AGENTS & UNINFORMED SEARCH

Lara J. Martin (she/they)
TA: Aydin Ayanzadeh (he)

9/7/2023

CMSC 671

By the end of class today, you will be able to:

1. Categorize agents based on their capabilities and behavior (Agents)
2. Formulate a search problem (Search)
3. Predict the order of nodes that breadth- and depth-first search will traverse (Search)

Agents slides by Dr. Cassandra Kent & Dr. Mark Riedl

SCHEDULE

Recap

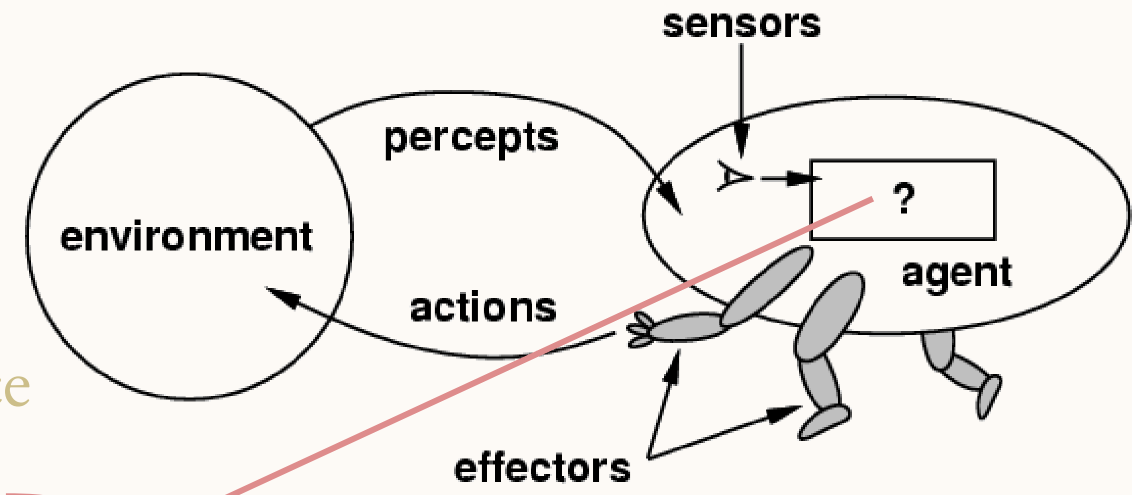
Types of agents

Why search?

Intro to uninformed search

RECAP: AGENT DEFINITION

- **Agent**: anything that perceives its environment through sensors, and acts on its environment through actuators
- **Percept**: input at an instant
- **Percept sequence**: history of inputs
- **Agent function**: mapping of percept sequence to action
- **Agent program**: (concise) implementation of an agent function



Picture from Dr. Cynthia Matuszek, source unknown

RECAP: WAYS TO DESCRIBE ENVIRONMENTS

Fully Observable vs Partially Observable

Deterministic vs Stochastic

Static vs Dynamic

Discrete vs Continuous

Episodic vs Sequential

Single Agent vs Multi Agent

TYPES OF AGENTS

TYPES OF AGENTS

Similar to environments, we can come up with a taxonomy of agents based on how they behave:

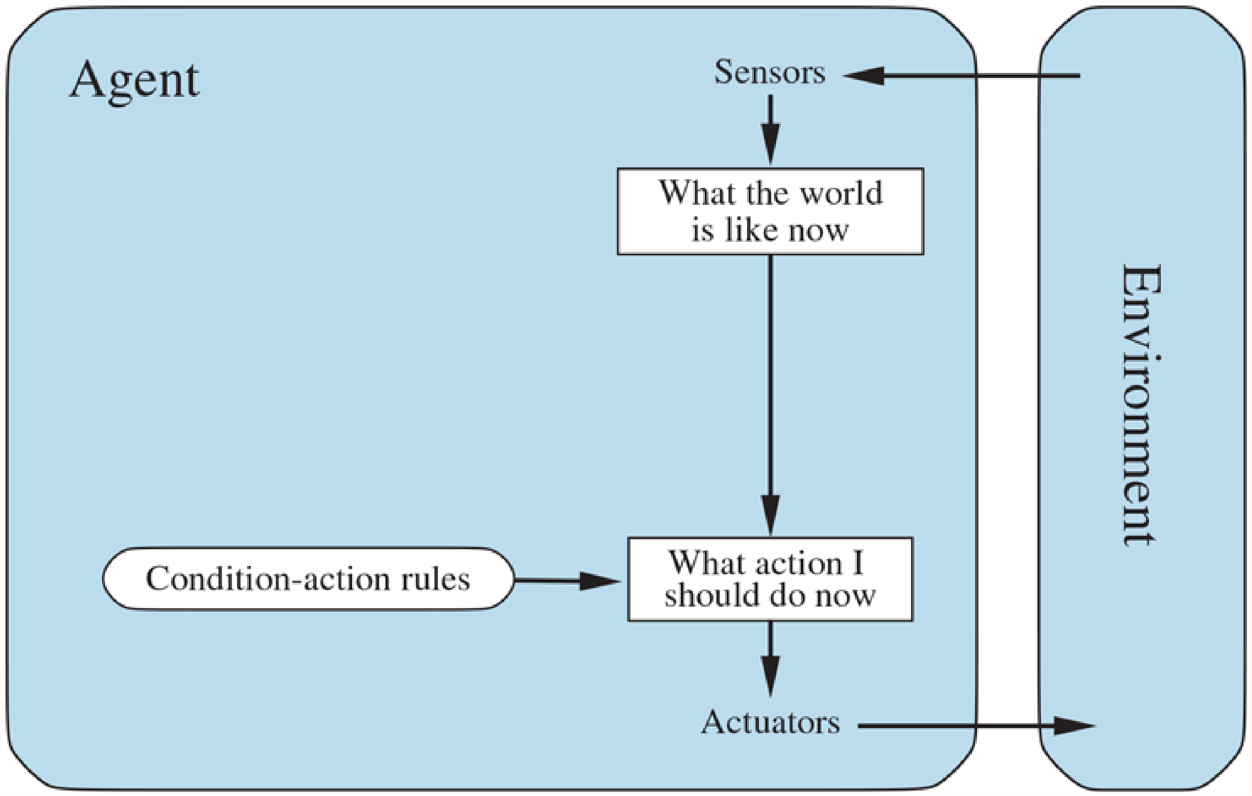
1. Simple Reflex Agent
2. Model-based Reflex Agent
3. Model-based Goal-based Agent
4. Model-based Utility-based Agent

TYPES OF AGENTS: SIMPLE REFLEX AGENT

Decide actions directly from the current percept sequence

Example:

```
if [A, Empty]: return Right
if [B, Empty]: return Left
if [A or B, Human]: return Suck
```



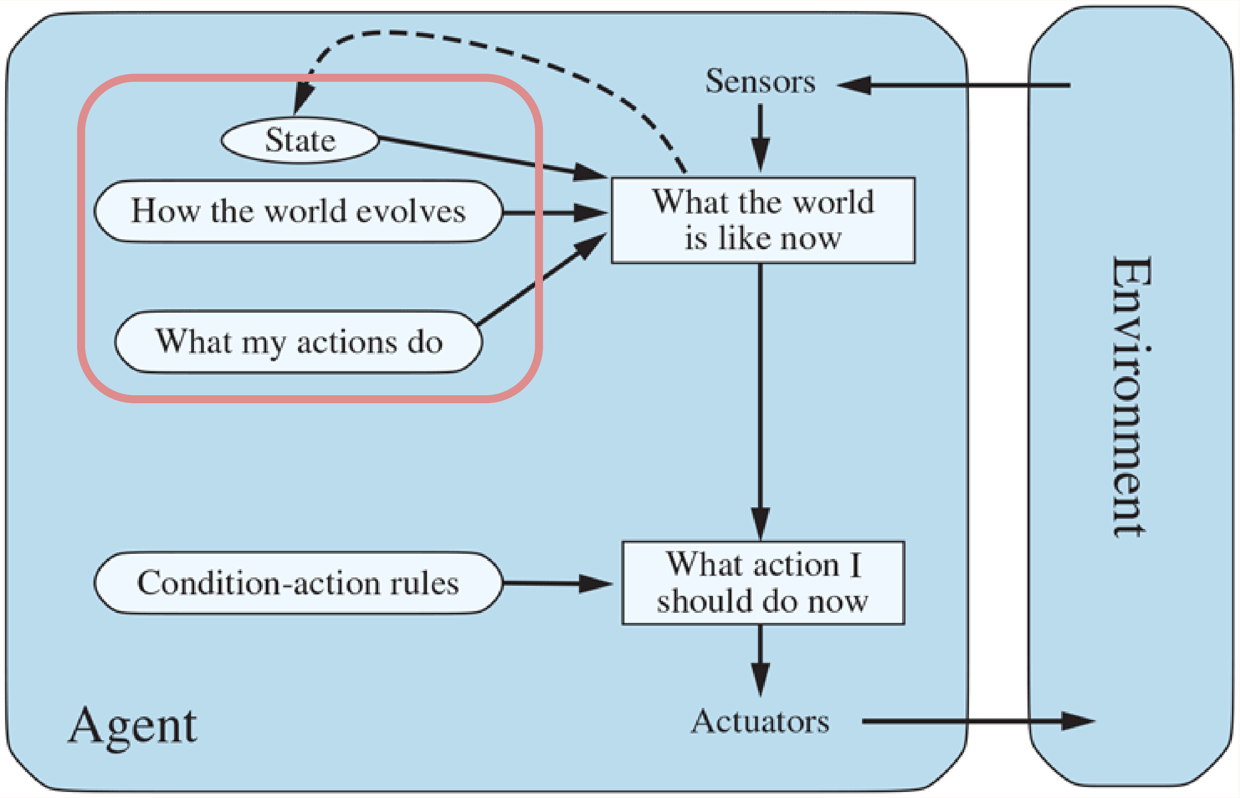
Pg. 50

TYPES OF AGENTS: MODEL-BASED REFLEX AGENT

Attempt to model the state of the world, decide action based on modeled ("remembered") state

Example:

- Observe that at a certain time, people are more likely to show up in A
- Move to location based on time of day



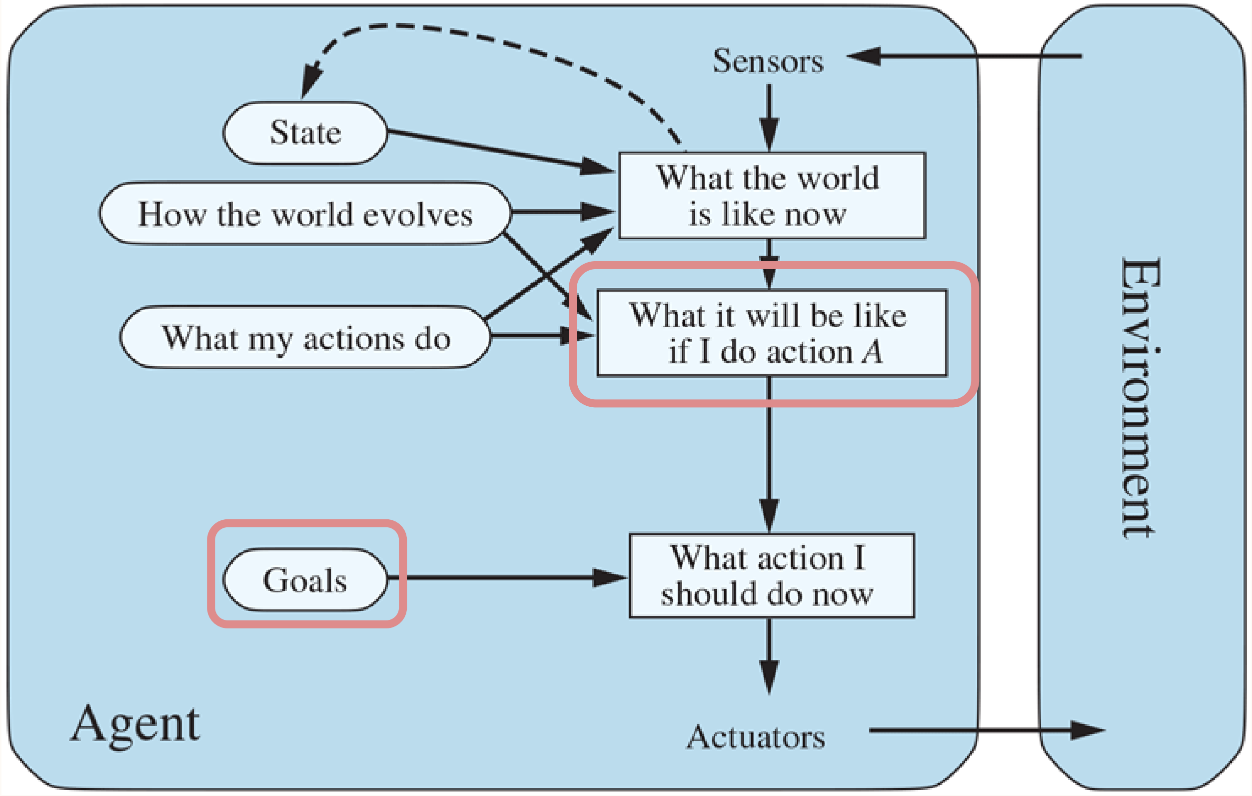
Pg. 52

TYPES OF AGENTS: MODEL-BASED GOAL-BASED AGENT

Action choices change as the goal changes

Example:

