Exceptional service in the national interest

Sandia National Laboratories

Evaluating Operating System Vulnerability to Memory Errors

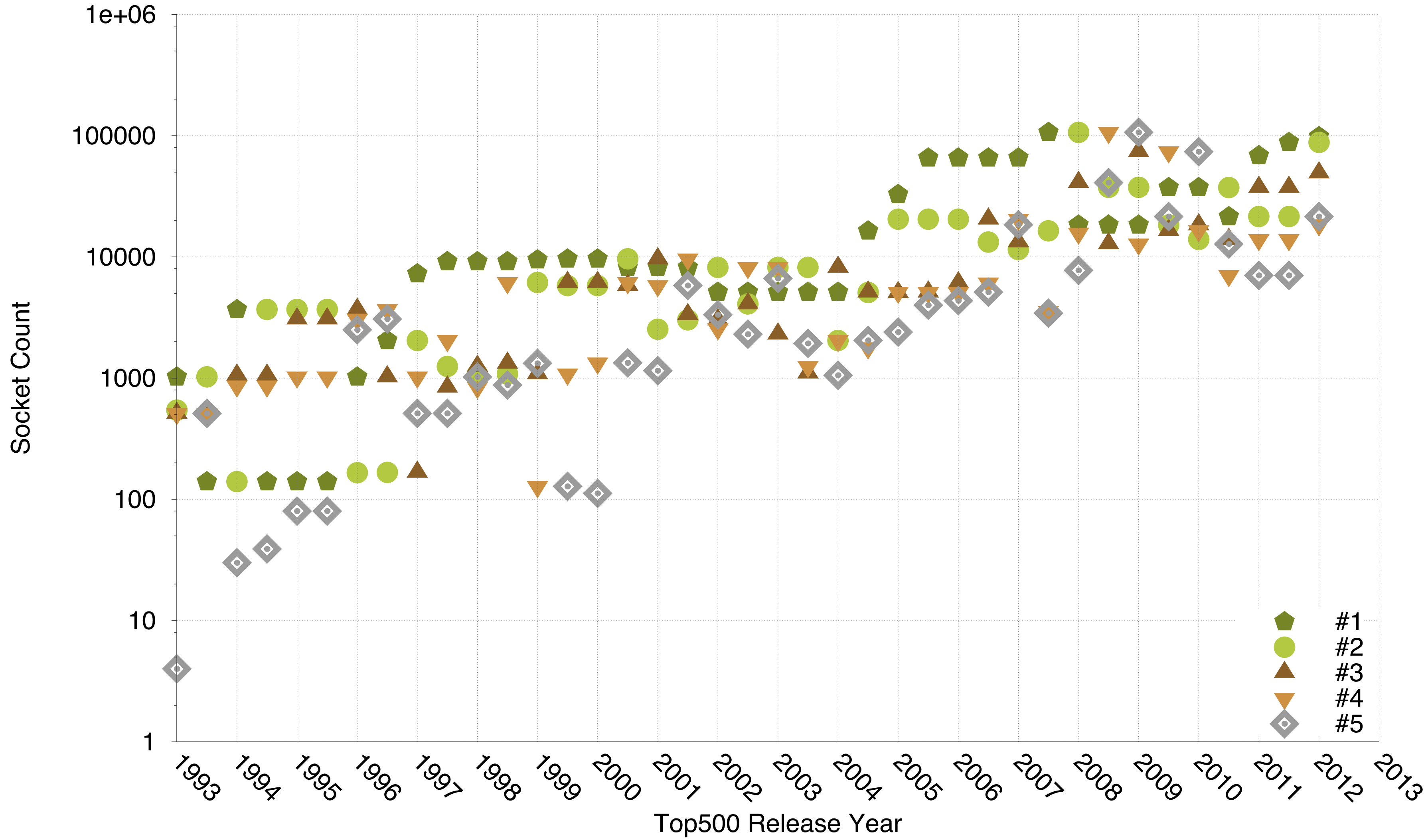
Kurt B. Ferreira, Kevin Pedretti, Patrick Bridges, **Ron Brightwell,** David Fiala, and Frank Mueller

