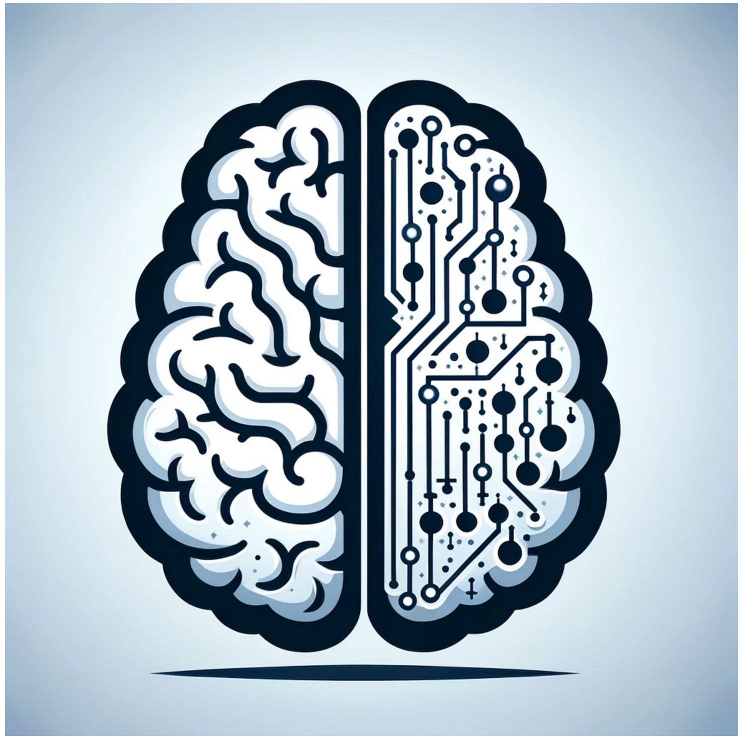
EN 601.473/601.673: Cognitive Artificial Intelligence (CogAI)



