+

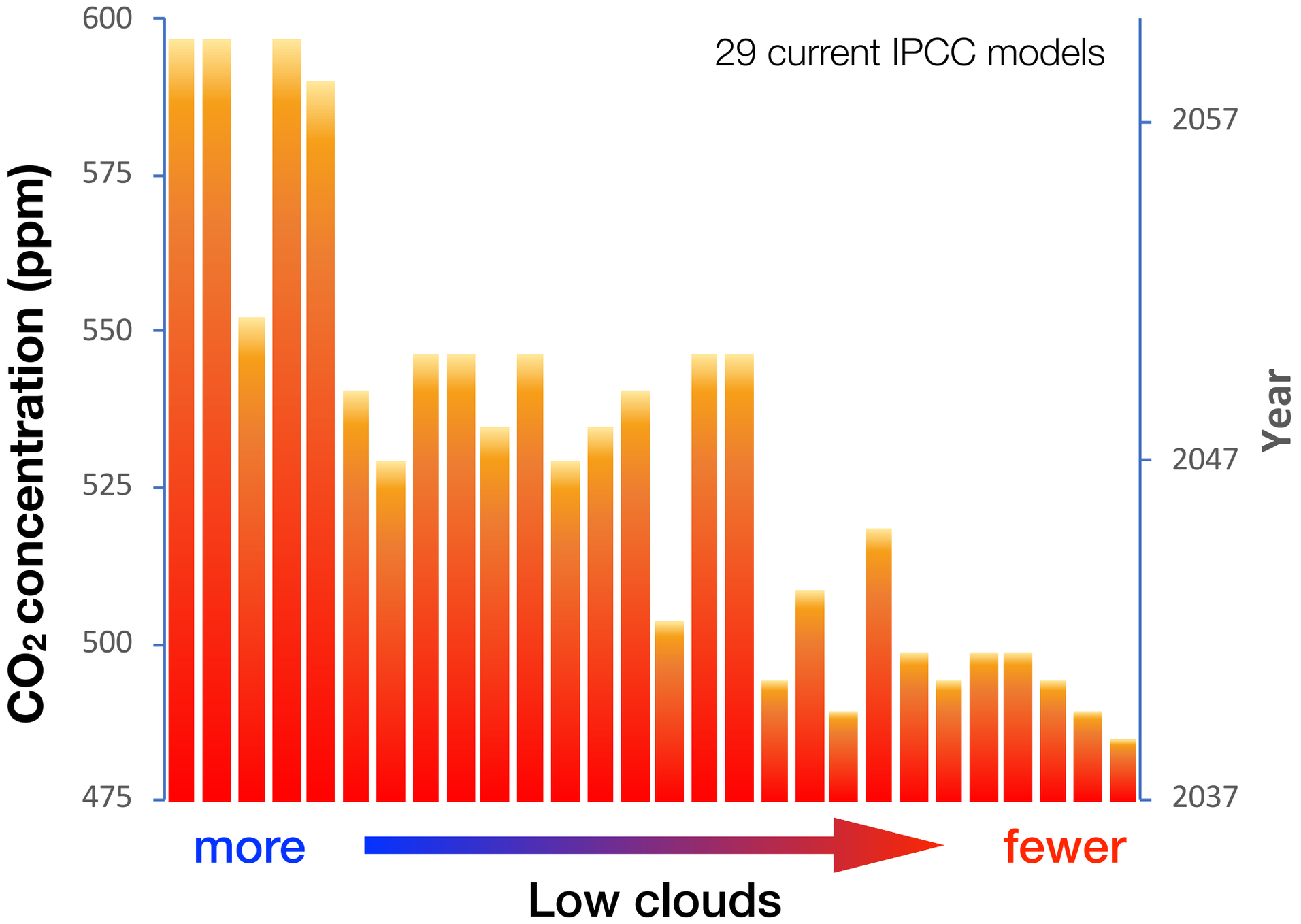
Expressiveness and Performance: Adventures in Climate Modelling using a Dynamic Language

