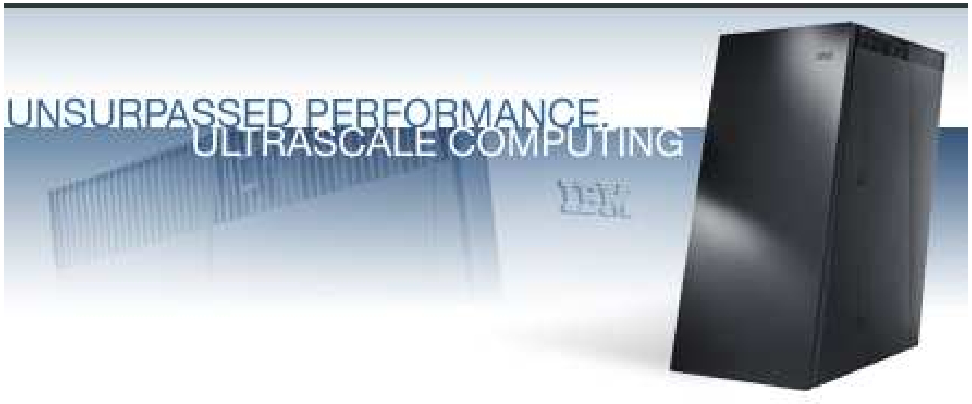
January 29, 2008

IBM

Tracing for Performance Monitoring on Parallel and Distributed Systems



Dr. Robert W. Wisniewski
Manager Blue Gene Software
IBM T. J. Watson Research
http://www.research.ibm.com/people/b/bob/

Outline

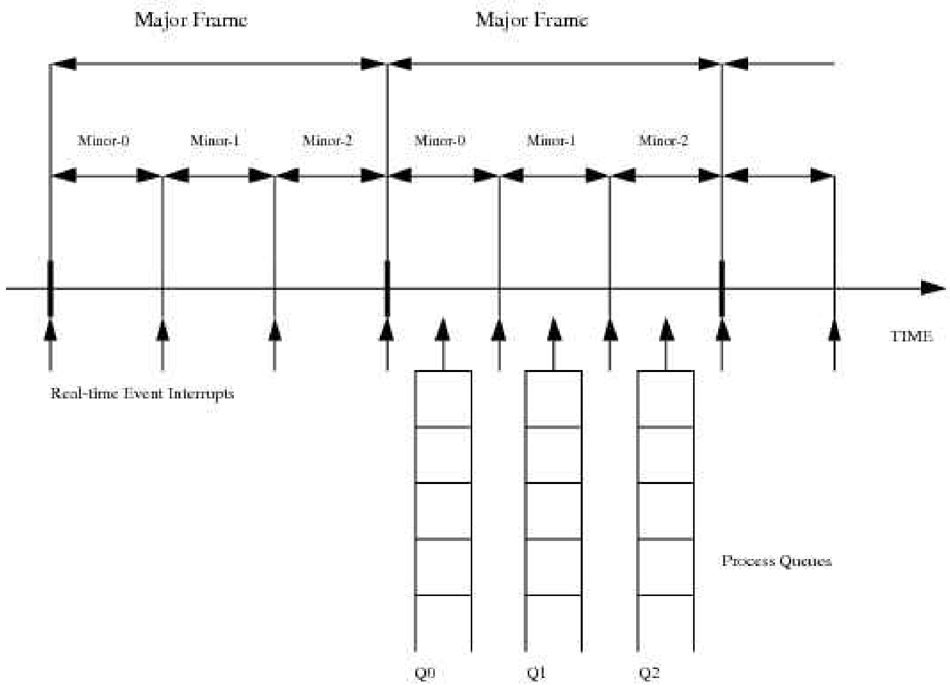
- SGI (RTAS – Real-Time Technology and Applications Symposium 95)
  - rtmon
  - Kernel and Cray Unification
  - Lessons
- K42 (Supercomputing 03)
  - Approach, scalability, and use
  - Lessons
- CPO (Continuous Program Optimization) (PAC2 2004)
  - PEM (Performance Environment Monitoring)
  - Lessons
- CSO (Commercial Scale-Out) (europar07 – slides thanks Jose Moreira)
  - Goals
  - Lessons
- Blue Gene / P (internal – slides thanks Valentina Salapura)
- Observations on Linux and LTT
- The “next system” - concluding remarks

SGI rtmon

Frame Scheduler

Major Frames: Determines period - a complete cycle of processes

Minor Frames: Independent units within major frame - used for setting up sepcific application behavior

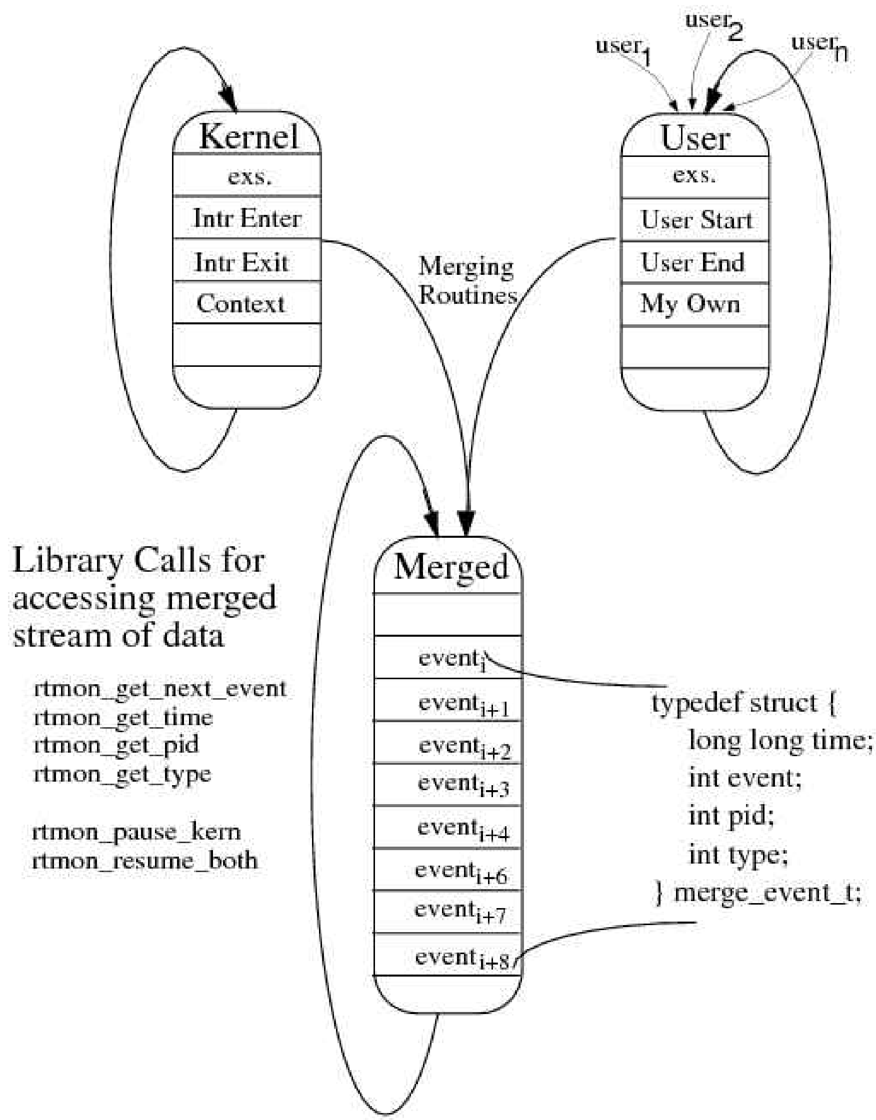


Processes can be enqueued in more than one process queue

SGI rtmon

- **rtmon bottom layer**
  - **Per-processor**
  - **Multiple writes user q**
    - **•atomicIncWrap gets index**
    - **• Set valid bit when done**
    - **•Reader clears valid bit**

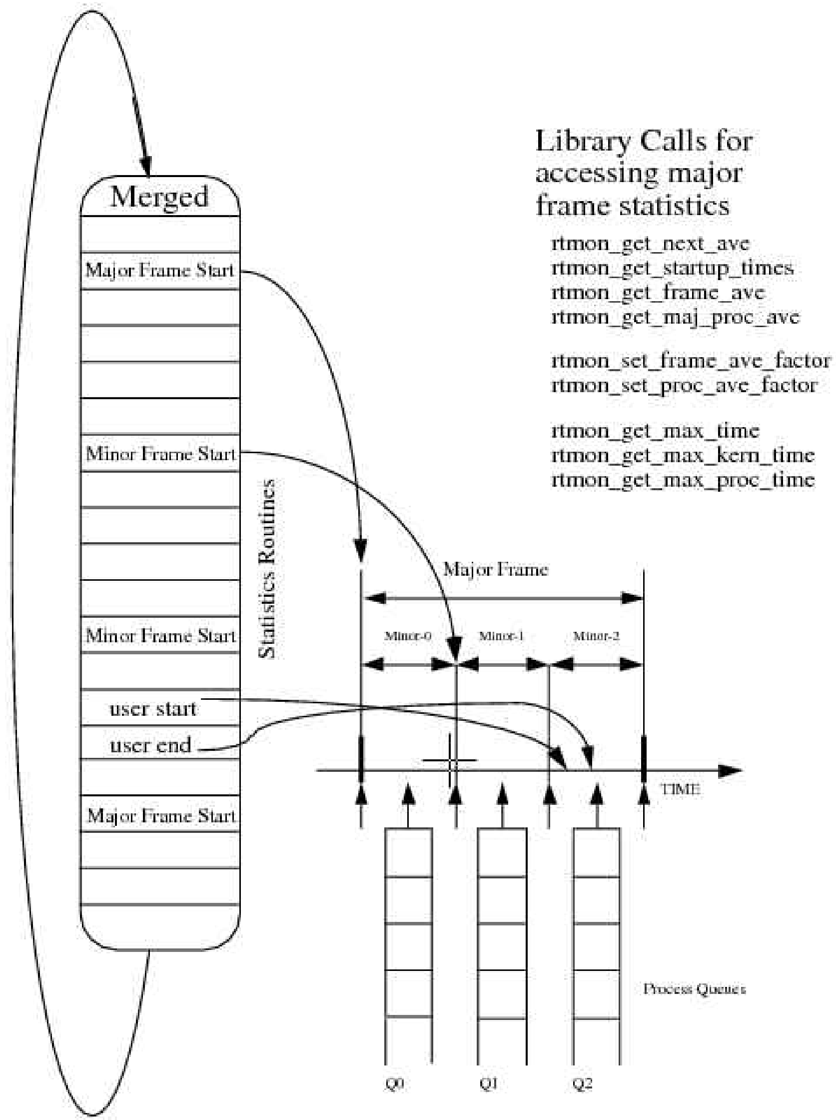
FrameView - Bottom Layer



SGI rtmon

FrameView - Middle Layer

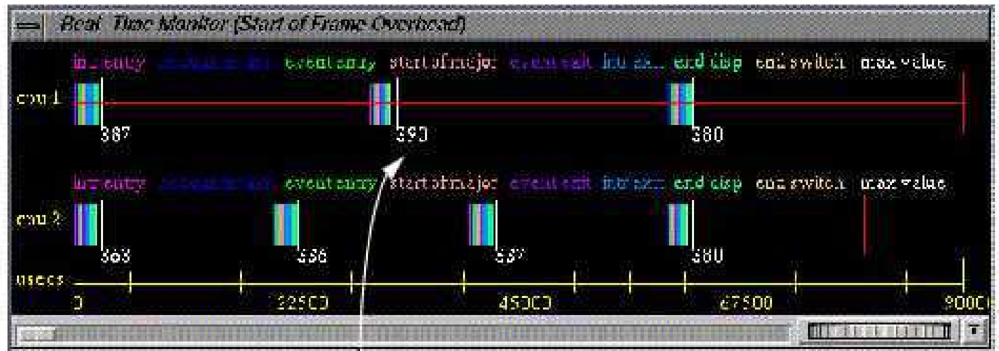
- **rtmon middle layer**
  - **Assign meaning to events, recreate frames**
  - **Report discrepancies**
  - **Calculate extreme value**



