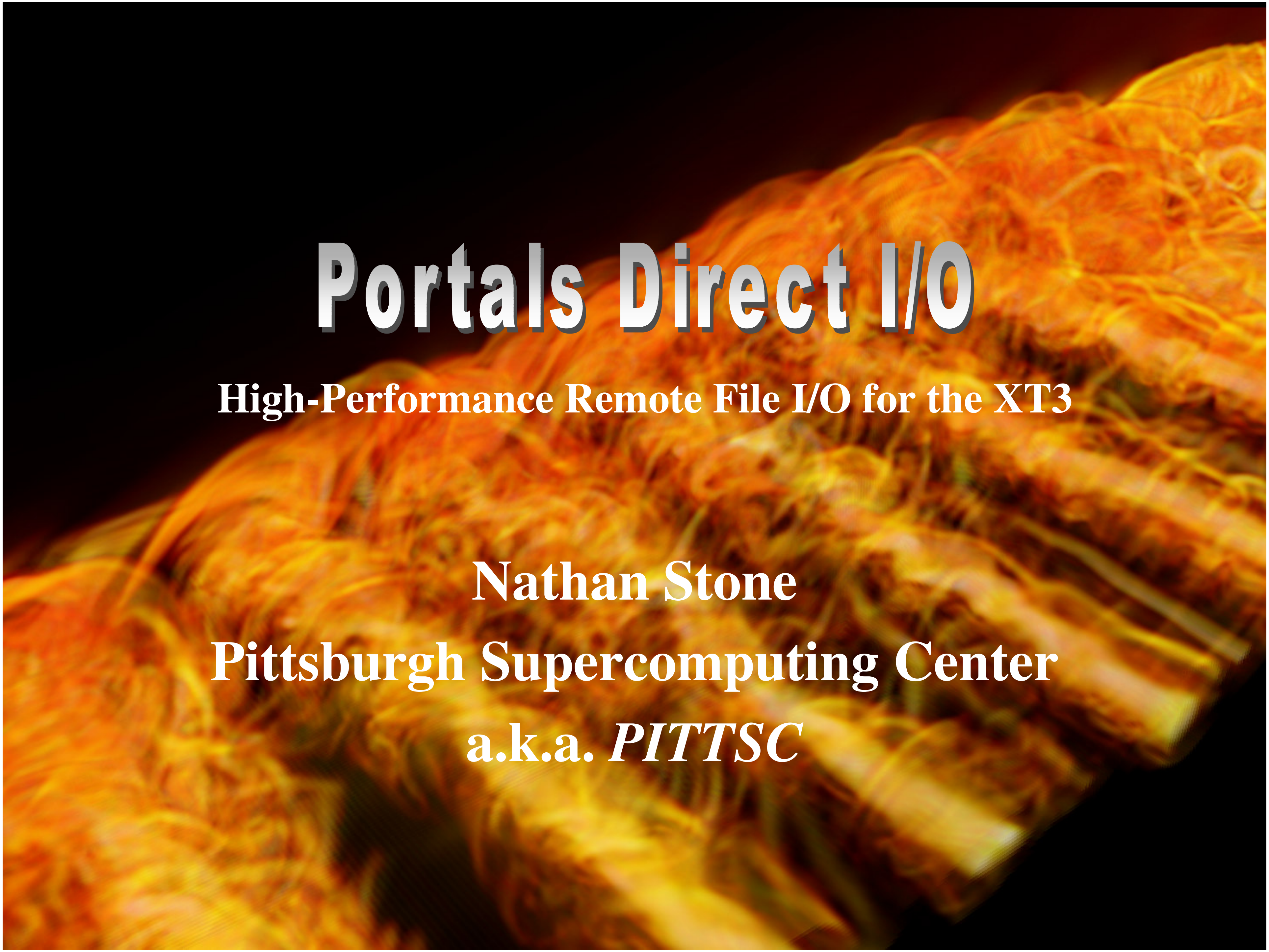


Portals Direct I/O

High-Performance Remote File I/O for the XT3

Nathan Stone

Pittsburgh Supercomputing Center

a.k.a. *PITTSC*

Explicit User Demand (!)