- For the same performance criteria (suck as much blood as possible), we can add a goal of keeping one room clear to act as a home
- As the goal changes, vampire changes how often it moves



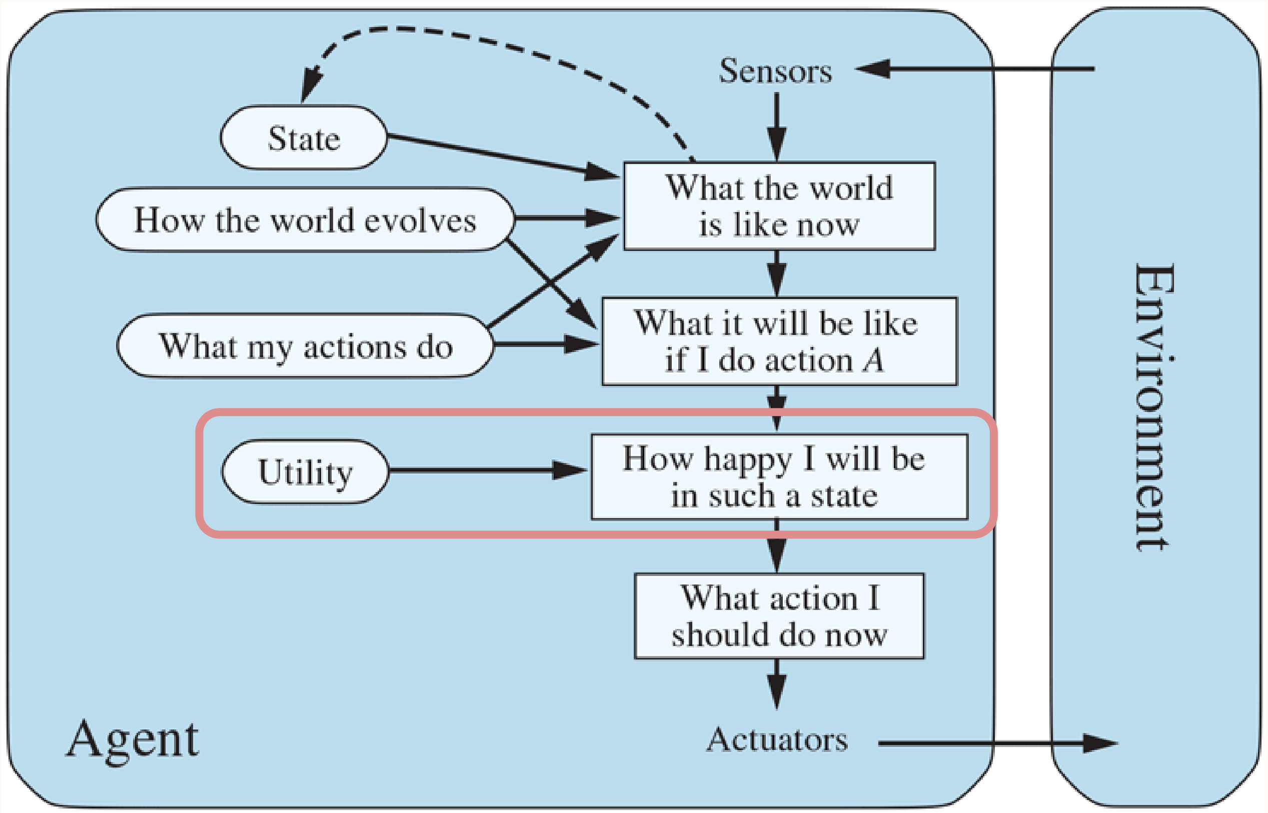
Pg. 53

TYPES OF AGENTS: MODEL-BASED UTILITY-BASED AGENT

Agent cares not only about reaching a goal, but also *how* it reaches its goal

Example:

- Energy-efficient vampire agent
- Still wants to suck as much blood as possible
- Limits how often it moves



Pg. 55

WHAT TYPE OF AGENT IS A ROOMBA?

1. Simple Reflex Agent
2. Model-based Reflex Agent ← Keeps a memory of the world but only reacts to its surroundings
3. Model-based Goal-based Agent
4. Model-based Utility-based Agent

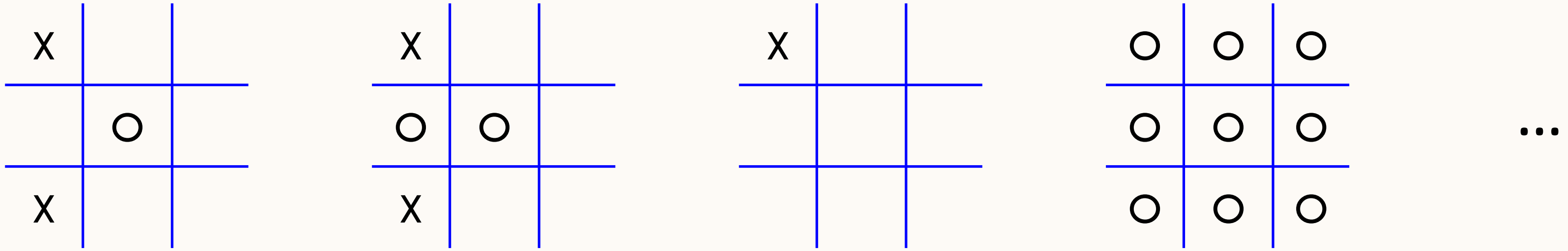
WHAT TYPE OF AGENT IS WATSON?

1. Simple Reflex Agent
2. Model-based Reflex Agent
3. Model-based Goal-based Agent
4. Model-based Utility-based Agent ← Not just to win but to get the most money

UNINFORMED SEARCH

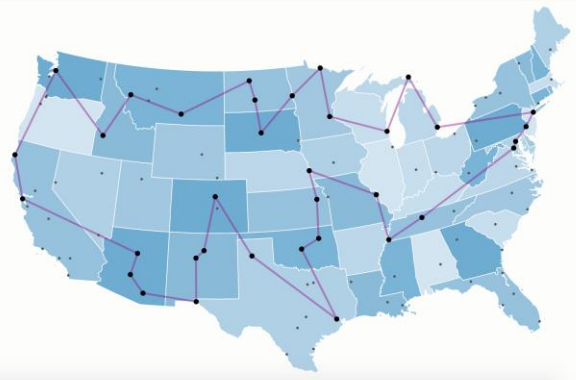
WHY SEARCH?

- Some problems are small enough to go through all possible states:



Tic-tac-toe: $3^9 = 19{,}683$ states (3 values for each cell, nine cells) (With fewer *valid* states)

- Most real-world problems are intractable
  - We need to be smart about how we visit states

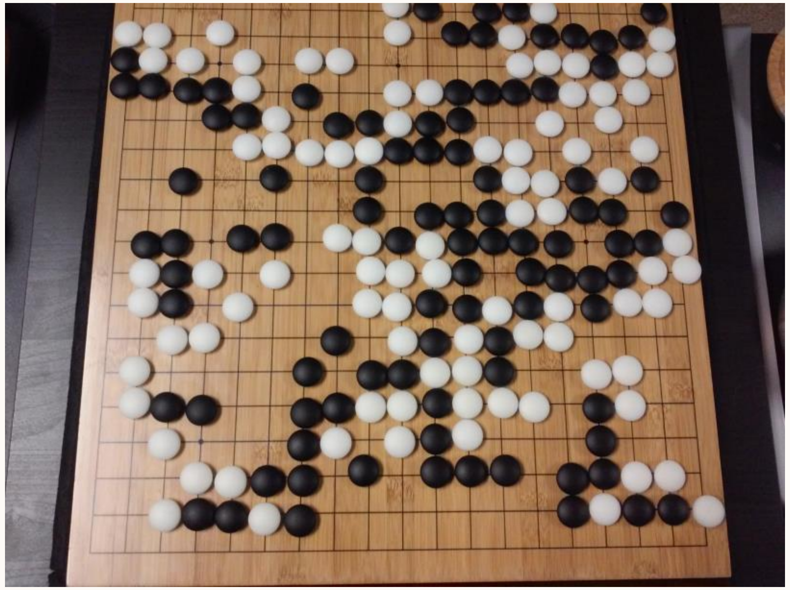


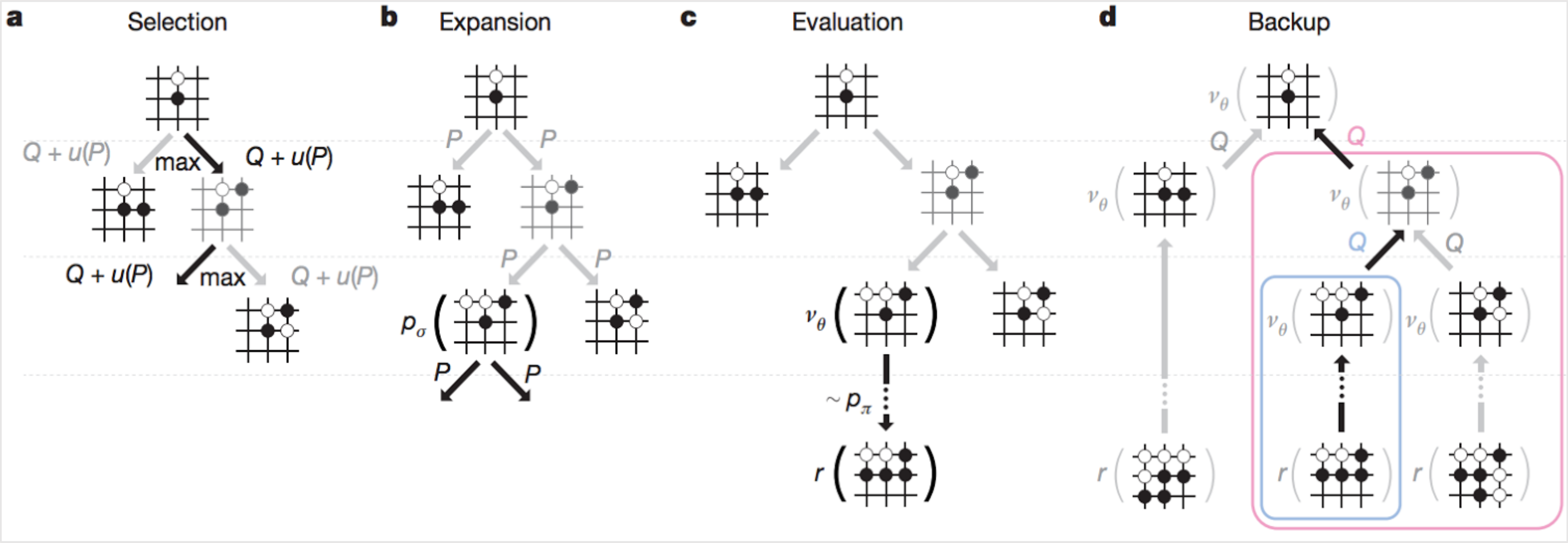
Modified from slides by Dr. Cynthia Matuszek
examples.gurobi.com/traveling-salesman-problem

WHY SEARCH?

For when we don’t know how to reach a goal (or **any** goal) from a given initial state (or **any** initial state)

SEARCH TODAY: ALPHAGO





Pictures from Cynthia Matuszek via needpix.com, machinelearnings.co/understanding-alphago-948607845bb1

