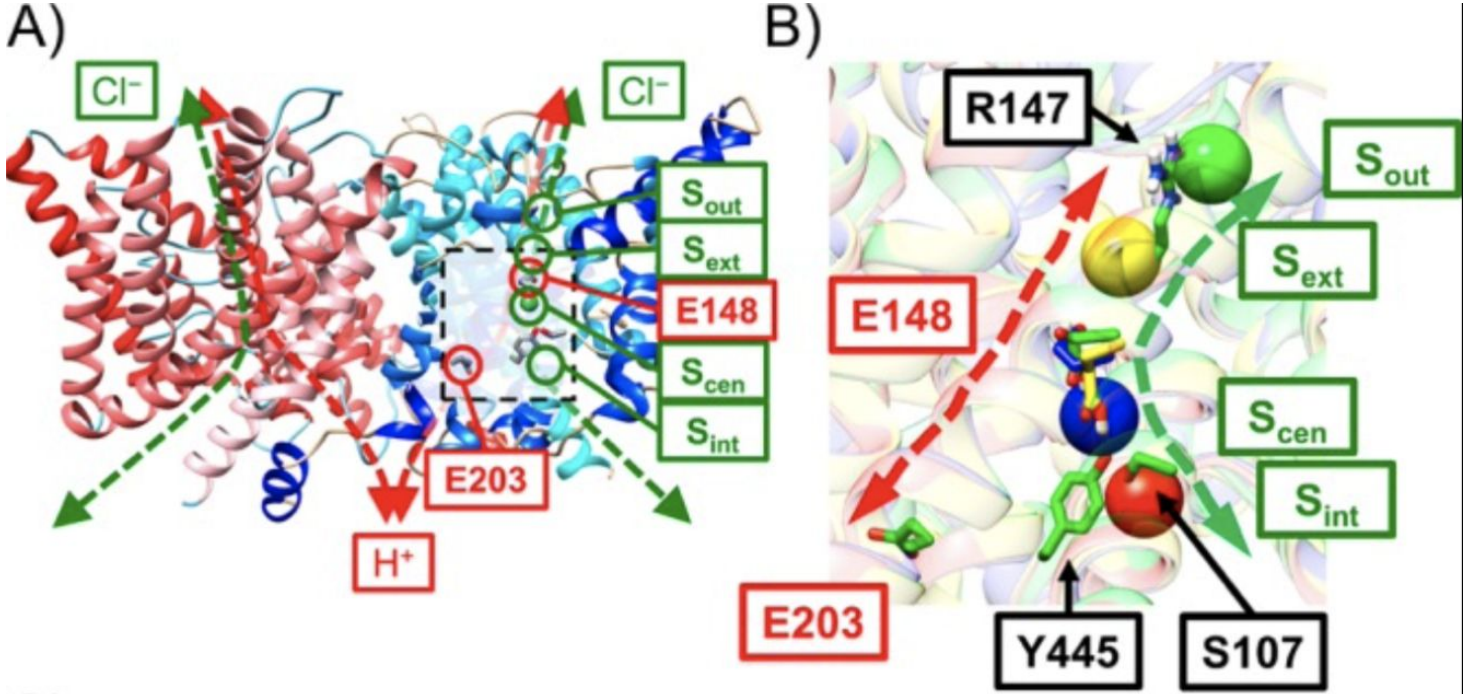
Current MKM

Check notes section for explanation for each slide. There is multiple sections that could be useful. Since we do not have a lot of time left, I thought it might be helpful to show you a bit more than you had originally asked for

