System-level Performance Management

Ken McDonell
Engineering Manager, CSBU
kenmcd@sgi.com

Overview

- Status quo for system-level performance monitoring and management in Linux.
- Factors conspiring to change this.
- Features of a desirable solution.
- Porting considerations.
- Support for distributed processing environments.

Influence of Linux Philosophies

- Anti-bloat mantra … available instrumentation is very sparse.
- 1-2p design center … many hard problems are off the radar screen.
- Developer-centric view leads to terse tools … and making them more like *sar* is not innovative.
- */proc/stat* model is both good and bad.
- Bias towards running tools on system under investigation.

Challenges to the Status Quo

- Linux deployment on larger platforms.
- Linux deployment in production environments.
- Cluster and federated server configurations.
- More complex application architectures.
- Focus shift from kernel performance:
  - applications performance is key
  - quality of service matters
  - systems-level performance mgmt

Large Systems Influences

- There may be a lot of data, e.g. for a large (128p) server 1000+ metrics and 30,000+ values from the platform & O/S.
- Data comes from the hardware, the operating system, the service layers, the libraries and the applications.
- Clustered and distributed architectures compound the difficulties.
- All of the data is needed at some time, but only a small part is needed for each specific problem.

Production Environment Influences

- Something is broken all of the time.
- Cyclic patterns of workload and demand.
- Transients are common.
- Service-level agreements are written in terms of performance as seen by an end-user.
- Environmental evolution changes the assumptions, rules and bottlenecks, e.g. upgrades, workload, filesystem age, re-organization.