SEARCH TODAY: STORY GENERATION

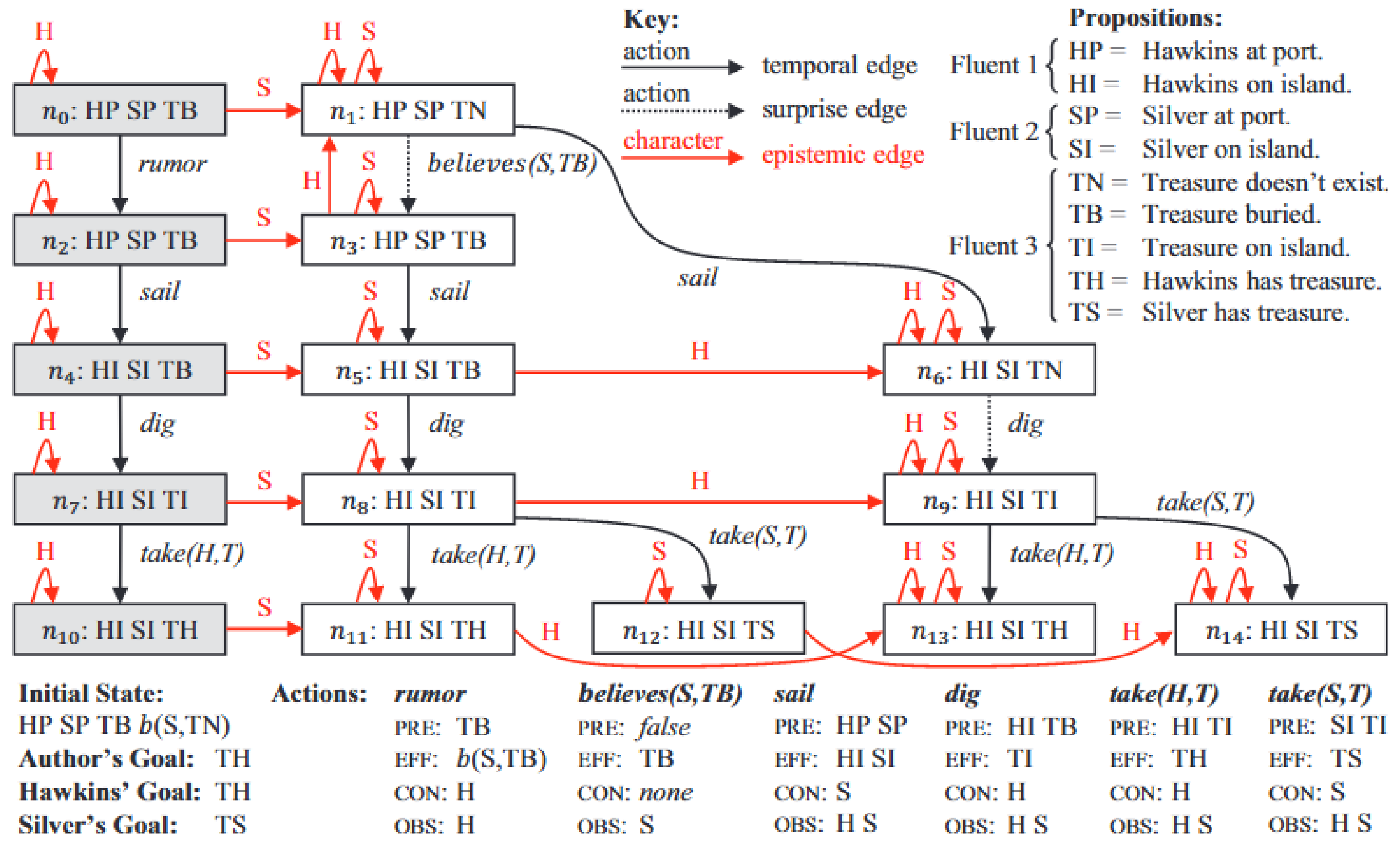


Figure 1: A narrative search space graph for the plot of *Treasure Island* (Stevenson 1919).

From "A Possible Worlds Model of Belief for State-Space Narrative Planning" by Shirvani et al. @ AIIDE 2017

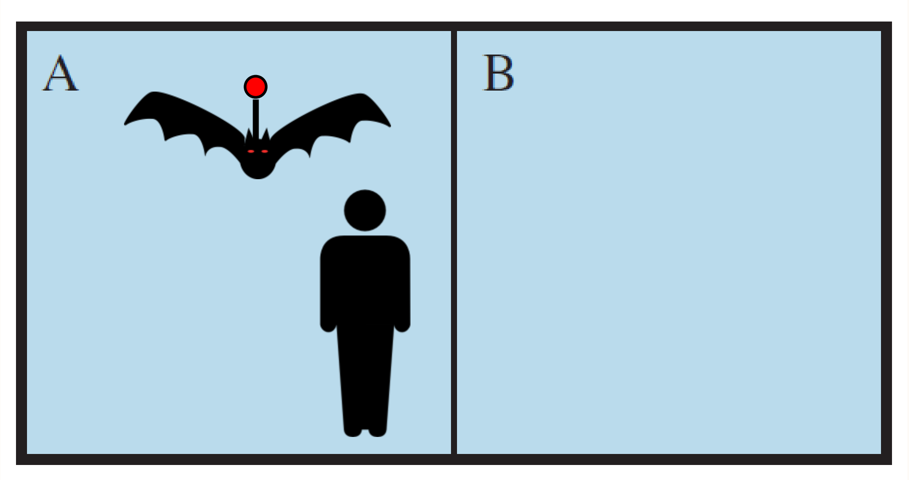
DEFINING A SEARCH PROBLEM

- Break down the problem space into states and actions (for each state)
- + these components:
- Initial state – where the agent starts
- Transition model – how actions change states
- Goal test – measuring if the current state is the goal
- Step cost (optional) – how expensive it is to take action *a* in state *s*

Problem: Find a sequence of actions that **transitions** the **initial state** into a state which passes the **goal test**
Solution: sequence of actions, or a **plan**

WHAT ARE STATES?

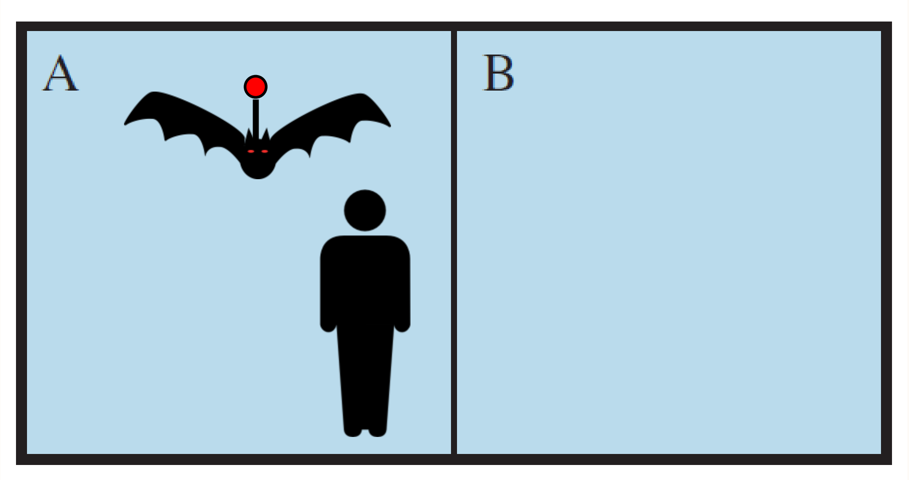
- Current position and attributes of everything in the environment
  - Only need the things relevant to the agent's decision making
- Like discrete snapshots over time



<u>State</u>
Location A: vampire, human
Location B: empty

WHAT ARE ACTIONS?

