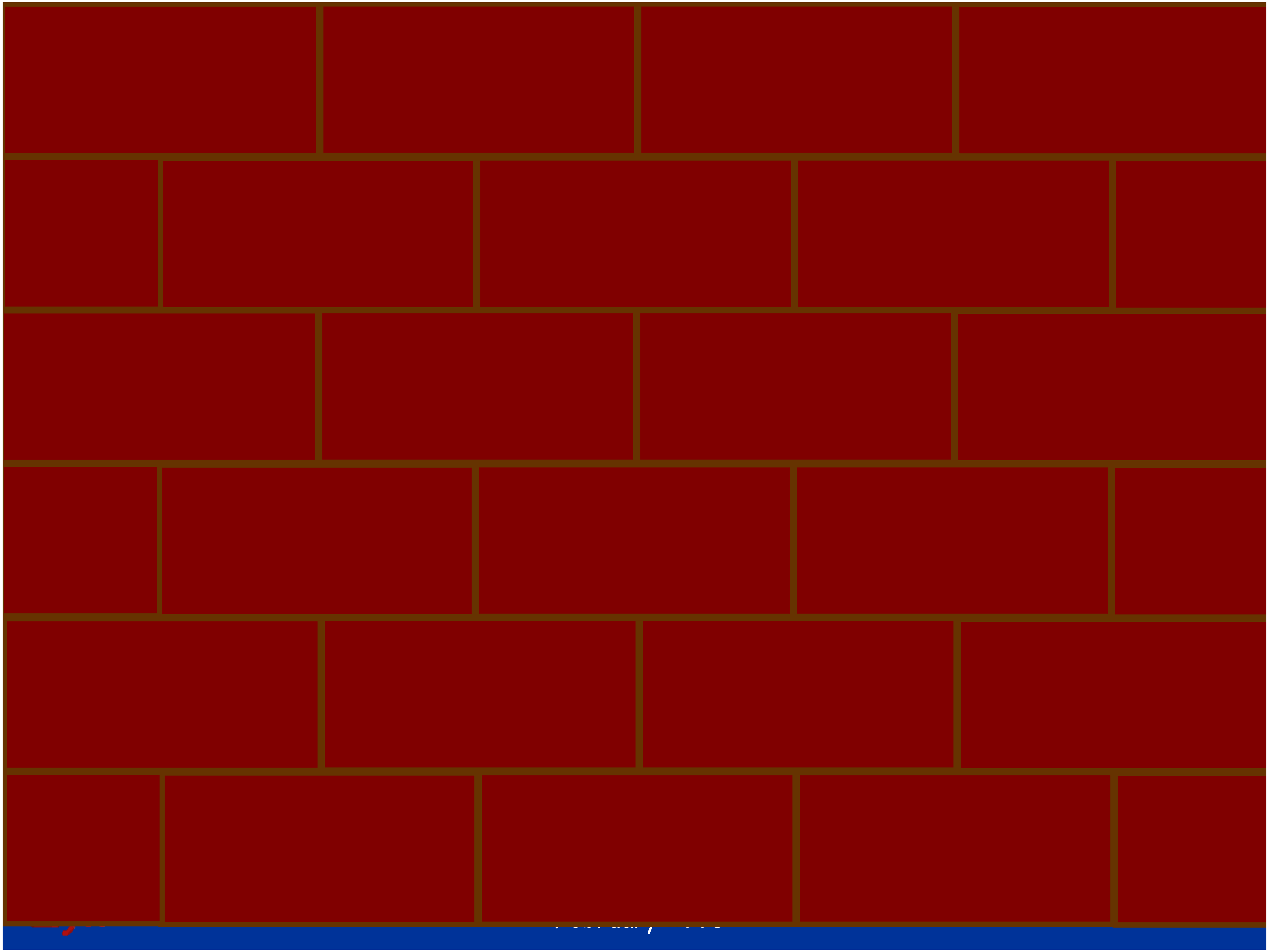
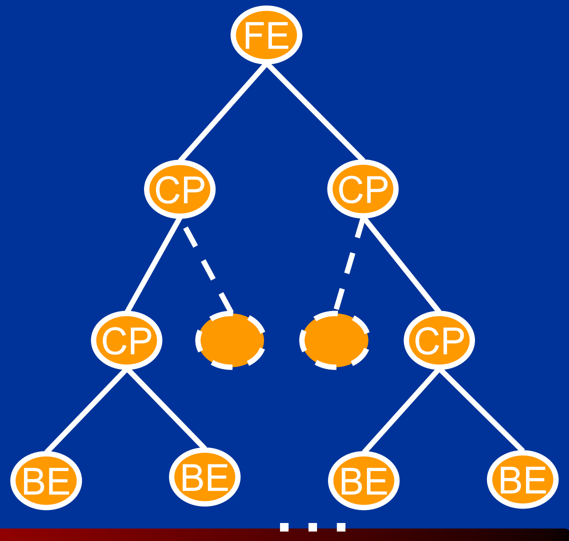


Overview

- Extremely large scale systems are here
- Effective, scalable programming is hard
- Start with a simple but powerful foundation: the Tree-based Overlay Networks (TBŌNs)

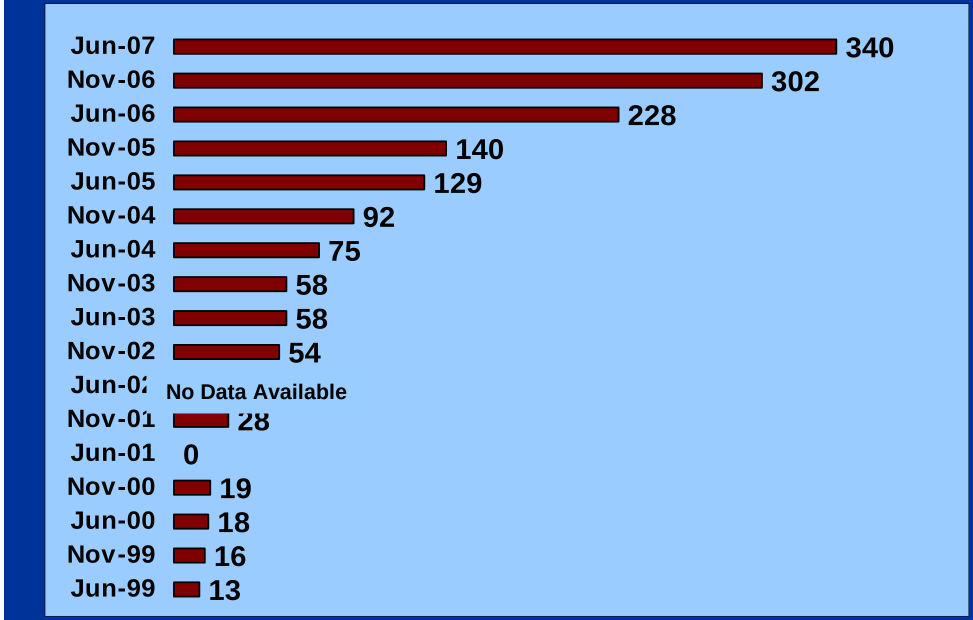


Overview

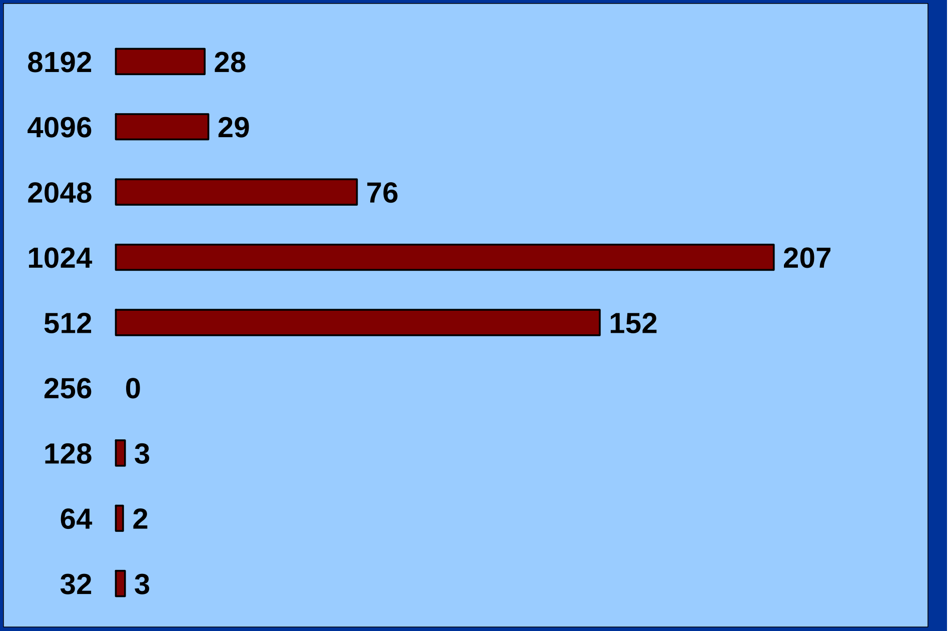
TBŌNs provide:

- An immediate path to scalable tools and infrastructure. Examples:
  - Paradyn Performance Tools
  - Vision algorithms
  - Stack trace analysis (new)
- A Research platform for new technologies:
  - New concepts in fault tolerance (no logs, no hot-backups).
  - As an framework for parallel applications
  - As a powerful alternative to the Map-Reduce idiom
  - As a generalized, scalable communication infrastructure

HPC Trends from TOP500 SUPERCOMPUTER SITES

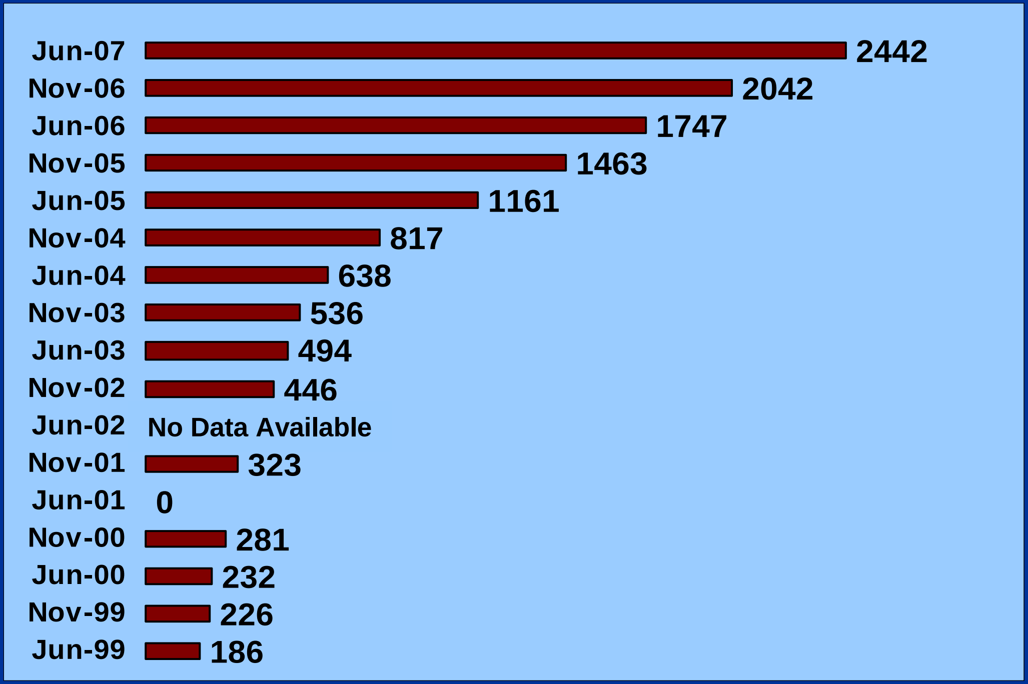


Systems Larger than 1024 Processors



06/2007 Processor Count Distribution

HPC Trends from TOP500 SUPERCOMPUTER SITES



Average Processor Counts

"I think that I shall never see
An algorithm lovely as a tree."

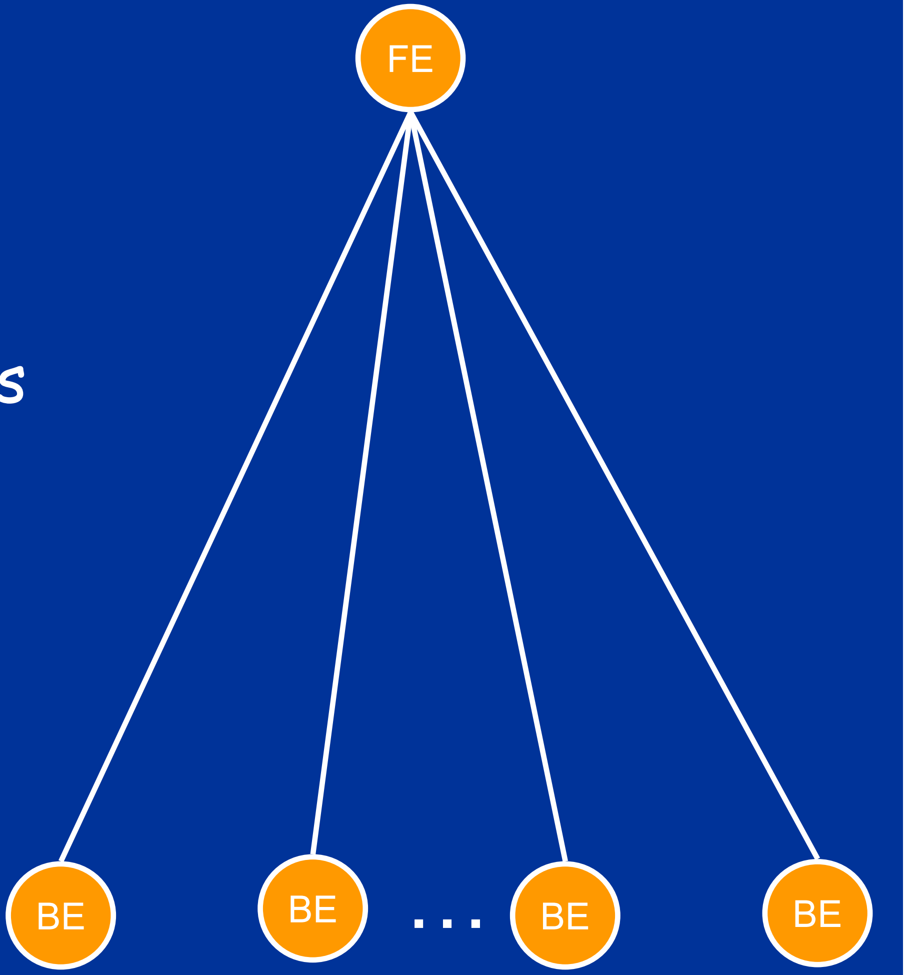
Trees by Joyce Kilmer (1919)

If you can formulate the problem so that it is hierarchically decomposed, you can probably make it run fast.

Distributed Control and Monitoring

- Hierarchical Topologies
  - Application Control
  - Data collection
  - Data centralization/analysis

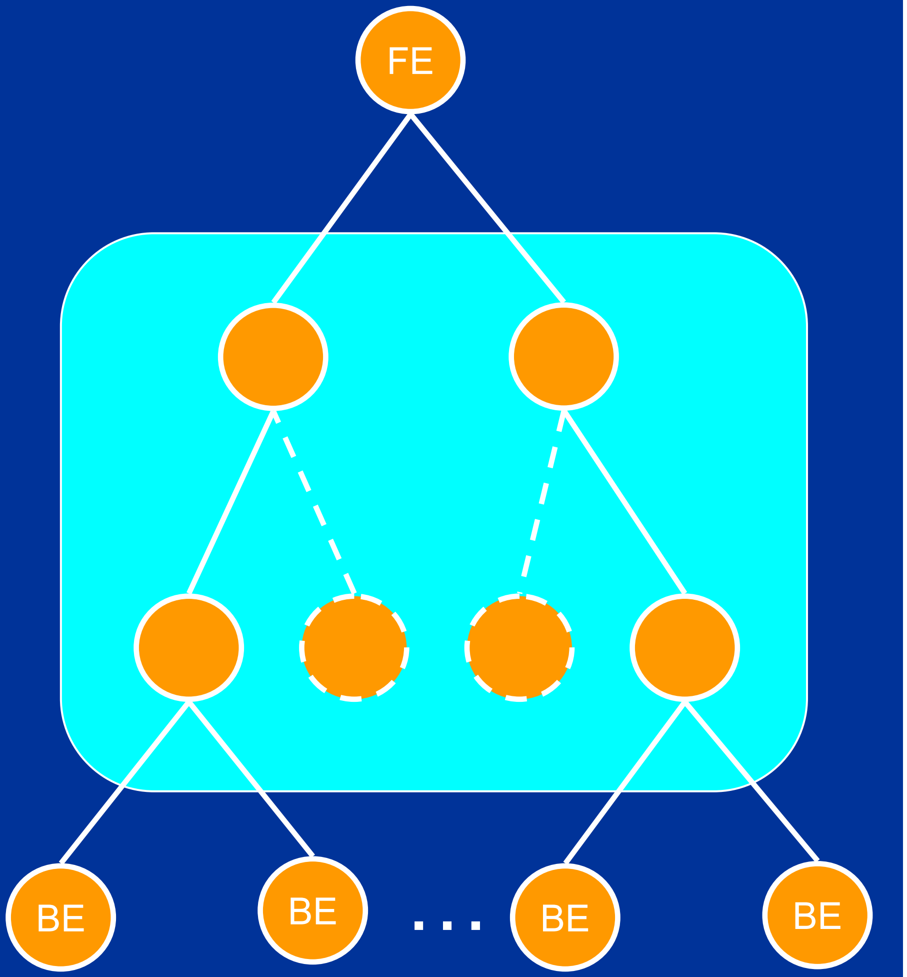
- As scale increases, front-end becomes bottleneck