C
California Institute of Technology

Limitations of Current Models

Climate predictions are uncertain

CO_2 concentration/time to 2°C warming threshold



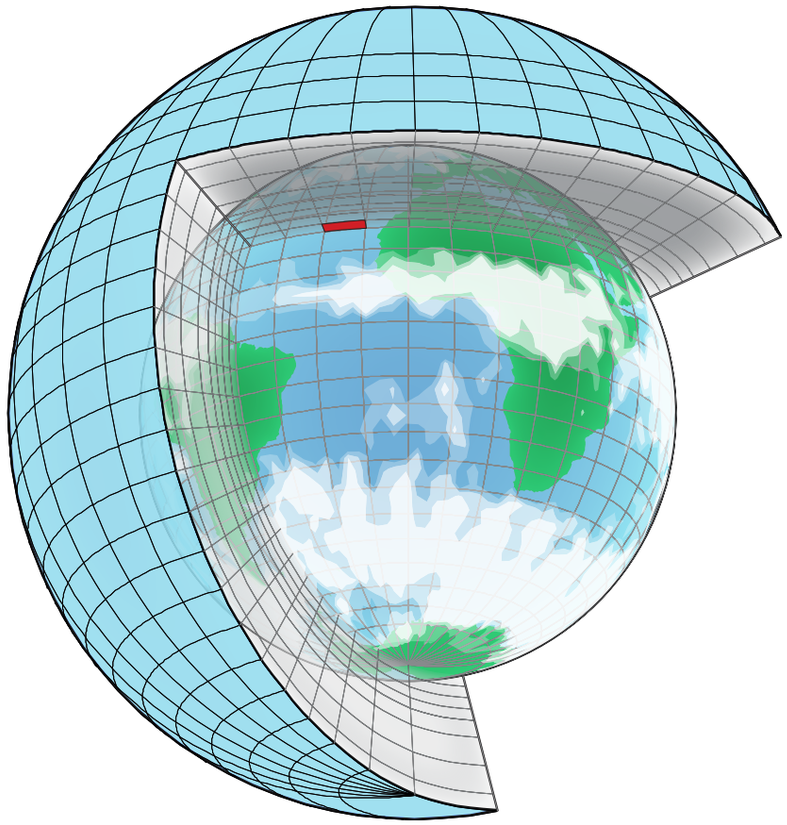
Schneider et al., *Nature Climate Change* 2017

More accurate climate projections with quantified uncertainties would enable…

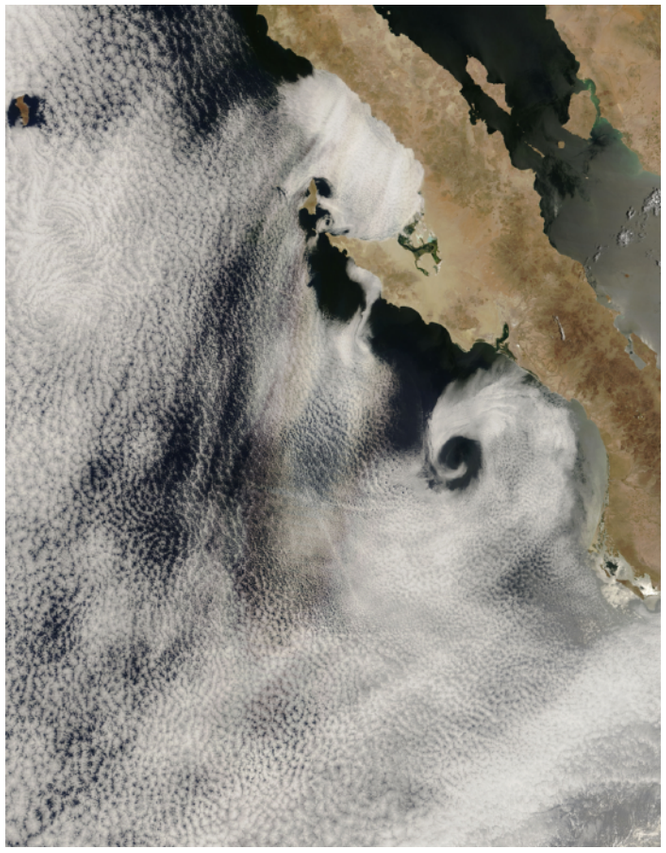
- Data-driven decisions about infrastructure planning, e.g.,
  - How high a sea wall should New York City build to protect itself against storm surges in 2050?
  - What water management infrastructure is needed to ensure food and water security in sub-Saharan Africa?
- Rational resource allocation for climate change adaptation: costs estimated to reach >$200B annually by 2050 (UNEP 2016)

