A two step hardware design method using CλaSH

Rinse Wester, Christiaan Baaij, Jan Kuper

University of Twente, Enschede

August 27, 2012

Contents

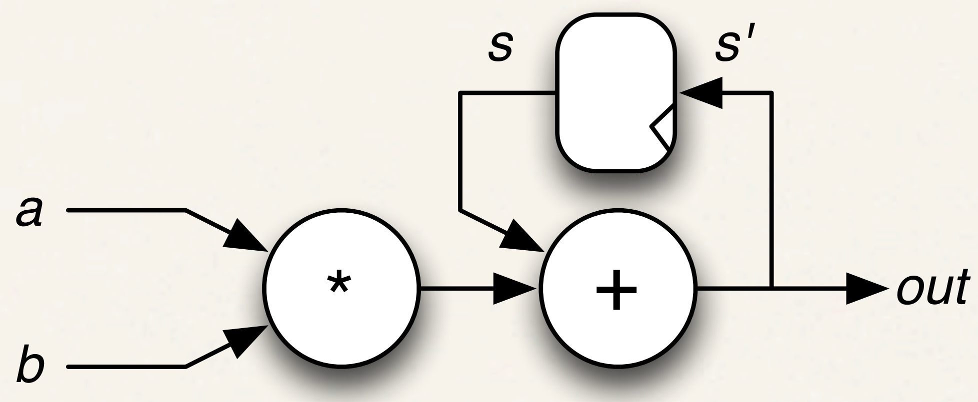
Introduction

- What is CλaSH?
  - Functional Language and Compiler for Concurrent Digital Hardware Design
- Motivation?
  - Evaluate CλaSH and design method on complex application
- Why a particle filter?
  - Covers important aspects of digital hardware design: massive parallelism, feedback loop and data dependent processing.

Background

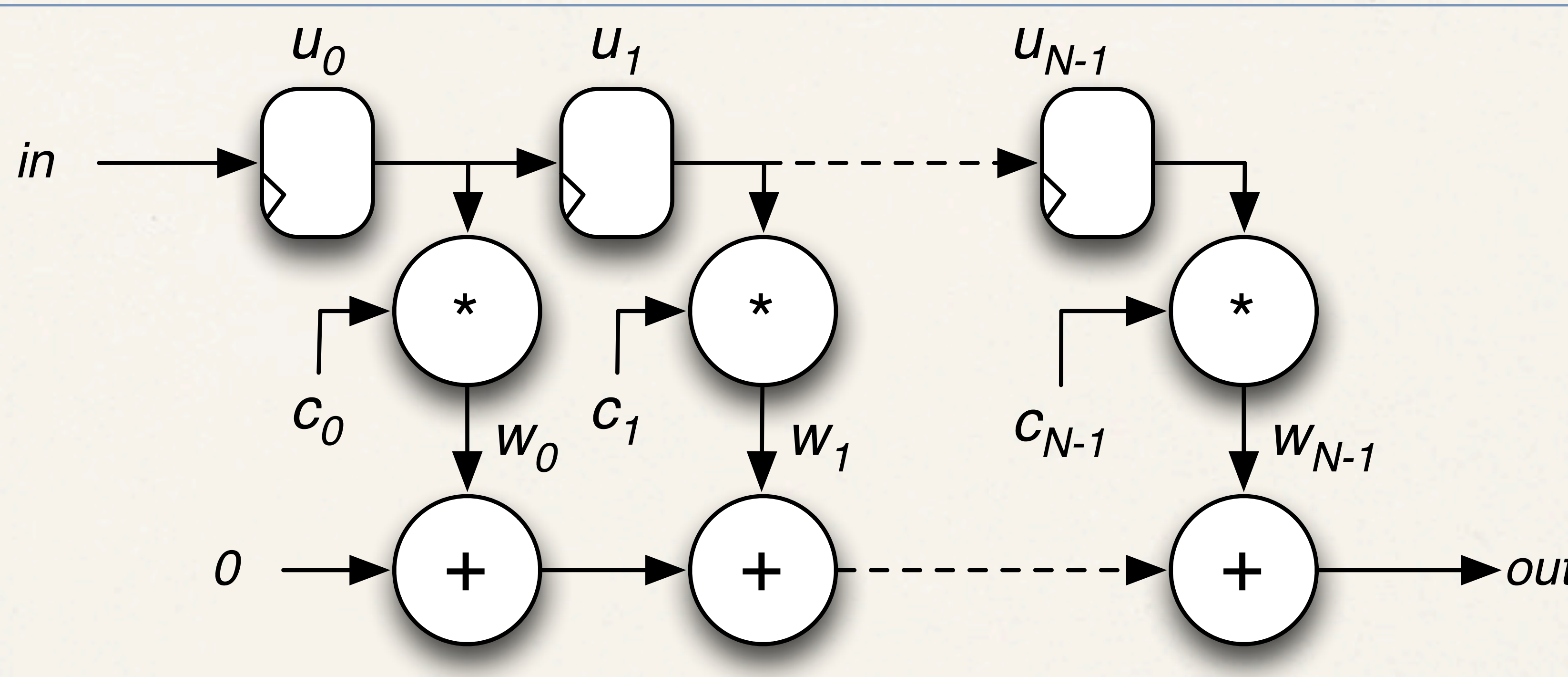
- CλaSH
  - A functional language and compiler for digital hardware design
  - On the lowest level, everything is a Mealy machine $f(s,i) = (s',o)$
  - A CλaSH description is purely structural i.e. all operations are performed in a single clock cycle
  - Simulation is cycle accurate

Background



$$mac\ (State\ s)\ (a, b) = (State\ s', out)$$
where
$$s' = s + a * b$$
$$out = s'$$

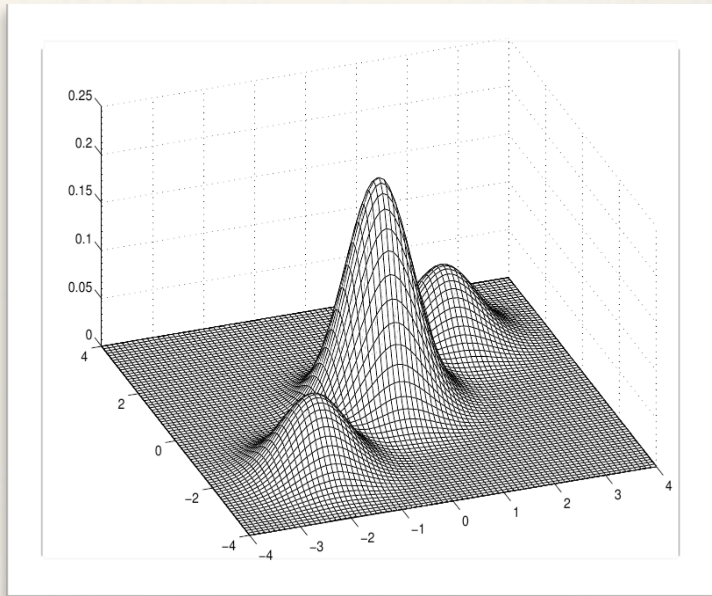
Background



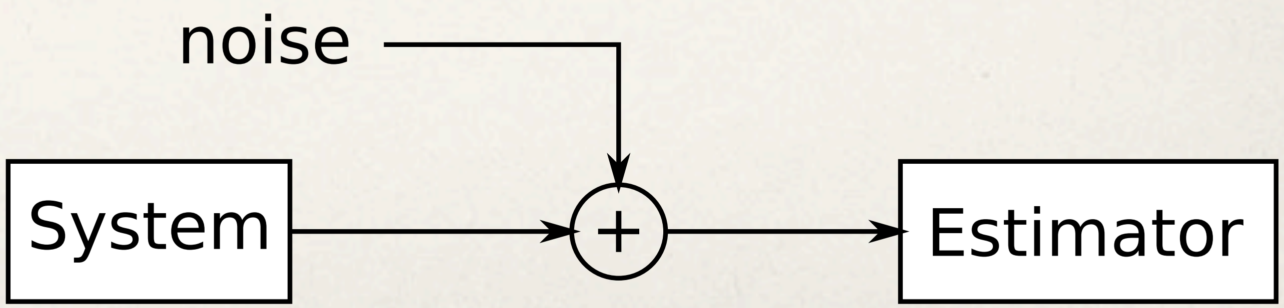
$$
\begin{aligned}
&\mathit{fir}\ \mathit{cs}\ (\mathit{State}\ \mathit{us})\ \mathit{inp} = (\mathit{State}\ \mathit{us}', \mathit{out})\\
&\quad \textbf{where}\\
&\qquad \mathit{us}' \ = \mathit{inp} +\!\!\gg \mathit{us}\\
&\qquad \mathit{ws} \ \ = \mathit{vzipWith}\ (*)\ \mathit{us}\ \mathit{cs}\\
&\qquad \mathit{out} = \mathit{vfoldl}\ (+)\ 0\ \mathit{ws}
\end{aligned}
$$

Background

- State estimation
  - Determine $p(x_k \mid z_k)$ recursively with noise
  - State variables: position, speed, angle, ...
  - Applications: tracking in radar and video

