

# Coping with Petascale Architectures

April 26, 2008



**Bronis R. de Supinski**  
**Center for Applied Scientific Computing**  
**Lawrence Livermore National Laboratory**

Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94551

This work performed under the auspices of the U.S. Department of Energy by  
Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

LLNL-PRES-403033

# Infrastructure Plan for ASC Petascale Environments



- ASC L2 milestone, a formal planning document
- Identifies, assesses, specifies the development and deployment approaches for petascale infrastructure
  - Programming Environments and Tools
  - Petascale Data Analysis
  - I/O, File Systems, and Storage
  - Networks and Interconnects



# Provides a roadmap for ASC petascale activities



- Technical working groups focused on five key questions
  - Most critical infrastructure components?
  - Gaps or issues that must be addressed?
  - Possible approaches to closing gaps?
  - Generalized or specific to platforms?
  - When are deployments/capabilities available?
- Three Programming Environments and Tools foci
  - Programming Models
  - Correctness Tools
  - Performance Analysis Tools

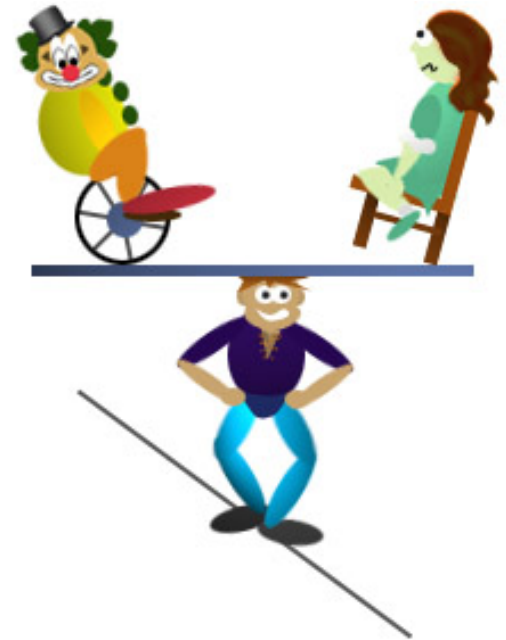
# How will this plan be used?



- Will help coordinate shorter-term tactical deployment
- Will guide long-term strategic research/development
- Approaches to address gaps are characterized by:
  - Priorities (how important is this to the program?)
  - Difficulties (how technically challenging will this be?)
  - Costs (what is the estimated size of the effort?)
- Phases used to show when approaches are applicable
  - Phase I - FY08/FY09/FY10 (Roadrunner, Sequoia ID)
  - Phase II - FY11/FY12/FY13 (Sequoia, new capability)
  - Phase III - FY14/FY15/FY16 (Future petascale systems)

# Driving assumptions for directions

- Must protect substantial investment in ASC codes
  - Newest integrated codes represent 10+ years of multiperson effort (i.e., millions of dollars)
  - Application teams want to focus on physics
- Significant hardware changes ahead
  - Multicore nodes
  - Millions of cores
  - Less memory per core
  - Heterogeneity

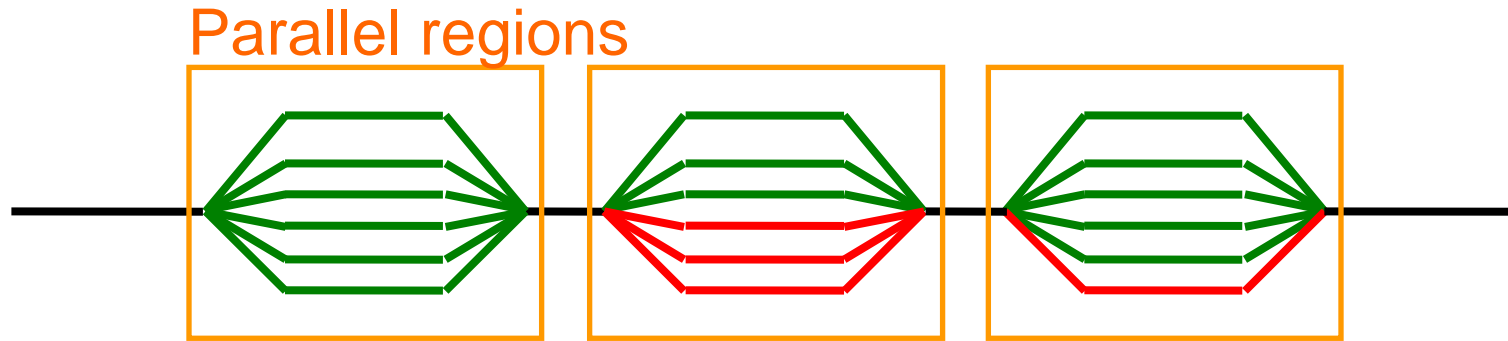


# Identified five critical programming models concerns



- No common programming model for new architectures
- **Explore new models, standardize via OpenMP**
- Need to control data, thread and task placement
- **APIs and solutions implemented in libraries**
- Lack of tools to support the migration of applications
- **Compiler infrastructure; talk w/Jeff Keasler re: TALC**
- Increasing occurrence of soft (hardware) errors
- **Novel checkpoint solutions; APIs; ddcMD example**
- Power consumption of large scale systems is untenable
- **Dynamic concurrency throttling; MPI DVS algorithms**

# Concurrency throttling is a no cost power saver



- Dynamic concurrency throttling (DCT) modifies:
  - Number of threads that execute a parallel region
  - Placement of threads on processing elements
- Why throttle concurrency?
  - Alleviate scalability bottlenecks to decrease run time
  - Deactivate execution units to reduce power consumption

# We apply DCT to OpenMP regions

- Each region (phase) executed many times
- First few iterations of each region:
  - Record events for prediction
  - Predict optimal configuration
- Use predicted optimal for remaining iterations

```
do i = 1, n
```

```
...
```

```
!$omp parallel do
```

```
do ...
```

```
...
```

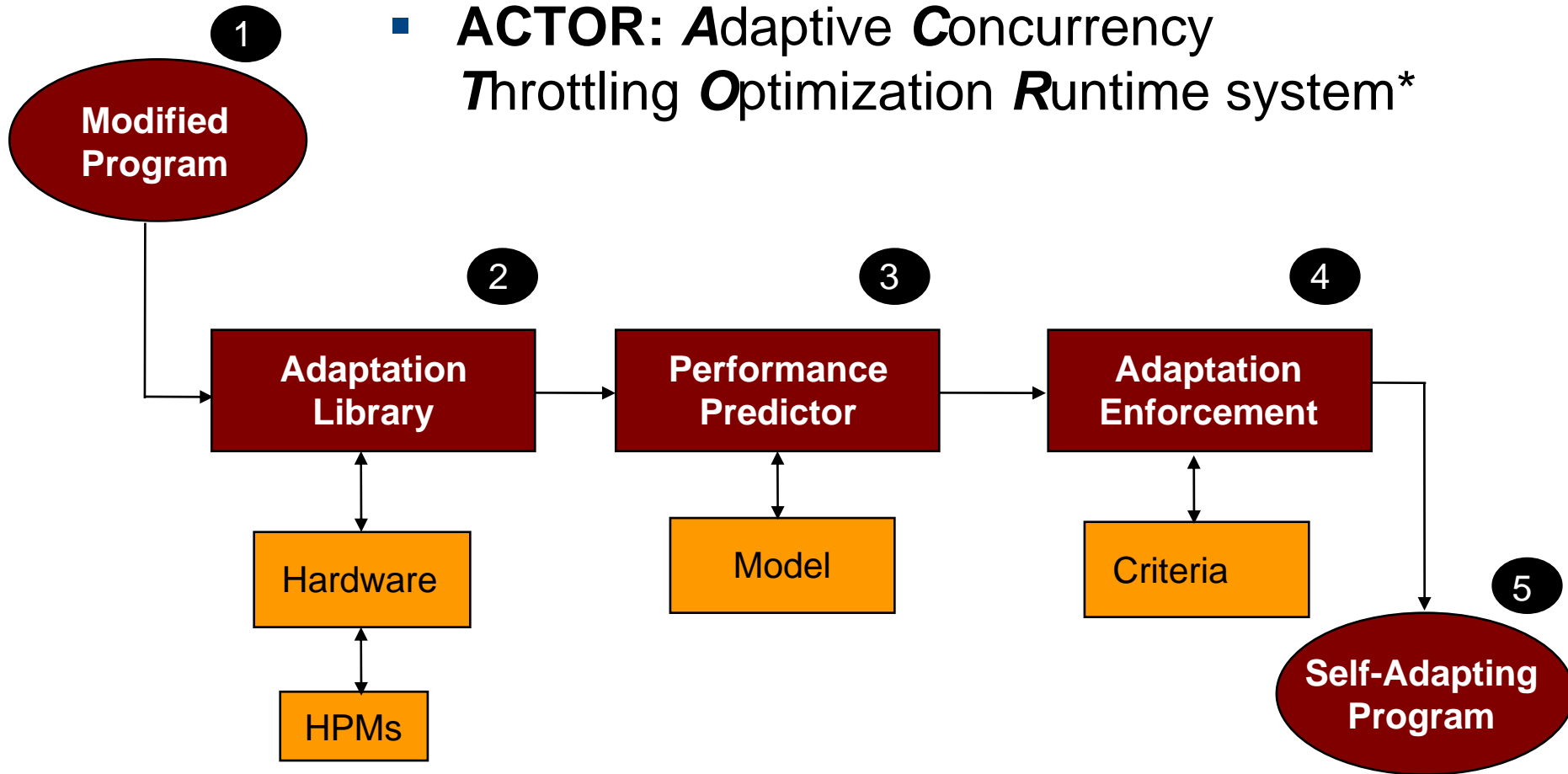
```
!$omp parallel do
```

```
do ...
```

```
...
```

```
end do
```

