An Adaptive Sequential Monte Carlo Framework with Runtime HW/SW Partitioning

Markus Happe
Enno Lübbers
Marco Platzner

{ markus.happe | enno.luebbers | platzner } @ uni-paderborn.de

Computer Engineering Group
University of Paderborn, Germany

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

The 2009 International Conference on Field-Programmable Technology (FPT'09)
Sydney, Australia, 10.12.2009

Outline

- motivation
- multithreaded OS for reconfigurable devices
  - programming model
  - execution model
- sequential Monte Carlo framework
  - sampling-importance-resampling algorithm
  - application modeling
  - runtime adaptation
- experimental results
- conclusion & outlook

Motivation

- sequential Monte Carlo (SMC) methods
  - on-line estimation of internal state of non-linear dynamic systems
  - track a number of state estimates (i.e. particles) over time
  - used for object tracking [Woelk 2005], data stream classification [Granmo 2005], ...
  - iterative methods
  - tracking accuracy generally increases with number of particles
    - ➡ computationally intensive

estimated system state (particle)

- position
- velocity
- size

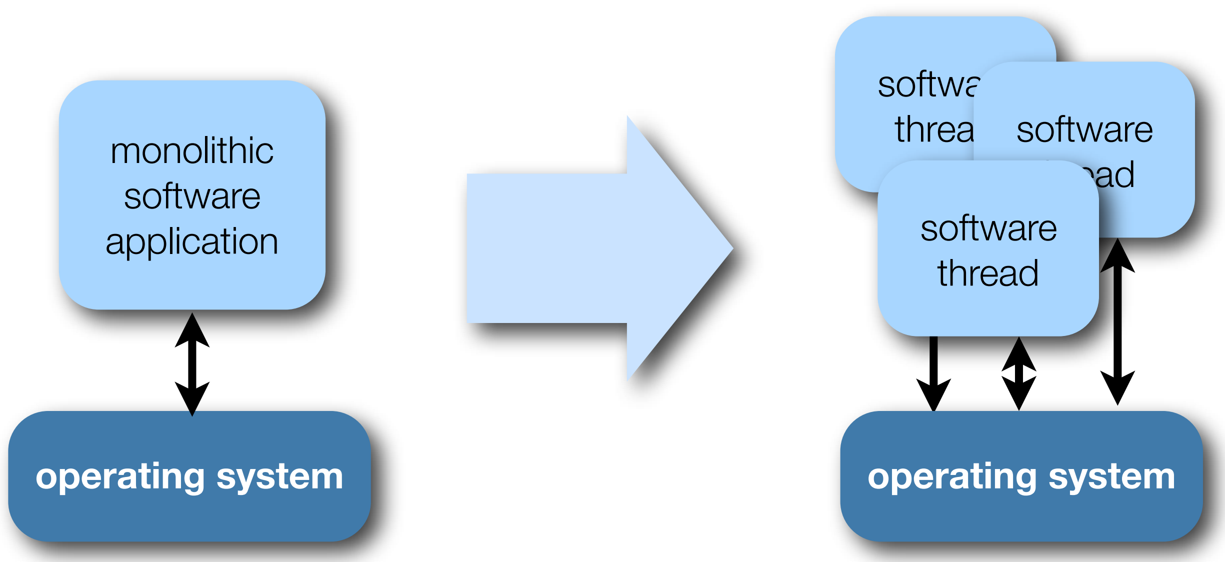
Motivation

- sequential Monte Carlo (SMC) methods
  - on-line estimation of internal state of non-linear dynamic systems
  - track a number of state estimates (i.e. particles) over time
  - used for object tracking [Woelk 2005], data stream classification [Granmo 2005], ...
  - iterative methods
  - tracking accuracy generally increases with number of particles
    - ➡ computationally intensive

- embedded SMC applications need hardware support
  - existing work [Athalye 2005, Sankaranarayanan 2005] focuses on HW-only systems
  - SMC-based algorithms have both **control-dominated** and **data-parallel** parts
    - combine software-based implementation with specialized hardware accelerators
  - ➡ flexible SMC framework for hybrid HW/SW systems

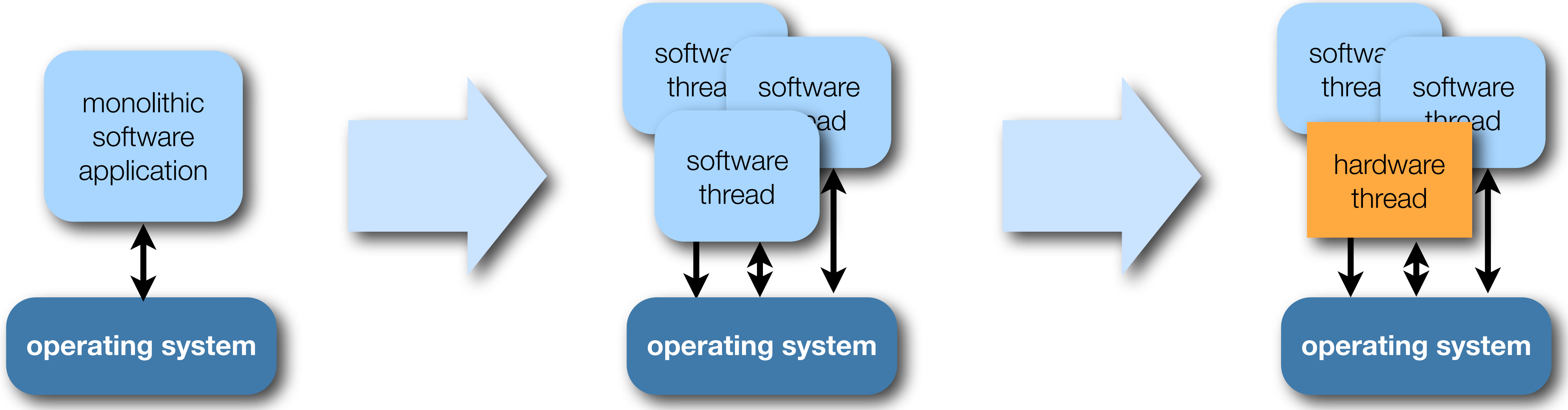
ReconOS

- multithreaded programming model
  - application is partitioned into **threads**
  - threads communicate and synchronize using **programming model primitives** e.g., *semaphores*, *mutexes*, *mailboxes*, *shared memory* provided by an OS
  - established model in software-based systems (e.g., POSIX pthreads, eCos)



ReconOS

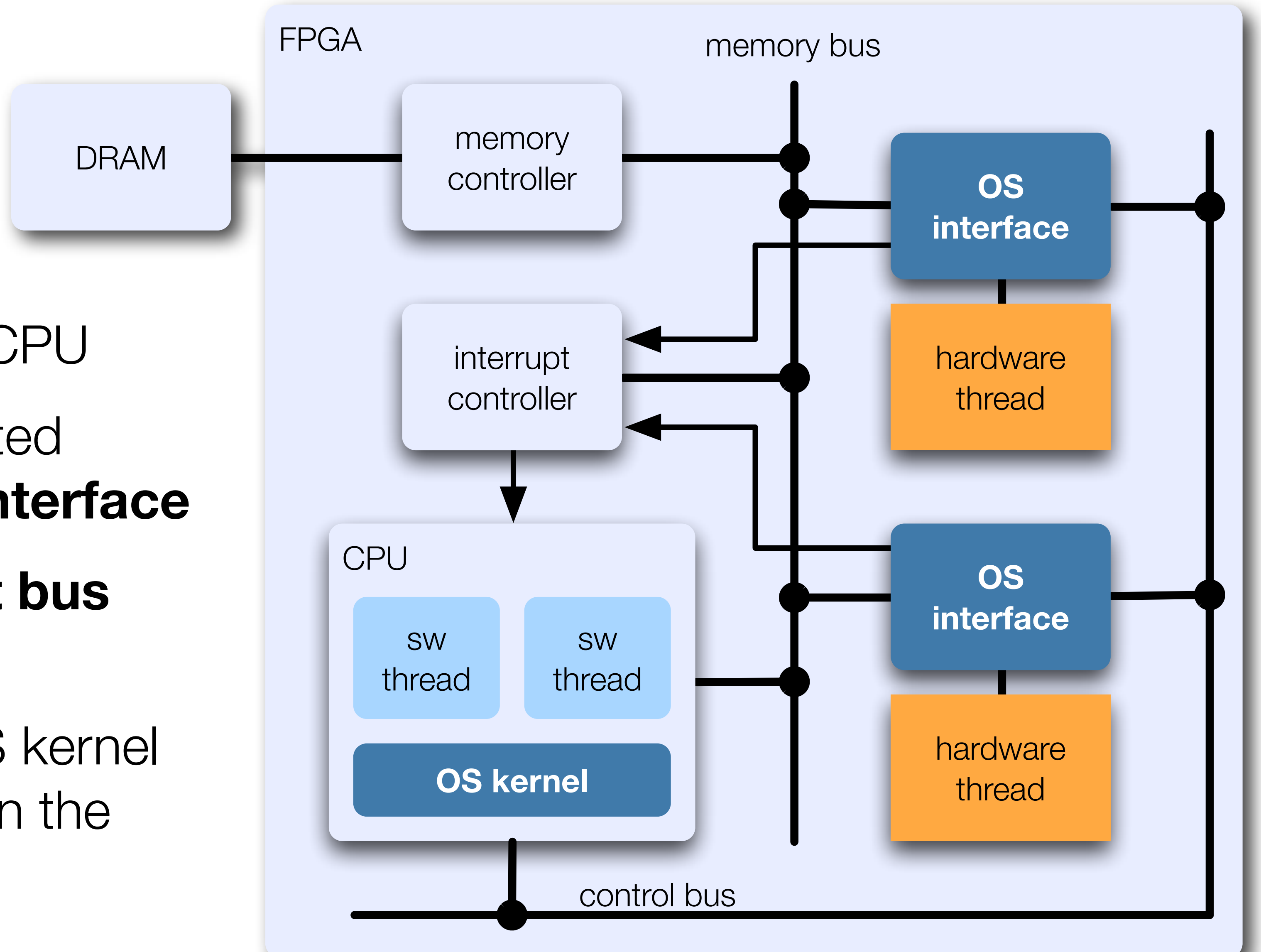
- multithreaded programming model
  - application is partitioned into **threads**
  - threads communicate and synchronize using **programming model primitives** e.g., *semaphores*, *mutexes*, *mailboxes*, *shared memory* provided by an OS
  - established model in software-based systems (e.g., POSIX pthreads, eCos)



