DiCE: DISTRIBUTED COLLABORATIVE COMPUTING AT THE EDGE (*HYRAX PROJECT – DISTRIBUTED QUEUE*)

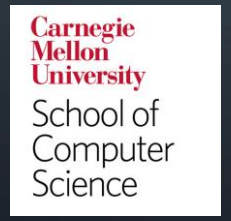
RÚBEN ANDRÉ BARREIRO – Nº. 42648

MESTRADO INTEGRADO DE ENGENHARIA INFORMÁTICA

FACULDADE DE CIÊNCIAS E TECNOLOGIA DA UNIVERSIDADE NOVA DE LISBOA

INTRODUCTION

- The Hyrax project it's been inserted in the context of Edge Computing and Mobile Devices' *Crowdsourcing*;
- This project it's been developed together by Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Faculdade de Ciências da Universidade do Porto, Carnegie Mellon University;
- This project, also have, some Academic, Industrial and Funding supports;
- The Distributed Queue of the Hyrax project was developed by some members of NOVALINCS team, including professors, BSc students, MSc students and PhD students, team in which I was inserted;



MOTIVATIONAL EXAMPLE (THE WEDDING) (1) - WHAT'S THE PROBLEM?

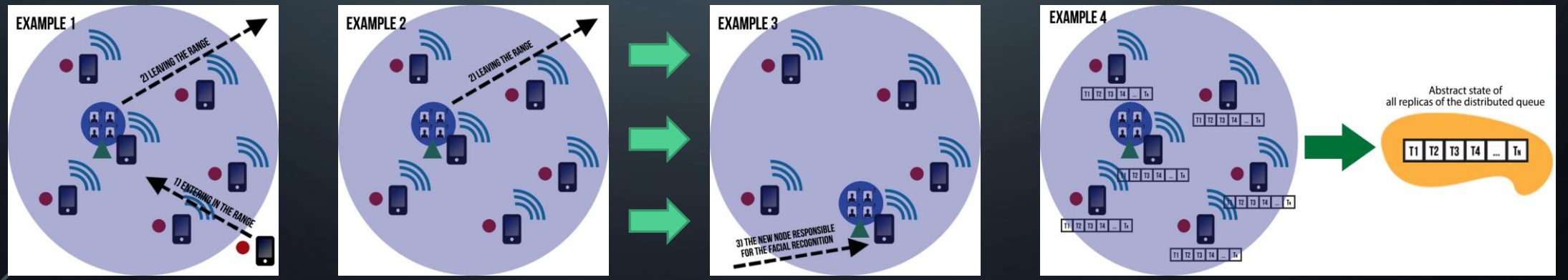
- For example, in a Wedding Party, there are a few guests' Devices that have an app to do facial recognition about the photos that are being taken in the event;
- That same few Devices doing the facial recognition of the photos are making and distributing the computation in a small network range or neighborhood;
- The Devices of the guests that want to find the photos of the its owners during the Wedding Party event, send computing requests by broadcast to the guests' Devices that are making the facial recognition;
- The problem it's that only the Devices that are doing the facial recognition of the photos produce all the computation work, while the other Devices are just consuming computation work;

MOTIVATIONAL EXAMPLE (THE WEDDING) (2) - WHAT'S THE SOLUTION?

- If was possible, the overall system would have a better performance, if all the Devices in the range/network can produce work, instead of, only the Devices that are doing the facial recognition and identifying the people in the photos;
- Here, it was adopted a most optimistic and relaxed architecture, in opposite to, the common Master/Slave or Centralized Cloud System architectures using locks and exclusive accesses;
- Thus, all Devices could consume and produce work, working together in the range/network, in a collaborative way, where all the Devices produce and consume work/computation;

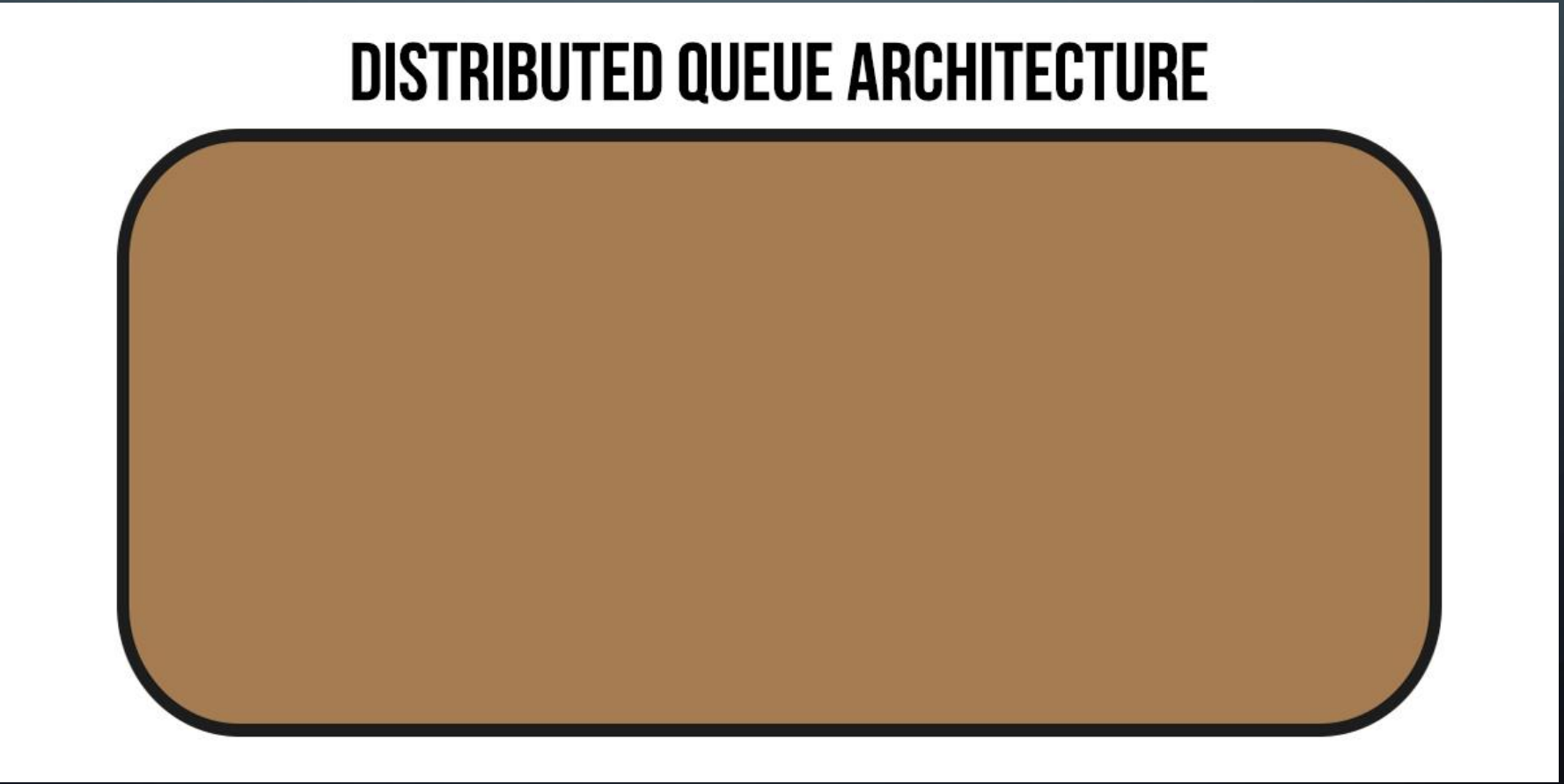
MOTIVATIONAL EXAMPLE (THE WEDDING) (2) - WHAT'S THE SOLUTION?

- This approach it's, mostly appropriated, to hand-held Devices, in a volatile environment, with a constant entrance and exit of Devices and it's never known by any Device which other specific Devices are in the range/network or not;
- So, the main idea, is that all the Devices could have a data structure instantiated and replicated by every single one of them, to keep and distribute computation requests, most specifically a distributed queue;
- In this approach, it's assumed that:
  - The Elements/Tasks are idempotents and commutative;
  - The Devices' clocks are synchronized;
  - The communication among the Devices are made by Broadcast/Hops;



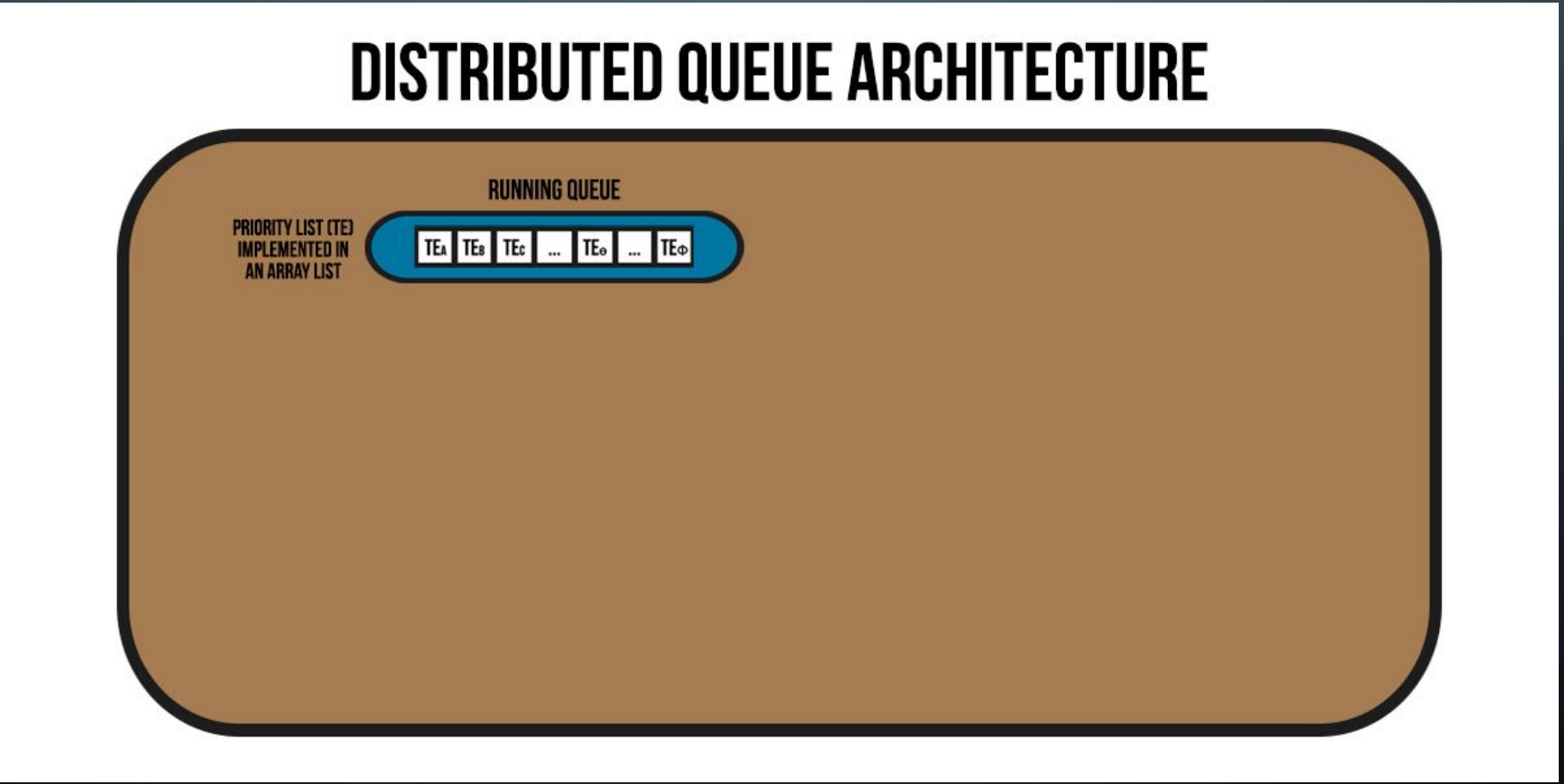
DISTRIBUTED QUEUE SYSTEM ARCHITECTURE

- The Distributed Queue System Architecture it's composed by:



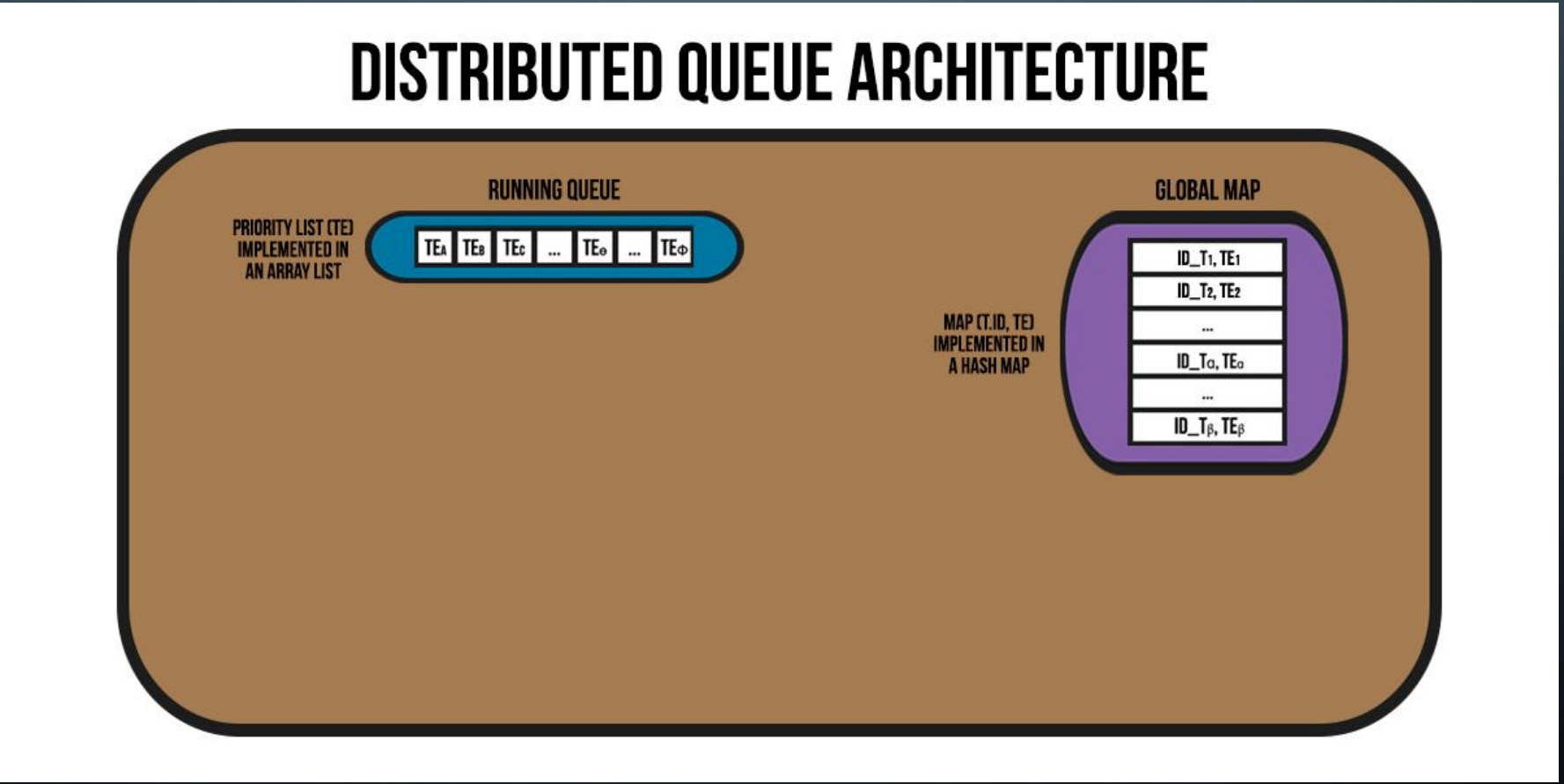
DISTRIBUTED QUEUE SYSTEM ARCHITECTURE

- The Distributed Queue System Architecture it's composed by:



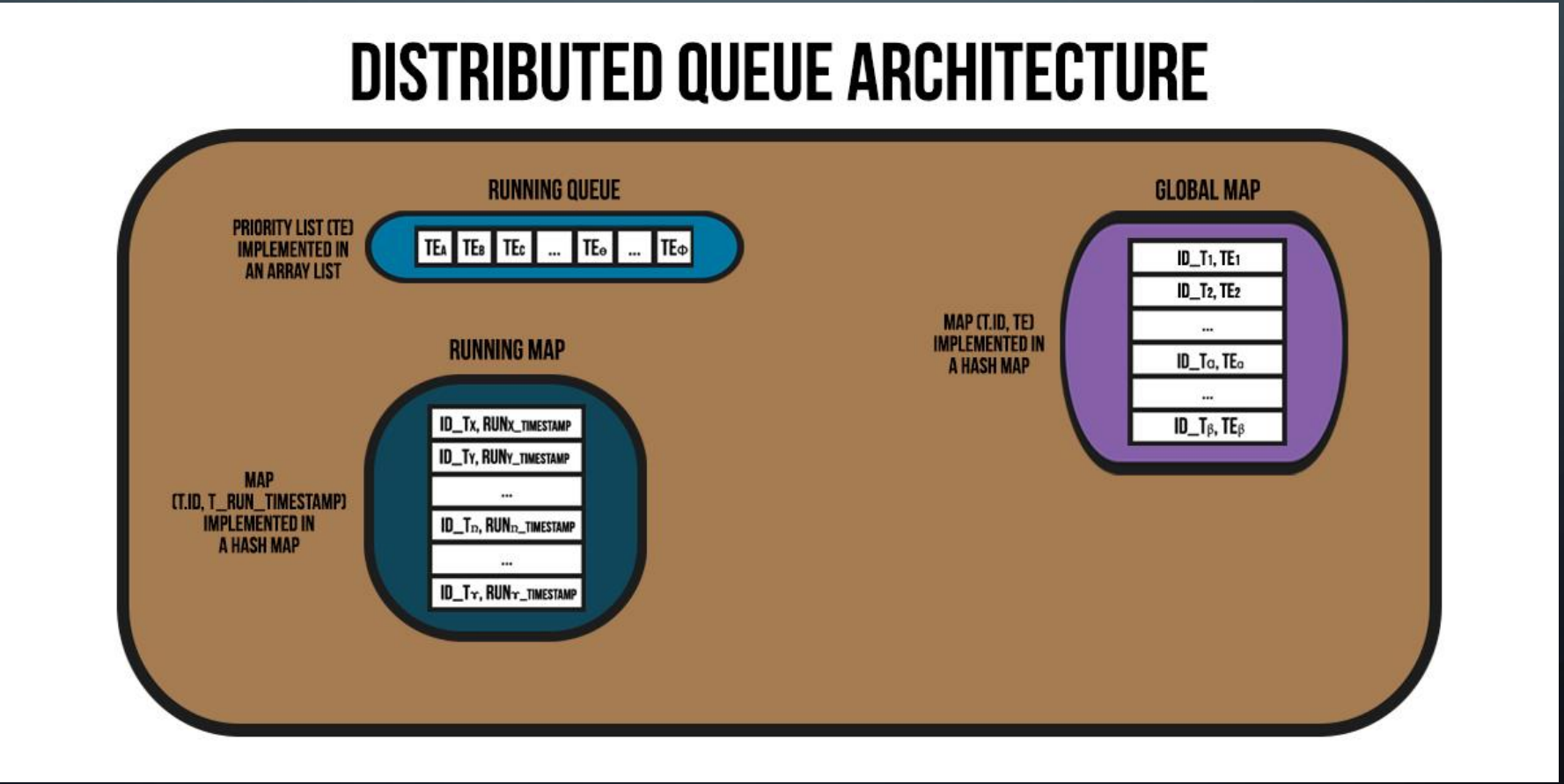
DISTRIBUTED QUEUE SYSTEM ARCHITECTURE

- The Distributed Queue System Architecture it's composed by:



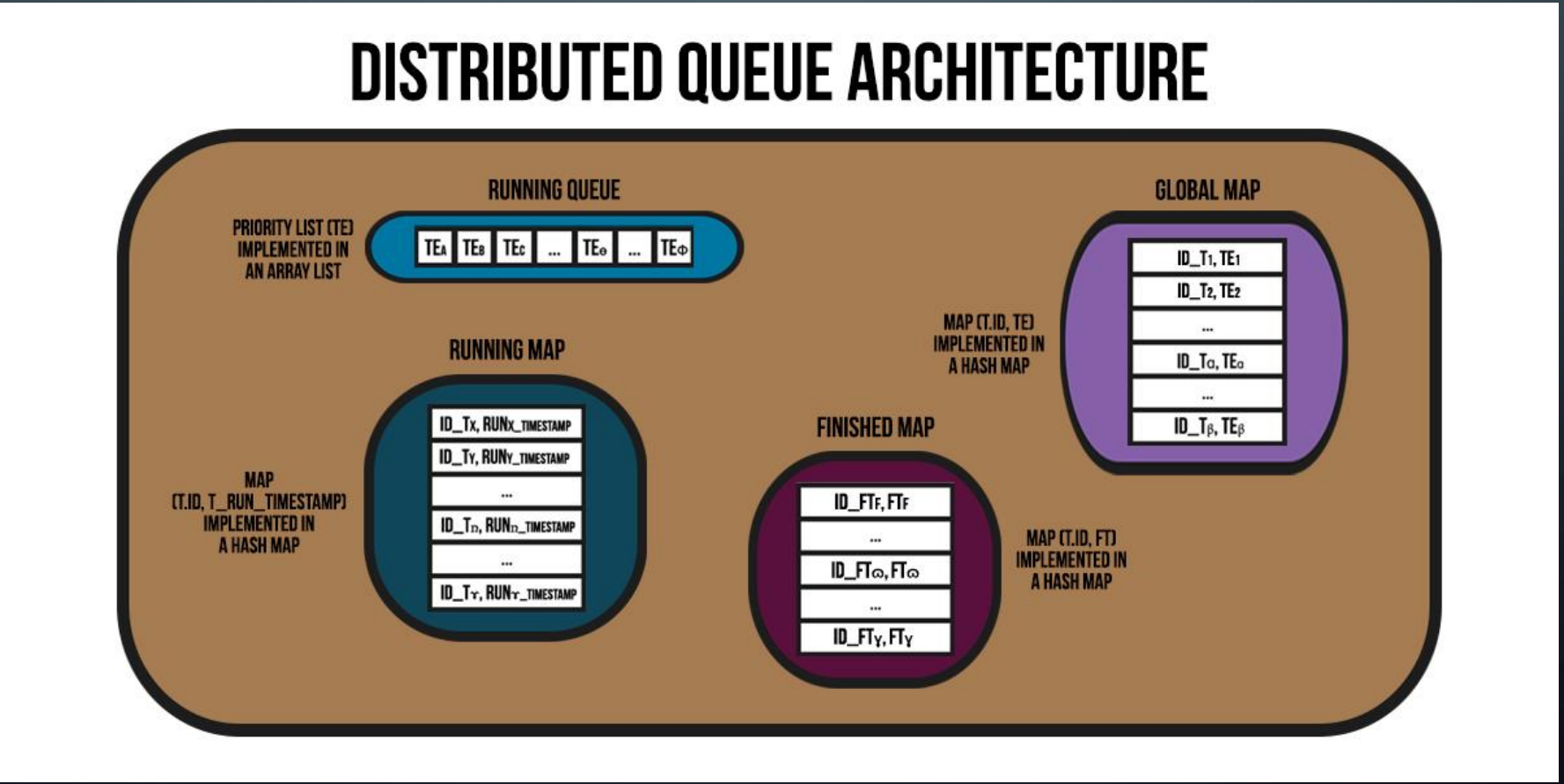
DISTRIBUTED QUEUE SYSTEM ARCHITECTURE

- The Distributed Queue System Architecture it's composed by:



DISTRIBUTED QUEUE SYSTEM ARCHITECTURE

- The Distributed Queue System Architecture it's composed by:



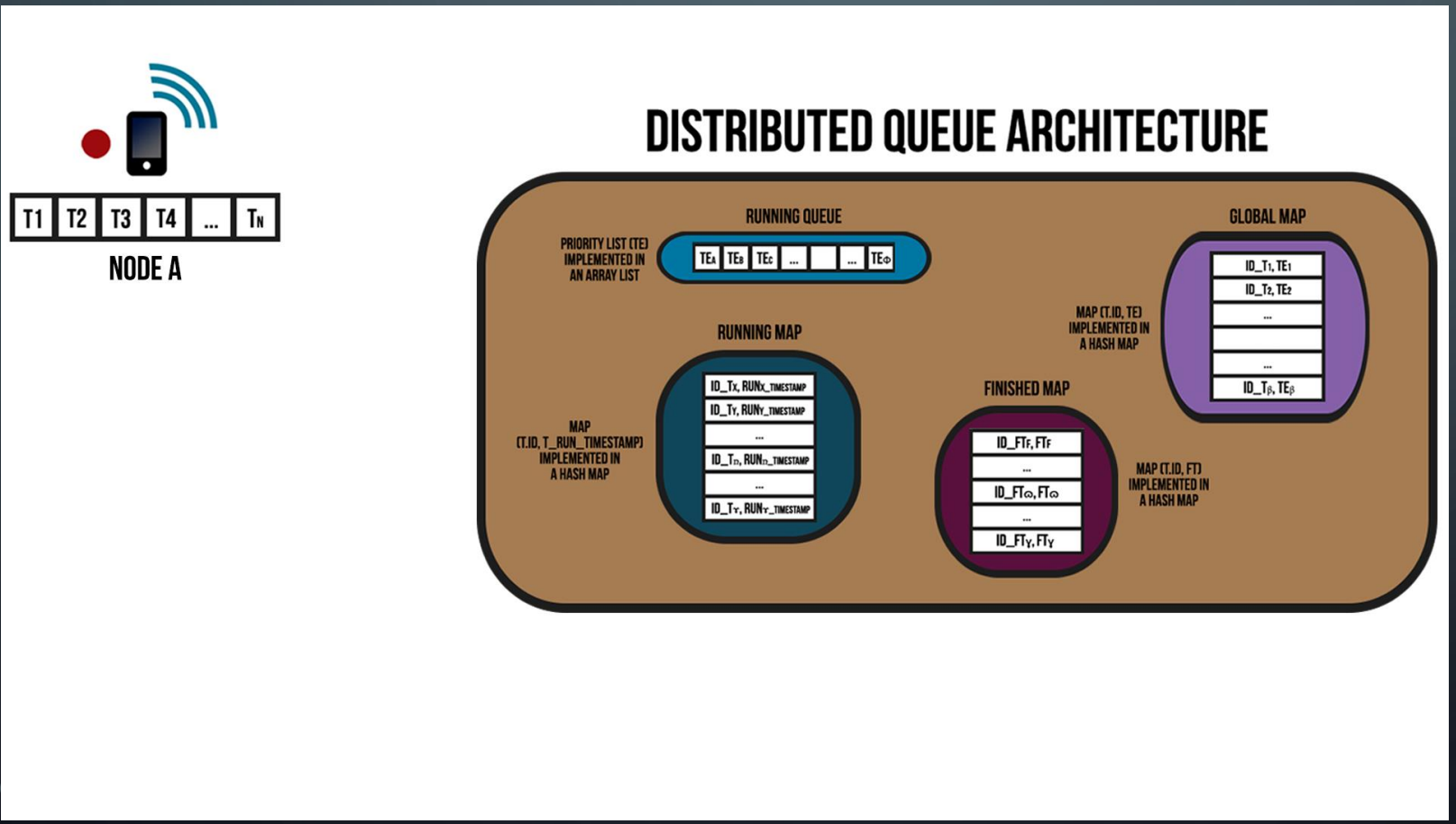
ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

STEPS/OPERATION DESCRITION:

ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

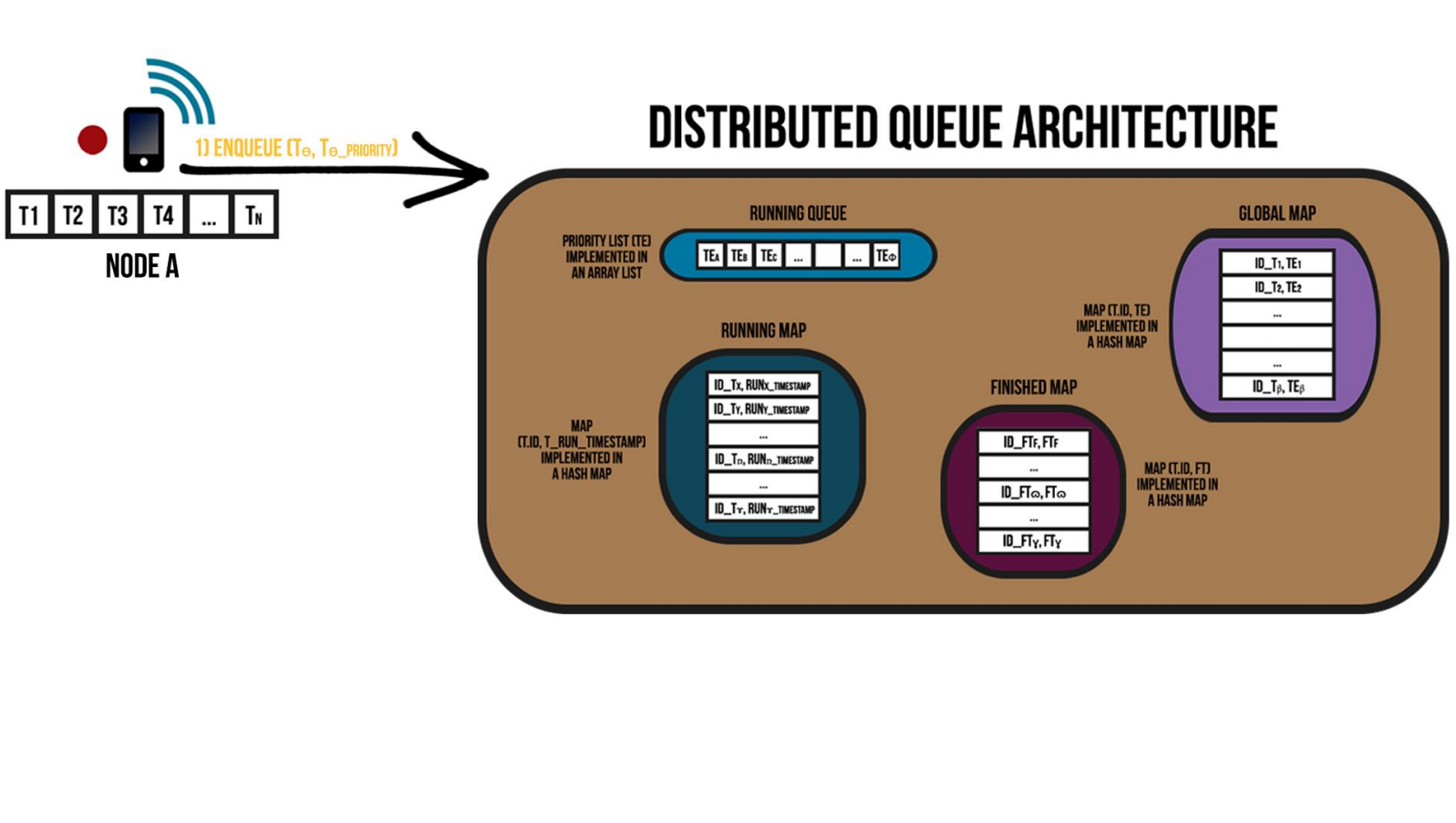
- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:



STEPS/OPERATION DESCRITION:

ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

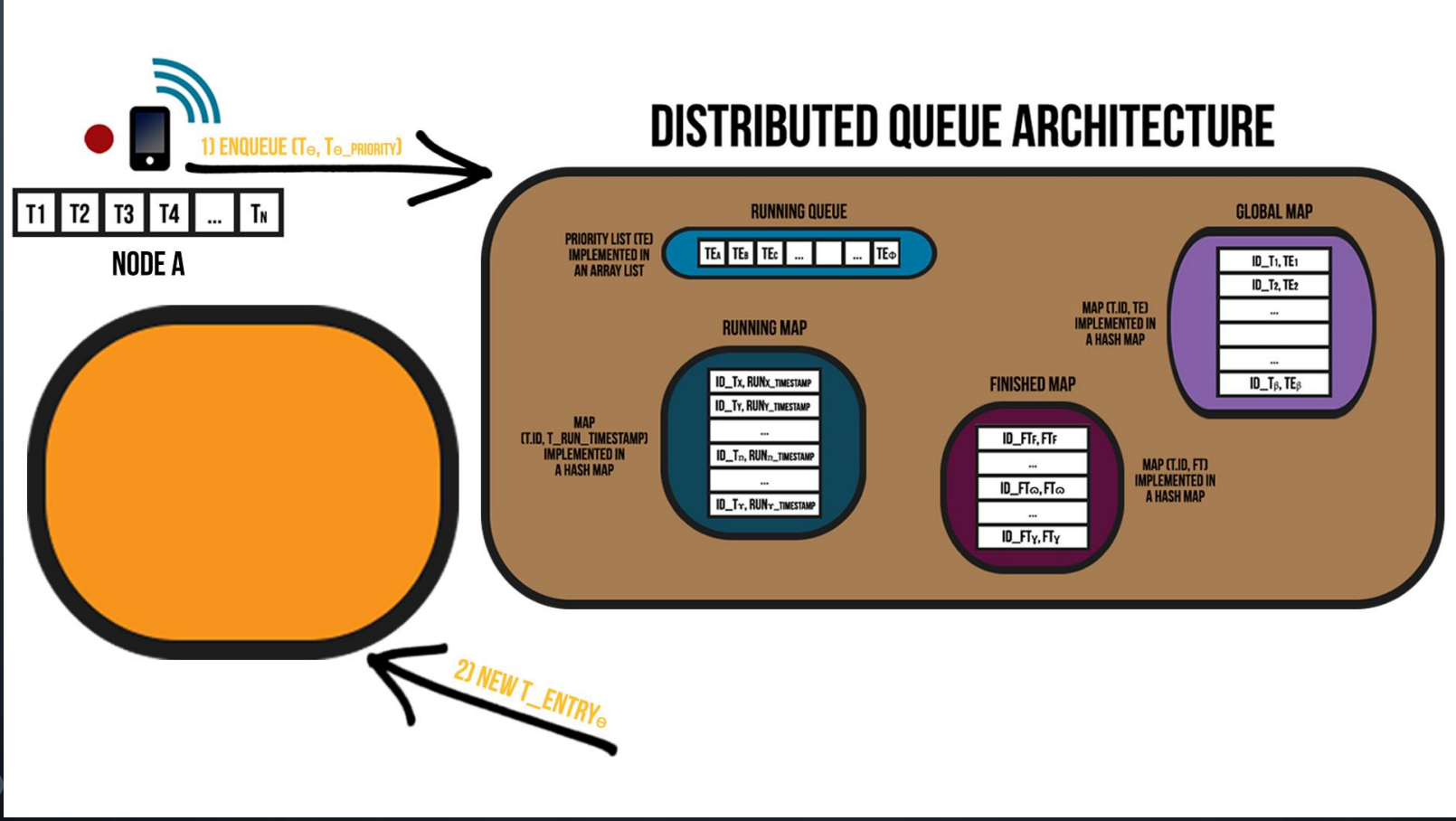


STEPS/OPERATION DESCRITION:

1) Enqueue (Te, Te_Priority)

ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

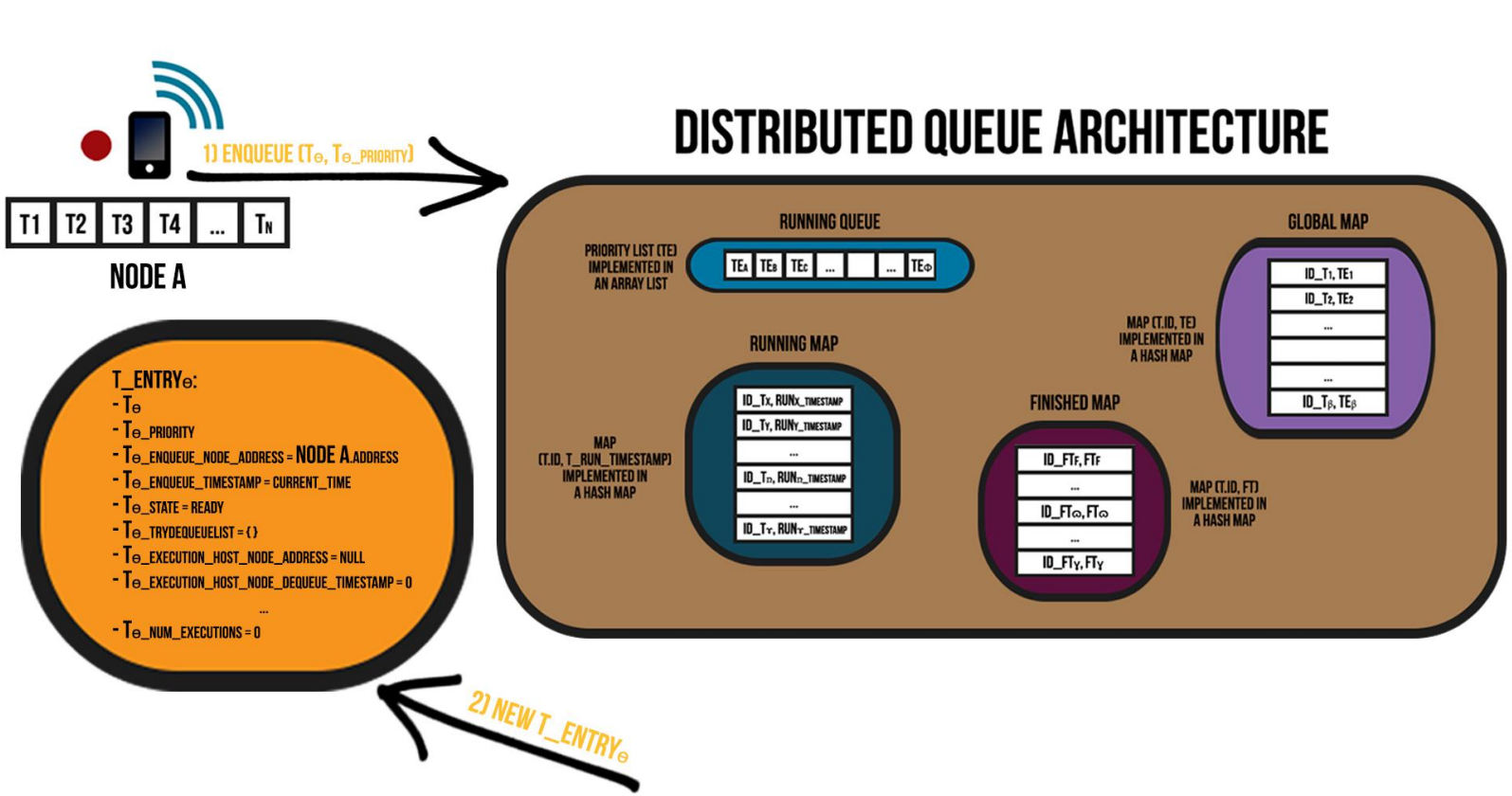


STEPS/OPERATION DESCRITION:

1) Enqueue (Te, Te_Priority)
2) New T_Entrye

ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

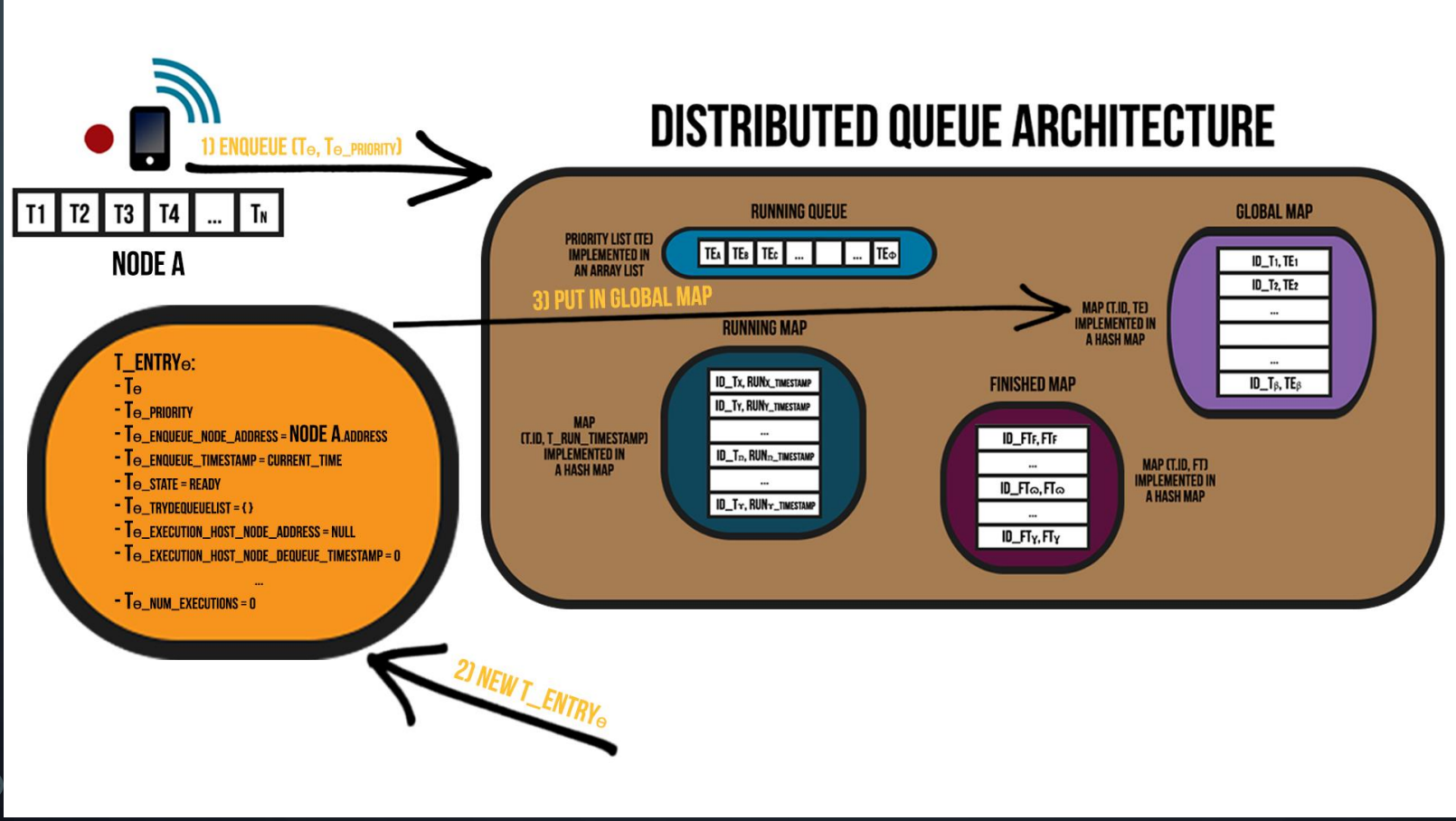


STEPS/OPERATION DESCRITION:

1) Enqueue (Te, Te_Priority)
2) New T_Entrye

ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

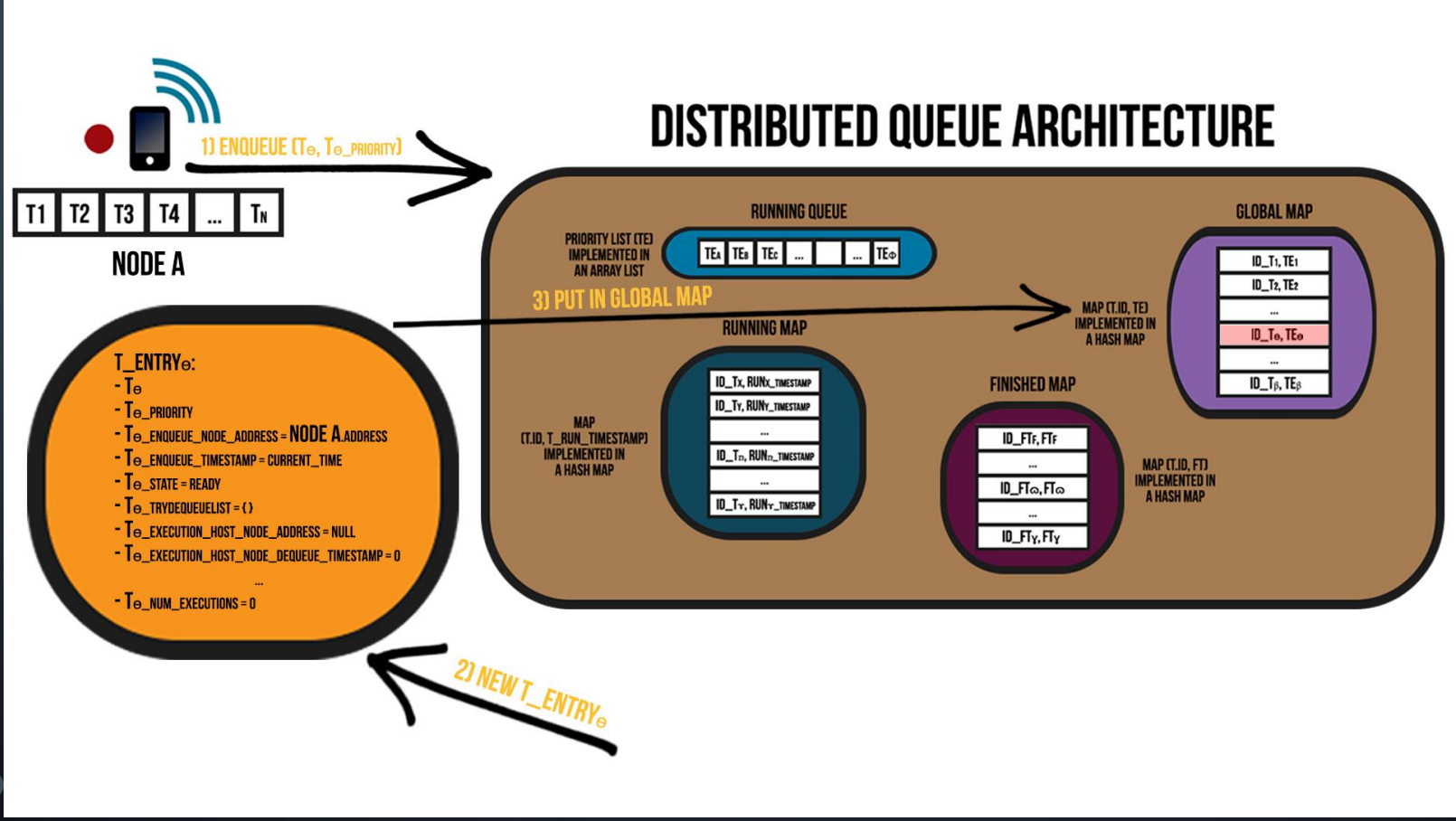


STEPS/OPERATION DESCRITION:

1) Enqueue (Te, Te_Priority)
2) New T_Entrye
3) Put in Global Map

ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

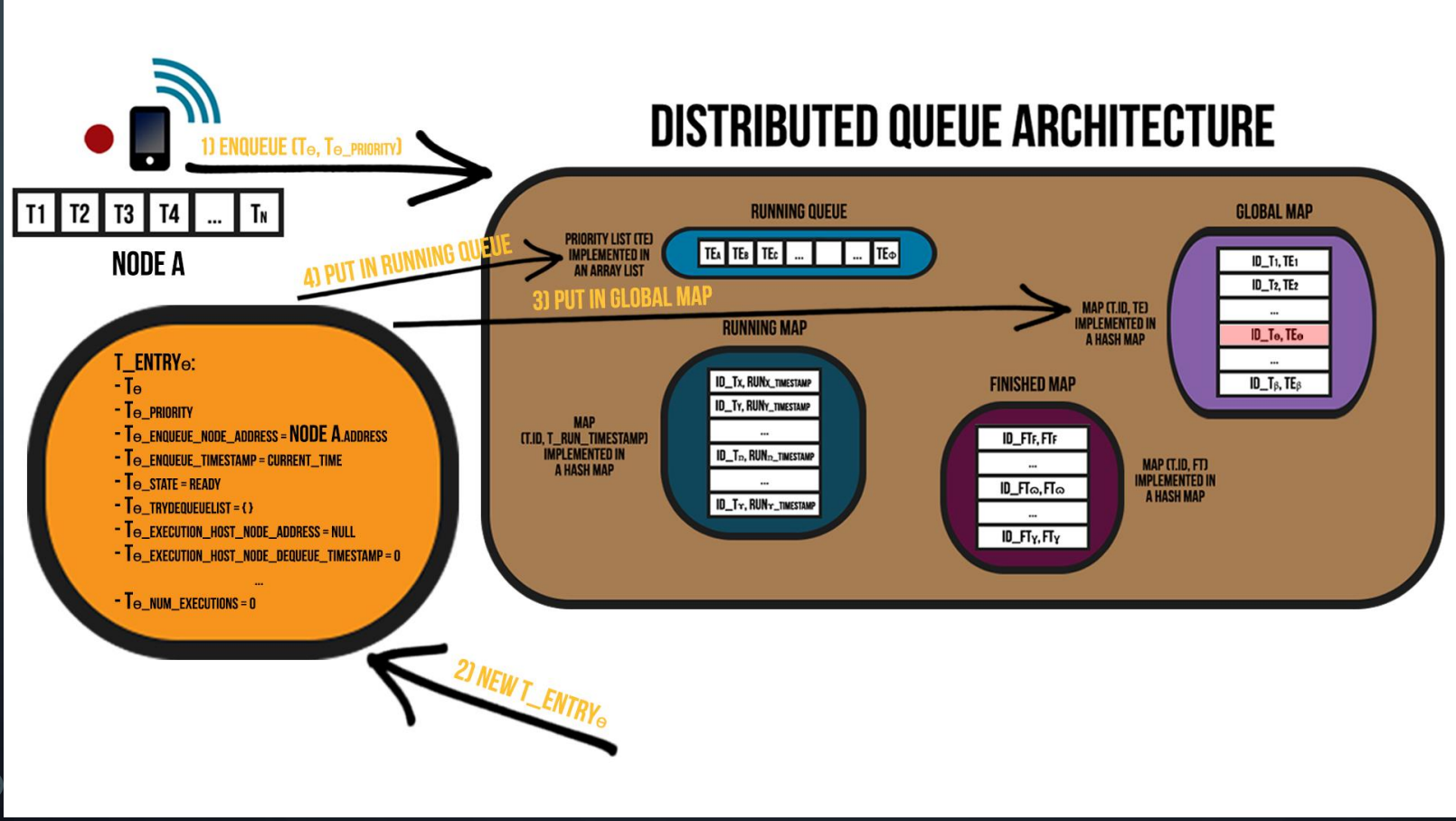


STEPS/OPERATION DESCRITION:

1) Enqueue (Te, Te_Priority)
2) New T_Entrye
3) Put in Global Map

ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

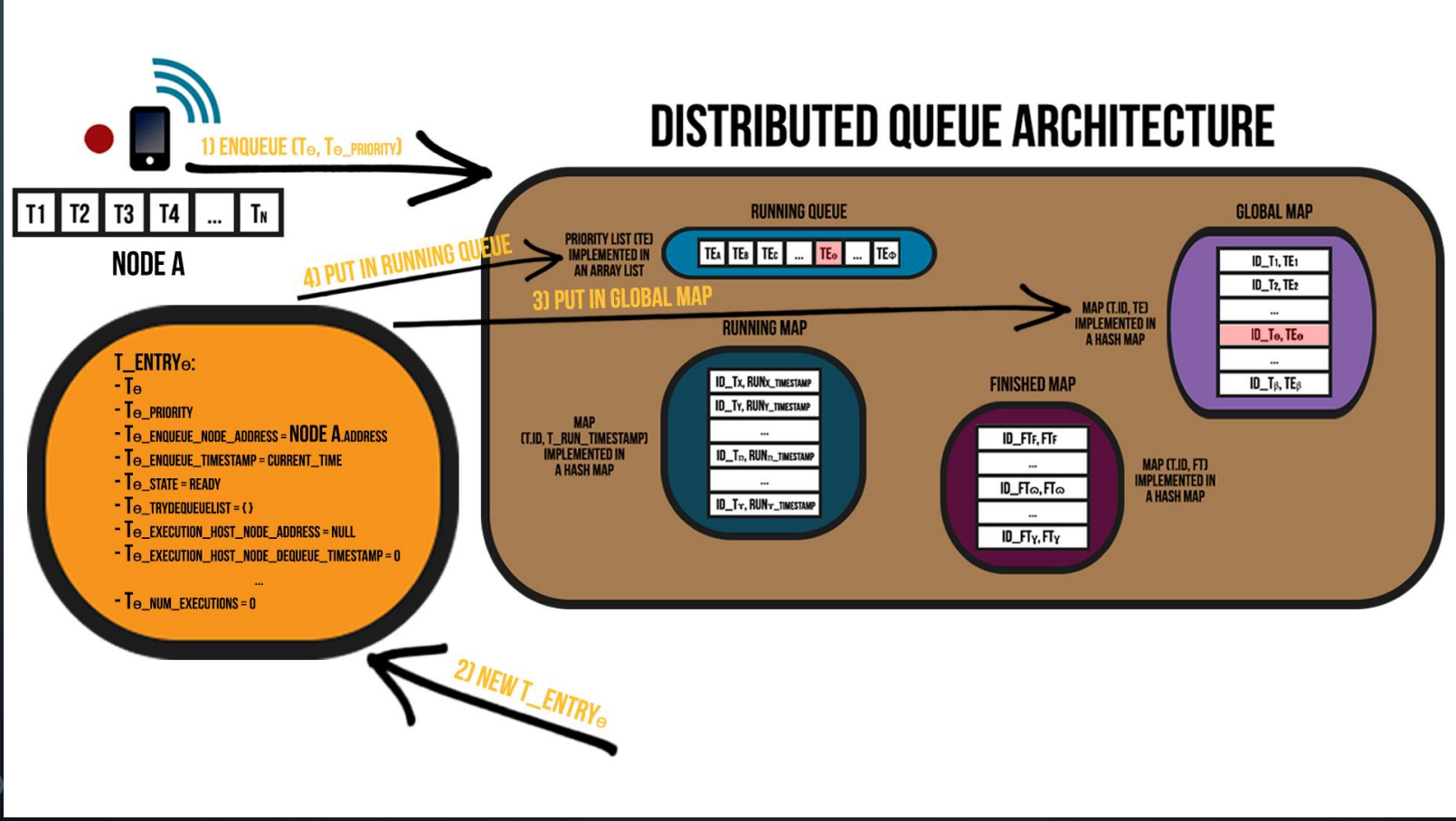


STEPS/OPERATION DESCRITION:

1) Enqueue (Te, Te_Priority)
2) New T_Entrye
3) Put in Global Map
4) Put in Running Queue

ENQUEUE(Te, Te_PRIORITY) OPERATION (1)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

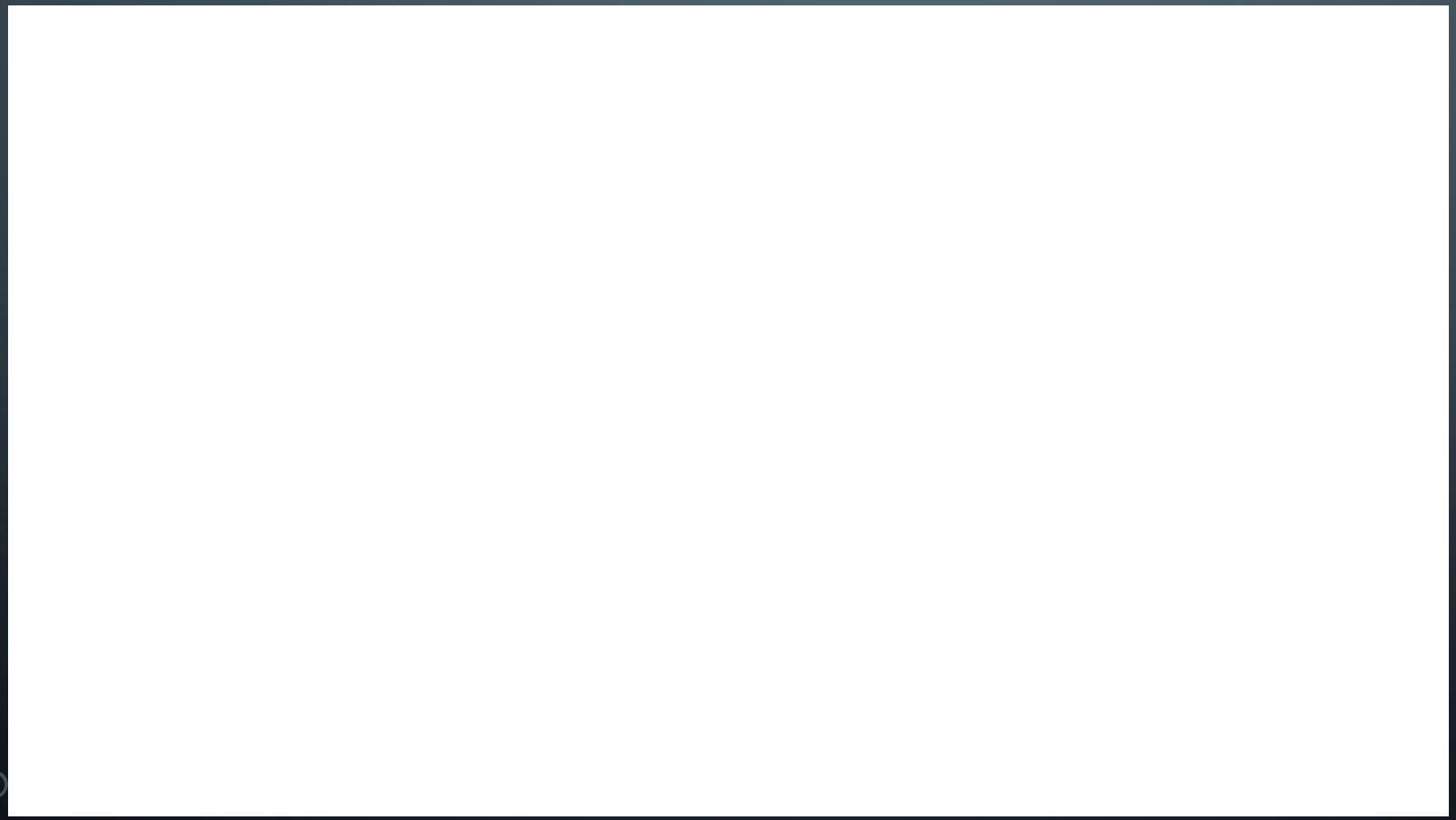


STEPS/OPERATION DESCRITION:

1) Enqueue (Te, Te_Priority)
2) New T_Entrye
3) Put in Global Map
4) Put in Running Queue

ENQUEUE(Te, Te_PRIORITY) OPERATION (2)

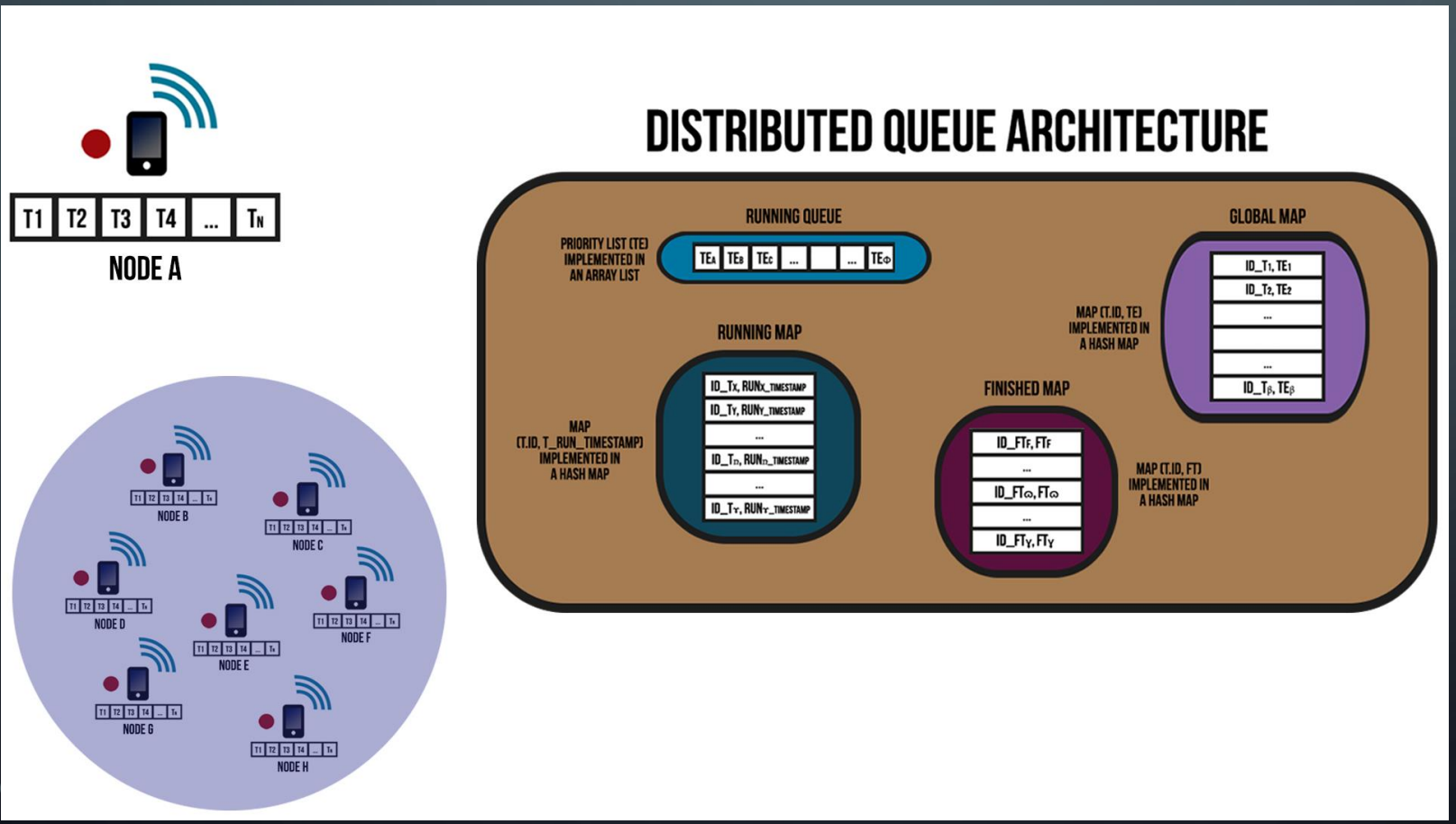
- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:



STEPS/OPERATION DESCRITION:

ENQUEUE(Te, Te_PRIORITY) OPERATION (2)

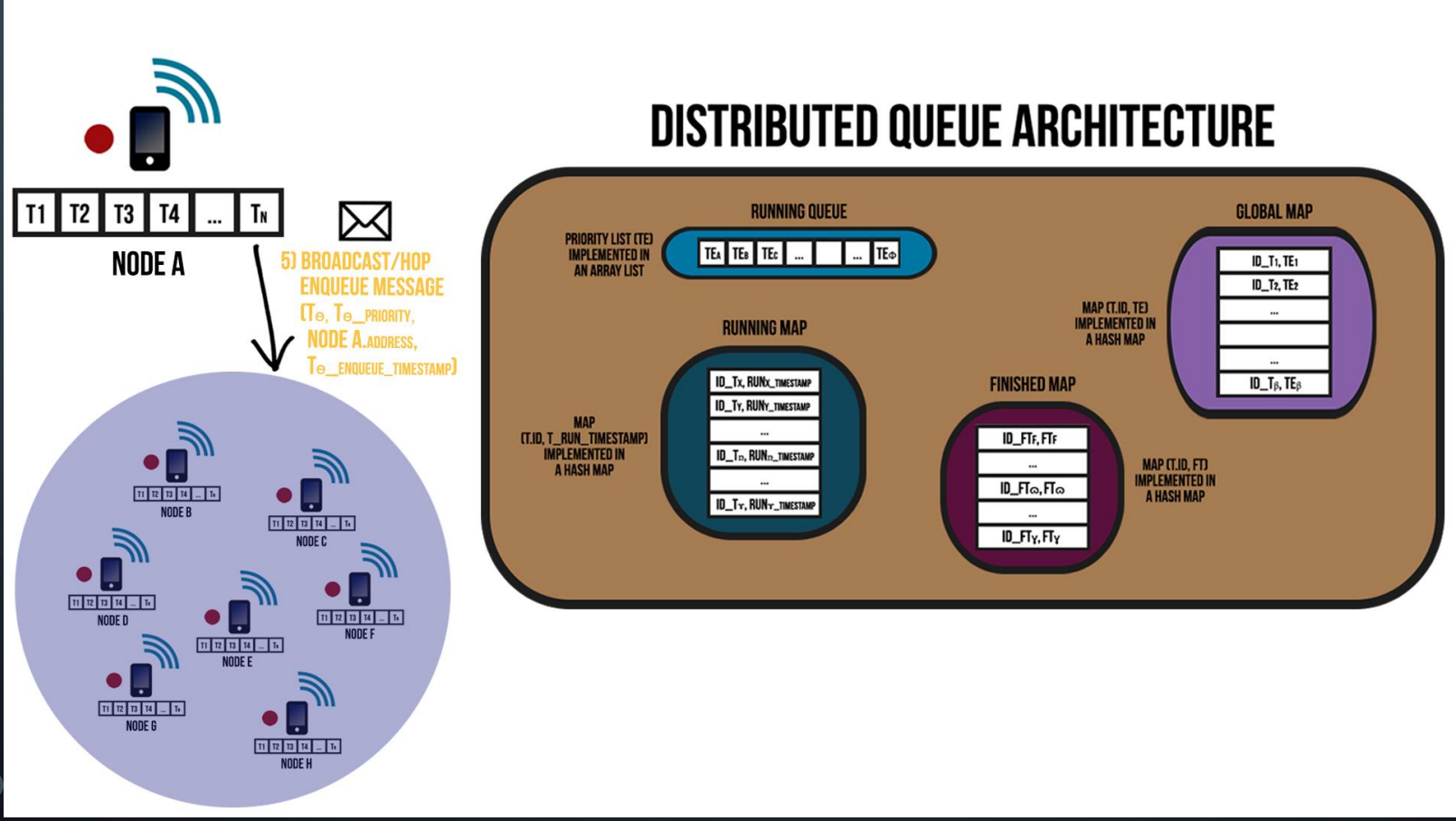
- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:



STEPS/OPERATION DESCRITION:

ENQUEUE(Te, Te_PRIORITY) OPERATION (2)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

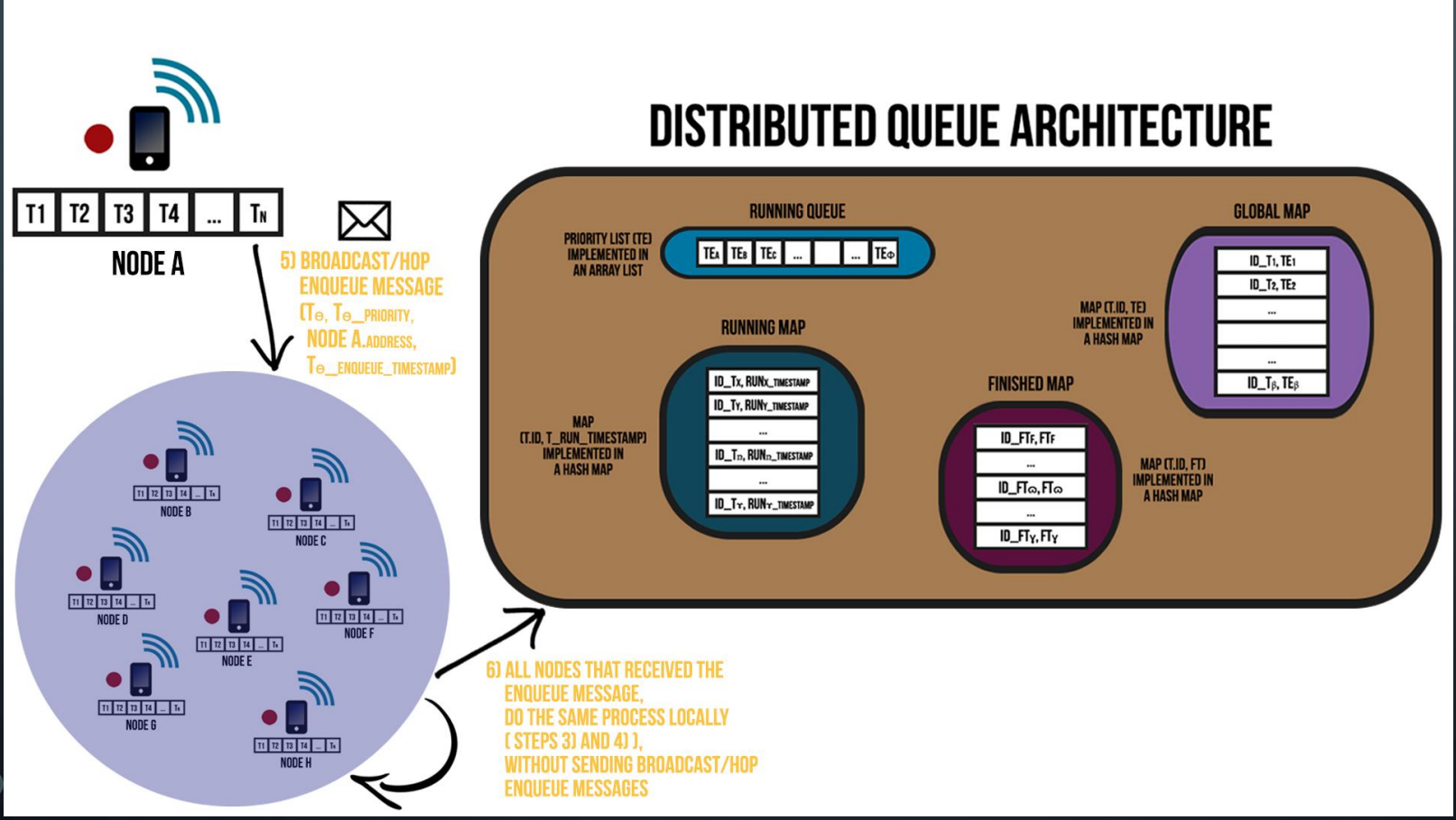


STEPS/OPERATION DESCRITION:

5) Broadcast/Hop Enqueue message (Te, Te_Priority, Node A.Address, Te_Enqueue_Timestamp)

ENQUEUE(Te, Te_PRIORITY) OPERATION (2)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

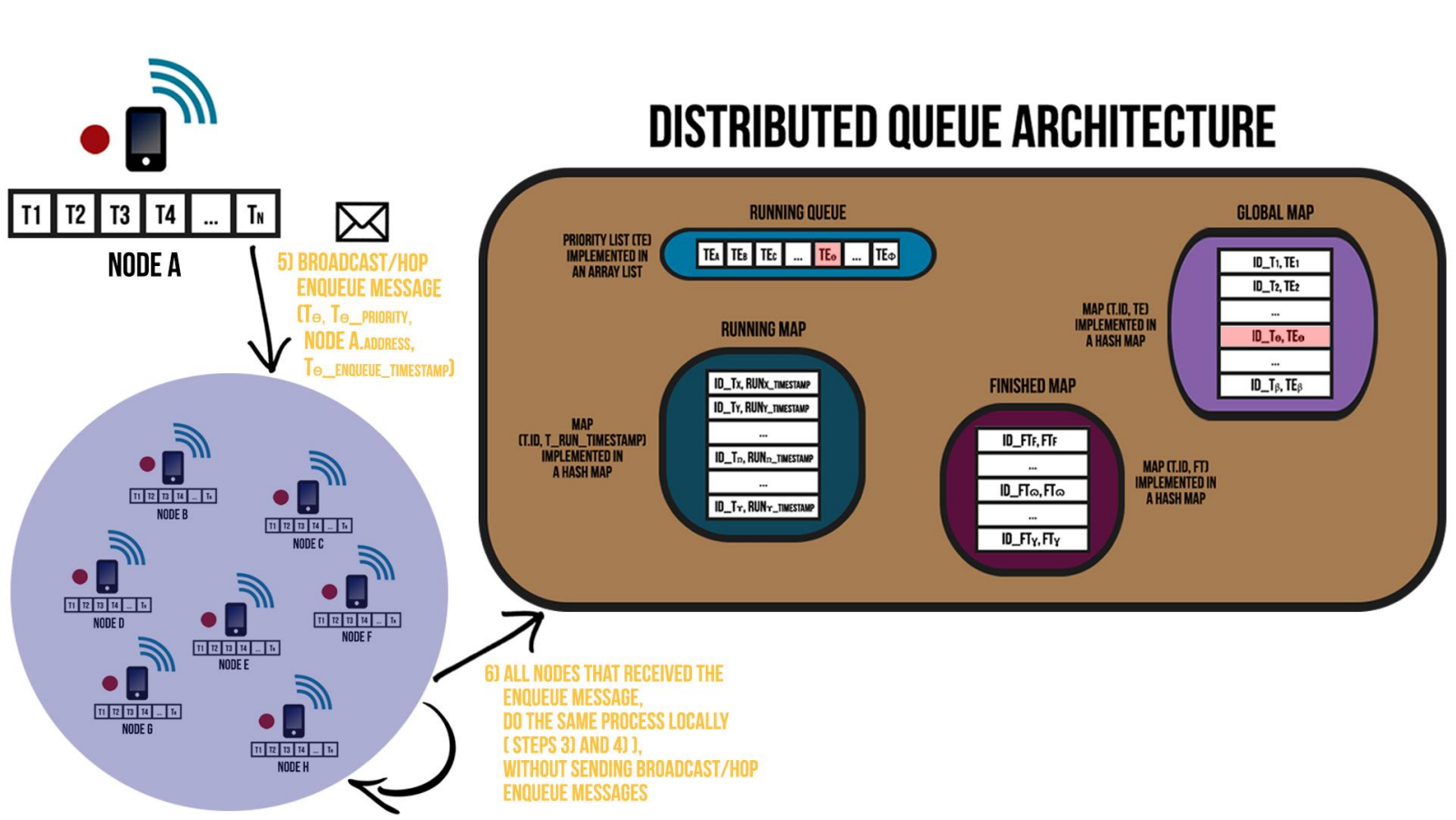


STEPS/OPERATION DESCRITION:

5) Broadcast/Hop Enqueue message (Te, Te_Priority, Node A.Address, Te_Enqueue_Timestamp)
6) All Nodes that received the ENQUEUE message, do the same process locally (Steps 3) and 4)), without sending Broadcast/Hop ENQUEUE messages

ENQUEUE(Te, Te_PRIORITY) OPERATION (2)

- The ENQUEUE(Te, Te_Priority) operation of the Distributed Queue System Architecture have the following behavior:

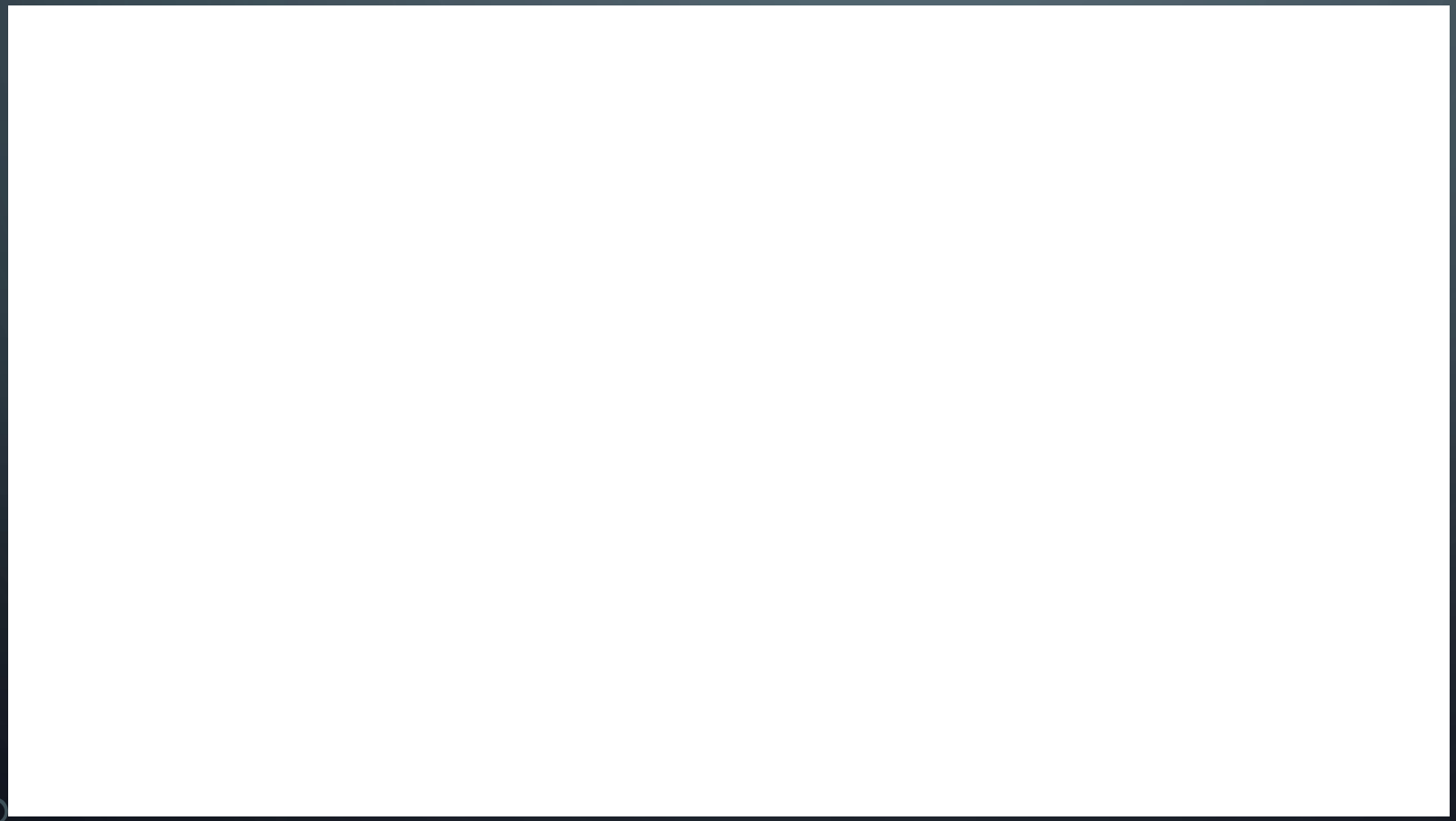


STEPS/OPERATION DESCRITION:

5) Broadcast/Hop Enqueue message (Te, Te_Priority, Node A.Address, Te_Enqueue_Timestamp)
6) All Nodes that received the ENQUEUE message, do the same process locally (Steps 3) and 4)), without sending Broadcast/Hop ENQUEUE messages

TRYDEQUEUE( ) OPERATION (1)

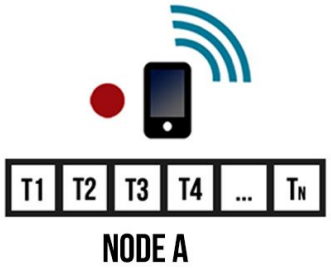
- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

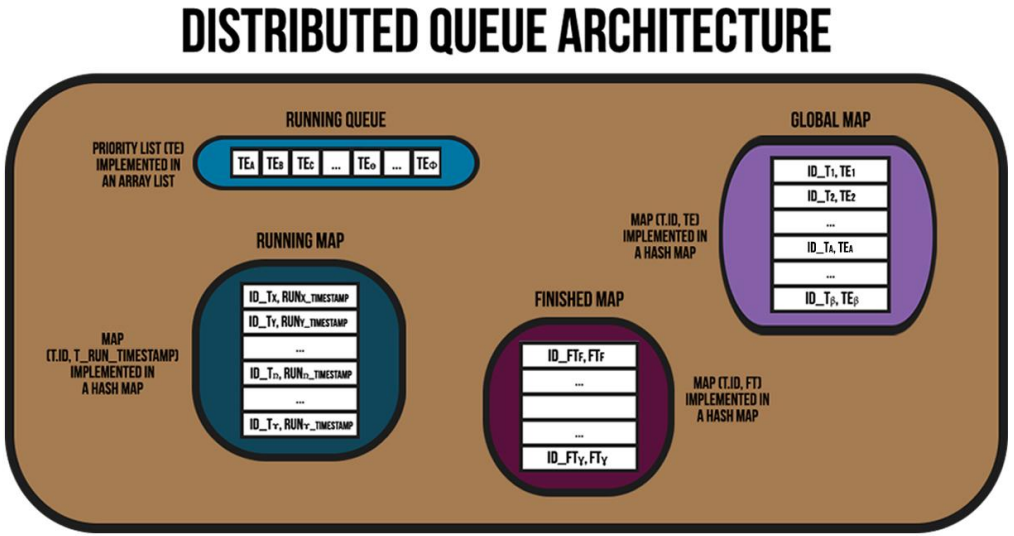


STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE( ) OPERATION (1)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

