# Example: Continuous Time Random Walk

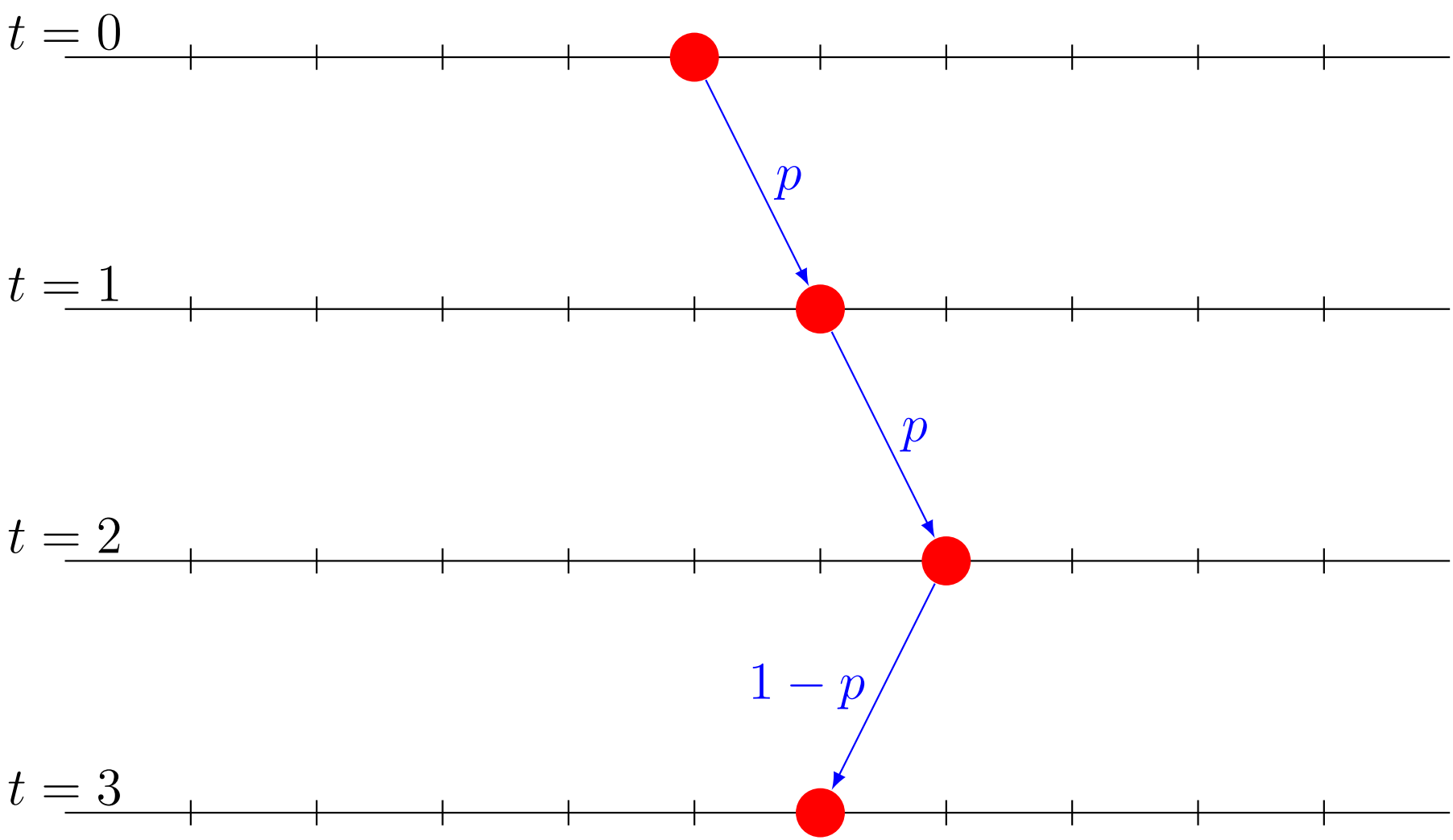
Object Oriented Programming
2022 First Semester
Shin-chi Tadaki (Saga University)

# Today's theme

- Example of asynchronous objects
- How to collect data from those asynchronous objects
- sample programs
  `https://github.com/oop-mc-saga/CTRW`

# Simplest Random Walk

- One dimensional lattice
  - Positions are integers
- Discrete time steps: $t = 0, 1, \cdots$
- A walker starts $x = 0$ at $t = 0$, and move right with $p$ and left with $q = 1 - p$.
- Consider many walkers running independently
  - no collisions between walkers
- Observe how many walkers at given $x$ and $t$.

