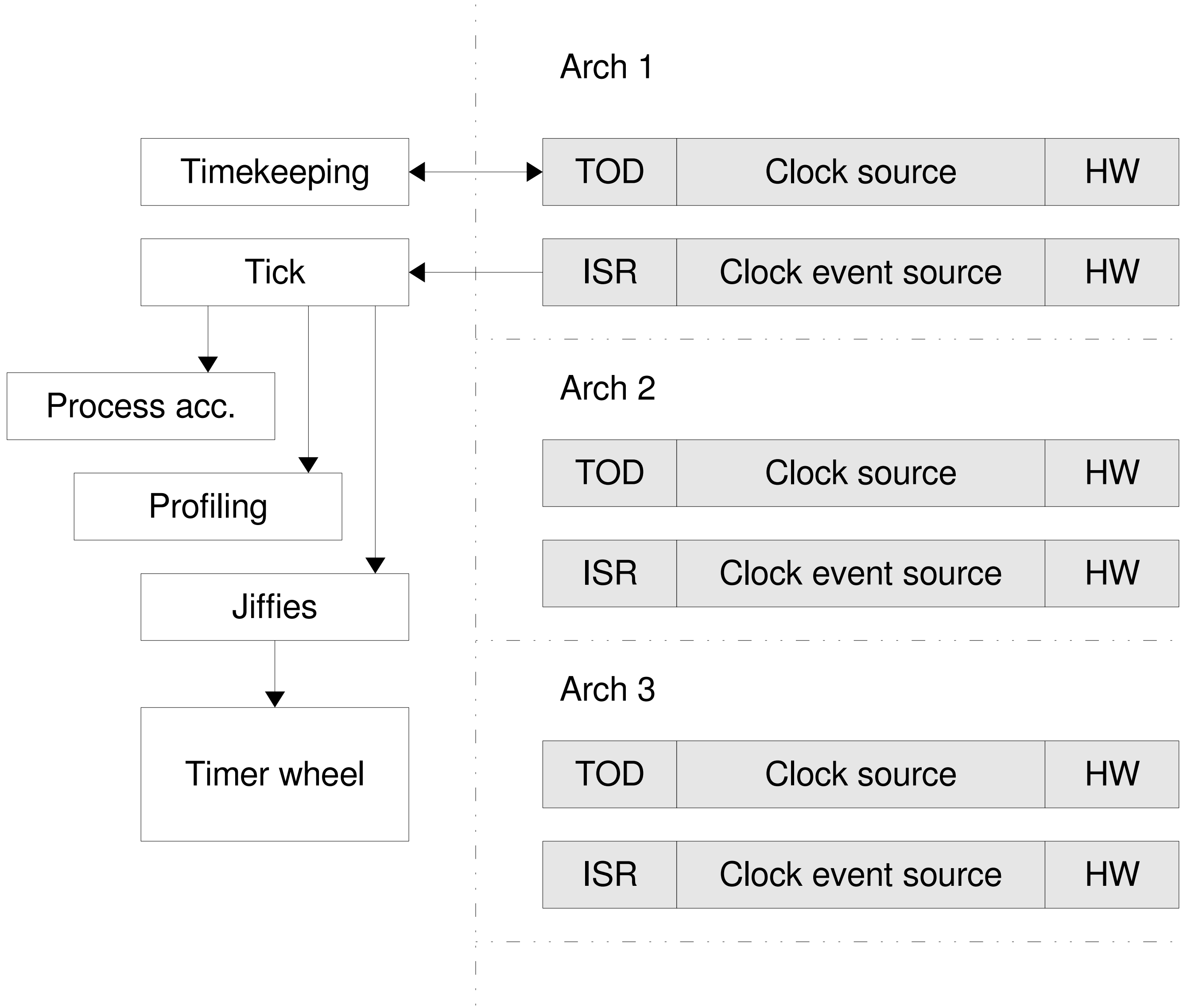
hrtimers and beyond
-
transformation of the Linux time(r) system

