

The PHENIX Event Builder

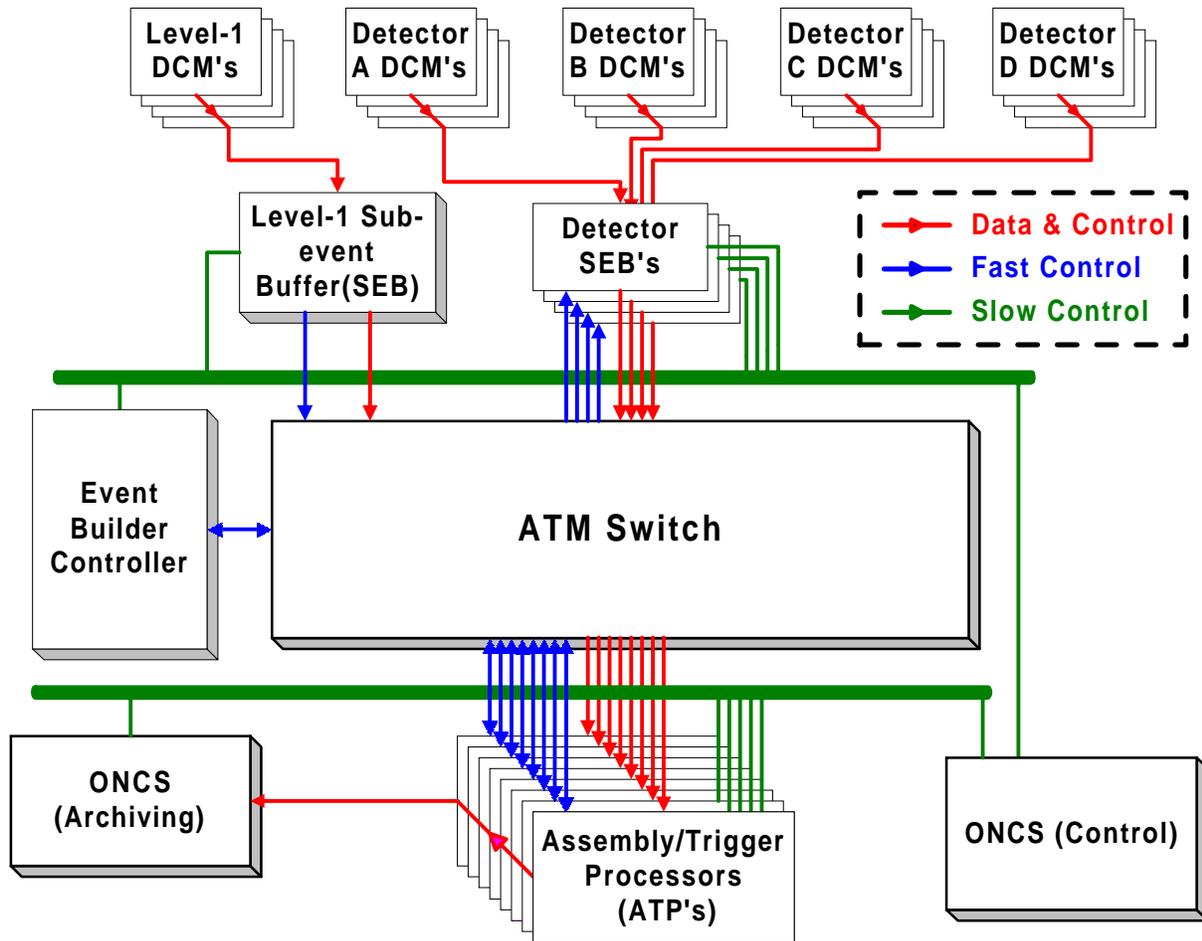
Brian A. Cole
Columbia University

PHENIX TAC Review
August 10, 1998

Outline

- 1) **Sub-system Overview**
- 2) **Performance Requirements**
- 3) **System Design**
- 4) **Implementation:**
 - a) **Data flow**
 - b) **Component architecture**
 - c) **ATP/Level-2**
 - d) **Control**
 - e) **General issues**
- 5) **ATM Transfer tests**
- 6) **Status, schedule**

Event Builder - Overview



Responsibilities

- Collect data from all DCM's -- In **Sub-Event Buffers (SEB's)**
- Collect event fragments -- In **Assembly/Trigger Processors (ATP's)**
- Assemble fragments into complete events -- In **ATP's**
- Perform Level-2 trigger calculations/rejection. -- In **ATP's**
- Transmit selected events to ONCS.

Event Builder - People/Responsibilities

Institution	Name	Position	Responsibilities
Columbia	B. Cole	Faculty	DC Member, overall administration of sub-system
			Event builder design/documentation
			Core C++ class development
	P. Steinberg	Post-doc	ATM Evaluation/implementation
			Event builder design/documentation
			SEB, ATP, Controller software implementation
S. Markacs	Ph.D. student	ONCS, Pamette integration	
		Debugging/testing	
		Controller data structure implementation	
Brookhaven	S. Durrant	Staff	Debugging/testing
			Pamette I/O and control software
	S. Lin	Engineer	Pamette (DCM-SEB PCI interface) FPGA coding
			Pamette daughter-card design, construction, testing
Georgia State	X. He	Faculty	ONCS-Event builder integration
			Level-2 trigger support software

- Expect to add additional post-docs @ Columbia (fall/winter '98), Georgia State (spring '99).
- Expect 2-3 additional Ph.D. students @ Columbia starting summer 1999.