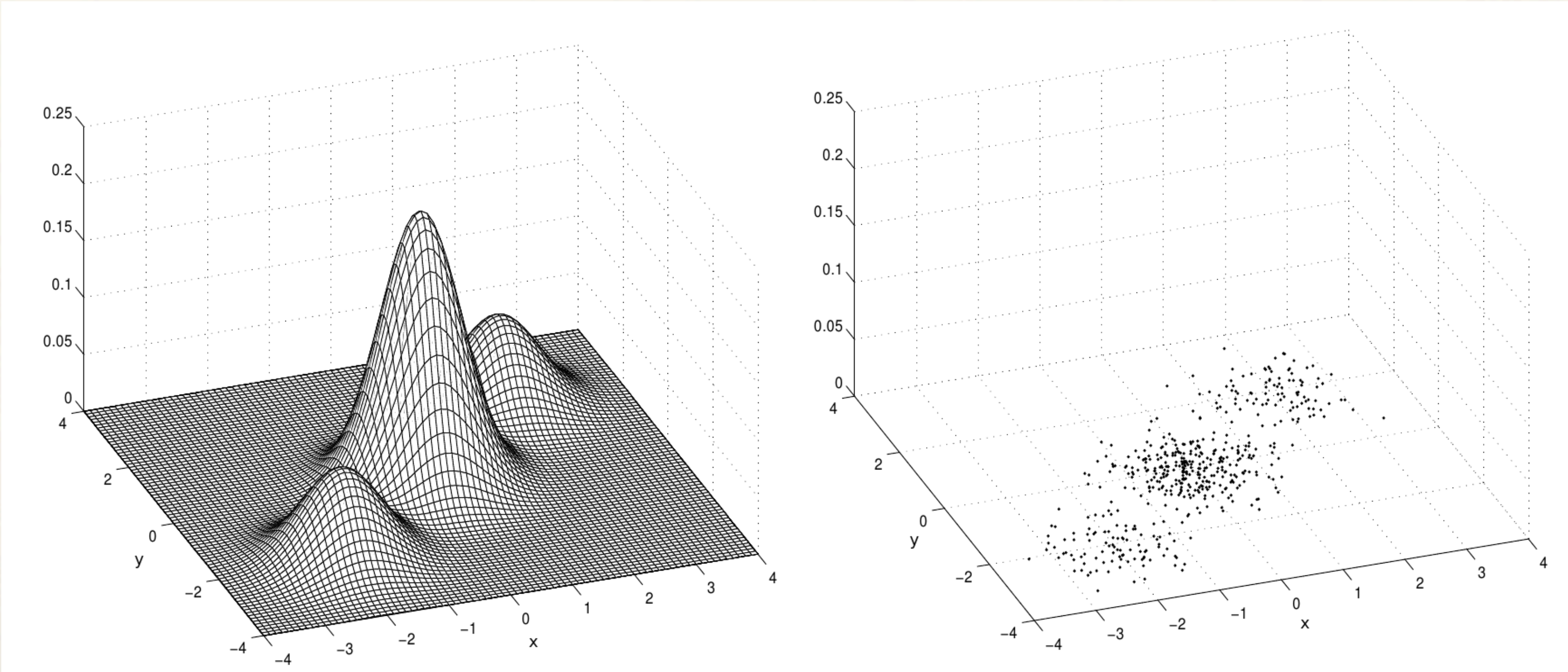


- Requirements for estimator
  - System dynamics
  - Measurement function



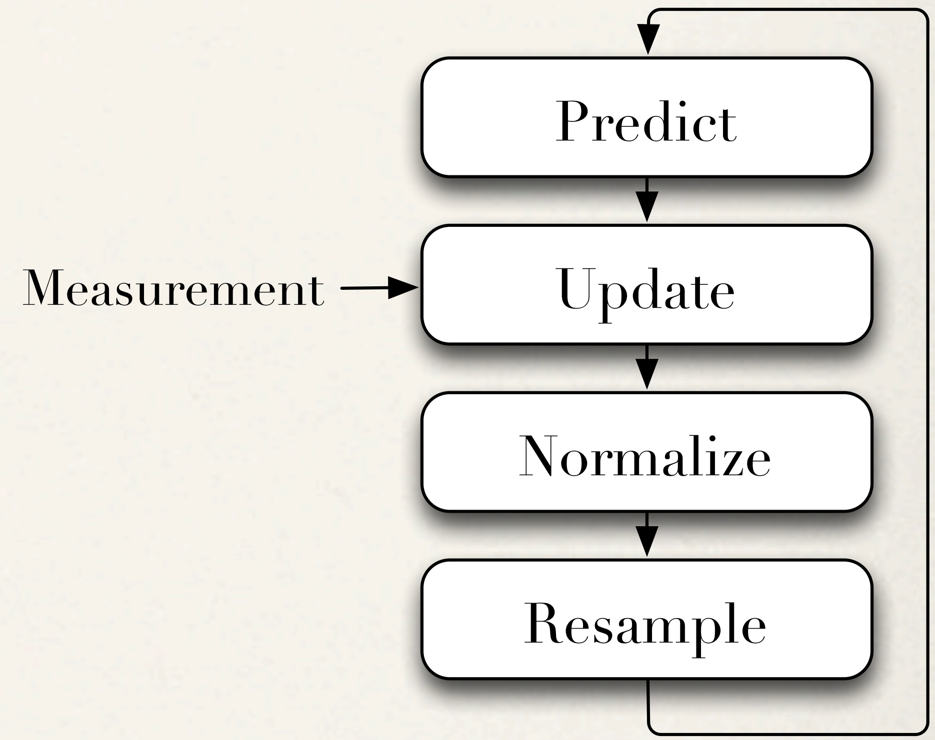
Background

- Monte Carlo approximation of $p(x_k \mid z_k)$ represented by concentration of points (particles)
- Applicable to non-linear, non gaussian systems (tracking, robotics,..)
- Parameterizable in and N, $F_{sys}(x)$ and $F_{meas}(x,m)$



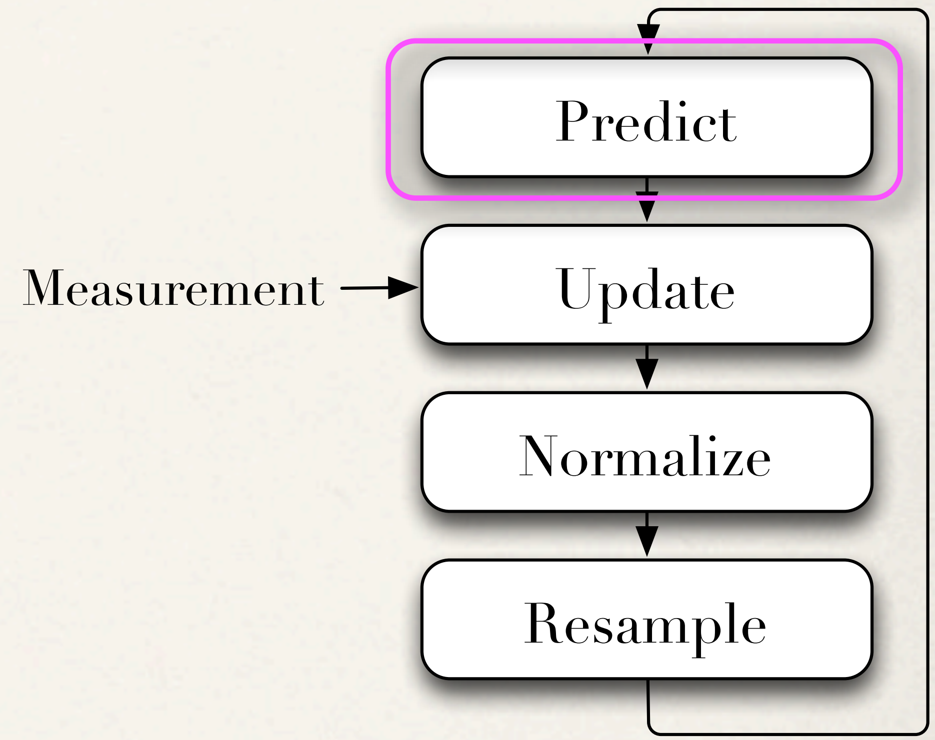
Background

- Prediction
  - Predict next state based on current $F_{sys}(x) \rightarrow x'$
- Update
  - Assign weights to particles based on measurement $F_{meas}(x,m) \rightarrow \omega$
- Normalize such that $\sum \omega^{(i)}=1$
- Resample



Background

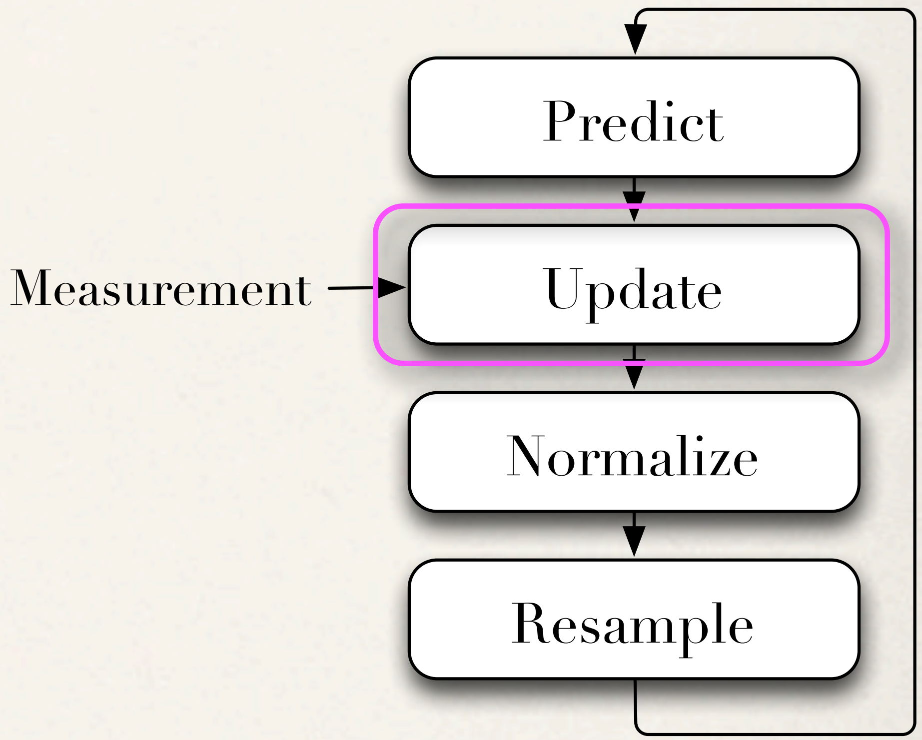
- Prediction
  - Predict next state based on current $F_{sys}(x) \rightarrow x'$
- Update
  - Assign weights to particles based on measurement $F_{meas}(x,m) \rightarrow \omega$
- Normalize such that $\sum \omega^{(i)}=1$
- Resample



