DR INŻ. TOMASZ PAWLAK
DR INŻ. MARCIN SZUBERT

WEB FUNDAMENTALS

HYPERTEXT TRANSFER PROTOCOL

PRESENTATION OUTLINE

- Evolution of the Web
- Building blocks of the Web
- HTTP — Hypertext Transfer Protocol
  - Messages: verbs, status codes, headers
  - Connections: performance, security, proxies
- HTTP extensions: SPDY, HTTP/2

MOTIVATION

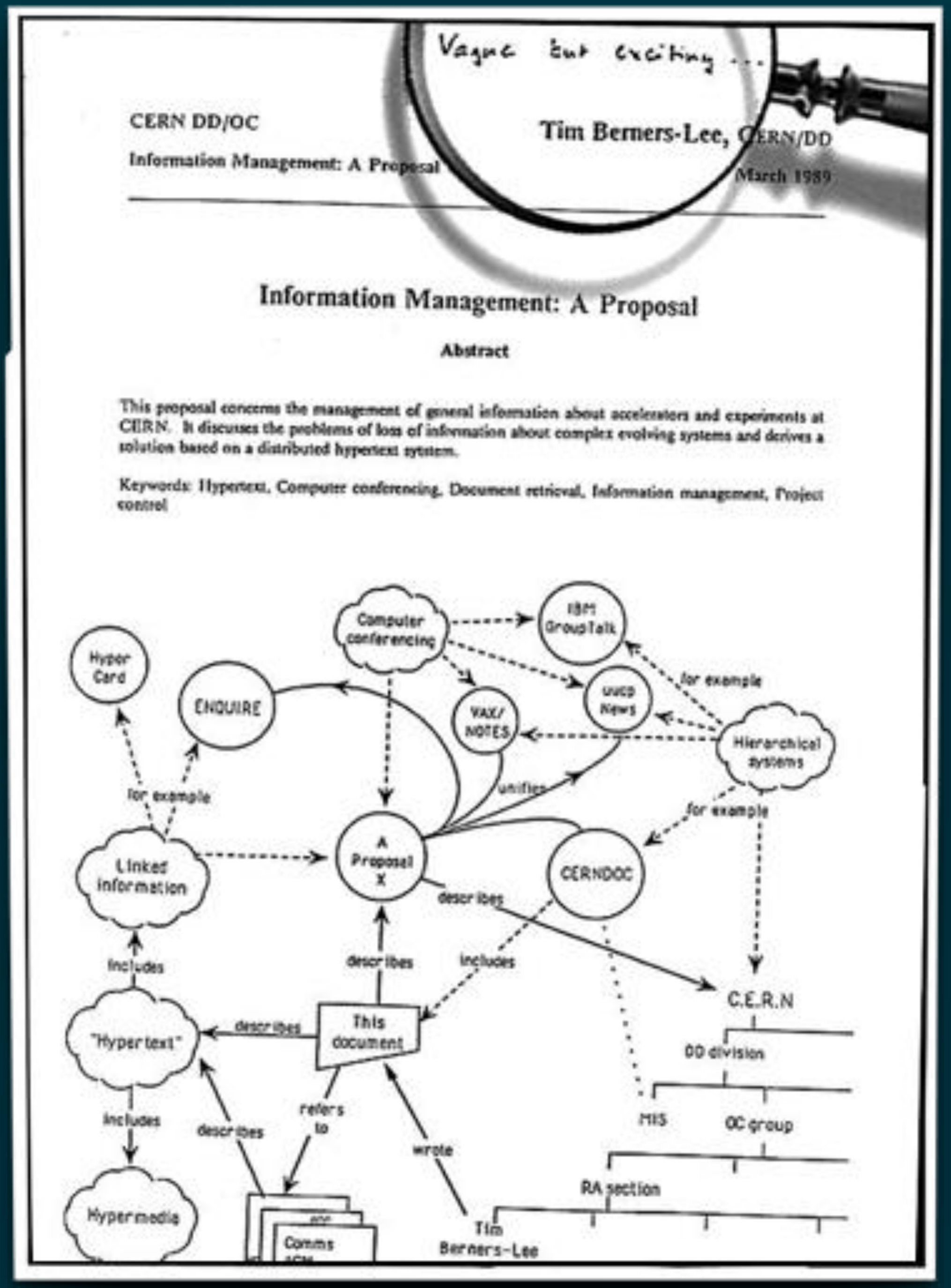
- *HTTP: the protocol every web developer must know.*
- HTTP is the foundation of the most successful distributed system ever built — the World Wide Web.
- Understanding HTTP is critical to:
  - designing a clean, simple and RESTful Web API,
  - implementing efficient and scalable web applications,
  - debugging web application.

SCALABILITY

- "*Scalability* is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth.", Wikipedia

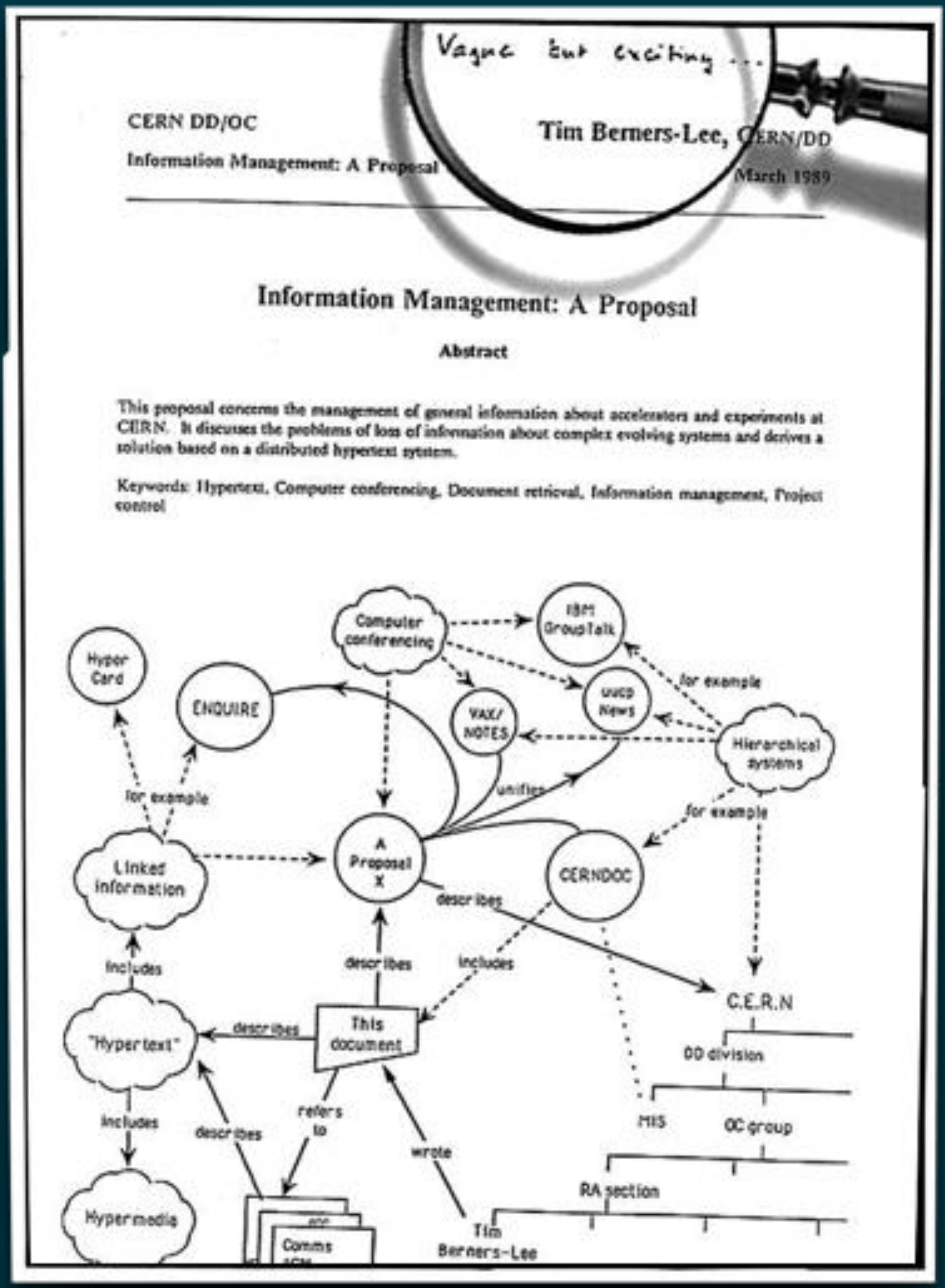
HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.



HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.
- **ARPANET** — open **decentralized** computer network architecture.


Vague but exciting ...

CERN DD/OC

Information Management: A Proposal

Tim Berners-Lee, CERN/DD

March 1989

Information Management: A Proposal

Abstract

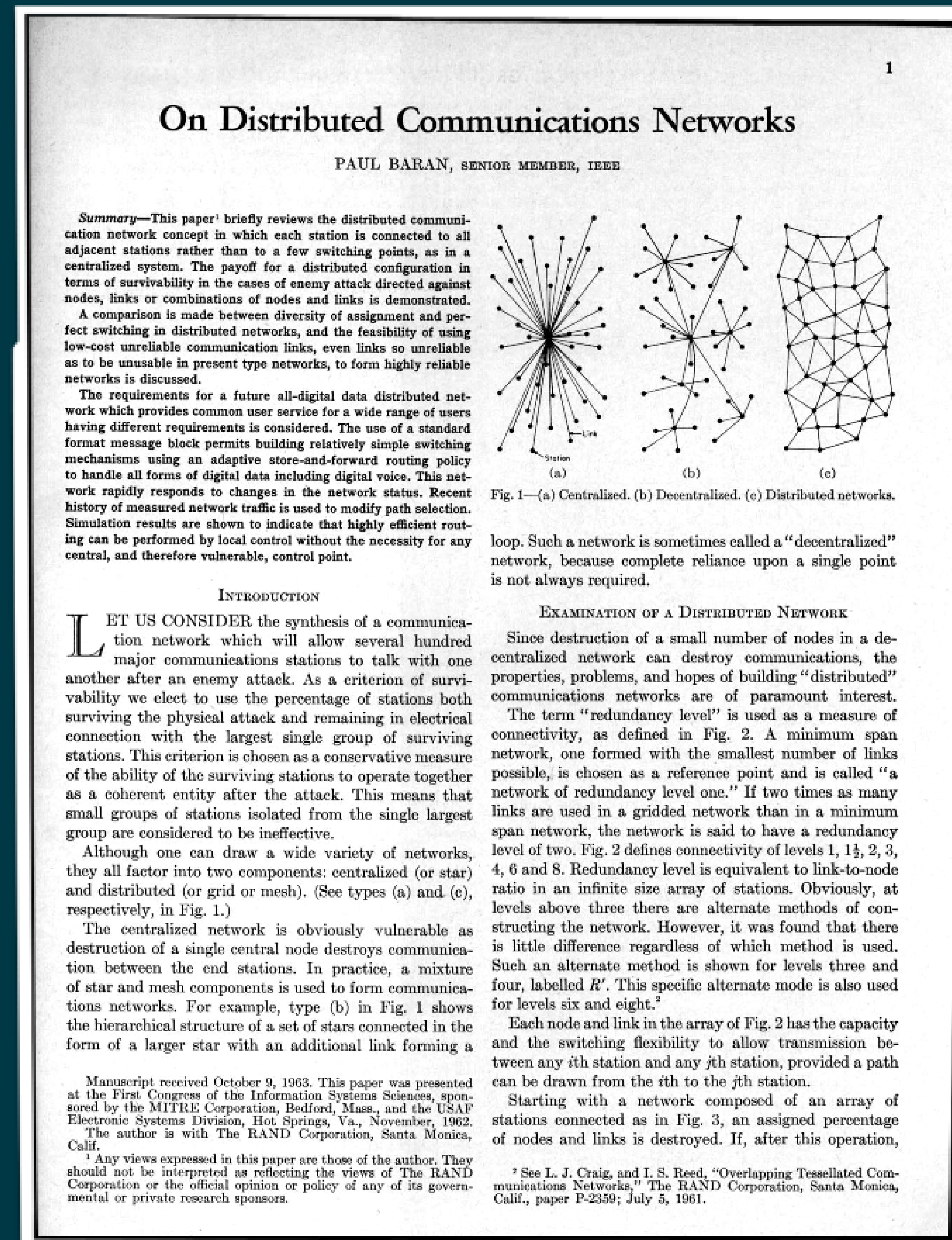
This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.

Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control

HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.

- **ARPANET** — open **decentralized** computer network architecture.



1

On Distributed Communications Networks

PAUL BARAN, SENIOR MEMBER, IEEE

Summary—This paper[1] briefly reviews the distributed communication network concept in which each station is connected to all adjacent stations rather than to a few switching points, as in a centralized system. The payoff for a distributed configuration in terms of survivability in the cases of enemy attack directed against nodes, links or combinations of nodes and links is demonstrated.

A comparison is made between diversity of assignment and perfect switching in distributed networks, and the feasibility of using low-cost unreliable communication links, even links so unreliable as to be unusable in present type networks, to form highly reliable networks is discussed.

The requirements for a future all-digital data distributed network which provides common user service for a wide range of users having different requirements is considered. The use of a standard format message block permits building relatively simple switching mechanisms using an adaptive store-and-forward routing policy to handle all forms of digital data including digital voice. This network rapidly responds to changes in the network status. Recent history of measured network traffic is used to modify path selection. Simulation results are shown to indicate that highly efficient routing can be performed by local control without the necessity for any central, and therefore vulnerable, control point.

Link

Station

(a) (b) (c)

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

INTRODUCTION

LET US CONSIDER the synthesis of a communication network which will allow several hundred major communications stations to talk with one another after an enemy attack. As a criterion of survivability we elect to use the percentage of stations both surviving the physical attack and remaining in electrical connection with the largest single group of surviving stations. This criterion is chosen as a conservative measure of the ability of the surviving stations to operate together as a coherent entity after the attack. This means that small groups of stations isolated from the single largest group are considered to be ineffective.

Although one can draw a wide variety of networks, they all factor into two components: centralized (or star) and distributed (or grid or mesh). (See types (a) and (c), respectively, in Fig. 1.)

The centralized network is obviously vulnerable as destruction of a single central node destroys communication between the end stations. In practice, a mixture of star and mesh components is used to form communications networks. For example, type (b) in Fig. 1 shows the hierarchical structure of a set of stars connected in the form of a larger star with an additional link forming a loop. Such a network is sometimes called a "decentralized" network, because complete reliance upon a single point is not always required.

EXAMINATION OF A DISTRIBUTED NETWORK

Since destruction of a small number of nodes in a decentralized network can destroy communications, the properties, problems, and hopes of building "distributed" communications networks are of paramount interest.

The term "redundancy level" is used as a measure of connectivity, as defined in Fig. 2. A minimum span network, one formed with the smallest number of links possible, is chosen as a reference point and is called "a network of redundancy level one." If two times as many links are used in a gridded network than in a minimum span network, the network is said to have a redundancy level of two. Fig. 2 defines connectivity of levels 1, $1\frac{1}{2}$, 2, 3, 4, 6 and 8. Redundancy level is equivalent to link-to-node ratio in an infinite size array of stations. Obviously, at levels above three there are alternate methods of constructing the network. However, it was found that there is little difference regardless of which method is used. Such an alternate method is shown for levels three and four, labelled R'. This specific alternate mode is also used for levels six and eight.[2]

Each node and link in the array of Fig. 2 has the capacity and the switching flexibility to allow transmission between any ith station and any jth station, provided a path can be drawn from the ith to the jth station.

Starting with a network composed of an array of stations connected as in Fig. 3, an assigned percentage of nodes and links is destroyed. If, after this operation,

Manuscript received October 9, 1963. This paper was presented at the First Congress of the Information Systems Sciences, sponsored by the MITRE Corporation, Bedford, Mass., and the USAF Electronic Systems Division, Hot Springs, Va., November, 1962.

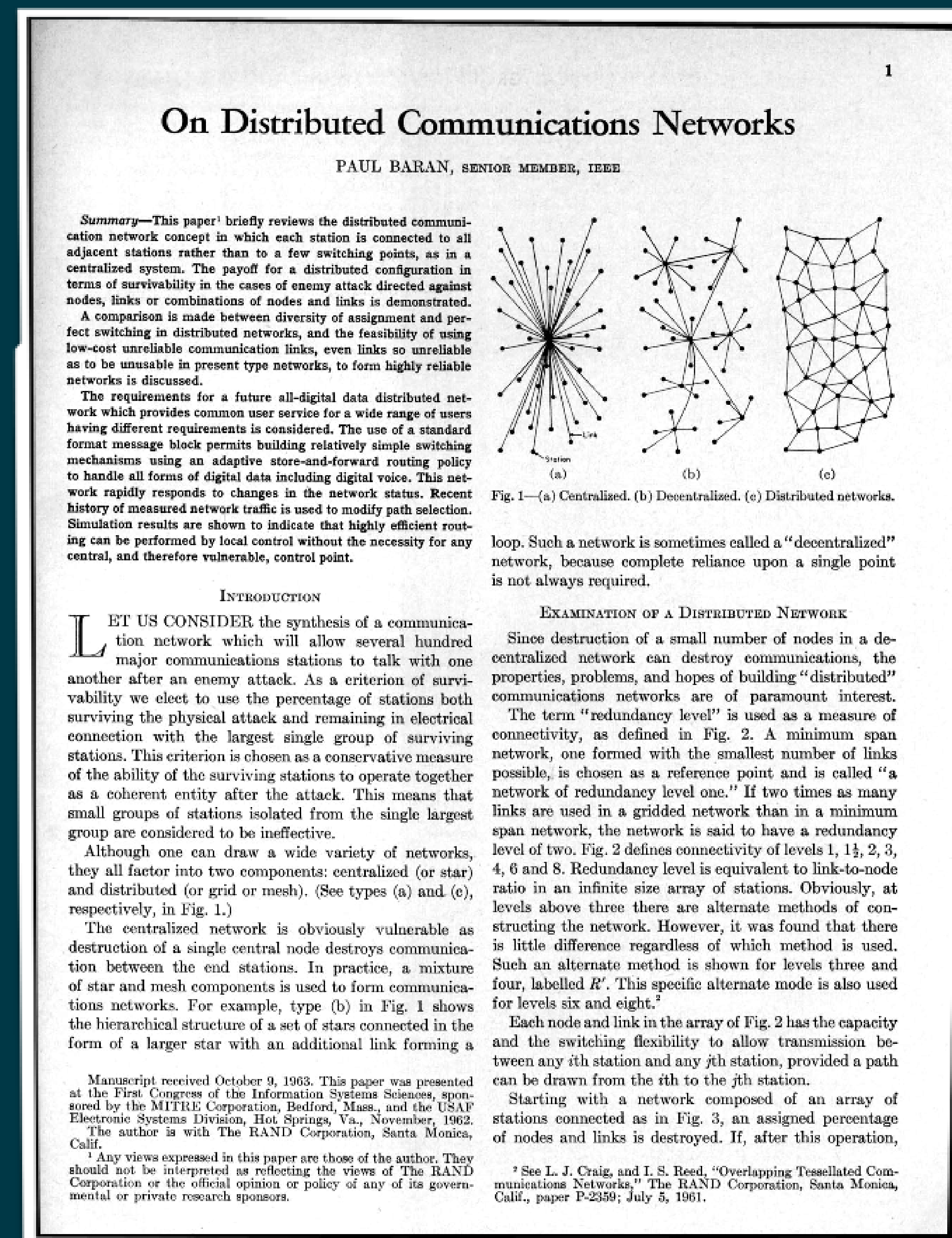
The author is with The RAND Corporation, Santa Monica, Calif.

[1] Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors.

[2] See L. J. Craig, and I. S. Reed, "Overlapping Tessellated Communications Networks," The RAND Corporation, Santa Monica, Calif., paper P-2359; July 5, 1961.

HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.
- **ARPANET** — open **decentralized** computer network architecture.
  - 1969: First message sent



On Distributed Communications Networks

PAUL BARAN, SENIOR MEMBER, IEEE

Summary—This paper[1] briefly reviews the distributed communication network concept in which each station is connected to all adjacent stations rather than to a few switching points, as in a centralized system. The payoff for a distributed configuration in terms of survivability in the cases of enemy attack directed against nodes, links or combinations of nodes and links is demonstrated.

A comparison is made between diversity of assignment and perfect switching in distributed networks, and the feasibility of using low-cost unreliable communication links, even links so unreliable as to be unusable in present type networks, to form highly reliable networks is discussed.

The requirements for a future all-digital data distributed network which provides common user service for a wide range of users having different requirements is considered. The use of a standard format message block permits building relatively simple switching mechanisms using an adaptive store-and-forward routing policy to handle all forms of digital data including digital voice. This network rapidly responds to changes in the network status. Recent history of measured network traffic is used to modify path selection. Simulation results are shown to indicate that highly efficient routing can be performed by local control without the necessity for any central, and therefore vulnerable, control point.

Introduction

LET US CONSIDER the synthesis of a communication network which will allow several hundred major communications stations to talk with one another after an enemy attack. As a criterion of survivability we elect to use the percentage of stations both surviving the physical attack and remaining in electrical connection with the largest single group of surviving stations. This criterion is chosen as a conservative measure of the ability of the surviving stations to operate together as a coherent entity after the attack. This means that small groups of stations isolated from the single largest group are considered to be ineffective.

Although one can draw a wide variety of networks, they all factor into two components: centralized (or star) and distributed (or grid or mesh). (See types (a) and (c), respectively, in Fig. 1.)

The centralized network is obviously vulnerable as destruction of a single central node destroys communication between the end stations. In practice, a mixture of star and mesh components is used to form communications networks. For example, type (b) in Fig. 1 shows the hierarchical structure of a set of stars connected in the form of a larger star with an additional link forming a loop. Such a network is sometimes called a "decentralized" network, because complete reliance upon a single point is not always required.

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

Examination of a Distributed Network

Since destruction of a small number of nodes in a decentralized network can destroy communications, the properties, problems, and hopes of building "distributed" communications networks are of paramount interest.

The term "redundancy level" is used as a measure of connectivity, as defined in Fig. 2. A minimum span network, one formed with the smallest number of links possible, is chosen as a reference point and is called "a network of redundancy level one." If two times as many links are used in a gridded network than in a minimum span network, the network is said to have a redundancy level of two. Fig. 2 defines connectivity of levels 1, $1\frac{1}{2}$, 2, 3, 4, 6 and 8. Redundancy level is equivalent to link-to-node ratio in an infinite size array of stations. Obviously, at levels above three there are alternate methods of constructing the network. However, it was found that there is little difference regardless of which method is used. Such an alternate method is shown for levels three and four, labelled R'. This specific alternate mode is also used for levels six and eight.[2]

Each node and link in the array of Fig. 2 has the capacity and the switching flexibility to allow transmission between any ith station and any jth station, provided a path can be drawn from the ith to the jth station.

Starting with a network composed of an array of stations connected as in Fig. 3, an assigned percentage of nodes and links is destroyed. If, after this operation,

Manuscript received October 9, 1963. This paper was presented at the First Congress of the Information Systems Sciences, sponsored by the MITRE Corporation, Bedford, Mass., and the USAF Electronic Systems Division, Hot Springs, Va., November, 1962.

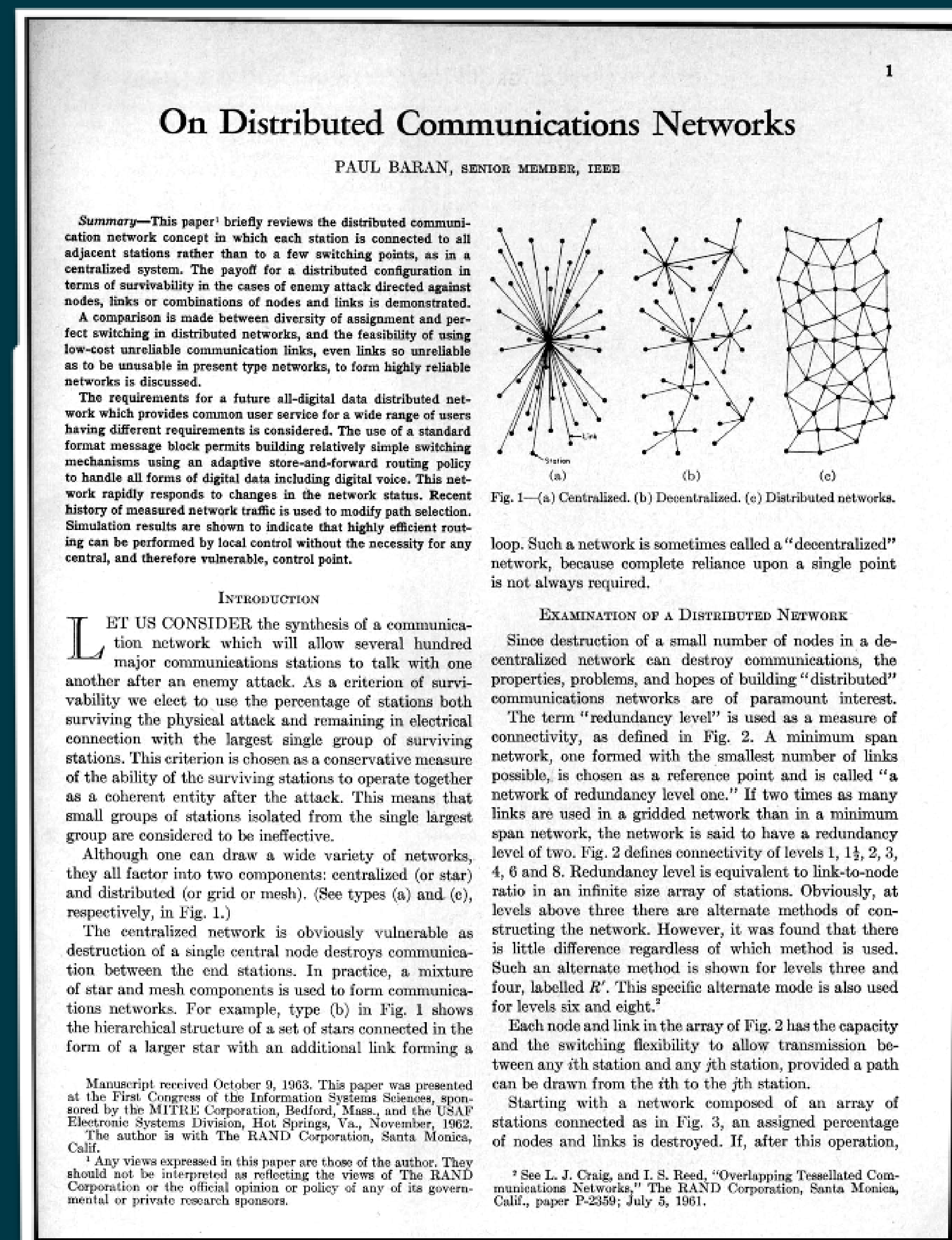
The author is with The RAND Corporation, Santa Monica, Calif.

[1] Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors.

[2] See L. J. Craig, and I. S. Reed, "Overlapping Tessellated Communications Networks," The RAND Corporation, Santa Monica, Calif., paper P-2359; July 5, 1961.

HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.
- **ARPANET** — open **decentralized** computer network architecture.
  - 1969: First message sent
  - 1971: First e-mail sent



1

On Distributed Communications Networks

PAUL BARAN, SENIOR MEMBER, IEEE

Summary—This paper[1] briefly reviews the distributed communication network concept in which each station is connected to all adjacent stations rather than to a few switching points, as in a centralized system. The payoff for a distributed configuration in terms of survivability in the cases of enemy attack directed against nodes, links or combinations of nodes and links is demonstrated.

A comparison is made between diversity of assignment and perfect switching in distributed networks, and the feasibility of using low-cost unreliable communication links, even links so unreliable as to be unusable in present type networks, to form highly reliable networks is discussed.

The requirements for a future all-digital data distributed network which provides common user service for a wide range of users having different requirements is considered. The use of a standard format message block permits building relatively simple switching mechanisms using an adaptive store-and-forward routing policy to handle all forms of digital data including digital voice. This network rapidly responds to changes in the network status. Recent history of measured network traffic is used to modify path selection. Simulation results are shown to indicate that highly efficient routing can be performed by local control without the necessity for any central, and therefore vulnerable, control point.

Introduction

LET US CONSIDER the synthesis of a communication network which will allow several hundred major communications stations to talk with one another after an enemy attack. As a criterion of survivability we elect to use the percentage of stations both surviving the physical attack and remaining in electrical connection with the largest single group of surviving stations. This criterion is chosen as a conservative measure of the ability of the surviving stations to operate together as a coherent entity after the attack. This means that small groups of stations isolated from the single largest group are considered to be ineffective.

Although one can draw a wide variety of networks, they all factor into two components: centralized (or star) and distributed (or grid or mesh). (See types (a) and (c), respectively, in Fig. 1.)

The centralized network is obviously vulnerable as destruction of a single central node destroys communication between the end stations. In practice, a mixture of star and mesh components is used to form communications networks. For example, type (b) in Fig. 1 shows the hierarchical structure of a set of stars connected in the form of a larger star with an additional link forming a loop. Such a network is sometimes called a "decentralized" network, because complete reliance upon a single point is not always required.

Link
Station
(a) (b) (c)

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

Examination of a Distributed Network

Since destruction of a small number of nodes in a decentralized network can destroy communications, the properties, problems, and hopes of building "distributed" communications networks are of paramount interest.

The term "redundancy level" is used as a measure of connectivity, as defined in Fig. 2. A minimum span network, one formed with the smallest number of links possible, is chosen as a reference point and is called "a network of redundancy level one." If two times as many links are used in a gridded network than in a minimum span network, the network is said to have a redundancy level of two. Fig. 2 defines connectivity of levels 1, $1\frac{1}{2}$, 2, 3, 4, 6 and 8. Redundancy level is equivalent to link-to-node ratio in an infinite size array of stations. Obviously, at levels above three there are alternate methods of constructing the network. However, it was found that there is little difference regardless of which method is used. Such an alternate method is shown for levels three and four, labelled R'. This specific alternate mode is also used for levels six and eight.[2]

Each node and link in the array of Fig. 2 has the capacity and the switching flexibility to allow transmission between any ith station and any jth station, provided a path can be drawn from the ith to the jth station.

Starting with a network composed of an array of stations connected as in Fig. 3, an assigned percentage of nodes and links is destroyed. If, after this operation,

Manuscript received October 9, 1963. This paper was presented at the First Congress of the Information Systems Sciences, sponsored by the MITRE Corporation, Bedford, Mass., and the USAF Electronic Systems Division, Hot Springs, Va., November, 1962.

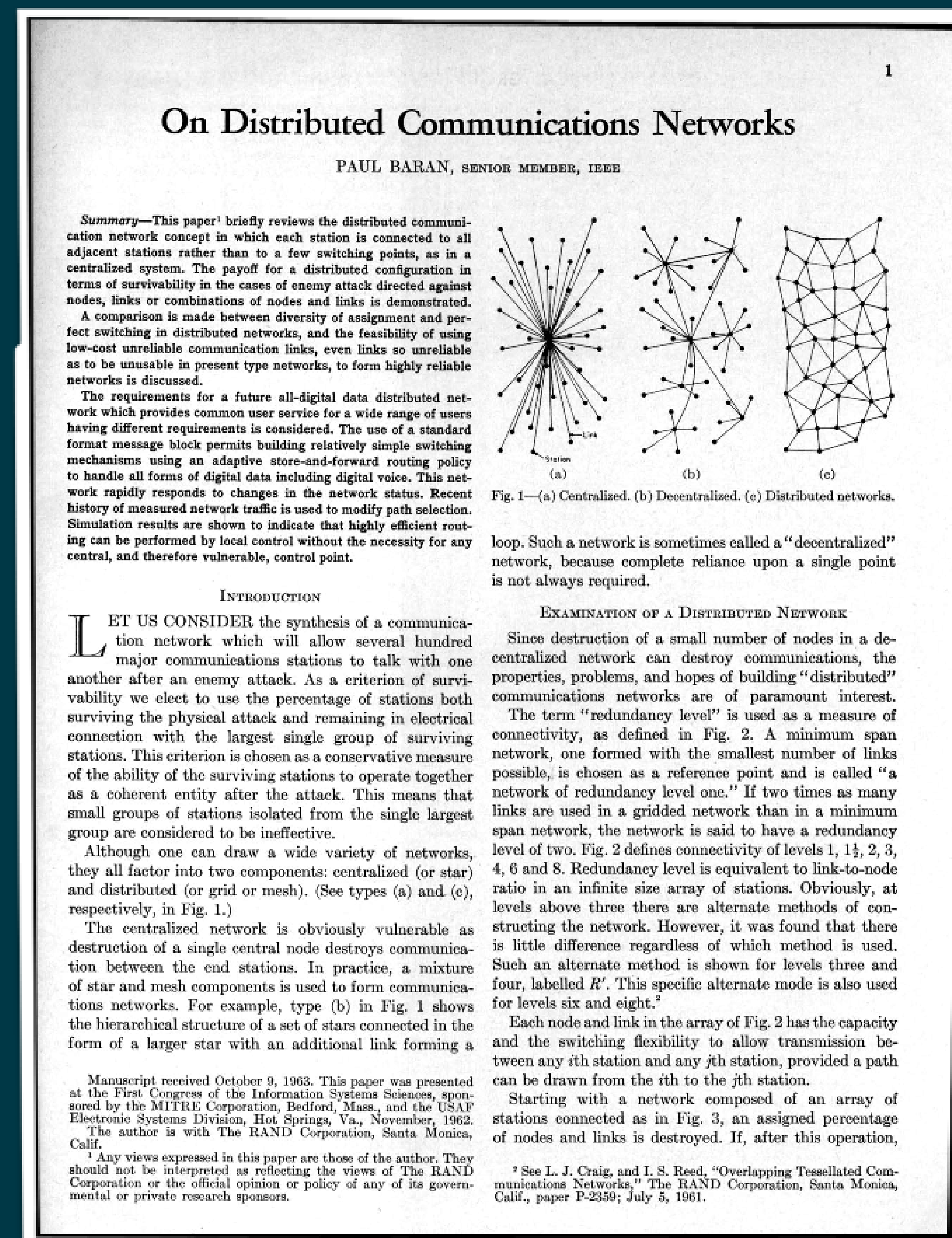
The author is with The RAND Corporation, Santa Monica, Calif.

[1] Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors.

[2] See L. J. Craig, and I. S. Reed, "Overlapping Tessellated Communications Networks," The RAND Corporation, Santa Monica, Calif., paper P-2359; July 5, 1961.

HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.
- **ARPANET** — open **decentralized** computer network architecture.
  - 1969: First message sent
  - 1971: First e-mail sent
  - 1973: File Transfer Protocol (RFC 354)



1

On Distributed Communications Networks

PAUL BARAN, SENIOR MEMBER, IEEE

Summary—This paper[1] briefly reviews the distributed communication network concept in which each station is connected to all adjacent stations rather than to a few switching points, as in a centralized system. The payoff for a distributed configuration in terms of survivability in the cases of enemy attack directed against nodes, links or combinations of nodes and links is demonstrated.

A comparison is made between diversity of assignment and perfect switching in distributed networks, and the feasibility of using low-cost unreliable communication links, even links so unreliable as to be unusable in present type networks, to form highly reliable networks is discussed.

The requirements for a future all-digital data distributed network which provides common user service for a wide range of users having different requirements is considered. The use of a standard format message block permits building relatively simple switching mechanisms using an adaptive store-and-forward routing policy to handle all forms of digital data including digital voice. This network rapidly responds to changes in the network status. Recent history of measured network traffic is used to modify path selection. Simulation results are shown to indicate that highly efficient routing can be performed by local control without the necessity for any central, and therefore vulnerable, control point.

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

Introduction

LET US CONSIDER the synthesis of a communication network which will allow several hundred major communications stations to talk with one another after an enemy attack. As a criterion of survivability we elect to use the percentage of stations both surviving the physical attack and remaining in electrical connection with the largest single group of surviving stations. This criterion is chosen as a conservative measure of the ability of the surviving stations to operate together as a coherent entity after the attack. This means that small groups of stations isolated from the single largest group are considered to be ineffective.

Although one can draw a wide variety of networks, they all factor into two components: centralized (or star) and distributed (or grid or mesh). (See types (a) and (c), respectively, in Fig. 1.)

The centralized network is obviously vulnerable as destruction of a single central node destroys communication between the end stations. In practice, a mixture of star and mesh components is used to form communications networks. For example, type (b) in Fig. 1 shows the hierarchical structure of a set of stars connected in the form of a larger star with an additional link forming a loop. Such a network is sometimes called a "decentralized" network, because complete reliance upon a single point is not always required.

Examination of a Distributed Network

Since destruction of a small number of nodes in a decentralized network can destroy communications, the properties, problems, and hopes of building "distributed" communications networks are of paramount interest.

The term "redundancy level" is used as a measure of connectivity, as defined in Fig. 2. A minimum span network, one formed with the smallest number of links possible, is chosen as a reference point and is called "a network of redundancy level one." If two times as many links are used in a gridded network than in a minimum span network, the network is said to have a redundancy level of two. Fig. 2 defines connectivity of levels 1, $1\frac{1}{2}$, 2, 3, 4, 6 and 8. Redundancy level is equivalent to link-to-node ratio in an infinite size array of stations. Obviously, at levels above three there are alternate methods of constructing the network. However, it was found that there is little difference regardless of which method is used. Such an alternate method is shown for levels three and four, labelled R'. This specific alternate mode is also used for levels six and eight.[2]

Each node and link in the array of Fig. 2 has the capacity and the switching flexibility to allow transmission between any ith station and any jth station, provided a path can be drawn from the ith to the jth station.

Starting with a network composed of an array of stations connected as in Fig. 3, an assigned percentage of nodes and links is destroyed. If, after this operation,

Manuscript received October 9, 1963. This paper was presented at the First Congress of the Information Systems Sciences, sponsored by the MITRE Corporation, Bedford, Mass., and the USAF Electronic Systems Division, Hot Springs, Va., November, 1962.

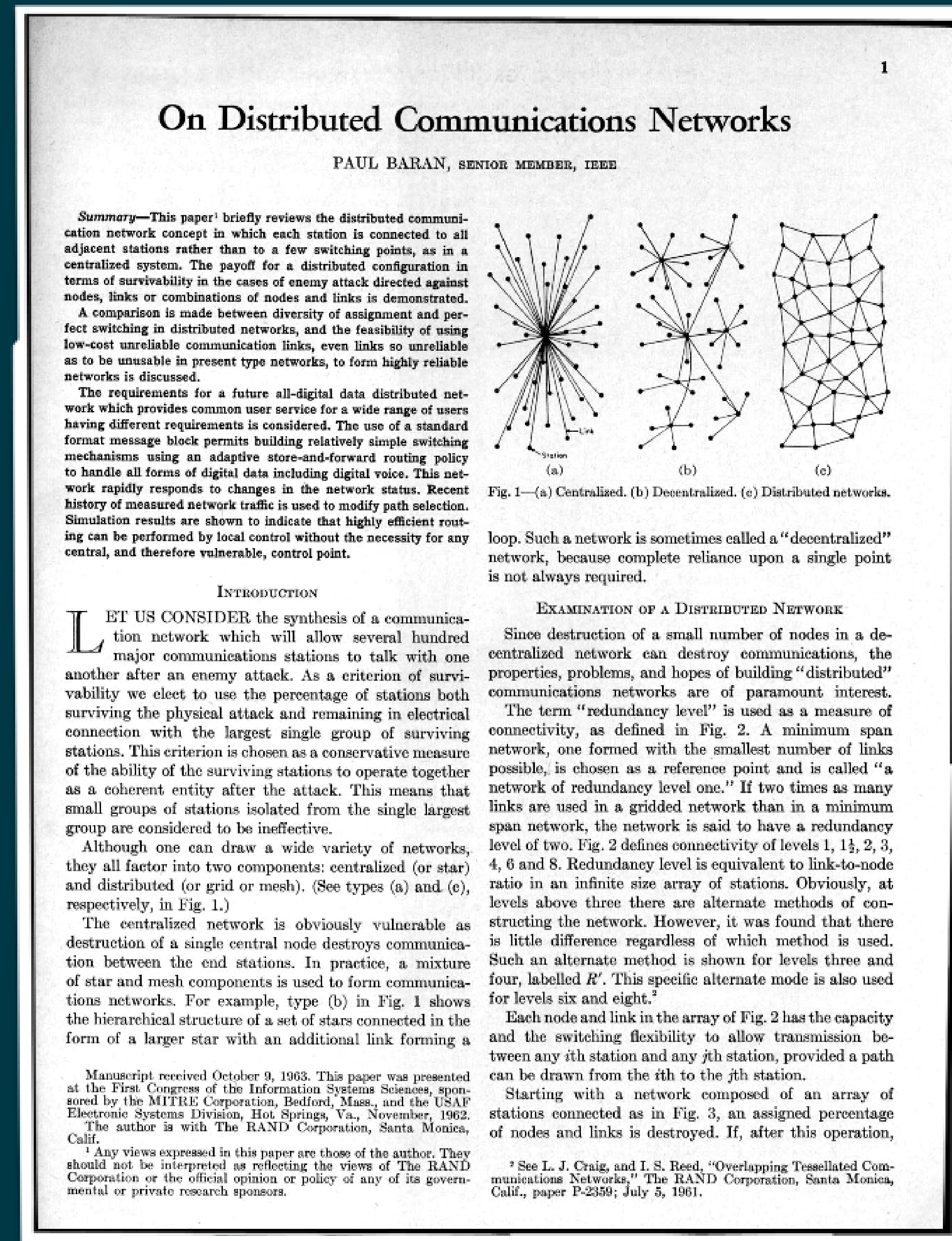
The author is with The RAND Corporation, Santa Monica, Calif.

[1] Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors.

[2] See L. J. Craig, and I. S. Reed, "Overlapping Tessellated Communications Networks," The RAND Corporation, Santa Monica, Calif., paper P-2359; July 5, 1961.

HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.
- **ARPANET** — open **decentralized** computer network architecture.
  - 1969: First message sent
  - 1971: First e-mail sent
  - 1973: File Transfer Protocol (RFC 354)
  - 1977: Network Voice Protocol (RFC 741)



On Distributed Communications Networks

PAUL BARAN, SENIOR MEMBER, IEEE

Summary—This paper[1] briefly reviews the distributed communication network concept in which each station is connected to all adjacent stations rather than to a few switching points, as in a centralized system. The payoff for a distributed configuration in terms of survivability in the cases of enemy attack directed against nodes, links or combinations of nodes and links is demonstrated.

A comparison is made between diversity of assignment and perfect switching in distributed networks, and the feasibility of using low-cost unreliable communication links, even links so unreliable as to be unusable in present type networks, to form highly reliable networks is discussed.

The requirements for a future all-digital data distributed network which provides common user service for a wide range of users having different requirements is considered. The use of a standard format message block permits building relatively simple switching mechanisms using an adaptive store-and-forward routing policy to handle all forms of digital data including digital voice. This network rapidly responds to changes in the network status. Recent history of measured network traffic is used to modify path selection. Simulation results are shown to indicate that highly efficient routing can be performed by local control without the necessity for any central, and therefore vulnerable, control point.

Introduction

LET US CONSIDER the synthesis of a communication network which will allow several hundred major communications stations to talk with one another after an enemy attack. As a criterion of survivability we elect to use the percentage of stations both surviving the physical attack and remaining in electrical connection with the largest single group of surviving stations. This criterion is chosen as a conservative measure of the ability of the surviving stations to operate together as a coherent entity after the attack. This means that small groups of stations isolated from the single largest group are considered to be ineffective.

Although one can draw a wide variety of networks, they all factor into two components: centralized (or star) and distributed (or grid or mesh). (See types (a) and (c), respectively, in Fig. 1.)

The centralized network is obviously vulnerable as destruction of a single central node destroys communication between the end stations. In practice, a mixture of star and mesh components is used to form communications networks. For example, type (b) in Fig. 1 shows the hierarchical structure of a set of stars connected in the form of a larger star with an additional link forming a loop. Such a network is sometimes called a "decentralized" network, because complete reliance upon a single point is not always required.

Link
Station
(a) (b) (c)

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

Examination of a Distributed Network

Since destruction of a small number of nodes in a decentralized network can destroy communications, the properties, problems, and hopes of building "distributed" communications networks are of paramount interest.

The term "redundancy level" is used as a measure of connectivity, as defined in Fig. 2. A minimum span network, one formed with the smallest number of links possible, is chosen as a reference point and is called "a network of redundancy level one." If two times as many links are used in a gridded network than in a minimum span network, the network is said to have a redundancy level of two. Fig. 2 defines connectivity of levels 1, $1\frac{1}{2}$, 2, 3, 4, 6 and 8. Redundancy level is equivalent to link-to-node ratio in an infinite size array of stations. Obviously, at levels above three there are alternate methods of constructing the network. However, it was found that there is little difference regardless of which method is used. Such an alternate method is shown for levels three and four, labelled R'. This specific alternate mode is also used for levels six and eight.[2]

Each node and link in the array of Fig. 2 has the capacity and the switching flexibility to allow transmission between any ith station and any jth station, provided a path can be drawn from the ith to the jth station.

Starting with a network composed of an array of stations connected as in Fig. 3, an assigned percentage of nodes and links is destroyed. If, after this operation,

Manuscript received October 9, 1963. This paper was presented at the First Congress of the Information Systems Sciences, sponsored by the MITRE Corporation, Bedford, Mass., and the USAF Electronic Systems Division, Hot Springs, Va., November, 1962.

The author is with The RAND Corporation, Santa Monica, Calif.

[1] Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors.

[2] See L. J. Craig, and I. S. Reed, "Overlapping Tessellated Communications Networks," The RAND Corporation, Santa Monica, Calif., paper P-2359; July 5, 1961.

HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.
- **ARPANET** — open **decentralized** computer network architecture.
  - 1969: First message sent
  - 1971: First e-mail sent
  - 1973: File Transfer Protocol (RFC 354)
  - 1977: Network Voice Protocol (RFC 741)
  - 1981: Internet Protocol v4

On Distributed Communications Networks

PAUL BARAN, SENIOR MEMBER, IEEE

Summary—This paper[1] briefly reviews the distributed communication network concept in which each station is connected to all adjacent stations rather than to a few switching points, as in a centralized system. The payoff for a distributed configuration in terms of survivability in the cases of enemy attack directed against nodes, links or combinations of nodes and links is demonstrated.

A comparison is made between diversity of assignment and perfect switching in distributed networks, and the feasibility of using low-cost unreliable communication links, even links so unreliable as to be unusable in present type networks, to form highly reliable networks is discussed.

The requirements for a future all-digital data distributed network which provides common user service for a wide range of users having different requirements is considered. The use of a standard format message block permits building relatively simple switching mechanisms using an adaptive store-and-forward routing policy to handle all forms of digital data including digital voice. This network rapidly responds to changes in the network status. Recent history of measured network traffic is used to modify path selection. Simulation results are shown to indicate that highly efficient routing can be performed by local control without the necessity for any central, and therefore vulnerable, control point.

Introduction

LET US CONSIDER the synthesis of a communication network which will allow several hundred major communications stations to talk with one another after an enemy attack. As a criterion of survivability we elect to use the percentage of stations both surviving the physical attack and remaining in electrical connection with the largest single group of surviving stations. This criterion is chosen as a conservative measure of the ability of the surviving stations to operate together as a coherent entity after the attack. This means that small groups of stations isolated from the single largest group are considered to be ineffective.

Although one can draw a wide variety of networks, they all factor into two components: centralized (or star) and distributed (or grid or mesh). (See types (a) and (c), respectively, in Fig. 1.)

The centralized network is obviously vulnerable as destruction of a single central node destroys communication between the end stations. In practice, a mixture of star and mesh components is used to form communications networks. For example, type (b) in Fig. 1 shows the hierarchical structure of a set of stars connected in the form of a larger star with an additional link forming a

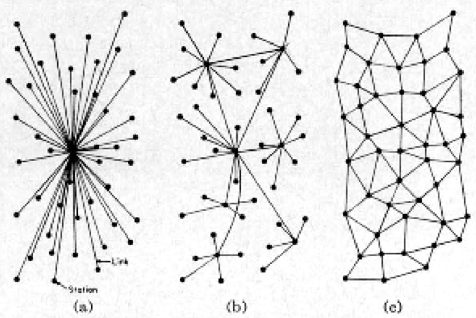


Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

loop. Such a network is sometimes called a "decentralized" network, because complete reliance upon a single point is not always required.

Examination of a Distributed Network

Since destruction of a small number of nodes in a decentralized network can destroy communications, the properties, problems, and hopes of building "distributed" communications networks are of paramount interest.

The term "redundancy level" is used as a measure of connectivity, as defined in Fig. 2. A minimum span network, one formed with the smallest number of links possible, is chosen as a reference point and is called "a network of redundancy level one." If two times as many links are used in a gridded network than in a minimum span network, the network is said to have a redundancy level of two. Fig. 2 defines connectivity of levels 1, $1\frac{1}{2}$, 2, 3, 4, 6 and 8. Redundancy level is equivalent to link-to-node ratio in an infinite size array of stations. Obviously, at levels above three there are alternate methods of constructing the network. However, it was found that there is little difference regardless of which method is used. Such an alternate method is shown for levels three and four, labelled R'. This specific alternate mode is also used for levels six and eight.[2]

Each node and link in the array of Fig. 2 has the capacity and the switching flexibility to allow transmission between any ith station and any jth station, provided a path can be drawn from the ith to the jth station.

Starting with a network composed of an array of stations connected as in Fig. 3, an assigned percentage of nodes and links is destroyed. If, after this operation,

Manuscript received October 9, 1963. This paper was presented at the First Congress of the Information Systems Sciences, sponsored by the MITRE Corporation, Bedford, Mass., and the USAF Electronic Systems Division, Hot Springs, Va., November, 1962.

The author is with The RAND Corporation, Santa Monica, Calif.

[1] Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors.

[2] See L. J. Craig, and I. S. Reed, "Overlapping Tessellated Communications Networks," The RAND Corporation, Santa Monica, Calif., paper P-2359; July 5, 1961.

HISTORICAL PERSPECTIVE

- 1989 — **Tim Berners-Lee** presented a proposal for an information management system that would enable sharing of resources over a **computer network**.
- **ARPANET** — open **decentralized** computer network architecture.
  - 1969: First message sent
  - 1971: First e-mail sent
  - 1973: File Transfer Protocol (RFC 354)
  - 1977: Network Voice Protocol (RFC 741)
  - 1981: Internet Protocol v4
  - 1987: number of hosts > 10 000

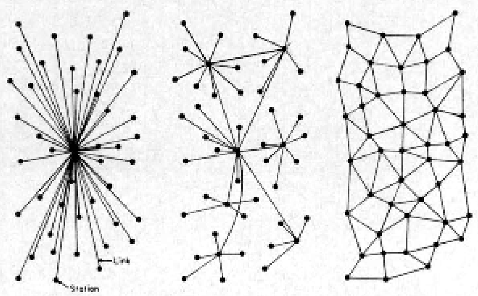
On Distributed Communications Networks

PAUL BARAN, SENIOR MEMBER, IEEE

Summary—This paper[1] briefly reviews the distributed communication network concept in which each station is connected to all adjacent stations rather than to a few switching points, as in a centralized system. The payoff for a distributed configuration in terms of survivability in the cases of enemy attack directed against nodes, links or combinations of nodes and links is demonstrated.