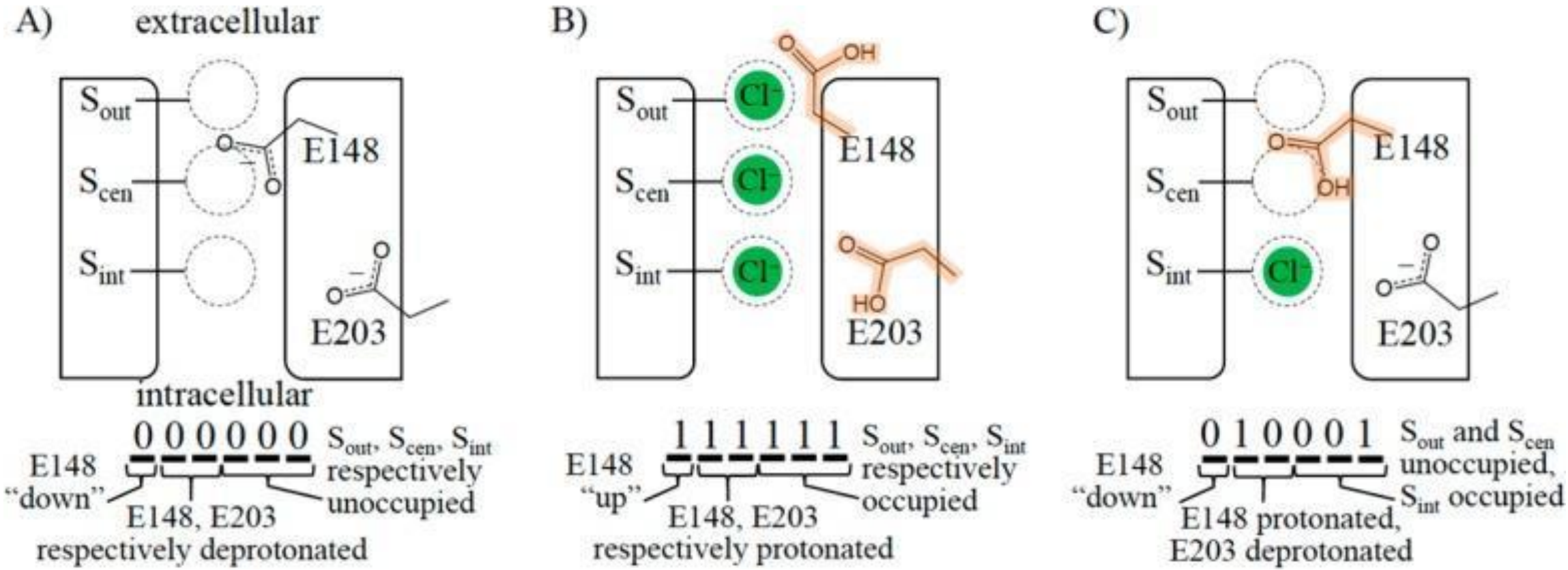
-Hannah

What is ClC-ec1?



- Prokaryotic homolog to ClC-3, ClC-5, ClC-7 and ClC-9
- Humans: Acidifies lysozyme, perisomes, inner mitochondrial matrix
- E.coli: primary role is in acid response

Mayes, Lee, Voth, Swanson, *JACS.*, 140, 1793-1804 (2018)


A)
extracellular
S_{out}
S_{cen}
S_{int}
E148
E203
intracellular
0 0 0 0 0 0
S_{out}, S_{cen}, S_{int} respectively unoccupied
E148 "down"
E148, E203 respectively deprotonated
B)
S_{out}
S_{cen}
S_{int}
Cl^-
OH
HO
E148
E203
1 1 1 1 1 1
S_{out}, S_{cen}, S_{int} respectively occupied
E148 "up"
E148, E203 respectively protonated
C)
S_{out}
S_{cen}
S_{int}
Cl^-
OH
E148
E203
0 1 0 0 0 1
S_{out} and S_{cen} unoccupied, S_{int} occupied
E148 "down"
E148 protonated, E203 deprotonated

How was the MKM created

Set this to 0

Solve for this

Probability Matrix Generation

Rate Matrix $\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}$

$\begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \lambda_n \end{bmatrix}$

