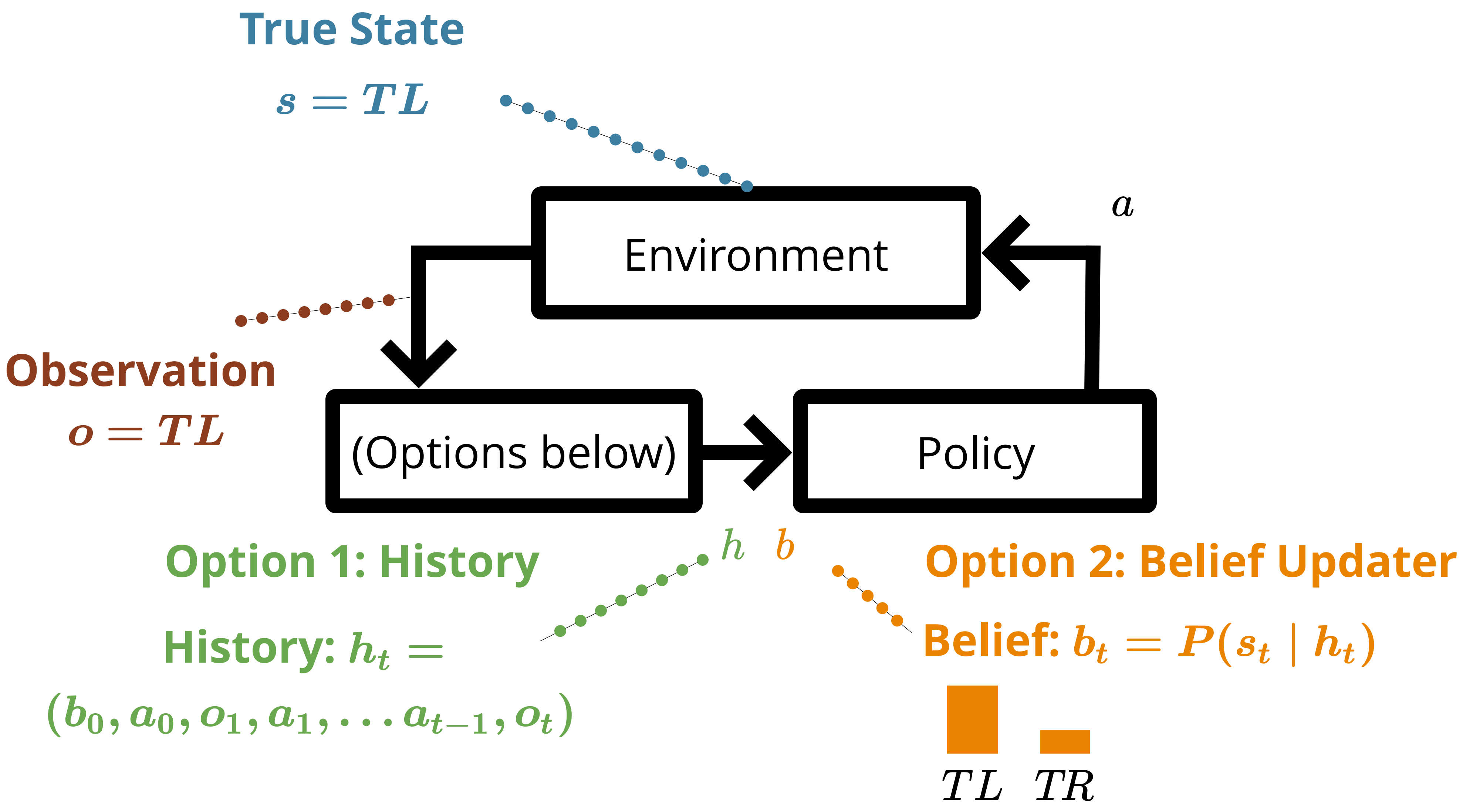
Particle Filters

POMDP Sense-Plan-Act Loop



True State

$s = TL$

Environment

a

Observation

$o = TL$

(Options below)

Policy

h b

Option 1: History

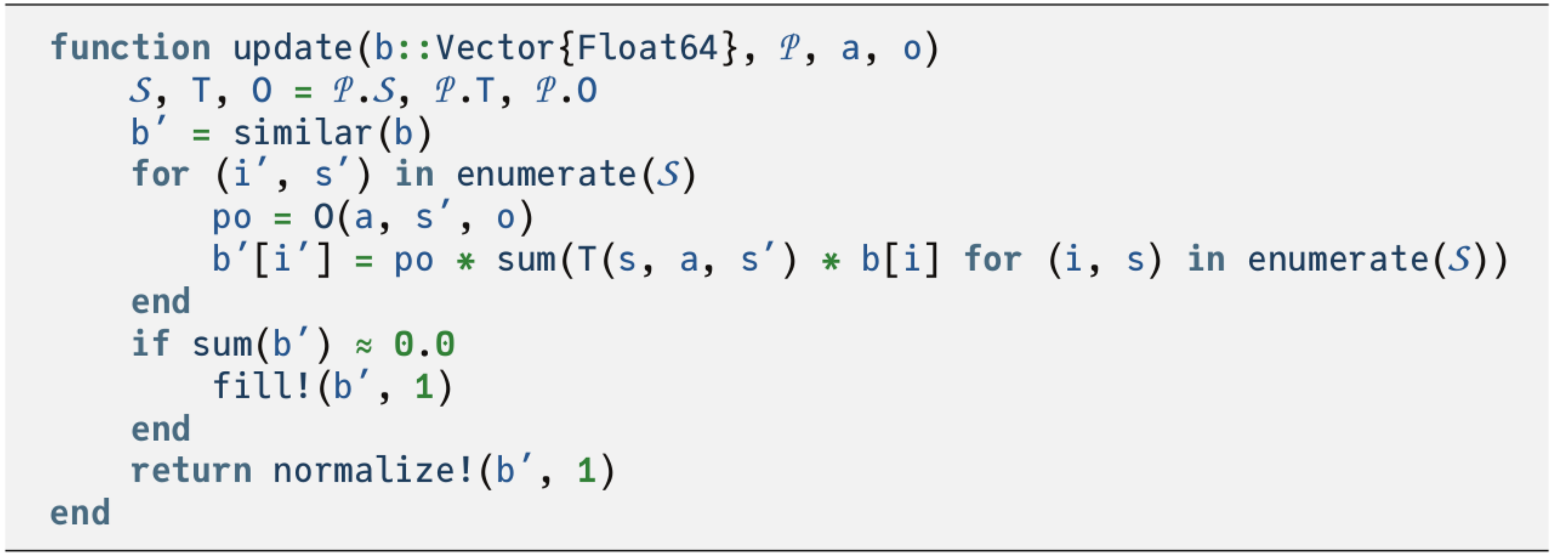
History: $h_t = (b_0, a_0, o_1, a_1, \ldots a_{t-1}, o_t)$

