# Dust Destruction in Core Collapse Supernovae

Sarah Stangl, Chris Mauney

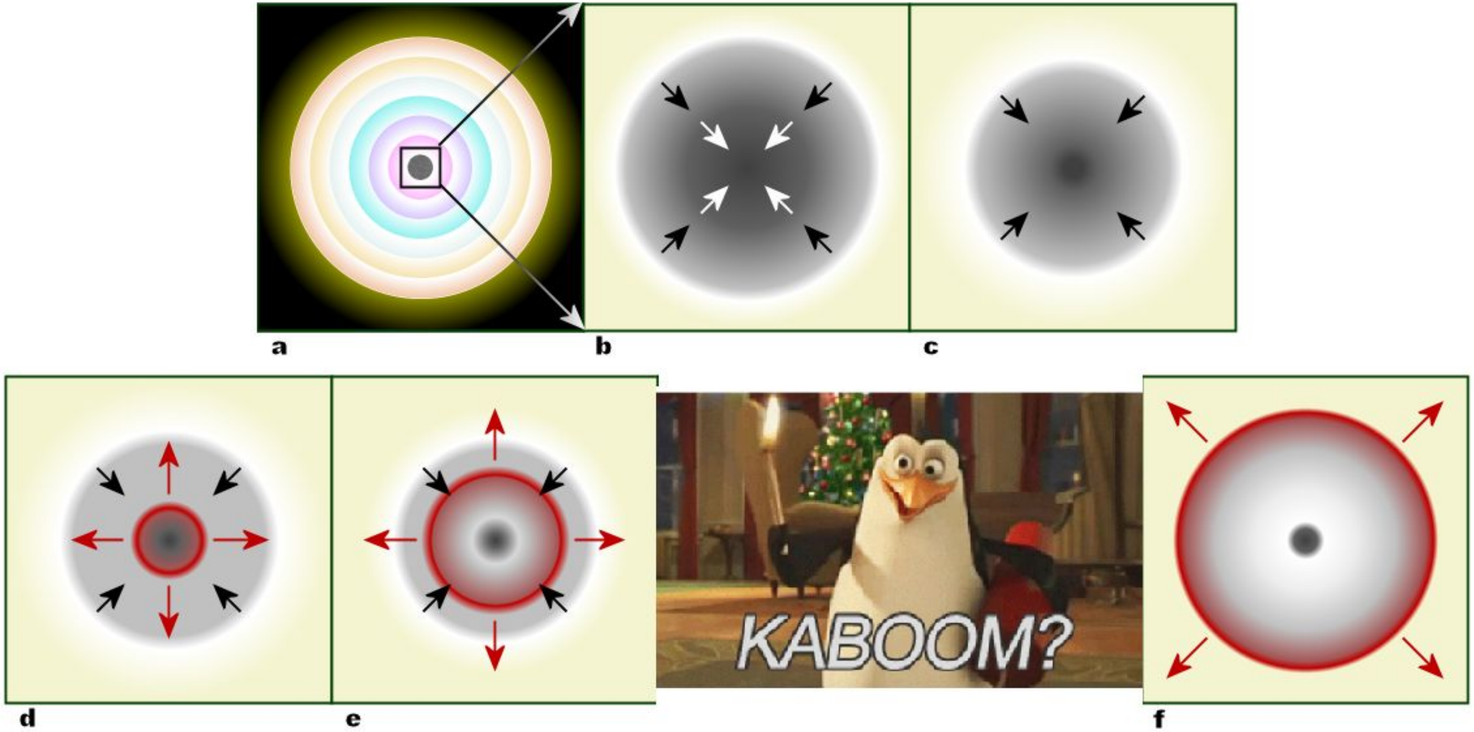
Mentors: Chris Fryer

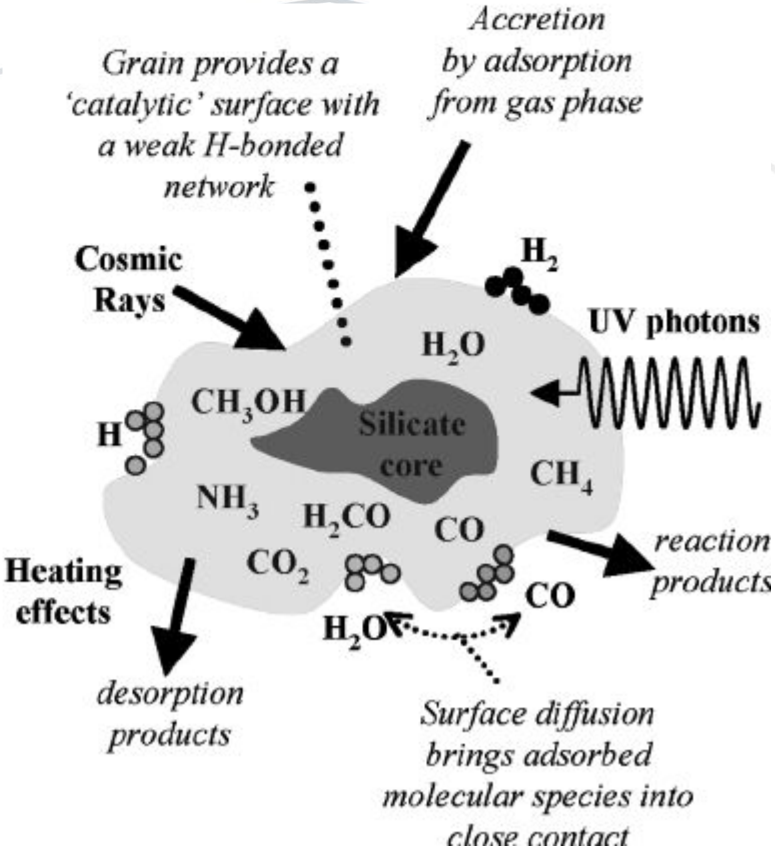
LA-UR-21-28201

# Why Dust?

- It is everywhere in space!
- Distorts light:
  - Absorbs and re-emits light in longer wavelengths
- Seed for more complicated molecules