SGI rtmon

- **rtmon top layer**
  - **Multiple view**

Kernel Startup Graph
Real-Time Monitoring Tools

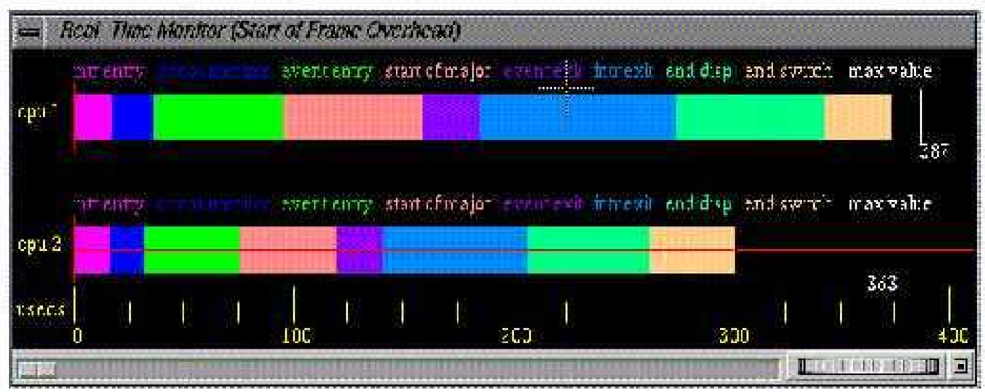


390
Max Kernel Startup Time
Minor Frame 1

Above represents full view matching the view as seen in the main graph

Below represents a blown up image of the kernel startup time for minor frame 0

Colors of event labels match color bars



SGI Kernel and Cray Unification

- **rtmon extended to kernel**
  - **3 separate tracing schemes depending on what you were doing**
    - **Confusing**
    - **Error prone**
    - **Hurts performance**
- **SGI purchases Cray**
  - **5 separate tracing schemes…**
  - **Cray introduces another aspect**
    - **Need data from machines in field that are not possible to build in house – requires extensive events and black-box capability**

SGI Lessons

- **rtmon**
  - **+ Collect cheaply on line more expensive off line processing**
  - **• Roughly ¼ of machine needed to get events off**
  - **• Tradeoff between application-specific design and generality**
  - **+ Single system of trace events useful**
  - **+ Possible to do non-locking tracing**
  - **– Fixed events are cumbersome**
  - **❶ Visualization is key**
    - **It’s the killer app for tracing**

Outline

- **SGI**
  - **rtmon**
  - **Kernel and Cray Unification**
  - **Lessons**
- **K42**
  - **Approach, scalability, and use**
  - **Lessons**
- **CPO (Continuous Program Optimization)**
  - **PEM (Performance Environment Monitoring)**
  - **Lessons**
- **CSO (Commercial Scale-Out)**
  - **Goals**
  - **Lessons**
- **Blue Gene / P**
- **Observations on Linux and LTT**
- **The "next system" - concluding remarks**

K42's Goals (started 1997)

- Scalability
  - Up to large MP and large applications
  - down for small-scale MP and small apps on large-scale MP
- Flexibility/Customizability:
  - policies/implementations of every physical/virtual resource instance can be customized to application needs
  - system can adapt to security and performance faults without penalizing common case performance
- Portability:
  - can be easily ported to new 64-bit platforms
  - can exploit features of HW
- Availability:
  - fault containment: should be able to survive HW failures on large MP
  - can be dynamically upgraded without bringing system or apps down
- Maintainability/Extensibility:
  - highly module structure
  - re-enable the OS research community
- Full Functionality and Linux compatibility:
  - support huge numbers of Linux apps and drivers without modification
  - transfer technology back and forth to vanilla Linux

Goals: Performance Monitoring

- Provide unified events for correctness and performance
- Allow events to be gathered efficiently on a multiprocessor
- Allow efficient logging of events from applications, servers, and the kernel into a unified buffer with monotonically increasing timestamps
- Have the infrastructure always compiled into the system allowing data gathering to be dynamically enabled
- Separate the collection of events from their analysis
- Have minimal impact on the system when tracing is not enabled; allow for zero impact by providing the ability to "compile out" events
- Provide cheap and flexible collection of data for either small or large amounts of data per event

The Answer To ...

Key Ideas

* lockless logging
* random access variable length events

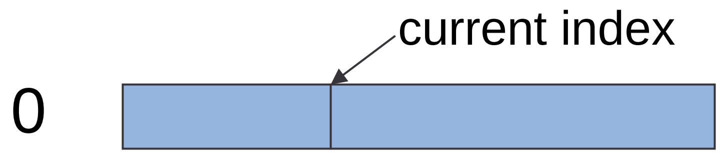
→ unified events

^ user-mapped per-processor buffers

^ major and minor ids

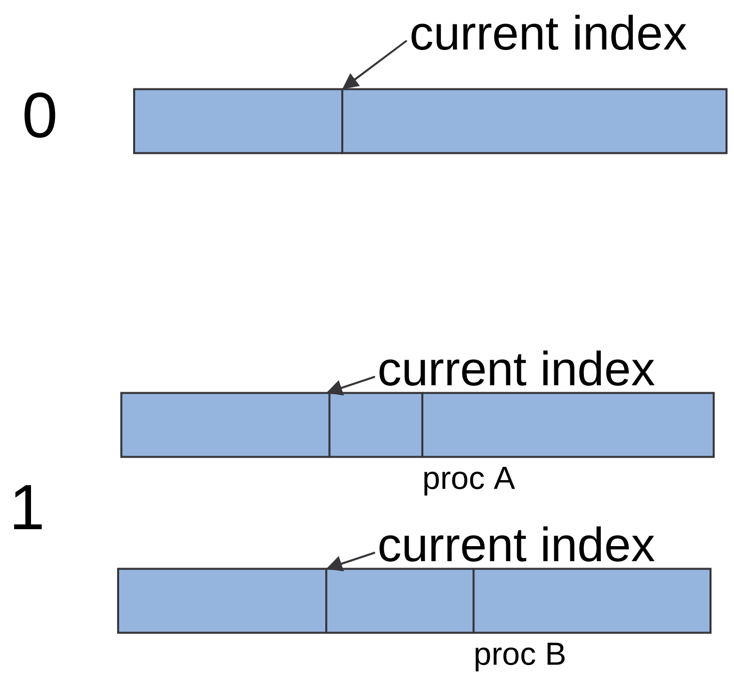
The Answer To ...