# Our runtime library implements DCT



- Model uIPC: “useful Instructions per Cycle”
- Map observed event rates of performance counters on sample configuration to predicted performance

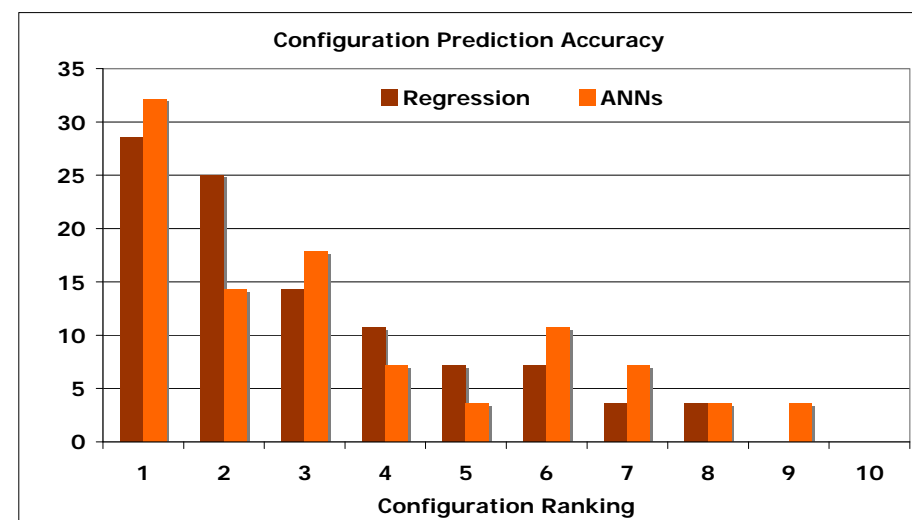
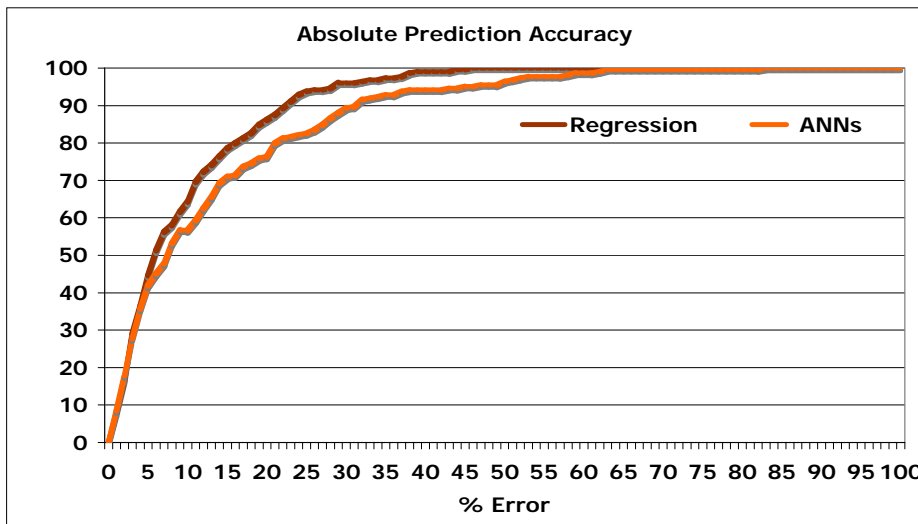
$$\text{uIPC}(t) = \text{uIPC}(s) * \alpha(e_1(s), \dots, e_n(s)) + \beta$$

$$\alpha() = \sum_{i=1}^n (e_i(s) * x_i + y_i) + z$$

$$\text{uIPC}(t) = \text{uIPC}(s) * \sum_{i=1}^n (e_i(s) * x_i) + \text{uIPC}(s) * \lambda + \beta$$

# Linear regression and ANNs provide highly accurate DCT models

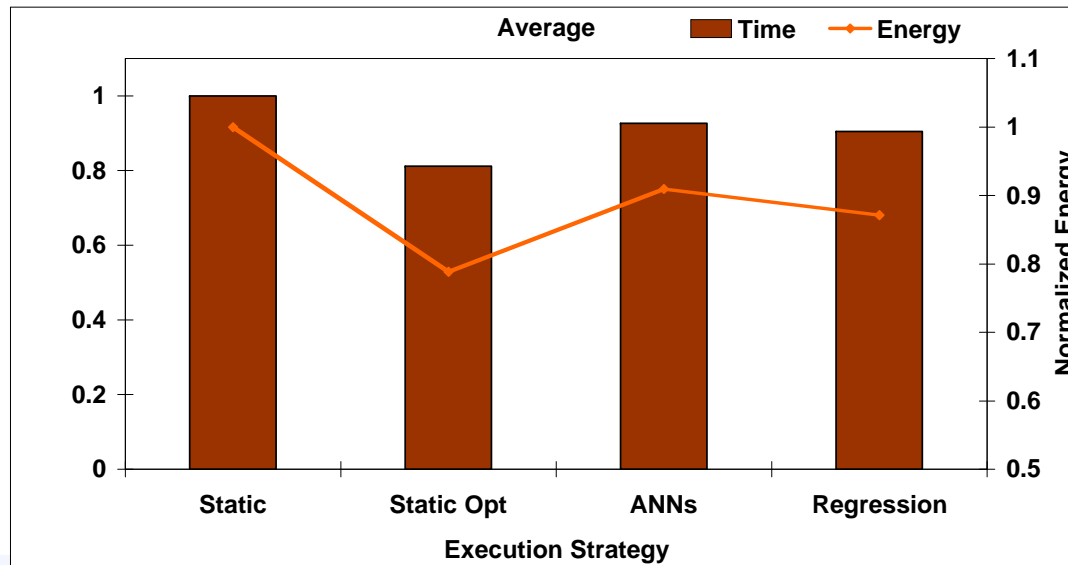
- Applied on dual-processor Intel Core 2 Quad
- Regression has higher median accuracy (5.3% vs 7.2%)
- ANNs predict best more often, regression better overall
- Postulate fairly linear relationship for this platform



# Overall results favor regression on test platform

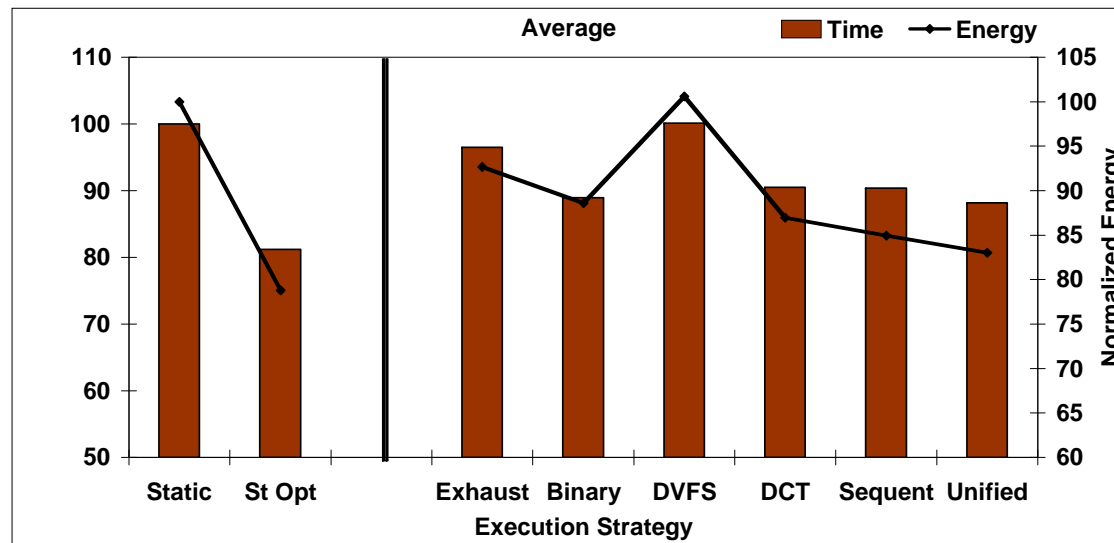


- ANN-based model: median performance gains of 7.4%
  - Reduces power (1.8%) and energy (9.1%) consumption
- Regression-based model: 9.5% performance improvements
  - Reduces power (3.7%) and energy (12.9%) consumption
- Different platforms, scale could change relative results