Option 2: Belief Updater

Belief: $b_t = P(s_t \mid h_t)$

TL TR

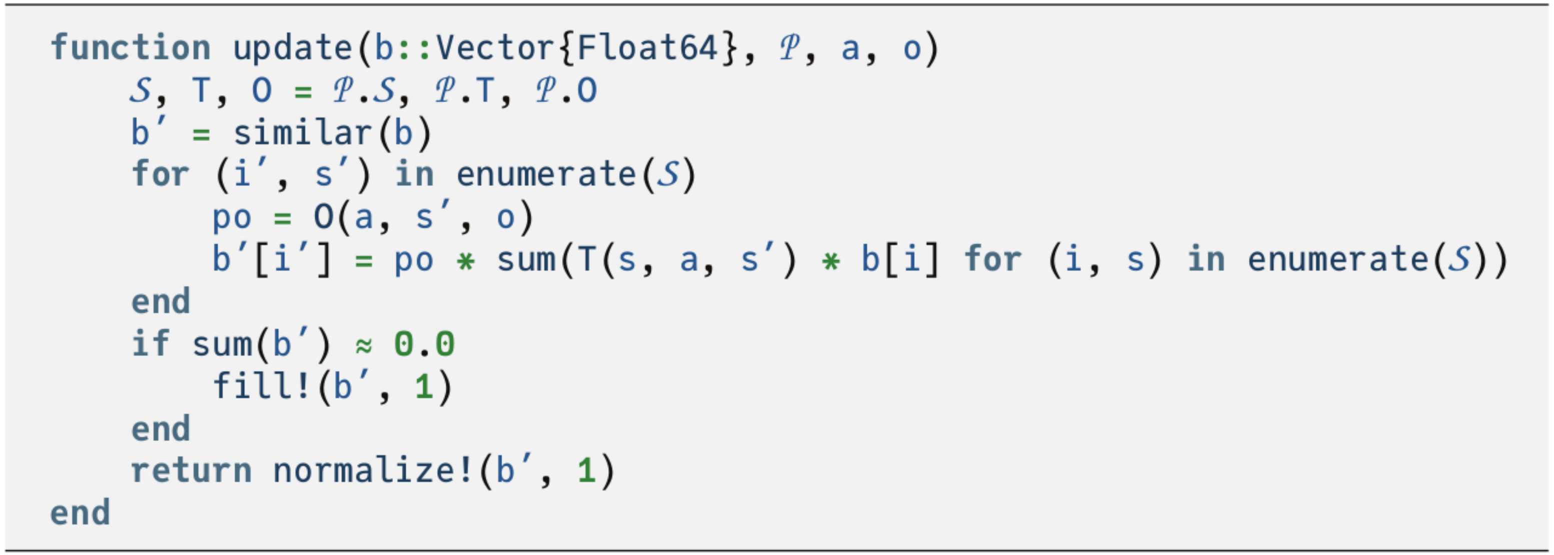
Review: Bayesian Filter



```
function update(b::Vector{Float64}, 𝒫, a, o)
    𝒮, T, O = 𝒫.𝒮, 𝒫.T, 𝒫.O
    b′ = similar(b)
    for (i′, s′) in enumerate(𝒮)
        po = O(a, s′, o)
        b′[i′] = po * sum(T(s, a, s′) * b[i] for (i, s) in enumerate(𝒮))
    end
    if sum(b′) ≈ 0.0
        fill!(b′, 1)
    end
    return normalize!(b′, 1)
end
```

Review: Bayesian Filter

$$b_t(s) = P(s_t = s \mid h_t)$$

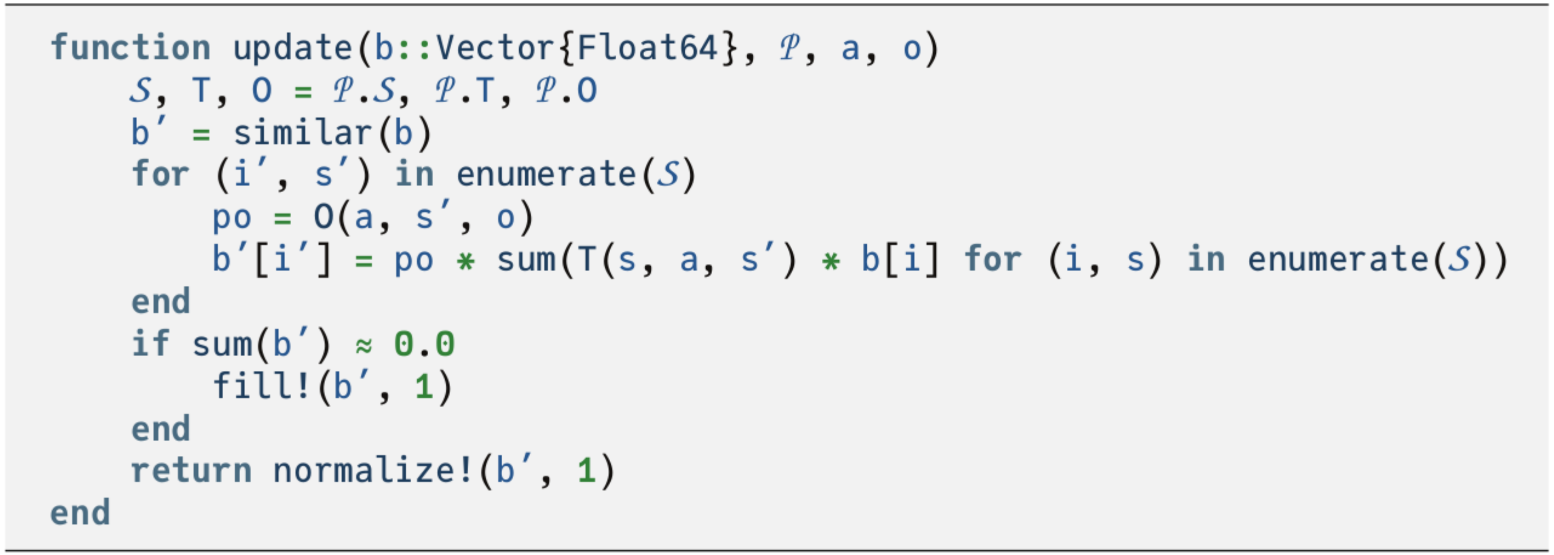


```
function update(b::Vector{Float64}, 𝒫, a, o)
    𝒮, T, O = 𝒫.𝒮, 𝒫.T, 𝒫.O
    b′ = similar(b)
    for (i′, s′) in enumerate(𝒮)
        po = O(a, s′, o)
        b′[i′] = po * sum(T(s, a, s′) * b[i] for (i, s) in enumerate(𝒮))
    end
    if sum(b′) ≈ 0.0
        fill!(b′, 1)
    end
    return normalize!(b′, 1)
end
```