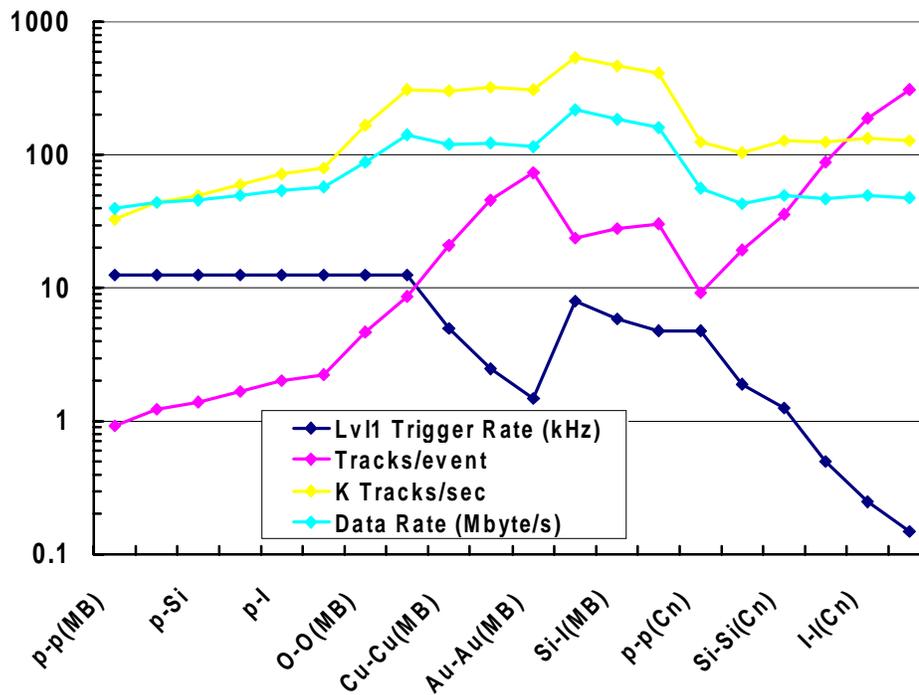
Event Builder - History

- PHENIX CDR
 - Custom cross-bar switch + DSP's.
 - High level design, no details.
- PHENIX UCDR
 - Same as above, slightly more detailed.
- PHENIX TAC ??
 - Partitioning rears its head.
 - Reality intrudes, custom solution deprecated.
- November 1996
 - Event builder re-design commences.
 - Meetings with RD-31 group ⇒ switch-based solution
- PHENIX TAC 1997
 - Complete high-level design presented.
 - ATM switch-based event builder, PC's for SEB/ATP.
 - Lack of manpower clearly evident.
- Winter 1998
 - **P. Steinberg & S. Durrant join effort.**
 - Subsequently observe tremendous progress on all fronts.

RD31/Atlas Demonstrator-C

- Our design relies heavily on results of studies by
 - CERN RD31 (Dufey *et al*)
 - Atlas Level-2 Demonstrator-C (LeDu *et al*).
- Demonstrator-C architecture very compatible with PHENIX DAQ
 - Digitization and (partial) event building after Level-1
 - Highly parallel processor-based Level-2 trigger
- We have adopted many of the features of the demonstrator-C design
 - It has been evaluated in a working system.
 - It has been shown to be scalable (at least to 64x64).
 - Design choices consistent with our requirements.
- We have communicated/collaborated with the Saclay group in Atlas over the last 1.5 years
 - We started by “porting” their code
 - We are now specializing it to our purposes
 - ⇒ Avoid using “custom” NIC + driver
 - ⇒ Use C++
- + there are significant differences in the details
 - We will likely end up using little of the Atlas code directly.

Trigger/Data rates @ RHIC Design



- Large dynamic range in
 - Collision rate: 25 kHz (p-p) → 100 Hz (Au+Au central)
 - Event Size: 5 kbyte/event → 350 kbyte/event
- **BUT !** Approximately constant *data* rate for all species
- PHENIX Online Performance specs
 - Baseline: **12.5 KHz Lvl-1 rate**, **500 Mbyte/s EvB bandwidth**
 - Upgrade: **25 KHz Lvl-1 rate**, **2 Gbyte/s EvB bandwidth.**

Event Builder - Day 1 Requirements

Assumptions

- “Day-1” luminosity \approx **1% of blue book**
 - Au-Au interaction rates: **min-bias 10 Hz, central 1 Hz.**
 - Expect $< 1/2$ of PHENIX channels read out
 - But also little or no zero suppression.
 \Rightarrow **Estimate initial data-rate of 5 Mbyte/s.**
- End year-1 run luminosity, \approx **10% of blue book.**
 - Au-Au interaction rates: **min-bias 100 Hz, central 10 Hz.**
 - Assume modest zero-suppression.
 \Rightarrow **~ 10 Mbyte/s data rate (20 Mbytes/s w/ x2 safety).**

Implications

- Modest year-1 performance requirements for event builder.
 - But, year-2 requirements will be aggressive (**~ 100 Mbyte/s**).
- \Rightarrow **For year-1 preparation, focus on robustness, integration.**

