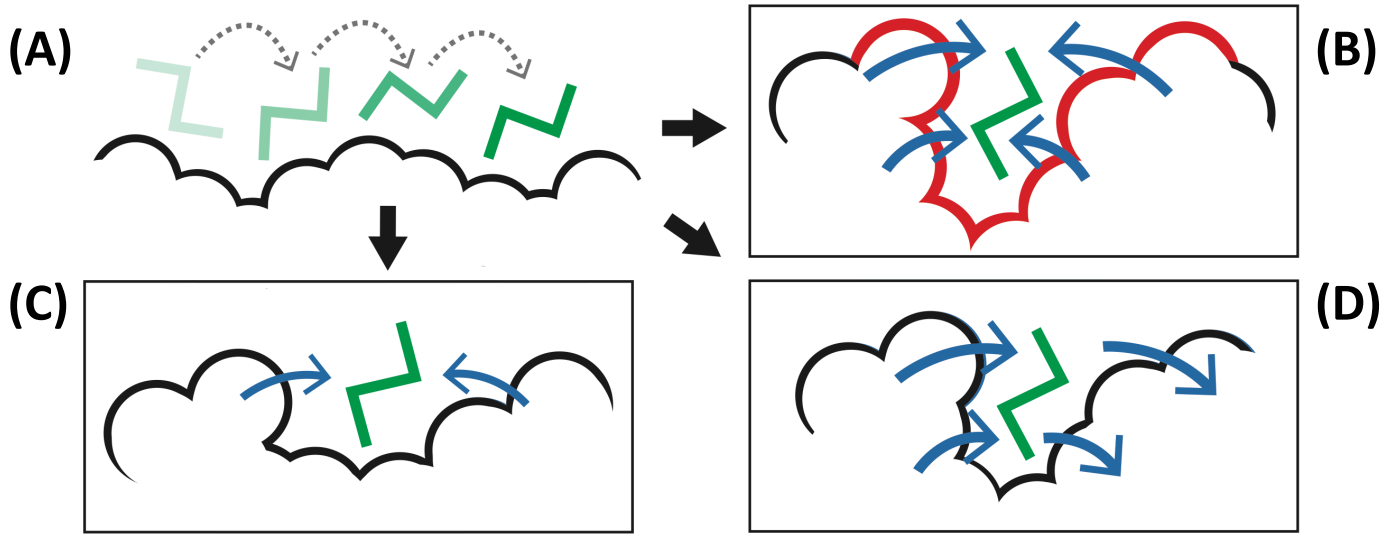
Basic Objective and Idea Behind the Formalism

For Predicting Allosteric Sites on the *Surface*



$$binding\ leverage = \sum_{m=1}^{10}\left(\sum_i\sum_j \Delta d^2_{ij(m)}\right)$$

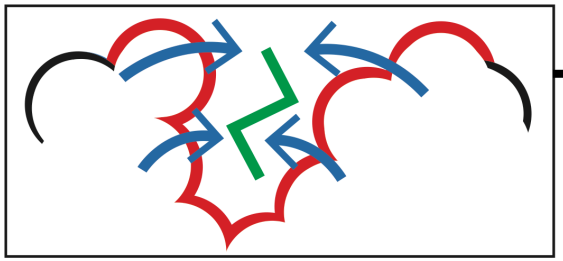
1. Identify pockets on the protein surface by running MC simulations in which a ligand walks along the proteins surface **(A)**

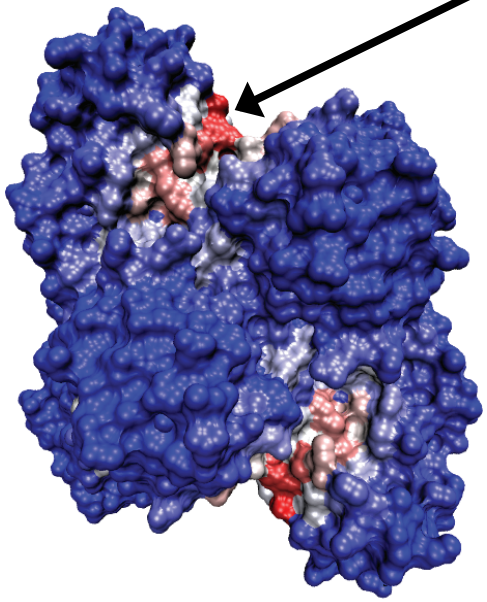
2. Score each pocket by the degree to which it perturbs large-scale motions. Shallow sites **(C)** or sites at which the ligand does not interfere with motion **(D)** earn low scores, whereas deeper pockets at which occlusion 'blocks' motion earn high scores **(B)**.