Review: Bayesian Filter

$b_t(s) = P(s_t = s \mid h_t)$

$b' = \tau(b, a, o)$



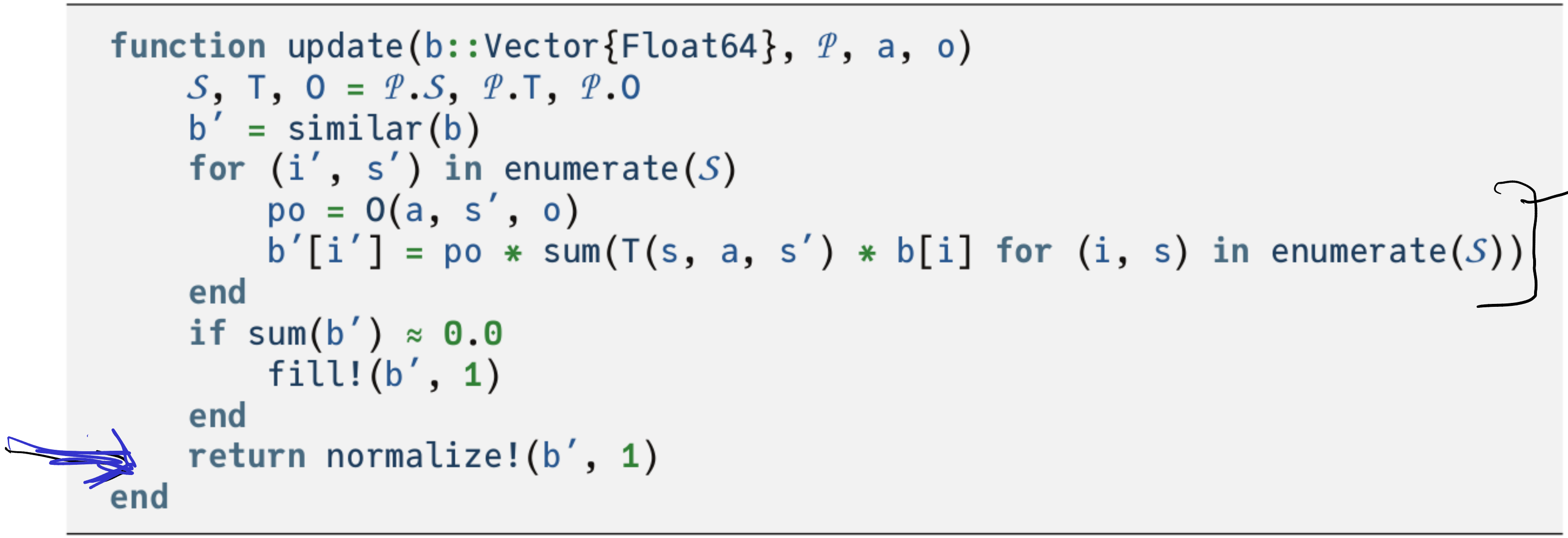
```
function update(b::Vector{Float64}, 𝒫, a, o)
    𝒮, T, O = 𝒫.𝒮, 𝒫.T, 𝒫.O
    b′ = similar(b)
    for (i′, s′) in enumerate(𝒮)
        po = O(a, s′, o)
        b′[i′] = po * sum(T(s, a, s′) * b[i] for (i, s) in enumerate(𝒮))
    end
    if sum(b′) ≈ 0.0
        fill!(b′, 1)
    end
    return normalize!(b′, 1)
end
```

Review: Bayesian Filter

$$b_t(s) = P(s_t = s \mid h_t)$$

$$b' = \tau(b, a, o)$$

$$b'(s') \propto Z(o \mid a, s') \sum_s T(s' \mid s, a)\, b(s)$$



```
function update(b::Vector{Float64}, 𝒫, a, o)
    𝒮, T, O = 𝒫.𝒮, 𝒫.T, 𝒫.O
    b′ = similar(b)
    for (i′, s′) in enumerate(𝒮)
        po = O(a, s′, o)
        b′[i′] = po * sum(T(s, a, s′) * b[i] for (i, s) in enumerate(𝒮))
    end
    if sum(b′) ≈ 0.0
        fill!(b′, 1)
    end
    return normalize!(b′, 1)
end
```