- Anything the agent can do to affect the environment



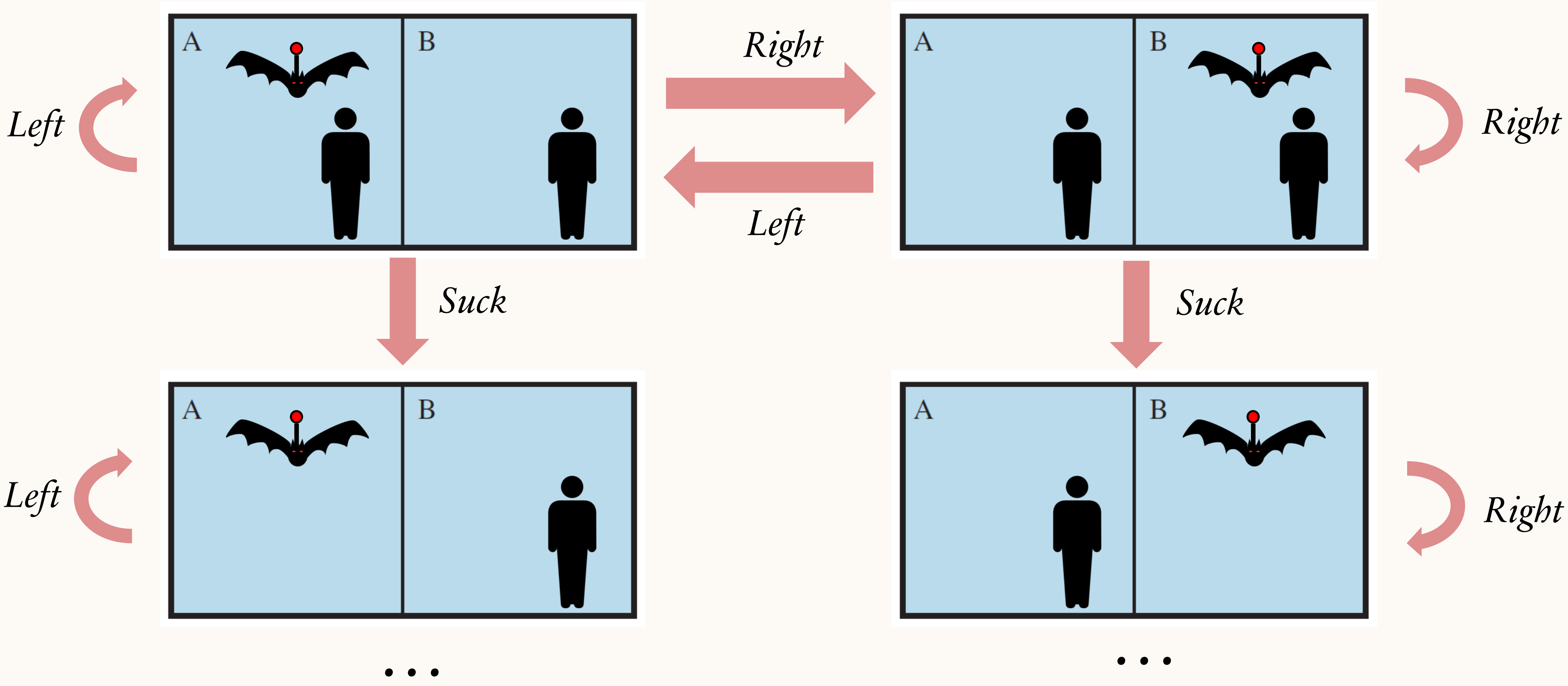
Actions

move left, move right, suck

Or for this particular state…

move right, suck

TRANSITION MODEL EXAMPLE

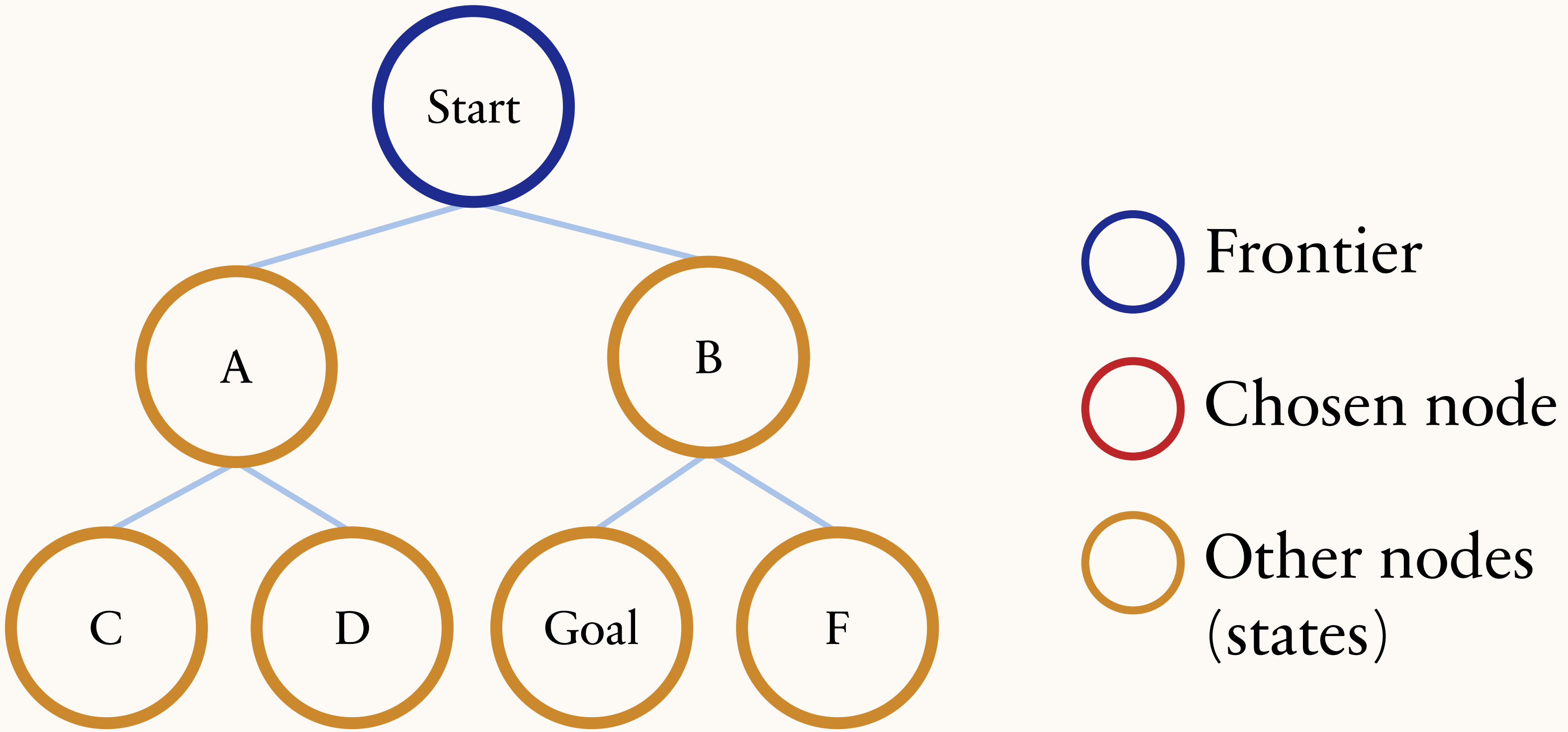


From Dr. Cassandra Kent

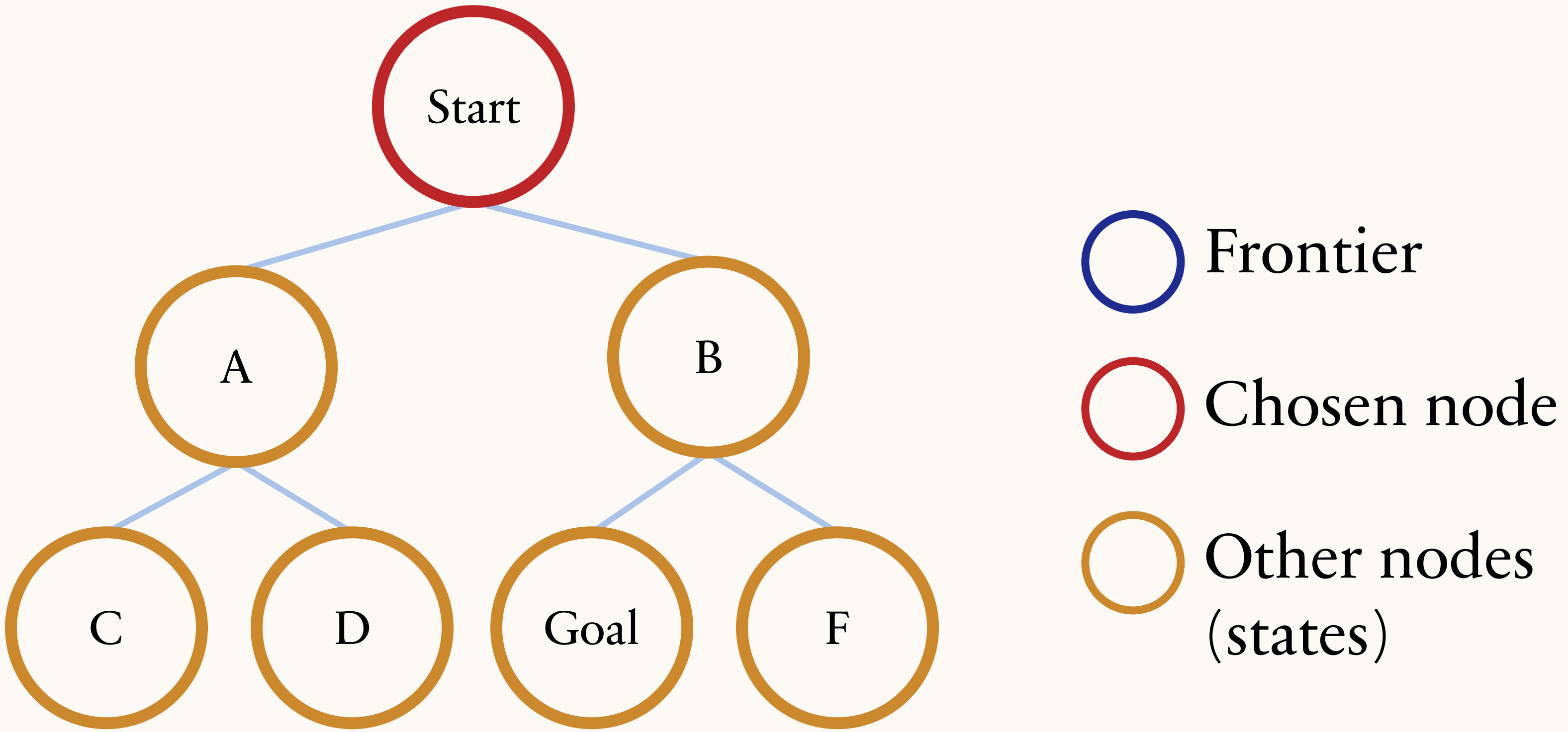
GENERIC SEARCH ALGORITHM™

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     expand the chosen node, adding the resulting nodes to the frontier
```

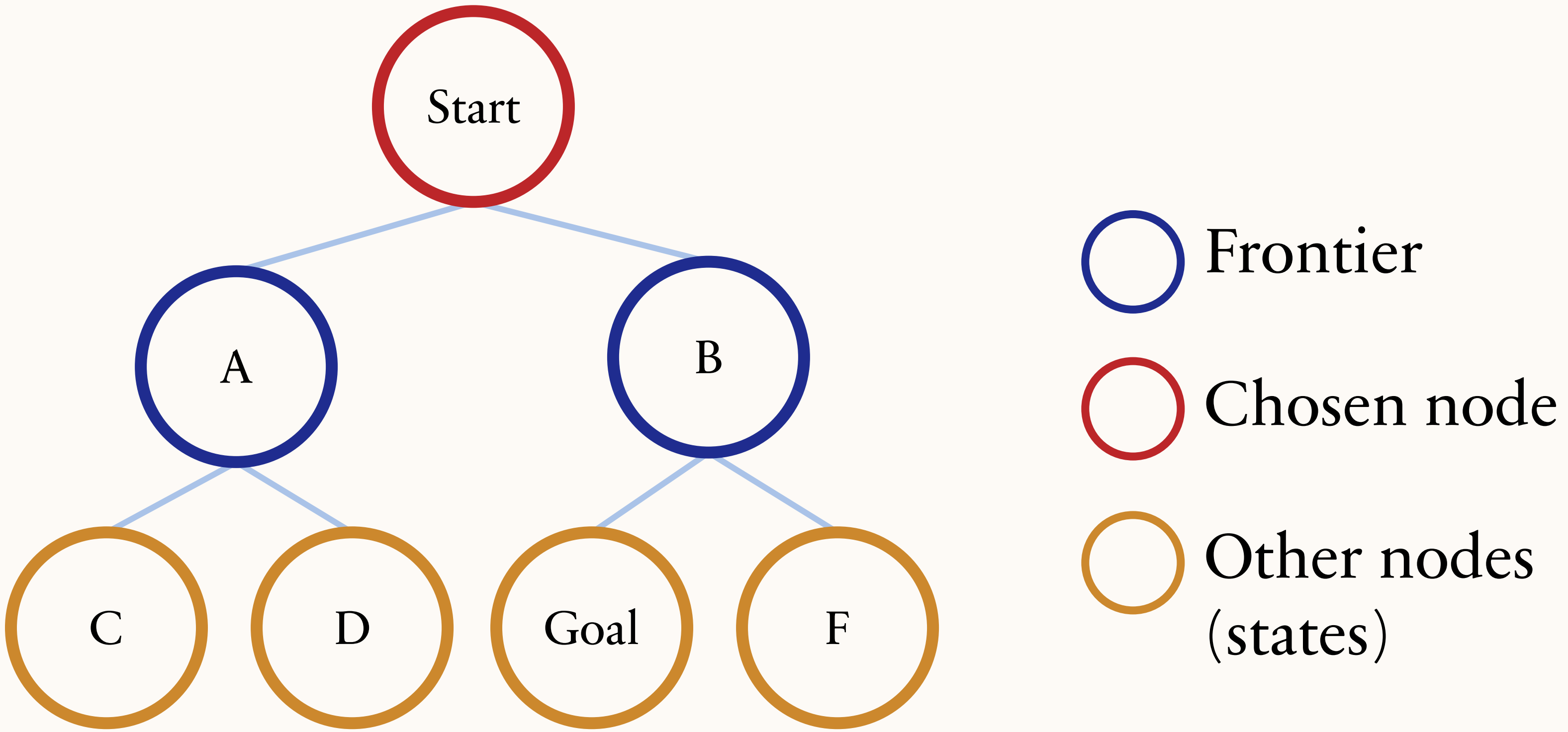
THE GENERIC SEARCH ALGORITHM



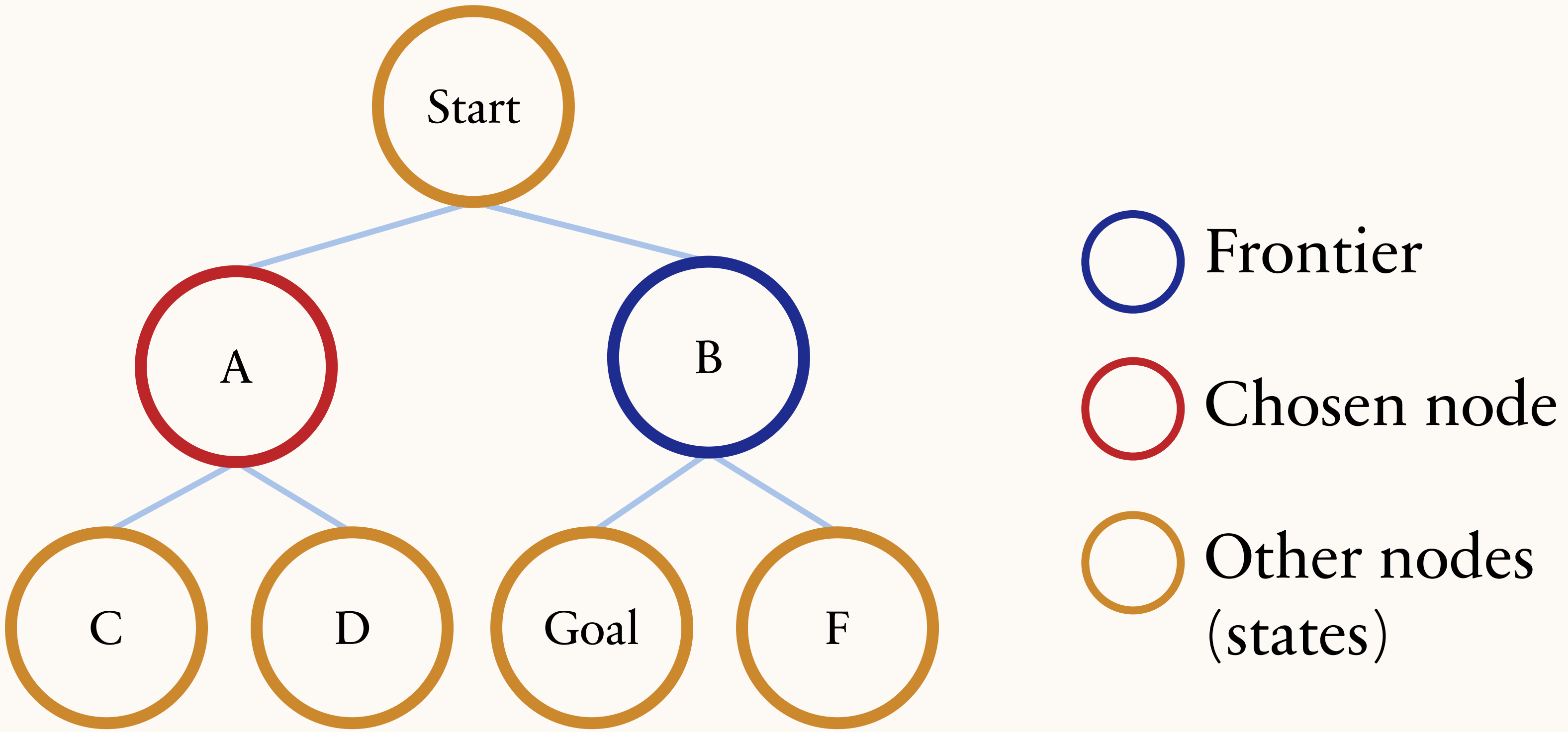
THE GENERIC SEARCH ALGORITHM



