

```

m1*W1; // double p2 = scosh(theta2)*(1.0+cos(theta3)); // double M2 = e*sinh(theta2)-p2; // double tz = sq
// Distance at which we sample from the halo distribution double R=100.0*km; if(R<100.0*km) { cerr << "Warning in
n, std::placeholder<_X,vEarth>); double u = Rejection_Sampling(pdf,0.0,(vEarth+vEarth),vmax,PRNG); //Velocity Directio
n. //Random Point on a flat disk at distance R Eigen::Vector3d exi=vini.normalized(); Eigen::Vector3d exj,exk;
exj=Disk*(cos(phi)*exi+sin(phi)*ey); // cout << "Norm = "<<xIni.norm()<<endl; // cout << "Norm = "<<(Bxexi.norm()<<
exi.norm()); rsum<<"t<<xc.Position()[1]/rsum<<"t<<xc.Position()[2]/rsum<<"t<<xc.Velocity();
Keppler_Shift(ic,RMAX,model); } //2. orbit simulation /Right hand sides of the 1st order equations of motion. double
ion with Euler-Cromer // void EC_Step(double &t,double Ar,double Aphi,double Aphi,double dt,sum1Model Amodel) {
del) // //RK4 steps: // double k_r[1]; // double k_v[4]; // double k_p[4]; // k_r[0]=dt*drdt(v); // k_r[0]
k_r[2]=dt*dvdt(r+k_r[1]/2.0,J,model); // k_p[2]=dt*dphidt(r+k_r[1]/2.0,J); // k_r[3]=dt*drdt(v+k_v[2])
2.0*k_v[2]+k_v[3]); // phi+=1.0/6.0*(k_p[0]+2.0*k_p[1]+2.0*k_p[2]+k_p[3]); // //ODE integration with Runge-Kutta Fehl
irdt(v); k_v[0]=dt*dvdt(r,J,model); k_p[0]=dt*dphidt(r,J); k_r[0]=dt*drdt(v+k_v[0]/4.0); k_v[1]=dt*dvdt(r+k_r[0]/4.0)
+9.0/32.0*k_v[2]); k_r[3]=dt*drdt(r+1932.0/2197.0*k_v[0]-7299.0/2197.0*k_v[1]+7296.0/2197.0*k_v[2]); k_v[3]=dt*dvdt
v[0]-8.0*k_v[2]+3688.0/553.0*k_v[3]-348.0/4180.0*k_v[4]); k_v[4]=dt*dvdt(r+439.0/216.0*k_r[0]-8.0*k_r[1]
+354.0/2565.0*k_v[2]+1150.0/4184.0*k_v[3]-11.0/40.0*k_v[4]); dt=dvdt(r-8.0/27.0*k_r[0]+2.0*k_r[1]-354
rors double r+=r+25.0/216.0*k_r[0]+1488.0/2565.0*k_r[1]+2197.0/4181.0*k_r[3]-1.0/5.0*k_r[4]; double v+=v+25.0/216.0
/12825.0*k_r[2]+28581.0/56430.0*k_r[3]-9.0/50.0*k_r[4]+2.0/55.0*k_r[5]; double v5=v+18.0/155.0*k_v[0]+858.0/12825.0*k_v[1]
{abs(r5-r4),abs(v5-v4),abs(phi5-phi4)}; std::vector<double> tolerance={1.0e-6,1e-3,km/sec,1e-6}; //New stepsize
dm(delta); std::end(delta)); } //Next steps if(err[0]<tolerance[0]&err[1]<tolerance[1]) { if(r<r
eed); logmi+=speed*dt/mfp; } t+=dt; r=r4; v=v4; phi=phi4; dt=std::min(max(dtMin, std::min(delta,dtMax)); } else { dt
= J/std::vector<Eigen::Vector3d> axes); { double w_phi = J/pow(r,2); Eigen::Vector3d xNew = rx*(cos(phi)*axes[0]*sin(phi)*w
velocity().normalized(); Eigen::Vector3d ex_x_y = mx_z.cross(mx_x); std::ivector<Eigen::Vector3d> exw = {mx_x,mx_y
velocity()).dot(ex_x); //Sample -log(1-x) double logK1 = -1.0*log(1.0-ProbabilitySample(PRNG)); //Start integr
as); RK45_step(t,r,v_r,phi,J,dt,logxi,DM,model); if(xOld==0) { Event xNow=PlaneTo3D(t,r,phi,v_r,J,axes); if(xNow.Posit
+xNow.Speed())-model.vEsc2(xNow.Radius())); <<"t<<xNow.Speed(),km/sec)<<endl; } // double dd=(xOld.Position
lar moment // f<<xNow.Position()[0]/rSun<<"t<<xNow.Position()[1]/rSun<<"t<<xNow.Position()[2]/rSun<<"t<</sec
use if(rOld<rMax&&rMax<=rMax&&(v_r*x_r+r*x_r)/model.vEsc2(r)) { propagatetfalse; x0=PlaneTo3D(t,r,phi,v_r,J,axes); } else
Scatter(Event& x0,DM_Particle& DM,UnitModels model, std::mt19937s PRNG) { // double speed=x.Speed(); // cout << "Scatt
icalCoordinate<1.0, ThetaSample(PRNG), PhiSample(PRNG) double vmat = (UTarget-x0.Velocity()).norm(); fig
eed)/speed<<endl; // if(speed>sqrt(model.vEsc2(r))&&xNow.norm()>sqrt(modet.vEsc2(r))) cout << "capture"<<endl; // 
ns3Model& model, unsigned int nScattering, std::mt19937s PRNG) { // Save trajectory of stream
ss/2.0*(x.Speed()*x.Speed())-model.vEsc2(x.Radius()))/E0<<endl; //Output // Event x = ic; bool success; //Counters nScat
tering if(r<rSun) { if(nScattering>nScattering_max) { success = false; break; } // double E1=DM.mass/2.0*(x.
mass/2.0*(x.Speed()*x.Speed())-model.vEsc2(x.Radius()))/E0<<endl; // cout << "capture"<<endl;
s<<"t<<r/rSun << endl; } / f. c. s<< endl;
xFinal.Position()[0]/rSun<<"t<<xFinal.Position()[1]/rSun<<"t<<xFinal.Position()[2]/rSun<<"t<<xFinal.Radius()
min) success=false; if(x.Time()/nsc<9999) return false; return success; } Result Generate_Data(Configurations config
==config.numproc>1) ? 0 : config.rank>1; int tag = 0; MPI_Status status; MPI_Request send_request; // /Datapoi
nPoint> data; data.resize(config.nRings); //Counters //Total number of simulated particles unsigned long int G_Cou
n,0); std::vector<unsigned long int> L_Counter_Data_new(config.nRings,0); //Number of passes // std::vector<unsigned
rage number of scatterings: std::vector<double> o_nScattering_AV(config.nRings,0.0); std::vector<double> L_nSc
} { MPI_Isend(&L_Counter_Data.front(),config.nRings,MPI_UNSIGNED_LONG,destination,tag,MPI_COMM_WORLD
odel,PRNG); unsigned int nScattering=0; bool success = Simulate_Trajectory(x,DM,model,nScattering,PRNG);
] -1.0)*L_nScattering_AV(IsoRing)+nScattering)/L_Counter_Data[IsoRing]; double speed=x.Speed(); f[nS
AG,MPI_COMM_WORLD,flag,&status]; if(flag) { //Receive the tokens MPI_Recv(&L_Counter_Data.front(),config.nRi
G_Counter_Data[i]+&L_Counter_Data_new[i]; L_Counter_Data_new[i]=0; } // cout << config.rank << "t" << G_Count
source<1; else if ((min1>config.nRings) tag = status.MPI_TAG; //Pass on the tokens, unless you are the very la
rk/BarLength cout <<"\r"; for(int i=0;i<2.0*BLength;i++) cout << " "; cout <<"\r"; for(int i=0;i<BarLength
,Free,&Counter_Free,1,MPI_UNSIGNED_LONG_LONG,MPI_SUM,MPI_COMM_WORLD); MPI_Allreduce(&L_nScat
taPoint> Global_Data for(int i = 0;i<config.nRings;i++) { unsigned long int Global_SampleSize; MPI
; int recv_displ[config.numproc]; MPI_Accumulate(&L_Counter_Data[i],1,MPI_UNSIGNED_LONG,&data_loc
(Global_SampleSize); // MPI_Gatherv(data[i].front(),data.size(),mpi_datapoint,&Ring_Data.front()
>Data); } MPI_BARRIER(MPI_COMM_WORLD); // For(int i=0;i<config.nRings;i++) // { // If(config.r
Adoration,1,MPI_DOUBLE,MPI_MAX,MPI_COMM_WORLD); return Result(Global_Data,0,M_Counter_Simulat
t nfrom, std::vector<double> nscav, double dt) { data=dt*particle - p; // sample size
size[i]); nScatterings_mean=nscav; nScatterings_mean_tot=0.0; for(unsigned int i=0;i<
uration & config) { if(config.rank==0) { std::cout << "Simulation sum<< endl; if(config.nRings==1) st
ations)<<, <<Round(1.0*nScatterings/duration)<<"/sec)"<<std::endl; if(config.nRings==1) st
sampl_size[i] << endl; <<Round(nScatterings_mean[i])<<std::endl; } } ); #include<
Physics/nfunctions.hpp> using namespace libonfig; //1. Struct with input para
ring nfrom; //Parallelisation rank = myrank; numproc= np; & config
//com: FileIOException &fioex) { std::cerr << "I/O error while re
try { size = cfg.Lookup("simid").c_str(); } catch(com
file. << endl; exit(EXIT_FAILURE); } // Error
"None,"<<endl<<endl; } //2. Event

```

# Solar reflection of light dark matter

Timon Emken  
Stockholm University



timon.emken@fysik.su.se



timonemken.com



@TimonEmken



/temken

State of the Universe Seminar  
Department of Theoretical Physics,  
TIFR, Mumbai  
October 29, 2021

Based on  
[arXiv:2102.12483]



# Solar reflection of sub-GeV DM

- I. Direct detection of (sub-GeV) dark matter
- II. Dark matter in the Sun
- III. Monte Carlo simulation of solar reflection
- IV. Results
- V. Summary & Outlook

I.

# Direct Detection of (sub-GeV) Dark Matter

# How do we know about dark matter?

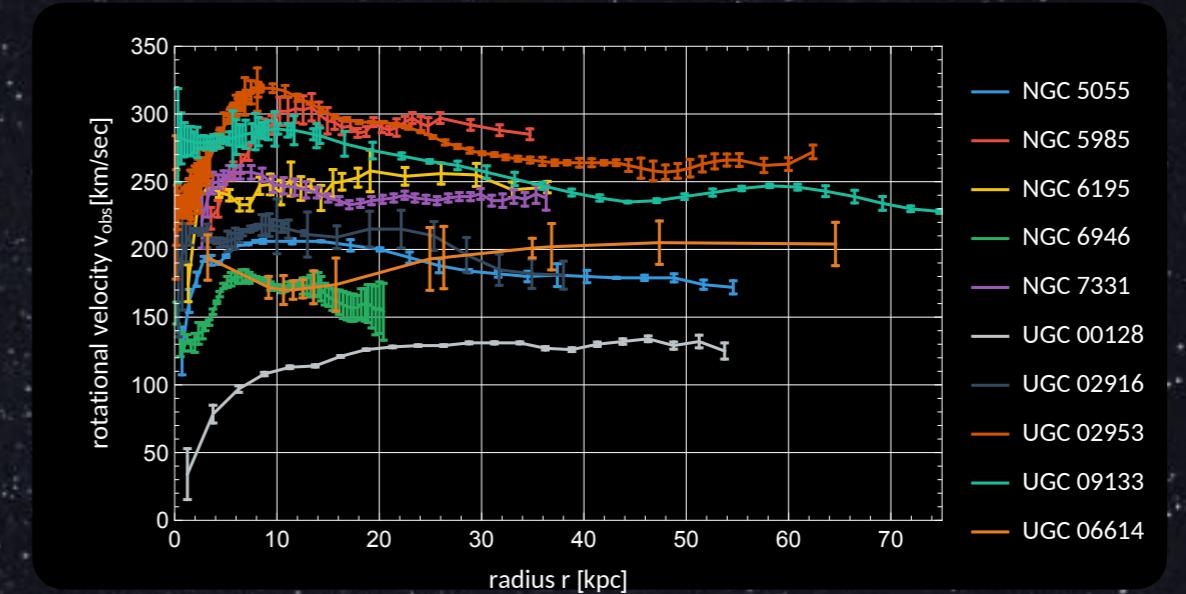
(a) Fritz Zwicky and the Coma cluster



Photo: F. Clark (1971)

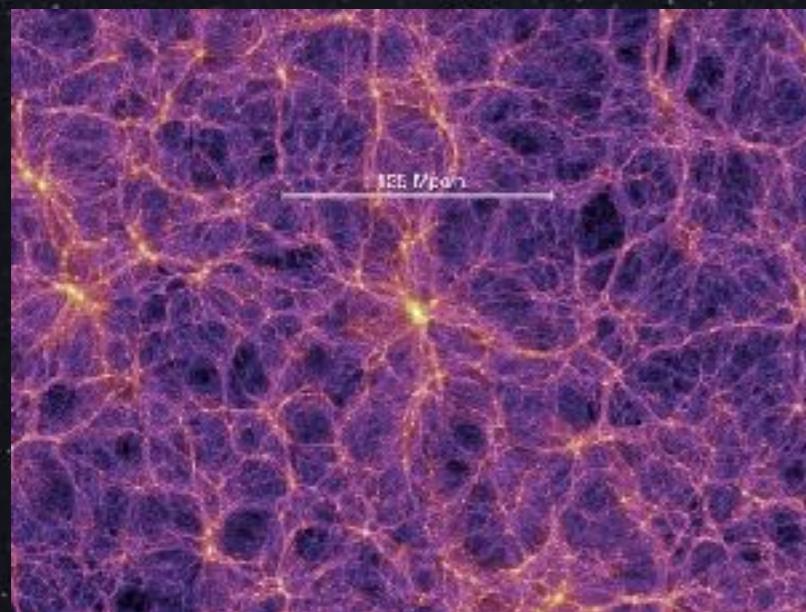
F. Zwicky, Helv. Phys. Acta 6 (1933), 249

(b) Galactic rotation curves



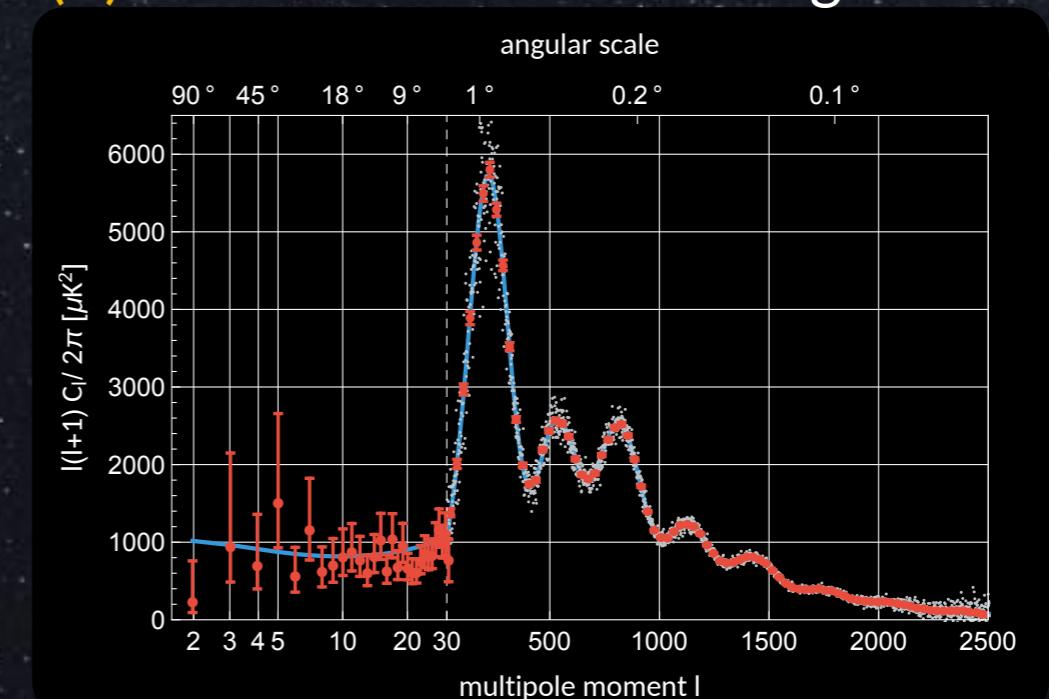
F. Lelli et al., Astron. J. 152 (2016), 157

(c) Cosmological structure formation



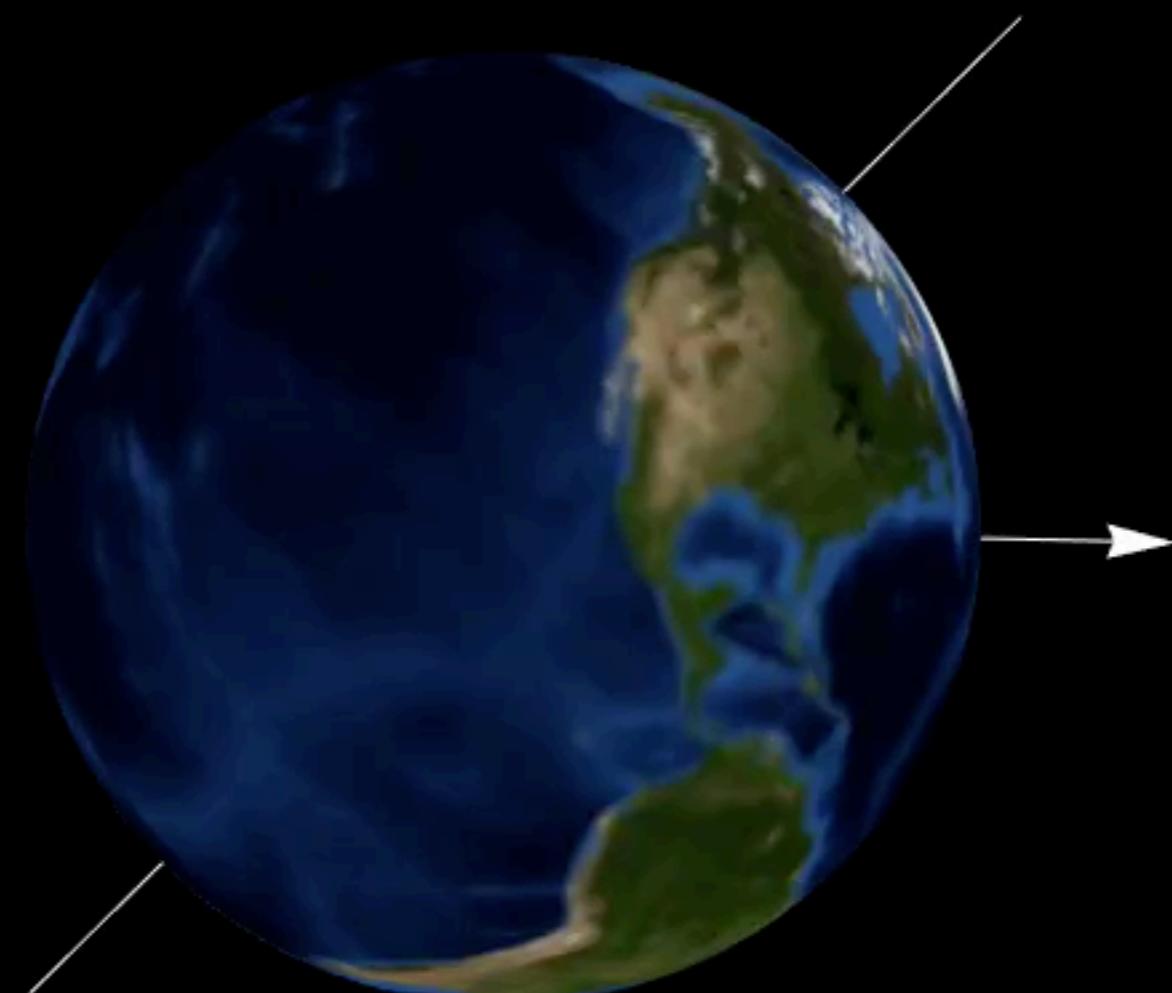
Springel et al., Nature 435 (2005), 629-636

(d) Cosmic microwave background



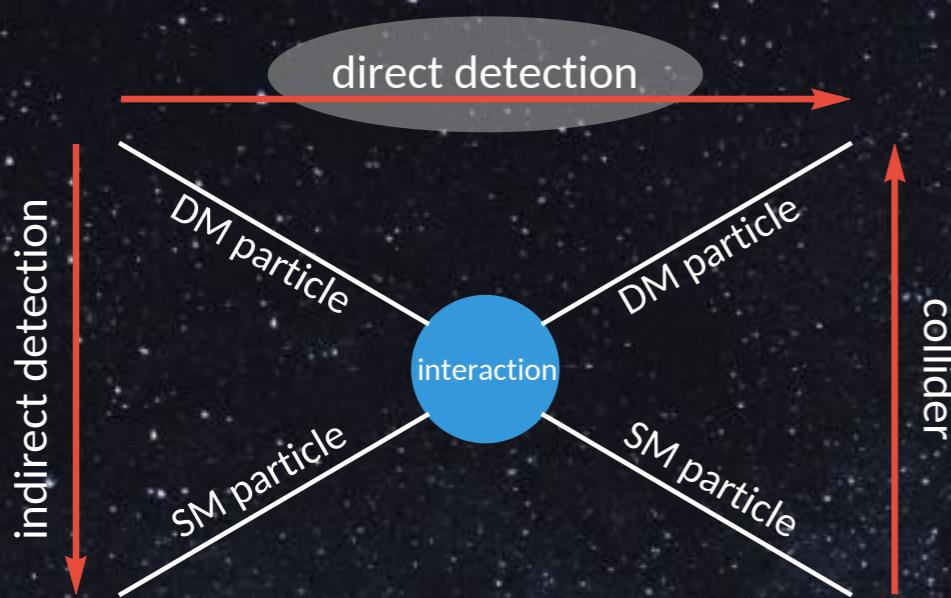
Planck Collaboration, Planck Legacy Archive, (2018)

