Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Analyzing System Calls in Multi-OS Hierarchical Environments

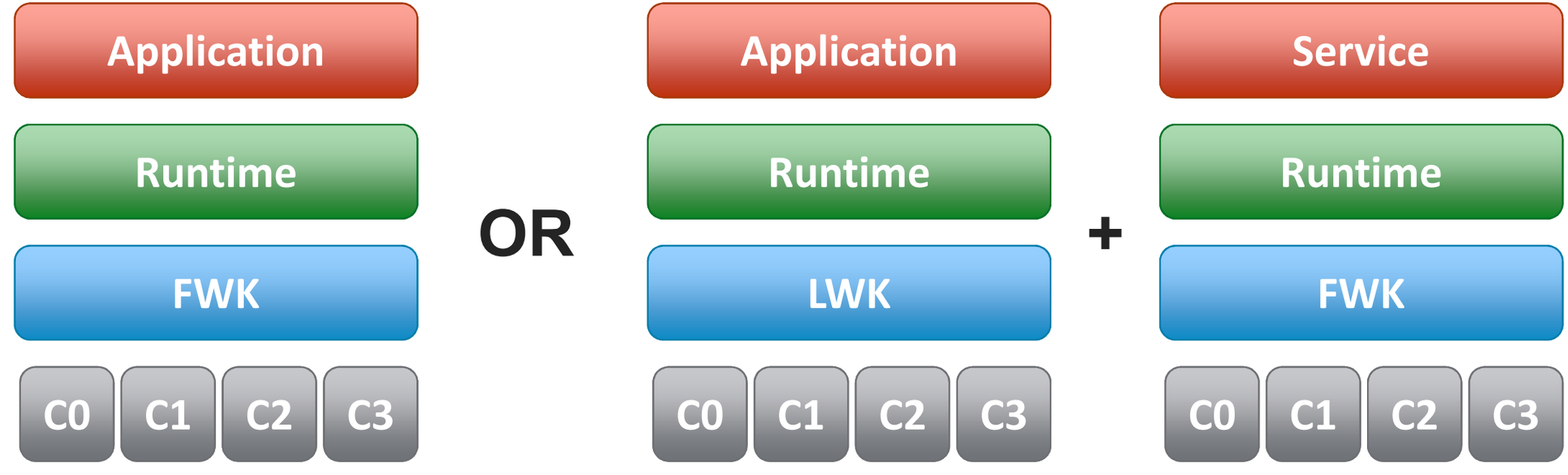
ROBERTO GIOIOSA, ROBERT W. WISNIEWSKI, RAVI MURTY, TODD INGLETT

Runtime and Operating Systems for Supercomputers (ROSS 2015)
Portland, OR

Outline

- Introduction
- Tracing Infrastructure
- Experimental Setup
  - Hardware & Software
  - Applications
- Experimental results
  - LULESH & NEKBone
  - Aggregate
- Conclusions