Lecture 12: SMC in Gen

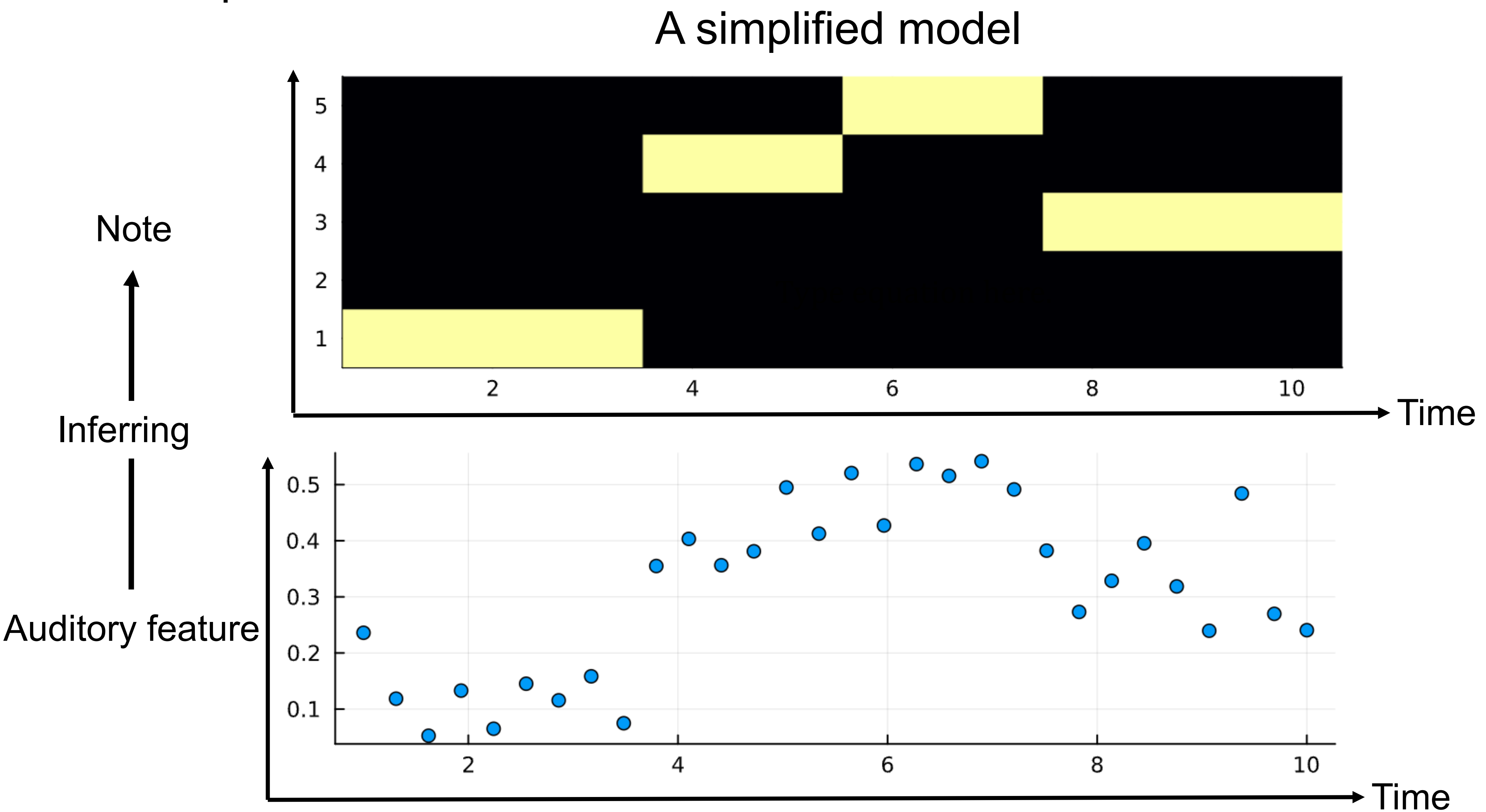
Tianmin Shu

SMC

- Step 1: Initialization
  - Set $t = 1$
  - For $i \in \{1, \cdots, K\}$, sample $h_1^{(i)} \sim q(h_1)$ initial proposals
  - Weight $w_1^{(i)} = \frac{p\left(x_1|h_1^{(i)}\right)p(h_1^{(i)})}{q(h_1^{(i)})}$
  - Normalize all weights → particles $\left\{\left(h_1^{(i)}, w_1^{(i)}\right)\right\}_{i=1}^{K}$
- Step 2: Resampling at step t
  - If ESS < threshold, resample the particles, and set $w_t^{(i)} = \frac{1}{K}$ (importance resampling)
- Step 3: Sampling at step $t + 1$
  - Set $t = t + 1$; particles at previous step $\left\{\left(h_{1:t-1}^{(i)}, w_{t-1}^{(i)}\right)\right\}_{i=1}^{K}$
  - For $i \in \{1, \cdots, K\}$, sample $h_t^{(i)} \sim p\left(h_t|h_{t-1}^{(i)}\right)$ conditional distribution
  - Weight $w_t^{(i)} = w_{t-1}^{(i)} p\left(x_t \middle| h_t^{(i)}\right)$ reweighted by the current likelihood
  - Normalize all weights → particles $\left\{\left(h_{1:t}^{(i)}, w_t^{(i)}\right)\right\}_{i=1}^{K}$

SMC in Gen (particle filtering)

- Example 1: Music notes



Generative model – transitions between notes $p(h_t|h_{t-1})$

- 5 musical notes, a composition is defined by a set of stochastic transitions between notes
- With a probability of 0.6, we stay on the current note; with a probability of 0.1, we transition to one of the new notes

```julia
function transition_prob(a, b)
    a == b ? 0.6 : 0.1
end

# setup transition probabilities
function transition_prob_matrix()
    note_transitions = Array{Float64}(undef, 5, 5)
    for i=1:5
        [note_transitions[i, j] = transition_prob(i,j) for j=1:5]
    end
    return note_transitions
end

# make this variable global (makes some of the coding more straightforward)
global note_transitions = transition_prob_matrix()
```

