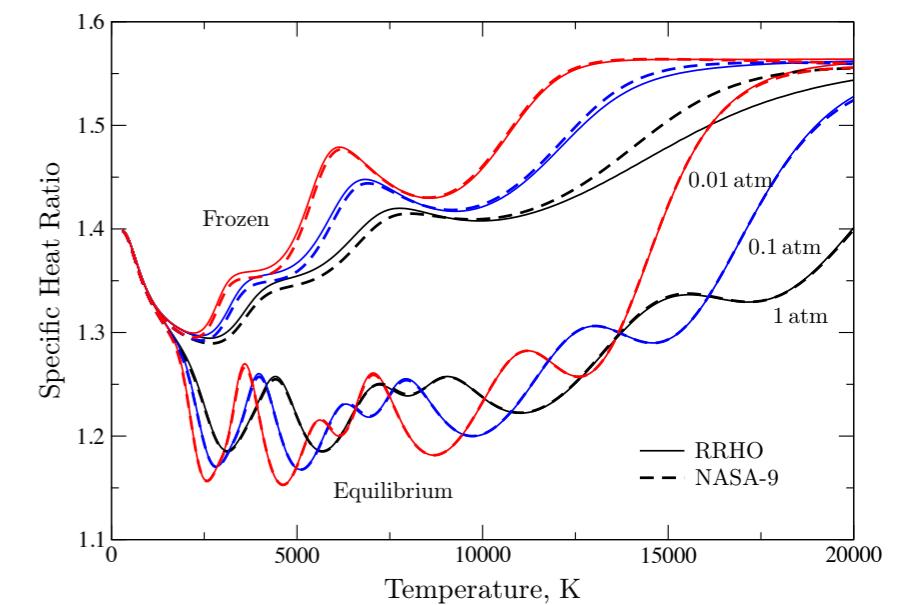
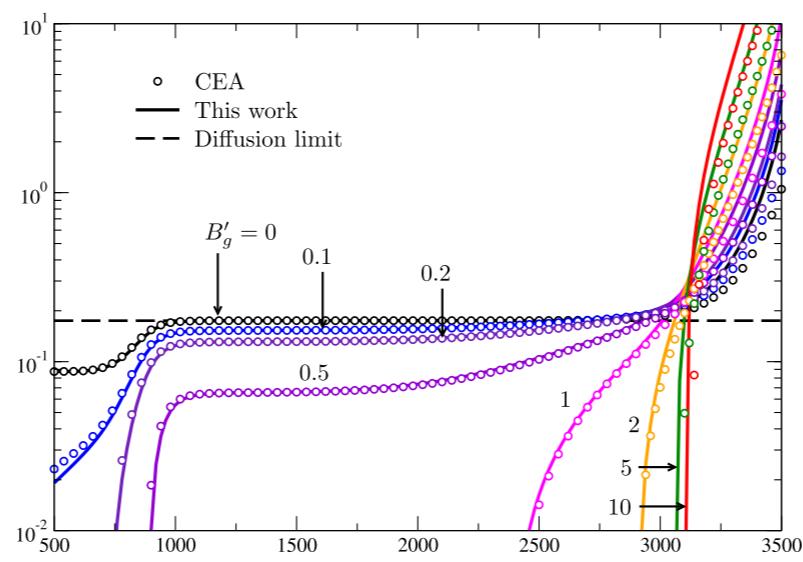
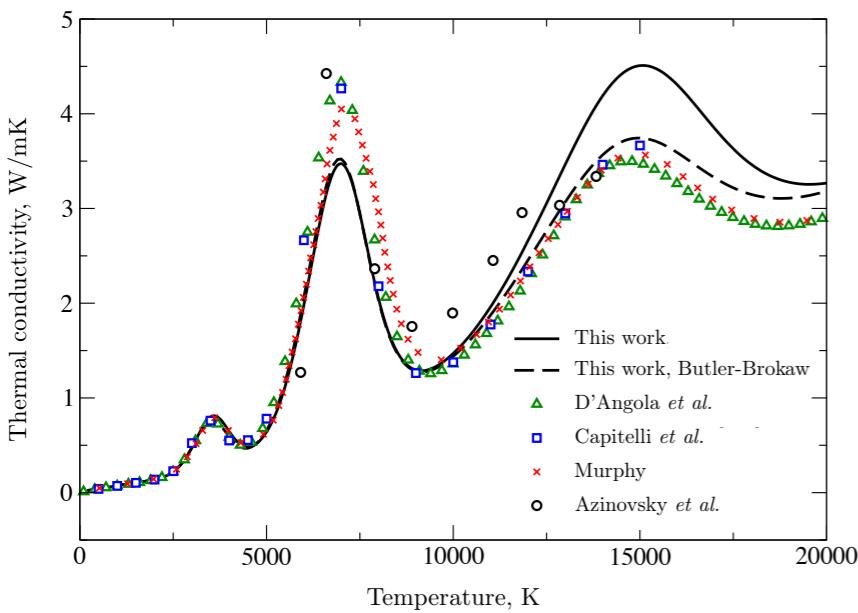


Mutation++

MULTicomponent Thermodynamic And Transport properties
of IONized gases library in C++



Presented by J.B. Scoggins, Georgios Bellas-Chatzigeorgis, and Bruno Dias

VKI Mutation++ Training Day

Friday, June 15, 2018

9:30 am	Welcome and overview of the library
9:50 am	Download and installation <ul style="list-style-type: none">• Using the Gitlab and wiki• Installing dependencies• Compiling• Running tests• Building the documentation
10:15 am	<i>Coffee break</i>
10:30 am	Discovering the built in tools of Mutation++ <ul style="list-style-type: none">• Theory: Gas mixtures, thermodynamics, equilibrium, finite-rate chemistry, and equilibrium surface chemistry (B')• Finding / modifying mixture files• <i>checkmix</i> – Check that a mixture has been formatted correctly• <i>mppequil</i> – Compute properties of a gas in equilibrium• <i>bprime</i> – Compute B' tables for a specific mixture and surface
12:15 pm	<i>Lunch</i>
1:15 pm	Coding with Mutation++* <ul style="list-style-type: none">• Setting up dependencies for an external project• Loading / creating a mixture in code• Implementing a 0D reactor with thermochemical nonequilibrium <p>* Examples will be provided in C++ and Fortran</p>
3:15 pm	<i>Coffee break</i>
3:30 pm	Coding with Mutation++ continued
4:00 pm	Discussion
4:30 pm	Closure

VKI Mutation++ Training Day

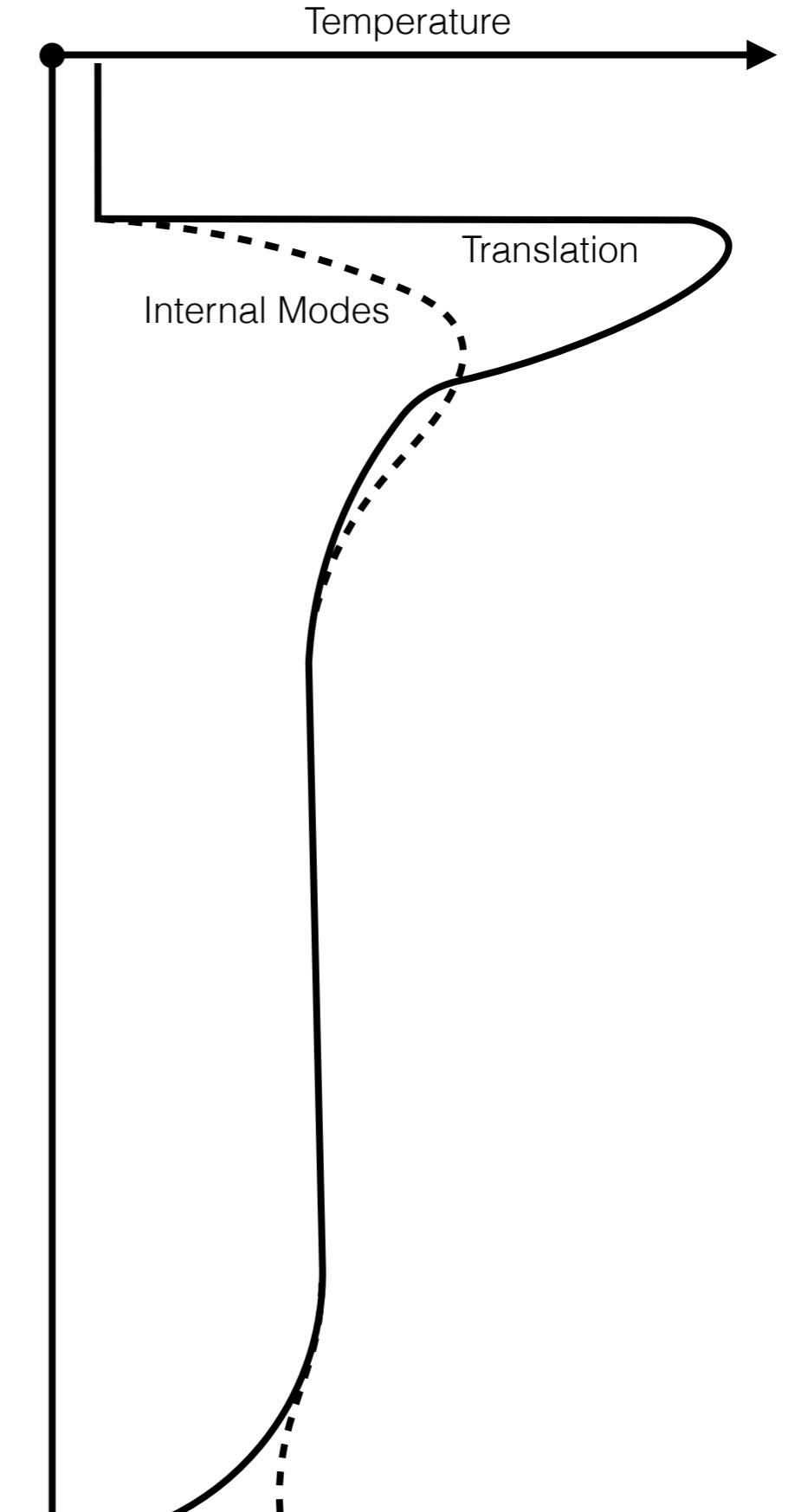
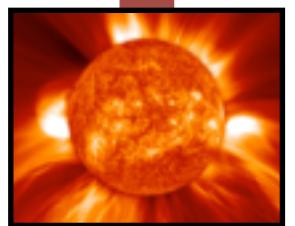
Friday, June 15, 2018

9:30 am	Welcome and overview of the library
9:50 am	Download and installation <ul style="list-style-type: none">• Using the Gitlab and wiki• Installing dependencies• Compiling• Running tests• Building the documentation
10:15 am	<i>Coffee break</i>
10:30 am	Discovering the built in tools of Mutation++ <ul style="list-style-type: none">• Theory: Gas mixtures, thermodynamics, equilibrium, finite-rate chemistry, and equilibrium surface chemistry (B')• Finding / modifying mixture files• <i>checkmix</i> – Check that a mixture has been formatted correctly• <i>mppequil</i> – Compute properties of a gas in equilibrium• <i>bprime</i> – Compute B' tables for a specific mixture and surface
12:15 pm	<i>Lunch</i>
1:15 pm	Coding with Mutation++* <ul style="list-style-type: none">• Setting up dependencies for an external project• Loading / creating a mixture in code• Implementing a 0D reactor with thermochemical nonequilibrium <p>* Examples will be provided in C++ and Fortran</p>
3:15 pm	<i>Coffee break</i>
3:30 pm	Coding with Mutation++ continued
4:00 pm	Discussion
4:30 pm	Closure

Atmospheric entry physics (Earth entries)



Atmospheric entry physics (Earth entries)

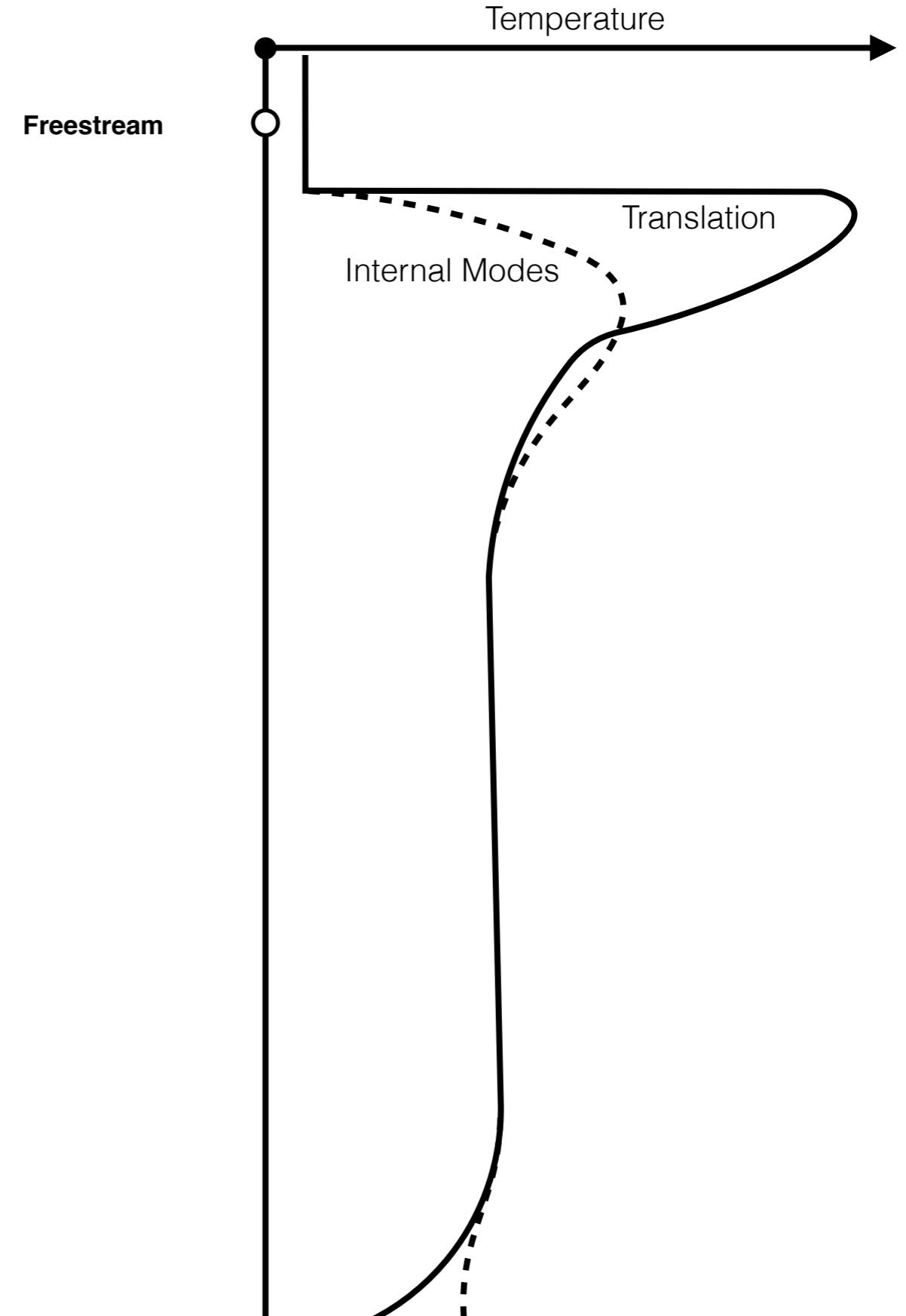
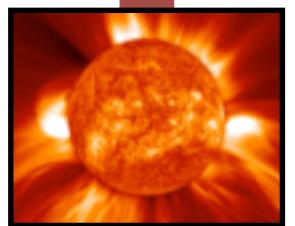
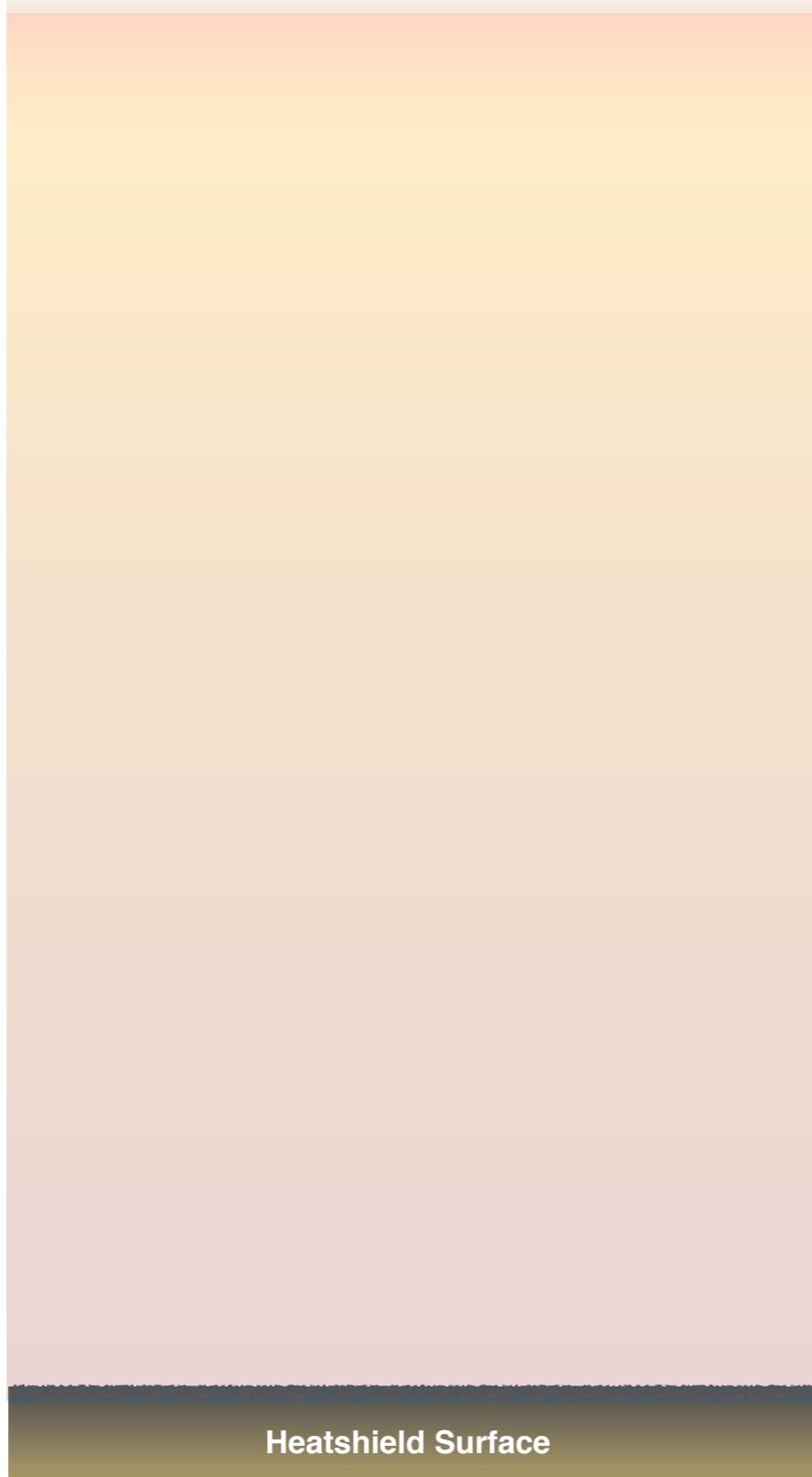


Internal Modes

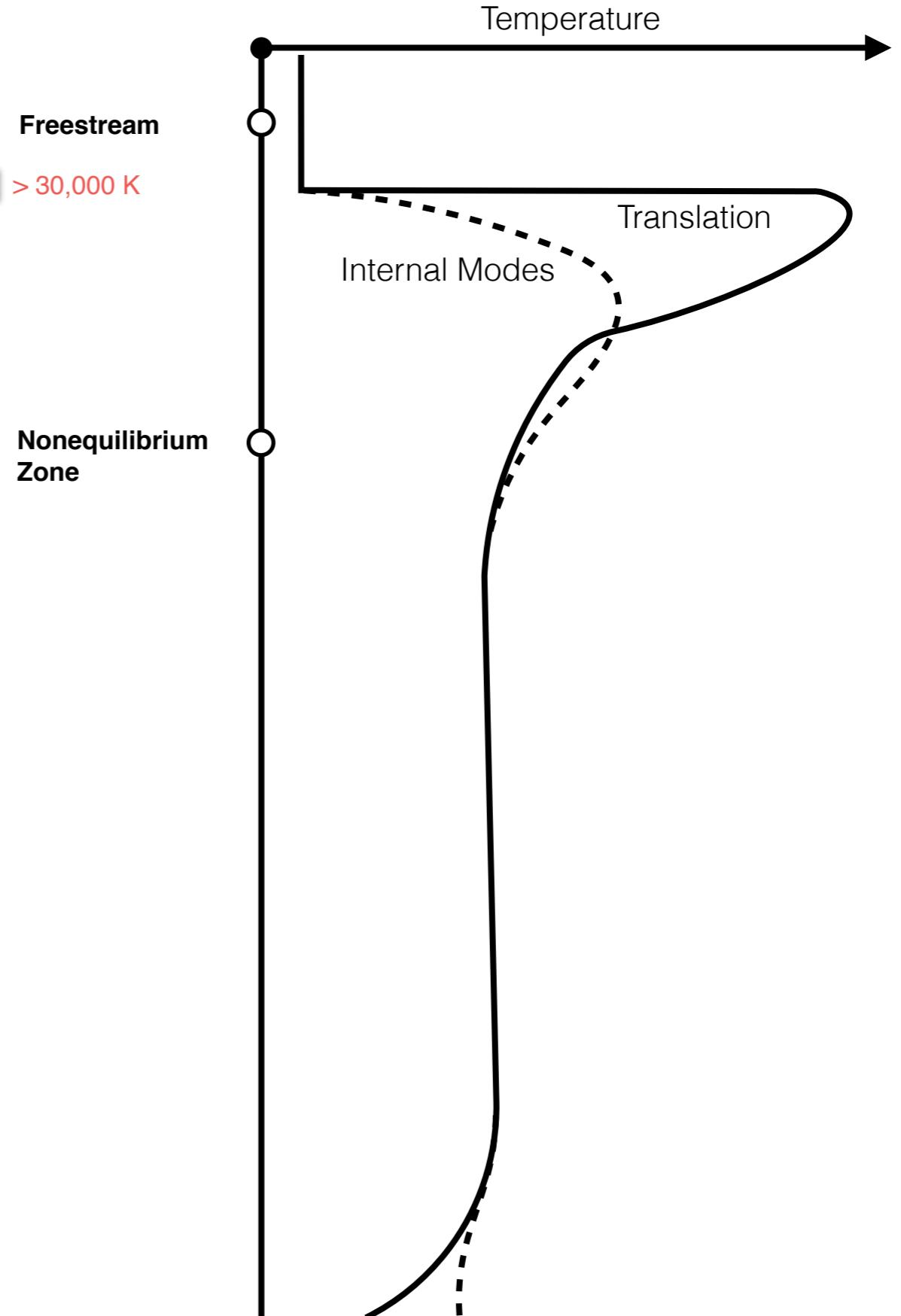
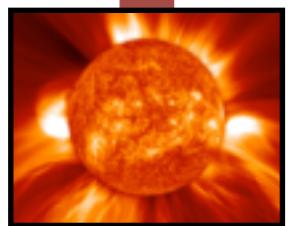
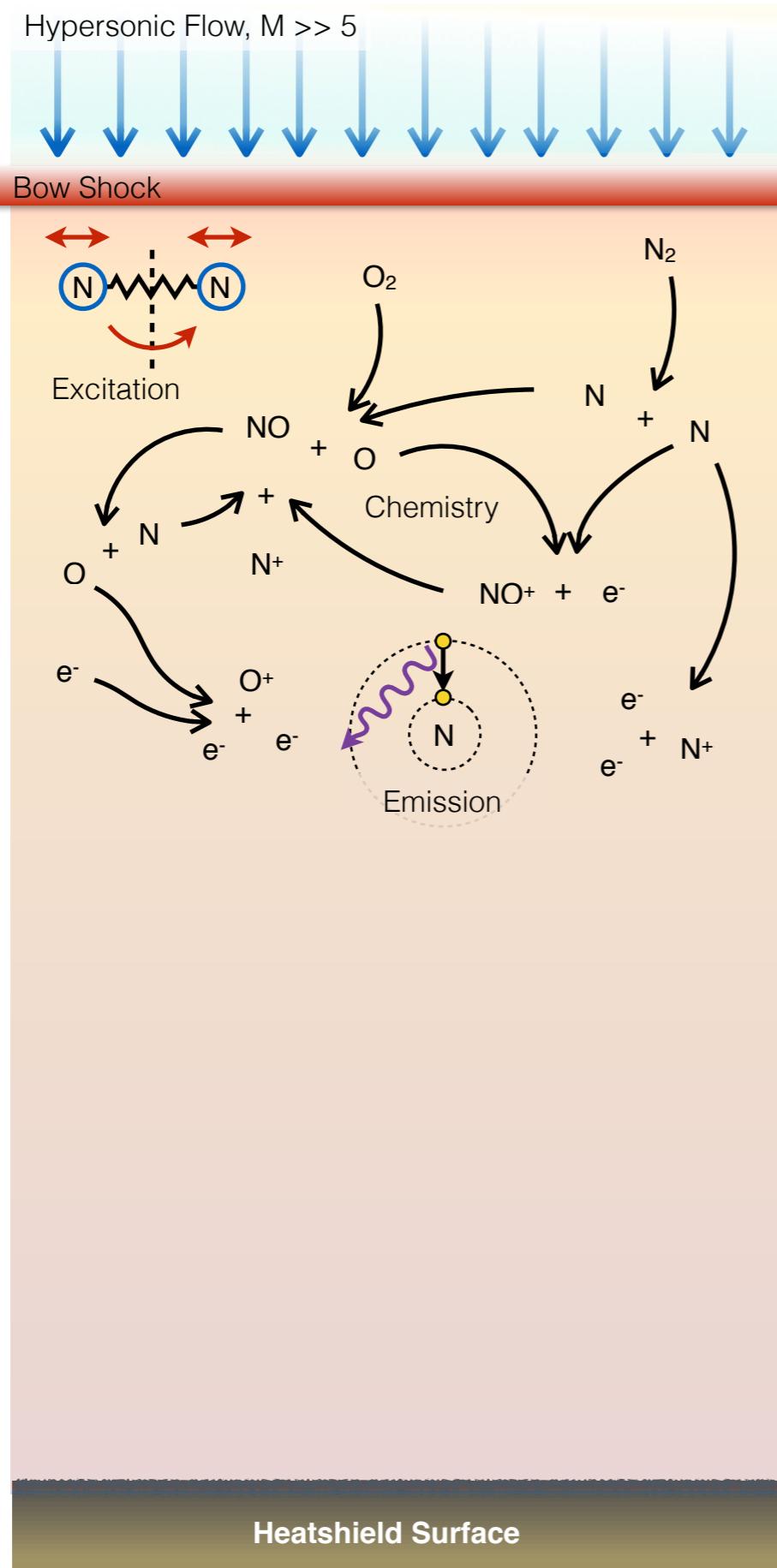
Translation

