Programming Vier Gewinnt

Part 1: Event Loop Basics
- Control Flow
- Background Compute Thread

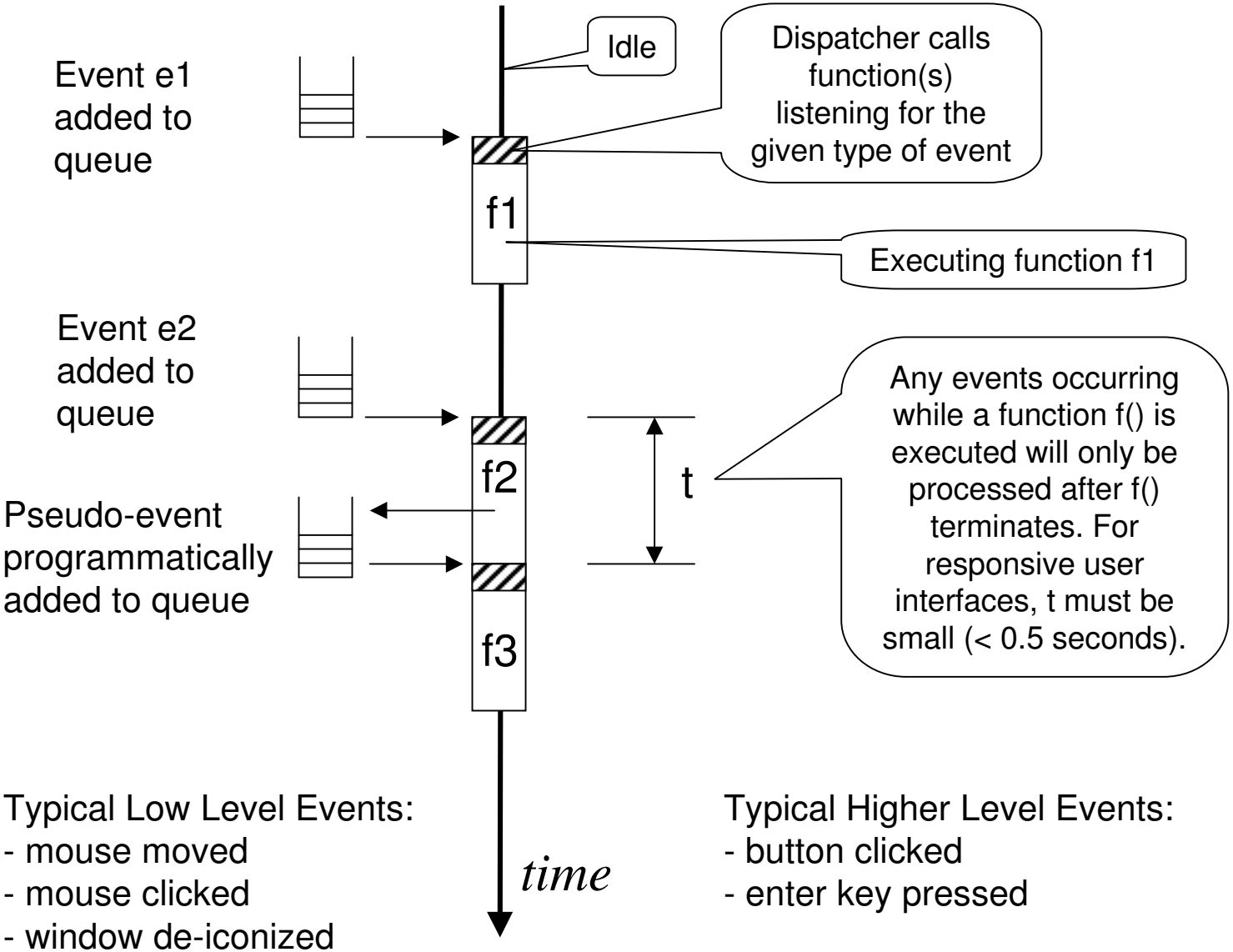
Part 2: How to determine the shortest path to victory