- Stellar and galactic formation and evolution
  - enriches ISM, relaxes and cools proto-stars/galaxies
- Pre-Solar grains:
  - isotopic signature of stars + fusion processes
- Molecular lines:
  - composition of object and underlying physics

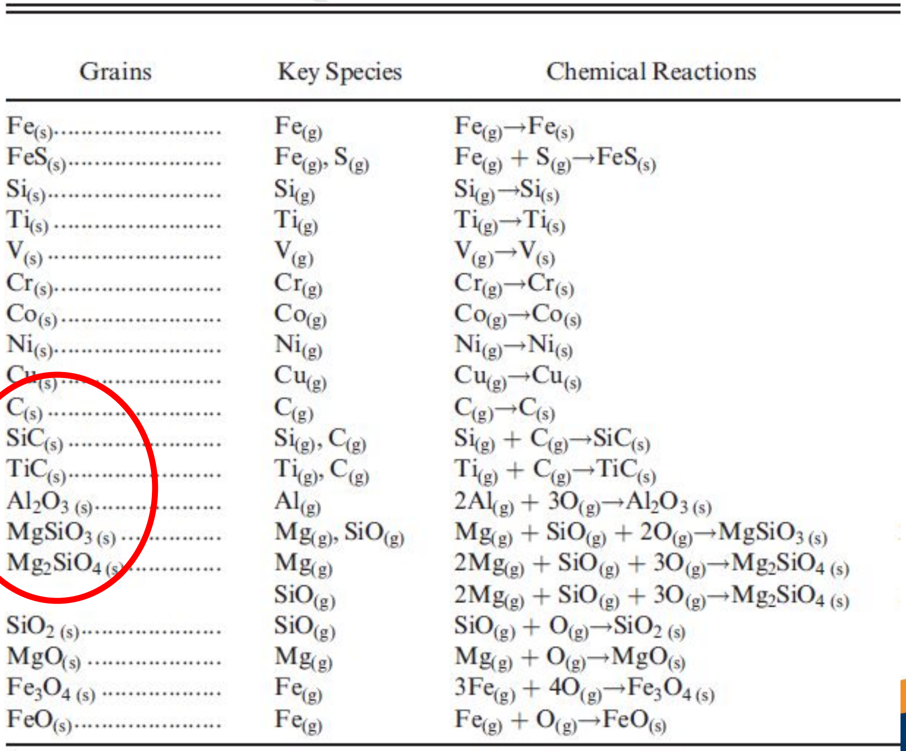
# Core Collapse Supernovae (CCSNe)




