

Event Store Redesign

Daniel Wang

Yemi Adesanya

Jacek Becla



Event Store Redesign - Agenda

- ◆ **09:30am Introduction**
- ◆ **10:00am Persistent Event**
- ◆ **11:30am Tags**
- ◆ **12:00pm End of morning session**

- ◆ **01:30pm Tags - cont**
- ◆ **02:30pm Discussions/Conclusions**

Main Features of EVS

- ◆ Scalable, hierarchical collections
 - Vector, tree, bridge
- ◆ Renewing & borrowing headers
- ◆ Logical / physical split, placement
- ◆ Ability to merge events

Main Issues with EVS

◆ Size in number 1

- Navigational components (col, evt, evshdr): ~2.7 kB
- Tag ~ 0.8 kB
- AOD ~ 3.5 kB
- ESD (well optimized now) ~8 kB (?)

◆ Few others

- Requires powerful hardware (disks, servers)
- Lack of real deletion
- ...

Motivations for Redesign

- ◆ Internal BaBar Computing Review (Apr'02)
 - Recommended to reduce size size of micro
 - Timescale ~ end of 2002
- ◆ Cost (disk space, IO)

Sizes – Current System

- ◆ Col $0.08 \text{ kB} * N + 52 \text{ kB}$
- ◆ Evt $0.8 \text{ kB} * N$
- ◆ Evshdr $1.5 \text{ kB} * N$
- ◆ Tag $0.6 \text{ kB} * N$
- ◆ **Total** $2.9 * N + 52 \text{ kB}$

Event Size

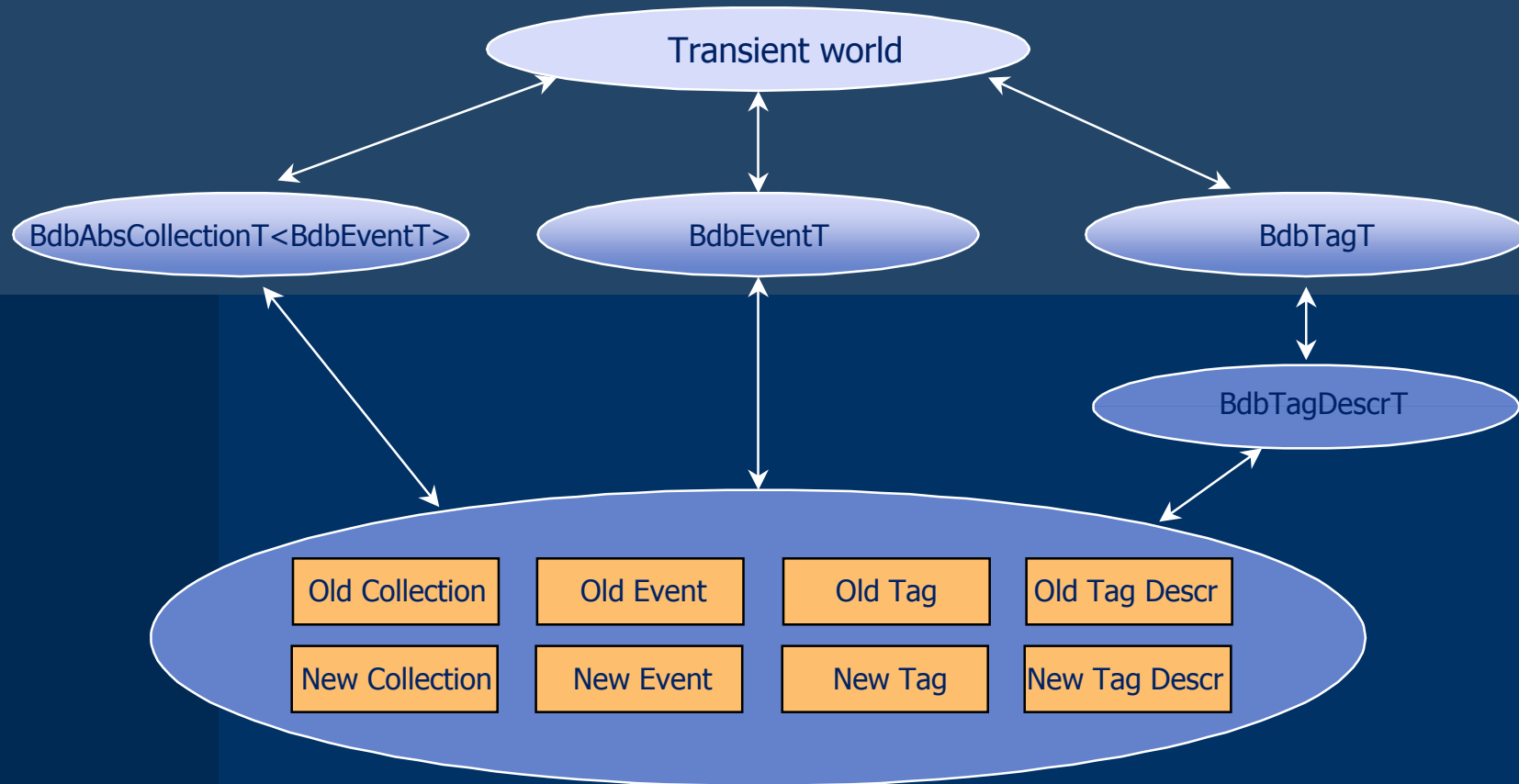
	default	ootidied
Navigational components (evshdr, tag, col, evt)	3541	3073
Nav + micro	6904	6048
Nav + mini	14539	12544
Nav + micro + mini	17902	15520

See Bdb website
for more details

Mods to Persistent Classes

- ◆ Major changes to persistent classes unavoidable
 - Tight shielding of persistent EVS
 - Collection
 - Event
 - Tag
 - Tag descriptor