- extension to reconfigurable hardware (ReconOS)
  - hardware modules are modeled as **hardware threads**
  - communicate and synchronize seamlessly with other threads

ReconOS

- execution model
  - implemented on Xilinx Virtex-4FX100
  - software operating system kernel (eCos) is executed on CPU
  - hardware threads are connected through **operating system interface**
  - hardware threads have **direct bus access** to shared memory
  - OS calls are relayed to the OS kernel through a **delegate thread** on the CPU

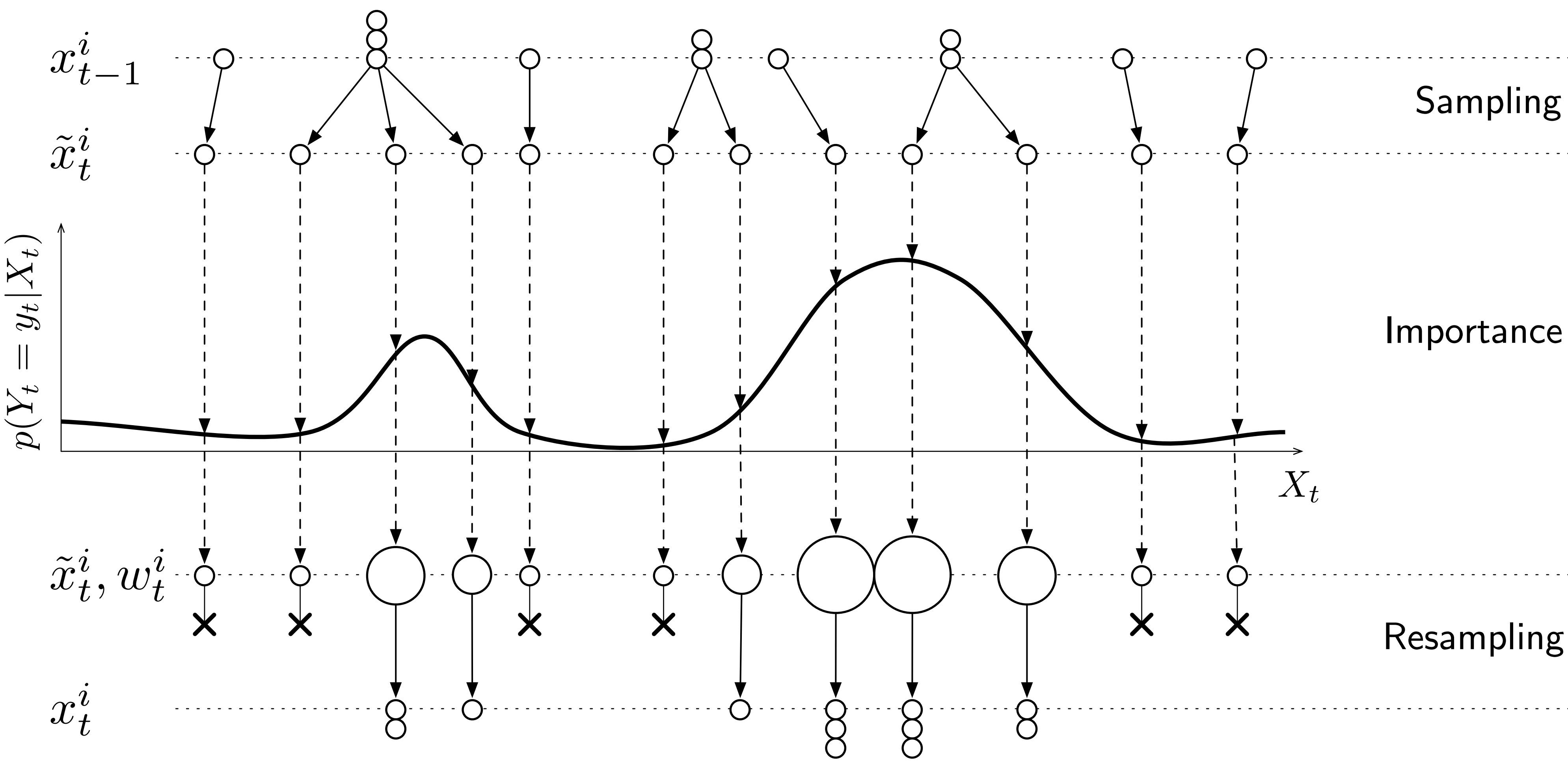


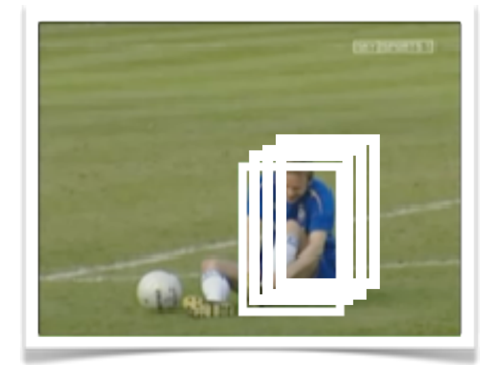
- run-time reconfiguration
  - hardware threads can be transparently reconfigured at run-time
  - scheduling of hardware threads is done in software

Outline

- motivation
- multithreaded OS for reconfigurable devices
  - programming model
  - execution model
- sequential Monte Carlo framework
  - sampling-importance-resampling algorithm
  - application modeling
  - runtime adaptation
- experimental results
- conclusion & outlook