Part 3: Known bugs

Detlef Wolf

2006-12-13

Event Loop Thread

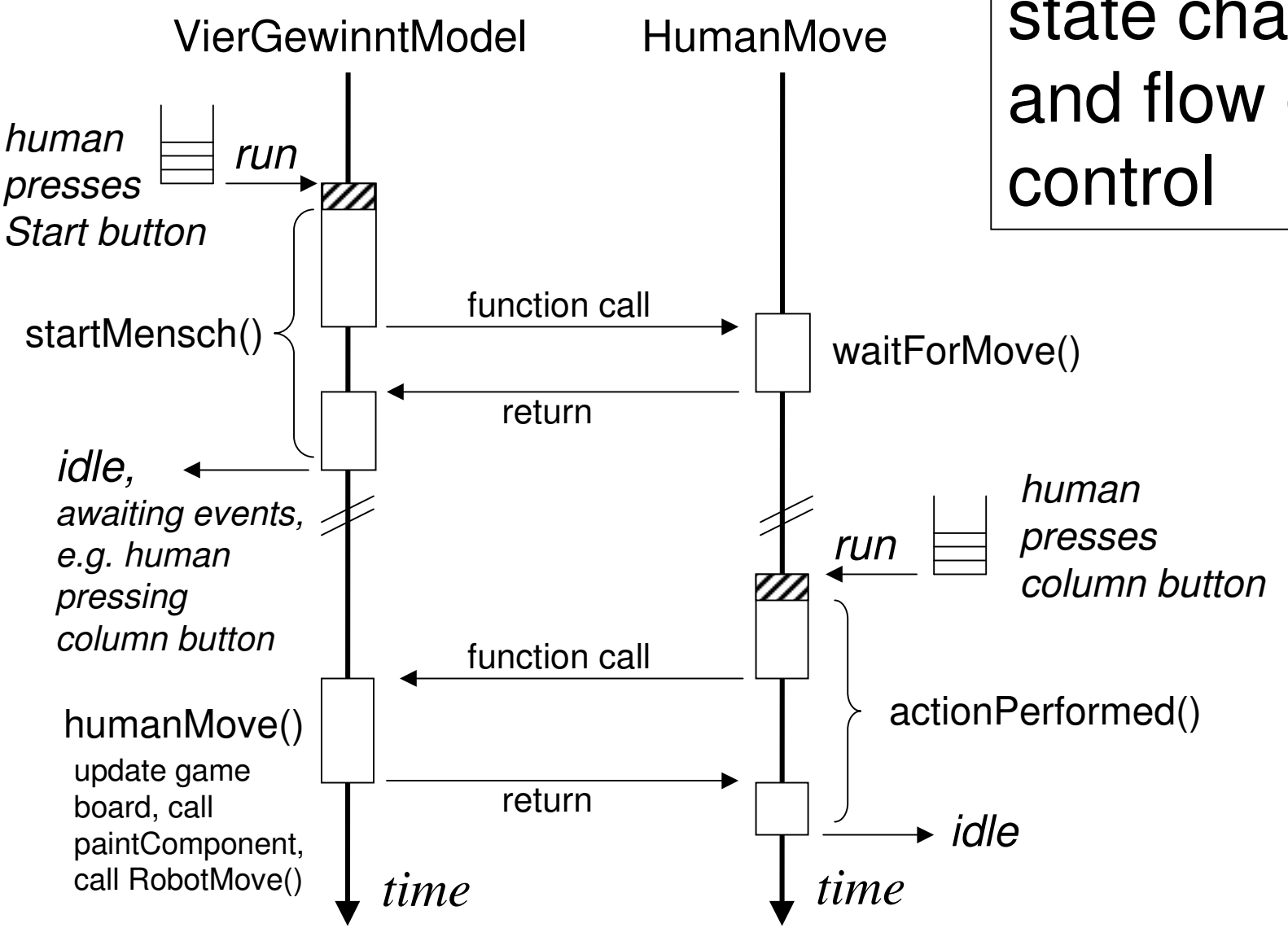


Typical Low Level Events:
- mouse moved
- mouse clicked
- window de-iconized

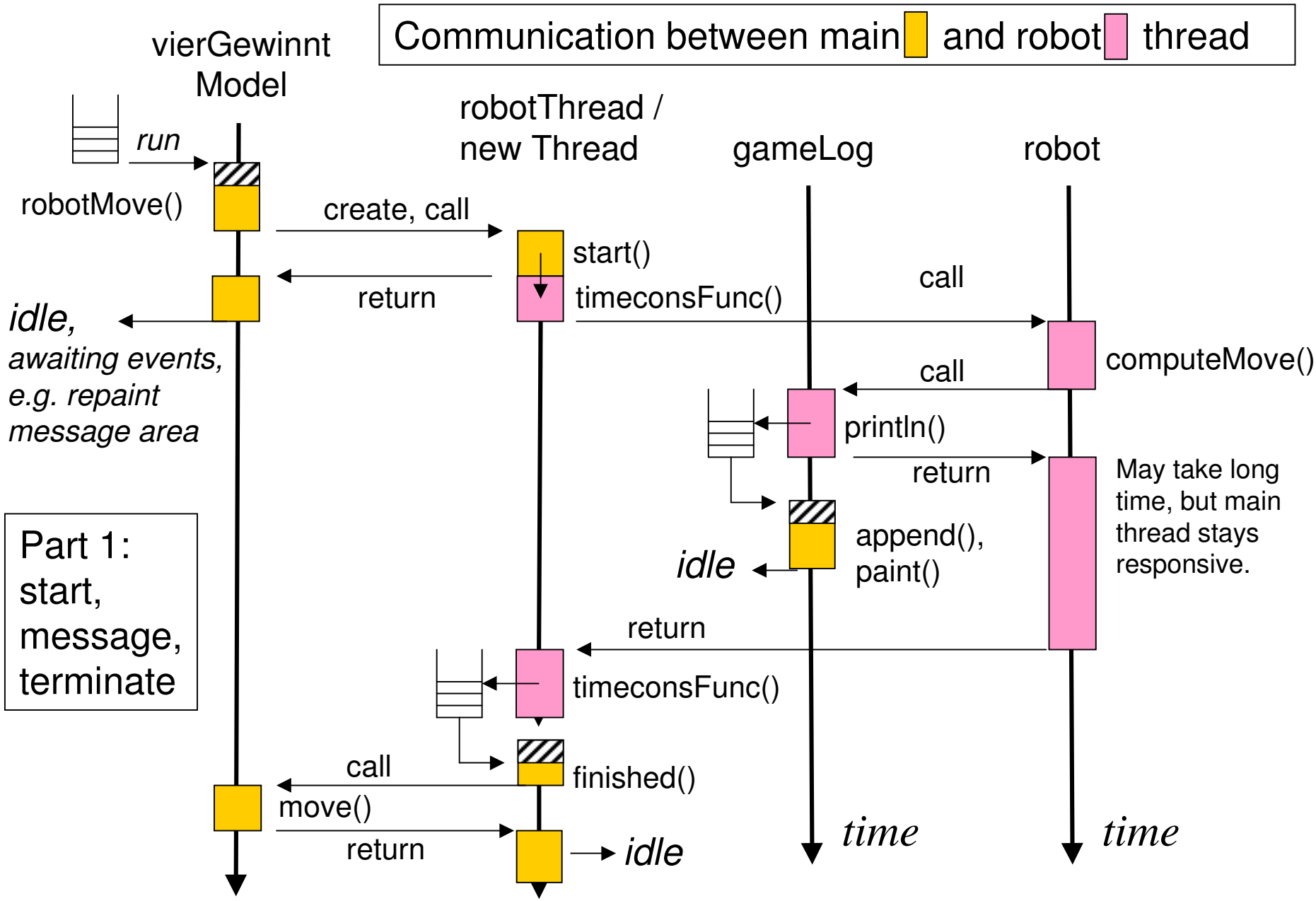
Typical Higher Level Events:
- button clicked
- enter key pressed

