August 2023

MEMORY REQUEST SCHEDULER WITH MODULAR CONFIGURABLE MEMORY REQUEST PRIORITY COMPARING UNITS

MEMORY REQUEST SCHEDULER WITH MODULAR CONFIGURABLE MEMORY REQUEST PRIORITY COMPARING UNITS

BACKGROUND OF THE DISCLOSURE

[0001] Dynamic random-access memory (DRAM) is a type of random-access semiconductor memory. DRAM has been widely used in computer systems for many years, due to its low-cost, simple structure, and scalability. High bandwidth memory (HBM) is a high-speed computer memory interface for 3D-stacked synchronous dynamic random-access memory (SDRAM).

[0002] A memory controller (MC) is a digital circuit that schedules the memory requests and manages the flow of data going to and from the memory in DRAM/HBM systems. A memory controller contains the logic necessary for many different functions, including the logic to read and write to the memory and the logic to track the physical states in order to maintain data integrity. A memory controller may have many goals, for example, maximizing throughput, minimizing latency, maximizing fairness, and the like. Therefore, improved memory controller scheduling techniques that can handle various memory accesses from different sources efficiently would be desirable.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] Various embodiments of the disclosure are disclosed in the following detailed description and the accompanying drawings.

[0004] Figure 1 illustrates an exemplary memory controller 100 that schedules the memory requests and manages the flow of data going to and from the memory in a memory system.

[0005] Figure 2 illustrates a plurality of DRAM chips (202, 204, and 206) that may be coupled to memory controller 100.

[0006] Figure 3 illustrates an exemplary memory request scheduling system 300.

[0007] Figure 4 illustrates an exemplary process 400 for scheduling memory requests in a memory system.

[0008] Figure 5 illustrates an exemplary process 500 for determining at least a part of the issuing schedule order of the memory request.

[0009] Figure 6 illustrates another exemplary memory request scheduling system 600.

[0010] Figure 7 illustrates another example that parallel processing may be achieved by pipelining the decision-making.

[0011] Figure 8 illustrates another exemplary memory request scheduling system 800.

[0012] Figure 9 illustrates another example that two different linked lists of memory request priority comparing function units may be used for determining the scheduling order of the memory requests.

DETAILED DESCRIPTION

[0013] The disclosure can be implemented in numerous ways, including as a process; an apparatus; a system; a composition of matter; a computer program product embodied on a computer readable storage medium; and/or a processor, such as a processor configured to execute instructions stored on and/or provided by a memory coupled to the processor. In this specification, these implementations, or any other form that the disclosure may take, may be referred to as techniques. In general, the order of the steps of disclosed processes may be altered within the scope of the disclosure. Unless stated otherwise, a component such as a processor or a memory described as being configured to perform a task may be implemented as a general component that is temporarily configured to perform the task at a given time or a specific component that is manufactured to perform the task. As used herein, the term 'processor' refers to one or more devices, circuits, and/or processing cores configured to process data, such as computer program instructions.

[0014] A detailed description of one or more embodiments of the disclosure is provided below along with accompanying figures that illustrate the principles of the disclosure. The disclosure is described in connection with such embodiments, but the disclosure is not limited to any embodiment. The scope of the disclosure is limited only by the claims and the disclosure encompasses numerous alternatives, modifications and equivalents. Numerous specific details are set forth in the following description in order to provide a thorough understanding of the disclosure. These details are provided for the purpose of example and the disclosure may be practiced according to the claims without some or all of these specific details. For the purpose of clarity, technical material that is known in the technical fields related to the disclosure has not been described in detail so that the disclosure is not unnecessarily obscured.

[0015] Figure 1 illustrates an exemplary memory controller 100 that schedules the memory requests and manages the flow of data going to and from the memory in a memory system. Memory controller 100 schedules or reorders the incoming memory requests and issues them on the bus. The memory system may use DRAM, HBM, or other media. DRAM/HBM systems have many timing constraints associated with accessing different regions of the memory, and if these timing constraints are not adhered to, data corruption may occur. Therefore, a good

scheduling policy is required in optimizing for best performance and to maintain the integrity of data.

[0016] Memory controller 100 includes a memory controller input/output (I/O) handler 102, a fragment assembler 104, an address decoder 108, a set of bank queues 110, a request scheduler 112, and a memory I/O handler 114. Memory controller I/O handler 102 receives various inputs of the memory controller and communicates with fragment assembler 104. For example, the inputs received by memory controller I/O handler 102 may include requests received from a host that is connected to the memory controller. Fragment assembler 104 sends the requests 106 to address decoder 108 for processing. Since the memory is divided into different banks, the processed requests are sent by address decoder 108 to a set of bank queues 110. For example, requests that are sent to bank 0 are sent to bank queue 0 (B0), requests that are sent to bank 1 are sent to bank queue 1(B1), and so on.

[0017] After memory requests have been queued in the set of bank queues 110, they are processed by memory controller 100, and its internal scheduler, request scheduler 112, is invoked. Request scheduler 112 manages and monitors the outstanding requests that are stored in the set of bank queues 110. It also monitors different aspects of the memory banks, such as which rows within the memory banks are open, and schedules and prioritizes certain requests 113 to be sent to the memory that is connected to the memory controller via memory I/O handler 114. Memory I/O handler 114 sends respond messages 116 to fragment assembler 104.

[0018] In a DRAM memory system, memory controller 100 may be connected to a plurality of dual in-line memory modules (DIMMs) via a channel. Each DIMM may have one or more ranks. Each rank has multiple DRAM chips. Each chip has multiple banks. Each bank is a matrix of rows and columns.

[0019] Figure 2 illustrates a plurality of DRAM chips (202, 204, and 206) that may be coupled to memory controller 100. Each DRAM chip has multiple banks, e.g., bank 0 memory array 210, bank 1 memory array (not shown), and up to bank n memory array 210. To read from or write to an address on a DRAM chip, a row address is sent to a row decoder 208. Row decoder 208 selects a row within a bank, and the row of data is sent to a row buffer 212. The row of data may be in any size, such as 1 kilobytes (kB), 2 kB, or 8 kB. A column address is

sent to a column decoder 214 to select the column(s) of the row in the row buffer. For example, only 64 bytes within a row of size 1 kB in the row buffer may be selected. In other words, there is an overhead of fetching extra bytes into the row buffer when only 64 bytes of memory is needed. Therefore, in some cases, an optimization of the scheduling of the requests is to minimize the amount of overhead by issuing requests that access data from the same row first before issuing requests that access data from other rows.

[0020] There are many ways to optimize the scheduling of the requests by memory controller 100, including reducing latency, increasing overall bandwidth, increasing memory bank parallelism, reducing idle time on the data bus, and the like. Each of the banks may operate independently in certain ways, and memory bank parallelism often improves memory access efficiency. For example, while row decoding is performed on bank 0, column decoding may be performed on another bank simultaneously, thereby reducing idle time on the data bus. The data bus may be kept busy by accessing multiple banks (or multiple bank groups) at the same time. Idle time on the data bus may also be reduced by issuing multiple reads before issuing multiple writes (or vice versa) because there is a wait time associated with the switching from reads to writes (or vice versa) on the data bus. The memory controller may arrange the accesses to be pipelined in such a way that data is fetched from the memory efficiently and all the constraints are met. There are many different types of constraints. For example, it takes a certain amount of time to read a row into the row buffer, and it takes a certain amount of time to read a column. And after the reading is done, pre-charging is needed before another row of the same bank may be fetched into the row buffer again. Therefore, request scheduler 112 acts as a constraint solver.

[0021] Request scheduler 112 monitors the pool of requests and filters the requests based on the priority of scheduling. For example, the highest priority requests may be filtered using a first level of filtering, then the filtered out requests that have the highest priority may be filtered using a second level of filtering, and so on.

[0022] In one particular example, the predetermined priority is to schedule read requests first, then requests that go to an open bank, and then the oldest pending requests. In this example, request scheduler 112 may filter out all the read requests first. The filtered read requests will then be subject to a second arbitration logic. In other words, the read requests to

read from an open bank are filtered out. Next, the older read requests targeted to an open bank will be processed earlier than the newer read requests. And if there are no pending read requests to read from an open bank, then the requests are processed in the order of decreasing age. Since there are many possible combinations of different policies that can be applied to memory controller 100 to achieve optimized memory access for different memory address accessing patterns, a highly flexible scheduling framework that can be easily configured or re-configured would be desirable.

