# Adding tracks to the Human Genome Browser

**Charles Sugnet**

This talk, some sample files, etc. Can be found at:

`http://www.soe.ucsc.edu/ sugnet/doc/trackHowto/`

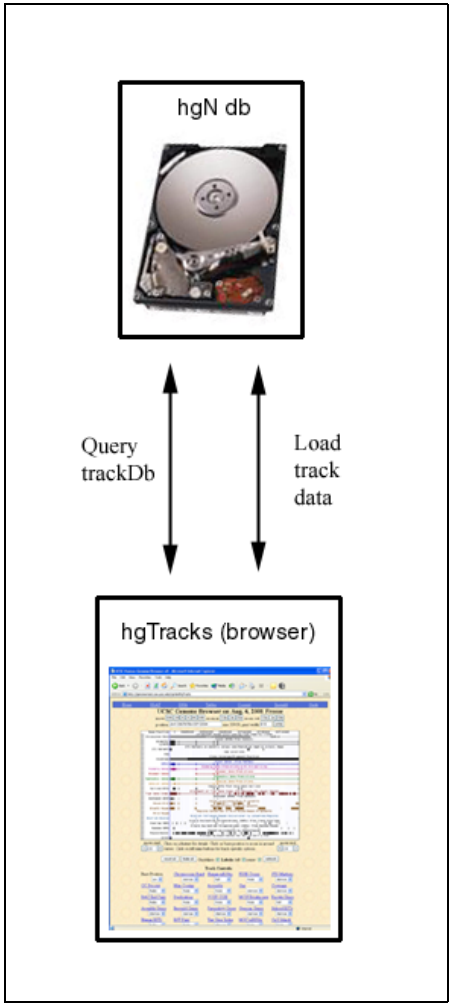
# Adding tracks to the browser.

Talk Outline:

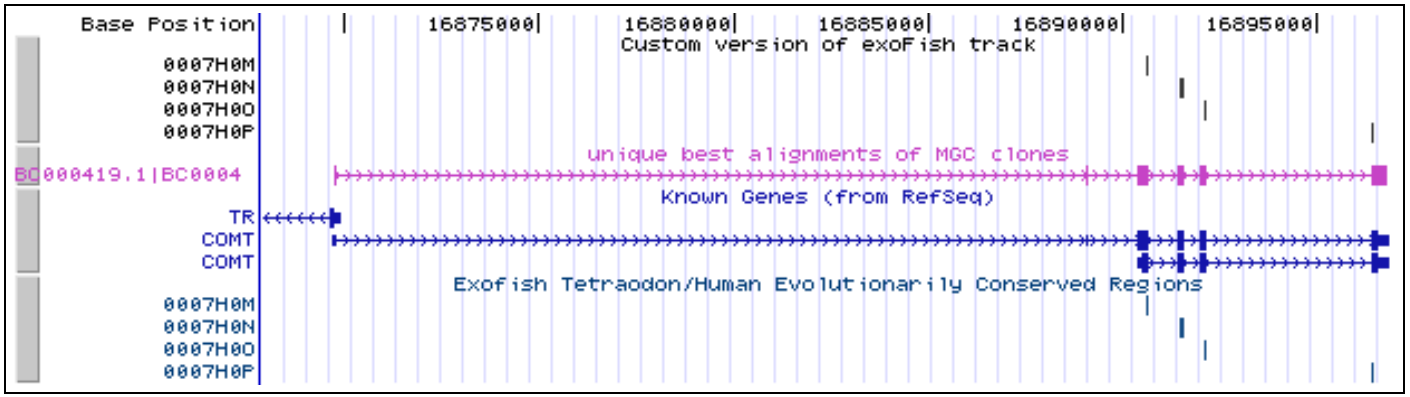
- **Adding a standard track.**
  - Custom tracks - Great way to test stuff.
  - Making tracks permanent - trackDb table.
  - Common browser formats - beds, psls, etc.
- Customizing browser display and details page.
  - Custom displays - Hacking the trackGroup data structure.
  - Inheritance (kind of...) in hgTracks.
  - Setting up custom displays in hgc program.
- Adding filters to tracks.
  - Filters in hgTracks.
  - Saving state in the cart.
  - Introducing hgTrackUi.

