Modelling intracellular cascades and neural plasticity

Simula summer school

27.6.2018

Tuomo Mäki-Marttunen

Outline

- Brief introduction to LTP/LTD
- Modeling approaches
- Formulation of the mass-action laws
- Simulation in NEURON’s RxD extension

Background

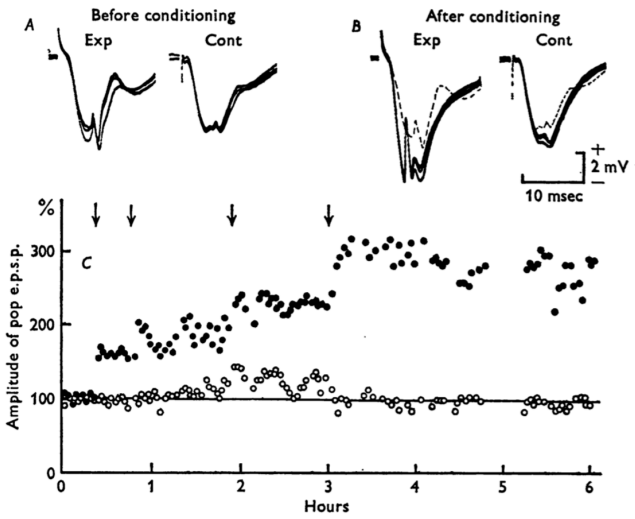
- W. James (1890): a modern perspective for the problem of plasticity
- S.R. y Cajal (1894): formation of new connections needed for learning
- Hebb's rule (1949): "neurons that fire together wire together"
- Bliss & Lomø (1973): LTP discovered
- Ito et al. (1982): LTD discovered
- Bhalla & Yiengar (1999): Unified modeling framework for many underlying pathways

J. Physiol. (1973), **232**, *pp.* 331–356
With 12 text-figures
Printed in Great Britain

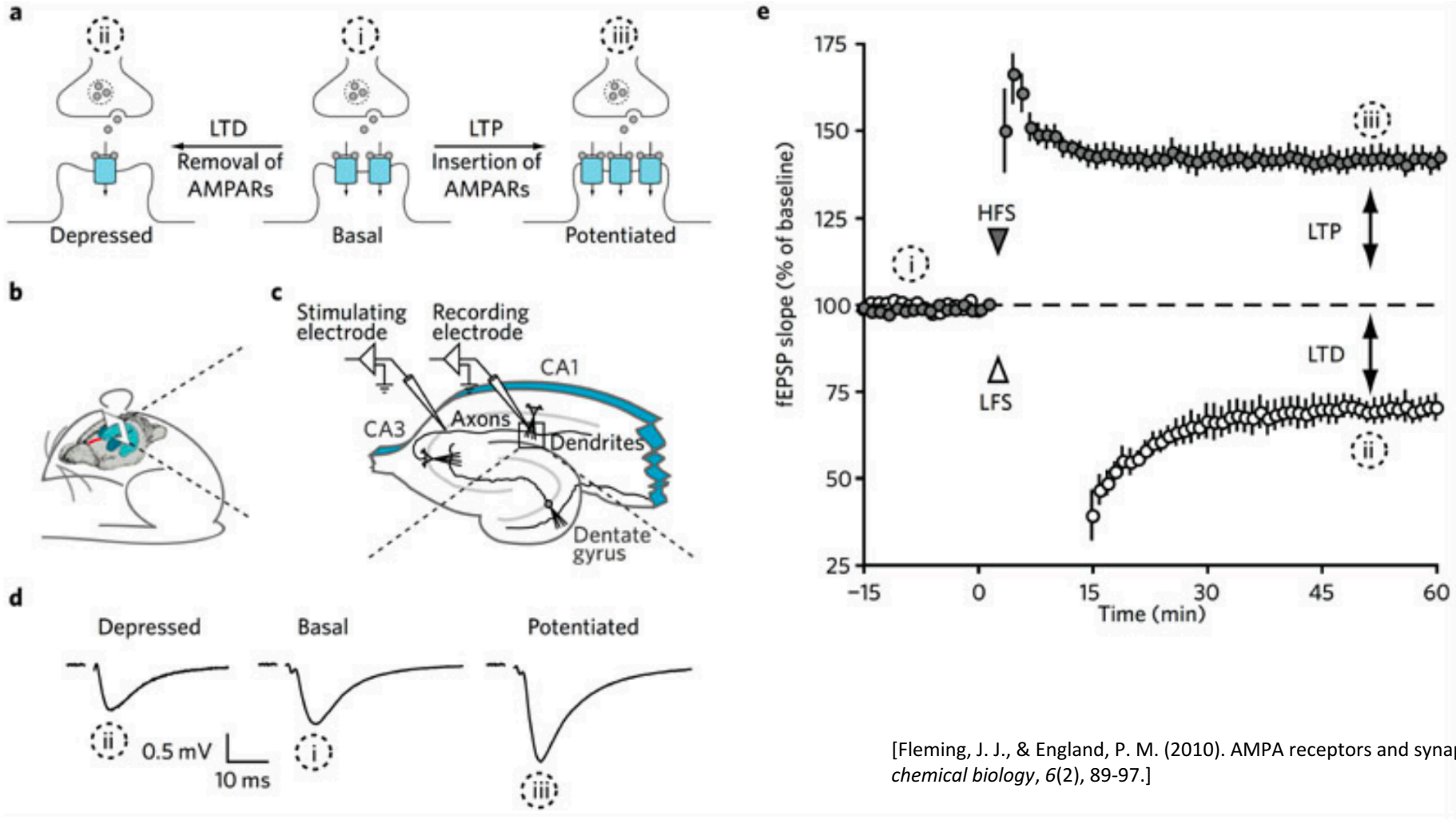
LONG-LASTING POTENTIATION OF SYNAPTIC TRANSMISSION IN THE DENTATE AREA OF THE ANAESTHETIZED RABBIT FOLLOWING STIMULATION OF THE PERFORANT PATH

By T. V. P. BLISS and T. LØMO
From the National Institute for Medical Research, Mill Hill, London NW7 1AA and the Institute of Neurophysiology, University of Oslo, Norway

(*Received 12 February 1973*)

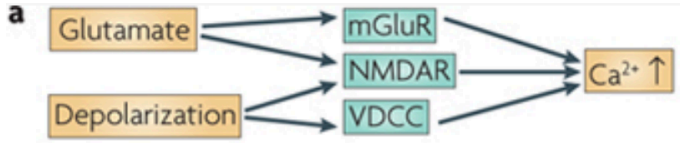


Long-term synaptic plasticity



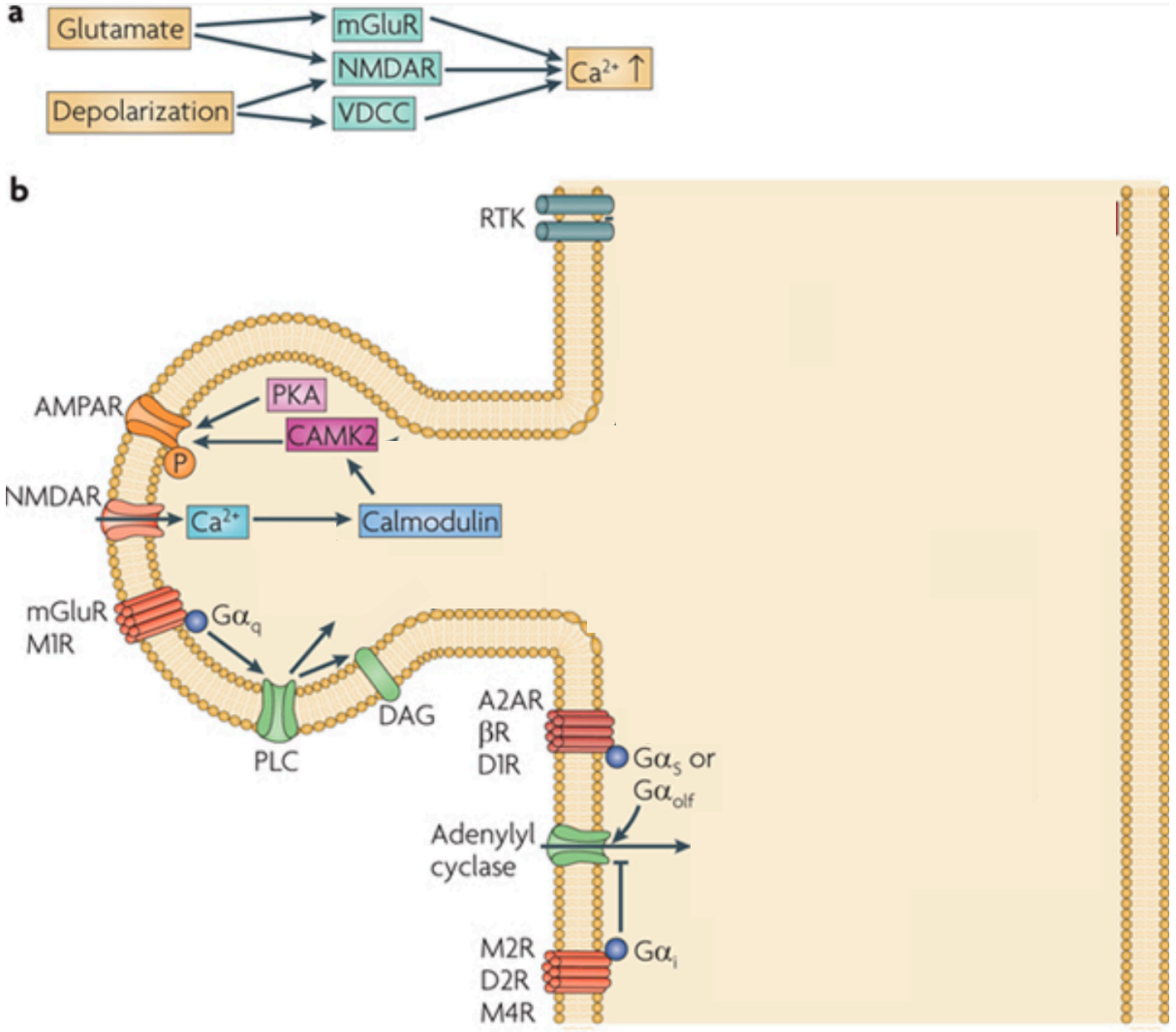
[Fleming, J. J., & England, P. M. (2010). AMPA receptors and synaptic plasticity: a chemist's perspective. *Nature chemical biology*, *6*(2), 89-97.]

