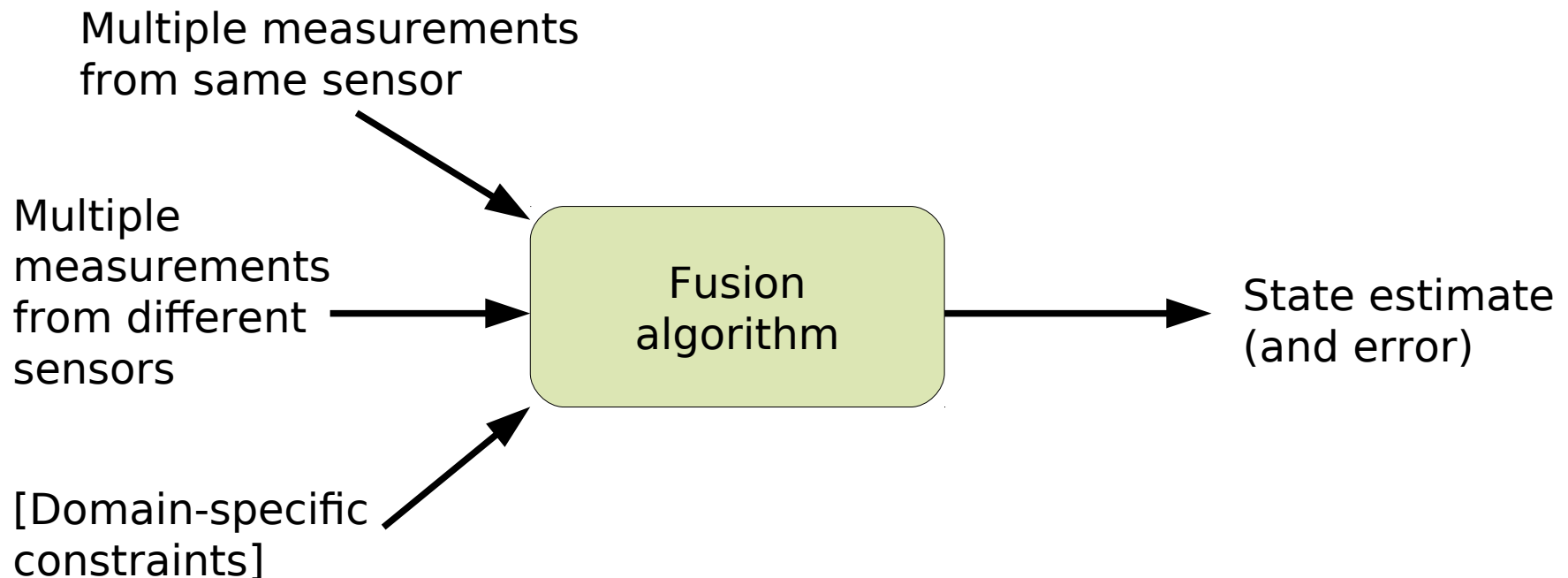# Sensor Fusion Techniques

Dr Robert Harle

Part II Mobile and Sensor Systems

Lent 2017/18

Measurements are Noisy

- A sensor measures some quantity with some accuracy. Whatever we do, noise will creep in

- We therefore need to fuse multiple measurements to get a robust idea of what's happening

Multiple measurements
from same sensor

Multiple
measurements
from different
sensors

Fusion
algorithm

State estimate
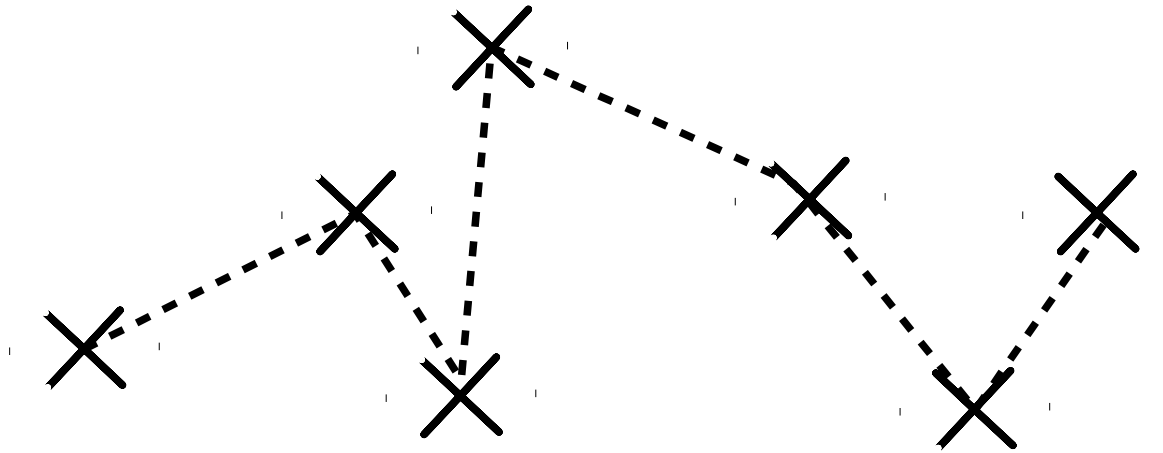(and error)

[Domain-specific
constraints]

Algorithms

- There are many fusion techniques and algorithms

- We will look at the two extremes: a very fast, very common algorithm that is limited in what it works with, and a general-purpose and flexible but more computationally demanding algorithm

- Both are based on bayesian probability

- We will use location tracking to illustrate the techniques because the problem is easy to relate to. **But everything is general.**
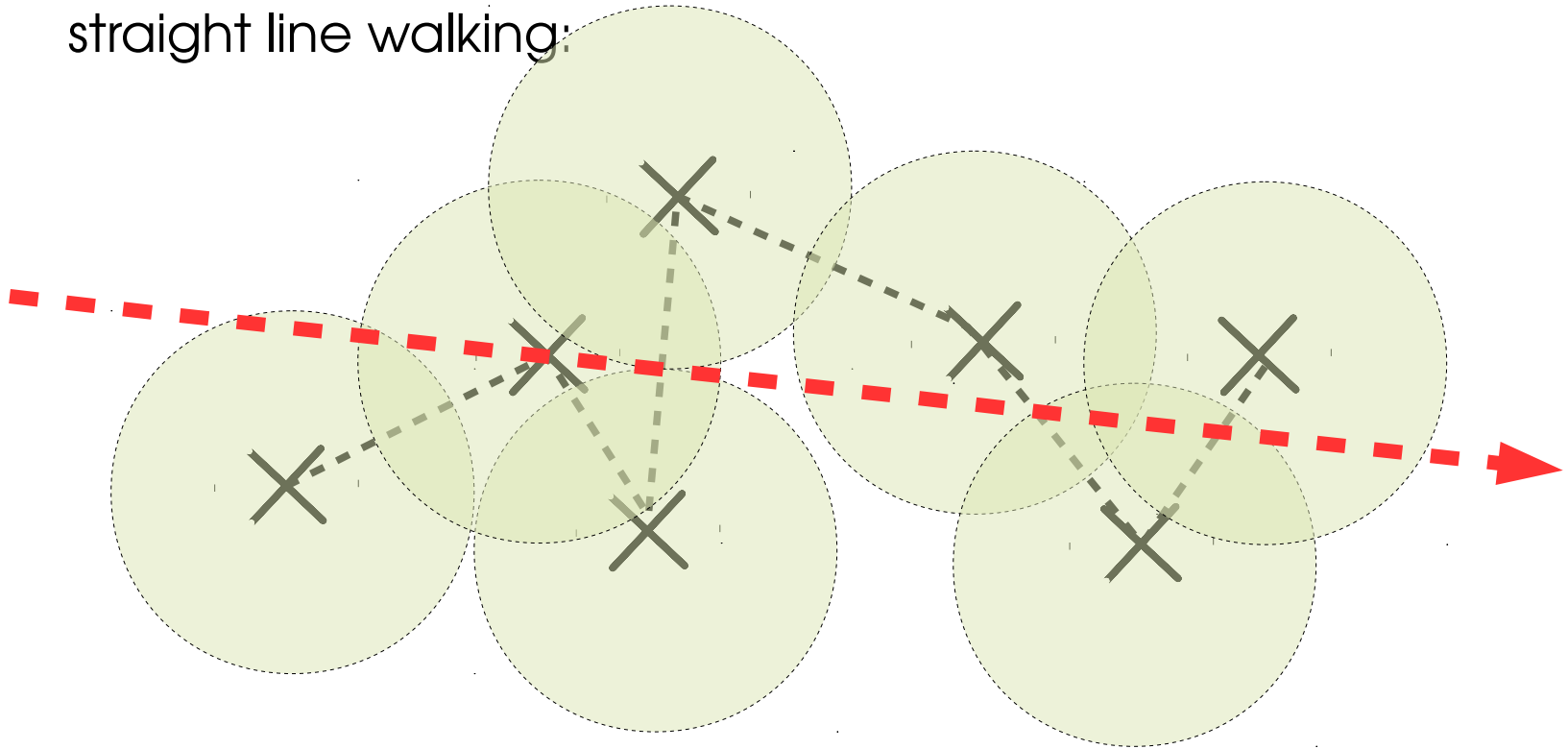
Simple Tracking Example

Consider a series of positions that come in a few seconds apart for a pedestrian. They will probably look rather unrealistic for a walking route:
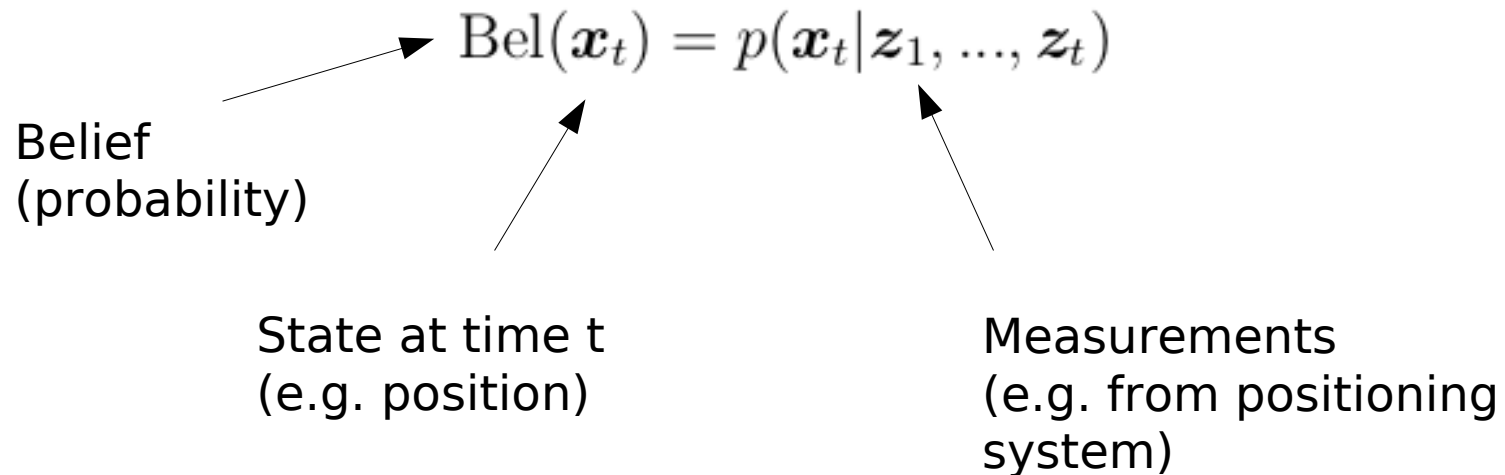
Simple Tracking Example