Temperature

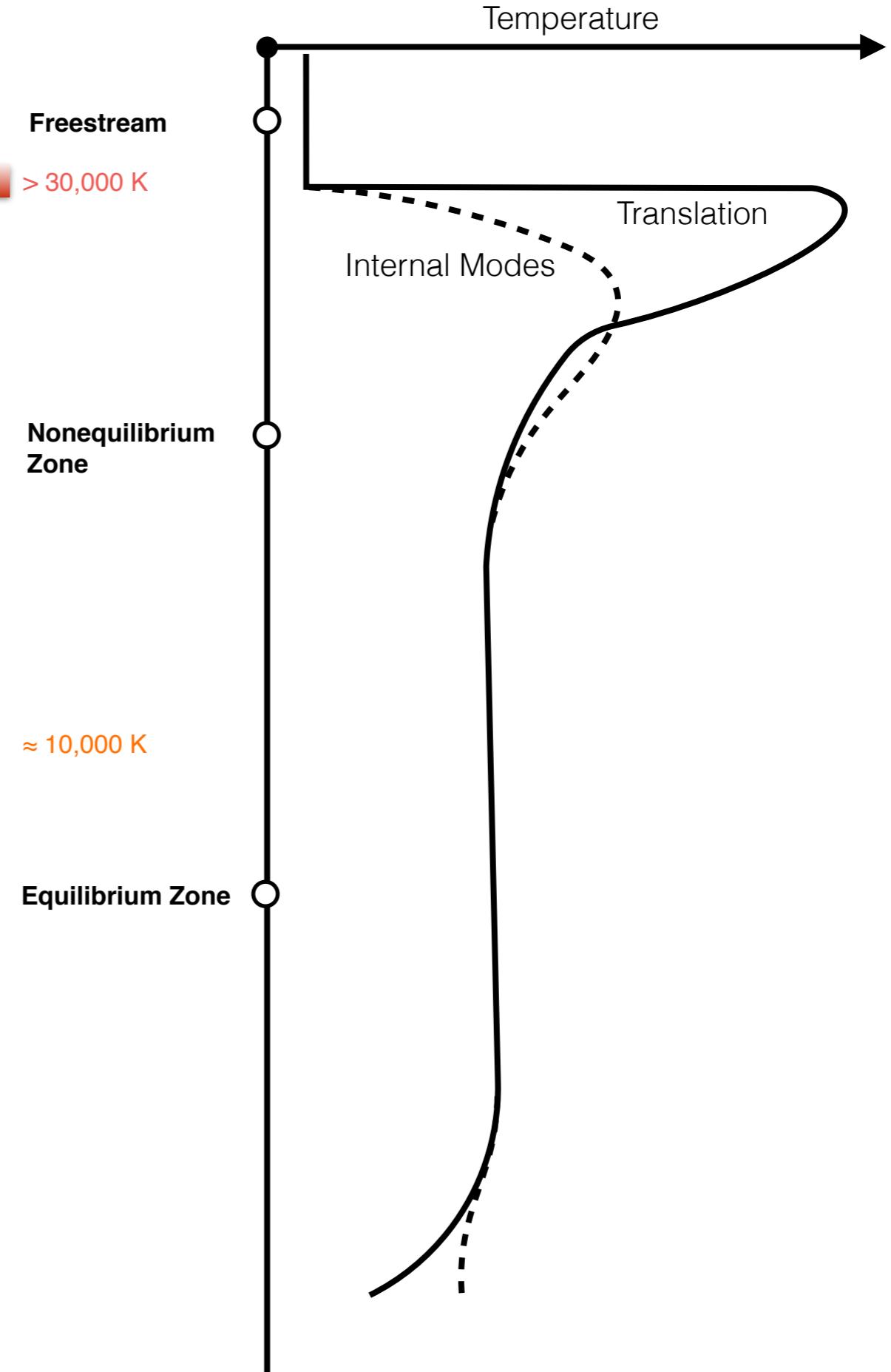
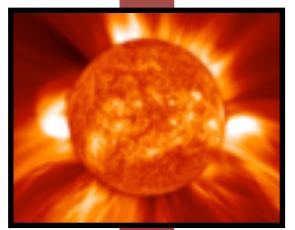
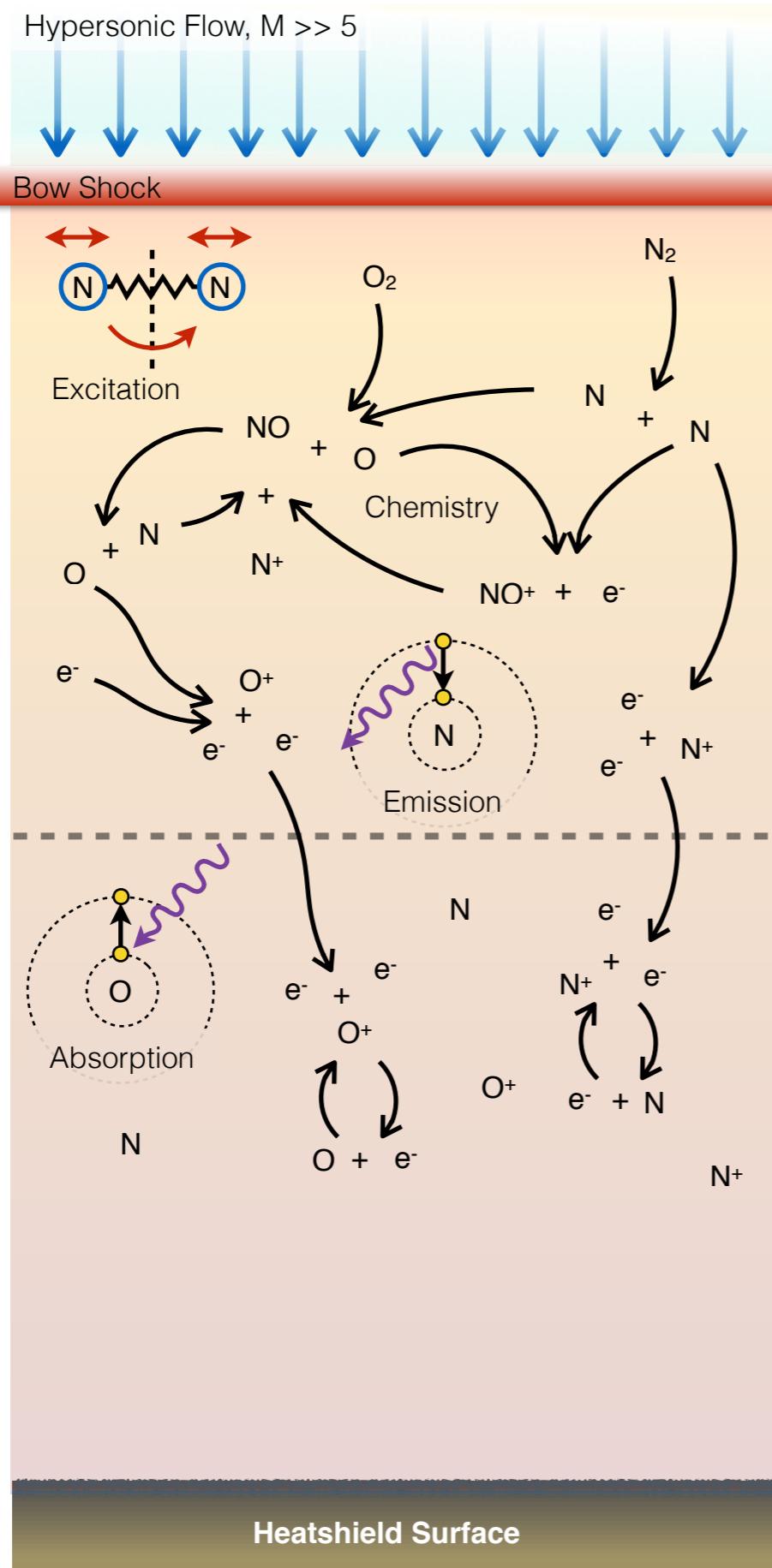
Atmospheric entry physics (Earth entries)



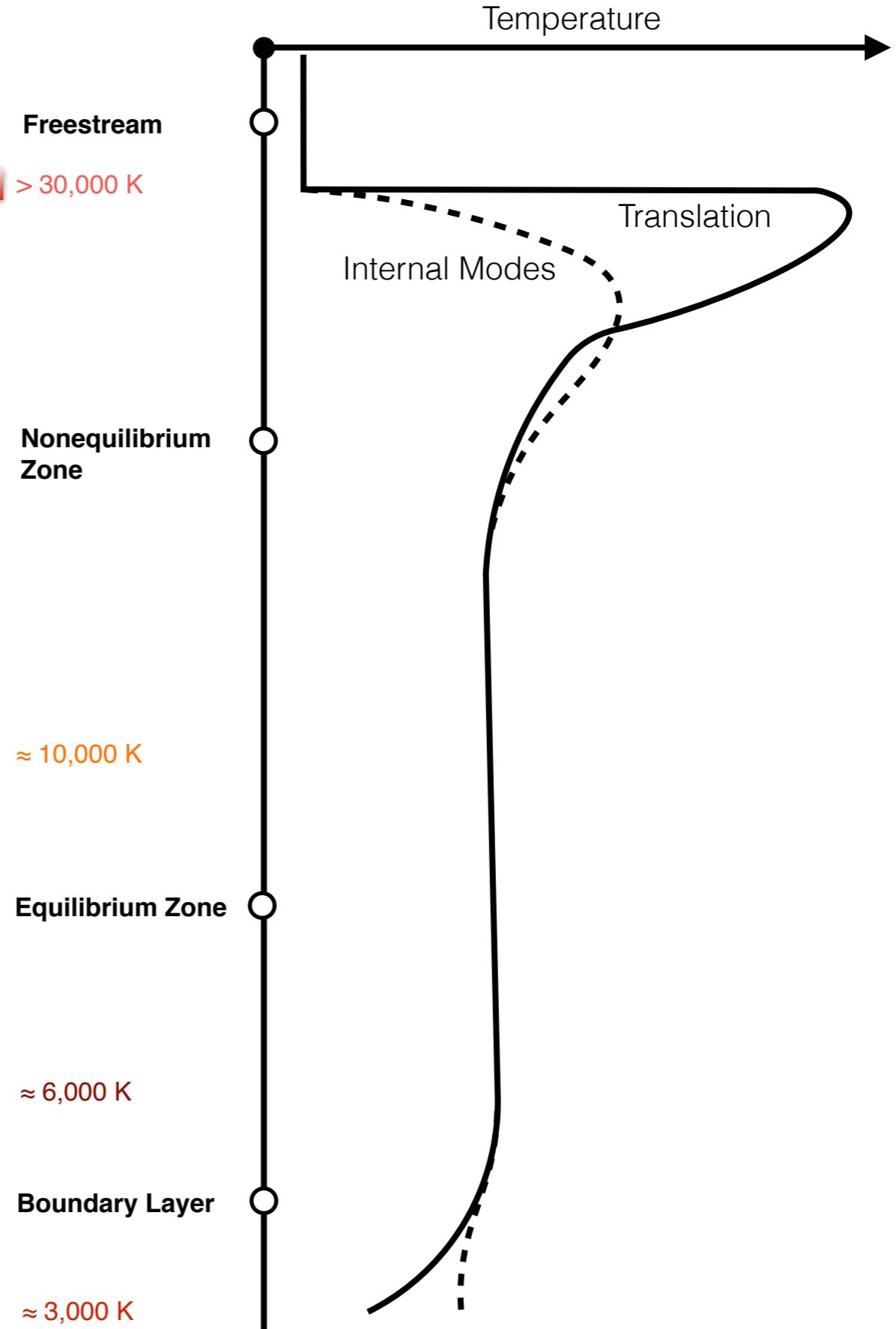
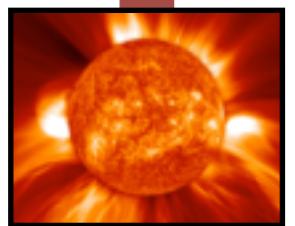
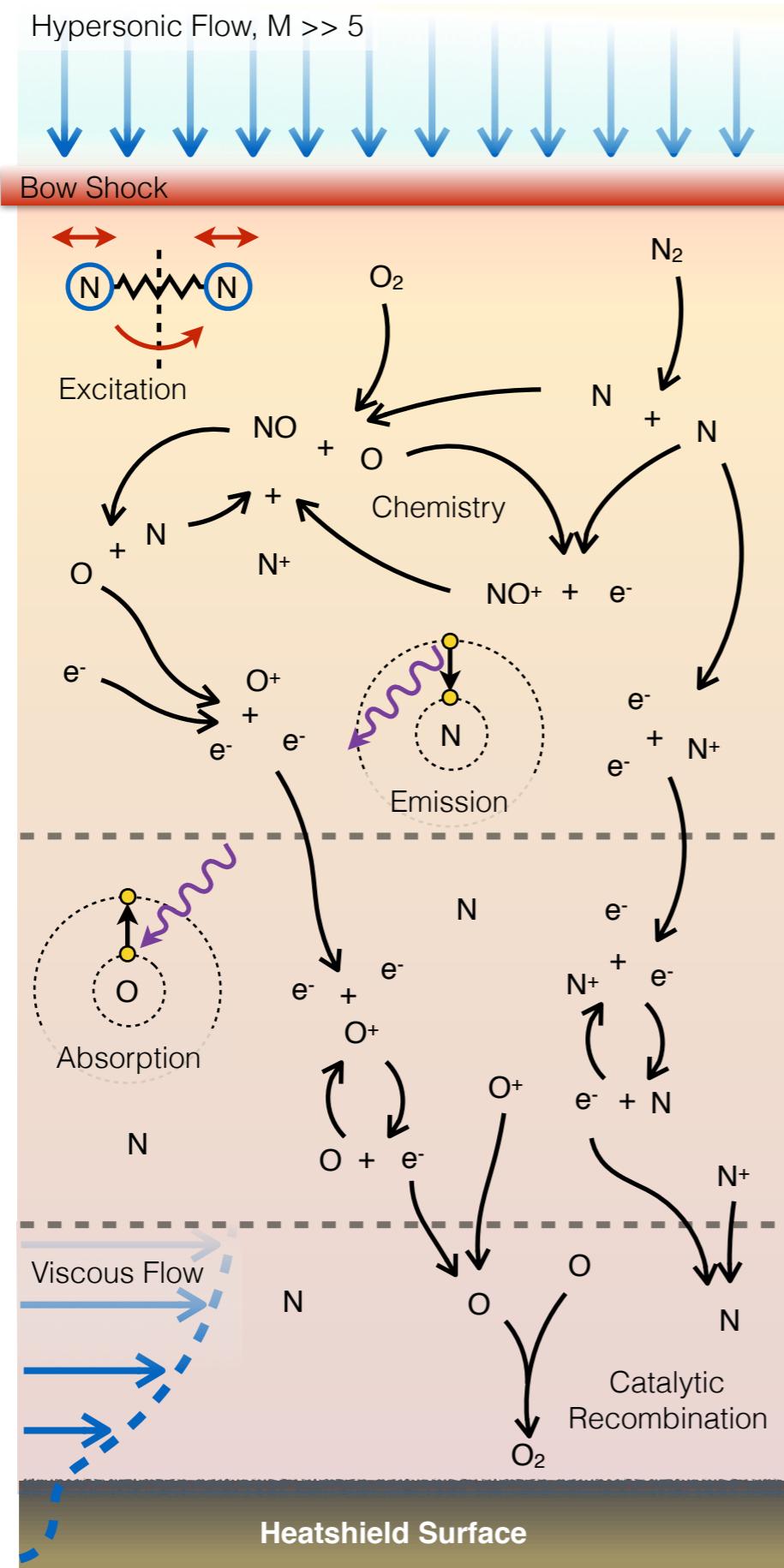
Atmospheric entry physics (Earth entries)



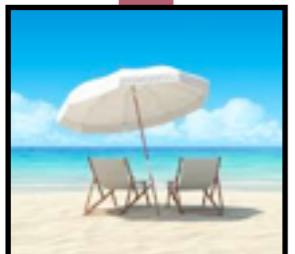
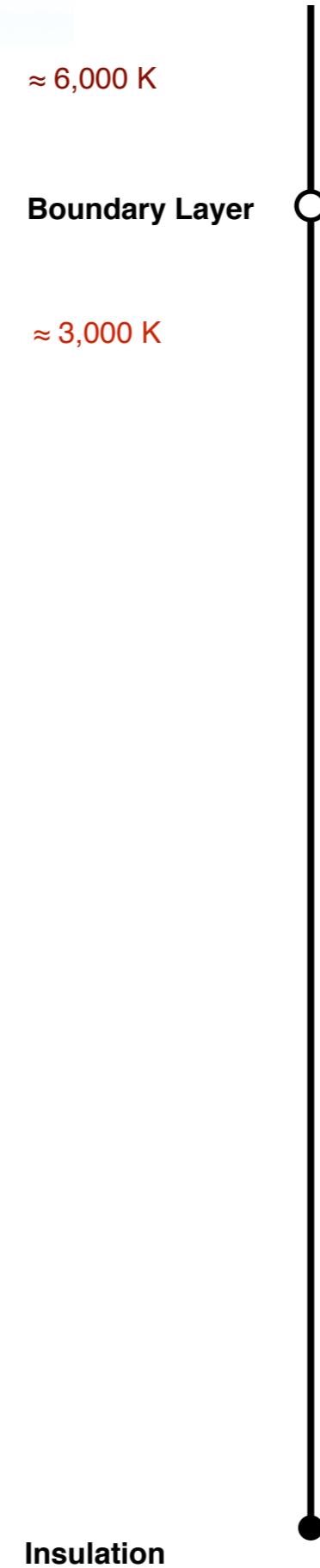
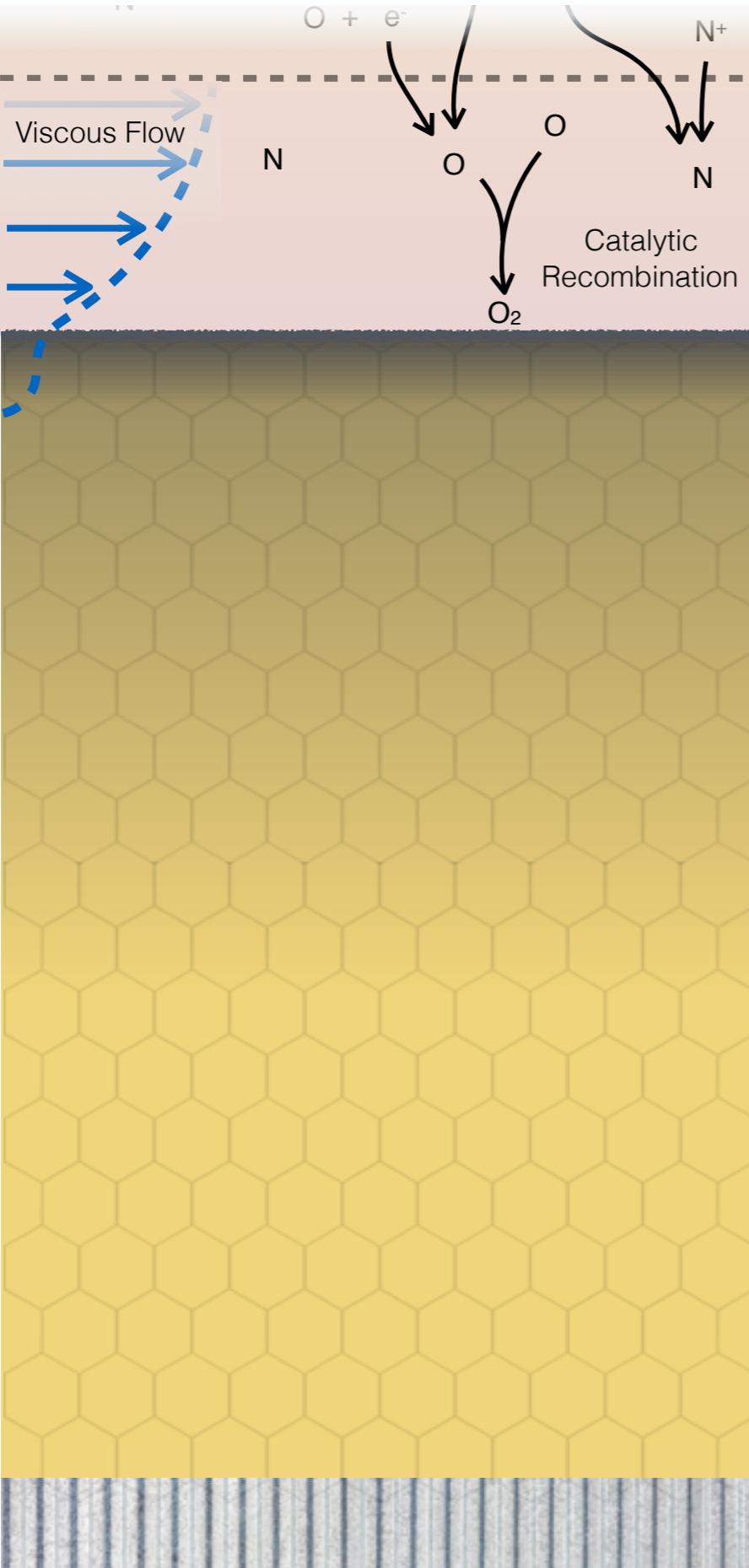
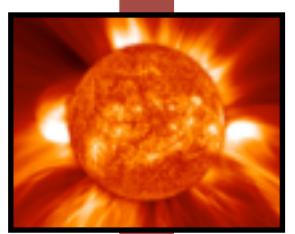
Atmospheric entry physics (Earth entries)



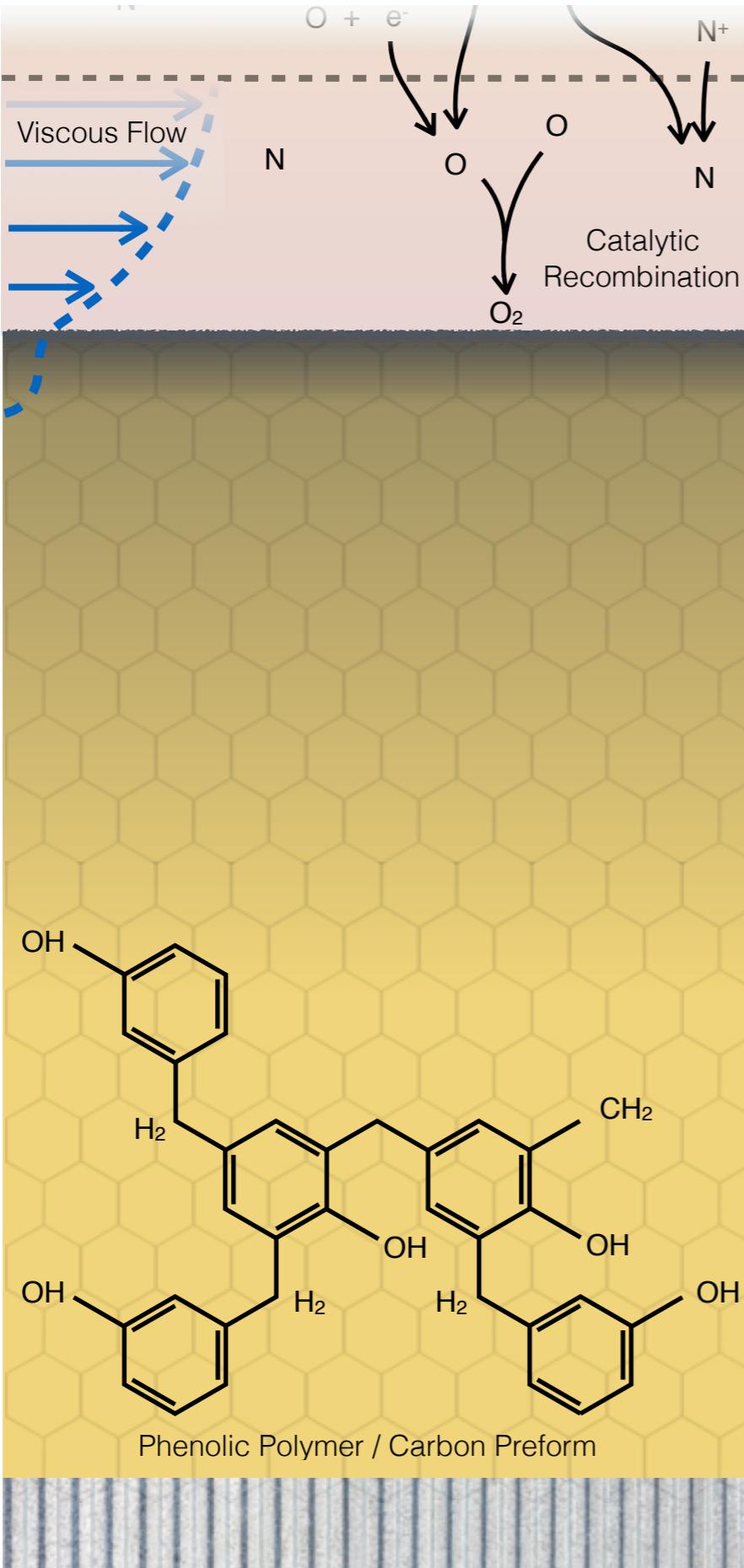
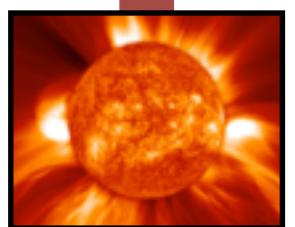
Atmospheric entry physics (Earth entries)



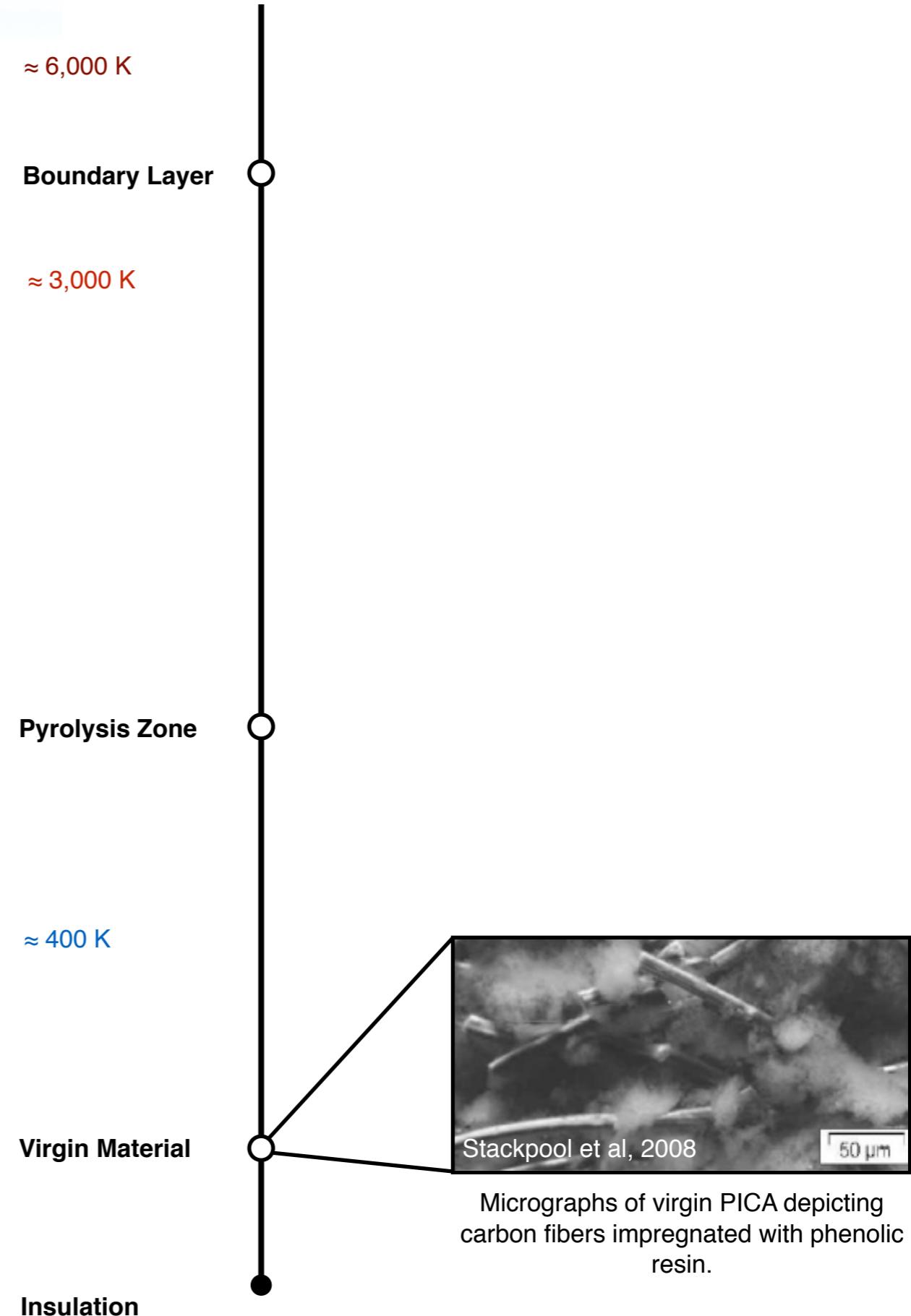
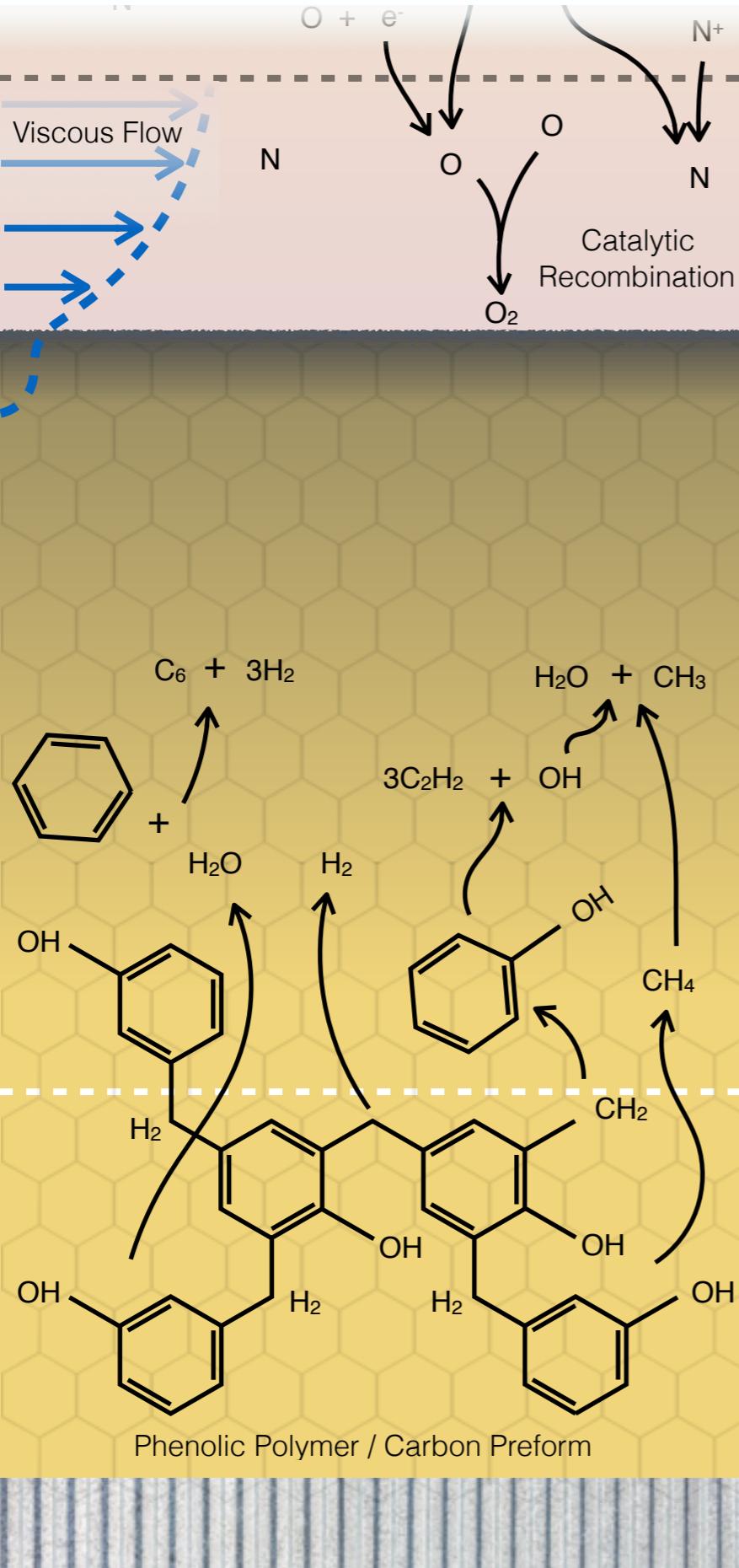
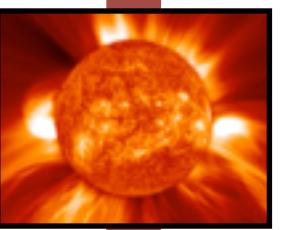
Thermal decomposition of the TPS