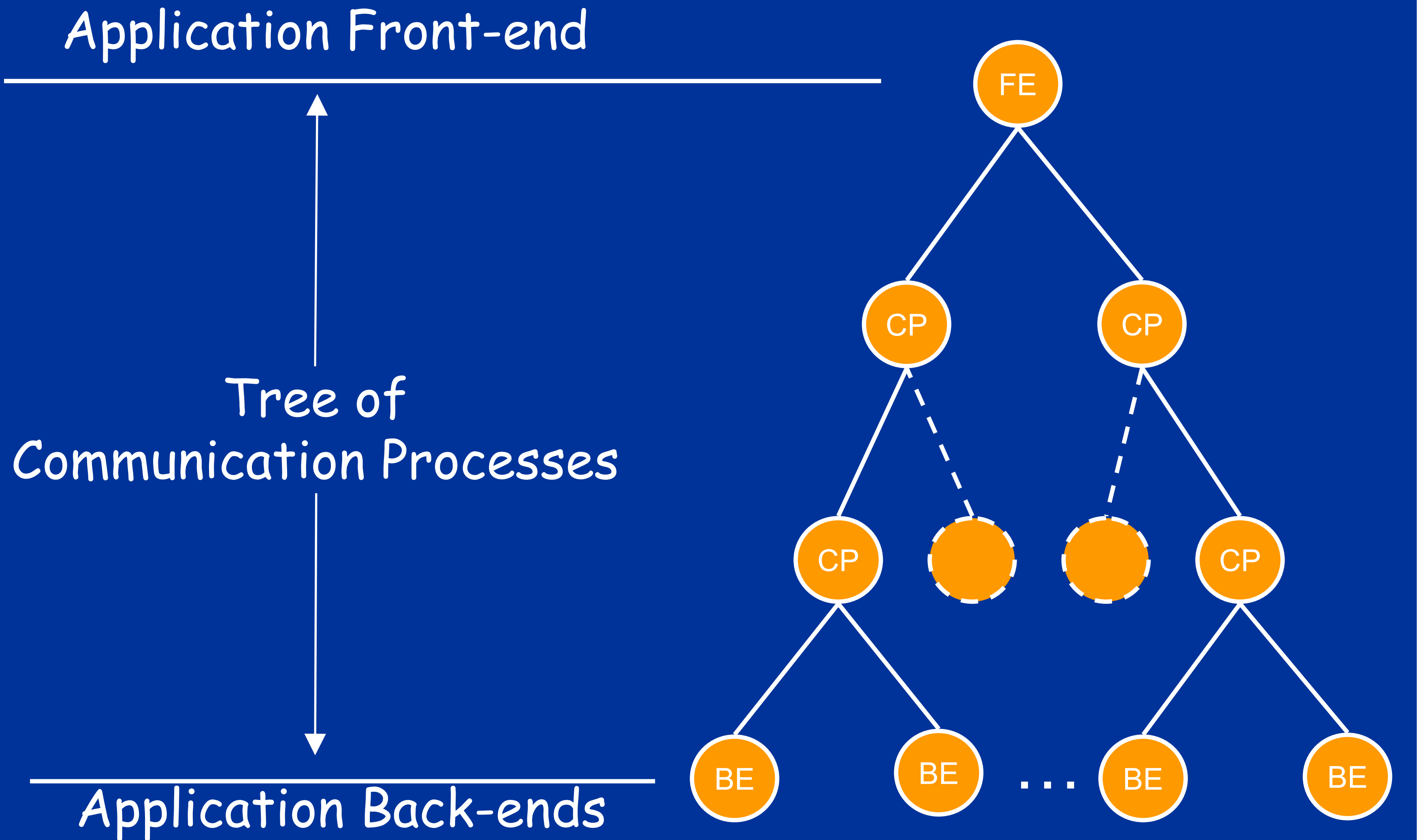
TBŌNs for Scalable Systems

TBŌNs for scalability

- Scalable multicast
- Scalable gather
- Scalable data aggregation



TBŌN Model

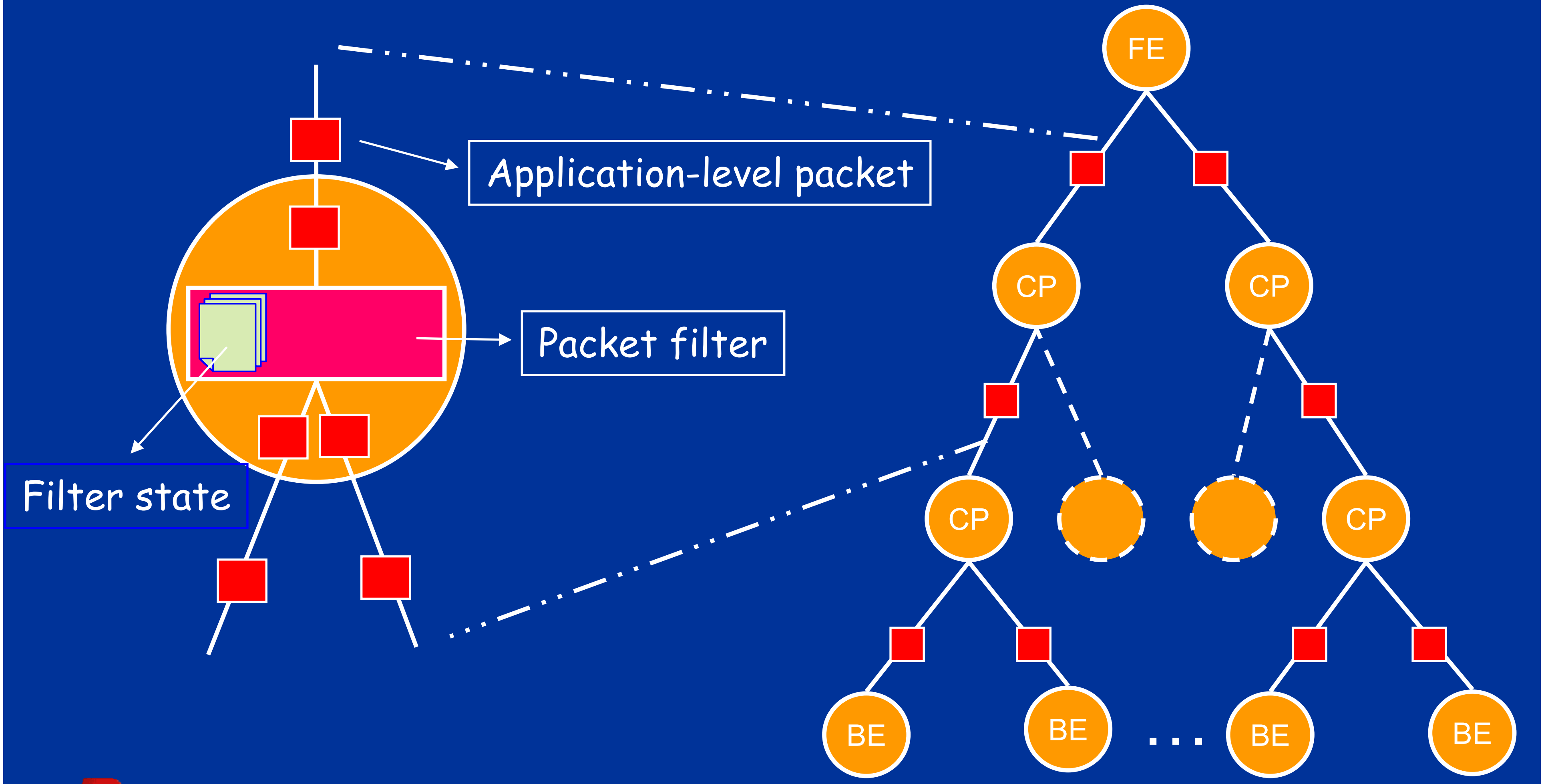


Application Front-end

Tree of
Communication Processes

Application Back-ends

MRNet: An Easy-to-Use TBŌN



TBŌNs at Work

- Multicast
  - ALMI [Pendarakis, Shi, Verma and Waldvogel '01]
  - End System Multicast [Chu, Rao, Seshan and Zhang '02]
  - Overcast [Jannotti, Gifford, Johnson, Kaashoek and O'Toole '00]
  - RMX [Chawathe, McCanne and Brewer '00]
- Multicast/gather (reduction)
  - Bistro (no reduction) [Bhattacharjee et al '00]
  - Gathercast [Badrinath and Sudame '00]
  - Lilith [Evensky, Gentile, Camp, and Armstrong '97]
  - MRNet [Roth, Arnold and Miller '03]
  - Ygdrasil [Balle, Brett, Chen, LaFrance-Linden '02]
- Distributed monitoring/sensing
  - Ganglia [Sacerdoti, Katz, Massie, Culler '03]
  - Supermon (reduction) [Sottile and Minnich '02]
  - TAG (reduction) [Madden, Franklin, Hellerstein and Hong '02]

Example TBŌN Reductions

- Simple
  - Min, max, sum, count, average
  - Concatenate
- Complex
  - Clock synchronization [Roth, Arnold, Miller '03]
  - Time-aligned aggregation [Roth, Arnold,Miller '03]
  - Graph merging [Roth, Miller '05]
  - Equivalence relations [Roth, Arnold, Miller '03]
  - Mean-shift image segmentation [Arnold, Pack, Miller '06]
  - Stack Trace Analysis Tool [Wisconsin, LLNL]

Using MRNet for Tool Scalability

MRNet integrated into Paradyn

- Efficient tool startup
- Performance data analysis
- Scalable visualization
- Distributed Performance Consultant

Equivalence computations

- Graph merging
- Trace analysis
- Data clustering (image analysis)
- Scalable stack trace analysis

Problem of Tool Start-Up Latency

Tools often transfer a lot of data at start-up

- Debugger needs function names and addresses to set breakpoints by name
- Paradyn needs information about modules, functions, processes, threads, synchronization objects, call graph

Front-end:

- Just cannot keep up with data and control.

This is an example of an important scenario

© Barton P. Miller 2008

Scalable Tool Start-up (Paradyn)

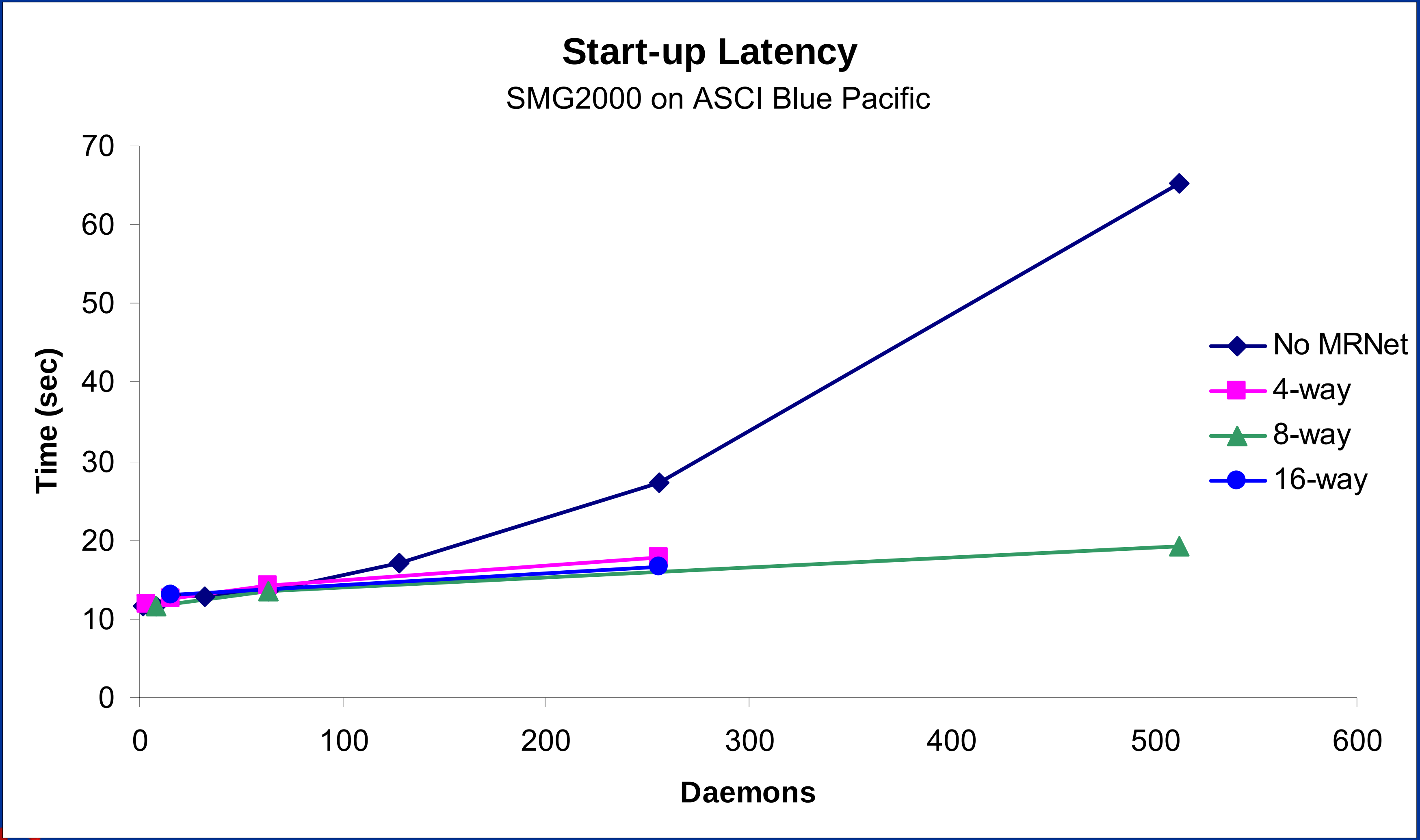
- Reduce redundant data transfer
  - Daemons deliver summary to front end using MRNet to find equivalence classes.
    - *Functions, modules, control-flow graph, call graph, etc.*
  - Front end asks equivalence class representatives for complete info.
  - Representative daemons send full info to front end.
- Reduce overhead of non-redundant data transfer
  - *Machine resources, daemon info, process info*
  - In-network concatenation of messages
    - Fewer send/recv operations
    - Front-end sees single message instead of many
- A log-time calculation: clock skew detection

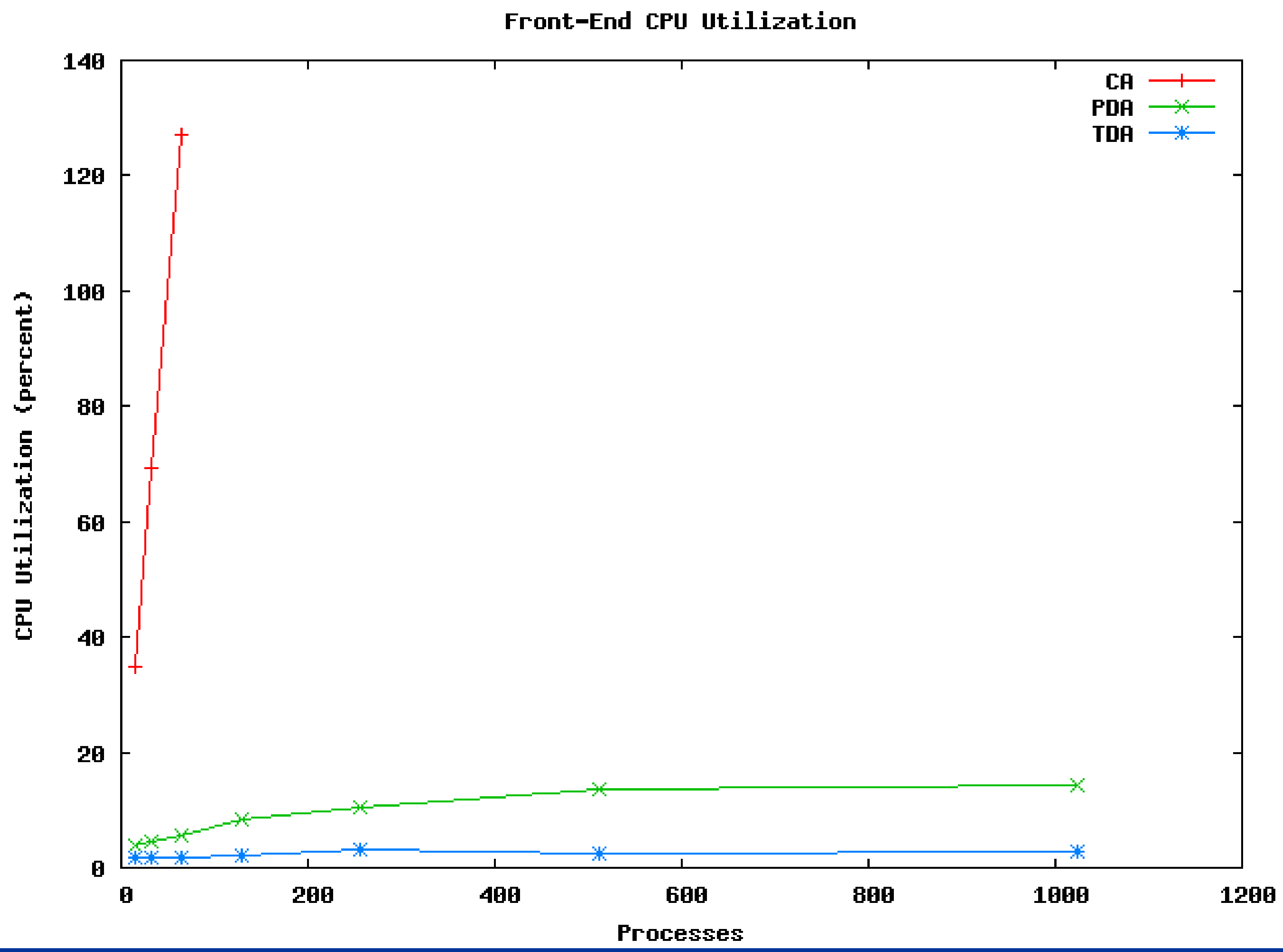
Clock Skew Detection Algorithm

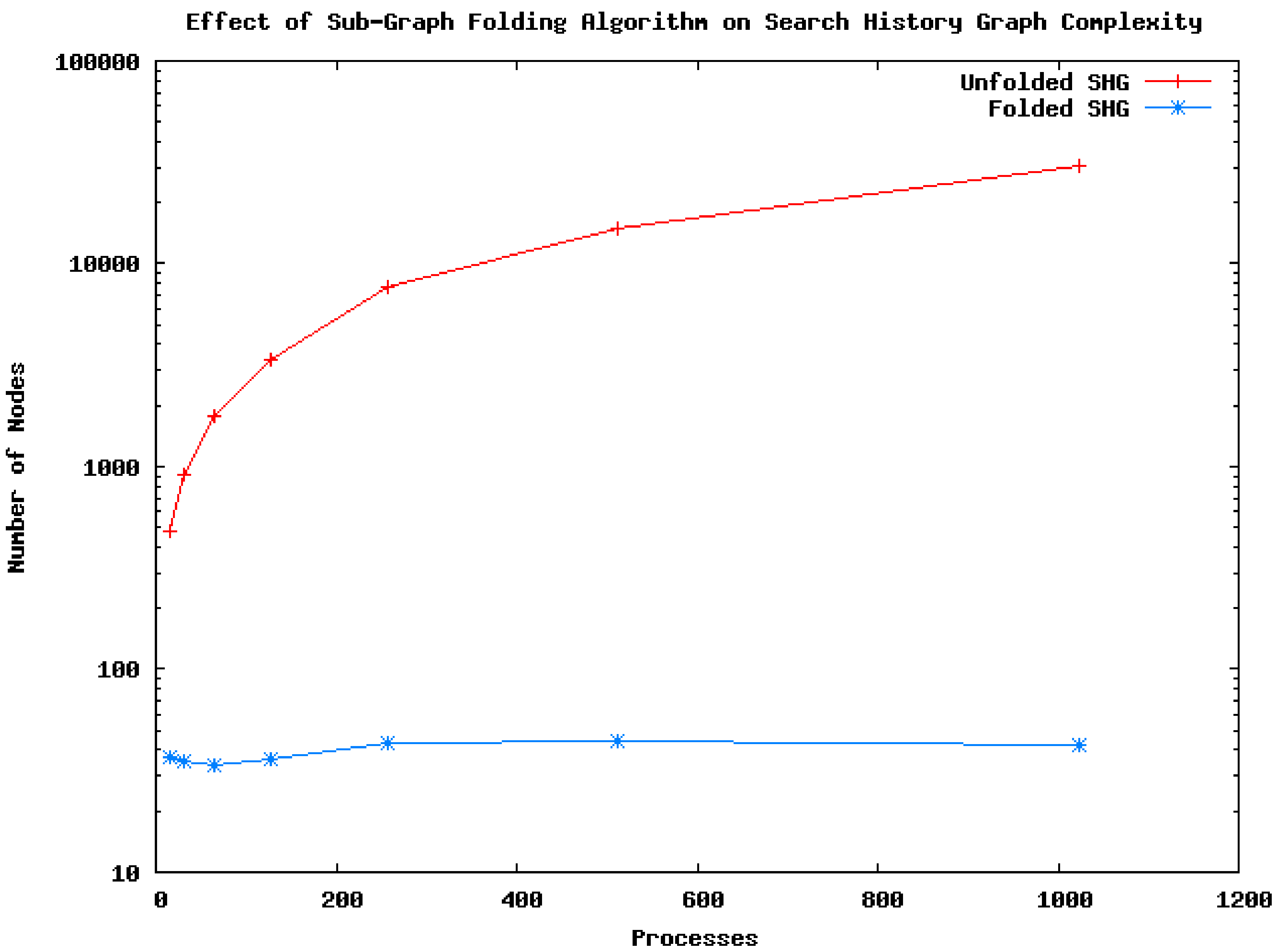
- Phase 1:
  - Repeated broadcast/reduce pairs to compute each process' clock skew with directly connected children
- Phase 2:
  - Upward sweep to compute cumulative clock skew to all reachable daemons

Paradyn Start-up Latency Results

Paradyn with SMG2000 on ASCI Blue Pacific




Front-End CPU Utilization
CPU Utilization (percent)
0
20
40
60
80
100
120
140
0
200
400
600
800
1000
1200
Processes
CA
PDA
TDA


Effect of Sub-Graph Folding Algorithm on Search History Graph Complexity
Unfolded SHG
Folded SHG
Number of Nodes
100000
10000
1000
100
10
0
200
400
600
800
1000
1200
Processes

Background: Performance Consultant

- Paradyn's automated performance diagnosis component: tells *why* there is a problem and points *where* to tune
- Automated search using dynamic instrumentation
  - Find performance problems with minimal user intervention
  - Insert and remove instrumentation code from processes as they run

  ⇒ Useful diagnosis results from a single run, with controlled overhead
- Tool daemons monitor and control application processes, tool front-end provides user interface

Background: Performance Consultant

- Search approach
  - Start with general, global experiments about application performance (e.g., CPU utilization is too high across all processes)
  - Collect performance data to test active experiments
  - Make decisions about experiments based on performance data
  - Refine search: if an experiment's performance data is above user-configurable threshold, create new, more specific experiments and repeat
- Performance data streams from tool daemons for analysis (i.e., refinement decisions)

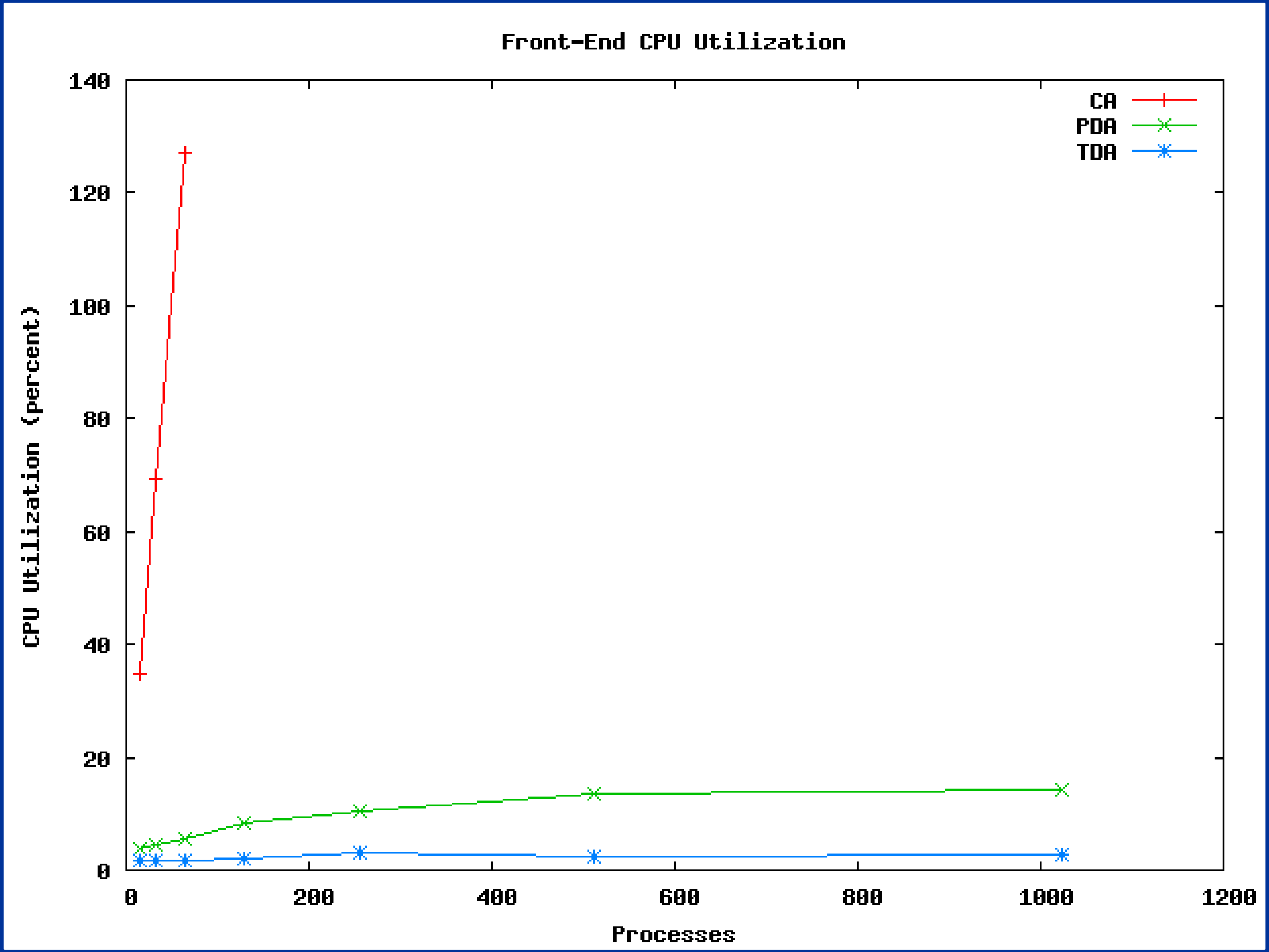
Three Approaches

- Centralized Approach (CA):
  - All performance data sent to the front end and all control from the front end.
- Partially Distributed Approach (PDA):
  - Global experiments (across all processes) monitored and controlled from the front-end using MRNet.
  - Local Performance Consultants on each node to look for local bottlenecks.
- Truly Distributed Approach (TDA):
  - Only local PC's on each node to monitor and search.
  - Global results from merging local data, using MRNet.