But if we consider noise and error in the measurements we see that the data supports a more realistic hypothesis of straight line walking:

Probabilistic Approach

So what we want to do is to estimate our current state while incorporating knowledge of recent measurements and all of the associated errors. To do this we will use probability:

$$\text{Bel}(\boldsymbol{x}_t) = p(\boldsymbol{x}_t | \boldsymbol{z}_1, ..., \boldsymbol{z}_t)$$

Belief
(probability)

State at time t
(e.g. position)

Measurements
(e.g. from positioning
system)

Filters and Smoothers

$$\text{Bel}(\boldsymbol{x}_t) = p(\boldsymbol{x}_t | \boldsymbol{z}_1, ..., \boldsymbol{z}_t)$$

- This is known as a filter because it estimates the current state based on current and past measurements (only)

- Sometimes you know the 'future' e.g. you may have logged data for postprocessing rather than live processing

- In that case you have a *smoother*

Recursive Bayesian Filters

- Apply a Markov model (next state depends only on last) to recursively build up our probabilities

$$\mathrm{Bel}^-(\boldsymbol{x}_t) = \int p(\boldsymbol{x}_t | \boldsymbol{x}_{t-\delta t}) \mathrm{Bel}(\boldsymbol{x}_{t-\delta t}) \, d\boldsymbol{x}_{t-\delta t}.$$

Prior

Propagation (motion) model

Evaluate over all previous states

- This is the **propagation** or **prediction** step
- We update the probabilities based on some model (e.g. constant velocity) → prior distribution

Recursive Bayesian Filters

- Apply Bayes' theorem to incorporate measurements

$$\text{Bel}(\boldsymbol{x}_t) = p(\boldsymbol{x}_t | \boldsymbol{z}_1, ..., \boldsymbol{z}_t)$$

$$\text{Bel}(\boldsymbol{x}_t) = \alpha_t p(\boldsymbol{z}_t | \boldsymbol{x}_t) \text{Bel}^-(\boldsymbol{x}_t)$$

Posterior

Norm factor

Measurement model

Prior
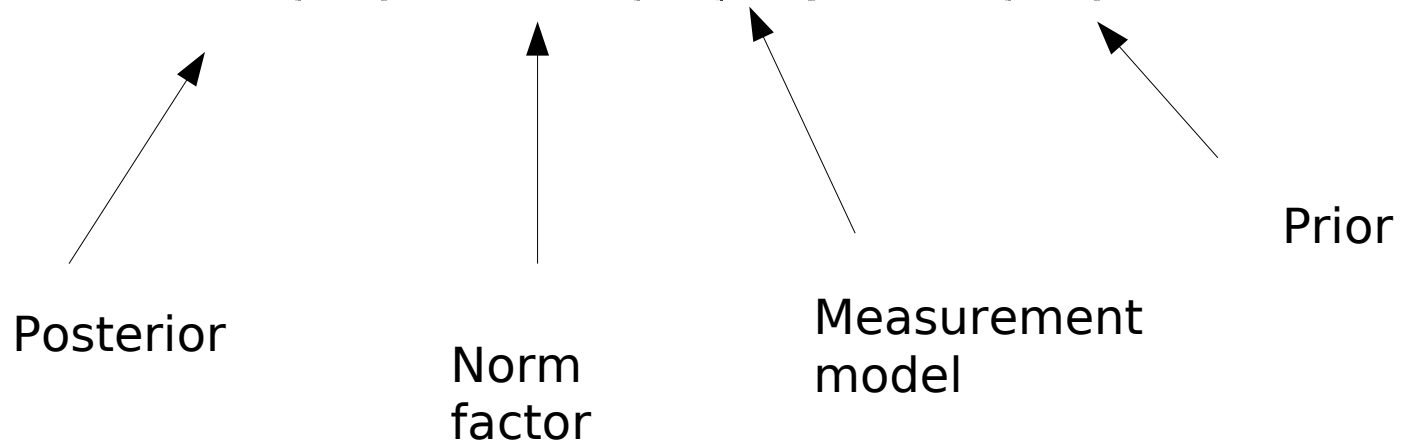
- This is the **correction** or **update** step
- We correct the probabilities on a measurement → posterior distribution

Implementation

$$\mathrm{Bel}^-(\boldsymbol{x}_t) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-\delta t})\mathrm{Bel}(\boldsymbol{x}_{t-\delta t})\,d\boldsymbol{x}_{t-\delta t}.$$ Propagation/predict

$$\mathrm{Bel}(\boldsymbol{x}_t) = \alpha_t p(\boldsymbol{z}_t|\boldsymbol{x}_t)\mathrm{Bel}^-(\boldsymbol{x}_t)$$ Correction/update

- There are broadly two classes of techniques to implement these "filters"

  1) Model all the probability distributions using mathematical models. This keeps everything continuous. But it's not always easy to do this (the distributions get complex). E.g. Use Gaussians everywhere → "Kalman Filter"

  2) Represent arbitrary distributions by sampling them. Nice and general but much more work involved.

The Kalman Filter

- Motion along a straight path

- State vector $(x, dx/dt)^T$
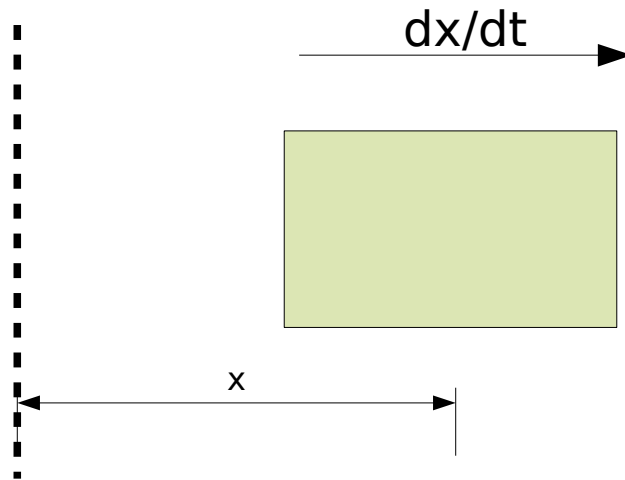
The Kalman Filter

- The simplest recursive Bayesian filter

- It is used *everywhere:* very important

- Requires that you can write the dynamics of your system using linear algebra (matrices etc)

- Boils down to 3 equations:

Current state

$$x_t = F_t x_{t-\delta t} + w_t$$

$$P_t^- = F_t P_{t-\delta t} F_t^T + Q_t$$

Propagation

$$z_t = H_t x_t + v_t$$
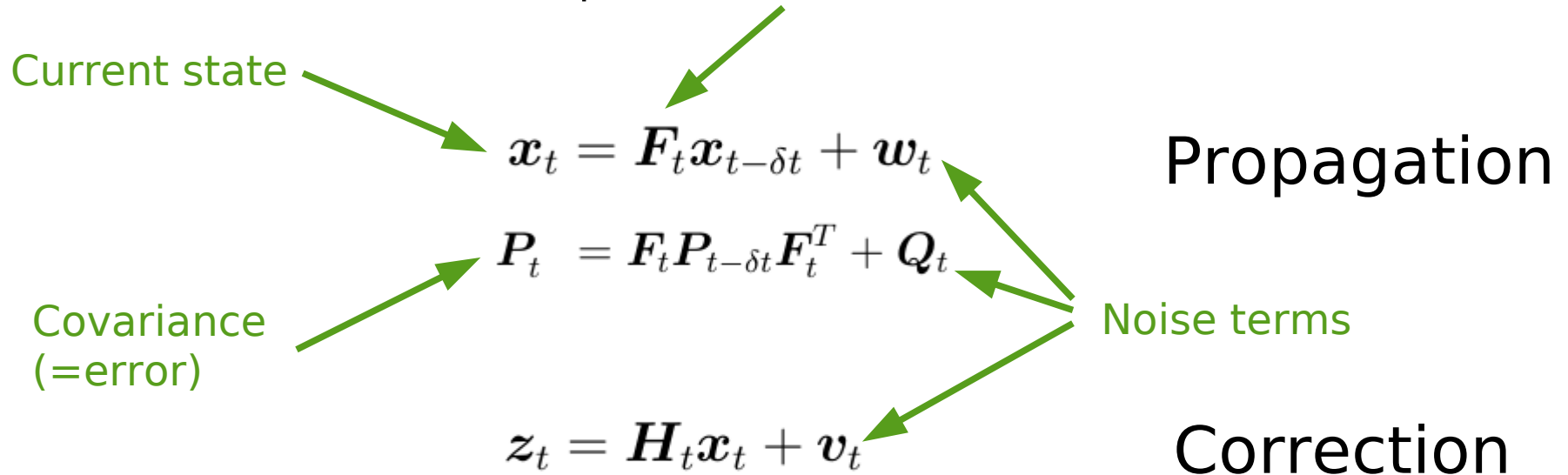
Correction

The Kalman Filter

- The simplest recursive Bayesian filter

- It is used *everywhere:* very important

- Requires that you can write the dynamics of your system using linear algebra (matrices etc)

- Boils down to 3 equations:

Encodes the motion model

Current state

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-\delta t} + \boldsymbol{w}_t$$

Propagation

$$\boldsymbol{P}_t^- = \boldsymbol{F}_t \boldsymbol{P}_{t-\delta t} \boldsymbol{F}_t^T + \boldsymbol{Q}_t$$

$$\boldsymbol{z}_t = \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{v}_t$$

Correction

Simple example

- Motion model (constant velocity)