Grain provides a 'catalytic' surface with a weak H-bonded network
Accretion by adsorption from gas phase
Cosmic Rays
$H_2$
UV photons
$H_2O$
$CH_3OH$
Silicate core
H
$CH_4$
$NH_3$
$H_2CO$
CO
$CO_2$
Heating effects
reaction products
CO
$H_2O$
desorption products
Surface diffusion brings adsorbed molecular species into close contact

# Formation of Dust - Key Species

- Nucleation rate
    - governed by key species
        - the reaction rate is much larger than the decay rate
        - species with the least collisional frequency, controls nucleation and growth



| Grains | Key Species | Chemical Reactions |
|---|---|---|
| $Fe_{(s)}$ | $Fe_{(g)}$ | $Fe_{(g)} \rightarrow Fe_{(s)}$ |
| $FeS_{(s)}$ | $Fe_{(g)}$, $S_{(g)}$ | $Fe_{(g)} + S_{(g)} \rightarrow FeS_{(s)}$ |
| $Si_{(s)}$ | $Si_{(g)}$ | $Si_{(g)} \rightarrow Si_{(s)}$ |
| $Ti_{(s)}$ | $Ti_{(g)}$ | $Ti_{(g)} \rightarrow Ti_{(s)}$ |
| $V_{(s)}$ | $V_{(g)}$ | $V_{(g)} \rightarrow V_{(s)}$ |
| $Cr_{(s)}$ | $Cr_{(g)}$ | $Cr_{(g)} \rightarrow Cr_{(s)}$ |
| $Co_{(s)}$ | $Co_{(g)}$ | $Co_{(g)} \rightarrow Co_{(s)}$ |
| $Ni_{(s)}$ | $Ni_{(g)}$ | $Ni_{(g)} \rightarrow Ni_{(s)}$ |
| $Cu_{(s)}$ | $Cu_{(g)}$ | $Cu_{(g)} \rightarrow Cu_{(s)}$ |
| $C_{(s)}$ | $C_{(g)}$ | $C_{(g)} \rightarrow C_{(s)}$ |
| $SiC_{(s)}$ | $Si_{(g)}$, $C_{(g)}$ | $Si_{(g)} + C_{(g)} \rightarrow SiC_{(s)}$ |
| $TiC_{(s)}$ | $Ti_{(g)}$, $C_{(g)}$ | $Ti_{(g)} + C_{(g)} \rightarrow TiC_{(s)}$ |
| $Al_2O_{3\,(s)}$ | $Al_{(g)}$ | $2Al_{(g)} + 3O_{(g)} \rightarrow Al_2O_{3\,(s)}$ |
| $MgSiO_{3\,(s)}$ | $Mg_{(g)}$, $SiO_{(g)}$ | $Mg_{(g)} + SiO_{(g)} + 2O_{(g)} \rightarrow MgSiO_{3\,(s)}$ |
| $Mg_2SiO_{4\,(s)}$ | $Mg_{(g)}$ | $2Mg_{(g)} + SiO_{(g)} + 3O_{(g)} \rightarrow Mg_2SiO_{4\,(s)}$ |
|  | $SiO_{(g)}$ | $2Mg_{(g)} + SiO_{(g)} + 3O_{(g)} \rightarrow Mg_2SiO_{4\,(s)}$ |
| $SiO_{2\,(s)}$ | $SiO_{(g)}$ | $SiO_{(g)} + O_{(g)} \rightarrow SiO_{2\,(s)}$ |
| $MgO_{(s)}$ | $Mg_{(g)}$ | $Mg_{(g)} + O_{(g)} \rightarrow MgO_{(s)}$ |
| $Fe_3O_{4\,(s)}$ | $Fe_{(g)}$ | $3Fe_{(g)} + 4O_{(g)} \rightarrow Fe_3O_{4\,(s)}$ |
| $FeO_{(s)}$ | $Fe_{(g)}$ | $Fe_{(g)} + O_{(g)} \rightarrow FeO_{(s)}$ |