Hiding Persistence to Allow Event Store Evolution



Further Benefits

- ◆ Many new features
- ◆ Addresses many mistakes and bad design decisions made
 - Cleaning up persistent classes, a lot of legacy code
 - Will reflect changes in computing model
 - E.g. no raw/rec
- ◆ Performance tuning
- ◆ “Schema evolution”
 - Well contained

Format Independence

- ◆ Most features format-independent
- ◆ Non-Objy based system would benefit from both
 - the new ideas
 - design

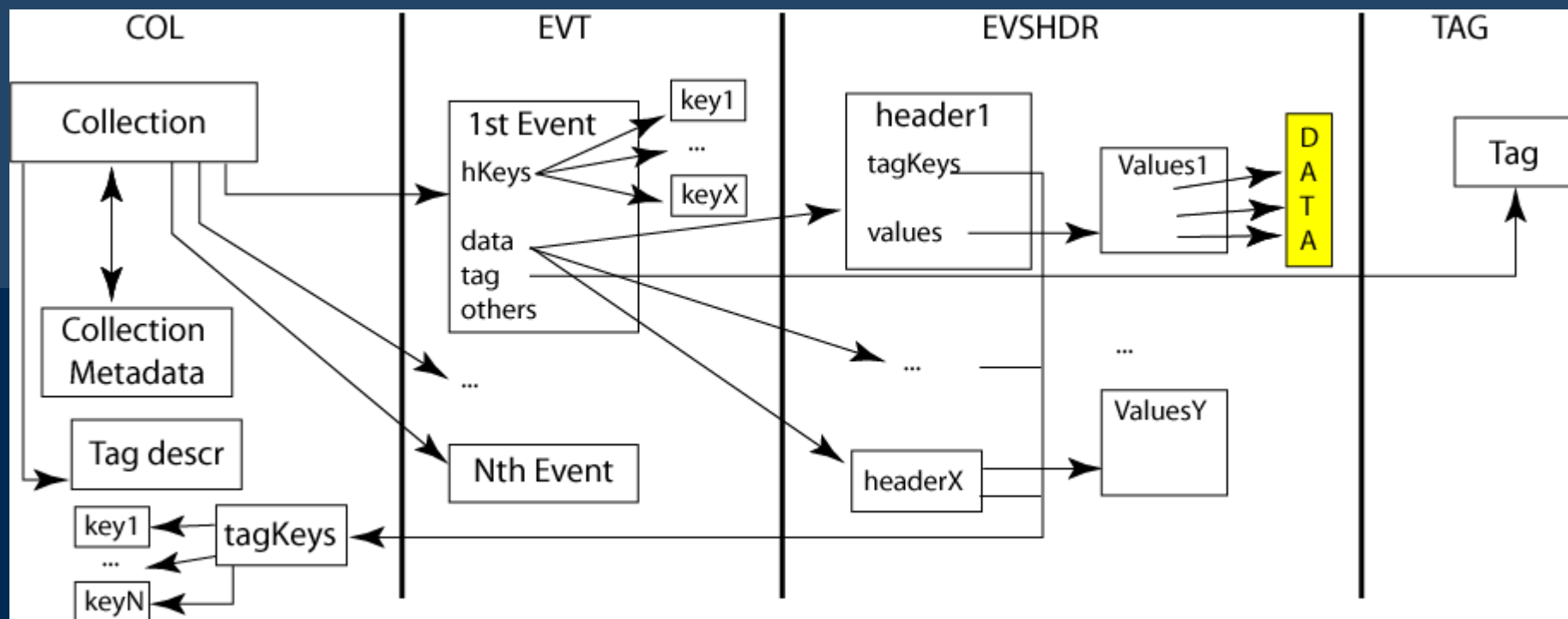
Action Plan (May '02) and Progress

- ✓ Shield persistent classes
- ✓ Estimate inefficiencies
- ✓ Design new Event Store
- ☞ Implement new classes
- ◆ Add backwards compatibility
- ◆ Test and deploy

Manpower Involved

- ◆ May – Sep
 - Jacek+Yemi ~ 1 FTE-month each month
- ◆ Oct - ~Feb
 - Daniel (100%), Yemi (80%), Jacek (25%)
- ◆ Assigned
 - Event, headers: Daniel
 - Tags, collections: Yemi

Current EVS Design



Techniques Used

- ◆ Reduce # small persistent classes
- ◆ Schema that well matches data
- ◆ Reduce duplication of same data
- ◆ Prefer short refs (4 bytes) to refs (8 bytes) when applicable
- ◆ Remove unused (obsolete) attributes
- ◆ Pack many strings into one

Migration Process

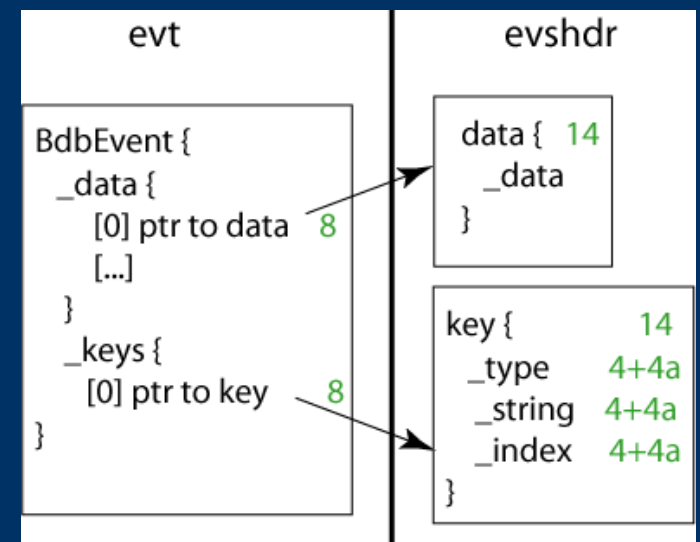
- ◆ Force all accesses to go through BdbEventT
 - Make EventT the translation layer for different types of persistent events
- ◆ Limit knowledge of persistent event implementation:
 - Only EventFactory, EventT(?) will have knowledge—
 - BdbEvent_002 does not inherit from anyone
 - Ease future migration by hiding implementation.