The vertical line symbolizes the main execution thread

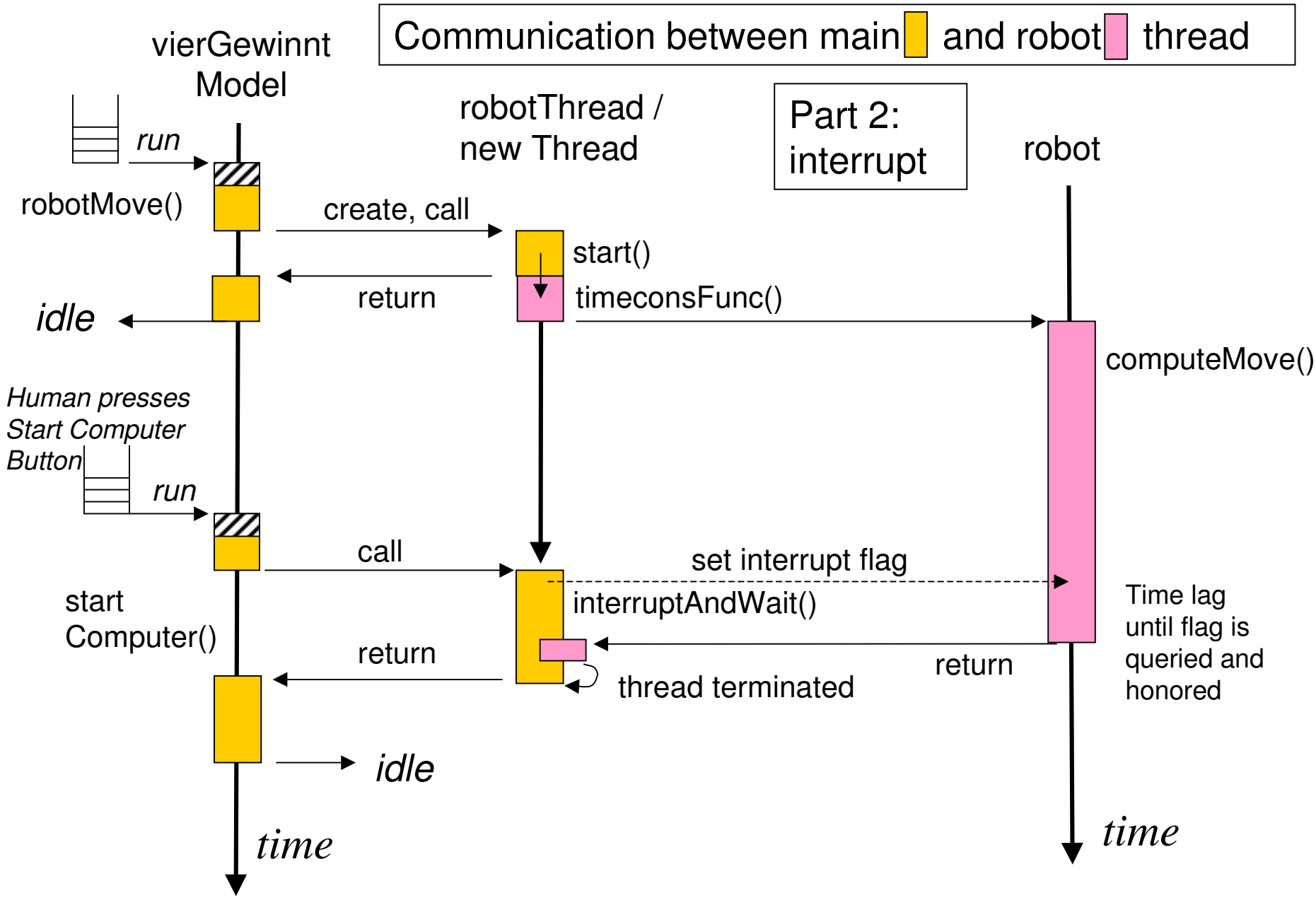
state changes and flow of control



Each of the two vertical lines symbolizes one object instance

