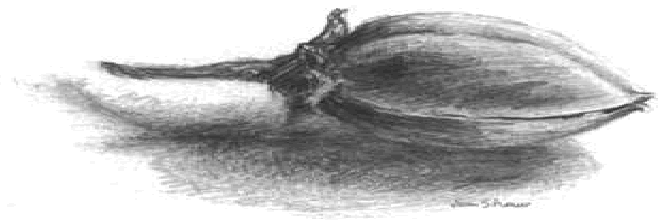
# Application of Data Mining to Physics Summary Data



Richard Partridge
Brown University / SLAC

SLAC Atlas Forum
February 27, 2008

# Motivation for Data Mining

- Original motivation derived from DØ Run I Experience
- Physics analyses typically made use of a small subset of the event data
  - Momentum vectors for jets, leptons, photons
  - Missing ET
  - Results of identification algorithms for electrons, muons, etc.
- Creating ntuples with the information needed for physics analysis was a resource intensive process
  - Required reading through large data samples to pick usable subset of events for further analysis
  - Jet energy correction and particle ID algorithms needed to be performed on each event before selection, requiring considerable CPU resources
- Difficult for non-experts to keep track of the latest algorithms, corrections, etc
  - Inhibits innovation!!

# Motivations II and IIIs

- A second source of motivation was that the problems encountered in Run I were expected to be worse for Run II
  - » Run I had ~60M events recorded by DØ during 1992 – 1996
  - » DØ currently writes ~300K events per hour, will probably end up with something like 8B events in Run II
  - » CPUs are much faster and disk drives have greatly increased capacity, but the speed of accessing data has seen only modest improvements
  - » There was no indication that making data easily available to non-experts was a goal in design of the DØ software effort
- A third source of motivation was trying to figure out an optimal strategy for SUSY discovery in Run II
  - » Large parameter space made optimizing analyses difficult
  - » Top discovery experience was that combining channels was critical
  - » Idea was to sample SUSY parameter space and automate search for signal

# Origin of POD

- What emerged from these motivations was the idea of storing physics summary data in a “Physics Object Database” (POD)
- I was told this was perhaps the least sexy name one could come up with…
- The idea was to store calibrated, corrected physics summary data in a commercial database
- Also store the results of the particle identification algorithms
- Utilize database queries to select the desired events in a physics analysis
- In database lingo, event selection is “Data Mining”

# Why use a Database?

- Designed to store, retrieve, update, and manage complex data samples
- Large number of data types
  - » Bits, integers, floats, characters, binary objects, etc.
- Many ways of organizing data
  - » Physics object, event, file, stream, run, etc.
- Architecture allows fast / flexible access to data
  - » Avoid reading/unpacking entire event to look at 1 bit
  - » Particle ID tables are small, making it possible to quickly eliminate events not having the desired set of physics objects
- Databases query engines produce highly optimized execution plans tailored to the specific query
  - » Some real surprises showed up when we looked at these execution plans

# More Reasons to use a Database

- Provides a repository for latest calibrations, corrections, algorithms
- Data, columns, tables, etc. can be added, updated, or deleted without recreating the database
  - » Example: new calibrations/algorithms can be added to the database and compared to the old ones
- Computationally efficient
  - » Local processing, minimal network IO
  - » Multiple processors can work on single task
  - » Sophisticated data caching algorithms to minimize disk access

# Why use a Relational Database?

- POD concept was developed when Object Oriented Databases were the hot ticket
  - » We got hammered for using an old-fashioned relational database…
  - » …but sometimes the old fashioned approach works best
- Physics objects typically have a fixed set of attributes used for event selection and analysis (ET, eta, phi, quality, etc.)
  - » Good mapping to tables in a relational database
- Independence of tables aids loading, updating database
  - » Data can be efficiently "bulk loaded", independently filling database tables as needed
  - » Need to provide a "primary key" that associates data from the same event
- Many strong database vendors with quite capable products serving a large commercial market

