

Forward algorithm (simple form)

- What is the state at time t

- $P(X_t) = \sum_{x^{t-1}} P(X_t, X_{t-1} = x_{t-1})$
- $= \sum_{x^{t-1}} P(X_{t-1} = x_{t-1}) P(X_t | X_{t-1} = x_{t-1})$

Probability from previous iteration

Transition model

- Iterate this update starting at $t=0$

- This is called a **recursive** update: $P_t = g(P_{t-1}) = g(g(g(g(\dots P_0))))$

And the same thing in linear algebra

- What is the weather like at time 2?
 - $P(X_2) = 0.6\langle 0.9, 0.1 \rangle + 0.4\langle 0.3, 0.7 \rangle = \langle 0.66, 0.34 \rangle$
- In matrix-vector form:
 - $P(X_2) = \begin{pmatrix} 0.9 & 0.3 \\ 0.1 & 0.7 \end{pmatrix} \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} = \begin{pmatrix} 0.66 \\ 0.34 \end{pmatrix}$
- I.e., multiply by T^T , transpose of transition matrix

X_{t-1}	$P(X_t X_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

Stationary Distributions

- The limiting distribution is called the **stationary distribution** P_∞ of the chain
- It satisfies $P_\infty = P_{\infty+1} = T^T P_\infty$
- Solving for P_∞ in the example:

$$\begin{pmatrix} 0.9 & 0.3 \\ 0.1 & 0.7 \end{pmatrix} \begin{pmatrix} p \\ 1-p \end{pmatrix} = \begin{pmatrix} p \\ 1-p \end{pmatrix}$$

$$0.9p + 0.3(1-p) = p$$

$$p = 0.75$$

Stationary distribution is $\langle 0.75, 0.25 \rangle$ **regardless of starting distribution**

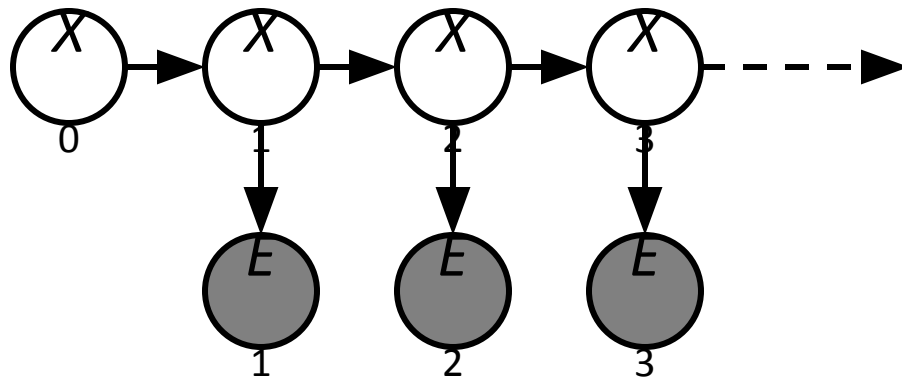


Hidden Markov Models



Hidden Markov Models

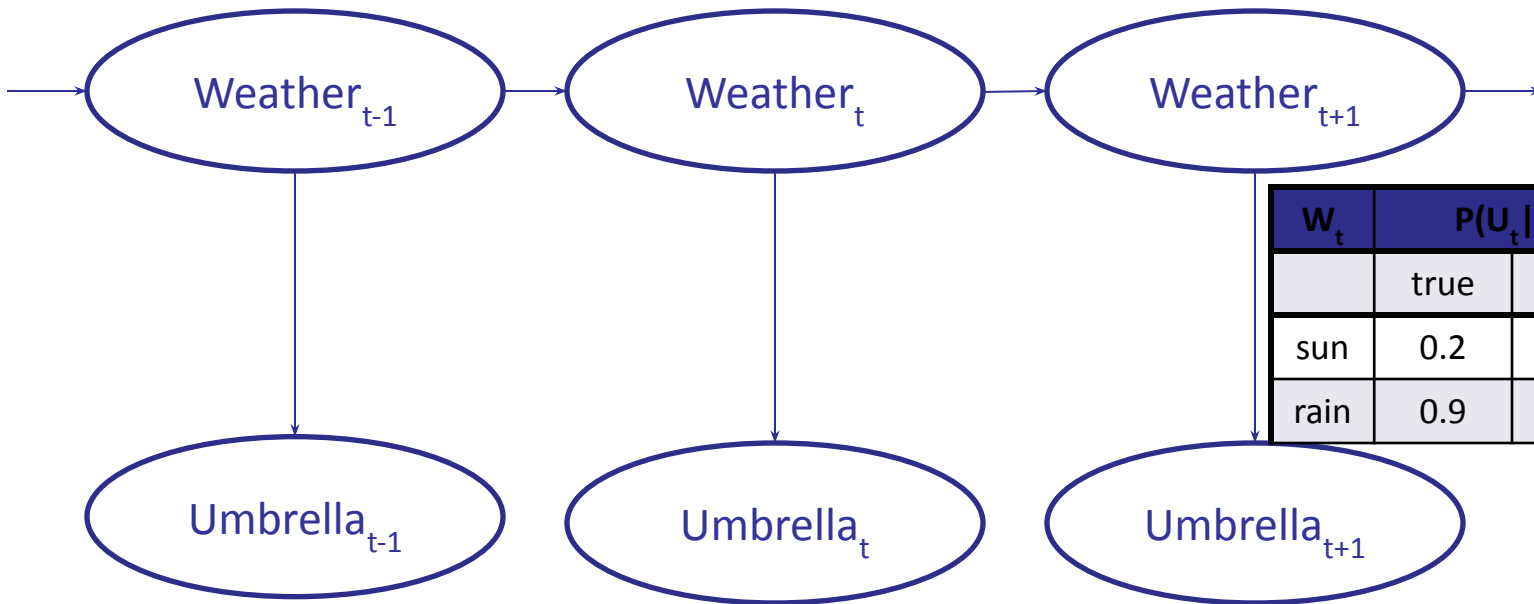
- Usually the true state is not observed directly
- Hidden Markov models (HMMs)
 - Underlying Markov chain over states X
 - You observe evidence E at each time step
 - X_t is a single discrete variable; E_t may be continuous and may consist of several variables



Example: Weather HMM

- An HMM is defined by:
 - Initial distribution: $P(X_0)$
 - Transition model: $P(X_t | X_{t-1})$
 - Sensor model: $P(E_t | X_t)$

W_{t-1}	$P(W_t W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

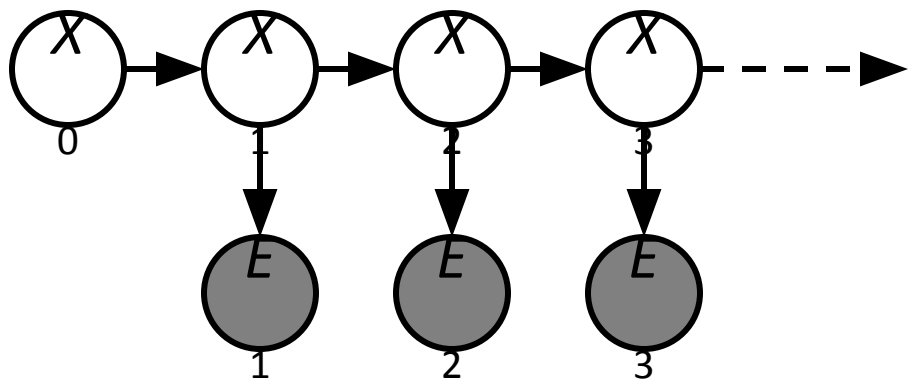


W_t	$P(U_t W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1



HMM as probability model

- Joint distribution for Markov model: $P(X_0, \dots, X_T) = P(X_0) \prod_{t=1:T} P(X_t | X_{t-1})$
- Joint distribution for hidden Markov model:
 $P(X_0, X_1, \dots, X_T, E_T) = P(X_0) \prod_{t=1:T} P(X_t | X_{t-1}) P(E_t | X_t)$
- Future states are independent of the past given the present
- Current evidence is independent of everything else given the current state
- Are evidence variables independent of each other?



Useful notation:

$$X_{a:b} = X_a, X_{a+1}, \dots, X_b$$

Real HMM Examples

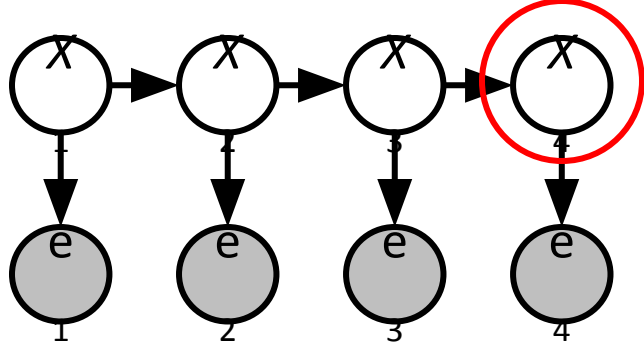
- **Speech recognition HMMs:**
 - Observations are acoustic signals (continuous valued)
 - States are specific positions in specific words (so, tens of thousands)
- **Machine translation HMMs:**
 - Observations are words (tens of thousands)
 - States are translation options
- **Robot tracking:**
 - Observations are range readings (continuous)
 - States are positions on a map (continuous)
- **Molecular biology:**
 - Observations are nucleotides ACGT
 - States are coding/non-coding/start/stop/splice-site etc.

Inference tasks

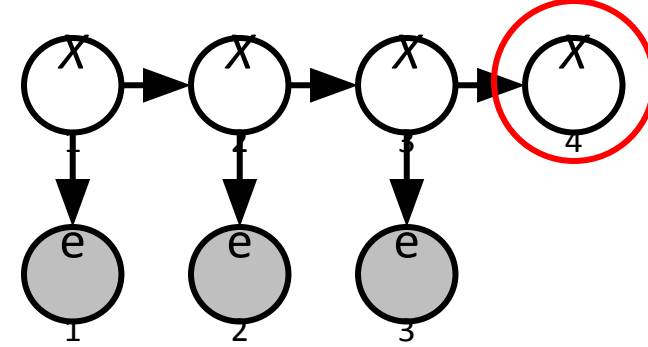
- **Filtering:** $P(X_t | e_{1:t})$
 - **belief state**—input to the decision process of a rational agent
- **Prediction:** $P(X_{t+k} | e_{1:t})$ for $k > 0$
 - evaluation of possible action sequences; like filtering without the evidence
- **Smoothing:** $P(X_k | e_{1:t})$ for $0 \leq k < t$
 - better estimate of past states, essential for learning
- **Most likely explanation:** $\arg \max_{x_{1:t}} P(x_{1:t} | e_{1:t})$
 - speech recognition, decoding with a noisy channel

Inference tasks

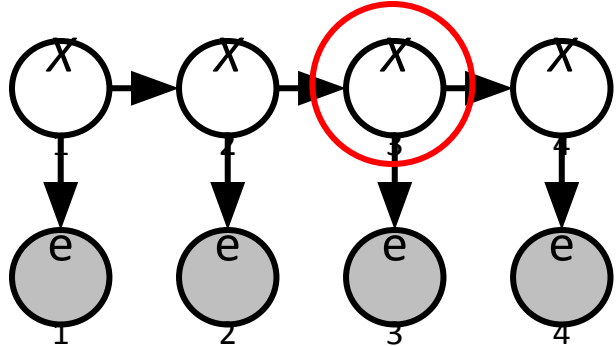
Filtering: $P(X_t | e_{1:t})$



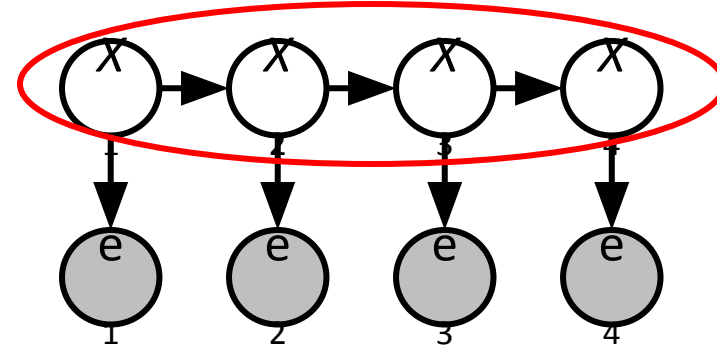
Prediction: $P(X_{t+k} | e_{1:t})$



Smoothing: $P(X_k | e_{1:t}), k < t$



Explanation: $P(X_{1:t} | e_{1:t})$

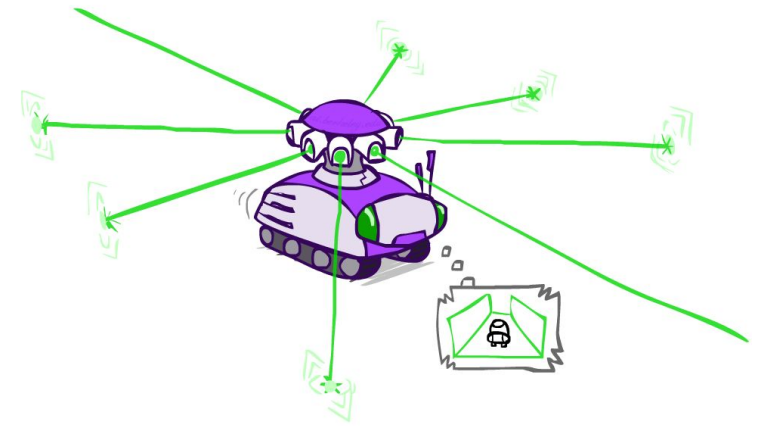
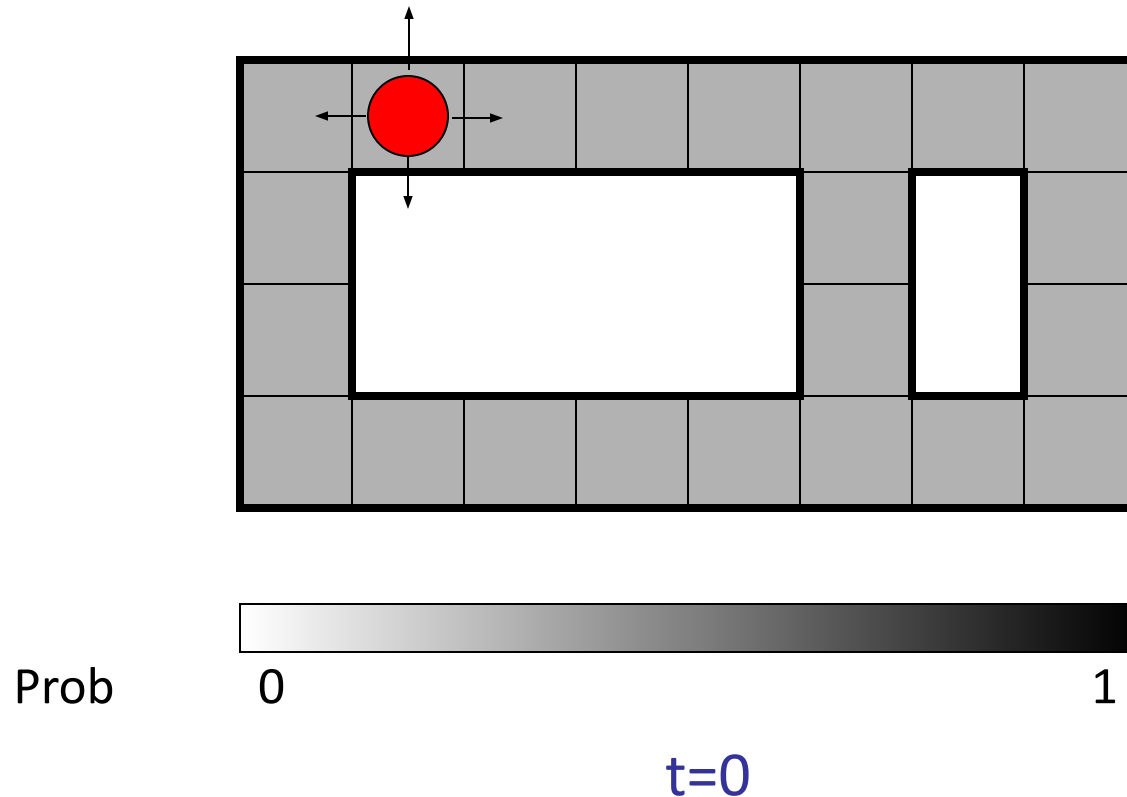


Filtering / Monitoring

- Filtering, or monitoring, or state estimation, is the task of maintaining the distribution $f_{1:t} = P(X_t | e_{1:t})$ over time
- We start with f_0 in an initial setting, usually uniform
- Filtering is a fundamental task in engineering and science
- The Kalman filter (continuous variables, linear dynamics, Gaussian noise) was invented in 1960 and used for trajectory estimation in the Apollo program; core ideas used by Gauss for planetary observations; **1,320,000** papers on Google Scholar

Example: Robot Localization

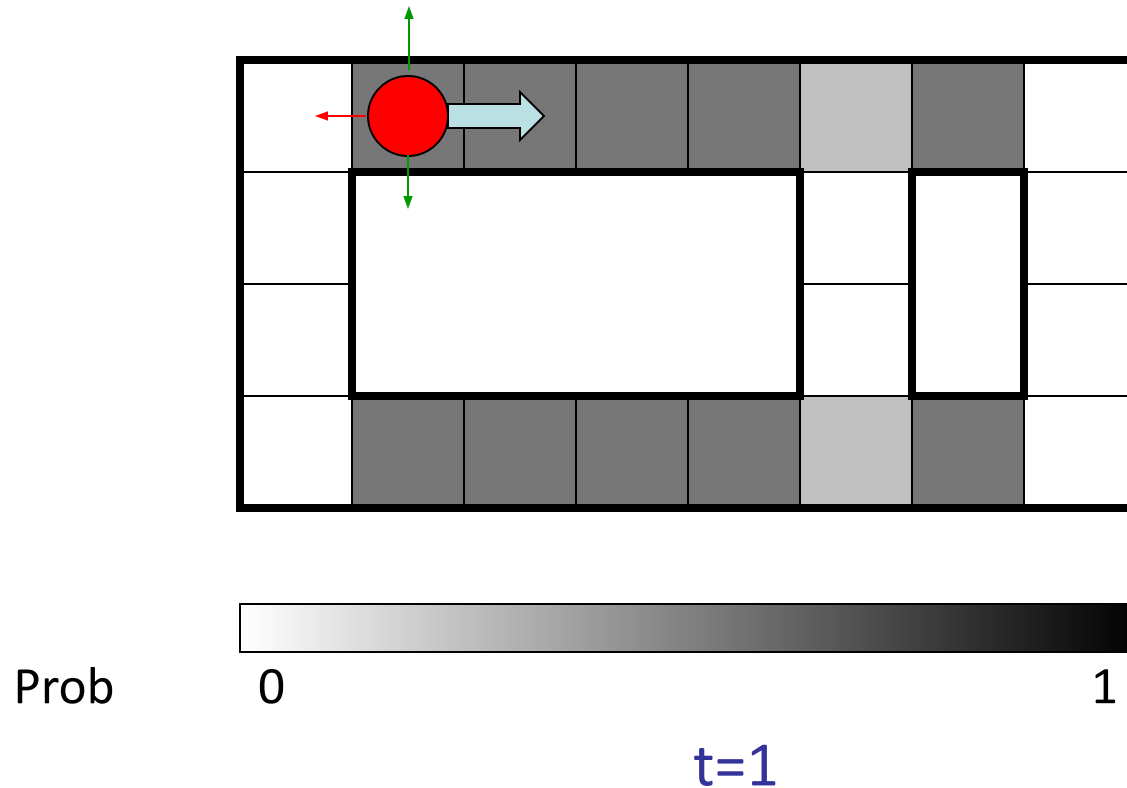
Example from
Michael Pfeiffer



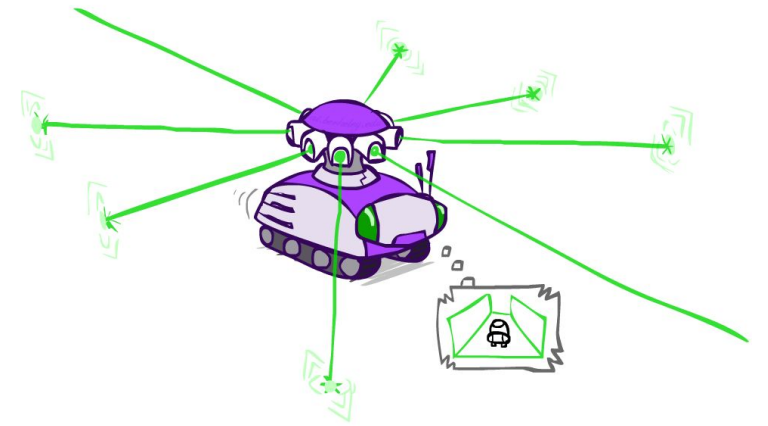
Sensor model: four bits for wall/no-wall in each direction,
never more than 1 mistake

Transition model: action may fail with small prob.