Long-term synaptic plasticity: cellular mechanisms



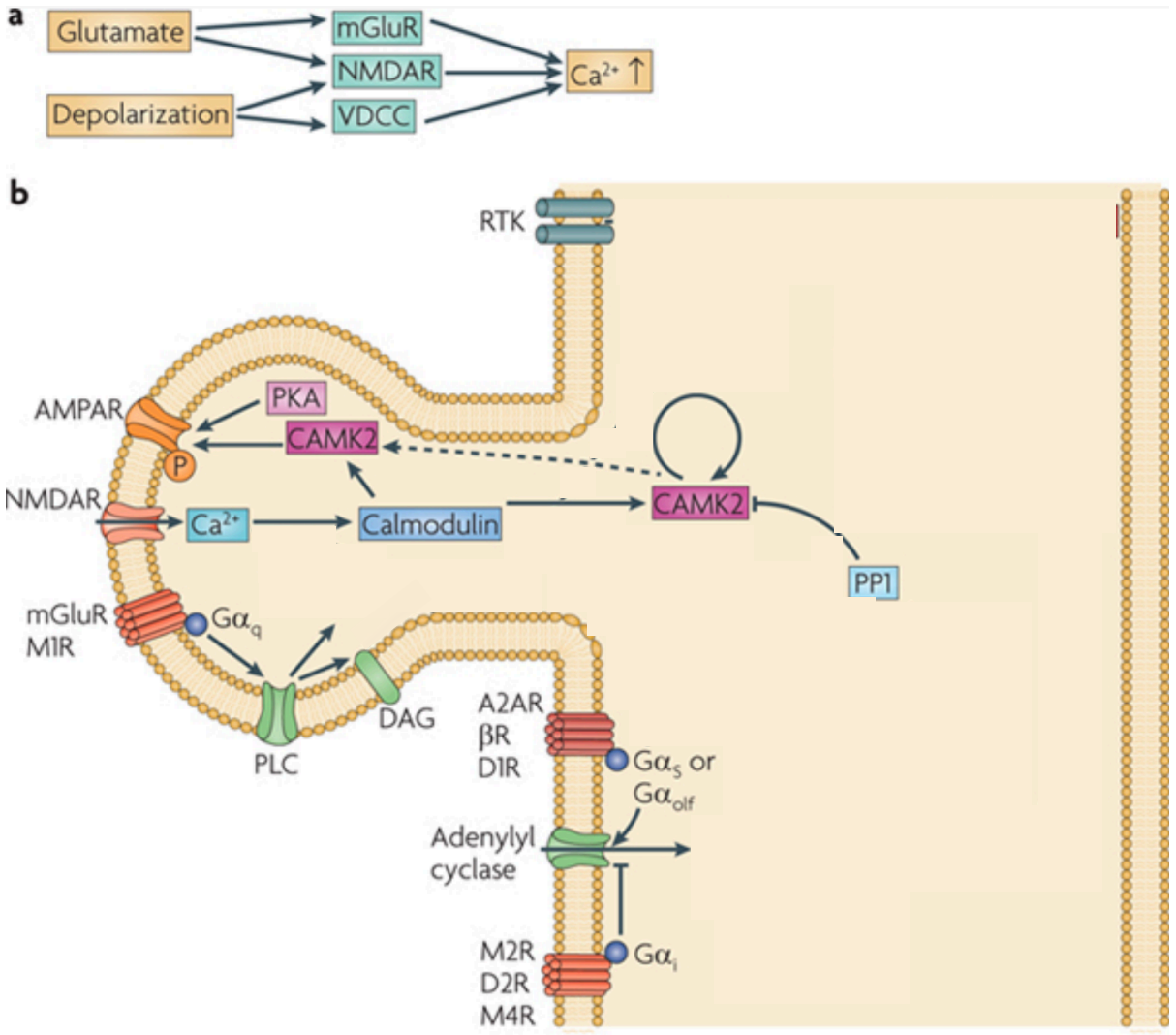
[Kotaleski, J. H., & Blackwell, K. T. (2010). Modelling the molecular mechanisms of synaptic plasticity using systems biology approaches. *Nature Reviews Neuroscience*, *11*(4), 239-251.]

Long-term synaptic plasticity: cellular mechanisms



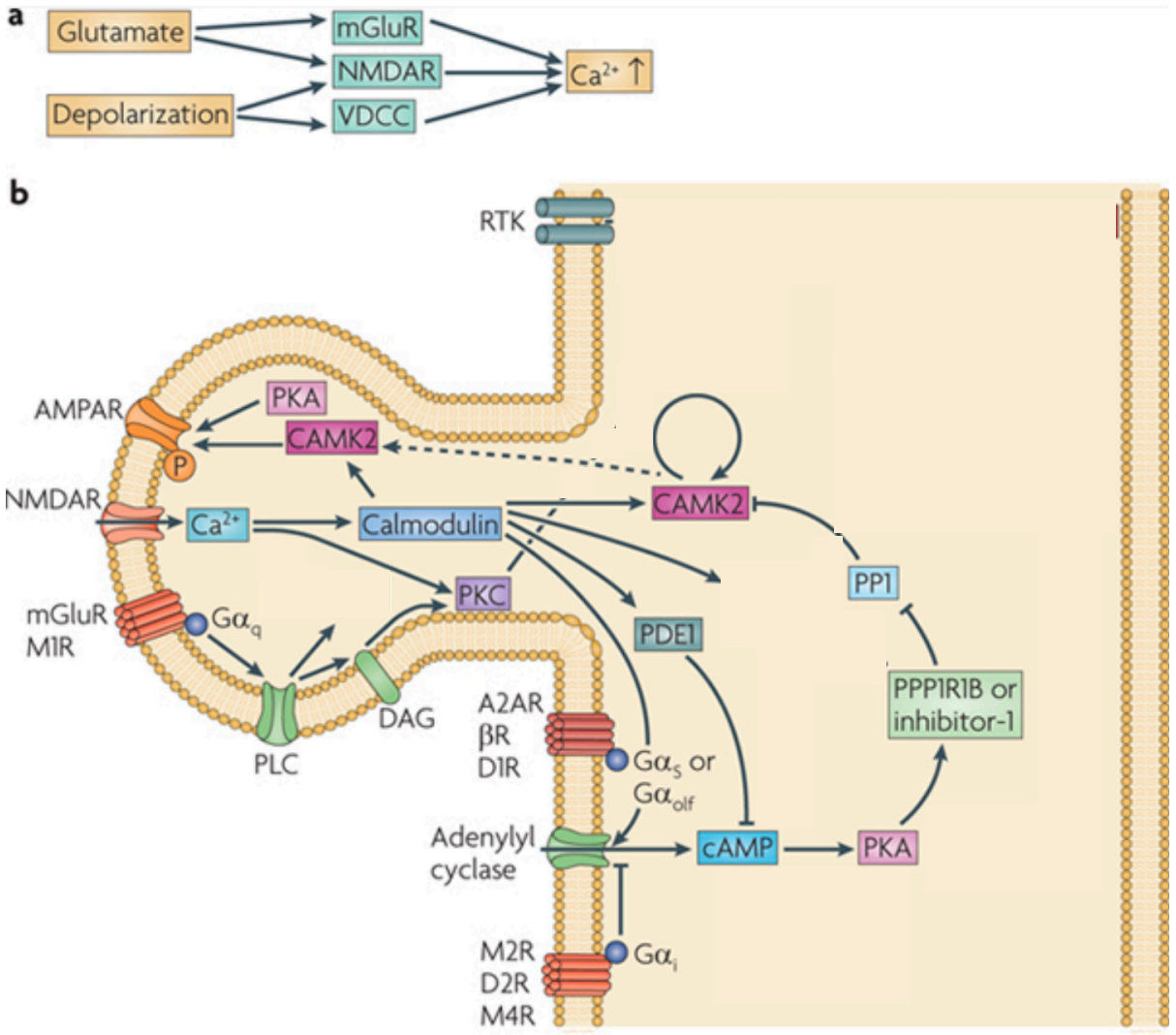
[Kotaleski, J. H., & Blackwell, K. T. (2010). Modelling the molecular mechanisms of synaptic plasticity using systems biology approaches. *Nature Reviews Neuroscience*, *11*(4), 239-251.]