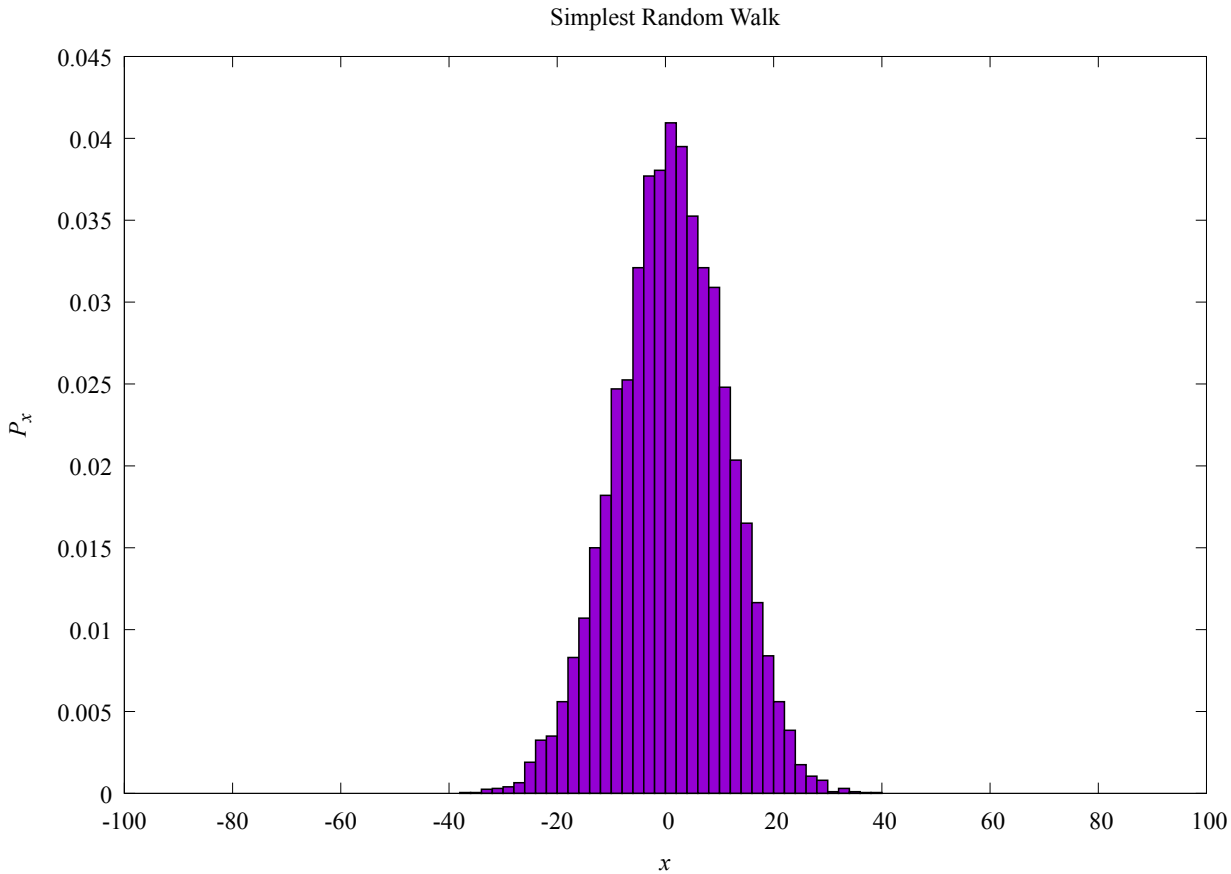

$t = 0$
$t = 1$
$t = 2$
$t = 3$
$p$
$p$
$1 - p$

# Class plan: `simplest` package

- `Walker` class
  - One walker with position $x$ and time $t$
  - `move()` method for moving one step
- `RandomWalk` class
  - Many walkers moving synchronously.
  - `update()` method calls `move()` of all walkers
- `Main` class
  - Has `main()` method only
  - Executes `RandomWalk` class and outputs histogram.