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

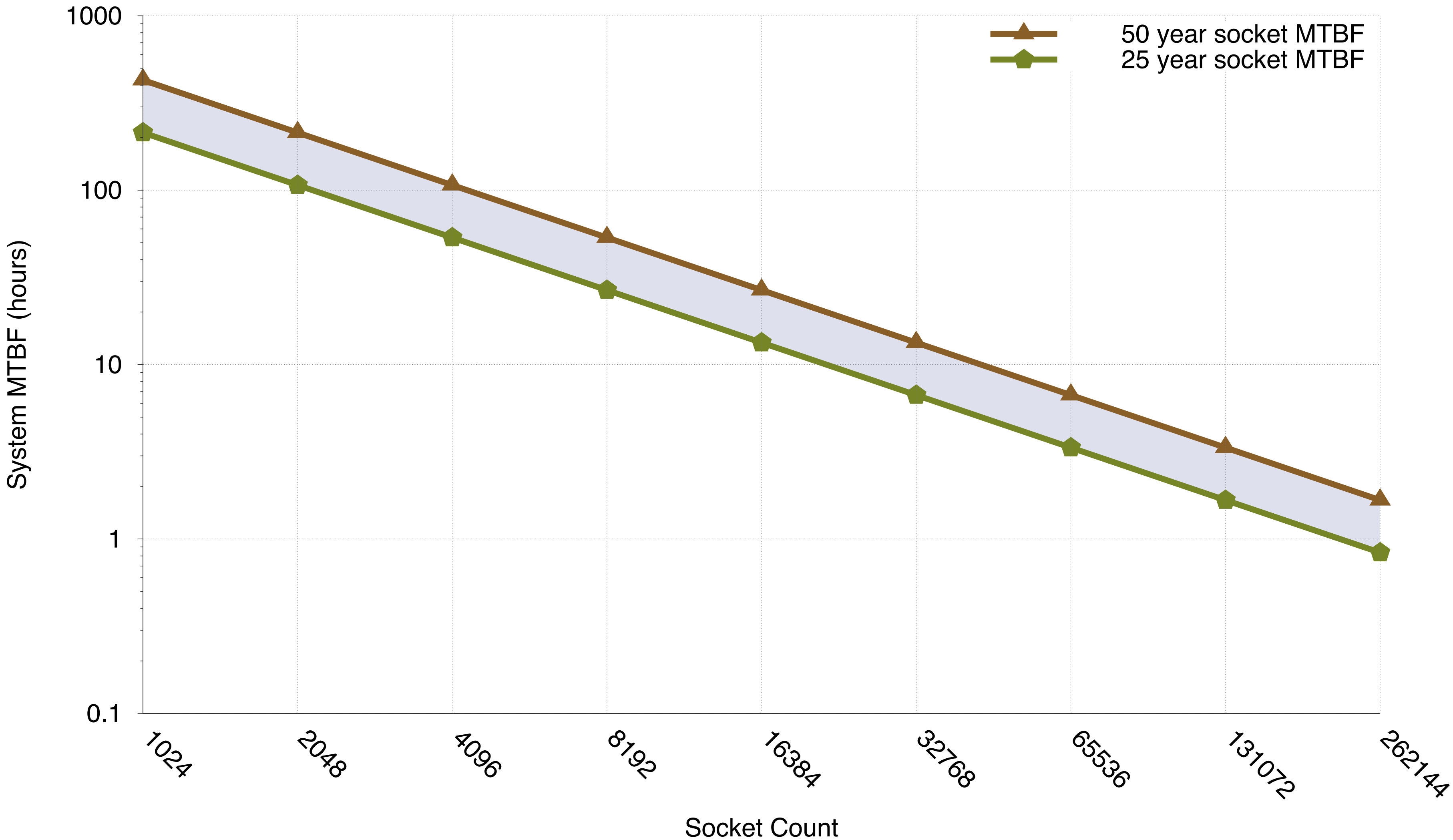
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