Shapeshifting

- ◆ Try to eliminate small persistent objects.
 - Overhead is fixed at 14 bytes
 - May lose more because of alignment
 - And, need a link to it from its container
 - Not worth it for small objects
- ◆ Try to eliminate small VArrays and Vstrings
 - Overhead is similar to persistent objects.

Event Headers

- ◆ Allow Event Store to handle arbitrary chunks of event data transparently
- ◆ Are accessed through an event
- ◆ Provide indirection between event and event data
- ◆ Reside in a different database



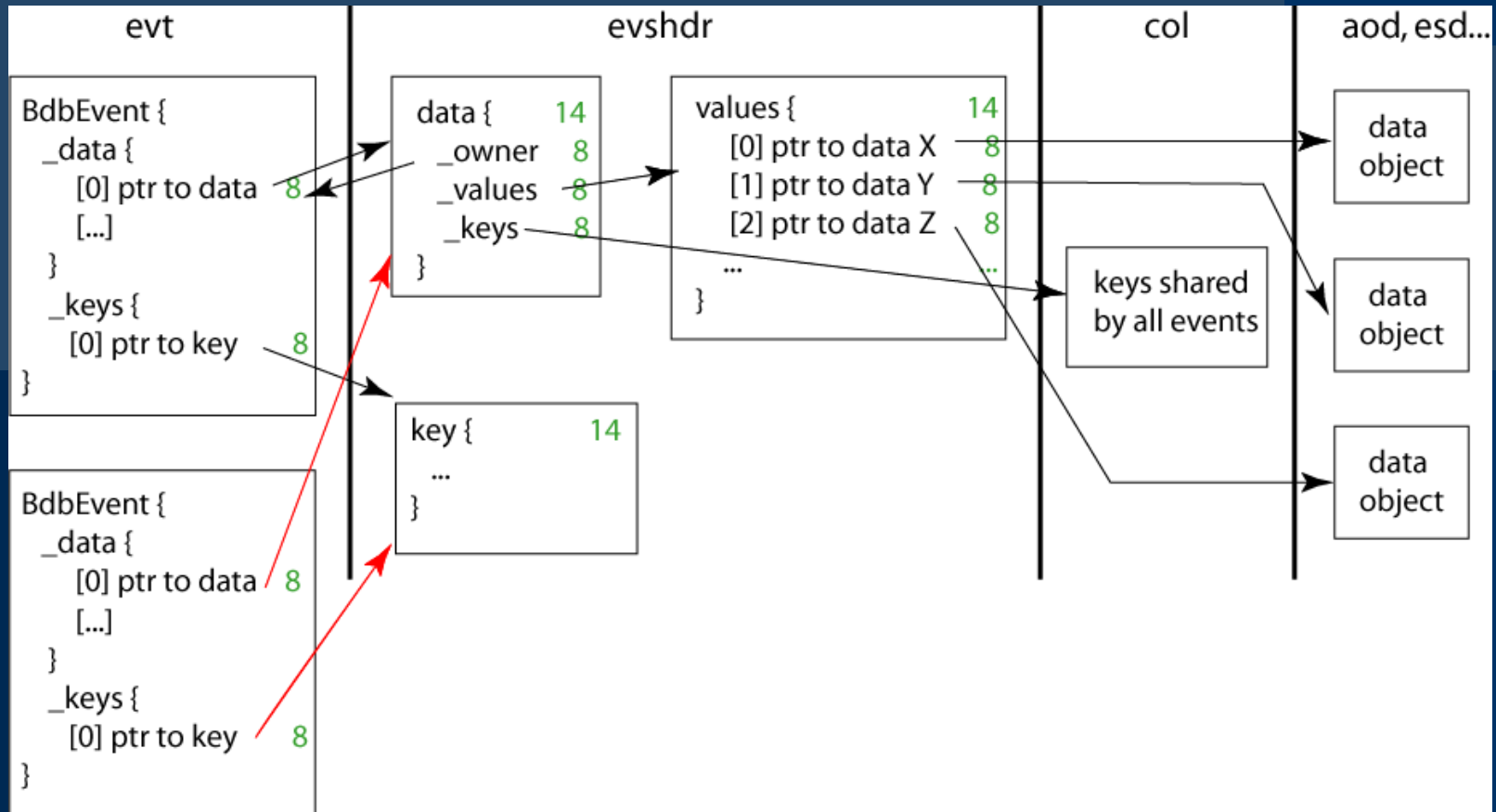
Shapeshifting step 1:

- ◆ Push all headers into the Event object
 - Eliminates objy overhead of 14 bytes per header
 - Put pointers to data directly in event
 - Retains original flexibility
 - Replaces two persistent objects per header, with one varray per event + one varray in common object

Shapeshifting step 2

- ◆ Store hdr keys in a single varray:
 - Maintains near-original read speed without obj overhead per key name.
 - Resizing slow, but rare(?)
 - Put this new varray in the common object