# Combined model for DCT+DVS

- Compare strategies for multi-dimensional adaptation
  - DCT with DVS applied to remaining cores
  - Unified and individual DCT, DVS models
  - Unified DCT+DVFS model sees best results: 11.8% performance, 5.9% power, 17.0% energy



# Identified five critical correctness tools concerns



- Traditional debuggers do not scale to required level
- **Work w/vendor(s) to make scale to 10,000 cores**
- Traditional debugging paradigm does not scale to 1M cores
- **Develop complementary lightweight debugging tools**
- Need automated mechanisms to identify root causes
- **Develop static and dynamic automated checkers**
- Future systems will have dramatically less memory/core
- **Develop tools to analyze and to reduce memory usage**
- No debugging solution exists for heterogeneous systems
- **Implement heterogeneous debugging tools**



# Traditional debugging techniques do not scale to full BG/L



## TotalView on BG/L – 4096 Processes

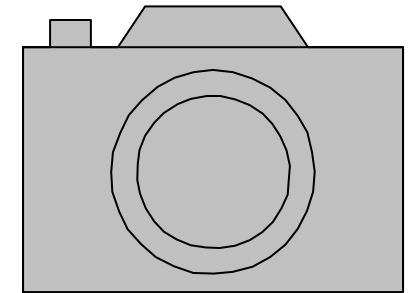
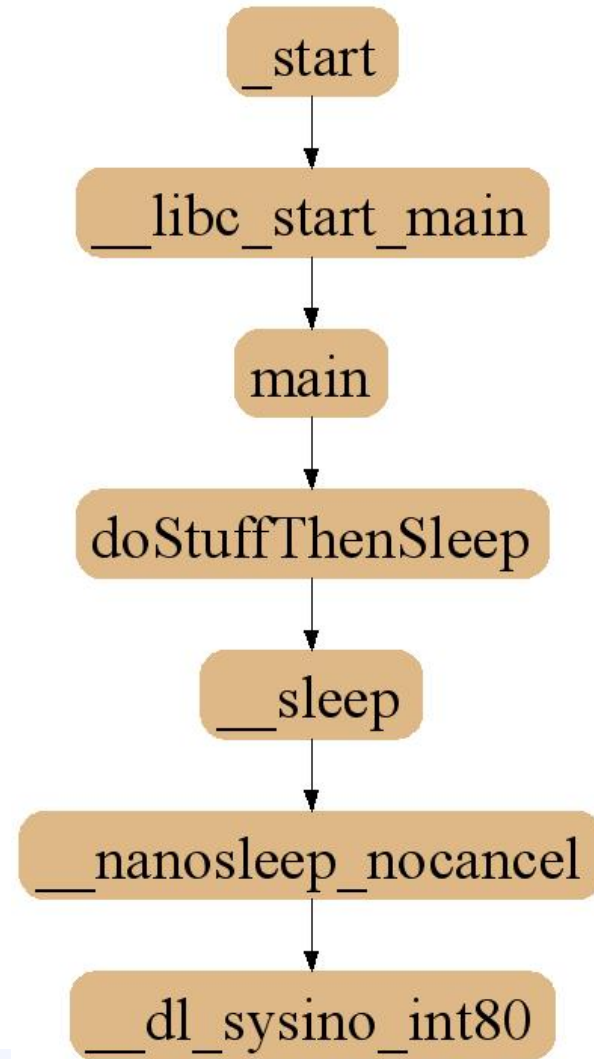
Operation	Latency
Single step	~15-20 secs.
Breakpoint Insertion	~30 secs.
Stack trace sampling	~120 secs.

- TotalView offers the best scalability in the industry
- Typical debug session includes many interactions

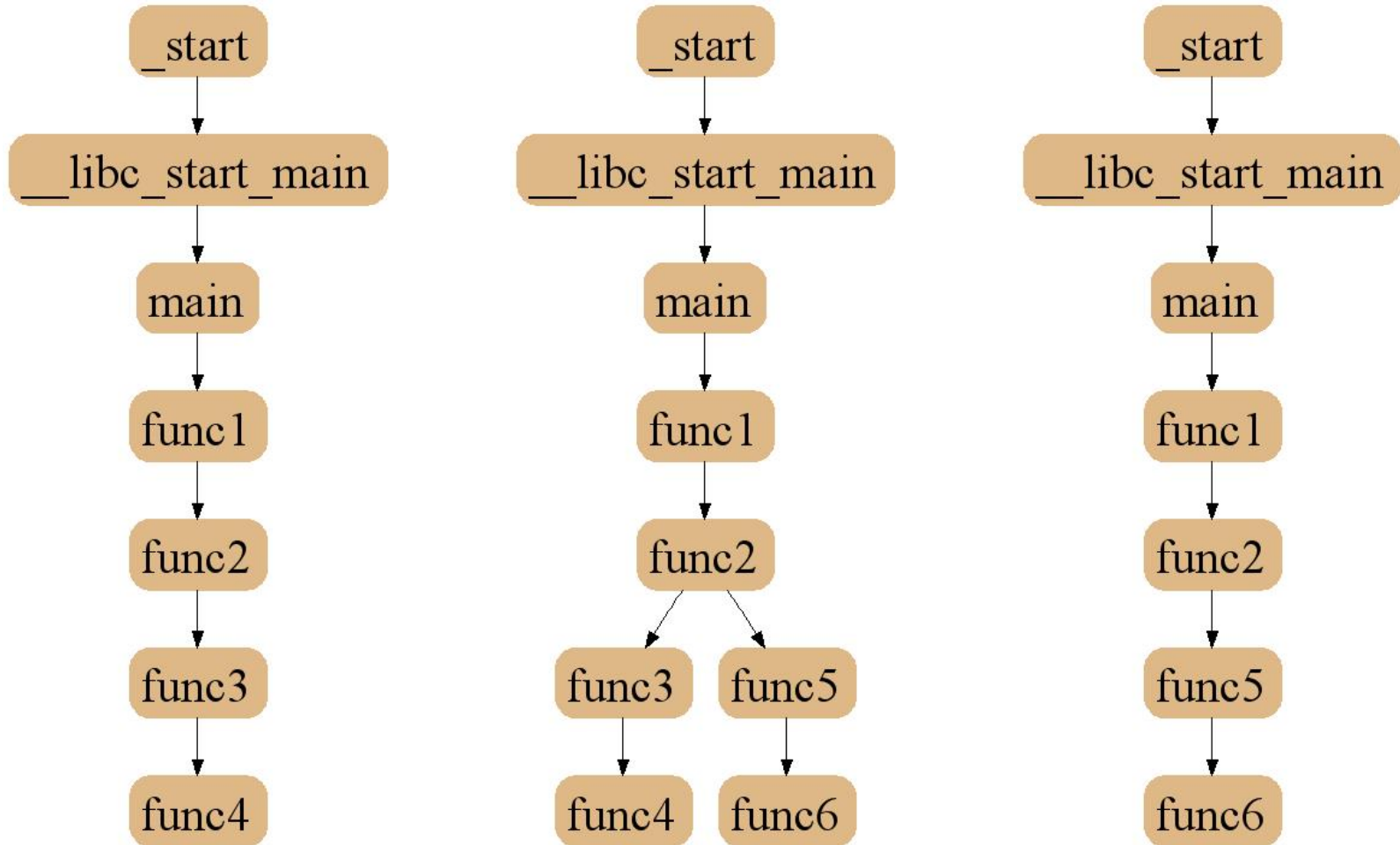
4096 is only 2% of BG/L!