Neanderthal Approaches

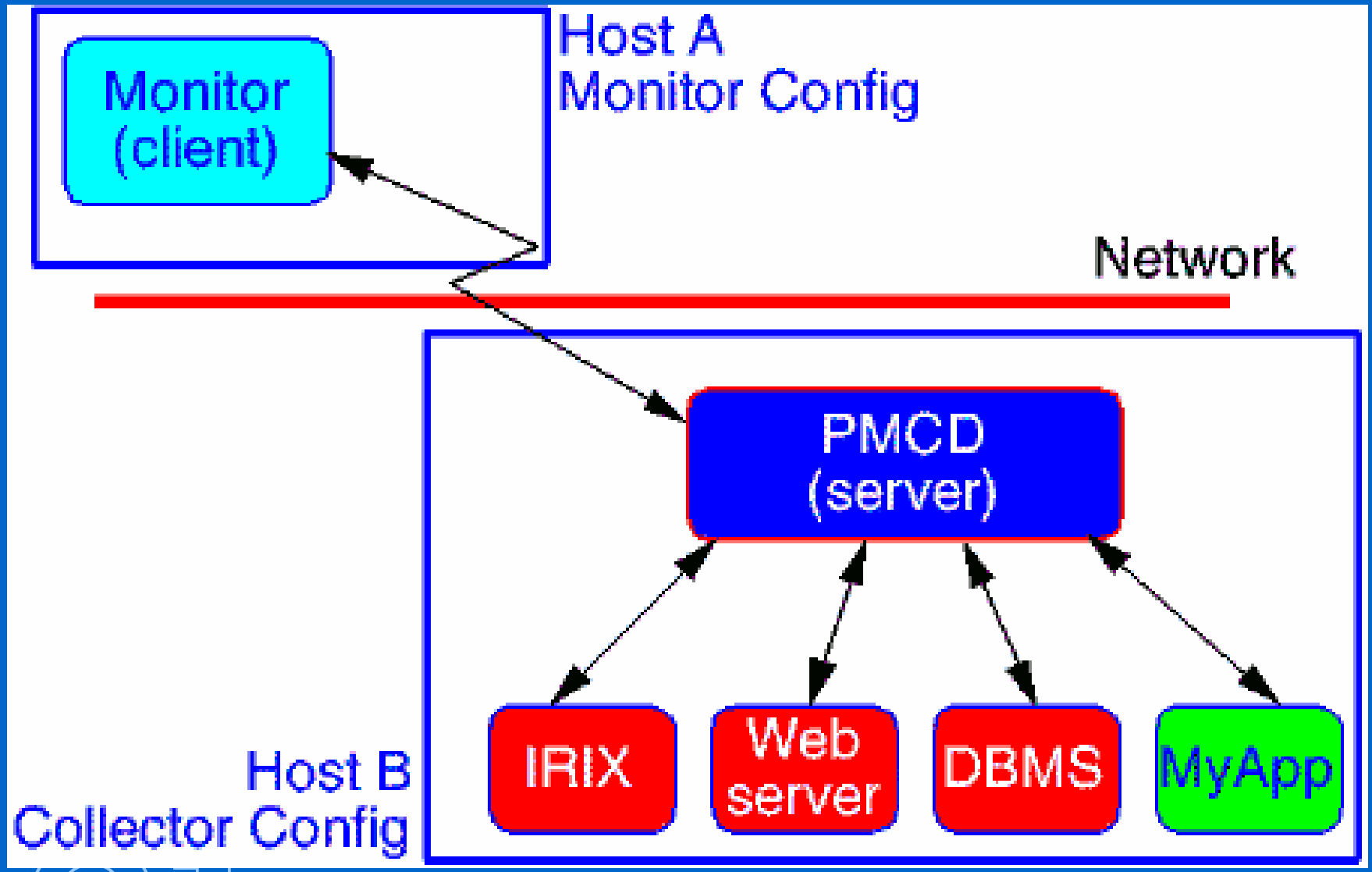
Making the Problem Harder

- Tool and data islands: ownership, functional, temporal and geographic domains.
- Primitive filtering and information presentation.
- Protocols and UIs that are not scalable.
- Emphasis on tools rather than toolkits.
- Very little automated monitoring that is useful for the hard problems.

Features of a Desirable Export Infrastructure

- Low overhead and small perturbation.
- Unified API for all performance data.
- Extensible (plug-in) architecture to accommodate new sources of performance data.
- Sufficient metadata to allow evolution and change.
- Support for remote access to performance data.
- Platform neutral protocols & data formats.

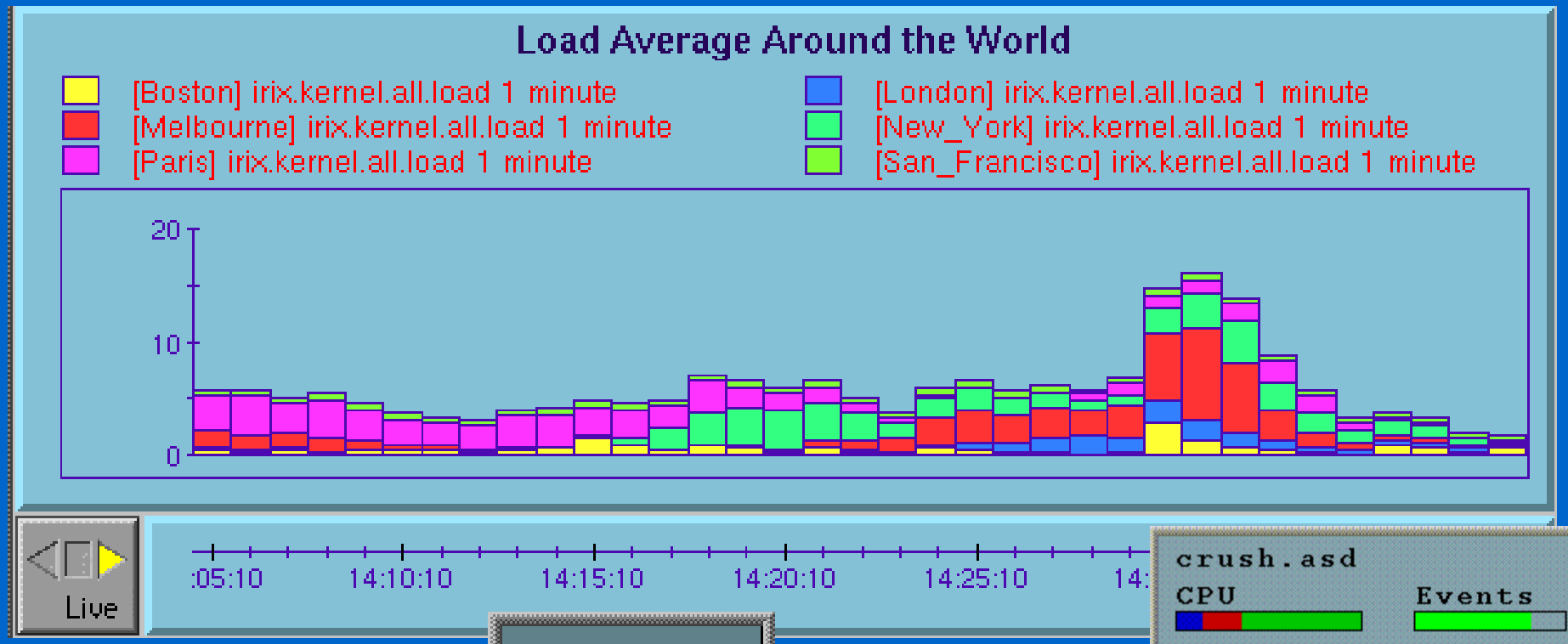
Plug-in Collector and Client-Server Architecture