A comparison is made between diversity of assignment and perfect switching in distributed networks, and the feasibility of using low-cost unreliable communication links, even links so unreliable as to be unusable in present type networks, to form highly reliable networks is discussed.

The requirements for a future all-digital data distributed network which provides common user service for a wide range of users having different requirements is considered. The use of a standard format message block permits building relatively simple switching mechanisms using an adaptive store-and-forward routing policy to handle all forms of digital data including digital voice. This network rapidly responds to changes in the network status. Recent history of measured network traffic is used to modify path selection. Simulation results are shown to indicate that highly efficient routing can be performed by local control without the necessity for any central, and therefore vulnerable, control point.



(a) (b) (c)

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

Introduction

LET US CONSIDER the synthesis of a communication network which will allow several hundred major communications stations to talk with one another after an enemy attack. As a criterion of survivability we elect to use the percentage of stations both surviving the physical attack and remaining in electrical connection with the largest single group of surviving stations. This criterion is chosen as a conservative measure of the ability of the surviving stations to operate together as a coherent entity after the attack. This means that small groups of stations isolated from the single largest group are considered to be ineffective.

Although one can draw a wide variety of networks, they all factor into two components: centralized (or star) and distributed (or grid or mesh). (See types (a) and (c), respectively, in Fig. 1.)

The centralized network is obviously vulnerable as destruction of a single central node destroys communication between the end stations. In practice, a mixture of star and mesh components is used to form communications networks. For example, type (b) in Fig. 1 shows the hierarchical structure of a set of stars connected in the form of a larger star with an additional link forming a loop. Such a network is sometimes called a "decentralized" network, because complete reliance upon a single point is not always required.

Examination of a Distributed Network

Since destruction of a small number of nodes in a decentralized network can destroy communications, the properties, problems, and hopes of building "distributed" communications networks are of paramount interest.

The term "redundancy level" is used as a measure of connectivity, as defined in Fig. 2. A minimum span network, one formed with the smallest number of links possible, is chosen as a reference point and is called "a network of redundancy level one." If two times as many links are used in a gridded network than in a minimum span network, the network is said to have a redundancy level of two. Fig. 2 defines connectivity of levels 1, $1\frac{1}{2}$, 2, 3, 4, 6 and 8. Redundancy level is equivalent to link-to-node ratio in an infinite size array of stations. Obviously, at levels above three there are alternate methods of constructing the network. However, it was found that there is little difference regardless of which method is used. Such an alternate method is shown for levels three and four, labelled R'. This specific alternate mode is also used for levels six and eight.[2]

Each node and link in the array of Fig. 2 has the capacity and the switching flexibility to allow transmission between any ith station and any jth station, provided a path can be drawn from the ith to the jth station.

Starting with a network composed of an array of stations connected as in Fig. 3, an assigned percentage of nodes and links is destroyed. If, after this operation,

Manuscript received October 9, 1963. This paper was presented at the First Congress of the Information Systems Sciences, sponsored by the MITRE Corporation, Bedford, Mass., and the USAF Electronic Systems Division, Hot Springs, Va., November, 1962.

The author is with The RAND Corporation, Santa Monica, Calif.

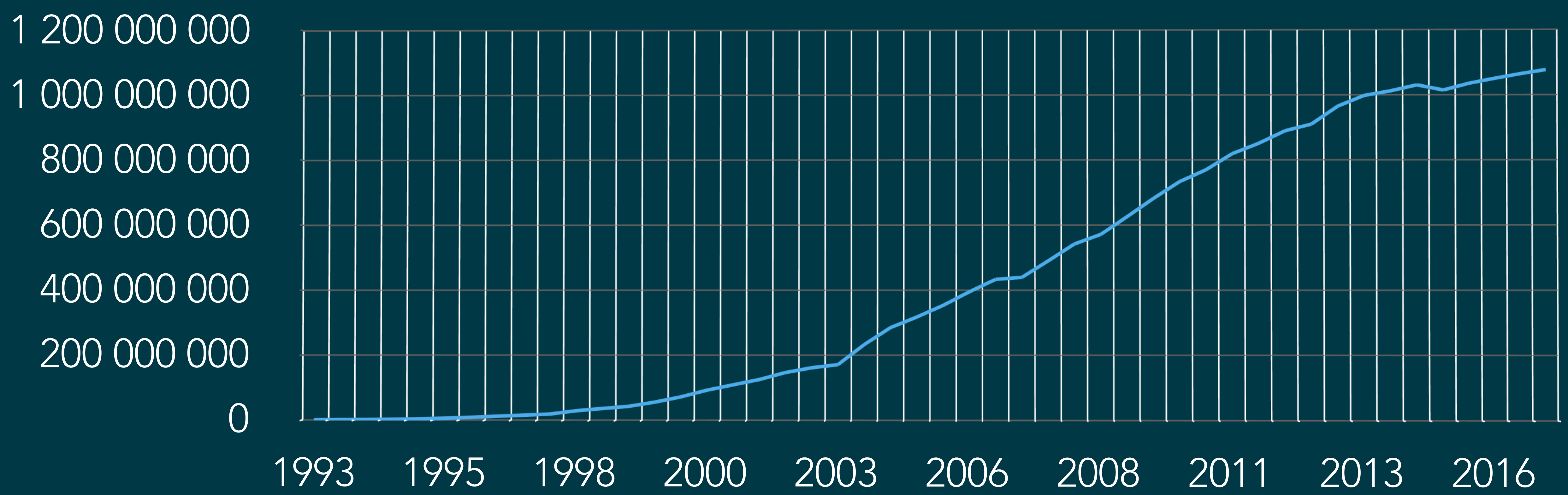
[1] Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors.

[2] See L. J. Craig, and I. S. Reed, "Overlapping Tessellated Communications Networks," The RAND Corporation, Santa Monica, Calif., paper P-2359; July 5, 1961.

NUMBER OF INTERNET HOSTS

INTERNET DOMAIN SURVEY
HTTPS://WWW.ISC.ORG/NETWORK/SURVEY/

NUMBER OF INTERNET HOSTS



INTERNET DOMAIN SURVEY

[HTTPS://WWW.ISC.ORG/NETWORK/SURVEY/](https://www.isc.org/network/survey/)

HISTORICAL PERSPECTIVE
WHAT HAS BEEN CHANGING?

- The Internet of Things (IoT)
  - The first **thing** (non-computer) connected to the Internet ever: the Toaster
    - It was connected to the Internet with TCP/IP networking and controlled with SNMP http://www.livinginternet.com/i/ia_myths_toast.htm
  - In 2008 the number of **things** connected to the Internet exceeded the number of people on earth.
    - Dave Evans, Cisco, The Internet of Things, http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf
- Web users and their expectations:
  - multiple platforms, simultaneous screening
  - consistent user experience — **responsive web design**
  - little or no latency — impatience of web users
    - Note: an average attention span of an Internet user shortened from 12sec in 2000 to 8sec in 2015
    - Less than a goldfish (9sec)!
    - http://time.com/3858309/attention-spans-goldfish/
- Technology:
  - shift of responsibility from the server to the client
  - http://evolutionoftheweb.com

HISTORICAL PERSPECTIVE
WHAT HAS BEEN CHANGING

- The Internet of Things (IoT)
  - The first **thing** (non-computer) connected to the Internet ever: t
    - It was connected to the Internet with TCP/IP networking and controlle
      http://www.livinginternet.com/i/ia_myths_toast.htm
  - In 2008 the number of **things** connected to the Internet exceed
    earth.
    - Dave Evans, Cisco, The Internet of Things,
      http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG
- Web users and their expectations:
  - multiple platforms, simultaneous screening
  - consistent user experience — **responsive web design**
  - little or no latency — impatience of web users
    - Note: an average attention span of an Internet user shortened from 12sec in 2000 to 8sec in 2015
    - Less than a goldfish (9sec)!
    - http://time.com/3858309/attention-spans-goldfish/
- Technology:
  - shift of responsibility from the server to the client
  - http://evolutionoftheweb.com

HISTORICAL PERSPECTIVE
WHAT HAS BEEN CHANGING?

- The Internet of Things (IoT)
  - The first **thing** (non-computer) connected to the Internet ever: the Toaster
    - It was connected to the Internet with TCP/IP networking and controlled with SNMP http://www.livinginternet.com/i/ia_myths_toast.htm
  - In 2008 the number of **things** connected to the Internet exceeded the number of people on earth.
    - Dave Evans, Cisco, The Internet of Things, http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf
- Web users and their expectations:
  - multiple platforms, simultaneous screening
  - consistent user experience — **responsive web design**
  - little or no latency — impatience of web users
    - Note: an average attention span of an Internet user shortened from 12sec in 2000 to 8sec in 2015
    - Less than a goldfish (9sec)!
    - http://time.com/3858309/attention-spans-goldfish/
- Technology:
  - shift of responsibility from the server to the client
  - http://evolutionoftheweb.com

WHAT HAS NOT CHANGED?
— BUILDING BLOCKS OF THE WEB

WHAT HAS NOT CHANGED?
— BUILDING BLOCKS OF THE WEB

1. **HTML** — a markup language for formatting and publishing **hypertext** documents.

WHAT HAS NOT CHANGED? — BUILDING BLOCKS OF THE WEB

1. **HTML** — a markup language for formatting and publishing **hypertext** documents.

2. **URL** — a uniform notation scheme for uniquely identifying accessible **resources** over the network.

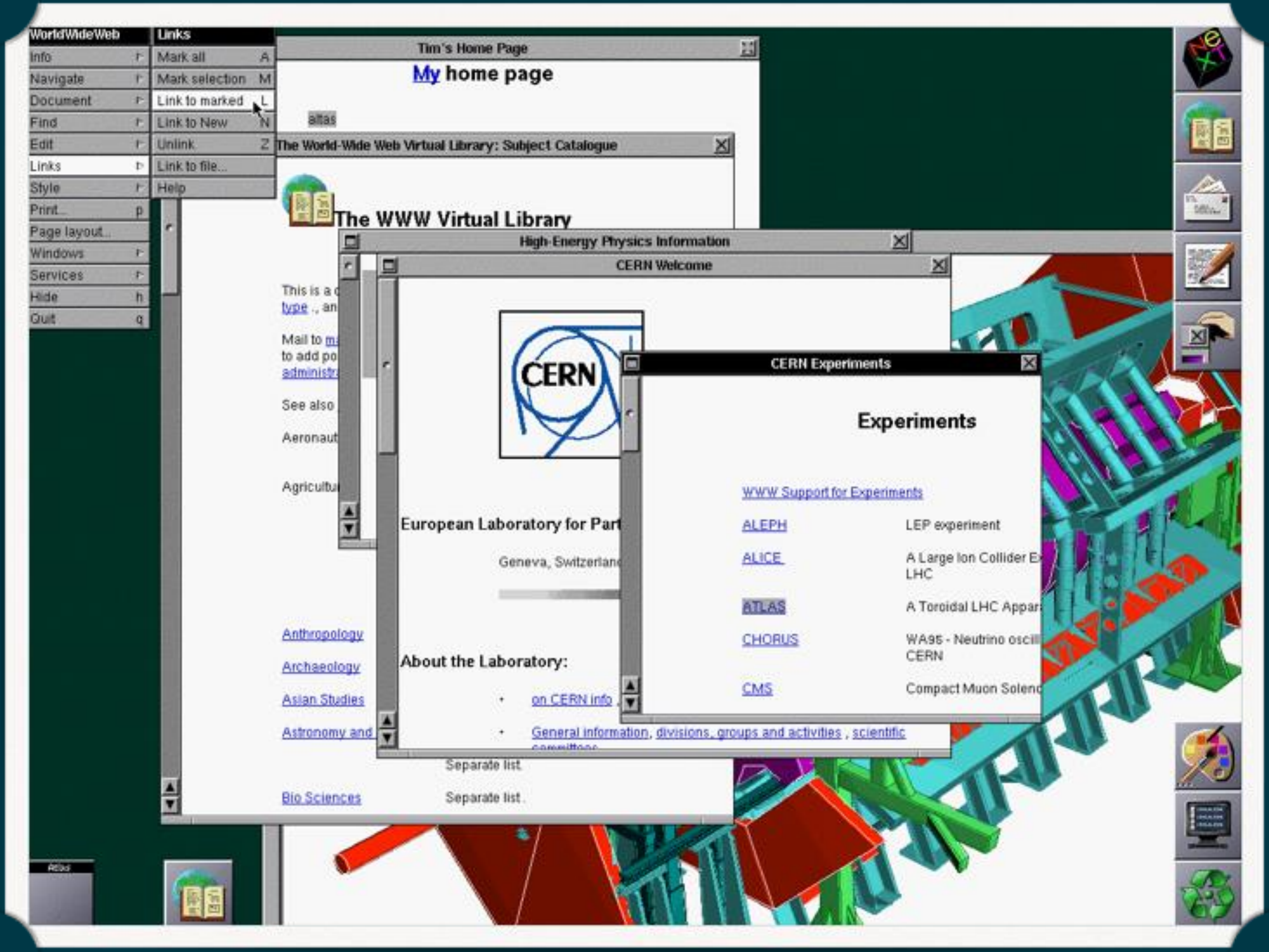
WHAT HAS NOT CHANGED?
— BUILDING BLOCKS OF THE WEB

1. **HTML** — a markup language for formatting and publishing **hypertext** documents.

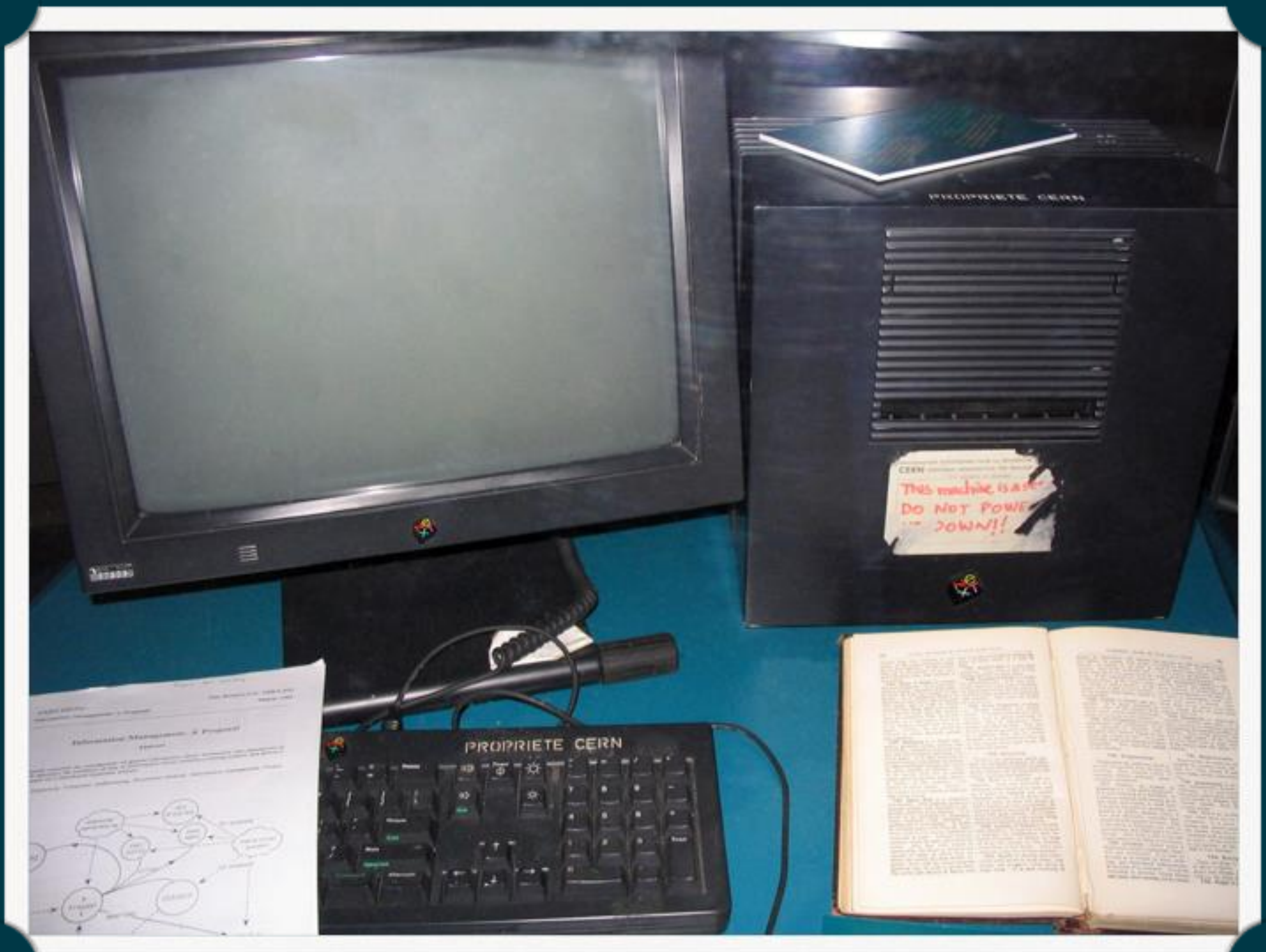
2. **URL** — a uniform notation scheme for uniquely identifying accessible **resources** over the network.

3. **HTTP** — a protocol for **transporting** messages (requests and responses) over the network.

BUILDING BLOCKS OF THE WEB
WORLDWIDEWEB BROWSER



BUILDING BLOCKS OF THE WEB
HTTPD WEB SERVER



HYPERTEXT MARKUP LANGUAGE

- HTML was defined by Tim Berners-Lee as an application of the Standard Generalized Markup Language (SGML).
- HTML is used for describing both the content and the structure of web pages.
- HTML is a markup language that web browsers use to interpret and compose text, images and other material into web pages.

HYPERTEXT MARKUP LANGUAGE

- Elements of HTML structure
  - Headings, paragraphs, tables, lists, photos, etc.
  - **Hyperlinks**
  - **Design forms** for conducting transactions with remote services, for use in searching for information, making reservations, ordering products.

—HTTP://WWW.W3.ORG/STANDARDS/WEBDESIGN/HTMLCSS

UNIFORM RESOURCE **IDENTIFIER**

- *A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an **abstract** or **physical** resource.*
- *A Uniform Resource Identifier (URI) provides a simple and extensible means for identifying a resource.*
- *The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanis (e.g., its network "location").*

—UNIFORM RESOURCE IDENTIFIER (URI): GENERIC SYNTAX (RFC 3986)

UNIFORM RESOURCE **LOCATOR**

```
scheme://[user:password@]host[:port]/path/…/[?query-string][#fragment]
```

- **scheme** — protocol used to connect to the server
  - // are optional in some protocols and compulsory in others
- **user:password** – optional user name and password for authentication
- **host** — IP address of the server (or its domain name)
- **port** — optional port number to which the target server listens
- **path** — path to the desired resource on the server
- **query-string** — key=value dynamic parameters separated by &
- **fragment** — positional marker within the requested document

UNIFORM RESOURCE LOCATOR

- Schemes
  - Case insensitive
  - By convention lowercase
  - May use +, ., -
- Path
  - Must begin with a single slash (/) if host is present
  - Must not begin with two slashes (//)
- Permitted characters in variable parts
  - Lowercase and uppercase letters
  - Arabic numbers
  - ASCII encoding
  - Other symbols must be octet-encoded (e.g., %26 instead of &)

HYPERTEXT TRANSFER PROTOCOL

- The Hypertext Transfer Protocol (HTTP)
  - Application-level
  - Textual
  - Stateless & sessionless
    - No requirement for persistent connection
    - Source of troubles for software developers
  - For distributed, collaborative, hypermedia information systems
  - Can be used for many tasks beyond its use for hypertext
- HTTP has been in use by the World-Wide Web global information initiative since 1990. HTTP/1.1 (RFC 2616) in wide use since 1999. HTTP/2.0 (RFC 7540) introduced in 2015.

HYPERTEXT TRANSFER PROTOCOL

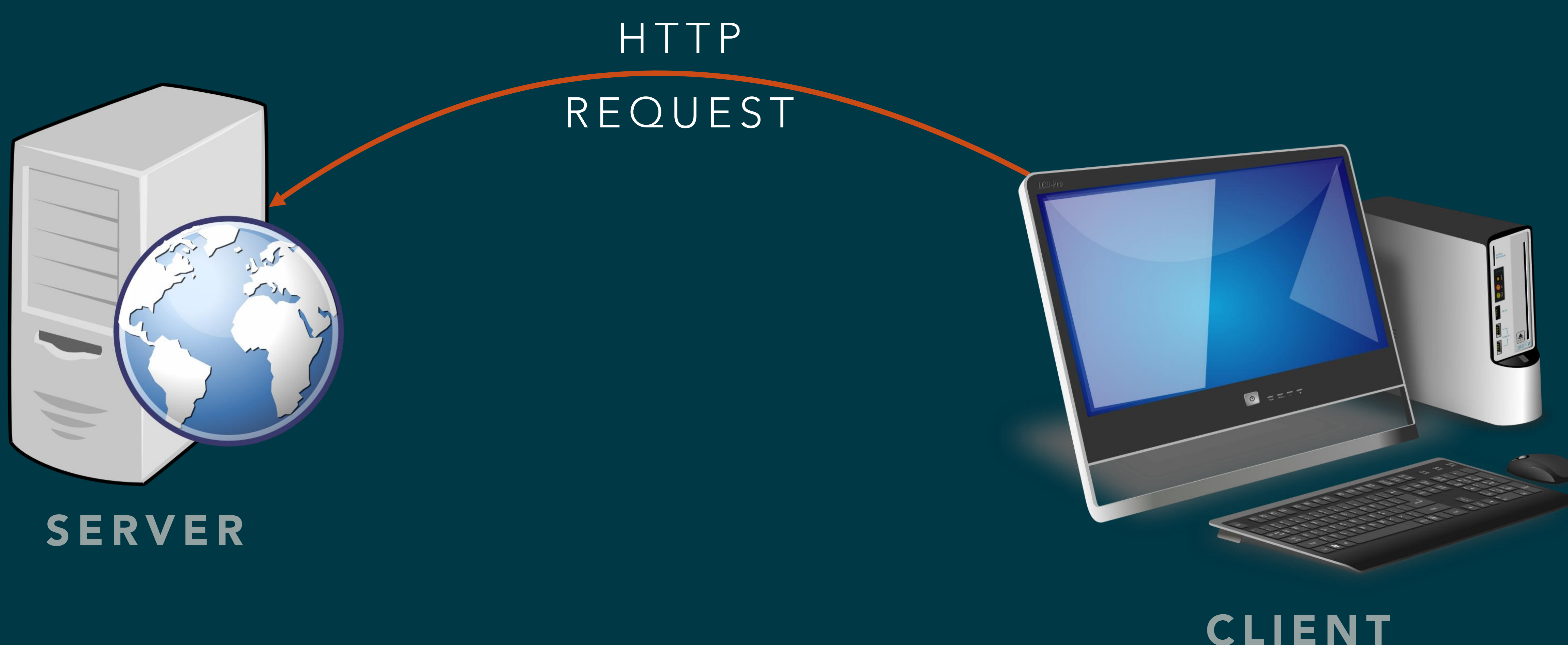
- In the following I will describe HTTP/1.1
- Then show what changed in HTTP/2.0

THE CLIENT—SERVER ARCHITECTURE

- The HTTP protocol uses the **request-response paradigm**.
- An **HTTP transaction** consists of a single request from a client, followed by a single response from the server.

THE CLIENT—SERVER ARCHITECTURE

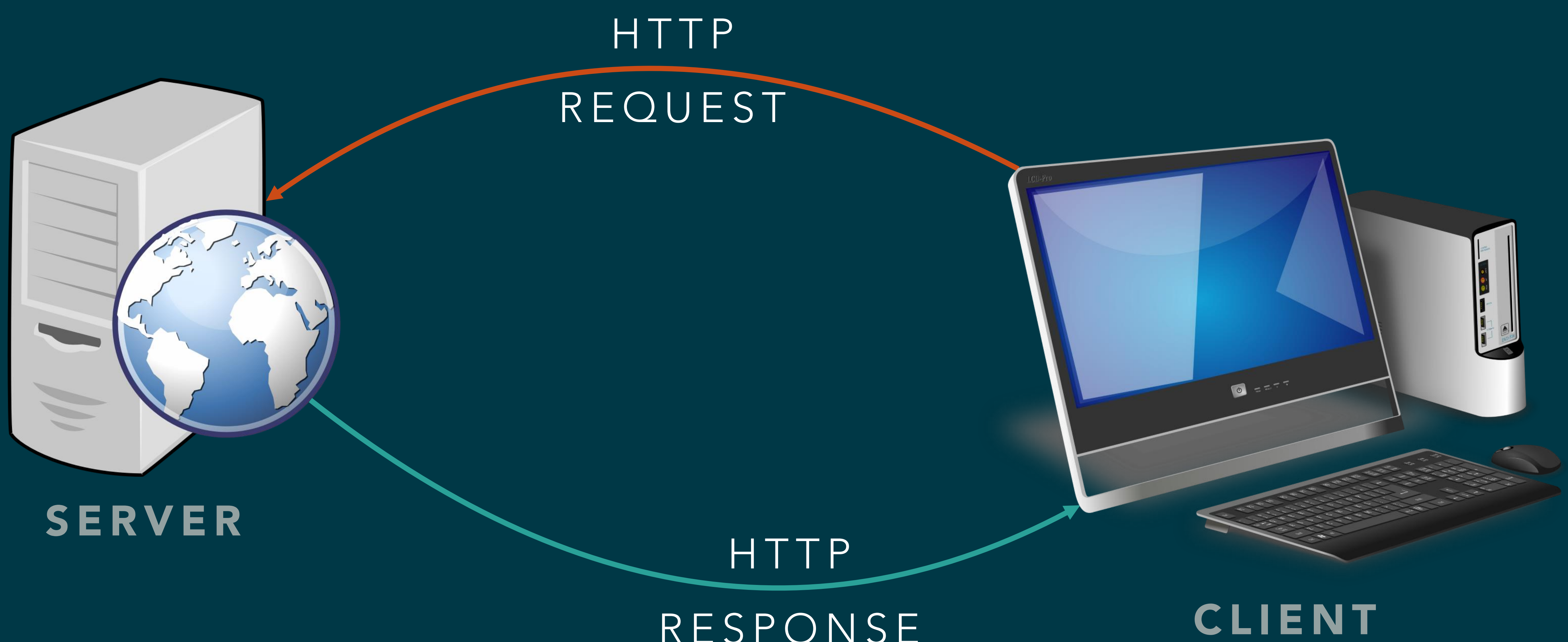
- The HTTP protocol uses the **request-response paradigm**.
- An **HTTP transaction** consists of a single request from a client, followed by a single response from the server.



THE CLIENT—SERVER ARCHITECTURE

- The HTTP protocol uses the **request-response paradigm**.

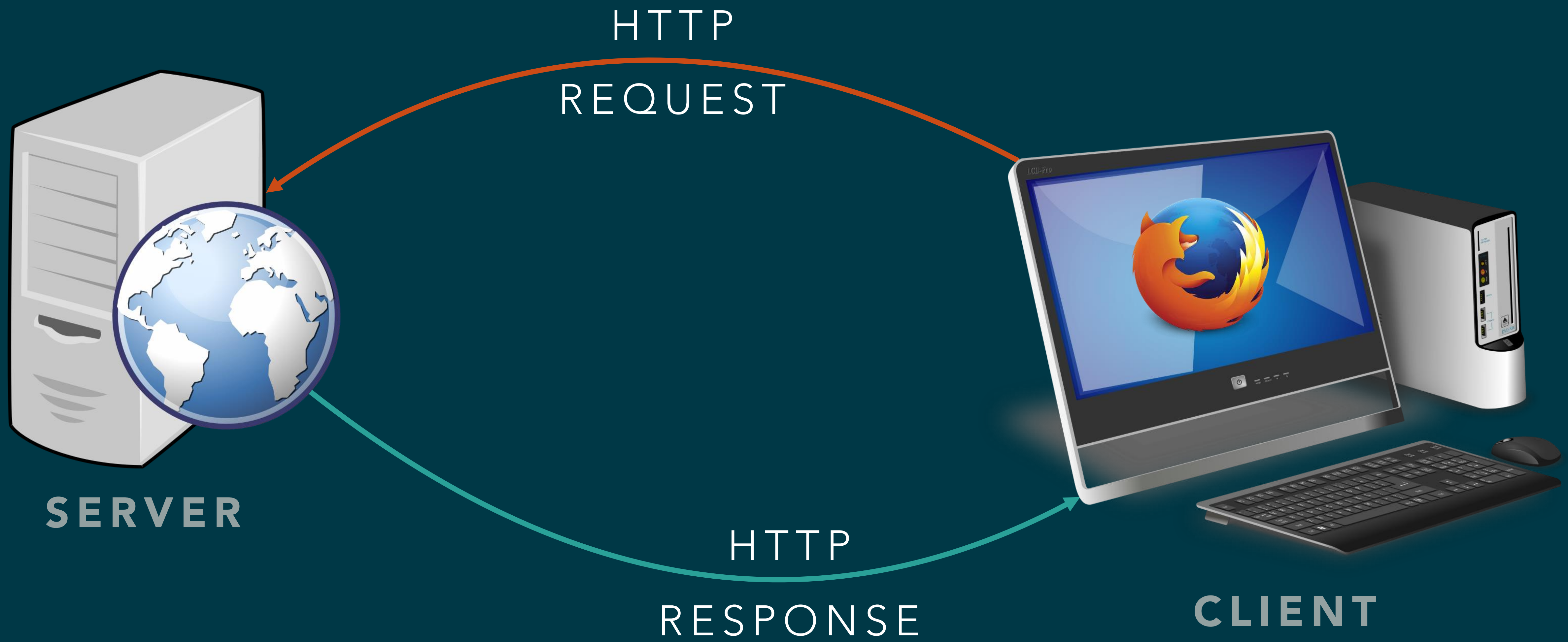
- An **HTTP transaction** consists of a single request from a client, followed by a single response from the server.



THE CLIENT—SERVER ARCHITECTURE

- The HTTP protocol uses the **request-response paradigm**.

- An **HTTP transaction** consists of a single request from a client, followed by a single response from the server.



HTTP MESSAGES STRUCTURE

- HTTP messages are simple, formatted blocks of data.
- Requests and response have a similar structure:

```
message = {start-line}\r\n
          ({message-header}\r\n)*
          \r\n
          {message-body}

{start-line} = Request-Line | Status-Line
{message-header} = Field-Name ':' Field-Value

Request-Line = {method} {URI} HTTP/{version}
Status-Line = HTTP/{version} {status} {explanation}
```

HTTP REQUEST METHODS (VERBS)

- **GET** — retrieves the specified resource. GET does not have a body and, until HTTP 1.1, was not required to have headers.

```
GET /standards/ HTTP/1.1
Host: www.w3.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:35.0)
```

- **POST** — requests that the target resource processes the data enclosed in the message body according to specific semantics.
- **PUT** — requests that the state of the resource be created or replaced with the state enclosed in the message body.
- **DELETE** — deletes the specified resource.

HTTP REQUEST METHODS (VERBS)

- **HEAD** — functionally similar to **GET**, except that the server responds without message body. It's used to retrieve the server headers for a particular resource, generally to check if the resource has changed, via timestamps.
- **OPTIONS** — Returns the HTTP methods that the server supports for the specified URL.
- **TRACE** — Echoes back the received request so that a client can see what (if any) changes or additions have been made by intermediate servers. Each intermediate proxy or gateway would inject its IP or DNS name into the `Via` header field. This can be used for diagnostic purposes.

HTTP REQUEST METHODS (VERBS)

- **PATCH** — Updates portion of resource at the given URL (RFC 5789)
- **CONNECT** — Establish a tunnel to the server identified by the target resource. It is intended only for use in requests to a proxy (RFC 2817)

HTTP METHOD PROPERTIES

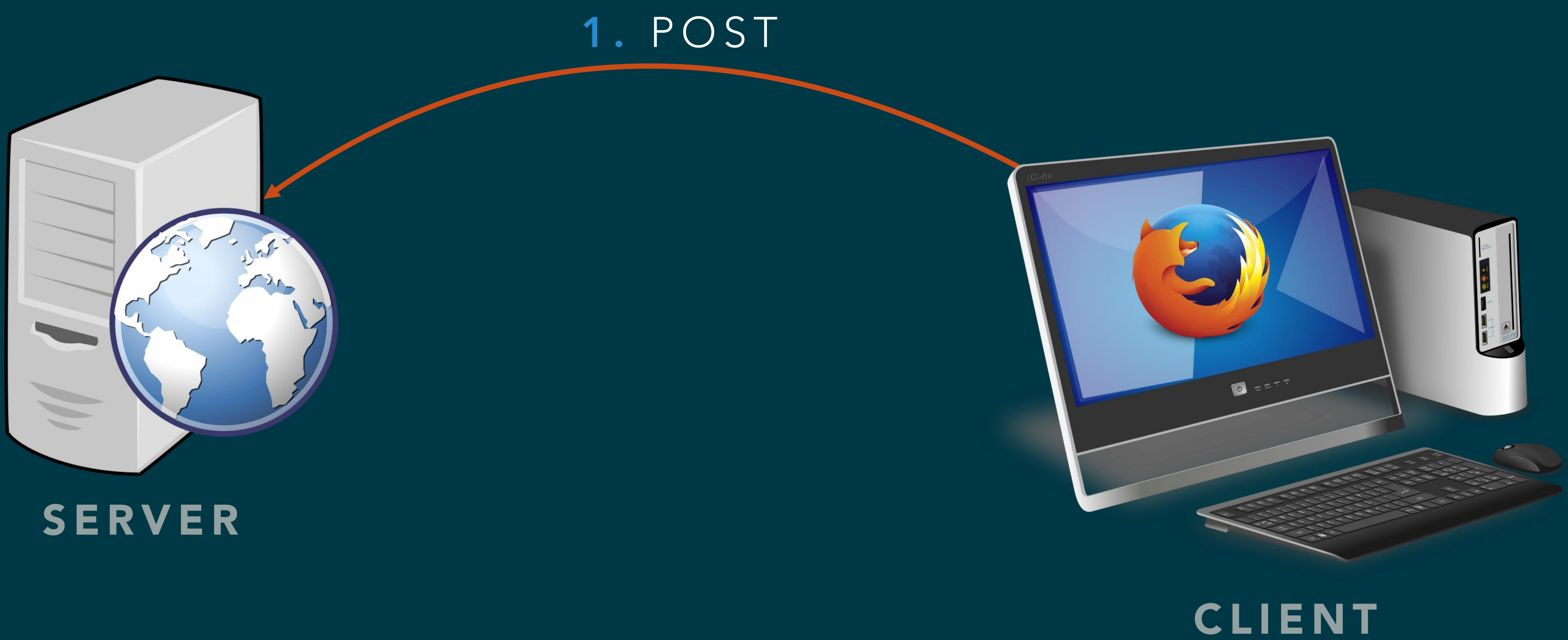
- A method is **"safe"** if its defined semantics is essentially **read-only**. Safe methods **does not** change the state of the server:
  - **GET**
  - **HEAD**
  - **OPTIONS**
  - **TRACE**
  - **CONNECT**
- A method is considered **"idempotent"** if the effect of multiple identical requests with that method is the same as the effect for a single such request:
  - safe methods
  - **PUT**
  - **DELETE**
  - **PATCH** (optionally, in conjunction with ETag, see RFC 5789 for details)

POST IS NOT SAFE, NOR IDEMPOTENT

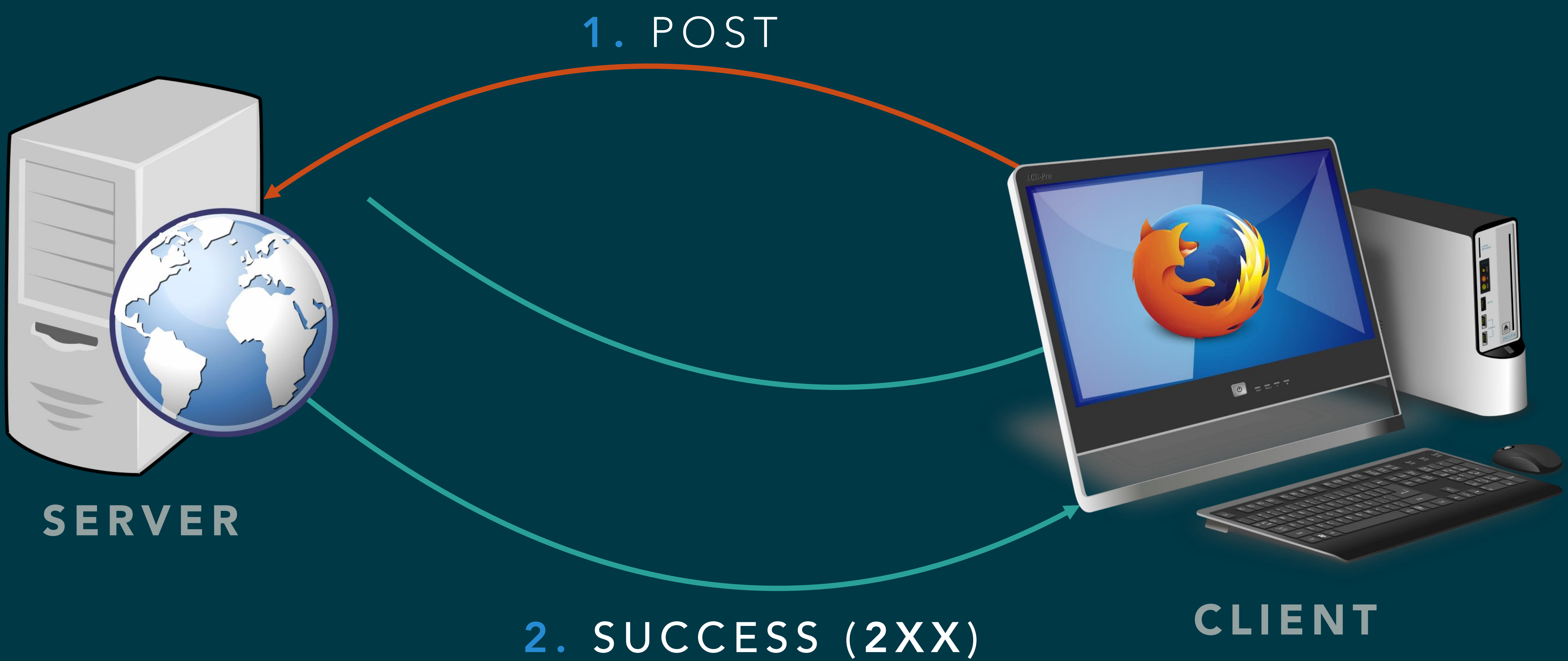
SERVER

CLIENT

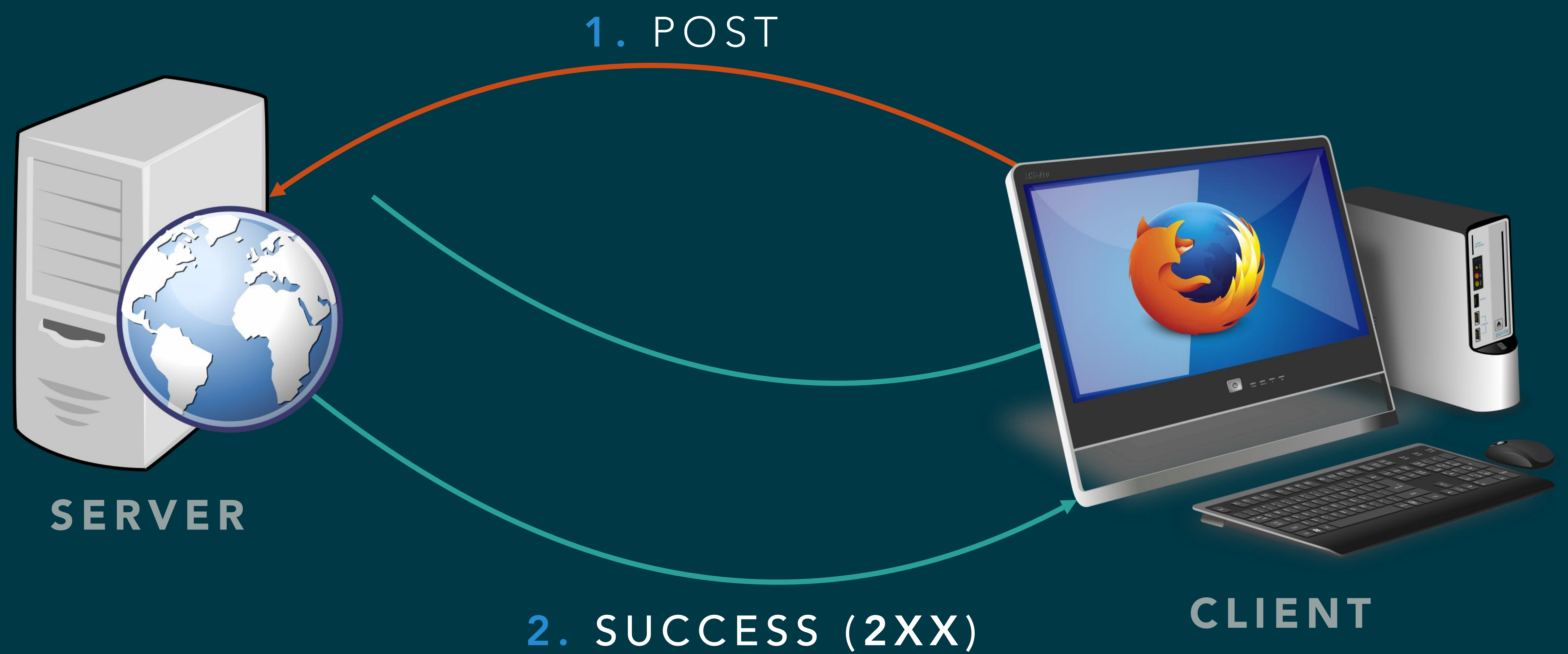
POST IS NOT SAFE, NOR IDEMPOTENT



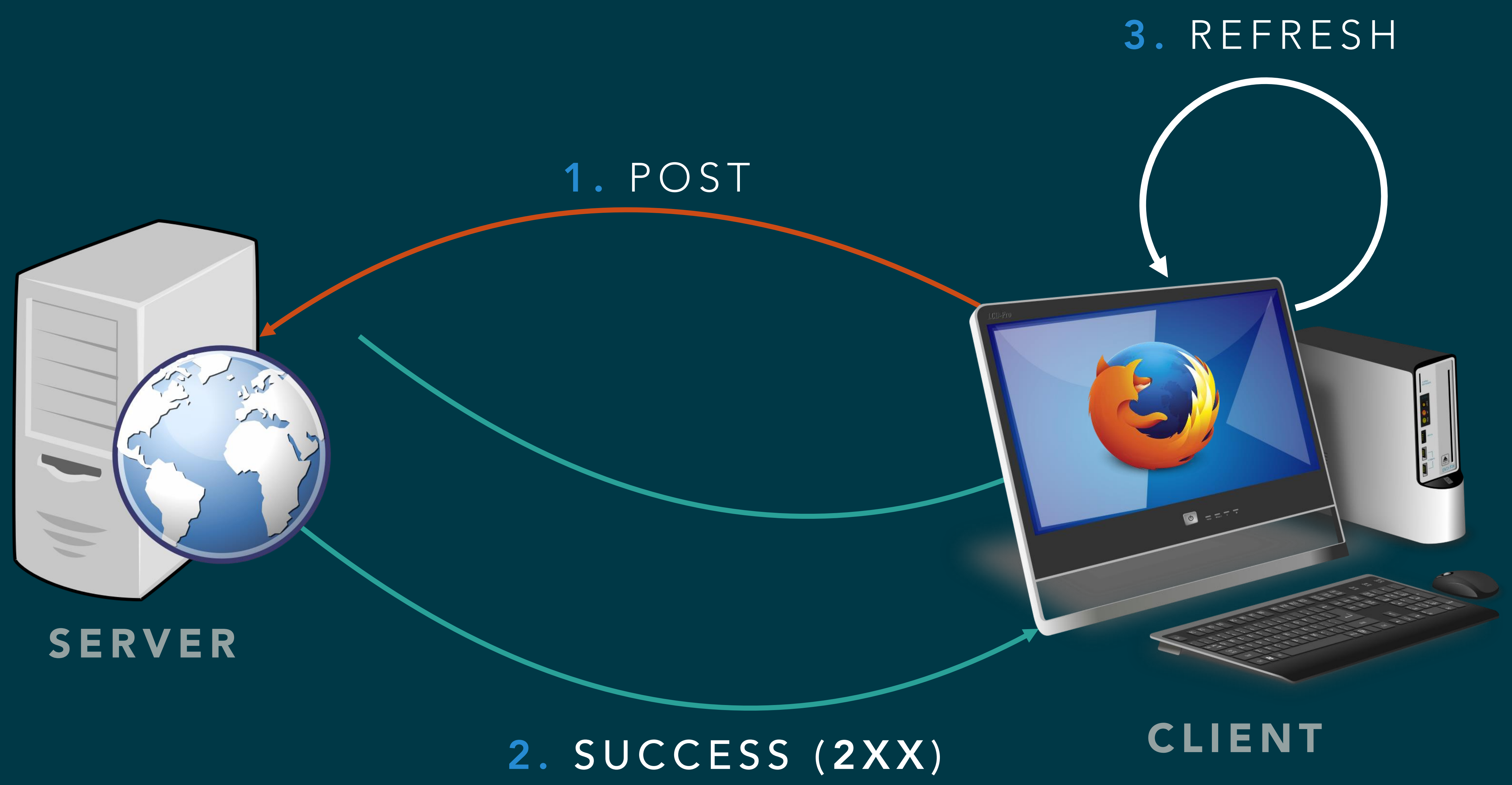
POST IS NOT SAFE, NOR IDEMPOTENT



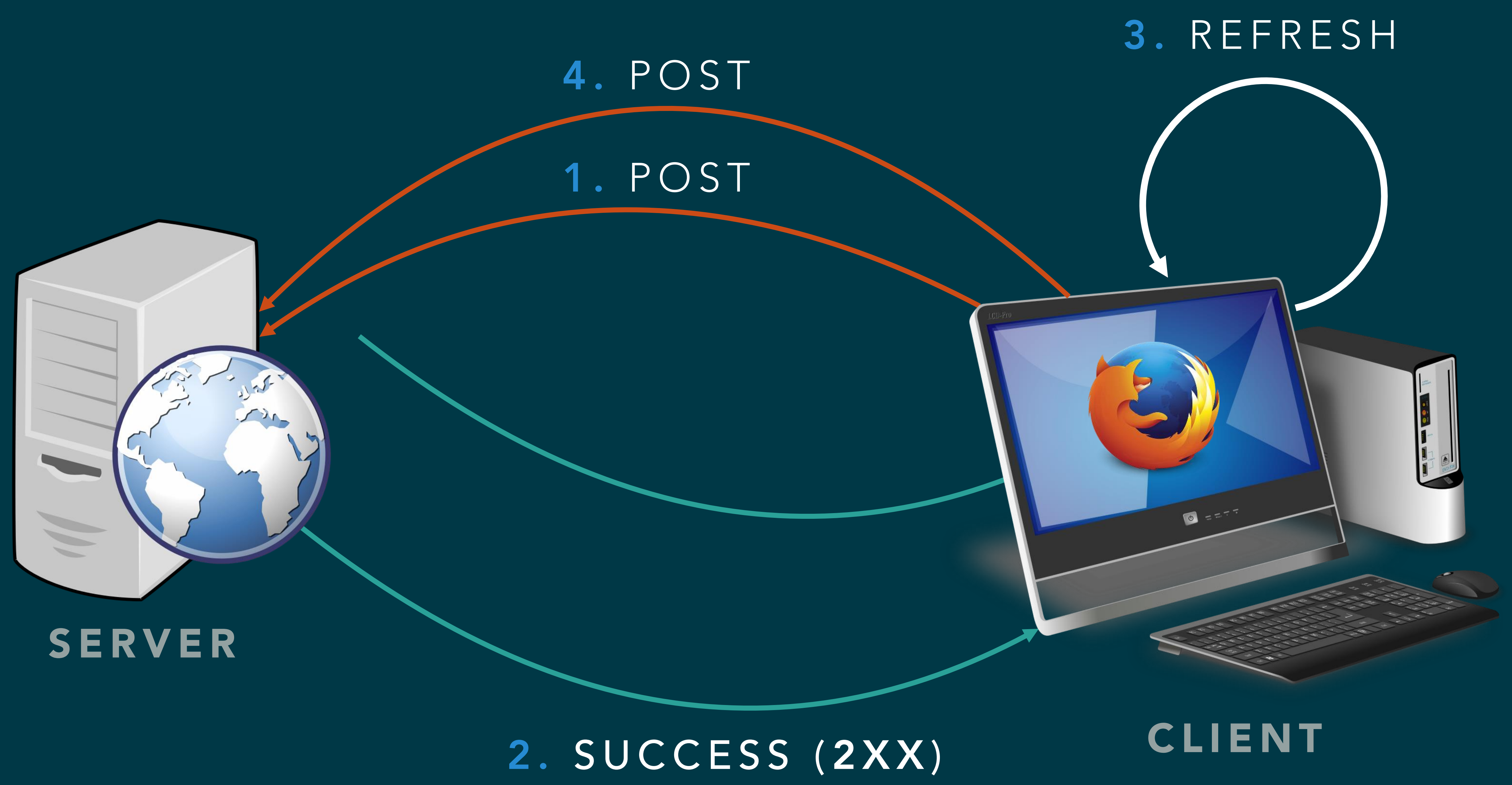
POST IS NOT SAFE, NOR IDEMPOTENT



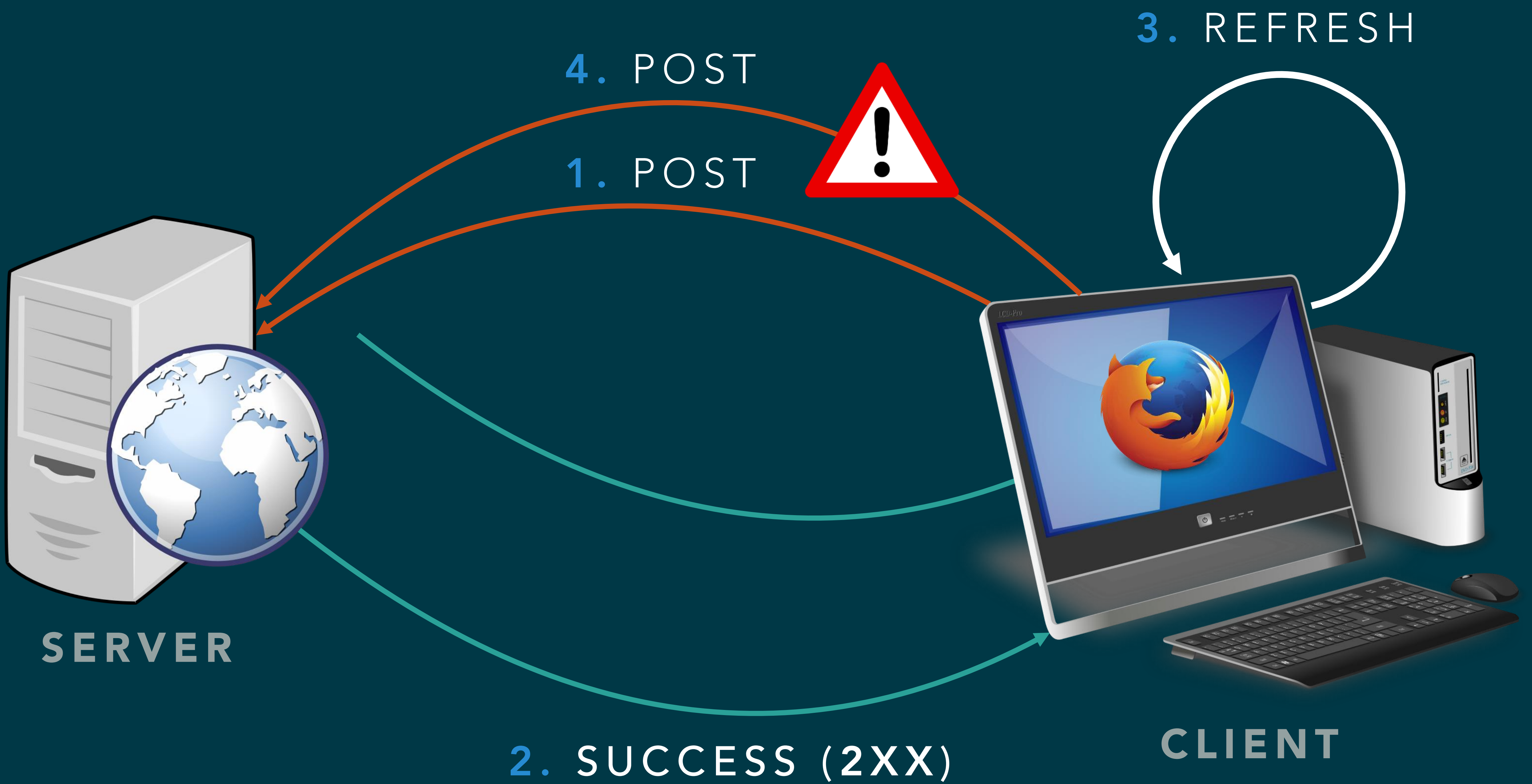
POST IS NOT SAFE, NOR IDEMPOTENT

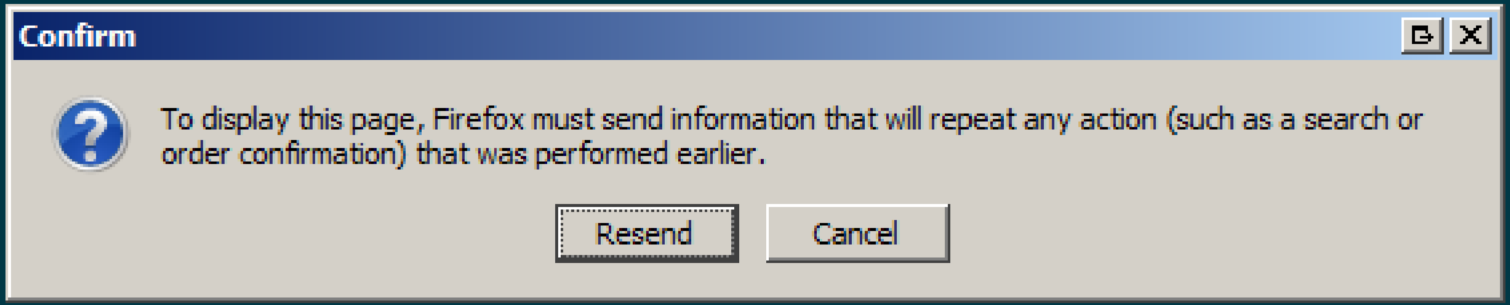


POST IS NOT SAFE, NOR IDEMPOTENT



POST IS NOT SAFE, NOR IDEMPOTENT





HTTP RESPONSE STATUS CODES

- **1xx** — Informational — does not include message body.
  - **100** — Continue
  - **101** — Switching protocols

