An Efficient Parallel Algorithm for Accelerating Computational Protein Design

Yichao Zhou [1] Wei Xu [1] Bruce R. Donald [2] Jianyang Zeng [1,*]

[1]Institute for Interdisciplinary Information Sciences
Tsinghua University

[2]Department of Computer Science, Department of Biochemistry
Duke University

Jul, 2014

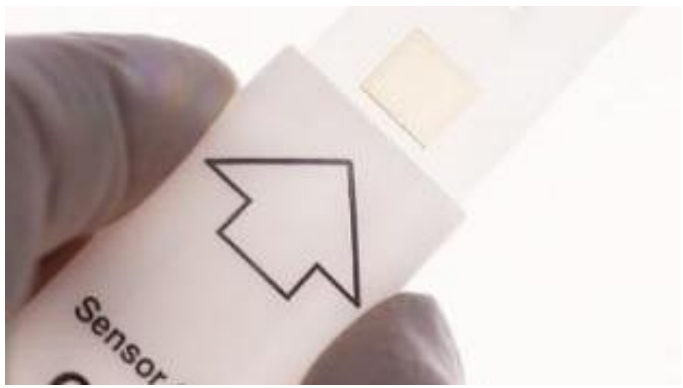
Why We Need Protein Design?

Applications

New Drug Discovery

Drug Resistance Prediction

Enzyme Optimization

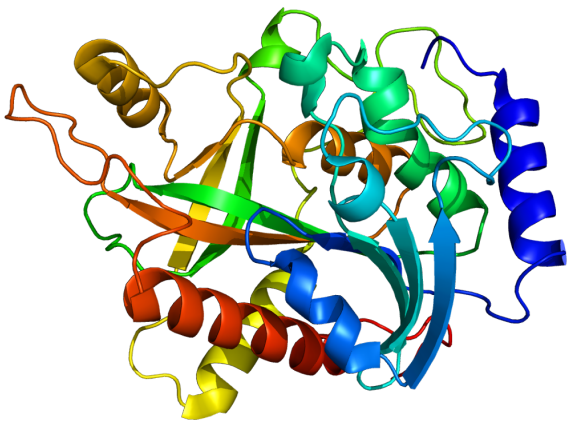


New Biosensor Design

What is Structure-Based Protein Design?

Toolbox

- Energy Functions
- Rotamer Library
- Backbone Template



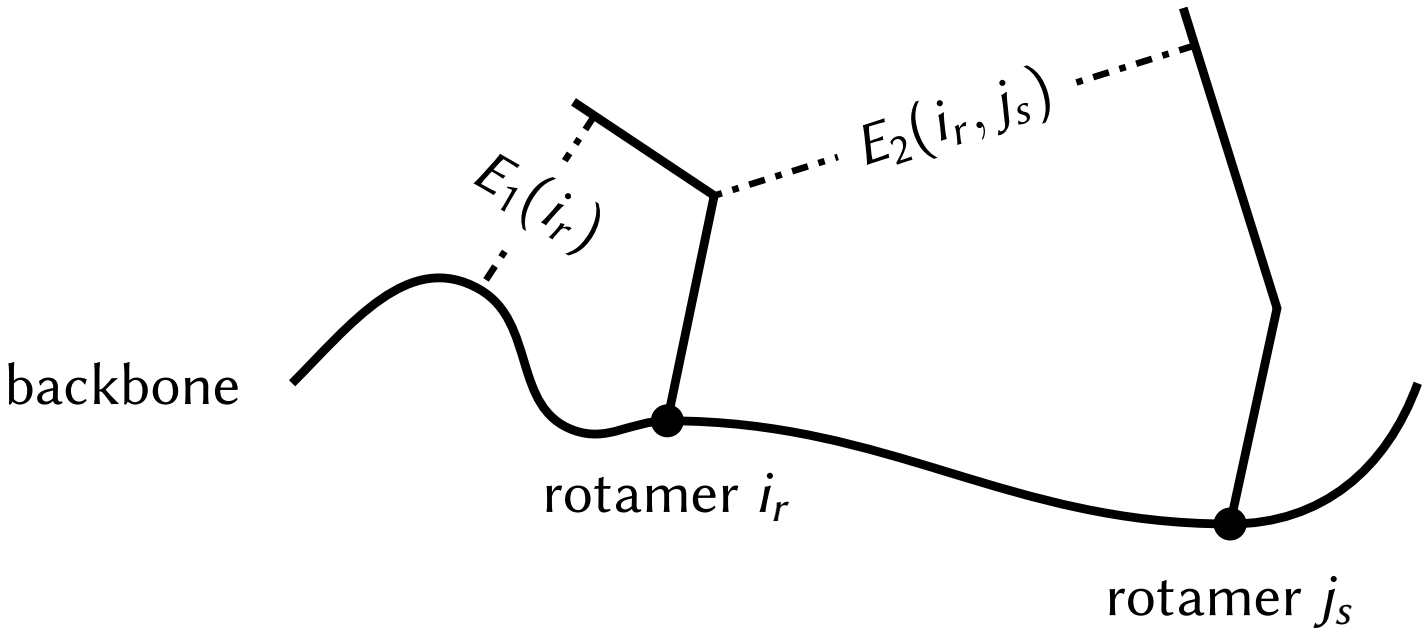
Wild-type protein's structure

Design Algorithm

· · ·	Ile	Ser	Ile
Pro	Ala	Ile	Trp
Val	Gly	Ser	Asp
Gly	Gly	Pro	Glu
Ile	Pro	His	· · ·

1D amino acid sequence

Protein Design as an Optimization Problem



Energy Function (Optimization Target)

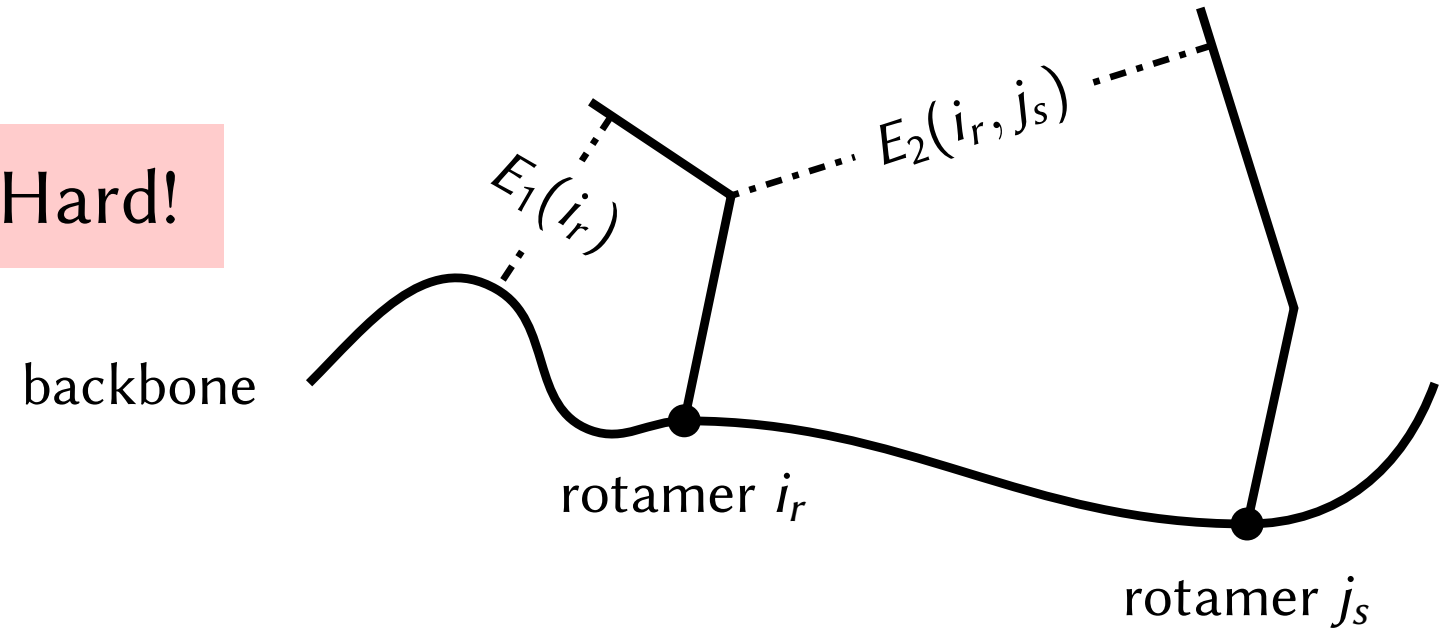
$$E_T = \sum_{i_r} E_1(i_r) + \sum_{i_r} \sum_{j_s, i<j} E_2(i_r, j_s)$$

Self energy of rotamer i_r

Pairwise energy between rotamer i_r and j_s

Protein Design as an Optimization Problem

NP Hard!



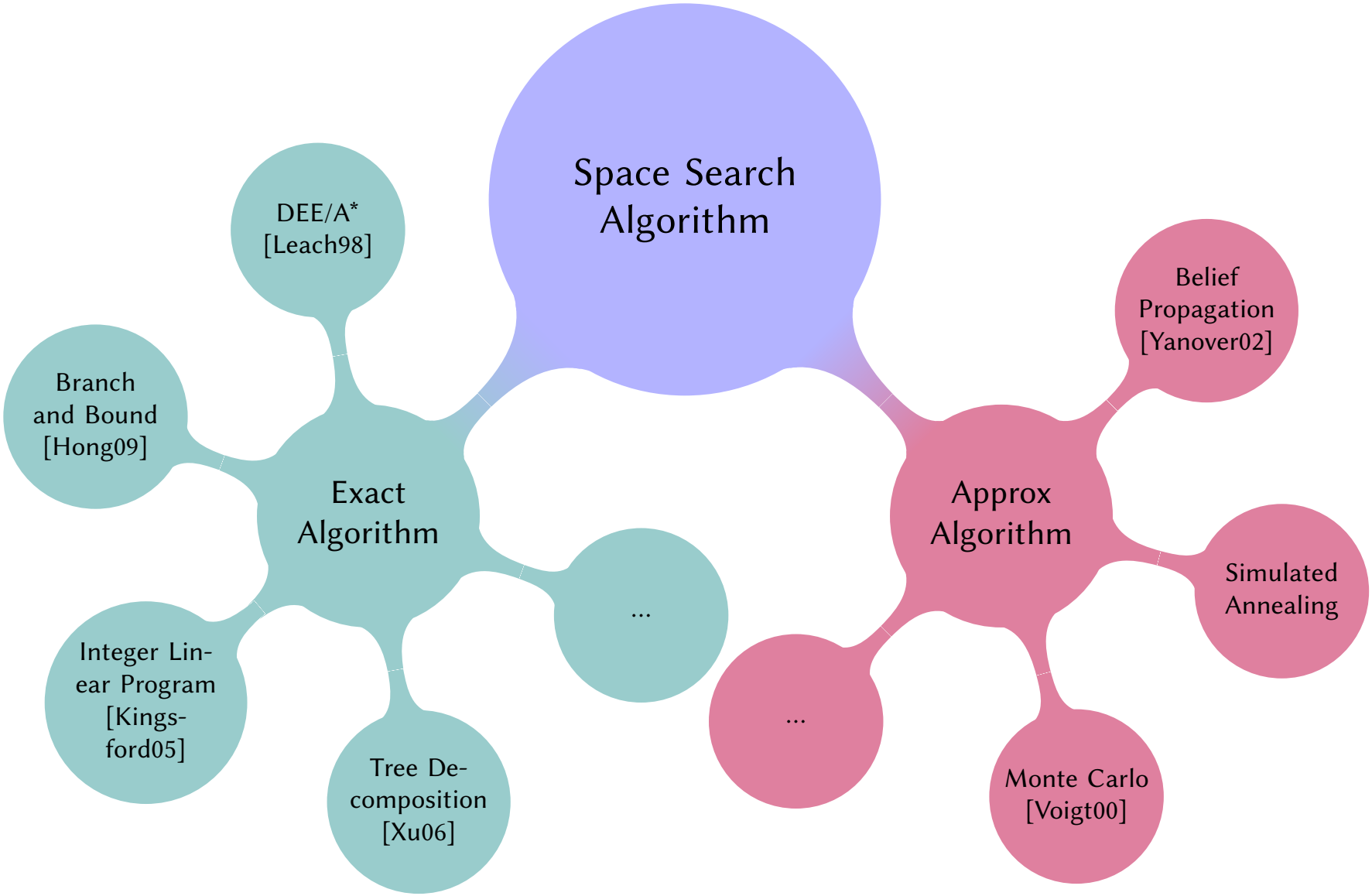
Energy Function (Optimization Target)

$$E_T = \sum_{i_r} E_1(i_r) + \sum_{i_r} \sum_{j_s, i<j} E_2(i_r, j_s)$$

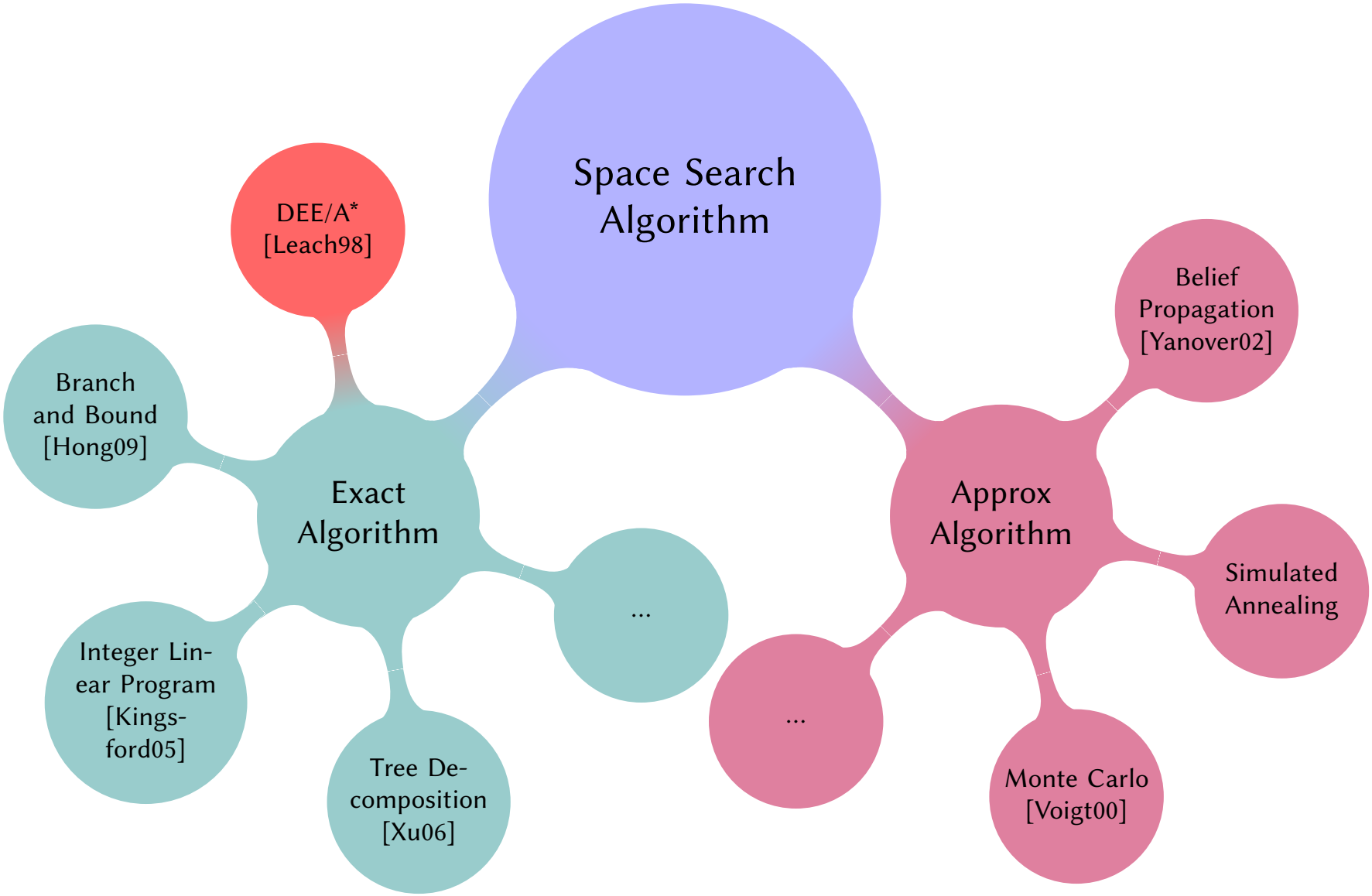
Self energy of rotamer i_r

Pairwise energy between rotamer i_r and j_s

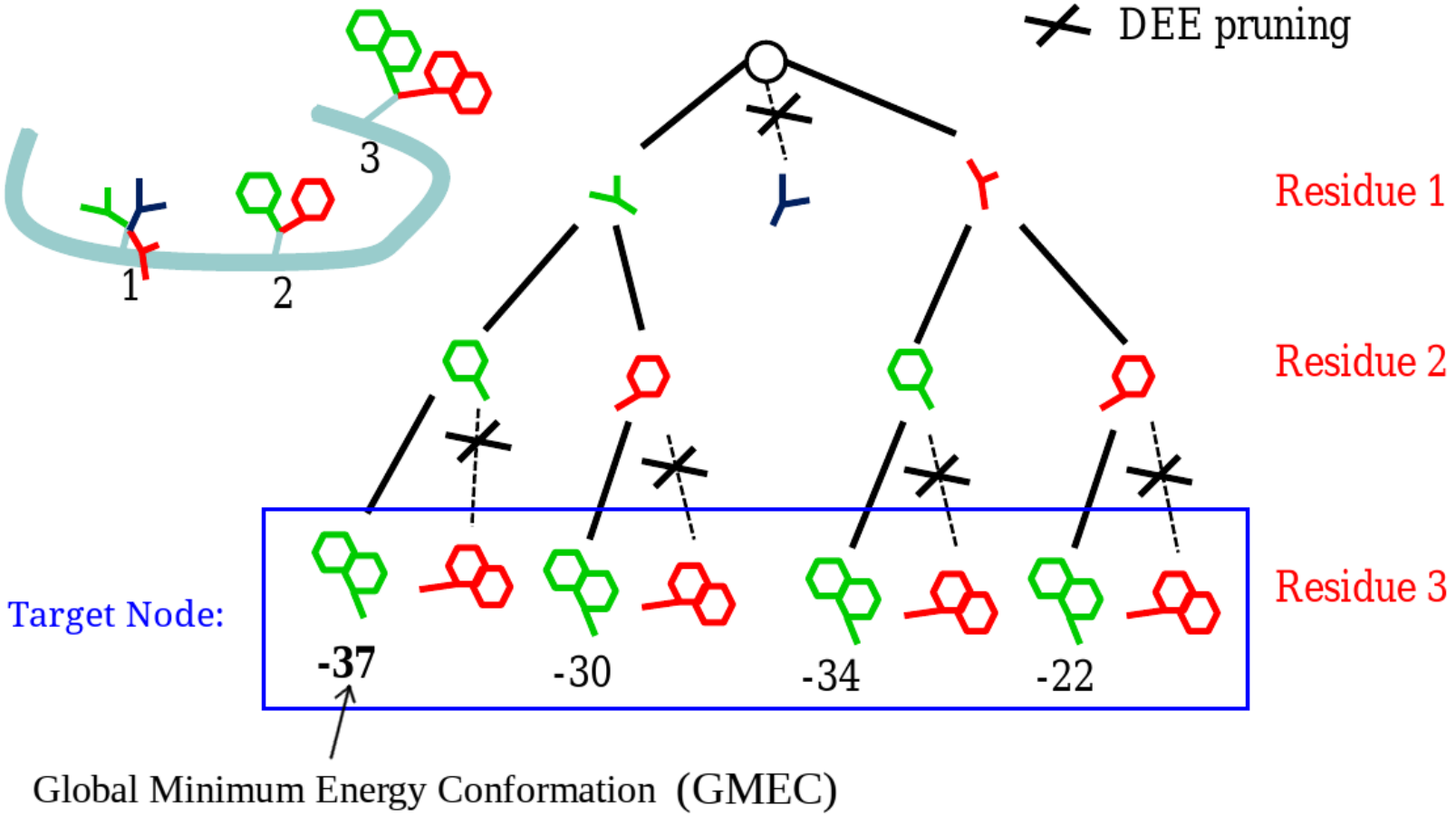
Search Algorithm



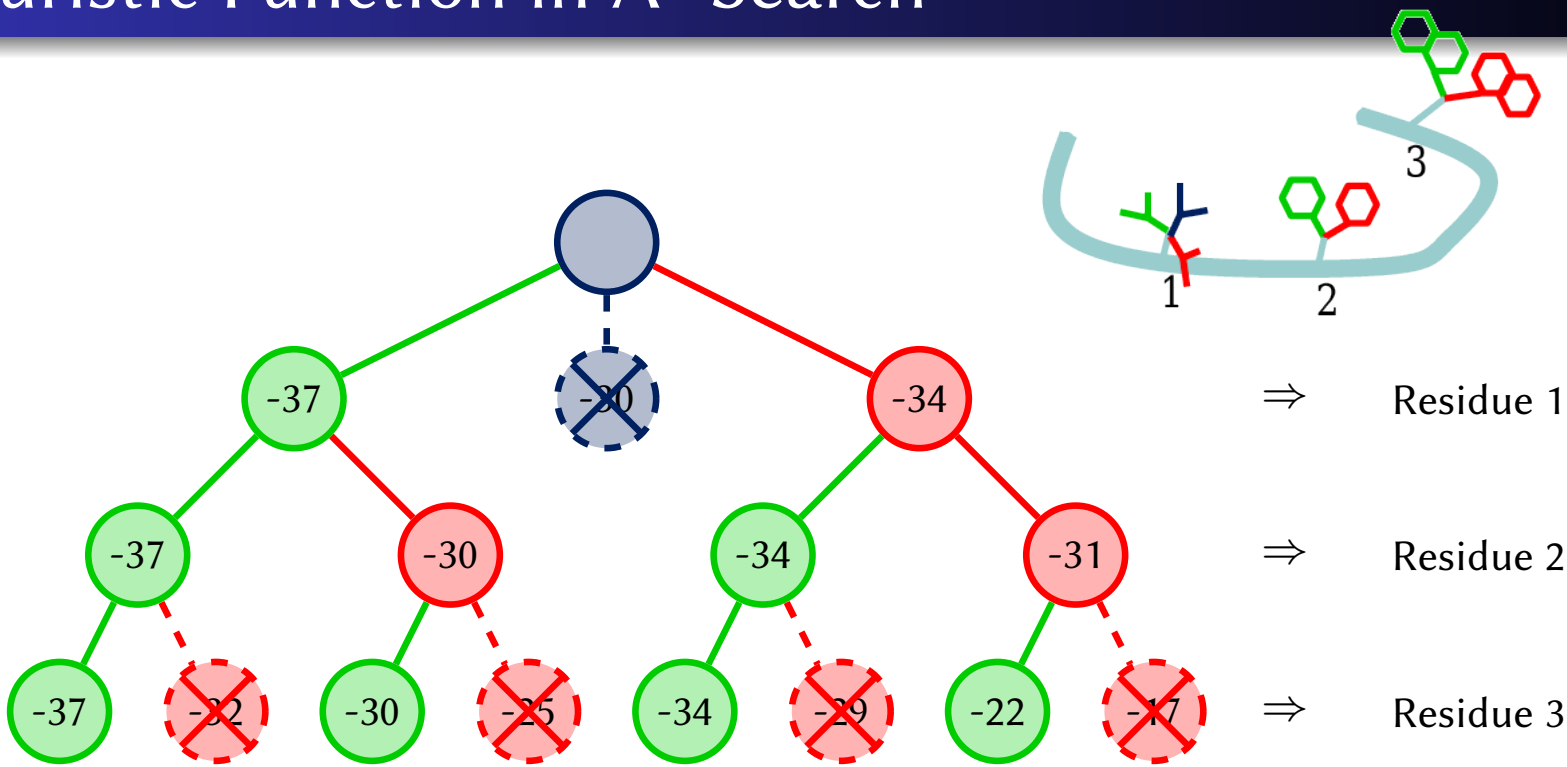
Search Algorithm



Dead End Elimination/A* Algorithm



Heuristic Function in A* Search

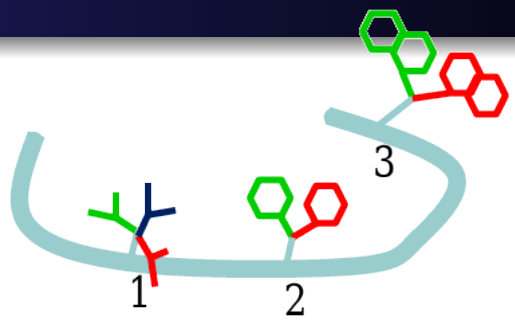


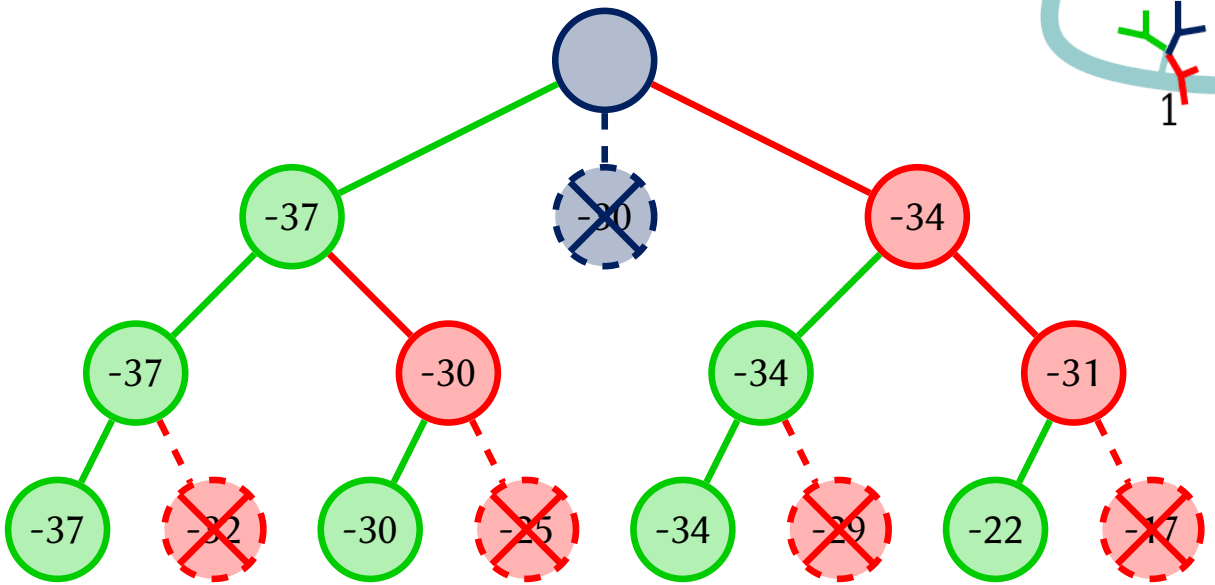
$$f(x) = g(x) + h(x)$$

$$g(x) = \sum_{i_r \in D(x)} E_1(i_r) + \sum_{i_r \in D(x)} \sum_{\substack{j_s \in D(x), \\ i<j}} E_2(i_r, j_s)$$

$$h(x) = \sum_{i \in U(x)} \min_r \left(E_1(i_r) + \sum_{j_s \in D(x)} E_2(i_r, j_s) + \sum_{k \in U(x)} \min_u E_2(i_r, k_u) \right)$$

Heuristic Function in A* Search





⇒ Residue 1

⇒ Residue 2

⇒ Residue 3

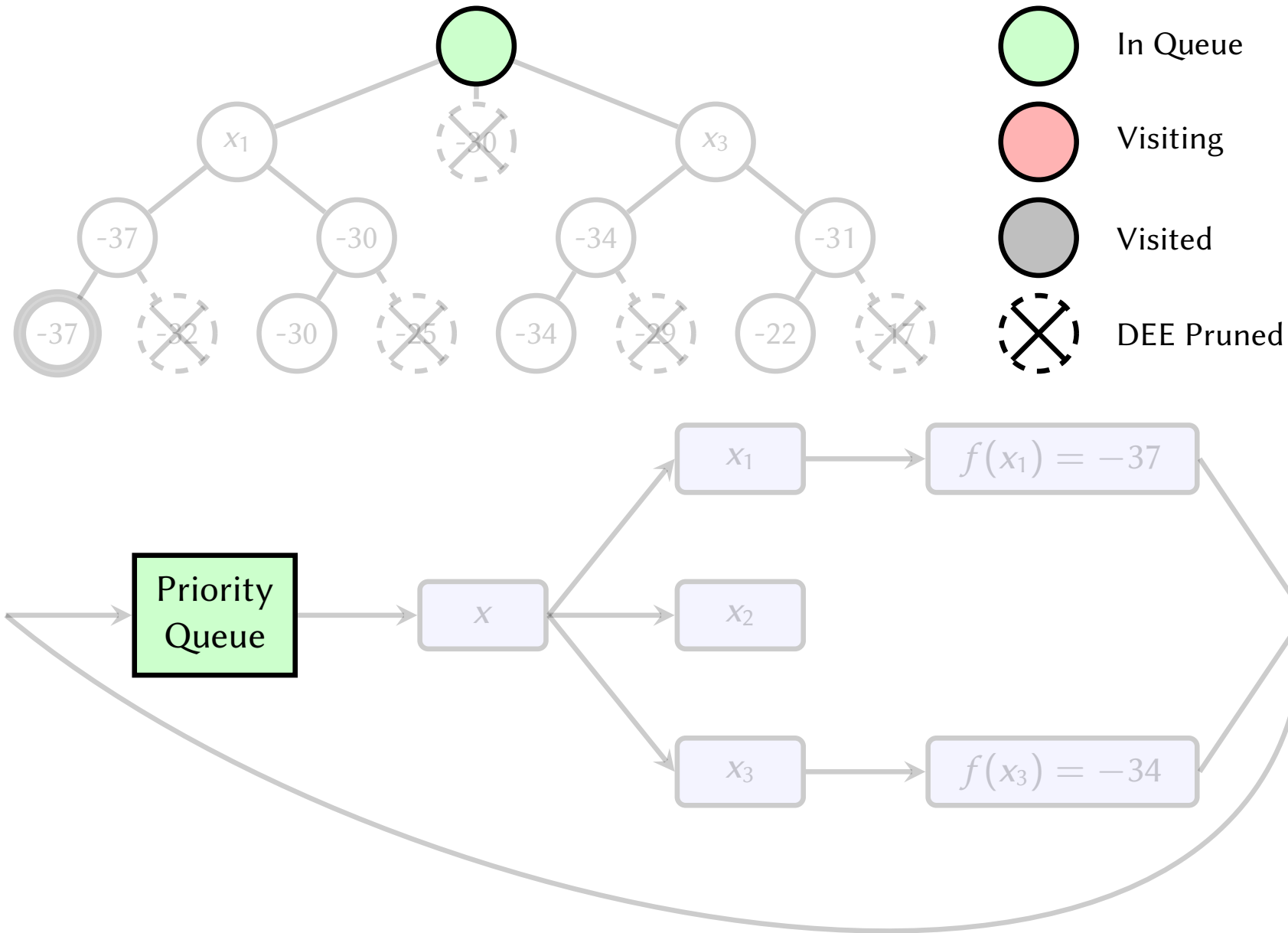
$$f(x) = g(x) + h(x) \qquad \Longrightarrow$$

