Discussion Section 8

- Baum-Welch

- NP-completeness proofs (or how to say “actually, this probably can't be done efficiently”)

Viterbi and Baum-Welch are maximizing different functions

Viterbi and Baum-Welch are maximizing different functions

$$\max_{p} P_{\theta}(p, S)$$

Viterbi and Baum-Welch are maximizing different functions

- Viterbi likelihood:

$$\max_{p} P_{\theta}(p, S)$$

- Baum-Welch likelihood:

$$\sum_{p} P_{\theta}(p, S)$$

Baum-Welch

Baum-Welch

1) Use **forward algorithm** to find log likelihood of the sequence (i.e. sum of all paths)

Baum-Welch

1) Use **forward algorithm** to find log likelihood of the sequence (i.e. sum of all paths)

2) Use **forward-backward** to get fractional counts for each edge type

Baum-Welch

1) Use **forward algorithm** to find log likelihood of the sequence (i.e. sum of all paths)

2) Use **forward-backward** to get fractional counts for each edge type

(total probability of paths passing through edge)/
(total probability of all paths)

Baum-Welch

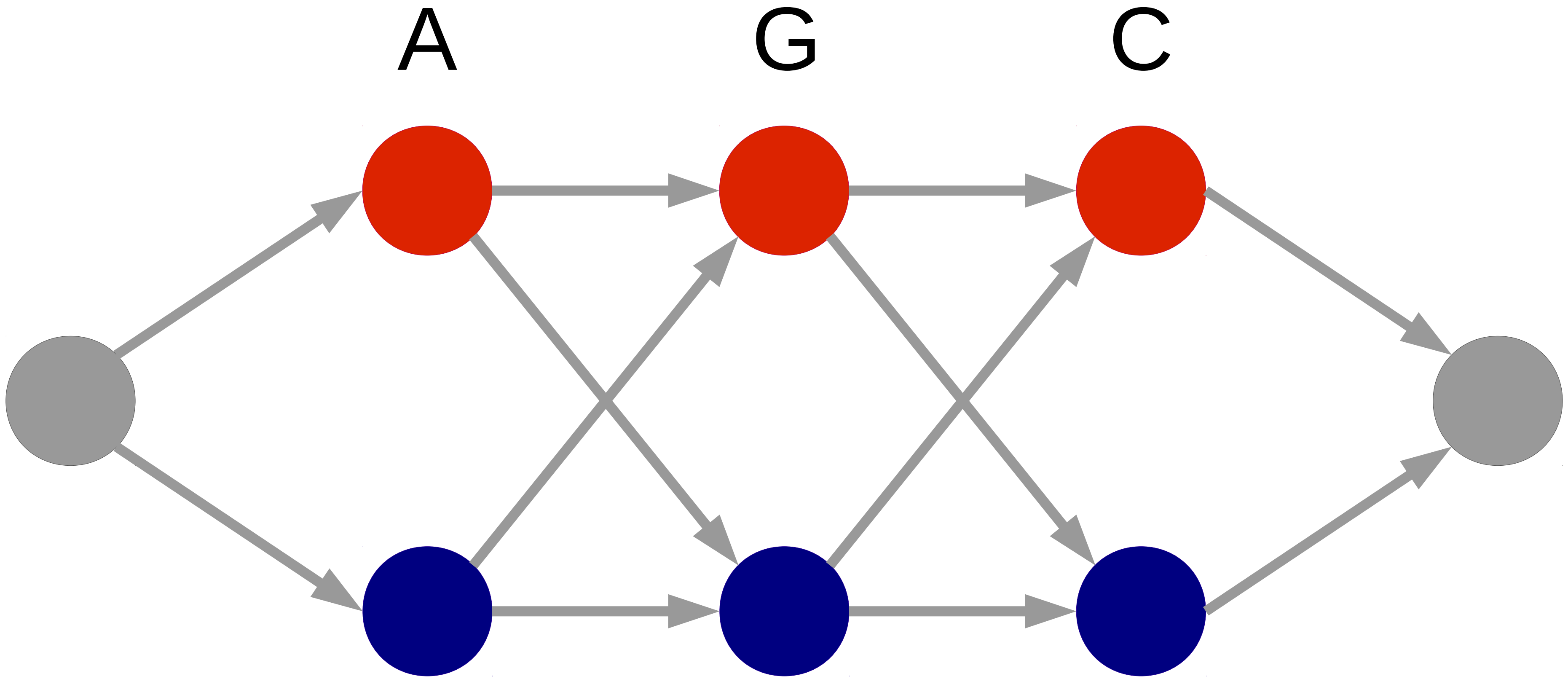
1) Use **forward algorithm** to find log likelihood of the sequence (i.e. sum of all paths)

2) Use **forward-backward** to get fractional counts for each edge type

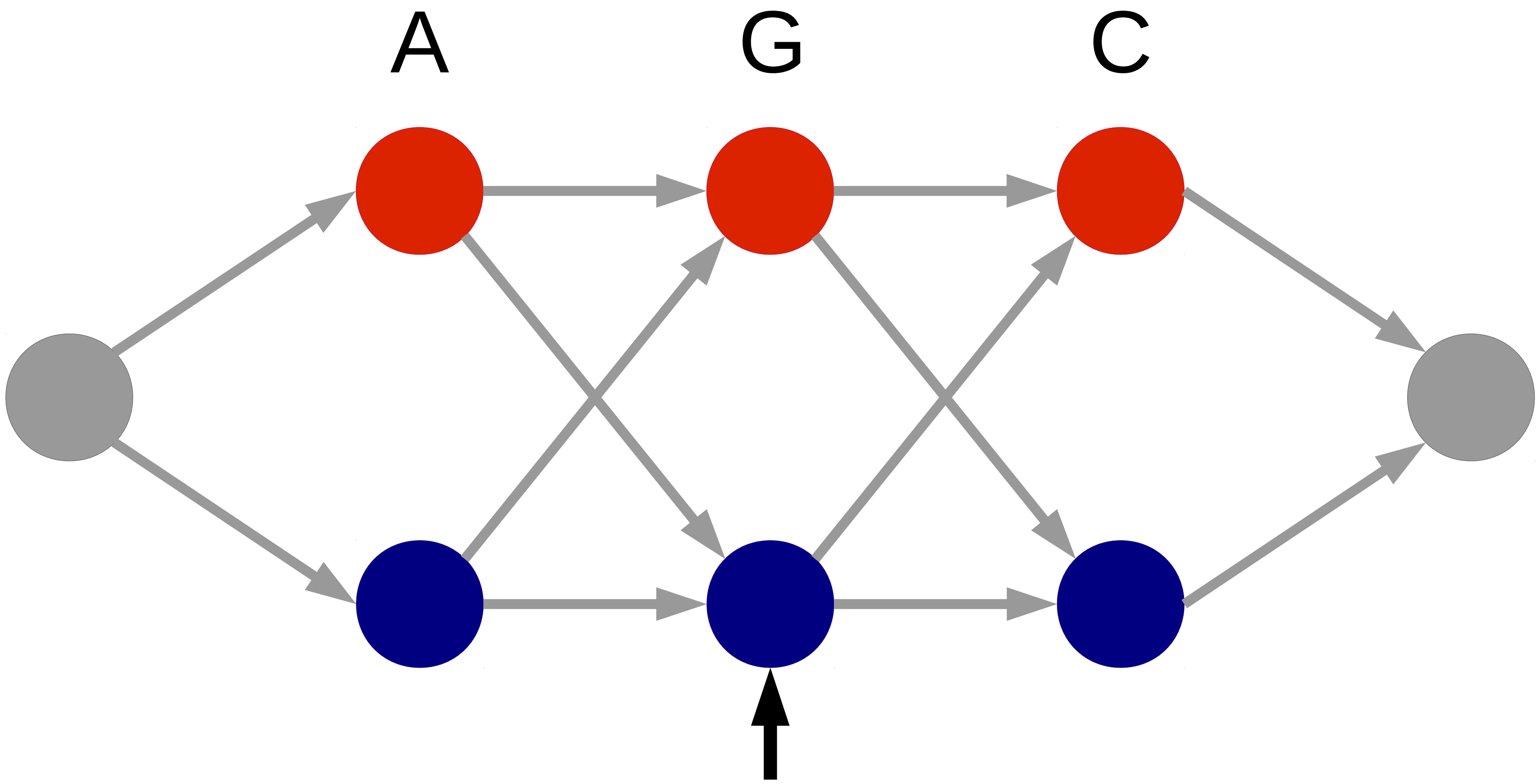
    (total probability of paths passing through edge)/
    (total probability of all paths)

3) Re-estimate transition and emission probabilities by calculating the expected number of each edge type