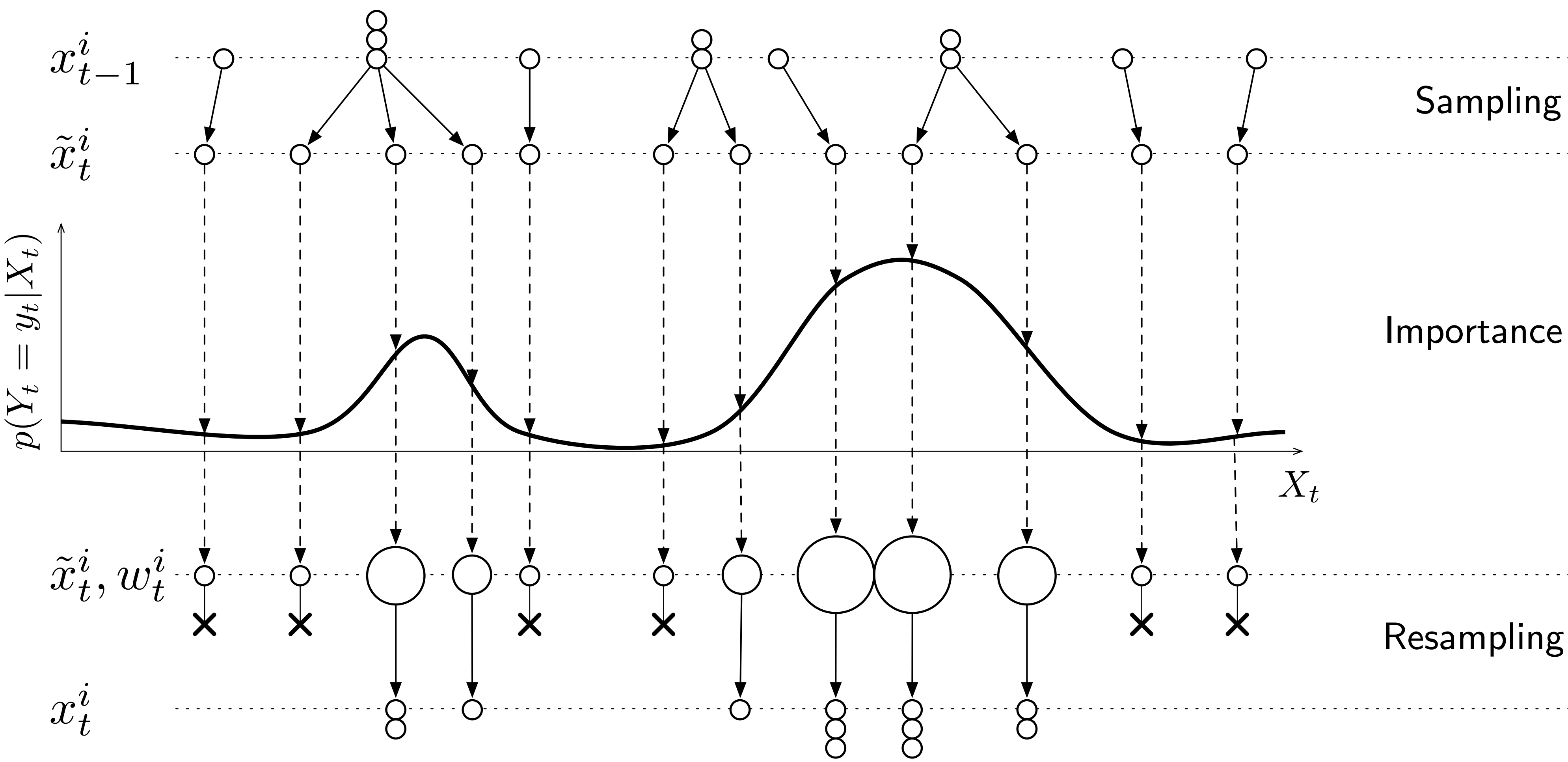
Sampling-Importance-Resampling (SIR)

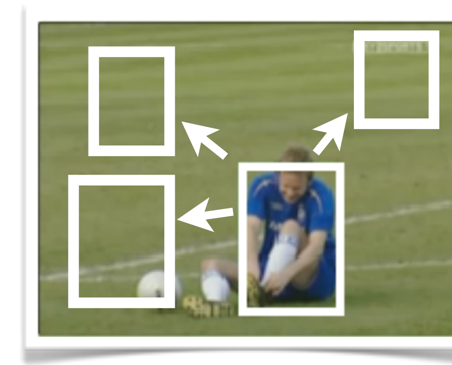




- iterative three-step algorithm
  - **sampling:** applies system model to generate new estimates
  - **importance:** weighs new particles according to new measurement
  - **resampling:** duplicates „good“ estimates, removes „bad“ estimates

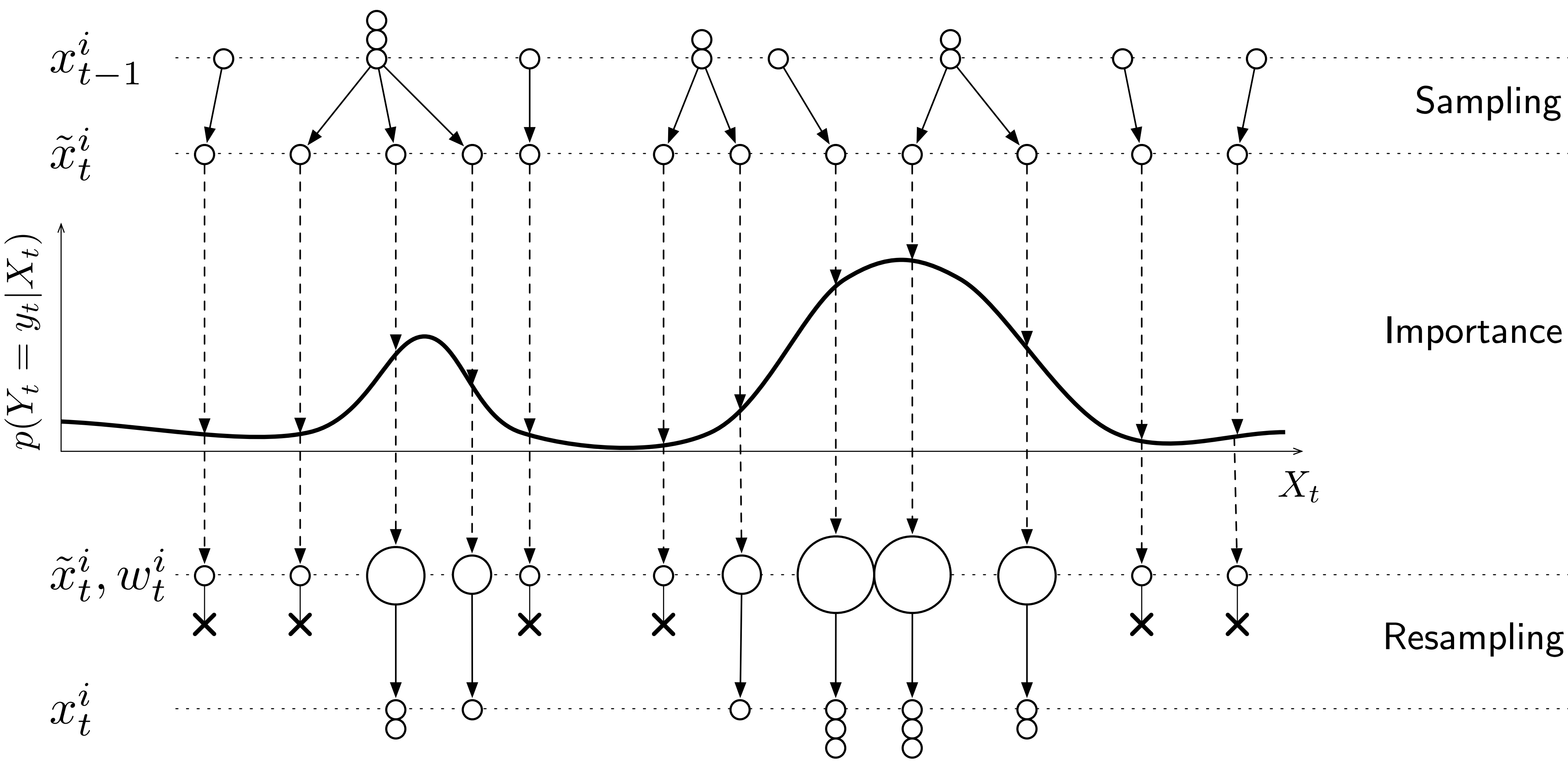
Sampling-Importance-Resampling (SIR)

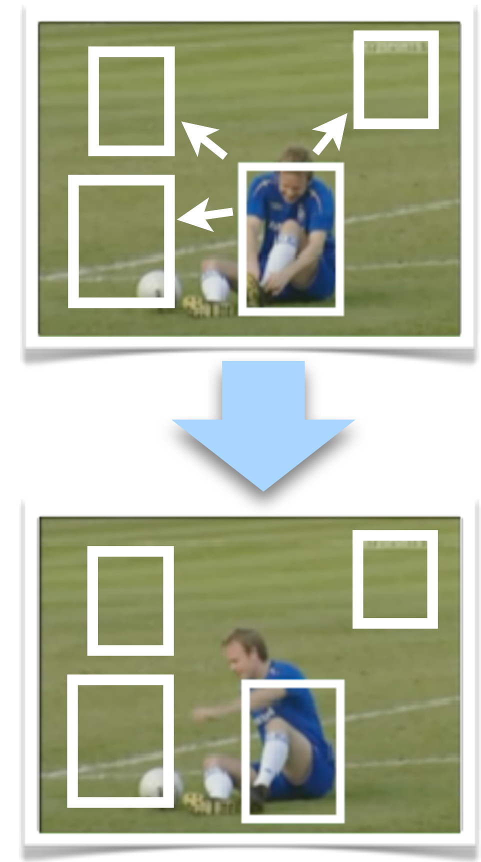




- iterative three-step algorithm
  - **sampling:** applies system model to generate new estimates
  - **importance:** weighs new particles according to new measurement
  - **resampling:** duplicates „good“ estimates, removes „bad“ estimates