Admissible

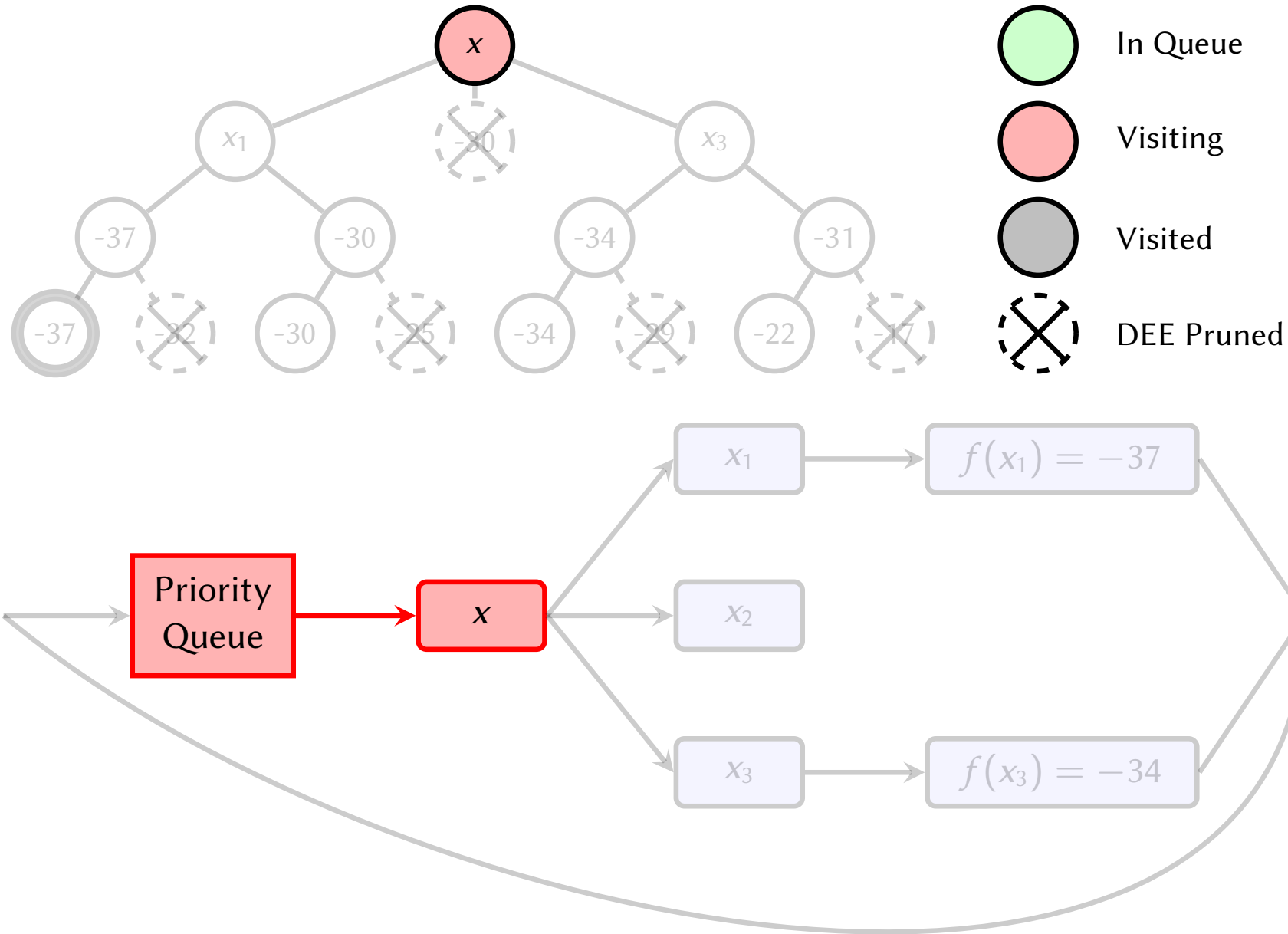
$$g(x) = \sum_{i_r \in D(x)} E_1(i_r) + \sum_{i_r \in D(x)} \sum_{\substack{j_s \in D(x), \\ i<j}} E_2(i_r, j_s)$$

$$h(x) = \sum_{i \in U(x)} \min_r \left(E_1(i_r) + \sum_{j_s \in D(x)} E_2(i_r, j_s) + \sum_{k \in U(x)} \min_u E_2(i_r, k_u) \right)$$

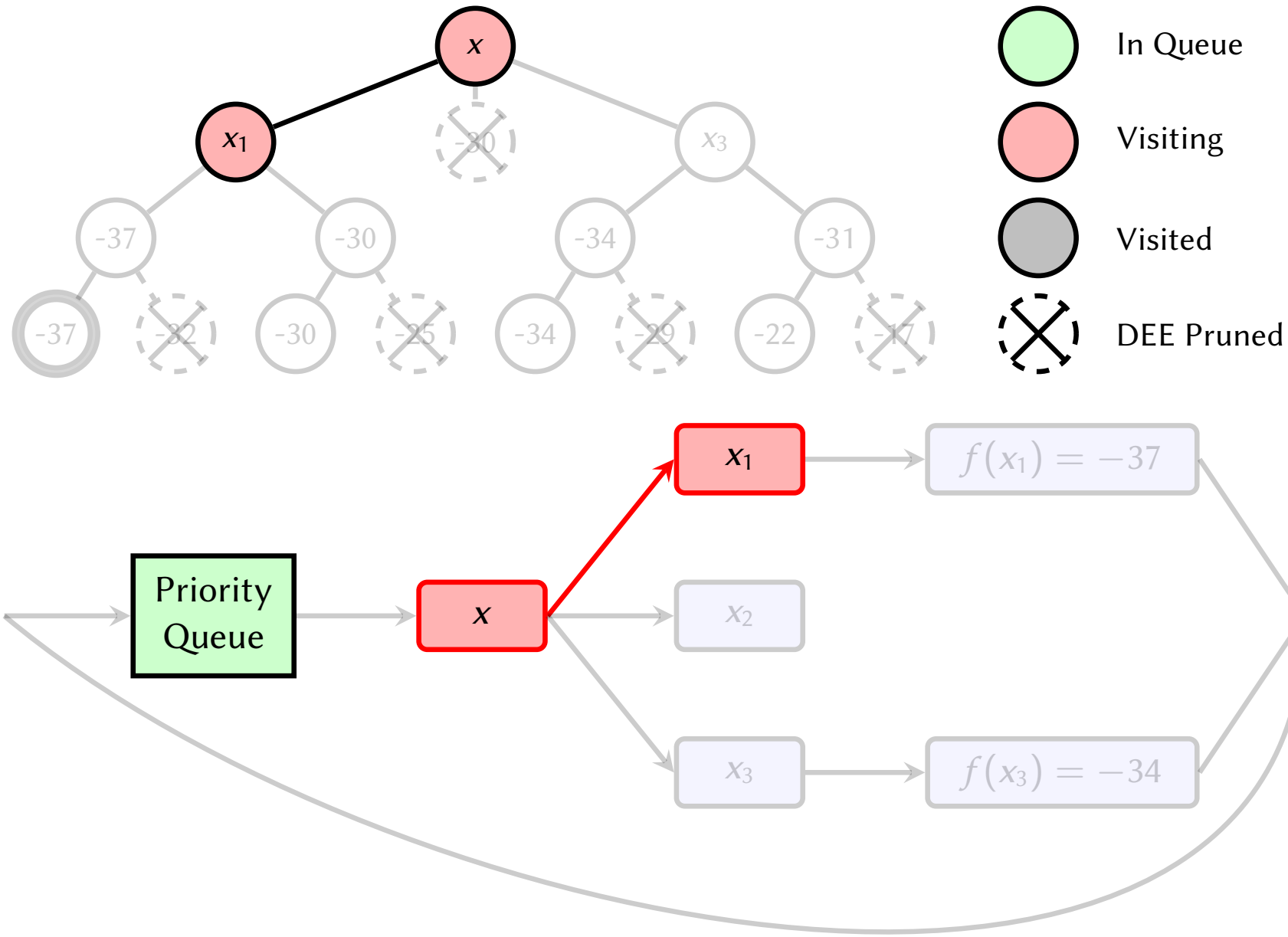
A* Search


x_1
-30
x_3
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x_1
x_2
x_3
$f(x_1) = -37$
$f(x_3) = -34$

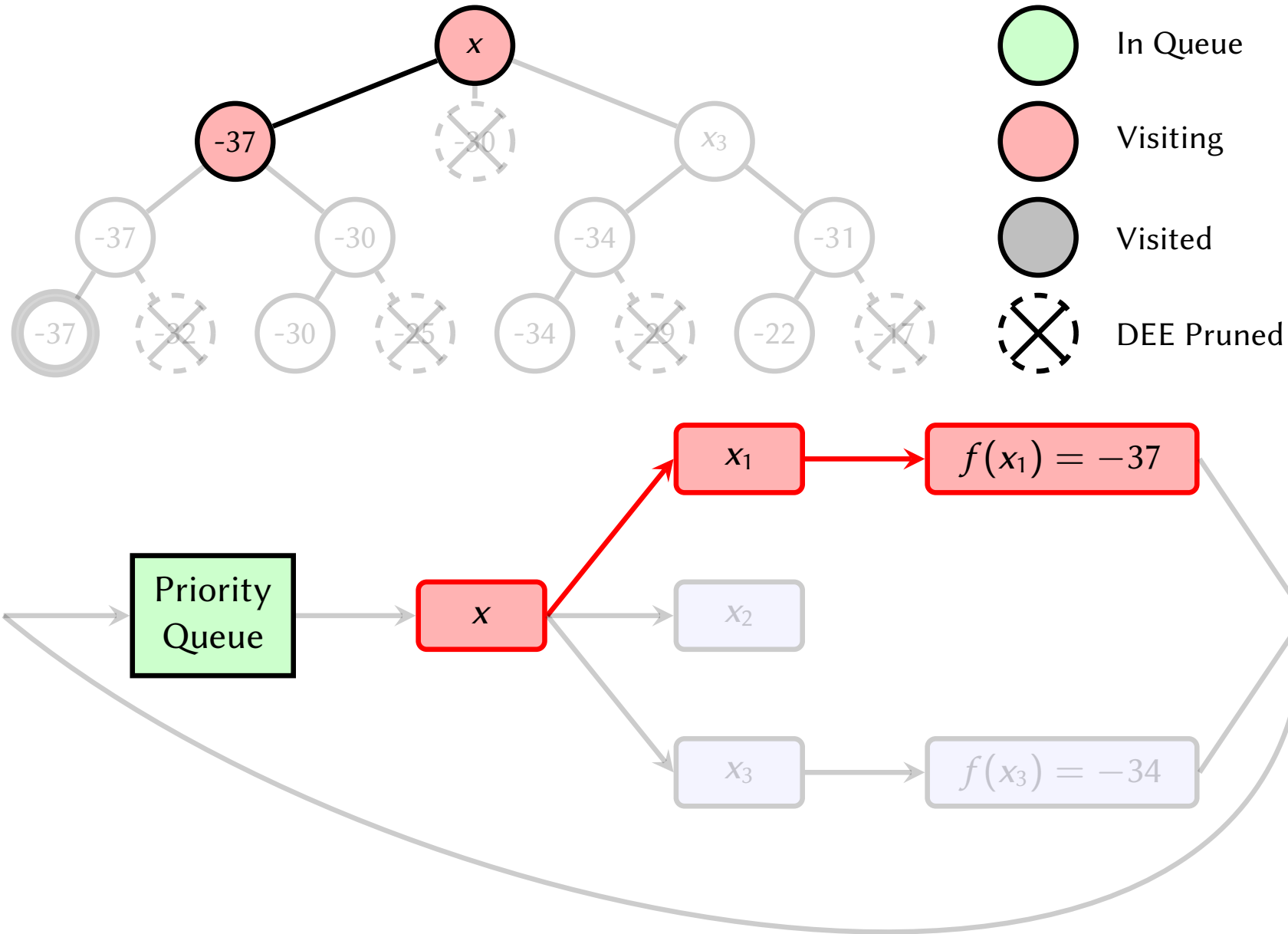
A* Search


x
x1
-30
x3
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x1
x2
x3
f(x1) = −37
f(x3) = −34

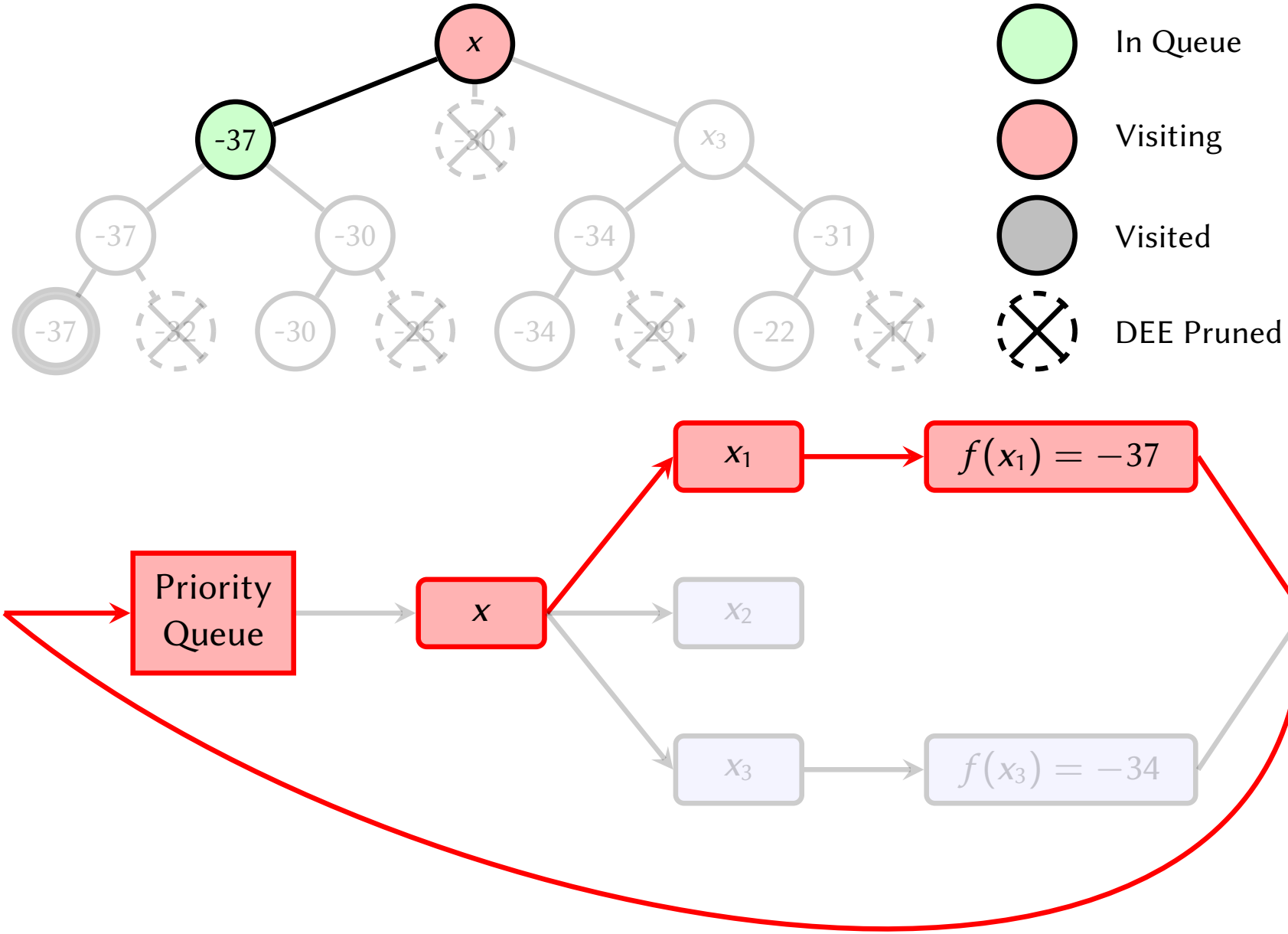
A* Search


x
x1
-30
x3
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x1
x2
x3
f(x1) = −37
f(x3) = −34

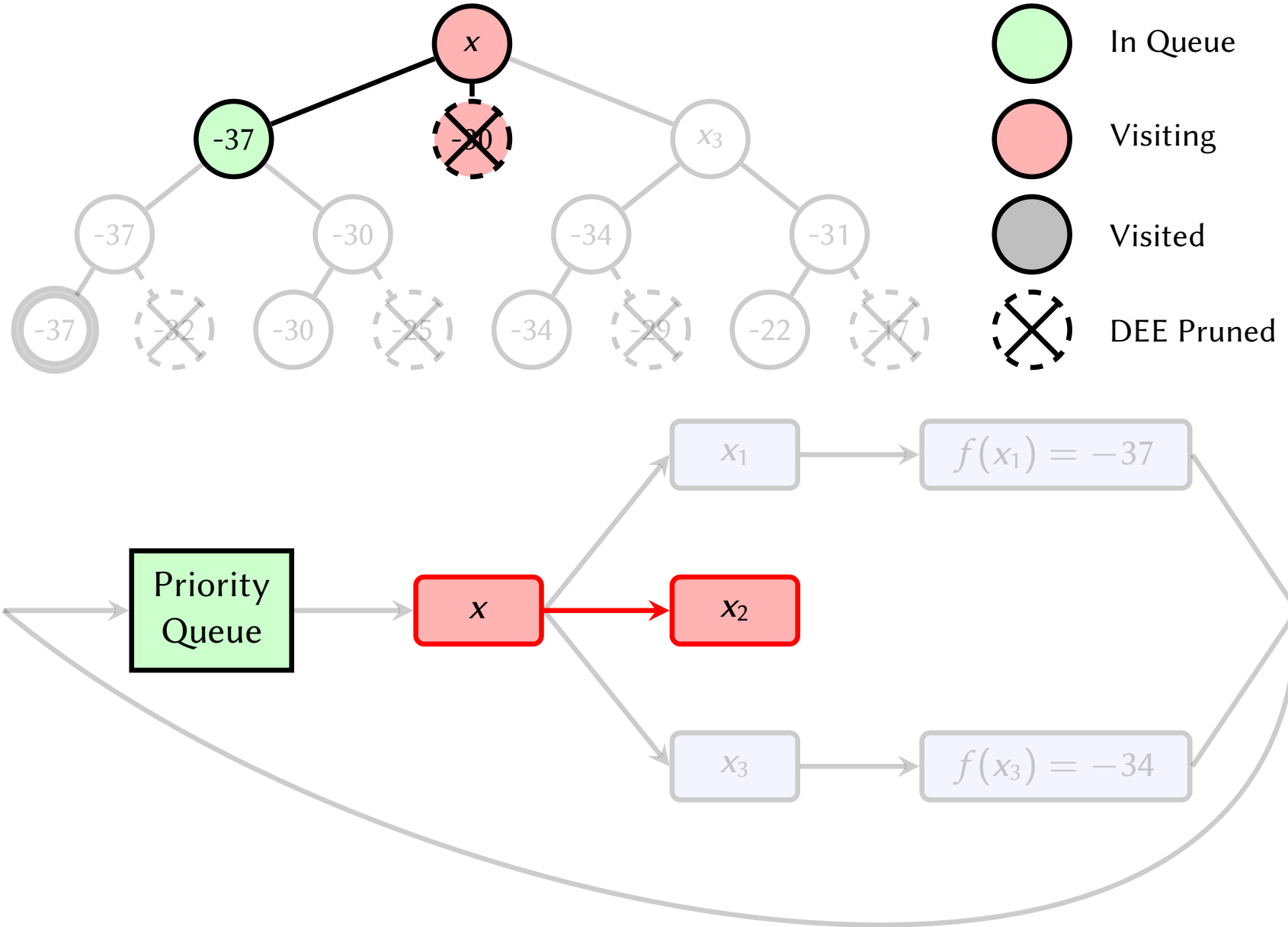
A* Search


x
-37
-30
x3
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x1
x2
x3
f(x1) = −37
f(x3) = −34

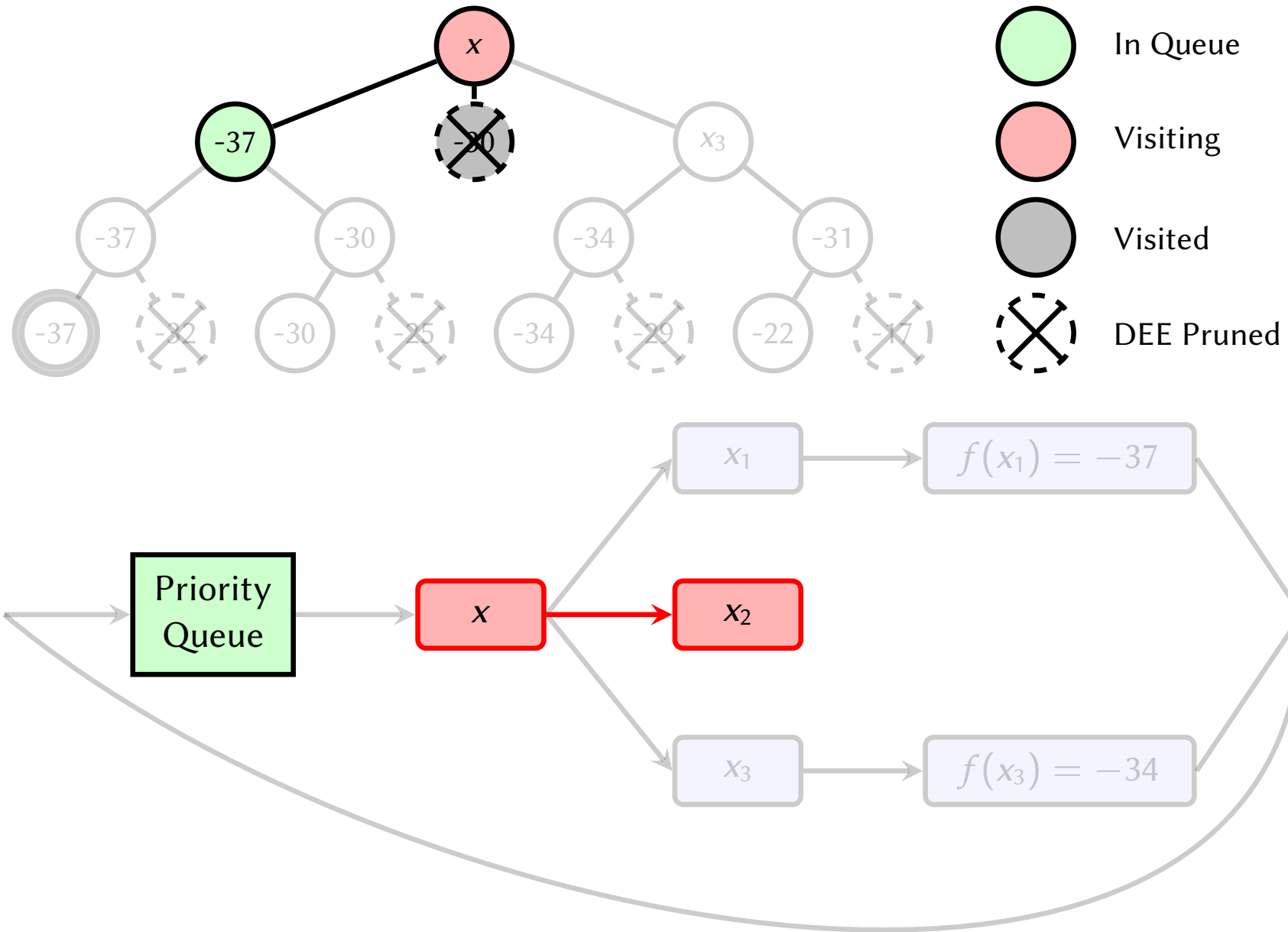
A* Search


x
-37
-30
x3
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x1
x2
x3
f(x1) = −37
f(x3) = −34

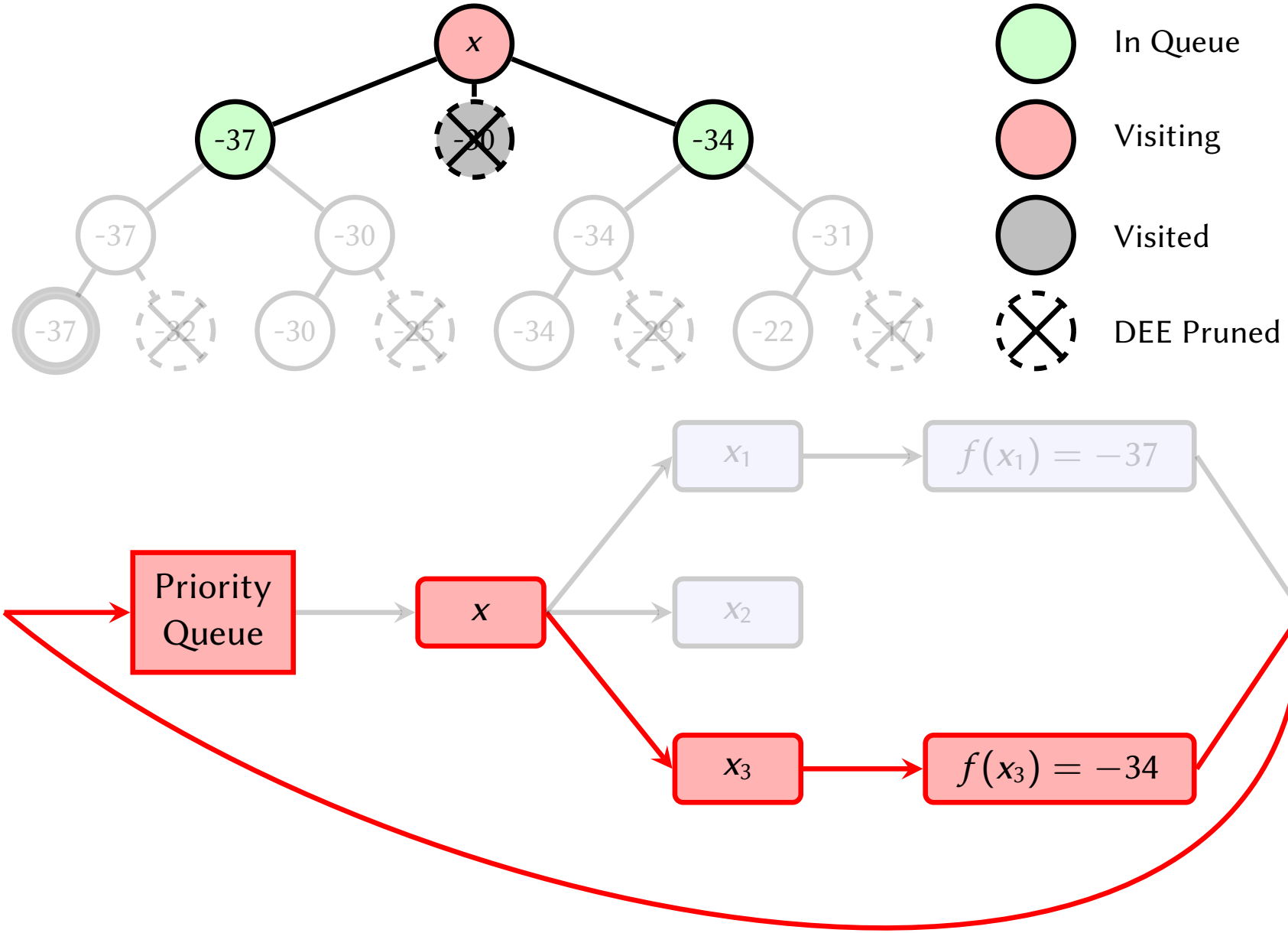
A* Search


x
-37
-30
x3
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x1
x2
x3
f(x1) = −37
f(x3) = −34