[0023] In the present application, an improved memory request scheduling system is disclosed. The system comprises one or more queues configured to store memory requests waiting to be issued to a memory. The system further comprises a memory request scheduler configured to determine an issuing schedule order of the memory requests. The issuing schedule order of the memory requests is determined using modular configurable memory request priority comparing function units that are connected together in a dynamically programmable evaluation order.

[0024] In the present application, an improved memory request scheduling method is disclosed. The method comprises storing memory requests waiting to be issued to a memory in one or more queues. The method comprises connecting modular configurable memory request priority comparing units together in a dynamically programmable evaluation order. The method further comprises determining, by the modular configurable memory request priority comparing units, an issuing schedule order of the memory requests.

[0025] In the present application, an improved memory request scheduling system is disclosed. The system comprises a processor configured to store memory requests waiting to be issued to a memory in one or more queues. The processor is configured to connect modular configurable memory request priority comparing units together in a dynamically programmable evaluation order. The processor is configured to determine, by the modular configurable memory request priority comparing units, an issuing schedule order of the memory requests. The system comprises a memory coupled to the processor and configured to provide the processor with instructions.

[0026] Figure 3 illustrates an exemplary memory request scheduling system 300. In

some embodiments, memory request scheduling system 300 is at least a part of request scheduler 112 of memory controller 100 in Figure 1. Figure 4 illustrates an exemplary process 400 for scheduling memory requests in a memory system. In some embodiments, process 400 may be performed by at least some of the components in memory controller 100, including request scheduling system 300.

[0027] At 402, memory requests waiting to be issued to a memory are stored in one or more queues. For example, with reference to Figure 1, the inputs received by memory controller I/O handler 102 may include requests received from a host that is connected to the memory controller. Fragment assembler 104 sends the requests 106 to address decoder 108 for processing. Since the memory is divided into different banks, the processed requests are sent by address decoder 108 to a set of bank queues 110. For example, requests that are targeted to bank 0 are sent to bank queue 0 (B0), requests that are targeted to bank 1 are sent to bank queue 1(B1), and so on.

[0028] At 404, modular configurable memory request priority comparing units are connected together in a dynamically programmable evaluation order. As shown in Figure 3, memory request scheduling system 300 reads the memory requests that are stored in a set of queues 302. The set of queues 302 may be a portion of the set of bank queues 110 in memory controller 100. Memory request scheduling system 300 uses a modular and programmable tie-breaker scheduling framework where the priority of selecting a request from the outstanding queue to be issued to the memory is defined by a plurality of modular configurable memory request priority comparing function (CF) units that are connected together in a dynamically programmable evaluation order. As shown in Figure 3, there are three CF units (CF1, CF2, and CF3) connected together. However, any number of CF units may be connected together in a dynamically programmable evaluation order for determining the issuing schedule order of the memory requests. The time axis 303 is shown in Figure 3 as increasing from the top to the bottom. Memory request priority comparing function unit CF1 is utilized a first time at instance 304A and a second time at instance 304B. Similarly, memory request priority comparing function unit CF2 is utilized a first time at instance 306A and a second time at instance 306B. Similarly, memory request priority comparing function unit CF3 is utilized a first time at instance 308A and a second time at instance 308B.

[0029] With reference to Figure 4, at 406, an issuing schedule order of the memory requests is determined by the modular configurable memory request priority comparing units. At 408, the memory requests are issued to the memory in the issuing schedule order. The memory may be DRAM, HBM, or other media. Figure 5 illustrates an exemplary process 500 for determining at least a part of the issuing schedule order of the memory request. In some embodiments, process 500 may be performed multiple times at step 406, each time determining the issuing schedule order of two memory requests at a time.

[0030] At 502, two memory requests are compared based on a first configurable priority comparing function associated with a first modular configurable memory request priority comparing unit to select one memory request as a winner. A priority comparing function unit selects a winner between two contending requests. The parameters, attributes, states, or properties of the memory requests, the memory controller, the memory channel, or the data bus may be used by the priority comparing function to identify which of the contenders is the winner at a given stage.

[0031] At 504, it is determined whether there is a winner. If there is a winner, then process 500 is terminated at 505. If the current priority CF unit cannot select a winner between the two requests, then process 500 proceeds to 506. At 506, a tie-breaker is performed by handing off the two contending requests to the next priority CF unit connected to the current CF unit to select one memory request as a winner. Steps 504 and 506 are repeated until a winner is determined.

[0032] Continuing with the illustrative example above, the CF units are configured to schedule the memory requests by scheduling read requests first, then requests that go to an open bank, and then the oldest pending requests. In other words, CF1 is programmed to implement a priority comparing function that selects and prioritizes a read request over a write request. CF2 is programmed to implement a priority comparing function that selects and prioritizes a request that is targeted to an open bank over a request that is targeted to a memory bank that is not open. And CF3 is programmed to implement a priority comparing function that selects and prioritizes an older request over another request that is newer. Each priority comparing function unit (CF1, CF2, and CF3) is a modular and identical unit that is programmable with a different priority

comparing function. The advantage is that the scheduling based on priority is modularized to simple functions which only compare the priority between two requests in the outstanding queue. The priority comparing function units may be dynamically programmed to be connected and evaluated in a particular order. In this example, the priority comparing function units are connected in a linked list structure where CF1 is evaluated first, and CF2 is evaluated second, and CF3 is evaluated third. However, the linked list may be dynamically programmed to connect the priority comparing function units in a different order in order to implement a different priority policy on the memory controller. For example, CF1 may be dynamically reconfigured to be connected to CF3, and CF3 is reconfigured to be connected to CF2 in order to implement a different priority policy on the memory controller.

[0033] With continued reference to Figure 3, suppose that request 302A and request 302B are two read requests that are being compared by CF1 at instance 304A. Since CF1 cannot select a winner based on its configured priority comparing function that prioritizes read requests, a tie-breaker is performed by handing off request 302A and request 302B to CF2 unit at instance 306A for a second stage of comparison. Suppose that request 302A is a read request targeted to an open bank and request 302B is a read request that is targeted to a bank that is currently not open. Since CF2 is programmed to select a request that is targeted to an open bank, request 302A is selected by CF2 unit to be the winner request 310. Note that at this point, winner request 310 is obtained after two stages of comparison performed by CF1 unit and CF2 unit, and the third stage of comparison by CF3 unit is skipped.

[0034] Each of the priority comparing function unit is programmed to make a decision based on different criteria, such as the attributes or parameters of the candidate memory requests or states of the memory controller, states of the data bus, or memory channel states. For example, a priority comparing function may prioritize a memory request that is going to the currently servicing read or write bus, and the criteria may include a parameter that indicates whether the request is a read/write request and a data bus state that indicates whether the data bus is busy or idle. In another example, a priority comparing function may prioritize a memory request that is going to an open row, and the criteria may include a parameter that indicates the request's row address and a memory row state that indicates whether a memory bank is open or not. In another example, a priority comparing function may prioritize a memory request that has

been queued up for longer than a predetermined time threshold, and the criteria may include an age parameter that indicates how long the request has been stored in the queue. In another example, a priority comparing function may select a memory request that follows the memory banks' round-robin order, and the criteria may include a parameter that indicates the memory bank ID. In another example, a priority comparing function may select a memory request that follows the memory bank groups' round-robin order, and the criteria may include a parameter that indicates the memory bank group ID.

[0035] At any stage, one priority comparing function accepts two requests and makes a decision on the winner based on the rules of the priority comparing function. If the current priority CF unit cannot select a winner between the two requests, a tie-breaker is performed by offloading the two contending requests to the next priority CF unit connected to the current CF unit. A CF unit may be connected to the next one as a linked list to handle the tie-break. Hence, the priority policy can be re-programmed by just switching out one CF for another, or by adding additional CFs in the linked list. There is no overhead to the scheduling logic as this policy also needs to iterate over the outstanding queue.