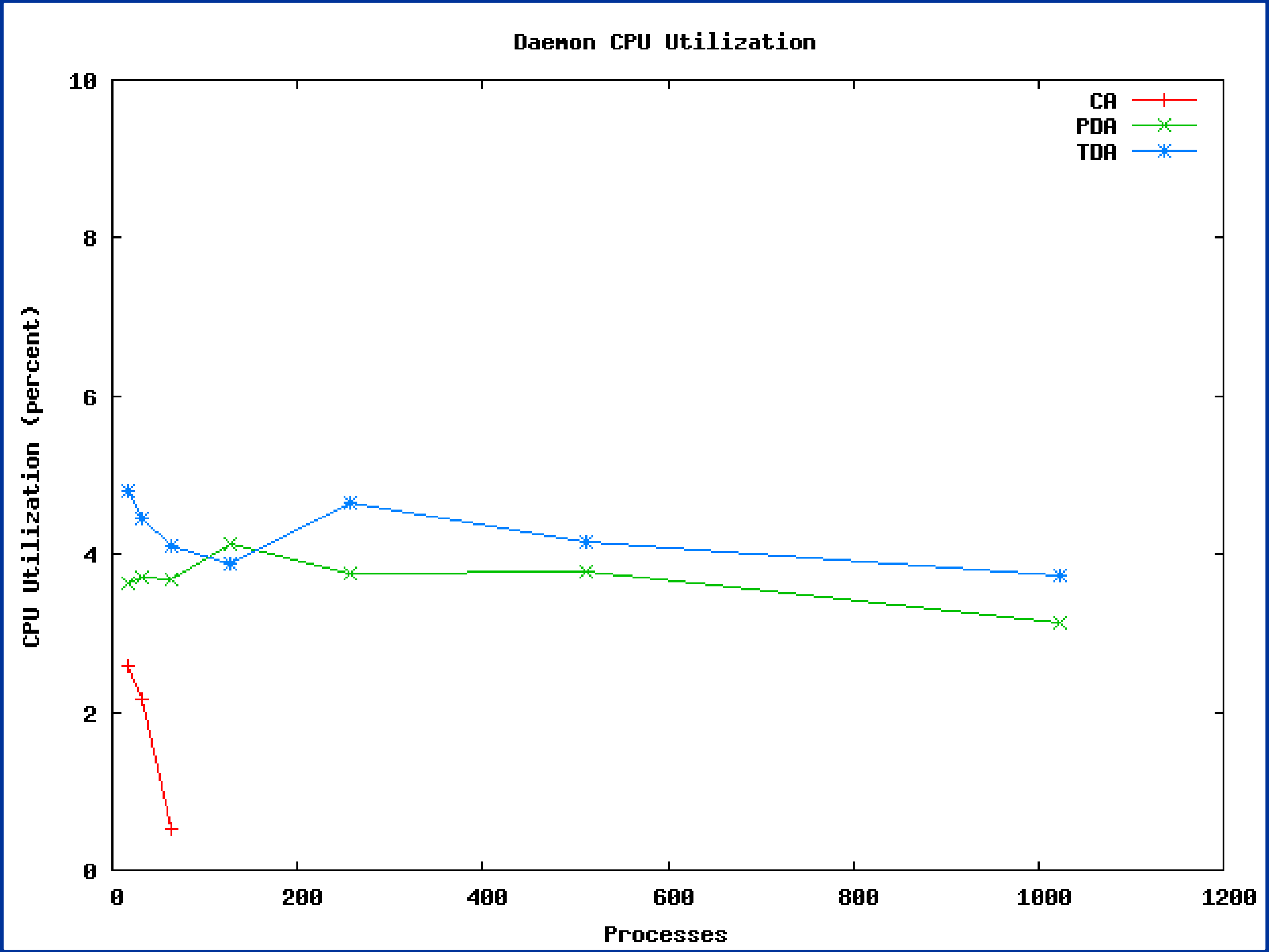
Evaluation: Experimental Environment

- LLNL MCR cluster
  - 1152 nodes (1048 compute nodes)
  - Two 2.4 GHz Intel Xeons per node
  - 4 GB memory per node
  - Quadrics Elan3 interconnect (fat tree)
  - Lustre parallel file system
- su3_rmd
  - Quantum chromodynamics pure lattice gauge theory code from MILC collaboration
  - C, MPI
  - Weak scaling scalability study

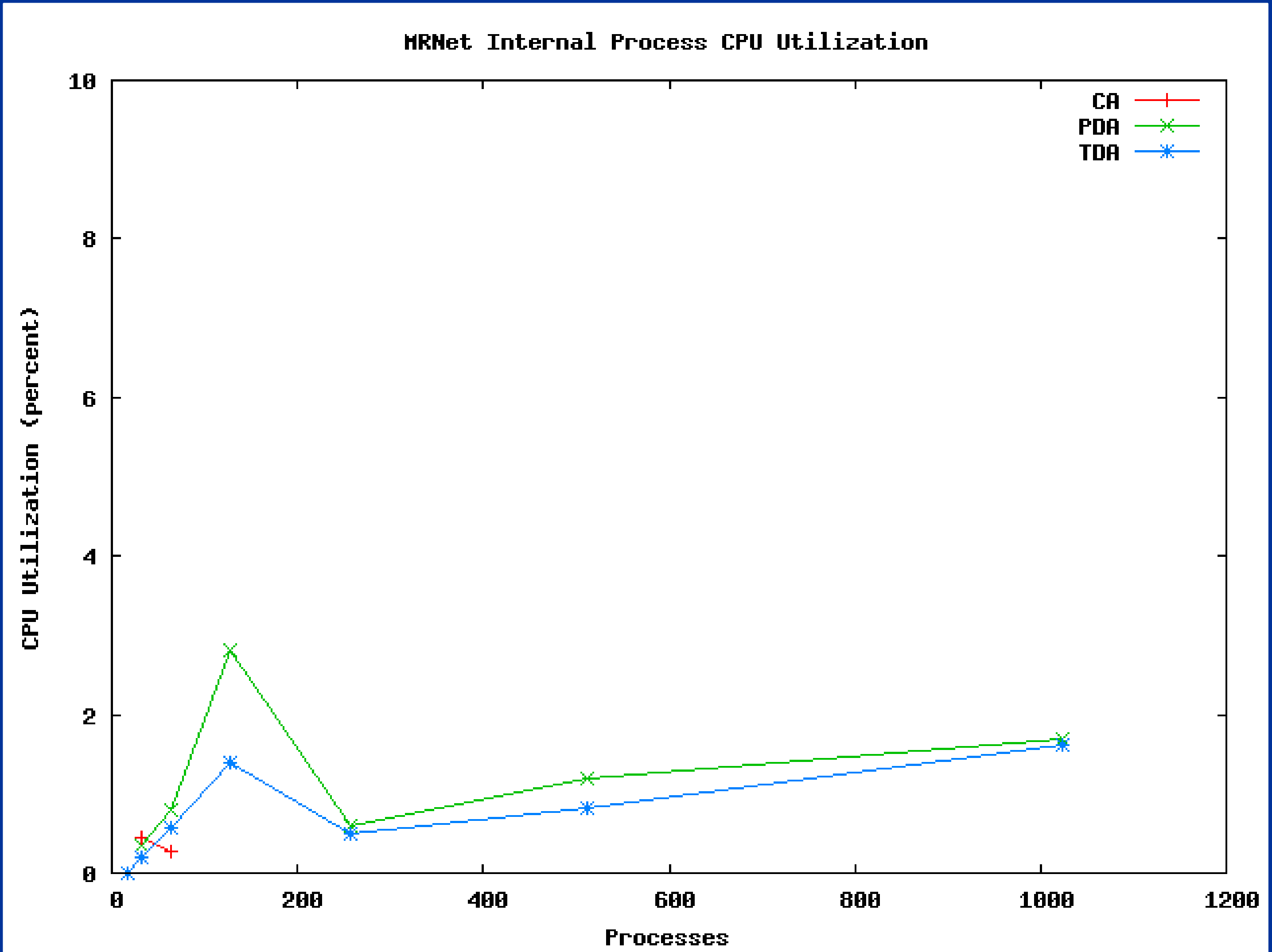
Evaluation: DPC Front-End CPU Load



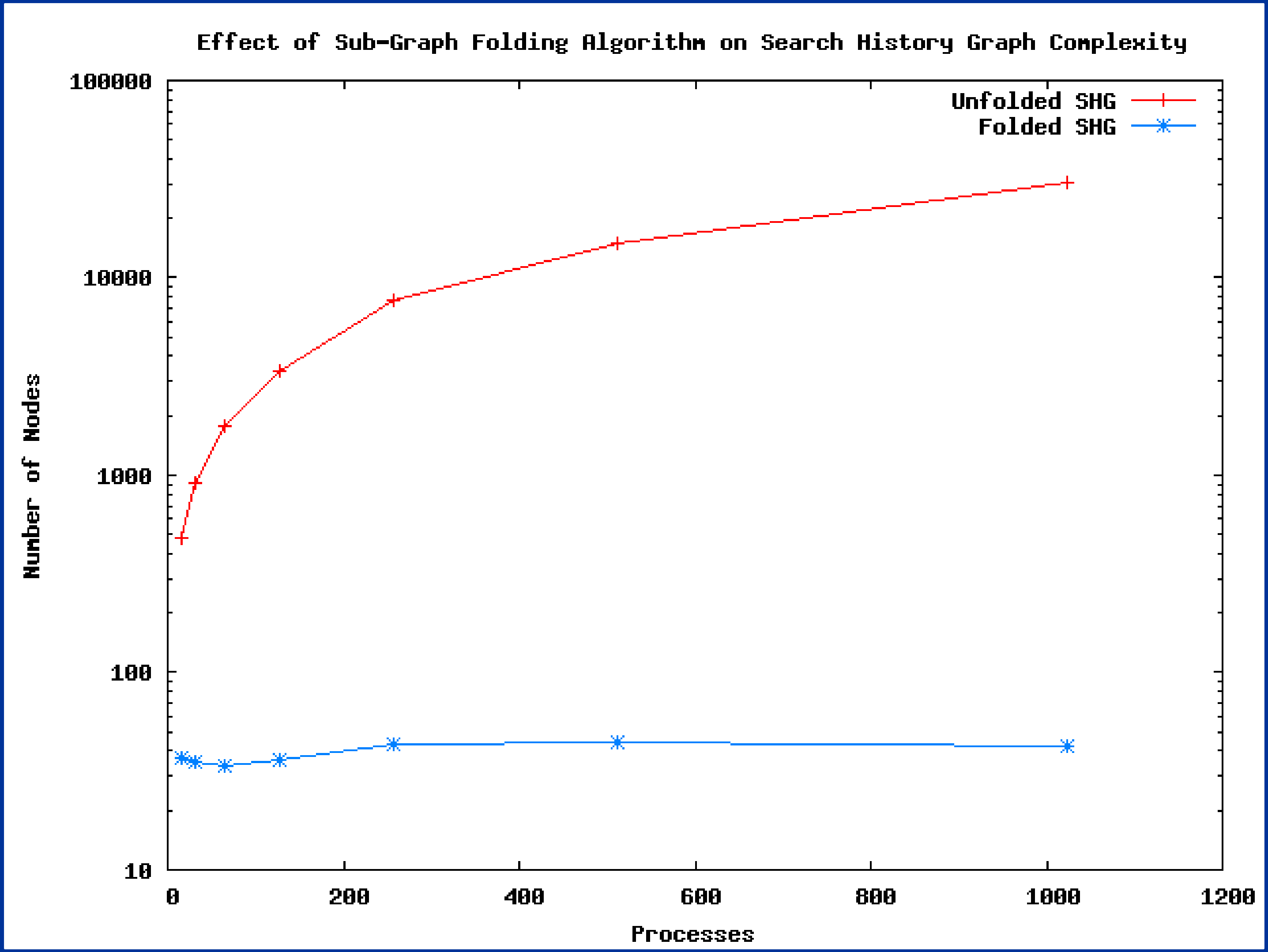
Evaluation: DPC Daemon CPU Load



Evaluation: DPC MRNet CPU Load

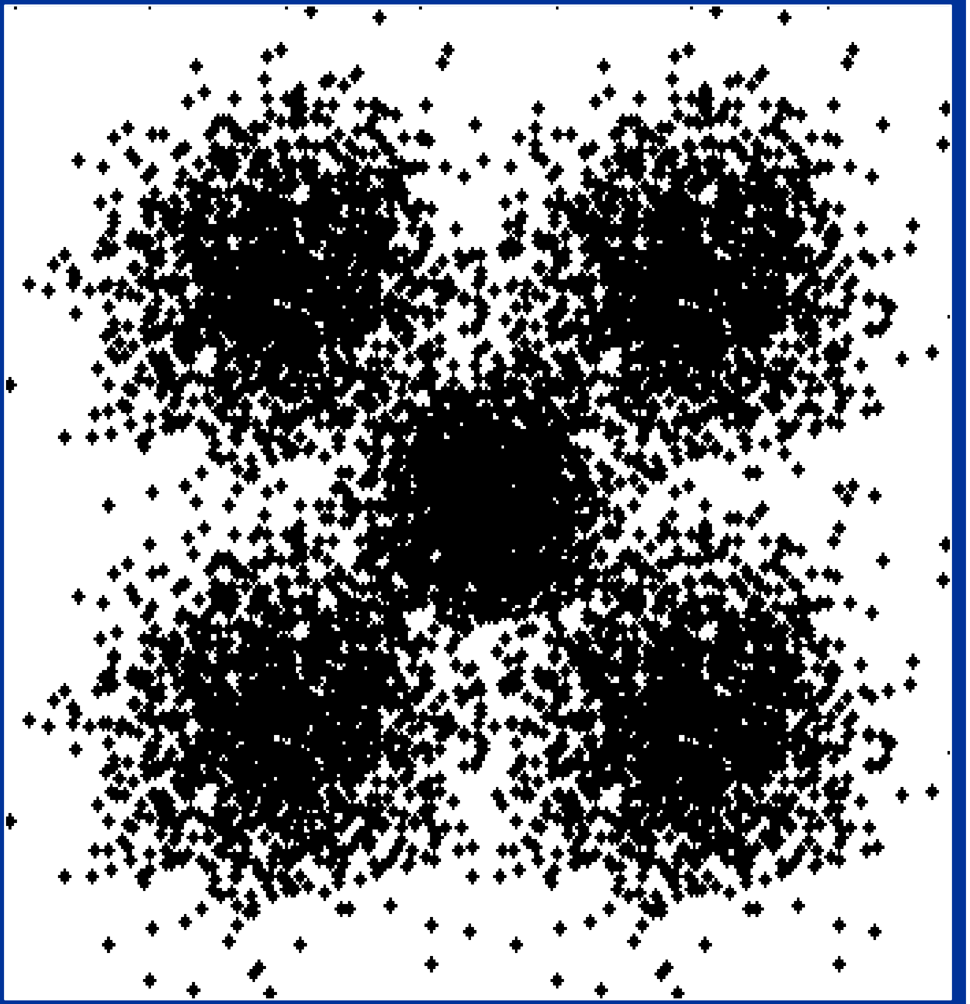


Evaluation: SGFA

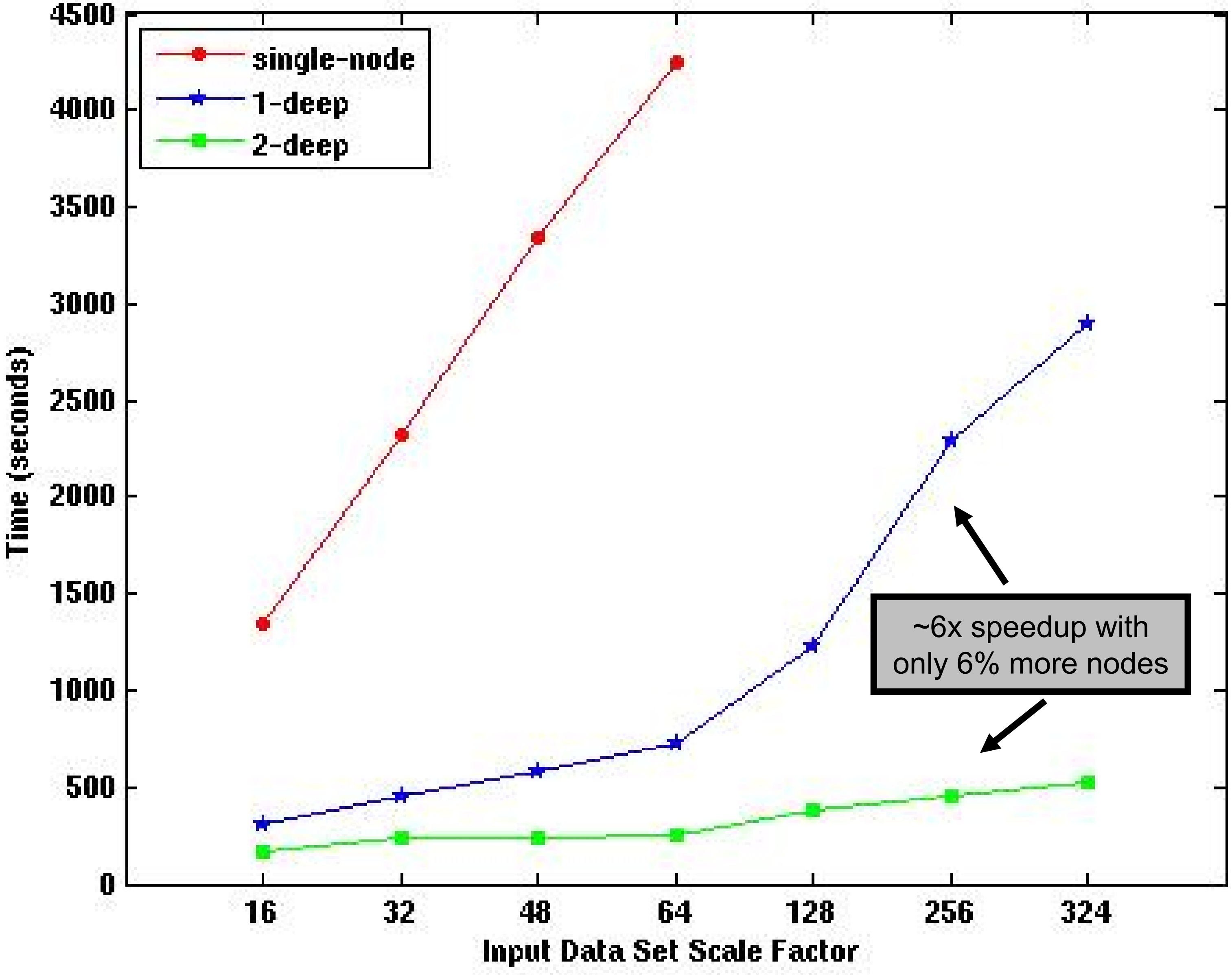


TBŌNs for Scalable Aps: Mean-Shift Algorithm

- Cluster points in feature spaces
- Useful for image segmentation
- Prohibitively expensive as feature space complexity increases



TBŌNs for Scalable Aps: Mean-Shift Algorithm



~6x speedup with only 6% more nodes

Recent Project: Peta-scalable Tools

With: Dong Ahn, Greg Lee, Martin Schulz, Bronis de Supinski @ LLNL

Stack Trace Analysis Tool (STAT)

- Data representation
- Data analyses
- Visualization of results

Build a simple, useful tool that works at scale.

TotalView on BG/L - 4096 Processes

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

Typical debug session includes many interactions

4096 is only 3% of BG/L!

