Building a genome browser with GIVE

AUTHORS

Xiaoyi Cao[1,2], Alvin Zheng[1], Sheng Zhong[1,2]*

[1] Department of Bioengineering, [2] 4D Nucleome Network Organizational Hub, University of California San Diego, La Jolla, CA, USA 92093

* Correspondence: szhong@ucsd.edu

ABSTRACT

Genome browsers that manage, communicate and visualize data have greatly facilitated genomics research. Here, we present an open-source programming library, called GIVE, that allows anyone with HTML programing experience to build custom genome browser websites or apps. With a few lines of codes, one can add to a personal webpage an interactive genome browser that host custom data. This portable library encapsulates novel data communication and data visualization technologies, including new data structures and new memory management methods that enable efficient data transfer between the data-hosting website and internet browsers.

INTRODUCTION

Genome browsers have greatly facilitated genomics research (Zhou et al. 2011; Tyner et al. 2017). However, establishing a genome browser on the internet requires very specific expertise. Large amounts of research grants are used on creating genome browser platforms for releasing datasets produced by national and international consortiums. We created the open-source GIVE programming library to simplify such efforts. GIVE enables nearly all people with programming experience, including high-school students, to build custom genome browsers. The web developer only needs rudimentary knowledge of mySQL and HTML to use GIVE. With approximately 20 lines of codes, the developer can publish a genome browser hosting RNA sequencing, ChIP sequencing, or other types of genomic datasets. This simplicity of use came from encapsulation of advanced computing and software technologies, as well as new data structures and data visualization technologies developed by the GIVE development team.

RESULTS

Overview of the GIVE library

GIVE is composed of two major components, that is the database component and the webpage component. The web developer uses the database component with mySQL to create two data tables, one holding genome annotation and the other holding custom genomic datasets (tracks) (Figure 1A). The developer uses the webpage component by inserting HTML tags to a web page, resulting in display of desired genomic regions with annotation and genomic data in web browsers (Figure 1B). Different HTML tags in the GIVE library provides flexibility for the developer to builder a variety of genome browsers, for examples single-cell transcriptome website (Biase et al. 2014) (https://singlecell.genemo.org/), ENCODE (The ENCODE Project Consortium 2012)(Yue et al. 2014) epigenome website (https://encode.genemo.org/, Figure 2), and RNA-chromatin interaction website (Sridhar et al. 2017) (https://give.genemo.org/?hg38).

We will use the website hosting single-cell transcriptomes (https://singlecell.genemo.org/) as an example to describe the process of building a genome browser with GIVE. A line by line pseudocode is provided in Table S1. This process involves 1) creating a database for the reference genome and specifying the chromosome number and sizes, 2) creating two track annotation tables, one holding the data type for each track, and the other holding other track metadata, 3) loading the data tracks with data and metadata, and 4) creating a website and inserting HTML tags to display the data.

Without additional coding, the website is automatically equipped with a few interactive features. These features were enabled by JavaScript codes that were encapsulated within the GIVE's HTML tags. Visitors to this website can input new genome coordinates (Figure S1A), choose any subset of data tracks to display (Figure S1B), or change genome coordinates by dragging the coordinates to left or right by mouse (Figure 1C) or zoom in or out the genome by scrolling the mouse wheel while the mouse pointer is on top of the genome coordinate area (Figure S1C).

New data structures for transfer and visualization of genomic data

We developed two data structures for optimal speed in transferring and visualizing genomic data. These data structures and their associated technologies are essential to GIVE. However, all the technologies described in this section are behind the scene. A website developer who uses GIVE does not have to recognize the existence of these data structures.

We will introduce the rationales for developing the new data structures by a use scenario. When a user browses a genomic region, all genome annotation and data tracks within this genomic region should be transferred from the web server to the user's computer. At this moment, only the data within this genomic region require transfer and display (Figure 3A). Next, the user shifts the genomic region to the left or right. Ideally, the previous data in user's computer should be re-utilized without transferring again, and only the new data in the additional genomic region should be transferred. After data transfer, the previous data and the new data in user's computer should be combined (Figure 3B).