# POD for DØ Run I Data

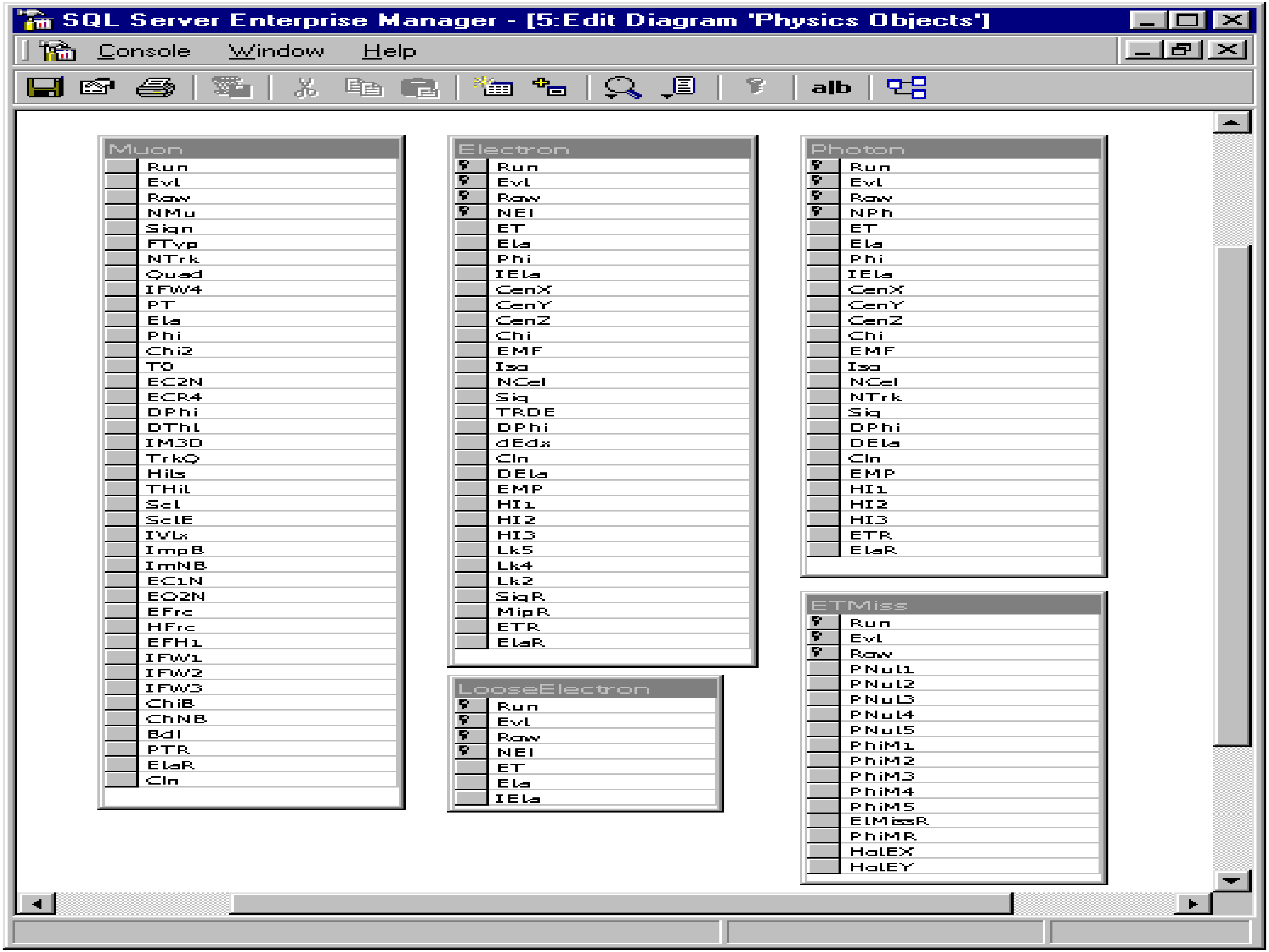
- Primary purpose was as an R&D project, not supported as a general analysis tool
  - A couple senior thesis projects were done using data from POD
  - Plan to do full implementation for Run II was shelved when RP was asked to lead what became the DØ Run IIb upgrade project
- Utilized DØ Run I data (1992 - 1996 running period)
- 62 million events loaded into the database
- Entire "All-Stream" data set loaded
  - Data set used by almost all DØ physics analyses
  - Only files with special processing or trigger conditions excluded
- Column-wise ntuple format used for importing/exporting data from "micro-DST" files that were the traditional basis for physics analyses