The Fastest Machines Are Getting Larger

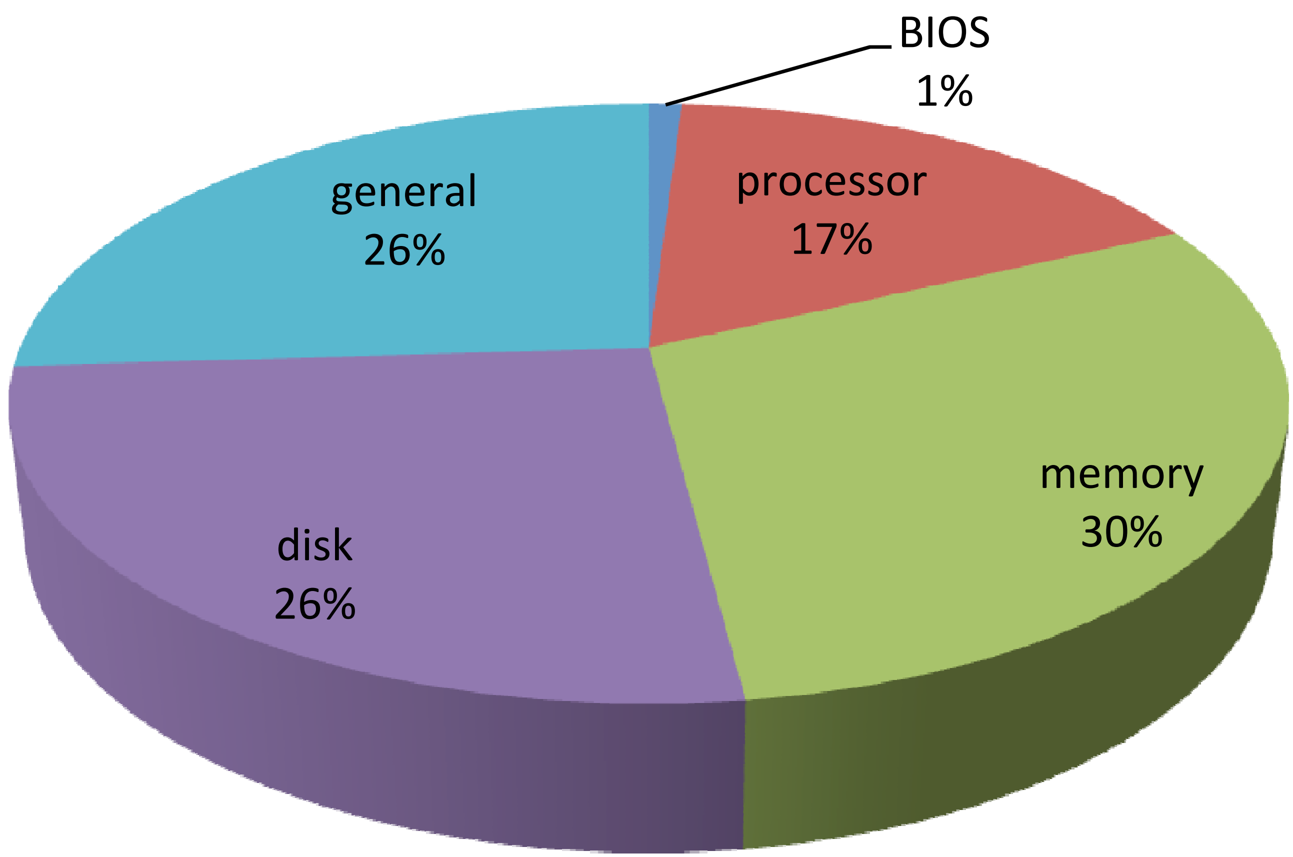
Sandia National Laboratories



… And Failures Are Becoming More Common

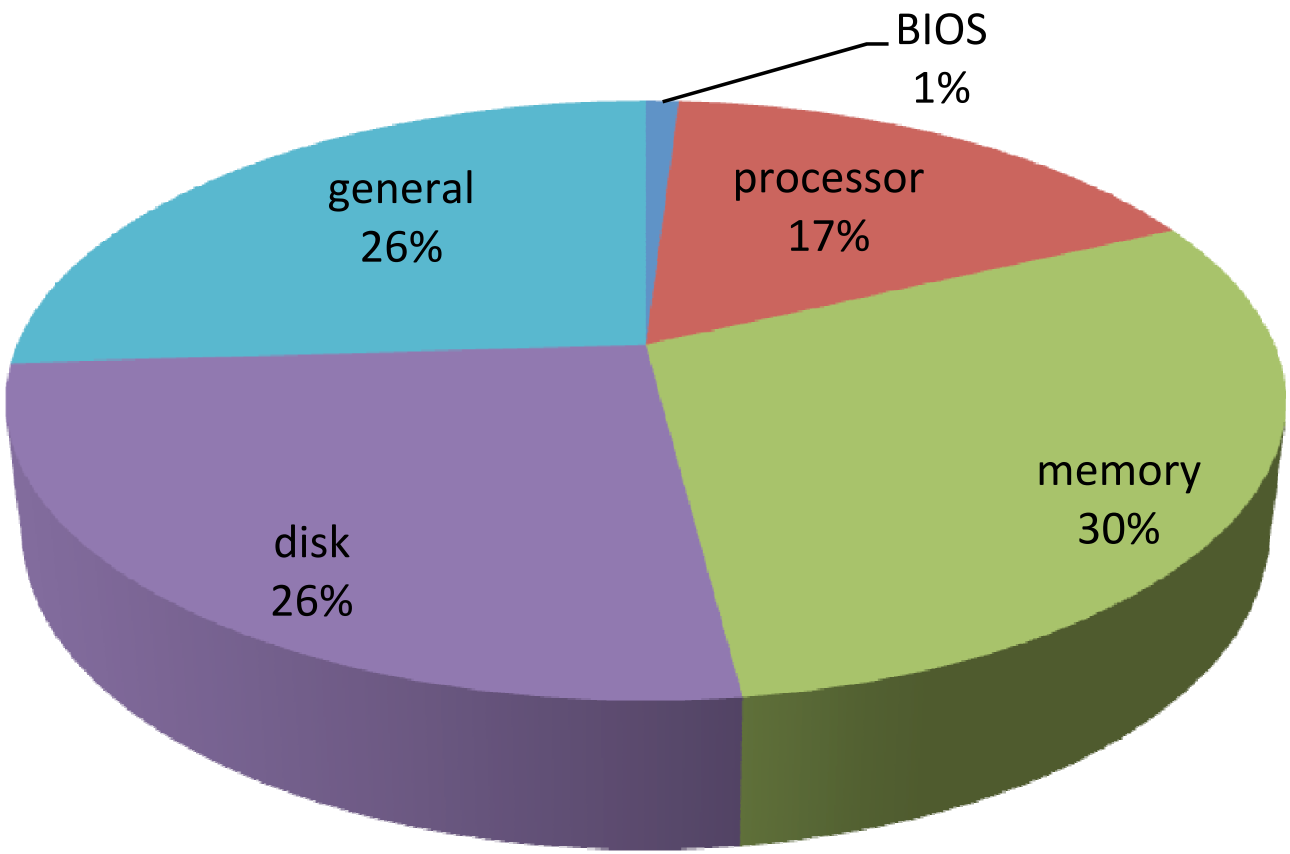


Most Common Hardware Failure Due to Memory



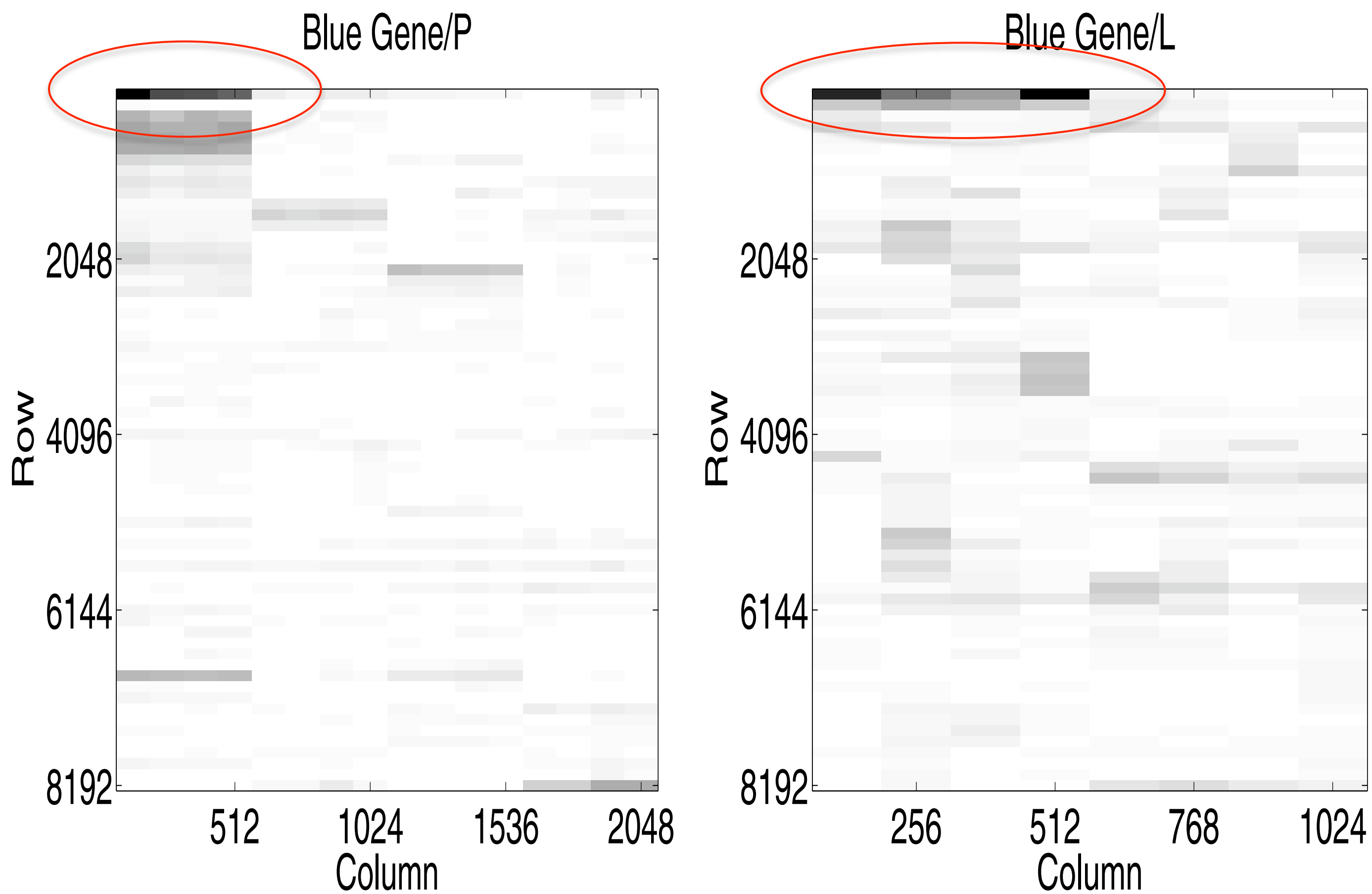
From “Automating Software Failure Reporting” by B. Murphy, *ACM Queue 2*, 2004

Most Common Hardware Failure Due to Memory



Memory failures expected to increase in future systems

Errors in OS Memory Shown To Be More Likely



From "Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors And the Implications for System Design" by A. Hwang et al, from *ASPLOS*'12

Current OS Handling of DRAM Failures

- Current OSs hard reboot for DRAM failures in OS memory

- If we can correct this error, we can avoid a restart and make forward progress

Our Goal

Construct an operating system/runtime resilient to soft errors.

○ The resilient OS/runtime key to the scalability of extreme-scale systems

○ A hardened OS/runtime can continue in the presence of failures

○ A resilient OS and runtime critical to the emerging forward-error recovery methods

○ These forward-error recovery methods are predicted to have lower overheads and avoid the wasted computation of current rollback/recovery methods

DRAM Failures: A Quick Refresher

Sandia National Laboratories

- DRAM becoming largest portion of machines power budget

- Prevalence of errors in HPC due to number of DRAM modules (10K – 100K in current systems)

- Susceptibility due to a combination of quantity and density

- Power optimizations, for example decreased supply voltages and increased density, will further increase error rates

- Types of errors: single-bit correctable, double-bit detectable, multi-bit undetectable errors

- Uncorrectable errors becoming increasing common – 8% of all DIMMS/year on current systems

Current State of the Practice

Sandia National Laboratories