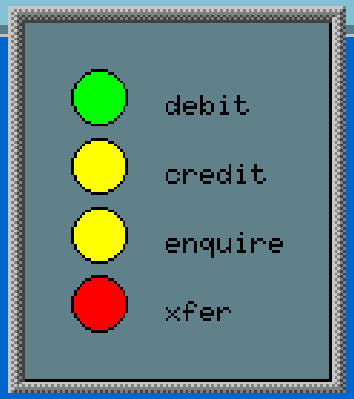


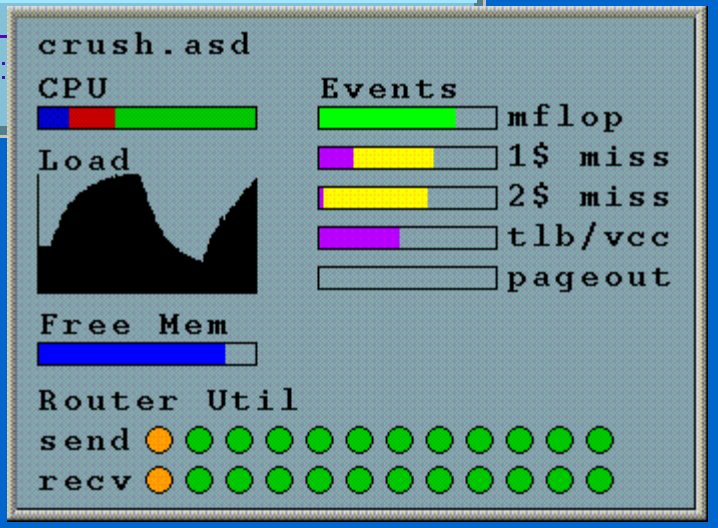
Features of a Desirable Performance Tool Environment

- Complement, not displace, simple tools.
- The same tools for both real-time and retrospective analysis.
- Visualization and drill-down user navigation.
- Remote and multi-host monitoring.
- Toolkits not tools.
- Smarter reasoning about performance data.