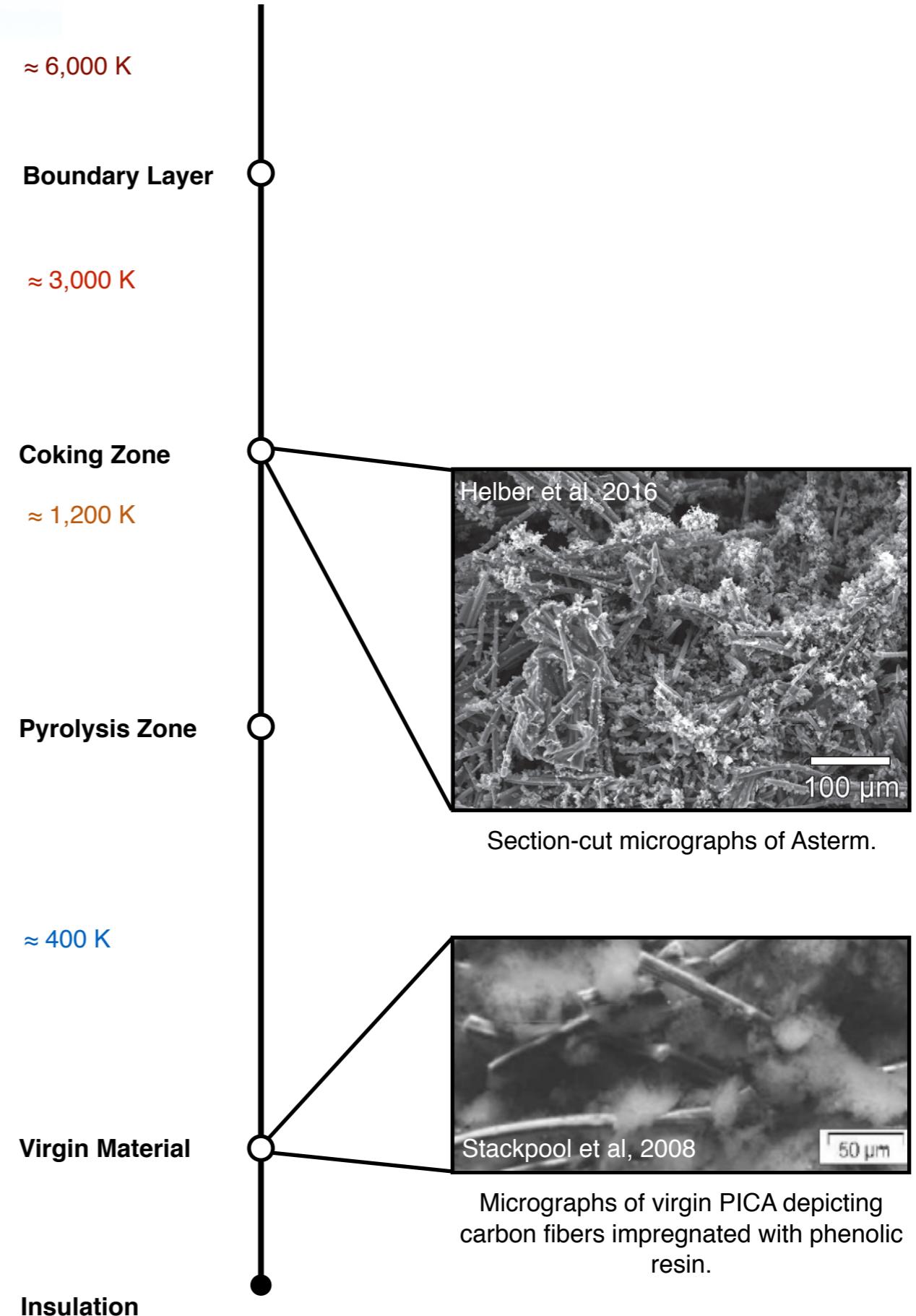
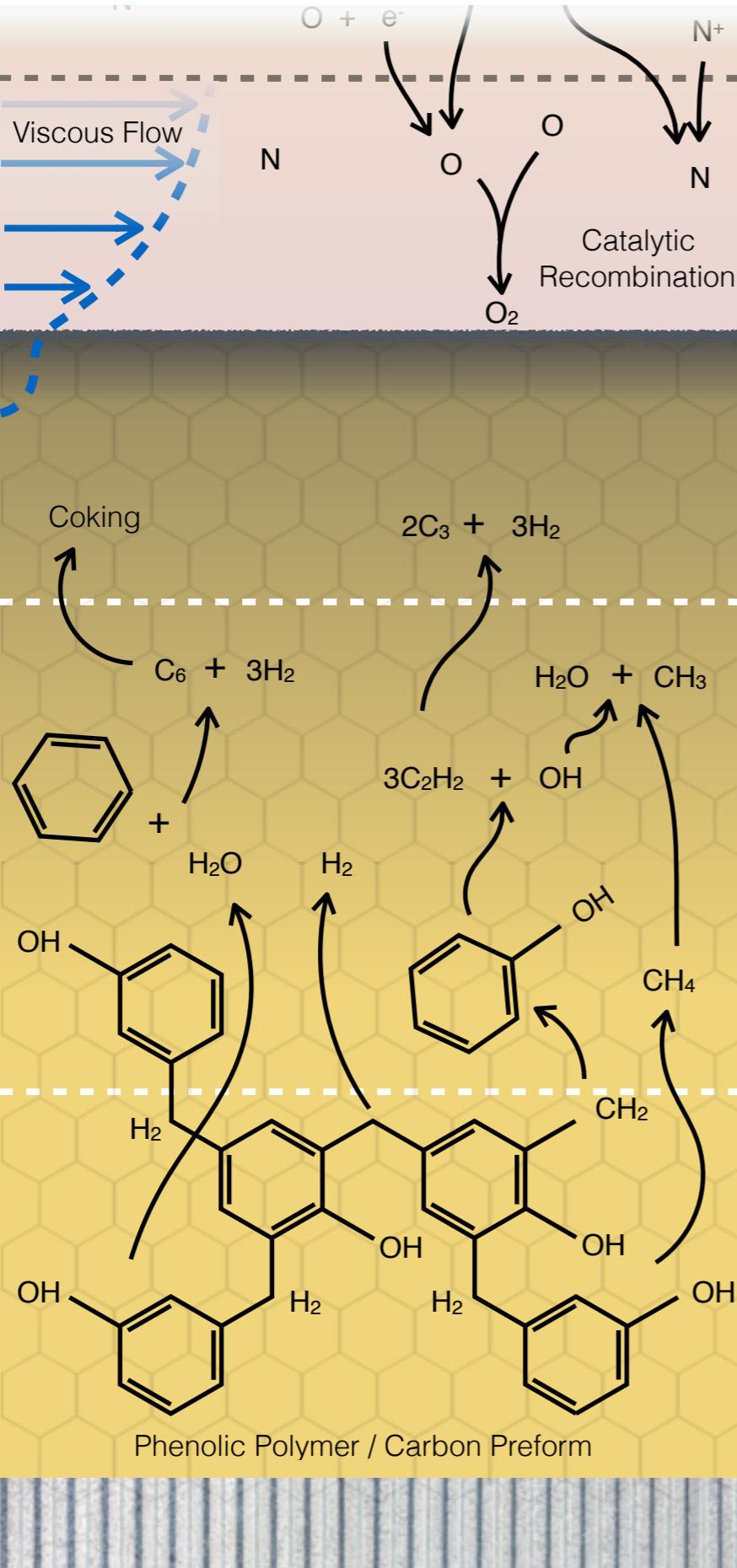
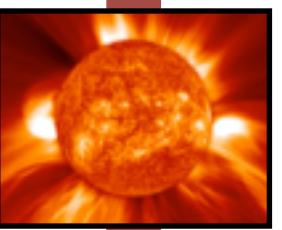
Thermal decomposition of the TPS



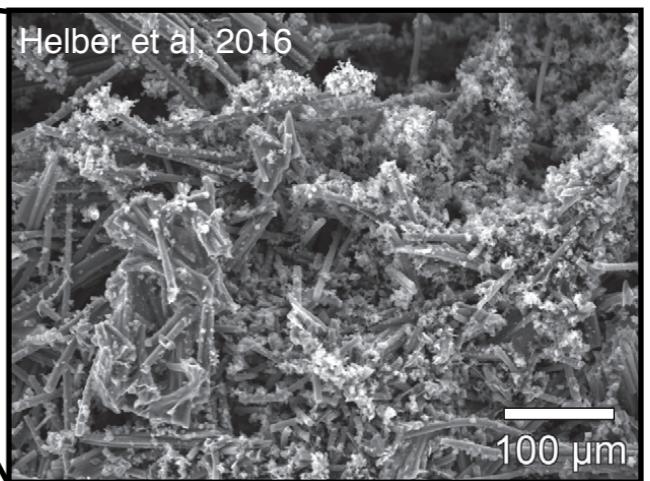
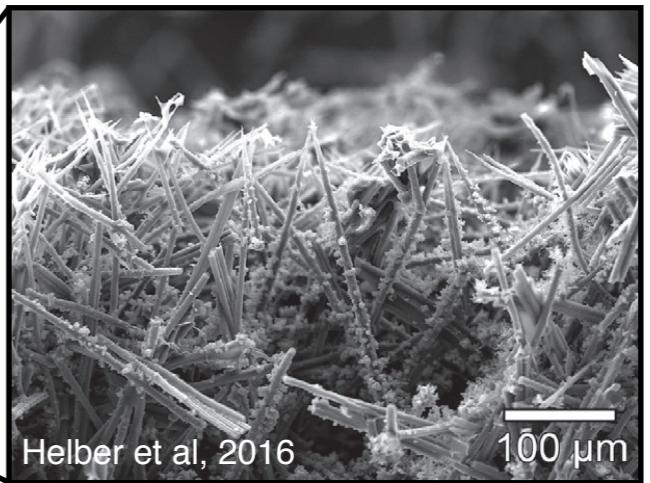
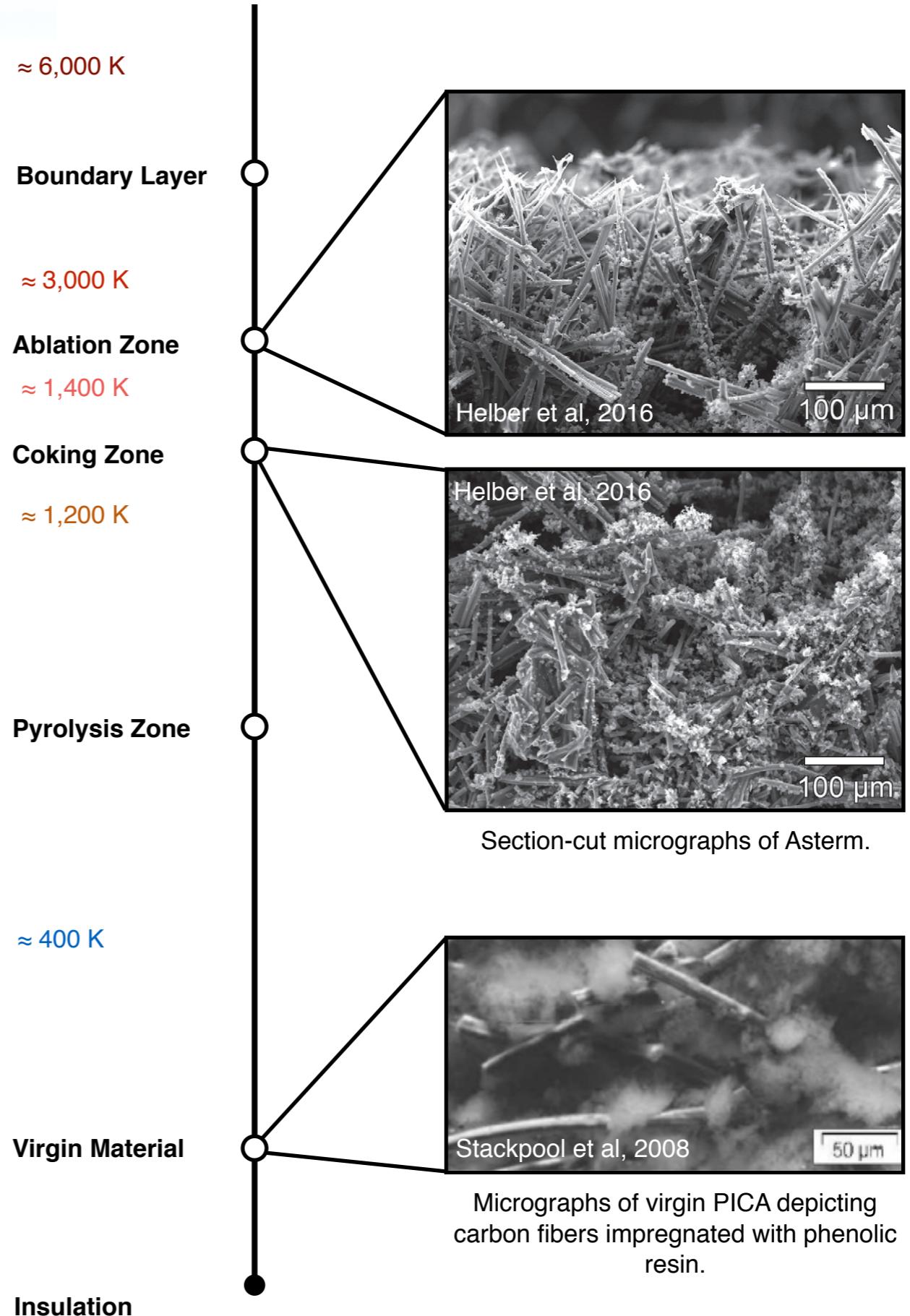
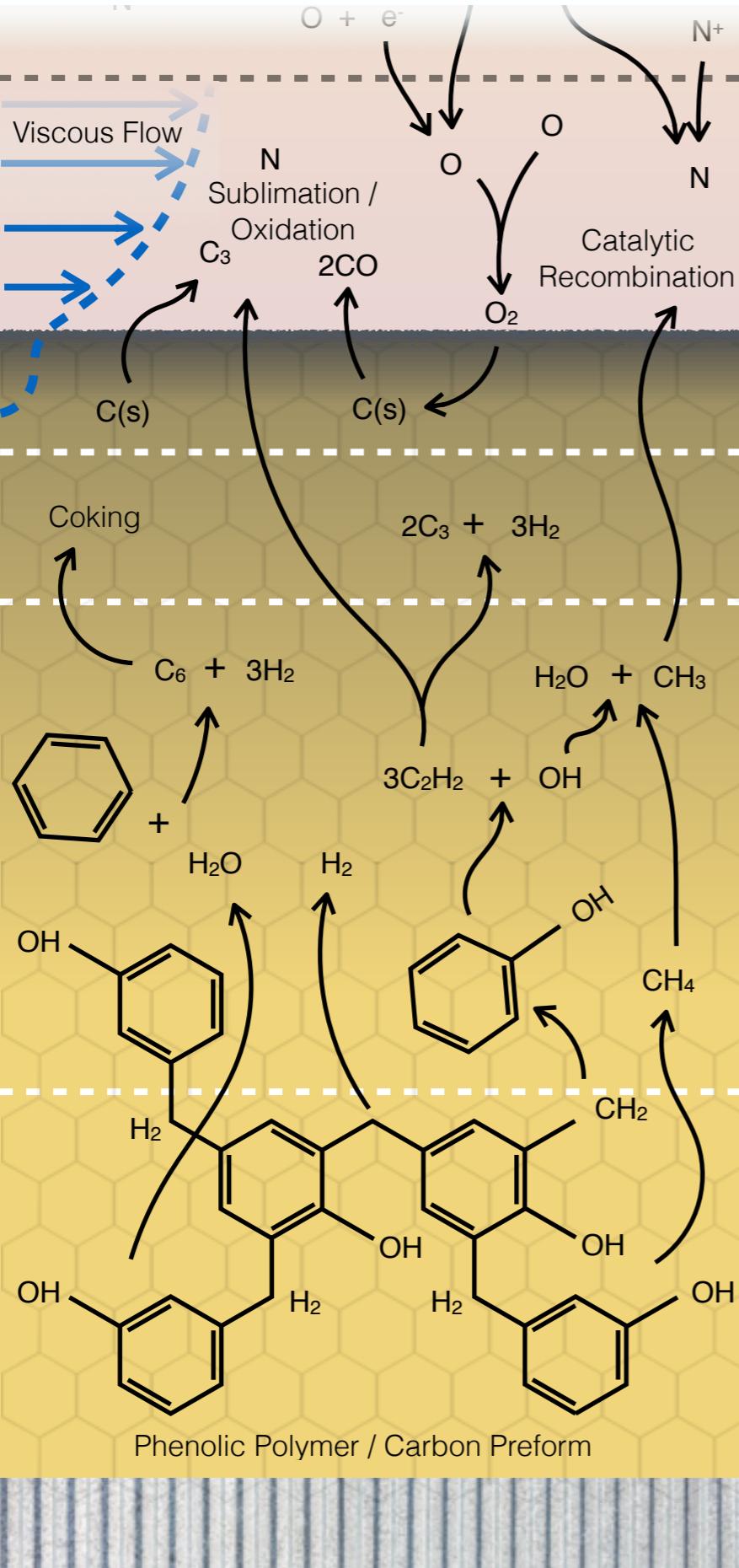
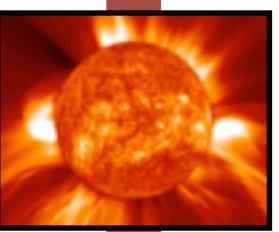
Thermal decomposition of the TPS



Thermal decomposition of the TPS



Thermal decomposition of the TPS



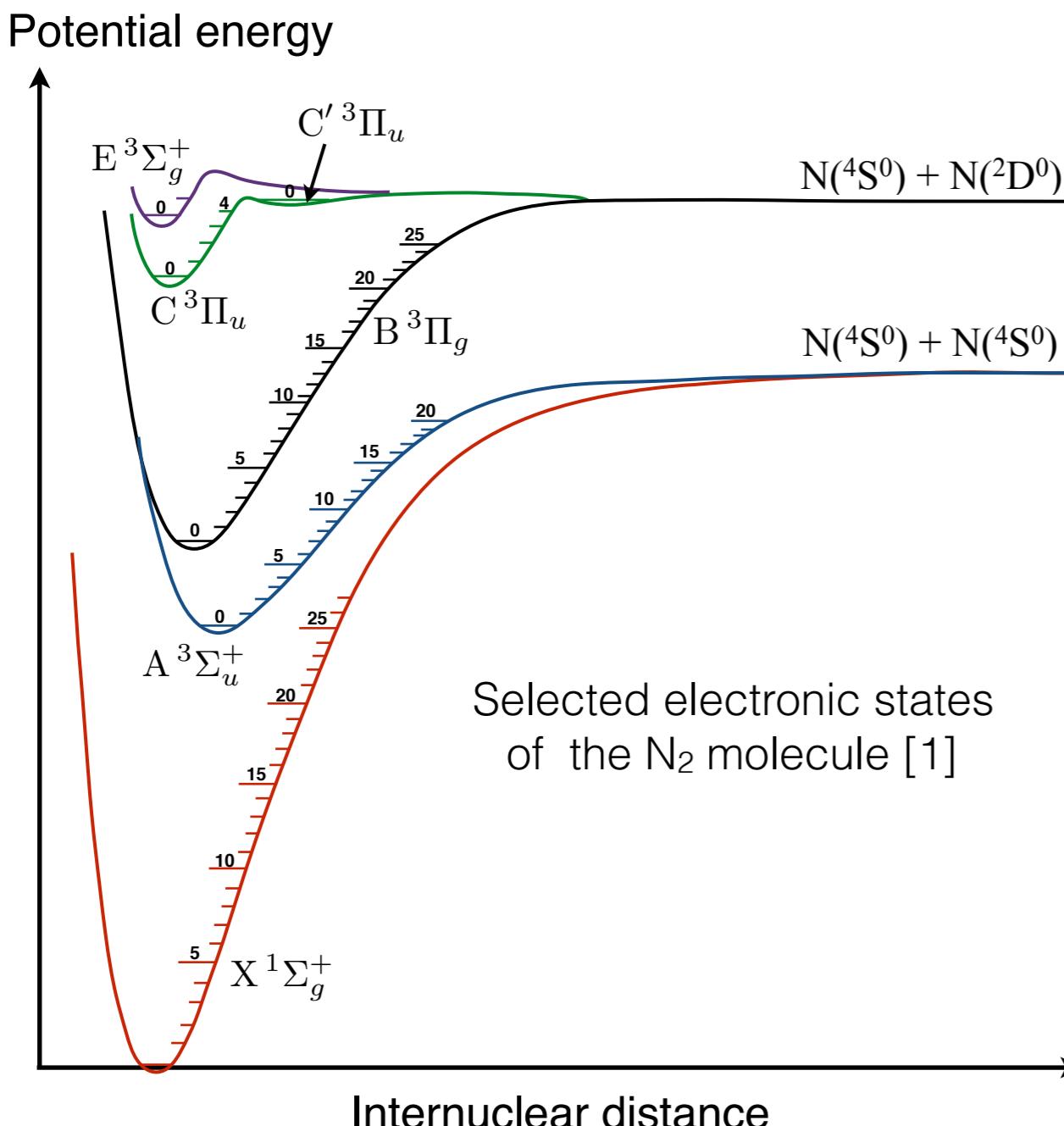
Section-cut micrographs of Asterm.



Micrographs of virgin PICA depicting carbon fibers impregnated with phenolic resin.

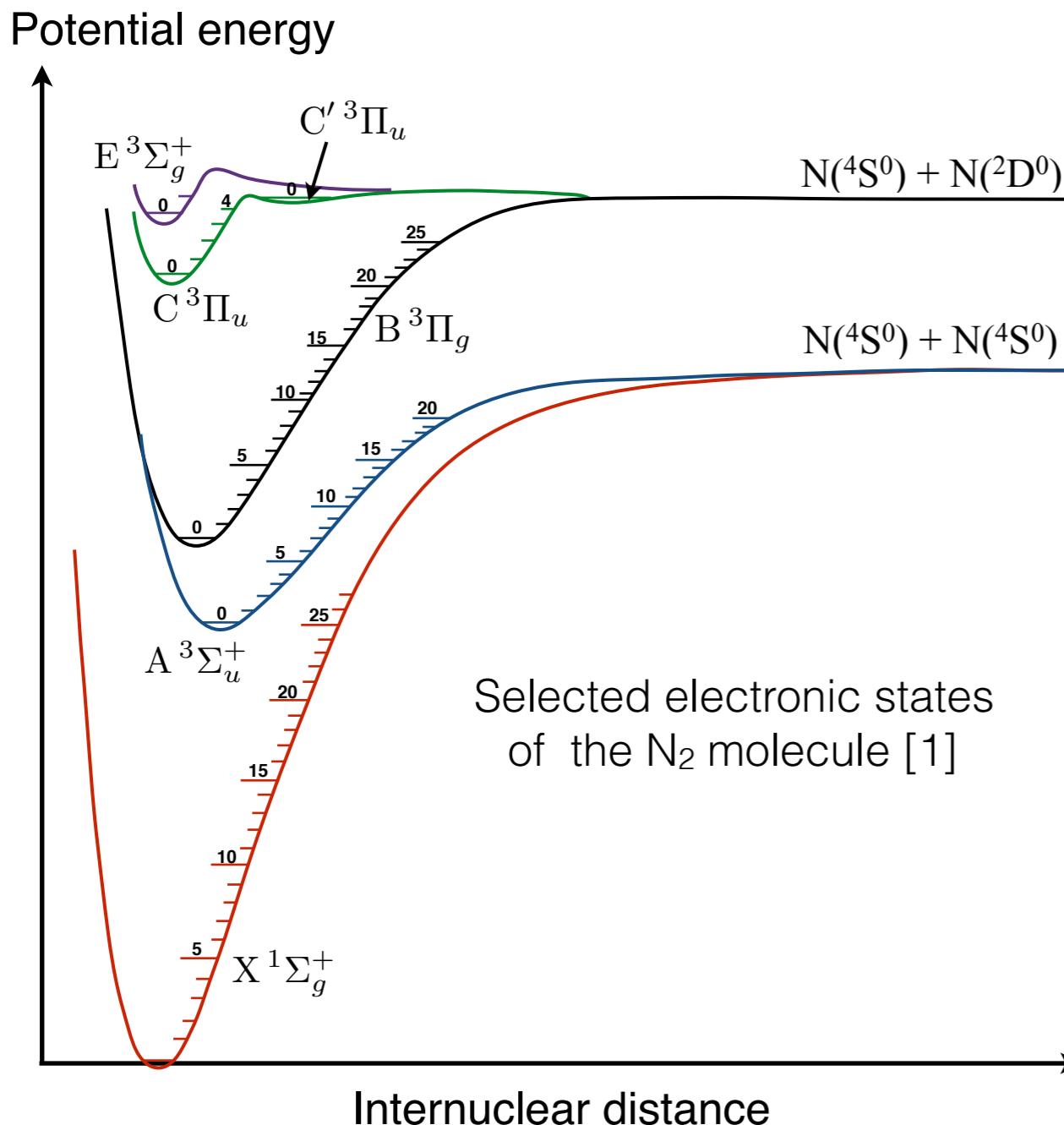
Energy partitioning regimes in hypersonics

Quantum mechanics dictates that atoms and molecule are permitted only **discrete energy levels**



Energy partitioning regimes in hypersonics

Quantum mechanics dictates that atoms and molecule are permitted only **discrete energy levels**



Thermal Equilibrium

$$\tau_{\text{kin}} \ll \tau_{\text{flow}}$$



Boltzmann Distribution

$$n_j^l = \frac{n_j}{Q_j^{\text{int}}(T)} a_j^l \exp \left(-\frac{E_j^l}{k_B T} \right)$$

Multitemperature

State-specific

Energy binning

State-to-state

Nonequilibrium

$$\tau_{\text{kin}} \approx \tau_{\text{flow}}$$

Motivation

Governing equations for hypersonic flows

Detailed balance yields the following relationships for the conservation of mass, momentum, and energy

Pseudo-species mass conservation

$$\frac{\partial}{\partial t}(\rho_k) + \nabla \cdot (\rho_k \mathbf{u}) + \nabla \cdot (\rho_k \mathbf{V}_k) = \dot{\omega}_k + \dot{\phi}_k, \quad \forall k \in \mathcal{S}^*$$

Mass continuity

$$\frac{\partial}{\partial t}(\rho) + \nabla \cdot (\rho \mathbf{u}) = 0$$

Momentum conservation

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot \bar{\Pi} + nq \mathbf{E}' + \mathbf{j} \times \mathbf{B}$$

Free-electron energy conservation

$$\frac{\partial}{\partial t}(\rho_e e_e) + \nabla \cdot (\rho_e e_e \mathbf{u}) = -p_e \nabla \cdot \mathbf{u} - \nabla \cdot \mathbf{q}_e + \Omega_e + \mathbf{j}_e \cdot \mathbf{E}' + \mathcal{P}_e$$

Internal energy conservation of mode m

$$\frac{\partial}{\partial t}(\rho e^m) + \nabla \cdot (\rho e^m \mathbf{u}) = -\nabla \cdot \mathbf{q}^m + \Omega^m + \mathcal{P}_m$$

Total energy conservation

$$\frac{\partial}{\partial t}(\rho E) + \nabla \cdot (\rho H \mathbf{u}) = \nabla \cdot (\bar{\Pi} \mathbf{u}) - \nabla \cdot \mathbf{q} + \mathcal{P} + \mathbf{j} \cdot \mathbf{E}'$$

Motivation

Governing equations for hypersonic flows

Detailed balance yields the following relationships for the conservation of mass, momentum, and energy

Pseudo-species mass conservation

$$\frac{\partial}{\partial t}(\rho_k) + \nabla \cdot (\rho_k \mathbf{u}) + \nabla \cdot (\rho_k \mathbf{V}_k) = \dot{\omega}_k + \dot{\phi}_k, \quad \forall k \in \mathcal{S}^*$$

Mass continuity

$$\frac{\partial}{\partial t}(\rho) + \nabla \cdot (\rho \mathbf{u}) = 0$$

Momentum conservation

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot \bar{\Pi} + nq \mathbf{E}' + \mathbf{j} \times \mathbf{B}$$

Free-electron energy conservation

$$\frac{\partial}{\partial t}(\rho_e e_e) + \nabla \cdot (\rho_e e_e \mathbf{u}) = -p_e \nabla \cdot \mathbf{u} - \nabla \cdot \mathbf{q}_e + \Omega_e + \mathbf{j}_e \cdot \mathbf{E}' + \mathcal{P}_e$$

Internal energy conservation of mode m

$$\frac{\partial}{\partial t}(\rho e^m) + \nabla \cdot (\rho e^m \mathbf{u}) = -\nabla \cdot \mathbf{q}^m + \Omega^m + \mathcal{P}_m$$

