TITLE: Dynamic Hybridized Positional Notation Instruction Set Computer Architecture to Enhance Security

INVENTORS: Donald Telesca, Bertrand Cambou, Paul Flikkema

CROSS REFERENCE TO RELATED APPLICATIONS AND PUBLICATIONS

This is the original provisional patent application. All references mentioned in this application are herein incorporated by reference without disclaimer.

FIELD OF THE INVENTION

The invention relates to computer architectures and related methods. More particularly, the invention relates to implementations of new computer architectures relating to security systems.

BACKGROUND OF THE INVENTION

Numbers in various bases look different on the surface, but in actuality all numbering systems are like. Consider the decimal notation of 19 for the following expression: $1 \times 10^1 = 9 \times 10^0$. The binary notation for 19 is 10011 which is understood to mean: $(1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 19$. The ternary notation for 19 is 201 which is understood as: $(2 \times 3^2) + (0 \times 3^1) + (1 \times 3^0) = 19$. The general formula to represent a base 10 numeral in any positional notation is defined as $\ldots d_3r^3 + d_2r^2 + d_1r^1 + d_0r^0 \ldots$ wherein d is a coefficient, r is the base of the number system (e.g., r=2 for binary, r=3 for ternary), and the exponent is the position of the digit.

Computers based on binary architectures are simply a convention that is the result of general scientific inquiry and available technological breakthroughs between the 1700's and the 1930's. Computer architectures beyond binary have been limited to ternary systems. The ternary system is one that uses positional notation with a base of 3, operating with trits instead of bits. In terms of computing, base 3 has a genuine mathematical advantage over other numbering systems as it is the most efficient of all integer bases and offers the most economical way of representing numbers. Recent technological breakthroughs have resulted in multi-state electronic devices that can be used as the building blocks for future, unrealized native ternary computing systems.

Existing multilevel NAND flash technologies are based on multiple threshold voltages to store ternary or quaternary bits. This technology is thereby able to store more information per chip, reducing the cost per stored gigabit. The state machines convert the ternary or quaternary information into binary information, making the flash devices appear as binary memories.

Existing dynamic random-access memories (DRAM) have a design that is also naturally ternary. As Q electric charges are trapped in a cell to store a "1" state, there is no charge to store a "0" state, and Q/2 charges are stored for the reference cells. As the charges of the cells within the memory array slowly leak over time, the sensing element during "read" mode compares the remaining charges to the charge left in the reference cell. The design of a ternary DRAM device is based on trapping 0, Q/2, or Q charges on the arrays as well as Q/4 and 3Q/4 on the reference cells. Such a design has lower design margins and needs to be compensated by a more accurate sensing element.

The concept of ternary memristors and resistive random-access memories (ReRAMs) may also be used to increase the bit capacity of the memory arrays. One of the methods creates three different levels of resistivity during a programming cycle. This is done by adding more filaments between the electrodes of each cell or by enlarging the cross-section of the filaments. The three states are then defined by three resistances, typically in the 1KΩ range for the "-1" state, 10KΩ for the "0", and 10MΩ for the "+1" state. In fact, a ten-state system has been demonstrated in hafnium oxide memristor devices.

Native ternary memories have been suggested with MRAM. The giant magnetoresistive (GMR) structure uses a stack of two antiferromagnetic layers with two tunnel oxides sandwiching a ferromagnetic layer and a self-reference design. The native ternary states have three different levels of resistivity. In addition, a ternary memory made from carbon nanotubes has been shown to be effective.

In binary logic, the most fundamental logic functions are "AND", "OR", and "NOT", where the corresponding functions in multi-valued logic (MVL) are called "MIN", "MAX", and "NOT". These binary logic gates can only be implemented in one manner, whereas MVL is capable of describing logic operations in a plurality of ways. A MIN function gives the minimum value of the input signal x. Where $x \in \{0, 1, 2, \ldots, r-1\}$, r is the radix of the signal. A MAX function gives the maximum value of the input signal x. It can be proven that the binary "AND" function is actually a "MIN" function, and the binary "OR" is a "MAX" function. The "NOT" function corresponds to the binary inverter. MVL logic introduces a vast array of new functions, but all will not necessarily be useful.

Morphing operation codes have been used to modify the internal instruction set architecture (ISA) between generations while supporting the broad range of legacy x86 software available. A code morphing manager has been described to provide a method and system for converting software from a source version to a target version. In one embodiment, a plurality of code morphers may be associated based on the input version and the output version of each code morpher. A sequence of code morphers may be determined based on the source version and the target version, and then applied to the software.

Dynamic code generation is the creation of executable code at runtime. Such "on-the-fly" code generation is a powerful technique, enabling applications to use runtime information to improve performance by up to an order of magnitude. The unit of compilation for traditional just-in-time compilers can provide improvement in performance. A trace-based compilation, in which the unit of compilation is a loop, potentially spans multiple methods and even library codes.

SUMMARY OF THE INVENTION

[ADD SUMMARY OF THE CLAIMS HERE]

BRIEF DESCRIPTION OF THE DRAWINGS

The present disclosure is illustrated by way of examples, embodiments and the like and is not limited by the accompanying figures, in which like reference numbers indicate similar elements. Elements in the figures are illustrated for simplicity and clarity and have not necessarily been drawn to scale. The figures along with the detailed description are incorporated and form part of the specification and serve to further illustrate examples, embodiments and the like, and explain various principles and advantages, in accordance with the present disclosure, where:

FIG. 1 depicts a multi-compiler for a set of operating systems (OS) 1 to N, according to one embodiment.

FIG. 2 shows an analysis of a data stream aimed at a particular computing unit 1 to N, according to one embodiment.

FIG. 3 depicts an analysis of a data stream comprised of several segments aimed at computing units 1 to N, according to one embodiment.

FIG. 4 depicts an analysis of a data stream aimed at a particular computing unit 1 to N with a single number system, according to one embodiment.

FIG. 5 depicts an analysis of a data stream comprised of several segments aimed at computing units 1 to N with a single number system, according to one embodiment.