[0036] With continued reference to Figure 3, after 302A is determined as the winner request 310, request 302A is being compared with another request 302C from the set of queues 302. Suppose that request 302C is also a read request, and request 302A and request 302C are being compared by CF1 at instance 304B. Since CF1 cannot select a winner based on its configured priority comparing function that prioritizes read requests, a tie-breaker is performed by handing off request 302A and request 302C to CF2 unit at instance 306B for a second stage of comparison. Suppose that both request 302A and request 302C are read requests that are targeted to an open bank. Since CF2 cannot select a winner based on its configured priority comparing function that prioritizes requests that are targeted to an open bank, a tie-breaker is performed by handing off request 302A and request 302C to CF3 unit at instance 308B for a third stage of comparison. Suppose that request 302A is an older request than request 302C. Since CF3 is programmed to select a request that is older over another request that is newer, request 302A is selected by CF3 unit to be the winner request 312. Winner request 312 is then compared with another request from the set of queues 302, and a similar process is iterated until the remaining requests in the set of queues 302 have all been compared with the previous winner

request. For example, all eight of the requests in the set of queues 302 may be processed after seven iterations. In the first iteration, a winner is selected from the first two requests. In the second iteration, a winner is selected from the first three requests. In the third iteration, a winner is selected from the first four requests. And after the seventh iteration, a winner is selected among all eight of the requests. Based on these comparisons, an issuing schedule order of the memory requests may be obtained. And the memory requests may be issued to the memory according to the issuing schedule order.

[0037] Since request scheduler 112 uses modular, independent, and programmable CF units to form the highly flexible scheduling framework, the scheduler framework efficiency may be further improved by pipelining the decision-making. Figure 6 illustrates another exemplary memory request scheduling system 600. Memory request scheduling system 600 is similar to memory request scheduling system 300. One difference between the two is that each of the CF units in system 600 may be re-used as soon as it has failed to decide a winner between one pair of memory requests and has handed off the two contending requests to the next tie-breaking priority CF unit.

[0038] As shown in Figure 6, memory request scheduling system 600 reads the memory requests that are stored in a set of queues 602. The set of queues 602 may be a portion of the set of bank queues 110 in memory controller 100. The time axis 603 is shown in Figure 6 as increasing from the top to the bottom. Memory request priority comparing function unit CF1 is utilized a first time at instance 604A and a second time at instance 604B. Similarly, memory request priority comparing function unit CF2 is utilized a first time at instance 606A and a second time at instance 606B. Similarly, memory request priority comparing function unit CF3 is utilized a first time at instance 608A and a second time at instance 608B.

[0039] As soon as CF1 unit has failed to decide a winner between memory requests 602A and 602B at instance 604A and has handed off the two contending requests to the next tie-breaking priority CF2 unit at instance 606A, CF1 unit is reloaded with a new pair of outstanding memory requests (602C and 602D) at instance 604B. The winner between memory requests 602A and 602B is winner request 610. The winner between memory requests 602C and 602D is winner request 612. The scheduling decision of a pair of requests may be set at every “n” cycles,

where “n” is the stage delay per CF unit.

[0040] The advantage of this parallel processing technique is that CF1 unit can begin to determine a winner between a different pair of memory requests while the previous pair of memory requests is still being arbitrated. Figure 7 illustrates another example that parallel processing may be achieved by pipelining the decision-making. In this example, there are five CF units (CF1, CF2, CF3, CF4, and CF5) connected together. At time slot 1, CF1 is used to determine a winner between memory request 1 (R1) and memory request 2 (R2). When there is a tie between R1 and R2, the two requests are passed to CF2 at time slot 2. And when CF2 decides that there is a tie again, R1 and R2 are passed to CF3 at time slot 3, and so on until a winner is finally determined by CF5 at time slot 5. Note that as soon as CF1 unit has failed to decide a winner between memory requests R1 and R2 and has handed off the two contending requests to the next tie-breaking priority CF2 unit, CF1 unit is reloaded with a new pair of outstanding memory requests (R3 and R4) at time slot 2.

[0041] Figure 8 illustrates another exemplary memory request scheduling system 800. In this example, memory request scheduling system 800 uses two different linked lists of memory request priority comparing function units for determining the scheduling order of the memory requests. The time axis 803 is shown in Figure 8 as increasing from the top to the bottom. The first linked list of CF units includes CF1 at instance 806, CF2 at instance 808, and CF3 at instance 812. The winner request is winner request 810. The second linked list of CF units includes CF4 at instance 814, CF5 at instance 816, and CF6 at instance 818. The winner request is winner request 820.

[0042] As shown in Figure 8, CF1 reads the memory requests (e.g., memory requests 802A and 802B) that are stored in a set of queues 802. The set of queues 802 may be a bank queue for a particular rank. CF4 reads the memory requests (e.g., memory requests 804A and 804B) that are stored in a set of queues 804. The set of queues 804 may be a bank queue for a different rank. Since different ranks may have different properties, and different properties may require a different priority policy, two different sets of memory request priority comparing function units may be used for determining the scheduling order of the memory requests for the two different ranks of memory. For example, each of the CF1-CF6 units may have a different

comparing function.

[0043] Figure 9 illustrates another example that two different linked lists of memory request priority comparing function units may be used for determining the scheduling order of the memory requests. In this example, each linked list of CF units includes three CF units that are connected together. The first linked list has CF1, CF2, and CF3 connected together. The second linked list has CF4, CF5, and CF6 connected together.

[0044] At time slot 1, CF1 is used to determine a winner between memory request 1 (R1) and memory request 2 (R2). When there is a tie between R1 and R2, the two requests are passed to CF2 at time slot 2. And when CF2 decides that there is a tie again, R1 and R2 are passed to CF3 at time slot 3, and a winner is finally determined by CF3 at time slot 3. Note that as soon as CF1 unit has failed to decide a winner between memory requests R1 and R2 and has handed off the two contending requests to the next tie-breaking priority CF2 unit, CF1 unit is reloaded with a new pair of outstanding memory requests (R3 and R4) at time slot 2. Therefore, the scheduler framework efficiency is improved by pipelining the decision-making.

[0045] Similarly, at time slot 1, CF4 is used to determine a winner between memory request 11 (R11) and memory request 12 (R12). When there is a tie between R11 and R12, the two requests are passed to CF5 at time slot 2. And when CF5 decides that there is a tie again, R11 and R12 are passed to CF6 at time slot 3, and a winner is finally determined by CF6 at time slot 3. Note that as soon as CF4 unit has failed to decide a winner between memory requests R11 and R12 and has handed off the two contending requests to the next tie-breaking priority CF5 unit, CF4 unit is reloaded with a new pair of outstanding memory requests (R13 and R14) at time slot 2.

[0046] Although the foregoing embodiments have been described in some detail for purposes of clarity of understanding, the disclosure is not limited to the details provided. There are many alternative ways of implementing the disclosure. The disclosed embodiments are illustrative and not restrictive.

[0047] WHAT IS CLAIMED IS:

CLAIMS

1. A system, comprising:

one or more queues configured to store memory requests waiting to be issued to a memory; and

a memory request scheduler comprising modular configurable memory request priority comparing units connected together in a dynamically programmable evaluation order, wherein the memory request scheduler is configured to determine an issuing schedule order of the memory requests.

2. The system of claim 1, wherein a first modular configurable memory request priority comparing unit is configured to:

compare two of the memory requests based on a first configurable priority comparing function associated with the first modular configurable memory request priority comparing unit, wherein the first configurable priority comparing function compares the two of the memory requests to select one memory request between the two of the memory requests.

3. The system of claim 2, wherein the first configurable priority comparing function compares the two of the memory requests based on at least a memory request attribute.

4. The system of claim 2, wherein the first configurable priority comparing function compares the two of the memory requests based on at least a state of a data bus.

5. The system of claim 2, wherein the first configurable priority comparing function compares the two of the memory requests based on at least a memory state.

6. The system of claim 1, wherein a first modular configurable memory request priority comparing unit is configured to:

compare two of the memory requests based on a first configurable priority comparing function associated with the first modular configurable memory request priority comparing unit; and

in response to not being able to select one memory request between the two of the memory requests based on the first configurable priority comparing function, pass the two of the memory requests to a tie-breaking second modular configurable memory request priority comparing unit that is connected next to the first modular configurable memory request priority

comparing unit according to the dynamically programmable evaluation order.

7. The system of claim 6, wherein a second configurable priority comparing function associated with the tie-breaking second modular configurable memory request priority comparing unit is different from the first configurable priority comparing function associated with the first modular configurable memory request priority comparing unit.

8. The system of claim 7, wherein the tie-breaking second modular configurable memory request priority comparing unit is configured to:

compare the two of the memory requests based on the second configurable priority comparing function associated with the tie-breaking second modular configurable memory request priority comparing unit, wherein the second configurable priority comparing function compares the two of the memory requests to select one memory request between the two of the memory requests.