Generative model – notes & auditory features $p(x_{1:t}, h_{1:t})$

```julia
@gen function musical_notes(K::Int)
    # sample an initial "note"
    current_note = {:notes => 1 => :θ} ~ uniform_discrete(1, 5)

    # "play" it: projecting it to "auditory features" of length 3
    note_play_length = 3
    mu = current_note * 0.1
    {:data => 1 => :y} ~ broadcasted_normal(repeat([mu], note_play_length), repeat([0.05], note_play_length))

    # keep going for K notes
    for k=1:K
        current_note = {:notes => k+1 => :θ} ~ categorical(note_transitions[current_note, :])
        mu = current_note * 0.1
        {:data => k+1 => :y} ~ broadcasted_normal(repeat([mu], note_play_length), repeat([0.05], note_play_length))
    end
end
```

See jupyter notebook

Particle filter

```
function particle_filter(num_particles::Int, zs::Matrix{Float64}, num_samples::Int)

    #initital observation
    init_obs = Gen.choicemap((:data => 1 => :y, zs[1, :]))

    # initialize the particle filter (Step 0) --
    # sampling from the prior and weighting by the likelihood
    state = Gen.initialize_particle_filter(musical_notes, (0,), init_obs, num_particles)

    for k=1:size(zs)[1]-1
        # Evolve and resample (Step 1a, 1b)
        maybe_resample!(state, ess_threshold=num_particles/2)
        # load observations of this time step
        obs = Gen.choicemap((:data => k+1 => :y, zs[k+1,:]))
        # Re-weight by the likelihood (Step 2)
        Gen.particle_filter_step!(state, (k,), (UnknownChange(),), obs)
    end

    # return a sample of unweighted traces from the weighted collection
    return Gen.sample_unweighted_traces(state, num_samples)
end
```

Inference results

Run particle filtering conditioned on data

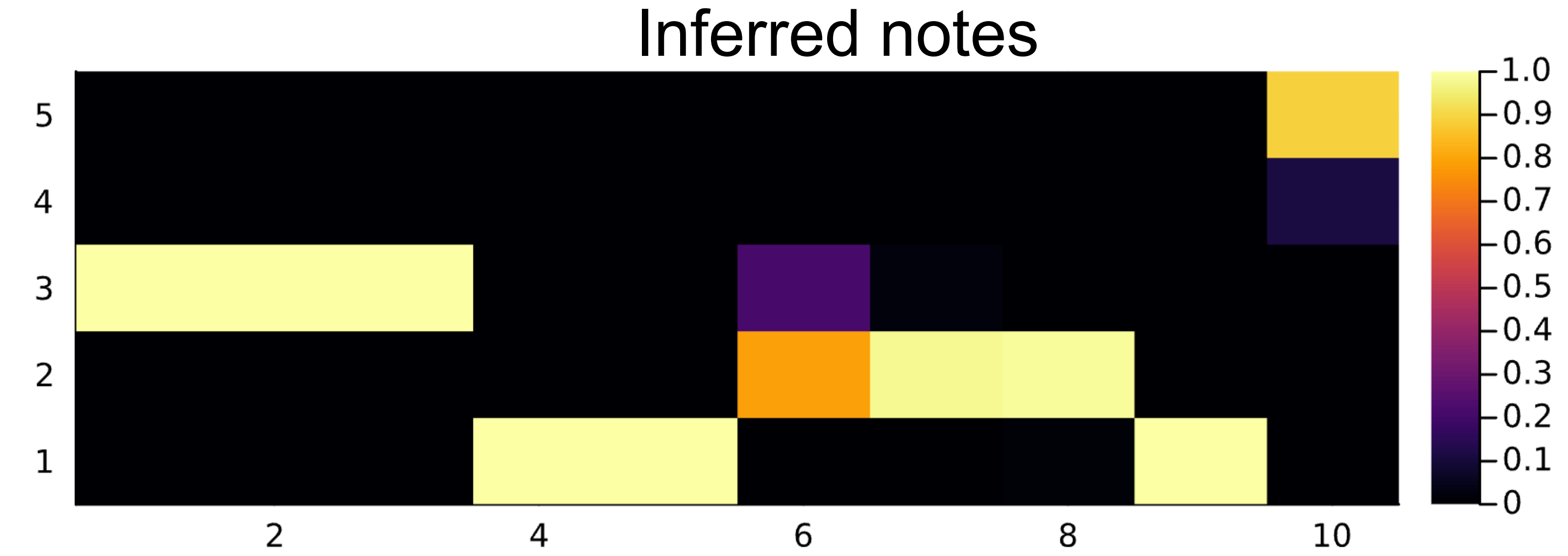
```
# load observations (sensory features)
zs = readdlm("notes_observe.txt", ',')
# run particle filter with 1000 particles,
# and return 100 unweighted posterior samples
pf_traces = particle_filter(1000, zs, 100);
```