FIG. 6 is an example of a generation of a stream of Operating Systems (OS) with a Random Number and multi-factor authentication, according to one embodiment.

FIG. 7 shows an analysis of a data stream comprised of several segments with multi-factor authentication, according to one embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The described features, advantages, and characteristics may be combined in any suitable manner in one or more embodiments. One skilled in the relevant art will recognize that the circuit may be practiced without one or more of the specific features or advantages of a particular embodiment. In other instances, additional features and advantages may be recognized in certain embodiments that may not be present in all embodiments.

Reference throughout this specification to "one embodiment," "an embodiment," or similar language means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment. Thus appearances of the phrase "in one embodiment," "in an embodiment," and similar language throughout this specification may, but do not necessarily, all refer to the same embodiment.

The paradigm of computing that relies on general purpose computing architectures is being revisited due to the increase of highly specialized computing problems. For example, neuromorphic computing is considered to be superior for optimizing classification schemes, but it does not bring the same advanced processing to all general-purpose computing needs. This diversity and complexity of computing problems is driving the need for heterogeneous hardware platforms that utilize multiple, highly specialized, innovative computer architectures. In this new paradigm disclosed herein, hardware advances should no longer be evaluated on their general-purpose advantages, but rather should be evaluated on their utility for addressing specific computing problems and interoperability in a diverse computing ecosystem.

The concept of a computing system consisting of several separate computing units, identified from 1 to N integer numbers, each computing unit capable of executing software codes written with the corresponding 1 to N positional notation instruction set, is a novel approach. This computing system will deliver increased information assurance from the plurality of operating configurations available. Similar to other abstracted coding languages above the machine language, high-level design of the operating system codes is accomplished at an abstract level. The developer is not required to have knowledge of the hardware requirements of each executing computing unit. A selecting scheme is used is to select the compiler or operating system (OS) for code compilation, and the router sends the code to the appropriate complier unit. **Error! Reference source not found.** depicts the input of a high-level design code into the router unit which has selected OS 3 for compilation. The path of the code (shown by solid black arrows) shows the transition from the higher abstraction level, into the router, to the appropriate compiler for OS 3, and resulting output code.

The multi-compiler architecture shown in **Error! Reference source not found.** is then used in a computing system that can select and execute operating system instruction sets 1 to N based on positional notation systems 1 to N. **Error! Reference source not found.** shows how the router unit is used to direct an incoming data stream to the necessary computing unit; the data stream will only run on computing unit 3 which utilizes OS 3 and has a bi-directional communication link with the appropriate Memory Unit. In this manner, the corresponding computing unit processes the data streams with the required OS 1 to N and the appropriate data stream generated from the resulting input data stream.

The computing system is then capable of inputting the data streams comprised of several segments that require different computing units. The router directs each data stream to the required computing unit 1 to N. Each Computing unit then processes the corresponding data stream for that OS 1 to N. The data streams generated from the multiple operating systems are then combined to form the resulting final data stream.

As shown in FIG. 3, the router directs a data stream comprised of code for unit 1, unit 2, and unit N into each of the respective computing units 1, 2, and N. In this example the computing units 1, 2 and N process the portions of data stream that are respectively associated with OS 1, OS 2, and OS N. The individual data streams generated for OS 1, OS 2, and OS N are combined to form the resulting data stream.

In the implementations described above, the computing system is capable of operating exclusively in any OS 1 to N or analyzing data streams with suitable for computing units 1 to N. In addition, the computing system is capable of operating in a hybridized manner, as described in **Error! Reference source not found.**. The degree of hybridization can be set to variable ratios. As such, P_n is defined as a variable representing either the available compiling units described in **Error! Reference source not found.** or the different data streams described in **Error! Reference source not found.**. In one embodiment, the configuration includes 20% P_1, 40% P_2 and 40% P_n. A generalized equation describing the configuration state, referred to as C_n, is shown as Equation 1 wherein n is constrained by the largest available positional notation for use, P_r maps to a positional notation (e.g., binary, ternary, quaternary), and α_r is a chosen coefficient to control the distribution of hybridization across the used positional notations within the constraints of the equation.

$$C_n = \sum_{r=1}^{n}(\alpha_r P_r, 0 \leq \alpha_r \leq 1) \text{ and } \sum (\alpha_r = 1) \qquad \text{(Equation 1)}$$

The system described in **Error! Reference source not found.** may also be used to implement a plurality of instruction sets within a single positional notation. For example, the system can be configured to use a ternary positional notation with different computing units set to use different ternary instruction sets. More specifically, the system could have 9 different ternary units with 9 different instruction sets. The difficulty to define the hardware/software architecture and handshake as expressed above would be lower with one base of number system.

For dynamical hybridization, the router unit will additionally be capable of temporal control of the distribution of configuration hybridization, allowing for the dynamic switching of the configuration hybridization. The temporal component is integrated into the configuration state model in Equation 1 through the α_r coefficients; Equation 1 then modified to Equation 2 below.

$$C(t)_n = \sum_{r=1}^{n}(\alpha(t)_r P_r, 0 \leq \alpha(t)_r \leq 1) \text{ and } \sum (\alpha(t)_r = 1) \text{ (Equation 2)}$$

This embodiment has a T_0 configuration that comprises 20% P_1 instructions, 40% P_2 instructions, and 40% P_n instructions. The temporal control router unit can be set for any recurrent time scale Δt to then switch to a hybridization of 30% P_1 instructions, 20% P_2 instructions, and 50% P_n instructions at a time equal to $T_0+\Delta t$. At $T_0+2\Delta t$, the configuration can be switched to a third hybridization and so on.

The present invention may be applied to advanced multi-state electronics including microelectronics. The different positional notations are supported by the requisite hardware that is integrated into the full computing platform. For example, in order to execute and store ternary logic, three-state electronics must

be used. Dedicated custom-state microelectronics can be used for each positional notation (e.g., three-state electronics for ternary, four-state electronics for quaternary, etc.) as shown in FIGS. 2 and 3.