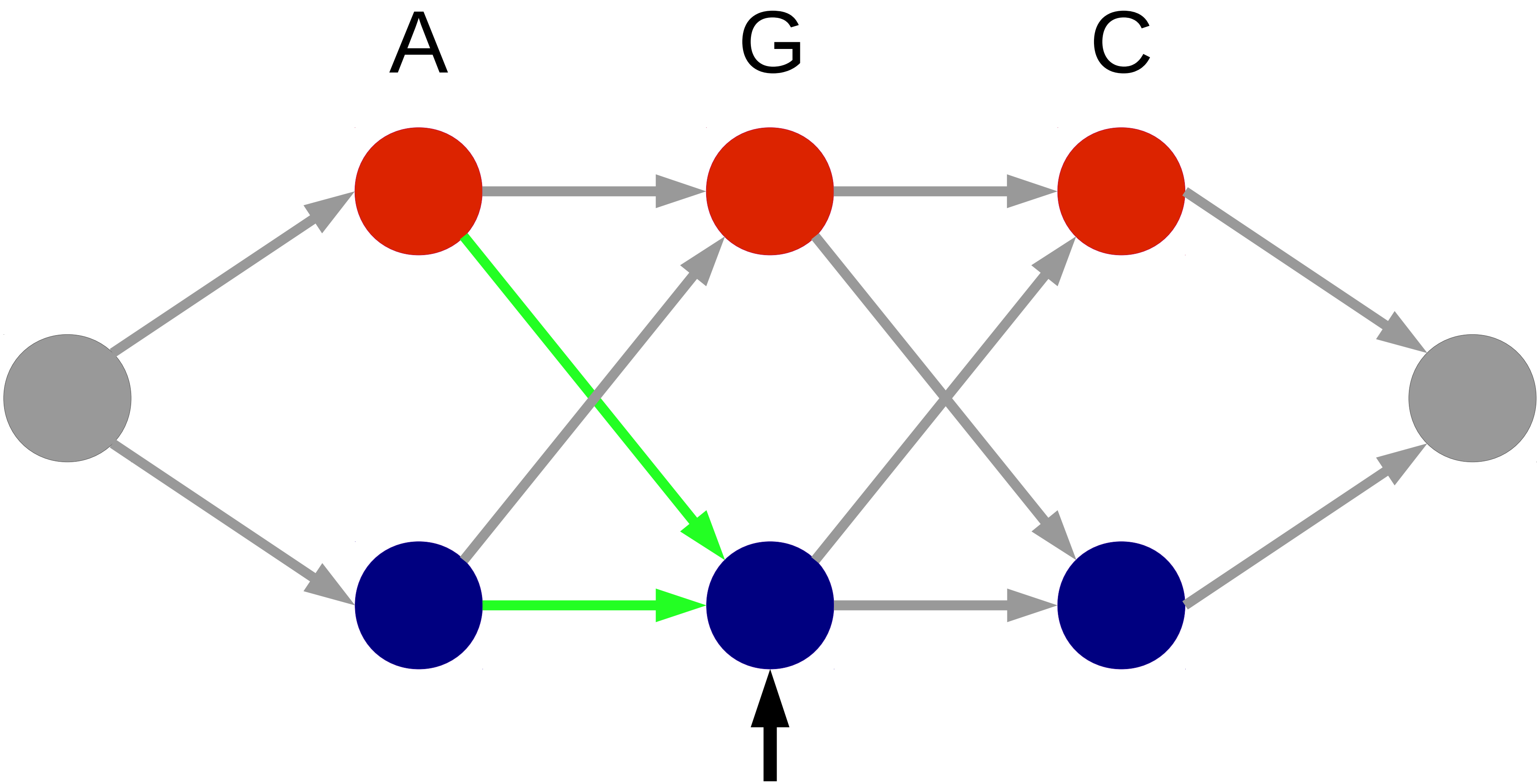
Forward-backward algorithm



Forward-backward algorithm



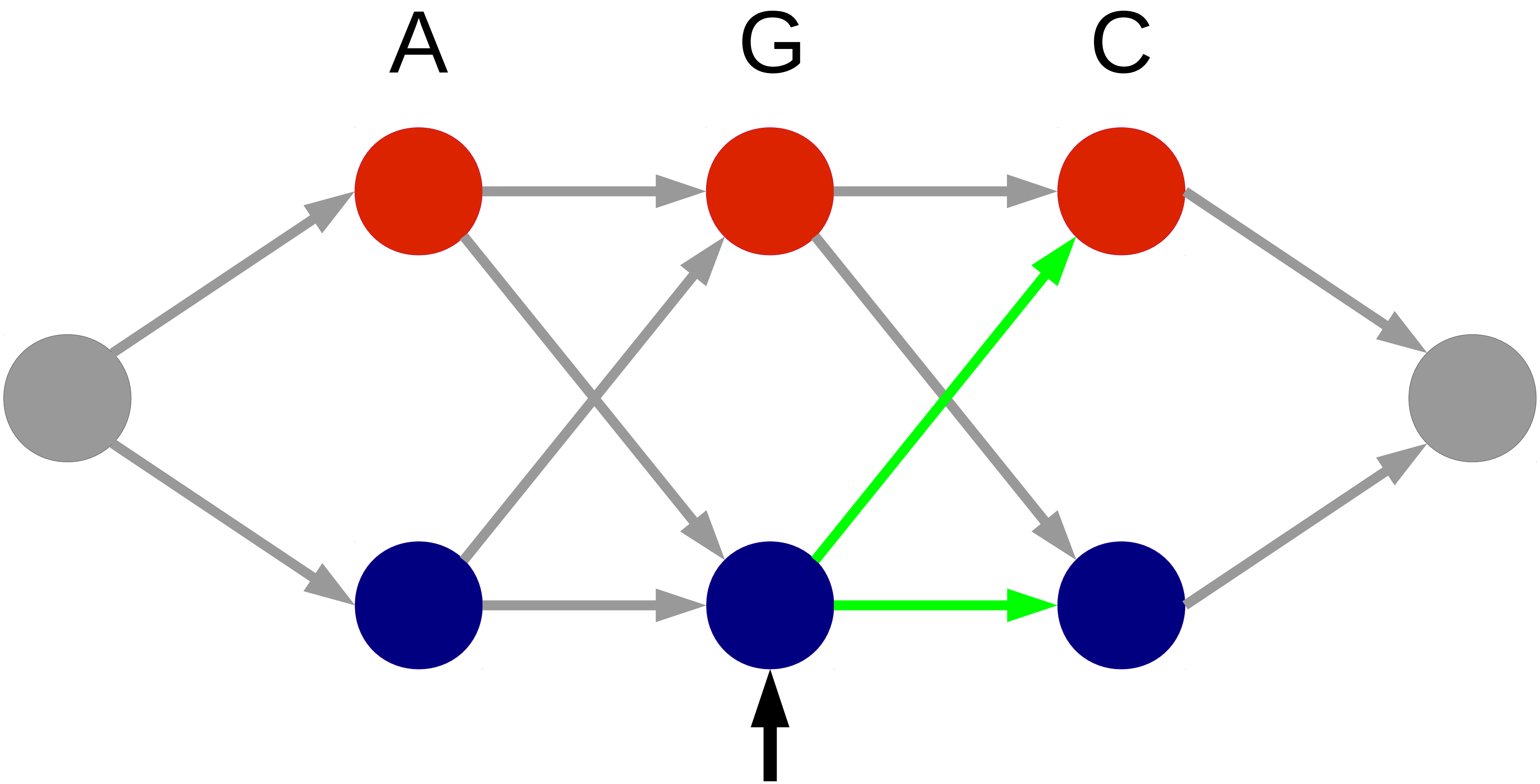
Forward-backward algorithm



For each node:

- Forward: Store the sum of probabilities of paths ending at position *i* state *k*

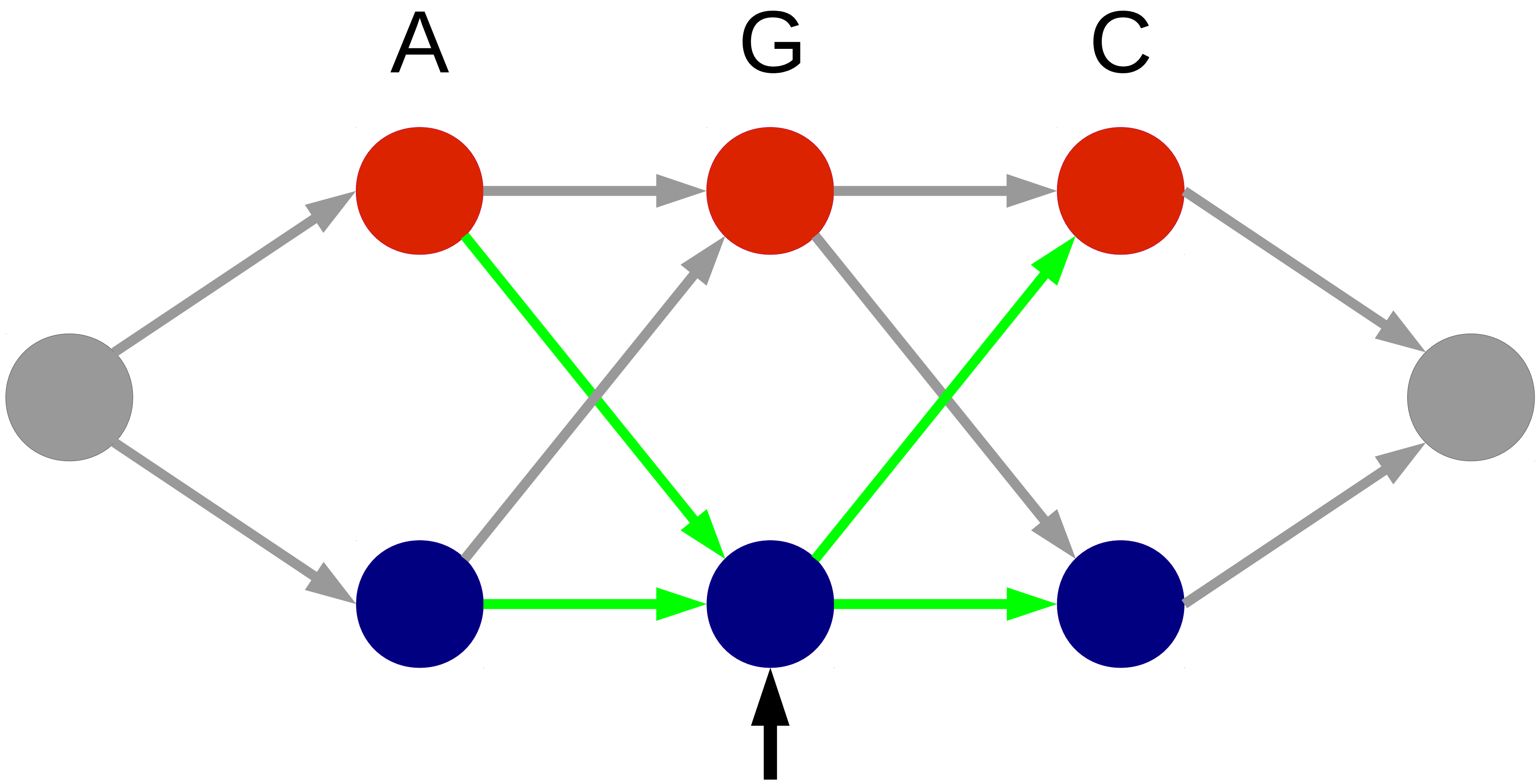
Forward-backward algorithm



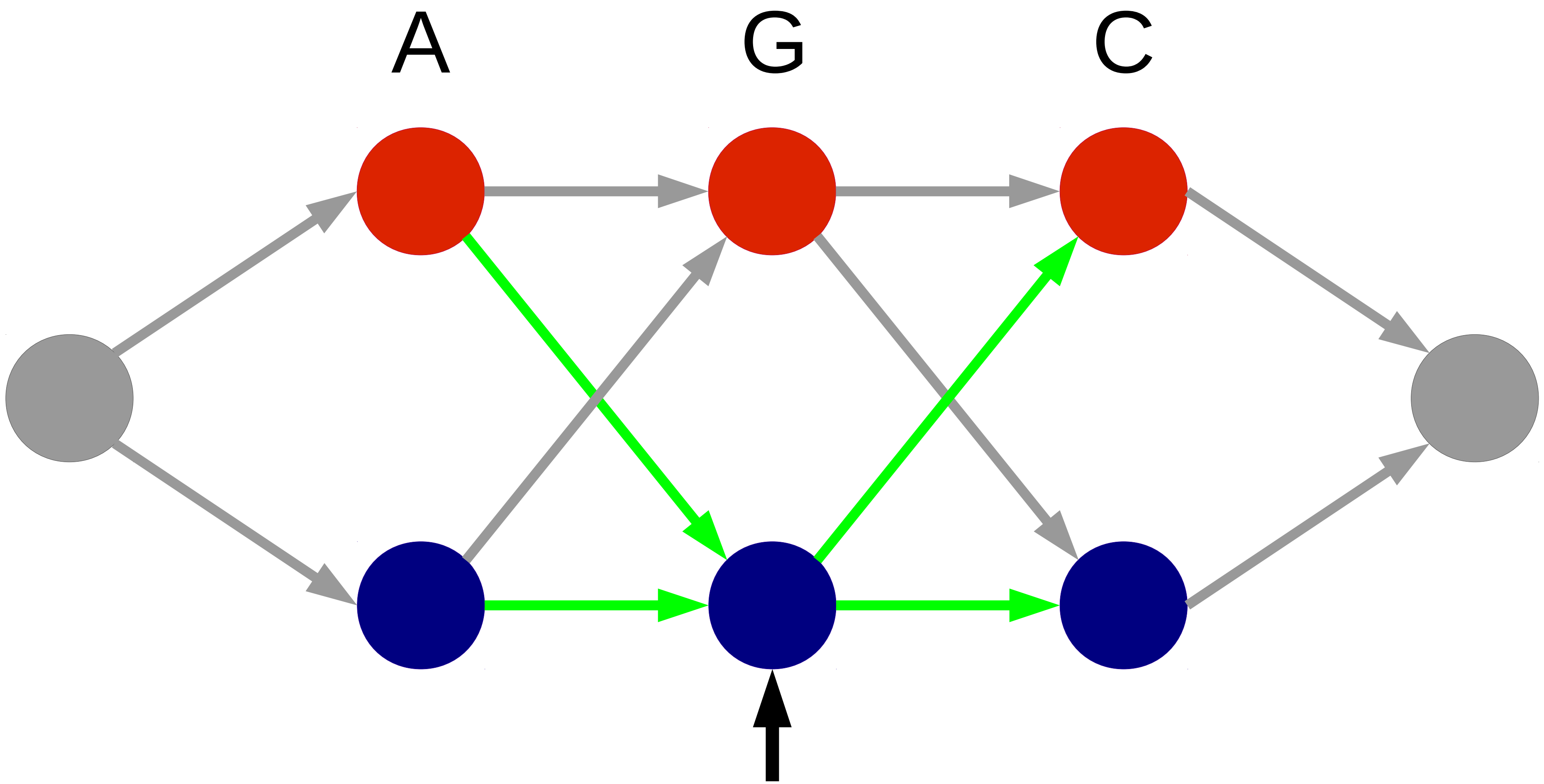
For each node:

- Forward: Store the sum of probabilities of paths ending at position *i* state *k*
- Backward: Store the sum of probabilities of paths starting at position *i* state *k*

Forward-backward algorithm

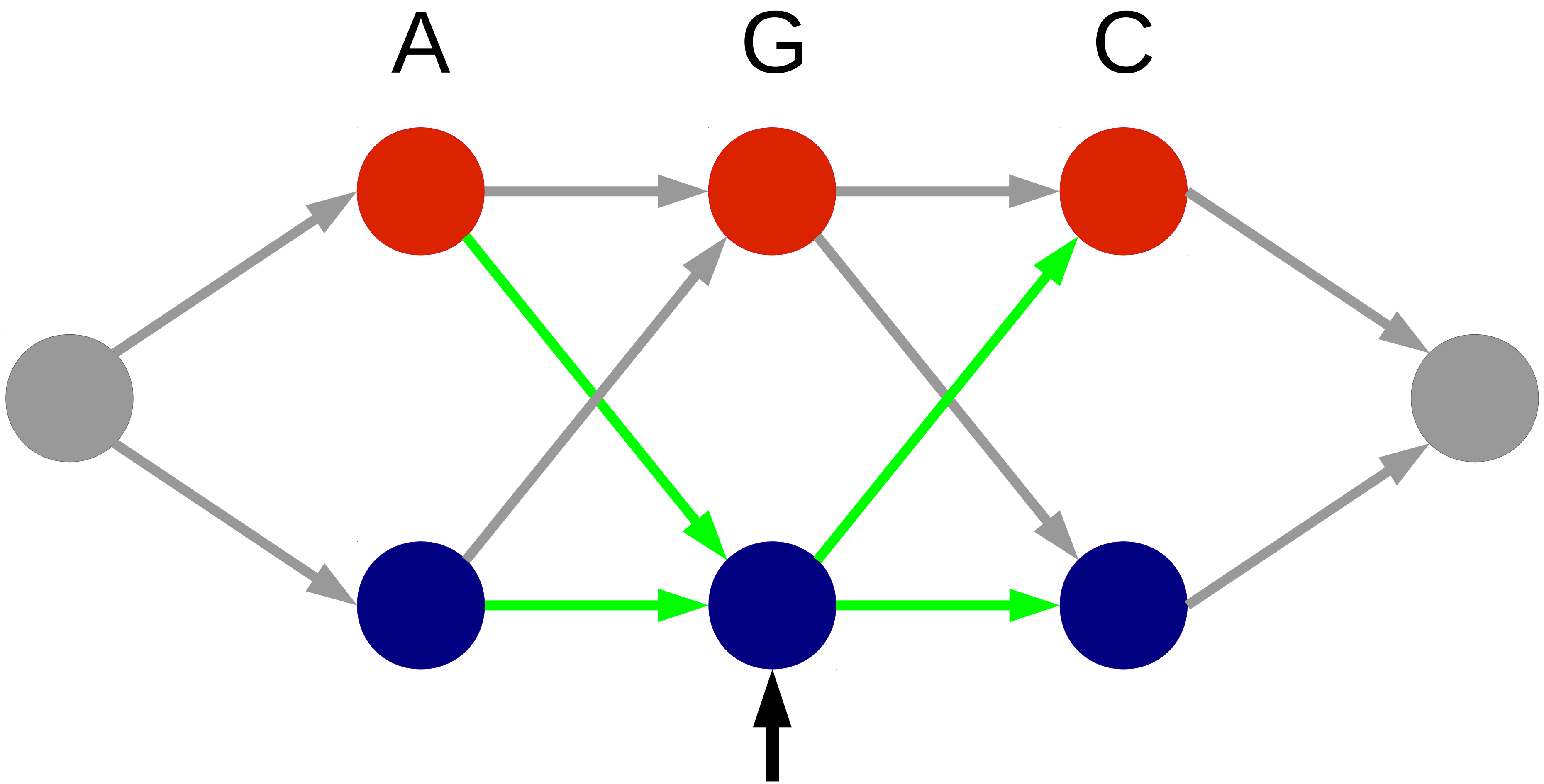


Forward-backward algorithm



Total probability of paths passing through position *i* state *k*:

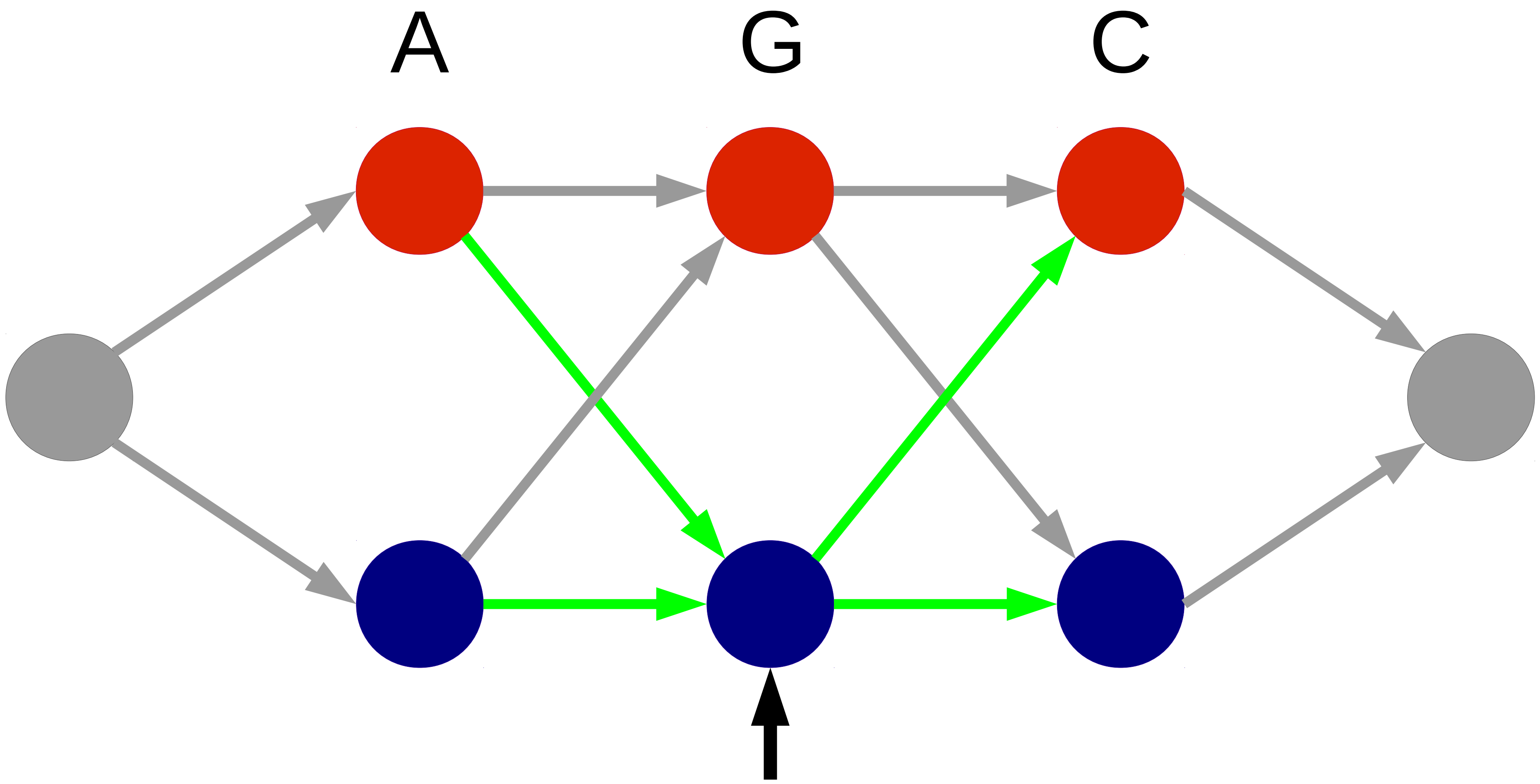
Forward-backward algorithm



Total probability of paths passing through position *i* state *k*:

- forward(*i*, *k*) x emission(S_i, *k*) x backward(*i*, *k*)