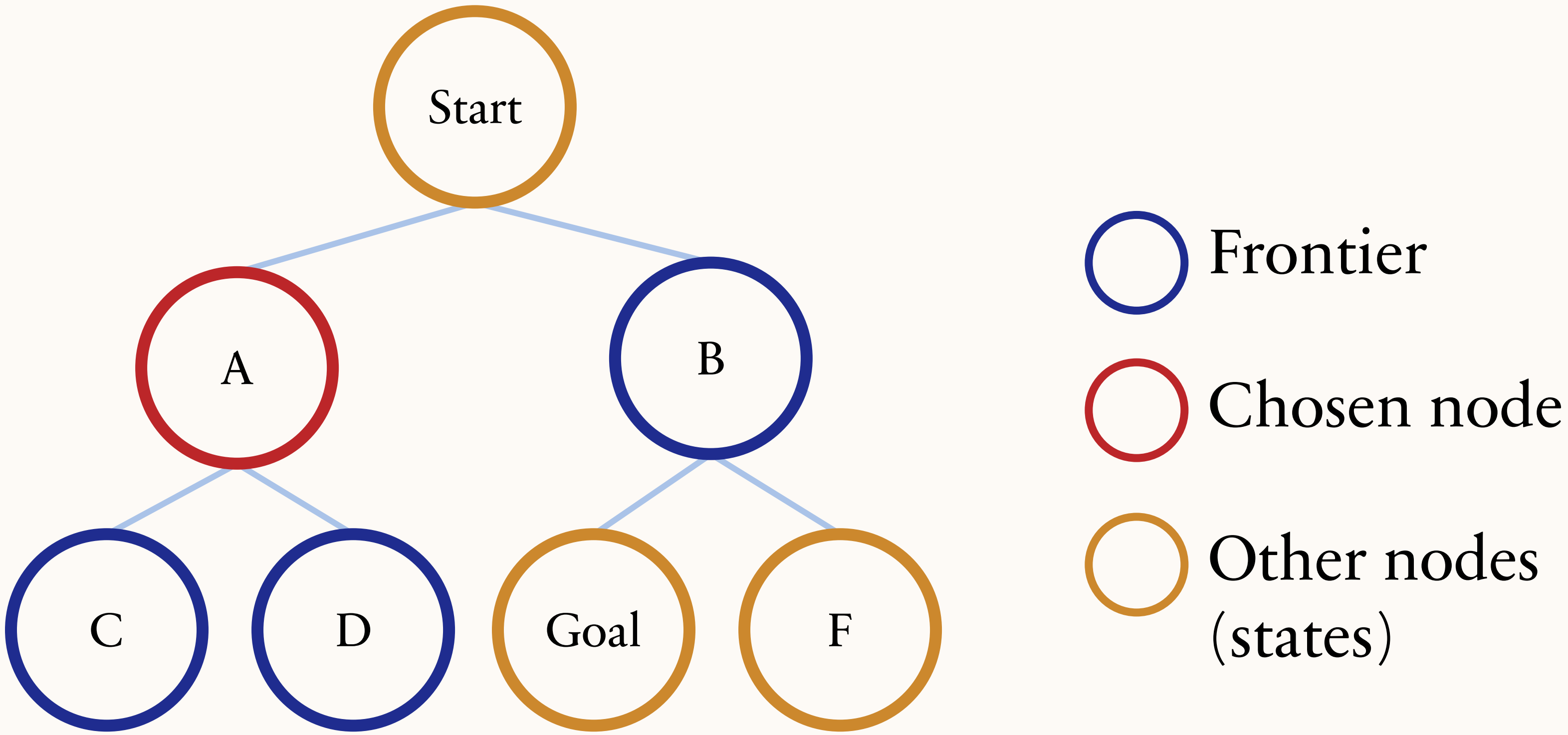
THE GENERIC SEARCH ALGORITHM



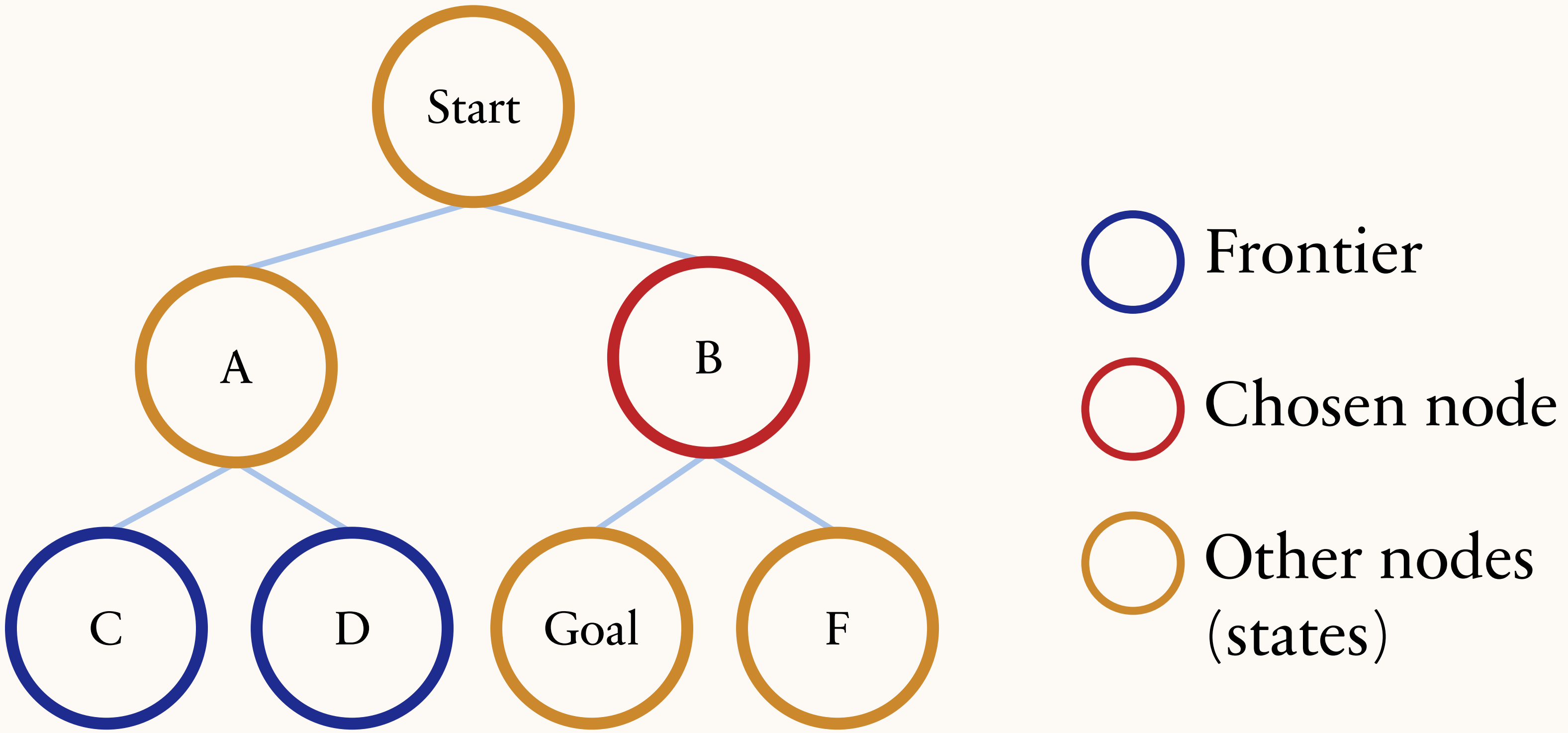
THE GENERIC SEARCH ALGORITHM



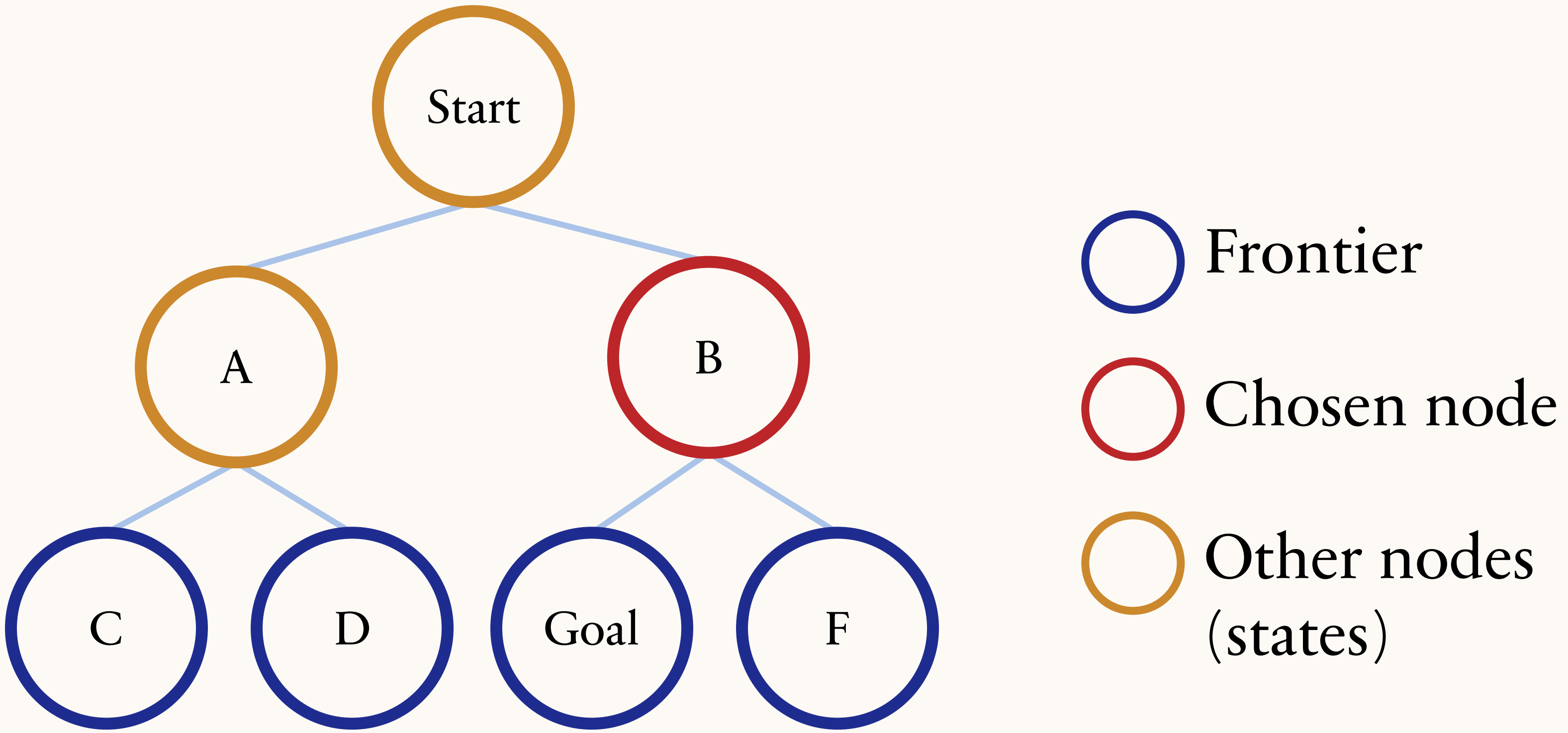
THE GENERIC SEARCH ALGORITHM



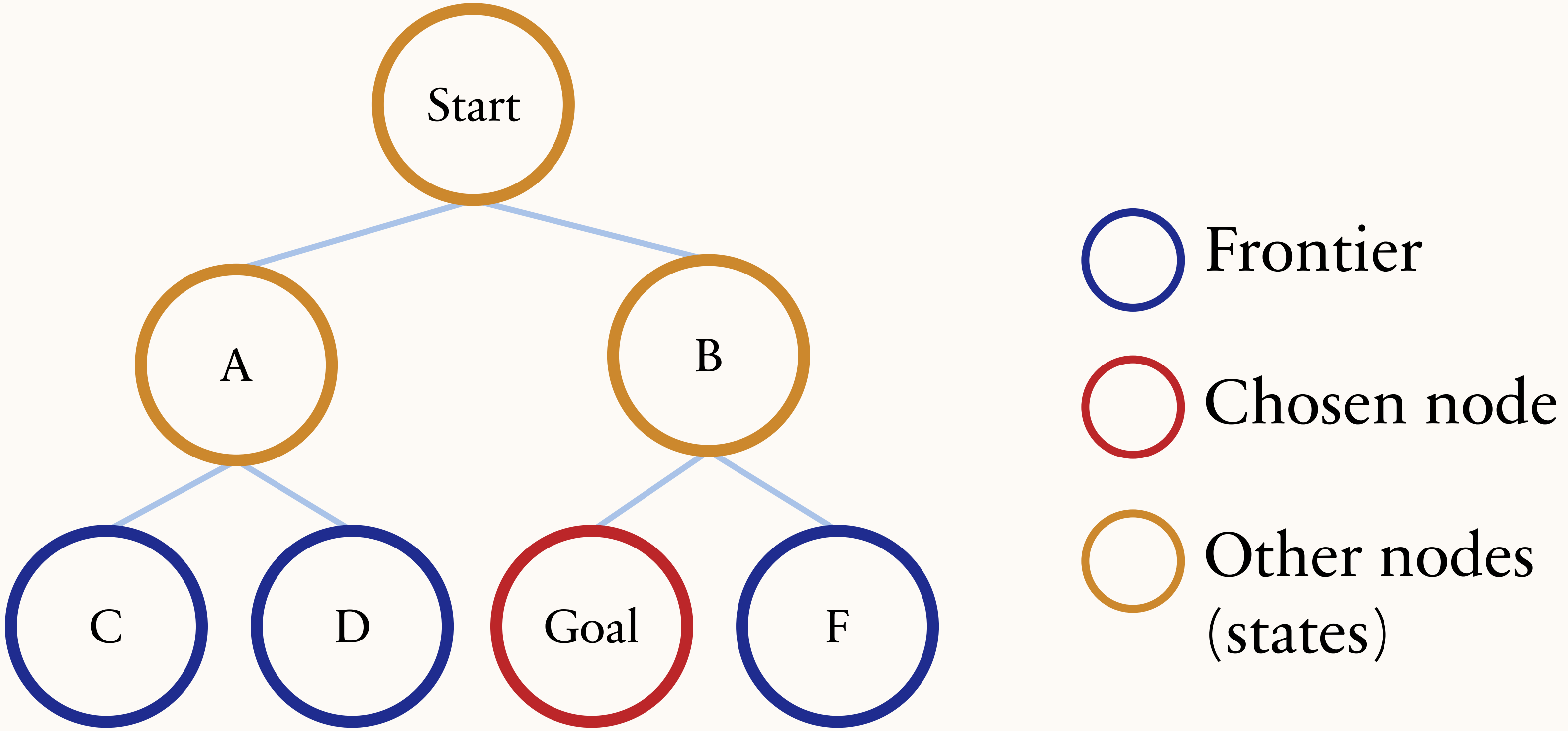
THE GENERIC SEARCH ALGORITHM



THE GENERIC SEARCH ALGORITHM

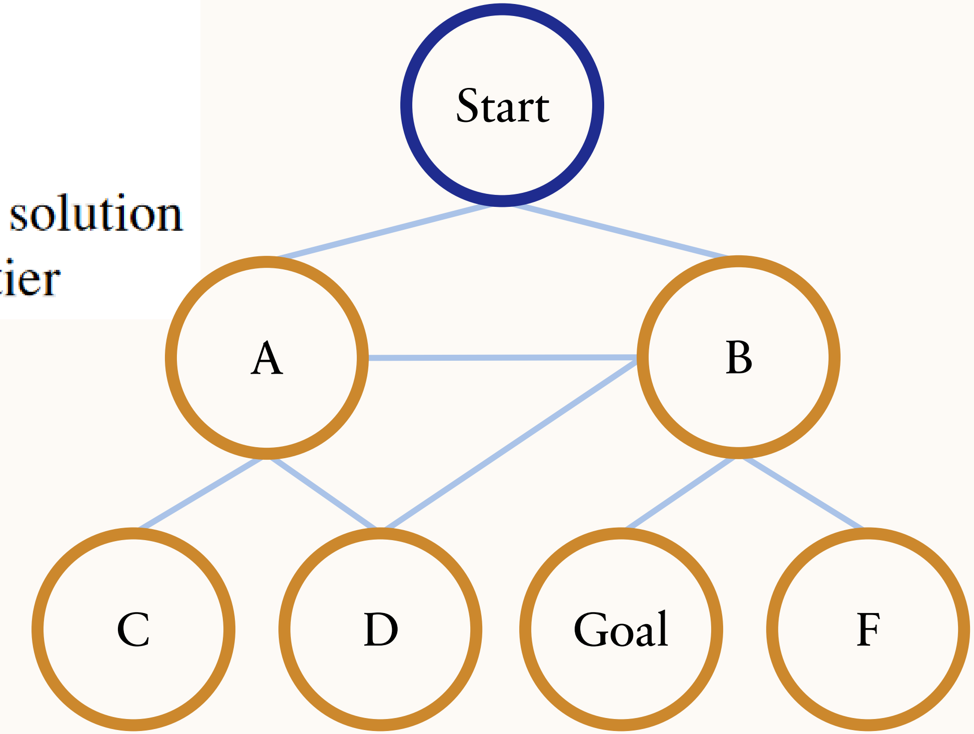


THE GENERIC SEARCH ALGORITHM



THE GENERIC SEARCH ALGORITHM

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

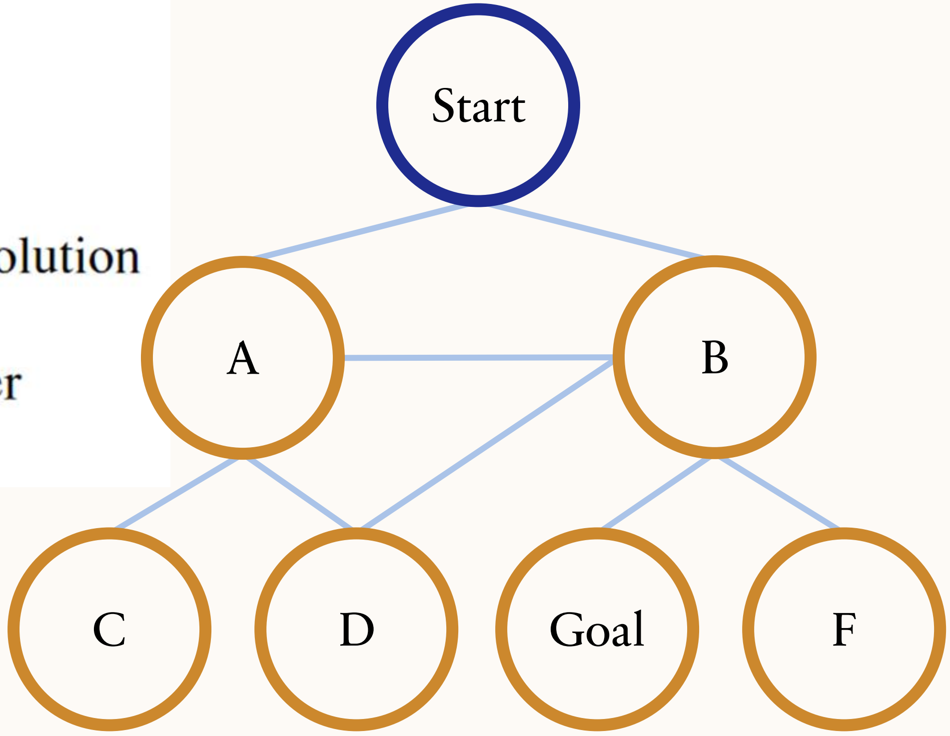


Challenges:

- Loops
- Inefficiency

THE GENERIC SEARCH ALGORITHM

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set