Thomas Gleixner
Douglas Niehaus

OLS 2006

Original time(r) system



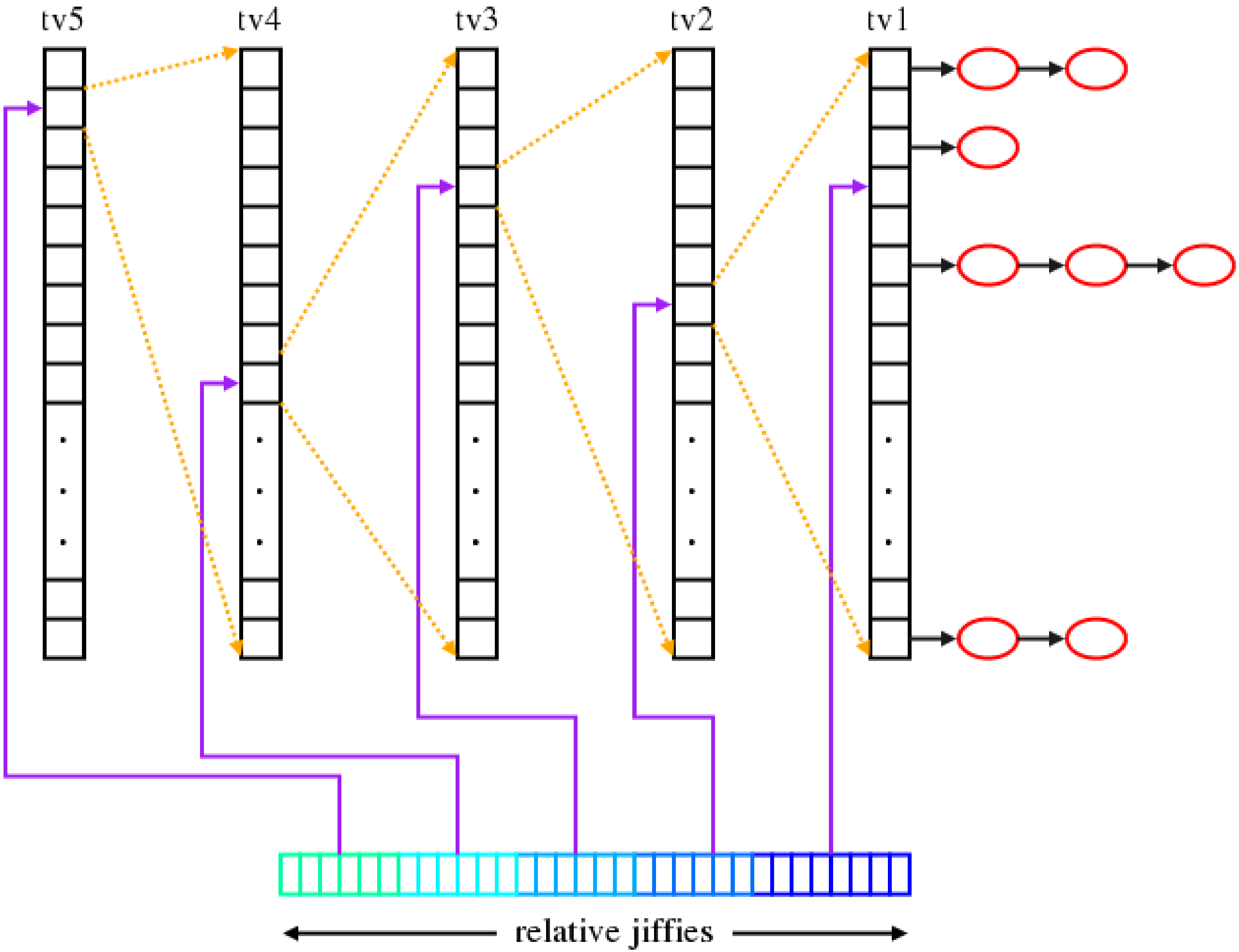
History

- double linked list sorted by expiry time
- UTIME (1996)
- timer wheel (1997)
- HRT (2001)
- hrtimers (2006)