9. The system of claim 6, wherein the first modular configurable memory request priority comparing unit is configured to:

after the two of the memory requests are passed to the tie-breaking second modular configurable memory request priority comparing unit, compare another two memory requests based on the first configurable priority comparing function associated with the first modular configurable memory request priority comparing unit, wherein the first configurable priority comparing function compares the another two memory requests to select one memory request between the another two memory requests.

10. The system of claim 1, wherein the one or more queues comprise one or more memory bank queues, wherein one of the one or more memory bank queues comprises a queue for memory requests targeted to a memory bank of the memory.

11. A method, comprising:

storing memory requests waiting to be issued to a memory in one or more queues;

connecting modular configurable memory request priority comparing units together in a dynamically programmable evaluation order; and

determining, by the modular configurable memory request priority comparing units, an issuing schedule order of the memory requests.

12. The method of claim 11, comprising:

comparing two of the memory requests based on a first configurable priority comparing function associated with a first modular configurable memory request priority comparing unit, wherein the first configurable priority comparing function compares the two of the memory requests to select one memory request between the two of the memory requests.

13. The method of claim 11, further comprising:

comparing two of the memory requests based on a first configurable priority comparing function associated with the first modular configurable memory request priority comparing unit; and

in response to not being able to select one memory request between the two of the memory requests based on the first configurable priority comparing function, passing the two of the memory requests from the first modular configurable memory request priority comparing unit to a tie-breaking second modular configurable memory request priority comparing unit that is connected next to the first modular configurable memory request priority comparing unit according to the dynamically programmable evaluation order.

14. The method of claim 13, wherein a second configurable priority comparing function associated with the tie-breaking second modular configurable memory request priority comparing unit is different from the first configurable priority comparing function associated with the first modular configurable memory request priority comparing unit.

15. The method of claim 14, further comprising:

comparing the two of the memory requests based on a second configurable priority comparing function associated with the tie-breaking second modular configurable memory request priority comparing unit, wherein the second configurable priority comparing function compares the two of the memory requests to select one memory request between the two of the memory requests.

16. The method of claim 13, further comprising:

after the two of the memory requests are passed to the tie-breaking second modular configurable memory request priority comparing unit, comparing another two memory requests based on the first configurable priority comparing function associated with the first modular configurable memory request priority comparing unit, wherein the first configurable priority

comparing function compares the another two memory requests to select one memory request between the another two memory requests.

17. A system, comprising:

a processor configured to:

store memory requests waiting to be issued to a memory in one or more queues;

connect modular configurable memory request priority comparing units together in a dynamically programmable evaluation order; and

determine, by the modular configurable memory request priority comparing units, an issuing schedule order of the memory requests; and

a memory coupled to the processor and configured to provide the processor with instructions.

18. The system of claim 17, wherein the processor is further configured to:

compare two of the memory requests based on a first configurable priority comparing function associated with a first modular configurable memory request priority comparing unit, wherein the first configurable priority comparing function compares the two of the memory requests to select one memory request between the two of the memory requests.

19. The system of claim 18, wherein the processor is further configured to:

in response to not being able to select one memory request between the two of the memory requests based on the first configurable priority comparing function, pass the two of the memory requests from the first modular configurable memory request priority comparing unit to a tie-breaking second modular configurable memory request priority comparing unit that is connected next to the first modular configurable memory request priority comparing unit according to the dynamically programmable evaluation order.

20. The system of claim 19, wherein a second configurable priority comparing function associated with the tie-breaking second modular configurable memory request priority comparing unit is different from the first configurable priority comparing function associated with the first modular configurable memory request priority comparing unit.

ABSTRACT OF THE DISCLOSURE

An improved memory request scheduling system is disclosed. The system comprises one or more queues configured to store memory requests waiting to be issued to a memory. The system further comprises a memory request scheduler configured to determine an issuing schedule order of the memory requests. The issuing schedule order of the memory requests is determined using modular configurable memory request priority comparing function units that are connected together in a dynamically programmable evaluation order.

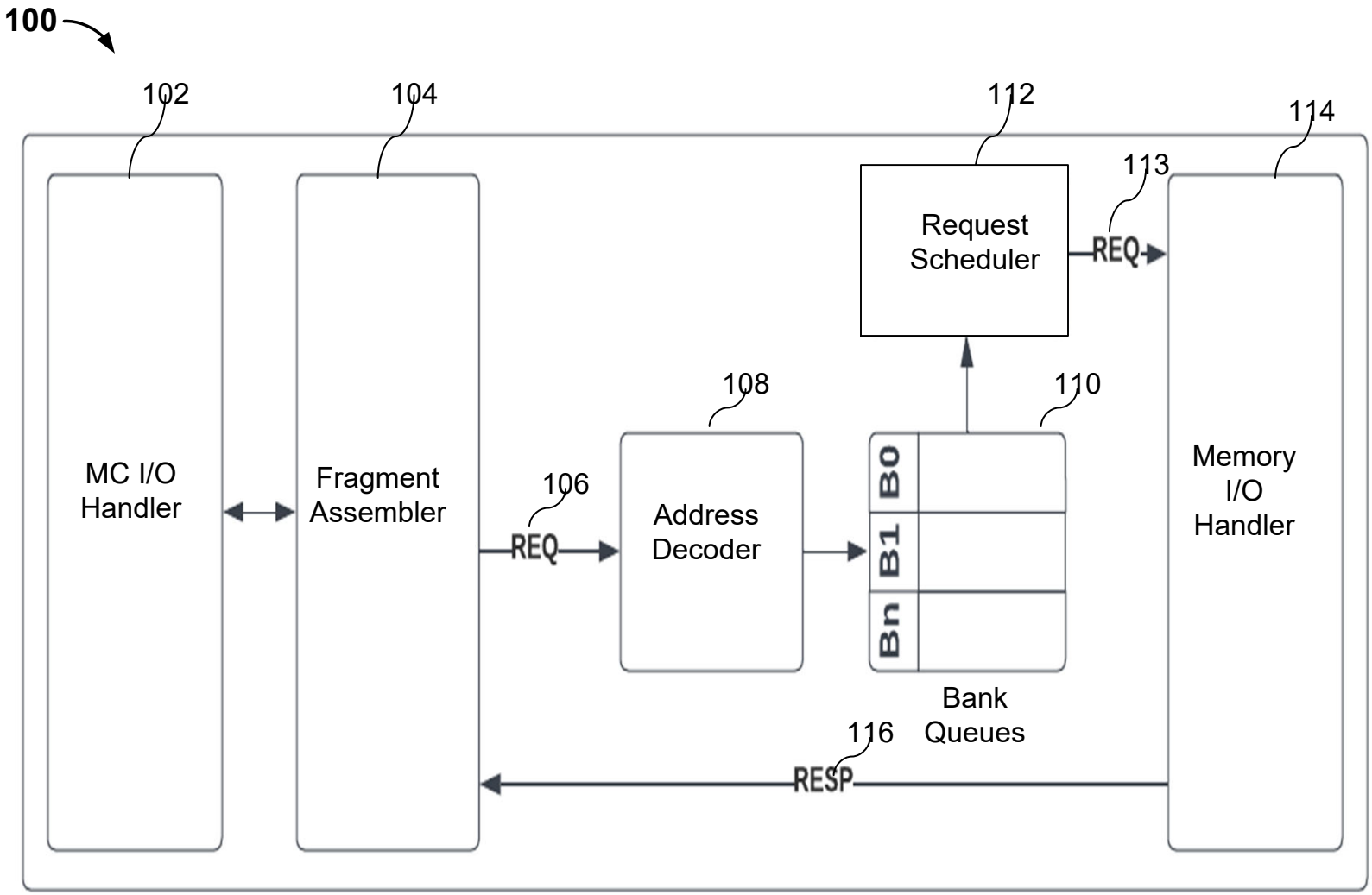
100
102
104
112
114
113
Request Scheduler
REQ
MC I/O Handler
Fragment Assembler
106
REQ
108
Address Decoder
110
B0
B1
Bn
Bank Queues
Memory I/O Handler
116
RESP

