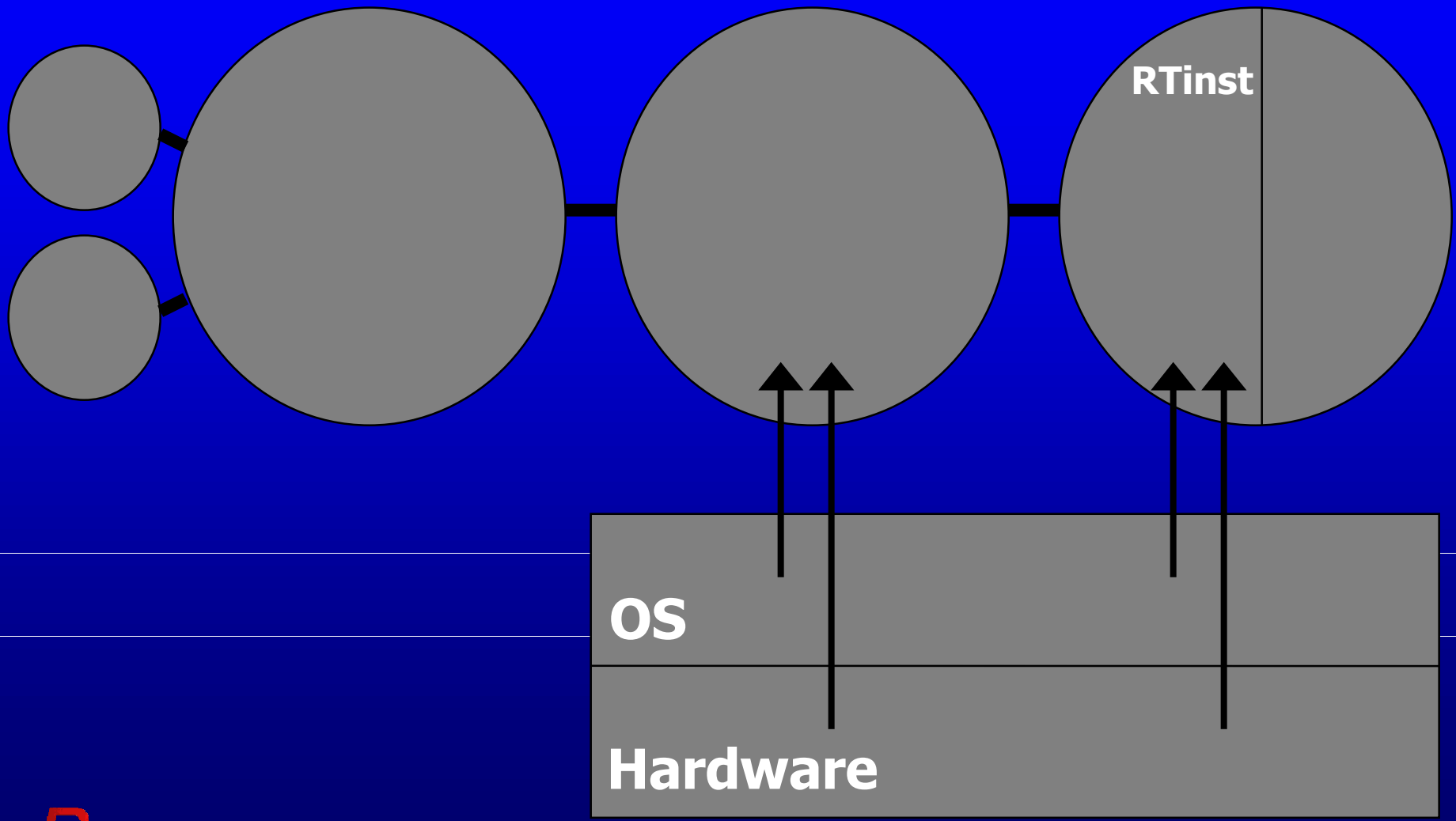
UNIVERSITY OF WISCONSIN MADISON

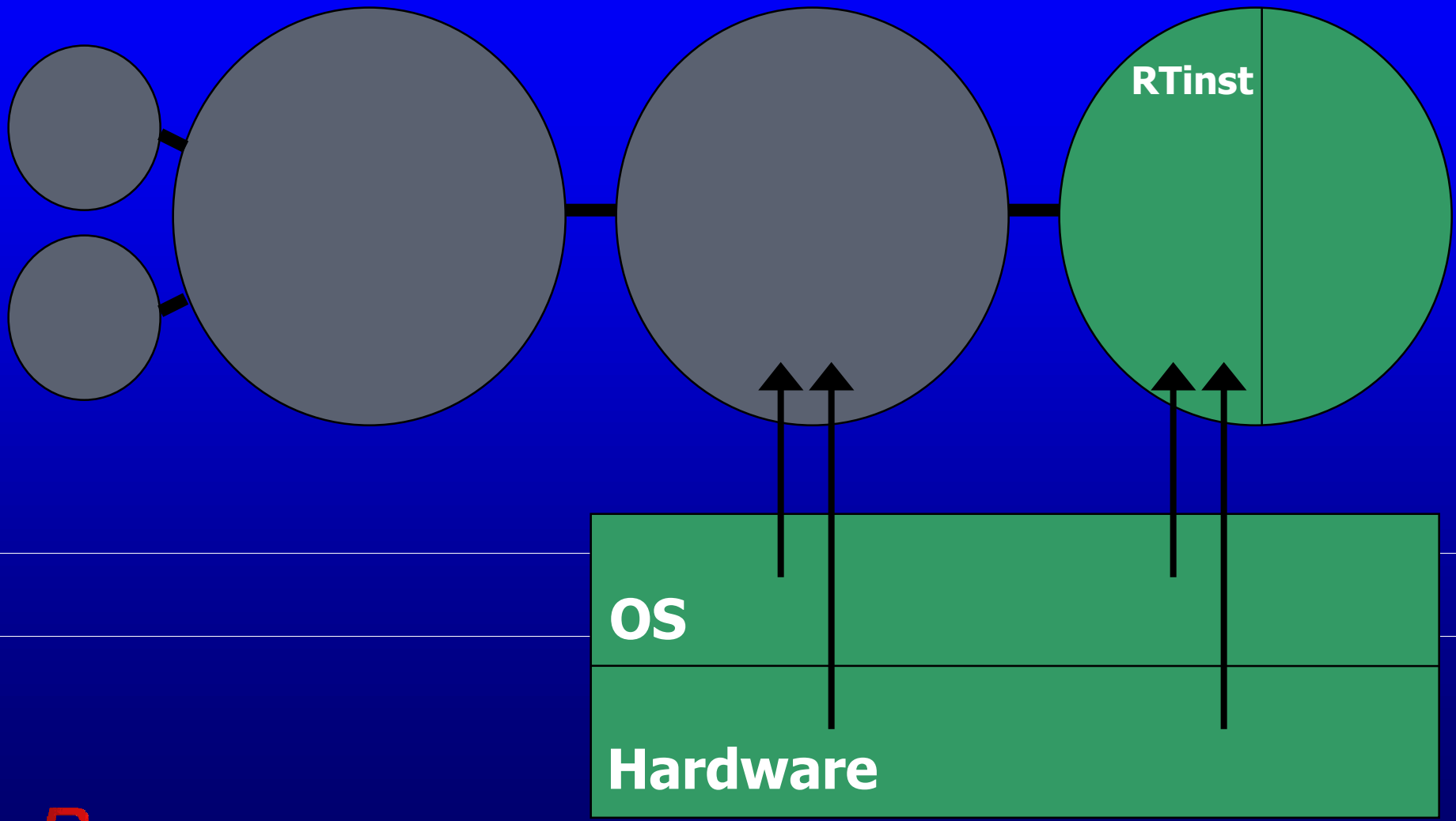
High-Resolution Timing Update

Nick Rasmussen
nick@cs.wisc.edu

Brandon Schendel
schendel@cs.wisc.edu

Computer Sciences Department
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706-1685
USA