Cumulative socioeconomic value of more accurate predictions estimated to lie in the trillions of USD (Hope 2015)

Subgrid-scale processes dominate uncertainty

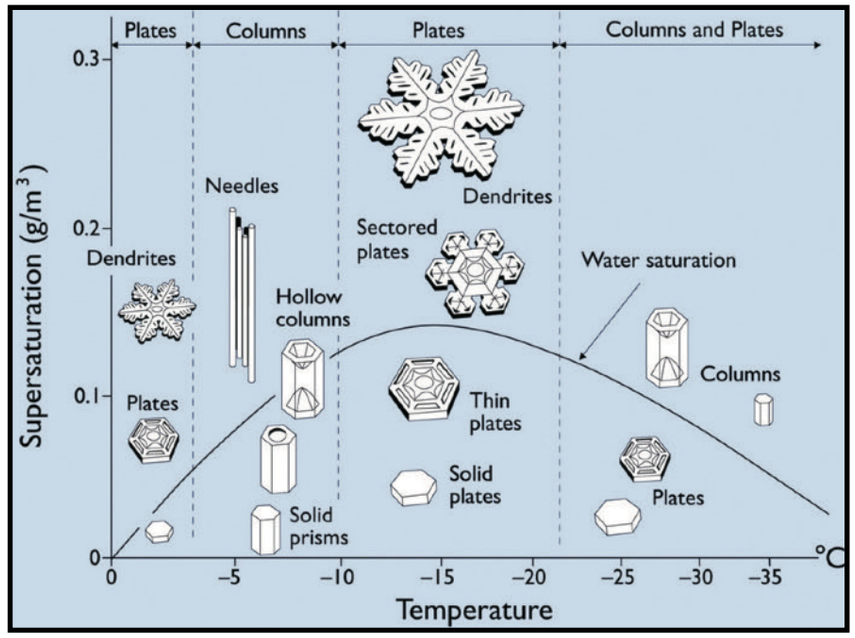


Global model:
~10-50 km resolution



NASA MODIS

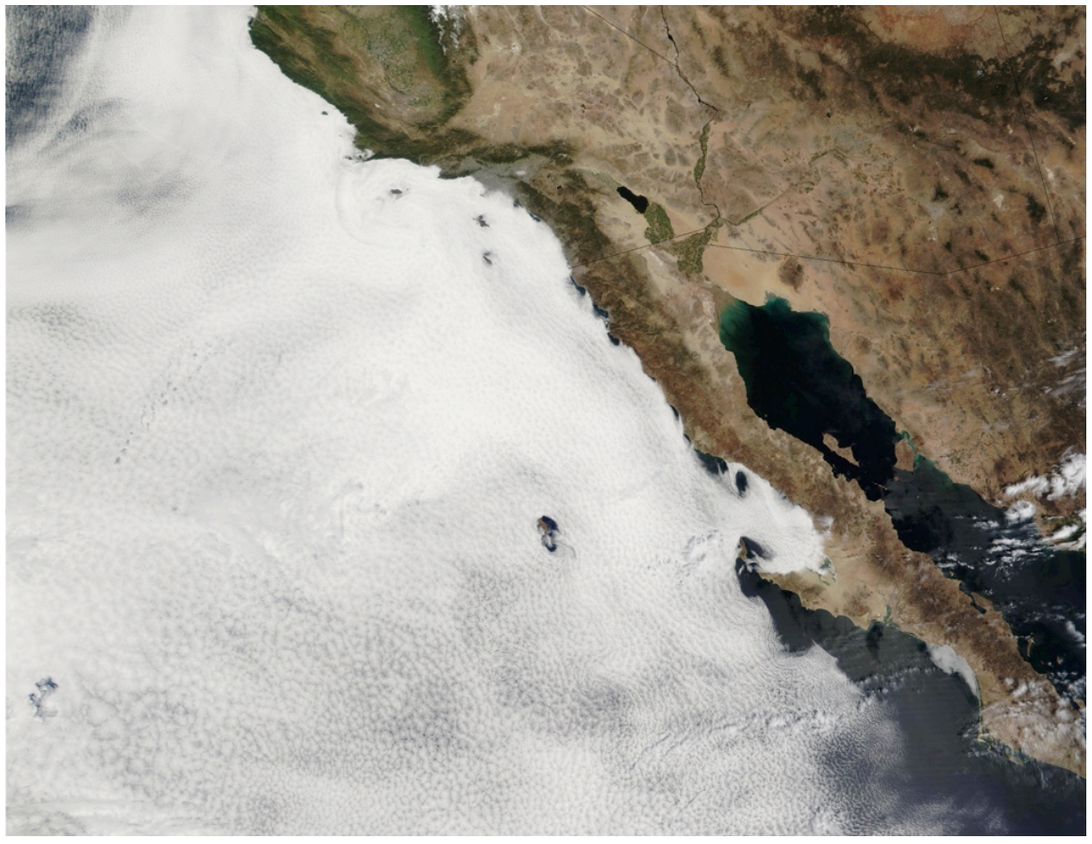
Cloud scales:
~10-100 m

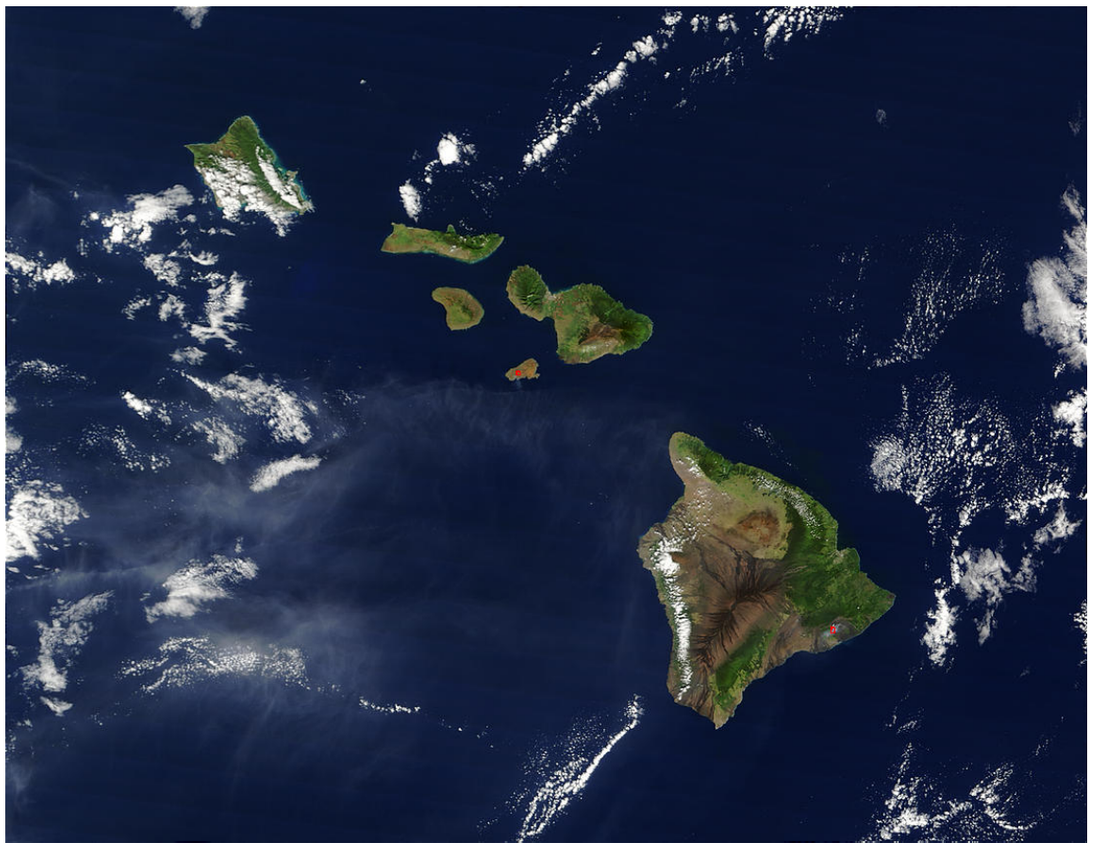


Microphysics:
<1m

e.g., clouds and turbulence represented in ad-hoc fashion (not data-driven)

