# Adaptative Fault Probing

presented by

Mathieu Desnoyers and
Michel Dagenais

École Polytechnique de Montréal

December 10th, 2009

# > Summary

- Introduction
- State of the art
- Methodology
- Experimental results
- Discussion
- Conclusion

# > Introduction (1/2)

- Large-scale multiprocessor
- Complexity increase
  - Virtual machines, OS, libraries, applications
- Problems harder to investigate
  - System-wide
  - Occurrence on production systems
  - Timing-related
- Need for system-wide analysis tools
  - Performance, debugging

# > Introduction (2/2)

- Tracing
  - Trace: sequence of events recorded by a probe
  - Purpose: debugging & performance monitoring
  - Typically intrusive
    - Increasing hardware resources not a solution
- Tracing vs profiling
  - Complete sequence of events vs sampling

# > Objectives (1/2)

- Meet requirements, solve problems identified by
  - The industry
  - Open source community
- Implement a tracer for Linux
  - Mainstream operating system

# > Objectives (2/2)

- Characteristics of each tracer component
  - *Scalability*
  - Low-impact on the operating system *throughput*
  - Low-impact on average *latency*
- Guarantee a deterministic impact of tracing on *real-time* response
- Provide high *portability* and *reentrancy* of tracer mechanisms

# > State of the Art

- Computer architectures
  - Increase in parallelism
  - Memory accesses increasingly costly
- Real-time
  - VxWorks, RTAI, Linux RT
- Distributed systems
  - From message passing (MPI)
  - To RPC (map-reduce)

# > State of the Art (Tracing)

- LTT
- SystemTAP
  - Kprobes, Linux Kernel Markers, Tracepoints
- KTAU
- K42
- Dtrace
- Ftrace
  - Kprobes, Tracepoints

# > Methodology

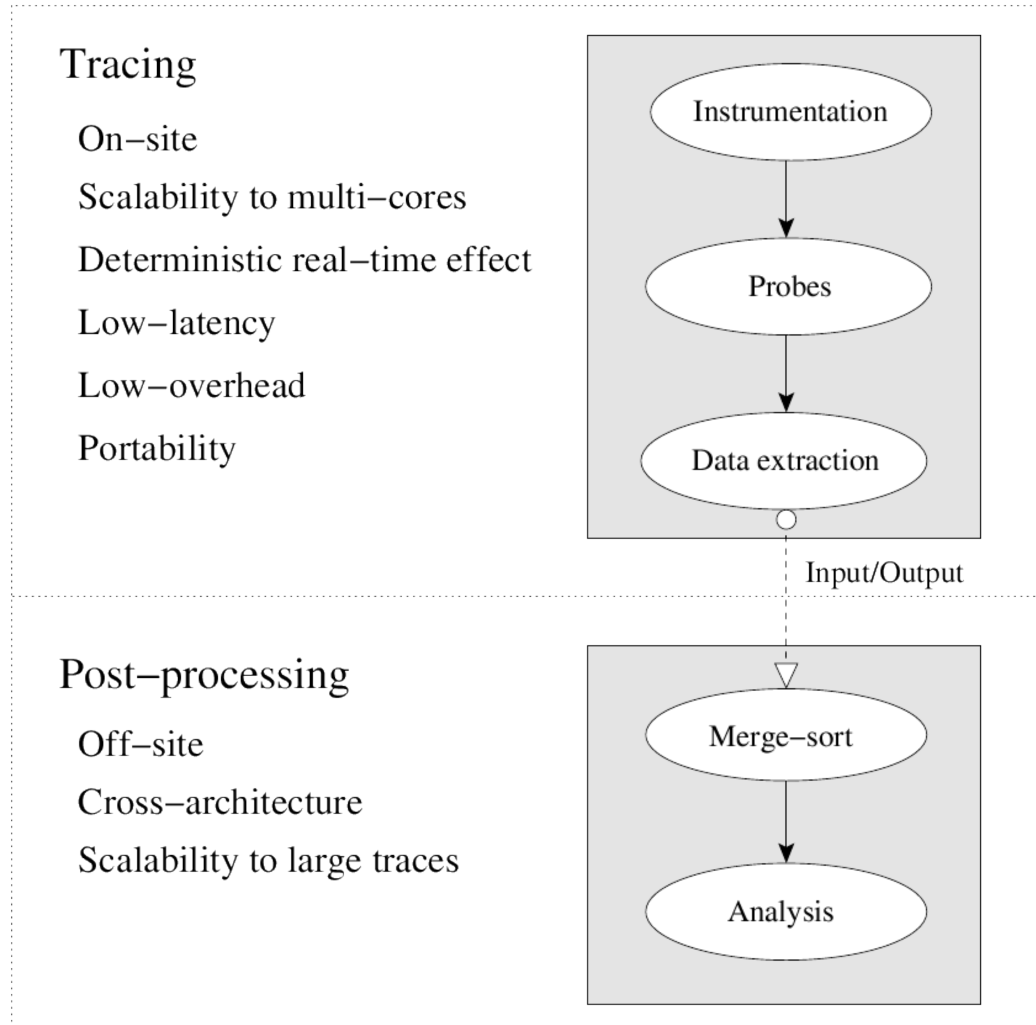
- Interaction with the community
- Tracer design
- Implementation
- Verification