Lockless Logging

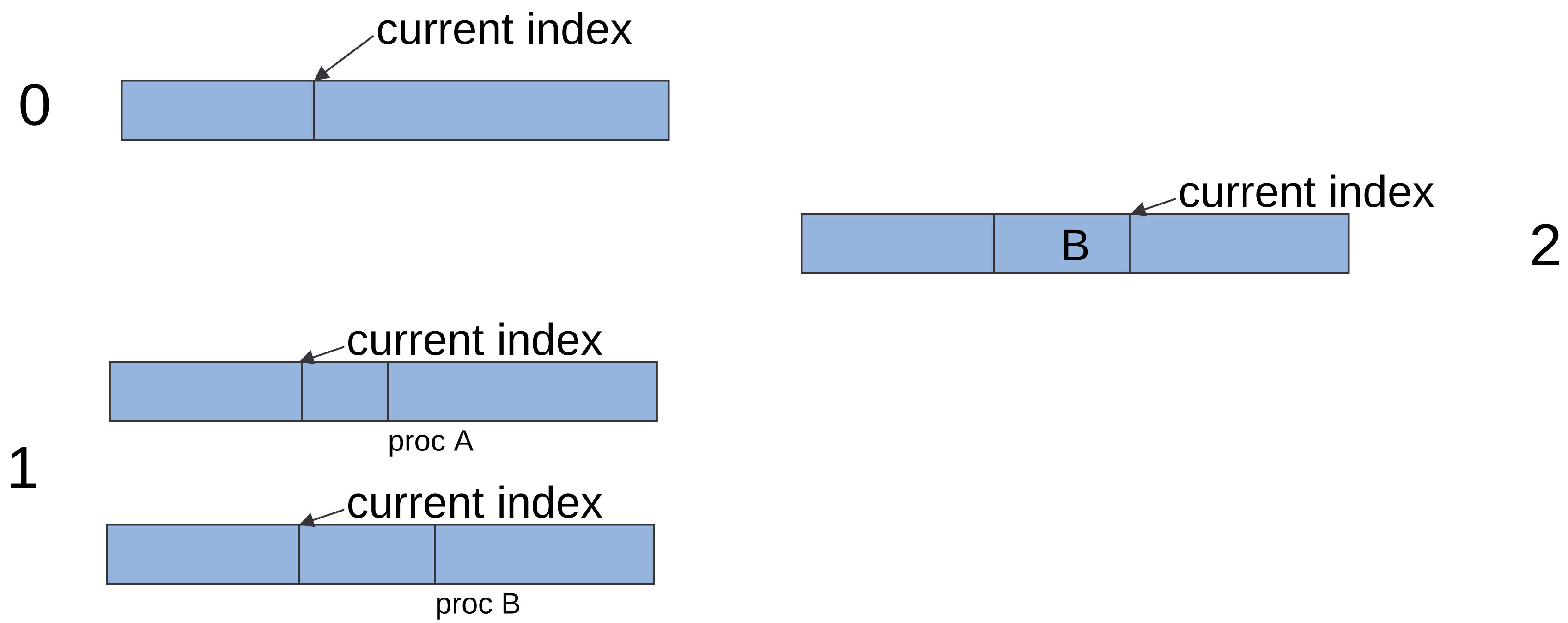


Process A – to log 2 words
Process B – to log 3 words

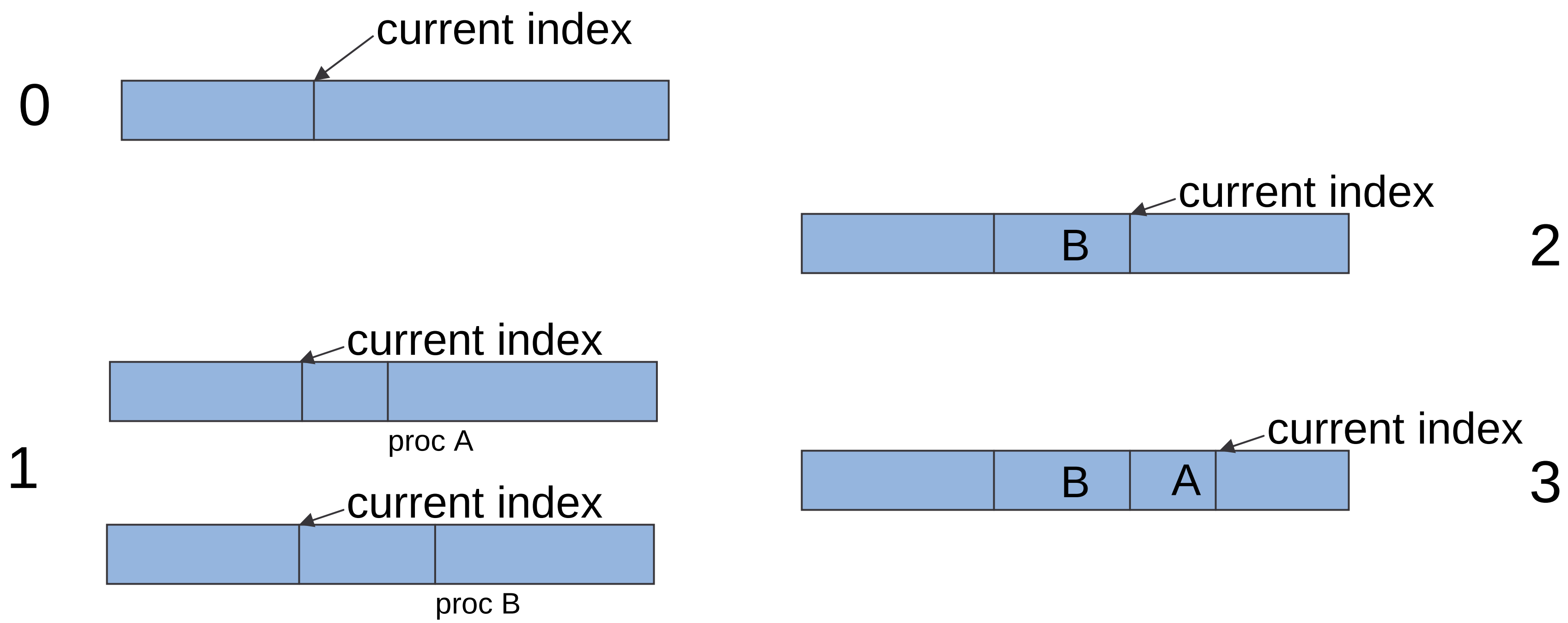
Lockless Logging



Lockless Logging



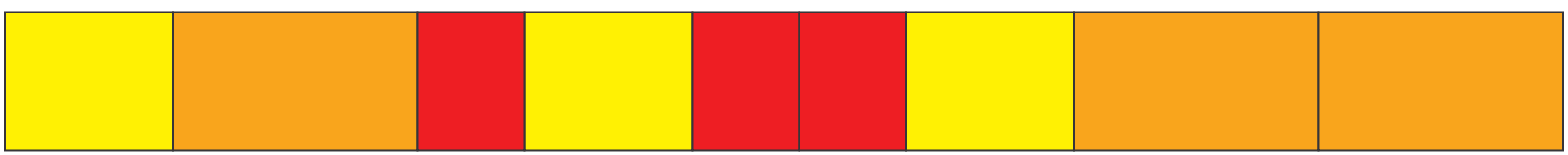
Lockless Logging



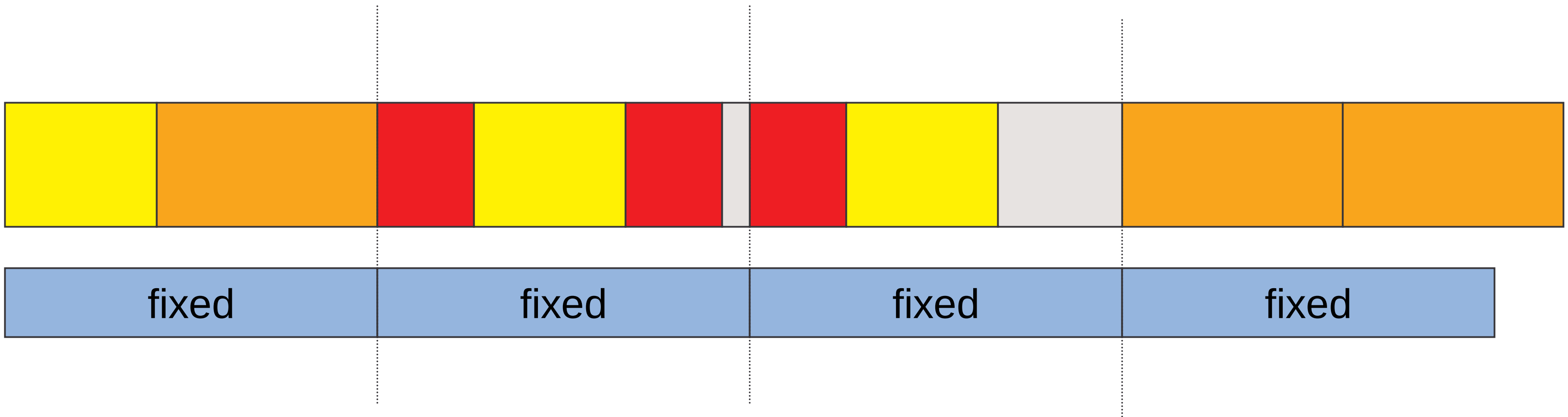
works between user, servers, kernel
potential problems – event loss etc.

Random Access
Variable Length Events

- Variable length events (vs fixed length)
  - more flexible
  - cheaper
    - space
    - time
  - easier for longer events



Lockless Logging



works for RAM and disk

Use
Event Listing

21.4747350 TRC_USER_RUN_UL_LOADER	process 6 created new process with id 7 name /shellServe
21.4747422 TRC_EXCEPTION_PGFLT	PGFLT, kernel thread 80000000c12b0f90, faultAddr 405e628,
21.4747882 TRC_EXCEPTION_PGFLT_DONE	PGFLT DONE, kernel thread 80000000c12b0f90, faultAddr 405
21.4748091 TRC_EXCEPTION_PPC_CALL	PPC CALL, commID 0
21.4748530 TRC_MEM_FCMCOM_ATCH_REG	Region 800000001022cc98 attached to FCM e100000000003f30
21.4748709 TRC_MEM_FCMCRW_CREATE	TRC_MEM_FCMCRW_CREATE ref e100000000003f90
21.4749142 TRC_EXCEPTION_PPC_RETURN	PPC RETURN, commID 600000000
21.4749247 TRC_EXCEPTION_PPC_CALL	PPC CALL, commID 0
21.4749573 TRC_MEM_REG_CREATE_FIX	Region default 10000000 created fixlen addr 113000
21.4749773 TRC_MEM_REG_DEF_INITFIXED	region default init fixed 80000000102b7c00 addr 10000000
21.4749873 TRC_MEM_ALLOC_REG_HOLD	alloc region holder addr 10000000 size 113000
21.4749962 TRC_MEM_ALLOC_REG_HOLD	alloc region holder addr 10000000 size 113000
21.4750293 TRC_MEM_FCMCOM_ATCH_REG	Region e100000000003fa0 attached to FCM e100000000003f90

The Answer To ...

Use
Fine-Grained Behavior

pid: 3d parent: 30 lpid: 163 lparent: 157
Exec:./runtest.sh /bin/rmdir

SCbrk	:	8.39/4/8	f:		p:	31.16/2
SCchild	:	338.43/4/120	f:	1041.17/80	p:	107.45/18
SCexecve	:	209.59/1/86	f:	273.20/15	p:	691.53/34
SCexit	:	13.43/1/9	f:		p:	24.19/5
SCmmap	:	53.39/4/42	f:		p:	199.94/19
SCrmdir	:	13.61/1/3	f:		p:	53.92/1
dispatcher	:	32.71/1/13	f:	87.53/7	p:	9.77/3
user	:	1718.56/27/104	f:	1304.87/76	p:	
In-process total:		2500.18/434				

---

cleanup	:	929.41/1/5	f:	p:	
fault	:	2804.31/184/186	f:	p:	
ppc	:	1274.52/93/210	f:	p:	
Ex-process total:		5008.23/401			

wall 10800.11/0

CRT::ForkChildPhase2	255.32/2
DispatcherDefault::AsyncMsgHandler	4.05/3
CRT::ForkWorker	246.10/4
COSMgrObject::CleanupDaemon	185.61/2
MPMsgMgrEnabled::ProcessMsgList	3.56/1

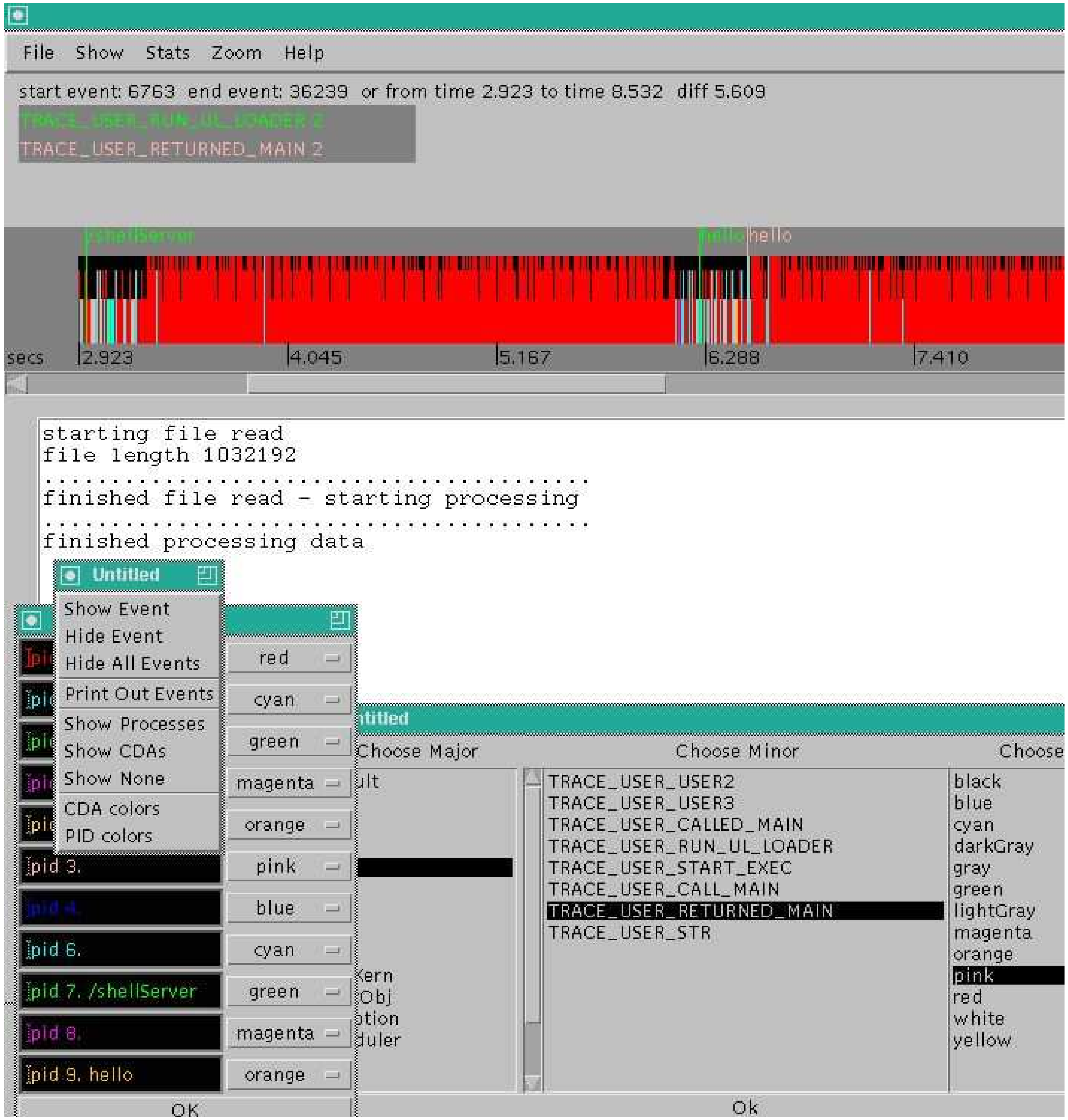
The Answer To ...