- The Woodward collaboration (umn.edu) “killer app”
  - Piecewise Parabolic Method (***“PPM”***)
    - Compressible turbulent fluid dynamics
  - Real-time visualization (or playback)
  - Interactive control (game controllers at SC|05) and steering
  - But it was missing *one* piece…
- Needed: ***“PDIO”***
  - Real-time remote file delivery
    - For interactive visualization
  - High-performance (100 MByte/sec)
    - For interactive timescales

Cray XT3: “BigBen”

tg-login.bigben.psc.teragrid.org

BigBen’s Relevant Characteristics

Compute Nodes (2068)

- Catamount (QK) *micro*kernel
  - No `fork/thread`
- No external connectivity (nor IP stack)
- All IPC is over Portals
- Presents a problem for getting in-memory data *directly* to remote hosts

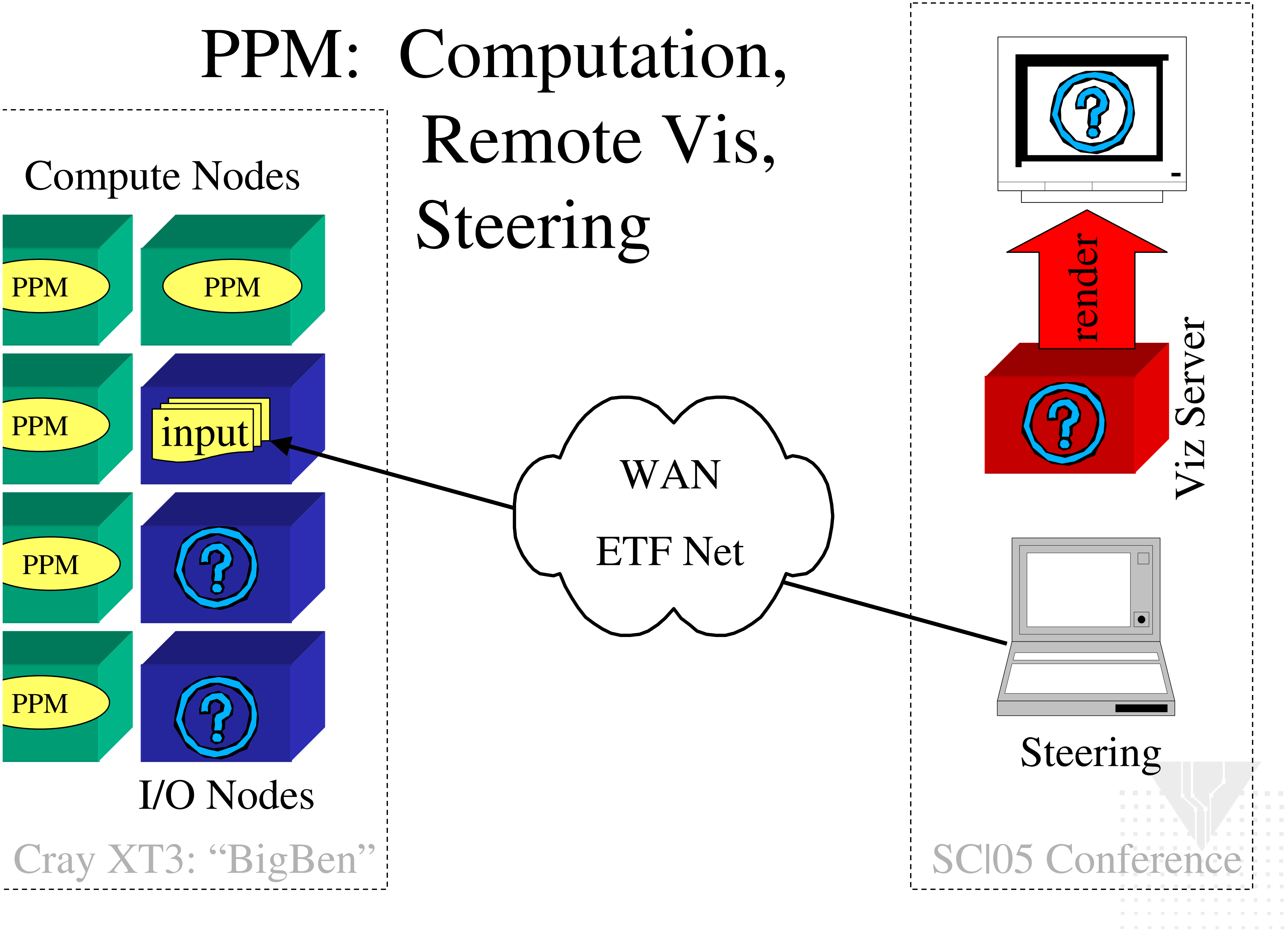
Service & I/O Nodes (22)

- Full Linux OS
- Portals/Seastar internal net
- 10/GigE external net (26 Gbit/sec)
- Connected to TeraGrid, Abilene/I2, commodity…
- Ideal place for data routing