# randomNumberGenerators package

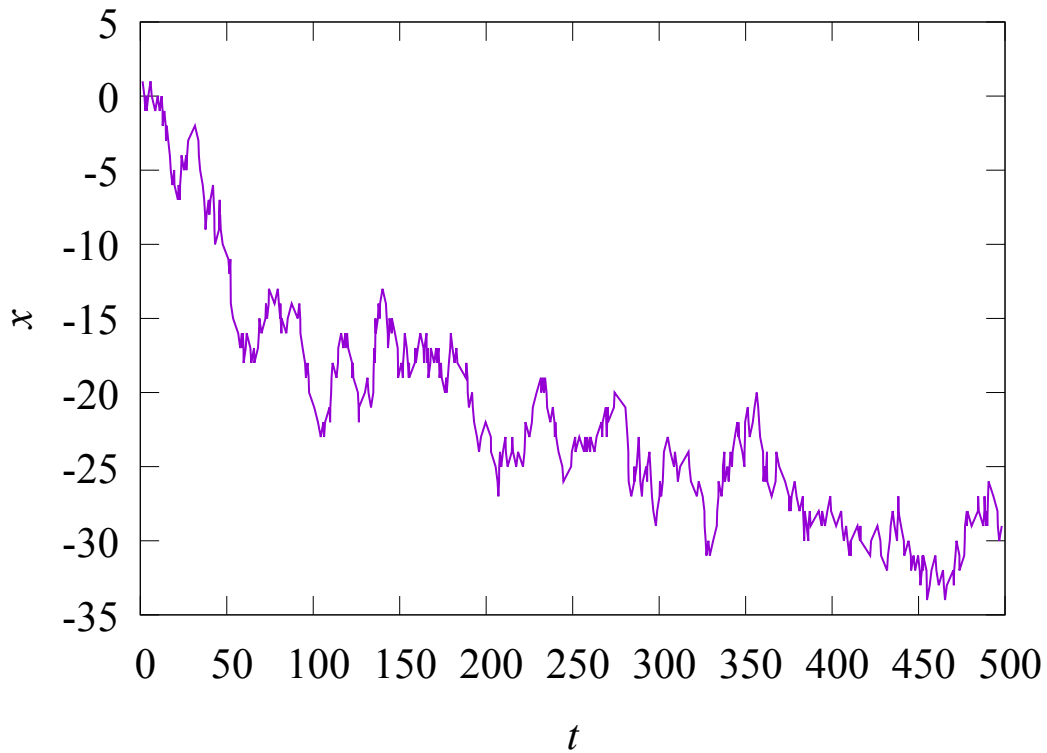
- AbstractRandom class
  - Abstract class for other random number generators
- Transform class
  - Random number generator with transform method.
- Rejection class
  - Random number generator with rejection method
- Uniform class
  - Random number generator of uniform distribution.


Simplest Random Walk
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
$P_x$
-100
-80
-60
-40
-20
0
20
40
60
80
100
$x$

# Continuous time random walk

- Extending the simplest RW.
- Introducing continuous time.
- A walker waits the next move with a probability density $\phi(t)$.
- All walkers move asynchronously.
- Positions are integers.

# History of positions of a walker




$$\phi(t) = e^{-t}$$

# Problems and solutions

- Walkers move asynchronously.
  - Need to run on threads.
  - Specify the maximum duration time $t_{\mathrm{max}}$ (not steps).
- RandomWalk class needs some method to confirm that all walkers stopped.
  - Stopped walker registers itself to some data.
  - Stopped walker notifies a special event.

# Abstract CTRW: Define abstract classes

- Common functions for CTRW realizations
- Independent on realization of confirming walkers stop events.
- `AbstractWalker`
- `AbstractCTRW`
- `AbstractMain`

`abstractCTRW` package

# `AbstractWalker` class

- Implementing `Runnable` for running on a thread
- Running up to $t_{\mathsf{max}}$.
- `stopMotion()` may be override depending on a method for sending stopping event.
- No abstract methods.

```
1  public void run() {
2      while (running) {
3          double dt = waitingTimeRandom.getNext();
4          if (dt + t > maxT) {
5              stopMotion();
6          } else {
7              t += dt;
8              continueMotion();
9          }
10     }
11 }
```