2-D Performance Visualization

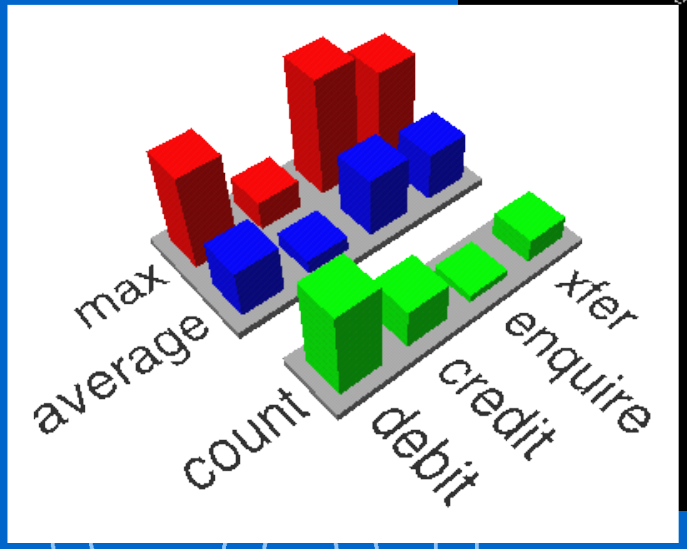




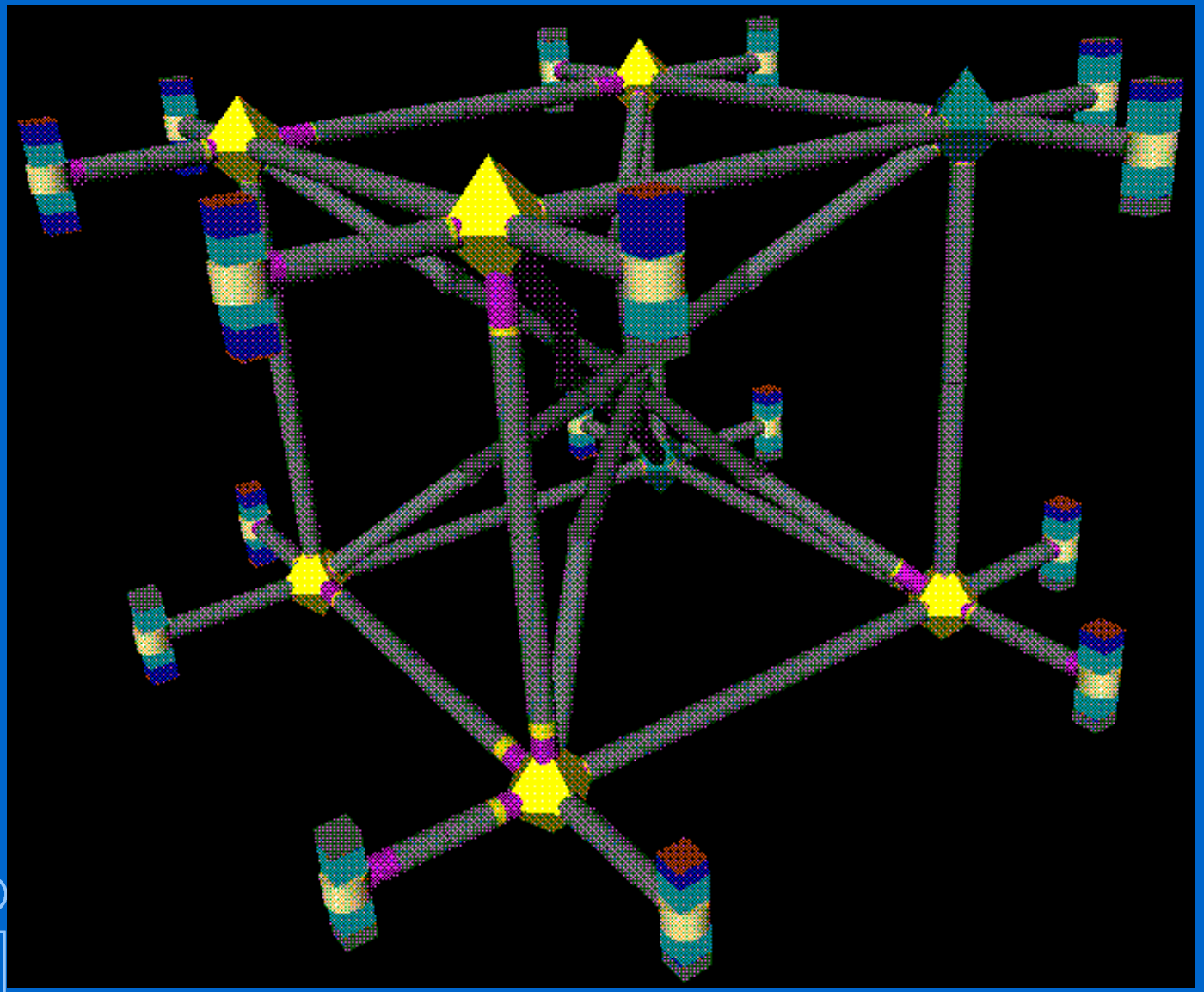


3-D Performance Visualization

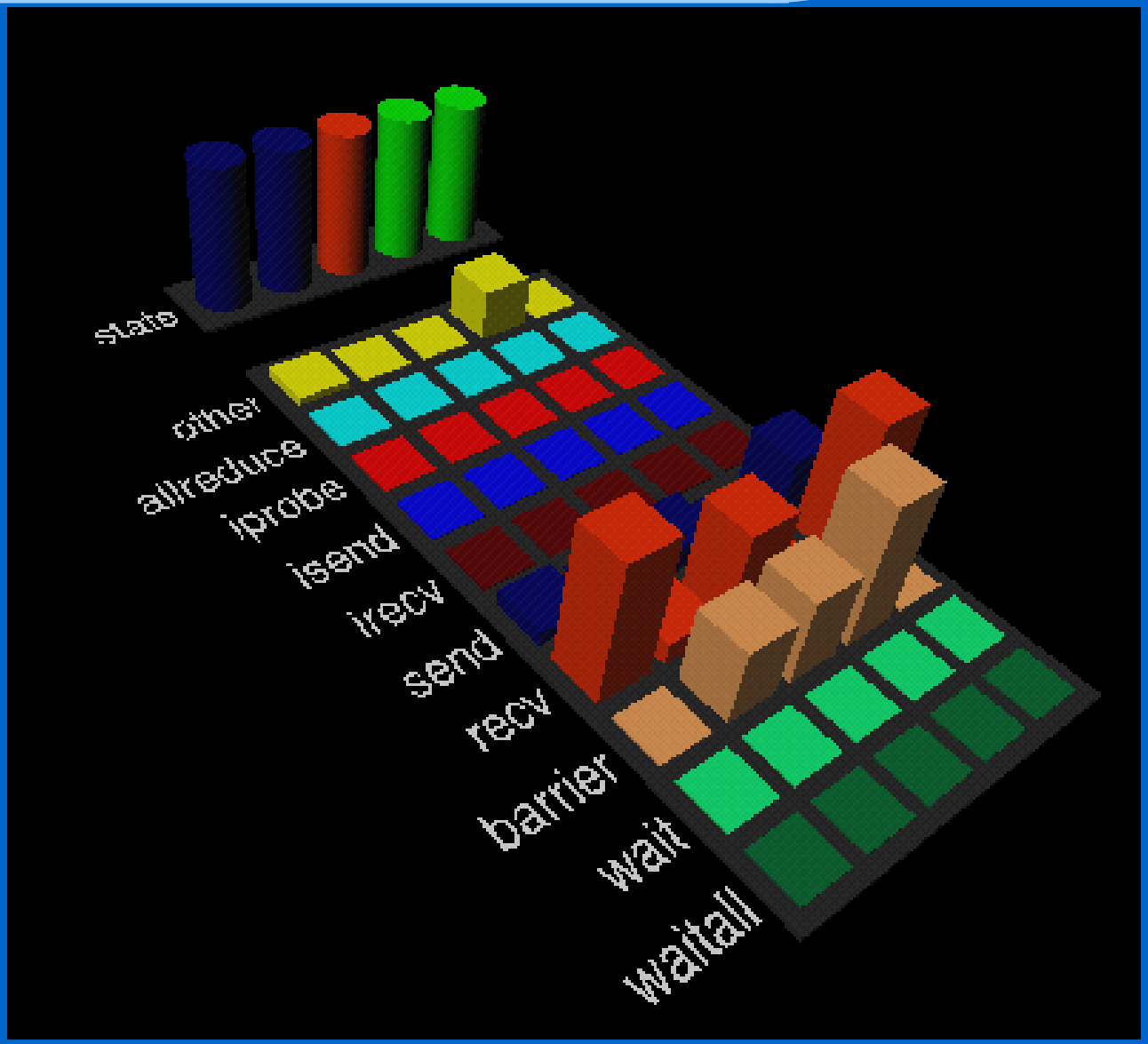
sgi



3-D Visualization of Platform Performance



3-D Visualization of Application Performance



Reasoning About Performance Data

Thresholds are not enough