An additional implementation of this computing systems employs advanced multi-state microelectronics capable of supporting positional notations less than or equal to the available maximum number of states restricted by the hardware. For example, if there was a reliable ten-state memory unit, all positional notations up to 10 could use the same memory unit. As illustrated in **Error! Reference source not found.**, if OS 3 was a ternary positional notation system, then OS 3 would only use three of the available ten states in the shared ten-state memory unit; a quaternary system would only use four of the available ten states.

In this embodiment shown in FIG. 4, the router directs the incoming data stream to its computing unit 3. The computing unit processes the data streams with OS 3, sharing the data streams with the memory unit and generating the resulting data stream. Similar to **Error! Reference source not found.**, the computing system is also capable of inputting data streams comprised of several segments that require different computing units. The router directs each data stream to the required computing unit 1 to N. Each computing unit then processes the corresponding data stream for that OS 1 to N using the single multi-state memory unit. The data streams generated from the multiple OS's are then combined to form the resulting final data stream. One embodiment is shown in **Error! Reference source not found.**, wherein the router directs each section of the incoming data stream to the respective computing units unit 1, unit 2, and unit N).

Each computing unit processes the corresponding data streams with the correct OS 1, OS 2, and OS N, utilizing the shared multi-state memory unit resource. The individual data streams generated for OS 1, OS 2, and OS N are combined to form the resulting data stream.
The novel systems and methods described herein may be applied to security protocols. The fundamental value of this computing architecture is to enhance security between parties communicating through a non-secure channel. Assuming that a code, or portion of a code, is written with high level software tools, as described in FIG. 1, it can be compiled to be executed by the computing units 1 to N. Cryptography can be used to transfer the information between the communicating parties based on what particular computing unit 1 to N is needed to execute the transmitted code. As described in FIG. 3, and FIG. 5, several portions of the code can be written to be executed by different computing units. The number of possible combinations is very high. For example, with a system with ten different units, there are 10 million ways to process a code in ten different segments. A malicious third party will have difficulty to attack the system without knowing which unit is needed to execute the corresponding portion of the code.

One embodiment of secure generation of a dynamic sequence of positions is presented in FIG. 6. A random number T_1 combined with a password generates a message digest through a hash function. A cryptographic scheme converts the message digest into the dynamic sequence of positions which is used to segment the code into a particular sequence, OS 2/OS 1/OS N in this embodiment.

At the receiving end, as shown in FIG. 7, the same random number T_1, the same password, the same hash function, and the same cryptographic scheme can generate the same dynamic sequence of positions. A third party without a password cannot generate the same dynamic sequence from T_1. The list of OS to be used during the exchange, OS 2/OS 3/OS N, is thereby independently generated by the communicating parties. The random number can be changed as often as needed to protect the communication.

This disclosure is intended to explain how to fashion and use various embodiments in accordance with the invention rather than to limit the true, intended, and fair scope and spirit thereof. The foregoing description is not intended to be exhaustive or to limit the invention to the precise form disclosed. Modifications or variations are possible in light of the above teachings. The embodiment(s) was chosen

and described to provide the best illustration of the principles of the invention and its practical application, and to enable one of ordinary skill in the art to utilize the invention in various embodiments and with various modifications as are suited to the particular use contemplated. All such modifications and variations are within the scope of the invention as determined by the appended claims, as may be amended during the pendency of this application for patent, and all equivalents thereof, when interpreted in accordance with the breadth to which they are fairly, legally, and equitably entitled.

CLAIMS

The invention claimed is:

1. A computing system, comprising:
    a. One or more computing units corresponding to 1 to N integer numbers, each computing unit capable of executing a first software code written with a corresponding instruction set 1 to N;

    wherein the computing system can be equipped with the capability to select one of its computing units at a specified time to execute said first software code; and

    a compilation of a second software code into the first software code is done by the computing system or independent of the computing system.

2. The system of claim 1, wherein when the compilation is performed independently, the computing system can respond to an instruction specifying which computing unit is to be used at a given time.

3. The system of claim 1, wherein the computing system is based on a dynamic hybridized positional position designed in a single heterogeneous computing system based on the availability of a supporting hardware to execute each different positional notation instruction set.

4. The system of claim 3, wherein the computing system can run exclusively in any one of a positional notation instructions sets incorporated into the computing system.

5. The system of claim 3, wherein the computing system can run in a configuration that hybridizes any combination of the available positional notation instruction sets incorporated into the computing system.

6. The system of claim 3, wherein the computing system is capable of dynamically switching between any singular or hybridized configuration of positional notation instruction sets.

7. The system of claim 1, wherein the computing system further comprises one or more ternary computing units corresponding to 1 to t integer numbers, each computing unit capable of executing a software code corresponding to one to t instructions sets.

8. The system of claim 1, wherein the computing system comprises one or more computing units corresponding to 1 to m integer numbers, all of the computing units are of the same positional notation and capable of executing a software code corresponding to 1 to m instruction sets.

9. The system of claim 8, wherein the positional notation is binary, ternary, quaternary, quinary, senary, septenary, octonary, nonary, denary, or a higher notation.

10. A compiler system comprising one or more units capable of generating one or more software codes variation corresponding to 1 to N integer numbers based on an instruction set 1 to N Wherein the software codes generated by the compiler can be executed by a computing system.

11. The system of claim 10, wherein the compiler system can generate an additional instruction with each software variation corresponding to 1 to N to instruct the computing system on which computing unit, corresponding to 1 to N, to use to execute the software variation.

12. The system of claim 10, further comprising a temporal control router wherein said router can select one of the available positional notation instruction sets to implement a computing system configuration.

13. The system of claim 12, wherein the temporal control router can hybridize the available positional notation instruction sets according to Equation 1.

14. The system of claim 13, wherein the temporal control router can set and change a recurrent time scale Δt that is used to dynamically switch between any available hybridized computing system configurations.

15. The system of claim 10, further comprising an encryption scheme to prevent a third party from knowing which computing unit to use to execute a software code.

16. The system of claim 15, wherein the encryption scheme is based on public key infrastructure, a hash function, symmetrical encryption, asymmetrical encryption, RSA, DES, AES, Elliptic Curve Cryptography, or quantum key distribution.