Debugger Scalability Challenges

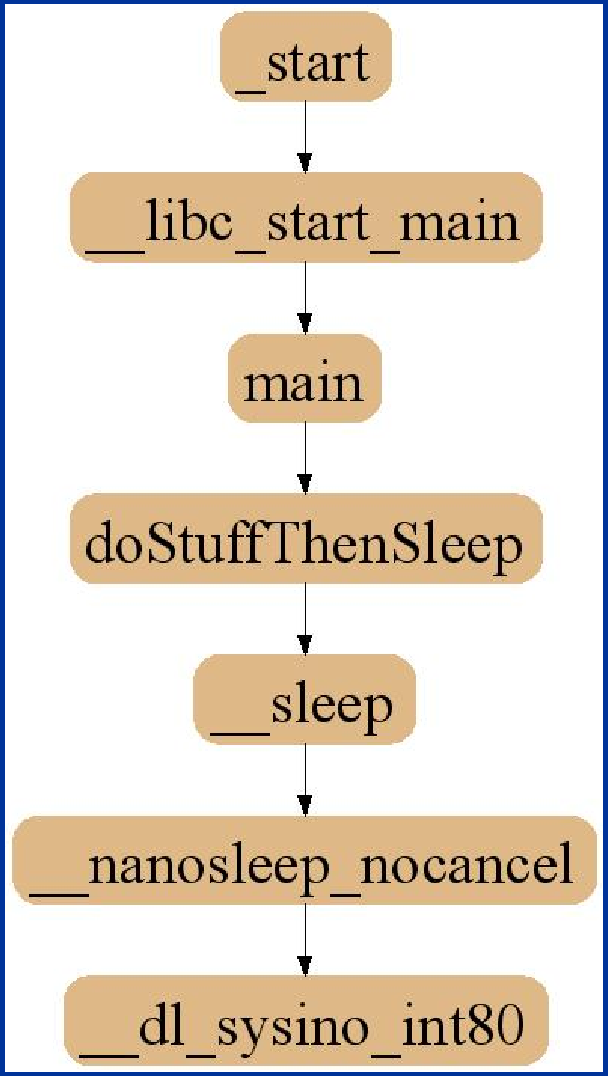
- Large volumes of debug data
- Many threads of control at front-end
- Vendor licensing limitations

- Approach: scalable, lightweight debugger
  - Reduce exploration space to small subset
  - Full-featured debugger for deeper digging

STAT Approach

- Sample application stack traces
- Merge/analyze traces:
  - Discover equivalent process behavior
  - Group similar processes
  - Facilitate scalable analysis/data presentation
- Leverage TBŌN model (MRNet)

Singleton Stack Trace

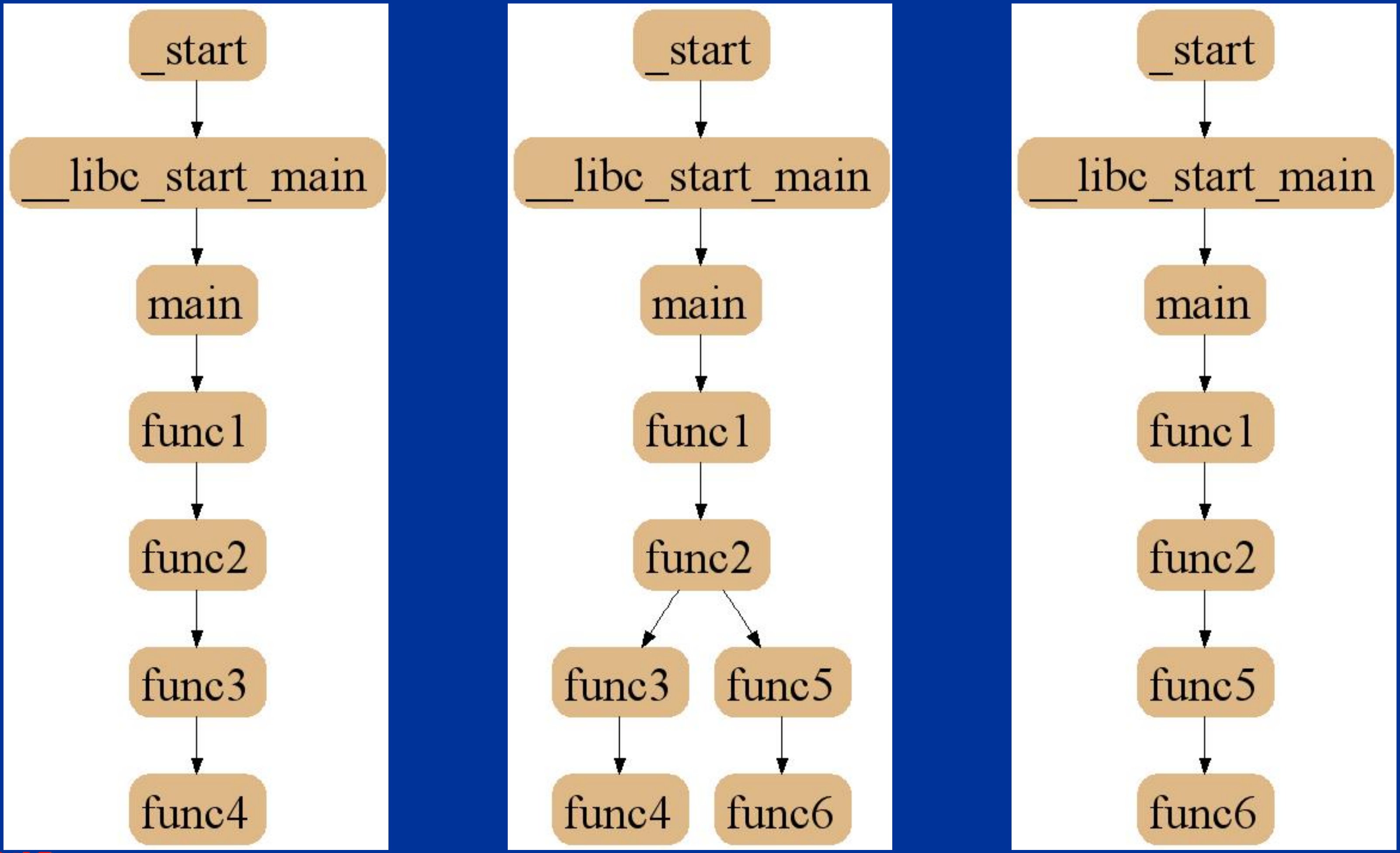


Appl.

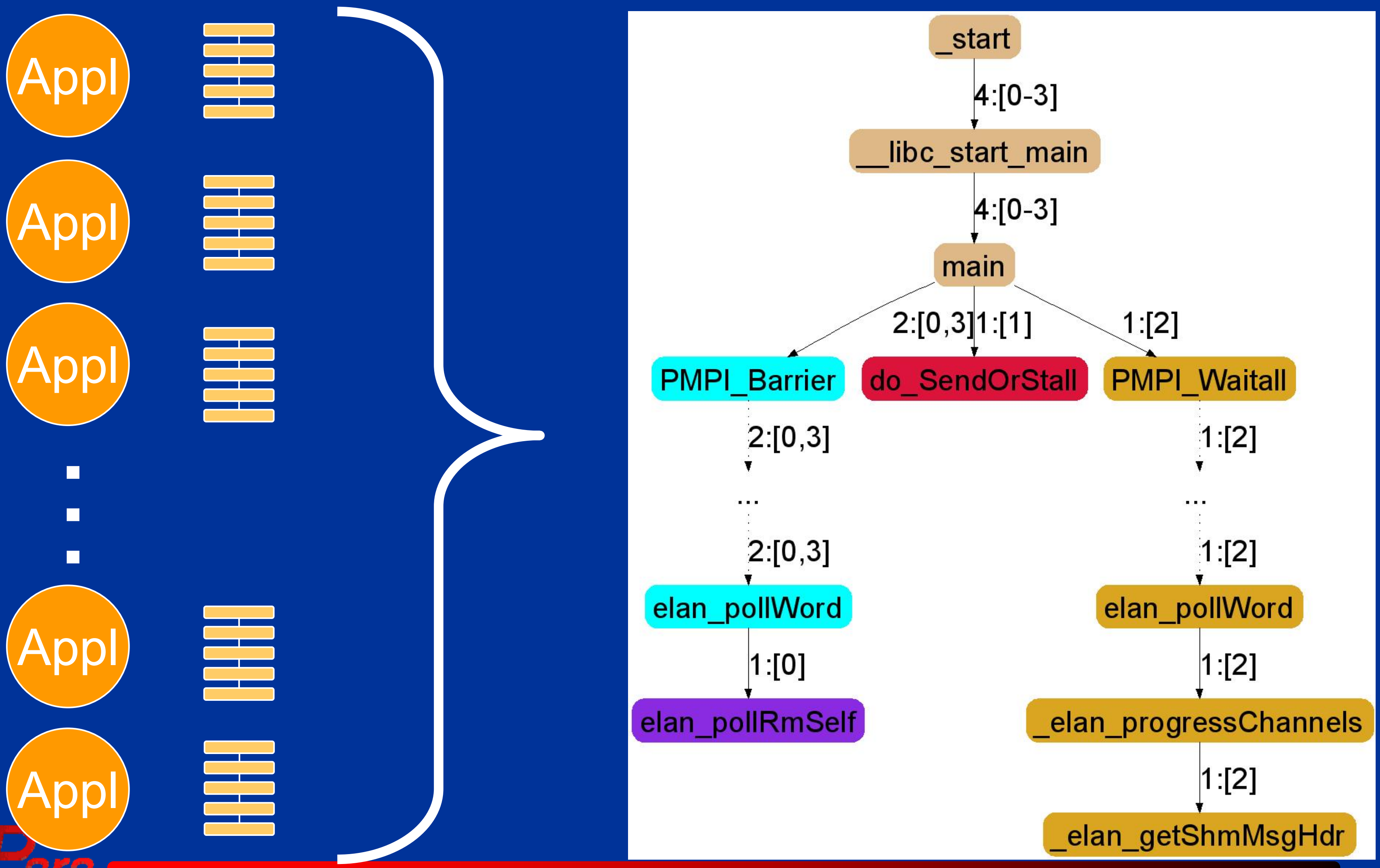
Merging Stack Traces

- Multiple traces over space or time
- Create call graph prefix tree
  - Compressed representation
  - Scalable visualization
  - Scalable analysis

Merging Stack Traces



2D-Trace/Space Analysis



Appl

Appl

Appl

Appl

Appl

_start

4:[0-3]

__libc_start_main

4:[0-3]

main

2:[0,3]

1:[1]

1:[2]

PMPI_Barrier

do_SendOrStall

PMPI_Waitall

2:[0,3]

...

2:[0,3]

elan_pollWord

1:[0]

elan_pollRmSelf

1:[2]

...

1:[2]

elan_pollWord

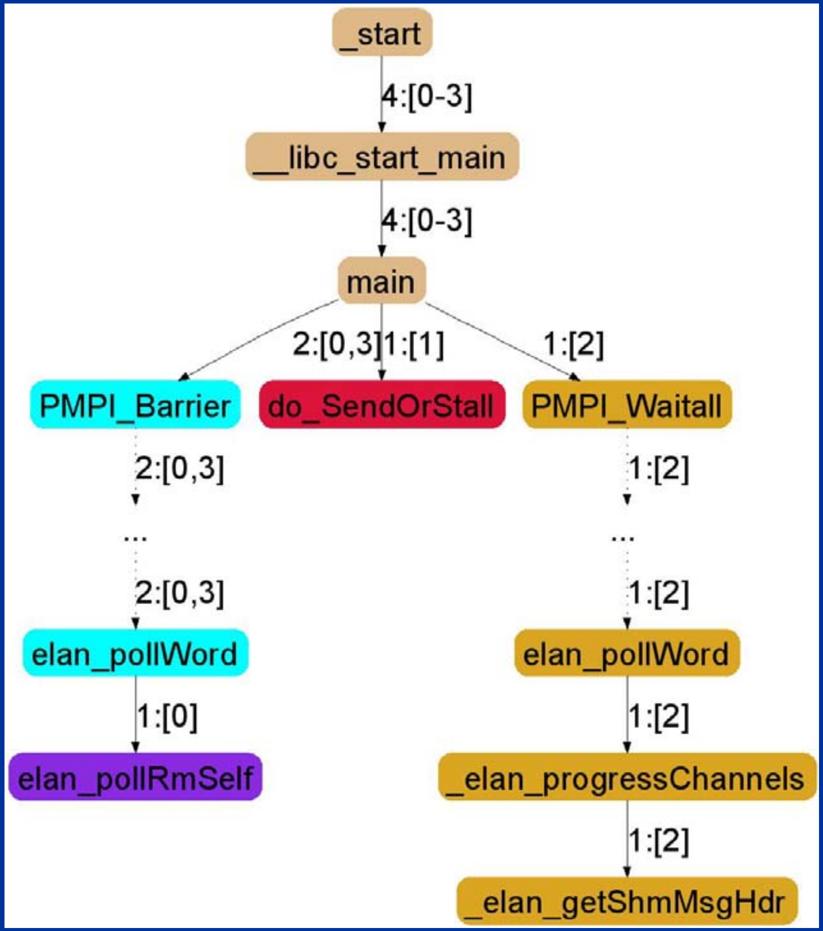
1:[2]

_elan_progressChannels

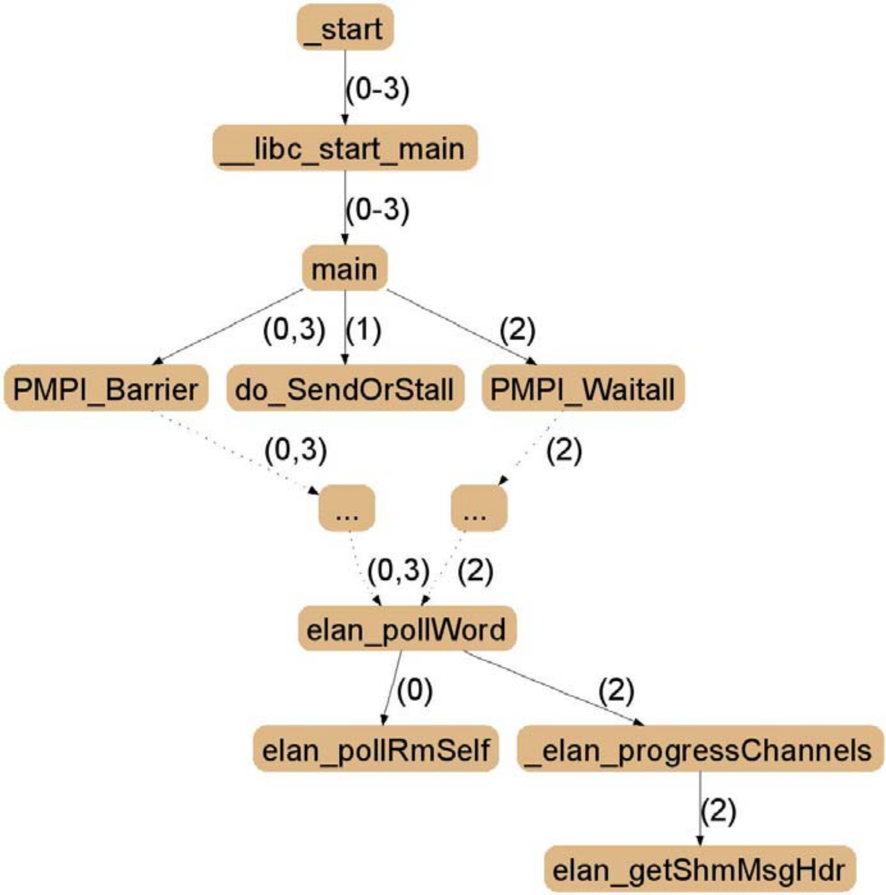
1:[2]

_elan_getShmMsgHdr

2D-Trace/Space Analysis



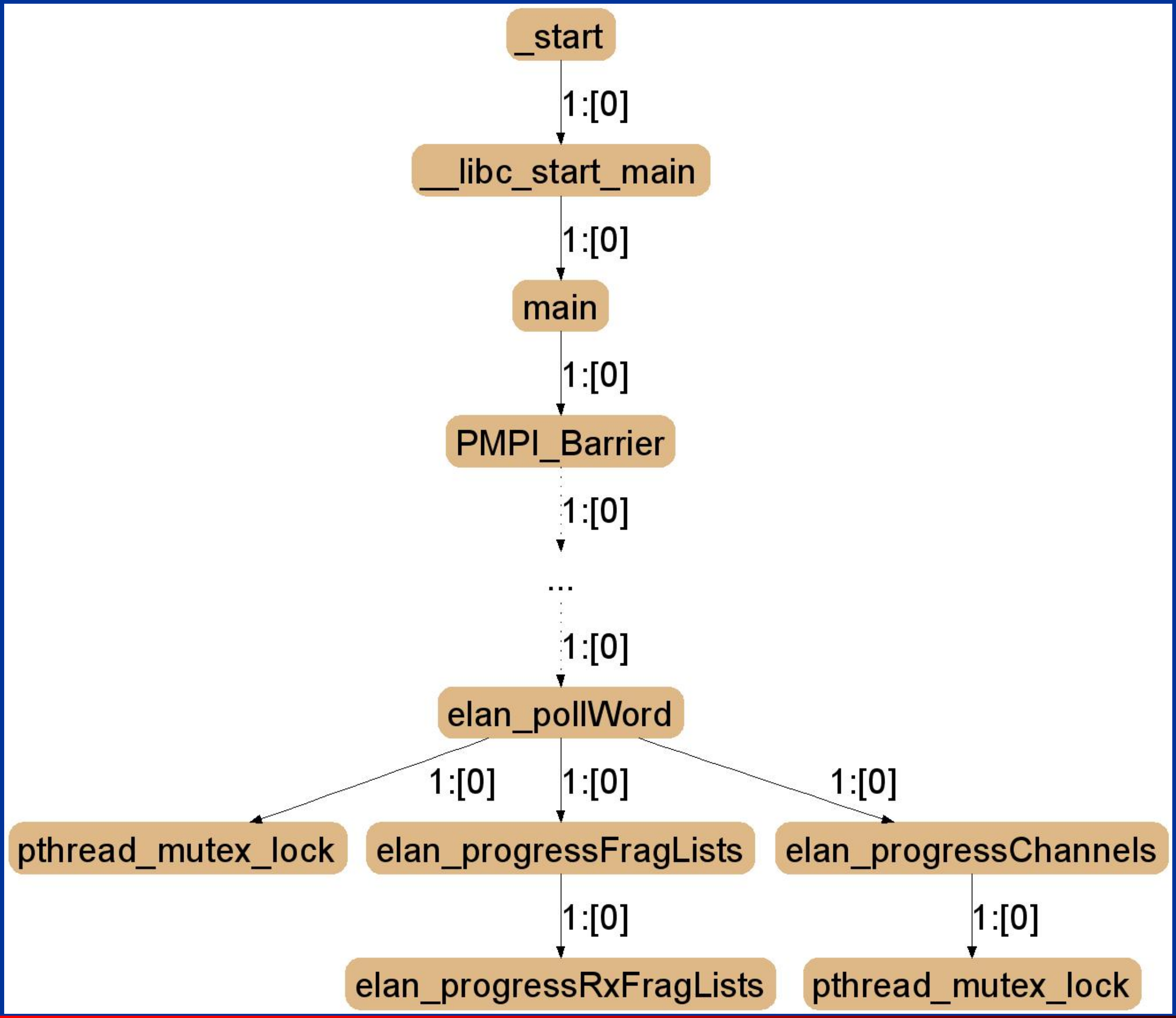
STAT



TotalView

2D-Trace/Time Analysis

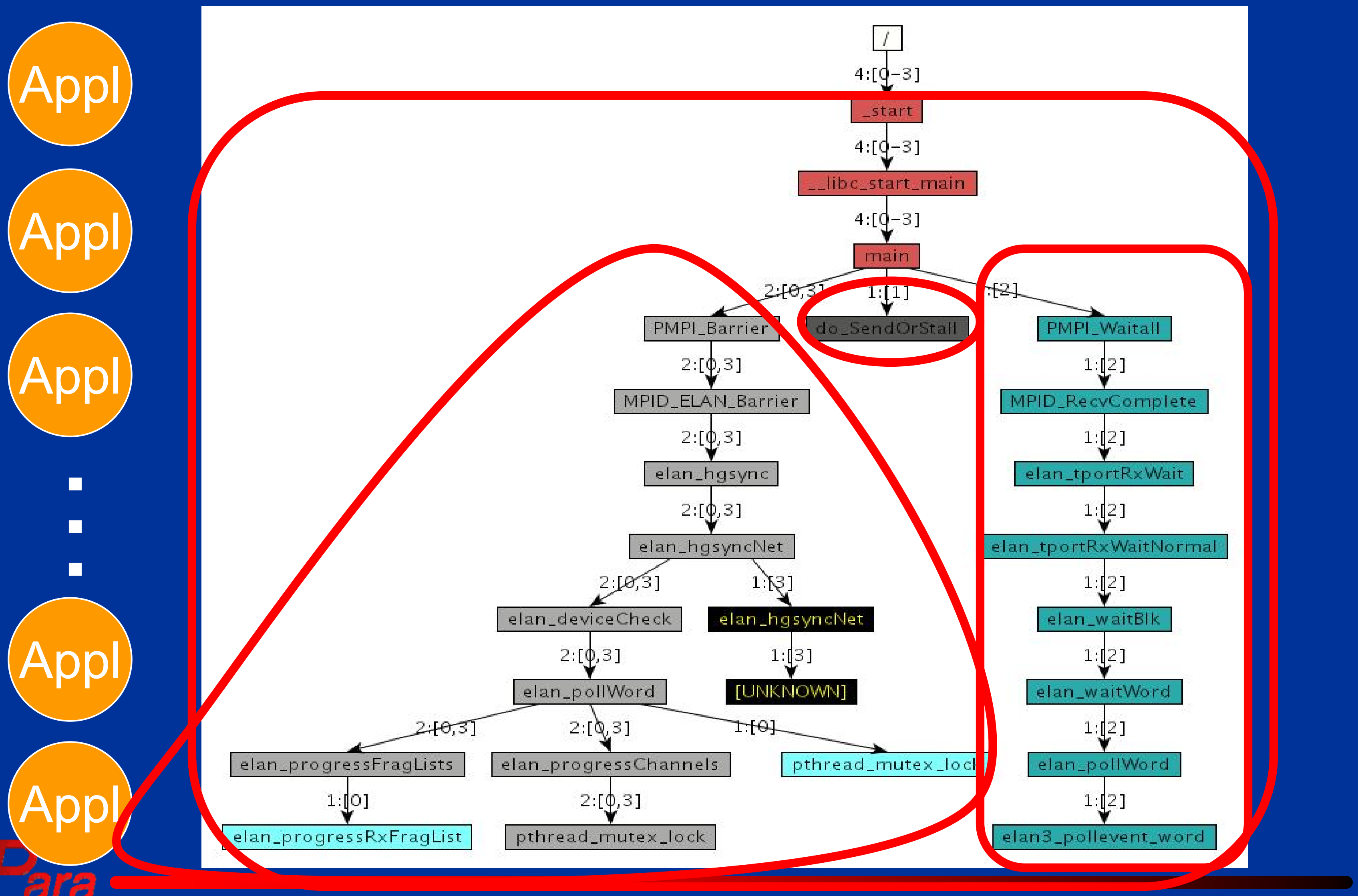
Appl



3D-Trace/Space/Time Analysis

- Multiple samples, multiple processes
  - Track global program behavior over time
  - Folds all processes together
  - Challenges:
    - Scalable data representations
    - Scalable analyses
    - Scalable and useful visualizations/results

