Achieving Keyless CDNs with Conclaves

Stephen Herwig

Christina Garman Dave Levin

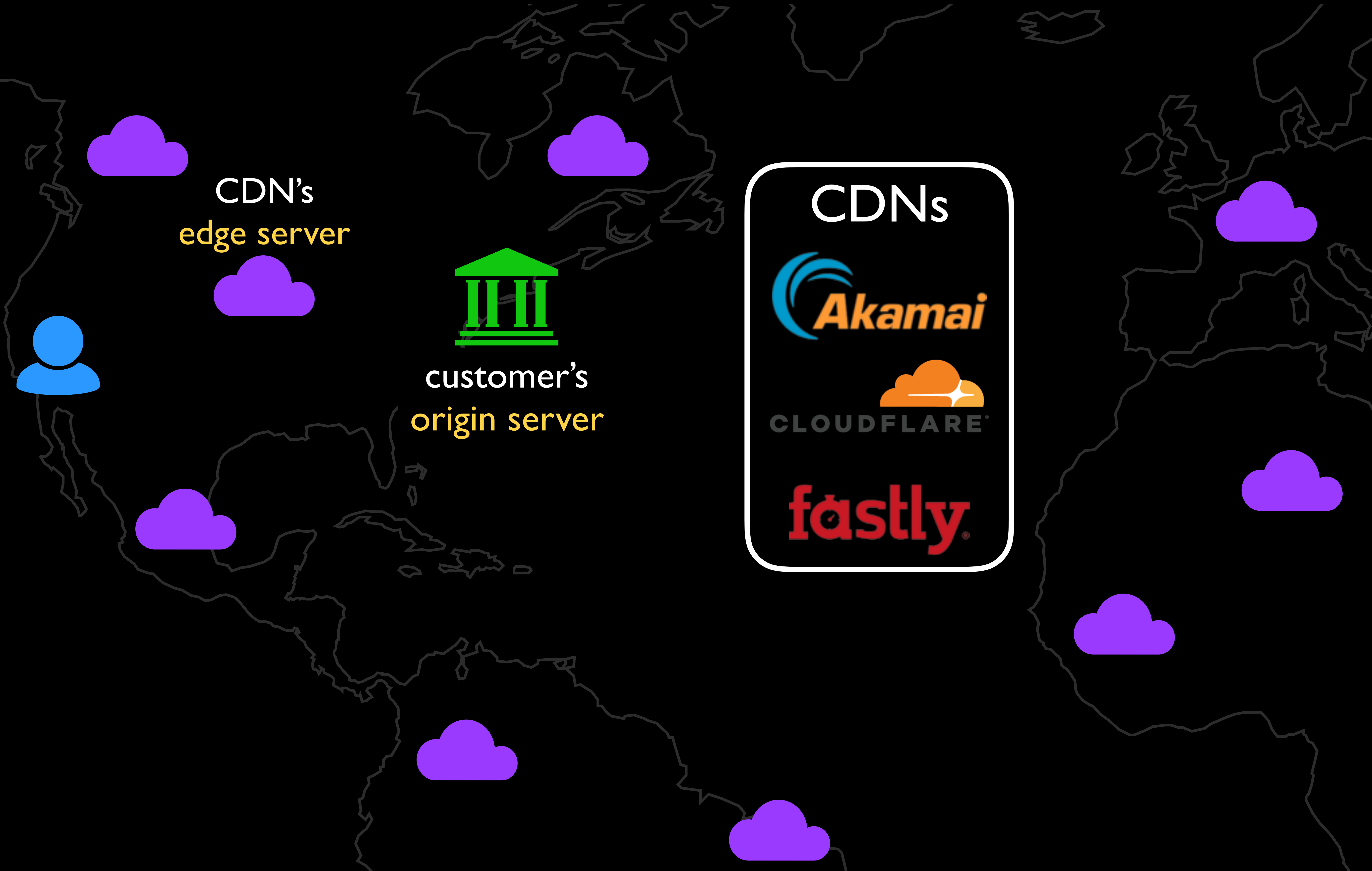
UNIVERSITY OF MARYLAND 1856

PURDUE UNIVERSITY

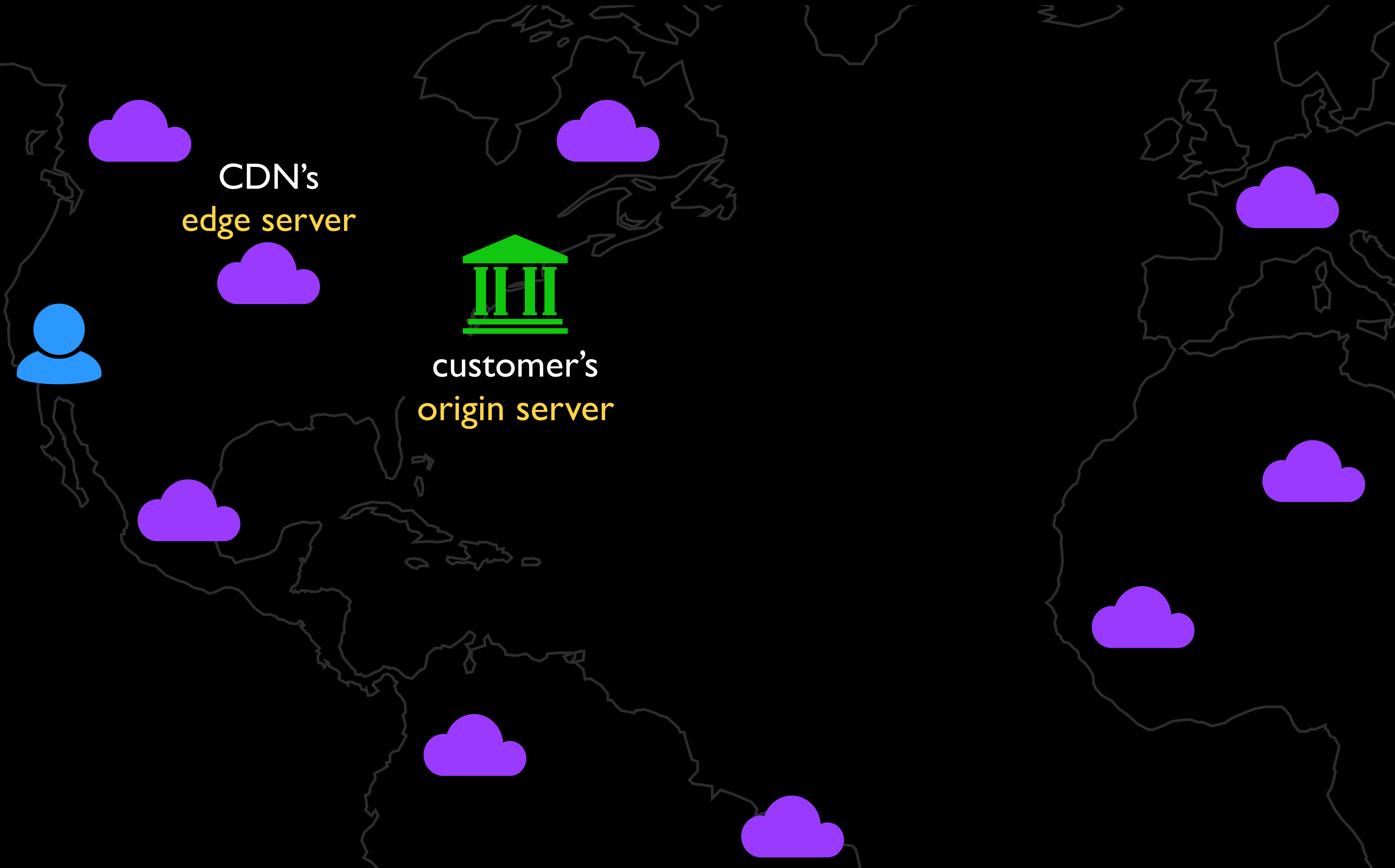

User
Bank

Content Delivery Networks host their customers' websites

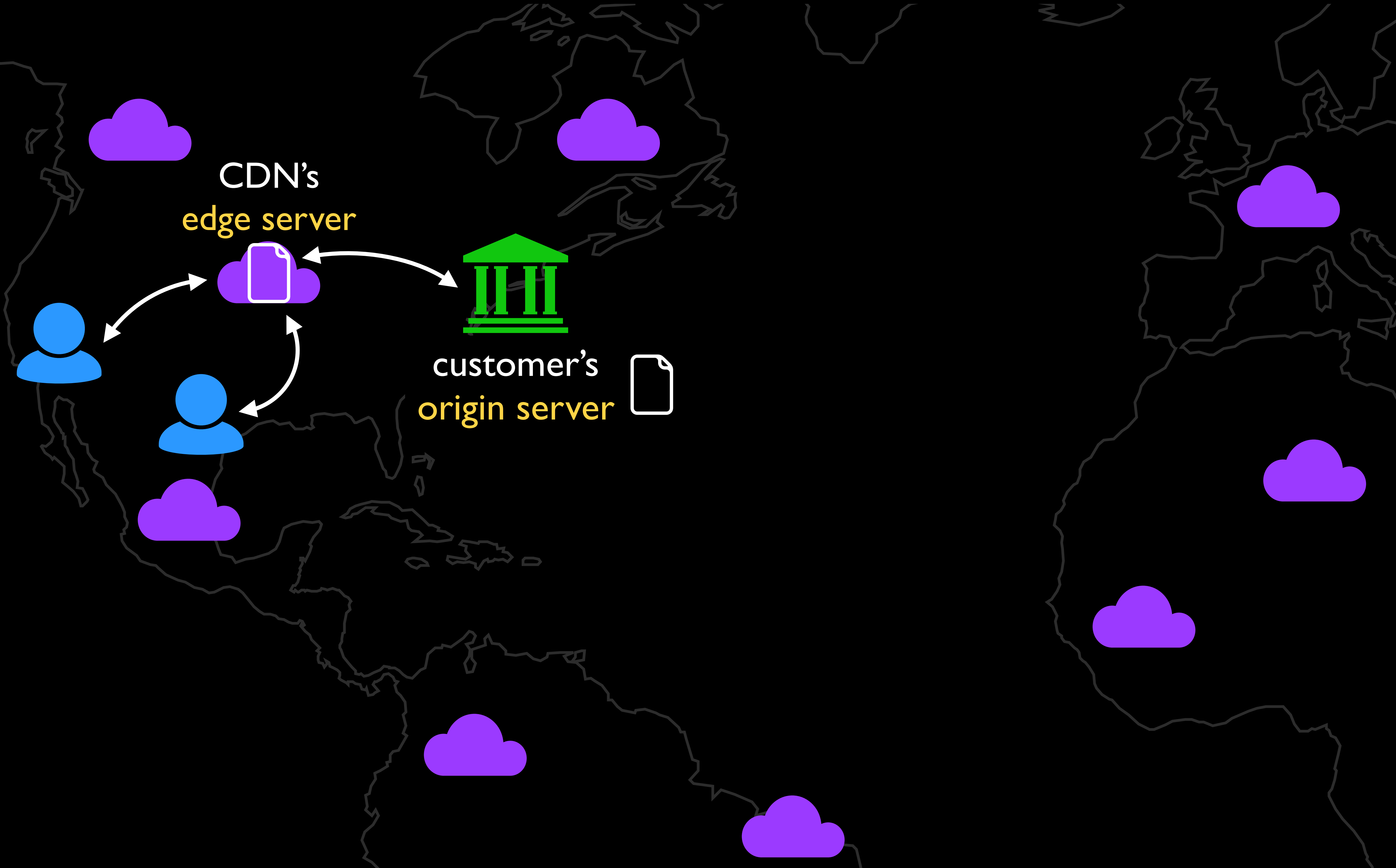
Content Delivery Networks host their customers’ websites



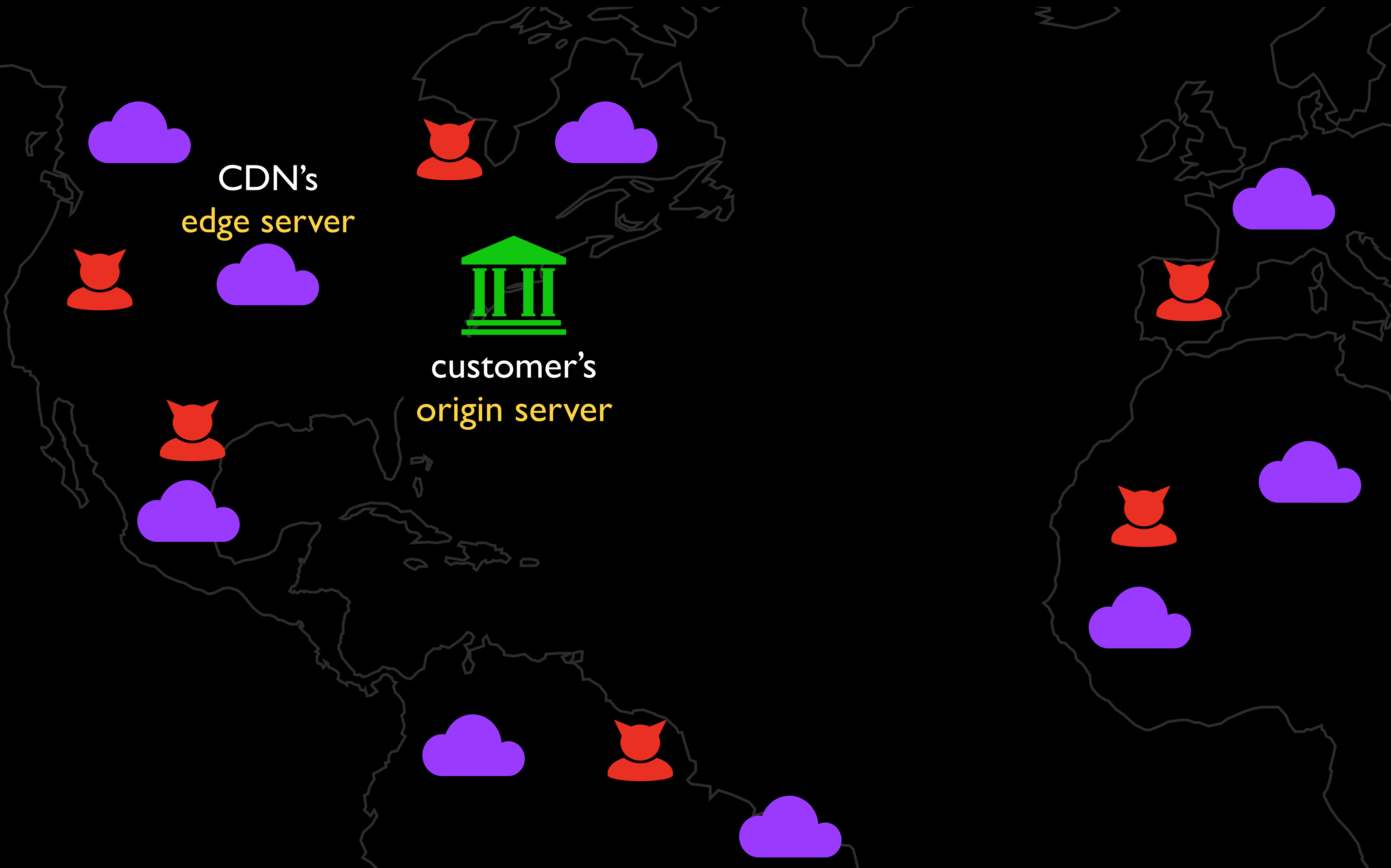
CDNs reduce page load times



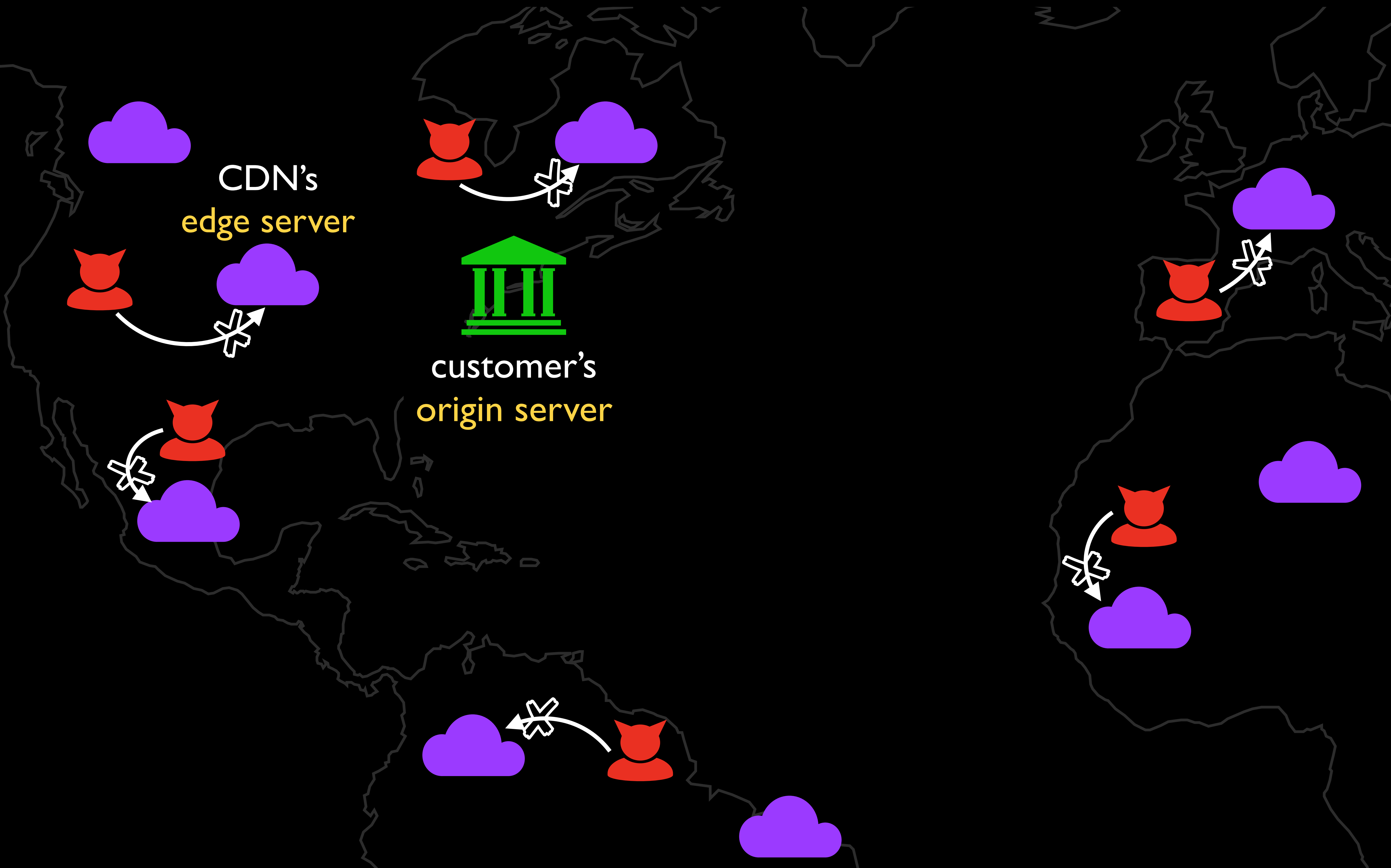
CDNs reduce page load times



CDNs mitigate and block attacks



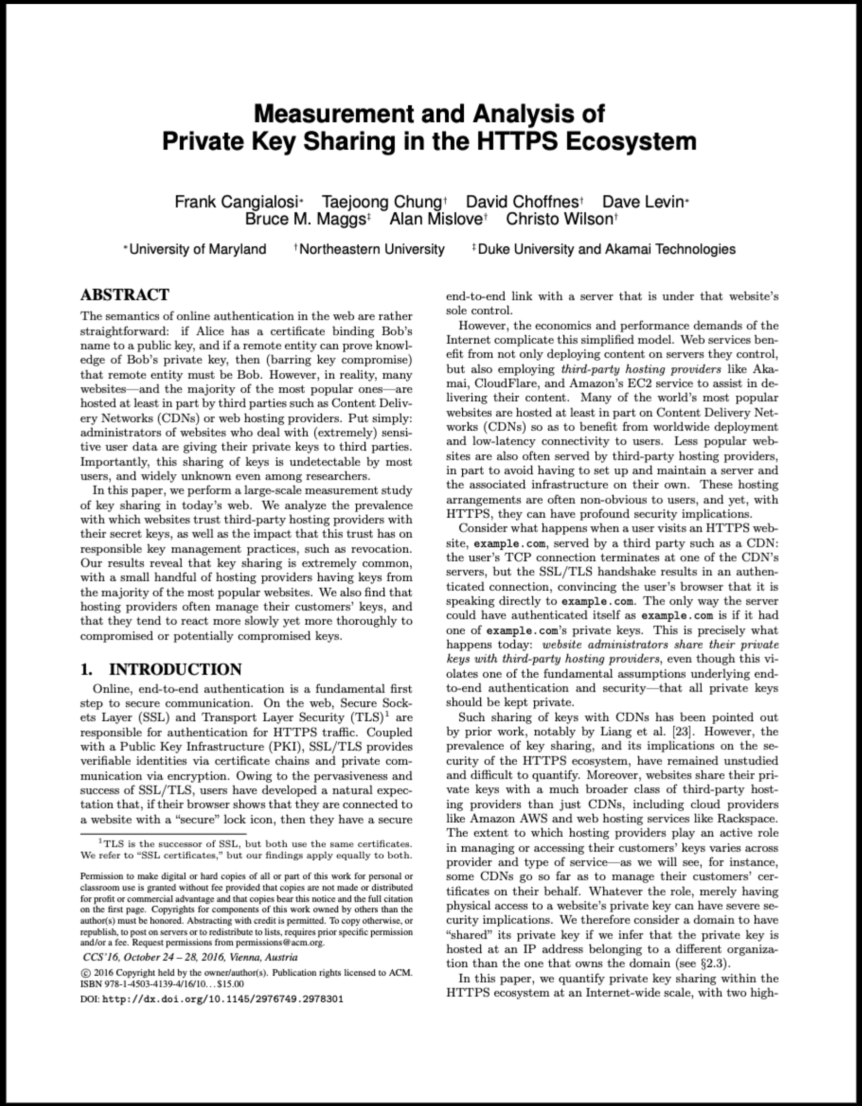
CDNs mitigate and block attacks



Customers share their keys with CDNs

Customers share their keys with CDNs

Key sharing is widespread



Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem

Frank Cangialosi* Taejoong Chung† David Choffnes† Dave Levin*
Bruce M. Maggs‡ Alan Mislove† Christo Wilson†

*University of Maryland †Northeastern University ‡Duke University and Akamai Technologies

ABSTRACT

The semantics of online authentication in the web are rather straightforward: if Alice has a certificate binding Bob's name to a public key, and if a remote entity can prove knowledge of Bob's private key, then (barring key compromise) that remote entity must be Bob. However, in reality, many websites—and the majority of the most popular ones—are hosted at least in part by third parties such as Content Delivery Networks (CDNs) or web hosting providers. Put simply: administrators of websites who deal with (extremely) sensitive user data are giving their private keys to third parties. Importantly, this sharing of keys is undetectable by most users, and widely unknown even among researchers.