Timer Wheel

- periodic tick necessary
- O(1) insertion / deletion
- recascading in bursts (can cause high latencies)
- higher tick frequencies don't scale due to long lasting timer callbacks and increased recascading

Cascading



Cascading

		100	250	1000	HZ
[1]	256	10	4	1	ms
[2]	64	2560	1024	256	ms
[3]	64	164	66	16	s
[4]	64	175	70	17	m
[5]	64	186	75	19	h

Cascading
CONFIG_BASE_SMALL=y

		100	250	1000	HZ
[1]	64	10	4	1	ms
[2]	16	640	256	64	ms
[3]	16	10240	4096	1024	ms
[4]	16	164	66	16	s
[5]	16	44	17	4	m

Cascading

- array sizes have to be chosen carefully taking tick frequency into account
- rare (multiple) cascades increase latency
  - use cases have to be analysed to avoid problematic cascading
- separating timers with high accuracy requirement from coarse grained timeouts will relax the situation

timers vs. timeouts

timers

- precise event scheduling
- accurate
- likely to expire

timeouts

- report error conditions
- coarser grained
- likely to be removed before expiration

History of high resolution timers

- UTIME – KURT-Linux
  - University of Kansas
- HRT – fork of UTIME
  - Monta Vista
- Hrtimers
  - Linutronix

Why hrtimers ?

- UTIME and HRT added a subjiffy field
  - Kept jiffy ticks by design to avoid broader kernel change impact
  - Modes: on top of the timer wheel or separate high-resolution event list
- HRT moved high resolution timers into a separate list one tick before expiry
  - Suffered from timer wheel latencies

hrtimers

- timers inserted into a red-black tree sorted by expiration time
- separate queue for each base clock, which allowed simplifying POSIX timers
- base code is still tick driven (softirq is called in the timer softirq context)
- time values are kept in new data type ktime_t (using nanosecond base)