Petascale Supercomputers



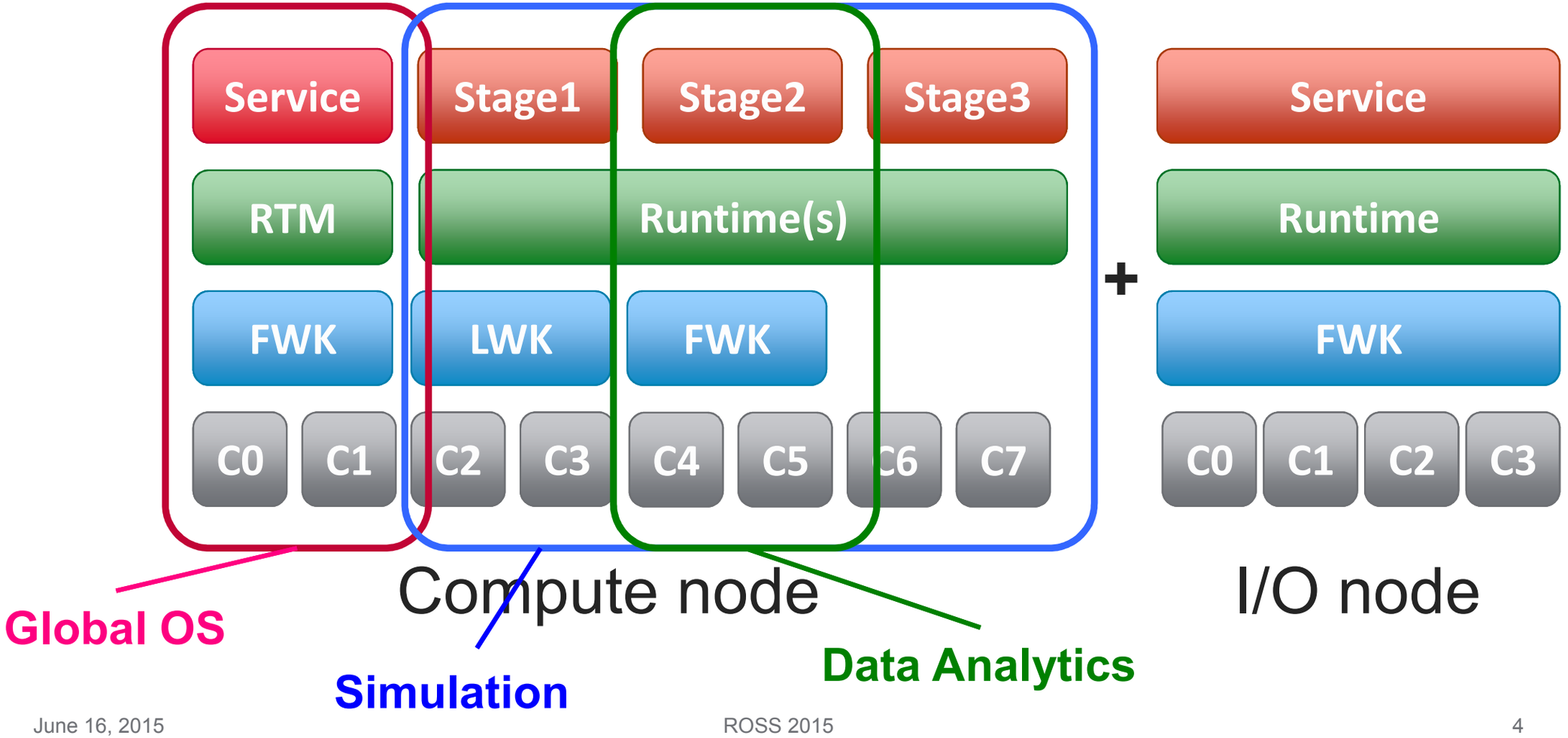
Compute node

Compute node

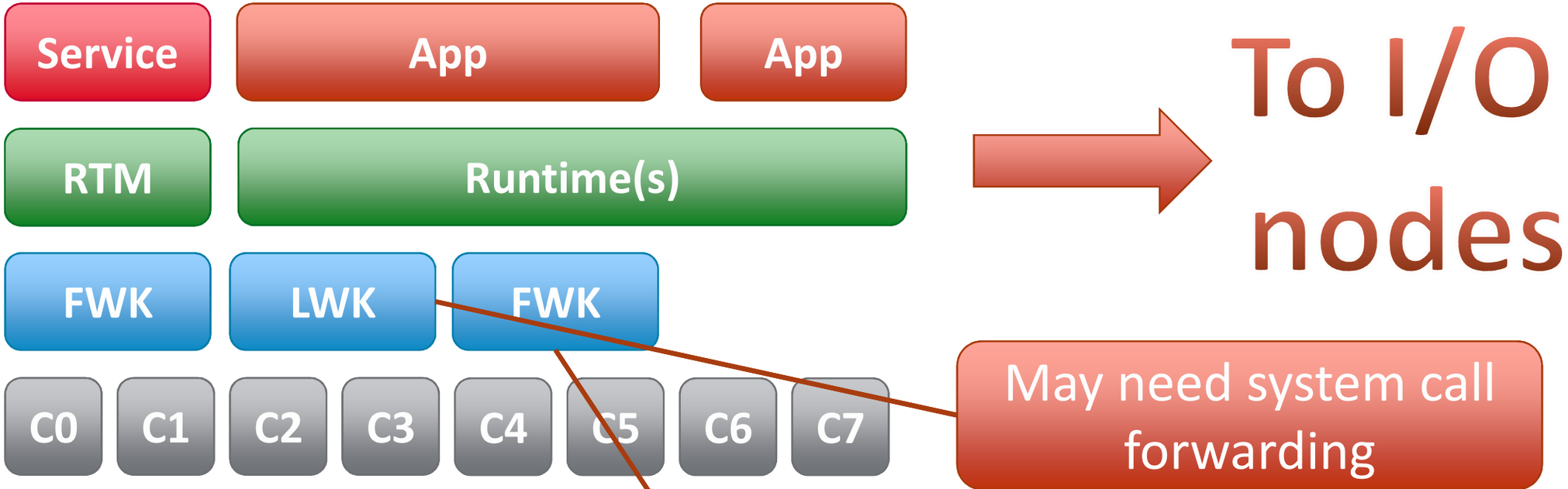
I/O node

Exascale Supercomputers

- Need of coupling simulations and in-situ analysis
- Low-system noise with rich ecosystem
- Heterogeneous architectures
- PIM

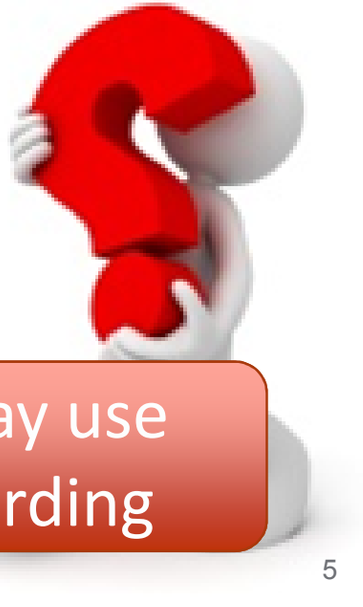


Designing Exascale Hierarchical Multi-OS

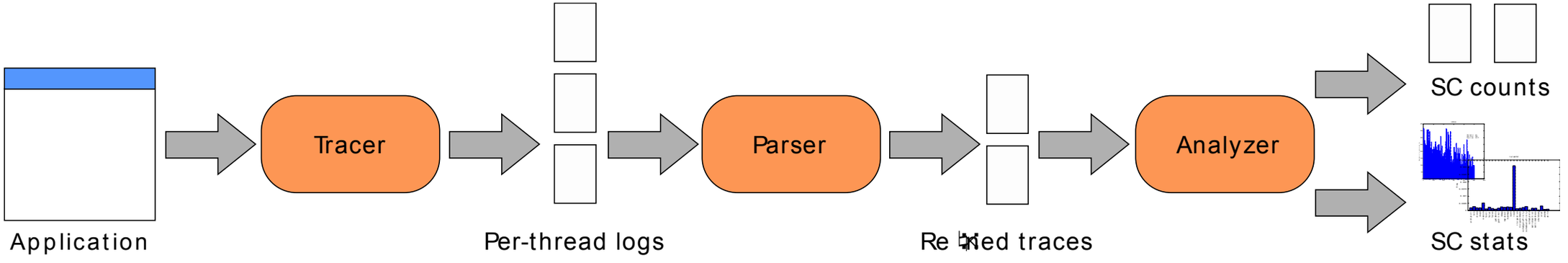


Goal: Make informed decisions based on experimental data about where to implement an OS service:

- Local kernel on the compute cores
- Local runtime on the compute cores
- Local kernel on the service cores
- Remote I/O nodes



Tracing Infrastructure



- Trace system calls used by scientific applications
- Analyze unmodified applications
- Collects data (e.g., parameters, execution time)
  - per-thread log
- Correlate system calls through producer-consumer chain
- Off-line analysis tools
  - Global and local view
  - Per-thread, per-application and aggregated results

Experimental Setup: Hardware & Software

- Hardware setup
  - 2xIntel SandyBridge E5-2680 @ 2.7 GHz
  - 8 core per socket (32KB L1D, 32KB L1I, 256KB L2)
  - Shared 20MB L3
  - 2 hardware threads per core
  - 128 GB RAM (2 NUMA domains)
  - 2xSATA disk
- Software infrastructures
  - Intel C/C++/Fortran compiler version 13.1.3
  - Intel OpenMP/MPI version 4.1
  - Linux 3.14
- Applications
  - From CORAL, SEQUOIA, NERSC, ANL

Experimental Setup: Applications

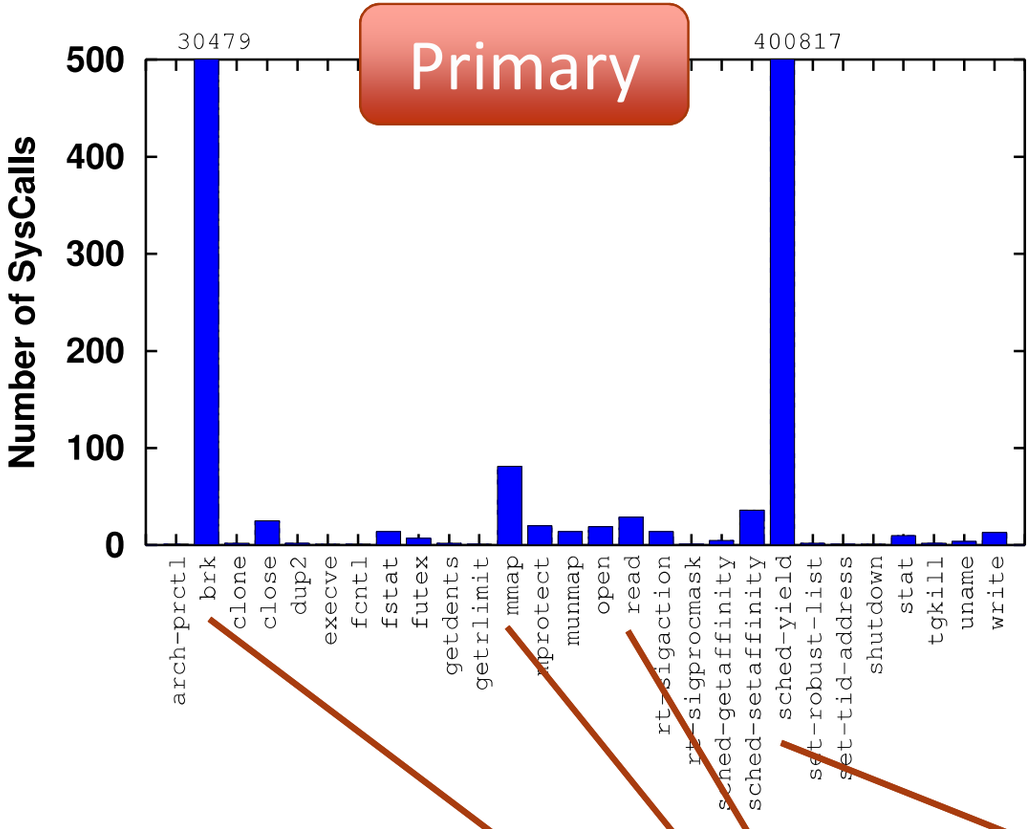
Application	Language	Parallelism	Application	Language	Parallelism
AMG2013	C	MPI/OpenMP	MCB	C	MPI/OpenMP
BT-MZ	Fortran	MPI/OpenMP	NAMD	Charm++	Charm++
GTC	Fortran	MPI/OpenMP	NEK5000	Fortran/C	MPI
HACC	C++	MPI/OpenMP	miniFE	C++	MPI/OpenMP
IRS	C	MPI/OpenMP	NEKBone	Fortran/C	MPI
LAMMPS	C++	MPI	QBOX	C++	MPI/OpenMP
LSMS	Fortran/C++	MPI/OpenMP	QMCPACK	C/C++	MPI/OpenMP
LULESH	C	MPI/OpenMP	SPHOT	Fortran	MPI/OpenMP
LU-MZ	Fortran	MPI/OpenMP	SP-MZ	Fortran	MPI/OpenMP
			UMT2013	Fortran/C/C++	MPI/OpenMP