Next, the user zooms out. This action changes the resolution of genome. It is unnecessary to transfer and infeasible to display data at the previous granularity. At this point, the program should adjust the granularity of the already transferred data, and then transfer additional data at the new granularity (Figure 3C). When the user zooms in, the program would adjust to finer granularity, and transfer data at this resolution (Figure 3D). In summary, what is needed is a multiscale data container that can add or remove data from both sides of a genomic window.

To substantiate the above described multiscale data container, we developed two data structures named Oak and Pine. Oak handles sparse data tracks such as genome annotation, gene tracks, peak tracks, and interaction regions (BED, interaction data). Pine handles dense data track in bigWig format (Kent et al. 2010)). Once the user changes the viewing area, Oak and Pine automatically adjust to the optimal tree structure for holding the data in the viewing area, which may involve change of data granularity, change of tree depths, adding or merging nodes, and rearrangement node assignment to branches. These operations minimize data transfer over the internet as well as the amount of data loaded in computer memory.

To optimize the use of memory, we developed an algorithm for removing obsolete data from the memory ("withering"). When the data stored in Oak or Pine nodes have not been accessed by the user for a long time, data in these nodes will be dumped and memory is recycled.

METHODS

System requirements

Both mySQL and PHP have to be available on the data server. Follow GIVE manual for installation of these prerequisites if necessary. The website (HTML file) can be placed on any server, including but not necessarily to be the data server.

Installation

GIVE does not require downloading or installation. Inserting the two following HTML lines with reference to the URL of GIVE library will import the entire GIVE library to your website (Lines 1, 2).

```
<script src="https://demo.give.genemo.org/components/bower_components/webcomponentsjs/webcomponents-lite.min.js"></script>    (Line 1)
```

```
<link rel="import" href="https://demo.give.genemo.org/components/bower_components/genemo-visual-components/chart-controller/chart-controller.html">    (Line 2)
```

To display genomics data, the web developer can use either <chart-controller> tag or <chart-area> tag. The <chart-controller> tag will display track data as well as genome navigation features such as shifting, zooming (Figure S1A-B). For example, adding the following line (in addition to the two line above) would create a website similar to Figure S1 (Line 3).

```
<chart-controller title-text="Single-cell RNA-Seq Genome Browser" group-id-list='["genes", "singleCell", "customTracks"]' num-of-subs="1"></chart-controller>    (Line 3)
```

Here, the title-text attribute sets the title text of your website. The <chart-area> tag will display the track data without metadata controls such as track selection buttons and coordinate input box, while retaining some interactive capacities including dragging and zooming. This option provides the developer greater flexibility for website design. In addition, the <chart-area> tag can be used in mobile apps, resulting in insertion of genome tracks to mobile apps.

Data types

The current release of GIVE supports BED and bigWig files. GIVE also accepts genome interaction data stored in BED format (see GIVE Manual).

The following technologies are wrapped inside the GIVE library. Website developers who uses GIVE do not have to understand them or even knowing their existence.

Query

A query is issued when the user views any genomic region (query region). A new query is issued when the user changes the genomic region. A query induces two actions, which are data retrieval and display of data.

Oak, a data structure

A data structure called Oak is developed to effectively load and transfer a subset of data in BED format. The subset is defined as continuous genomic region within a chromosome. Oak is a type of tree data structures, with nodes defined as follows.

A node is composed of a list of key-value pairs and a set of attributes. a key is a pair of starting and ending genomic coordinates, termed left key and right key, respectively. When populated with data, a node keeps the data for a genomic region defined by the first left key and the last right key. The keys in a node partition the genomic region into non-overlapping sub-regions. A node can be either a branch node or a leaf node. Their differences lie in the values. A branch node is a node where the values are other nodes. A leaf node is a node where each value is a set of two lists of

data points (Figure S2). Each data point is a row of a BED file. When populated with data, the first list contains all the rows in the BED file where the start position matches the left key. The second list contains all the rows where the start and the end positions cover (span across) the left key. A value in a leaf node can also be empty. Leaf nodes with empty values are used to mark the genomic regions outside the query region.

Creating an Oak instance, populating data, and updating Oak

An Oak instance will be created, populated with data, or get updated in response to a query. These actions accomplish data transfer from server to user's computer. Only the data within the queried region will be transferred. Hereafter we will refer an Oak instance as an Oak.