Challenges:

- ~~Loops~~
- Inefficiency

WHAT ABOUT EFFICIENCY?

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

Use data structures to control which nodes get explored next!

How we store the frontier affects the performance of the algorithm.

BREADTH-FIRST SEARCH (BFS)

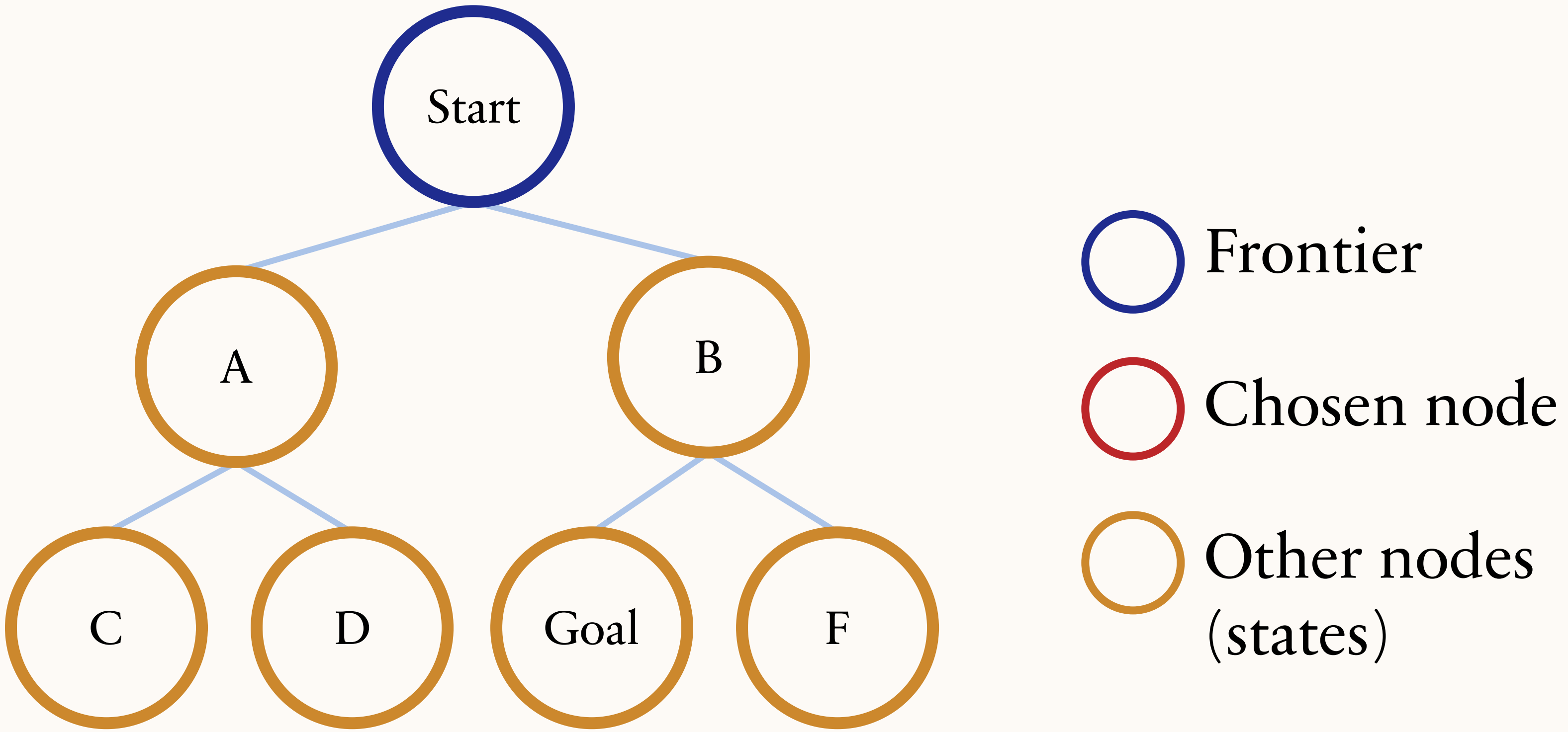
function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

Store frontier as first-in first-out (FIFO) queue

BREADTH-FIRST SEARCH (BFS): FIFO QUEUE

Frontier:
[Start]

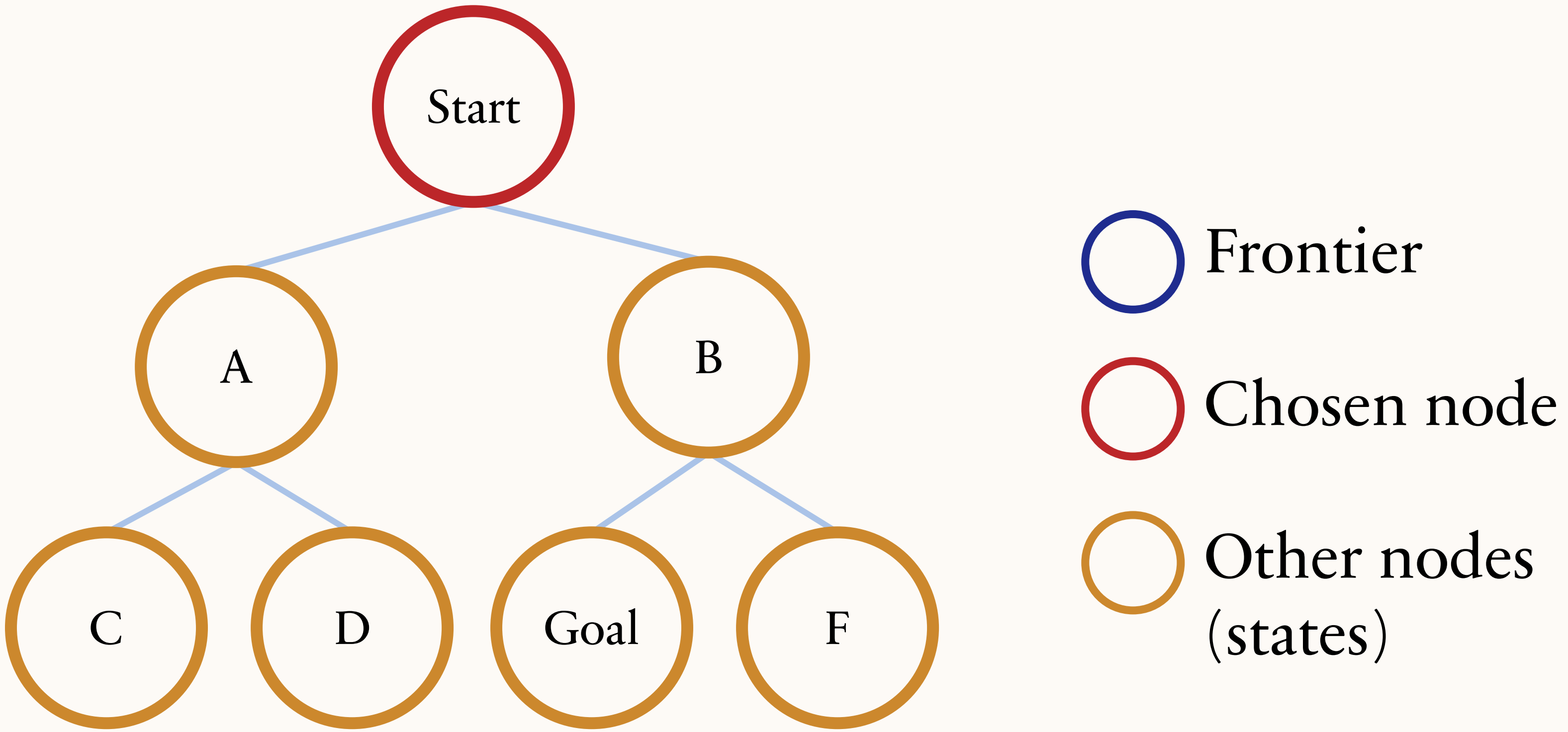
Current node:
none



BREADTH-FIRST SEARCH (BFS): FIFO QUEUE

Frontier:
[]

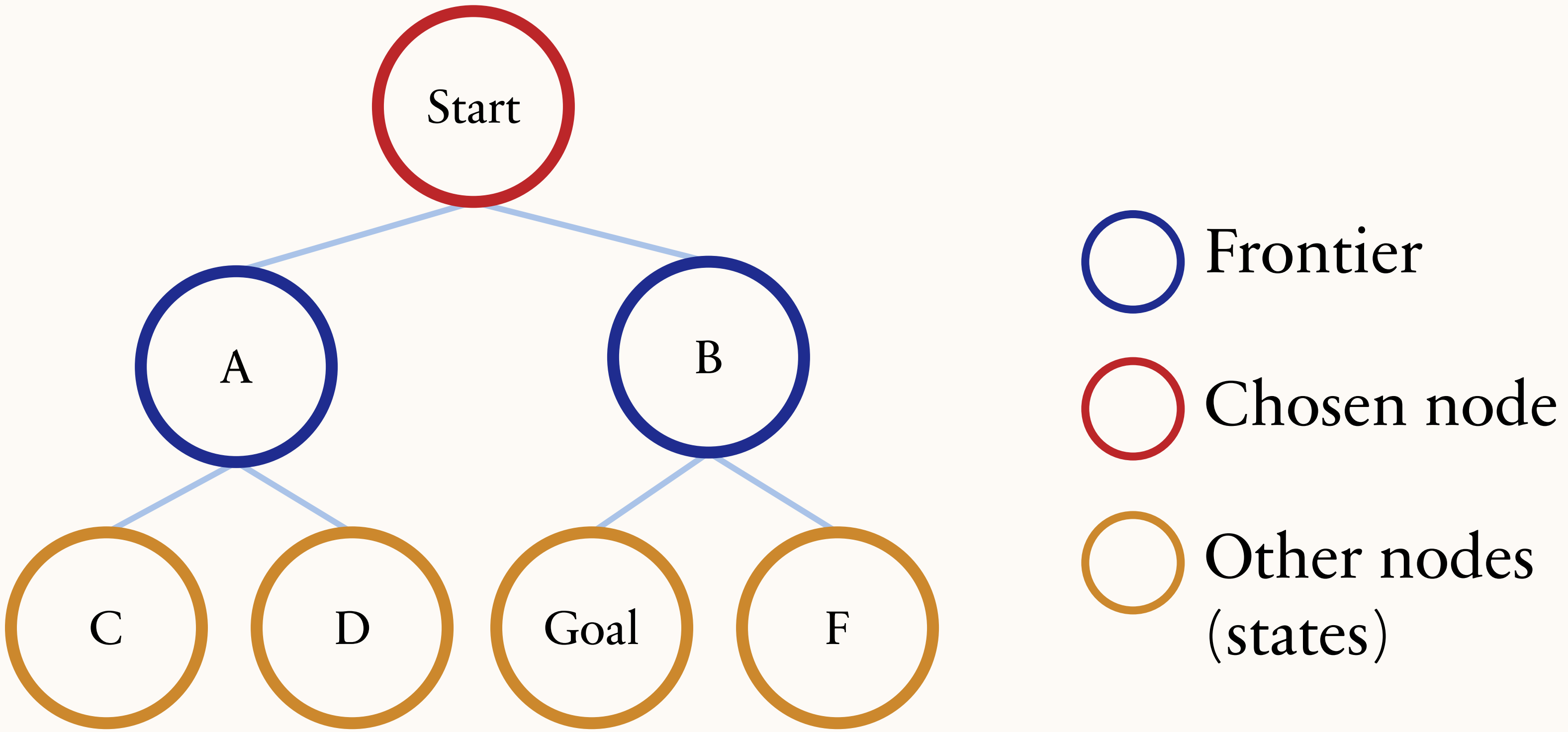