In this paper, we perform a large-scale measurement study of key sharing in today's web. We analyze the prevalence with which websites trust third-party hosting providers with their secret keys, as well as the impact that this trust has on responsible key management practices, such as revocation. Our results reveal that key sharing is extremely common, with a small handful of hosting providers having keys from the majority of the most popular websites. We also find that hosting providers often manage their customers' keys, and that they tend to react more slowly yet more thoroughly to compromised or potentially compromised keys.

1. INTRODUCTION

Online, end-to-end authentication is a fundamental first step to secure communication. On the web, Secure Sockets Layer (SSL) and Transport Layer Security (TLS)[1] are responsible for authentication for HTTPS traffic. Coupled with a Public Key Infrastructure (PKI), SSL/TLS provides verifiable identities via certificate chains and private communication via encryption. Owing to the pervasiveness and success of SSL/TLS, users have developed a natural expectation that, if their browser shows that they are connected to a website with a "secure" lock icon, then they have a secure end-to-end link with a server that is under that website's sole control.

However, the economics and performance demands of the Internet complicate this simplified model. Web services benefit from not only deploying content on servers they control, but also employing *third-party hosting providers* like Akamai, CloudFlare, and Amazon's EC2 service to assist in delivering their content. Many of the world's most popular websites are hosted at least in part on Content Delivery Networks (CDNs) so as to benefit from worldwide deployment and low-latency connectivity to users. Less popular websites are also often served by third-party hosting providers, in part to avoid having to set up and maintain a server and the associated infrastructure on their own. These hosting arrangements are often non-obvious to users, and yet, with HTTPS, they can have profound security implications.

Consider what happens when a user visits an HTTPS website, `example.com`, served by a third party such as a CDN: the user's TCP connection terminates at one of the CDN's servers, but the SSL/TLS handshake results in an authenticated connection, convincing the user's browser that it is speaking directly to `example.com`. The only way the server could have authenticated itself as `example.com` is if it had one of `example.com`'s private keys. This is precisely what happens today: *website administrators share their private keys with third-party hosting providers*, even though this violates one of the fundamental assumptions underlying end-to-end authentication and security—that all private keys should be kept private.

Such sharing of keys with CDNs has been pointed out by prior work, notably by Liang et al. [23]. However, the prevalence of key sharing, and its implications on the security of the HTTPS ecosystem, have remained unstudied and difficult to quantify. Moreover, websites share their private keys with a much broader class of third-party hosting providers than just CDNs, including cloud providers like Amazon AWS and web hosting services like Rackspace. The extent to which hosting providers play an active role in managing or accessing their customers' keys varies across provider and type of service—as we will see, for instance, some CDNs go so far as to manage their customers' certificates on their behalf. Whatever the role, merely having physical access to a website's private key can have severe security implications. We therefore consider a domain to have "shared" its private key if we infer that the private key is hosted at an IP address belonging to a different organization than the one that owns the domain (see §2.3).

In this paper, we quantify private key sharing within the HTTPS ecosystem at an Internet-wide scale, with two high-

[1] TLS is the successor of SSL, but both use the same certificates. We refer to "SSL certificates," but our findings apply equally to both.

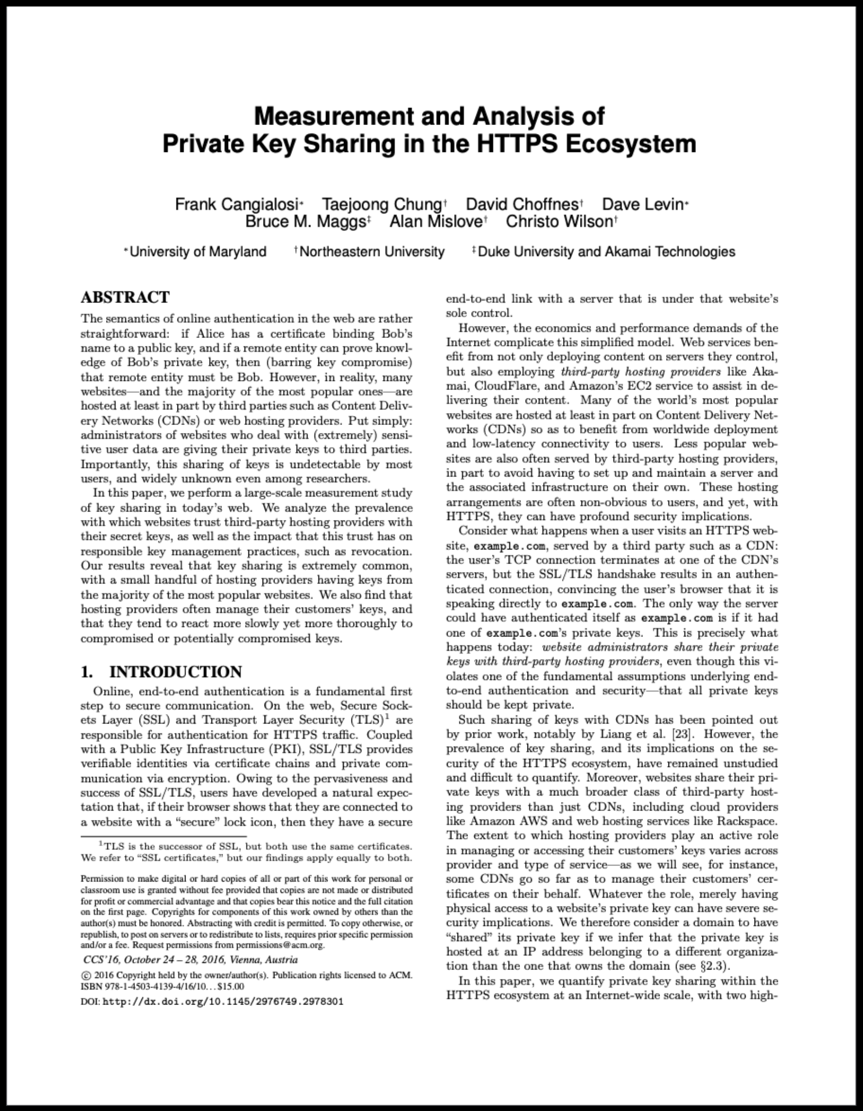
CCS'16, October 24 – 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978301

Cangialosi et al., CCS 2016

Key sharing is widespread



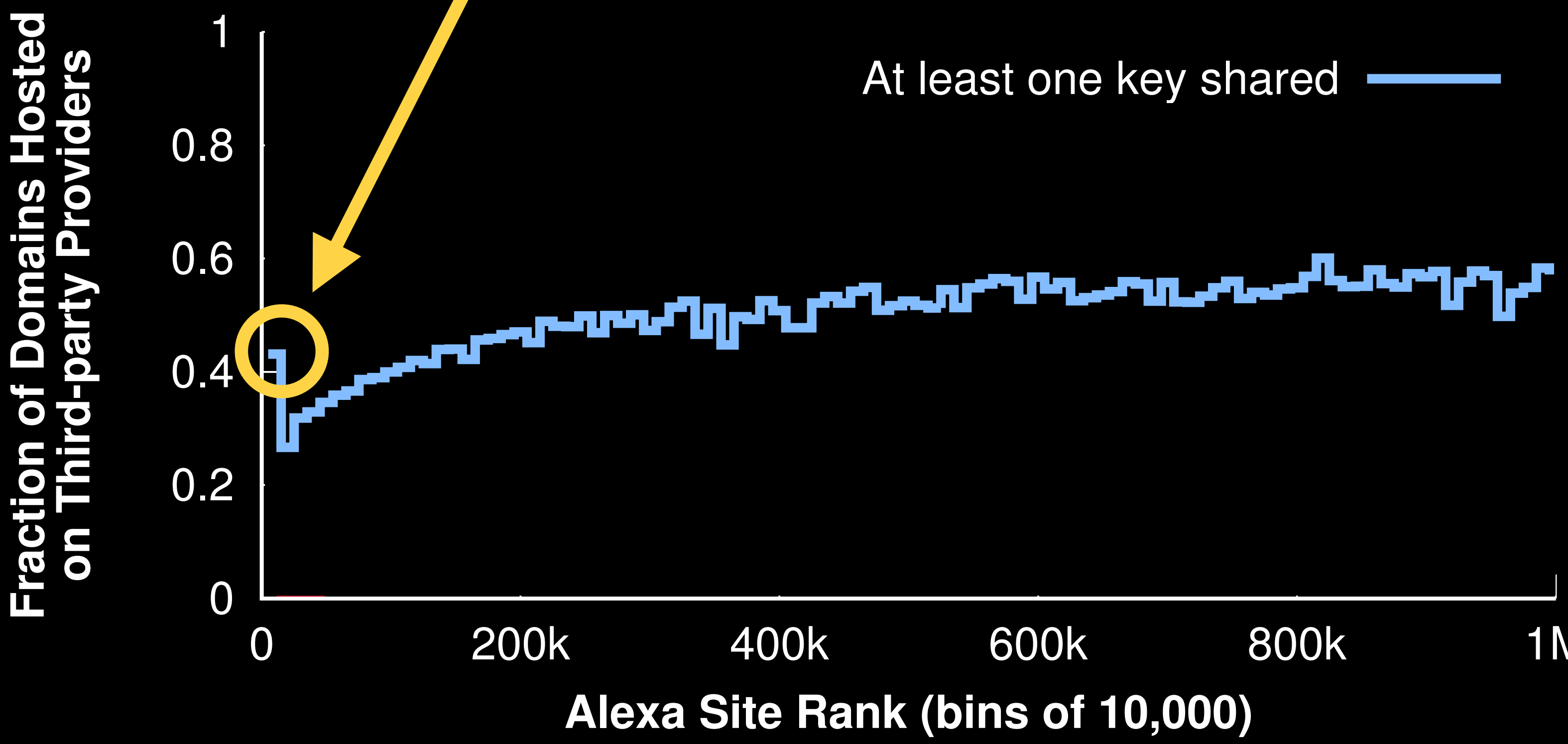
Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem

Frank Cangialosi Taejoong Chung David Choffnes Dave Levin Bruce M. Maggs Alan Mislove Christo Wilson

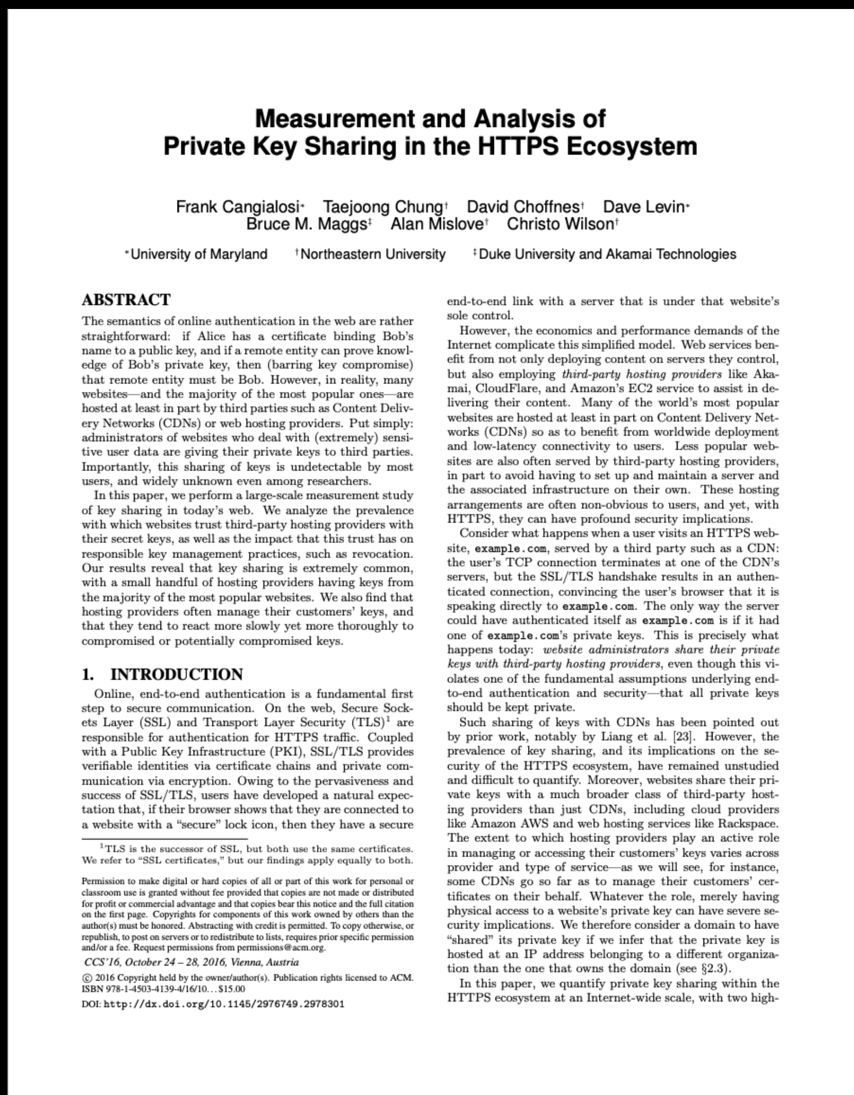
University of Maryland Northeastern University Duke University and Akamai Technologies

Cangialosi et al., CCS 2016

43% of the top 10k most popular websites



Key sharing is widespread



Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem

Frank Cangialosi[*] Taejoong Chung[†] David Choffnes[†] Dave Levin[*]
Bruce M. Maggs[‡] Alan Mislove[†] Christo Wilson[†]

[*]University of Maryland [†]Northeastern University [‡]Duke University and Akamai Technologies

ABSTRACT

The semantics of online authentication in the web are rather straightforward: if Alice has a certificate binding Bob's name to a public key, and if a remote entity can prove knowledge of Bob's private key, then (barring key compromise) that remote entity must be Bob. However, in reality, many websites—and the majority of the most popular ones—are hosted at least in part by third parties such as Content Delivery Networks (CDNs) or web hosting providers. Put simply: administrators of websites who deal with (extremely) sensitive user data are giving their private keys to third parties. Importantly, this sharing of keys is undetectable by most users, and widely unknown even among researchers.

In this paper, we perform a large-scale measurement study of key sharing in today's web. We analyze the prevalence with which websites trust third-party hosting providers with their secret keys, as well as the impact that this trust has on responsible key management practices, such as revocation. Our results reveal that key sharing is extremely common, with a small handful of hosting providers having keys from the majority of the most popular websites. We also find that hosting providers often manage their customers' keys, and that they tend to react more slowly yet more thoroughly to compromised or potentially compromised keys.

1. INTRODUCTION

Online, end-to-end authentication is a fundamental first step to secure communication. On the web, Secure Sockets Layer (SSL) and Transport Layer Security (TLS)[1] are responsible for authentication for HTTPS traffic. Coupled with a Public Key Infrastructure (PKI), SSL/TLS provides verifiable identities via certificate chains and private communication via encryption. Owing to the pervasiveness and success of SSL/TLS, users have developed a natural expectation that, if their browser shows that they are connected to a website with a "secure" lock icon, then they have a secure end-to-end link with a server that is under that website's sole control.

However, the economics and performance demands of the Internet complicate this simplified model. Web services benefit from not only deploying content on servers they control, but also employing *third-party hosting providers* like Akamai, CloudFlare, and Amazon's EC2 service to assist in delivering their content. Many of the world's most popular websites are hosted at least in part on Content Delivery Networks (CDNs) so as to benefit from worldwide deployment and low-latency connectivity to users. Less popular websites are also often served by third-party hosting providers, in part to avoid having to set up and maintain a server and the associated infrastructure on their own. These hosting arrangements are often non-obvious to users, and yet, with HTTPS, they can have profound security implications.

Consider what happens when a user visits an HTTPS website, `example.com`, served by a third party such as a CDN: the user's TCP connection terminates at one of the CDN's servers, but the SSL/TLS handshake results in an authenticated connection, convincing the user's browser that it is speaking directly to `example.com`. The only way the server could have authenticated itself as `example.com` is if it had one of `example.com`'s private keys. This is precisely what happens today: *website administrators share their private keys with third-party hosting providers*, even though this violates one of the fundamental assumptions underlying end-to-end authentication and security—that all private keys should be kept private.

Such sharing of keys with CDNs has been pointed out by prior work, notably by Liang et al. [23]. However, the prevalence of key sharing, and its implications on the security of the HTTPS ecosystem, have remained unstudied and difficult to quantify. Moreover, websites share their private keys with a much broader class of third-party hosting providers than just CDNs, including cloud providers like Amazon AWS and web hosting services like Rackspace. The extent to which hosting providers play an active role in managing or accessing their customers' keys varies across provider and type of service—as we will see, for instance, some CDNs go so far as to manage their customers' certificates on their behalf. Whatever the role, merely having physical access to a website's private key can have severe security implications. We therefore consider a domain to have "shared" its private key if we infer that the private key is hosted at an IP address belonging to a different organization than the one that owns the domain (see §2.3).

In this paper, we quantify private key sharing within the HTTPS ecosystem at an Internet-wide scale, with two high-

[1]TLS is the successor of SSL, but both use the same certificates. We refer to "SSL certificates," but our findings apply equally to both.