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-\delta t} + \boldsymbol{w}_t$$

$$F = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}$$
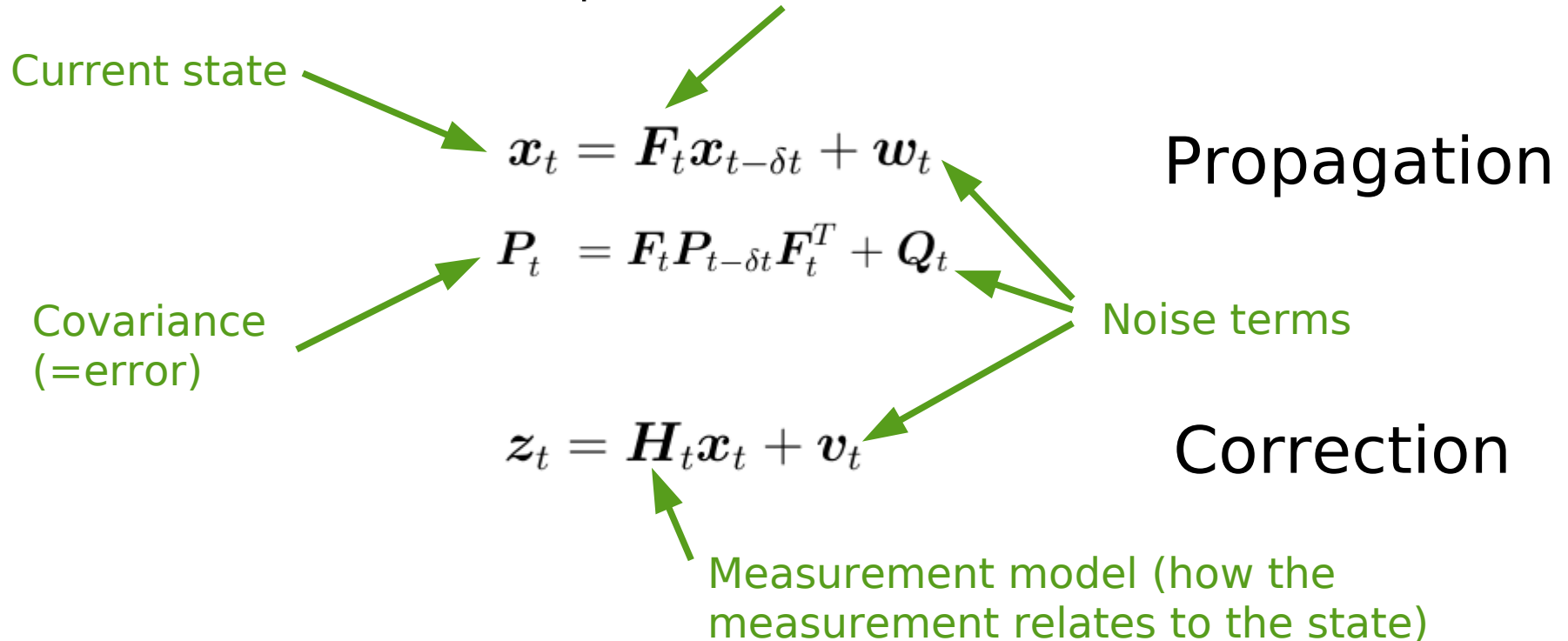
The Kalman Filter

- The simplest recursive Bayesian filter

- It is used *everywhere:* very important

- Requires that you can write the dynamics of your system using linear algebra (matrices etc)

- Boils down to 3 equations: Encodes the motion model

Current state

$$x_t = F_t x_{t-\delta t} + w_t$$

Propagation

$$P_t^- = F_t P_{t-\delta t} F_t^T + Q_t$$

Noise terms

$$z_t = H_t x_t + v_t$$

Correction

The Kalman Filter

- The simplest recursive Bayesian filter

- It is used *everywhere:* very important

- Requires that you can write the dynamics of your system using linear algebra (matrices etc)

- Boils down to 3 equations: Encodes the motion model

Current state

Covariance (=error)

Noise terms

Propagation

Correction

$$x_t = F_t x_{t-\delta t} + w_t$$

$$P_t = F_t P_{t-\delta t} F_t^T + Q_t$$

$$z_t = H_t x_t + v_t$$

The Kalman Filter

- The simplest recursive Bayesian filter

- It is used *everywhere:* very important

- Requires that you can write the dynamics of your system using linear algebra (matrices etc)

- Boils down to 3 equations:

Encodes the motion model

Current state

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-\delta t} + \boldsymbol{w}_t$$

Propagation

$$\boldsymbol{P}_t = \boldsymbol{F}_t \boldsymbol{P}_{t-\delta t} \boldsymbol{F}_t^T + \boldsymbol{Q}_t$$

Covariance (=error)

Noise terms

$$\boldsymbol{z}_t = \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{v}_t$$

Correction

Measurement model (how the measurement relates to the state)

- Measurement model (just measure position directly)

$$z_t = H_t x_t + v_t$$

$$H = (1 \quad 0)$$

The Nitty Gritty

Predict [edit]

Predicted (*a priori*) state estimate

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k$$

Predicted (*a priori*) estimate covariance

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}\mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k$$

Update [edit]

Innovation or measurement residual

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}$$

Innovation (or residual) covariance

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k$$

Optimal Kalman gain

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1}$$

Updated (*a posteriori*) state estimate

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$$

Updated (*a posteriori*) estimate covariance

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$$

(Thanks to wikipedia. No, you aren't expected to learn these)

Initially we have some position estimate that is associated with a normal distribution
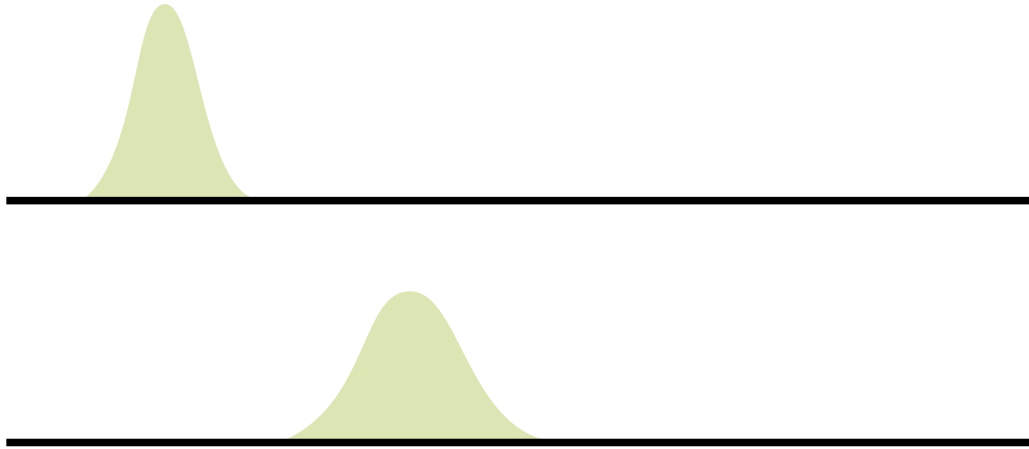
We propagate the state, meaning we use the motion model to move it forward. Since we had no actual input, we increase the error (→ Gaussian gets shorter and fatter)

Key to the Kalman Filter
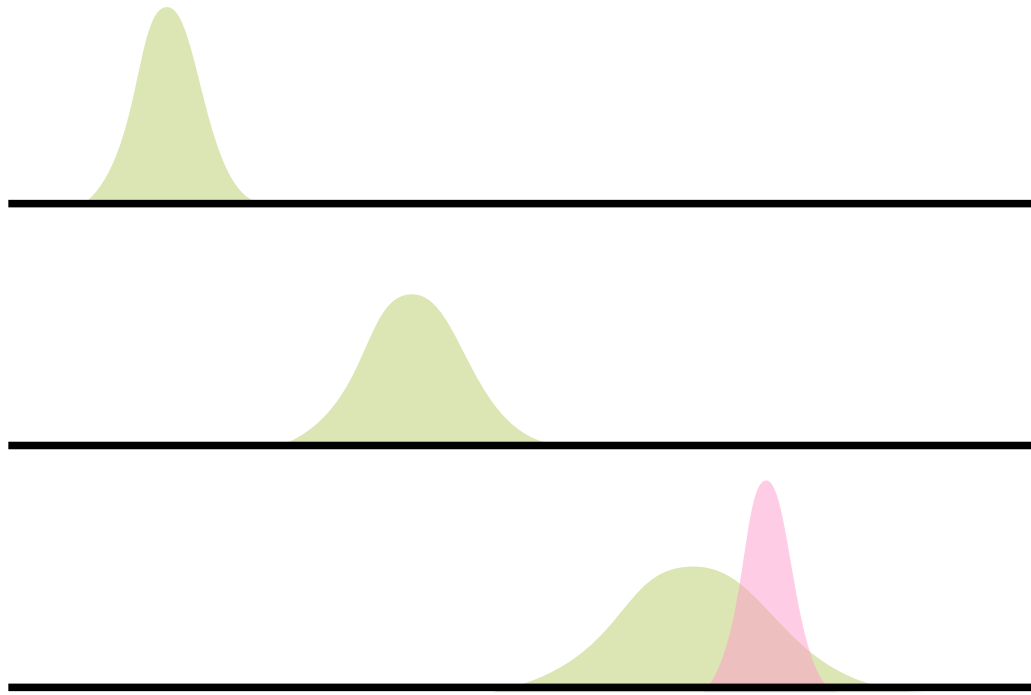
We repeat the propagation but then a measurement comes in. This is associated with another Gaussian, although thinner because it's an OK estimate

Key to the Kalman Filter
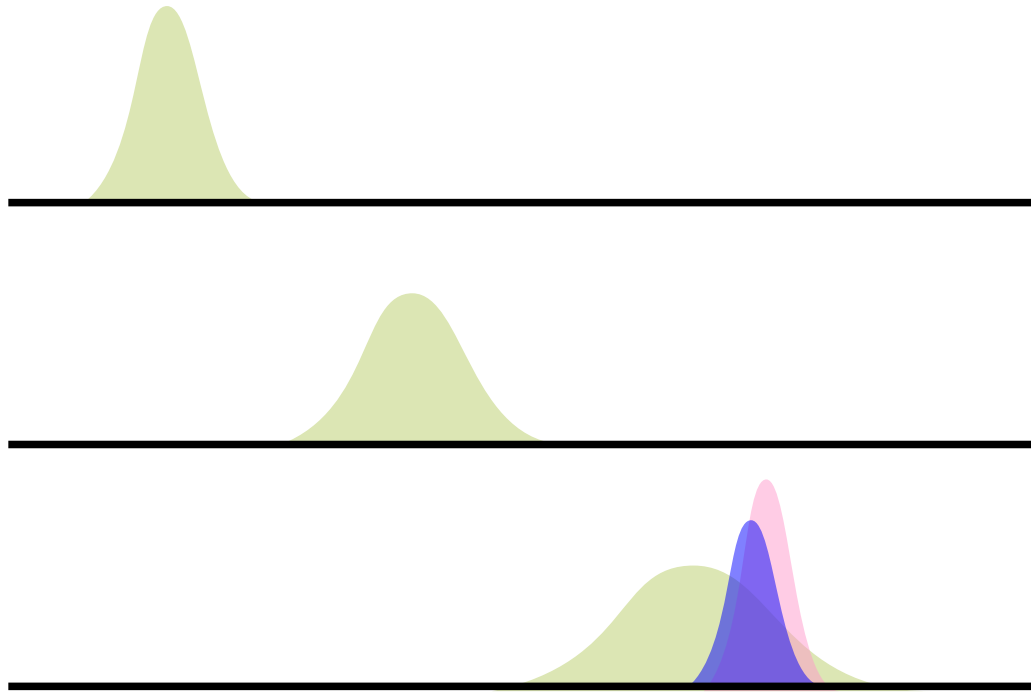
The beauty of a Gaussian is that when you multiply two together you get another Gaussian. Thus we always finish a cycle with a new Gaussian estimate → we can represent it using just two parameters, making it amenable to linear algebra