Master Kinetic equation

$$dp_i(t)/dt = \sum_{j=1}^{M} k_{ij} p_j(t) - \sum_{i=1}^{N} k_{ji} p_i(t)$$

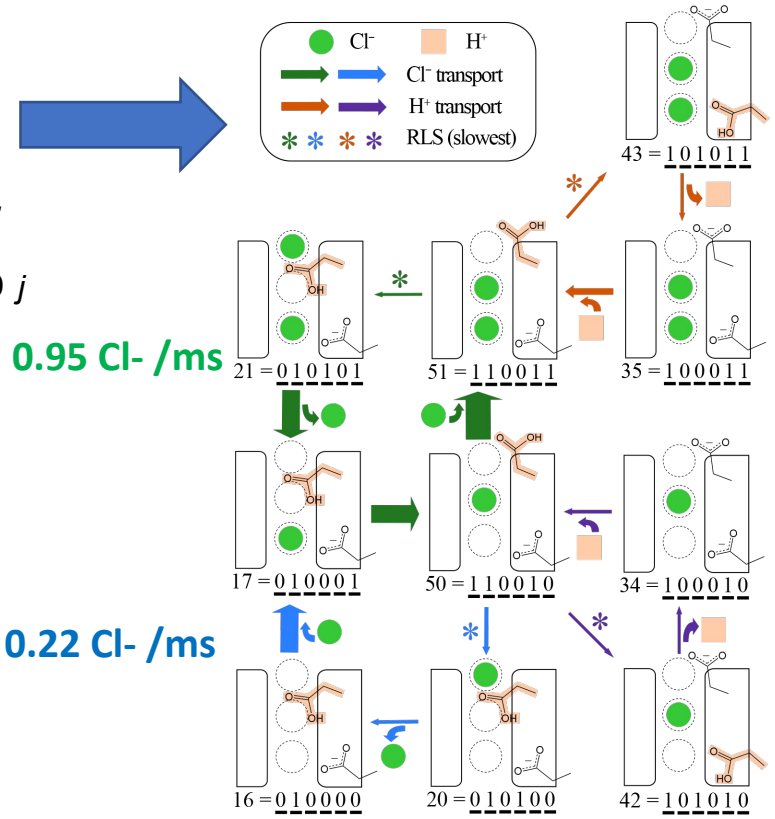
$p_i(t)$: the population of the state i
k_{ij}: the rate constant of the in-flux from j to i
k_{ji}: the rate constant of the out-flux from i to j

Flux Equation

$$\boldsymbol{F_{ij} = P_i k_{ij}}$$

F_{ij}: *The flux going through transition i to j*
$p_i(t)$: the population of the state i
k_{ij}: the rate constant of the in-flux from j to i

Cyclic Decomposition



0.95 Cl- /ms

0.22 Cl- /ms

Mayes, Lee, Voth, Swanson, *JACS.*, 140, 1793-1804 (2018)

Generation of the associated matrix

Step 4. Update the known coefficient matrix

Copy the respective transition rate coefficient to the ‘associated’ matrix (self.As) which maps out each rate coefficient for each possible transition in a 64x64 matrix

$$\begin{bmatrix} k_{1,1} & \cdots & k_{1,64} \\ \vdots & \cdots & \vdots \\ k_{64,1} & \cdots & k_{64,64} \end{bmatrix}$$

```
# Produces rate matrices in a list with orientations
# biological (self.coeff_mats[0]) and opposite (self.coeff_mats[1])
# And associated solution matrices in self.As
def update_coeff_mats(self):
    np.fill_diagonal(self.coeff_mats[0], 0)
    np.fill_diagonal(self.coeff_mats[1], 0)
    np.fill_diagonal(self.coeff_mats_A[0], 0)
    np.fill_diagonal(self.coeff_mats_A[1], 0)
    for j in range(self.N_coeffs): #for all of the rate_coefficients
        self.update_expanded_coeff(j)
        for t in self.transitions[j]: #for each tuple in transition matrix
            for k in range(self.N_orient):
                self.coeff_mats[k][t[1],t[0]]  = self.expanded_coeff[k]
                self.coeff_mats[k][t[0],t[0]] -= self.expanded_coeff[k]
                self.coeff_mats_A[k][t[1], t[0]] = self.expanded_coeff_A[k]
                self.coeff_mats_A[k][t[0], t[0]] -= self.expanded_coeff_A[k]
    for k in range(self.N_orient):
        np.copyto(self.As[k,:-1,:],self.coeff_mats[k,:-1,:])
        np.copyto(self.As_A[k, :-1, :], self.coeff_mats_A[k, :-1, :])
```

