Fast algorithms for Monte Carlo simulation of self-avoiding walks

Nathan Clisby
MASCOS / Department of Mathematics and Statistics
The University of Melbourne

ICMS 2014
Hanyang University, Seoul
August 8, 2014

Outline

- Self-avoiding walks (SAW)

Outline

- Self-avoiding walks (SAW)
- Monte Carlo

Outline

- Self-avoiding walks (SAW)
- Monte Carlo
- Pivot algorithm

Outline

- Self-avoiding walks (SAW)
- Monte Carlo
- Pivot algorithm
- Global moves for local cost via SAW-tree data structure:

Outline

- Self-avoiding walks (SAW)
- Monte Carlo
- Pivot algorithm
- Global moves for local cost via SAW-tree data structure:
  - pivot moves for SAW

Outline

- Self-avoiding walks (SAW)
- Monte Carlo
- Pivot algorithm
- Global moves for local cost via SAW-tree data structure:
  - pivot moves for SAW
  - backbite moves for Hamiltonian paths

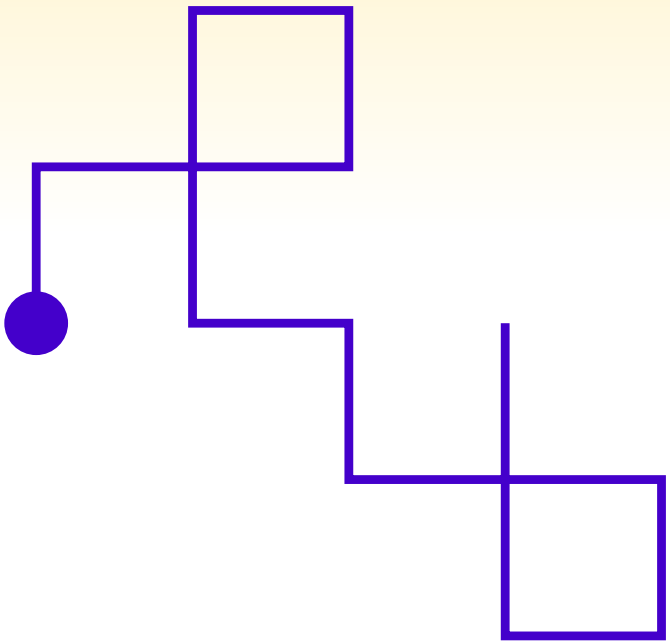
Outline

- Self-avoiding walks (SAW)
- Monte Carlo
- Pivot algorithm
- Global moves for local cost via SAW-tree data structure:
  - pivot moves for SAW
  - backbite moves for Hamiltonian paths
- Conclusion

Self-avoiding walk model (SAW)

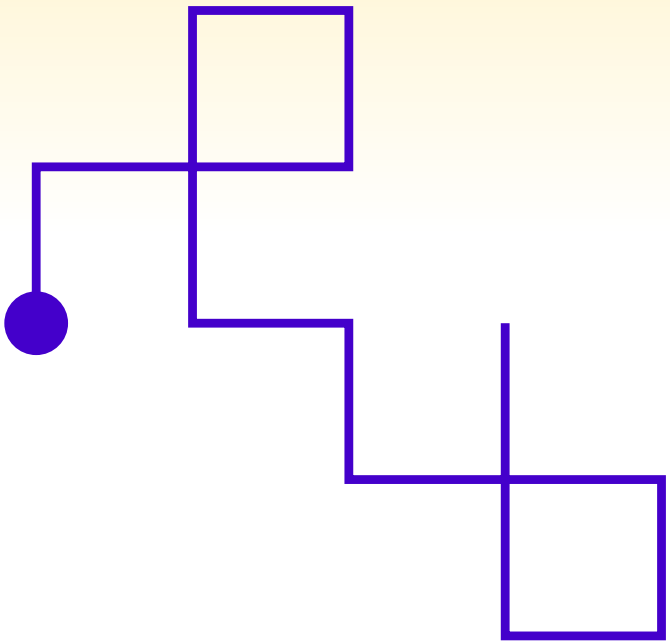
Simple random walk

Self-avoiding walk model (SAW)



Simple random walk

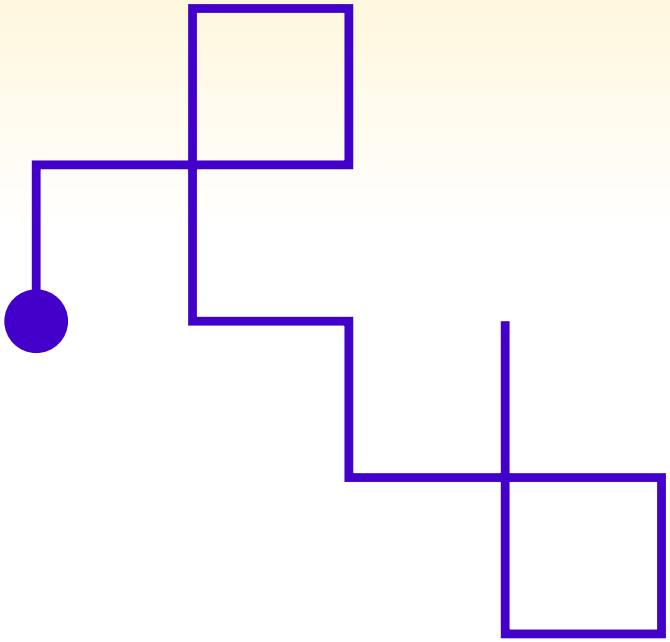
Self-avoiding walk model (SAW)



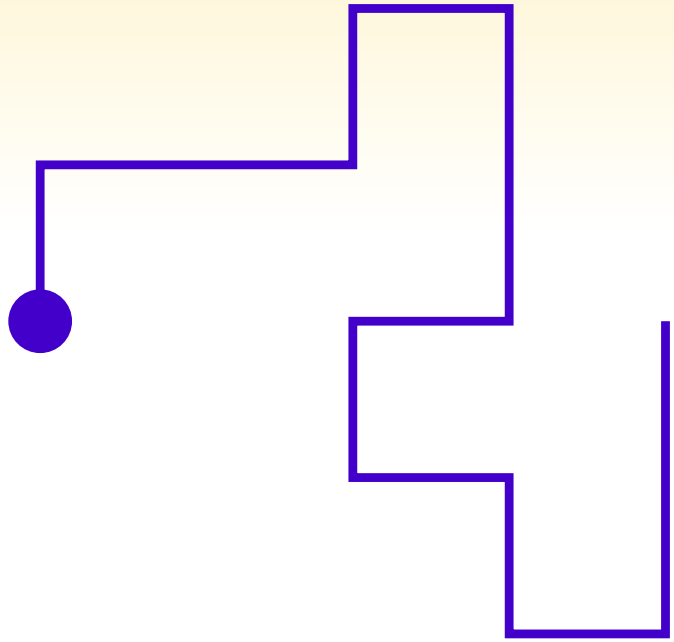
Simple random walk

Self-avoiding walk

Self-avoiding walk model (SAW)

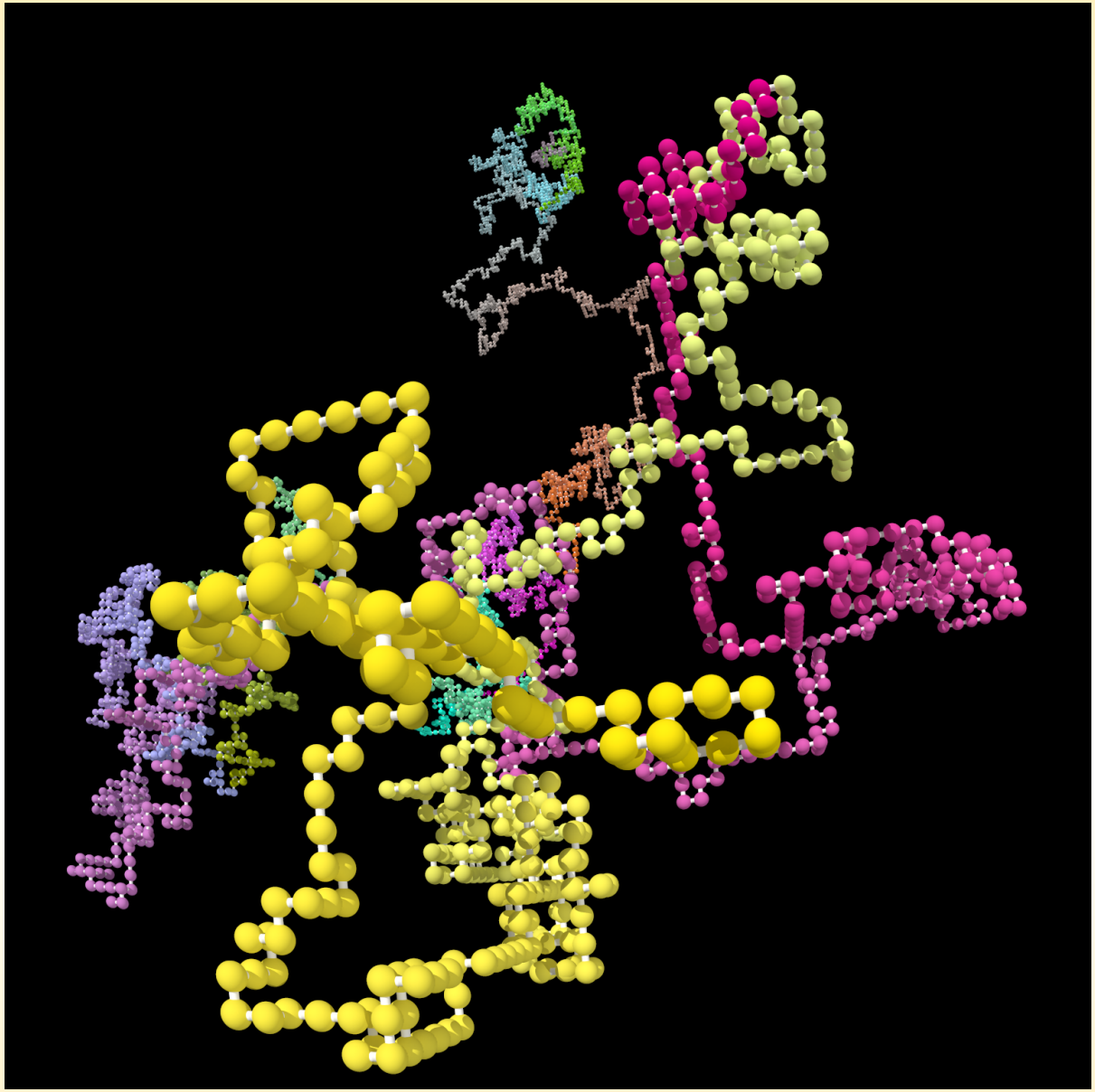


Simple random walk



Self-avoiding walk

A typical SAW of 5000 steps on the simple cubic lattice:



- SAW *exactly* captures universal properties of polymers.

- SAW *exactly* captures universal properties of polymers.
- E.g. typical size of a SAW / polymer grows with the number of monomers, N, as:

$$\mathbf{R} = DN^{\nu}$$

- SAW *exactly* captures universal properties of polymers.
- E.g. typical size of a SAW / polymer grows with the number of monomers, N, as:

$$\mathbf{R} = DN^{\nu}$$

- D is model dependent, but the critical exponent ν is a universal quantity.

- SAW *exactly* captures universal properties of polymers.
- E.g. typical size of a SAW / polymer grows with the number of monomers, N, as:

$$\mathbf{R} = DN^{\nu}$$

- D is model dependent, but the critical exponent ν is a universal quantity.
- No exact solution for SAW.

- SAW *exactly* captures universal properties of polymers.
- E.g. typical size of a SAW / polymer grows with the number of monomers, N, as:

$$\mathbf{R} = DN^{\nu}$$

- D is model dependent, but the critical exponent ν is a universal quantity.
- No exact solution for SAW.
- Markov chain Monte Carlo (MCMC) sampling very powerful for SAW, especially for $d = 3$.