Event Builder Design

Primary considerations/guidelines

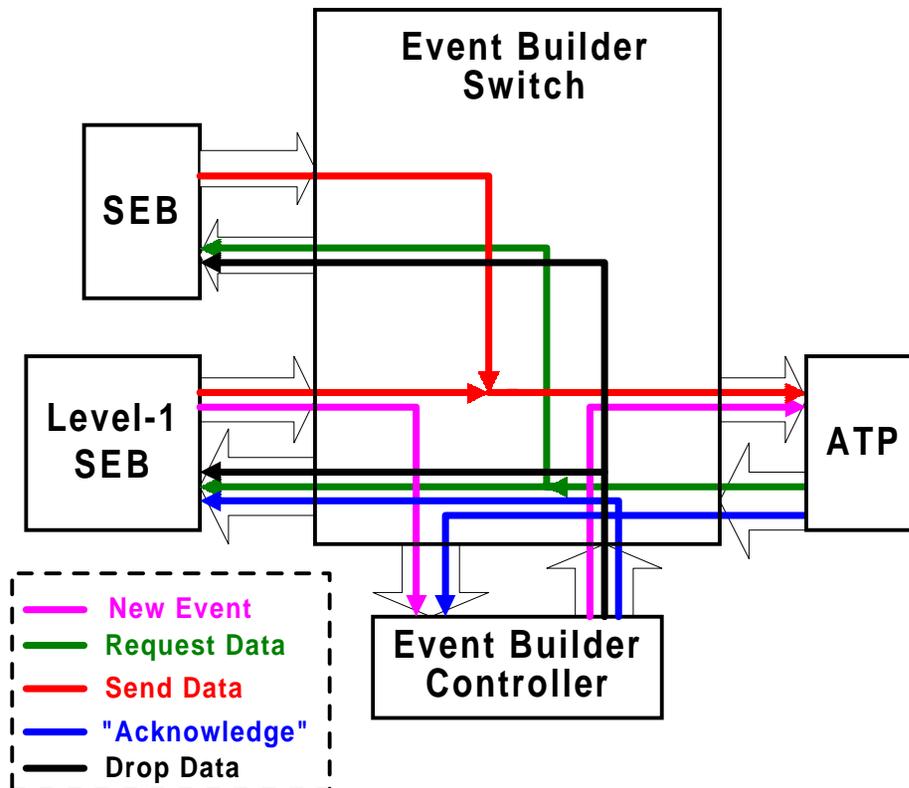
- Satisfy PHENIX's performance requirements.
- Make it as commercial as possible (manpower/cost).
- Allow easy upgrade of components.
- Make it scalable to follow luminosity/PHENIX growth.

These result in:

- Switch-based event builder (commercial, scalable).
- Over ATM (performance, commercial, scalable, upgradeable).
 - Switches, NIC's now available for PHENIX's ultimate needs.
 - Provides required flow control in hardware.
- Using PCI-based processors (commercial, upgradeable).
 - Highest performance bus with significant market share.
 - 64 bit, 66 Mhz will soon be a reality.
- Running Windows-NT (commercial, upgradeable).
 - All ATM hardware guaranteed to work on NT
 - Rational synchronization, asynchronous I/O support.
- Using Threads (performance)
- Using object-oriented code (commercial, scalable, upgradeable).
 - Control structures, I/O, memory management greatly simplified

Event Builder - Data Flow (High-level)

- Up to SEB's, DAQ is data-driven and parallel by channel.
 - Data pushes into SEB's.
 - “Hold's” propagate back to DCM



- ATP's processing is parallelized by event.

⇒ Switch to “pull” architecture

- Level-1 SEB notifies controller of new events.
- Controller allocates ATP(s).
 - ⇒ Use dynamic load leveling
 - ⇒ Events assembled by partition.
- ATP's pull data from SEB's, assemble events, perform L2 trigger calculations.
 - ⇒ Assembled events sent to ONCS via ethernet.
- Events dropped from SEB under direction of controller.
- All operations are “pipelined”.

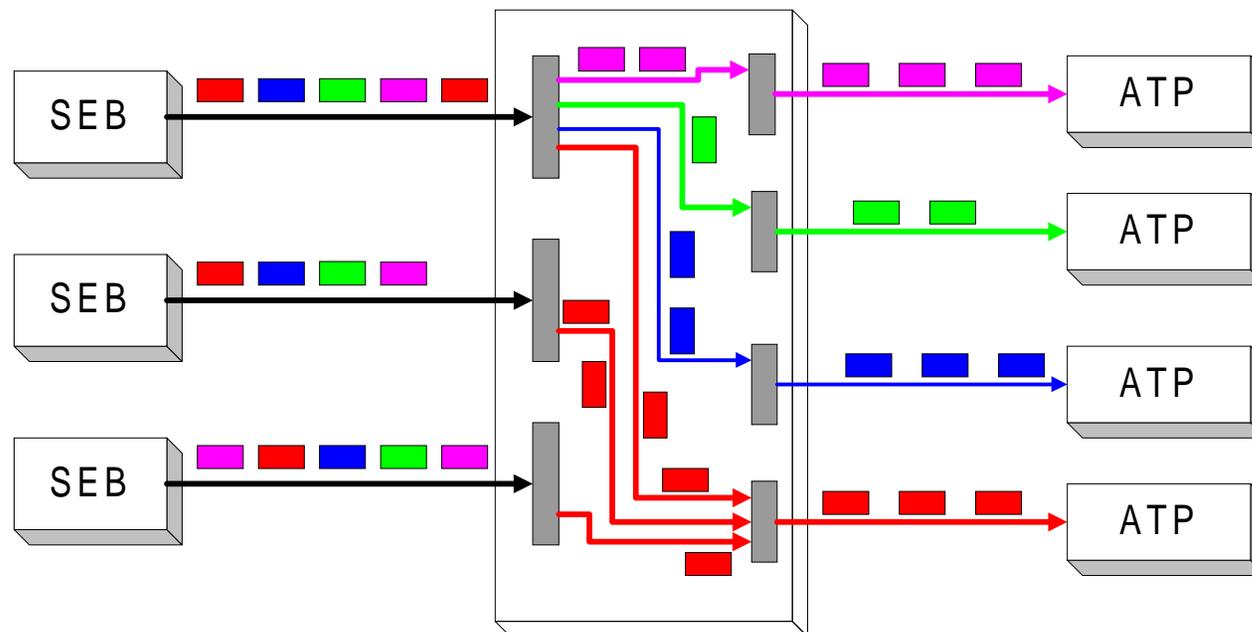
Event-Builder - Data Flow (low-level)

Virtual Circuits (VC's)

- ATM is connection-oriented.
- VC connects source & destination.
- We will use permanent VC's
- One PVC needed between each pair of nodes communicating.
- Multi-cast VC's will connect controller to all ATP's and SEB's.