When the query region is on a new chromosome, an Oak will be created as follows. Every unique start position in the BED file that is contained within the query region is used to create a leaf node. The genomic regions on queried chromosome but outside the query region are inserted as pairs of keys and empty values (placeholders) to the nodes with the nearest keys. The leaf nodes are ordered by their first left keys and sequentially linked by their pointers. A root node is created with all the leaf nodes are its children. This initial tree is fed into a self-balancing algorithm (Bayer and McCreight 1972; Comer 1979) to construct a weight balanced tree, thus finishes the construction of an Oak.

When the query region is on a previously queried chromosome, the query region will be compared with the Oak of that chromosome and the overlapping region will be identified. The data of the overlapping region are already loaded in the Oak and therefore for the purpose of saving time this should not be loaded again. The data in the rest of the query region will be loaded to the Oak. This is done by first creating a leaf node for every additional unique start position, removing the placeholder key-value pairs, and adding new placeholder key-value pairs for the rest of the chromosome. The weight balancing algorithm (Bayer and McCreight 1972) is invoked again to re-balance this Oak. The weight balancing step prepares the Oak for efficient response to future queries.

Pine, a data structure

A data structure called Pine is developed to effectively load and transfer a subset of data in bigWig format. The subset is defined as continuous genomic region within a chromosome. Pine can automatically determine the data granularity, which avoids transferring data at a higher than necessary resolution. The resolution of displayed data is limited by the number of pixels on the screen. Pine instances are always constructed to the appropriate depth and match the limit of the resolution.

A node is composed of a list of key-value pairs and a set of attributes. The attributes are the same as that of Oak nodes, except for having one additional attribute, called data summary. The data summary includes the following metrics for this node (the genomic region defined by the first left key and the last right key of this node): the number of bases, sum of values (summing over every base), sum of squares of the values, maximum value, and minimum value. A key is a pair of starting and ending genomic coordinates, termed left key and right key, respectively. The keys in a node

partition the genomic region into non-overlapping sub-regions. A node can be either a branch node or a leaf node. Their differences lie in the values. A branch node is a node where the values are other nodes (Figure S3A). A leaf node is a node where each value is a list of data points (Figure S3B). Each data point is a row (binary format) of a bigWig file.

A note in Pine can have an empty key-value list and an empty data summary, and in this case we call it a placeholder node.

Creating a Pine instance, populating data, and updating Pine

A Pine is created when a query to a new chromosome is issued. A Pine is created with the following steps. First, the depth of the Pine tree is calculated as:

$Tree\ depth = Ceiling\ (log_n(chromosome\ length) - log_n(resolution))$ (Equation 1).

The limit of the resolution (length of genomic region per pixel) is the total length the queried genomic region (viewing area) divided by the number of horizontal pixels, namely the width of the SVG element in JavaScript.

Next, a root node is created with keys covering the entire chromosome, where the query region is contained within. Until reaching the calculated depth, for any node that overlaps with the query region, create a fixed number (n, n=20 in current release) of child nodes by equal partitioning its genomic region. If any of the created child node does not overlap with the query region, use a placeholder node. For each node, point the pointer to the "right hand" node at the same depth. Thus, a Pine is created. This Pine has not loaded with actual data.

To load data, every leaf node issues a request to retrieve the summary data of its covered region (between the first left key and the last right key), which will be responded by a PHP function wrapped within GIVE. This function returns summary data between the input coordinates from the bigWig file. After filling the summary data for all nodes at the deepest level, all parent nodes will be filled, where the summary data are calculated from the summary data of their child nodes. This process continues until reaching the root node.

A Pine will be updated when a new query partially overlaps with a previous query. In this case, the new depth (d2) is calculated using Equation 1. This depth (d2) reflects the new data granularity. If d2 is greater than the previous depth, extend the Pine by adding placeholder nodes until d2 is reached. From root to depth d2-1, if any placeholder node overlaps with the query region, partition it by creating n child nodes. If any of the newly created child node does not overlap with the query region, use a placeholder node. For any newly created node, point the pointer to the "right hand" node at the same depth. At this step the Pine structure is updated into proper depth. Finally, at depth d2, retrieve summary data for every non-placeholder node that has not had summary data. Update the summary data of their parent nodes until reaching the root. In this way, only the new data within the query region that had not been transferred before will get transferred.