Background

Particle Filter

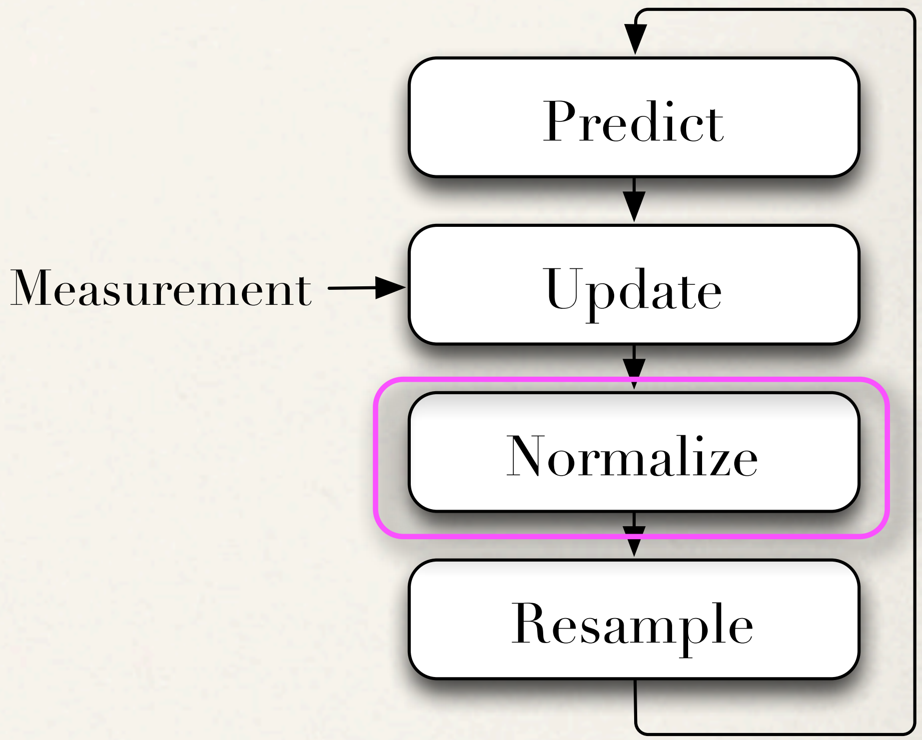
- Prediction
  - Predict next state based on current $F_{sys}(x) \rightarrow x'$
- Update
  - Assign weights to particles based on measurement $F_{meas}(x,m) \rightarrow \omega$
- Normalize such that $\sum \omega^{(i)}=1$
- Resample



Background

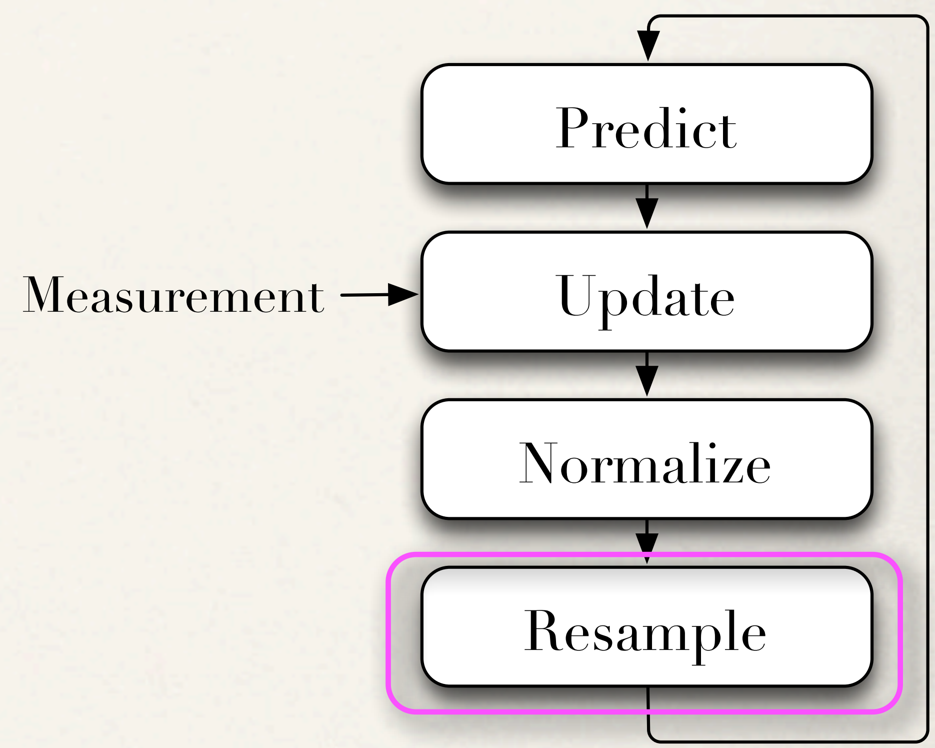
Particle Filter

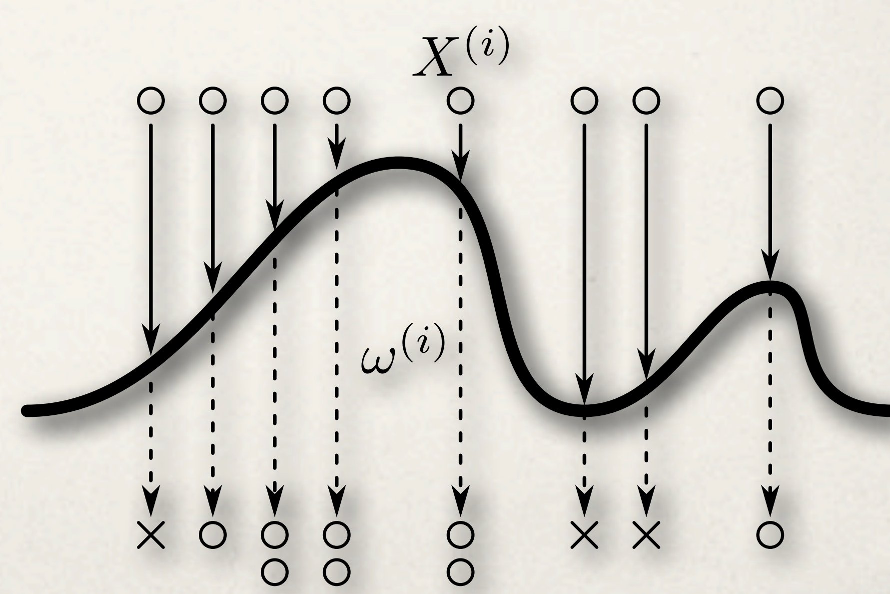
- Prediction
  - Predict next state based on current $F_{sys}(x) \rightarrow x'$
- Update
  - Assign weights to particles based on measurement $F_{meas}(x,m) \rightarrow \omega$
- Normalize such that $\sum \omega^{(i)}=1$
- Resample



Background

- Prediction
  - Predict next state based on current $F_{sys}(x) \rightarrow x'$
- Update
  - Assign weights to particles based on measurement $F_{meas}(x,m) \rightarrow \omega$
- Normalize such that $\sum \omega^{(i)}=1$
- Resample