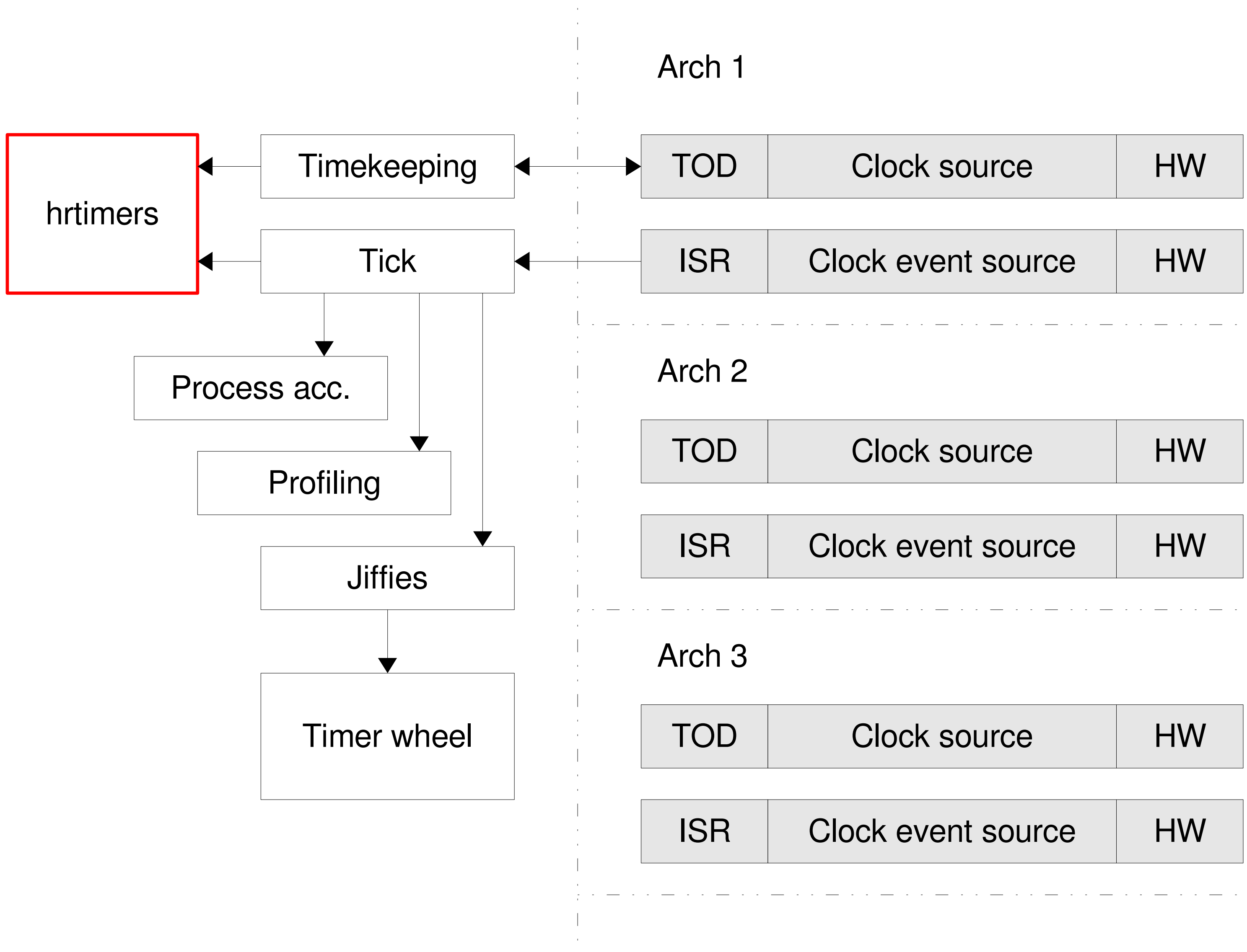
ktime_t

- optimizable data type for both 32 and 64 bit machines
- plain nanosecond value on 64 bit CPU
- (seconds, nanoseconds) pair on 32 bit CPUs with field order allowing (depending on the endianess) 64 bit add, subtract, compare operations.

hrtimer users

- nanosleep
- itimer
- POSIX timers
- timed futex operations

hrtimers



Arch 1
hrtimers
Timekeeping
TOD
Clock source
HW
Tick
ISR
Clock event source
HW
Process acc.
Profiling
Jiffies
Timer wheel
Arch 2
TOD
Clock source
HW
ISR
Clock event source
HW
Arch 3
TOD
Clock source
HW
ISR
Clock event source
HW