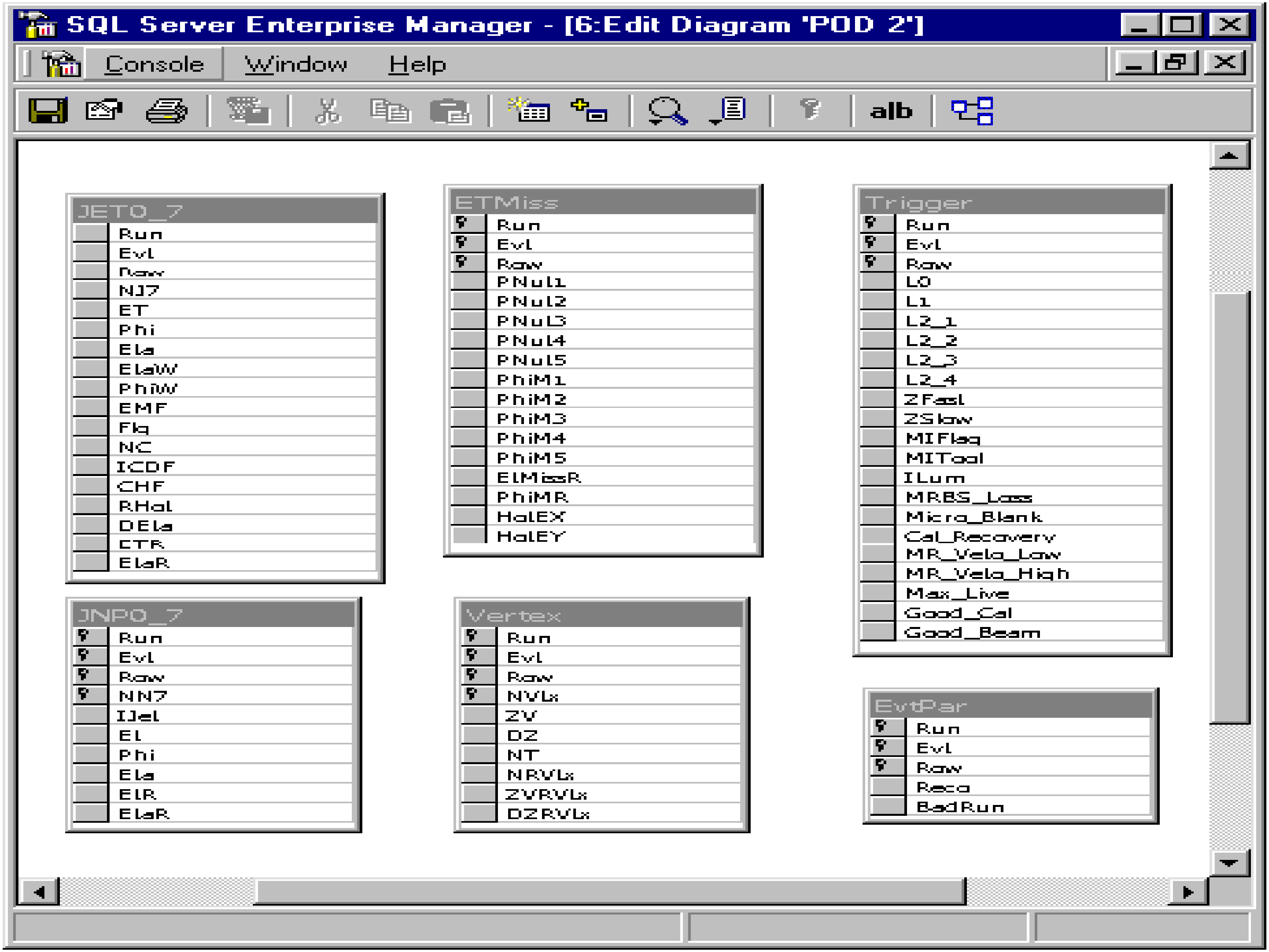
# Database Server

- Computing resources used were rather modest
  - » Dual 450 MHz Pentium II processors with 256 MB RAM
  - » 250 GB SCSI RAID disk array
  - » IO bandwidth limited to 18 MB/s by RAID controller
  - » Windows NT 4.0
- Microsoft SQL Server database software
  - » User friendly diagnostic and management tools were a big help
- Database loaded using info in ntuples created from μDSTs
  - » 8,381 All-Stream μDST files were processed
  - » 62,353,601 events (~10% are duplicates)
  - » 62.4 GB of ntuples generated
  - » 1000 CPU hours used to make ntuples