# > Interaction with the Community

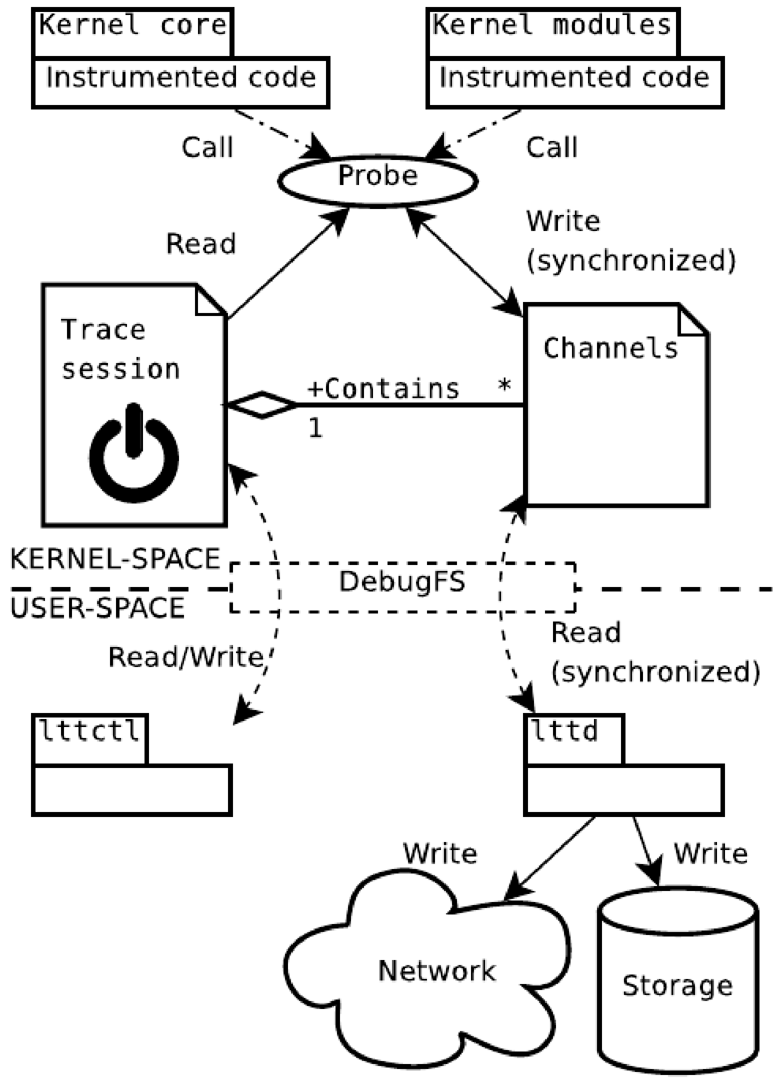
- Industry
  - Autodesk, IBM Research, Google
- Open Source Community
- Conferences
  - Linux Symposium
  - Linux Foundation Collaboration Summit
  - Linux Plumbers Conference
  - Embedded Linux Conference
  - Recon

# > Tracer Design

## Tracing phases properties



# > Tracer Components Overview

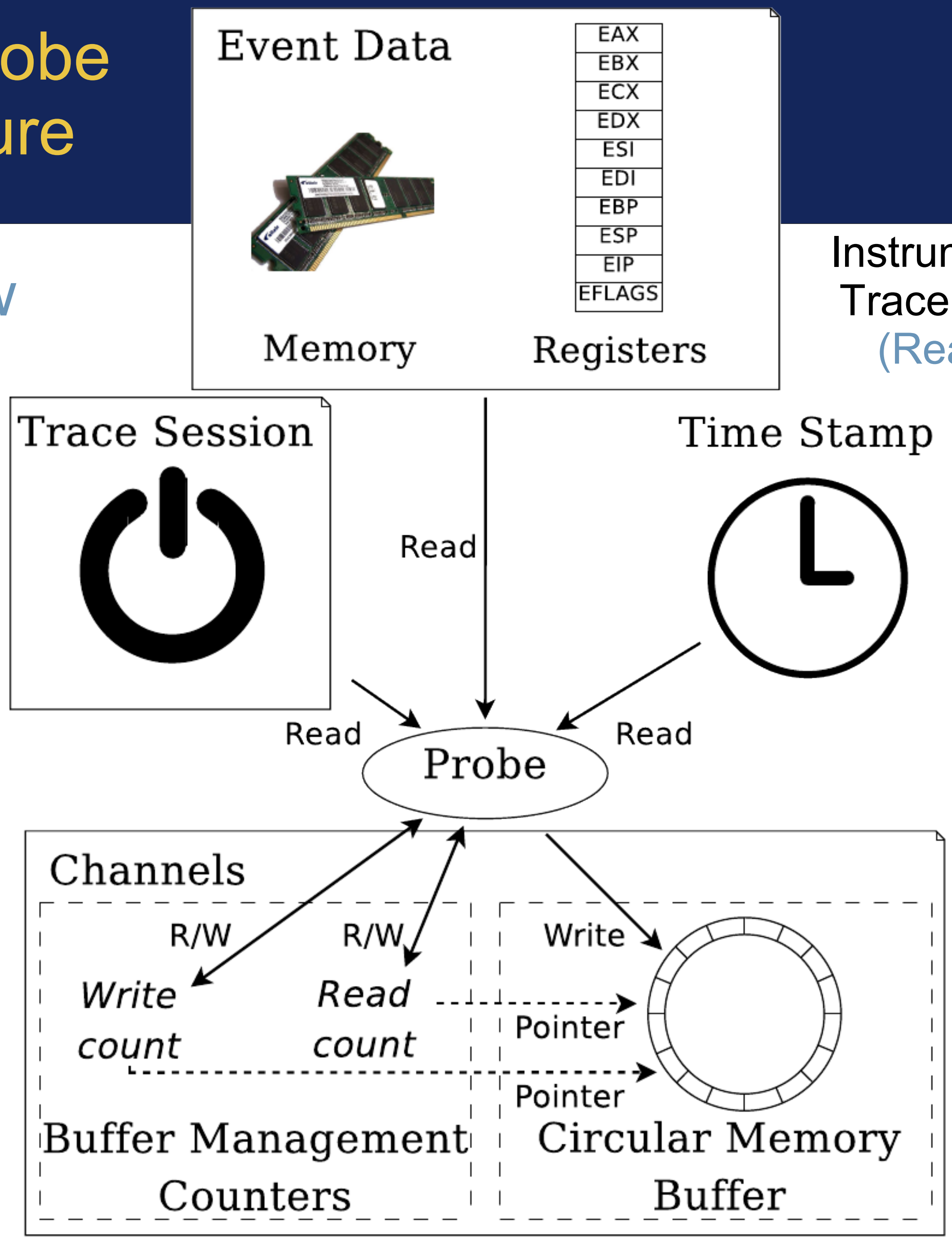


# > Tracer Probe Architecture

## Probe data flow

Instrumentation: Kernel Markers, Tracepoints, Immediate Values.
(Read-Copy Update (RCU))