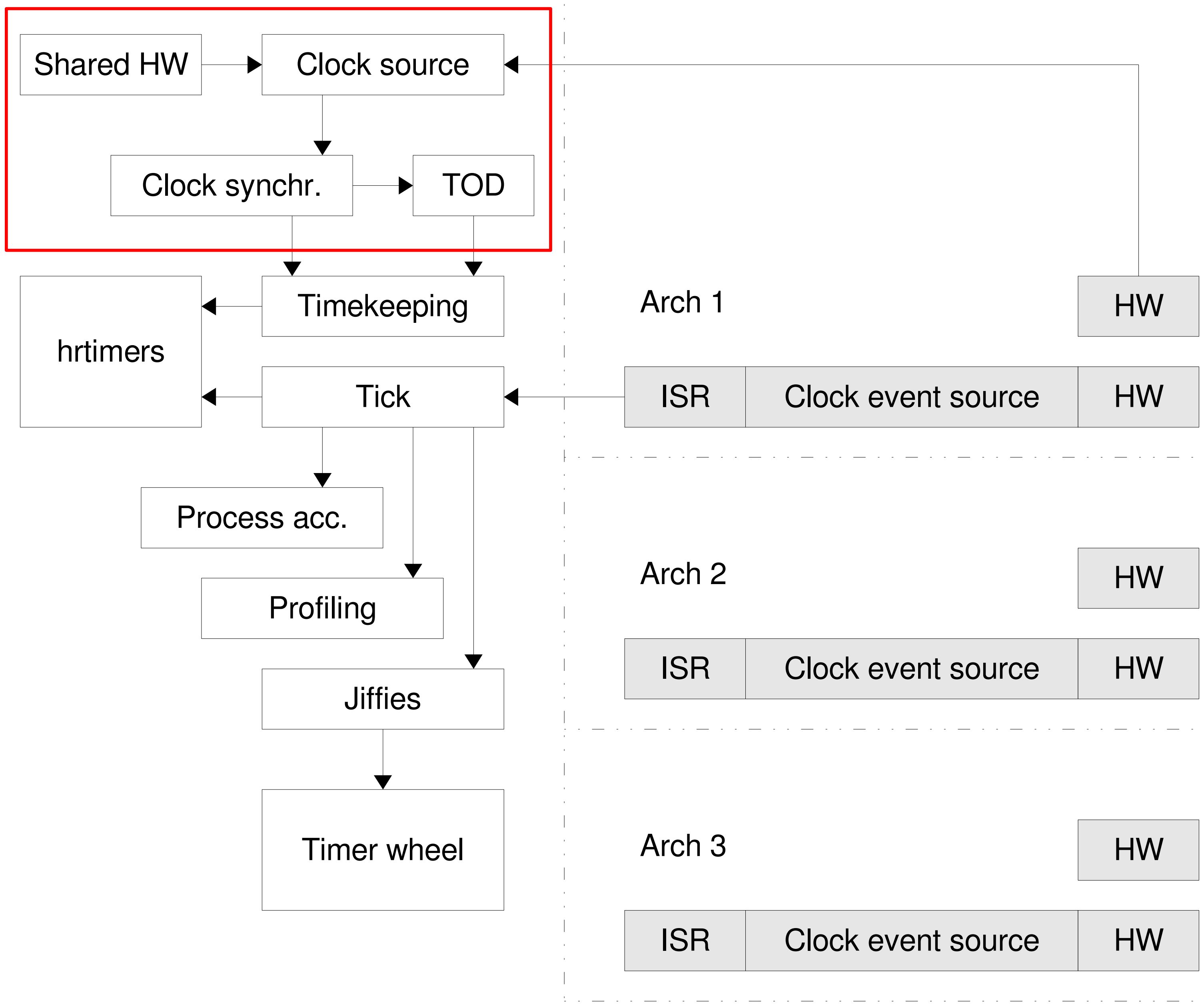
how to get high resolution timers ?

- solve the tick (jiffy) dependency of timekeeping
- create a generic framework for next event interrupt programming
- replace the periodic tick interrupt by timers under hrtimers

Timekeeping

- Make use of John Stultz's Generic Time of Day framework
  - architecture independent
  - generic framework replaces duplicated architecture code
  - better decoupling from tick