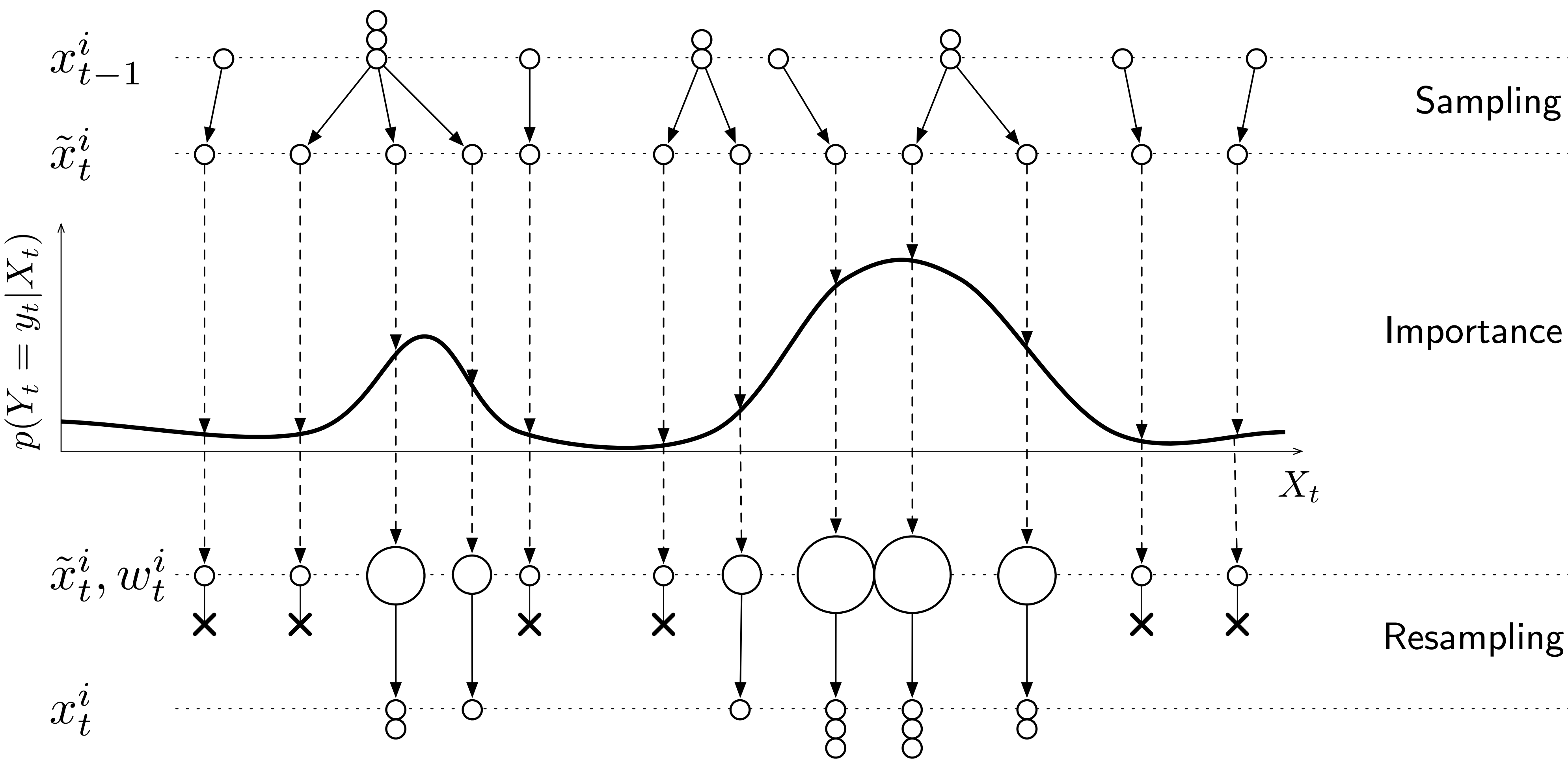
Sampling-Importance-Resampling (SIR)

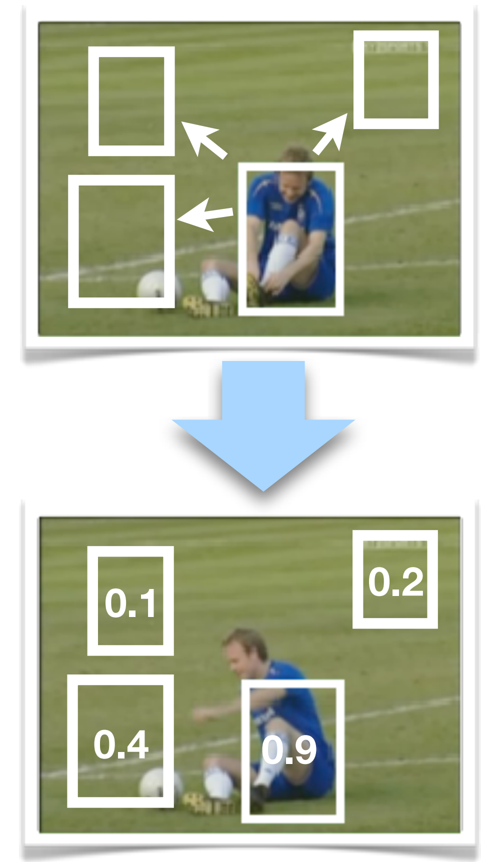




- iterative three-step algorithm
  - **sampling:** applies system model to generate new estimates
  - **importance:** weighs new particles according to new measurement
  - **resampling:** duplicates „good“ estimates, removes „bad“ estimates

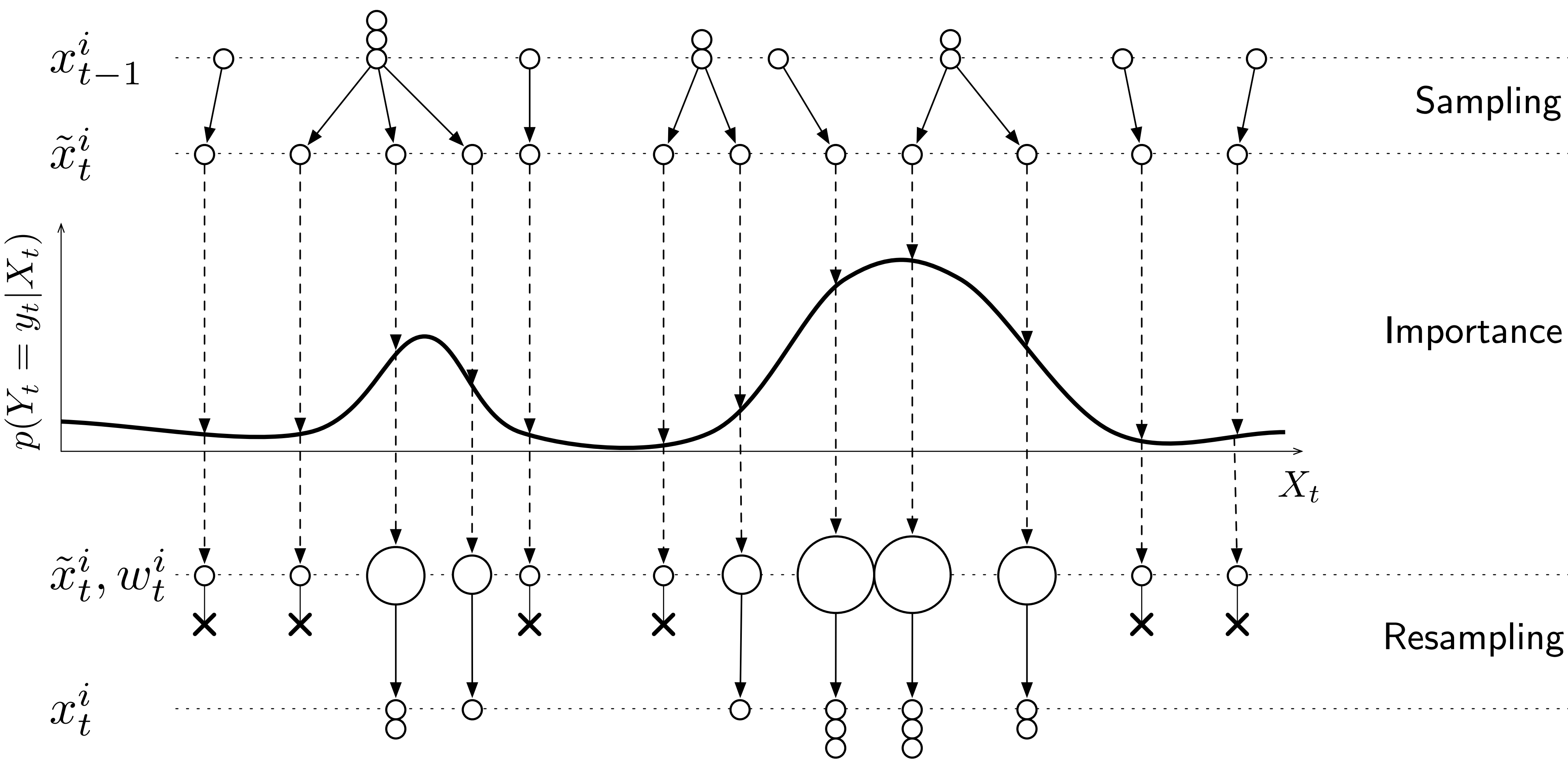
Sampling-Importance-Resampling (SIR)