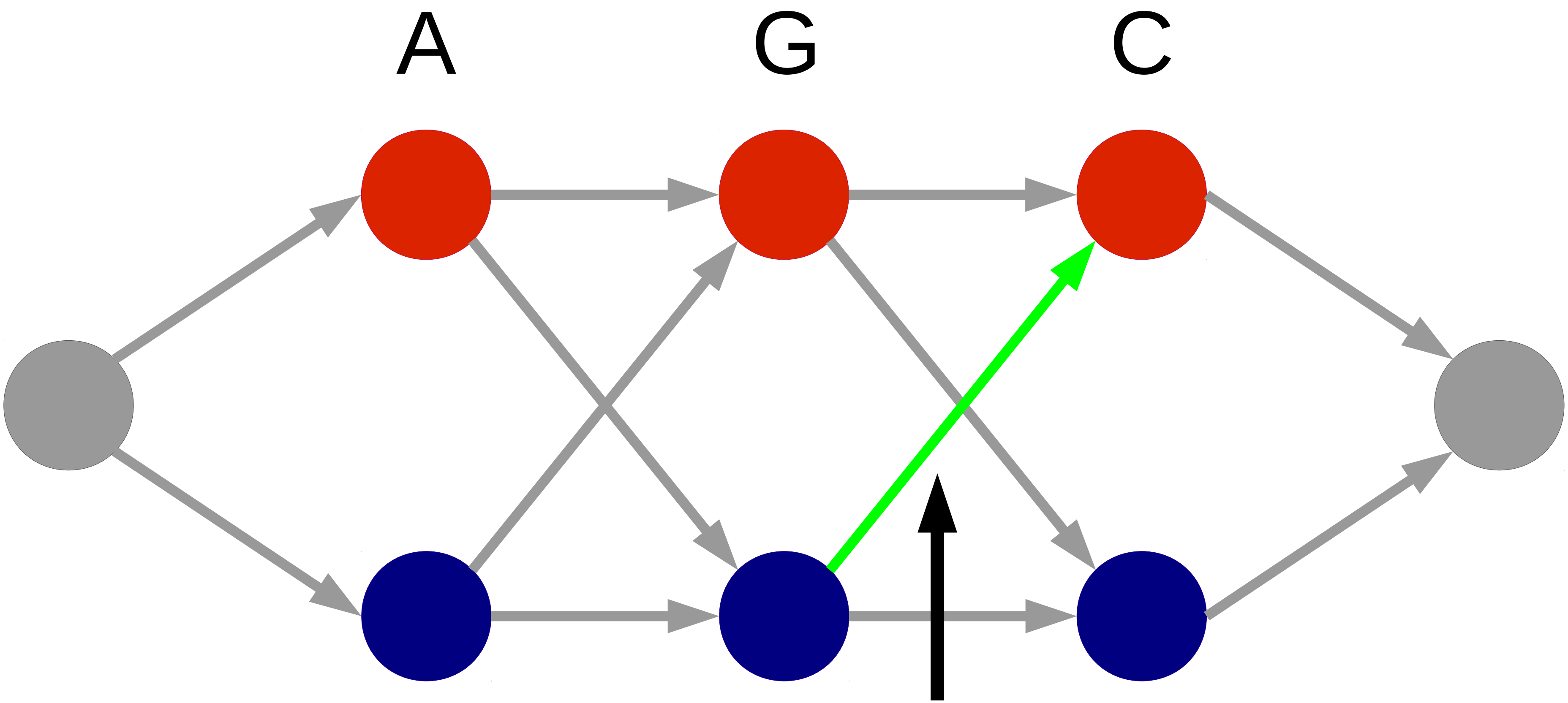
Forward-backward algorithm



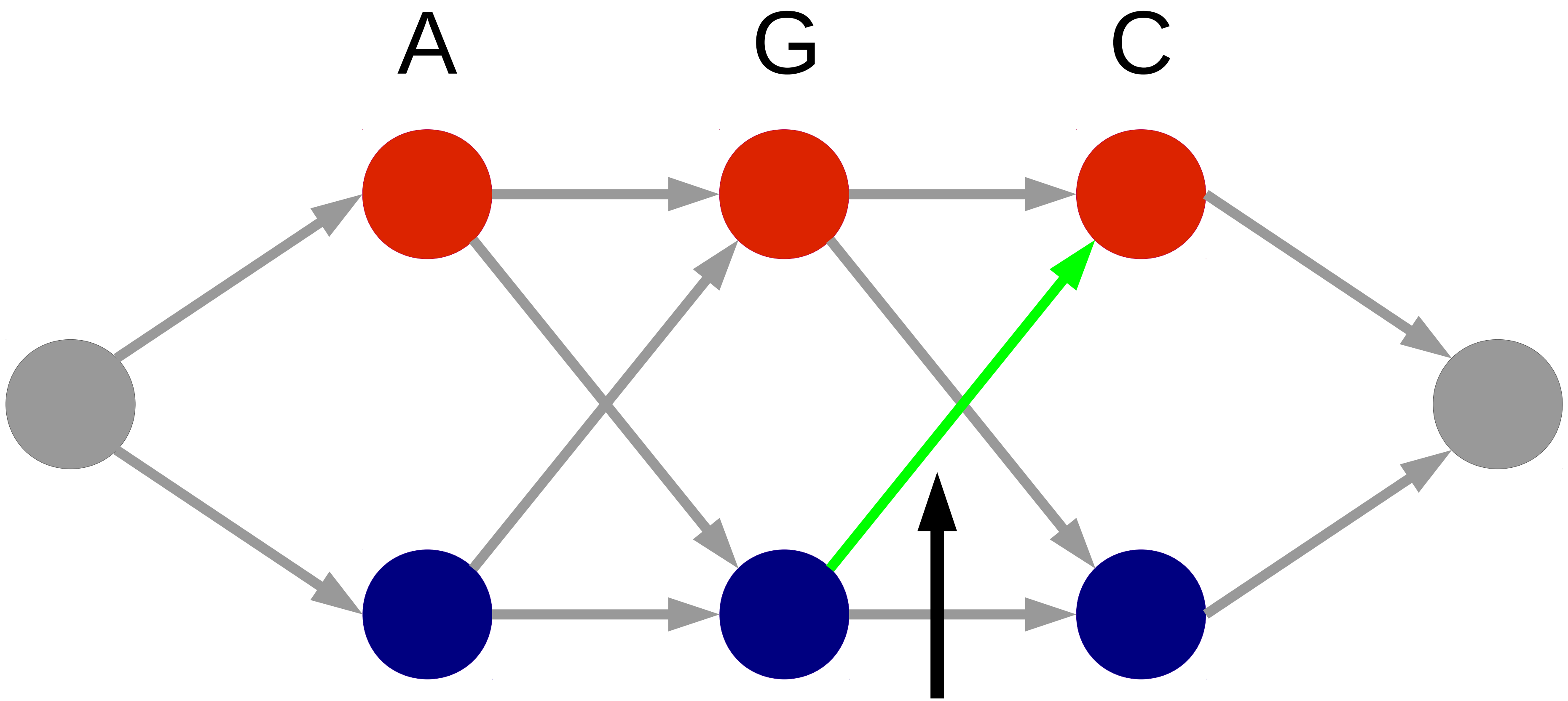
Total probability of paths passing through position *i* state *k*:

- forward(*i*, *k*) x emission(S_i, *k*) x backward(*i*, *k*)
- In this example, add this weighted count to the numerator for the blue state emitting 'G' and the denominator for all blue state emission probabilities

Forward-backward algorithm

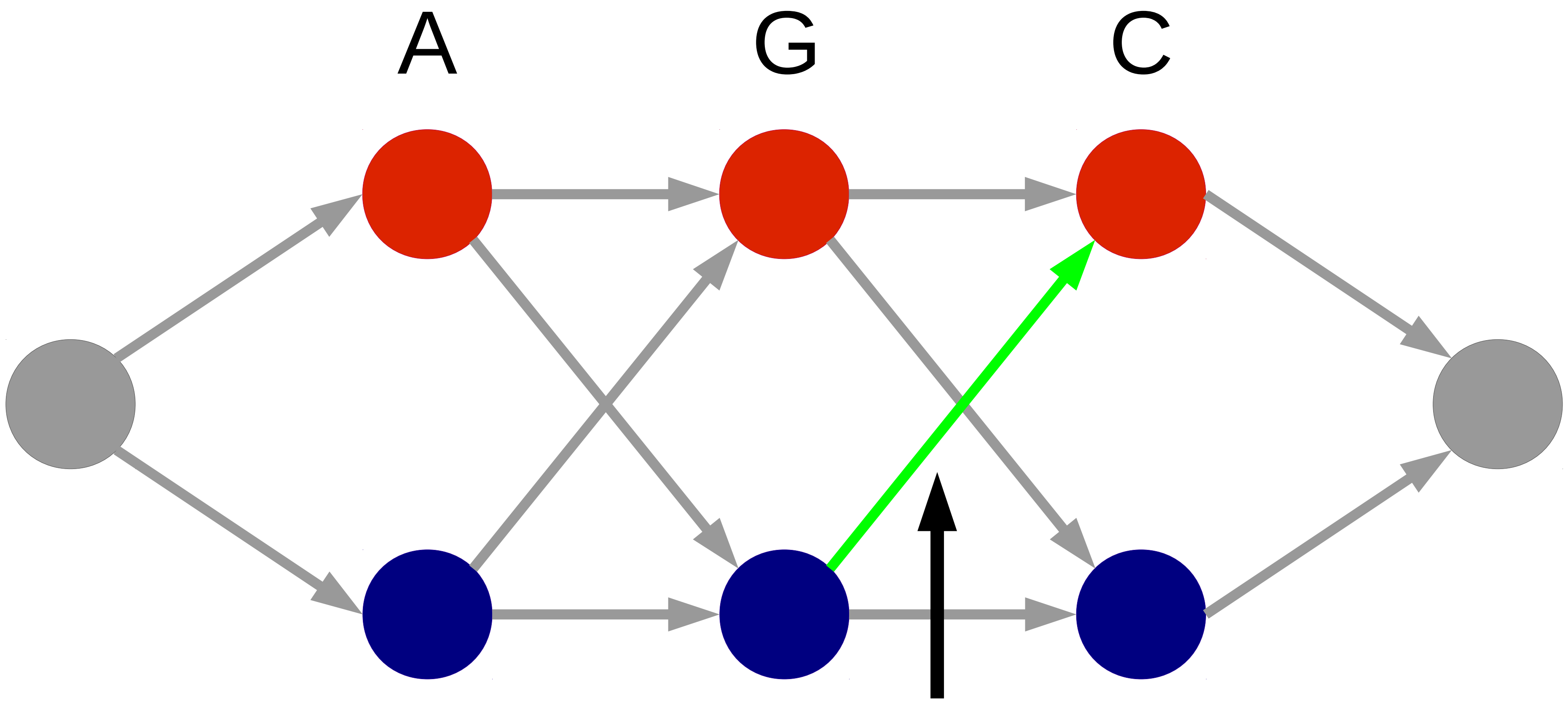


Forward-backward algorithm



Total probability of paths passing from position *i*-1 state *k*' to position *i* state *k*:

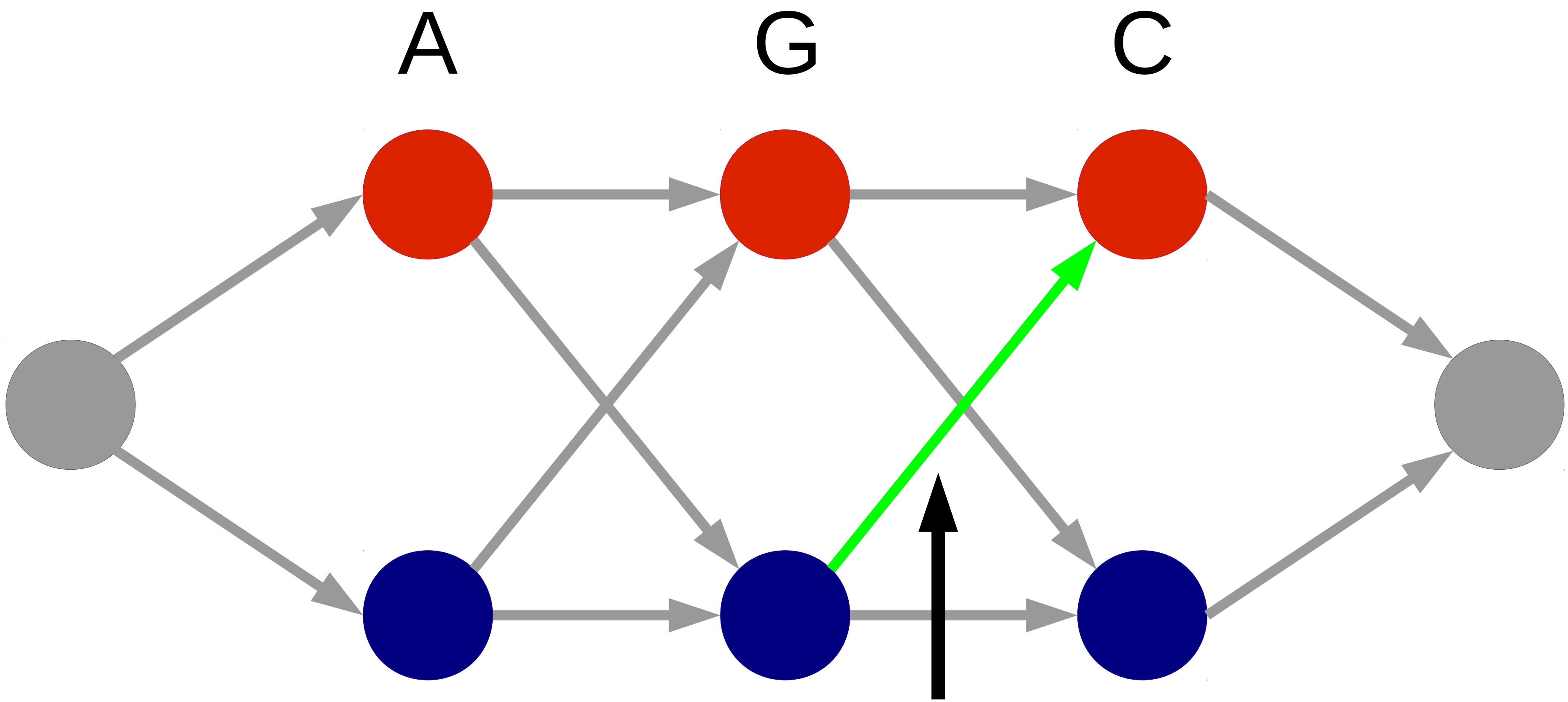
Forward-backward algorithm



Total probability of paths passing from position *i*-1 state *k*' to position *i* state *k*:

- forward(*i*-1, *k*') x emission(S_{i-1}, *k*') x transition(*k*', *k*) x emission(S_i, *k*) x backward(*i*, *k*)

Forward-backward algorithm



Total probability of paths passing from position *i*-1 state *k*' to position *i* state *k*:

- forward(*i*-1, *k*') x emission(S_{i-1}, *k*') x transition(*k*', *k*) x emission(S_i, *k*) x backward(*i*, *k*)
- In this example, add this weighted count to the numerator for the transitions from blue to red and the denominator for all transition out of blue states

An alternative way to think about updating

An alternative way to think about updating

- Some terminology for the following slides

An alternative way to think about updating

- Some terminology for the following slides
  - $\alpha_k(i)$: The forward probability of being in state *k* at position *i*

An alternative way to think about updating

- Some terminology for the following slides
  - $\alpha_k(i)$: The forward probability of being in state *k* at position *i*
  - $\beta_k(i)$: The backward probability of being in state *k* at position *i*

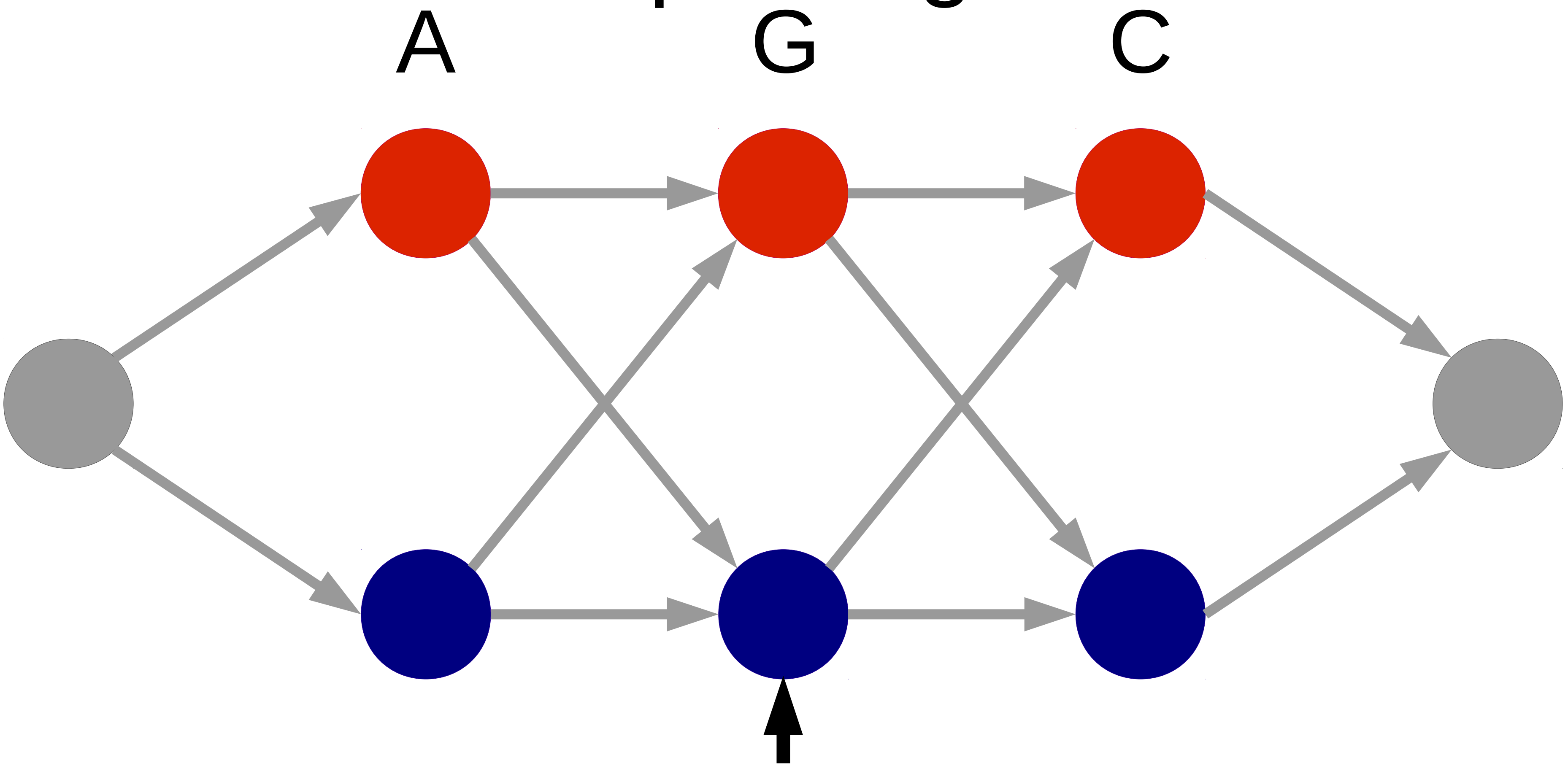
An alternative way to think about updating

- Some terminology for the following slides
  - $\alpha_k(i)$: The forward probability of being in state *k* at position *i*
  - $\beta_k(i)$: The backward probability of being in state *k* at position *i*
  - $e_k(S_i)$: The emission probability of the character at position *i* in state *k*

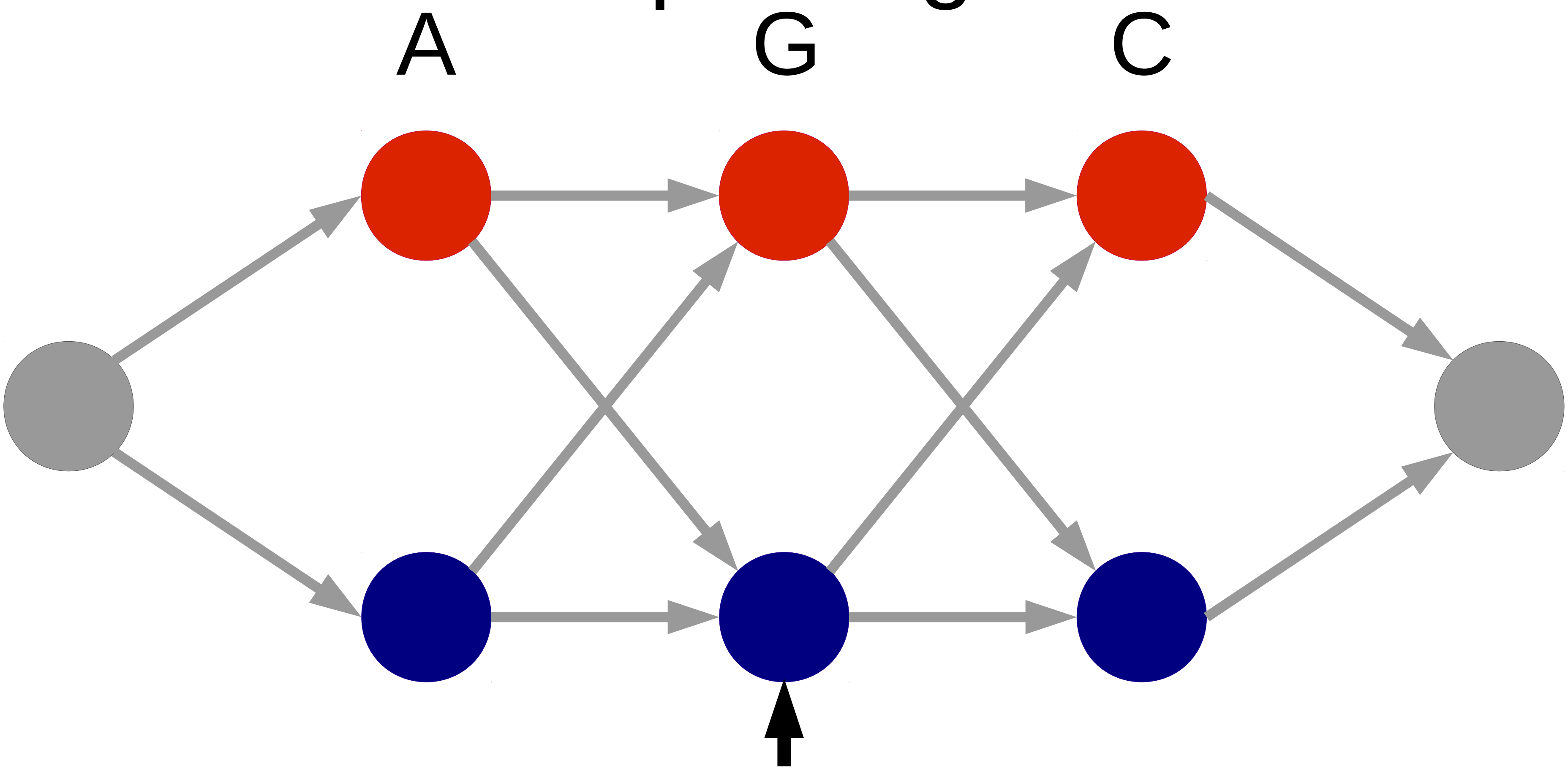
An alternative way to think about updating