Tracer control
(RCU)



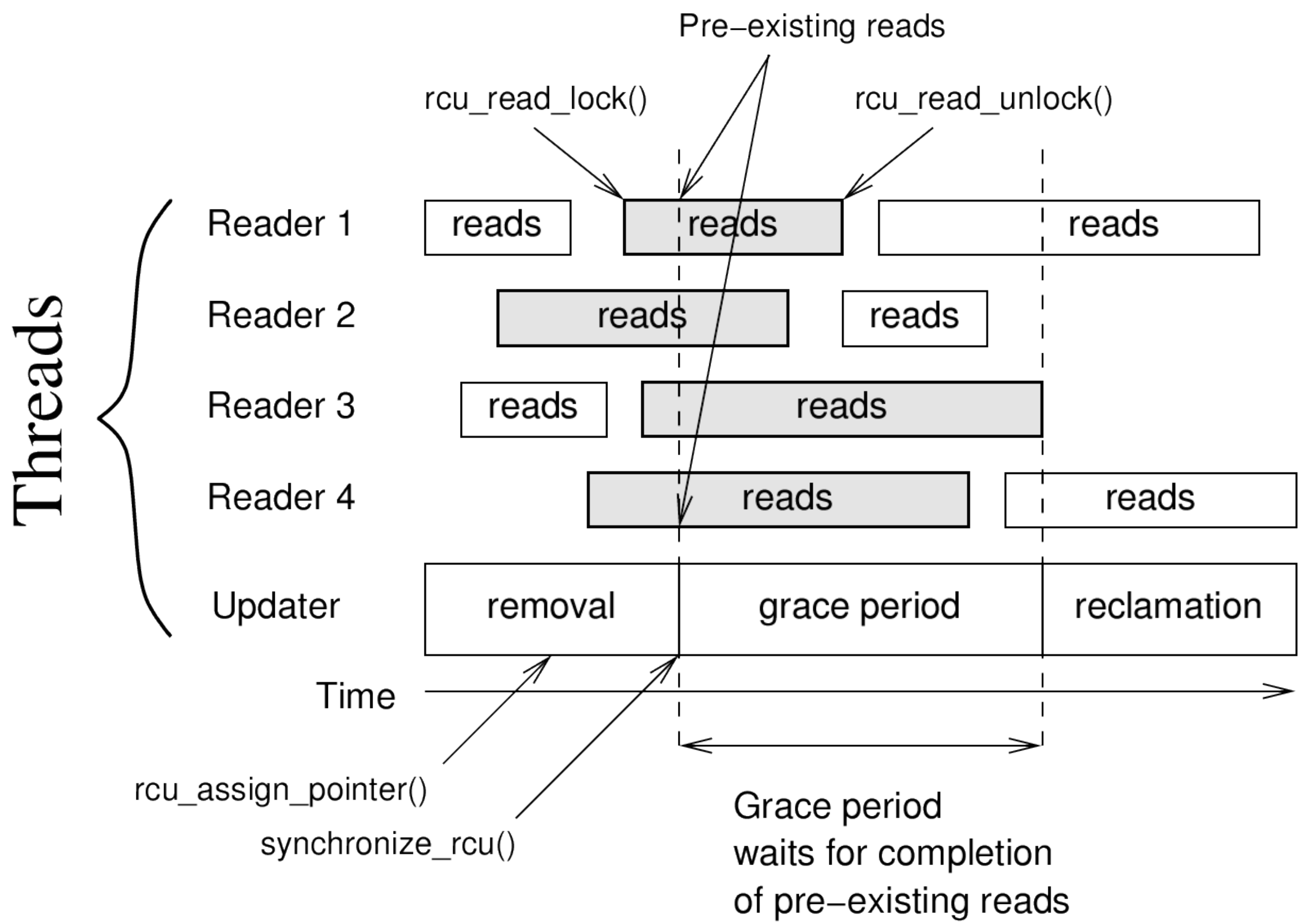

Trace clock extension
(RCU)

LTTng wait-free buffering scheme
(local atomic operations)

# > Implementation

- User-space RCU library (liburcu)
- Static instrumentation
  - Tracepoints, Markers, Immediate Values
- LTTng kernel tracer
  - Buffering scheme
  - Trace clocks

# > Read-Copy Update (RCU)




Schematic of RCU grace period and read-side critical sections

# > User-space RCU

- Goal for user-space tracing
  - Highly scalable
  - Trace signal handlers
- Need to support being used from tracer library without modifying the application
- Need for high-performance read-side
  - Signal-based memory barriers
  - Use thread-local storage

# > Instrumentation Mechanisms

- Static tracepoints
  - **Tracepoints**, **Markers**, Trace events
  - Optimizations
    - **Immediate values**
    - **Static jump patching**
- Dynamic tracepoints
  - Kprobes, GDB tracepoints

