End-to-End Monitoring and Grid Troubleshooting

Brian L. Tierney

BLTierney@lbl.gov

Distributed Systems Department
Lawrence Berkeley National Laboratory

The Problem

- Assume a Grid job is:
  - submitted to a resource broker, uses a reliable file transfer service to copy several files, then runs the job.
- This normally takes 15 minutes to complete. But…
  - two hours have passed and the job has not yet completed.
- What, if anything, is wrong?
  - Is the job still running or did one of the software components crash?
  - Is the network particularly congested? Broken TCP stack?
  - Is the CPU particularly loaded?
  - Is there a disk problem?
  - Was a software library containing a bug installed somewhere?

The Solution

- End-to-End Monitoring
  - All components between the application endpoints must be monitored.
- This includes:
  - software (e.g., applications, services, middleware, operating systems)
  - end-host hardware (e.g., CPUs, disks, memory, network interface)
  - networks (e.g., routers, switches, or end-to-end paths)

Monitoring Components

- A complete End-to-End monitoring framework includes:
  - Instrumentation Tools
    - Facilities for precision monitoring of all software (applications, middleware, and operating systems) and hardware (host and network) resources
  - Monitoring Data Publication
    - Standard schemas, discovery and publication mechanisms, and access policies for monitoring event data are required
  - Sensor Management
    - The amount of monitoring data produced can quickly become overwhelming
    - A mechanism for activating sensors “on demand” is required
  - Data Analysis Tools
    - event analysis and visualization tools
  - Event Archives
    - Historical data used to establish a baseline to compare current and predict future performance

Uses for Monitoring Data

- Troubleshooting and Fault Detection
  - Detect failures and recovery
- Performance analysis and tuning
  - Better program design (e.g.: will better pipelining of I/O and computation help?)
  - Network-aware Applications (TCP buffer size tuning, # of parallel streams, etc.)
- Debugging
  - Complex, multithreaded, distributed programs are difficult to debug without the proper monitoring data
- Guiding scheduling decisions
  - Grid Schedulers
    - Find the best match of CPUs and data sets for a given job
  - Grid Replica Selection
    - Find the “best” copy of a data set to use
- Auditing and intrusion detection

Grid Troubleshooting Example

- Step 1: insert instrumentation code during the development stage
  - to ensure the program is operating as expected
- Step 2: establish a performance baseline for this service, and store this information in the monitoring event archive.
  - Include system information such as processor type and speed, OS version, CPU load, disk load, network load data, etc.
- Step 3: put service into production, and everything works fine
  - Until…..
- One day users start complaining that service X is taking much longer than previously

Grid Troubleshooting Example

- To collect data for analysis, one must:
  - Locate sources of relevant monitoring data, and “subscribe” to that data.
  - Activate any missing sensors, and subscribe to the their data.
  - Activate debug-level instrumentation in the service, and subscribe.
  - Locate monitoring data in the monitoring event archive for the baseline test from when things were last working.
- Data analysis can be then begin:
  - Check the hardware and OS information to see if anything changed.
  - Look at the application instrumentation data to see if anything looks unusual.
  - Look at the system monitoring data to see of anything looks unusual (e.g., unusually high CPU load).
  - Correlate the application and middleware instrumentation data with the host and network monitoring data.

U.S. Department of Energy
Office of Science

Troubleshooting Example: Step 1: Generate Grid Job "Lifeline"