Memory management

We developed a memory management algorithm called "withering". Every time a query is issued, this algorithm is invoked to dump the obsolete data, which are not used in the previous 10 queries.

"Withering" works as follows. All nodes are added with a new integer attribute called 'life span'. When a node is created, its life span is set to 10. Every time a query is issued, all nodes overlapping with the query region as well as all their ancestral nodes get their life span reset to 10. The other nodes that do not overlap with the query region get their life span reduced by 1. All the nodes with life span equals 0 are replaced by placeholder nodes.

DISCUSSION

The GIVE library is designed to reduce the needs of specialized knowledge and programming time for building web based genome browsers. GIVE is open-source software. The open-source nature allows the community at large to contribute to enhancing GIVE. The name GIVE (Genome Interaction Visualization Engine) was given when this project started with a smaller goal. Although it has grown into a more general purpose library, we have decided to keep the acronym.

An important technical consideration is efficient data transfer between the server and user's computers. This is because users typically wish to get instant response when browsing data. To this end, we developed several technologies to optimize the speed of data transfer. The central idea is in three folds, that are 1) only transfer the data in the query region, and 2) minimize repeated data transfer by reusing previously transferred data, and 3) only transfer data at the necessary resolution. To implement these ideas, we developed two new approaches to index the genome, and formalized these approaches with two new data structures, named Oak and Pine.

The Oak and Pine are indexing systems for sparse data (BED) and dense data (bigWig), respectively. BED data typically store genomic segments that have variable lengths. Given this particular feature, we did not index the genome base-by-base but rather developed a new strategy (Oak) to index variable-size segments. The bigWig files contain base-by-base data, which for a large genomic region can become too slow for web browsing. We therefore designed the Pine data structure that can automatically assess and adjust data granularity, which exponentially cut down unnecessary data transfer.

ADDITIONAL INFORMATION

GIVE demo:
https://github.com/Zhong-Lab-UCSD/Genomic-Interactive-Visualization-Engine/tree/master/tutorials

GIVE manual:
https://github.com/Zhong-Lab-UCSD/Genomic-Interactive-Visualization-Engine/blob/master/manuals/index.md

ACKNOWLEDGEMENT

We thank the open source community especially authors of the Polymer programming library (Polymer Authors 2017). This work is funded by U01CA200147.

REFERENCES

Bayer R, McCreight EM. 1972. Organization and maintenance of large ordered indexes. *Acta Informatica* **1**(3): 173-189.

Biase FH, Cao X, Zhong S. 2014. Cell fate inclination within 2-cell and 4-cell mouse embryos revealed by single-cell RNA sequencing. *Genome Research* **24**(11): 1787-1796.

Comer D. 1979. Ubiquitous B-Tree. *ACM Comput Surv* **11**(2): 121-137.

Kent WJ, Zweig AS, Barber G, Hinrichs AS, Karolchik D. 2010. BigWig and BigBed: enabling browsing of large distributed datasets. *Bioinformatics* **26**(17): 2204-2207.

Polymer Authors. 2017. Polymer. https://www.polymer-project.org/.

Sridhar B, Rivas-Astroza M, Nguyen TC, Chen W, Yan Z, Cao X, Hebert L, Zhong S. 2017. Systematic Mapping of RNA-Chromatin Interactions In Vivo. *Current Biology* **27**(4): 602-609.

The ENCODE Project Consortium. 2012. An integrated encyclopedia of DNA elements in the human genome. *Nature* **489**(7414): 57-74.

Tyner C, Barber GP, Casper J, Clawson H, Diekhans M, Eisenhart C, Fischer CM, Gibson D, Gonzalez JN, Guruvadoo L et al. 2017. The UCSC Genome Browser database: 2017 update. *Nucleic Acids Research* **45**(D1): D626-D634.