Use
Lock Contention Analysis

top 10 contended locks by time - for full list see traceLockStatsTime

time (secs)	count	spin	max time	pid
		call chain		
3.466320753	1209	188795433	0.012220087	0x1
		AllocRegionManager::alloc(unsigned PMallocDefault::pMalloc(unsigned GMalloc::gMalloc()		
0.684612632	573	37233770	0.007647854	0x0
		AllocRegionManager::alloc(unsigned PMallocDefault::pMalloc(unsigned GMalloc::gMalloc()		
0.104643241	11885	4910595	0.000322320	0x1
		PageAllocatorDefault::deallocPages(unsigned PageAllocatorUser::deallocPages(unsigned AllocPool::largeFree(void*,		

The Answer To ...



File Show Stats Zoom Help
start event: 6763 end event: 36239 or from time 2.923 to time 8.532 diff 5.609
TRACE_USER_RUN_UL_LOADER 2
TRACE_USER_RETURNED_MAIN 2
/shellServer
hello
hello
secs
2.923
4.045
5.167
6.288
7.410
starting file read
file length 1032192
finished file read - starting processing
finished processing data
Untitled
Show Event
Hide Event
Hide All Events
Print Out Events
Show Processes
Show CDAs
Show None
CDA colors
PID colors
red
cyan
green
magenta
orange
pid 3.
pink
blue
pid 6.
cyan
pid 7. /shellServer
green
pid 8.
magenta
pid 9. hello
orange
OK
Choose Major
Choose Minor
TRACE_USER_USER2
TRACE_USER_USER3
TRACE_USER_CALLED_MAIN
TRACE_USER_RUN_UL_LOADER
TRACE_USER_START_EXEC
TRACE_USER_CALL_MAIN
TRACE_USER_RETURNED_MAIN
TRACE_USER_STR
black
blue
cyan
darkGray
gray
green
lightGray
magenta
orange
pink
red
white
yellow
Ok

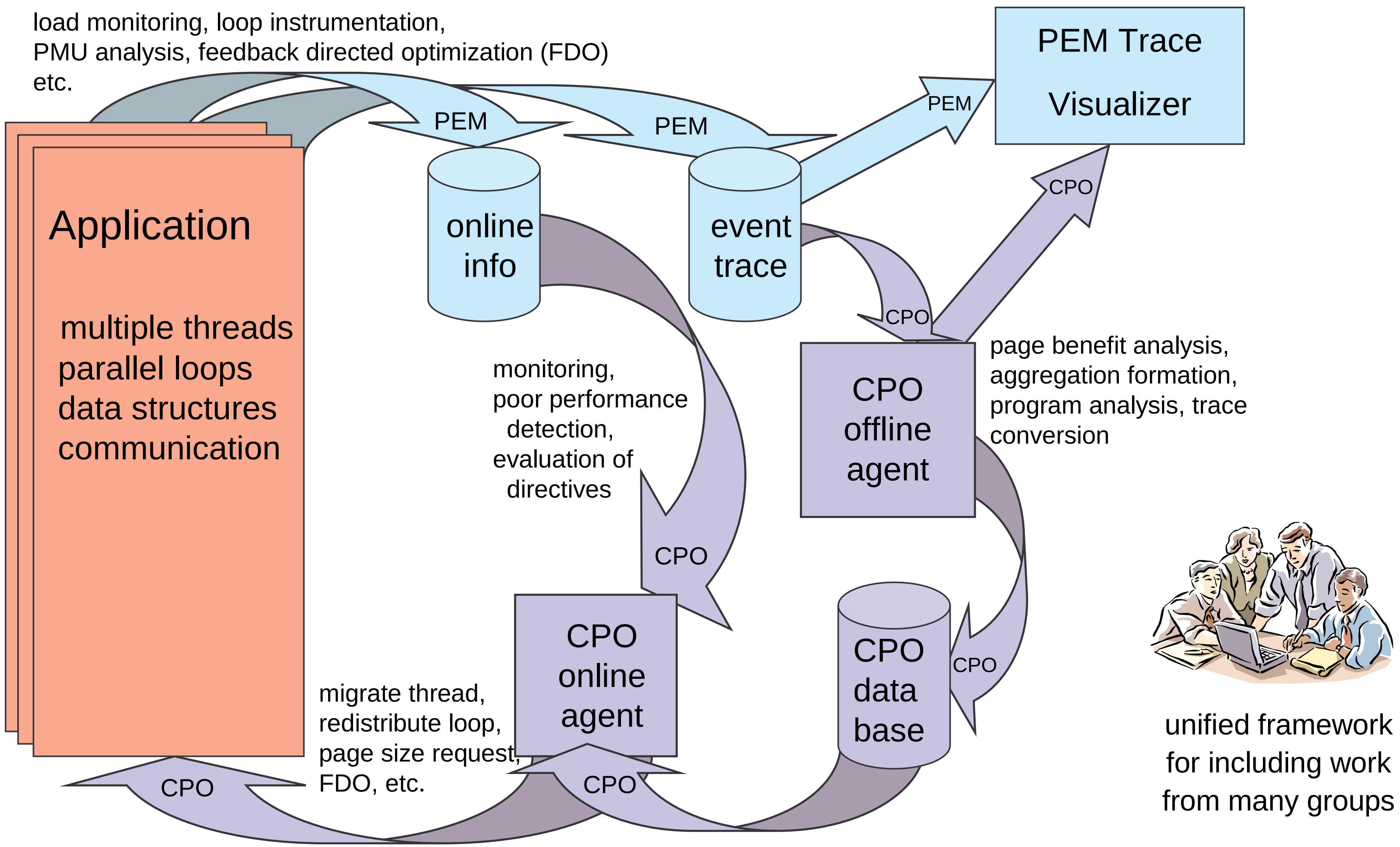
K42 Lessons

- **K42**
  - \+ **Static trace points valuable**
    - **•More efficient (94 cycles on K42)**
    - **•Modified when code is modified**
  - \+ **Separate definition files useful**
  - \+ **Breakdown into major and minor classes useful**
  - \+ **Variable length events**
  - \+ **Single unified system for events**
  - \+ **Dynamic enabling and disabling useful**
  - – **No dynamic events**
  - – **No flexibility at event time**
  - ➊ **Visualization is key**

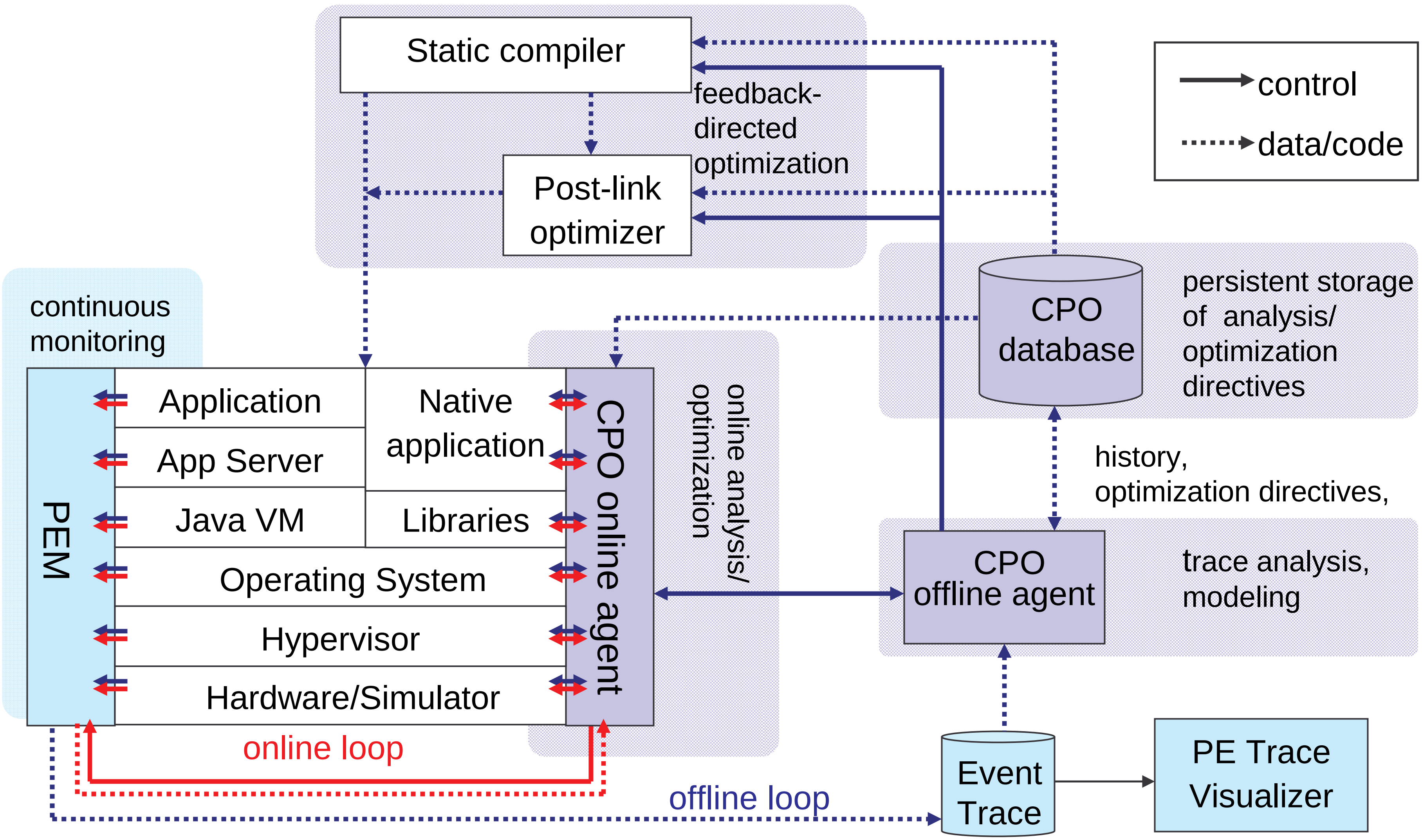
Outline

- **SGI**
  - **rtmon**
  - **Kernel and Cray Unification**
  - **Lessons**
- **K42**
  - **Approach, scalability, and use**
  - **Lessons**
- ➢ **CPO (Continuous Program Optimization)**
  - **PEM (Performance Environment Monitoring)**
  - **Lessons**
- **CSO (Commercial Scale-Out)**
  - **Goals**
  - **Lessons**
- **Blue Gene / P**
- **Observations on Linux and LTT**
- **The “next system” - concluding remarks**