# Data Storage

- Relational database consists of “tables” of data
- Each table has “columns” and “rows”
- Columns define the data that is stored in the table
  - For example, a simple electron table might have run, event, instance, ET, eta, and phi columns
  - While these are all numeric values, there are a large number of data types available
- Each entry in the table is a row
  - An event with two electrons would have two rows, one for each electron
  - Instance column contains 1 for the first electron, 2 for the second electron
- Each table should have an index
  - An index is one or more columns that uniquely identify each row
  - POD uses the run, event, ntuple row, and instance columns for its index (ntuple row used to distinguish duplicate events in μDST files)


SQL Server Enterprise Manager - [5:Edit Diagram 'Physics Objects']
Console
Window
Help
alb
Muon
Run
Evt
Raw
NMu
Sign
FTyp
NTrk
Quad
IFW4
PT
Eta
Phi
Chi2
T0
EC2N
ECR4
DPhi
DThl
IM3D
TrkQ
Hits
THit
Sct
SctE
IVtx
ImpB
ImNB
EC1N
EO2N
EFrc
HFrc
EFH1
IFW1
IFW2
IFW3
ChiB
ChNB
Bdl
PTR
EtaR
Cln
Electron
Run
Evt
Raw
NEl
ET
Eta
Phi
IEta
CenX
CenY
CenZ
Chi
EMF
Iso
NCel
Sig
TRDE
DPhi
dEdx
Cln
DEta
EMP
HI1
HI2
HI3
Lk5
Lk4
Lk2
SigR
MipR
ETR
EtaR
LooseElectron
Run
Evt
Raw
NEl
ET
Eta
IEta
Photon
Run
Evt
Raw
NPh
ET
Eta
Phi
IEta
CenX
CenY
CenZ
Chi
EMF
Iso
NCel
NTrk
Sig
DPhi
DEta
Cln
EMP
HI1
HI2
HI3
ETR
EtaR
ETMiss
Run
Evt
Raw
PNul1
PNul2
PNul3
PNul4
PNul5
PhiM1
PhiM2
PhiM3
PhiM4
PhiM5
EtMissR
PhiMR
HalEX
HalEY



SQL Server Enterprise Manager - [6:Edit Diagram 'POD 2']
Console
Window
Help
JETO_7
Run
Evt
Raw
NJ7
ET
Phi
Eta
EtaW
PhiW
EMF
Flg
NC
ICDF
CHF
RHot
DEta
CTR
EtaR
ETMiss
Run
Evt
Raw
PNut1
PNut2
PNut3
PNut4
PNut5
PhiM1
PhiM2
PhiM3
PhiM4
PhiM5
EtMissR
PhiMR
HotEX
HotEY
Trigger
Run
Evt
Raw
L0
L1
L2_1
L2_2
L2_3
L2_4
ZFast
ZSlow
MIFlag
MITool
ILum
MRBS_Loss
Micro_Blank
Cal_Recovery
MR_Veto_Low
MR_Veto_High
Max_Live
Good_Cal
Good_Beam
JNPO_7
Run
Evt
Raw
NN7
IJet
Et
Phi
Eta
EtR
EtaR
Vertex
Run
Evt
Raw
NVtx
ZV
DZ
NT
NRVtx
ZVRVtx
DZRVtx
EvtPar
Run
Evt
Raw
Reco
BadRun

# DØ Run 1 POD

| Object | Columns | Rows | Size (GB) |
|---|---|---|---|
| Electron | 28 | 52,540,491 | 6.8 |
| Muon | 37 | 79,688,956 | 13.2 |
| Photon | 22 | 69,278,259 | 7.4 |
| Jets (3 cone sizes) | 3 x 14 | 472,626,080 | 35.7 |
| Jets with e/γ removed (3 cone sizes) | 3 x 6 | 67,003,537 | 3.1 |
| Missing $E_T$ | 14 | 62,353,601 | 4.8 |
| Vertex | 6 | 90,004,529 | 4.1 |
| Trigger | 19 | 62,353,601 | 3.5 |
| Event Parameters | 5 | 62,353,601 | 1.8 |
| Totals | 191 | 1,018,202,655 | 80.4 |

- Including indexes, Run 1 POD occupies ~100 GB
  - » 58% physics object data
  - » 18% indexes on object $E_T$
  - » 12% primary keys
  - » 12% database overhead

# Example Query

- Example: get the run and event number for events with:
  - At least one loose electron with ET > 20
  - At least one tight muon with pT > 15
- Your query might look something like:

```
select run, event from eloose, mutight where
eloose.run = mutight.run and
eloose.event = mutight.event and
eloose.ET > 20 and
muloose.pT > 15
```