- Single-symbol Error Correction and Double-symbol Error Detection (SEC-DED) but at a cost of increased energy and reduced performance
- Similarly for Chipkill which tolerates an entire DRAM chip failure
- Errors result in a Machine Check Exception (MCE) which logs the error and kills the application which memory belongs to
- Failures that occur in the OS typically result in a hard-reboot
- Common resilience methods (i.e. rollback recovery) focus only on application and not the OS

Our Approach: Compare Vulnerability of Linux and Kitten LWK

Sandia National Laboratories

○ Comparing two OSs similar to ones likely seen on an exascale-class system; Kitten and CLE

○ Purpose of this evaluation is to look at the ease of protecting each respective OS to DRAM failure

○ This initial study looks at a number of factors:

- ○ Code complexity. Metric: Source Lines Of Code (SLOC) count
- ○ Memory footprint of the OS while running an HPC application (LAMMPS)
- ○ Indentifying representative critical state that must be protected

○ These factors used to suggest failure mitigation mechanism. For example, small, simple state may effectively protected with redundancy

Kitten: A Next Generation Lightweight Kernel

- A special-purpose, limited functionality OS developed at Sandia
- Latest in SUNMOS, Puma, Cougar, Catamount line of lightweight kernels
- Code base derived from Linux, therefore ABI compatible
- Compute node kernel providing only that functionality needed for a set of mission critical HPC applications
- Functionality not needed in kernel pushed into user space
- Full-featured guest OS can be loaded on-demand via the Palacios virtual machine monitor

Open Source: Kitten (*http://code.google.com/p/kitten*),
Palacios (*http://www.v3vee.org/palacios*)

CLE: The Cray Linux Environment

○ Cray's scalable OS for the XT and XE line of supercomputers

○ Stripped-down kernels run on compute nodes while full featured kernels run on the IO nodes

○ Both kernels based on the Linux general purpose OS

○ Compute node optimizations include:

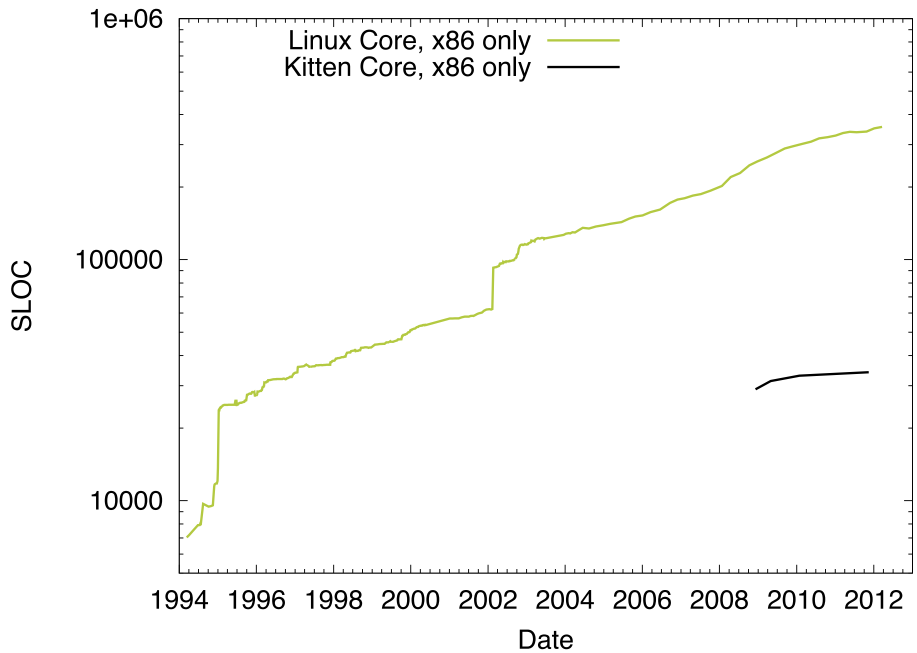
  ○ Enhancements to memory management

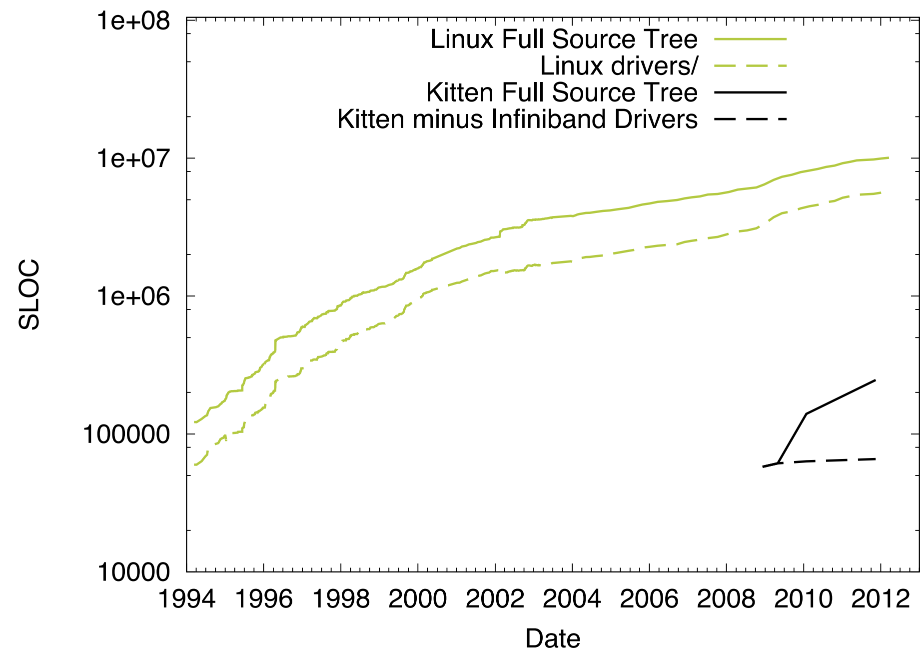
  ○ Improved out-of-memory handling

  ○ Modifications for decreased OS "jitter"