3D-Trace/Space/Time Analysis



Motivating Case Study: Pthread Deadlock Exposed by CCSM

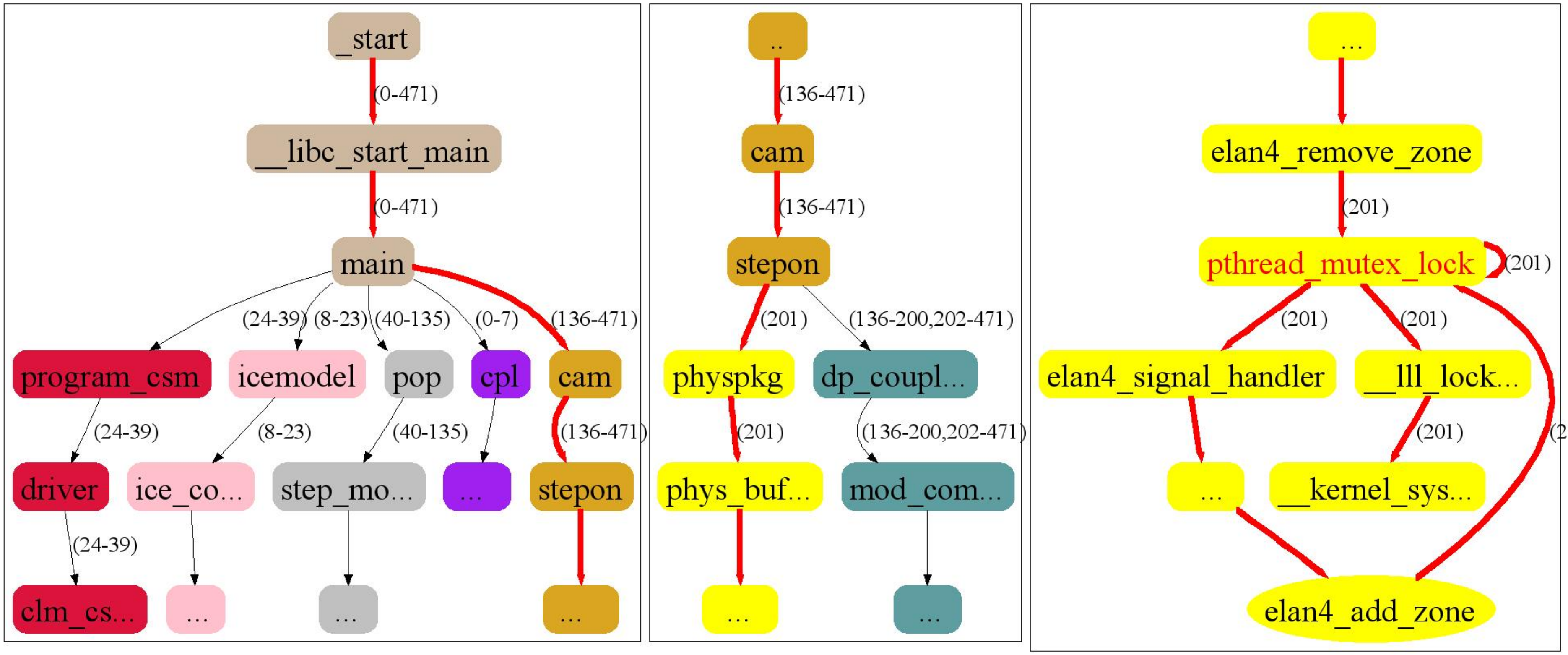
CCSM: Community Climate System Model

- Multiple Program Multiple Data (MPMD) model
- Comprises atmosphere, ocean, sea ice and land surface models
- Used to make climate predictions
- MPI-based application

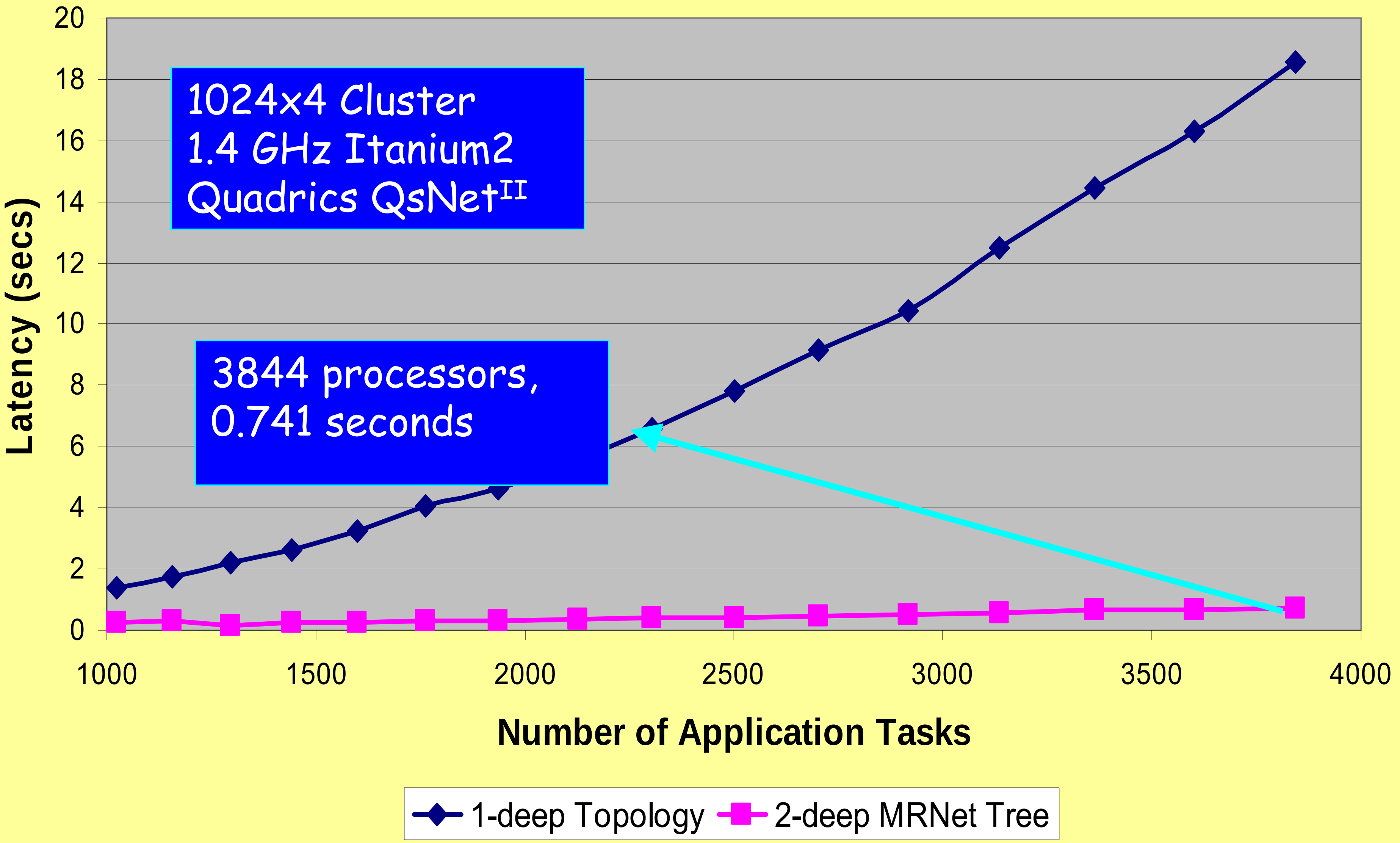
Motivating Case Study: Pthread Deadlock Exposed in the CCSM

- Intermittently hangs:
  - Non-deterministic
  - Only at large scale
  - Appears at seemingly random code locations
  - Hard to reproduce
    - 2 hangs over 10 days (~50 runs)
- Stack traces can provide useful insight
- Many bugs are temporal in nature
  - Error not because behavior occurs, but because behavior persists!
- Need tools that run effectively at scale

STAT on CCSM Case Study



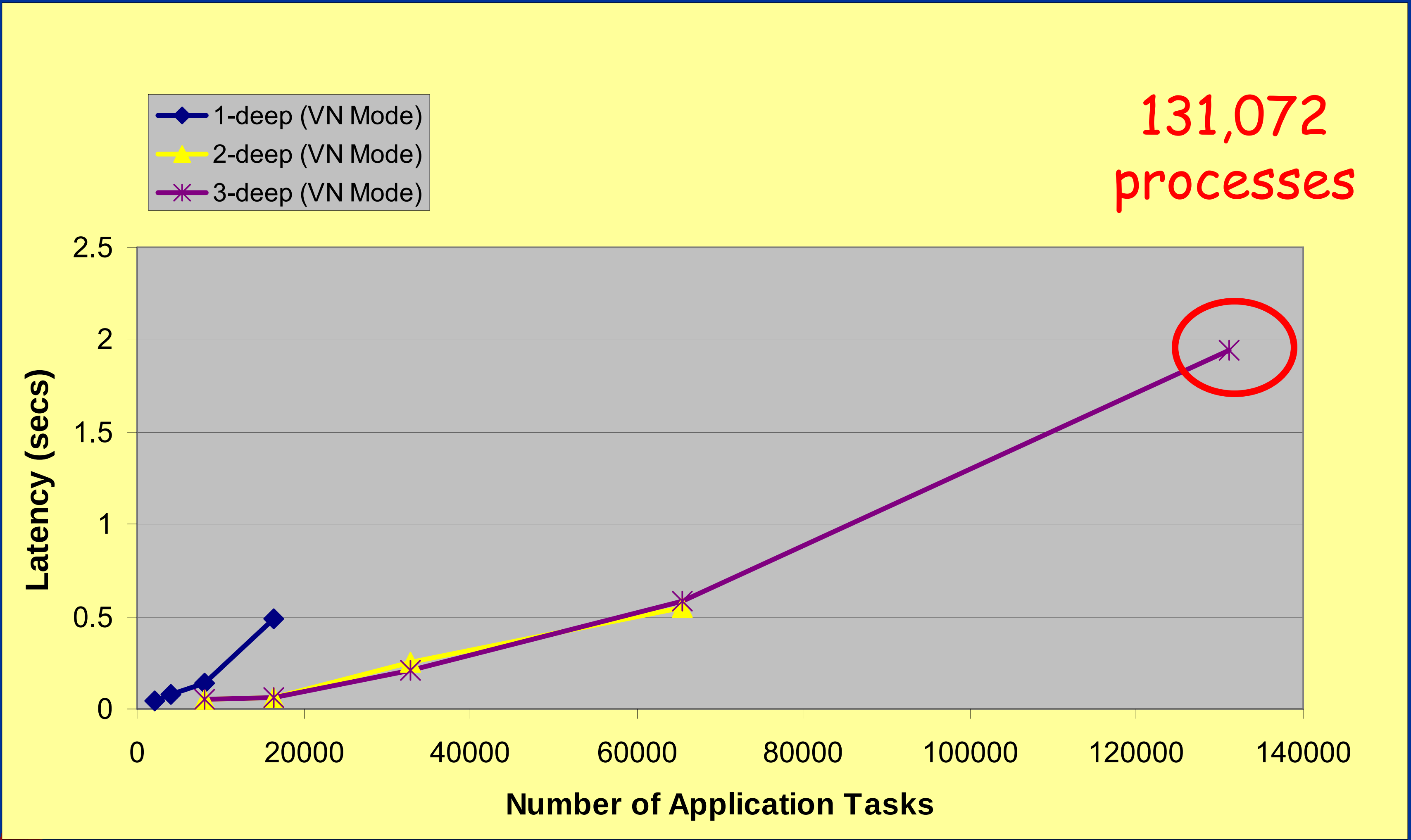
STAT Performance on an IA64 Cluster



1024x4 Cluster
1.4 GHz Itanium2
Quadrics QsNet^{II}

3844 processors,
0.741 seconds

STAT Performance on BlueGene/L



131,072 processes

A Platform for Research: Fault Tolerance

Two key observations:

- Leverage TBŌN properties
  - Inherent information redundancies
- Weak data consistency model: convergent recovery
  - Final output stream converges to non-failure case
  - Intermediate output packets may differ
  - Preserves all output information

Results in:

- No overhead during normal operation
- Rapid recovery
  - Limited process participation
- General recovery model
  - Applies to broad classes of computations

Current Reliability Approaches

- Fail-over (hot backup)
  - Replace failed primary w/ backup replica
  - Extremely high overhead: 100% minimum!

- Rollback recovery
  - Rollback to checkpoint after failure
  - May require dedicated resources and lead to overloaded network/storage resources

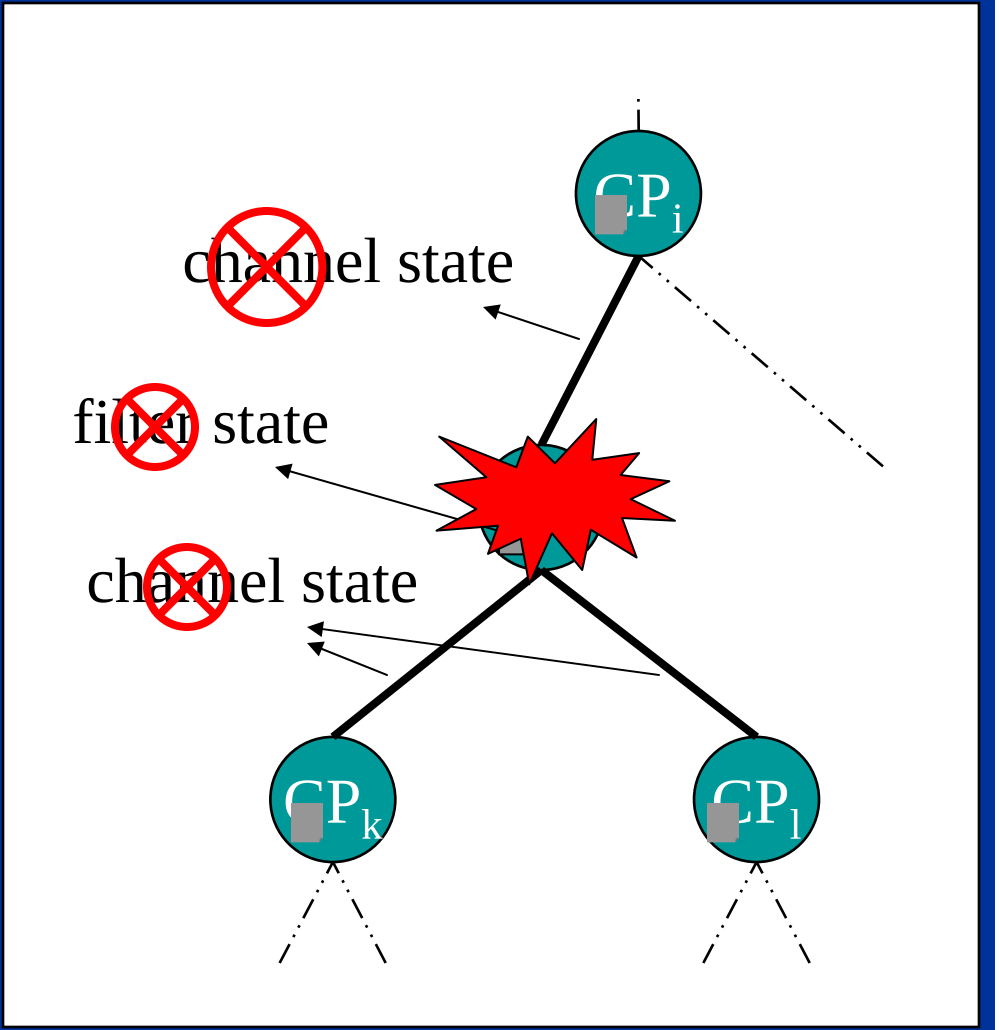
TBON Theory

TBŌN Output Theorem
Output depends only on channel states and root filter state

All-encompassing Leaf State Theorem
State at leaves subsume channel state (all state throughout TBŌN)

Result: only need leaf state to recover from root/internal failures

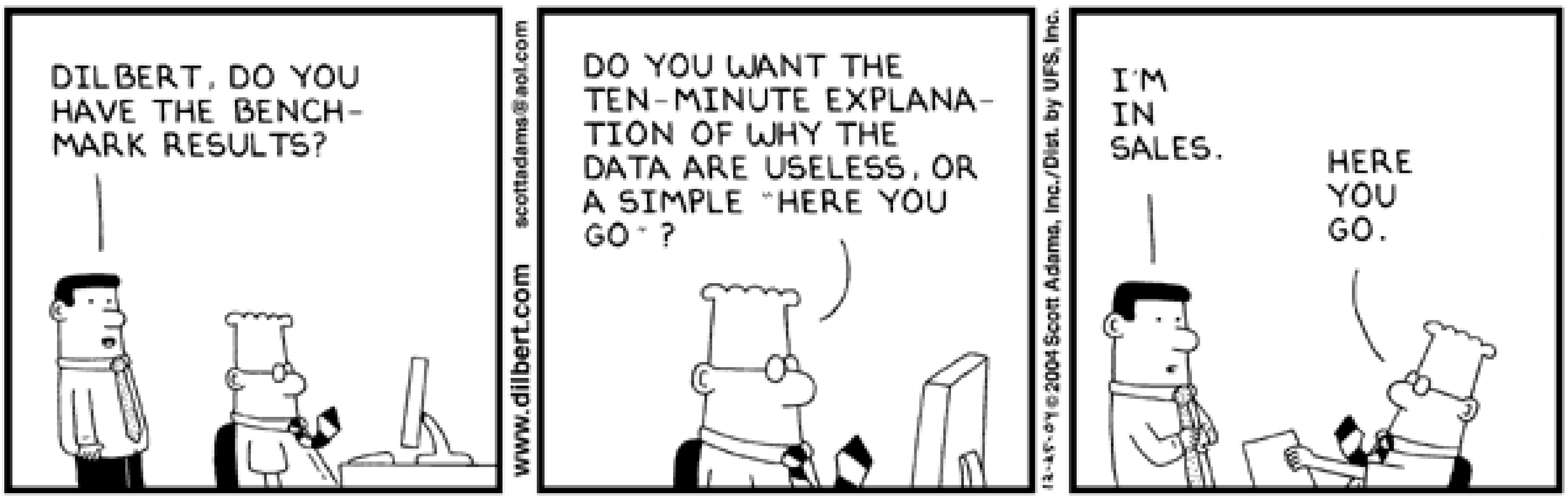
Filter requirements:
- Associative
- Communicative
- Idempotent



End Where We Started

TBŌNs provide:

- An immediate path to scalable tools and infrastructure. Examples:
  - Paradyn Performance Tools
  - Vision algorithms
  - Stack trace analysis (new)
- A Research platform for new technologies:
  - New concepts in fault tolerance (no logs, no hot-backups).
  - As an framework for parallel applications
  - As a powerful alternative to the Map-Reduce idiom
  - As a generalized, scalable communication infrastructure


DILBERT, DO YOU HAVE THE BENCHMARK RESULTS?
scottadams@aol.com
www.dilbert.com
DO YOU WANT THE TEN-MINUTE EXPLANATION OF WHY THE DATA ARE USELESS, OR A SIMPLE "HERE YOU GO"?
© 2004 Scott Adams, Inc./Dist. by UFS, Inc.
I'M IN SALES.
HERE YOU GO.

© UFS, Inc.

MRNet References

- Arnold and Miller, "State Compensation: A Scalable Failure Recovery Model for Tree-based Overlay Networks", UW-CS Technical Report, February 2007
- Arnold, Pack and Miller: "Tree-based Overlay Networks for Scalable Applications", *Workshop on High-Level Parallel Programming Models and Supportive Environments*, April 2006.
- Roth and Miller, "The Distributed Performance Consultant and the Sub-Graph Folding Algorithm: On-line Automated Performance Diagnosis on Thousands of Processes", *PPoPP,* March 2006.
- Schulz et al, "Scalable Dynamic Binary Instrumentation for Blue Gene/L", *Workshop on Binary Instrumentation and Applications,* September, 2005.
- Roth, Arnold and Miller, "Benchmarking the MRNet Distributed Tool Infrastructure: Lessons Learned", *2004 High-Performance Grid Computing Workshop*, April 2004.
- Roth, Arnold and Miller, "MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools", *SC 2003,* November 2003.

www.cs.wisc.edu/paradyn

Extra Slides

TBŌNs in the Wild

Universitat Politèchnica de Catalunya (Jesus Labarta):

- Use MRNet to adaptively select trace granularity.
- Cluster analysis of traces to select representatives.

University of Oregon (Al Malony):

- TauOverMRNet -- Collect and analyze TAU trace data using MRNet framework.
- Filters include random sampling, statistical analysis (mean, var., std. dev., etc.). Filter to throttle data rates based on feedback from nodes and trace data merging filter.