- Note how the run and event columns are used to join the disassociated information in the electron and muon tables back together as an event

# POD Benchmarks

- $Z \rightarrow e^+e^-$ candidate event selection:
  - 7 seconds to identify ~6k events
- $W \rightarrow e\nu$ candidate event selection:
  - 18 seconds to identify ~86k events
- Matt Bowen's senior thesis demonstrated that even complex queries, such as reproducing the final Run I top event selections, could be efficiently executed
- Object ID tables "turbo-charge" the database queries
- Indexes on key selection variables (missing $E_T$, for example) can also make a big difference
- Event selection times compare very favorably with ~1000 CPU hours required to generate ntuples used in this study

# Database Indexes

- Indexes are special tables used by the database to provide a way of identifying and sorting database rows
- Indexes are generally stored separate from the data tables
  - » Can have a "covering index" where all columns are used in the index
- Indexes form a binary tree or "b-tree"
  - » A search that uses an index will make a series of binary decisions to find the desired row(s)
  - » A pointer in the index table is then used to locate the row in the data table
- Primary key indexes identify each row in the database and must have unique values within a given table
- Additional indexes can be created to speed searches
- The choice to use, or not use, an index is made by the database query optimizer
  - » Experience + trial and error used to identify helpful indexes

# Database Query Optimization

- Database can access data two ways:
  - » Full table scan where the entire table is read
  - » Index scan where the database uses the index to find specific records
- Full table scans are more efficient when you need to look at a significant fraction of the table
  - » Disk IO can be optimized to read database table in large blocks
- Index scans are more efficient when you only want to look up a few rows
  - » Typically requires many I/Os to traverse index and find a particular record
  - » Disk seek time limits performance
- “Clustered Indexes” were particularly useful
  - » Data table is ordered according to primary key index (run, event)
  - » This makes matching run/event numbers from multiple tables very fast when you do full table scans since both tables are already sorted by matching criteria
  - » Not clear if this feature is specific to SQL Server or not