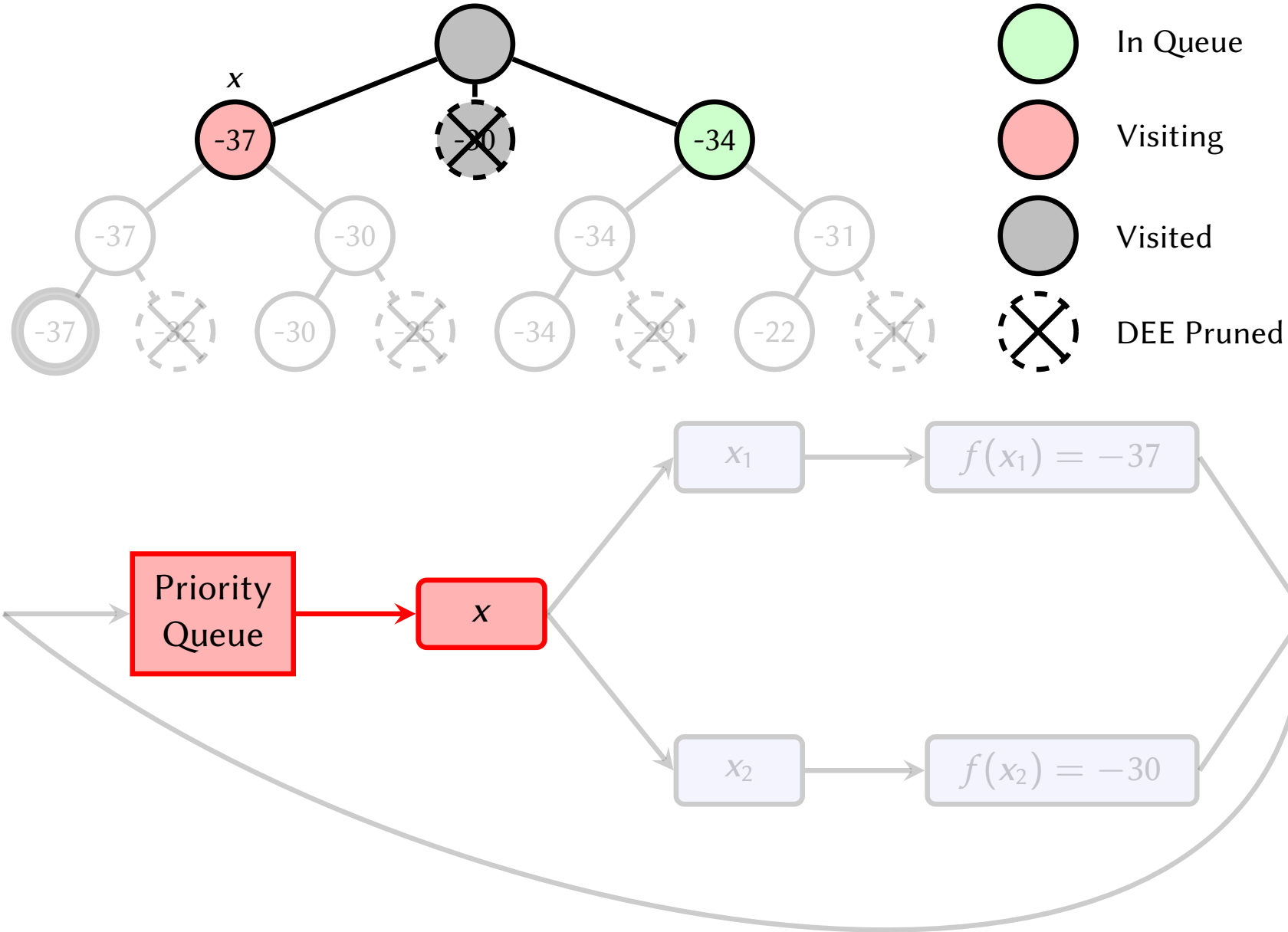
A* Search


x
-37
-30
x_3
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x_1
x_2
x_3
$f(x_1) = -37$
$f(x_3) = -34$

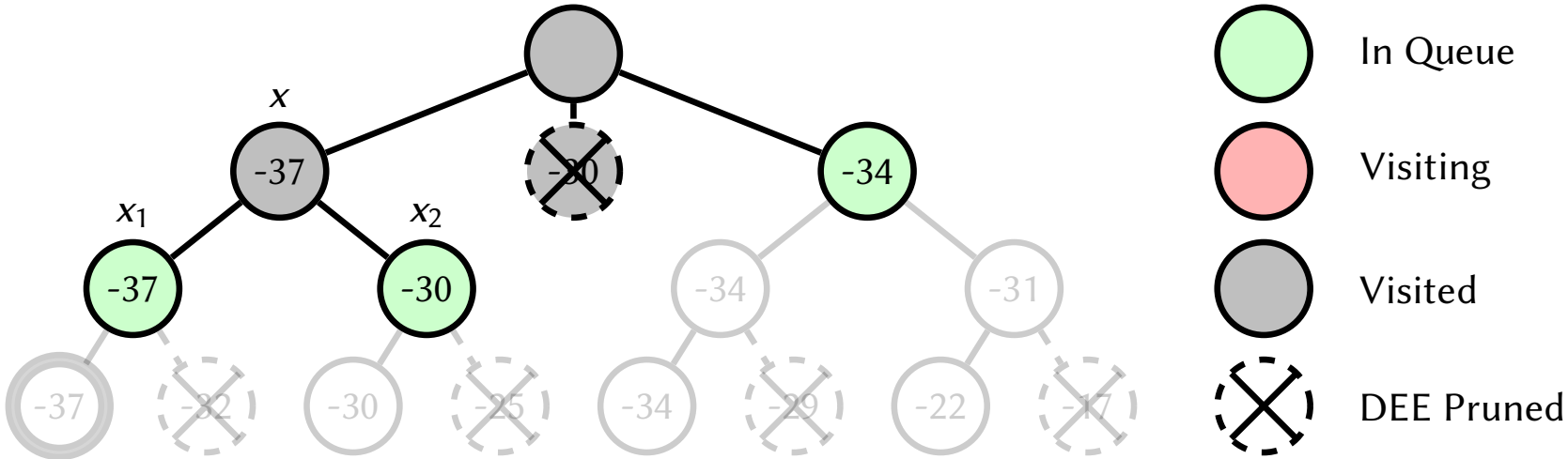
A* Search

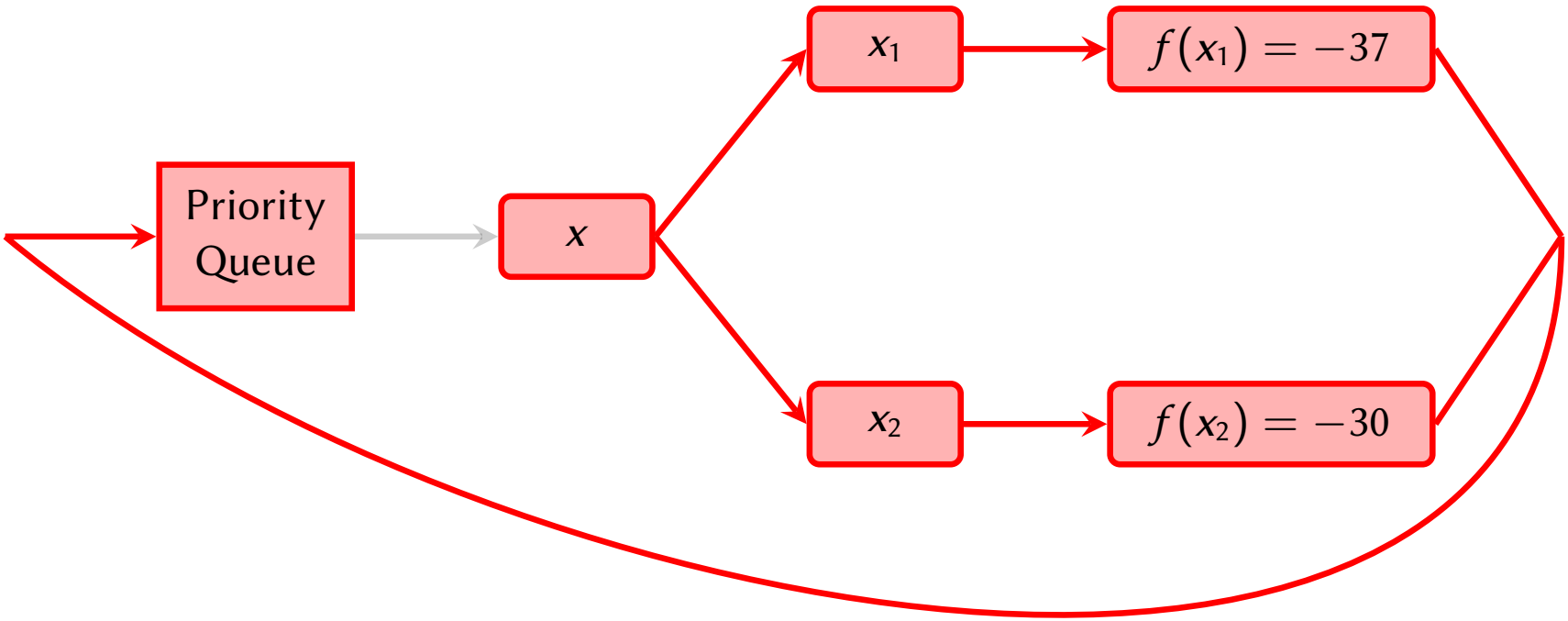

x
-37
-30
-34
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x_1
x_2
x_3
$f(x_1) = -37$
$f(x_3) = -34$

A* Search

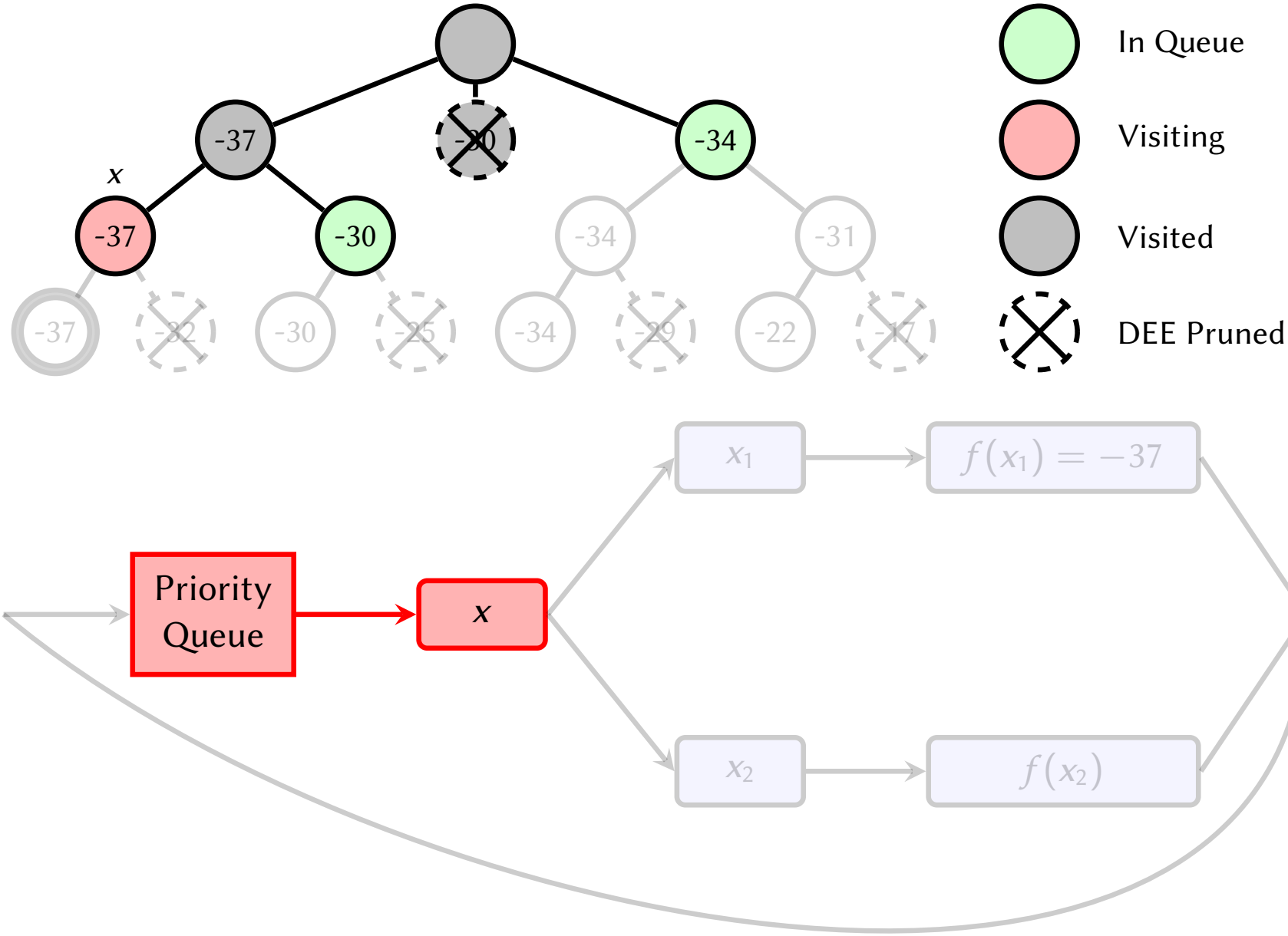

x
-37
-30
-34
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x_1
$f(x_1) = -37$
x_2
$f(x_2) = -30$

A* Search

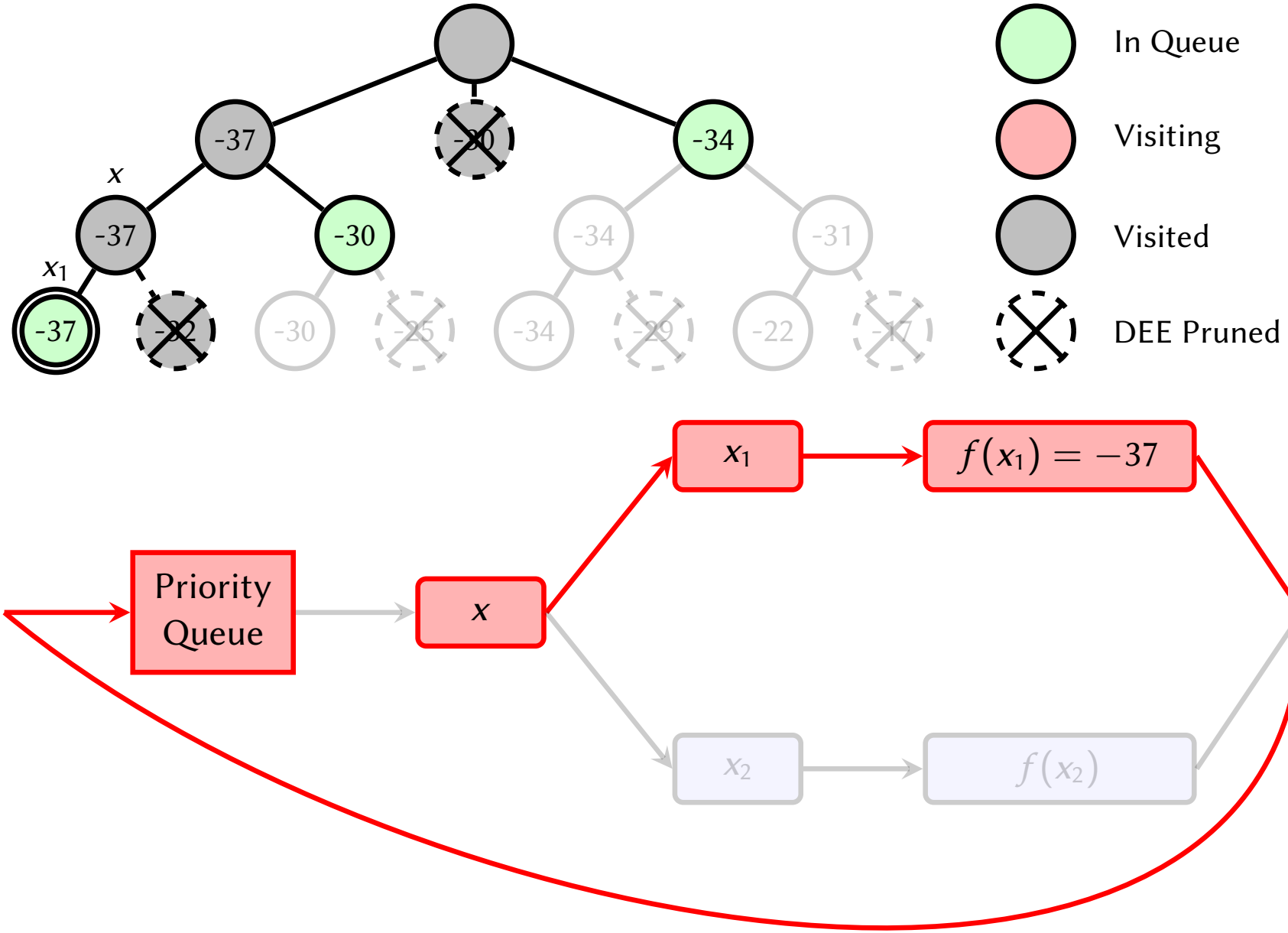

x
-37
-30
-34
x_1
x_2
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned


Priority Queue
x
x_1
x_2
$f(x_1) = -37$
$f(x_2) = -30$

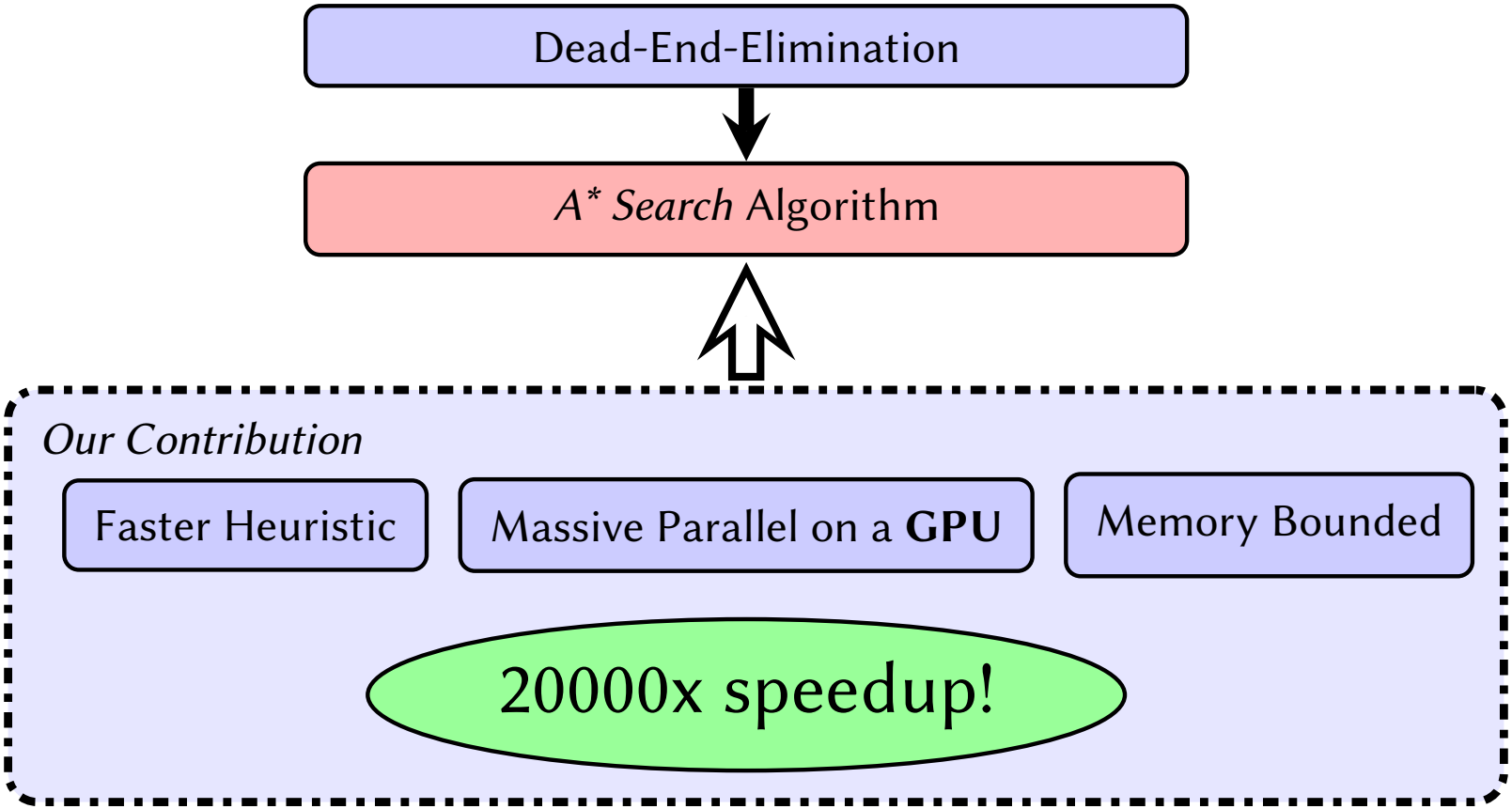
A* Search


-37
-30
-34
x
-37
-30
-34
-31
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x_1
$f(x_1) = -37$
x_2
$f(x_2)$

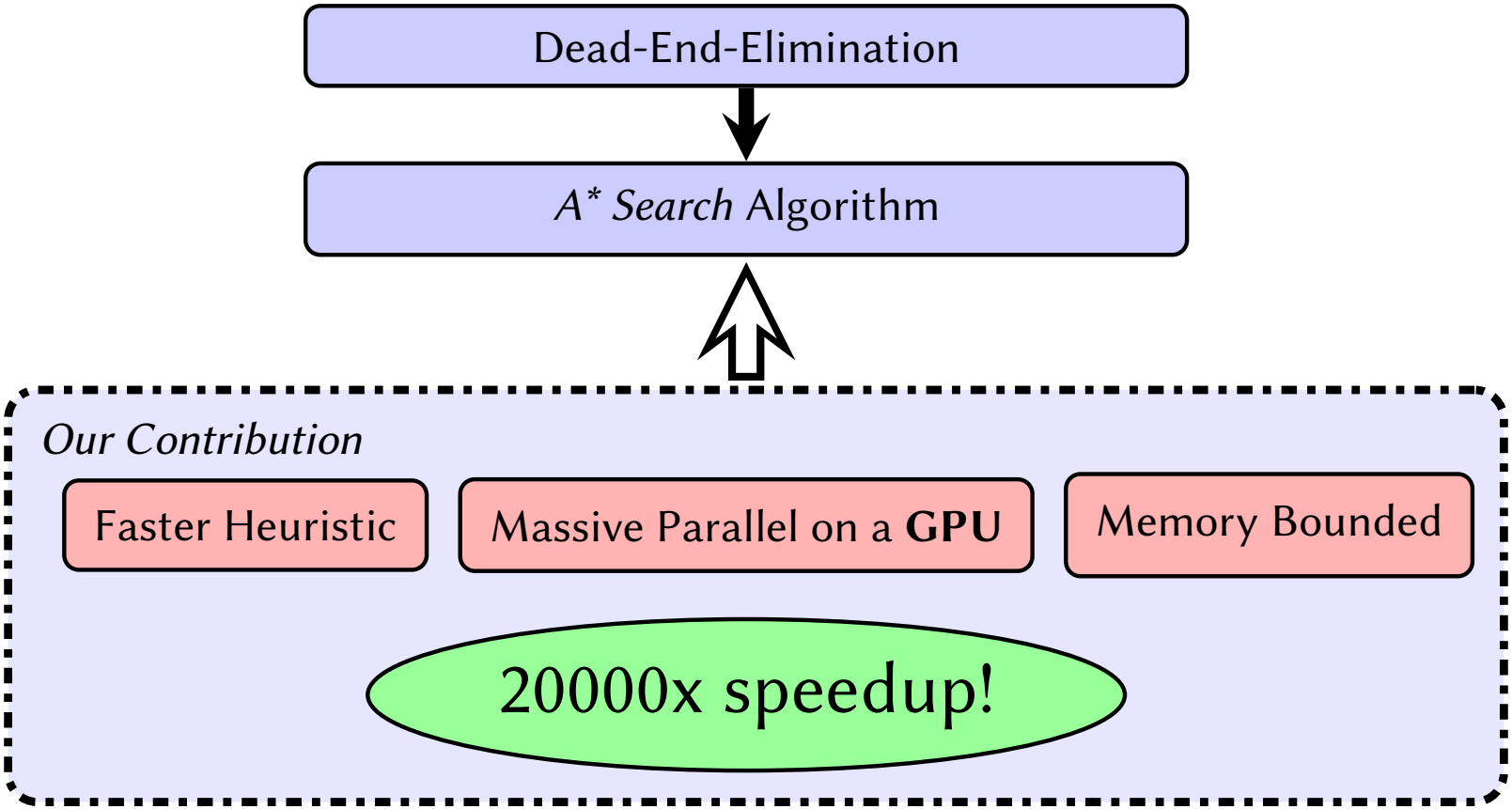
A* Search


-37
-30
-34
x
-37
-30
-34
-31
x_1
-37
-32
-30
-25
-34
-29
-22
-17
In Queue
Visiting
Visited
DEE Pruned
Priority Queue
x
x_1
$f(x_1) = -37$
x_2
$f(x_2)$

Our Contribution

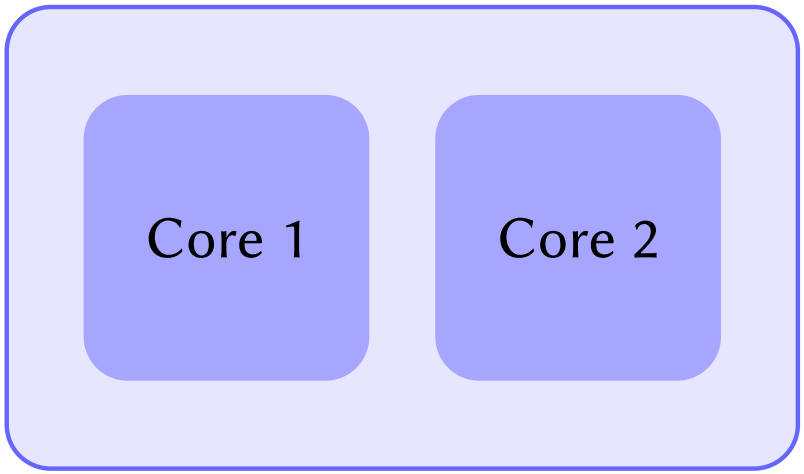


Our Contribution



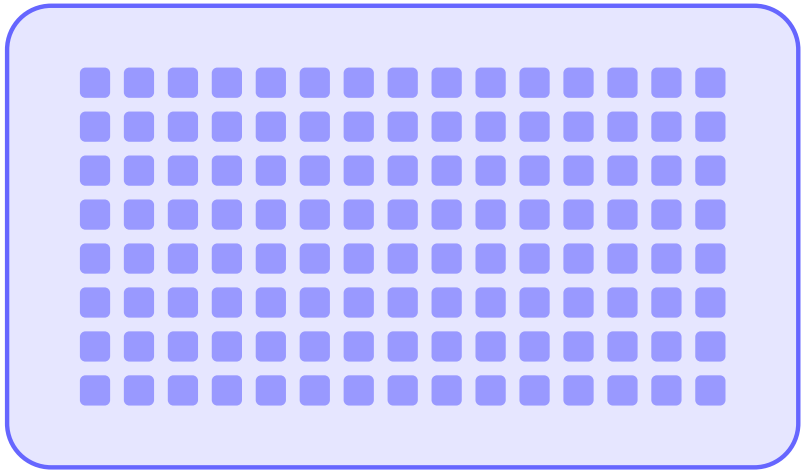
A Brief Concept to GPU

CPU (several cores)



- ☺ Decent single thread performance
- ☹ Limited in parallelism

GPU (thousands of cores)