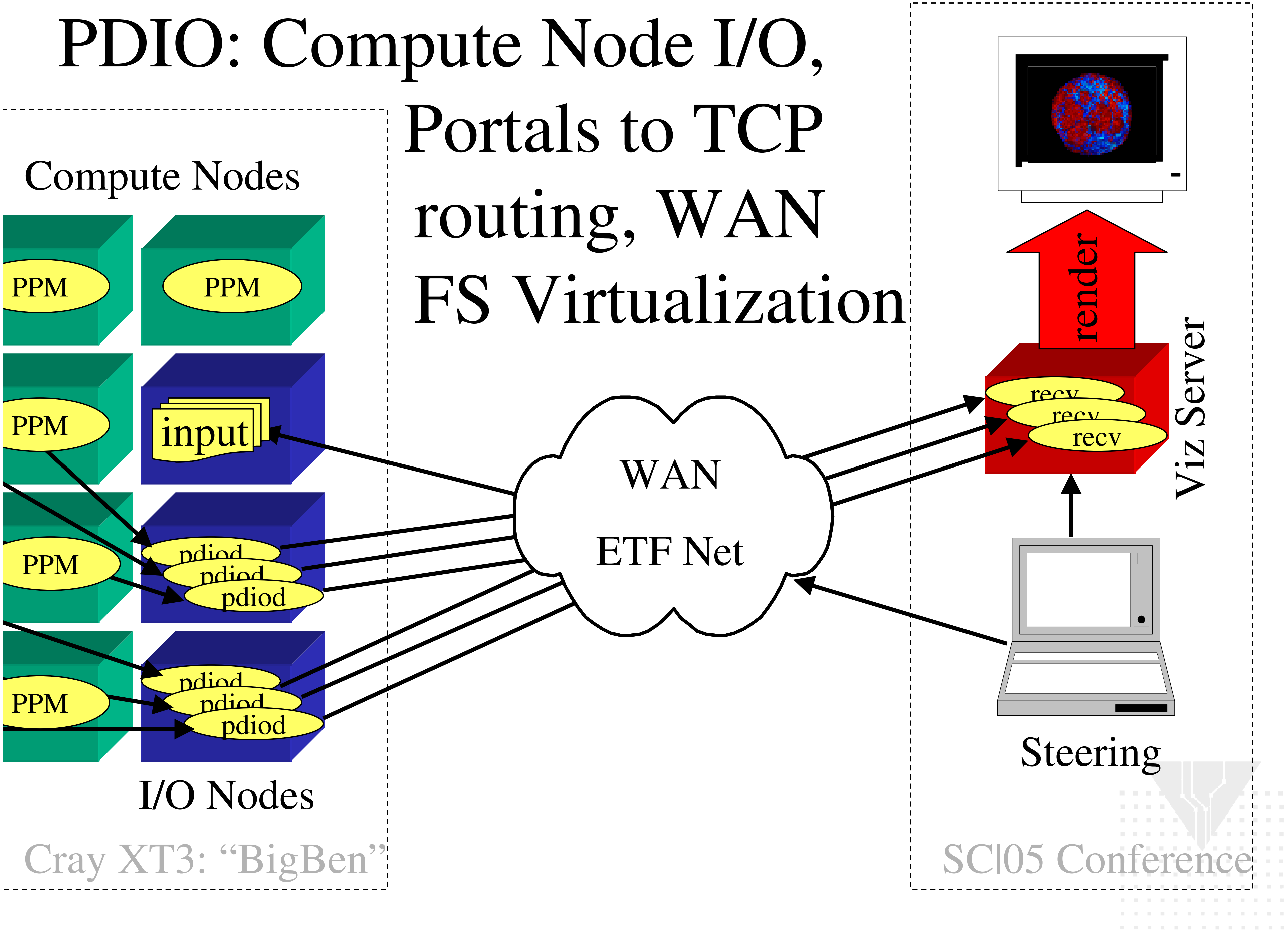
PPM meets ***PDIO***

Anatomy of a Live Demo

PPM: Computation, Remote Vis, Steering



PDIO: Compute Node I/O, Portals to TCP routing, WAN FS Virtualization



Prototype Results

- Functionality
  - Demonstrated live at three different int'l conference venues (and two labs)
  - No PDIO failures after *alpha* deployment
- Performance
  - First Results: Maxed out the LCSE's link at 40 MByte/sec
  - Delivered >125 MByte/sec to the SC|05 show floor (10GigE)
  - Sustained 25 MByte/sec for 90 minutes
    - User-limited to avoid filling remote disk
- Interface
  - Fixed API (e.g. `pdio_write`), required 35 lines of code mods
  - Individual writes constrained to unique files

Recap: Prototype Design Goals

- Seamless remote file delivery
  - As *files*, for post-processing
    - Target was remote FS, not process (socket, memory, etc.)
  - Via "normal" `write()` mechanisms
    - Look like local writes: for testing, compatibility, Q/A

Write files...