Long-term synaptic plasticity: cellular mechanisms



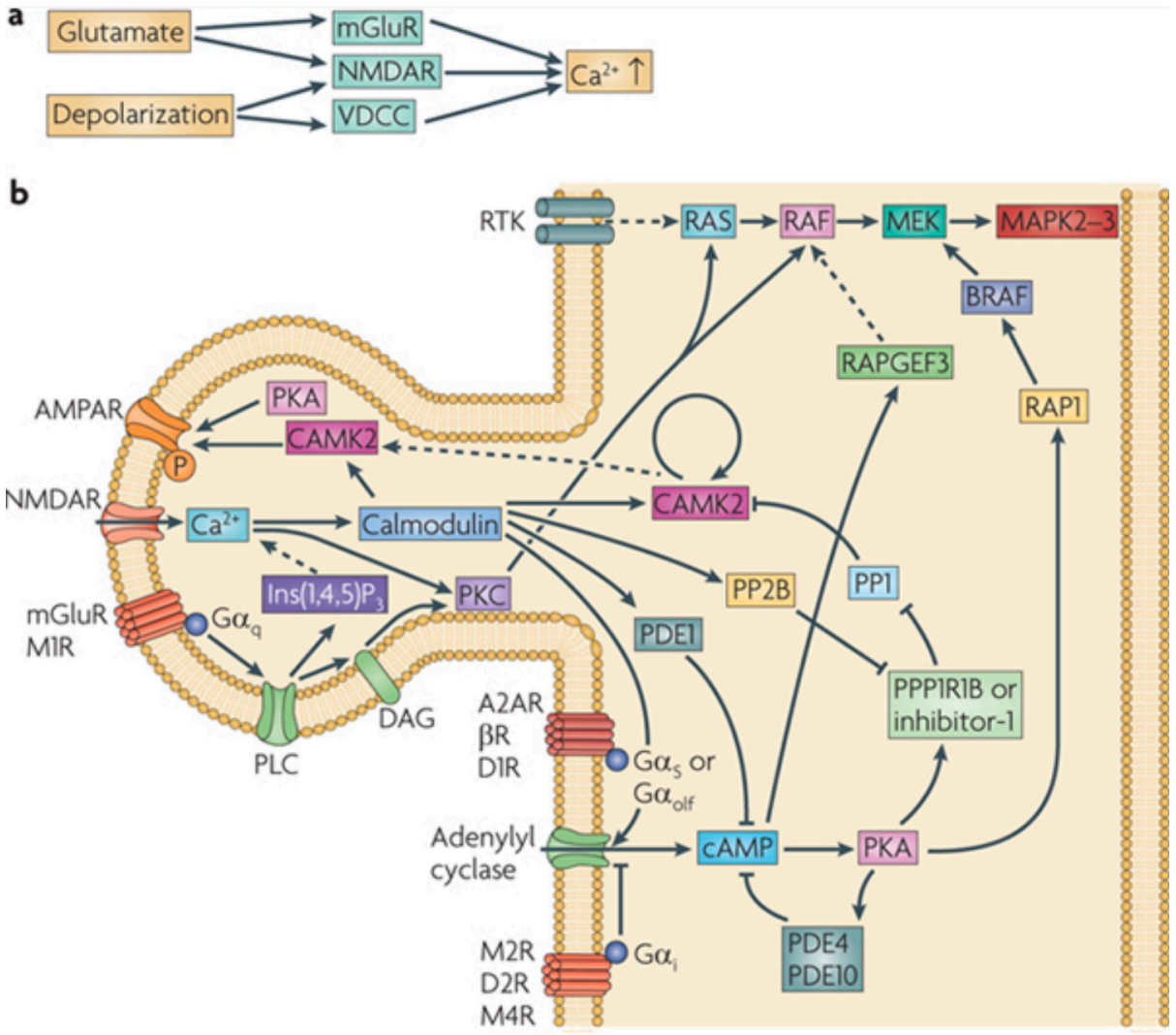
[Kotaleski, J. H., & Blackwell, K. T. (2010). Modelling the molecular mechanisms of synaptic plasticity using systems biology approaches. *Nature Reviews Neuroscience*, *11*(4), 239-251.]

Long-term synaptic plasticity: cellular mechanisms



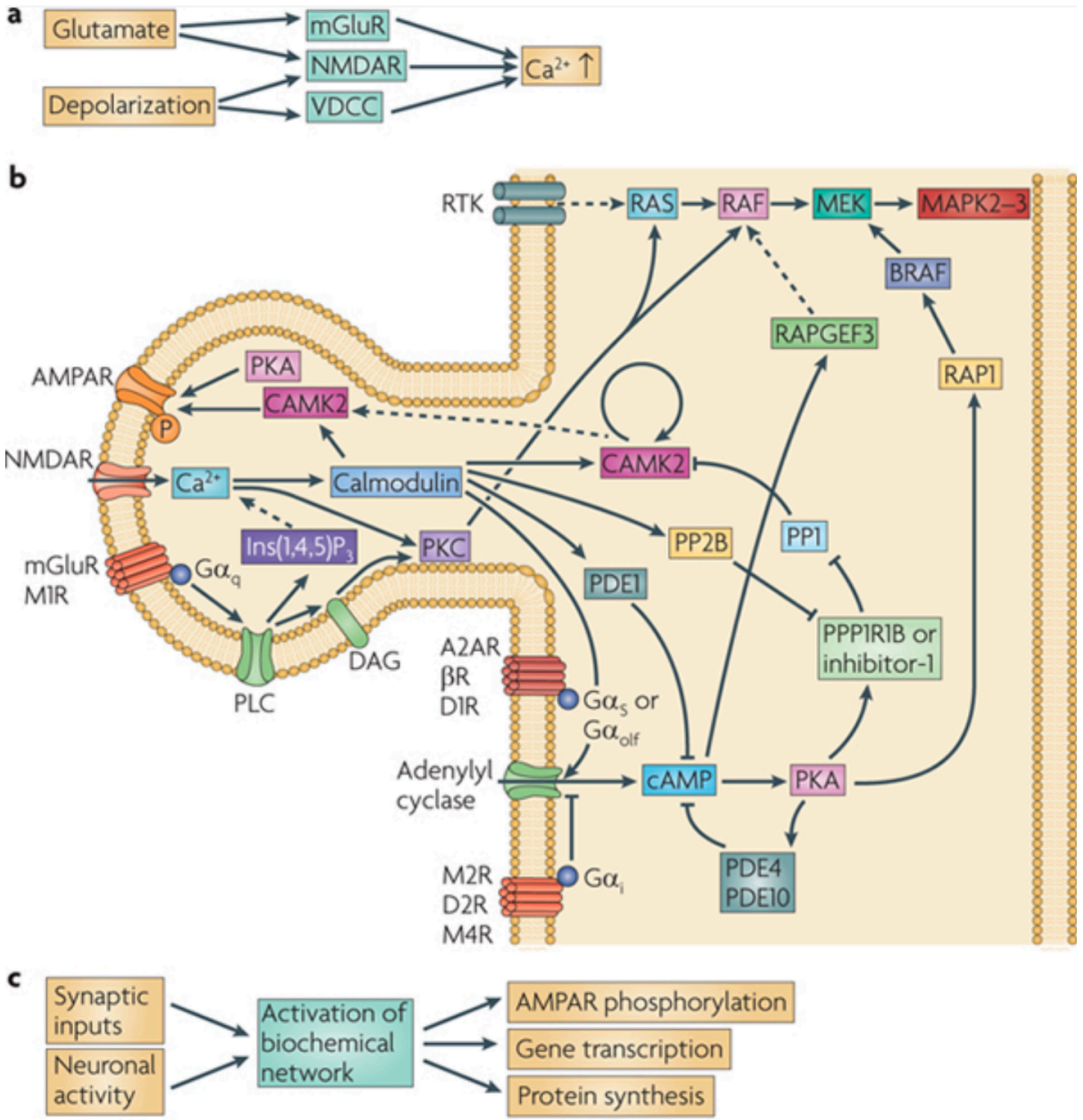
[Kotaleski, J. H., & Blackwell, K. T. (2010). Modelling the molecular mechanisms of synaptic plasticity using systems biology approaches. *Nature Reviews Neuroscience*, *11*(4), 239-251.]