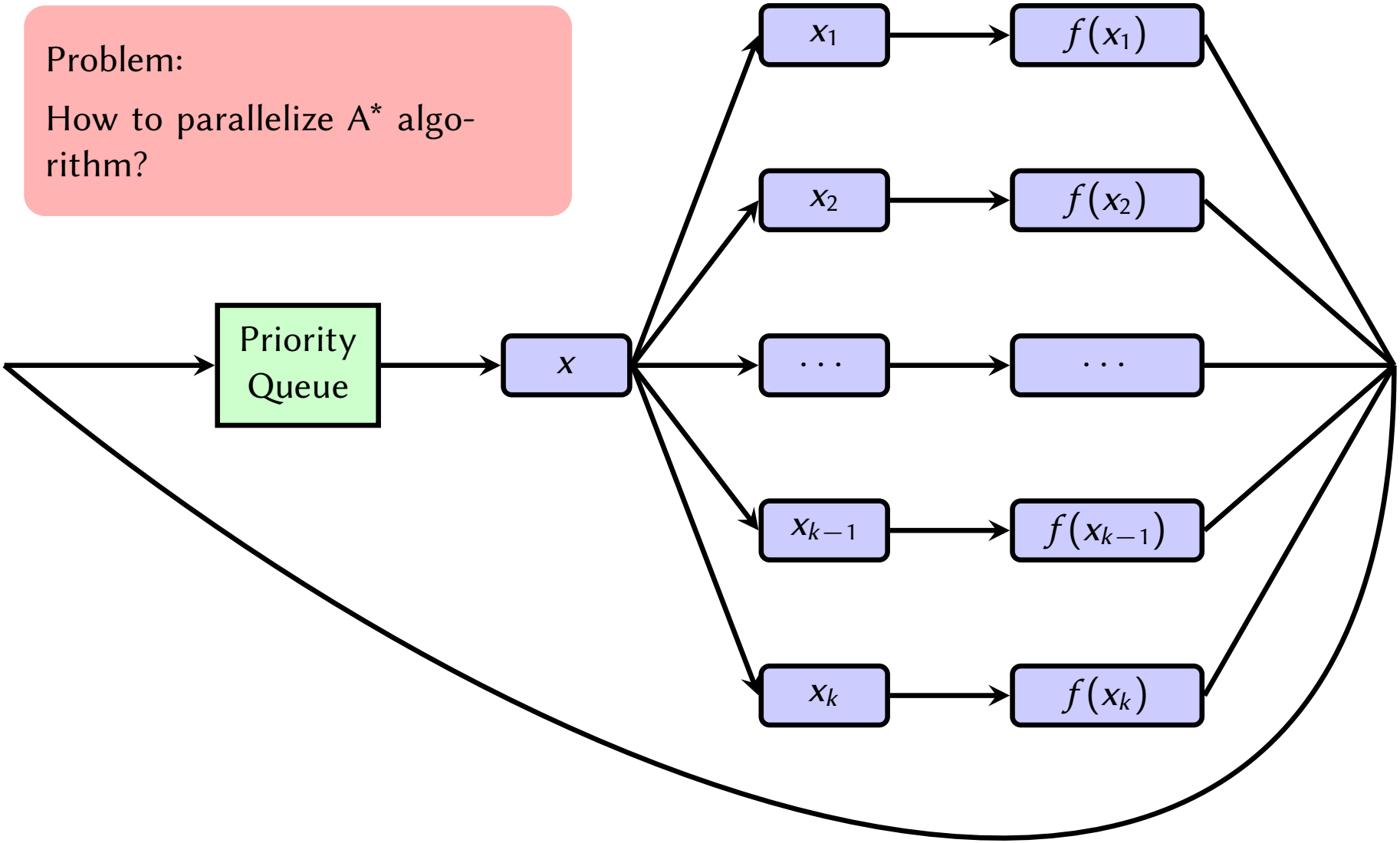


- ☺ More computational units
- ☺ Energy efficient
- ☹ Need massive parallelism

First Level of Parallelism

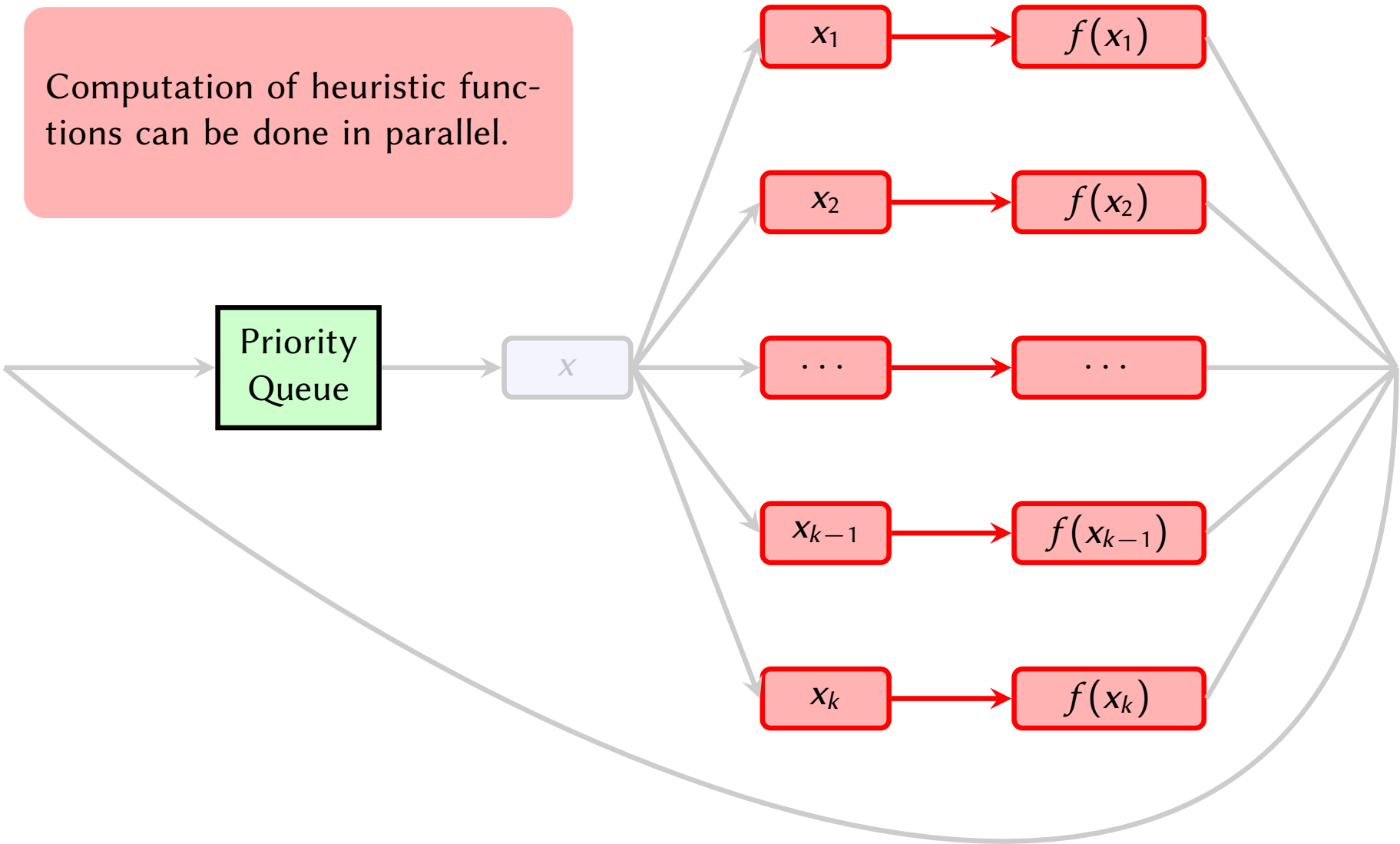
Problem:

How to parallelize A* algorithm?



First Level of Parallelism

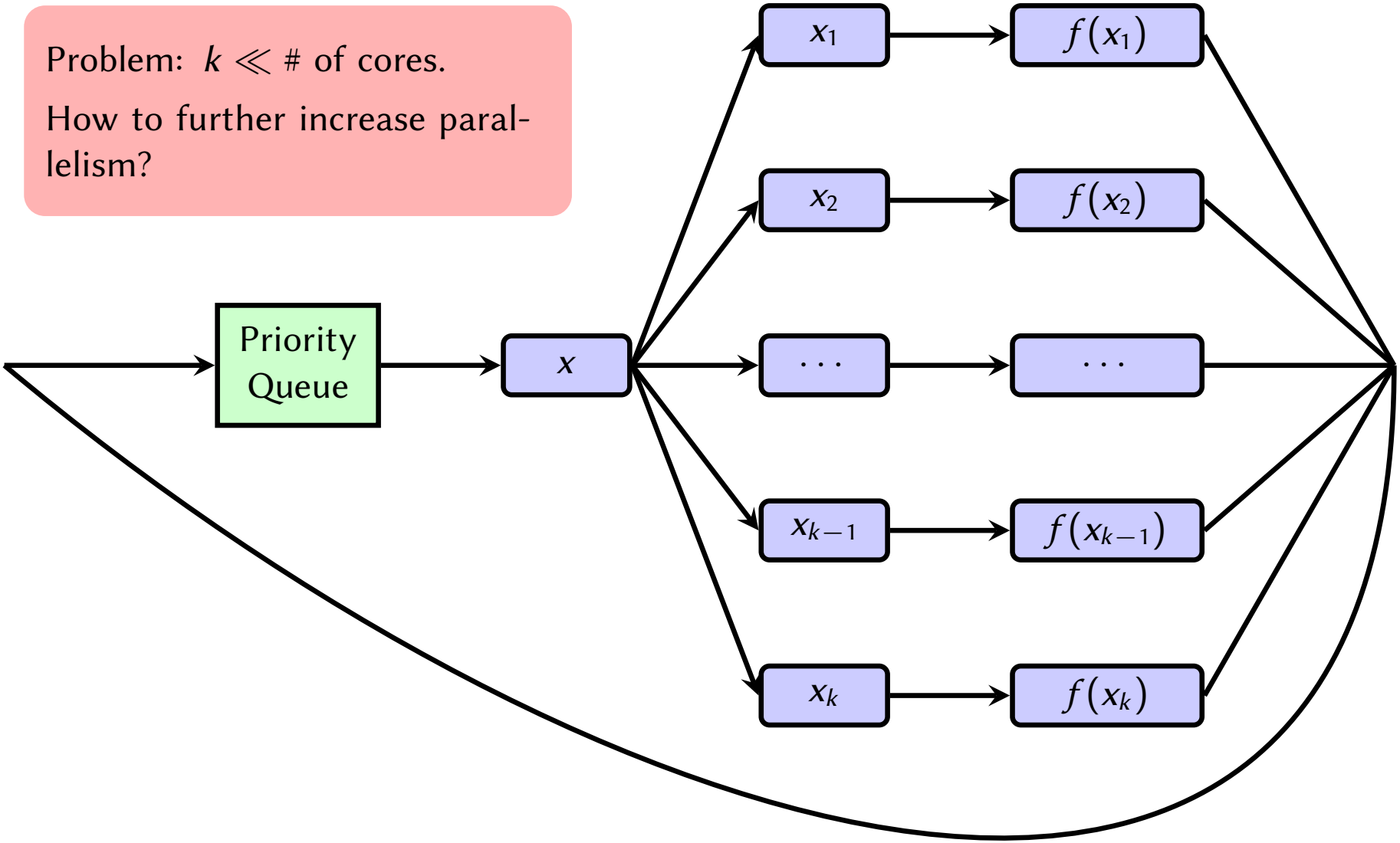
Computation of heuristic functions can be done in parallel.



First Level of Parallelism

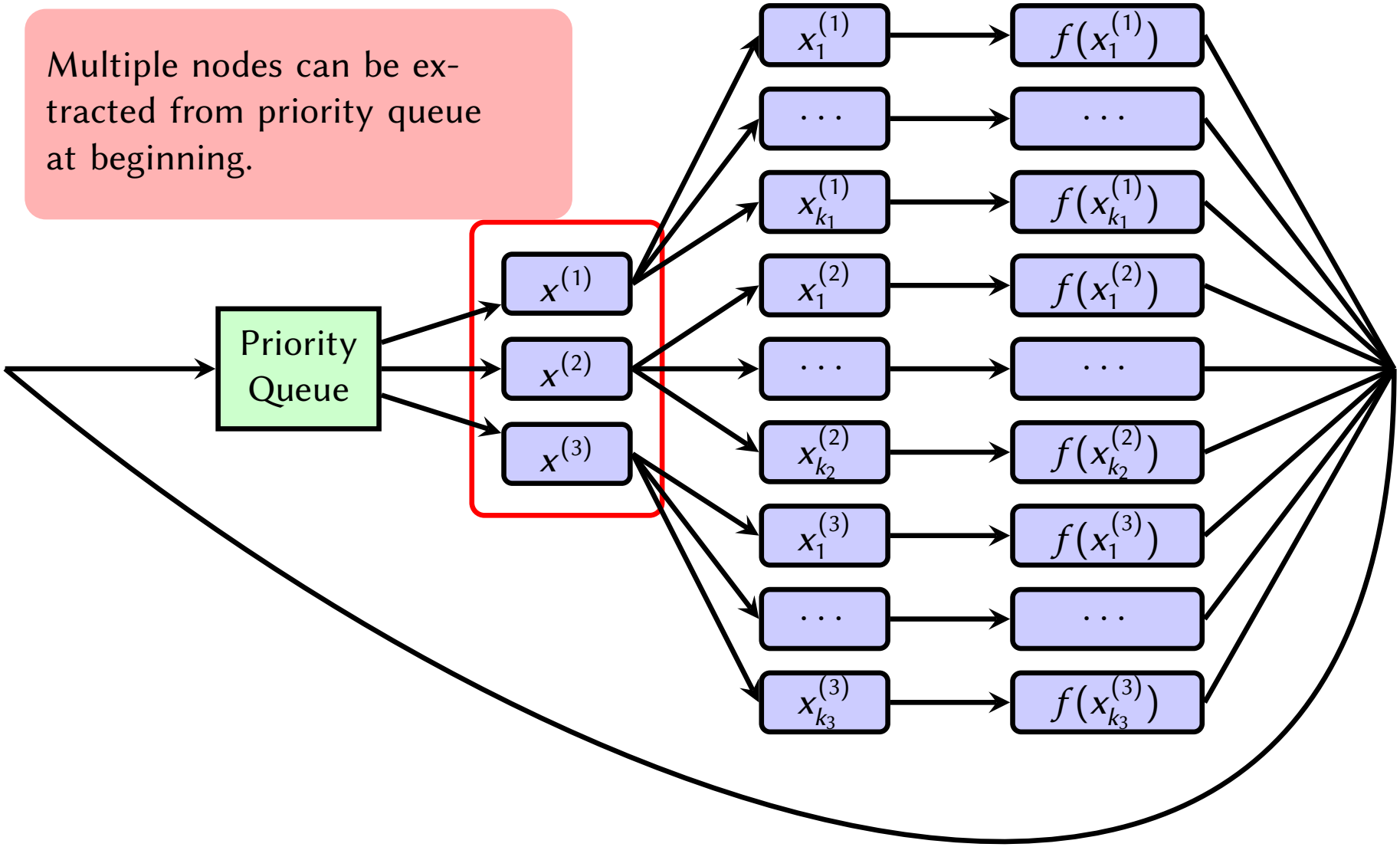
Problem: $k \ll$ # of cores.

How to further increase parallelism?

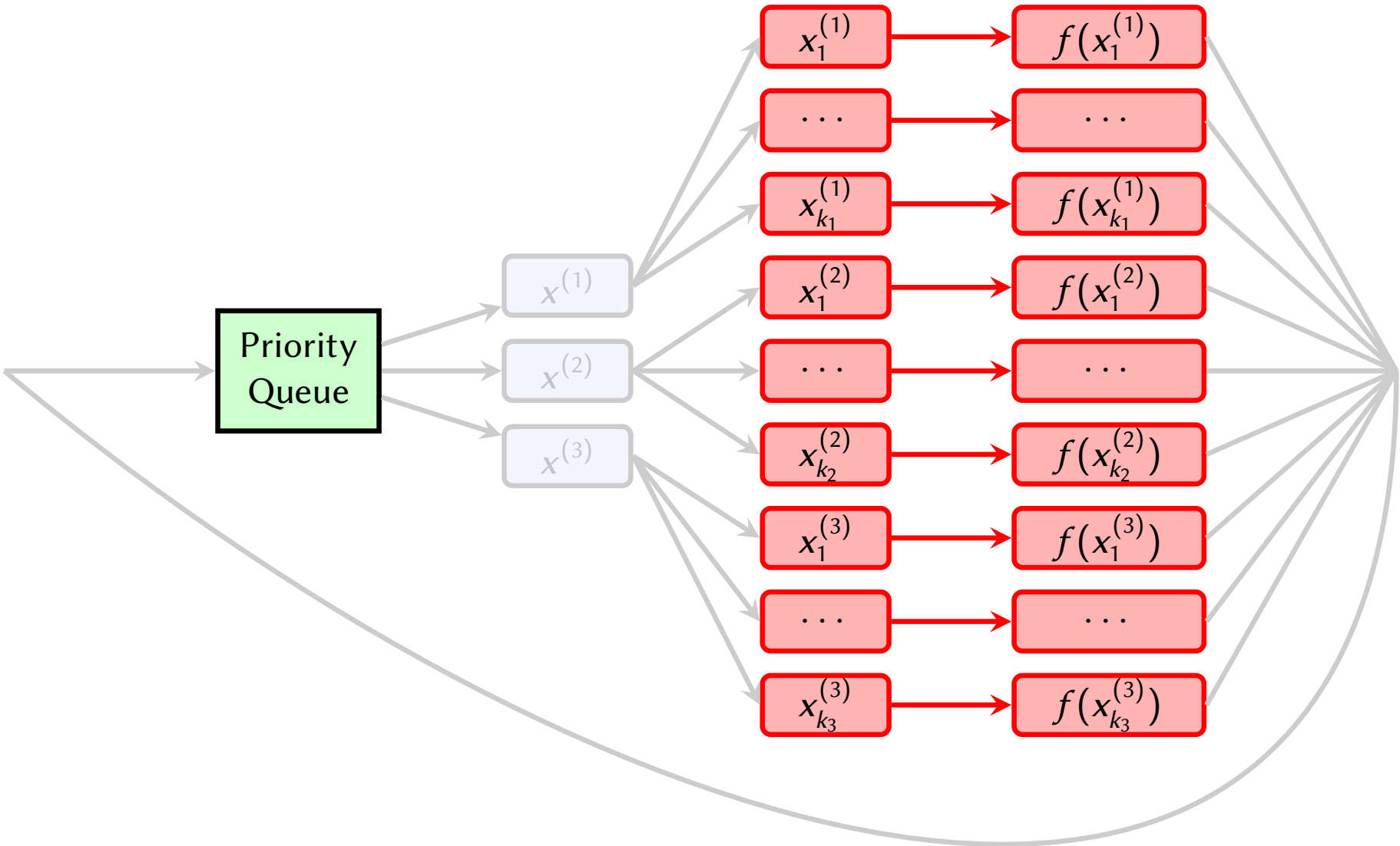


One Possible Solution

Multiple nodes can be extracted from priority queue at beginning.



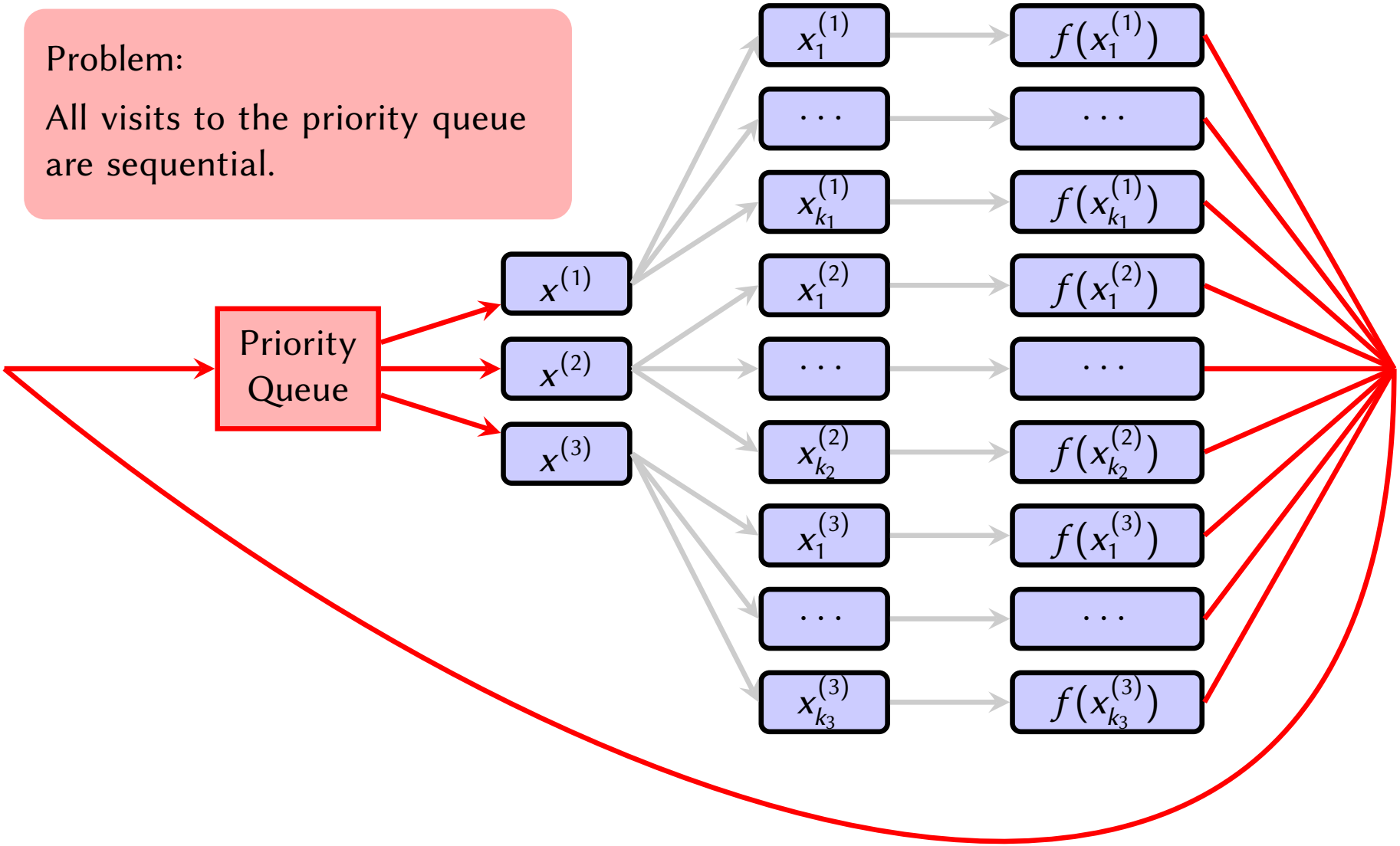
One Possible Solution



One Possible Solution

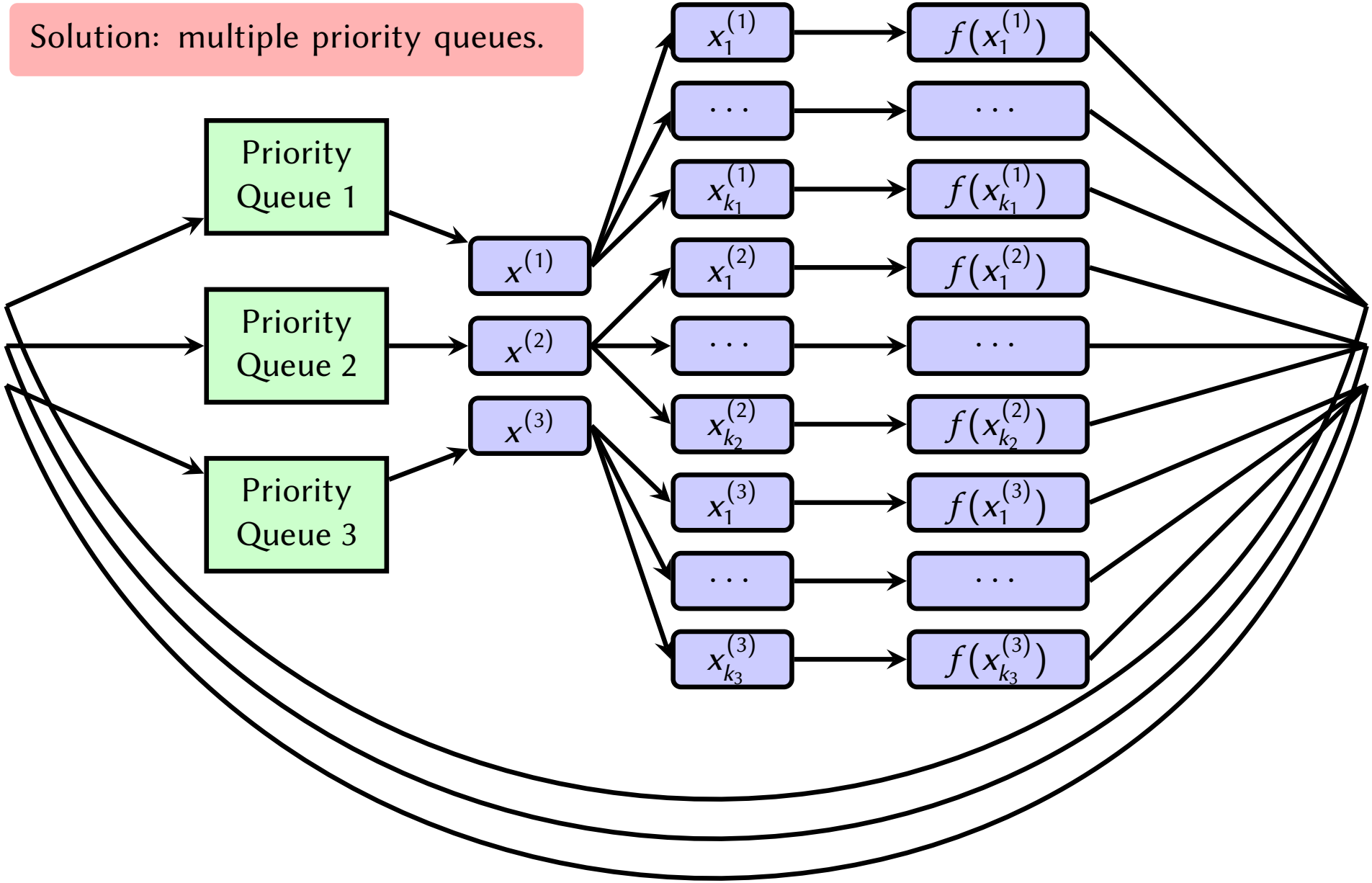
Problem:

All visits to the priority queue are sequential.

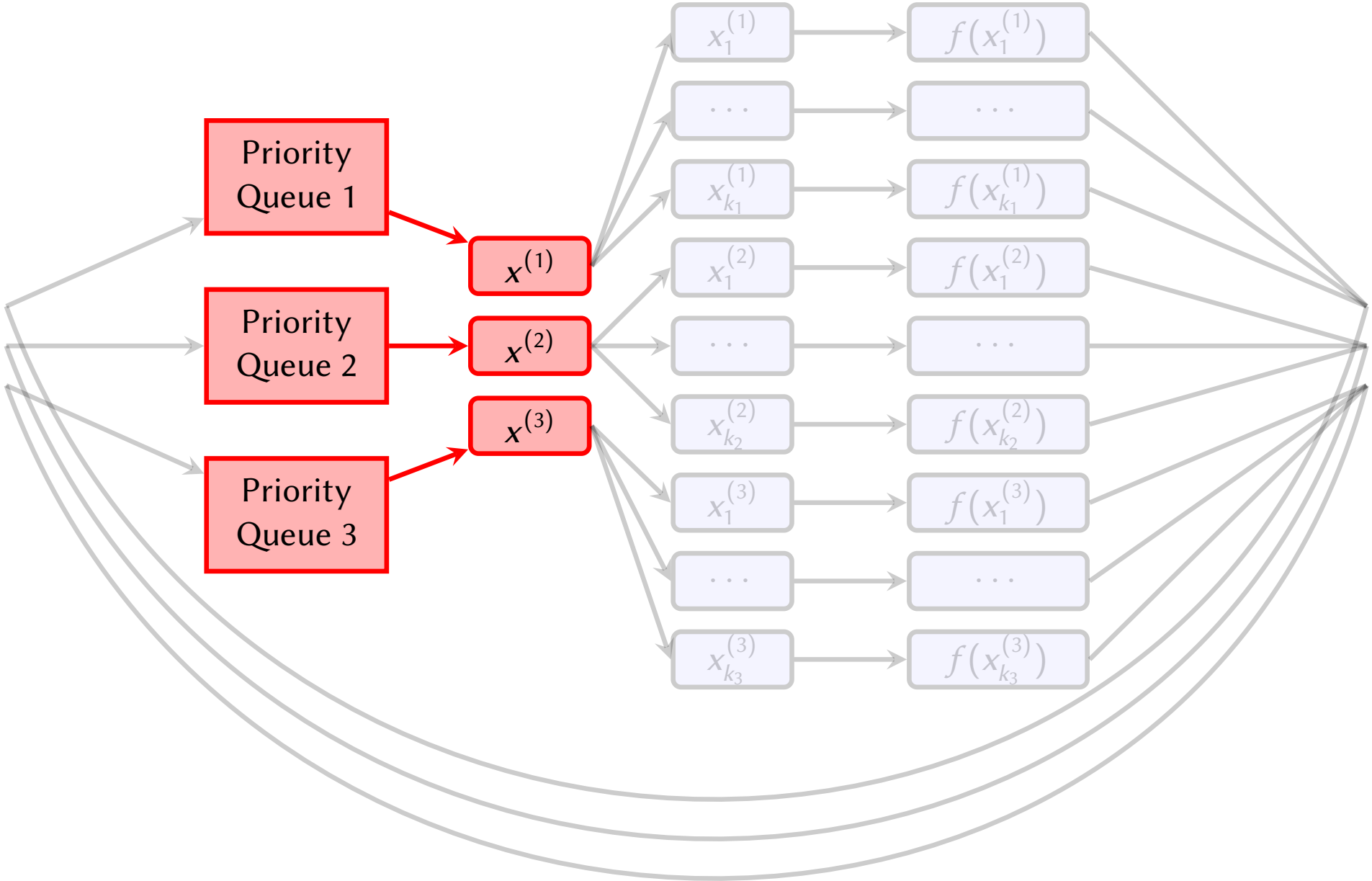


Full Parallelization of A* Search Algorithm

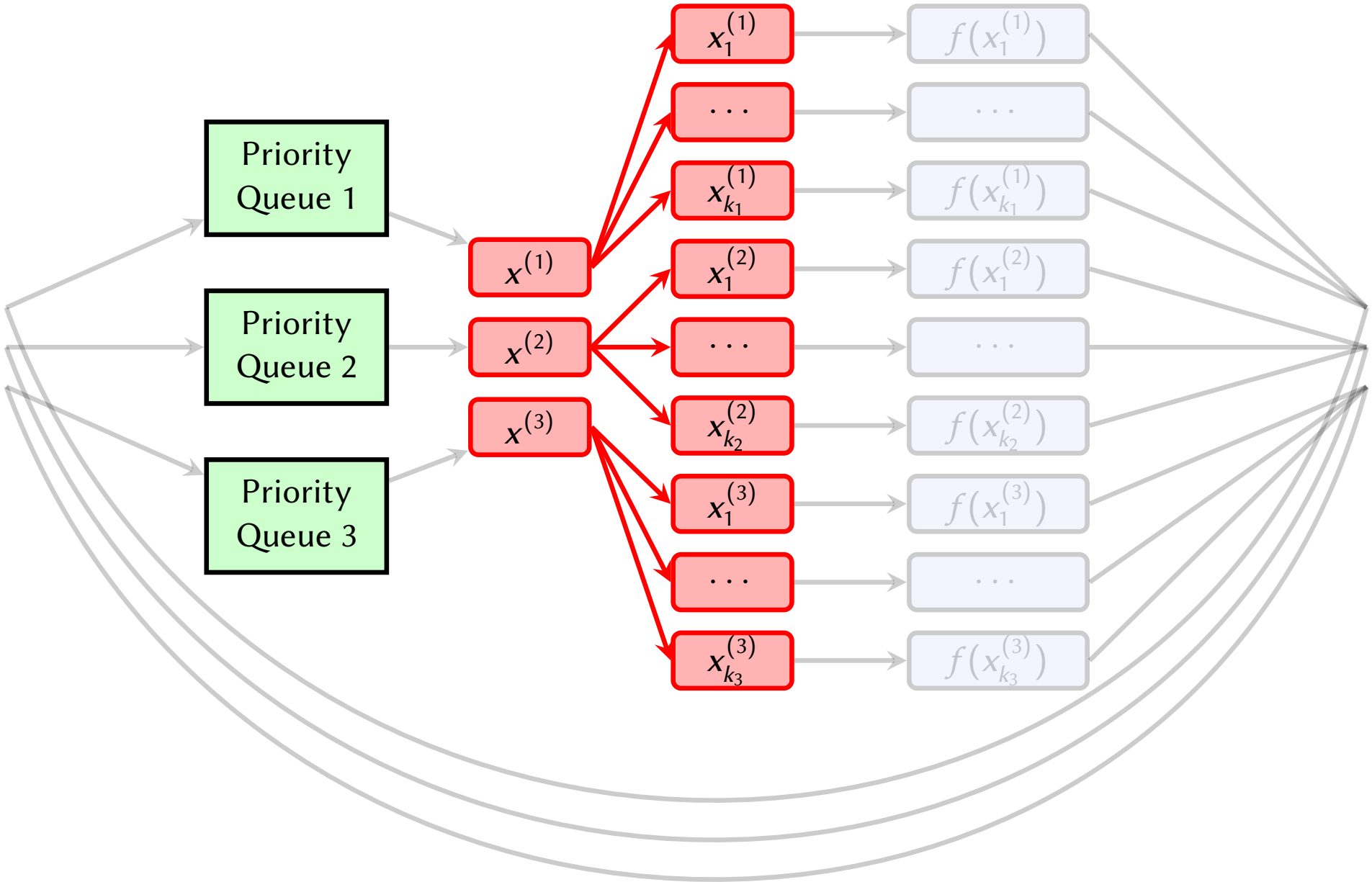
Solution: multiple priority queues.