# The Browser

- At most basic level, browser is a visualization tool for data. Common reference is the genomic sequence, metaphor is different "tracks" of data layered on top of genomic sequence. Tools for navigating, customizing, and downloading the data.
- Two important parts:
  1. The data. Putting data into database, common formats are bed, psl, genePred.
  2. The display. Loading, Drawing, Customizing data.



# As simple as cut and paste...




- Top two tracks are custom tracks pasted into the browser. Colors, names, scores, links, etc. are configurable. Most of the browser tracks can be reproduced this way.
- Exofish track comes from tab-delimited bed file format. Paints continuous bases of genome where high homology to Puffer Fish.
- MGC track comes from psl alignment file. Paints non-contiguous bases of the human genome, but bases painted are linked to each other.

# Everybody's favorite: File Formats!

Browser knows how to visualize data from a number of different file formats. Internally tracks are represented as simple bed structures and linkedFeatures structures. Externally tracks are have a lot of different tab delimited formats but four main formats dominate.

1. Pat Space Alignments - .psl files generated by aligning sequences to genome using BLAT.
2. Simple Beds - .bed files, originally started with chrom, chromStart, chromEnd, name format.
3. Extended Beds - still .bed files but have grown to include many more fields and generalized to include all the features of the other file formats.
4. Gene Predictions - Derived `genePred` structure, simpler version of psl format with extension of cdsStart and cdsEnd. Mainly an internal format.

# File Format Gotchas!

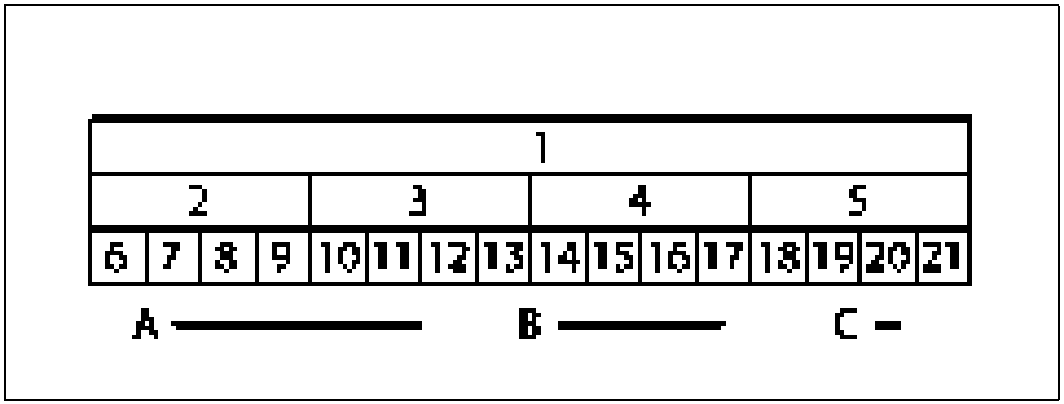
- Documentation: `http://genome.ucsc.edu/goldenPath/help/customTrack.html` and at: `http://genome.ucsc.edu/goldenPath/gbdDescriptions.html`.
- The psl Gotcha (from gbdDescriptions.html): While qStart and qEnd are in relation to the plus strand the qStarts[] are in relation to the negative strand.

```
0         1         2         3 tens position in query
01234567890123456789012345678 90 ones position in query
            ++++          +++++ plus strand alignment on query
    --------    ----------      minus strand alignment on query
Plus strand:
qStart 12 qEnd 31 blockSizes 4,5 qStarts 12,26
Minus strand:
qStart 4 qEnd 26 blockSizes 10,8 qStarts 5,19
```

- Order of Bed Fields: Cannot change order of bed fields, i.e. if you want to include a value of the $Nth$ field you must include all $N-1$ fields up to it.