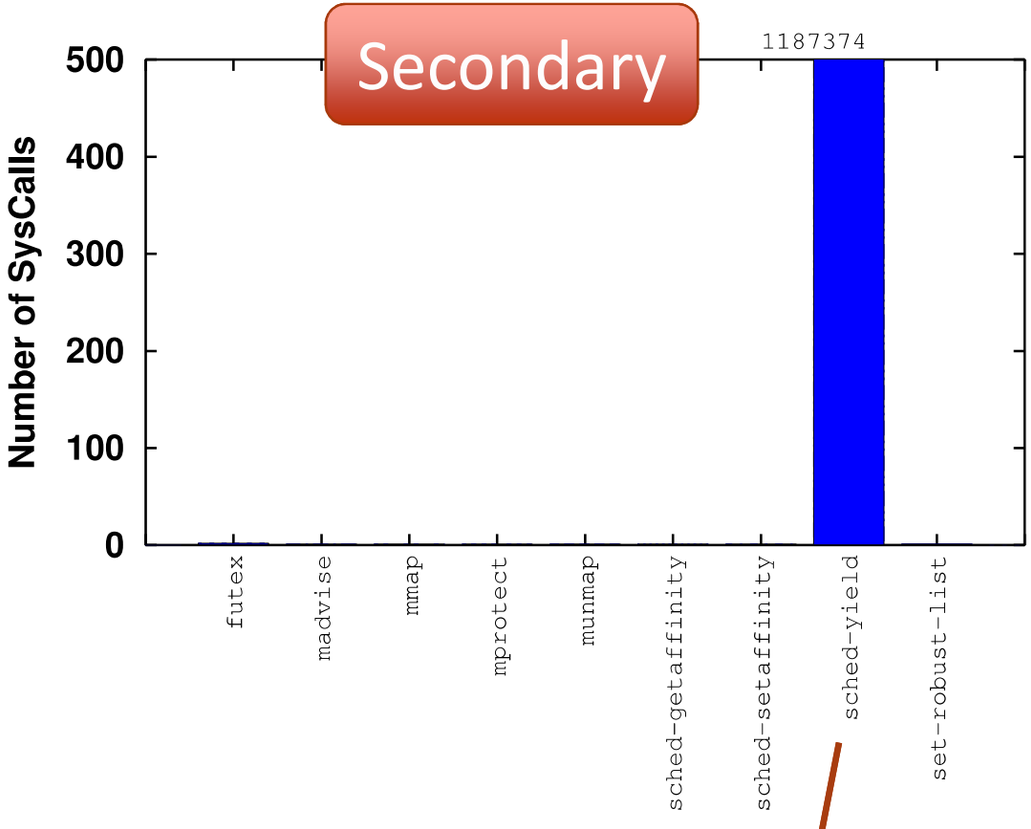
From: CORAL, NERSC and Sequoia benchmark suites, DOE ASCR Co-Design Centers, DOE Office of Science

Assumptions and Caveats

- Check-pointing I/O not considered
  - Frequency set by the user
  - Depends on the MTBF of the system
- Debugging I/O not considered
  - Interactive VS offline
  - Console VS disk
- Applications may require OS services not observed in these experiments

OpenMP Parallelism (LULESH)