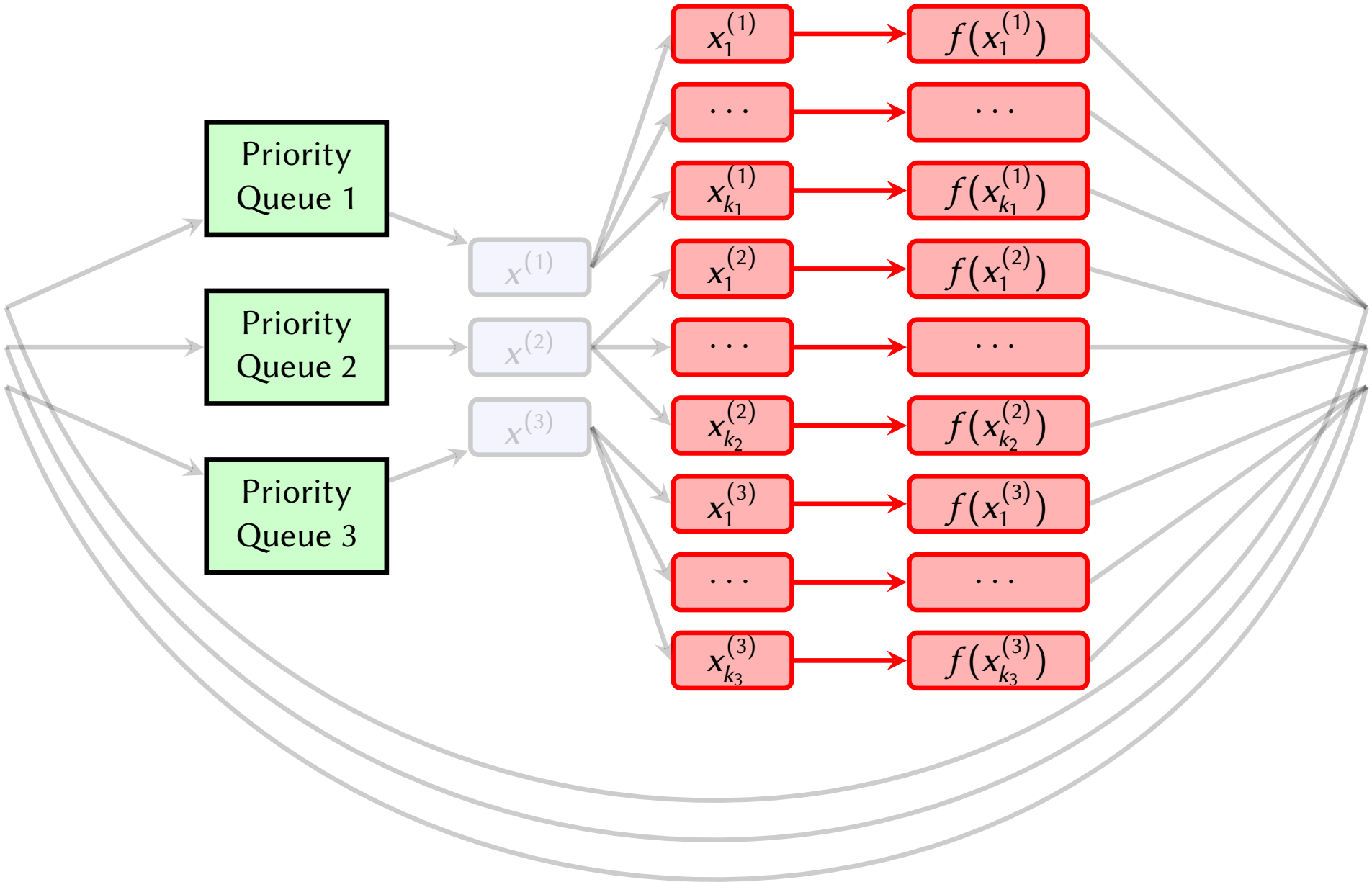
Full Parallelization of A* Search Algorithm



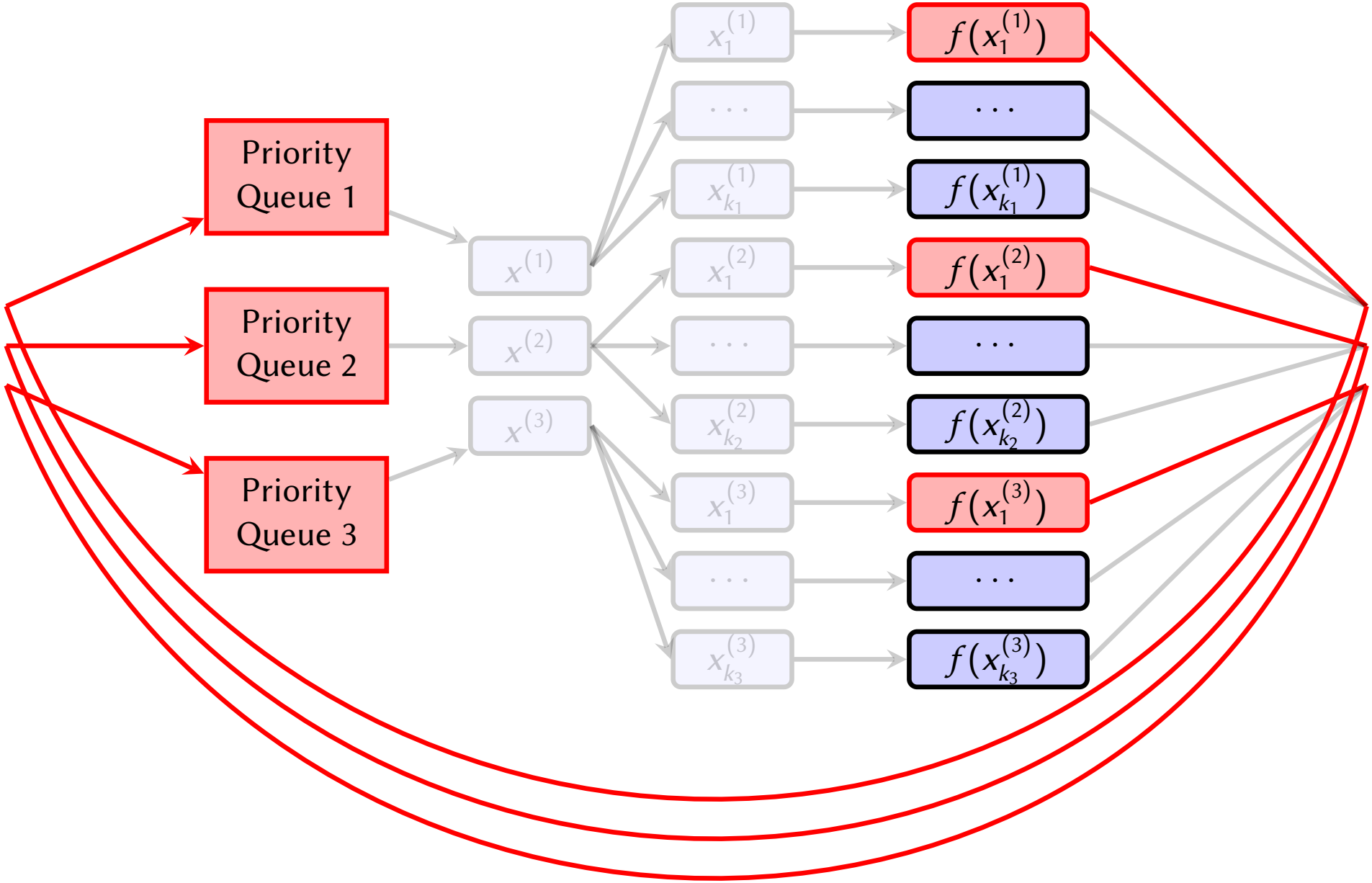
Full Parallelization of A* Search Algorithm



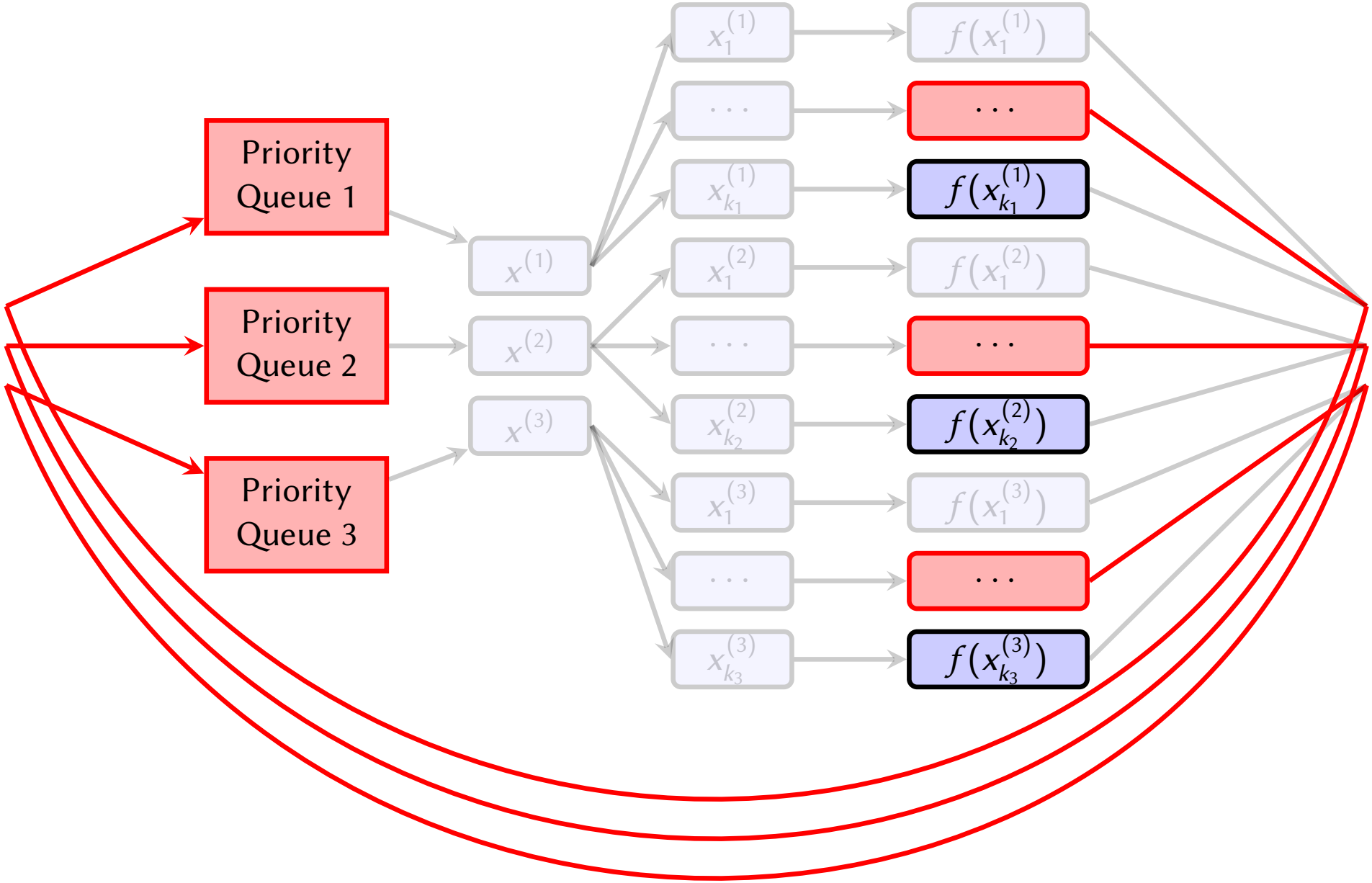
Full Parallelization of A* Search Algorithm



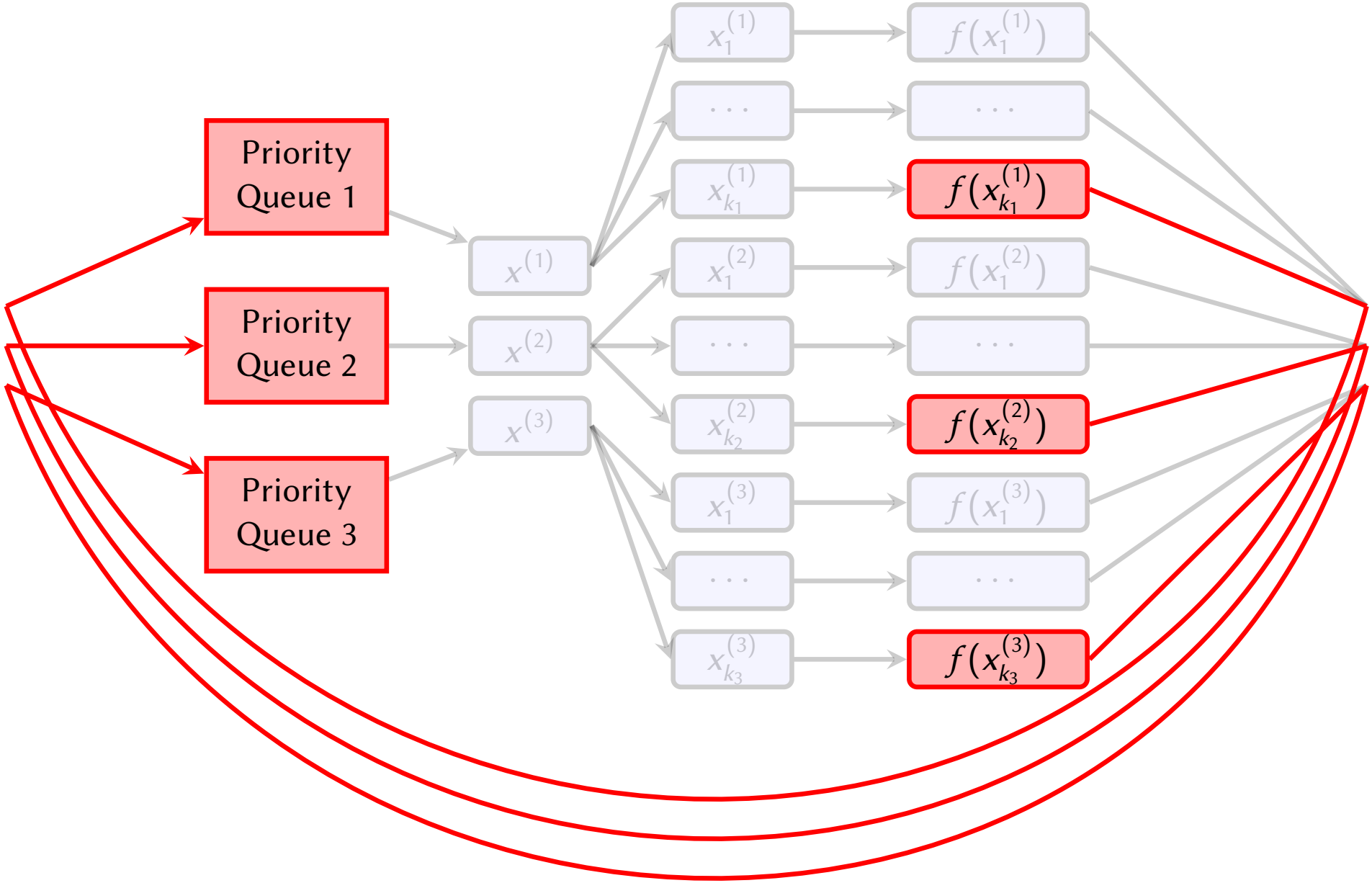
Full Parallelization of A* Search Algorithm



Full Parallelization of A* Search Algorithm



Full Parallelization of A* Search Algorithm



Parallel Memory Bounded A* Algorithm

Memory Exausted Problem

- Memory Resources prevents us from solving larger problem!
- Basic idea: throws aways unpromising nodes
  - Do a global sort according to nodes' heuristic values
  - Only keep nodes whose ranks exceed threshold

Pros and cons

- ☺ Continue to execute after running out of memory
- ☹ Cannot fully guarantee of GMEC
- 😐 Compared to traditional heuristic algorithm:
  - ☺ Guarantee GMEC solution when memory is large enough
  - ☺ Possible to know whether we get GMEC solution

Parallel Memory Bounded A* Algorithm

Memory Exausted Problem

- Memory Resources prevents us from solving larger problem!
- Basic idea: throws aways unpromising nodes
  - Do a global sort according to nodes' heuristic values
  - Only keep nodes whose ranks exceed threshold

Pros and cons

- ☺ Continue to execute after running out of memory
- ☹ Cannot fully guarantee of GMEC
- 😐 Compared to traditional heuristic algorithm:
  - ☺ Guarantee GMEC solution when memory is large enough
  - ☺ Possible to know whether we get GMEC solution

Environment

Experiment

- On OSPREY platform from *Donald Lab* (Duke University)
- Native sequence recovery (core redesign)
- iMinDEE to consider *side-chain flexibility*
- To measure:
  - speed and memory consumption of GPU-Based A*
  - *correctness* of parallel A* under bounded memory
- Availability: `http://github.com/zhou13/gOSPREY`

Environment Information

CPU: Intel Xeon™ E5-1620 3.6GHz

GPU: NVIDIA Tesla K20C (2496 logic cores)

Environment

Experiment

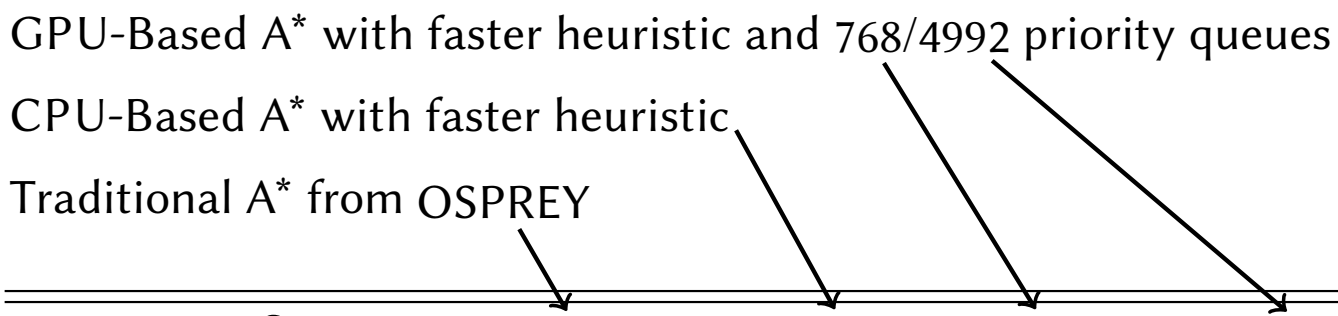
- On OSPREY platform from *Donald Lab* (Duke University)
- Native sequence recovery (core redesign)
- iMinDEE to consider *side-chain flexibility*
- To measure:
  - speed and memory consumption of GPU-Based A*
  - *correctness* of parallel A* under bounded memory
- Availability: `http://github.com/zhou13/gOSPREY`

Environment Information

CPU: Intel Xeon™ E5-1620 3.6GHz

GPU: NVIDIA Tesla K20C (2496 logic cores)

Performance of GPU-Based A*



PDB	Space	OSPREY	A*1	GA*768	GA*4992
2QCP	$2 \cdot 10^{17}$	21551916	51091	3075	1146
1XMK	$2 \cdot 10^{14}$	247585	2990	296	121
1X6I	$7 \cdot 10^{13}$	96990	1406	138	73
1UCS	$6 \cdot 10^{12}$	88135	1771	182	79
1CC8	$3 \cdot 10^{14}$	77614	1078	99	53
2CS7	$8 \cdot 10^{12}$	64187	1154	149	57
...	...	...	...	...	...

Time is measured in millisecond.

Performance of GPU-Based A*

GPU-Based A* with faster heuristic and 768/4992 priority queues

CPU-Based A* with faster heuristic

Traditional A* from OSPREY

PDB	Space	OSPREY	A*1	GA*768	GA*4992
2QCP	$2 \cdot 10^{17}$	21551916	51091	3075	1146
1XMK	$2 \cdot 10^{14}$	247585	2990	296	121
1X6I	$7 \cdot 10^{13}$	96990	1406	138	73
1UCS	$6 \cdot 10^{12}$	88135	1771	182	79
1CC8	$3 \cdot 10^{14}$	77614	1078	99	53
2CS7	$8 \cdot 10^{12}$	64187	1154	149	57
...	...	...	...	...	...

Time is measured in millisecond.

Performance of GPU-Based A*

GPU-Based A* with faster heuristic and 768/4992 priority queues

CPU-Based A* with faster heuristic

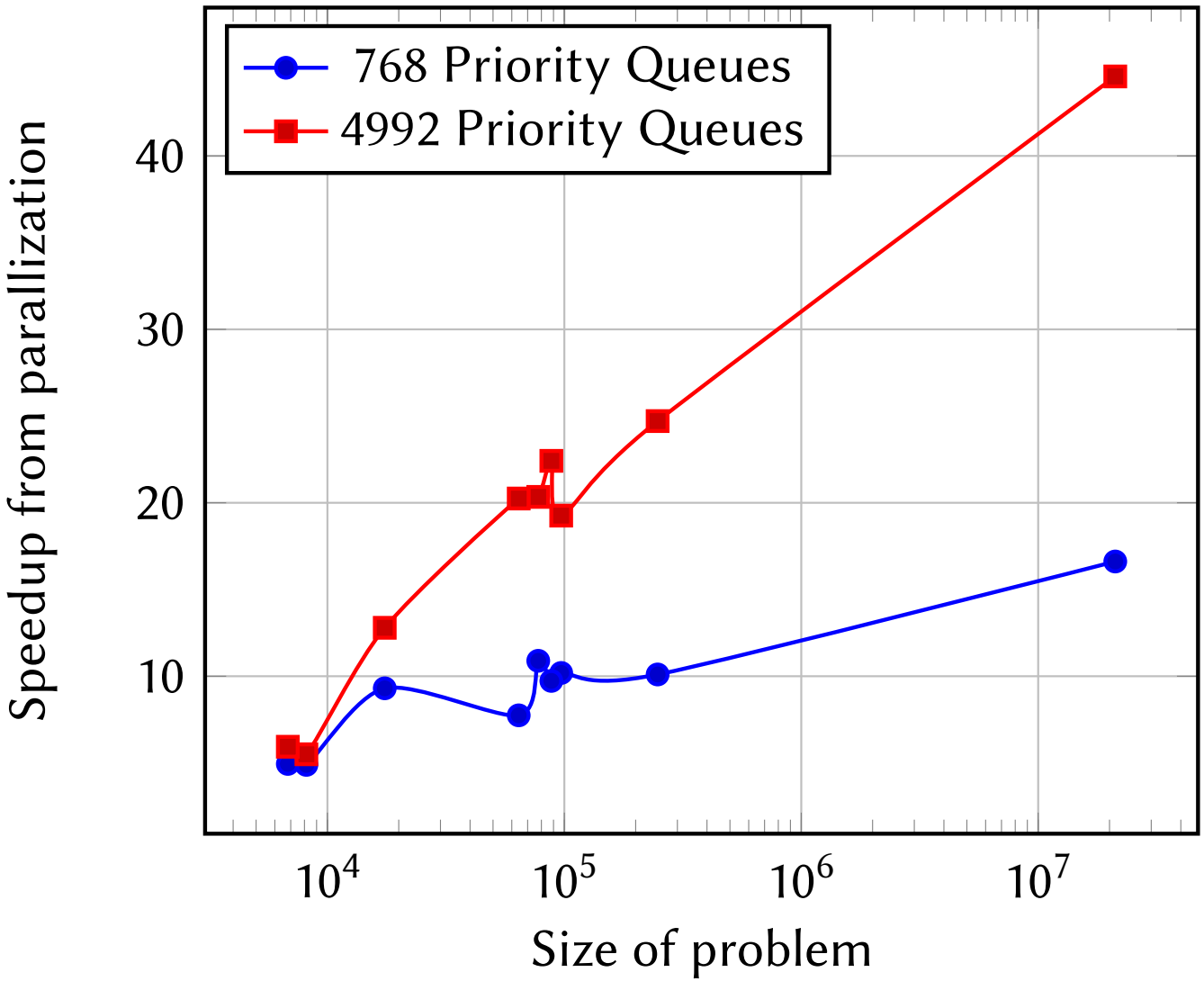
Traditional A* from OSPREY

20 000x speedup

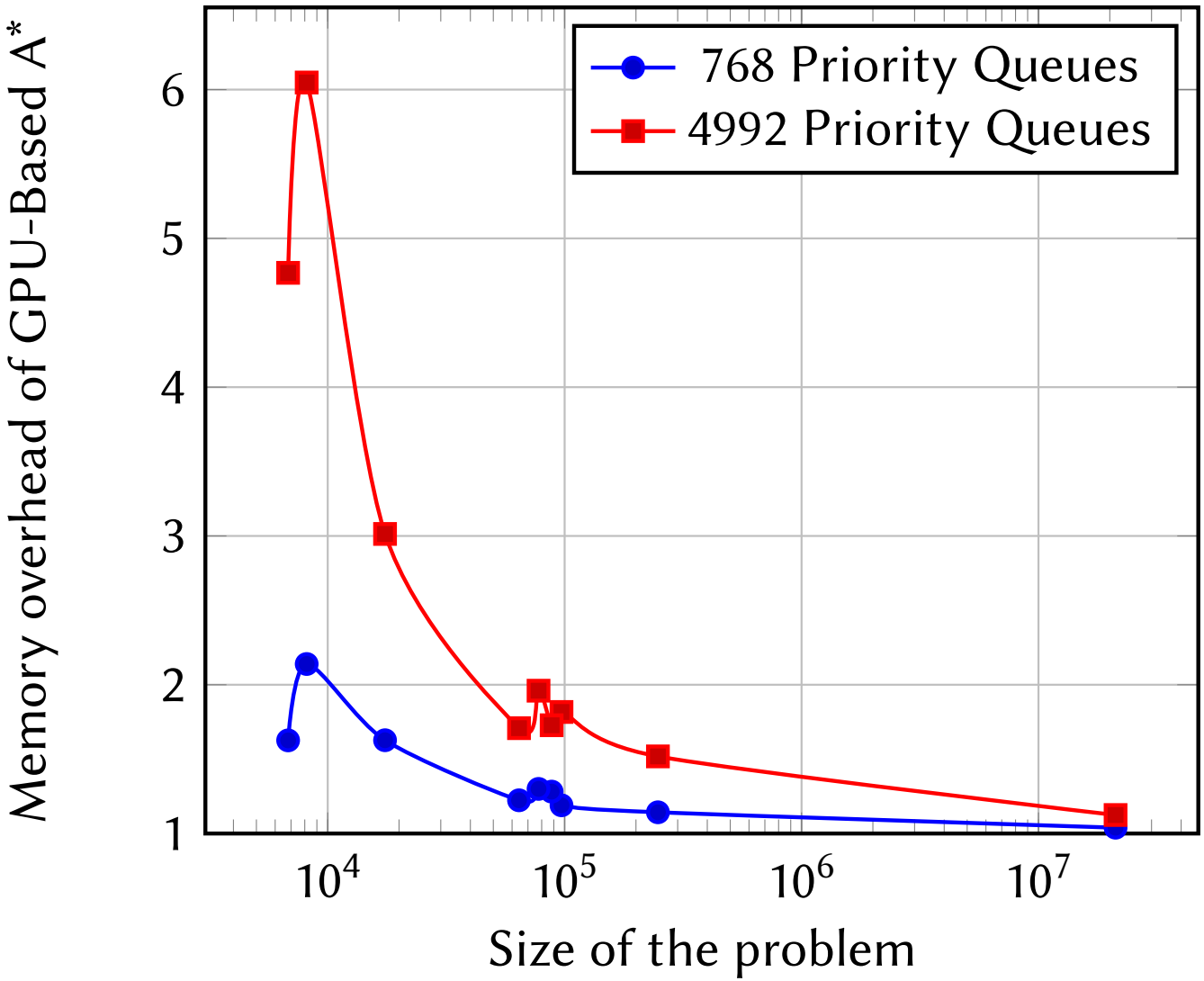
PDB	Space	OSPREY	A*1	GA*768	GA*4992
2QCP	$2 \cdot 10^{17}$	21551916	51091	3075	1146
1XMK	$2 \cdot 10^{14}$	247585	2990	296	121
1X6I	$7 \cdot 10^{13}$	96990	1406	138	73
1UCS	$6 \cdot 10^{12}$	88135	1771	182	79
1CC8	$3 \cdot 10^{14}$	77614	1078	99	53
2CS7	$8 \cdot 10^{12}$	64187	1154	149	57
...	...	...	...	...	...