- Some terminology for the following slides
  - $\alpha_k(i)$: The forward probability of being in state *k* at position *i*
  - $\beta_k(i)$: The backward probability of being in state *k* at position *i*
  - $e_k(S_i)$: The emission probability of the character at position *i* in state *k*
  - a_{kl}: The transition probability from state *k* to state *l*

An alternative way to think about updating

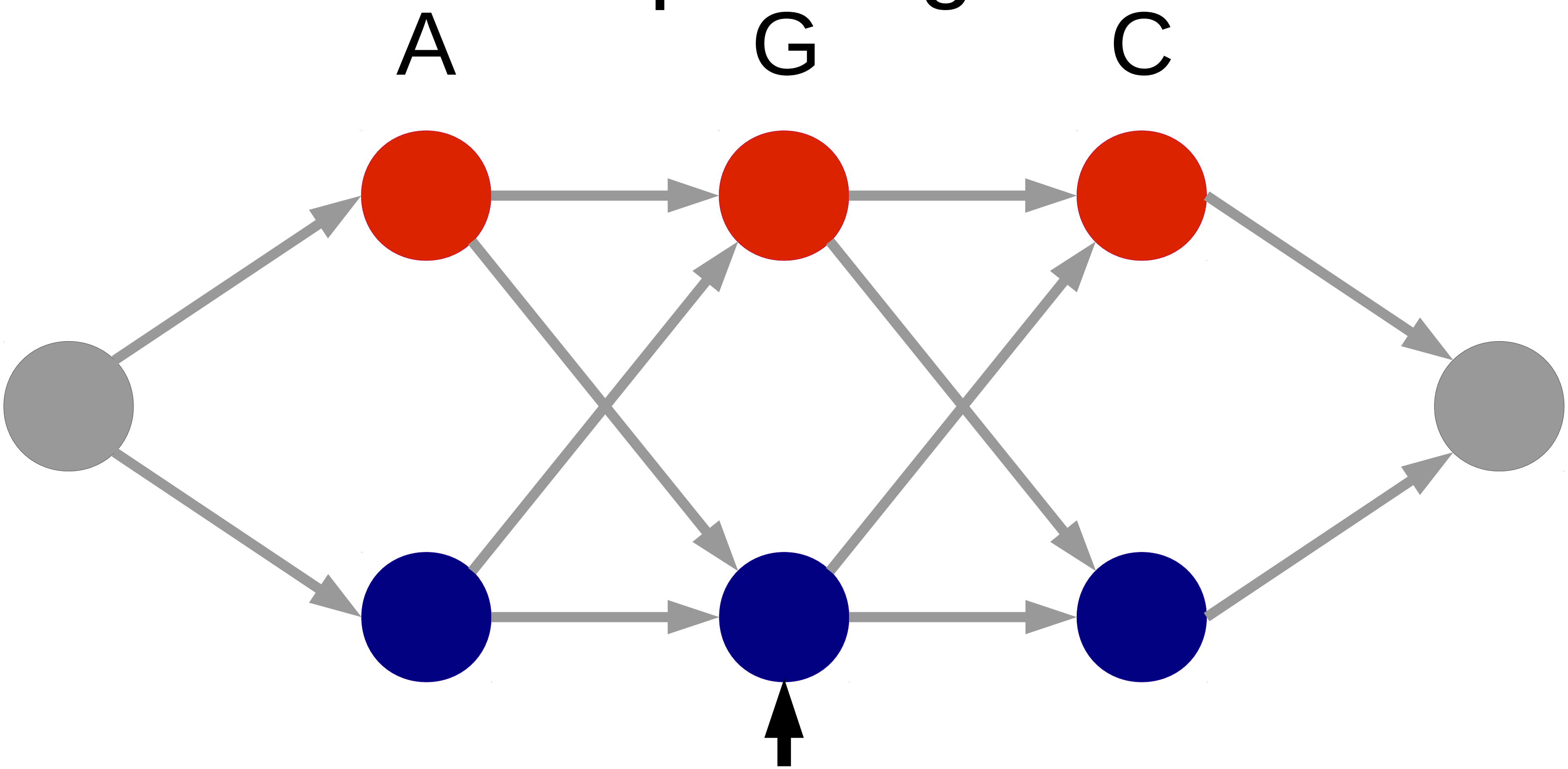


An alternative way to think about updating



Consider the probabilities at each position:

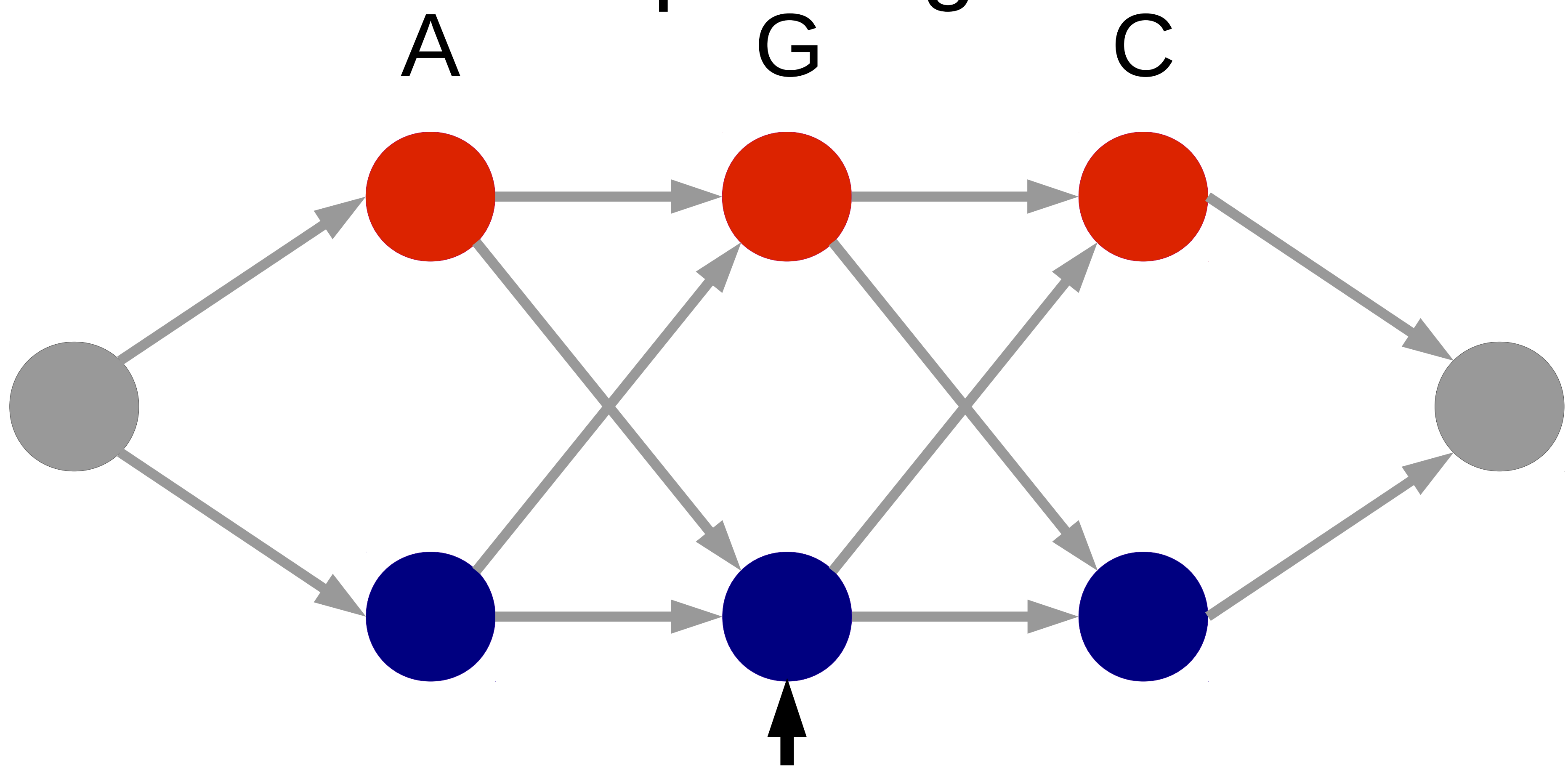
An alternative way to think about updating



Consider the probabilities at each position:

- figure out the probability of being in state *k* at position *i*

An alternative way to think about updating

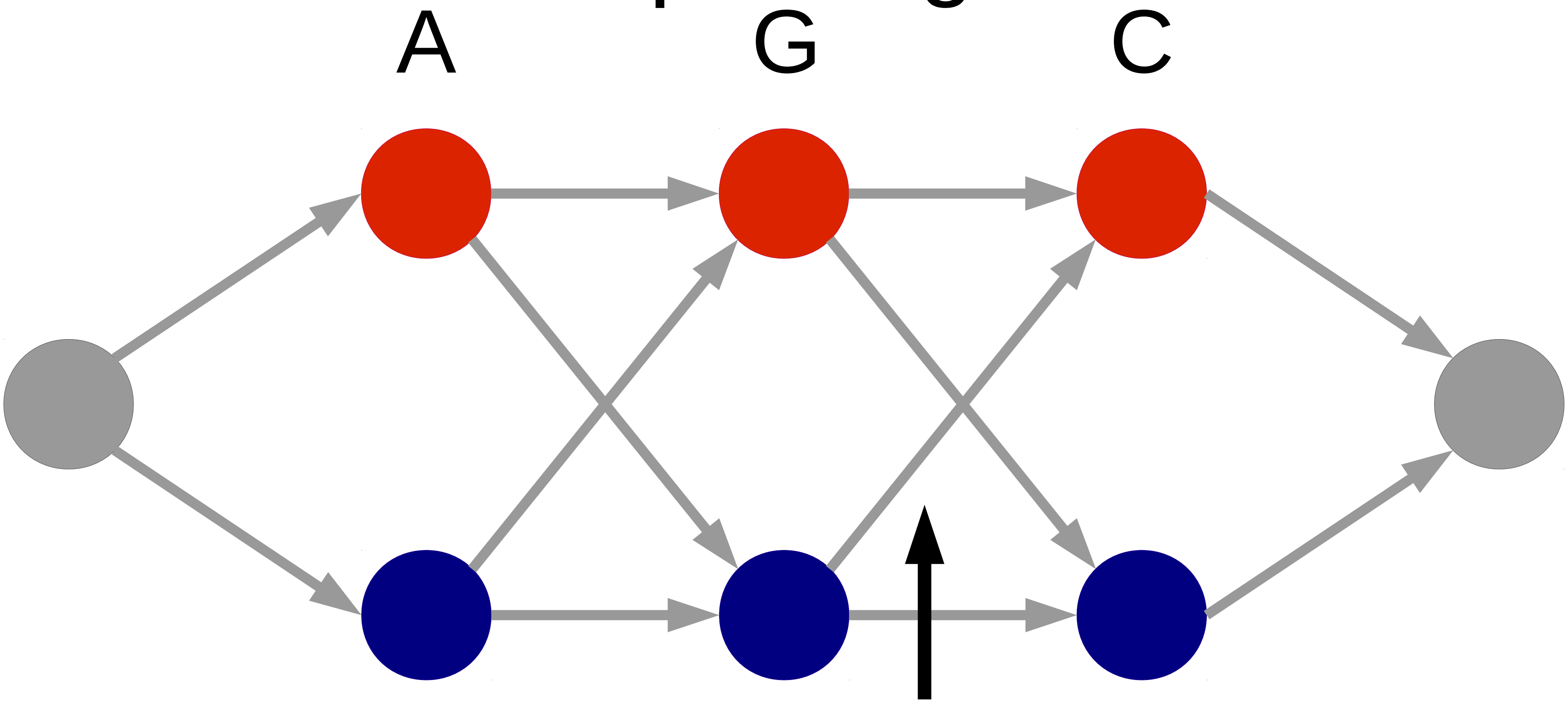


Consider the probabilities at each position:

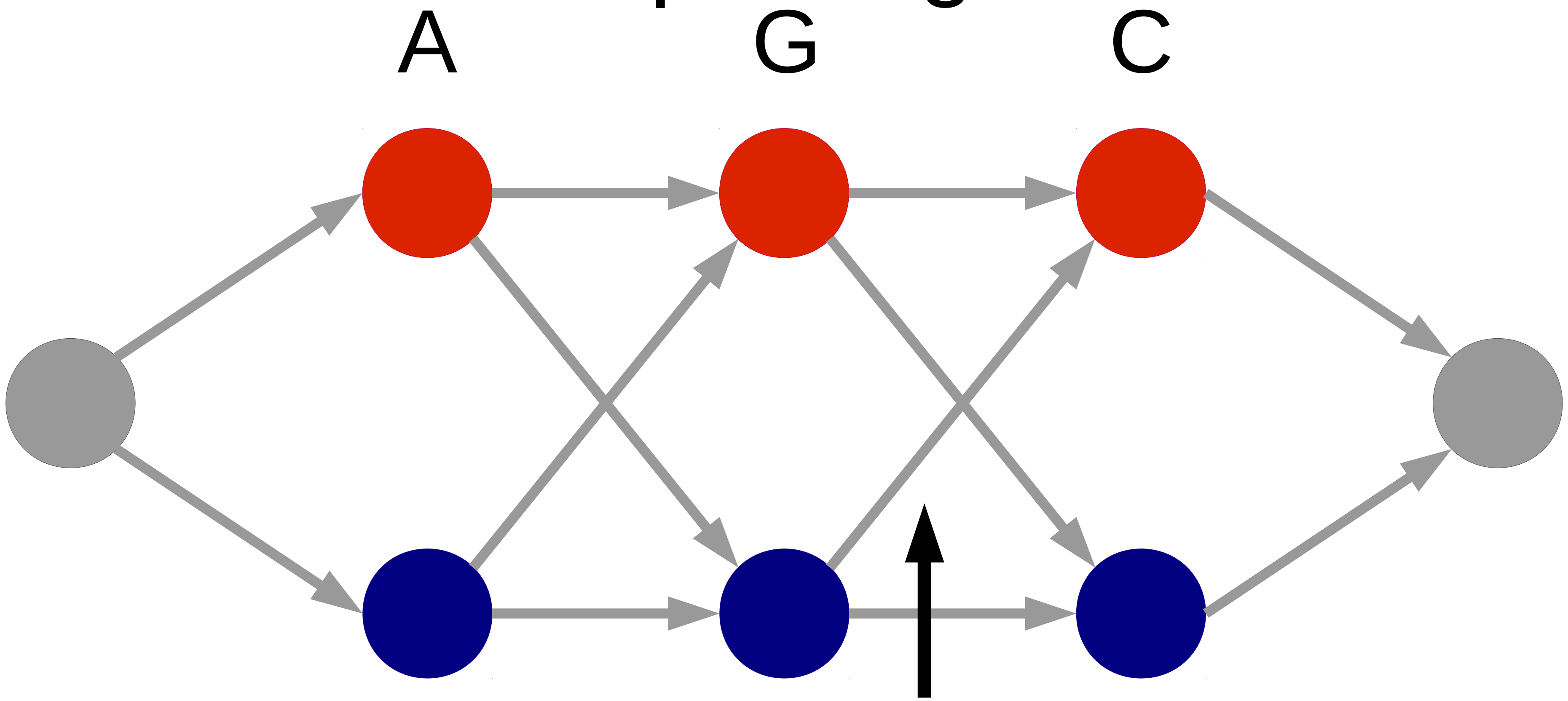
- figure out the probability of being in state *k* at position *i*

$$\gamma_k(i) = \frac{\alpha_k(i) e_k(S_i) \beta_k(i)}{\sum_{j=1}^{N} \alpha_j(i) e_j(S_i) \beta_j(i)}$$

An alternative way to think about updating

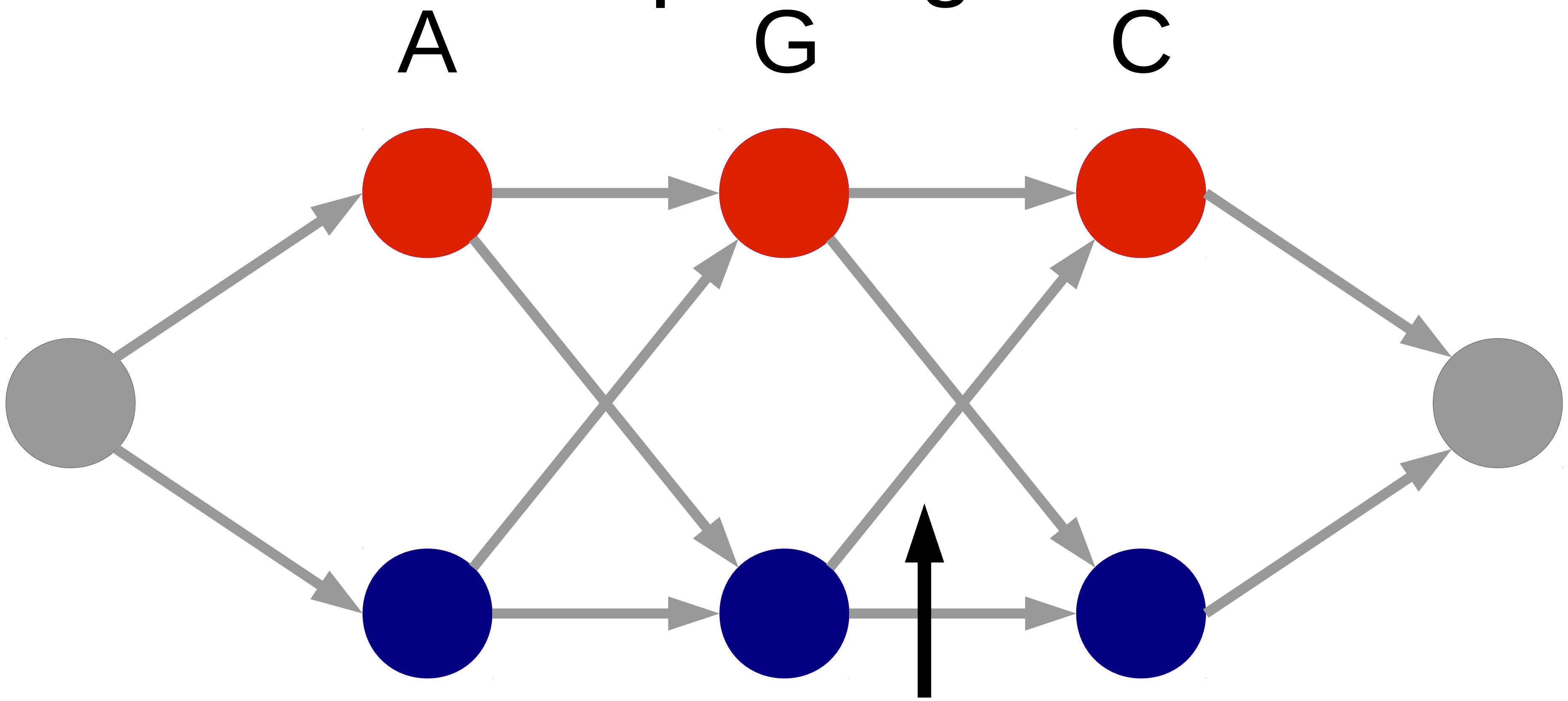


An alternative way to think about updating



Consider the probabilities at each position:

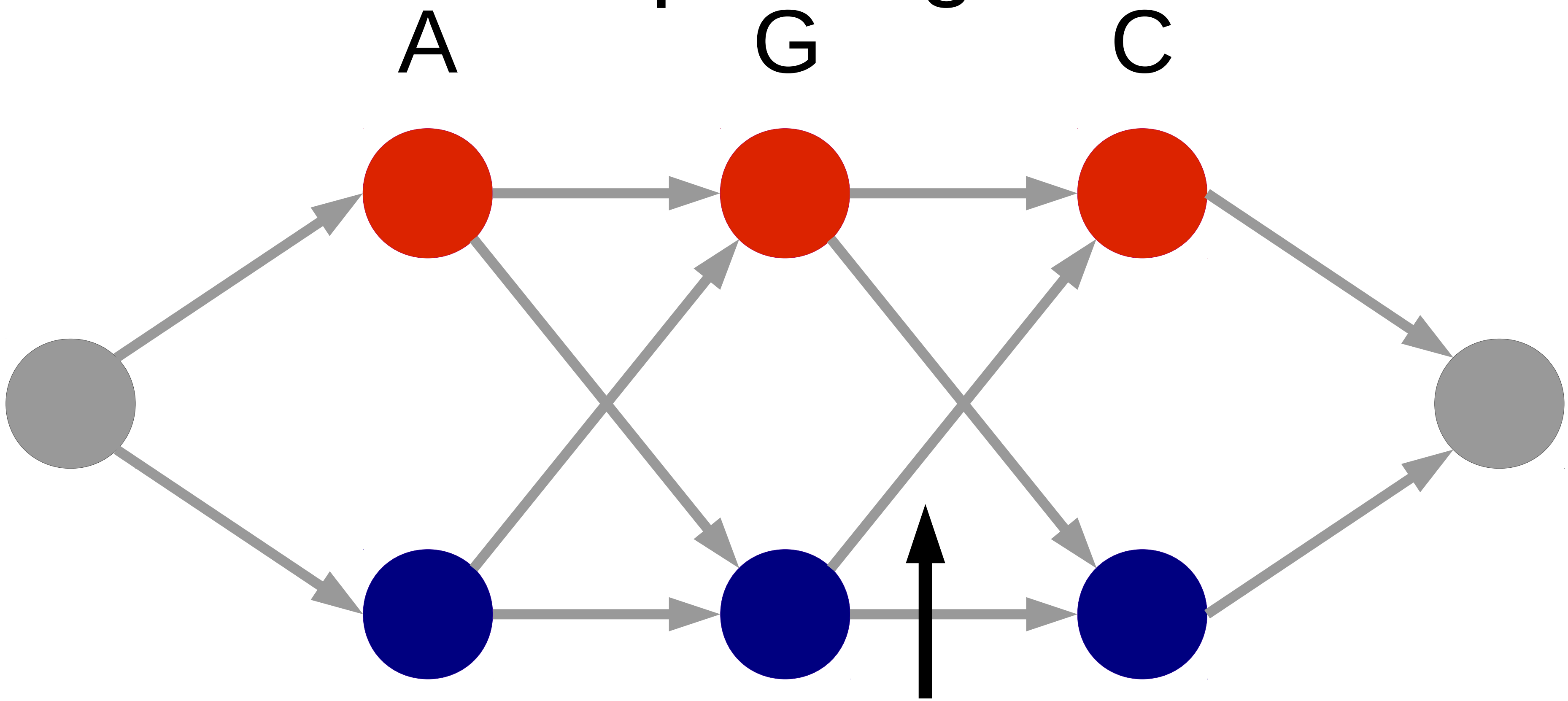
An alternative way to think about updating