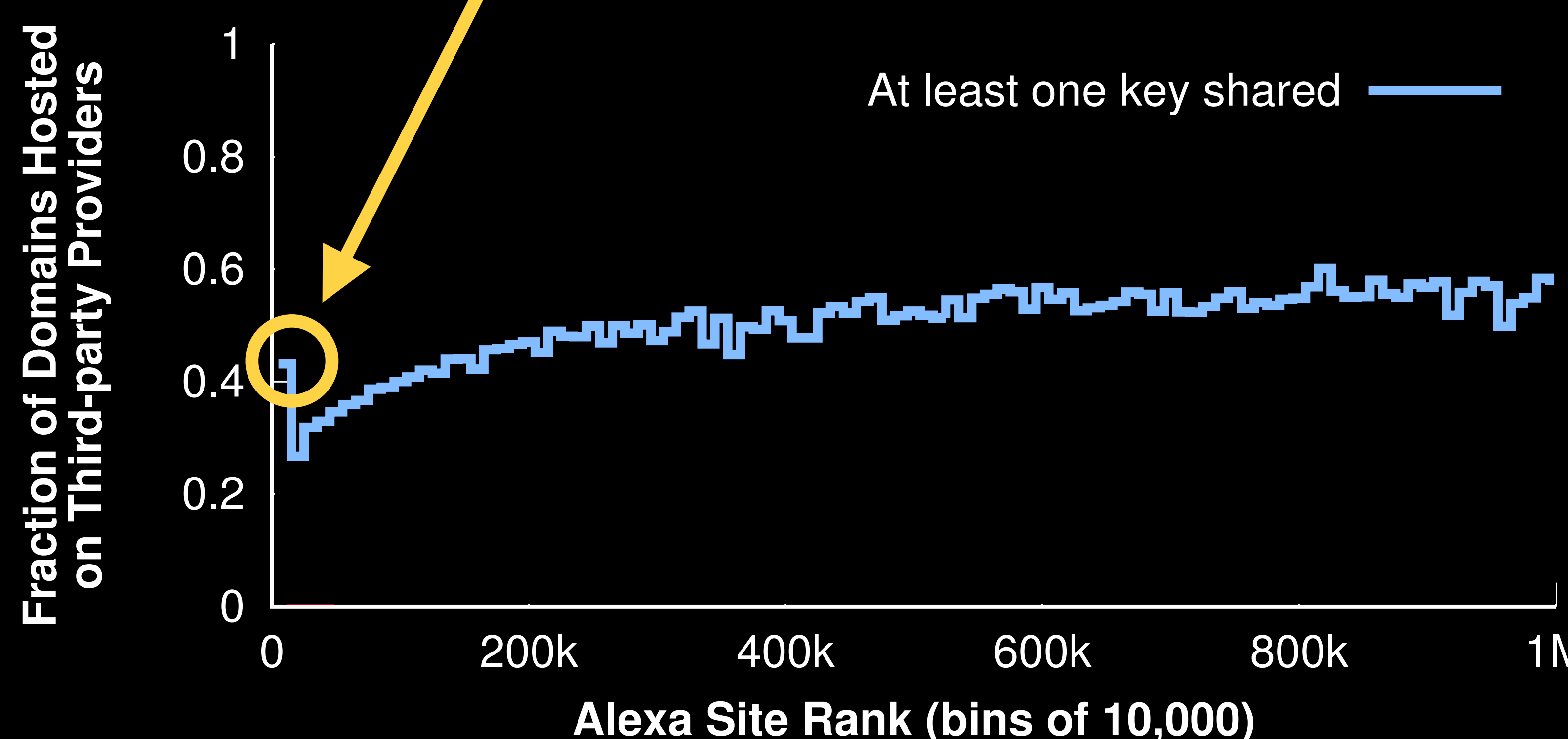
CCS'16, October 24 – 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: `http://dx.doi.org/10.1145/2976749.2978301`

Cangialosi et al., CCS 2016

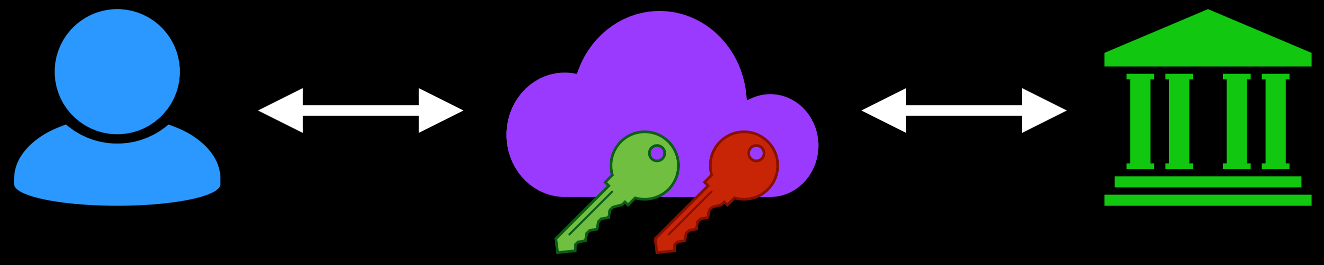
43% of the top 10k most popular websites



The web has consolidated keys in the hands of a few CDNs

Keyless SSL

Introduced by Cloudflare to mitigate key sharing



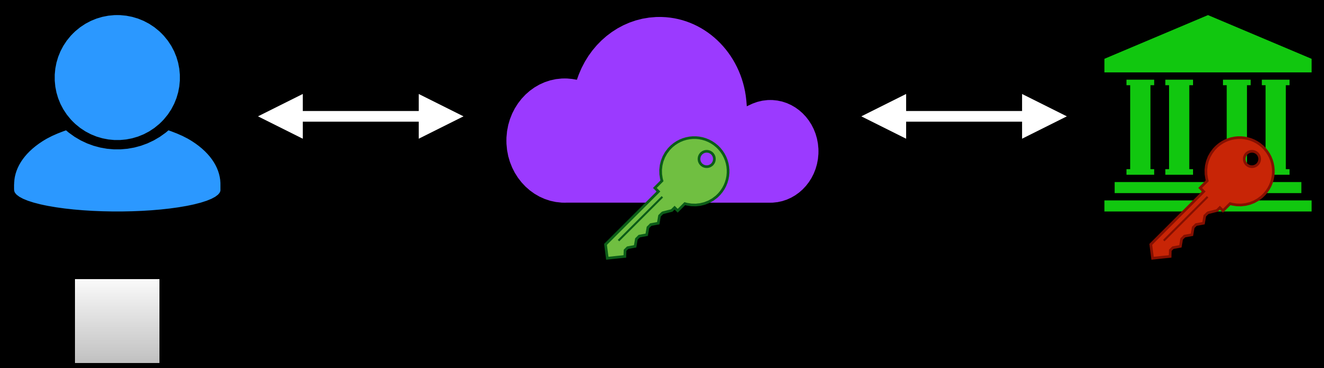
Keyless SSL

Introduced by Cloudflare to mitigate key sharing

Private keys stay at the key server (origin)

Keyless SSL

Introduced by Cloudflare to mitigate key sharing

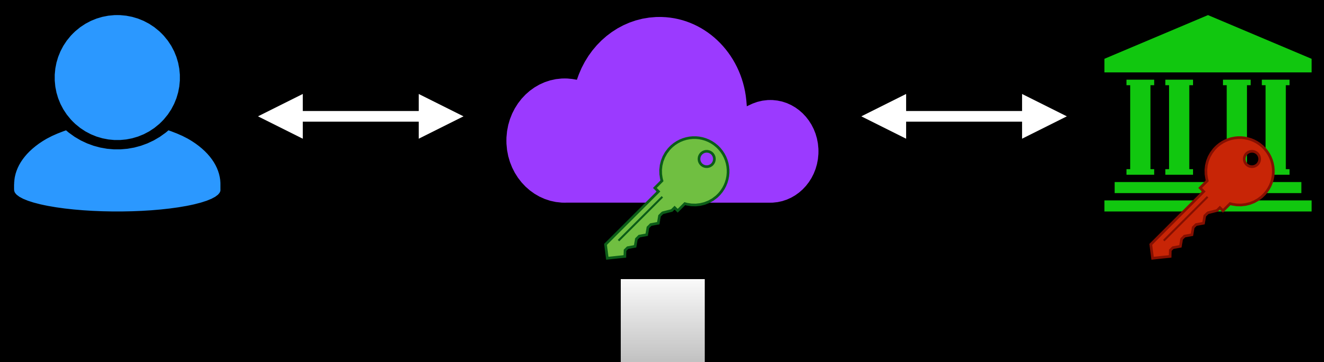


Private keys stay at the key server (origin)

Key server performs actions requiring private key

Keyless SSL

Introduced by Cloudflare to mitigate key sharing

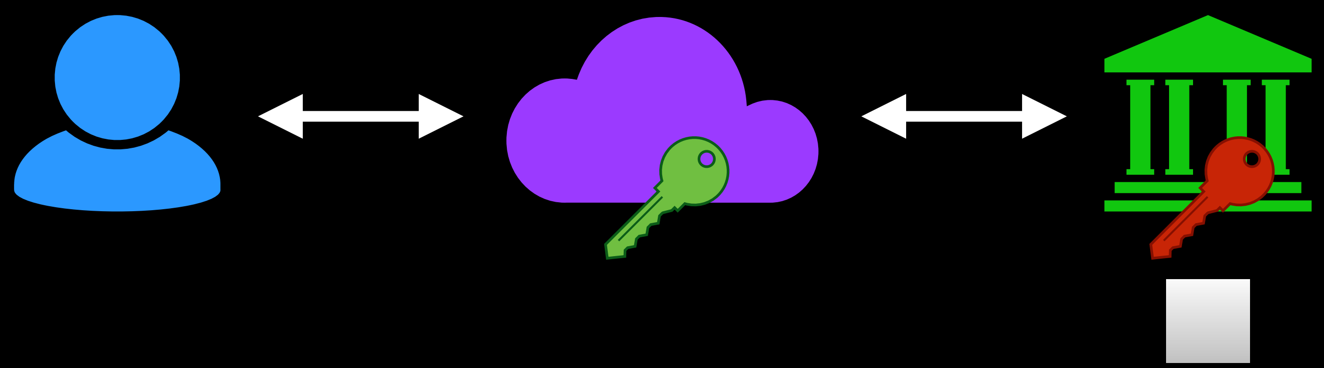


Private keys stay at the key server (origin)

Key server performs actions requiring private key

Keyless SSL

Introduced by Cloudflare to mitigate key sharing

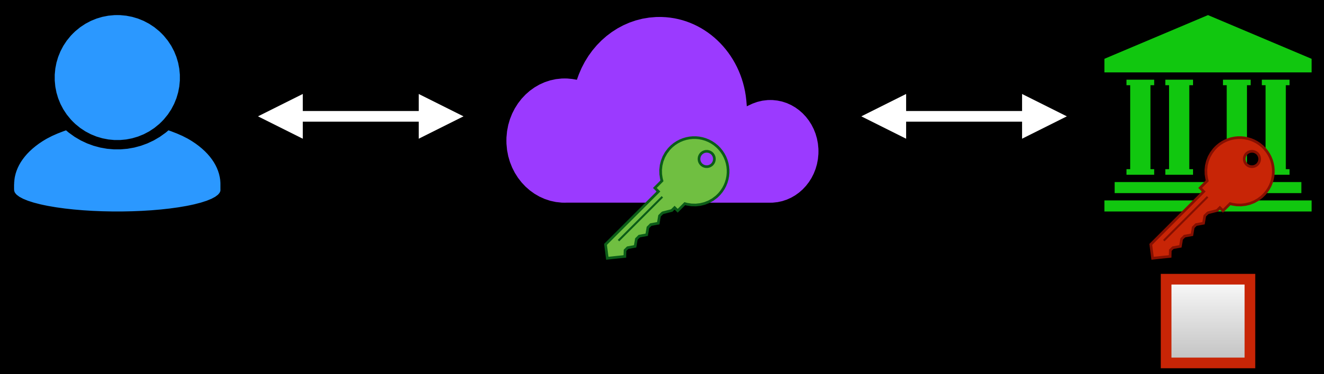


Private keys stay at the key server (origin)

Key server performs actions requiring private key

Keyless SSL

Introduced by Cloudflare to mitigate key sharing

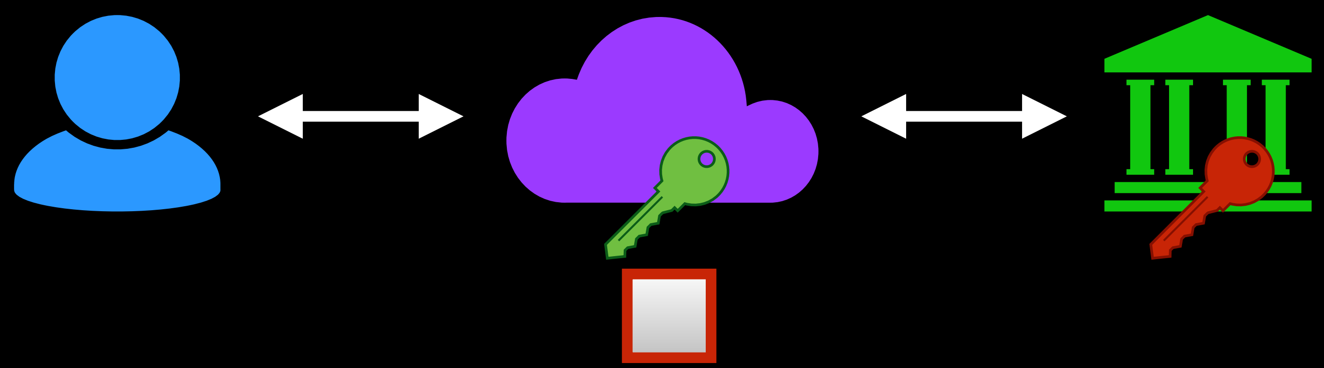


Private keys stay at the key server (origin)

Key server performs actions requiring private key

Keyless SSL

Introduced by Cloudflare to mitigate key sharing

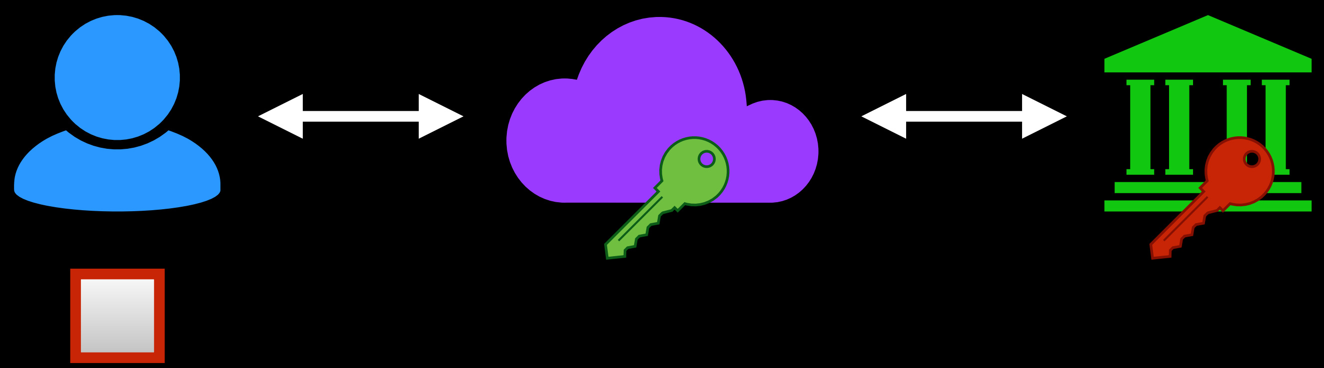


Private keys stay at the key server (origin)

Key server performs actions requiring private key

Keyless SSL

Introduced by Cloudflare to mitigate key sharing

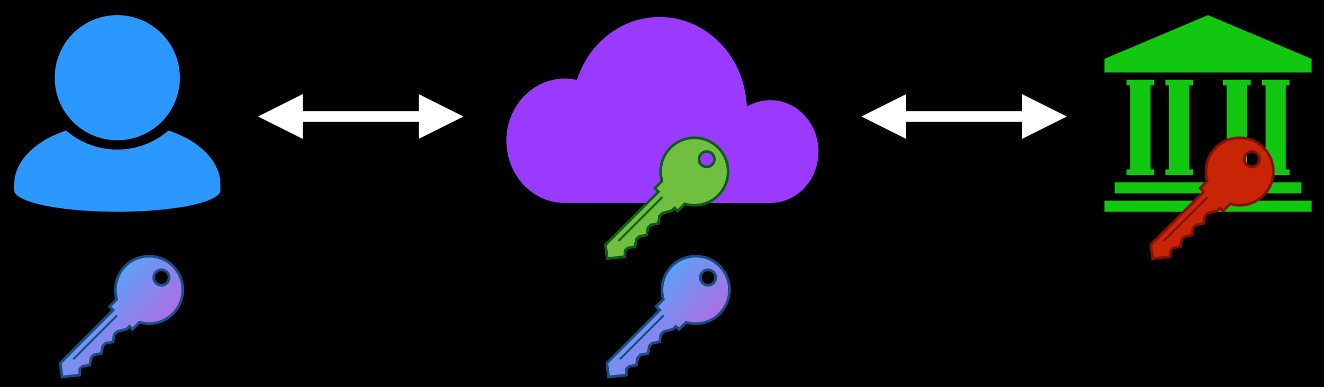


Private keys stay at the key server (origin)

Key server performs actions requiring private key

Keyless SSL

Introduced by Cloudflare to mitigate key sharing



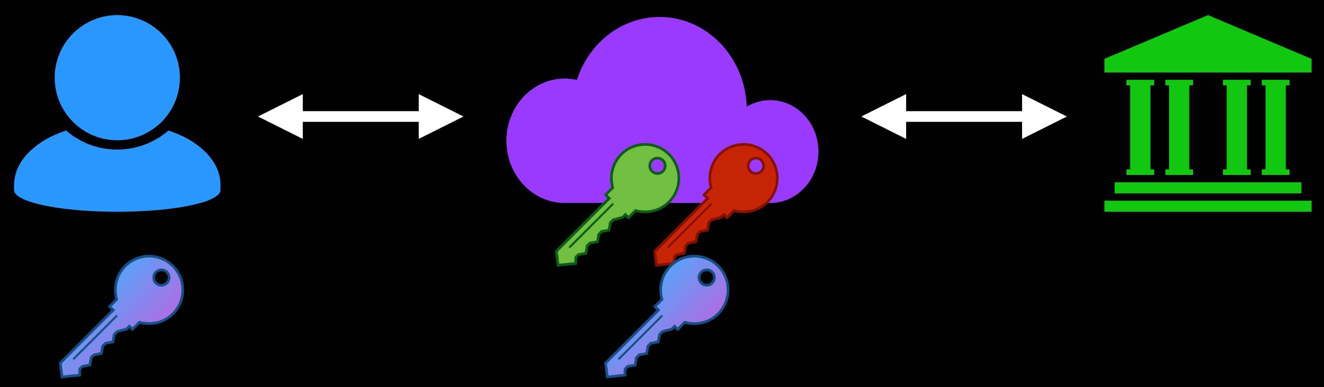
Private keys stay at the key server (origin)

Key server performs actions requiring private key

The CDN learns all session keys

Keyless SSL

Introduced by Cloudflare to mitigate key sharing

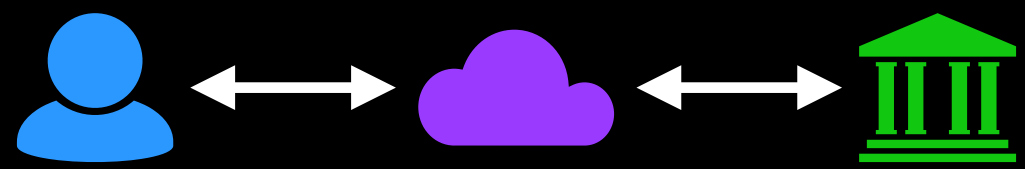


In practice: CDN

Private keys stay at the key server ~~(origin)~~

Key server performs actions requiring private key

The CDN learns all session keys



Can we Maintain privacy

using Legacy applications

on Third-party resources?