HTTP RESPONSE STATUS CODES

- **1xx** — Informational — does not include message body.
  - **100** — Continue
  - **101** — Switching protocols
- **2xx** — Successful — the action requested by the client was received, understood, accepted and processed successfully.
  - **200** — OK
  - **201** — Created
  - **202** — Accepted

HTTP RESPONSE STATUS CODES

- **1xx** — Informational — does not include message body.
  - **100** — Continue
  - **101** — Switching protocols
- **2xx** — Successful — the action requested by the client was received, understood, accepted and processed successfully.
  - **200** — OK
  - **201** — Created
  - **202** — Accepted
- **3xx** — Redirection — the client must take additional action to complete the request.
  - **301** — Moved permanently
  - **302** — Found (in HTTP/1.0: Moved temporarily)
  - **303** — See other (changes method to GET)
  - **304** — Not modified
  - **307** — Temporary Redirect (does not change method)
  - **308** — Permanent Redirect (does not change method)

HTTP REDIRECTION PURPOSES

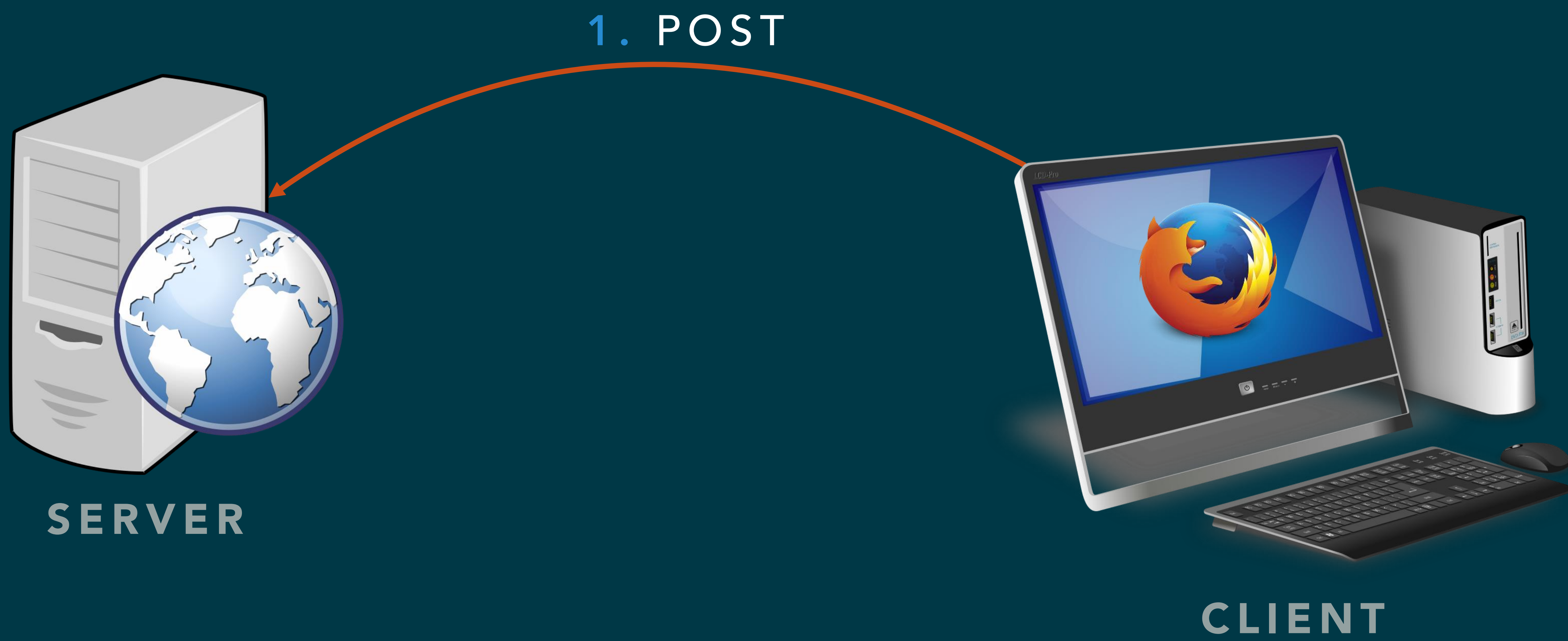
- Similar domain names
  - wikipedia.net, wikipedia.org, wikipedia.com
- URL shortening services
  - http://goo.gl, http://bitly.com
- Request to a directory without terminating slash
  - http://www.cs.put.poznan.pl/mszubert
- Redirecting users to a login page (**301** vs. **302**).
- Post/Redirection/Get

POST / REDIRECT / GET

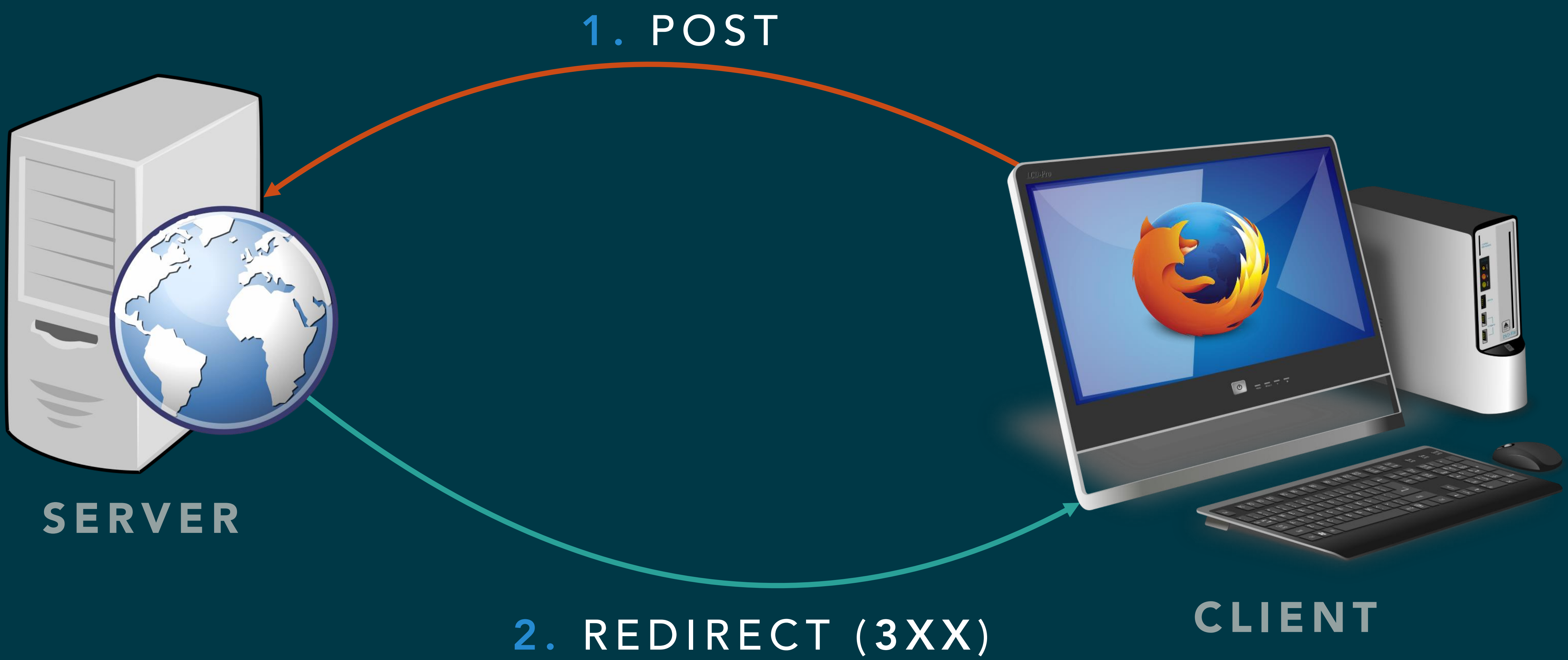
SERVER

CLIENT

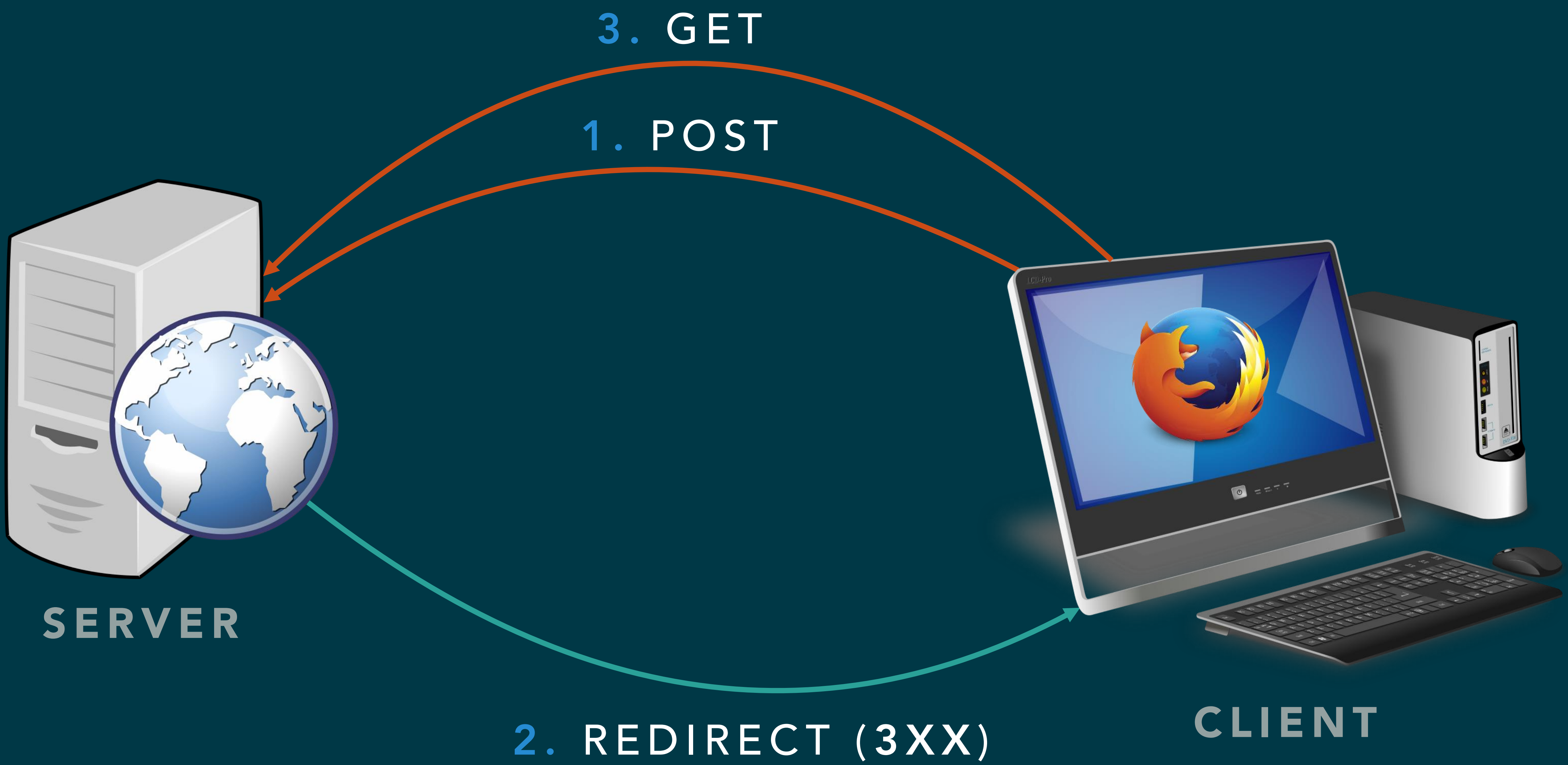
POST / REDIRECT / GET



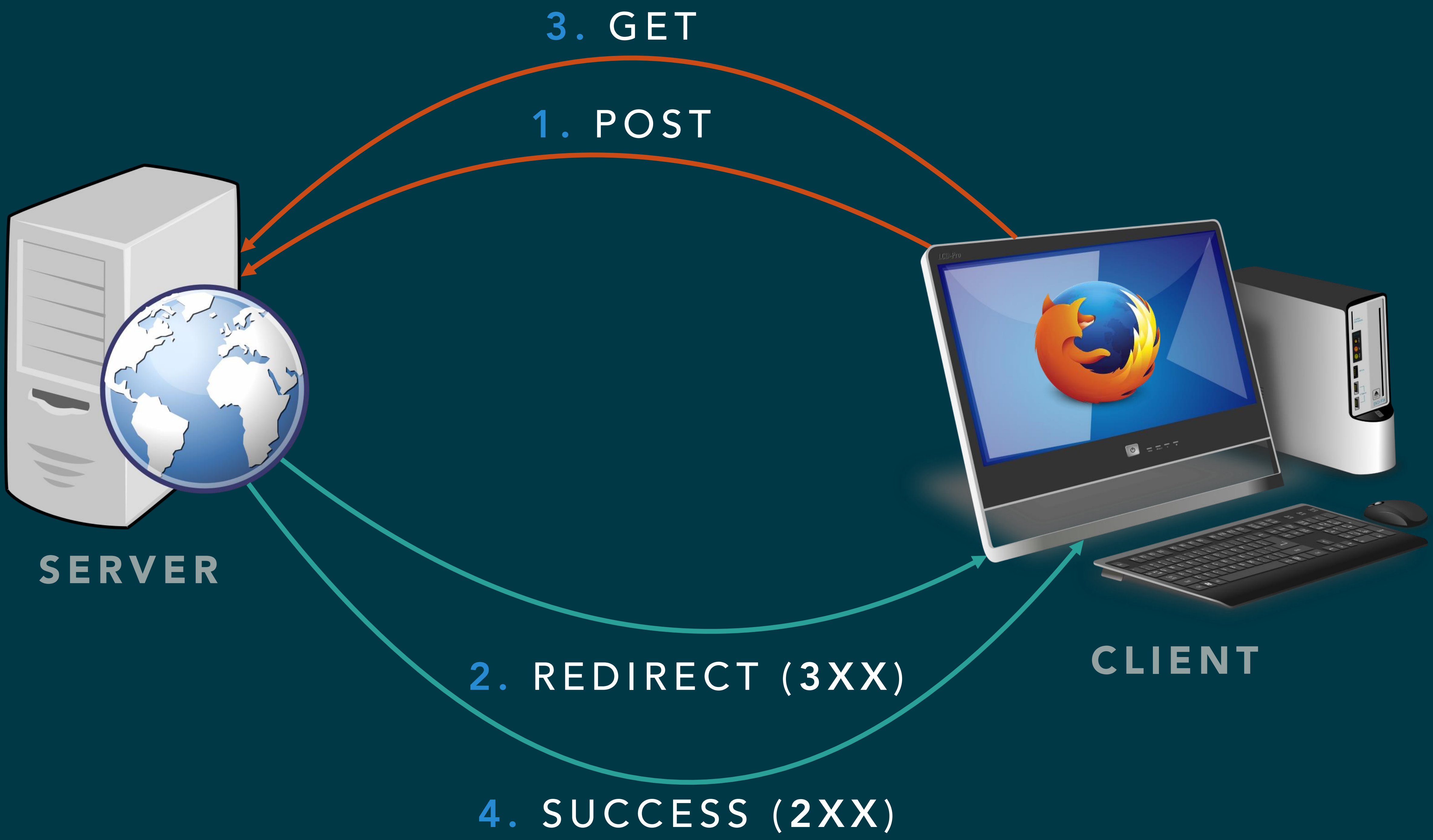
POST / REDIRECT / GET



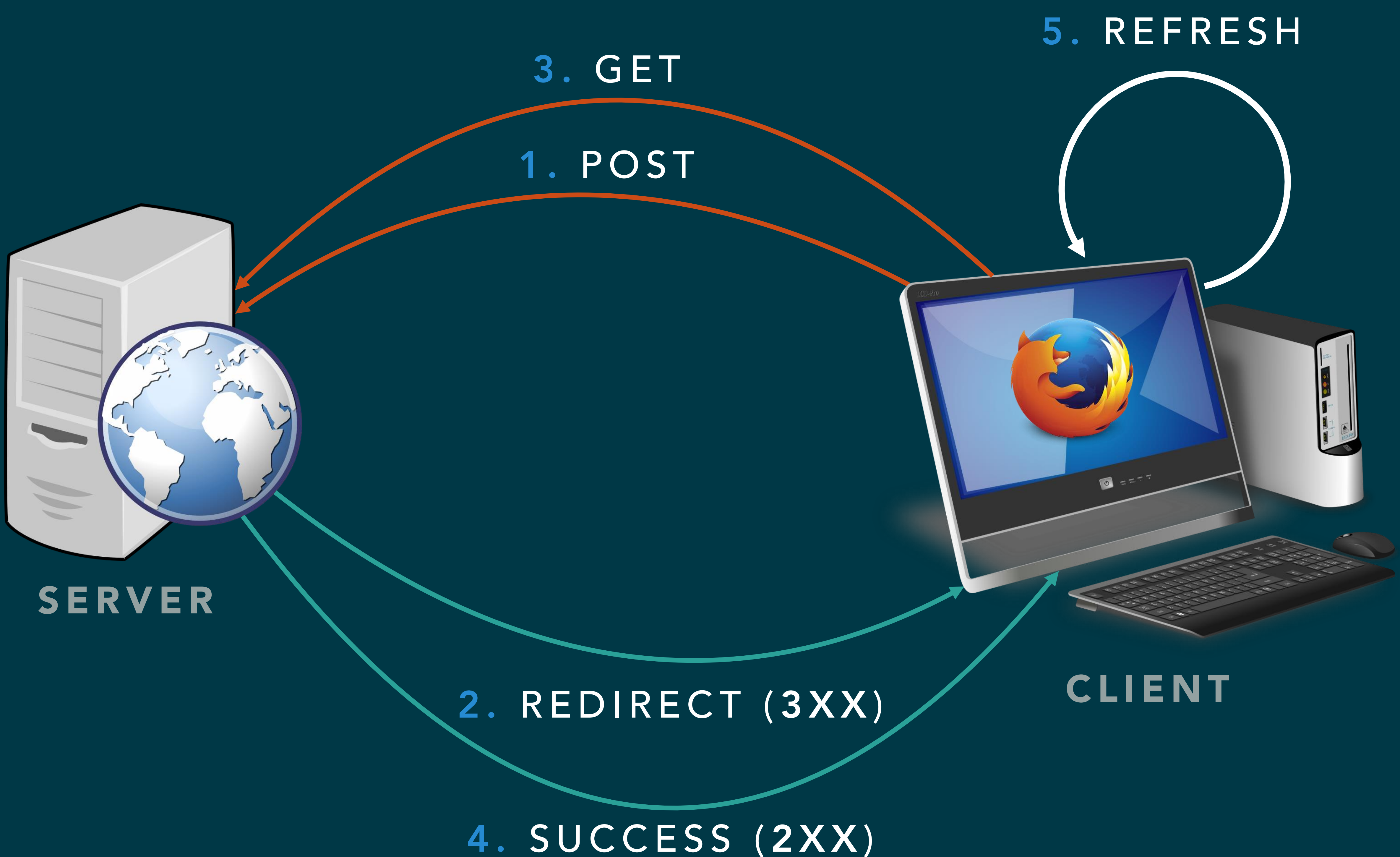
POST / REDIRECT / GET



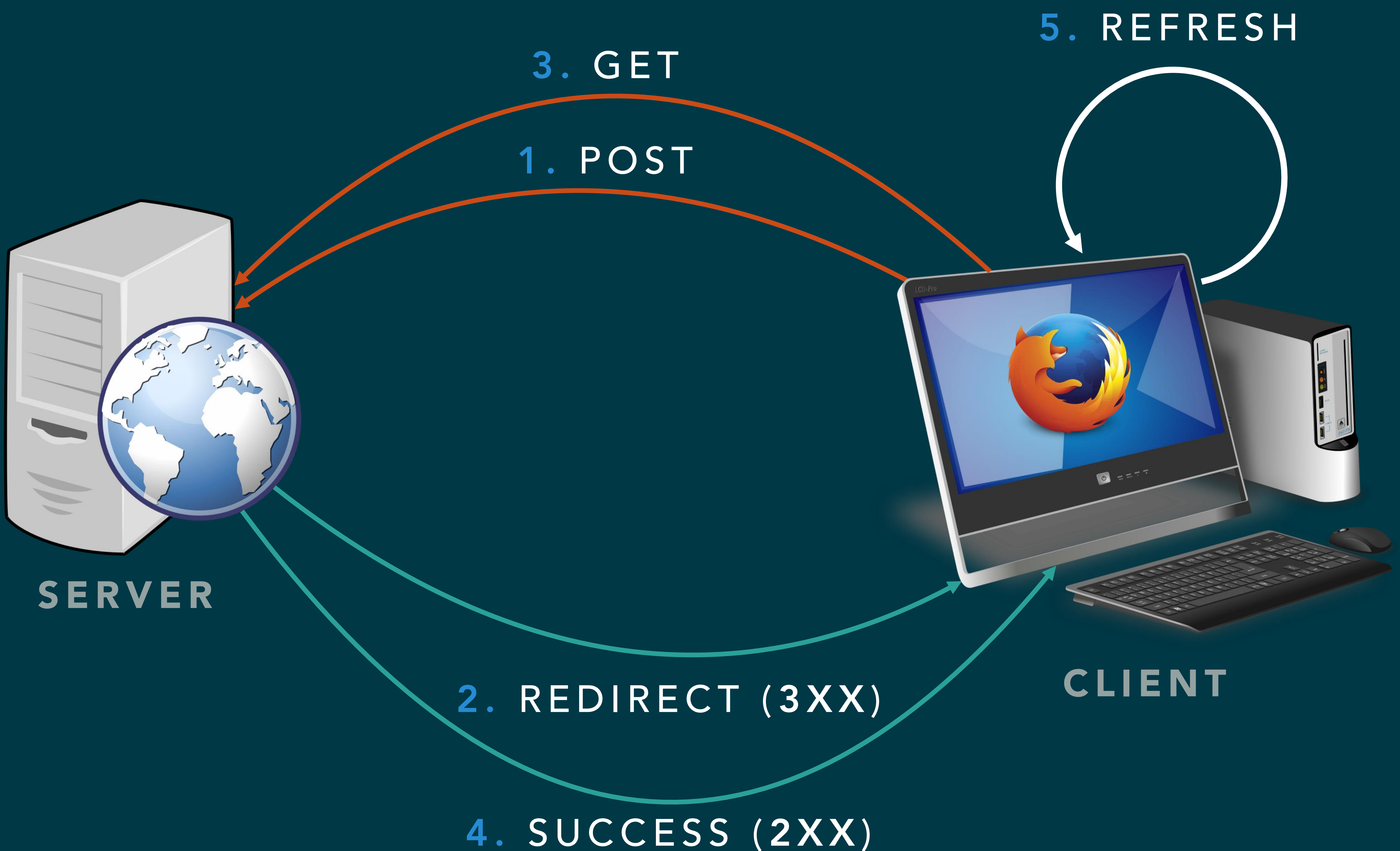
POST / REDIRECT / GET



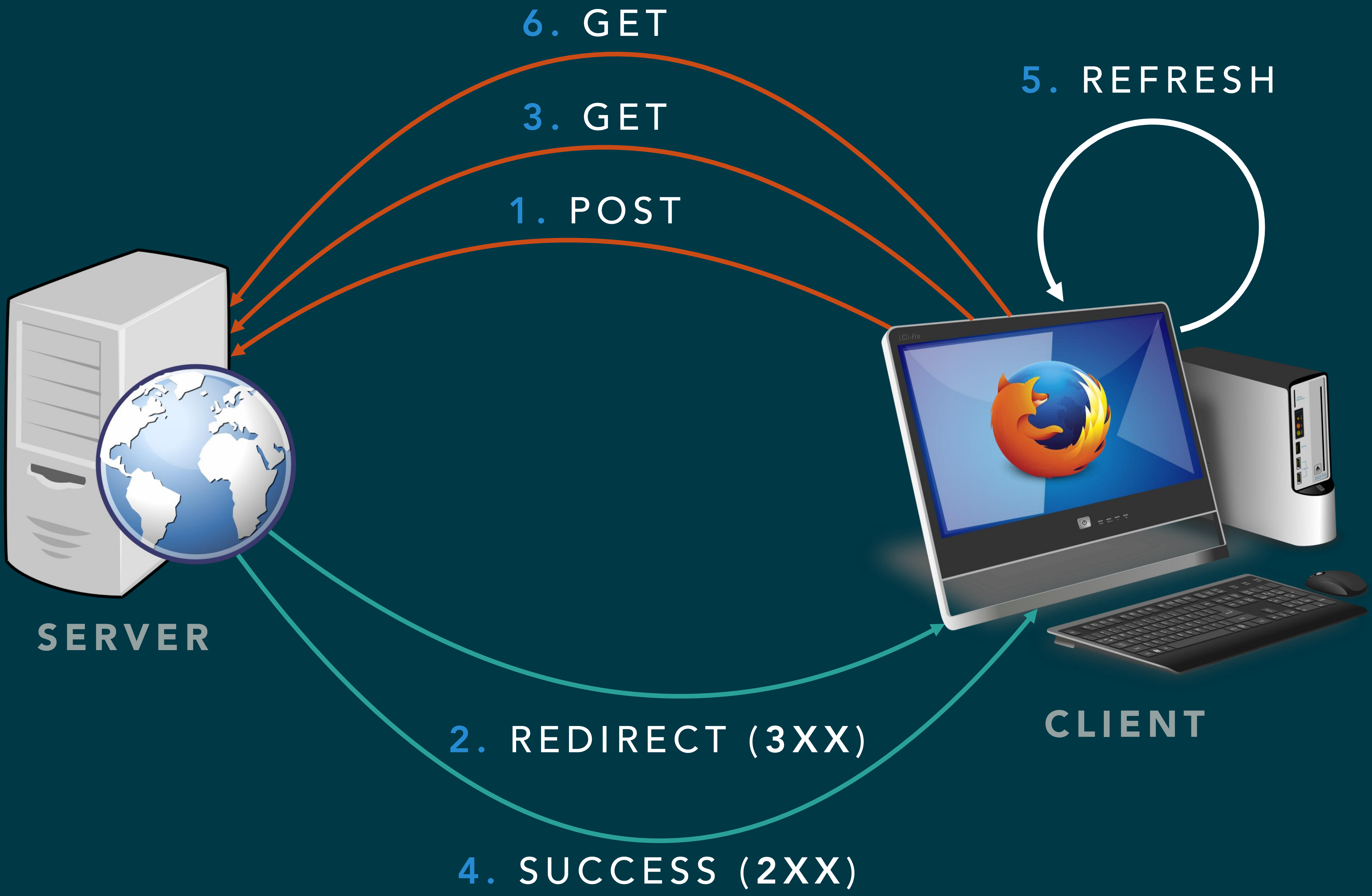
POST / REDIRECT / GET



POST / REDIRECT / GET



POST / REDIRECT / GET



HTTP RESPONSE ERROR CODES

- **4xx** — Client Error — the client failed either by requesting an invalid resource or making a bad request.
  - **400** — Bad request
  - **401** — Not authorized
  - **402** — Payment required
  - **403** — Forbidden
  - **404** — Not found
  - **405** — Method not allowed
  - **406** — Not acceptable
  - **418** — I'm a teapot

HTTP RESPONSE ERROR CODES

- **4xx** — Client Error — the client failed either by requesting an invalid resource or making a bad request.
  - **400** — Bad request
  - **401** — Not authorized
  - **402** — Payment required
  - **403** — Forbidden
  - **404** — Not found
  - **405** — Method not allowed
  - **406** — Not acceptable
  - **418** — I'm a teapot
- **5xx** — Server Error — the server failed to fulfill a valid request.
  - **500** — Internal server error
  - **501** — Not implemented
  - **503** — Service unavailable

HTTP MESSAGES STRUCTURE

```
message = {start-line}\r\n
          ({message-header}\r\n)*
          \r\n
          {message-body}

{start-line} = Request-Line | Status-Line
{message-header} = Field-Name ':' Field-Value
```

HTTP MESSAGES STRUCTURE

```
message = {start-line}\r\n
          ({message-header}\r\n)*
          \r\n
          {message-body}

{start-line} = Request-Line | Status-Line
{message-header} = Field-Name ':' Field-Value
```

- Headers are a form of message **metadata** and are broadly classified into:
    - general headers
    - request-specific headers
    - response-specific headers
    - entity headers

USES OF HEADERS

- Informational
- Virtual hosting
- Content negotiation
- Client identification
- Authentication
- Caching

GENERAL HEADERS

- `Date` — provides a date and time stamp telling when the message was created.
- `Via` — shows what intermediaries (proxies, gateways) the message has gone through.
- `Connection` — allows clients and servers to specify options about the request/response connection.
- `Cache-Control` — used to pass caching directions along with the message.
- `Transfer-Encoding` — used to compress or to break the response into smaller parts (with the `chunked` value).

INFORMATIONAL REQUEST HEADERS

- `Host` — gives the hostname and port to which the request is being sent; introduced to enable a single server to service multiple domains (virtual hosting).
- `Referer` — identifies the address of the webpage that linked to the resource being requested.
- `User-agent` — tells the server the name of the application making the request.
- `From` — the email address of the user making the request.
- `Client-IP` — the IP address of the client's machine.

INFORMATIONAL RESPONSE HEADERS

- `Server` — identifies the server generating the message.
- `Warning` — stores text for human consumption, something that would be useful when tracing a problem.
- `Location` — contains the new URL when redirecting.
- `Age` — provided by proxies, time in seconds since the message was generated on the server.
- `Allow` — valid actions for a specified resource.

CONTENT NEGOTIATION
REQUEST HEADERS — ACCEPT

- Content negotiation — a mechanism that allows to serve different versions of a document at the same URI, so that user agents can specify which version fit their **capabilities** the best.

```
Accept: text/html, text/plain;q=0.3
Accept-Charset: utf-8, iso-8859-13;q=0.8
Accept-Encoding: gzip;q=1.0, identity;q=0.5, *;q=0
Accept-Language: pl, en-us;q=0.7
```

- `Accept` — accepted Internet media types (MIME). https://www.iana.org/assignments/media-types/media-types.xhtml
- `Accept-Encoding` — used mainly for HTTP compression.

CONTENT NEGOTIATION RESPONSE ENTITY HEADERS

- Message typing is necessary for both servers and browsers to determine proper actions in **processing** messages.
- Browsers use types and sub-types either to select a proper **content-rendering** module or to invoke a third-party tool.

```
Content-Type: text/html; charset=utf-8
Content-Encoding: gzip
Content-Language: pl
Content-Length: 348
Content-Location: /index.html
```

CLIENT IDENTIFICATION
STATELESS NATURE OF HTTP

- HTTP is stateless and sessionless, each request-response transaction is **independent**.

- Most of the web applications are highly **stateful**, rely on tracking and storing **user sessions**.

CLIENT IDENTIFICATION
STATELESS NATURE OF HTTP

- HTTP is stateless and sessionless, each request-response transaction is **independent**.
- Most of the web applications are highly **stateful**, rely on tracking and storing **user sessions**.
- How to determine which requests come from **the same** user?

CLIENT IDENTIFICATION
STATELESS NATURE OF HTTP

- HTTP is stateless and sessionless, each request-response transaction is **independent**.
- Most of the web applications are highly **stateful**, rely on tracking and storing **user sessions**.
- How to determine which requests come from **the same** user?
- The server can identify and track users by employing:
  - **HTTP headers** — informational request headers: `From`, `Referer`, `User-Agent`
  - **Client IP address tracking** — identify users by their IP addresses: `Client-IP`
  - Extending **URLs** — generating user-specific URLs by embedding identity
  - **Cookies** — the **most popular** and non-intrusive approach (RFC 6265)
  - **ETag** – unique identifier of resource version
  - **User login** — authentication headers: `WWW-Authenticate`, `Authorization`

CLIENT IDENTIFICATION
IP ADDRESS TRACKING

- The client IP address typically is not present in the HTTP headers, but web servers can find the IP address of the other side of the TCP connection.

CLIENT IDENTIFICATION
IP ADDRESS TRACKING

- The client IP address typically is not present in the HTTP headers, but web servers can find the IP address of the other side of the TCP connection.

- Using the client IP address to identify the user has weaknesses:
  - Client IP addresses describe only the computer being used, not the user.
  - Many ISPs assign IP addresses to users **dynamically**.
  - Users are hidden behind **Network Address Translation** (NAT) devices.
  - HTTP proxies and gateways typically open **new TCP connections** to the origin server. Some proxies add `Client-ip` or `X-Forwarded-For` extension headers to preserve the original IP address.
  - **Anonymous proxies** make tracking IP adress impractical.

CLIENT IDENTIFICATION EMBEDDING INFORMATION INTO URLS

- Special versions of each URL for each user (also called **fat URL**s).
- Typically, a real URL is extended by adding some state information (e.g. unique session ID) to the end of the URL or to a **query string**, e.g. `http://[host]/edit.jsp;jsessionid=123`

CLIENT IDENTIFICATION EMBEDDING INFORMATION INTO URLS

- Special versions of each URL for each user (also called **fat URL**s).
- Typically, a real URL is extended by adding some state information (e.g. unique session ID) to the end of the URL or to a **query string**, e.g. `http://[host]/edit.jsp;jsessionid=123`
- Problems:
  - Ugly URLs — URLs displayed in the browser are confusing for new users.
  - Can't share URLs — URLs contain state information about a particular session.
  - Extra server load — the server needs to rewrite HTML to *fatten* the hyperlinks.
  - Not persistent across sessions — all information is lost when the user logs out, unless he **bookmarks** the particular URL.

CLIENT IDENTIFICATION
COOKIES

- **HTTP State Management Mechanism** (RFC 6265)

SERVER

CLIENT

CLIENT IDENTIFICATION
COOKIES

- **HTTP State Management Mechanism** (RFC 6265)

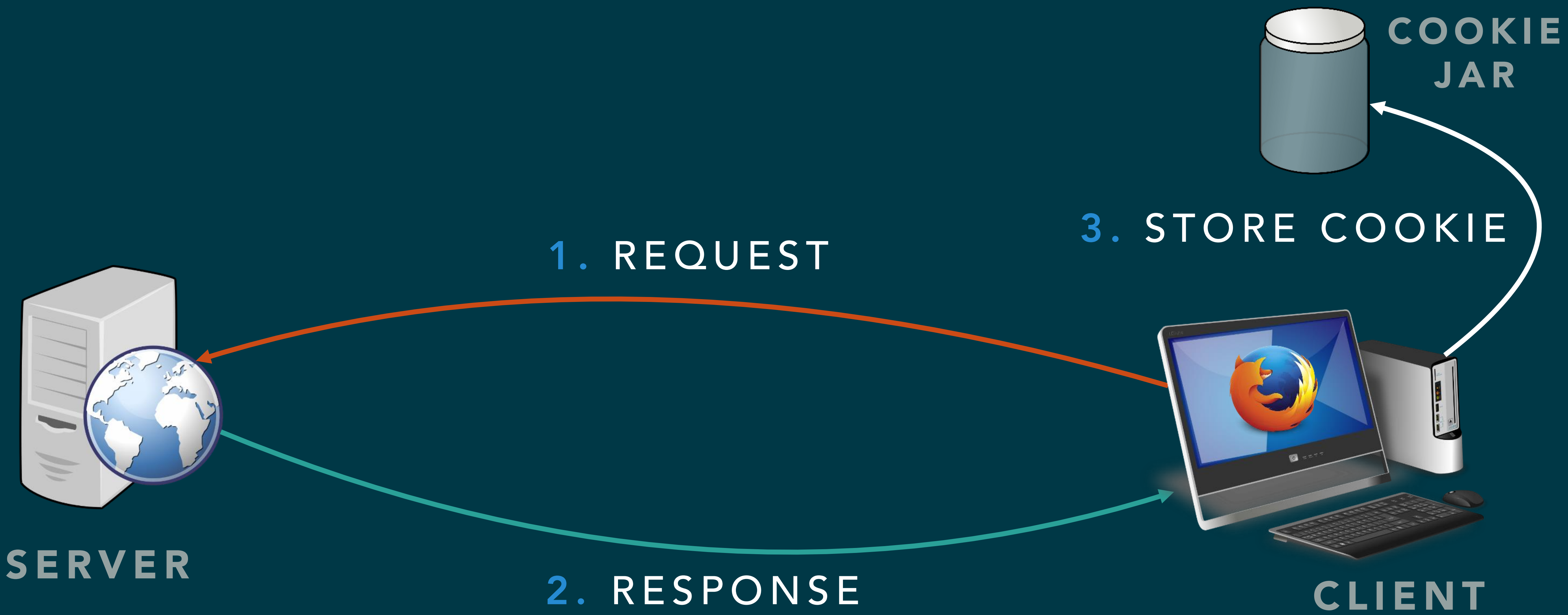
CLIENT IDENTIFICATION
COOKIES

- **HTTP State Management Mechanism** (RFC 6265)


```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: ID=494647c; domain=.msn.com; path=/
```

CLIENT IDENTIFICATION
COOKIES

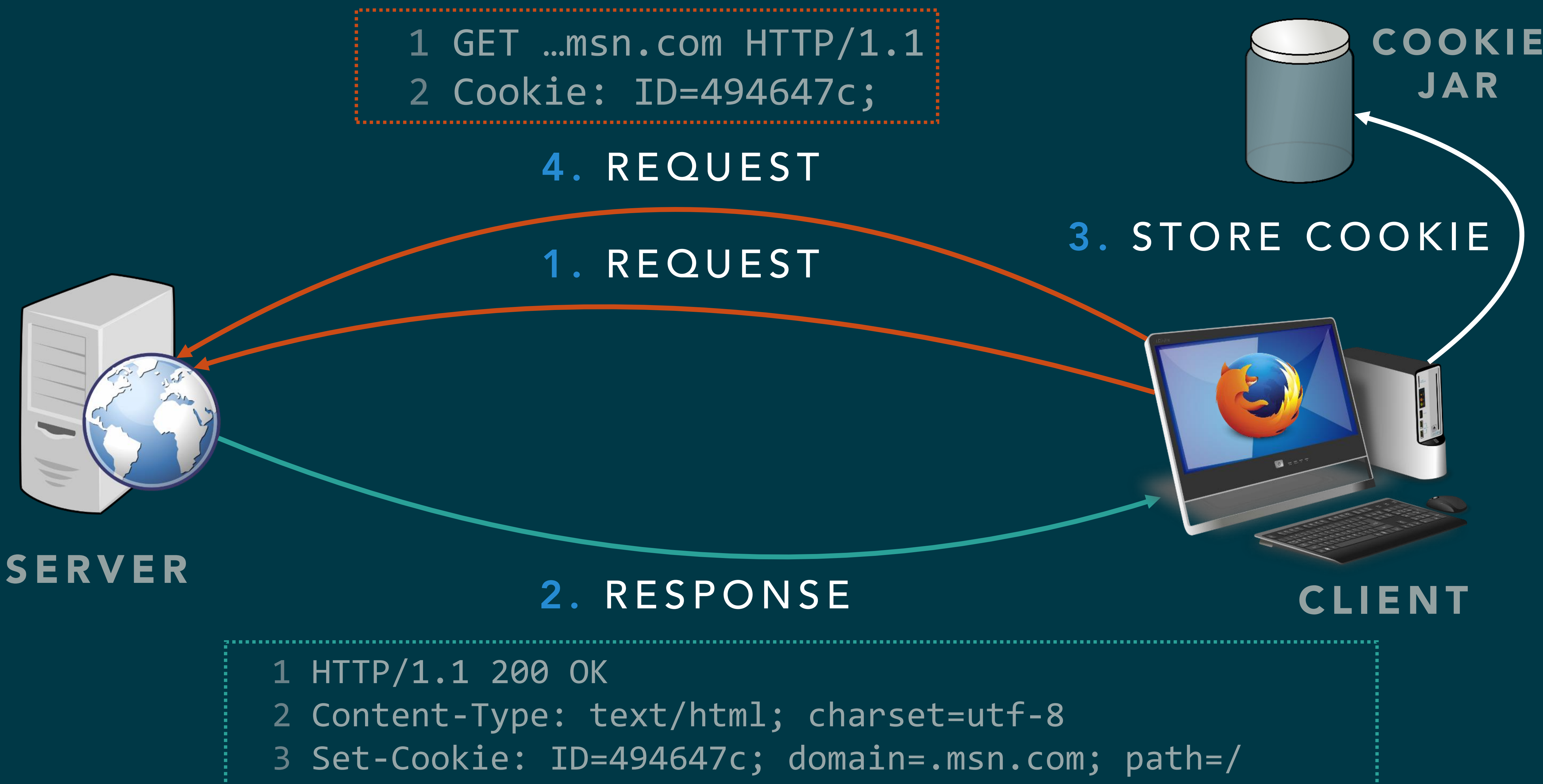
- **HTTP State Management Mechanism** (RFC 6265)




```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: ID=494647c; domain=.msn.com; path=/
```

CLIENT IDENTIFICATION
COOKIES

- **HTTP State Management Mechanism** (RFC 6265)



CLIENT IDENTIFICATION
TYPES OF COOKIES

- **Session cookies** — also known as **in-memory** cookies or **transient** cookies. Web browsers normally delete session cookies when the user closes the browser.
- **Persistent cookies** — also referred to as **tracking** cookies. Instead of expiring when the web browser is closed, persistent cookies **expire** at a specific date or after a specific length of time.