[An example]

A more complex example

- Consider the Inertial GPS systems you find in vehicles

- They need to estimate where the car is at all times *between* GPS measurements

- We compute position by concatenating a series of displacements and headings (dead reckoning)

- We use inertial sensors to estimate the displacements (wheel encoders) and headings (gyroscopes) since the last state estimate

t=2
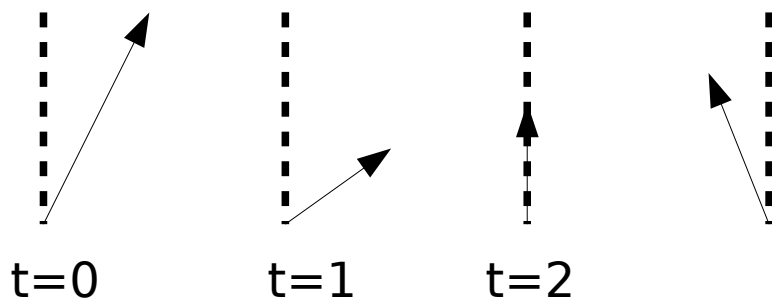
t=0   t=1   t=2
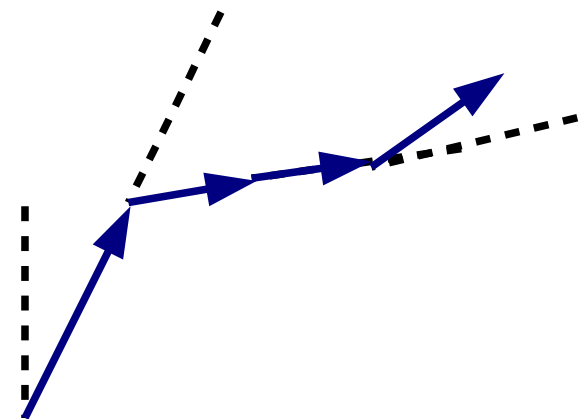
Inertial Nav

- We integrate the gyroscope signal to estimate the heading change (note the motion model uses the inertial inputs)

- But gyros are subject to bias errors (a bias is a bogus offset reported when it's not rotating) and we often see erroneous bending:

True
(unobservable)

Estimate
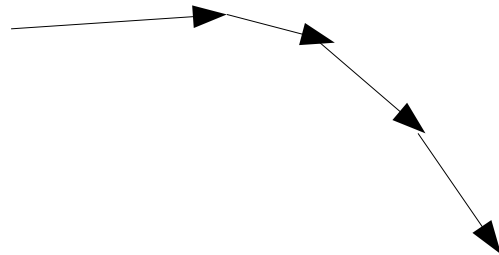
INS bias bends heading

- When a GPS measurement comes in we can fix things

True
(unobservable)

GPS

INS

GPS correction
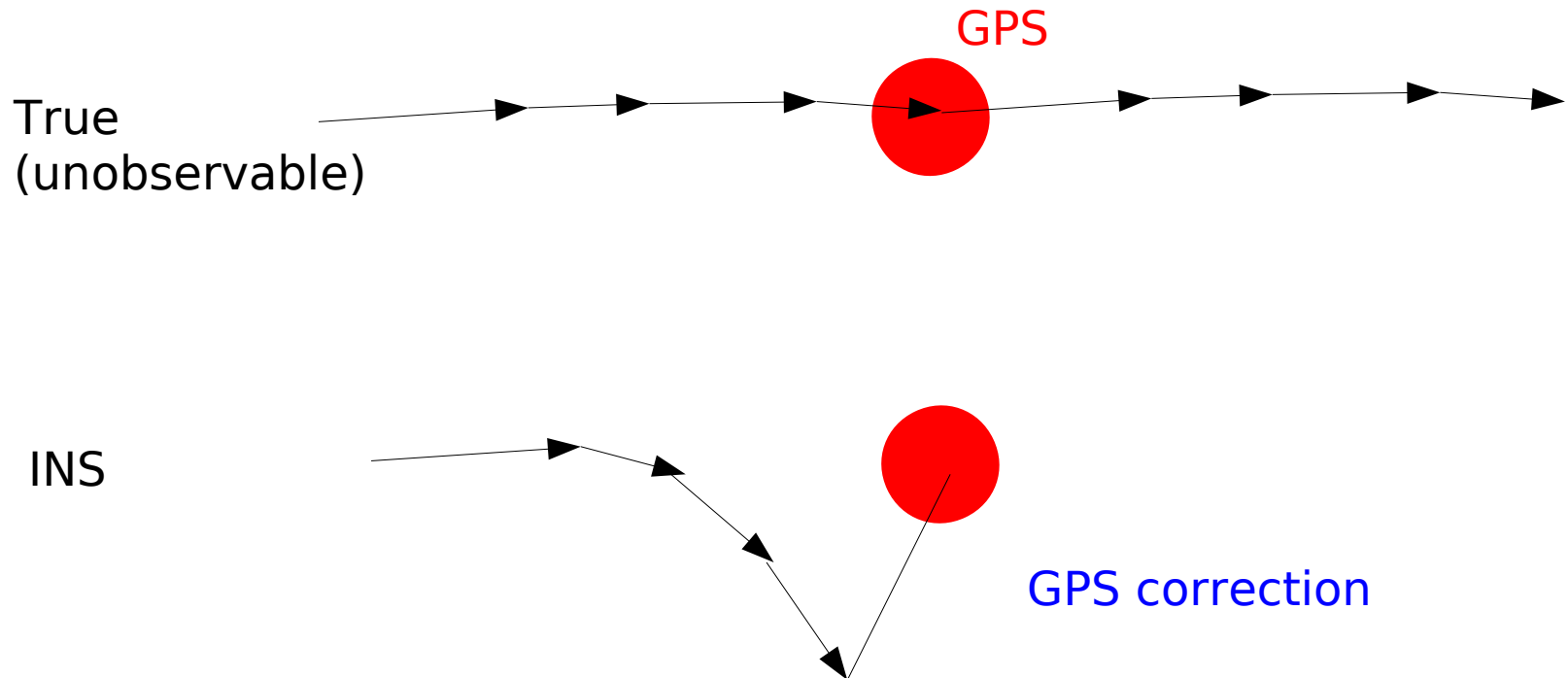
- But if we just correct position, it goes wrong again

GPS

True
(unobservable)

INS
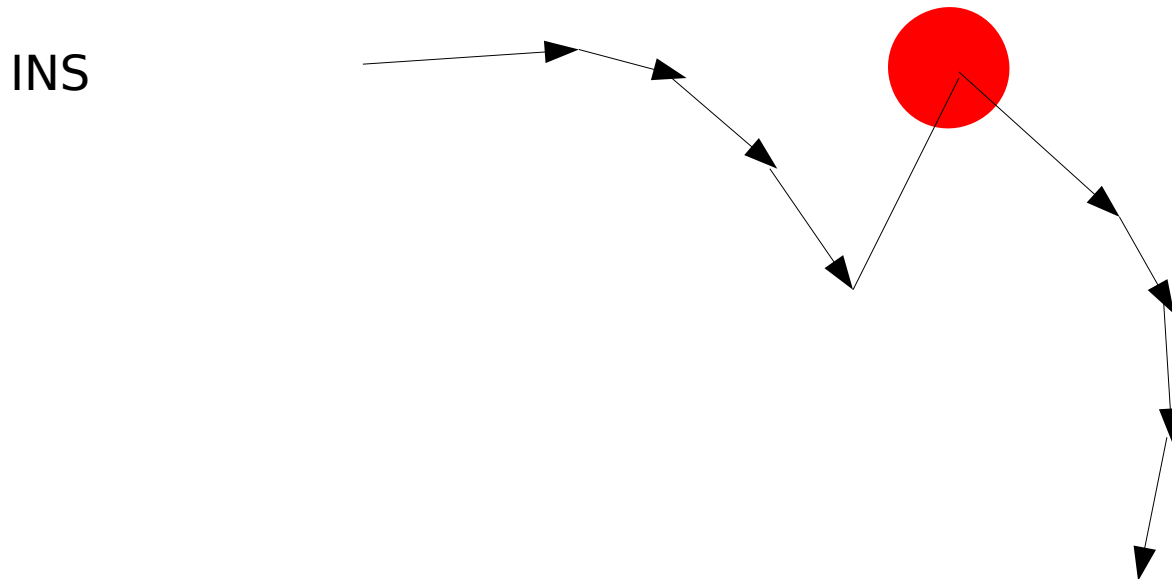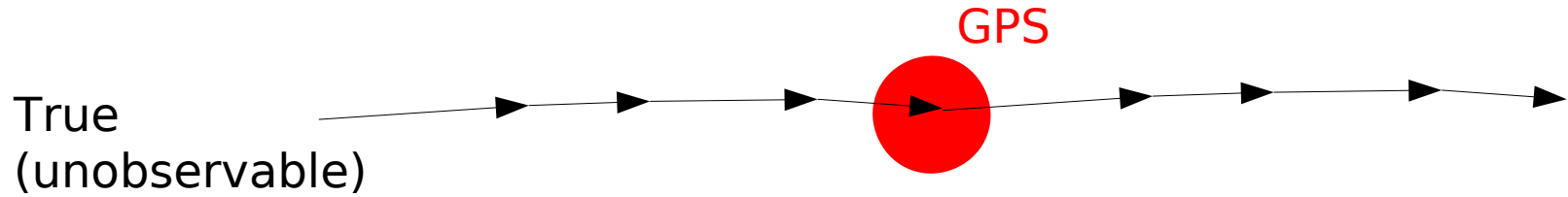
- But if we add the bias to the state in the kalman filter, it will estimate that for us too

True
(unobservable)