Congestion Avoidance

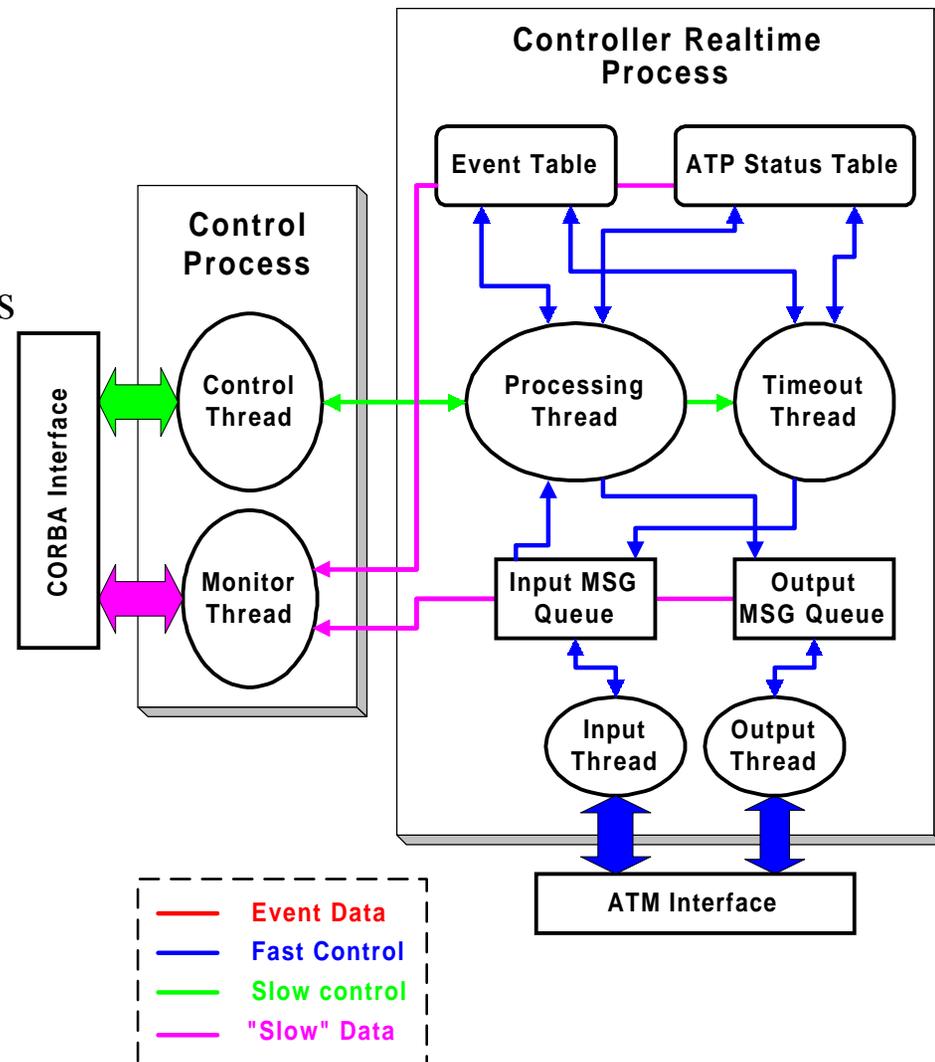
- Event collapses to single output port on switch.
 - Without flow control this overloads output port bandwidth.
- ⇒ Use VC multiplexing.
- Interleave ATM cells from multiple events.
- ⇒ “Build” multiple events simultaneously



Event Builder - Component Architecture

Common architecture

- Two processes per “component”
 - multi-threaded real-time process
 - control process w/ CORBA interface
- Inter-process communication through shared memory + signals
 - Configuration/state changes
 - monitoring (“counters” + histograms)
- In “real-time” process:
 - configuration and control performed by “main” thread
 - actual processing performed by “worker” threads.
 - Configuration/state information maintained in component (SEB, ATP, controller) objects.

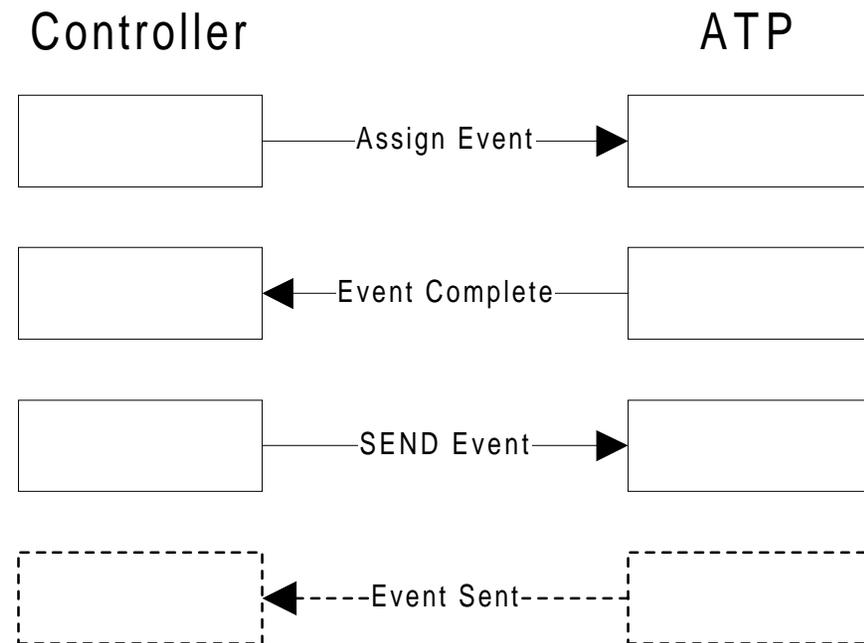


Event Builder - Control (fast)

- All fast control resides in controller.
 - Except for “extra” Level-1 SEB code to notify controller of new event.
- Controller makes all decisions regarding the fate of an event
 - Who tells SEB’s to drop event -- still undecided.
- Why all the handshaking ?