Long-term synaptic plasticity: cellular mechanisms



[Kotaleski, J. H., & Blackwell, K. T. (2010). Modelling the molecular mechanisms of synaptic plasticity using systems biology approaches. *Nature Reviews Neuroscience*, *11*(4), 239-251.]

Long-term synaptic plasticity: cellular mechanisms



[Kotaleski, J. H., & Blackwell, K. T. (2010). Modelling the molecular mechanisms of synaptic plasticity using systems biology approaches. *Nature Reviews Neuroscience*, *11*(4), 239-251.]

Modeling approaches

- Almost all models based on the law of mass action

$$A + B \underset{k_b}{\overset{k_f}{\rightleftharpoons}} C + D$$

- Deterministic models: $d[A]/dt = k_b[\mathrm{C}][\mathrm{D}] - k_f[\mathrm{A}][\mathrm{B}]$
- Stochastic models
  - Gillespie's algorithm and its variants: at each iteration, sample the type and time of the next reaction in each voxel
  - Particle-based models: simulate movement of each particle separately, apply reactions when collisions

Modeling approaches

- Almost all models based on the law of mass action

$$A + B \underset{k_b}{\overset{k_f}{\rightleftharpoons}} C + D$$

- Deterministic models: $d[A]/dt = k_b[\mathrm{C}][\mathrm{D}] - k_f[\mathrm{A}][\mathrm{B}]$ COPASI NEURON
- Stochastic models
  - Gillespie's algorithm and its variants: at each iteration, sample the type and time of the next reaction in each voxel STEPS NeuroRD
  - Particle-based models: simulate movement of each particle separately, apply reactions when collisions MCell

Modeling approaches

- Almost all models based on the law of mass action

$$A + B \underset{k_b}{\overset{k_f}{\rightleftharpoons}} C + D$$

- Deterministic models: $d[A]/dt = k_b[\mathrm{C}][\mathrm{D}] - k_f[\mathrm{A}][\mathrm{B}]$

COPASI NEURON

- Stochastic models
  - Gillespie's algorithm and its variants: at each iteration, sample the type and time of the next reaction in each voxel

STEPS NeuroRD

  - Particle-based models: simulate movement of each particle separately, apply reactions when collisions

MCell

Challenges

- Model construction
  - Reaction rates usually cannot be fitted to data from the modeled system
  - Difficult to determine the concentrations of the species
  - Large variability in the reproduced phenomena

- Model simulation
  - Multiple spatial and temporal scales
  - Model interaction with HH-type models often not considered

Differential equations from reactions

- Which of these diff. equations describes the following reaction?

$$A \underset{k_b}{\overset{k_f}{\longleftrightarrow}} B$$

1) $\frac{d[A]}{dt} = k_b[B], \frac{d[B]}{dt} = k_f[A]$

2) $\frac{d[A]}{dt} = \frac{k_f}{k_b}\frac{[A]}{[B]}, \frac{d[B]}{dt} = \frac{k_b}{k_f}\frac{[B]}{[A]}$

3) $\frac{d[A]}{dt} = k_f[A] - k_b[B], \frac{d[B]}{dt} = k_b[B] - k_f[A]$

4) $\frac{d[A]}{dt} = -k_f[A] + k_b[B], \frac{d[B]}{dt} = k_f[A] - k_b[B]$

Differential equations from reactions

- Which of these diff. equations describes the following reaction?

$$A + B \underset{k_b}{\overset{k_f}{\longleftrightarrow}} 2A$$

1) $\frac{d[A]}{dt} = k_f[A][B] - 2k_b[A]^2, \frac{d[B]}{dt} = -k_f[A][B]$

2) $\frac{d[A]}{dt} = \frac{1}{2}k_f[A][B] - k_b[A]^2, \frac{d[B]}{dt} = -k_f[A][B]$

3) $\frac{d[A]}{dt} = k_f[A][B] - k_b[A]^2, \frac{d[B]}{dt} = -k_f[A][B]$

4) $\frac{d[A]}{dt} = 2k_f[A][B] - k_b[A]^2, \frac{d[B]}{dt} = -k_f[A][B]$

Differential equations from reactions

- Which of these diff. equations describes the following reaction?

$$A + B \underset{k_b}{\overset{k_f}{\longleftrightarrow}} AB \xrightarrow{k_{cat}} A + C$$

1) $\frac{d[A]}{dt} = -k_f[A][B] + k_b[AB] + k_{cat}[AB], \frac{d[B]}{dt} = -k_f[A][B], \frac{d[AB]}{dt} = k_f[A][B] - k_{cat}[AB], \frac{d[C]}{dt} = k_{cat}[AB]$

2) $\frac{d[A]}{dt} = -k_f[A][B] + k_b[AB] + k_{cat}[AB], \frac{d[B]}{dt} = -k_f[A][B] + k_b[AB], \frac{d[AB]}{dt} = k_f[A][B] - k_b[AB] - k_{cat}[AB], \frac{d[C]}{dt} = k_{cat}[AB]$

3) $\frac{d[A]}{dt} = k_b[AB] + k_{cat}[AB], \frac{d[B]}{dt} = k_b[AB], \frac{d[AB]}{dt} = k_f[A][B] - k_{cat}[AB], \frac{d[C]}{dt} = k_{cat}[AB]$

4) $\frac{d[A]}{dt} = -k_f[A][B] + k_b[AB] + k_{cat}[AB], \frac{d[B]}{dt} = -k_f[A][B] + k_b[AB], \frac{d[AB]}{dt} = -k_b[AB] + k_{cat}[A][C], \frac{d[C]}{dt} = k_{cat}[A][C]$

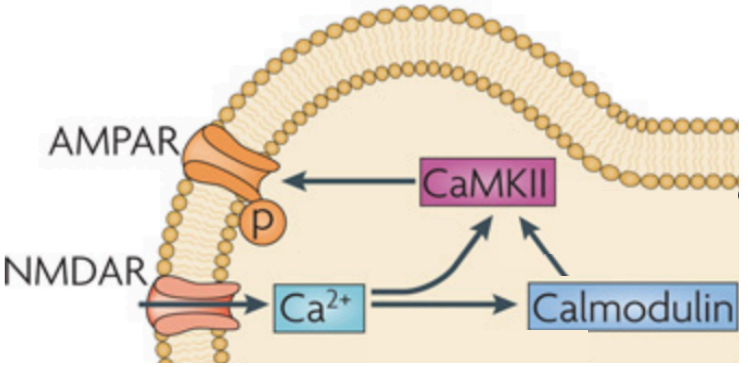
Differential equations from reactions

- Which of these diff. equations describes the following reaction?

$$A + B \underset{k_b}{\overset{k_f}{\longleftrightarrow}} AB \xrightarrow{k_{cat}} A + C$$

2) $\frac{d[A]}{dt} = -k_f[A][B] + k_b[AB] + k_{cat}[AB]$, $\frac{d[B]}{dt} = -k_f[A][B] + k_b[AB]$, $\frac{d[AB]}{dt} = k_f[A][B] - k_b[AB] - k_{cat}[AB]$, $\frac{d[C]}{dt} = k_{cat}[AB]$

Mass action law in LTP – CaMKII model



- Ca^{2+} enters through NMDA channels

$$\mathrm{Ca}_{(\mathrm{ext})} \longrightarrow \mathrm{Ca}$$

- Ca^{2+} binds with calmodulin

$$2\mathrm{Ca} + \mathrm{CaM} \rightleftharpoons \mathrm{CaMCa}_2$$
$$2\mathrm{Ca} + \mathrm{CaMCa}_2 \rightleftharpoons \mathrm{CaMCa}_4$$