Nozawa et al. 2003

# Dust Growth via grain nucleation

- Growth (key species)
    - material collides and sticks to the grain
    - once the key species is used up, reaction stops
    - abundance of key species is determined by a system of coupled nonlinear ODEs

radius
$$\frac{dr_j}{dt} = \alpha_{sj}\Omega_j\left(\frac{kT}{2\pi m_{1j}}\right)^{1/2}$$

concentration
$$c_{1j}(t) = \frac{1}{3}a_{0j}\tau^{-1}_{\mathrm{coll},j}(t)$$

- Moment Equations

$$\frac{dK_j^{(0)}}{dt} = \frac{J_j(t)}{\tilde{c}_{1j}(t)}\frac{4\pi}{3\Omega_j}$$

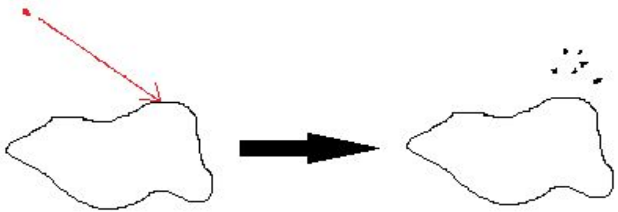
$$\frac{dK_j^{(i)}}{dt} = \frac{J_j(t)}{\tilde{c}_{1j}(t)}\frac{4\pi}{3\Omega_j}r^i_{c,j} + iK_j^{(i-1)}\frac{dr_j}{dt} \quad (\text{for } i = 1\text{–}3)$$

K0: grain number density, K1: average radius, K2: average surface area, K3: key species depletion

Nozawa et al. 2003
Nozawa et al. 2013

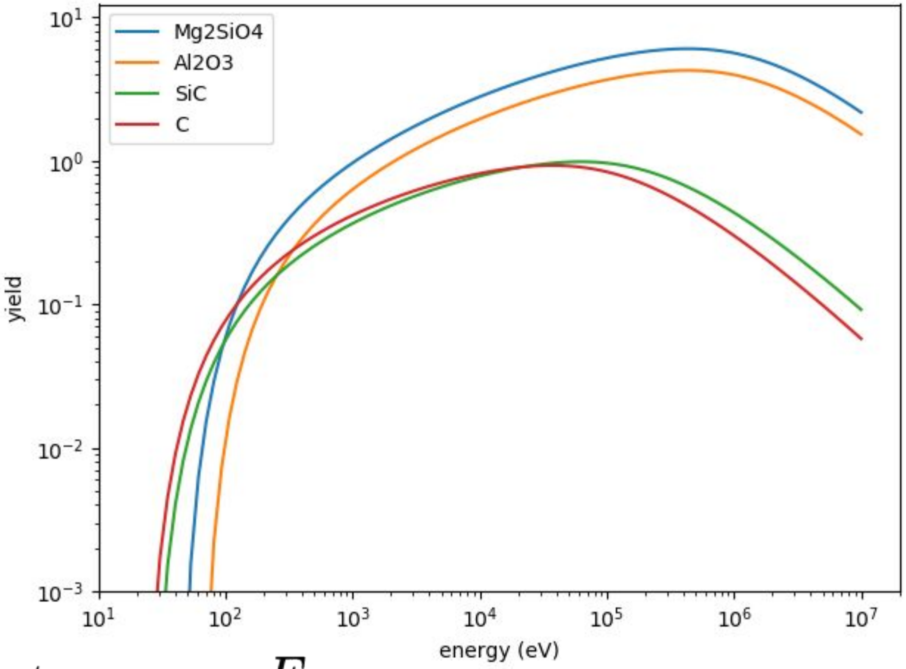
# Sputtering

- Chemical: gas or reactive ion interacts with the grain's surface forming an unstable compound
  - the instabilities cause material to sputter off the grain's surface
  - occurs at low energies
- Physical: kinetic energy from the colliding ion/particle is transferred to the grain
  - with enough energy to overcome surface binding forces, material sputters off the grain
  - occurs at high energies