# We build on simple concept: singleton stack trace

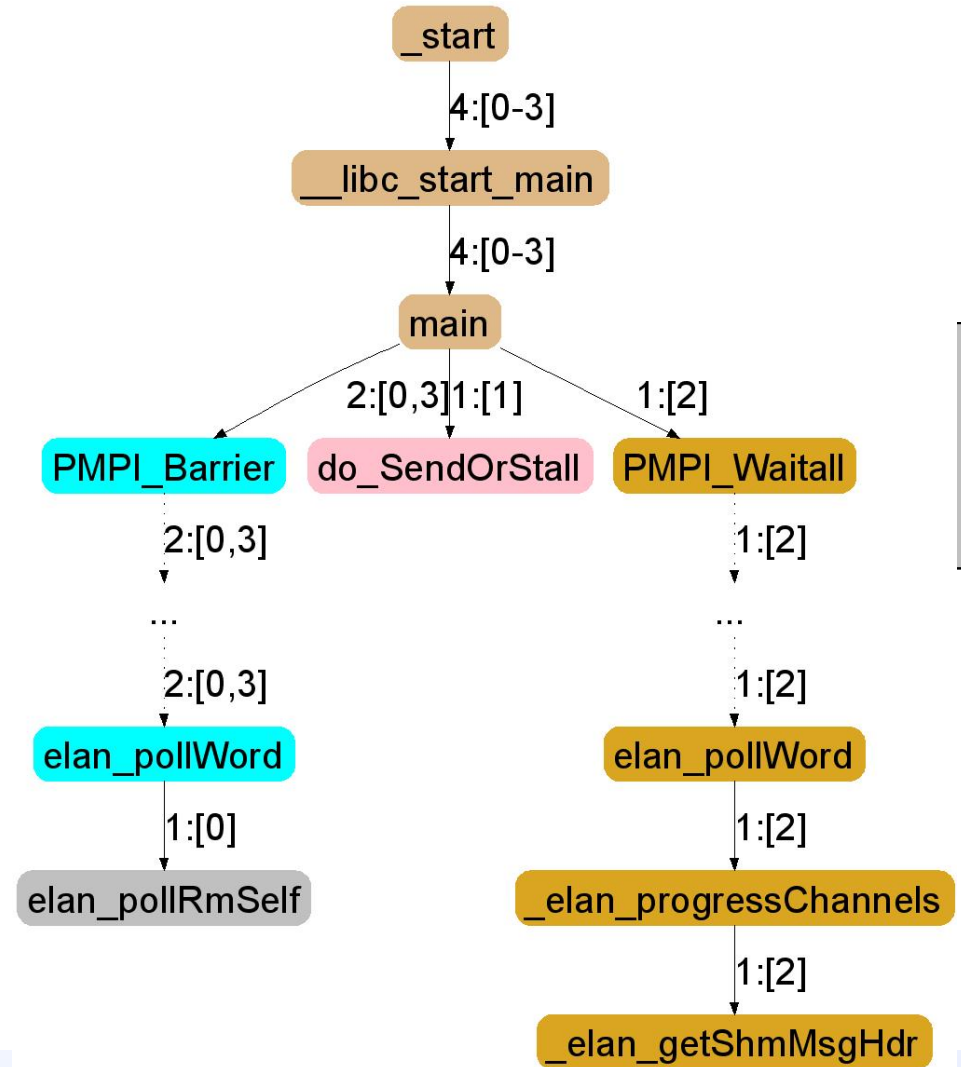
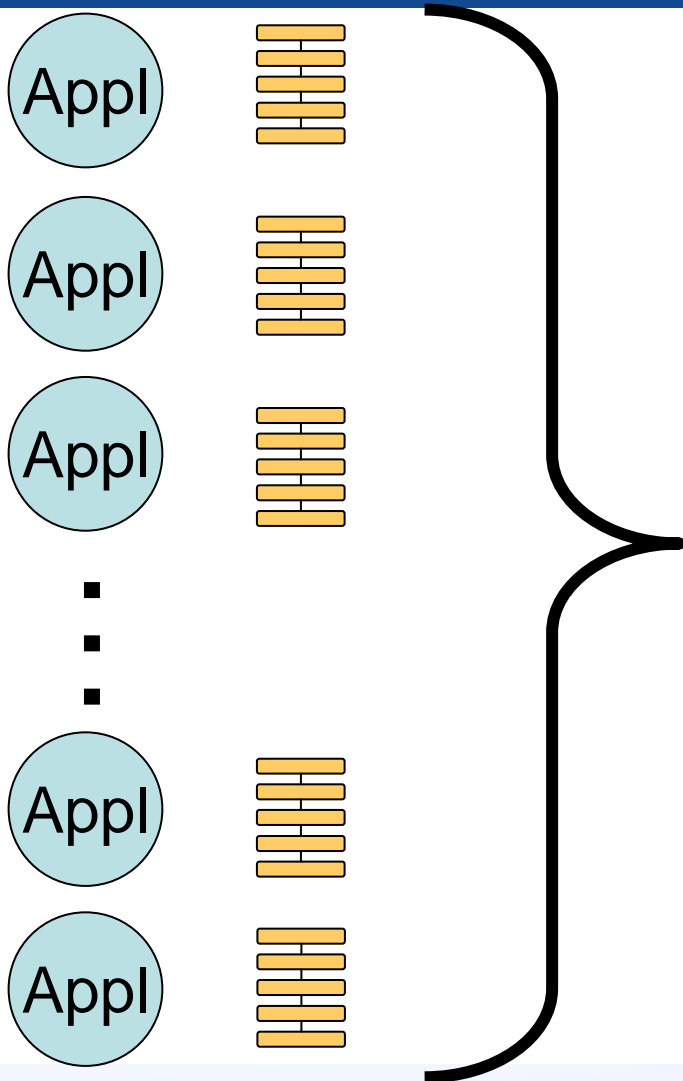
Appl.



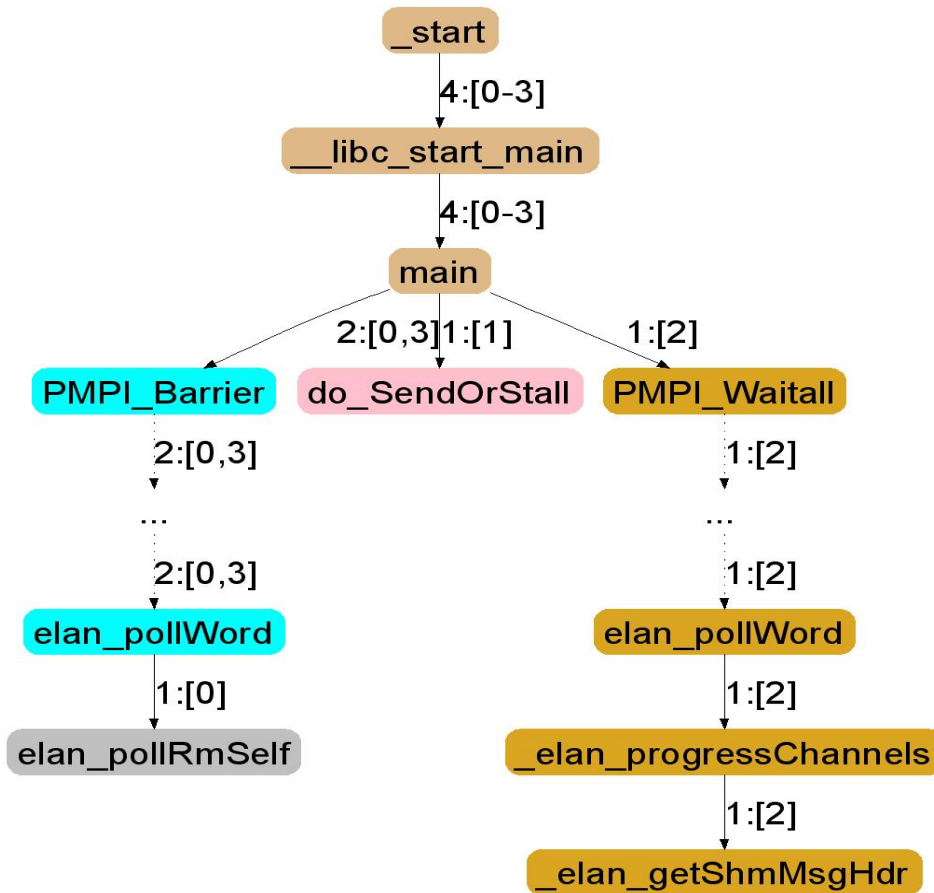
# We build on this concept by merging stack traces