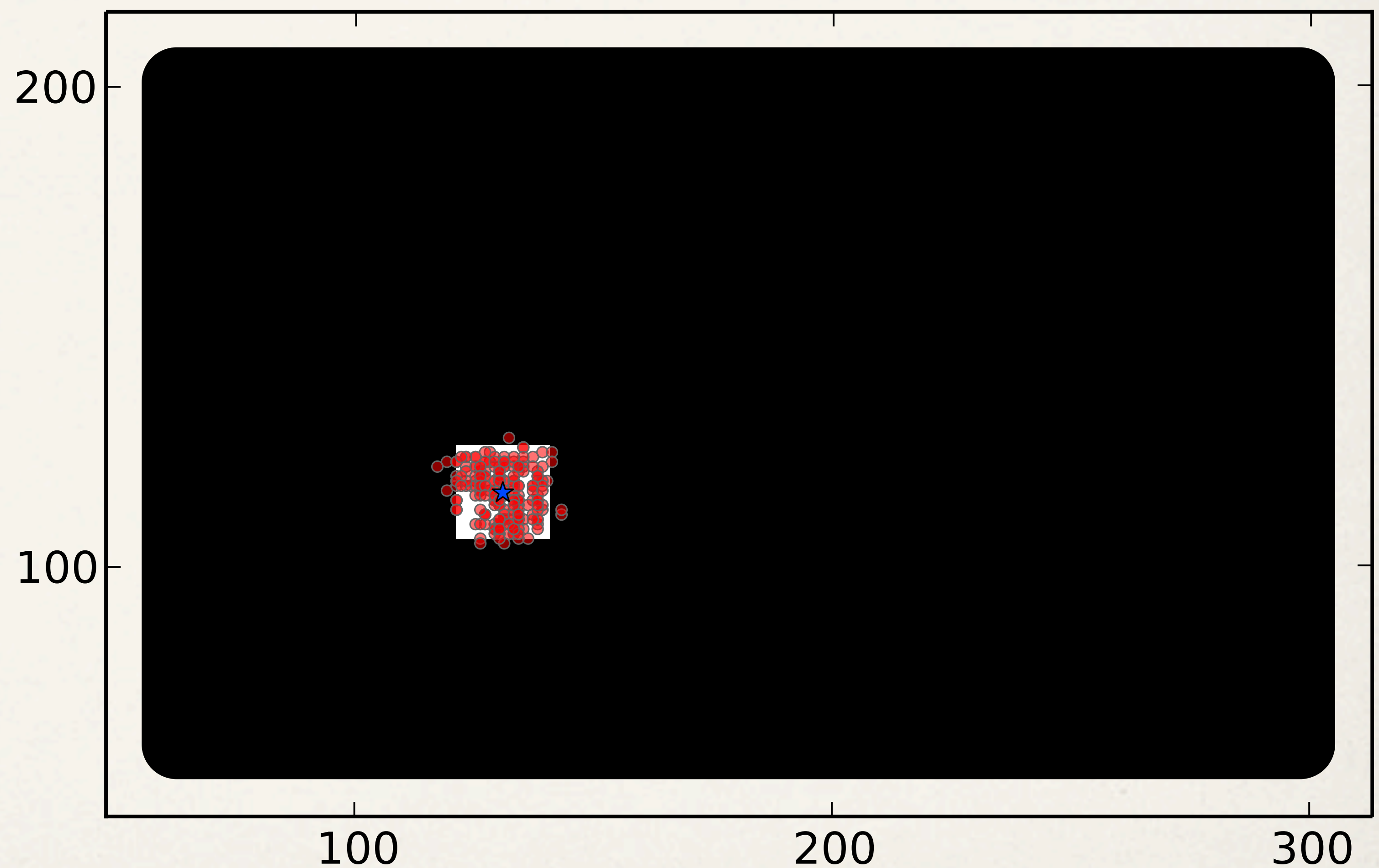
Background

Mathematical formulation of Particle Filter

- Prediction $x_k^{(i)} \sim p(x_k | x_{k-1})$
  - System dynamics function $x_k^{(i)} = f(x_{k-1}^{(i)}, u_k)$
- Update $\omega_k^{(i)} = p(z_k | x_k^{(i)})$
  - Measurement function $\omega_k^{(i)} = g(x_k^{(i)}, z_k, v_k), \quad \text{for} \quad i = 1 \ldots N$
- Normalize $\tilde{\omega}^{(i)} = \frac{\omega^{(i)}}{\sum_{n=1}^{N} \omega^{(n)}} \quad \text{for} \quad i = 1 \ldots N$
- Resample $\{\tilde{x}_k^{(1)}, \tilde{x}_k^{(2)} \ldots \tilde{x}_k^{(N)}\} = \Big\|_{n=1}^{N} \mathit{replicate}(x_k^{(i)}, r_i)$

Background

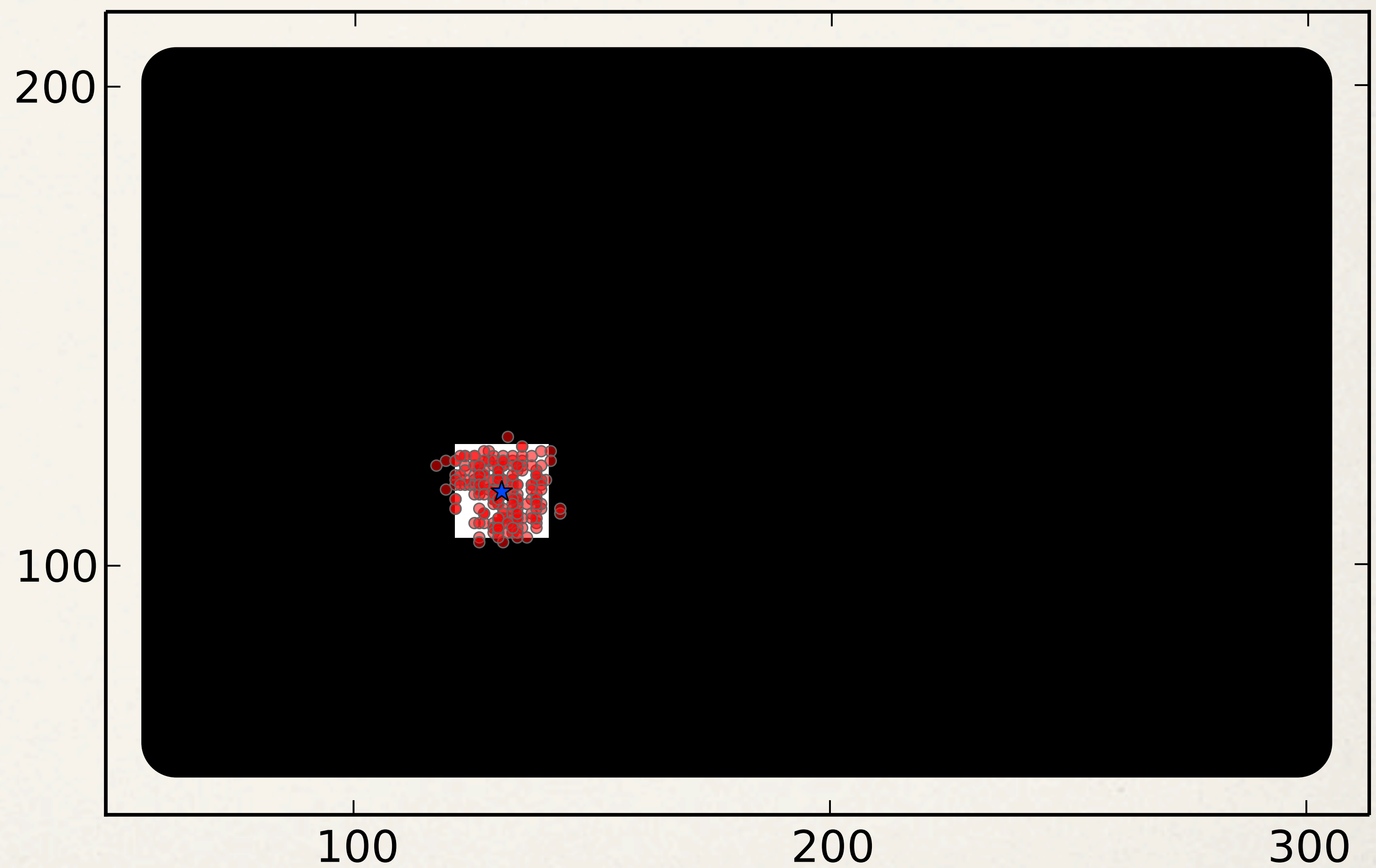
Simple tracking application



Background

Simple tracking application

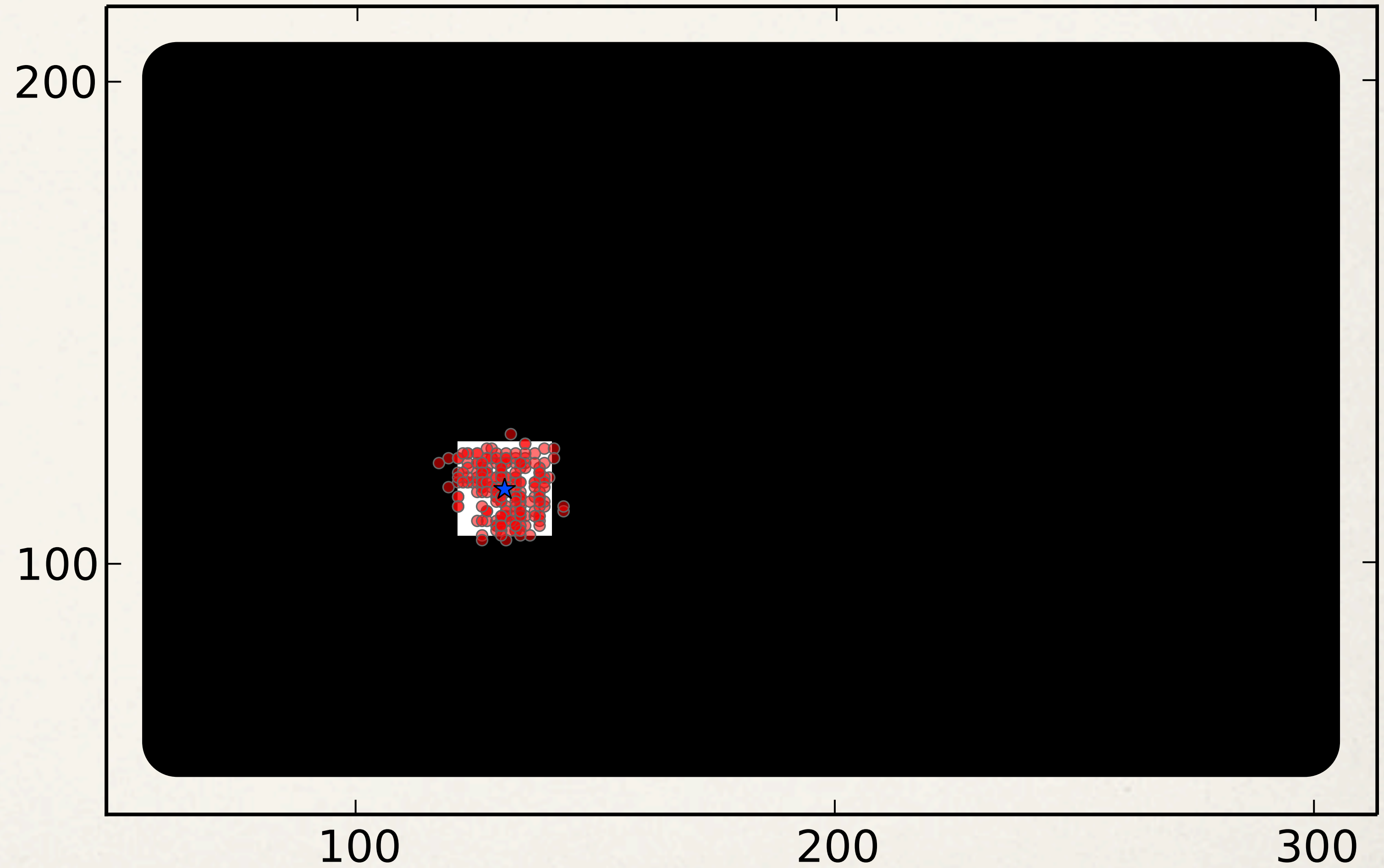
- Tracking a square on a dark background