  ○ Forwarding of IO to full-featured nodes

Kitten Significantly Less Complex Than CLE



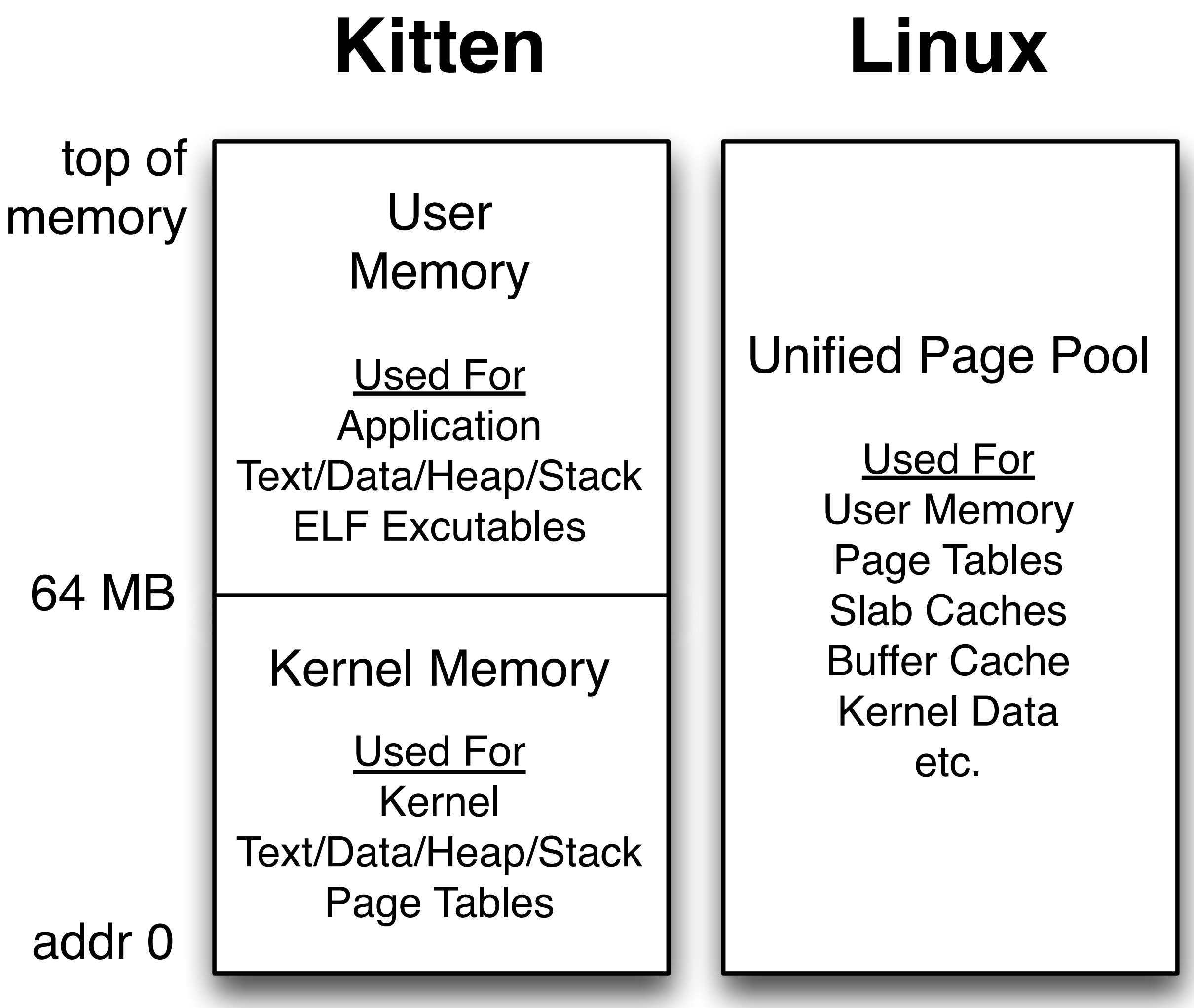


Kitten Simplicity Also Seen With OS Task Structure

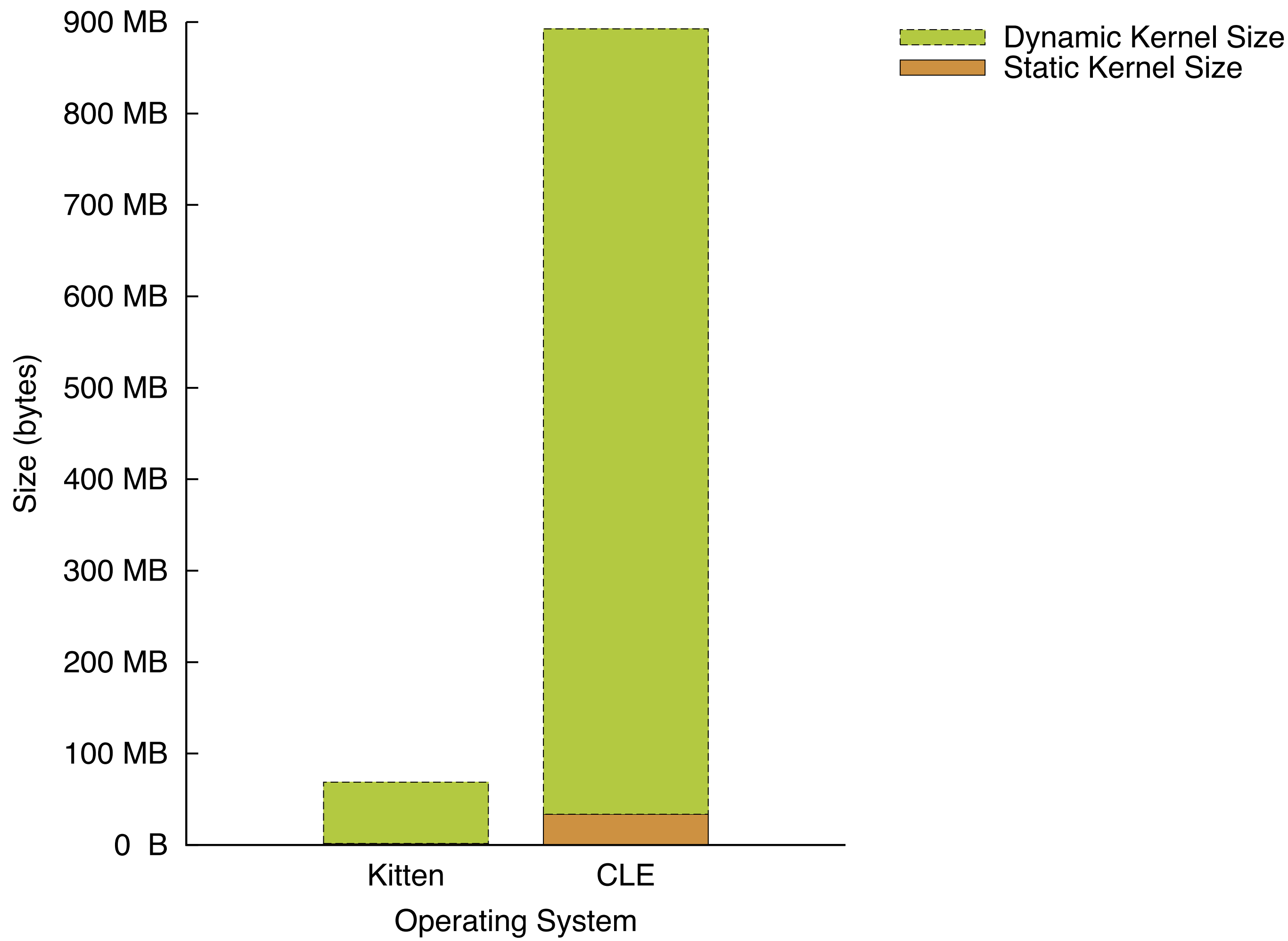
Linux: task_struct
volatile long state;
void *stack;
atomic_t usage;
unsigned int flags;
unsigned int ptrace;
struct llist_node wake_entry;
int on_cpu;
int on_rq;
int prio, static_prio, normal_prio;
unsigned int rt_priority;
const struct sched_class *sched_class;
struct sched_entity se;
struct sched_rt_entity rt;
struct hlist_head preempt_notifiers;
unsigned char fpu_counter;
unsigned int btrace_seq;
unsigned int policy;
cpumask_t cpus_allowed;
int rcu_read_lock_nesting;
char rcu_read_unlock_special;
struct list_head rcu_node_entry;
struct rcu_node *rcu_blocked_node;
struct rt_mutex *rcu_boost_mutex;
struct sched_info sched_info;
struct list_head tasks;
struct plist_node pushable_tasks;
struct mm_struct *mm, *active_mm;
unsigned brk_randomized:1;
struct task_rss_stat
int exit_state;
int exit_code, exit_signal;
int pdeath_signal;
unsigned int jobctl;
unsigned int personality;
unsigned did_exec:1;
unsigned in_execve:1;
unsigned sched_reset_on_fork:1;
unsigned sched_contributes_to_load:1;
unsigned irq_thread:1;
pid_t pid;
pid_t tgid;
unsigned long stack_canary;
struct task_struct __rcu *real_parent;
struct task_struct __rcu *parent;
struct list_head children;
struct list_head sibling;
struct task_struct *group_leader;
struct list_head ptraced;
struct list_head ptrace_entry;
struct pid_link pids[PIDTYPE_MAX];
struct list_head thread_group;
int __user *set_child_tid;
int __user *clear_child_tid;
cputime_t utime, stime, utimescaled, stimescaled;
cputime_t gtime;
cputime_t prev_utime, prev_stime;
unsigned long nvcsw, nivcsw;
struct timespec start_time;
struct timespec real_start_time;
unsigned long min_flt, maj_flt;
struct task_cputime cputime_expires;
struct list_head cpu_timers[3];
const struct cred __rcu *real_cred;
const struct cred __rcu *cred;
struct cred *replacement_session_keyring;
char comm[TASK_COMM_LEN];
int link_count, total_link_count;
struct sysv_sem sysvsem;
unsigned long last_switch_count;
struct thread_struct thread;
struct fs_struct *fs;
struct files_struct *files;
struct nsproxy *nsproxy;
struct signal_struct *signal;
struct sighand_struct *sighand;
sigset_t blocked, real_blocked;
sigset_t saved_sigmask;
struct sigpending pending;