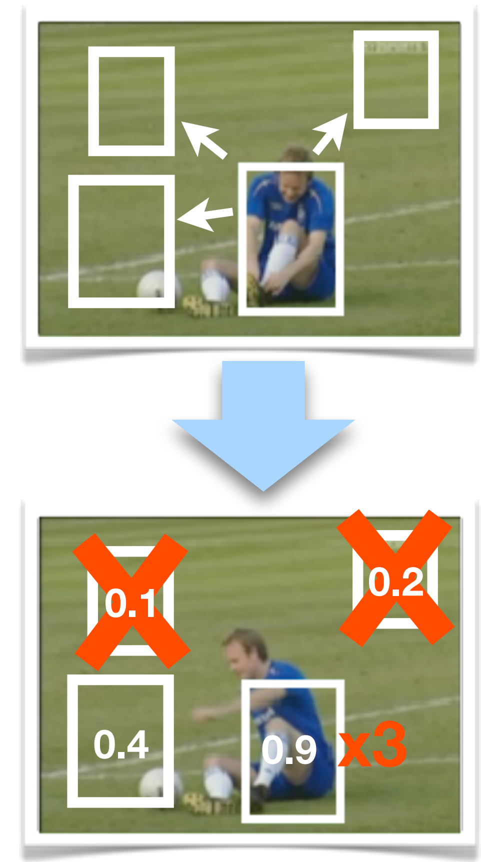




- iterative three-step algorithm
  - **sampling:** applies system model to generate new estimates
  - **importance:** weighs new particles according to new measurement
  - **resampling:** duplicates „good“ estimates, removes „bad“ estimates

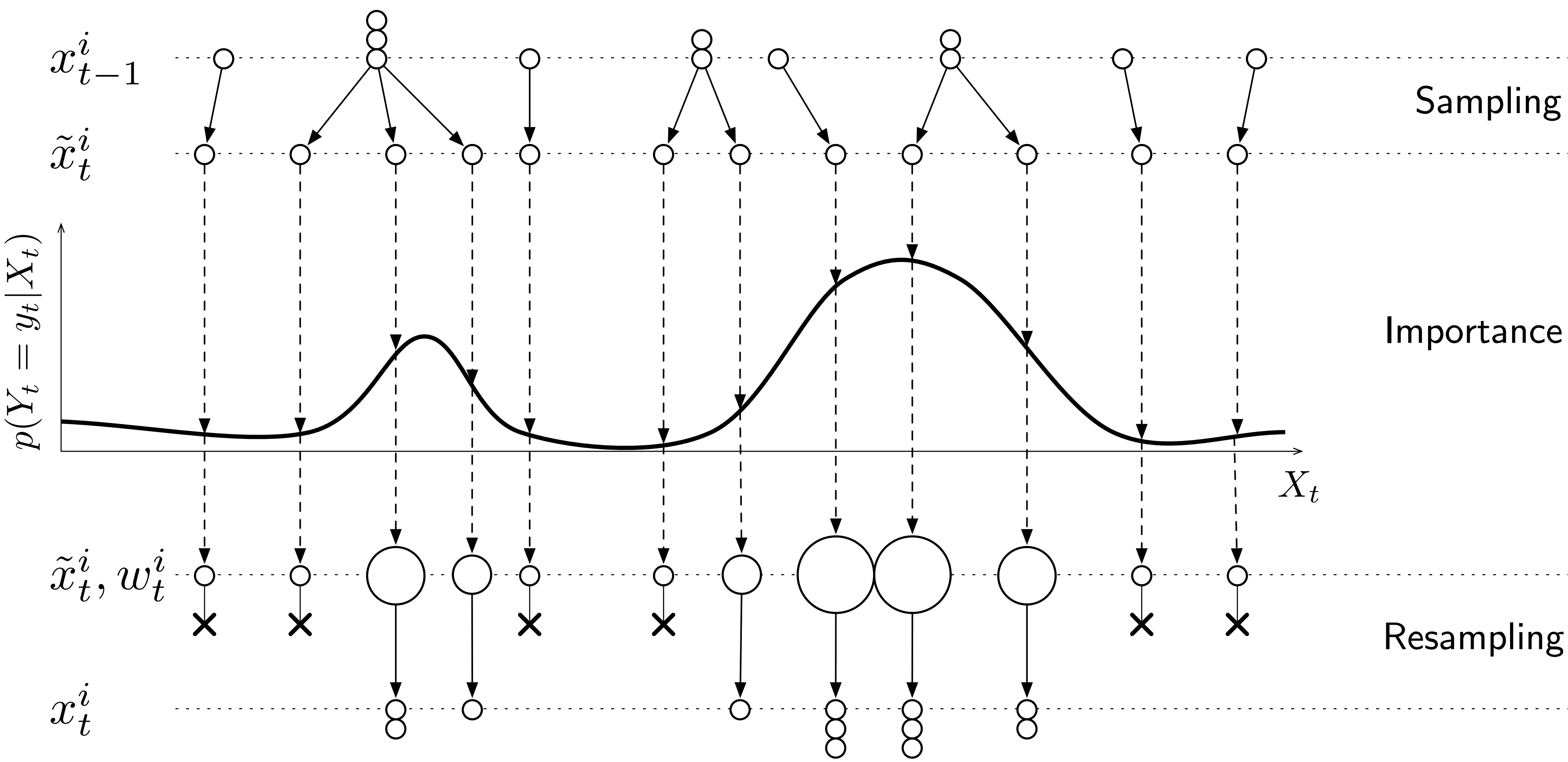
Sampling-Importance-Resampling (SIR)

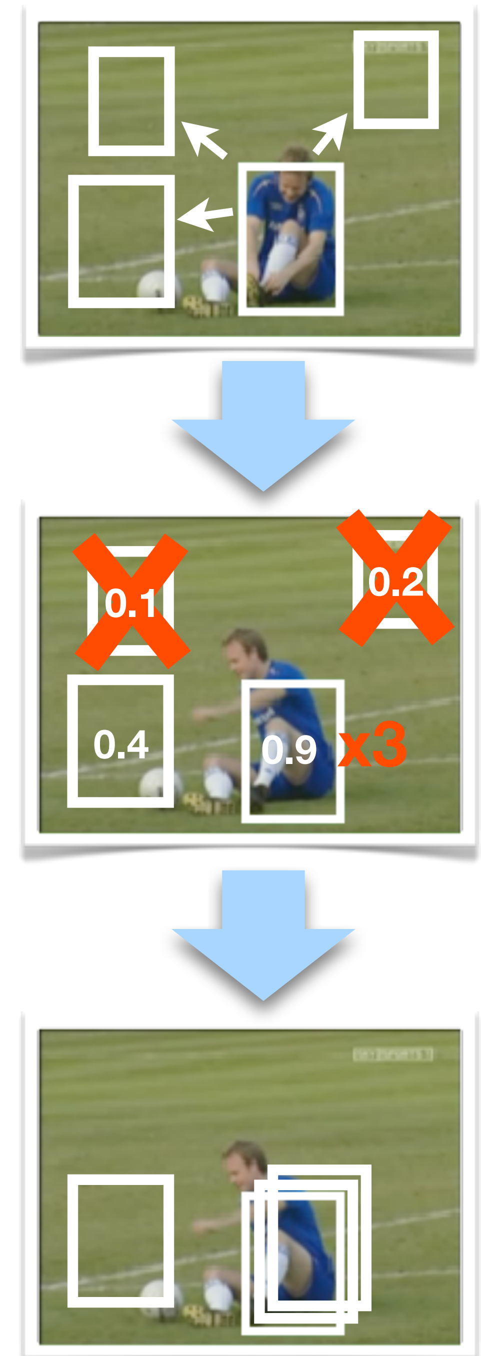




- iterative three-step algorithm
  - **sampling:** applies system model to generate new estimates
  - **importance:** weighs new particles according to new measurement
  - **resampling:** duplicates „good“ estimates, removes „bad“ estimates