Background

Simple tracking application

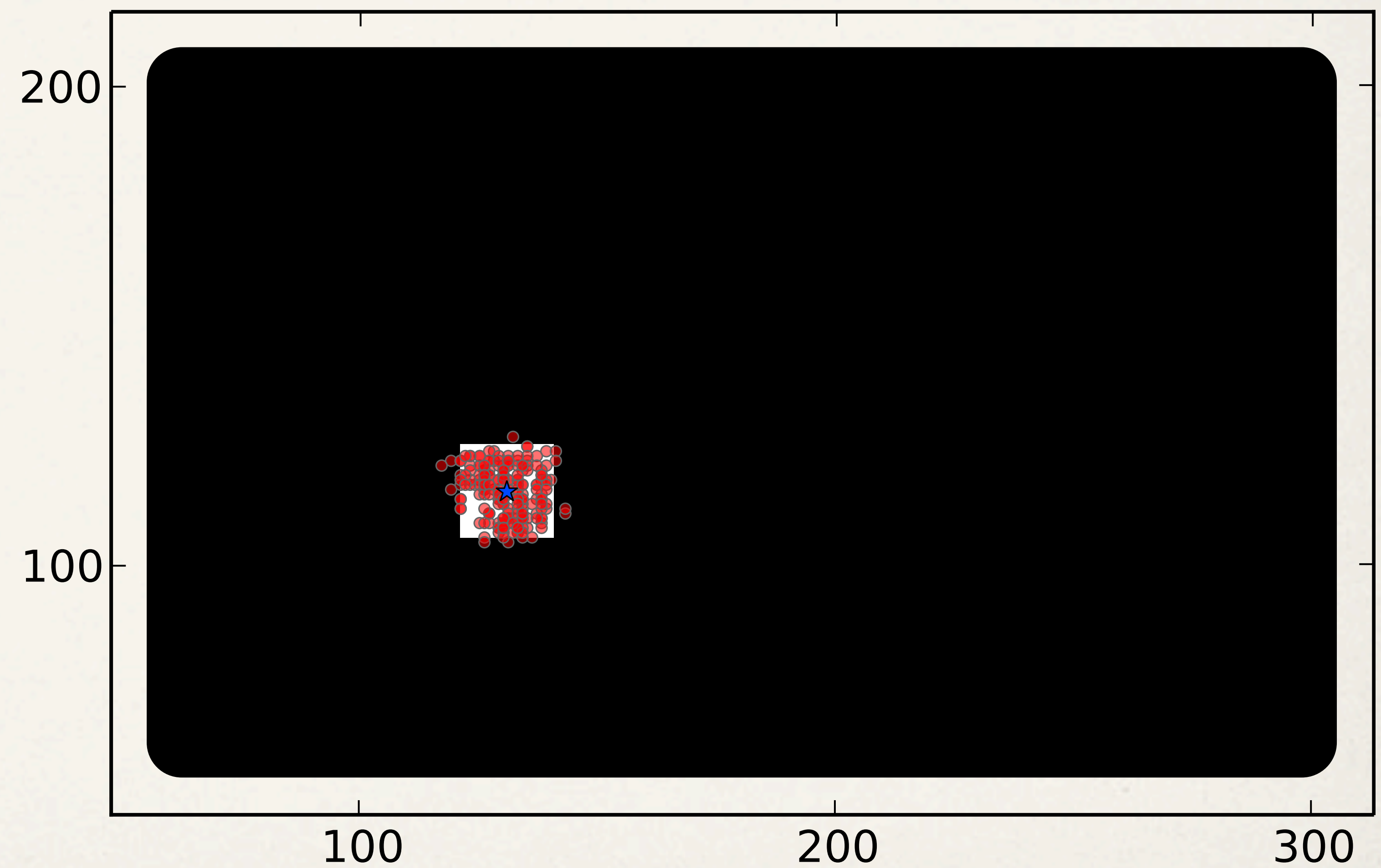
- Tracking a square on a dark background
  - Particle: $X^{(i)} = <x,y,\omega>$



Background

Simple tracking application

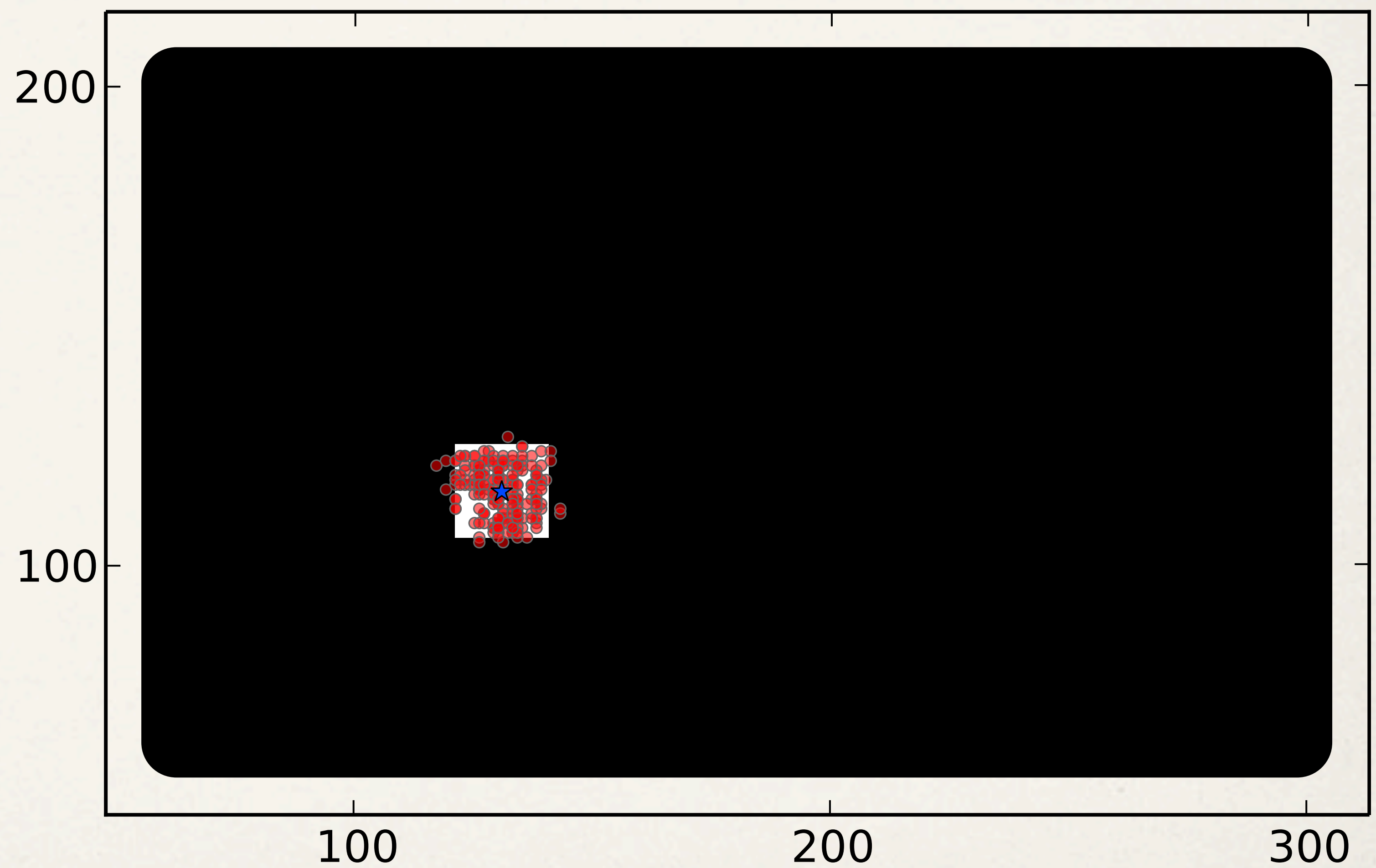
- Tracking a square on a dark background
  - Particle: $X^{(i)} = <x,y,\omega>$
- System dynamics