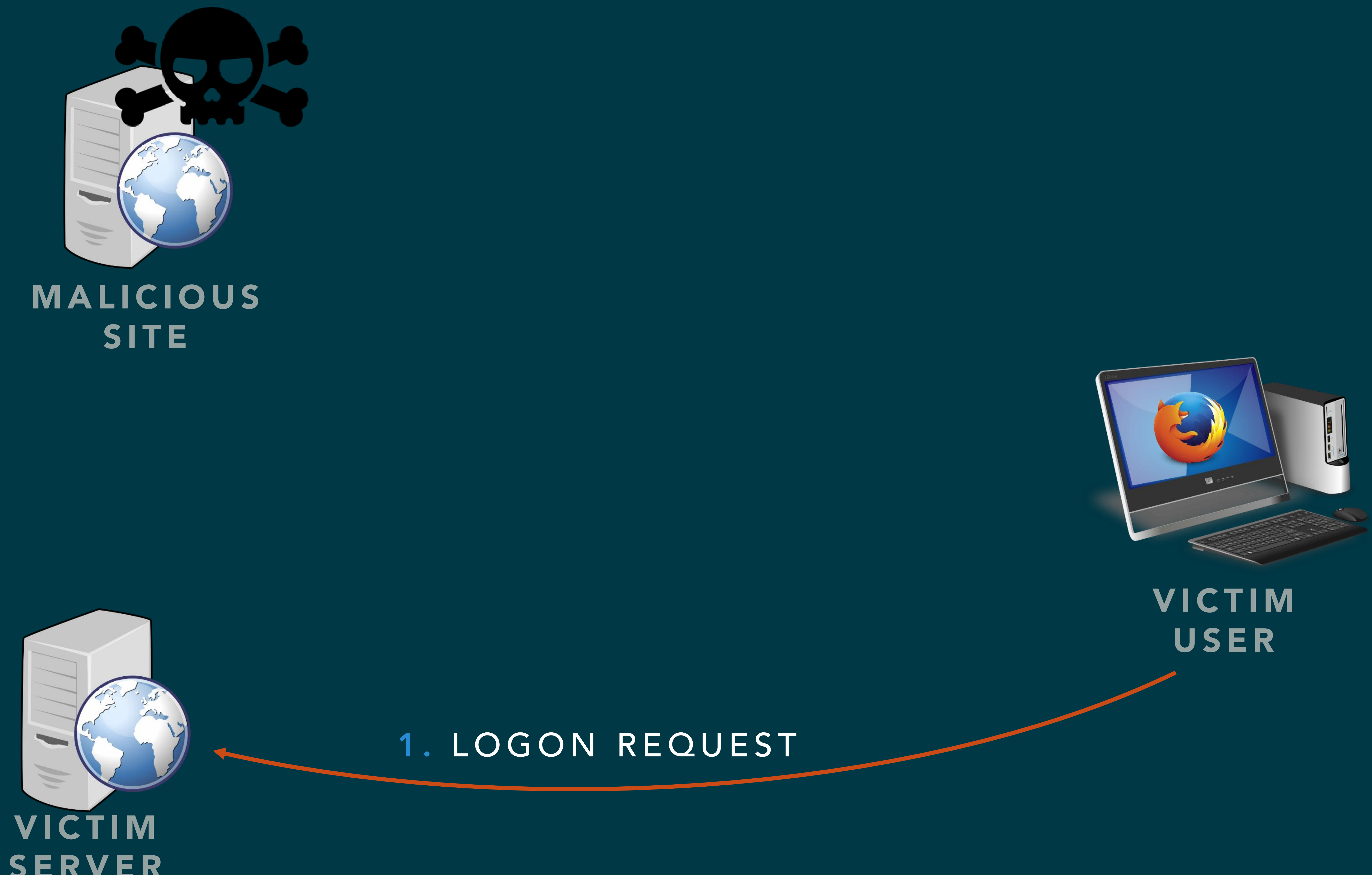
```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: ID=494647c; Max-Age=86400
Set-Cookie: ID=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT
```

CLIENT IDENTIFICATION
PROBLEMS WITH COOKIES

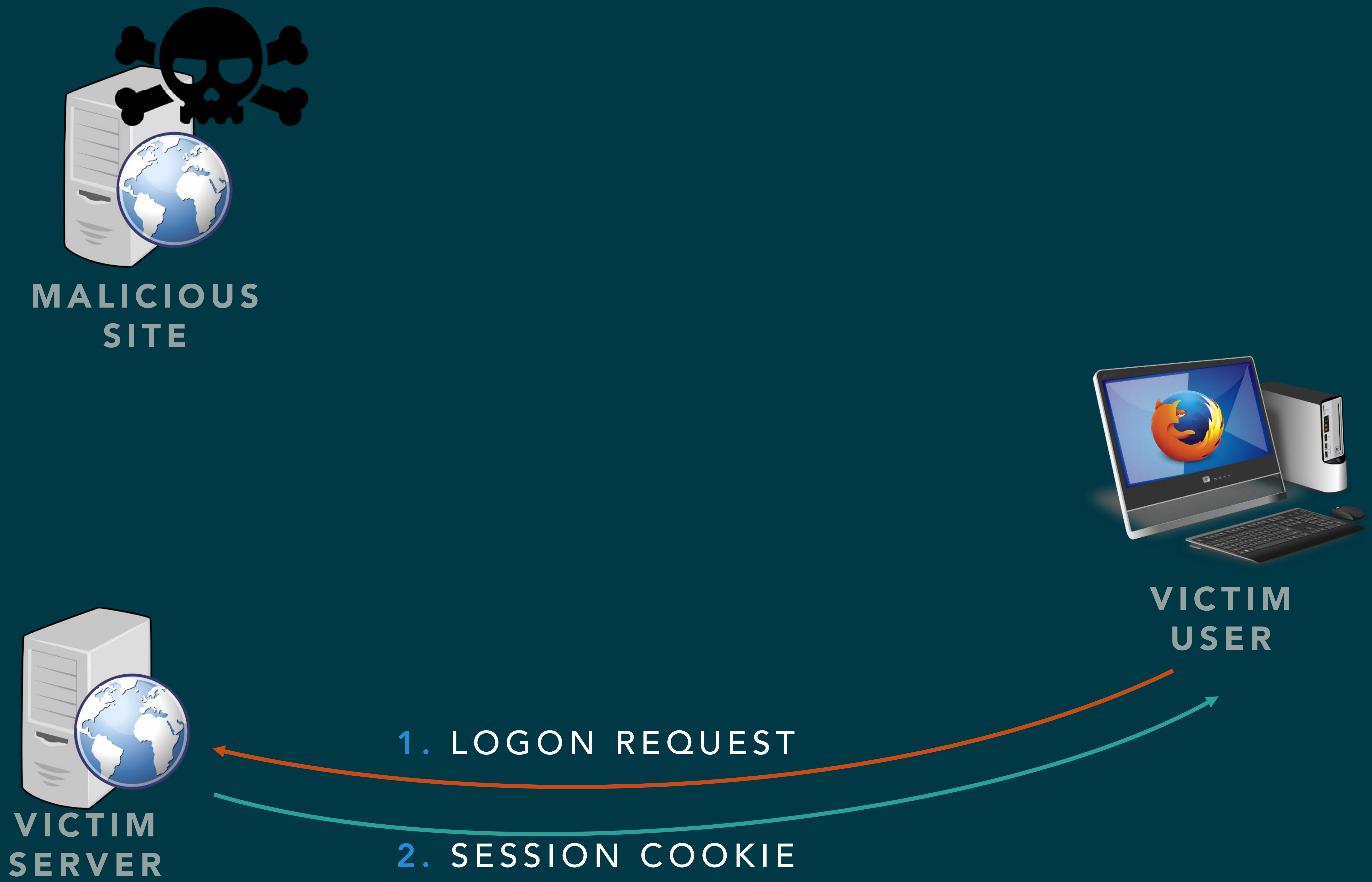
- Session hijacking and cookie theft:
  - **network sniffing** — resolved by using `Secure` cookies
  - **cross-site scripting** — mitigated by using `HttpOnly` cookies
  - **cross-site request forgery**
- Can be disabled or deleted by users in their browsers.
- Privacy concerns:
  - **Third party cookies** may track users across the Internet
    - E.g., Google Analytics
  - **EU: *The Right to be Forgotten***
    Service providers are required to ask users whether they accept use of a tracking mechanism (in Poland from 2013)
    - Penalties up to €1 million or 2% of their sale

CROSS-SITE REQUEST FORGERY

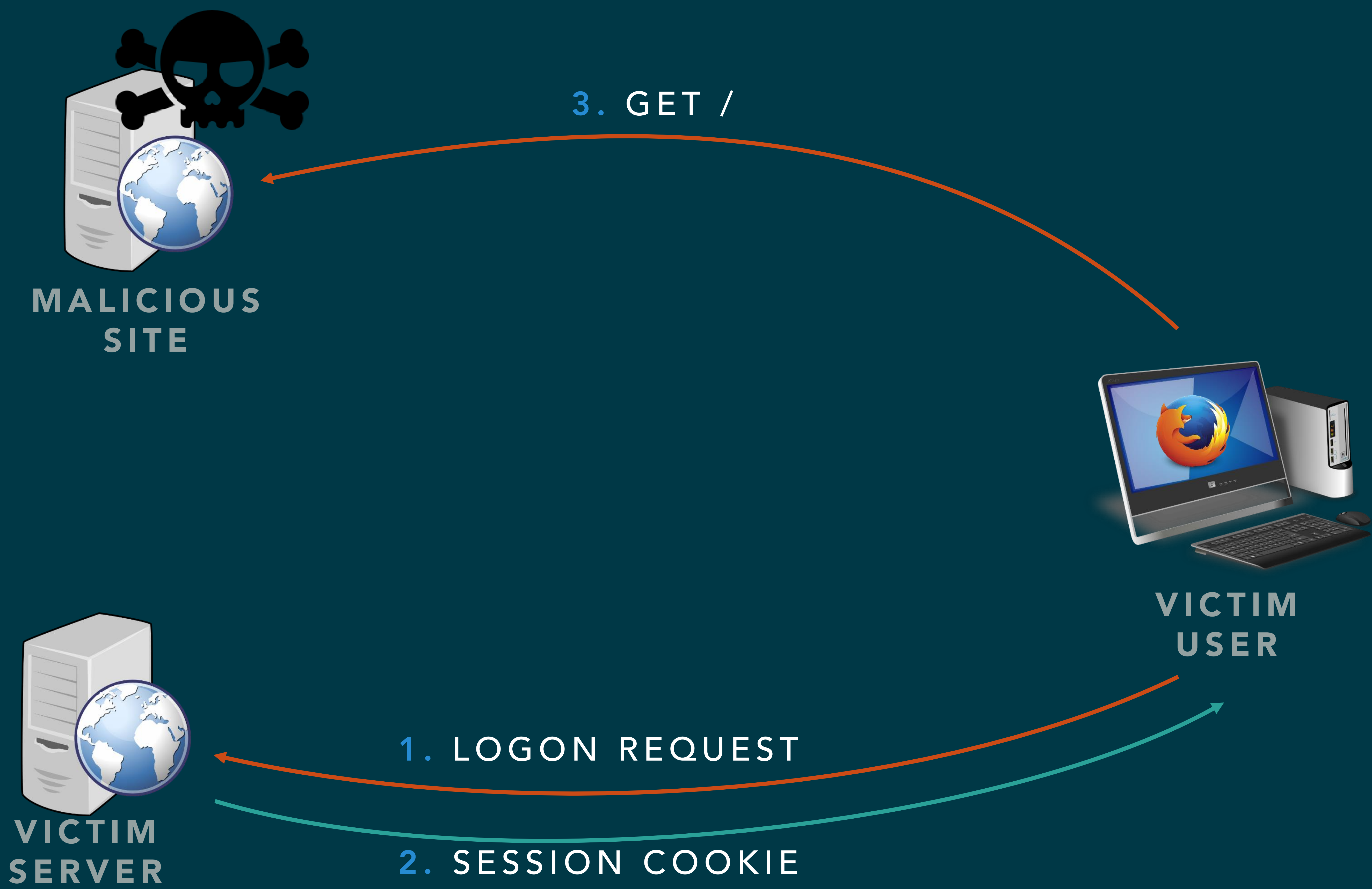
CROSS-SITE REQUEST FORGERY



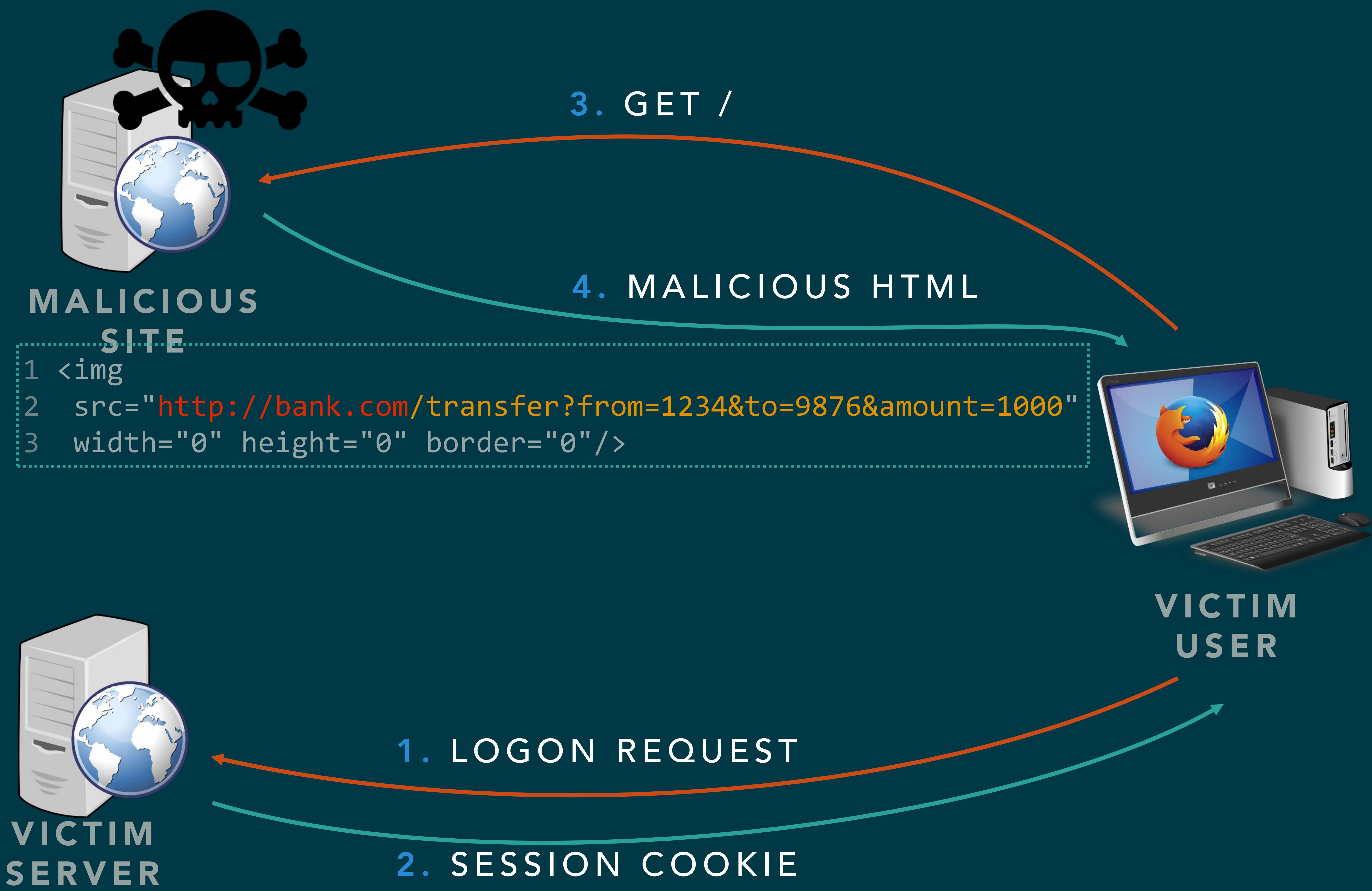
CROSS-SITE REQUEST FORGERY



CROSS-SITE REQUEST FORGERY

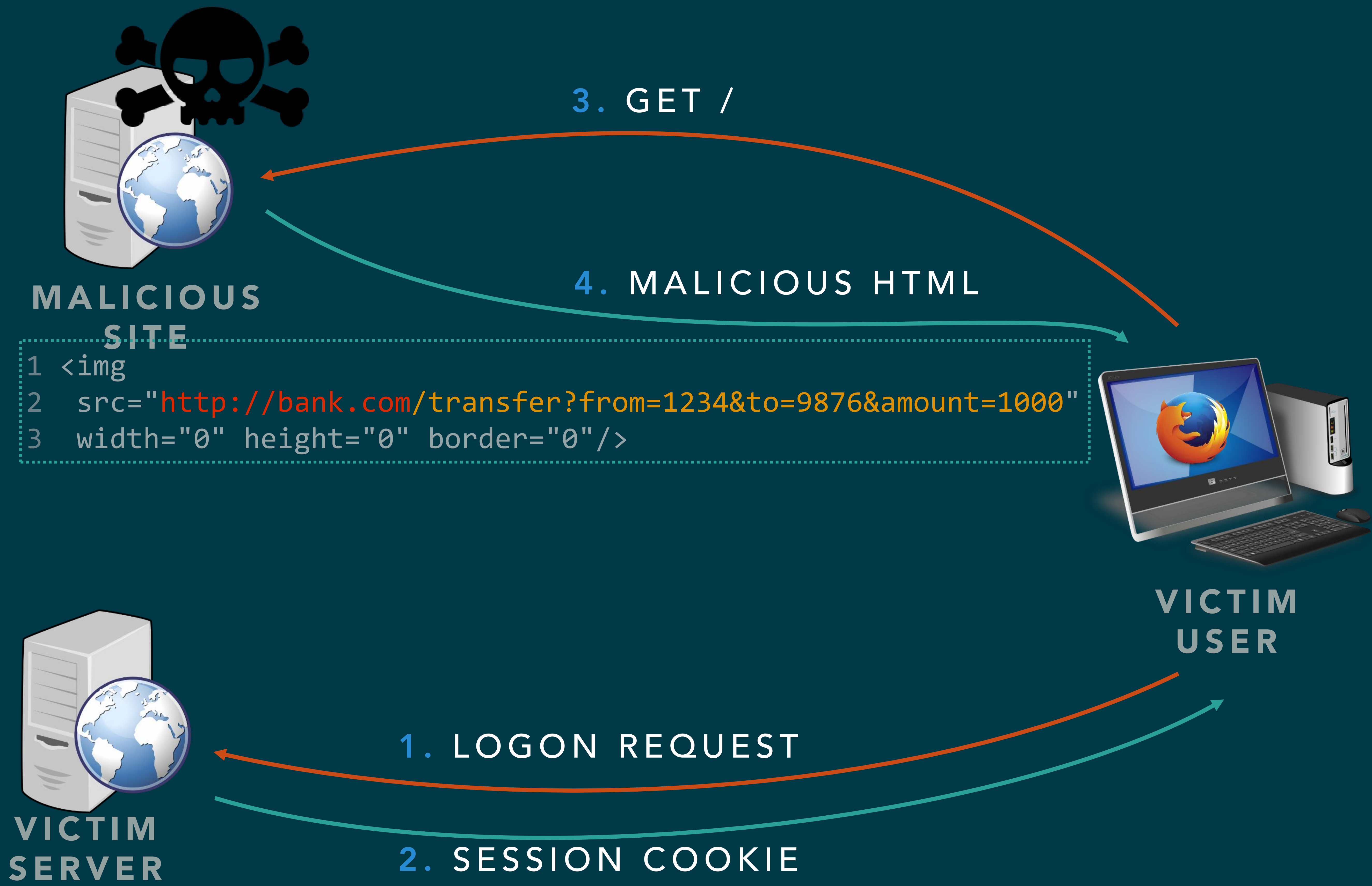


CROSS-SITE REQUEST FORGERY



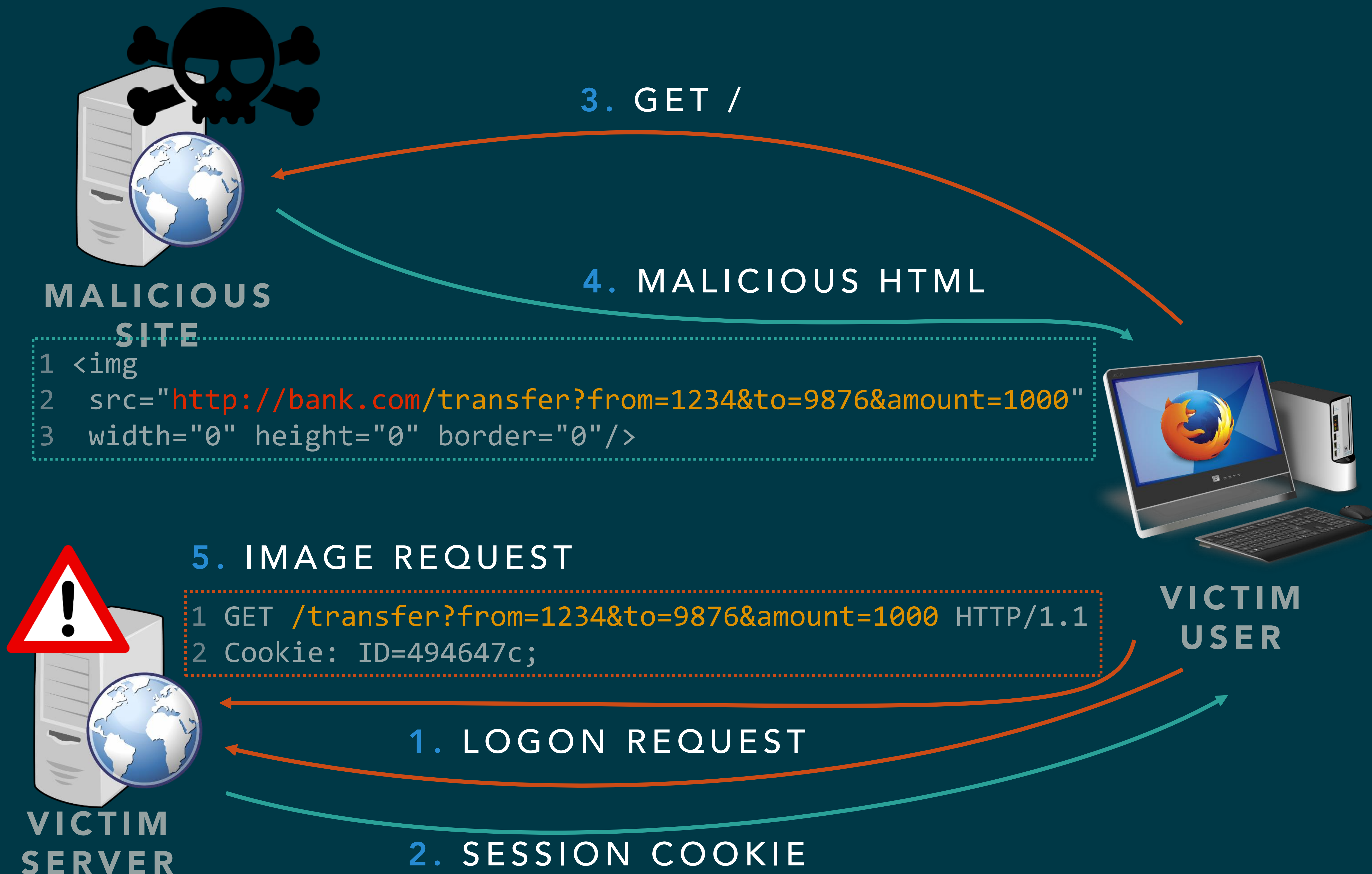

```
<img
 src="http://bank.com/transfer?from=1234&to=9876&amount=1000"
 width="0" height="0" border="0"/>
```

CROSS-SITE REQUEST FORGERY




```
<img
 src="http://bank.com/transfer?from=1234&to=9876&amount=1000"
 width="0" height="0" border="0"/>
```

CROSS-SITE REQUEST FORGERY



CLIENT IDENTIFICATION
ETAG

- ETag
  - Piece of information that uniquely identifies a resource and its version
    - E.g., a cryptographic sum: crc, md5, sha-1, sha-256,…
  - Sent by server in HTTP headers
  - Intended for effective caching
- Browser that supports ETags
  - Sends header in every subsequent request:
  - `If-None-Match: “etag-value”`
- Server responses
  - 304 Not Modified or
  - 200 Ok

CLIENT IDENTIFICATION
ETAG

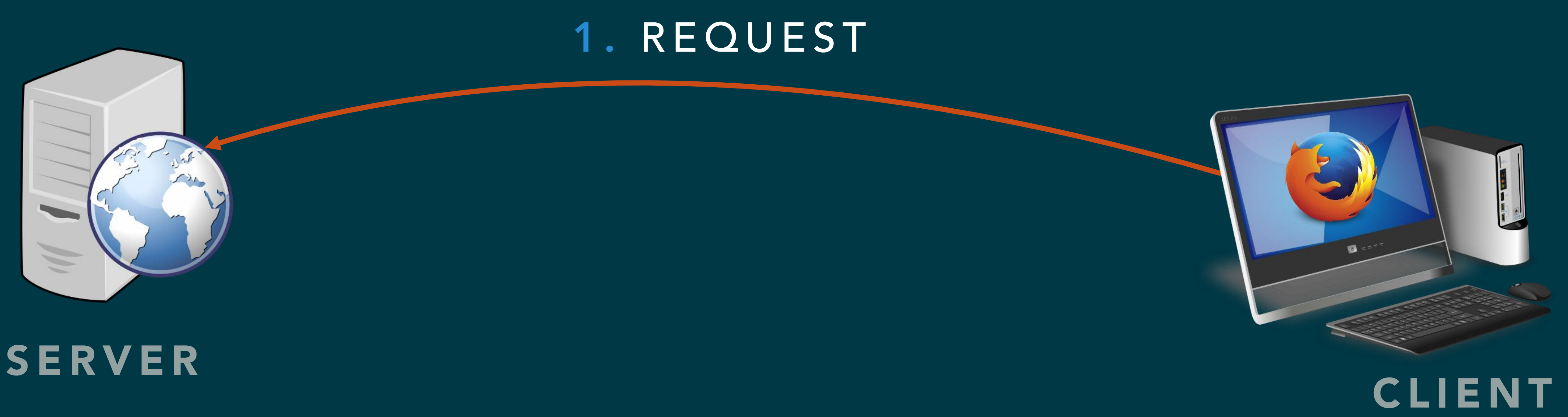
- To track a user, send different ETag for the same resource each time, a request has no ETag included

CLIENT IDENTIFICATION ETAG

- To track a user, send different ETag for the same resource each time, a request has no ETag included

CLIENT IDENTIFICATION ETAG

- To track a user, send different ETag for the same resource each time, a request has no ETag included



CLIENT IDENTIFICATION ETAG

- To track a user, send different ETag for the same resource each time, a request has no ETag included

CLIENT IDENTIFICATION
ETAG

- To track a user, send different ETag for the same resource each time, a request has no ETag included


```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
ETag: "686897696a7c876b7e"
```

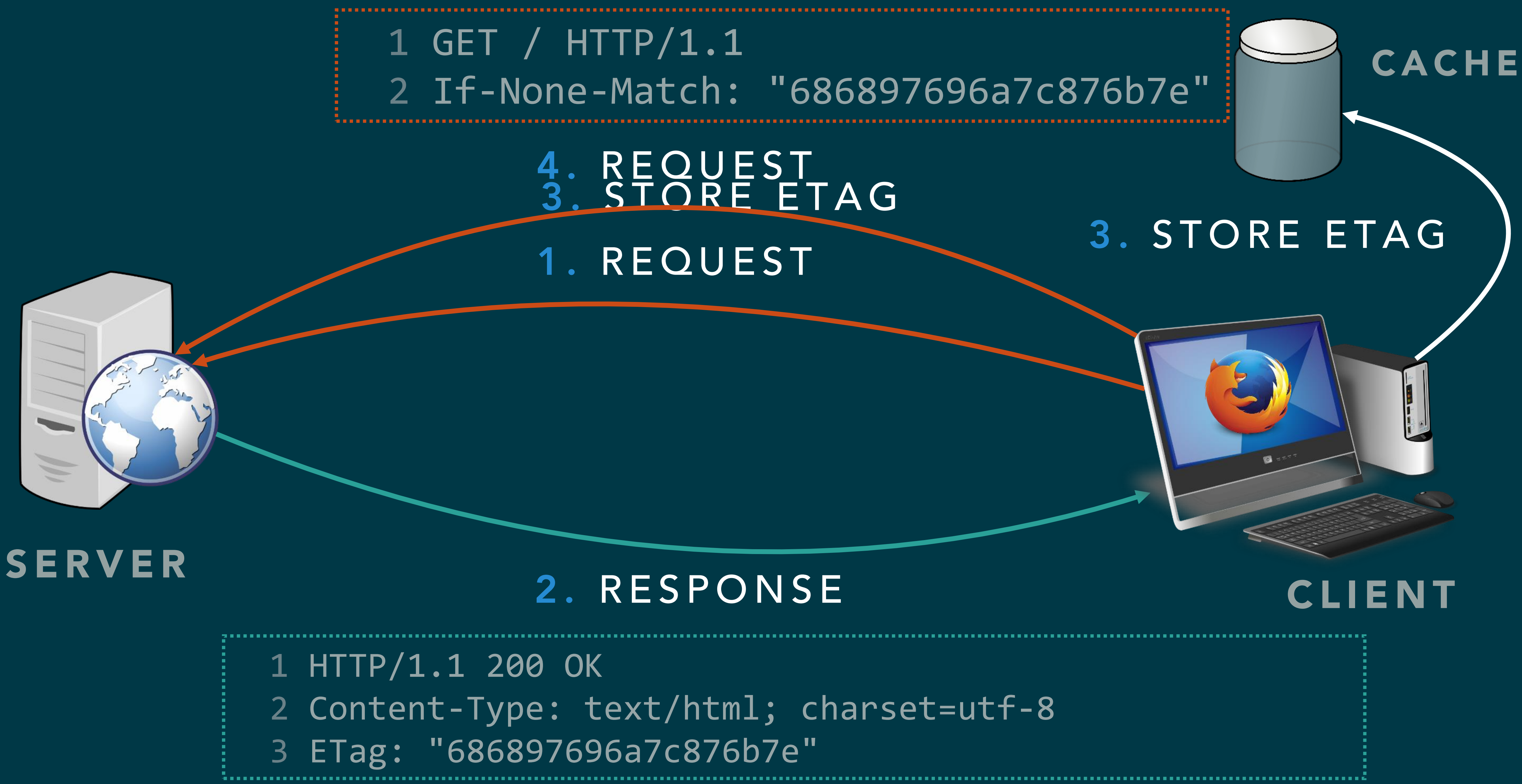
CLIENT IDENTIFICATION
ETAG

- To track a user, send different ETag for the same resource each time, a request has no ETag included


```
1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 ETag: "686897696a7c876b7e"
```

CLIENT IDENTIFICATION
ETAG

- To track a user, send different ETag for the same resource each time, a request has no ETag included



CLIENT IDENTIFICATION
HTTP BASIC AUTHENTICATION

- HTTP provides built-in support for Basic Authentication, where user credentials are transmitted via the Authorization header as a single **Base64**-encoded string.

CLIENT IDENTIFICATION
HTTP BASIC AUTHENTICATION

- HTTP provides built-in support for Basic Authentication, where user credentials are transmitted via the Authorization header as a single **Base64**-encoded string.

```
HTTP/1.1 401 Unauthorized
Server: Apache/2.2.4
WWW-Authenticate: Basic
```

CLIENT IDENTIFICATION
HTTP BASIC AUTHENTICATION

- HTTP provides built-in support for Basic Authentication, where user credentials are transmitted via the Authorization header as a single **Base64**-encoded string.

```
HTTP/1.1 401 Unauthorized
Server: Apache/2.2.4
WWW-Authenticate: Basic
```

```
GET http://localhost/protected/ HTTP/1.1
Authorization: Basic bA808gER3F9dfafag
```

CLIENT IDENTIFICATION
HTTP BASIC AUTHENTICATION

- HTTP provides built-in support for Basic Authentication, where user credentials are transmitted via the Authorization header as a single **Base64**-encoded string.

```
HTTP/1.1 401 Unauthorized
Server: Apache/2.2.4
WWW-Authenticate: Basic
```

```
GET http://localhost/protected/ HTTP/1.1
Authorization: Basic bA808gER3F9dfafag
```

- If the server validates the authorization credentials, browser uses them as the value of the Authorization header in future requests to **dependent URLs**.

CLIENT IDENTIFICATION
HTTP BASIC AUTHENTICATION

- HTTP provides built-in support for Basic Authentication, where user credentials are transmitted via the Authorization header as a single **Base64**-encoded string.

```
HTTP/1.1 401 Unauthorized
Server: Apache/2.2.4
WWW-Authenticate: Basic
```

```
GET http://localhost/protected/ HTTP/1.1
Authorization: Basic bA808gER3F9dfafag
```

- If the server validates the authorization credentials, browser uses them as the value of the Authorization header in future requests to **dependent URLs**.

- Basic authentication is **insecure by default** — credentials are simply encoded (not encrypted) — rarely used without **HTTPS**.

CLIENT IDENTIFICATION
HTTP DIGEST AUTHENTICATION

- Server sends a seed to the client
- Client sends MD5 of credentials concatenated with seed
- Algorithm for calculating response (RFC 2069):

CLIENT IDENTIFICATION
HTTP DIGEST AUTHENTICATION

- Server sends a seed to the client
- Client sends MD5 of credentials concatenated with seed
- Algorithm for calculating response (RFC 2069):

```
HA1=MD5(username:realm:password)
HA2=MD5(method:digestURI)
response=MD5(HA1:nonce:HA2)
```

CLIENT IDENTIFICATION
HTTP DIGEST AUTHENTICATION

CLIENT IDENTIFICATION
HTTP DIGEST AUTHENTICATION

```
HTTP/1.1 401 Unauthorized
Date: Sun, 10 Apr 2014 20:26:47 GMT
WWW-Authenticate: Digest realm="testrealm@host.com",
                  nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093"
```

CLIENT IDENTIFICATION
HTTP DIGEST AUTHENTICATION

```
HTTP/1.1 401 Unauthorized
Date: Sun, 10 Apr 2014 20:26:47 GMT
WWW-Authenticate: Digest realm="testrealm@host.com",
                  nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093"
```

```
GET /dir/index.html HTTP/1.1
Host: localhost
Authorization: Digest username="Mufasa",
                   realm="testrealm@host.com",
                   nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                   uri="/dir/index.html",
                   response="6629fae49393a05397450978507c4ef1"
```

CLIENT IDENTIFICATION
HTTP DIGEST AUTHENTICATION

- RFC 2617 defines more secure way to digest authentication
  - Recurrent MD5 hashes
  - Counter of requests incremented by client
  - Client-generated seed
- Advantages:
  - More secure than basic and RFC 2069 digest authentication
    - Password is not sent explicitly
    - But: MD5 collisions are easy to generate

HTTP CONNECTIONS

- How do HTTP messages move through the network?

SERVER

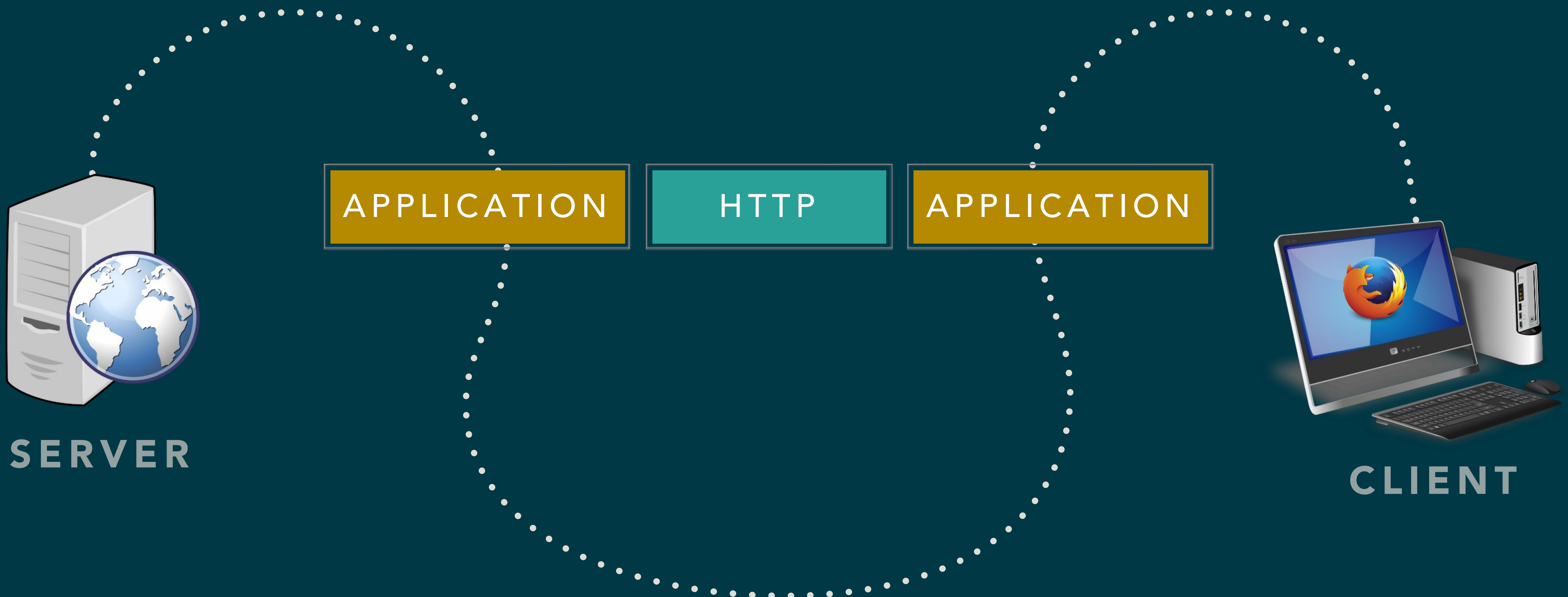
CLIENT

HTTP CONNECTIONS

- How do HTTP messages move through the network?
- A **TCP connection** must be established between the client and server before they can communicate with each other.

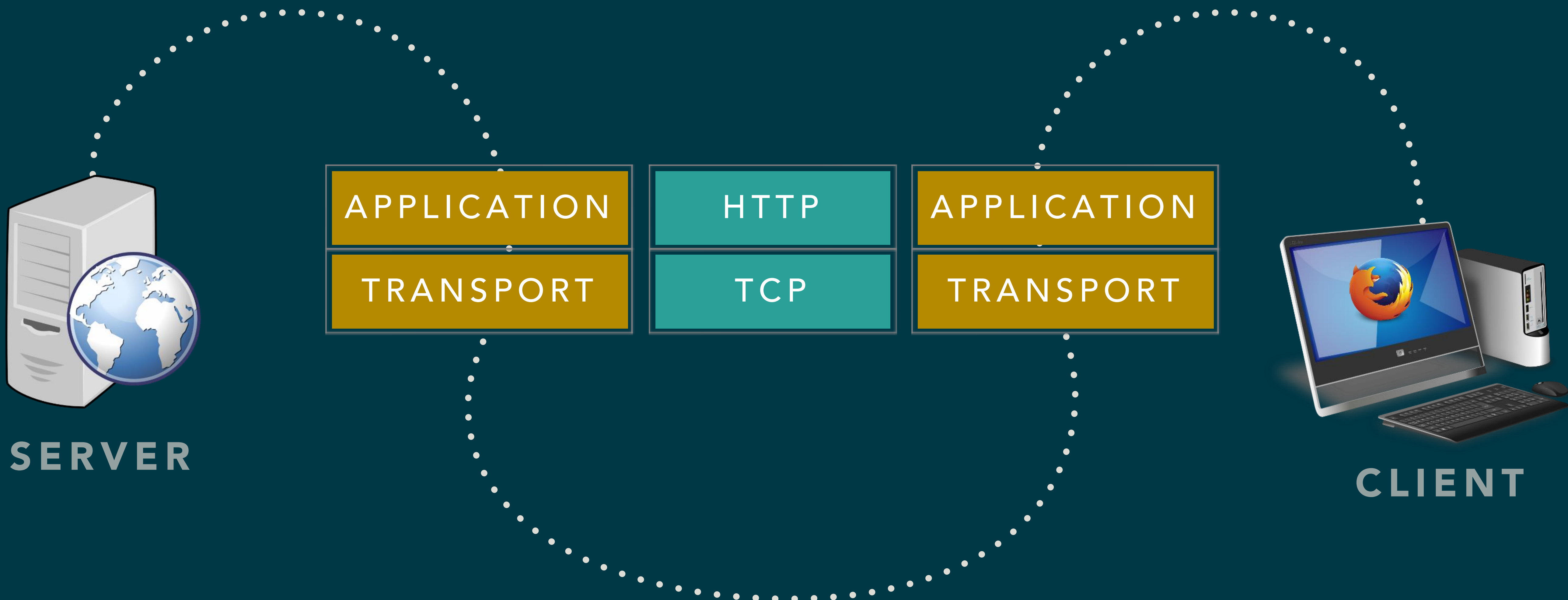
HTTP CONNECTIONS

- How do HTTP messages move through the network?
- A **TCP connection** must be established between the client and server before they can communicate with each other.



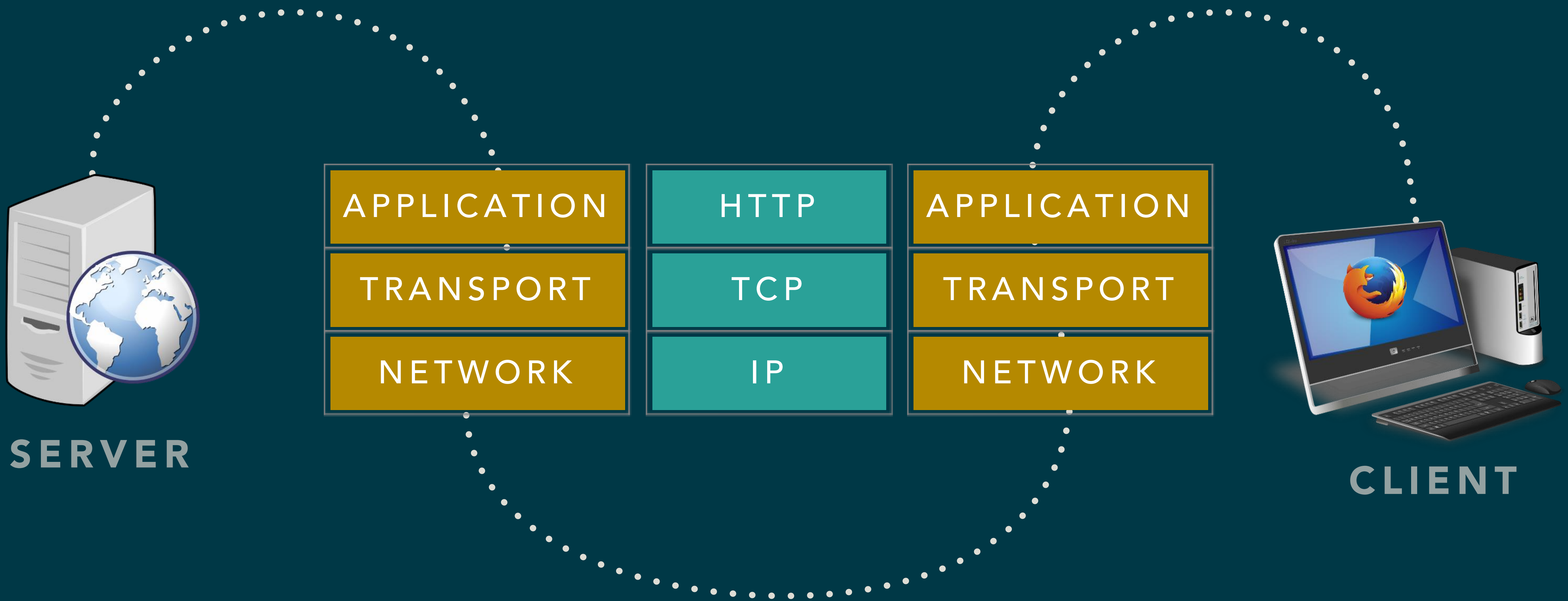
HTTP CONNECTIONS

- How do HTTP messages move through the network?
- A **TCP connection** must be established between the client and server before they can communicate with each other.



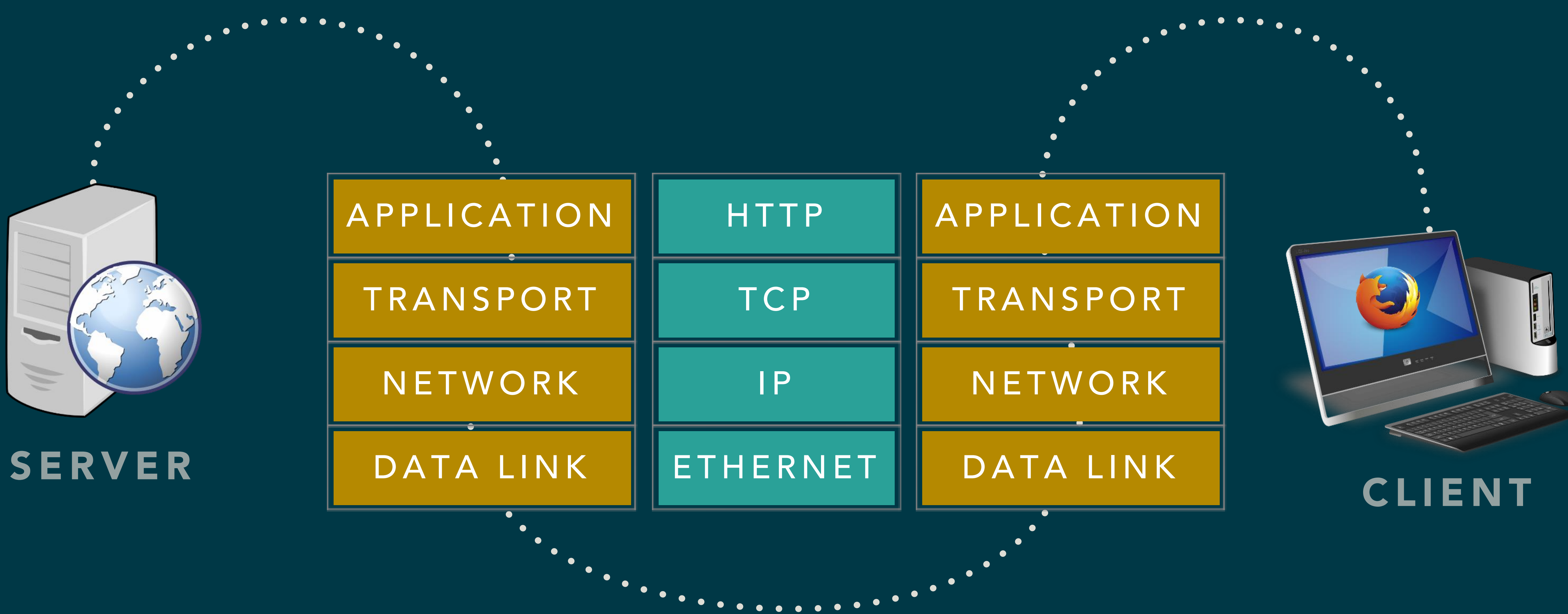
HTTP CONNECTIONS

- How do HTTP messages move through the network?
- A **TCP connection** must be established between the client and server before they can communicate with each other.



HTTP CONNECTIONS

- How do HTTP messages move through the network?
- A **TCP connection** must be established between the client and server before they can communicate with each other.



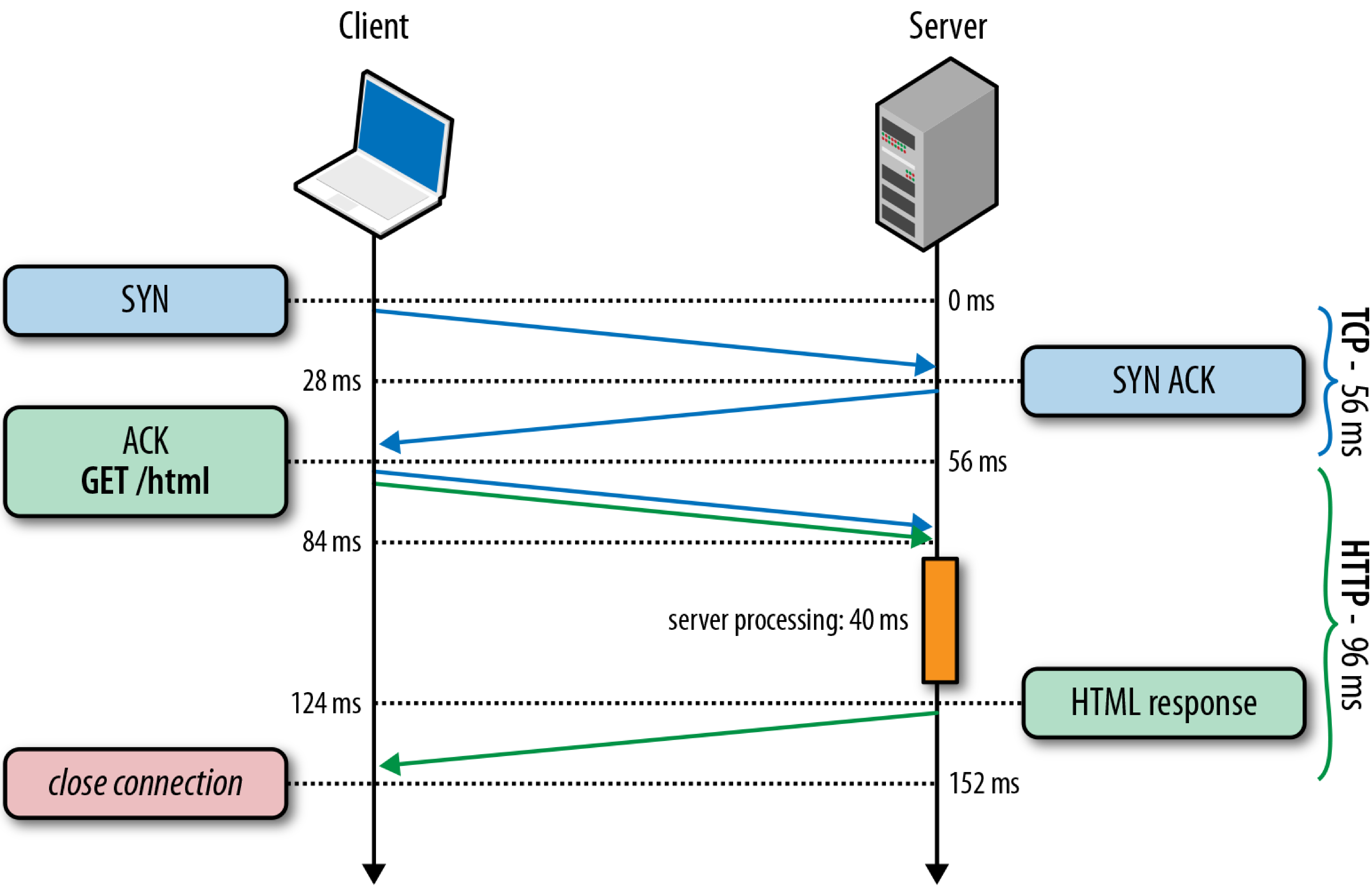
PERSISTENT CONNECTIONS

- When does a browser open and close a **connection**?

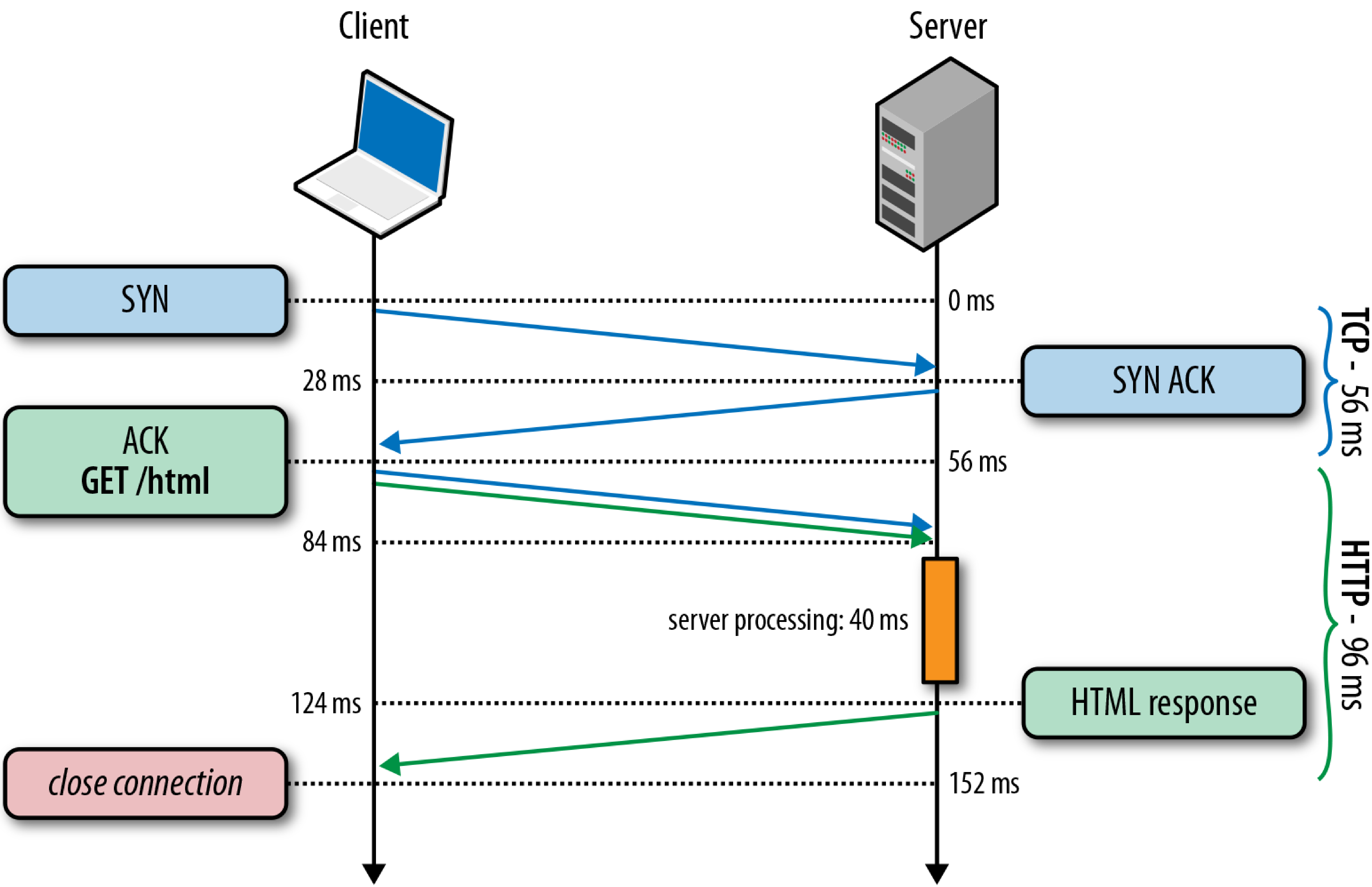
PERSISTENT CONNECTIONS

- When does a browser open and close a **connection**?
- **HTTP/1.0** — all connections were closed after a **single** transaction.
  - HTTP is stateless — it does not require extended connection lifetime.
  - Lot of network delays due to **three-way handshake** and **slow-start**.