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".
- Local moves:

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".
- Local moves:
  - fast, easy to implement;

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".
- Local moves:
  - fast, easy to implement;
  - move slowly around state space.

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".
- Local moves:
  - fast, easy to implement;
  - move slowly around state space.
- Global moves:

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".
- Local moves:
  - fast, easy to implement;
  - move slowly around state space.
- Global moves:
  - difficult to find one which is not always rejected;

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".
- Local moves:
  - fast, easy to implement;
  - move slowly around state space.
- Global moves:
  - difficult to find one which is not always rejected;
  - slow, harder to implement;

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".
- Local moves:
  - fast, easy to implement;
  - move slowly around state space.
- Global moves:
  - difficult to find one which is not always rejected;
  - slow, harder to implement;
  - move rapidly around state space.

Markov chain Monte Carlo (MCMC)

- Estimate physical properties of system by sampling from all possible configurations (state space).
- Basic idea: generate new configurations by deforming current configuration via a "move".
- Local moves:
  - fast, easy to implement;
  - move slowly around state space.
- Global moves:
  - difficult to find one which is not always rejected;
  - slow, harder to implement;
  - move rapidly around state space.

MCMC estimation of π

Key ideas for radically efficient MCMC algorithm:

- large deformations which are "allowed" by the system with fair probability.

Key ideas for radically efficient MCMC algorithm:

- large deformations which are "allowed" by the system with fair probability.
- efficient implementation which implements global move for local cost.

Pivot algorithm

- Local moves, such as adding or removing a monomer, require $O(N^2)$ moves to get an "essentially new" configuration.

Pivot algorithm

- Local moves, such as adding or removing a monomer, require $O(N^2)$ moves to get an "essentially new" configuration.
- Global moves can do much better.

Pivot algorithm

- Local moves, such as adding or removing a monomer, require $O(N^2)$ moves to get an "essentially new" configuration.
- Global moves can do much better.
- Pivot move: $O(1)$ successful moves for an essentially new configuration.

Pivot algorithm

- Local moves, such as adding or removing a monomer, require $O(N^2)$ moves to get an "essentially new" configuration.
- Global moves can do much better.
- Pivot move: $O(1)$ successful moves for an essentially new configuration.
- Lal (1969) invented pivot algorithm, but key paper is by Madras and Sokal (1988).

Pivot algorithm

- Procedure:

Pivot algorithm

- Procedure:
  - Choose a pivot site at random

Pivot algorithm

- Procedure:
  - Choose a pivot site at random
  - Then rotate or reflect one of the two parts of the walk.

Pivot algorithm

- Procedure:
  - Choose a pivot site at random
  - Then rotate or reflect one of the two parts of the walk.
  - Retain new walk if it is self-avoiding (success!), otherwise restore original walk.

Pivot algorithm

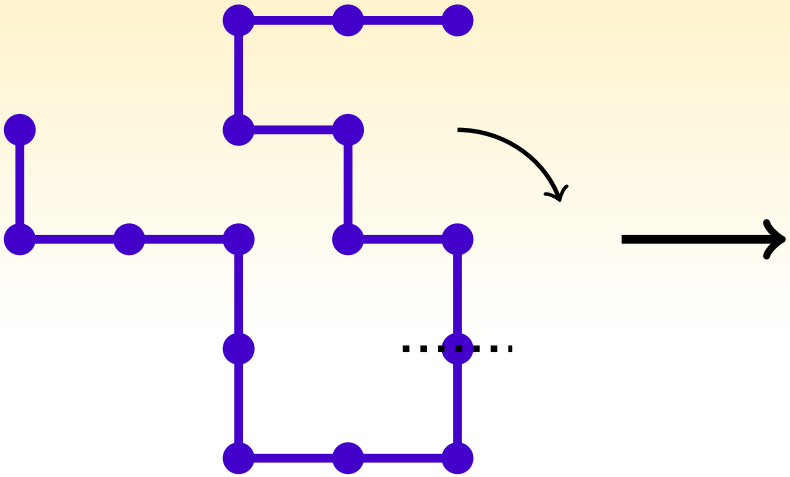
- Procedure:
  - Choose a pivot site at random
  - Then rotate or reflect one of the two parts of the walk.
  - Retain new walk if it is self-avoiding (success!), otherwise restore original walk.
- Often successful because individual parts necessarily remain self-avoiding (local structure preserved) – only need to check that no new intersections are introduced between the parts.

Pivot algorithm

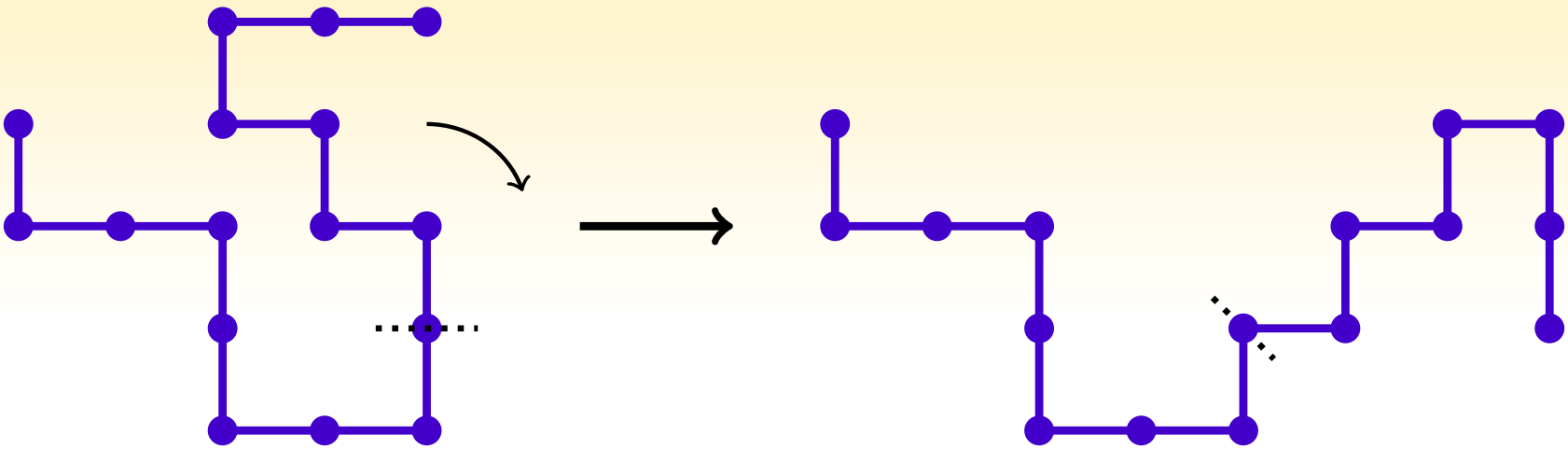
- Procedure:
  - Choose a pivot site at random
  - Then rotate or reflect one of the two parts of the walk.
  - Retain new walk if it is self-avoiding (success!), otherwise restore original walk.
- Often successful because individual parts necessarily remain self-avoiding (local structure preserved) – only need to check that no new intersections are introduced between the parts.
- "Global" because on average half of the monomers are moved.

Pivot algorithm

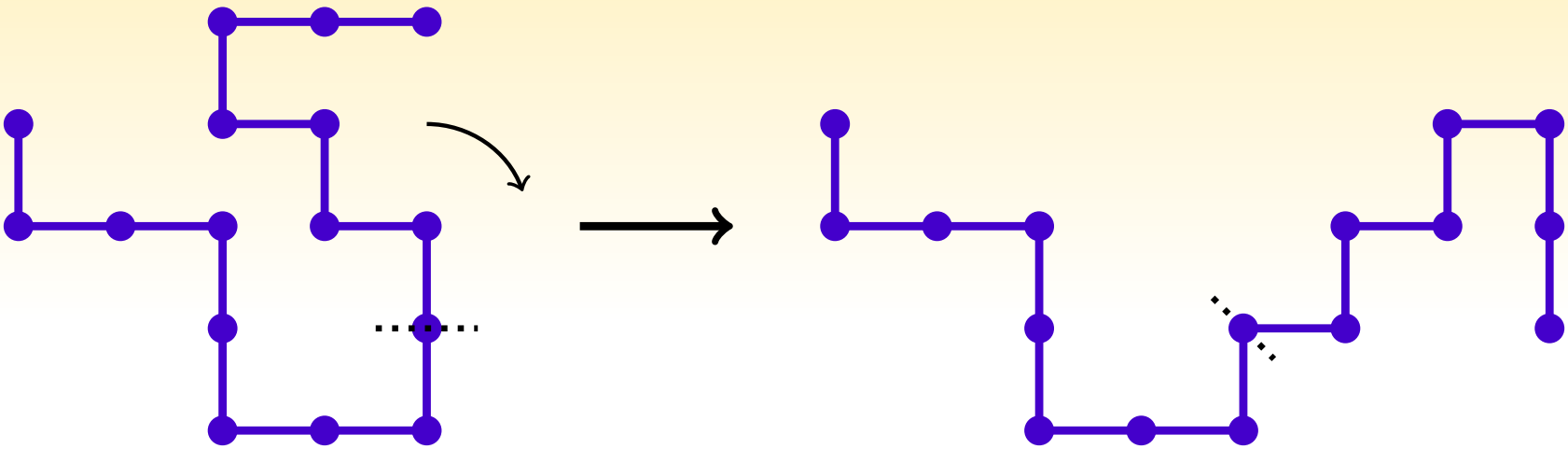
- Procedure:
  - Choose a pivot site at random
  - Then rotate or reflect one of the two parts of the walk.
  - Retain new walk if it is self-avoiding (success!), otherwise restore original walk.
- Often successful because individual parts necessarily remain self-avoiding (local structure preserved) – only need to check that no new intersections are introduced between the parts.
- "Global" because on average half of the monomers are moved.
- Ergodic, samples SAWs uniformly at random.



Example pivot move



Example pivot move



Example pivot move

Pivot algorithm simulation

- Time $O(N)$ to write down an N-step walk, so this must be best possible for pivot move?

- Time $O(N)$ to write down an N-step walk, so this must be best possible for pivot move?
- In fact, don't need to write down! $\Rightarrow$ $o(N)$ (Kennedy, 2002)

- Time $O(N)$ to write down an N-step walk, so this must be best possible for pivot move?
- In fact, don't need to write down! $\Rightarrow o(N)$ (Kennedy, 2002)
- Just need to verify that walk is self-avoiding after a pivot move, and be able to answer queries about global properties.

- Time $O(N)$ to write down an N-step walk, so this must be best possible for pivot move?
- In fact, don't need to write down! $\Rightarrow o(N)$ (Kennedy, 2002)
- Just need to verify that walk is self-avoiding after a pivot move, and be able to answer queries about global properties.
- Bookkeeping can be handled efficiently in binary tree structure. $\Rightarrow O(\log N)$ (C., 2010)

Key properties of the SAW-tree data structure.

- Each node contains:

Key properties of the SAW-tree data structure.

- Each node contains:
  - Symmetry information (for rotations / reflections);

Key properties of the SAW-tree data structure.

- Each node contains:
  - Symmetry information (for rotations / reflections);
  - "Bounding box" information, aka bounding volume hierarchy (for intersection testing);

Key properties of the SAW-tree data structure.

- Each node contains:
  - Symmetry information (for rotations / reflections);
  - "Bounding box" information, aka bounding volume hierarchy (for intersection testing);
  - Information about observables.

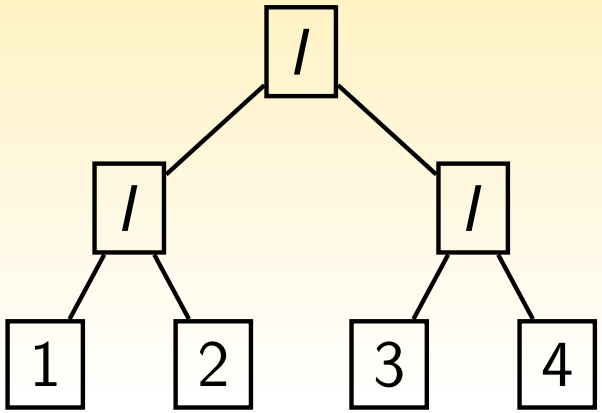
Key properties of the SAW-tree data structure.