# The dark matter “wind”

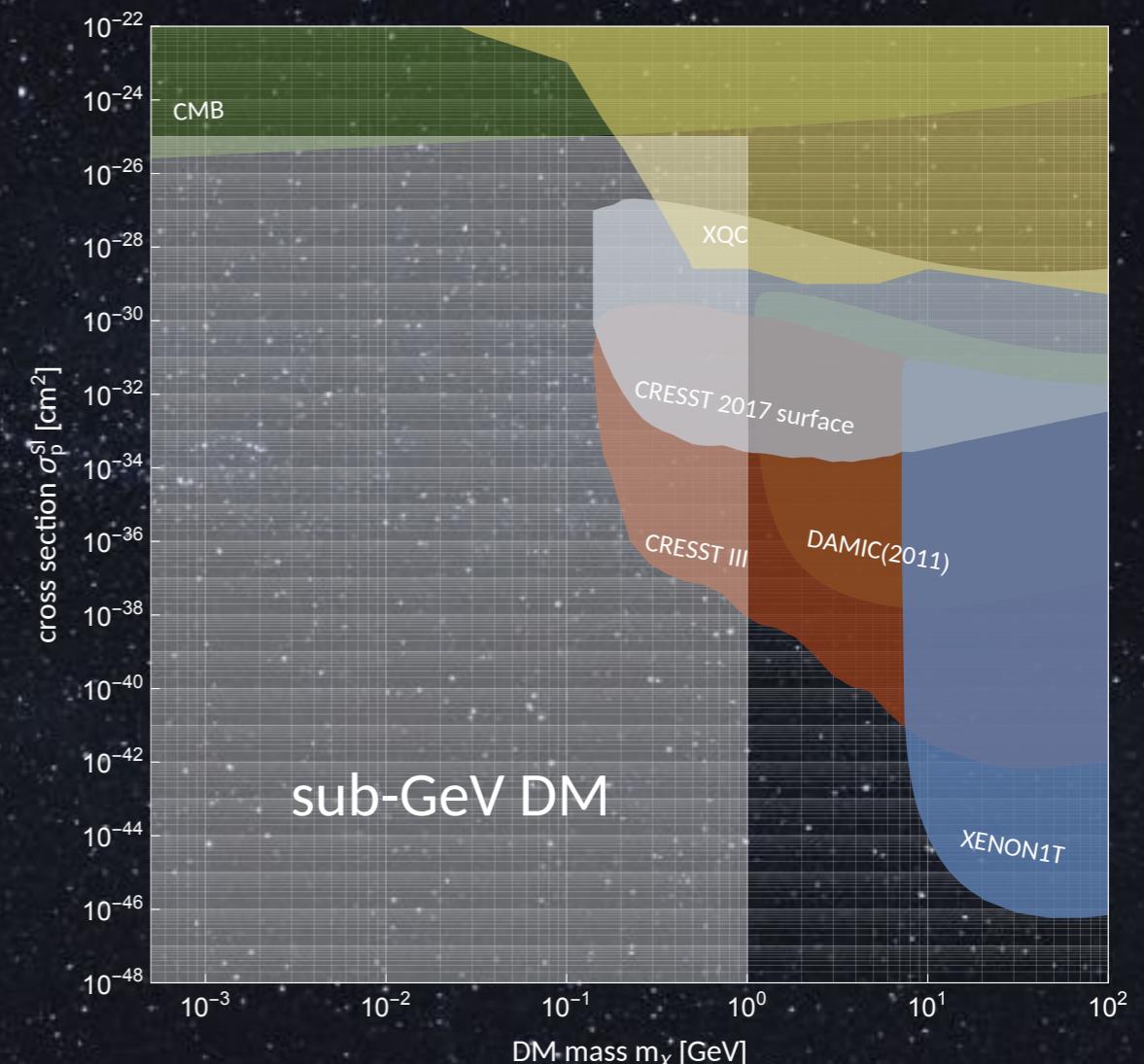
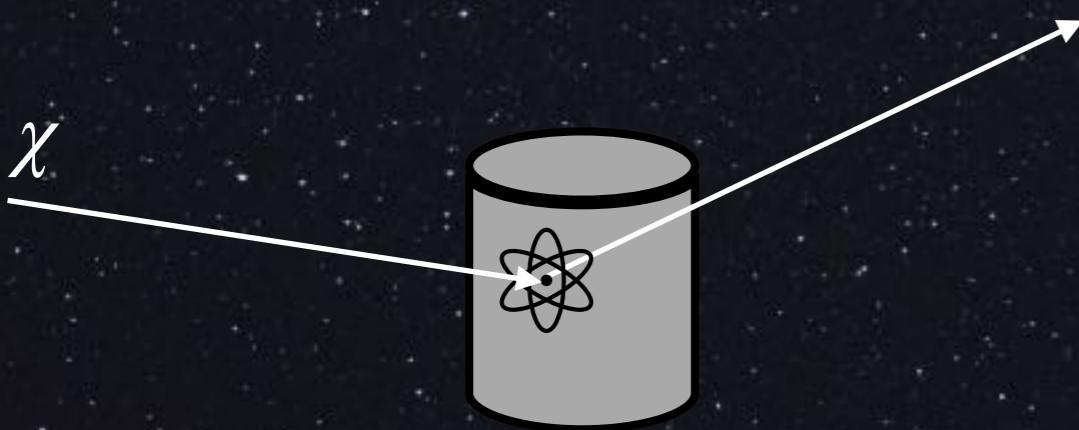


Emken 2016

# Direct detection of dark matter

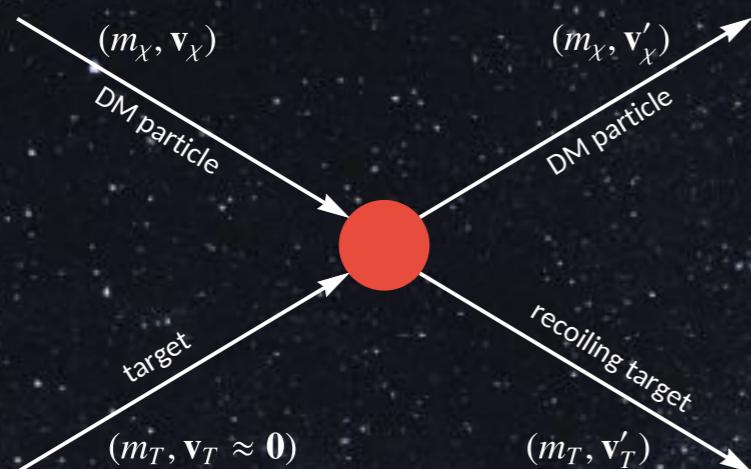


**Basic idea:** Measure the energy deposit of a DM-matter collision inside a detector.



- Conventional direct DM searches lose sensitivity to DM masses below  $\sim 1$  GeV.
- Why? And what can be done to probe lighter masses?

# Detecting DM via nuclear recoils



Event spectrum:

$$\frac{dR}{dE_R} = N_T \frac{\rho_\chi}{m_\chi} \iiint d^3v \, v f_\chi(v) \frac{d\sigma_N}{dE_R} \Theta(v - v_{\min}(E_R)).$$

Detector size  
Astrophysics

Particle physics  
Kinematics

DM velocity distribution:

$$f_{\text{halo}}(\mathbf{v}) = \frac{1}{N_{\text{esc}} \pi^{3/2} v_0^3} \exp\left(-\frac{\mathbf{v}^2}{v_0^2}\right) \Theta(v_{\text{gal}} - |\mathbf{v}|)$$

sharp speed  
cutoff

# The low mass frontier of DM searches

- Typical nuclear recoil:

$$E_R^{\text{typ}} \sim \frac{2m_\chi^2 v_\chi^2}{m_T} \approx 2 \text{ keV}$$

$$\times \left( \frac{m_\chi}{10 \text{ GeV}} \right)^2 \left( \frac{v_\chi}{10^{-3}} \right)^2 \left( \frac{m_T}{m_{\text{Xe}}} \right)^{-1}$$

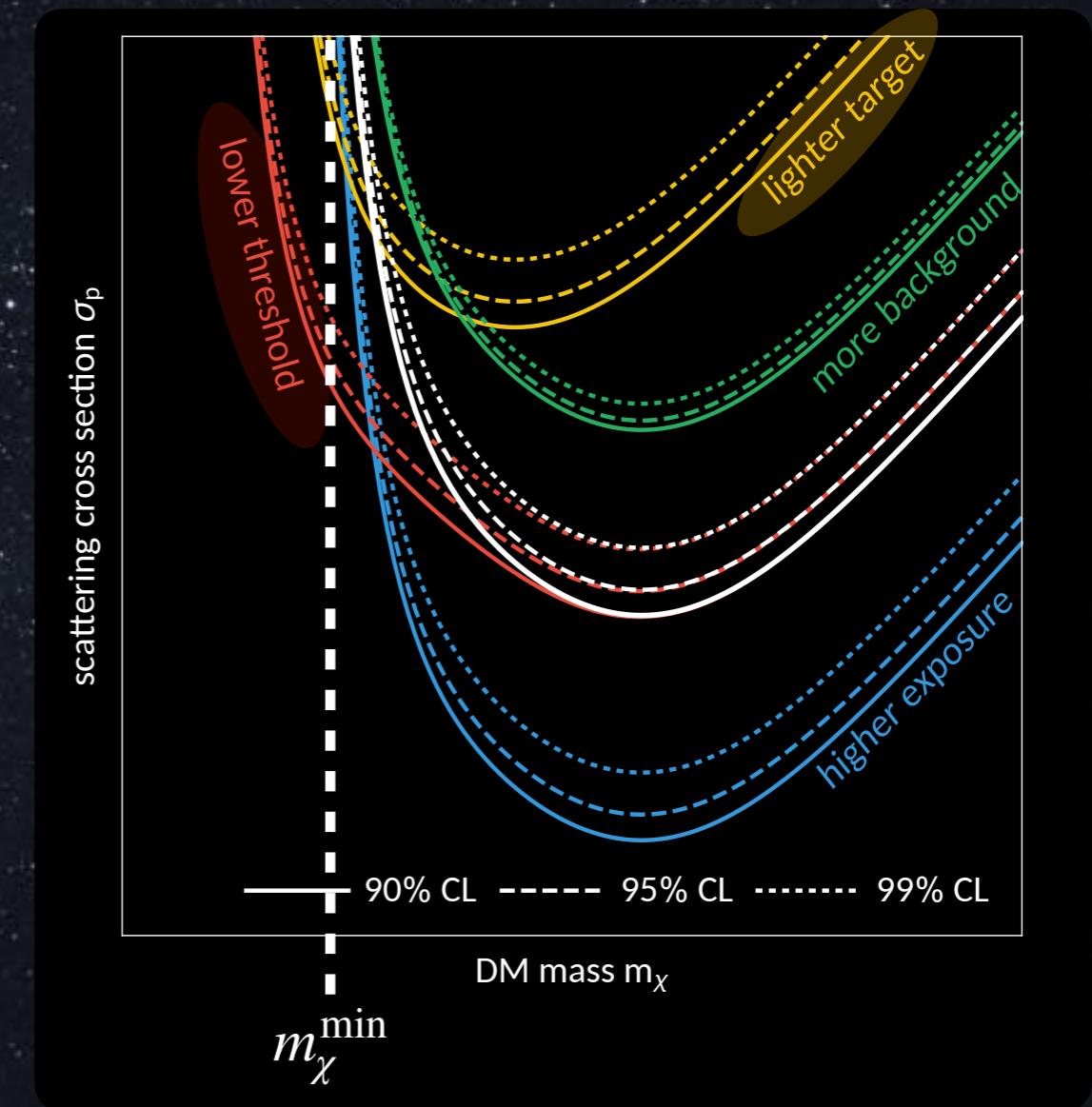
- Kinematic lower bound on the testable mass for a given recoil threshold.

$$m_\chi^{\min} = \frac{m_T}{\sqrt{\frac{2m_T}{E_{\text{thr}}} v_{\max} - 1}}$$

- Example: Threshold of XENON1T

$$E_{\text{thr}} \approx 5 \text{ keV} \Rightarrow m_\chi^{\min} \approx 7 \text{ GeV}$$

Aprile et al., PRL 119 (2017) 181301



Larger exposures do not probe lower masses.

# Direct Detection of Dark Matter via electron recoils

J. Kopp et al., Phys. Rev. D80 (2009) 083502

R. Essig et al., Phys. Rev. D85 (2012) 076007

Instead of nuclear recoils, search for DM-electron interactions.

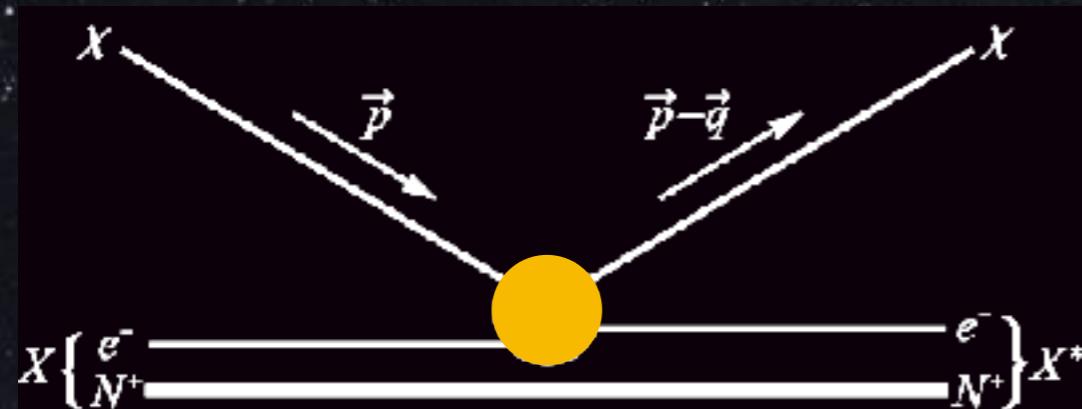


Figure from Essig et al., [arXiv:1509.01598]

- No kinematic penalty:  $E_e \leq E_\chi$ .
- Lowest DM mass to excite/ionize an electron in...
  - ...an isolated atom:
  - ...a semiconductor:

$$E_B \approx 10 \text{ eV}$$
$$\Rightarrow m_\chi^{\min} \approx 5 \text{ MeV}$$

$$E_{\text{gap}} \approx 1 \text{ eV}$$
$$\Rightarrow m_\chi^{\min} \approx 500 \text{ keV}$$

Lee et al., PRD 92 (2015) 083517

Essig et al., JHEP 1605 (2016) 046

# DM induced electron transitions & ionizations

- **Complication:** Target electrons are bound states, energy eigenstates, but not momentum eigenstates.

$$\frac{dR_{\text{ion}}}{dE_e} = \frac{1}{m_N} \frac{\rho_\chi}{m_\chi} \sum_{nl} \frac{\langle d\sigma_{\text{ion}}^{nl} v \rangle}{dE_e}$$

Ionization form factor

$$\frac{d\langle \sigma_{\text{ion}}^{nl} v \rangle}{dE_e} = \frac{\sigma_e}{8\mu_{\chi e}^2 E_e} \int dq q \left| F_{\text{DM}}(q) \right|^2 \left| f_{\text{ion}}^{nl}(k', q) \right|^2 \eta(v_{\min}(\Delta E_e, q))$$

- Depending on the target, this requires condensed matter theory.
- The ionization form factor only applies to a certain class of DM models.

Catena, R., TE, Spaldin N., Tarantino, W., PRR 2 (2020) 033195

Catena, R., TE, Matas, M., Spaldin, N., Urdshals, E., PRR 3 (2021) 033149

# Recent results on DM electron scattering experiments

- Semiconductor target experiments

## 1. SENSEI

Barak et al, PRL 125 (2020) 17, 171802

Abramoff et al., PRL 122 (2019) 161801

Crisler et al., PRL 121 (2018) 061803

## 2. SuperCDMS

Agnese et al., PRL 121 (2018) 051301

## 3. DAMIC at SNOLAB

Aguilar-Arevalo et al., [arXiv:1907.12628]

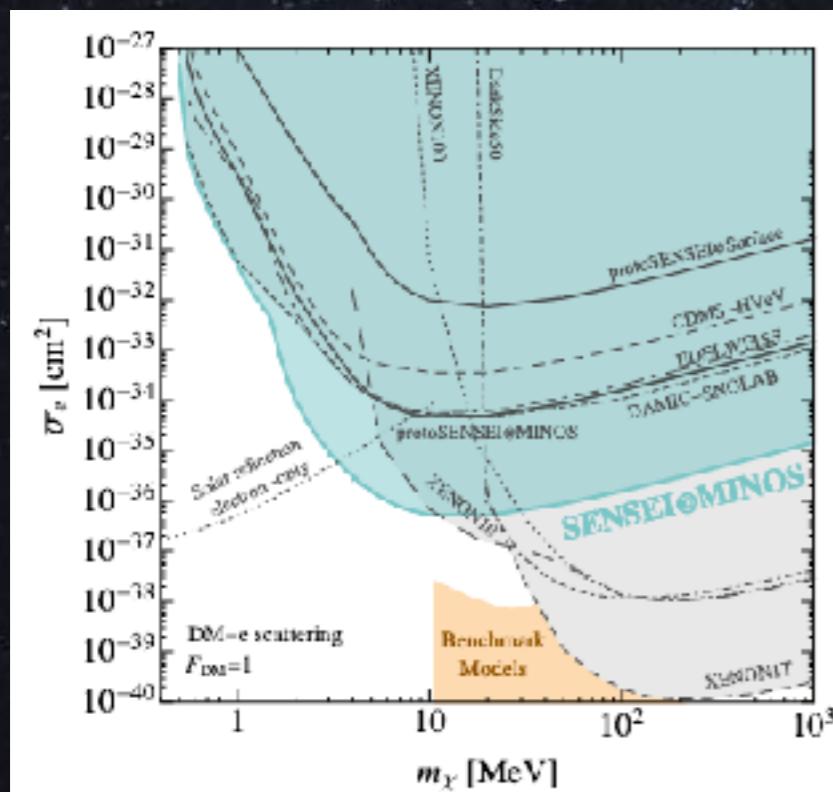


Figure from [2004.11378].

- Liquid noble gas targets

## 1. XENON10, XENON100, XENON1T

Essig et al., PRL 109 (2012), 021301

Essig et al., PRD 96 (2017), 043017

Aprile et al., PRL 123 (2019), 251801

## 2. DarkSide-50 (liquid argon)

Agnes et al., PRL 121 (2018) 111303

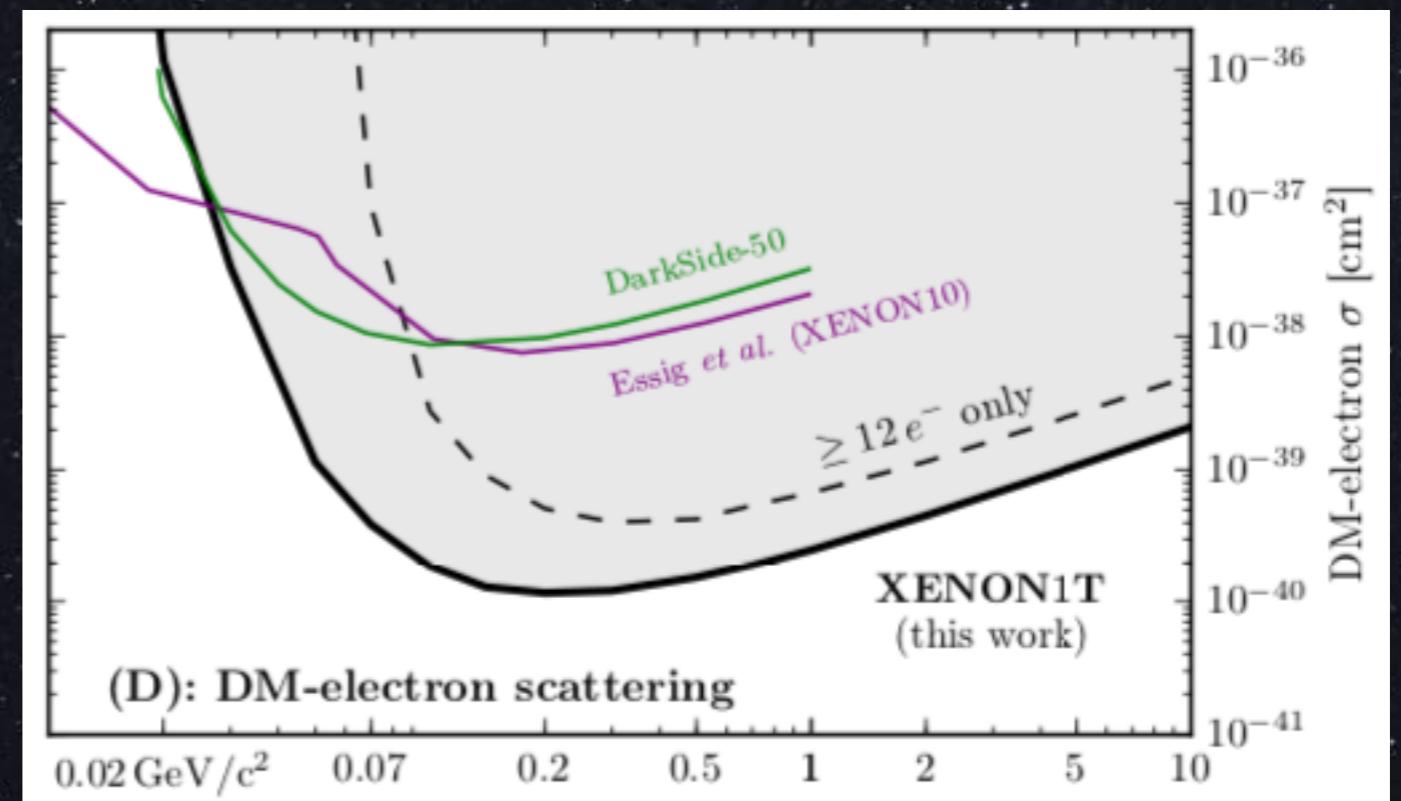


Figure from [1907.11485].