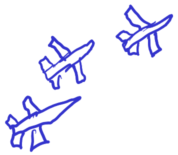
$O(|S|^2)$

$|S| = k^d$

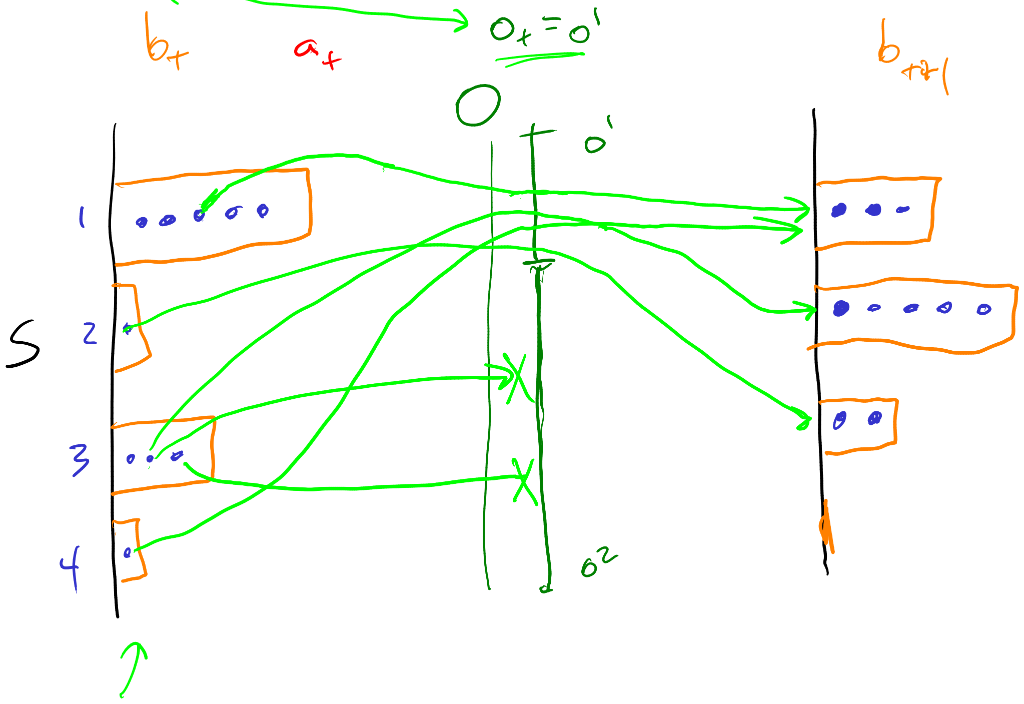
Rejection Particle Filter

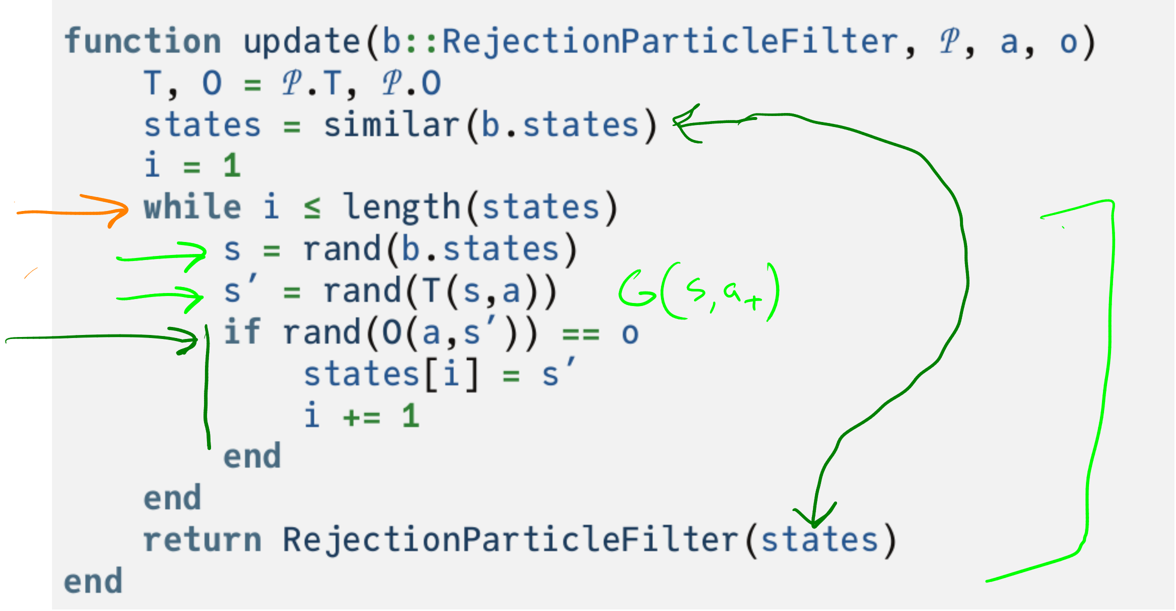
Rejection Particle Filter

State



$s', o = G(s, a_t)$



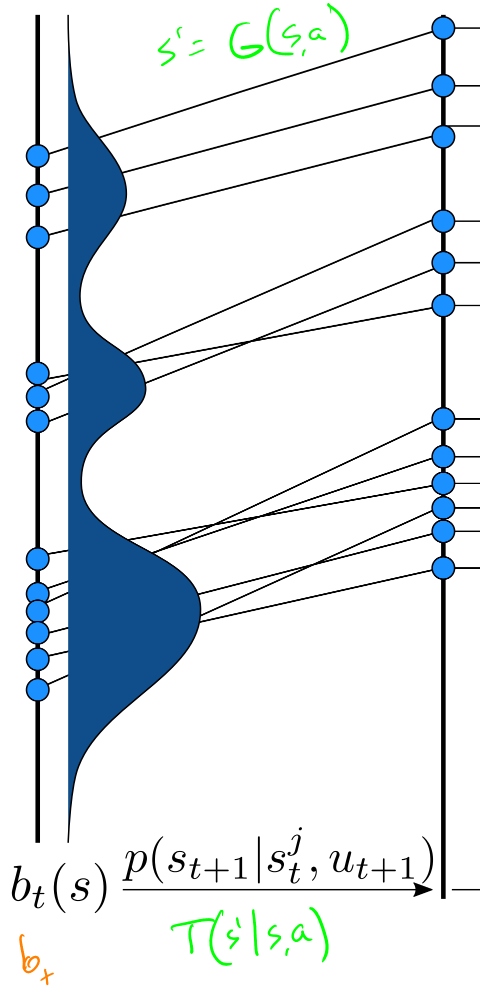


```
function update(b::RejectionParticleFilter, 𝒫, a, o)
    T, O = 𝒫.T, 𝒫.O
    states = similar(b.states)
    i = 1
    while i ≤ length(states)
        s = rand(b.states)
        s′ = rand(T(s,a))
        if rand(O(a,s′)) == o
            states[i] = s′
            i += 1
        end
    end
    return RejectionParticleFilter(states)
end
```