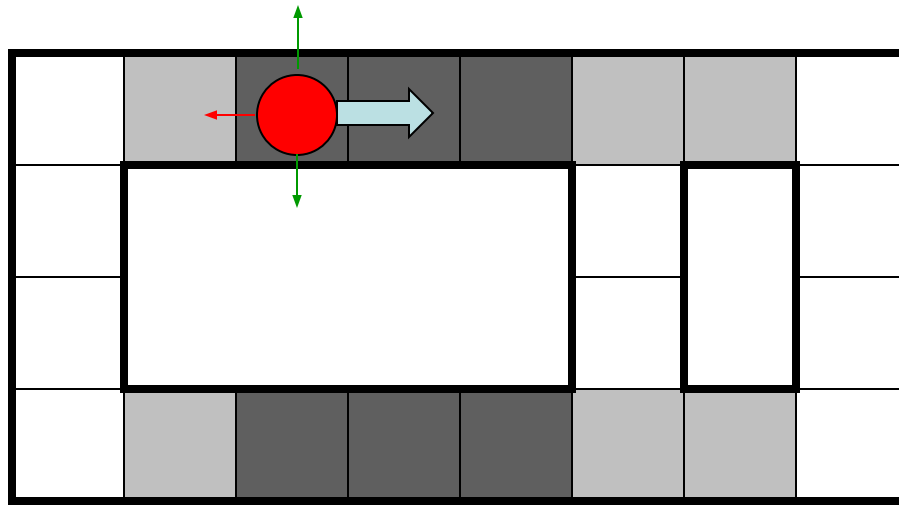
Example: Robot Localization



Lighter grey: was *possible* to get the reading,
but *less likely* (required 1 mistake)



Example: Robot Localization

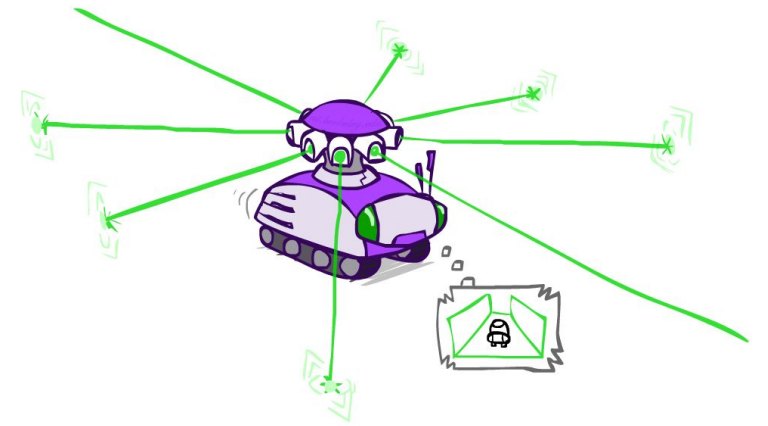


Prob

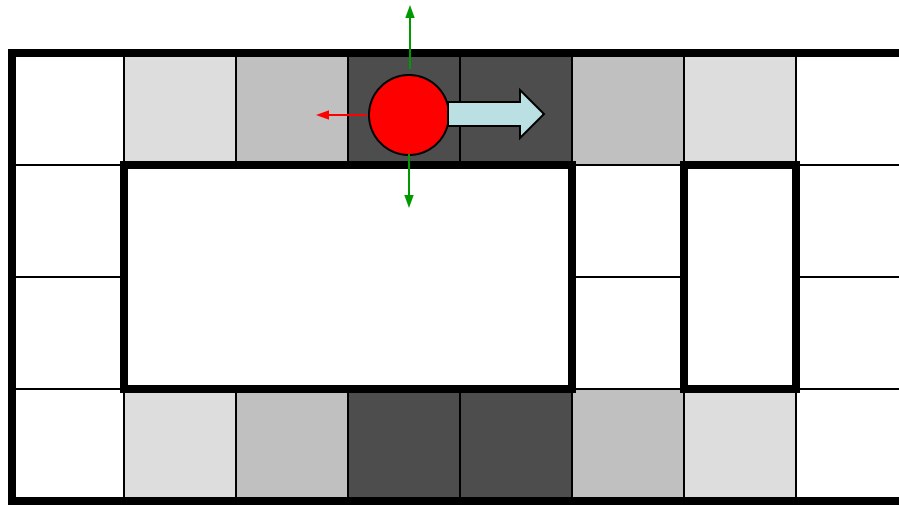
0

1

t=2



Example: Robot Localization

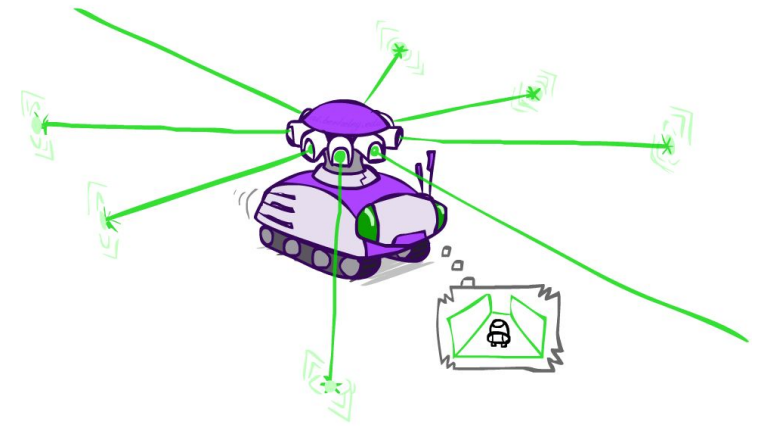


Prob

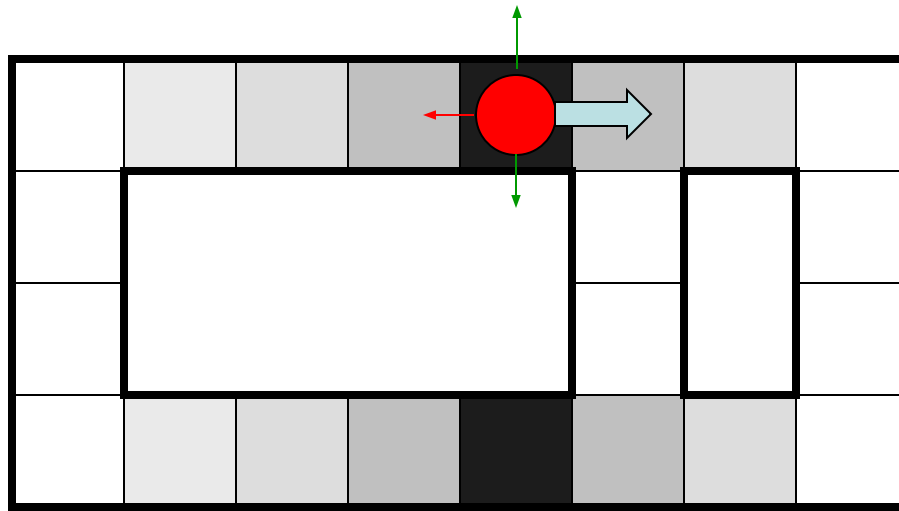
0

1

t=3



Example: Robot Localization

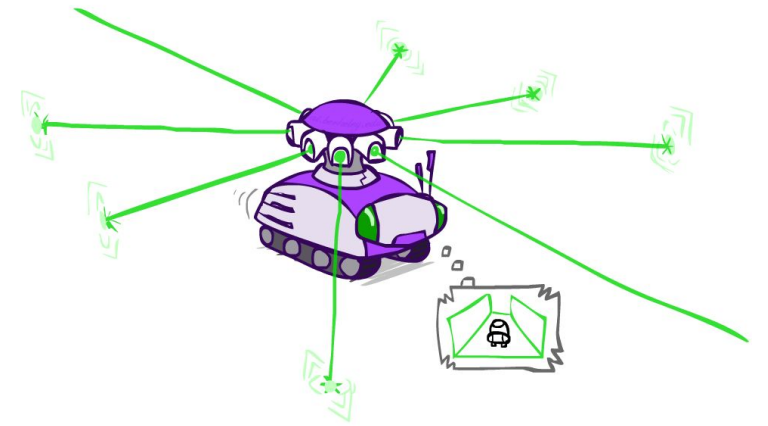


Prob

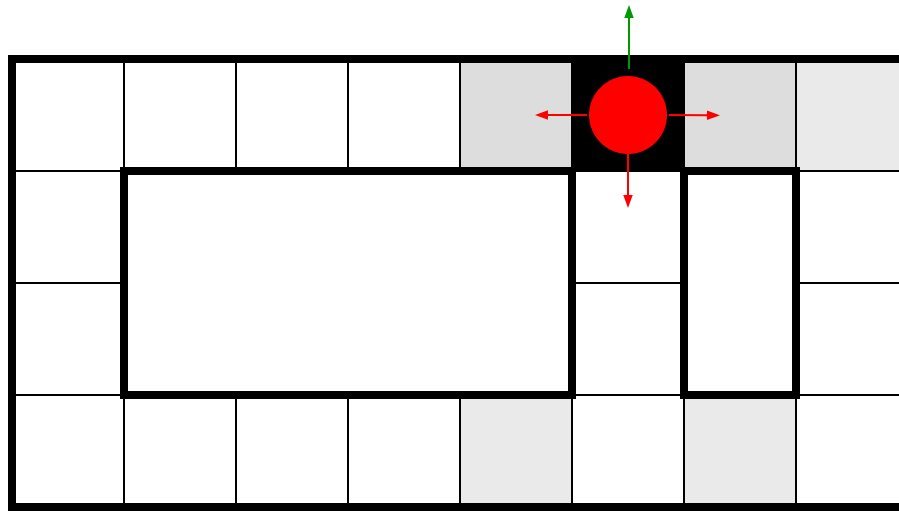
0

1

$t=4$



Example: Robot Localization

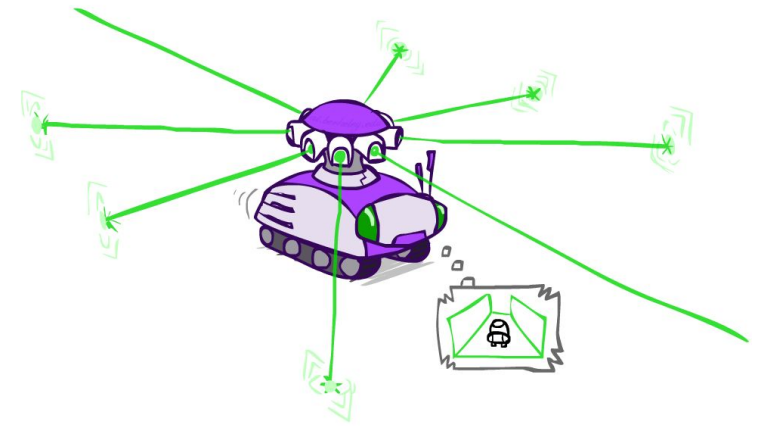


Prob

0

1

t=5



Filtering algorithm

- Aim: devise a **recursive filtering** algorithm of the form

- $P(X_{t+1} | e_{1:t+1}) = g(e_{t+1}, P(X_t | e_{1:t}))$

- $P(X_{t+1} | e_{1:t+1}) =$

Filtering algorithm

- Aim: devise a **recursive filtering** algorithm of the form

- $P(X_{t+1} | e_{1:t+1}) = g(e_{t+1}, P(X_t | e_{1:t}))$

- $P(X_{t+1} | e_{1:t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1})$

Apply Bayes' rule

Apply conditional independence

- $= \alpha P(e_{t+1} | X_{t+1}, e_{1:t}) P(X_{t+1} | e_{1:t})$

Condition on X_t

- $= \alpha P(e_{t+1} | X_{t+1}) P(X_{t+1} | e_{1:t})$

Apply conditional independence

- $= \frac{1}{Z} P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t | e_{1:t}) P(X_{t+1} | x_t, e_{1:t})$

Normalize

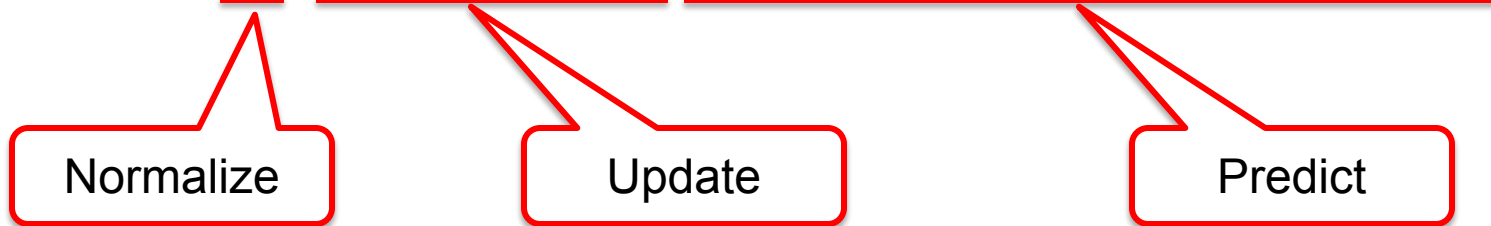
Update

Predict

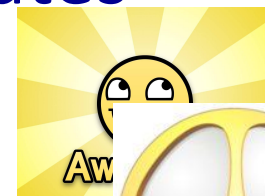
- $P(X_{t+1} | e_{1:t+1}) = \frac{1}{Z} P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t | e_{1:t}) P(X_{t+1} | x_t, e_{1:t})$

Filtering algorithm

$$\blacksquare P(X_{t+1} | e_{1:t+1}) = \alpha P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t | e_{1:t}) P(X_{t+1} | x_t)$$



- $f_{1:t+1} = \text{FORWARD}(f_{1:t}, e_{t+1})$
- Cost per time step: $O(|X|^2)$ where $|X|$ is the number of states
- Time and space costs are **constant**, independent of t
- $O(|X|^2)$ is infeasible for models with many state variables
- We get to invent really cool approximate filtering algorithms



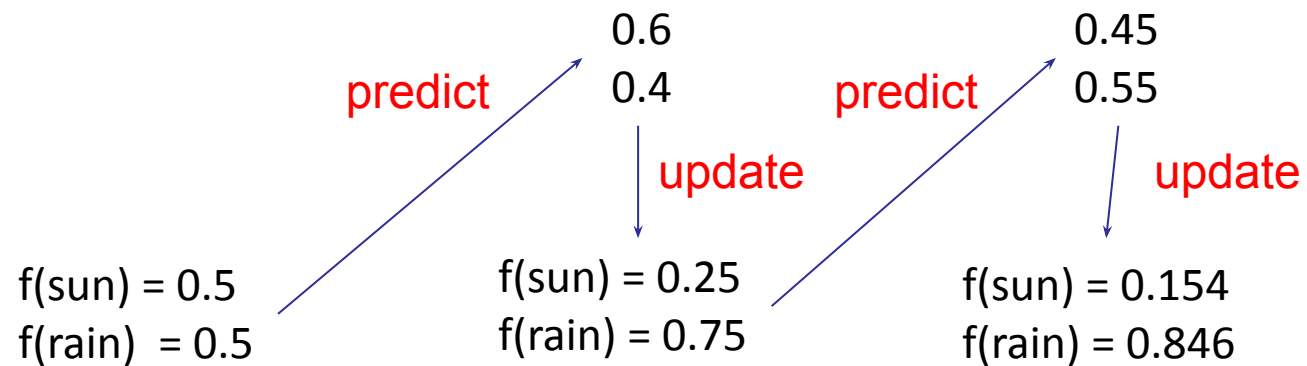
And the same thing in linear algebra

- Transition matrix T , observation matrix O_t
 - Observation matrix has state likelihoods for E_t along diagonal
 - E.g., for $U_1 = \text{true}$, $O_1 = \begin{pmatrix} 0.2 & 0 \\ 0 & 0.9 \end{pmatrix}$
- Filtering algorithm becomes
 - $f_{1:t+1} = \alpha O_{t+1} T^T f_{1:t}$