# The bin field.

As the tables in the browser database got bigger (i.e. Mouse homology tracks) the browser got significantly slower. To speed up queries Jim implemented a binning scheme on a chromosome basis.




- By placing features into the smallest bin in which they fit queries can be optimized by specifying the bin instead of a time consuming range query.
- Because this optimization was implemented later in the design cycle, programs like autoSql don't know about it. For the most part don't need to worry about if using `hgLoadBed` or `hgLoadPsl` to load tables. If writing custom load function use `hRangeQuery()` to query for items.
- Note that the bin field is not present in a bed file, only in the database table. When specifying number of fields in your track in `trackDb.ra` don't include the bin field. Specifically you're much more likely to have 12 fields rather than 13 for an extended bed.

# Making a track permanent

- Browser relies on a table called trackDb to determine what tracks are available and how they are to be displayed. Values include trackTypes, labels, colors, urls, html, etc. associated with the table. See `kent/src/hg/makeDb/hgTrackDb/hgRoot/README` for detailed information.
- To create a track permanently:
  1. Create table in database. Use program like `hgLoadBed`, `hgLoadPsl`, and `hgLoadRna` to load the tab delimited file containing data.
  2. Create an entry in `kent/src/hg/makeDb/hgTrackDb/hgRoot/trackDb.ra`. Also write an html blurb which will be written as an explanation of your track into file called 'yourTrackName'.html into same directory. (Note do **not** put <html><body> and </html></body> in file.)
  3. `cvs` add 'yourTrackName'.html and `cvs` commit both the html file and `trackDb.ra`.
  4. Do a "`make update`" in `hgTrackDb` directory.

# Checklist for adding a track.

- ☐ Put data in psl, simple bed, extended bed, or genePred format. (Preferably extended bed).
- ☐ Load data into hgN database using `hgLoadBed` or `hgLoadPsl`.
- ☐ Do "`cvs update -d`" in `trackDb.ra` directory. Create entry in `trackDb.ra` file and create an html blurb in same directory in file 'yourTrackName'.html.
- ☐ Do "`cd .. && make update`". Spend some time testing the behavior of your track and then
  "`cvs add 'yourTrackName'.html`"
  and "`cvs commit 'yourTrackName'.html trackDb.ra`".
- ☐ Ask system administrators to push table and new trackDb.

# Adding tracks to the browser.

Talk Outline:

- Adding a standard track.
  - Custom tracks - Great way to test stuff.
  - Making tracks permanent - trackDb table.
  - Common browser formats - beds, psls, etc.
- **Customizing browser display and details page**.
  - Custom displays - Hacking the trackGroup data structure.
  - Inheritance (kind of...) in hgTracks.
  - Setting up custom displays in hgc program.
- Adding filters to tracks.
  - Filters in hgTracks.
  - Saving state in the cart.
  - Introducing hgTrackUi.

# Hacking hgTracks Display


- Perlegen track is a unique type of track. Shows which haplotype blocks of SNPs can be tested for using a small subset of SNPs.
- Data stored in familiar bed format. However custom display requested to help visualize break points and recombination hot spots.
- The thicker middle line and custom sized smaller black tics for the SNPs requires a custom function. Have to address code reusage issues, should old functions be refactored to allow new fuctionality or create all new function?
- Either way time to override default behavior. In C++ or Java would inherit functionality from base classes and customize particular function. In C, write the drawing function and assign it to function pointer in trackGroup structure. Seems a little awkward at first but at least no virtual functions...