Maintain privacy

The CDN is no more trusted than a standard on-path attacker

Legacy applications

Third-party resources

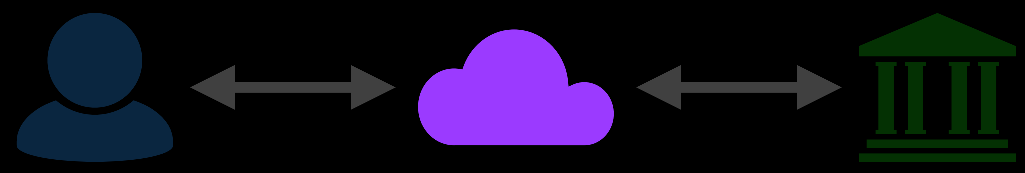
Maintain privacy

The CDN is no more trusted than a standard on-path attacker

Legacy applications

No changes to existing code-bases; facilitates deployment and adoption

Third-party resources



Maintain privacy

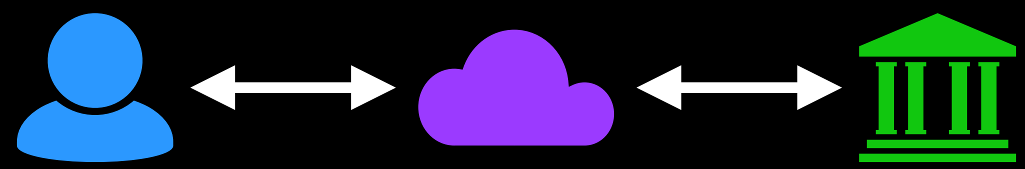
The CDN is no more trusted than a standard on-path attacker

Legacy applications

No changes to existing code-bases; facilitates deployment and adoption

Third-party resources

Leverage the existing infrastructure. One additional assumption: TEEs



Maintain privacy — *The CDN is no more trusted than a standard on-path attacker*

Legacy applications — *No changes to existing code-bases; facilitates deployment and adoption*

Third-party resources — *Leverage the existing infrastructure. One additional assumption: TEEs*

Maintain privacy

The CDN is no more trusted than a standard on-path attacker

Legacy applications

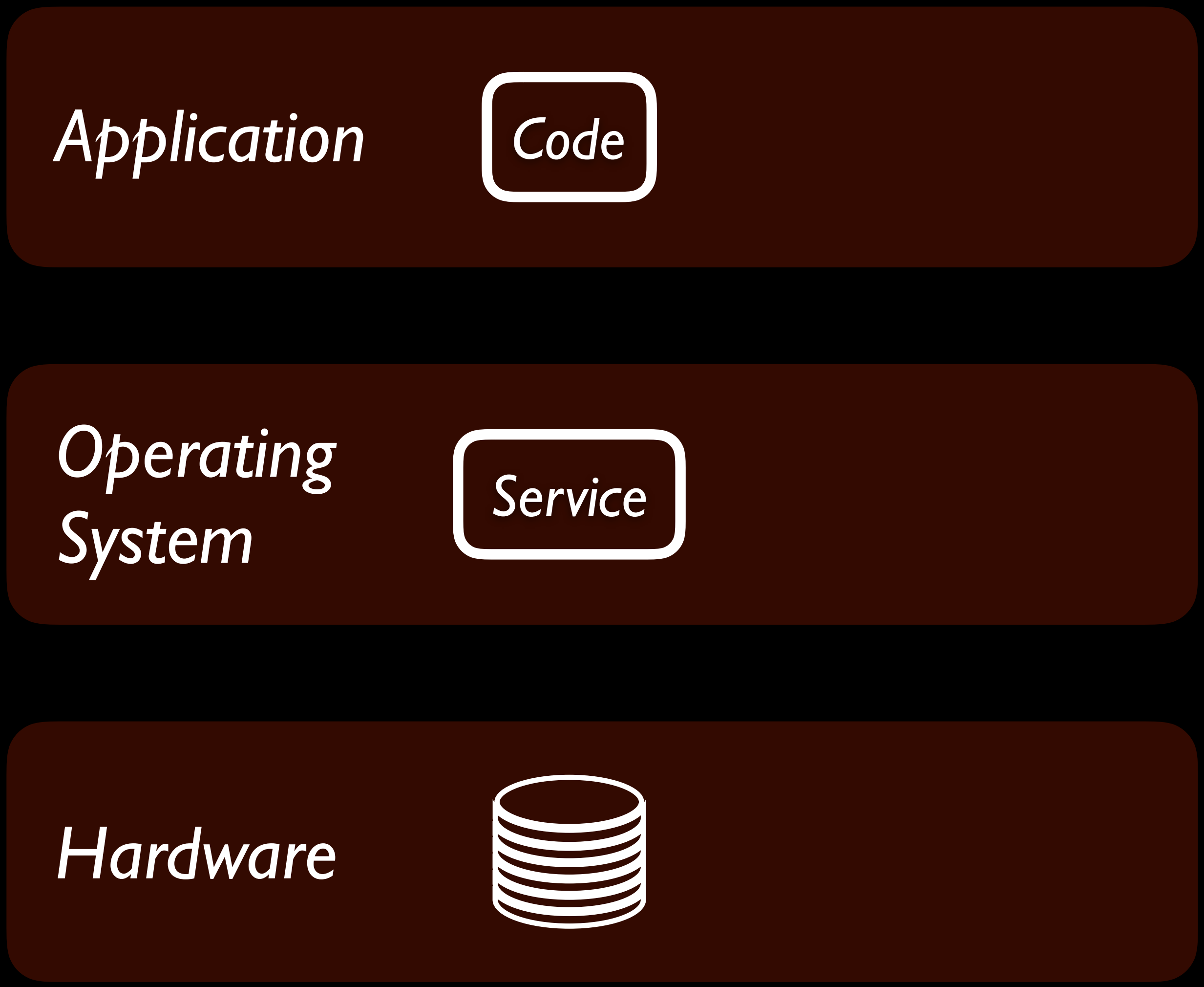
No changes to existing code-bases; facilitates deployment and adoption