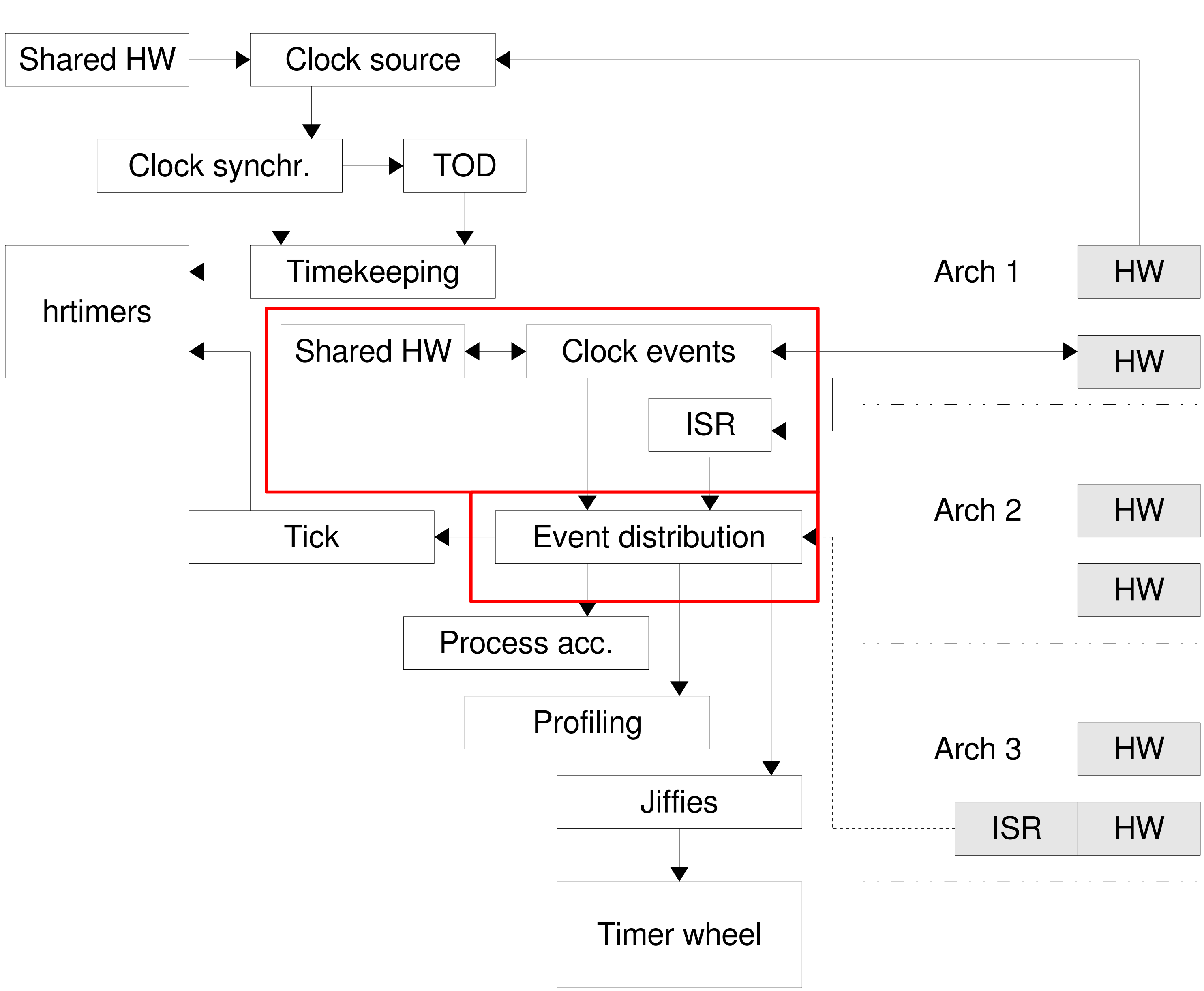
hrtimers + GTOD



clockevents

- Generic infrastructure to distribute timer related events
  - architecture independent
  - generic framework replaces duplicated architecture code
  - allows quality based selection of clock event hardware