- High performance
  - >100 MByte/sec (for interactive feedback)

Really fast...

- Arbitrary destinations
  - Portable remote receiver (never know where…)
  - Build/install suitable to "research admins"
    - People who don't read GQ (Globus Quarterly)

Wherever I want

Prototype Implementation Details

- Portals-to-TCP routing
  - Heterogenous Portals (QK-Linux), a la Lustre & YOD
  - Daemons aggregate incoming portals data streams (many-to-1) into outgoing TCP streams
- Explicit Parallelism
  - Configurable # of daemons (on SIO nodes)
    - Distributed across multiple 10GigE-connected Service & I/O (SIO) nodes
  - Corresponding # of TCP streams (over the WAN)
    - one per daemon/target recvr pair
  - Parallel TCP recvrs (on remote hosts)
    - Supports a variable/dynamic number of connections

Prototype Implementation Details

- Inherent flow-control feedback to application
  - Aggregation protocol allows TCP transmission or remote FS to throttle the data streams coming out of the application
    - just like a local FS
    - *"That's not a bug, that's a feature."*
- Multi-threaded ring buffer in the PDIO daemon
  - Allows computation/Portals receiver/TCP sender to proceed asynchronously

Toward a General Solution

- Although it worked for Woodward…
  we still weren't satisfied
  - Users 2-N are coming… (*e.g.* Δ src, 1 wr-1f, wronly)

Top three priorities:

1. Transparent invocation
2. File semantics
3. Resource management
4. (Read)

Revised Design: 1-Invocation

- Transparent invocation (eliminate explicit API)
  - Invoked via intercept library (`open, write, close`)
  - **Now**: No changes to source code!
- ➢ But how? (the Challenges)
  - Catamount: all static linking
    - *So: no dynamic linking*
  - `libsysio` explicitly reimplemented the standard functions
    - so: *no weak linking,* and *no reimplementing,*
    - new `libsysio` protocol requires changing QK src (as of XT3)
      - lose PDIO portability AND system support
  - Used replacement macros (in high-level codes):
    `#define open pdio_open`
  - Requires one addition to compile/link lines (but *NOT* `src`)