# trackGroup Function Reference

- struct trackGroup* next
- void* items
- Lots of other data values ....
- **void (* loadItems )(struct trackGroup *tg)**
- **void (* freeItems )(struct trackGroup *tg)**
- **char* (* itemName )(struct trackGroup *tg, void *item)**
- **void (* drawItems )(struct trackGroup *tg, int seqStart, int seqEnd, struct memGfx *mg, int xOff, int yOff, int width, MgFont * font, Color color, enum trackVisibility vis)**
- **Color (* itemColor )(struct trackGroup *tg, void *item, struct memGfx *mg)**
- char* (* mapItemName )(struct trackGroup *tg, void *item)
- int (* totalHeight )(struct trackGroup *tg, enum trackVisibility vis)
- int (* itemHeight )(struct trackGroup *tg, void *item)
- int (* itemStart )(struct trackGroup *tg, void *item)
- int (* itemEnd )(struct trackGroup *tg, void *item)
- void (* mapItem )(struct trackGroup *tg, void *item, char * itemName, int start, int end, int x, int y, int width, int height)
- void* extraUiData
- void (* trackFilter )(struct trackGroup *tg)
- void* customPt

```
/* do some setup then enter core routine. Almost all of Jim's strutures are also linked lists */
for(lf = tg->items; lf != NULL; lf = lf->next)
    {
    /* draw the thick blue line from the start SNP to the stop SNP */
    if (lf->components != NULL && !hideLine)
       {
       /* find our pixel coordinates from genomic coordinates */
       x1 = round((double)((int)lf->start - winStart)*scale) + xOff;
       x2 = round((double)((int)lf->end - winStart)*scale) + xOff;
       w = x2-x1;

       /* get the base line color, in this case shade of blue and draw thick line with it */
       color =  shades[lf->grayIx+isXeno];
       mgDrawBox(mg, x1, y+shortOff+1, w, shortHeight-2, color);
       }
    for (sf = lf->components; sf != NULL; sf = sf->next)
       {
       color = perlegenColor(tg, lf, sf, mg);
       heightPer = perlegenHeight(tg, lf, sf);
       s = sf->start;
       e = sf->end;

       /* draw appropriate box, either a black small tic or blue large tic at the ends */
       drawScaledBox(mg, s, e, scale, xOff, y+((tg->heightPer - heightPer)/2), heightPer, color);

       /* if we're at the stop or start of a blockadd a black tick for the snp to the larger start tic */
       if(heightPer == tg->heightPer)
           drawScaledBox(mg, s, e, scale, xOff,
                         y+((tg->heightPer - heightPer - 4)/2), (heightPer -4), blackIndex());
       }
    /* if we're in full mode increment the height */
    if (isFull) y += lineHeight;
    }
```

# Internal Representation for psls,genePreds, beds

```
struct simpleFeature
/* Minimal feature - just stores position in browser coordinates. */
    {
    struct simpleFeature *next;
    int start, end;                     /* Start/end in browser coordinates. */
    int grayIx;                         /* Level of gray usually. */
    };

struct linkedFeatures
/* Container for multiple simple features that are linked together */
{
    struct linkedFeatures *next;
    int start, end;                     /* Start/end in browser coordinates. */
    int tallStart, tallEnd;             /* Start/end of fat display. */
    int grayIx;                         /* Average of components. */
    int filterColor;                    /* Filter color (-1 for none) */
    float score;                        /* score for this feature */
    char name[32];                      /* Accession of query seq. */
    int orientation;                    /* Orientation. */
    struct simpleFeature *components; /* List of component simple features. */
    void *extra;                        /* Extra info that varies with type. */
    };
}
```

- Custom load routines usually involve writing functions that transform custom data into linkedFeatures, and free them later.
- Check out Documentation at: `http://www.soe.ucsc.edu/ sugnet/kentDoc/`