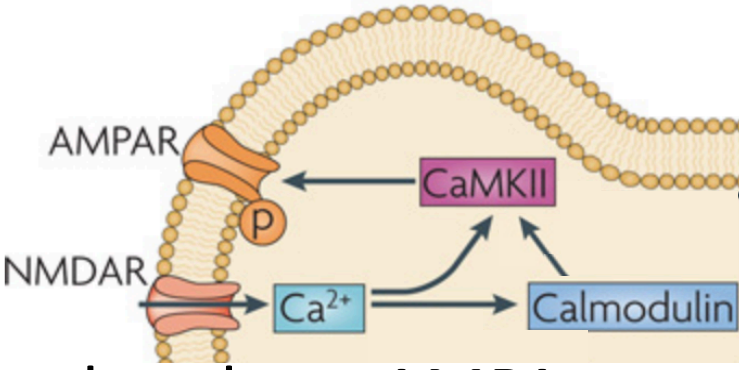
- Ca^{2+}-bound calmodulin binds to CaMKII

$$\mathrm{CaMKII} + \mathrm{CaMCa}_4 \rightleftharpoons \mathrm{CaMKIICaMCa}_4$$

- Calmodulin-bound CaMKII is autophosphorylated

$$\mathrm{CaMKIICaMCa}_4 + \mathrm{CaMKIICaMCa}_4 \rightleftharpoons \mathrm{Complex}$$
$$\mathrm{Complex} + \mathrm{Complex} \rightleftharpoons \mathrm{Complex} + \mathrm{pComplex}$$
$$\mathrm{pCaMKIICaMCa}_4 + \mathrm{CaMKIICaMCa}_4 \rightleftharpoons \mathrm{pComplex}$$

Mass action law in LTP – CaMKII model



- Phosphorylated calmodulin-bound CaMKII phosphorylates AMPA receptors

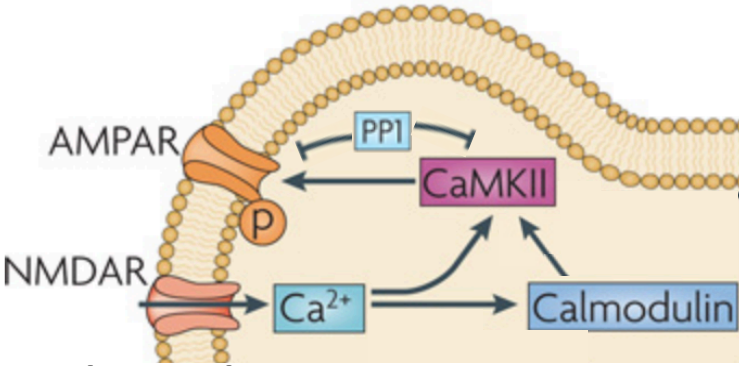
$$\mathrm{GluR1} + \mathrm{pCaMKIICaMCa_4} \rightleftharpoons \mathrm{GluR1\_pCaMKIICaMCa_4} \longrightarrow \mathrm{GluR1\_pS831} + \mathrm{pCaMKIICaMCa_4}$$

- Phosphorylated AMPA receptors have higher ion-channel conductance and are more easily inserted to the membrane

Jędrzejewska-Szmek, Joanna, et al. "β-adrenergic signaling broadly contributes to LTP induction." *PLoS computational biology* 13.7 (2017): e1005657.

Carroll, Reed C., et al. "Role of AMPA receptor endocytosis in synaptic plasticity." *Nature Reviews Neuroscience* 2.5 (2001): 315.

Mass action law in LTP – CaMKII model



- Phosphorylated calmodulin-bound CaMKII phosphorylates AMPA receptors

$$\mathrm{GluR1} + \mathrm{pCaMKIICaMCa_4} \rightleftharpoons \mathrm{GluR1\_pCaMKIICaMCa_4} \longrightarrow \mathrm{GluR1\_pS831} + \mathrm{pCaMKIICaMCa_4}$$

- Phosphorylated AMPA receptors have higher ion-channel conductance and are more easily inserted to the membrane
- AMPA receptors are dephosphorylated by phosphatases such as PP1

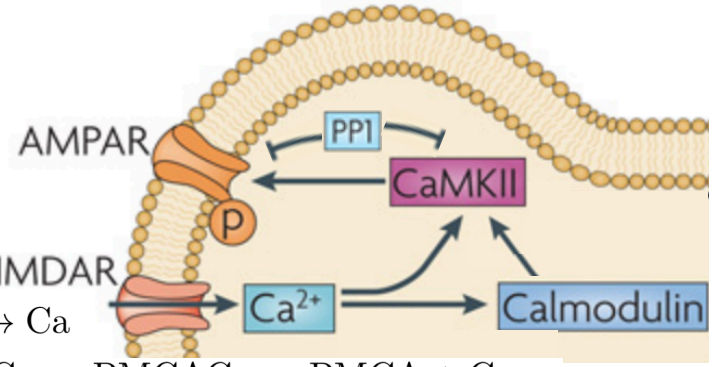
$$\mathrm{GluR1\_pS831} + \mathrm{PP1} \rightleftharpoons \mathrm{GluR1\_pS831\_PP1} \longrightarrow \mathrm{GluR1} + \mathrm{PP1}$$

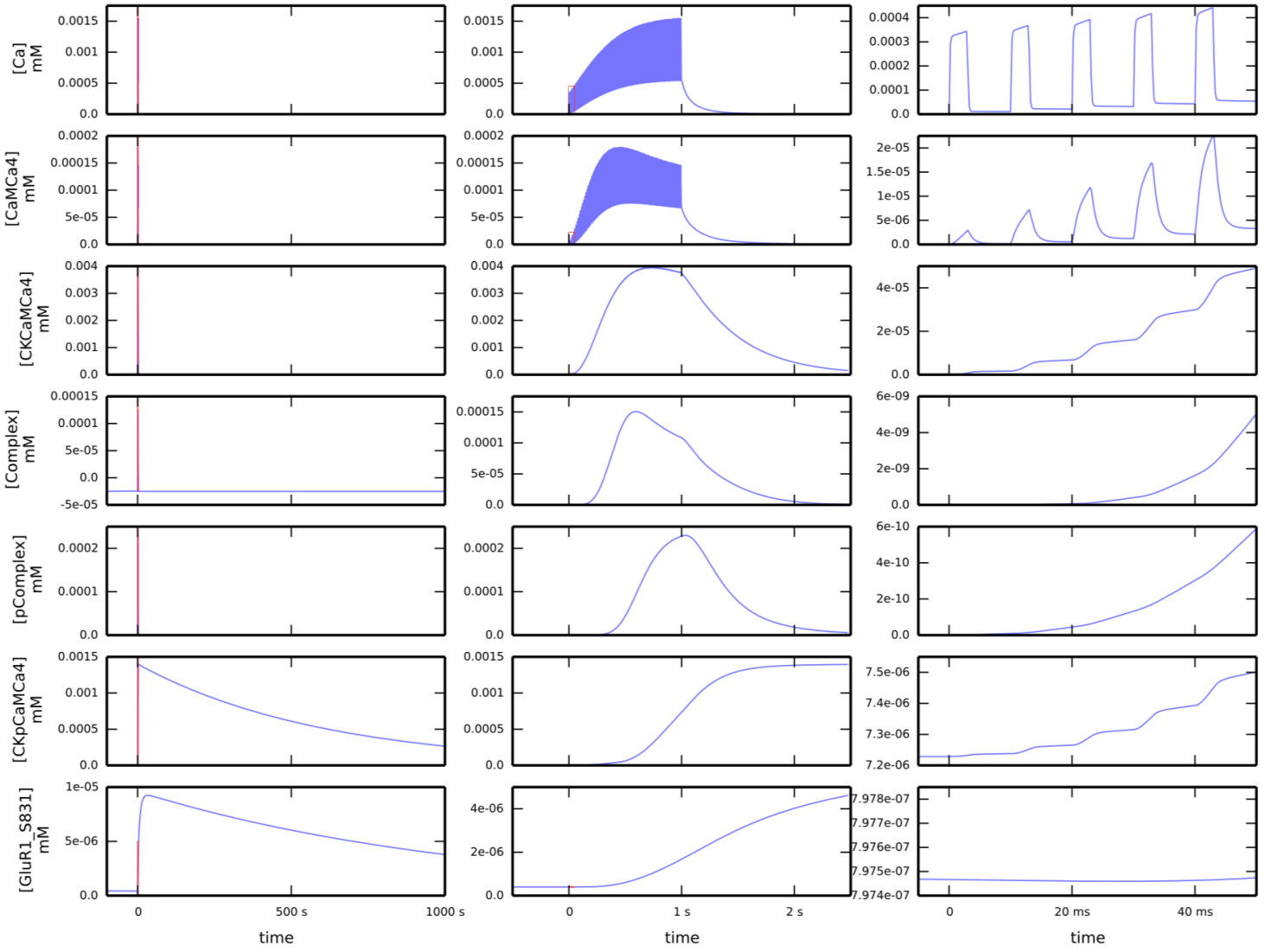
$$\mathrm{pCaMKIICaMCa} + \mathrm{PP1} \rightleftharpoons \mathrm{pCaMKIICaMCa_4PP1} \longrightarrow \mathrm{CaMKIICaMCa_4} + \mathrm{PP1}$$

Jędrzejewska-Szmek, Joanna, et al. "β-adrenergic signaling broadly contributes to LTP induction." *PLoS computational biology* 13.7 (2017): e1005657.