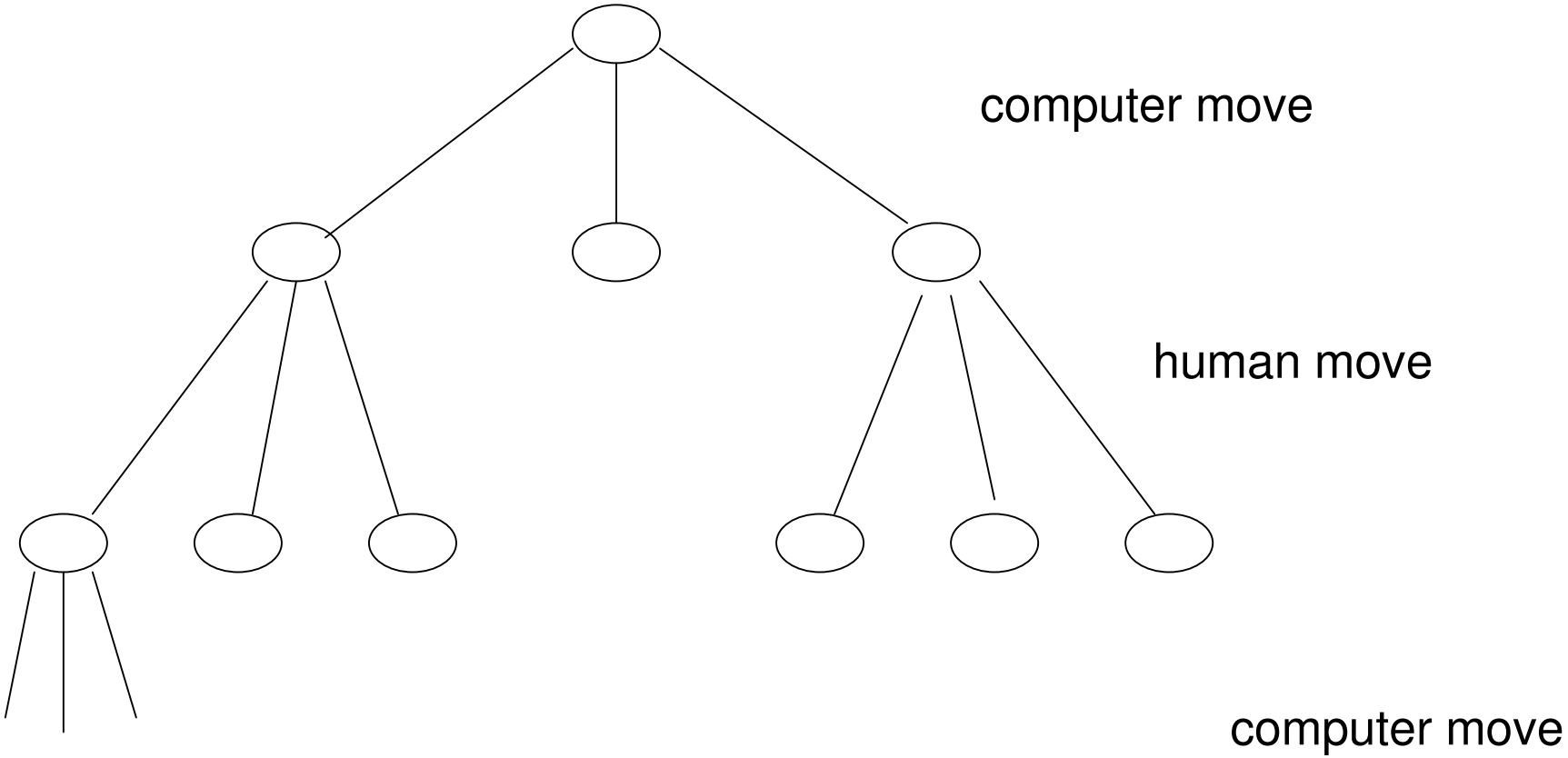


Each of the lines symbolizes one object instance, color symbolizes threads



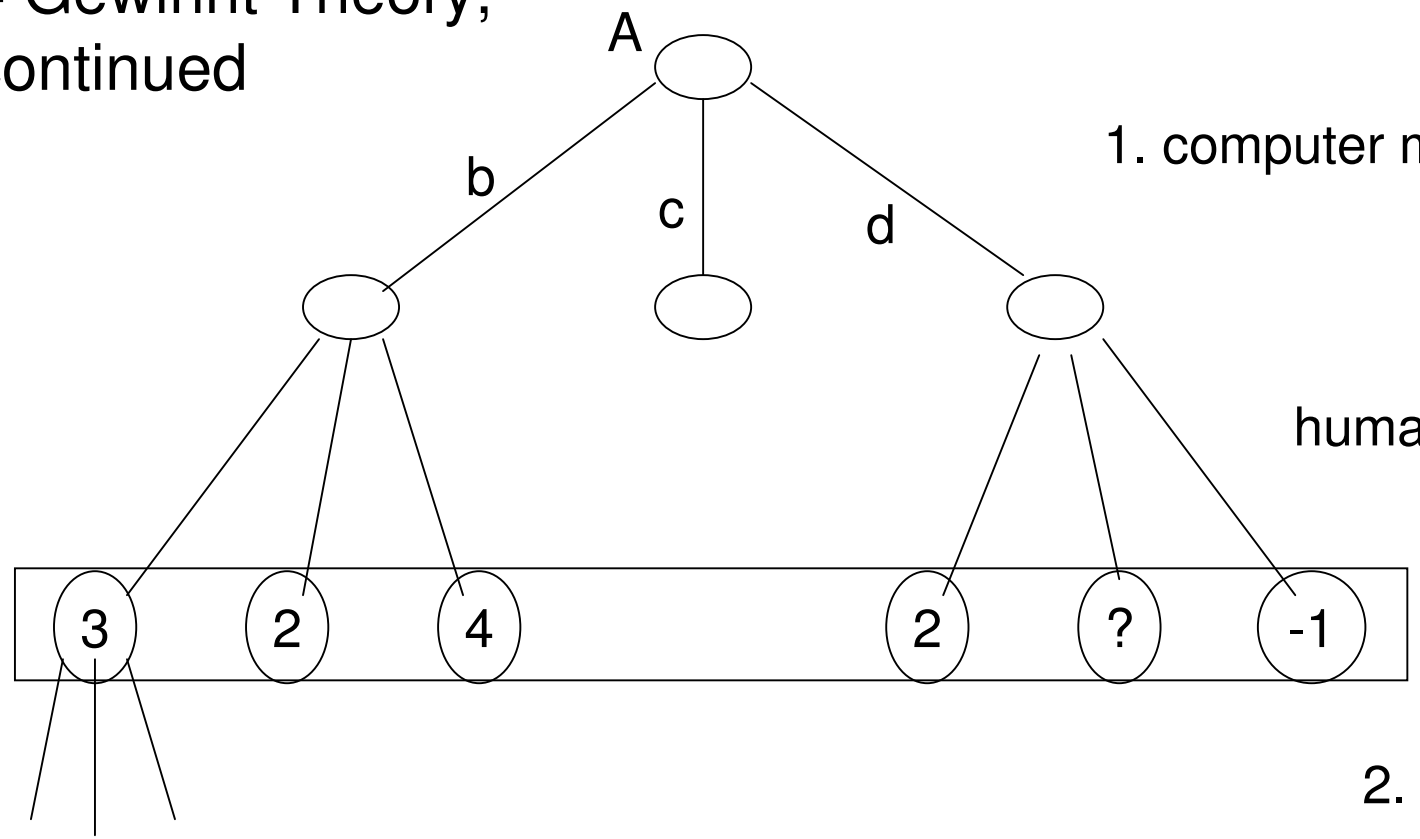
Each of the lines symbolizes one object instance, color symbolizes threads