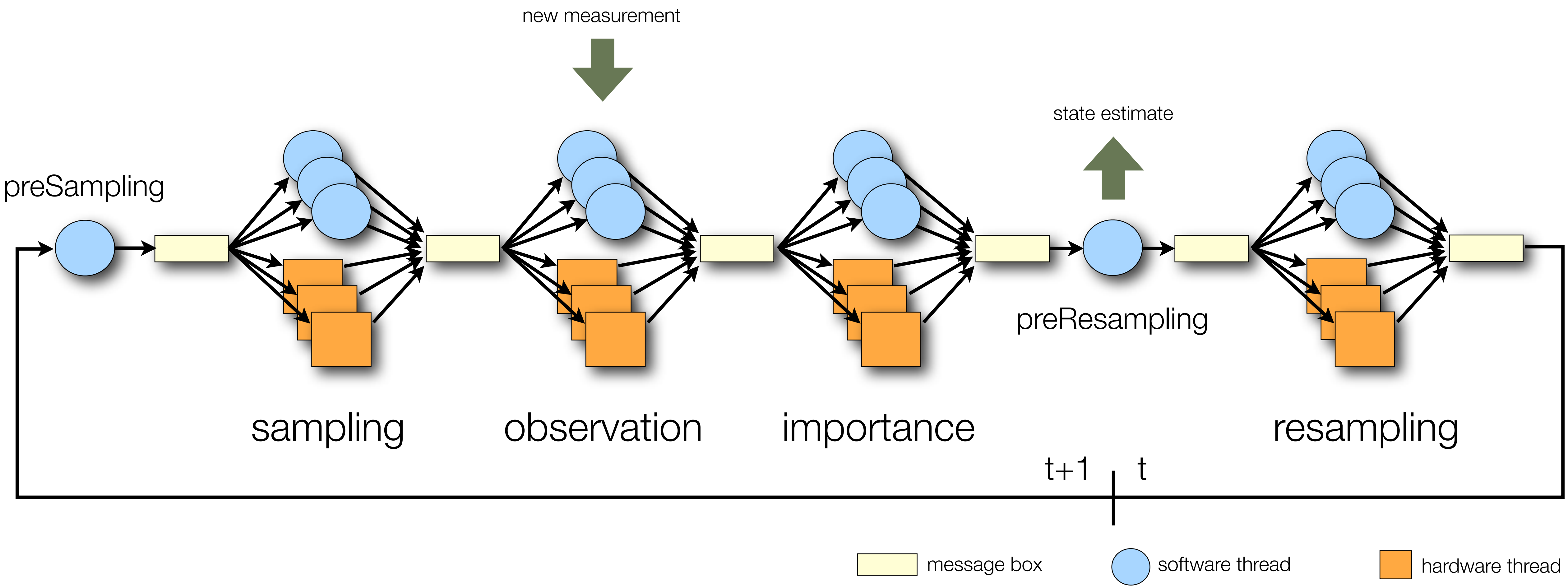
Sampling-Importance-Resampling (SIR)





- iterative three-step algorithm
  - **sampling:** applies system model to generate new estimates
  - **importance:** weighs new particles according to new measurement
  - **resampling:** duplicates „good“ estimates, removes „bad“ estimates

Multithreaded SMC Framework



- implementation with four stages
  - each stage can be implemented using multiple threads (hardware or software)
  - sampling, importance, and observation stages can process data in parallel
  - resampling stage needs data from all previous stages
  - preSampling and preResampling threads synchronize iterations and manage data granularity

Application Modeling

- system model
  - predicts new particle based on previous particle

$$p(X_t|X_{t-1} = x^i_{t-1})$$

`prediction()`

- observation model
  - extracts relevant features from given measurement

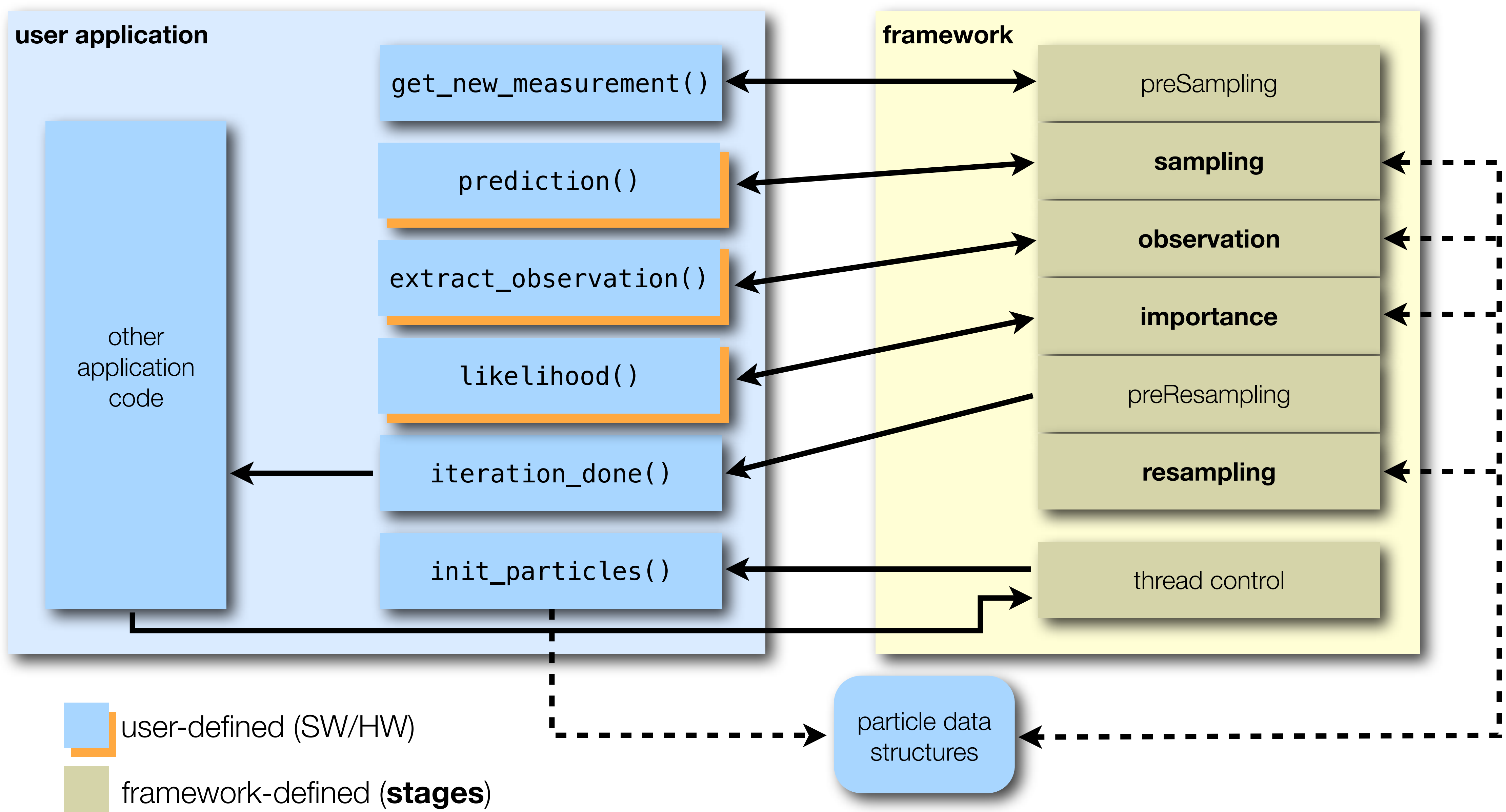
`extract_observation()`

- measurement model
  - determines likelihood that the current measurement fits the predicted state