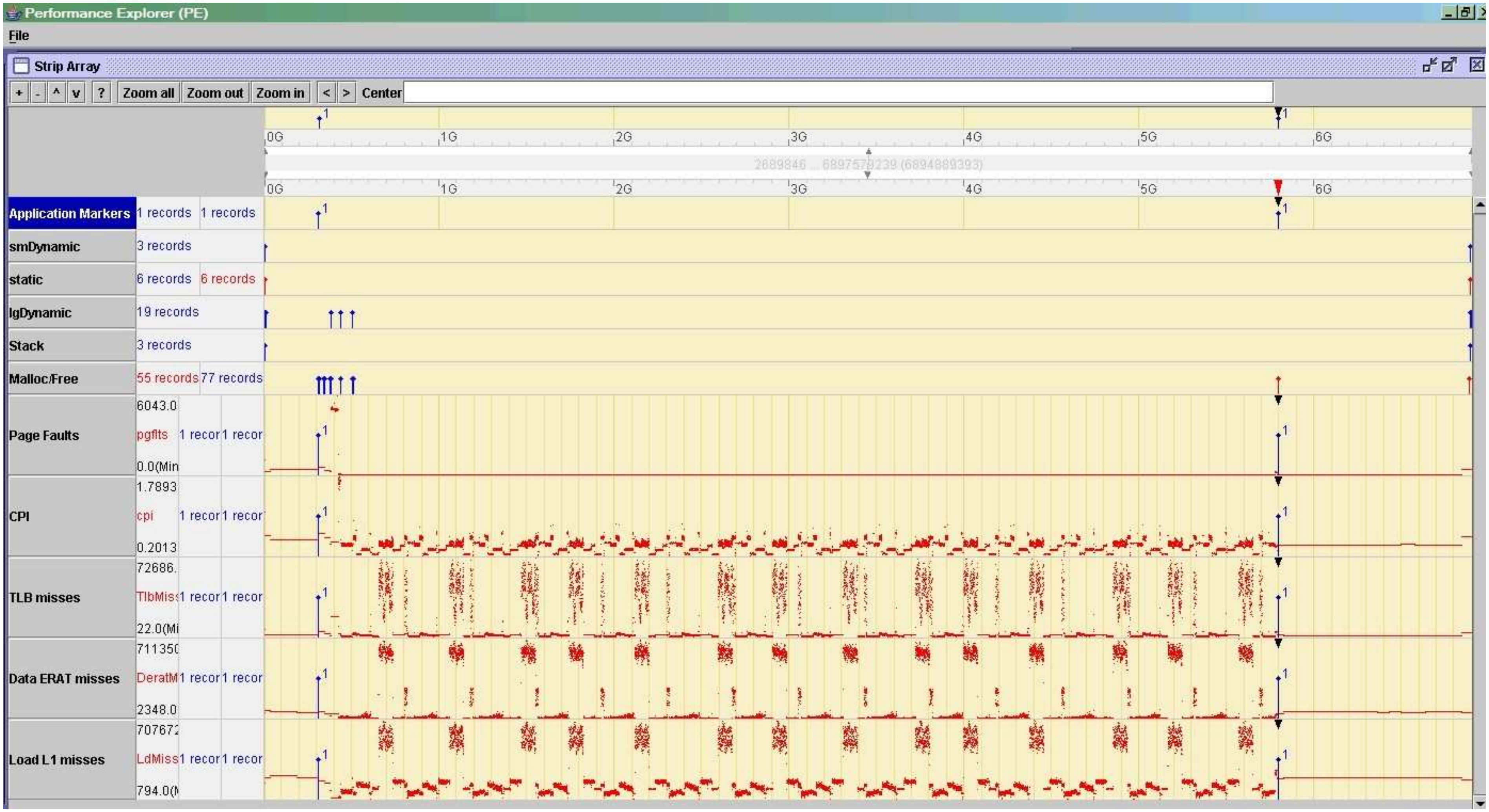
CPO Vision and Potential



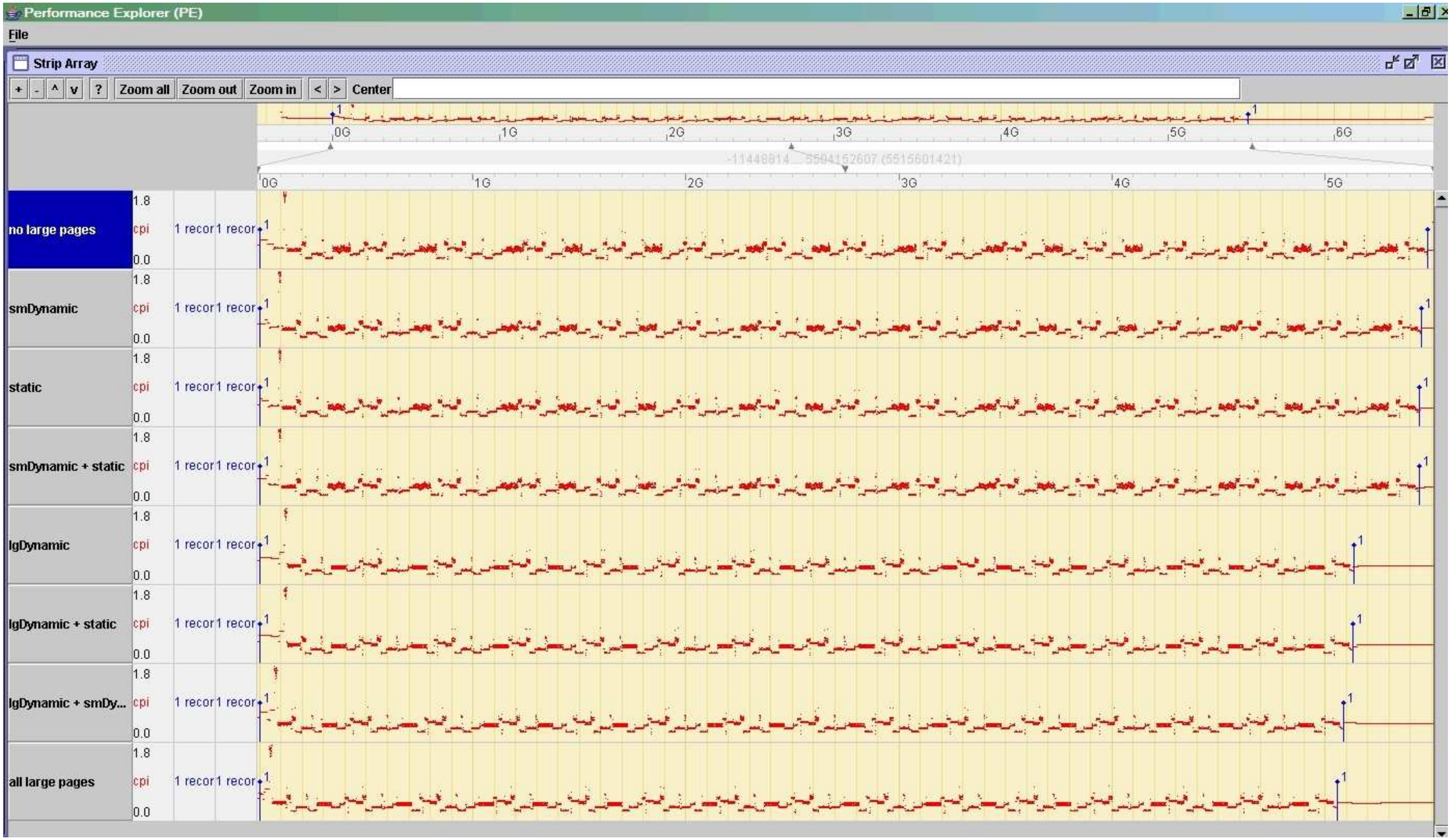
CPO Architecture



Overview of cth performance using PE



Comparison of large page mapping categories shown in PE



CPO Implementation

- **Extended K42's infrastructure**
  - **Events from a wider range of layers**
    - **•Extended notion of majors and minors to layers**
  - **Integrated HW performance counters**
  - **Self describing event definitions in XML**
  - **Extended to more than tracing, at each "event":**
    - **•Trace event**
    - **•Gather statistics on event, with tracing at threshold**
    - **•Call a handler for event**
    - **•All of the above**

CPO Lessons

- **CPO**
  - + **Vertical integration with HPCs powerful**
  - + **Addition of statistics option good for online monitoring**
  - + **Multiplexing hardware counters (ICS 05)**
  - – **No dynamic events**
  - – **No automatic packaging of trace and description files**
  - ❶ **Visualization was valuable**

Outline

- SGI
  - rtmon
  - Kernel and Cray Unification
  - Lessons
- K42
  - Approach, scalability, and use
  - Lessons
- CPO (Continuous Program Optimization)
  - PEM (Performance Environment Monitoring)
  - Lessons
- ➢ CSO (Commercial Scale-Out)
  - Goals
  - Lessons
- Blue Gene / P
- Observations on Linux and LTT
- The “next system” - concluding remarks

Introduction

- **In scientific/technical computing, parallel processing became mainstream in the 80's**
- **Since the early 90's there has been a strong move of commercial computing away from single-processor machines to multi-processor systems, as the latter became more *cost efficient***
- **Two different approaches to multiprocessors:**
  - Scale-up: large shared-memory machines
  - Scale-out: clusters of interconnected smaller machines

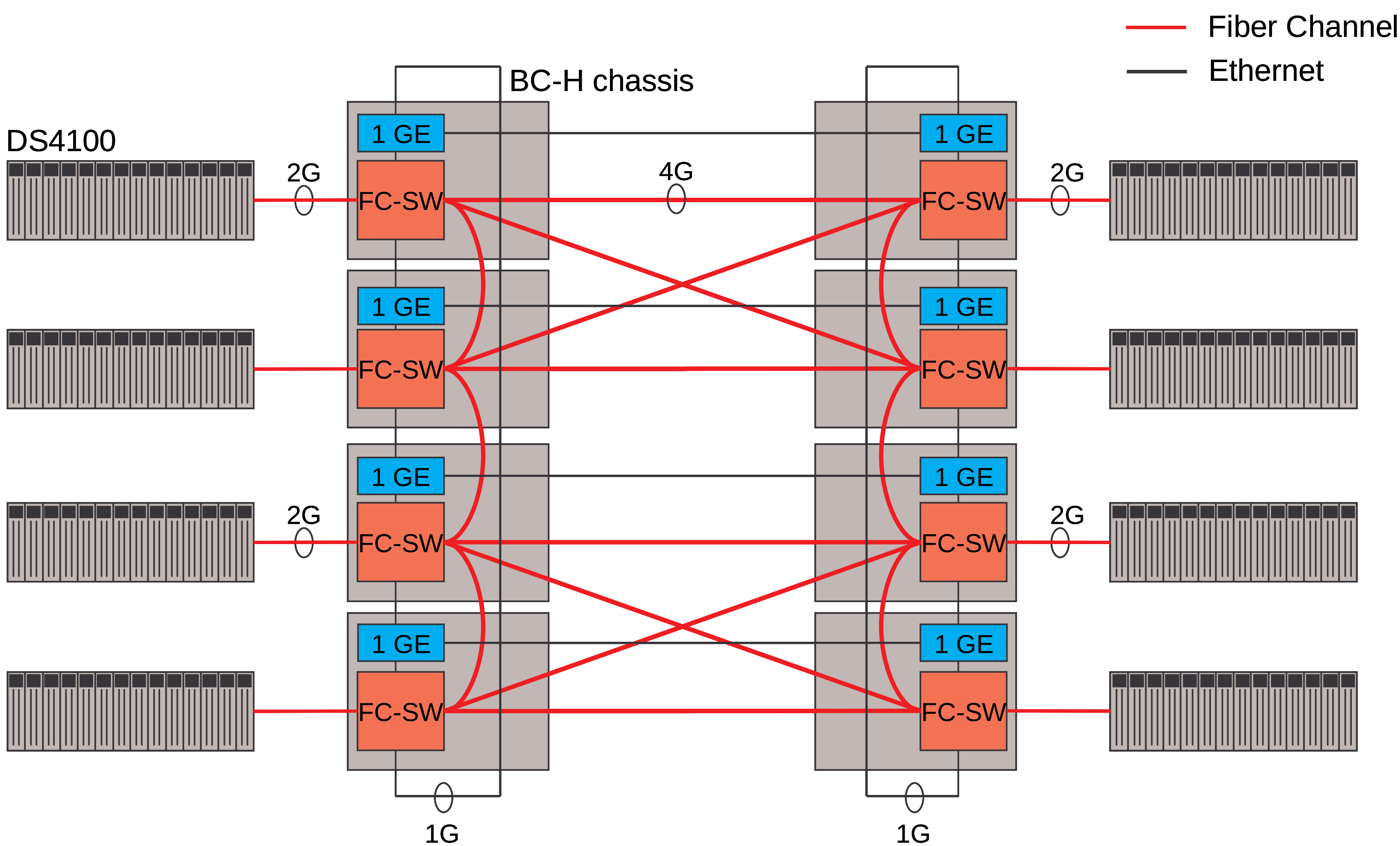
Introduction

- **In scientific/technical computing, parallel processing became mainstream in the 80's**
- **Since the early 90's there has been a strong move of commercial computing away from single-processor machines to multi-processor systems, as the latter became more *cost efficient***
- **Two different approaches to multiprocessors:**
  - Scale-up: large shared-memory machines
  - Scale-out: clusters of interconnected smaller machines