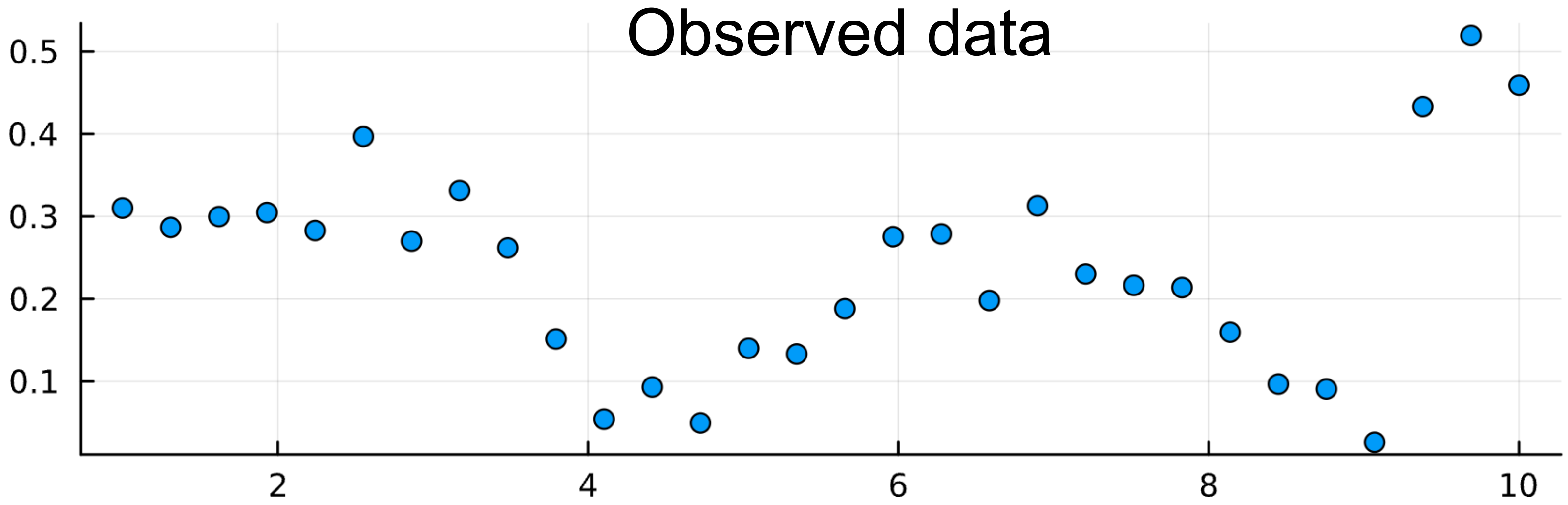
Hypothesis averaging by aggregating all particles

```
scores = Vector{Float64}(undef, 100)
inferred_notes = zeros(Int, 100, 10)
aggregate_notes_matrix = zeros(Float64, 5, 10)
for i=1:100
    tr = pf_traces[i]
    scores[i] = get_score(tr)
    for k = 1:size(zs)[1]
        inferred_notes[i, k] = tr[:notes => k => :0]
        aggregate_notes_matrix[inferred_notes[i, k], k] += 1.0
    end
end

aggregate_notes_matrix = aggregate_notes_matrix ./ 100
```