4 Gewinnt Theory:
How to determine the shortest path to victory



In “4 Gewinnt”, up to seven moves are possible. Without loss of generality, only three are considered here.

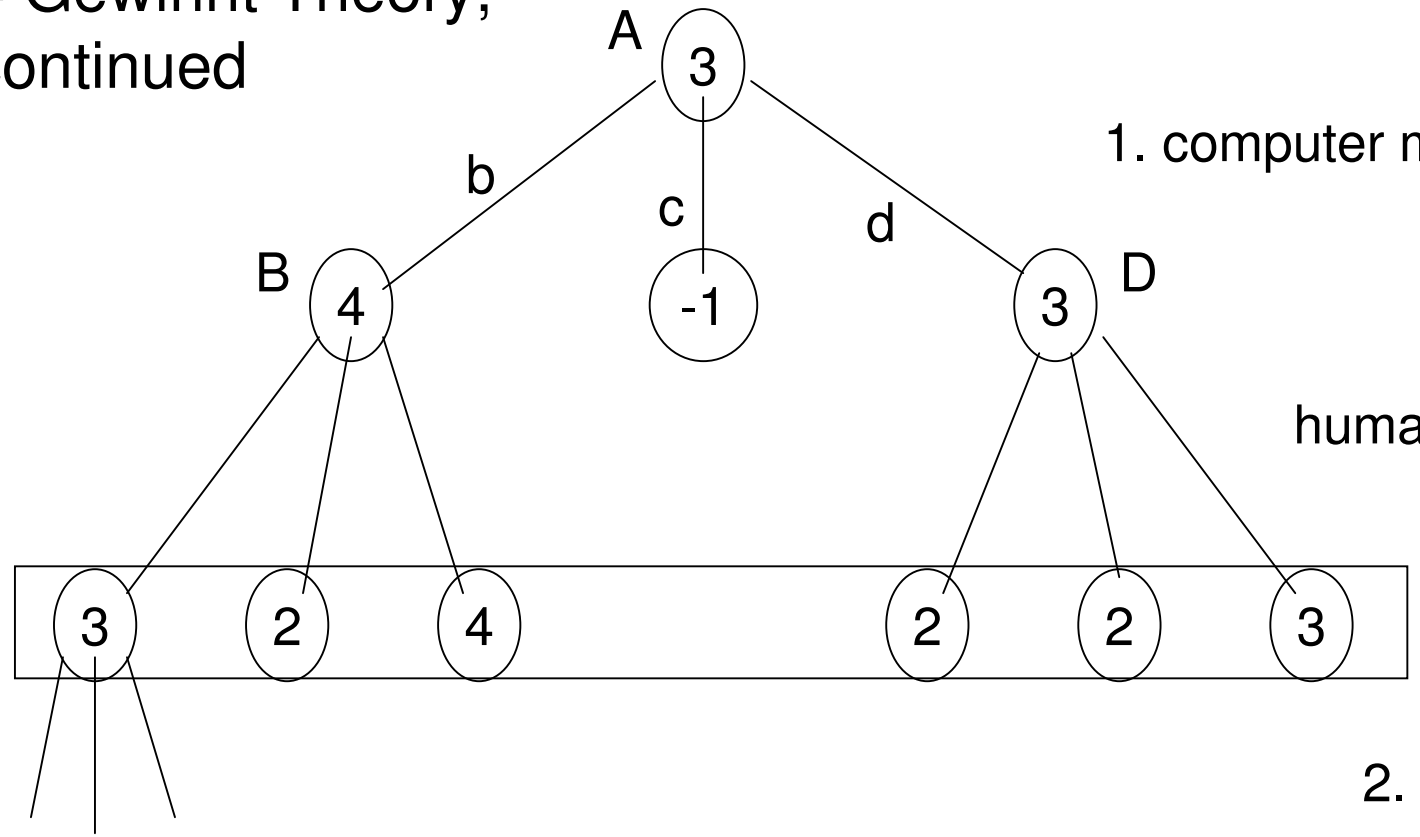
4 Gewinnt Theory, continued



Lets assume
- none of the moves b,c,d leads to a direct win of the computer.
- none of the positions (ovals) in the box is a winning position for the human.
- for all the positions in the box we know the maximum number of moves it takes for the computer to win starting from that position.