How do we calculate flow and the populations

Diagonalization of the associated state coefficient matrix (self.As) to get the steady state populations

```
# Updates self.ss_pops using self.coeffs via 1 iteration
def iterate_ss_pops(self):
    for j in range(self.N_orient):
        try:
            self.ss_pops[:,j] = np.ravel(
                np.linalg.solve(self.As[j],self.b))
            self.ss_pops_A[:, j] = np.ravel(
                np.linalg.solve(self.As_A[j], self.b_A))
        except np.linalg.LinAlgError as e:
            if str(e) == 'Singular matrix':
                print('Warning: Singular matrix encountered.
                self.ss_pops[:,j] = float('NaN')
```

Set this to 0

Solve for this

Master Kinetic equation

$$dp_i(t)/dt = \sum_{j=1}^{M} k_{ij} p_j(t) - \sum_{i=1}^{N} k_{ji} p_i(t)$$

$p_i(t)$: the population of the state i
k_{ij}: the rate constant of the in-flux from j to i
k_{ji}: the rate constant of the out-flux from i to j

$$X = A^{-1}B$$

$$\begin{bmatrix} k_{1,1} & \dots & k_{1,64} \\ \vdots & \dots & \vdots \\ k_{64,1} & \dots & k_{64,64} \end{bmatrix}^{-1} \begin{vmatrix} 0 \\ \vdots \\ 1 \end{vmatrix} = \begin{vmatrix} P_1(t) \\ \vdots \\ P_{64}(t) \end{vmatrix}$$

How do we calculate flow and the populations

Calculate the flow matrix as :

$$flux_{ij} = \frac{k_{ij}P_j}{2}$$

Calculate the total ion flows as a sum of each transition flow

$$Flux_{Cl-}^{Biological} = \sum (Flux_{ij}^{Biological} - Flux_{ji}^{Biological})$$

Calculate the total flow as a sum of orientations

$$Flux_{Cl-}^{Total} = Flux_{Cl-}^{Biological} + Flux_{Cl-}^{Opposite}$$

```
# Updates self.flow_mats using self.ss_pops
def update_flow_mats(self):
    np.copyto(self.flow_mats,self.coeff_mats)
    np.copyto(self.flow_mats_A,self.coeff_mats_A)
    for j in range(self.N_orient):
        np.fill_diagonal(self.flow_mats[j], 0)
        np.fill_diagonal(self.flow_mats_A[j], 0)
        for k in range(self.N_states):
            self.flow_mats[j,:,k] *= self.ss_pops[k,j] /2
            self.flow_mats_A[j, :, k] *= self.ss_pops_A[k, j] /2
```

```
# Produces orientational ion flows with format
# [bio_flows,opp_flows] with subformat
# [cl_flow,h_flow] with positive flows corresponding
# to flow directed from extracellular to intracellular
def update_ion_flows(self):
    for j in range(self.N_orient):
        for k in range(2):
            self.ion_flows[j,k] = 0
            self.ion_flows_A[j,k] = 0
    for t in self.flow_transitions:
        for k in range(2):
            self.ion_flows[0, t[2]] += (t[3] * self.flow_mats[0][t[1], t[0]]) #[0,
            self.ion_flows[1, t[2]] -= (t[3] * self.flow_mats[1][t[1], t[0]])
            self.ion_flows_A[0, t[2]] += (t[3] * self.flow_mats_A[0][t[1], t[0]])
            self.ion_flows_A[1, t[2]] -= (t[3] * self.flow_mats_A[1][t[1], t[0]])
```

Conditions on ODEs/Population

1. E148 cannot be down with Scen occupied
2. E148 has to be protonated to allow for transfer between Scen and Sout
    a. Does not matter for Sint to Scen
3. E148up can transfer to E203 down but must have Scen or Sout to facilitate
    a. E148 down can transfer to E203 with no stipulations
4. E203 can only transfer to E148up when Scen is occupied
    a. E203 can transfer to E148 down with no stipulations
    b. E203 cannot transfer to E148 up with Sout occupied, needs to be down
5. Transfer to the Scen site requires that E148 is up.