Visis
Paradyn Front-end
Paradyn Daemon
Application
RTinst
OS
Hardware


Visis
Paradyn Front-end
Paradyn Daemon
Application
RTinst
OS
Hardware

Motivation

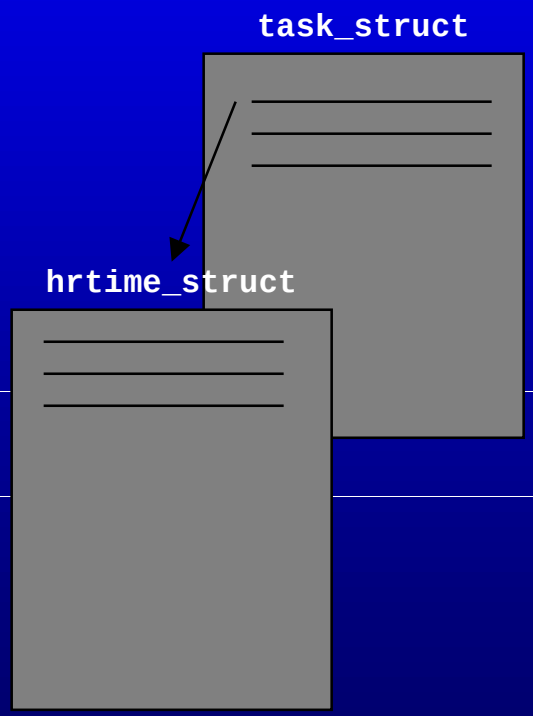
- We already have access to high-resolution elapsed time
  - TSC: 20 cycle query cost, ~2ns resolution
- Our only interface to get process time is through the *times* system call
  - *times*: 350 cycle query cost, 10ms resolution
- Goal: Provide access to high-resolution process time with low query cost

High-Resolution Timers

- Solaris *gethrtime*, *gethrvtime*
- Timers previously added to other OSs:
  - VAX/BSD, SunOS 2.x, CM5, Sequent Symmetry

Linux Kernel Modifications

- Add TSC-based process time bookkeeping to `task_struct`
- Update on context switch
- Update on kernel entry and exit
- Expose to user via *mmap*
  - /proc/PID/hrtime
- Allows cycle-accurate query of other processes



hrtime_struct

```
struct hrtime_struct {
        volatile hrtime_t last_us_dispatch;
        volatile hrtime_t utime;
        volatile hrtime_t stime;
        volatile long     in_system;
        long              has_ustime;
        hrtime_t          start_time;
        volatile hrtime_t last_dispatch;
        volatile hrtime_t vtime;
        volatile long     offset_to_cpu0;
        long              refcount;
        spinlock_t        reflock;
}
```

hrtime_struct

```
struct hrtime_struct {
        volatile hrtime_t last_us_dispatch;
        volatile hrtime_t utime;
        volatile hrtime_t stime;
        volatile long     in_system;
        long              has_ustime;
        hrtime_t          start_time;
        volatile hrtime_t last_dispatch;
        volatile hrtime_t vtime;
        volatile long     offset_to_cpu0;
        long              refcount;
        spinlock_t        reflock;
}
```