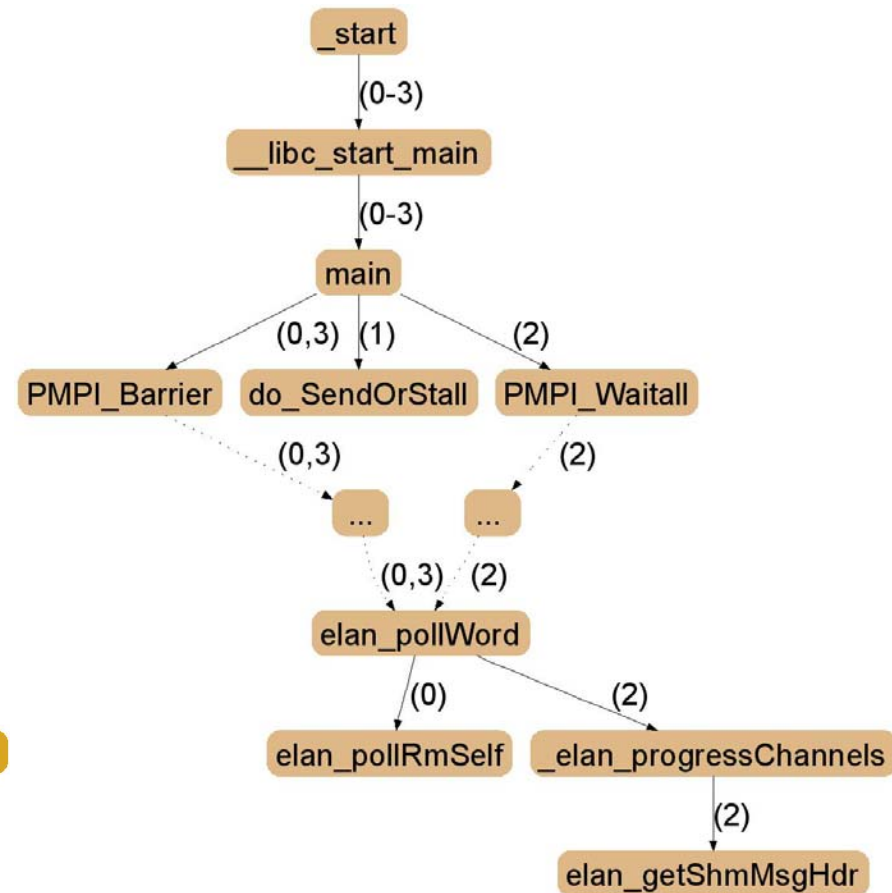
# We merge stack traces across tasks for 2D-trace/space analysis