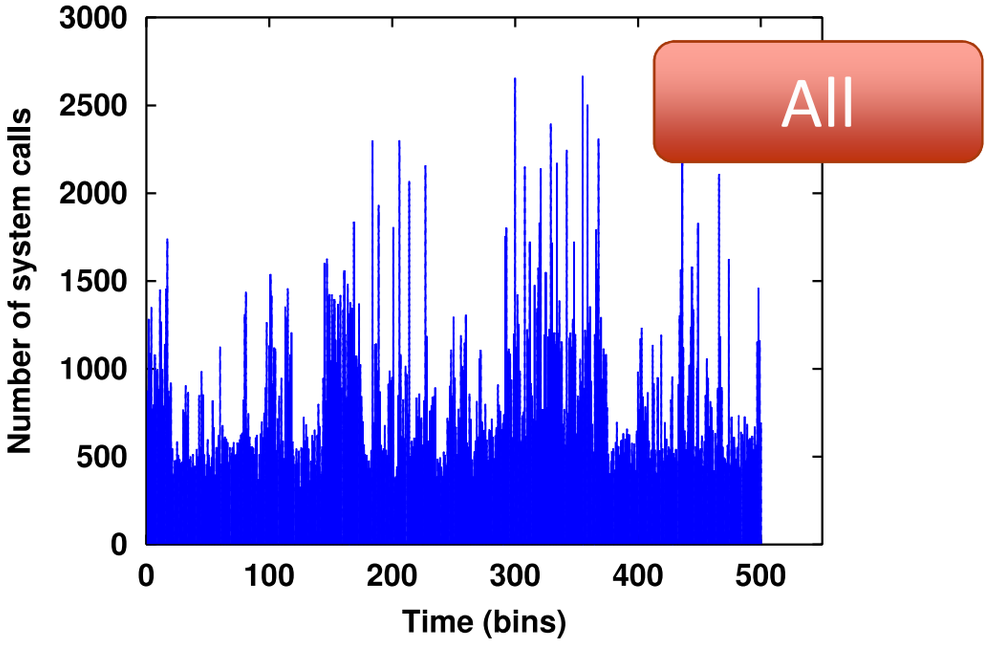


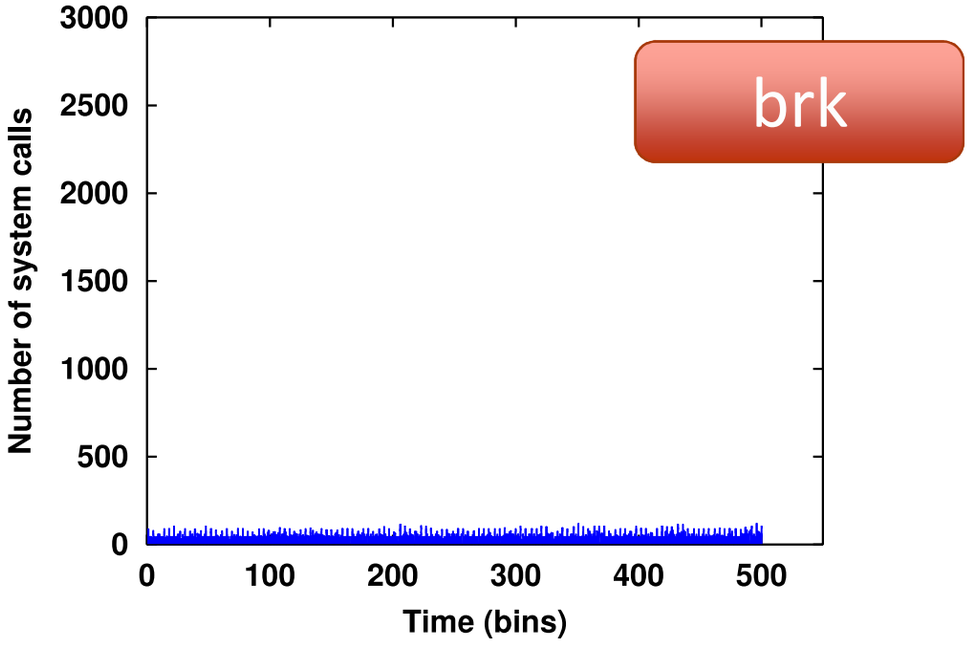
- Primary OpenMP threads issue more system calls:
  - Memory allocation, I/O, signaling, etc.
- Secondary OpenMP threads mainly perform computation
  - Synchronization and scheduling calls when there is no computation to perform

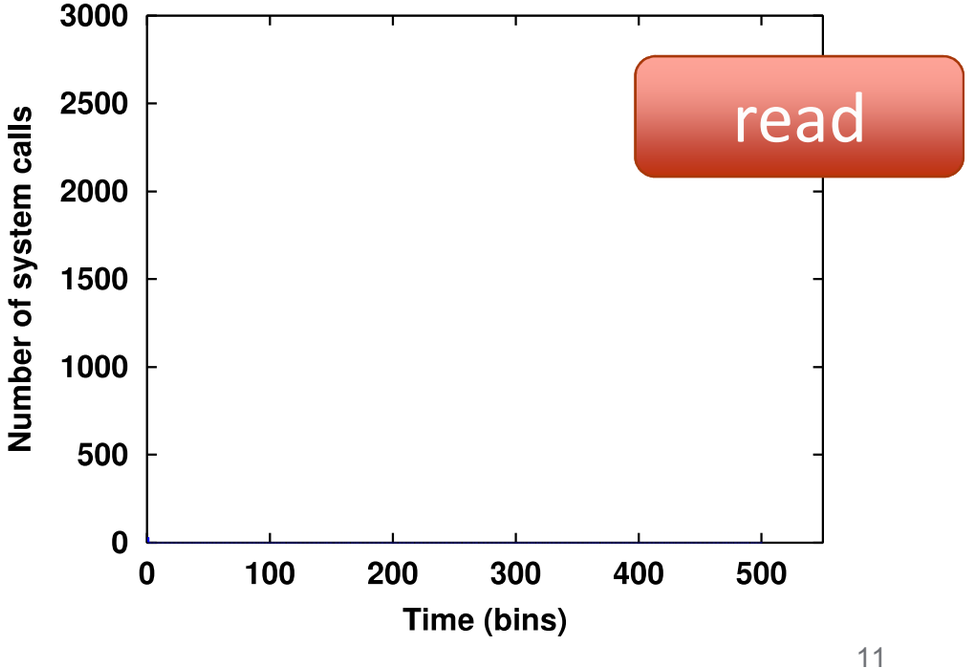
Memory mgnt and I/O

Scheduling

System calls time trace (LULESH)