Scale-up

Scale-out

Commercial Scale Out experimental system

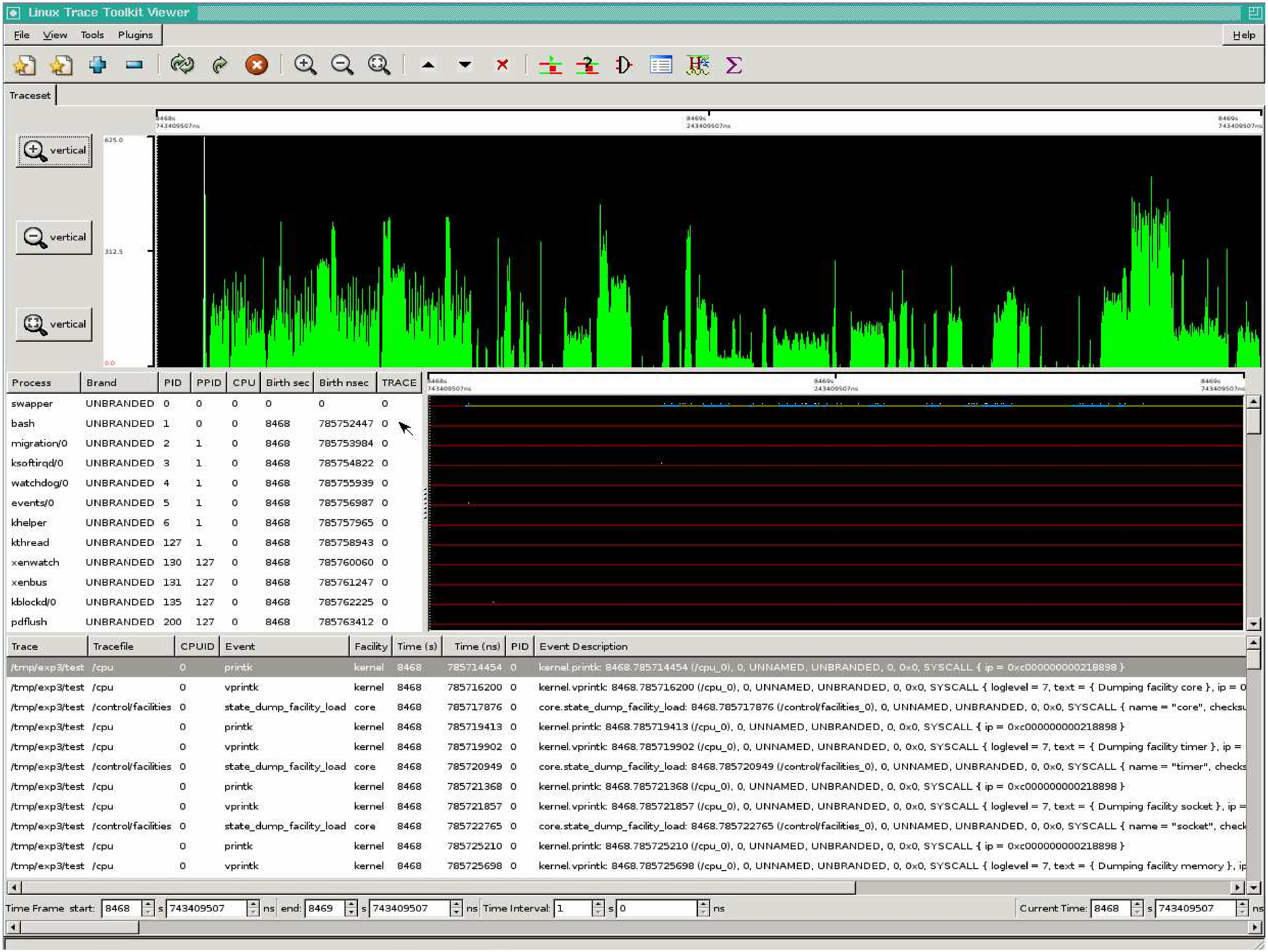


Understanding performance in commercial scale-out

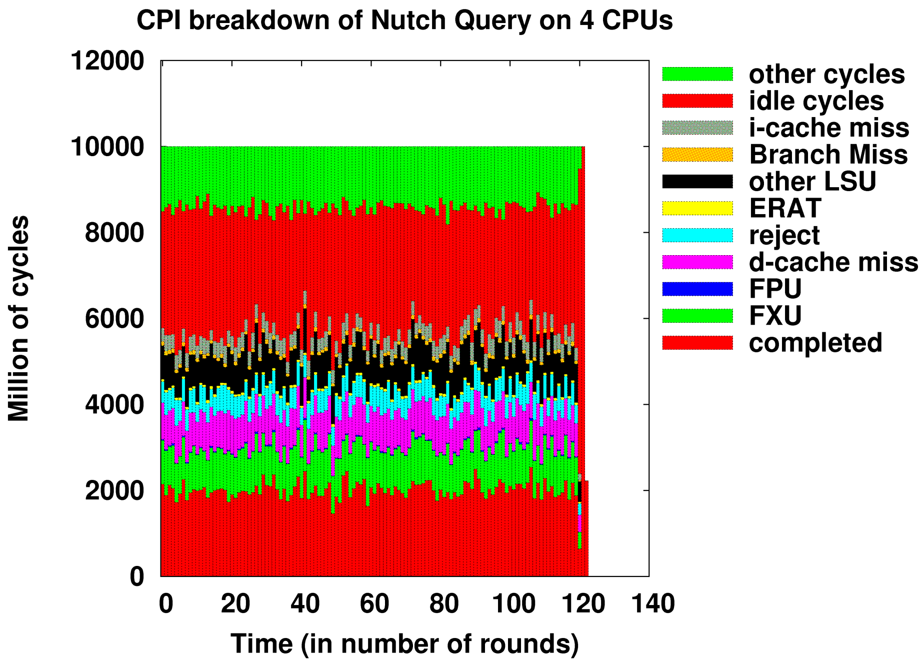
- **Two challenges similar to scientific computing:**
  - Lots of processing elements → lots of trace data: need techniques to limit data and identify important parts
  - Correlate events from different machines → need synchronized time
- **Two challenges unique to commercial:**
  - Complexity of the software stack → hypervisor, operating system, Java, middleware, application
  - Many threads of execution per processing element → multiple threads per process and multiple processes per processor – it is not unusual to see hundreds to thousands of threads per machine!

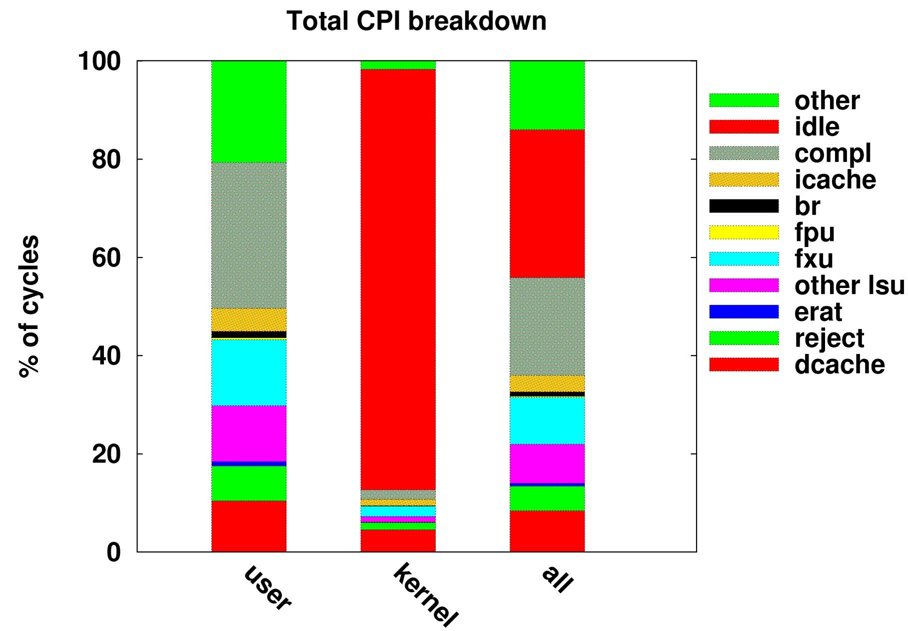
Starting point

- **Linux Trace Toolkit Next Generation (LTTng):**
  - Extracts information from hypervisor to application
  - Requires instrumentation but it is uniform across layers
  - Low overhead
- **Linux Trace Toolkit Viewer (LTTV):**
  - Merges data collected by each software layer
  - Identifies the producer of each event (node, process, thread)
  - Classifies the execution context (process, trap, interrupt, system call)
- **Enhancements to LTTng:**
  - PowerPC-specific instrumentation
  - Tracing support for Java – addition of *thread branding* (also LTTV)
- **Performance monitoring facility**
  - Uses hardware performance counters in PowerPC
  - Identified bottlenecks through statistical sampling


Linux Trace Toolkit Viewer
File View Tools Plugins
Help
Traceset
vertical
vertical
vertical
625.0
312.5
0.0
8468s 743409507ns
8469s 243409507ns
8469s 743409507ns
Process Brand PID PPID CPU Birth sec Birth nsec TRACE
swapper UNBRANDED 0 0 0 0 0 0
bash UNBRANDED 1 0 0 8468 785752447 0
migration/0 UNBRANDED 2 1 0 8468 785753984 0
ksoftirqd/0 UNBRANDED 3 1 0 8468 785754822 0
watchdog/0 UNBRANDED 4 1 0 8468 785755939 0
events/0 UNBRANDED 5 1 0 8468 785756987 0
khelper UNBRANDED 6 1 0 8468 785757965 0
kthread UNBRANDED 127 1 0 8468 785758943 0
xenwatch UNBRANDED 130 127 0 8468 785760060 0
xenbus UNBRANDED 131 127 0 8468 785761247 0
kblockd/0 UNBRANDED 135 127 0 8468 785762225 0
pdflush UNBRANDED 200 127 0 8468 785763412 0
Trace Tracefile CPUID Event Facility Time (s) Time (ns) PID Event Description
/tmp/exp3/test /cpu 0 printk kernel 8468 785714454 0 kernel.printk: 8468.785714454 (/cpu_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { ip = 0xc000000000218898 }
/tmp/exp3/test /cpu 0 vprintk kernel 8468 785716200 0 kernel.vprintk: 8468.785716200 (/cpu_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { loglevel = 7, text = { Dumping facility core }, ip = 0
/tmp/exp3/test /control/facilities 0 state_dump_facility_load core 8468 785717876 0 core.state_dump_facility_load: 8468.785717876 (/control/facilities_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { name = "core", checksu
/tmp/exp3/test /cpu 0 printk kernel 8468 785719413 0 kernel.printk: 8468.785719413 (/cpu_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { ip = 0xc000000000218898 }
/tmp/exp3/test /cpu 0 vprintk kernel 8468 785719902 0 kernel.vprintk: 8468.785719902 (/cpu_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { loglevel = 7, text = { Dumping facility timer }, ip =
/tmp/exp3/test /control/facilities 0 state_dump_facility_load core 8468 785720949 0 core.state_dump_facility_load: 8468.785720949 (/control/facilities_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { name = "timer", checks
/tmp/exp3/test /cpu 0 printk kernel 8468 785721368 0 kernel.printk: 8468.785721368 (/cpu_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { ip = 0xc000000000218898 }
/tmp/exp3/test /cpu 0 vprintk kernel 8468 785721857 0 kernel.vprintk: 8468.785721857 (/cpu_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { loglevel = 7, text = { Dumping facility socket }, ip =
/tmp/exp3/test /control/facilities 0 state_dump_facility_load core 8468 785722765 0 core.state_dump_facility_load: 8468.785722765 (/control/facilities_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { name = "socket", check
/tmp/exp3/test /cpu 0 printk kernel 8468 785725210 0 kernel.printk: 8468.785725210 (/cpu_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { ip = 0xc000000000218898 }
/tmp/exp3/test /cpu 0 vprintk kernel 8468 785725698 0 kernel.vprintk: 8468.785725698 (/cpu_0), 0, UNNAMED, UNBRANDED, 0, 0x0, SYSCALL { loglevel = 7, text = { Dumping facility memory }, ip
Time Frame start: 8468 s 743409507 ns end: 8469 s 743409507 ns Time Interval: 1 s 0 ns
Current Time: 8468 s 743409507 ns