INS

Bias corrected
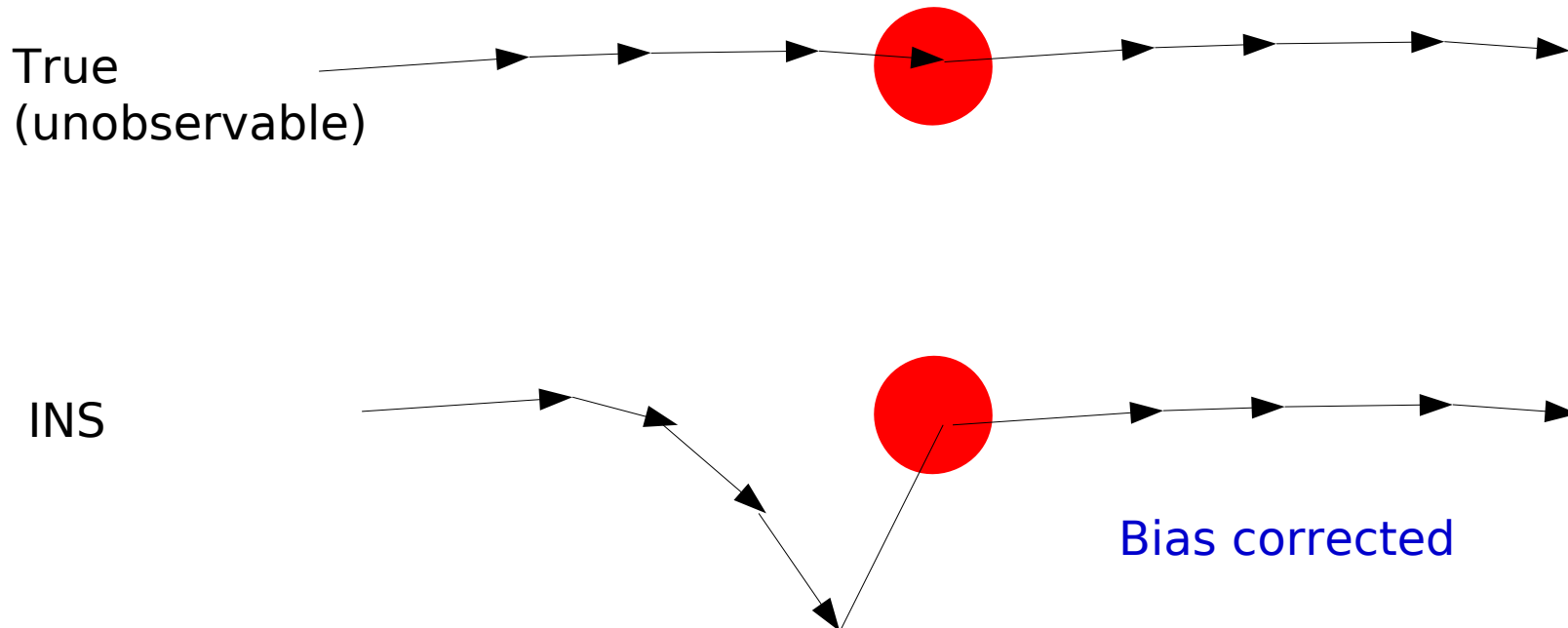
KF Limitations

$$\mathrm{Bel}^-(\boldsymbol{x}_t) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-\delta t})\mathrm{Bel}(\boldsymbol{x}_{t-\delta t})\,d\boldsymbol{x}_{t-\delta t}\,.$$ Propagation

$$\mathrm{Bel}(\boldsymbol{x}_t) = \alpha_t p(\boldsymbol{z}_t|\boldsymbol{x}_t)\mathrm{Bel}^-(\boldsymbol{x}_t)$$ Correction

- What if those probability distributions don't lend themselves to being normal?

- Our example will be constraining movement to be on a building floorplan. How could you build a motion model matrix that incorporated a floorplan??!

The Particle Filter

Our Example

- Imagine tracking someone around a building using the sensors on their phone and a floorplan

- We now estimate *step events* where a step has a length associated with it and a direction.
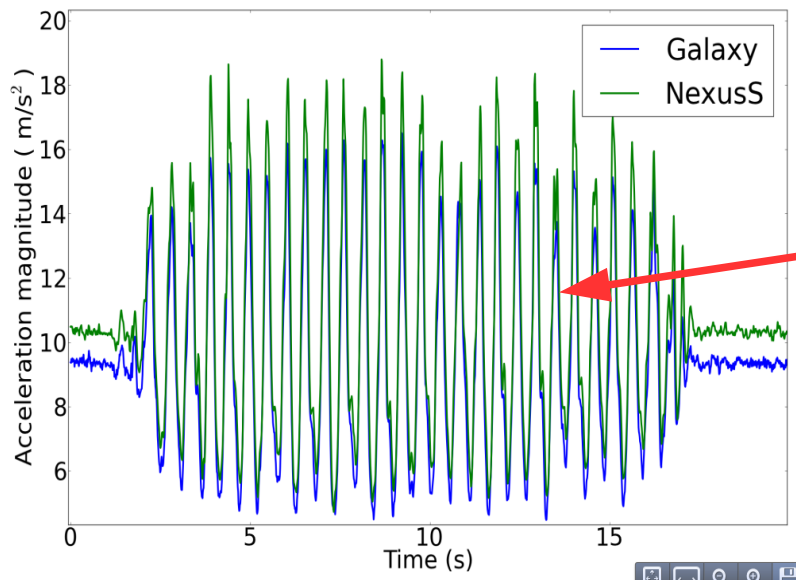
Easy to spot steps when looking at the accelerometer

Integrating the gyro gives direction change as before

Particle Filters for Location

- Encode state in **particles**. Each particle is just an individual hypothesis about the position and orientation of the user

- Each particle has an assigned probability

- Particles are updated by:
  - Propagation/Predict
  - Correct
  - (Resample)
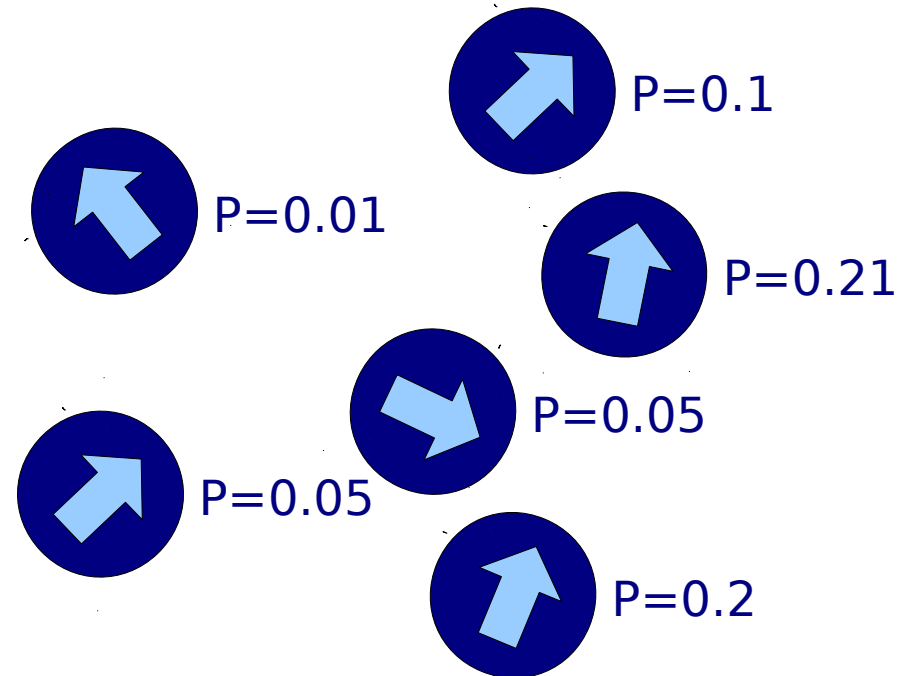
P=0.1
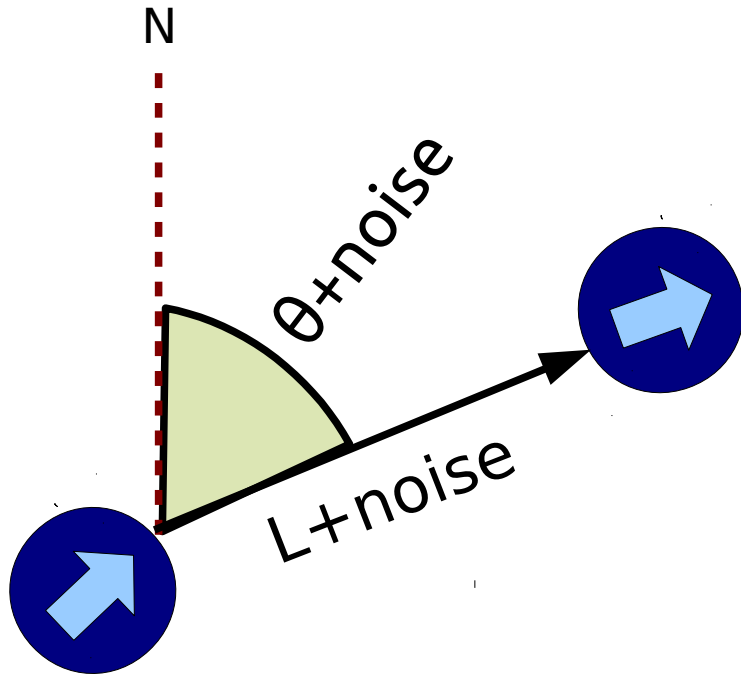
P=0.01