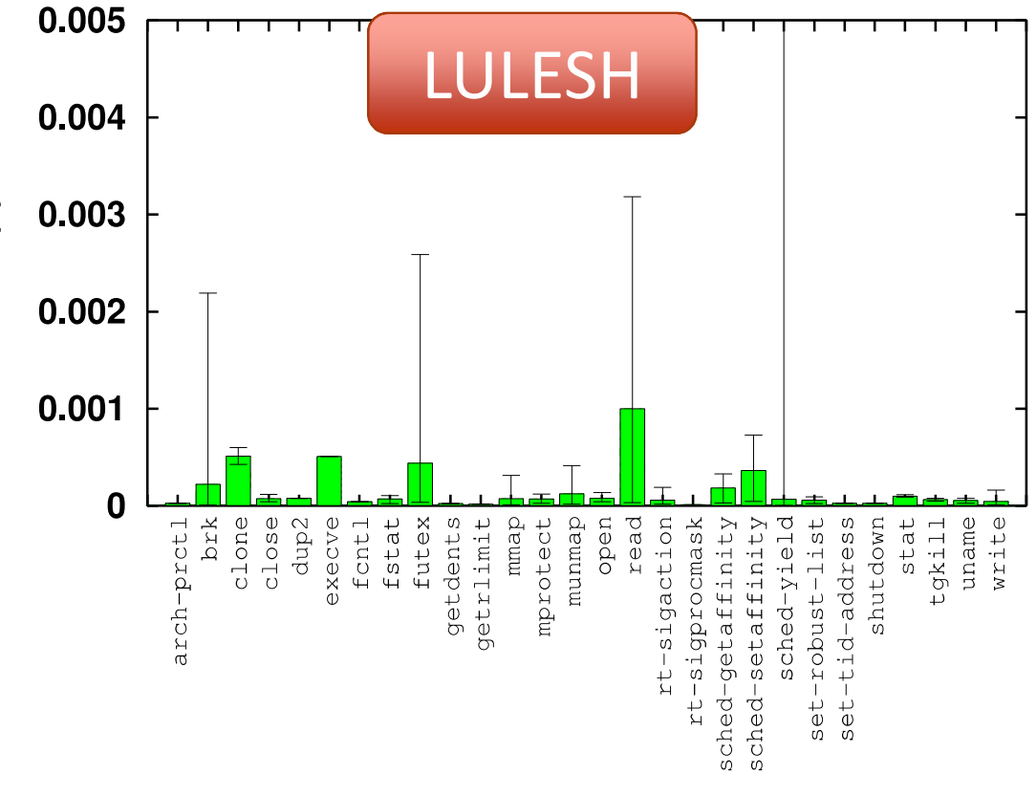


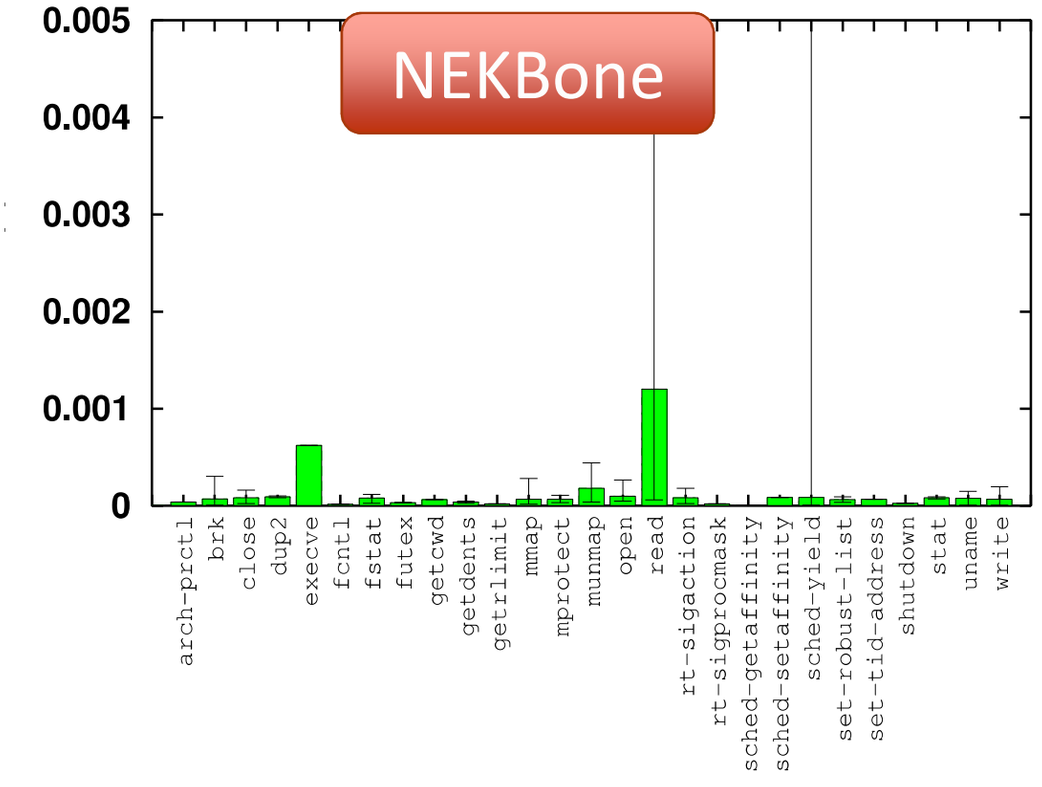




- System calls are issued throughout the entire execution of the application
- The `brk()` calls is heavily used during the solving phase
- I/O is usually contained at the beginning/end of the execution
- Scheduling and locking calls are common in the solving phase too