Low clouds dominate uncertainty in projections





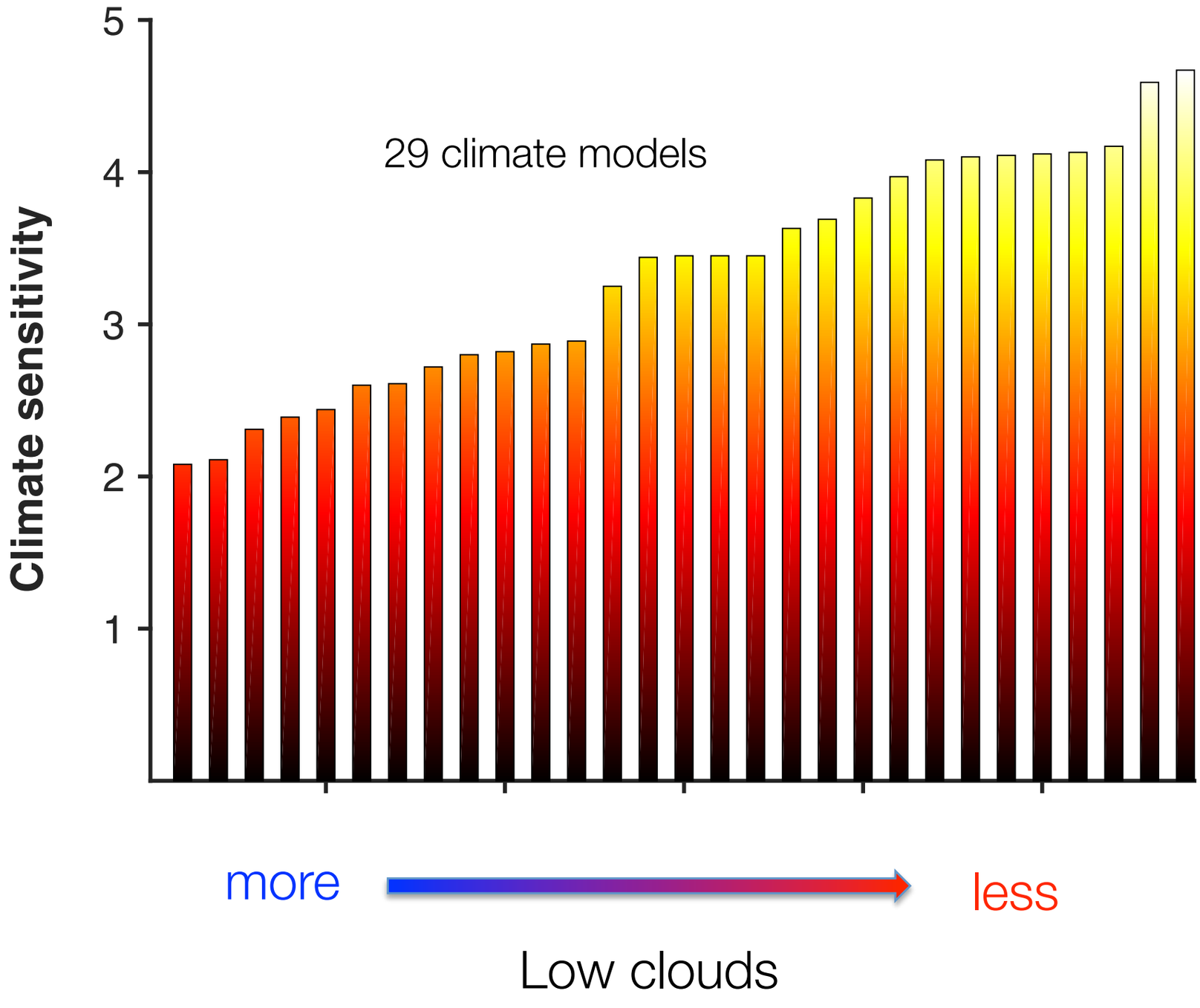
http://eoimages.gsfc.nasa.gov

Stratocumulus: colder

Cumulus: warmer

We don't know if we will get more