P=0.21

P=0.05

P=0.05

P=0.2

- Each particle is moved by the measured step length and direction

- Plus some additive noise that represents the imperfect measurement

N
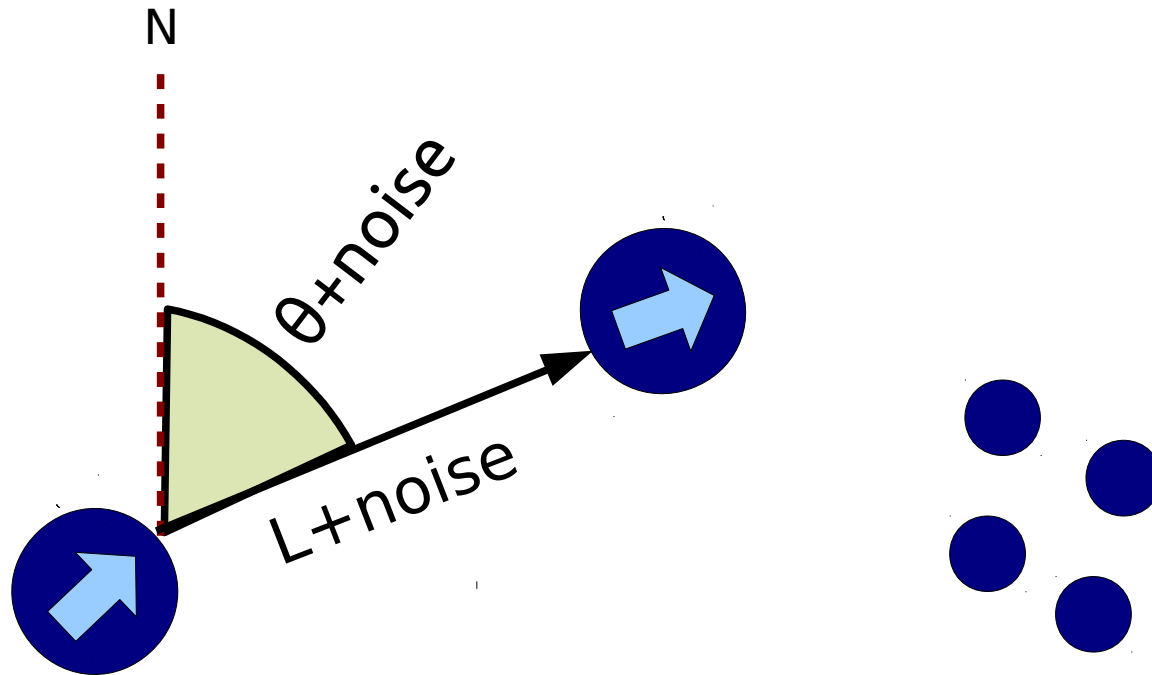
θ+noise

L+noise

- Each particle is moved by the measured step length and direction

- Plus some additive noise that represents the imperfect measurement
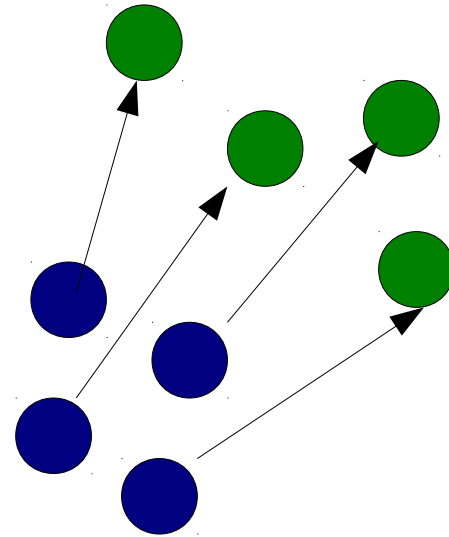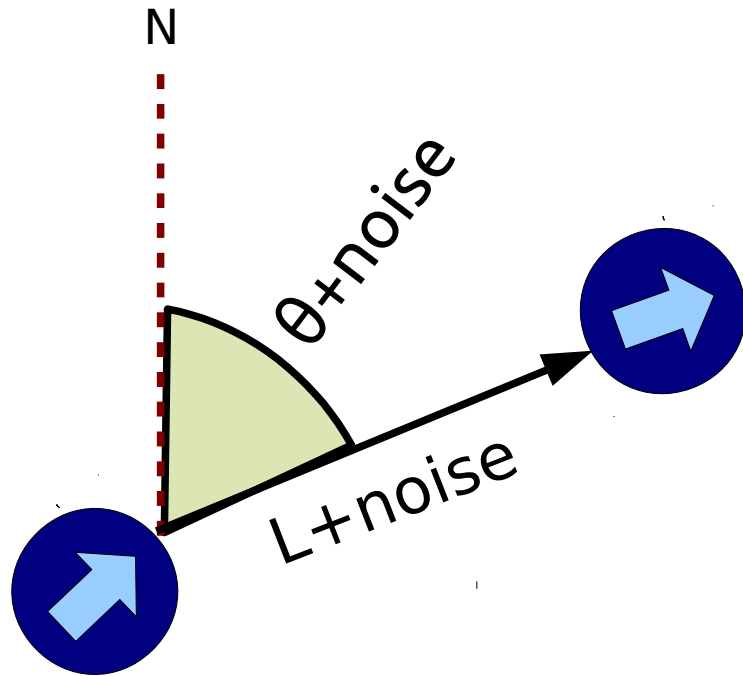
- Each particle is moved by the measured step length and direction

- Plus some additive noise that represents the imperfect measurement

Cloud Spread

- With each step, the particle cloud spreads naturally due to the noise we add

- This is good: it represents that our drift ($\rightarrow$ uncertainty) is growing (c.f. the Gaussian growing fatter with each step of the KF)

- Given a measurement we can reassign the particle probabilities

- If we had an absolute position (maybe a GPS fix) we could weight to that position

0.2

0.2

0.2

0.2

0.2
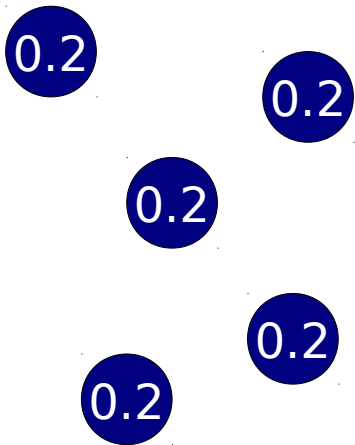
- Given a measurement we can reassign the particle probabilities

- If we had an absolute position (maybe a GPS fix) we could weight to that position
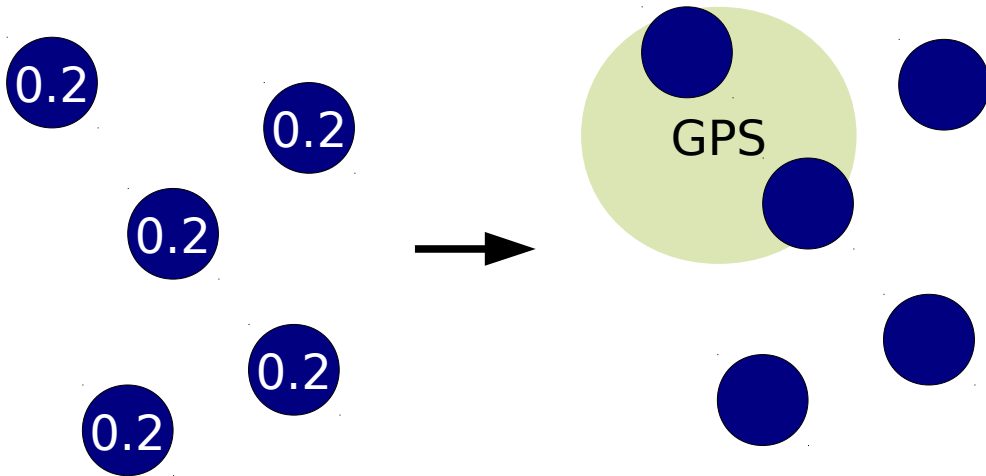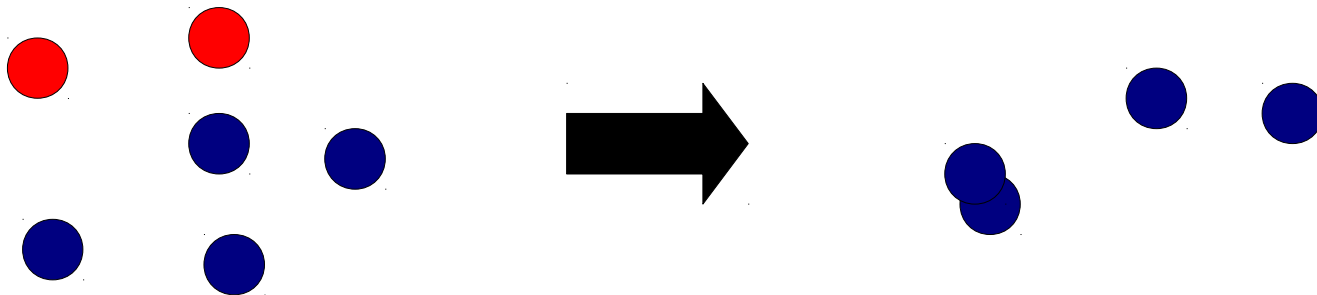
- Given a measurement we can reassign the particle probabilities

- If we had an absolute position (maybe a GPS fix) we could weight to that position
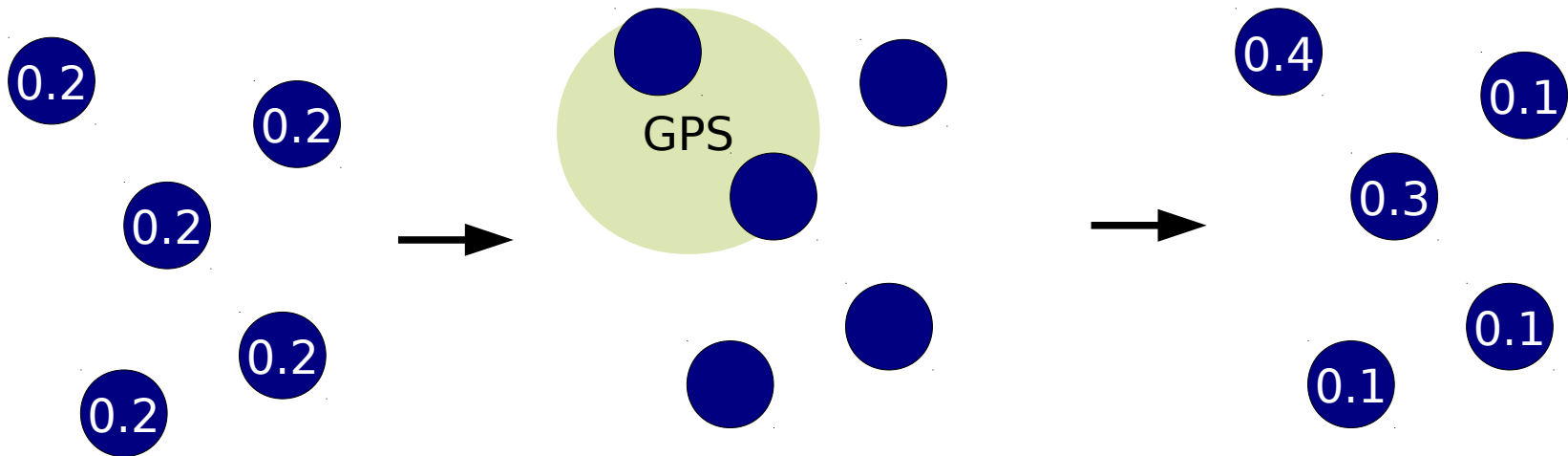
- Constraints can be included as pseudo-measurements