Carroll, Reed C., et al. "Role of AMPA receptor endocytosis in synaptic plasticity." *Nature Reviews Neuroscience* 2.5 (2001): 315.

Mass action law in LTP – CaMKII model





$$Ca_{(ext)} \longrightarrow Ca$$

$$PMCA + Ca \rightleftharpoons PMCACa \rightarrow PMCA + Ca_{out}$$
$$ncx + Ca \rightleftharpoons ncxCa \rightarrow ncx + Ca_{out}$$
$$Ca_{Out} + Leak \rightleftharpoons Ca_{Out}Leak \rightarrow Ca + Leak$$

$$2Ca + CaM \rightleftharpoons CaMCa_2$$
$$2Ca + CaMCa_2 \rightleftharpoons CaMCa_4$$

$$CaMKII + CaMCa_4 \rightleftharpoons CaMKIICaMCa_4$$

$$CaMKIICaMCa_4 + CaMKIICaMCa_4 \rightleftharpoons Complex$$

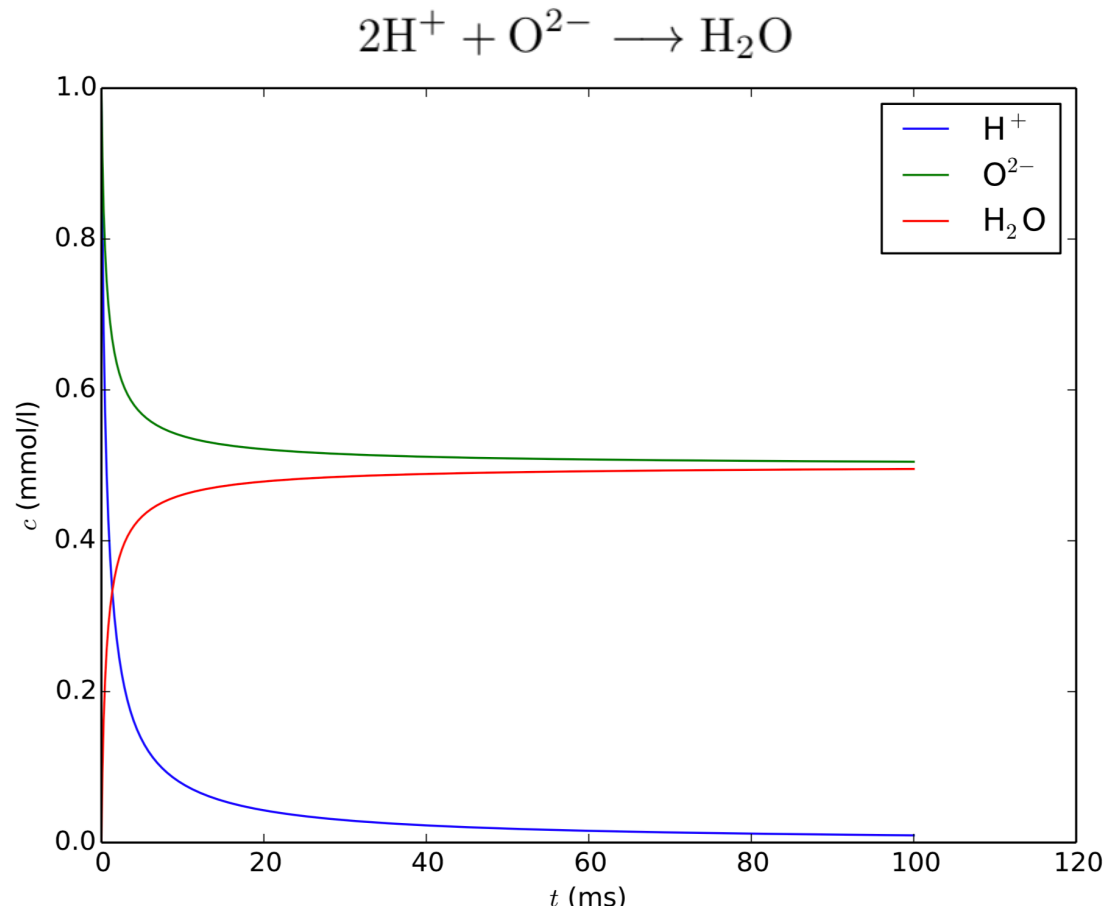
$$Complex + Complex \rightleftharpoons Complex + pComplex$$

$$pCaMKIICaMCa_4 + CaMKIICaMCa_4 \rightleftharpoons pComplex$$

$$GluR1 + pCaMKIICaMCa_4 \rightleftharpoons GluR1\_pCaMKIICaMCa_4 \longrightarrow GluR1\_pS831 + pCaMKIICaMCa_4$$
$$GluR1\_pS831 + PP1 \rightleftharpoons GluR1\_pS831\_PP1 \longrightarrow GluR1 + PP1$$

Examples for NEURON's RxD extension

```
from neuron import h, rxd
import time
from pylab import *

h.load_file('stdrun.hoc')

dend = h.Section()
r = rxd.Region(h.allsec())

hydrogen = rxd.Species(r, name='hydrogen',charge=1,initial=1)
oxygen = rxd.Species(r, name='oxygen',charge=-2,initial=1)
water = rxd.Species(r, name='water',charge=0,initial=0)

reaction = rxd.Reaction(2 * hydrogen + oxygen > water, 1)

vec_t = h.Vector()
vec_H = h.Vector()
vec_O = h.Vector()
vec_H2O = h.Vector()
vec_t.record(h._ref_t)
vec_H.record(hydrogen.nodes(dend)(0.5)[0].._ref_concentration)
vec_O.record(oxygen.nodes(dend)(0.5)[0].._ref_concentration)
vec_H2O.record(water.nodes(dend)(0.5)[0].._ref_concentration)

h.finitialize()
timenow = time.time()
h.continuerun(100)
print('Simulation done in '+str(time.time()-timenow)+' seconds')

f,ax = subplots(1,1)
ax.plot(array(vec_t),array(vec_H),label='H$^+$')
ax.plot(array(vec_t),array(vec_O),label='O$^{2-}$')
ax.plot(array(vec_t),array(vec_H2O),label='H$_2$O')
ax.legend()
ax.set_xlabel('$t$ (ms)')
ax.set_ylabel('$c$ (nmol/l)')

f.savefig('H2O.eps')
```



Examples for NEURON's RxD extension

```
from neuron import h, rxd
import time
from pylab import *

h.load_file('stdrun.hoc')

dend = h.Section()
r = rxd.Region(h.allsec())

hydrogen = rxd.Species(r, name='hydrogen',charge=1,initial=1)
oxygen = rxd.Species(r, name='oxygen',charge=-2,initial=1)
water = rxd.Species(r, name='water',charge=0,initial=0)

reaction = rxd.Reaction(2 * hydrogen + oxygen > water, 1)
```

Load python libraries

Load standard hoc functions that may be useful

Create a compartment called 'dend'

Create a region where reactions occur (include all sections, i.e. dend)

Create species for region r. Set initial concentrations 1 mM (H^+), 1 mM (O^{2-}), and 0 mM (H_2O)

Create reaction $2H^+ + O^{2-} \longrightarrow H_2O$ with rate constant 1 $(mM)^2/ms$

Examples for NEURON's RxD extension

```
vec_t = h.Vector()
vec_H = h.Vector()
vec_O = h.Vector()
vec_H2O = h.Vector()
vec_t.record(h._ref_t)
vec_H.record(hydrogen.nodes(dend)(0.5)[0]._ref_concentration)
vec_O.record(oxygen.nodes(dend)(0.5)[0]._ref_concentration)
vec_H2O.record(water.nodes(dend)(0.5)[0]._ref_concentration)

h.finitialize()
timenow = time.time()
h.continuerun(100)
print('Simulation done in '+str(time.time()-timenow)+' seconds')

f,ax = subplots(1,1)
ax.plot(array(vec_t),array(vec_H),label='H$^+$')
ax.plot(array(vec_t),array(vec_O),label='O$^{2-}$')
ax.plot(array(vec_t),array(vec_H2O),label='H$_2$O')
ax.legend()
ax.set_xlabel('$t$ (ms)')
ax.set_ylabel('$c$ (mmol/l)')

f.savefig('H2O.eps')
```