BERKELEY LAB

Application Events

Copy output data

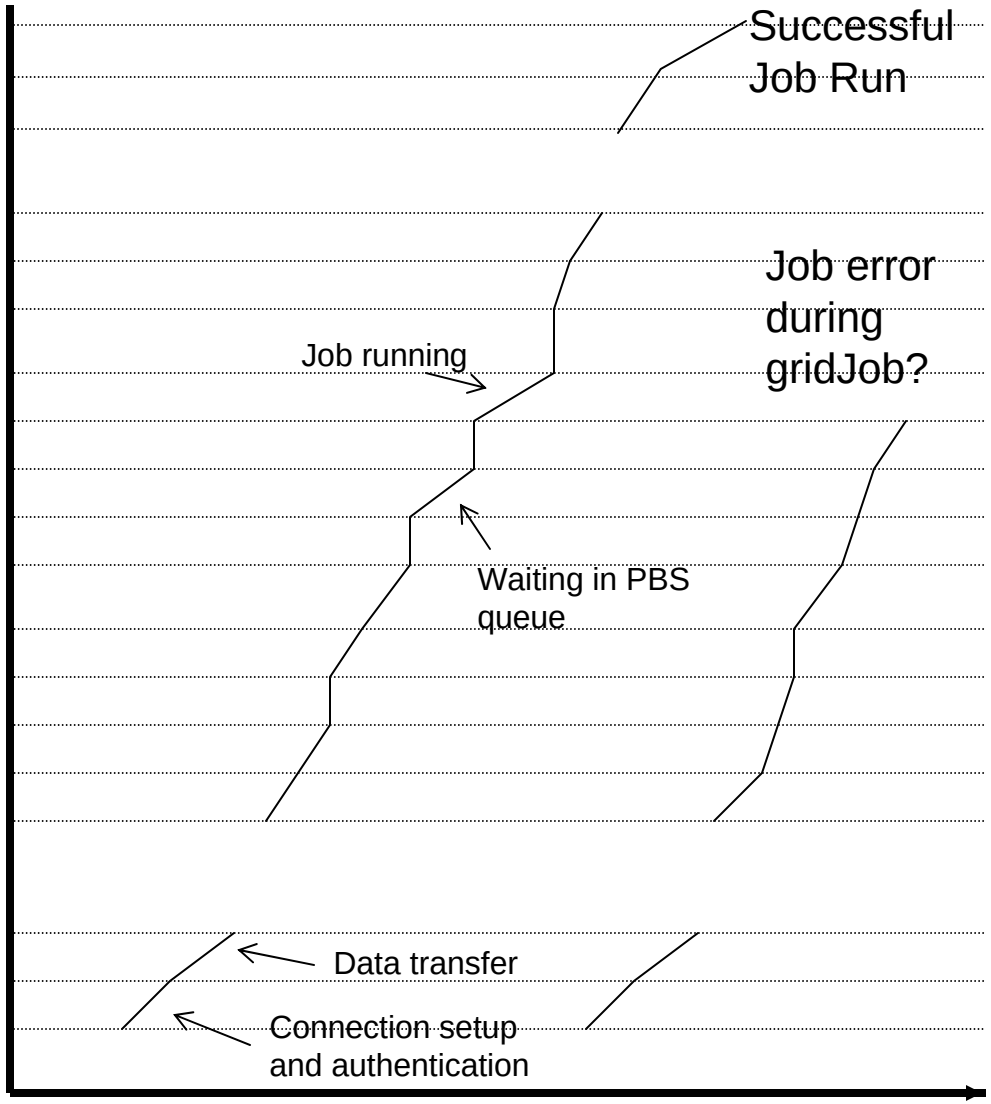
Run Grid Job

Copy input data

GlobusUrlCopy.put.end
GlobusUrlCopy.put.transferStart
GlobusUrlCopy.put.start

GlobusJobRun.end
jobManager.end
jobManger.jobState.done
gridJob.end
gridJob.start
jobManager.jobState.active
jobManager.jobState.pending
akentiAuthorization.end
akentiAuthorization.start
gateKeeper.end
jobManager.start
gateKeeper.start
GlobusJobRun.start

GlobusUrlCopy.get.end
GlobusUrlCopy.get.transferStart
GlobusUrlCopy.get.start

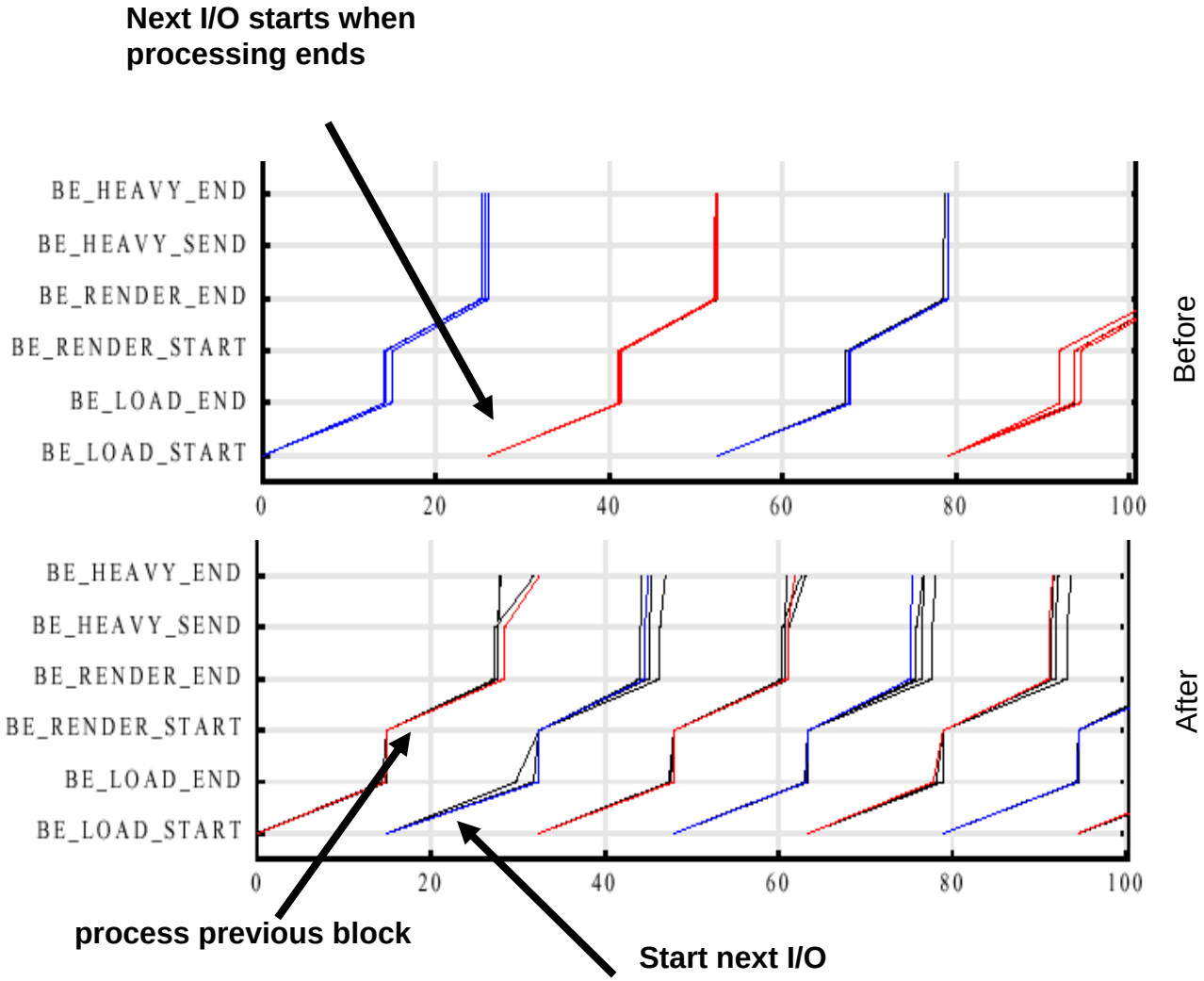


U.S. Department of Energy
Office of Science

Step 2: Add detailed application instrumentation, (1st example)