Yue F Cheng Y Breschi A Vierstra J Wu W Ryba T Sandstrom R Ma Z Davis C Pope BD et al. 2014. A comparative encyclopedia of DNA elements in the mouse genome. *Nature* **515**(7527): 355-364.

Zhou X, Maricque B, Xie M, Li D, Sundaram V, Martin EA, Koebbe BC, Nielsen C, Hirst M, Farnham P et al. 2011. The Human Epigenome Browser at Washington University. *Nat Meth* **8**(12): 989-990.

FIGURE LEGENDS

Figure 1. Components of GIVE and number of lines of codes required. A) Loading the data at the backend requires a modest number of SQL commands; B) Building a browser upon loaded data involves a minimal number of HTML tags in the desired webpage.

Figure 2. Screenshot of a website hosting ENCODE datasets.

Figure 3. Scenarios for browser use. A) Displaying a segment of the genome. While no data is stored in cache (blank blocks), only those within the queried region needs to be fetched from the server (colored blocks) and is stored in cache for later use; B) Shifting display window. Only the part not in cache needs to be fetched from the server (colored blocks) and merged in cache; C) Zooming out. Existing cache data are used to recalculate new cache at a coarser granularity level, after which non-overlapping data are requested; D) Zooming in. Because no cached data exists at a finer granularity level, all data within the queried region needs to be fetched at that level.

Figure 1

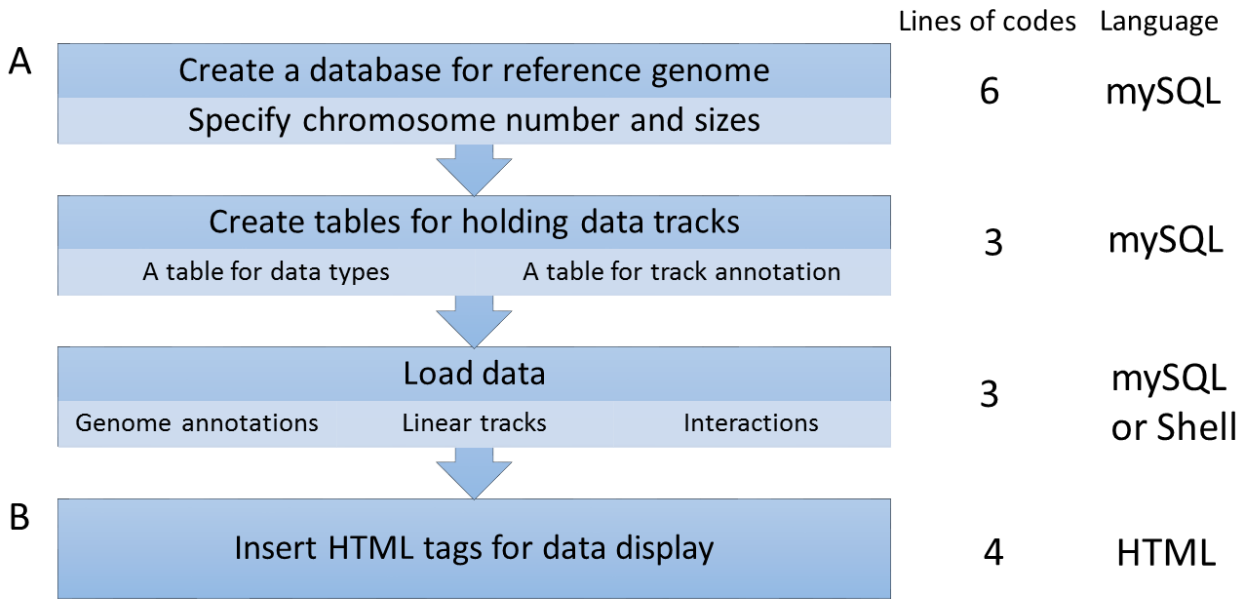

Lines of codes
Language
A
Create a database for reference genome
Specify chromosome number and sizes
6
mySQL
Create tables for holding data tracks
A table for data types
A table for track annotation
3
mySQL
Load data
Genome annotations
Linear tracks
Interactions
3
mySQL
or Shell
B
Insert HTML tags for data display
4
HTML