# > Static Tracepoints

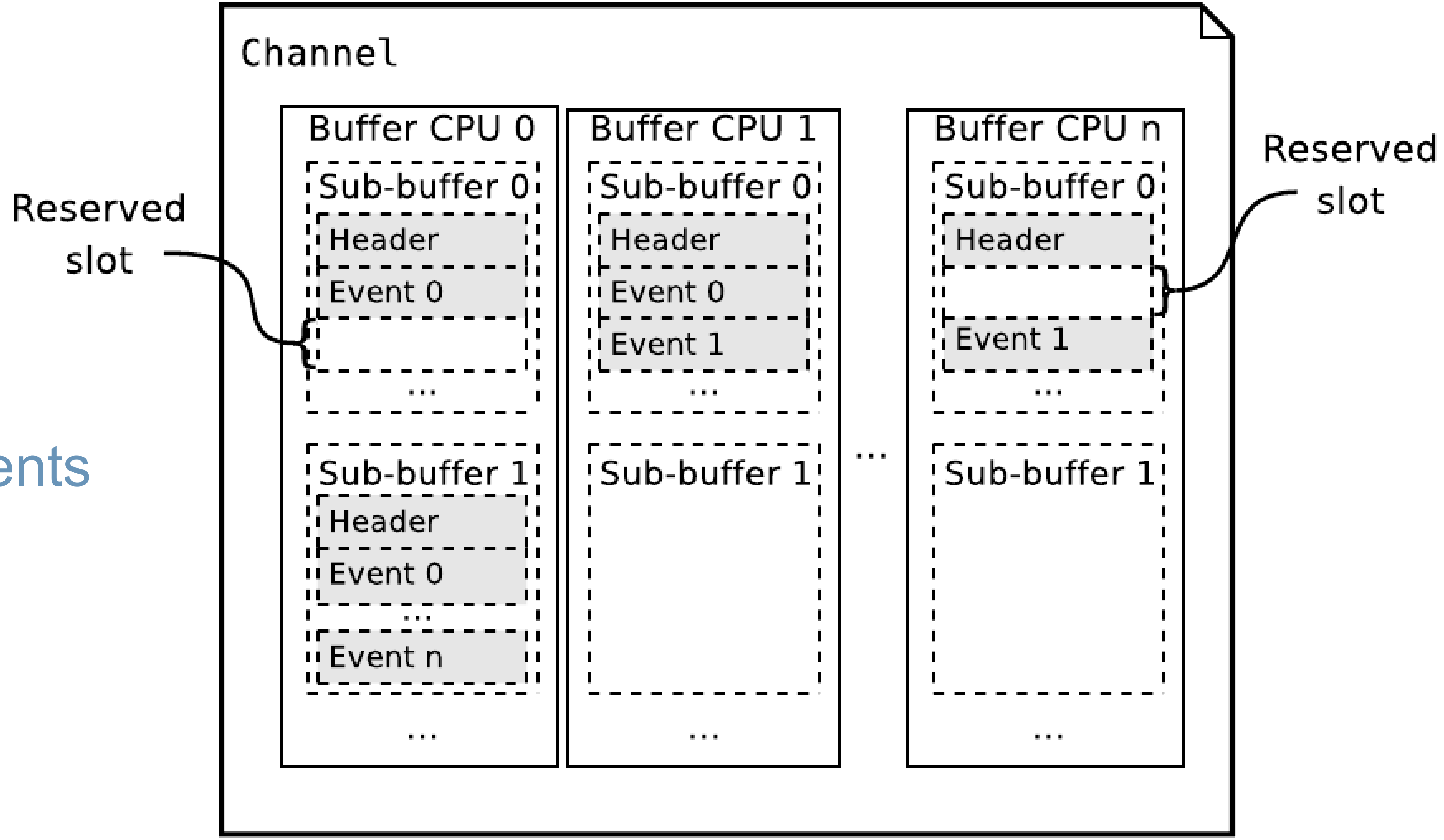
- Declared at source-code level, enabled dynamically
- Easy to manage within distributed source-control
- Easy to use by field engineers
- Based on a branch over a function call
- GCC optimization-friendly
  - Guarantee presence of parameters at call site
- Faster than dynamic tracepoints when enabled
- Adding new TP requires to recompile

# > Immediate Values

- Efficient tracepoint activation
- Encode branch condition in instruction stream
- Low-latency instruction patching
  - Based on djprobes work
- Led to gcc “asm goto” (gcc 4.5)