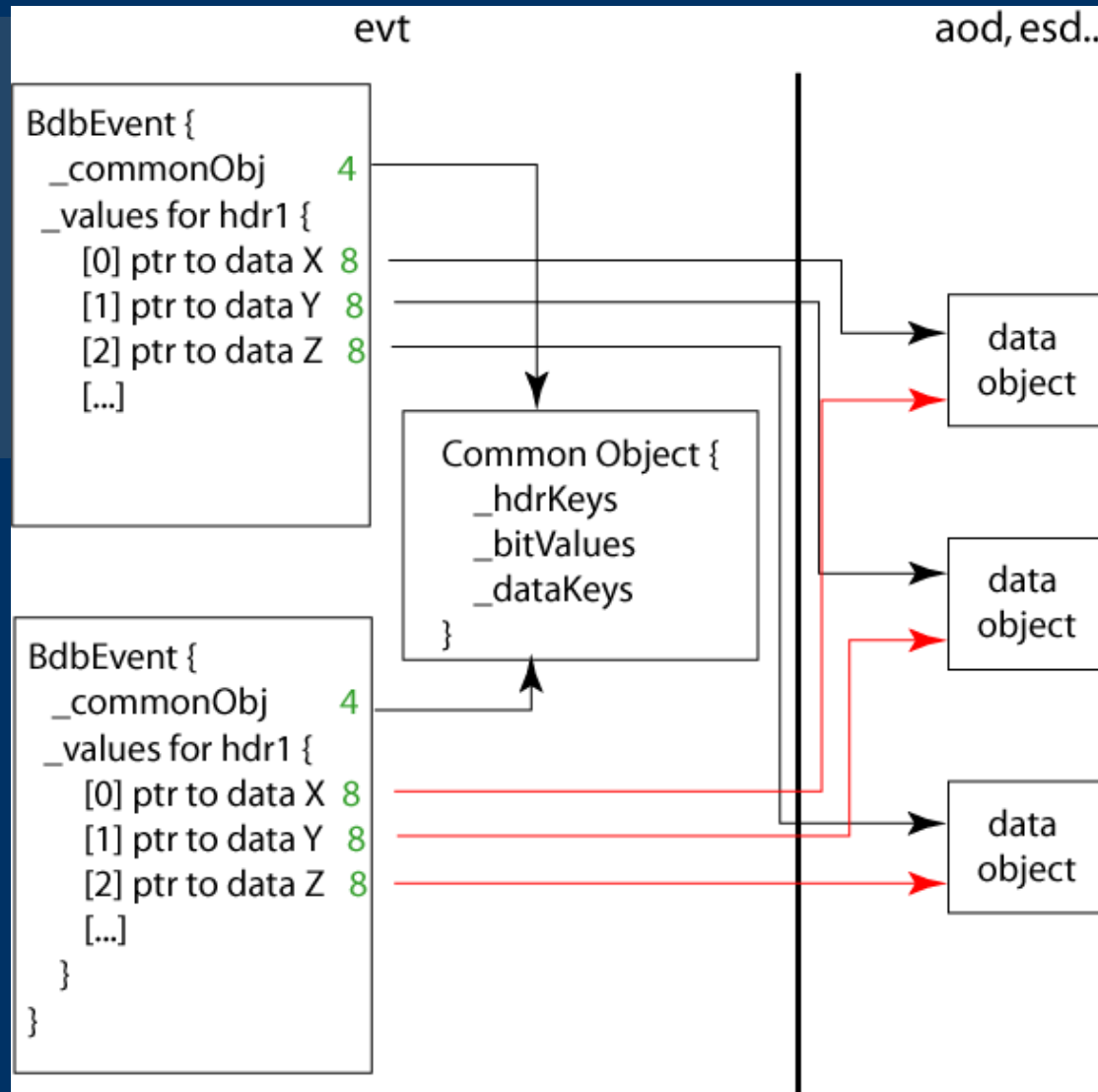
Borrowing (Current)



Borrowing

- ◆ Instead of borrowing headers, borrow the data directly
- ◆ Faster, and more flexible
- ◆ valuebits: can mark data as borrowed or owned → possible to implement “delete”
- ◆ Also: can avoid duplicating data ptrs for borrowed data at the cost of system complexity (use link to orig event)

Borrowing (Proposed)



Side Issue: Original Event

- ◆ An event stores a link to its “original” event
- ◆ Handy when an event is borrowed/skimmed
- ◆ Currently links to oldest ancestor, not immediate parent
- ◆ Is this intended?

Store Only References to Renewed Data...?

- ◆ Requires altering meaning of “original”
- ◆ Eliminate redundant references to borrowed data → reference through original event
- ◆ Is this worth it? Potential for slow performance traversing multiple layers of events
- ◆ Make this an option?

Deletion / Recovering Space

- ◆ Easier – header's owner can now be determined
- ◆ Can now reclaim space from data pointed to by an event
- ◆ But... still likely to have dangling pointers

Shall We Reorganize Headers?

- ◆ Current: rec, trk, bta, svt, ifr, drc, dch
- ◆ Better?
 - mini, micro?
- ◆ Is it worth?

Do We Need BdbEventID?

```
Class BdbEventID {  
    BdbTime _time;  
};
```

- ◆ Nsec always 0

[Jacek] It should not be forgotten that in addition to BdbEidContainer, BdbEvent > has "BdbEventID _eventID"

[Gregory] The BdbEventID is obsolete and, to the best of my knowledge, of no use whatsoever. If it is being used somewhere, that is almost certainly an error and I would like to know about it.