# We use an intuitive visualization to present merged stack traces



**STAT**



**TotalView**

# 3D-trace/space/time analysis captures persistent behavior

Appl

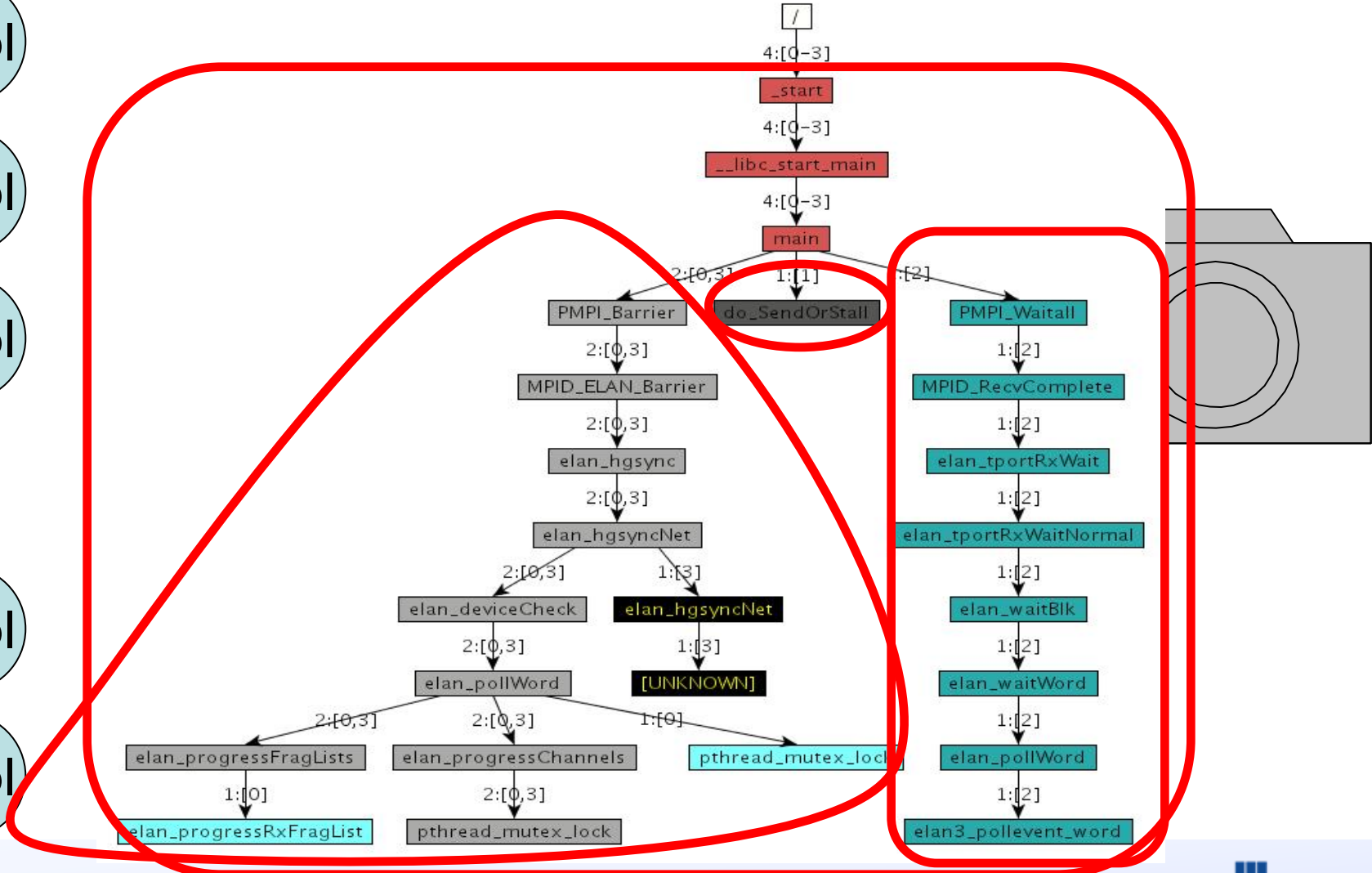
Appl

Appl

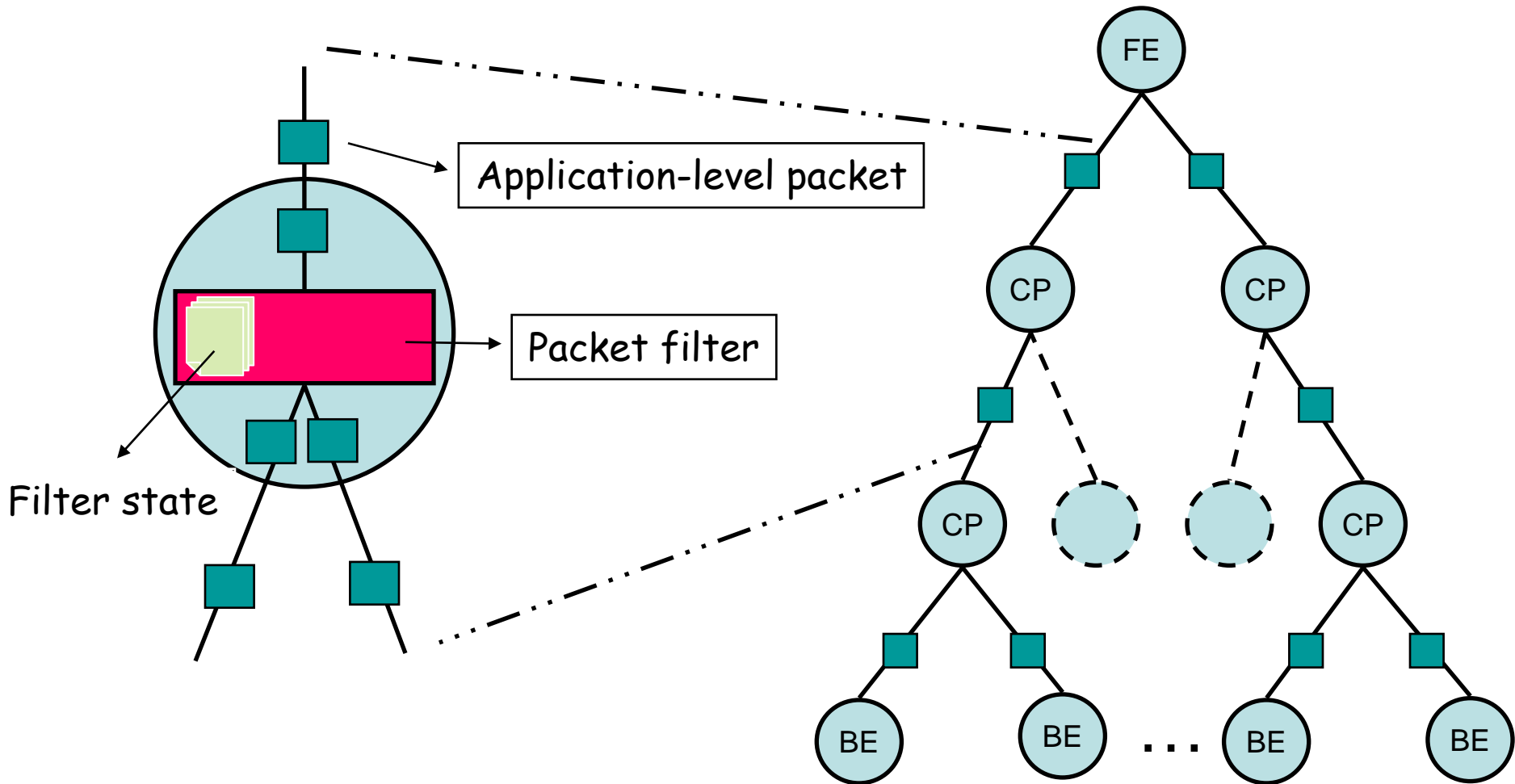
▪  
▪  
▪

Appl

Appl



# MRNet: an easy-to-use TBON

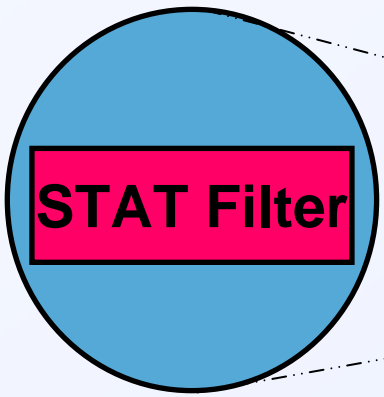
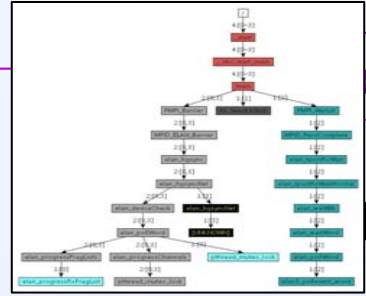


# STAT prototype using MRNet

front-end

(nt, freq.)

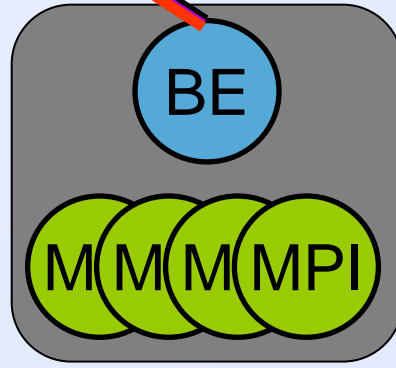
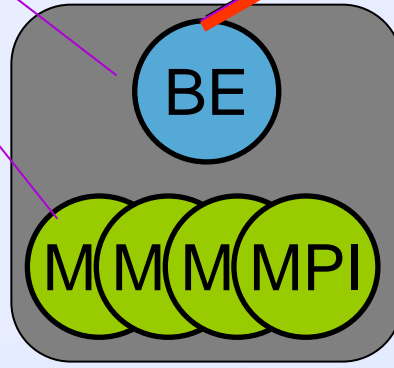
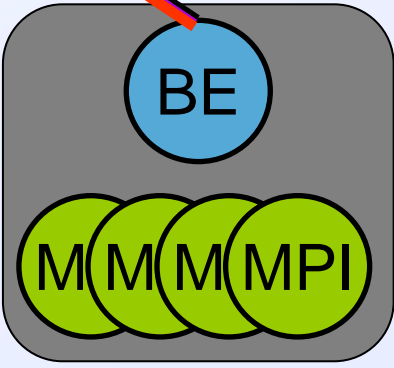
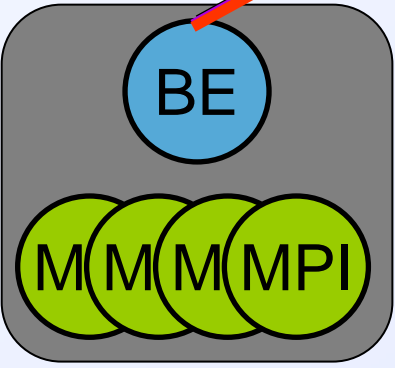
MRNet  
Communication  
Process