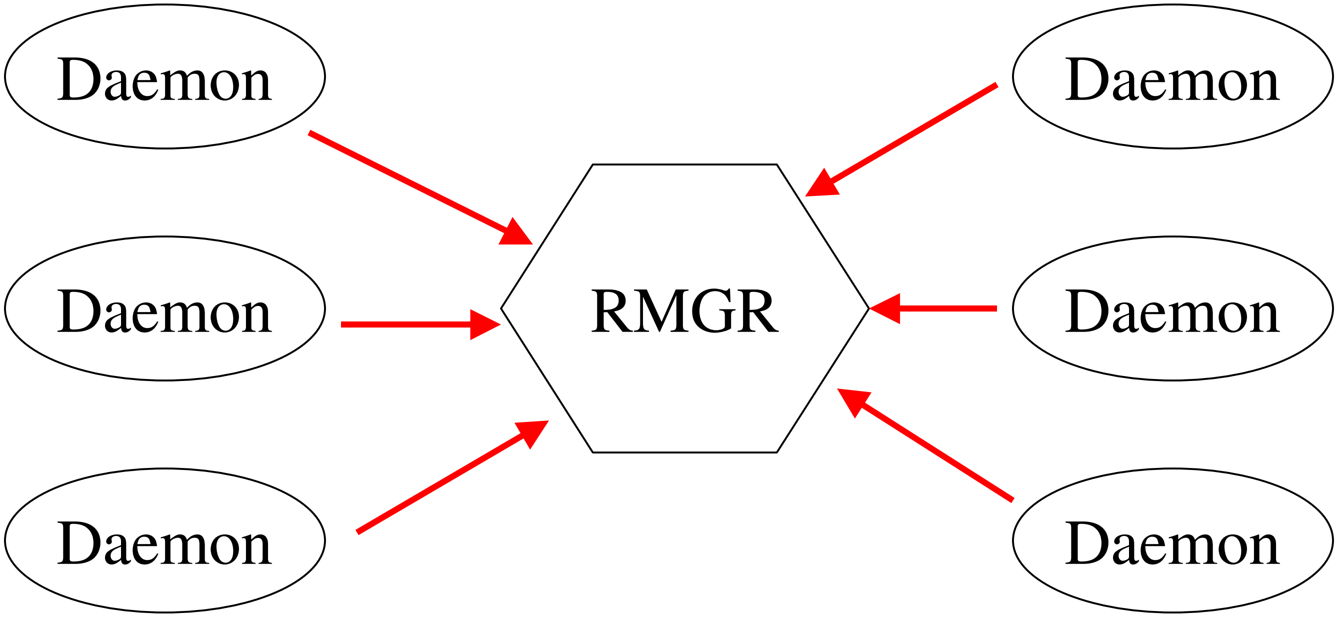
Revised Design: 2-File Semantics

- Support for general file access patterns
  - Previous constraint was “one write-one file”
  - **Now**: Multiple writes, parallel/concurrent access, lseek, etc.
  - Store and check all FDs explicitly at each function invocation
    - Open an empty shadow file in `/__incore` to get a valid, unique FD
    - Pass invocations on PDIO FDs to the Client Library
  - Each operation results in either metadata modification or bulk data (with MD) transmission

Revised Design: 3-Resource Mgmt.

- For the Prototype:
  - Required explicit launch of daemon(s) by the user
    - Required user to have access to “routing” I/O node
    - No limits on user largess
    - Many sites: disallowed for security reasons
  - Remember: It was all about function…
- In the Revised Design:
  - Single “Mother Hen” daemon launched at boot
    - MH dynamically launches/manages a pool of routing daemons
  - Daemons register presence (and function?) with RMGR
    - Presence is determined by persistent TCP socket connection
  - RMGR dynamically allocates daemons to jobs (agents) upon request