Time is measured in millisecond.

Speedup from Parallelization



Memory Overhead



Native sequence recovery experiment

PDB		1OAI	1U2H	1ZZK	2CS7	2DSX	3D3B
# of Mutable Residues		16	18	14	15	15	15
Conformation Space		$2 \cdot 10^{22}$	$2 \cdot 10^{20}$	$2 \cdot 10^{15}$	$2 \cdot 10^{23}$	$3 \cdot 10^{20}$	$6 \cdot 18^{18}$
A* Search Space		$4 \cdot 10^{7}$	$8 \cdot 10^{6}$	$8 \cdot 10^{6}$	$4 \cdot 10^{7}$	$4 \cdot 10^{7}$	$3 \cdot 10^{7}$
3×10^{6} node limit	GMEC Gotten	YES	YES	YES	YES	YES	YES
	GMEC Assured	YES	YES	YES	YES	YES	YES
	Recover Ratio	74%	75%	87%	46%	48%	53%
3×10^{5} node limit	GMEC Gotten	YES	YES	YES	YES	YES	YES
	GMEC Assured	NO	YES	YES	NO	NO	NO
	Recover Ratio	74%	75%	87%	46%	48%	54%
3×10^{4} node limit	GMEC Gotten	NO	YES	YES	NO	YES	NO
	GMEC Assured	NO	NO	NO	NO	NO	NO
	Recover Ratio	62%	75%	85%	48%	46%	48%

Native sequence recovery experiment

PDB		1OAI	1U2H	1ZZK	2CS7	2DSX	3D3B
# of Mutable Residues		16	18	14	15	15	15
Conformation Space		$2 \cdot 10^{22}$	$2 \cdot 10^{20}$	$2 \cdot 10^{15}$	$2 \cdot 10^{23}$	$3 \cdot 10^{20}$	$6 \cdot 18^{18}$
A* Search Space		$4 \cdot 10^{7}$	$8 \cdot 10^{6}$	$8 \cdot 10^{6}$	$4 \cdot 10^{7}$	$4 \cdot 10^{7}$	$3 \cdot 10^{7}$
3×10^{6} node limit	GMEC Gotten	YES	YES	YES	YES	YES	YES
	GMEC Assured	YES	YES	YES	YES	YES	YES
	Recover Ratio	74%	75%	87%	46%	48%	53%
3×10^{5} node limit	GMEC Gotten	YES	YES	YES	YES	YES	YES
	GMEC Assured	NO	YES	YES	NO	NO	NO
	Recover Ratio	74%	75%	87%	46%	48%	54%
3×10^{4} node limit	GMEC Gotten	NO	YES	YES	NO	YES	NO
	GMEC Assured	NO	NO	NO	NO	NO	NO
	Recover Ratio	62%	75%	85%	48%	46%	48%

Native sequence recovery experiment

PDB		1OAI	1U2H	1ZZK	2CS7	2DSX	3D3B
# of Mutable Residues		16	18	14	15	15	15
Conformation Space		$2 \cdot 10^{22}$	$2 \cdot 10^{20}$	$2 \cdot 10^{15}$	$2 \cdot 10^{23}$	$3 \cdot 10^{20}$	$6 \cdot 18^{18}$
A* Search Space		$4 \cdot 10^{7}$	$8 \cdot 10^{6}$	$8 \cdot 10^{6}$	$4 \cdot 10^{7}$	$4 \cdot 10^{7}$	$3 \cdot 10^{7}$
3×10^{6} node limit	GMEC Gotten	YES	YES	YES	YES	YES	YES
	GMEC Assured	YES	YES	YES	YES	YES	YES
	Recover Ratio	74%	75%	87%	46%	48%	53%
3×10^{5} node limit	GMEC Gotten	YES	YES	YES	YES	YES	YES
	GMEC Assured	NO	YES	YES	NO	NO	NO
	Recover Ratio	74%	75%	87%	46%	48%	54%
3×10^{4} node limit	GMEC Gotten	NO	YES	YES	NO	YES	NO
	GMEC Assured	NO	NO	NO	NO	NO	NO
	Recover Ratio	62%	75%	85%	48%	46%	48%

Native sequence recovery experiment

PDB		1OAI	1U2H	1ZZK	2CS7	2DSX	3D3B
# of Mutable Residues		16	18	14	15	15	15
Conformation Space		$2 \cdot 10^{22}$	$2 \cdot 10^{20}$	$2 \cdot 10^{15}$	$2 \cdot 10^{23}$	$3 \cdot 10^{20}$	$6 \cdot 18^{18}$
A* Search Space		$4 \cdot 10^{7}$	$8 \cdot 10^{6}$	$8 \cdot 10^{6}$	$4 \cdot 10^{7}$	$4 \cdot 10^{7}$	$3 \cdot 10^{7}$
3×10^{6} node limit	GMEC Gotten	YES	YES	YES	YES	YES	YES
	GMEC Assured	YES	YES	YES	YES	YES	YES
	Recover Ratio	74%	75%	87%	46%	48%	53%
3×10^{5} node limit	GMEC Gotten	YES	YES	YES	YES	YES	YES
	GMEC Assured	NO	YES	YES	NO	NO	NO
	Recover Ratio	74%	75%	87%	46%	48%	54%
3×10^{4} node limit	GMEC Gotten	NO	YES	YES	NO	YES	NO
	GMEC Assured	NO	NO	NO	NO	NO	NO
	Recover Ratio	62%	75%	85%	48%	46%	48%

Conclusion and Future Work

Our Contribution

- Optimized computation of heuristic function
- Massively parallel GPU-Based A*
- Memory bounded search algorithm

Results

- a 20000x speedup in *native sequence recovery*
- 1/10 memory while guaranteeing GMEC

Future Work

- Testing on more affordable GPUs
- Porting to large CPUs/GPUs cluster
- Parallelizing other parts of the framework

Acknowledgement

- Duke University
  - Mr. Kyle Roberts
  - Mr. Mark Hallen
  - Mr. Pablo Gainza
- Tsinghua University
  - Mr. Pufan He
  - Mr. Qiwei Feng

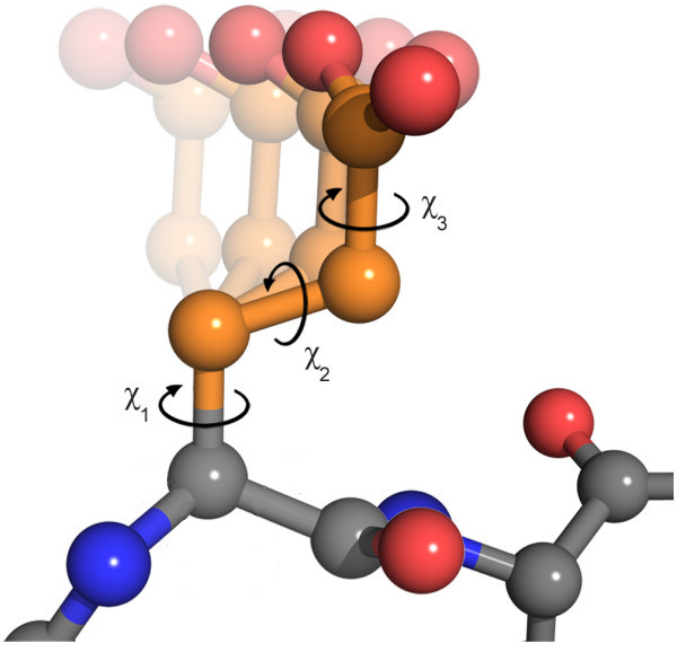
Funding

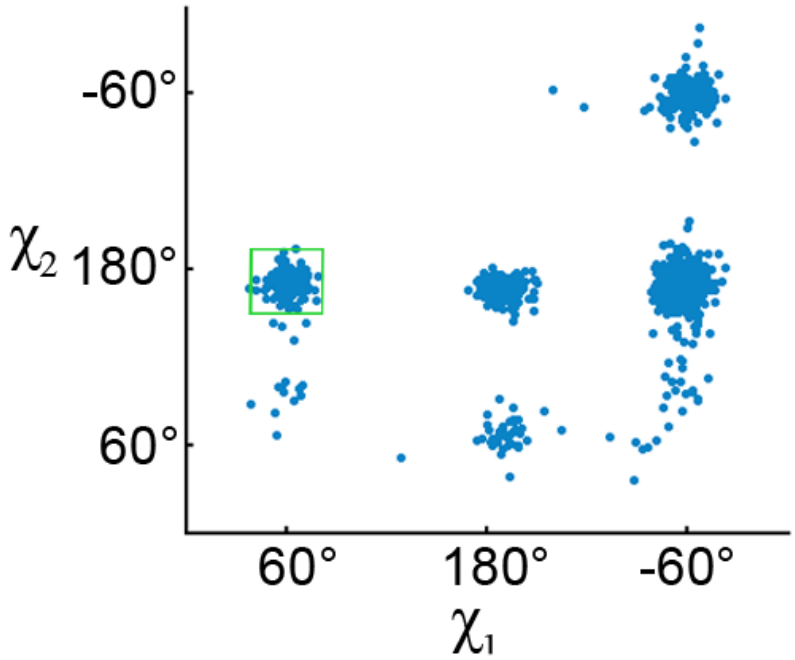
- National Basic Research Program of China
- National Natural Science Foundation of China
- National Institutes of Health from B.R.D

Question and Answer

Thank you!

Rotamer Library



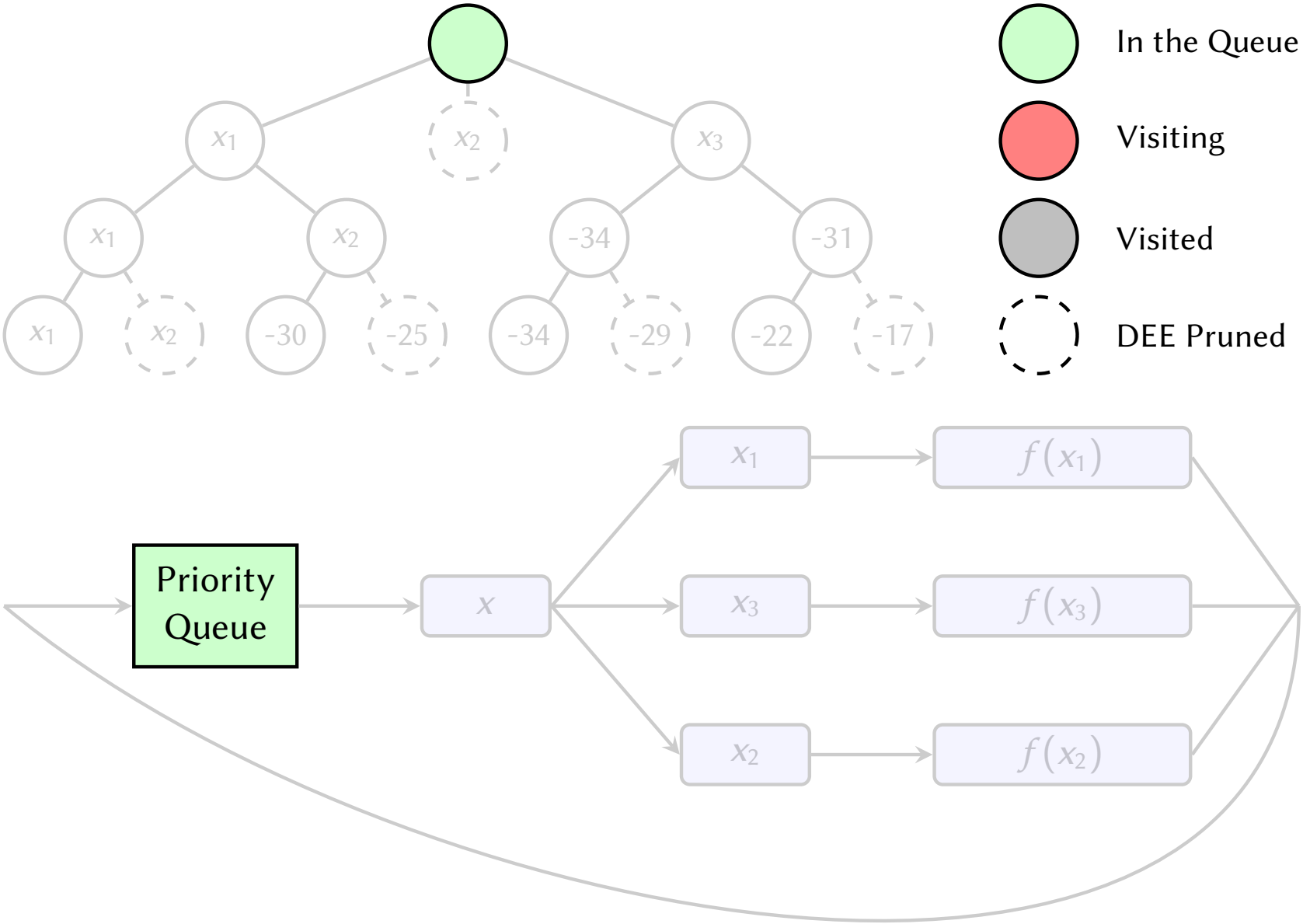


Harder, Tim, et al. (2010) and Gainza, P., Roberts, K. E., & Donald, B. R. (2012).

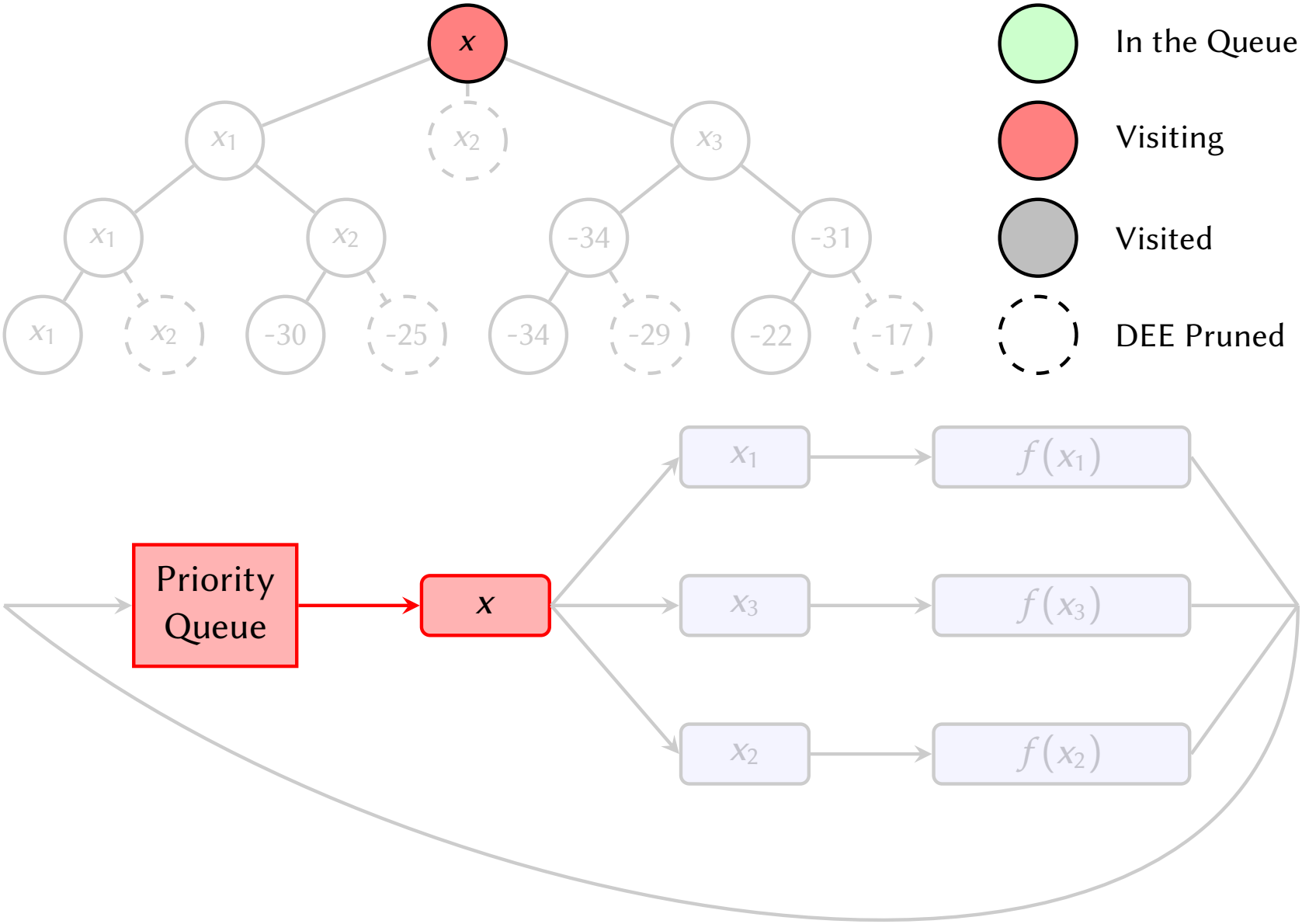
Price Table (2014)

Model	Price	Release Year	SP GFLOPS
Intel Xeon E5-1620	300$	2012	488
NVIDIA Tesla K20C	3000$	2012	3520
NVIDIA GeForce GTX 680	300$	2012	3090
NVIDIA GeForce GTX 770	400$	2013	3213
NVIDIA GeForce GTX 780	600$	2013	3977