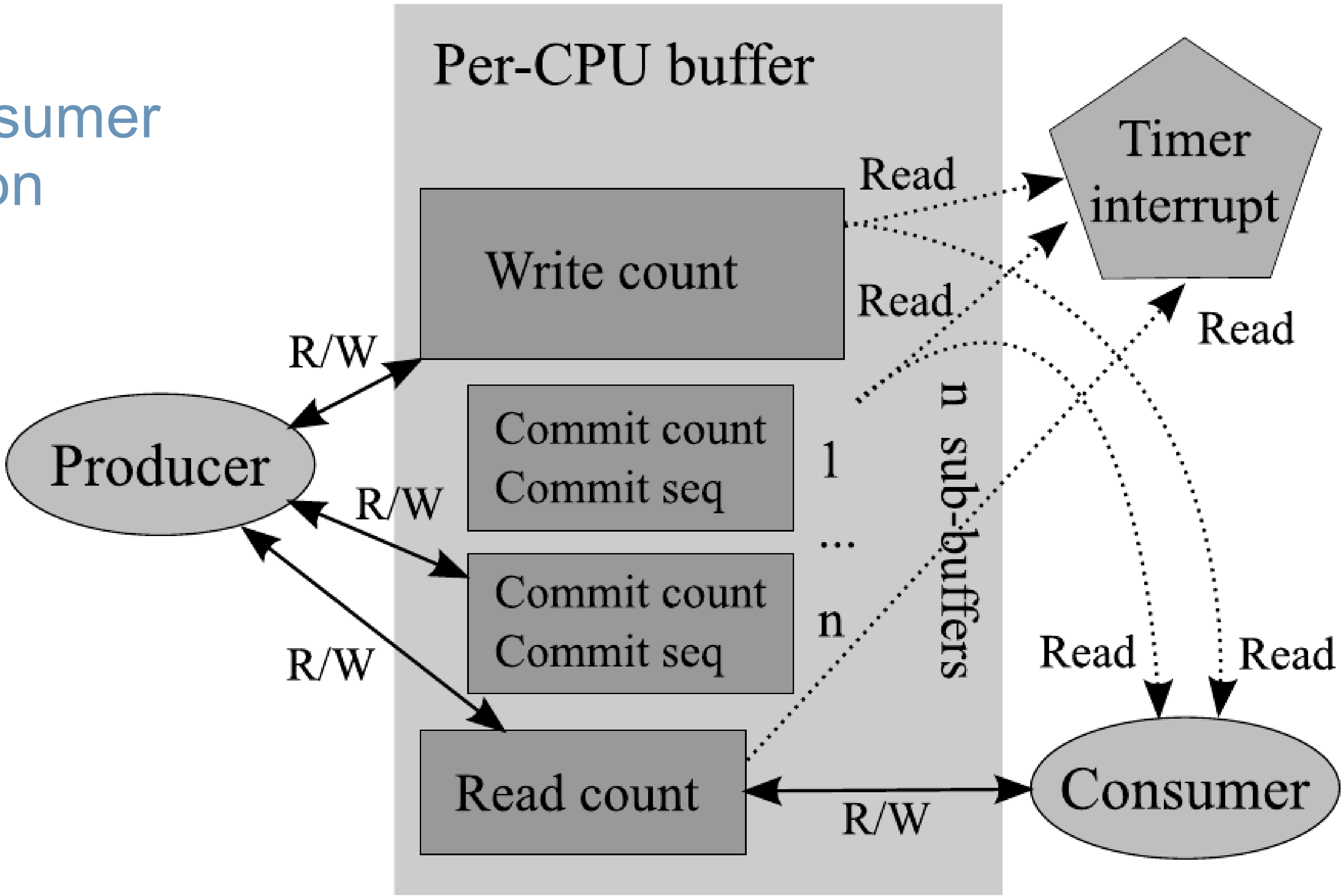
# > LTTng Buffering Scheme (1/2)

Channel components

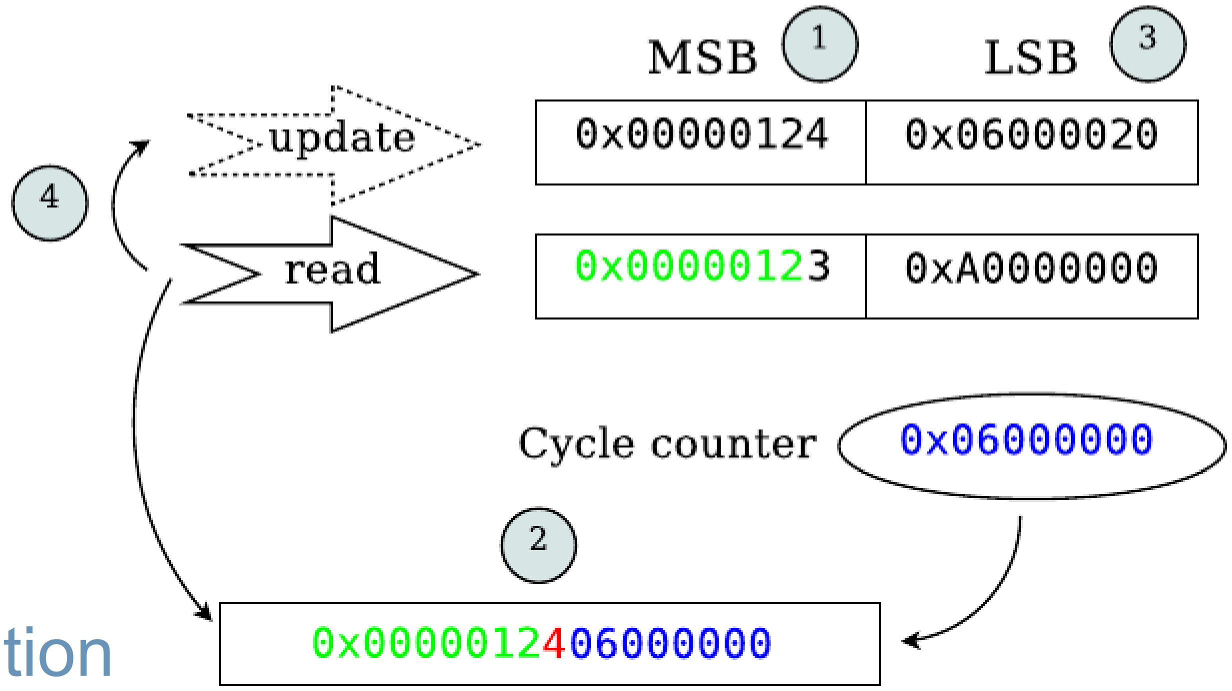


# > LTTng Buffering Scheme (2/2)

Producer-Consumer Synchronization



# > LTTng Trace Clocks




RCU-based synchronization

Trace clock update (1, 3, 4) interrupted by a read (2)