unsigned long sas_ss_sp;
size_t sas_ss_size;
int (*notifier)(void *priv);
void *notifier_data;
sigset_t *notifier_mask;
struct audit_context *audit_context;
uid_t loginuid;
unsigned int sessionid;
seccomp_t seccomp;
u32 parent_exec_id;
u32 self_exec_id;
spinlock_t alloc_lock;
raw_spinlock_t pi_lock;
struct plist_head pi_waiters;
struct rt_mutex_waiter *pi_blocked_on;
struct mutex_waiter *blocked_on;
unsigned int irq_events;
unsigned long hardirq_enable_ip;
unsigned long hardirq_disable_ip;
unsigned int hardirq_enable_event;
unsigned int hardirq_disable_event;
int hardirqs_enabled;
int hardirq_context;
unsigned long softirq_disable_ip;
unsigned long softirq_enable_ip;
unsigned int softirq_disable_event;
unsigned int softirq_enable_event;
int softirqs_enabled;
int softirq_context;
u64 curr_chain_key;
int lockdep_depth;
unsigned int lockdep_recursion;
struct held_lock held_locks[MAX_LOCK_DEPTH];
gfp_t lockdep_reclaim_gfp;
void *journal_info;
struct bio_list *bio_list;
struct blk_plug *plug;
struct reclaim_state *reclaim_state;
struct backing_dev_info *backing_dev_info;
struct io_context *io_context;
unsigned long ptrace_message;
siginfo_t *last_siginfo; /* For ptrace use. */
struct task_io_accounting ioac;
u64 acct_rss_mem1;
u64 acct_vm_mem1;
cputime_t acct_timexpd;
nodemask_t mems_allowed;
seqcount_t mems_allowed_seq;
int cpuset_mem_spread_rotor;
int cpuset_slab_spread_rotor;
struct css_set __rcu *cgroups;
struct list_head cg_list;
struct robust_list_head __user *robust_list;
struct compat_robust_list_head __user *compat_robust_list;
struct list_head pi_state_list;
struct futex_pi_state *pi_state_cache;
struct perf_event_context *perf_event_ctxp[perf_nr_task_contexts];
struct mutex perf_event_mutex;
struct list_head perf_event_list;
struct mempolicy *mempolicy;
short il_next;
short pref_node_fork;
struct rcu_head rcu;
struct pipe_inode_info *splice_pipe;
struct task_delay_info *delays;
int make_it_fail;
int nr_dirtied;
int nr_dirtied_pause;
unsigned long dirty_paused_when;
int latency_record_count;
struct latency_record latency_record[LT_SAVECOUNT];
unsigned long timer_slack_ns;
unsigned long default_timer_slack_ns;
struct list_head
int curr_ret_stack;
struct ftrace_ret_stack
unsigned long long ftrace_timestamp;
atomic_t trace_overrun;
atomic_t tracing_graph_pause;
unsigned long trace;
unsigned long trace_recursion;
struct memcg_batch_info {
atomic_t ptrace_bp_refcnt;

