage request/release of clocks
    - *through clock framework => control their clocks on a request basis*
    - *not transaction based => use inactivity timer to cut off clocks after a period of inactivity*
  - need to register with the LDM
    - *implement suspend() and resume() calls*
  - specify constraints when required for its functionalities
    - *e.g. min. frequency, max latency, ...*
  - need to implement context save/restore
    - *be aware of power OFF mode*
  - configure optimal power settings
    - *according to standing system constraints*

Patrick Bellasi bellasi@elet.polimi.it

Linux PM

Examples of changes being made

- Transaction based drivers will control clocks based on activity
  - e.g. i2c driver enables clocks when there are pending requests and disables them when there are no pending requests
- Camera – Clocks will be enabled as long as the driver is required
- Display – The fbdev inactivity timer (which is tied to user activity) can be used to turn off display clock
- MMC – Clocks are controlled on a per command basis
- GPTimer – Clocks are controlled as per requirement
  - i.e. when a timer is in use, they will be enabled and will be disabled when they are not in use
- UART – Console clocks are cut in the idle loop (before putting core domain to retention) and other UART clocks could be controlled on a need basis
- USB – Clocks can be controlled as per requirement (only when transfers are going on)

Patrick Bellasi bellasi@elet.polimi.it

- When all drivers in a power domain release their clocks, the power domain can go to RET or OFF state

  the shared resource framework programs the power domain to target state depending on the latency constraints in the system

- Drivers can follow any of the following methods to save and restore their context
  - *Always save/ always restore*

    drivers which do not have lots of registers can always save context and restore context because it will not cause a lot of overhead
  - *Early save/ restore on demand*

    drivers save context every time they release their clocks but restore it only if the power domain has actually gone to off after saving

    this makes sense for drivers which have a large restore time with save time being minimal

Patrick Bellasi bellasi@elet.polimi.it

References

1. V. Pallipadi, S. Li, A. Belay, cpuidle - Do nothing, efficiently... Proceedings of the Linux Symposium, Vol. 2, June 2007. Ottawa, Ontario Canada.
2. Walfson Microelectronics, Linux Voltage and Current Regulator Framework
3. An API for specifying latency constraints, LWN Article, August 2006.
4. M. Gross, Power Management QoS: How You Could Use it in Your Embedded Application, Intel, Open Source Technology Center.
5. J. Stults, N. Aravamuda, D. Hart, We Are Not Getting Any Younger: A New Approach to Time and Timers, Proceedings of the Linux Symposium, Vol. 1, July 2005. Ottawa, Ontario Canada.
6. T. Gleixner, D. Niehaus, Hrtimers and Beyond: Transforming the Linux Time Subsystems, Proceedings of the Linux Symposium, Vol. 1, July 2006. Ottawa, Ontario Canada.
7. Clockevents and dyntick, LWN Article, February 2007.
8. Deferrable timers, LWN Article, March 2007.
9. V. Pallipandi, A. Starikovskiy, The Ondemand Governor, Proceedings of the Linux Symposium, Vol. 2, July 2006. Ottawa, Ontario Canada.
10. R. Woodruff, Linux system power management on OMAP3430, Embedded Linux Conference, April 2008. Silicon Valley, US.
11. PowerTOP,