Inference results





For loop can be slow

```
function particle_filter(num_particles::Int, zs::Matrix{Float64}, num_samples::Int)

    #initital observation
    init_obs = Gen.choicemap((:data => 1 => :y, zs[1, :]))

    # initialize the particle filter (Step 0) --
    # sampling from the prior and weighting by the likelihood
    state = Gen.initialize_particle_filter(musical_notes, (0,), init_obs, num_particles)

    for k=1:size(zs)[1]-1
        # Evolve and resample (Step 1a, 1b)
        maybe_resample!(state, ess_threshold=num_particles/2)
        # load observations of this time step
        obs = Gen.choicemap((:data => k+1 => :y, zs[k+1,:]))
        # Re-weight by the likelihood (Step 2)
        Gen.particle_filter_step!(state, (k,), (UnknownChange(),), obs)
    end

    # return a sample of unweighted traces from the weighted collection
    return Gen.sample_unweighted_traces(state, num_samples)
end
```

Unfold combinator for accelerated inference

- Key idea: write a kernel K(h_{t-1}, h_t) and unfold the temporal sequence by applying the kernel at repeatedly instead of using a for loop
- In Gen: implemented by *static* modeling language, a variant of the built-in modeling language
- Better inference performance (more inference operations per second and less memory consumption), than the full built-in modeling language

Unfold generative model

```
@gen (static) function musical_notes_kernel(k::Int, prev_note)
    # draw the note
    current_note = {:θ} ~ categorical(note_transitions[prev_note, :])

    # how long it takes to play a note
    note_play_length = 3
    # draw the sensory features
    mu = current_note * 0.1
    {:y} ~ broadcasted_normal(repeat([mu], note_play_length), repeat([0.05], note_play_length))

    #return the current note for recursion
    return current_note
end
```

```
@gen (static) function unfold_musical_notes(K::Int)
    # sample an initial "note"
    init_note = uniform_discrete(1, 5)

    # call the temporal kernel and unfold it for K time steps
    music ~ Unfold(musical_notes_kernel)(K, init_note)

end

trace = simulate(unfold_musical_notes, (10,))
```

Particle filtering with unfold generative model

```
function particle_filter_with_unfold(num_particles::Int, zs::Matrix{Float64}, num_samples::Int)

    #initital observation
    init_obs = Gen.choicemap((:music => 0 => :y, zs[1, :]))

    # initialize the particle filter (Step 0) --
    # sampling from the prior and weighting by the likelihood
    state = Gen.initialize_particle_filter(unfold_musical_notes, (0,), init_obs, num_particles)

    for k=1:size(zs)[1]
        # Evolve and resample (Step 1a, 1b)
        maybe_resample!(state, ess_threshold=num_particles/2)
        # load observations of this time step
        obs = Gen.choicemap((:music => k => :y, zs[k,:]))
        # Re-weight by the likelihood (Step 2)
        Gen.particle_filter_step!(state, (k,), (UnknownChange(),), obs)
    end

    # return a sample of unweighted traces from the weighted collection
    return Gen.sample_unweighted_traces(state, num_samples)
end
```

Non-unfold generative model vs unfold generative model

10-30 data points

Non-unfold

1.061129 seconds	(7.95 M allocations:	505.549 MiB,
1.132377 seconds	(9.57 M allocations:	608.880 MiB,
1.772438 seconds	(11.34 M allocations:	720.202 MiB,
1.764492 seconds	(13.24 M allocations:	840.005 MiB,
1.781908 seconds	(15.28 M allocations:	967.801 MiB,
2.391841 seconds	(17.46 M allocations:	1.078 GiB,
2.484812 seconds	(19.77 M allocations:	1.219 GiB,
2.714673 seconds	(22.23 M allocations:	1.368 GiB,
3.416379 seconds	(24.83 M allocations:	1.525 GiB,
3.586414 seconds	(27.56 M allocations:	1.690 GiB,
4.418381 seconds	(30.44 M allocations:	1.863 GiB,
4.243229 seconds	(33.45 M allocations:	2.044 GiB,
5.001303 seconds	(36.60 M allocations:	2.233 GiB,
5.060083 seconds	(39.89 M allocations:	2.430 GiB,
5.531252 seconds	(43.32 M allocations:	2.635 GiB,
6.064709 seconds	(46.89 M allocations:	2.848 GiB,
6.173191 seconds	(50.60 M allocations:	3.069 GiB,
6.401637 seconds	(54.45 M allocations:	3.298 GiB,
7.215783 seconds	(58.43 M allocations:	3.535 GiB,
8.304483 seconds	(62.56 M allocations:	3.781 GiB,
9.221072 seconds	(66.82 M allocations:	4.034 GiB,