FIG. 1

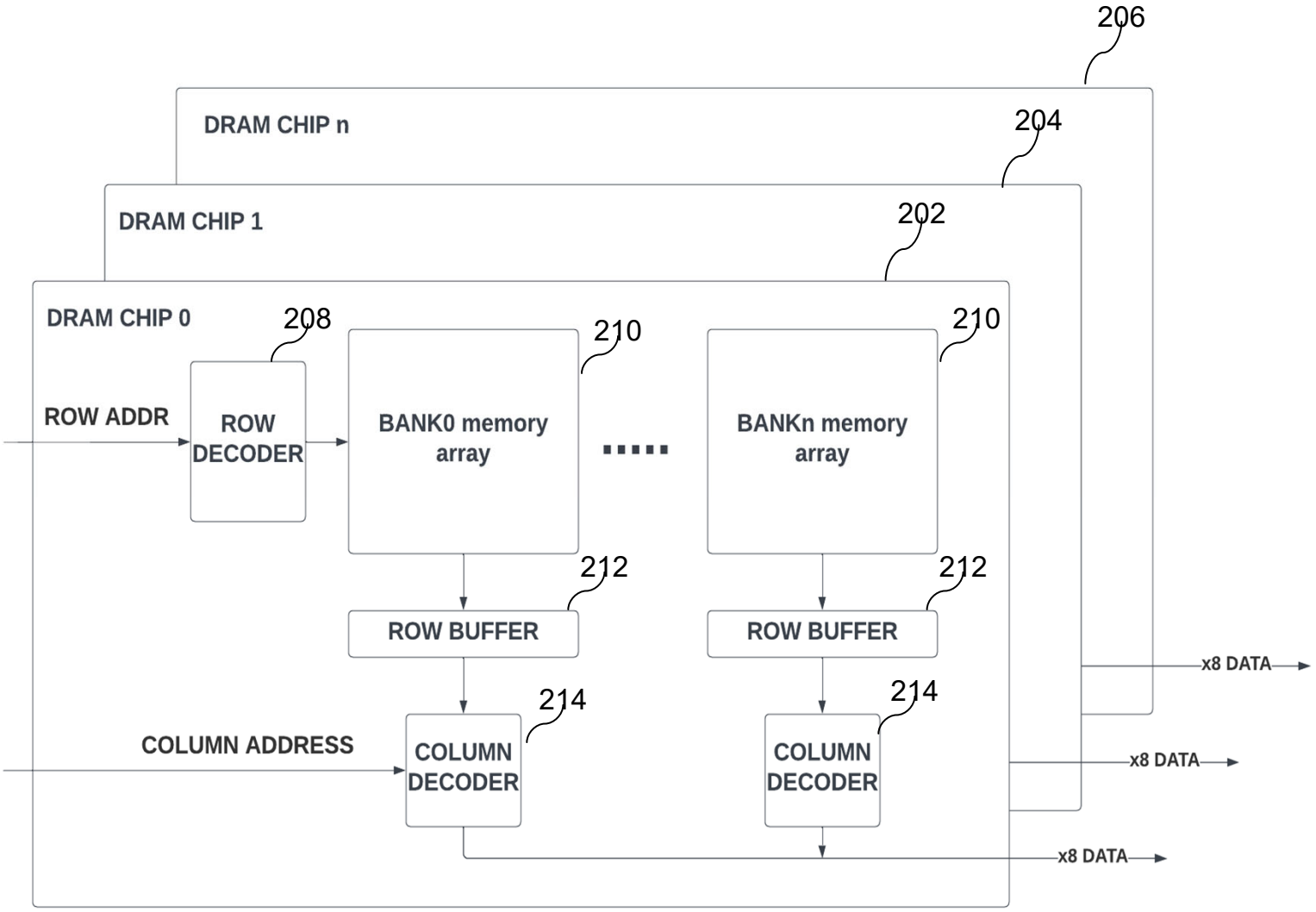
206
DRAM CHIP n
204
DRAM CHIP 1
202
DRAM CHIP 0
208
210
210
ROW ADDR
ROW DECODER
BANK0 memory array
BANKn memory array
212
212
ROW BUFFER
ROW BUFFER
x8 DATA
214
214
COLUMN ADDRESS
COLUMN DECODER
COLUMN DECODER
x8 DATA
x8 DATA

FIG. 2

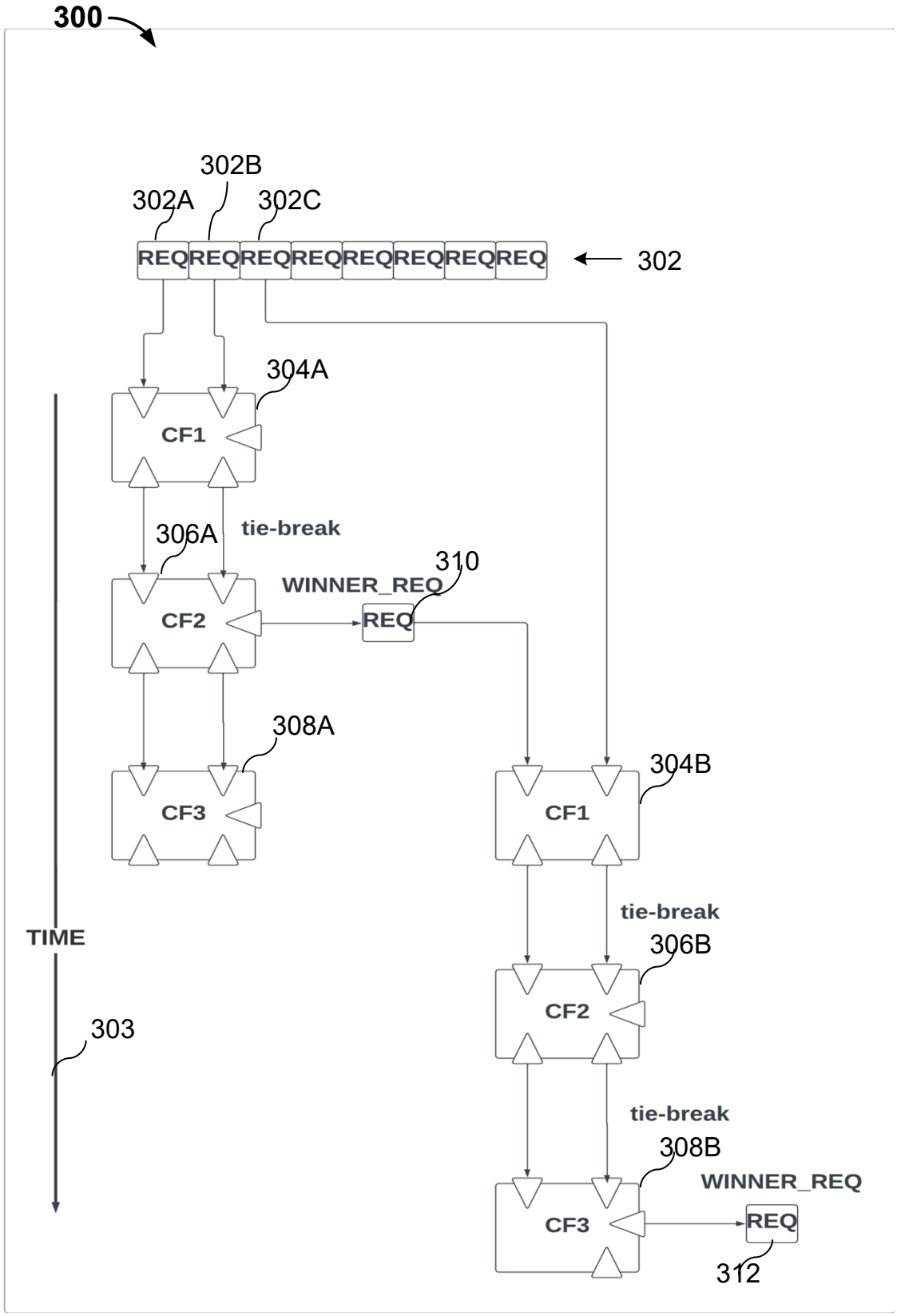
300
302B
302A
302C
REQ REQ REQ REQ REQ REQ REQ REQ
302
304A
CF1
306A
tie-break
310
WINNER_REQ
CF2
REQ
308A
CF3
304B
CF1
tie-break
306B
CF2
TIME
303
tie-break
308B
WINNER_REQ
CF3
REQ
312