# Inheritance C style

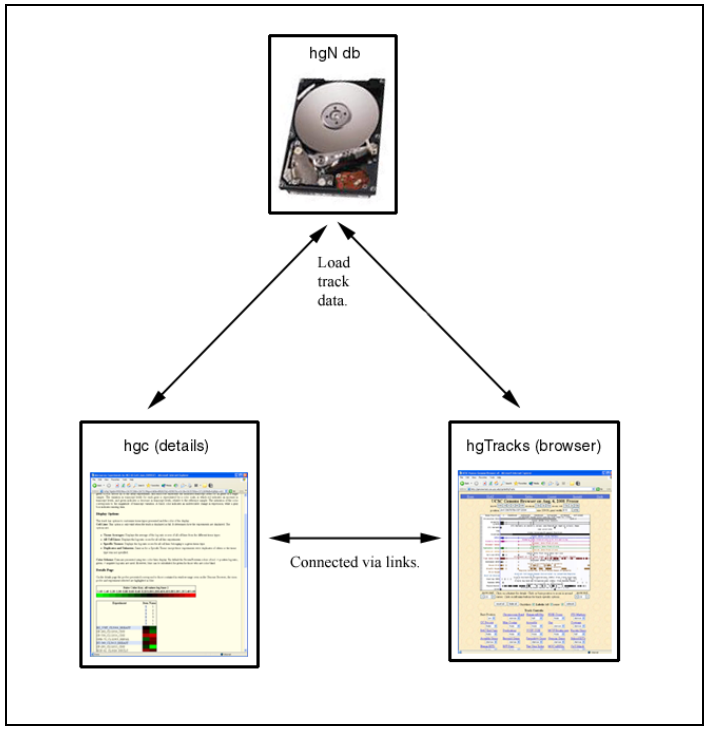
- Want to provide default behavior for tracks, as it is cumbersome to do everything manually. Yet need to provide mechanism for customizing `trackGroup` behavior.
- Meet `void registerTrackHandler(char *name, TrackHandler handler);` this function registers a function pointer which will be called on the track of the same name.
- Example Code:

```
/* call to cutomize perlegen track */
registerTrackHandler("perlegen",perlegenMethods);

void perlegenMethods(struct trackGroup *tg)
/* setup special methods for haplotype track */
{
tg->drawItems = perlegenLinkedFeaturesDraw;
tg->itemName = perlegenName;
tg->colorShades = shadesOfSea;
}
```

# Detail pages:

- All detail pages are handled by hgc "human genome click" program. Links are handled by the `mapItem` function pointer in `trackGroup` structure.
- hgc reads in track name, coordinates, and item name from cgi and uses them to generate html page. Possible to use default page and link to outside resources using url string (i.e. http://somewhere.edu/$$.html).
- Customizing details page means writing cgi response in C and calling function in large switch statement at end of hgc.



# The “Large Switch Statement”

```
tdb = hashFindVal(trackHash, track);
if (sameWord(track, "getDna"))
    {
    doGetDna1();
    }
/* ... bunch of other 'if(sameWord(track,"someTrackName")' checks ...*/
else if (sameWord(track, "perlegen"))
    {
    perlegenDetails(tdb, item);
    }
else if(sameWord(track, "rosetta"))
    {
    rosettaDetails(tdb, item);
    }
else if (tdb != NULL)
   {
   genericClickHandler(tdb, item, NULL);
   }
else
   {
   cartWebStart(track);
   printf("Sorry, clicking there doesn't do anything yet (%s).", track);
   webEnd();
   }
}
```

# Checklist for adding a track.

- ☐ Put data in genePred, psl, simple bed or extended bed format. (Preferably extended bed).
- ☐ Load data into hgN database using `hgLoadBed` or `hgLoadPsl`.
- ☐ Add track information to `trackDb.ra`, don't forget cvs update, add, commit.
- ☐ Backup current version of `hgTracks`. Do **not** let `hgTracks` be dead in the water. cvs update `hgTracks.c`. Add custom functions to `hgTracks.c`, testing as you go. Have someone else look at your code. cvs commit when finished.
- ☐ Backup current version of `hgc`. Do **not** let `hgc` be dead in the water. cvs update `hgc.c`. Add custom functions to `hgc.c`, testing as you go. Have someone else look at your code. cvs commit when finished.
- ☐ Ask system administrators to push new table,trackDb, hgTracks, and hgc.