17. The system of claim 15, wherein a software code is segmented into sequences of different code variations aimed at different computing units to enhance the efficiency of the encryption scheme.

ABSTRACT

Disclosed herein is a computer architecture with the capability to execute instructions in different positional notation values. The definition of a positional notation value is given by the general formula that represent a base 10 numeral in any positional notation in the following manner: $\ldots d_3r^3 + d_2r^2 + d_1r^1 + d_0r^0$, where d is a coefficient, r is the base of the positional number system (i.e. r=2 for binary, or r=3 for ternary), and the exponent is the position of the digit. The computer architecture may include a special routing unit that can hybridize the instructions of multiple positional notation values in variable ratios. This routing unit may have a temporal control able to dynamically switch between the multiple hybridized instruction sets. Furthermore, this novel computer architecture may be applied to enhance security.

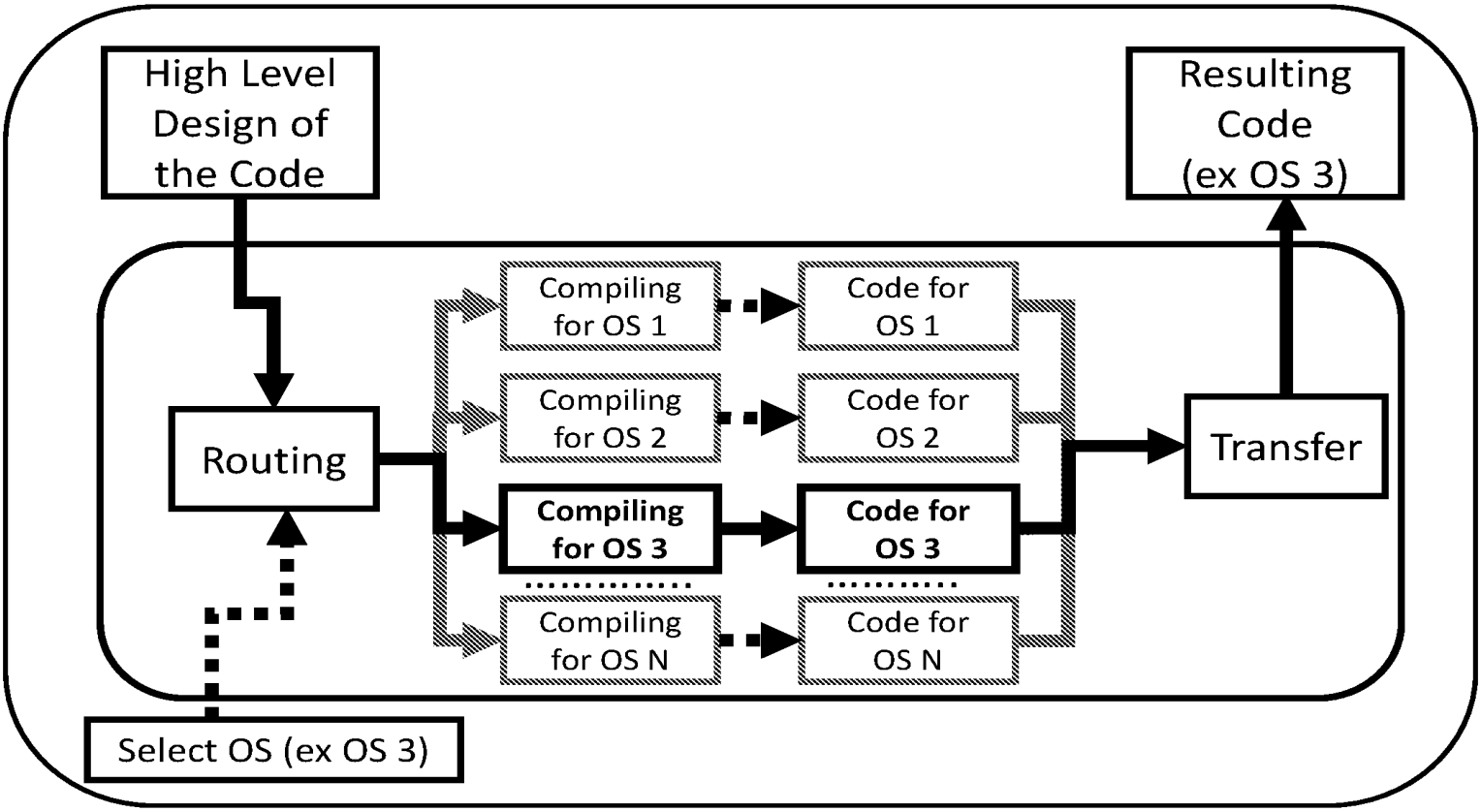
High Level Design of the Code
Resulting Code (ex OS 3)
Compiling for OS 1
Code for OS 1
Compiling for OS 2
Code for OS 2
Routing
Compiling for OS 3
Code for OS 3
Transfer
Compiling for OS N
Code for OS N
Select OS (ex OS 3)