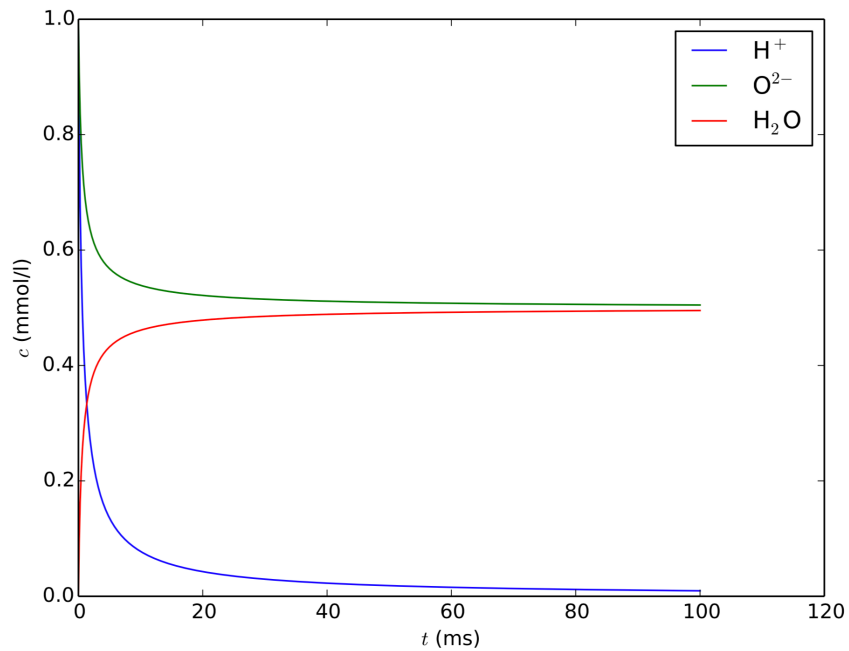
Create NEURON vectors

Record to the time to vector vec_t

Record the concentrations of $[H^+]$, $[O^{2-}]$, and $[H_2O]$

Initialize, set the timer and run for 100 ms

Create an axis, plot the figure and print it into an eps file



Examples for NEURON's RxD extension

```
initvalues = [7.3e-05, 1.877026, 0.001814, 0.000225, 0.145869, 0.014658, 0.0, 0.0, 2.502689, 0.00049, 1.3e-05, 0.016767, 0.537911, 0.005711, 0.00406, 9e-06, 5.4e-05, 0.00382, 0.002623, 1e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.\
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.001508, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.000114, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.994911, 5.5e-05, 5e-06, 8.1e-05, 0.003709, 2e-06, 0.0, 0.011482, 0.00\
0297, 0.0, 0.000976, 0.002143, 0.018013, 0.011485, 0.00018, 4e-06, 0.0, 0.00206, 0.000266, 3e-06, 0.021291, 7e-06, 0.001059, 0.000868, 0.0, 0.0, 7e-06, 6e-06, 0.002021, 0.000302, 4.3e-05, 0.0, 0.0, 0.000729, 0.0, 6e-06, 0.001121, 0.00\
0442, 0.0, 0.0, 1.1e-05, 0.00027, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.000314, 0.0, 0.0, 4.5e-05, 0.0, 0.0, 1.0, 0.0, 0.0, 0.002472, 0.0, 0.0, 0.0, 1.019075, 0.001441, 0.0, 0.0, 0.000445, 1\
.8e-05, 0.0, 0.0, 0.023783, 3.4e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.000293, 0.000402, 0.014994, 0.0, 0.0, 9.4e-05, 0.000327, 0.0, 0.0, 0.001556, 0.000253, 9.2e-05, 0.0, 0.0, 0.000601]

spec_Ca = rxd.Species(cyt, name='Ca', charge=2, initial=initvalues[0])
spec_CaOut = rxd.Species(cyt, name='CaOut', charge=0, initial=initvalues[1])
        ...

spec_GluR1_CKp = rxd.Species(cyt, name='GluR1_CKp', charge=0, initial=initvalues[107])
spec_GluR1_S831_PP1 = rxd.Species(cyt, name='GluR1_S831_PP1', charge=0, initial=initvalues[114])

Ca_flux_rate = rxd.Parameter(cyt, initial=0)

reaction000 = rxd.Reaction(spec_Ca + spec_pmca <> spec_pmcaCa, 50.0, 0.007)
reaction001 = rxd.Reaction(spec_pmcaCa > spec_pmca + spec_CaOut, 0.0035)
reaction002 = rxd.Reaction(spec_Ca + spec_ncx <> spec_ncxCa, 16.8, 0.0112)
reaction003 = rxd.Reaction(spec_ncxCa > spec_ncx + spec_CaOut, 0.0056)
reaction004 = rxd.Reaction(spec_CaOut + spec_Leak <> spec_CaOutLeak, 1.5, 0.0011)
reaction005 = rxd.Reaction(spec_CaOutLeak > spec_Ca + spec_Leak, 0.0011)

reaction067 = rxd.Reaction(spec_CaM + spec_Ca <> spec_CaMCa2, 6.0, 0.0091)
reaction068 = rxd.Reaction(spec_CaMCa2 + spec_Ca <> spec_CaMCa4, 100.0, 1.0)
reaction075 = rxd.Reaction(spec_CaMCa4 + spec_CK <> spec_CKCaMCa4, 10.0, 0.003)
reaction076 = rxd.Reaction(spec_CKCaMCa4*2 <> spec_Complex, 0.1, 0.01)
reaction077 = rxd.Reaction(spec_CKpCaMCa4 + spec_CKCaMCa4 <> spec_pComplex, 0.1, 0.01)
reaction078 = rxd.Reaction(spec_CKpCaMCa4 + spec_Complex > spec_CKpCaMCa4 + spec_pComplex, 0.1)
reaction079 = rxd.Reaction(spec_CKCaMCa4 + spec_Complex > spec_CKCaMCa4 + spec_pComplex, 0.1)
reaction080 = rxd.Reaction(spec_Complex*2 > spec_Complex + spec_pComplex, 10.0)
reaction081 = rxd.Reaction(spec_Complex + spec_pComplex > spec_pComplex*2, 30.0)
reaction082 = rxd.Reaction(spec_CKpCaMCa4 <> spec_CaMCa4 + spec_CKp, 8e-07, 10.0)
reaction083 = rxd.Reaction(spec_CKp + spec_PP1 <> spec_CKpPP1, 0.004, 0.00034)
reaction084 = rxd.Reaction(spec_CKpPP1 > spec_PP1 + spec_CK, 8.6e-05)
reaction085 = rxd.Reaction(spec_CKpCaMCa4 + spec_PP1 <> spec_CKpCaMCa4PP1, 0.004, 0.00034)
reaction086 = rxd.Reaction(spec_CKpCaMCa4PP1 > spec_PP1 + spec_CKCaMCa4, 8.6e-05)

reaction100 = rxd.Reaction(spec_GluR1 + spec_CKCaMCa4 <> spec_GluR1_CKCam, 0.02224, 0.0016)
reaction101 = rxd.Reaction(spec_GluR1_CKCam > spec_GluR1_S831 + spec_CKCaMCa4, 0.0004)
reaction102 = rxd.Reaction(spec_GluR1 + spec_CKpCaMCa4 <> spec_GluR1_CKpCam, 0.0278, 0.002)
reaction103 = rxd.Reaction(spec_GluR1_CKpCam > spec_GluR1_S831 + spec_CKpCaMCa4, 0.0005)
reaction104 = rxd.Reaction(spec_GluR1 + spec_CKp <> spec_GluR1_CKp, 0.02224, 0.0016)
reaction105 = rxd.Reaction(spec_GluR1_CKp > spec_GluR1_S831 + spec_CKp, 0.0004)
reaction119 = rxd.Reaction(spec_GluR1_S831 + spec_PP1 <> spec_GluR1_S831_PP1, 0.875, 0.0014)
reaction120 = rxd.Reaction(spec_GluR1_S831_PP1 > spec_GluR1 + spec_PP1, 0.00035)

reaction_Ca_flux = rxd.Rate(spec_Ca, Ca_flux_rate)
T = 1000./Ca_input_freq
tnow = 0
for itrain in range(0,Ntrains):
    for istim in range(0,Ca_input_N):
      tnew = Ca_input_onset + istim*T + trainT*itrain
      h.cvode.event(tnew, lambda: set_param(Ca_flux_rate, Ca_input_flux/6.022e23/my_volume*1e3))
      h.cvode.event(tnew+Ca_input_dur, lambda: set_param(Ca_flux_rate, 0))
```

$\{\} \longrightarrow \mathrm{Ca}$

$\mathrm{PMCA} + \mathrm{Ca} \rightleftharpoons \mathrm{PMCACa} \rightarrow \mathrm{PMCA} + \mathrm{Ca}_{\mathrm{out}}$

$\mathrm{ncx} + \mathrm{Ca} \rightleftharpoons \mathrm{ncxCa} \rightarrow \mathrm{ncx} + \mathrm{Ca}_{\mathrm{out}}$