Stall breakdown





- ~2 billion completing cycles/sec (20% of total 10 billion)
- 6 billion instructions/sec
- Non-stall CPI (CPI_C): 0.34 Average for SPECcpu 2000: 0.35
- Average bundle size: 3

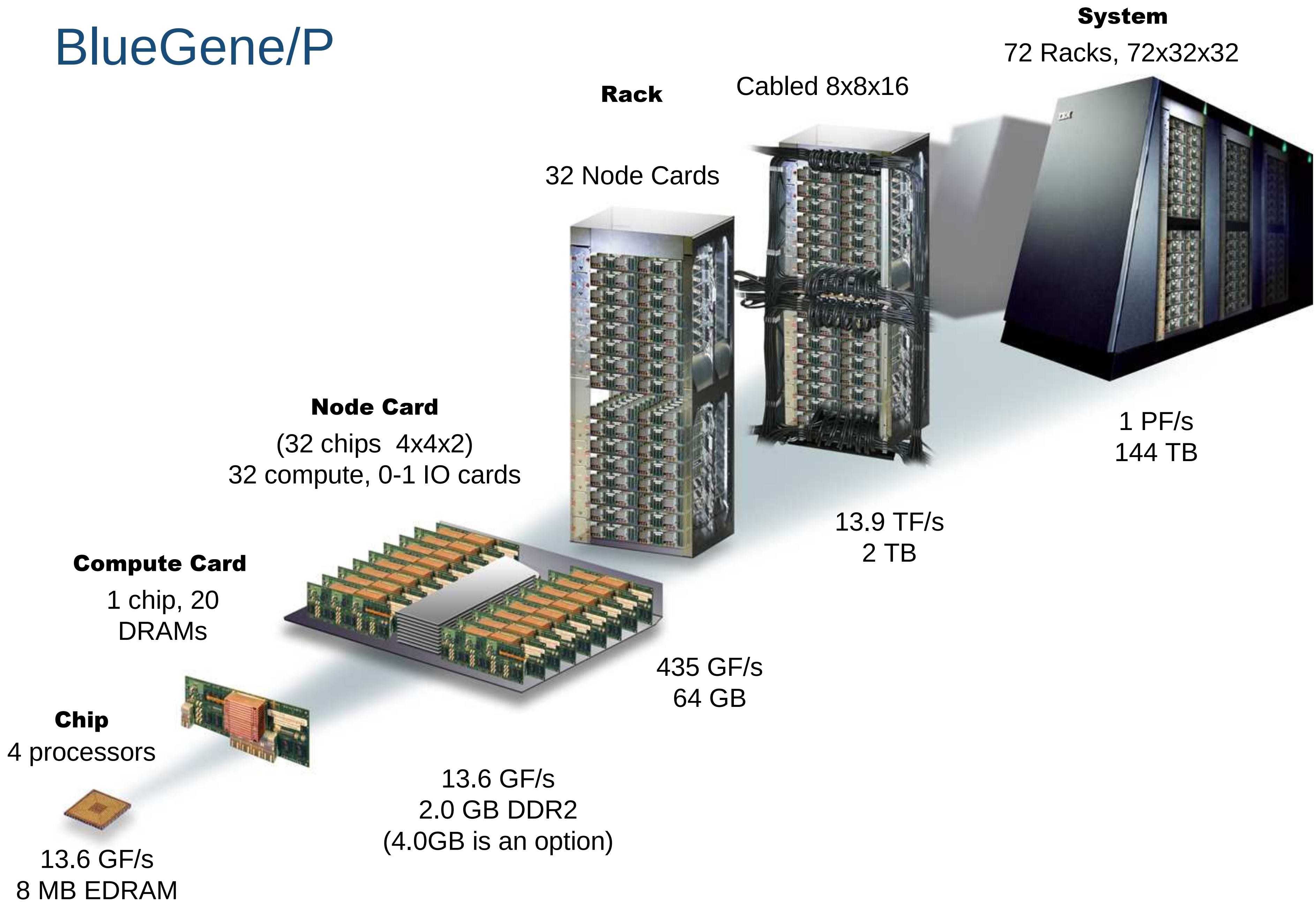
CSO Lessons

- **CSO**
  - \+ **Tracing useful**
  - \+ **HPCs useful**
  - – **Performance monitoring for distributed commercial workloads needs more work**
    - **•Handling many small, in terms of CPU usage, tasks**
    - **•Automatic process branding**
    - **•Inter-machine timer synchronization**
    - **•Automatic idle determination**
    - **•Cross machine logical causality**
    - **•Tree-based causality**
    - **•Selective aggregation of performance data**
    - **•Virtualization**

Outline

- **SGI**
  - **rtmon**
  - **Kernel and Cray Unification**
  - **Lessons**
- **K42**
  - **Approach, scalability, and use**
  - **Lessons**
- **CPO (Continuous Program Optimization)**
  - **PEM (Performance Environment Monitoring)**
  - **Lessons**
- **CSO (Commercial Scale-Out)**
  - **Goals**
  - **Lessons**
- ➢ **Blue Gene / P**
- **Observations on Linux and LTT**
- **The “next system” - concluding remarks**

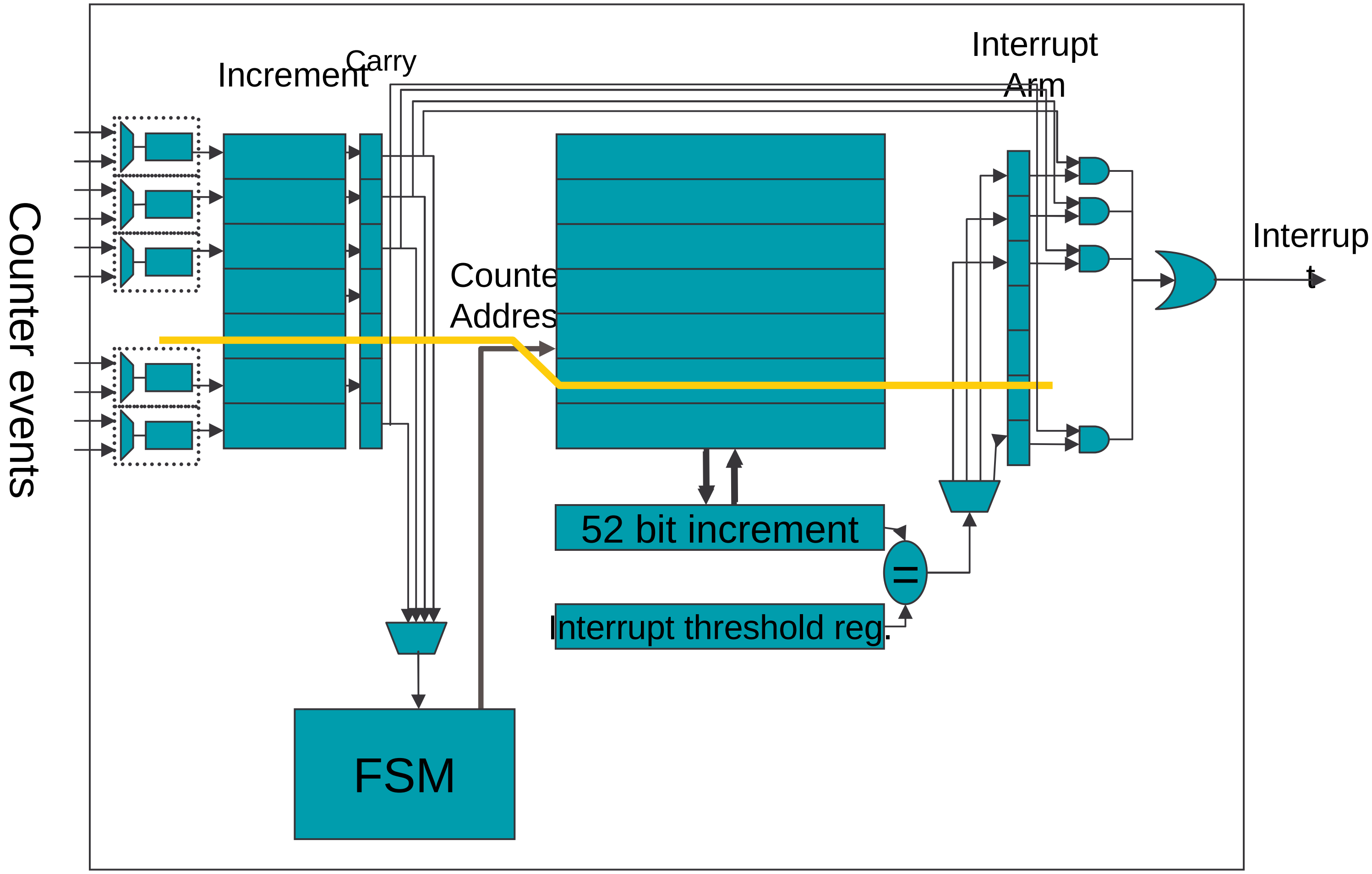
BlueGene/P


System
72 Racks, 72x32x32
Rack
Cabled 8x8x16
32 Node Cards
Node Card
(32 chips 4x4x2)
32 compute, 0-1 IO cards
Compute Card
1 chip, 20
DRAMs
Chip
4 processors
13.6 GF/s
8 MB EDRAM
13.6 GF/s
2.0 GB DDR2
(4.0GB is an option)
435 GF/s
64 GB
13.9 TF/s
2 TB
1 PF/s
144 TB