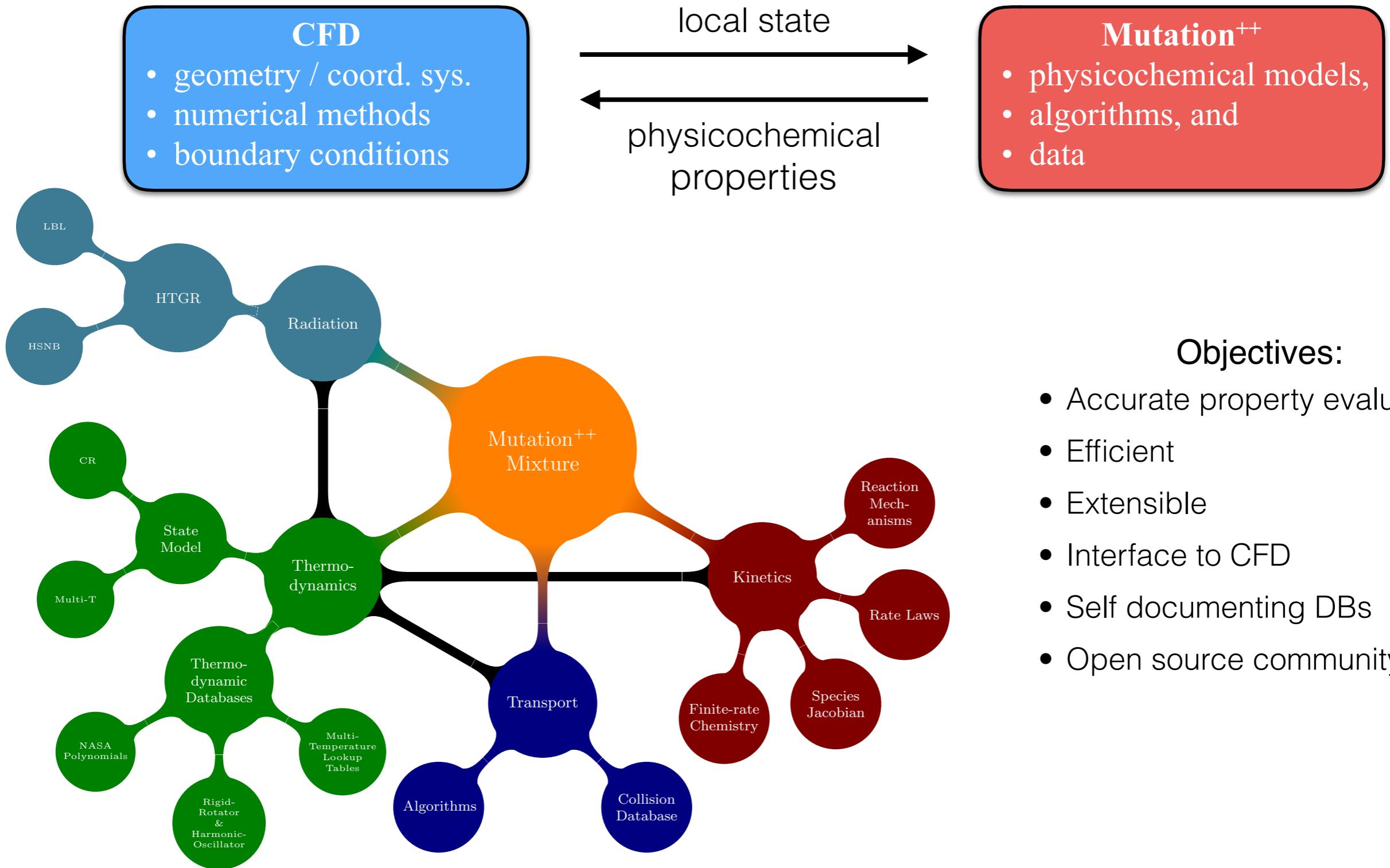
Total energy conservation

$$\frac{\partial}{\partial t}(\rho E) + \nabla \cdot (\rho H \mathbf{u}) = \nabla \cdot (\bar{\Pi} \mathbf{u}) - \nabla \cdot \mathbf{q} + \mathcal{P} + \mathbf{j} \cdot \mathbf{E}'$$

Requires knowledge of:

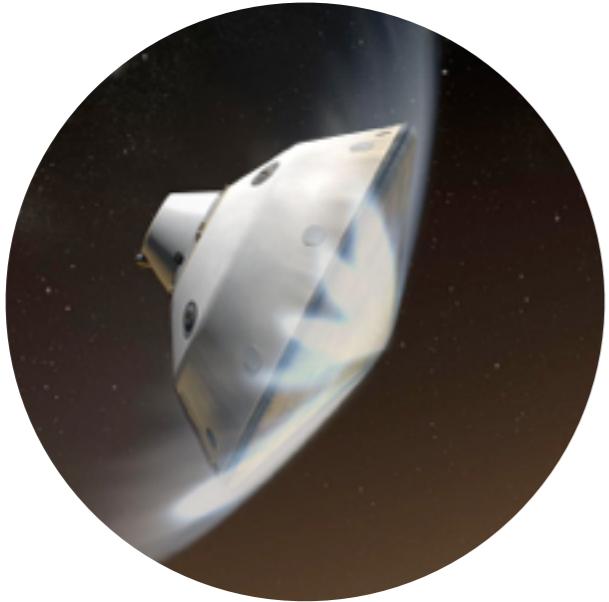
- Thermodynamic and transport properties
- Chemical kinetics
- Energy exchange mechanisms
- Radiative source terms

Development of Mutation++



Design allows for broad application domain

Atmospheric entry



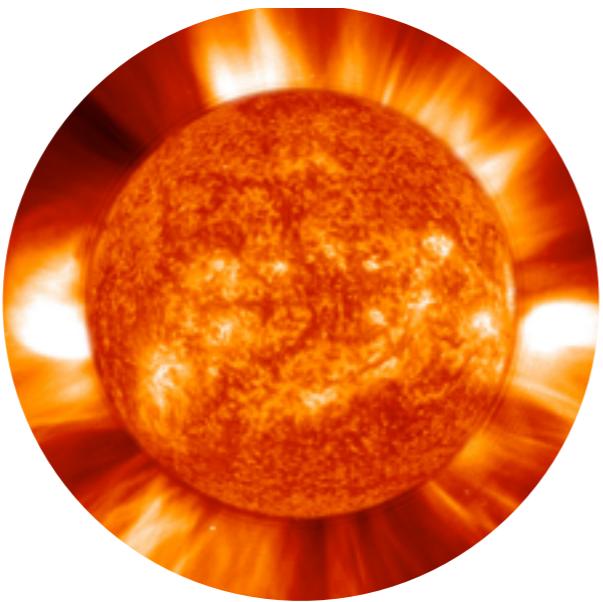
Magnetized plasmas



Meteor phenomena



Solar physics



Biomass pyrolysis



Flexible coupling strategies: “on-the-fly” or through tables

VKI Mutation++ Training Day

Friday, June 15, 2018

9:30 am	Welcome and overview of the library
9:50 am	Download and installation <ul style="list-style-type: none">• Using the Gitlab and wiki• Installing dependencies• Compiling• Running tests• Building the documentation
10:15 am	<i>Coffee break</i>
10:30 am	Discovering the built in tools of Mutation++ <ul style="list-style-type: none">• Theory: Gas mixtures, thermodynamics, equilibrium, finite-rate chemistry, and equilibrium surface chemistry (B')• Finding / modifying mixture files• <i>checkmix</i> – Check that a mixture has been formatted correctly• <i>mppequil</i> – Compute properties of a gas in equilibrium• <i>bprime</i> – Compute B' tables for a specific mixture and surface
12:15 pm	<i>Lunch</i>
1:15 pm	Coding with Mutation++* <ul style="list-style-type: none">• Setting up dependencies for an external project• Loading / creating a mixture in code• Implementing a 0D reactor with thermochemical nonequilibrium <p>* Examples will be provided in C++ and Fortran</p>
3:15 pm	<i>Coffee break</i>
3:30 pm	Coding with Mutation++ continued
4:00 pm	Discussion
4:30 pm	Closure

Let's get started...

Project Homepage

<https://sync.vki.ac.be/mpp/mutationpp>

Exercise

- 1) *Navigate to the project wiki and find the installation instructions.*
- 2) *Download and install the library.*

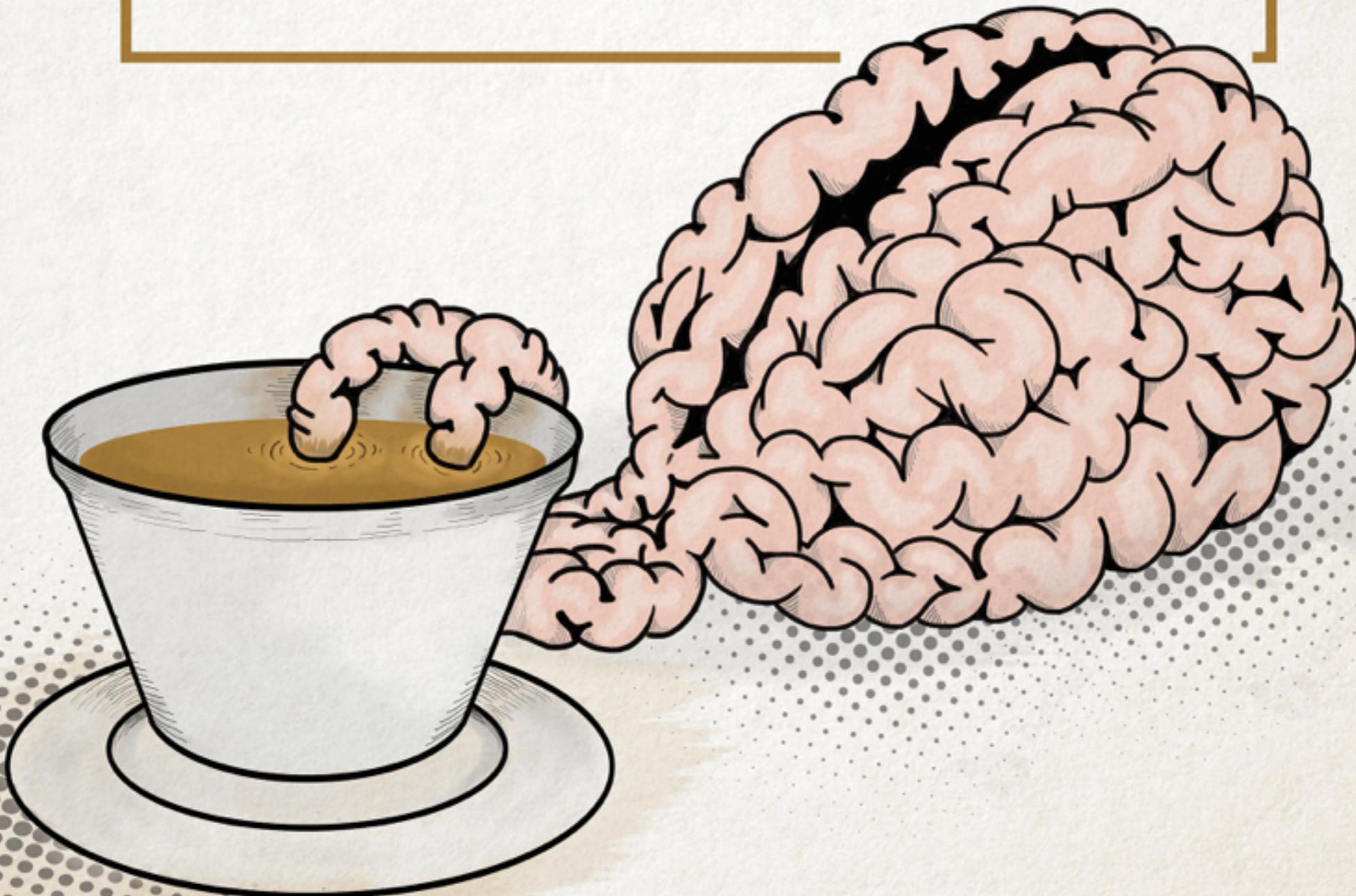
Testing wiki page

<https://sync.vki.ac.be/mpp/mutationpp/wikis/testing>

Exercise

Run the tests and verify that the library is correctly installed.

COFFEE BREAK



VKI Mutation++ Training Day

Friday, June 15, 2018

9:30 am	Welcome and overview of the library
9:50 am	Download and installation <ul style="list-style-type: none">• Using the Gitlab and wiki• Installing dependencies• Compiling• Running tests• Building the documentation
10:15 am	<i>Coffee break</i>
10:30 am	Discovering the built in tools of Mutation++ <ul style="list-style-type: none">• Theory: Gas mixtures, thermodynamics, equilibrium, finite-rate chemistry, and equilibrium surface chemistry (B')• Finding / modifying mixture files• <i>checkmix</i> – Check that a mixture has been formatted correctly• <i>mppequil</i> – Compute properties of a gas in equilibrium• <i>bprime</i> – Compute B' tables for a specific mixture and surface
12:15 pm	<i>Lunch</i>
1:15 pm	Coding with Mutation++* <ul style="list-style-type: none">• Setting up dependencies for an external project• Loading / creating a mixture in code• Implementing a 0D reactor with thermochemical nonequilibrium <p>* Examples will be provided in C++ and Fortran</p>
3:15 pm	<i>Coffee break</i>
3:30 pm	Coding with Mutation++ continued
4:00 pm	Discussion
4:30 pm	Closure

Locating and modifying data

Types and locations of database files.

Type	Data Directory
Mixture definitions	data/mixtures
Thermodynamic data	data/thermo
Transport data	data/transport
Reaction mechanisms	data/mechanisms
Energy transfer data	data/transfer

Search order for database files.

	Local Directory	Data Directory
Uncategorized	1	3
Categorized	2	4

Locating and modifying data

Types and locations of database files.

Type	Data Directory
Mixture definitions	data/mixtures
Thermodynamic data	data/thermo
Transport data	data/transport
Reaction mechanisms	data/mechanisms
Energy transfer data	data/transfer

Search order for database files.

	Local Directory	Data Directory
Uncategorized	1	3
Categorized	2	4

Locating and modifying data

Types and locations of database files.

Type	Data Directory
Mixture definitions	data/mixtures
Thermodynamic data	data/thermo
Transport data	data/transport
Reaction mechanisms	data/mechanisms
Energy transfer data	data/transfer

Exercise

- 1) Locate the “air5” mixture
- 2) Locate the reaction mechanism named in the mixture file.

Built-in Tool: checkmix

Loads a mixture and presents elements, species, and reactions in the mixture.

- ✓ Check if a mixture is found.
- ✓ Check exact location of the mixture file being loaded.
- ✓ Check if a mixture and corresponding support database files are formatted correctly.
- ✓ Check that no data is missing.
- ✓ Check which order the species and reactions are ordered internally.

Usage 1:

```
checkmix mixture
```

Usage 2:

```
checkmix [NASA-7 | NASA-9 | RRHO] species_list
```

Built-in Tool: checkmix

Loads a mixture and presents elements, species, and reactions in the mixture.

- ✓ Check if a mixture is found.
- ✓ Check exact location of the mixture file being loaded.
- ✓ Check if a mixture and corresponding support database files are formatted correctly.
- ✓ Check that no data is missing.
- ✓ Check which order the species and reactions are ordered internally.

Usage 1:

```
checkmix mixture
```

Usage 2:

```
checkmix [NASA-7 | NASA-9 | RRHO] species_list
```

Exercise

- 1) Run *checkmix* for “air5”.
- 2) Try the second usage with the same species as “air5”.

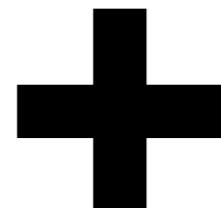
We are dealing with Mixtures

N₂ : 0.79
O₂ : 0.21
 $p = 1 \text{ atm}$

Perfect Gas Law

$$p = \rho RT$$

$$p_{\text{N}_2} = 0.79 \text{ atm}$$



$$p_{\text{O}_2} = 0.21 \text{ atm}$$

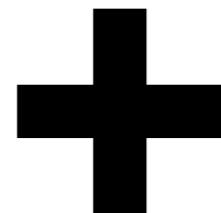
Dalton's Law

$$p = \sum_i^{n_s} p_i$$

We are dealing with Mixtures

$\text{N}_2 : 0.79$
 $\text{O}_2 : 0.21$
 $p = 1 \text{ atm}$

$p_{\text{N}_2} = 0.79 \text{ atm}$



$p_{\text{O}_2} = 0.21 \text{ atm}$

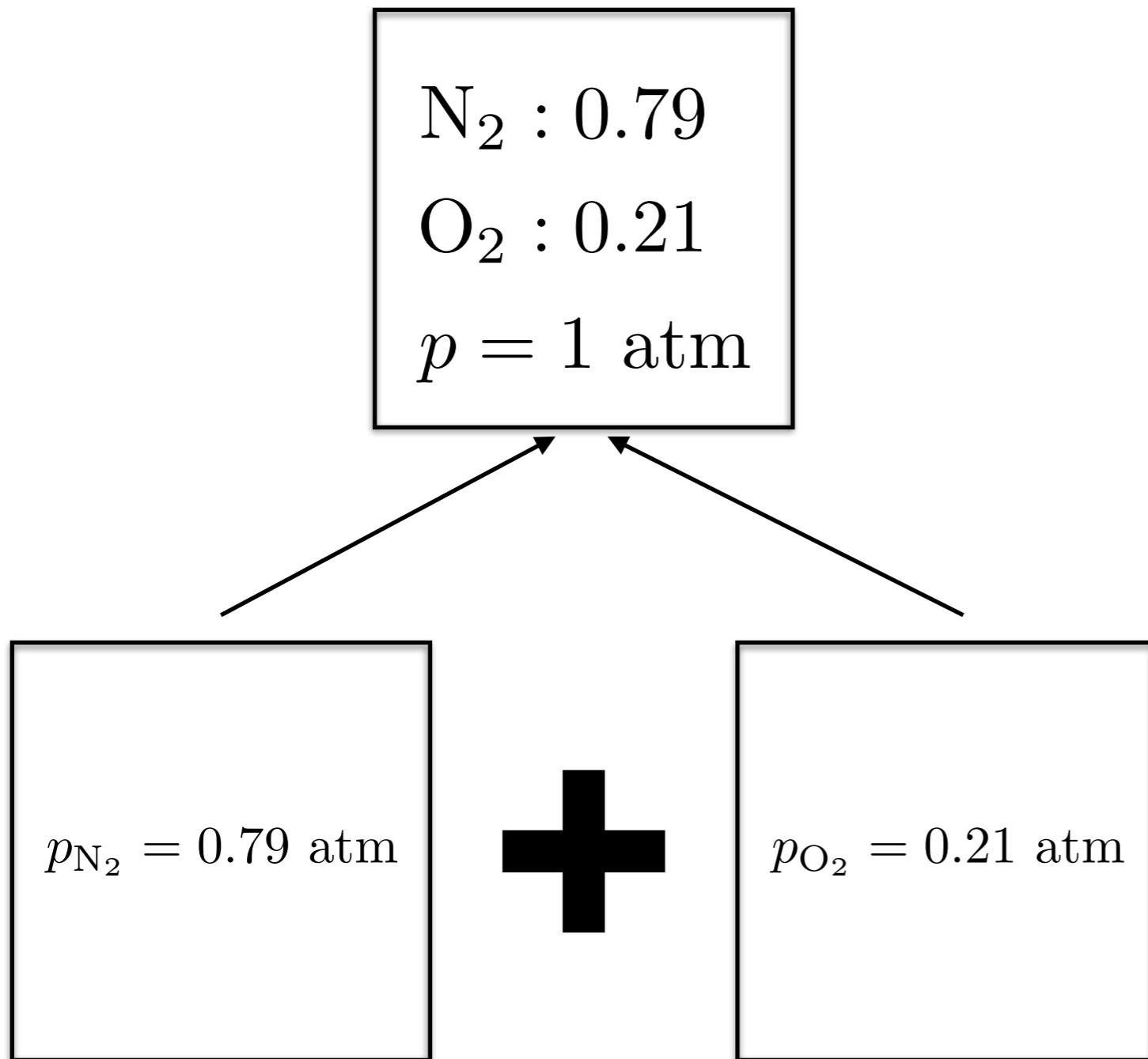
Perfect Gas Law

$$p = \rho R T$$

Dalton's Law

$$p = \sum_i^{n_s} p_i$$

We are dealing with Mixtures



Perfect Gas Law

$$p = \rho RT$$

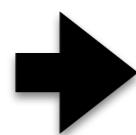
Dalton's Law

$$p = \sum_i^{n_s} p_i$$

Describing Chemical Mixtures

Mass Fractions

$$y_i = \frac{\rho_i}{\rho}$$



Mass of species i
Mass of mixture

$$\sum_i^{n_s} y_i = 1$$

Mole Fractions

$$x_i = \frac{n_i}{n}$$



Moles of species i
Moles of mixture

$$\sum_i^{n_s} x_i = 1$$

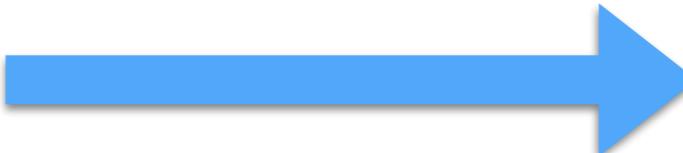
Elemental Mole Fraction

$$x_e = \frac{n_e}{n_{e,tot}}$$



Elements e
Total Elements

Species Properties



Mixture Properties

Describing Chemical Mixtures

Mass Fractions

$$y_i = \frac{\rho_i}{\rho}$$



Mass of species i
Mass of mixture

$$\sum_i^{n_s} y_i = 1$$

Mole Fractions

$$x_i = \frac{n_i}{n}$$



Moles of species i
Moles of mixture

$$\sum_i^{n_s} x_i = 1$$

Elemental Mole Fraction

$$x_e = \frac{n_e}{n_{e,tot}}$$



Elements e
Total Elements

Species Properties



Mixture Properties

Describing Chemical Mixtures

Mass Fractions

$$y_i = \frac{\rho_i}{\rho}$$



Mass of species i
Mass of mixture

$$\sum_i^{n_s} y_i = 1$$

Mole Fractions

$$x_i = \frac{n_i}{n}$$



Moles of species i
Moles of mixture

$$\sum_i^{n_s} x_i = 1$$

Elemental Mole Fraction

$$x_e = \frac{n_e}{n_{e,tot}}$$



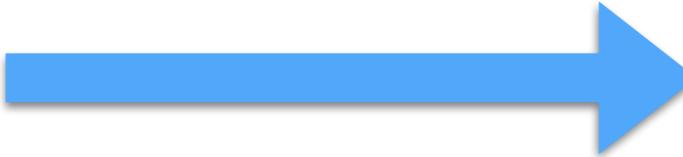
Elements e
Total Elements

Example:

$$x_{CO_2} = 0.8 \quad x_{CN} = 0.2$$

$$n_{e,tot} = 0.8 \cdot 1 + 0.8 \cdot 2 \quad CO_2 \\ + 0.2 \cdot 1 + 0.2 \cdot 1 \quad CN$$

Species Properties



Mixture Properties

Species Properties

NASA-9 Polynomials

- Computing properties based on polynomial fits

Enthalpy:

$$\frac{h_i}{R_i T} = -a_{0i} T^{-2} + a_{1i} \frac{\ln T}{T} + a_{2i} + \frac{a_{3i}}{2} T + \frac{a_{4i}}{3} T^2 + \frac{a_{5i}}{4} T^3 + \frac{a_{6i}}{5} T^4 + \frac{a_{7i}}{T}$$

Entropy:

$$\frac{s_i}{R_i} = -\frac{a_{0i}}{2} T^{-2} - a_{1i} T^{-1} + a_{2i} \ln T + a_{3i} T + \frac{a_{4i}}{2} T^2 + \frac{a_{5i}}{3} T^3 + \frac{a_{6i}}{4} T^4 + a_{8i}$$

Gibbs Free Energy (or any other thermodynamic property):

$$\frac{g_i}{R_i T} = \frac{s_i}{R_i T} - \frac{h_i}{T}$$

Rigid Rotator-Harmonic Oscillator

- Computing properties with analytical formulas (statistical thermodynamics)

e.g.: $e_i^{\text{trans}} = \frac{3}{2} R_i T$

Species to Mixture Properties

Energy and Enthalpy

$$e = \sum_{i=1}^{n_s} y_i e_i \quad \text{and} \quad h = \sum_{i=1}^{n_s} y_i h_i$$

Entropy

$$s = \sum_{i=1}^{n_s} y_i s_i - \frac{n k_B}{\rho} \sum_i x_i \ln x_i$$

Entropy of Mixing

Here all properties are in **kg**. When the properties are in **mole**, one should multiply with mole fractions!

Constrained multiphase equilibrium

A mixture is at equilibrium when the system Gibbs free energy is minimized, under mass balance constraints.

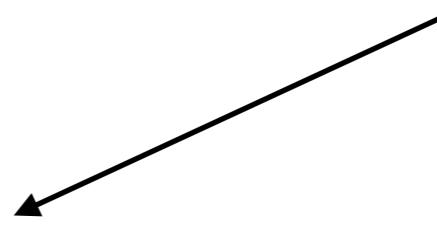
$$\tilde{G} \equiv \frac{G}{R_u T} = \sum_{m \in \mathcal{P}} \sum_{j \in \mathcal{S}_m} N_j (\tilde{g}_j + \ln N_j - \ln \bar{N}_m)$$

$$\bar{N}_m = \sum_{j \in \mathcal{S}_m} N_j, \quad \forall m \in \mathcal{P}$$

$$\sum_{j \in \mathcal{S}} B_{ji}^e N_j = c_i^e \quad \forall i \in \mathcal{E}$$

Constrained multiphase equilibrium

A mixture is at equilibrium when the system Gibbs free energy is minimized, under mass balance constraints.



$$\tilde{G} \equiv \frac{G}{R_u T} = \sum_{m \in \mathcal{P}} \sum_{j \in \mathcal{S}_m} N_j (\tilde{g}_j + \ln N_j - \ln \bar{N}_m)$$

$$\bar{N}_m = \sum_{j \in \mathcal{S}_m} N_j, \quad \forall m \in \mathcal{P}$$

$$\sum_{j \in \mathcal{S}} B_{ji}^e N_j = c_i^e \quad \forall i \in \mathcal{E}$$

Constrained multiphase equilibrium

A mixture is at equilibrium when the system Gibbs free energy is minimized, under mass balance constraints.

$$\tilde{G} \equiv \frac{G}{R_u T} = \sum_{m \in \mathcal{P}} \sum_{j \in \mathcal{S}_m} N_j (\tilde{g}_j + \ln N_j - \ln \bar{N}_m)$$

$$\bar{N}_m = \sum_{j \in \mathcal{S}_m} N_j, \quad \forall m \in \mathcal{P}$$

$$\sum_{j \in \mathcal{S}} B_{ji}^e N_j = c_i^e \quad \forall i \in \mathcal{E}$$

Constrained multiphase equilibrium

A mixture is at equilibrium when the system Gibbs free energy is minimized, under mass balance constraints.

$$\tilde{G} \equiv \frac{G}{R_u T} = \sum_{m \in \mathcal{P}} \sum_{j \in \mathcal{S}_m} N_j (\tilde{g}_j + \ln N_j - \ln \bar{N}_m)$$

$$\bar{N}_m = \sum_{j \in \mathcal{S}_m} N_j, \quad \forall m \in \mathcal{P}$$

$$\sum_{j \in \mathcal{S}} B_{ji}^e N_j = c_i^e \quad \forall i \in \mathcal{E}$$

Species	B columns			P columns	
	C	O	\bar{N}_{mix}	Gas	C(gr)
C	1	0	1	1	0
CO	1	1	1	1	0
CO ₂	1	2	1	1	0
O ₂	0	2	1	1	0
C(gr)	1	0	1	0	1

Constrained multiphase equilibrium

A mixture is at equilibrium when the system Gibbs free energy is minimized, under mass balance constraints.

$$\begin{aligned} \tilde{G} &\equiv \frac{G}{R_u T} = \sum_{m \in \mathcal{P}} \sum_{j \in \mathcal{S}_m} N_j (\tilde{g}_j + \ln N_j - \ln \bar{N}_m) \\ \bar{N}_m &= \sum_{j \in \mathcal{S}_m} N_j, \quad \forall m \in \mathcal{P} \\ \sum_{j \in \mathcal{S}} B_{ji}^e N_j &= c_i^e \quad \forall i \in \mathcal{E} \end{aligned}$$

Can use the Lagrange multiplier technique to solve constrained minimization problem.

$$\begin{aligned} L &= \tilde{G} - \sum_{i \in \mathcal{C}} \lambda_i \left(\sum_{j \in \mathcal{S}} B_{ji} N_j - c_i \right) \\ &= \sum_{j \in \mathcal{S}} N_j \left(\tilde{g}_j + \ln N_j - \ln \sum_{k \in \mathcal{S}_{p_j}} N_k \right) - \sum_{i \in \mathcal{C}} \lambda_i \left(\sum_{j \in \mathcal{S}} B_{ji} N_j - c_i \right) \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial N_j} &= \frac{\partial \tilde{G}}{\partial N_j} - \sum_{i \in \mathcal{C}} \lambda_i B_{ji} \\ &= \tilde{g}_j + \ln N_j - \ln \sum_{k \in \mathcal{S}_{p_j}} N_k - \sum_{i \in \mathcal{C}} \lambda_i B_{ji} = 0, \quad \forall j \in \mathcal{S}. \end{aligned}$$

Constrained multiphase equilibrium

A mixture is at equilibrium when the system Gibbs free energy is minimized, under mass balance constraints.

$$\begin{aligned} \tilde{G} &\equiv \frac{G}{R_u T} = \sum_{m \in \mathcal{P}} \sum_{j \in \mathcal{S}_m} N_j (\tilde{g}_j + \ln N_j - \ln \bar{N}_m) \\ \bar{N}_m &= \sum_{j \in \mathcal{S}_m} N_j, \quad \forall m \in \mathcal{P} \\ \sum_{j \in \mathcal{S}} B_{ji}^e N_j &= c_i^e \quad \forall i \in \mathcal{E} \end{aligned}$$