# An incomplete list of search strategies for sub-GeV DM

- Bremsstrahlung emission from the recoiling nucleus.

Kouvaris & Pradler, PRL 118 (2017) 031803

McCabe, PRD 96 (2017) 043010

- Migdal effect

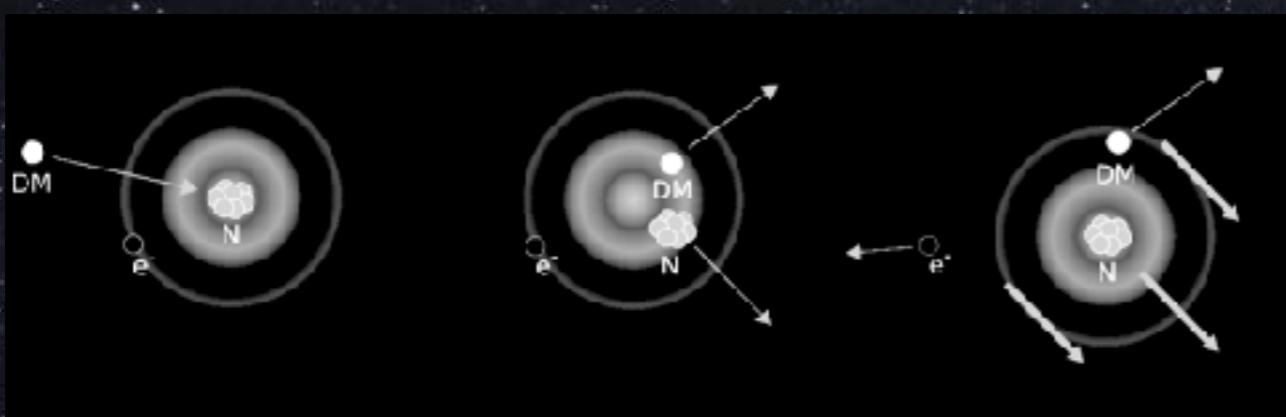


Figure from [1711.09906].

- Dirac materials (e.g. graphene) for directional detection.

Hochberg et al., Phys. Lett. B772 (2017) 239

Coskuner et al., PRD, [arXiv:1909.09170]

Geilhufe et al., PRD, [arXiv:1910.02091]

- Excitations of molecules in gases.

Essig et al., PRR, [arXiv:1907.07682]

- Up-scattering of DM by cosmic rays.

Bringmann & Pospelov, PRL 122 (2019) 171801

Ema et al., PRL 122 (2019) 181802

Alvey et al., PRL, [arXiv:1905.05776]

Cappiello & Beacom, PRD, [arXiv:1906.11283]

# Dark matter in the Sun



```
//2)*ax_y; //9. New velocity Eigen::Vector3d v2 = sqrt(GNewton*mSun/p)*(e+sin(theta2)*x2.normalized();
// double F2 = asinh((e+cos(theta2))/(1.0+ewcos(theta2))); // double M2 = ewsinh(F2)-F2; // double t2 = sq
stance at which we sample from the halo distribution double R=100.0*AU; if(R<100.0*rSun) { cerr <<"Warning in .
placeholderIC_1,vEarth); double u = Rejection_Sampling(prdr,e,B,(vEsc+vEarth),yMax,PRNG); //Velocity Directio
dem Point on a flat disk at distance R Eigen::Vector3d ez=-vIni.normalized(); Eigen::Vector3d ex{0,ez[2]
cos(phi)+ex+sin(phi)*ey}; // cout <<"Norm = "<<xIni.norm()/AU<<endl; // cout <<"Norm = "<<(R+ex).norm()/A
)[0]/rSun<<"\t"<<IC.Position()[1]/rSun<<"\t"<<IC.Position()[2]/rSun<<"\t"<<IC.Radius()/rSun<<"\t"<<IC.Velocity(
r_Shift(IC,rMax,model); } //2. orbit simulation //Right hand sides of the 1st order equations of motion. double
Euler_Cromer // void EC_Step(double &t,double &r,double &v,double &phi,double &J,double dt,SunModel &model) //{
// //RK coefficients // double k_r[4]; // double k_v[4]; // k_r[0]=dt*drdt(v); // k_v[0]
= dt*dvdt(r+k_r[1]/2.0,J,model); // k_p[2]= dt*dphidt(r+k_r[1]/2.0,J); // k_r[3]= dt*drdt(v+k_v[2])
+k_v[3]); // phi+= 1.0/6.0*(k_p[0]+2.0*k_p[1]+2.0*k_p[3]); // //ODE integration with Runge Kutta Fehl
k_v[0]= dt*dvdt(r,J,model); k_p[0]= dt*dphidt(r,J); k_r[1]= dt*drdt(v+k_v[0]/4.0); k_v[1]= dt*dvdt(r+k_r[0])
+0*k_r[1],J); k_r[3]= dt*drdt(v+1932.0/2197.0*k_v[0]-7200.0/2197.0*k_v[1]+7296.0/2197.0*k_v[2]); k_v[3]= dt*dr
+0*k_v[1]+3680.0/513.0*k_v[2]-845.0/4184.0*k_v[3]); k_v[4]= dt*dvdt(r+439.0/216.0*k_r[0]-8.0*k_r[1]
-5544.0/2505.0*k_v[2]+1859.0/4184.0*k_v[3]-11.0/40.0*k_v[4]); k_v[5]= dt*dvdt(r-8.0/27.0*k_r[0]+2.0*k_r[1]-384
ble r4=r+28.0/216.0*k_r[0]+1408.0/2868.0*k_r[2]+2101.0*k_r[3]-1.0/50.0*k_r[4]+2.0/55.0*k_r[5]; double v3M v+16.0/135.0*k_v[0]+6656.0/12825.0*k
r4),abs(v3-v4),abs(phi3-phi4)); std::vector<double> tolerance = {1.0*xM,1e-3*xM/sec,1e-6}; //New stepsize
},std::end(deltas)); //Next steps if(err[0]<tolerance[0]&&err[1]<tolerance[1]&&err[2]<tolerance[2]) { if(r<
x1 = speed*dt/mfp; } t+=dt; r= r4; v= v4; phi= phi4; dt= std::max(dtMin, std::min(delta*dt,dtMax)); } else { dt
Eigen::Vector3d& axes) { double w_phi = J/sqr(r2); Eigen::Vector3d xNew = r*(cos(phi)*axes[0]+sin(phi)*axes[1]
scattering or (b) leaving the sun. bool Propagate(Event& xe,DN_Particle& DN,SunModel &model, std::in
)).normalized(); Eigen::Vector3d ax_y = ax_z.cross(ax_x); std::vector<Eigen::Vector3d> axes = {ax_x,ax_y
}).dot(ax_z); //Sample -log(1-xi) double logXi = -1.0*log(1.0-ProbabilitySample(PRNG)); //Start integr
_step(t,r,v_r,phi,J,dt,logXi, DN,model); if(xM10==0) { Event xNew = PlaneToSP(t,r,phi,v_r,J,axes); t <=xNew.Position
ment // t <<xNew.Position()[0]/rSun<<"\t"<<xNew.Position()[1]/rSun<<"\t"<<xNew.Position()[2]/rSun<<"\t" <</sec
Old<rMax&mr>rMax(x_r*v_r+J*v_r)/r/r>model.vEsc2(r)) { propagate=false; xe=PlaneToSP(t,r,phi,v_r,J,axes); } else
(Event &x0,DN_Particle& DN,SunModel &model, std::mt19937& PRNG) // double speed=x0.Speed(); } // cout <<"Scat
ordinates(1.0,Thetasample(PRNG),Phisample(PRNG)); double vRel = (vTarget-x0.Velocity()).norm(); Eigen
eed0<<endl; // if(speed>sqr(model.vEsc2(r))&&vnew.norm()<sqr(model.vEsc2(r))) cout <<"tcapture"<<endl; //
ol&model, unsigned int &nScattering, std::mt19937& PRNG) { //Save trajectory of stream
x.Speed()-model.vEsc2(x.Radius()))/E0<<endl; //Output // Event x = IC; bool success; //Counters nScat
if(r<rSun) { if(nScattering>nScattering_max) { success = false; break; } // double E1=DN.mass/2.0*(x
*(x.Speed()+x.speed()-model.vEsc2(x.Radius()))/E0<<endl; // if(E1>0 && E2<0) { // cout <<"capture"<<endl;
<"\t" <<r/rSun << endl; // t .close(); } } // if (x .Speed () < sqrt ( model .V
olution[0]/rSun<<"\t"<<xFinal.Position()[1]/rSun<<"\t"<<xFinal.Position()[2]/rSun<<"\t"<<xFinal.Radiu
cess=false; if(x.Time()/sec<9999) return false; return success; } Result generate_Data(Conf
t.nmpnrcs-1)? 0 : config.rank+1; int tag = 0; MPI_Status status; MPI_Request req; iquest; // /Datapoi
data; data.resize(config.nRings); //Counters //Total number of simulated particles unsigned long int &count
Eigen::vector<unsigned long int> L_Counter_Data_new(config.nRings,0); //Number of passes // std::vector<unsig
ber of scatterings; std::vector<double> G_nScattering_Av(config.nRings,0.0); std::vector<double> L_nScat
Isend(&G_Counter_Data.Front(),config.nRings,MPI_UNSIGNED_LONG,destination,tag,MPI_COMM_WORLD
G); unsigned int nScattering=0; bool success = Simulate_Trajectory(x, DN, model, nScattering, PRNG)
_nScattering_Av(IsoRing)+nScattering)/L_Counter_Data[IsoRing]; double speed=x.Speed(); fns
DN_WORLD,&flag,&status); if(flag) { //Receive the tokens MPI_Recv(&Counter_Data.front(),config.nR
r_data[i]+=L_Counter_Data_new[i]; L_Counter_Data_new[i]=0; } // cout <<config.rank <<"\t" <<G_Cou
else if (nmini>config.nData) tag = status.MPI_TAG; //Pass on the tokens unless you are the very la
gth cout <<"\r"; for(int i=0;i<2.0*BarLength;i++) cout <<" "; cout <<"\r"; for(int i=0;i<BarLength-
{ cout <<"\r"; for(int j=0;j<10*BarLength;j++) cout <<" "; cout <<"\r"; } // for(int i=
_Counter_Free,1,MPI_UNSIGNED_LONG_LONG,MPI_SUM,MPI_COMM_WORLD); MPI_Allreduce(&L_nScat
> Global_Data; for(int i = 0;i<config.nRings;i++) { unsigned long int Global_SampleSize; MPI
cv_displs[config.nmpnrcs]; MPI_Allgather(&L_Counter_Data[i],MPI_UNSIGNED_LONG, &nData_loc
SampleSize); // MPI_Gatherv(&data[i].front(),data.size(),MPI_Datapoint,&Ring_Data.front()
MPI_BARRIER(MPI_COMM_WORLD); // for(int i=0;i<config.nRings;i++) // { // if(config.r
,i,MPI_DOUBLE,MPI_MAX,MPI_COMM_WORLD); return Result(Global_Data, DN, &Counter_Simulat
std::vector<double> nScatterings_mean=nScav, double dt) { data=dat; particle = p; // sample_size
} nScatterings_mean=nScav; nScatterings_mean_tot=0.0; for(unsigned int i=0;i<
e & config) { if(config.rank==0) { std::cout <<"Simulation summe
<Round(1.0*nSimulations/duration)<<"/sec)"<<std::endl; // (config.nRings==1) st
<< " , " <<Round(100.0*nFails/nSimulations)<<")" <<std::endl; } } } } #inc
functions.hpp" using namespace libconfig; //1. String with input param
mbar=mD; //Parallelisation rank = myrank; numprocs=sp; // Configuration
const FileIOException Affex) { std::cerr <<"I/O error while re
try { simTB = cfg.lookup("simTB").c_str(); l.readh(const
file." << endl; exit(EXIT_FAILURE); } //2. Event >
"None." << endl<<endl; } //2. Event >
"None." << endl<<endl; }
```

# Solar Reflection of DM (SRDM)

H. An, M. Pospelov, J. Pradler, A. Ritz, PRL 120 (2018), 141801

TE, C. Kouvaris, N. G. Nielsen, PRD, 97 (2018), 063007

- Pro memoriam:

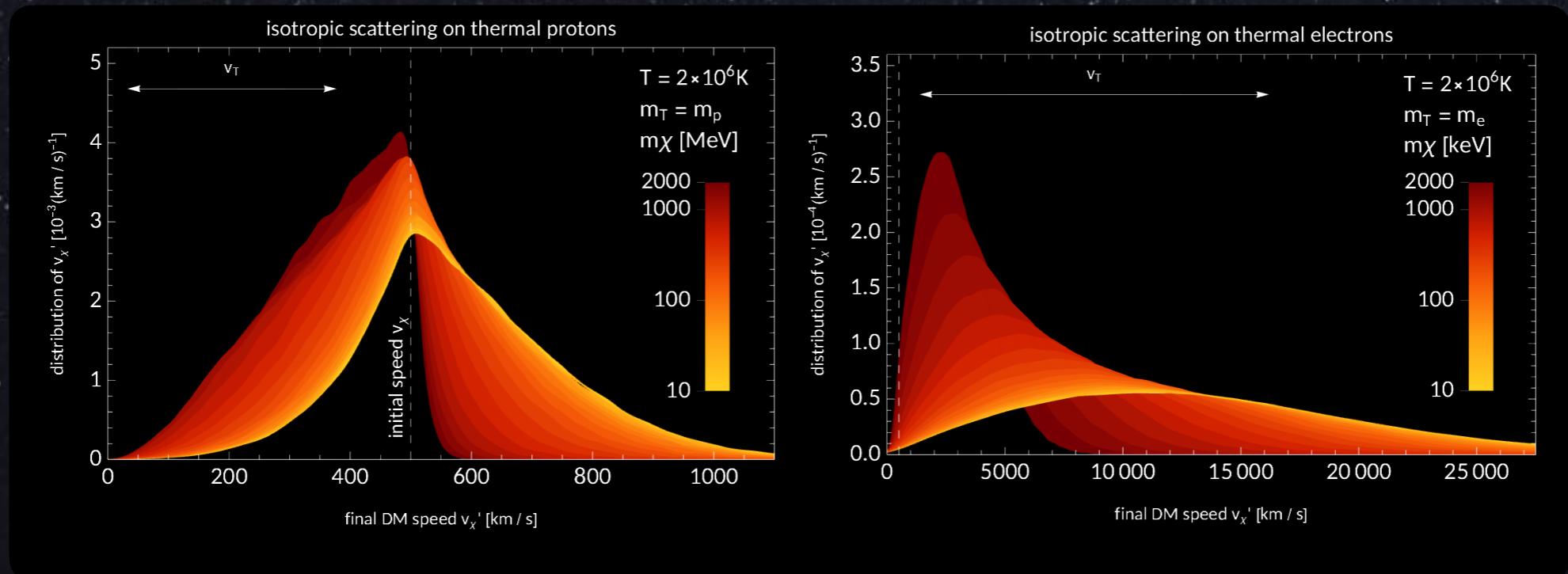
$$m_\chi \gtrsim \frac{m_T}{\sqrt{\frac{2m_T}{E_{\text{thr}}}v_{\max} - 1}}$$

$$m_\chi \gtrsim \frac{2E_{\text{gap}}}{v_{\max}^2}$$

- What if some process could increase the maximum DM speed somehow?

$$v_{\max} > v_{\text{esc}}^{(\text{gal})} + v_{\oplus}$$

- Elastic scatterings on thermal targets inside the Sun could do just that.



$t = 0.$  hr

$N_s = 0$

$v_x = 368 \frac{\text{km}}{\text{s}}$

$E_x/E_0 = 1.$



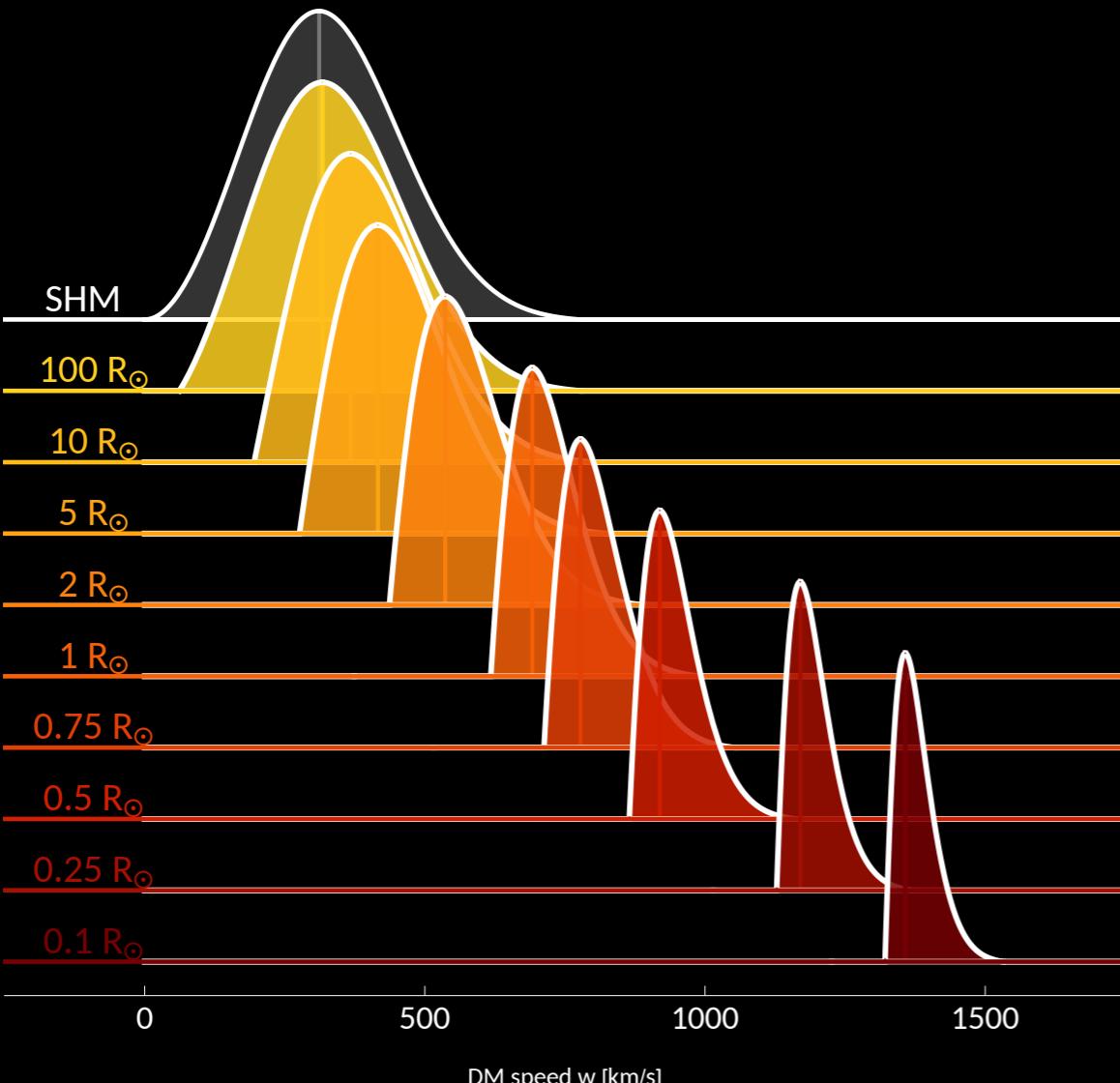
$m_\chi = 100 \text{ MeV}$

$\sigma_p = 0.2 \text{ pb}$

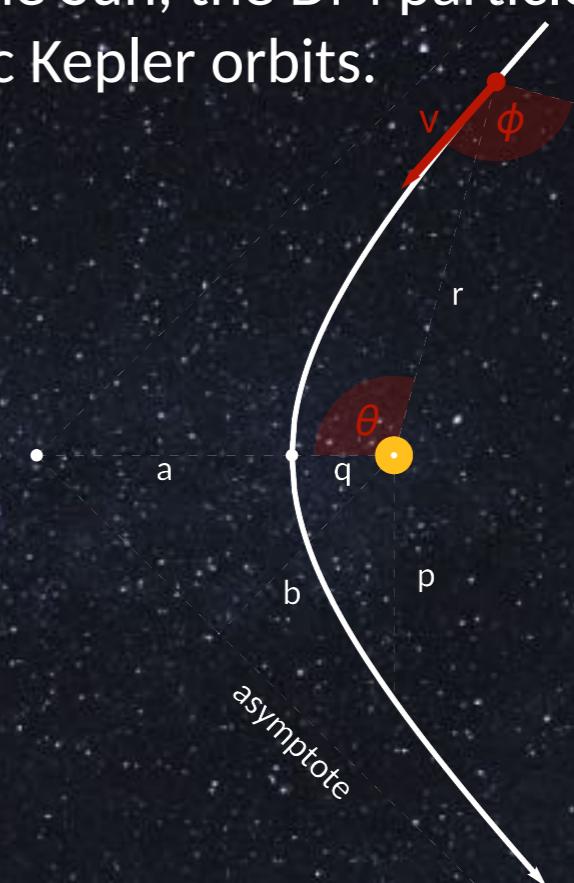
Emken 2019

# DM particles passing through the Sun

- Gravitational acceleration



- Outside the Sun, the DM particles follow hyperbolic Kepler orbits.