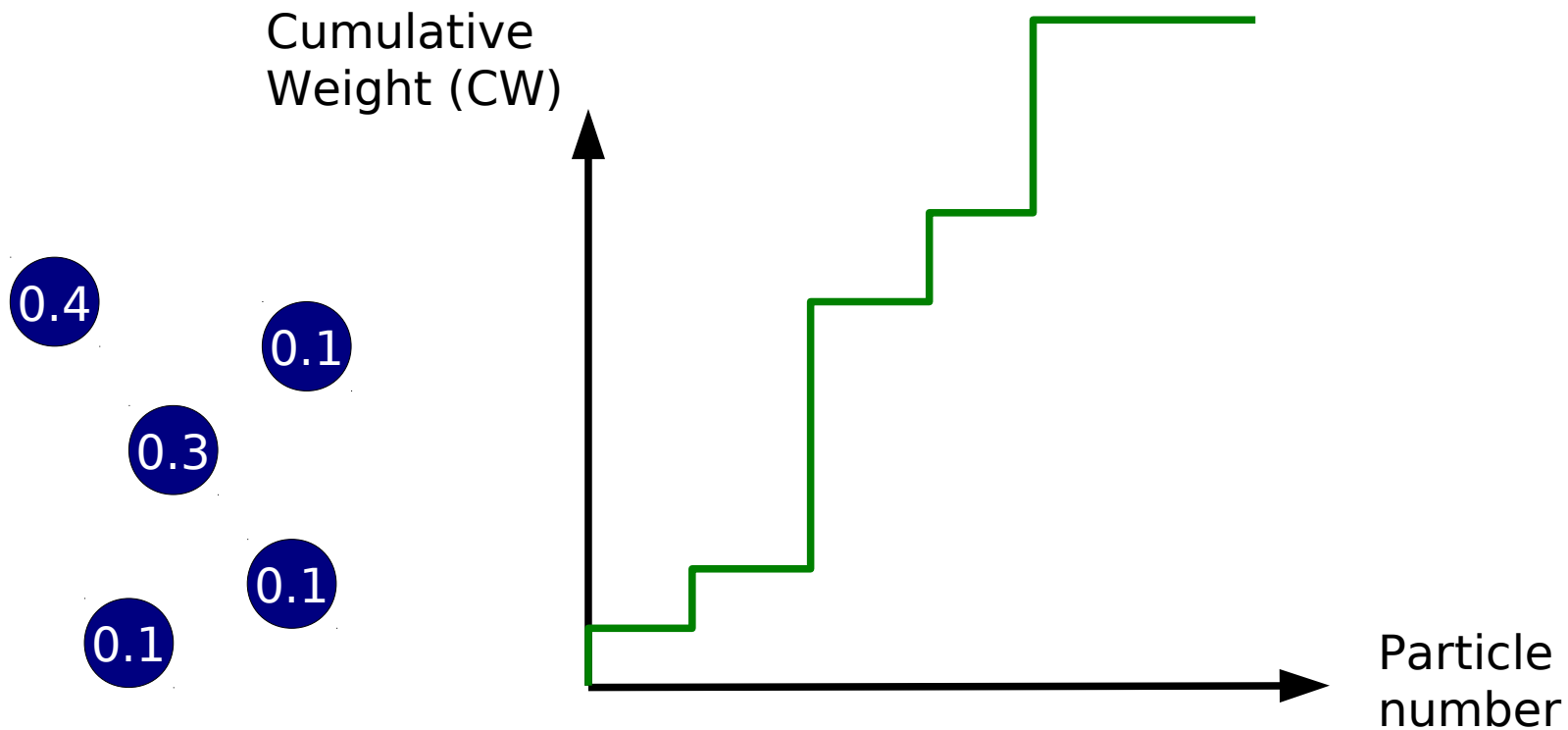- For walls we can simply set p=0 if the particle crossed a wall and leave it alone otherwise

$P_{n+1}=0$

$P_{n+1}=1$

$P_n=1$

$P_n=1$
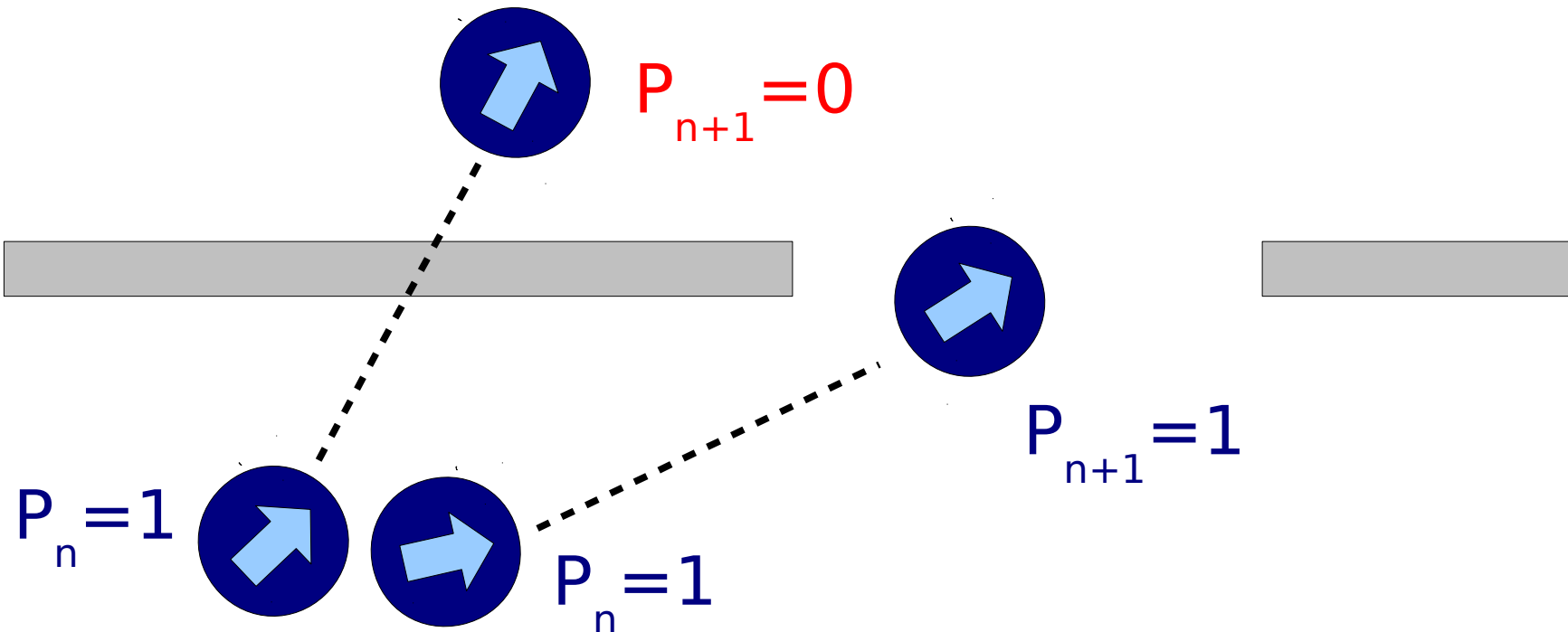
- Want to get rid of p=0 particles but still need some particles!

- We **resample**: generate a new particle set by sampling the old one **in proportion to the particle weights**

- P=0 → won't go any further

- P=1 → may be reproduced (multiple times) if chosen at random

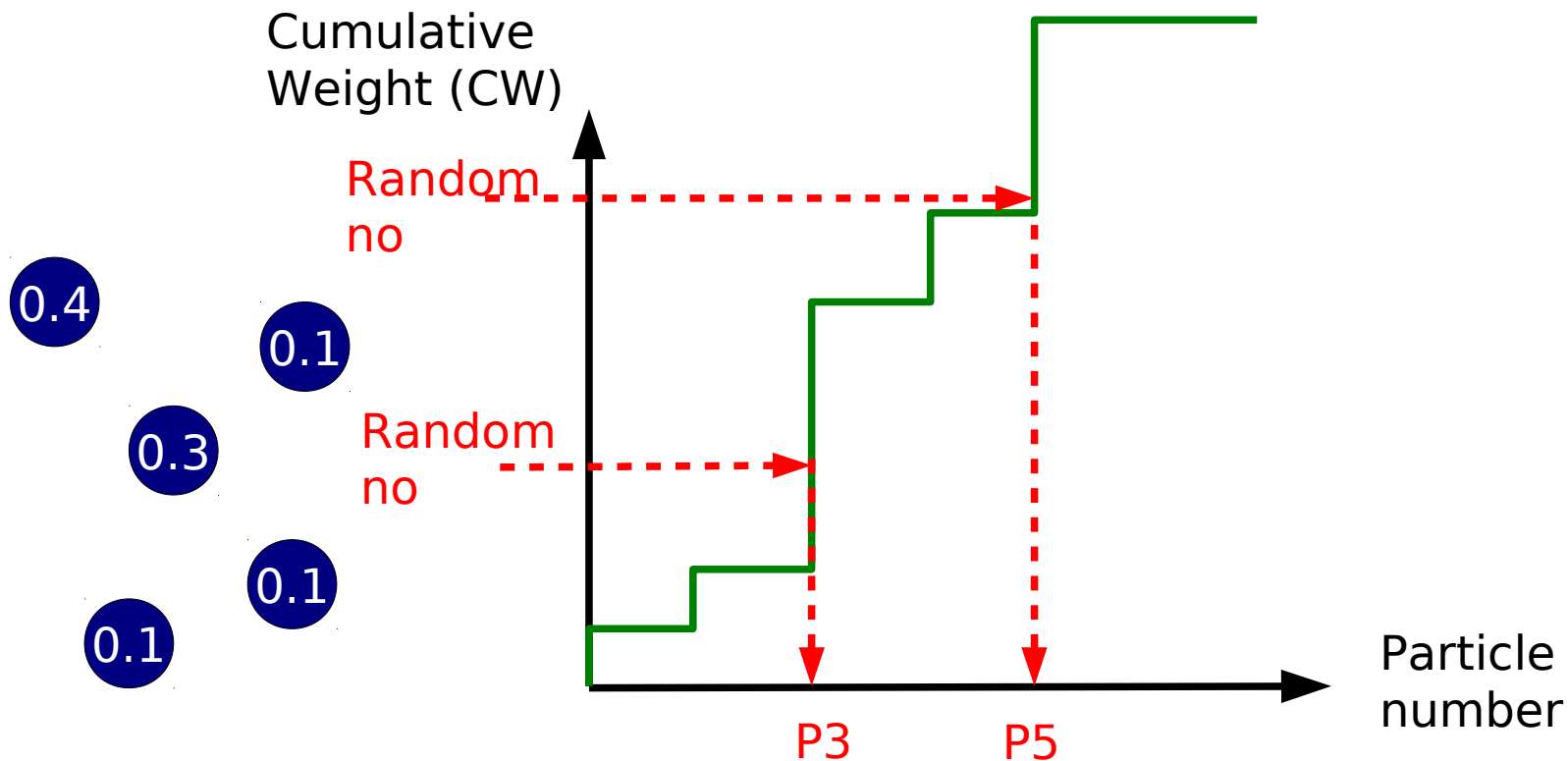- To select in proportion form a cumulative weight...