Unfold

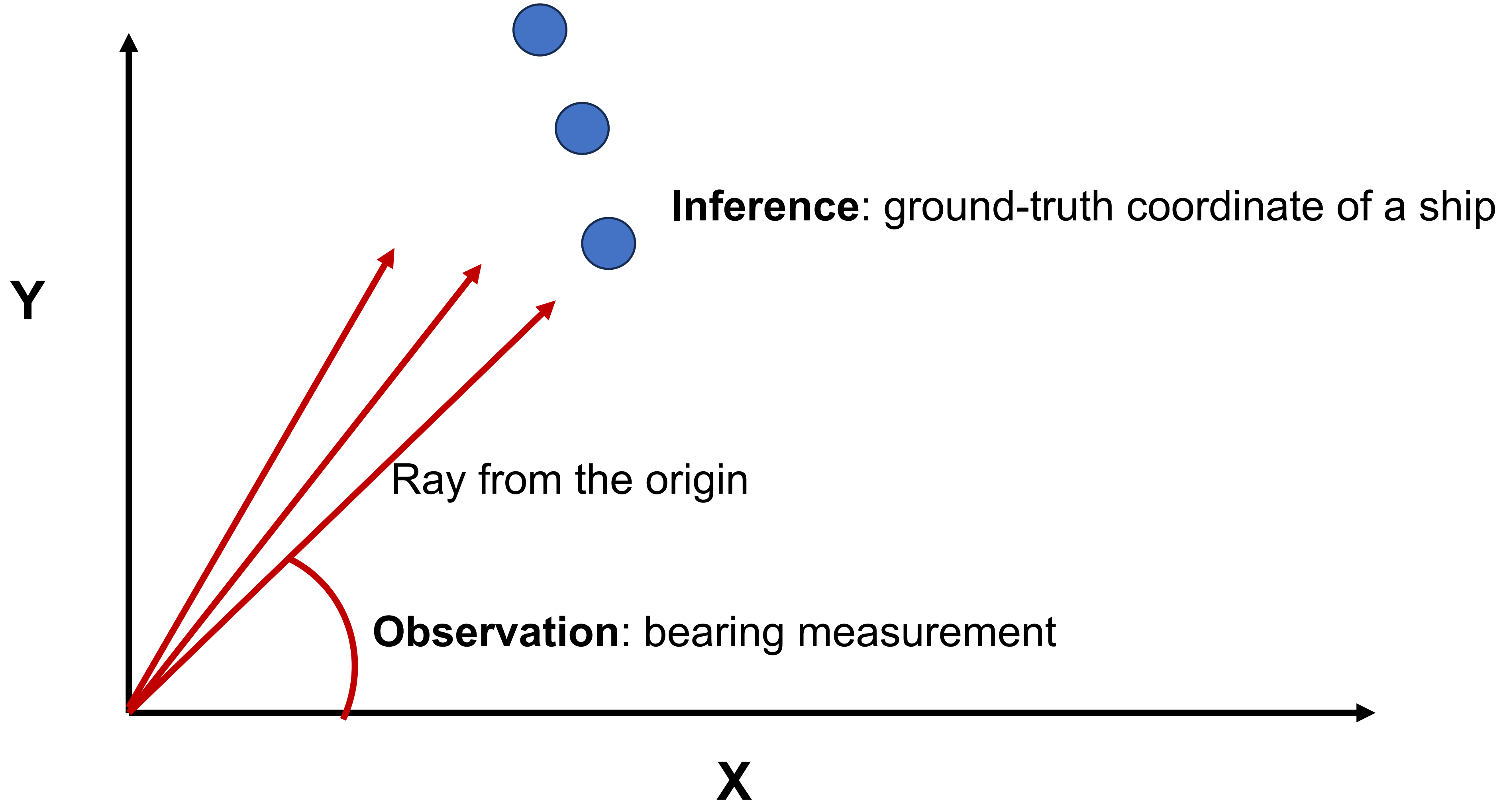
0.212010 seconds	(1.40 M allocations:	96.100 MiB,
0.181091 seconds	(1.53 M allocations:	106.052 MiB,
0.199166 seconds	(1.67 M allocations:	116.211 MiB,
0.178509 seconds	(1.80 M allocations:	126.429 MiB)
0.255829 seconds	(1.94 M allocations:	137.039 MiB,
0.268262 seconds	(2.07 M allocations:	147.540 MiB,
0.290940 seconds	(2.21 M allocations:	158.248 MiB,
0.252676 seconds	(2.34 M allocations:	169.132 MiB,
0.297692 seconds	(2.48 M allocations:	180.154 MiB,
0.294334 seconds	(2.62 M allocations:	191.027 MiB,
0.332496 seconds	(2.75 M allocations:	202.308 MiB,
0.355684 seconds	(2.89 M allocations:	213.657 MiB,
0.371295 seconds	(3.02 M allocations:	225.143 MiB,
0.433432 seconds	(3.16 M allocations:	236.643 MiB,
0.394131 seconds	(3.29 M allocations:	248.443 MiB,
0.421146 seconds	(3.43 M allocations:	260.311 MiB,
0.475033 seconds	(3.56 M allocations:	272.208 MiB,
0.478837 seconds	(3.70 M allocations:	284.389 MiB,
0.483925 seconds	(3.84 M allocations:	296.762 MiB,
0.496521 seconds	(3.97 M allocations:	309.040 MiB,
0.517246 seconds	(4.11 M allocations:	321.402 MiB,