- Each node contains:
  - Symmetry information (for rotations / reflections);
  - "Bounding box" information, aka bounding volume hierarchy (for intersection testing);
  - Information about observables.
- Tree structure can be altered via "tree rotations", so that symmetry operations can be applied to any section of the walk.

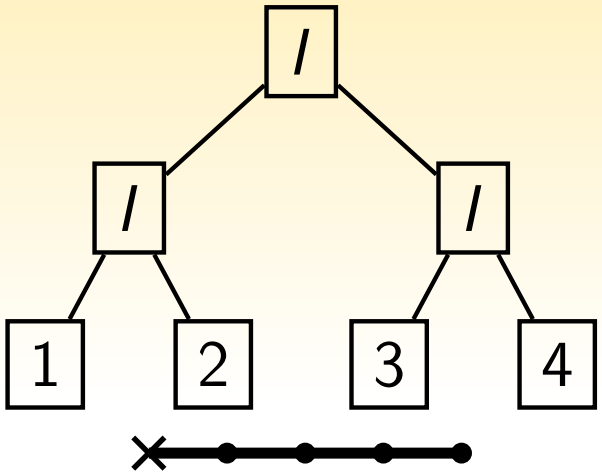
Key properties of the SAW-tree data structure.

- Each node contains:
  - Symmetry information (for rotations / reflections);
  - "Bounding box" information, aka bounding volume hierarchy (for intersection testing);
  - Information about observables.
- Tree structure can be altered via "tree rotations", so that symmetry operations can be applied to any section of the walk.
- Binary tree has typical height $O(\log N)$.

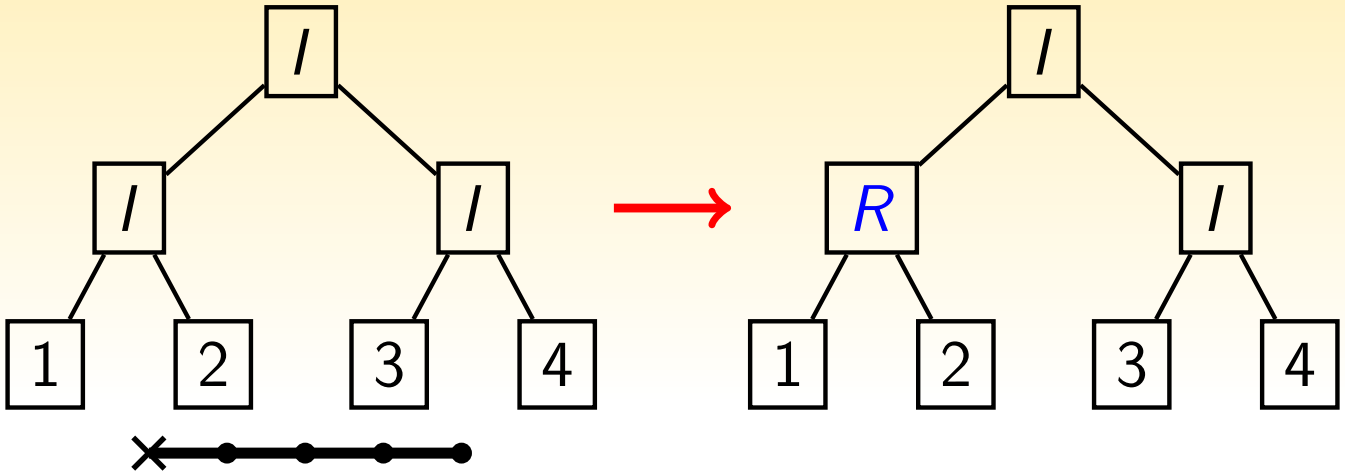
Example SAW-tree moves.



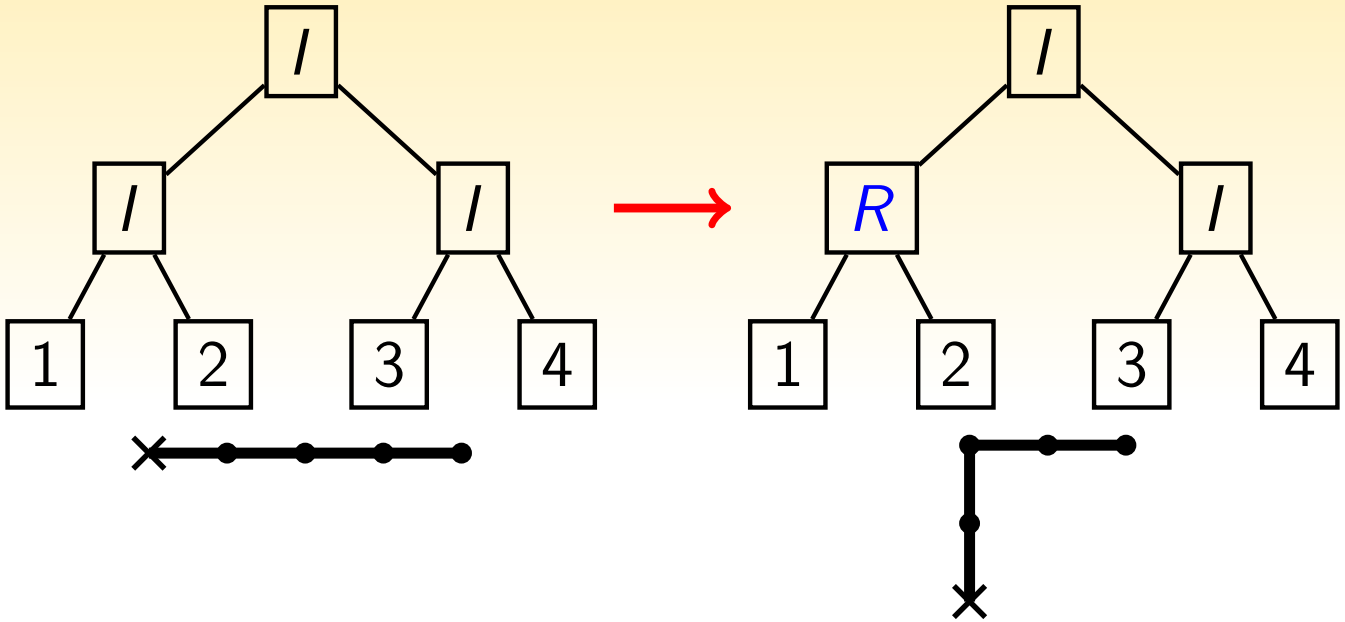
Example SAW-tree moves.



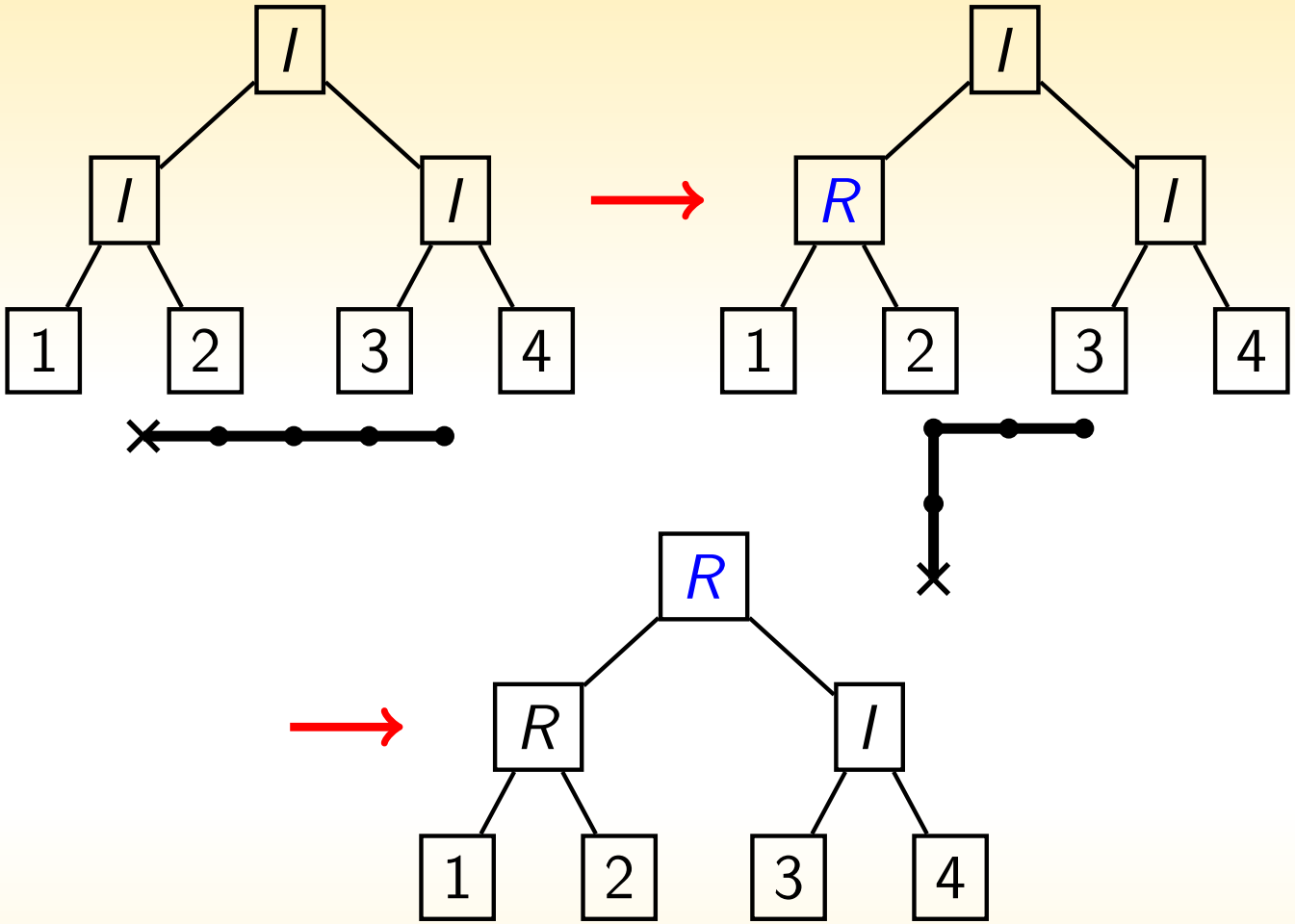
Example SAW-tree moves.