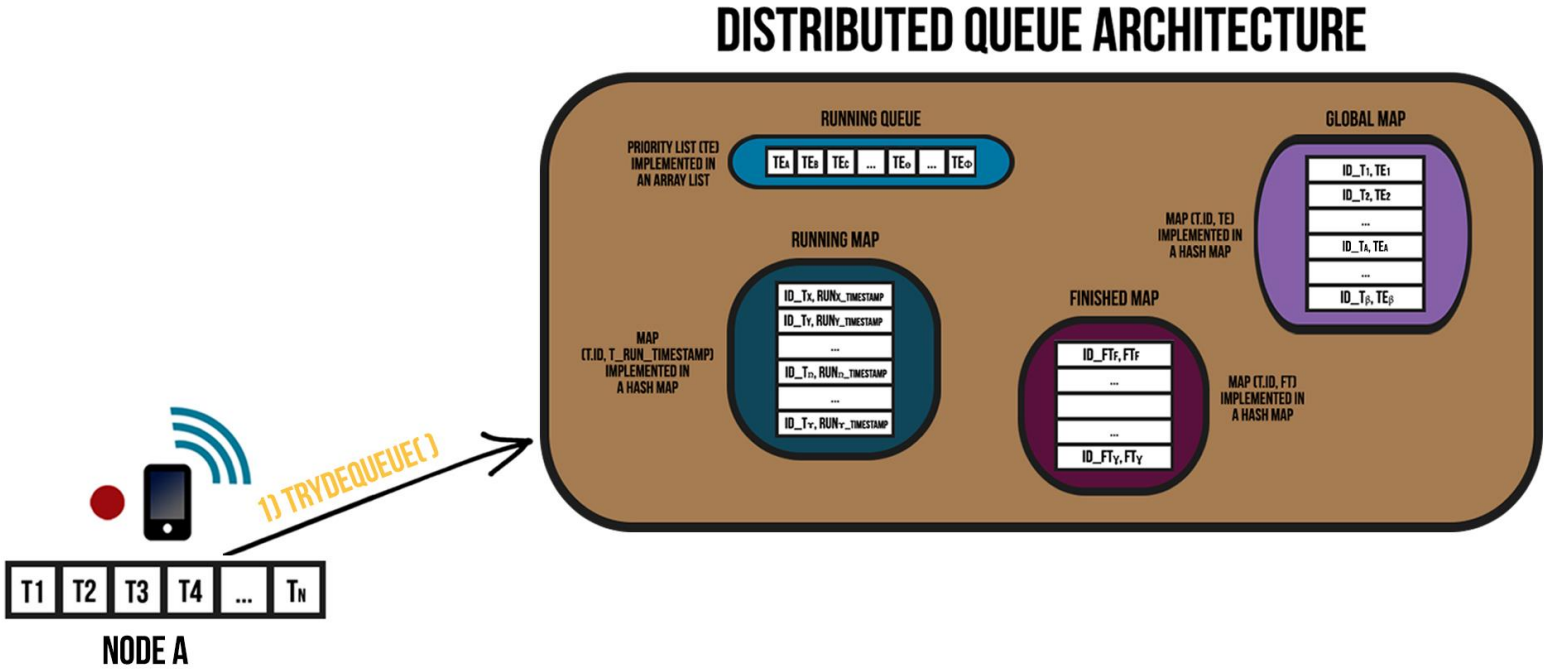




STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE( ) OPERATION (1)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

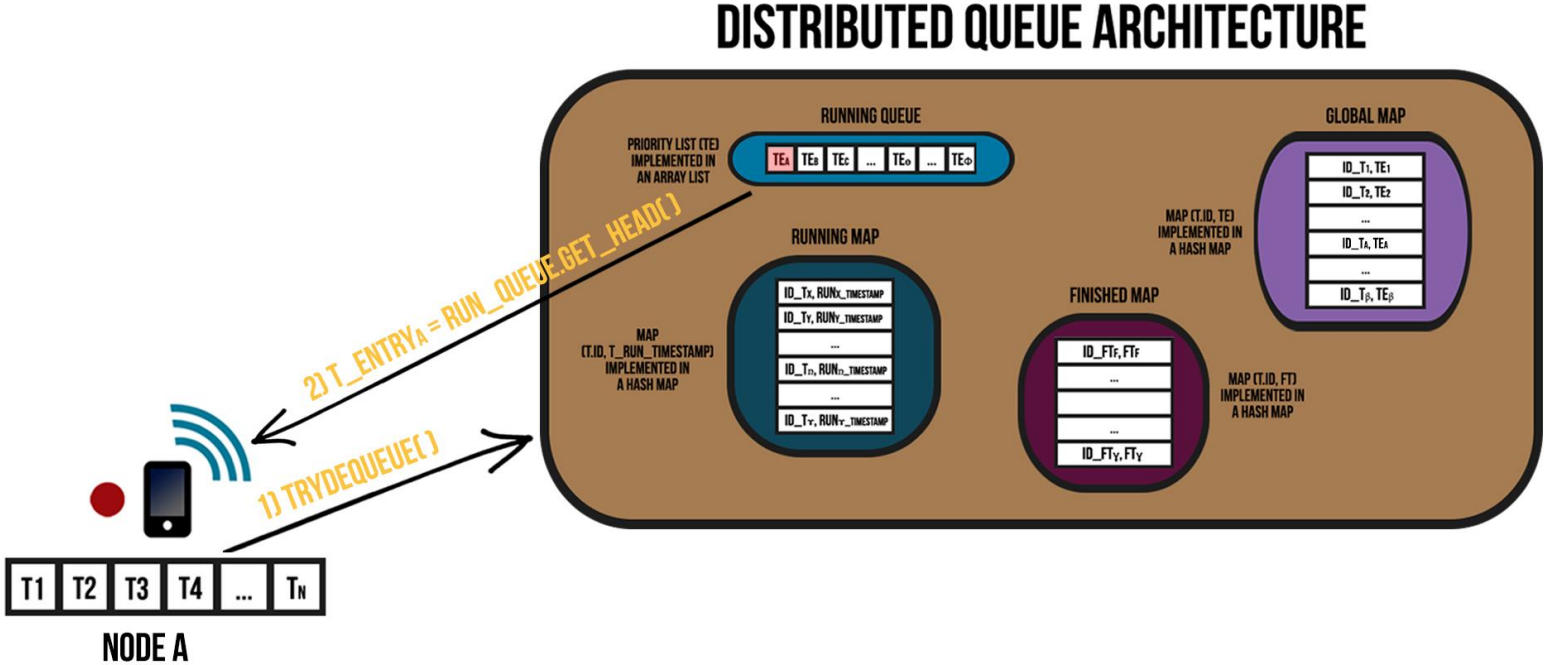


STEPS/OPERATION DESCRIPTION:

1) TryDequeue( )

TRYDEQUEUE( ) OPERATION (1)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

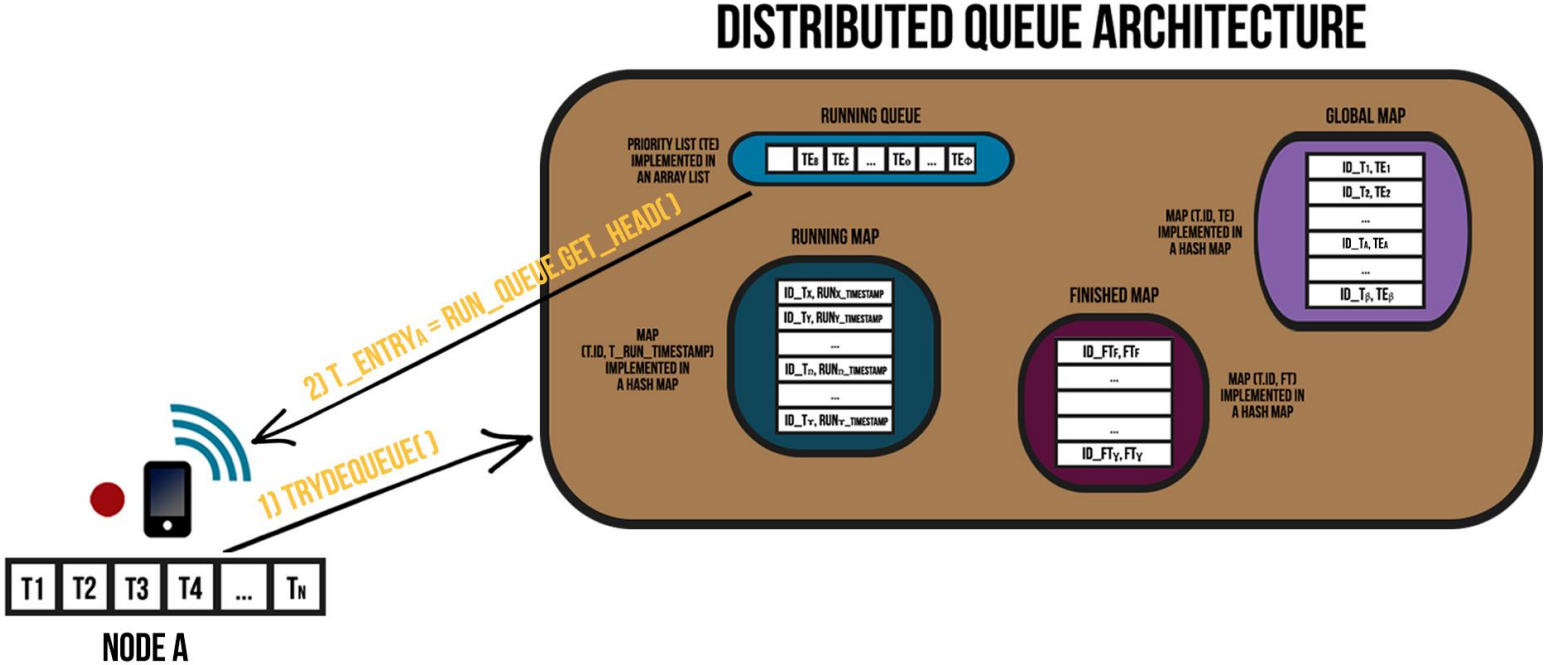


STEPS/OPERATION DESCRIPTION:

1) TryDequeue( )
2) T_Entry$_A$ = RUN_QUEUE.get_Head( )

TRYDEQUEUE( ) OPERATION (1)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

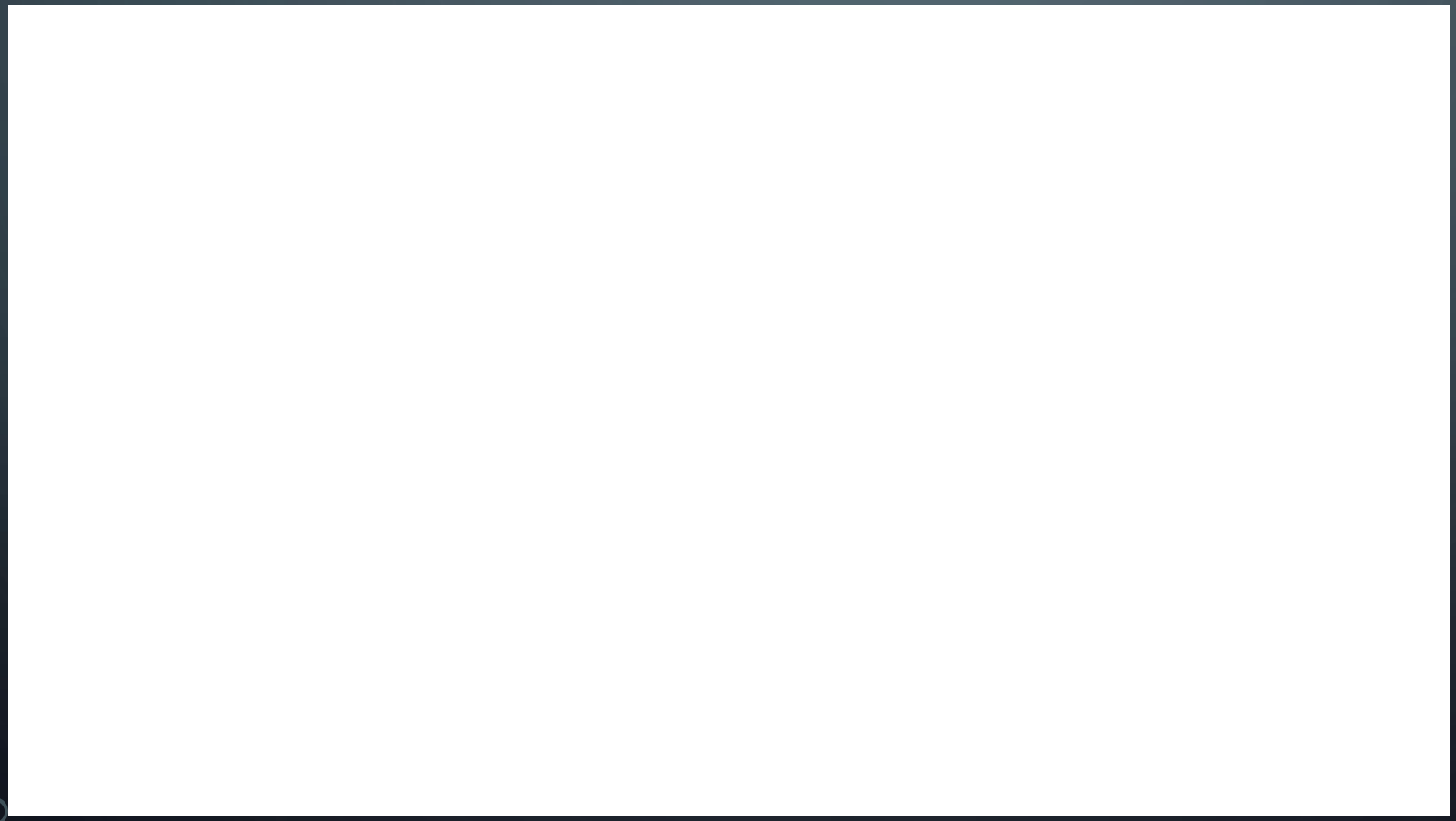


STEPS/OPERATION DESCRIPTION:

1) TryDequeue( )
2) T_Entry_A = RUN_QUEUE.get_Head( )

TRYDEQUEUE( ) OPERATION (2)

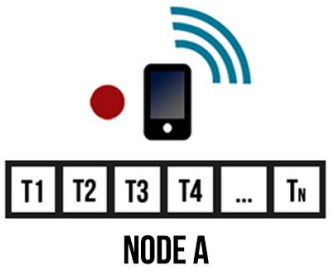
- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

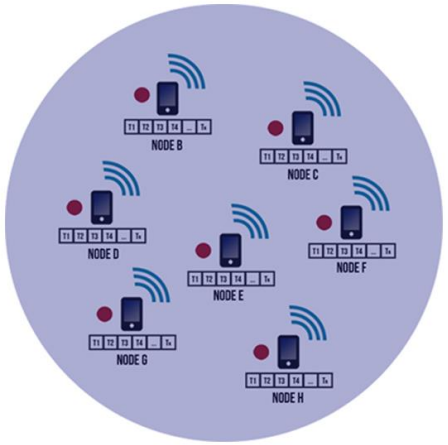


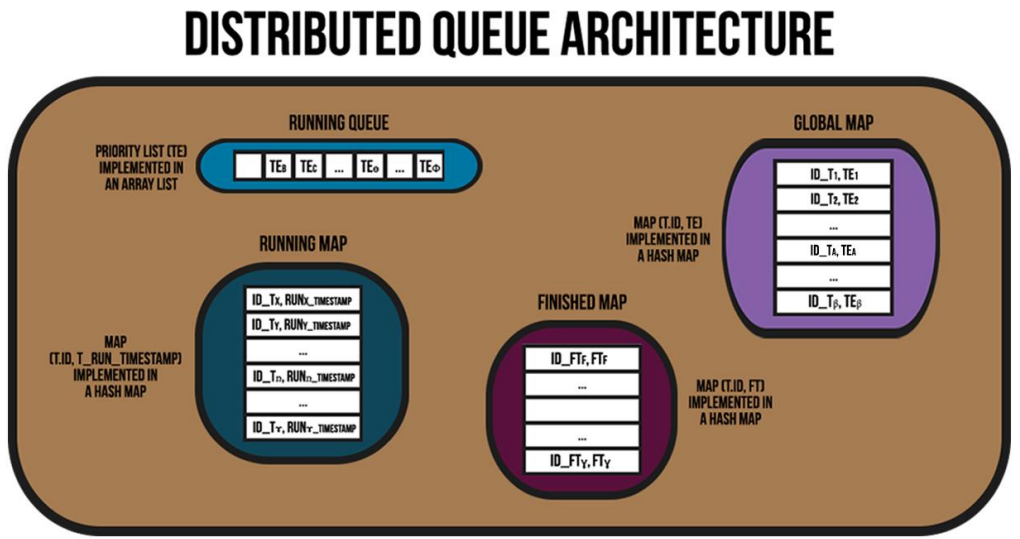
STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE( ) OPERATION (2)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:



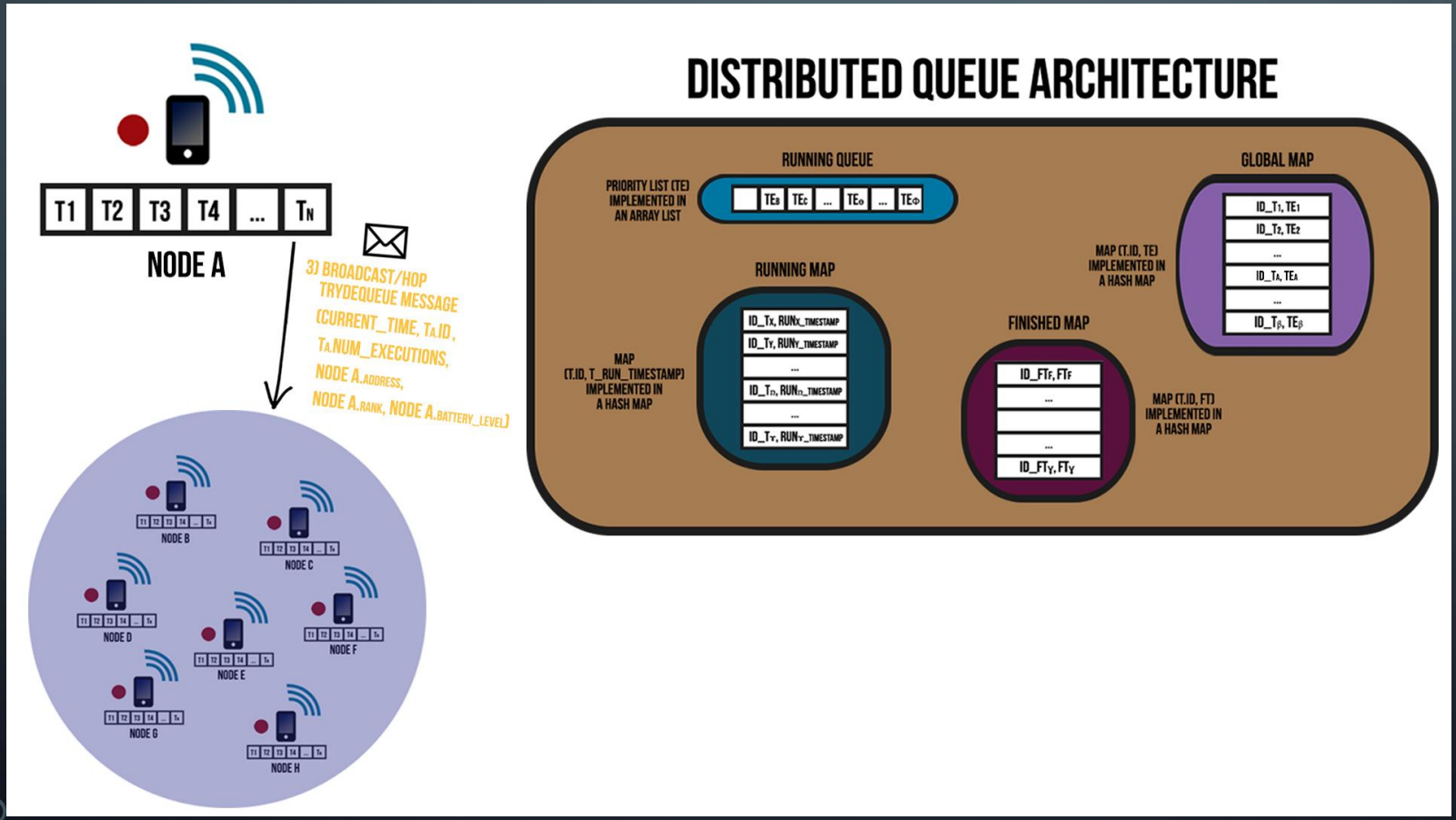




STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE( ) OPERATION (2)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

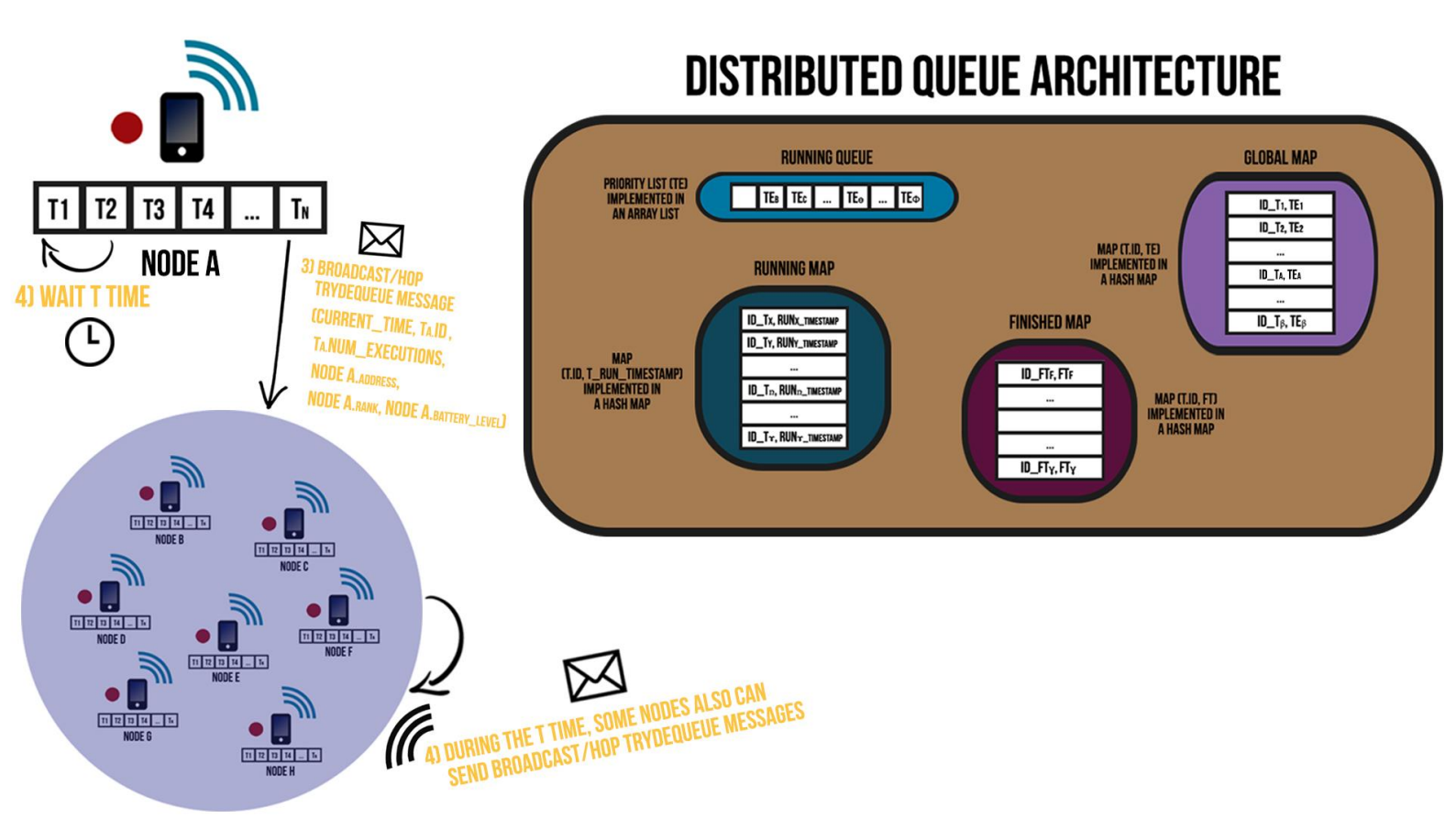


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop TRYDEQUEUE message (Current_Time, T_A.ID, T_A.Num_Executions, Node A.Address, Node A.Rank, Node A.Battery_Level)

TRYDEQUEUE( ) OPERATION (2)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:



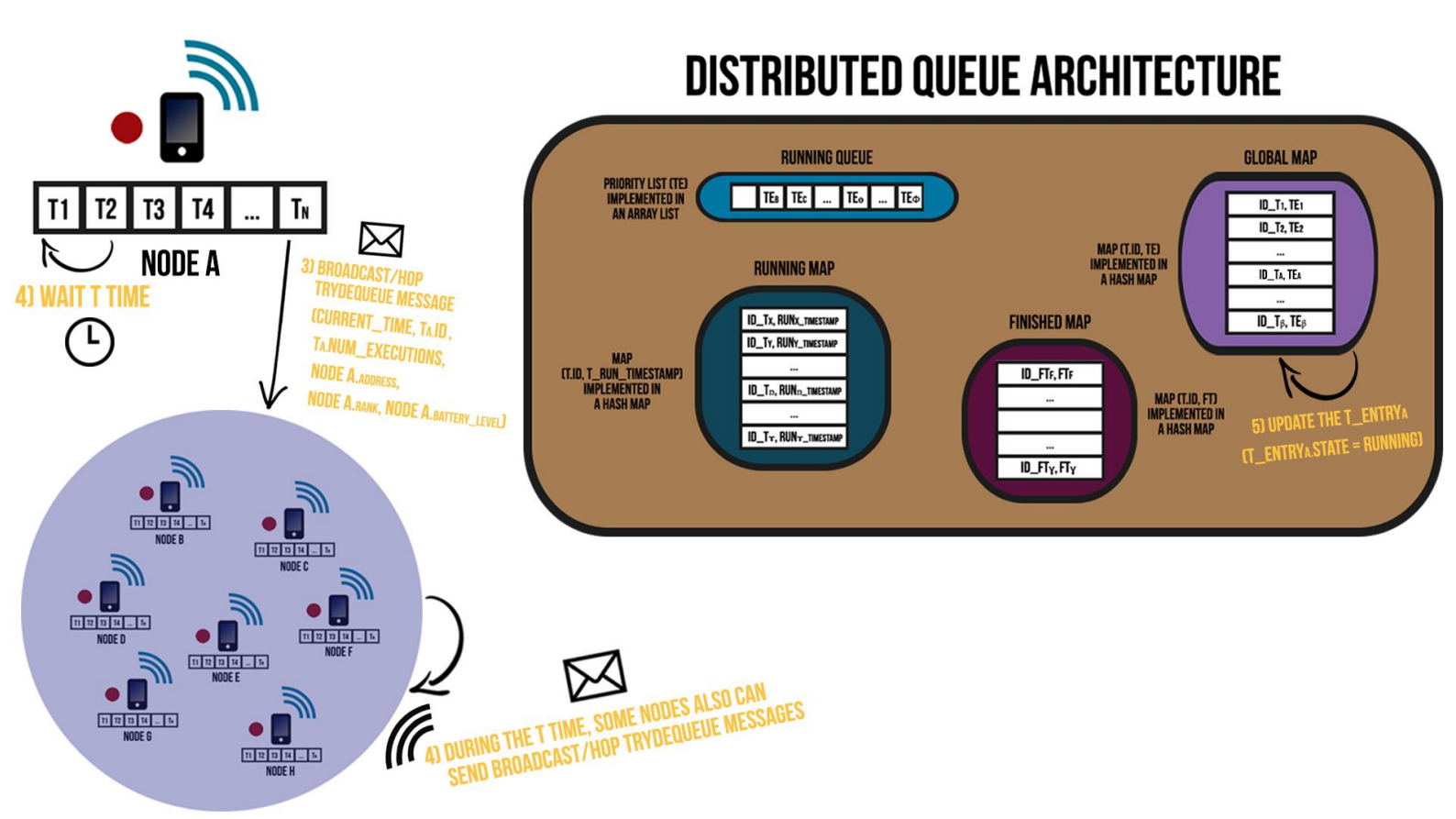
STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop TRYDEQUEUE message (Current_Time, T_A.ID, T_A.Num_Executions, Node A.Address, Node A.Rank, Node A.Battery_Level)
4) Wait T time / During the T time, some Nodes also can send Broadcast/Hop TRYDEQUEUE messages

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE( ) OPERATION (2)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:



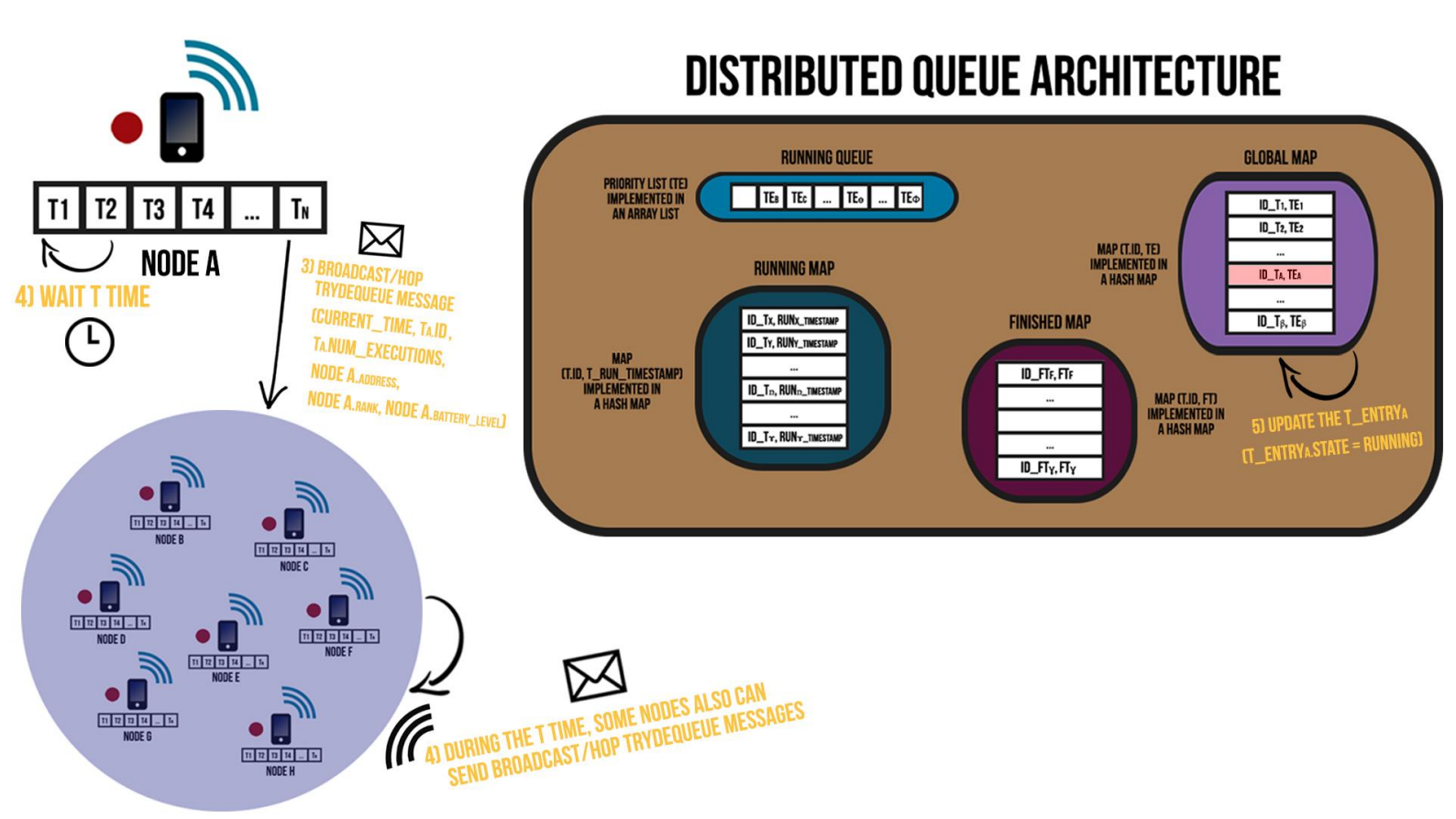
STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop TRYDEQUEUE message (Current_Time, T_A.ID, T_A.Num_Executions, Node A.Address, Node A.Rank, Node A.Battery_Level)
4) Wait T time / During the T time, some Nodes also can send Broadcast/Hop TRYDEQUEUE messages
5) Update the T_Entry_A (T_Entry_A.State = RUNNING)

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE( ) OPERATION (2)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:



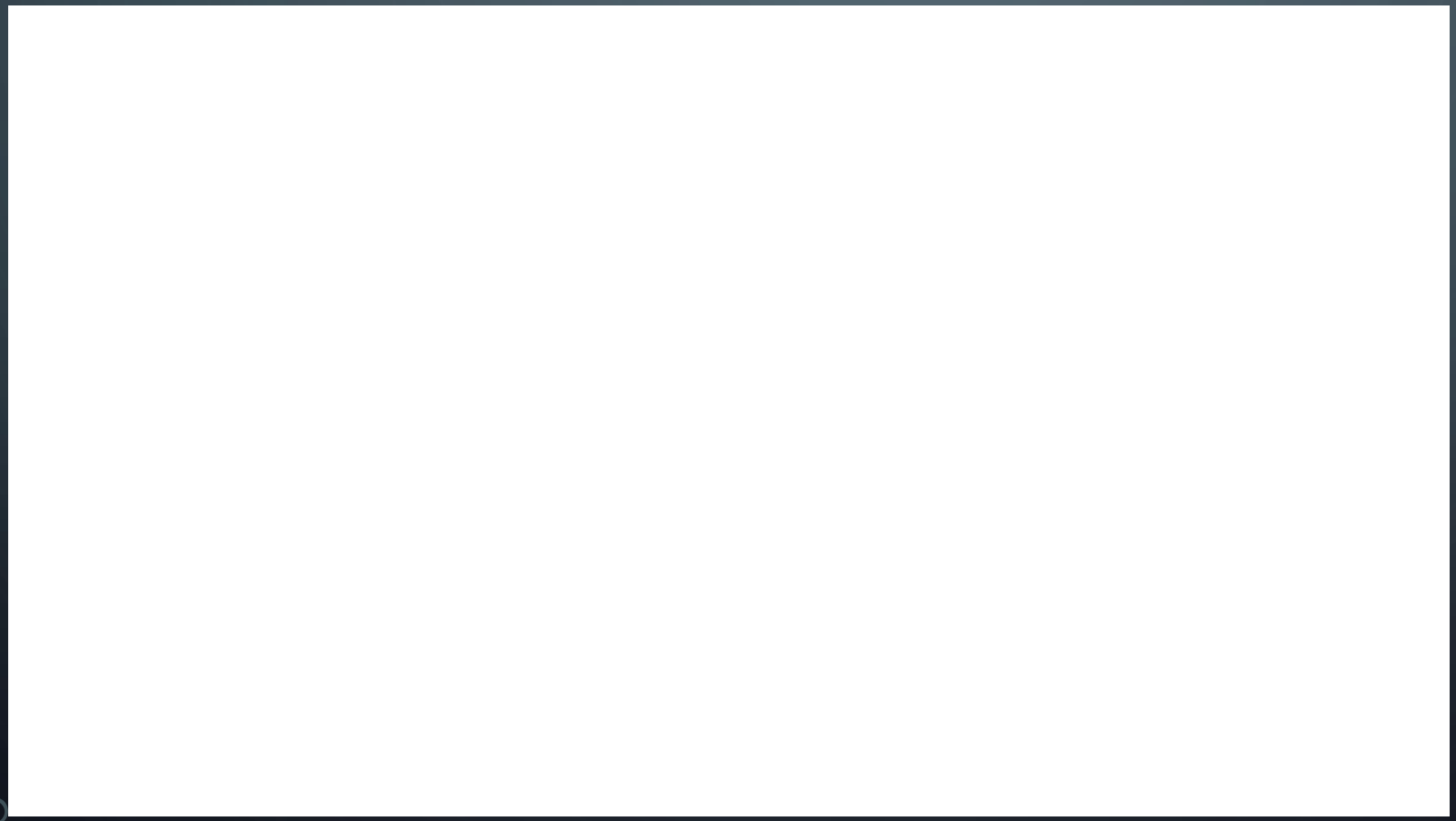
STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop TRYDEQUEUE message (Current_Time, T_A.ID, T_A.Num_Executions, Node A.Address, Node A.Rank, Node A.Battery_Level)
4) Wait T time / During the T time, some Nodes also can send Broadcast/Hop TRYDEQUEUE messages
5) Update the T_Entry_A (T_Entry_A.State = RUNNING)

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE( ) OPERATION (3)

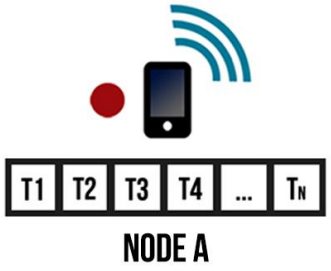
- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

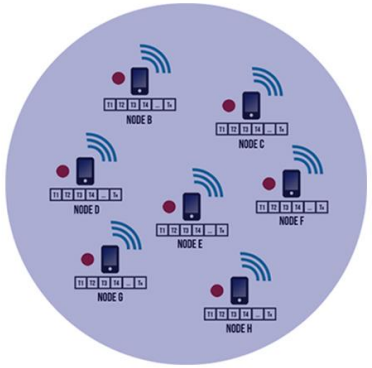


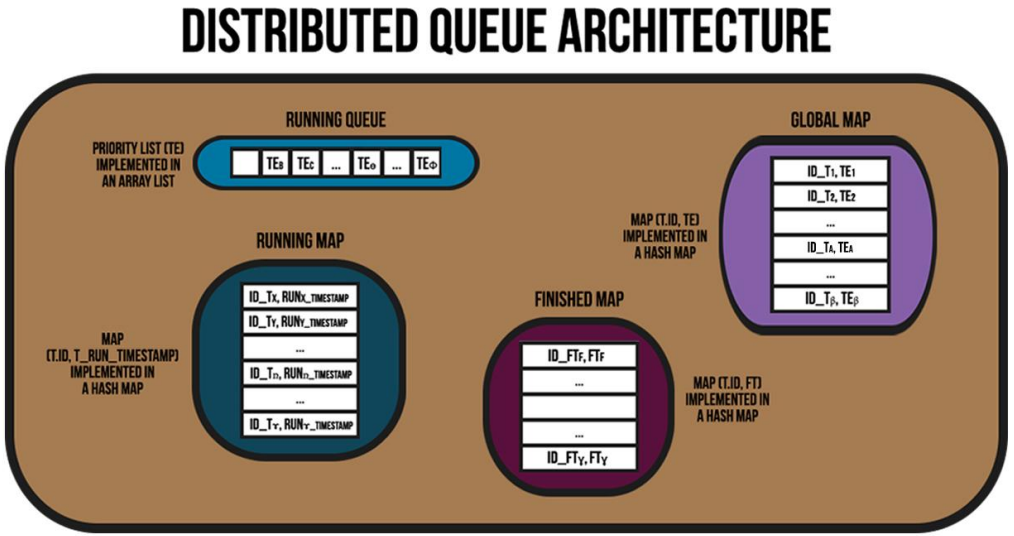
STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE( ) OPERATION (3)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:



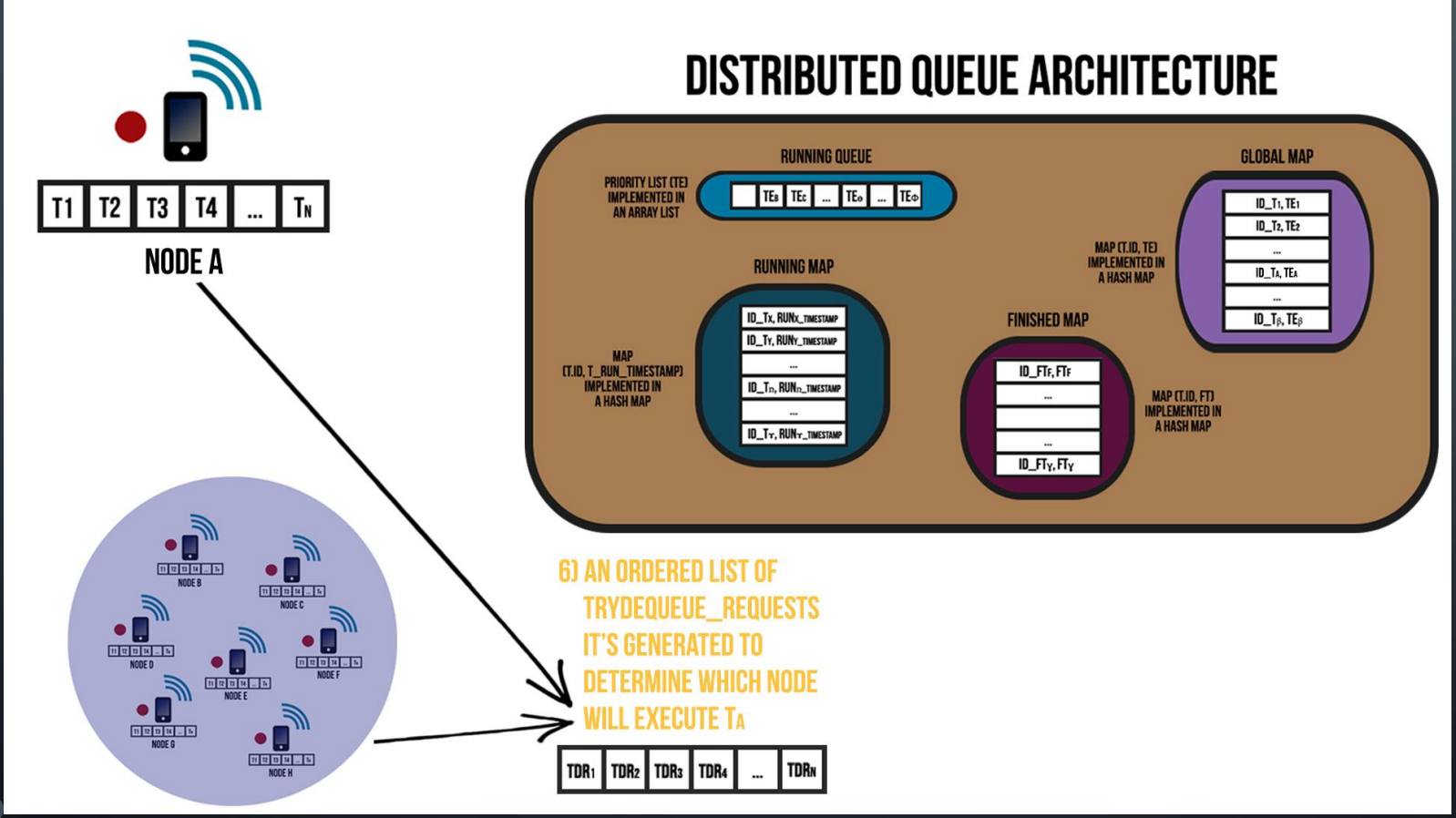




STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE( ) OPERATION (3)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