FIG. 1

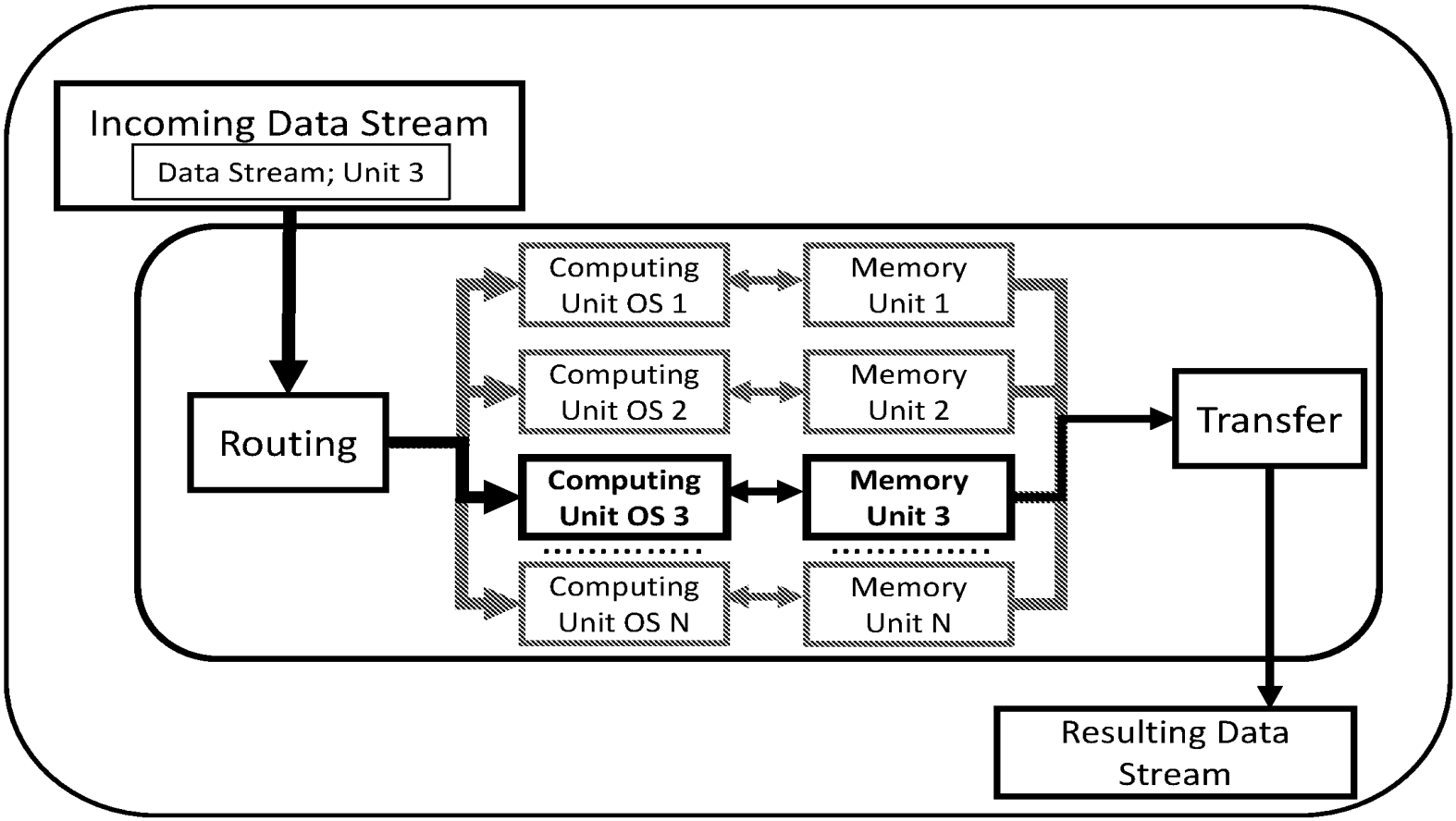
Incoming Data Stream
Data Stream; Unit 3
Routing
Computing Unit OS 1
Memory Unit 1
Computing Unit OS 2
Memory Unit 2
Computing Unit OS 3
Memory Unit 3
Computing Unit OS N
Memory Unit N
Transfer
Resulting Data Stream

FIG. 2

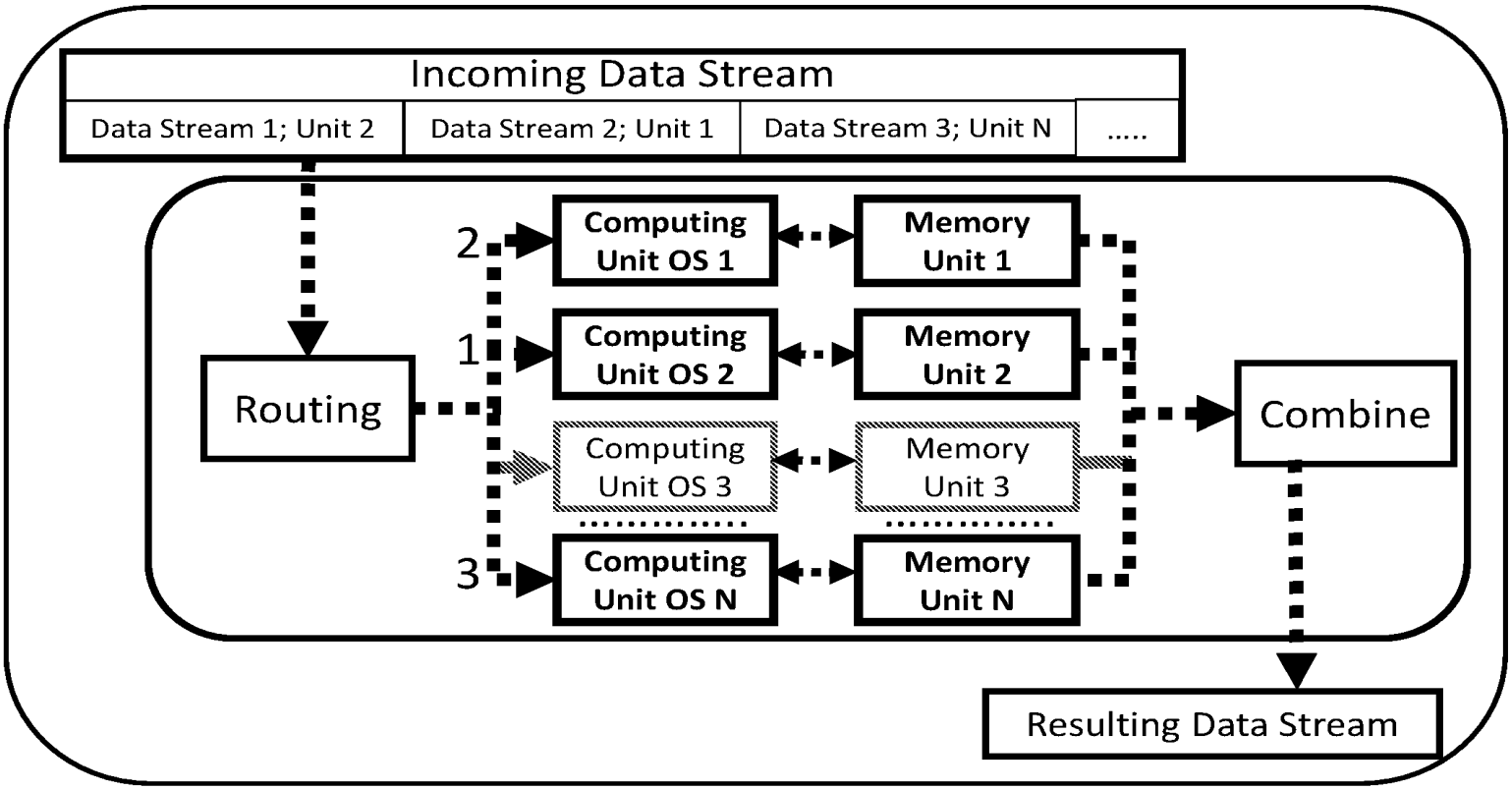
Incoming Data Stream
Data Stream 1; Unit 2
Data Stream 2; Unit 1
Data Stream 3; Unit N
.....
Routing
2
1
3
Computing Unit OS 1
Computing Unit OS 2
Computing Unit OS 3
Computing Unit OS N
Memory Unit 1
Memory Unit 2
Memory Unit 3
Memory Unit N
Combine
Resulting Data Stream