TCP connection #1, Request #1: HTML request



TCP connection #1, Request #1: HTML request



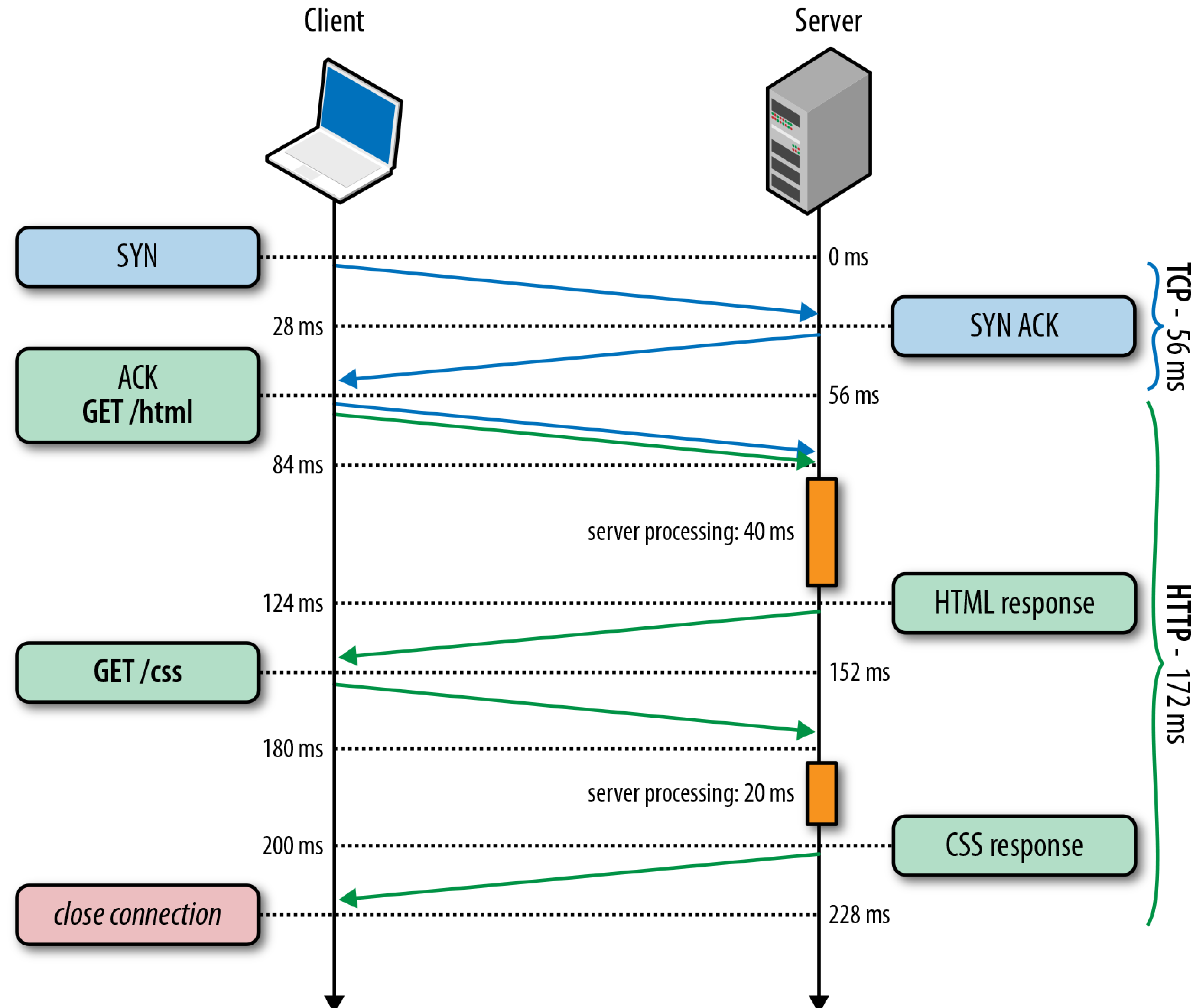
PERSISTENT CONNECTIONS

- When does a browser open and close a **connection**?
- **HTTP/1.0** — all connections were closed after a **single** transaction.
  - HTTP is stateless — it does not require extended connection lifetime.
  - Lot of network delays due to **three-way handshake** and **slow-start**.

PERSISTENT CONNECTIONS

- When does a browser open and close a **connection**?
- **HTTP/1.0** — all connections were closed after a **single** transaction.
  - HTTP is stateless — it does not require extended connection lifetime.
  - Lot of network delays due to **three-way handshake** and **slow-start**.
- **HTTP/1.1** — introduced **persistent connections**:
  - Reducing connection-establishment delays,
  - Long-lived connections that stay open until the client closes them.
  - Persistent connections are **default**, `Connection: keep-alive` is redundant.
  - Making a single transaction connection requires the client to set the `Connection: close` request header.
  - Most web servers close a persistent connection if it is **idle** for some period.

TCP connection #1, Request #1-2: HTML + CSS

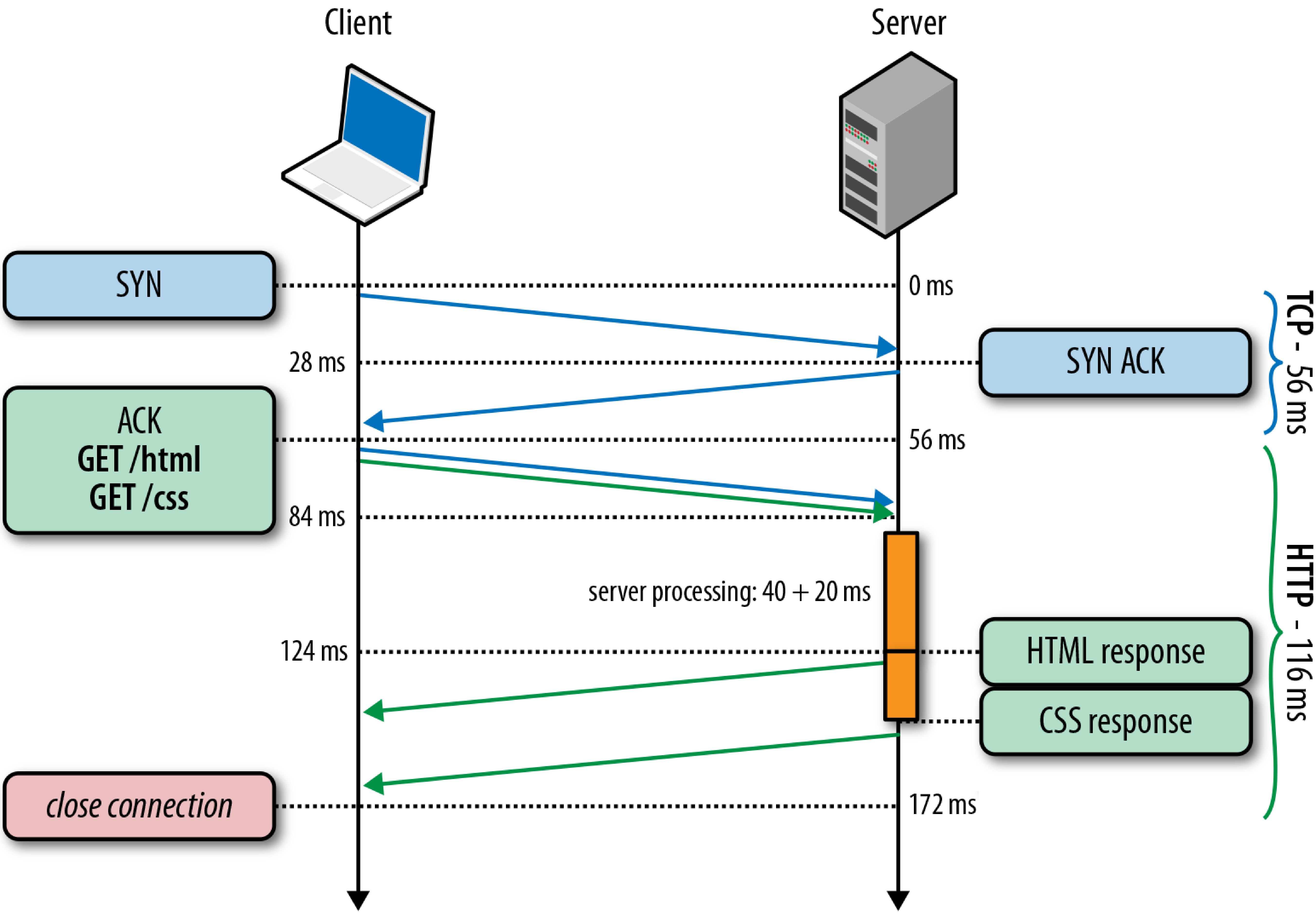


PIPELINING REQUESTS

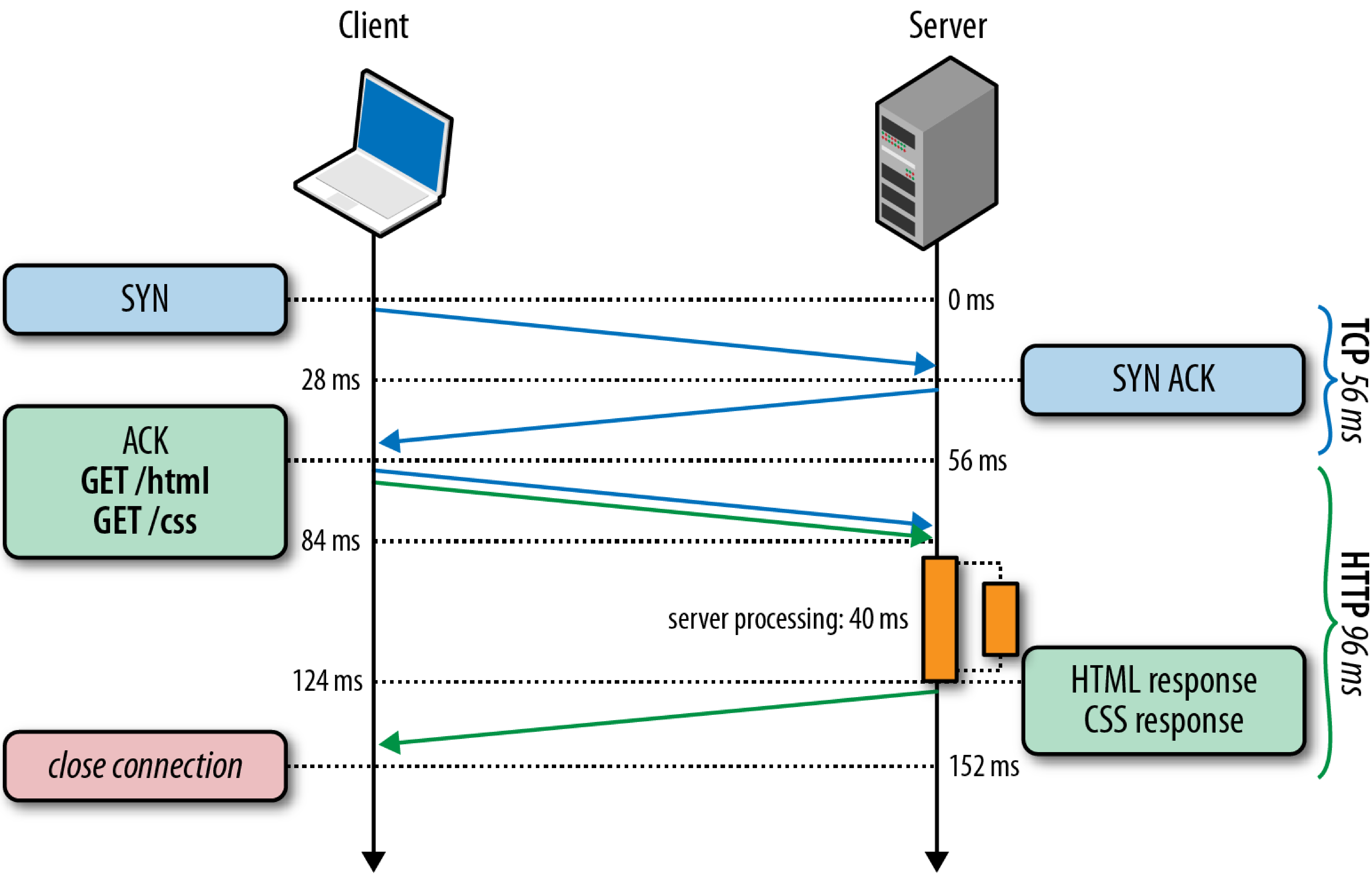
- Persistent HTTP allows us to reuse an existing TCP connection between multiple application requests, but it implies a strict first in, first out (**FIFO**) queuing order on the client.

PIPELINING REQUESTS

- Persistent HTTP allows us to reuse an existing TCP connection between multiple application requests, but it implies a strict first in, first out (**FIFO**) queuing order on the client.
- **HTTP pipelining** is a small but important optimization to this workflow, which allows us to relocate the FIFO queue from the client (**request queuing**) to the server (**response queuing**):
  - Browsers can send requests without waiting for responses.
  - Servers are responsible for submitting responses to browser requests in the order of their arrival.



Client
Server
SYN
0 ms
SYN ACK
28 ms
ACK
GET /html
GET /css
56 ms
84 ms
server processing: 40 + 20 ms
124 ms
HTML response
CSS response
close connection
172 ms
TCP - 56 ms
HTTP - 116 ms



Client
Server
SYN
0 ms
28 ms
SYN ACK
TCP 56 ms
ACK
GET /html
GET /css
56 ms
84 ms
server processing: 40 ms
124 ms
HTML response
CSS response
HTTP 96 ms
close connection
152 ms

PIPELINING REQUESTS

- What if the first request **hangs** indefinitely or simply takes a very long time to generate on the server?

PIPELINING REQUESTS

- What if the first request **hangs** indefinitely or simply takes a very long time to generate on the server?
- **Head-of-line blocking** results in suboptimal delivery:
  - underutilized network links,
  - server buffering costs,
  - unpredictable latency delays for the client.

PIPELINING REQUESTS

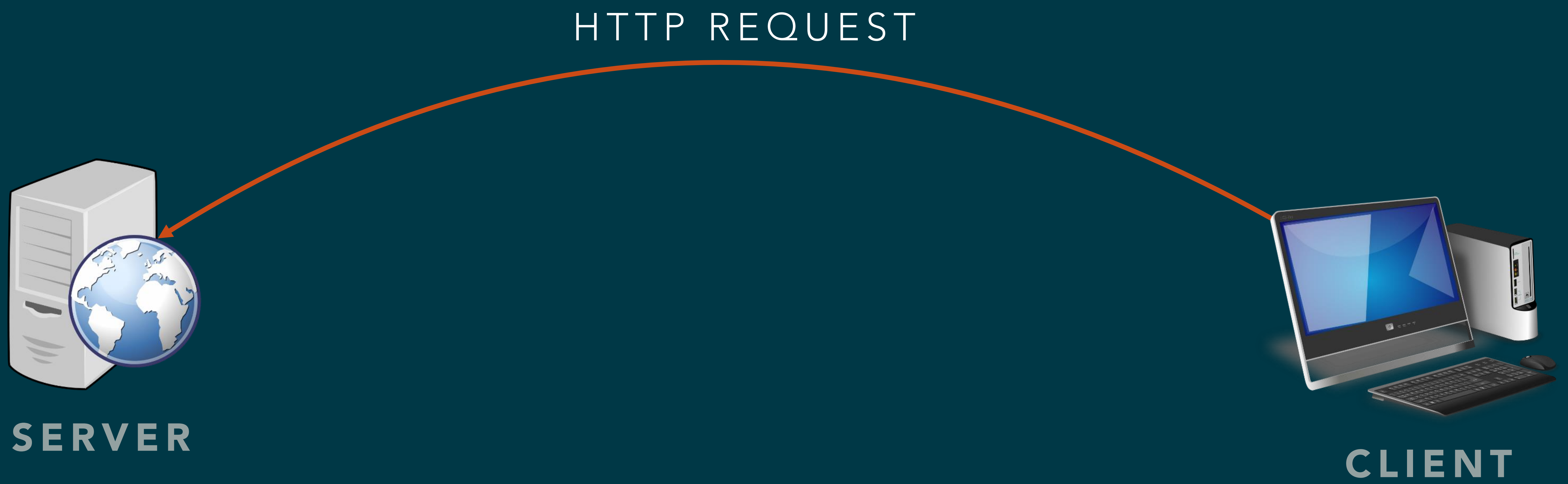
- What if the first request **hangs** indefinitely or simply takes a very long time to generate on the server?
- **Head-of-line blocking** results in suboptimal delivery:
  - underutilized network links,
  - server buffering costs,
  - unpredictable latency delays for the client.
- **HTTP pipelining adoption has remained very limited despite its many benefits — some browsers support pipelining, usually as an advanced option, but most have it disabled.**

CONNECTIONS AND PROXIES

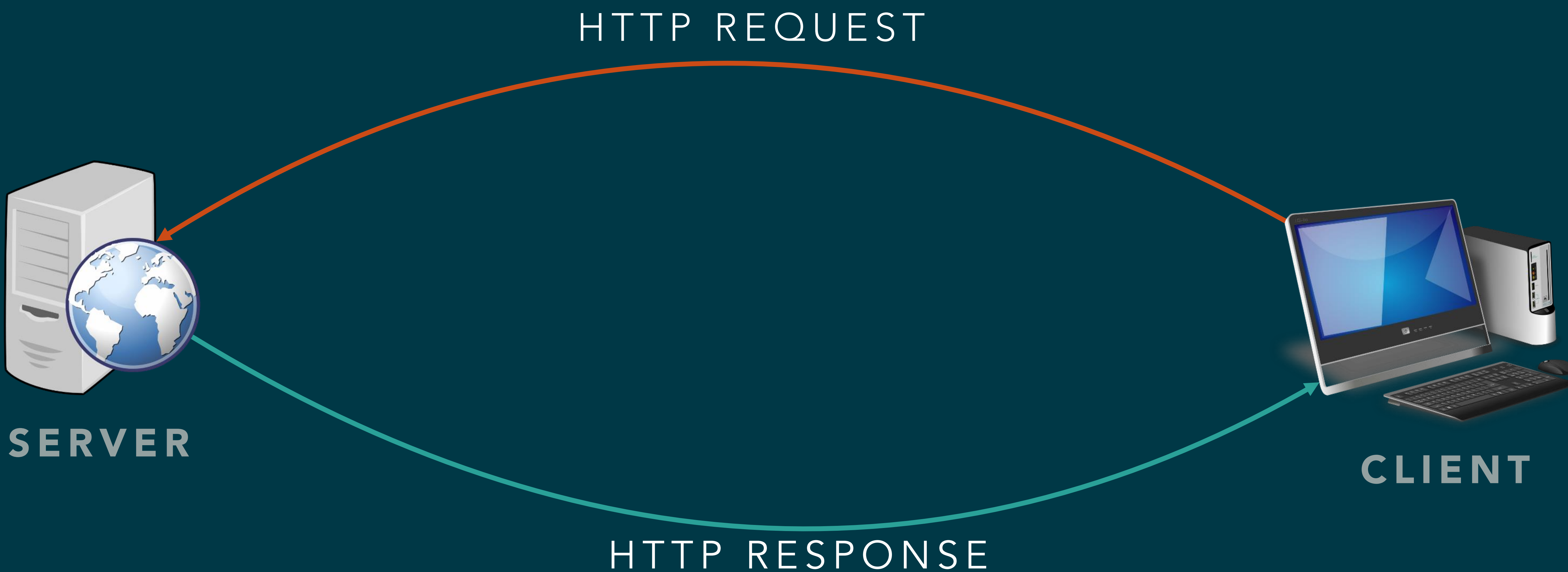
SERVER

CLIENT

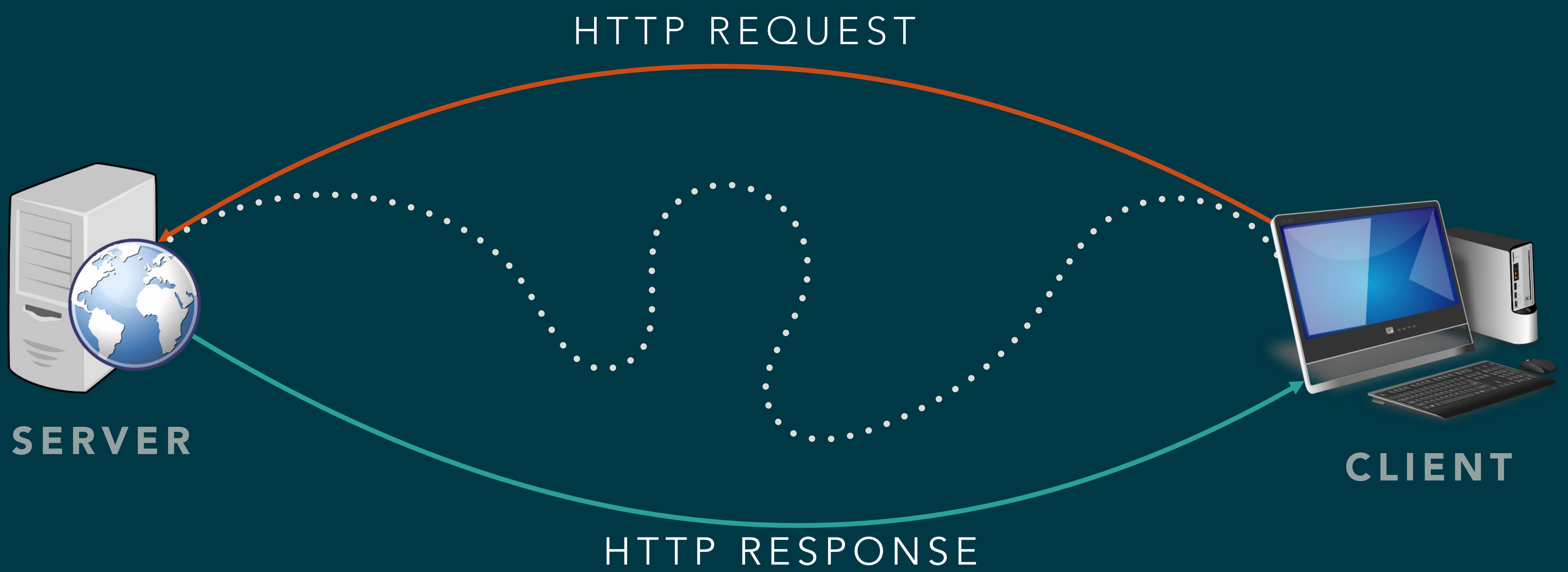
CONNECTIONS AND PROXIES



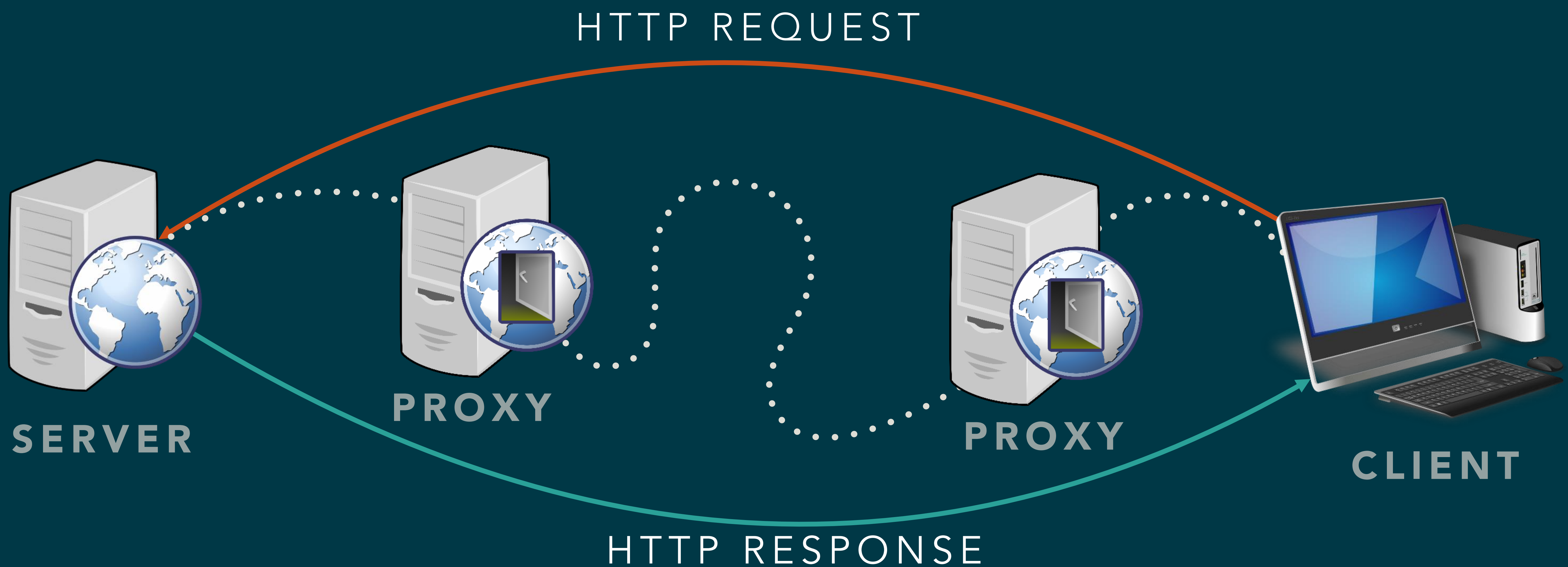
CONNECTIONS AND PROXIES



CONNECTIONS AND PROXIES

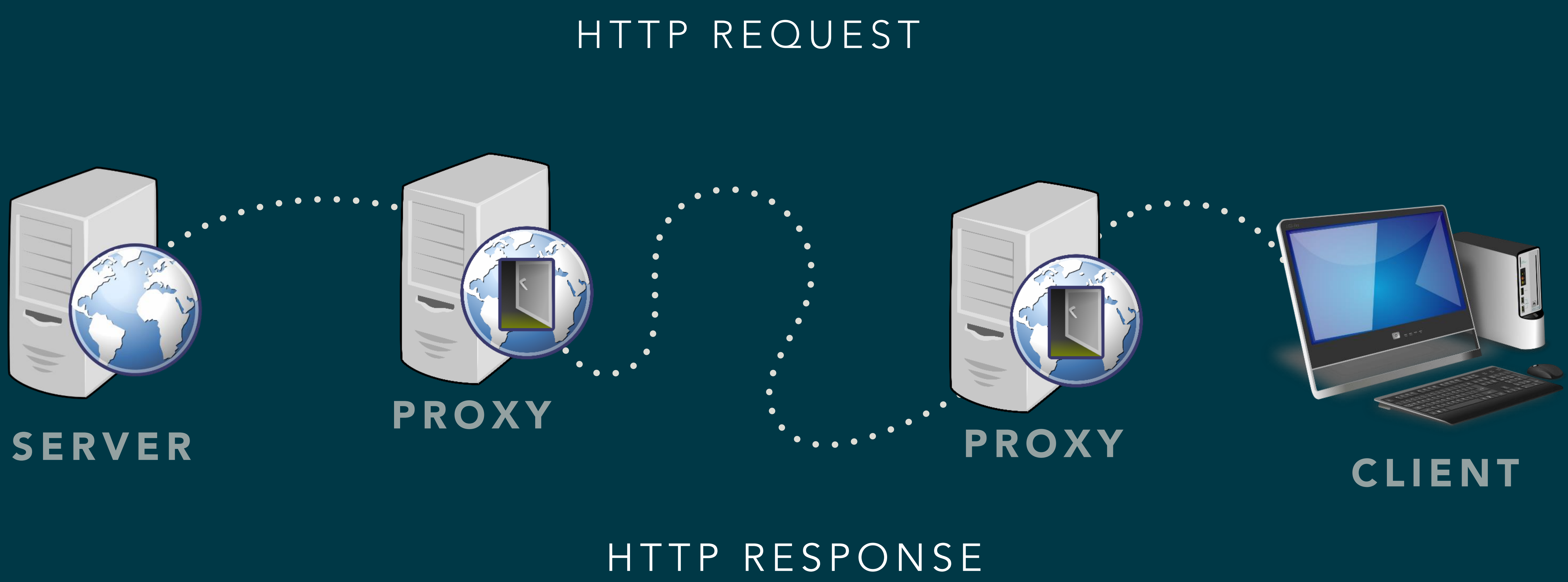


CONNECTIONS AND PROXIES



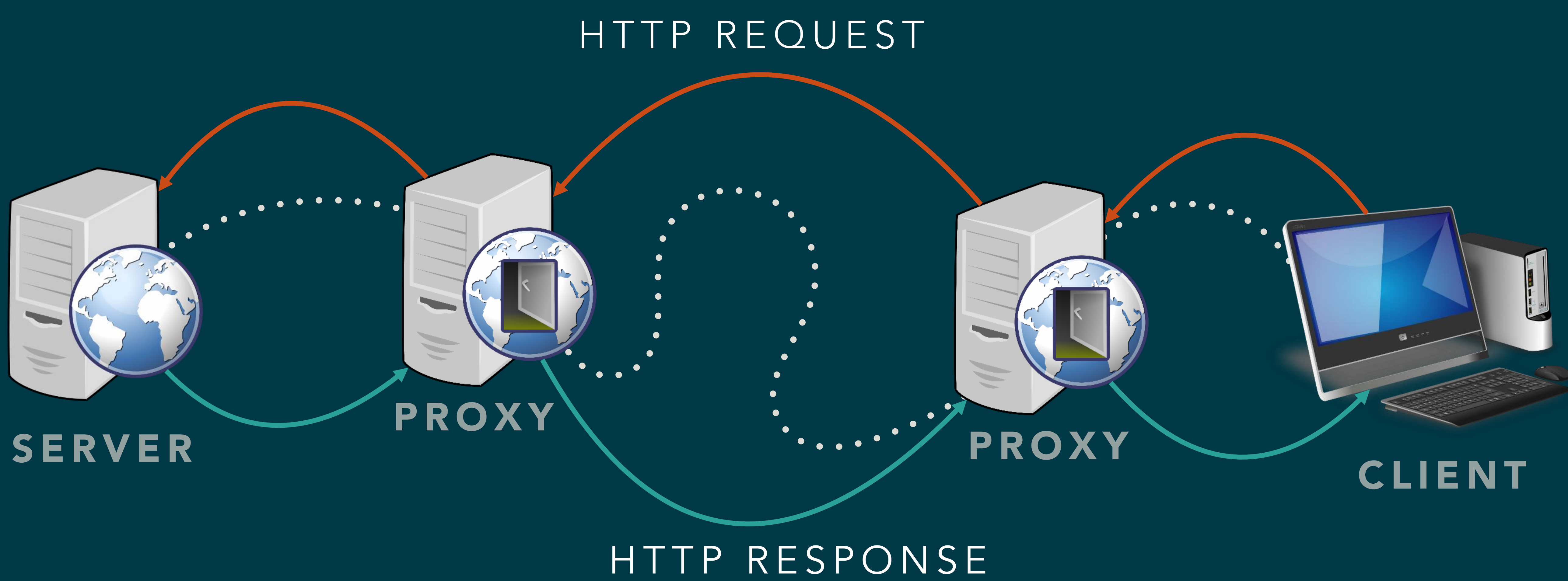
- **Connection** — a **virtual** circuit established between two programs for the purpose of communication.
- **Proxy** — an intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.

CONNECTIONS AND PROXIES



- **Connection** — a **virtual** circuit established between two programs for the purpose of communication.
- **Proxy** — an intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.

CONNECTIONS AND PROXIES



- **Connection** — a **virtual** circuit established between two programs for the purpose of communication.
- **Proxy** — an intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.

TYPES OF PROXIES: TRANSPARENCY

- A **transparent** proxy — does not modify the request or response; client is unaware of its existence:
  - load-balancing
  - monitoring, logging, debugging

TYPES OF PROXIES: TRANSPARENCY

- A **transparent** proxy — does not modify the request or response; client is unaware of its existence:
  - load-balancing
  - monitoring, logging, debugging
- A **non-transparent proxy** — modifies the request or response in order to provide some added **service**:
  - content filtering
  - removing confidential data
  - providing online anonymity

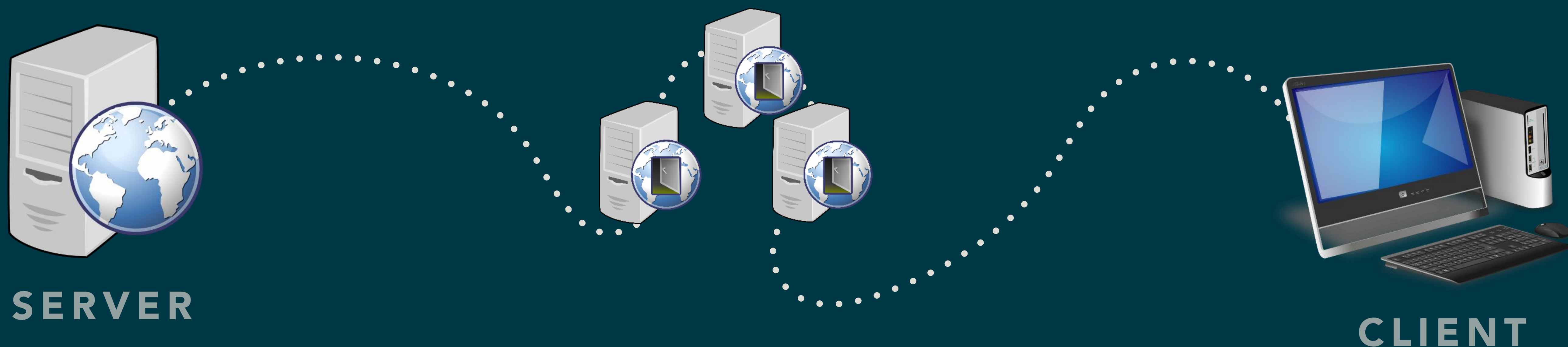
FORWARD AND REVERSE PROXIES

SERVER

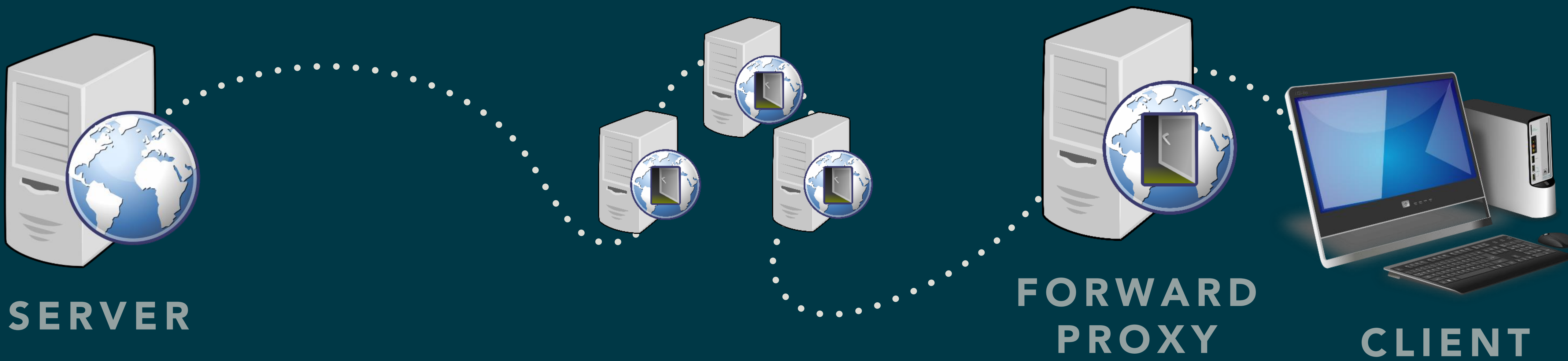
CLIENT

FORWARD AND REVERSE PROXIES

FORWARD AND REVERSE PROXIES



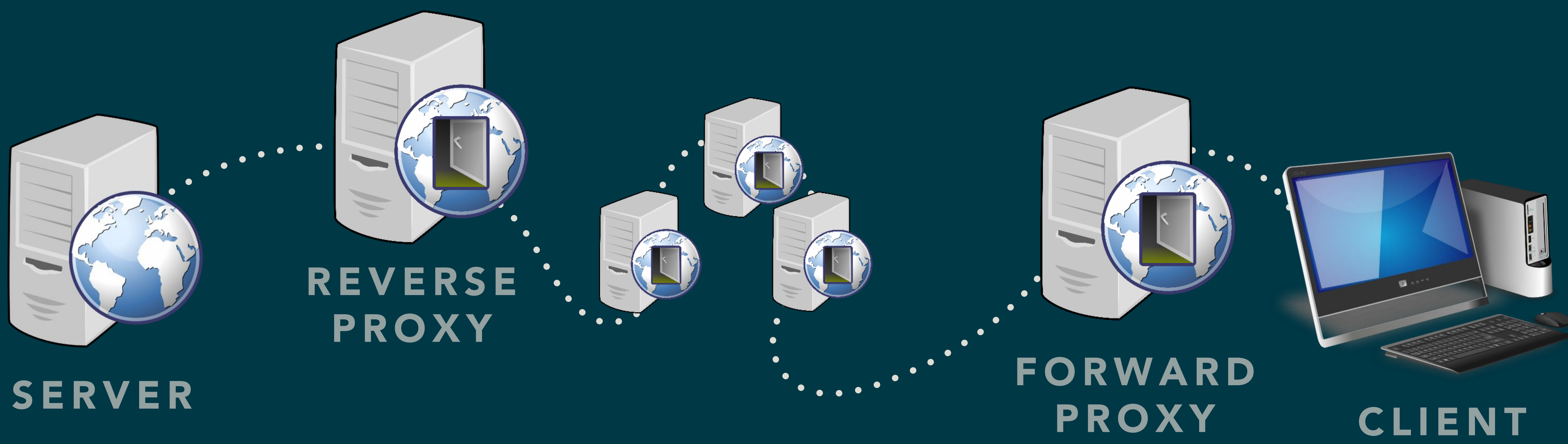
FORWARD AND REVERSE PROXIES



- **Forward** proxy — proxies in behalf of requesting hosts, each client must be configured to explicitly use this proxy.

- **Reverse** proxy — proxies in behalf of servers, appears to clients as ordinary server, used to take the computational load off the web servers, e.g. **TLS acceleration**.

FORWARD AND REVERSE PROXIES



- **Forward** proxy — proxies in behalf of requesting hosts, each client must be configured to explicitly use this proxy.

- **Reverse** proxy — proxies in behalf of servers, appears to clients as ordinary server, used to take the computational load off the web servers, e.g. **TLS acceleration**.

HTTP CACHING

- **HTTP caching** — a set of mechanisms allowing HTTP responses to be held in some form of temporary storage.
- Instead of satisfying future requests by going back to the original data source, the **saved** copy of the data can be used.

HTTP CACHING

- **HTTP caching** — a set of mechanisms allowing HTTP responses to be held in some form of temporary storage.
- Instead of satisfying future requests by going back to the original data source, the **saved** copy of the data can be used.
- Caching can reduce **latency**, help prevent **bandwith bottlenecks** as well as improve user experience.

HTTP CACHING

- **HTTP caching** — a set of mechanisms allowing HTTP responses to be held in some form of temporary storage.
- Instead of satisfying future requests by going back to the original data source, the **saved** copy of the data can be used.
- Caching can reduce **latency**, help prevent **bandwith bottlenecks** as well as improve user experience.
- Two types of caches can be employed:
  - **public** cache — shared among multiple users, resides on a **proxy** (forward or reverse).
  - **private** cache — stored by a **browser** for a single user.

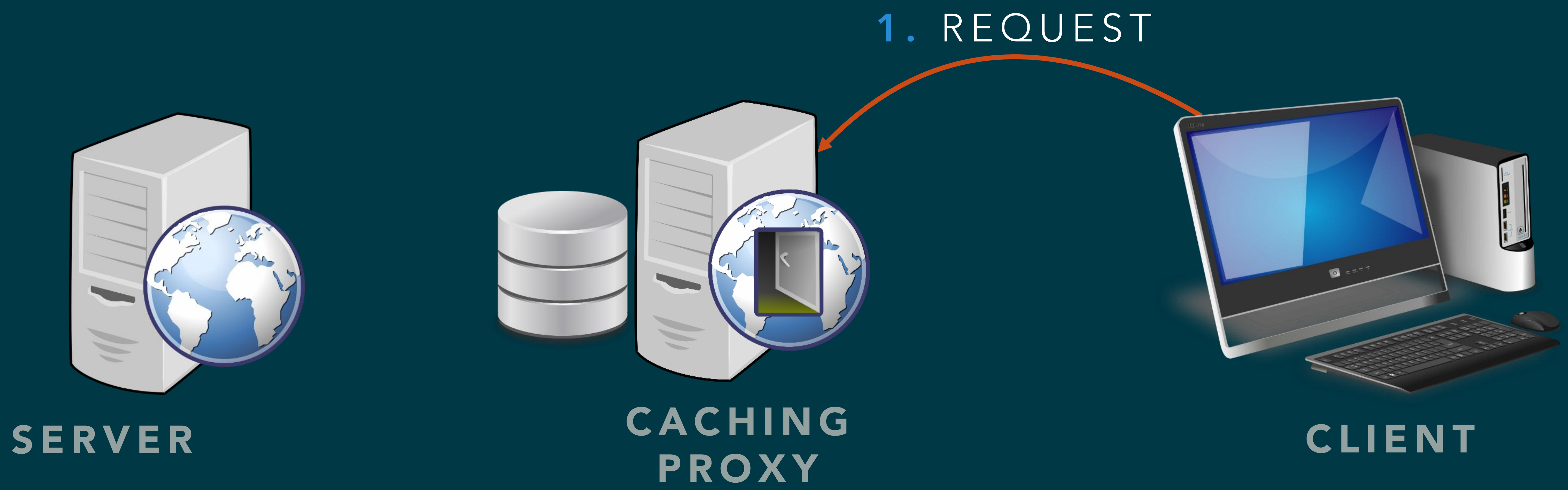
CACHING IN ACTION

SERVER

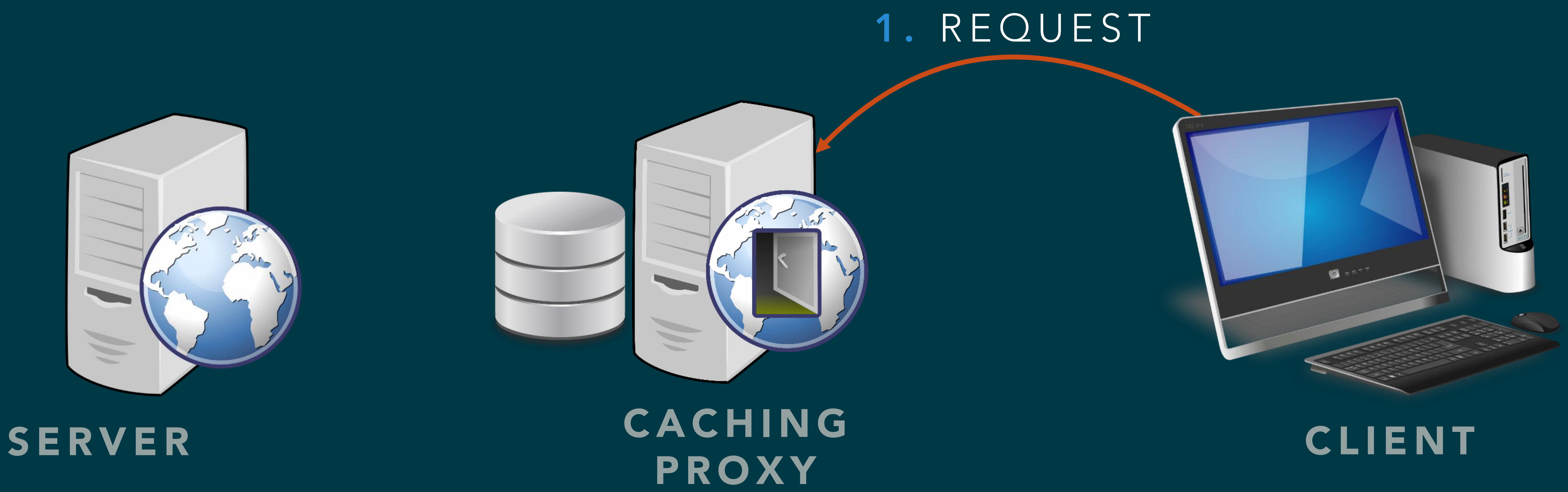
CACHING PROXY

CLIENT

CACHING IN ACTION

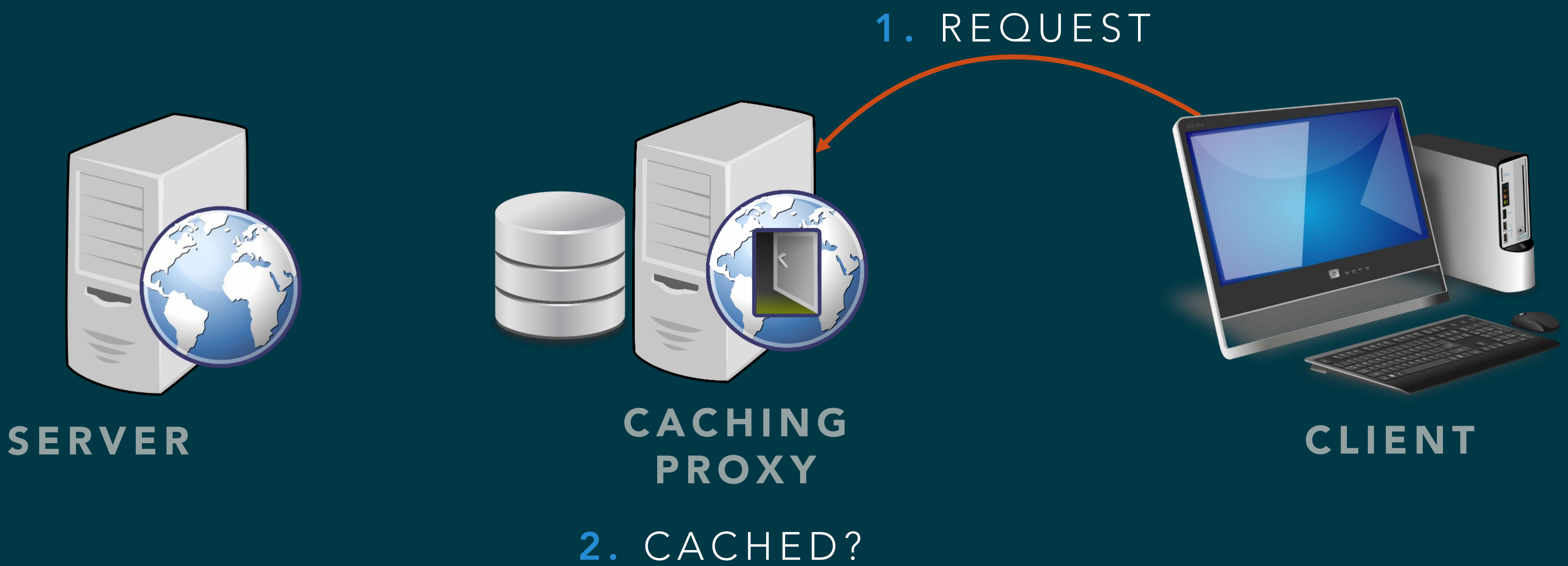


CACHING IN ACTION



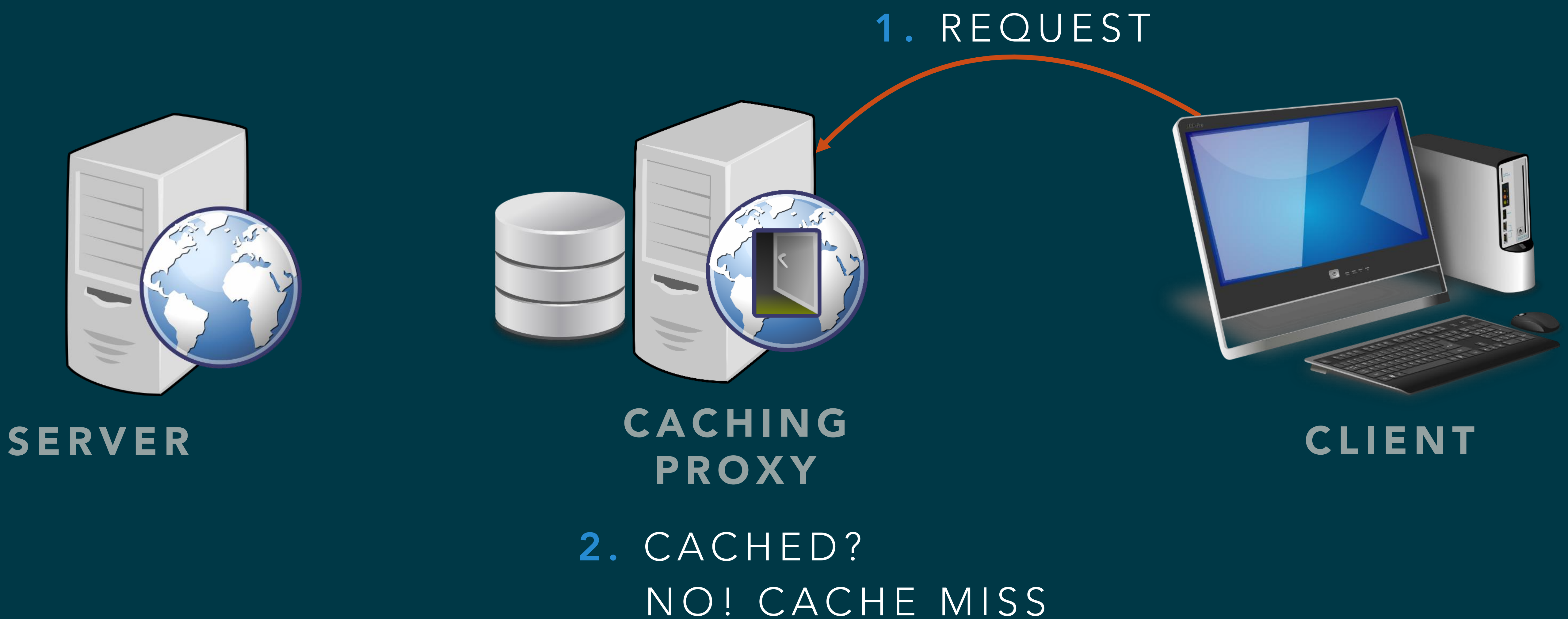
- Request methods can be defined as **"cacheable"** to indicate that responses to them are allowed to be **stored** for future reuse. In general, **GET** and **HEAD** are cacheable.

CACHING IN ACTION



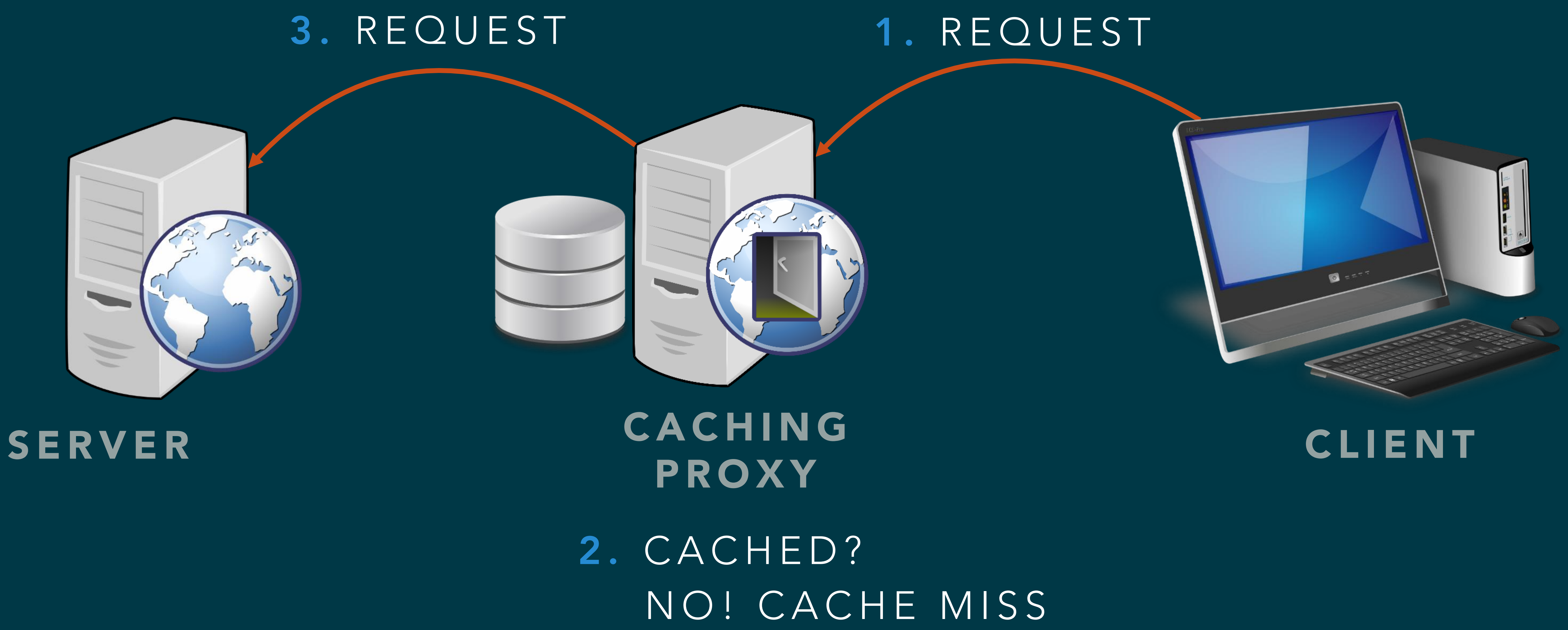
- Request methods can be defined as **"cacheable"** to indicate that responses to them are allowed to be **stored** for future reuse. In general, **GET** and **HEAD** are cacheable.