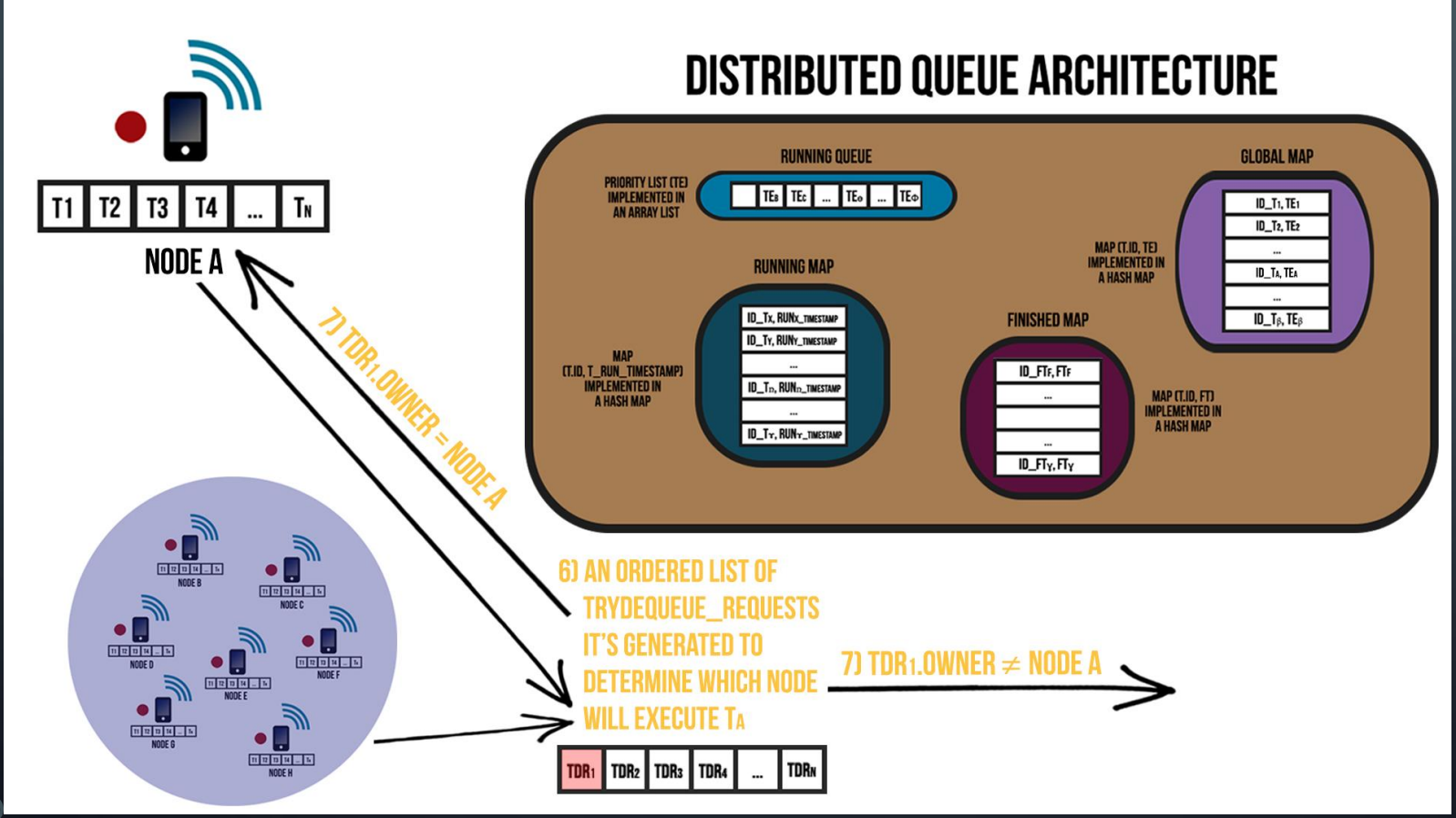


STEPS/OPERATION DESCRIPTION:

6) An ordered list of TryDequeue_Requests it's generated to determine which Node will execute T_A

TRYDEQUEUE( ) OPERATION (3)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

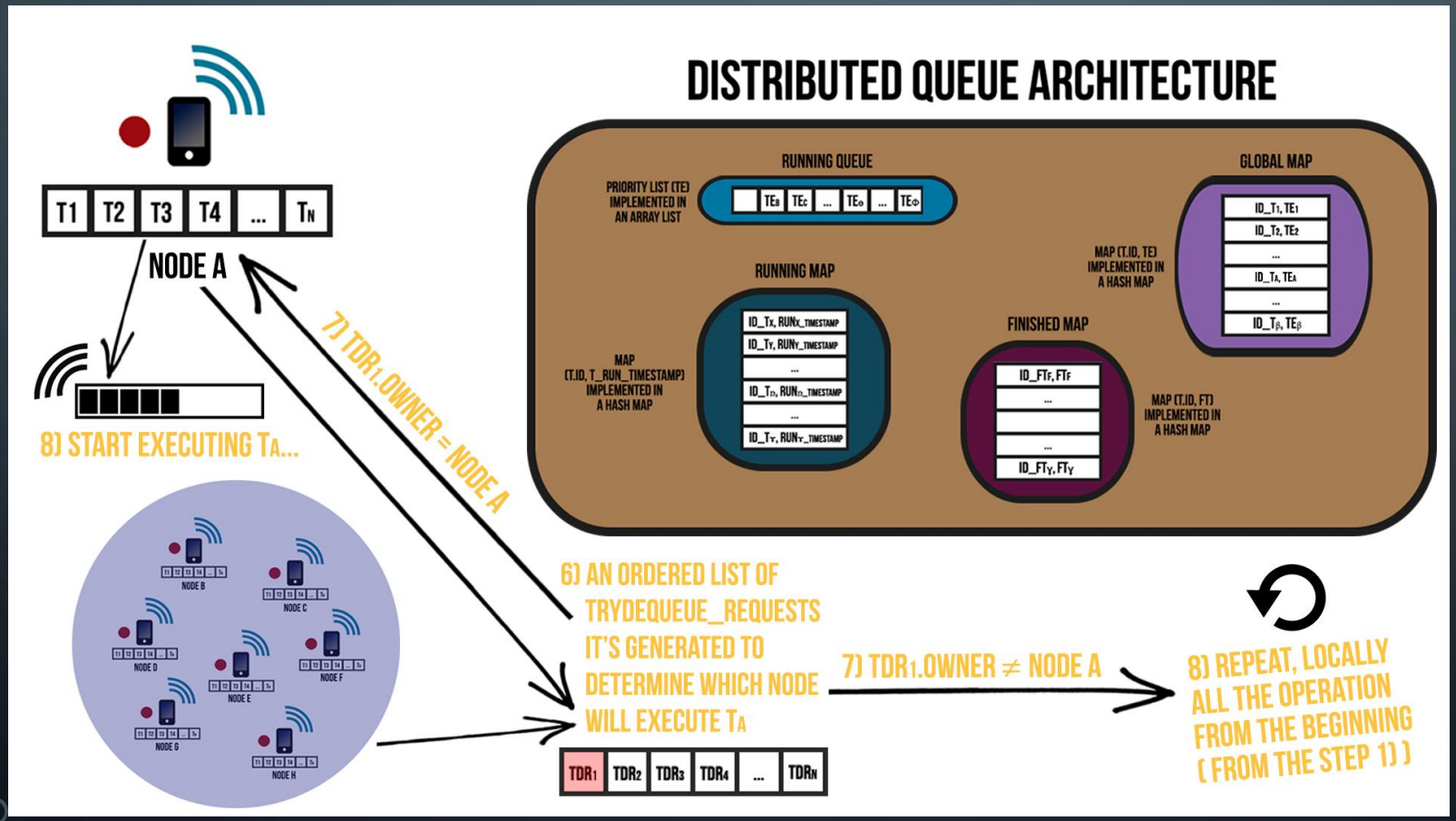


STEPS/OPERATION DESCRIPTION:

6) An ordered list of TryDequeue_Requests it's generated to determine which Node will execute T_A
7) TDR_1.owner = Node A / TDR_1.owner ≠ Node A

TRYDEQUEUE( ) OPERATION (3)

- The TRYDEQUEUE( ) operation of the Distributed Queue System Architecture have the following behavior:

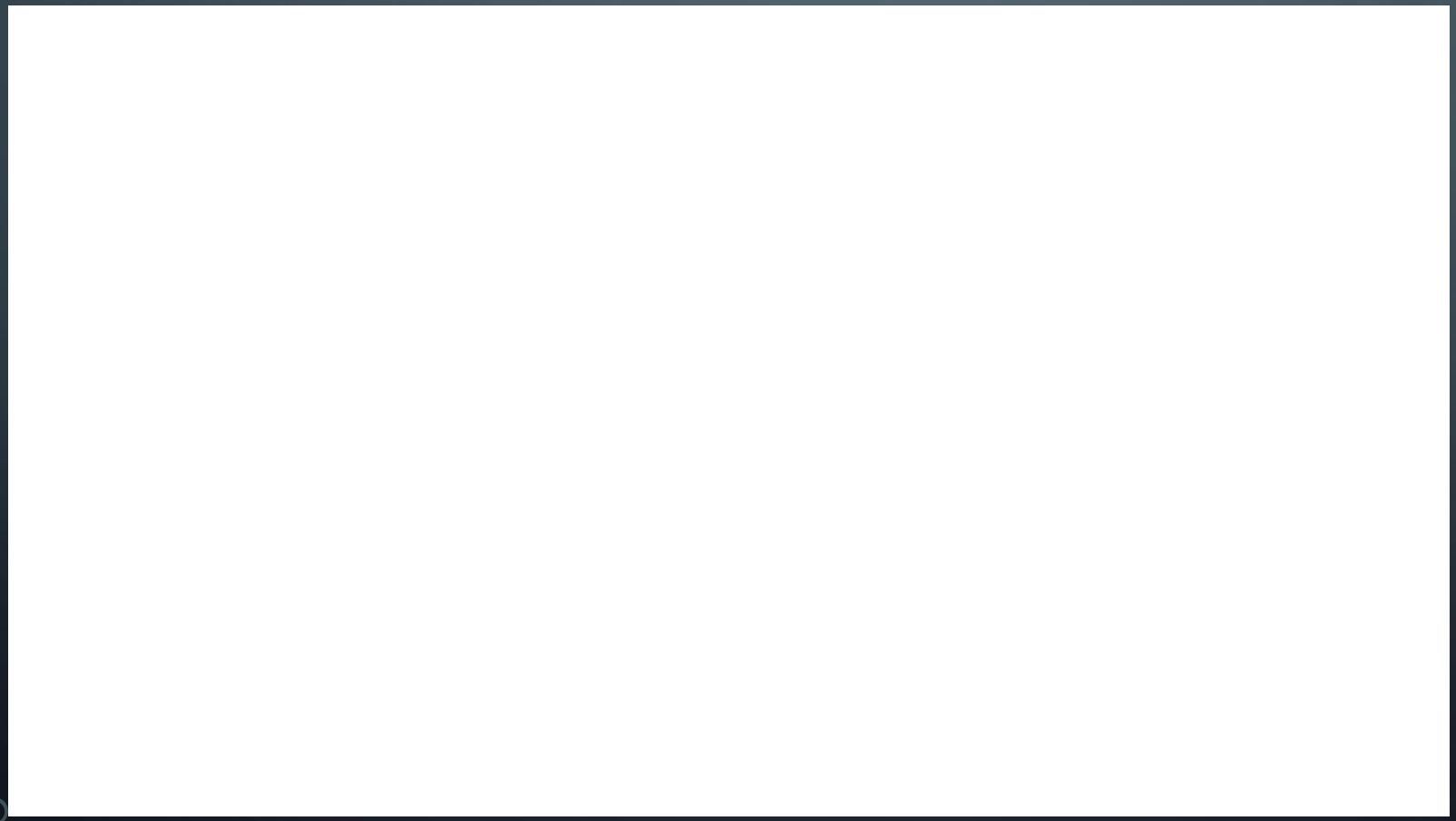


STEPS/OPERATION DESCRIPTION:

6) An ordered list of TryDequeue_Requests it's generated to determine which Node will execute T_A
7) TDR_1.owner = Node A / TDR_1.owner ≠ Node A
8) Start executing T_A... / Repeat, locally, all the operation from the beginning (from the Step 1))

TRYDEQUEUE'S TIME WINDOW (1)

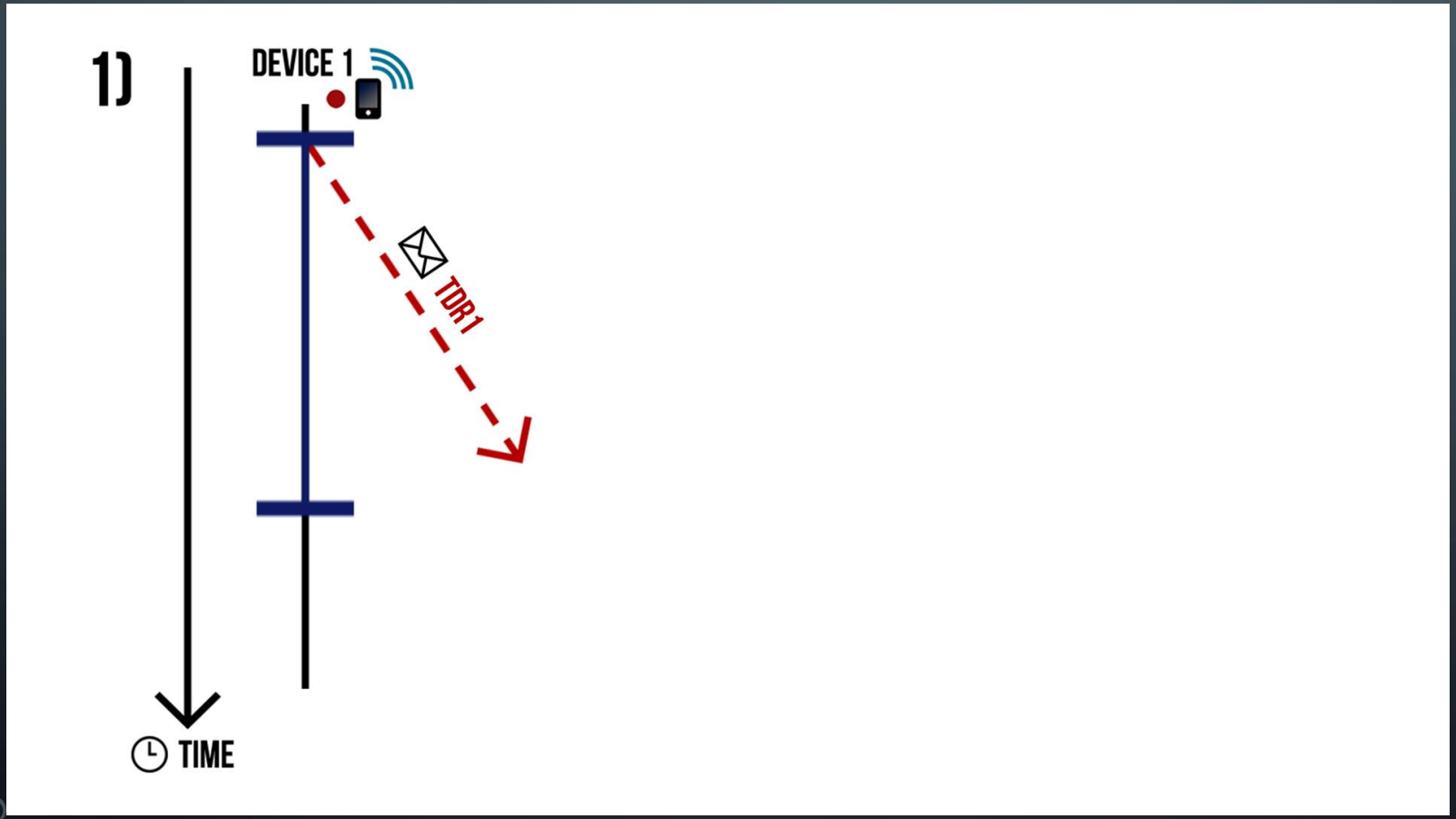
- Situation 1) – TRYDEQUEUE Contest with only 1 Device:



STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE'S TIME WINDOW (1)

- Situation 1) – TRYDEQUEUE Contest with only 1 Device:



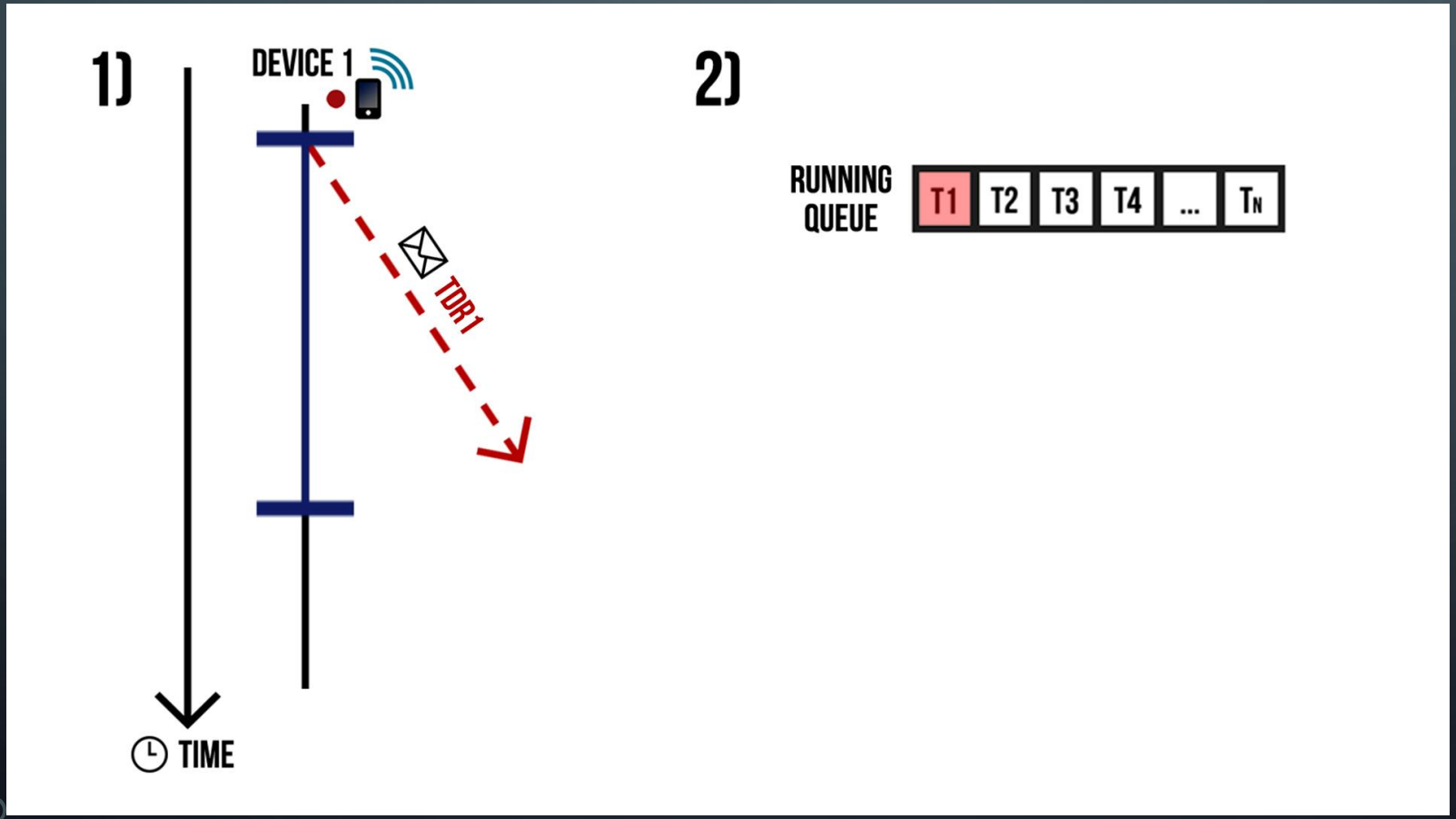
STEPS/OPERATION DESCRIPTION:

1) The Device 1 Broadcast/Hop TRYDEQUEUE message and starts a T time window to receive another TRYDEQUEUE messages

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (1)

- Situation 1) – TRYDEQUEUE Contest with only 1 Device:



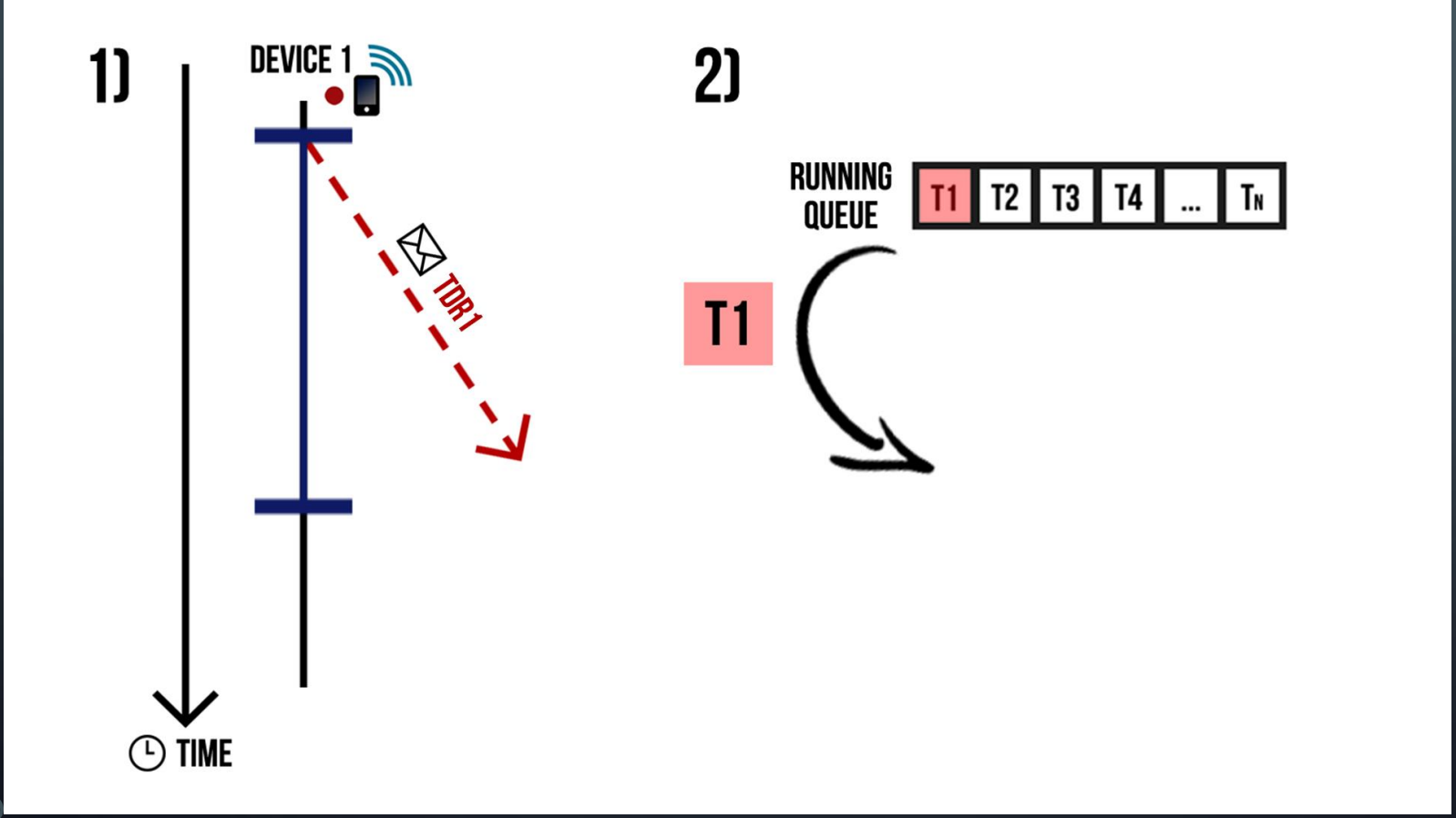
STEPS/OPERATION DESCRIPTION:

1) The Device 1 Broadcast/Hop TRYDEQUEUE message and starts a T time window to receive another TRYDEQUEUE messages
2) As the Device 1 was the only one to try to dequeue T1, it will assume its execution

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (1)

- Situation 1) – TRYDEQUEUE Contest with only 1 Device:



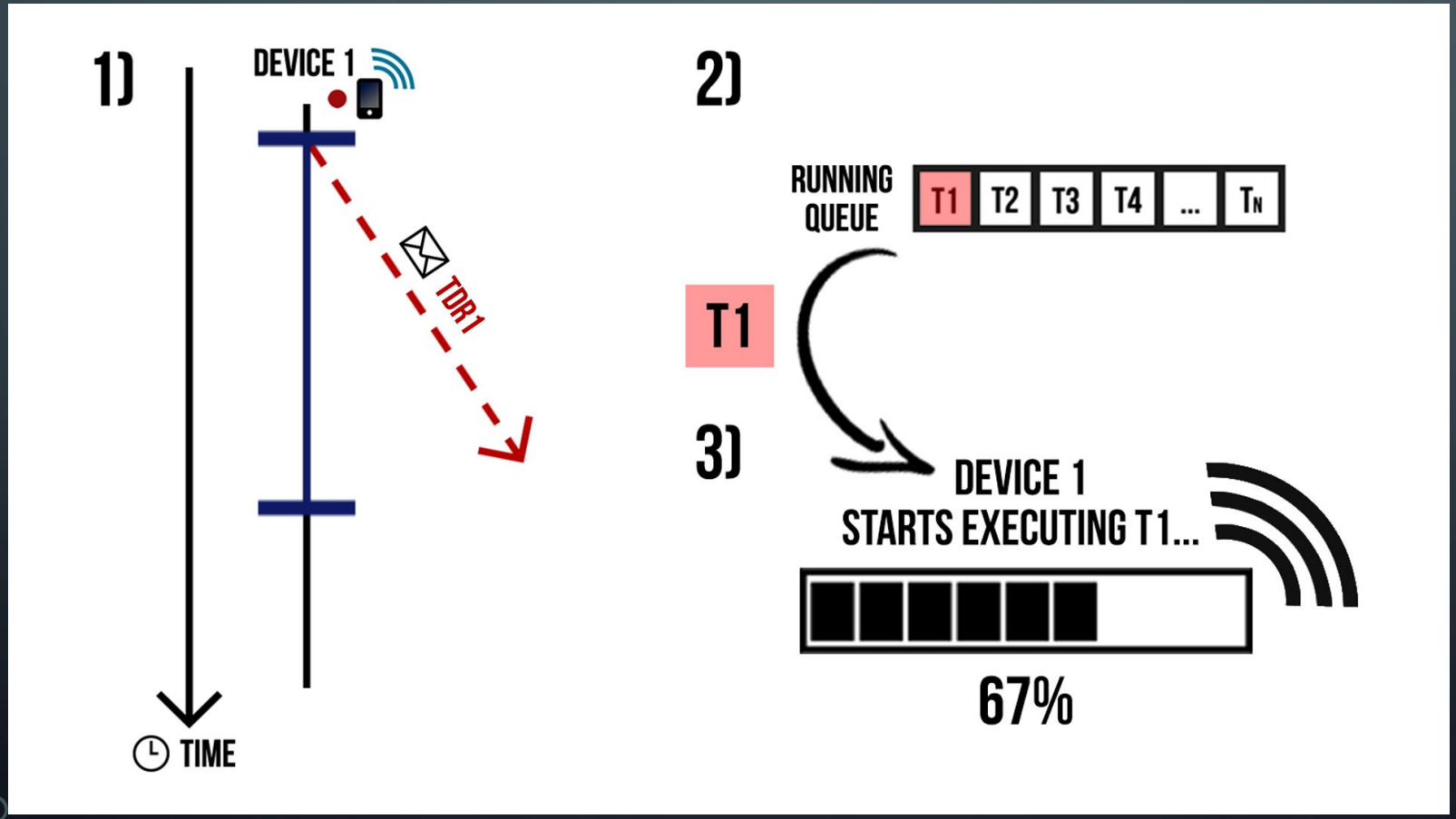
STEPS/OPERATION DESCRIPTION:

1) The Device 1 Broadcast/Hop TRYDEQUEUE message and starts a T time window to receive another TRYDEQUEUE messages
2) As the Device 1 was the only one to try to dequeue T1, it will assume its execution

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (1)

- Situation 1) – TRYDEQUEUE Contest with only 1 Device:



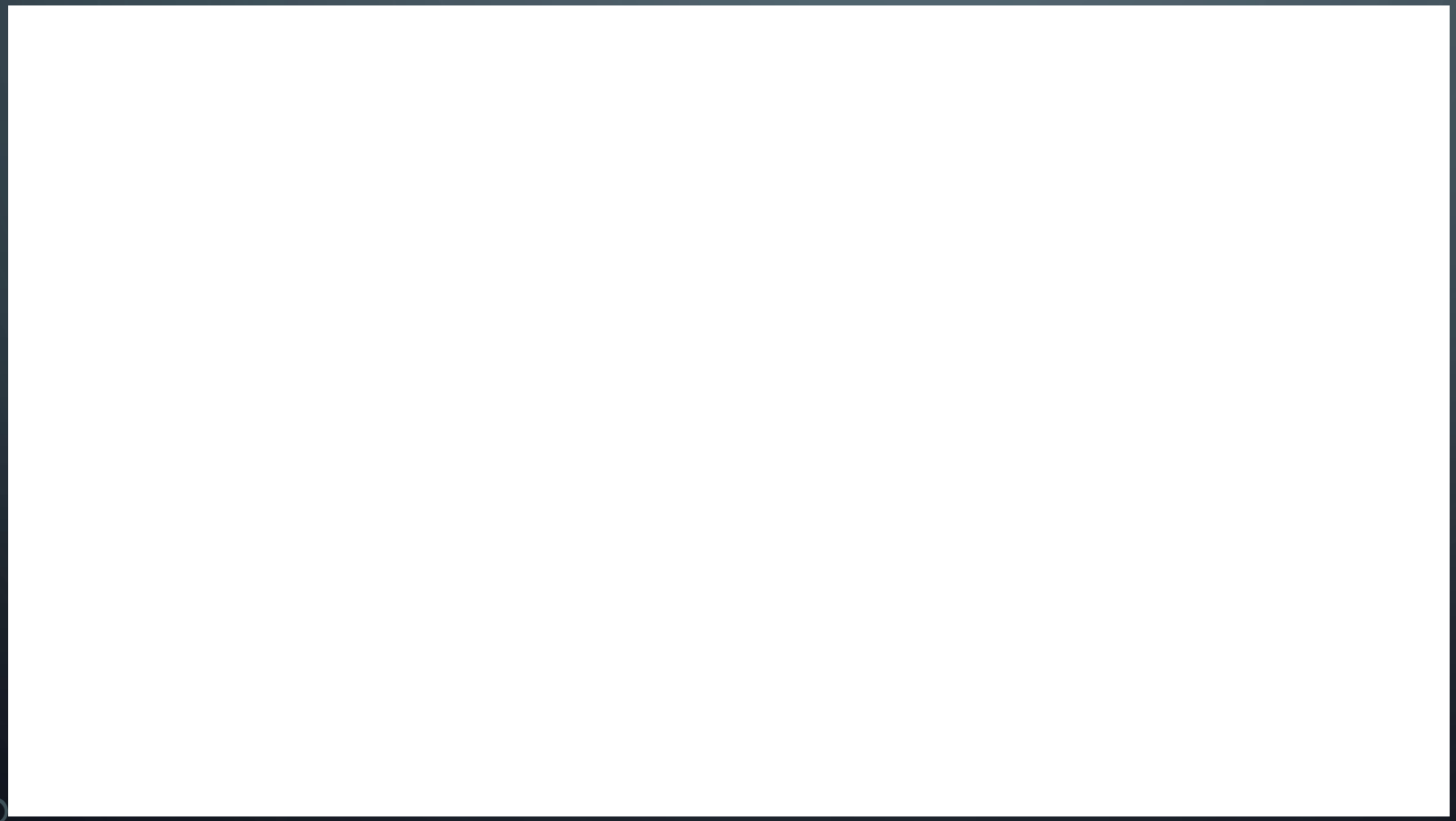
STEPS/OPERATION DESCRIPTION:

1) The Device 1 Broadcast/Hop TRYDEQUEUE message and starts a T time window to receive another TRYDEQUEUE messages
2) As the Device 1 was the only one to try to dequeue T1, it will assume its execution
3) The Device 1 starts executing T1 and periodically, Broadcast/Hop RUNNING messages

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (2)

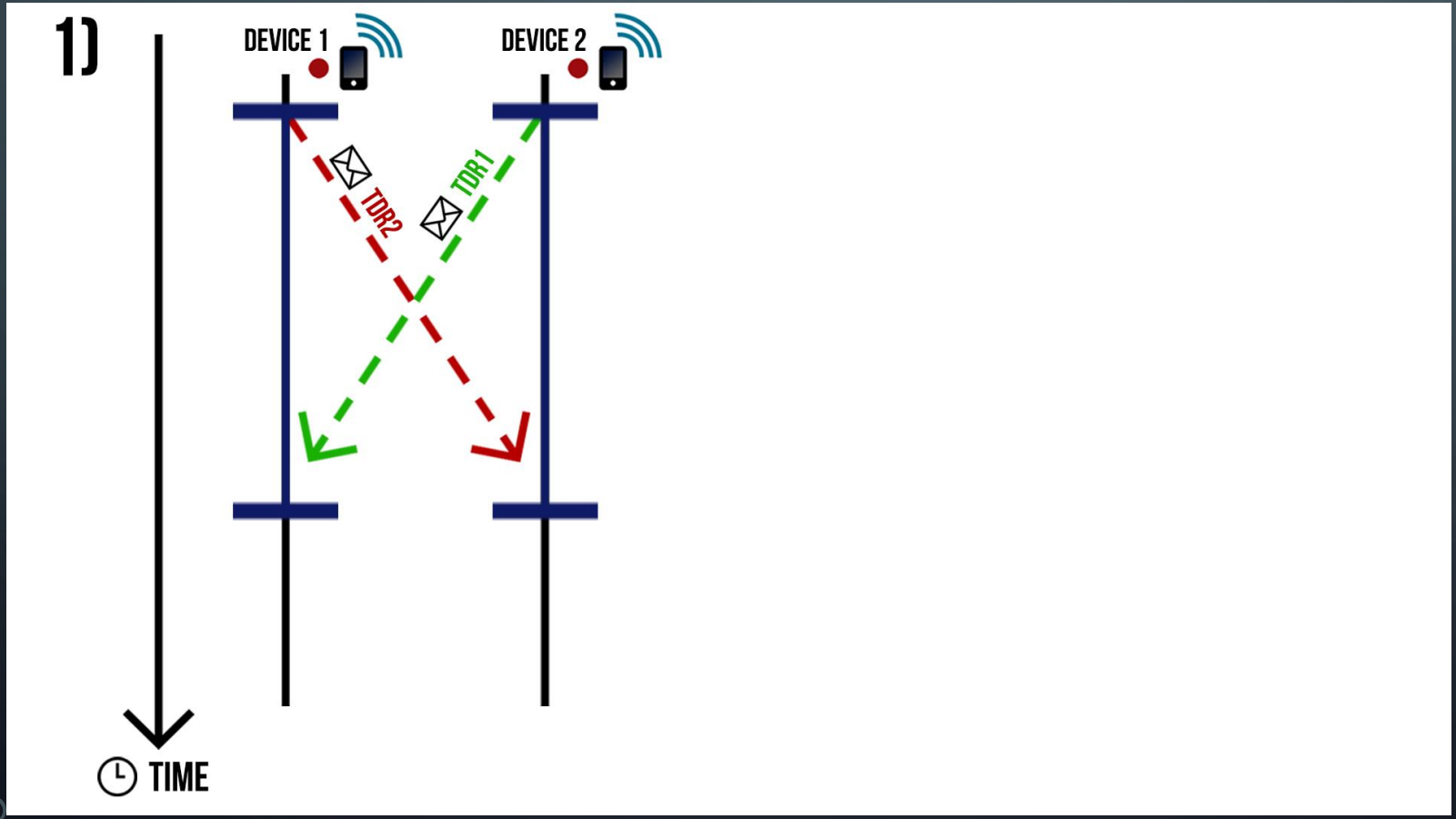
- Situation 2) – TRYDEQUEUE Contest with 2 Devices:



STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE'S TIME WINDOW (2)

- Situation 2) – TRYDEQUEUE Contest with 2 Devices:



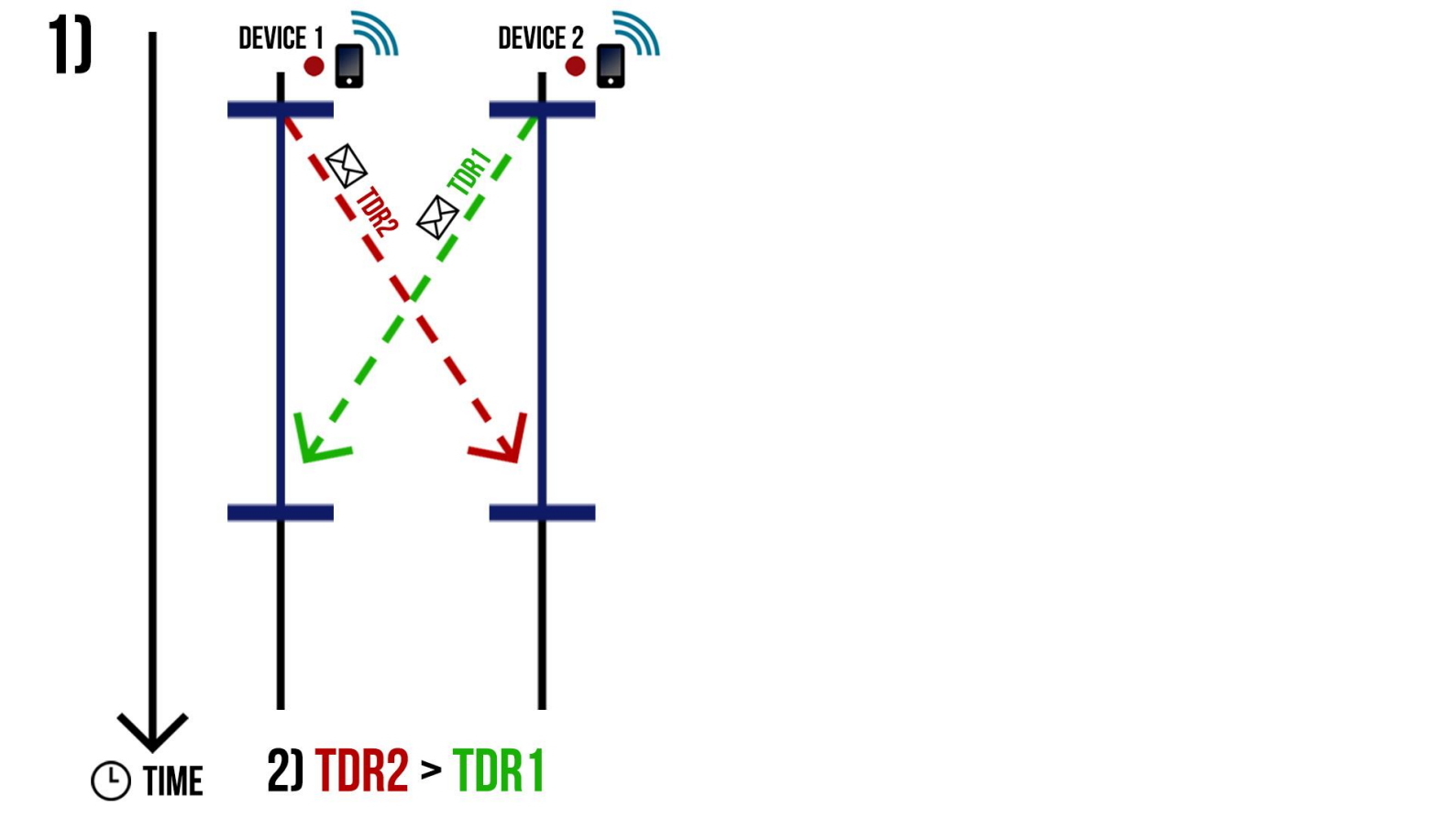
STEPS/OPERATION DESCRIPTION:

1) The Devices 1 and 2 Broadcast/Hop TRYDEQUEUE messages and start a T time window to receive another TRYDEQUEUE messages

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (2)

- Situation 2) – TRYDEQUEUE Contest with 2 Devices:



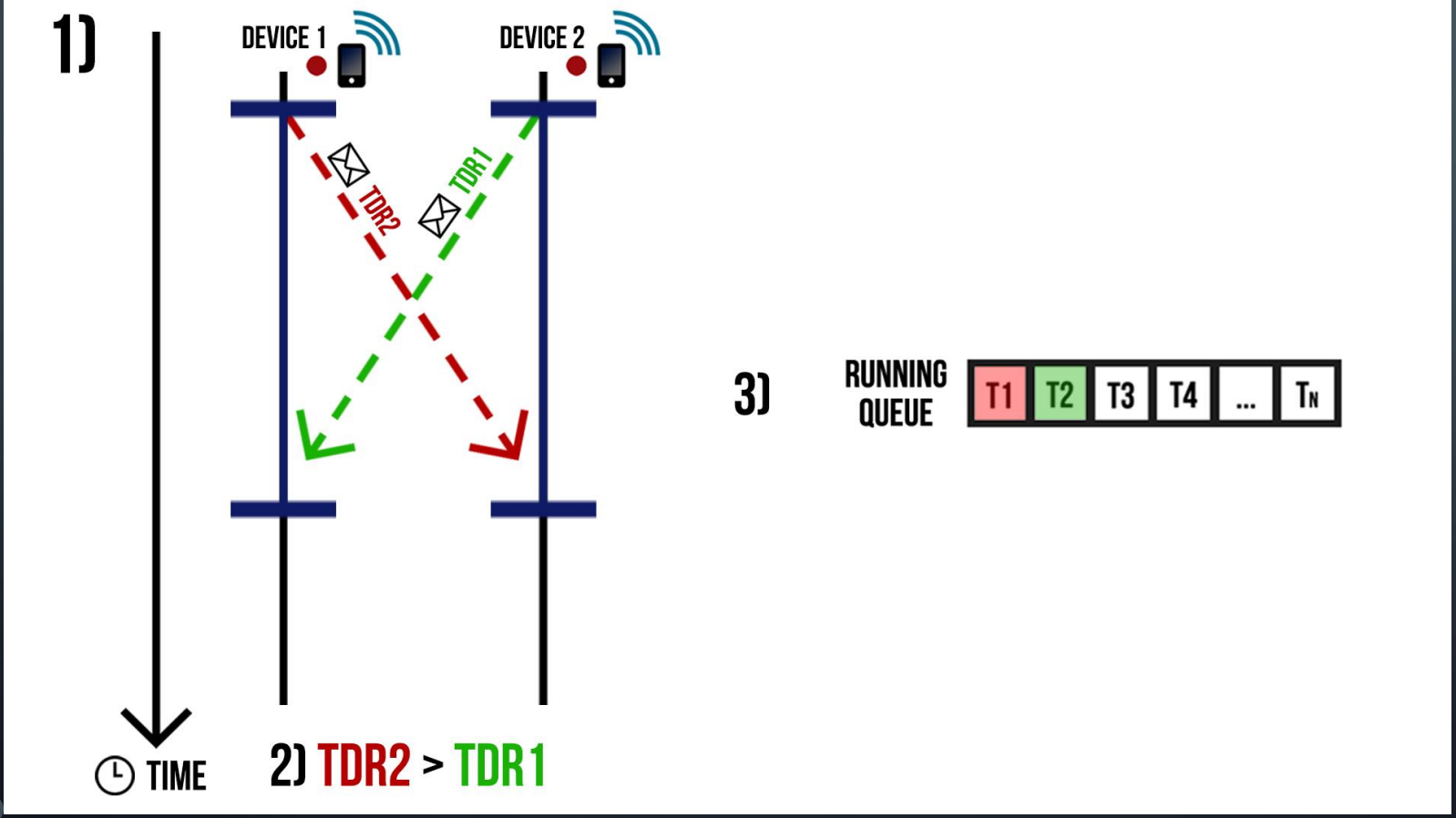
STEPS/OPERATION DESCRIPTION:

1) The Devices 1 and 2 Broadcast/Hop TRYDEQUEUE messages and start a T time window to receive another TRYDEQUEUE messages
2) Sort of the all TryDequeue Requests for the Element/Task received during the T time window (TDR2 > TDR1)

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (2)

- Situation 2) – TRYDEQUEUE Contest with 2 Devices:



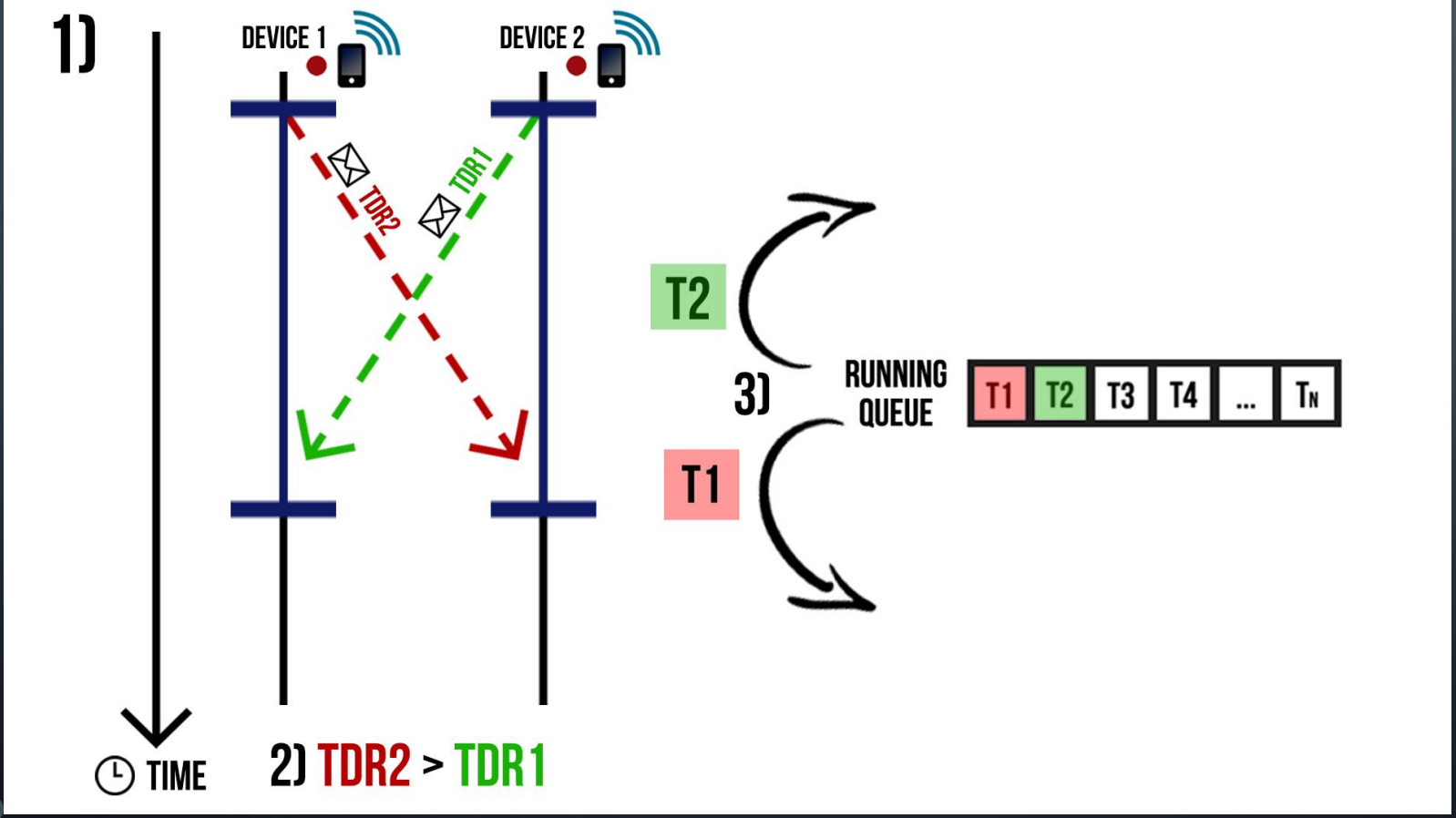
STEPS/OPERATION DESCRIPTION:

1) The Devices 1 and 2 Broadcast/Hop TRYDEQUEUE messages and start a T time window to receive another TRYDEQUEUE messages
2) Sort of the all TryDequeue Requests for the Element/Task received during the T time window (TDR2 > TDR1)
3) The Device 1 win the TryDequeue Requests' Contest, so will dequeue the Element/Task in the 1st position of the Running Queue and the Device 2 (2nd classified) will dequeue the Element/Task in the 2nd position

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (2)

- Situation 2) – TRYDEQUEUE Contest with 2 Devices:



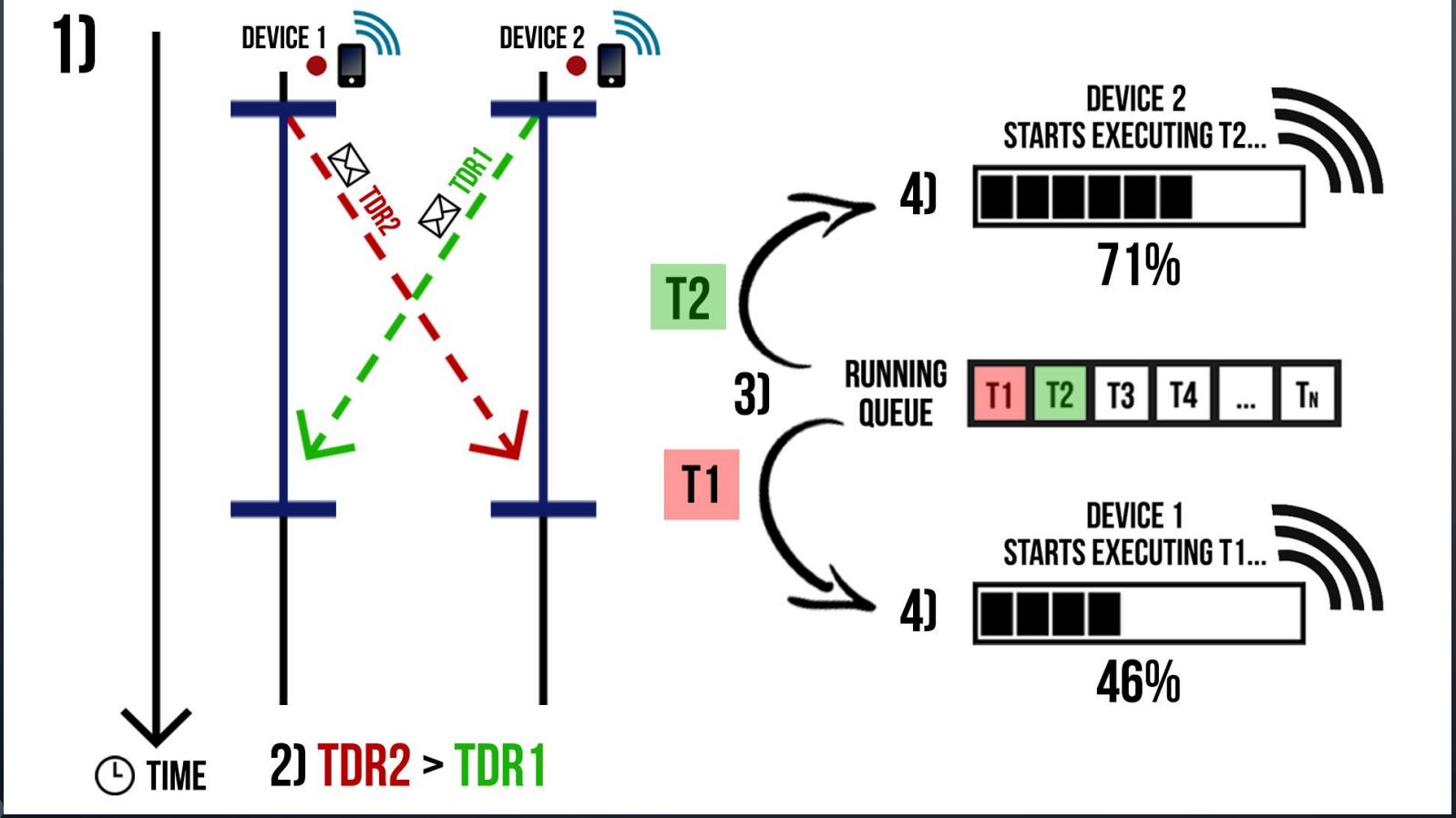
STEPS/OPERATION DESCRIPTION:

1) The Devices 1 and 2 Broadcast/Hop TRYDEQUEUE messages and start a T time window to receive another TRYDEQUEUE messages
2) Sort of the all TryDequeue Requests for the Element/Task received during the T time window (TDR2 > TDR1)
3) The Device 1 win the TryDequeue Requests' Contest, so will dequeue the Element/Task in the 1st position of the Running Queue and the Device 2 (2nd classified) will dequeue the Element/Task in the 2nd position

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (2)

- Situation 2) – TRYDEQUEUE Contest with 2 Devices:



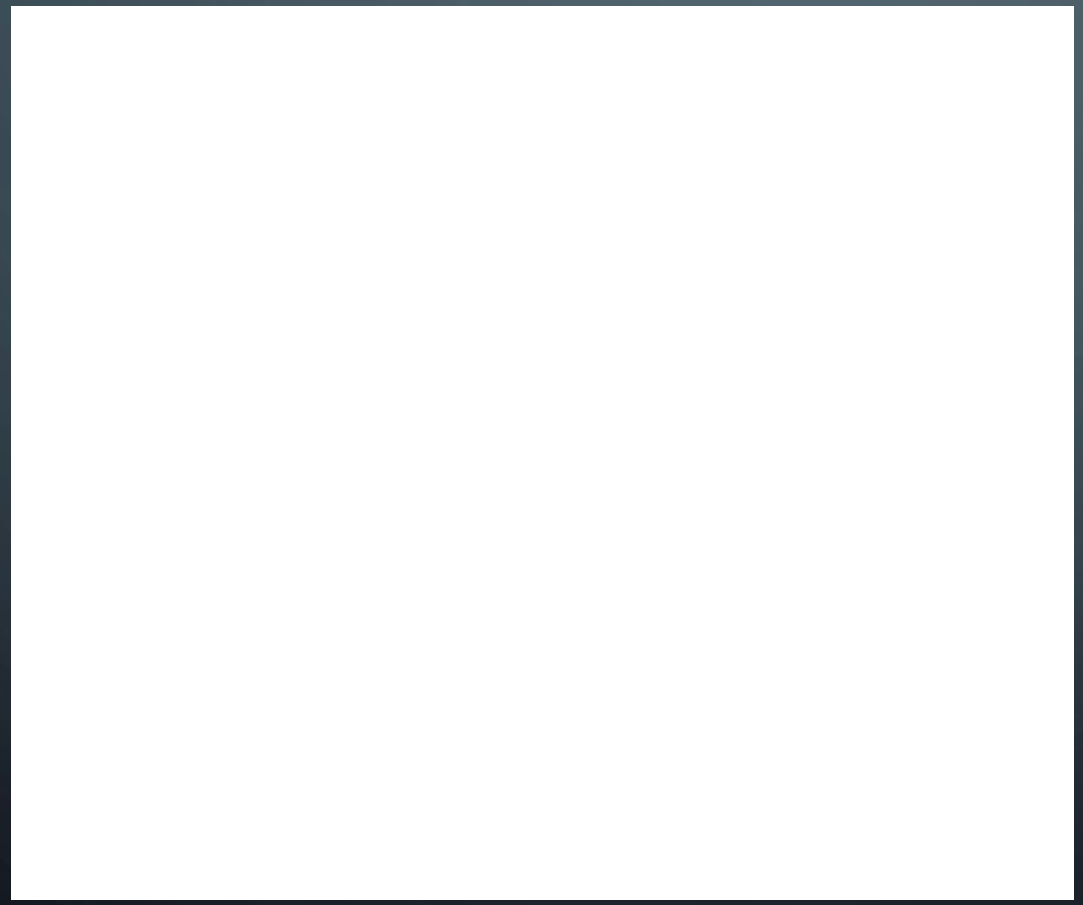
(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

STEPS/OPERATION DESCRIPTION:

1) The Devices 1 and 2 Broadcast/Hop TRYDEQUEUE messages and start a T time window to receive another TRYDEQUEUE messages
2) Sort of the all TryDequeue Requests for the Element/Task received during the T time window (TDR2 > TDR1)
3) The Device 1 win the TryDequeue Requests' Contest, so will dequeue the Element/Task in the 1st position of the Running Queue and the Device 2 (2nd classified) will dequeue the Element/Task in the 2nd position
4) Device 1 starts executing T1 and Device 2 starts executing T2. Both Devices periodically Broadcast/Hop RUNNING messages

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

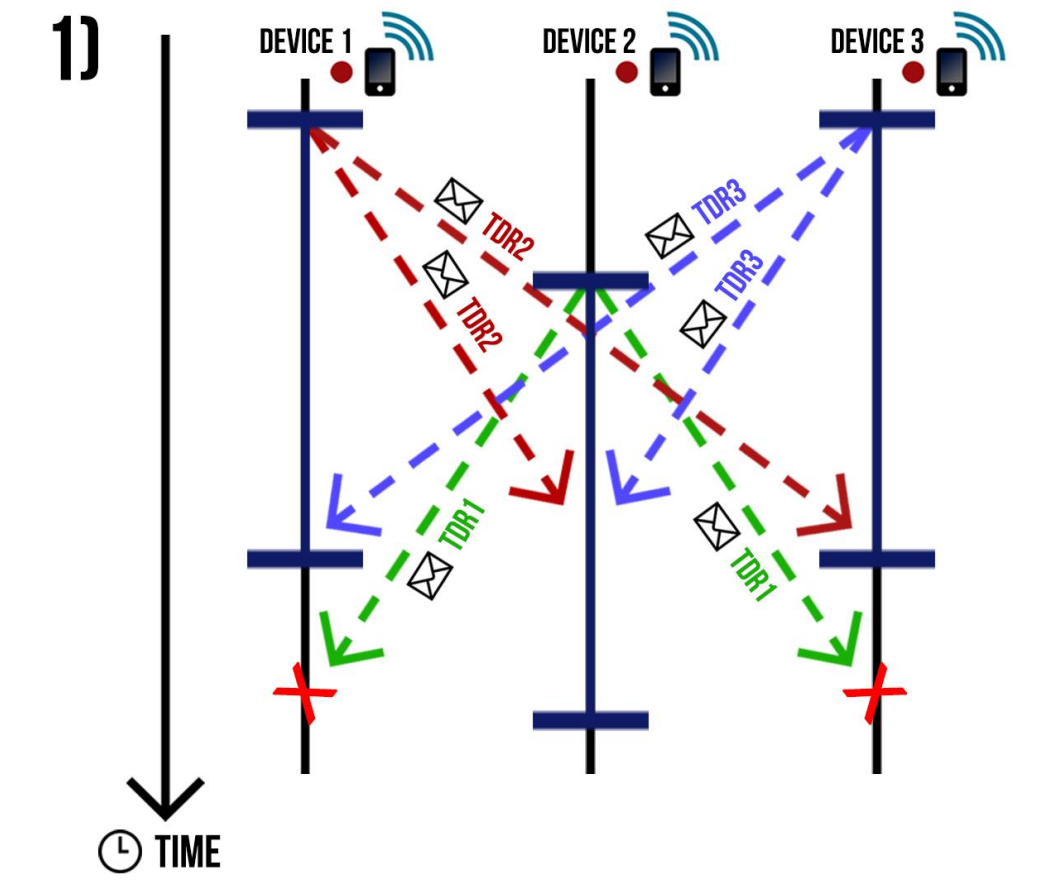


STEPS/OPERATION DESCRIPTION:

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:



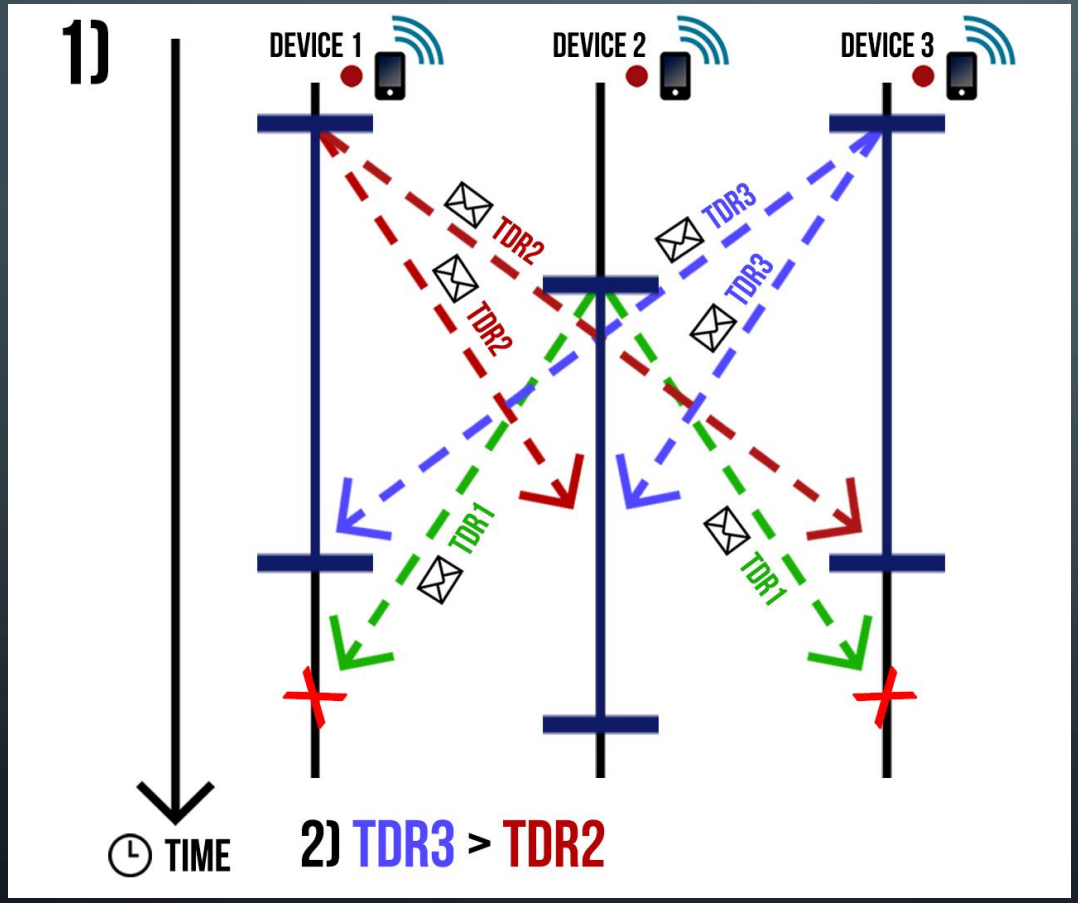
STEPS/OPERATION DESCRIPTION:

1) The Devices 1, 2 and 3 Broadcast/Hop TRYDEQUEUE messages and start a T time window to receive another TRYDEQUEUE messages. But the Broadcast/Hop TRYDEQUEUE messages sent by the Device 2 was received outside of the T time window started by the Devices 1 and 3. So, in this step, it's only considered the TryDequeue Requests for the Element/Task made by the Devices 1 and 3

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:



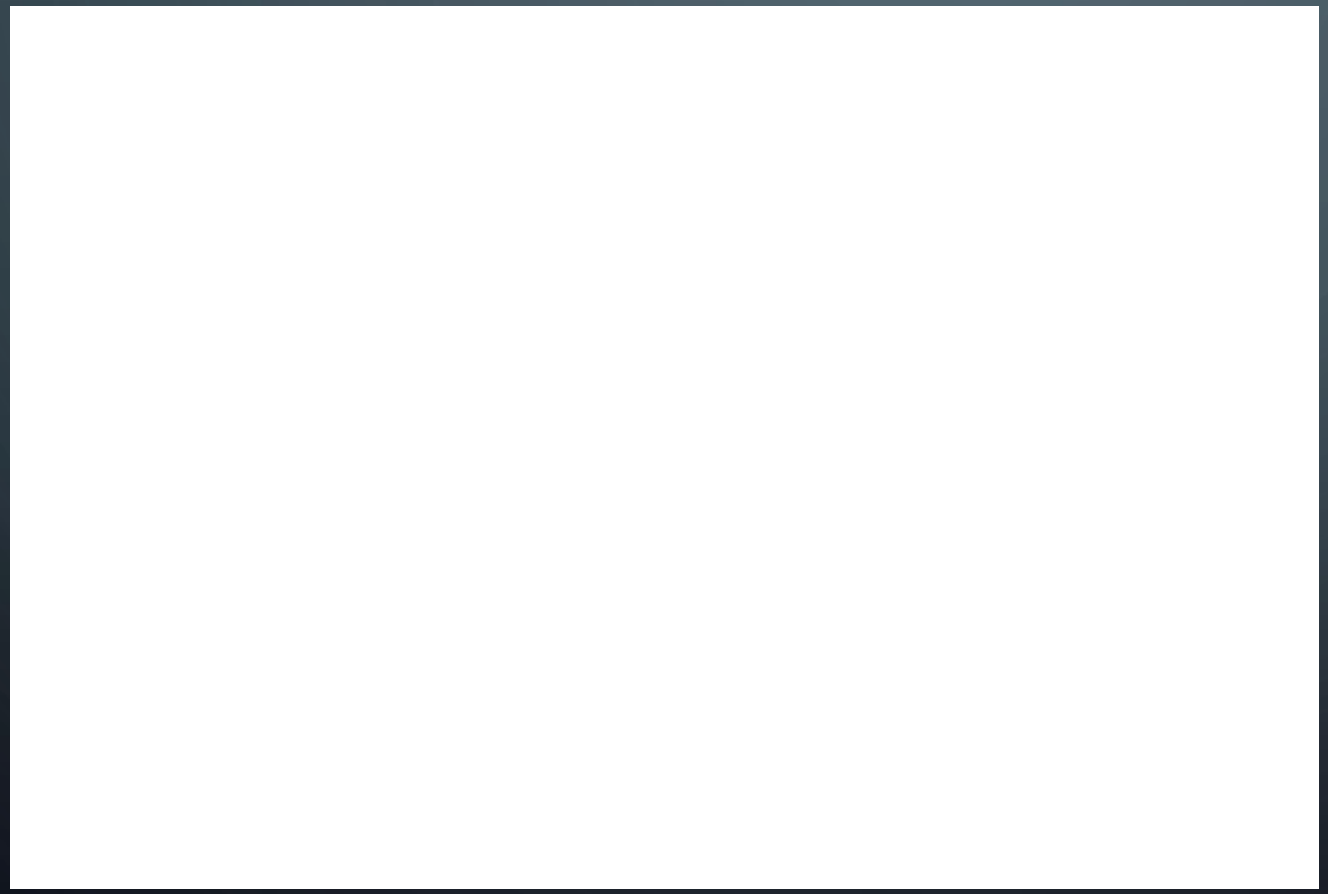
STEPS/OPERATION DESCRIPTION:

1) The Devices 1, 2 and 3 Broadcast/Hop TRYDEQUEUE messages and start a T time window to receive another TRYDEQUEUE messages. But the Broadcast/Hop TRYDEQUEUE messages sent by the Device 2 was received outside of the T time window started by the Devices 1 and 3. So, in this step, it's only considered the TryDequeue Requests for the Element/Task made by the Devices 1 and 3
2) Sort of the all TryDequeue Requests for the Element/Task received during the T time window (TDR3 > TDR2)

(*) – T varies accordingly to the time that Broadcast/Hop message takes to propagate in the network/range

TRYDEQUEUE'S TIME WINDOW (3)

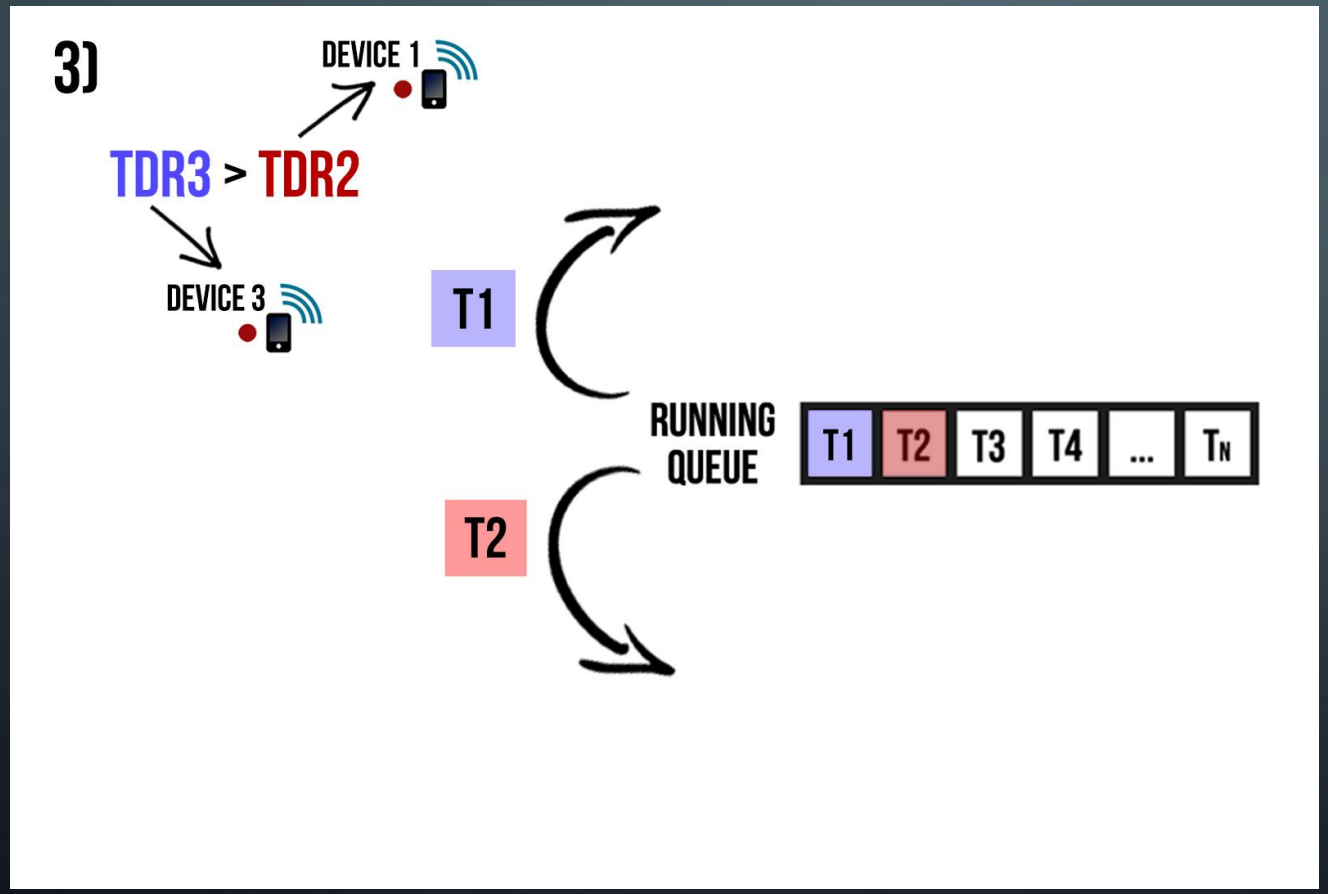
- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:



STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

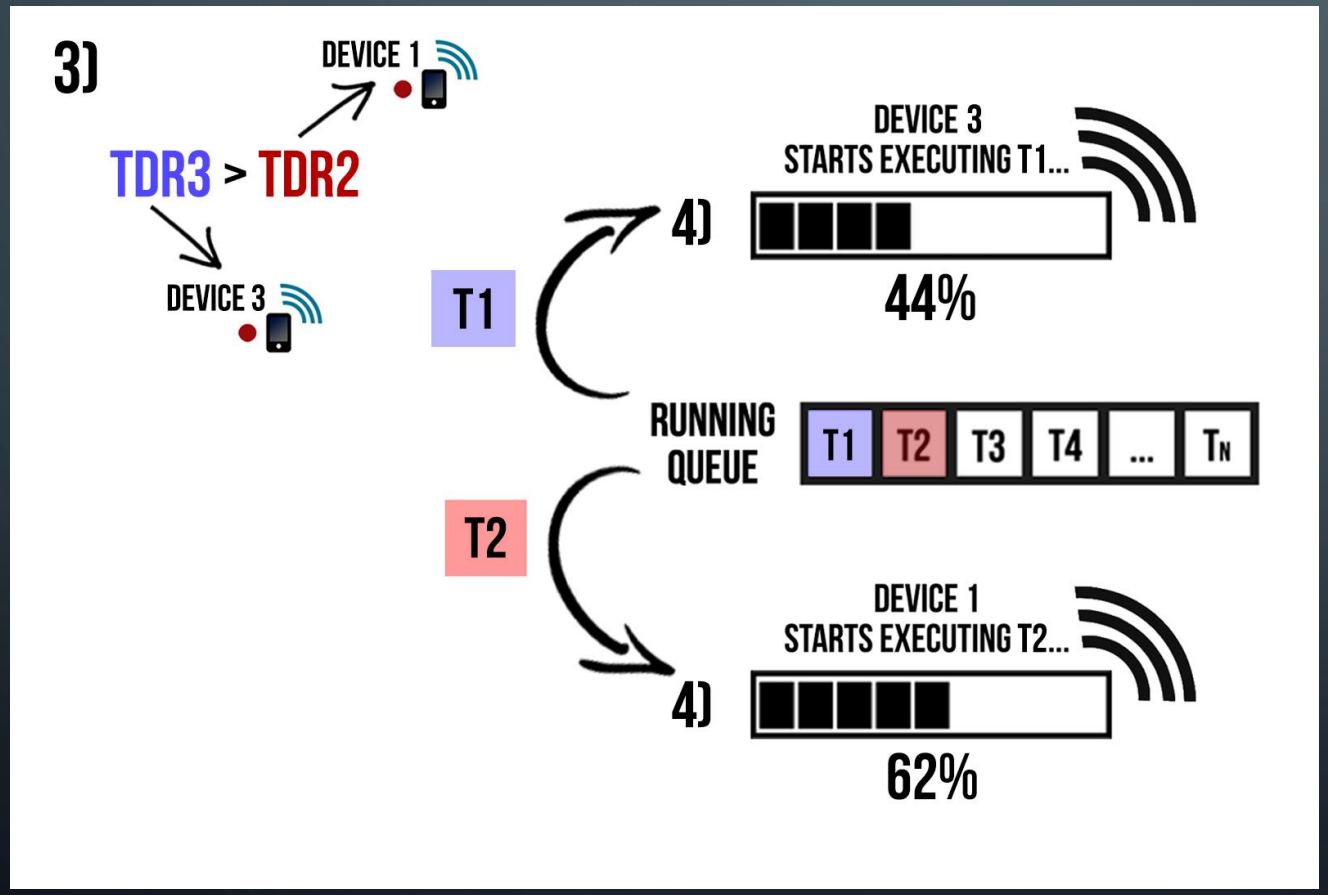


STEPS/OPERATION DESCRIPTION:

3) The Device 3 win the TryDequeue Requests' Contest, so will dequeue the Element/Task in the 1st position of the Running Queue and the Device 1 (2nd classified) will dequeue the Element/Task in the 2nd position

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

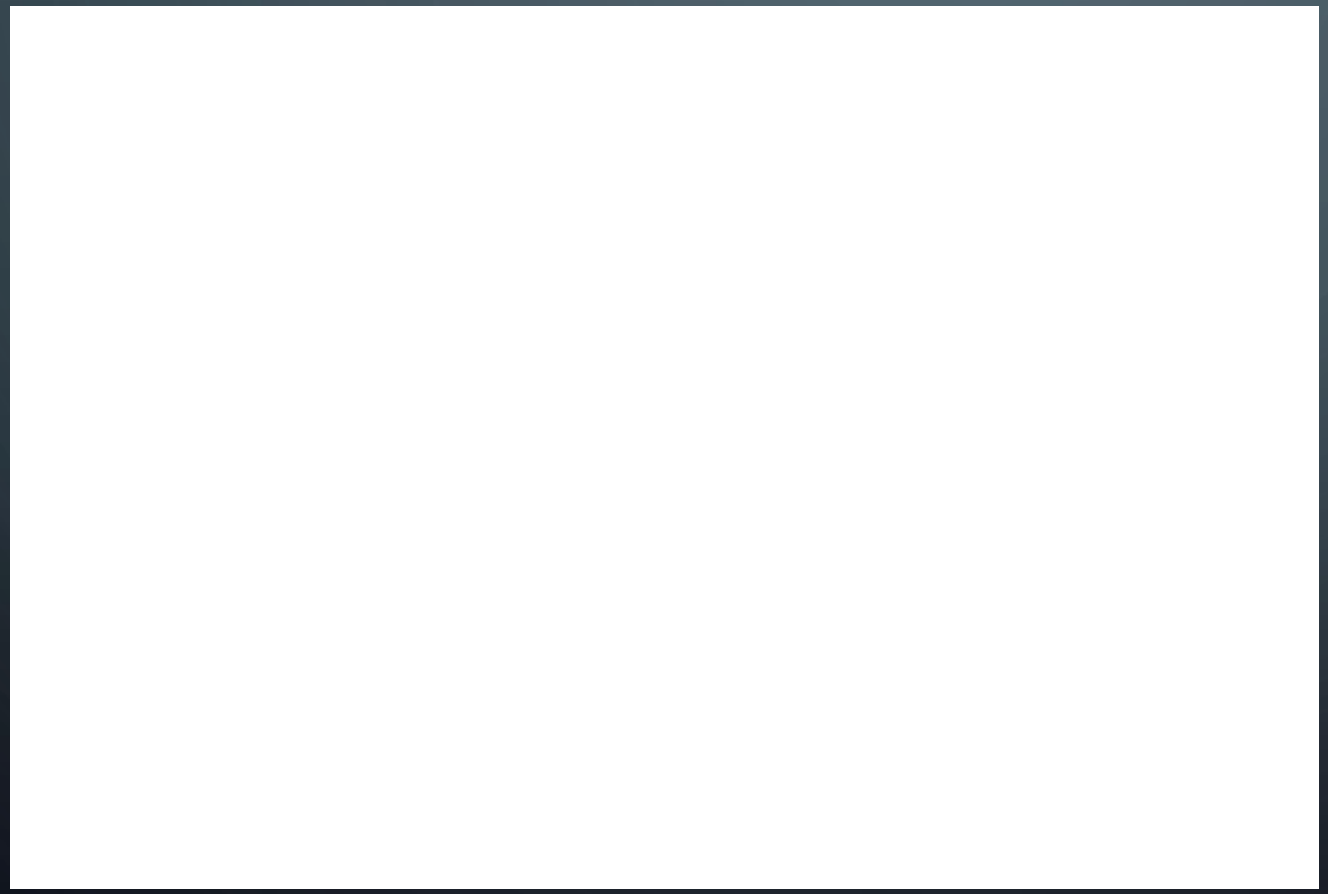


STEPS/OPERATION DESCRIPTION:

3) The Device 3 win the TryDequeue Requests' Contest, so will dequeue the Element/Task in the 1st position of the Running Queue and the Device 1 (2nd classified) will dequeue the Element/Task in the 2nd position
4) Device 3 starts executing T1 and Device 1 starts executing T2. Both Devices periodically Broadcast/Hop RUNNING messages

TRYDEQUEUE'S TIME WINDOW (3)