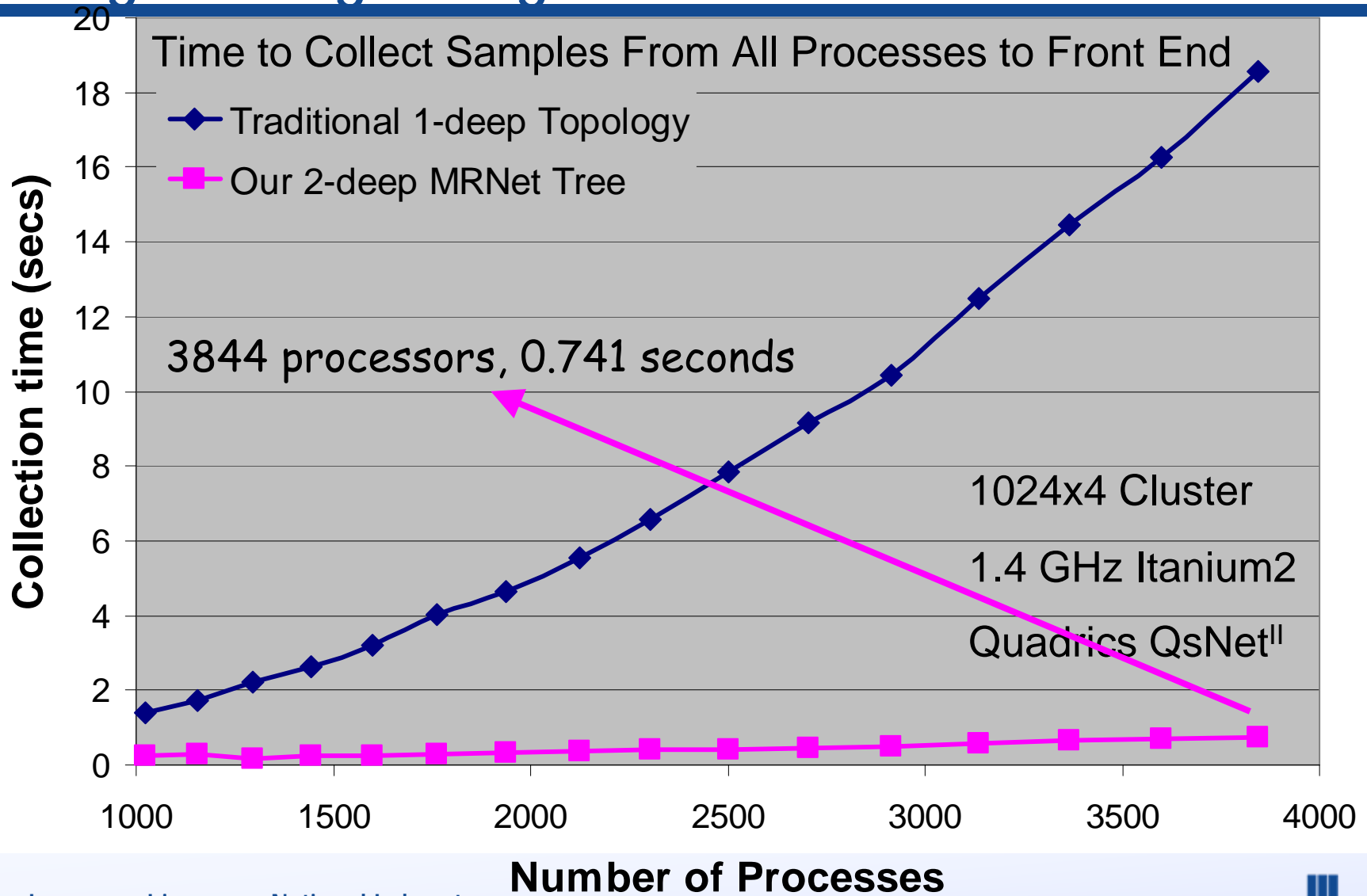
STAT Tool Daemon

Application Processes

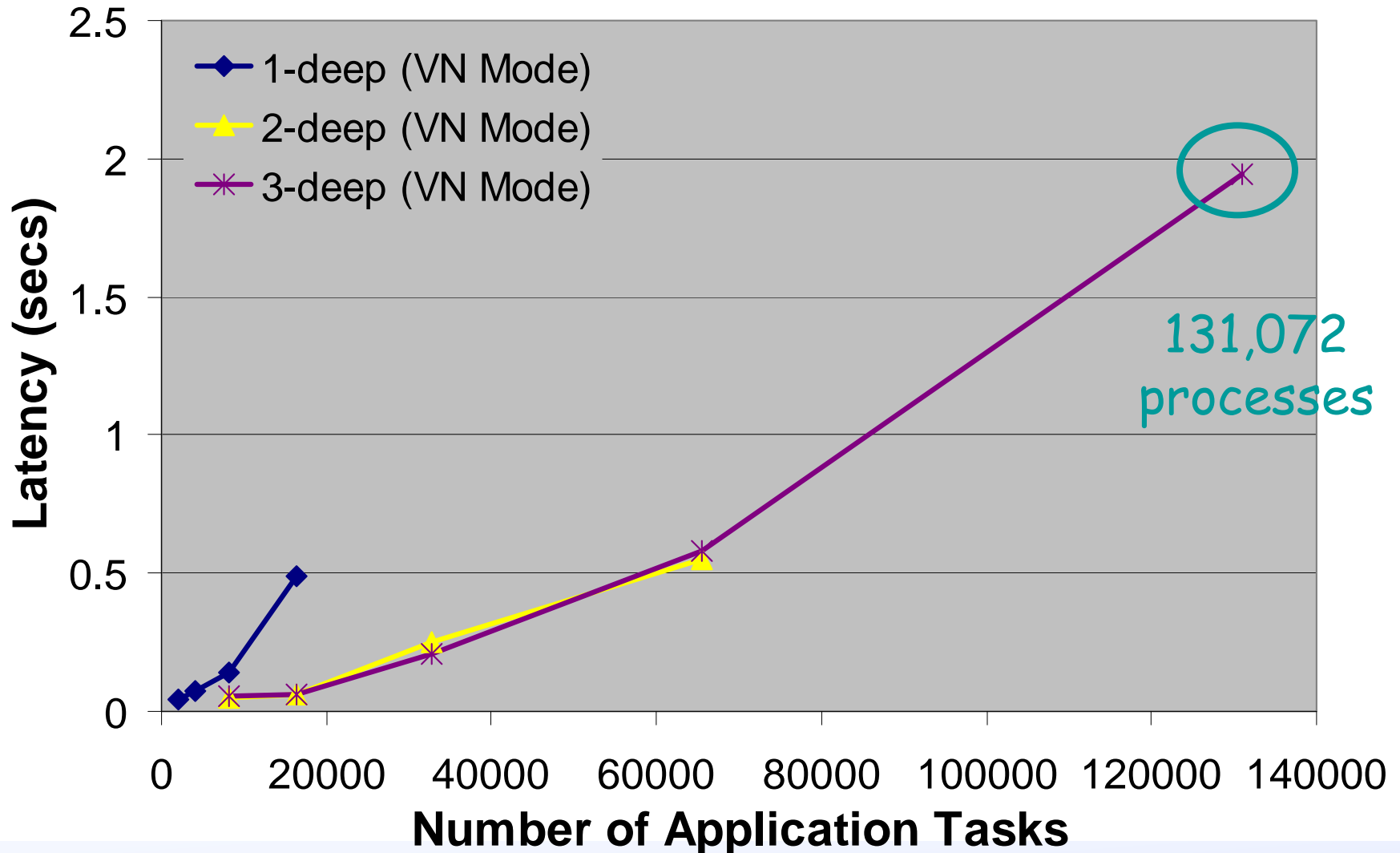
...



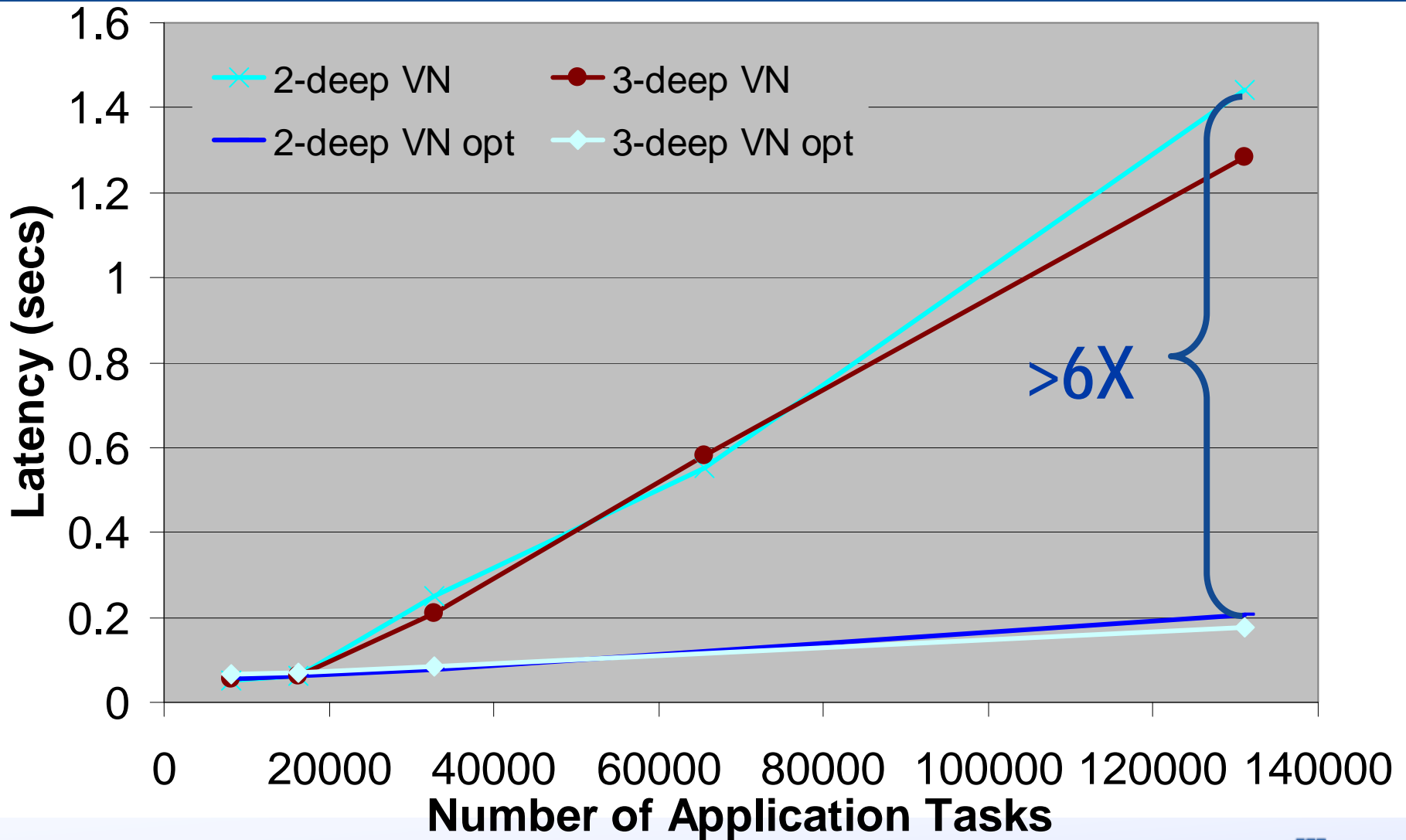
# Performance supports finding bugs arising at large scales