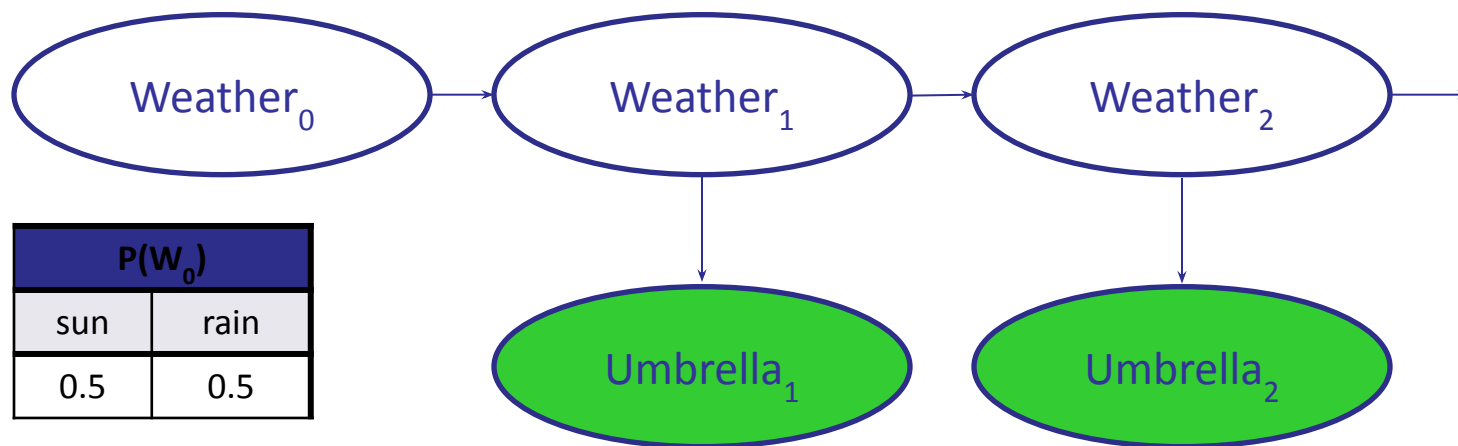
X_{t-1}	$P(X_t X_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

W_t	$P(U_t W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

Example: Weather HMM



W_{t-1}	$P(W_t W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7



$P(W_0)$	
sun	rain
0.5	0.5

W_t	$P(U_t W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

Pacman – Hunting Invisible Ghosts with Sonar



[Demo: Pacman – Sonar – No Beliefs(L14D1)]

Video of Demo Pacman – Sonar



Most Likely Explanation

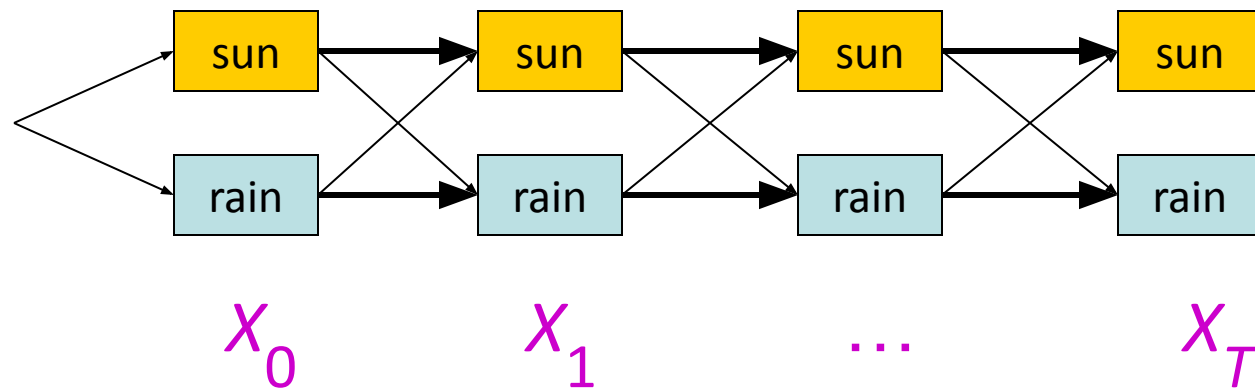


Inference tasks

- **Filtering:** $P(X_t | e_{1:t})$
 - **belief state**—input to the decision process of a rational agent
- **Prediction:** $P(X_{t+k} | e_{1:t})$ for $k > 0$
 - evaluation of possible action sequences; like filtering without the evidence
- **Smoothing:** $P(X_k | e_{1:t})$ for $0 \leq k < t$
 - better estimate of past states, essential for learning
- **Most likely explanation:** $\arg \max_{x_{1:t}} P(x_{1:t} | e_{1:t})$
 - speech recognition, decoding with a noisy channel

Most likely explanation = most probable path

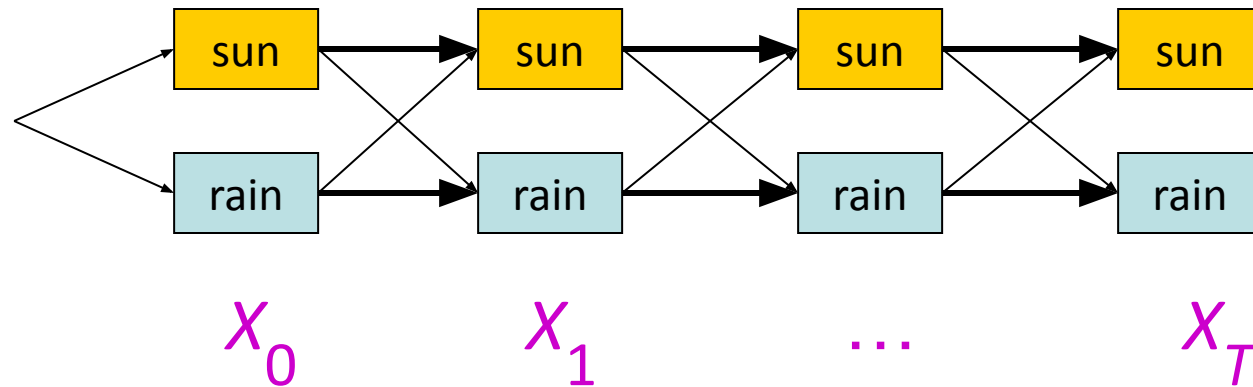
- **State trellis**: graph of states and transitions over time



- $\arg \max_{x_{1:t}} P(x_{1:t} | e_{1:t})$
= $\arg \max_{x_{1:t}} \alpha P(x_{1:t}, e_{1:t})$
= $\arg \max_{x_{1:t}} P(x_{1:t}, e_{1:t})$
= $\arg \max_{x_{1:t}} P(x_0) \prod_t P(x_t | x_{t-1}) P(e_t | x_t)$

- Each arc represents some transition $x_{t-1} \rightarrow x_t$
- Each arc has weight $P(x_t | x_{t-1}) P(e_t | x_t)$ (arcs to initial states have weight $P(x_0)$)
- The **product** of weights on a path is proportional to that state sequence's probability
- Forward algorithm computes sums of paths, **Viterbi algorithm** computes best paths

Forward / Viterbi algorithms



Forward Algorithm (sum)

For each state at time t , keep track of the **total probability of all paths** to it

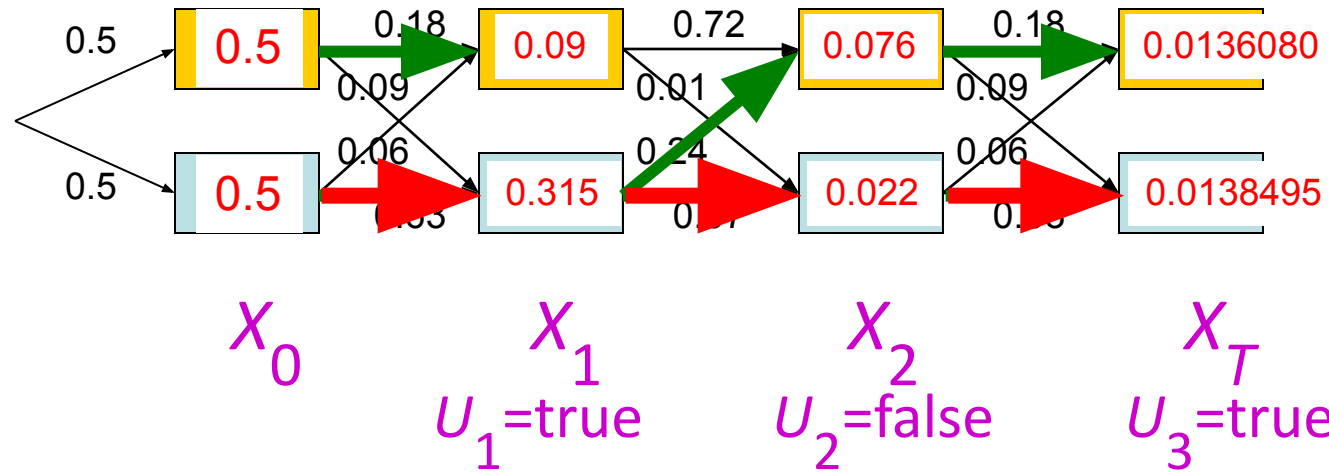
$$\begin{aligned} \mathbf{f}_{1:t+1} &= \text{FORWARD}(\mathbf{f}_{1:t}, e_{t+1}) \\ &= \alpha P(e_{t+1} | X_{t+1}) \sum_{x_t} P(X_{t+1} | x_t) \mathbf{f}_{1:t} \end{aligned}$$

Viterbi Algorithm (max)

For each state at time t , keep track of the **maximum probability of any path** to it

$$\begin{aligned} \mathbf{m}_{1:t+1} &= \text{VITERBI}(\mathbf{m}_{1:t}, e_{t+1}) \\ &= P(e_{t+1} | X_{t+1}) \max_{x_t} P(X_{t+1} | x_t) \mathbf{m}_{1:t} \end{aligned}$$

Viterbi algorithm contd.



W_{t-1}	$P(W_t W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

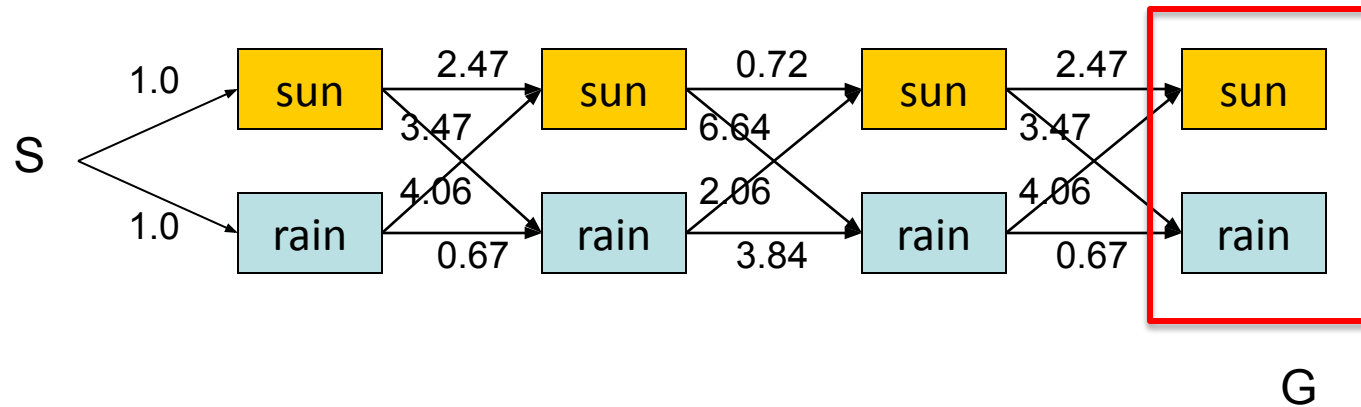
W_t	$P(U_t W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

Time complexity?
 $O(|X|^2 T)$

Space complexity?
 $O(|X| T)$

Number of paths?
 $O(|X|^T)$

Viterbi in negative log space



W_{t-1}	$P(W_t W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

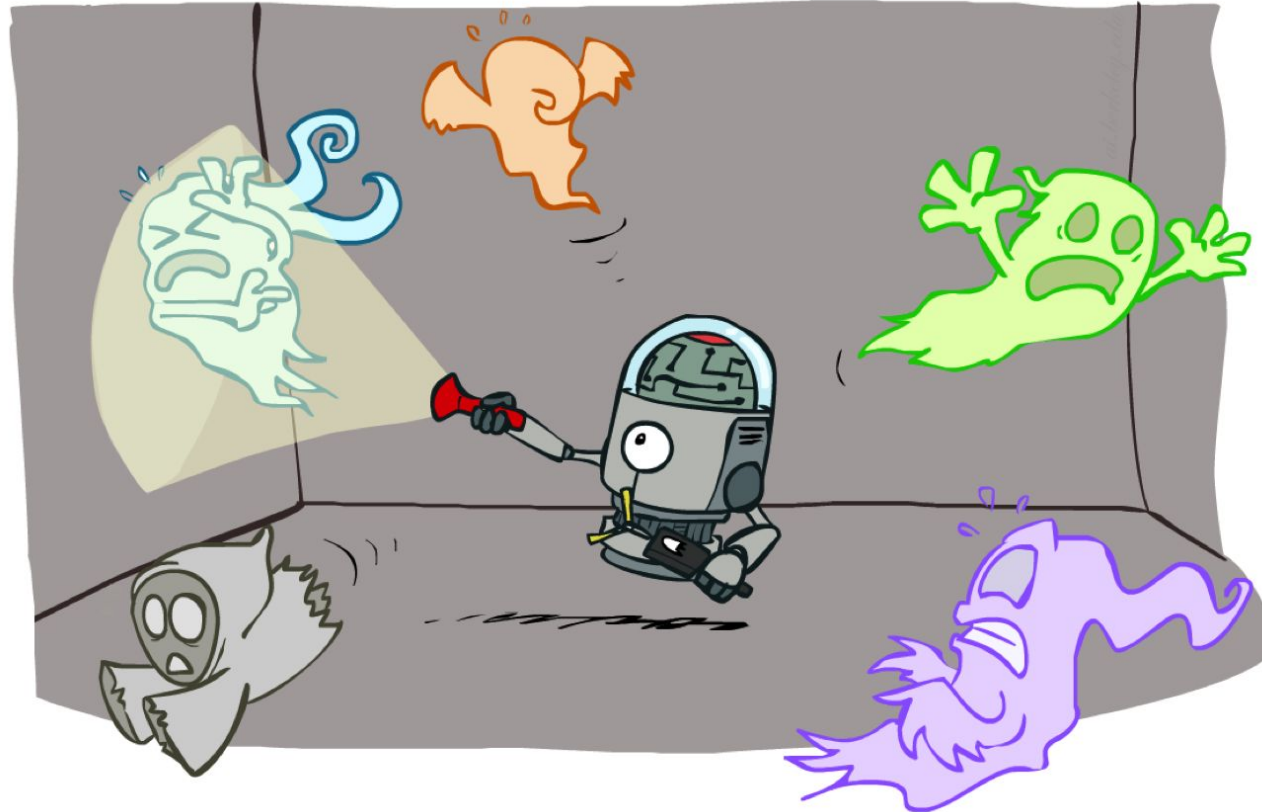
W_t	$P(U_t W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

argmax of product of probabilities
= argmin of sum of negative log probabilities
= minimum-cost path

Viterbi is essentially breadth-first graph search
What about A*?