FIG. 3

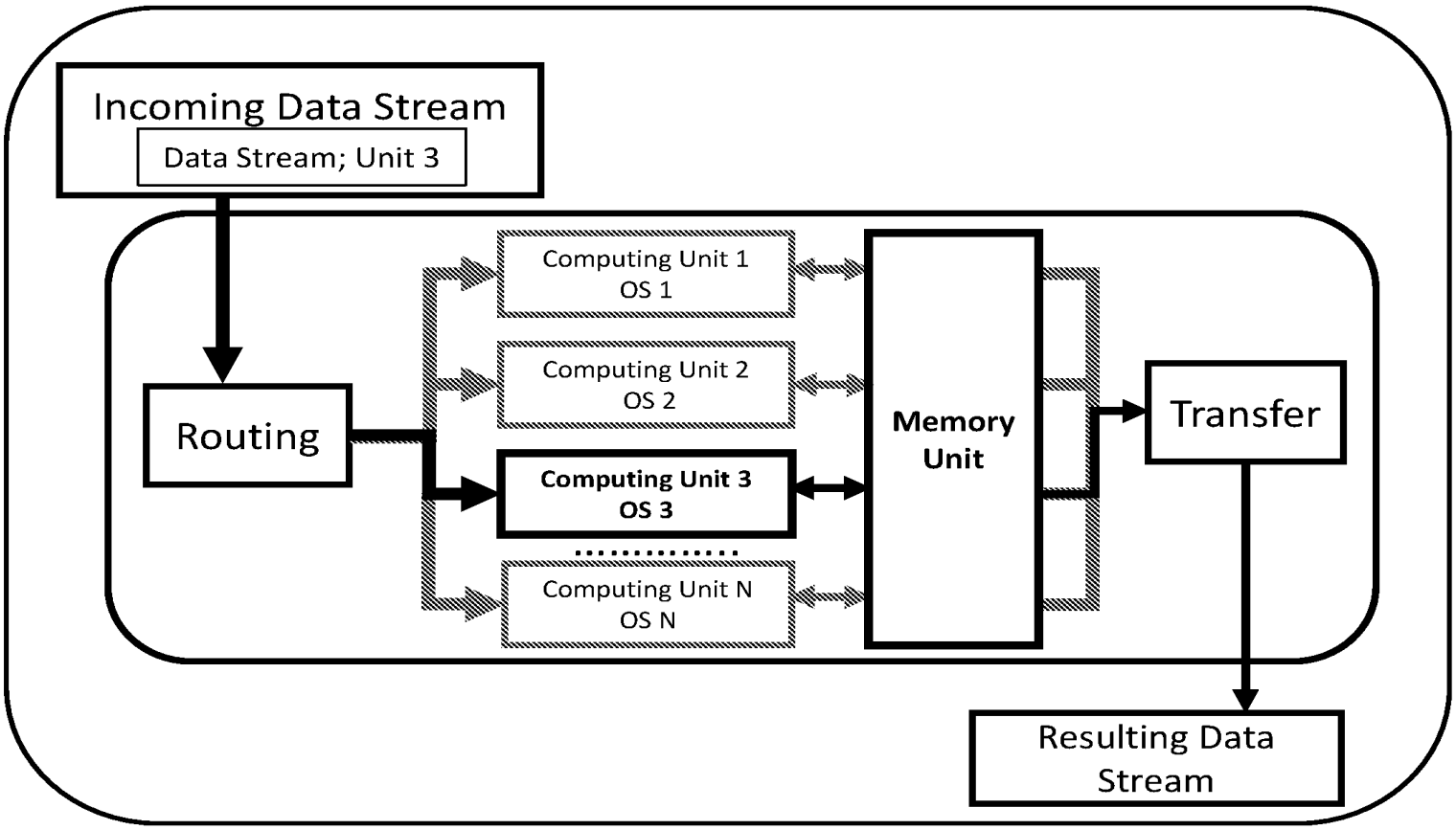
Incoming Data Stream
Data Stream; Unit 3
Routing
Computing Unit 1
OS 1
Computing Unit 2
OS 2
Computing Unit 3
OS 3
Computing Unit N
OS N
Memory Unit
Transfer
Resulting Data Stream