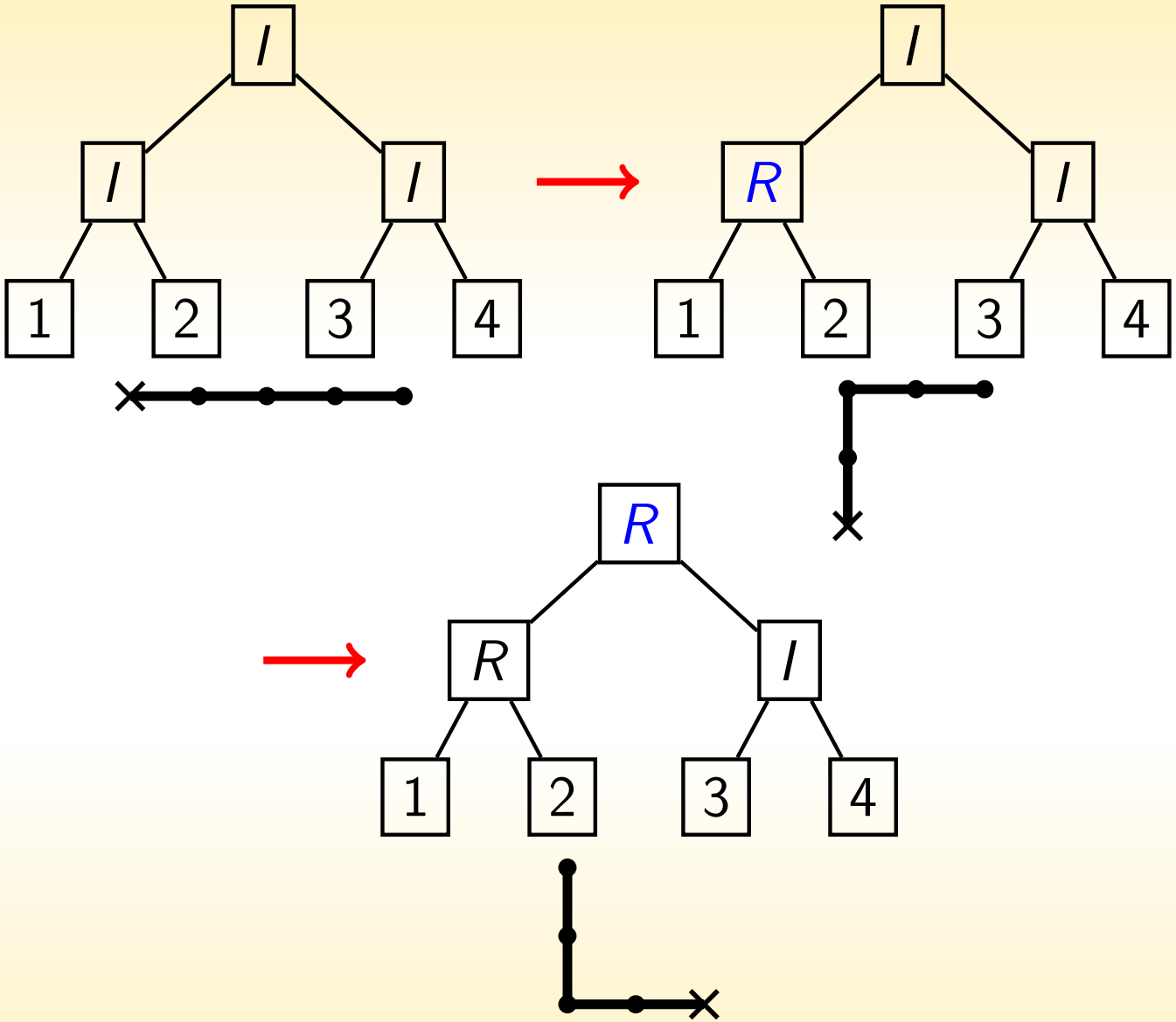
Example SAW-tree moves.



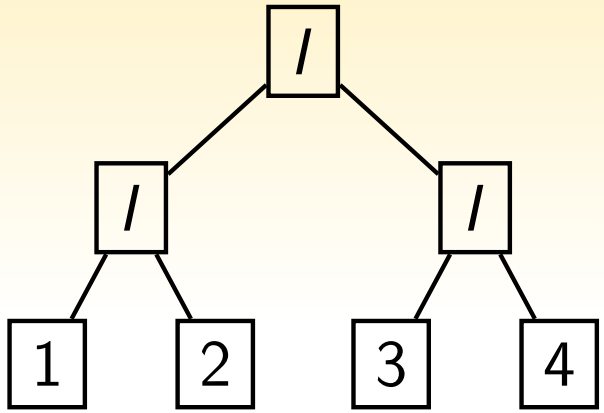
Example SAW-tree moves.



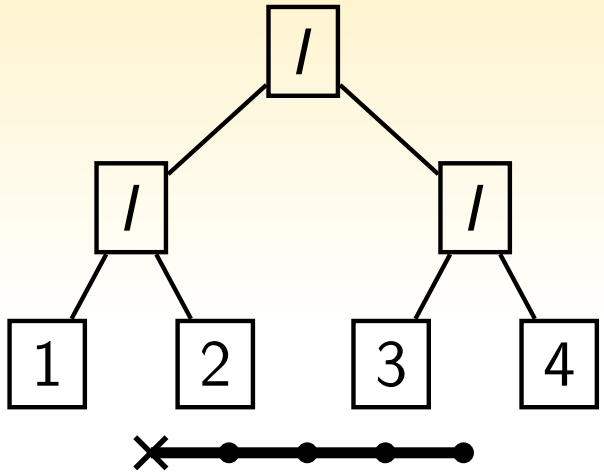
Example SAW-tree moves.



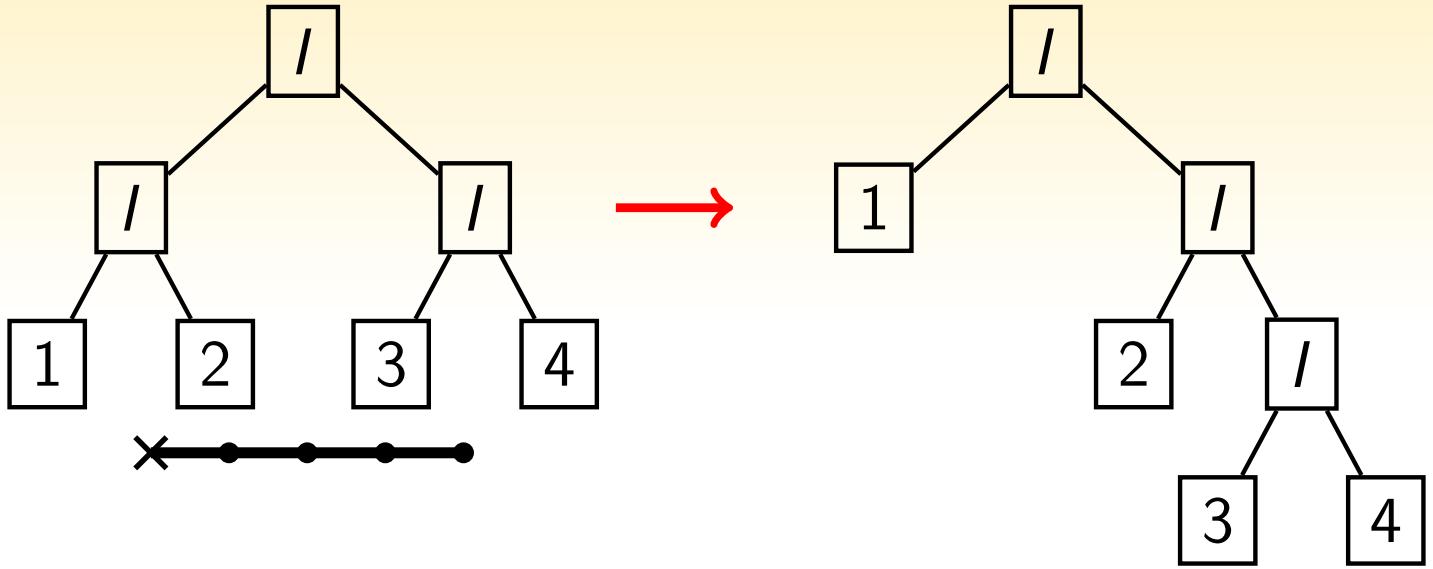
How do we apply symmetry to steps 2,3,4? Restructure the binary tree via a "tree rotation".



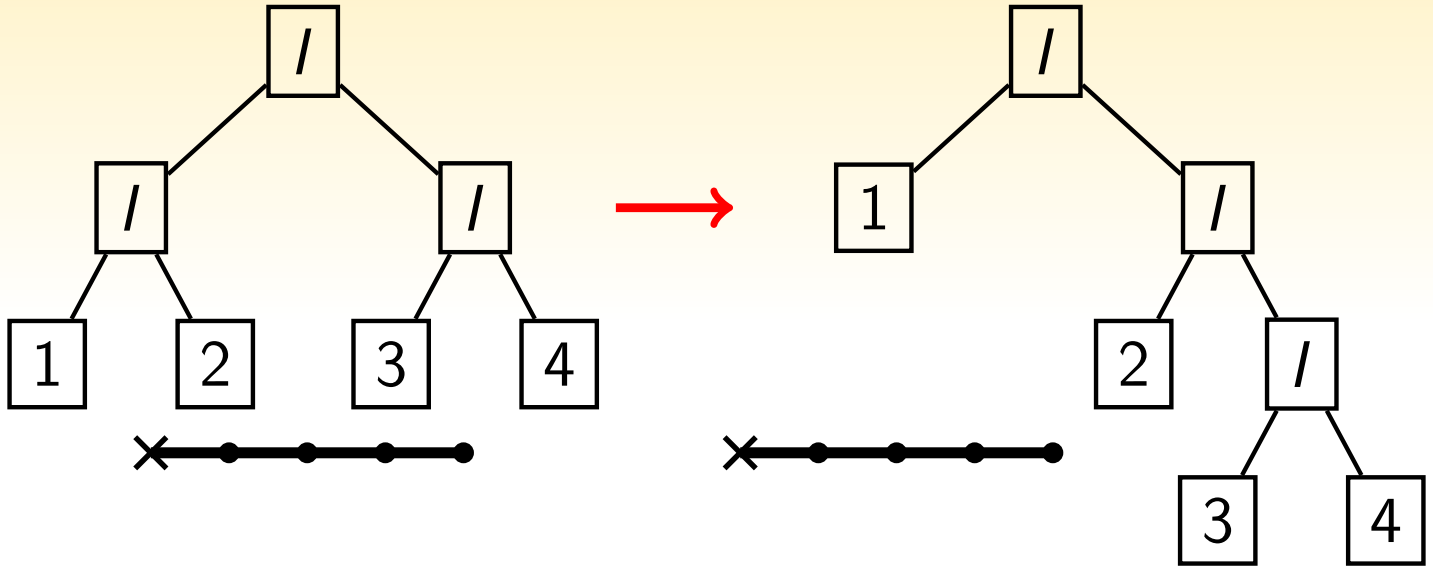
How do we apply symmetry to steps 2,3,4? Restructure the binary tree via a "tree rotation".



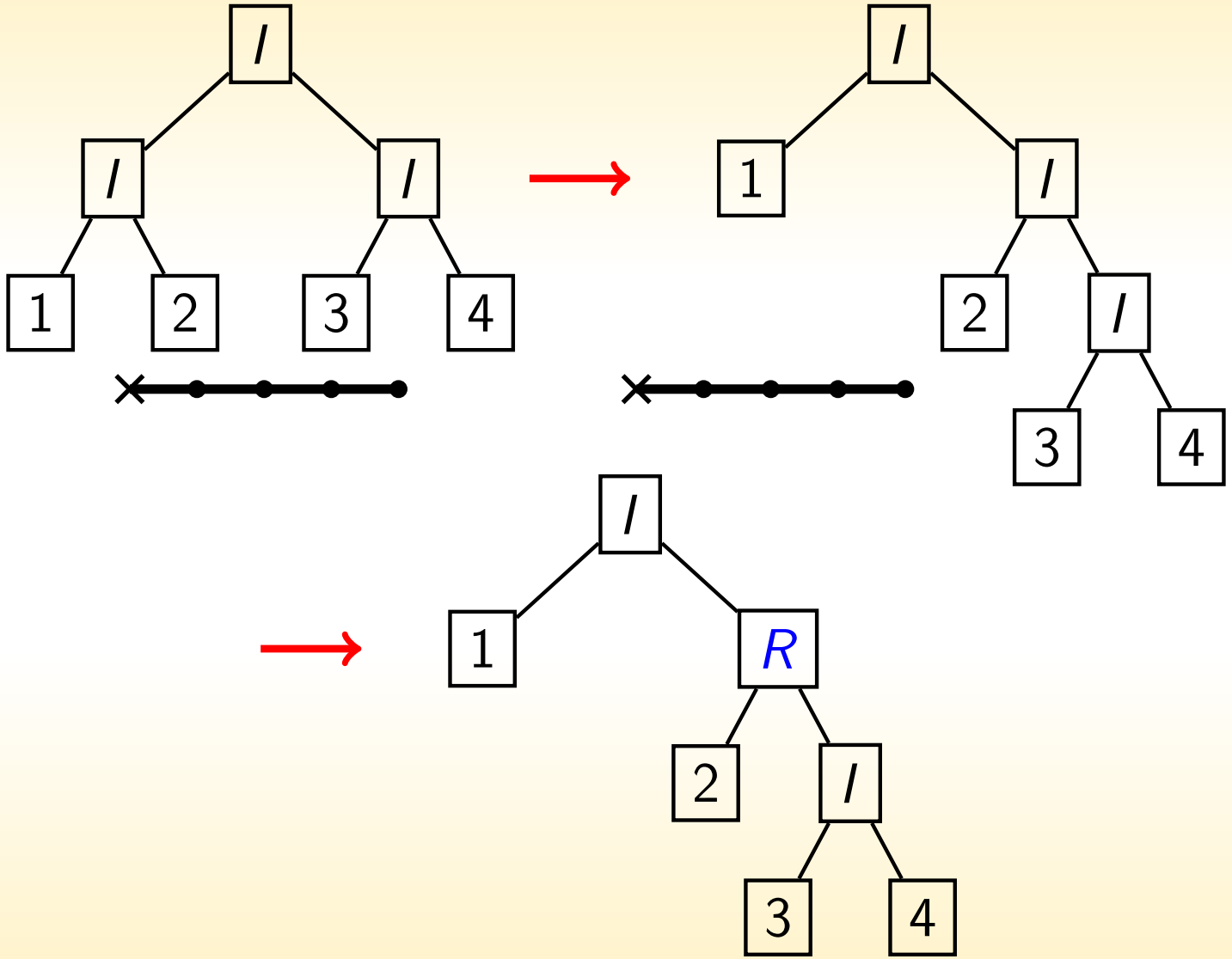
How do we apply symmetry to steps 2,3,4? Restructure the binary tree via a "tree rotation".