- Need quantification predicates: existential, universal, percentile, temporal, instantial.
- Multi-source predicates for client-server and distributed applications.
- Retrospection is essential.
- Customized alarms and notification.

Performance Co-Pilot Porting History

- Initial development for IRIX
- 1994 Linux experiments
- 1995-96 HP/UX port
- 1998 NT port
- 1998-99 Linux port

Performance Co-Pilot Porting

Some things that did not help

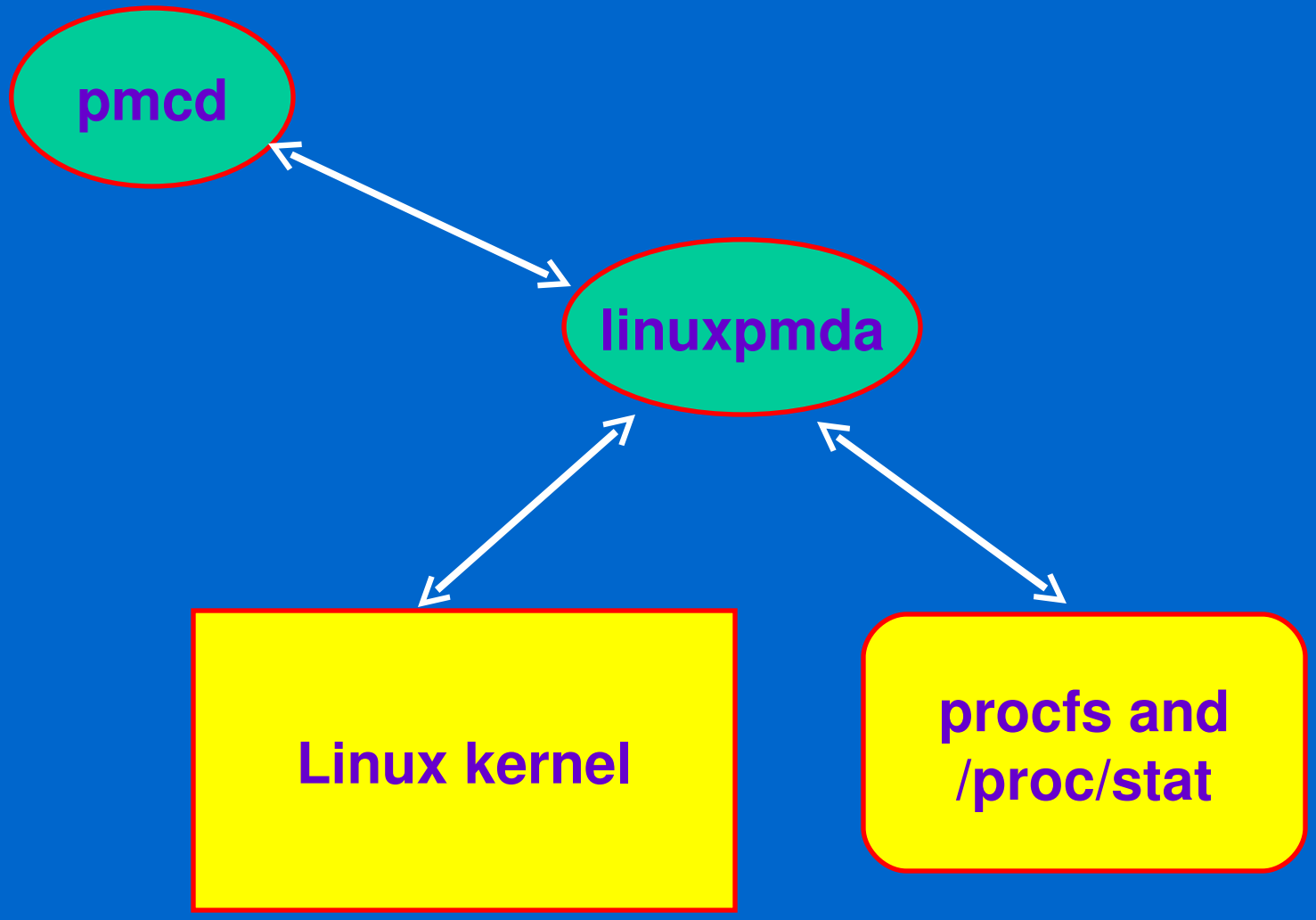
- For efficiency and historical reasons we'd chosen to avoid xdr and SNMP.
- HP/UX secrets.
- Lack of instrumentation in the Linux kernel.
- Tool frameworks used for IRIX development are not universally available, e.g. Motif, ViewKit, OpenInventor, XRT.