Third-party resources

Leverage the existing infrastructure. One additional assumption: TEEs

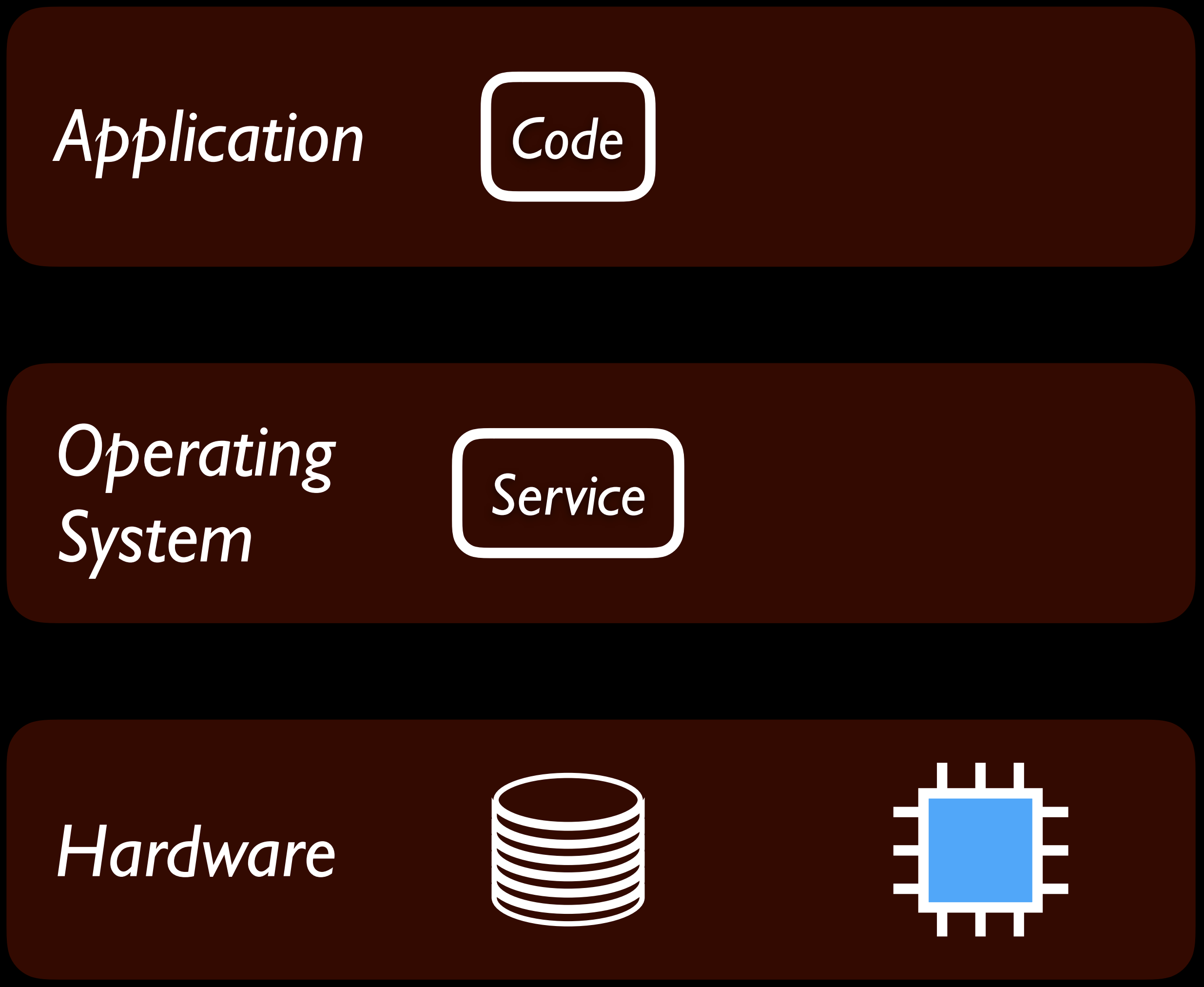
Trusted execution environments

By default, assume all system components are untrusted



Trusted execution environments

By default, assume all system components are untrusted

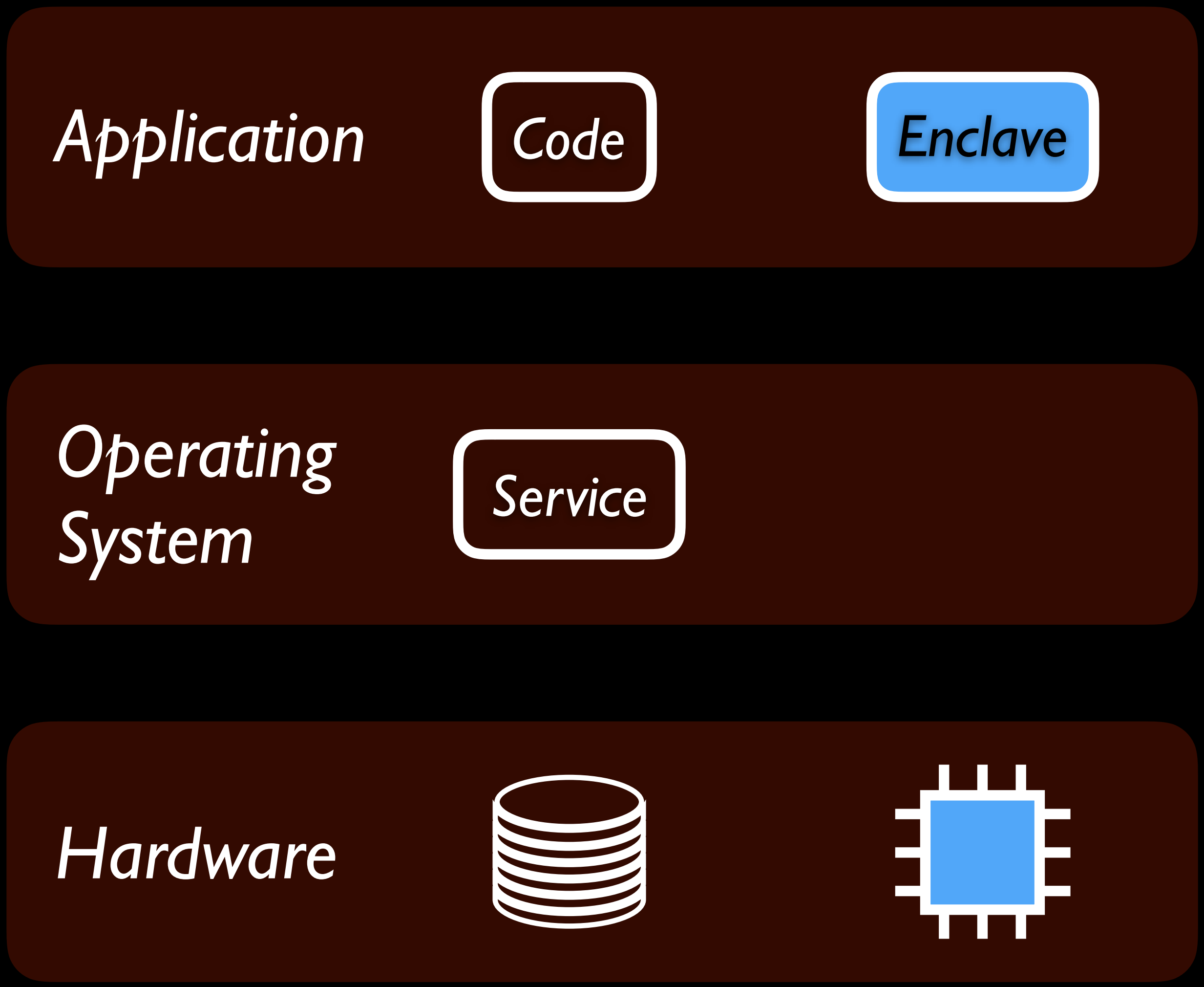


Small trusted CPU
Resistant to physical attacks

Trusted execution environments

By default, assume all system components are untrusted

Enclave: Isolated application memory

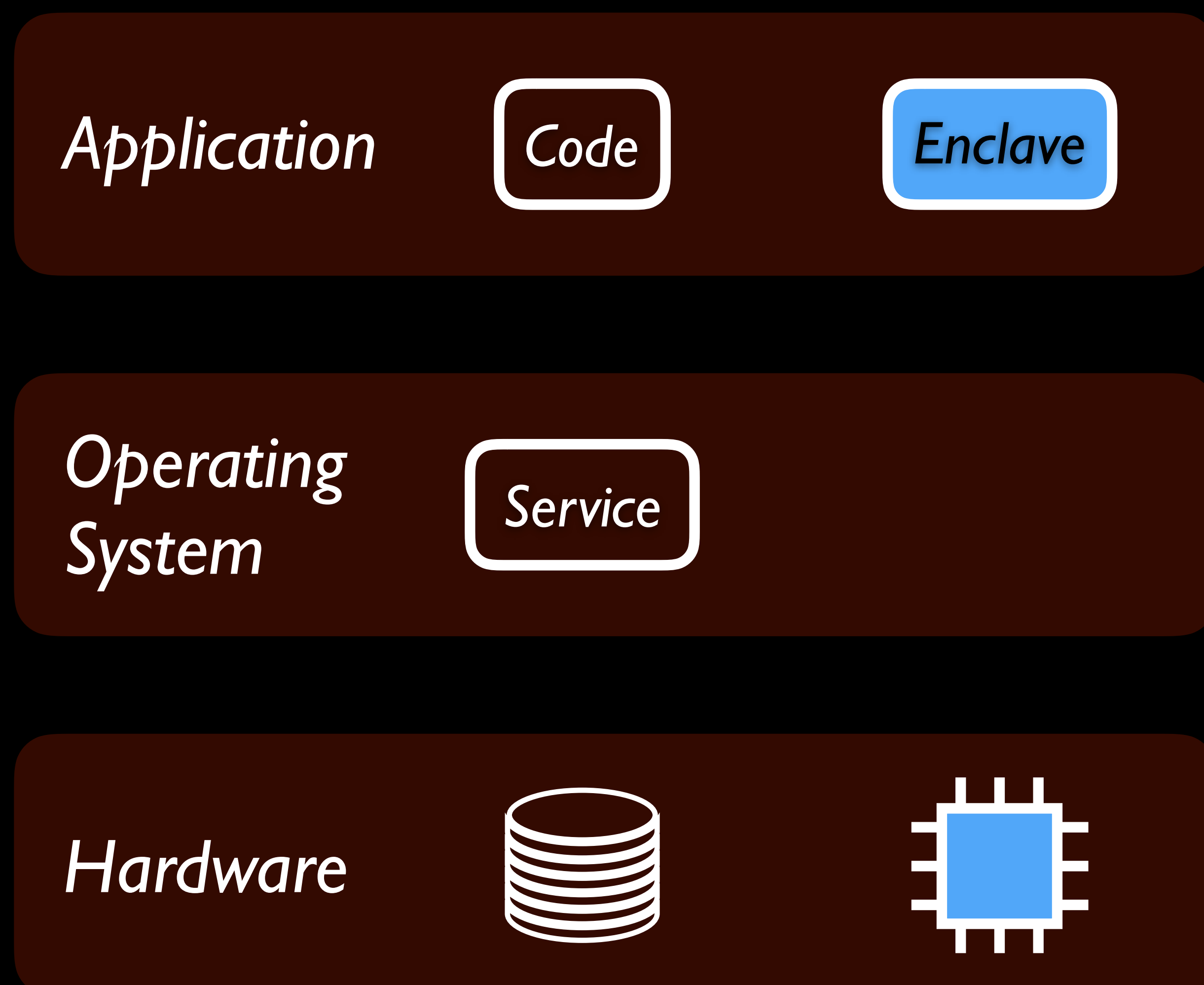


Small trusted CPU
Resistant to physical attacks

Trusted execution environments

By default, assume all system components are untrusted

Enclave: Isolated application memory

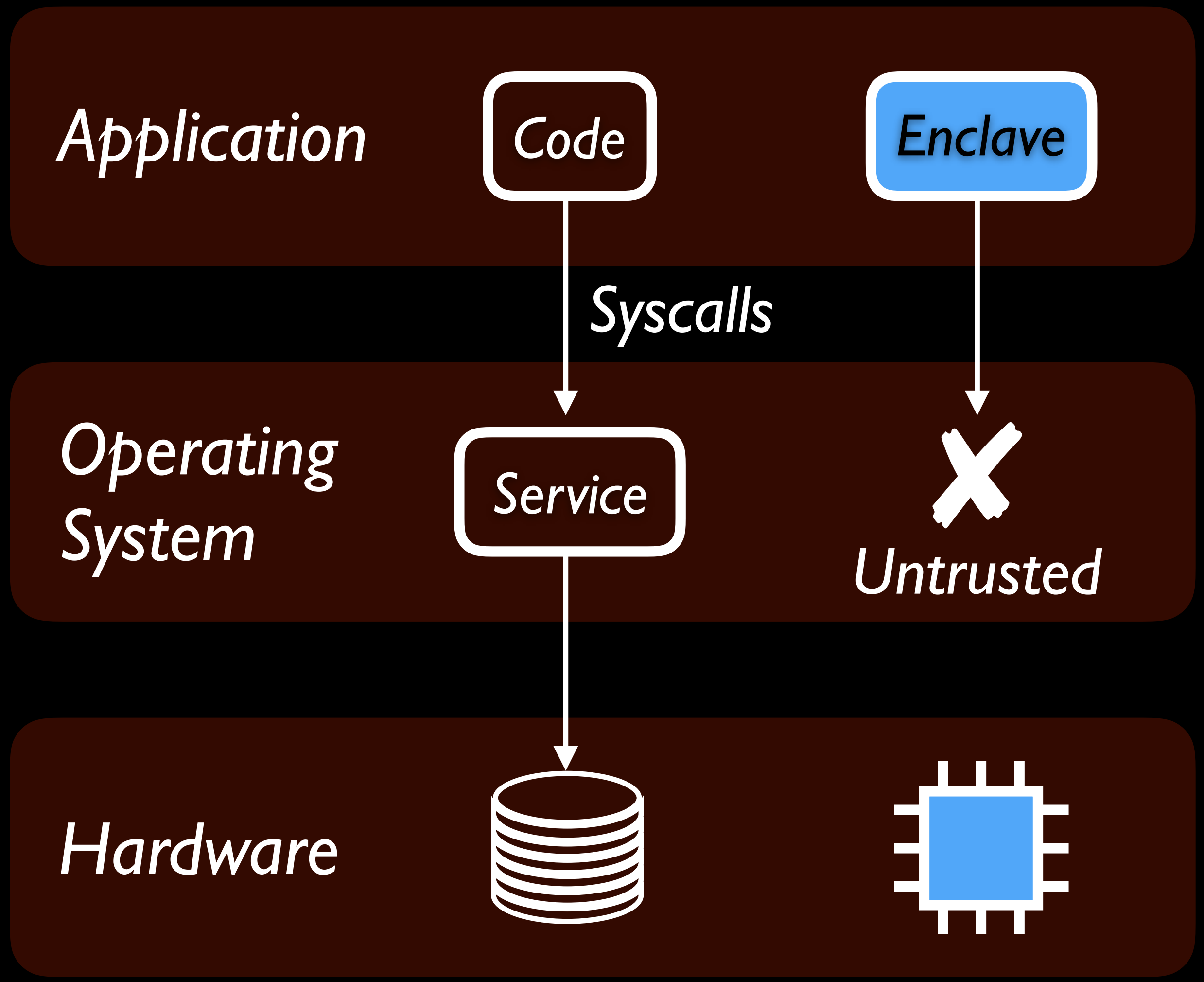


Small trusted CPU
Resistant to physical attacks

Model: Code and data can safely reside inside an enclave

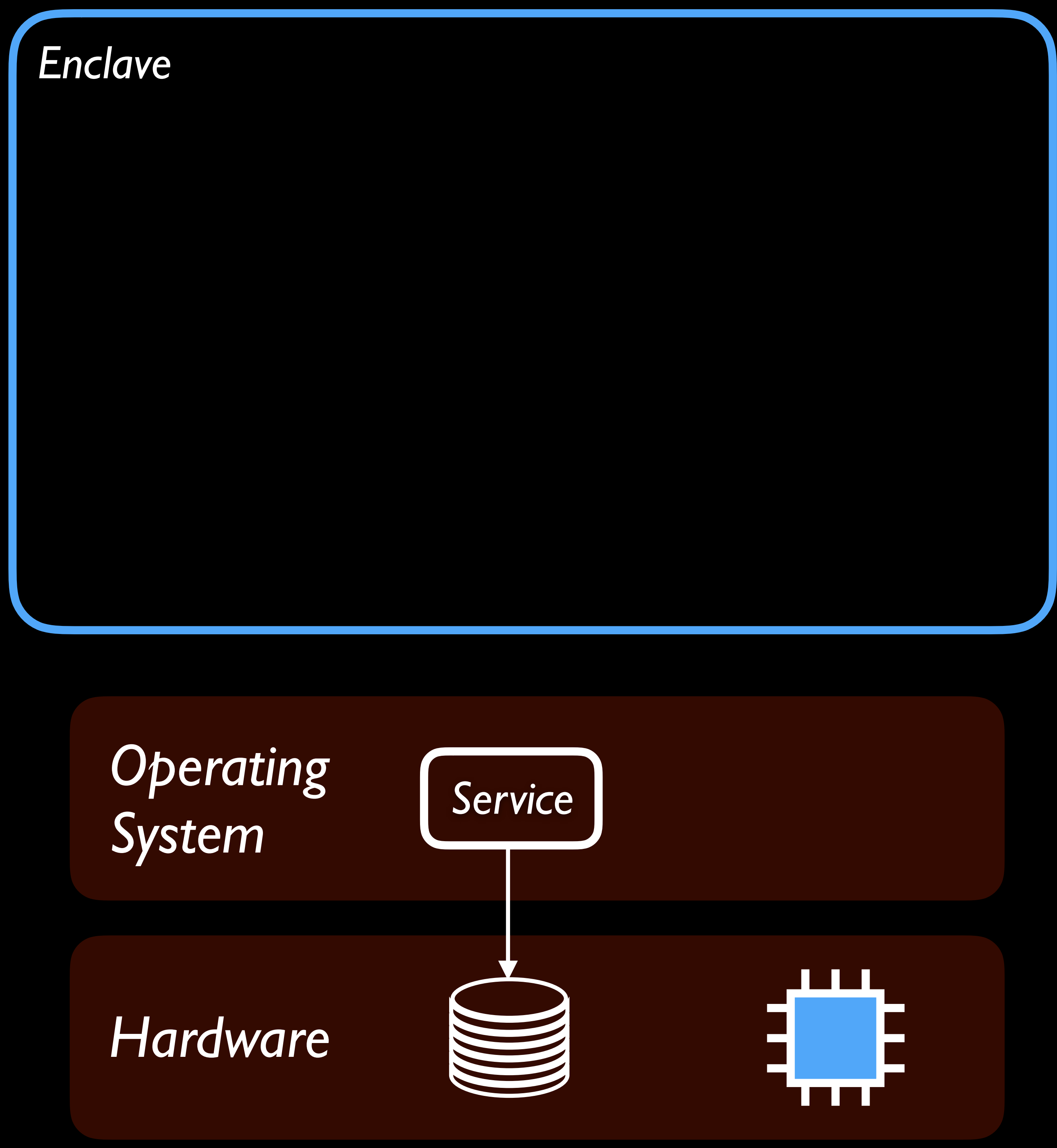
Practical limitations of TEEs

Applications inside enclaves cannot make syscalls



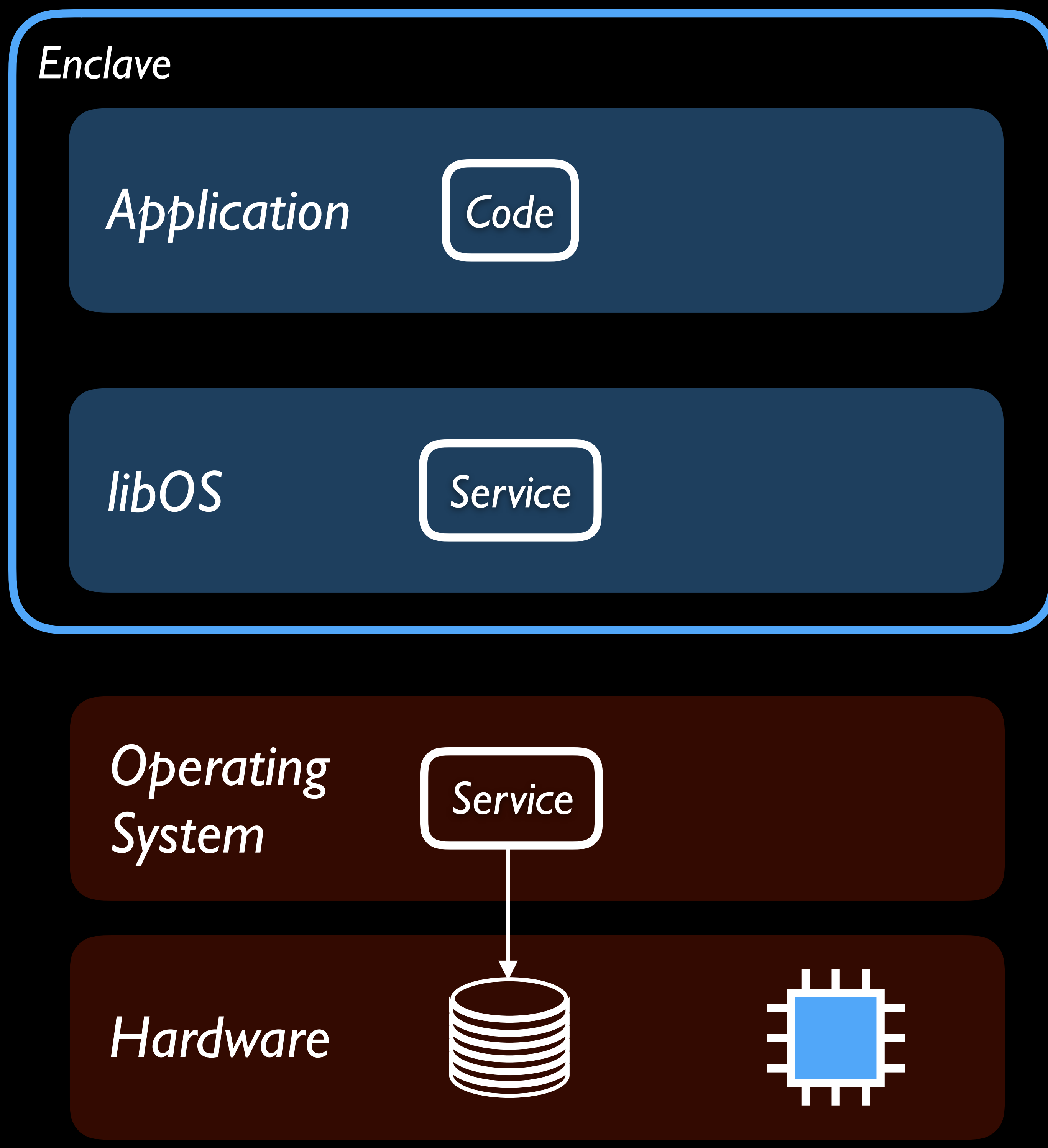
libOSes

Idea: Implement a small "OS" inside the enclave



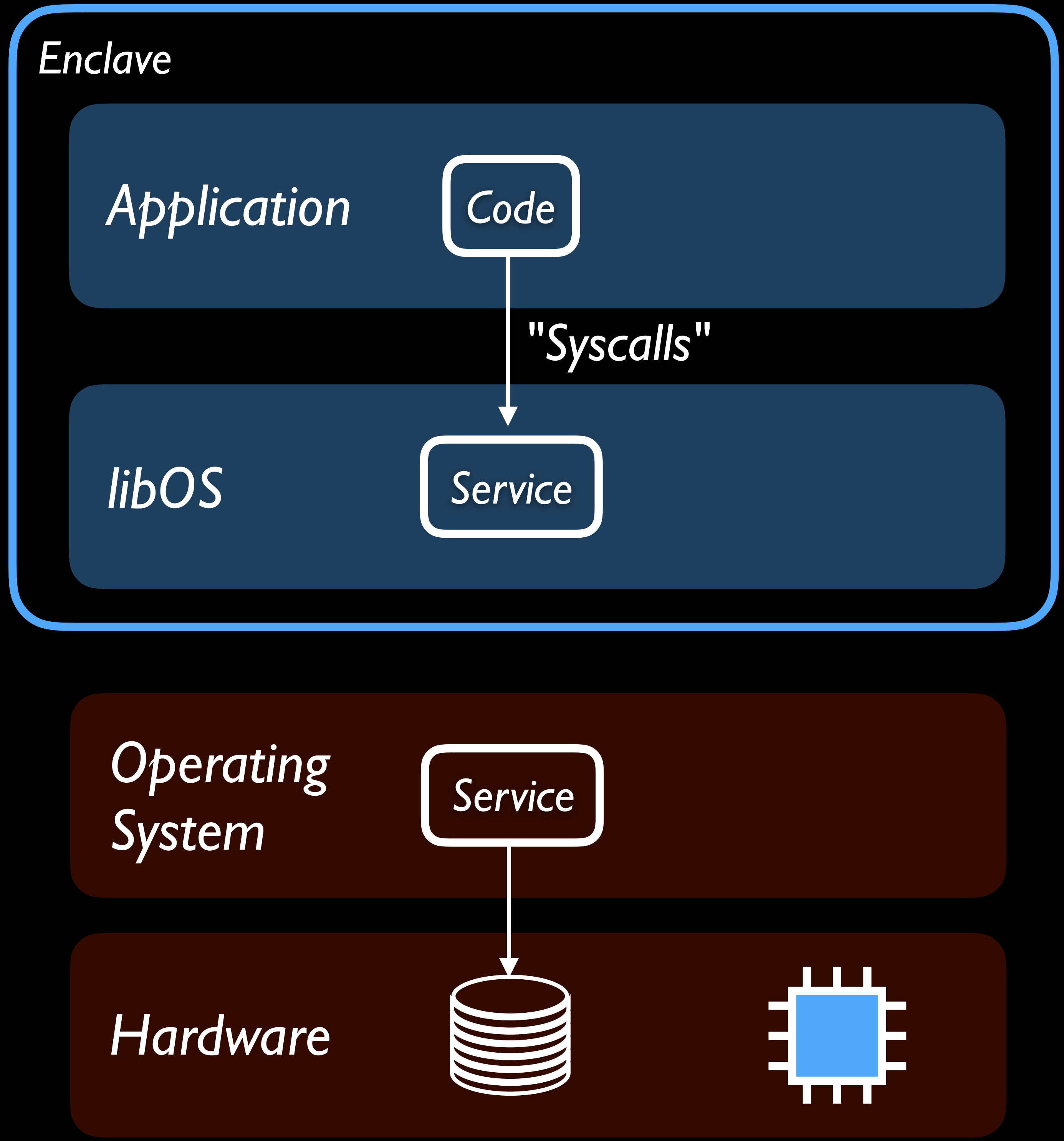
libOSes

Idea: Implement a small "OS" inside the enclave



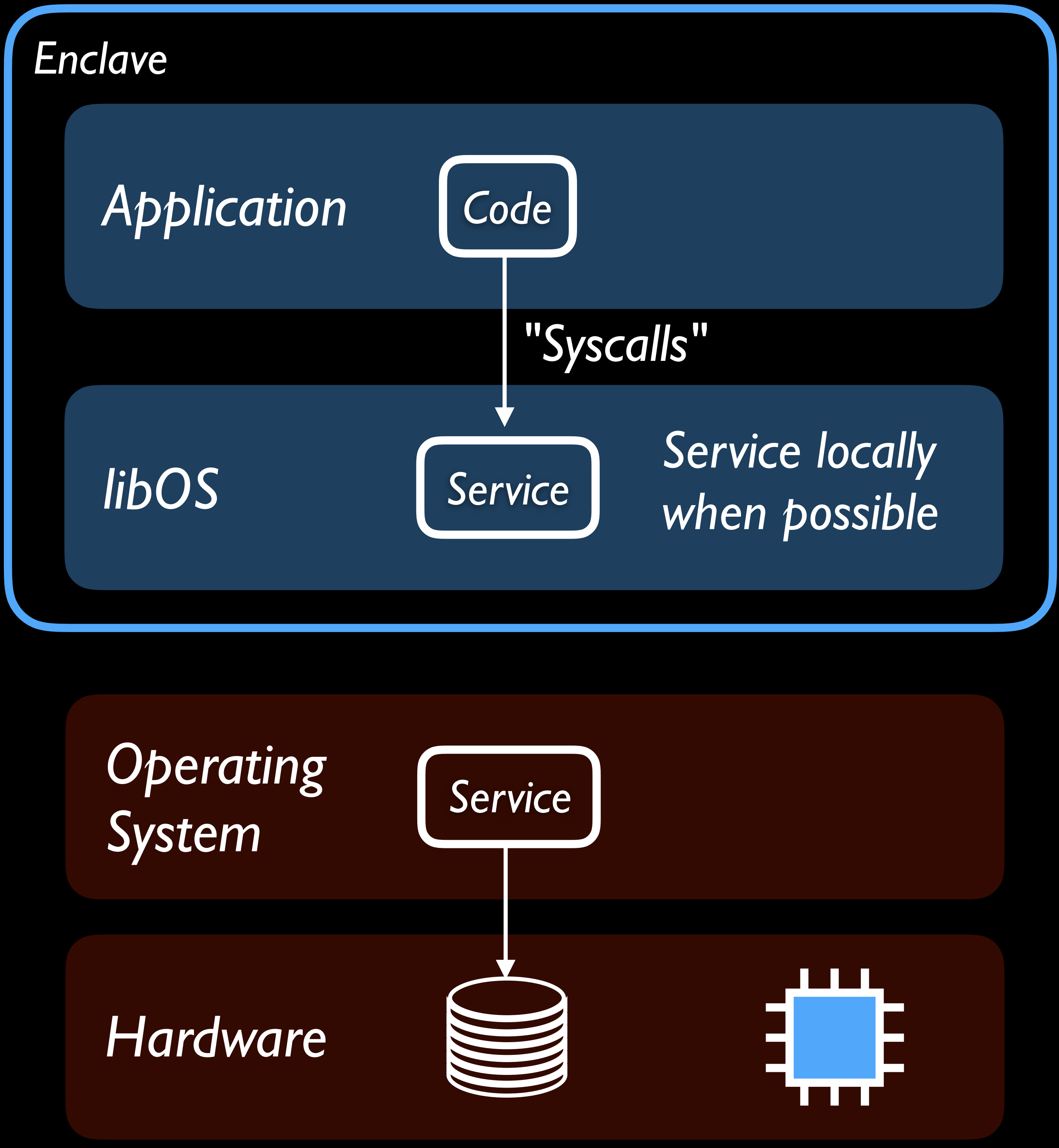
libOSes

Idea: Implement a small "OS" inside the enclave



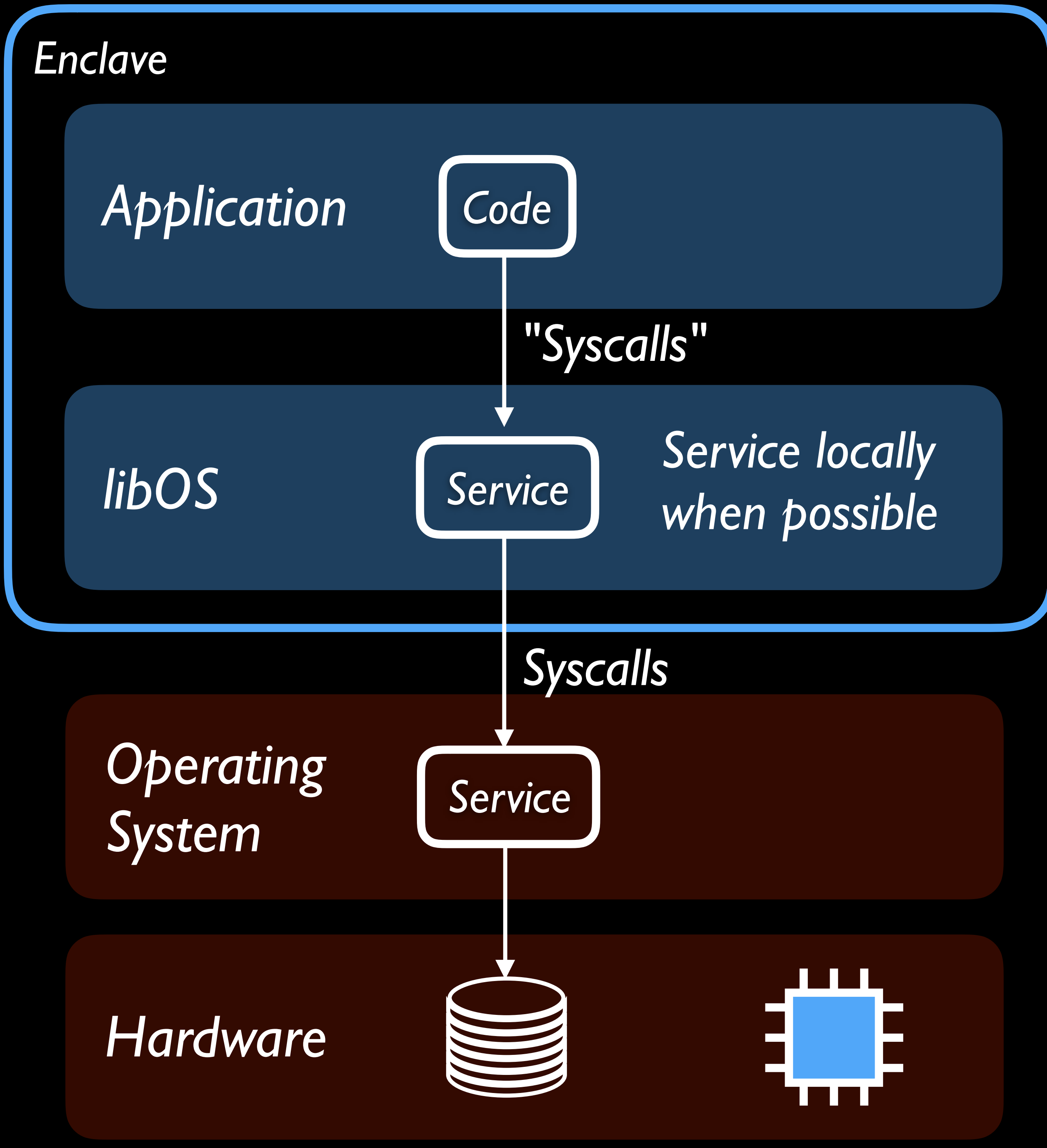
libOSes

Idea: Implement a small “OS” inside the enclave



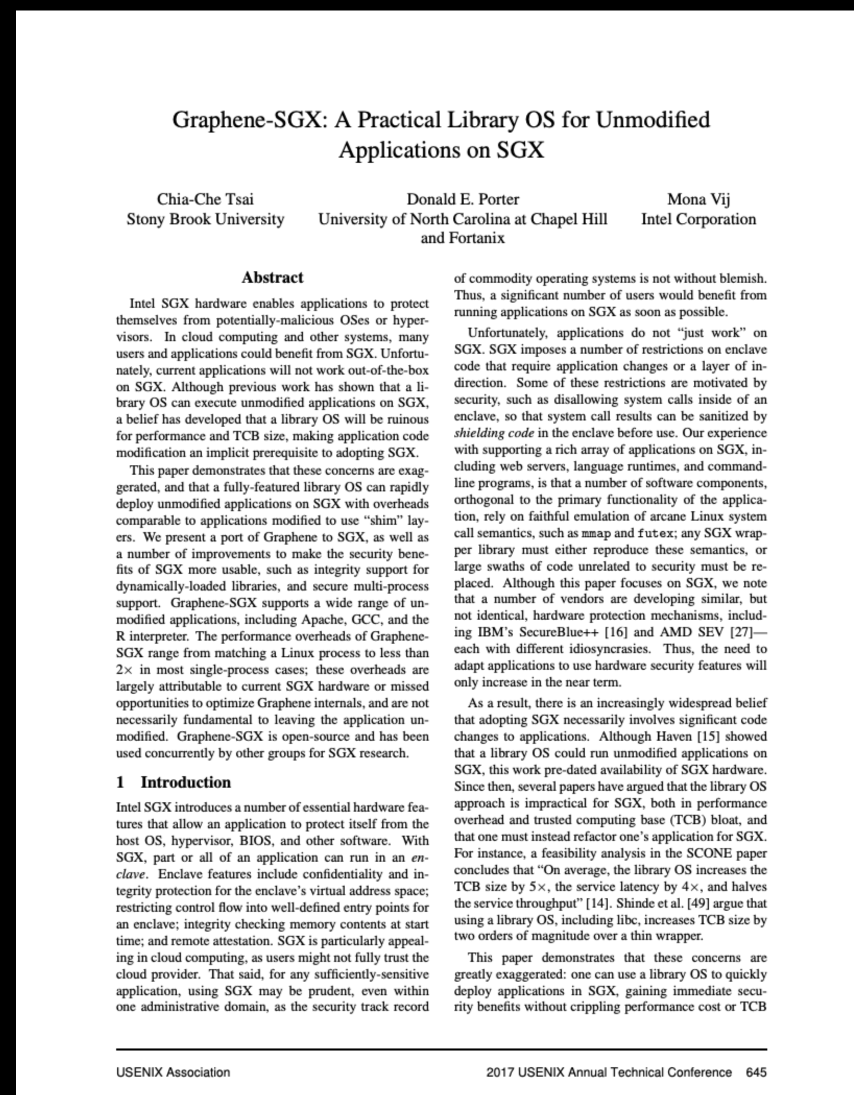
libOSes

Idea: Implement a small “OS” inside the enclave



Graphene-SGX

A libOS for Intel SGX that supports *some* services



Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX

Chia-Che Tsai
Stony Brook University

Donald E. Porter
University of North Carolina at Chapel Hill and Fortanix

Mona Vij
Intel Corporation

Abstract

Intel SGX hardware enables applications to protect themselves from potentially-malicious OSes or hypervisors. In cloud computing and other systems, many users and applications could benefit from SGX. Unfortunately, current applications will not work out-of-the-box on SGX. Although previous work has shown that a library OS can execute unmodified applications on SGX, a belief has developed that a library OS will be ruinous for performance and TCB size, making application code modification an implicit prerequisite to adopting SGX.

This paper demonstrates that these concerns are exaggerated, and that a fully-featured library OS can rapidly deploy unmodified applications on SGX with overheads comparable to applications modified to use "shim" layers. We present a port of Graphene to SGX, as well as a number of improvements to make the security benefits of SGX more usable, such as integrity support for dynamically-loaded libraries, and secure multi-process support. Graphene-SGX supports a wide range of unmodified applications, including Apache, GCC, and the R interpreter. The performance overheads of Graphene-SGX range from matching a Linux process to less than 2× in most single-process cases; these overheads are largely attributable to current SGX hardware or missed opportunities to optimize Graphene internals, and are not necessarily fundamental to leaving the application unmodified. Graphene-SGX is open-source and has been used concurrently by other groups for SGX research.

1 Introduction

Intel SGX introduces a number of essential hardware features that allow an application to protect itself from the host OS, hypervisor, BIOS, and other software. With SGX, part or all of an application can run in an *enclave*. Enclave features include confidentiality and integrity protection for the enclave's virtual address space; restricting control flow into well-defined entry points for an enclave; integrity checking memory contents at start time; and remote attestation. SGX is particularly appealing in cloud computing, as users might not fully trust the cloud provider. That said, for any sufficiently-sensitive application, using SGX may be prudent, even within one administrative domain, as the security track record of commodity operating systems is not without blemish. Thus, a significant number of users would benefit from running applications on SGX as soon as possible.

Unfortunately, applications do not "just work" on SGX. SGX imposes a number of restrictions on enclave code that require application changes or a layer of indirection. Some of these restrictions are motivated by security, such as disallowing system calls inside of an enclave, so that system call results can be sanitized by *shielding code* in the enclave before use. Our experience with supporting a rich array of applications on SGX, including web servers, language runtimes, and command-line programs, is that a number of software components, orthogonal to the primary functionality of the application, rely on faithful emulation of arcane Linux system call semantics, such as `mmap` and `futex`; any SGX wrapper library must either reproduce these semantics, or large swaths of code unrelated to security must be replaced. Although this paper focuses on SGX, we note that a number of vendors are developing similar, but not identical, hardware protection mechanisms, including IBM's SecureBlue++ [16] and AMD SEV [27]—each with different idiosyncrasies. Thus, the need to adapt applications to use hardware security features will only increase in the near term.

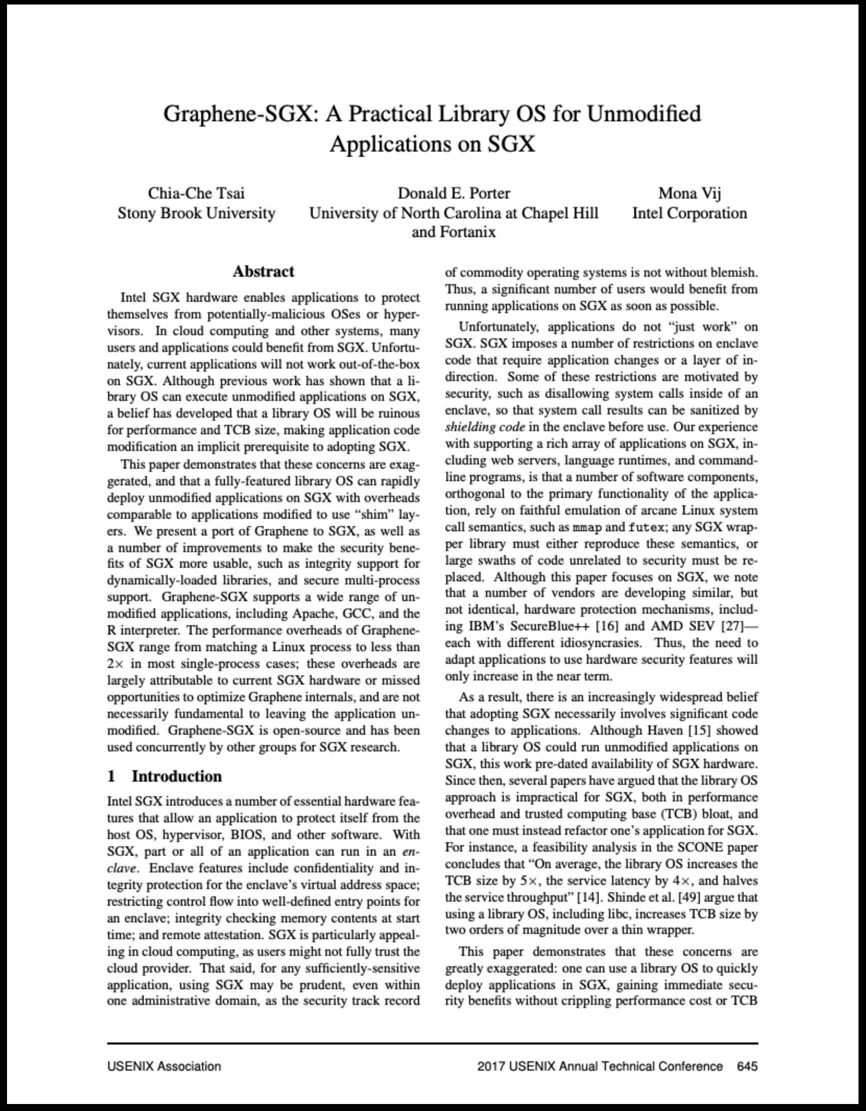
As a result, there is an increasingly widespread belief that adopting SGX necessarily involves significant code changes to applications. Although Haven [15] showed that a library OS could run unmodified applications on SGX, this work pre-dated availability of SGX hardware. Since then, several papers have argued that the library OS approach is impractical for SGX, both in performance overhead and trusted computing base (TCB) bloat, and that one must instead refactor one's application for SGX. For instance, a feasibility analysis in the SCONE paper concludes that "On average, the library OS increases the TCB size by 5×, the service latency by 4×, and halves the service throughput" [14]. Shinde et al. [49] argue that using a library OS, including libc, increases TCB size by two orders of magnitude over a thin wrapper.