Background

Simple tracking application

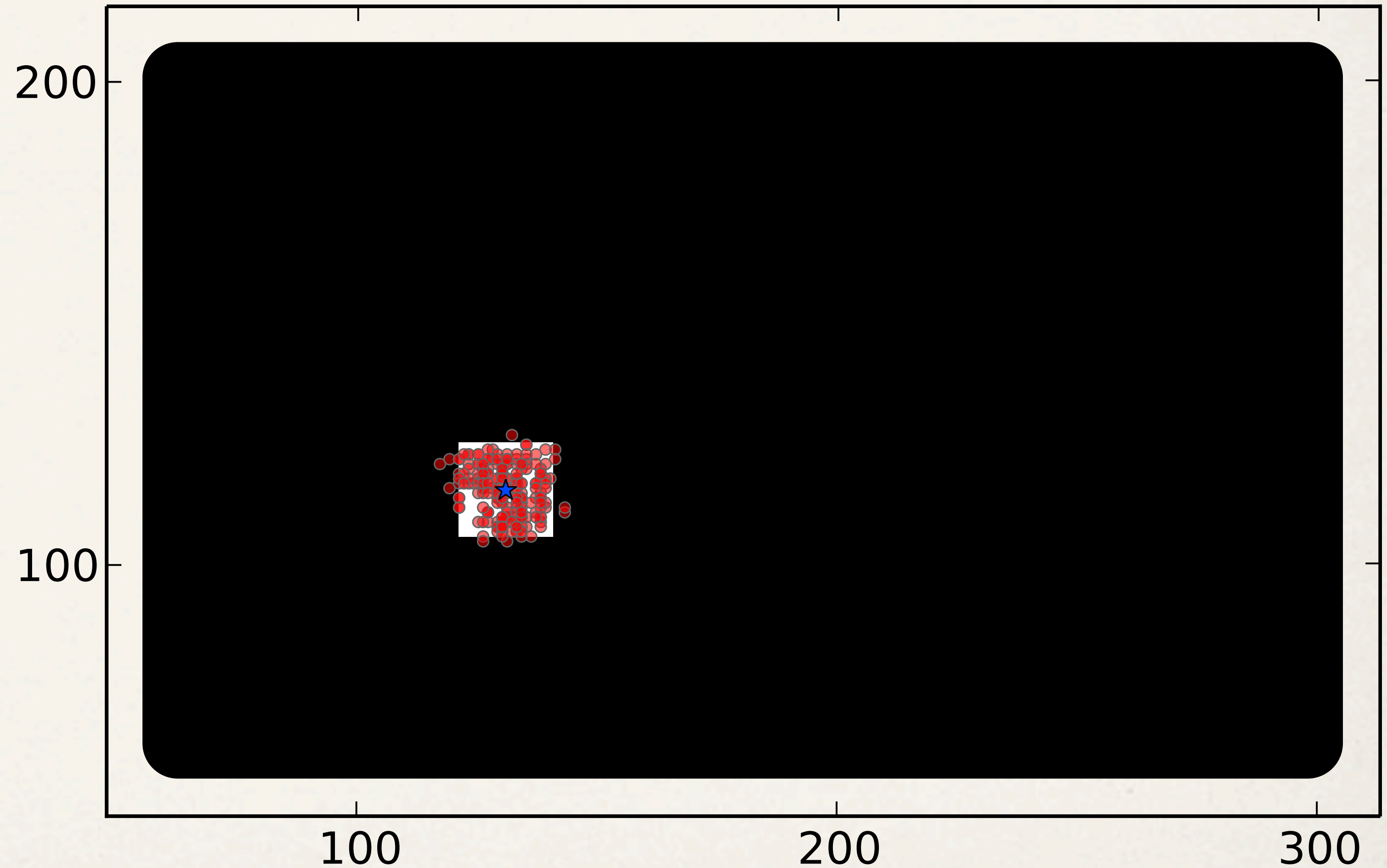
- Tracking a square on a dark background
  - Particle: $X^{(i)} = <x,y,\omega>$
- System dynamics
  - $(x', y') = (x + \delta_x, y + \delta_y)$ where $\delta_x, \delta_y \sim U(-a,a)$



Background

Simple tracking application

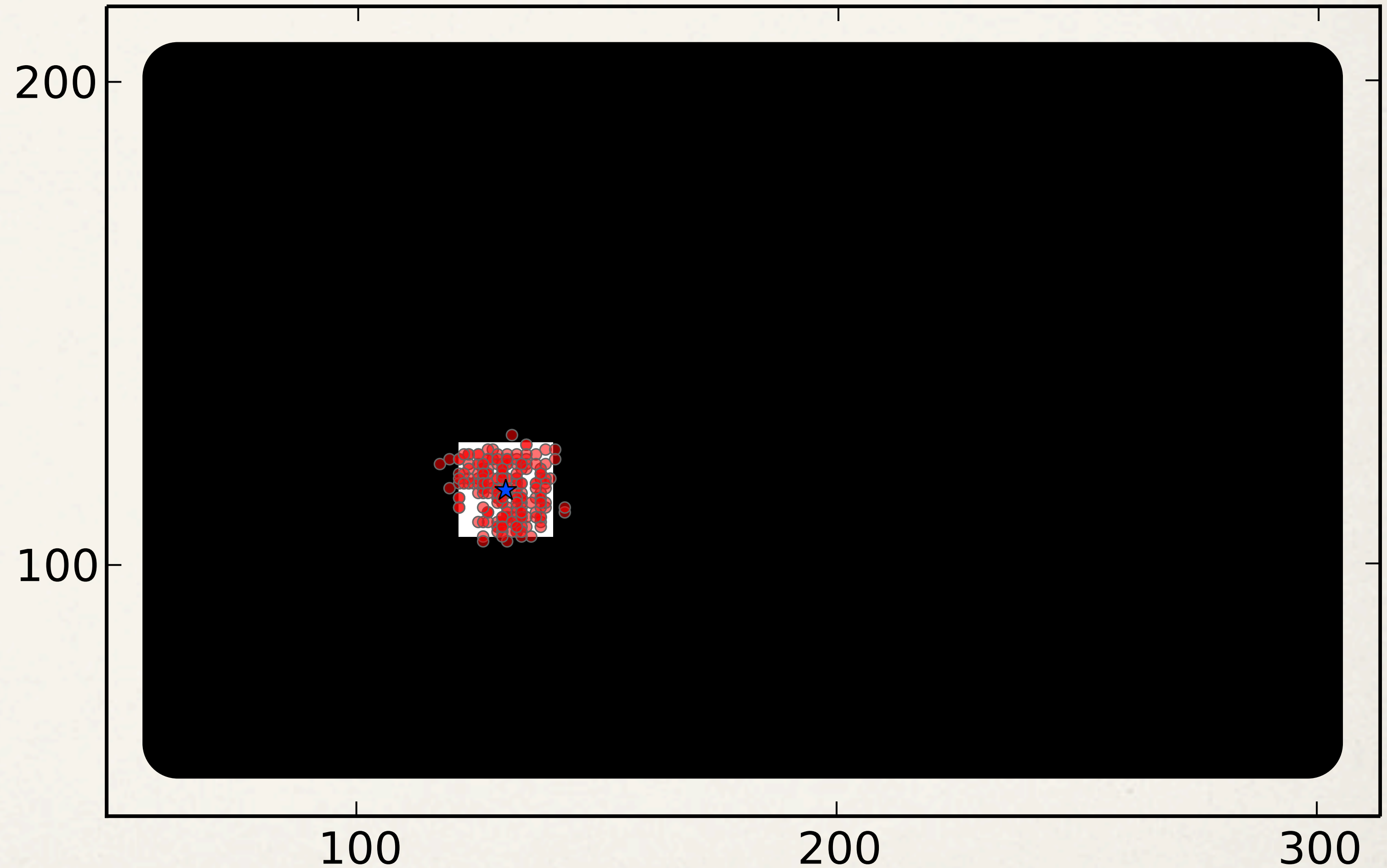
- Tracking a square on a dark background
  - Particle: $X^{(i)} = <x,y,\omega>$
- System dynamics
  - $(x', y') = (x + \delta_x, y + \delta_y)$ where $\delta_x, \delta_y \sim U(-a,a)$
- Measurement function