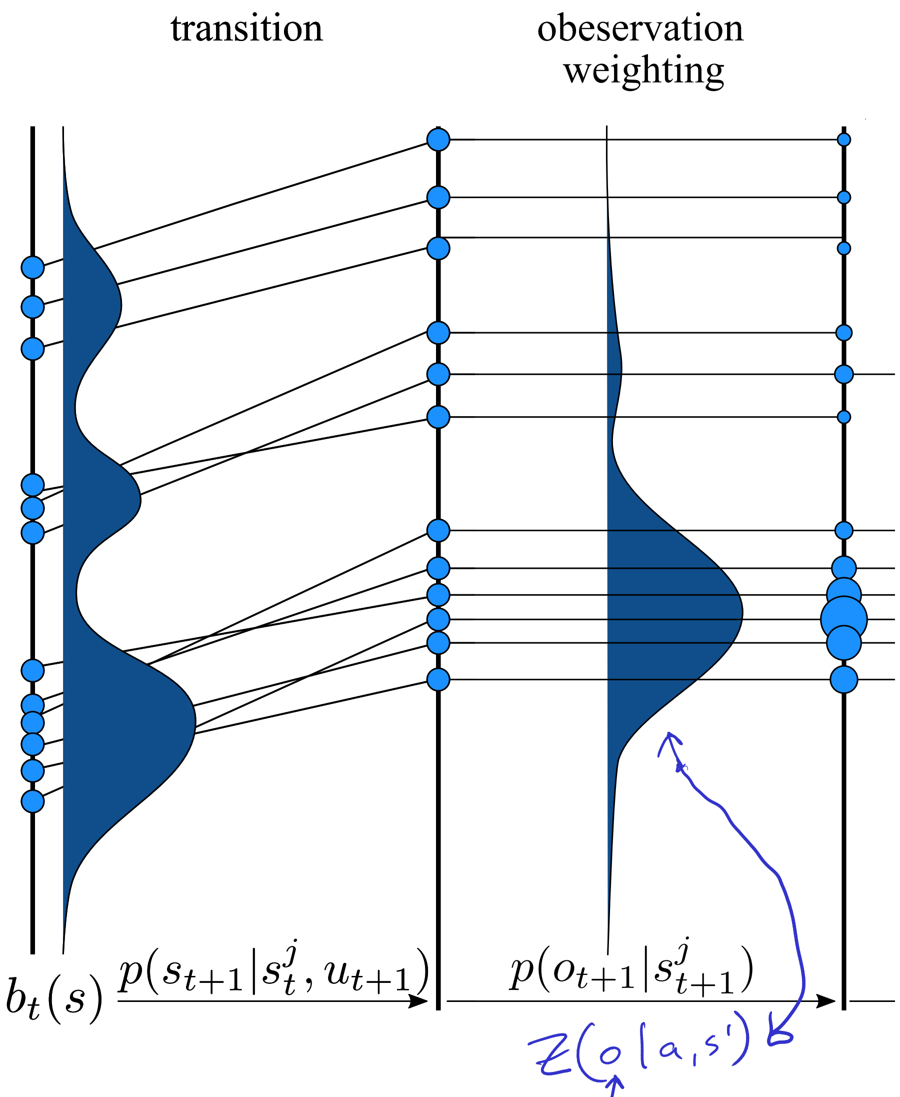
Weighted Particle Filtering

Weighted Particle Filtering

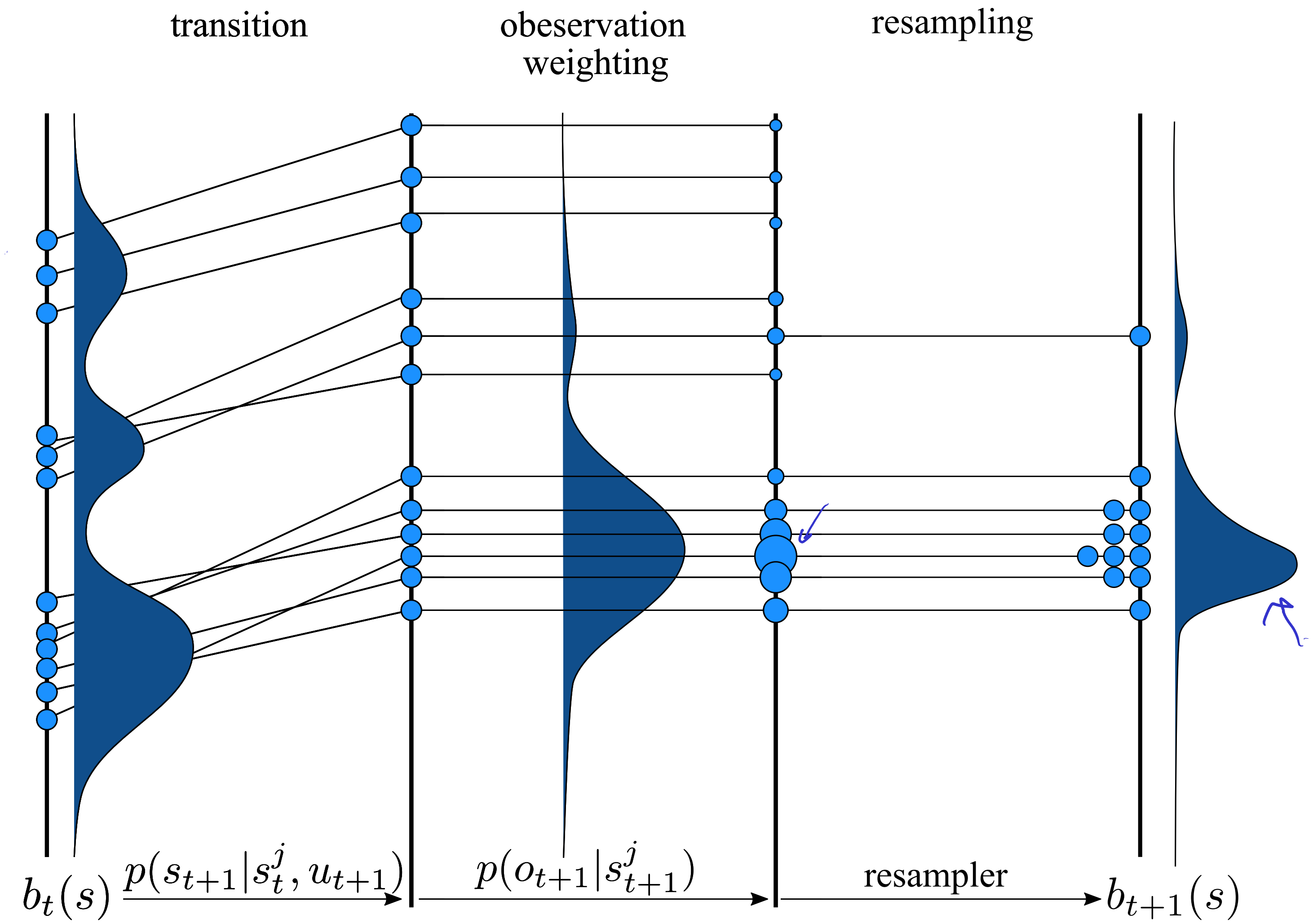
transition



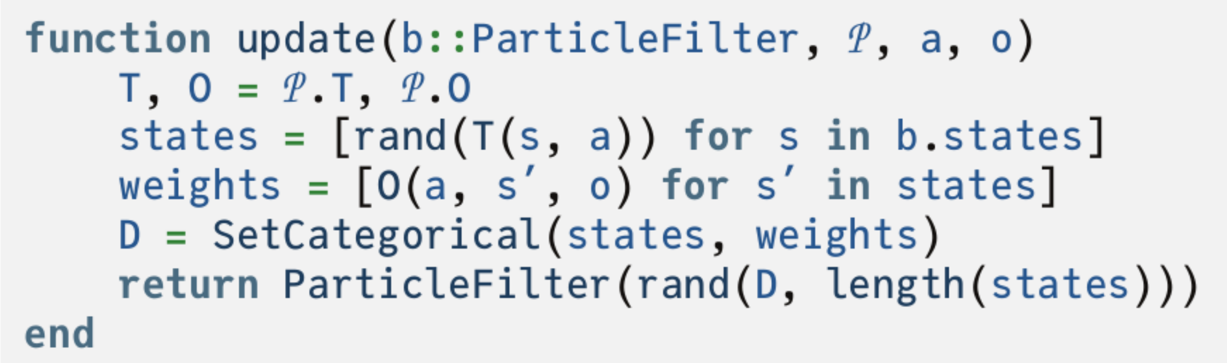
Weighted Particle Filtering



Weighted Particle Filtering

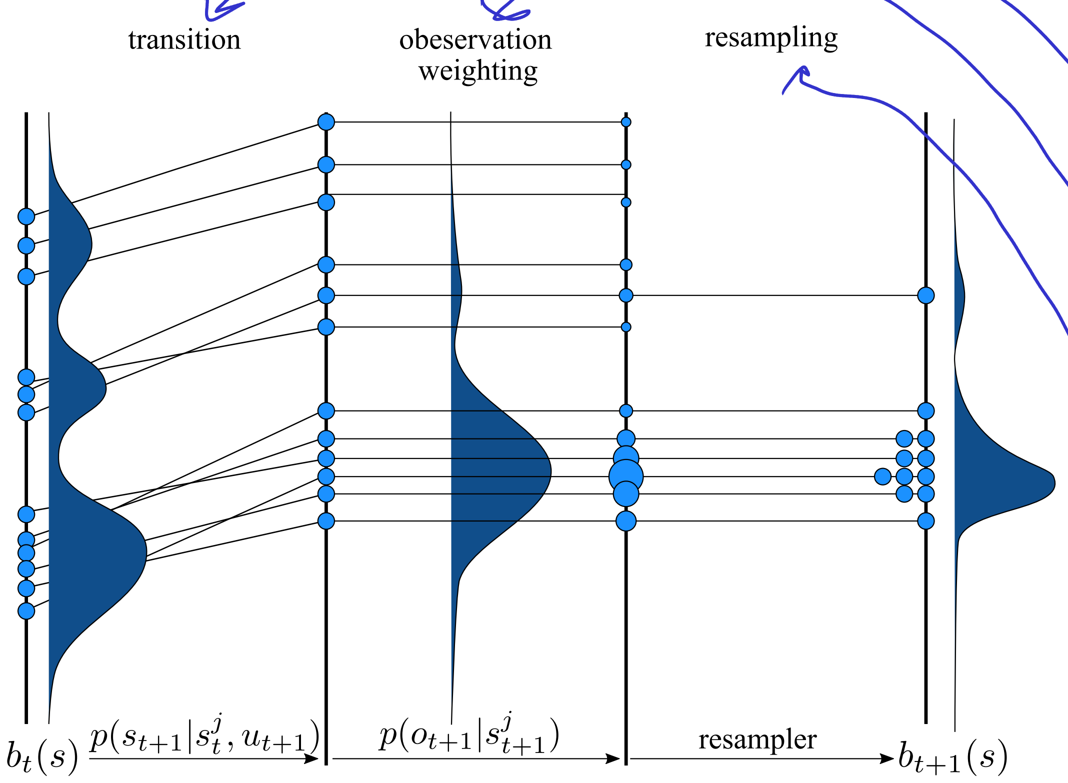