FIG. 4

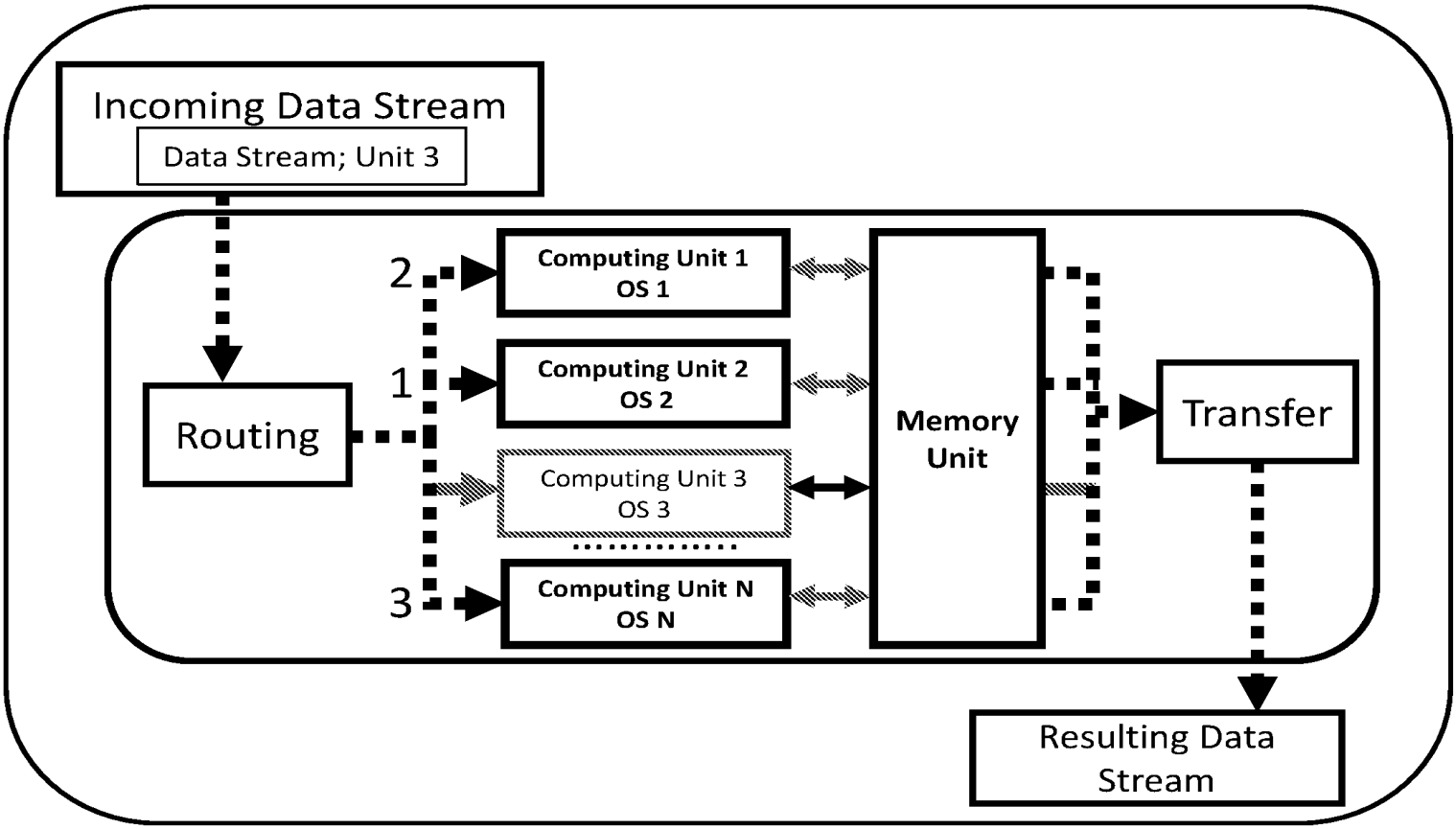
Incoming Data Stream
Data Stream; Unit 3
Routing
2
1
3
Computing Unit 1
OS 1
Computing Unit 2
OS 2
Computing Unit 3
OS 3
Computing Unit N
OS N
Memory
Unit
Transfer
Resulting Data
Stream

FIG. 5

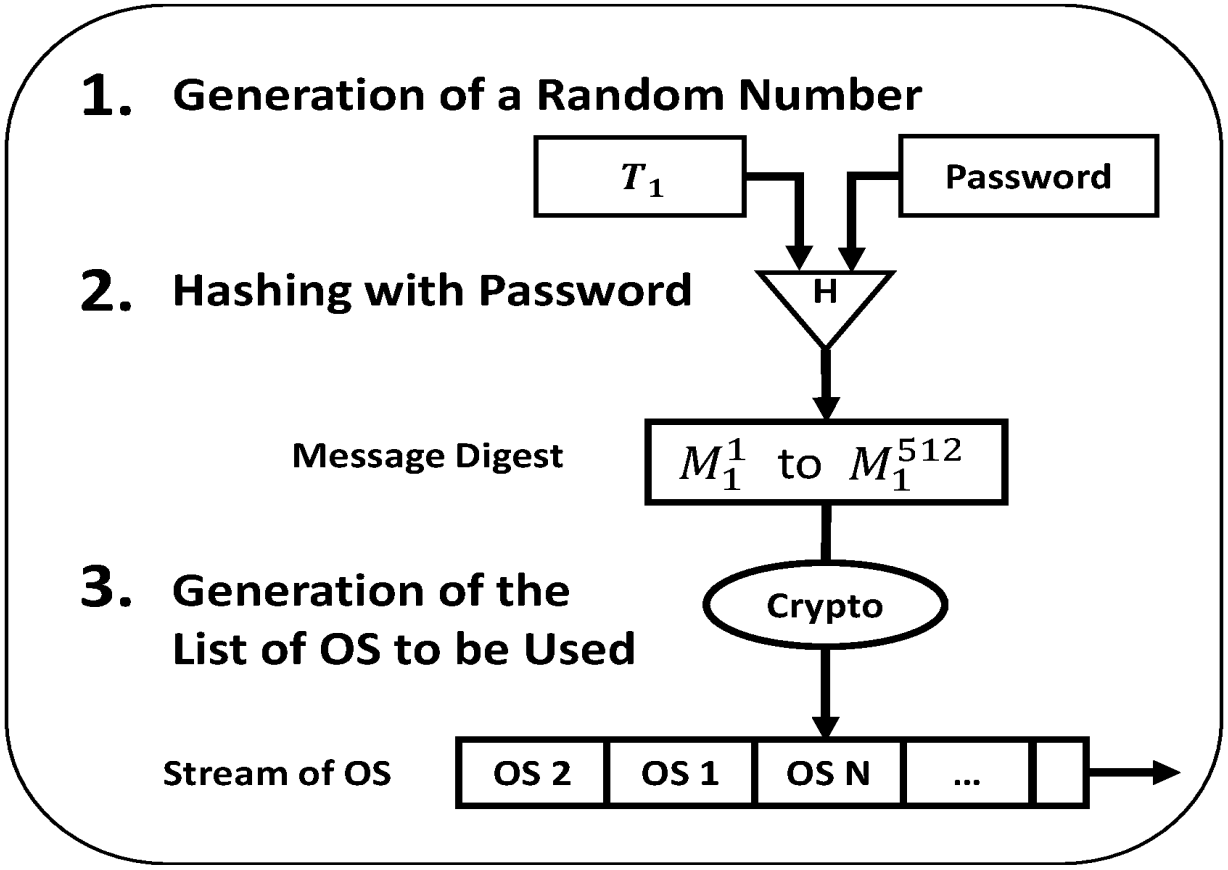
1. Generation of a Random Number
T_1
Password
2. Hashing with Password
H
Message Digest
M_1^1 to M_1^{512}
3. Generation of the List of OS to be Used
Crypto
Stream of OS
OS 2
OS 1
OS N
...