FIG. 3

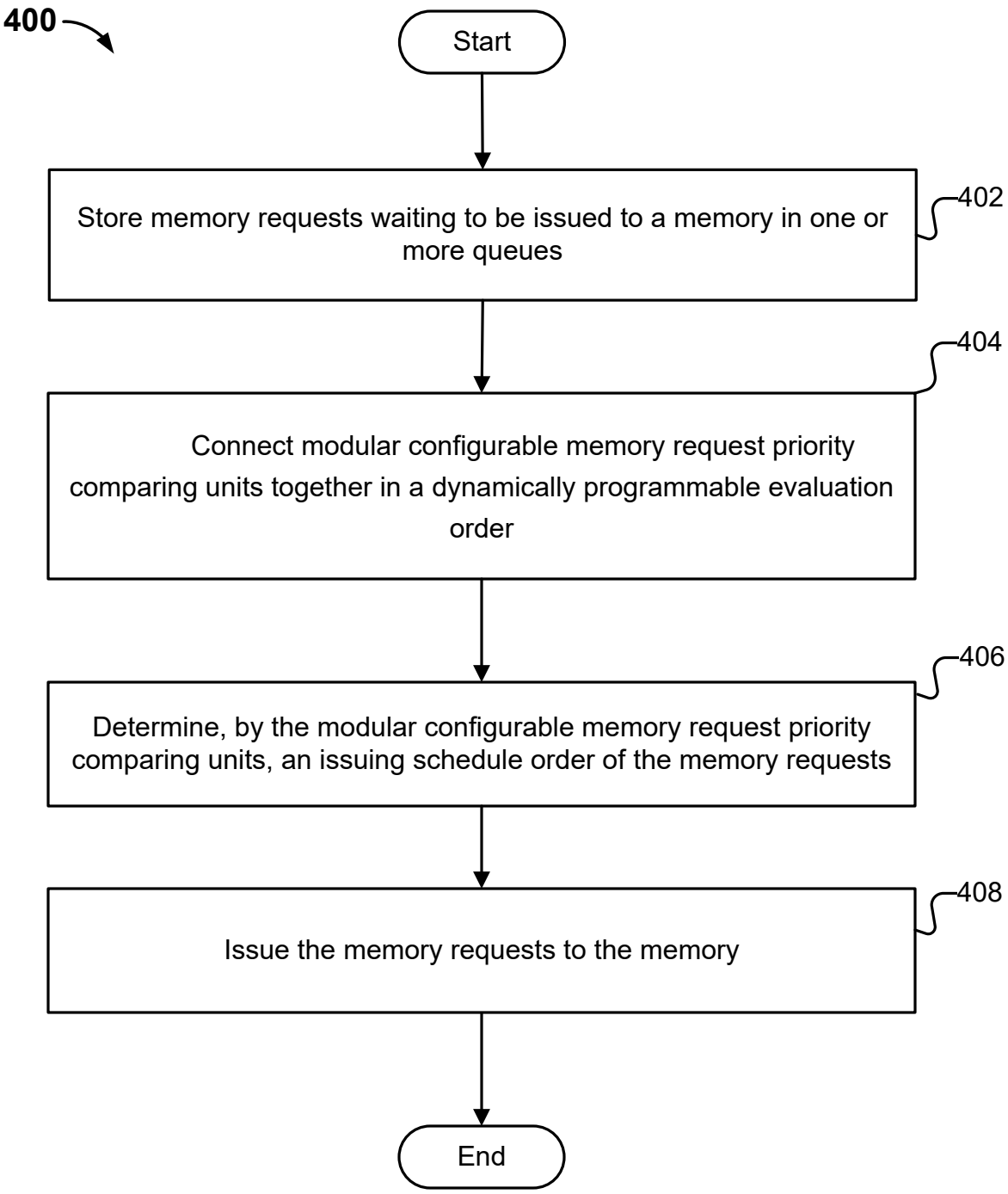
400
Start
Store memory requests waiting to be issued to a memory in one or more queues
402
404
Connect modular configurable memory request priority comparing units together in a dynamically programmable evaluation order
406
Determine, by the modular configurable memory request priority comparing units, an issuing schedule order of the memory requests
408
Issue the memory requests to the memory
End

FIG. 4

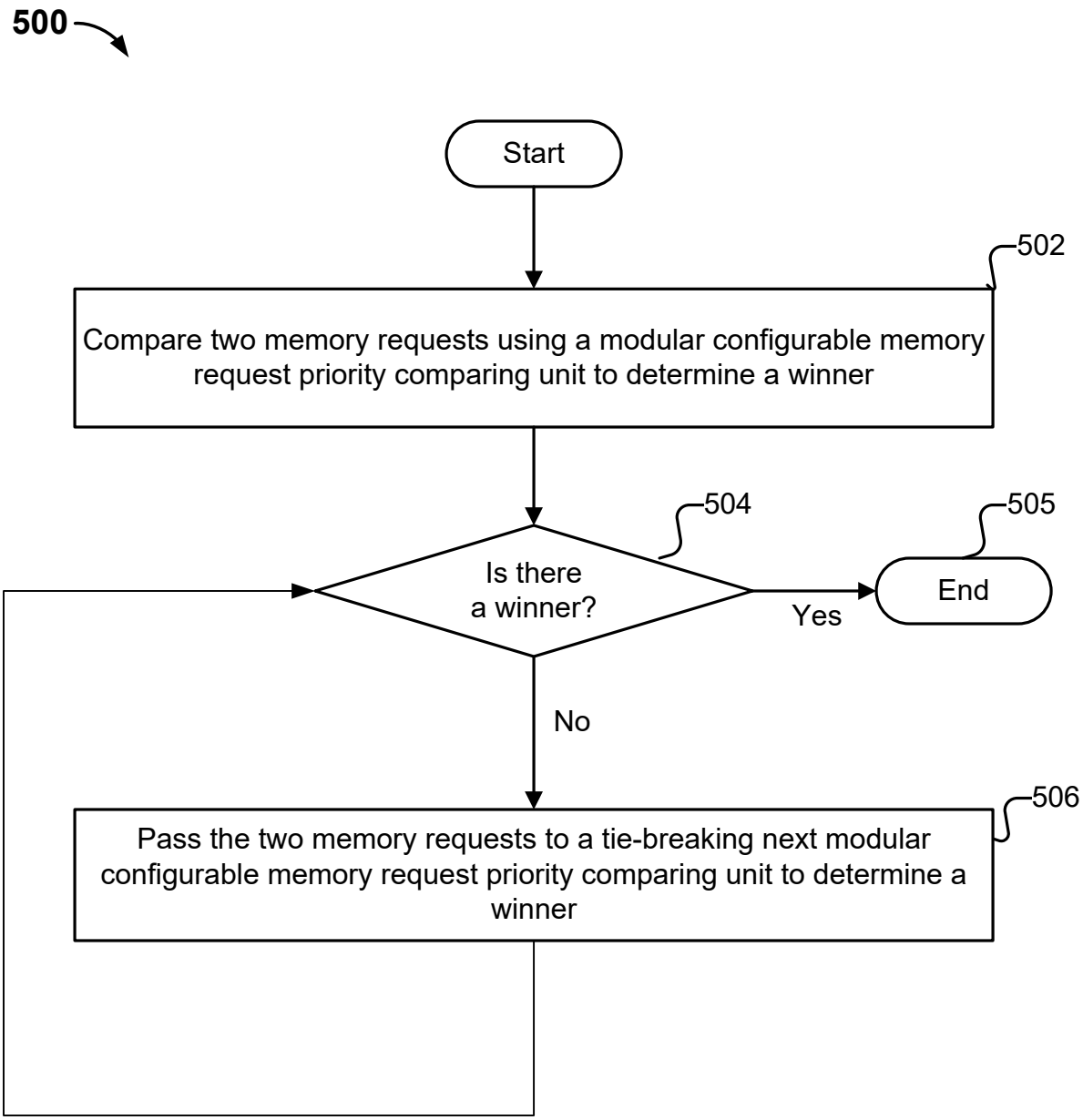
500
Start
502
Compare two memory requests using a modular configurable memory request priority comparing unit to determine a winner
504
Is there a winner?
505
End
Yes
No
506
Pass the two memory requests to a tie-breaking next modular configurable memory request priority comparing unit to determine a winner