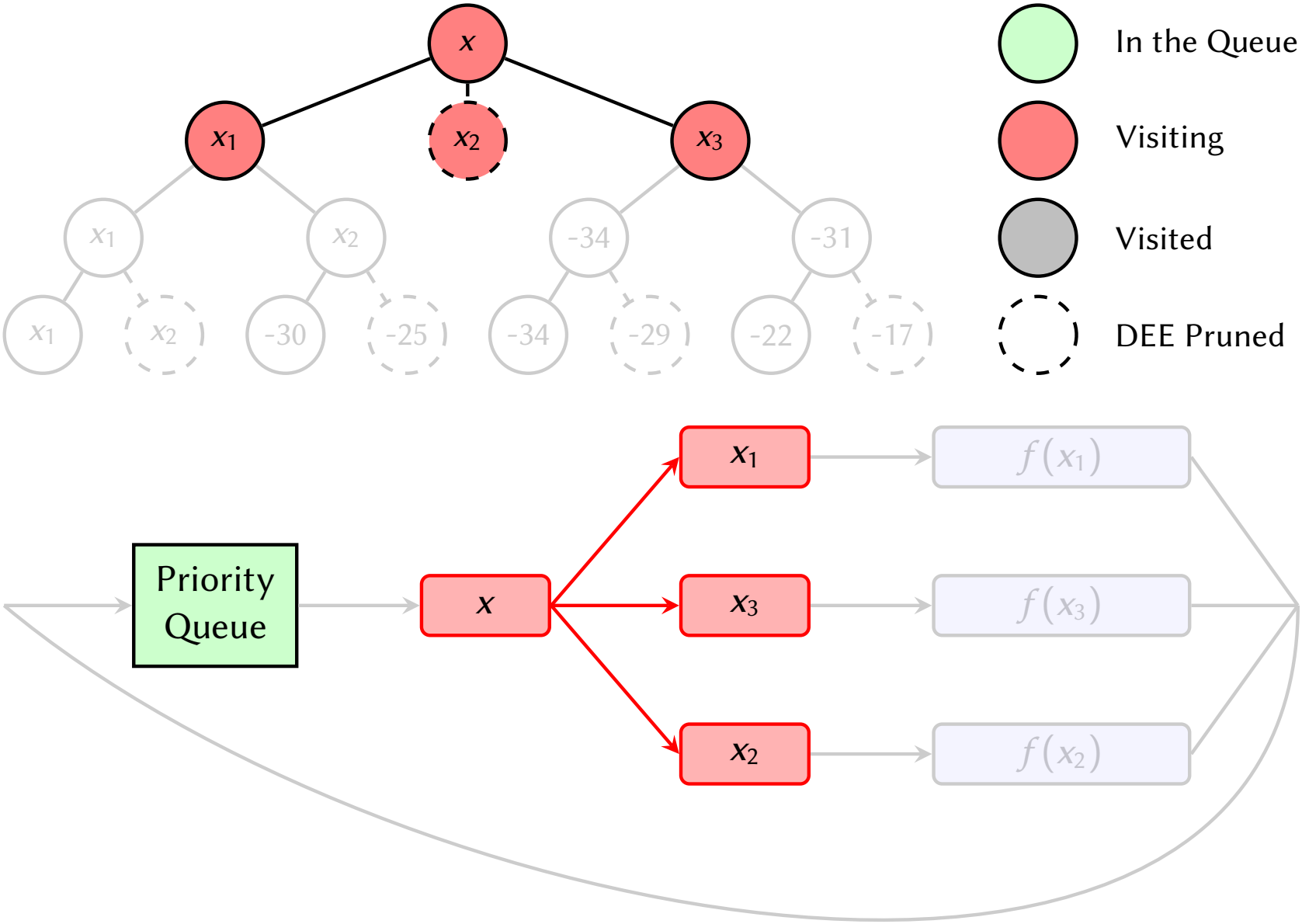
Full A* Search



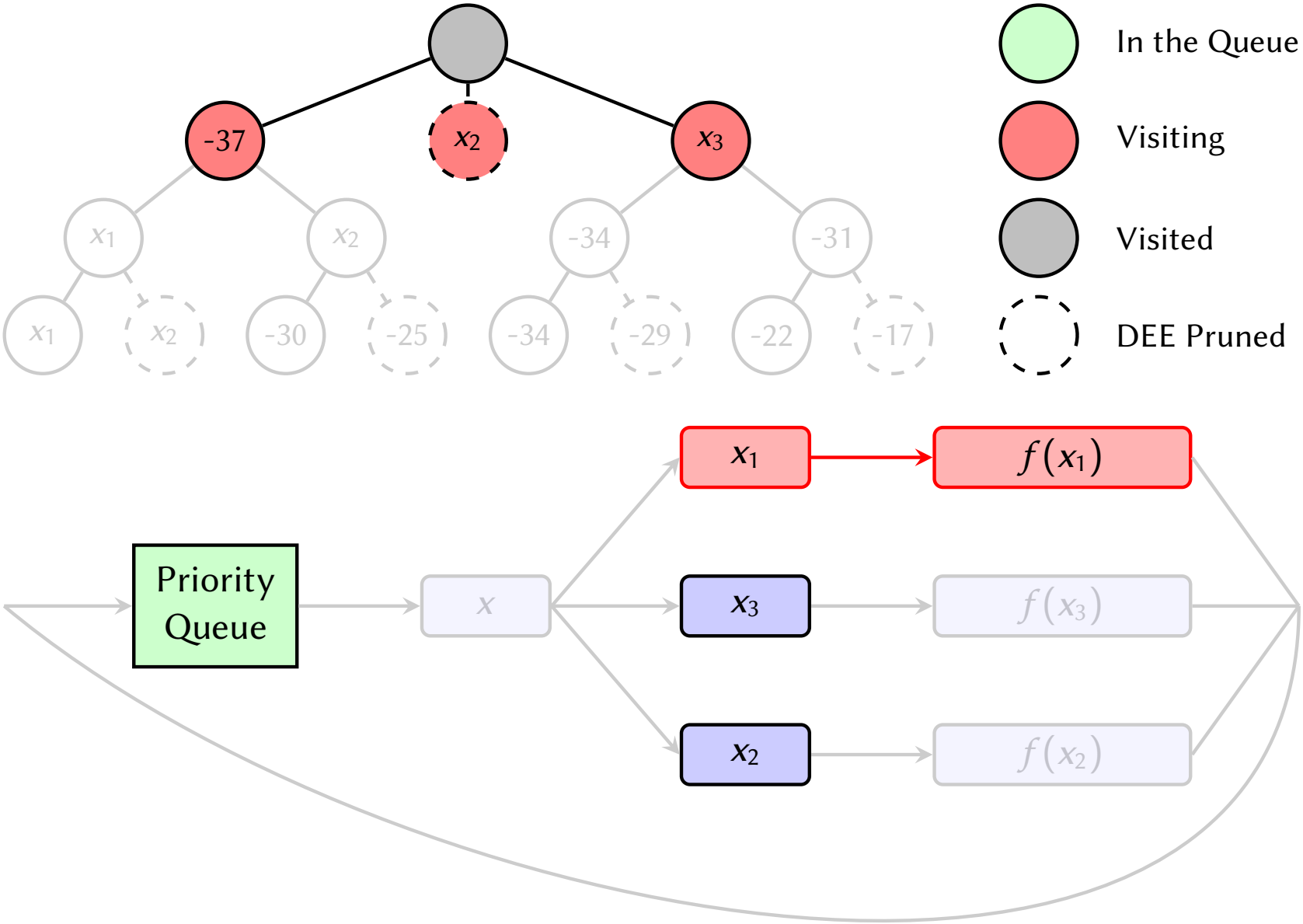
Full A* Search



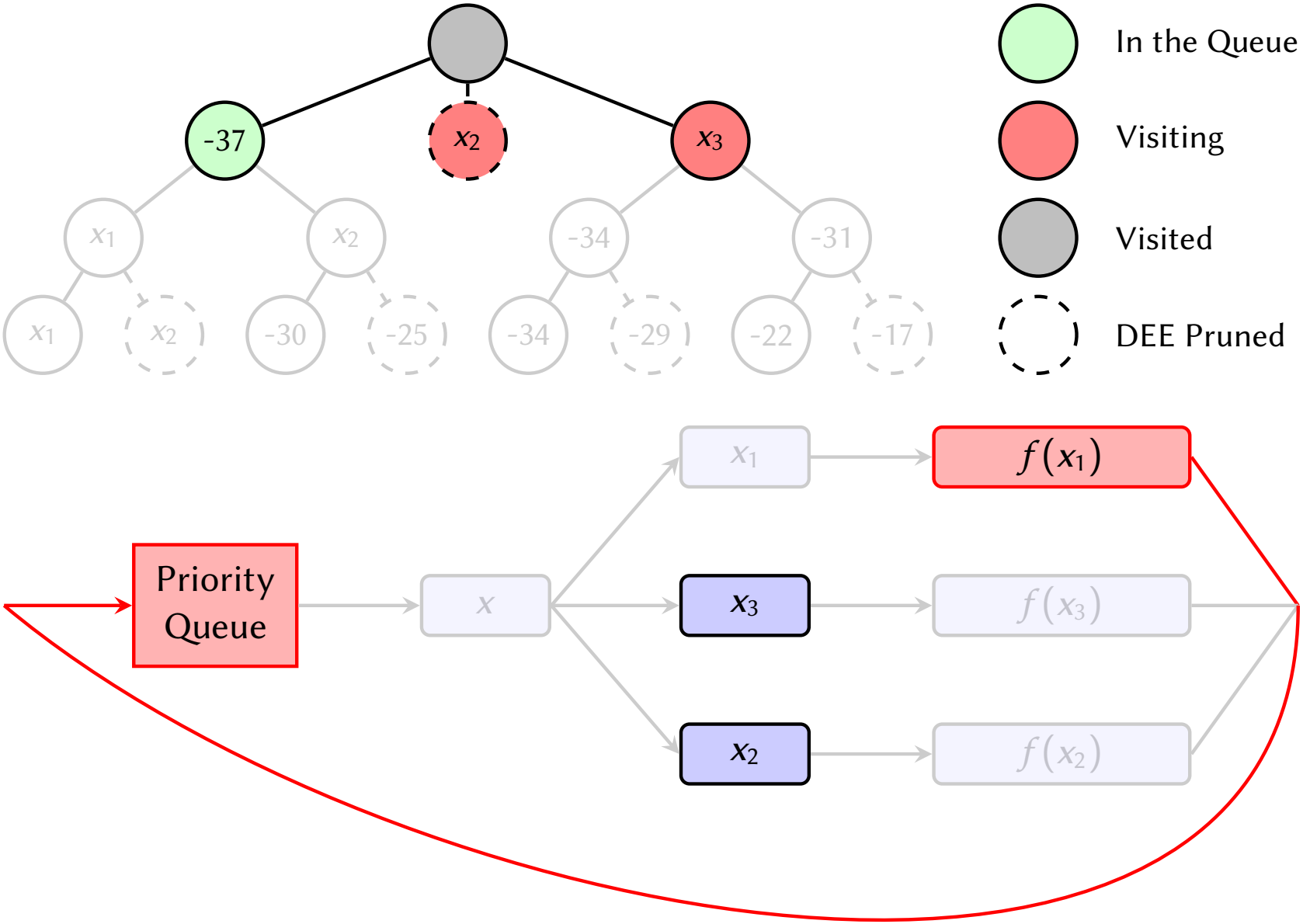
Full A* Search



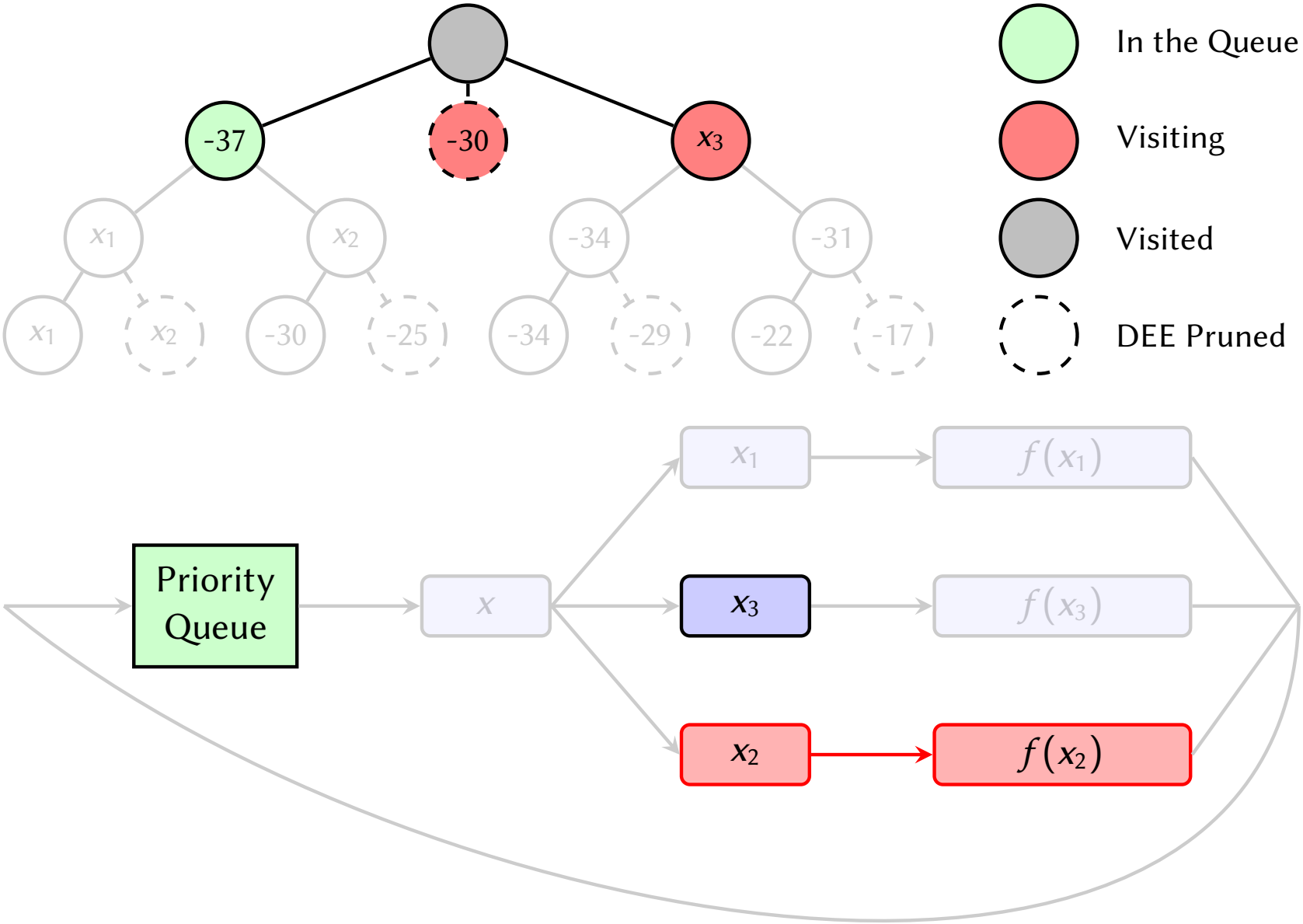
Full A* Search



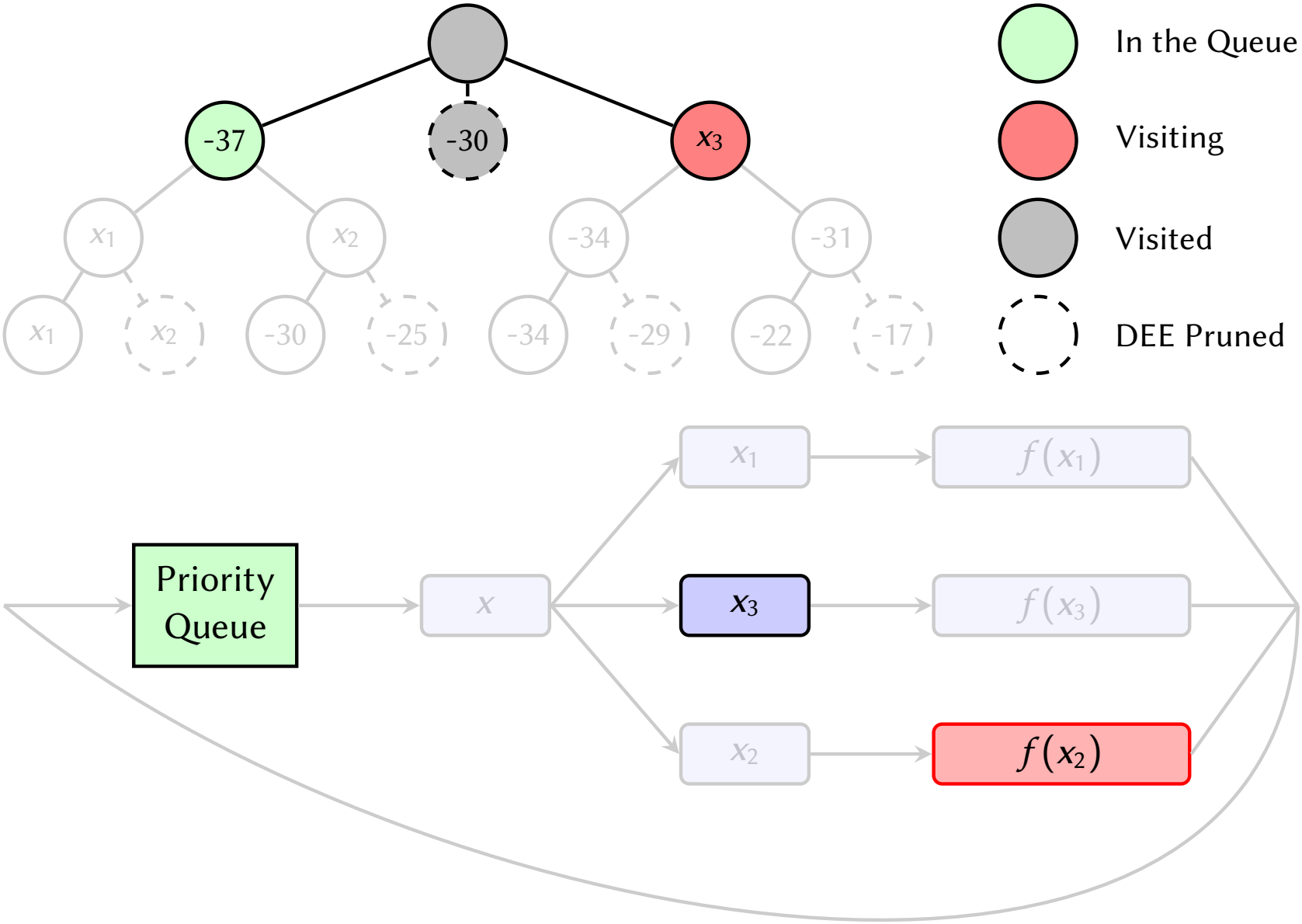
Full A* Search



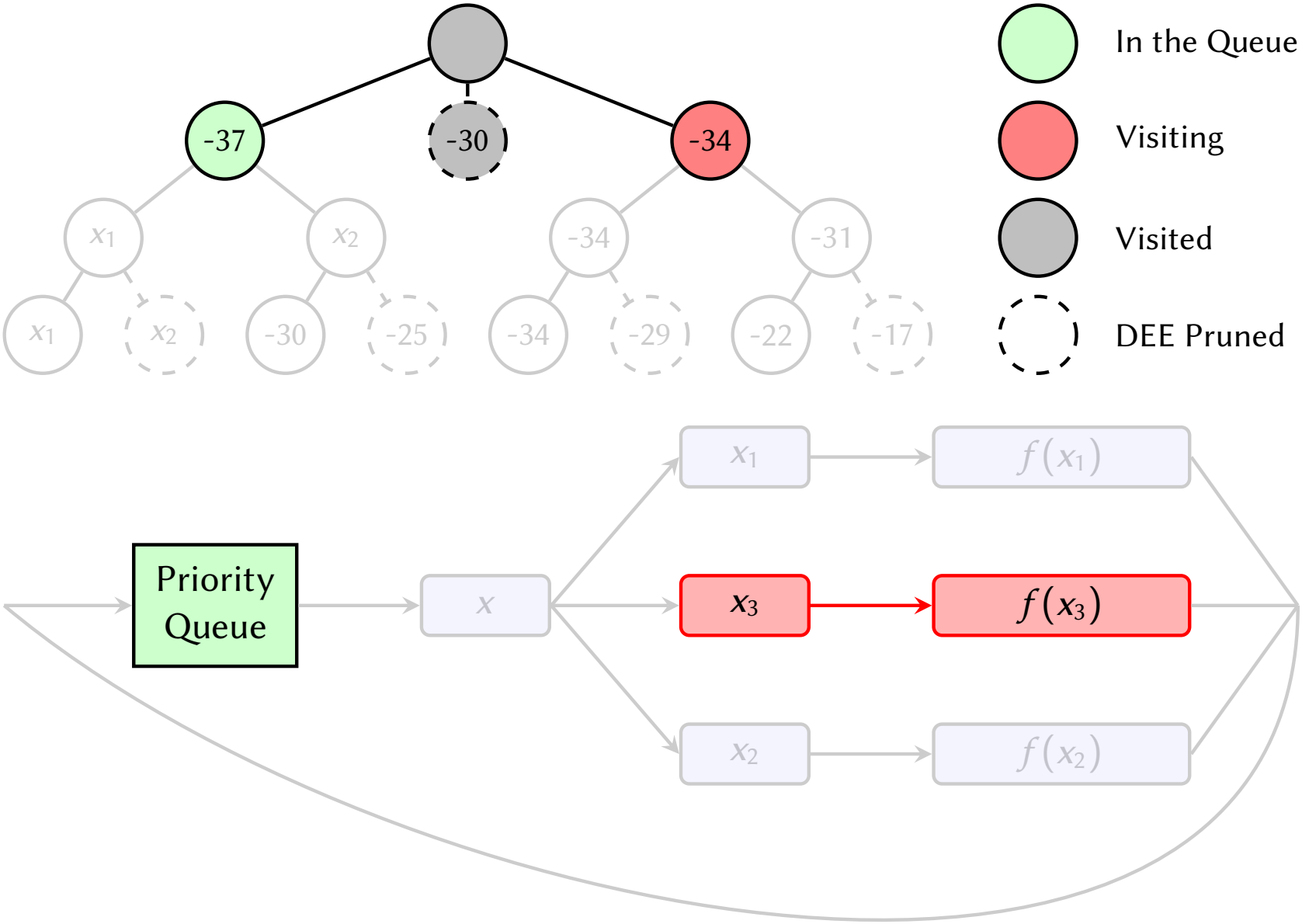
Full A* Search



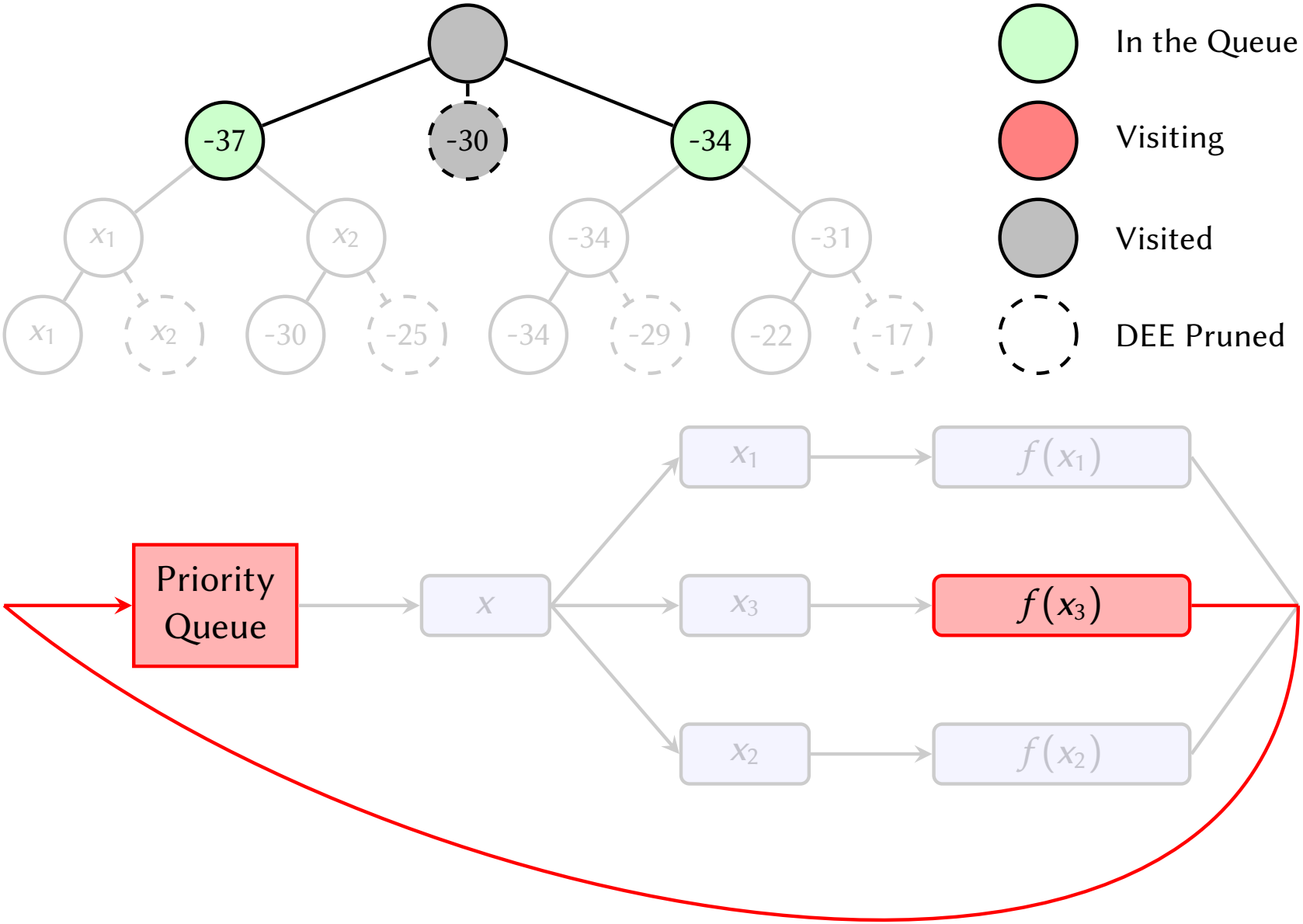
Full A* Search



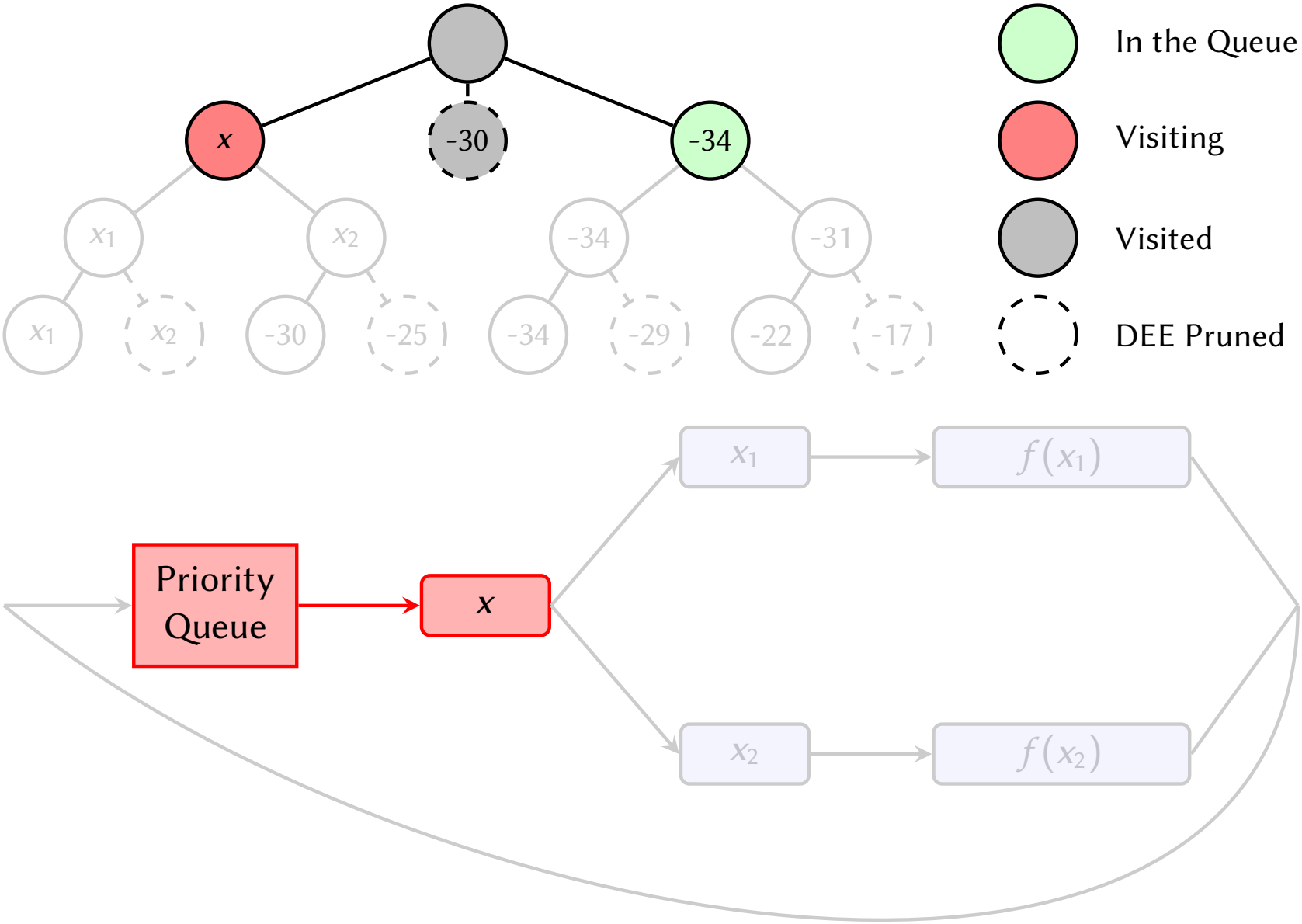
Full A* Search



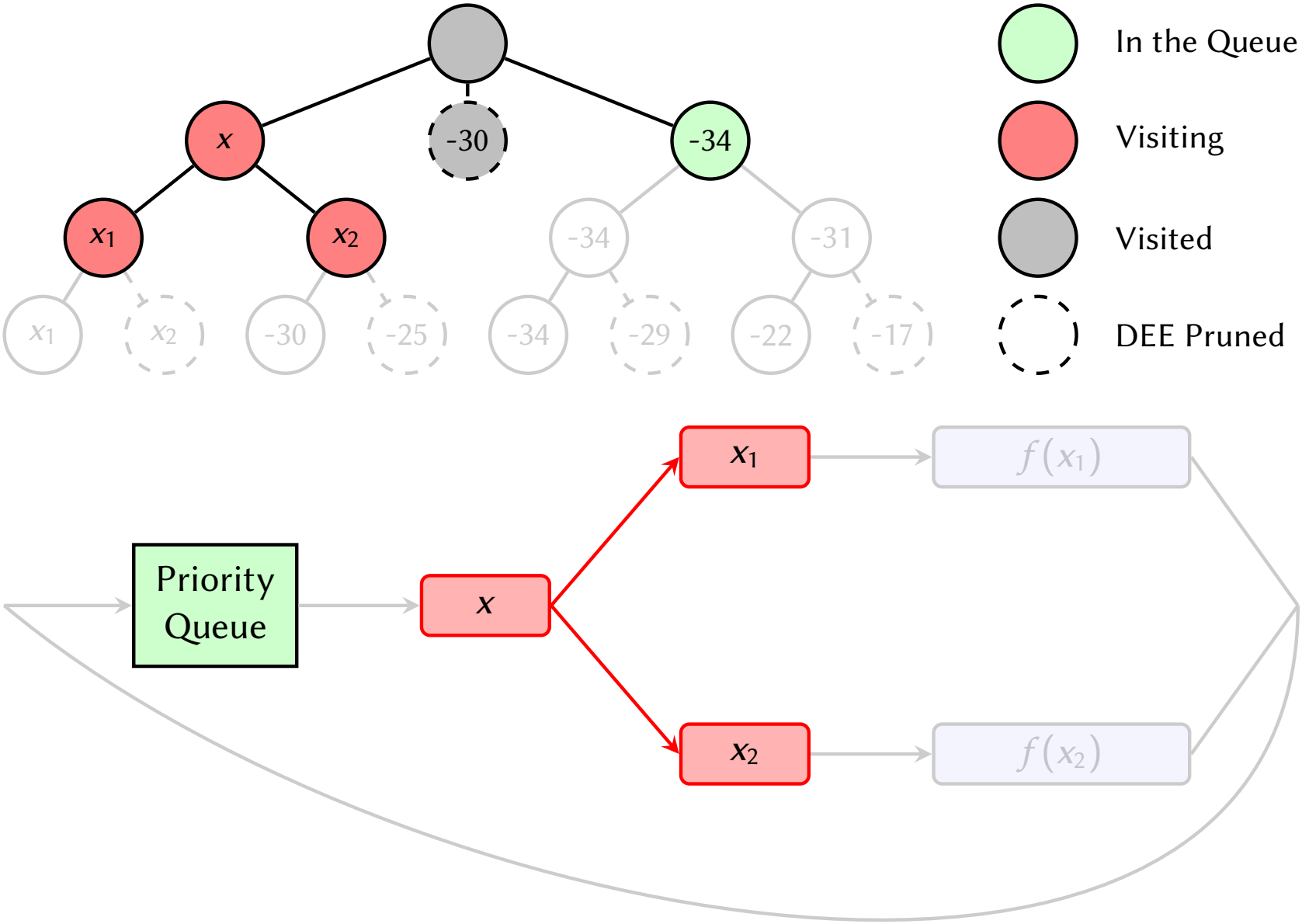
Full A* Search



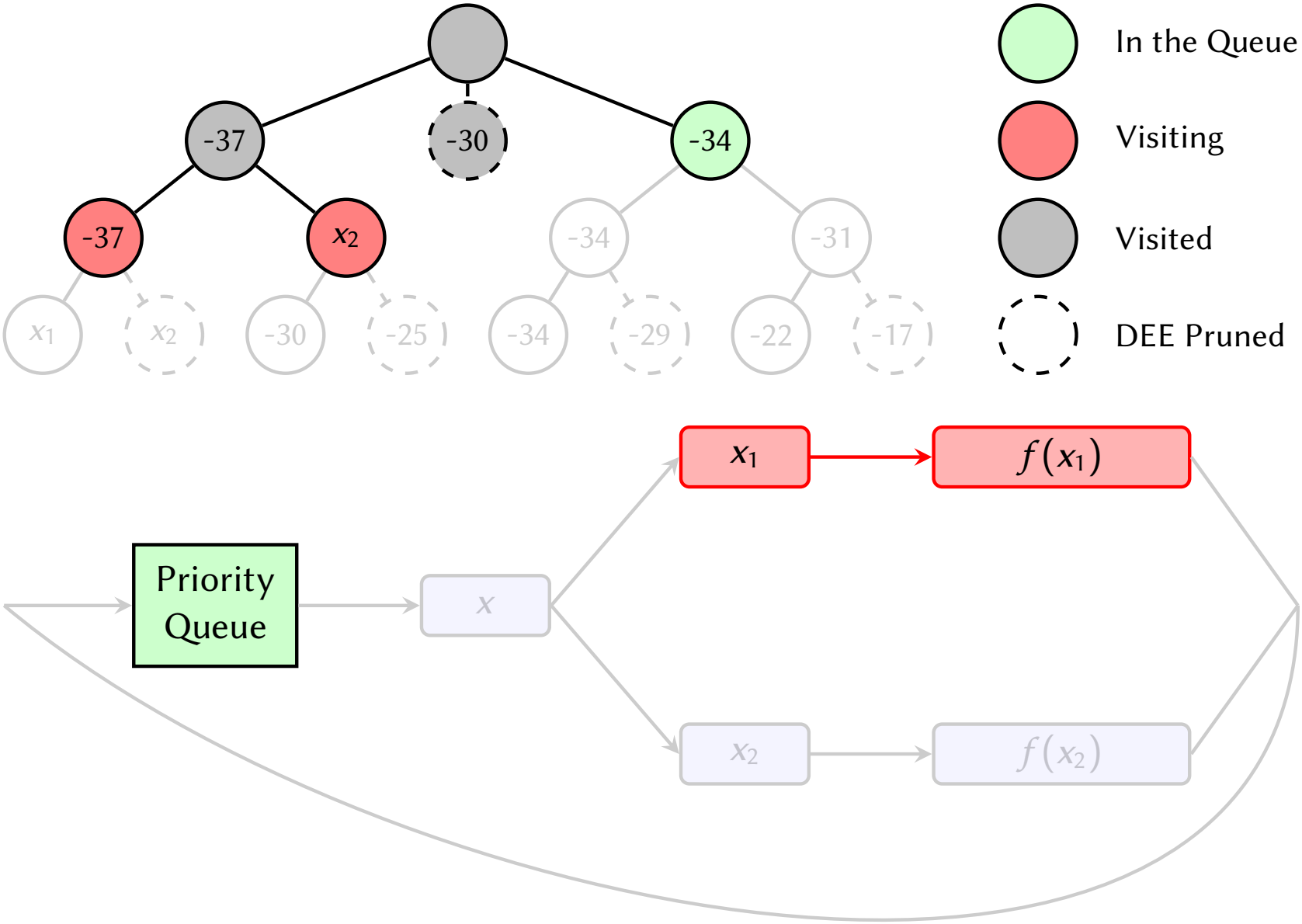
Full A* Search