# Adding tracks to the browser.

## Talk Outline:

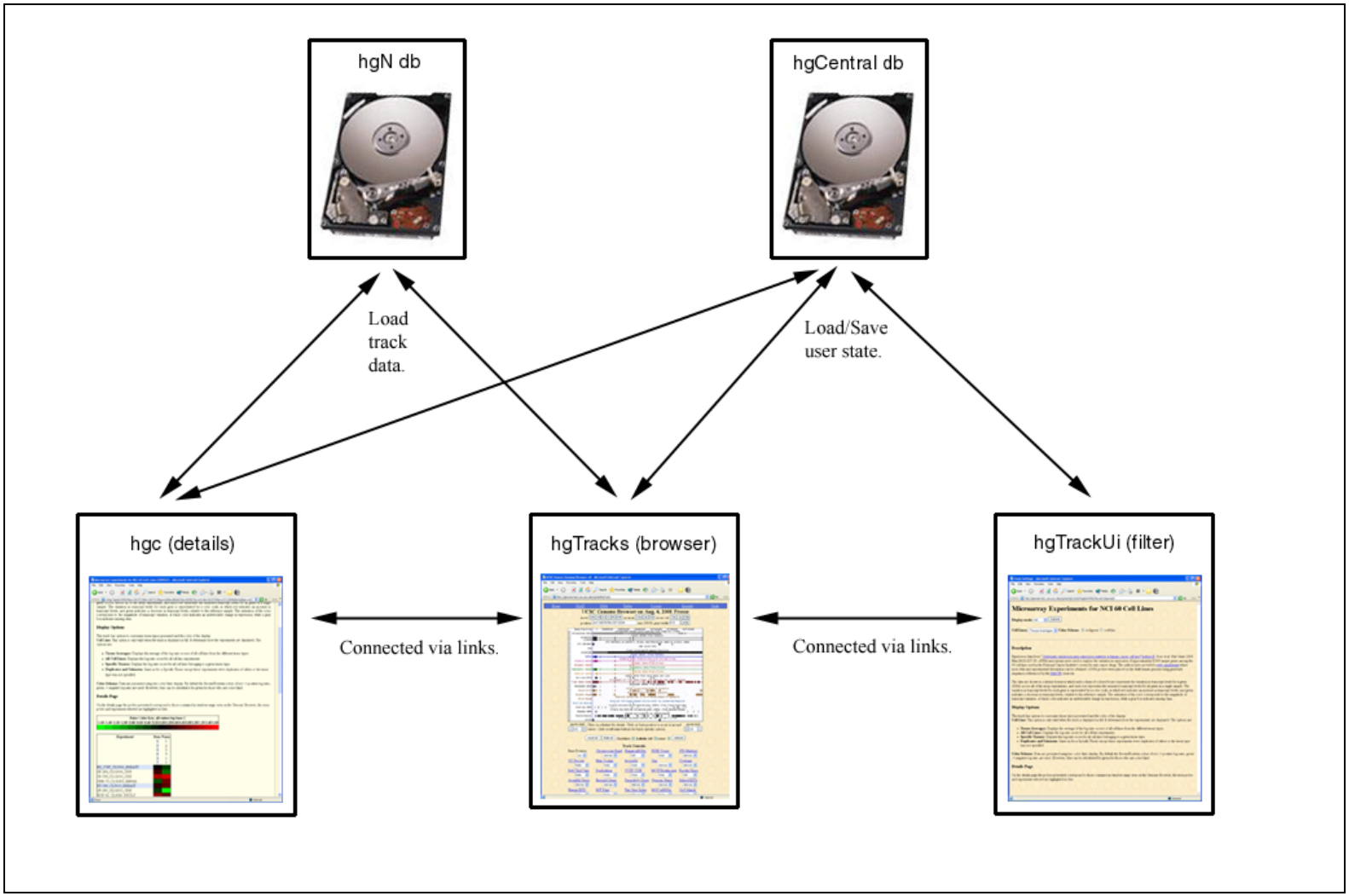
- Adding a standard track.
  - Custom tracks - Great way to test stuff.
  - Making tracks permanent - trackDb table.
  - Common browser formats - beds, psls, etc.
- Customizing browser display and details page.
  - Custom displays - Hacking the trackGroup data structure.
  - Inheritance (kind of...) in hgTracks.
  - Setting up custom displays in hgc program.
- **Adding filters to tracks**.
  - Filters in hgTracks.
  - Saving state in the cart.
  - Introducing hgTrackUi.