- The rate of DM particles entering the Sun is given by

$$\Gamma(m_\chi) = n_\chi \pi R_\odot^2 \int du f_{\text{halo}}(u) \left( u + \frac{v_{\text{esc}}(R_\odot)^2}{u} \right)$$

$$\approx 1.1 \cdot 10^{33} \left( \frac{m_\chi}{\text{MeV}} \right)^{-1} \text{s}^{-1}$$

$$w(u, r) = \sqrt{u^2 + v_{\text{esc}}(r)^2}$$

$$v_{\text{esc}}(r) = \sqrt{\frac{2G_N M_\odot}{r}}$$

# DM interaction models with heavy mediators

- We focused on the 4 standard scenarios in this study.

## 1. Spin-Independent nuclear interactions (SI)

$$\frac{d\sigma_N^{\text{SI}}}{dE_R} = \frac{m_N \sigma_p^{\text{SI}}}{2\mu_{\chi p}^2 v_\chi^2} \left[ Z + \frac{f_n}{f_p} (A - Z) \right]^2 \left| F_N(q) \right|^2$$

## 2. Spin-Dependent nuclear interactions (SD)

$$\frac{d\sigma_N^{\text{SD}}}{dE_R} = \frac{2m_N \sigma_p^{\text{SD}}}{3\mu_{\chi p}^2 v_\chi^2} \frac{J+1}{J} \left[ \langle S_p \rangle + \frac{f_n}{f_p} \langle S_n \rangle \right]^2 \left| F_N^{\text{SD}}(q) \right|^2$$

## 3. Electron interactions only.

$$\frac{d\sigma_e}{dq^2} = \frac{\bar{\sigma}_e}{4\mu_{\chi e}^2 v_\chi^2} F_{\text{DM}}(q)^2$$

## 4. Dark photon model

A small dark sector with an addition U(1) gauge group (broken) and kinetic mixing term.

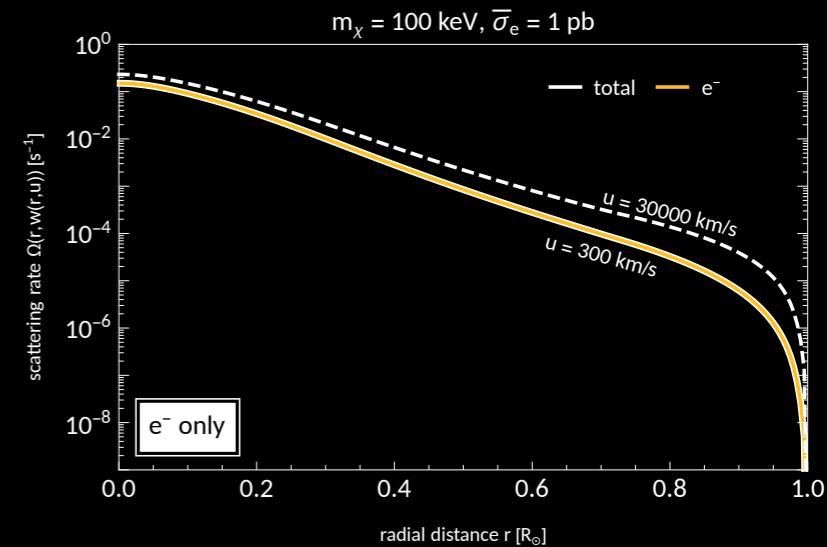
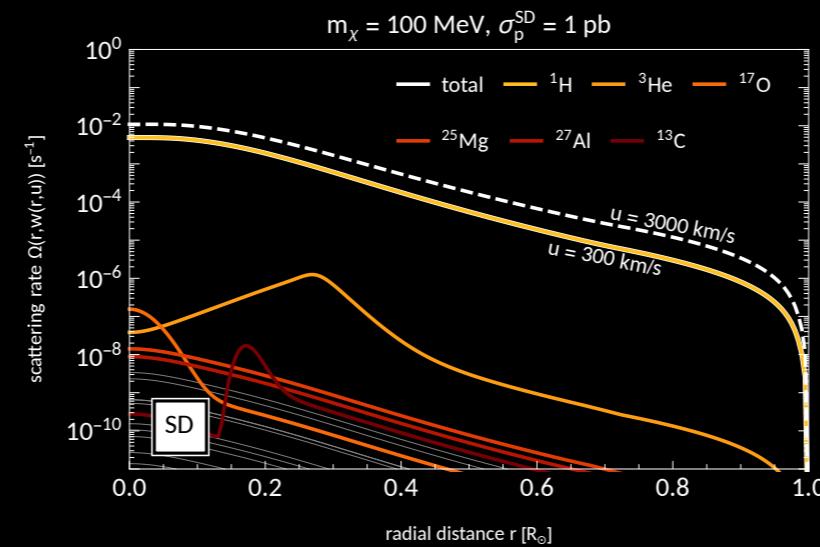
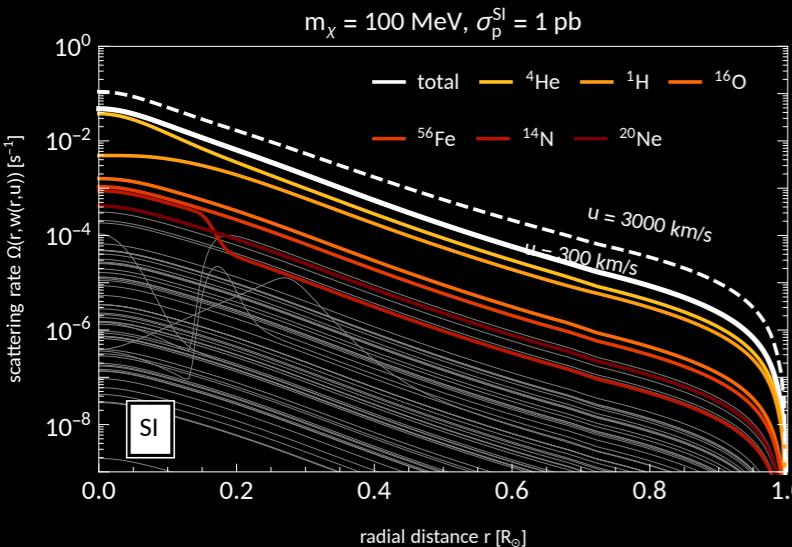
$$\mathcal{L}_D = \bar{\chi}(i\gamma^\mu D_\mu - m_\chi)\chi + \frac{1}{4}F'_{\mu\nu}F'^{\mu\nu} + m_{A'}^2 A'_\mu A'^\mu + \frac{\epsilon}{2}F_{\mu\nu}F'^{\mu\nu}$$

$$\frac{\bar{\sigma}_p}{\bar{\sigma}_e} = \left( \frac{\mu_{\chi p}}{\mu_{\chi e}} \right)^2 \quad \Rightarrow \text{Hierarchy of cross sections}$$

# Underground DM scattering rates

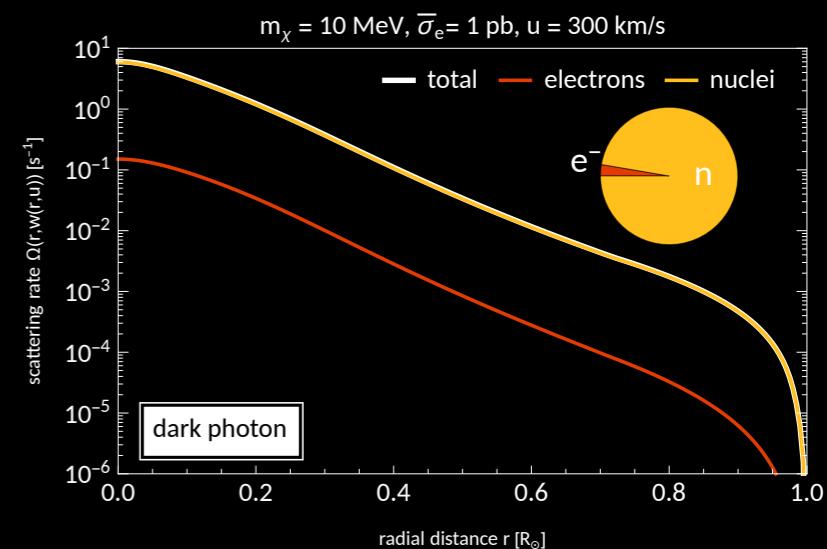
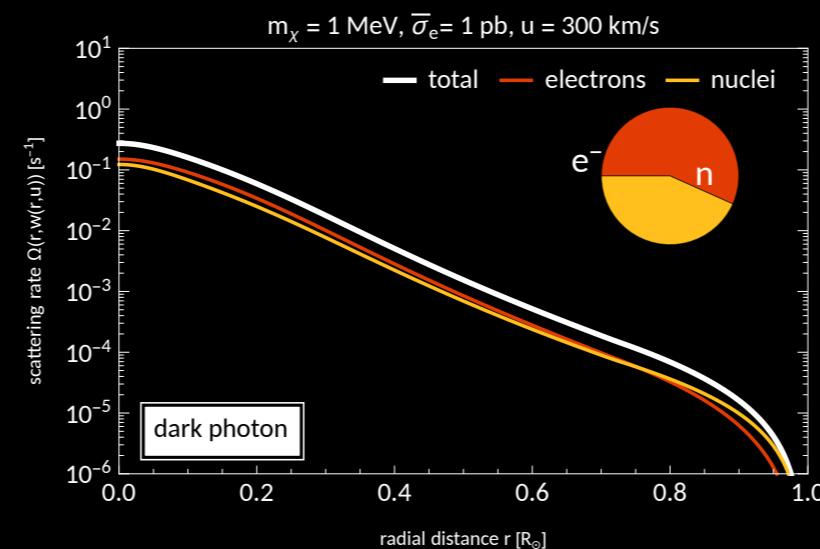
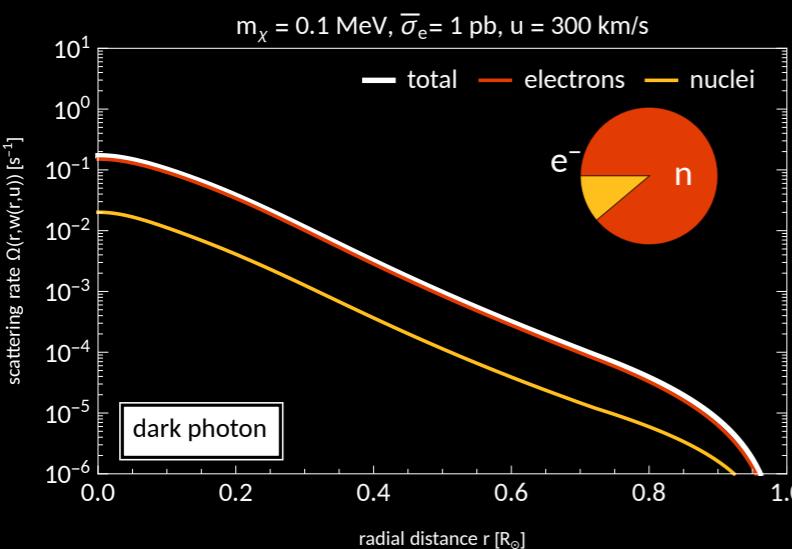
$$\Omega(r, \mathbf{v}_\chi) = \sum_i n_i(r) \langle \sigma_i | \mathbf{v}_\chi - \mathbf{v}_{T,i} | \rangle$$

Number density  
Total cross section  
Relative speed  
 $\langle \cdot \rangle$  : Thermal average



- Dark photon model:

$$\frac{\Omega_e}{\Omega_p} \propto \frac{\bar{\sigma}_e}{\bar{\sigma}_p} \frac{\langle |\vec{v}_\chi - \vec{v}_e| \rangle}{\langle |\vec{v}_\chi - \vec{v}_p| \rangle} = \left( \frac{\mu_{\chi p}}{\mu_{\chi e}} \right)^2 \times \mathcal{O}(10)$$



# First estimate of the SRDM flux

- We can make a first estimate of the total solar reflection rate.

$$\mathcal{R}_\odot(m_\chi) = p_{\text{refl}} \times \Gamma(m_\chi)$$

- Here we assume an average probability for a DM particle to get reflected,  $p_{\text{refl}}$ .
- The total flux is therefore

$$\begin{aligned}\Phi_\odot(m_\chi) &= \frac{\mathcal{R}_\odot}{4\pi\ell^2} \\ &\approx 3.8 \cdot 10^5 p_{\text{refl}} \left( \frac{m_\chi}{\text{MeV}} \right)^{-1} \text{s}^{-1}\text{cm}^{-2}\end{aligned}$$

- Compare this to the halo DM flux

$$\Phi_{\text{halo}}(m_\chi) \approx 1.3 \cdot 10^{10} \left( \frac{m_\chi}{\text{MeV}} \right)^{-1} \text{s}^{-1}\text{cm}^{-2}$$

# Direct detection of SRDM

- The nuclear recoil spectrum caused by a generic DM flux is given by

$$\frac{dR}{dE_R} = N_T \int_{v_\chi > v_{\min}} dv_\chi \frac{d\Phi_\chi}{dv_\chi} \frac{d\sigma_{\chi N}}{dE_R}$$

- For halo DM, we usually directly insert

$$\frac{d\Phi_{\text{halo}}}{dv_\chi} = \frac{\rho_\chi}{m_\chi} v_\chi f_\chi(v_\chi)$$

- The SRDM flux can simply be added,

$$\frac{dR}{dE_R} = N_T \int dv_\chi \left( \frac{d\Phi_{\text{halo}}}{dv_\chi} + \frac{d\Phi_{\odot}}{dv_\chi} \right) \frac{d\sigma_{\chi N}}{dE_R}$$

Halo DM  
SRDM

- But one of the two fluxes will always dominate the other one.

# Distinguishing SRDM from background or halo DM

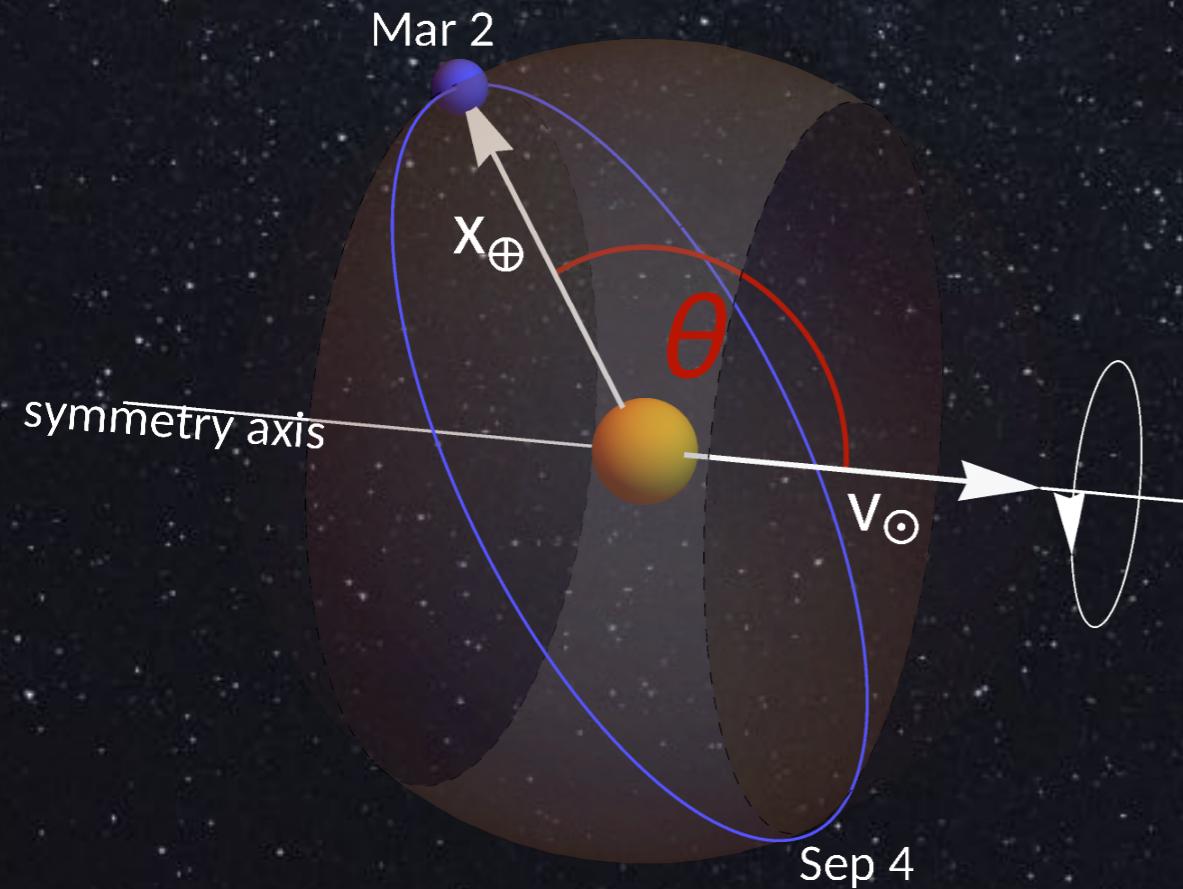
Signatures of a SRDM signal:

1. Directionality: If the detector is sensitive to the DM particle's incoming direction in the Sky, the Sun could be identified as the signal's origin.
2. Annual modulation: There are two source of annual modulation:
  - Orbital modulation: The flux scales as  $\ell^{-2}$ . Due to the Earth's orbital eccentricity this causes a small annual modulation.
  - Anisotropy modulation: If the Sun ejects the SRDM flux anisotropically, this could be a 2nd source of annual modulations.
3. Diurnal modulations: Underground scatterings inside the Earth would typically cause a day-night modulation.

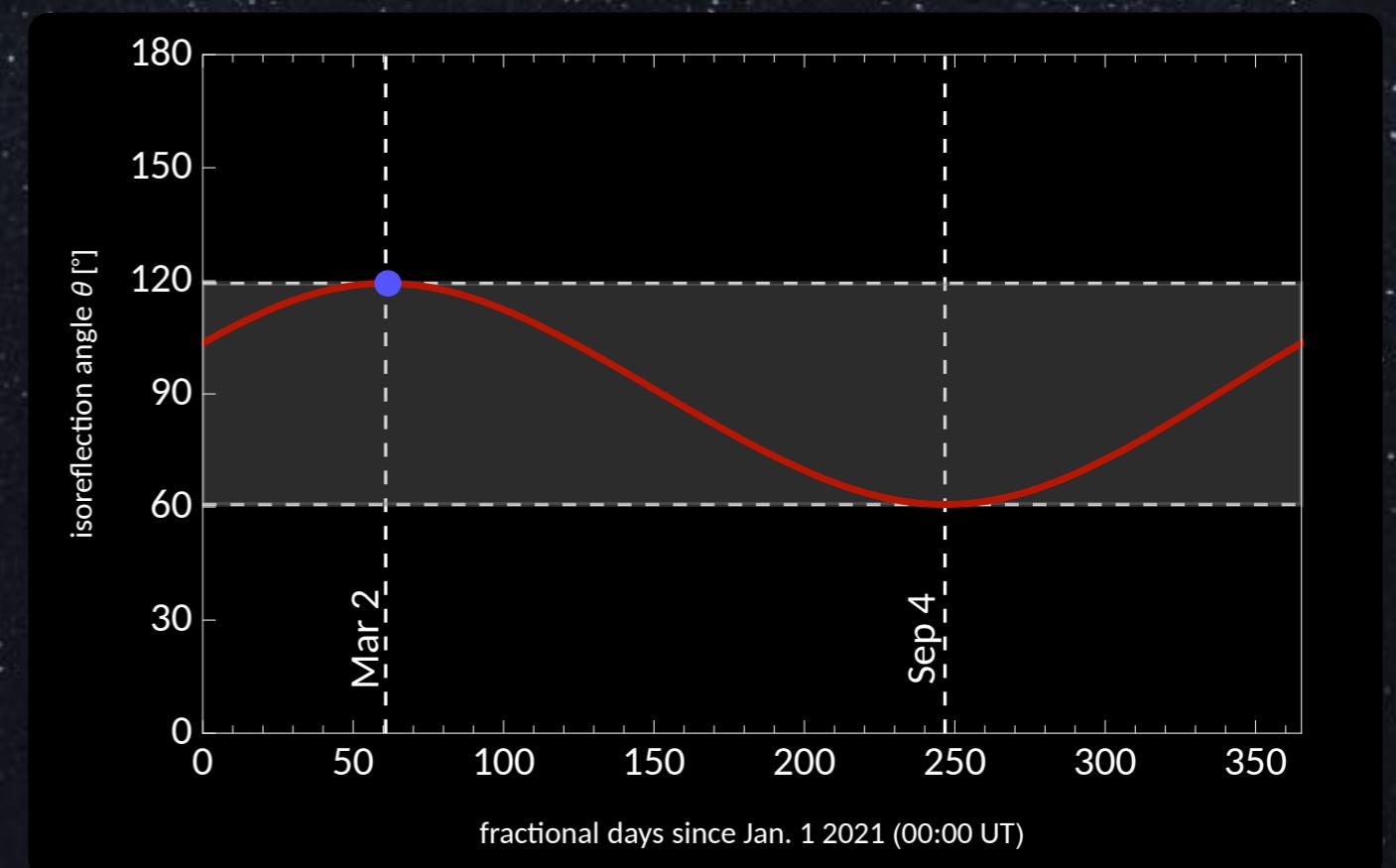
# The isoreflection angle

- The boost into the Sun's rest frame gives rise to the DM wind.
- The system is symmetric under rotations around the Sun's velocity.
- We define the polar angle of this symmetry axis as the **isoreflection angle**:

$$\theta = \angle(\mathbf{x}_\oplus, \mathbf{v}_\odot)$$



- Time evolution of the Earth's isoreflection angle



# III. Monte Carlo simulation of solar reflection

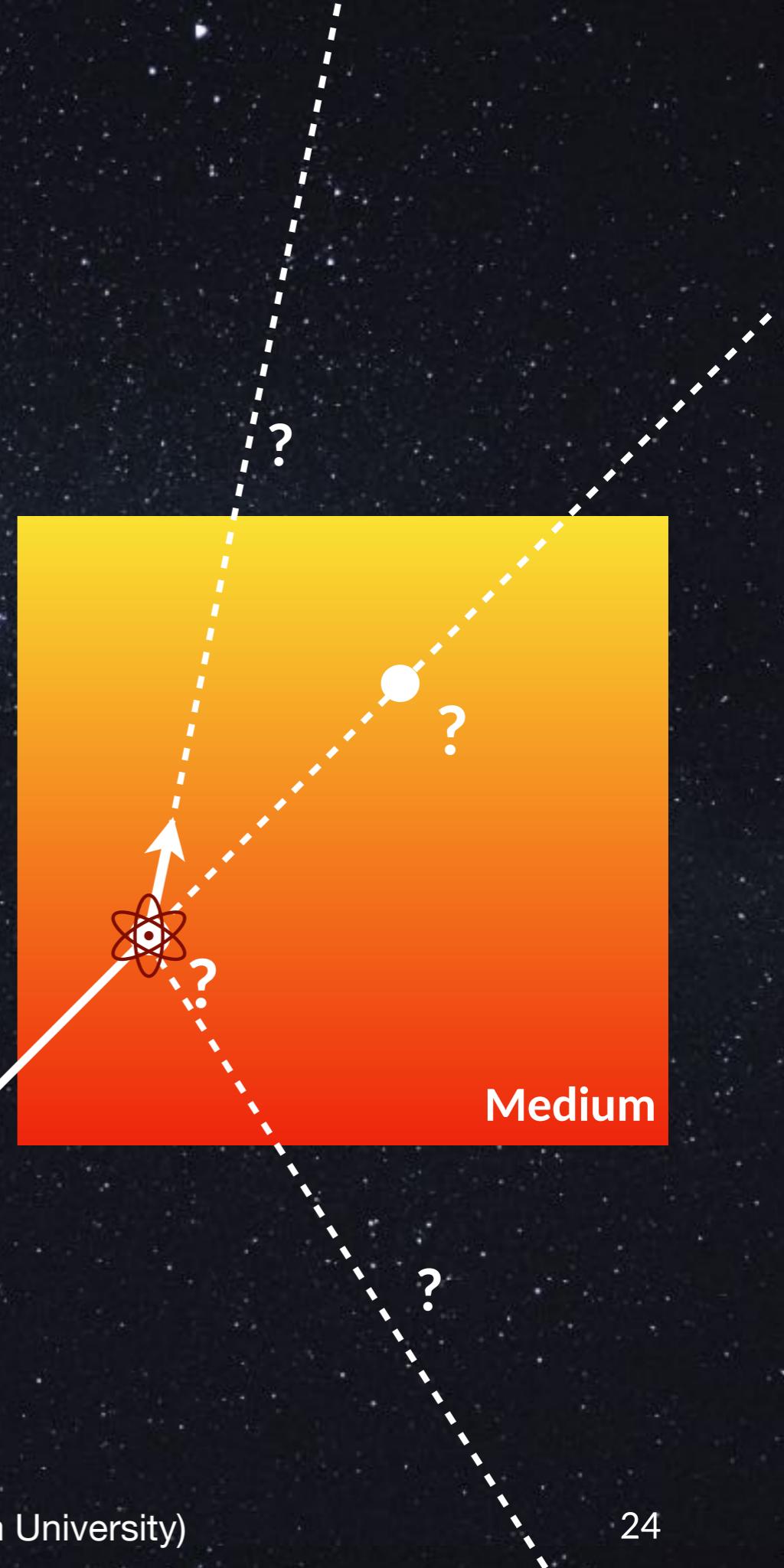
```
2)*ax_y; //9. New velocity Eigen::Vector3d v2 = sqrt(GNewton*mSun/p)*(e+sin(theta2))*x2.normalized();
// double F2 = asinh((e+cos(theta2))/(1.0+ew*cos(theta2))); // double M2 = ew*inh(F2)-F2; // double t2 = sq
stance at which we sample from the halo distribution double R=100.0*AU; if(R<100.0*rSun) { cerr <<"Warning in .
placeholderIC_1,vEarth); double u = Rejection_Sampling(prdr,e,B,(vosc+vEarth),ymax,PRNG); //Velocity Directio
dem Point on a flat disk at distance R Eigen::Vector3d ez=-vIni.normalized(); Eigen::Vector3d ex{e,ez*2
cos(phi)+ex+sin(phi)*ey}; // cout <<"Norm = "<<xIni.norm() /AU<<endl; // cout <<"Norm = "<<(R+ex).norm() /A
[0]/rSun<<"\t"<<IC.Position()[1]/rSun<<"\t"<<IC.Position()[2]/rSun<<"\t"<<IC.Radius() /rSun<<"\t"<<IC.Velocity(
r_Shift(IC,rMax,model); } //2. orbit simulation //Right hand sides of the 1st order equations of motion. double
Euler_Cromer // void EC_Step(double &t,double &r,double &v,double &phi,double &J,double dt,SunModel &model) //{
// //RK coefficients // double k_r[4]; // double k_v[4]; // double k_p[4]; // k_r[0]=dt*drdt(v); // k_v[0]
= dt*dvdt(r+k_r[1]/2.0,J,model); // k_p[2]= dt*dphidt(r+k_r[1]/2.0,J); // k_r[3]= dt*drdt(v+k_v[2])
k_v[1]= dt*dvdt(r,J,model); k_p[1]= dt*dphidt(r,J); k_r[1]= dt*drdt(v+k_v[0]/4.0); k_v[1]= dt*dvdt(r+k_r[0])
+ k_r[1],J); k_r[3]= dt*drdt(v+1932.0/2197.0*k_v[0]-7200.0/2197.0*k_v[1]+7296.0/2197.0*k_v[2]); k_v[3]= dt*dr
+ k_v[1]+3680.0/513.0*k_v[2]-845.0/4184.0*k_v[3]); k_v[4]= dt*dvdt(r+439.0/216.0*k_r[0]-8.0*k_r[1]
-5544.0/2505.0*k_v[2]+1859.0/4184.0*k_v[3]-11.0/40.0*k_v[4]); k_v[5]= dt*dvdt(r-8.0/27.0*k_r[0]+8.0*k_r[1]-384
double r4=r+28.0/216.0*k_r[0]+1408.0/2868.0*k_r[2]+2107.0/4101.0*k_r[3]-1.0/8.0*k_r[4]; double v4=v+28.0/216.0
k_r[2]+28561.0/56430.0*k_r[3]-9.0/50.0*k_r[4]+2.0/55.0*k_r[5]; double v5=v+16.0/135.0*k_v[0]+6656.0/12825.0*k
r4),abs(v5-v4),abs(phi5-phi4)); std::vector<double> tolerance = {1.0*xm,1e-3*xm/sec,1e-6}; //New stepsize
},std::end(deltas)); //Next steps if(err[0]<tolerance[0]&&err[1]<tolerance[1]&&err[2]<tolerance[2]) { if(r<
x1 = speed*dt/mfp; } t+=dt; r= r4; v= v4; phi= phi4; dt= std::min(dtMin, std::min(delta*dt,dtMax)); } else { dt
vector<Eigen::Vector3d> axes; { double w_phi = J/sin(r2); Eigen::Vector3d xNew = r*(cos(phi)*axes[0]+sin(phi)*w
scattering or (b) leaving the sun. bool Propagate(Event& xe,DH_Particle& DH_SunModel& model, std::min
).normalized(); Eigen::Vector3d ax_y = ax_z.cross(ax_x); std::vector<Eigen::Vector3d> axes = {ax_x,ax_y
}).dot(ax_z); //Sample -log(1-xi) double logXi = -1.0*log(1.0-ProbabilitySample(PRNG)); //Start integr
_step(t,r,v_r,phi,J,dt,logxi,DH_model); if(xNew==0) { Event xNew=PlaneToSP(t,r,phi,v_r,J,axes); t <=xNew.Position
num // T <<xNew.Position()[0]/rSun<<"\t"<<xNew.Position()[1]/rSun<<"\t"<<xNew.Position()[2]/rSun<<"\t"
<</sec old<rMax&mr>rMax(x_r*v_r+J*v_r)/r/r>model.vEsc2(r)) { propagate=false; xe=PlaneToSP(t,r,phi,v_r,J,axes); } else
(Event& xe,DH_Particle& DH_SunModel& model, std::min t19937& PRNG) [ // double speed=x.Speed(); } cout <<"Scat
ordinates(1.0,Thetasample(PRNG),Phisample(PRNG)); double VRel = (vTarget-xe.Velocity()).norm(); Eig
eede<endl; // if(speed>sqrt(model.vEsc2(r))&&vnew.norm()<sqrt(model.vEsc2(r))) cout <<"tcapture"<<endl; //
ol&model, unsigned int &nScattering, std::min t19937& PRNG) { // Save trajectory of stream
x.Speed()*x.Speed()-model.vEsc2(x.Radius()))/E0<<endl; //Output // Event x = IC; bool success; //Counters nScat
if(r<rSun) { if(nScattering>nScattering_max) { success = false; break; } // double E1=DH.mass/2.0*(x
*(x.Speed()+x.Speed()-model.vEsc2(x.Radius()))/E0<<endl; // if(E1>0 && E2<0) { // cout <<"capture"<<endl;
<"\t" << r / rSun << endl; // T .close(); } } // if (x .Speed () < sqrt ( model .V
olution()[0]/rSun<<"\t"<<xFinal.Position()[1]/rSun<<"\t"<<xFinal.Position()[2]/rSun<<"\t"<<xFinal.Radiu
cess=false; if(x.Time()/sec<9999) return false; return success; } Result generate_Data(Confir
t,numprocs-1) ? 0 : config.rank+1; int tag = 0; MPI_Status status; MPI_Request req; iquest; // /Datapoi
data; data.resize(config.nRings); //Counters //Total number of simulated particles unsigned long int &coun
lvector<unsigned long int> L_Counter_Data_new(config.nRings,0); //Number of passes // std::vector<unsigne
ber of scatterings; std::vector<double> G_nScattering_Av(config.nRings,0.0); std::vector<double> L_nScat
Isend(&G_Counter_Data.Front(),config.nRings,MPI_UNSIGNED_LONG,destination,tag,MPI_COMM_WORLD
G); unsigned int nScattering=0; bool success = Simulate_Trajectory(x,DH,model,nScattering,PRNG)
_nScattering_Av(IsoRing)+nScattering)/L_Counter_Data[IsoRing]; double speed=x.Speed(); fns
DH_WORLD,&flag,&status); if(flag) { //Receive the tokens MPI_Recv(&Counter_Data.Front(),config.nRi
r_data[i]+=L_Counter_Data_new[i]; L_Counter_Data_new[i]=0; } // cout <<config.rank <<"\t" <<G_Cou
else if (nmini>config.nData) tag = status.MPI_TAG; //Pass on the tokens unless you are the very la
gth cout <<"\r"; for(int i=0;i<2.0*BarLength;i++) cout <<" "; cout <<"\r"; for(int i=0;i<BarLength
{ cout <<"\r"; for(int j=0;j<10*BarLength;j++) cout <<" "; cout <<"\r"; } // for(int i=
_Counter_Free,1,MPI_UNSIGNED_LONG_LONG,MPI_SUM,MPI_COMM_WORLD); MPI_Allreduce(&L_nScat
> Global_Data; for(int i = 0;i<config.nRings;i++) { unsigned long int Global_SampleSize; MPI
cv_displs[config.numprocs]; MPI_Allgather(&L_Counter_Data[i],MPI_UNSIGNED_LONG,Global_Data_loc
SampleSize); // MPI_Gatherv(&data[i].front(),data.size(),MPI_Datapoint,&Ring_Data.front()
MPI_BARRIER(MPI_COMM_WORLD); // for(int i=0;i<config.nRings;i++) // { // if(config.r
,i,MPI_DOUBLE,MPI_MAX,MPI_COMM_WORLD); return Result(Global_Data,DH,G_Counter_Simulat
std::vector<double> nScatterings_mean=nscav, nscatterings_mean_tot=0.0; for(unsigned int i=0;i<
e & config) { if(config.rank==0) { std::cout <<"Simulation summe
<Round(1.0*nSimulations/duration)<<"/sec)"<<std::endl; // (config.nRings==1) st
<" , " <<Round(100.0*nFails/nSimulations)<<")" <<std::endl; } } } } #inc
functions.hpp" using namespace libconfig; //1. String with input param
mbava=mD; //Parallelisation rank = myrank; numprocs= np; Configuration
const FileIOException Affex) { std::cerr <<"I/O error while re
try { SimTB = cfg.lookup("simTB").c_str(); l.readh(const
file." << endl; exit(EXIT_FAILURE); } //2. Event >
"None."<<endl<<endl; } //2. Event >
"None."<<endl<<endl;
```

# MC Simulations of DM in the Sun

The main random processes are

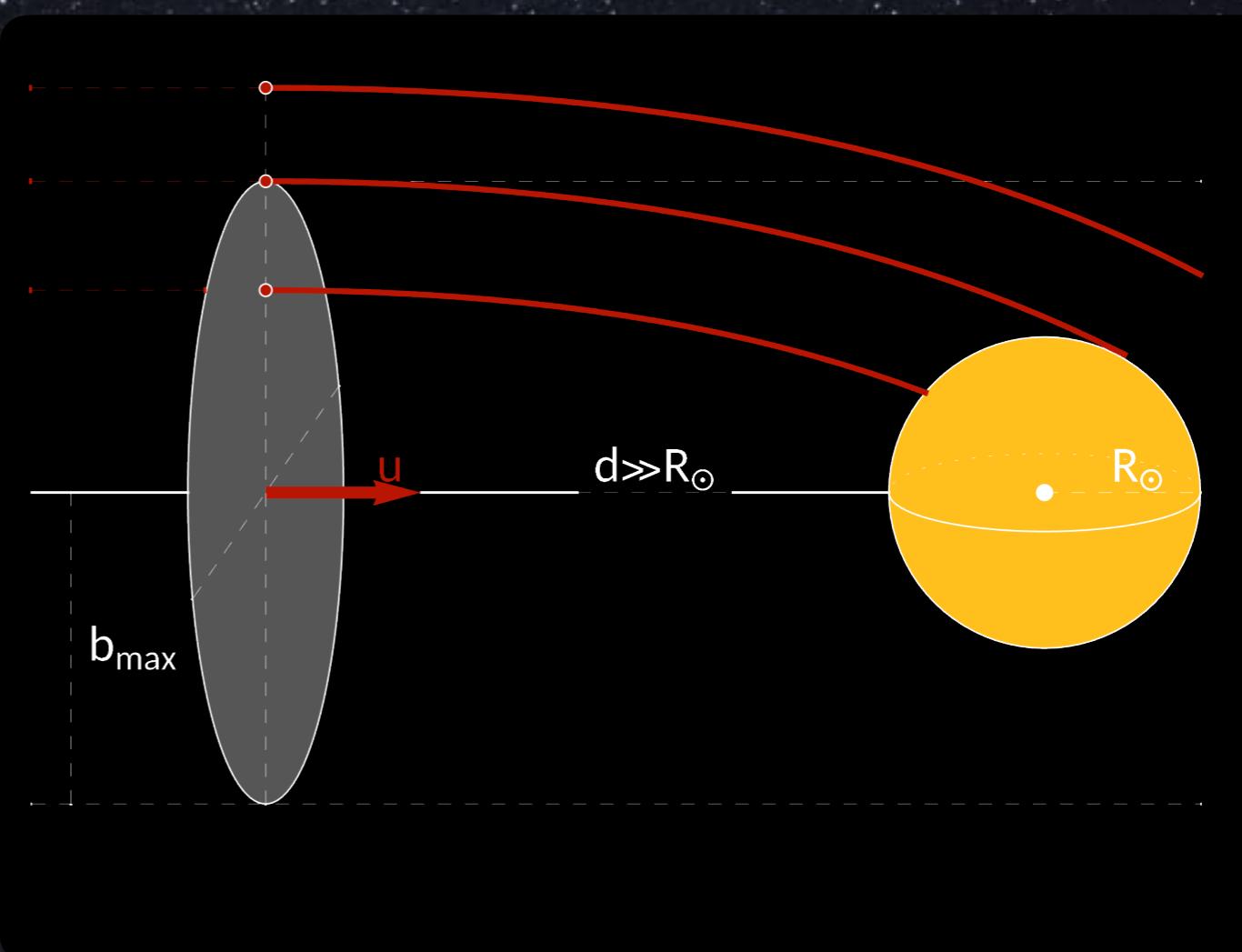
1. **Initial Conditions:** Where does the particle start?
2. **Free distance:** Where does the particle scatter?
3. **Target:** What does the particle scatter on? How fast is it?
4. **Scattering angle:** How does the particle scatter?

**Repeat steps 2-4.**



# Initial conditions

1. Sample a velocity  $u$  far away from the Sun from the halo.
2. Sample the initial position with a maximum impact parameter.
3. Propagate the particle analytically on its hyperbolic Kepler orbit to a location close to the Sun.



# Free propagation

- The probability for a DM particle to propagate underground without scattering:

$$P(\Delta t) = 1 - \exp\left(-\int_0^{\Delta t} \frac{dt}{\tau(r(t), v(t))}\right)$$

- Here, we used the mean free time

$$\tau(r, v) = \Omega(r, v)^{-1}$$

- This enables us to find the location of the next scattering.
- We solve the equations of motion iteratively using the Runge-Kutta-Fehlberg method. The particle scatters as soon as

$$\sum_i \frac{\Delta t_i}{\tau(r_i, v_i)} > -\ln(\xi)$$

# Scatterings inside the Sun

- Once we know that the DM particle scatters at a radial distance  $r$ , we can identify the target species. The probability to scatter on target  $i$  is

$$P_i = \frac{\Omega_i(r, v_\chi)}{\Omega(r, v_\chi)}$$

- The target's velocity can be sampled from the following distribution,

$$f(\mathbf{v}_T) = \frac{|\mathbf{v}_\chi - \mathbf{v}_T|}{\langle |\mathbf{v}_\chi - \mathbf{v}_T| \rangle} f_i(\mathbf{v}_T, T)$$

Maxwell-Boltzmann distribution

- Since we assume contact interactions with heavy mediators, the scattering is isotropic.
- Finally, the new DM velocity is given by

$$\mathbf{v}'_\chi = \frac{m_T |\mathbf{v}_\chi - \mathbf{v}_T|}{m_T + m_\chi} \mathbf{n} + \frac{m_\chi \mathbf{v}_\chi + m_T \mathbf{v}_T}{m_T + m_\chi}$$

$t = 0.$  hr

$N_s = 0$

$v_\chi = 456 \frac{\text{km}}{\text{s}}$

$E_\chi/E_0 = 1.$



$m_\chi = 100 \text{ MeV}$

$\sigma_p = 0.2 \text{ pb}$

Emken 2019

# From MC simulation to the SRDM flux

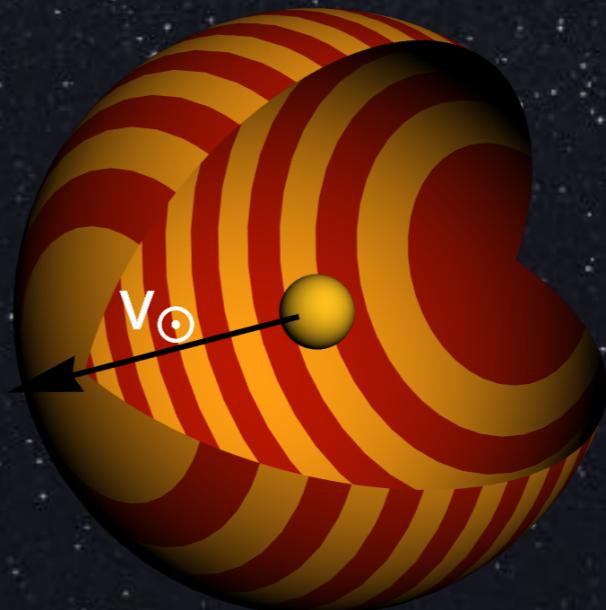
- The total reflection rate and flux is found given by the amount of reflections among the simulated particles.

$$\mathcal{R}_\odot^{\text{MC}} = \frac{N_{\text{refl}}}{N_{\text{sim}}} \Gamma(m_\chi) \quad \Phi_\odot^{\text{MC}} = \frac{\mathcal{R}_\odot^{\text{MC}}}{4\pi\ell^2} \quad \ell = 1 \text{ AU}$$

- The reflection spectrum is based on the speed data recorded at 1 AU distance. Instead of histograms, we prefer a smooth distribution and use Kernel density estimation (KDE).

$$\frac{d\mathcal{R}_\odot}{dv_\chi} = \mathcal{R}_\odot^{\text{MC}} f_\odot^{\text{KDE}}(v_\chi) \quad \frac{d\Phi_\odot^{\text{MC}}}{dv_\chi} = \frac{1}{4\pi\ell^2} \frac{d\mathcal{R}_\odot}{dv_\chi}$$

- To study anisotropies of the SRDM flux, we define equal-area isoreflection rings and perform the analysis for each ring.