# AbstractCTRW class

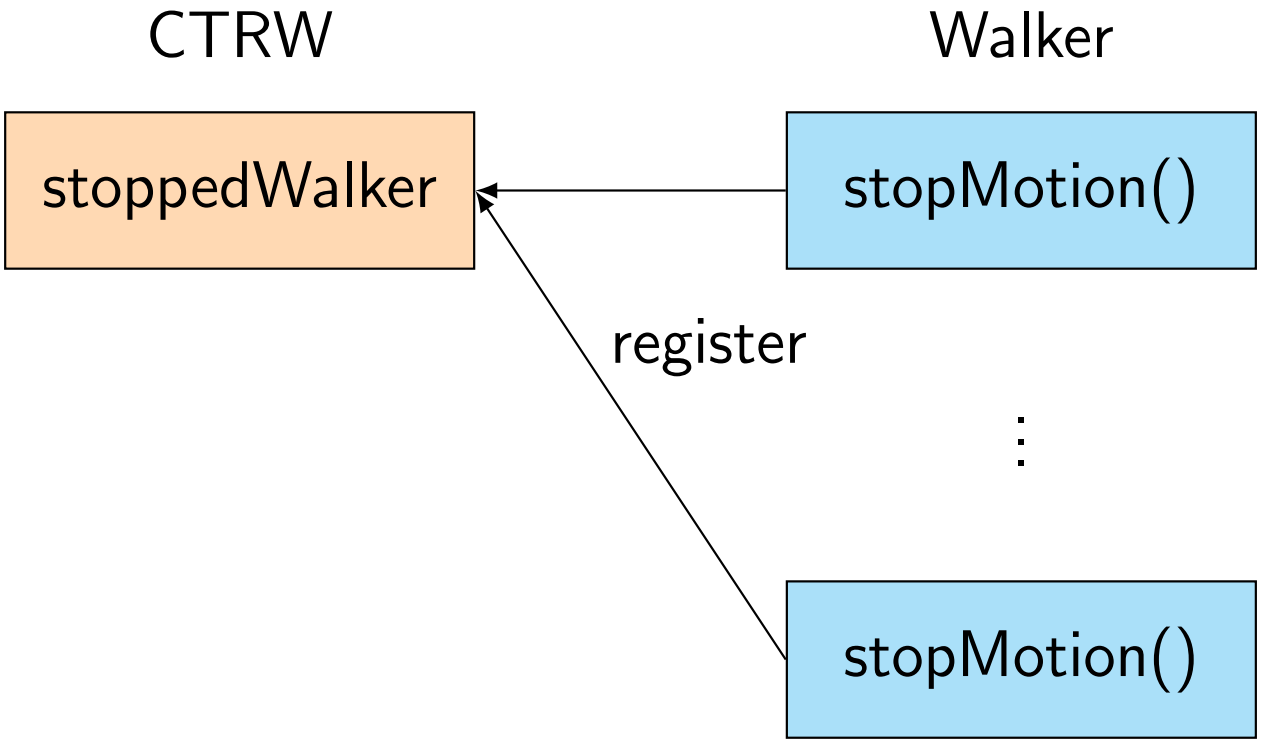
- Stopped walkers are registered to `stopedWalker` set.
- `isFinished()` method confirms that all walkers stopped.

# `AbstractMain` class

- Common static methods for `Main` classes
- Waiting that all walkers stopped.
  - Using `Runnable`
- After confirming all walkers stopped,
  - Output histogram
  - Output motion history of a walker