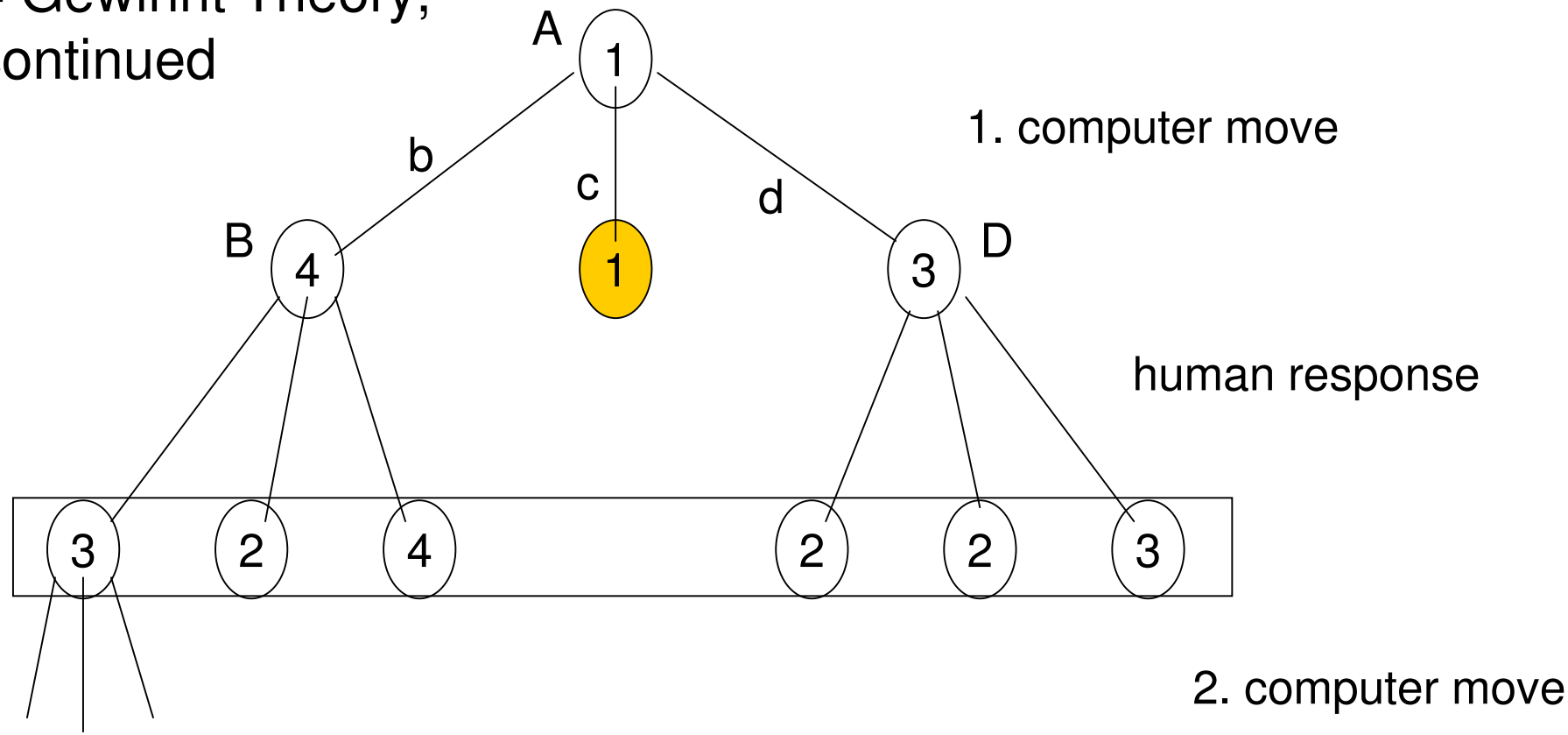
Can we derive the maximum number of moves it takes to win from position A?