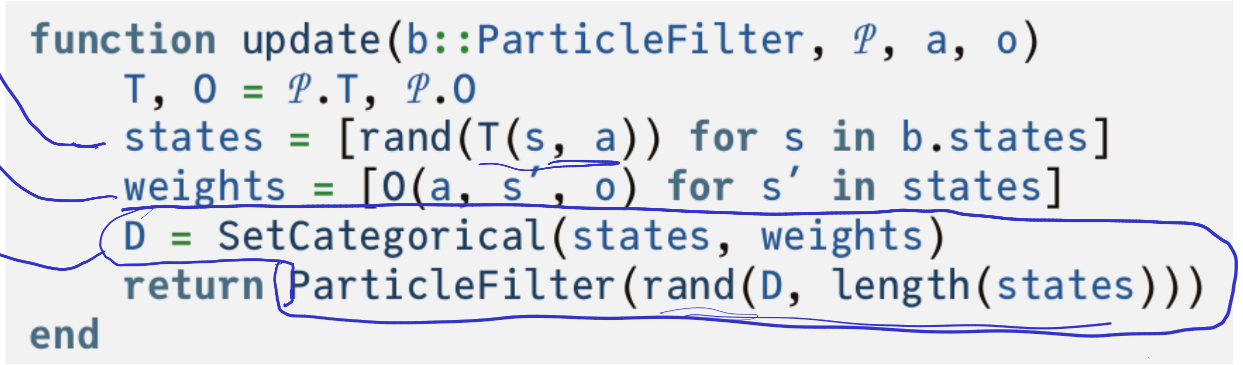
Weighted Particle Filtering



```
function update(b::ParticleFilter, 𝒫, a, o)
    T, O = 𝒫.T, 𝒫.O
    states = [rand(T(s, a)) for s in b.states]
    weights = [O(a, s′, o) for s′ in states]
    D = SetCategorical(states, weights)
    return ParticleFilter(rand(D, length(states)))
end
```