Background

Simple tracking application

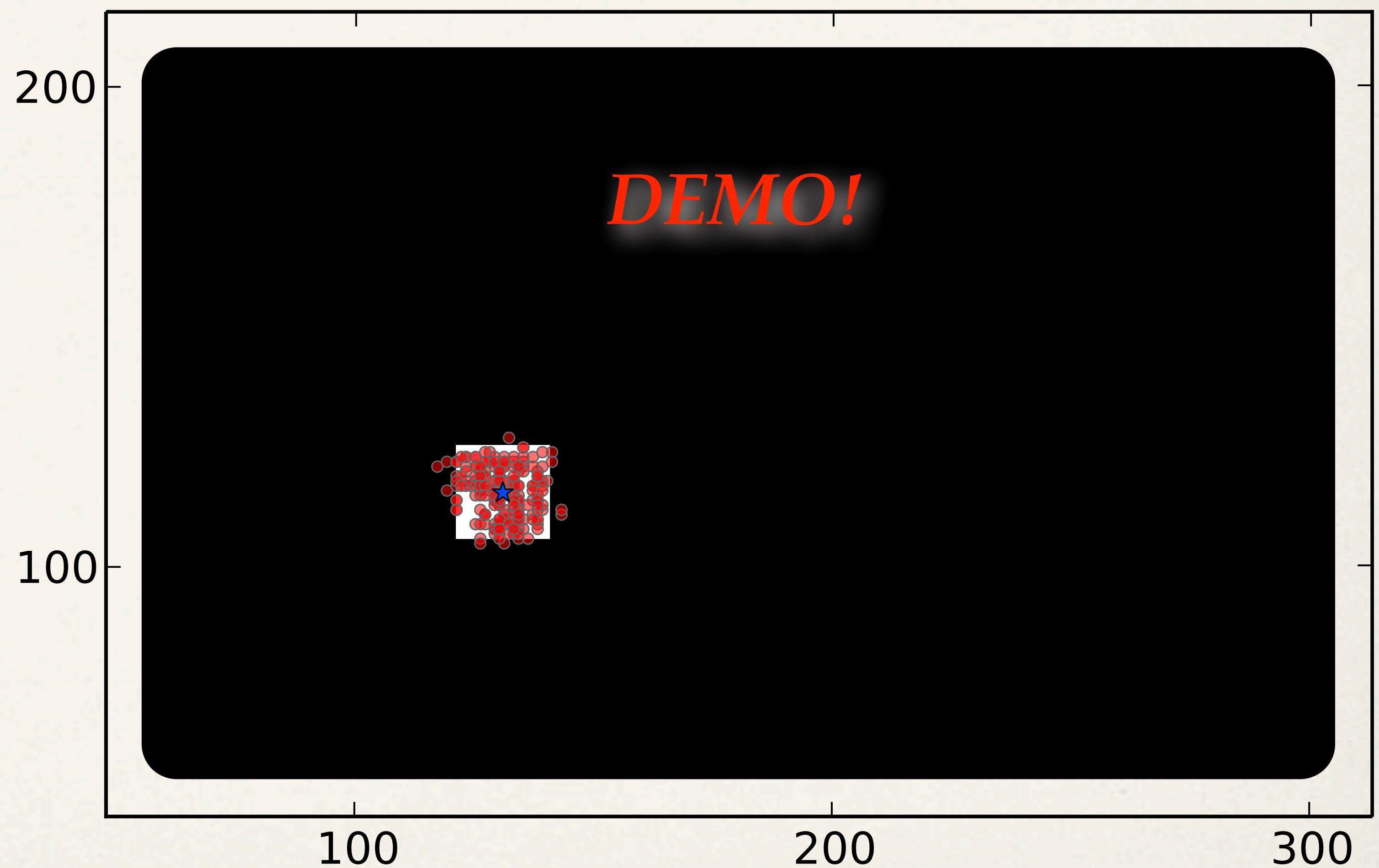
- Tracking a square on a dark background
  - Particle: $X^{(i)} = \langle x,y,\omega \rangle$
- System dynamics
  - $(x', y') = (x + \delta_x, y + \delta_y)$ where $\delta_x, \delta_y \sim U(-a,a)$
- Measurement function
  - $\omega = 1 / (1 + (255\text{-}pxl)^2)$



Background

Simple tracking application

- Tracking a square on a dark background
  - Particle: $X^{(i)} = \langle x,y,\omega \rangle$
- System dynamics
  - $(x', y') = (x + \delta_x, y + \delta_y)$ where $\delta_x, \delta_y \sim U(-a,a)$
- Measurement function
  - $\omega = 1 / (1 + (255\text{-}pxl)^2)$

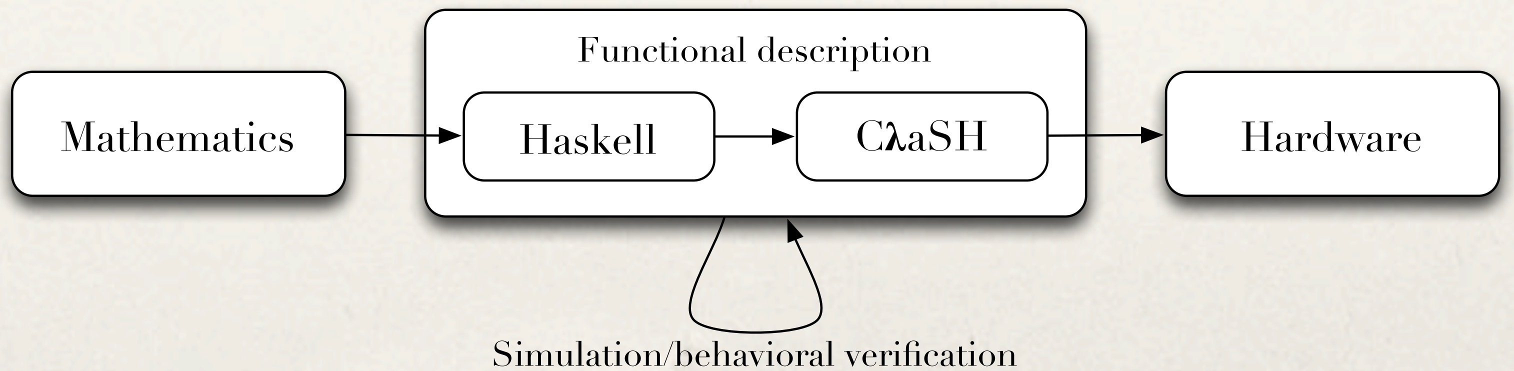


Implementing the particle filter

- Design method
- Math to Haskell
- Haskell to CλaSH

Design method

- First step
  - Reformulate the mathematics of Particle filtering into plain Haskell
- Second step
  - Apply small modifications to Haskell code such that it is accepted by the CλaSH compiler



Math to Haskell

- Apply the state space model to all particles $x_k^{(i)} = f(x_{k-1}^{(i)}, u_k)$
  - All operations are performed independently $f\left(x_k^{(i)}, u_k\right) = x_k^{(i)} + u_k$
  - Corresponding higher order function is ***zipWith***

$$\begin{aligned} &\text{predict } f\ ps\ us = ps' \\ &\quad \textbf{where} \\ &\qquad ps' = \textbf{zipWith}\ f\ ps\ us \end{aligned}$$

$$\begin{aligned} &f\ (x, y, \omega)\ (\delta_x, \delta_y) = (x', y', \omega) \\ &\quad \textbf{where} \\ &\qquad x' = x + \delta_x \\ &\qquad y' = y + \delta_y \end{aligned}$$

Math to Haskell

- Determine sum of weights and apply to all particles
  - Corresponding higher order functions are ***foldl*** and ***zipWith***

$$\tilde{\omega}^{(i)} = \frac{\omega^{(i)}}{\sum_{n=1}^{N} \omega^{(n)}} \quad \text{for} \quad i = 1 \ldots N$$

$$\begin{aligned}
&normalize\ ps = ps' \\
&\quad \mathbf{where} \\
&\qquad tot\omega = sum\ (\mathbf{map}\ weight\ ps) \\
&\qquad ps' \quad = \mathbf{map}\ (\lambda\ (x, y, \omega) \to (x, y, \omega\ /\ tot\omega))\ ps
\end{aligned}$$

Haskell to CλaSH

- Translate lists to Vectors

$$
\begin{aligned}
&\mathit{predict} :: (\mathit{Ptl} \rightarrow \mathit{Ns} \rightarrow \mathit{Ptl}) \rightarrow [\mathit{Ptl}] \rightarrow [\mathit{Ns}] \rightarrow [\mathit{Ptl}] \\
&\mathit{predict}\ \ f\ \ \mathit{ps}\ \ \mathit{us} = \mathit{ps}' \\
&\quad \mathbf{where} \\
&\qquad \mathit{ps}' = \mathit{zipWith}\ f\ \mathit{ps}\ \mathit{us}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{predict} :: (\mathit{Ptl} \rightarrow \mathit{Ns} \rightarrow \mathit{Ptl}) \rightarrow (\mathit{Vector}\ \mathit{D32}\ \mathit{Ptl}) \rightarrow (\mathit{Vector}\ \mathit{D32}\ \mathit{Ns}) \rightarrow (\mathit{Vector}\ \mathit{D32}\ \mathit{Ptl}) \\
&\mathit{predict}\ \ f\ \ \mathit{ps}\ \ \mathit{us} = \mathit{ps}' \\
&\quad \mathbf{where} \\
&\qquad \mathit{ps}' = \mathit{vzipWith}\ f\ \mathit{ps}\ \mathit{us}
\end{aligned}
$$

Haskell to CλaSH

- Translate lists to Vectors

$$
\begin{aligned}
&\mathit{predict} :: (\mathit{Ptl} \rightarrow \mathit{Ns} \rightarrow \mathit{Ptl}) \rightarrow [\mathit{Ptl}] \rightarrow [\mathit{Ns}] \rightarrow [\mathit{Ptl}] \\
&\mathit{predict}\ \ f\ \ \mathit{ps}\ \ \mathit{us} = \mathit{ps}' \\
&\quad \textbf{where} \\
&\qquad \mathit{ps}' = \mathit{zipWith}\ f\ \mathit{ps}\ \mathit{us}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{predict} :: (\mathit{Ptl} \rightarrow \mathit{Ns} \rightarrow \mathit{Ptl}) \rightarrow (\mathit{Vector}\ \mathit{D32}\ \mathit{Ptl}) \rightarrow (\mathit{Vector}\ \mathit{D32}\ \mathit{Ns}) \rightarrow (\mathit{Vector}\ \mathit{D32}\ \mathit{Ptl}) \\
&\mathit{predict}\ \ f\ \ \mathit{ps}\ \ \mathit{us} = \mathit{ps}' \\
&\quad \textbf{where} \\
&\qquad \mathit{ps}' = \mathit{vzipWith}\ f\ \mathit{ps}\ \mathit{us}
\end{aligned}
$$

Haskell to CλaSH

- Translate lists to Vectors
- Use fixed point representation for weights

$$
\begin{aligned}
&normalize :: [Ptl] \to [Ptl] \\
&normalize \;\; ps \;\; = ps' \\
&\quad \textbf{where} \\
&\qquad tot\omega = sum\ (map\ weight\ ps) \\
&\qquad ps \;\; = map\ (\lambda\ (x, y, \omega) \to (x, y, \omega\ /\ tot\omega))\ ps
\end{aligned}
$$

$$
\begin{aligned}
&normalize :: (Vector\ D32\ Ptl) \to (Vector\ D32\ Ptl) \\
&normalize \;\; ps \;\; = ps' \\
&\quad \textbf{where} \\
&\qquad tot\omega \;\; = sum\ (vmap\ weight\ ps) \\
&\qquad tot\omega_{recip} = fprecip\ tot\omega \\
&\qquad ps \;\; = vmap\ (\lambda\ (x, y, \omega) \to (x, y, \omega * tot\omega_{recip}))\ ps
\end{aligned}
$$