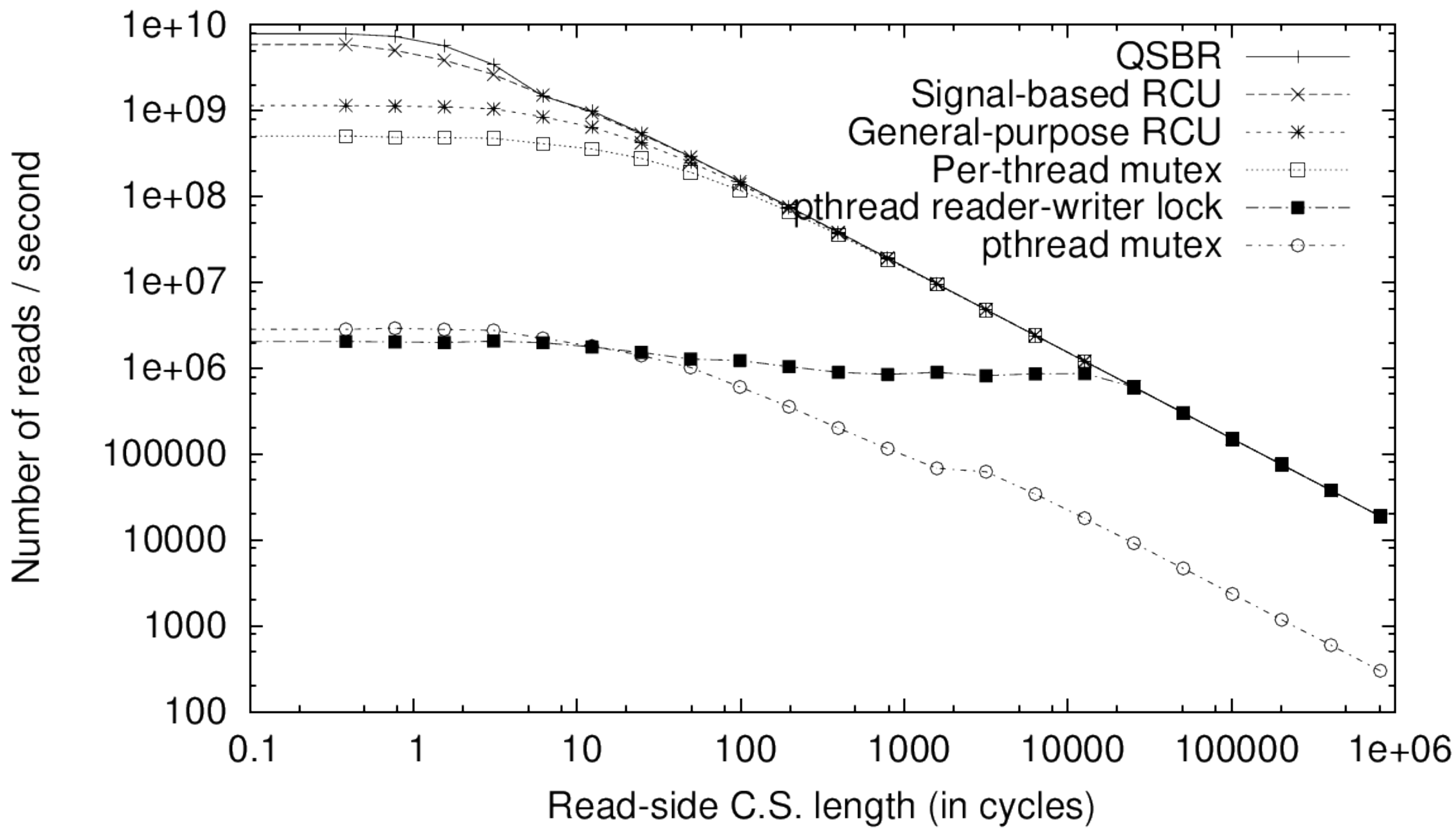
# > Experimental Results

- Benchmarks
- Formal verification

# > Benchmarks

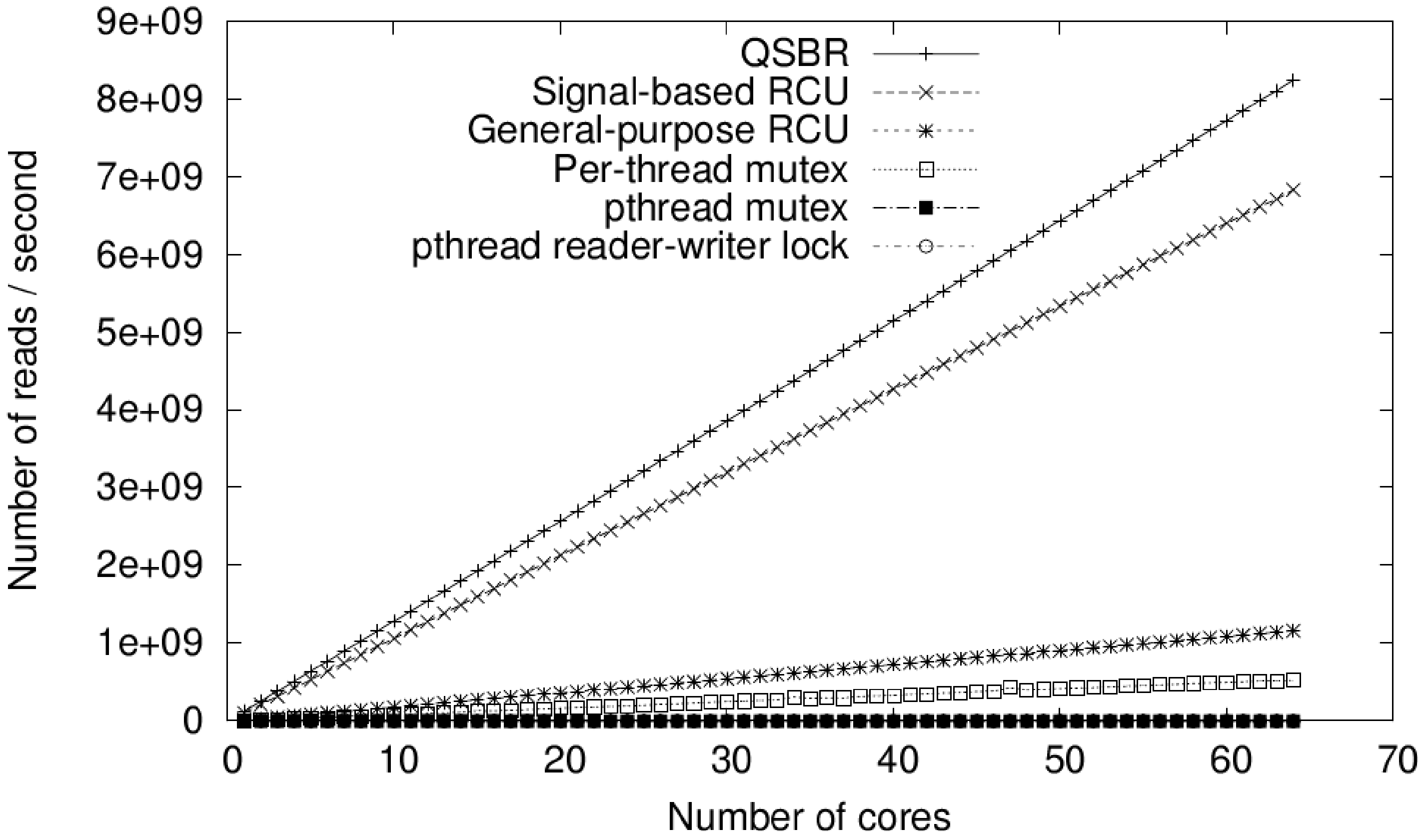
- Read-Copy Update (user-level)
  - Read-side overhead
  - Read-side scalability
- LTTng buffering scheme
  - Latency
  - Throughput
  - Scalability