Figure 2

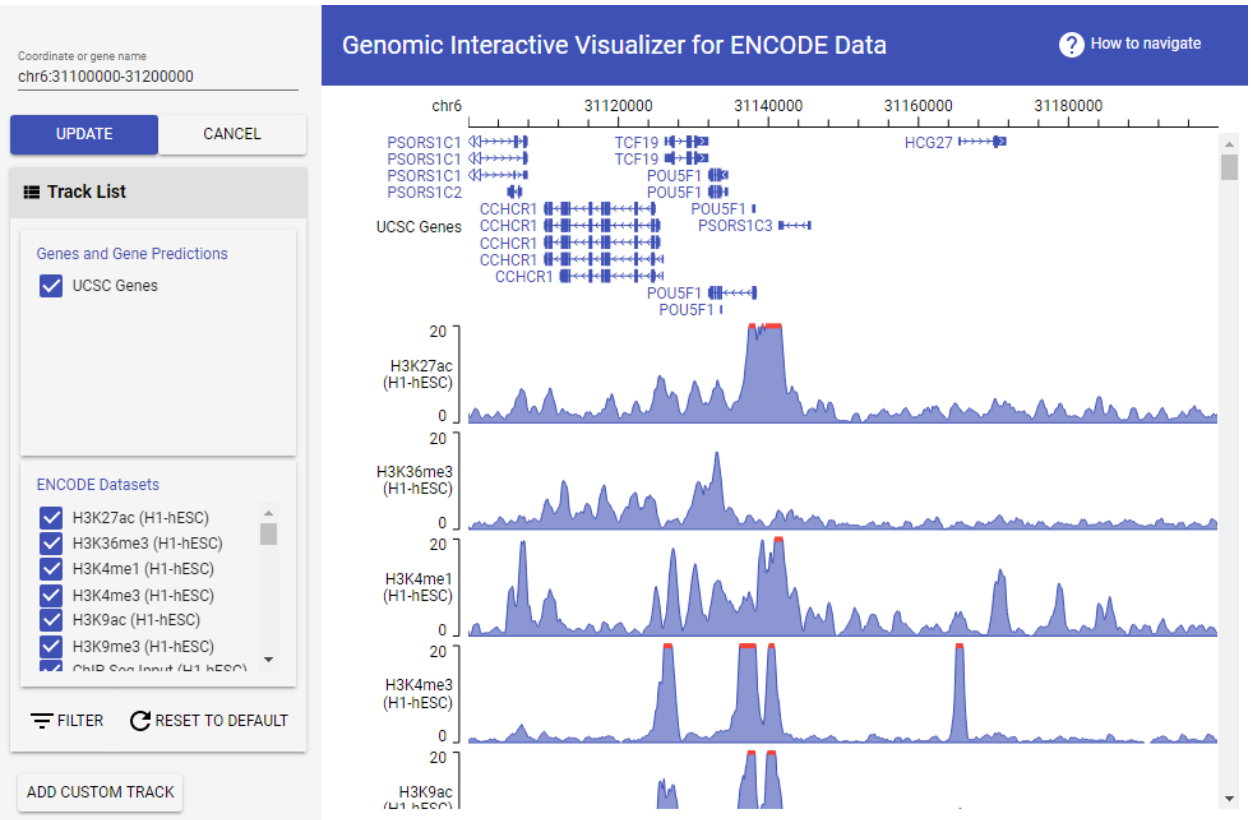
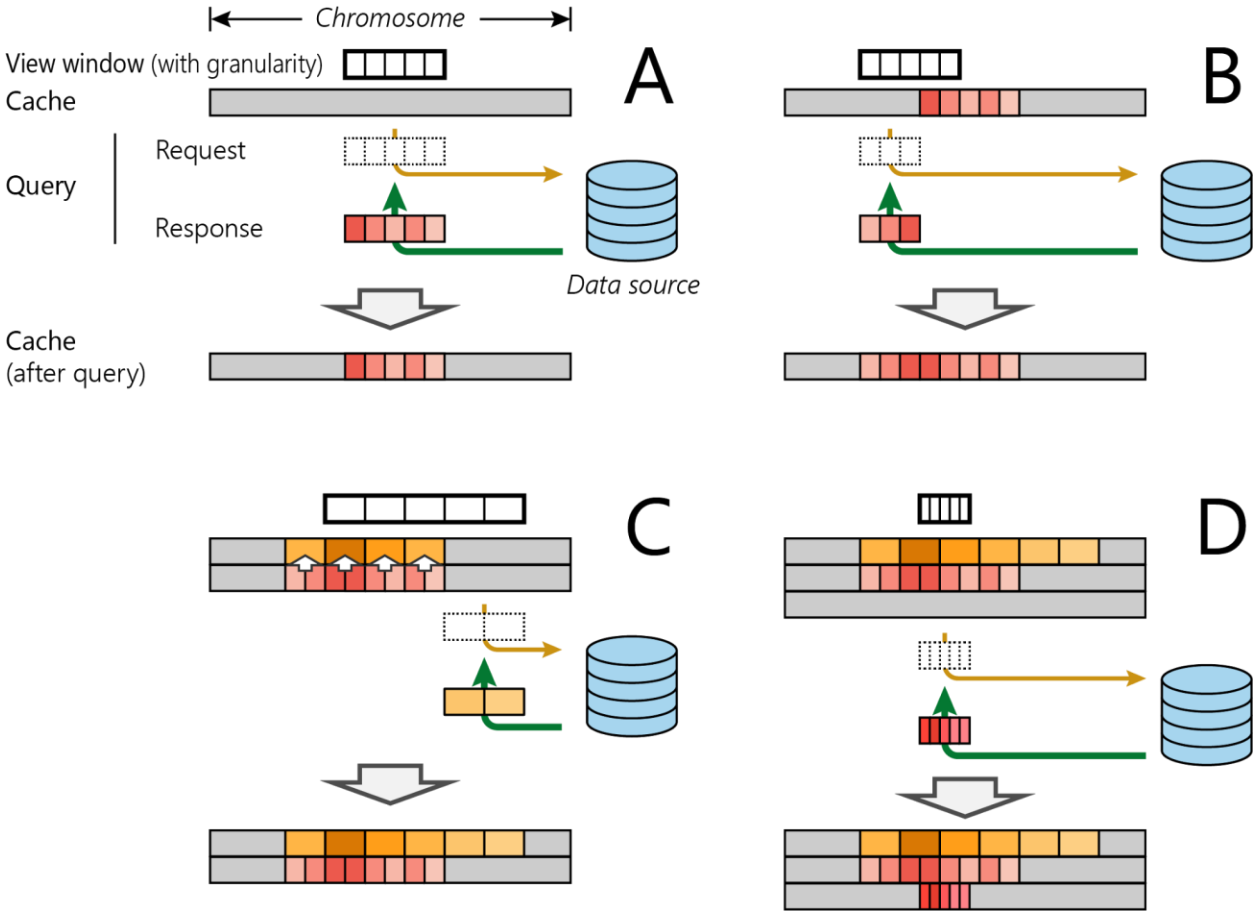

Coordinate or gene name
chr6:31100000-31200000
UPDATE
CANCEL
Track List
Genes and Gene Predictions
UCSC Genes
ENCODE Datasets
H3K27ac (H1-hESC)
H3K36me3 (H1-hESC)
H3K4me1 (H1-hESC)
H3K4me3 (H1-hESC)
H3K9ac (H1-hESC)
H3K9me3 (H1-hESC)
FILTER
RESET TO DEFAULT
ADD CUSTOM TRACK
Genomic Interactive Visualizer for ENCODE Data
How to navigate
chr6
31120000
31140000
31160000
31180000
PSORS1C1
PSORS1C1
PSORS1C1
PSORS1C2
TCF19
TCF19
POU5F1
POU5F1
HCG27
CCHCR1
CCHCR1
CCHCR1
CCHCR1
CCHCR1
POU5F1
PSORS1C3
POU5F1
POU5F1
UCSC Genes
H3K27ac (H1-hESC)
H3K36me3 (H1-hESC)
H3K4me1 (H1-hESC)
H3K4me3 (H1-hESC)
H3K9ac
20
0

Figure 3


Chromosome
View window (with granularity)
Cache
A
B
Query
Request
Response
Data source
Cache
(after query)
C
D