FIG. 6

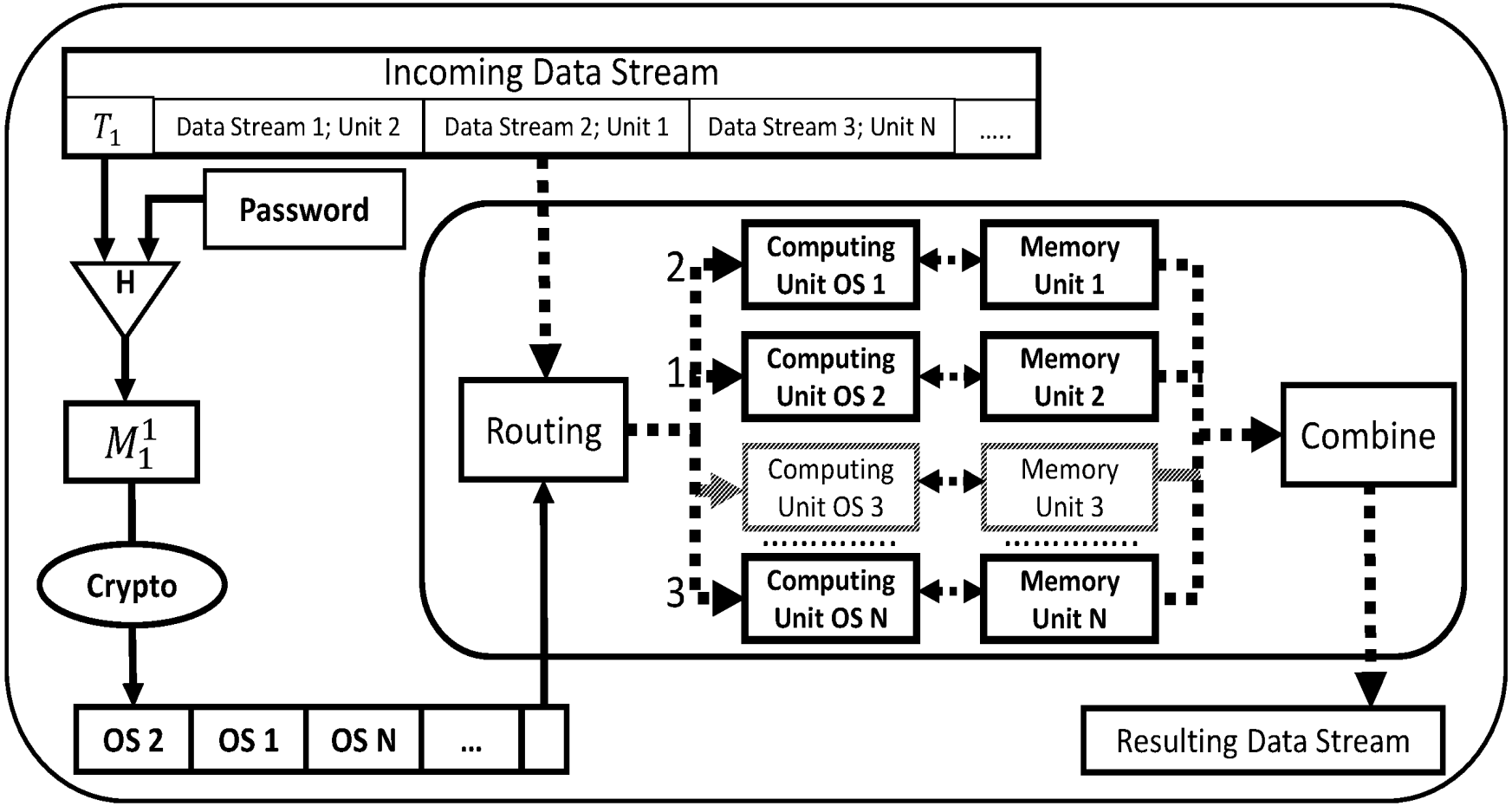
Incoming Data Stream
T_1
Data Stream 1; Unit 2
Data Stream 2; Unit 1
Data Stream 3; Unit N
.....
Password
H
M_1^1
Crypto
OS 2
OS 1
OS N
...
Routing
2
1
3
Computing Unit OS 1
Computing Unit OS 2
Computing Unit OS 3
Computing Unit OS N
Memory Unit 1
Memory Unit 2
Memory Unit 3
Memory Unit N
Combine
Resulting Data Stream

FIG. 7