BERKELEY LAB

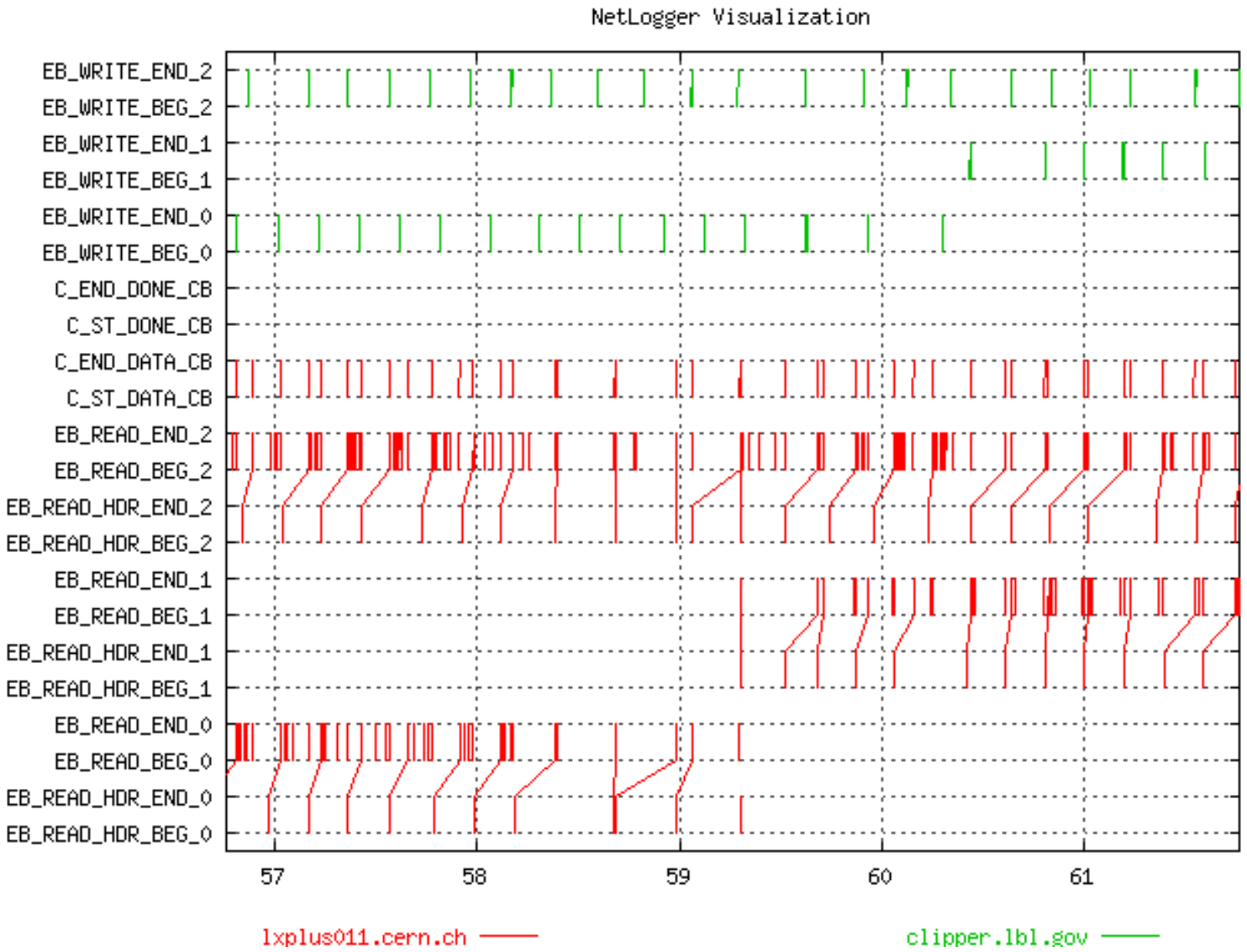
- I/O followed by processing
- overlapped I/O processing



almost a 2:1 speedup

U.S. Department of Energy
Office of Science

Step 2: Add detailed application instrumentation, (2nd example)