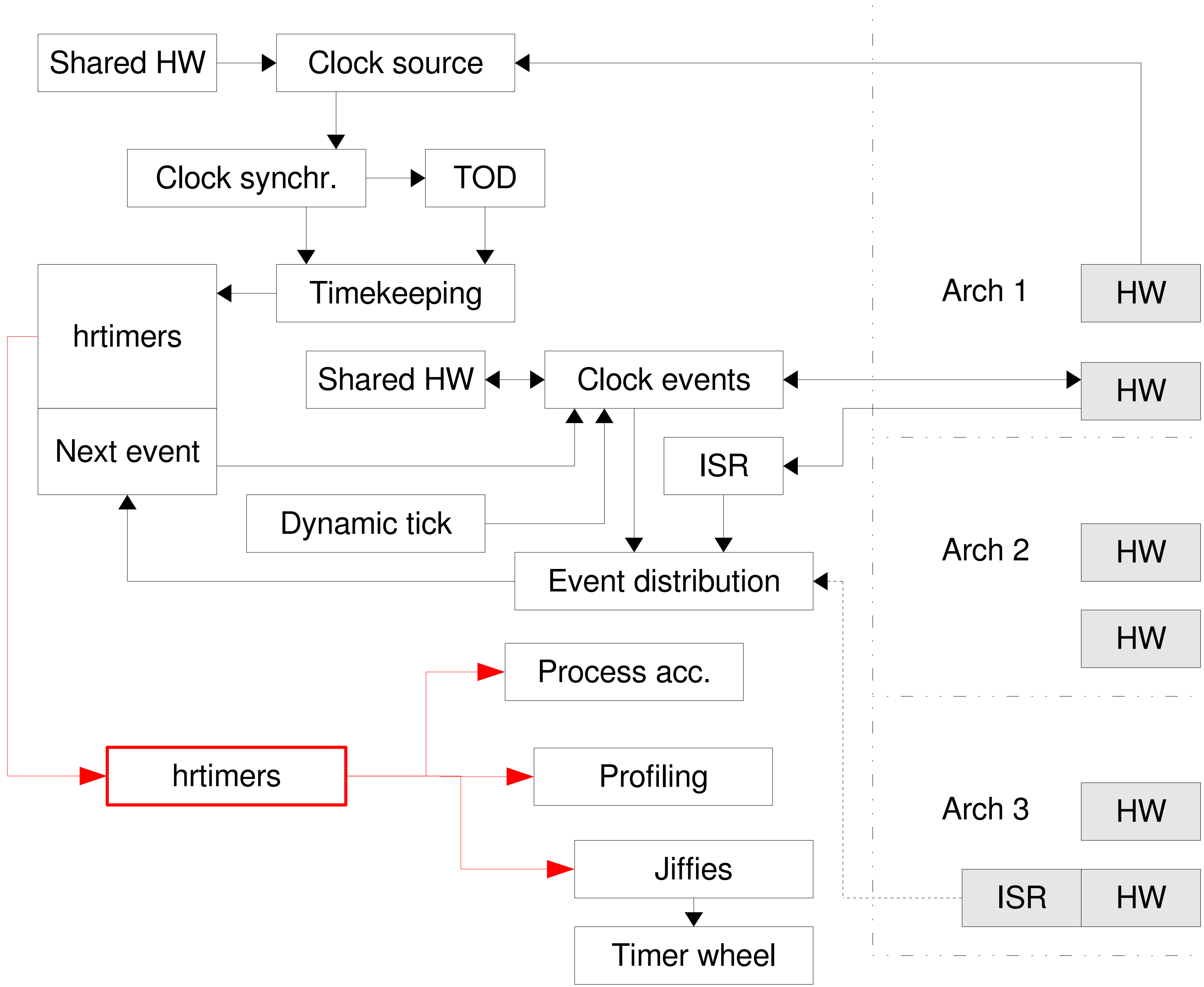
hrtimers + GTOD + clockevents



tick emulation

- Use a per-CPU hrtimer to emulate tick
  - update jiffies and NTP adjustments
  - per-CPU calls
    - process accounting and profiling
- Allows high resolution timers and/or dynamic ticks

hrtimers + GTOD + clockevents + tick emulation



high resolution performance

clock_nanosleep(ABS_TIME)
interval: 10ms
10000 loops
no load

Kernel	min	max	avg	
2.6.16	24	4042	1989	µs
2.6.16-hrt	12	94	20	µs
2.6.16-rt	6	40	10	µs

high resolution performance

clock_nanosleep(ABS_TIME)
interval 10ms
10000 loops
100% load

Kernel	min	max	avg	
2.6.16	55	4280	2198	µs
2.6.16-hrt	11	458	55	µs
2.6.16-rt	16	55	20	µs

dynamic tick idle behaviour

- timer interrupts reduced to ~1 per second.
  - instrumentation to identify the timer (ab)users to improve the idle sleep length

timer wheel batching

- run the timer wheel at a lower frequency than the scheduler tick by skipping timer wheel processing for a user space configurable number of ticks
- improves interactivity

things to be done

- get it merged (target is 2.6.19)
- support more architectures (prototypes for ARM and PPC available)
- tighter integration into power management

Conclusions

- significant changes are necessary but the benefit is significant increases in:
  - architecture independent code
  - ease of using wide range of time keeping and timer event hardware
  - increased resolution for scheduled events when desired