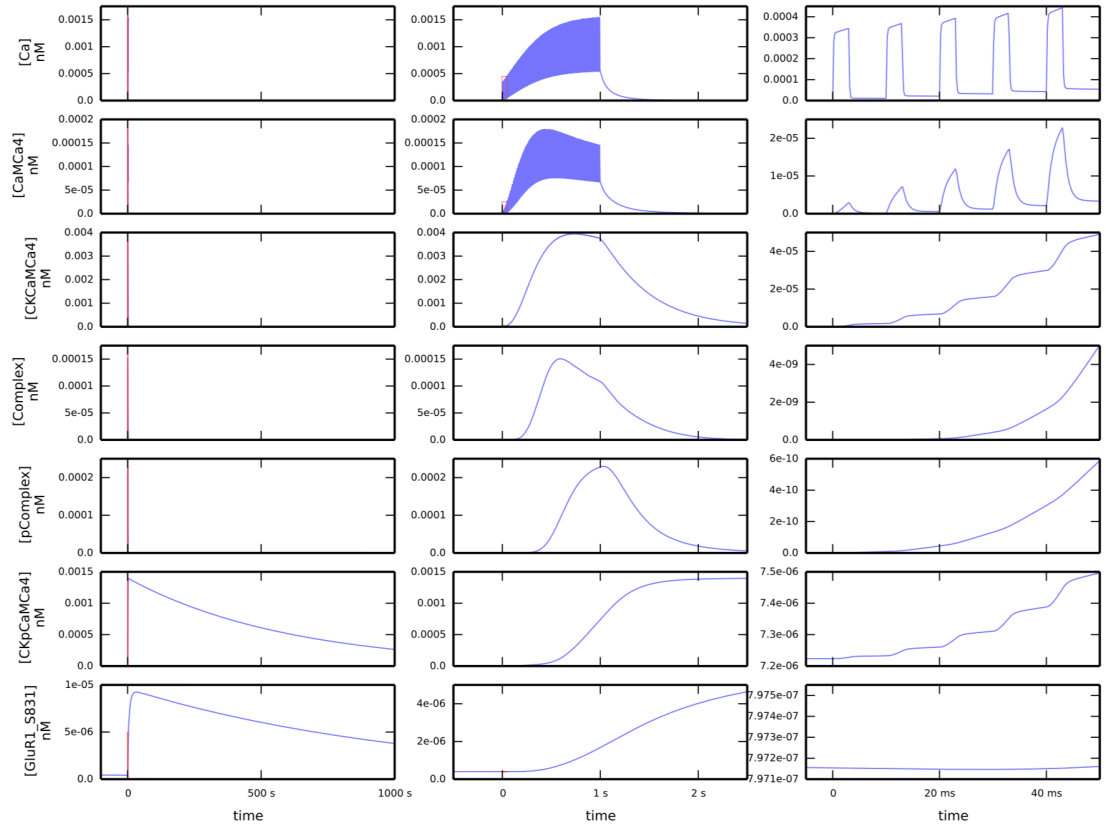
$\mathrm{Ca}_{\mathrm{Out}} + \mathrm{Leak} \rightleftharpoons \mathrm{Ca}_{\mathrm{Out}}\mathrm{Leak} \rightarrow \mathrm{Ca} + \mathrm{Leak}$

$2\mathrm{Ca} + \mathrm{CaM} \rightleftharpoons \mathrm{CaMCa}_2$

$2\mathrm{Ca} + \mathrm{CaMCa}_2 \rightleftharpoons \mathrm{CaMCa}_4$

$\mathrm{CaMKII} + \mathrm{CaMCa}_4 \rightleftharpoons \mathrm{CaMKIICaMCa}_4$

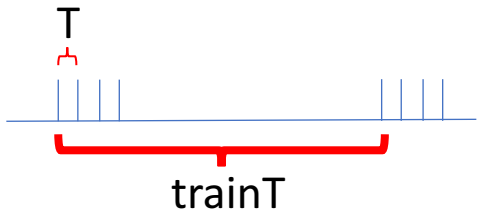
...



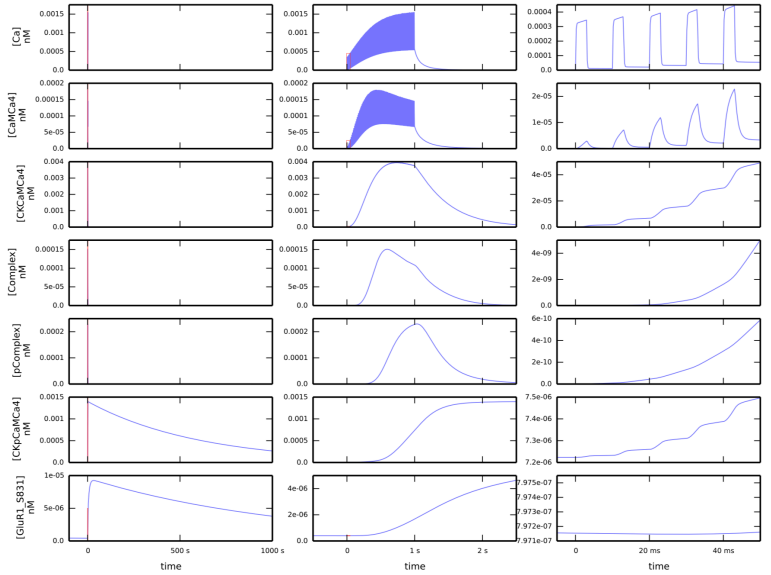
Examples for NEURON's RxD extension

```
Ca_flux_rate = rxd.Parameter(cyt, initial=0)
reaction_Ca_flux = rxd.Rate(spec_Ca, Ca_flux_rate)
```

$$\mathrm{Ca}_{(\mathrm{ext})} \longrightarrow \mathrm{Ca}$$




```
T = 1000./Ca_input_freq
tnow = 0
for itrain in range(0,Ntrains):
    for istim in range(0,Ca_input_N):
        tnew = Ca_input_onset + istim*T + trainT*itrain
        h.cvode.event(tnew, lambda: set_param(Ca_flux_rate, Ca_input_flux/6.022e23/my_volume*1e3))
        h.cvode.event(tnew+Ca_input_dur, lambda: set_param(Ca_flux_rate, 0))
```




```
reaction000 = rxd.Reaction(spec_Ca + spec_pmca <> spec_pmcaCa, 50.0, 0.007)
reaction001 = rxd.Reaction(spec_pmcaCa > spec_pmca + spec_CaOut, 0.0035)
reaction002 = rxd.Reaction(spec_Ca + spec_ncx <> spec_ncxCa, 16.8, 0.0112)
reaction003 = rxd.Reaction(spec_ncxCa > spec_ncx + spec_CaOut, 0.0056)
reaction004 = rxd.Reaction(spec_CaOut + spec_Leak <> spec_CaOutLeak, 1.5, 0.0011)
reaction005 = rxd.Reaction(spec_CaOutLeak > spec_Ca + spec_Leak, 0.0011)
```

$$\mathrm{PMCA} + \mathrm{Ca} \rightleftharpoons \mathrm{PMCACa} \rightarrow \mathrm{PMCA} + \mathrm{Ca}_{\mathrm{out}}$$

$$\mathrm{ncx} + \mathrm{Ca} \rightleftharpoons \mathrm{ncxCa} \rightarrow \mathrm{ncx} + \mathrm{Ca}_{\mathrm{out}}$$

$$\mathrm{Ca}_{\mathrm{Out}} + \mathrm{Leak} \rightleftharpoons \mathrm{Ca}_{\mathrm{Out}}\mathrm{Leak} \rightarrow \mathrm{Ca} + \mathrm{Leak}$$

```
reaction067 = rxd.Reaction(spec_CaM + spec_Ca**2 <> spec_CaMCa2, 6.0*spec_CaM*spec_Ca, 0.0091*spec_CaMCa2, custom_dynamics=True)
reaction068 = rxd.Reaction(spec_CaMCa2 + spec_Ca**2 <> spec_CaMCa4, 100.0*spec_CaMCa2*spec_Ca, 1.0*spec_CaMCa4, custom_dynamics=True)
reaction075 = rxd.Reaction(spec_CaMCa4 + spec_CK <> spec_CKCaMCa4, 10.0, 0.003)
```

$$2\mathrm{Ca} + \mathrm{CaM} \rightleftharpoons \mathrm{CaMCa}_2$$

$$2\mathrm{Ca} + \mathrm{CaMCa}_2 \rightleftharpoons \mathrm{CaMCa}_4$$

$$\mathrm{CaMKII} + \mathrm{CaMCa}_4 \rightleftharpoons \mathrm{CaMKIICaMCa}_4$$

Python3: <> replaced by !=