low clouds (damped global warming), or fewer low clouds (amplified warming)

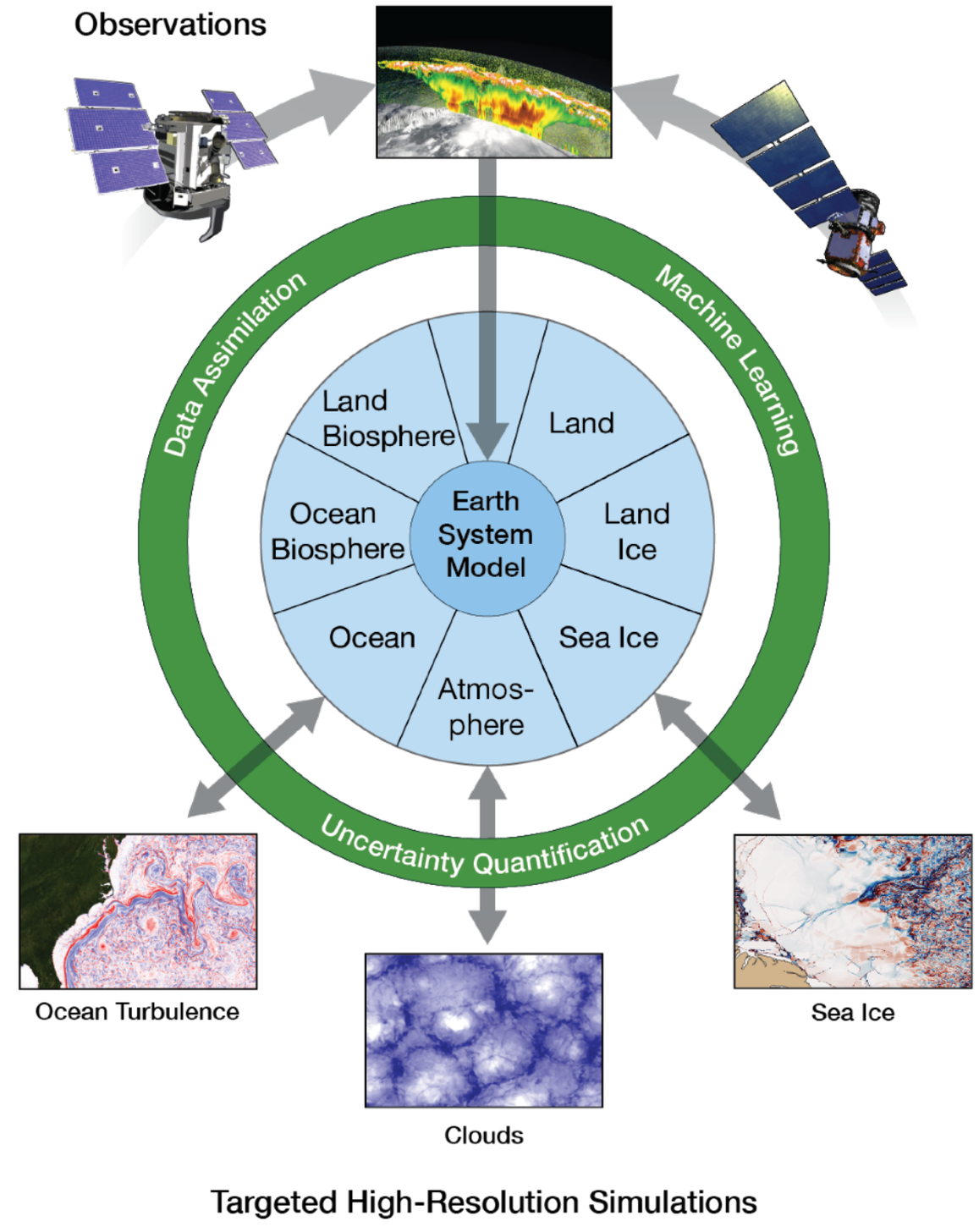
In some models, low clouds dampen warming; in some, they amplify warming