Performance Co-Pilot Porting

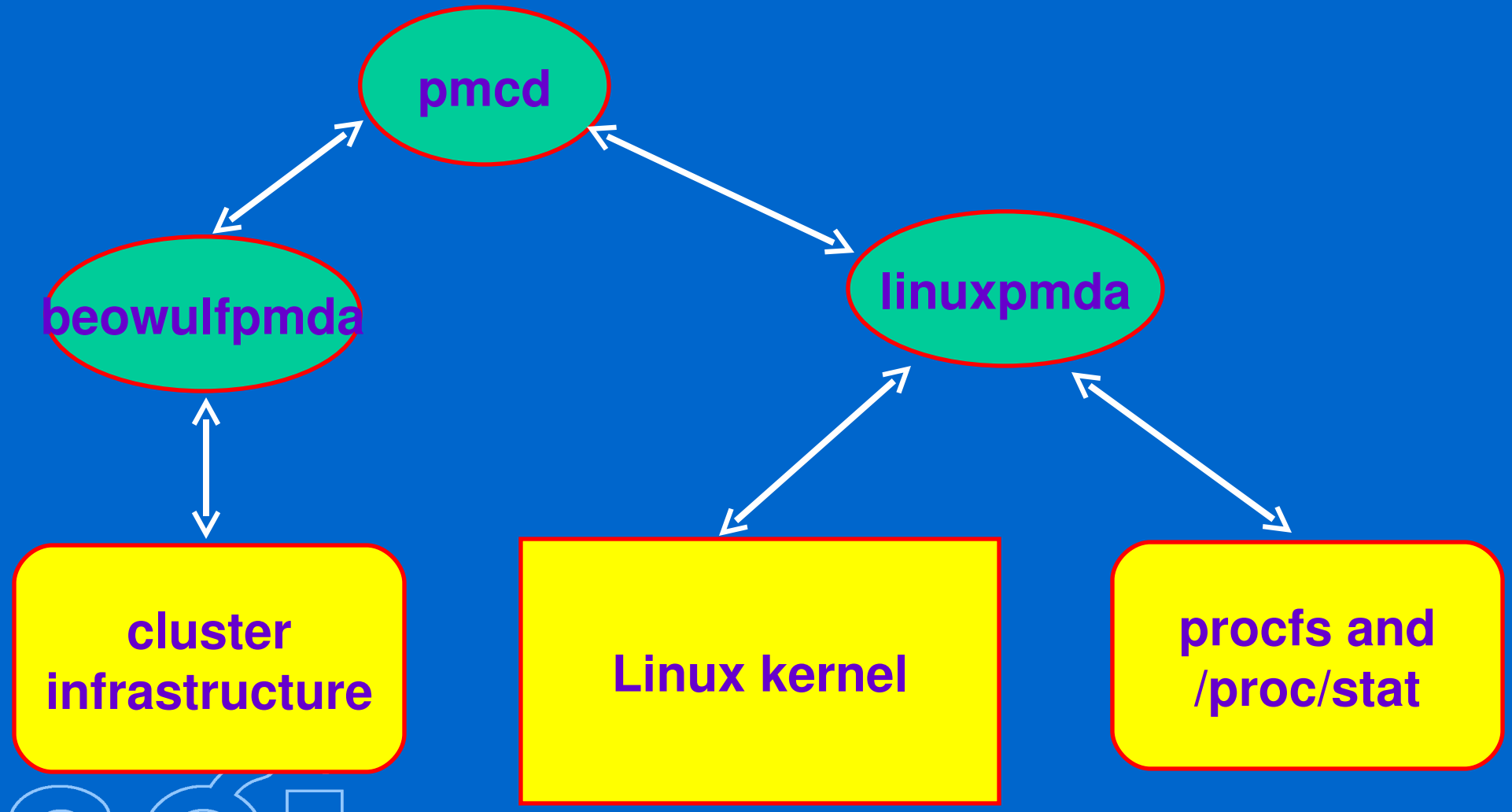
Some things that did help

- **Programmer discipline.**
- **Obsessive attitude to automated QA.**
- **Orthogonal functionality, especially for APIs.**
- **Monitoring tools that are predominantly shell scripts in front of a small number of generic applications (the “toolkit” approach).**

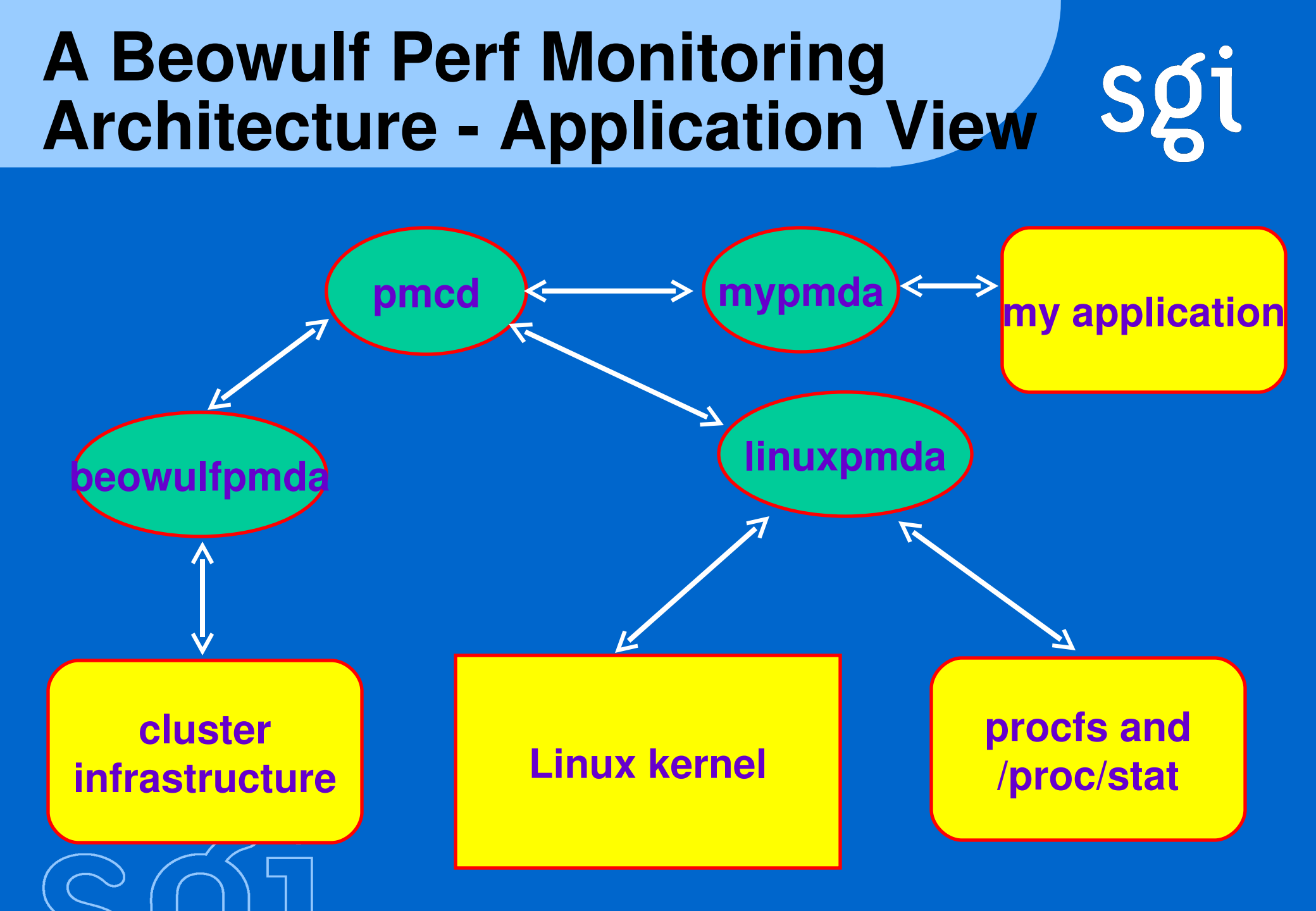
A Linux Performance Monitoring Architecture