# The Simulation Code

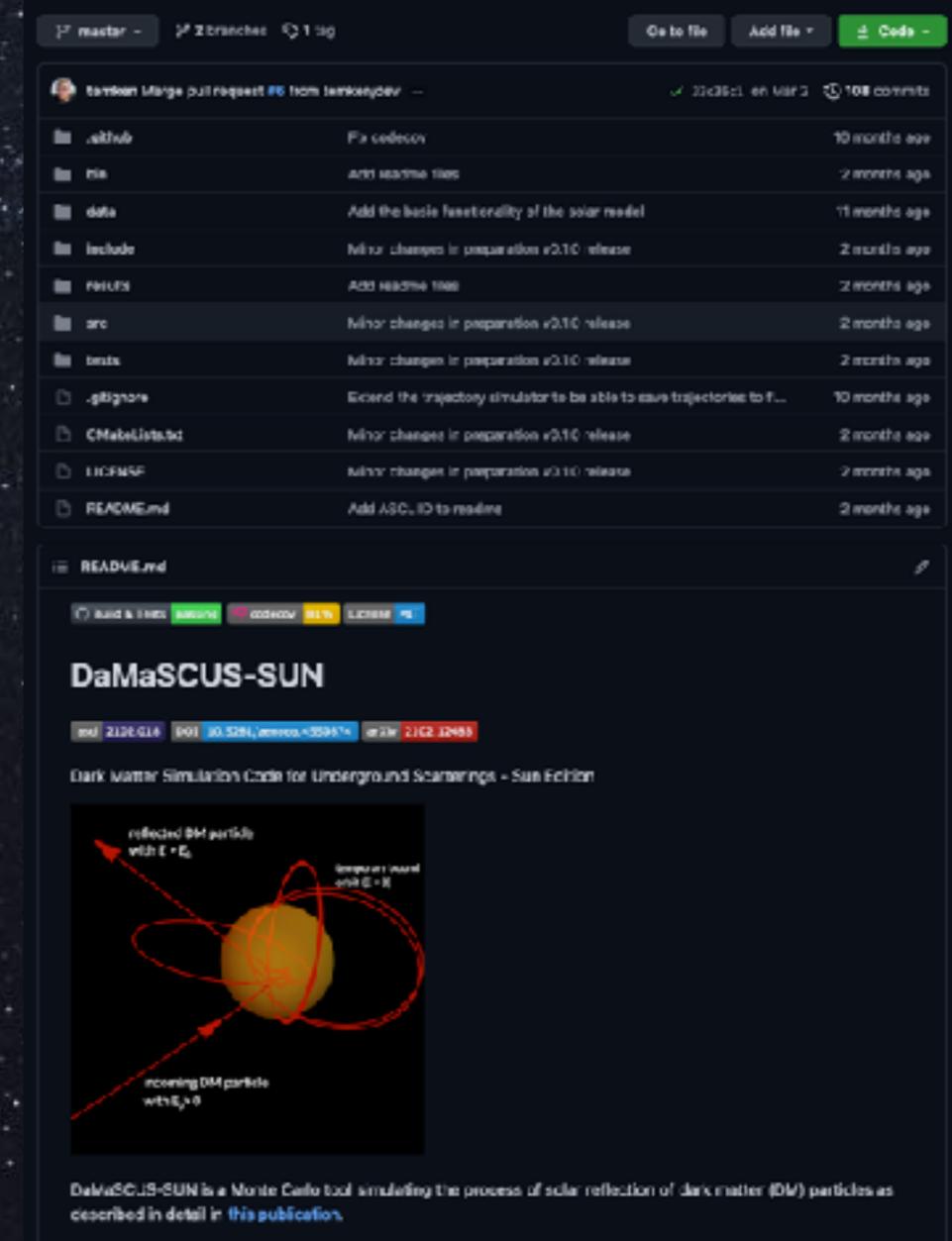
"The code is what you did, the paper is what you think you did."

-Patrick Koppenburg on Twitter

- The *Dark Matter Simulation Code for Underground Scatterings - Sun Edition* (DaMaSCUS-SUN) is publicly available on *github*.

<https://github.com/temken/DaMaSCUS-SUN>

- Written in C++, built with CMake, set up with unit and build testing, code coverage,...
- Fully parallelized (MPI), ready to run on HPC clusters.



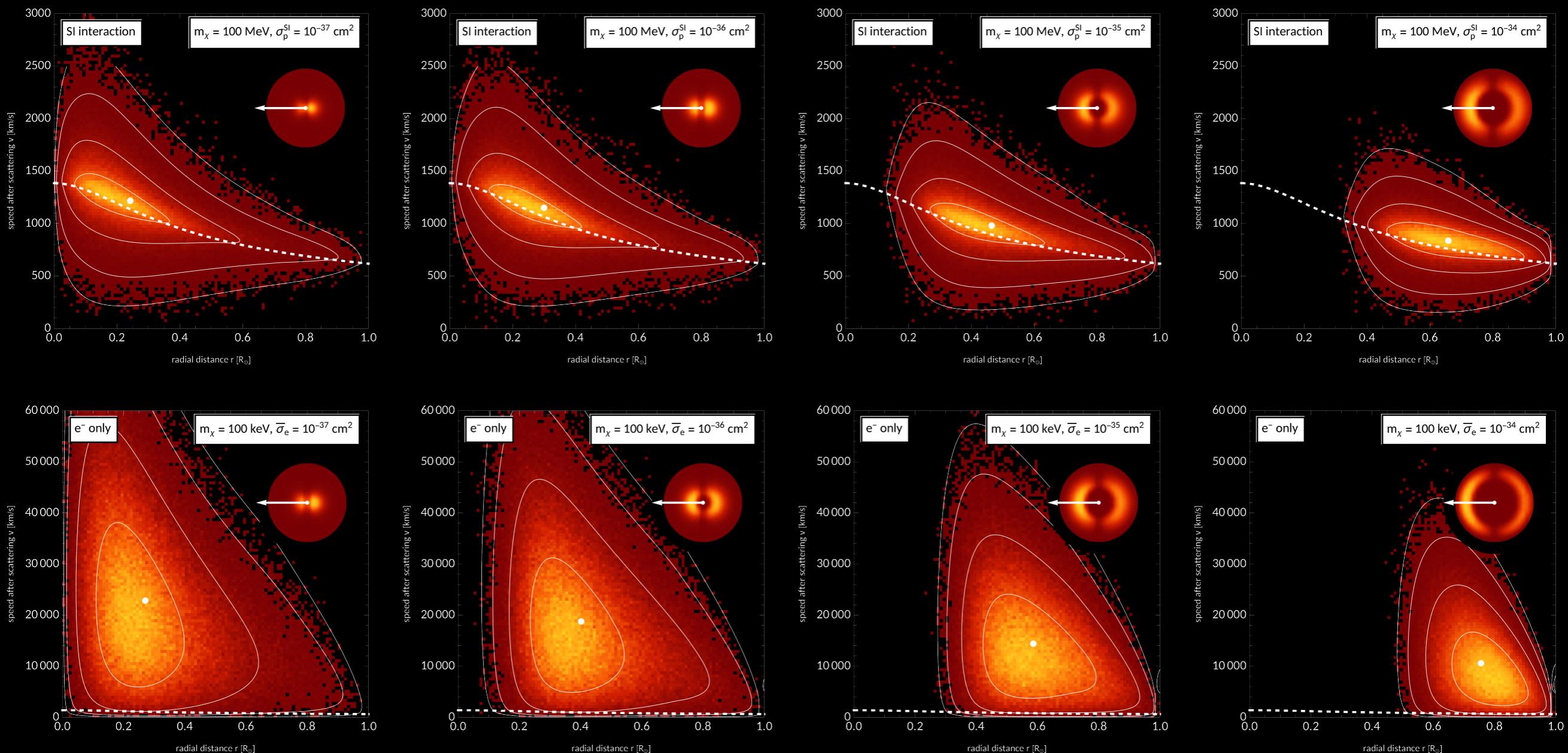
# IV. Results

```
// New velocity Eigen::Vector3d v2 = sqrt(GNewton*mSun/p)*(e+sin(theta2)*x2.normalized());
// double F2 = e*cosh((e+cos(theta2))/(1.0+e*cos(theta2))); // double M2 = e*sinh(F2)-F2; // double t2 = e*q
// instance at which we sample from the halo distribution double R=100.0*AU; if(R<100.0*rSun) { cerr <<"Warning in .
// placeholderIC_1,vEarth); double u = Rejection_Sampling(prdr,e,B,(vEsc+vEarth),ymax,PRNG); //Velocity Directio
// dem Point on a flat disk at distance R Eigen::Vector3d ez=-vIni.normalized(); Eigen::Vector3d ex{0,ez[2]
// cos(phi)+ex+sin(phi)*ey}; // cout <<"Norm = "<<ex.norm()<<endl; // cout <<"Norm = "<<(R+ex).norm()<<
// r[0]/rSun<<"\t"<<IC.Position()[1]/rSun<<"\t"<<IC.Position()[2]/rSun<<"\t"<<IC.Radius()</rSun<<"\t"<<IC.Velocity(
// r_Shift(IC,rMax,model); } //2. orbit simulation //Right hand sides of the 1st order equations of motion. double
// Euler_Cromer // void EC_Step(double &t,double &r,double &v,double &phi,double &J,double dt,SunModel &model) //{
// // //RK coefficients // double k_r[4]; // double k_v[4]; // double k_p[4]; // k_r[0]=dt*drdt(v); // k_v[0]
// = dt*dvdt(r+k_r[1]/2.0,J,model); // k_p[2]= dt*dphidt(r+k_r[1]/2.0,J); // k_r[3]= dt*drdt(v+k_v[2])
// k_v[1]= dt*dvdt(r,J,model); k_p[1]= dt*dphidt(r,J); k_r[1]= dt*drdt(v+k_v[0]/4.0); k_v[1]= dt*dvdt(r+k_r[0]);
// .0*k_r[1],J); k_r[3]= dt*drdt(v+1932.0/2197.0*k_v[0]-7200.0/2197.0*k_v[1]+7296.0/2197.0*k_v[2]); k_v[3]= dt*drdt(v+k_v[1]
// +3680.0/513.0*k_v[2]-845.0/4184.0*k_v[3]); k_v[4]= dt*dvdt(r+439.0/216.0*k_r[0]-8.0*k_r[1]
// -5544.0/2505.0*k_v[2]+1859.0/4184.0*k_v[3]-11.0/40.0*k_v[4]); k_v[5]= dt*dvdt(r-8.0/27.0*k_r[0]+2.0*k_r[1]-384
// 0*k_r[2]+r+28.0/216.0*k_r[0]+1408.0/2868.0*k_r[2]+2167.0/4181.0*k_r[3]-1.0/8.0*k_r[4]; double v4= v+28.0/216.0
// *k_r[2]+28561.0/56430.0*k_r[3]-9.0/50.0*k_r[4]+2.0/55.0*k_r[5]; double v5m v+16.0/135.0*k_v[0]+6656.0/12825.0*k
// r4,abs(v5-v4),abs(phis-phi4)); std::vector<double> tolerance = {1.0*xm,1e-3*xm/sec,1e-6}; //New stepsize
// s, std::end(deltas)); //Next steps if(err[0]<tolerance[0]&&err[1]<tolerance[1]&&err[2]<tolerance[2]) { if(r<
// x1 = speed*dt/mfp; } t+=dt; r= r4; v= v4; phi= phi4; dt= std::min(dtMin, std::min(delta*dt,dtMax)); } else { dt
// Eigen::Vector3d& axes) { double w_phi = J/norm(r2); Eigen::Vector3d xNew = r*(cos(phi)*axes[0]+sin(phi)*axes[1]
// scattering or (b) leaving the sun. bool Propagate(Event& xe,DN_Particle& DN,SunModel &model, std::min
// ).normalized(); Eigen::Vector3d ax_y = ax_z.cross(ax_x); std::vector<Eigen::Vector3d> axes = {ax_x,ax_y
// }).dot(ax_z); //Sample -log(1-xi) double logxi = -1.0*log(1.0-ProbabilitySample(PRNG)); //Start integr
// _Step(t,r,v_r,phi,J,dt,logxi, DN,model); if(xNew==0) { Event xNew = PlaneToSP(t,r,phi,v_r,J,axes); t <=xNew.Posit
// need()-model.vEsc2(xNew.Radius()); } <<"\t" <<InUnits(xNew.Speed(),km/sec)<<endl; } // double dd = (xOld.Position
// -xOld.RMaxes)>xMaxes(v_r*x_r+J*x_J/r/r)>model.vEsc2(r)) { propagate=false; xe=PlaneToSP(t,r,phi,v_r,J,axes); } else
// (Event &xe,DN_Particle& DN,SunModel &model, std::min t19937A PRNG) // double speed=x.Speed(); } // cout <<"Scatt
// ering coordinates(1.0,Thetasample(PRNG),Phisample(PRNG)); double vRel = (vTarget-xe.Velocity()).norm(); Eigen
// ede<<endl; // if(speed>sqrt(model.vEsc2(r))>>vNew.norm()>>sqrt(model.vEsc2(r))) cout <<"\tcapture"<<endl; //
// & model, unsigned int &nScattering, std::min t19937A PRNG) { // Save trajectory of stream
// x.Speed()-model.vEsc2(x.Radius()))/E0<<endl; //Output // Event x = IC; bool success; //Counters nScat
// if(r>rSun) { if(nScattering>nScattering_max) { success = false; break; } // double E1=DN.mass/2.0*(x
// *(x.Speed()+x.speed()-model.vEsc2(x.Radius()))/E0<<endl; // if(E1>0 && E2<0) { // cout <<"capture"<<endl;
// <"\t" << r / rSun << endl; } // f. close(); } // if (x . Speed () < sqrt ( model . V
// olation(0)/rSun<<"\t"<<xFinal.Position()[1]/rSun<<"\t"<<xFinal.Position()[2]/rSun<<"\t"<<xFinal.Radiu
// cess=false; if(x.Time()/sec<9999) return false; return success; } Result generate_Data(Conf
// i, numproc-1) ? 0 : config.rank+1; int tag = 0; MPI_Status status; MPI_Request req; unsigned int request; // /Datapoi
// nt; data.resize(config.nRings); //Counters //Total number of simulated particles unsigned long int &count
// Eigen::vector<unsigned long int> L_Counter_Data_new(config.nRings,0); //Number of passes // std::vector<unsig
// ne scatters; std::vector<double> G_nScattering_Av(config.nRings,0.0); std::vector<double> L_nScat
// ISend(&G_Counter_Data.Front(),config.nRings,MPI_UNSIGNED_LONG,destination,tag,MPI_COMM_WORLD
// ); unsigned int nScattering=0; bool success = Simulate_Trajectory(x, DN, model, nScattering, PRNG)
// .nScattering_Av(IsoRing)+nScattering)/L_Counter_Data[IsoRing]; double speed=x.Speed(); fns
// DN_WORLD,&flag,&status); if(flag) { //Receive the tokens MPI_Recv(&Counter_Data.front(),config.nR
// r_data[i]+=L_Counter_Data_new[i]; L_Counter_Data_new[i]=0; } // cout <<config.rank <<"\t" <<G_Co
// else if (nmini>config.nData) tag = status.MPI_TAG; //Pass on the tokens unless you are the very la
// st. cout <<"\r"; for(int i=0;i<2.0*BarLength;i++) cout <<" "; cout <<"\r"; for(int i=0;i<BarLength
// -Counter_Free,1,MPI_UNSIGNED_LONG_LONG,MPI_SUM,MPI_COMM_WORLD); MPI_Allreduce(&L_nScat
// > Global_Data; for(int i = 0;i<config.nRings;i++) { unsigned long int Global_SampleSize; MPI
// MPI_Gatherv(&data[i].front(),data.size(),MPI_Datapoint,&Ring_Data.front()
// ) MPI_BARRIER(MPI_COMM_WORLD); // for(int i=0;i<config.nRings;i++) // { // if(config.r
// ,1,MPI_DOUBLE,MPI_MAX,MPI_COMM_WORLD); return Result(Global_Data, DN, &Counter_Simulat
// std::vector<double> nscav, double dt) { data=dat; particle = p; // sample_size
// } nscatterings_mean=nscav; nscatterings_mean_tot=0.0; for(unsigned int i=0;i<
// & config) { if(config.rank==0) { std::cout <<"Simulation summe
// <Round(1.0*nSimulations/duration)<</sec)<<std::endl; // (config.nRings==1) st
// << " , " <<Round(100.0*nFails/nSimulations)<<")<<std::endl; } } } } #inc
// functions.hpp" using namespace libconfig; //1. String with input param
// mbaaa=mD; //Parallelisation rank = myrank; numprocs=ap; Configuration
// const FileIOException Affex) { std::cerr << "I/O error while re
// trv { simTB = cfg.lookup("simTB").c_str(); l = affex(const
// file." << endl; exit(EXIT_FAILURE); } //2. Event <
// "None." << endl<<endl; } //2. Event <
// << "None." << endl<<endl;
```

# The first scattering inside the Sun

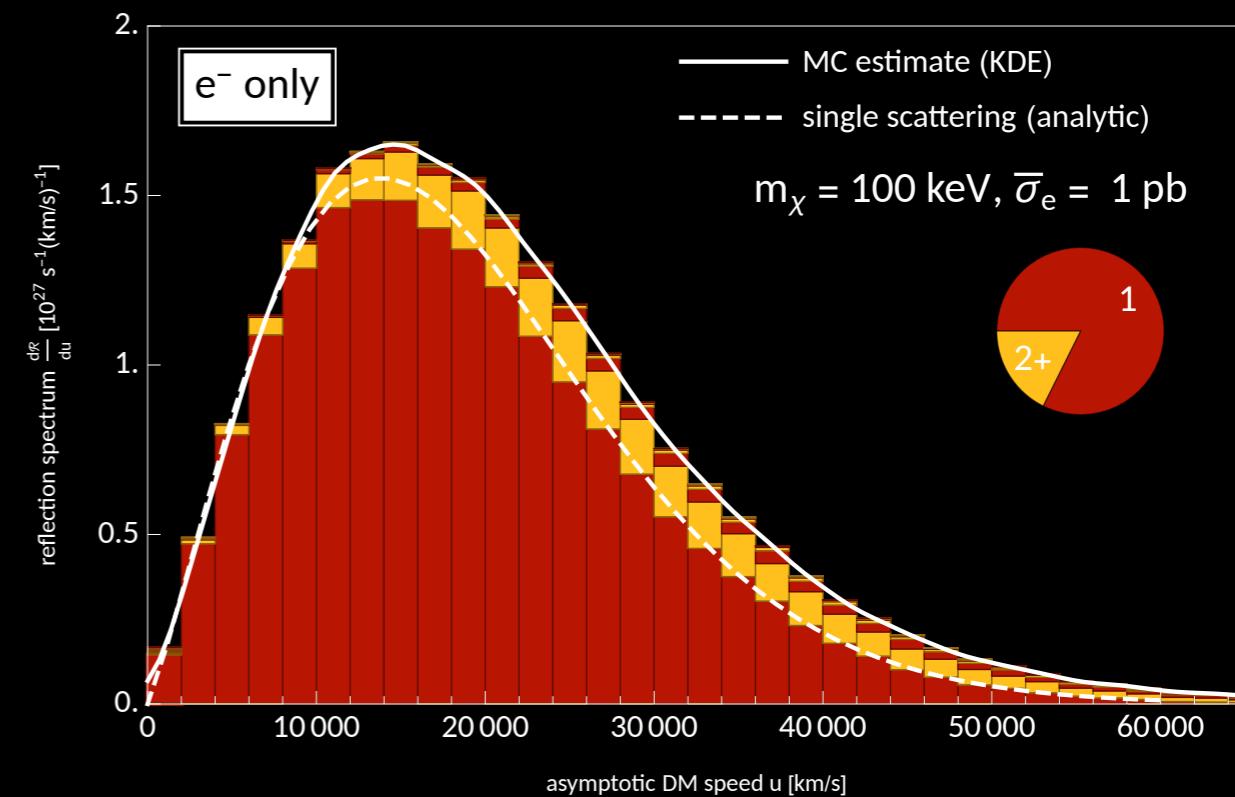
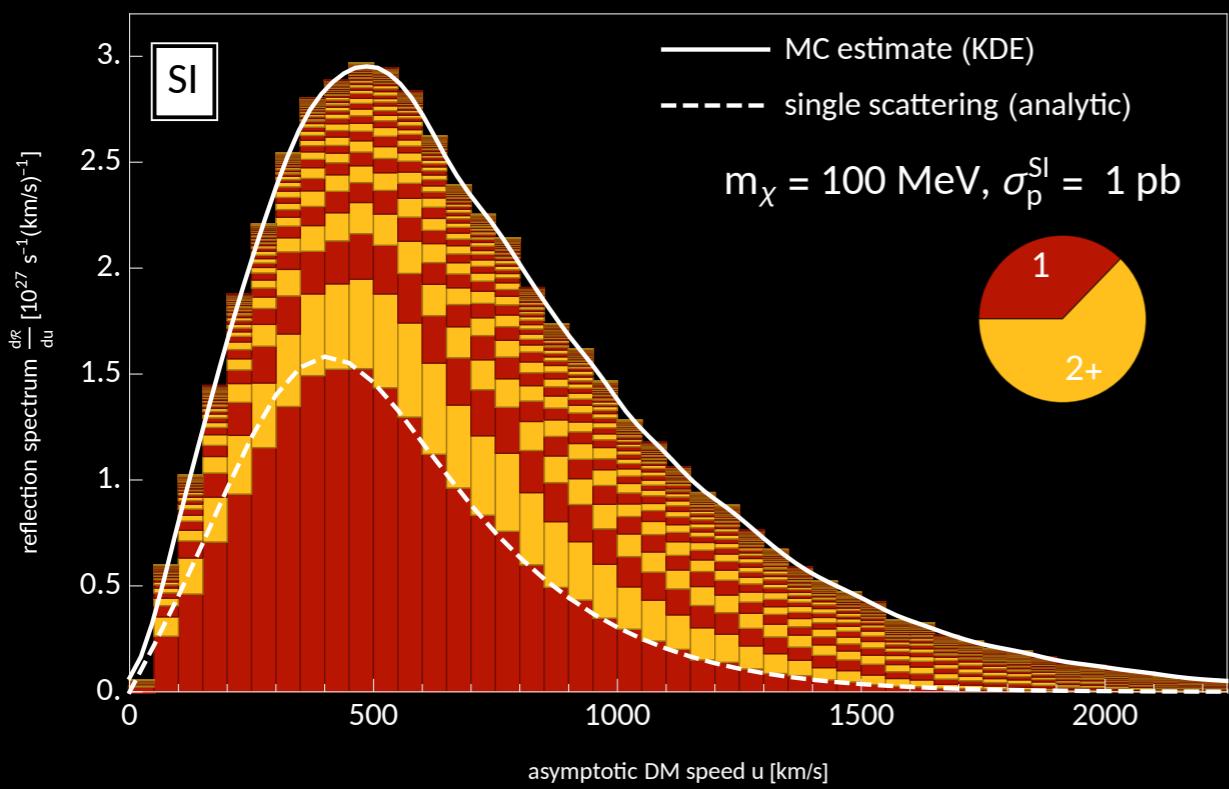
- The first scattering allows comparison with our analytic description of DM scatterings inside the Sun.