# > RCU Read-side Overhead




Impact of read-side critical section length, 64 reader threads on POWER5+.
*Logarithmic scale.*

# > RCU Read-side Scalability




Read-side scalability for various synchronization primitives, 64-core POWER5+.
*Linear scale.*

# > LTTng Latency Impact

| Test | Events / round-trip | avg. ($\mu$s) | std.dev. ($\mu$s) |
|---|---|---|---|
| No tracing | – | 40.0 | 12.8 |
| Flight recorder tracing | 27 | 49.0 | 14.3 |

Tracer latency overhead for a ping round-trip. Local host, Linux 2.6.30.9, Intel Xeon 2.0 GHz, 100 000 requests sample, at 2 ms interval. With background noise.

- Added latency between 328 and 338 ns per event (95 % confidence interval).
  - 666 cycles per event (normal cache behavior)
- Cache-hot micro-benchmarks: 119 ns
  - 238 cycles per event (cache hot)

# > LTTng Latency Impact (cache-hot)

| Architecture | Cycles | Core freq. (GHz) | Time (ns) |
|---|---|---|---|
| Intel Pentium 4 | 545 | 3.0 | 182 |
| AMD Athlon64 X2 | 628 | 2.0 | 314 |
| Intel Core2 Xeon | 238 | 2.0 | 119 |
| ARMv7 OMAP3 | 507 | 0.5 | 1014 |

Cycles taken to execute a LTTng 0.140 probe, Linux 2.6.30.

# > LTTng Throughput Impact (1/4)

| Test | Tbench Throughput (MB/s) | Overhead (%) | Trace Throughput ($*10^3$ events/s) |
|---|---|---|---|
| Mainline Linux kernel | 12.45 | 0 | – |
| Dormant instrumentation | 12.56 | 0 | – |
| Overwrite (flight recorder) | 12.49 | 0 | 104 |
| Normal tracing to disk | 12.44 | 0 | 107 |

`tbench` client network throughput tracing overhead.

# > LTTng Throughput Impact (2/4)

| Test | Tbench Throughput (MB/s) | Overhead (%) | Trace Throughput ($*10^3$ events/s) |
|---|---|---|---|
| Mainline Linux kernel | 2036.4 | 0 | – |
| Dormant instrumentation | 2047.1 | -1 | – |
| Overwrite (flight recorder) | 1474.0 | 28 | 9768 |
| Normal tracing to disk | – | – | – |

`tbench` localhost client/server throughput tracing overhead.

# > LTTng Throughput Impact (3/4)

| Test | Dbench Throughput (MB/s) | Overhead (%) | Trace Throughput ($*10^3$ events/s) |
|---|---|---|---|
| Mainline Linux kernel | 1334.2 | 0 | – |
| Dormant instrumentation | 1373.2 | -2 | – |
| Overwrite (flight recorder) | 1297.0 | 3 | 2840 |
| Non-overwrite tracing to disk | 872.0 | 35 | 2562 |