Current node:
Start



BREADTH-FIRST SEARCH (BFS): FIFO QUEUE

Frontier:
[A, B]

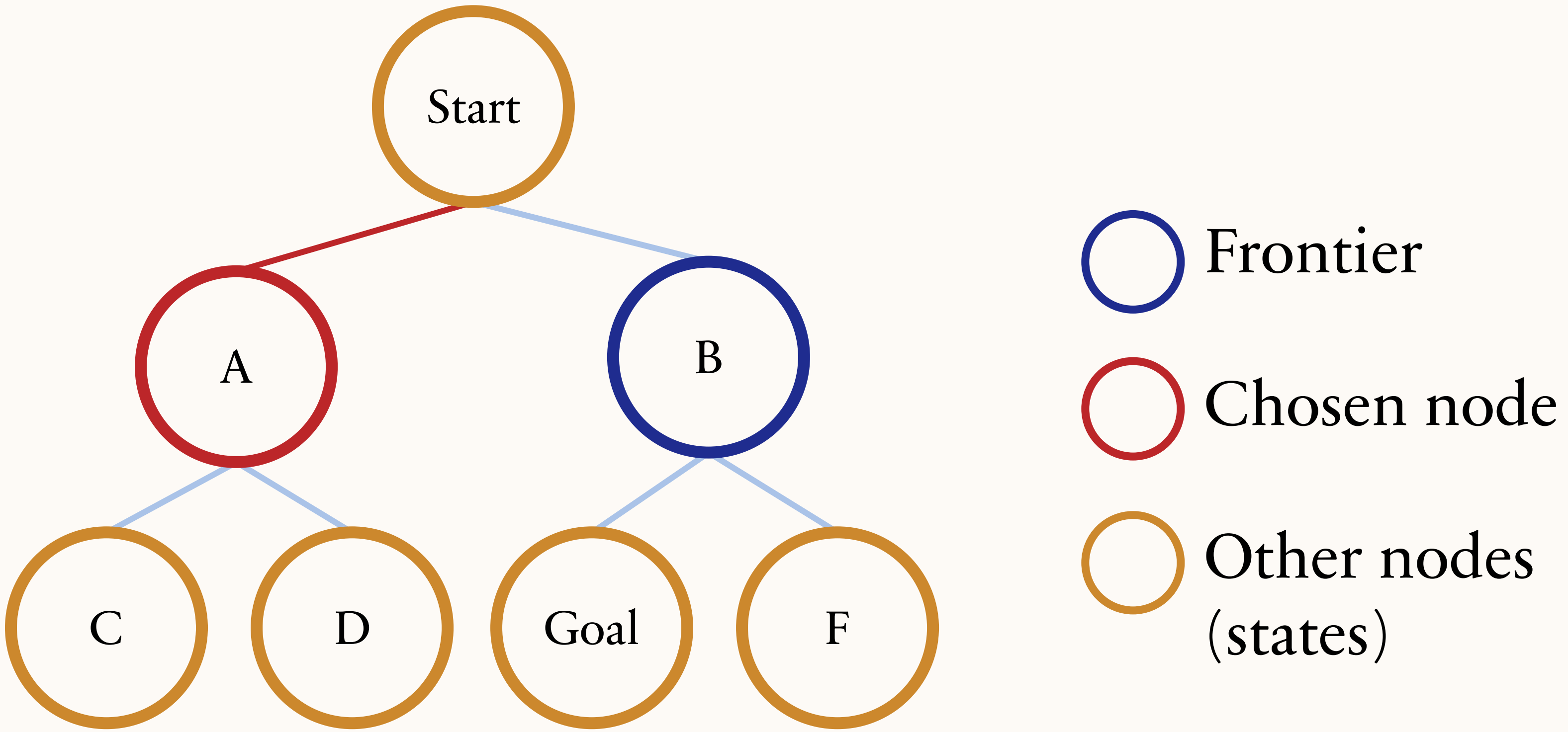
Current node:
Start



BREADTH-FIRST SEARCH (BFS): FIFO QUEUE

Frontier:
[B]

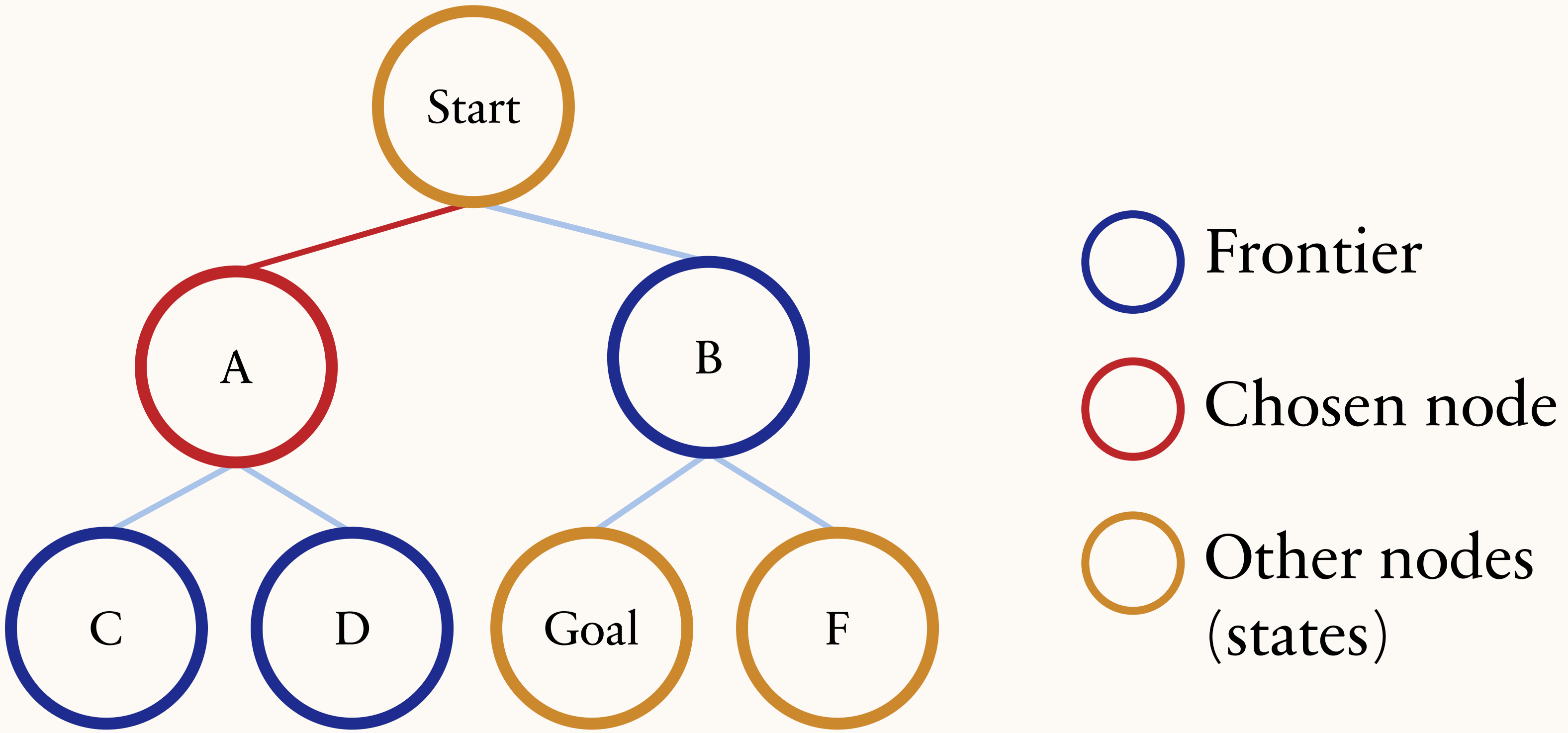
Current node:
A



BREADTH-FIRST SEARCH (BFS): FIFO QUEUE

Frontier:
[B, C, D]

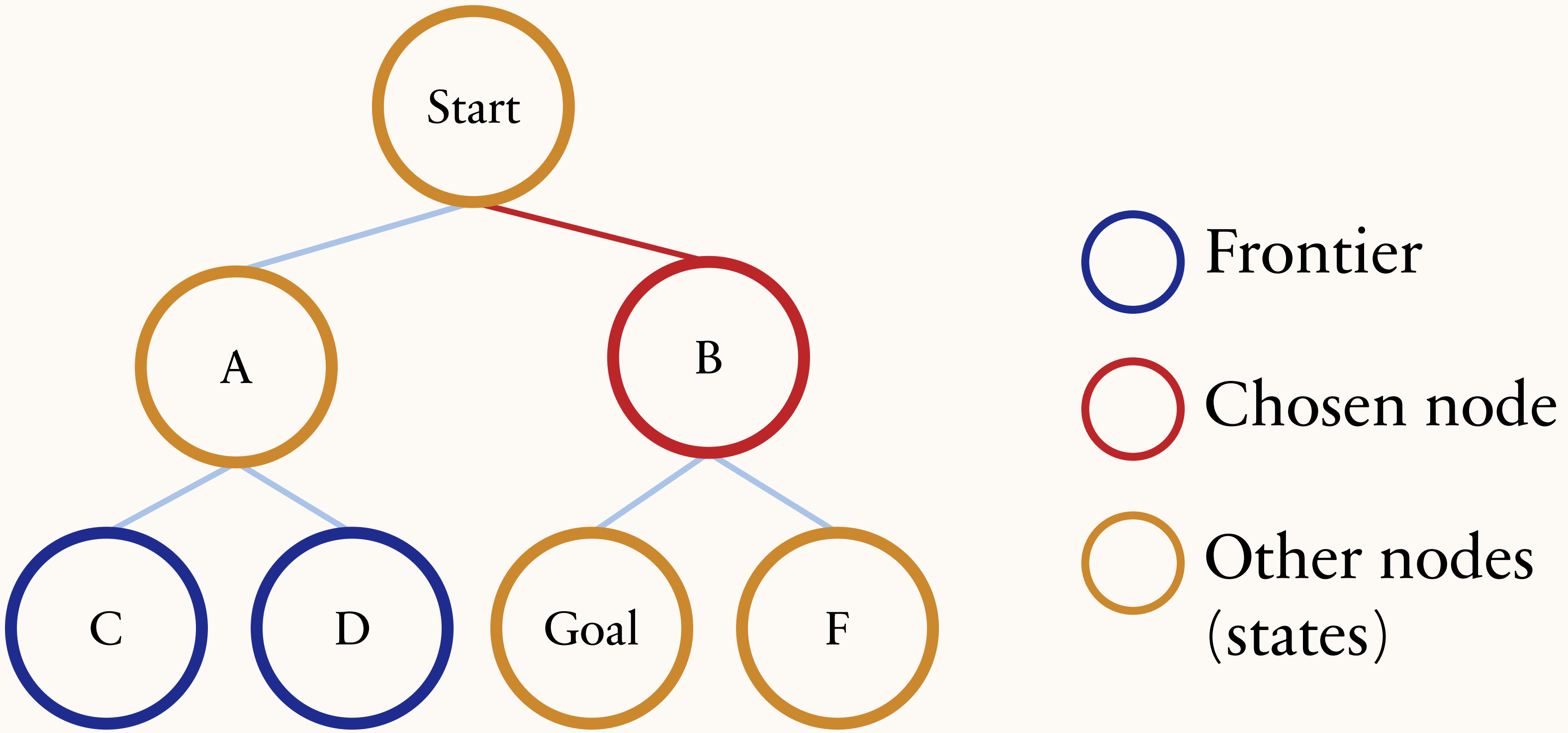
Current node:
A



BREADTH-FIRST SEARCH (BFS): FIFO QUEUE

Frontier:
[C, D]

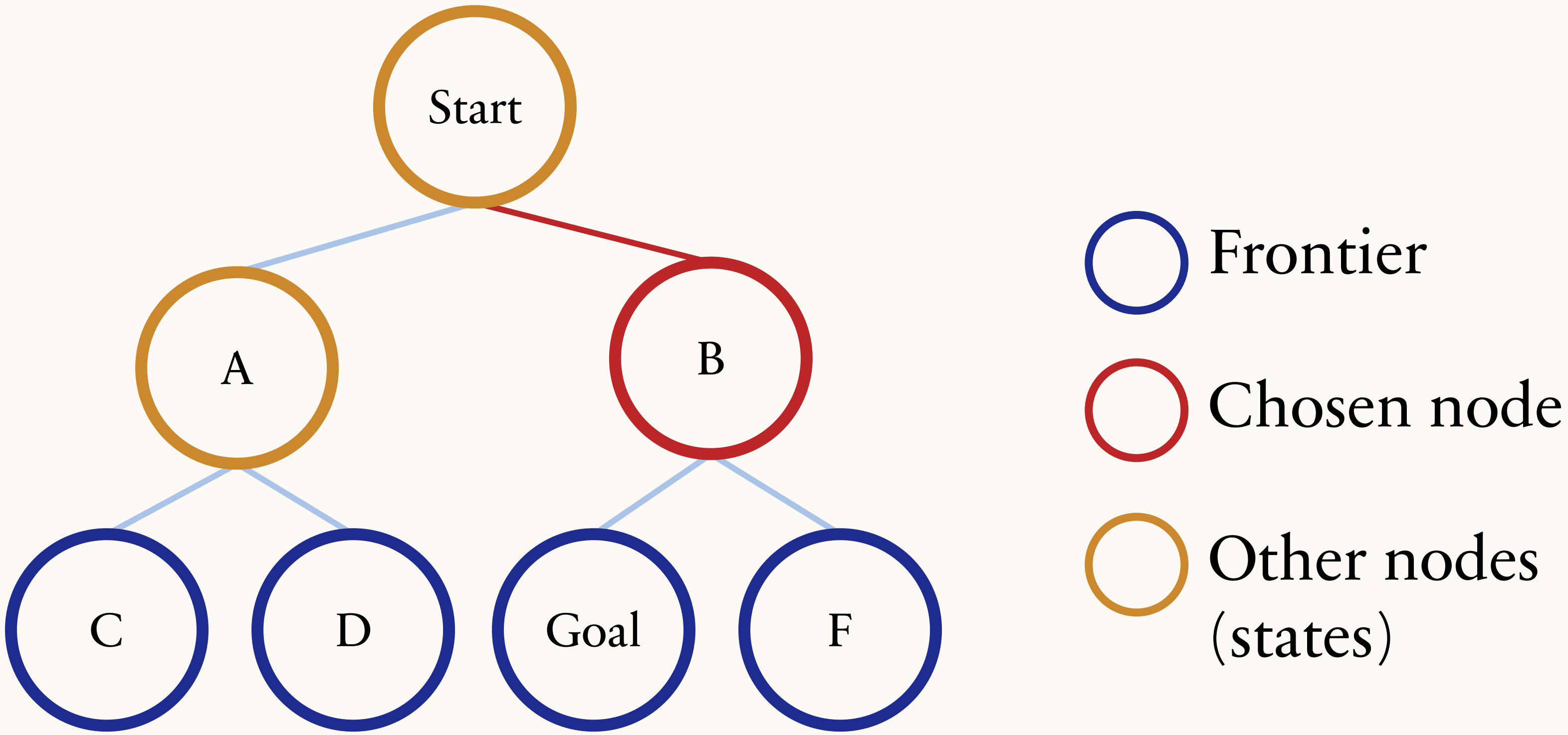
Current node:
B



BREADTH-FIRST SEARCH (BFS): FIFO QUEUE

Frontier:
[C, D, Goal, F]

Current node:
B



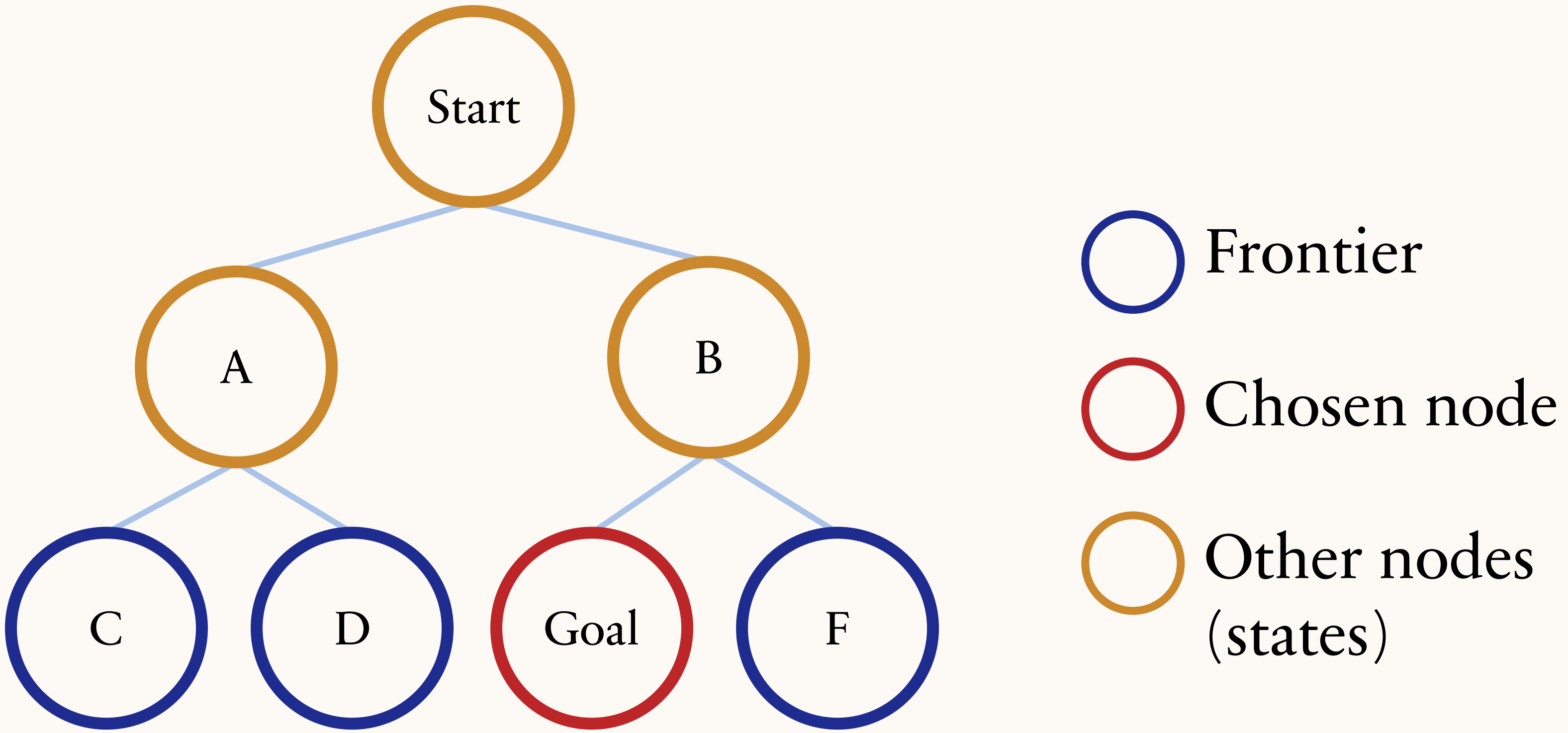