# First solution: Walkers register itself: ctrw1 package

Each walker registers itself to stoppedWalker set



# Registering itself in Walker class

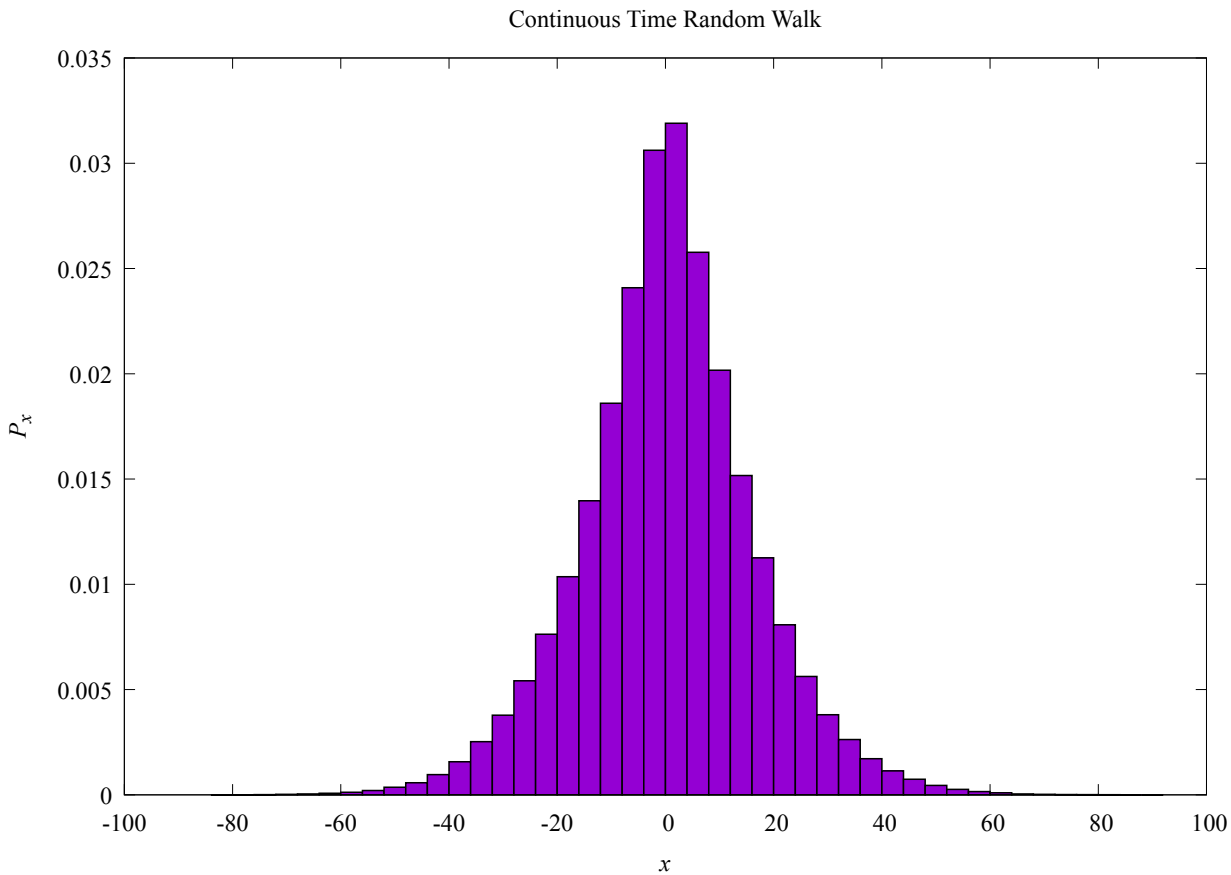
```
1  protected void stopMotion() {
2      super.stopMotion();
3      stoppedWalkers.add(this);
4  }
5
6  public void setStoppedWalkers(Set<AbstractWalker> stoppedWalkers) {
7      this.stoppedWalkers = stoppedWalkers;
8  }
```

# How to protect a set from asynchronous accesses

- Walkers run on separate threads.
- Accessing a set instance may destroy it.
- `Collections.synchronizedSet()` wraps a set instance to be thread-safe

```
1  public void start() {
2      //stoppedWalkers are used in walker instances running on threads
3      //Need protection
4      stoppedWalkers = Collections.synchronizedSet(new HashSet<>());
5      walkers.forEach(
6          w -> ((Walker) w).setStoppedWalkers(stoppedWalkers));
7      super.start_sub();
8  }
```