Self-Query Example

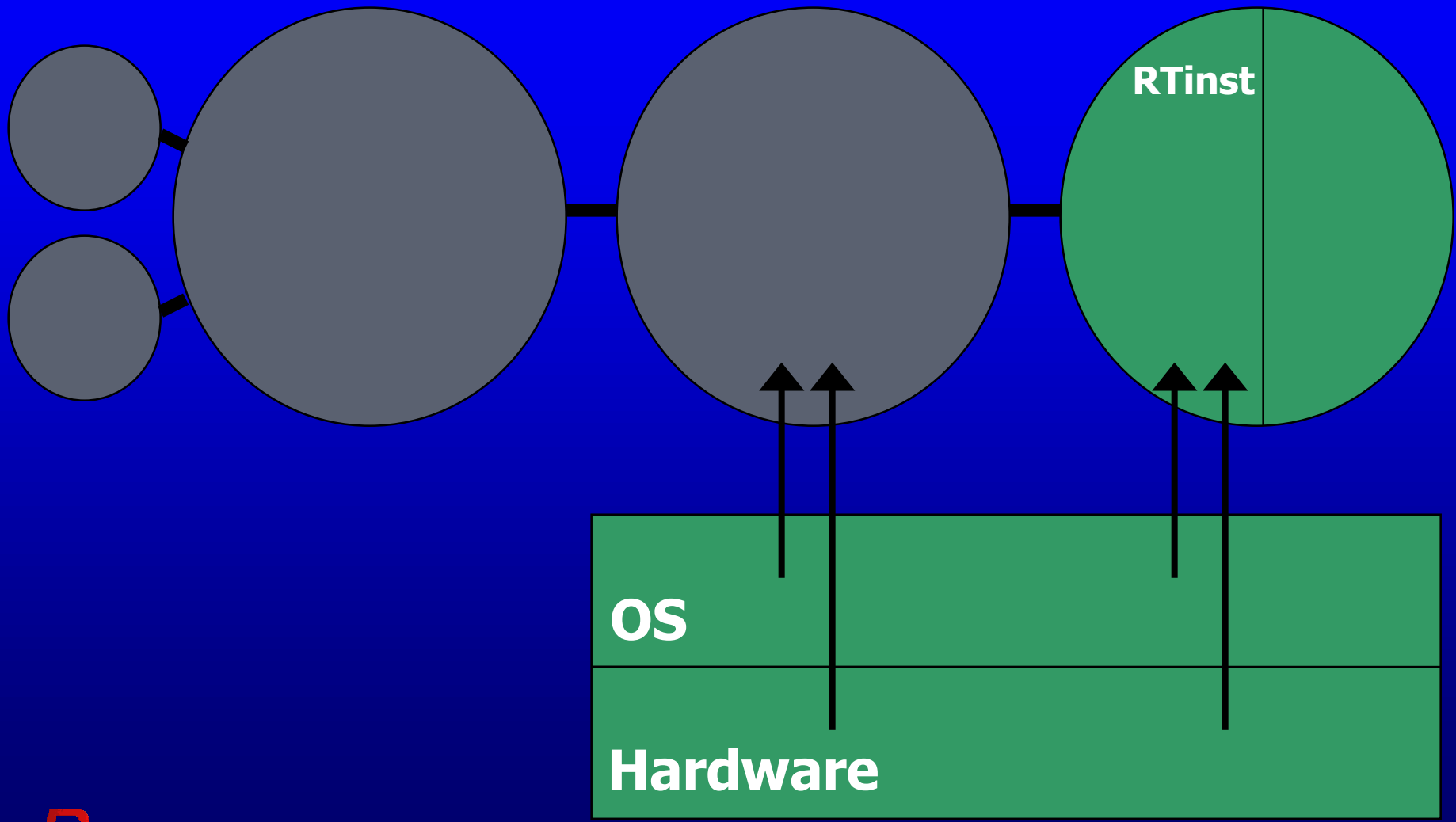
```
// hr points to the mmapped hrtime_struct
// for the current process
do {
  begin = rdtsc();
  vtime = begin – hr->last_dispatch
            + hr->vtime;
  end   = rdtsc();
} while (end - begin > THRESHOLD)
```

Subtle Issues

- TSC synchronization on SMP boxes
- Process state change during query
  - Context switch, kernel entry/exit
- TSC → nanosecond conversion
  - Drift from real time if conversion factor is not accurate