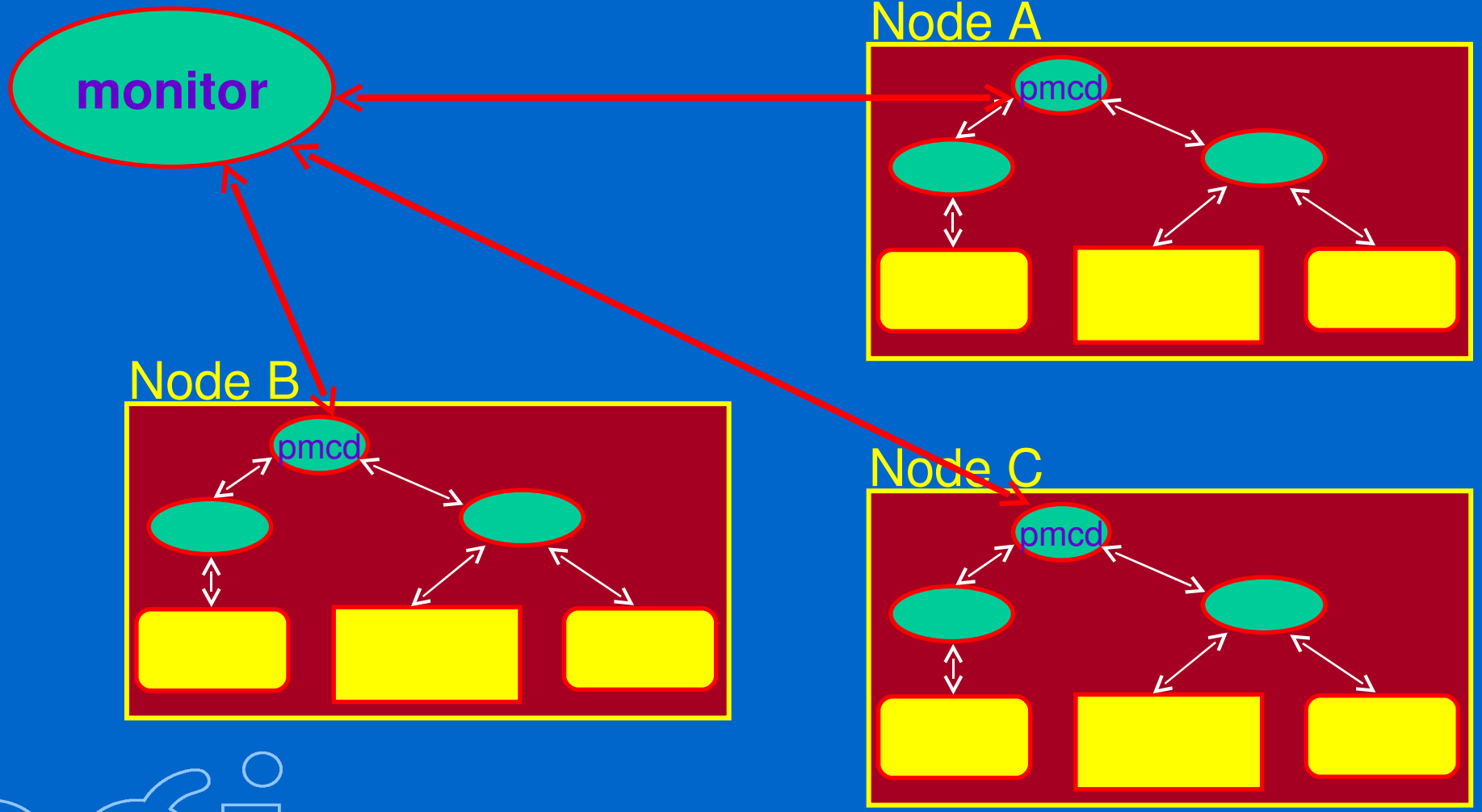
A Beowulf Perf Monitoring Architecture - Node View



A Beowulf Perf Monitoring Architecture - Application View



A Beowulf Perf Monitoring Architecture - Cluster View



Some Concluding Comments

- System-level performance management for large systems is a hard problem.
- Simple solutions do not exist.
- Need an extensible collection architecture
- Monitoring tools should provide centralized control for distributed processing.
- Retrospection is not optional.
- Linux offers real opportunities for “better” solutions in this area.