4 Gewinnt Theory, continued



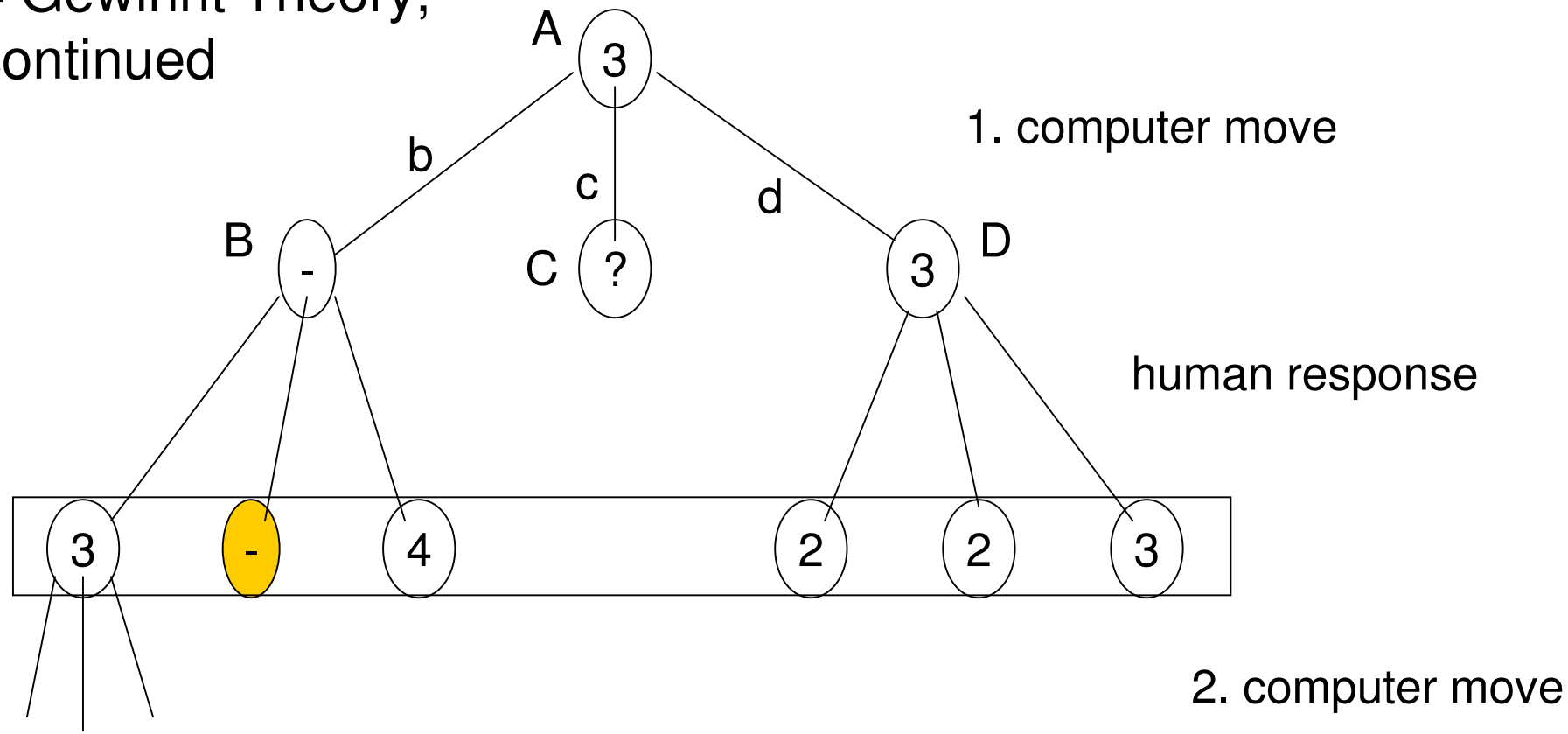
Lets assume for move c the outcome cannot be determined.
Lets assume the human tries to delay defeat. This means we need to take the maximum from 3,2,4 to get the number of moves for B, and the maximum of 2,2,3 for D.
Lets assume the computer plays to win as fast a possible, so it would take the minimum of 4,3 and play move d.
The maximum number of moves it takes to win from position A is thus 3.

4 Gewinnt Theory, continued



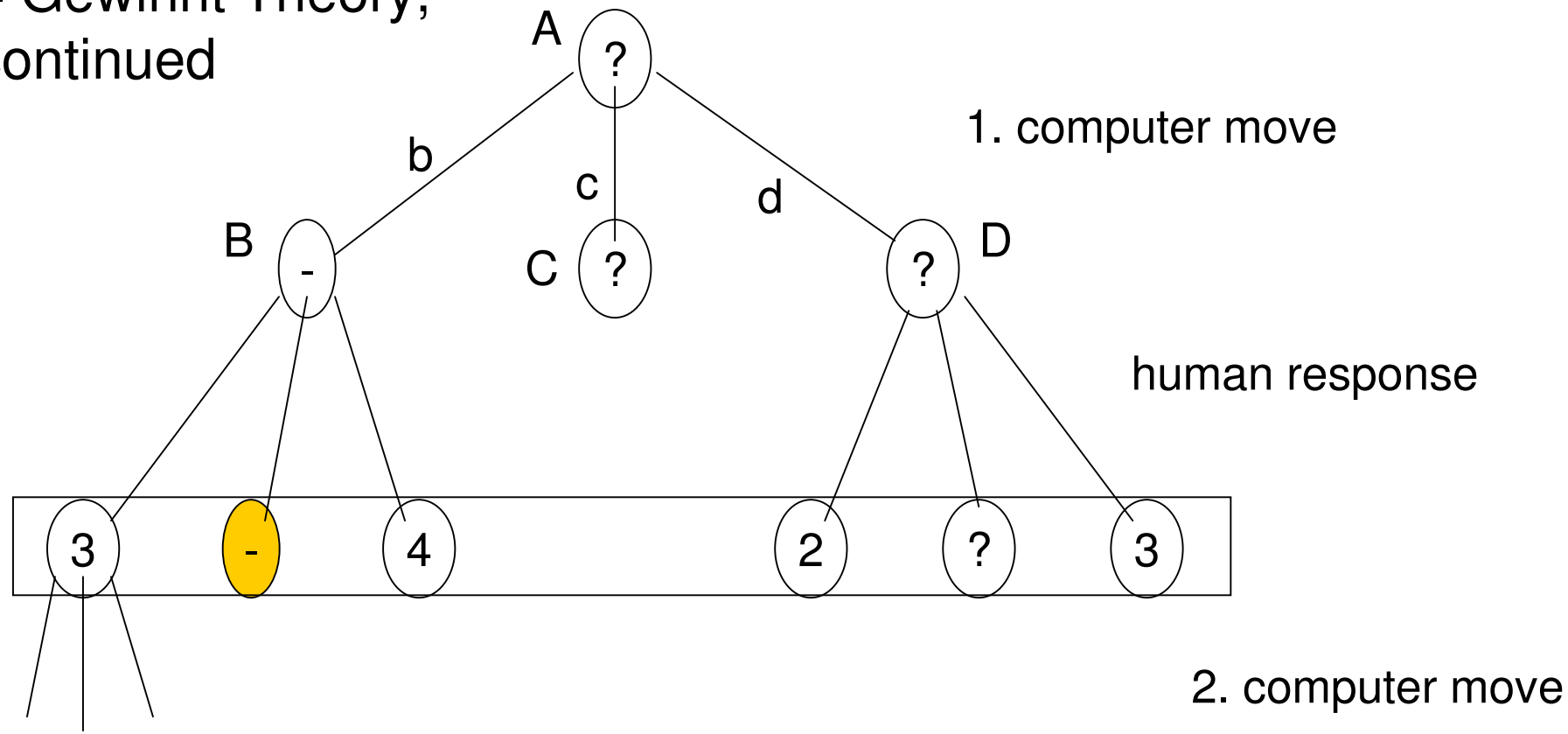
Other cases:
Move c is a direct win for the computer.
The maximum number of moves it takes to win from position A is thus 1.