Kitten: task_struct
id_t id;
char name[32];
taskstate_t state;
id_t uid;
id_t gid;
struct aspace *aspace;
struct list_ehad aspace_link;
struct sigpending sigpending;
sigset_t sigblocked;
id_t cpu_id;
cpumask_t cpu_mask;
id_t cpu_target_id;
struct list_head sched_link;
bool sched_irqs_on;
int __user *set_child_tid;
int __user *clear_child_tid;
unsigned long ptrace;
uint32_t flags;
int exit_status;
struct arch_Task arch;
struct fdTable *fdTable;
char *comm;
struct aspace *mm;

Kitten Runtime Memory Footprint Bounded, Unlike Generic Linux



In Practice, CLE's Footprint is Also Bounded



Example of OS Critical State: Page Table Entries

- Correct PTEs critical to proper execution of an application

- Both Kitten and Linux store PTEs in kernel memory

- Memory used dependent on page size
  - 4K pages: 2MB per GB of application memory
  - 2MB pages: 8KB per GB of application memory
  - 1GB pages: 8B per GB of application memory

- Kitten always able to use larger page size due to its segment-based, static memory allocation policy

- Linux attempts to use larger page size but fragmentation is an issue

- Kitten's deterministic page mappings makes PTE verification as easy as storing a base and offset. Linux verification much more complicated

- Note: when a corrected PTE is found, computation must rollback as corrupted value may have been used

Summary

- DRAM failures in the OS a common error on current systems and expected to increase in frequency
- An OS resilient to these failures critical to scalability of extreme-scale systems
- In this work we focus on two HPC OSs likely seen on extreme-scale system: Kitten and Linux
- Presented the complexity of each OS in term of SLOC count, examine memory footprint, and evaluate vulnerability of critical state
- Overall, our results suggest hardening the Kitten LWK to be more tractable due to its smaller, deterministic state

Current and Ongoing Work

Sandia National Laboratories

○ Kitten our initial target for investigation due to its smaller, simpler, and largely static state

○ Identification of critical state nearly complete

○ Conducting analysis of failure mitigation methods, outlining both the space and performance overheads

○ Where appropriate, mechanisms being evaluated for Linux

○ Developed library to also allow applications to recovery from memory failures while avoiding rollback

○ Errors in the OS beyond DRAM also being evaluated

Acknowledgements

9lives Resilience Group:

- Sandia
- UNM
- IBM
- NCSU
- ASU

This work is supported by the Sandia National Laboratories Laboratory Directed Research and Development (LDRD) program

DOE Exascale OS/Runtime Technical Council

- Summarize the challenges of exascale OS/R
- Assess the impact on OS/R of exascale requirements of facilities and production support, applications and programming models, and hardware architectures
- Describe a model to interact with vendors such that there can be proprietary innovation but still support specific APIs that allow interoperability/portability and a minimum set of requirements for functionality
- Identify promising approaches to challenges and requirements by engaging the HPC research community and by drawing on novel approaches from other related areas such as embedded computing and cloud computing
- Articulate dependencies, conflicting requirements, and priorities of the OS/R research agenda
- Submit findings to the DOE Office of Science and NNSA in a report

https://collab.mcs.anl.gov/display/exaosr

Sandia National Laboratories

10^18 **Call for Position Papers**

Added by Peter Beckman, last edited by Peter Beckman on Jun 19, 2012

PAPERS ARE DUE July 13, 2012

The PDF version of the Exascale Operating System and Runtime Software call for Position Papers:
OSR_CallForPositionPapers.pdf

Call for Position Papers: Operating System and Runtime Software for Exascale Systems

The research challenges associated with Exascale systems have been articulated in numerous reports. We clearly need new approaches to address these tremendous challenges. The Exascale Operating Systems and Runtime (OS/R) Technical Council (TC) has been convened by the Department of Energy (DOE) Office of Science and National Nuclear Security Administration to develop a research agenda and plan to address them. This activity will result in a report describing the research priorities and the process used to develop this plan. The report will be published in November 2012. As part of the process, we solicit community input, in the form of position papers that describe novel research approaches for exascale OS/R. Responsive submissions will be made public via the OS/R TC website and portions may be included in the final report. The OS/R TC will review these position papers and invite selected contributors to participate in a workshop to be held on October 4-5, 2012 in Washington, DC.

Necessary background information, including previous exascale reports and submission instructions, can be found on the public web pages for the OS/R TC: https://collab.mcs.anl.gov/display/exaosr

Requirements:
Position papers should describe a fundamental computer science research approach in the development of operating systems or runtime software to addresses the key challenges associated with exascale computing systems. This description should be followed by a brief summary of related work and an assessment of the approach based on the following dimensions:

- **Challenges addressed**: Which exascale OS/R challenges does this approach address?
- **Maturity**: What are the indicators that this approach will address the identified challenges?
- **Uniqueness**: To what extent is the proposed approach unique to Exascale systems? Could it be addressed by other research programs?
- **Novelty**: How is this approach different than existing solutions?
- **Applicability**: To what extent will the proposed approach, if successful, be applicable to other areas?
- **Effort**: How much effort is needed to effectively explore this approach?

Each position paper must be no more than two pages. The paper may include any number of authors, but must provide contact information for a single "contact" author. There is no limit to the number of position papers that an individual or group can submit. Authors are strongly encouraged to follow the structure presented above.