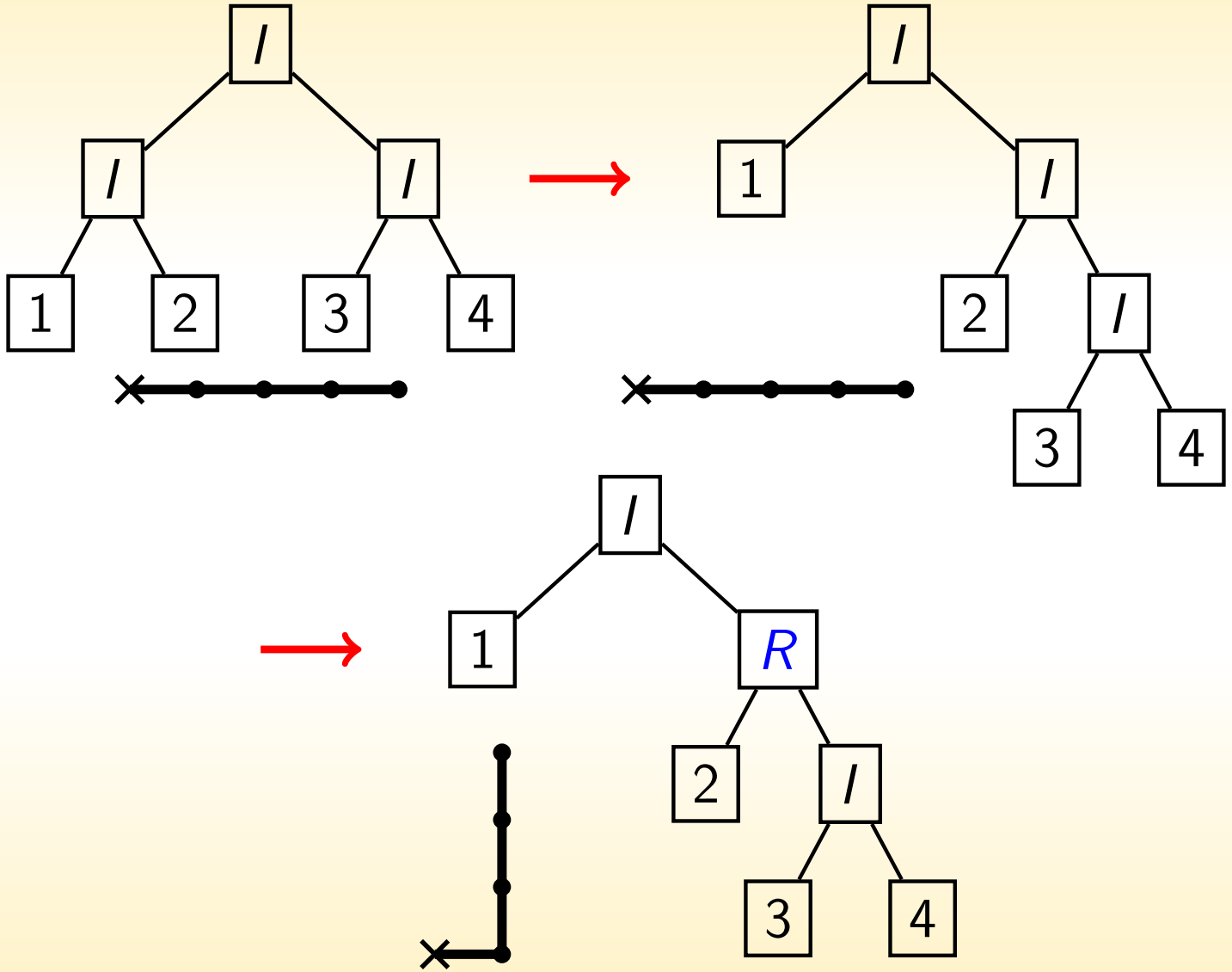
How do we apply symmetry to steps 2,3,4? Restructure the binary tree via a "tree rotation".



How do we apply symmetry to steps 2,3,4? Restructure the binary tree via a "tree rotation".



How do we apply symmetry to steps 2,3,4? Restructure the binary tree via a "tree rotation".



- With a binary tree implementation, pivot move has *global effect* for *local cost*.

- With a binary tree implementation, pivot move has *global effect* for *local cost*.
- $O(\log N)$ for:

- With a binary tree implementation, pivot move has *global effect* for *local cost*.
- $O(\log N)$ for:
  - Rotating part of the walk.

- With a binary tree implementation, pivot move has *global effect* for *local cost*.
- $O(\log N)$ for:
  - Rotating part of the walk.
  - Checking for self-intersections between two pieces.

- With a binary tree implementation, pivot move has *global effect* for *local cost*.
- $O(\log N)$ for:
  - Rotating part of the walk.
  - Checking for self-intersections between two pieces.
  - Calculating global observables such as R_{e}^2 and R_{g}^2.

- With a binary tree implementation, pivot move has *global effect* for *local cost*.
- $O(\log N)$ for:
  - Rotating part of the walk.
  - Checking for self-intersections between two pieces.
  - Calculating global observables such as R_{e}^2 and R_{g}^2.
- Very fast, can rapidly simulate SAW with many millions of steps.

- With a binary tree implementation, pivot move has *global effect* for *local cost*.
- $O(\log N)$ for:
  - Rotating part of the walk.
  - Checking for self-intersections between two pieces.
  - Calculating global observables such as R_{e}^2 and R_{g}^2.
- Very fast, can rapidly simulate SAW with many millions of steps.
- How can we extend this key idea - fast global moves - to other models?

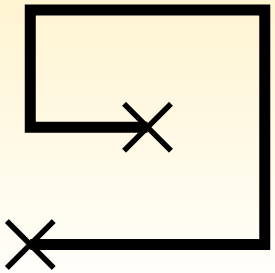
Hamiltonian paths

- Hamiltonian paths are self-avoiding walks which visit every site in a graph.

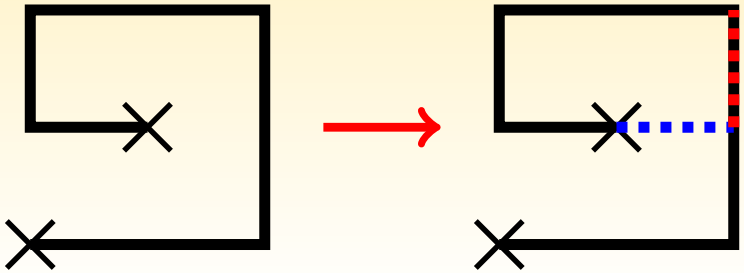
Hamiltonian paths

- Hamiltonian paths are self-avoiding walks which visit every site in a graph.
- Hamiltonian path generator, see
  `http://lattice.complex.unimelb.edu.au/hamiltonian_path`

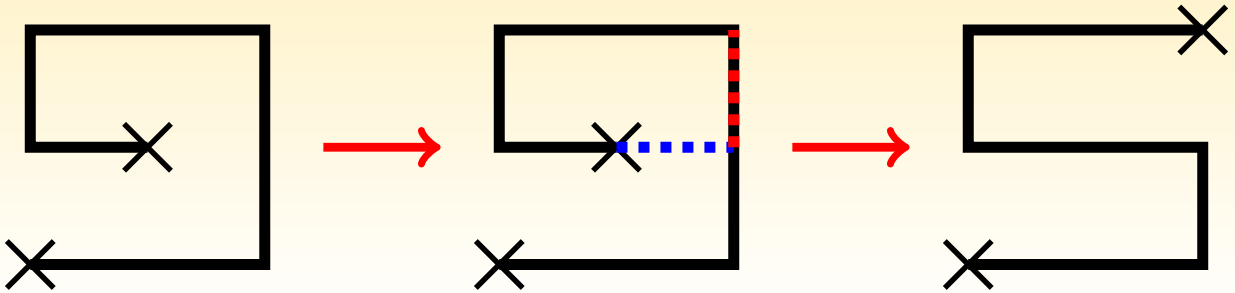
Backbite moves for sampling Hamiltonian paths.



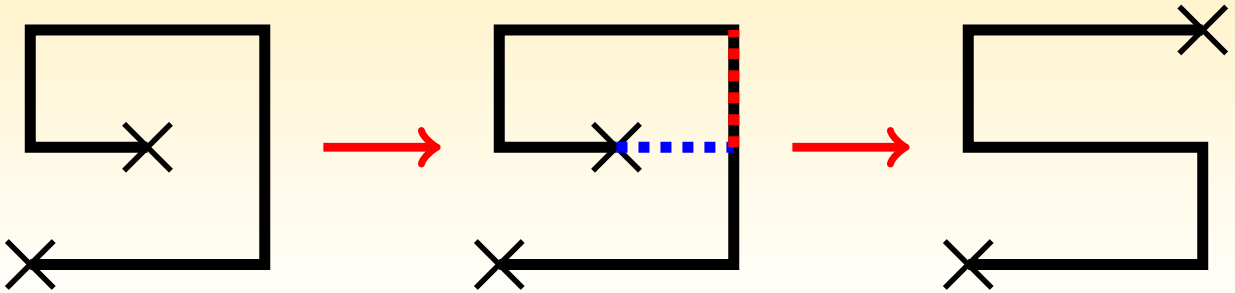
Backbite moves for sampling Hamiltonian paths.

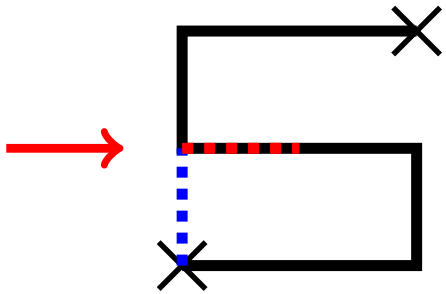


Backbite moves for sampling Hamiltonian paths.