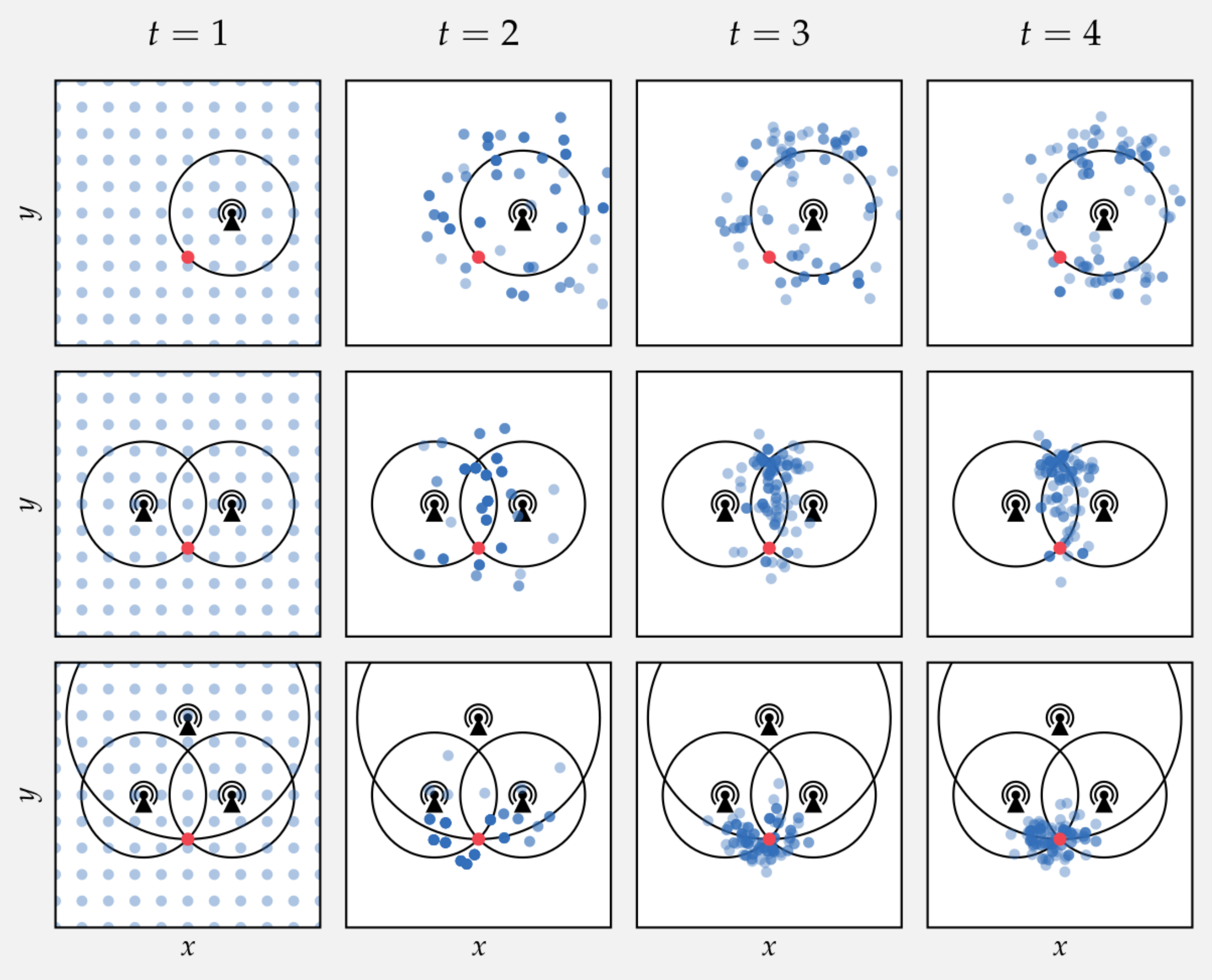
Weighted Particle Filtering





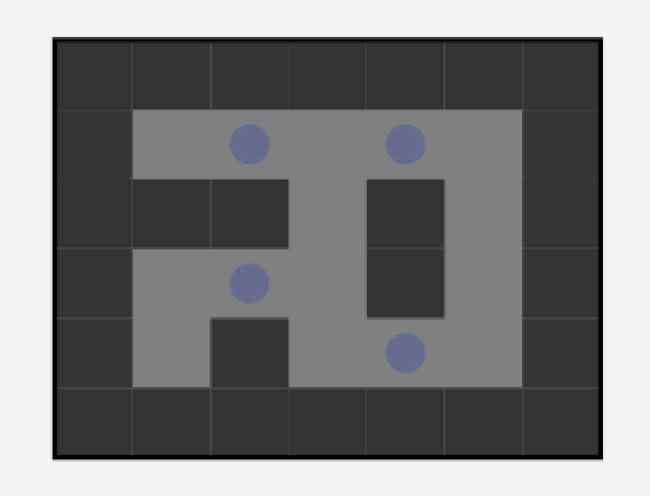
```
function update(b::ParticleFilter, 𝒫, a, o)
    T, O = 𝒫.T, 𝒫.O
    states = [rand(T(s, a)) for s in b.states]
    weights = [O(a, s′, o) for s′ in states]
    D = SetCategorical(states, weights)
    return ParticleFilter(rand(D, length(states)))
end
```