CACHING IN ACTION



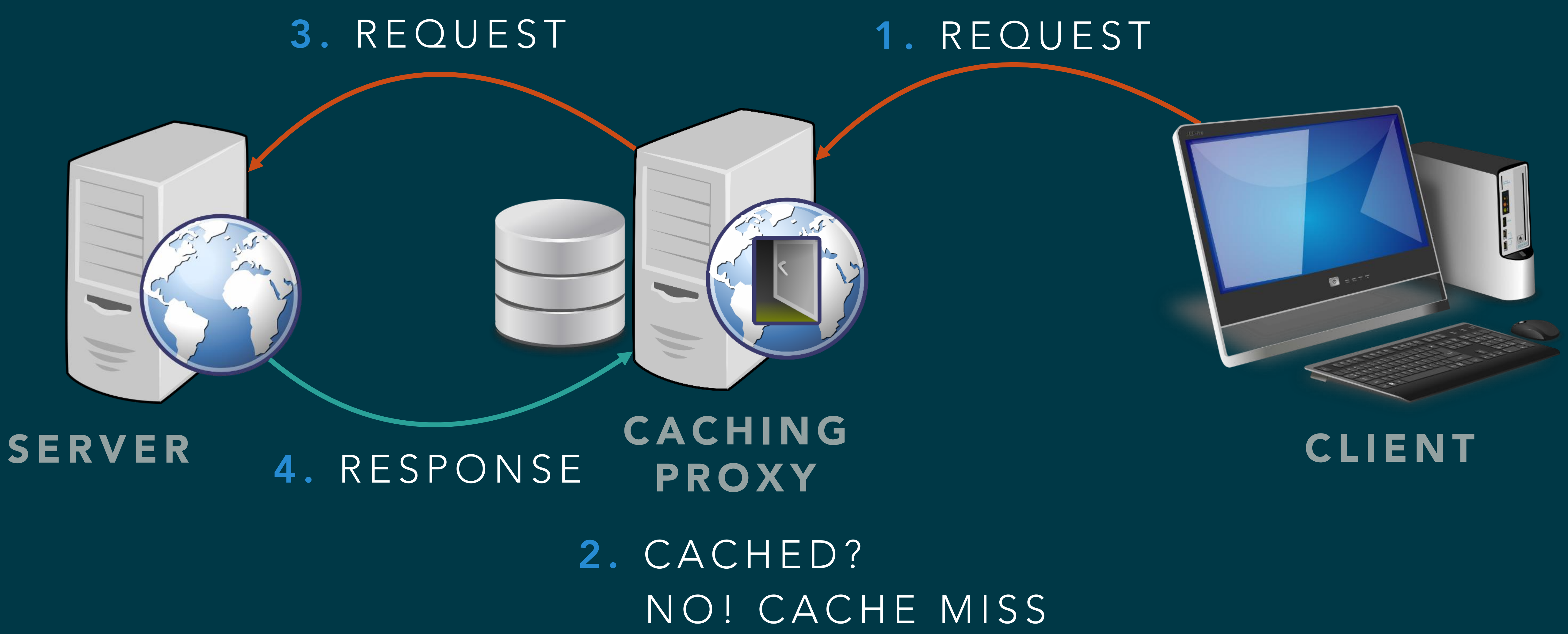
- Request methods can be defined as **"cacheable"** to indicate that responses to them are allowed to be **stored** for future reuse. In general, **GET** and **HEAD** are cacheable.

CACHING IN ACTION



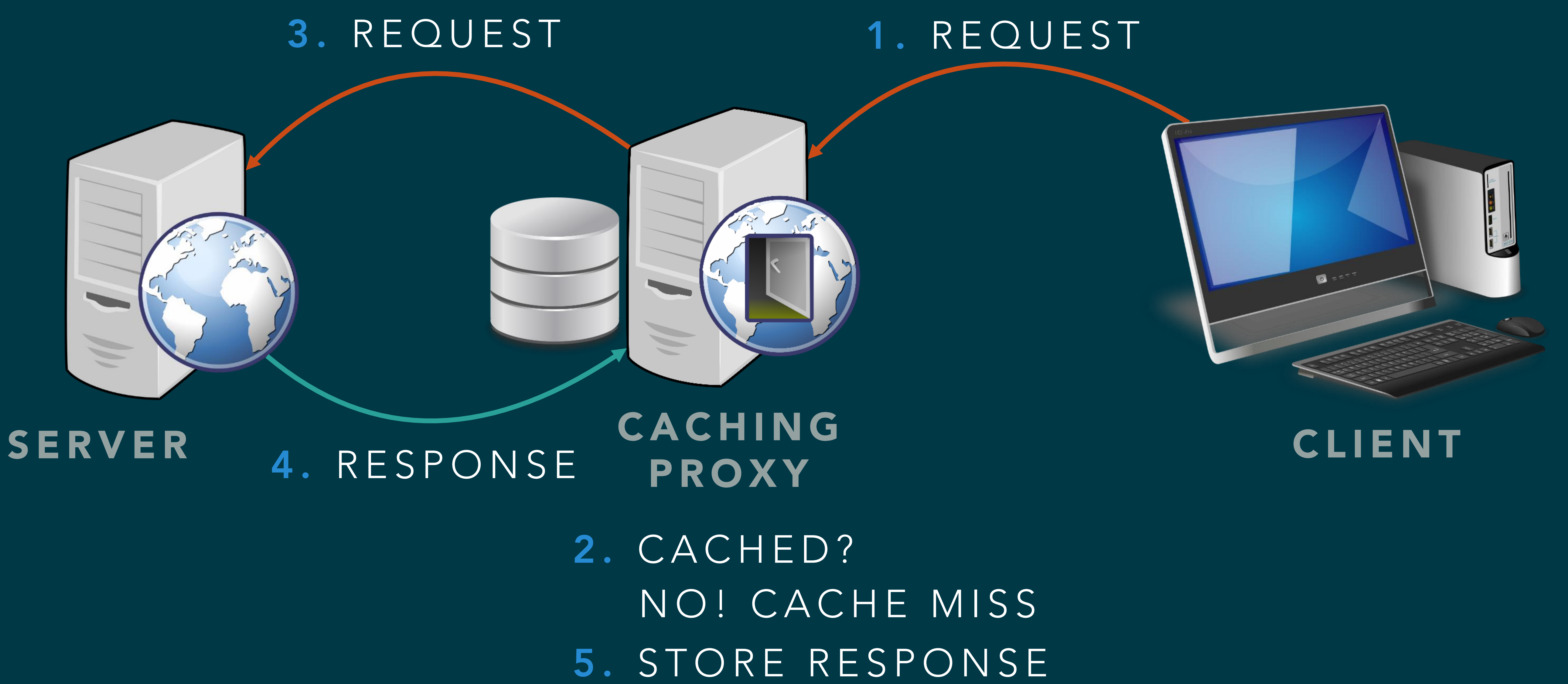
- Request methods can be defined as **"cacheable"** to indicate that responses to them are allowed to be **stored** for future reuse. In general, **GET** and **HEAD** are cacheable.

CACHING IN ACTION



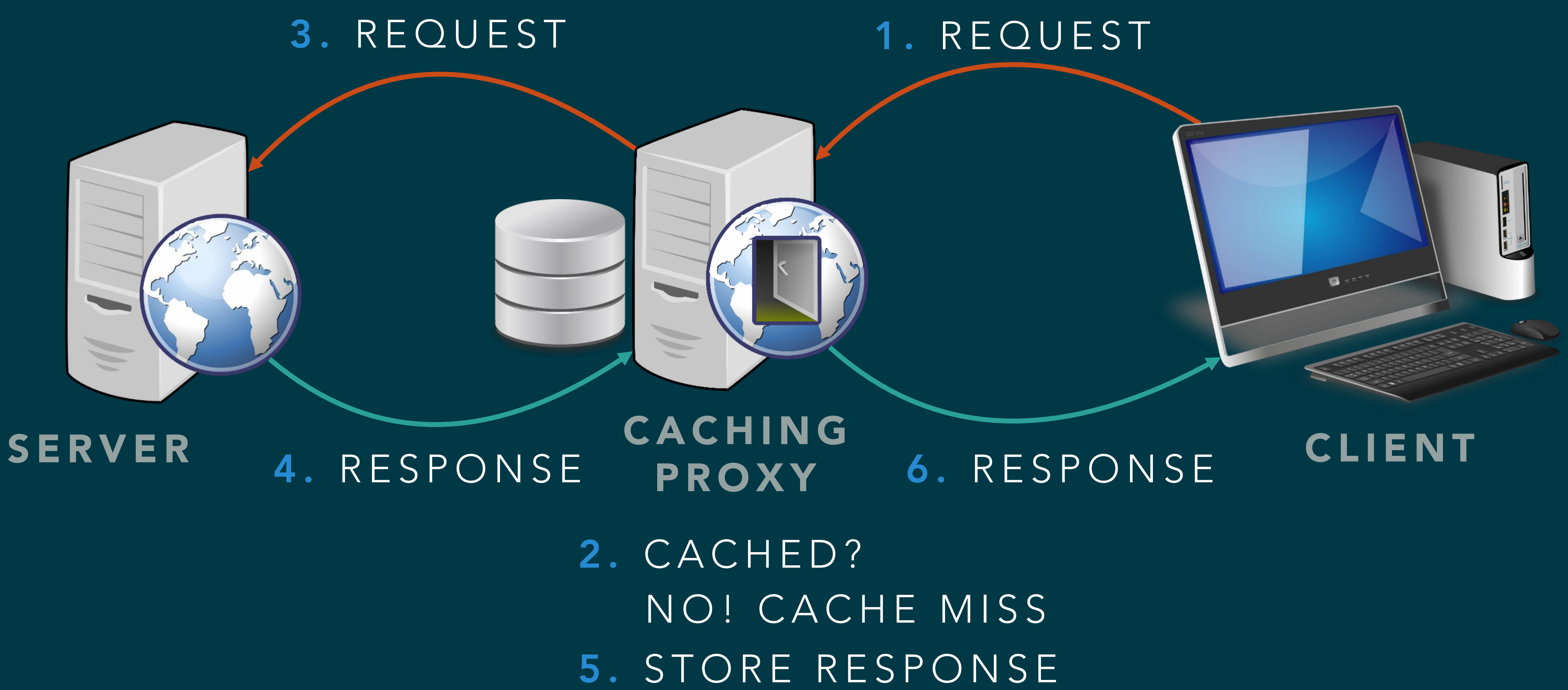
- Request methods can be defined as **"cacheable"** to indicate that responses to them are allowed to be **stored** for future reuse. In general, **GET** and **HEAD** are cacheable.

CACHING IN ACTION



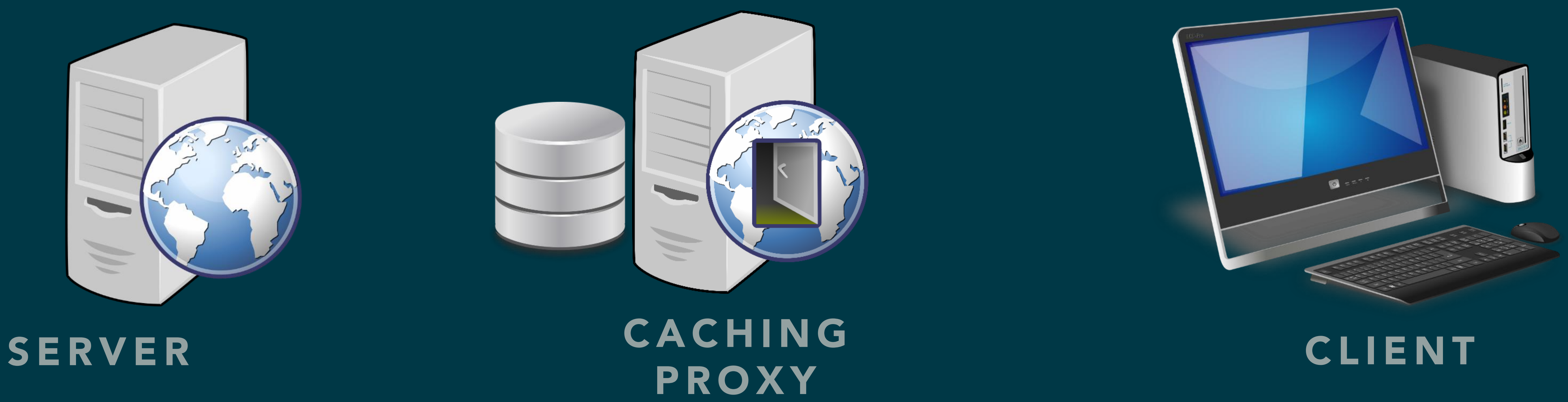
- Request methods can be defined as **"cacheable"** to indicate that responses to them are allowed to be **stored** for future reuse. In general, **GET** and **HEAD** are cacheable.

CACHING IN ACTION

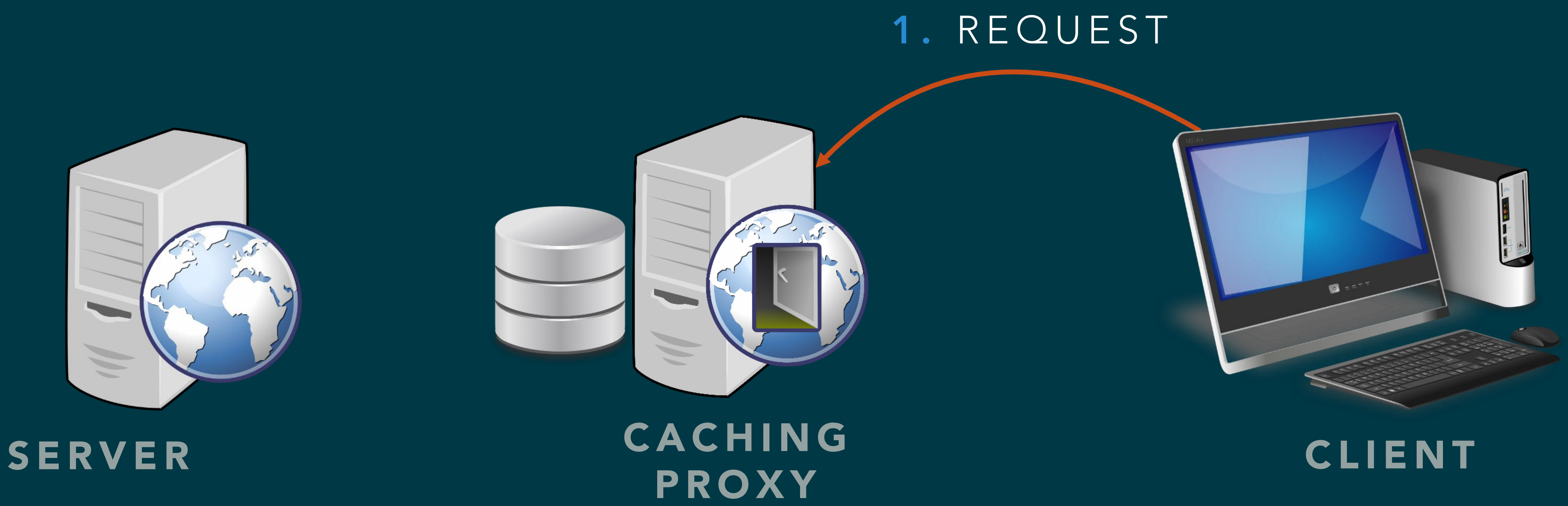


- Request methods can be defined as **"cacheable"** to indicate that responses to them are allowed to be **stored** for future reuse. In general, **GET** and **HEAD** are cacheable.

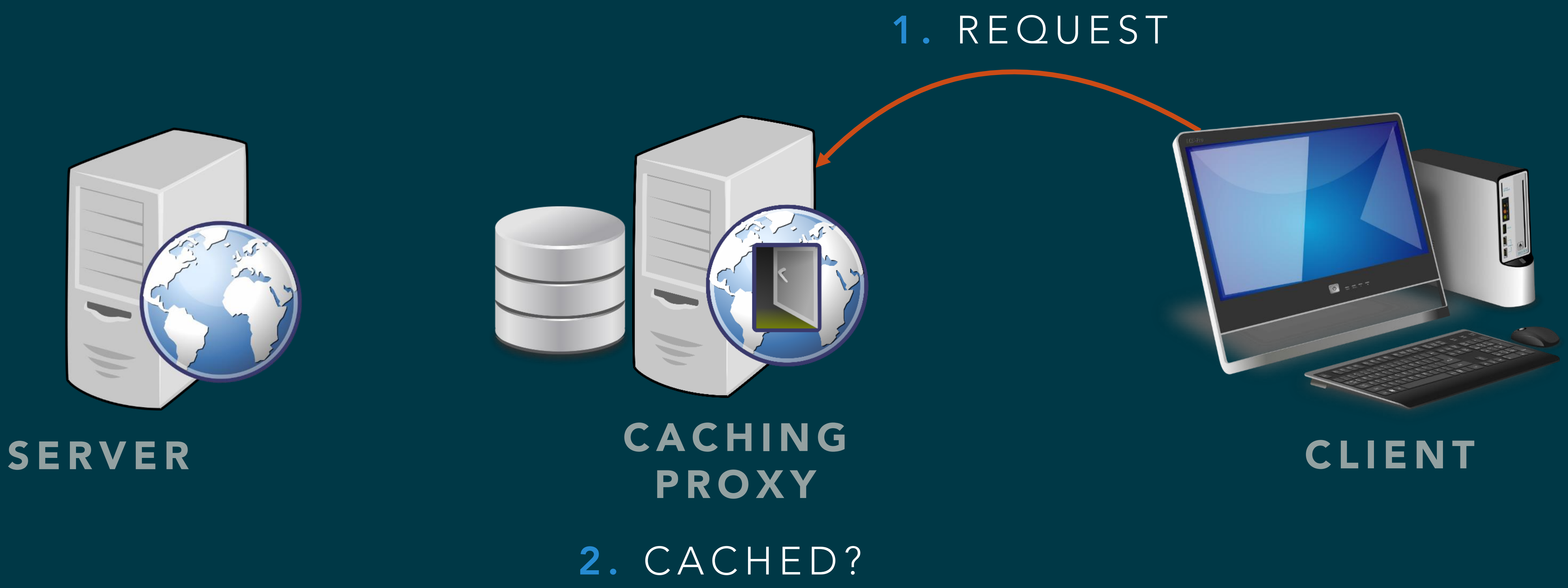
CACHE HIT



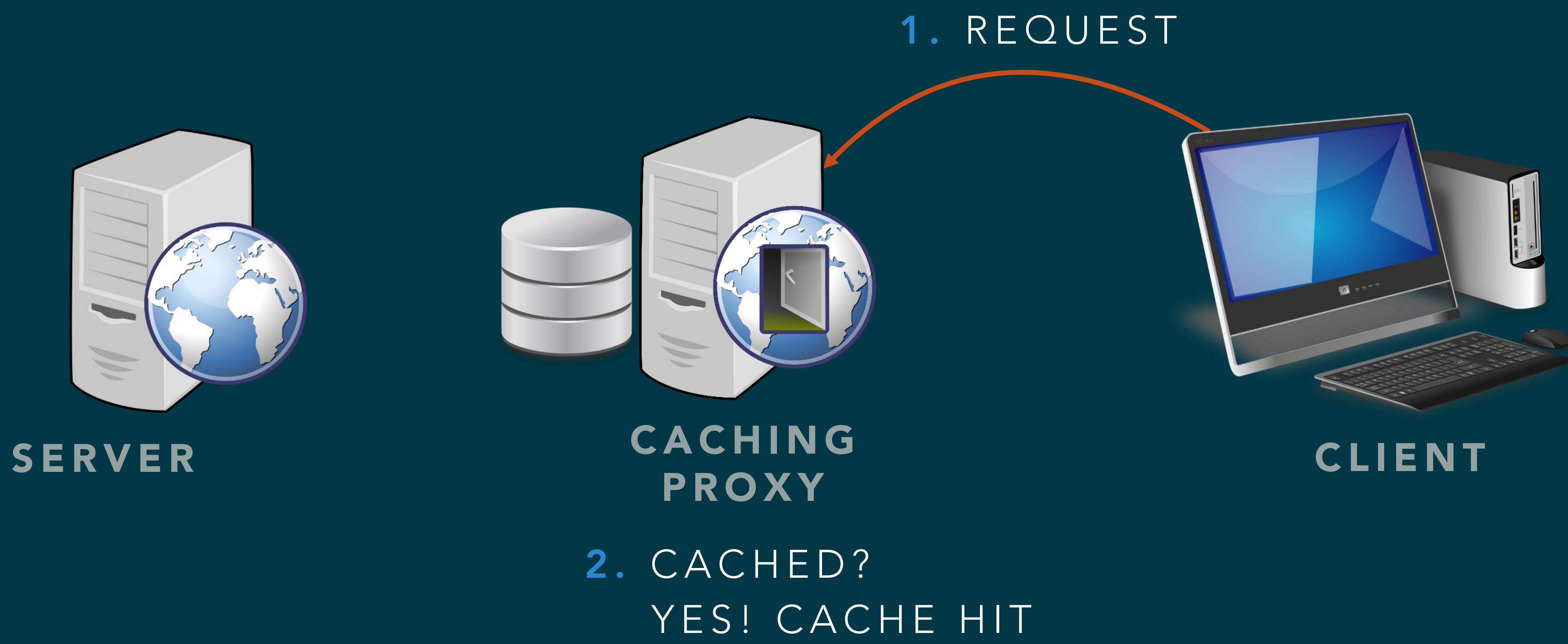
CACHE HIT



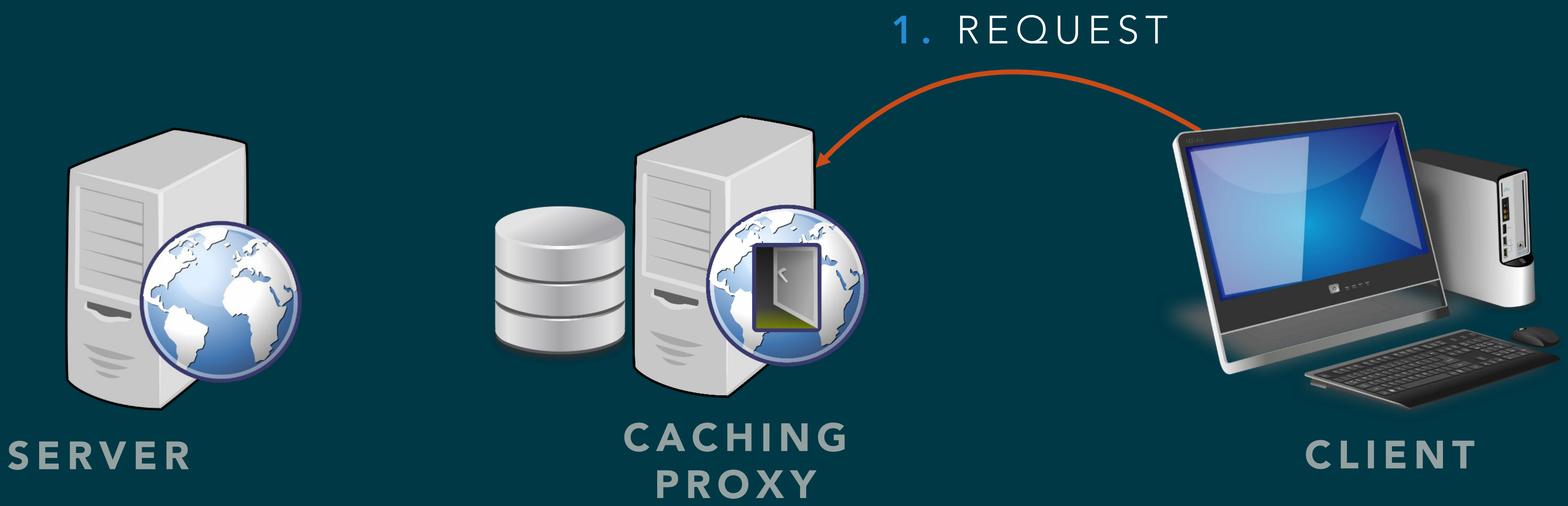
CACHE HIT



CACHE HIT


1. REQUEST
SERVER
CACHING PROXY
CLIENT
2. CACHED?
YES! CACHE HIT

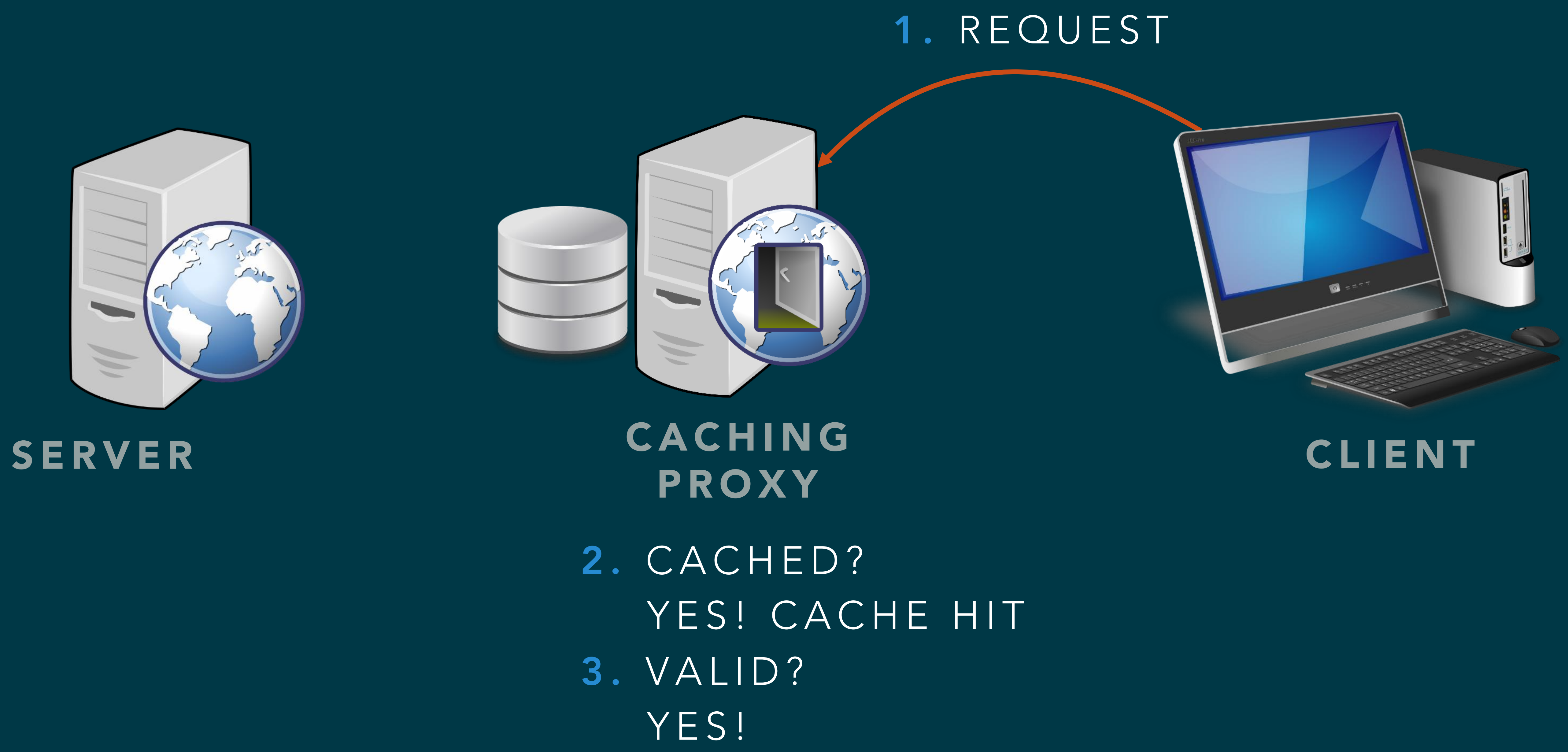
CACHE HIT



2. CACHED?
   YES! CACHE HIT
3. VALID?

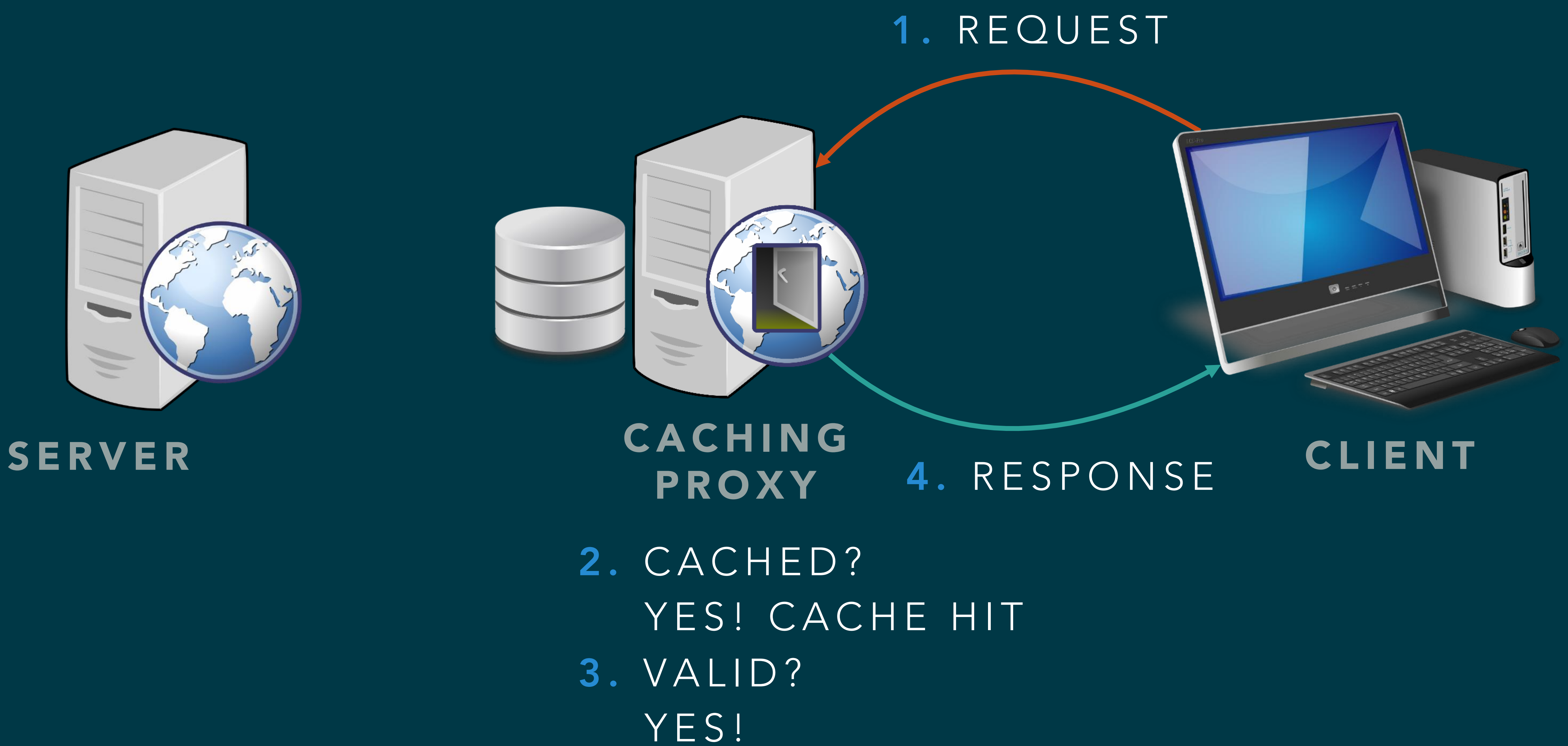
- Keeping the content fresh is one of the primary responsibilities of the cache — HTTP provides a simple mechanism of **document expiration**.

CACHE HIT



- Keeping the content fresh is one of the primary responsibilities of the cache — HTTP provides a simple mechanism of **document expiration**.

CACHE HIT



- Keeping the content fresh is one of the primary responsibilities of the cache — HTTP provides a simple mechanism of **document expiration**.

DOCUMENT EXPIRATION

- HTTP server can attach an **expiration date** to each response using the `Cache-Control` and `Expires` headers.

```
HTTP/1.1 200 OK
Last-Modified: Wed, 25 Jan 2012 17:55:15 GMT
Expires: Sat, 22 Jan 2022 17:55:15 GMT
Cache-Control: max-age=315360000,public
```

- The cache can serve the copy as long as the age of the document is within the expiration date.

DOCUMENT EXPIRATION

- HTTP server can attach an **expiration date** to each response using the `Cache-Control` and `Expires` headers.

```
HTTP/1.1 200 OK
Last-Modified: Wed, 25 Jan 2012 17:55:15 GMT
Expires: Sat, 22 Jan 2022 17:55:15 GMT
Cache-Control: max-age=315360000,public
```

- The cache can serve the copy as long as the age of the document is within the expiration date.

- Once a cached document expires, the cache must **revalidate** with the server to check if the document has changed and **update** its local copy accordingly.

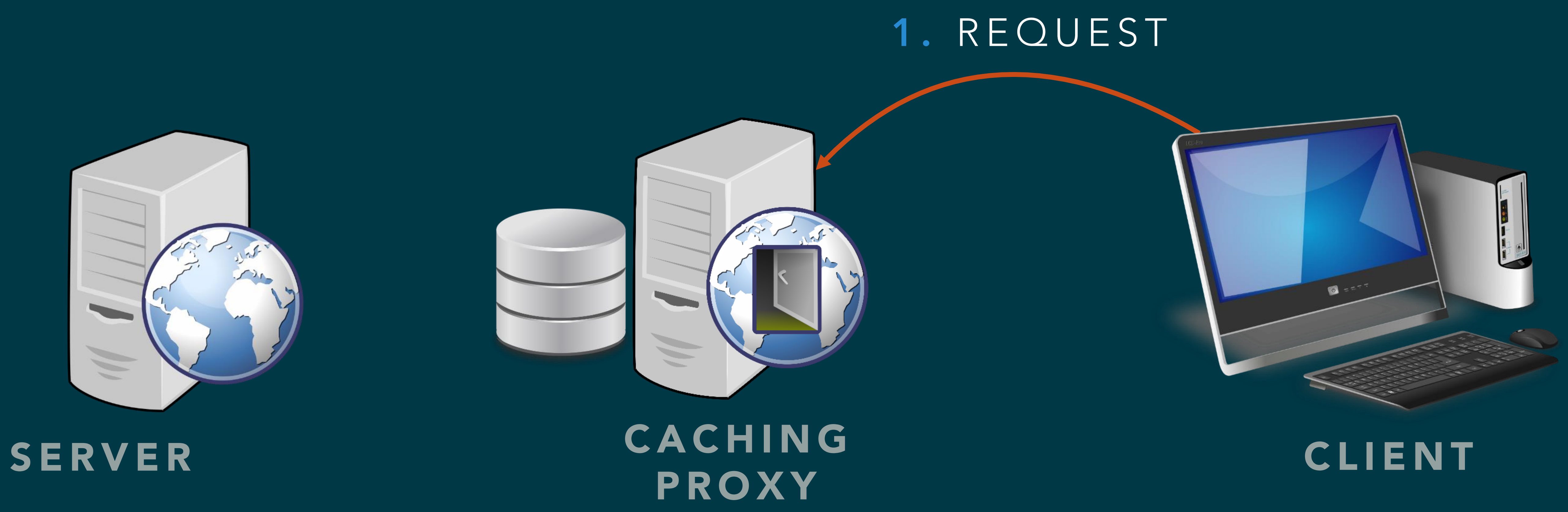
REVALIDATE HIT

SERVER

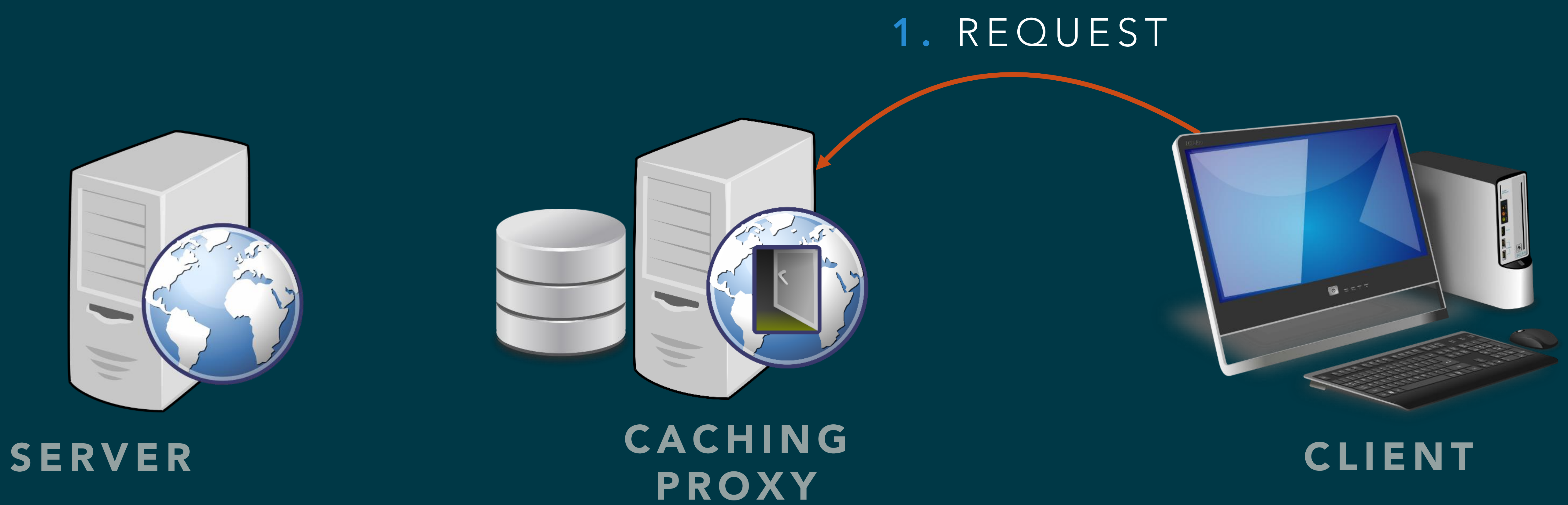
CACHING PROXY

CLIENT

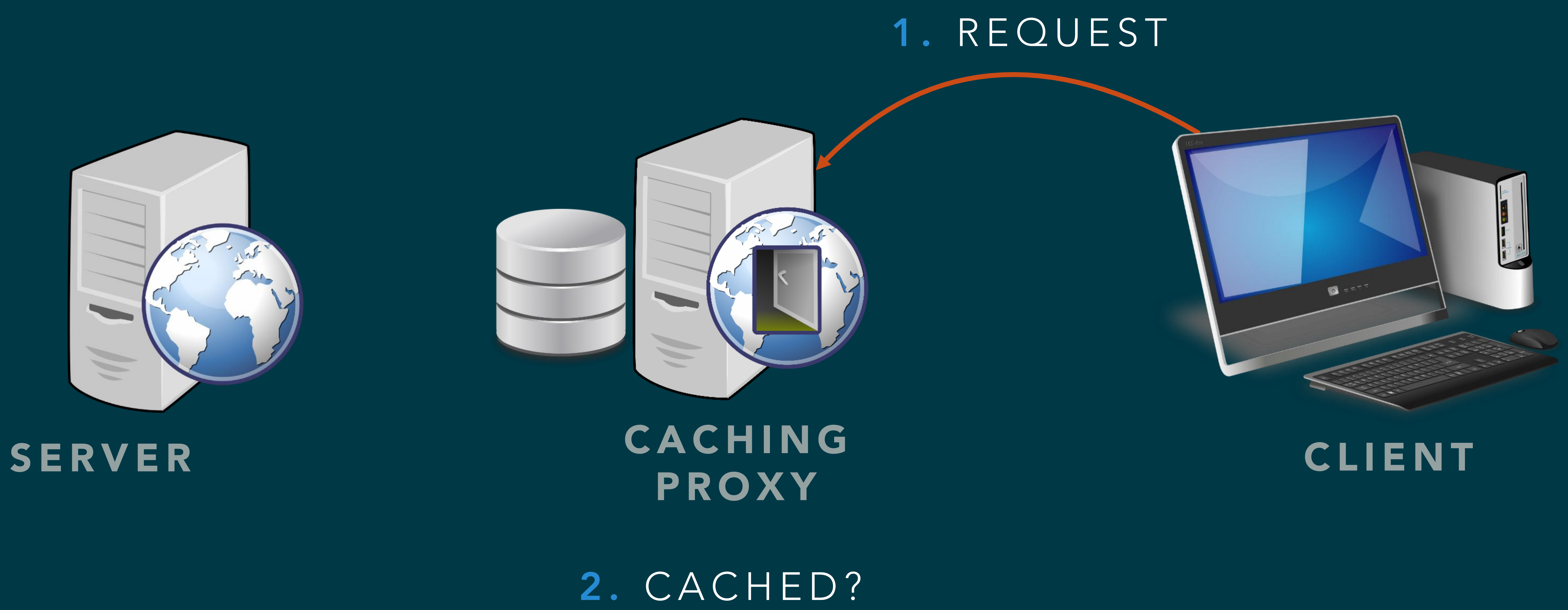
REVALIDATE HIT



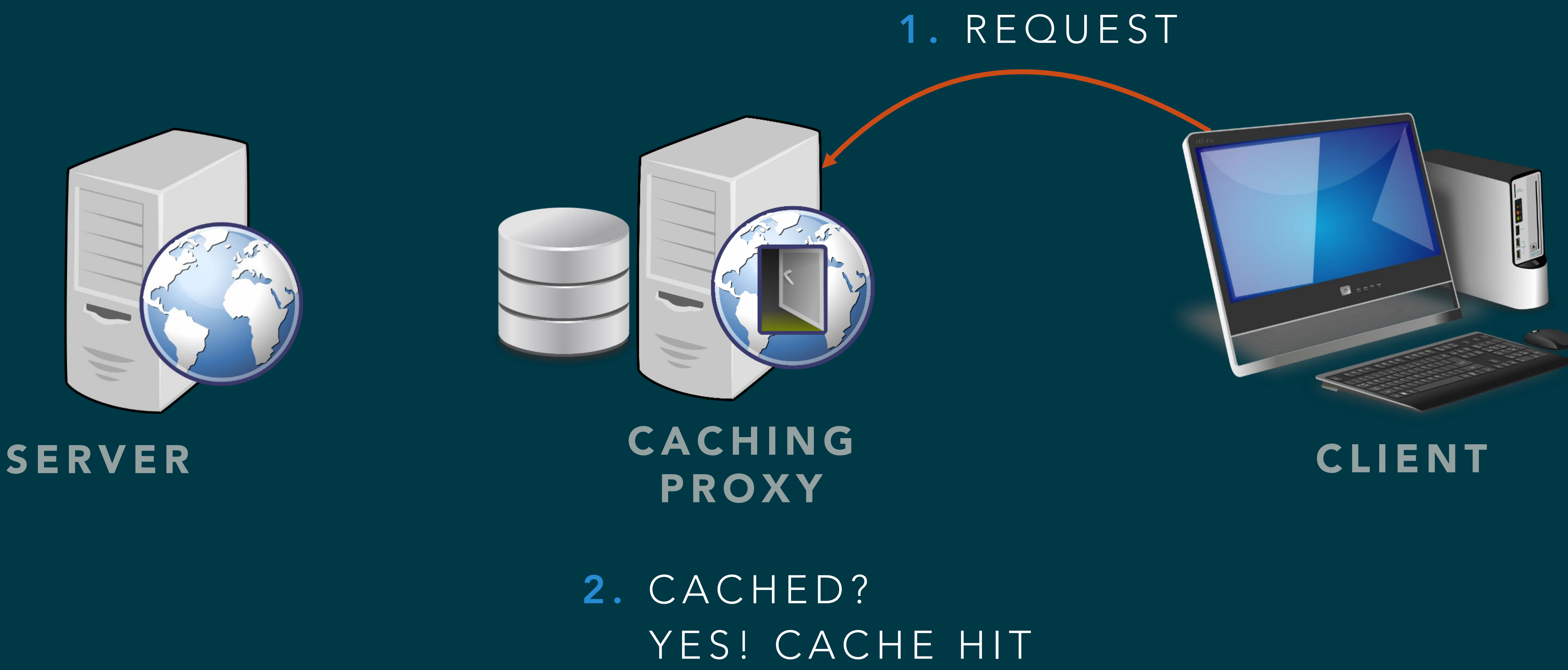
REVALIDATE HIT



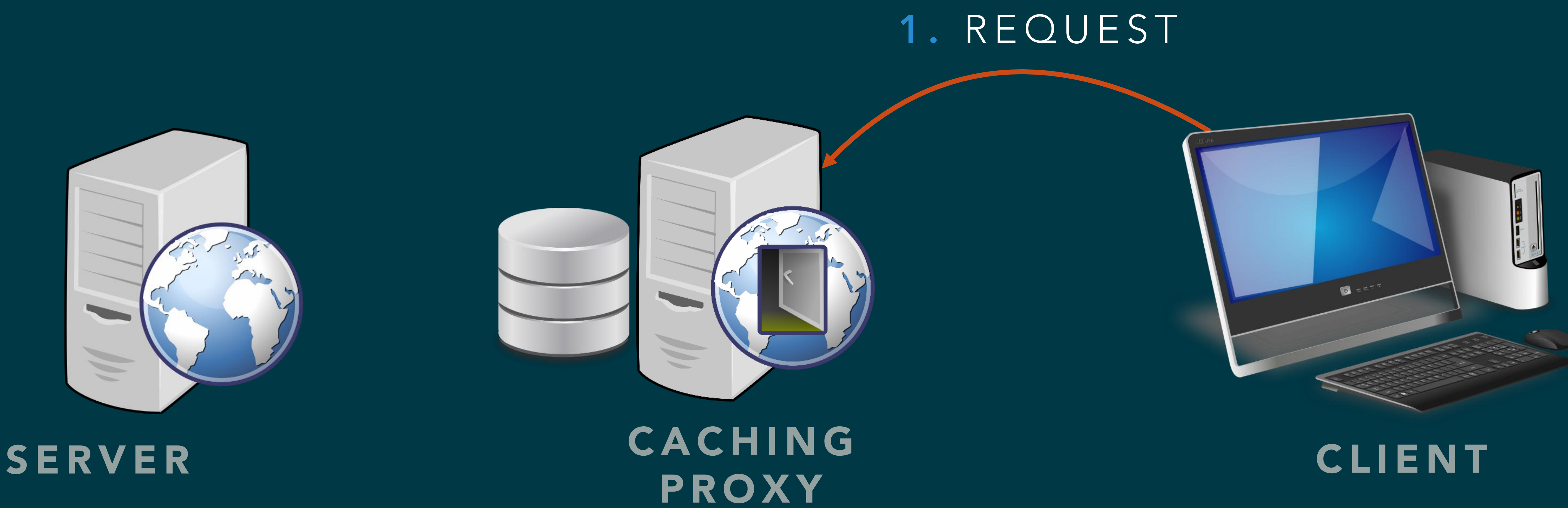
REVALIDATE HIT



REVALIDATE HIT

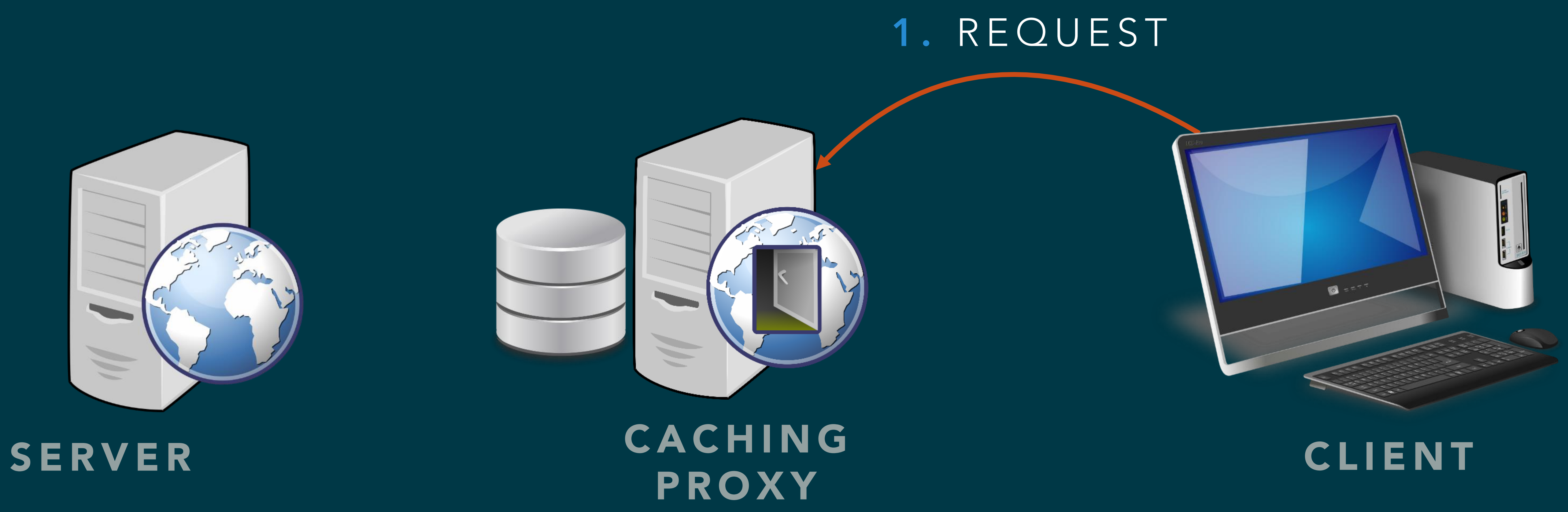


REVALIDATE HIT



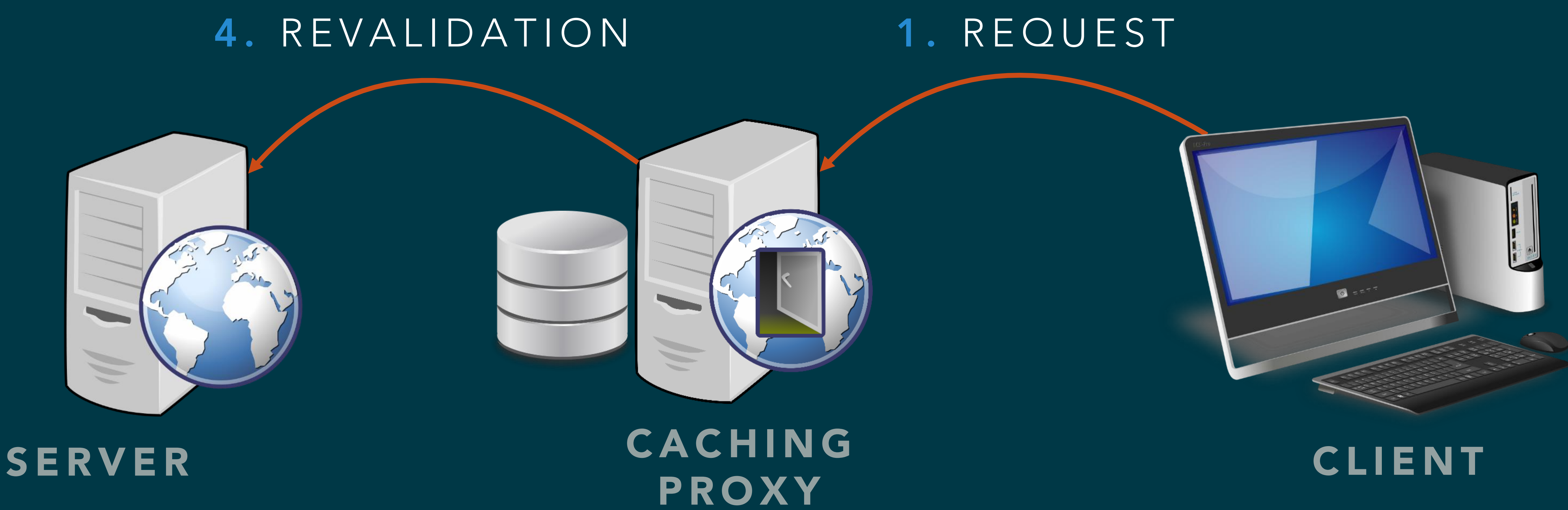
2. CACHED?
   YES! CACHE HIT
3. VALID?