Climate Modeling Alliance

- A coalition of scientists, engineers, and applied mathematicians from **Caltech**, **MIT**, the **Naval Postgraduate School**, and the **Jet Propulsion Laboratory**.

- We are building **Climate Machine**, an Earth system model that automatically learns from diverse data sources to produce accurate climate predictions with quantified uncertainties.



Climate Machine: technical aims

- Support CPUs and GPUs using a common codebase
- Limit number of languages (ideally one)
- Be accessible and extensible by mixture of users
- Various options were evaluated
  - Initial pilot of Julia to demonstrate feasibility for GPUs
- Overall decision has worked out very well
  - Embraced by researchers
  - Overcome initial skepticism from funders

Julia for CliMA

- Previous use as part of the Celeste project on Cori (1.5 PF on KNL)
- Initial pilot by NPS & MIT to demonstrate feasibility for GPUs
- Overall decision has worked out very well
  - Embraced by researchers (typically coming from Python or Fortran)
    - Contribute directly to production code
  - Allowed us to make fairly rapid progress
  - Initial concern from funders, now mostly satisfied
- Single monolithic repository (https://github.com/CliMA/ClimateMachine.jl)
  - Moving components to packages as codebase & interfaces mature

Julia GPU support

- CUDA.jl is the primary package for Nvidia GPUs
  - LLVM NVPTX backend + hooks into the Julia compiler
  - High-level `CuArray` interface
    - Generates kernels for broadcasting or reductions
  - Exposes various CUDA toolkit functions
    - Sparse arrays, FFTs, linear algebra
  - Integrates with existing Julia APIs
  - Low-level interface for writing and launching CUDA kernels
- AMDGPU.jl is under active development
  - Shares common infrastructure (GPUArrays.jl)

```
# multiply each element by 2
A = map(x -> 2*x, A)

# "dot" broadcasting (in-place)
A .= 2 .* A
```

```
# solve Ax = b via Cholesky
x = cholesky(A) \ b
```

```
function mul2!(A)
    i = threadIdx().x
    A[i] = 2*A[i]
    return nothing
end
@cuda threads=length(A) mul2!(A)
```