Haskell to CλaSH

- Translate lists to Vectors
- Use fixed point representation for weights

$$
\begin{aligned}
&\mathit{normalize} :: [\mathit{Ptl}] \rightarrow [\mathit{Ptl}] \\
&\mathit{normalize}\ \ \mathit{ps} = \mathit{ps}' \\
&\quad \textbf{where} \\
&\qquad \mathit{tot}\omega = \mathit{sum}\ (\mathit{map}\ \mathit{weight}\ \mathit{ps}) \\
&\qquad \mathit{ps}\ \ = \mathit{map}\ (\lambda\ (x, y, \omega) \rightarrow (x, y, \omega\ /\ \mathit{tot}\omega))\ \mathit{ps}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{normalize} :: (\mathit{Vector}\ \mathit{D32}\ \mathit{Ptl}) \rightarrow (\mathit{Vector}\ \mathit{D32}\ \mathit{Ptl}) \\
&\mathit{normalize}\ \ \mathit{ps} = \mathit{ps}' \\
&\quad \textbf{where} \\
&\qquad \mathit{tot}\omega = \mathit{sum}\ (\mathit{vmap}\ \mathit{weight}\ \mathit{ps}) \\
&\qquad \mathit{tot}\omega_{recip} = \mathit{fprecip}\ \mathit{tot}\omega \\
&\qquad \mathit{ps} = \mathit{vmap}\ (\lambda\ (x, y, \omega) \rightarrow (x, y, \omega * \mathit{tot}\omega_{recip}))\ \mathit{ps}
\end{aligned}
$$

Haskell to CλaSH

- Translate lists to Vectors
- Use fixed point representation for weights

$$
\begin{aligned}
&normalize :: [Ptl] \to [Ptl] \\
&normalize\ ps = ps' \\
&\quad \textbf{where} \\
&\qquad tot\omega = sum\ (map\ weight\ ps) \\
&\qquad ps' = map\ (\lambda\ (x, y, \omega) \to (x, y, \omega\ /\ tot\omega))\ ps
\end{aligned}
$$

$$
\begin{aligned}
&normalize :: (Vector\ D32\ Ptl) \to (Vector\ D32\ Ptl) \\
&normalize\ ps = ps' \\
&\quad \textbf{where} \\
&\qquad tot\omega = sum\ (vmap\ weight\ ps) \\
&\qquad tot\omega_{recip} = fprecip\ tot\omega \\
&\qquad ps' = vmap\ (\lambda\ (x, y, \omega) \to (x, y, \omega * tot\omega_{recip}))\ ps
\end{aligned}
$$

Haskell to CλaSH

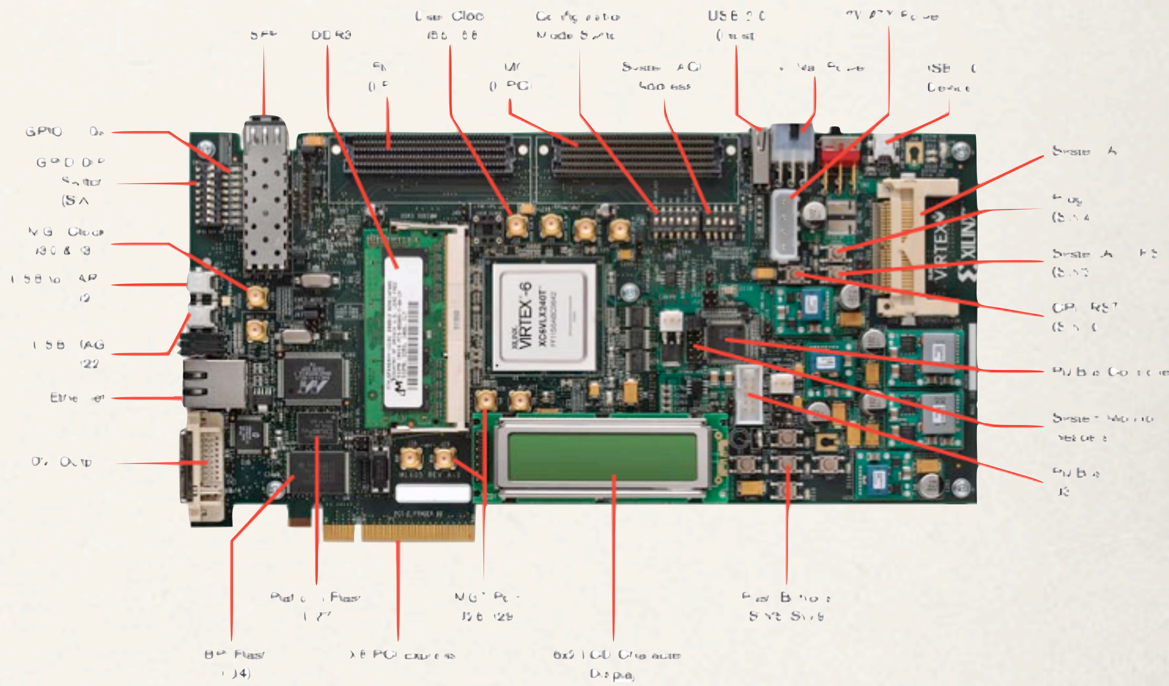
- Translate lists to Vectors
- Use fixed point representation for weights

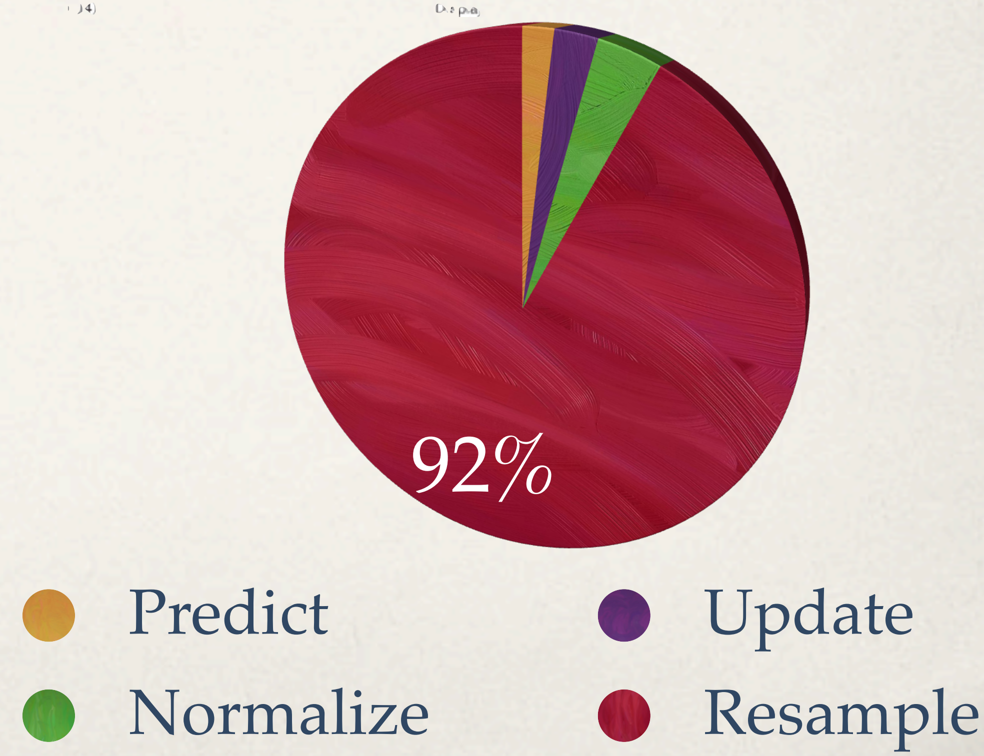
$$
\begin{aligned}
&normalize :: [Ptl] \rightarrow [Ptl] \\
&normalize\ \ ps = ps' \\
&\quad \textbf{where} \\
&\qquad tot\omega = sum\ (map\ weight\ ps) \\
&\qquad ps = map\ (\lambda\ (x, y, \omega) \rightarrow (x, y, \omega\ /\ tot\omega))\ ps
\end{aligned}
$$

$$
\begin{aligned}
&normalize :: (Vector\ D32\ Ptl) \rightarrow (Vector\ D32\ Ptl) \\
&normalize\ \ ps = ps' \\
&\quad \textbf{where} \\
&\qquad tot\omega = sum\ (vmap\ weight\ ps) \\
&\qquad tot\omega_{recip} = fprecip\ tot\omega \\
&\qquad ps = vmap\ (\lambda\ (x, y, \omega) \rightarrow (x, y, \omega * tot\omega_{recip}))\ ps
\end{aligned}
$$

Results

- Parallel particle filter with 32 particles synthesized for FPGA
  - Area = about 40k LUTs
  - PF can be synthesized but is slow
  - Resampling step is bottleneck in both area and clockfrequency
- For larges PFs, we need a trade off between area and execution time





Conclusions

- A completely parallel Particle Filter has been implemented
- Higher order functions are a natural way to to reason about structure in both the mathematical formulation and hardware
- Haskell code needs only small modifications before it is accepted by the CλaSH compiler
- Fully parallel resampling is a bottleneck in both area and clock frequency

Future Work

- Extend particle filter to more particles and more complex tracking
- Develop area vs time time trade off based on functional description

Questions ?