⇒ Robustness !

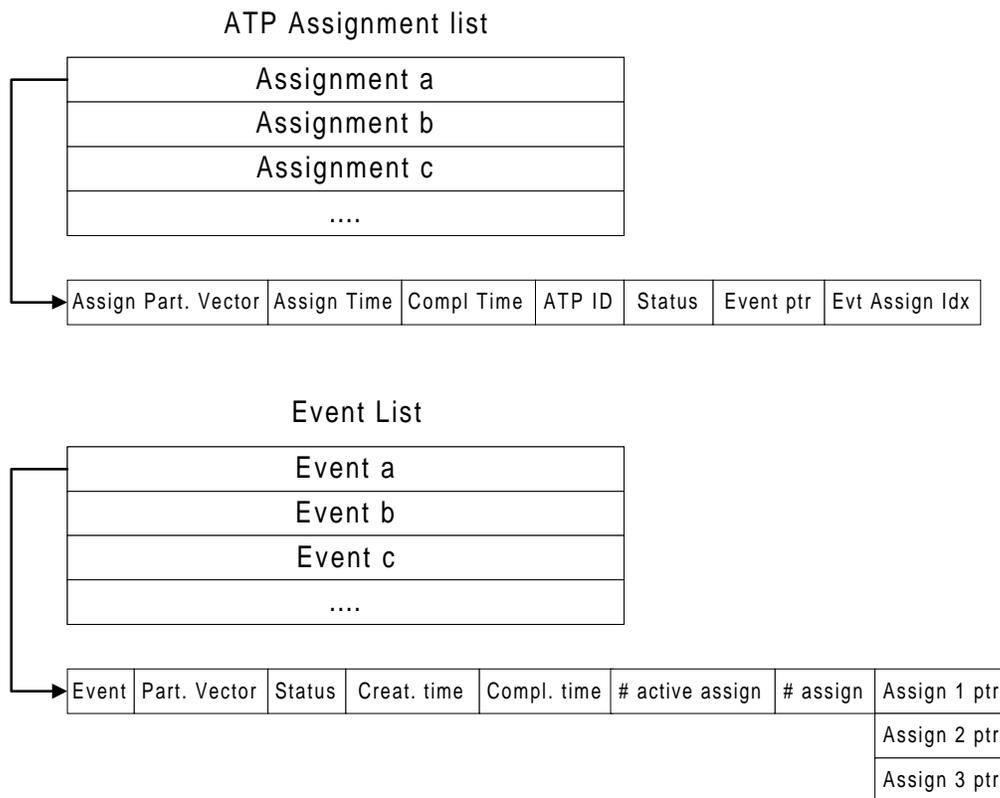
- If ATP “times out”, event can be reallocated.
- If first ATP later revives we can prevent duplication.
 - Above made possible by not dropping data in SEB’s when fetched.
 - Data only dropped in SEB when event is successfully sent to ONCS.



Event Builder Control - Fast (2)

Controller data structures & partitioning

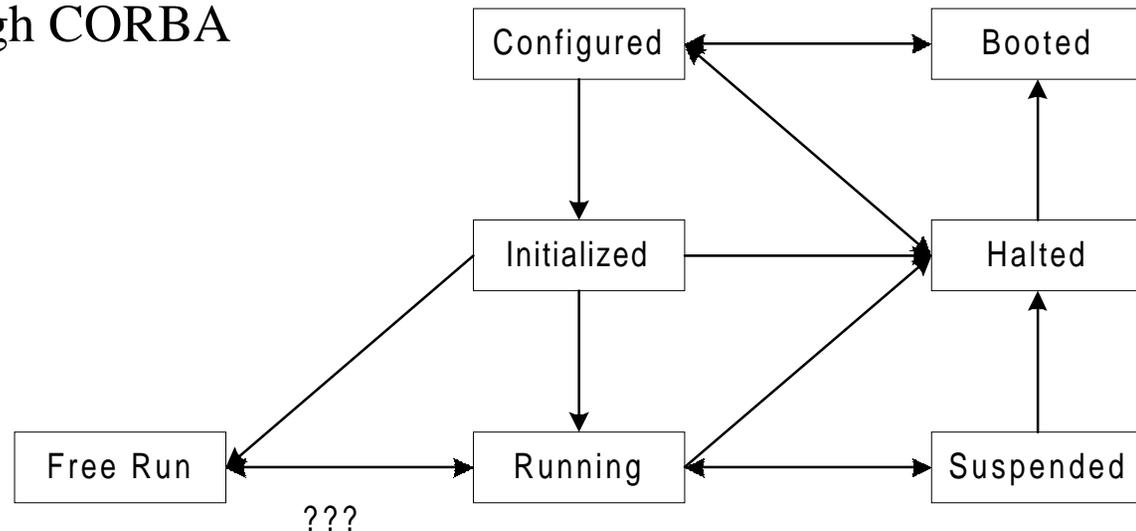
- Allow for different partitions in an “event” to be handled by separate ATP’s.
- So there may be multiple ATP **assignments** per “event” (actually crossing)
- Algorithm for “allocating” ATP’s not yet specified.