TE, C. Kouvaris, N. G. Nielsen, PRD, 97 (2018), 063007



# The impact of multiple scatterings

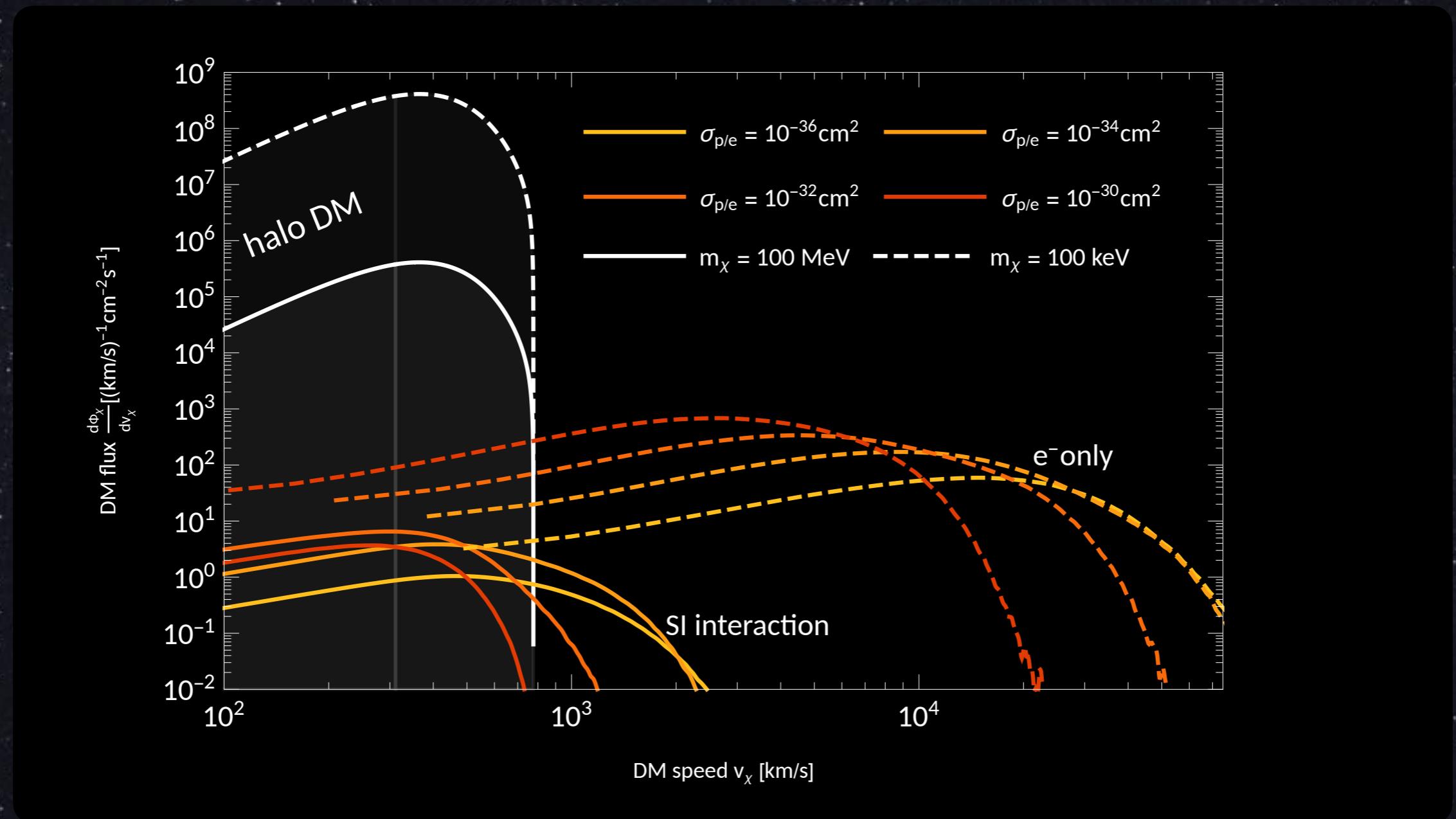
- Using MC simulations allows to quantize the contribution of multiple scatterings to the SRDM flux.



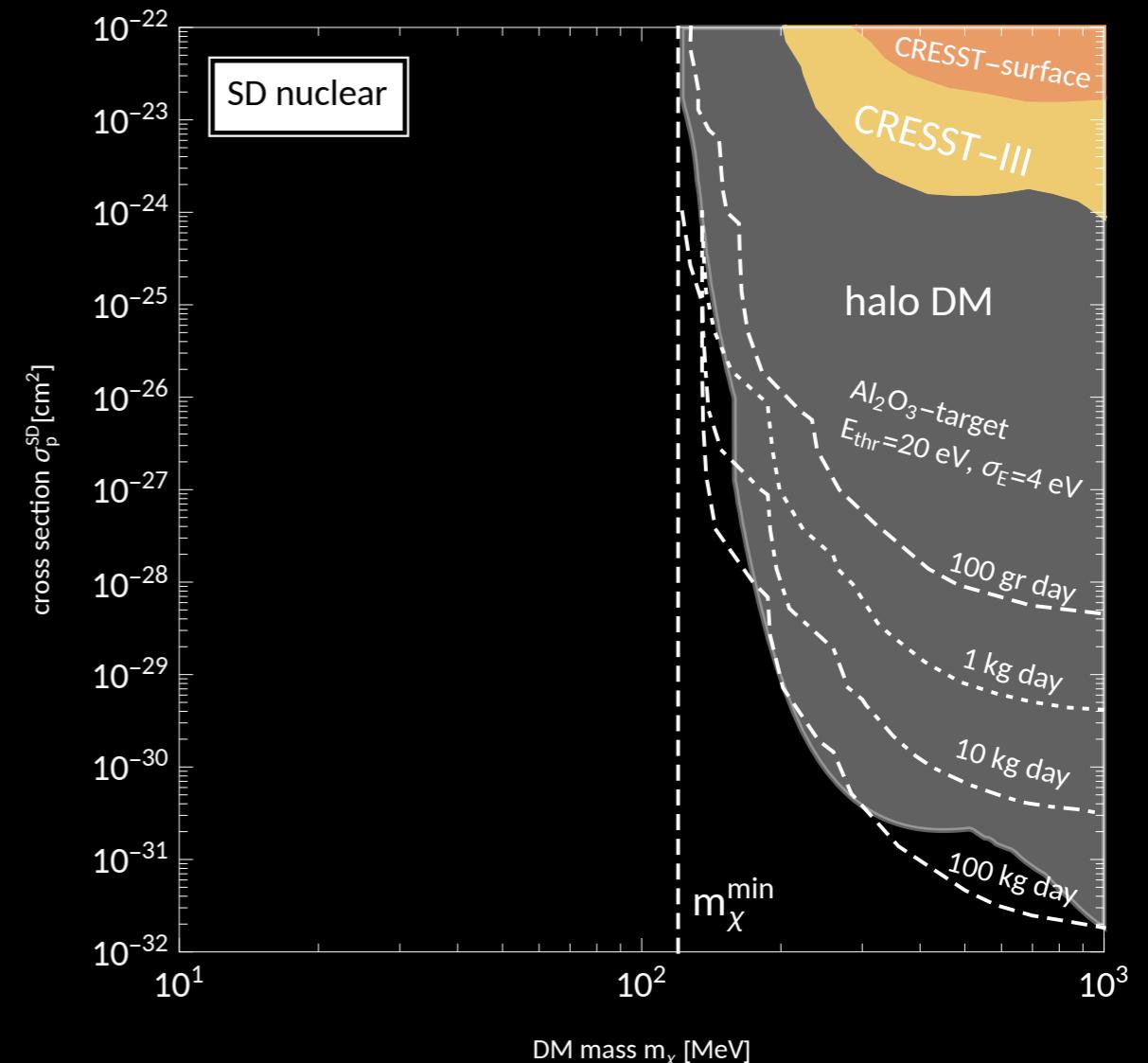
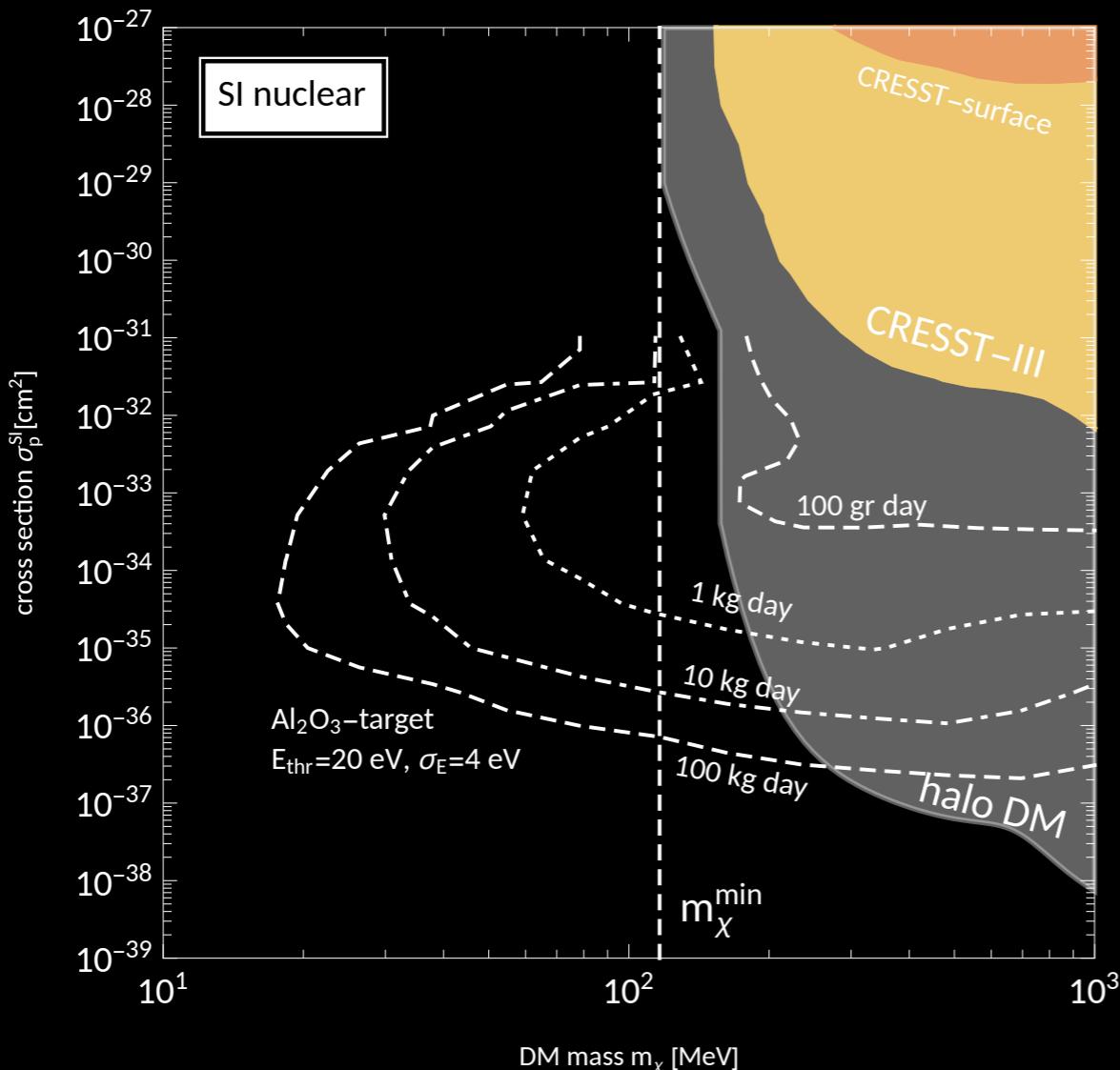
→ Great consistency between analytic formalism and MC description!

# The solar reflection DM flux

- Compare the SRDM flux to the halo DM flux.



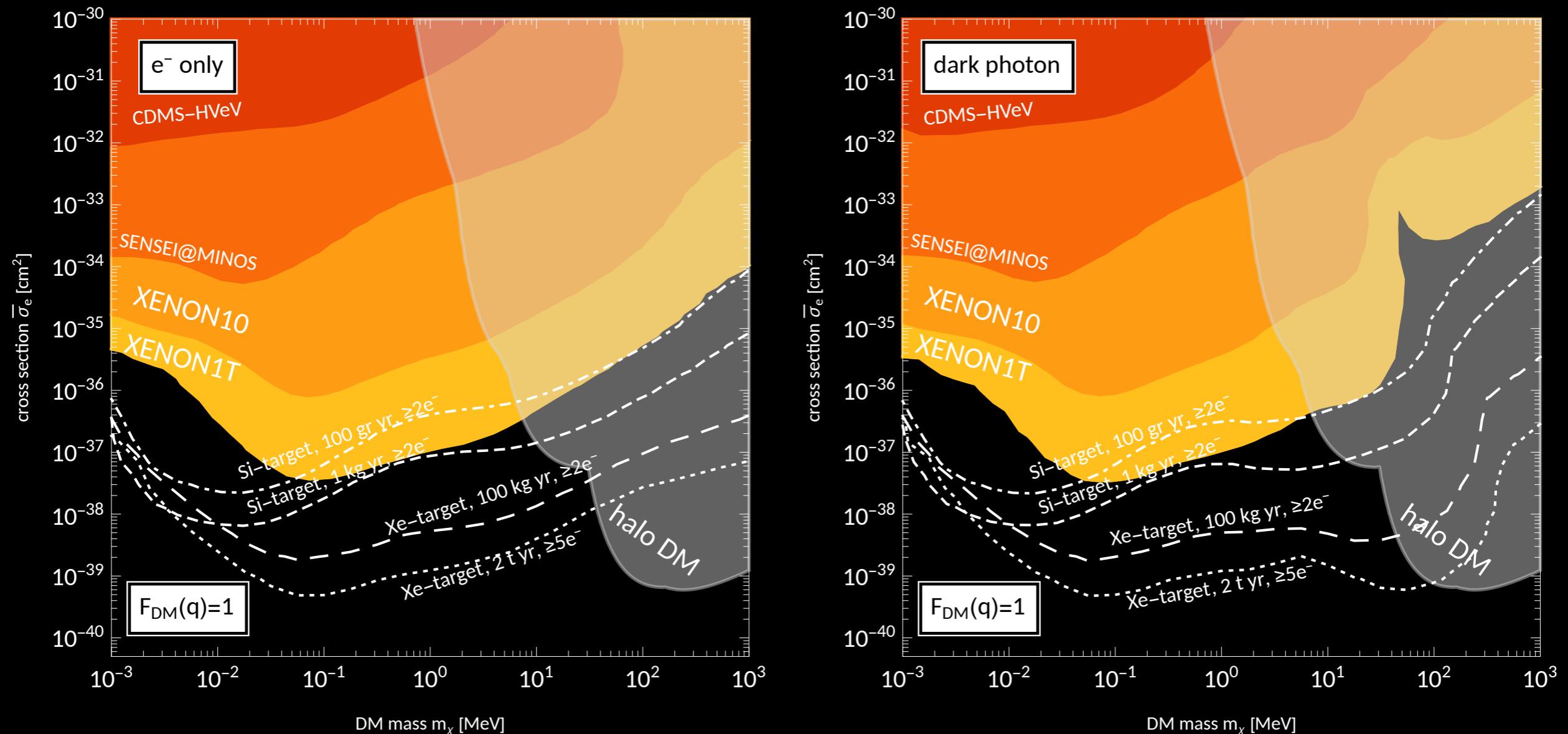
# Exclusion limits for DM-nucleus interactions



→ Higher exposures probe lower masses!

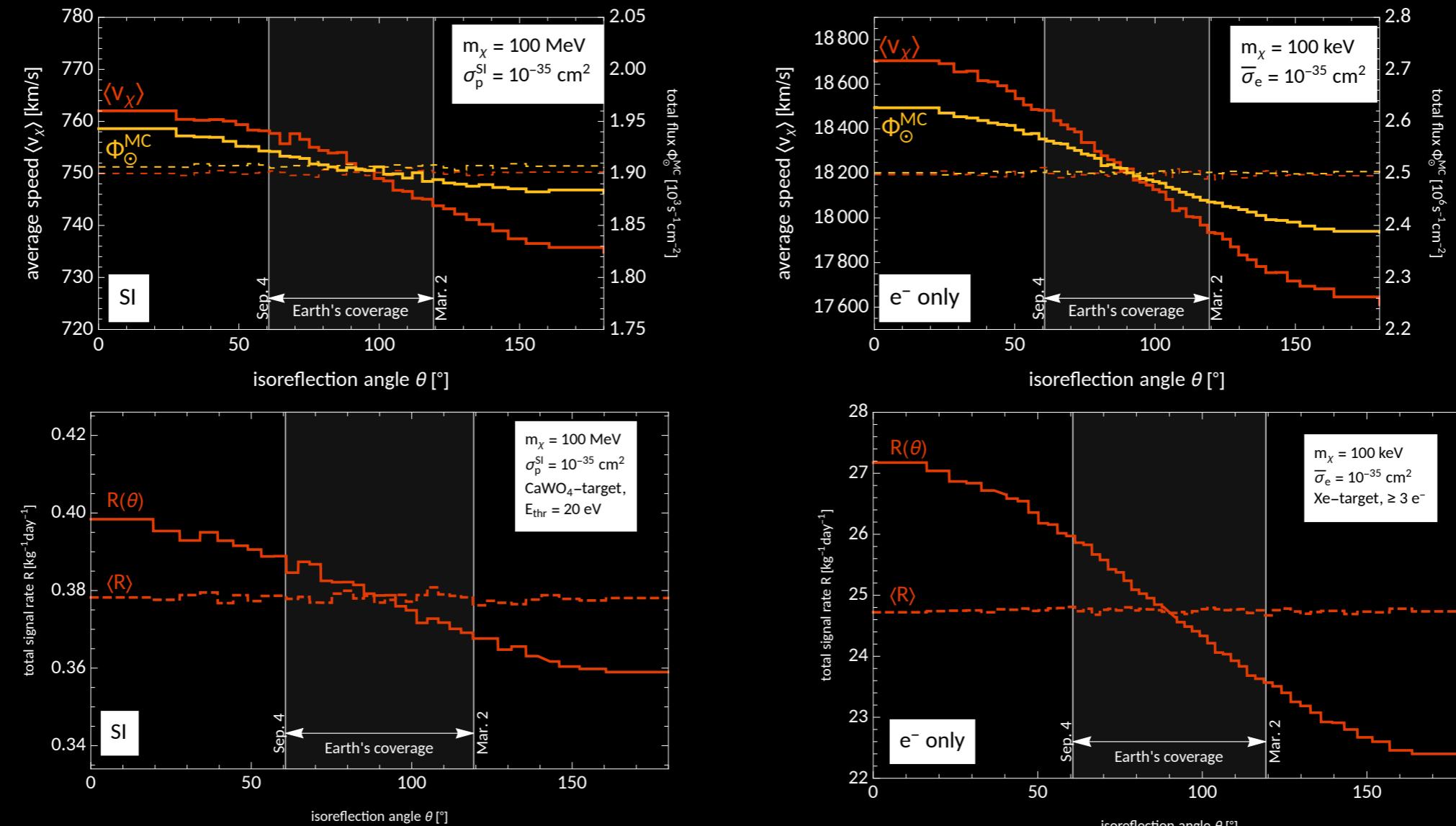
- For SI interactions, next generation experiments can extend their sensitivity to lower masses through SRDM.