BREADTH-FIRST SEARCH (BFS): FIFO QUEUE

Frontier:
[C, D, Goal, F]

Eventually…

Current node:
B

Return:
[Start, B, Goal]



DEPTH-FIRST SEARCH (DFS)

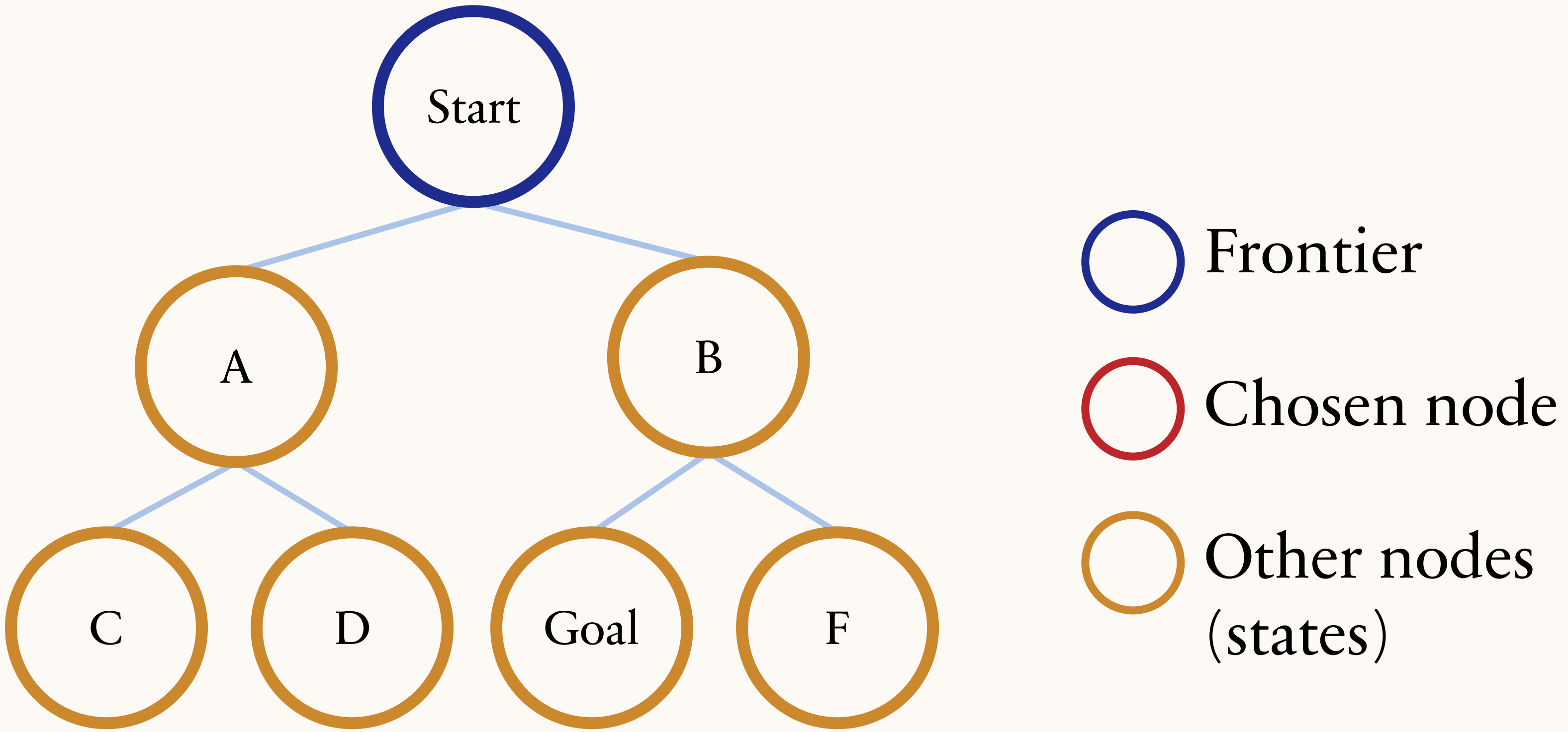
function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

Store frontier as last-in first-out (LIFO) queue, or a stack.

YOUR TURN – DFS: LIFO STACK

Frontier:
[Start]

Current node:
none



You can work in pairs! Put both your names on it, please!
To be submitted on Blackboard after class

FOR NEXT CLASS

- Fill out the paper presentation survey --- Due TOMORROW!
- Submit DFS answers
- Read Chapter 3.1-3.4 (if you haven't already)