Resource Management Details



Connections

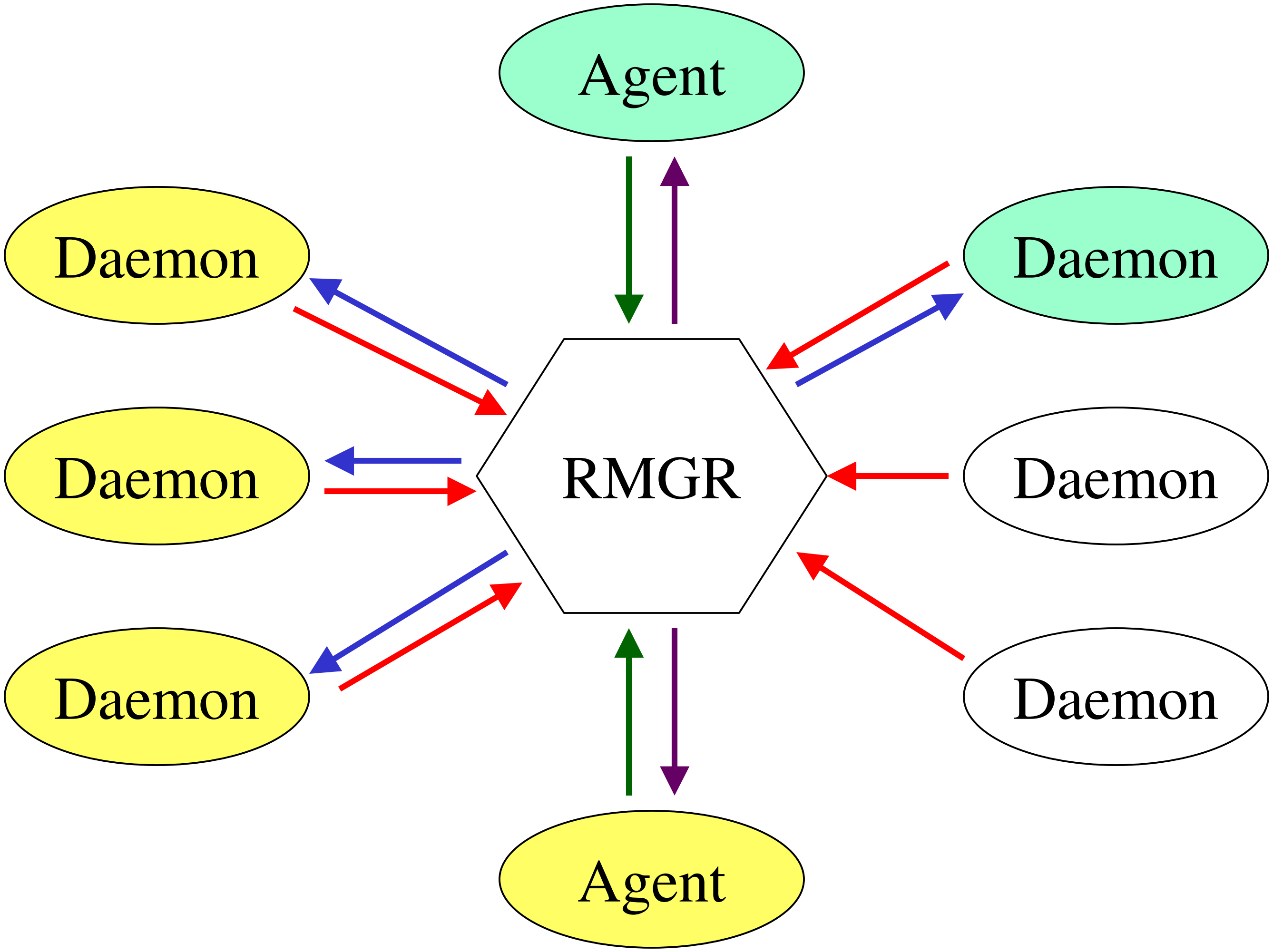
Persistent TCP sockets

All processes on SIO nodes

Legend

1. SVC Available
2. Request SVC
3. Allocate SVC
4. Grant REQ
5. Terminate

Resource Management Details



Connections

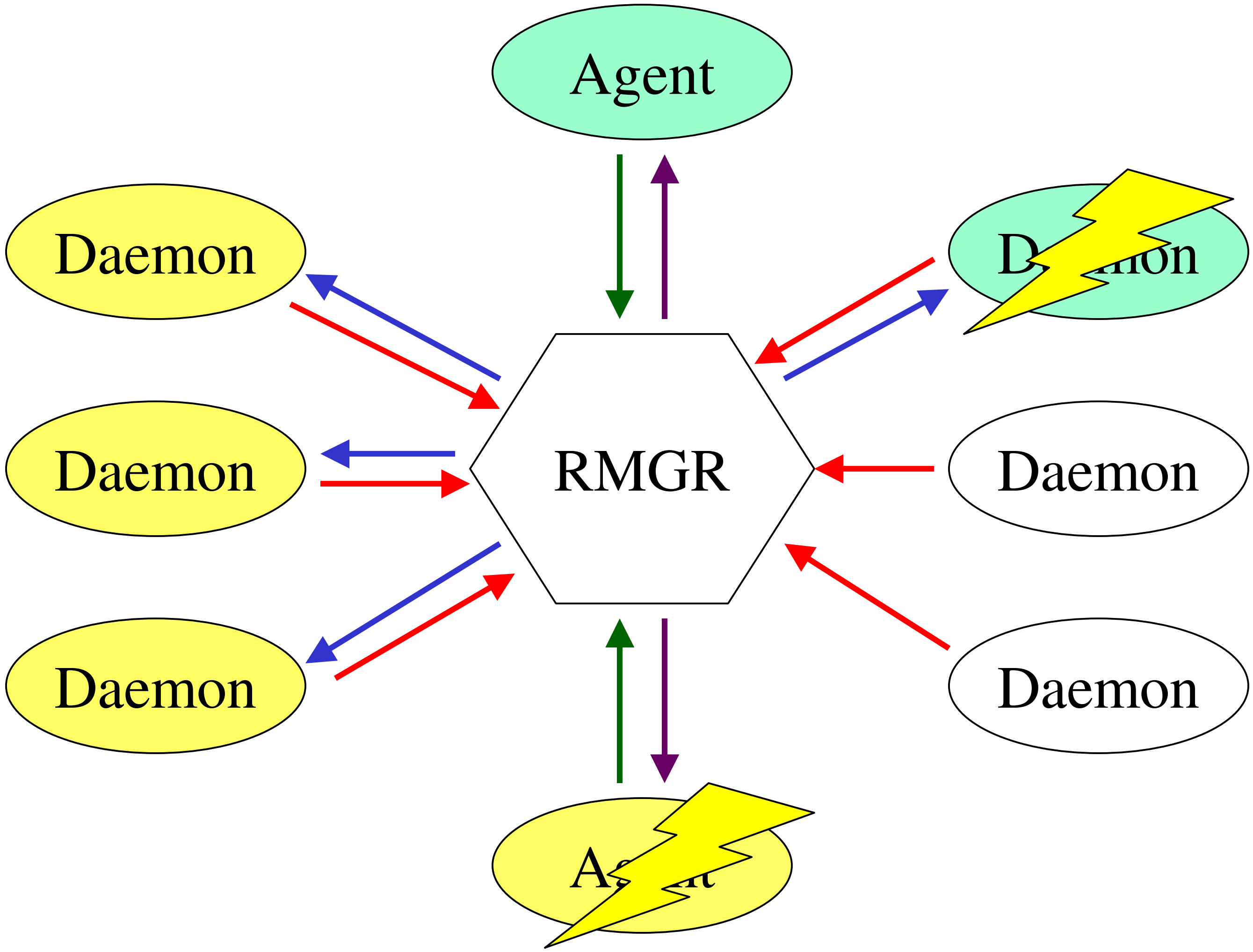
Persistent TCP sockets

All processes on SIO nodes

Legend

1. SVC Available
2. Request SVC
3. Allocate SVC
4. Grant REQ
5. Terminate

Resource Management Details



Connections

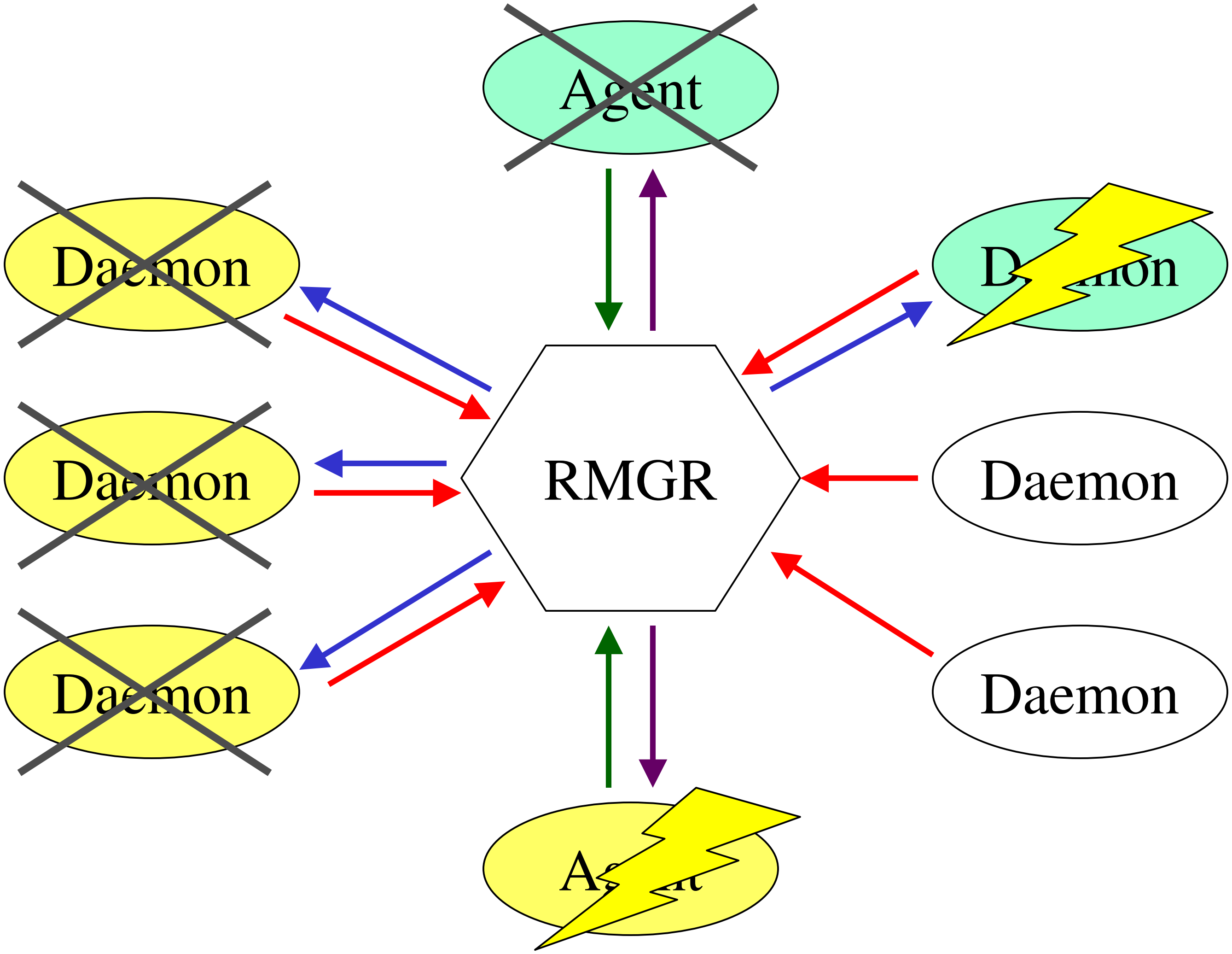
Persistent TCP sockets

All processes on SIO nodes

Legend

1. SVC Available
2. Request SVC
3. Allocate SVC
4. Grant REQ
5. Terminate

Resource Management Details



Connections

Persistent TCP sockets