CS 188: Artificial Intelligence

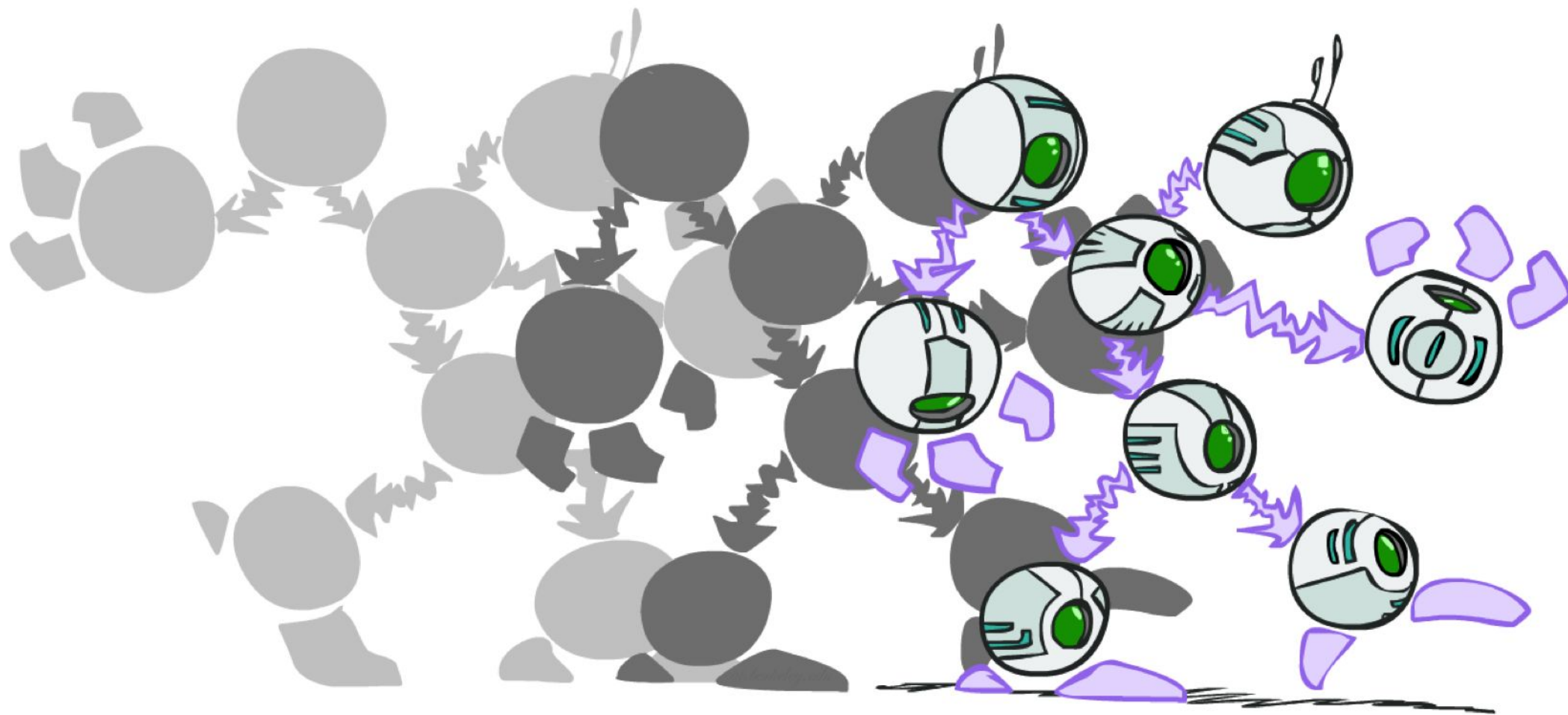
Dynamic Bayes Nets and Particle Filters



Instructor: Dan Klein and Stuart Russell

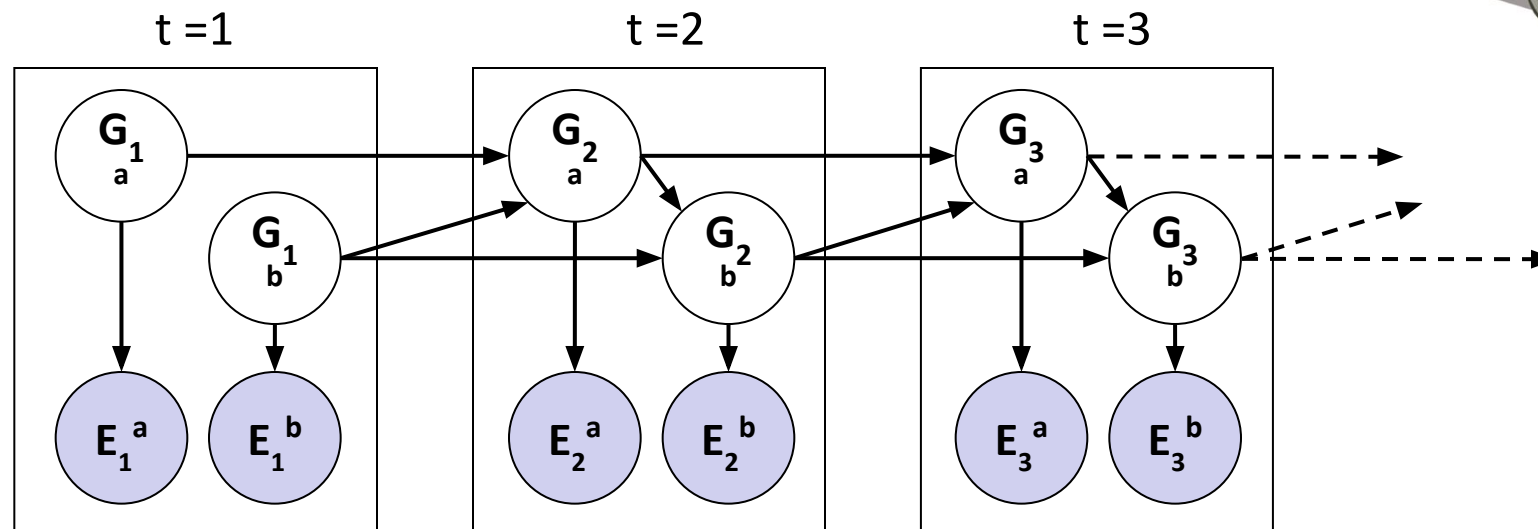
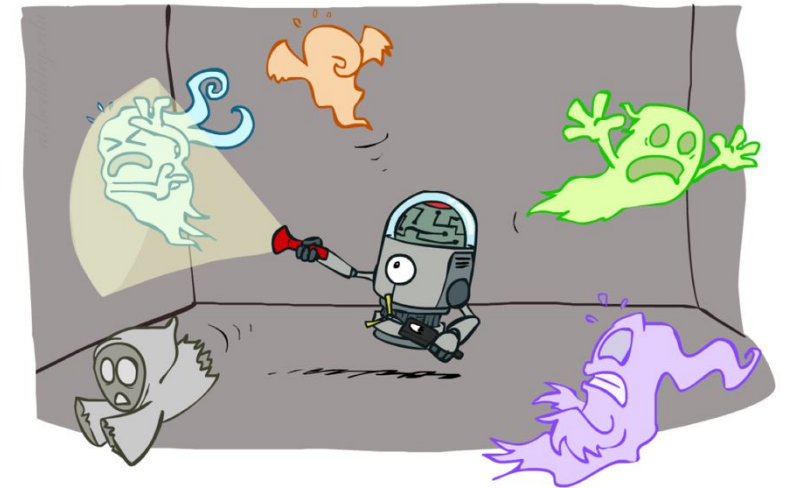
University of California, Berkeley

Dynamic Bayes Nets



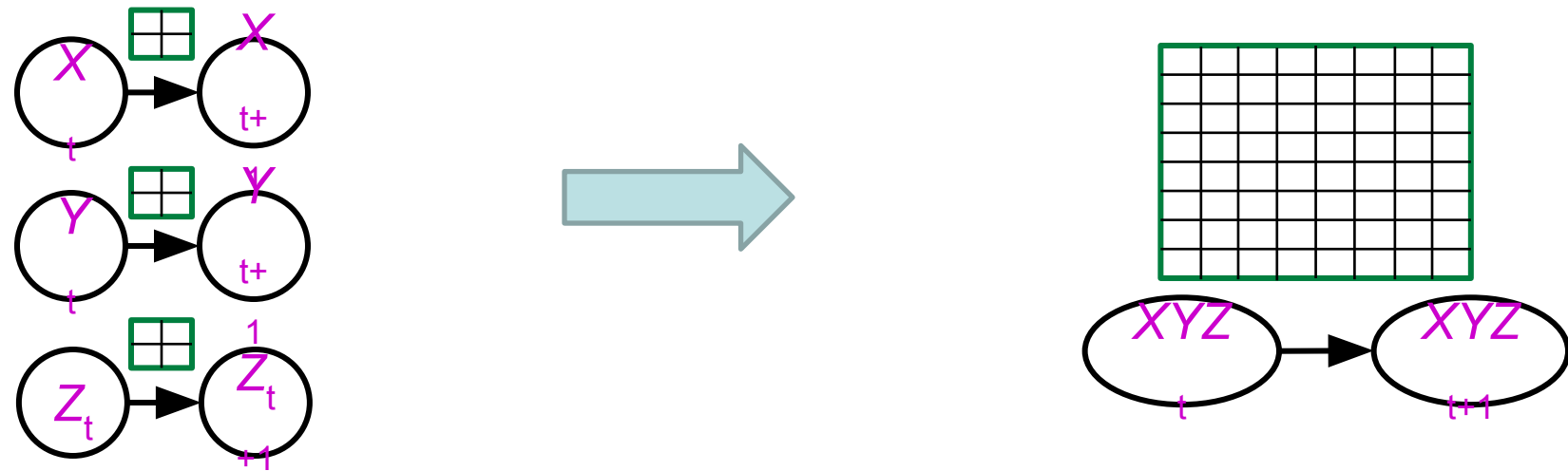
Dynamic Bayes Nets (DBNs)

- We want to track multiple variables over time, using multiple sources of evidence
- Idea: Repeat a fixed Bayes net structure at each time
- Variables at time t can have parents at time $t-1$



DBNs and HMMs

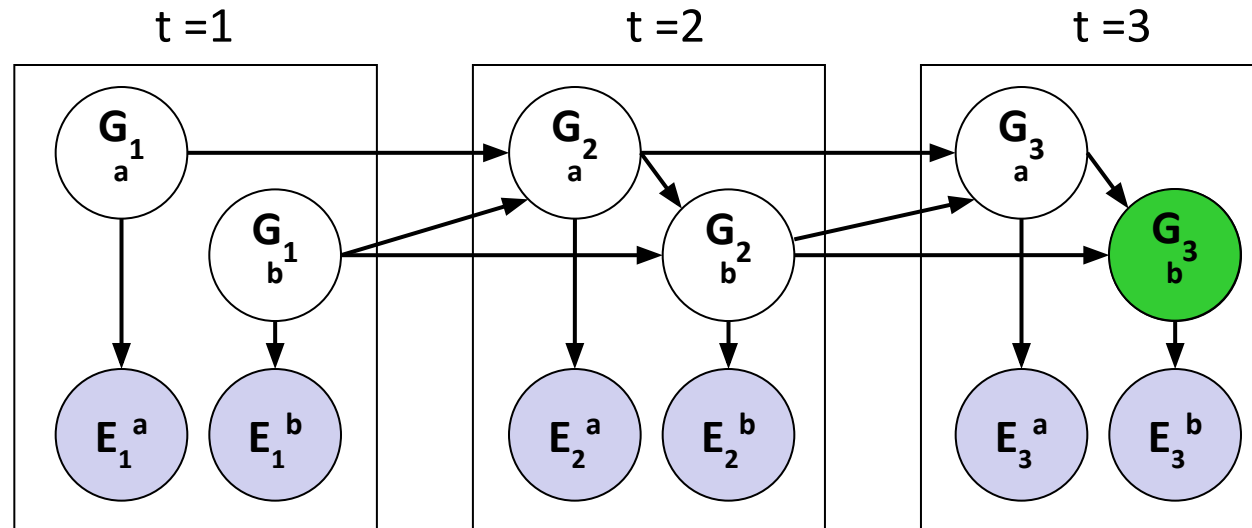
- Every HMM is a single-state-variable DBN
- Every discrete DBN is an HMM
 - HMM state is Cartesian product of DBN state variables



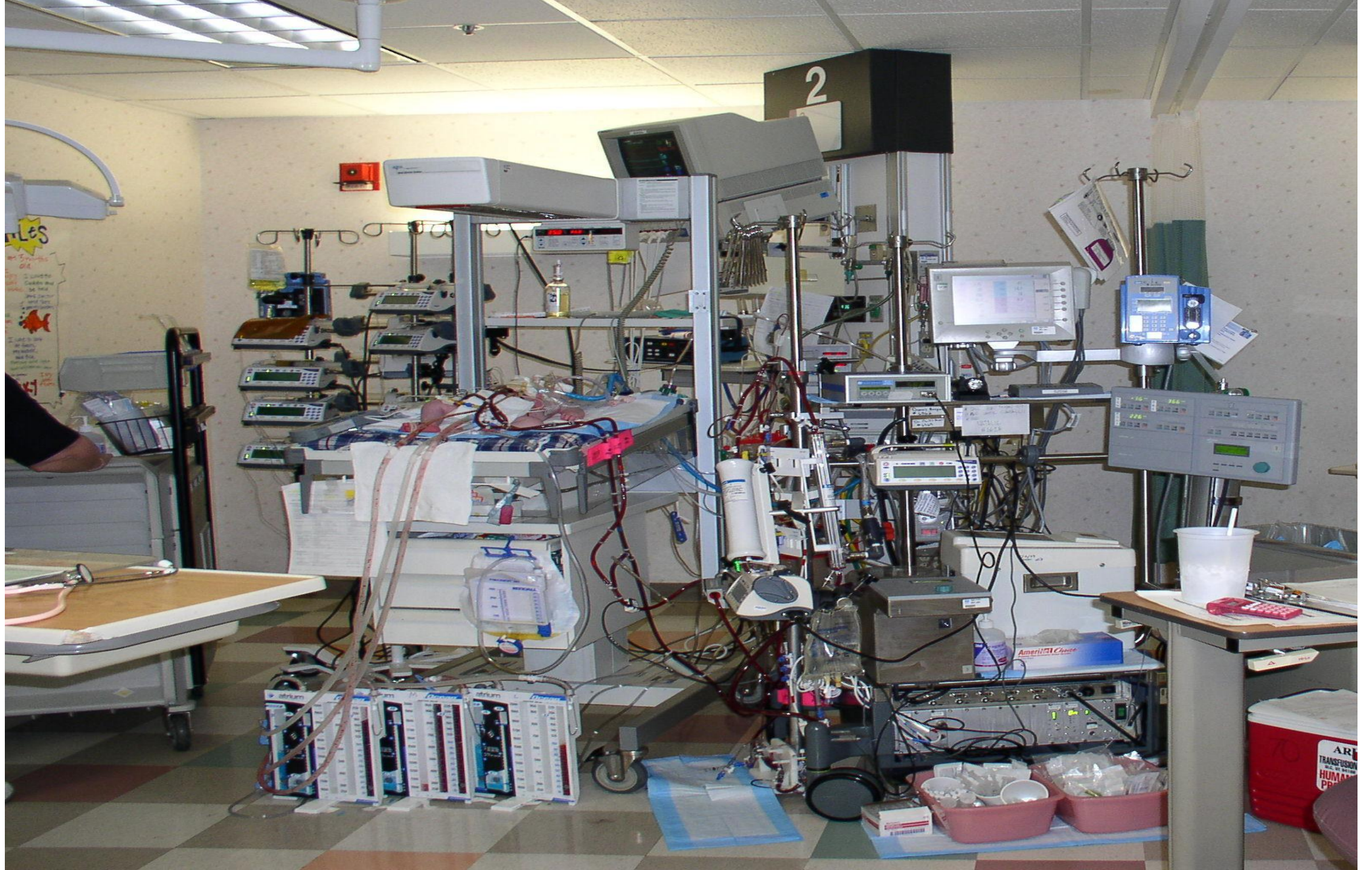
- Sparse dependencies \Rightarrow exponentially fewer parameters in DBN
 - E.g., 20 Boolean state variables, 3 parents each;
DBN has $20 \times 2^3 = 160$ parameters, HMM has $2^{20} \times 2^{20} \approx 10^{12}$ parameters

Exact Inference in DBNs

- Variable elimination applies to dynamic Bayes nets
- Offline: “unroll” the network for T time steps, then eliminate variables to find $P(X_T | e_{1:T})$



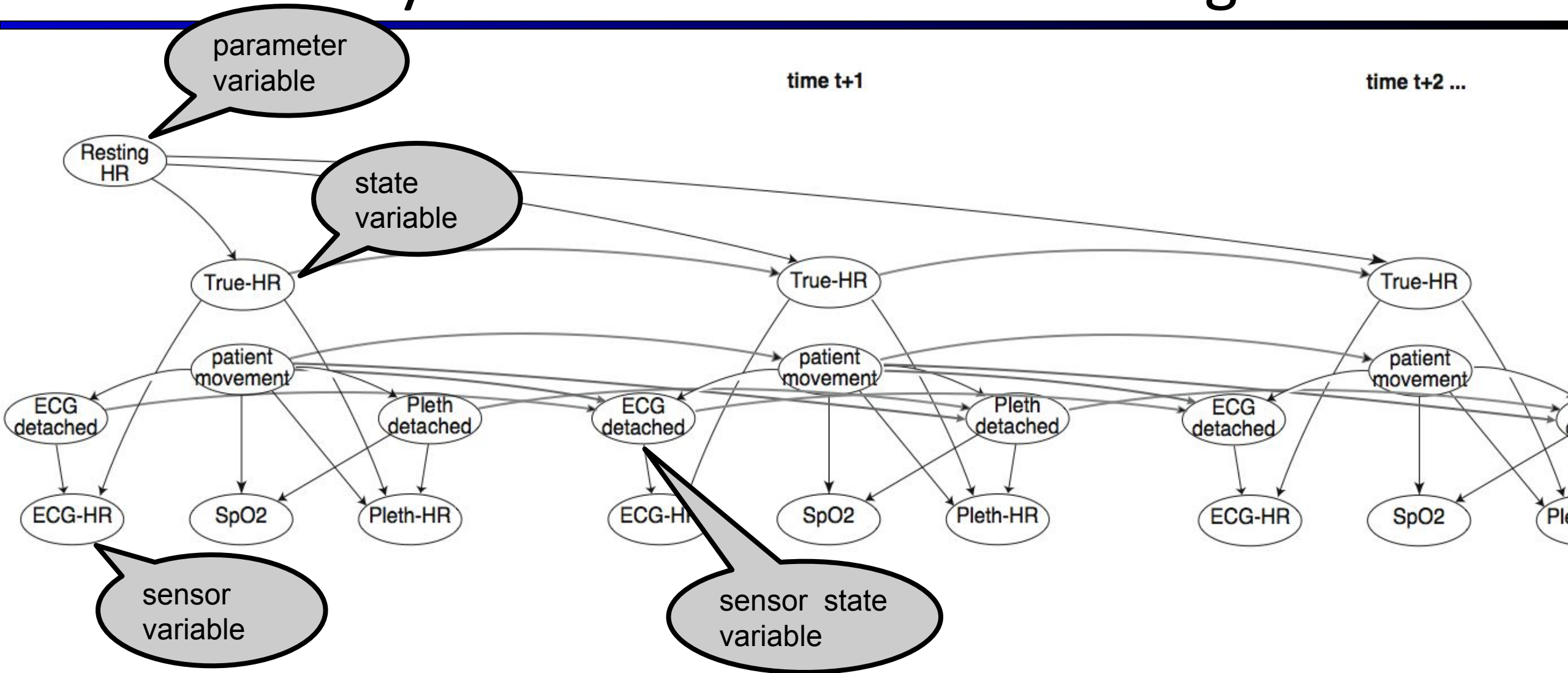
- Online: eliminate all variables from the previous time step; store factors for current time only
- Problem: largest factor contains all variables for current time (plus a few more), so it fails for models with many state variables and evidence variables



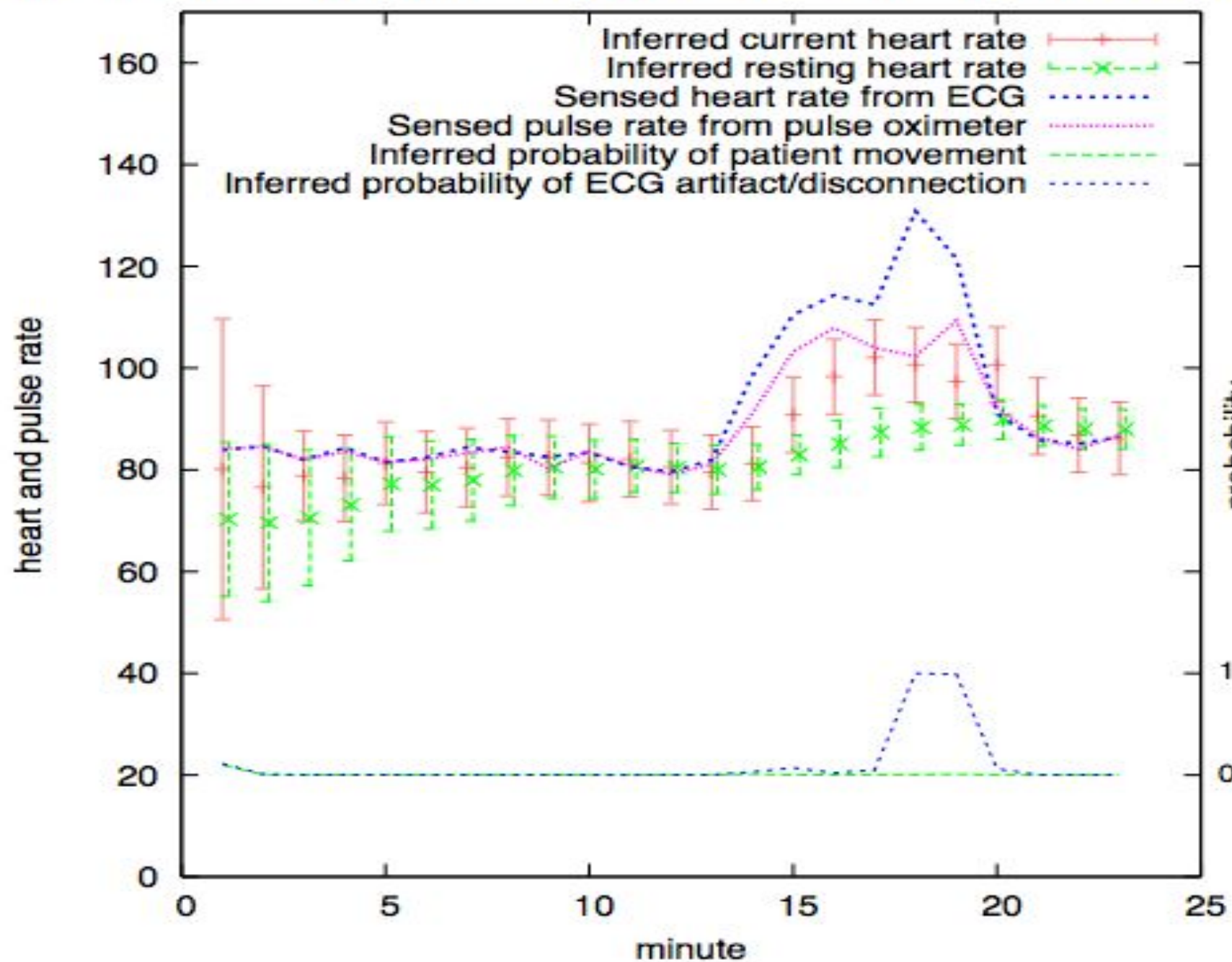
Application: ICU monitoring

- ***State***: variables describing physiological state of patient
- ***Evidence***: values obtained from monitoring devices
- ***Transition model***: physiological dynamics, sensor dynamics
- ***Query variables***: pathophysiological conditions (a.k.a. bad things)