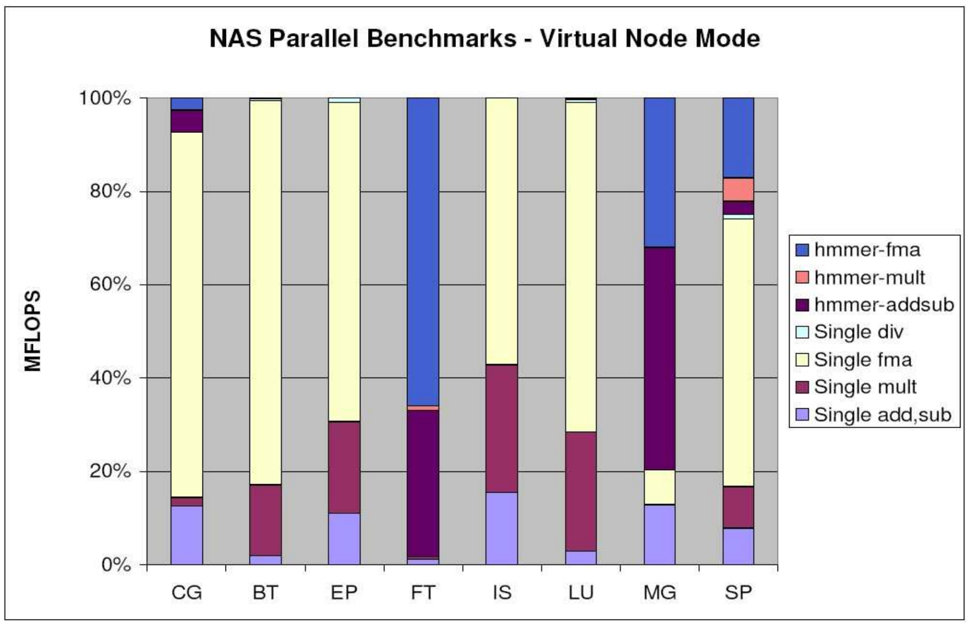
Performance monitoring unit in the Blue Gene/P system

- Implements 256 counters, 64bits wide
  - 1024 possible counter events
  - Monitors 4 processor cores and FPU, L3, L2, snoop filters, torus and collective network
- Novel architecture
  - Hybrid implementation using SRAM arrays
  - High density, high capacity on-chip performance monitor unit
- Hybrid architecture
  - 12 low order bits of a counter implemented using discrete logic
  - 52 high order bits stored in an SRAM array
  - SRAM state updated at a regular basis under state machine control
  - Configurable input selection and interrupt
  - Interrupt indication when the threshold value is reached

Hybrid PMU architecture



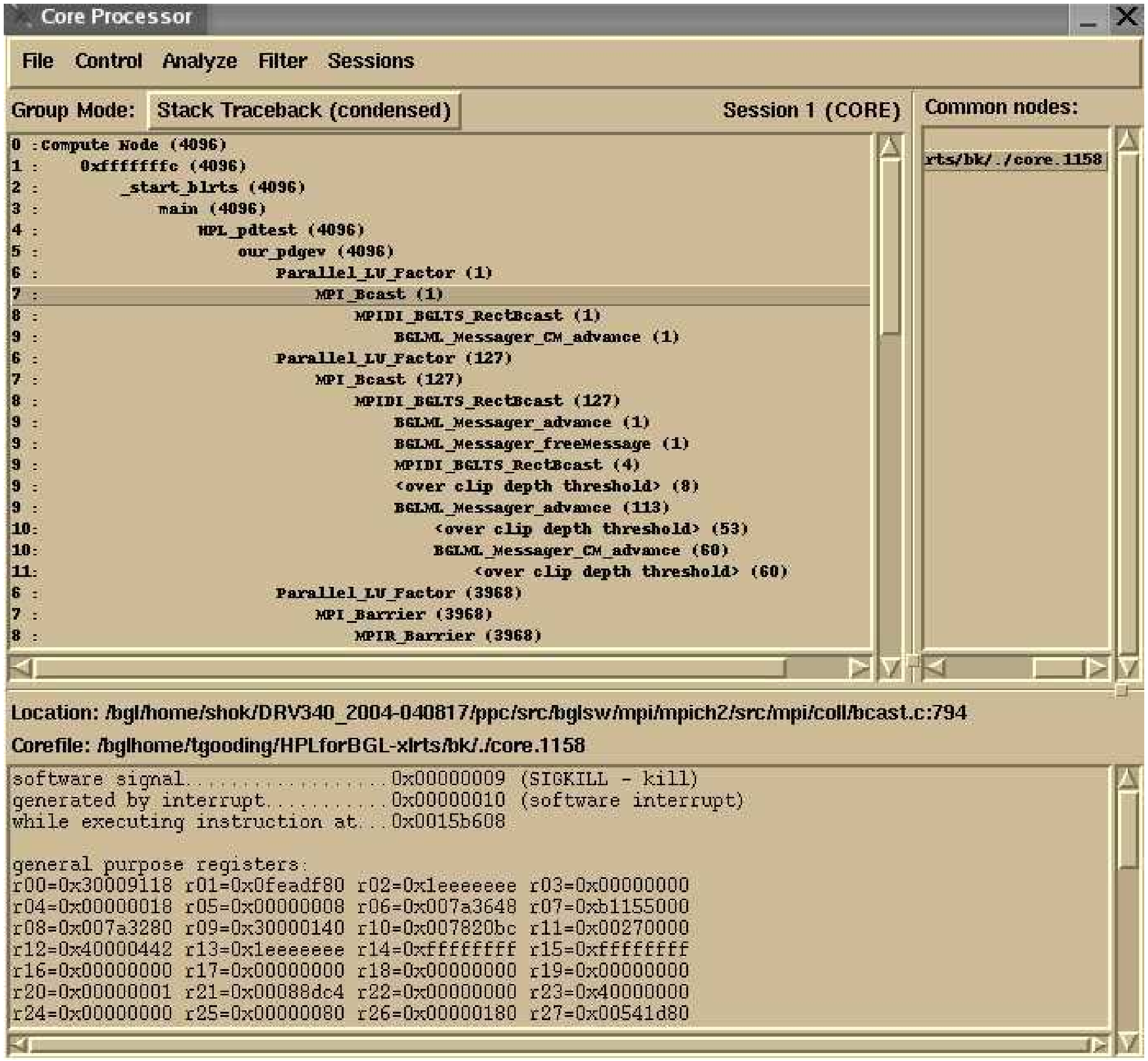
Usage of PMU in BGP



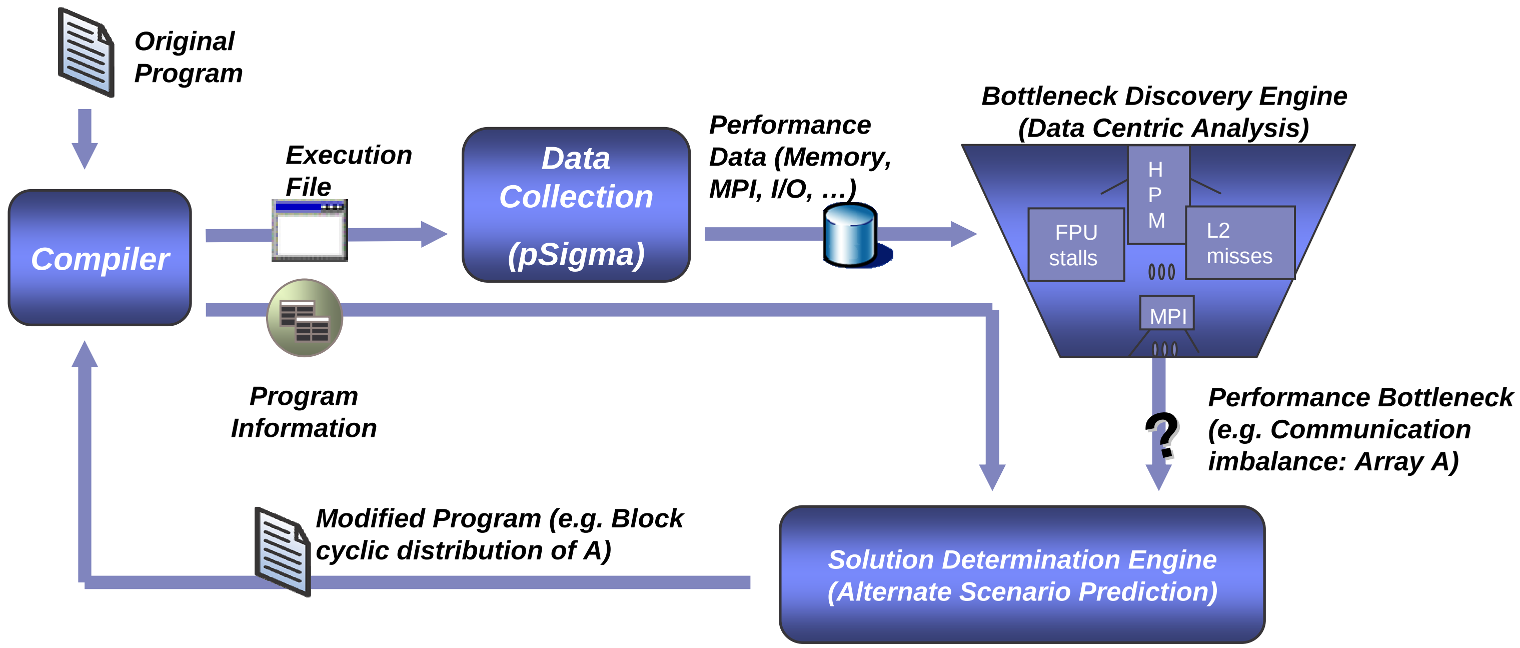
Breakdown of FP operations

- Opens countless possibilities – some usage examples
  - Analyze the execution profile for different compiler optimizations and infer their effectiveness
  - Conclude on the effectiveness of the various hardware & software settings to determine the optimal configuration
  - Profile and characterize workloads for various modes of operation to achieve maximum performance on multiple cores

Coreprocessor showing program counter on 4 racks



Data and Control Flow of HPCS Toolkit



HPCS Toolkit provides Autonomic Application Performance Capability.

- Intelligent automation of performance evaluation and decision system
- Interactive capability with graphical/visual interface always available, but always optional

Outline

- **SGI**
  - **rtmon**
  - **Kernel and Cray Unification**
  - **Lessons**
- **K42**
  - **Approach, scalability, and use**
  - **Lessons**
- **CPO (Continuous Program Optimization)**
  - **PEM (Performance Environment Monitoring)**
  - **Lessons**
- **CSO (Commercial Scale-Out)**
  - **Goals**
  - **Lessons**
- **Blue Gene / P**
- ➢ **Observations on Linux and LTT**
- **The "next system" - concluding remarks**

What is right for Linux and How

- **Patches** versus **dynamic points** versus **markers** versus **static**
- **One infrastructure** versus **many**
- **Get performance monitoring community active on lkml**
- **Get nose in tent**

Outline

- **SGI**
  - **rtmon**
  - **Kernel and Cray Unification**
  - **Lessons**
- **K42**
  - **Approach, scalability, and use**
  - **Lessons**
- **CPO (Continuous Program Optimization)**
  - **PEM (Performance Environment Monitoring)**
  - **Lessons**
- **CSO (Commercial Scale-Out)**
  - **Goals**
  - **Lessons**
- **Blue Gene / P**
- **Observations on Linux and LTT**
- ➢ **The "next system" - concluding remarks**

The ~~Final~~ Next System

- **Efficient, Flexible tracing**
  - **Single unified space over all layers including HW counters**
  - **Use static events or event markers**
  - **Enable system to trace, gather stats, or callback at event**
  - **Allow additional dynamic events**
  - **Break into categories and allow dynamic enabling**
  - **Provide automatic tool for packaging up data and description**
  - **Timer synchronization built into infrastructure**
  - **Variable sized events**
  - **Non-locking and scalable gathering**
  - **Efficient online gathering for more extensive offline analysis**
  - **Negligible impact when disabled**

The ~~Final~~ Next System

- **Configurable visualization**
  - **Ability to add new graphs and have system save view**
  - **Pluggable modules to interpret application-specific events**
  - **Ability to handle massive (100G +) trace data**
    - **Quick start up**
    - **Summary and stats information on selectable portion**
  - **Handle multicore, multiprocessor, and distributed data**
  - **Handle real-time, scientific, and commercial data**
  - **Lots of interesting work left to understand commercial systems**
  - **Nice default views**
    - **Time-centric time by process, thread-centric view, statistics, histogram, event list**