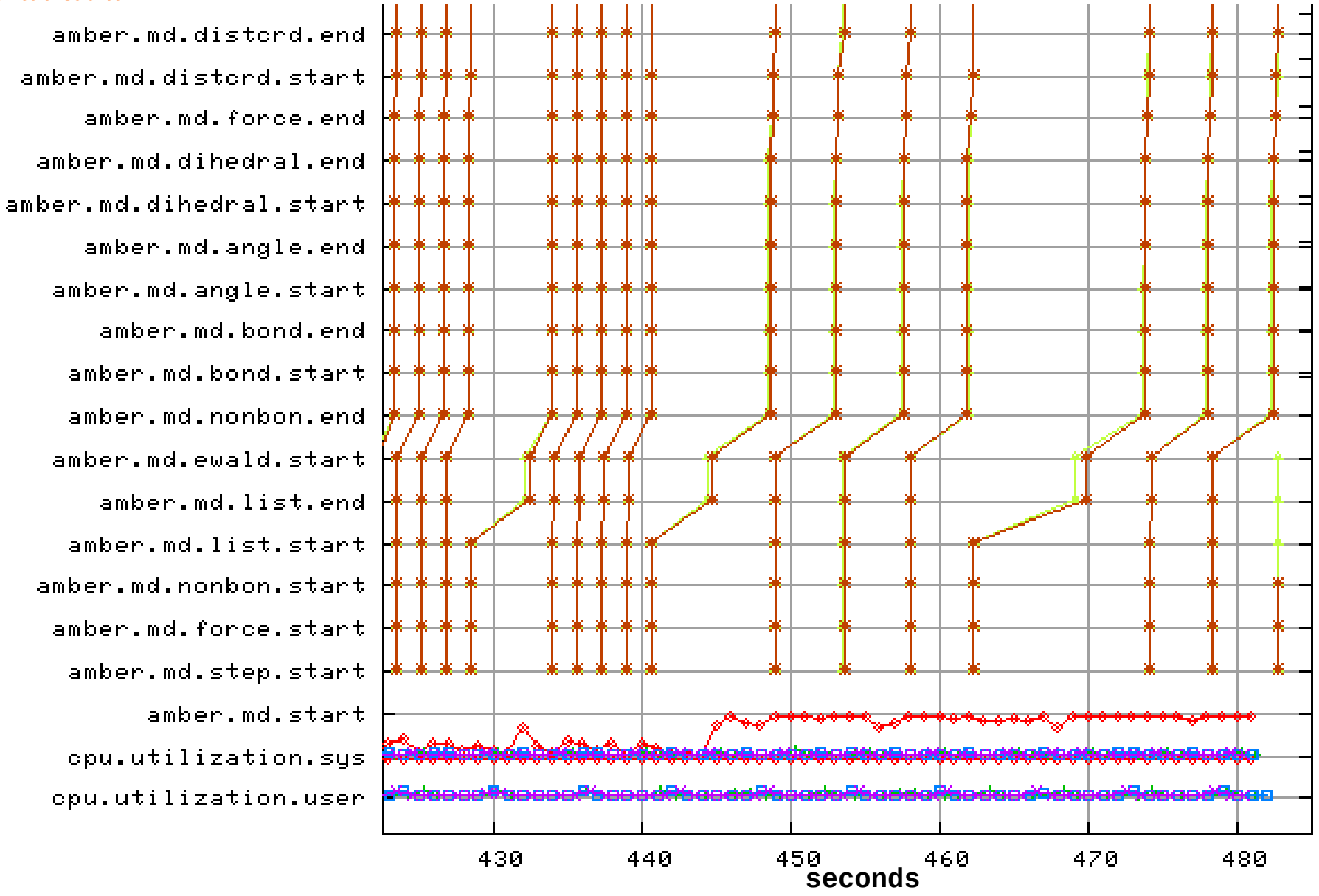


U.S. Department of Energy

Office of Science

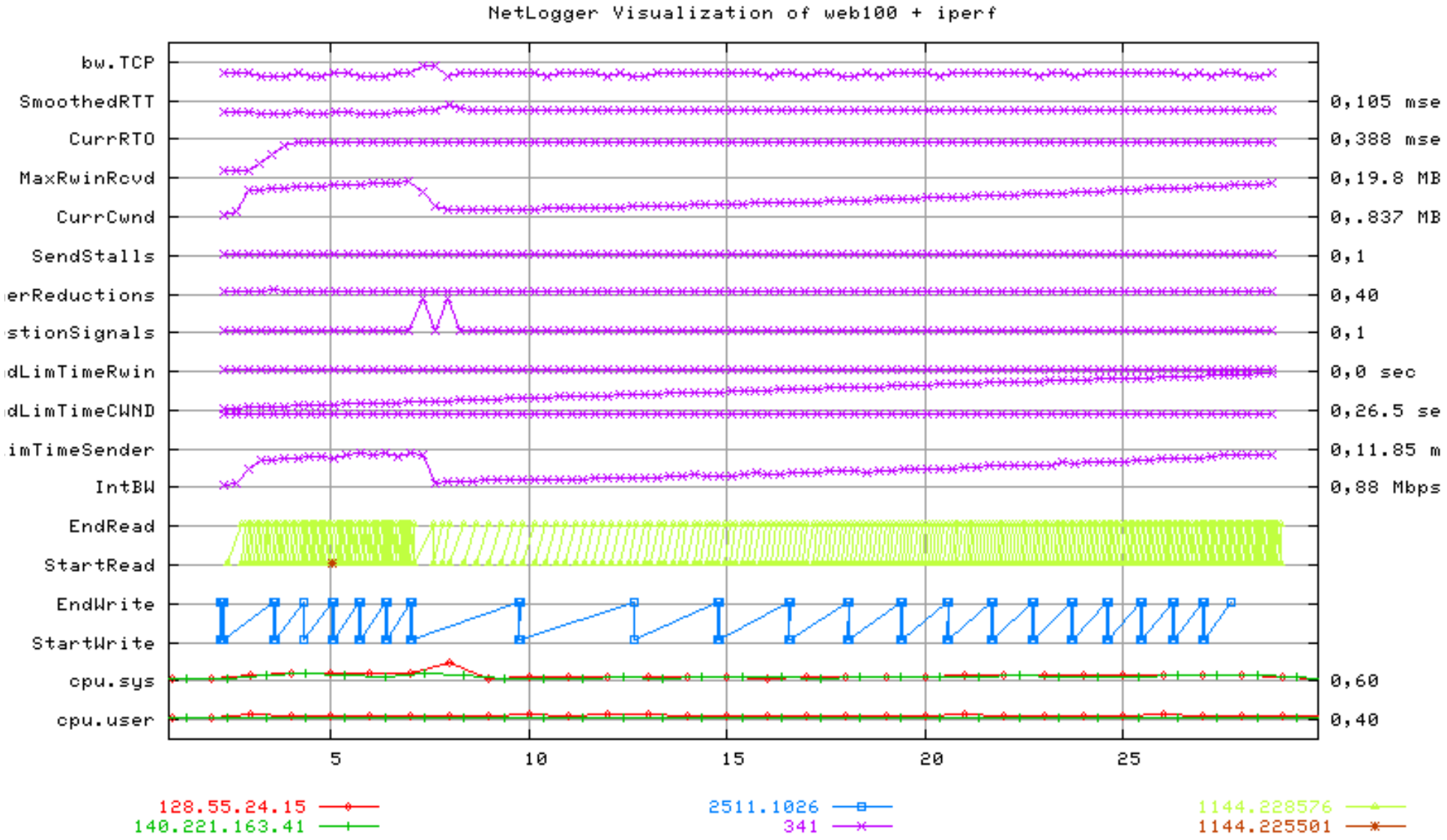
Step 3: add host monitoring

(e.g.: CPU load or TCP retransmits)



U.S. Department of Energy

Step 4: add TCP monitoring



Challenges

- How to getting developers to instrument their applications/middleware?
  - Issue: Getting common message formats and common "Grid IDs" into all log messages
- How to export network monitoring data to Application/middleware?
  - Issues: Schemas, APIs, prediction, derived metrics, ….

U.S. Department of Energy
Office of Science

For More Information

BERKELEY LAB

- http://dsd.lbl.gov/NetLogger/
  - All software components are available for download under DOE/LBNL open source license (BSD-style)
- email: BLTierney@LBL.GOV
- Other Useful URLs:
  - http://dsd.lbl.gov/NMWG/ (GGF Network Measurement WG)
  - PFLDnet 2004: http://dsd.lbl.gov/PFLDnet2004/program.htm
  - TCP Tuning: http://dsd.lbl.gov/TCP-tuning/TCP-tuning.html