TBŌNs in the Wild

Krell Insititute (formerly SGI):

- Open|Speedshop: An open source performance tool suite.
- Used to use IBM's DPCL for distributed monitoring and control, but switching MRNet for scalability.

RENCI (Dan Reed, Todd Gamblin, Frank Mueller):

- MPI tracing facility that includes local process-level performance statistics.
- Use MRNet to control the granularity of the collection of performance data.

TBŌNs in the Wild

Paradyn Project (Mike Brim):

TBON-FS: Scalable file I/O for process control and monitoring.

Introduces the notion of a *group file* to operate on many instances of /proc.

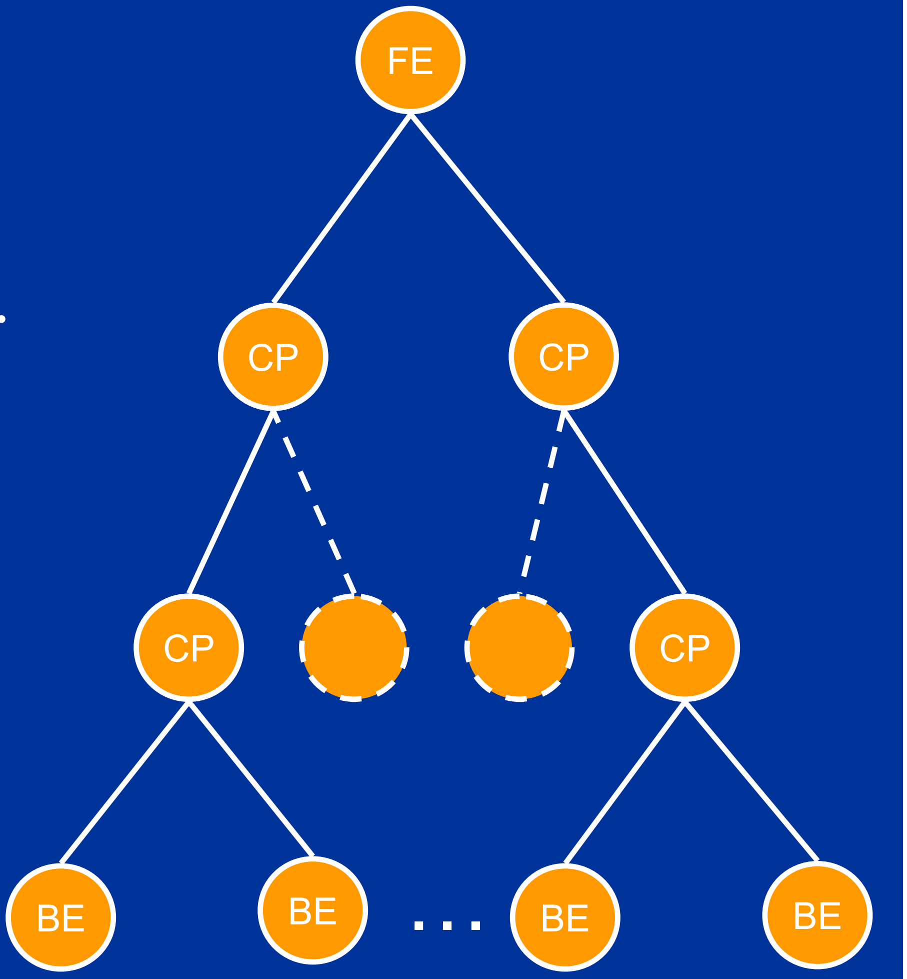
Initial projects:

- Highly scalable Ganglia implementation (also simplifies the architecture.
- Group file shell.
- Highly scalable debugger – in collaboration with Totalview Tech.

TBŌN Model

Efficiency:

- Zero-copy paths
- Scatter-gather
- Binary data representation.



STAT Filter

```
sta_filter(vector<Packet> pkts_in,
           vector<Packet> pkts_out)
{
  for( i=0; i<pkts_in.size; i++){
    trace = deserialize( pkts_in[i] );
    ret_trace = merge( ret_trace,
                       trace );
  }

  Packet p = serialize( ret_trace );

  pkts_out.pushback(p);
}
```

MRNet Front-end Interface

```
front_end_main(){
  Network * net = new Network (topology);

  Communicator * comm = net->
    get_BroadcastCommunicator();

  Stream * stream =
    new Stream( comm, IMAX_FILT, WAITFORALL);

  stream->send(“%s”, “go”);

  stream->recv(“%d”, &result);
}
```

MRNet Back-end Interface

```
back_end_main(){
  Stream * stream;
  char *s;

  Network * net = new Network();

  net->recv(“%s”, &s, &stream);

  if(s == “go”){
    stream->send(“%d”, rand_int);
  }
}
```

MRNet Filter Interface

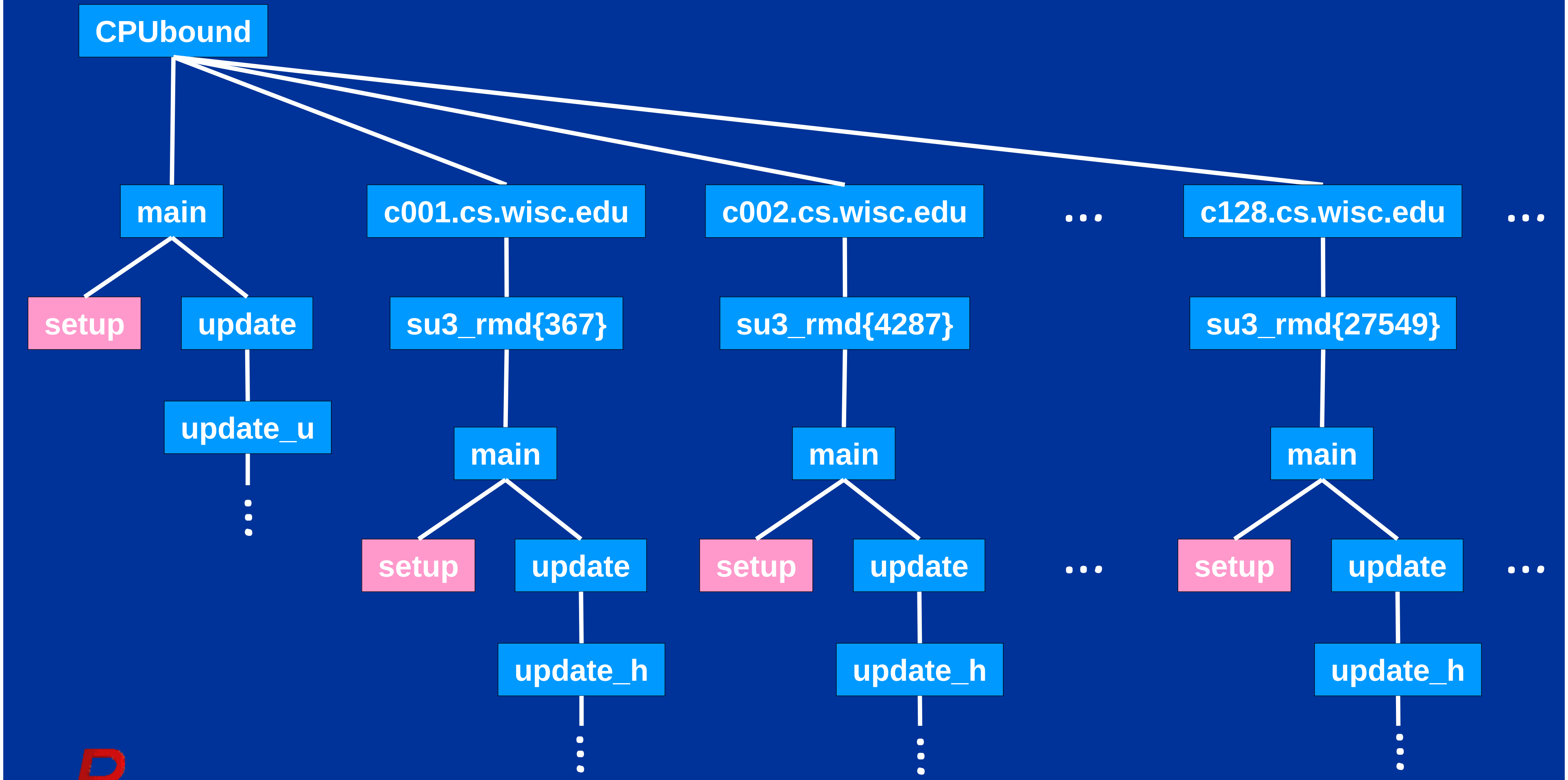
```
imax_filter(vector<Packet> packets_in,
            vector<Packet> packets_out)
{
  for( i=0; i<packets_in.size; i++){
    result = max( result,
                  packets_in[i].get_int() );
  }

  Packet p(“%d”, result);

  packets_out.pushback(p);
}
```

Evaluation: Results Overview

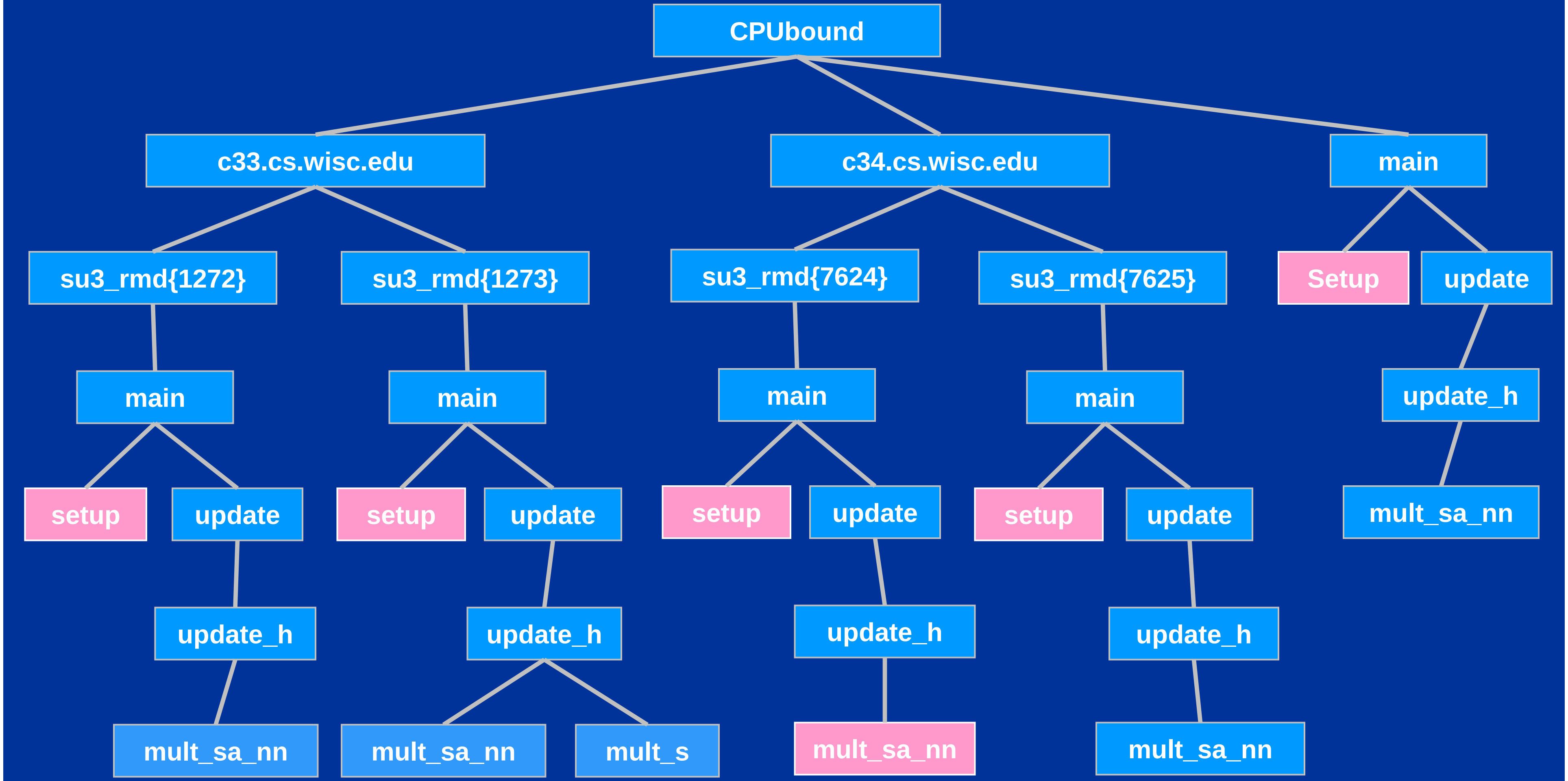
- PDA and TDA: bottleneck searches with up to 1024 processes (limited by LLNL batch allocation size policy, not by our software or approach)
- CA: scalability limit at less than 64 processes
- Crucial: similar qualitative results using each approach