# Sputtering Yield

- The amount of sputtered atoms per ion.
- Depends on the nuclear stopping cross section, surface binding energy, the threshold energy (min KE), and the energy of the incoming particle.



$$Y_i(E) \approx \frac{S_i(E)}{U_0}[1-(\frac{E_{th}}{E})^{2/3}](1-\frac{E_{th}}{E})^2$$

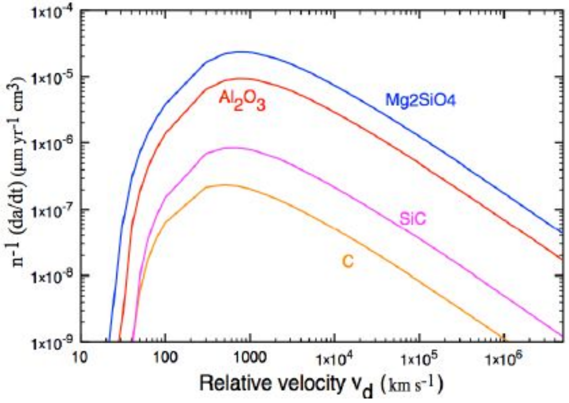
# Grain Erosion Rate

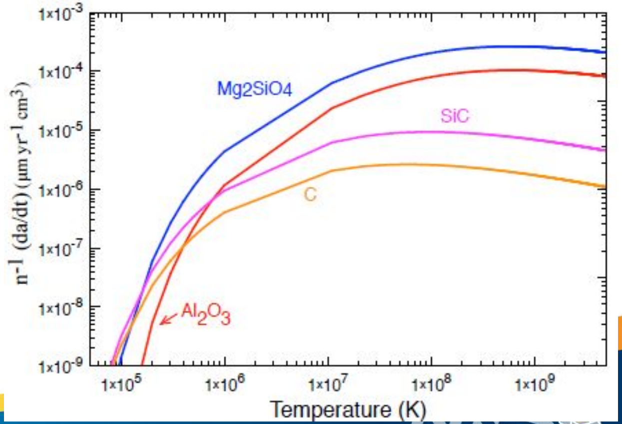
- Non-thermal sputtering: non-thermal sputtering erodes a hypersonic grain

$$\frac{1}{n_H}\frac{da}{dt} \approx -v_d \sum A_i Y_i(E = 1/2 m_i v_d^2)$$

- Thermal sputtering: the grain moves with the shock and collide with the ionized gas