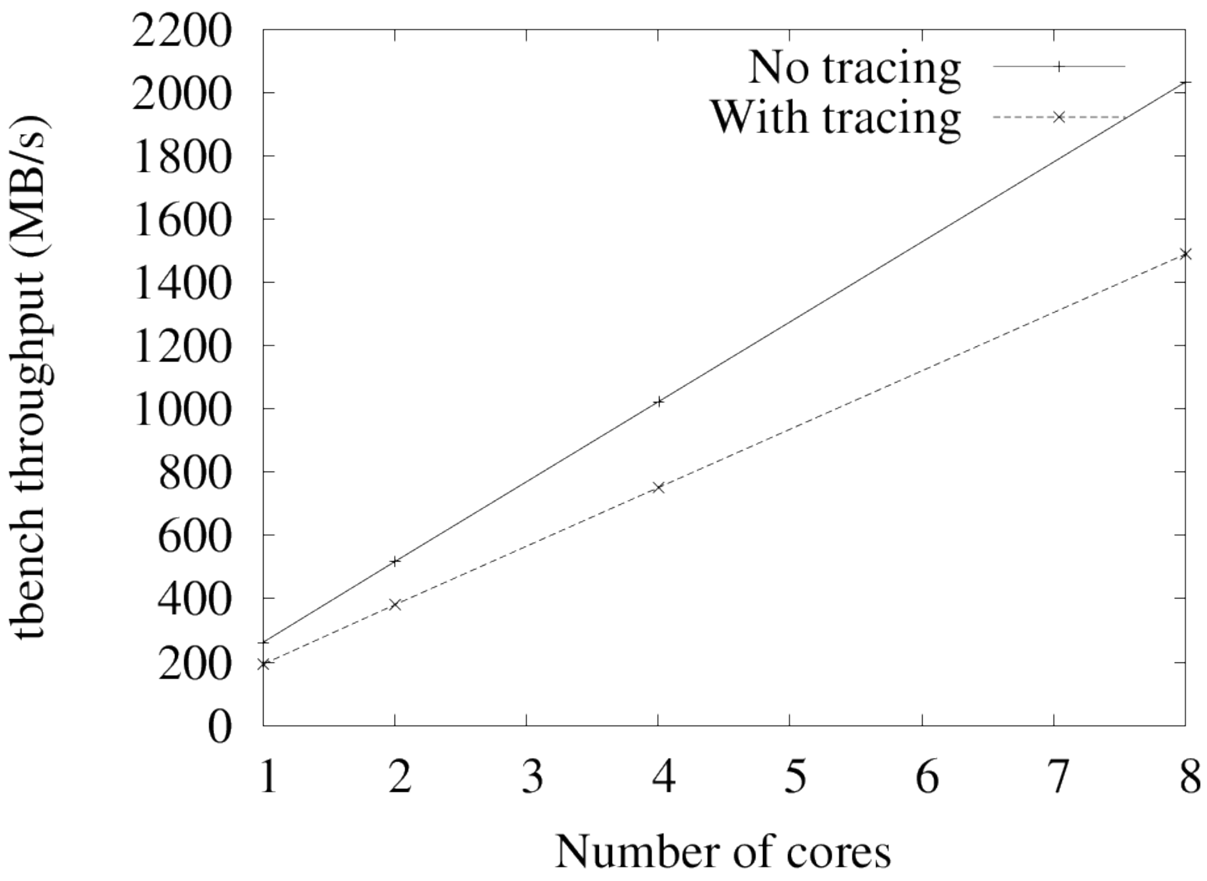
`dbench` disk write throughput tracing overhead.

# > LTTng Throughput Impact (4/4)

| Test | Time (s) | Overhead (%) | Trace Throughput ($*10^3$ events/s) |
|---|---|---|---|
| Mainline Linux kernel | 85 | 0 | – |
| Dormant instrumentation | 84 | -1 | – |
| Overwrite (flight recorder) | 87 | 3 | 822 |
| Normal tracing to disk | 90 | 6 | 816 |

Linux kernel compilation tracing overhead.

# > LTTng Scalability Impact




Impact of tracing overhead on localhost tbench workload scalability.

# > Formal Verification

- Model-checking
  - SPIN model-checker
- Models
  - LTTng buffering scheme
  - Read-Copy Update implementations

# > LTTng Buffering Scheme Model

- Characteristics verified:
  - Correctness
    - No buffer data corruption
  - Real-time impact
    - Wait-free (kernel)
    - Lock-free (user-space)
  - Reentrancy
    - Nested NMI-handler progress ensured by wait-free and lock-free guarantees.
- Model coverage verified with error-injection

# > RCU Implementations Model

- Out-of-order memory access model
- Weakly-ordered instruction scheduling model
- Model coverage verified with error-injection
- Correctness
  - Publication and grace-period guarantees
- Progress verification
  - Read-side wait-free
  - Write-side is never starved by readers

# > Discussion

- Tracer properties
- Application domain

# > Tracer Properties

- Latency
- Throughput
- Scalability
- Real-time
- Portability
- Reentrancy

# > Application Domain

- Live production commercial servers
  - Stability (correctness proofs)
  - Require low-overhead tracer
- Soft real-time applications
  - Video edition, telecommunication
  - Soft real-time, high-throughput
- Real-time distributions
  - Wind River Linux, Monta Vista, STLinux
  - Require predictable RT impact (wait-free)

# > Conclusion

- Research
- Original scientific contributions
- Future research perspectives

# > Research (1/4)

- Brings further
  - Lock-less buffering schemes, pioneered by the K42 tracer (Robert Wisniewski)
  - User-level RCU implementations
    - Usable in production (Debian, Gentoo)
  - Formal verification of parallel algorithms at the architecture level

# > Research (2/4)

- Journal articles
  - Wiley Software – Practice and Experience
    - Synchronization for Fast and Reentrant Operating System Kernel Tracing
      - Recommended for publication
  - ACM TOCS
    - Lockless Multi-Core High-Throughput Buffering Scheme for Kernel Tracing
  - IEEE TPDS
    - User-Level Implementations of Read-Copy Update
    - Multi-Core Systems Modeling for Formal Verification of Parallel Algorithms

# > Research (3/4)

- Impact (research articles using LTTng)
  - Power variations over time in disk operations
  - Study which applications are run concurrently over a long period of time
  - Feed information to an anomaly detection service, part of an operating system
  - Hooks to monitor kernel execution inspired from Tracepoints (Lemona)

# > Research (4/4)

- Original scientific contribution
  - LTTng buffer synchronization algorithm
  - Creation of an RCU-based trace clock
  - Design of complete kernel tracer
    - Wait-free, linearly scalable, NMI-safe algorithms
  - Self-modifying code technique to activate instrumentation
  - User-space RCU improvements
  - Out-of-order architecture model for formal verification

# > Objectives (1/2)

- All tracer properties met
  - Latency
  - Throughput
  - Scalability
  - Real-time
  - Portability
  - Reentrancy

# > Objectives (2/2)

- Used by the industry
  - Google
  - IBM
  - Ericsson
  - Autodesk
  - Wind River
  - Fujitsu
  - Monta Vista
  - STMicroelectronic
  - C2 Microsystems
  - Sony
  - Siemens
  - Nokia
  - Defence Research and Development Canada.

# > Future Research Perspectives

- New analysis
  - System-wide traces from production systems
  - Energy efficiency
  - Performance improvements
- Trace time synchronization
  - Multi-nodes
  - Non-synchronized TSC
- Architectures with non-coherent caches
  - Blackfin, Intel 48-core

# > Questions ?

?

- LTTng project website: http://www.lttng.org