- ◆ Does not seem to be used – will be removed

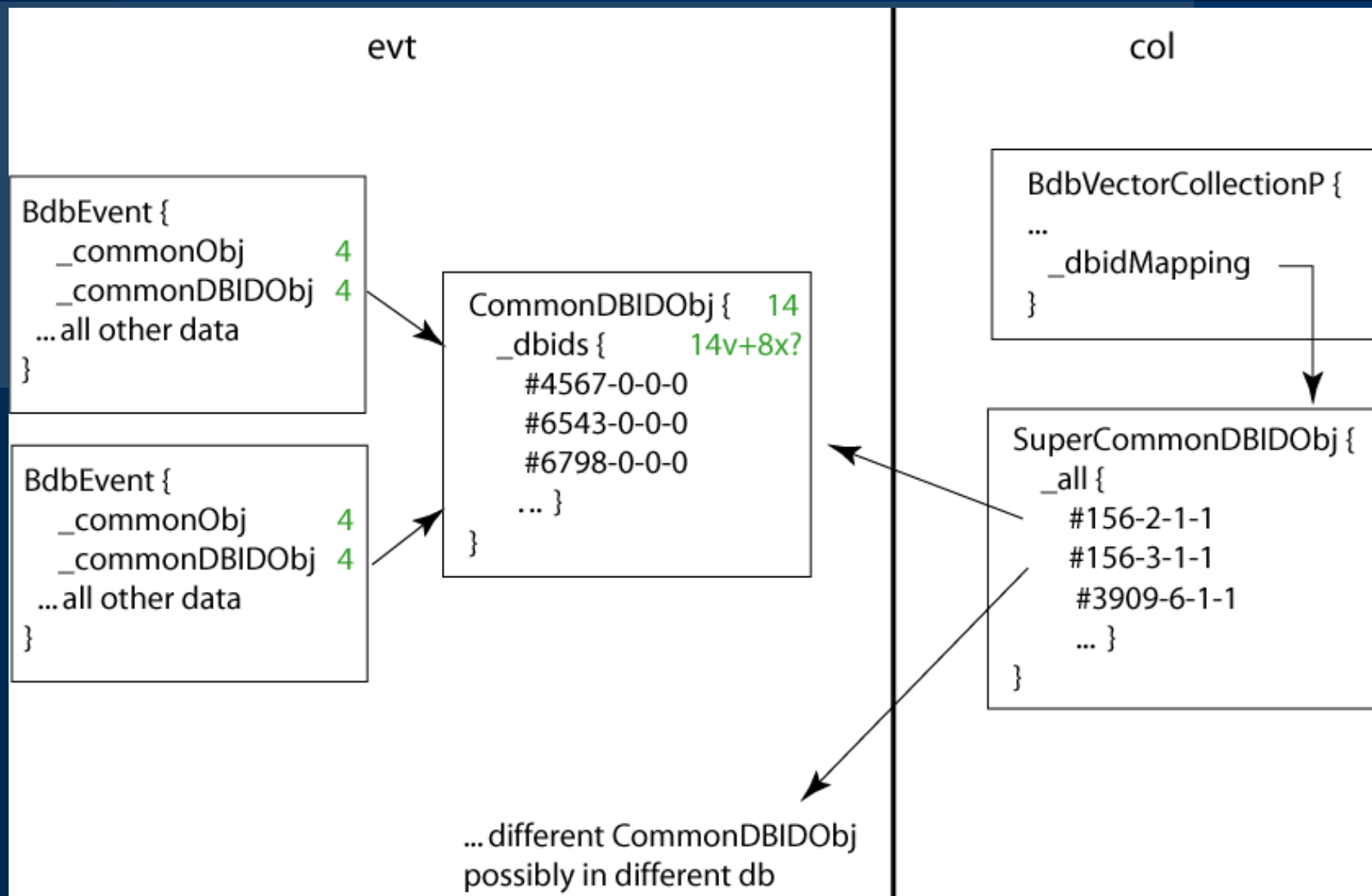
Common Objects

- ◆ Store fields common to multiple events
- ◆ Worst case: original read-speed
- ◆ Can have several per collection
- ◆ For all fields that vary less than once every few events

Common Object Contents

- ◆ Data keys
- ◆ Header keys and value bits
- ◆ StateID list
- ◆ Part of eidCont
 - `_eventPlatform`, `_eventPartitionMask`,
`_condKeyPlatform`, `_condKeyPartitionMask`,
`_configKey`, `_run`
- ◆ eidList
- ◆ Link to tag descriptor (in same container)

Collection – DBID Mapping



Scalable Collections

- ◆ Use paged varrays
 - Brought to memory in chunks (one page at a time)
 - Technology not available in '99
 - We use similar techniques in CDB
- ◆ Implement after 'bulk' work on EVS migration done

Some Implementation Details

- ◆ Avoiding varray resize
 - Causes fragmentation
- ◆ Matching events with common objects
 - Hashing?

Persistent Tag

Type	No elements	Overhead [bytes]
d_Float	64	$14+64*4$
d_Double	0	4
d_ULong	44	$14+44*4$
d_Short	0	4
d_Boolean	0	4
d_Char	64 (on average)	$14+64$

Total: 550 bytes

What is a Tag?

- ◆ A group of values that summarize an event
- ◆ Intended to support fast event iteration/selection
- ◆ Should be relatively small compared to event size

BaBar Tag Interface

◆ Analysis

- Tag values are copied into the transient event (AbsEvent)
- Updated values may later be output

◆ ‘Fast’ Filtering

- Tag based event-level selection
- No caching overhead

TagTransient interface

- ◆ Included in shared AbsEvent object
- ◆ Created at the Event loading stage
- ◆ Tag values stored using <key,value> map structures
- ◆ Type specific 'get' and 'set' methods
- ◆ Option of copying entire persistent tag during loading and selecting specific values to skip
- ◆ Modules can define default tag values
- ◆ TagTransient may be 'locked'

TagAttribute<T> interface

- ◆ Direct access to a single persistent tag value
- ◆ Instanciated using the type of the tag value
- ◆ No caching
- ◆ Templated assignment operators
- ◆ Not contained in AbsEvent

Persistent Event Tag

- ◆ Implemented using varrays of floats, longs, chars, etc.
- ◆ 1:1 relationship between event and tag using bi-directional association
 - Originally intended to support collections of tags
- ◆ Updated design (8.x series?) allows tag sharing by storing borrowed tag in header list

Persistent Tag Descriptor

- ◆ Look-up table for tag values
- ◆ 1 descriptor per event collection
- ◆ Maps the name of each tag value to its type, and offset within the event tag varray
- ◆ The tag is useless without a descriptor!