Less time

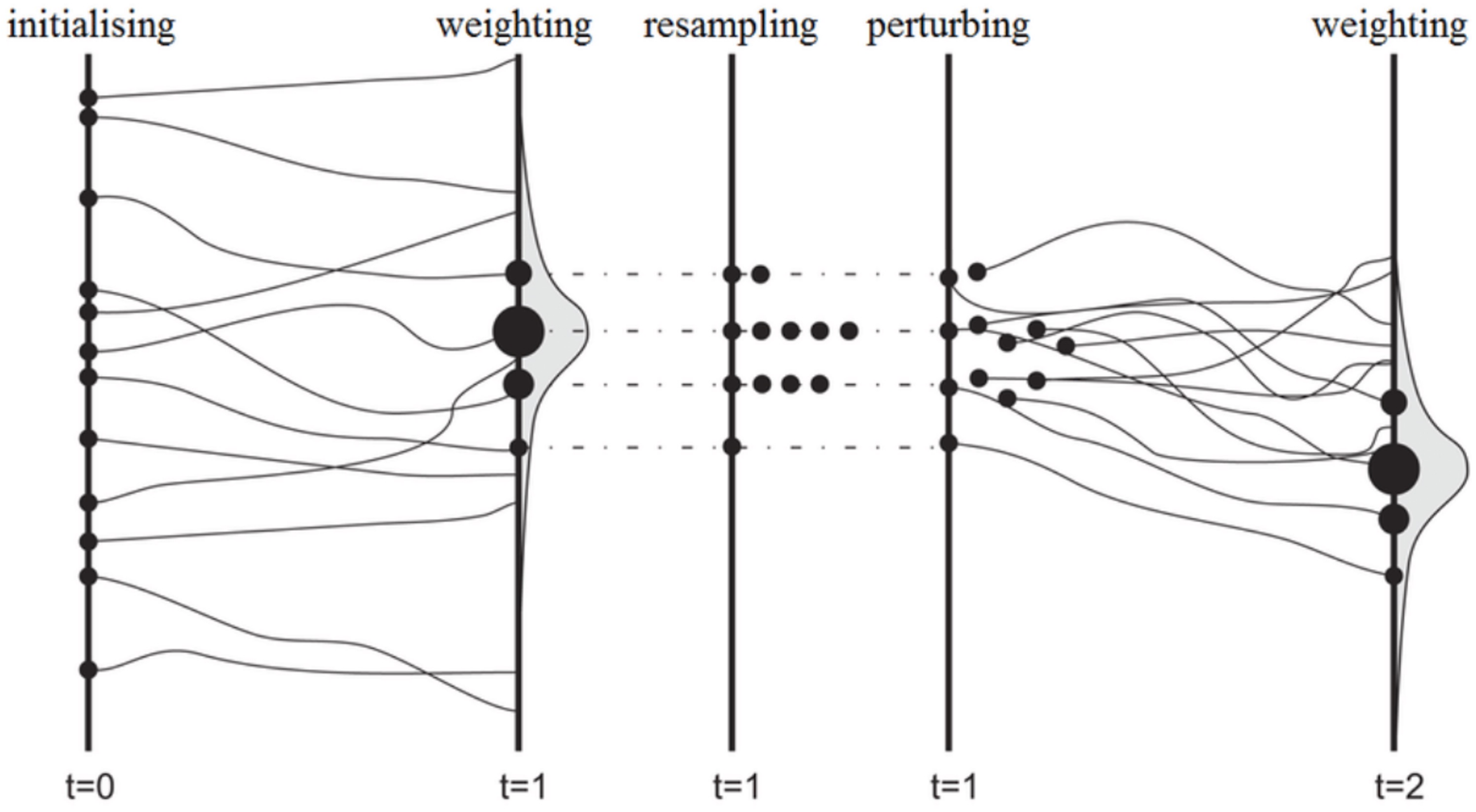
Less memory

SMC in Gen

- Example 2: Object tracking



Resample-move SMC



(sometimes called
MCMC rejuvenation)

MCMC Rejuvenation in SMC

- Step 1: Initialization
  - Set $t = 1$
  - For $i \in \{1, \cdots, K\}$, sample $h_1^{(i)} \sim q(h_1)$ initial proposals
  - Weight $w_1^{(i)} = \frac{p\left(x_1|h_1^{(i)}\right)p(h_1^{(i)})}{q(h_1^{(i)})}$
  - Normalize all weights → particles $\left\{\left(h_1^{(i)}, w_1^{(i)}\right)\right\}_{i=1}^{K}$
- Step 2: Resampling at step t
  - If ESS < threshold,
    - resample the particles, and set $w_t^{(i)} = \frac{1}{K}$ (importance resampling)
    - **Rejuvenate the particles**
- Step 3: Sampling at step $t + 1$
  - Set $t = t + 1$; particles at previous step $\left\{\left(h_{1:t-1}^{(i)}, w_{t-1}^{(i)}\right)\right\}_{i=1}^{K}$
  - For $i \in \{1, \cdots, K\}$, sample $h_t^{(i)} \sim p\left(h_t|h_{t-1}^{(i)}\right)$ conditional distribution
  - Weight $w_t^{(i)} = w_{t-1}^{(i)} p\left(x_t\middle|h_t^{(i)}\right)$ reweighted by the current likelihood
  - Normalize all weights → particles $\left\{\left(h_{1:t}^{(i)}, w_t^{(i)}\right)\right\}_{i=1}^{K}$

Sampling-based inference algorithms

- Rejection sampling
  - May waste a lot of samples
- Importance sampling
  - Only sample once and cannot update samples
- Markov chain Monte Carlo (MCMC)
  - Iteratively update samples
  - Better proposal distributions can speed up convergence
  - Proposals can be data–driven and can transition between hypotheses with different parameters (RJMCMC)
- Sequential Monte Carlo (SMC)
  - Inferring sequential latent variables
  - MCMC moves can rejuvenate particles

How to use Gen to implement sampling-based inference algorithms?