# Distribution of positions



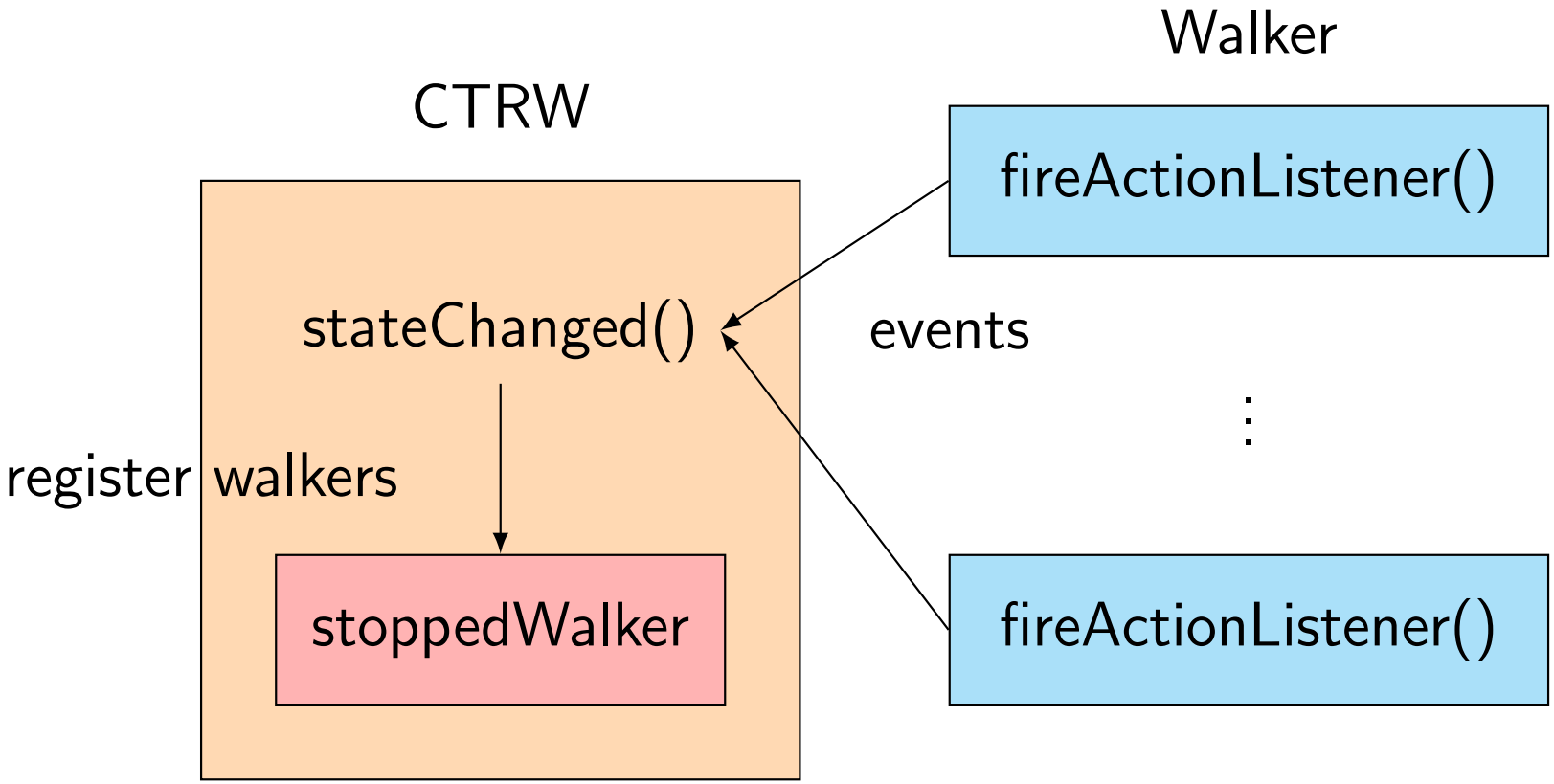

Use `ctrw.plt` gnuplot script

# Second solution: Walkers send events: ctrw2 package

- A walker sends a predefined event to `CTRW`, if stopped.
- When `CTRW` receives a event from a walker, the walker is registered to to `stoppedWalker` set.

# How to work

- Define a customized event class and its listener class.
- `Walker` class
  - Call the listener's `stateChange()` method when stopped
- `CTRW` class
  - Implement the customized listener
  - Register the stopped walker when invoking `stateChange()` method


Walker
CTRW
fireActionListener()
stateChanged()
events
register walkers
stoppedWalker
fireActionListener()

# WalkerEvent

```
1  public class WalkerEvent extends EventObject {
2
3      //Event types
4      public static enum EventType{
5          INITIALIZE,FINISH;
6      }
7      private final EventType eventType;
8
9      public WalkerEvent(AbstractWalker source,EventType eventType) {
10         super(source);
11         this.eventType = eventType;
12     }
13
14     public EventType getEventType() {
15         return eventType;
16     }
17 }
```

# WalkerEventListener

```
1  public interface WalkerEventListener {
2
3  public void stateChanged(WalkerEvent e);
4  }
```

# In Walker class

```
1      protected void stopMotion() {
2          super.stopMotion();
3          fireActionListener();
4      }
5
6      private void fireActionListener() {
7          listener.stateChanged(
8                  new WalkerEvent(this, WalkerEvent.EventType.FINISH));
9      }
```

# In CTRW class

```
1       public synchronized void stateChanged(WalkerEvent e) {
2           stoppedWalkers.add((Walker)e.getSource());
3       }
```