Event Re-use

- ◆ Original source event:
 - Constructed entirely from transient objects
 - References no other events (points to itself)
 - Currently produced by Elf, Deep Copy, etc.

Event Re-use

- ◆ **Non-skim event:**
 - Construction based on an existing persistent input event
 - RenewHeaderList specified what components are updated using the transient AbsEvent
 - ‘All’ will automatically renew all headers + tag
 - Remaining headers are borrowed from input event
 - Points to original source event
- ◆ **Skim event:**
 - No new persistent event created

Which Descriptor?

- ◆ Persistent event contains:
 - Reference to event tag
 - Reference to original source event
 - Reference to parent collection
- ◆ Use current or original descriptor?
- ◆ We think there should only be 1 valid option

Major Limitations

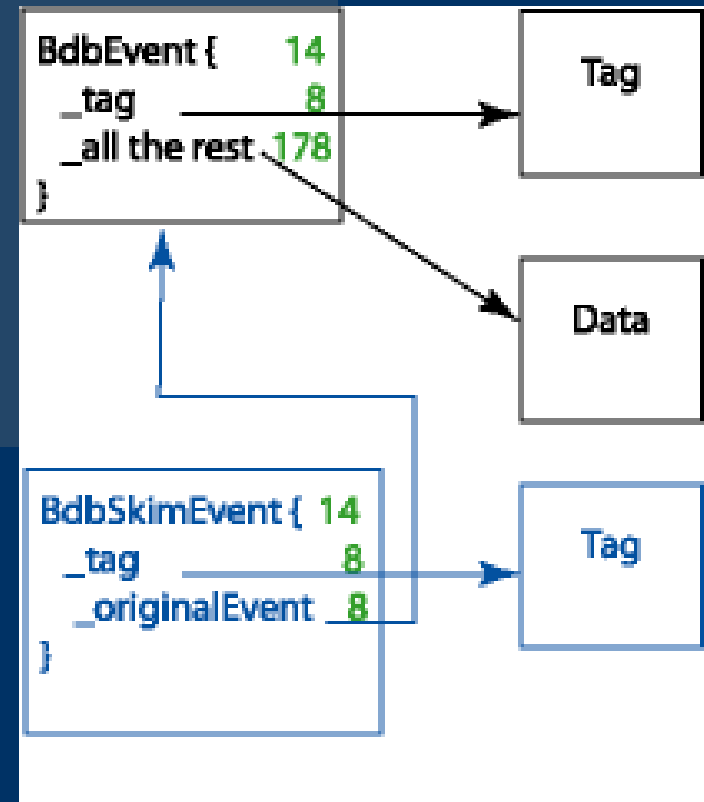
- ◆ Tag <-> Descriptor coupling is very weak:
 - No means of validating that descriptor matches tag
- ◆ Constrained by 1 collection per descriptor:
 - Descriptor is used to resize tag varrays
 - Collections may have events with different tag layouts
- ◆ Descriptor string names consume a lot of space:
 - More than 500 names on average
 - Each name stored as char[64] array
- ◆ An event may only have one tag:
 - Some jobs only update the tag
 - Need a new tag? Write a new event

Persistent Improvements

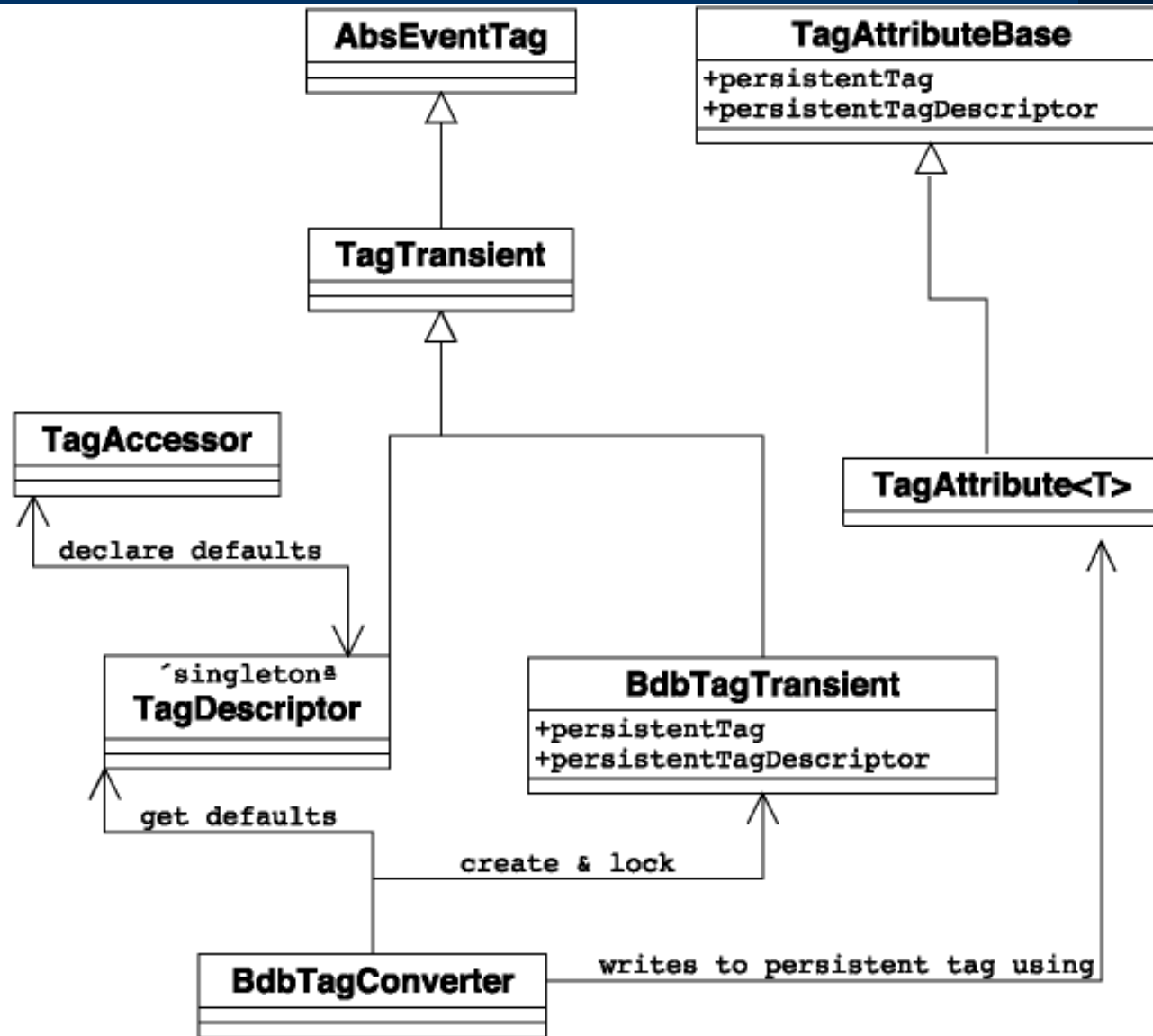
- ◆ Many events will share a single descriptor through common object
 - No more 'invalid' tag entries
 - automatic descriptor selection
- ◆ Pack tag names into single string
 - Current system uses char[64] but average length is 10 chars!
 - Compare packed string using hashing algorithm?