FIG. 5

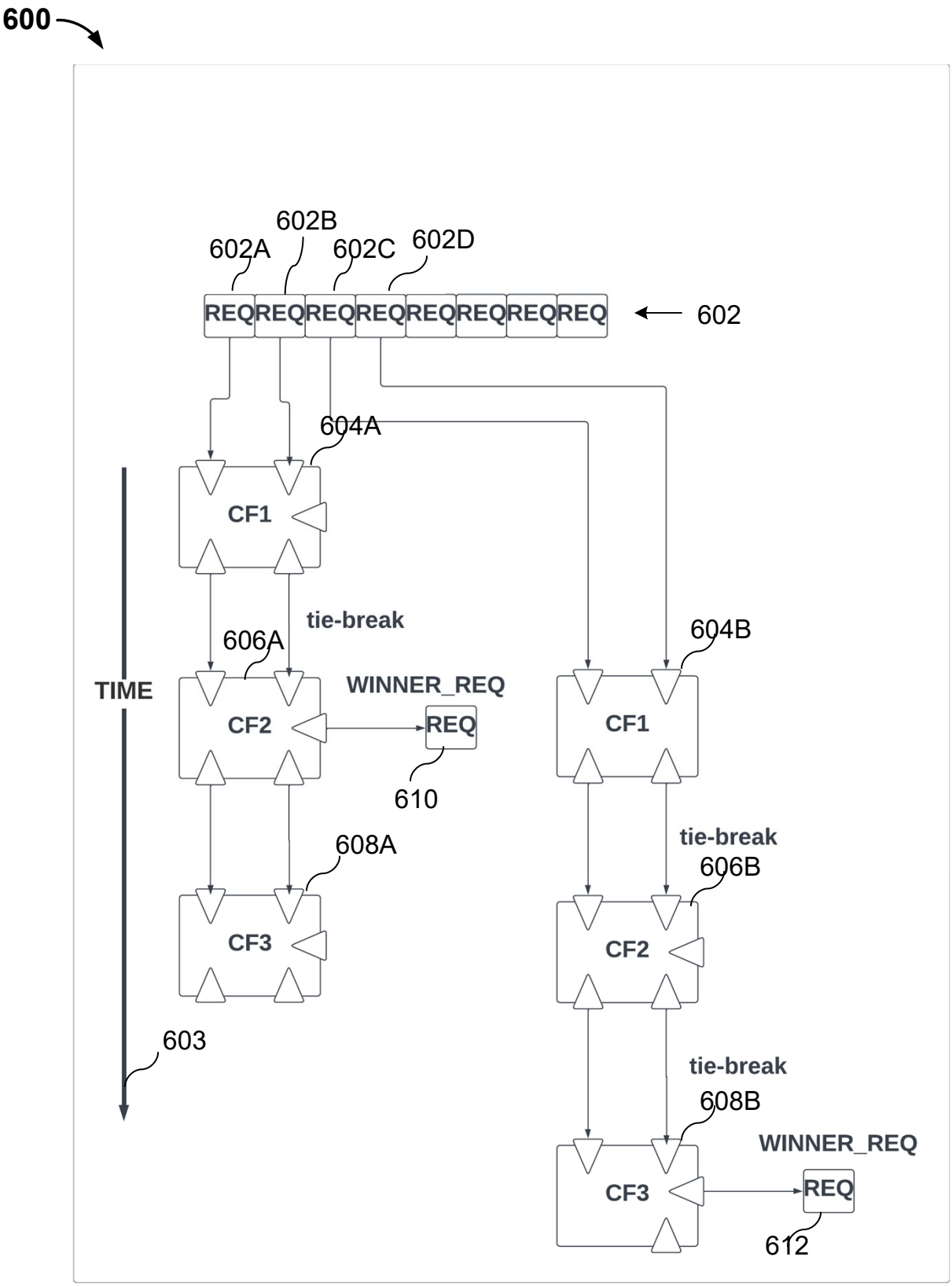
600
602A
602B
602C
602D
REQ
REQ
REQ
REQ
REQ
REQ
REQ
REQ
602
604A
CF1
tie-break
606A
604B
TIME
WINNER_REQ
CF2
REQ
CF1
610
608A
tie-break
606B
CF3
CF2
603
tie-break
608B
WINNER_REQ
CF3
REQ
612

FIG. 6

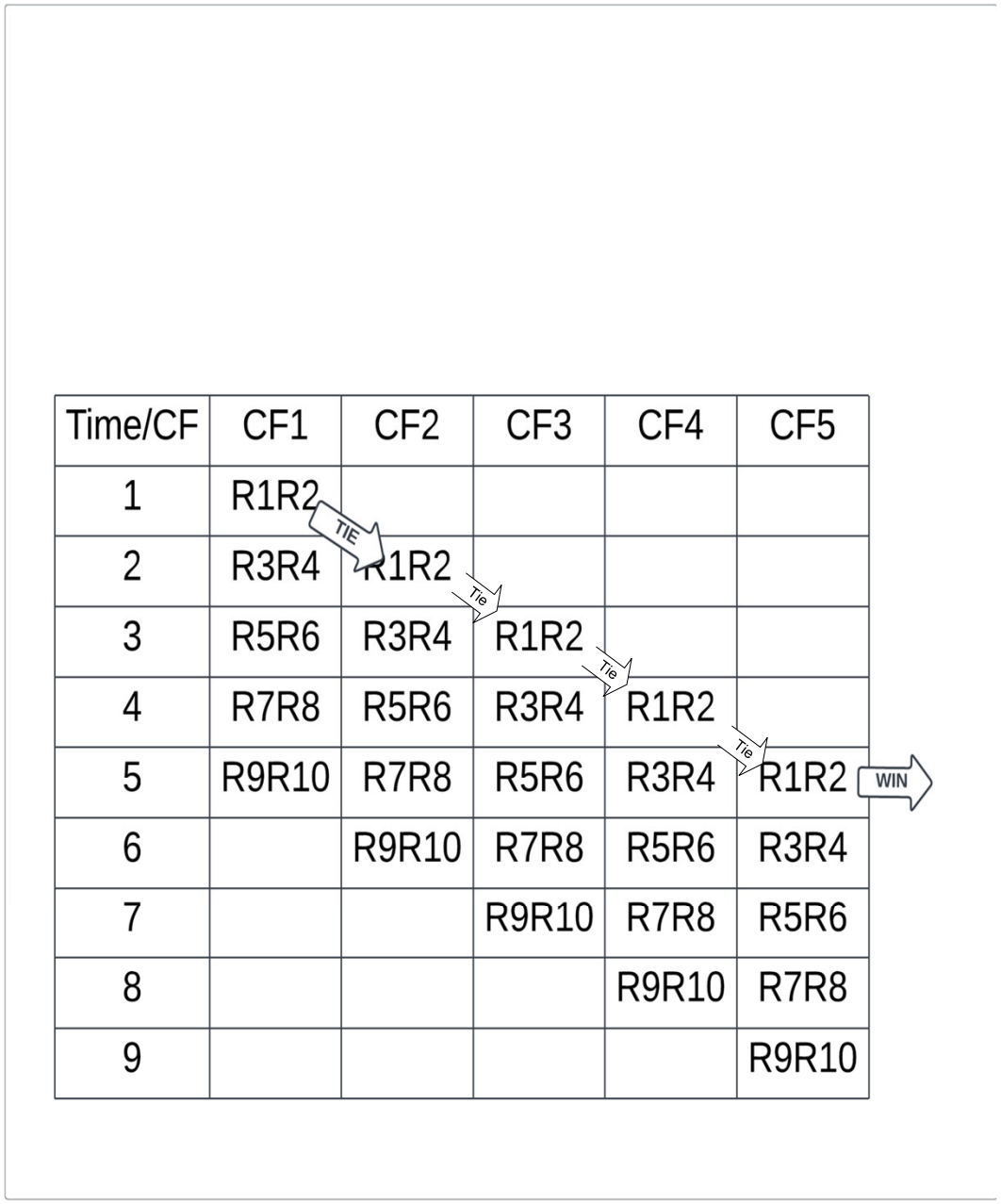
Time/CF
CF1
CF2
CF3
CF4
CF5
1
R1R2
2
R3R4
R1R2
3
R5R6
R3R4
R1R2
4
R7R8
R5R6
R3R4
R1R2
5
R9R10
R7R8
R5R6
R3R4
R1R2
6
R9R10
R7R8
R5R6
R3R4
7
R9R10
R7R8
R5R6
8
R9R10
R7R8
9
R9R10
TIE
Tie
Tie
Tie
WIN