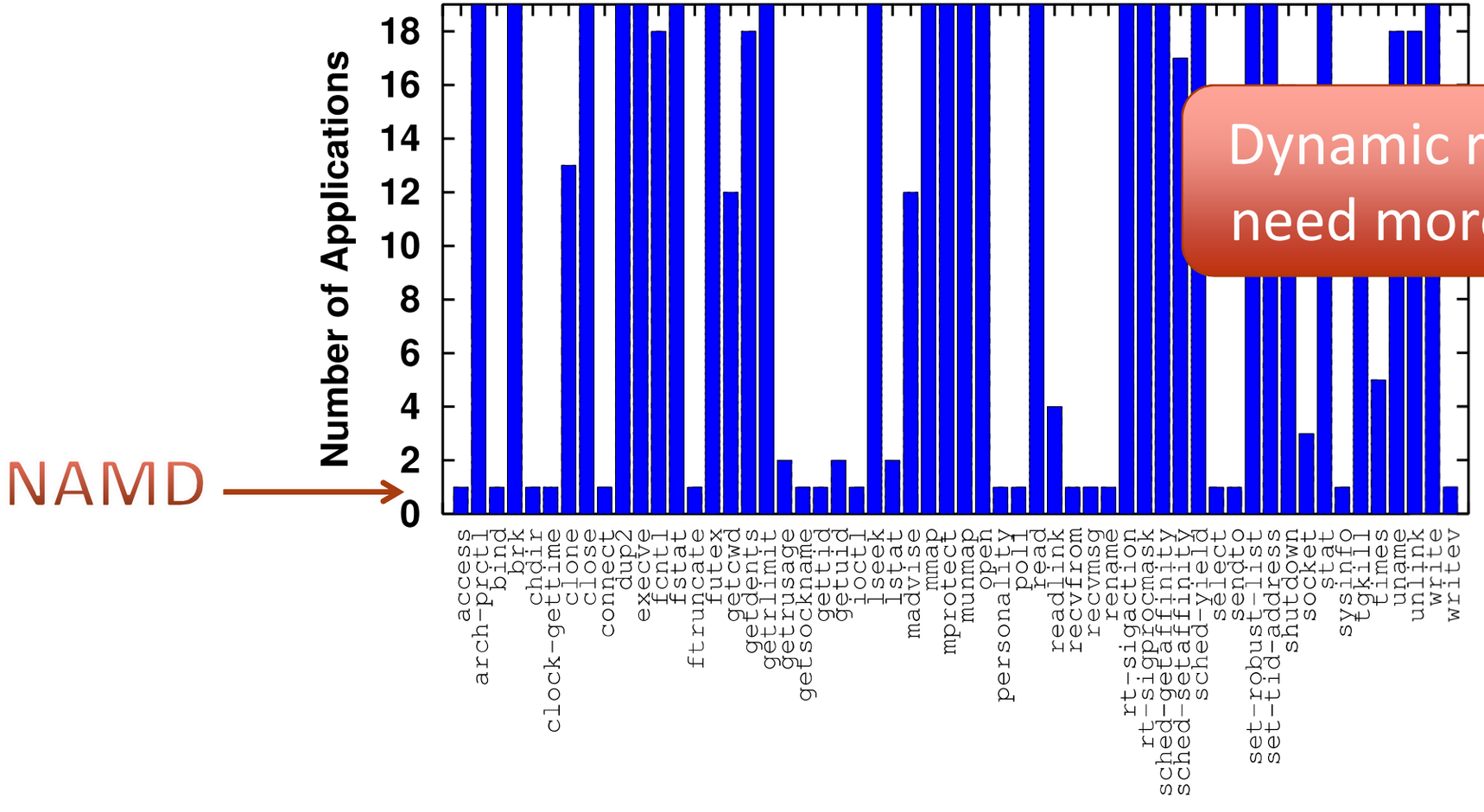
SysCall Execution Time Analysis





- Some system call have bounded execution time
- Other may block or spin until an event occurs:
  - `sched_yield()`, `mutex()`, `read()`, etc.
  - Execution time varies considerably
- There are commonalities but also differences among the applications

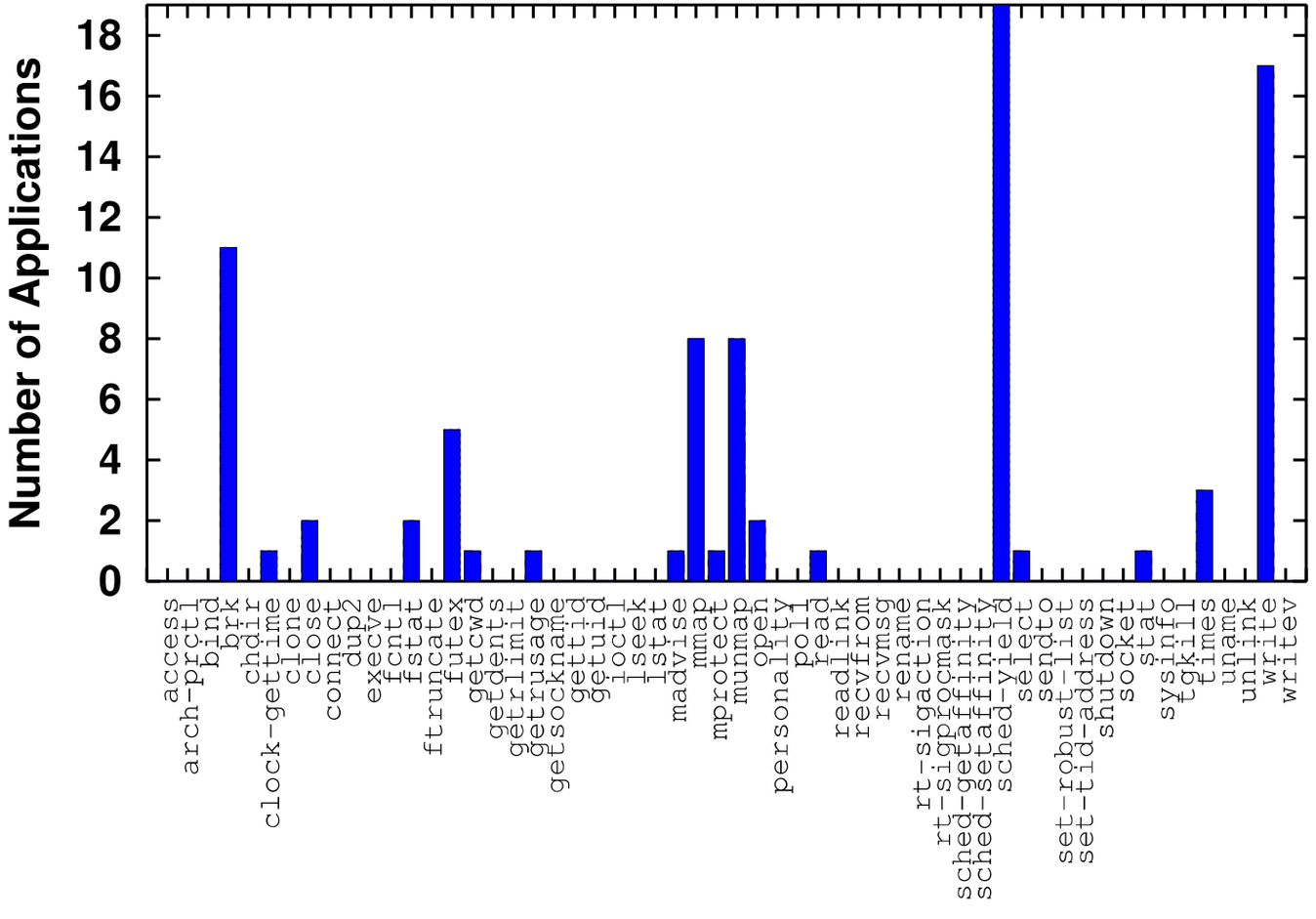
Aggreagete Results: Overall



Dynamic runtimes may need more OS services

- Only 56 systems calls where observed in the 19 applications
- 22 system calls are used by all applications
- In 18 cases, a system call is only used by one application
  - In 13/18 this application is NAMD