Persistent Improvements

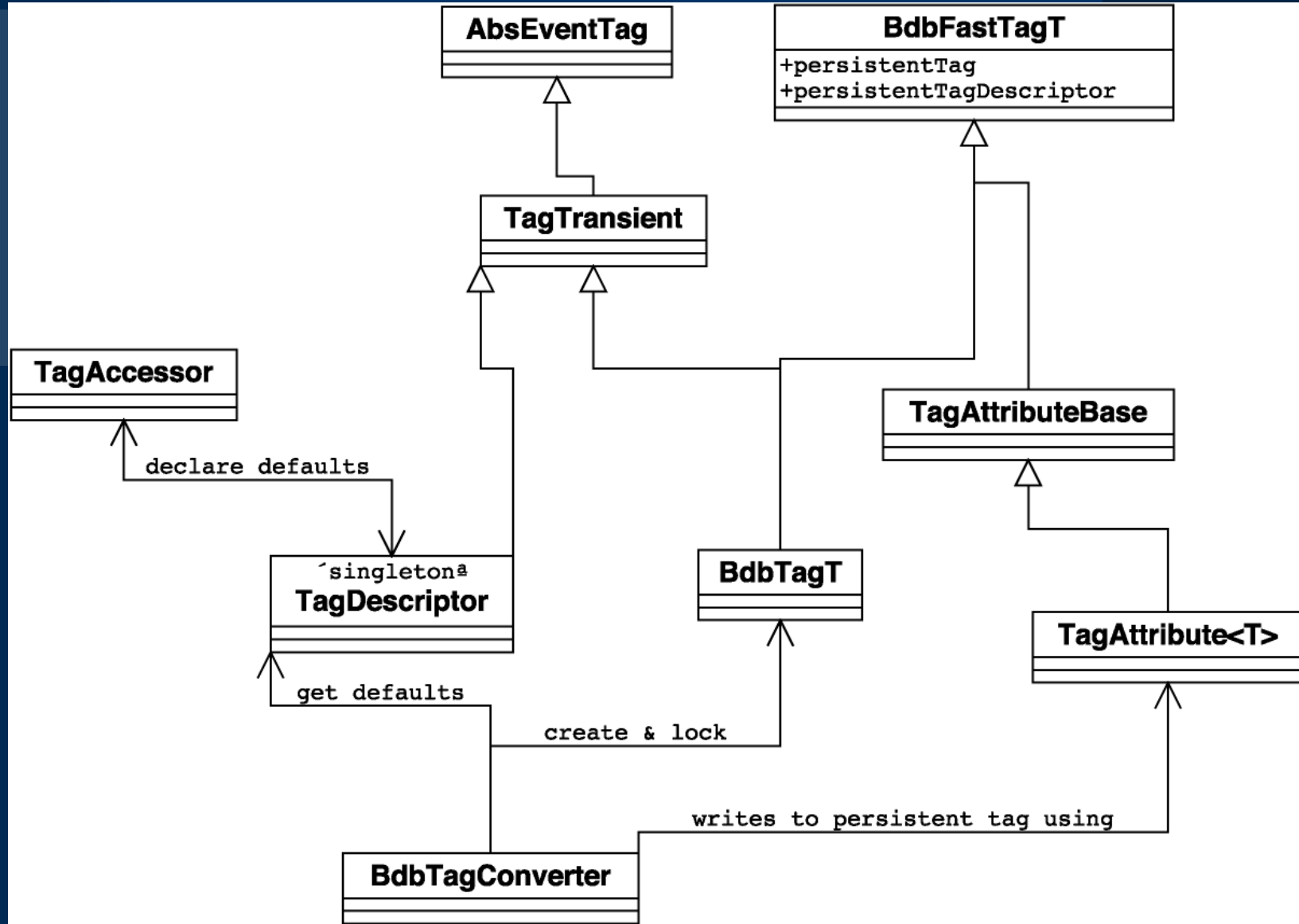
- ◆ BdbSkimEvent:
 - Similar to BdbEvent minus all the header details
 - Refers to a BdbEvent for header information
 - Significantly (~80%) smaller
- ◆ Tag reference is now uni-directional to facilitate tag sharing