3. Threshold the list or sites to identify the set predicted allosteric sites at the surface.

STRESS Output for Predicted Allosteric Sites on the *Surface*

stress.molmovdb.org





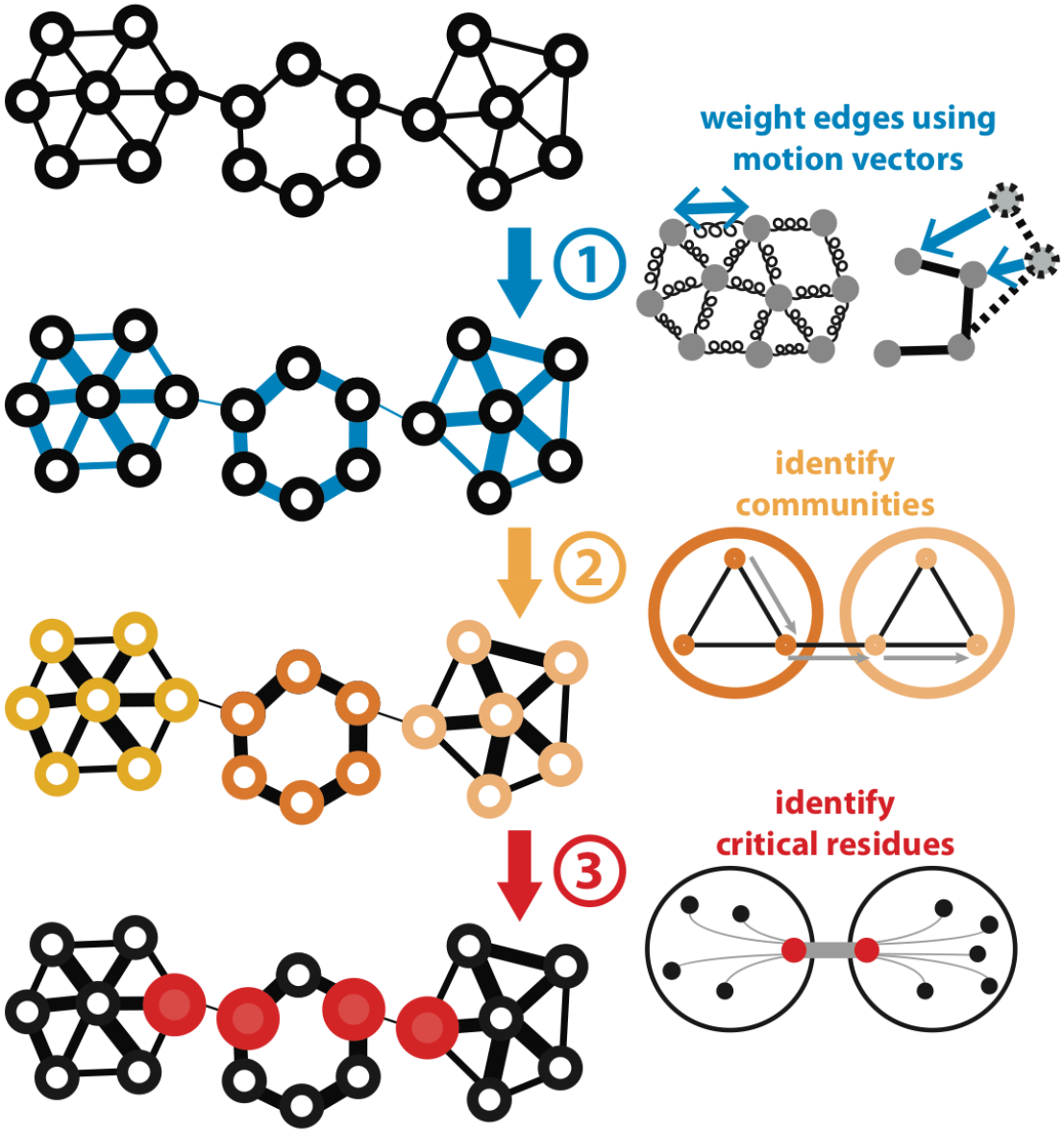
PDB: 3PFK

Output

SITE 1	101_A	10_A	10_B	99_A	98_A . . .
SITE 2	10_A	55_B	56_B	18_A	27_B . . .
SITE 3	34_B	37_A	42_B	108_A	97_A . . .
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

Basic Objective and Idea Behind the Formalism

For Predicting Allosteric Residues within the *Interior*



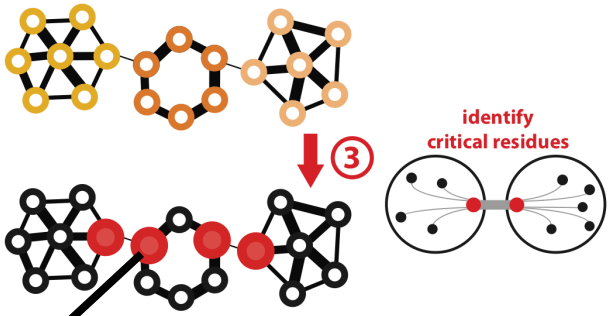
$$Cov_{ij} = \langle \mathbf{r}_i \bullet \mathbf{r}_j \rangle$$