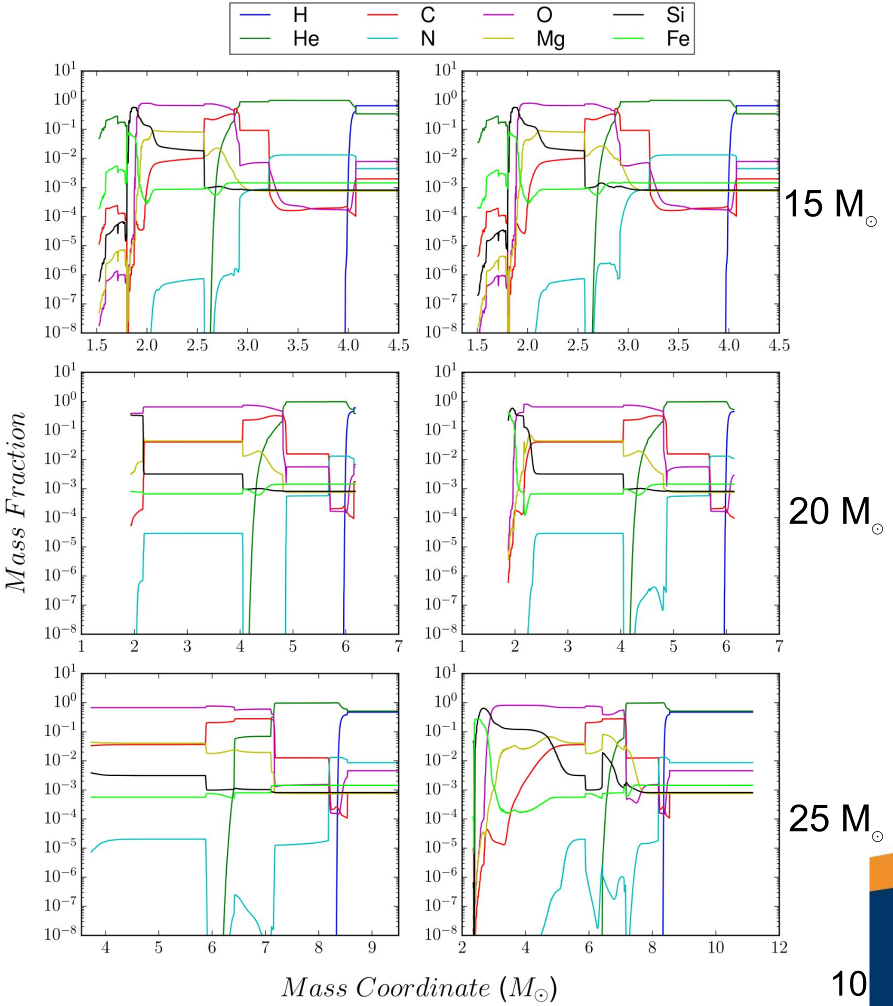
$$\frac{1}{n_H}\frac{da}{dt} \approx -\sum A_i \left(\frac{8kT}{\pi m_i}\right) \int \epsilon_i e^{-\epsilon_i} Y_i(\epsilon_i) d\epsilon_i$$





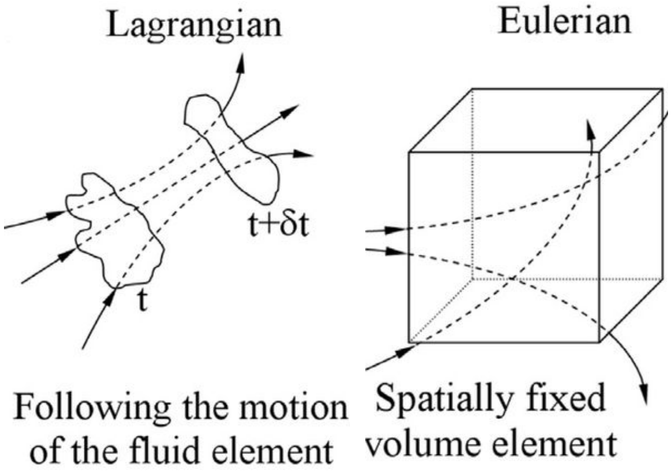
# CCSNe Models

- Models (Fryer et al. 2018)
  - Progenitor mass: 15, 20, 25 $M_\odot$
  - Explosion energy: 0.5 - 125 foe
  - Unmixed ejecta: No Mixing!
  - 1-D: assumes spherical symmetry



# Hydrodynamical-code (Hydrocode)

- 1-D Lagrangian
- Remove compact core
- Add thermal stellar wind profile onto the stellar surface
- Evolve ejecta out to 1157 days
  - Allows for cooling and expansion of ejecta to values agreable to dust formation



# Code

- *nuDust*: **nu**cleating **dust** code in python
- Takes in composition and hydrodynamical profiles
- Pre-formation of CO and SiO gas phase molecules
- Solves system of coupled nonlinear ODEs for all grain species simultaneously
  - LSODA integrator
    - switches between the nonstiff Adams method and the stiff BDF method
- Parallelization: *multiprocessing* library

# JIT

- *Numba* for just-in-time (*JIT*) compilation to increase efficiency and optimization
  - converts Python to optimized machine code
  - runs at native machine code speed
- Good for code with:
  - a lot of math **(faster intrinsics)**
  - for loops **(vectorization / parallelization)**
  - Numpy **(some) routines converted to C/CUDA function calls**
- Large “one-time” cost at runtime (for compiling)
- Can improve runtime ~$10^6$x, **tho larger/more complicated codes see more modest overall boost.**