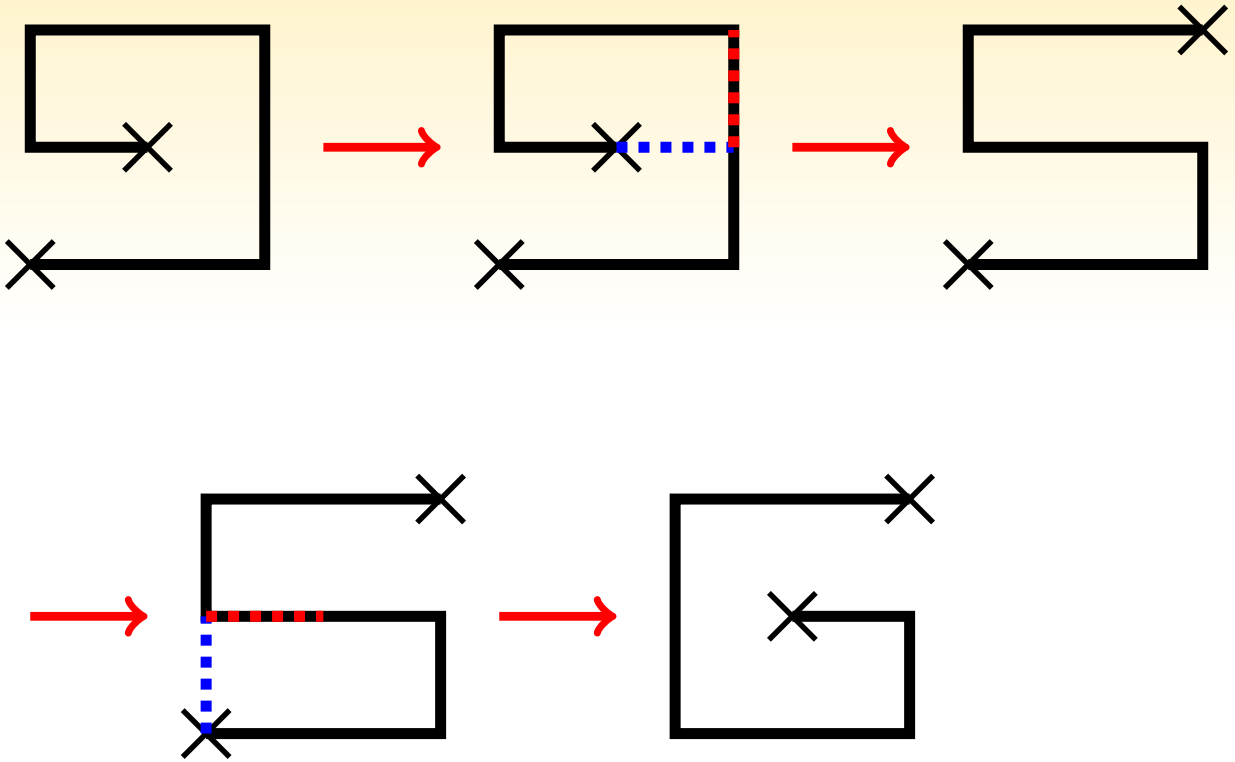


Backbite moves for sampling Hamiltonian paths.

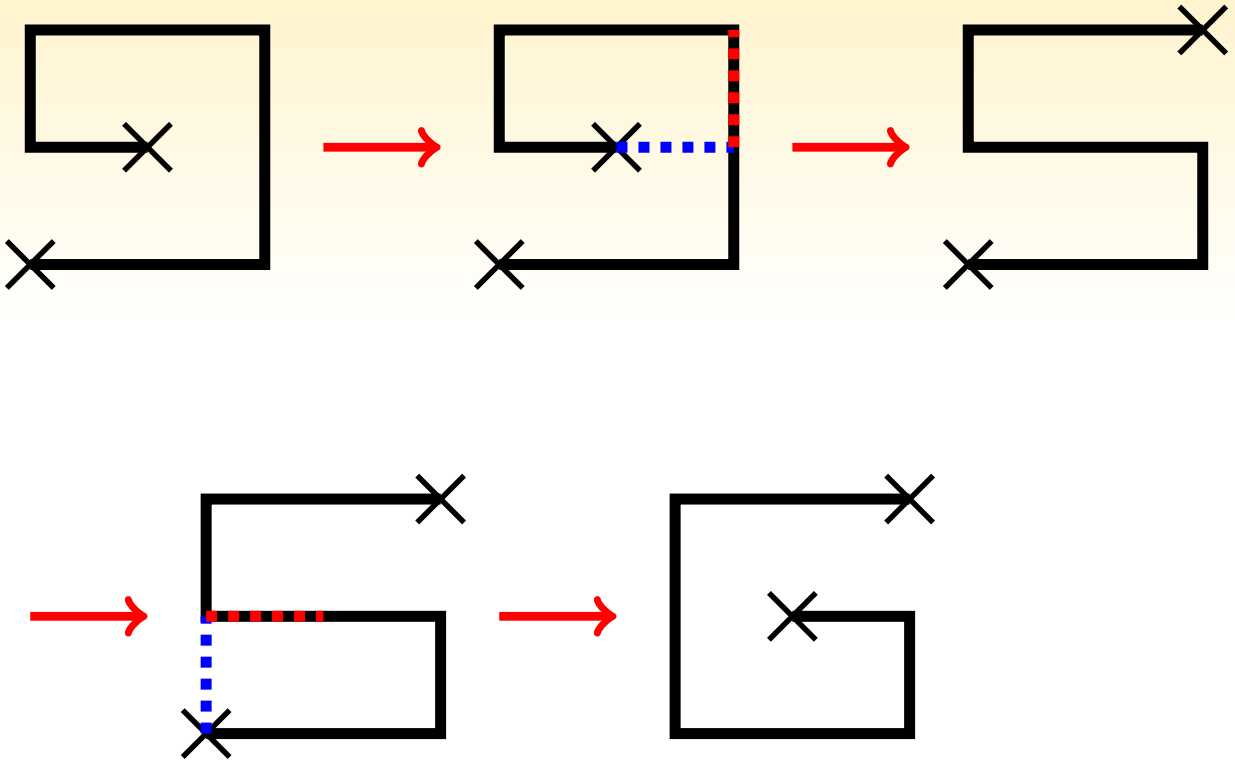




Backbite moves for sampling Hamiltonian paths.



Backbite moves for sampling Hamiltonian paths.



Each time we make a backbite move we create a loop, delete the edge which completes the loop, and *reverse* the orientation of the remaining edges of the loop.

Backbite move

- For the simple cubic lattice, loops are of mean size $O(N)$.

Backbite move

- For the simple cubic lattice, loops are of mean size $O(N)$.
- This suggests time $O(N)$ to perform a backbite move.

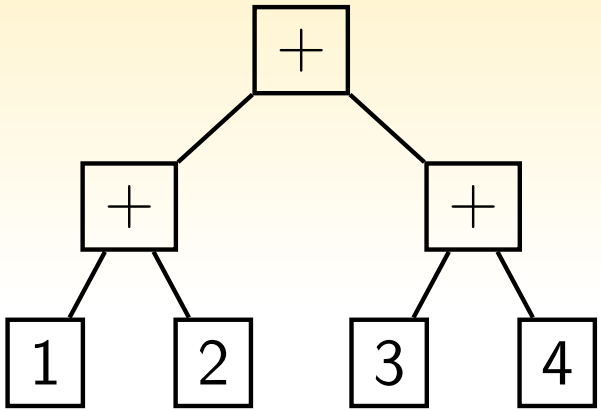
Backbite move

- For the simple cubic lattice, loops are of mean size $O(N)$.
- This suggests time $O(N)$ to perform a backbite move.
- But, again we don't need to explicitly "write down" each step of the walk, just need to be able to store information about structure, and find neighbours of walk ends.

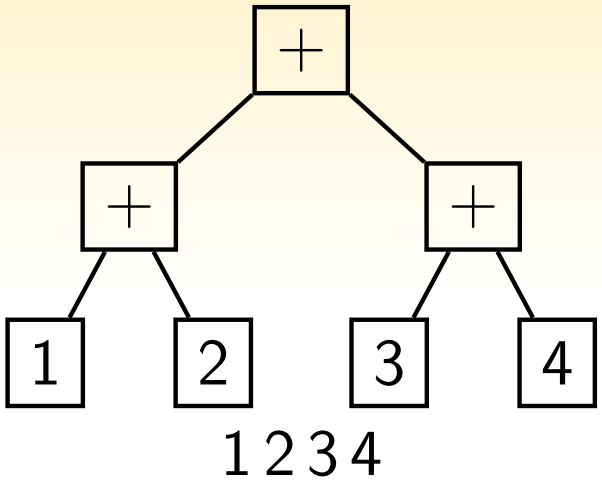
Backbite move

- For the simple cubic lattice, loops are of mean size $O(N)$.
- This suggests time $O(N)$ to perform a backbite move.
- But, again we don't need to explicitly "write down" each step of the walk, just need to be able to store information about structure, and find neighbours of walk ends.
- Use binary tree data structure with *time reversal* as our symmetry operation, with bookkeeping for determining neighbours.

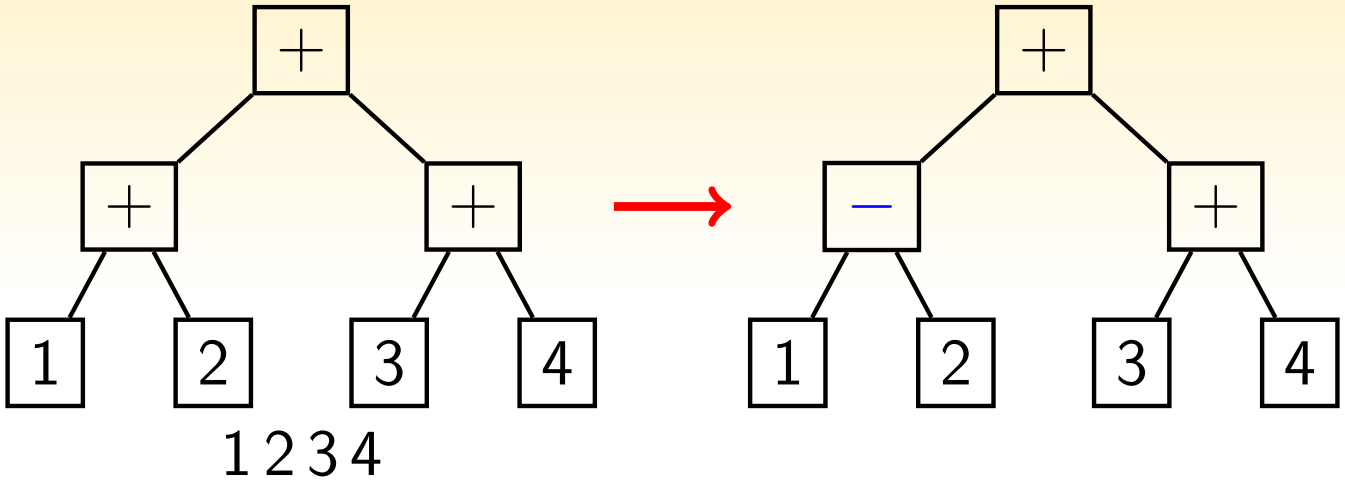
This time binary tree has time reversal symmetry elements in the nodes.