Can use the Lagrange multiplier technique to solve constrained minimization problem.

$$\begin{aligned} L &= \tilde{G} - \sum_{i \in \mathcal{C}} \lambda_i \left(\sum_{j \in \mathcal{S}} B_{ji} N_j - c_i \right) && \text{general linear constraints} \\ &= \sum_{j \in \mathcal{S}} N_j \left(\tilde{g}_j + \ln N_j - \ln \sum_{k \in \mathcal{S}_{p_j}} N_k \right) - \sum_{i \in \mathcal{C}} \lambda_i \left(\sum_{j \in \mathcal{S}} B_{ji} N_j - c_i \right) \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial N_j} &= \frac{\partial \tilde{G}}{\partial N_j} - \sum_{i \in \mathcal{C}} \lambda_i B_{ji} \\ &= \tilde{g}_j + \ln N_j - \ln \sum_{k \in \mathcal{S}_{p_j}} N_k - \sum_{i \in \mathcal{C}} \lambda_i B_{ji} = 0, \quad \forall j \in \mathcal{S}. \end{aligned}$$

Constrained multiphase equilibrium

A mixture is at equilibrium when the system Gibbs free energy is minimized, under mass balance constraints.

$$\begin{aligned}\tilde{G} &\equiv \frac{G}{R_u T} = \sum_{m \in \mathcal{P}} \sum_{j \in \mathcal{S}_m} N_j (\tilde{g}_j + \ln N_j - \ln \bar{N}_m) \\ \bar{N}_m &= \sum_{j \in \mathcal{S}_m} N_j, \quad \forall m \in \mathcal{P} \\ \sum_{j \in \mathcal{S}} B_{ji}^e N_j &= c_i^e \quad \forall i \in \mathcal{E}\end{aligned}$$

Can use the Lagrange multiplier technique to solve constrained minimization problem.

$$\begin{aligned}L &= \tilde{G} - \sum_{i \in \mathcal{C}} \lambda_i \left(\sum_{j \in \mathcal{S}} B_{ji} N_j - c_i \right) && \text{general linear constraints} \\ &= \sum_{j \in \mathcal{S}} N_j \left(\tilde{g}_j + \ln N_j - \ln \sum_{k \in \mathcal{S}_{p_j}} N_k \right) - \sum_{i \in \mathcal{C}} \lambda_i \left(\sum_{j \in \mathcal{S}} B_{ji} N_j - c_i \right)\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial N_j} &= \frac{\partial \tilde{G}}{\partial N_j} - \sum_{i \in \mathcal{C}} \lambda_i B_{ji} \\ &= \tilde{g}_j + \ln N_j - \ln \sum_{k \in \mathcal{S}_{p_j}} N_k - \sum_{i \in \mathcal{C}} \lambda_i B_{ji} = 0, \quad \forall j \in \mathcal{S}.\end{aligned}$$

necessary conditions
for minimization

Constrained multiphase equilibrium

A mixture is at equilibrium when the system Gibbs free energy is minimized, under mass balance constraints.

$$\tilde{G} \equiv \frac{G}{R_u T} = \sum_{m \in \mathcal{P}} \sum_{j \in \mathcal{S}_m} N_j (\tilde{g}_j + \ln N_j - \ln \bar{N}_m)$$

$\bar{N}_m = \sum_{j \in \mathcal{S}_m} N_j, \quad \forall m \in \mathcal{P}$ $\sum_{j \in \mathcal{S}} B_{ji}^e N_j = c_i^e \quad \forall i \in \mathcal{E}$

yields small non linear system
of equations to be solved

Can use the Lagrange multiplier technique to solve constrained minimization problem.

$$L = \tilde{G} - \sum_{i \in \mathcal{C}} \lambda_i \left(\sum_{j \in \mathcal{S}} B_{ji} N_j - c_i \right)$$

general linear
constraints

$$= \sum_{j \in \mathcal{S}} N_j \left(\tilde{g}_j + \ln N_j - \ln \sum_{k \in \mathcal{S}_{p_j}} N_k \right) - \sum_{i \in \mathcal{C}} \lambda_i \left(\sum_{j \in \mathcal{S}} B_{ji} N_j - c_i \right)$$

necessary conditions
for minimization

$$\frac{\partial L}{\partial N_j} = \frac{\partial \tilde{G}}{\partial N_j} - \sum_{i \in \mathcal{C}} \lambda_i B_{ji}$$
$$= \tilde{g}_j + \ln N_j - \ln \sum_{k \in \mathcal{S}_{p_j}} N_k - \sum_{i \in \mathcal{C}} \lambda_i B_{ji} = 0, \quad \forall j \in \mathcal{S}.$$

Multiphase Gibbs function continuation

Minimization of Gibbs energy under constraints

[1] Pope. *Combustion and Flame*, 139(3):222–226, 2004.

[2] Scoggins, Magin. *Combustion and Flame*, 162(12):4514–4522, 2015.

Multiphase Gibbs function continuation

Minimization of Gibbs energy under constraints

Solution of nonlinear system for phase moles
and constraint potentials

[1] Pope. *Combustion and Flame*, 139(3):222–226, 2004.

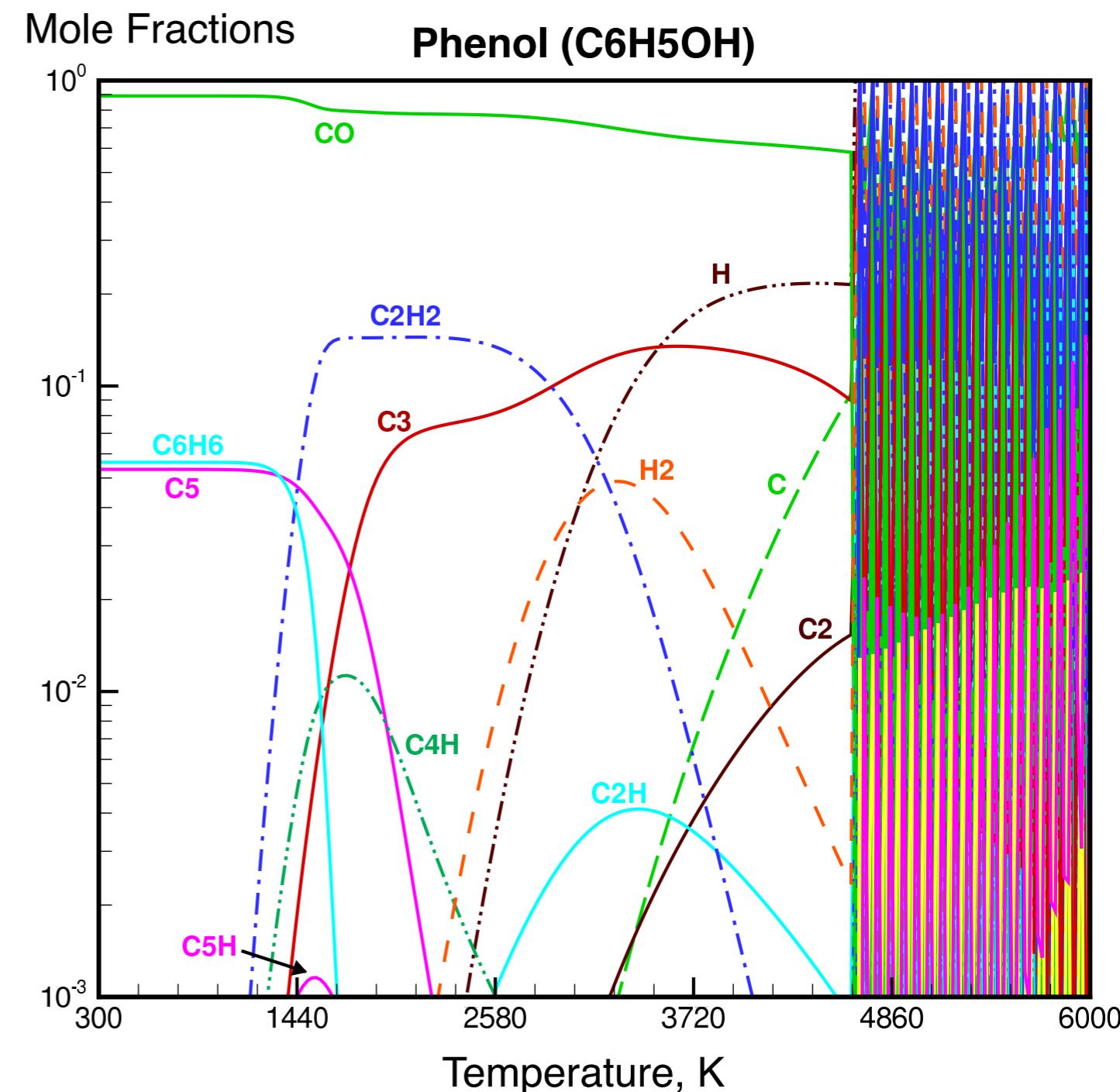
[2] Scoggins, Magin. *Combustion and Flame*, 162(12):4514–4522, 2015.

Multiphase Gibbs function continuation

Minimization of Gibbs energy under constraints

Solution of nonlinear system for phase moles
and constraint potentials

- Newton iterations may fail to converge



[1] Pope. *Combustion and Flame*, 139(3):222–226, 2004.

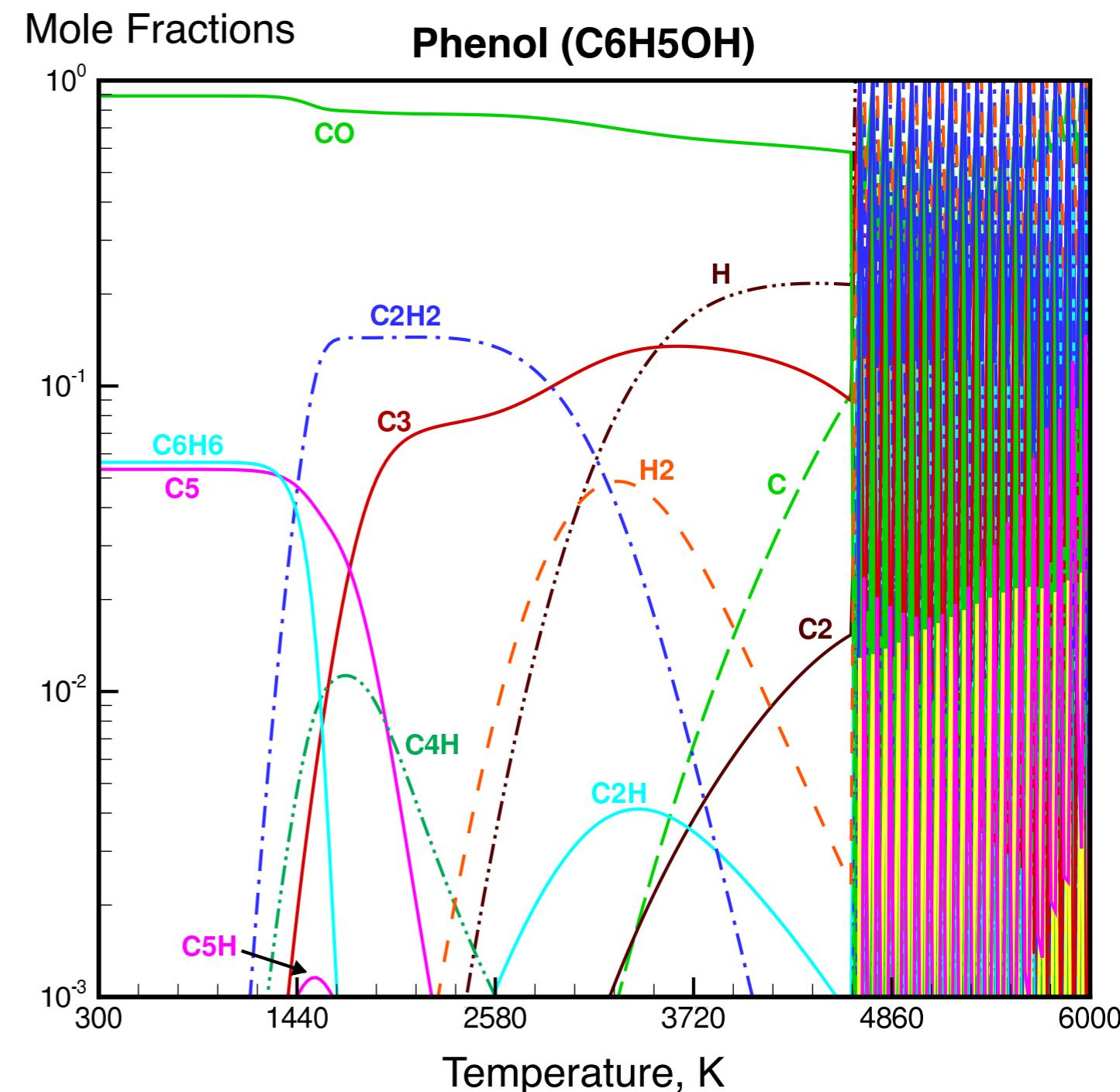
[2] Scoggins, Magin. *Combustion and Flame*, 162(12):4514–4522, 2015.

Multiphase Gibbs function continuation

Minimization of Gibbs energy under constraints

Solution of nonlinear system for phase moles and constraint potentials

- Newton iterations may fail to converge
- Pope [1] introduced Gibbs function continuation (GFC) method for one phase



[1] Pope. *Combustion and Flame*, 139(3):222–226, 2004.

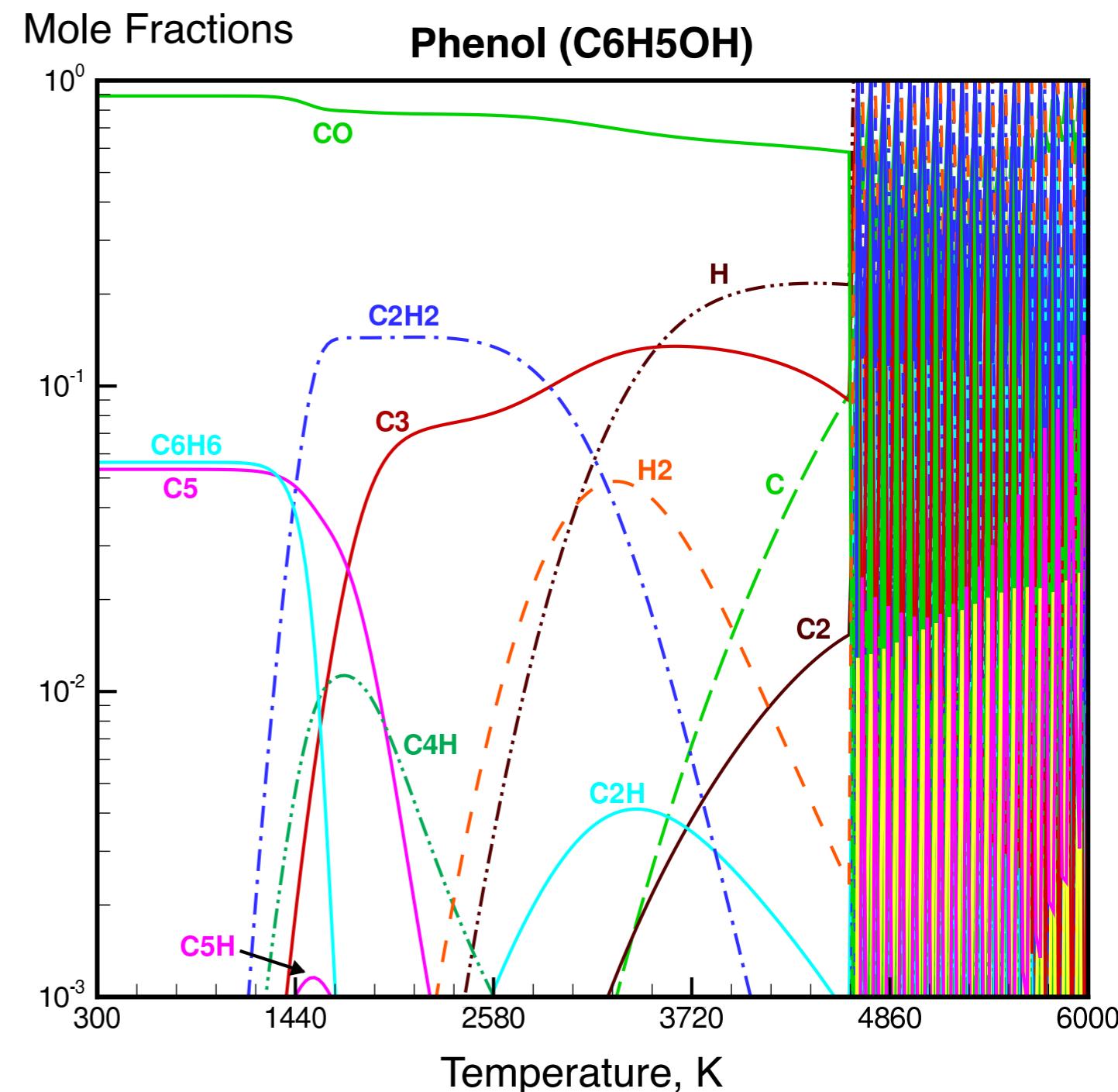
[2] Scoggins, Magin. *Combustion and Flame*, 162(12):4514–4522, 2015.

Multiphase Gibbs function continuation

Minimization of Gibbs energy under constraints

Solution of nonlinear system for phase moles and constraint potentials

- Newton iterations may fail to converge
- Pope [1] introduced Gibbs function continuation (GFC) method for one phase
- Have extended Popes formulation to multiple phases (MPGFC) in [2]



[1] Pope. *Combustion and Flame*, 139(3):222–226, 2004.

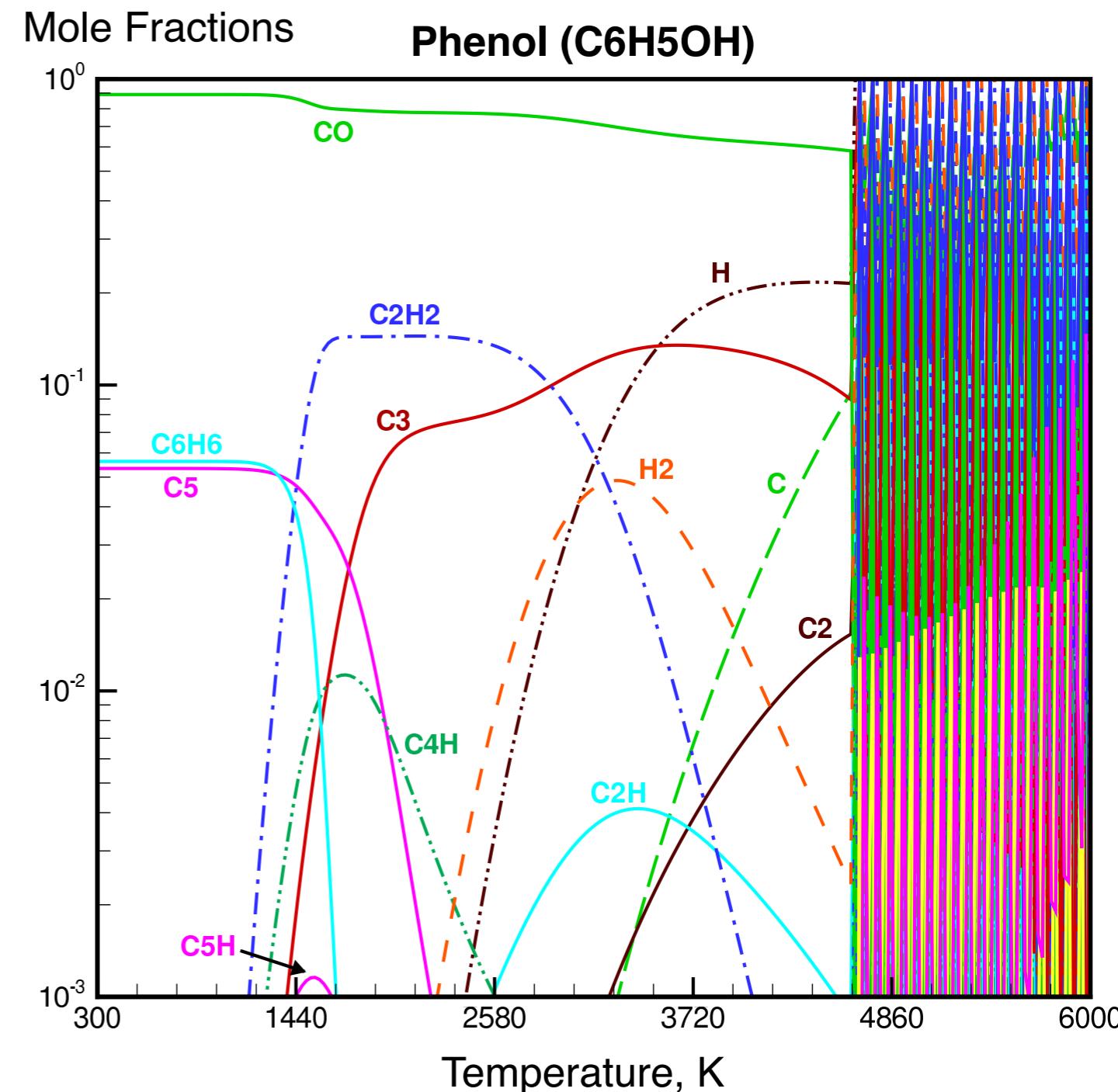
[2] Scoggins, Magin. *Combustion and Flame*, 162(12):4514–4522, 2015.

Multiphase Gibbs function continuation

Minimization of Gibbs energy under constraints

Solution of nonlinear system for phase moles and constraint potentials

- Newton iterations may fail to converge
- Pope [1] introduced Gibbs function continuation (GFC) method for one phase
- Have extended Popes formulation to multiple phases (MPGFC) in [2]
- Proof of guaranteed convergence when constraints are valid
- Showed that Pope convergence proof is a special case of the general proof



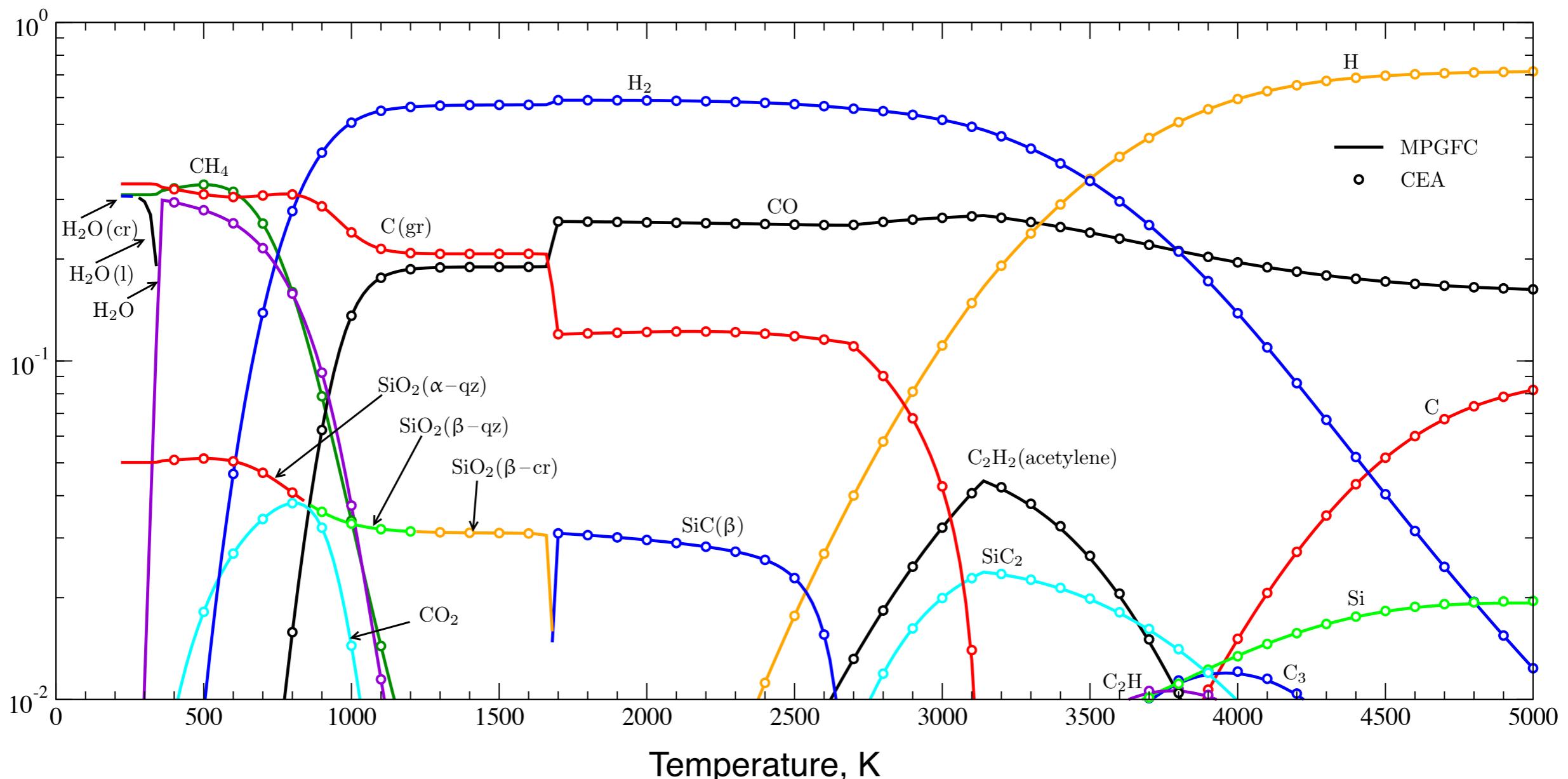
[1] Pope. *Combustion and Flame*, 139(3):222–226, 2004.

[2] Scoggins, Magin. *Combustion and Flame*, 162(12):4514–4522, 2015.

Multiphase Gibbs function continuation

Equilibrium mixture of 90% phenol (C_6H_5OH) with 10% silicon

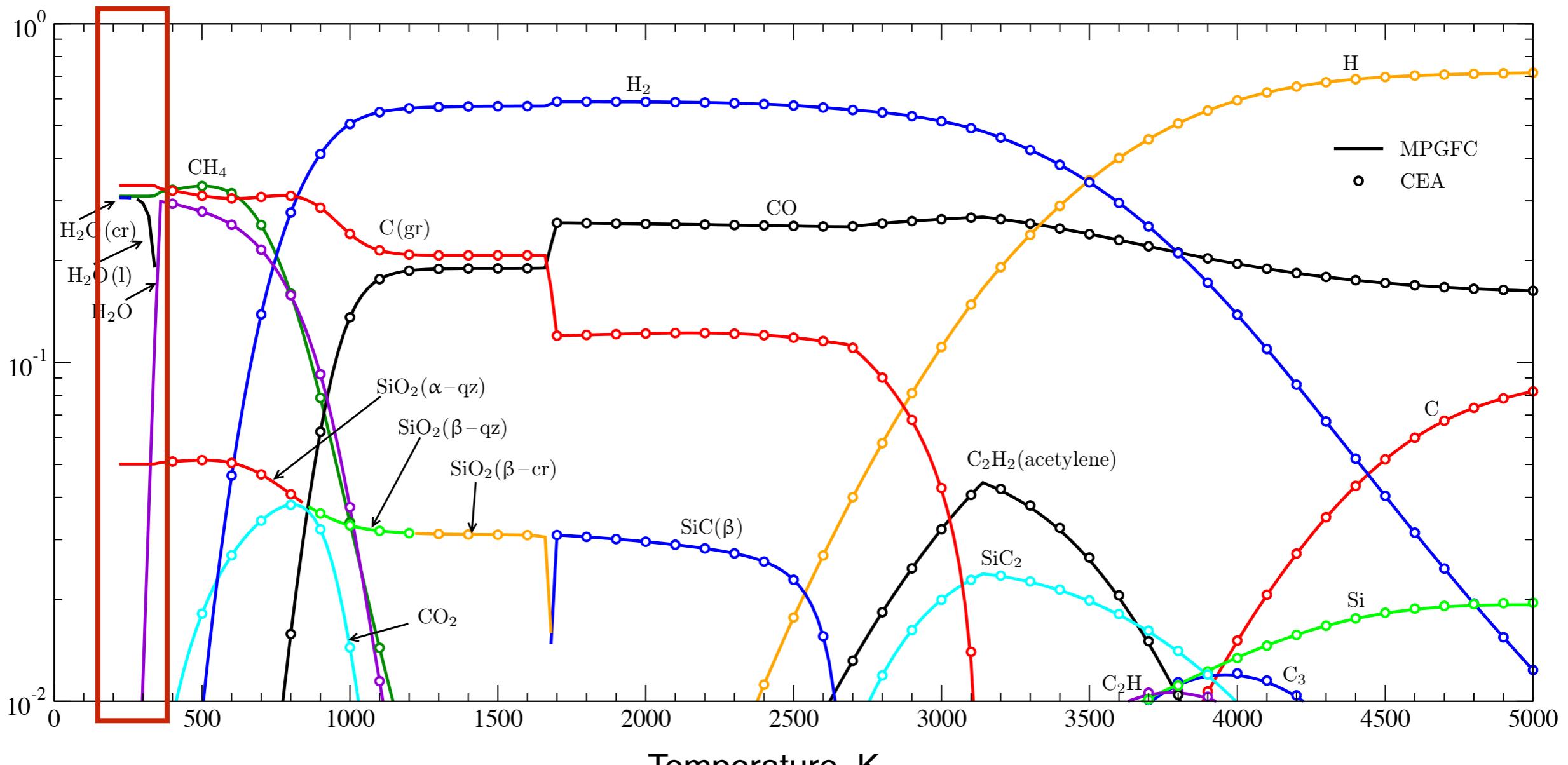
Global mole fractions [1]



Multiphase Gibbs function continuation

Equilibrium mixture of 90% phenol (C_6H_5OH) with 10% silicon

Global mole fractions [1]



NASA CEA solver fails
to converge at low
temperatures

Built-in Tool: mppequil

Generates a table of equilibrium properties for a given mixture and conditions.