This paper demonstrates that these concerns are greatly exaggerated: one can use a library OS to quickly deploy applications in SGX, gaining immediate security benefits without crippling performance cost or TCB

Tsai et al., ATC 2017

Graphene-SGX

A libOS for Intel SGX that supports *some* services



Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX

Chia-Che Tsai, Stony Brook University
Donald E. Porter, University of North Carolina at Chapel Hill and Fortanix
Mona Vij, Intel Corporation

Abstract

Intel SGX hardware enables applications to protect themselves from potentially-malicious OSes or hypervisors. In cloud computing and other systems, many users and applications could benefit from SGX. Unfortunately, current applications will not work out-of-the-box on SGX. Although previous work has shown that a library OS can execute unmodified applications on SGX, a belief has developed that a library OS will be ruinous for performance and TCB size, making application code modification an implicit prerequisite to adopting SGX.

This paper demonstrates that these concerns are exaggerated, and that a fully-featured library OS can rapidly deploy unmodified applications on SGX with overheads comparable to applications modified to use "shim" layers. We present a port of Graphene to SGX, as well as a number of improvements to make the security benefits of SGX more usable, such as integrity support for dynamically-loaded libraries, and secure multi-process support. Graphene-SGX supports a wide range of unmodified applications, including Apache, GCC, and the R interpreter. The performance overheads of Graphene-SGX range from matching a Linux process to less than 2× in most single-process cases; these overheads are largely attributable to current SGX hardware or missed opportunities to optimize Graphene internals, and are not necessarily fundamental to leaving the application unmodified. Graphene-SGX is open-source and has been used concurrently by other groups for SGX research.

1 Introduction

Intel SGX introduces a number of essential hardware features that allow an application to protect itself from the host OS, hypervisor, BIOS, and other software. With SGX, part or all of an application can run in an *enclave*. Enclave features include confidentiality and integrity protection for the enclave's virtual address space; restricting control flow into well-defined entry points for an enclave; integrity checking memory contents at start time; and remote attestation. SGX is particularly appealing in cloud computing, as users might not fully trust the cloud provider. That said, for any sufficiently-sensitive application, using SGX may be prudent, even within one administrative domain, as the security track record of commodity operating systems is not without blemish. Thus, a significant number of users would benefit from running applications on SGX as soon as possible.

Unfortunately, applications do not "just work" on SGX. SGX imposes a number of restrictions on enclave code that require application changes or a layer of indirection. Some of these restrictions are motivated by security, such as disallowing system calls inside of an enclave, so that system call results can be sanitized by *shielding code* in the enclave before use. Our experience with supporting a rich array of applications on SGX, including web servers, language runtimes, and command-line programs, is that a number of software components, orthogonal to the primary functionality of the application, rely on faithful emulation of arcane Linux system call semantics, such as `mmap` and `futex`; any SGX wrapper library must either reproduce these semantics, or large swaths of code unrelated to security must be replaced. Although this paper focuses on SGX, we note that a number of vendors are developing similar, but not identical, hardware protection mechanisms, including IBM's SecureBlue++ [16] and AMD SEV [27]—each with different idiosyncrasies. Thus, the need to adapt applications to use hardware security features will only increase in the near term.

As a result, there is an increasingly widespread belief that adopting SGX necessarily involves significant code changes to applications. Although Haven [15] showed that a library OS could run unmodified applications on SGX, this work pre-dated availability of SGX hardware. Since then, several papers have argued that the library OS approach is impractical for SGX, both in performance overhead and trusted computing base (TCB) bloat, and that one must instead refactor one's application for SGX. For instance, a feasibility analysis in the SCONE paper concludes that "On average, the library OS increases the TCB size by 5×, the service latency by 4×, and halves the service throughput" [14]. Shinde et al. [49] argue that using a library OS, including libc, increases TCB size by two orders of magnitude over a thin wrapper.

This paper demonstrates that these concerns are greatly exaggerated: one can use a library OS to quickly deploy applications in SGX, gaining immediate security benefits without crippling performance cost or TCB

USENIX Association — 2017 USENIX Annual Technical Conference 645

Tsai et al., ATC 2017

Graphene's supported services:

- *fork*
- *exec*
- *pipes, signals, semaphores*

Graphene-SGX

A libOS for Intel SGX that supports *some* services

What constitutes a CDN?

Multiple tenants	*Web server*
Needs disk	*Cache*
Needs plaintext	*Web Application Firewall*
Needs safe storage	*Key Server*

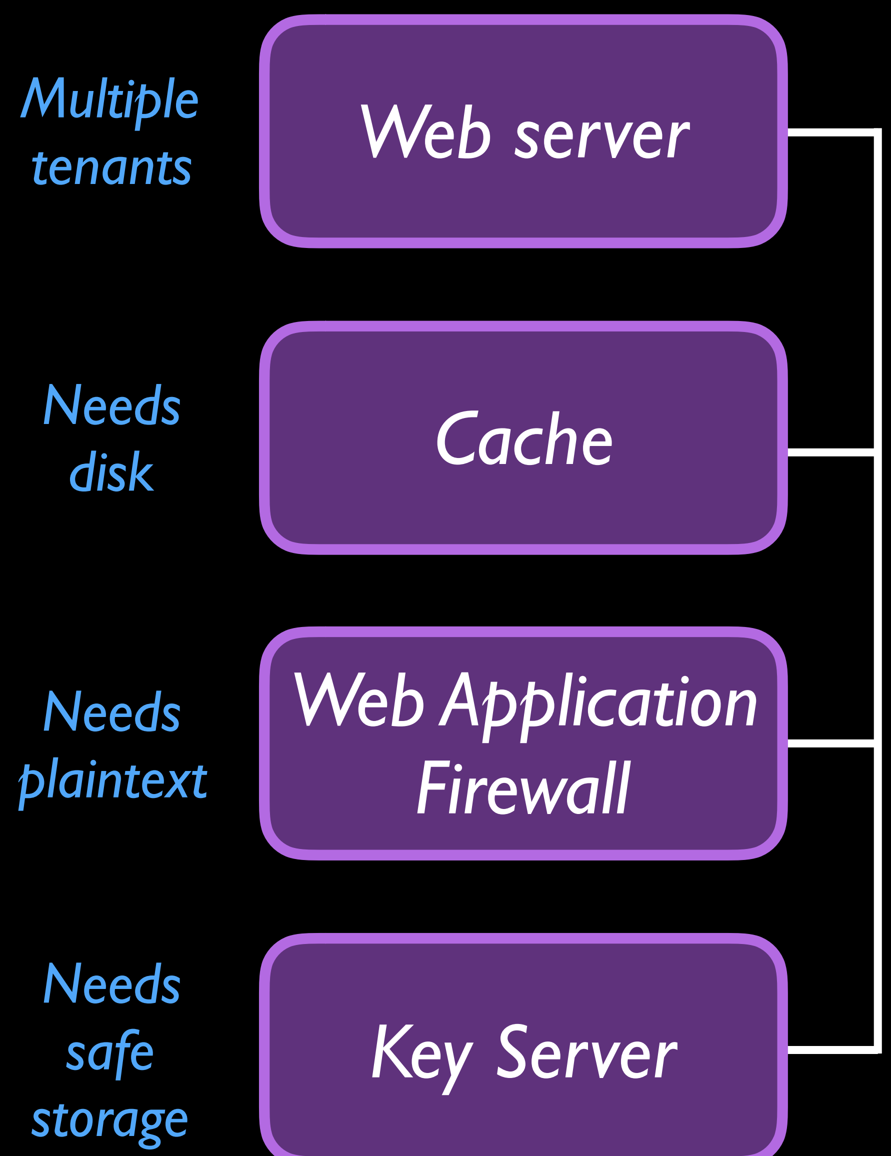
Graphene's supported services:

- ✓ *fork*
- ✓ *exec*
- ✓ *pipes, signals, semaphores*

Graphene-SGX

A libOS for Intel SGX that supports *some* services

What constitutes a CDN?



Graphene's supported services:

- ✓ *fork*
- ✓ *exec*
- ✓ *pipes, signals, semaphores*

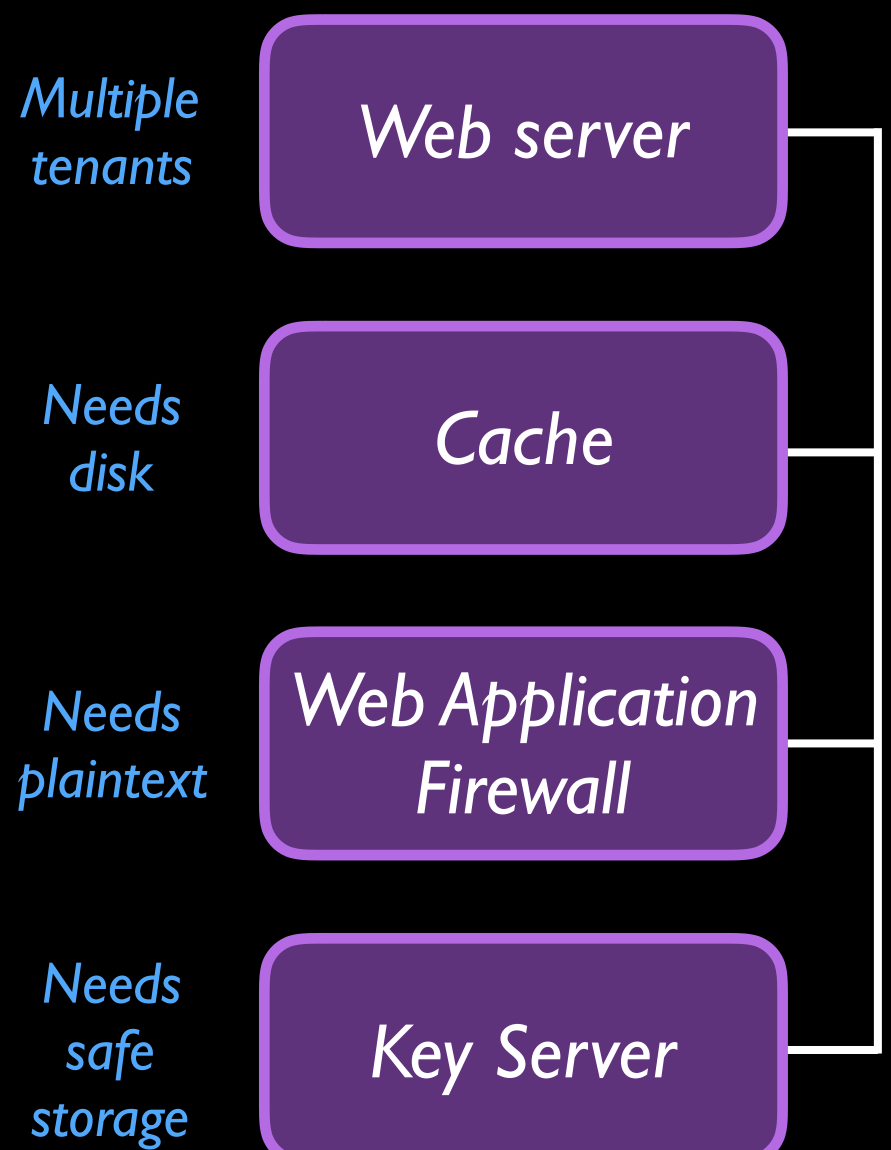
Also critical to a CDN:

- ✗ *Reading & writing files*
- ✗ *Shared memory*
- ✗ *Access to private keys*

Phoenix The first truly *keyless CDN*

Conclaves *Containers of enclaves*

What constitutes a CDN?