Domain-Specific Languages in Julia

- Macros make it very easy to write DSLs
  - Built-in parser
  - Generator and generated code in one language
  - String macros if Julia syntax isn't flexible enough
- DSLs have downsides
  - Obscure what the code is doing
  - Non-standard language semantics
  - Can be difficult to debug
  - Easy to have ill-defined behavior
  - Users might want to escape DSL constraints
  - Work to build and maintain

```
# StatsModels.jl
f = @formula(y ~ 1 + a + b + c + b&c)


# ModelingToolkit.jl
@parameters t σ ρ β
@variables x(t) y(t) z(t)
D = Differential(t)

eqs = [D(D(x)) ~ σ*(y-x),
       D(y) ~ x*(ρ-z)-y,
       D(z) ~ x*y - β*z]
```

```
# Pumas.jl
@model begin
    @param begin
        CL ∈ RealDomain(lower=0.001, upper=10.0)
        Vc ∈ RealDomain(lower=0.001, upper=10.0)
        ω  ∈ RealDomain(lower=0.001, upper=10.0)
    end

    @dynamics begin
        Central' = -CL/Vc*Central
    end

    @derived begin
        cp := @. Central/Vc
        dv ~ @. Normal(cp, ω^2)
    end
end


# APL.jl
apl"{ (⍵=(1]α) +∘ (2]α)) ] ⍵}"(x,M)
```

ClimateMachine.jl dynamical core (current)

- Discontinuous Galerkin (DG) discretization
- PDE in conservative form

$$\frac{\partial \boldsymbol{Y}}{\partial t} = -\nabla \cdot \boldsymbol{\mathcal{F}} + \boldsymbol{\mathcal{S}}(\boldsymbol{Y})$$

  - Users supply arbitrary functions for flux, source, diffusive flux, etc
  - Operator volume/face kernels written in KernelAbstractions.jl
- Overlaps computation & communication
  - Distributed via MPI.jl
  - Exchange boundary faces during volume & internal face integrals
- Efficient, but somewhat inflexible

Evolution of Climate Machine numeric kernels

1. Separate CPU and GPU (CUDA.jl) kernels
    - Limited code re-use
2. GPUifyLoops.jl
    - Annotate for loops with thread indices
    - “scratch” memory
        - Local to GPU thread
        - Add implicit dimensions on CPU
    - Annotations for loop unrolling
    - Replaces Julia math functions (e.g. `log`) with CUDA intrinsics via Cassette.jl

```
# GPUifyLoops.jl
function mul2!(A)
    @loop for i in (1:size(A,1);
                    threadIdx().x)
        A[i] = 2*A[i]
    end
    @synchronize
end

# using CPU: acts like a regular function
mul2!(A)

# using GPU: launched as a CUDA kernel
@launch CUDA() mul2!(G, threads=length(A))
```

3. KernelAbstractions.jl

Valentin Churavy (MIT Julia Lab)

- Unified programming model
  - Similar to OpenCL / SYCL
  - CUDA, AMD, CPU multithreading
- Fully asynchronous
- Macros for language annotations
  - Non-aliased input arguments
  - Shared and private memory
  - Uniform variables across workgroups
  - Synchronization points
- Primarily “GPU code which runs on CPU”
  - Very useful for debugging

```
@kernel function gradient!(DfX, X, f,
        D::SMatrix{Nq,Nq,FT}) where {Nq,FT}

    fX = @localmem Float64 (Nq, Nq)
    e = @index(Group, Linear)
    i, j = @index(Local, NTuple)

    fX[i,j] = f(X[i,j,e])

    @synchronize

    G1 = G2 = zero(FT)
    @unroll for k = 1:Nq
        G1 += D[i,k]*fX[k,j]
        G2 += D[j,k]*fX[i,k]
    end

    DfX[1,i,j,e] = G1
    DfX[2,i,j,e] = G2

    return nothing
end

n = size(X,3)
Nq = size(D,1)

# CPU
cpu_gradient! = gradient!(CPU(), (Nq,Nq))
event = cpu_gradient!(DfX, X, f, D, ndrange=(Nq,Nq,n))
# GPU
cuda_gradient! = gradient!(CUDADevice(), (Nq,Nq))
event = cuda_gradient!(DfX, X, f, D, ndrange=(Nq,Nq,n))

wait(event)
```

Thanks

SCHMIDT **FUTURES**

MOUNTAIN PHILANTHROPIES

CHARLES TRIMBLE

RONALD AND MAXINE LINDE CLIMATE CHALLENGE

PAUL G. ALLEN FAMILY FOUNDATION

NSF

Google Cloud