REVALIDATE HIT



2. CACHED?
YES! CACHE HIT
3. VALID?
NO! EXPIRED

REVALIDATE HIT



2. CACHED?
YES! CACHE HIT
3. VALID?
NO! EXPIRED

REVALIDATE HIT

4. REVALIDATION

1. REQUEST

SERVER

CACHING PROXY

CLIENT

5. VALID?

2. CACHED?
   YES! CACHE HIT
3. VALID?
   NO! EXPIRED

REVALIDATE HIT

4. REVALIDATION

1. REQUEST

SERVER

CACHING PROXY

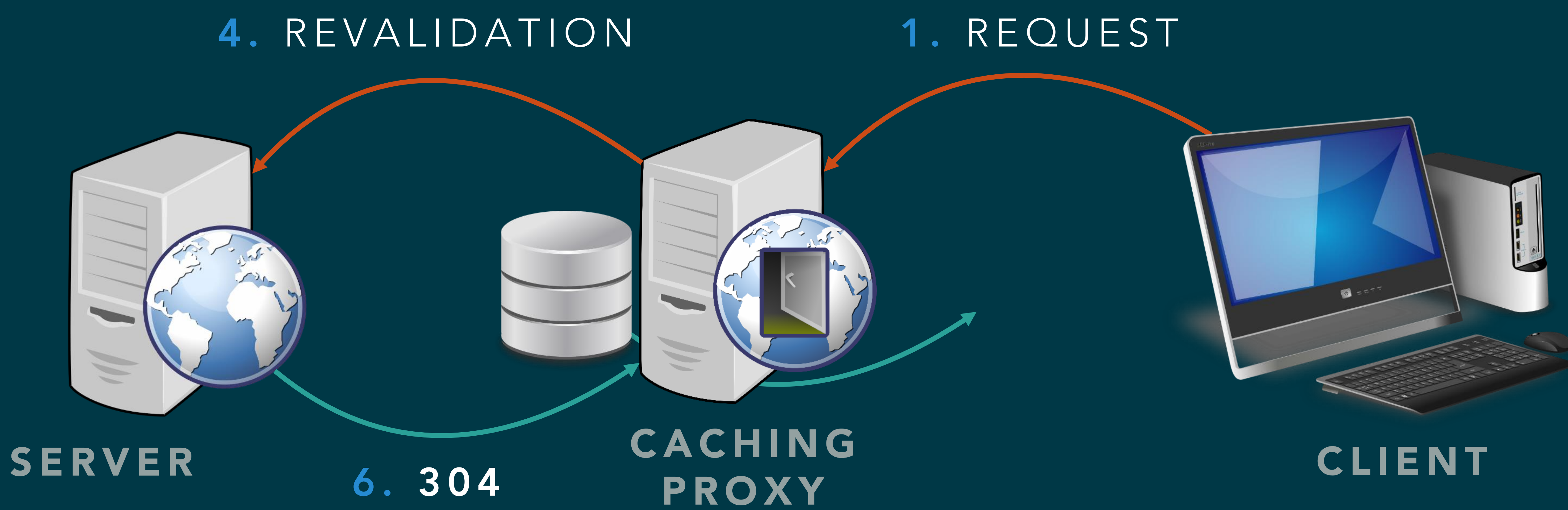
CLIENT

5. VALID?
YES!
REVALIDATE HIT

2. CACHED?
YES! CACHE HIT

3. VALID?
NO! EXPIRED

REVALIDATE HIT

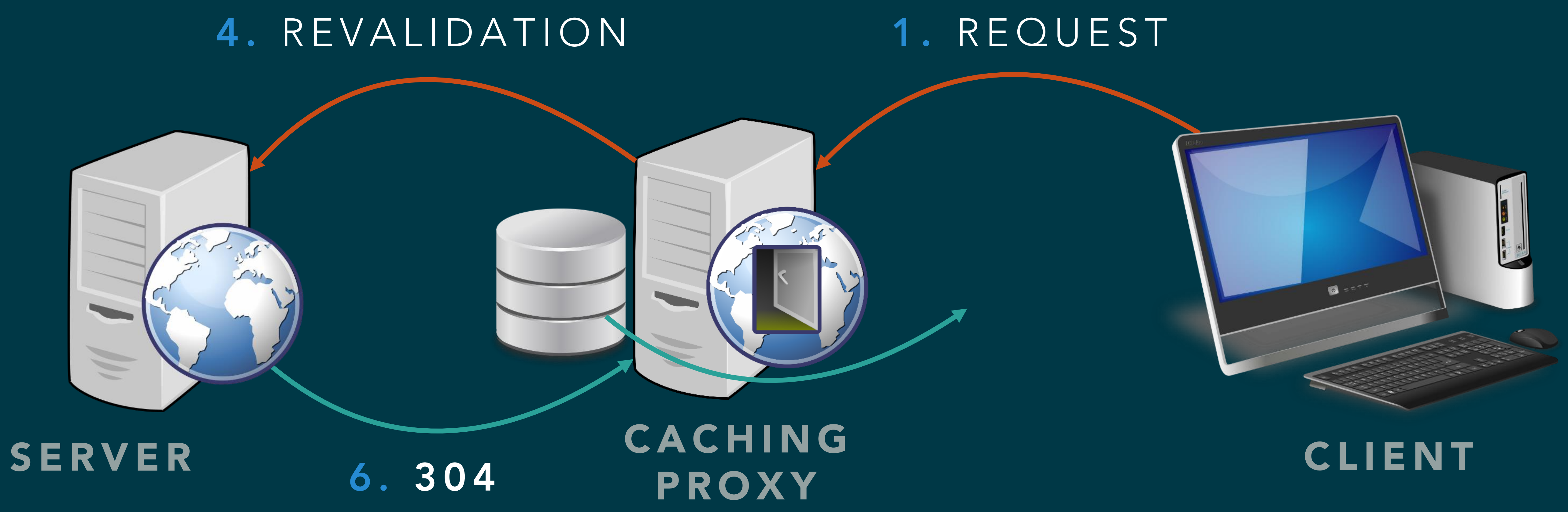


5. VALID?
YES!
REVALIDATE HIT

2. CACHED?
YES! CACHE HIT

3. VALID?
NO! EXPIRED

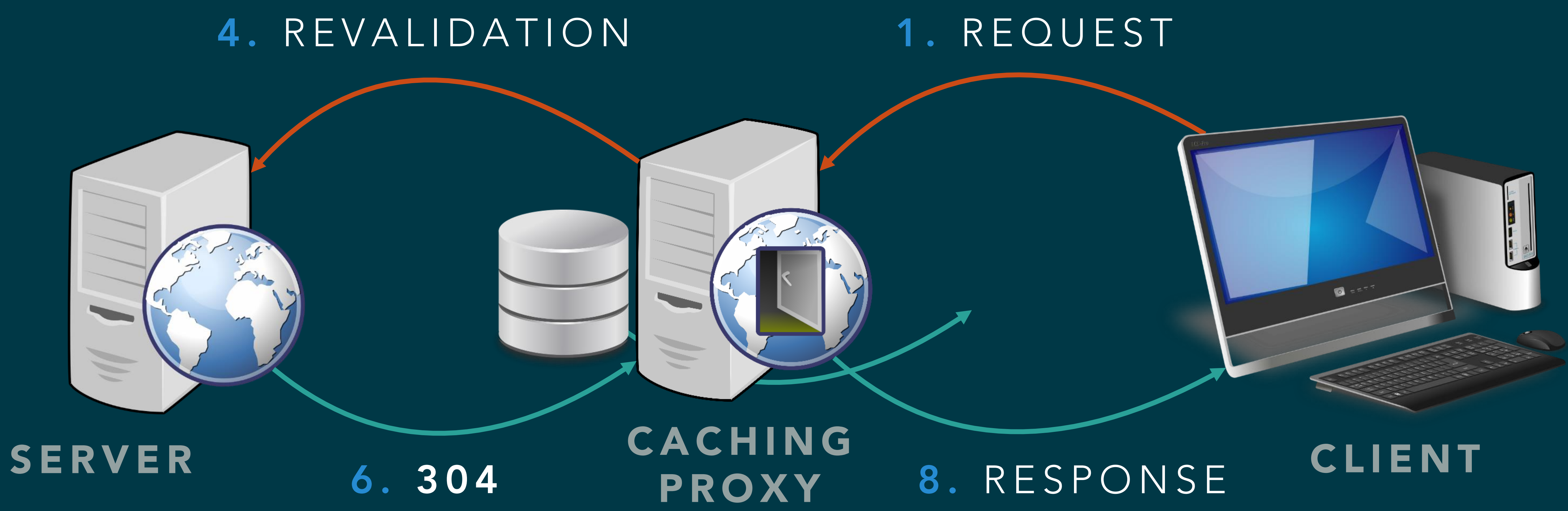
REVALIDATE HIT



5. VALID?
   YES!
   REVALIDATE HIT

2. CACHED?
   YES! CACHE HIT
3. VALID?
   NO! EXPIRED
7. UPDATE EXPIRATION DATE

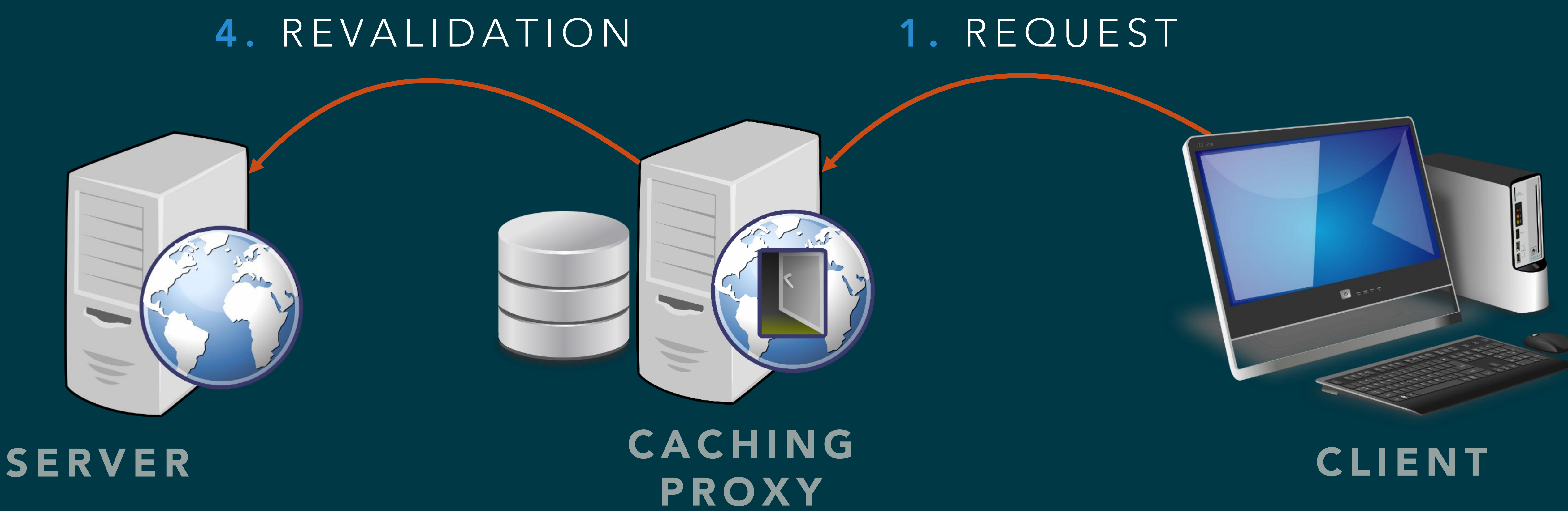
REVALIDATE HIT



5. VALID?
   YES!
   REVALIDATE HIT

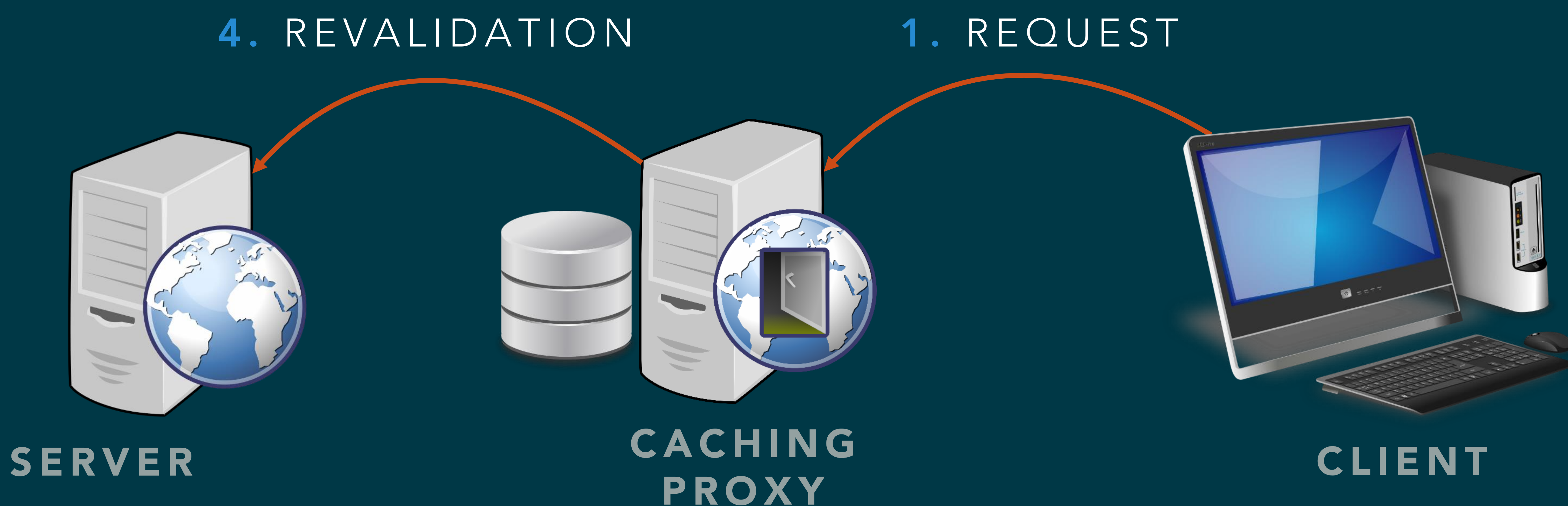
2. CACHED?
   YES! CACHE HIT
3. VALID?
   NO! EXPIRED
7. UPDATE EXPIRATION DATE

REVALIDATE MISS



2. CACHED?
   YES! CACHE HIT
3. VALID?
   NO! EXPIRED

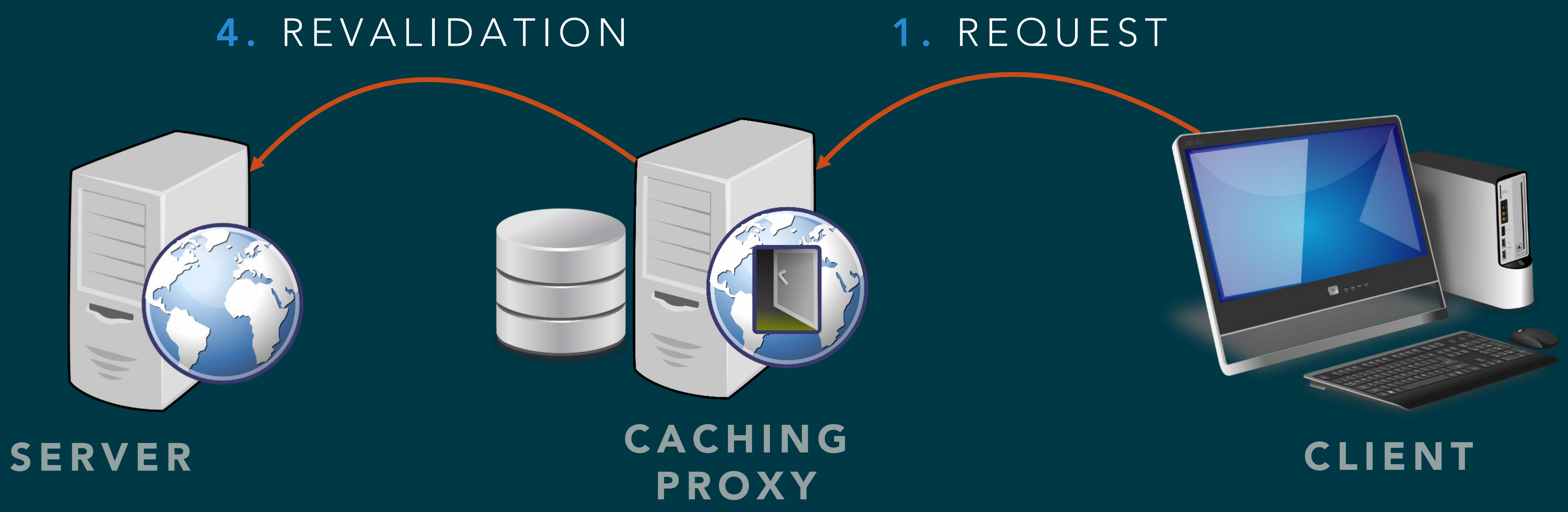
REVALIDATE MISS



5. VALID?

2. CACHED?
   YES! CACHE HIT
3. VALID?
   NO! EXPIRED

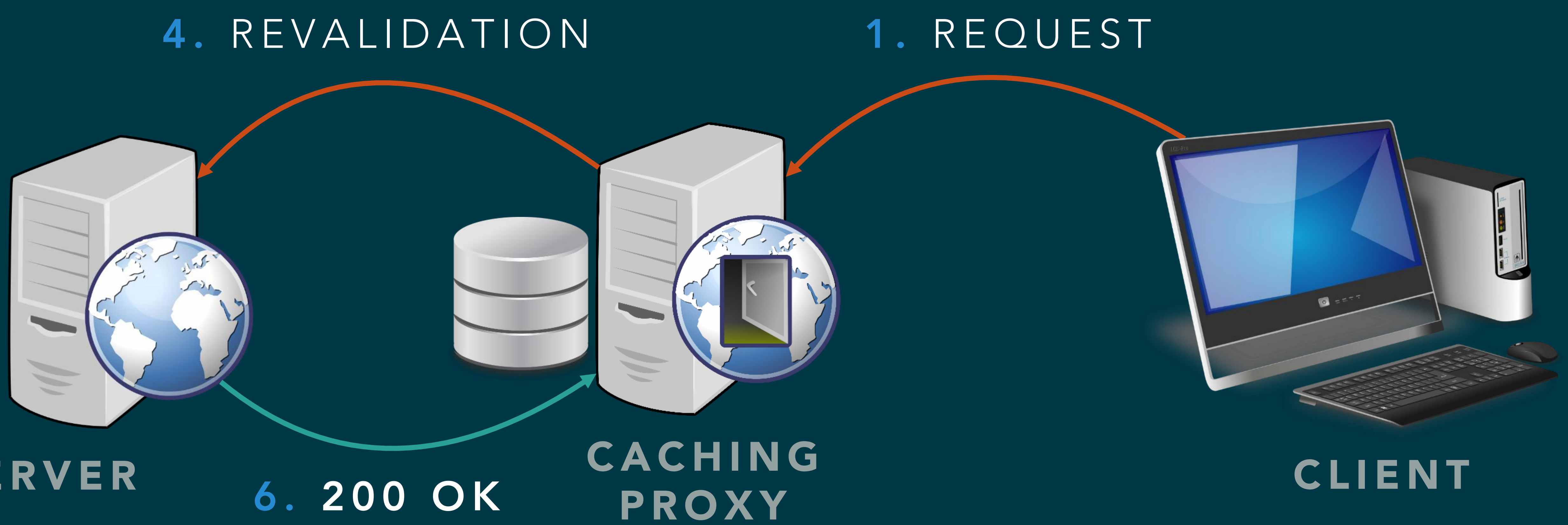
REVALIDATE MISS



5. VALID?
NO!
REVALIDATE MISS

2. CACHED?
YES! CACHE HIT
3. VALID?
NO! EXPIRED

REVALIDATE MISS



5. VALID?
NO!
REVALIDATE MISS

2. CACHED?
YES! CACHE HIT
3. VALID?
NO! EXPIRED

REVALIDATE MISS

4. REVALIDATION

1. REQUEST

SERVER

6. **200 OK**

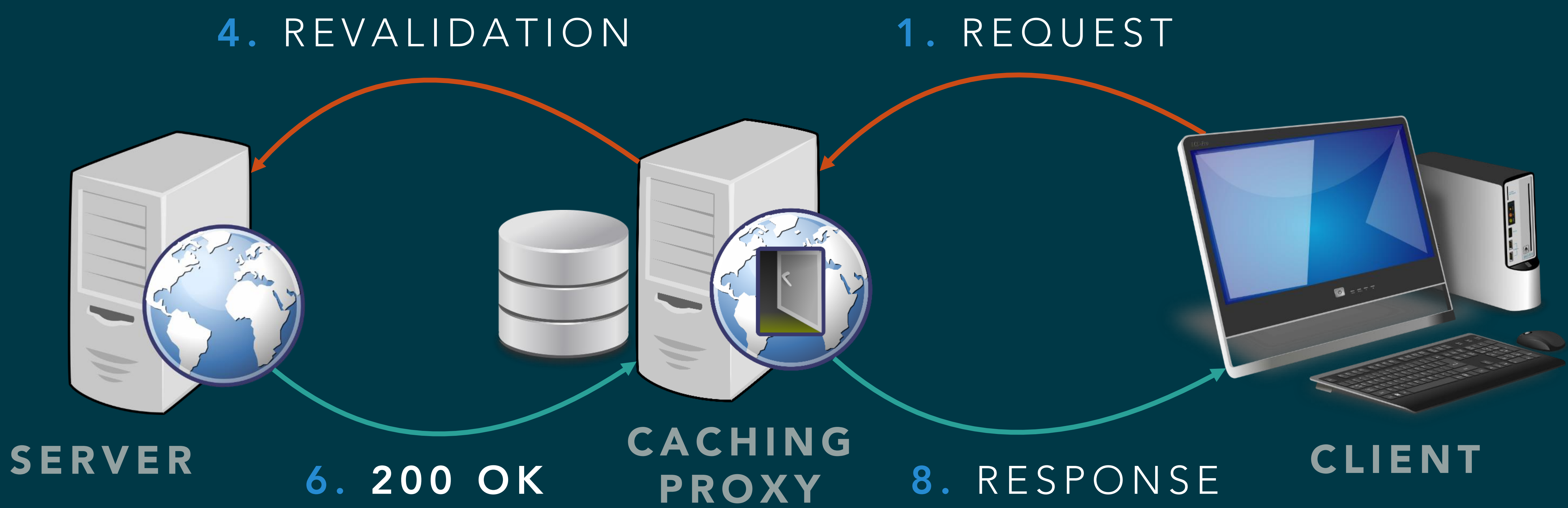
CACHING PROXY

CLIENT

5. VALID?
   NO!
   REVALIDATE MISS

2. CACHED?
   YES! CACHE HIT
3. VALID?
   NO! EXPIRED
7. STORE RESPONSE

REVALIDATE MISS



5. VALID?
NO!
REVALIDATE MISS

2. CACHED?
YES! CACHE HIT
3. VALID?
NO! EXPIRED
7. STORE RESPONSE

SERVER REVALIDATION

- **Document expiration** date
  - The cache doesn't revalidate with the server for every request
  - Save of bandwidth, time and reduction of the traffic
- Server revalidation is made with **conditional methods**.
- **Conditional GET**
  - Ask the server to send back an object body only if the document is different than in the cache
  - Otherwise, server responses with a small `304 Not Modified` message without body
- Freshness check and the object fetch are combined into a single request by adding special **conditional headers**

SERVER REVALIDATION CONDITIONAL HEADERS

- HTTP defines **five** conditional request headers; **two** of them are commonly used for cache revalidation.

SERVER REVALIDATION CONDITIONAL HEADERS

- HTTP defines **five** conditional request headers; **two** of them are commonly used for cache revalidation.

- `If-Modified-Since` — performs the requested method if the document has been modified since the specified date. This is used in conjunction with the `Last-Modified` server response header.

-

SERVER REVALIDATION CONDITIONAL HEADERS

- HTTP defines **five** conditional request headers; **two** of them are commonly used for cache revalidation.

- `If-Modified-Since` — performs the requested method if the document has been modified since the specified date. This is used in conjunction with the `Last-Modified` server response header.

- `If-None-Match` — the server may provide special tags (`ETag`) on the document that act like serial numbers. The `If-None-Match` header performs the requested method if the cached tag differs from the tag in the server's document.

SERVER REVALIDATION
ENTITY TAG REVALIDATION

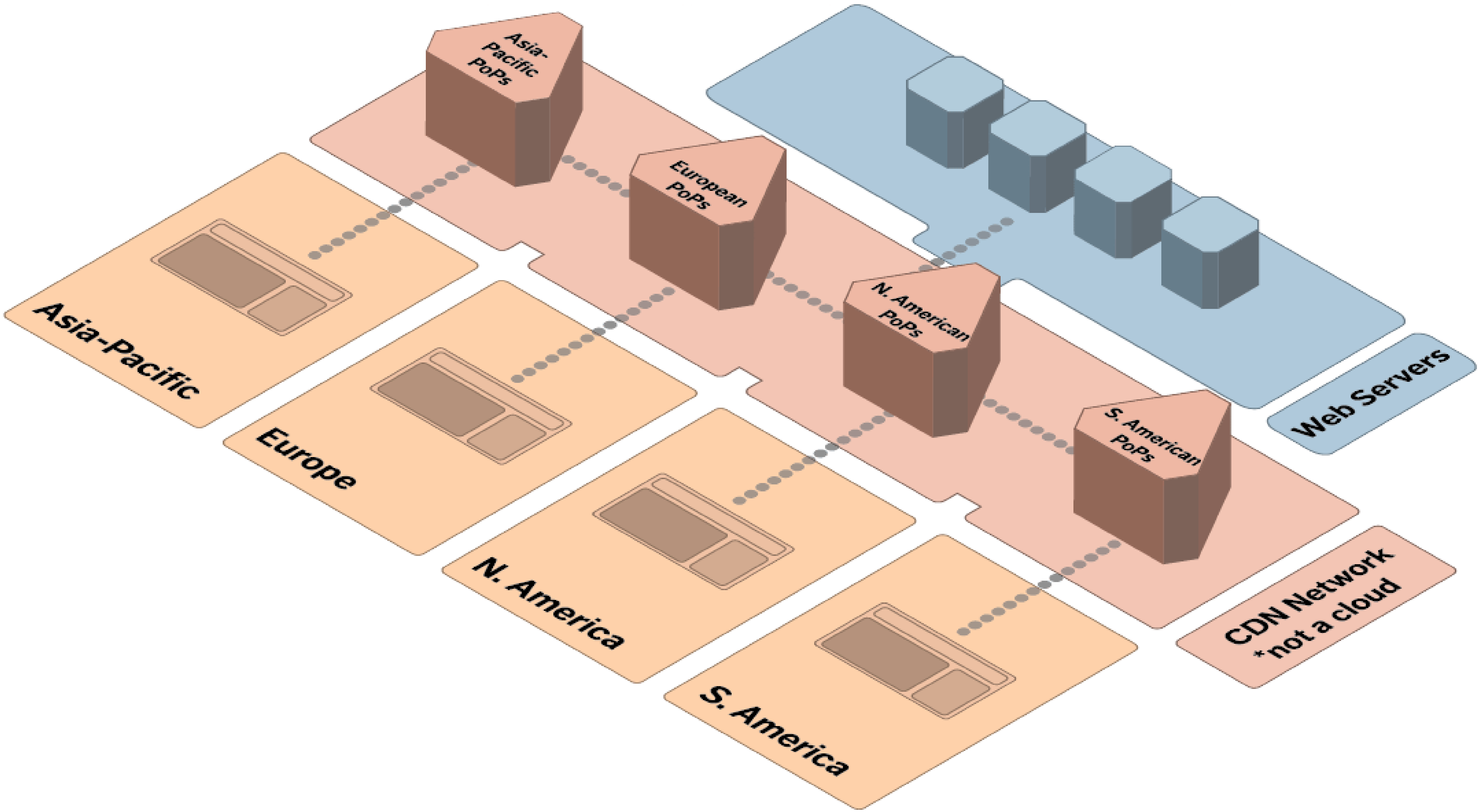
- **Date-based revalidation** is the most common technique, but there are situations when it isn't adequate:
  - documents rewritten **periodically** but containing the same data.
  - servers cannot **accurately** determine modification dates.
  - one-second **granularity** of modification dates is not enough.

SERVER REVALIDATION
ENTITY TAG REVALIDATION

- **Date-based revalidation** is the most common technique, but there are situations when it isn’t adequate:
  - documents rewritten **periodically** but containing the same data.
  - servers cannot **accurately** determine modification dates.
  - one-second **granularity** of modification dates is not enough.
- HTTP allows you to compare **document version identifiers** called entity tags (ETags).
- Entity tags are arbitrary labels attached to the document which might contain a **serial number**, a **checksum** or other **fingerprint** of the document content.

CONTENT DELIVERY NETWORKS

- Content delivery network (CDN) — a large, **geographically distributed** network of specialized servers that accelerate the delivery of web content to internet-connected devices.
- The primary technique that a CDN uses to speed the delivery of web content to end users is **edge caching**.
- **Edge caching** entails storing replicas of static text, image, audio, and video content in multiple servers around the "edges" of the internet, so that user requests can be served by a nearby edge server rather than by a far-off origin server.



Asia-Pacific PoPs
European PoPs
N. American PoPs
S. American PoPs
Web Servers
CDN Network
*not a cloud
Asia-Pacific
Europe
N. America
S. America

CONTROLLING CACHABILITY

- `Cache-Control` header has a few different values to constrain how clients should be caching the response.
- `public` — public proxy servers can cache the response
- `private` — only the browser can cache the response
- `no-cache` — one must not cache the response, or one must revalidate cached response with use of other criteria
- `no-store` — one must not cache the response

CONTROLLING CACHABILITY

- `Cache-Control: max-age` — sets a **relative** expiration time (in seconds) from the time the response is generated.
- `Cache-Control: s-maxage` — acts like `max-age` but applies only to shared (public) caches.

CONTROLLING CACHABILITY

- `Cache-Control: max-age` — sets a **relative** expiration time (in seconds) from the time the response is generated.

- `Cache-Control: s-maxage` — acts like `max-age` but applies only to shared (public) caches.

- If the server does not send expiration date, the client can use its own **heuristic expiration algorithm** to determine freshness:

```
time_since_modify = max(0, fetch_time - server_last_modified);
new_expiration_time = time_since_modify * lm_factor;
```

CONTROLLING CACHABILITY

- `Cache-Control: max-age` — sets a **relative** expiration time (in seconds) from the time the response is generated.

- `Cache-Control: s-maxage` — acts like `max-age` but applies only to shared (public) caches.

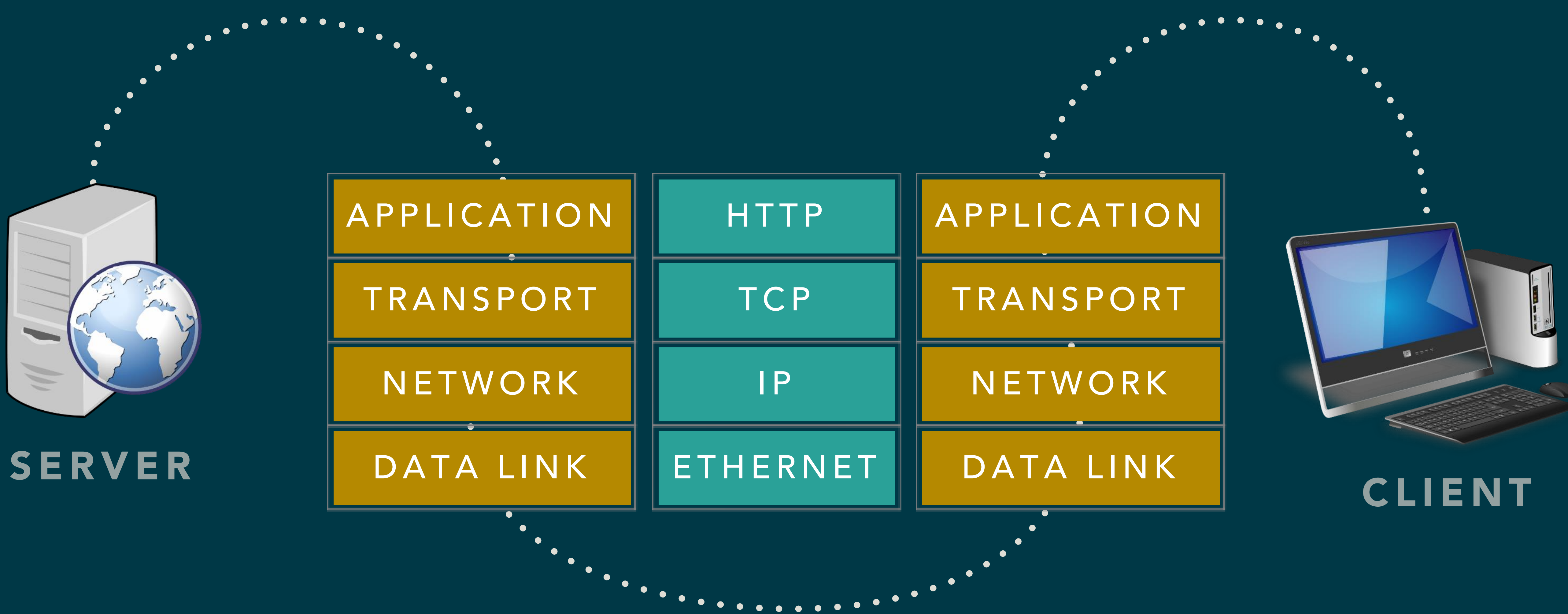
- If the server does not send expiration date, the client can use its own **heuristic expiration algorithm** to determine freshness:

```
time_since_modify = max(0, fetch_time - server_last_modified);
new_expiration_time = time_since_modify * lm_factor;
```

- `Cache-Control: must-revalidate` — tells caches they cannot serve a **stale** copy of this object without first revalidating with the origin server. Caches are still free to serve fresh copies.

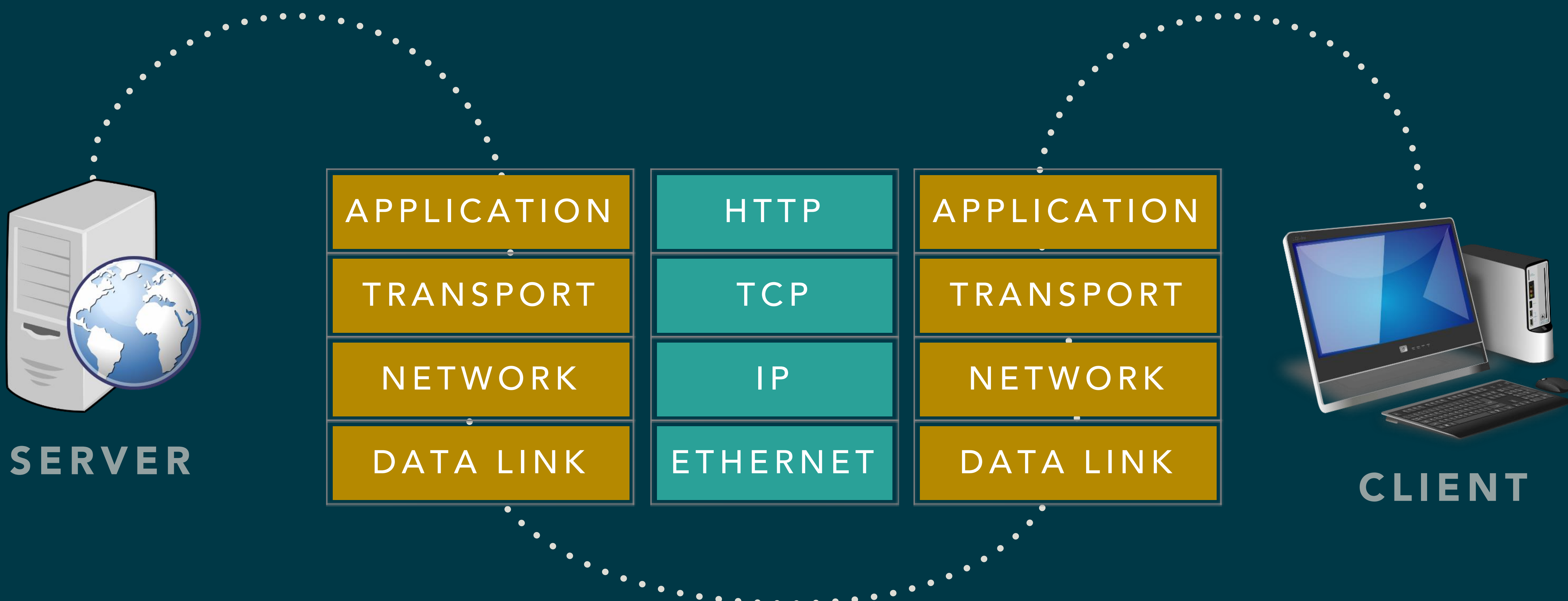
HTTPS — SECURE CONNECTIONS

- An additional security layer in the network protocol stack, between HTTP and TCP — the Secure Sockets Layer (**SSL**) or the improved Transport Layer Security (**TLS**).



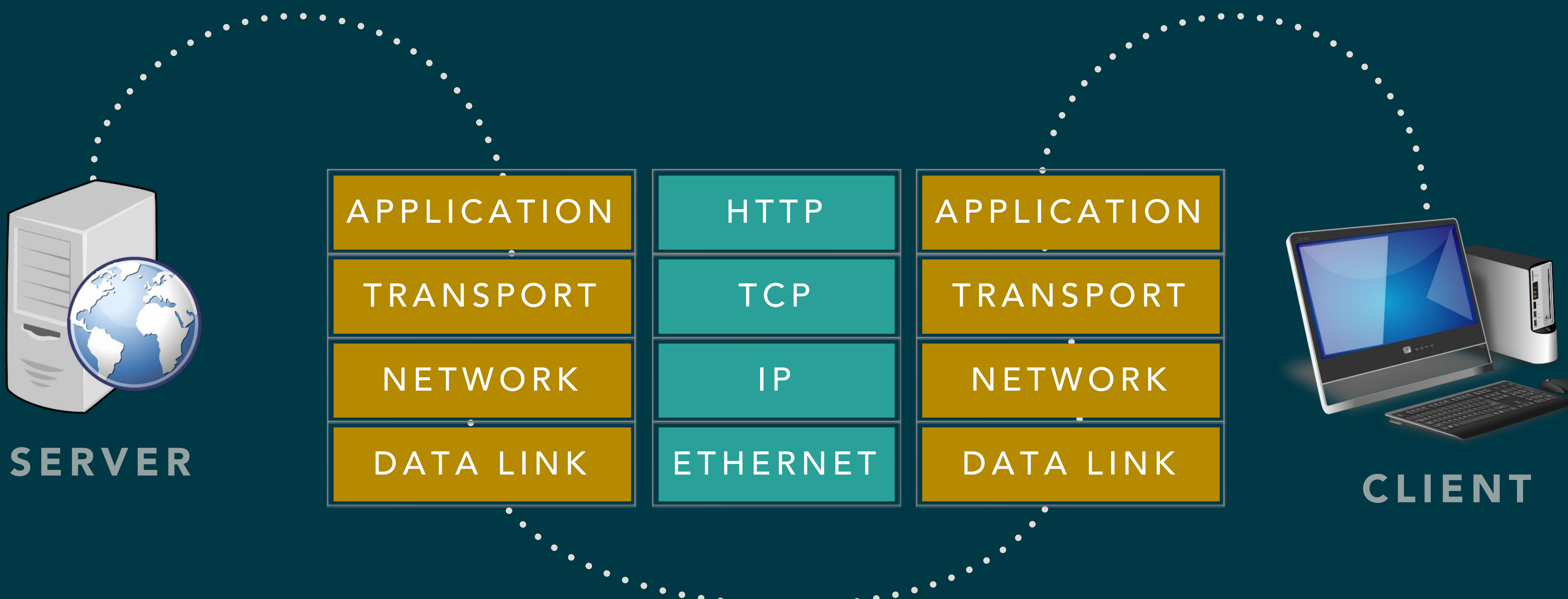
HTTPS — SECURE CONNECTIONS

- An additional security layer in the network protocol stack, between HTTP and TCP — the Secure Sockets Layer (**SSL**) or the improved Transport Layer Security (**TLS**).



HTTPS — SECURE CONNECTIONS

- An additional security layer in the network protocol stack, between HTTP and TCP — the Secure Sockets Layer (**SSL**) or the improved Transport Layer Security (**TLS**).



TRANSPORT LAYER SECURITY

- The TLS protocol provides three essential services that form a foundation of secure communication:
  - **encryption** — using public-key cryptography allows the peers to negotiate a shared secret key (within a TLS handshake).
  - **authentication** — to verify the identity of the server/client;
  - **integrity** — to detect message tampering and forgery.

TRANSPORT LAYER SECURITY

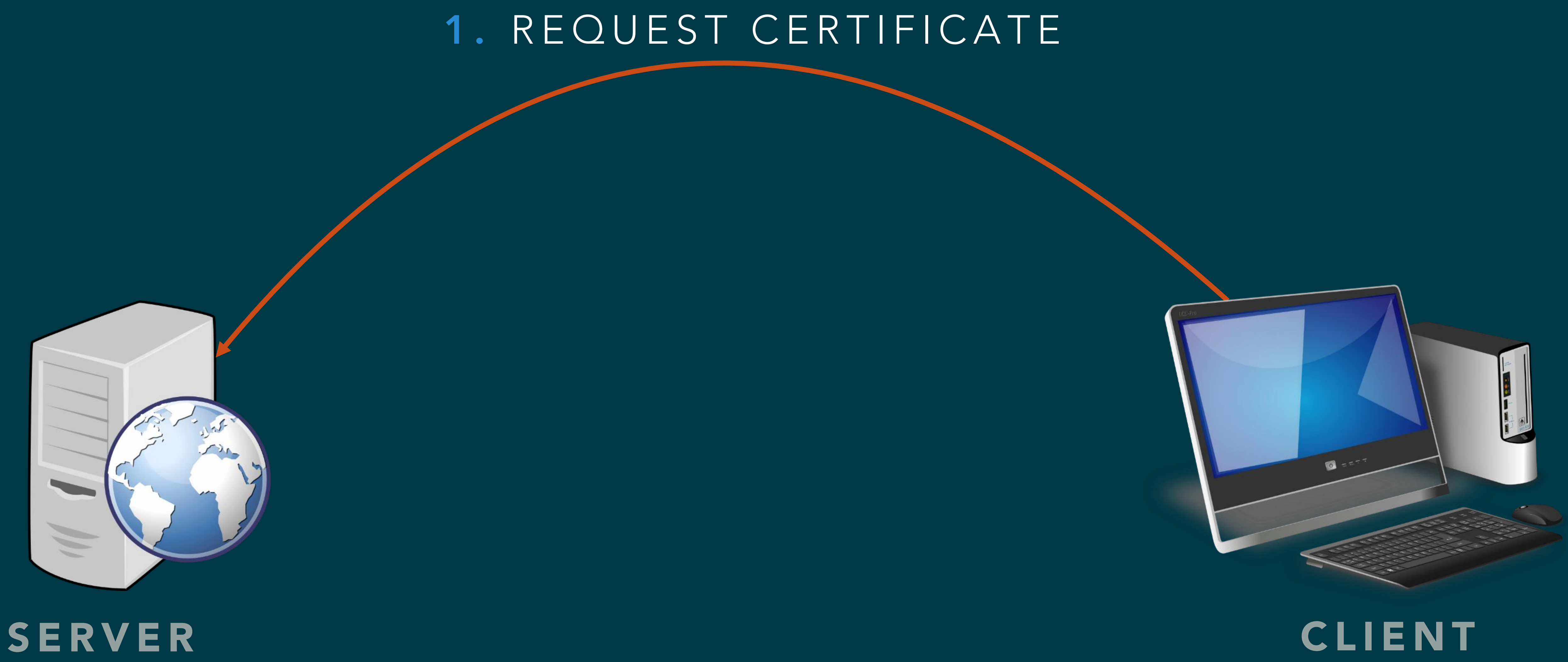
- The TLS protocol provides three essential services that form a foundation of secure communication:
  - **encryption** — using public-key cryptography allows the peers to negotiate a shared secret key (within a TLS handshake).
  - **authentication** — to verify the identity of the server/client;
  - **integrity** — to detect message tampering and forgery.
- HTTPS encrypts all request and response traffic, including the HTTP **headers** and message **body**, and everything after the host name in the **URL**.