Performance Consultant: Example

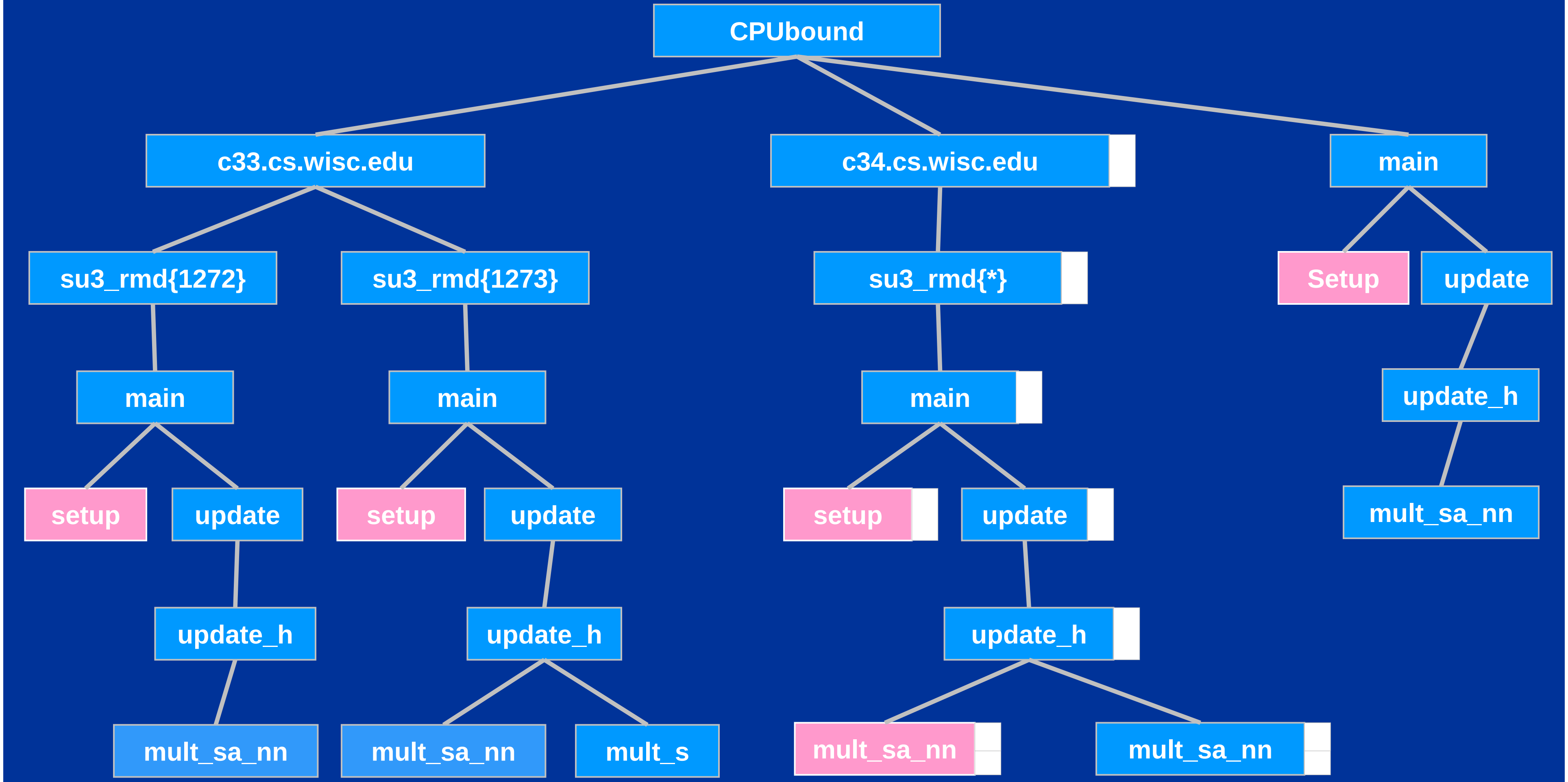
- Part of a *search history graph*



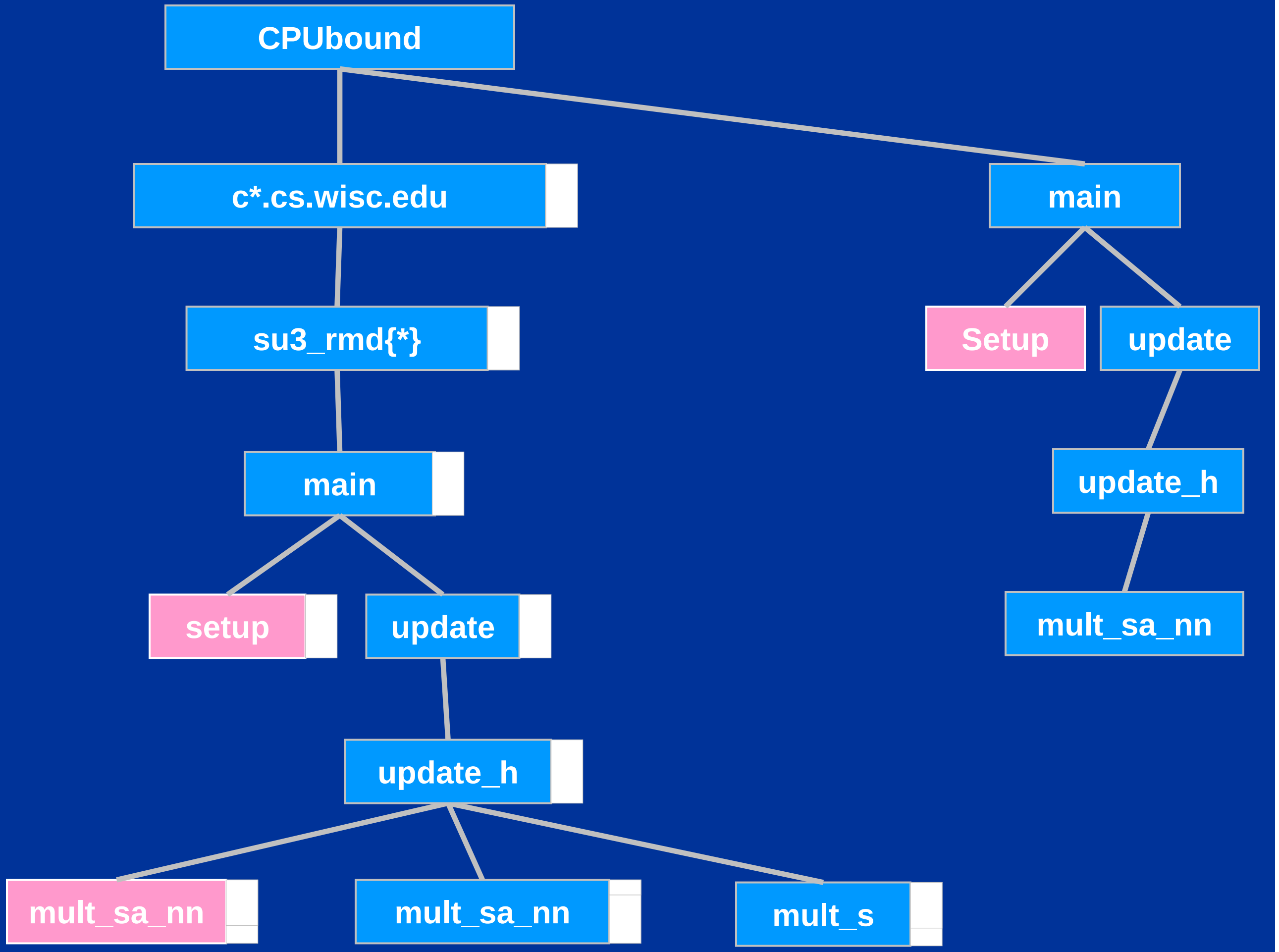
SGFA: Example



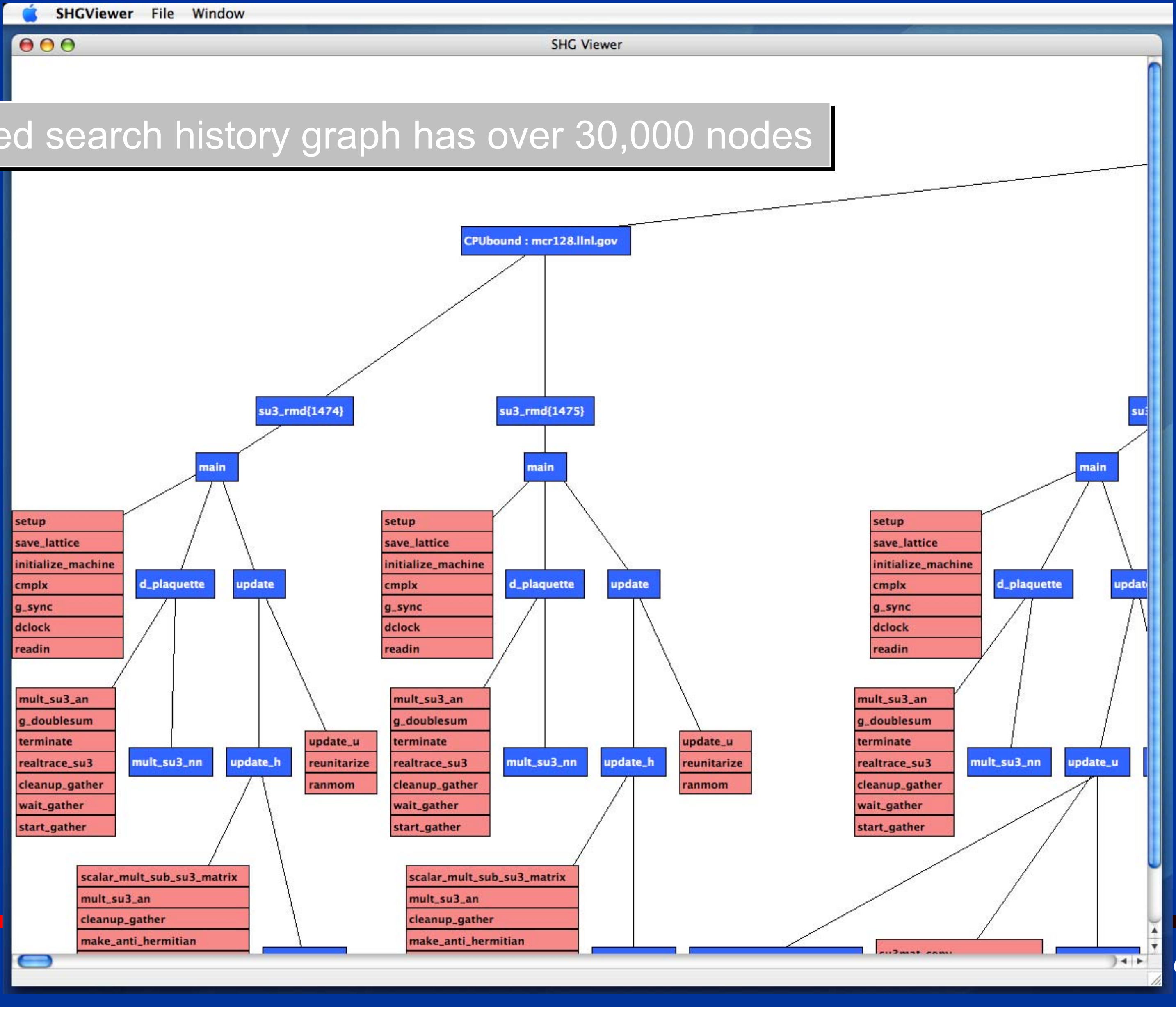
SGFA: Example



SGFA: Example



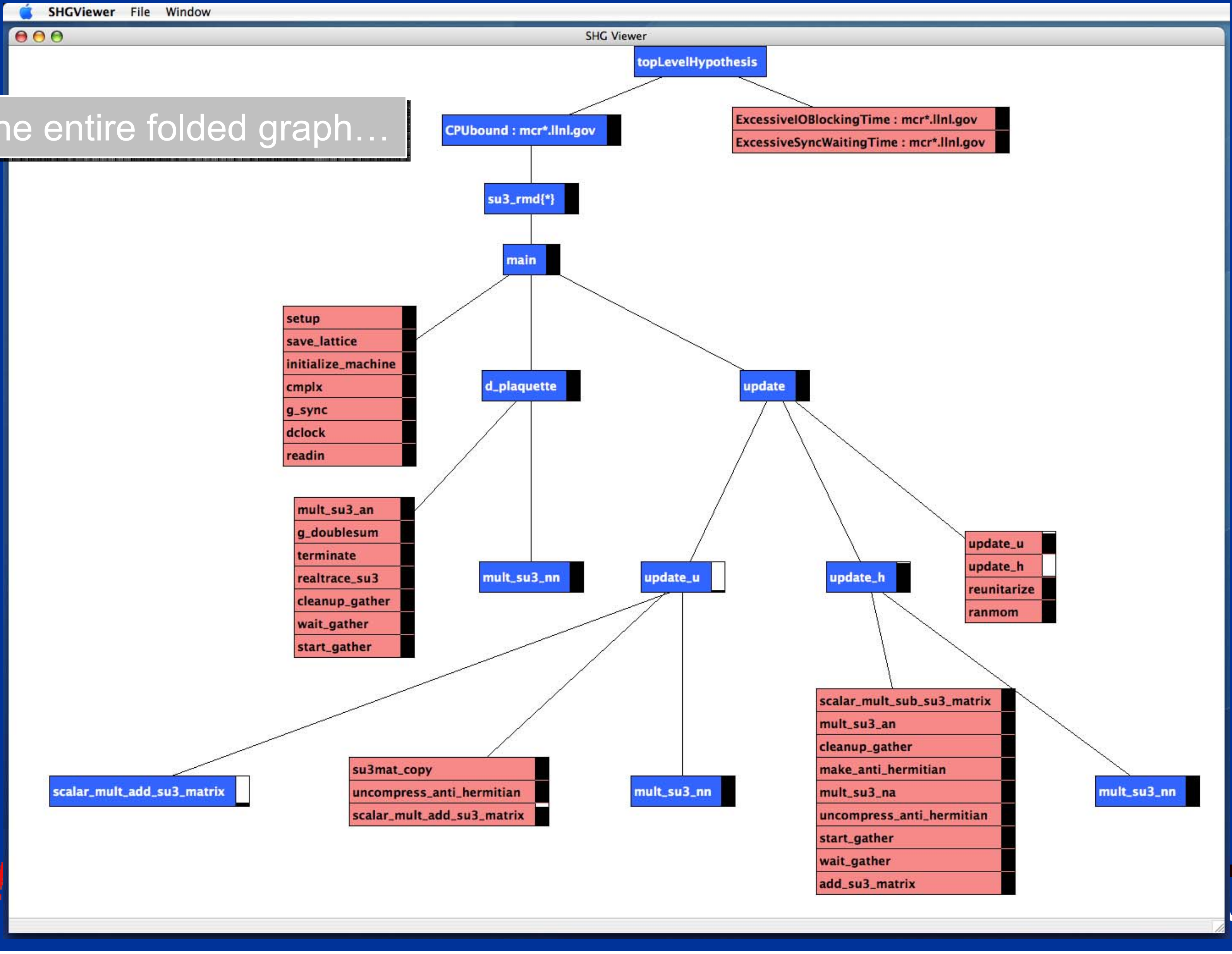
Evaluation: SGFA



Un-folded search history graph has over 30,000 nodes

Evaluation: SGFA

The entire folded graph…



TBŌNs for Scalable Applications

- Many algorithms ⇒ equivalence computation
  - (Non-)equivalence to summarize/analyze input

Application	Input	Filter	Output
Trace Analysis	Trace file	Trace equivalence / Anomaly detector	Compressed traces, anomalous traces
Graph Merging	Sub-graphs	Sub-graph equivalence	Merged graphs
Data Clustering	Data Files	Object classifiers	Partitioned data

STAT Motivation

Stack
Trace
Analysis
Tool

- Discover application behavior
  - Progressing or deadlock?
  - Infinite loop?
  - Starvation?
  - Load balanced?
- Tool goals:
  - Pin-point symptoms as much as possible
  - Direct user's to root cause

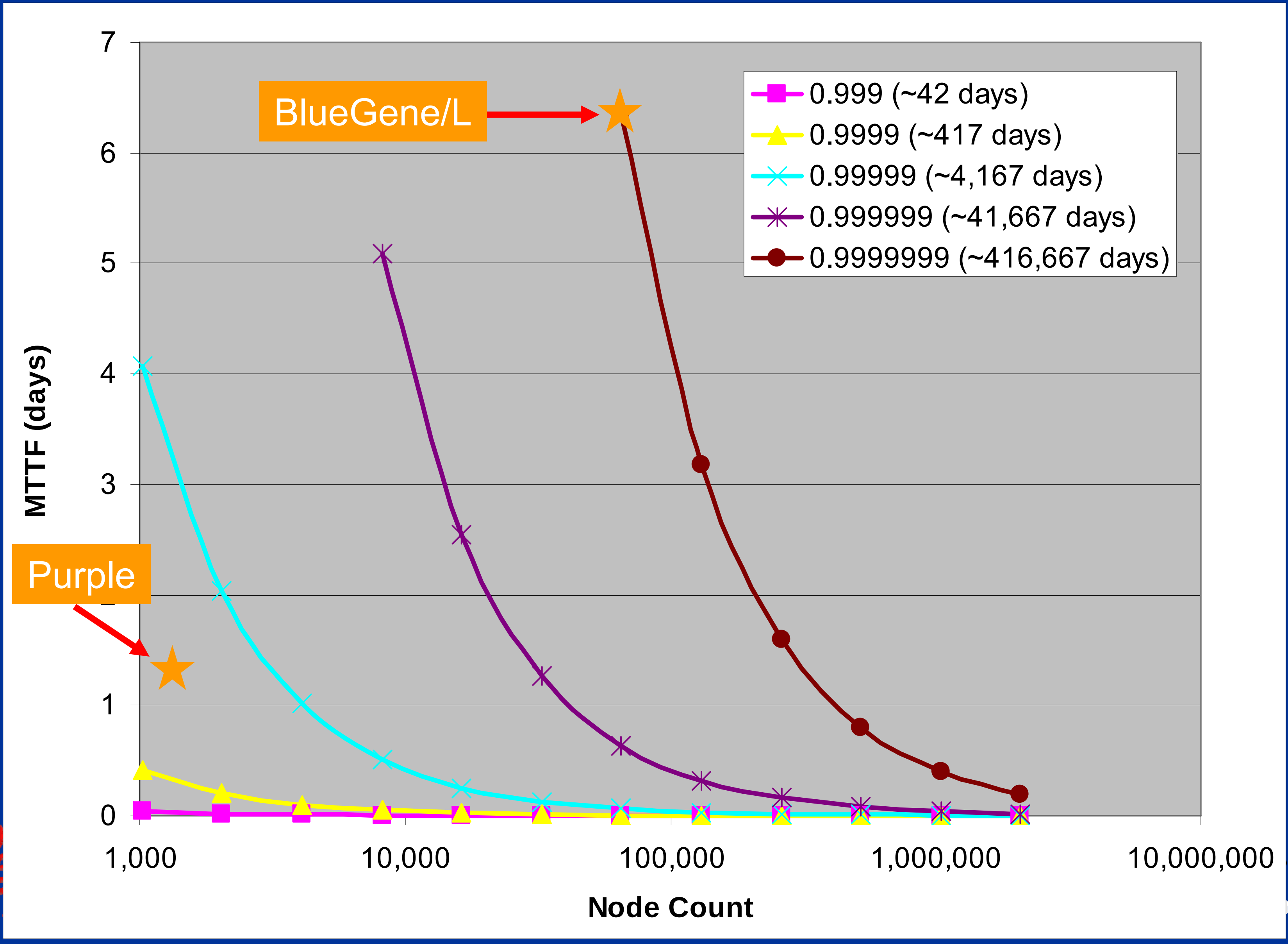
BG/L Scaling Test Setup

- Run on a single rack BG/L system
  - 1024 compute nodes allows emulation of up to 1024 I/O node daemons (full BG/L)
- Emulate both coprocessor mode and virtual node mode of full BG/L (64 and 128 tasks per daemon, respectively)
- Ran 2-deep and 3-deep topologies

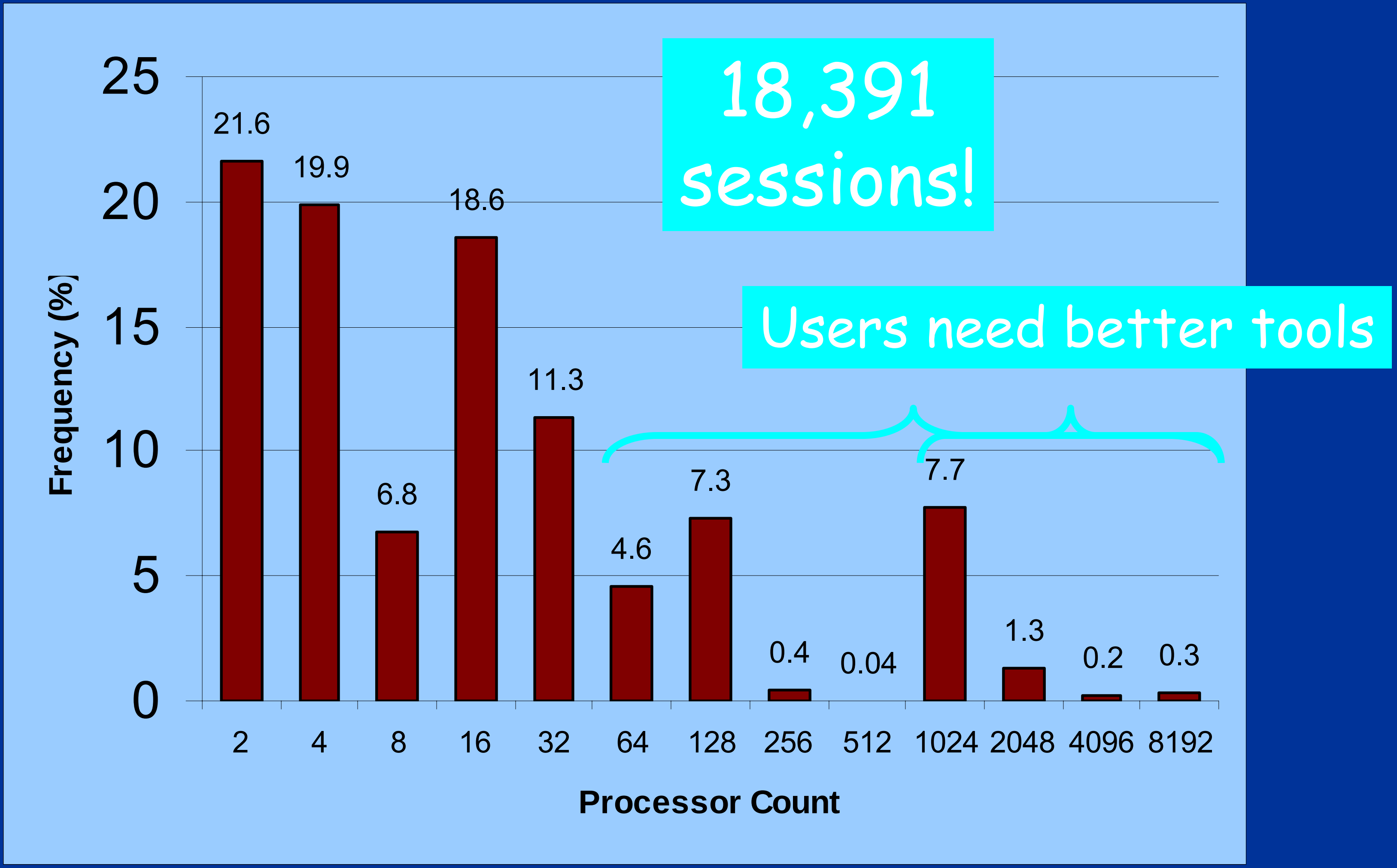
STATBench Revealed a STAT Scalability Issue

- Edge labels represented by task lists
  - Original implementation as strings
  - [1,3,4,5,6,9,10,11,15] -> "[1,3-6,9-11,15]"
  - Up to 75KB at 32,768 tasks
- Re-implemented edge label as a bit vector
  - 1 bit per task
    - Set to 1 if the task is in the list
    - Set to 0 otherwise

Large Scale System Reliability



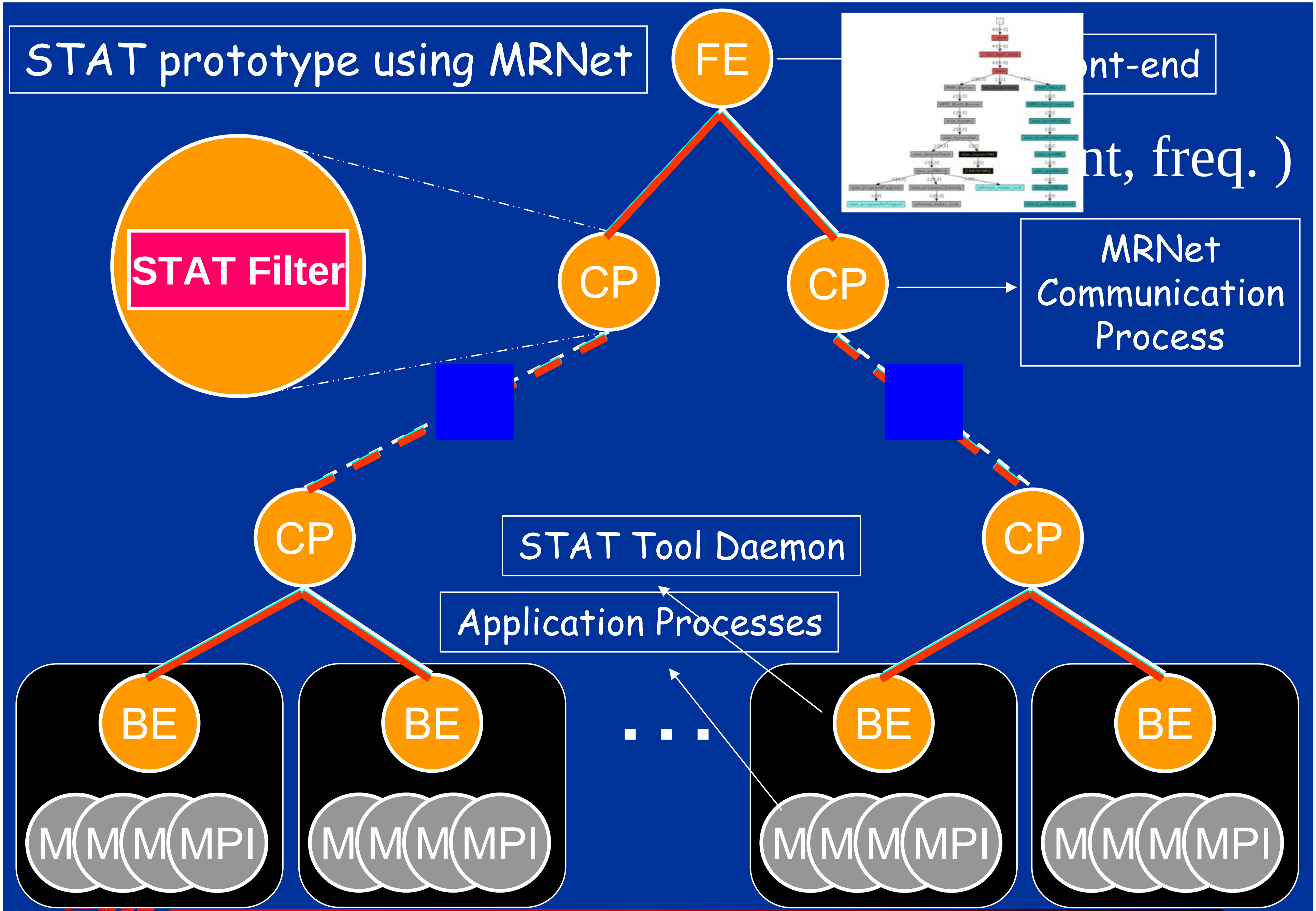
LLNL Parallel Debug Sessions (03/01/2006 - 05/11/2006)



18,391 sessions!