FIG. 7

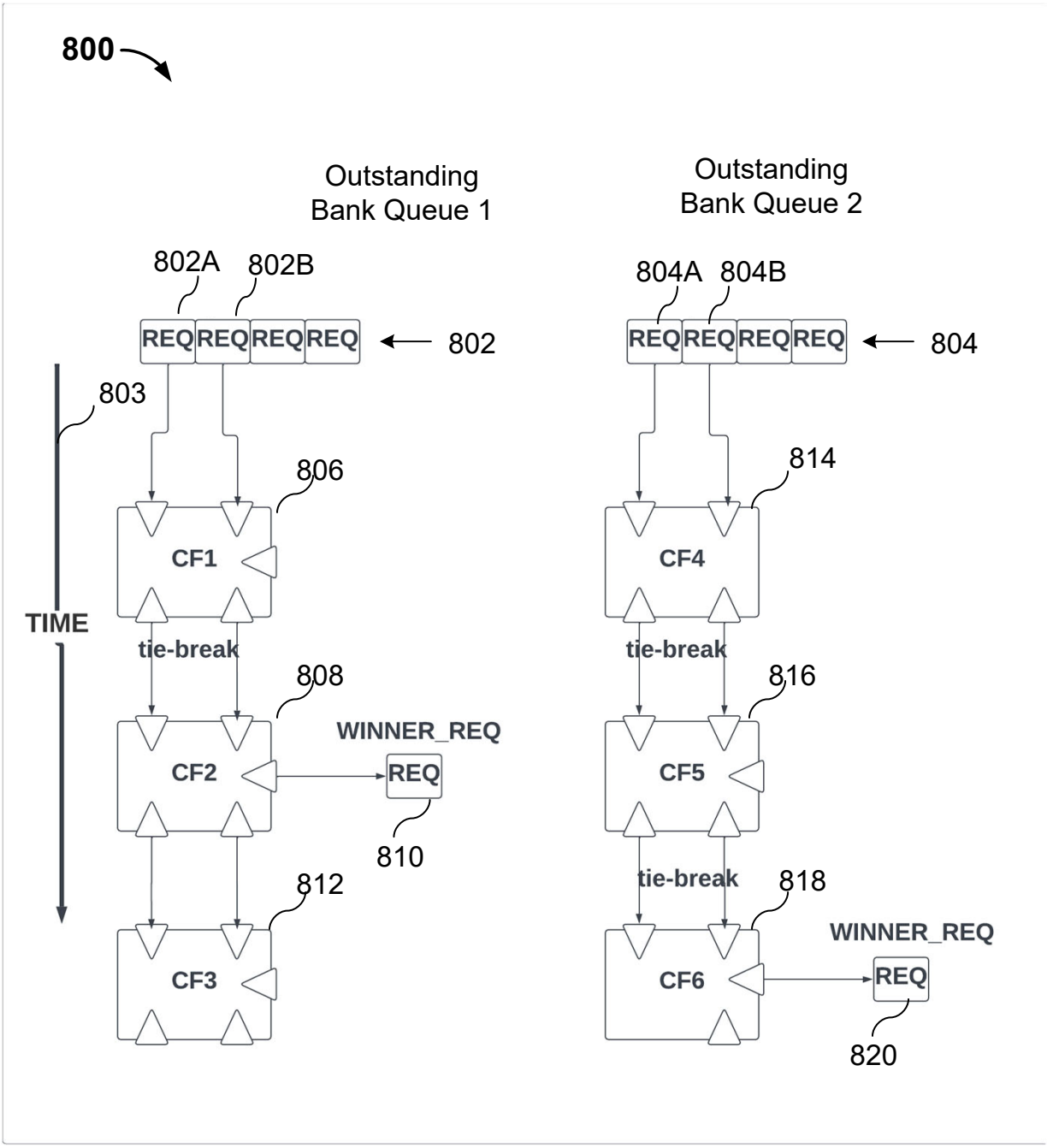
800
Outstanding
Bank Queue 1
Outstanding
Bank Queue 2
802A
802B
804A
804B
REQ
REQ
REQ
REQ
802
804
803
TIME
806
814
CF1
CF4
tie-break
tie-break
808
816
WINNER_REQ
CF2
CF5
REQ
810
tie-break
818
812
WINNER_REQ
CF3
CF6
REQ
820

FIG. 8

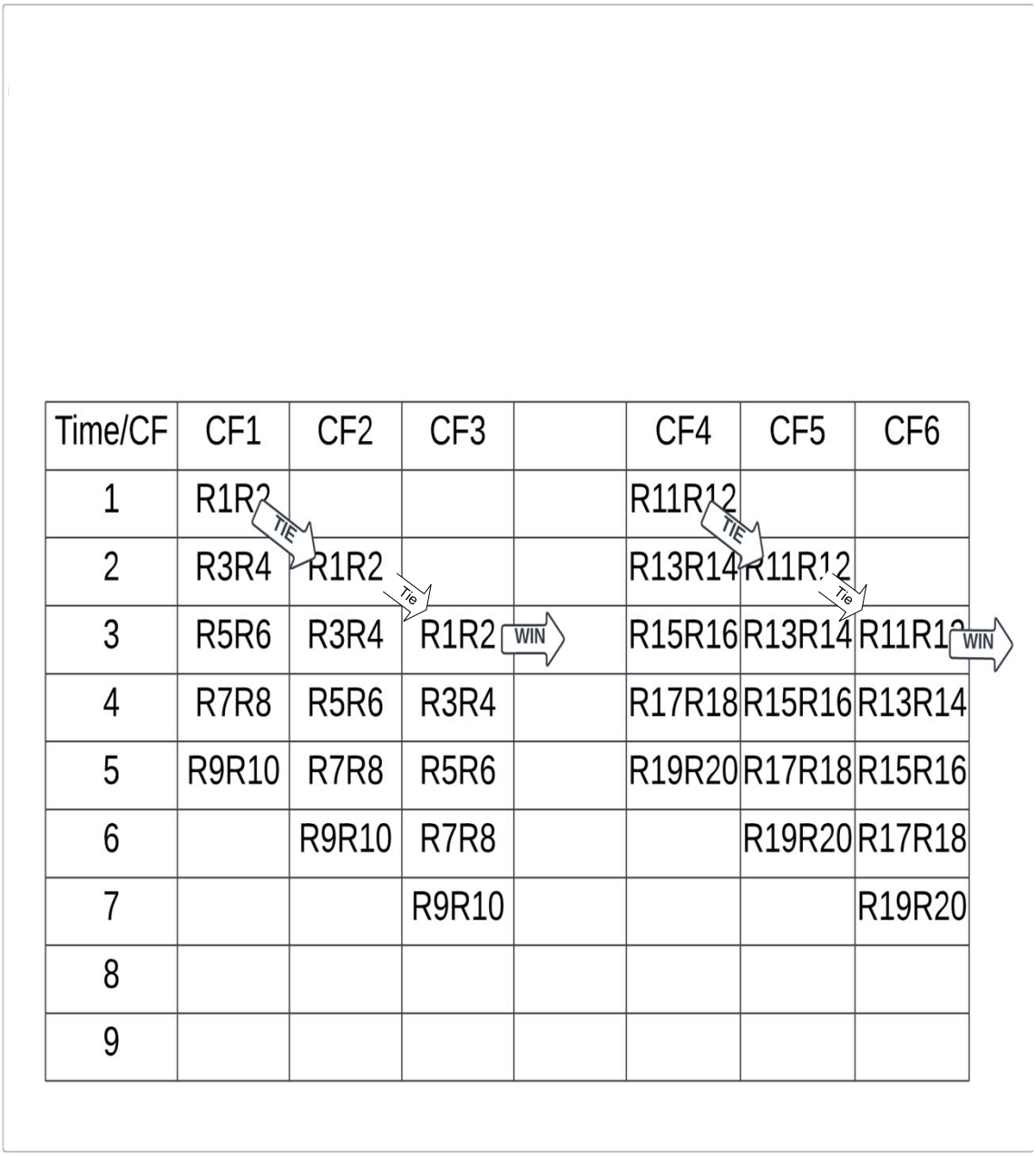
Time/CF
CF1
CF2
CF3
CF4
CF5
CF6
1
R1R2
R11R12
2
R3R4
R1R2
R13R14
R11R12
3
R5R6
R3R4
R1R2
R15R16
R13R14
R11R12
4
R7R8
R5R6
R3R4
R17R18
R15R16
R13R14
5
R9R10
R7R8
R5R6
R19R20
R17R18
R15R16
6
R9R10
R7R8
R19R20
R17R18
7
R9R10
R19R20
8
9
TIE
TIE
Tie
Tie
WIN
WIN

FIG. 9