# Performance on BG/L supports debugging at very large scales



# Performance on BG/L supports debugging at very large scales

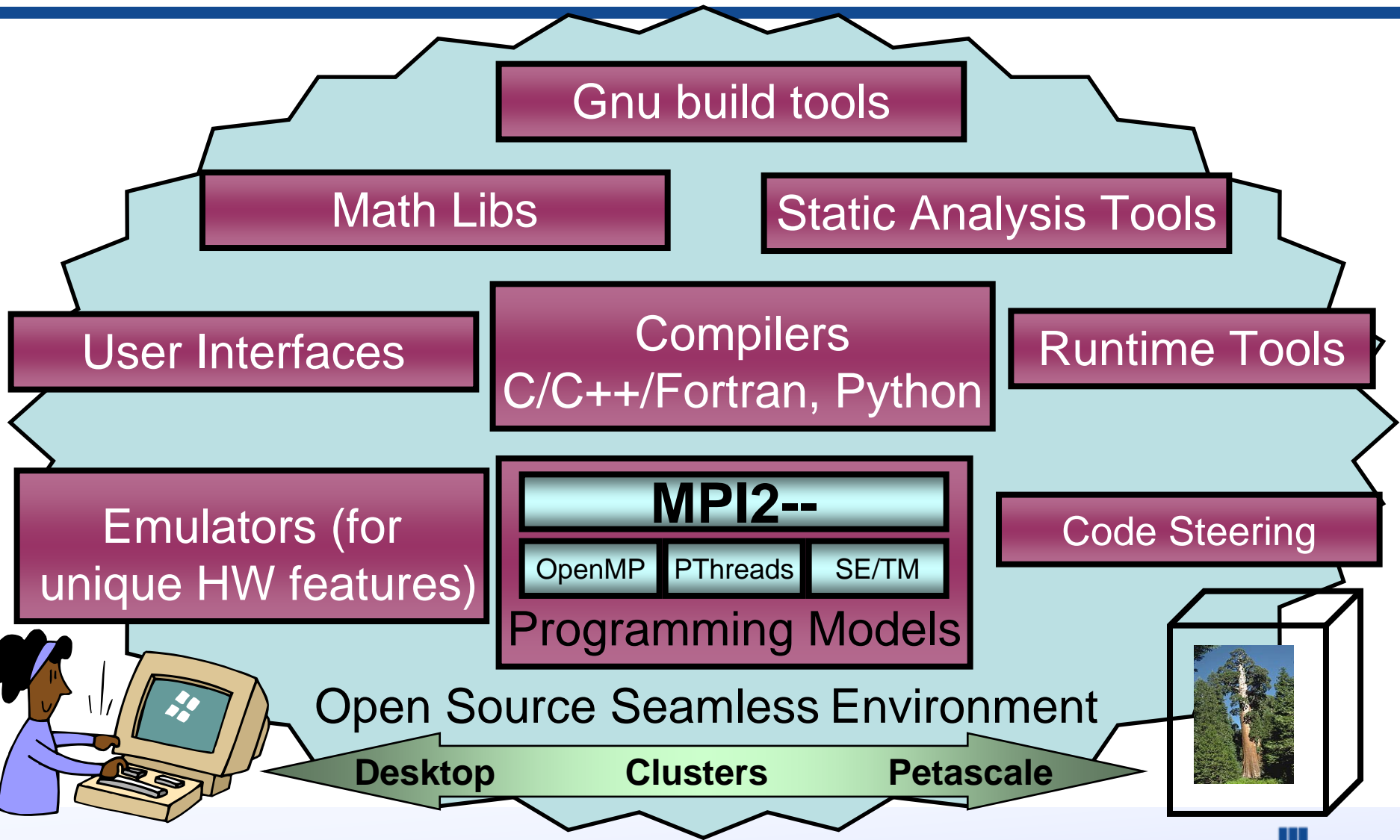


# Identified five critical performance tools concerns

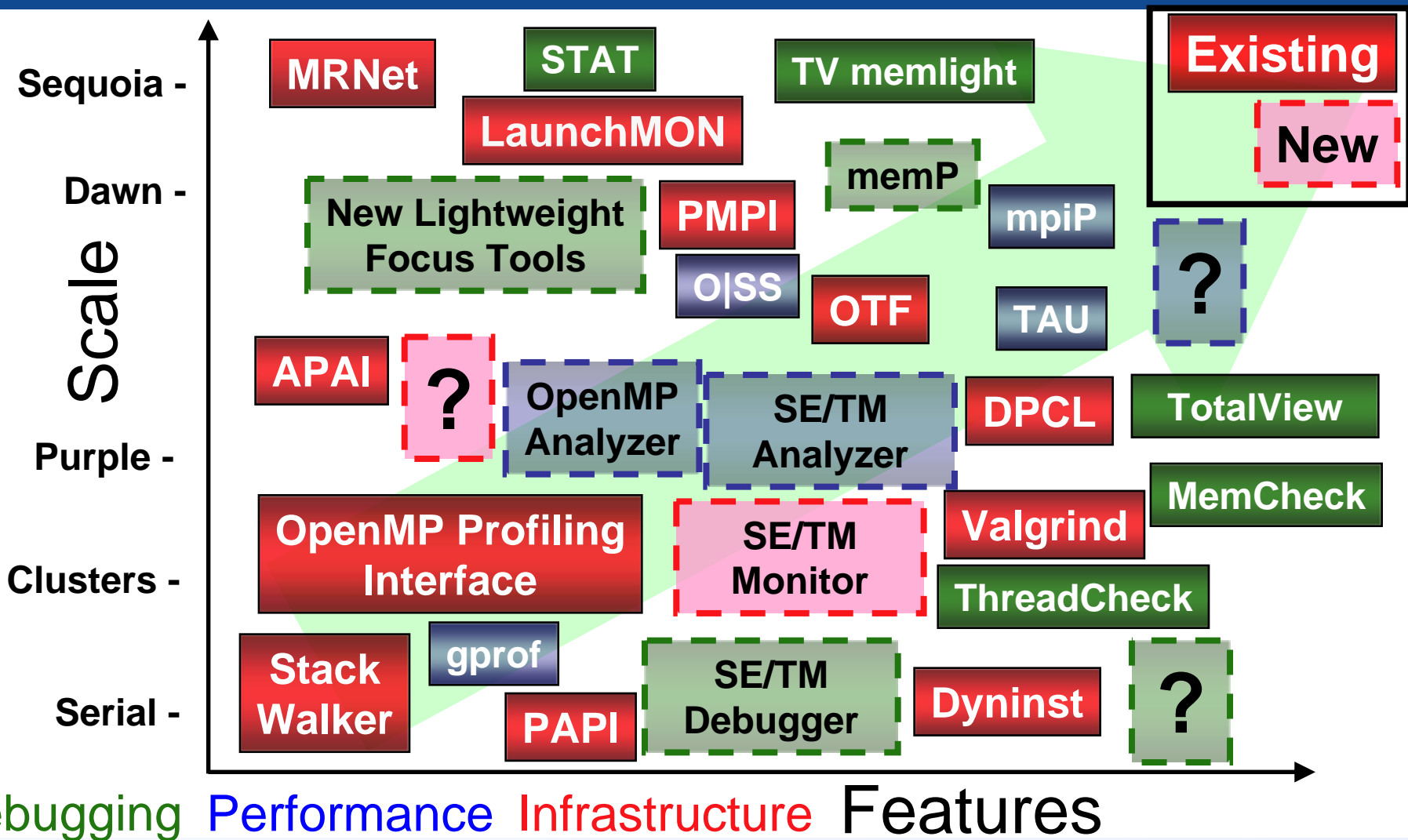


- Load imbalance will prevent significant performance gains
- **Develop automated load balance analysis tools**
- Need tools to assist data, thread & task placement decisions
- **Explore new placement tools, productize existing ones**
- Inadequate memory system performance analysis tools
- **Continue research into memory system performance**
- Possible divergences from expected performance
- **Improve application performance models**
- Tool infrastructure development & maintenance undervalued
- **Develop and maintain scalable community infrastructure**

# Codes developed on desktop; issues arise at scale



# Seamless environment requires scalable and portable infrastructure and tools



# ASC is planning for petascale successes



- Annual review & update of targets by teams and mgmt
- Rewritten every three years (at the start of each phase)
- Plan is applicable to multiple ASC petascale platforms
- Current research solving pressing issues
  - Power-aware high performance computing
  - Lightweight debuggers
  - Load balance analysis tool
  - Scalable, modular infrastructure