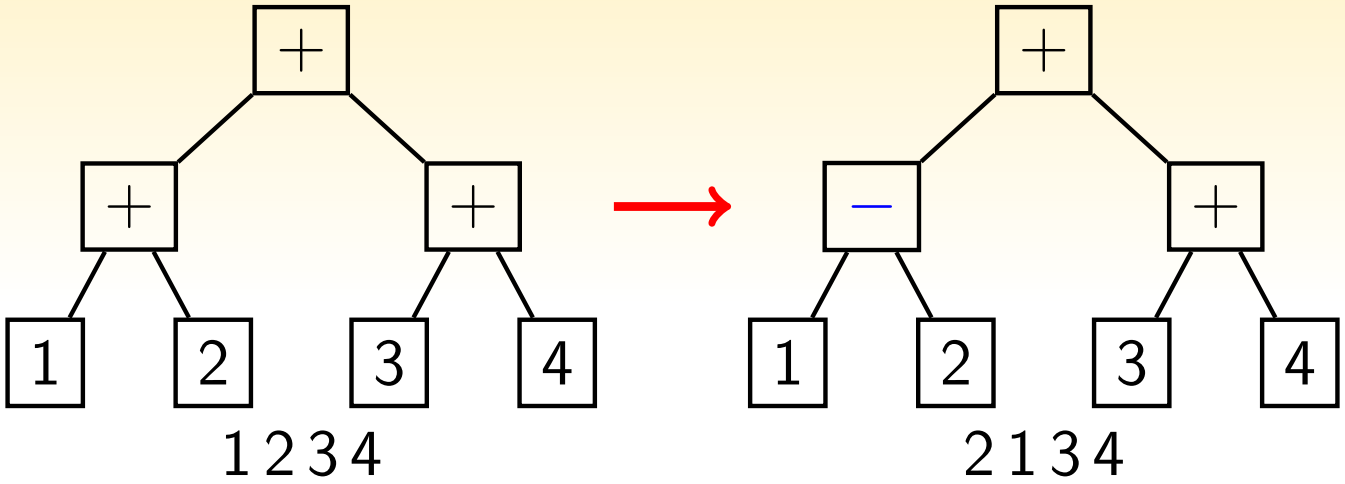
This time binary tree has time reversal symmetry elements in the nodes.



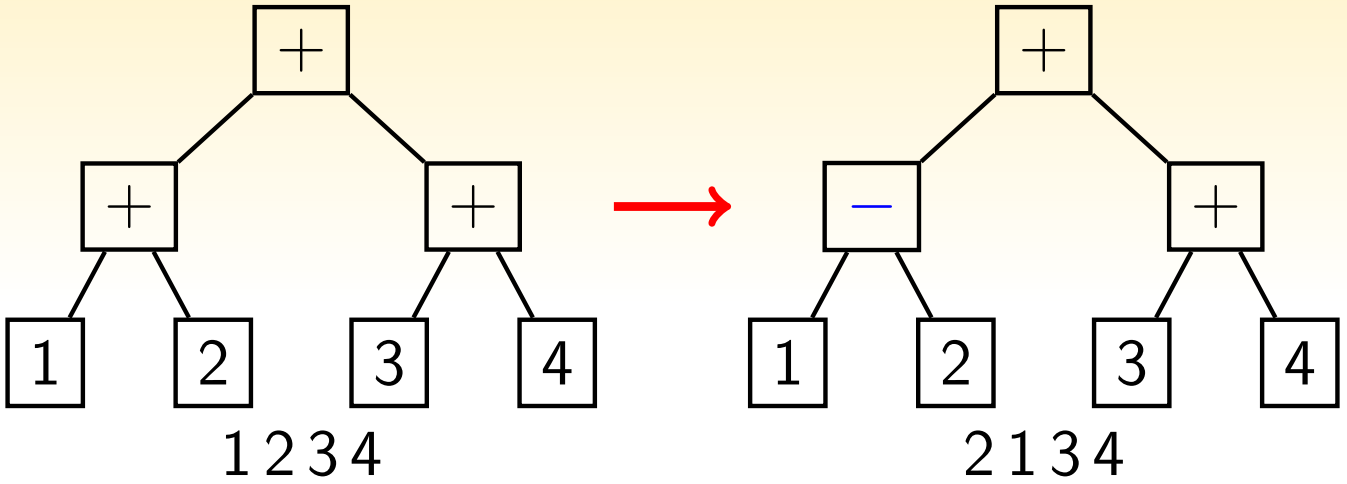
This time binary tree has time reversal symmetry elements in the nodes.

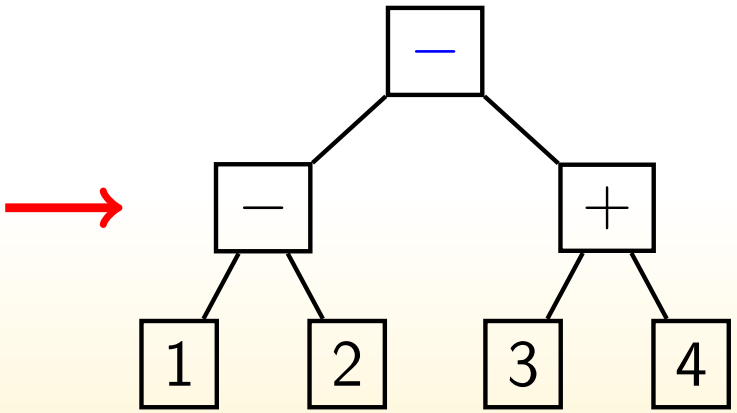


This time binary tree has time reversal symmetry elements in the nodes.

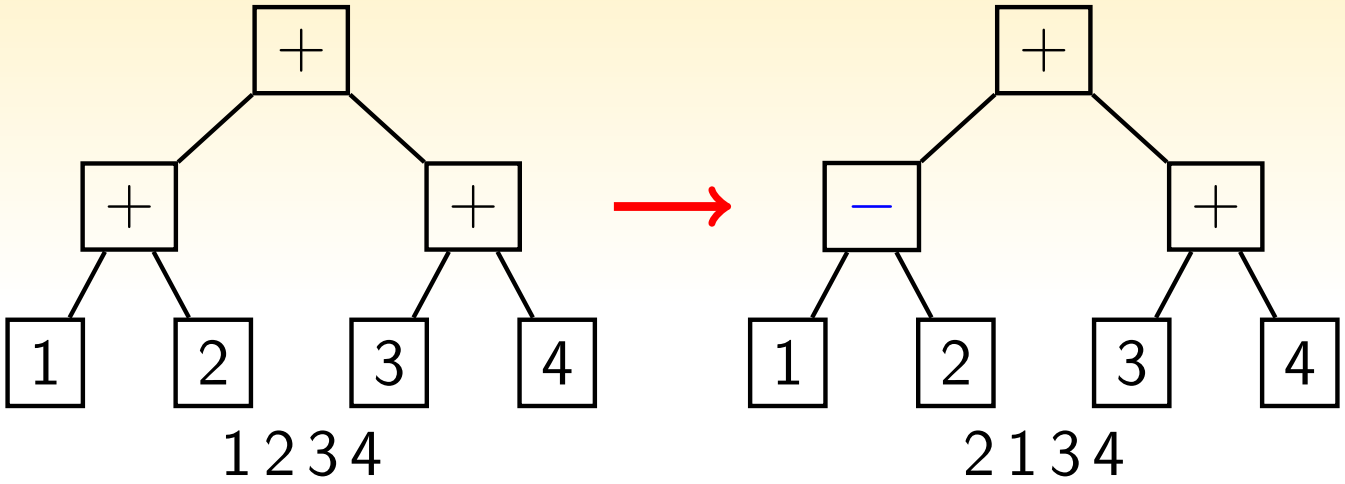


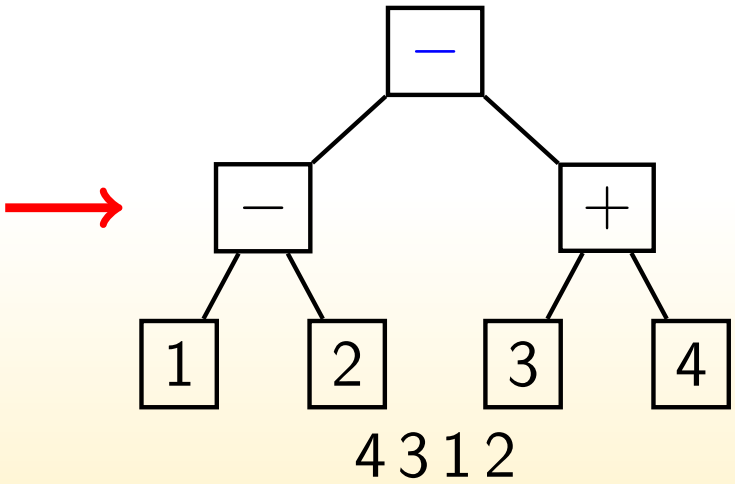
This time binary tree has time reversal symmetry elements in the nodes.



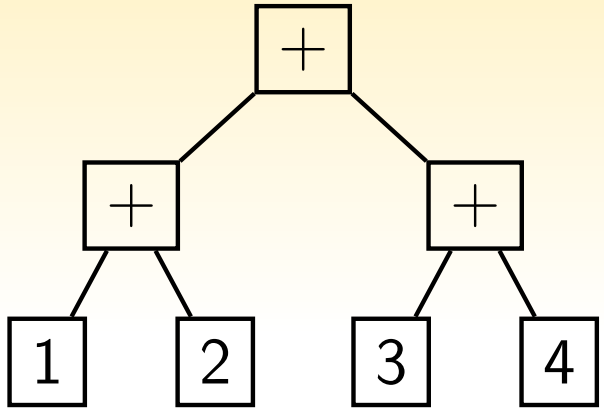


This time binary tree has time reversal symmetry elements in the nodes.

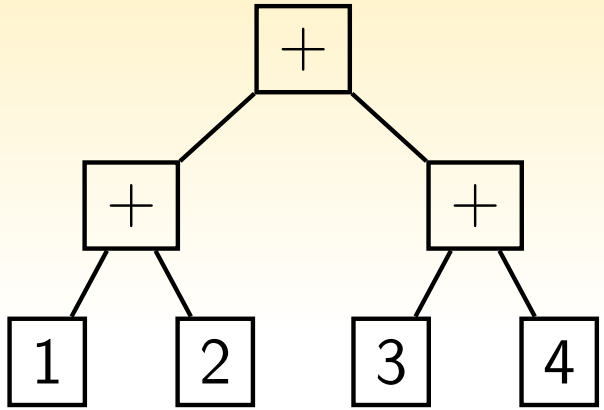




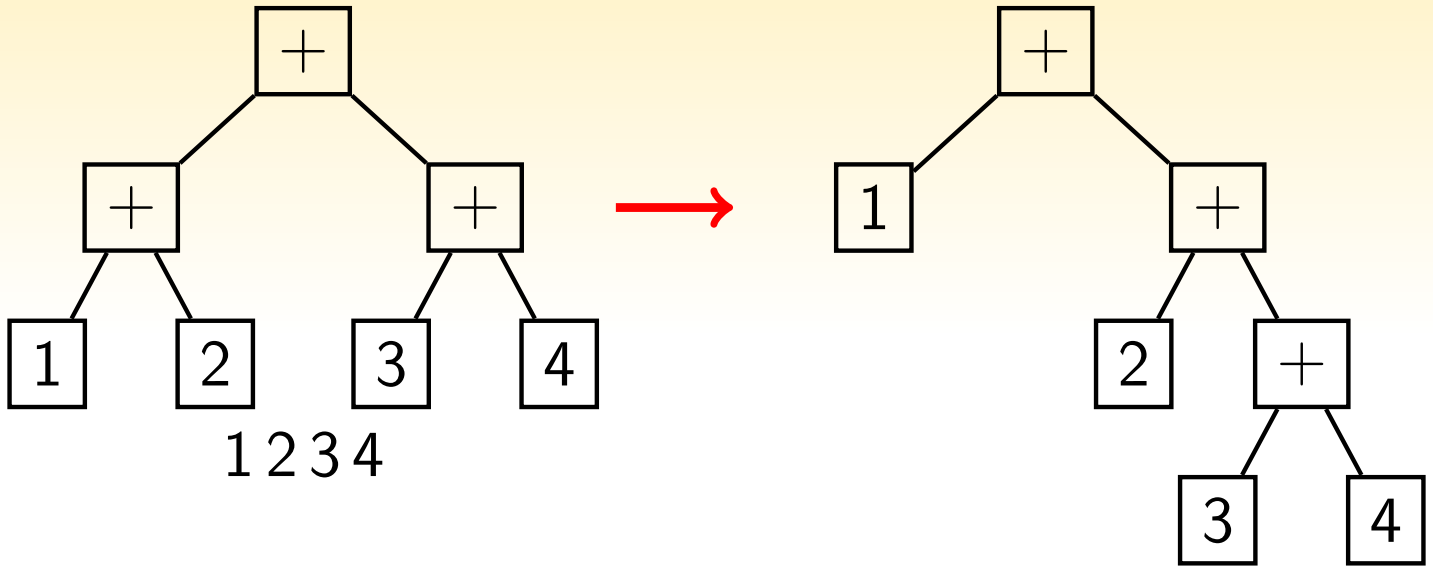
How do we reverse sequences of steps which don't align with the tree?