Graphene's supported services:

- ✓ *fork*
- ✓ *exec*
- ✓ *pipes, signals, semaphores*

Phoenix The first truly *keyless CDN*

Conclaves *Containers of enclaves*

What constitutes a CDN?

Multiple tenants	*Web server*
Needs disk	*Cache*
Needs plaintext	*Web Application Firewall*
Needs safe storage	*Key Server*

Graphene's supported services:

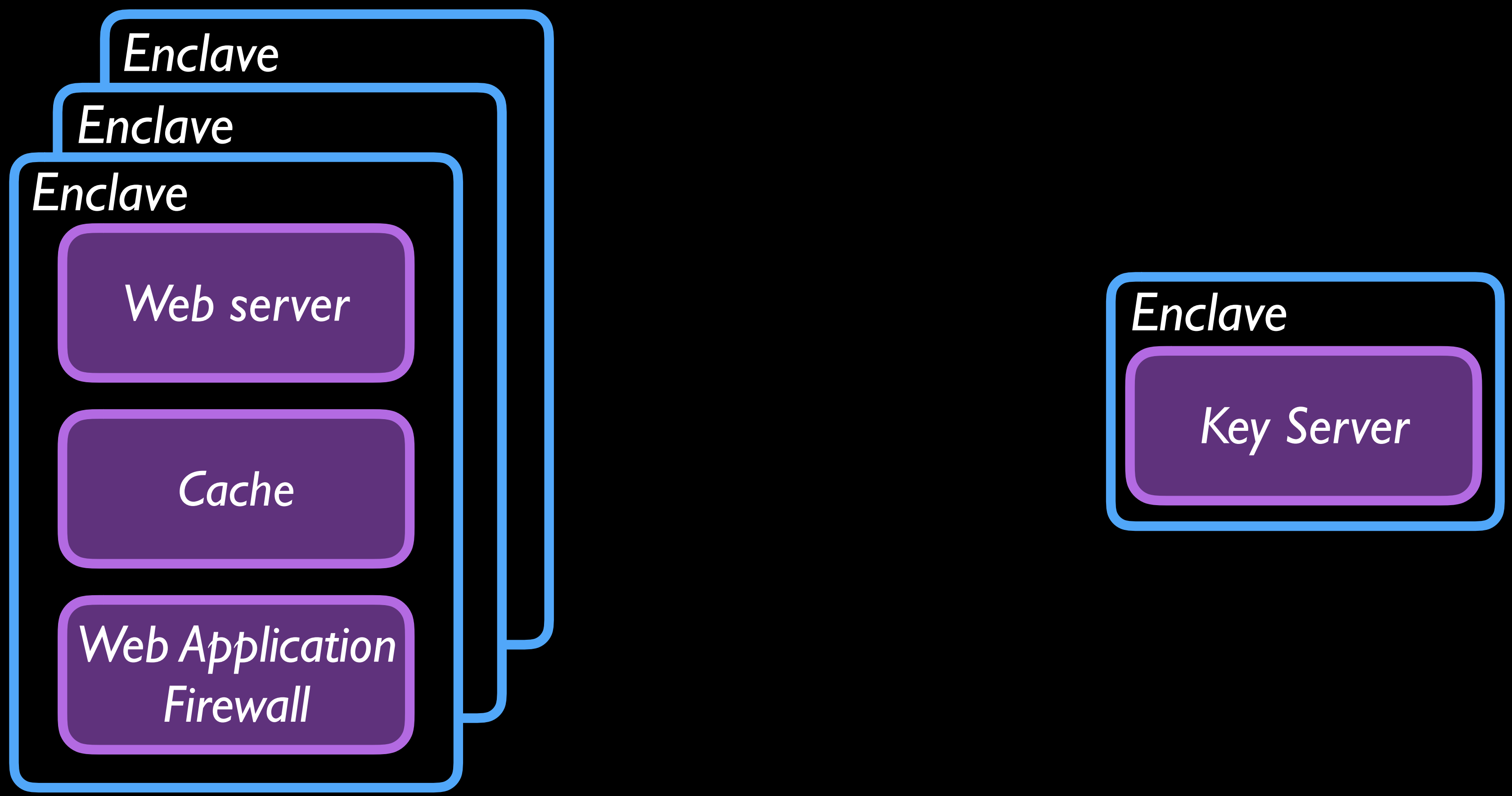
- ✓ *fork*
- ✓ *exec*
- ✓ *pipes, signals, semaphores*

Also critical to a CDN:

- ✗ *Reading & writing files*
- ✗ *Shared memory*
- ✗ *Access to private keys*

Phoenix The first truly *keyless CDN*

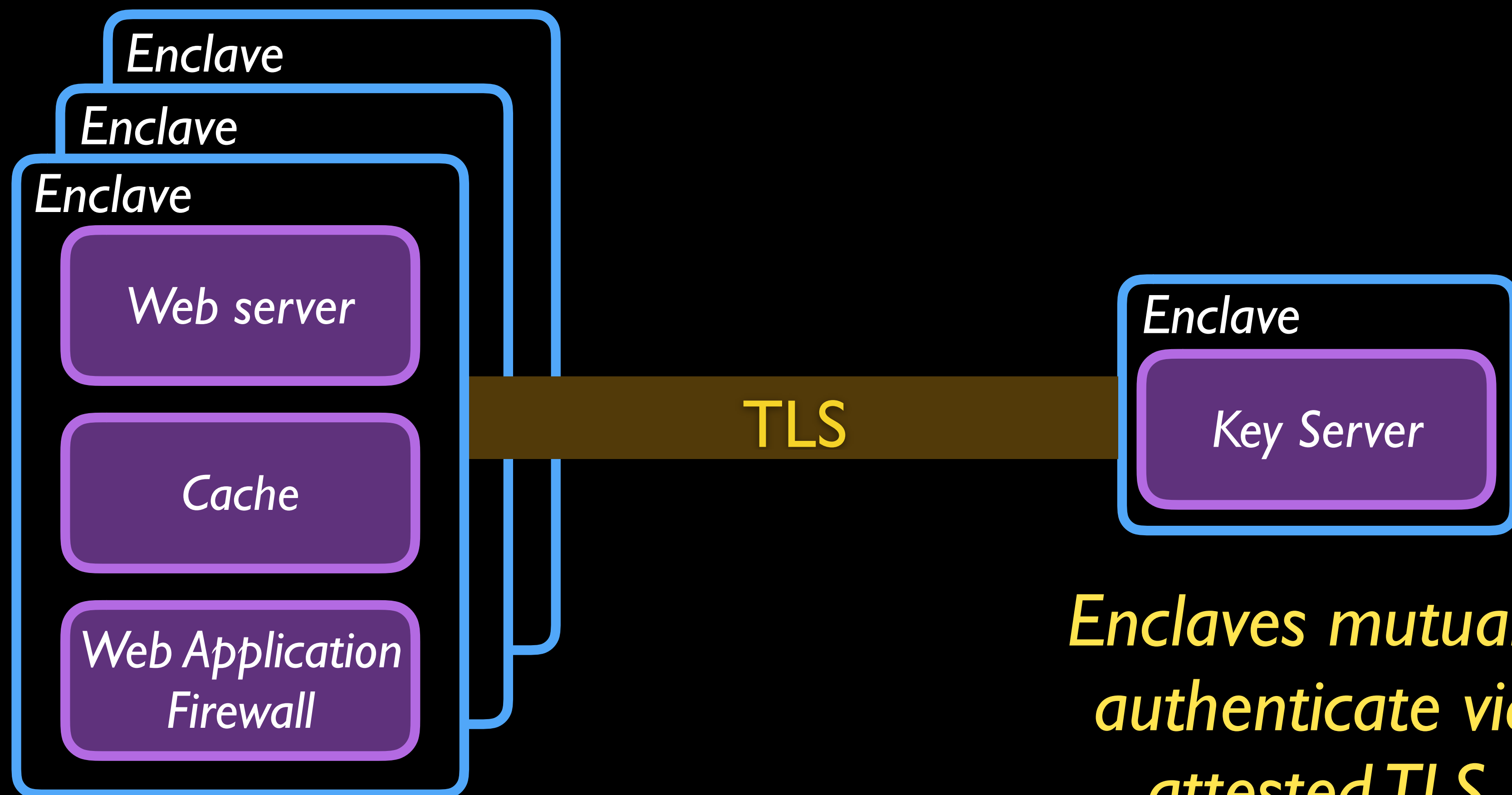
Conclaves *Containers of enclaves*



Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

Conclaves *Containers of enclaves*



Enclaves mutually authenticate via attested TLS

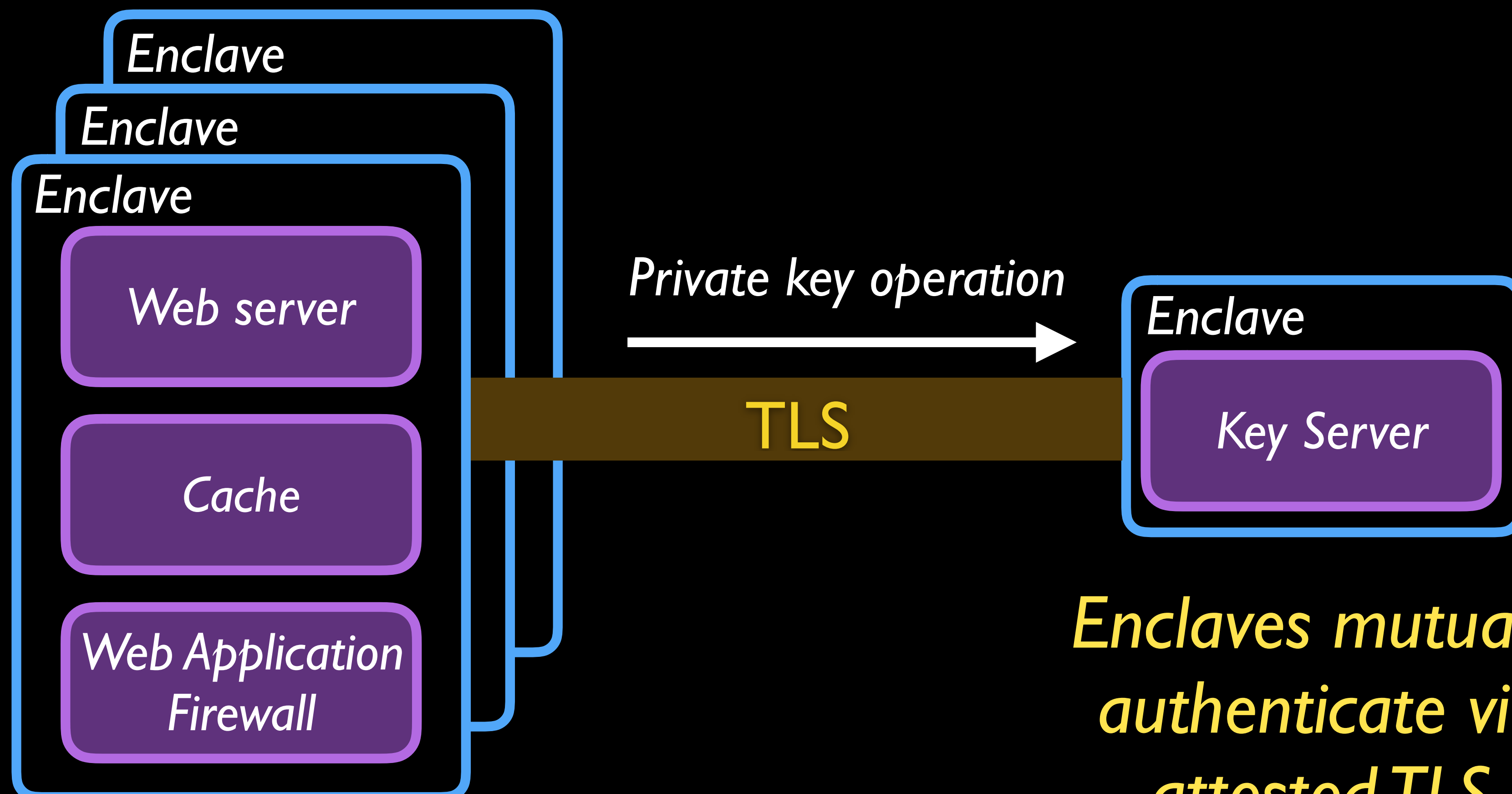
Knauth et al., 2018

Insight: Treat enclaves like a *distributed system*

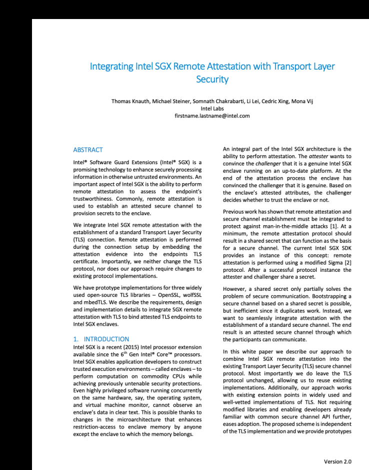
Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