Aggregate Results: Solving Phase



- Fewer system calls (18) are used during the solving phase
- Only 7 system calls are used by more than two applications
- Memory management, scheduling and locking are the most common

Conclusions

Exascale challenges, the new hardware technologies, new execution models => re-design of OS/Runtime

We developed an framework to analyzed unmodified HPC applications

Data collected can help OS/Runtime designers to make informed decisions on where to implement each OS service => e.g., ARGO

Our analysis shows that 56 system calls are used by the tested applications, 18 of which in the solving phase

Applications that use dynamic runtimes might need a richer system software ecosystem

U.S. DEPARTMENT OF ENERGY

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

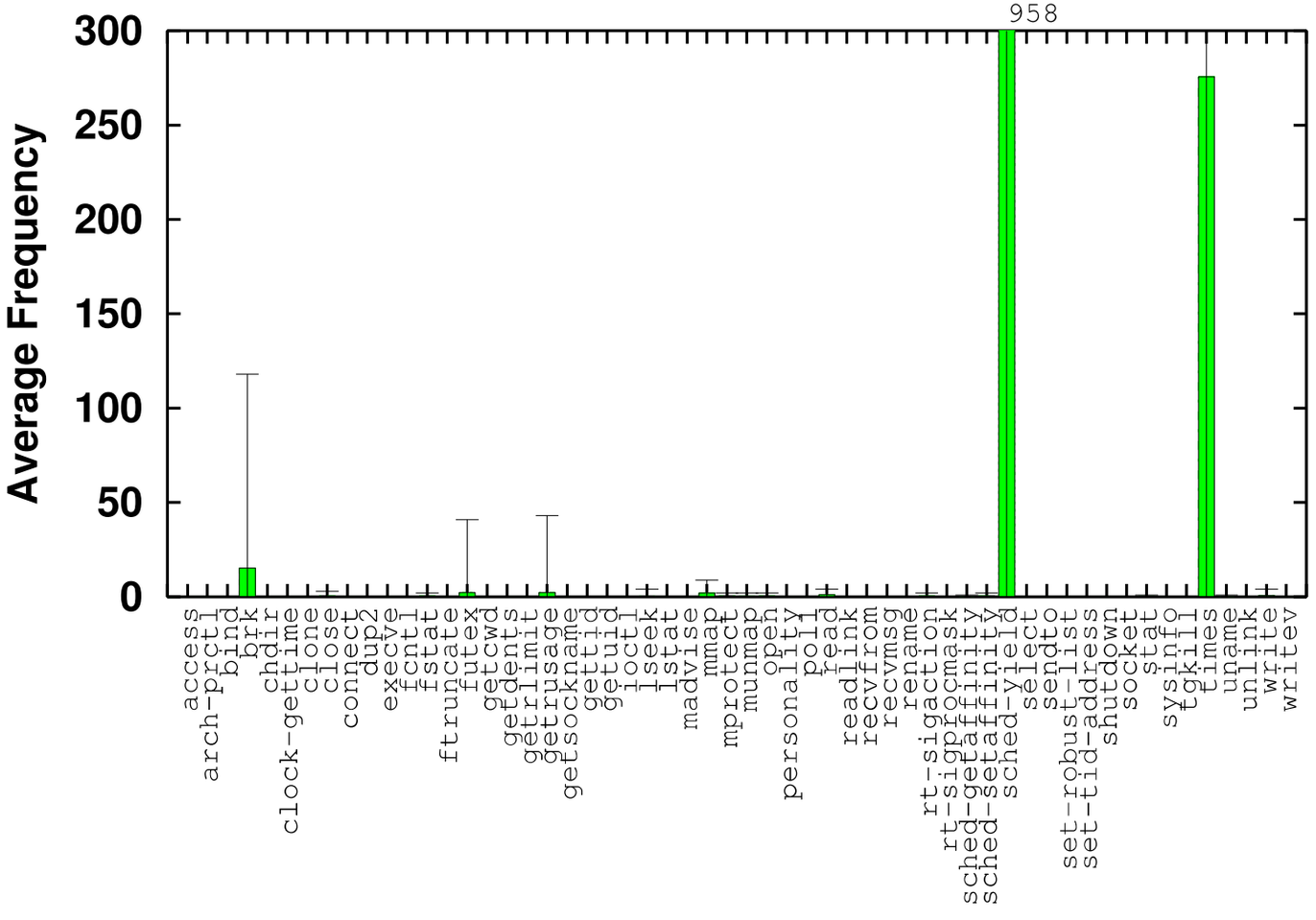
intel®

Questions?

Contacts: roberto.gioiosa@pnnl.gov

Backup

Overall: frequency



- High-frequency system calls are invoked during the solving phase
- `sched_yield()` and `mutex()` are the most frequent because of the OpenMP loops
- High variance caused by varying behavior of applications