Event Builder - Control (Slow)

Component control

- “slow” control of components through control processes.
- Each component implemented as finite-state machine
 - Manipulated through CORBA
- e.g. Controller
- Mainly for
 - configuration
 - initialization
 - error recovery



Overall Event Builder control

- Controller provides all explicit control of the entire event builder (e.g. start/stop/pause)
- Mainly through handling/lack thereof of new event messages.
 - When stopping a run all events in progress will complete before controller acknowledges state change

Event Builder Control - Error handling

Frame losses

- We plan to use AAL5 transport over ATM
 - But w/ Winsock2 this decision is easily changed if desired later..
- AAL5 provides unreliable transport -- delivery not guaranteed.
 - **In principle, cell loss is possible \Rightarrow frames may be “lost”.**
- Our design explicitly accounts for this possibility
 - **BUT, error handling is under our control.**
- Where required, have explicit checking for time-outs
 - e.g. if ATP times out in returning trigger status on an [assignment](#), the controller can reassign the analysis on that (crossing, partition).
 - e.g. if data frame from SEB is dropped, the ATP will re-try
 - \Rightarrow we always require a frame even if there's no data so we can distinguish between dropped frame and empty frame.
 - \Rightarrow If frame is dropped on a given VC (i.e. from given SEB) then the re-tries will be performed serially -- slows down ATP but prevents congestion at output or ATP buffer overflow.

Event Builder Control - Error Handling (2)

Component Failures

- Failures in SEB, Controller are necessarily “fatal”
- Failure in ATP can, in principle be detected and “handled”
 - ⇒ ATP is simply “removed” from controller tables
 - ⇒ This can be done in controller if an ATP continues to time out.
- What is “failure” ? -- **termination of process/thread.**
- How to detect failures ?
 - Control process failure will be detected through lack of monitoring data.
 - Control processes will detect termination of real-time processes.
 - Real-time process main thread will detect termination of worker threads.
 - Worker threads will terminate on **fatal** exceptions.

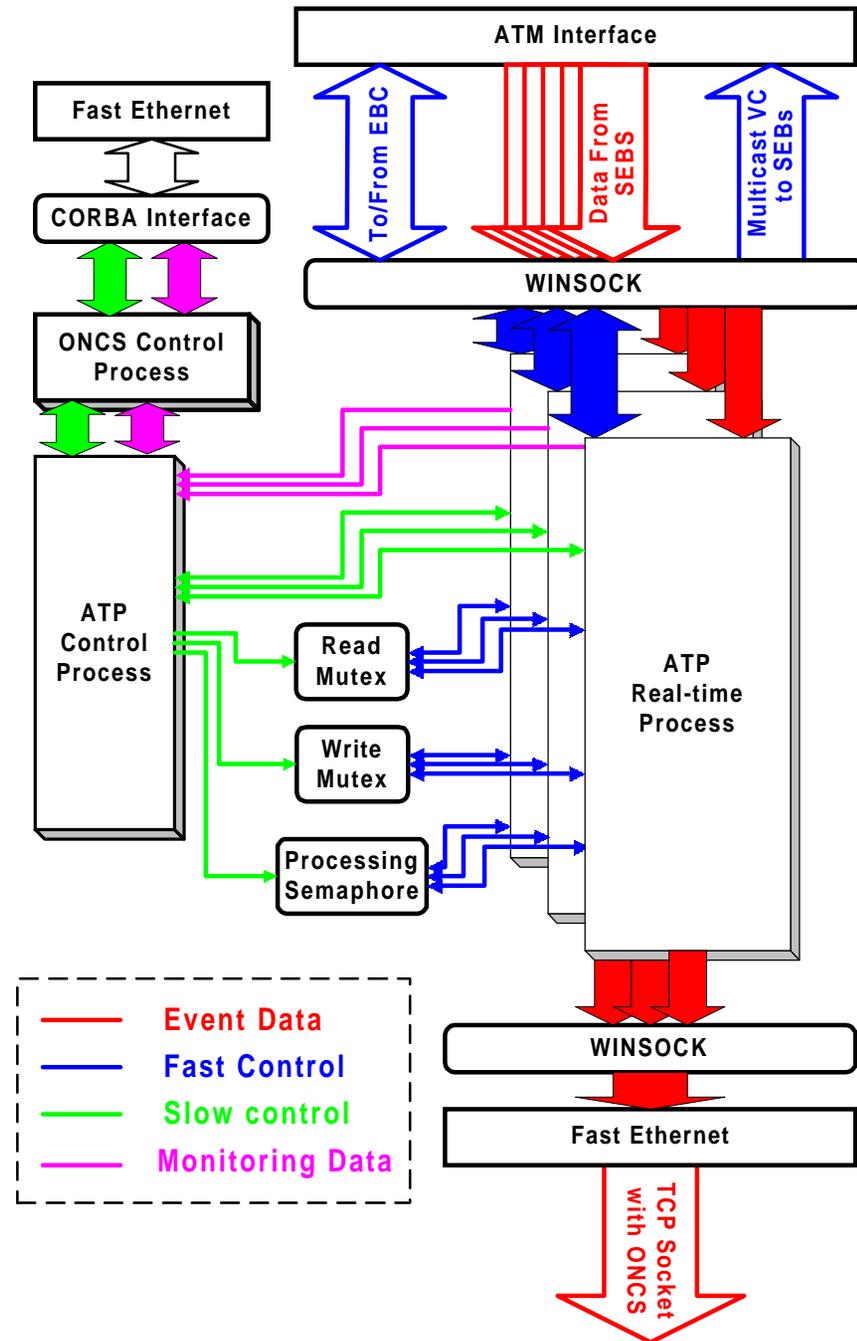
Errors

- **Serious** exceptions will be reported to control thread through asynchronous messages in shared memory.
- **Normal** exceptions are handled using C++ structured exception handling.
- **Data errors** will be recorded in data stream, counted & in some instances reported to ONCS.