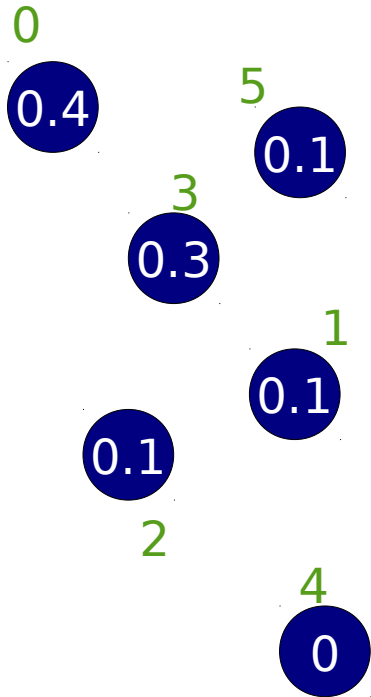- To select in proportion form a cumulative weight…

Cumulative Weight (CW)

Random no

Random no

0.4

0.1

0.3

0.1

0.1

Particle number

P3   P5

- To select in proportion form a cumulative weight…

0	1	2	3	4	5
0.4	0.5	0.6	0.9	0.9	1.0

0

0.4

5

0.1

3

0.3

1

0.1

0.1

2

4

0

Random → 0.66 → Particle 3
Random → 0.43 → Particle 1
Random → 0.01 → Particle 0
Random → 0.88 → Particle 3
Random → 0.23 → Particle 0

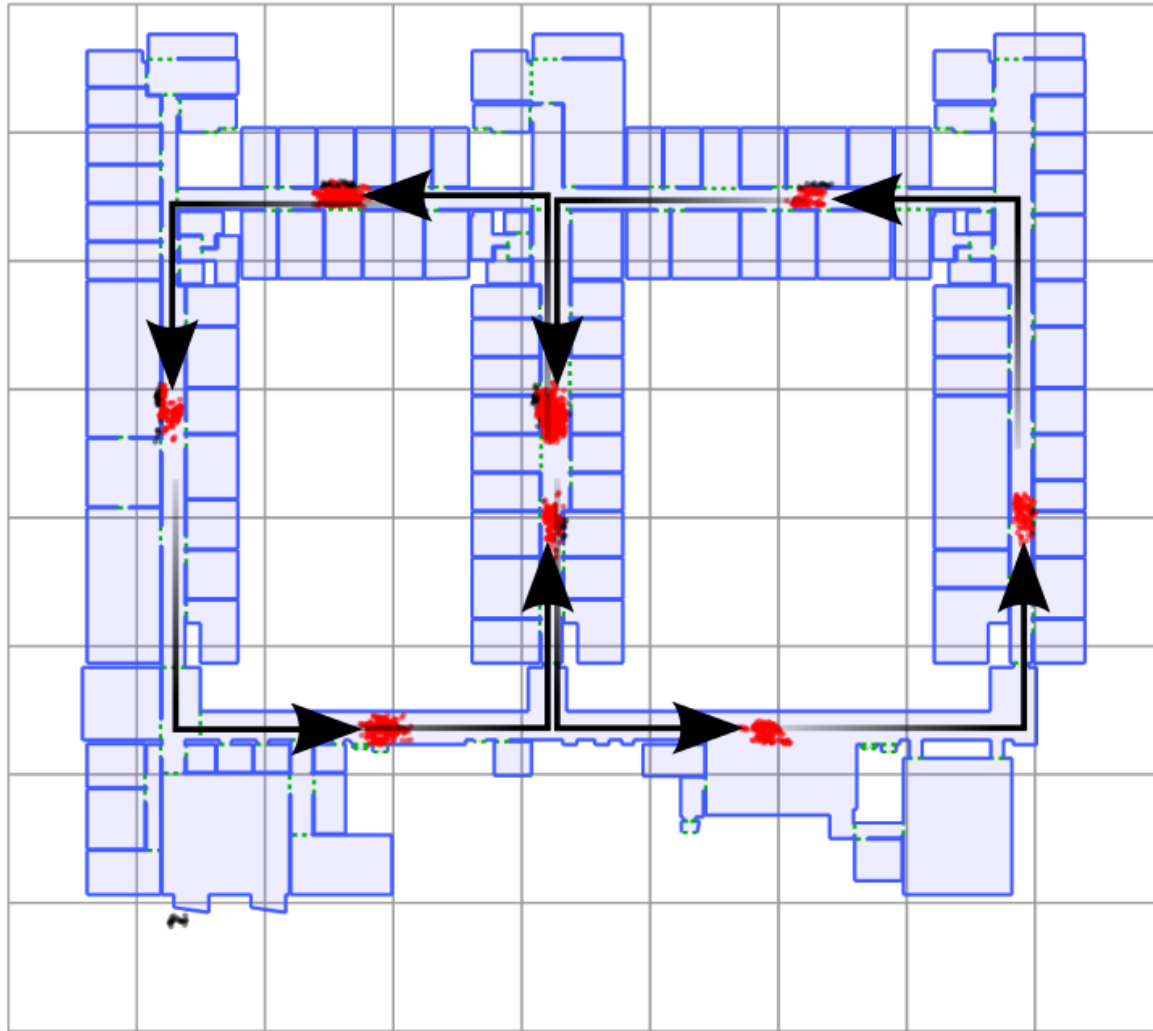(More probable particles more likely to be resampled)

A Note on Performance

- Update and correct steps are nicely parallelisable

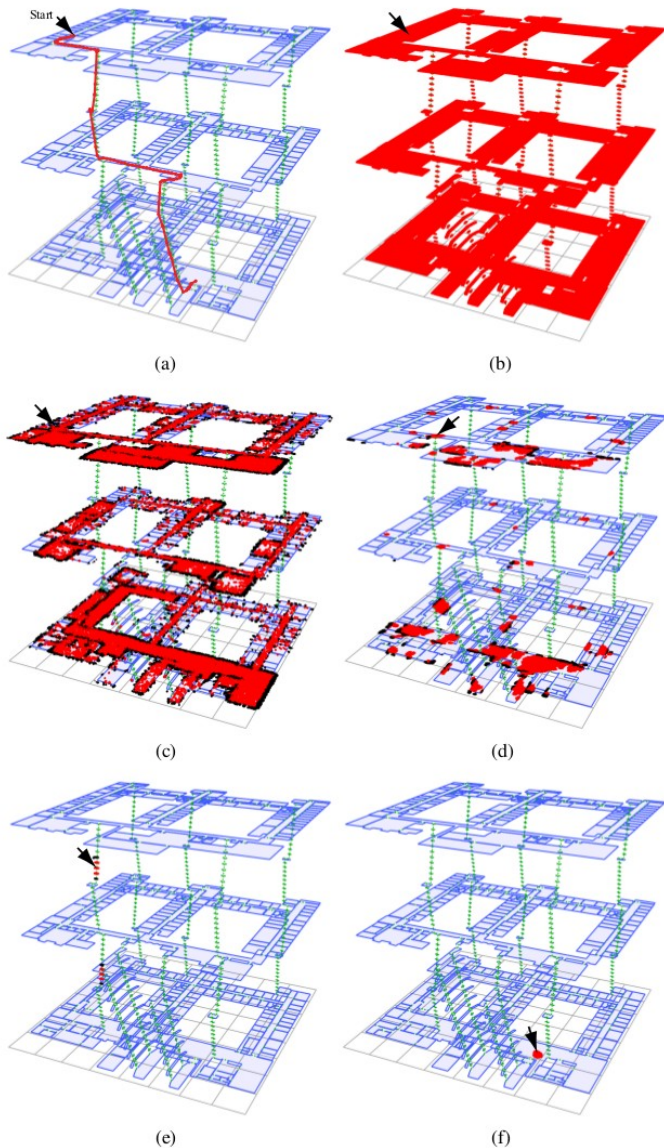- But forming the cumulative weight for resampling is fundamentally sequential...

(a)   (b)   (c)   (d)   (e)   (f)
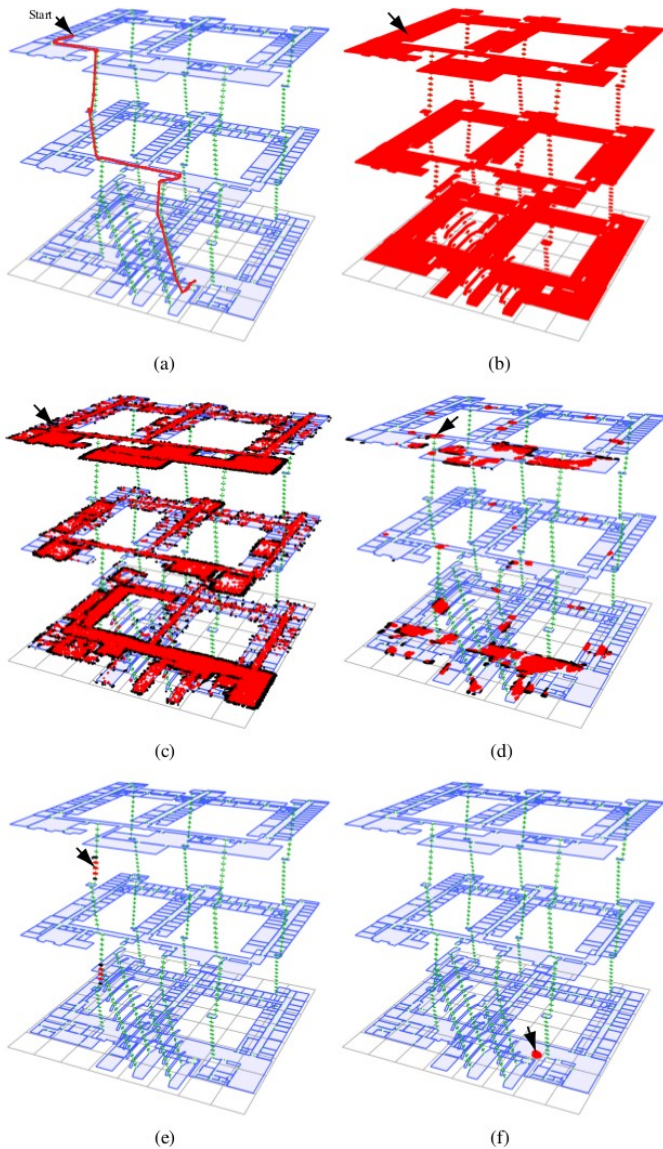
- Initially we have no knowledge of the user's position
  - Lots of particles
  - Localisation Phase

Works well...

(a) (b) (c) (d) (e) (f)

- Eventually we figure out where they are and the problem becomes easier

  - Fewer particles needed

  - Tracking Phase

- We got ~ 0.75m accuracy 95% of the time with a sensor on the shoe

In General

- Particle filters are easy to implement and highly flexible

- But:

  - Every particle you add costs you in terms of computation

  - The results are not deterministic

  - Too few particles gives bad/failed results, while too many wastes precious CPU cycles