# Exclusion limits for DM-electron interactions



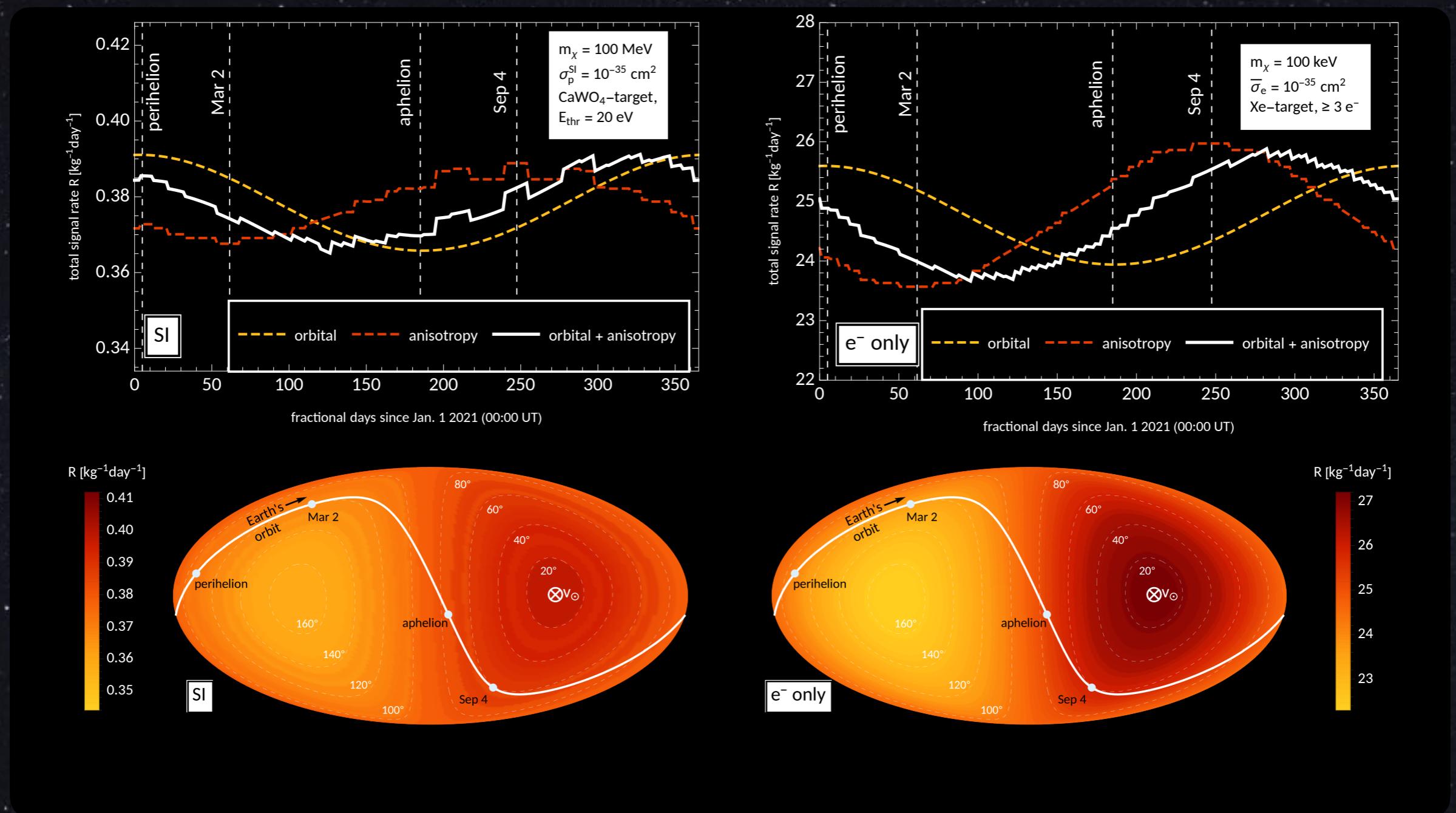
→ Solar reflection extends the experiments' mass sensitivity by multiple orders of magnitude.

# Anisotropies of the SRDM flux



→ The Sun does **not** eject SRDM particles isotropically into the solar system.

# Annual signal modulations



→ The orbital and anisotropy modulations feature comparable amplitudes.

# V. Summary & Outlook

- Scatterings on hot solar targets can accelerate light DM particles.
  - The energetic flux of reflected DM particles can extend the sensitivity of direct detection experiments to lower masses especially for DM-electron interactions.
  - We performed MC simulations to describe the SRDM flux for 4 different DM scenarios and derived exclusion limits based on existing and planned experiments.
  - An SRDM signal would feature a rich modulation signature. We investigated the annual modulation.
- 

- So far we only considered heavy mediators. What about light mediators?  
An, H., et al., [arXiv:2108.10332]
- Migdal scatterings + SRDM could potentially extend sensitivity to low mass DM searches via nuclear recoils.

# Thank you!

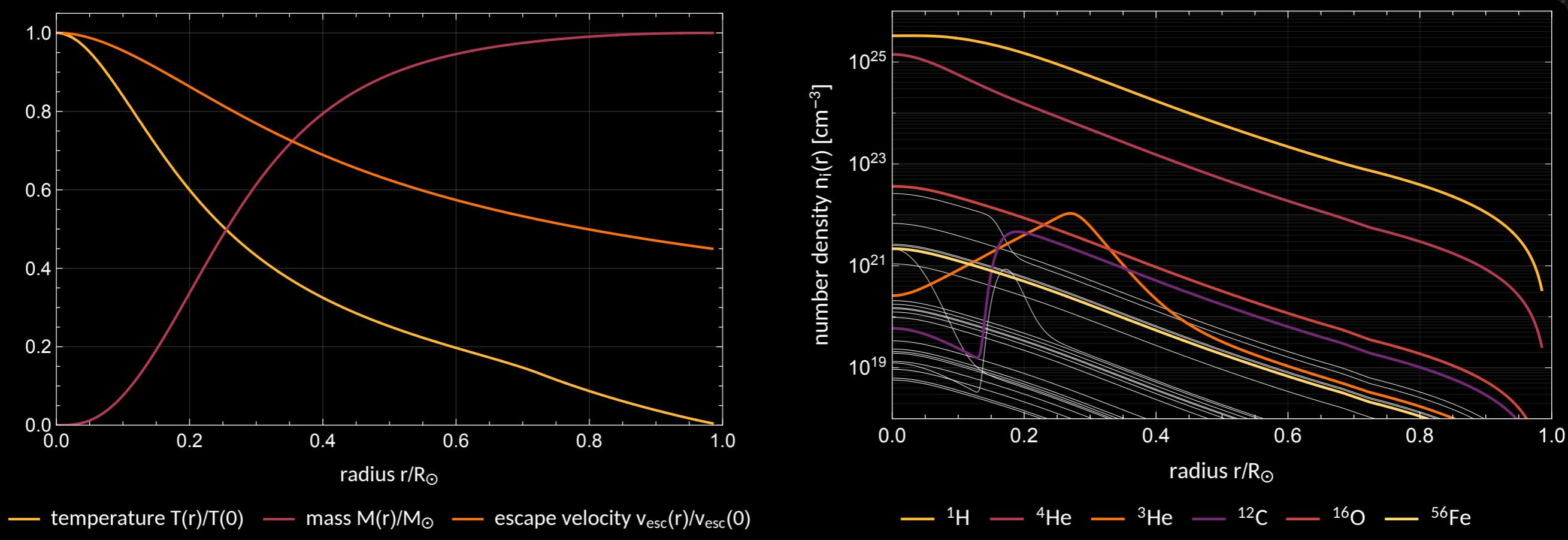
```
//2)*ax_y; //9. New velocity Eigen::Vector3d v2 = sqrt(GNewton*mSun/p)*(e+sin(theta2))*x2.normalized();
// double F2 = asinh((e+cos(theta2))/(1.0+ew*cos(theta2))); // double M2 = ew*inh(F2)-F2; // double t2 = sq
stance at which we sample from the halo distribution double R=100.0*AU; if(R<100.0*rSun) { cerr <<"Warning in .
laceholderIC_1,vEarth); double u = Rejection_Sampling(pdr,e,B,(vEsc+vEarth),ymax,PRNG); //Velocity Directio
dem Point on a flat disk at distance R Eigen::Vector3d ez=-vIni.normalized(); Eigen::Vector3d ex{0,ez[2]
cos(phi)*ex+sin(phi)*ey}; // cout <<"Norm = "<<xIni.norm() /AU<<endl; // cout <<"Norm = "<<(R+ex).norm() /A
[0]/rSun<<"\t"<<IC.Position()[1]/rSun<<"\t"<<IC.Position()[2]/rSun<<"\t"<<IC.Radius() /rSun<<"\t"<<IC.Velocity(
r_Shift(IC,rMax,model); } //2. orbit simulation //Right hand sides of the 1st order equations of motion. double
Euler_Cromer // void EC_Step(double &t,double &r,double &v,double &phi,double &J,double dt,SunModel &model) // {
// //RK coefficients // double k_r[4]; // double k_v[4]; // double k_p[4]; // k_r[0]=dt*drdt(v); // k_v[0]
= dt*dvdt(r+k_r[1]/2.0,J,model); // k_p[2]= dt*dphidt(r+k_r[1]/2.0,J); // k_r[3]= dt*drdt(v+k_v[2])
+k_v[3]); // phi+= 1.0/6.0*(k_p[0]+2.0*k_p[1]+k_p[3]); // //ODE integration with Runge Kutta Fehl
k_v[0]= dt*dvdt(r,J,model); k_r[1]= dt*drdt(v+k_v[0]/4.0); k_v[1]= dt*dvdt(r+k_r[1]);
+k_r[1],J); k_r[3]= dt*drdt(v+1932.0/2197.0*k_v[0]-7200.0/2197.0*k_v[1]+7296.0/2197.0*k_v[2]); k_v[3]= dt*dr
+k_v[1]+3680.0/513.0*k_v[2]-845.0/4184.0*k_v[3]); k_v[4]= dt*dvdt(r+439.0/216.0*k_r[0]-8.0*k_r[1]
-5544.0/2505.0*k_v[2]+1859.0/4184.0*k_v[3]-11.0/40.0*k_v[4]); k_v[5]= dt*dvdt(r-8.0/27.0*k_r[0]+2.0*k_r[1]-384
ble r4= r+28.0/216.0*k_r[0]+1408.0/2868.0*k_r[2]+2107.0/4181.0*k_r[3]-1.0/8.0*k_r[4]; double v4= v+28.0/216.0
*k_r[2]+28561.0/56430.0*k_r[3]-9.0/50.0*k_r[4]+2.0/55.0*k_r[5]; double v5= v+16.0/135.0*k_v[0]+6656.0/12825.0*k_v[1]
,abs(v5-v4),abs(phi5-phi4)); std::vector<double> tolerance = {1.0*km,1e-3*km/sec,1e-6}; //New stepsize
},std::end(deltas)); //Next steps if(err[0]<tolerance[0]&&err[1]<tolerance[1]&&err[2]<tolerance[2]) { if(r<
x1 = speed*dt/dtP; } t+=dt; r= r4; v= v4; phi= phi4; dt= std::min(dtMin, std::min(delta*dt,dtMax)); } else { dt
Eigen::Vector3d& axes) { double w_phi = J/pow(r,2); Eigen::Vector3d xNew = r*(cos(phi)*axes[0]+sin(phi)*w
scattering or (b) leaving the sun. bool Propagate(Event& xe,DN_Particle& DN,SunModel &model, std::in
)).normalized(); Eigen::Vector3d ax_y = ax_z.cross(ax_x); std::vector<Eigen::Vector3d> axes = {ax_x,ax_y
}).dot(ax_z); //Sample -log(1-xi) double logXi = -1.0*log(1.0-ProbabilitySample(PRNG)); //Start integr
_step(t,r,v_r,phi,J,dt,logXi,DM,model); if(xNew==0) { Event xNew=PlaneToSP(t,r,phi,v_r,J,axes); t <=xNew.Posit
Speed()-model.vEsc2(xNew.Radius())); <<"\t" <<InUnits(xNew.Speed(),km/sec) <<endl; } // double dd = (xOld.Position
num // t <<xNew.Position()[0]/rSun<<"\t" <<xNew.Position()[1]/rSun<<"\t" <<xNew.Position()[2]/rSun<<"\t" <</sec
Old<rMax& rMax(x_r*x_r+J*J/r/r)>model.vEsc2(r)) { propagate=false; xe=PlaneToSP(t,r,phi,v_r,J,axes); } else
(Event &x, DN_Particle& DN,SunModel &model, std::mt19937& PRNG) { // double speed=x.Speed(); } // cout <<"Scatt
ordinates(1.0,Thetasample(PRNG),PhiSample(PRNG)); double vRel = (vTarget-x.Velocity()).norm(); Eigen
eed0<<endl; // if(speed>sqrt(model.vEsc2(r))&&xNew.norm()<sqrt(model.vEsc2(r))) cout <<"\tcapture"<<endl; //
oI& model, unsigned int &nScattering, std::mt19937& PRNG) { // Save trajectory of stream
x.Speed()-model.vEsc2(x.Radius()))/E0<<endl; //Output // Event x = IC; bool success; //Counters nScat
if(r<rSun) { if(nScattering>nScattering_max) { success = false; break; } // double E1=DN.mass/2.0*(x
*(x.Speed()+x.speed()-model.vEsc2(x.Radius()))/E0<<endl; // if(E1>0 & E2<0) { // cout <<"capture"<<endl;
<"\t" <<r/rSun << endl; } // t .close(); } // if (x .Speed () < sqrt ( model .V
olution()[0]/rSun<<"\t" <<xFinal.Position()[1]/rSun<<"\t" <<xFinal.Position()[2]/rSun<<"\t" <<xFinal.Radius()
cess=false; if(x.Time()/.sec<9999) return false; return success; } Result generate_data(Configurations config
,numprefs-1)? 0 : config.rank+1; int tag = 0; MPI_Status status; MPI_Request send_request; // /Data point
data; data.resize(config.nRings); //Counters //Total number of simulated particles unsigned long int &count
l<vector<unsigned long int> L_Counter_Data_new(config.nRing,0); //Number of passes // std::vector<unsigne
ber of scatterings; std::vector<double> G_nScattering_Av(config.nRings,0.0); std::vector<double> L_nScat
Isend(&G_Counter_Data.Front(),config.nRings,MPI_UNSIGNED_LONG,destination,tag,MPI_COMM_WORLD
G); unsigned int nScattering=0; bool success = Simulate_Trajectory(x,DM,model,nScattering,PRNG);
_nScattering_Av(IsoRing)+nScattering)/L_Counter_Data[IsoRing]; double speed=x.Speed(); fns
DM_WORLD,&flap,&status); if(flag) { //Receive the tokens MPI_Recv(&G_Counter_Data.Front(),config.nRi
r_data[i]+=L_Counter_Data_new[i]; L_Counter_Data_new[i]=0; } // cout <<config.rank <<"\t" <<G_Coun
else if (nmini>config.nData) tag = status.MPI_TAG; //Pass on the tokens unless you are the very la
gth cout <<"\r"; for(int i=0;i<2.0*BarLength;i++) cout <<" "; cout <<"\r"; for(int i=0;i<BarLength
{ cout <<"\r"; for(int j=0;j<10*BarLength;j++) cout <<" "; cout <<"\r"; } // for(int i=0
_Counter_Free,1,MPI_UNSIGNED_LONG_LONG,MPI_SUM,MPI_COMM_WORLD); MPI_Allreduce(&L_nScat
> Global_Data; for(int i = 0;i<config.nRings;i++) { unsigned long int Global_SampleSize; MPI
cv_displs[config.numprefs]; MPI_Allgather(&L_Counter_Data[i],1,MPI_UNSIGNED_LONG,&nData_loc
SampleSize); // MPI_Gatherv(&data[i].front(),data.size(),mpi_datapoint,&Ring_Data.front(
) MPI_BARRIER(MPI_COMM_WORLD); // for(int i=0;i<config.nRings;i++) // { // if(config.r
a,1,MPI_DOUBLE,MPI_MAX,MPI_COMM_WORLD); return Result(Global_Data,DM,&Counter_Simulat
std::vector<double>& mscav,double dt) { data=dat; particle = p; // sample_size
} nscatterings_mean=nscav; nscatterings_mean_tot=0.0; for(unsigned int i=0;i<
e & config) { if(config.rank==0) { std::cout <<"Simulations summe
<Round(1.0*nSimulations/duration)<<"/sec)"<<std::endl; if(config.nRings==1) st
<< ", "<<Round(100.0*nFails/nSimulations)<<")" <<std::endl <<"\tduration[s]">
size[i] <<"\t\t" <<Round(nScatterings_mean[i])<<std::endl; } } } #inc
functions.hpp" using namespace libconfig; //1. Struct with input param
mbaria=mD; //Parallelisation rank = myrank; numprefs= np! Configuration
const FileIOException Affex) { std::cerr <<"I/O error while re
try { simTB = cfg.lookup("simTB").c_str(); } catch(const
file." << endl; exit(EXIT_FAILURE); } //2. Event <<
"done." << endl; } //2. Event <<
```

# VI. Backup Slides

# The solar model

Standard Solar Model AGSS09

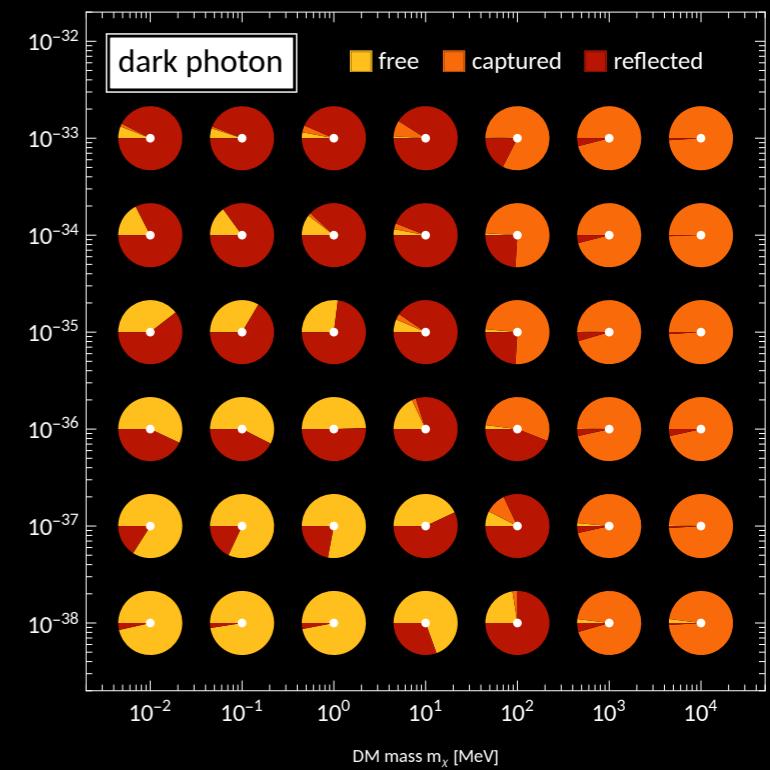
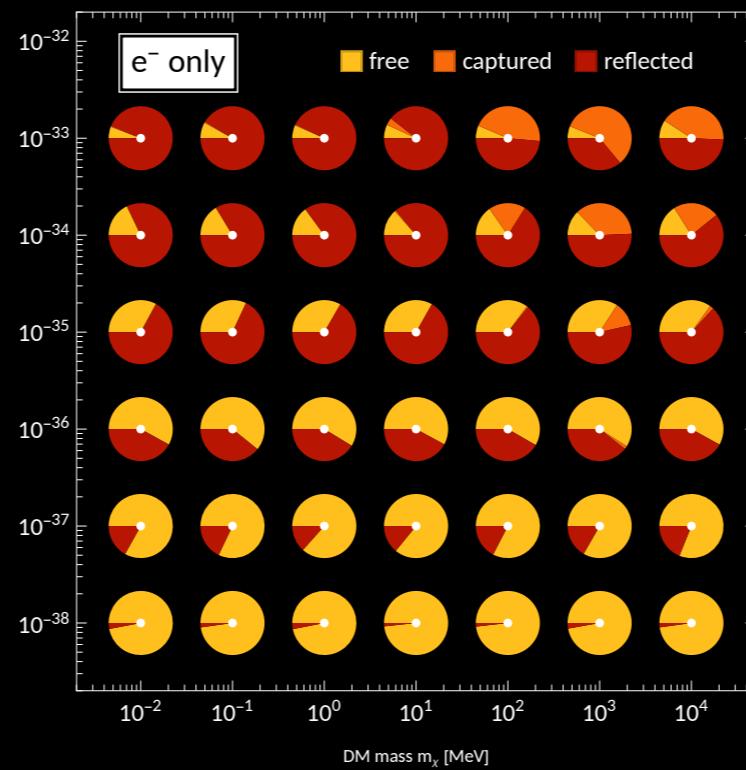
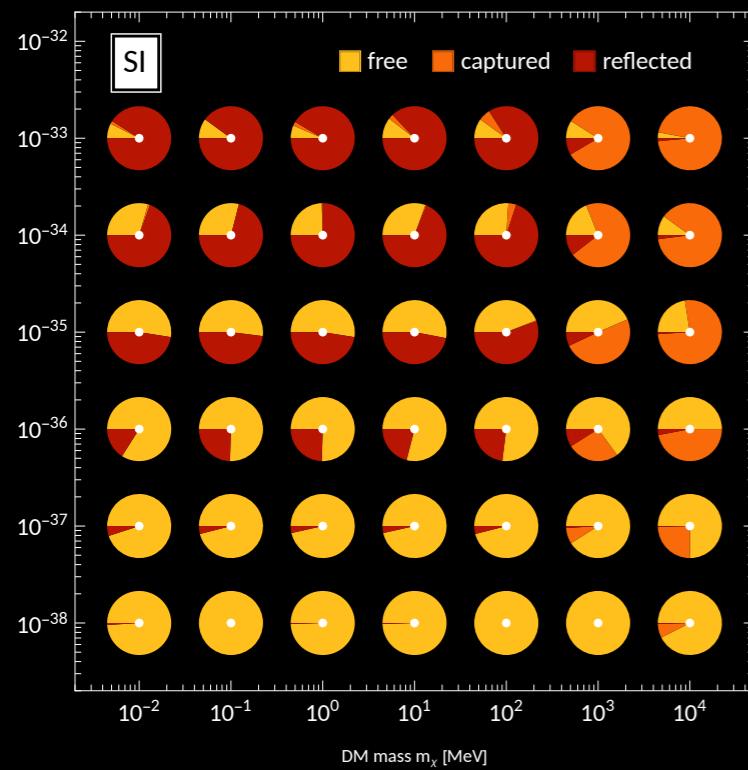
A. Serenelli et al., *Astrophys. J.* 705 (2009), L123-L127



- Mass Function  $M(r)$
- Temperature  $T(r)$

- Nuclear composition
- Number densities

# DM Gravitational capture rate



# Runge-Kutta-Fehlberg (RK45)

E. Fehlberg, NASA technical report, NASA, 1969

Adaptive method for the numerical solution of 1st order ODEs.

$$\dot{y} = f(t, y), \quad y(t_0) = y_0,$$

Requires the same number of function evaluation of RK6, but is rather a combination of RK4,

$$y_{k+1} = y_k + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4101}k_4 - \frac{1}{5}k_5,$$

and RK5,

$$\tilde{y}_{k+1} = y_k + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6.$$

The comparison of both yields an estimate of the error, and allows to choose the step size adaptively.

$$\Delta t_{k+1} = 0.84 \left( \frac{\epsilon}{|y_{k+1} - \tilde{y}_{k+1}|} \right)^{1/4} \Delta t_k$$