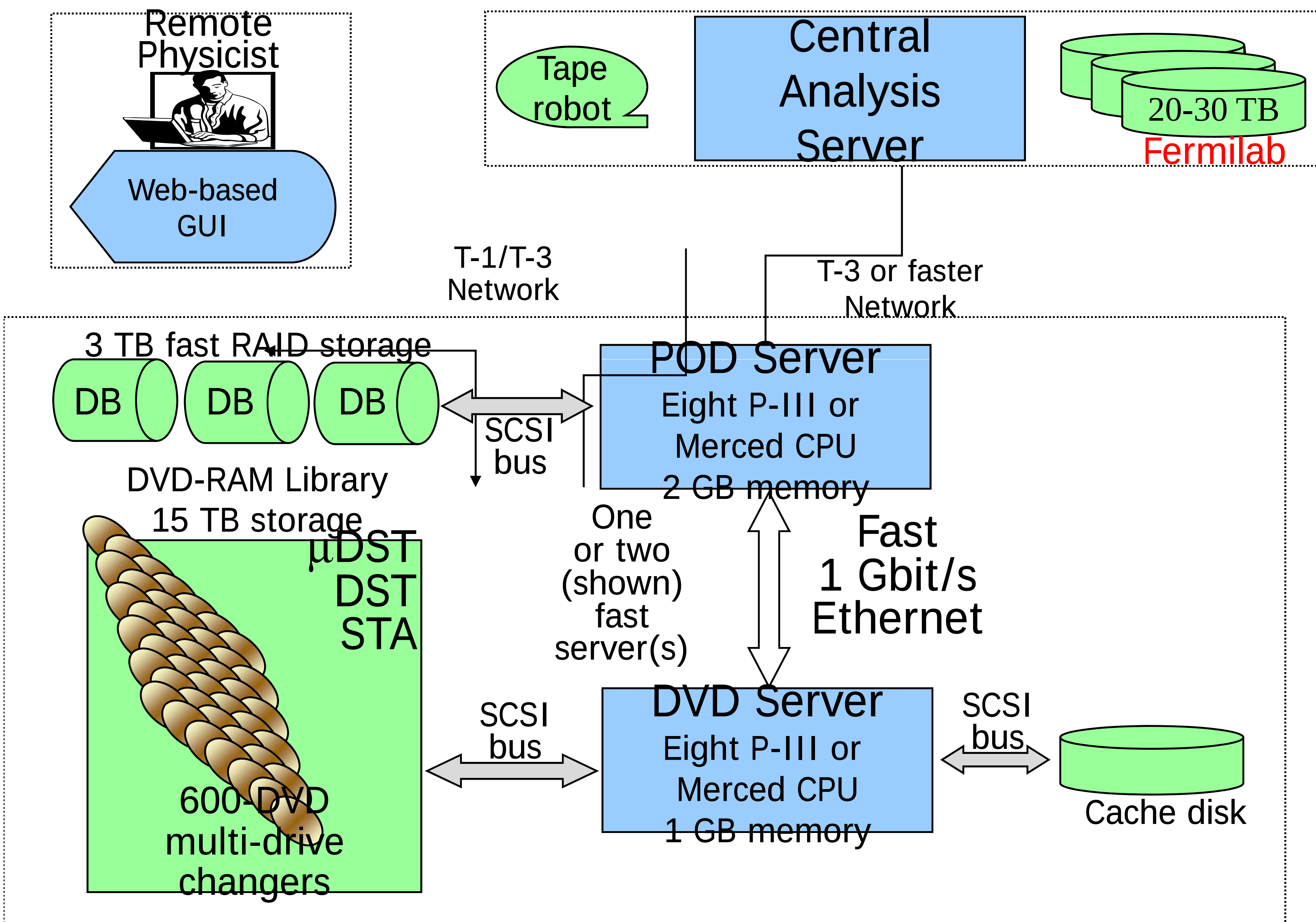
# Accessing POD

Web-based user interface developed as a summer undergrad research project

- Physicist enters event selection info using a Java GUI
  - Which physics objects are required to be present
  - Object ID algorithms to be used
  - Kinematic cuts ($E_T$, $\eta$, missing $E_T$, etc.)
  - Triggers and other global quantities could be added
- Event selection criteria are translated into the appropriate SQL query and sent to POD by a "middleware" layer
- Output options were not fully developed, but might include
  - Run, Event list
  - Root-tuple with POD variables
  - $\mu$DST of selected events

# Beyond POD: PHASER

- R&D project by Greg Landsberg and RP to significantly improve access to large data sets
- Include sufficient information in POD to perform event selection and generate ntuples for “classical” analyses that use standard object ID and calibrations
- PHysics Analysis SERver (PHASER) integrates POD event selection with the full μDST data set for “contemporary” analyses that require variables not in POD
  - » You can probably tell who thought up the PHASER acronym…
- Fast integration of new information (new object ID, updated calibrations, etc.) is essential to avoid having database become “stale”



Remote Physicist
Web-based GUI
Tape robot
Central Analysis Server
20-30 TB
Fermilab
T-1/T-3 Network
T-3 or faster Network
3 TB fast RAID storage
DB
DB
DB
SCSI bus
POD Server
Eight P-III or Merced CPU
2 GB memory
DVD-RAM Library
15 TB storage
µDST
DST
STA
600-DVD multi-drive changers
One or two (shown) fast server(s)
Fast 1 Gbit/s Ethernet
SCSI bus
DVD Server
Eight P-III or Merced CPU
1 GB memory
SCSI bus
Cache disk

# Data Mining for Atlas

- At present I am blissfully ignorant of how Atlas plans to store / access data for physics analysis
- Nevertheless, we had a very positive experience in developing the POD project and it would be interesting to see if something like this made sense for Atlas
- Scaling the database to the much larger samples Atlas will generate should be possible
  - » Databases are designed to scale well, and many large databases are in existence
  - » Large databases tables require extra care / testing – more than once we made a database change that inadvertantly sent us to database hell
- To me, what is most intriguing is using a database to provide an organized structure that makes physics quality data accessible and usable to a wide audience

# Conclusions

- Feasibility of loading "Run 1" size physics object info into a relational database has been demonstrated
- Significant improvements in event selection efficiency has been observed for W/Z benchmarks
- Expect these results will scale up to LHC datasets
  - » Multi-TB relational databases are a proven technology
  - » Will require more resources than a dual 450 MHz database server…
- Provides a way for both experts, novices, and "dinosaurs" to quickly perform event selection and extract key quantities used for physics analysis
- Database technology is also potentially useful for helping manage complex analyses and storing intermediate results