Usage:

```
mppequil [options] mixture
```

Help:

```
mppequil -h
```

Built-in Tool: mppequil

Generates a table of equilibrium properties for a given mixture and conditions.

Usage:

```
mppequil [options] mixture
```

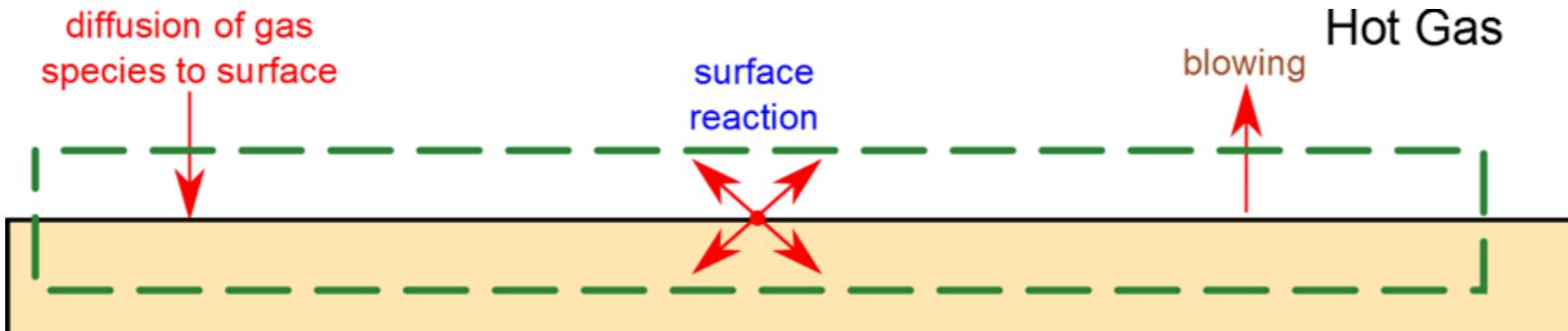
Help:

```
mppequil -h
```

Exercise

- 1) *Check out the help for mppequil.*
- 2) *Compute the equilibrium mole fractions for 5-species air from 500K to 8000K at 1atm.*

Surface mass balance



- Species i mass balance:

$$J_{i,w} + \dot{m}_{evap,i} = (\rho v)_w y_{i,w} \quad i=1, \dots, N_s$$

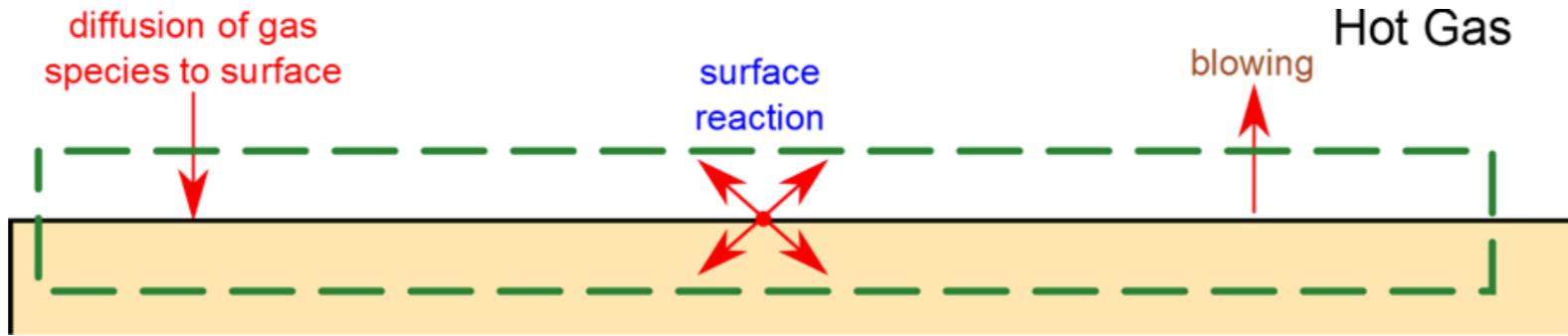
- Element k mass balance:

$$\sum_{i=1}^{N_s} \sigma_{i,k} \frac{M_k}{M_i} (1) \Rightarrow J_{k,w} + \dot{m} y_{k,s} = (\rho v)_w y_{k,w} \quad k=1, \dots, \epsilon$$

- Summing over all elements:

$$\sum_{k=1}^{\epsilon} (J_{k,w} + \dot{m} y_{k,s} = (\rho v)_w y_{k,w}) \Rightarrow \dot{m} = (\rho v)_w$$

Nondimensional mass blowing rates



- From the Couette flow we obtain:

$$J_{k,w} = \rho_e u_e C_m (y_{k,w} - y_{e,k})$$

- Replacing into the elemental SMB:

$$\rho_e u_e C_m (y_{k,w} - y_{e,k}) + \dot{m} y_{k,s} = (\rho v)_w y_{k,w}$$

$$y_{k,w} - y_{e,k} + B' y_{k,s} = B' y_{k,w}$$

$$B' = \frac{y_{k,w} - y_{k,e}}{y_{k,s} - y_{k,w}}$$

$$B' = \frac{\dot{m}}{\rho_e u_e C_m}$$

- With pyrolysis:

$$B' = \frac{B'_g (y_{y,g} - y_{k,w}) - y_{k,w} + y_{k,e}}{y_{k,w} - y_{k,s}}$$

Built-in Tool: bprime

Usage:

```
bprime T1 T2 dT P bg mixture BL_comp Pyro_comp
```

bprime : exec

T1 T2 dT : temperature range (T1:dT:T2), eg: 300:10:4000

P: Pressure in atm, eg: 1

bg: non-dimensional pyrolysis blowing rate, eg: 100

mixture: mixture name, eg: carbonPhenol

```
<mixture thermo_db="NASA-9" state_model="EquilTP" mechanism="none">

  <species>
    C(gr) C H O N CH CH4 CO CO2 CN C2H C2H2,vinylidene C3 C3H C4 C4H
    C4H2,butadiyne C5 HCN H2 H2O N2 </species>

  <element_compositions>
    <composition name="Gas"> C:0.375, H:0.162, O:0.463, N:0. </composition>
    <composition name="BLedge"> C:0., H:0., O:0.21, N:0.79 </composition>
    <composition name="Char"> C:1., H:0., O:0., N:0. </composition>
  </element_compositions>
</mixture>
```

Built-in Tool: bprime

BL_comp: Boundary layer composition, eg: BLedge

```
<composition name="BLedge"> C:0., H:0., O:0.21, N:0.79 </composition>
```

Pyro_comp: Pyrolysis composition, eg: Gas

```
<composition name="Gas"> C:0.375, H:0.162, O:0.463, N:0. </composition>
```

Example of usage:

```
bprime 300 4000 10 1 100 carbonPhenol BLedge Gas > name_of_file.dat
```

Species: C(gr) C H O N CH CH₄ CO CO₂ CN C₂H C₂H₂,vinylidene C₃ C₃H C₄ C₄H C₄H₂,butadiyne C₅ HCN H₂ H₂O N₂

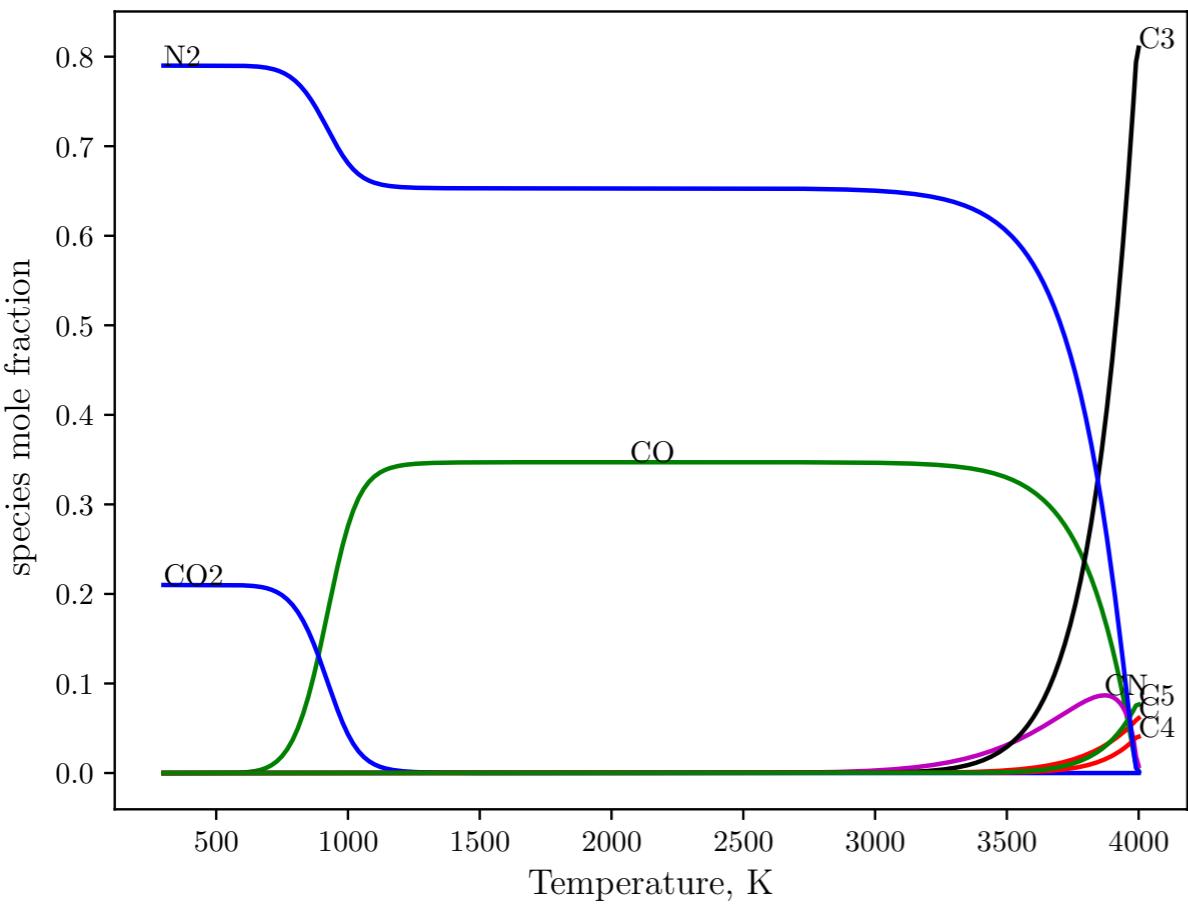
Gas composition: C:0.0, H:0.0, O:0.21, N:0.79

Char composition: C:1.0, H:0.0, O:0.0, N:0.

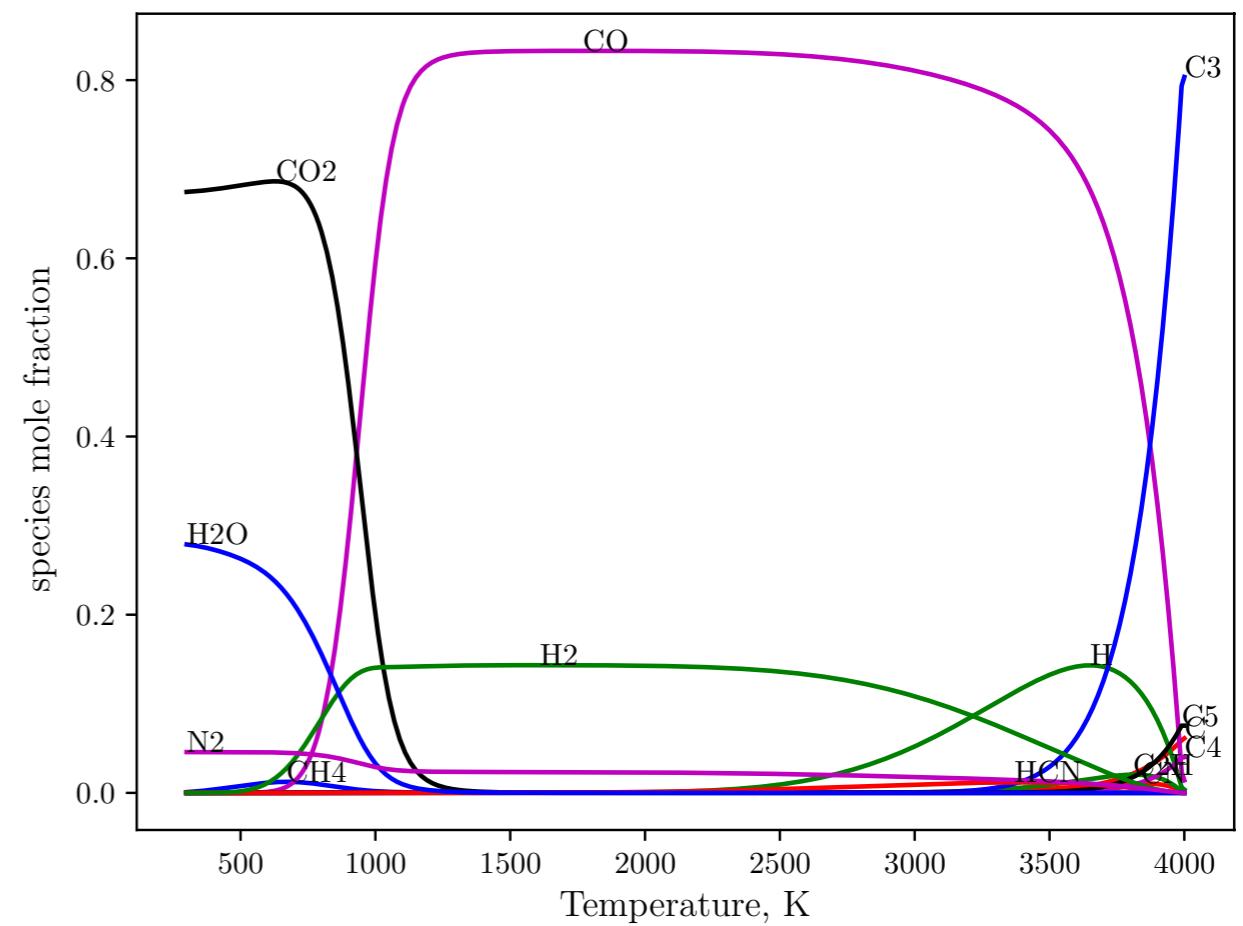
Pyrolysis composition: C:0.375, H:0.162, O:0.463, N:0.

Equilibrium composition with temperature at 1 atm:

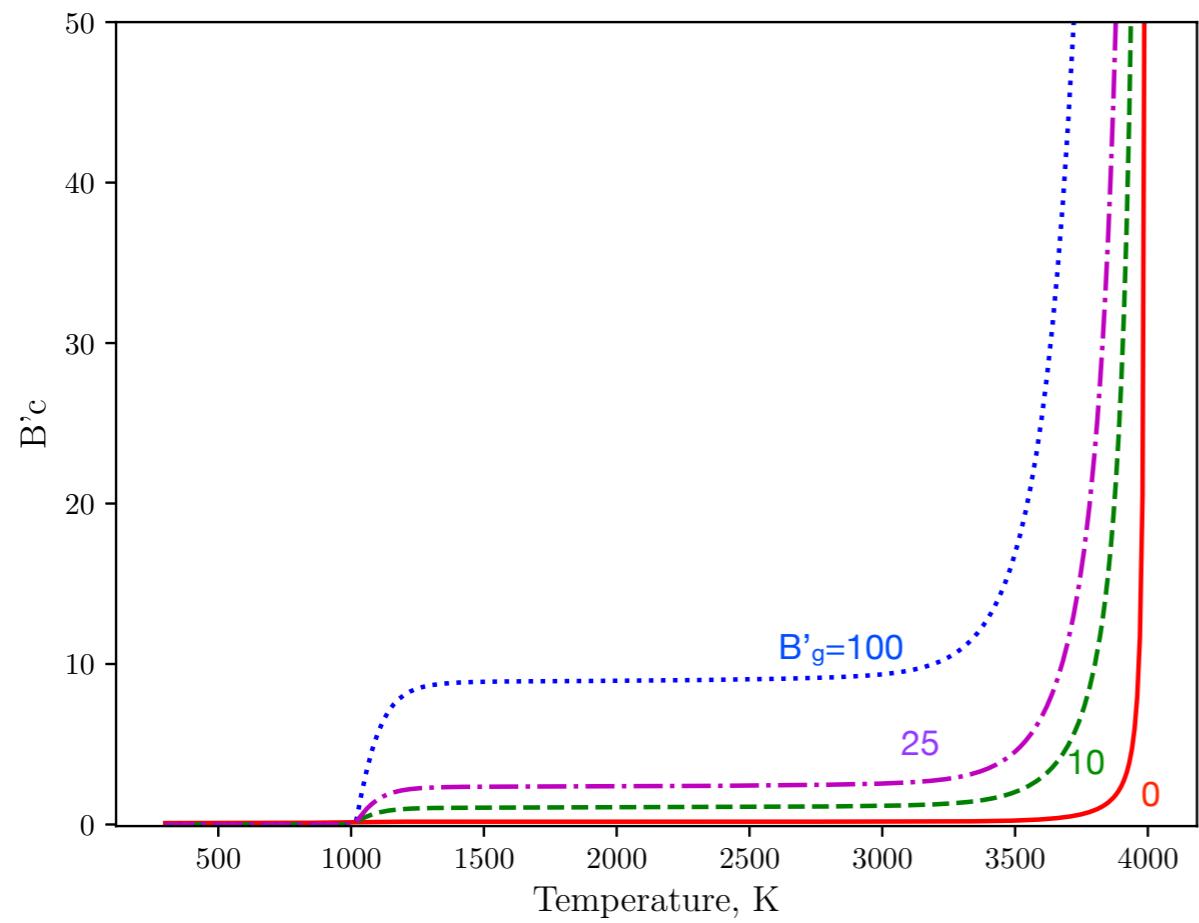
B'_g=0



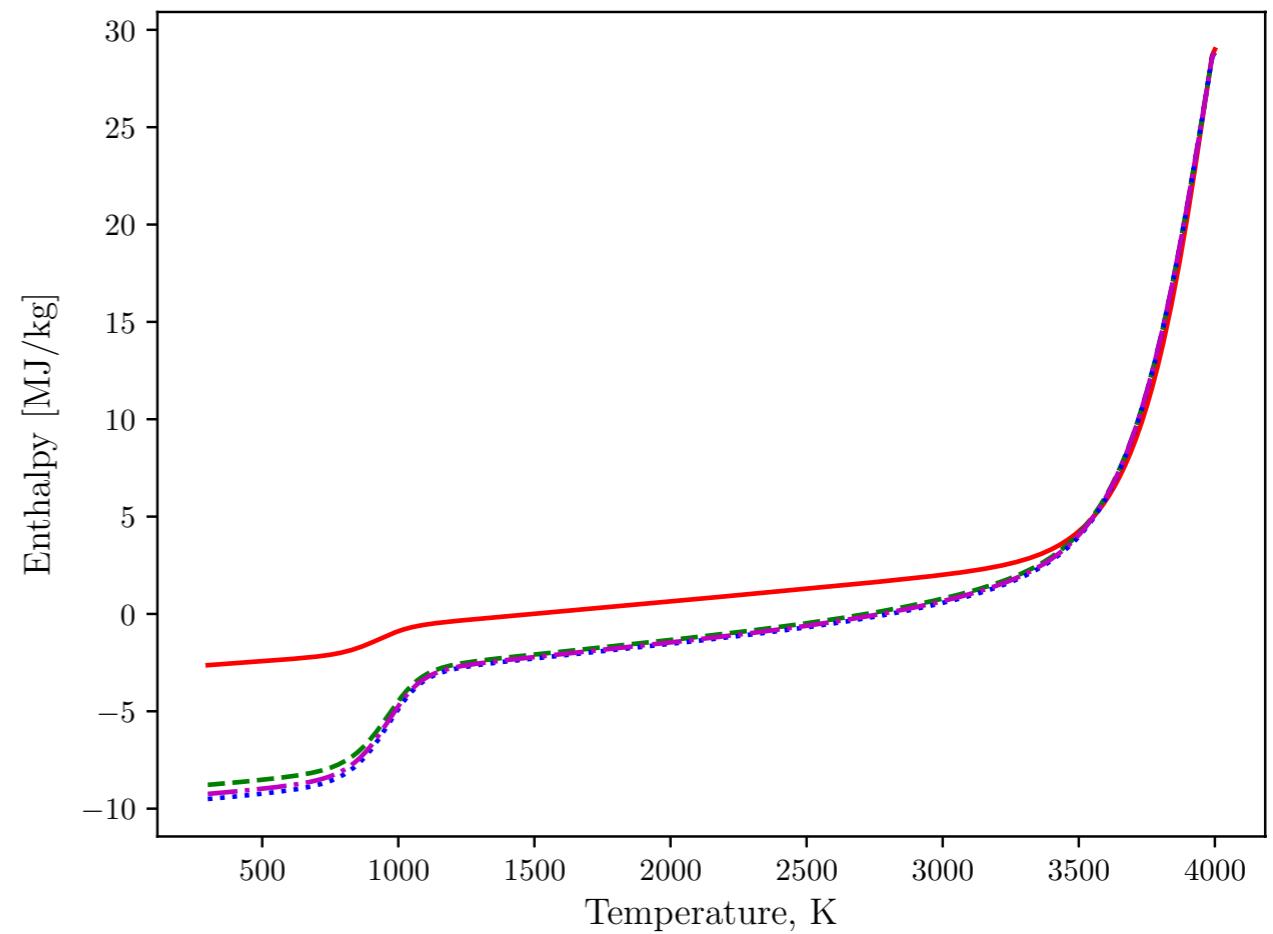
B'_g=25



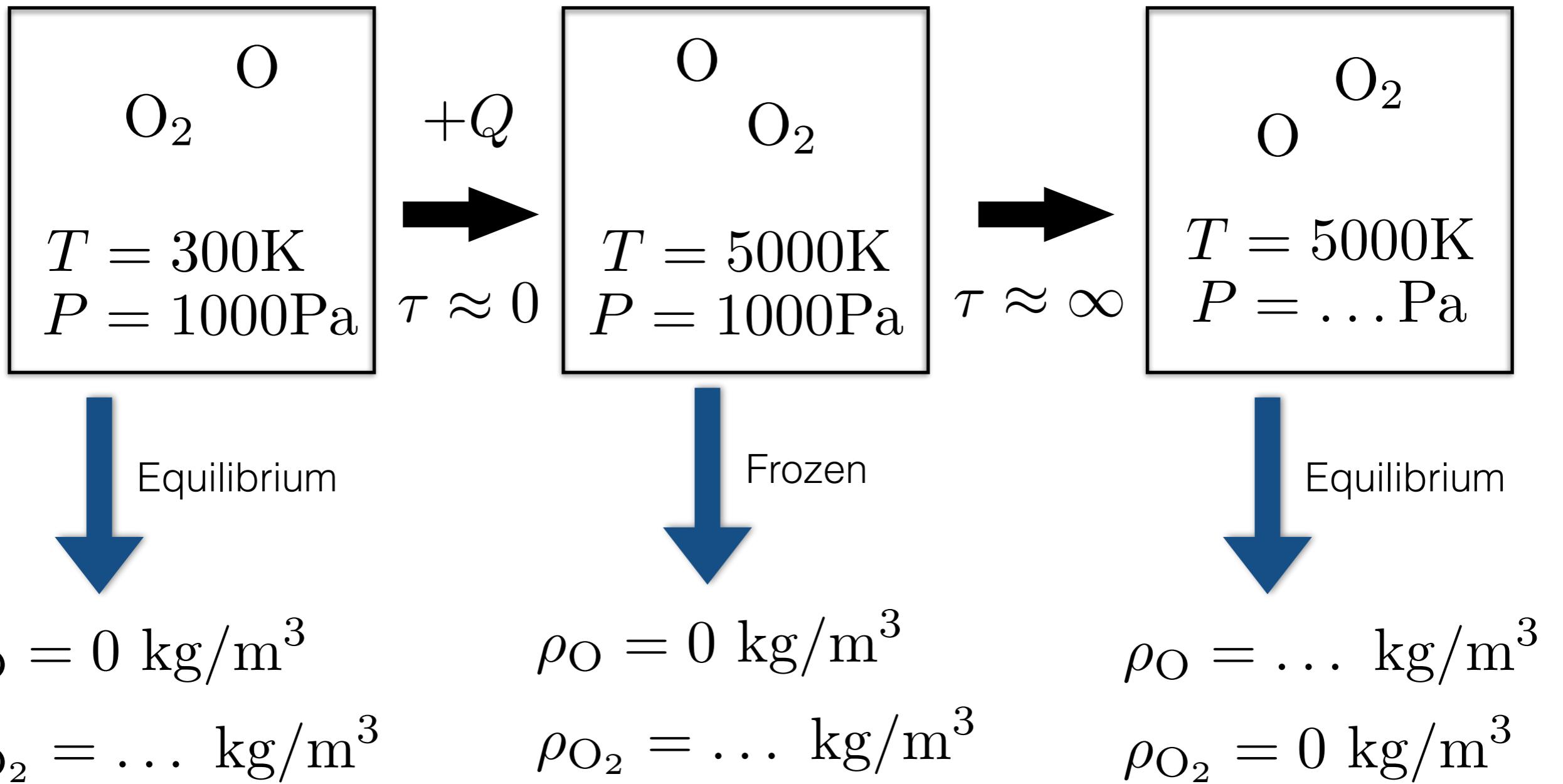
B' with temperature at 1 atm:



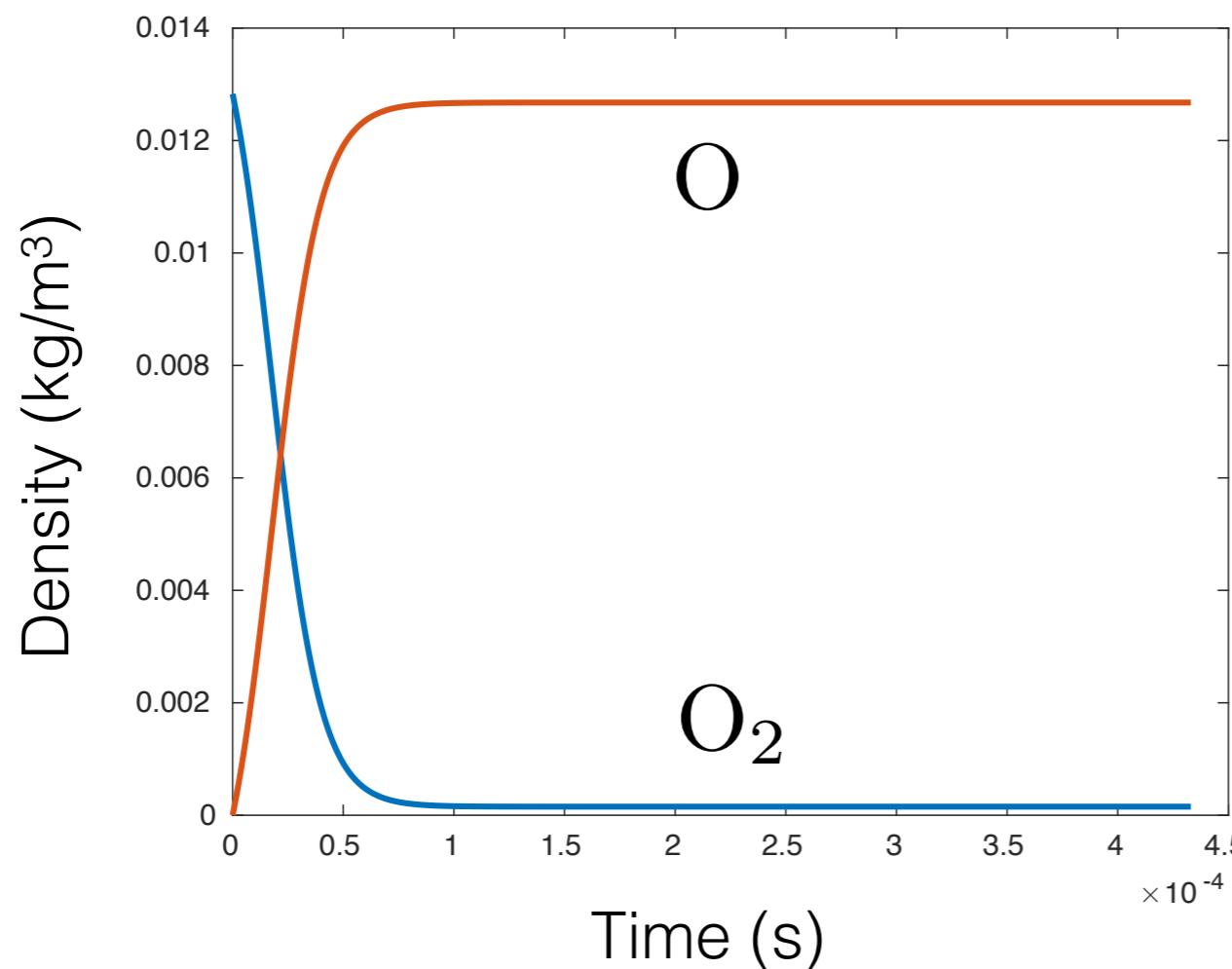
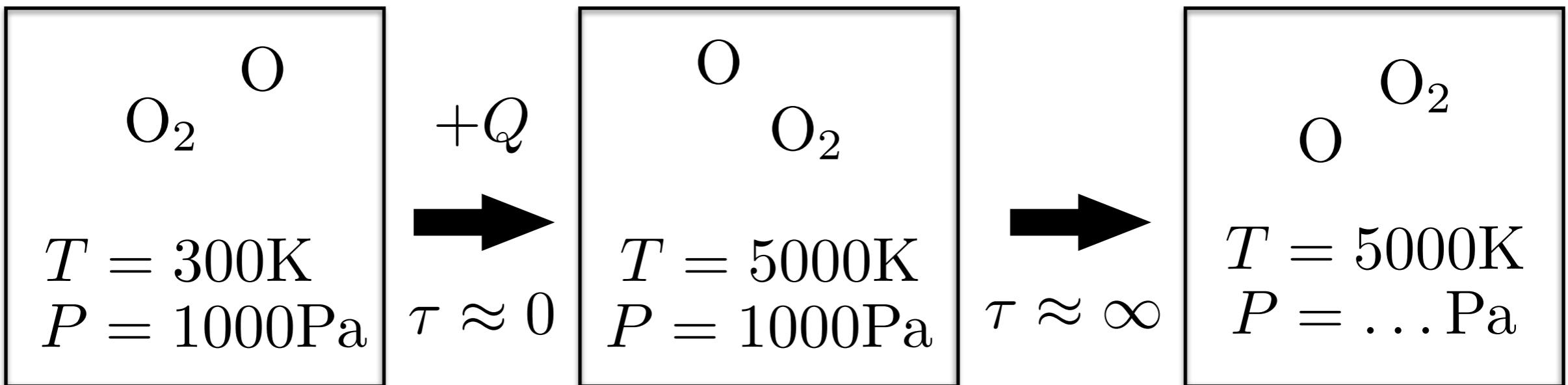
Enthalpy with temperature at 1 atm:



Reaction occur at finite rate till equilibrium

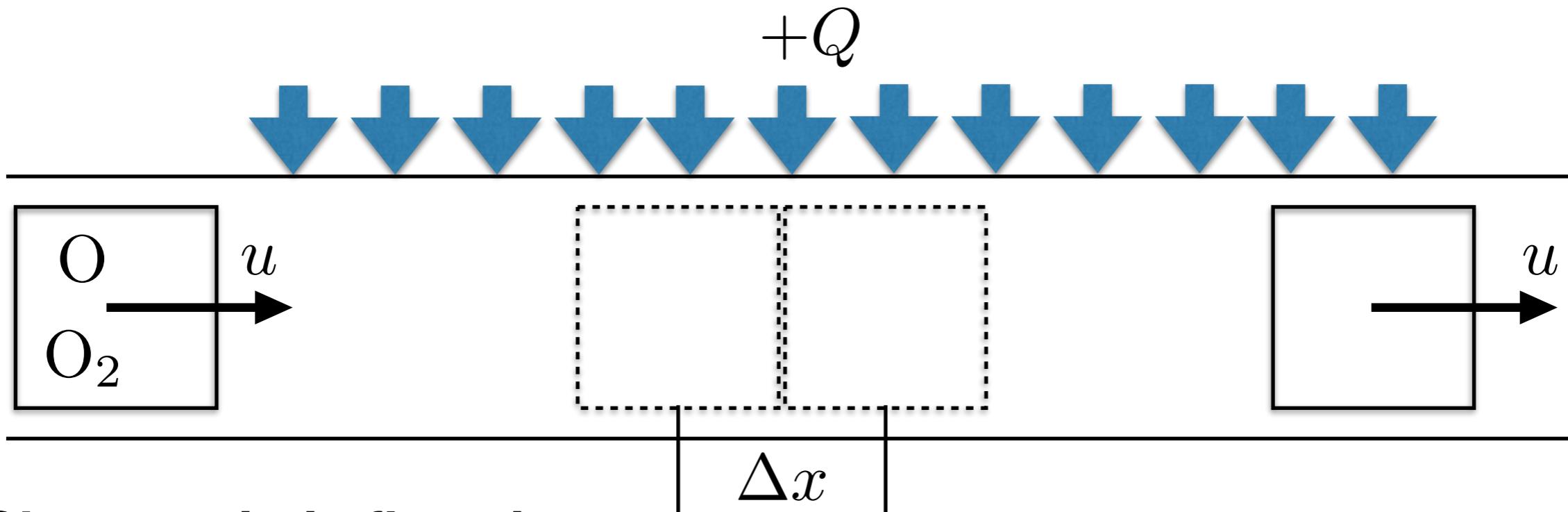


Reaction occur at finite rate till equilibrium



Finite Rate Chemistry
 $O + O \longrightarrow O_2$

When is finite-rate chemistry necessary?