Toy DBN: heart rate monitoring

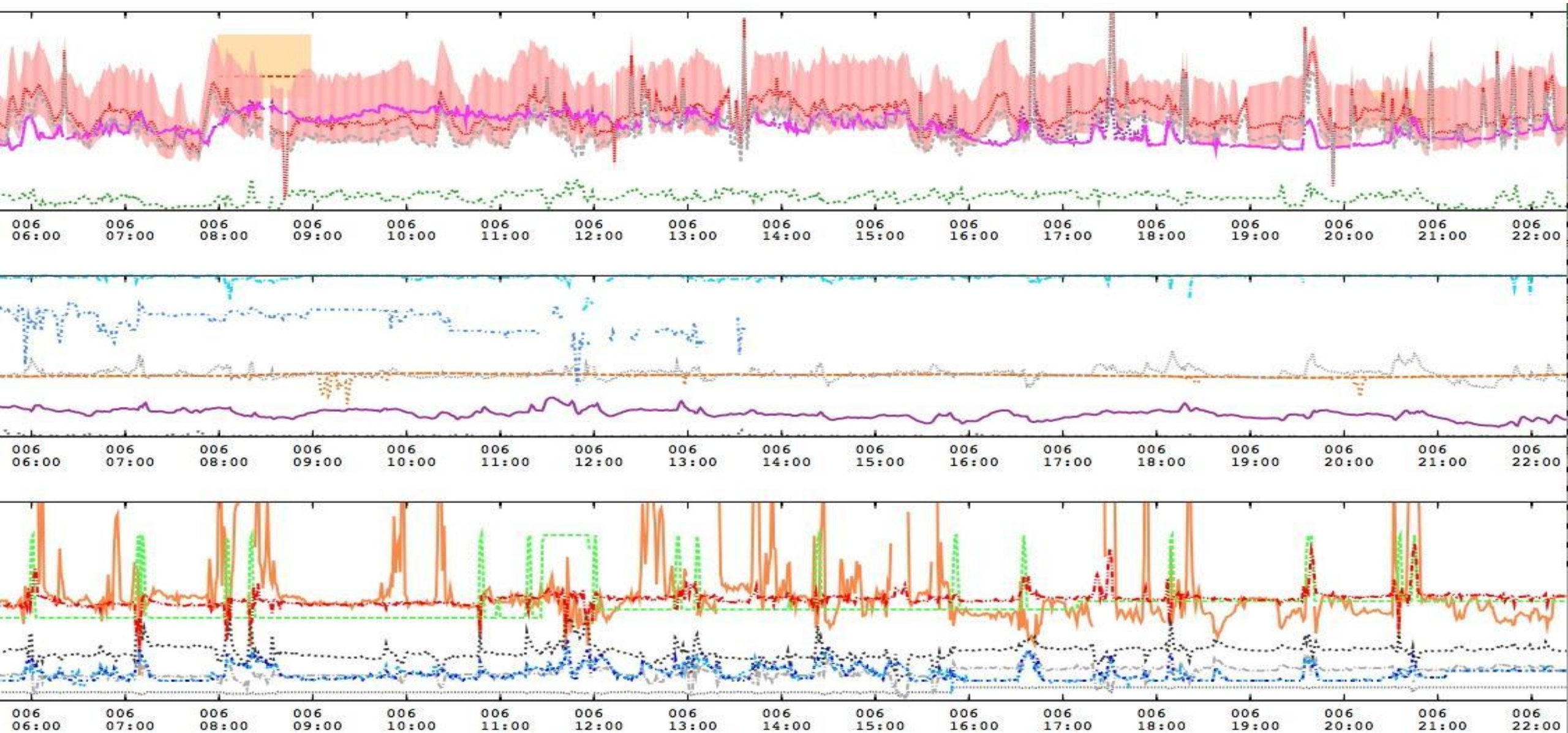


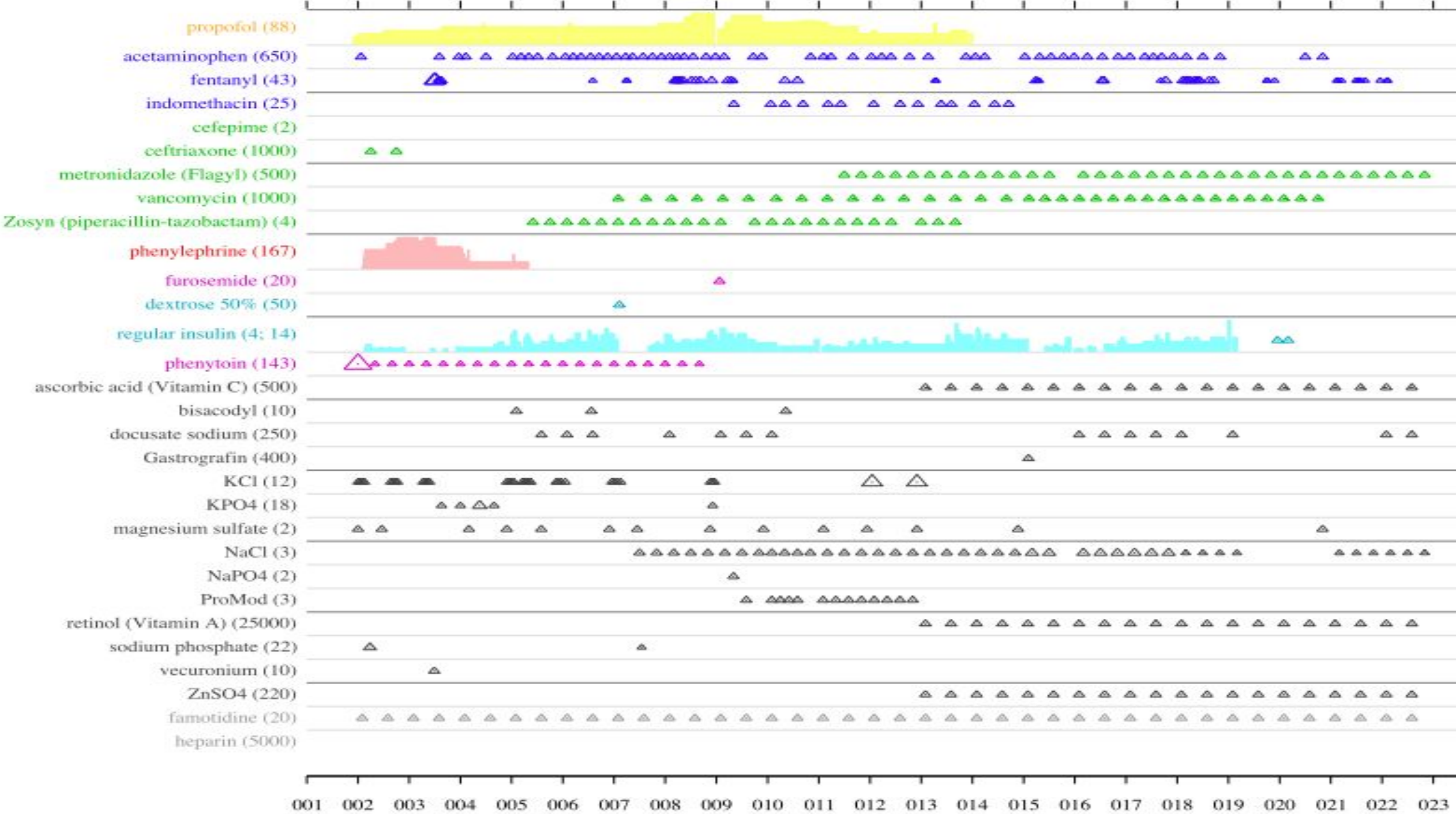
The enhanced heart-rate DBN's inferences on data from a healthy 40-year-old



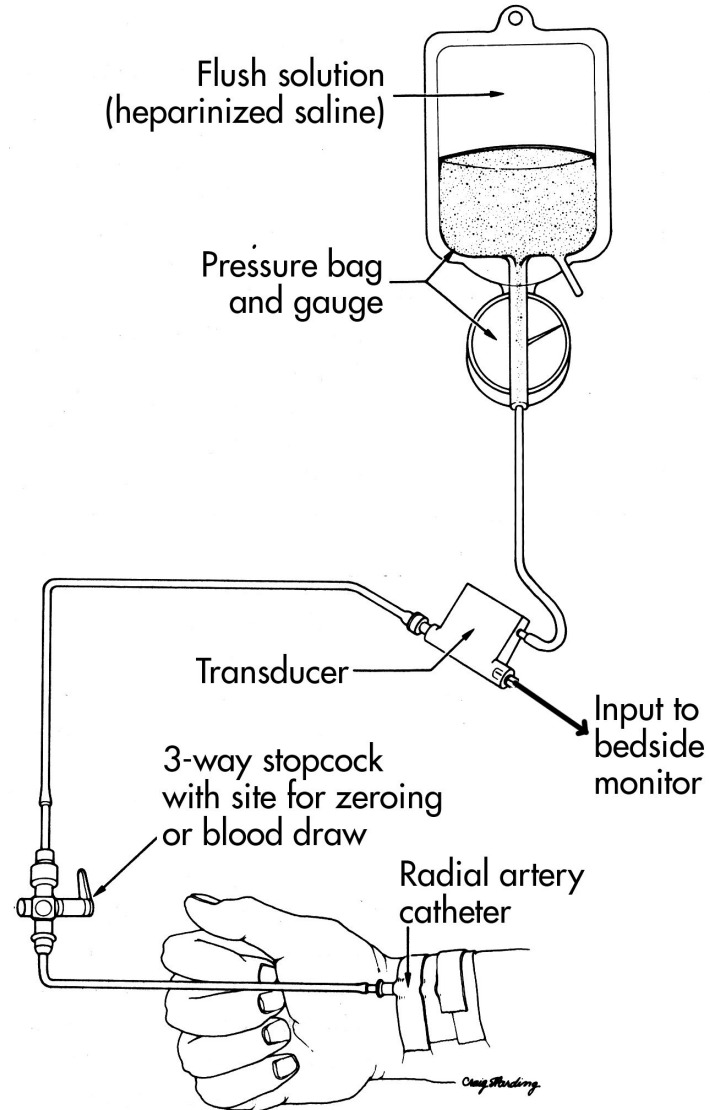
probability

ICU data: 22 variables, 1min ave

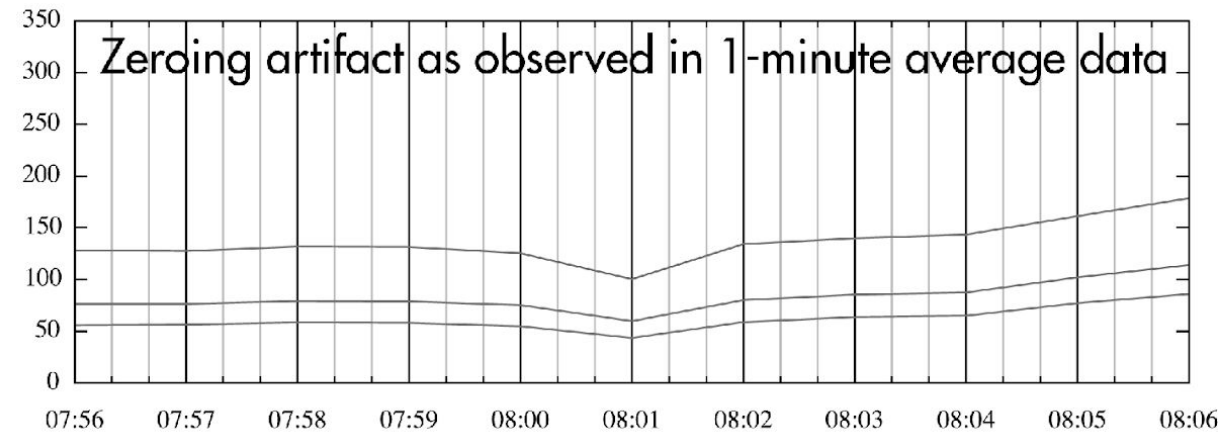
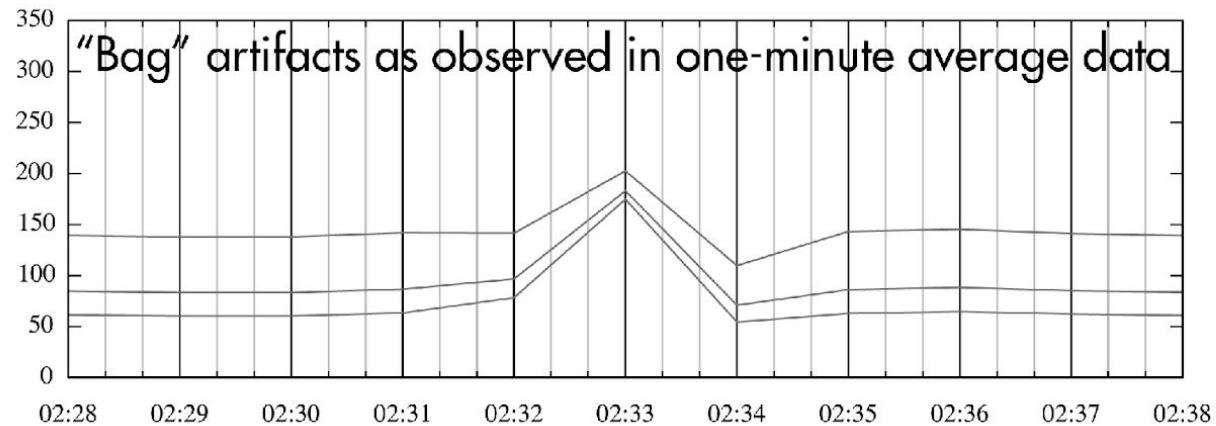
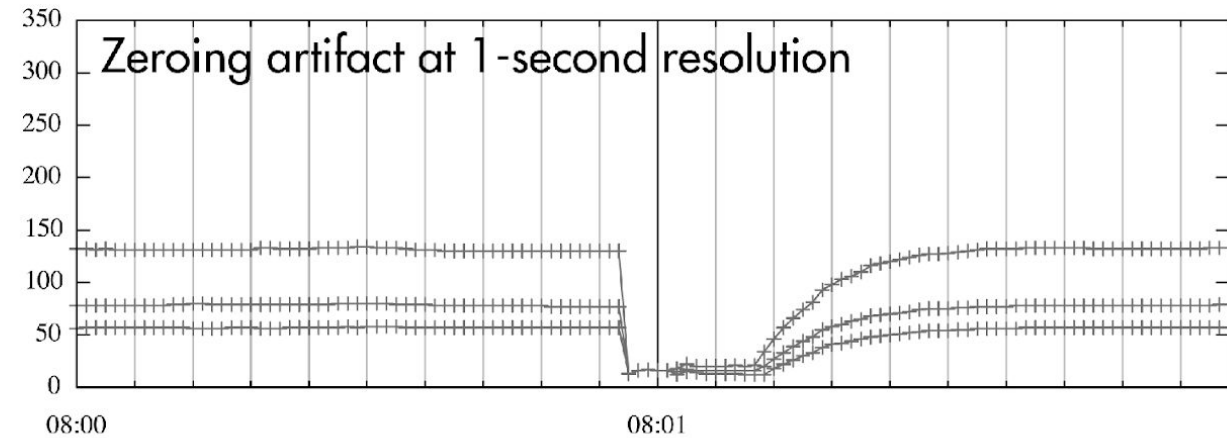
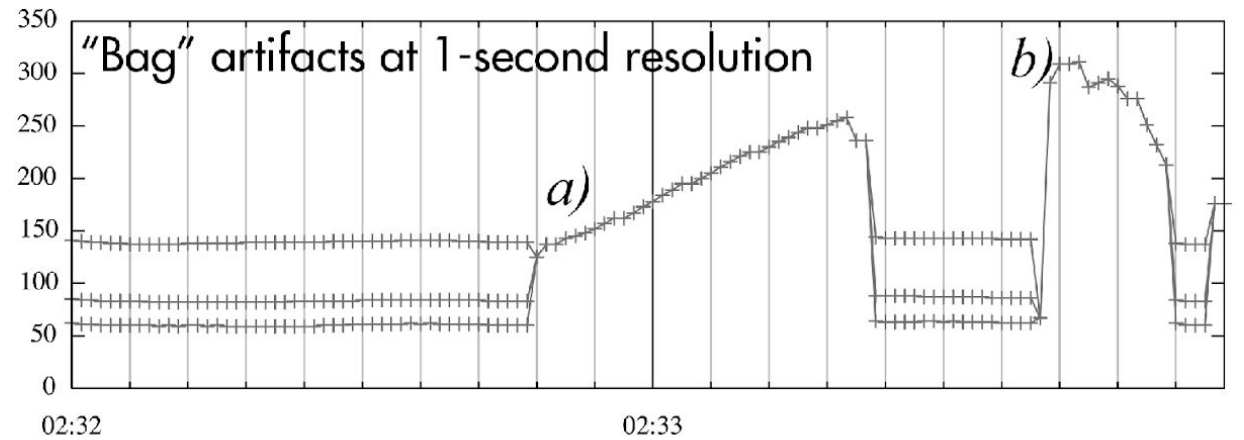