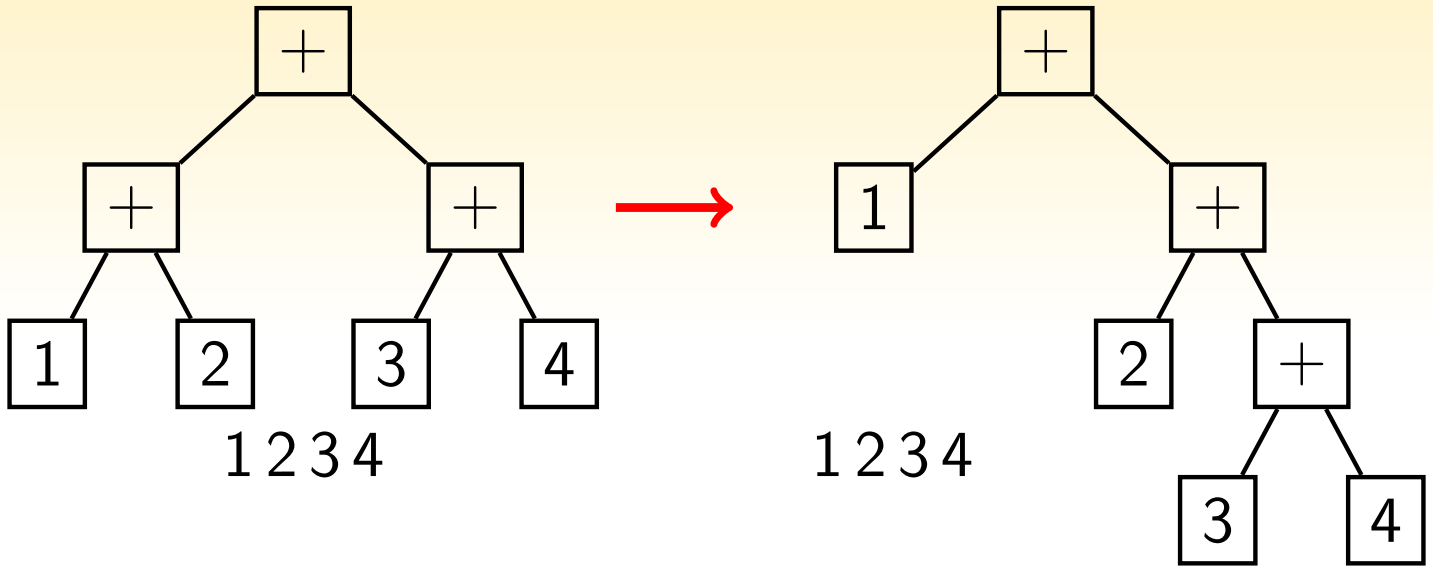
How do we reverse sequences of steps which don't align with the tree?



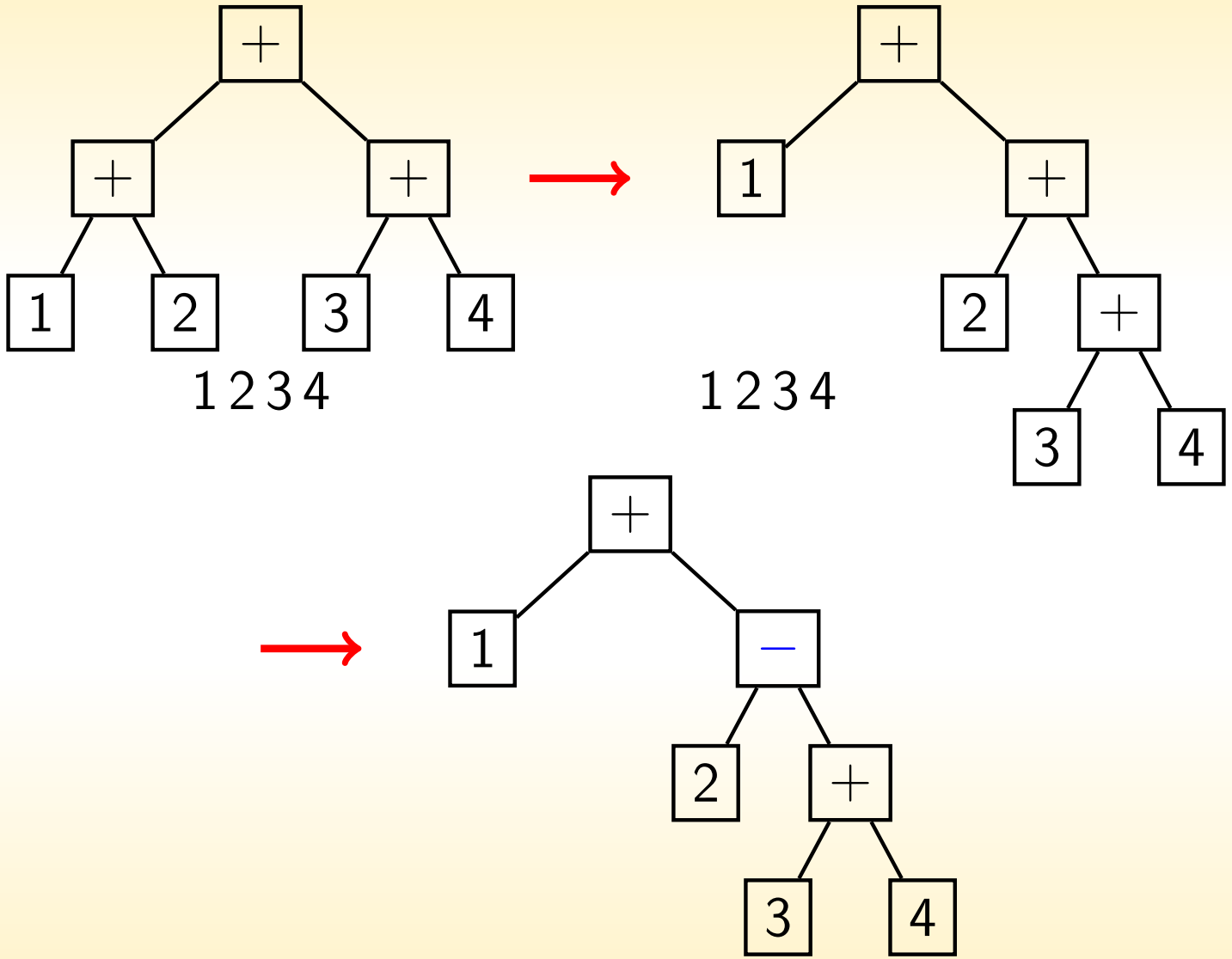
How do we reverse sequences of steps which don't align with the tree?



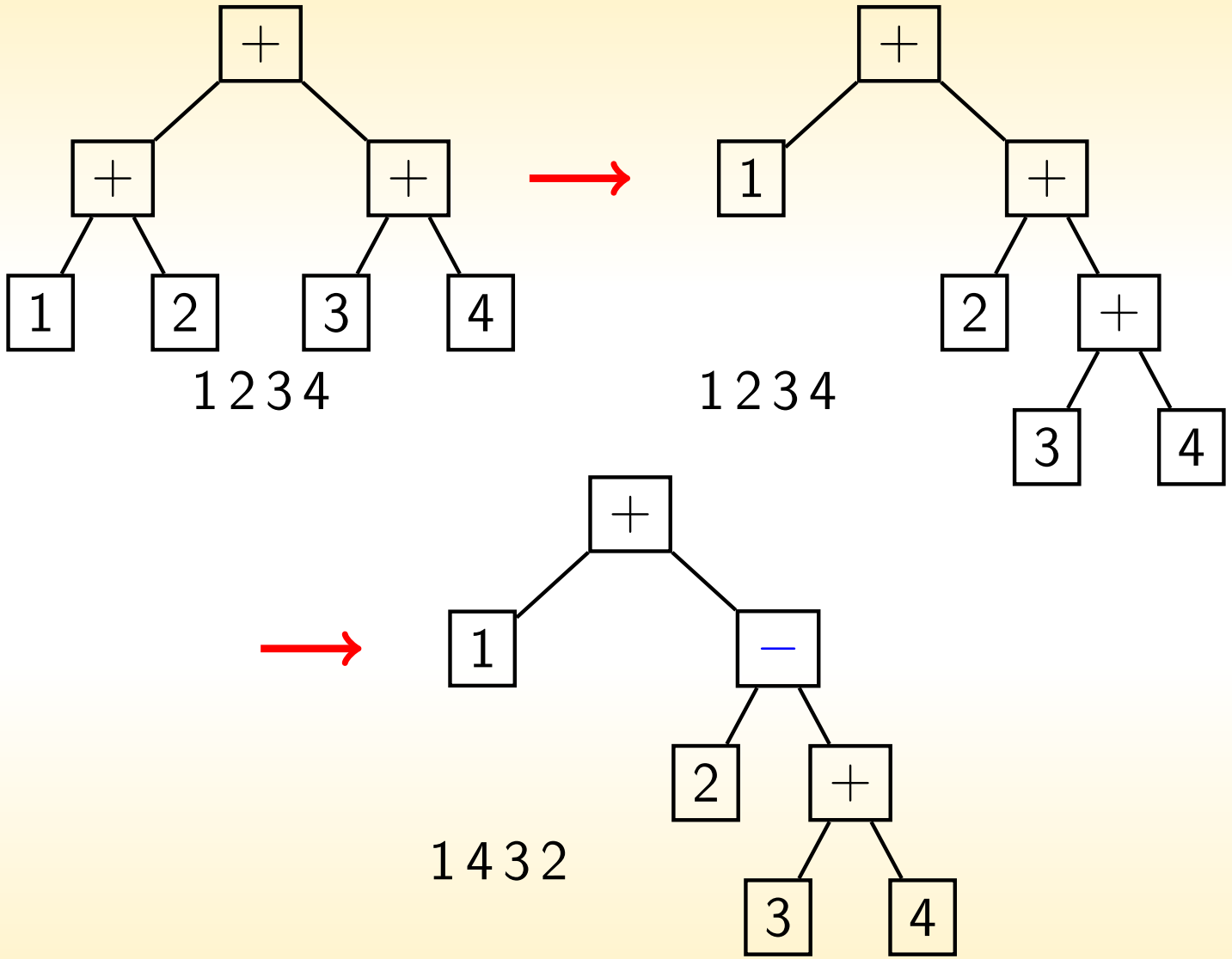
How do we reverse sequences of steps which don't align with the tree?



How do we reverse sequences of steps which don't align with the tree?



How do we reverse sequences of steps which don't align with the tree?



Results

- $\mathbb{Z}^2$: $L = 8192$, $N = 67\,108\,864$: 360 000 steps, versus 6.77 tree rotations.

Results

- $\mathbb{Z}^2$: $L = 8192$, $N = 67\,108\,864$: 360 000 steps, versus 6.77 tree rotations.
- $\mathbb{Z}^3$: $L = 512$, $N = 134\,217\,728$: 46 million steps, versus 20.6 tree rotations.

Conclusion

- Finding the right computer representation can allow for global changes to made for local cost.

Conclusion

- Finding the right computer representation can allow for global changes to made for local cost.
- Apply global move without explicitly evaluating all of the effects of the move - just need to be able to test some properties of the object efficiently.

Conclusion

- Finding the right computer representation can allow for global changes to made for local cost.
- Apply global move without explicitly evaluating all of the effects of the move - just need to be able to test some properties of the object efficiently.
- Linear order of walks makes things easy; can we apply these principles elsewhere?