Characteristic flow time

$$\tau_{\text{flow}} = \frac{\Delta x}{u}$$

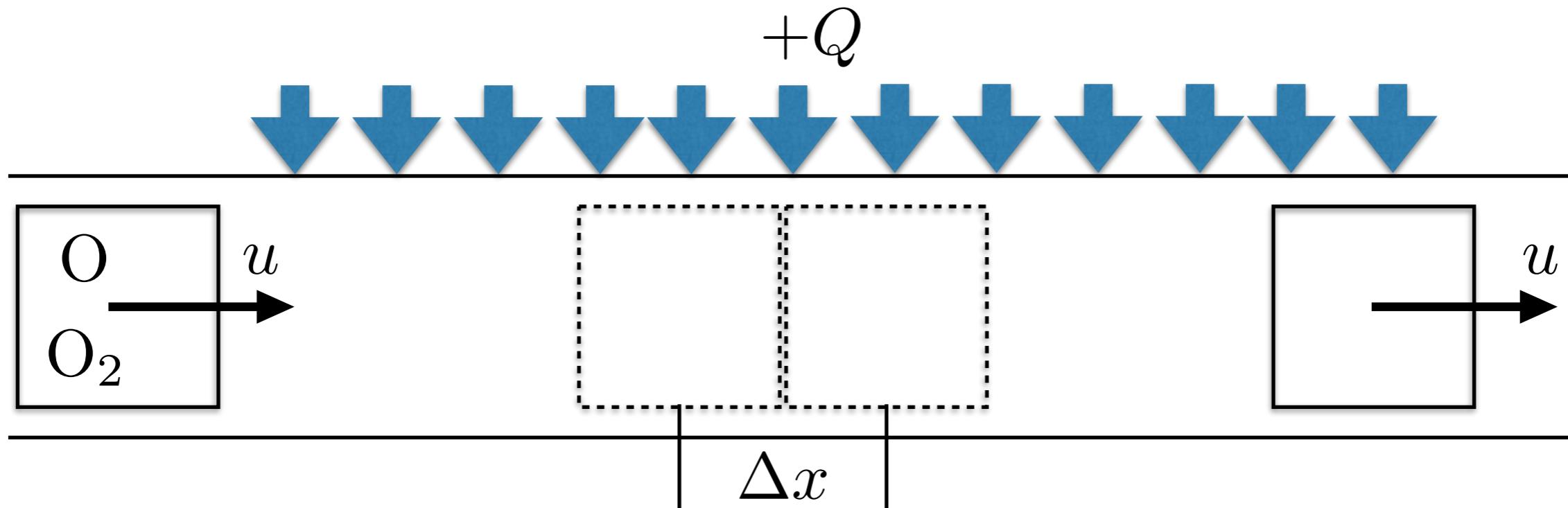
Damköhler Number:

$$Da = \frac{\tau_{\text{flow}}}{\tau_{\text{chem}}}$$

$Da \ll 1$	Frozen
$Da \gg 1$	Equilibrium

Real flows are neither frozen nor in equilibrium

When is finite-rate chemistry necessary?



Frozen/Equilibrium:

$$\frac{\partial \rho}{\partial t} + \nabla (\rho u) = 0$$

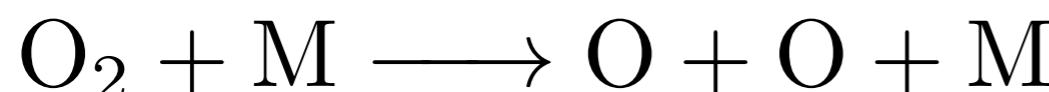
Finite Rate Chemistry

$$\frac{\partial \rho_i}{\partial t} + \nabla (\rho_i u + j_{\text{diff}}) = \dot{\omega}_i$$

Chemical Production: $\dot{\omega}_i \sim \frac{d[O]}{dt}$

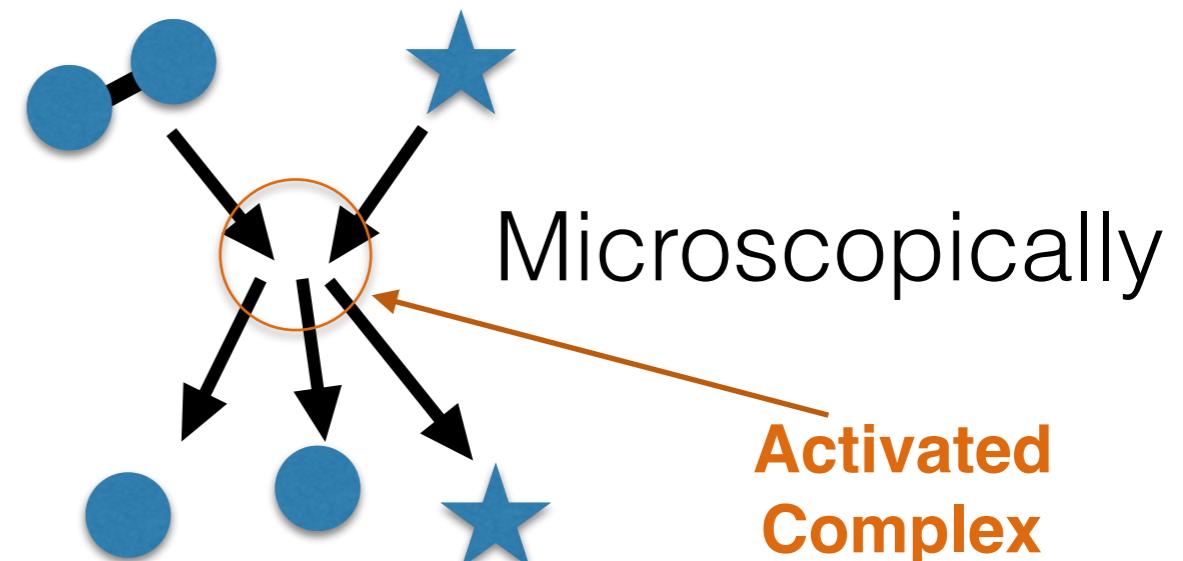
Chemical Reactions and their rate

Oxygen Dissociation



Law of Mass Action

$$r = k_f[\text{O}_2][\text{M}] \quad \text{Reaction Rate}$$



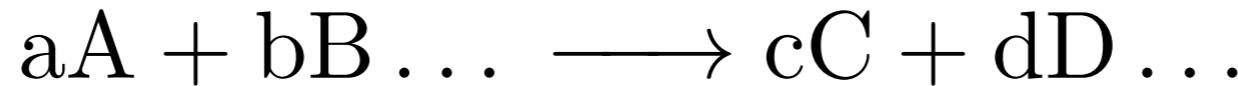
Chemical Production Rate of Species

$$\frac{d\text{O}_2}{dt} = -k_f[\text{O}_2][\text{M}]$$

Stoichiometry

$$\frac{d\text{O}}{dt} = 2k_f[\text{O}_2][\text{M}]$$

Generic Chemical Reaction



Reactants

$$\frac{d[A]}{dt} = -ak_f [A]^a[B]^b$$

$$\frac{d[B]}{dt} = -bk_f [A]^a[B]^b$$

Products

$$\frac{d[D]}{dt} = dk_f [A]^a[B]^b$$

$$\frac{d[C]}{dt} = ck_f [A]^a[B]^b$$

To go from **moles** to **kg** we multiply by the **molar mass**

Reaction Rate Coefficient

Arrhenius Law:

$$k_f = C \exp \frac{-E_a}{RT}$$

Activation Energy
Kinetic Energy of Atoms
Pre-exponential

The diagram illustrates the Arrhenius equation $k_f = C \exp \frac{-E_a}{RT}$. It highlights three key components: 'Activation Energy' at the top, 'Pre-exponential' on the left, and 'Kinetic Energy of Atoms' on the right. Orange arrows point from each label to its respective term in the equation.

Generalized Arrhenius Law: $k_f = CT^b \exp \frac{-E_a}{RT}$

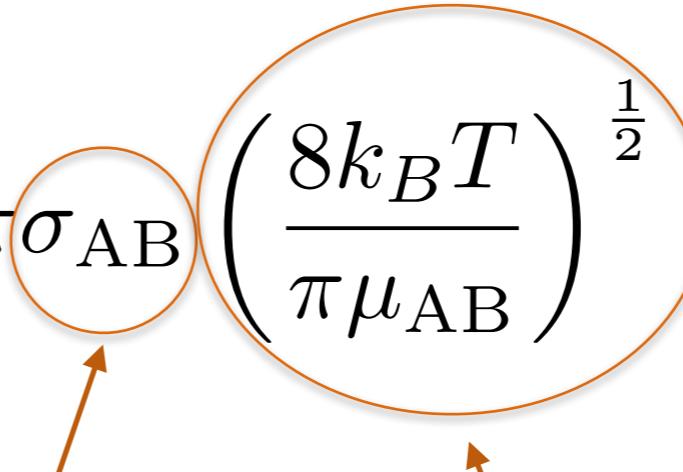
Determined Experimentally or from Quantum Chemistry

Kinetic Theory “explains” Reaction Rate Coef

Number of collisions between A and B per sec

$$Z_{AB} = [A][B]\pi\sigma_{AB}\left(\frac{8k_B T}{\pi\mu_{AB}}\right)^{\frac{1}{2}}$$

Collision Cross Section Average Relative Velocity



From these collisions only some of them are effective

$$r = \Pi Z_{AB}$$

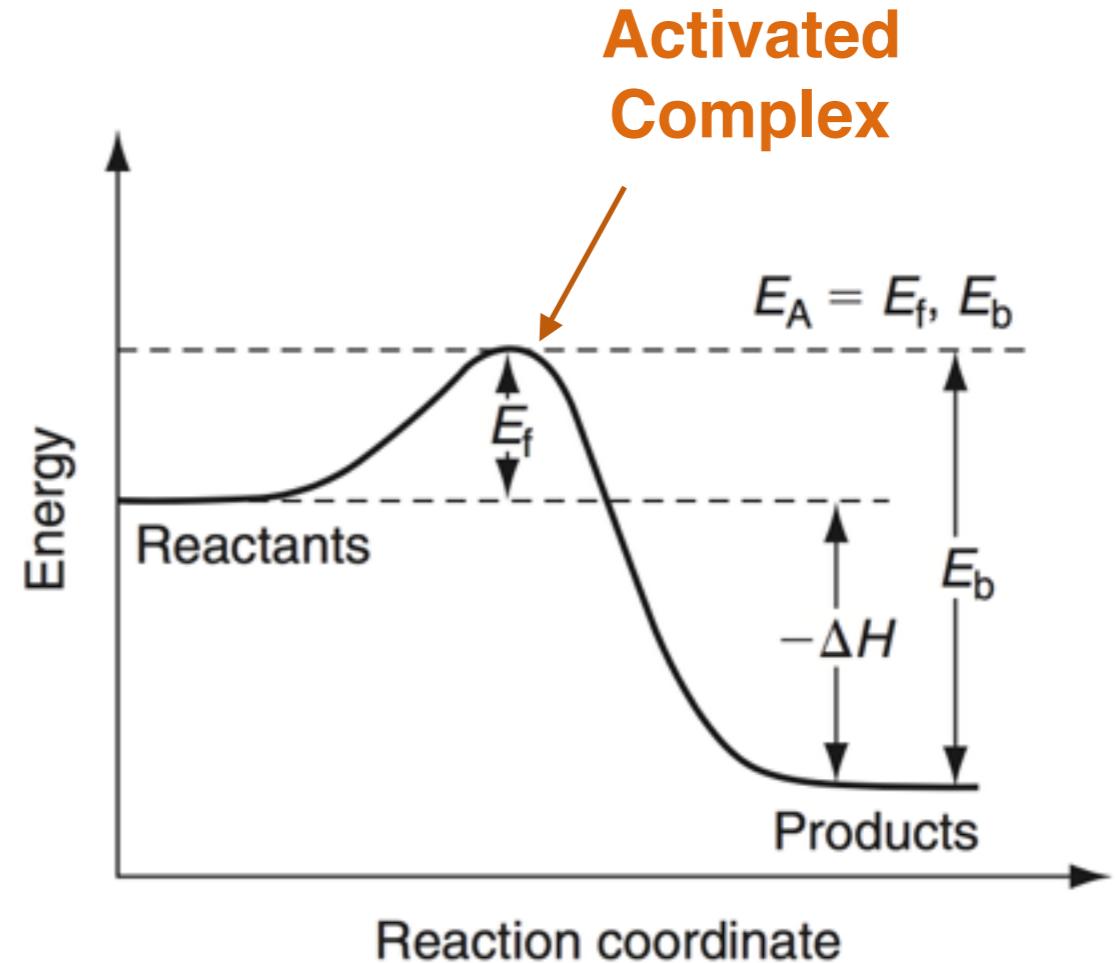
Reaction Probability Π

Activation Energy

$$\Pi_{\text{act}} = \exp \frac{-E_a}{RT}$$

Steric (Orientation) Factor

p



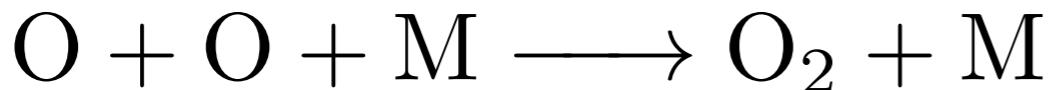
Substituting:

$$r = \sigma_{AB} p \left(\frac{8k_B T}{\pi \mu_{AB}} \right)^{\frac{1}{2}} \exp \frac{-E_a}{RT} [A][B]$$

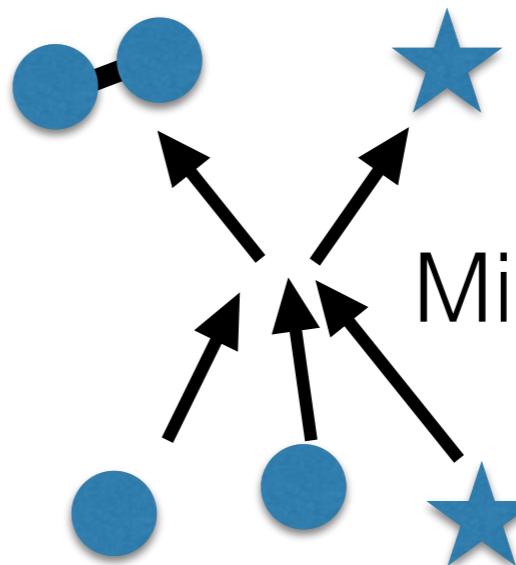
$$k_f = CT^b \exp \frac{-E_a}{RT}$$

Kinetic theory does not provide closure of the two parameters

Backward Reactions (micro-reversibility)



$$r_b = k_b[\text{O}]^2[\text{M}]$$



Forward and backward rates are interdependent!

At Equilibrium:

$$r_f = r_b$$

or:

$$\frac{k_f(T)}{k_b(T)} = \frac{[\text{O}_2]}{[\text{O}]^2}$$

Equilibrium Constant: $K_C(T)$

Provided rigorously
from thermodynamics

What happens when I have multiple reactions?

```
<!-- Park Air-5 Reaction Mechanism -->
<mechanism name="air5_park">

    <arrhenius_units A="mol,cm,s,K" E="kcal,mol,K" />

    <!-- 1 -->
    <reaction formula="N2+M=2N+M">
        <arrhenius A="3.0E+22" n="-1.6" T="113200.0" />
        <M>N2:0.2333, N0:0.2333, O2:0.2333</M>
    </reaction>

    <!-- 2 -->
    <reaction formula="O2+M=2O+M">
        <arrhenius A="1.0E+22" n="-1.5" T="59360.0" />
        <M>N2:0.2, N0:0.2, O2:0.2</M>
    </reaction>

    <!-- 3 -->
    <reaction formula="N0+M=N+O+M">
        <arrhenius A="5.0E15" n="+0.0" T="75500.0" />
        <M>NO:20.0, N:22.0, O:22.0</M>
    </reaction>

    <!-- 4 -->
    <reaction formula="N2+O=NO+N">
        <arrhenius A="5.69E+12" n="+0.42" T="42938.0" />
    </reaction>

    <!-- 5 -->
    <reaction formula="O2+N=NO+O">
        <arrhenius A="2.49E+09" n="+1.18" T="4005.5" />
    </reaction>
</mechanism>
```

What happens when I have multiple reactions?

```
<!-- Park Air-5 Reaction Mechanism -->
<mechanism name="air5_park">

  <arrhenius_units A="mol,cm,s,K" E="kcal,mol,K" />

  <!-- 1 -->
  <reaction formula="N2+M=2N+M">
    <arrhenius A="3.0E+22" n="-1.6" T="113200.0" />
    <M>N2:0.2333, NO:0.2333, O2:0.2333</M>
  </reaction>      Pre-exp      b      Activation
                                         Temperature
                                         Ea/R

  <!-- 2 -->
  <reaction formula="O2+M=2O+M">
    <arrhenius A="1.0E+22" n="-1.5" T="59360.0" />
    <M>N2:0.2, NO:0.2, O2:0.2</M>
  </reaction>

  <!-- 3 -->
  <reaction formula="NO+M=N+O+M">
    <arrhenius A="5.0E15" n="+0.0" T="75500.0" />
    <M>NO:20.0, N:22.0, O:22.0</M>
  </reaction>

  Third Body Efficiency
  <!-- 4 -->
  <reaction formula="N2+O=NO+N">
    <arrhenius A="5.69E+12" n="+0.42" T="42938.0" />
  </reaction>

  <!-- 5 -->
  <reaction formula="O2+N=NO+O">
    <arrhenius A="2.49E+09" n="+1.18" T="4005.5" />
  </reaction>
</mechanism>
```

What happens when I have multiple reactions?

```
<!-- Park Air-5 Reaction Mechanism -->
<mechanism name="air5_park">

<arrhenius_units A="mol,cm,s,K" E="kcal,mol,K" />

<!-- 1 -->
<reaction formula="N2+M=2N+M">
    <arrhenius A="3.0E+22" n="-1.6" T="113200.0" />
    <M>N2:0.2333, N0:0.2333, O2:0.2333</M>
</reaction>

<!-- 2 -->
<reaction formula="O2+M=2O+M">
    <arrhenius A="1.0E+22" n="-1.5" T="59360.0" />
    <M>N2:0.2, N0:0.2, O2:0.2</M>
</reaction>

<!-- 3 -->
<reaction formula="N0+M=N+O+M">
    <arrhenius A="5.0E15" n="+0.0" T="75500.0" />
    <M>NO:20.0, N:22.0, O:22.0</M>
</reaction>

<!-- 4 -->
<reaction formula="N2+O=NO+N">
    <arrhenius A="5.69E+12" n="+0.42" T="42938.0" />
</reaction>

<!-- 5 -->
<reaction formula="O2+N=NO+O">
    <arrhenius A="2.49E+09" n="+1.18" T="4005.5" />
</reaction>
</mechanism>
```

Example:

$$\frac{d[O]}{dt} = \begin{array}{l} +2k_{f,2}[O_2][M] \\ +k_{f,3}[NO][M] \\ -k_{f,4}[N_2][O] \\ +k_{f,5}[O_2][N] \end{array}$$

$$\begin{array}{l} -2k_{b,2}[O]^2[M] \\ -k_{b,3}[N][O][M] \\ +k_{b,4}[NO][N] \\ -k_{b,5}[NO][O] \end{array}$$

Forward

Lunch Time!



VKI Mutation++ Training Day

Friday, June 15, 2018

9:30 am	Welcome and overview of the library
9:50 am	Download and installation <ul style="list-style-type: none">• Using the Gitlab and wiki• Installing dependencies• Compiling• Running tests• Building the documentation
10:15 am	<i>Coffee break</i>
10:30 am	Discovering the built in tools of Mutation++ <ul style="list-style-type: none">• Theory: Gas mixtures, thermodynamics, equilibrium, finite-rate chemistry, and equilibrium surface chemistry (B')• Finding / modifying mixture files• <i>checkmix</i> – Check that a mixture has been formatted correctly• <i>mppequil</i> – Compute properties of a gas in equilibrium• <i>bprime</i> – Compute B' tables for a specific mixture and surface
12:15 pm	<i>Lunch</i>
1:15 pm	Coding with Mutation++* <ul style="list-style-type: none">• Setting up dependencies for an external project• Loading / creating a mixture in code• Implementing a 0D reactor with thermochemical nonequilibrium <p>* Examples will be provided in C++ and Fortran</p>
3:15 pm	<i>Coffee break</i>
3:30 pm	Coding with Mutation++ continued
4:00 pm	Discussion
4:30 pm	Closure

Before we begin...

Please download the training materials, including the example programs and this presentation:

```
git clone https://sync.vki.ac.be/mpp/training_materials.git
```

Hello World!

Hello ~~World!~~ Mixture

Printing species in a mixture

Hello World! Mixture

Printing species in a mixture

hello.cpp

```
#include "mutation++.h"
#include <iostream>

int main(int argc, char* argv[])
{
    // Load air5 mixture
    Mutation::Mixture mix("air5");

    // Print the species
    std::cout << "Species: ";
    for (auto& s : mix.species())
        std::cout << s.name() << ' ';
    std::cout << '\n';

    return 0;
}
```

hello.f90

```
program main
    use mutationpp
    implicit none
    character(len=10) :: mixture
    character(len=12) :: species, state_model
    integer :: j, ns

    mixture      = "air5"
    state_model = "ChemNonEq1T"

    ! Initialize the mutation++ library
    call mpp_initialize(mixture, state_model)
    ns = mpp_nspecies()

    write(*,'(A12)',advance='no') "Species: "
    do j = 1,ns
        call mpp_species_name(j, species)
        write(*,'(A12)',advance='no') species
    end do
    write(*,*)

    ! Clean up the memory in the mutation++ library
    call mpp_destroy()

end program main
```

Using M++ in your own projects

Setting up dependencies

Makefile for C++ code.

```
# Change this path to your Mutation++ install path
MPP_INSTALL_DIR = $(MPP_DIRECTORY)/install

CXX_FLAGS = -I $(MPP_INSTALL_DIR)/include \
            -I $(MPP_INSTALL_DIR)/include/mutation++ \
            -O3 -std=c++11

# use .dylib for Mac, .so for Unix
UNAME_S := $(shell uname -s)
ifeq ($(UNAME_S),Linux)
    CXX      = g++
    CXX_LIBS = $(MPP_INSTALL_DIR)/lib/libmutation++.so
endif
ifeq ($(UNAME_S),Darwin)
    CXX      = c++
    CXX_LIBS = $(MPP_INSTALL_DIR)/lib/libmutation++.dylib
endif

hello : hello.o
    $(CXX) -o $@ $(CXX_FLAGS) $^ $(CXX_LIBS)

%.o : %.cpp
    $(CXX) -c $(CXX_FLAGS) $<
```

Makefile for Fortran code.

```
# Change this path to your Mutation++ install path
MPP_INSTALL = $(MPP_DIRECTORY)/install

FC_FLAGS = -J$(MPP_INSTALL)/include/mutation++ \
           -O3 -g -fdefault-real-8

# use .dylib for Mac, .so for Unix
UNAME_S := $(shell uname -s)
ifeq ($(UNAME_S),Linux)
    FC      = gfortran
    FC_LIBS = $(MPP_INSTALL)/lib/libmutation++_fortran.so
endif
ifeq ($(UNAME_S),Darwin)
    FC      = gfortran
    FC_LIBS = $(MPP_INSTALL)/lib/libmutation++_fortran.dylib
endif

hello : hello.o
    $(FC) -o $@ $(FC_FLAGS) $^ $(FC_LIBS)

%.o : %.f90
    $(FC) -c $(FC_FLAGS) $<
```

Using M++ in your own projects

Setting up dependencies

Makefile for C++ code.

```
# Change this path to your Mutation++ install path
MPP_INSTALL_DIR = $(MPP_DIRECTORY)/install

CXX_FLAGS = -I $(MPP_INSTALL_DIR)/include \
            -I $(MPP_INSTALL_DIR)/include/mutation++ \
            -O3 -std=c++11

# use .dylib for Mac, .so for Unix
UNAME_S := $(shell uname -s)
ifeq ($(UNAME_S),Linux)
    CXX      = g++
    CXX_LIBS = $(MPP_INSTALL_DIR)/lib/libmutation++.so
endif
ifeq ($(UNAME_S),Darwin)
    CXX      = c++
    CXX_LIBS = $(MPP_INSTALL_DIR)/lib/libmutation++.dylib
endif

hello : hello.o
    $(CXX) -o $@ $(CXX_FLAGS) $^ $(CXX_LIBS)

%.o : %.cpp
    $(CXX) -c $(CXX_FLAGS) $<
```

Makefile for Fortran code.

```
# Change this path to your Mutation++ install path
MPP_INSTALL = $(MPP_DIRECTORY)/install

FC_FLAGS = -J$(MPP_INSTALL)/include/mutation++ \
           -O3 -g -fdefault-real-8

# use .dylib for Mac, .so for Unix
UNAME_S := $(shell uname -s)
ifeq ($(UNAME_S),Linux)
    FC      = gfortran
    FC_LIBS = $(MPP_INSTALL)/lib/libmutation++_fortran.so
endif
ifeq ($(UNAME_S),Darwin)
    FC      = gfortran
    FC_LIBS = $(MPP_INSTALL)/lib/libmutation++_fortran.dylib
endif

hello : hello.o
    $(FC) -o $@ $(FC_FLAGS) $^ $(FC_LIBS)

%.o : %.f90
    $(FC) -c $(FC_FLAGS) $<
```

Using M++ in your own projects

Setting up dependencies

Makefile for C++ code.

```
# Change this path to your Mutation++ install path
MPP_INSTALL_DIR = $(MPP_DIRECTORY)/install

CXX_FLAGS = -I $(MPP_INSTALL_DIR)/include \
            -I $(MPP_INSTALL_DIR)/include/mutation++ \
            -O3 -std=c++11

# use .dylib for Mac, .so for Unix
UNAME_S := $(shell uname -s)
ifeq ($(UNAME_S),Linux)
    CXX      = g++
    CXX_LIBS = $(MPP_INSTALL_DIR)/lib/libmutation++.so
endif
ifeq ($(UNAME_S),Darwin)
    CXX      = c++
    CXX_LIBS = $(MPP_INSTALL_DIR)/lib/libmutation++.dylib
endif

hello : hello.o
    $(CXX) -o $@ $(CXX_FLAGS) $^ $(CXX_LIBS)

%.o : %.cpp
    $(CXX) -c $(CXX_FLAGS) $<
```

Makefile for Fortran code.

```
# Change this path to your Mutation++ install path
MPP_INSTALL = $(MPP_DIRECTORY)/install

FC_FLAGS = -J$(MPP_INSTALL)/include/mutation++ \
           -O3 -g -fdefault-real-8

# use .dylib for Mac, .so for Unix
UNAME_S := $(shell uname -s)
ifeq ($(UNAME_S),Linux)
    FC      = gfortran
    FC_LIBS = $(MPP_INSTALL)/lib/libmutation++_fortran.so
endif
ifeq ($(UNAME_S),Darwin)
    FC      = gfortran
    FC_LIBS = $(MPP_INSTALL)/lib/libmutation++_fortran.dylib
endif

hello : hello.o
    $(FC) -o $@ $(FC_FLAGS) $^ $(FC_LIBS)

%.o : %.f90
    $(FC) -c $(FC_FLAGS) $<
```

Exercise

Compile and run the `hello.cpp` or `hello.f90` programs.