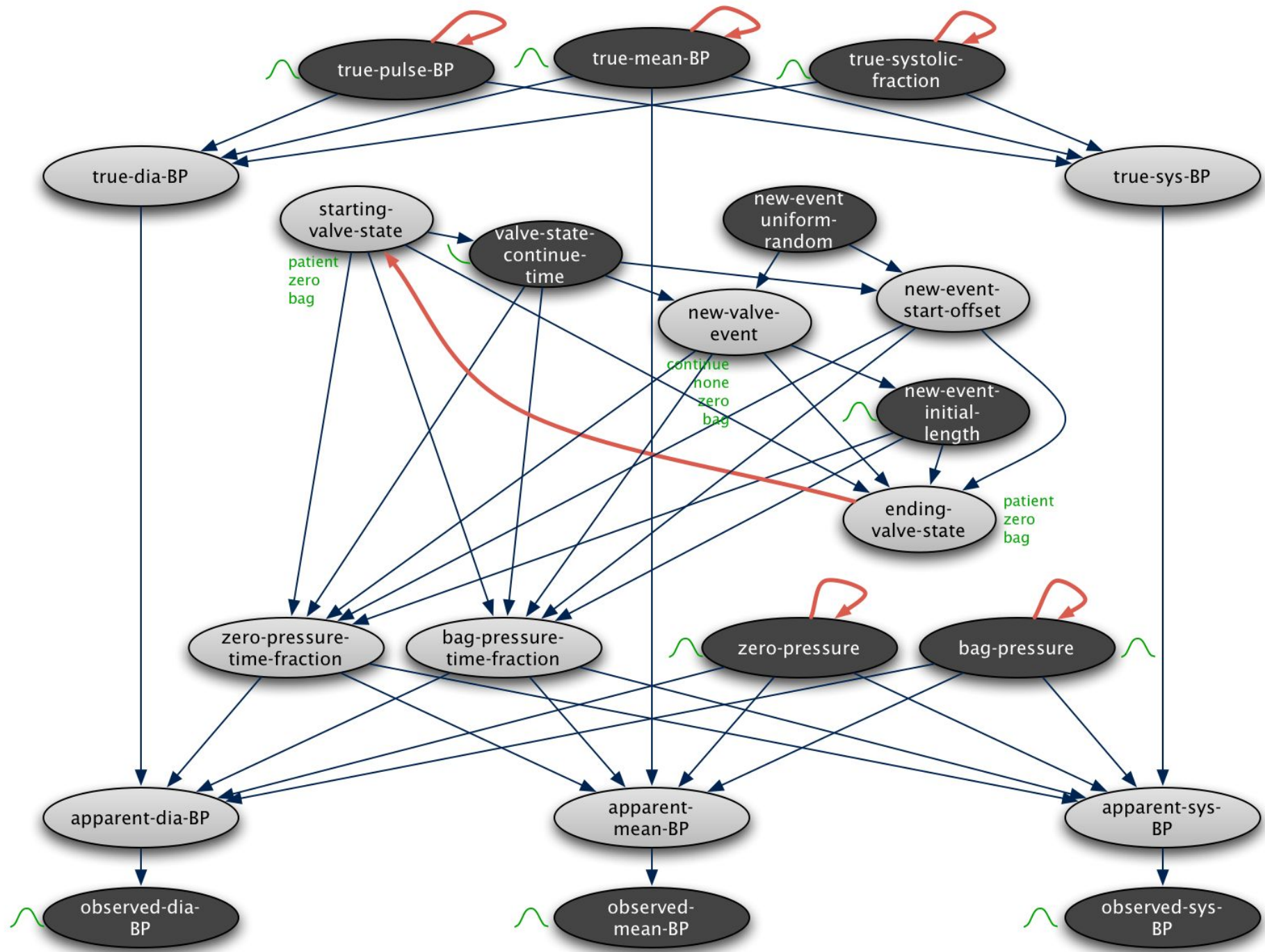


Blood pressure measurement

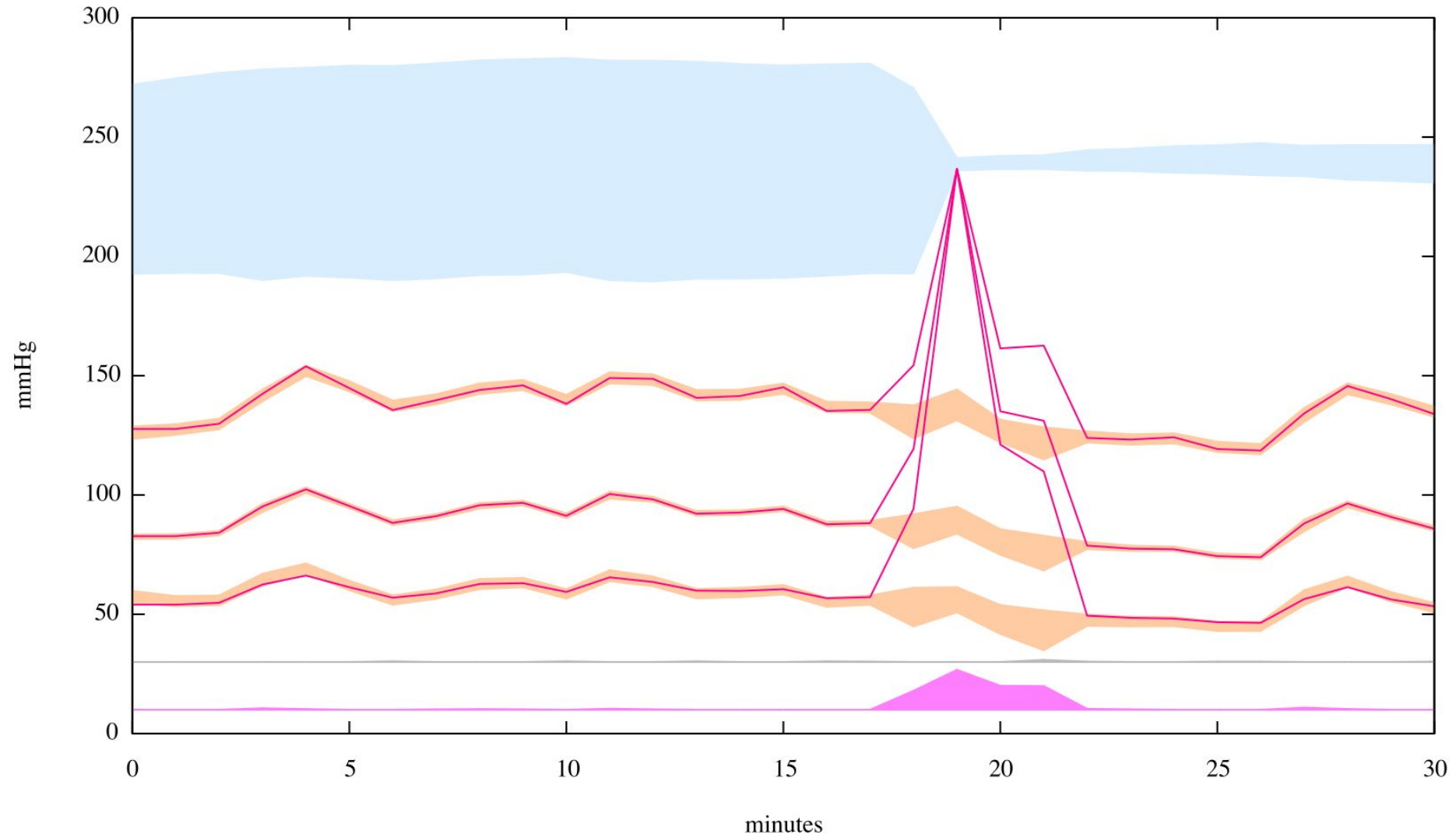




One-second vs one-minute data







Sample blood-draw dataset no. 11

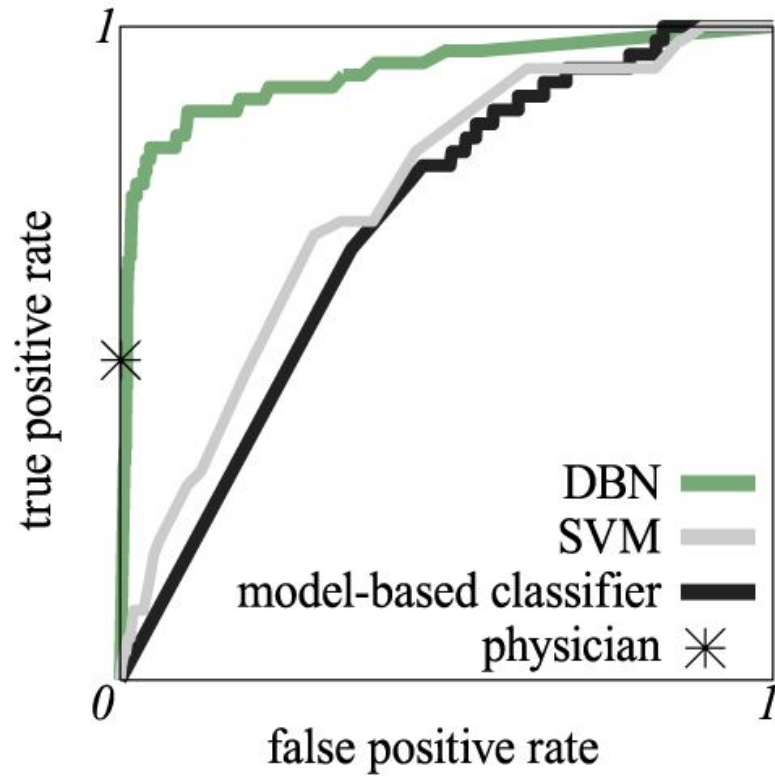


bag pressure estimate 
valve open to bag 

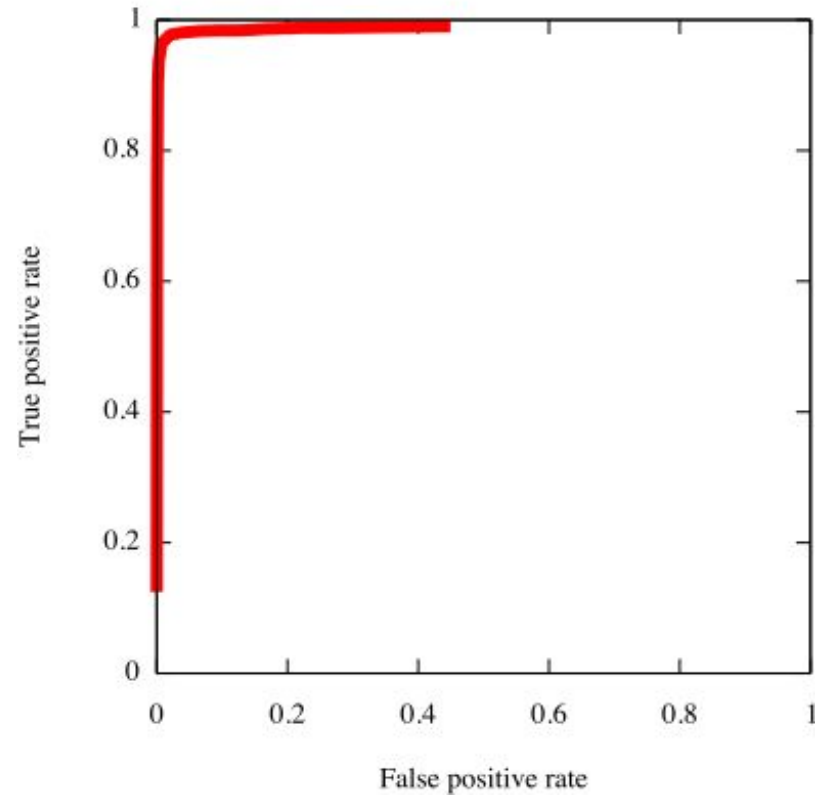
valve open to air 
BP estimate 

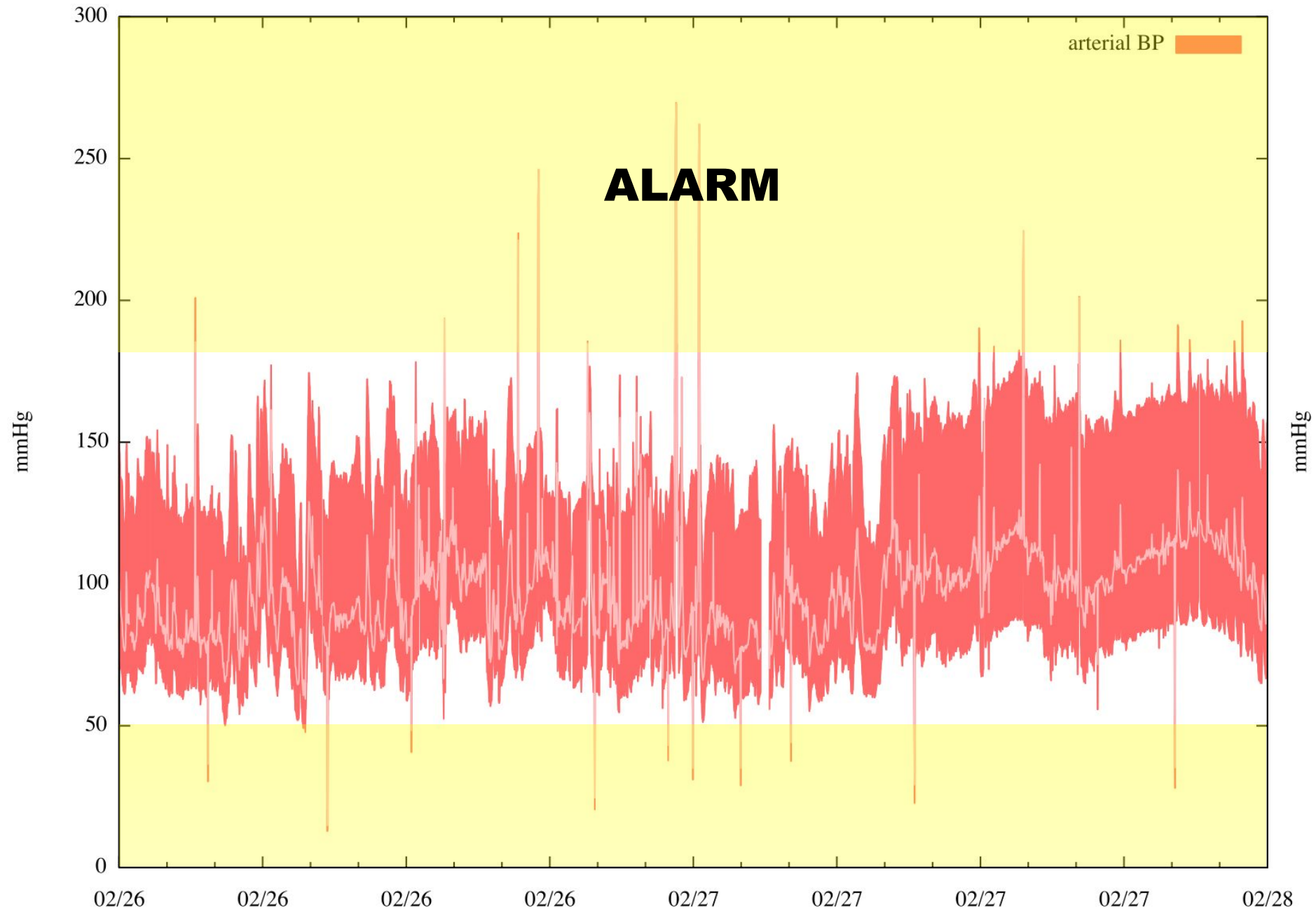
observed BP 

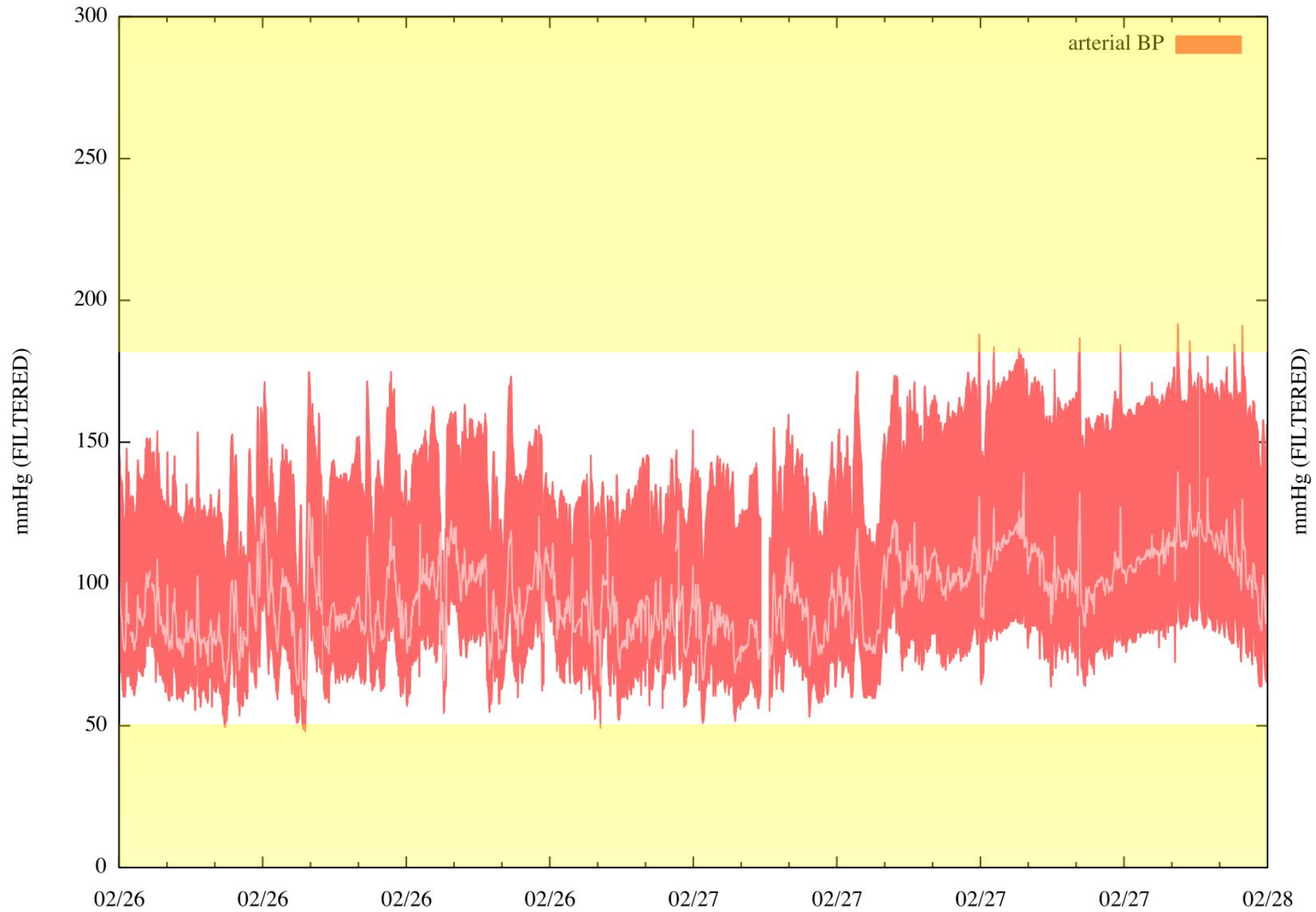
Detection of “bag” events



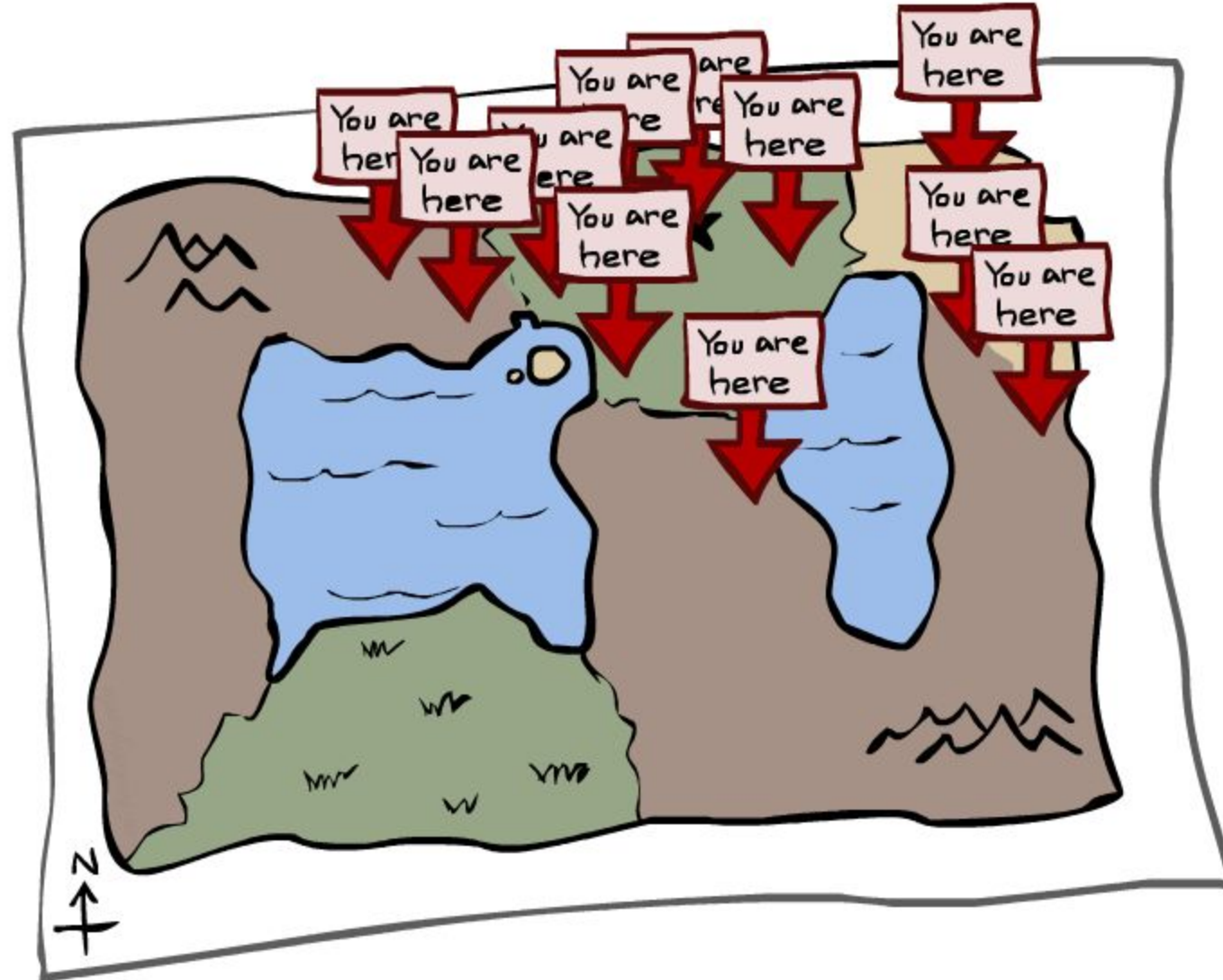
ROC curve for hypertension detection (SBP>160mmHg)





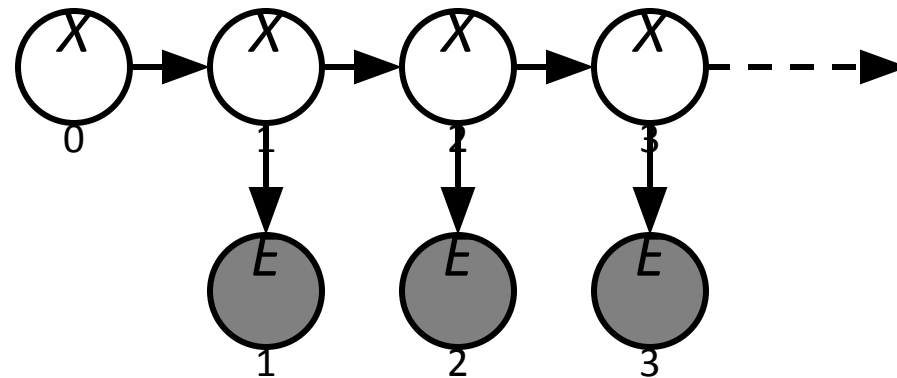
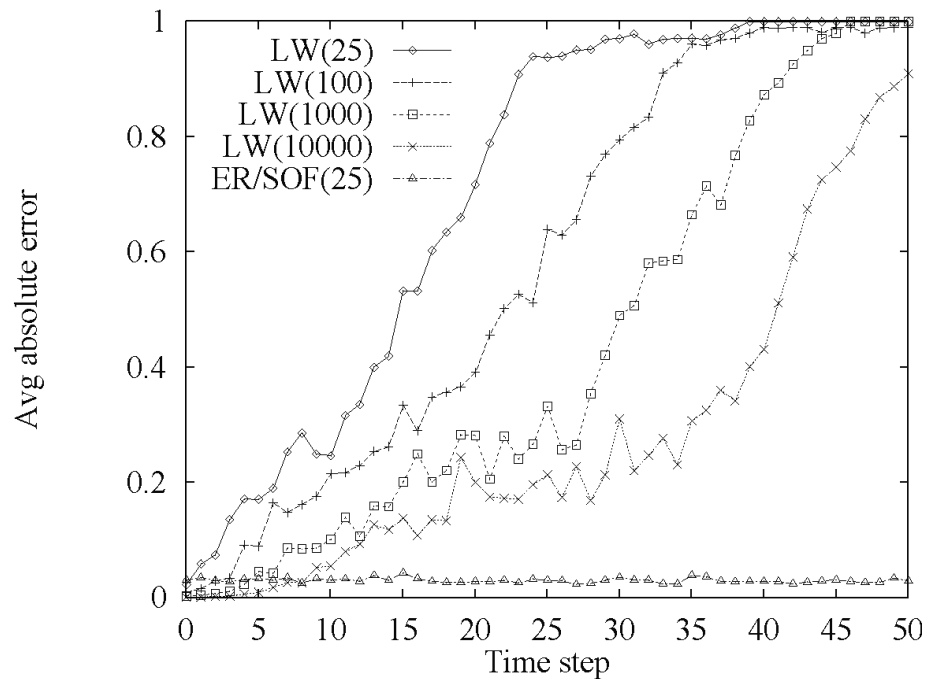


Particle Filtering

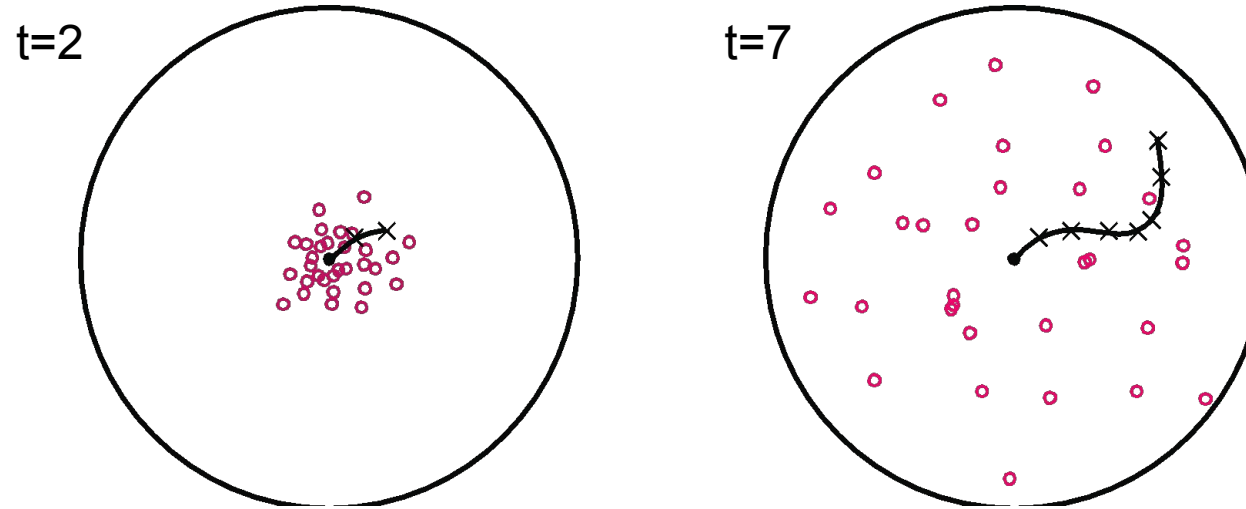


We need a new algorithm!

- When $|X|$ is more than 10^6 or so (e.g., 3 ghosts in a 10x20 world), exact inference becomes infeasible
- Likelihood weighting fails completely – number of samples needed grows *exponentially* with T



We need a new idea!

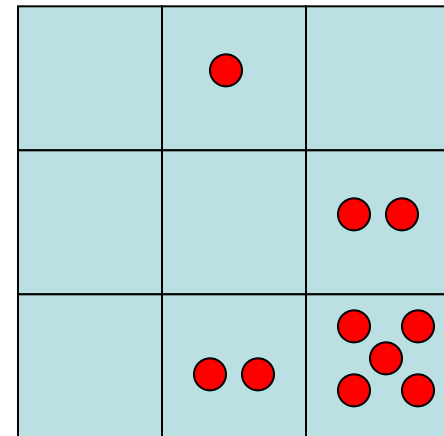


- The problem: sample state trajectories go off into low-probability regions, ignoring the evidence; too few “reasonable” samples
- Solution: kill the bad ones, make more of the good ones
- This way the population of samples stays in the high-probability region
- This is called *resampling* or survival of the fittest

Particle Filtering

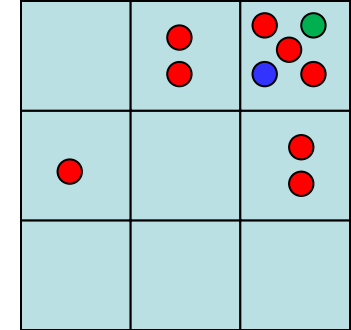
- Represent belief state by a set of samples
 - Samples are called *particles*
 - Time per step is linear in the number of samples
 - But: number needed may be large in worst case
- This is how robot localization works in practice

0	0.1	0
0	0	0.2
0	0.2	0.5



Representation: Particles

- Our representation of $P(X)$ is now a list of $N \ll |X|$ particles
- $P(x)$ approximated by number of particles with value x
 - So, many x may have $P(x) = 0$!
 - More particles => more accuracy (cf. frequency histograms)
 - Usually we want a **low-dimensional** marginal