Weighted Particle Filtering

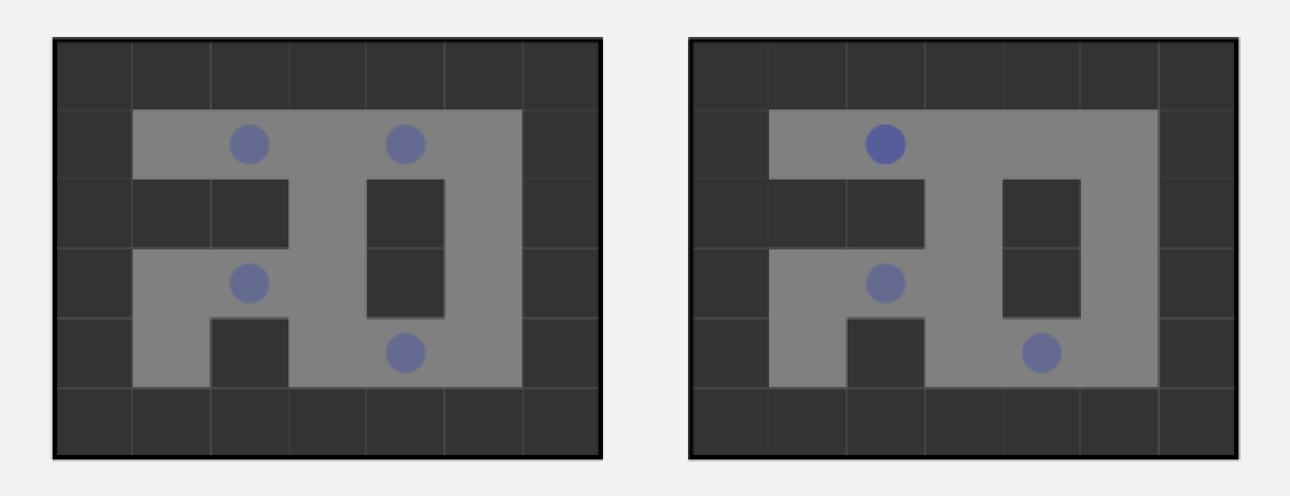


Particle Depletion

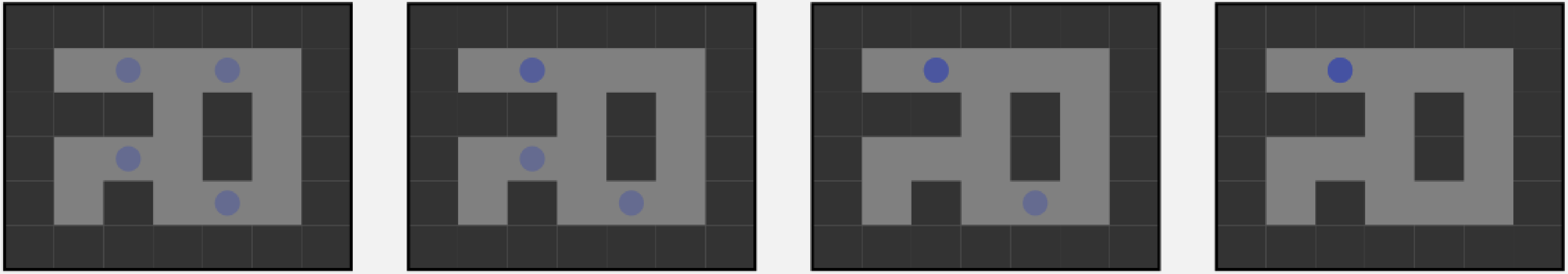
Particle Depletion



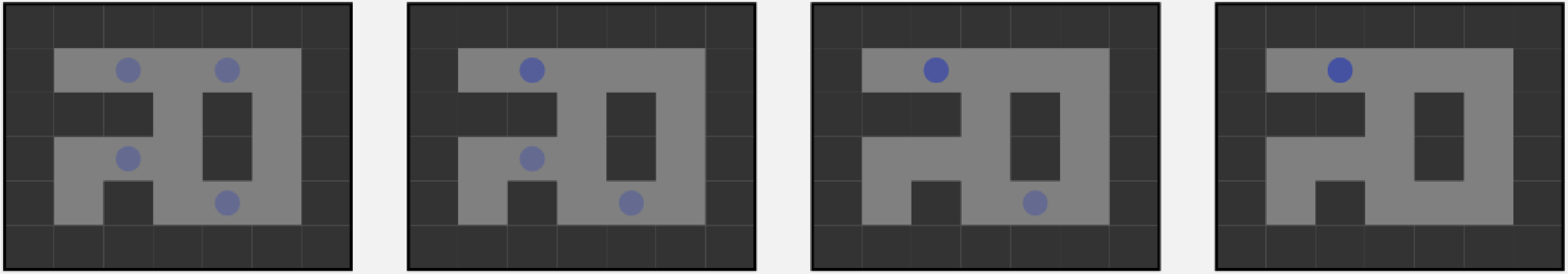
Particle Depletion



Particle Depletion

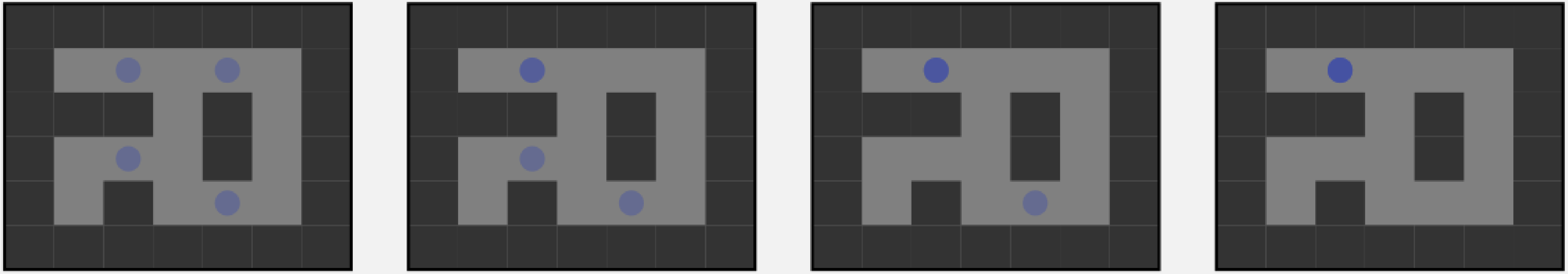


Particle Depletion



Solution: Domain specific particle injection based on:

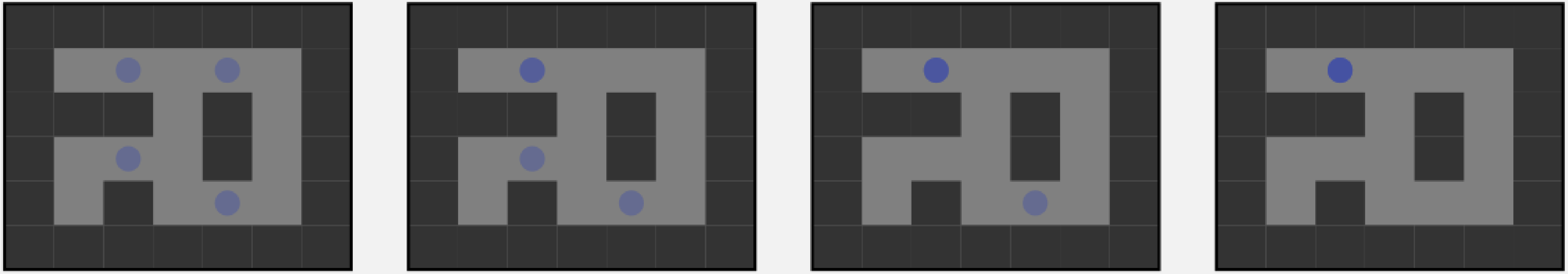
Particle Depletion



Solution: Domain specific particle injection based on:

- Weights

Particle Depletion



Solution: Domain specific particle injection based on:

- Weights
- Particle Diversity

Important Particle Filter Properties

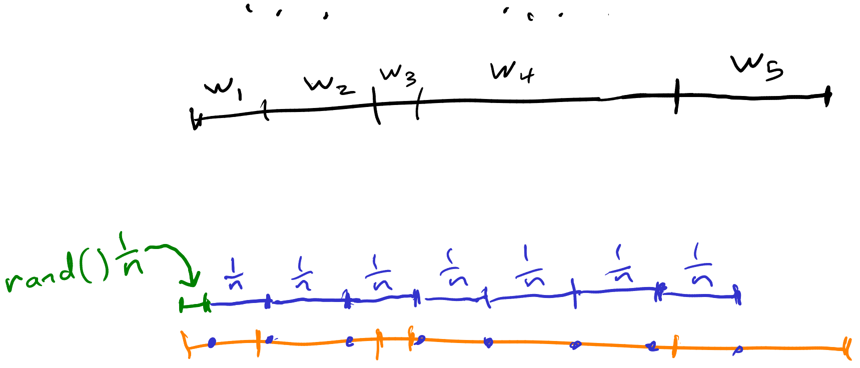
Important Particle Filter Properties

- Often the number of particles does **NOT** need to scale exponentially with the dimension (i.e. $n \neq k^d$)

Important Particle Filter Properties

- Often the number of particles does **NOT** need to scale exponentially with the dimension (i.e. $n \neq k^d$)
- Implementation should have $O(n)$ complexity.

$\vec{w}$ = weights



$O(n)$

```
for i in 1:n
    rand($\vec{w}$)        $O(n)$ ] $O(n^2)$
end
```

rand($\vec{w}$, n) $O(n)$

Low variance resampling