All processes on SIO nodes

Legend

1. SVC Available
2. Request SVC
3. Allocate SVC
4. Grant REQ
5. Terminate (triggers notice)

Revised Design: Others…

- Performance and robustness enhancements
  - – Prototype had some tuned timeouts
  - – **Now**: Using Portals NULL-ACK (truncated match)
- Dynamically configurable
  - – **Now**: User/administrator config param.
    - By config files or CMD line (in case of agent)
- Enhanced back-channel communication
  - – **Now**: All remote error messages back to the client, not just for performance…

Current Development Status

- Completed:
  - – Client Library
  - – Resource Management
  - – Revised Portals IPC
- In-Progress:
  - – Portals Client-Daemon protocol
  - – TCP Daemon-Recvr protocol

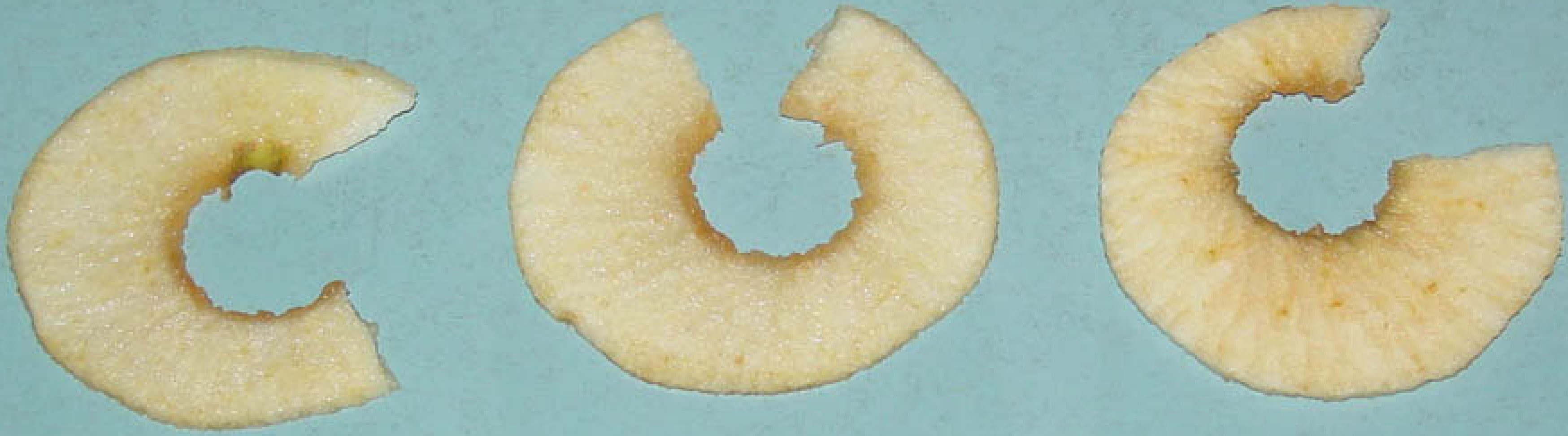
Future Status (?)

- Finish the revised (beta) release (Q3’06)
- Add `read()`
  - – PSC Strategy: Consider our (current) users
    - Do what is necessary, quickly, and no more
- Consider replacing WAN-TCP with other more secure protocols (*e.g*. Globus XIO)
  - – Currently: Grid application w/o “grid” software
  - – Could complicate aggregation/parallelization…
- Consider using PDIO on other platforms?
  - – Any compute node running Portals
  - – NALs exist for: Seastar, Myrinet, ELAN/Linux, IP

Questions?

Nathan Stone

<stone@psc.edu>



PSC Advanced Systems Group

http://www.psc.edu/advanced_systems/

Whitepapers for ongoing work at PSC

http://www.psc.edu/publications/tech_reports/