Users need better tools

STAT prototype using MRNet



MRNet Communication Process

STAT Tool Daemon

Application Processes

STAT BG/L Experimental Setup

- STAT Front-end and communication processes on login nodes
- STAT Back-end on I/O nodes
- MPI task on compute nodes

BG/L Configuration

- Each node has 2 CPUs
- 14 login nodes
- 1,664 I/O nodes
  - Each I/O node connects to 64 compute nodes
- 106,496 compute nodes
  - Co-processor (CO) mode: 1 CPU for application process, 1 CPU for communication
  - Virtual node (VN) mode: 2 CPUs for application process

Future of STAT

- Future research detailed in paper
- Plans to make generally available
  - http://www.paradyn.org/STAT
- TBŌN computing papers & open-source prototype, MRNet, available at:
  - http://www.paradyn.org/mrnet

(More) HPC Trends from TOP500®

SUPERCOMPUTER SITES

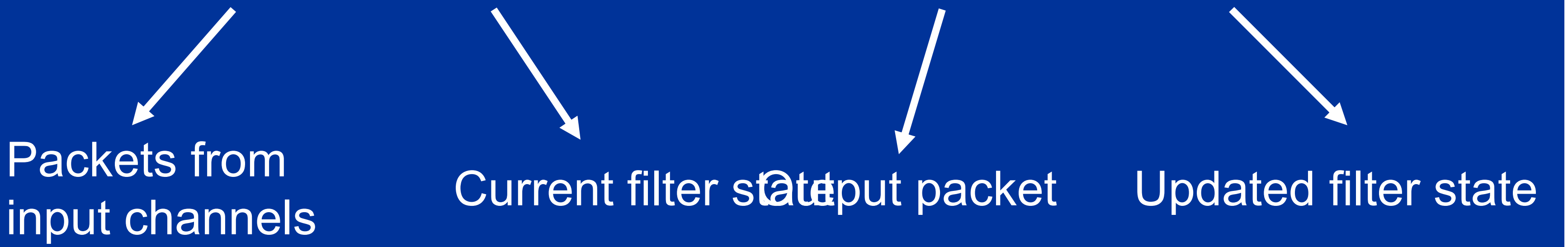
- 60% are larger than 10^3 processors
- 10 systems larger than 10^4 processors

System	Location	Size	Time Frame
RoadRunner	LANL	~3.2×10^4	2008
Jaguar	ORNL	~4.2×10^4	2007
SunFire x64	TACC	~5.2×10^4	2007
Cray XT4	ORNL	~2×10^5	2008
BlueGene/P	ANL	~5×10^5	2008
BlueGene/Q	ANL/LLNL	~10^6	2010-2012

Background: Data Aggregation

Filter function:

$$f\,(in_n(CP_i);f\,s_n(CP_i))\;!\quad f\,out_n(CP_i);f\,s_{n+1}(CP_i)g$$



Packets from input channels

Current filter state

Output packet

Updated filter state

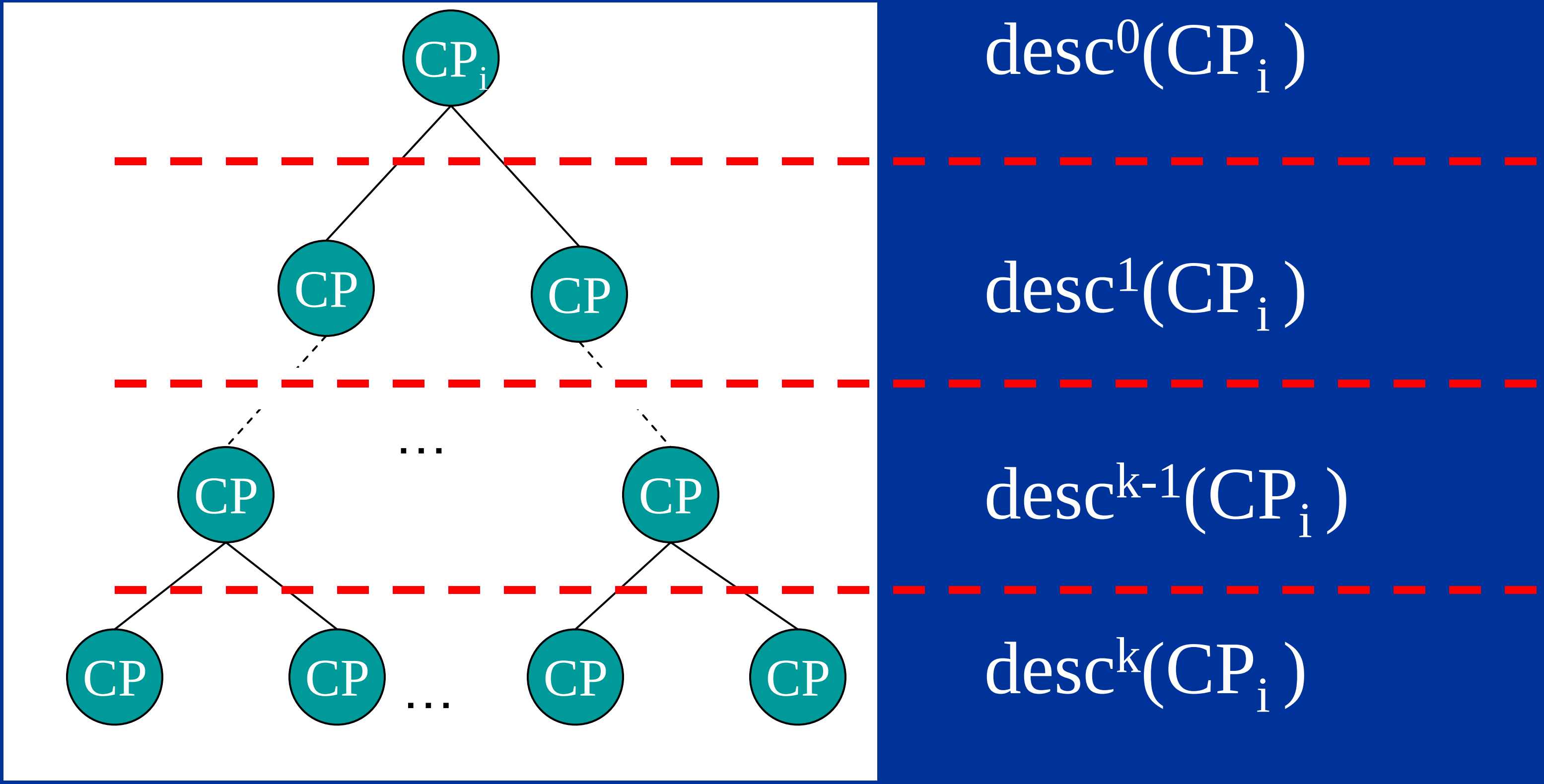
Background: Filter Function

- Built on state join and difference operators
- State join operator, **t**
  - Update current state by merging inputs

$$in_n(CP_i)\,t\;\,f\,s_n(CP_i)\,!\quad f\,s_{n+1}(CP_i)$$

  - Commutative: $a\,t\;b = b\,t\;a$
  - Associative: $(a\,t\;b)\,t\;c = a\,t\;(b\,t\;c)$
  - Idempotent: $a\,t\;a = a$

Background: Descendant Notation



$desc^0(CP_i)$

$desc^1(CP_i)$

$desc^{k-1}(CP_i)$

$desc^k(CP_i)$

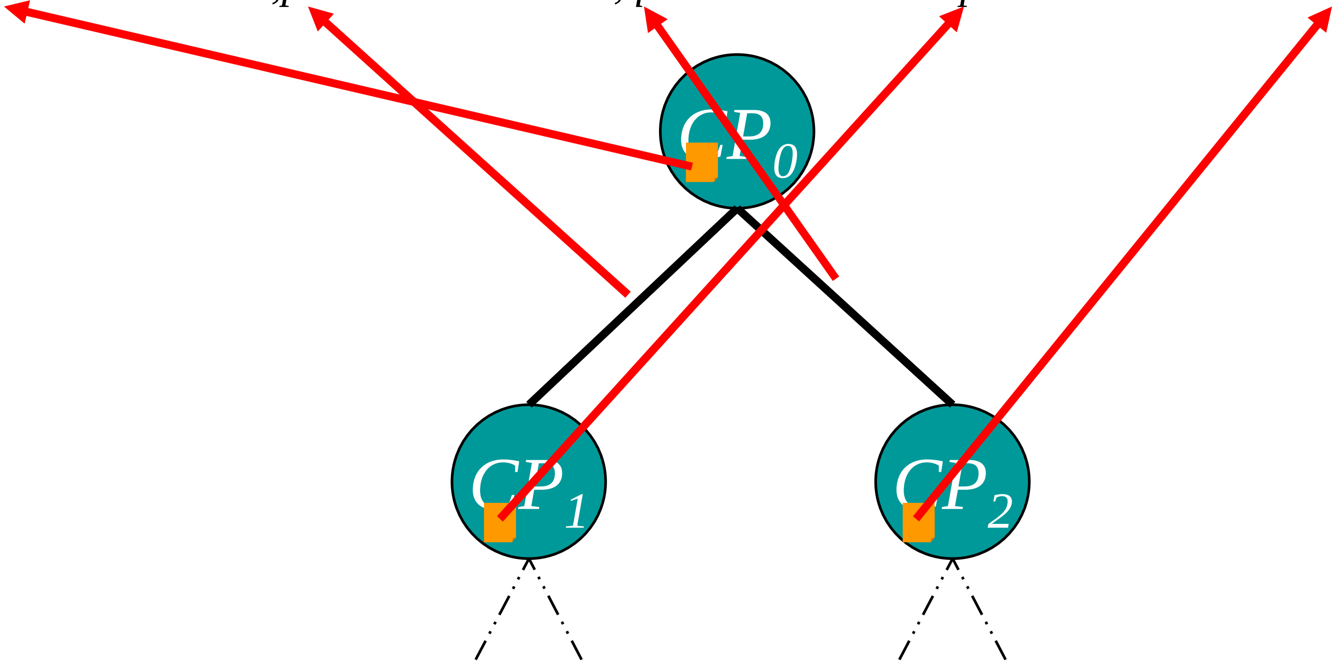
$fs(\ desc^k(CP_i)\ )$: join of filter states of specified processes

$cs(\ desc^k(CP_i)\ )$: join of channel states of specified processes

TBŌN Properties: Inherent Redundancy Theorem

The join of a CP’s filter state with its pending channel state equals the join of the CP’s children’s filter states.

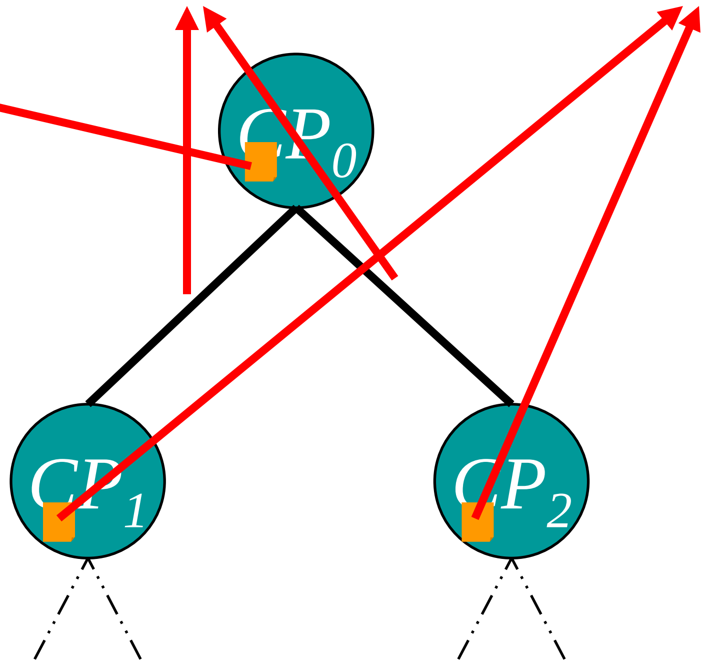
$$fs_n(CP_0) \sqcup cs_{n,p}(CP_0) \sqcup cs_{n,q}(CP_0) = fs_p(CP_1) \sqcup fs_q(CP_2)$$



TBŌN Properties: Inherent Redundancy Theorem

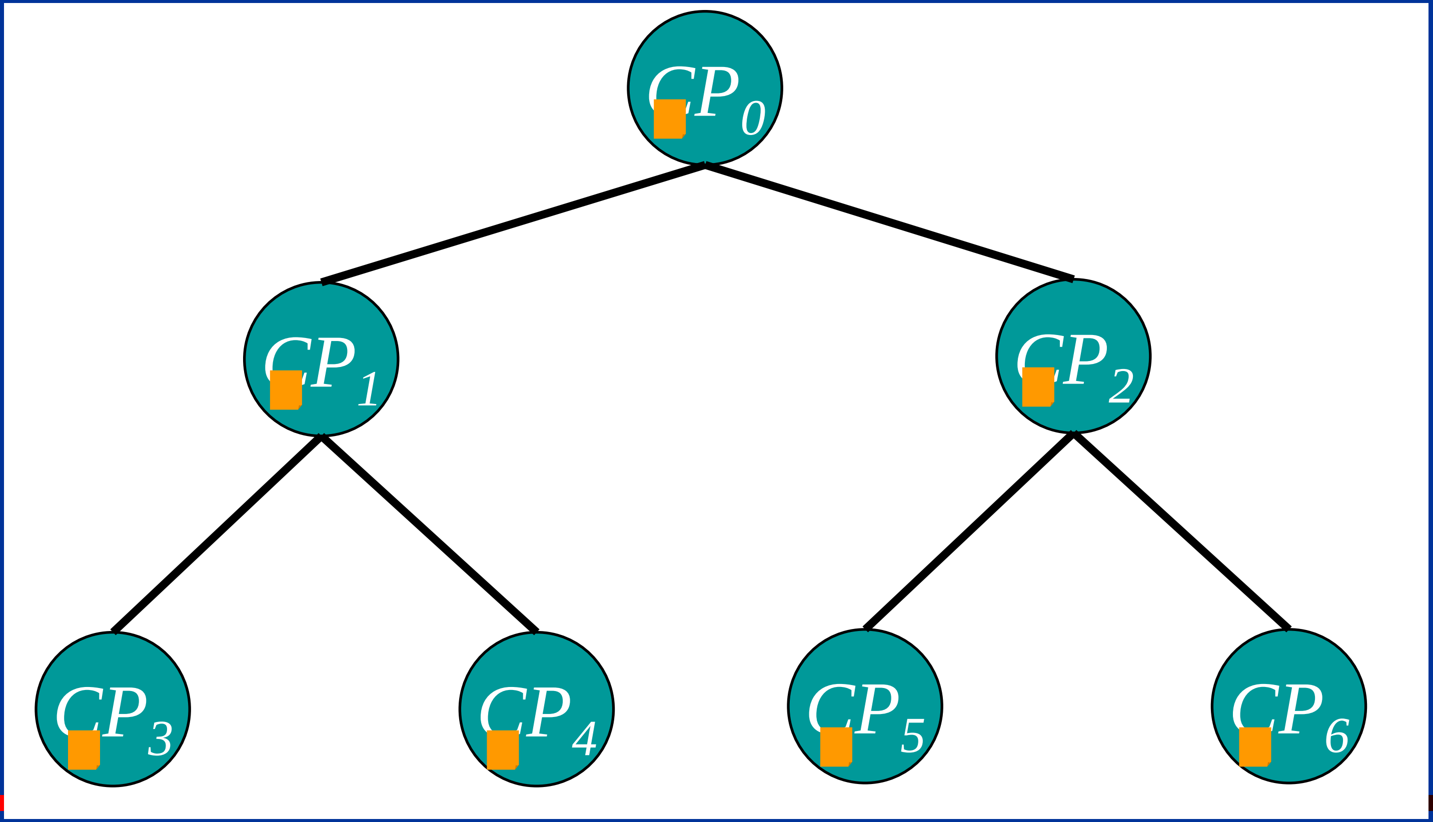
The join of a CP’s filter state with its pending channel state equals the join of the CP’s children’s filter states.

$$fs(\,desc^0(CP_0)) \quad \text{t} \quad cs(\,desc^0(CP_0)) \;=\; fs(\,desc^1(CP_0))$$



TBŌN Properties:
All-encompassing Leaf State Theorem

The join of the states from a sub-tree's leaves equals the join of the states at the sub-tree's root and all in-flight data



TBŌN Properties:
All-encompassing Leaf State Theorem

The join of the states from a sub-tree’s leaves equals
the join of the states at the sub-tree’s root and all in-flight data

From Inherent Redundancy Theorem:

$$f\ s(desc^1(CP_0)) = f\ s(desc^0(CP_0))\ t\ \ cs(desc^0(CP_0))$$

$$f\ s(desc^2(CP_0)) = f\ s(desc^1(CP_0))\ t\ \ cs(desc^1(CP_0))$$

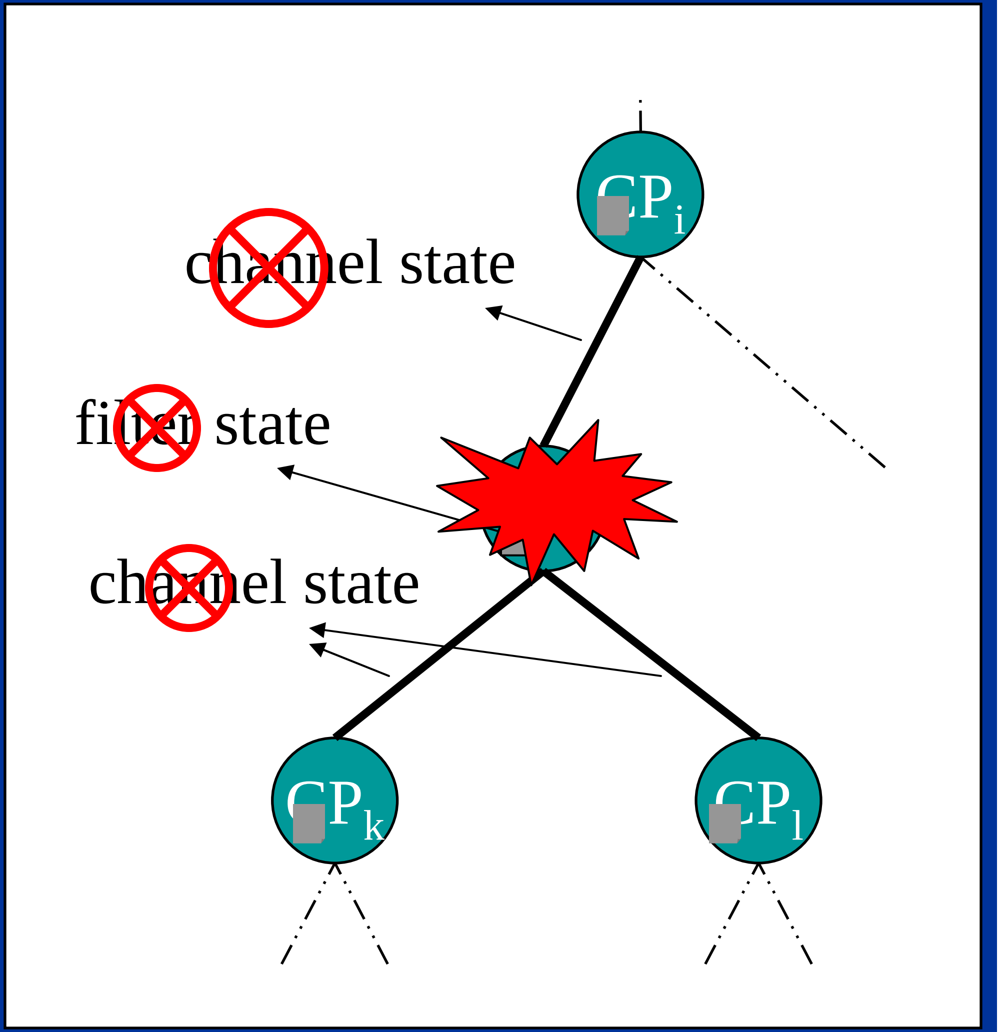
…

$$f\ s(desc^k(CP_0)) = f\ s(desc^{k\,i\ \ 1}(CP_0))\ t\ \ cs(desc^{k\,i\ \ 1}(CP_0))$$

$$f\ s(desc^k(CP_0)) = f\ s(CP_0)\ t\ \ cs(desc^0(CP_0))\ t\ \ :::\ t\ \ cs(desc^{k\,i\ \ 1}(CP_0))$$

TBON Theory

- TBŌN end-to-end argument: output only depends on state at the end-points
- Can recover from lost of any internal filter and channel states



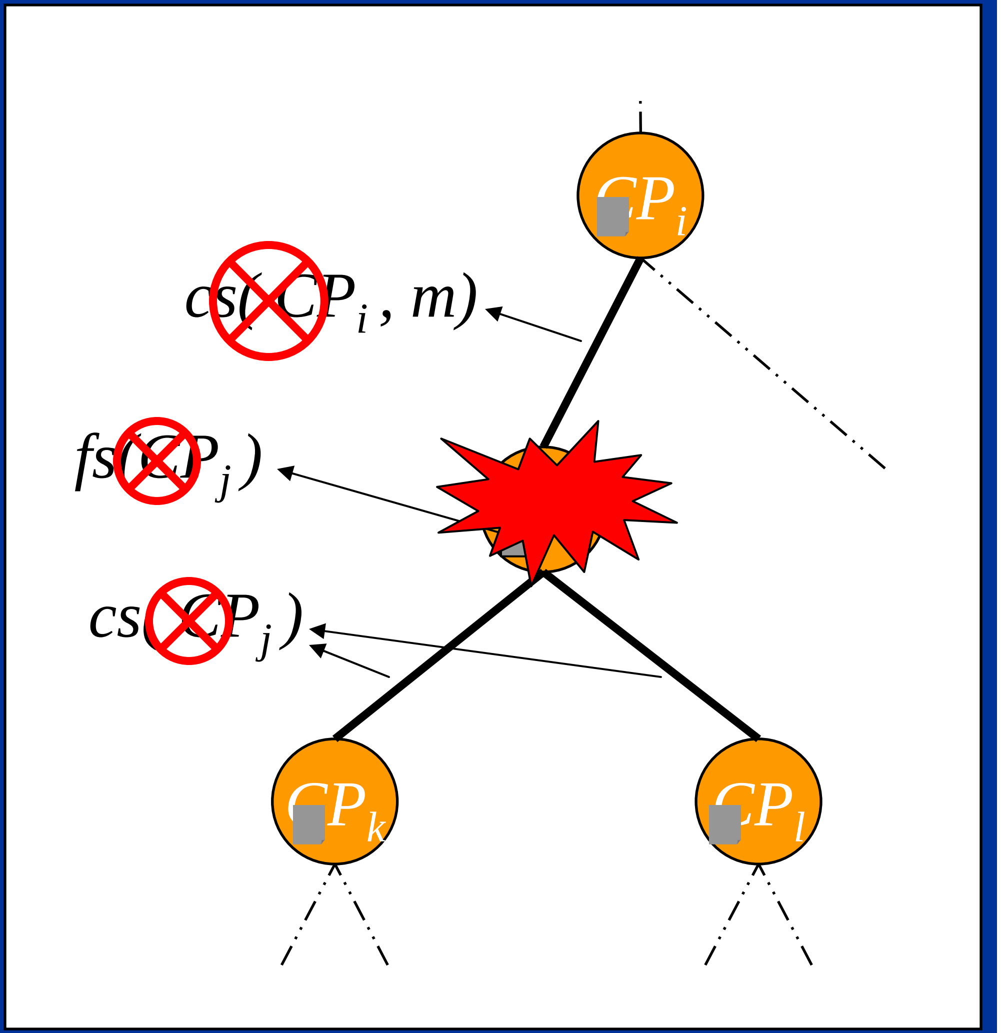
State Composition

If CP$_j$ fails, all state associated with CP$_j$ is lost

TBŌN Output Theorem:
Output depends only on channel states and root filter state

All-encompassing Leaf State Theorem:
State at leaves subsume channel state (all state throughout TBŌN)

Therefore, leaf states can replace lost channel state without changing computation’s semantics



State Composition Algorithm

```
if detect child failure
    remove failed child from input list
    resume filtering from non-failed children
endif

if detect parent failure
    do
        determine/connect to new parent
    while failure to connect

    propagate filter state to new parent
endif
```