Libhrtime

- User library to access timers
- Timer query cost - ~120 cycles

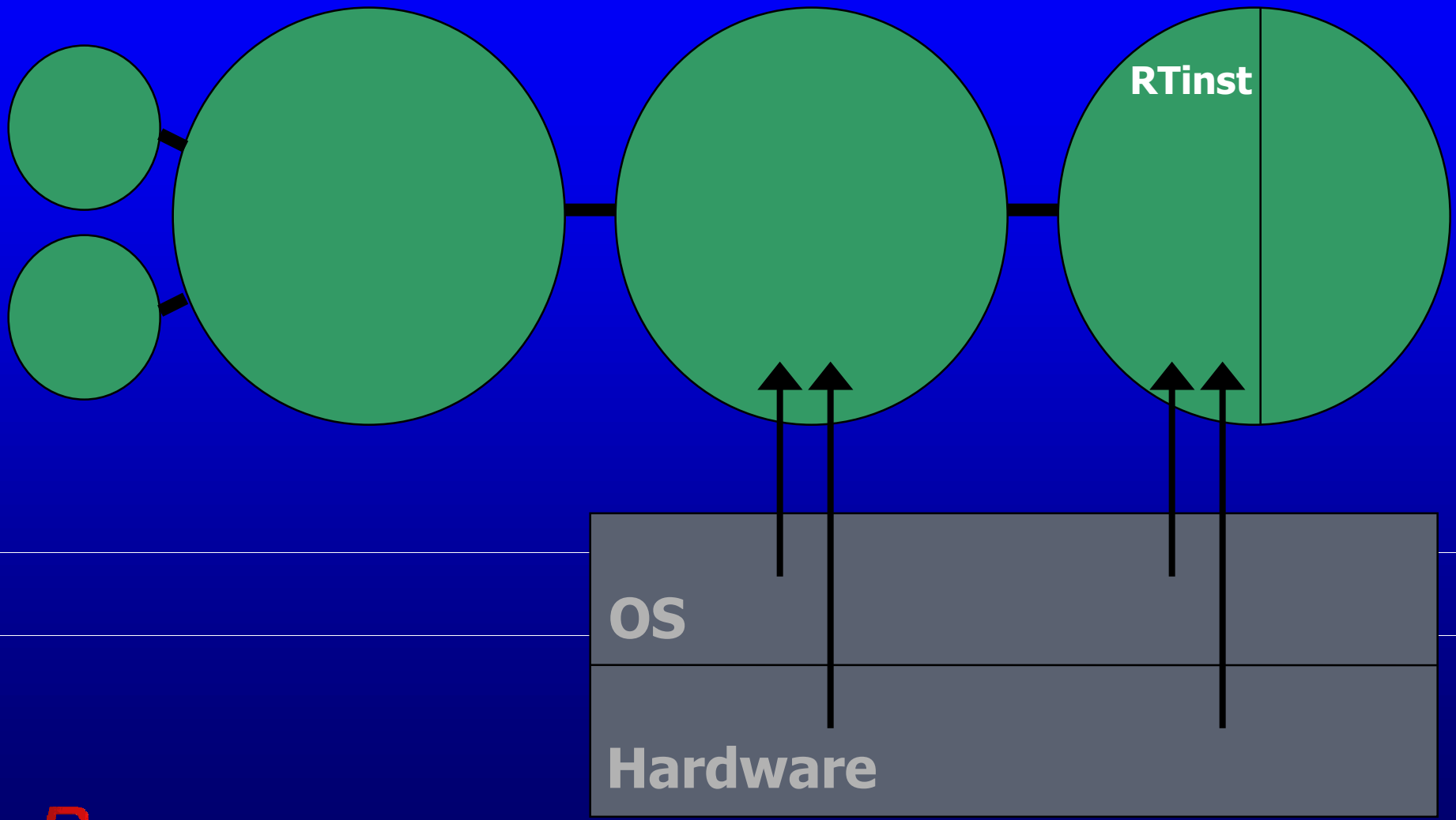

Visis
Paradyn Front-end
Paradyn Daemon
Application
RTinst
OS
Hardware

Visis

Paradyn Front-end

Paradyn Daemon

Application

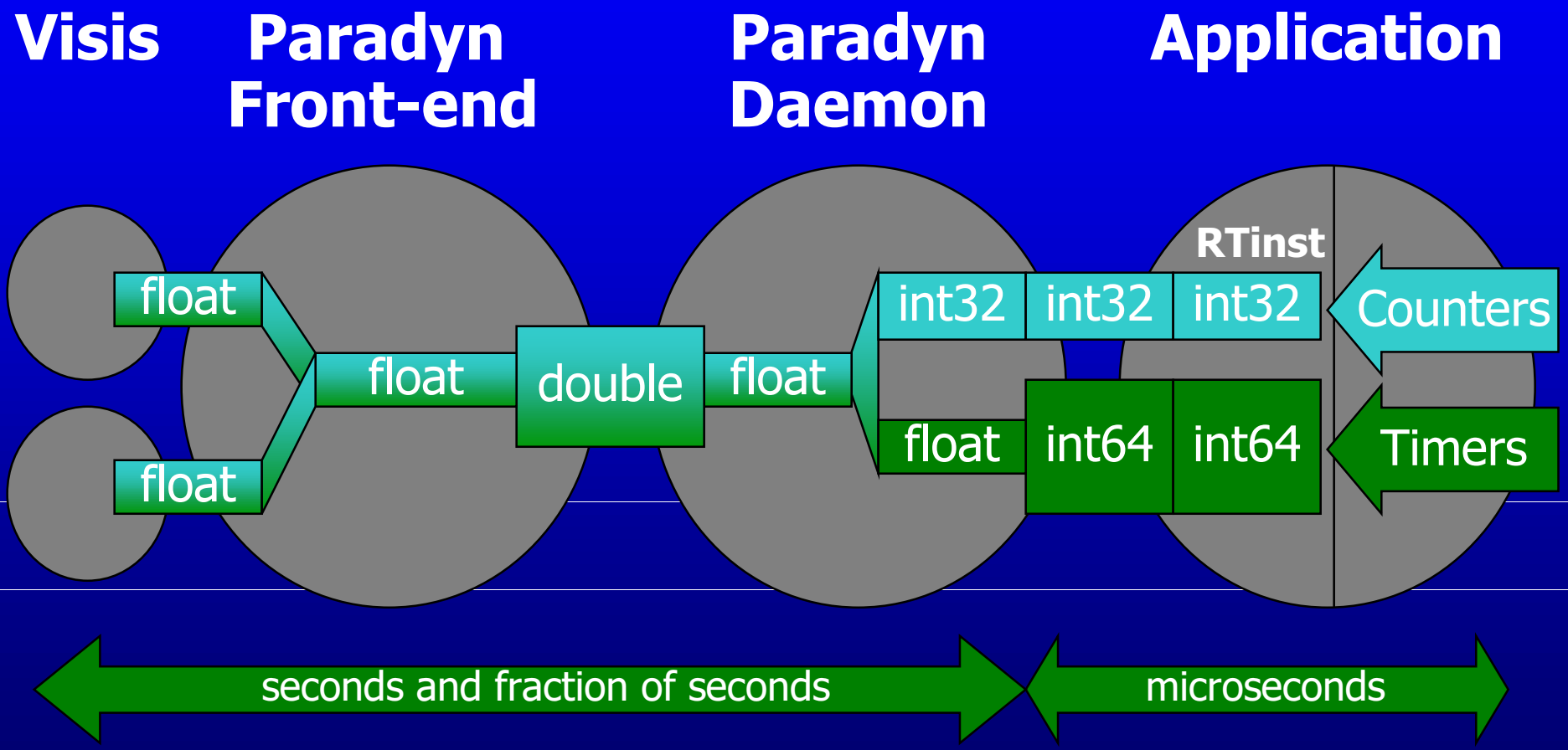


Motivation

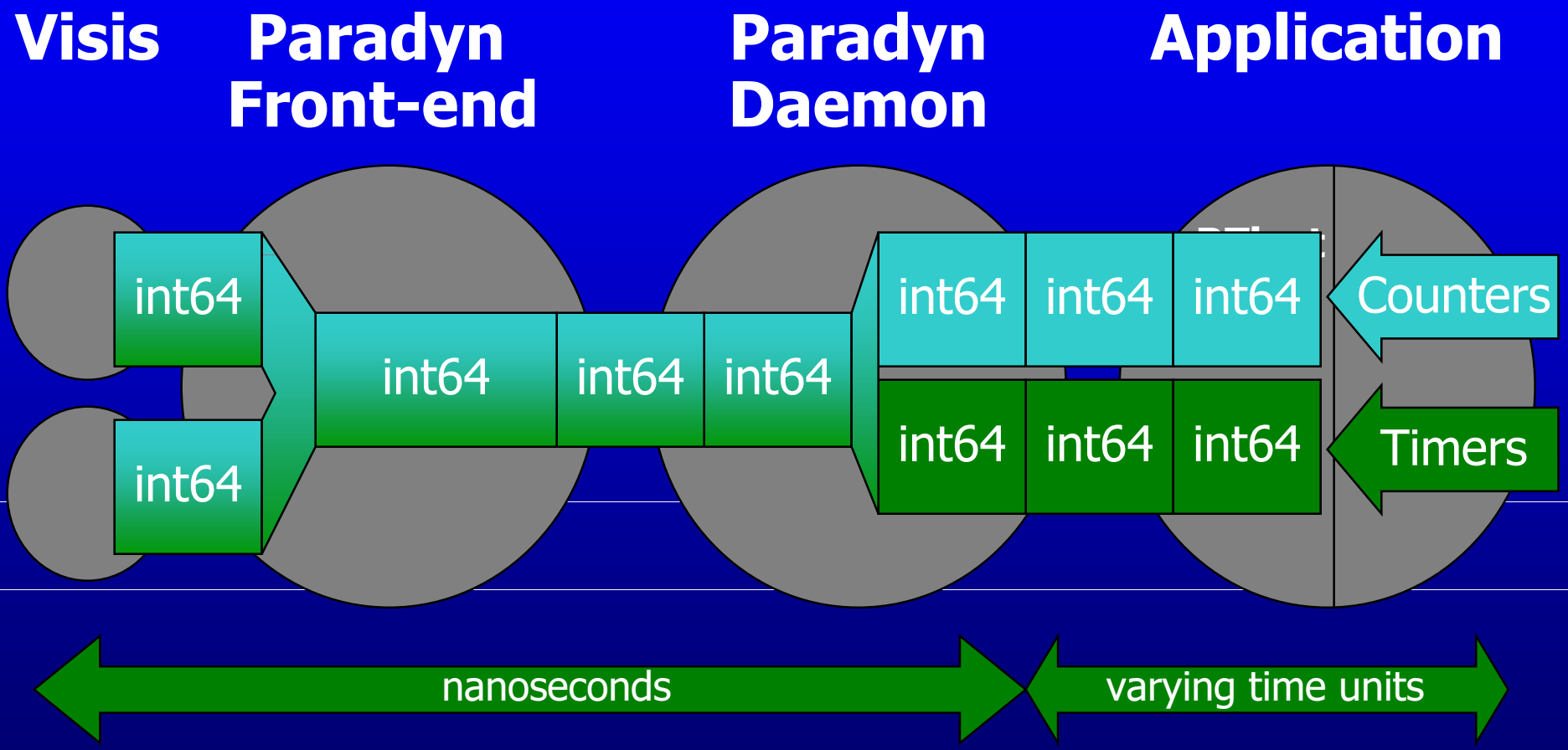
- Sample data pipeline through Paradyn is too narrow to support high resolution counters.

Used with permission of Bob Thaves.

Current Sampling Pipeline



Future Sampling Pipeline



Data Type Tradeoffs

How long can we exactly count nanoseconds?

	float	double	int64
bits of precision	23	52	63
largest counting value	8×10^6	4×10^{15}	9×10^{18}
time	8 ms	52 days	292 years
aggr w/ 100 processes	80 μs	13 hours	2.9 years
aggr w/ 1000 processes	8 μs	1.3 hours	106 days

Choosing Time Retrieval Method

- Run time check of best available timer
  - Linux
    - use libhrtime if available
    - else default to standard system calls
  - Other platforms
    - similar timer selections

Current Status: Sampling Pipeline

- RTinst library and daemon in release 3.0
- Next step
  1) Front-end
  2) Daemon to Front-end RPC calls
  3) Visis
  4) Visi to Front-end RPC calls

Current Status: Linux Timer Support

- Linux kernel support and library completed
  - Library and patches against the latest stable and development kernels at:

    http://www.cs.wisc.edu/paradyn/libhrtime/
- TSC offset measurement on SMP boxes unimplemented
- Not yet accepted into the main kernel tree