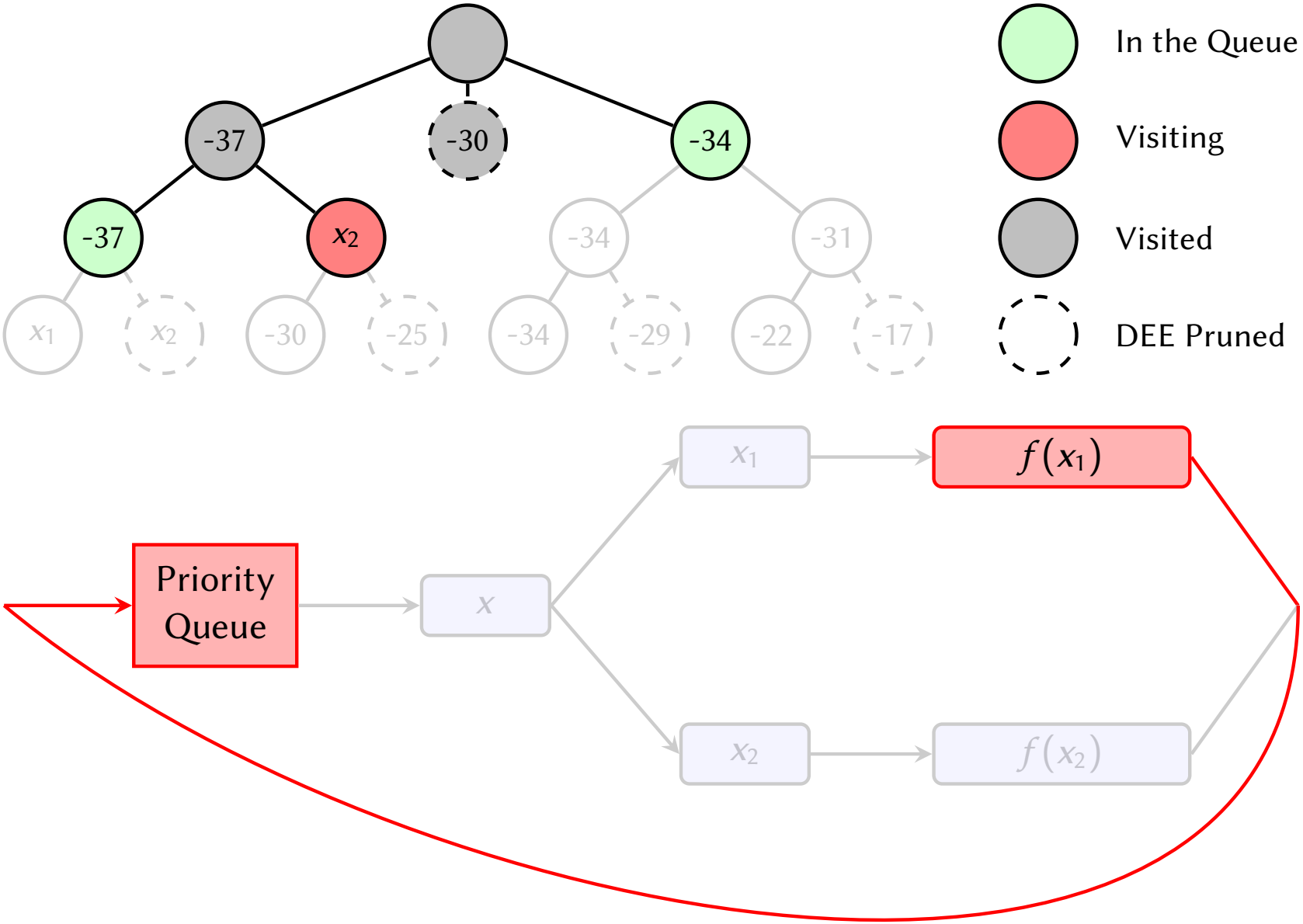
Full A* Search



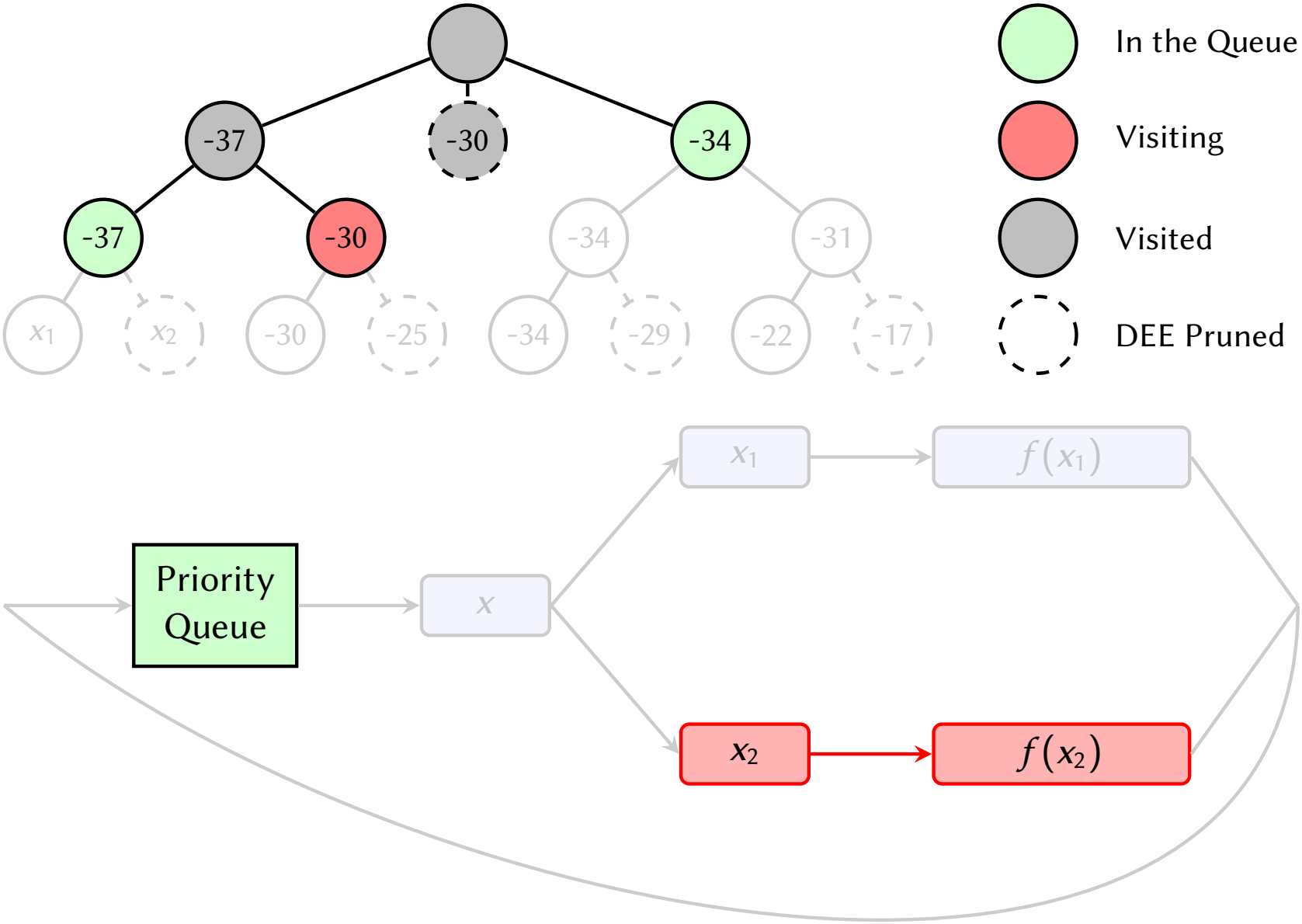
Full A* Search



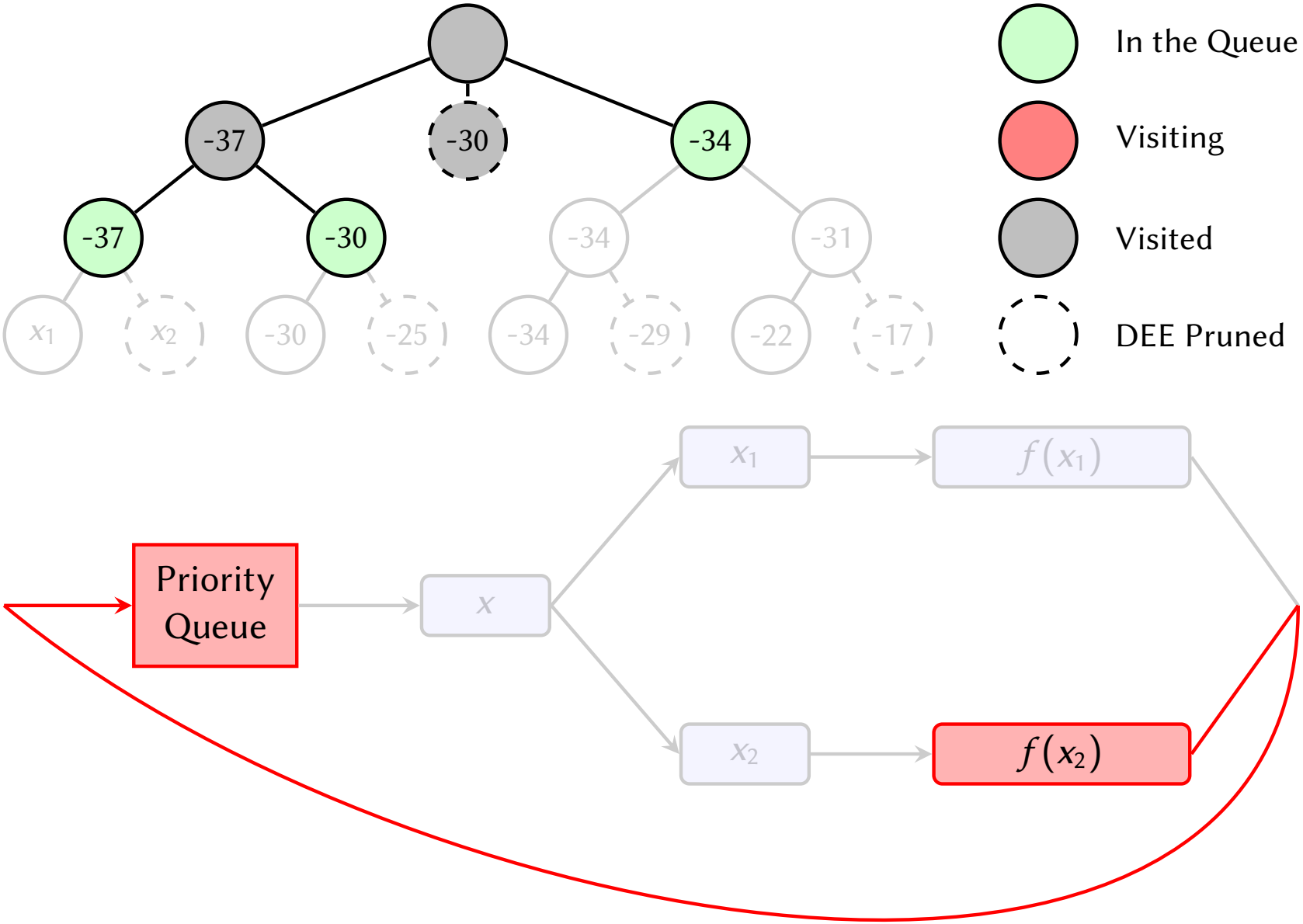
Full A* Search



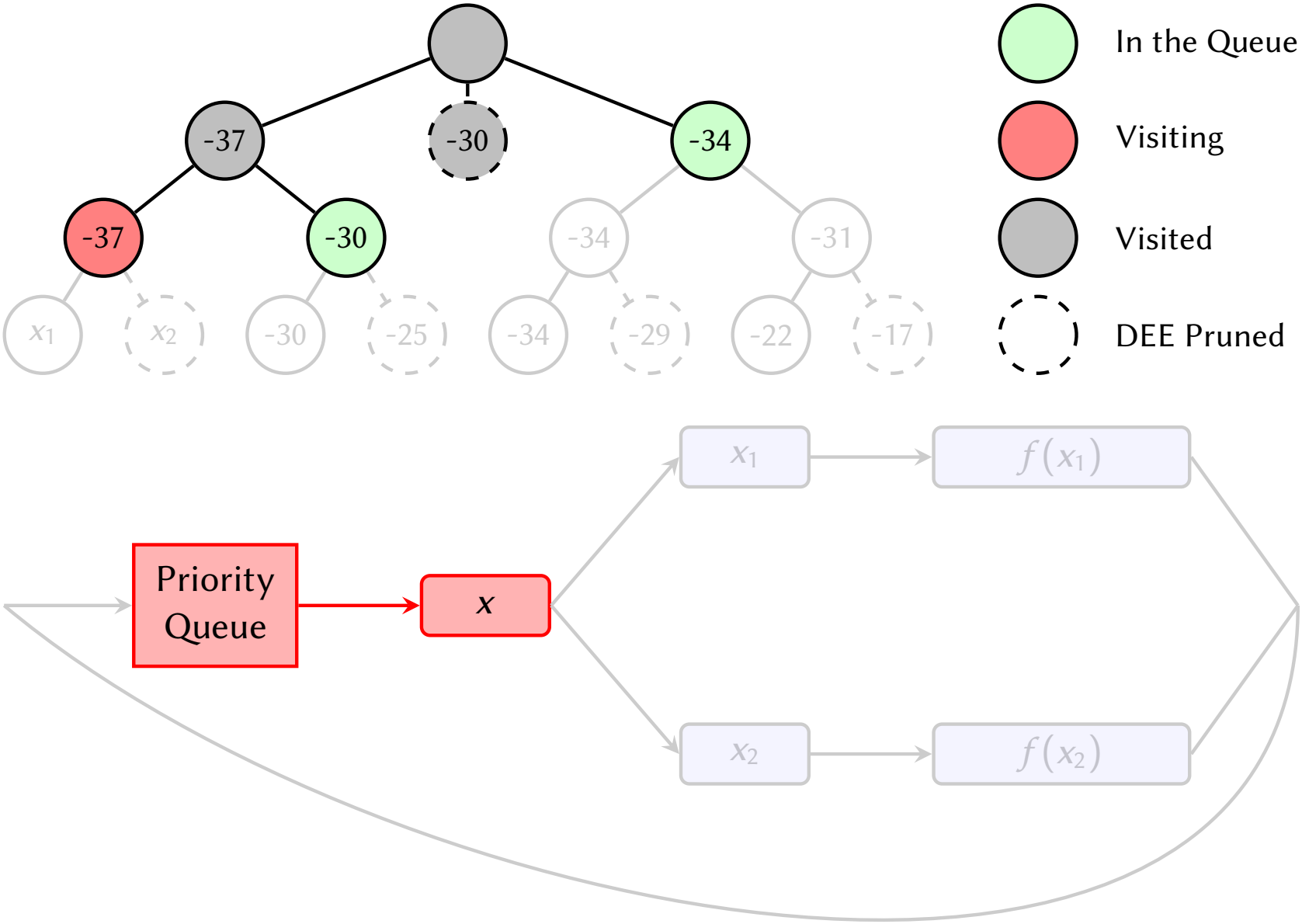
Full A* Search



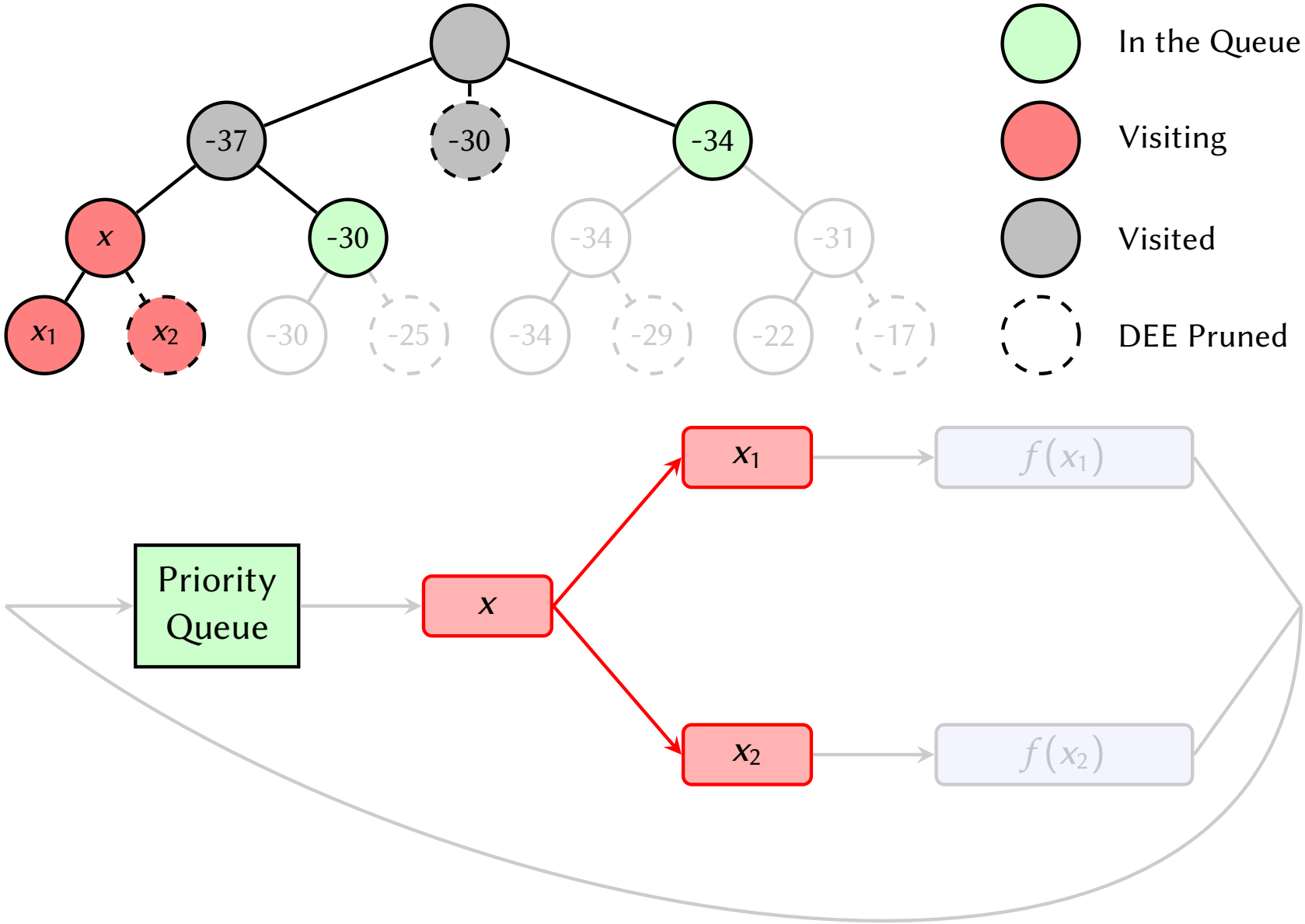
Full A* Search



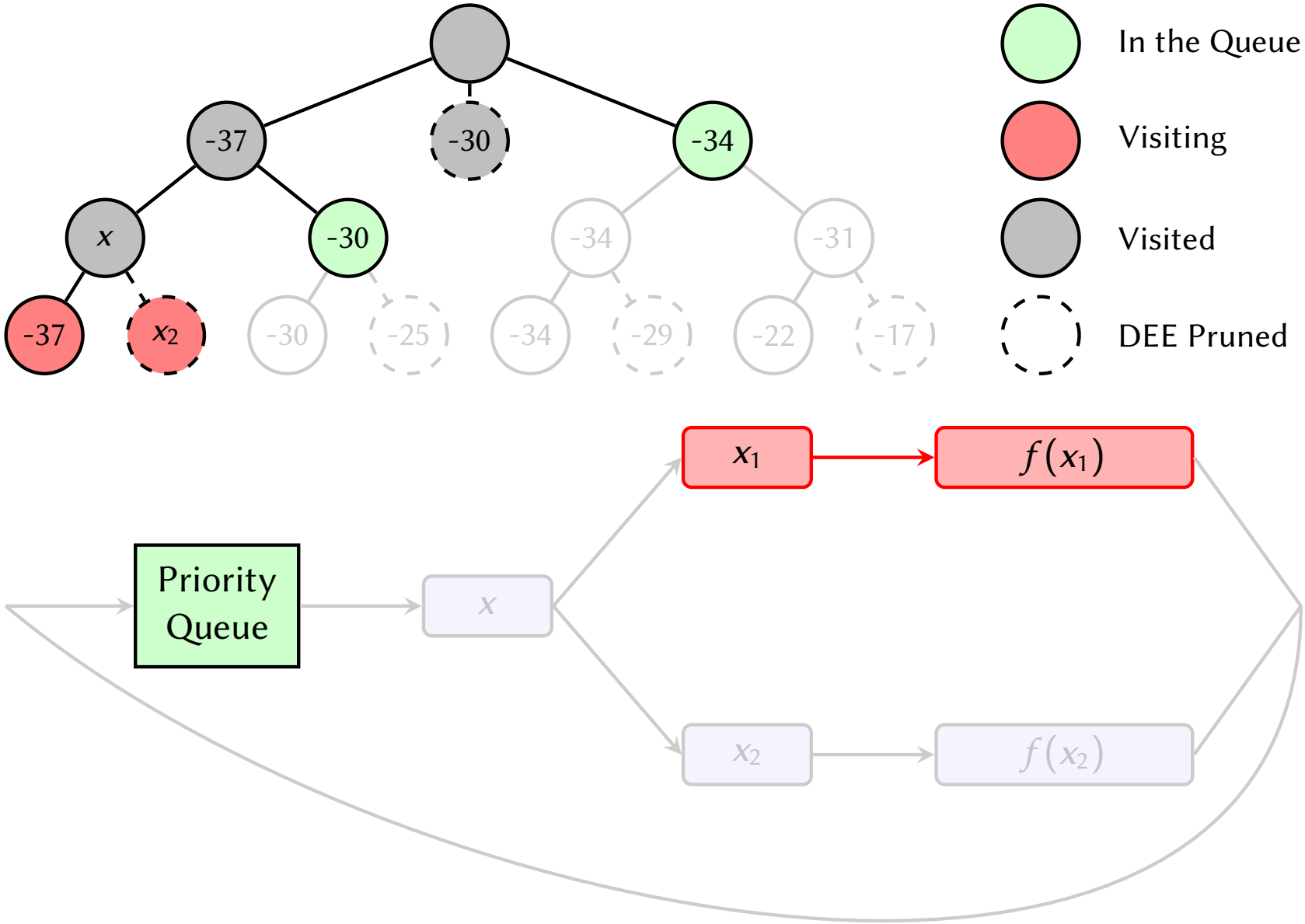
Full A* Search



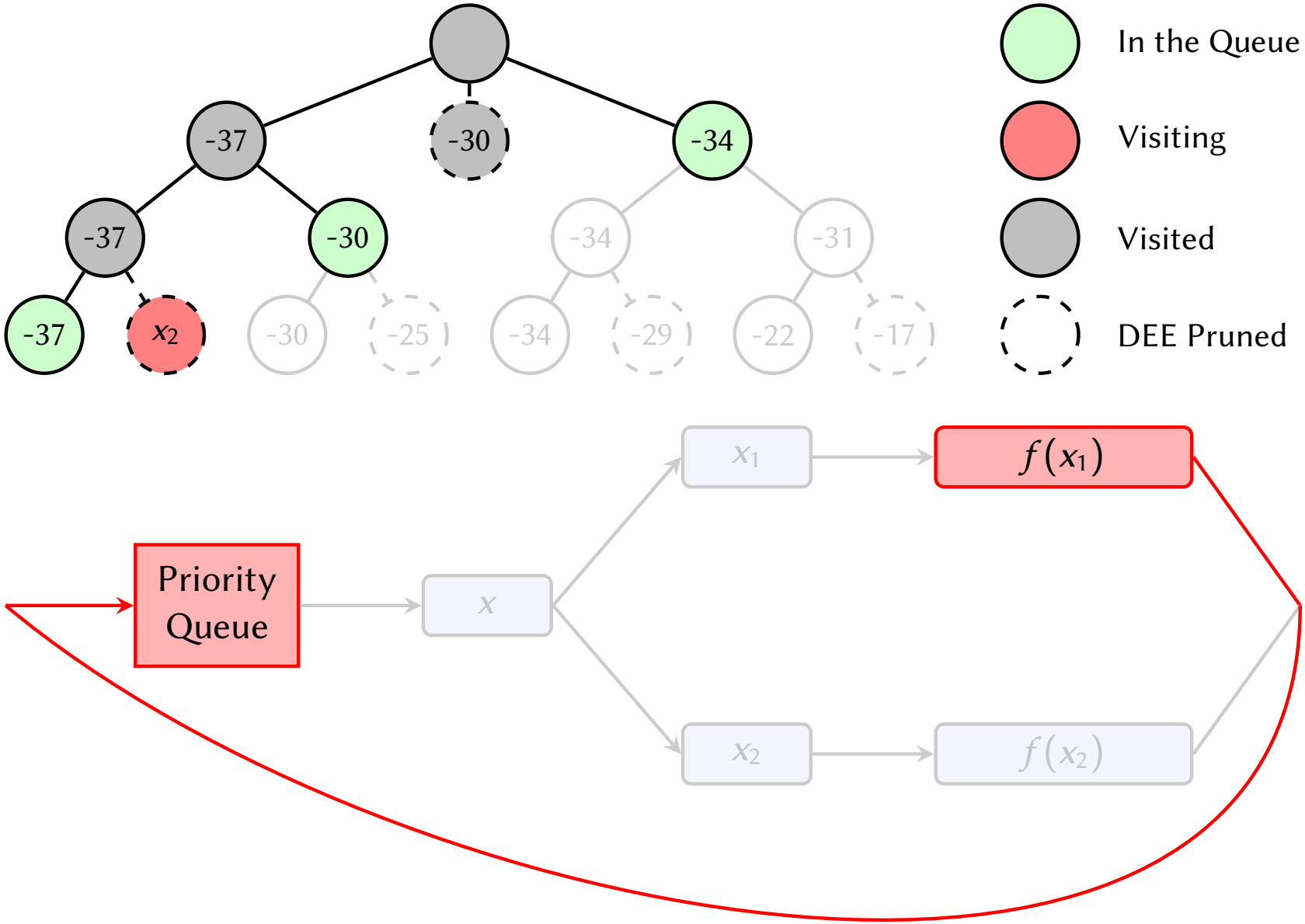
Full A* Search



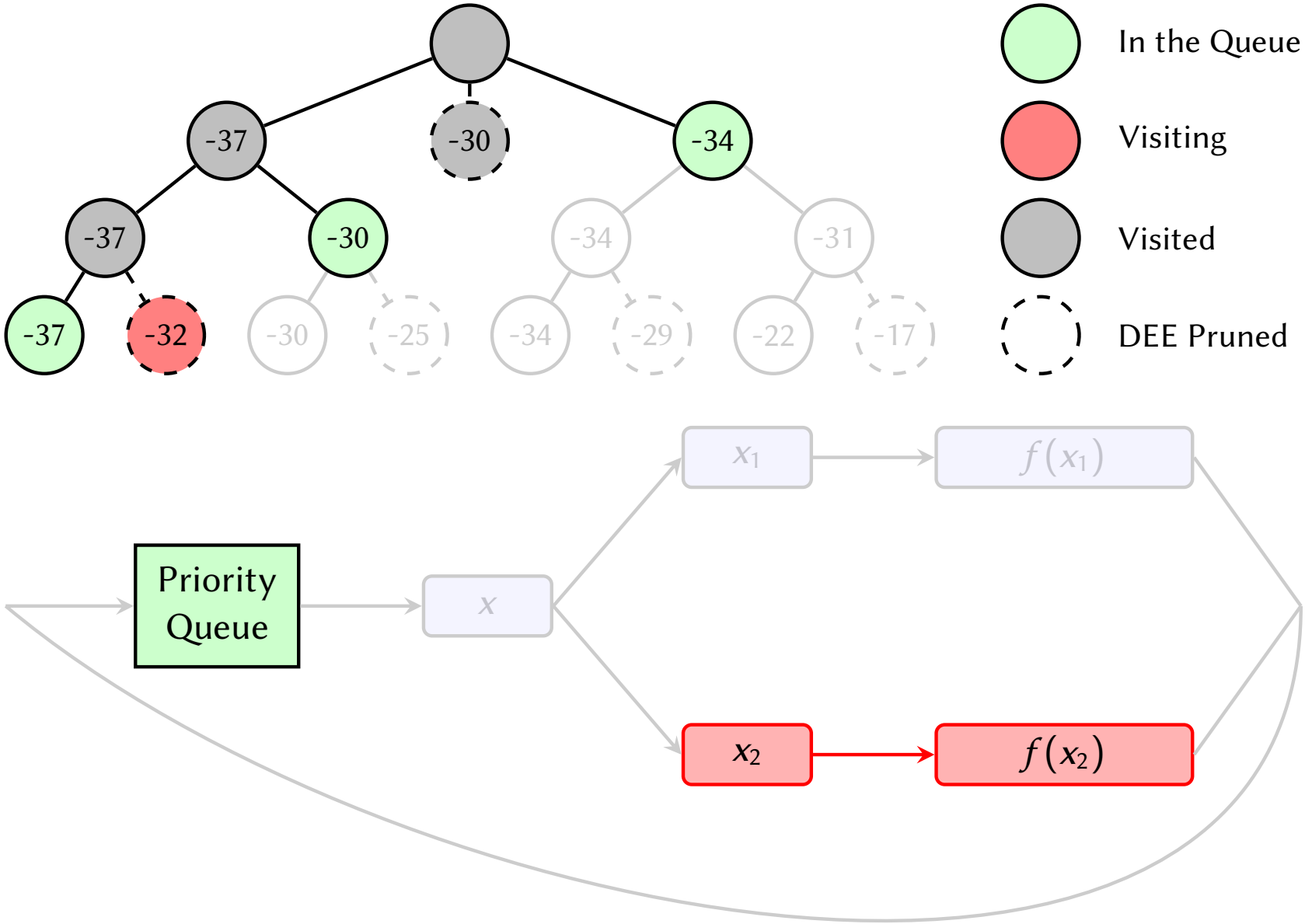
Full A* Search



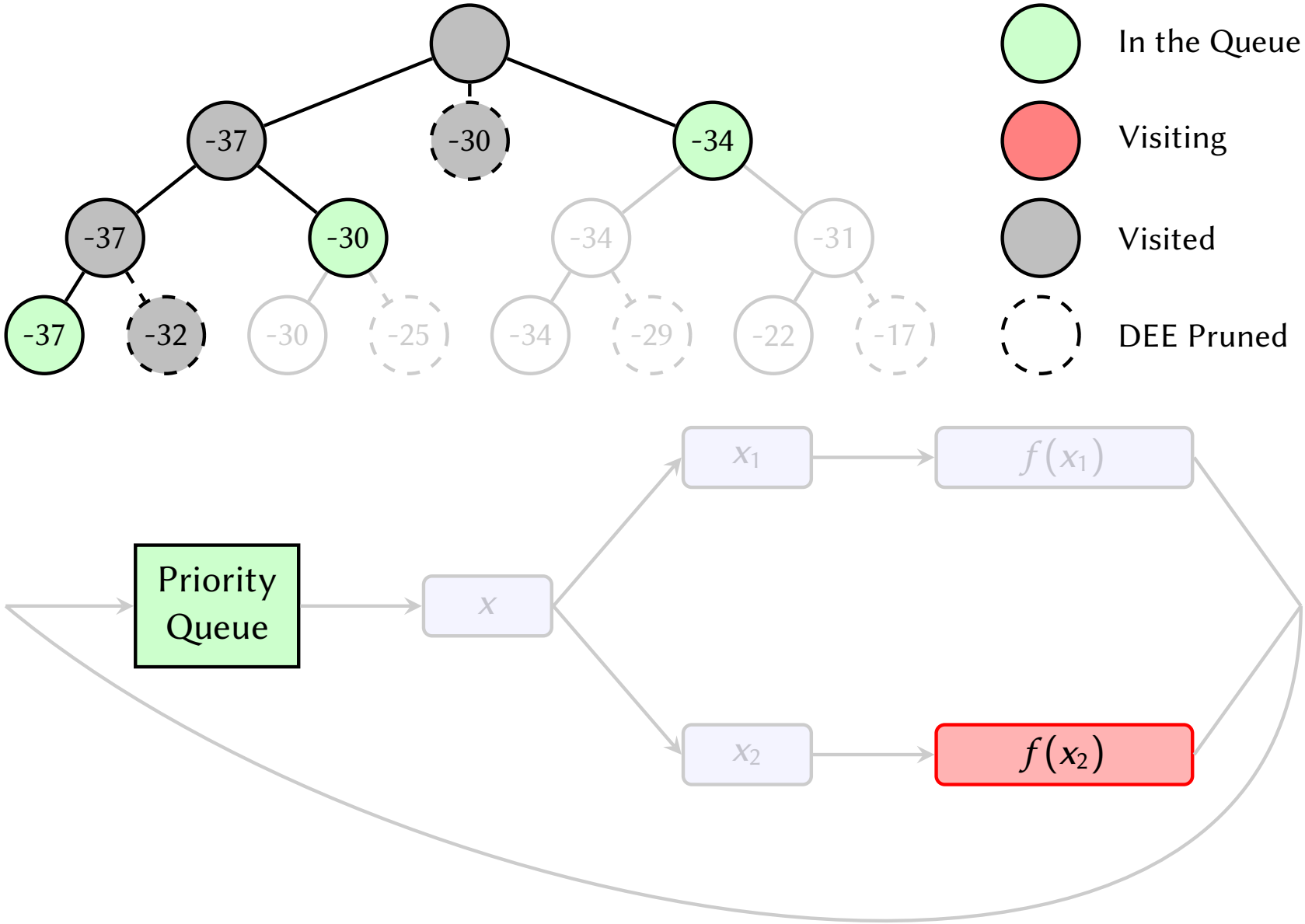
Full A* Search



Full A* Search



Full A* Search



Full A* Search