# JIT in nuDust

- Ideal: use Python to load data, setup calculation, and handle I/O (e.g. one-time and/or low-cost code) & use Numba to do generate efficient integration code (i.e. heaviest workload)
- In reality:
  - Numba struggles to capture all but the simplest types (e.g. numbers). Much of the actual “low-level” aspects need to be hand-crafted
  - Communicating state-specific data (error states, intermediate values) is not natively supported in Numba
- What’s been done:
  - With recent addition of Numba-accelerated interpolation, the forward-rate evaluation is completely compiled by LLVM.

$$\frac{dy}{dt} = \boxed{f(t, x)}$$

# CUDA

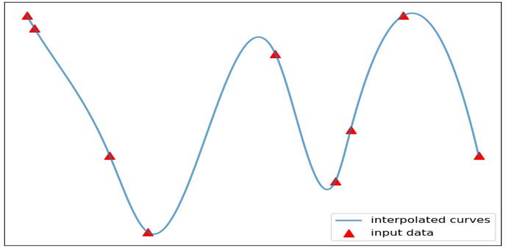
- translates Python functions into PTX (Parallel Thread Execution) code
    - graphics driver converts PTX code into binary code and runs on GPUs
- Large overhead
    - use blocked algorithm to reduce memory calls
- Uses shared memory for threads in a block
    - loads small blocks at a time

# CUDA (not) in nuDust

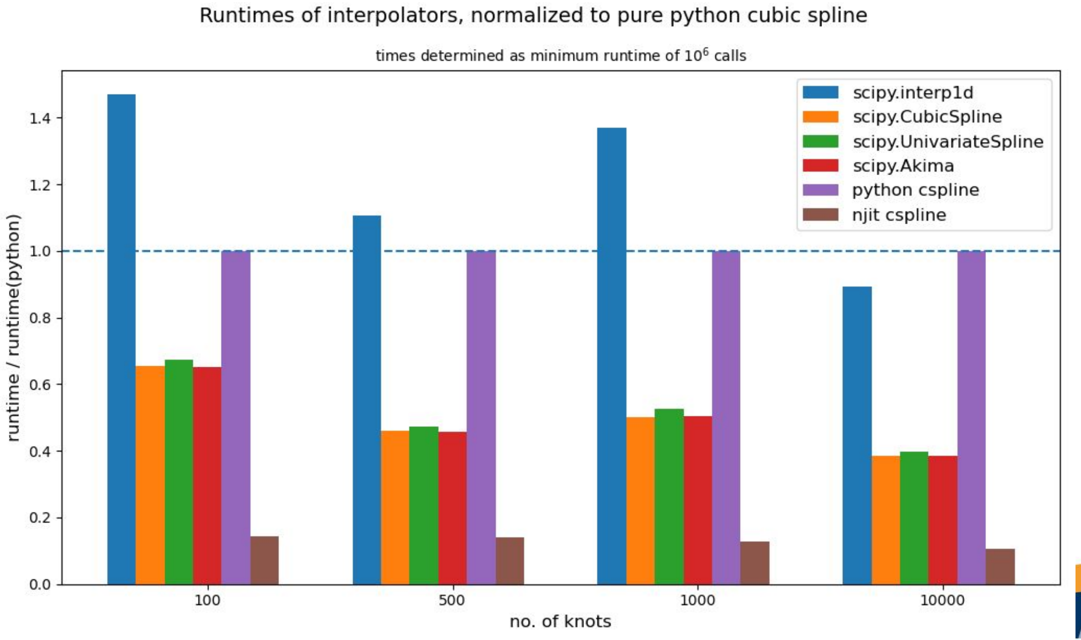
- While Numba-LLVM has trouble with advanced types, Numba-CUDA seems mostly incapable.
- Some routines can be easily translated, however they are too short/serial to retrieve performance from.
- nuDust requires a substantial amount of refactoring to get benefit from CUDA-enabled Numba