Creating a Mixture from scratch

Understanding the MixtureOptions class

hello.cpp

```
#include "mutation++.h"
#include <iostream>

int main(int argc, char* argv[ ])
{
    // Load air5 mixture
    Mutation::Mixture mix("air5");

    // Print the species
    std::cout << "Species: ";
    for (auto& s : mix.species())
        std::cout << s.name() << ' ';
    std::cout << '\n';

    return 0;
}
```

Creating a Mixture from scratch

Understanding the MixtureOptions class

hello.cpp

```
#include "mutation++.h"
#include <iostream>

int main(int argc, char* argv[])
{
    // Load air5 mixture
    Mutation::Mixture mix("air5");

    // Print the species
    std::cout << "Species: ";
    for (auto& s : mix.species())
        std::cout << s.name() << ' ';
    std::cout << '\n';

    return 0;
}
```

mixture_options.cpp

```
#include "mutation++.h"
#include <iostream>

int main(int argc, char* argv[])
{
    // Setup mixture options
    Mutation::MixtureOptions options;
    options.setSpeciesDescriptor("N O NO N2 O2");
    options.setThermodynamicDatabase("RRHO");

    // Create the mixture
    Mutation::Mixture mix(options);

    // Print the species
    std::cout << "Species: ";
    for (auto& s : mix.species())
        std::cout << s.name() << ' ';
    std::cout << '\n';

    return 0;
}
```

Creating a Mixture from scratch

Understanding the MixtureOptions class

hello.cpp

```
#include "mutation++.h"
#include <iostream>

int main(int argc, char* argv[])
{
    // Load air5 mixture
    Mutation::Mixture mix("air5");

    // Print the species
    std::cout << "Species: ";
    for (auto& s : mix.species())
        std::cout << s.name() << ' ';
    std::cout << '\n';

    return 0;
}
```

mixture_options.cpp

```
#include "mutation++.h"
#include <iostream>

int main(int argc, char* argv[])
{
    // Setup mixture options
    Mutation::MixtureOptions options;
    options.setSpeciesDescriptor("N O NO N2 O2");
    options.setThermodynamicDatabase("RRHO");

    // Create the mixture
    Mutation::Mixture mix(options);

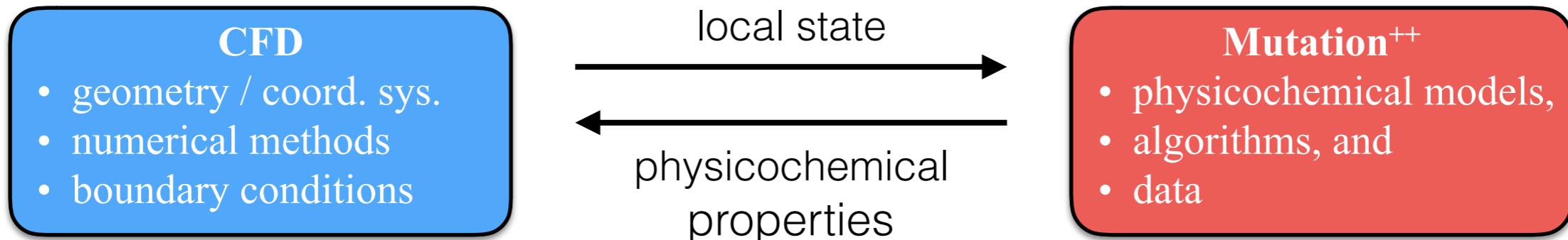
    // Print the species
    std::cout << "Species: ";
    for (auto& s : mix.species())
        std::cout << s.name() << ' ';
    std::cout << '\n';

    return 0;
}
```

Exercise

- 1) Modify hello.cpp to use MixtureOptions as above.
- 2) Try using more generic species descriptor to load all species with N and O (including ions).

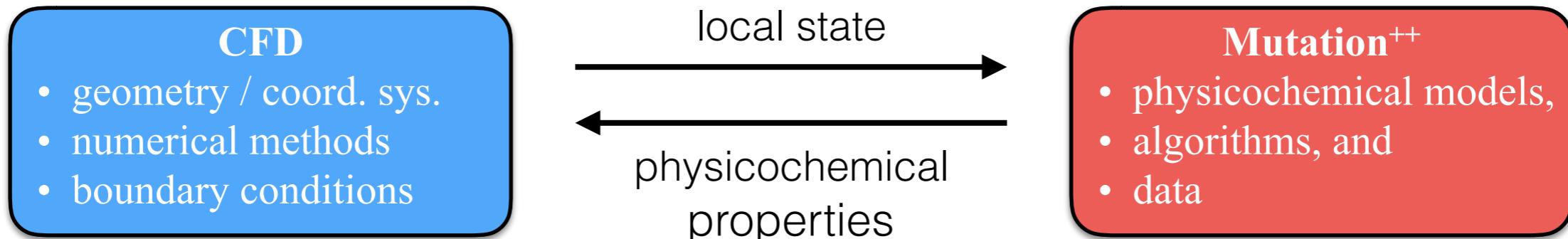
Setting the state of the mixture



```
mix.setState(const double* const vec1, const double* const vec2, int vars = 0);
```

State Model	vars	vec1	vec2
Equil	0	mixture density	mixture energy density
	1	static pressure	temperature
	2	element mole fractions	pressure, temperature
ChemNonEq1T	0	species densities	mixture energy density
	1	species densities	temperature
	2	species mass fractions	pressure, temperature
ChemNonEqTTv	0	species densities	total and vibronic energy densities
	1	species densities	trans. and vibronic temperatures

Setting the state of the mixture



```
mix.setState(const double* const vec1, const double* const vec2, int vars = 0);
```

State Model	vars	vec1	vec2
Equil	0	mixture density	mixture energy density
	1	static pressure	temperature
	2	element mole fractions	pressure, temperature
ChemNonEq1T	0	species densities	mixture energy density
	1	species densities	temperature
	2	species mass fractions	pressure, temperature
ChemNonEqTTv	0	species densities	total and vibronic energy densities
	1	species densities	trans. and vibronic temperatures

*Natural variables for CFD are always the default (0) variable set.

Setting the state of the mixture

Example equilibrium code

set_state_equil.cpp

```
#include "mutation++.h"
#include <iostream>

int main(int argc, char* argv[])
{
    // Create the mixture
    Mutation::MixtureOptions options("air5");
    options.setStateModel("Equil");

    Mutation::Mixture mix(options);
    mix.addComposition("N2:0.79, O2:0.21", true);

    double P = Mutation::ONEATM;
    double T = 7000.0;
    mix.setState(&P, &T, 1);

    for (int i = 0; i < mix.nSpecies(); ++i) {
        std::cout << std::setw(10) << mix.speciesName(i);
        std::cout << std::setw(15) << mix.X()[i] << '\n';
    }

    return 0;
}
```

Setting the state of the mixture

Example equilibrium code

set_state_equil.cpp

```
#include "mutation++.h"
#include <iostream>

int main(int argc, char* argv[])
{
    // Create the mixture
    Mutation::MixtureOptions options("air5");
    options.setStateModel("Equil");

    Mutation::Mixture mix(options);
    mix.addComposition("N2:0.79, O2:0.21", true);

    double P = Mutation::ONEATM;
    double T = 7000.0;
    mix.setState(&P, &T, 1);

    for (int i = 0; i < mix.nSpecies(); ++i) {
        std::cout << std::setw(10) << mix.speciesName(i);
        std::cout << std::setw(15) << mix.X()[i] << '\n';
    }

    return 0;
}
```

Exercise

Try above code with a pure nitrogen mixture.

Setting the state of the mixture

Example non-equilibrium code

set_state_noneq.cpp

```
int main(int argc, char* argv[])
{
    // Same setup as before...

    // Equilibrate the solution to T and P
    double P = Mutation::ONEATM; double T = 7000.0;
    mix.equilibrate(T, P);

    // Print mole fractions
    std::cout << "-- Before changing temperature --\n";
    std::cout << "Mole fractions:\n";
    printMoleFractions(mix);

    // Print the species production rates
    std::cout << "Production rates:\n";
    printProductionRates(mix);

    // Change the temperature but leave the densities the same
    std::vector<double> rho(mix.nSpecies());
    mix.densities(rho.data());
    T = 1000.0;
    mix.setState(rho.data(), &T, 1);

    // Print mole fractions
    std::cout << "-- After changing temperature --\n";
    std::cout << "Mole fractions:\n";
    printMoleFractions(mix);

    // Print the species production rates
    std::cout << "Production rates:\n";
    printProductionRates(mix);

    return 0;
}
```

Setting the state of the mixture

Example non-equilibrium code

set_state_noneq.cpp

```
int main(int argc, char* argv[])
{
    // Same setup as before...

    // Equilibrate the solution to T and P
    double P = Mutation::ONEATM; double T = 7000.0;
    mix.equilibrate(T, P);

    // Print mole fractions
    std::cout << "-- Before changing temperature --\n";
    std::cout << "Mole fractions:\n";
    printMoleFractions(mix);

    // Print the species production rates
    std::cout << "Production rates:\n";
    printProductionRates(mix);

    // Change the temperature but leave the densities the same
    std::vector<double> rho(mix.nSpecies());
    mix.densities(rho.data());
    T = 1000.0;
    mix.setState(rho.data(), &T, 1);

    // Print mole fractions
    std::cout << "-- After changing temperature --\n";
    std::cout << "Mole fractions:\n";
    printMoleFractions(mix);

    // Print the species production rates
    std::cout << "Production rates:\n";
    printProductionRates(mix);

    return 0;
}
```

Output

```
-- Before changing temperature --
Mole fractions:
    N      0.499208
    O      0.258275
    NO     0.00258515
    N2     0.239898
    O2     3.42487e-05
Production rates:
    N      4.125e-13
    O      -1.66924e-13
    NO     -2.81131e-13
    N2     -2.81269e-13
    O2     3.16824e-13
-- After changing temperature --
Mole fractions:
    N      0.499208
    O      0.258275
    NO     0.00258515
    N2     0.239898
    O2     3.42487e-05
Production rates:
    N      -1036.07
    O      -331.057
    NO     320.88
    N2     886.286
    O2     159.962
```

Setting the state of the mixture

Example non-equilibrium code

set_state_noneq.cpp

```
int main(int argc, char* argv[])
{
    // Same setup as before...

    // Equilibrate the solution to T and P
    double P = Mutation::ONEATM; double T = 7000.0;
    mix.equilibrate(T, P);

    // Print mole fractions
    std::cout << "-- Before changing temperature --\n";
    std::cout << "Mole fractions:\n";
    printMoleFractions(mix);

    // Print the species production rates
    std::cout << "Production rates:\n";
    printProductionRates(mix);

    // Change the temperature but leave the densities the same
    std::vector<double> rho(mix.nSpecies());
    mix.densities(rho.data());
    T = 1000.0;
    mix.setState(rho.data(), &T, 1);

    // Print mole fractions
    std::cout << "-- After changing temperature --\n";
    std::cout << "Mole fractions:\n";
    printMoleFractions(mix);

    // Print the species production rates
    std::cout << "Production rates:\n";
    printProductionRates(mix);

    return 0;
}
```

Output

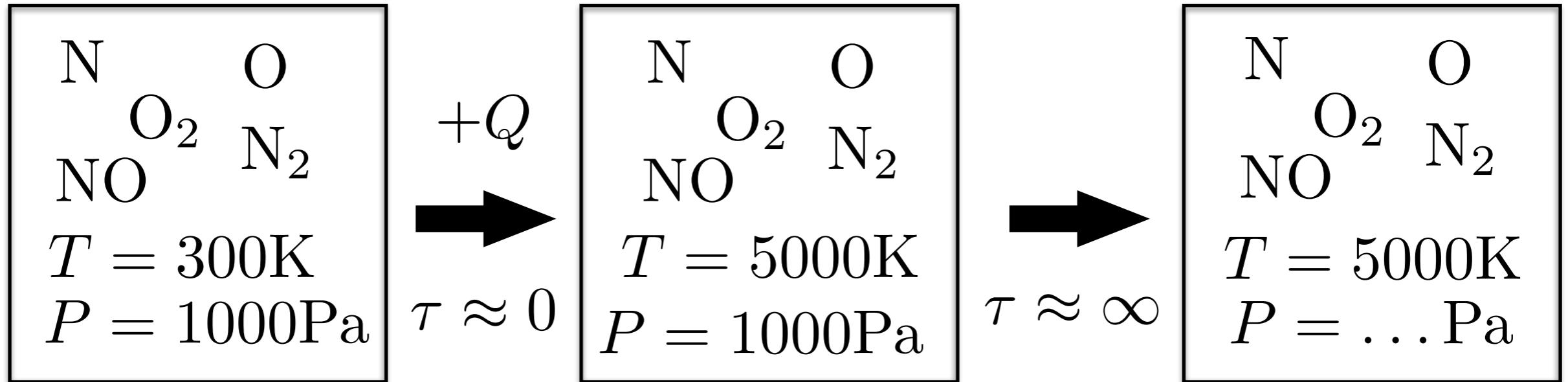
```
-- Before changing temperature --
Mole fractions:
    N      0.499208
    O      0.258275
    NO     0.00258515
    N2     0.239898
    O2     3.42487e-05
Production rates:
    N      4.125e-13
    O      -1.66924e-13
    NO     -2.81131e-13
    N2     -2.81269e-13
    O2     3.16824e-13
-- After changing temperature --
Mole fractions:
    N      0.499208
    O      0.258275
    NO     0.00258515
    N2     0.239898
    O2     3.42487e-05
Production rates:
    N      -1036.07
    O      -331.057
    NO     320.88
    N2     886.286
    O2     159.962
```

Atoms are recombining at the lower temperature.



Now for a more interesting example

Implementing a 0D reactor at constant volume and temperature



$$\frac{d\rho_i}{dt} = \dot{\omega}_i \quad \xrightarrow{\text{Explicit}} \quad \rho_i^{n+1} = \rho_i^n + \Delta t \cdot \dot{\omega}_i^n$$

Instead: Runge-Kutta 4

$$\rho_i^{n+1} = \rho_i^n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$t^{n+1} = t^n + \Delta t$$

$$k_1 = \Delta t \cdot \dot{\omega}_i(\rho^n)$$

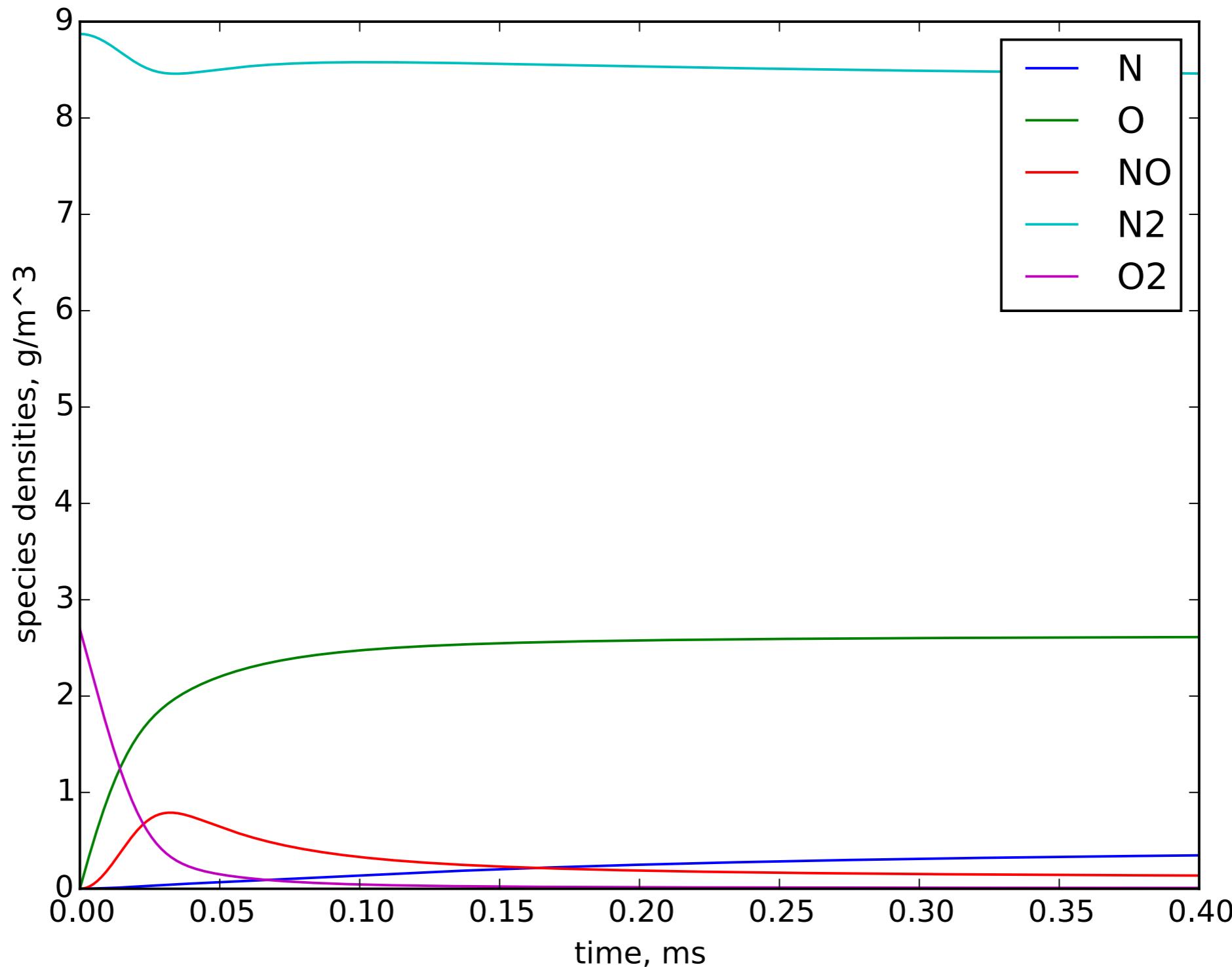
$$k_2 = \Delta t \cdot \dot{\omega}_i(\rho^n + \frac{k_1}{2})$$

$$k_3 = \Delta t \cdot \dot{\omega}_i(\rho^n + \frac{k_2}{2})$$

$$k_4 = \Delta t \cdot \dot{\omega}_i(\rho^n + k_3)$$

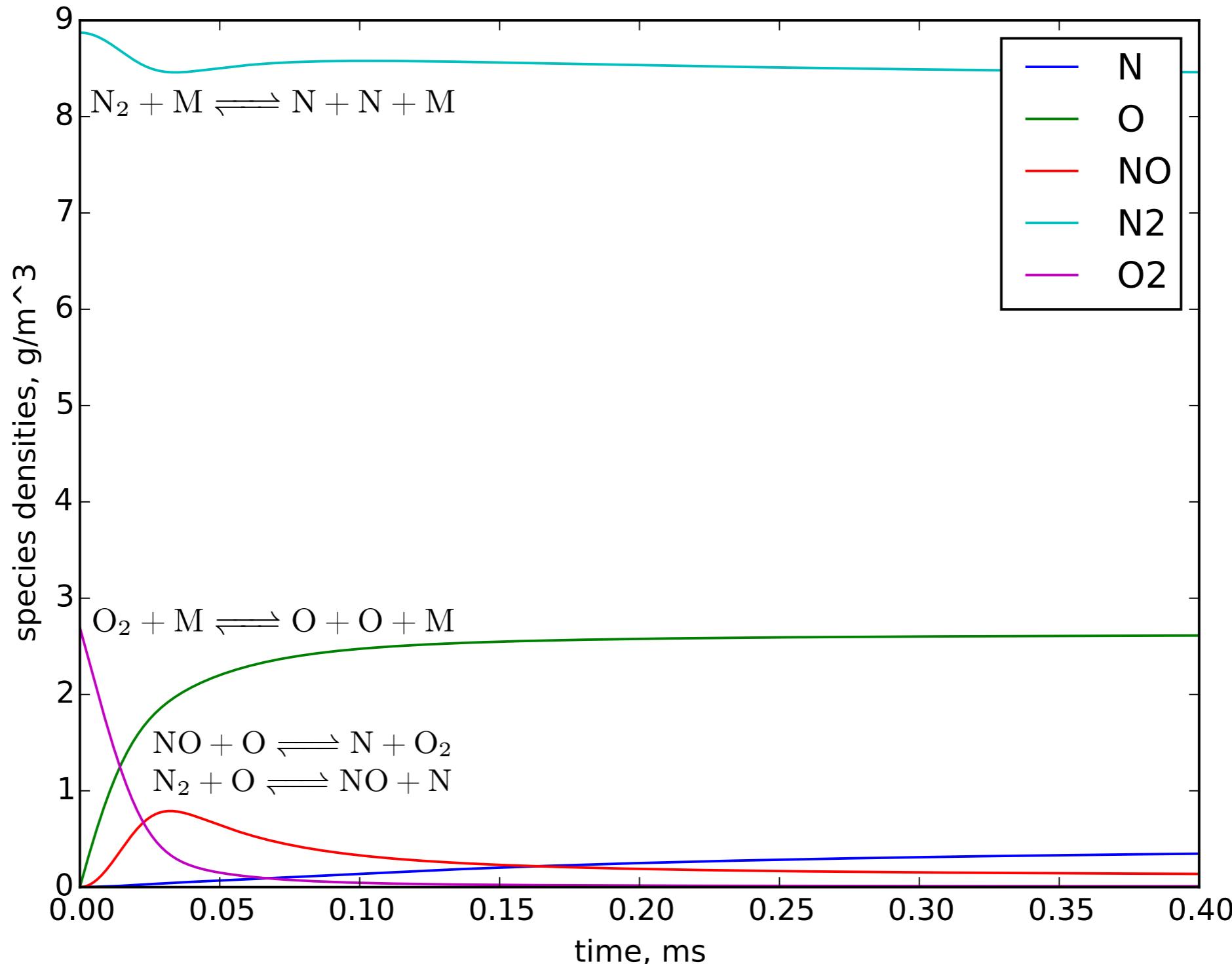
Results (cvisoRK4)

Implementing a 0D reactor at constant volume and temperature



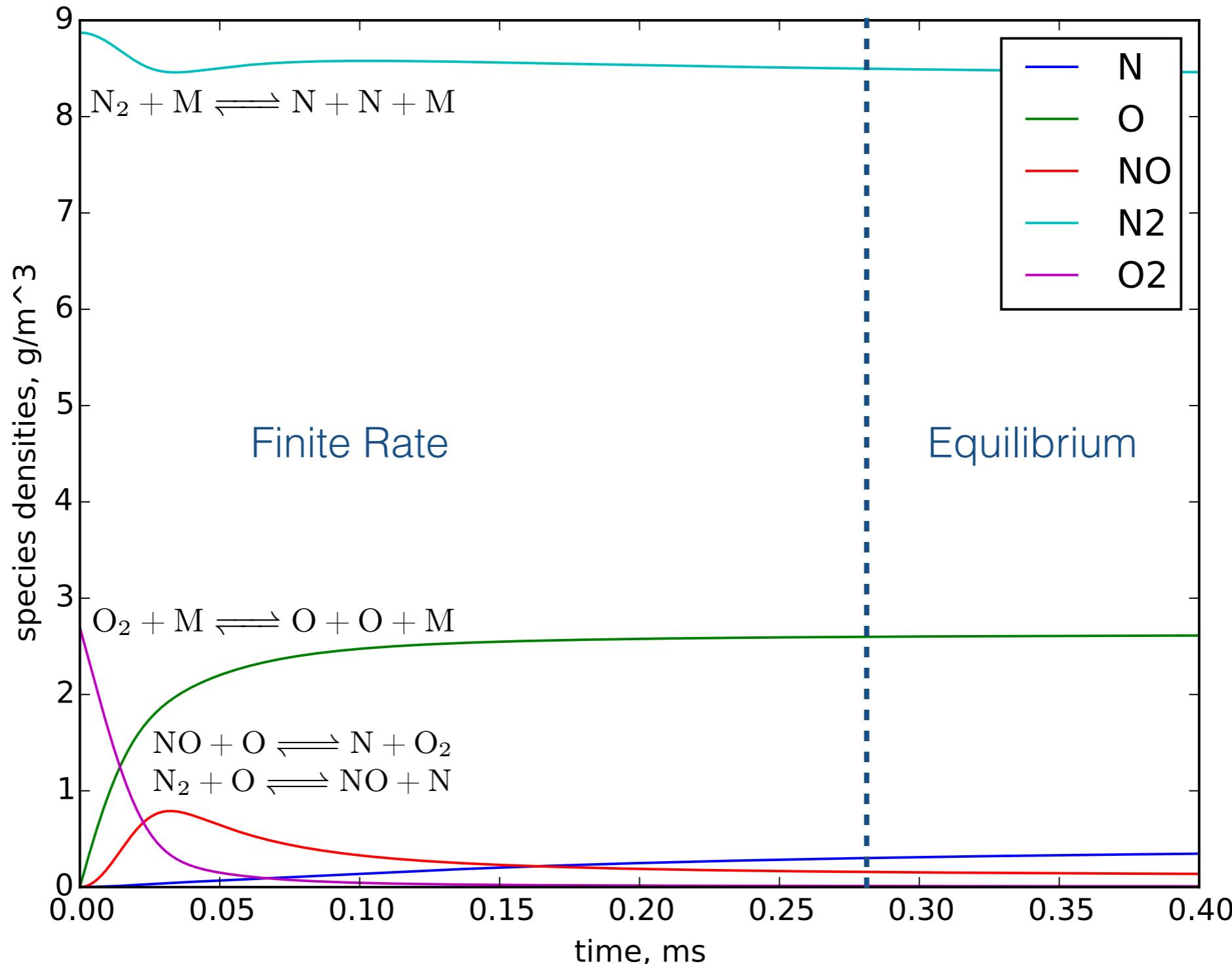
Results (cvisoRK4)

Implementing a 0D reactor at constant volume and temperature



Results (cvisoRK4)

Implementing a 0D reactor at constant volume and temperature



IT'S TIME FOR



BREAK

What else is available in the library?

Building the API documentation

What you will need: Doxygen (<http://www.stack.nl/~dimitri/doxygen/index.html>)

1. Change to `mutation++/build` directory
2. Run `ccmake ..`
3. Turn `BUILD_DOCUMENTATION` to ON
4. Type '`c`' '`c`' '`g`'
5. Run `make docs`
6. Open `mutation++/docs/html/index.html` in your favorite browser

What else is available in the library?

Building the API documentation

What you will need: Doxygen (<http://www.stack.nl/~dimitri/doxygen/index.html>)

1. Change to `mutation++/build` directory
2. Run `ccmake ..`
3. Turn `BUILD_DOCUMENTATION` to ON
4. Type '`c`' '`c`' '`g`'
5. Run `make docs`
6. Open `mutation++/docs/html/index.html` in your favorite browser

Happy Coding!

VKI Mutation++ Training Day

Friday, June 15, 2018

9:30 am	Welcome and overview of the library
9:50 am	Download and installation <ul style="list-style-type: none">• Using the Gitlab and wiki• Installing dependencies• Compiling• Running tests• Building the documentation
10:15 am	<i>Coffee break</i>
10:30 am	Discovering the built in tools of Mutation++ <ul style="list-style-type: none">• Theory: Gas mixtures, thermodynamics, equilibrium, finite-rate chemistry, and equilibrium surface chemistry (B')• Finding / modifying mixture files• <i>checkmix</i> – Check that a mixture has been formatted correctly• <i>mppequil</i> – Compute properties of a gas in equilibrium• <i>bprime</i> – Compute B' tables for a specific mixture and surface
12:15 pm	<i>Lunch</i>
1:15 pm	Coding with Mutation++* <ul style="list-style-type: none">• Setting up dependencies for an external project• Loading / creating a mixture in code• Implementing a 0D reactor with thermochemical nonequilibrium <p>* Examples will be provided in C++ and Fortran</p>
3:15 pm	<i>Coffee break</i>
3:30 pm	Coding with Mutation++ continued
4:00 pm	Discussion
4:30 pm	Closure