TLS HANDSHAKE

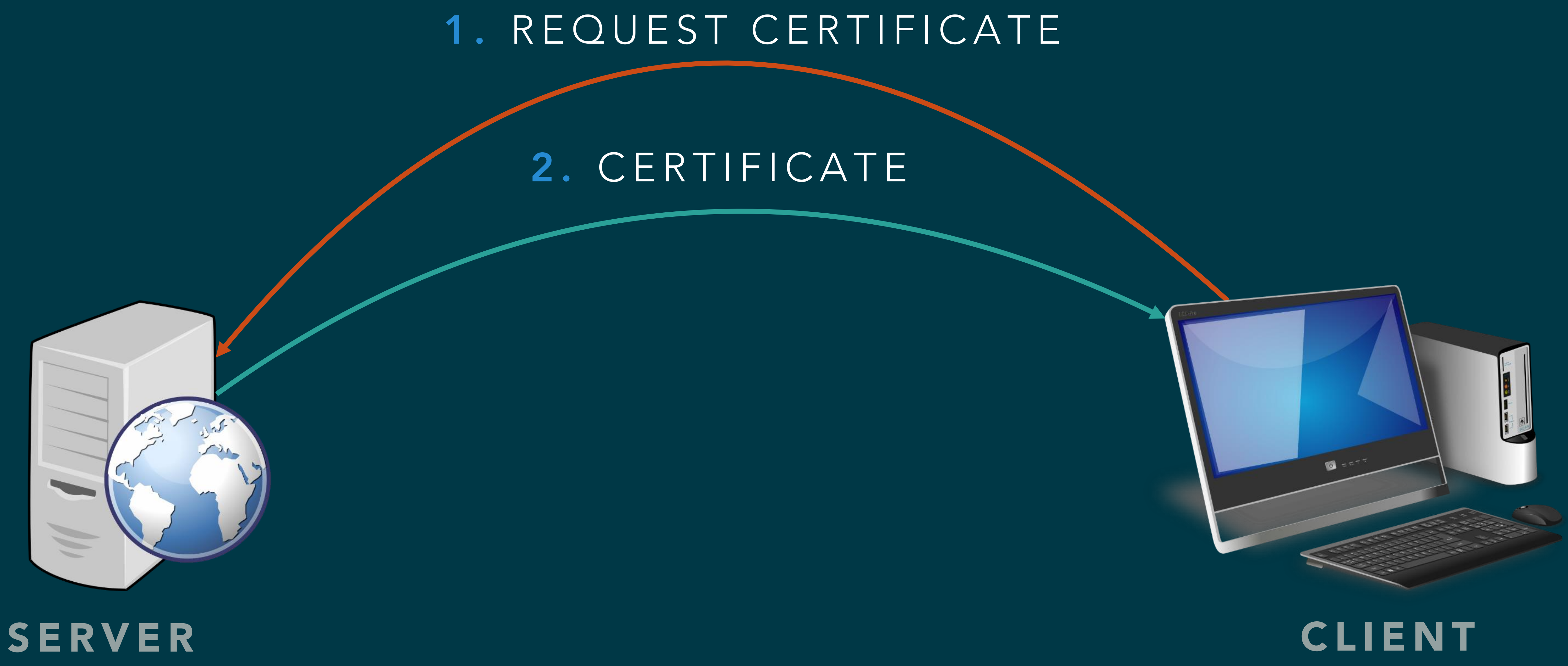
SERVER

CLIENT

TLS HANDSHAKE

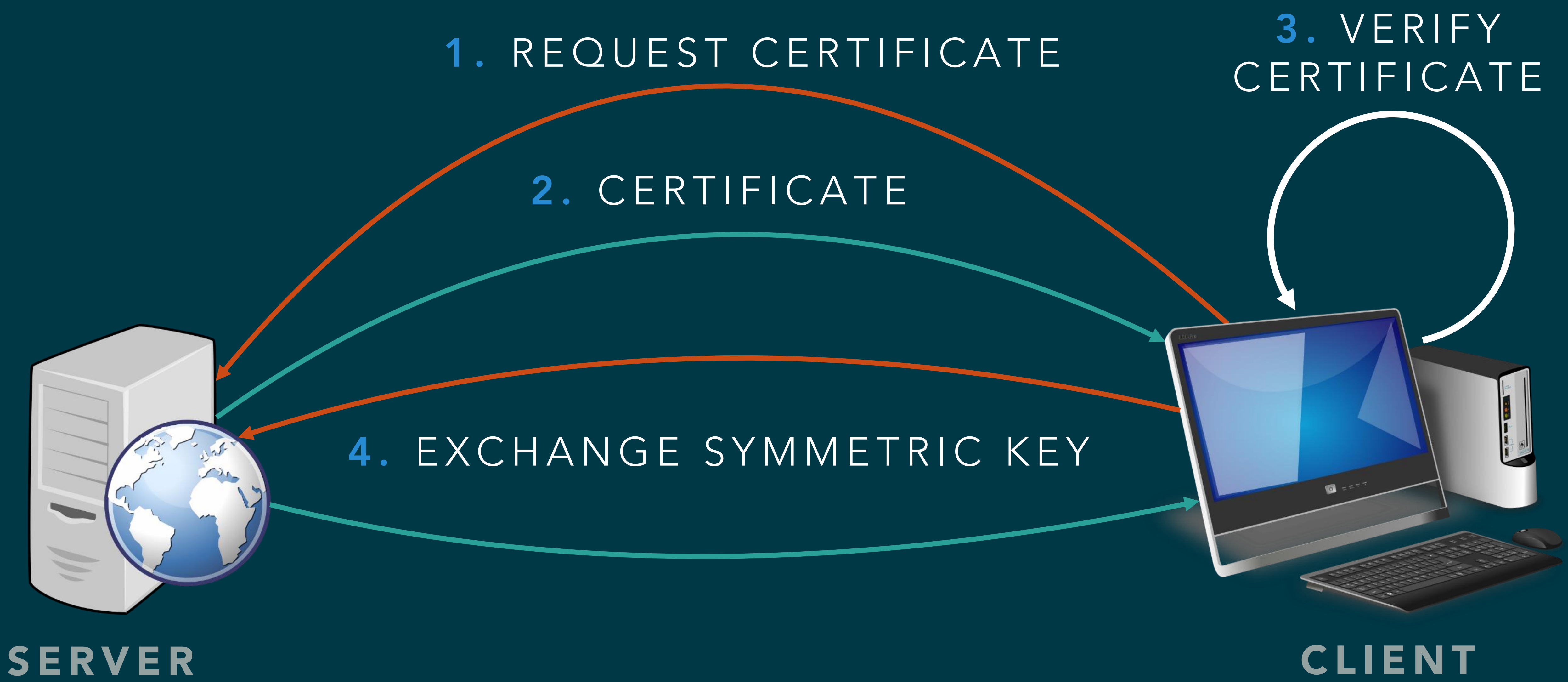


TLS HANDSHAKE

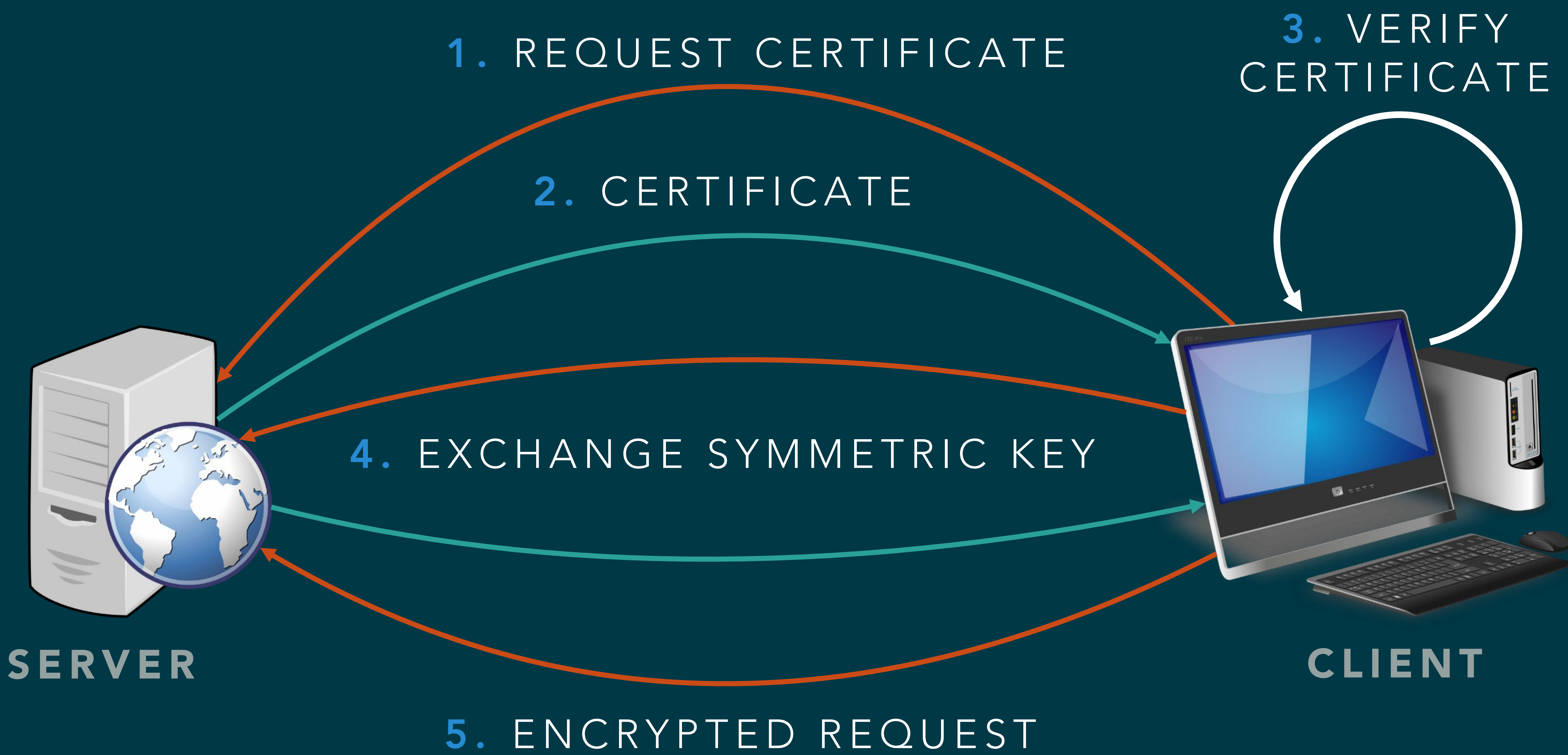


TLS HANDSHAKE

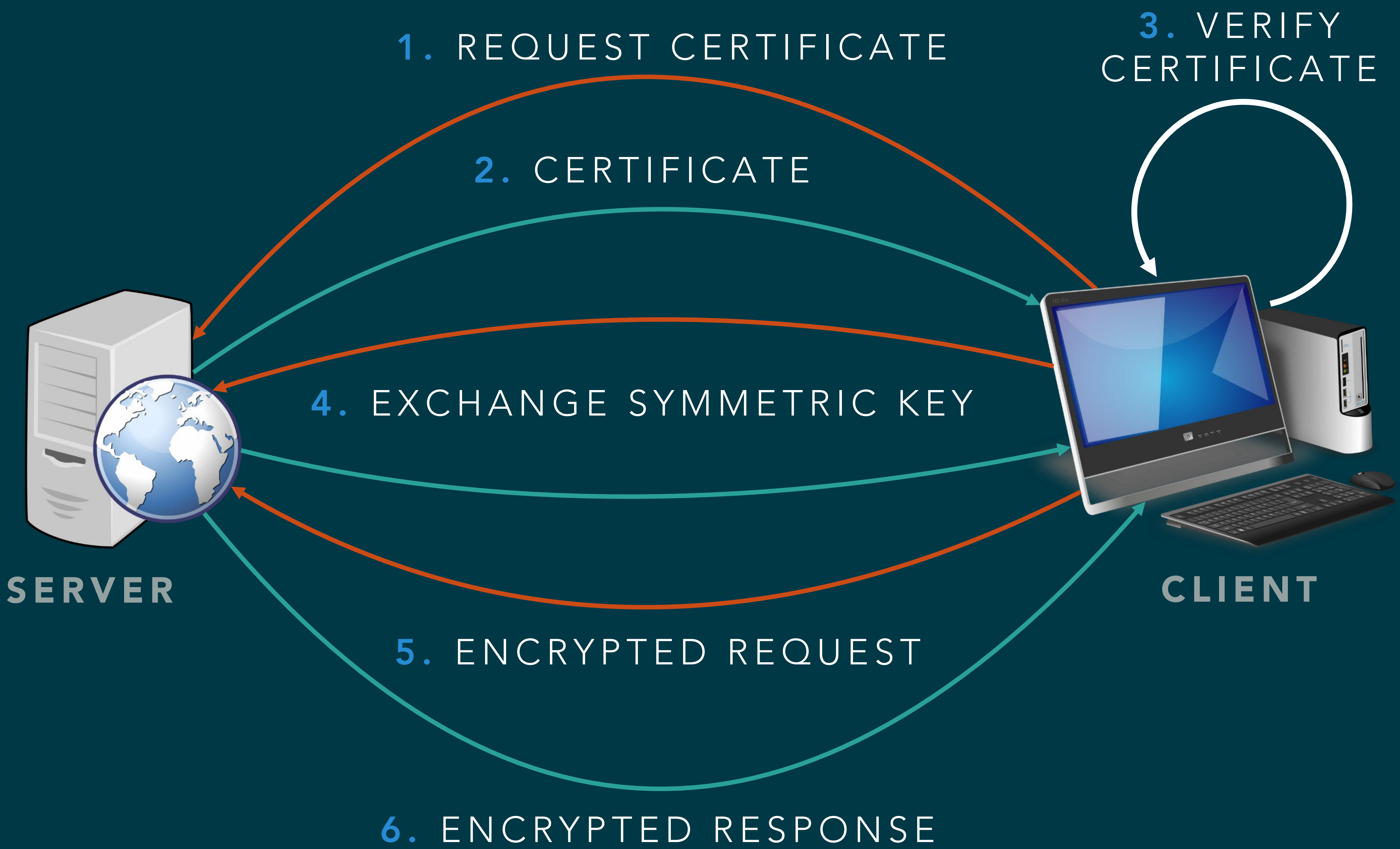
TLS HANDSHAKE



TLS HANDSHAKE



TLS HANDSHAKE



HTTP/2

- HTTP history:
  - 1991: HTTP 0.9
  - 1996: HTTP 1.0 (RFC 1945)
  - 1997: HTTP 1.1 (RFC 2068)
  - 1999: HTTP 1.1 improved (RFC 2616)
  - 05.2015 — HTTP/2 (RFC 7540)
- HTTP/2 maintains high-level compatibility with HTTP/1.1 (methods, status codes, header fields)
- Based on **SPDY** — developed by Google since 2009.

SPeeDY — MOTIVATION

- The Web has changed — http://httparchive.org/

SPeeDY — MOTIVATION

- The Web has changed — http://httparchive.org/
- HTTP was not designed for optimal performance:
  - **single** request per connection (until HTTP/1.1)
  - exclusively **client-initiated** requests
  - **uncompressed** and **redundant** headers
  - **optional** data compression

SPeeDY — MOTIVATION

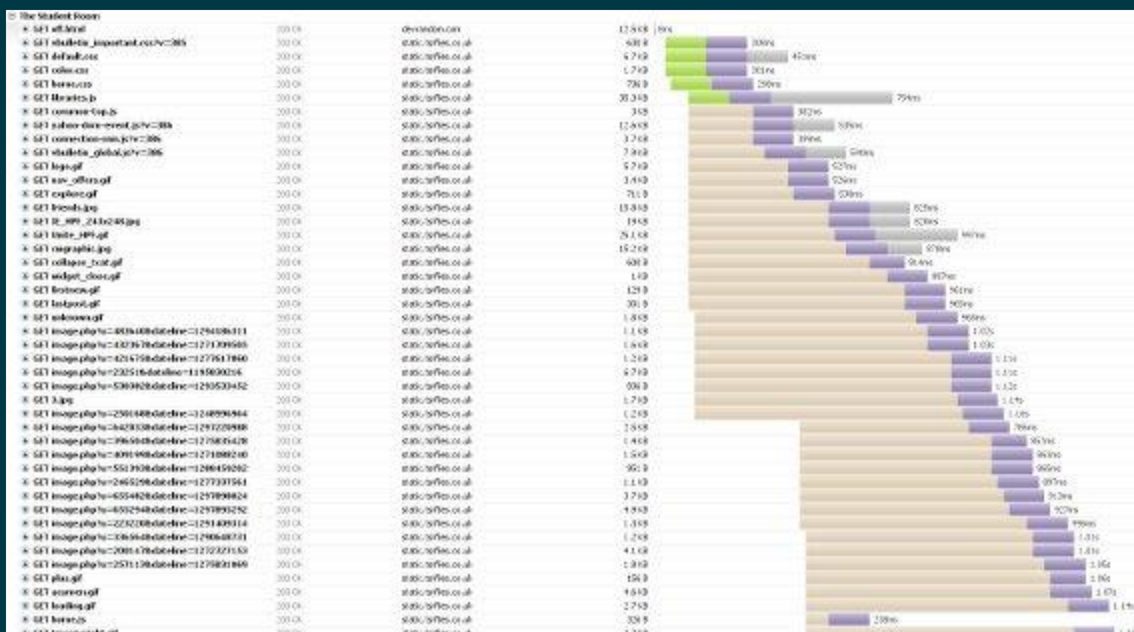
- The Web has changed — http://httparchive.org/
- HTTP was not designed for optimal performance:
  - **single** request per connection (until HTTP/1.1)
  - exclusively **client-initiated** requests
  - **uncompressed** and **redundant** headers
  - **optional** data compression
- Browsers and applications employ a number of **tricks** to improve the performance of the HTTP protocol.

PARALLEL CONNECTIONS

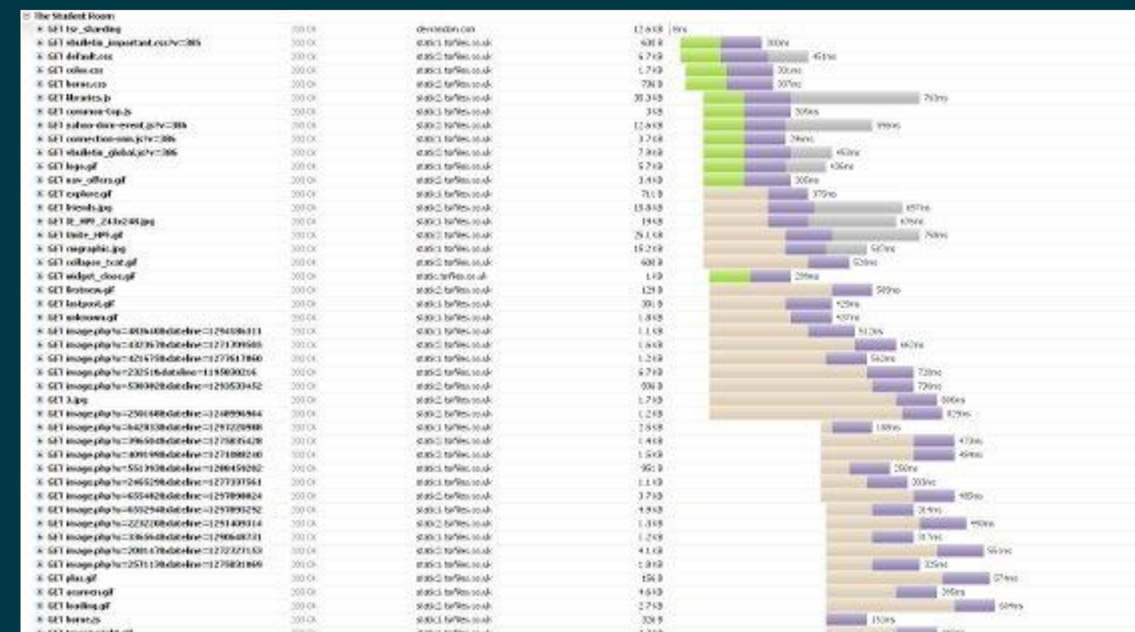
- **Parallel connections** — a technique employed by browsers to minimize network delays and improve overall performance.
- A **pool** of parallel connections allows the client to download the assets simultaneously rather than in a **serial** fashion.
- *According to **HTTP 1.1**: A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy.*
- Most browser use a set of heuristics to decide on how many parallel connections to establish (typically from 4 to 8).

DOMAIN SHARDING

No sharding



Sharding

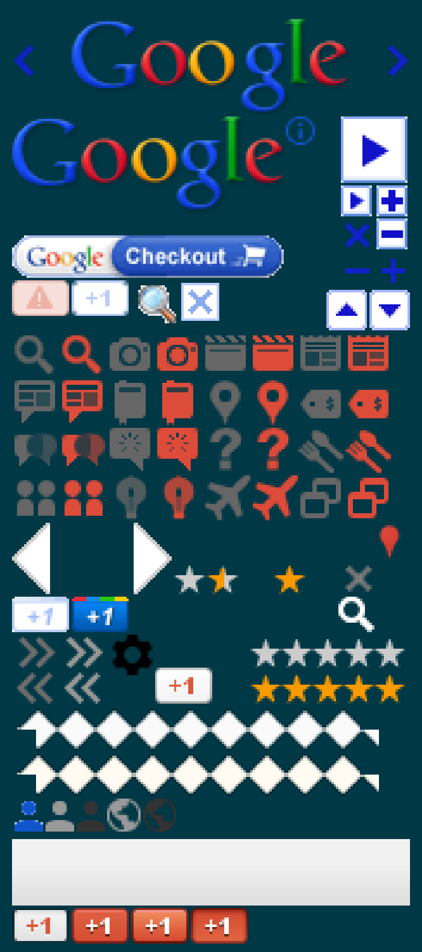


- **Domain sharding** — distributing web resources across multiple domains or content delivery networks.
- Domain sharding is often overused and can hurt performance due to additional DNS lookups and TCP slow-start.

Source: https://gtmetrix.com/parallelize-downloads-across-hostnames.html

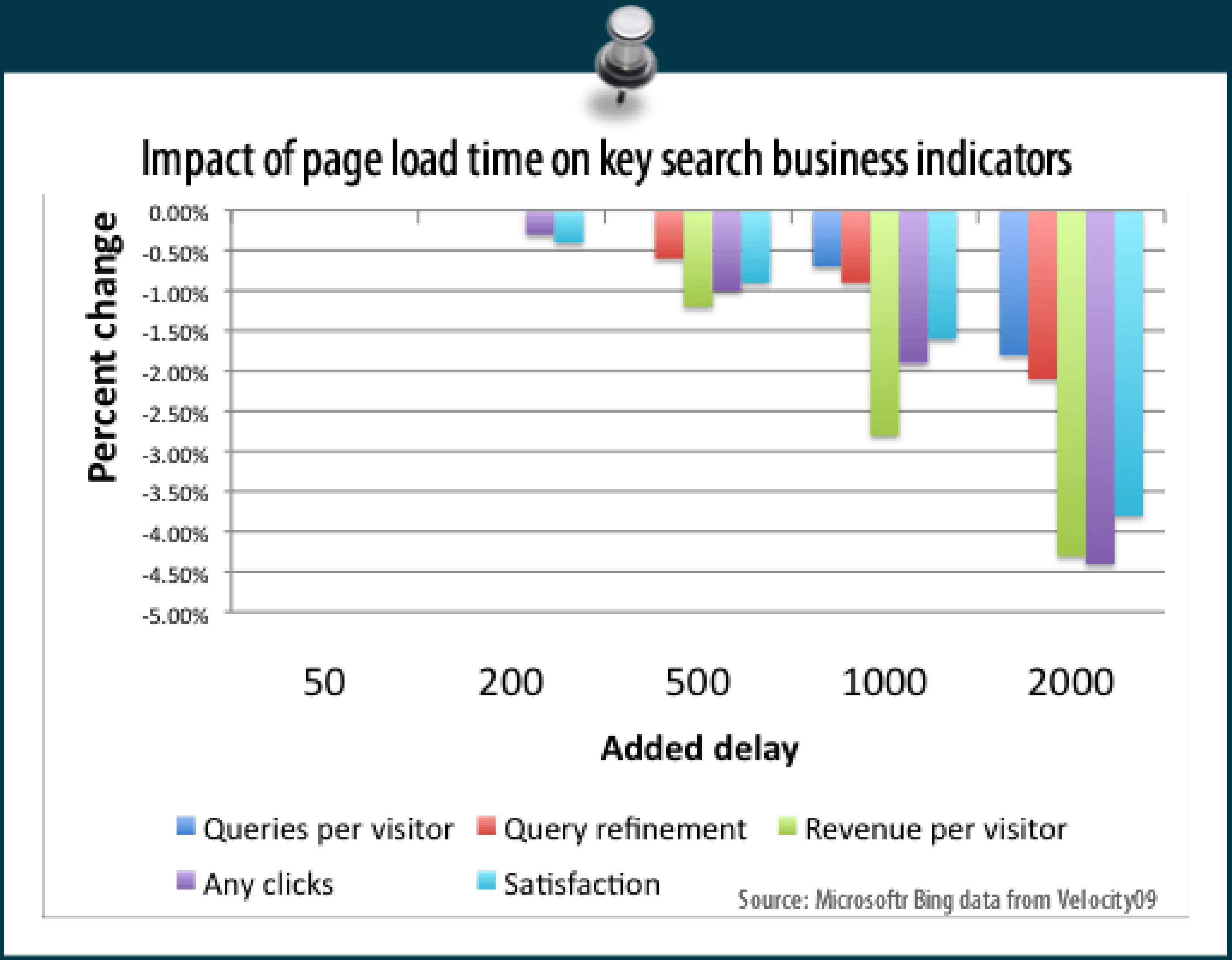
RESOURCE CONCATENATION, SPRITING AND INLINING

- *The fastest request is a request not made.*
- **Concatenation** — multiple JavaScript or CSS files are combined into a single resource.
- **Spriting** — multiple images are combined into a larger, composite image.
- **Resource Inlining**
    - JavaScript and CSS can be included in HTML via the appropriate tags
    - Binary data (e.g., images) can be included in HTML/CSS using data-URI



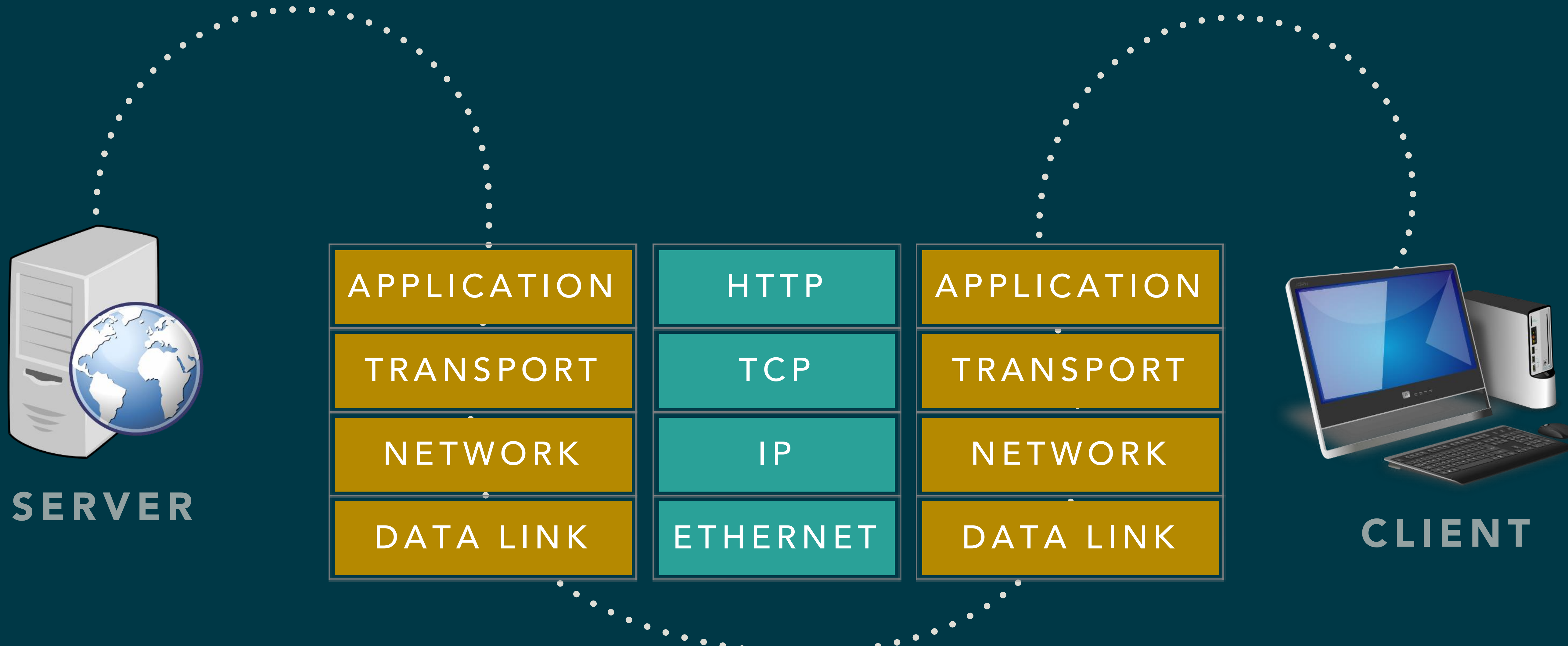
SPeeDY — BUSINESS MOTIVATION

- **SPDY** — a protocol for transporting content over the web, designed specifically for **end-user perceived latency** (the target was a 50% reduction in page load time).



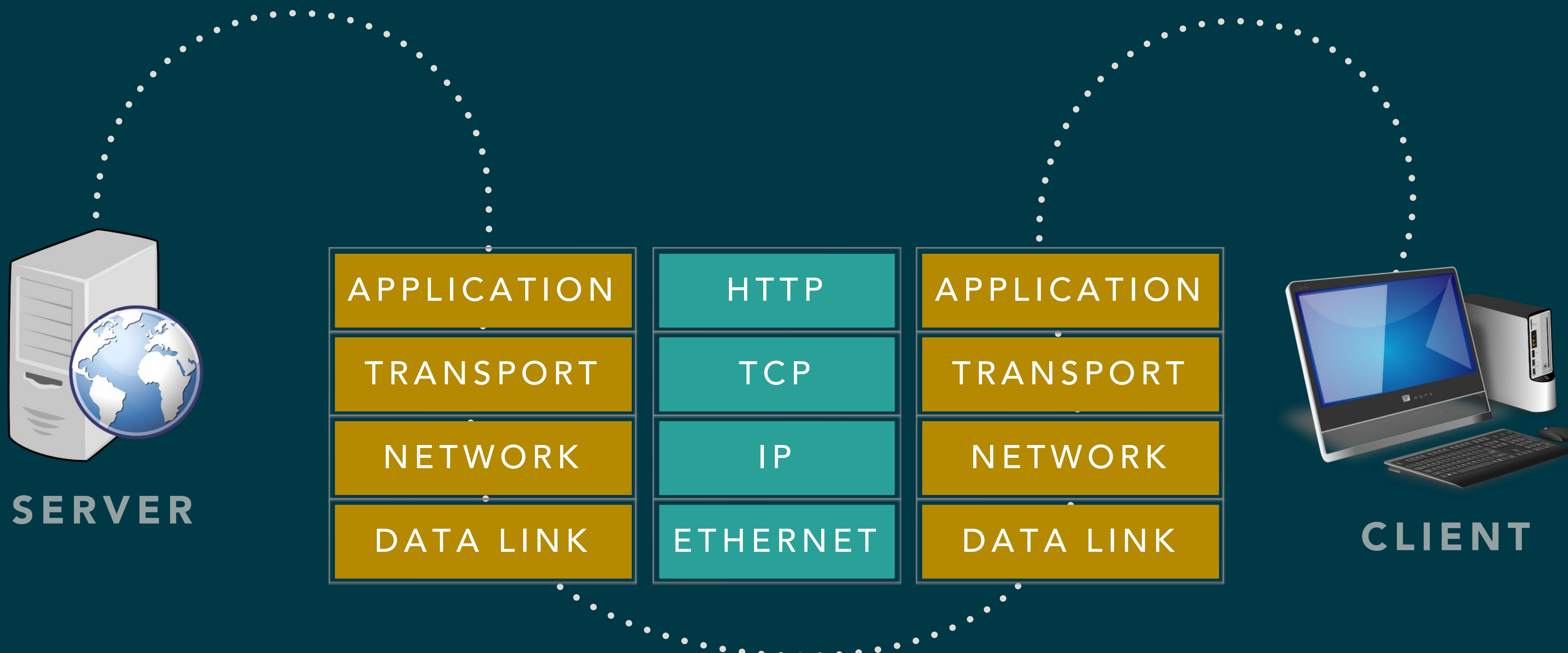
SPeeDY — DESIGN

- SPDY requires **no changes** to existing networking infrastructure.
- SPDY uses **TCP** as the transport layer but **requires** also **SSL/TLS**.



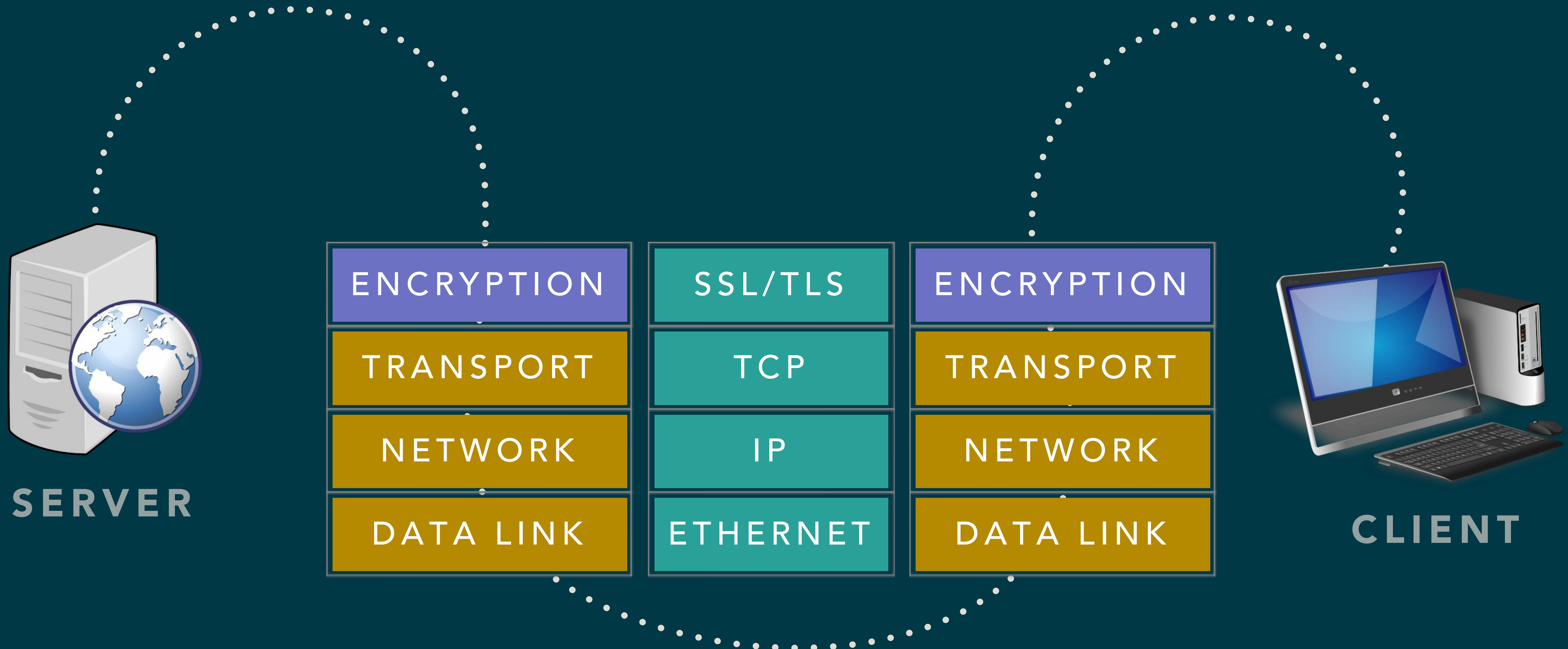
SPeeDY — DESIGN

- SPDY requires **no changes** to existing networking infrastructure.
- SPDY uses **TCP** as the transport layer but **requires** also **SSL/TLS**.



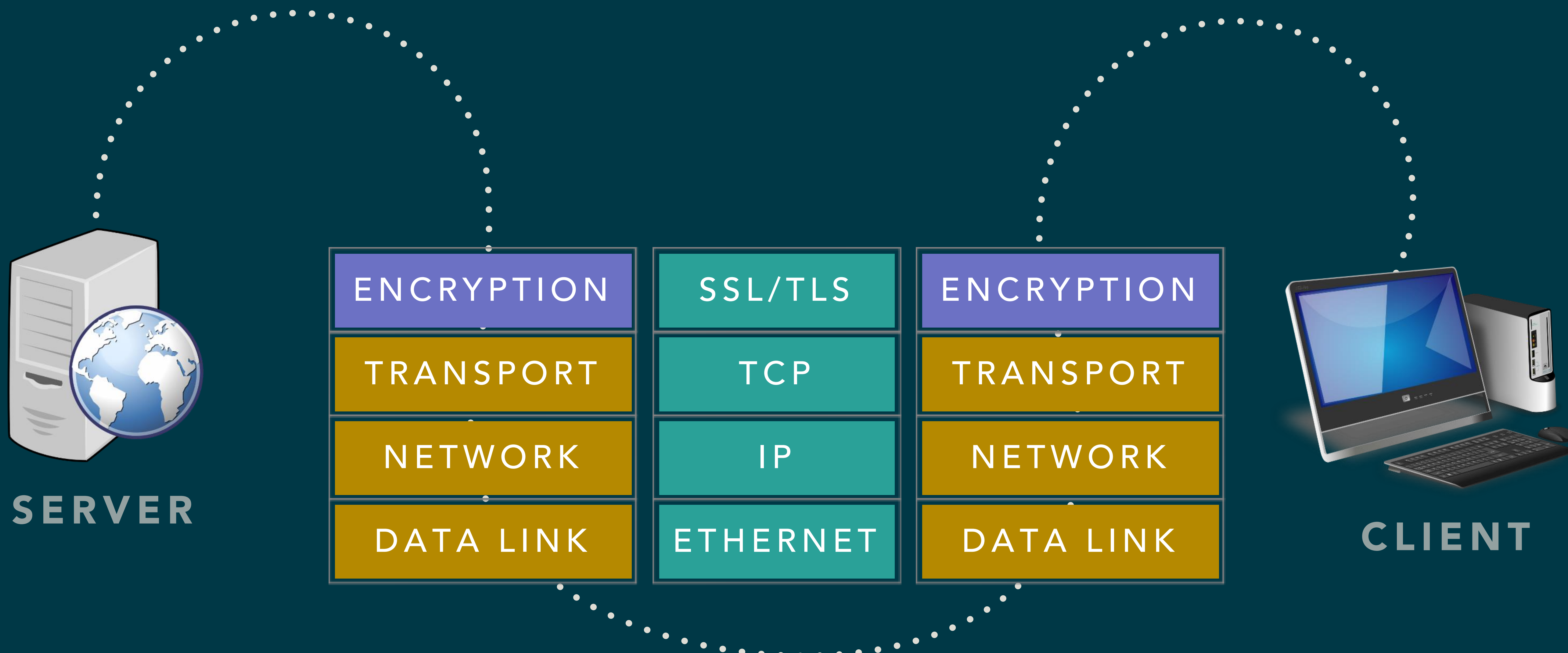
SPeeDY — DESIGN

- SPDY requires **no changes** to existing networking infrastructure.
- SPDY uses **TCP** as the transport layer but **requires** also **SSL/TLS**.



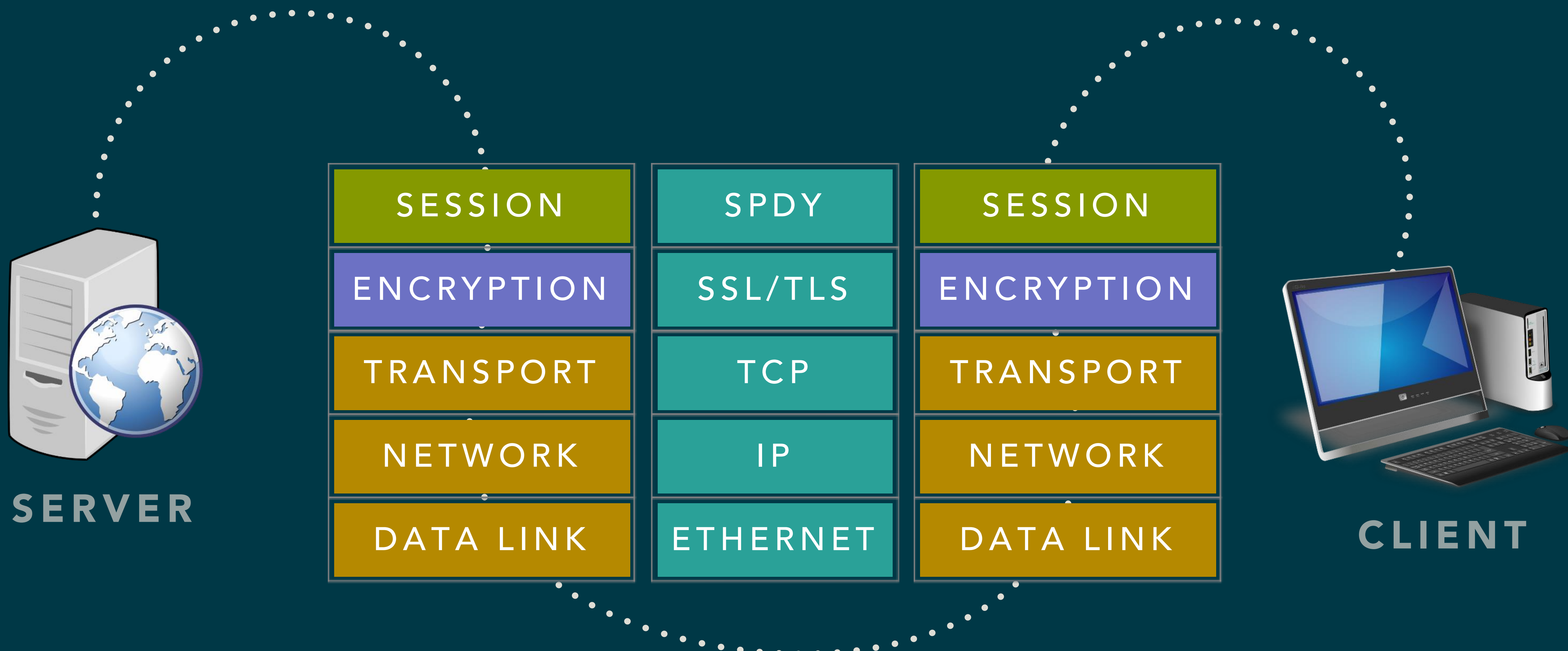
SPEEDY — DESIGN

- SPDY requires **no changes** to existing networking infrastructure.
- SPDY uses **TCP** as the transport layer but **requires** also **SSL/TLS**.



SPEEDY — DESIGN

- SPDY requires **no changes** to existing networking infrastructure.
- SPDY uses **TCP** as the transport layer but **requires** also **SSL/TLS**.



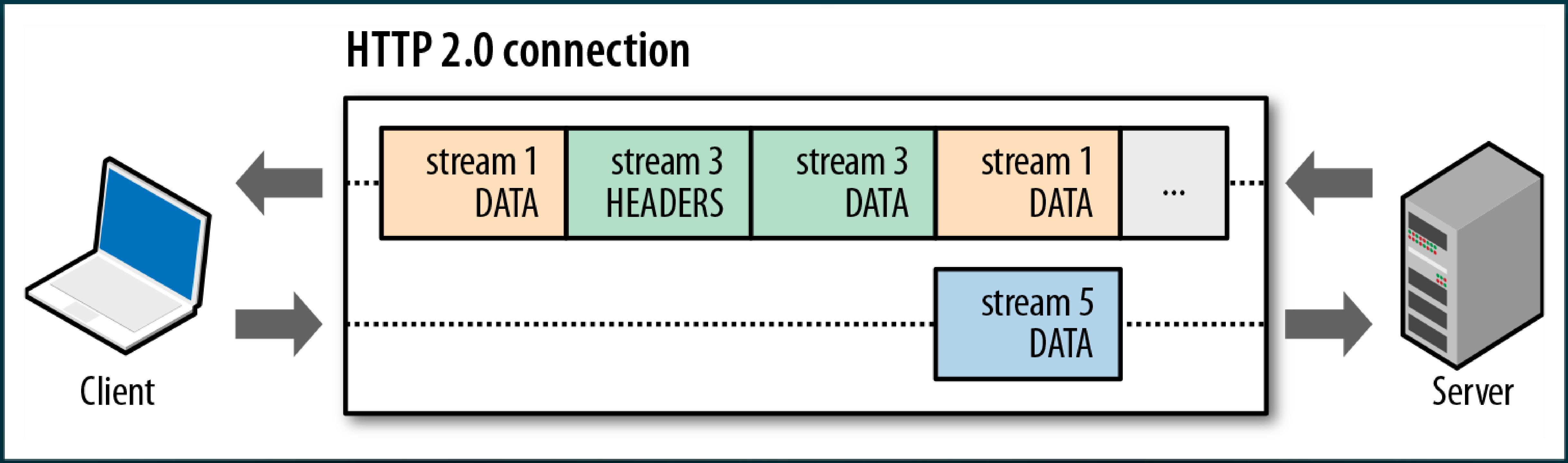
SPeeDY — FEATURES

- **Multiplexed streams** — SPDY allows for unlimited concurrent streams over a single TCP connection.

- The ability to divide HTTP messages into independent **binary frames**, interleave them, and reassemble them on the other end is the **most important enhancement** of SPDY and HTTP/2.

SPeeDY — FEATURES

- **Multiplexed streams** — SPDY allows for unlimited concurrent streams over a single TCP connection.
- The ability to divide HTTP messages into independent **binary frames**, interleave them, and reassemble them on the other end is the **most important enhancement** of SPDY and HTTP/2.



SPeeDY — FEATURES

- **Request prioritization** — the client can request many items from the server and assign a **priority** to each request.
- **HTTP header compression** — SPDY compresses request and response HTTP headers
- **Server-initiated streams** — allows to deliver content to the client without the client needing to ask for it:
  - **Server push** — server can push data to clients via `X-Associated-Content` header.
  - **Server hint** — server uses `X-Subresources` header to suggest to the client that it should ask for specific resources.

FROM SPEEDY TO HTTP/2

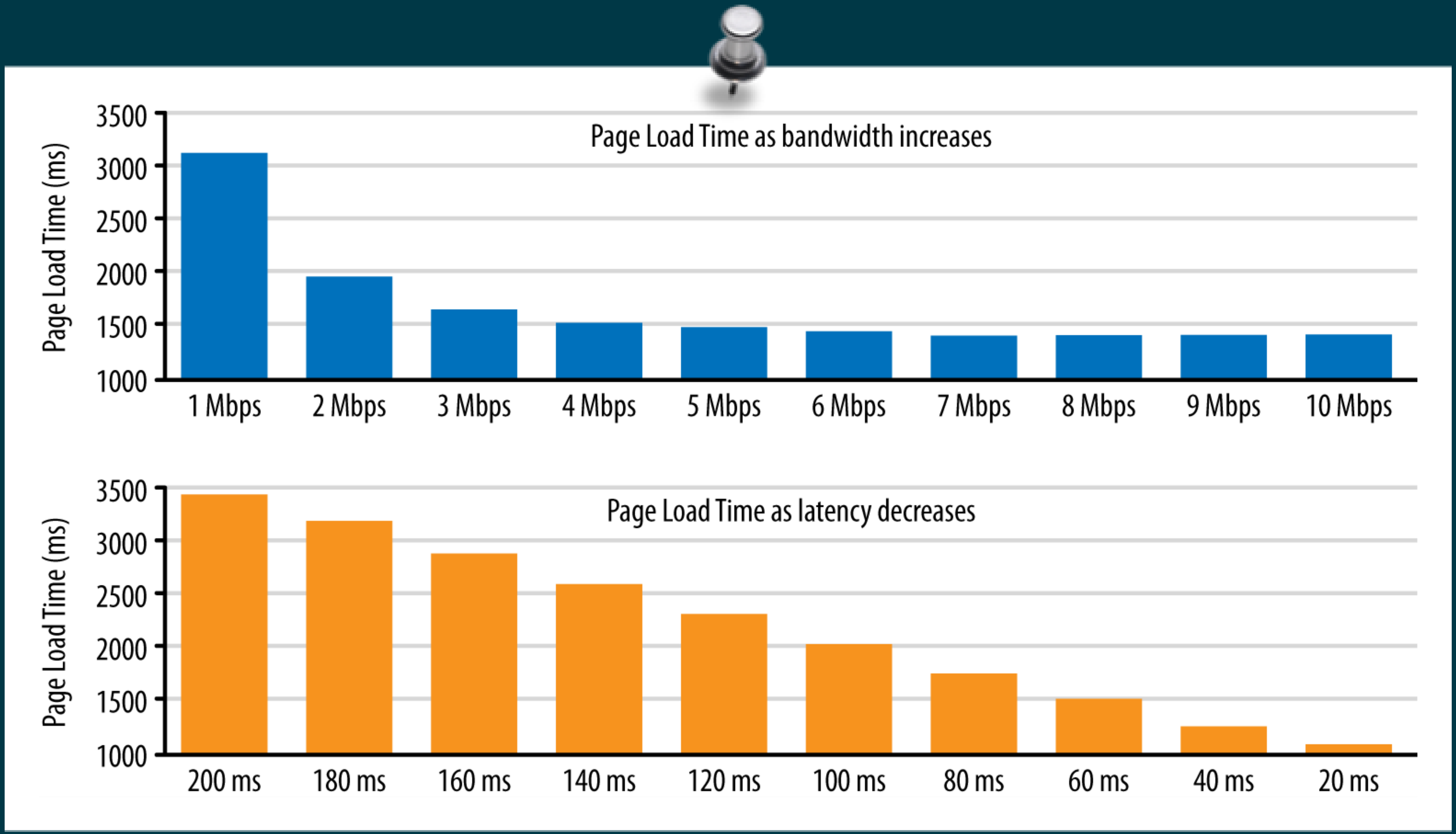
- *Chrome has supported SPDY since Chrome 6, but since most of the benefits are present in HTTP/2, it's time to say goodbye. We plan to remove support for SPDY in early 2016.* HTTP://BLOG.CHROMIUM.ORG/2015/02/HELLO-HTTP2-GOODBYE-SPDY.HTML
- Features inherited by **HTTP/2** from **SPDY**:
  - **multiplexed** streams — can use one connection for parallelism
  - priorities and **dependencies** — one stream can depended on another
  - header **compression** — uses **HPACK** algorithm to reduce overhead
  - allows servers to "**push**" responses proactively into client caches
  - is **binary**, instead of textual HTTP/1.1
- However, in HTTP/2 **encryption** is not mandatory.

HTTP/2 IMPLEMENTATIONS

- Firefox:
    - experimental support for HTTP/2 in version 34
    - enabled by default in version 36
    - only supports HTTP/2 over encrypted connection (TLS)
- Google Chrome:
    - support from version 40
    - enabled by default in 41
    - only supports HTTP/2 over encrypted connection (TLS)
- Microsoft Edge:
    - support from version 12
    - only supports HTTP/2 over encrypted connection (TLS)
- Performance comparison: https://blog.httpwatch.com/2015/01/16/a-simple-performance-comparison-of-https-spdy-and-http2/

CONCLUSIONS

- *More Bandwidth Doesn't Matter (Much).*
- *Latency is a Performance Bottleneck.*

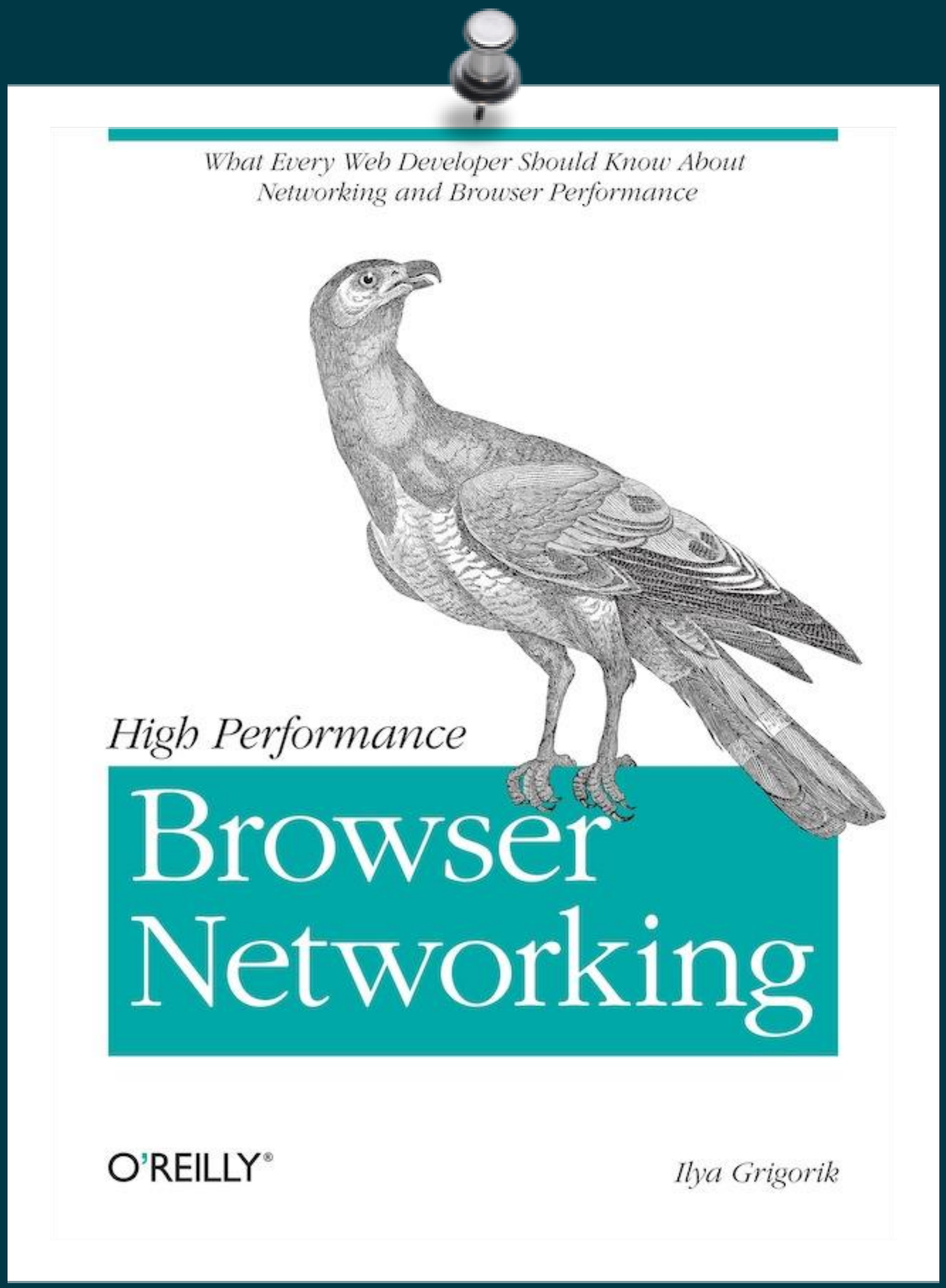


CONCLUSIONS

- HTTP is an essential building block of the Web and a prerequisite for utilizing the full power of Internet technologies.
- HTTP headers allow for more advanced features like caching, authentication or client identification.
- Performance of HTTP 1.x can be improved by using persistent and parallel connections.
- HTTP/2 offers further performance improvements, including multiplexed streams and header compression.

REFERENCES

- *High Performance Browser Networking — Ilya Grigorik, O'Reilly Media, Inc., 2013, available online at: http://chimera.labs.oreilly.com/books/1230000000545*
- *What Every Web Developer Should Know About HTTP — K. Scott Allen, OdeToCode LLC, 2012, available online at: http://odetocode.com/Articles/741.aspx*



REFERENCES

- *http2 explained: background, the protocol, the implementations and the future* — Daniel Stenberg, http://daniel.haxx.se/http2/, 2015
- *HTTP: The Protocol Every Web Developer Must Know* http://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177
- *HTTP: The Definitive Guide* — David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, Sailu Reddy, O'Reilly Media, Inc., 2002