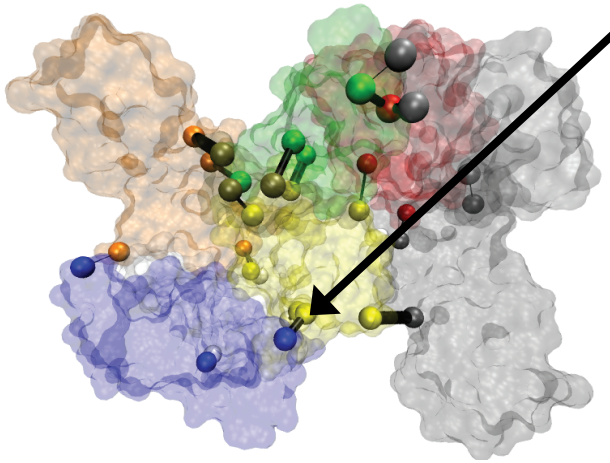
$$C_{ij} = Cov_{ij} \; / \; \sqrt{(\langle \mathbf{r}_i^2 \rangle \langle \mathbf{r}_j^2 \rangle)}$$

$$D_{ij} = -\log(|C_{ij}|)$$

STRESS Output for Predicted Allosteric Residues within the *Interior*

stress.molmovdb.org





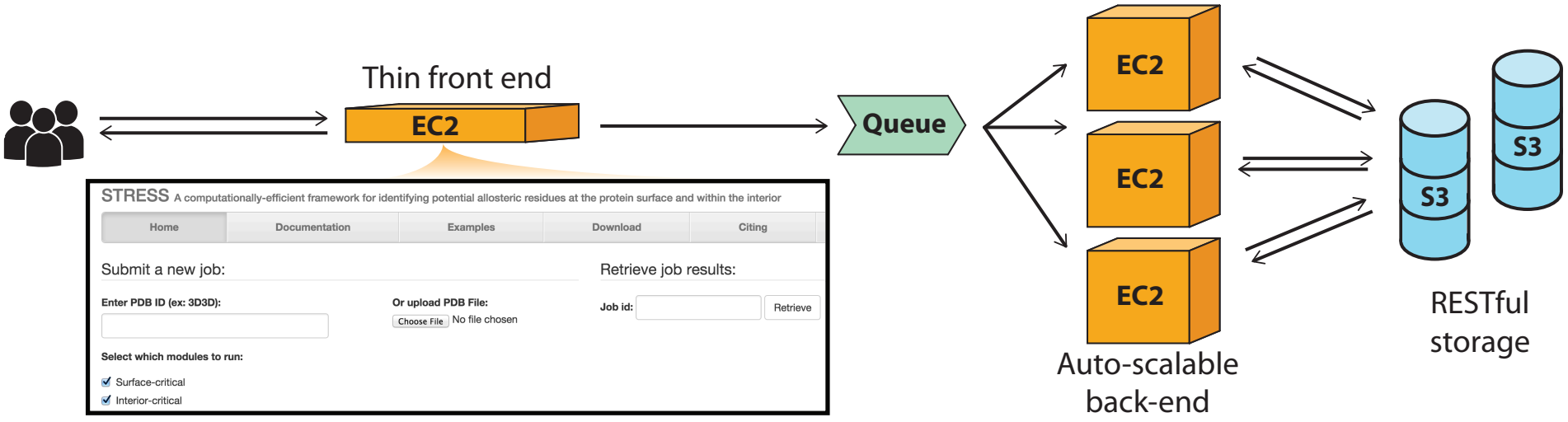
PDB: 4AKE

Output

```
11   LYS A
25   GLU A
136  ASP A
204  ASN A
.
.
```

STRESS Server Architecture

stress.molmovdb.org



STRESS Server Architecture: Highlights
stress.molmovdb.org

Users submit a PDB, and the output is the set of predicted allosteric residues.

Running times are minimized by using a scalable server architecture that runs on the Amazon cloud. A light front-end server handles incoming requests, and powerful back-end servers perform the algorithmic calculations.

Amazon's Elastic Beanstalk enables dynamic scalability. Auto Scaling adjusts the number of back-end servers as needed.

Elastic Load Balancer automatically distributes incoming network traffic, ensuring that it can handle varying levels of demand.

Input and output files are stored remotely in an S3 bucket, which is accessible to each server via RESTful conventions.

By optimizing for speed (with optimizations introduced through changes in the workflow, data structures, numerical arithmetic, etc.), a typical case takes ~30 minutes on a E5-2660 v3 (2.60GHz) core.

Source code is available through Github: github.com/gersteinlab/STRESS