Consider the probabilities at each position:

- figure out the probability of going from state k to state l from position i to position i+1

An alternative way to think about updating



Consider the probabilities at each position:

- figure out the probability of going from state *k* to state *l* from position *i* to position *i*+1

$$\xi_{kl}(i) = \frac{\alpha_k(i) e_k(S_i) a_{kl} e_l(S_{i+1}) \beta_l(i+1)}{\sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_m(i) e_m(S_i) a_{mn} e_n(S_{i+1}) \beta_n(i+1)}$$

An alternative way to think about updating

An alternative way to think about updating

- The initial probabilities for each state *k* can be updated to

$$\gamma_k(1)$$

An alternative way to think about updating

- The initial probabilities for each state *k* can be updated to

$$\gamma_k(1)$$

- The transition probability from state *k* to state *l* can be updated to

$$\frac{\sum_{i=1} \xi_{kl}(i)}{\sum_{i=1} \gamma_k(i)}$$

An alternative way to think about updating

- The initial probabilities for each state *k* can be updated to

$$\gamma_k(1)$$

- The transition probability from state *k* to state *l* can be updated to

$$\frac{\sum_{i=1} \xi_{kl}(i)}{\sum_{i=1} \gamma_k(i)}$$

Remember to ignore the last position

An alternative way to think about updating

- The initial probabilities for each state *k* can be updated to

$$\gamma_k(1)$$

- The transition probability from state *k* to state *l* can be updated to

$$\frac{\sum_{i=1} \xi_{kl}(i)}{\sum_{i=1} \gamma_k(i)}$$

Remember to ignore the last position

- The emission probability for symbol *v* from state *k* can be updated to

$$\frac{\sum_{i=1} 1_{S_i=v} \gamma_k(i)}{\sum_{i=1} \gamma_k(i)}$$

An alternative way to think about updating

- The initial probabilities for each state *k* can be updated to

$$\gamma_k(1)$$

- The transition probability from state *k* to state *l* can be updated to

$$\frac{\sum_{i=1} \xi_{kl}(i)}{\sum_{i=1} \gamma_k(i)}$$

Remember to ignore the last position

- The emission probability for symbol *v* from state *k* can be updated to

$$\frac{\sum_{i=1} 1_{S_i=v}\gamma_k(i)}{\sum_{i=1} \gamma_k(i)}$$

$$1_{S_i=v}=\begin{cases}1 & \text{if } S_i=v\\ 0 & \text{otherwise}\end{cases}$$

Notes for debugging

1) Try calculating some simple forward and backward probabilities by hand to check your algorithm

2) Make sure the sum of the numerators for a single state or transition from a given state equals the associated denominator

3) The likelihood at each iteration should increase, if it decreases then you have a bug

Formal definition of P, NP, and NP-hard

Formal definition of P, NP, and NP-hard

- P: The set of all problems such that you can find a solution in polynomial time

Formal definition of P, NP, and NP-hard

- P: The set of all problems such that you can find a solution in polynomial time

- NP: The set of all problems such that you can verify a solution is correct in polynomial time

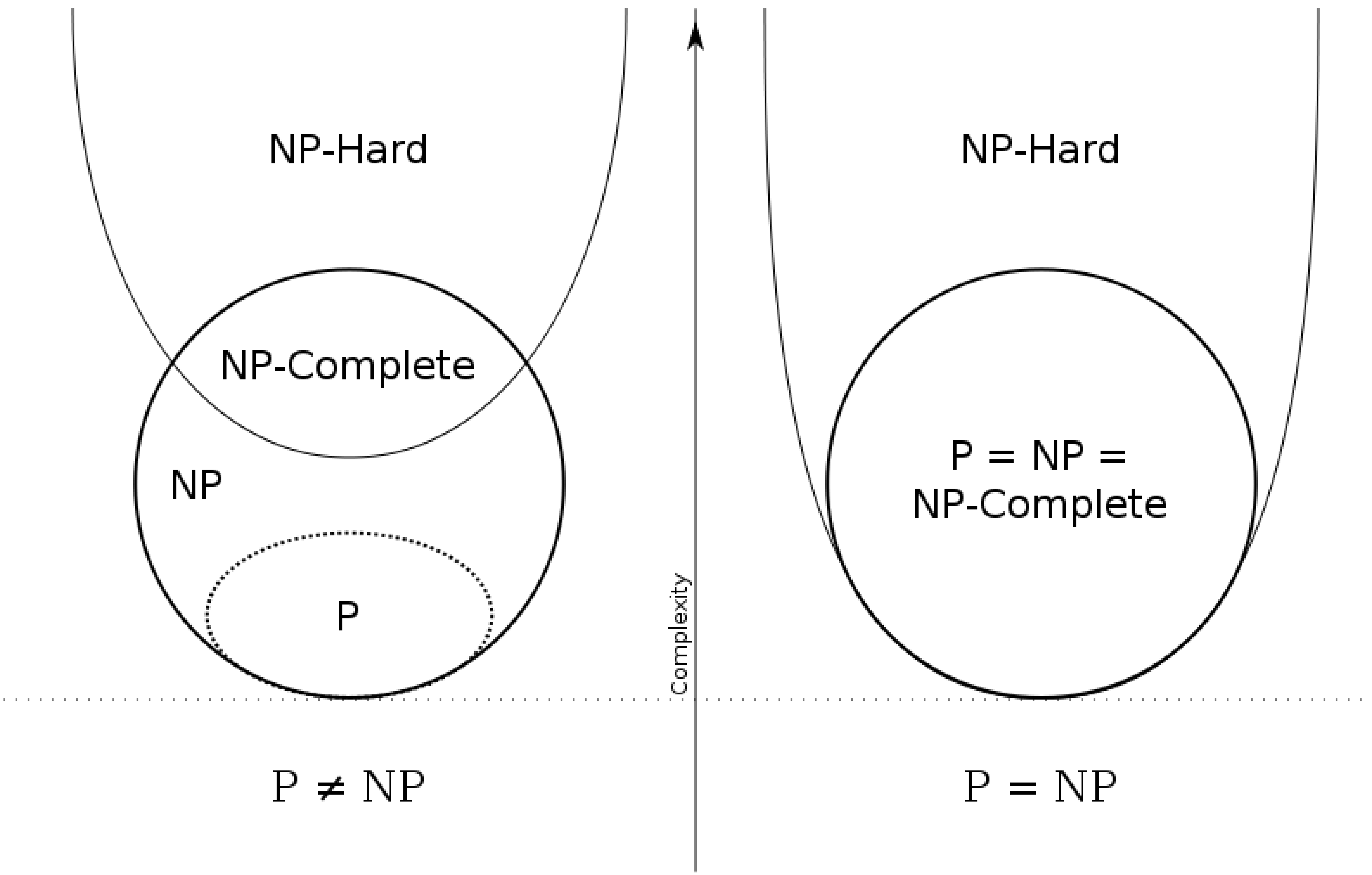
Formal definition of P, NP, and NP-hard

- P: The set of all problems such that you can find a solution in polynomial time
- NP: The set of all problems such that you can verify a solution is correct in polynomial time
- NP-hard: The set of all problems that can be reduced to the hardest NP problem

Formal definition of P, NP, and NP-hard

- P: The set of all problems such that you can find a solution in polynomial time
- NP: The set of all problems such that you can verify a solution is correct in polynomial time
- NP-hard: The set of all problems that can be reduced to the hardest NP problem
- Open question: Does P = NP?

Formal definition of P, NP, and NP-hard



How to prove that a problem is NP-complete

How to prove that a problem is NP-complete

- NP-complete problems are NP problems that are also NP-hard

How to prove that a problem is NP-complete

- NP-complete problems are NP problems that are also NP-hard

- By proving a problem is NP complete, you prove that it is at least as difficult as any known NP-complete problem

How to prove that a problem is NP-complete

- NP-complete problems are NP problems that are also NP-hard
- By proving a problem is NP complete, you prove that it is at least as difficult as any known NP-complete problem
- This doesn't necessarily mean you should give up, approximate P algorithms may exist for your NP problem

How to prove that a problem is NP-complete

How to prove that a problem is NP-complete

- There are four steps to an NP-completeness proof

How to prove that a problem is NP-complete

- There are four steps to an NP-completeness proof

1) Prove the problem is in NP

How to prove that a problem is NP-complete

- There are four steps to an NP-completeness proof

  1) Prove the problem is in NP

  2) Construct an algorithm to transform a known NP-complete problem into your problem

How to prove that a problem is NP-complete

- There are four steps to an NP-completeness proof

  1) Prove the problem is in NP

  2) Construct an algorithm to transform a known NP-complete problem into your problem

  3) Prove that solutions to your problem are correct if and only if they are solutions to the reduced NP-complete problem

How to prove that a problem is NP-complete

- There are four steps to an NP-completeness proof

1) Prove the problem is in NP

2) Construct an algorithm to transform a known NP-complete problem into your problem

3) Prove that solutions to your problem are correct if and only if they are solutions to the reduced NP-complete problem

4) Prove your reduction algorithm is in P

How to prove that a problem is NP-complete

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from
  - There's even a book that lists them and their proofs

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from
  - There's even a book that lists them and their proofs
- Some examples:

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from
  - There's even a book that lists them and their proofs
- Some examples:
  - 3SAT (the canonical example)

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from
  - There's even a book that lists them and their proofs
- Some examples:
  - 3SAT (the canonical example)
  - Set cover

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from
  - There's even a book that lists them and their proofs
- Some examples:
  - 3SAT (the canonical example)
  - Set cover
  - Knapsack problem

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from
  - There's even a book that lists them and their proofs
- Some examples:
  - 3SAT (the canonical example)
  - Set cover
  - Knapsack problem
  - Traveling salesman

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from
  - There's even a book that lists them and their proofs
- Some examples:
  - 3SAT (the canonical example)
  - Set cover
  - Knapsack problem
  - Traveling salesman
  - Bejeweled/Candy Crush (there's a paper on arXiv)

How to prove that a problem is NP-complete

- There are many problems that have been proven to be NP-complete that you can select from
  - There's even a book that lists them and their proofs
- Some examples:
  - 3SAT (the canonical example)
  - Set cover
  - Knapsack problem
  - Traveling salesman
  - Bejeweled/Candy Crush (there's a paper on arXiv)
  - Classic Nintendo games (again, check out arXiv)