Event Builder - ATP

Design Considerations

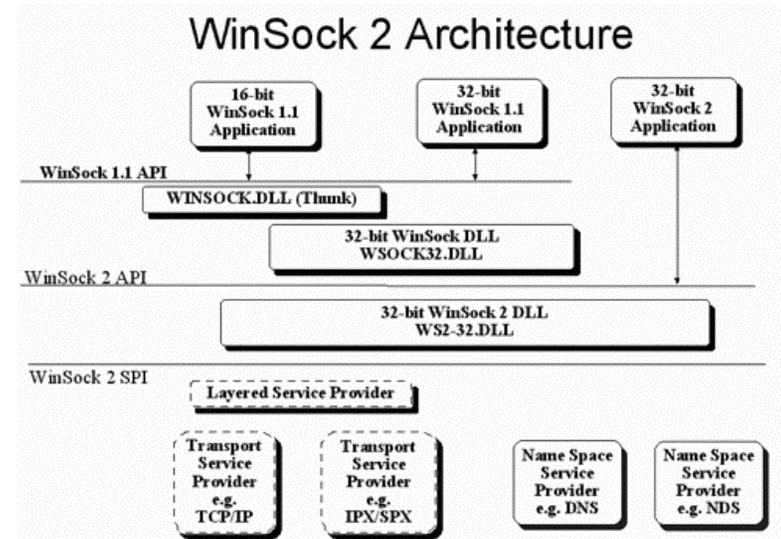
- Provide event parallelism
 - Process/write events while “reading” others from SEB
 - Trivially take advantage of multi-cpu platform.
- But, handle events in separate **processes**.
 - Prohibits possible cross-event corruption from L2 algorithms.
- Use shared Winsock2 sockets for
 - reading from SEB
 - writing to ONCS
- With synchronization for exclusive use of read/write sockets.
- Use semaphore to control # of events running L2 simultaneously.
- Algorithm “timeouts” handled within event process.



ATM on Windows-NT

Winsock2

- Socket-based API extended from BSD sockets.
- Transport independent API for network I/O
 - Provides flexibility in ATM implementation.
- High-level interface to
 - ATM quality-of-service control
 - ATM “raw” AAL1/ 5 transports
 - ATM signalling
- Explicit support for
 - scatter-gather transfers
 - asynchronous I/O
 - “pre-declaration” of receive buffers
- Uniform interface to NIC’s from different vendors.
 - Facilitates upgrades/migration.

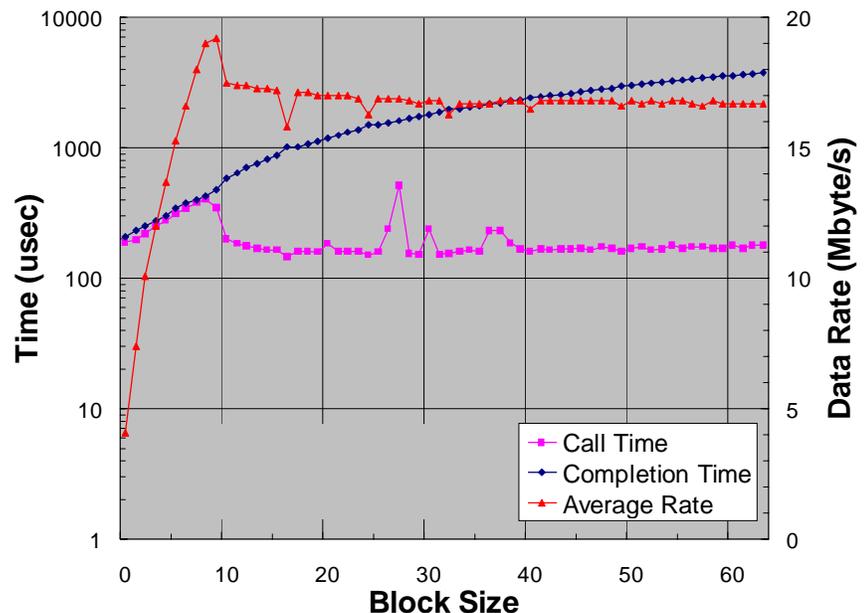


The primary concern:

- does Winsock provided the required performance ?
- Since it implements a layered interface it may have substantial overheads

Performance tests of Winsock/NT

- Measure send/receive rates on two different NIC's (Fore, Efficient)
- Using 200 Mhz Pentium PC's.
- Use Saclay timer routines
 - Use CPU clock counter
 - checked using NIC monitors
- Use asynchronous I/O.
- Measure vs block size:
 - I/O call time
 - time to completion routine
 - average time to execute loop
- Focus on sending
 - ~200 μ s overhead per transfer
 - Can sustain line rate w/ >7Kbyte blocks
 - Identical results for 2 NICs



Winsock Performance (2)

Good News

- Software can saturate and sustain OC-3 line speed.
- Basically works “out of the box”
 - No fussing with PCI issues (yet)
- Results are consistent for 2 NICs+Winsock SPI's

Bad News

- Overhead of 170 μ s to every transfer.
- Asynch. I/O shows odd NT behavior for small blocks.
- Results are consistent for 2 NICs+Winsock SPI's

Future

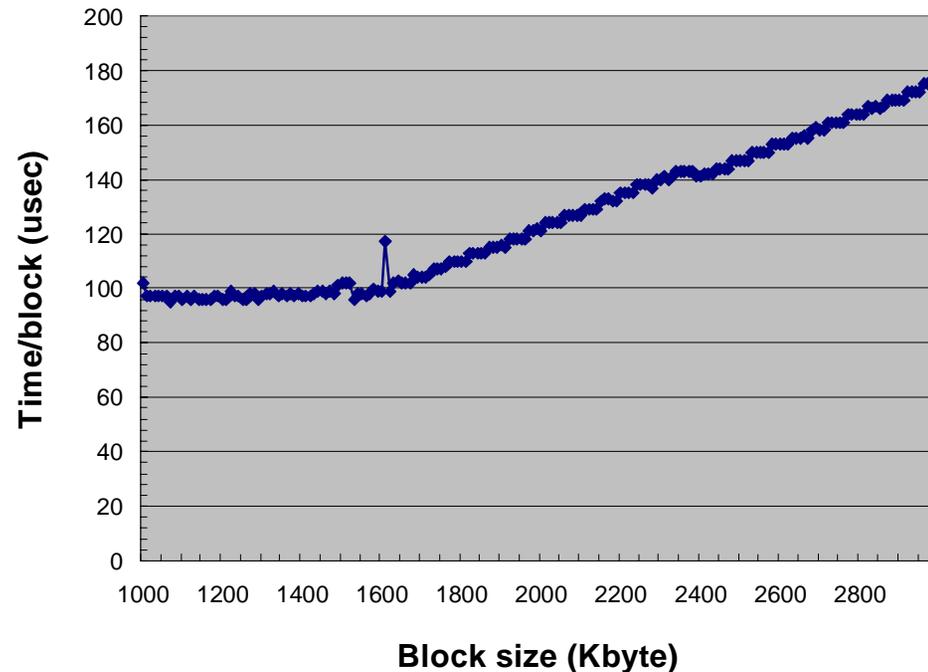
- Microsoft recognizes overheads in Winsock
 - ATM support moved into NT kernel in NT5.0
 - Supposedly the observed overheads are much reduced
- But ! We can't test this claim because NT5 beta delayed