Extra Slides

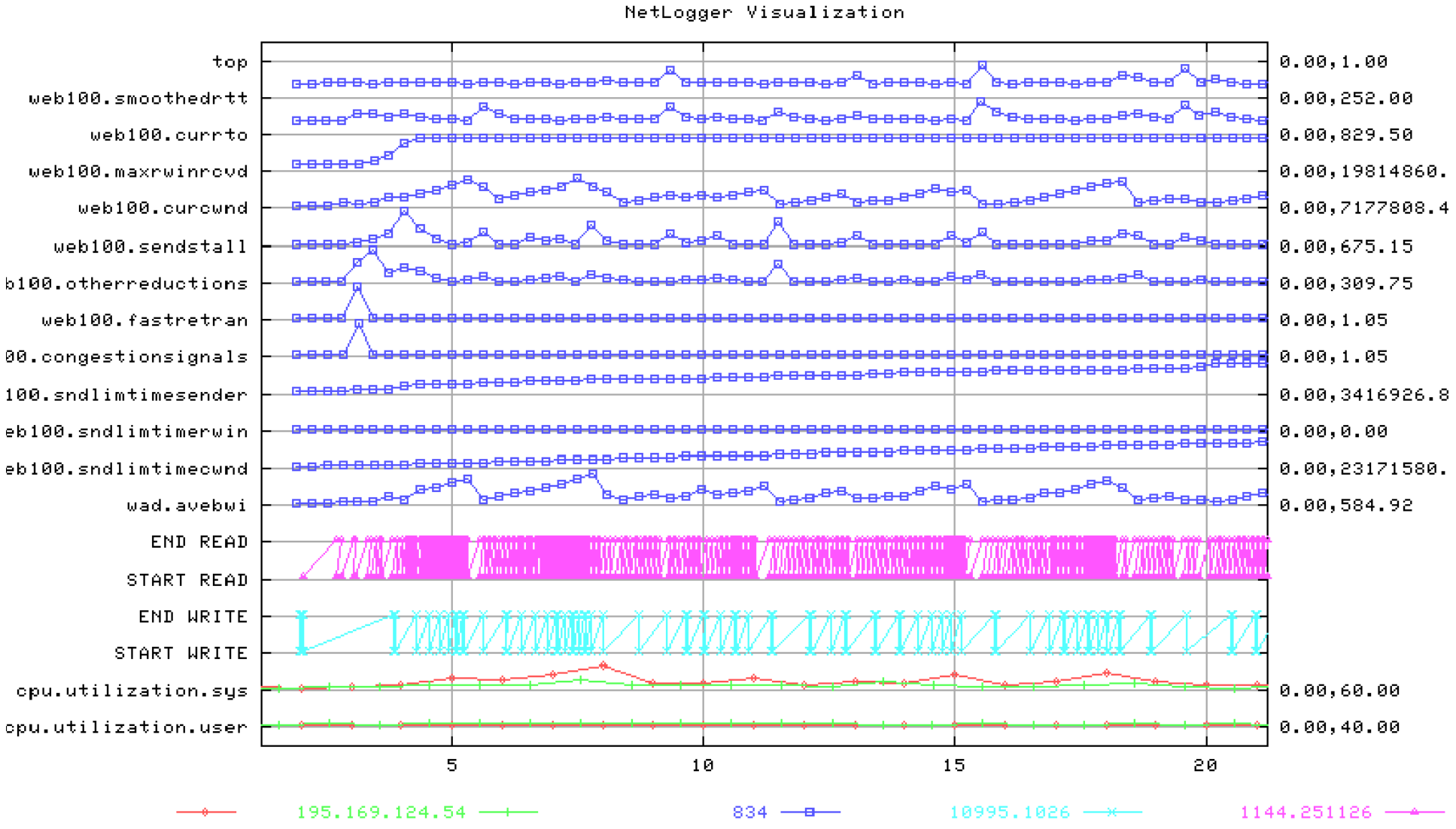
Grid Job ID

- In order to graphically link events from several Grid components
  - monitoring events for the same “job” needs the same “Grid Job ID” (GID)
- We have instrumented the following pyGlobus components with NetLogger with a GID
  - globus-job-run, globus-url-copy, Globus gatekeeper Globus job manager
  - globus-job-run generates the GID using *uuidgen*
  - GID passed to gatekeeper via RSL