 - E.g., “Where is ghost 1?” rather than “Are ghosts 1,2,3 in [2,6], [5,6], and [8,11]?”



Particles:

(3,3)

(2,3)

(3,3)

(3,2)

(3,3)

(3,2)

(1,2)

(3,3)

(3,3)

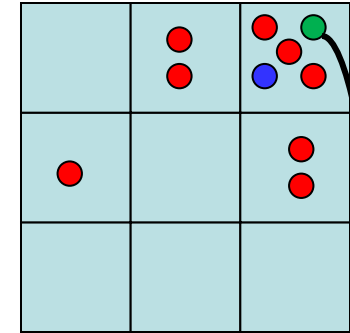
(2,3)

Particle Filtering: Prediction step

- Particle j in state $x_t^{(j)}$ samples a new state directly from the transition model:
 - $x_{t+1}^{(j)} \sim P(X_{t+1} | x_t^{(j)})$
 - Here, most samples move clockwise, but some move in another direction or stay in place

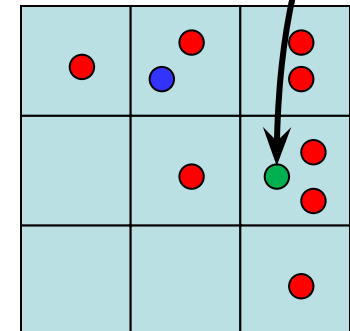
Particles:

(3,3)
(2,3)
(3,3)
(3,2)
(3,3)
(3,2)
(1,2)
(3,3)
(3,3)
(2,3)



Particles:

(3,2)
(2,3)
(3,2)
(3,1)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(2,2)

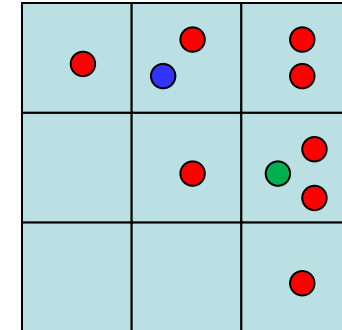


Particle Filtering: Update step

- After observing e_{t+1} :
 - As in likelihood weighting, weight each sample based on the evidence
 - $w^{(j)} = P(e_{t+1} | x_{t+1}^{(j)})$
 - Particles that fit the data better get higher weights, others get lower weights
 - Normalize the weights across all particles

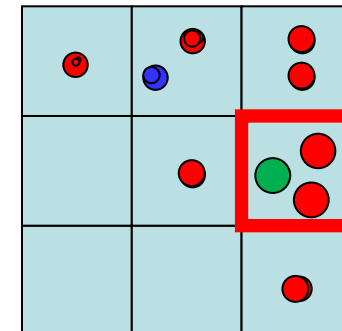
Particles:

(3,2)
 (2,3)
 (3,2)
 (3,1)
 (3,3)
 (3,2)
 (1,3)
 (2,3)
 (3,2)
 (2,2)



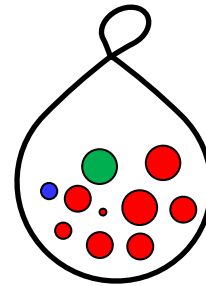
Particles:

(3,2) $w=\mathbf{X}$.17
 (2,3) $w=\mathbf{X}$.04
 (3,2) $w=\mathbf{X}$.17
 (3,1) $w=\mathbf{X}$.08
 (3,3) $w=\mathbf{X}$.08
 (3,2) $w=\mathbf{X}$.17
 (1,3) $w=\mathbf{X}$.02
 (2,3) $w=\mathbf{X}$.04
 (3,2) $w=\mathbf{X}$.17
 (2,2) $w=\mathbf{X}$.08



Particle Filtering: Resample

- Rather than tracking weighted samples, we *resample*
- N times, we choose from our weighted sample distribution (i.e., draw with replacement)
- Now the update is complete for this time step, continue with the next one (with weights reset to $1/N$)

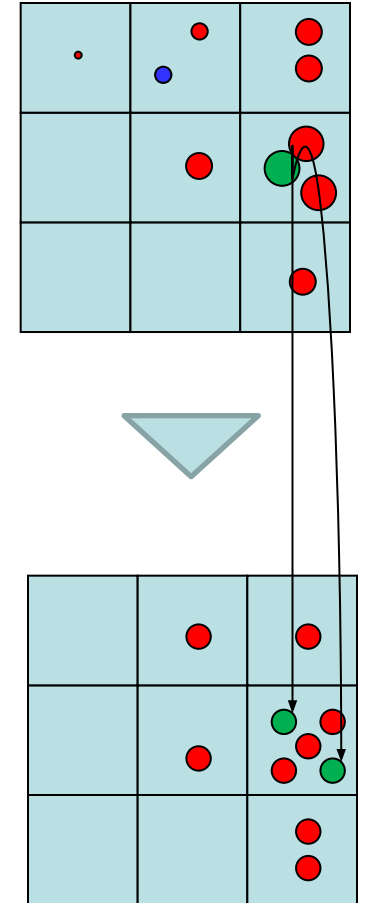


Particles:

(3,2) $w=.17$
(2,3) $w=.04$
(3,2) $w=.17$
(3,1) $w=.08$
(3,3) $w=.08$
(3,2) $w=.17$
(1,3) $w=.02$
(2,3) $w=.04$
(3,2) $w=.17$
(2,2) $w=.08$

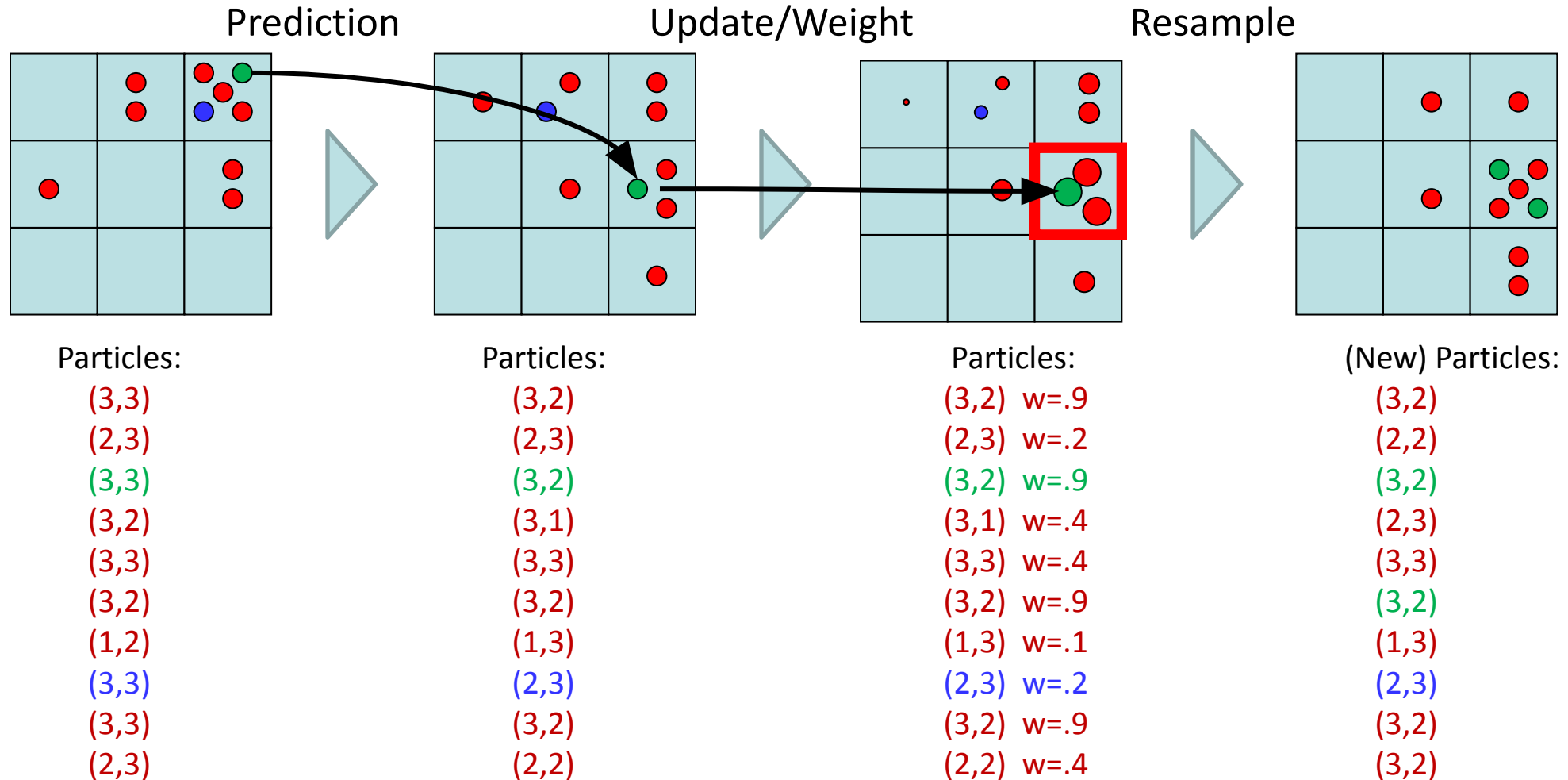
(New) Particles:

(3,2)
(2,2)
(3,2)
(2,3)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(3,2)

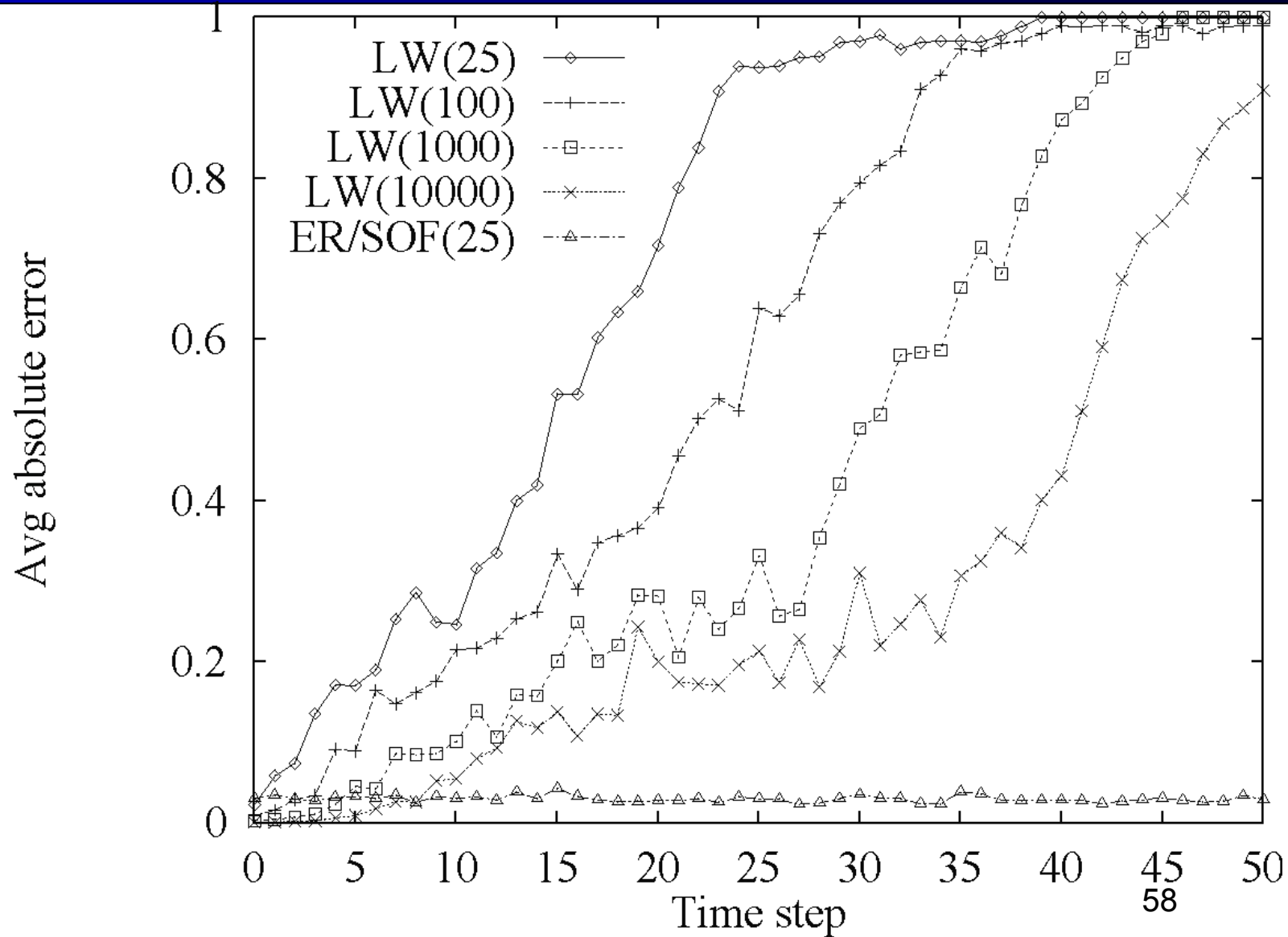


Summary: Particle Filtering

- Particles: track samples of states rather than an explicit distribution

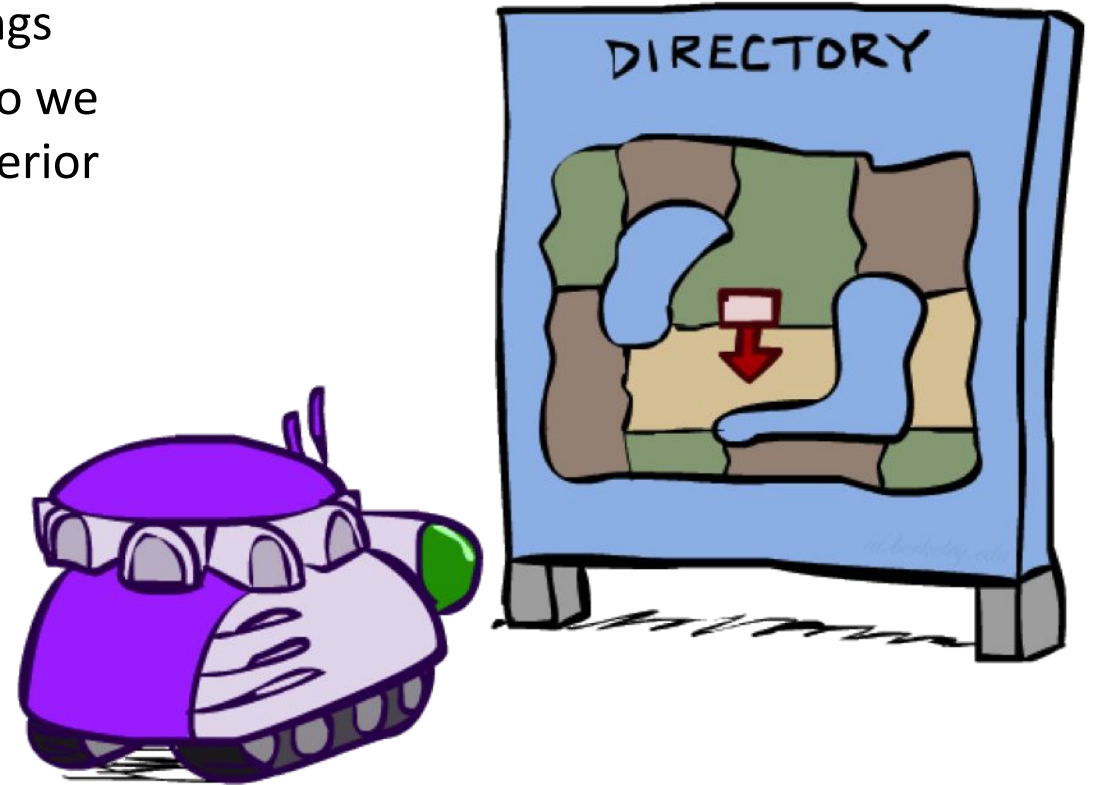
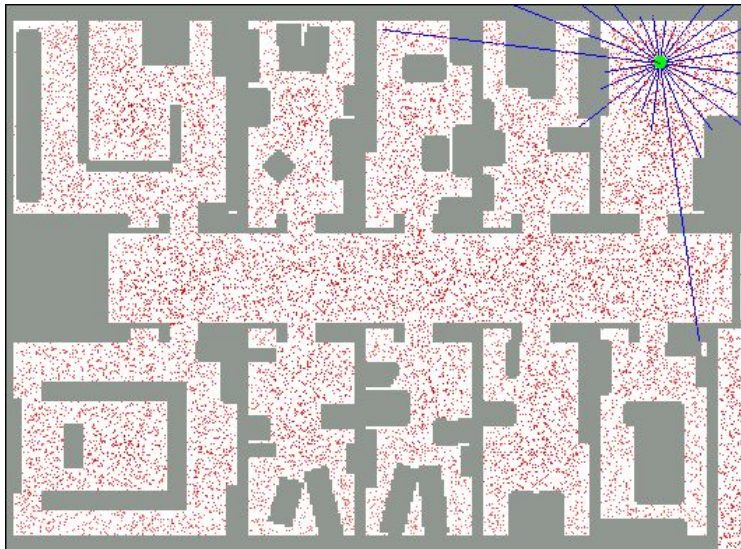


Particle filtering on umbrella model



Robot Localization

- In robot localization:
 - We know the map, but not the robot's position
 - Observations may be vectors of range finder readings
 - State space and readings are typically continuous so we cannot usually represent or compute an exact posterior
 - Particle filtering is a main technique

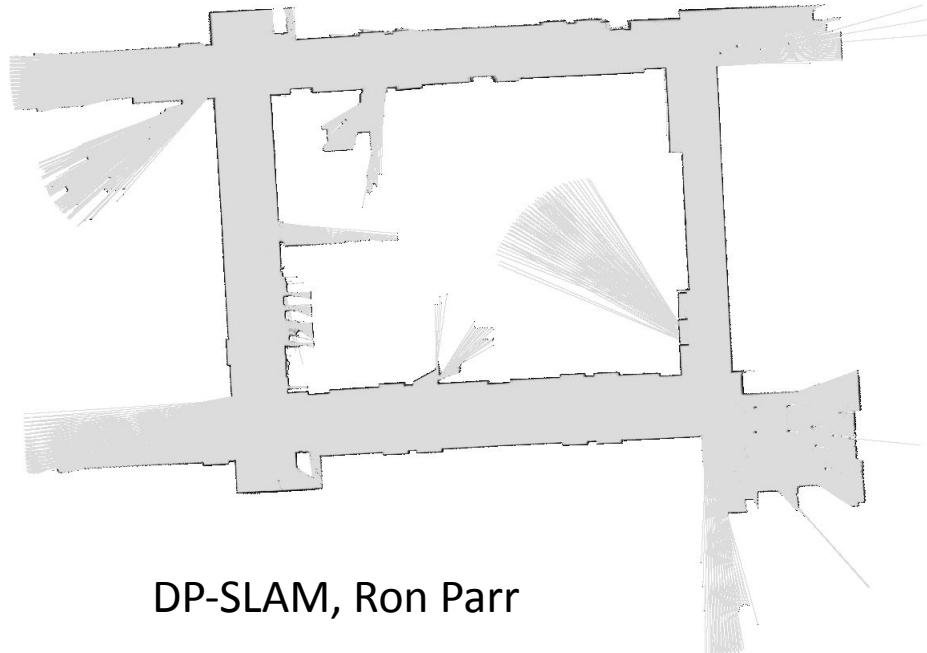


Particle Filter Localization (Sonar)

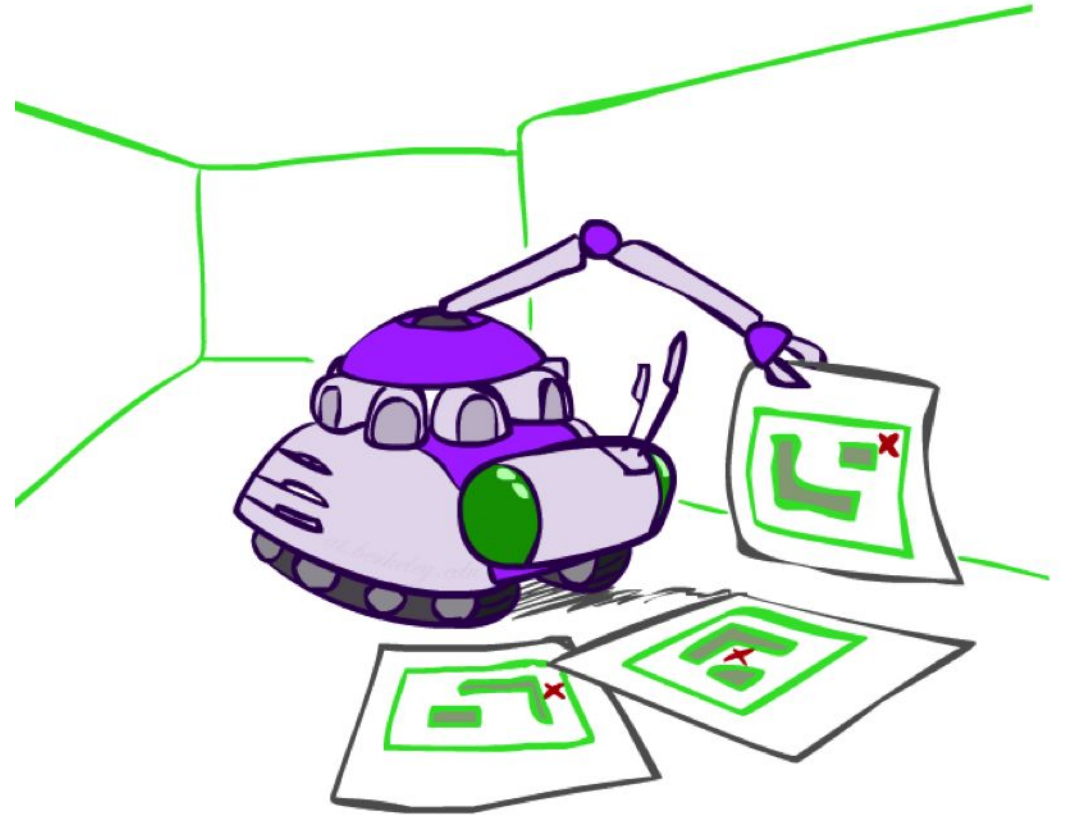


Robot Mapping

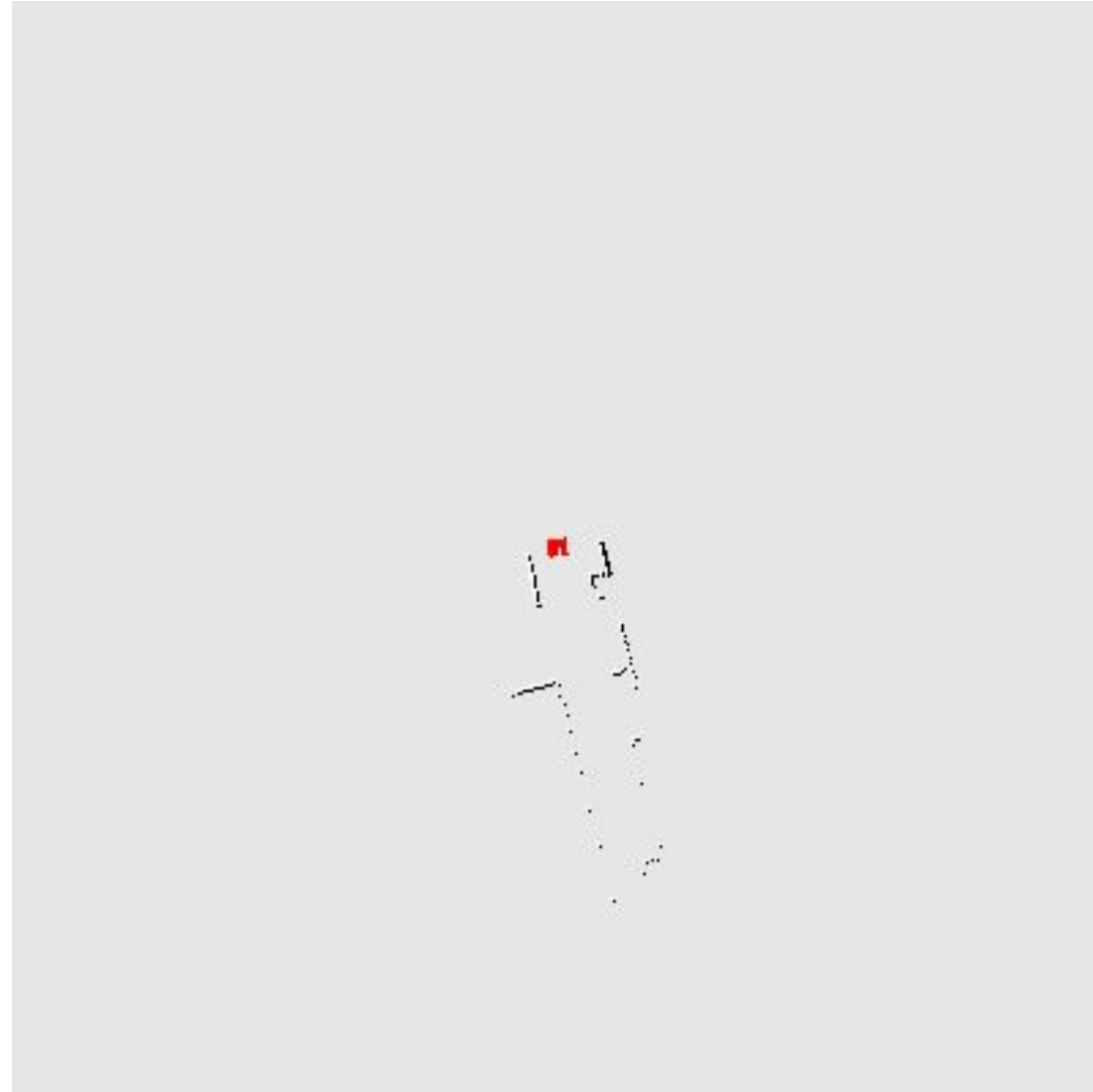
- SLAM: Simultaneous Localization And Mapping
 - Robot does not know map or location
 - State $x_t^{(j)}$ consists of [**position, orientation, map**]!
 - (Each map usually inferred exactly given sampled position+orientation sequence: RBPF)



DP-SLAM, Ron Parr



Particle Filter SLAM – Video 1



Particle Filter SLAM – Video 2