Our plans

- Performance is sufficient for now thru year-2 run.
- Forge ahead but re-evaluate after NT5 measurements.
- Study Winsock on Alpha- smaller OS overheads ?
- Re-measure on 400 Mhz Pentium-II machines.

New Winsock Measurements

- Using DELL 400 Mhz P2
- 100 Mhz memory bus
- Cache still runs at 1/2 memory bus speed.
 - XION cache runs 100 Mhz
 - ⇒ Expect even lower overheads per operation
- BUT **very costly now!!**
 - XION uses Custom Intel in-house SRAM



Observations / conclusions

- Memory+CPU can outpace NIC on per-byte basis
 - Not true for Micron 200 Mhz P1 (see previous slides).
- Non-optimized code, optimization on 200 Mhz gives 1/3 improvement in performance.
- Winsock/NT overheads are less of an issue.

Implementation Issues - Hardware

ATM

- We have and are using a FORE ASX-1000 switch.
 - 64 ports max (currently equipped with 4 - soon 8)
 - Max bandwidth of 10 Gbit/s
 - Max usable data bandwidth 5 Gbit/s
 - ⇒ gets us to 500 Mbyte/s (Baseline)
- We have tested different NIC's
 - We seem to be insensitive to differences
 - For now will use FORE NIC's, cost + convenience.

PC's

- We plan to continue using Pentium PC's for SEB/ATP
 - but Alpha prices are coming down, may continue.
- Most likely will use rack-mount chassis w/ motherboards
 - passive backplanes + SBC's investigated, but w/ recent dramatic technology changes are less viable.
- Will use Alpha for controller
- Likely dual-cpu systems for ATP's after year-1.

Implementation Issues - Software

Operating System

- We have chosen NT as our operating system for now.
- We will maintain choice at least through year-1.
 - Essential for stable development, decision making.
- Try not to tie ourselves too heavily to NT
 - No MFC, Active-X, Windows, ...
 - Use Winsock2 socket interface for ATM
- But, will take advantage of
 - “Cleaner” NT synchronization mechanisms
 - Intrinsic asynchronous I/O support
- OS-specific features are wrapped up in objects
 - Localizes OS-specific code
 - Re-write classes for different OS if ever required.

Future alternatives

- Linux most likely
 - Not thrilled w/ POSIX synchronization, AI/O.
- Lynx-OS also possible but expensive.

Implementation Issues- Software (2)

Use of C++

- Many people express concerns regarding speed of C++
 - Clearly not a concern for control/monitoring
- But we're also using C++ for "real-time" structures
 - Such concerns are irrelevant for year-1, probably 2.
 - They may also be simply wrong (or un-informed)
 - ⇒ see preliminary benchmarks by P. Steinberg
- Our approach:
 - Be reasonable, but use C++ features (e.g. polymorphism) where appropriate.
 - Use templates where possible.
 - Reduce on-the-fly creation/destruction of objects.
- Libraries/STL
 - Currently we are using SGI STL implementation
 - ⇒ even on NT -- Visual C++ templates less stable
 - We are not currently using any commercial libraries.
 - ⇒ We are considering using RougeWave STL & threads++

Progress/Future Schedule

Current Status

- We have nearly complete implementation of SEB code.
 - ⇒ Also yields much infrastructure needed for controller/ATP.
 - ⇒ **This includes handling of PRDF, frame assembly.**
 - Integration w/ ONCS not quite complete.
 - 1st Chain test of DCM, Pamette, SEB in 1-2 weeks.
 - Incorporation of ATM messaging + Asynch I/O RSN.
- Currently developing Controller code
 - Have implementation of main control structures.
 - Incorporating messaging with input/output queues.
- Goal is to have standalone 2x2 event builder in Sept.
 - Limited monitoring
 - Error handling will not be complete.
 - No real trigger algorithm support.
 - Extremely simple Controller “policy” algorithms.
- From there:
 - Flesh out exception handling & error reporting everywhere.
 - Implement real controller policy algorithms (e.g. load-leveling).
 - Flesh out trigger algorithm support software.
 - Implement object “pools” to avoid **new**, **delete**

Schedule - Engineering Run

Hardware configuration

- 2 SEB's w/ single Pamette (400 Mhz Pentium)
 - likely will be first 2 rack-mount systems
- 2 ATP's (400 Mhz pentium) - in hand
- 1 Controller (533 Mhz Alpha) - in hand
- ASX-1000 w/ 8 ports - in hand
- Terminal switch - Yet to be purchased (Raritan)

Software

- “Version 1” of Event Builder code
 - Fully functional SEB interface.
 - All components functional w/ full control interfaces
 - At least minimal monitoring (counters, buffer depths)
 - Functional connection to ONCS data logging.
- Goal is to have this in November 1998.
 - Even if we “miss” we'll be ready by engineering run.