$$p(Y_t = y_t|X_t)$$

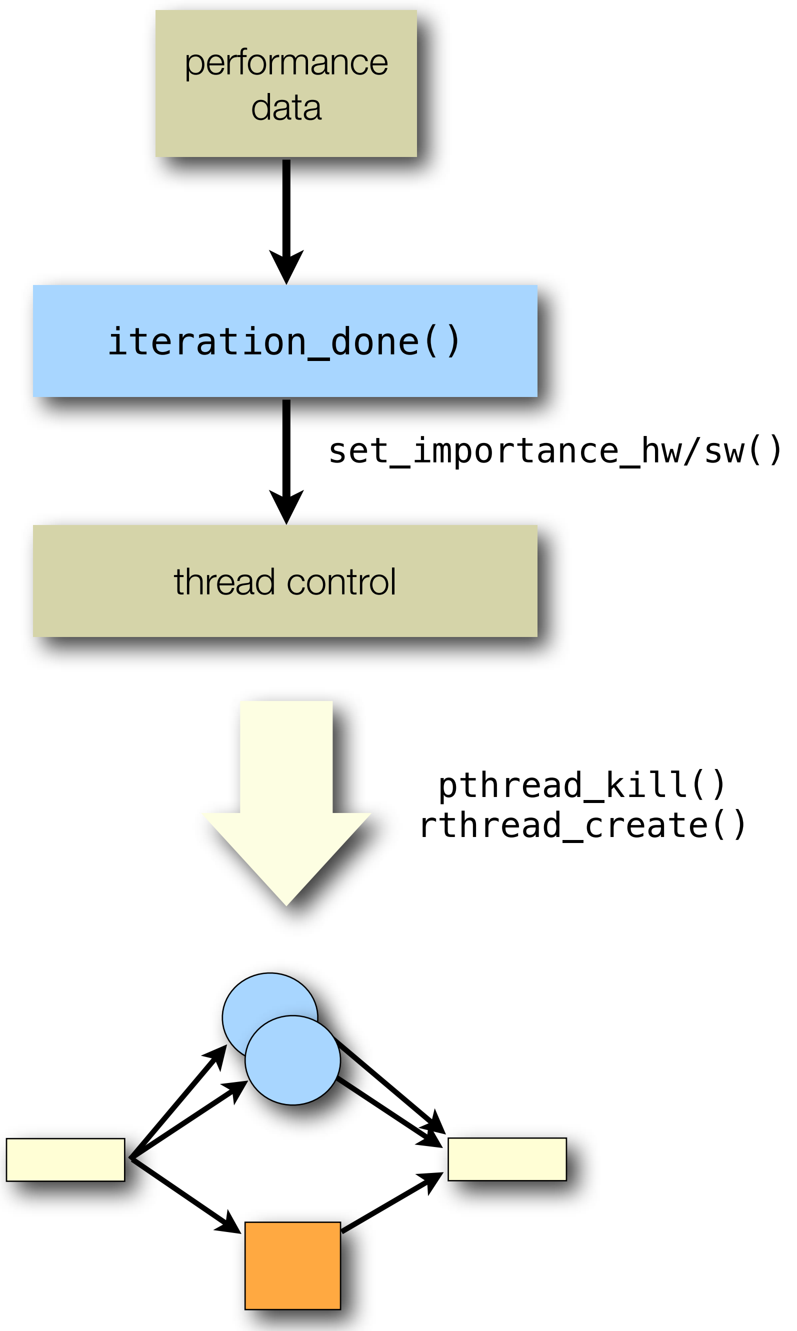
`likelihood()`

System Composition



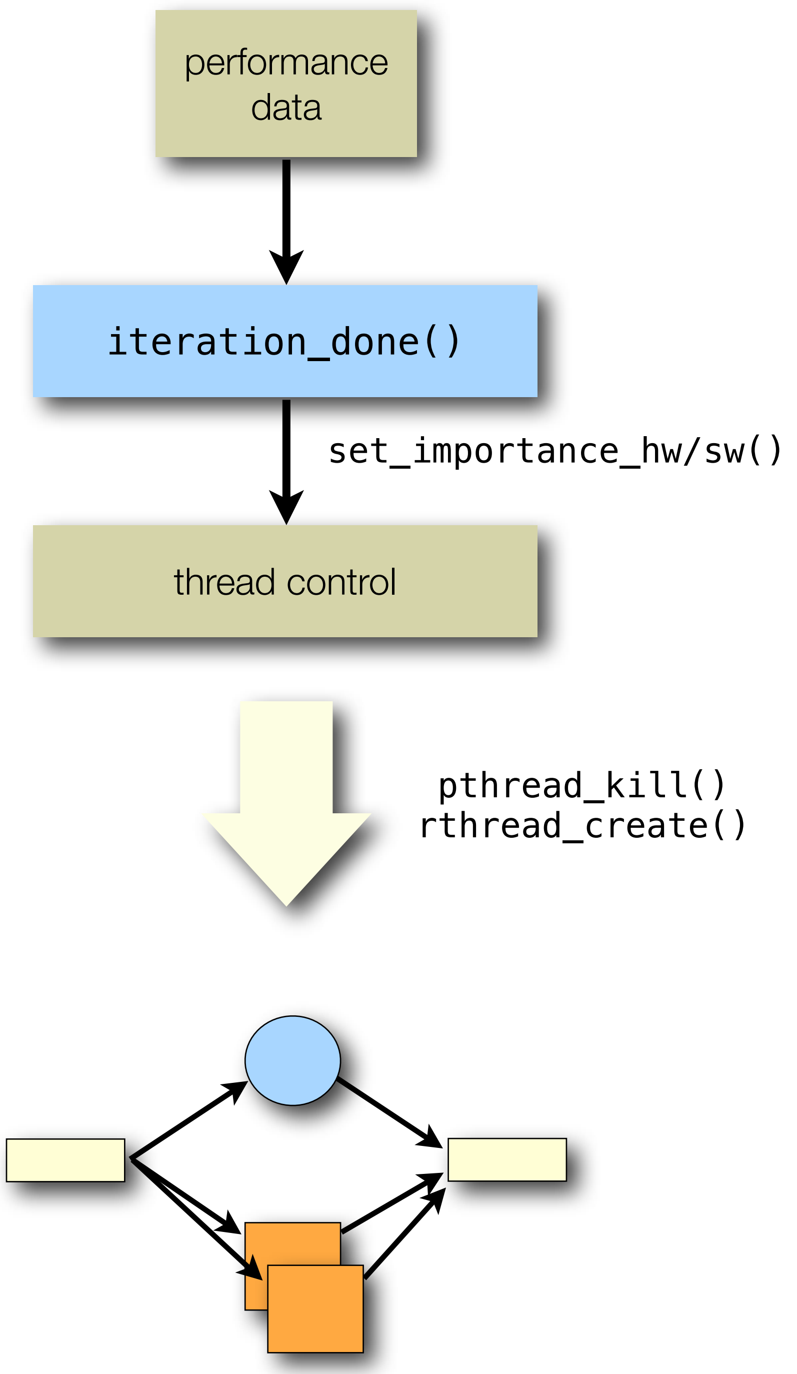
Adaptive HW/SW Partitioning

- dynamic change of a stage's HW/SW thread composition
  - after each iteration, `preResampling()` calls `iteration_done()`
  - based on current performance data (e.g. cycles per iteration), user code decides on new partitioning
  - user code sets new numbers of HW/SW threads for each stage
  - framework transparently terminates/creates threads
  - operating system handles low-level thread management and reconfiguration



Adaptive HW/SW Partitioning

- dynamic change of a stage's HW/SW thread composition
  - after each iteration, `preResampling()` calls `iteration_done()`
  - based on current performance data (e.g. cycles per iteration), user code decides on new partitioning
  - user code sets new numbers of HW/SW threads for each stage
  - framework transparently terminates/creates threads
  - operating system handles low-level thread management and reconfiguration



Application Example

- object tracking in a video sequence
  - particle data / system state: position p, velocity v, scaling factor s
  - system model: $p_t = p_{t-1} + v_{t-1} + \mathcal{N}(0, \sigma^2)$
  - measurement: video frame
  - observation: HSV color histogram $H_i(k),\ k = 0, \ldots, l-1$
  - likelihood: $w_t^i = \exp^{-\left(1 - \sum_{0 \le k < l} \sqrt{(H_i(k) H_R(k))}\right)}$

reference

frame 5

frame 90

frame 150

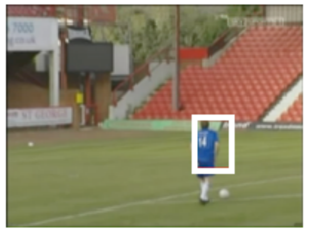
frame 260

Application Example

- performance of individual partitionings
  - sw: all threads run in software
  - hw*: a number of threads run in hardware

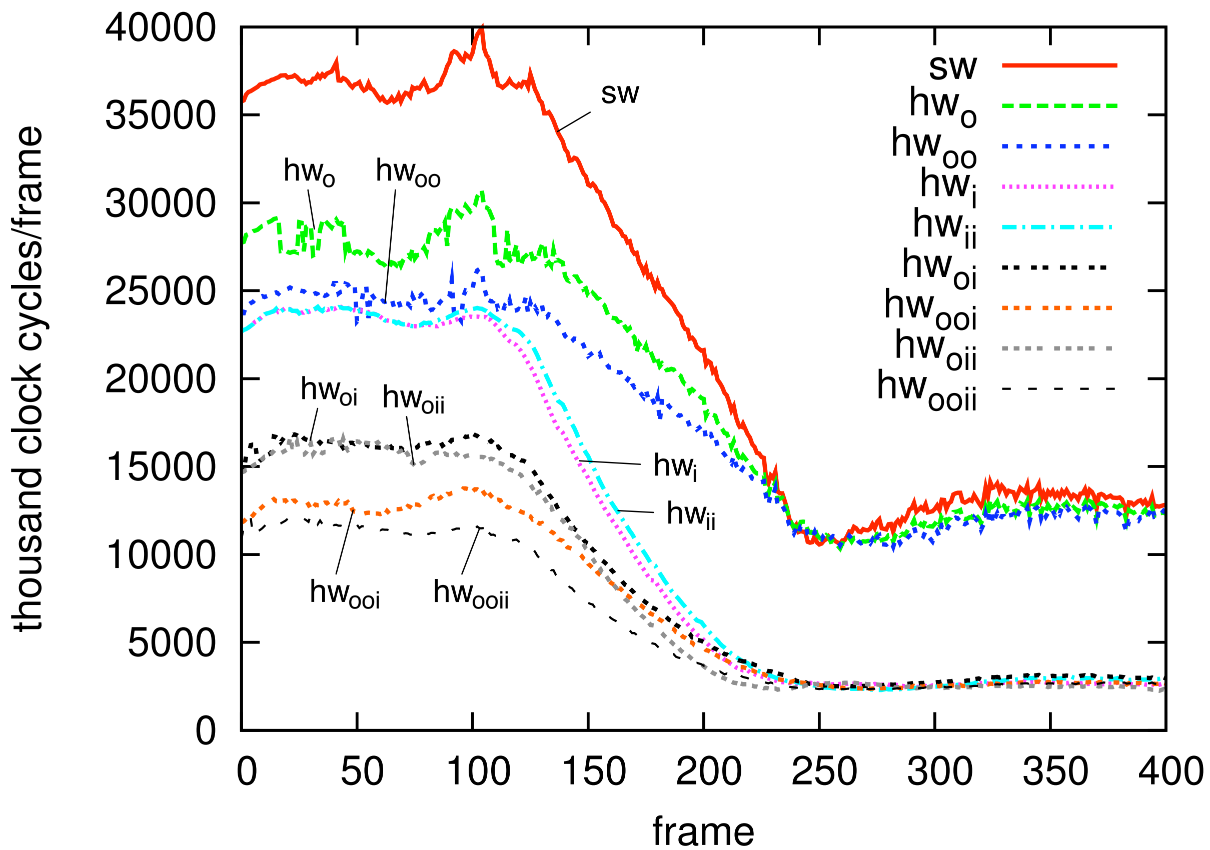
frame 5

frame 90



frame 150

frame 260

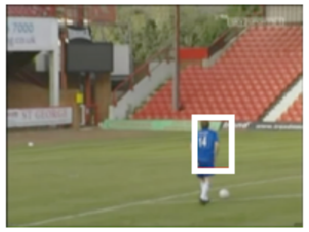


Application Example

- performance of individual partitionings
  - sw: all threads run in software
  - hw*: a number of threads run in hardware
  - adaptive: run-time change between hw_o and sw