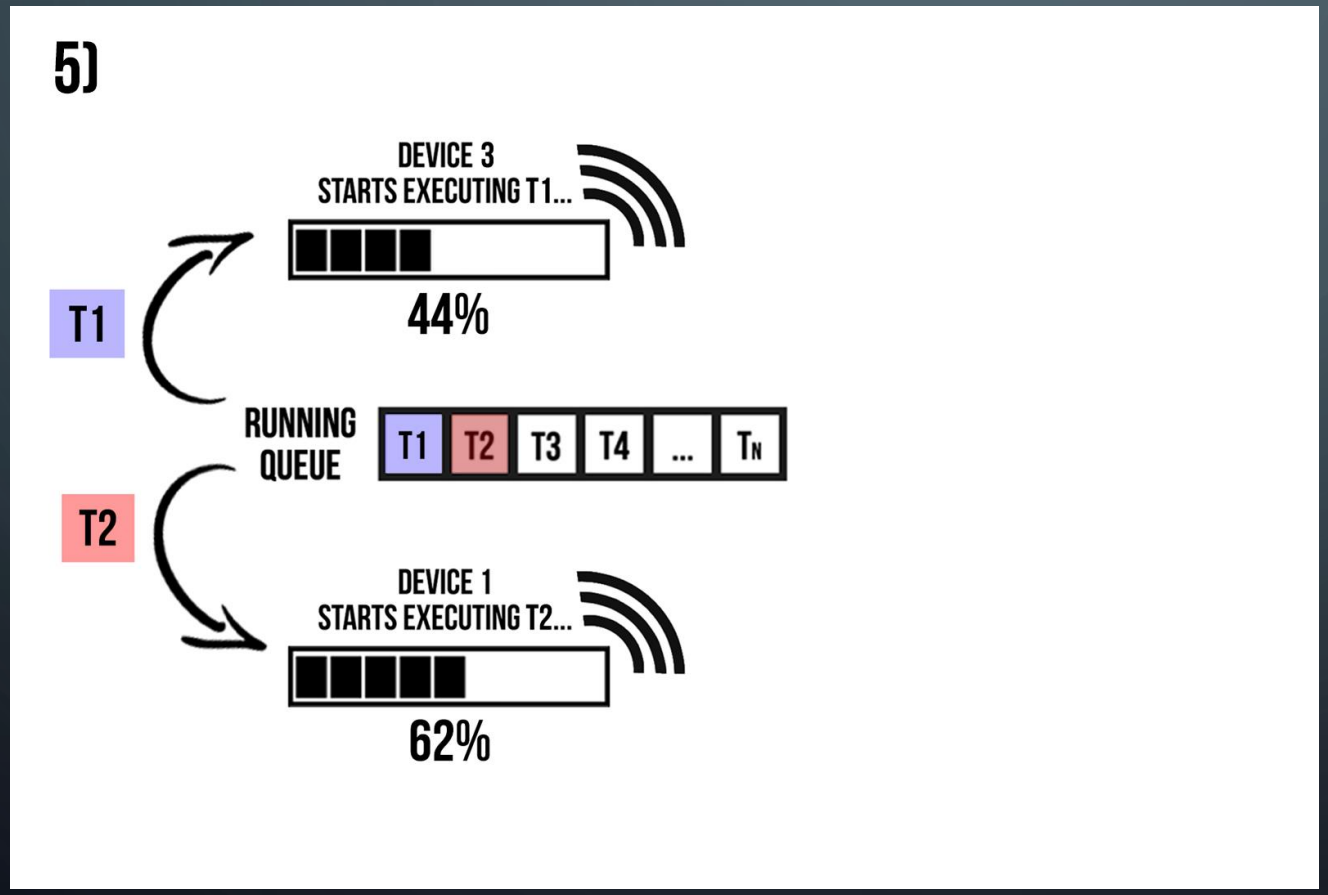
- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:



STEPS/OPERATION DESCRIPTION:

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

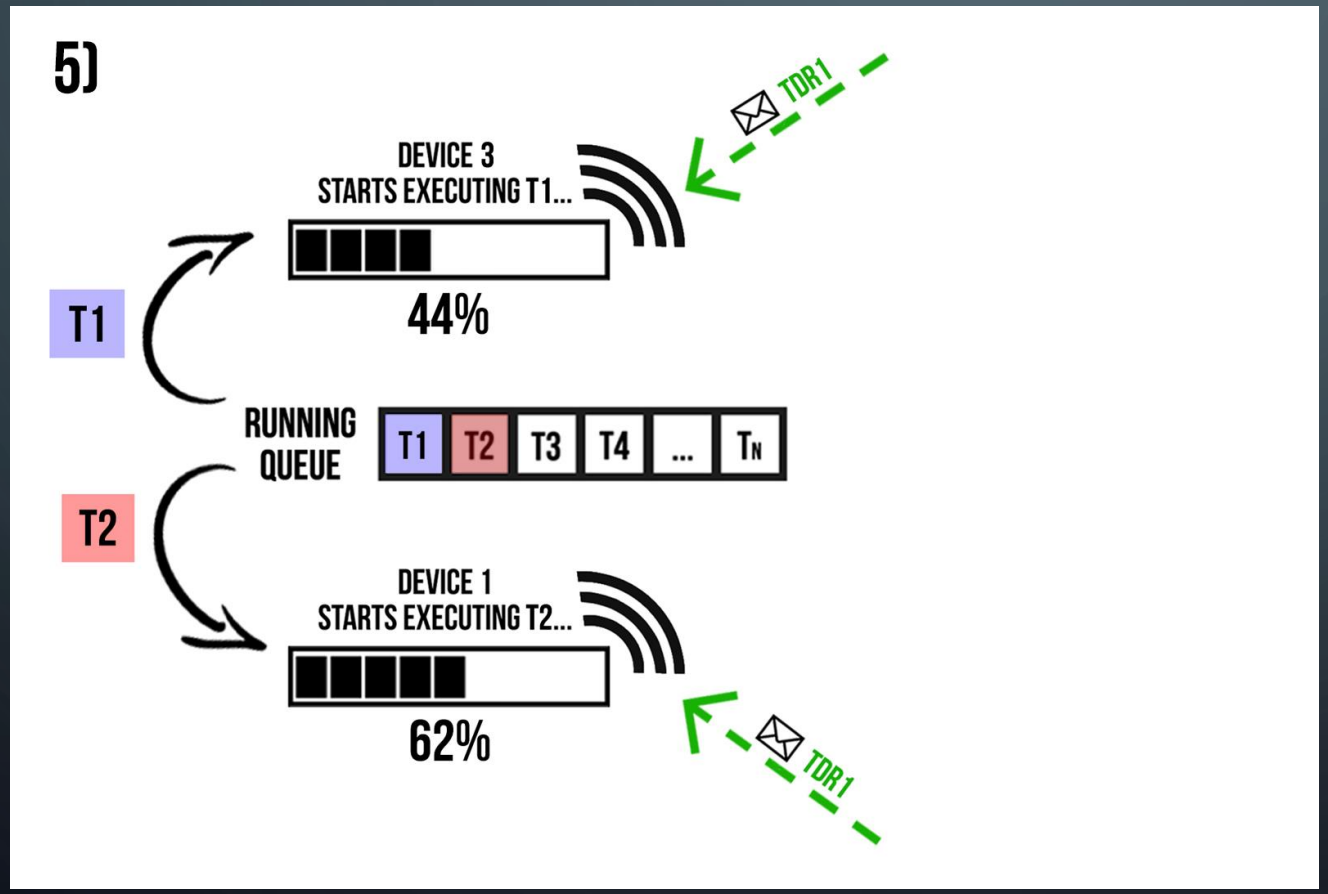


STEPS/OPERATION DESCRIPTION:

5) During the execution of the Element/Task T1 and T2 by the Devices 3 and 1, respectively, was received the TryDequeue Request made by the Device 2 for the Element/Task T1

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

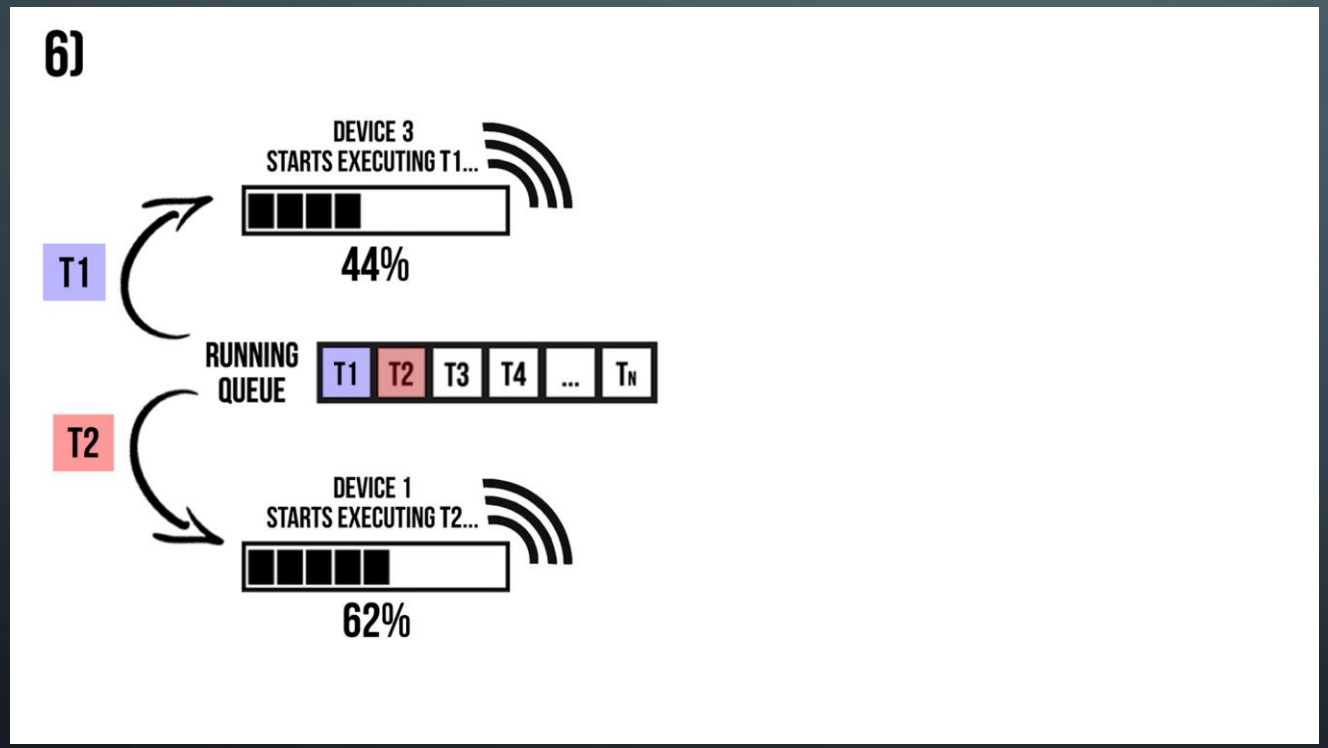


STEPS/OPERATION DESCRIPTION:

5) During the execution of the Element/Task T1 and T2 by the Devices 3 and 1, respectively, was received the TryDequeue Request made by the Device 2 for the Element/Task T1

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:



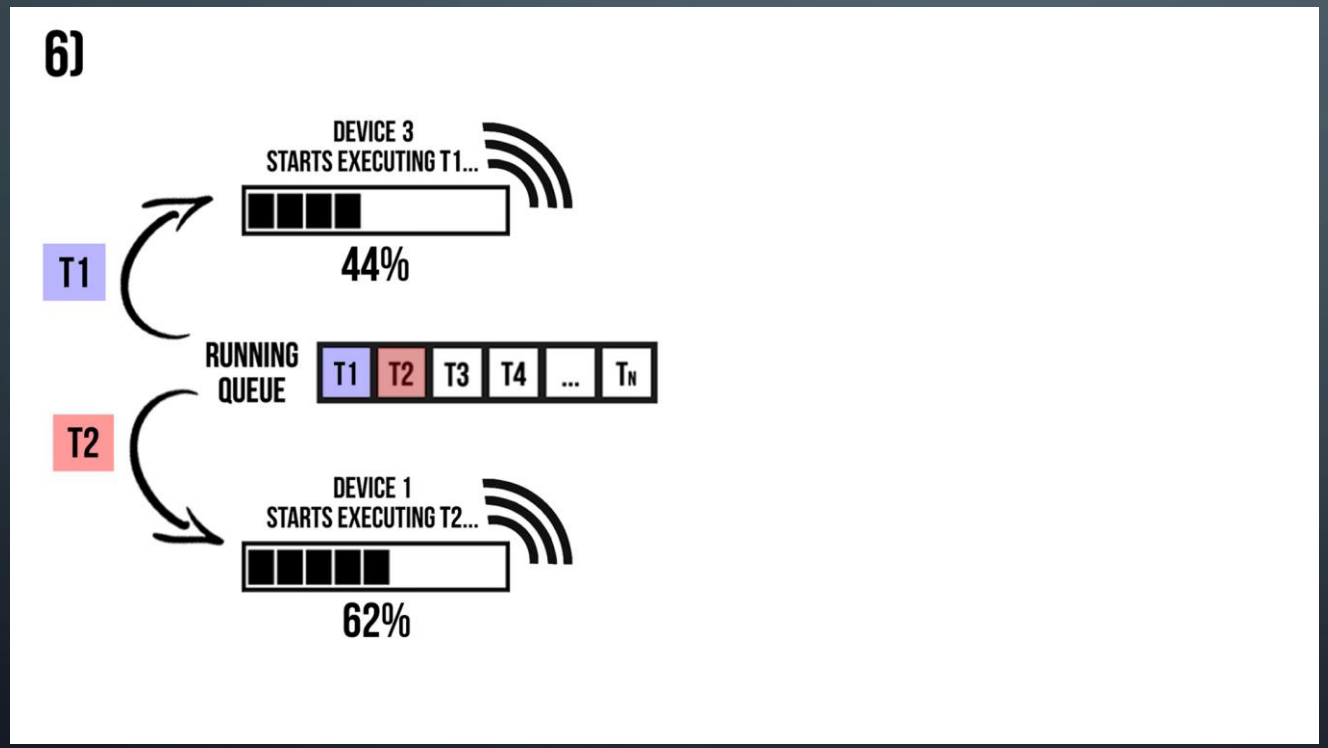
STEPS/OPERATION DESCRIPTION:

6) Occurs a conflict for the Execution Host of the Element/Task T1 (and the following Elements/Tasks, if it's necessary). In the beginning, it's realized an Execution Host Contest between the Devices 2 and 3 for the execution of the Element/Task T1, if the Device 2 don't win the Execution Host Contest for the Element/Task, will start another Execution Host Contest for the Element/Task T2 with its Execution's Host, and will repeat the same process for the following Elements/Tasks, until wins a Execution Host Contest

TRYDEQUEUE'S TIME WINDOW (3)

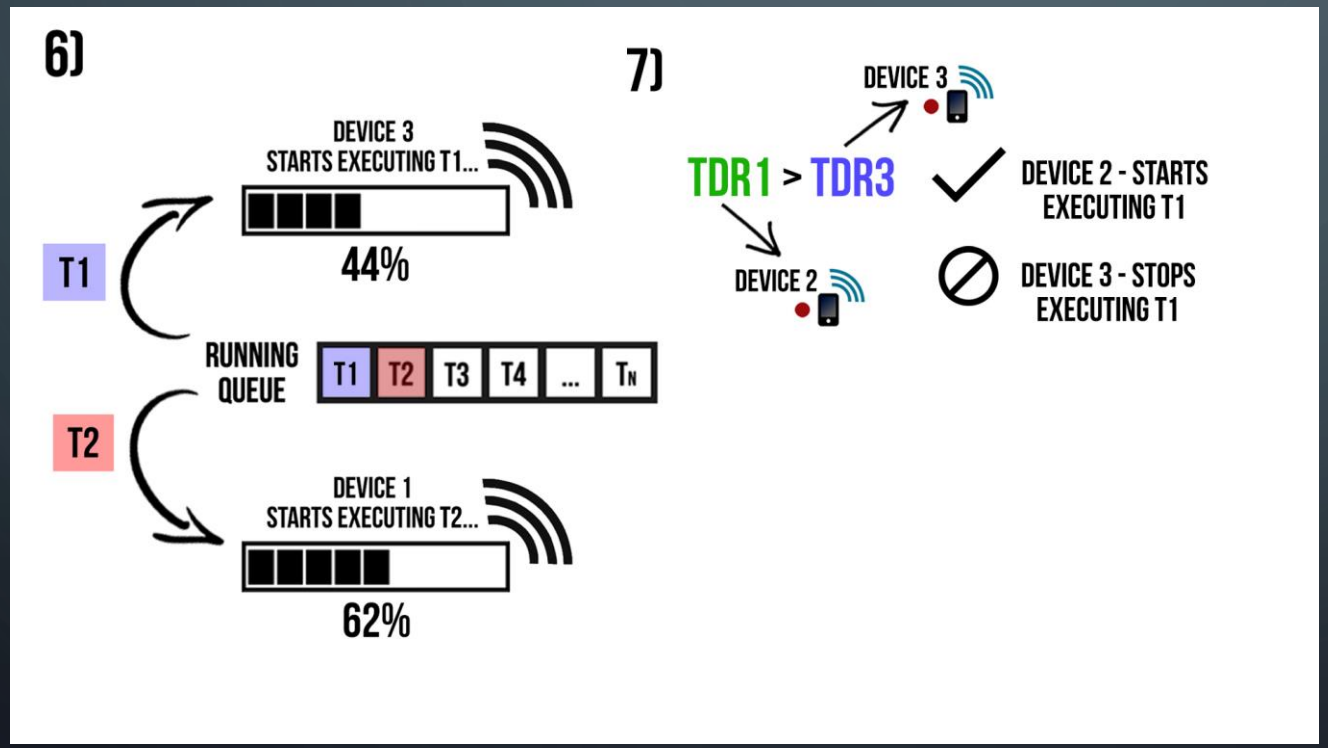
- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

STEPS/OPERATION DESCRIPTION:



TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

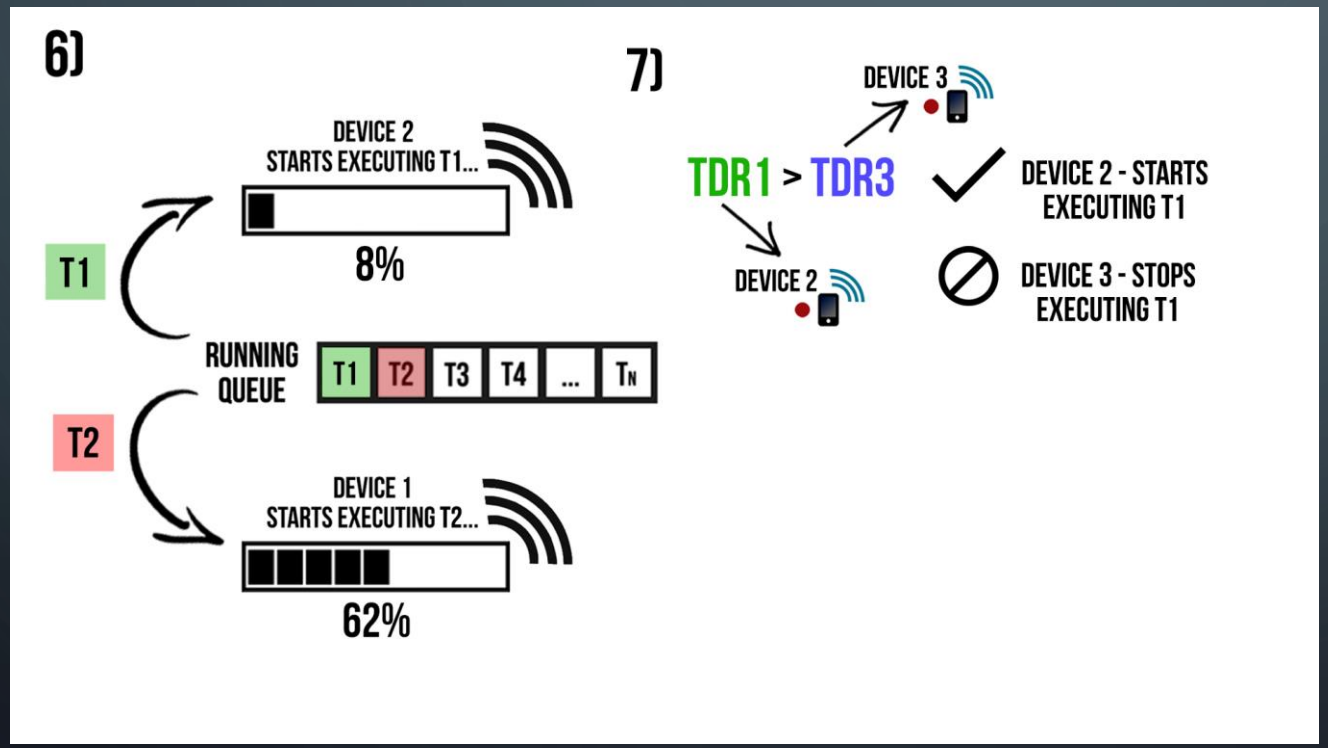


STEPS/OPERATION DESCRIPTION:

7) Situation 1 (TDR1 > TDR3):
In this situation, the winner of the Host Contest it's the Device 2. So the Execution's Host of T1 changes. The new Execution's Host of T1 will be the Device 2. The Device 3 stops the execution of T1 and Device 2 starts its execution. The Device 3 will try dequeue another Element/Task

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:



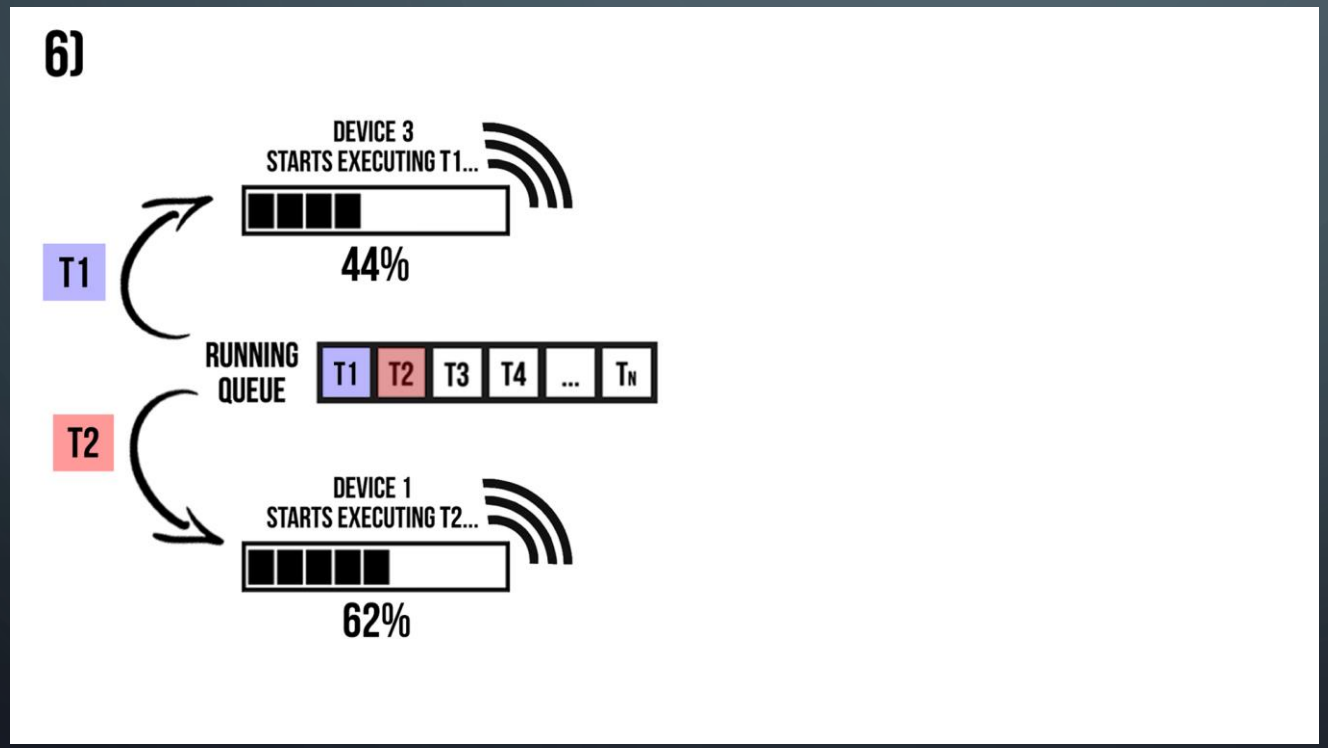
STEPS/OPERATION DESCRIPTION:

7) Situation 1 (TDR1 > TDR3):
In this situation, the winner of the Host Contest it's the Device 2. So the Execution's Host of T1 changes. The new Execution's Host of T1 will be the Device 2. The Device 3 stops the execution of T1 and Device 2 starts its execution. The Device 3 will try dequeue another Element/Task

TRYDEQUEUE'S TIME WINDOW (3)

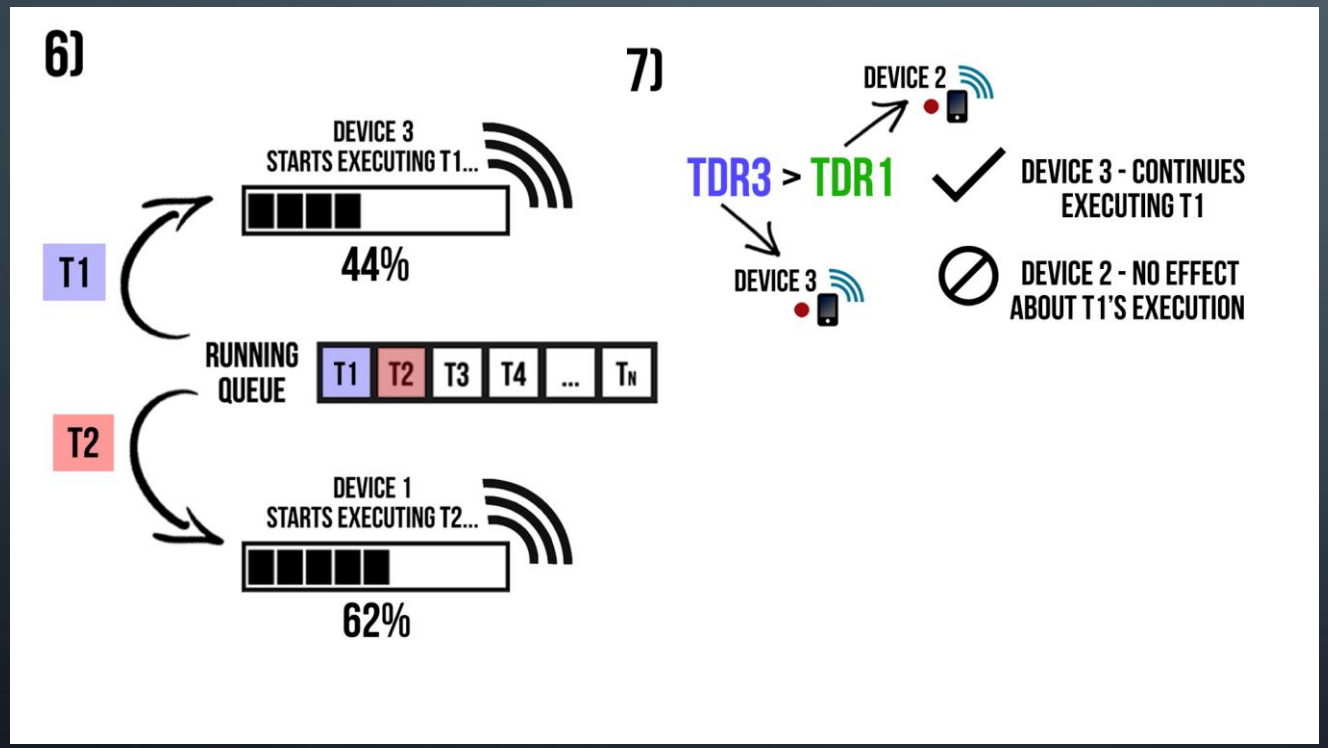
- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

STEPS/OPERATION DESCRIPTION:



TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

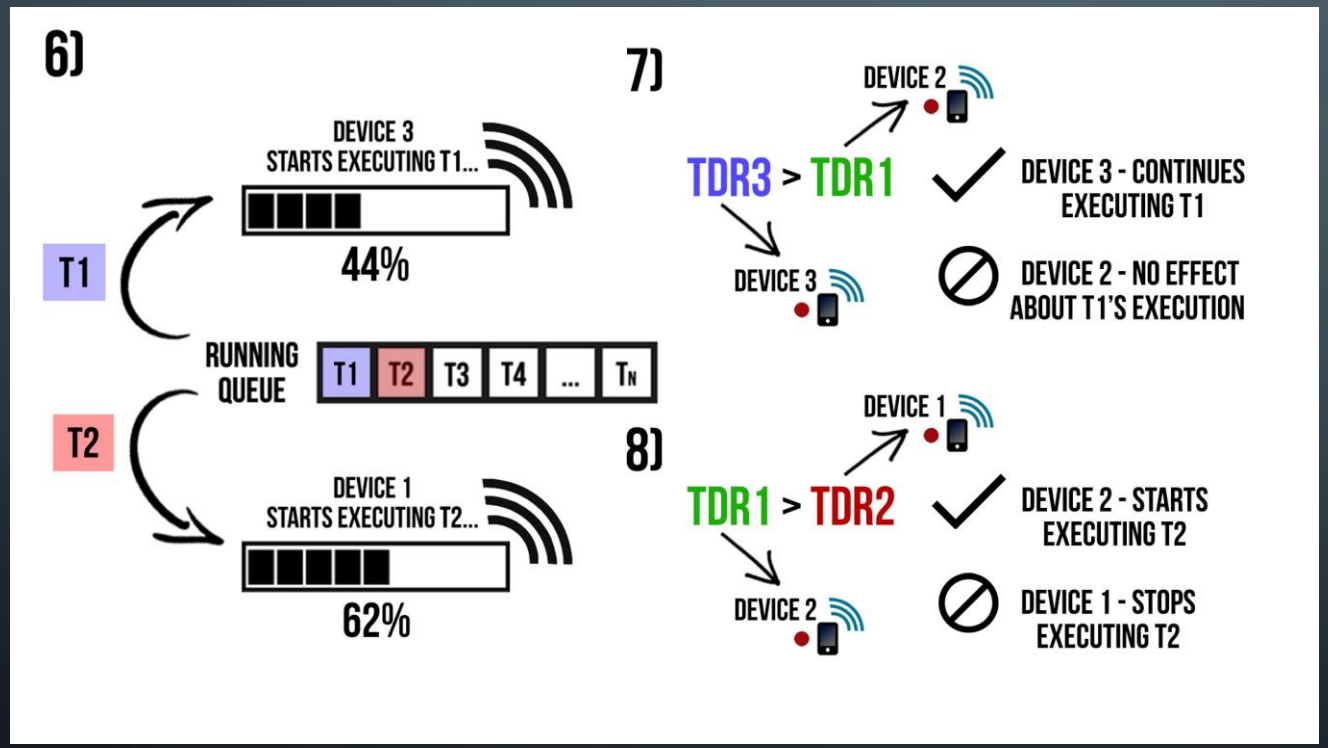


STEPS/OPERATION DESCRIPTION:

7) Situation 2 (TDR3 > TDR1):
In this situation, the winner of the Host Contest it's the Device 3. So the Execution's Host of T1 remains the same. The Execution's Host of T1 continues to be the Device 3. The Device 2 will start another Execution's Host Contest for the Element/Task T2. As the Execution's Host of Element/Task T2 it's the Device 1, the Execution's Host Contest will be between the Devices 2 and 1

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

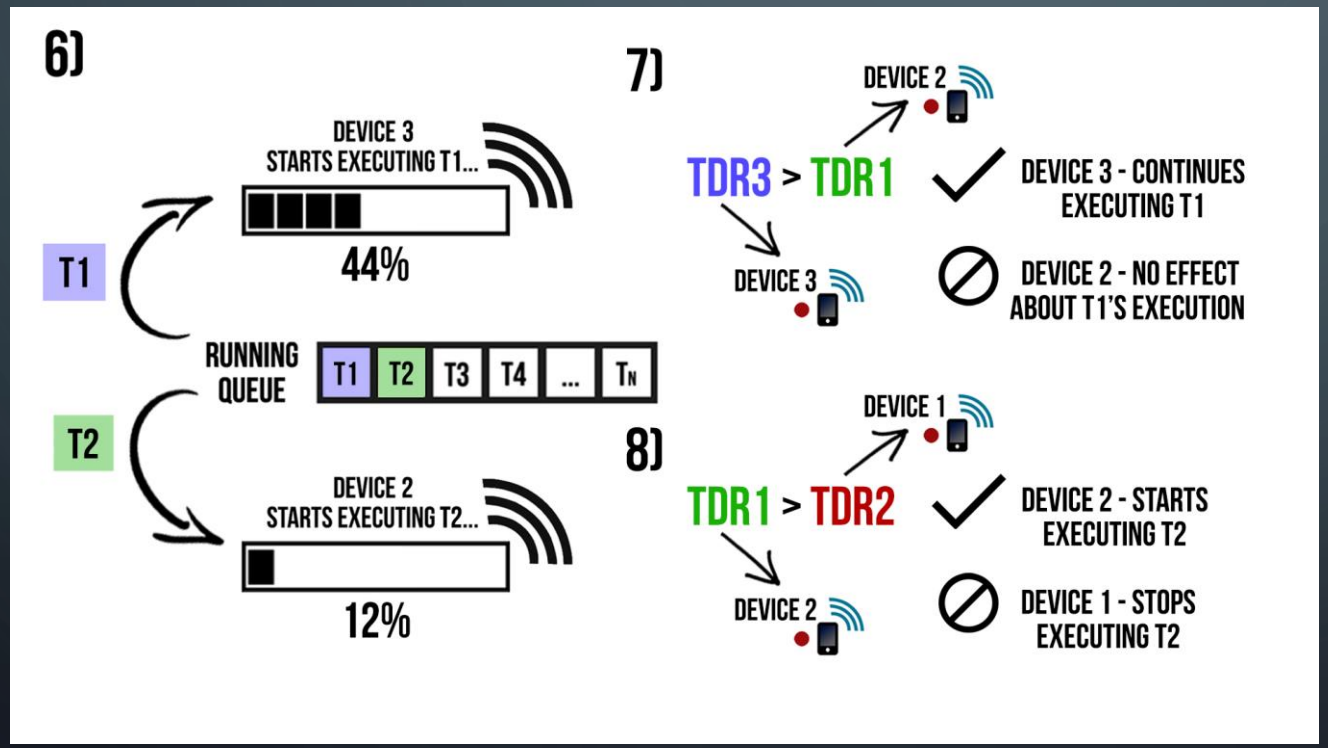


STEPS/OPERATION DESCRIPTION:

8) Situation 2.1 (TDR1 > TDR2):
   In this situation, the winner of the Host Contest it's the Device 2. So the Execution's Host of T2 changes. The new Execution's Host of T2 will be the Device 2. The Device 1 stops the execution of T2 and Device 2 starts its execution. The Device 1 will try dequeue another Element/Task

TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:



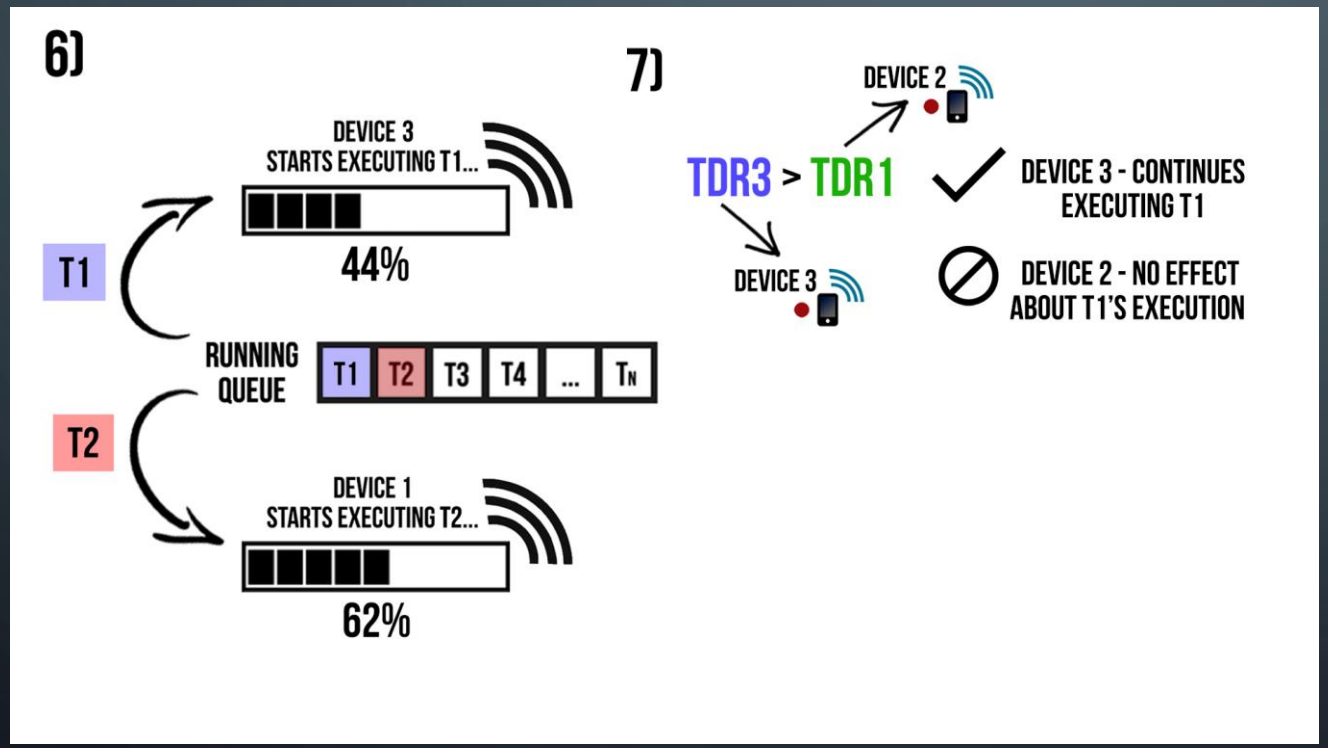
STEPS/OPERATION DESCRIPTION:

8) Situation 2.1 (TDR1 > TDR2):
   In this situation, the winner of the Host Contest it's the Device 2. So the Execution's Host of T2 changes. The new Execution's Host of T2 will be the Device 2. The Device 1 stops the execution of T2 and Device 2 starts its execution. The Device 1 will try dequeue another Element/Task

TRYDEQUEUE'S TIME WINDOW (3)

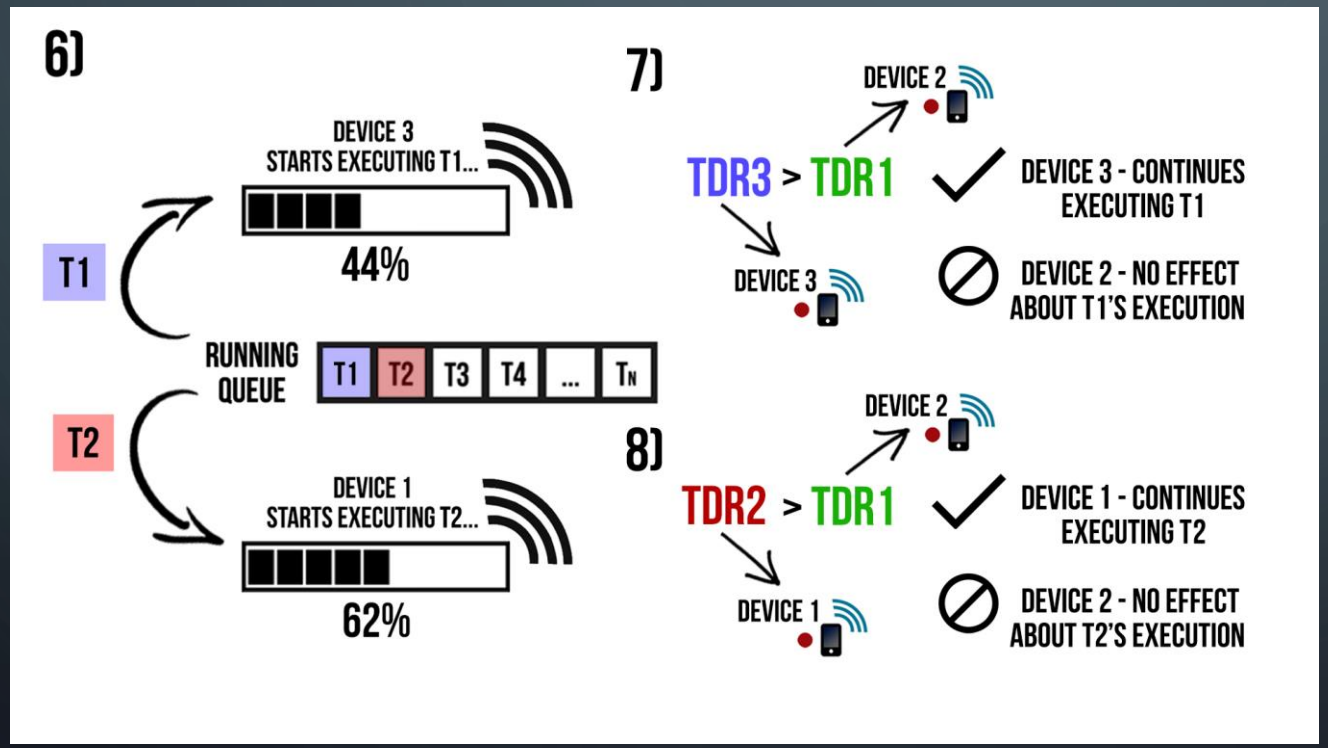
- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