4 Gewinnt Theory, continued



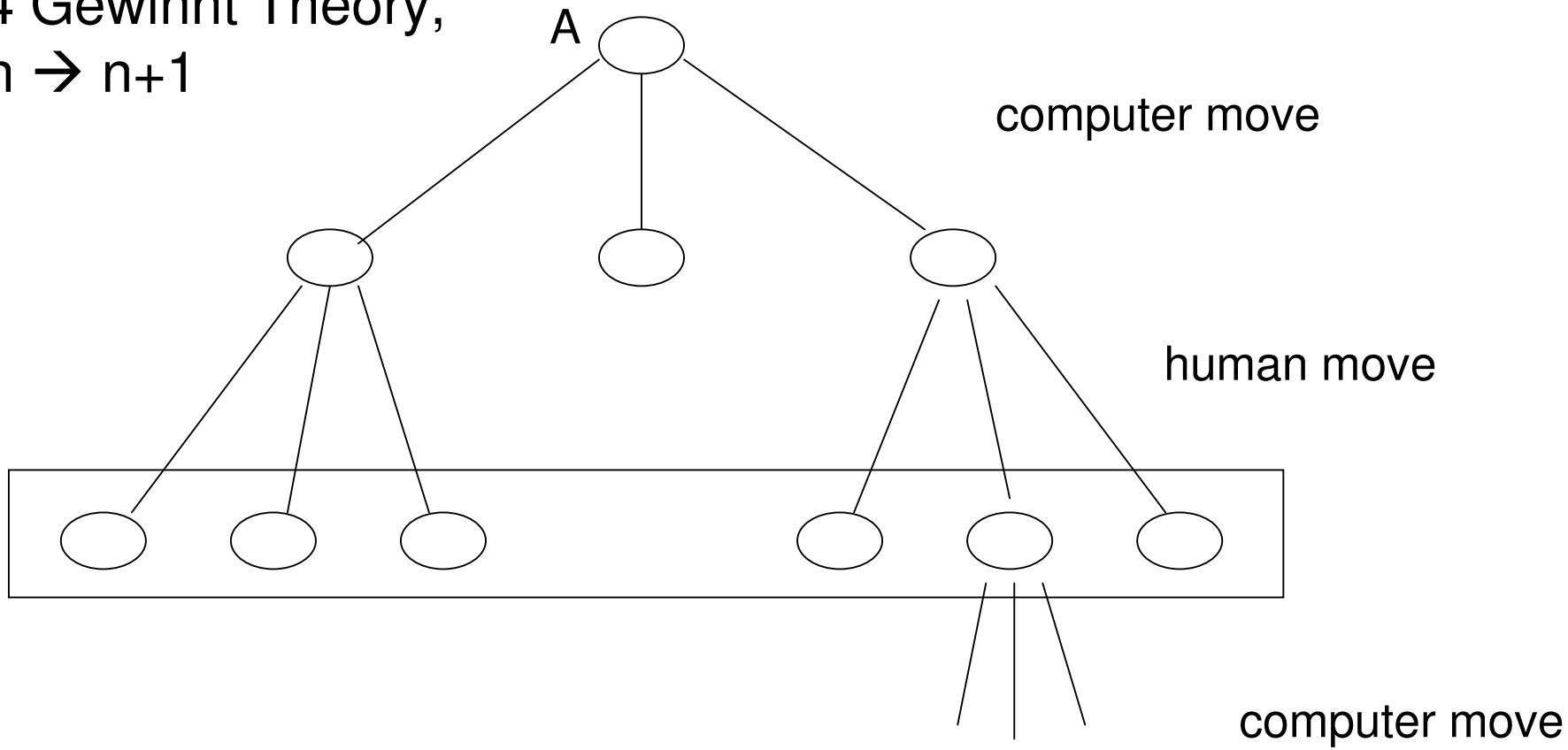
Other cases:
Move b would allow the human to win. Use move d, which is a sure win.
The maximum number of moves it takes to win from position A is thus 3.

4 Gewinnt Theory, continued



Other cases:
If the computer uses move b, the human can win. If the computer plays d, there is one case where it cannot be determined who will win.
So the maximum number of moves it takes to win from position A is unknown.

4 Gewinnt Theory, n → n+1



The previous discussion showed that if one can compute the shortest number of winning moves for every node in the box, then one can derive the shortest number of winning moves for the top node A by taking the maximum of the minima.
As for every node in the box we are in the same situation as for node A (computer move), all we need is to do is drawing the whole tree (or only up to a certain depth) and apply the procedure again and again for bottom to top.

Known bugs

• resizing the window such that is less wide than required by the longest line in gameLog makes the gameLog textArea shrink to one line in height.
• The “check for interrupt” in Robot.computeMove() is of type “busy wait”. This is not elegant and wastes compute time. Isn’t there a better solution?
• computation could be made much faster: currently the tree is computed once for a computer and once from a human perspective. These two runs could be merged.