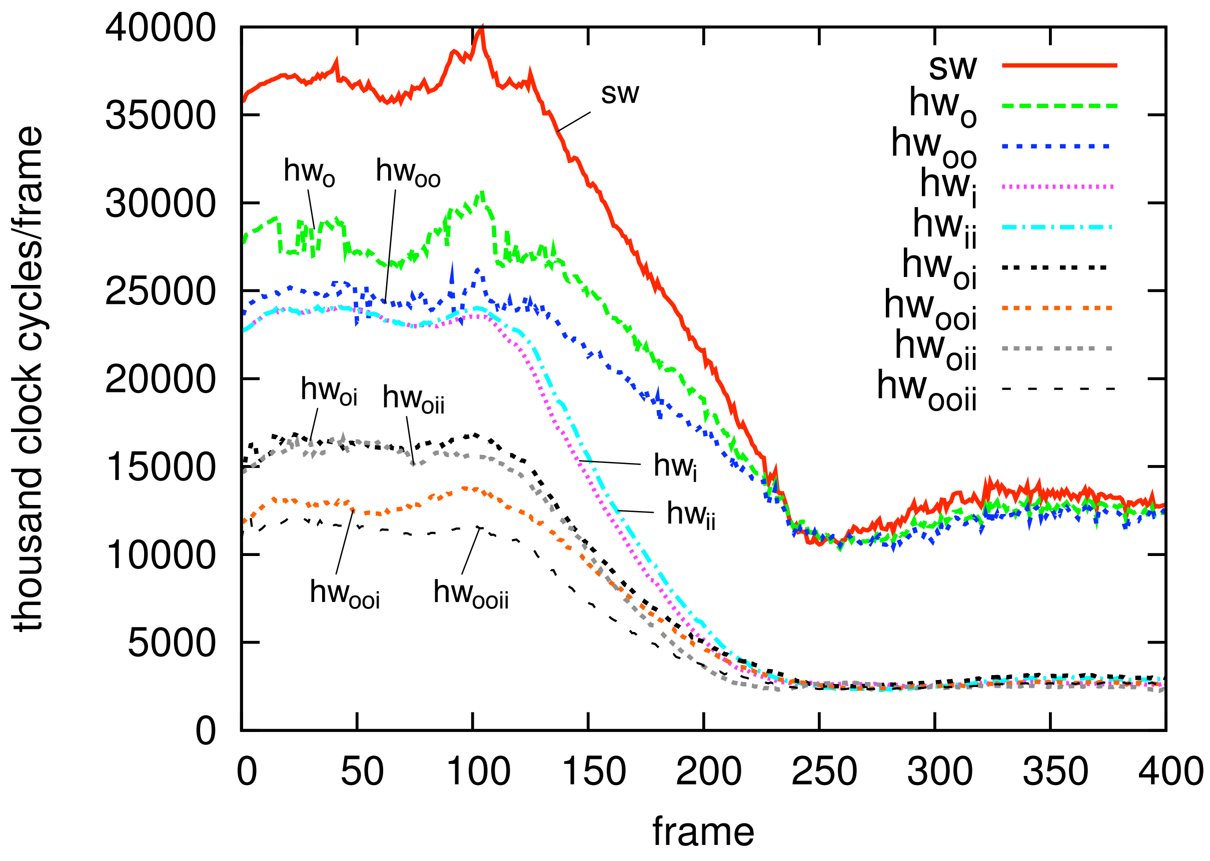
frame 5

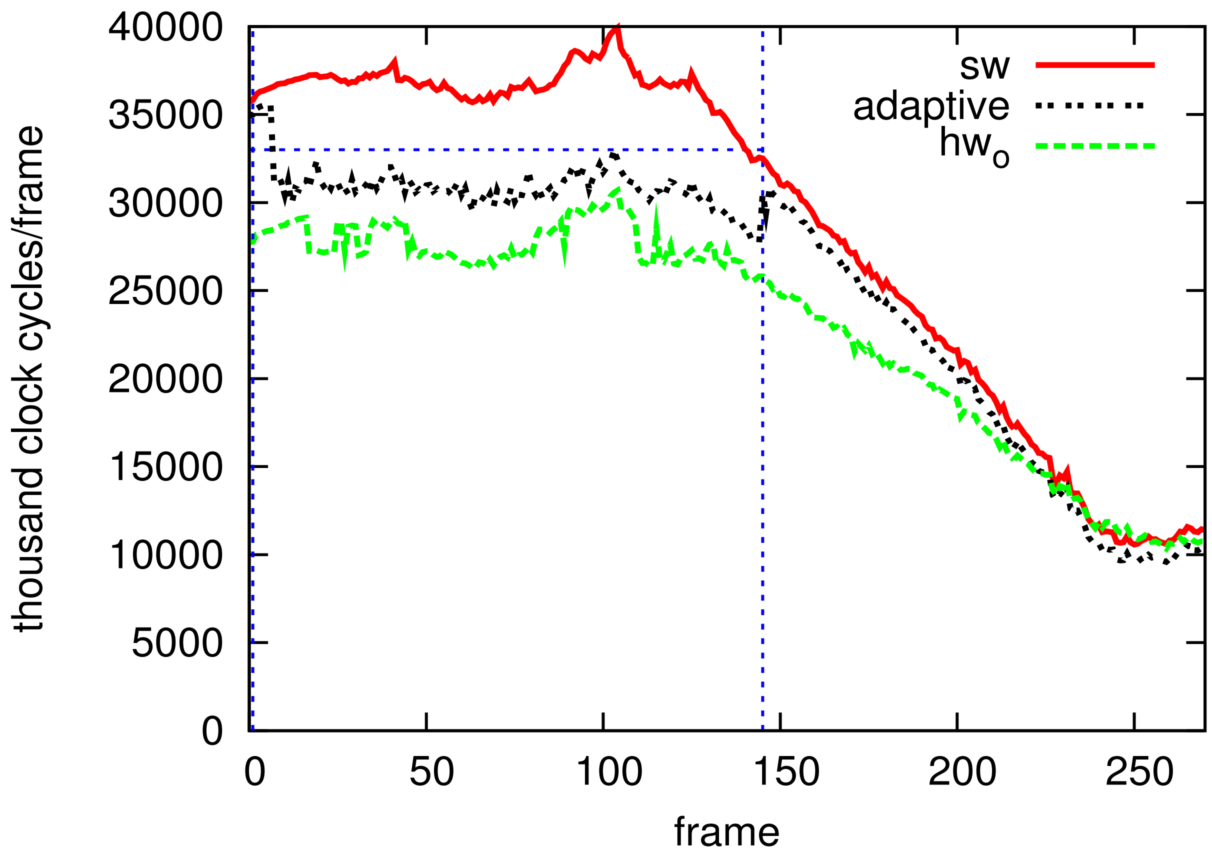
frame 90



frame 150

frame 260





Conclusion

- multithreaded framework for sequential Monte-Carlo methods
  - allows creation of hardware-accelerated **SMC applications** from different application domains, manages recurring SMC-related tasks
  - based on **SIR algorithm** with added observation stage
  - implemented on top of the multithreaded operating system **ReconOS**
  - simplifies creation of prototypes for **HW/SW design space exploration**
  - can exploit data-dependent thread performance through **adaptive repartitioning**
- future work
  - enable a greater degree of run-time reconfigurability (RTR)
  - reduce reconfiguration overhead ➡ increase applicability and feasibility of RTR
  - research into scheduling and migration of hardware threads

- motivation
- multithreaded OS for reconfigurable devices
  - programming model
  - execution model
- sequential Monte Carlo framework
  - sampling-importance-resampling algorithm
  - application modeling
  - runtime adaptation
- experimental results
- conclusion & outlook

Thank you.

info@reconos.de
www.reconos.de