Existing Tag Interface



Migrated Tag Interface



TransientTagValue

- ◆ Defines a Tag value that exists only in the transient world
- ◆ User supplies a TransientTagValue object:
 - defines the tag value name, type and a function that generates it's value
 - Value is computed using 'normal' tag values only
- ◆ TagTransient may hold a list of TransientTagValue objects
- ◆ Access to values is transparent through TagTransient:
 - You can only 'get' but you cannot 'set'

Tag Migration Progress (so far)

- ◆ Succeeded in moving interface to BdbFastTagT and BdbTagT
- ◆ Tested with example Beta jobs
- ◆ Great opportunity to scrutinize code and make performance gains
 - Bug discovered related to loading packed bools into TagTransient
 - Valeri Sytnik identified inefficient use of tag descriptor handles
- ◆ We can now output a tag 4x faster than before without persistent changes

How You Can Help

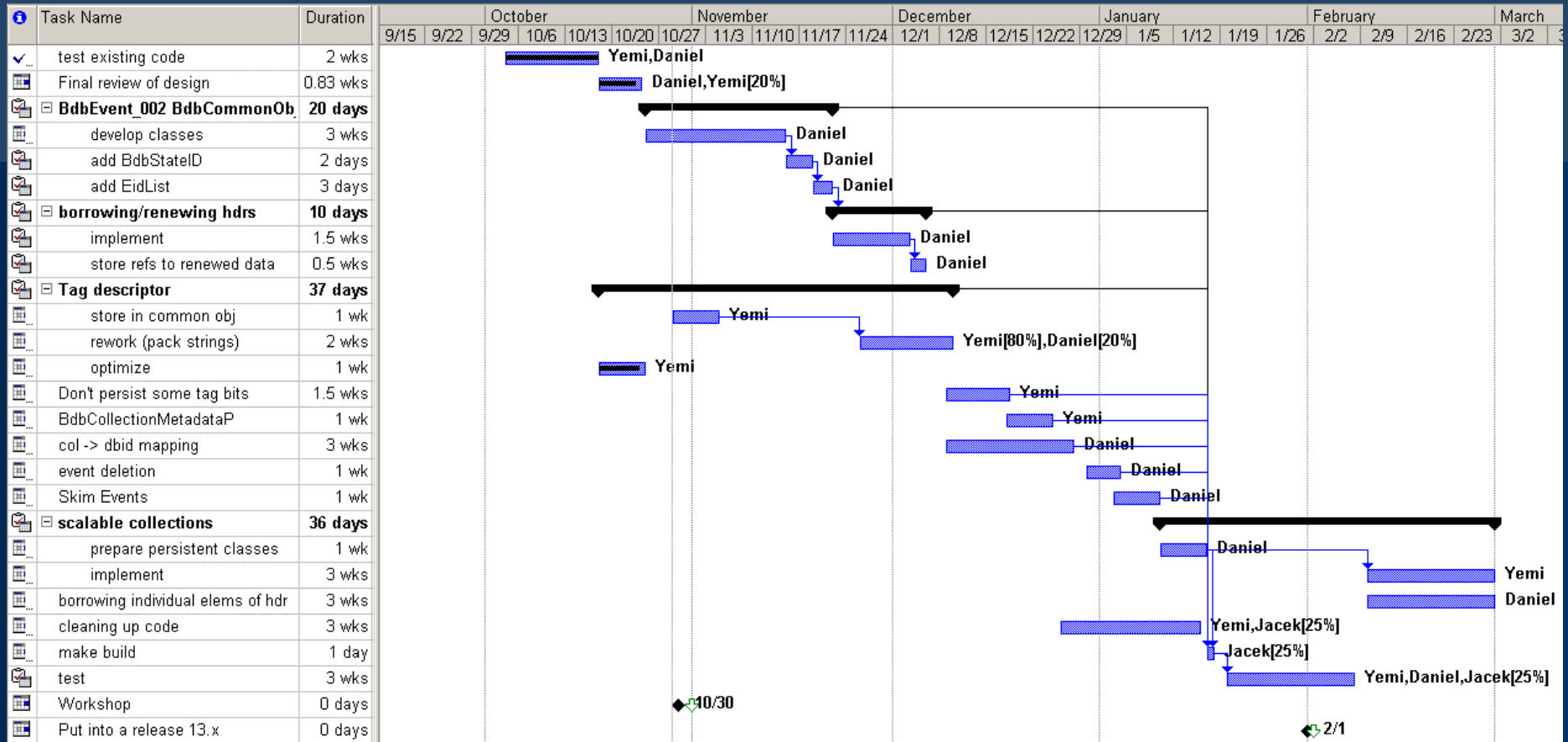
- ◆ Can we eliminate any tag types?
- ◆ Let's try and reduce the number of tag values

Compatibility

- ◆ Changes transparent to users
- ◆ Old releases unable to use new persistent classes
- ◆ Dealing with “old” type event (reprocessing, skimming)
→ product = “old” type event
- ◆ Mixing “old” and “new” events in one collection will be supported
- ◆ No migration necessary

Cost

- ◆ Total estimated manpower
 - Total estimated ~16 FTE-months (done 7, do to 9)
- ◆ Tentative schedule:



Summary

- ◆ Fresh look at Event Store was really needed
 - Optimizing size & performance
 - Lots of cleanup
- ◆ Deadline: all finished before 13 / 14 series in production
 - Planning to finish much earlier