STEPS/OPERATION DESCRIPTION:



TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:



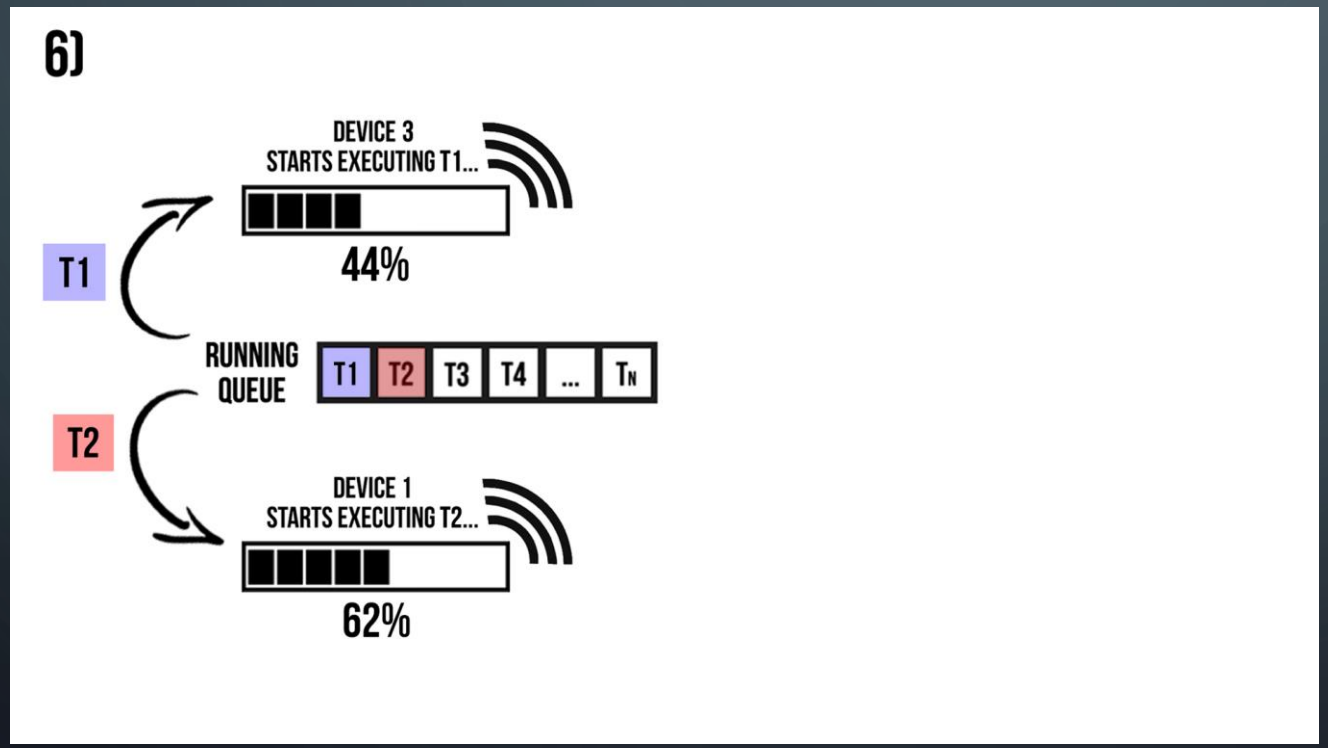
STEPS/OPERATION DESCRIPTION:

8) Situation 2.2 (TDR2 > TDR1):
   In this situation, the winner of the Host Contest it's the Device 1. So the Execution's Host of T2 remains the same. The Execution's Host of T2 continues to be the Device 1. The Device 2 will start another Execution's Host Contest for the Element/Task T3

TRYDEQUEUE'S TIME WINDOW (3)

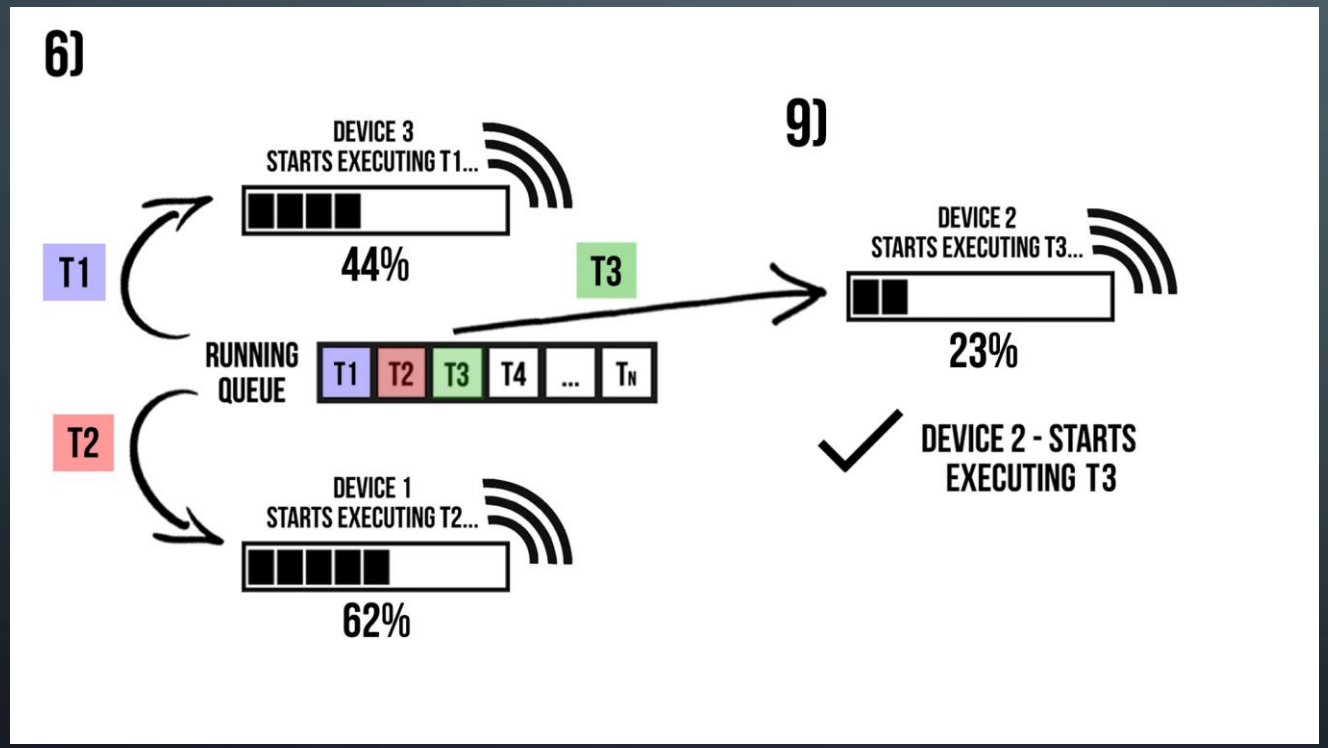
- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

STEPS/OPERATION DESCRIPTION:



TRYDEQUEUE'S TIME WINDOW (3)

- Situation 3) – TRYDEQUEUE Contest with 3 Devices and 1 TRYDEQUEUE message outside of the T time Window:

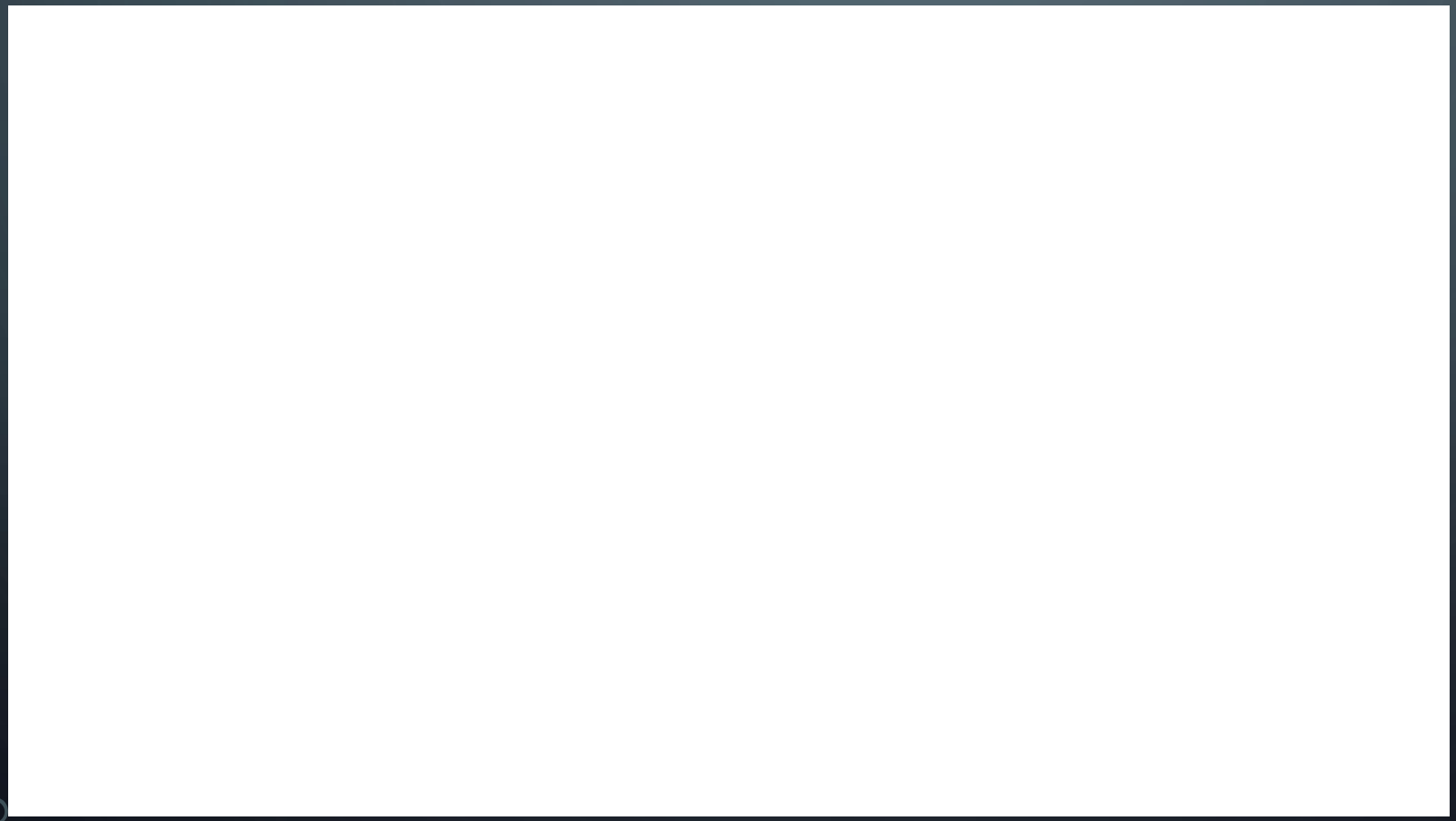


STEPS/OPERATION DESCRIPTION:

9) Situation 2.2.1 (No Execution Host): As Element/Task T3 don't have any Execution's Host, the Device 2 will dequeue it from the queue and starts its execution

TASK'S EXECUTION PROCESS (1)

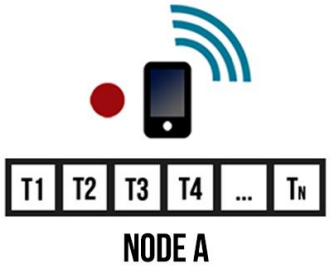
- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

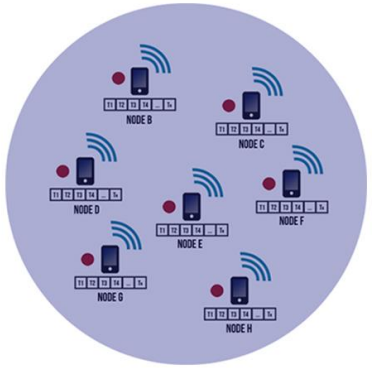


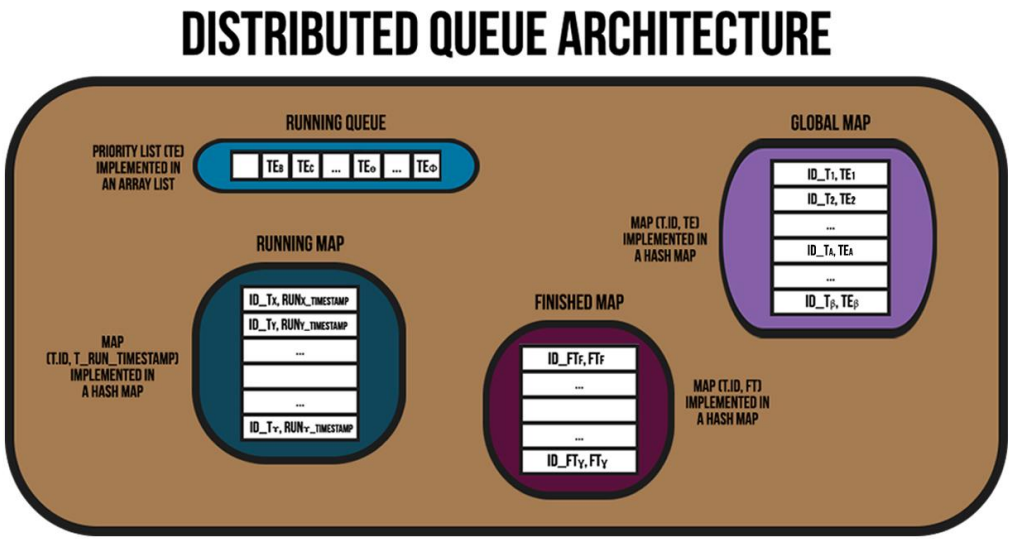
STEPS/OPERATION DESCRIPTION:

TASK'S EXECUTION PROCESS (1)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:



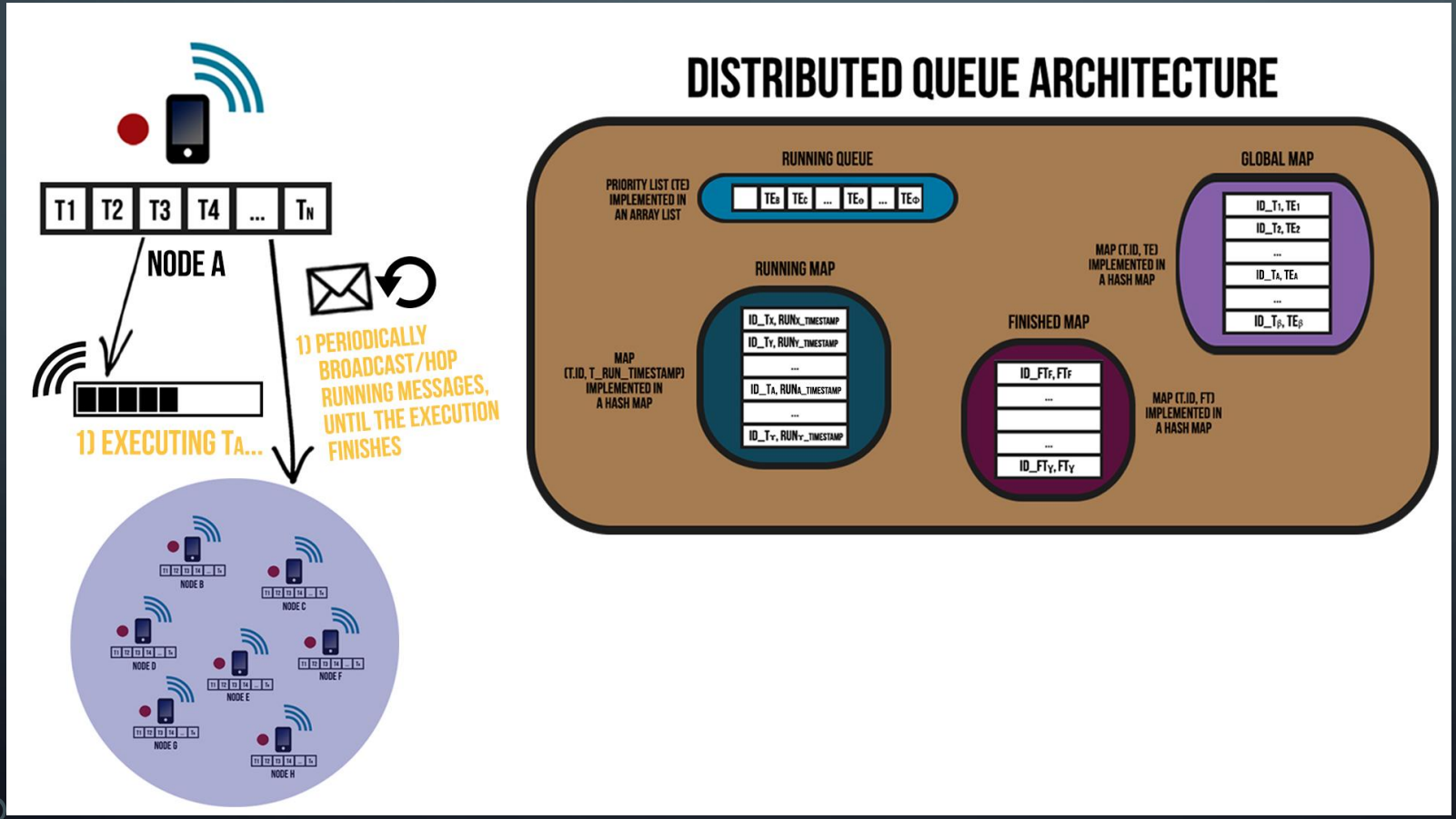




STEPS/OPERATION DESCRIPTION:

TASK'S EXECUTION PROCESS (1)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

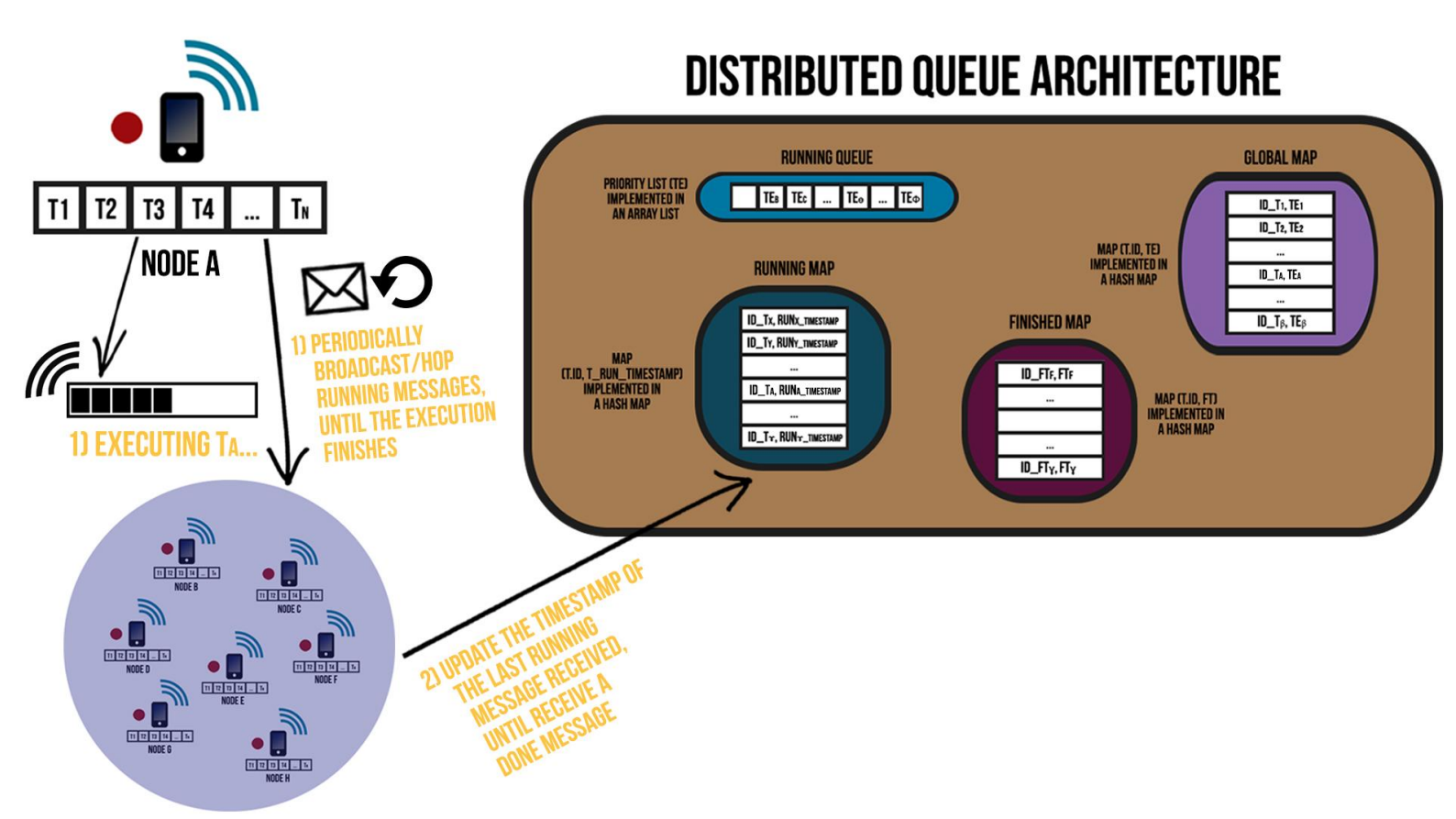


STEPS/OPERATION DESCRIPTION:

1) Executing TA... / Periodically Broadcast/Hop RUNNING messages, until the execution finishes

TASK'S EXECUTION PROCESS (1)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

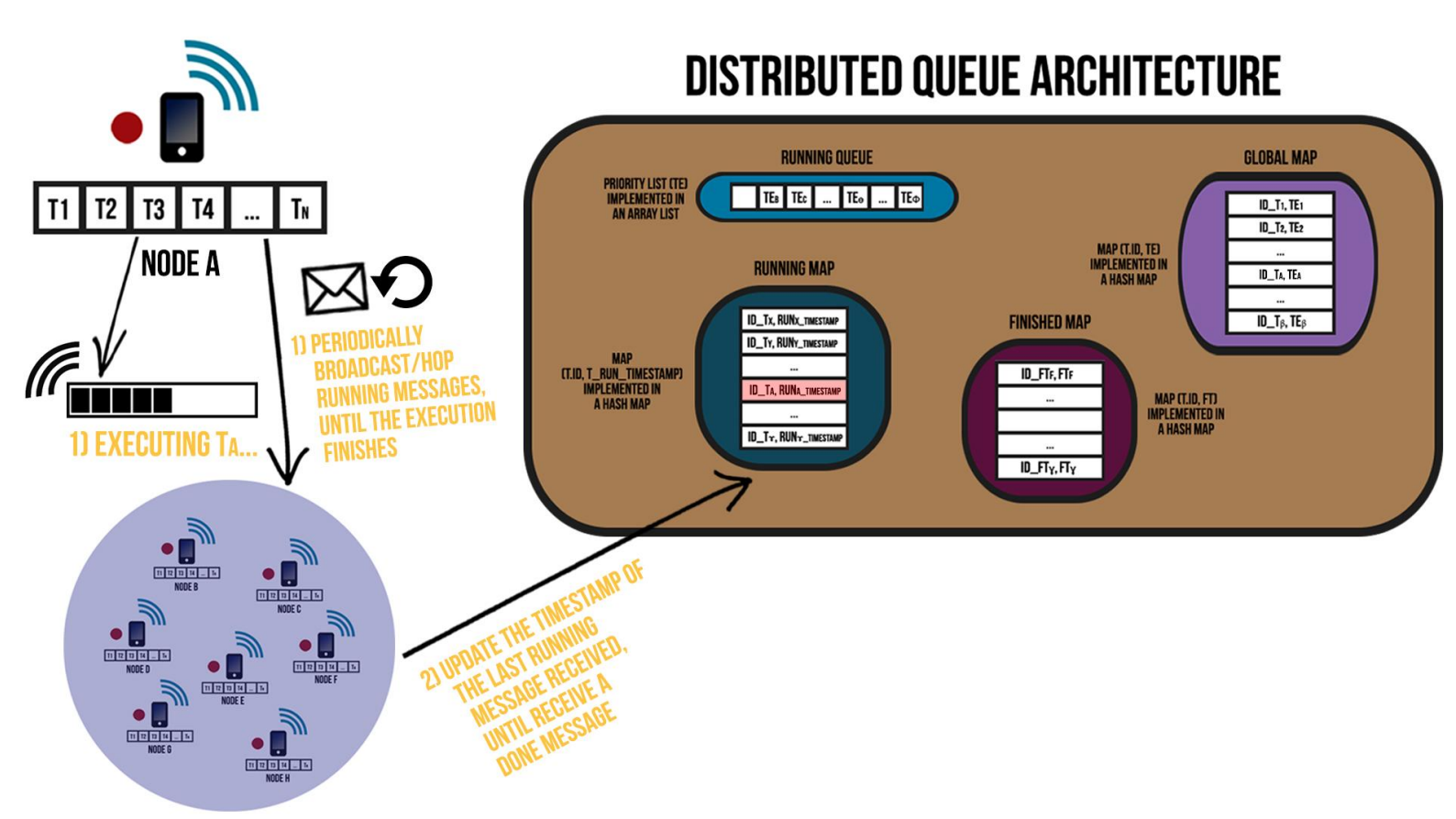


STEPS/OPERATION DESCRIPTION:

1) Executing TA... / Periodically Broadcast/Hop RUNNING messages, until the execution finishes
2) Update the timestamp of the last RUNNING message received, until receive a DONE message

TASK'S EXECUTION PROCESS (1)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

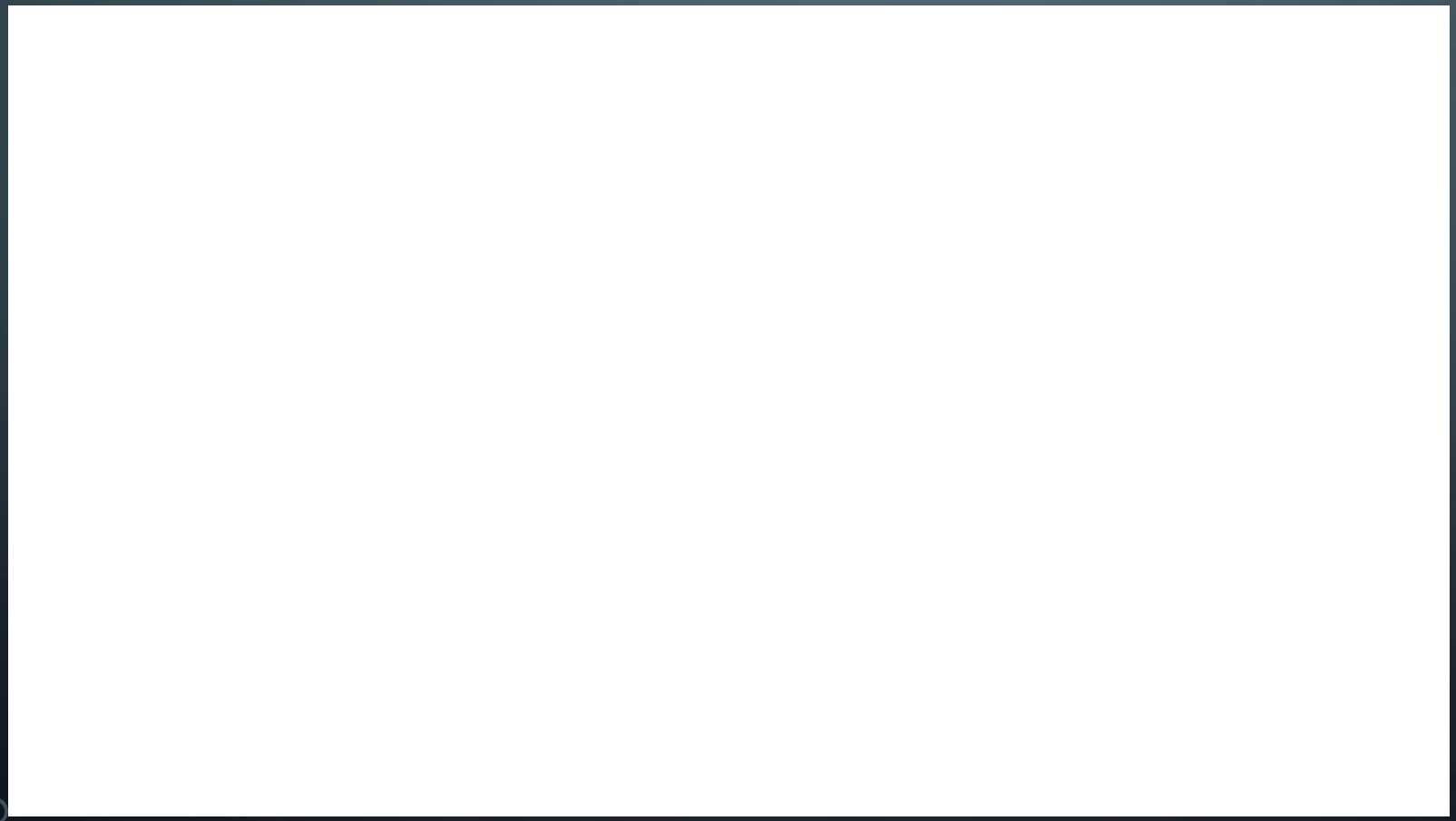


STEPS/OPERATION DESCRIPTION:

1) Executing TA... / Periodically Broadcast/Hop RUNNING messages, until the execution finishes
2) Update the timestamp of the last RUNNING message received, until receive a DONE message

TASK'S EXECUTION PROCESS (2)

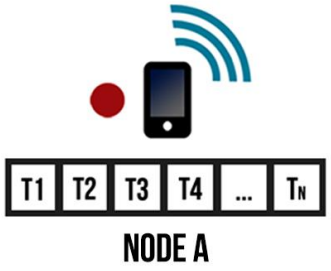
- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

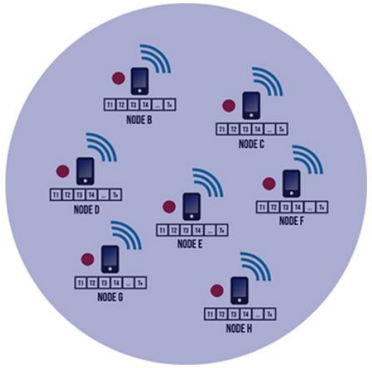


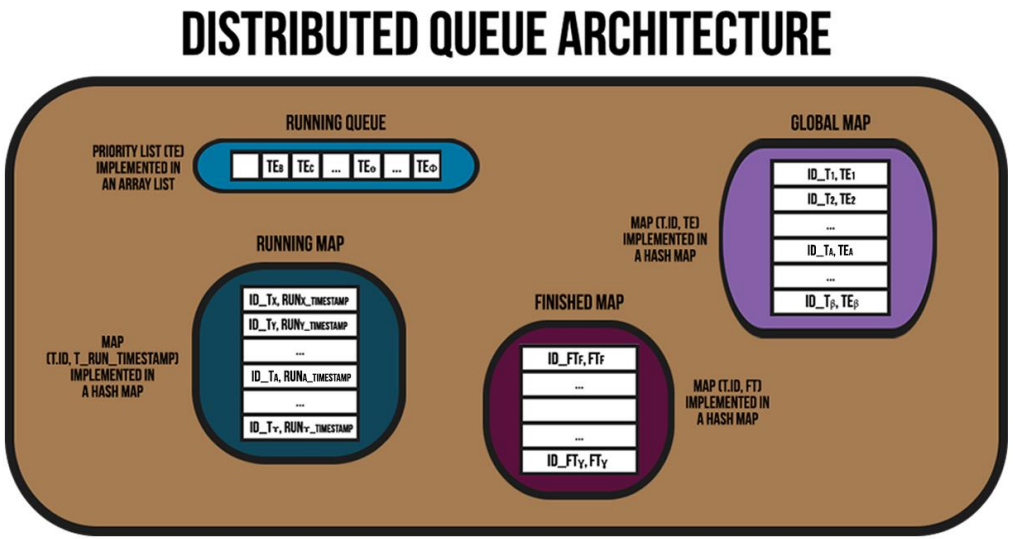
STEPS/OPERATION DESCRIPTION:

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:



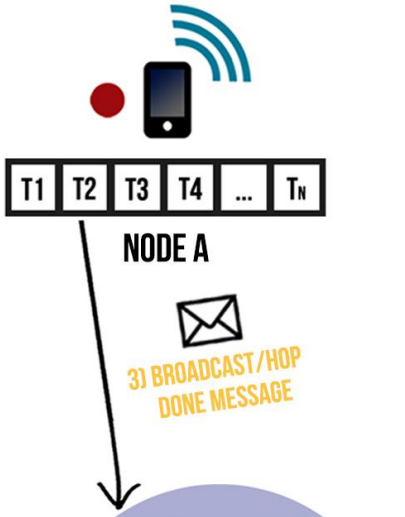


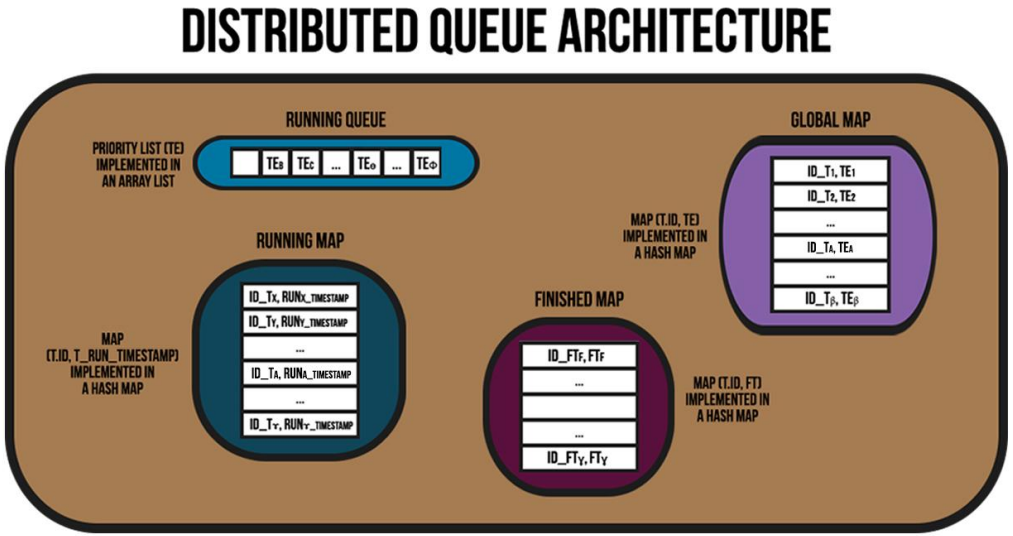


STEPS/OPERATION DESCRIPTION:

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:



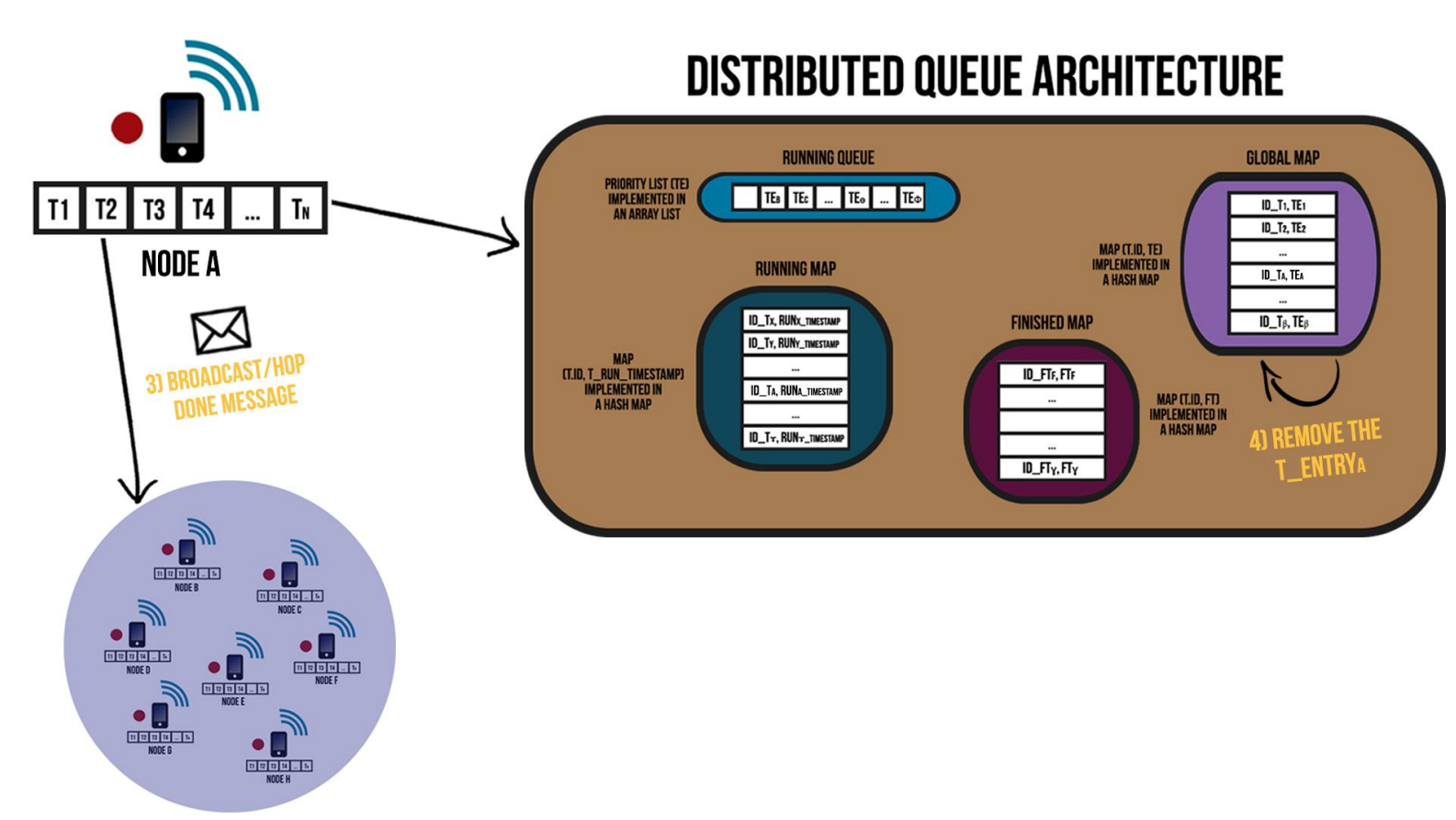


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

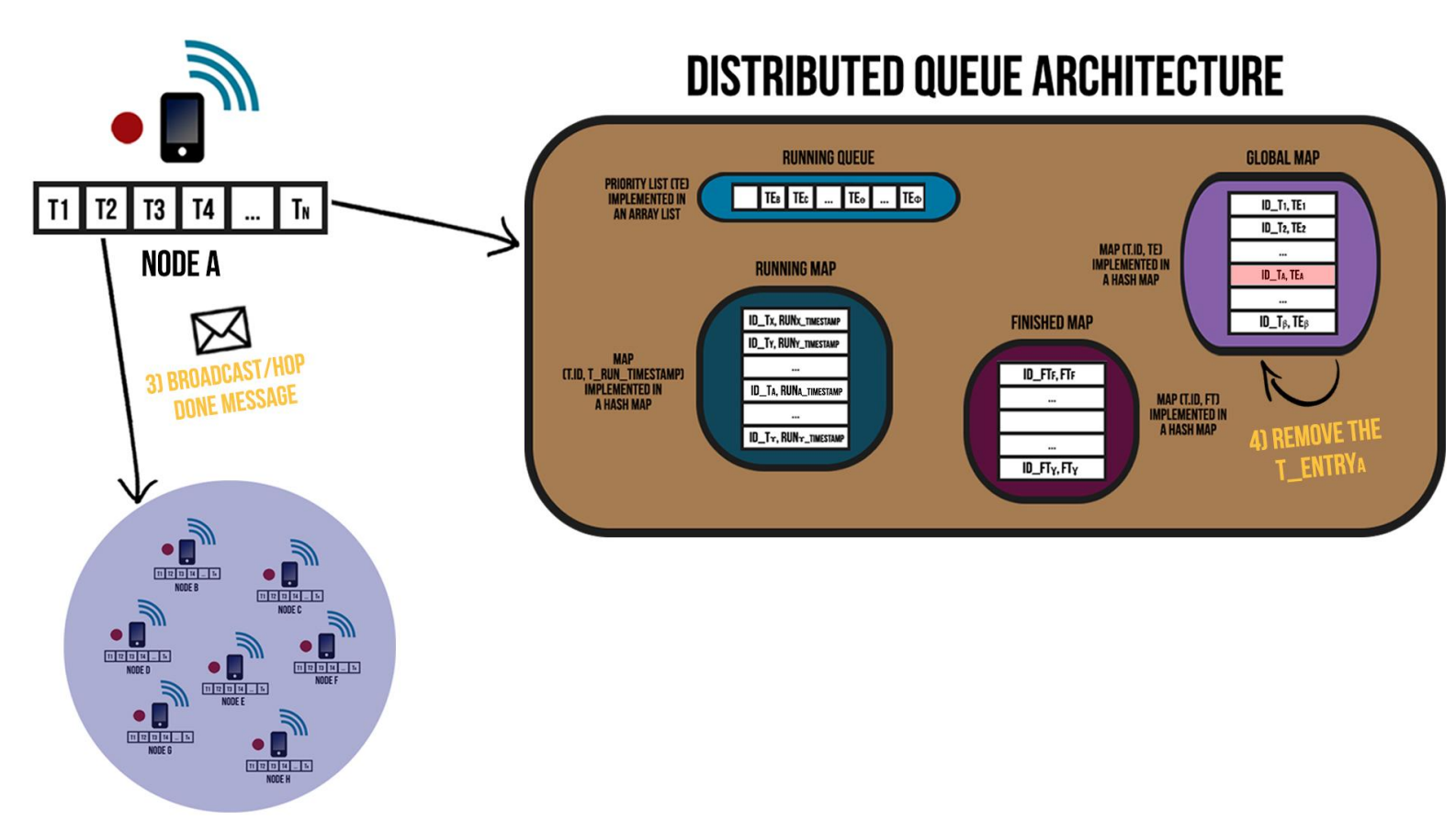


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

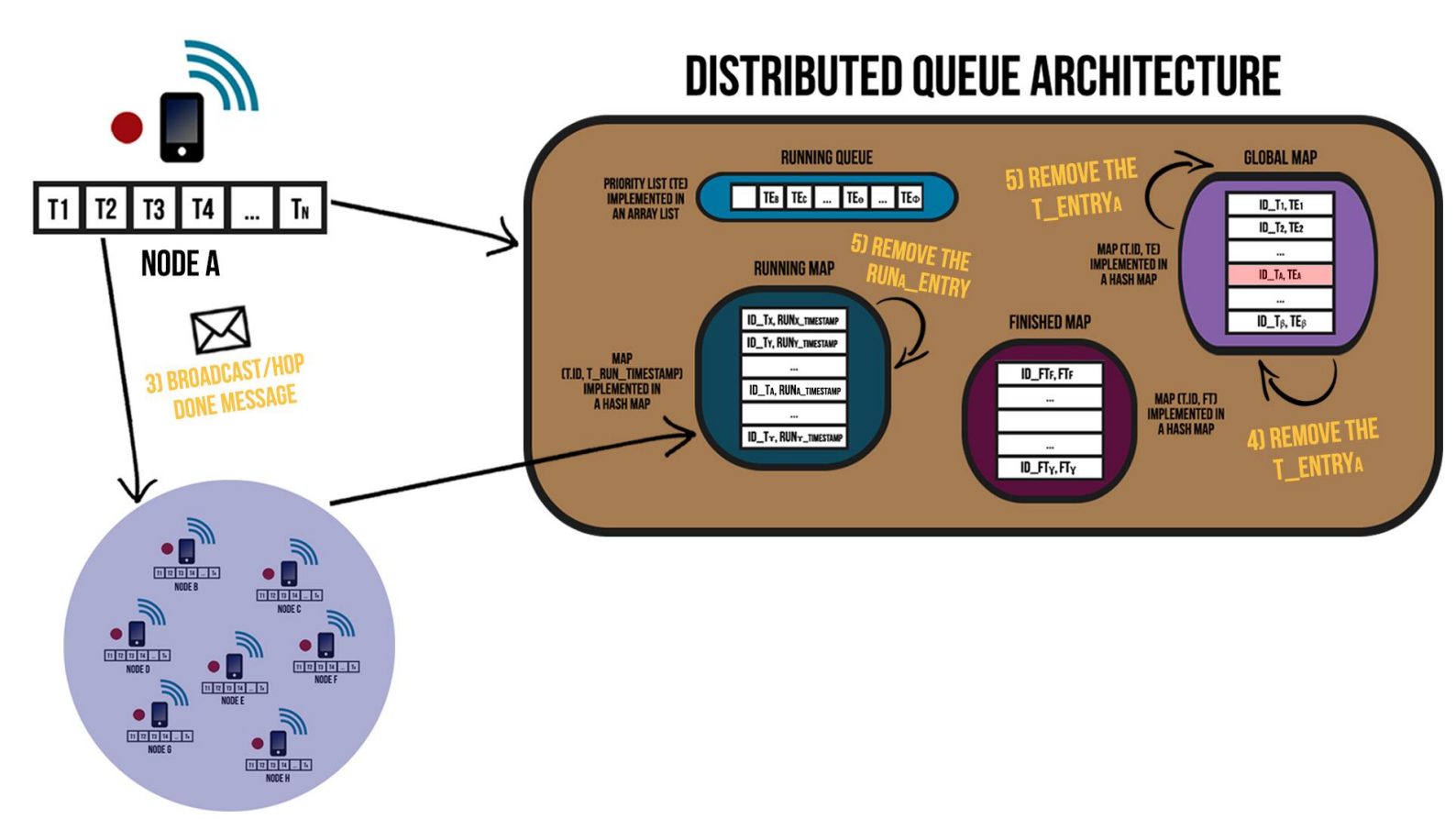


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

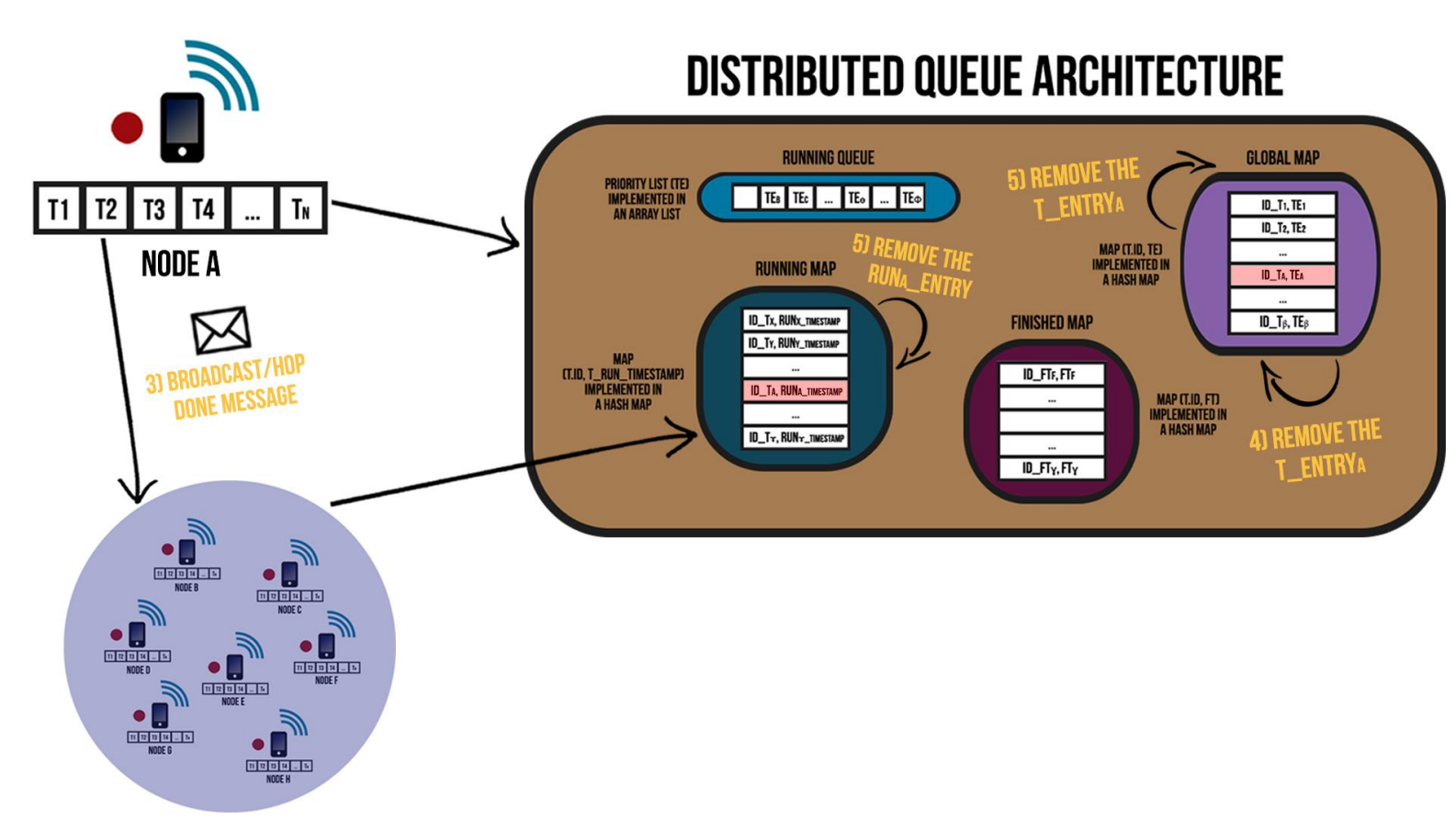


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

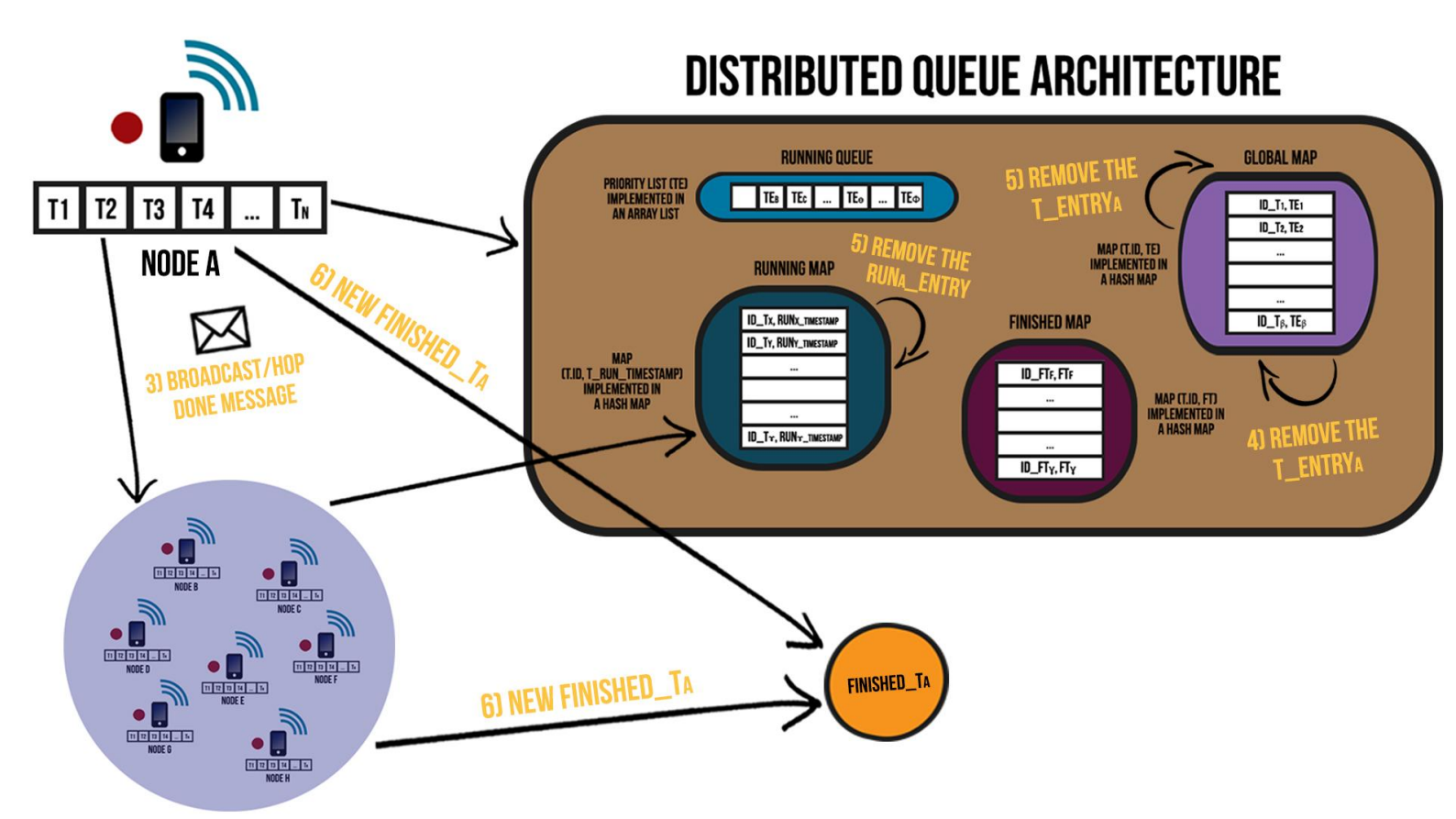


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

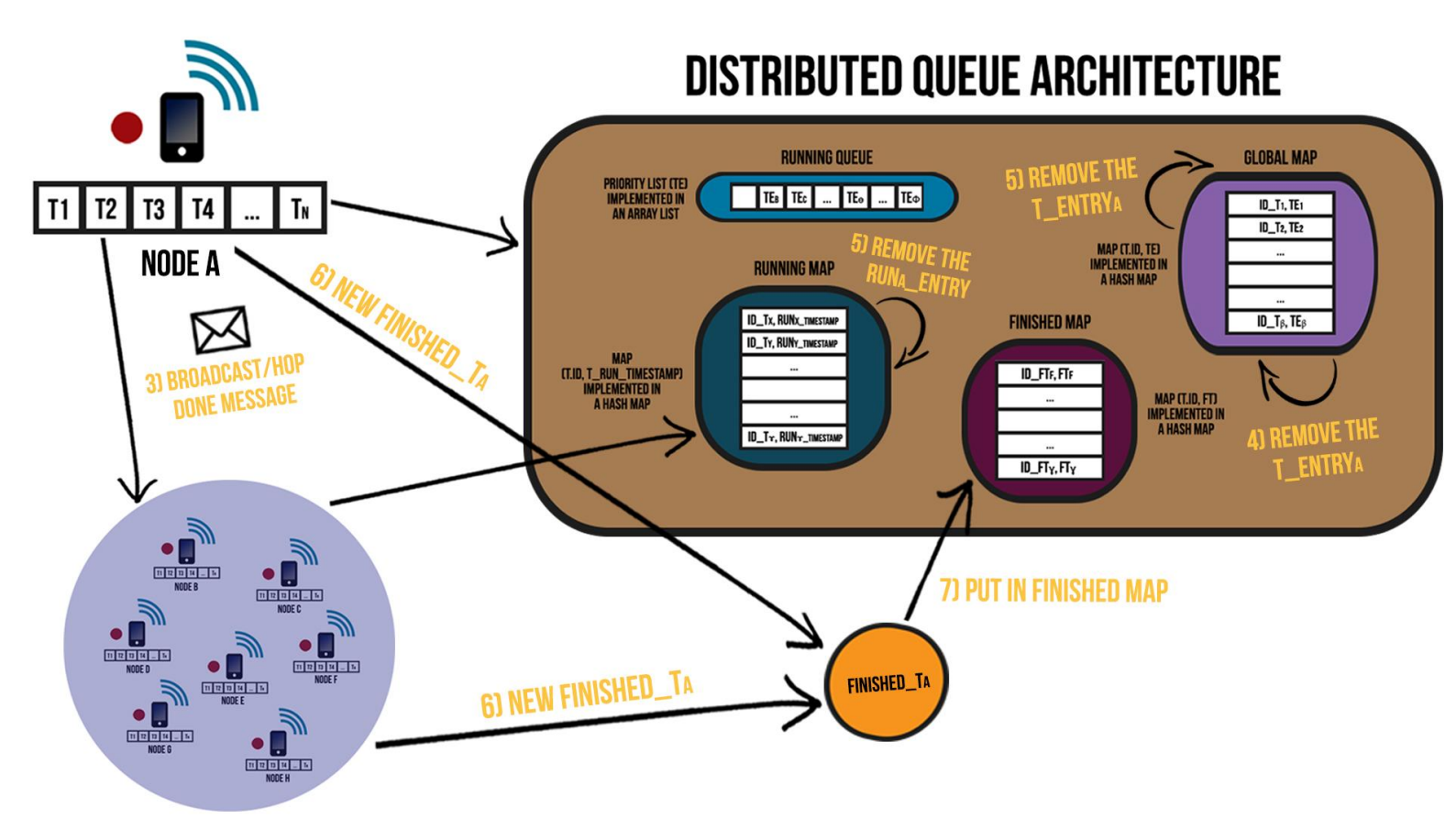


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA
6) New Finished_TA

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

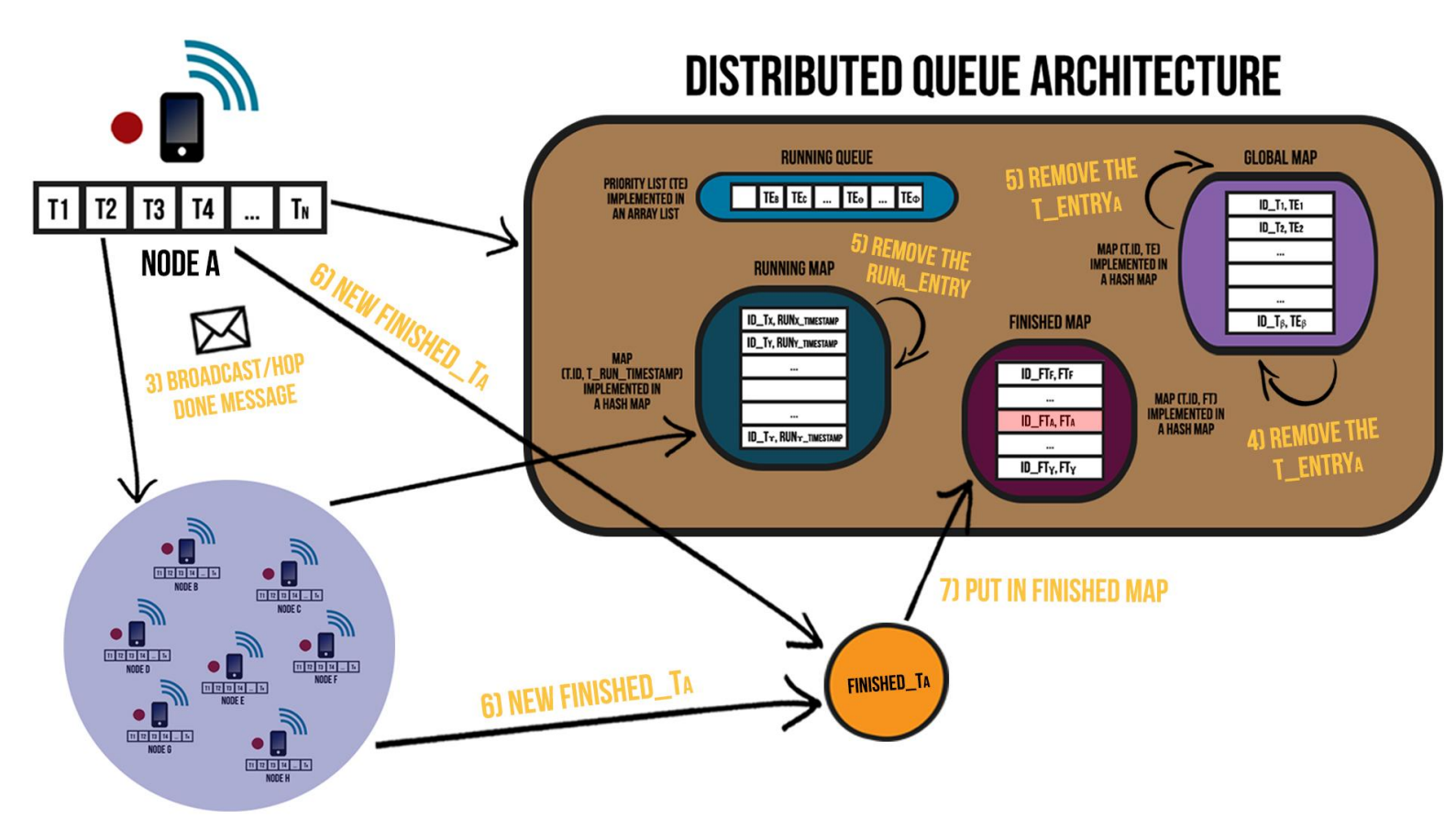


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA
6) New Finished_TA
7) Put in Finished Map

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:

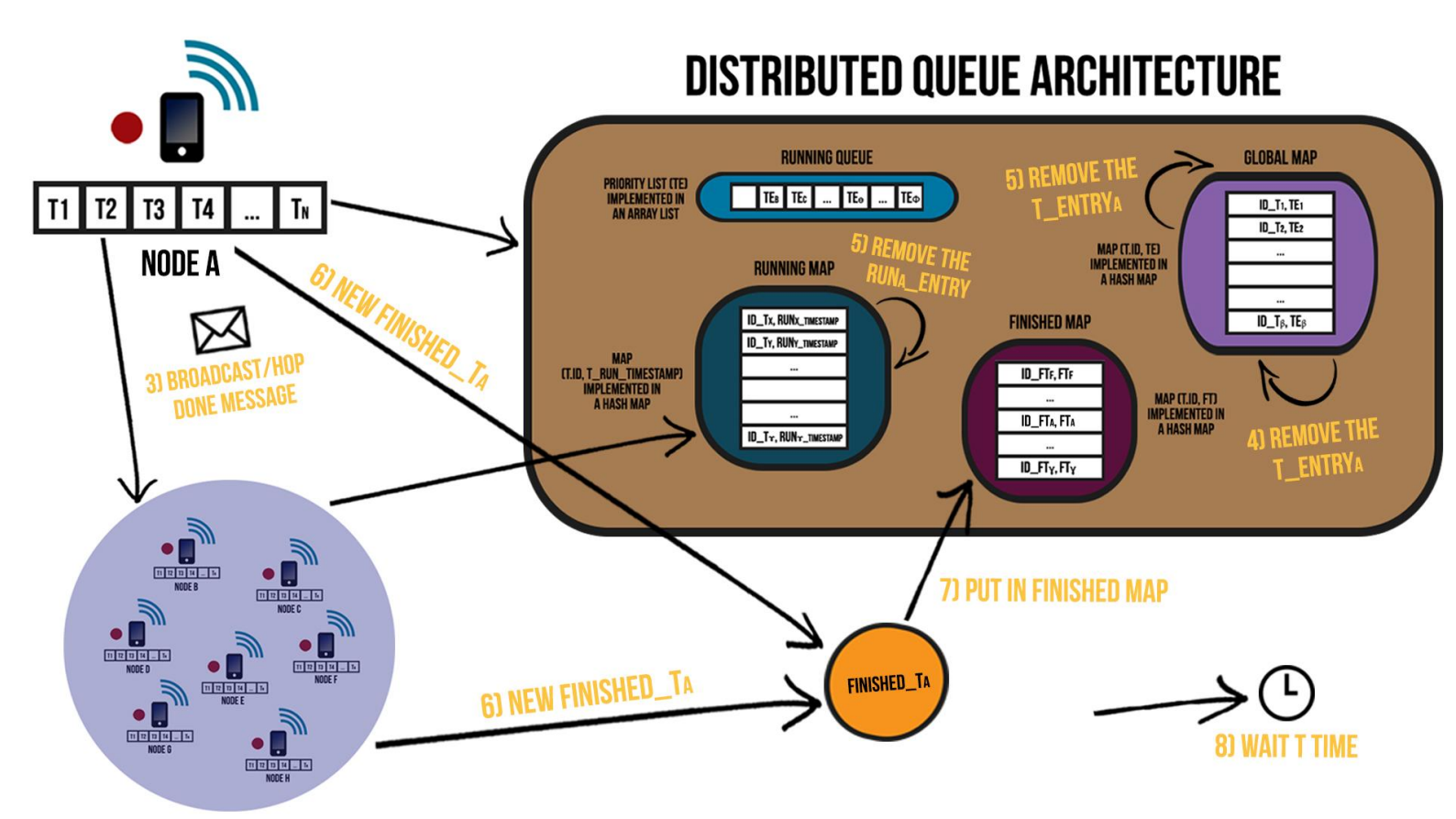


STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA
6) New Finished_TA
7) Put in Finished Map

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:



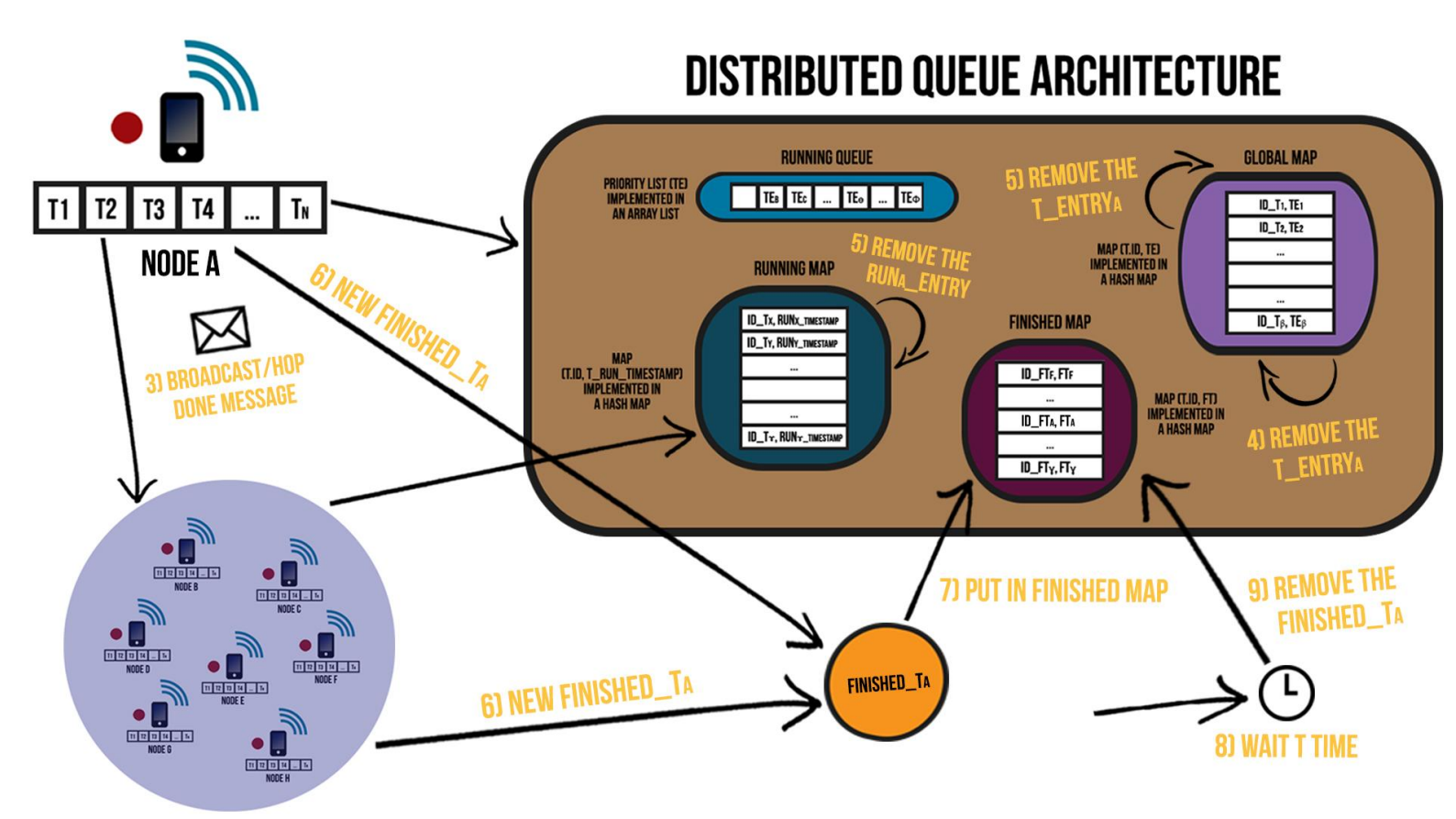
STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA
6) New Finished_TA
7) Put in Finished Map
8) Wait T time

(*) – T is previously defined accordingly to the pre-definition of the system architecture and network environment

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:



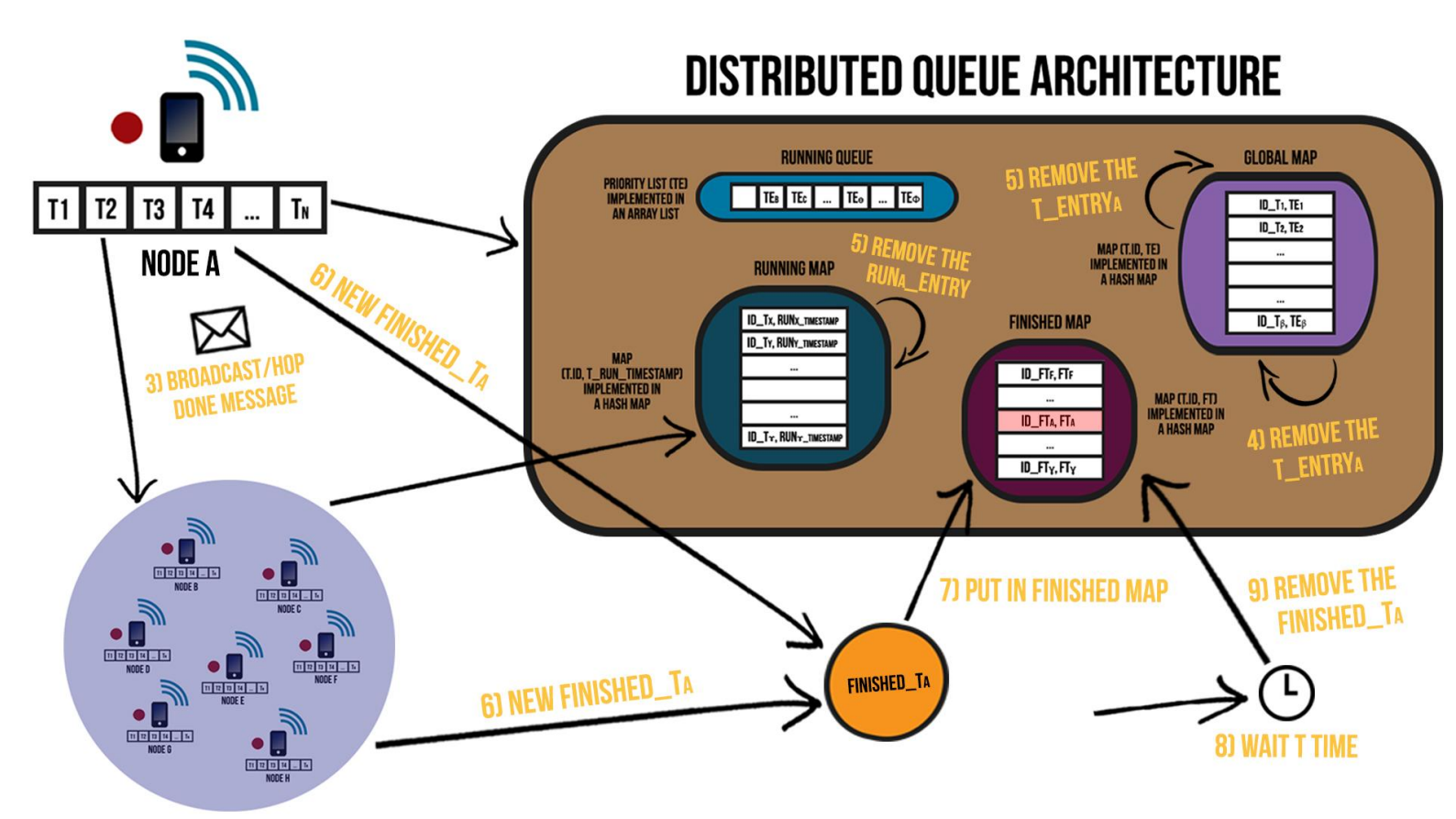
STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA
6) New Finished_TA
7) Put in Finished Map
8) Wait T time
9) Remove the Finished_TA

(*) – T is previously defined accordingly to the pre-definition of the system architecture and network environment

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:



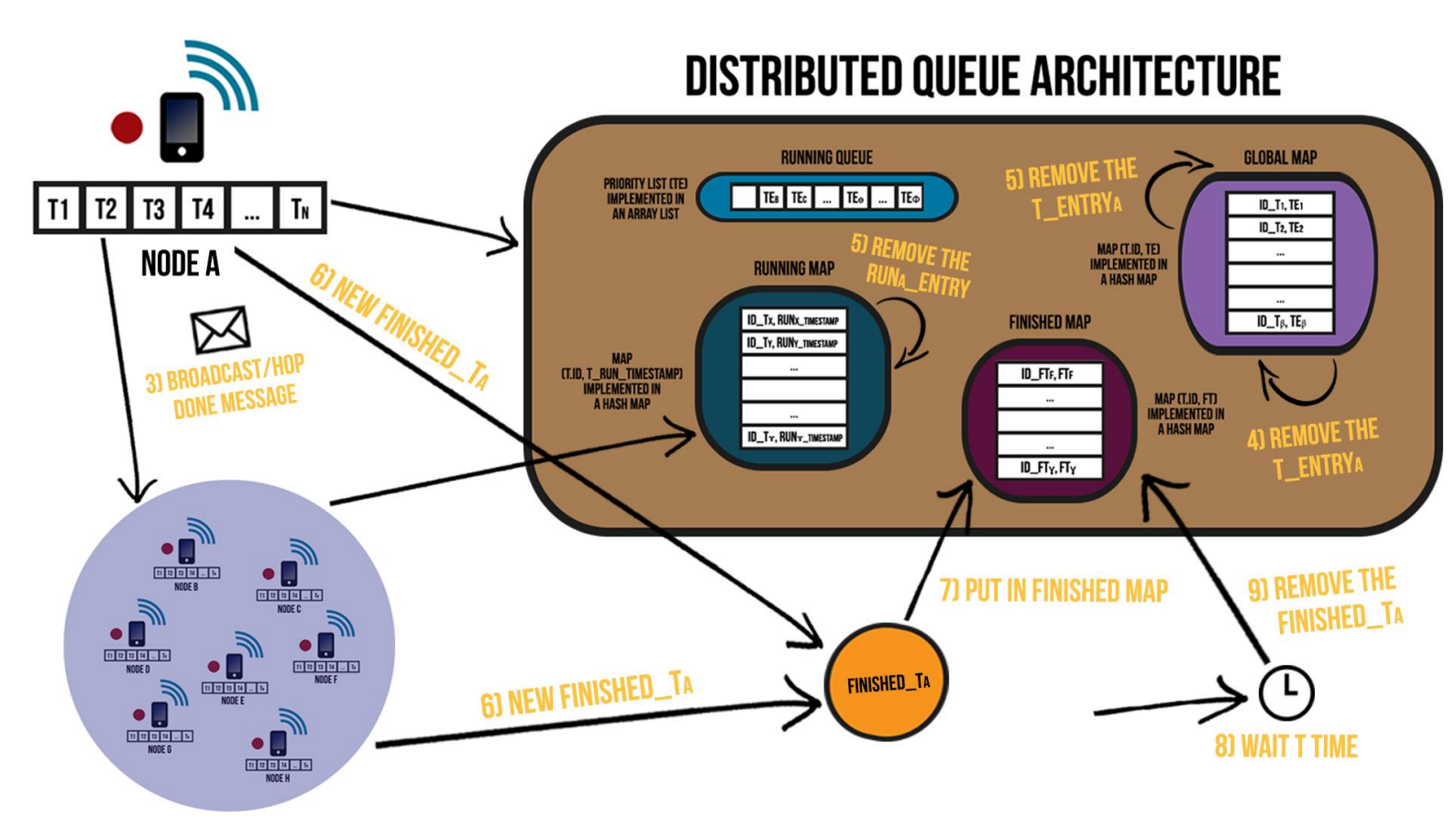
STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA
6) New Finished_TA
7) Put in Finished Map
8) Wait T time
9) Remove the Finished_TA

(*) – T is previously defined accordingly to the pre-definition of the system architecture and network environment

TASK'S EXECUTION PROCESS (2)

- The Task's execution process of the Distributed Queue System Architecture have the following behavior:



STEPS/OPERATION DESCRIPTION:

3) Broadcast/Hop DONE message
4) Remove the T_EntryA
5) Remove the RunA_Entry / Remove the T_EntryA
6) New Finished_TA
7) Put in Finished Map
8) Wait T time
9) Remove the Finished_TA

(*) – T is previously defined accordingly to the pre-definition of the system architecture and network environment

PROBLEMS WITH THIS SOLUTION?

- How all the communication it's made by Broadcast/Hop messages and in volatile environments, with the constant entrance and exit of Devices from the range/neighborhood, can happen that some Broadcast/Hop messages will be lost or some Devices not will receive always all the messages;
- In some cases, it's possible that a Device can receive a message about an unknown Task, locally;
- Also, can happen that a Device receives messages by a different order that the one that was to be supposed;

FAILED RESOLUTION SERVICE

- When a Device starts to execute locally a Task, sends periodically Broadcast/Hop RUNNING messages (Heartbeats);
- If the other Devices remain a T time without receiving that RUNNING messages (Heartbeats), will assume that the execution of that Task by its Host Device failed for some reason (e.g. : the Device's system failed, the Device leaves the network/range, etc.);
- In that case, the Task that is allocated in the Devices' Running Map, will be removed from it and allocated again in the Running Queue. The information about that Task will be also updated;
- A Device that detects this situation, sends a Broadcast/Hop FAILED message to the other Devices in the network/range;

DOYOUKNOWTHISELEMENT/ TAKEMYELEMENT/ALREADYFINISHEDELEMENT (UNKNOWN TASKS) (1)

- When a Device receives a message about a Task that doesn't know, locally, sends a Broadcast/Hop DOYOUKNOWTHISELEMENT to the other Devices in the network/range, asking for the data and information of that unknown Task;
- When a Device receives a DOYOUKNOWTHISELEMENT message, verifies if knows that Task locally, and if it's known, do the following:
  - Generates random number T, between a certain interval and waits T time;
  - During T time, verify if some another Device answer for the same situation and only answer for it, if no other Device answer too for it, inside the T time window. This procedure allows to avoid *bottleneck* situations in the network/range;

DOYOUKNOWTHISELEMENT/ TAKEMYELEMENT/ALREADYFINISHEDELEMENT (UNKNOWN TASKS) (2)

- A Device can answer to the DOYOUKNOWTHISELEMENT messages received by two ways:
  - TAKEMYELEMENT – When the Element/Task that, it was being asked for, isn't executed yet. And in that situation, the Device that receives the message, sends all the data and information about that Element/Task;
  - ALREADYFINISHEDELEMENT – When the Element/Task that, it was being asked for, was already executed and finished its execution. And in that situation, the Device that receives the DOYOUKNOWTHISELEMENT message, sends only a Broadcast/Hop message informing that Element/Task, was already executed, for the Device can update the information about it;

DOYOUKNOWTHISELEMENT - STRATEGIES USED

- Was adopted some strategies, for the situations that, a Device receives a message about a Task that doesn't know locally:
  - ALWAYSWANTTHISELEMENT – A Device with this strategy, when receives a message about an unknown Task, ALWAYS ask the other Devices, for the data and information of it;
  - DEPENDSONDEVICESNUMWANTTHISELEMENT - A Device with this strategy, when receives a message about an unknown Task, as the other Devices, for the data and information of it, only if, exists a certain minimum number of Devices in the network/range;
  - NEVERWANTTHISELEMENT – A Device with this strategy, when receives a message about an unknown Task, NEVER ask the other Devices, for the data and information of it;

EVALUATION – SOME EXPERIMENTAL RESULTS (1)

(4 DEVICES WITH 128 ELEMENTS (TASKS), 1 DEVICE ENQUEUEING ALL ELEMENTS, WITHOUT PACKETS LOSS) (*)

(*) – Experimental results still with some Bugs

EVALUATION – SOME EXPERIMENTAL RESULTS (2)

(8 DEVICES WITH 128 ELEMENTS (TASKS), 4 DEVICES ENQUEUEING 32 ELEMENTS EACH ONE, WITHOUT PACKETS LOSS)

(*) – Experimental results still with some Bugs