U.S. Department of Energy

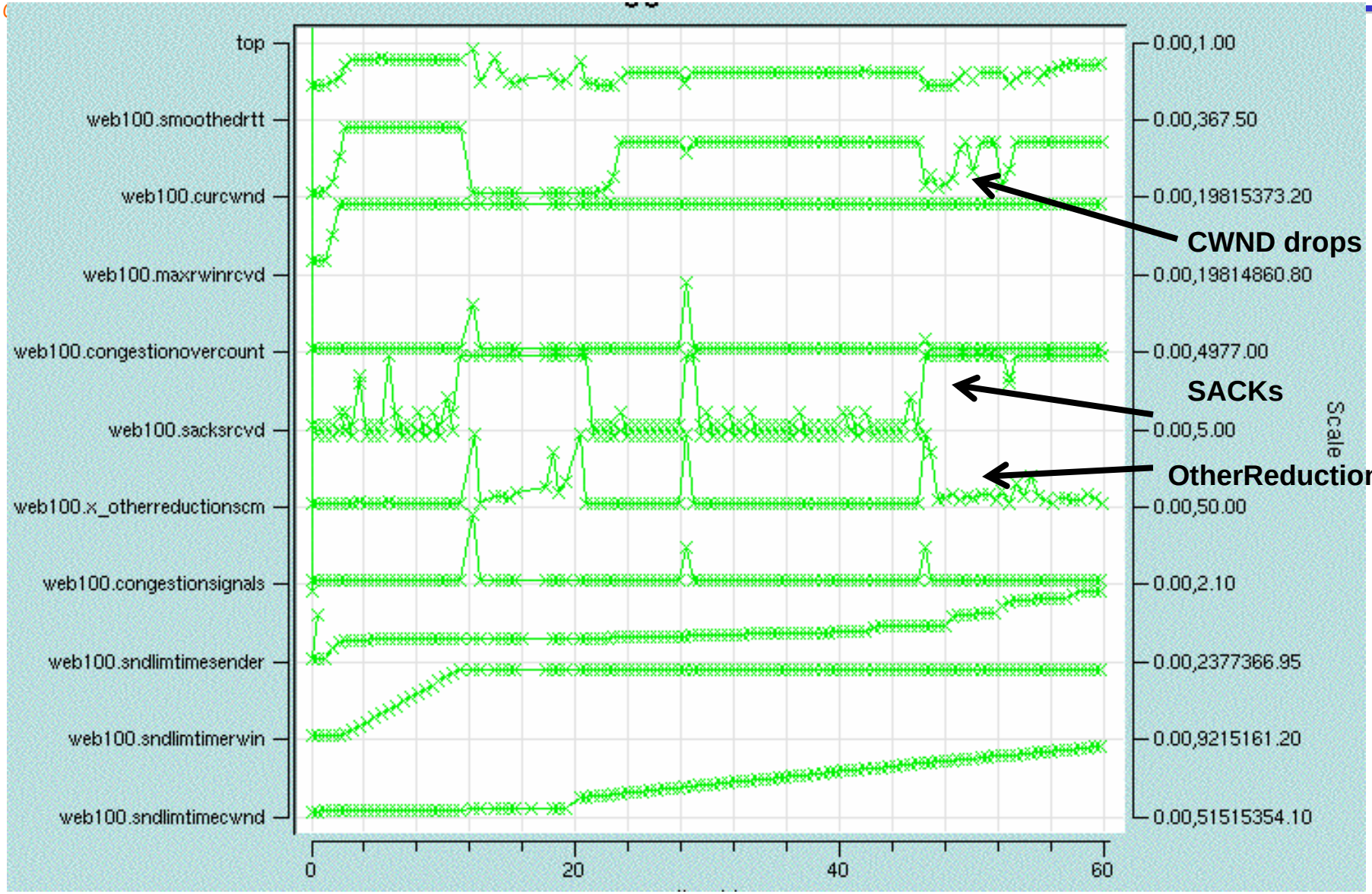
Office of Science

Step 3b: add more TCP monitoring



U.S. Department of Energy

Detailed TCP Analysis: Correlation of Sack and OtherReductionsCM



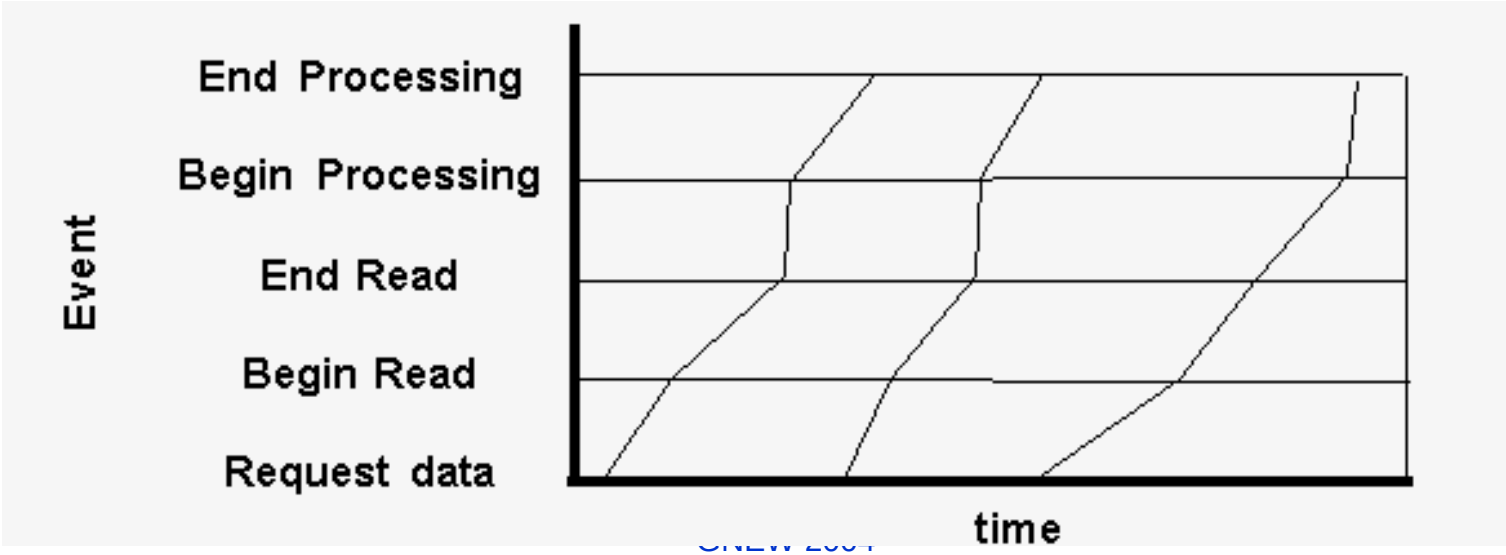
NetLogger Toolkit

NetLogger Toolkit

- We have developed the NetLogger Toolkit (short for Networked Application Logger), which includes:
  - tools to make it easy for distributed applications to log interesting events at every critical point
    - NetLogger client library (C, C++, Java, Perl, Python)
    - Extremely light-weight: can generate > 900,000 events / second on current systems (9000 events / sec with 1% app. perturbation)
  - tools for host and network monitoring
  - event visualization tools that allow one to correlate application events with host/network events
  - NetLogger event archive and retrieval tools
- NetLogger combines network, host, and application-level monitoring to provide a complete view of the entire system.

NetLogger Analysis: Key Concepts

- NetLogger visualization tools are based on time correlated and object correlated events.
  - precision timestamps (default = microsecond)
- If applications specify an "object ID" for related events, this allows the NetLogger visualization tools to generate an object "**lifeline**"
- In order to associate a group of events into a "lifeline", you must assign an "object ID" to each NetLogger event
  - Sample Event ID: file name, block ID, frame ID, Grid Job ID, etc.



U.S. Department of Energy
Office of Science

Sample NetLogger Instrumentation


```
log = netlogger.open("x-netlog://log.lbl.gov","w")
done = 0
while not done:
  log.write(0,"EVENT_START","TEST.SIZE=%d",size)
  # perform the task to be monitored
  done = do_something(data,size)
  log.write(0,"EVENT_END")
```

- Sample Event:

```
DATE=20000330112320.957943 HOST=gridhost.lbl.gov \
PROG=gridApp LVL=1 NL.EVNT=WriteData SEND.SZ=49332
```