# Adding filters to your tracks.

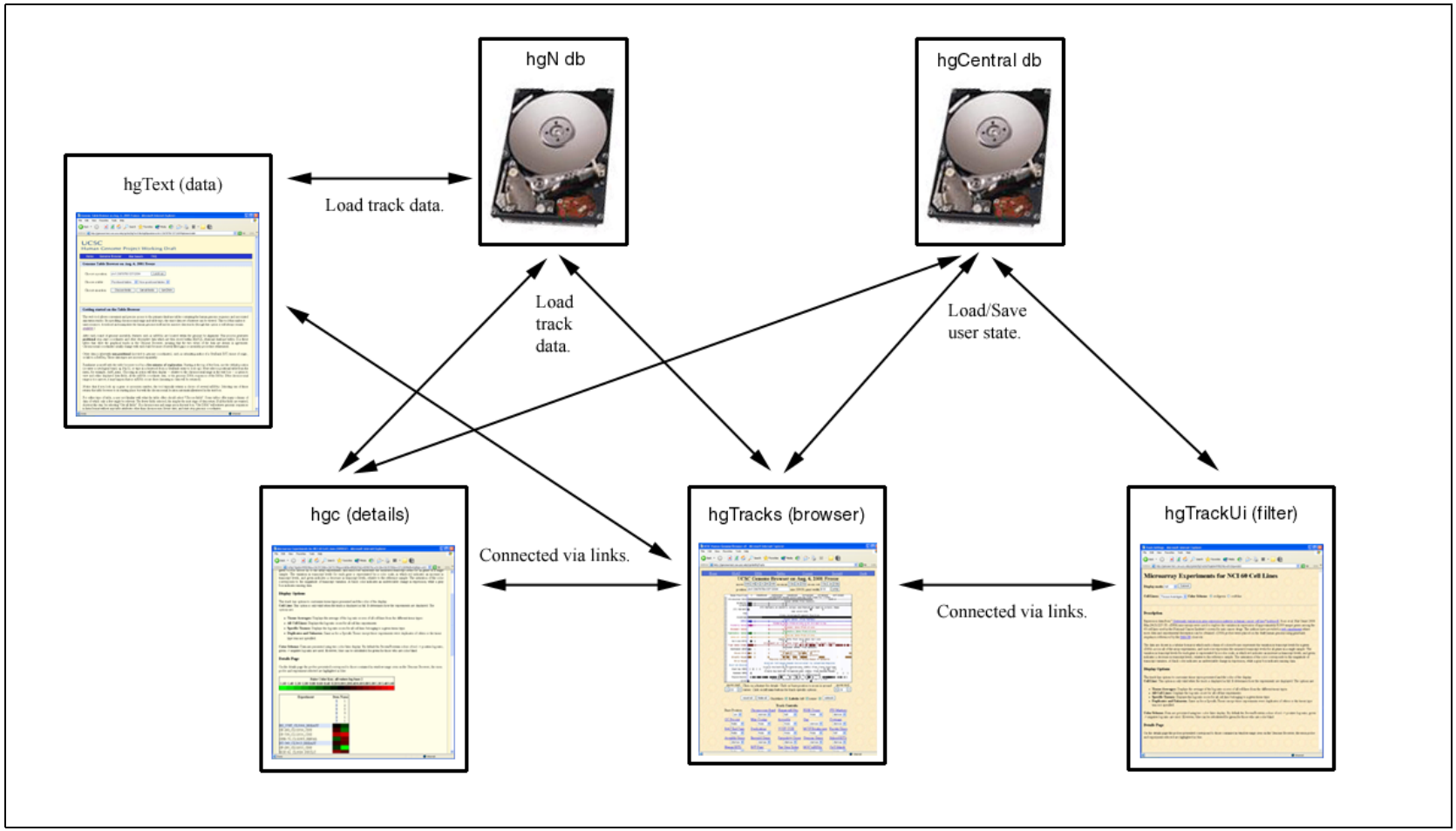
Filters provide a way for user to customize the display. For example, coloring mRNAs from a particular lab or excluding all ESTs that aren't from a certain tissue. Filters are in general run when a track is being loaded. Once items in the current coordinates have been loaded they are run through the filter to establish if they should be displayed and if so what color, etc.
General idea:

- Create user interface in `hgTrackUi`. Currently a lot of details happen in `hui.h` and `hgTrackUi` is mainly a big switch statement.
- Get ui fields of interest from cart. Cart works much the same as cheapcgi module. The cart is queried by different functions of the form:
  `char *exonTypes = cartUsualString(cart, "rosetta.et", "Confirmed Only");`
- Once data from user is loaded use it to "filter" item list that has been loaded. This can include removing items, coloring items, etc. As long as at the end of filtering you have a linked list of `linkedFeatures`.
- Let the rest of hgTracks work its magic...

# hgTrackUi in the world.



# The Browser World.



# Checklist for adding a track.

- ☐ Put data in genePred, psl, simple bed or extended bed format. (Preferably extended bed).
- ☐ Load data into hgN database using `hgLoadBed` or `hgLoadPsl`.
- ☐ Add track information to `trackDb.ra`, don't forget cvs update, add, commit.
- ☐ Backup current version of `hgTracks`. Do **not** let `hgTracks` be dead in the water. cvs update `hgTracks.c`. Add custom functions to `hgTracks.c`, testing as you go. Have someone else look at your code. cvs commit when finished.
- ☐ Backup current version of `hgTrackUi`. Do **not** let `hgTrackUi` be dead in the water. Add code to display user interface, keep things common to `hgTracks` and `hgTrackUi` in `hui.h`. Add code to implement filters (usually in your loading function). Again do testing and have someone else check you code when finished. cvs commit when finished.
- ☐ Backup current version of `hgc`. Do **not** let `hgc` be dead in the water. cvs update `hgc.c`. Add custom functions to `hgc.c`, testing as you go. Have someone else look at your code. cvs commit when finished.
- ☐ Ask system administrators to push new table,trackDb, hgTracks, hgTrackUi and hgc.

# Tips and tricks

- Use a debugger (ddd?) and take the time to walk through your routines and make sure they're doing what you think they should. `cgiSpoof()` allows cgi program to take arguments on the command line of the form: `someCgiProgram cgiVar1=<value1> cgiVar2=<value2>` etc.
- Learn to use cvs and live by it. Can get tricky with lots of people adding/changing things in different programs. **Never ever edit the files in `$CVSROOT`** (`aka /projects/compbio/cvsroot`).
- In general open files read only. (C-x C-r in `emacs`, use `view` instead of `vi`).
- Please respect the style guide. I have an emacs style which will automatically format code for you ( sugnet/jkent-c.emacs).
- Always write a `README` file, helps everyone.
- May be a little much at first, over 80,000 lines of code just in the core libraries and browser programs, over 280,000 lines in `kent/src`. But well worth it! Code to do alignments, parse gff, fasta files, all sorts of other code tested in the fiery pit of 1024 CPUs.