# JIT in nuDust - Interpolation

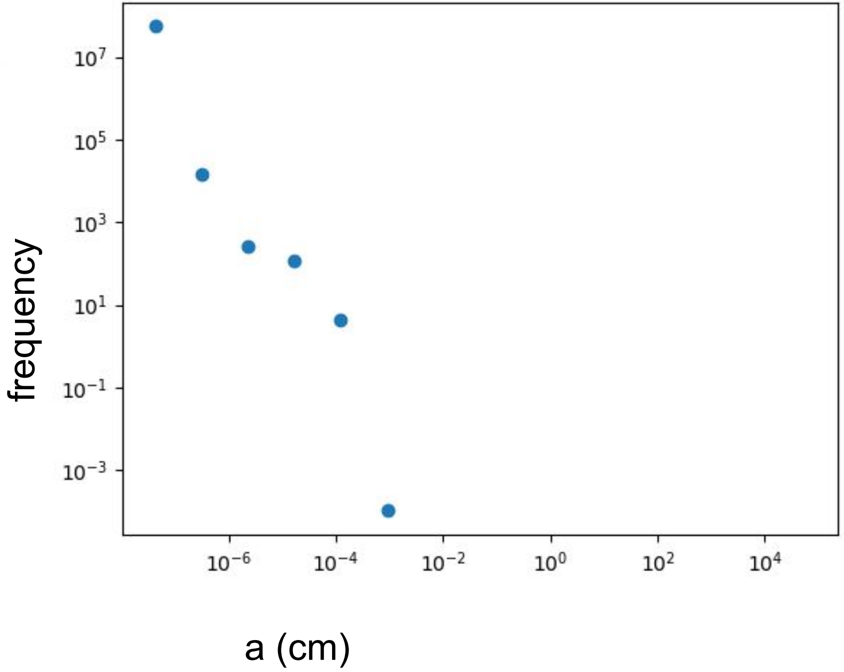
The initial integrator used the built-in scipy interpolators. Using a Numba-accelerated interpolator, this stage of the forward rate evaluation is significantly faster

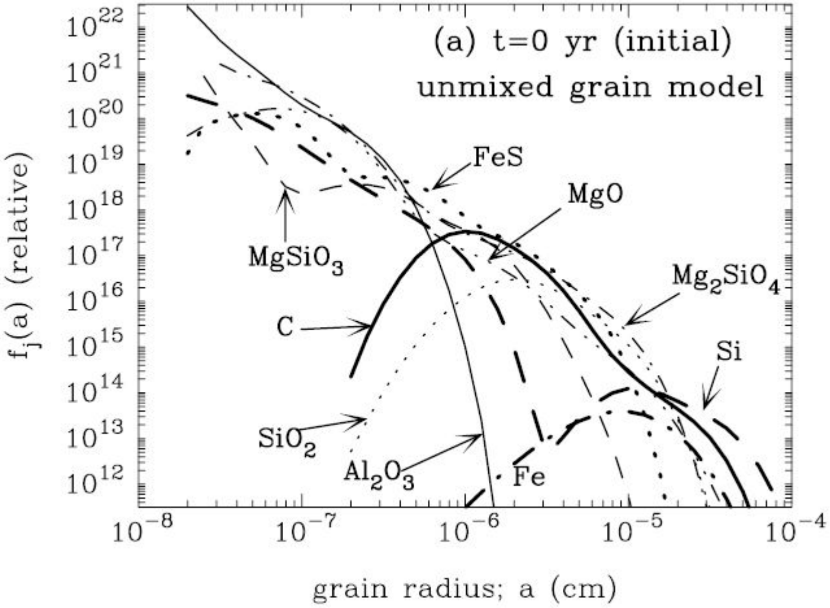


Interpolation is a critical component of the determining the forward rate - we must be able to reconstruct values that fall between the input datapoints. This takes up ~50% of FLOPS this stage.



# Results





Nozawa 2006

# Future Work

- Include more physics
  - grain accretion, gas chemistry, etc.
- Produce Spectra + Light Curves
  - Look for impacts of grains on spectral lines
- Compare dust and spectra with Observations
  - SN IIb?
- More efficient integrator, refactor of code for vectorization/CUDA performance.

# Thanks for Listening

Questions?