NetLogger Activation Service

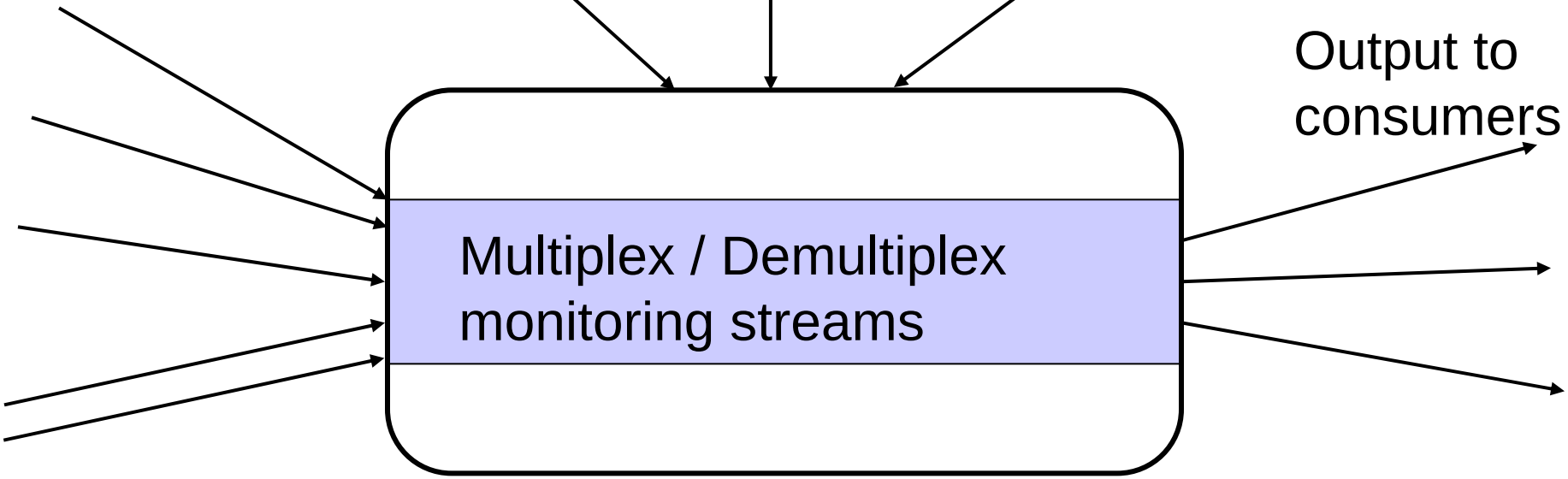
- Do not want all monitoring data collected all the time
  - Potentially way too much data
  - Need to adjust the level of monitoring as needed for:
    - Debugging
    - Performance tuning
    - Error analysis
- NetLogger Activation Service addresses this issue
  - NetLogger-based sensors register with the activation service
  - Very useful debugging tool for MPI / PC cluster-based jobs

NetLogger Filter and Activation Service

Subscription A: send me all monitoring data for Grid Job # 23

Subscription B: send all level 0 monitoring data to archive at host a.lbl.gov

Subscription C: change the logging level of program *ftpd* to level 2, and send me the results



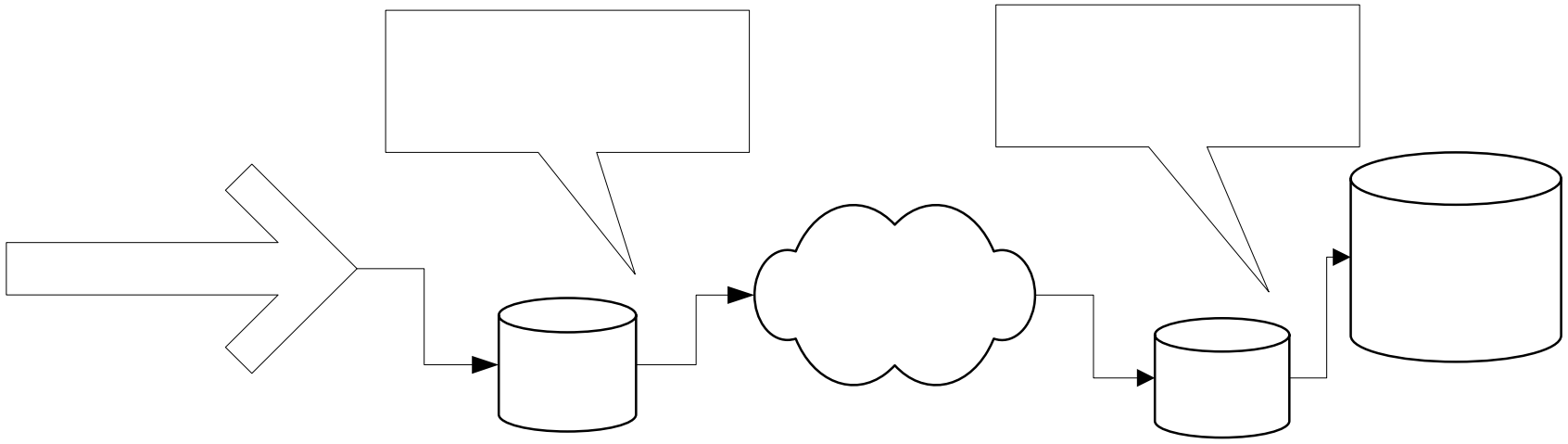
Incoming monitoring data:

- application,
- middleware,
- host

NetLogger Filter and Activation Service

NetLogger Archive Architecture

- Architecture must be scalable and capable of handling large amounts of application event data,
- None of the components can cause the pipeline to “block” while processing the data, as this could cause the application to block
  - For example, instrumented FTP server could send > 6000 events/second to the archive (500 KB/sec (1.8 GB/hr) of monitoring event data)



NetLogger Tools

- nlforward: Log file forwarder
  - forwards a single NetLogger file or directory of files to an output URL

- netlogd: TCP socket server daemon
  - accepts one or more NetLogger TCP streams and writes them to one or more NetLogger output URL's

U.S. Department of Energy

Office of Science

Conclusions

- Ability to allow a Grid User or developer to easily “drill down” from high-level to low-level analysis it needed
- Grid ID is essential for correlating events