Conclaves *Containers of enclaves*



Enclaves mutually authenticate via attested TLS


Integrating Intel SGX Remote Attestation with Transport Layer Security

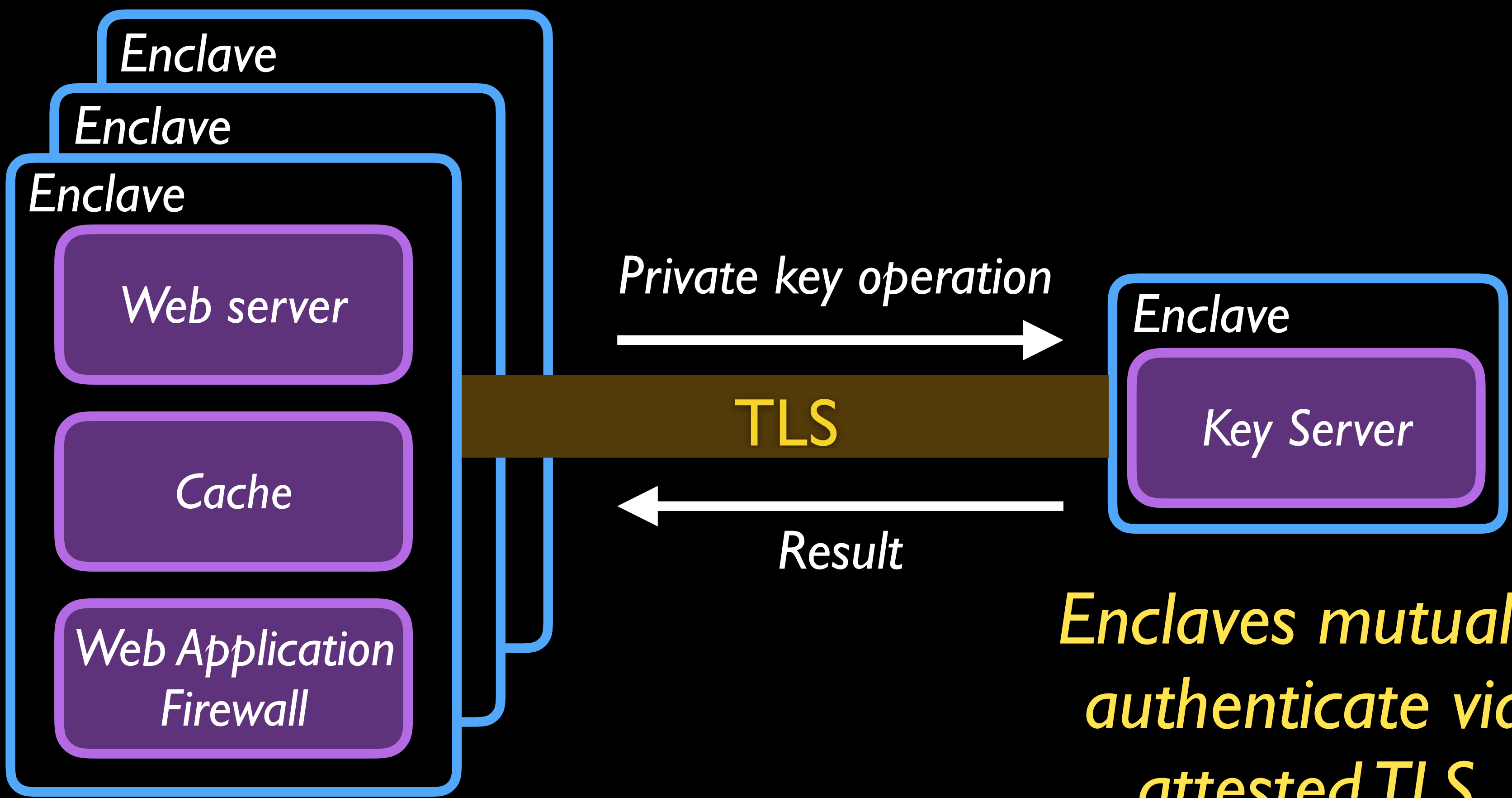
Knauth et al., 2018

Insight: Treat enclaves like a *distributed system*

Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

Conclaves *Containers of enclaves*



Enclaves mutually authenticate via attested TLS

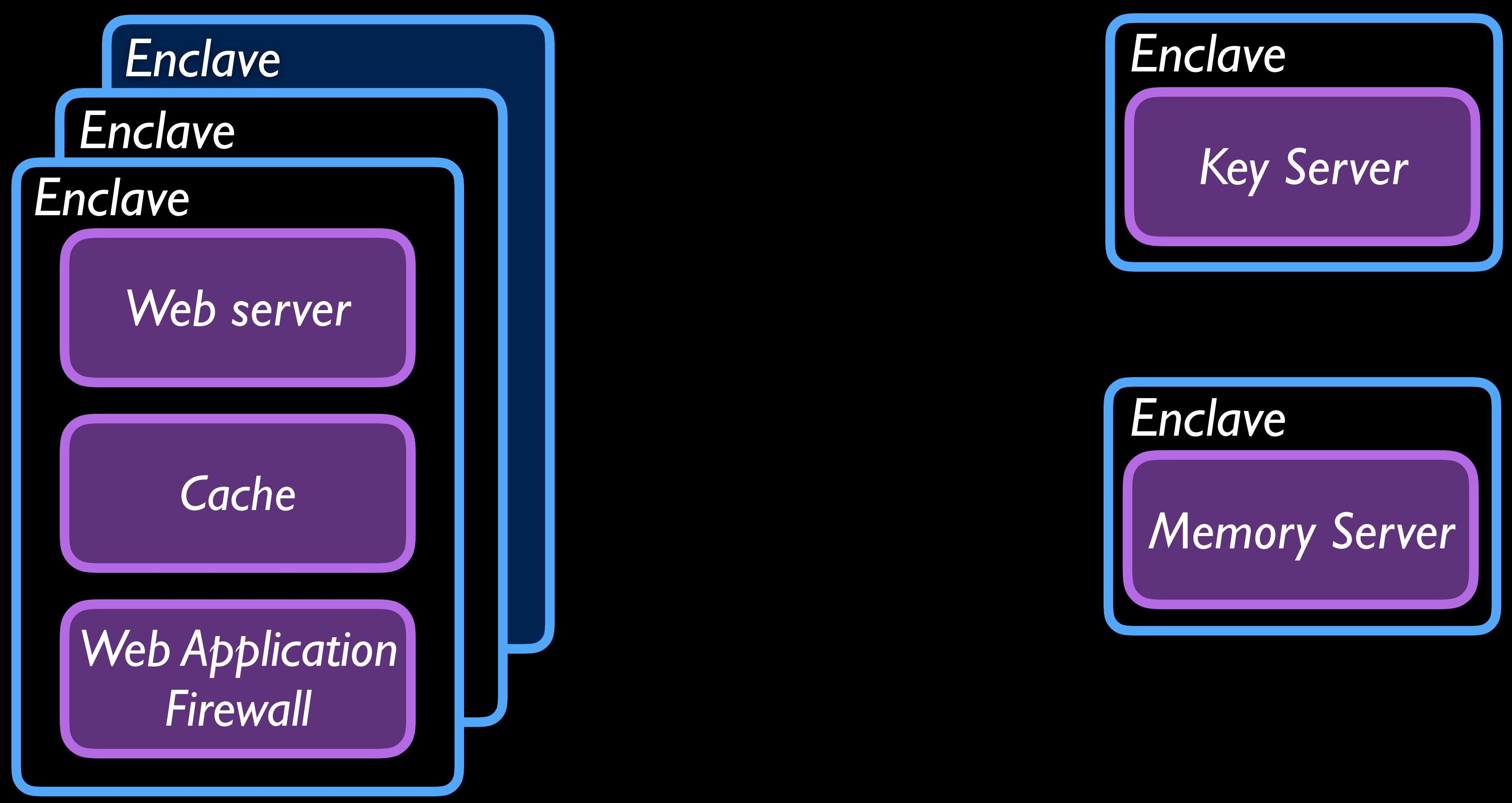

Integrating Intel SGX Remote Attestation with Transport Layer Security

Knauth et al., 2018

Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

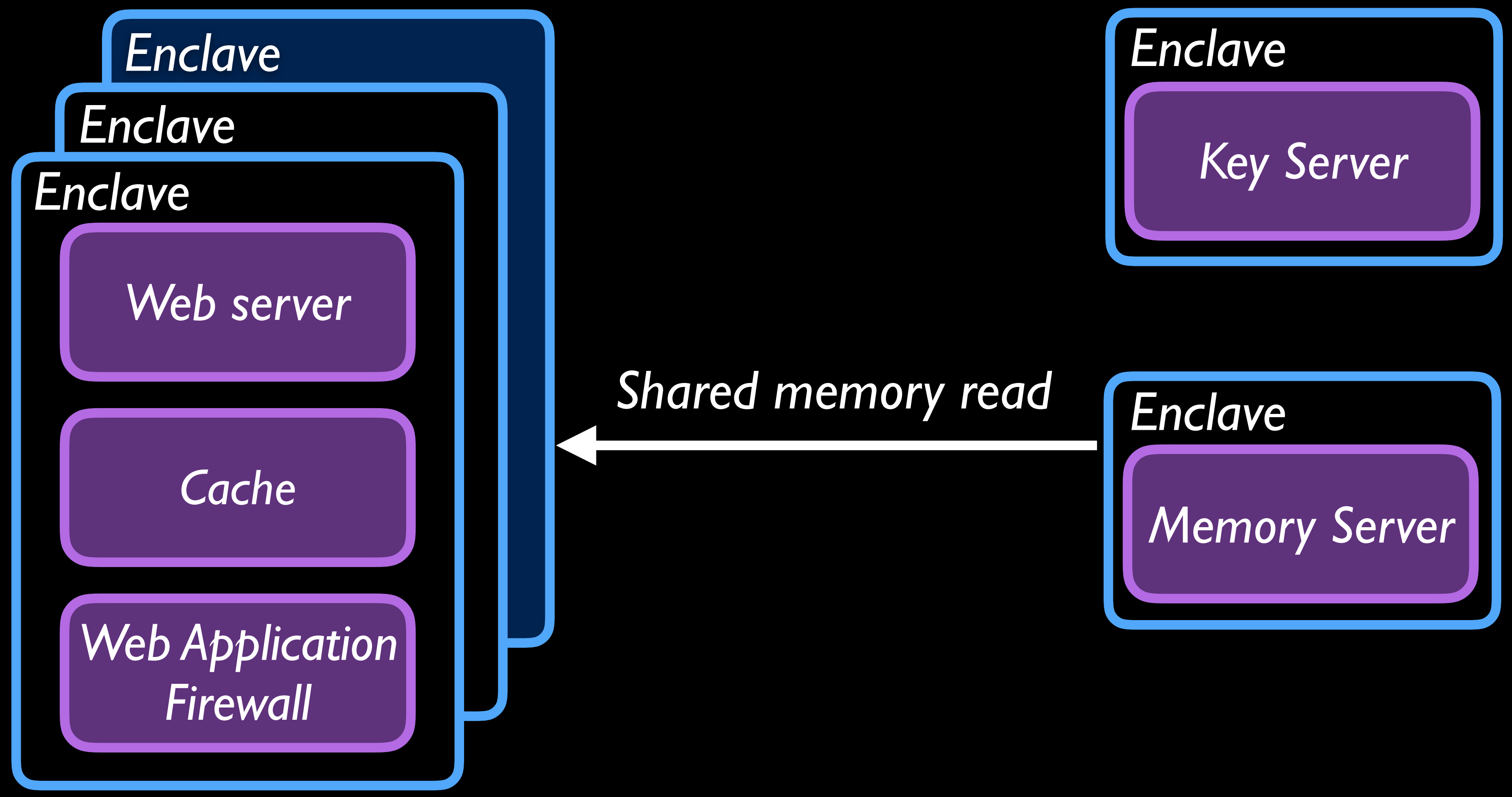
Conclaves *Containers of enclaves*



Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

Conclaves *Containers of enclaves*

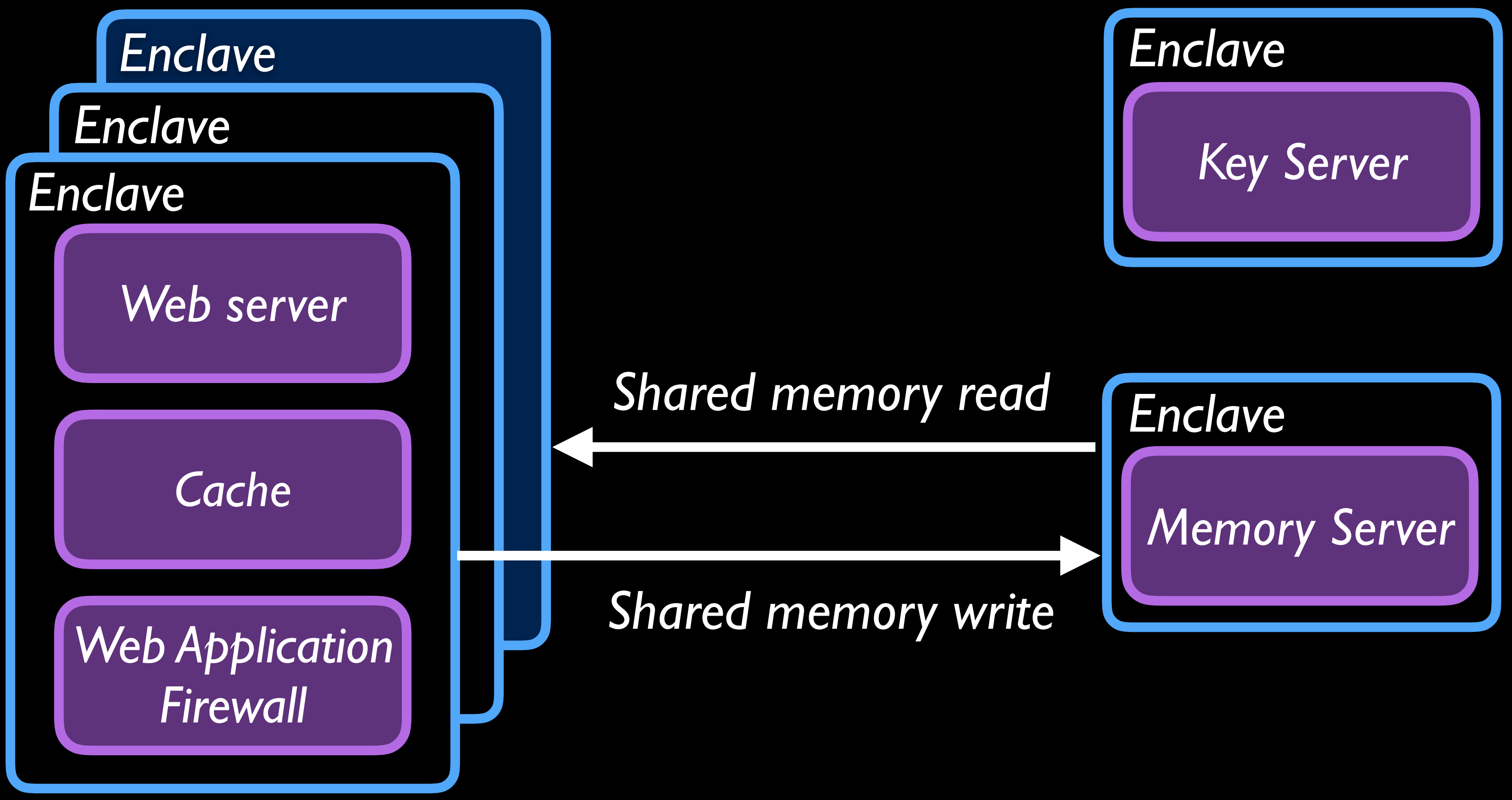


Insight: Treat enclaves like a *distributed system*

Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

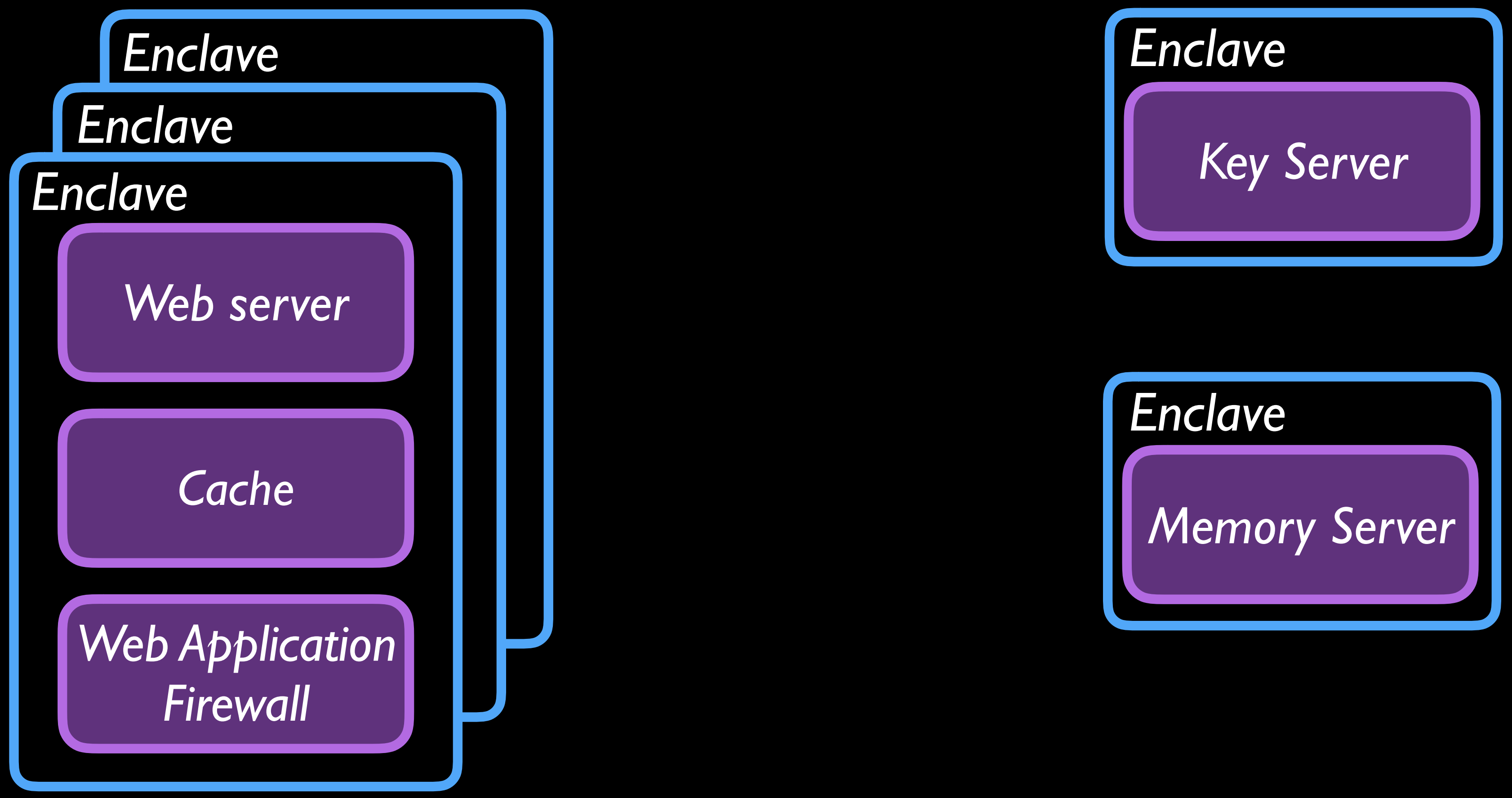
Conclaves *Containers of enclaves*



Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

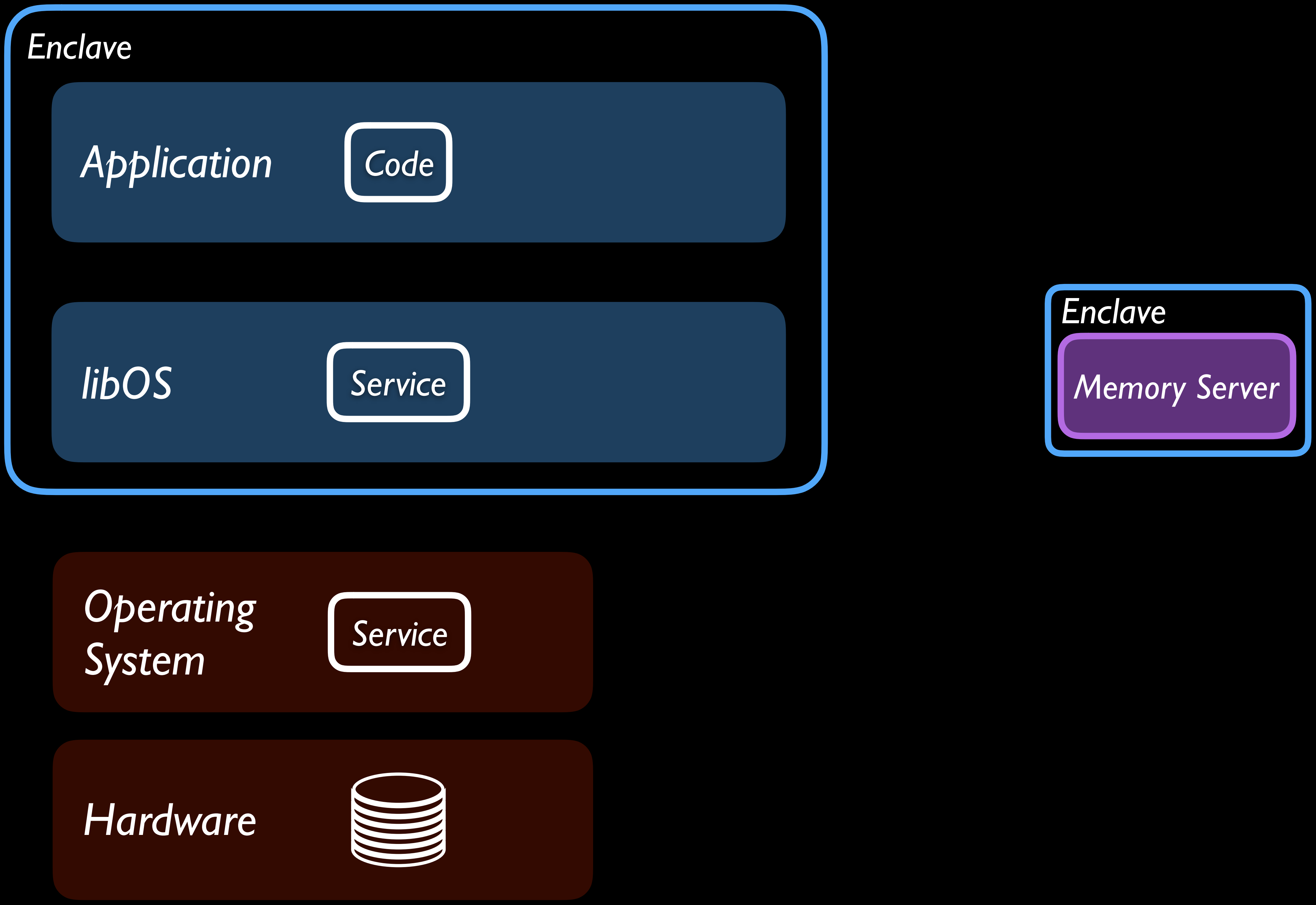
Phoenix The first truly *keyless CDN*

Conclaves *Containers of enclaves*

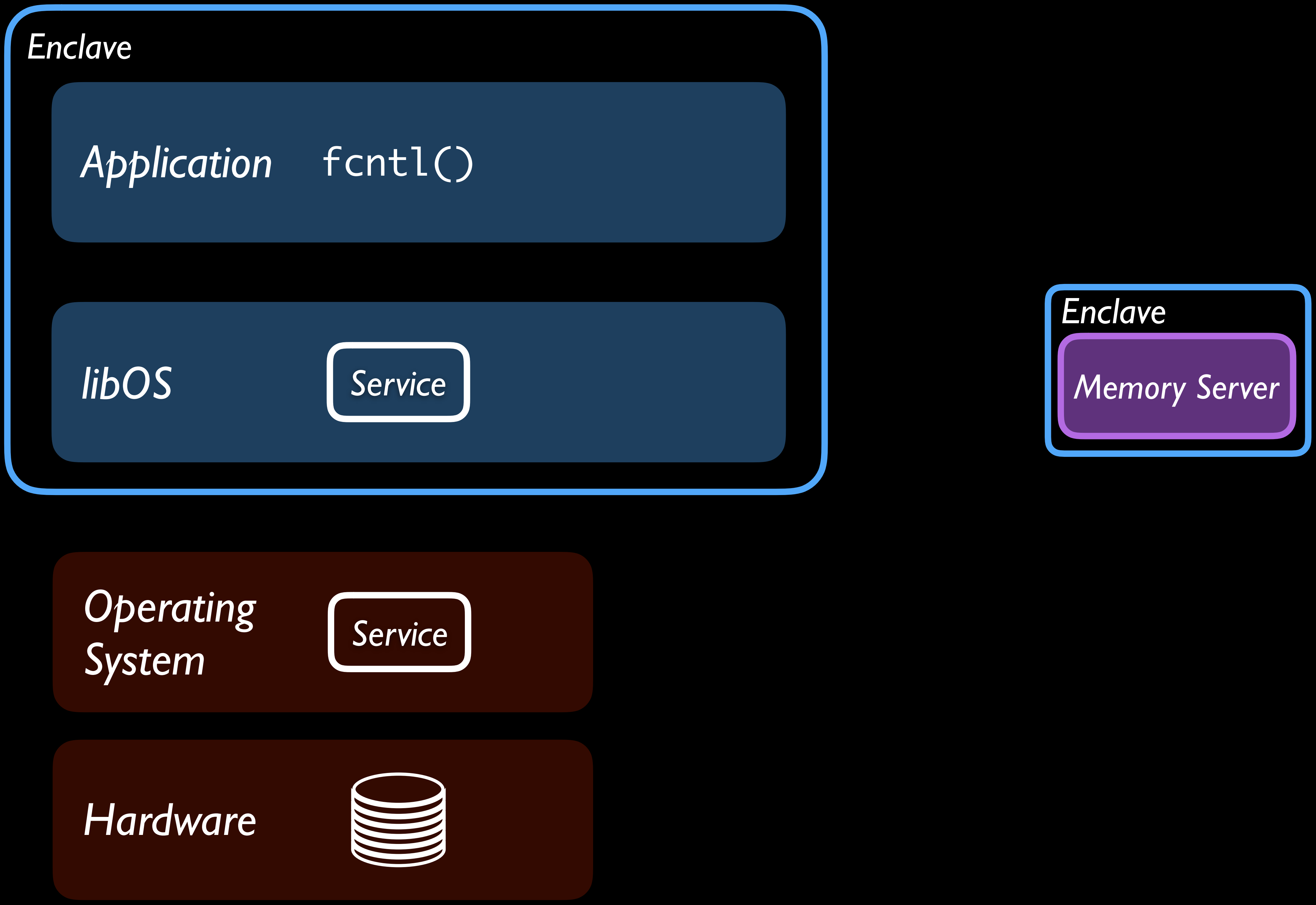


Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

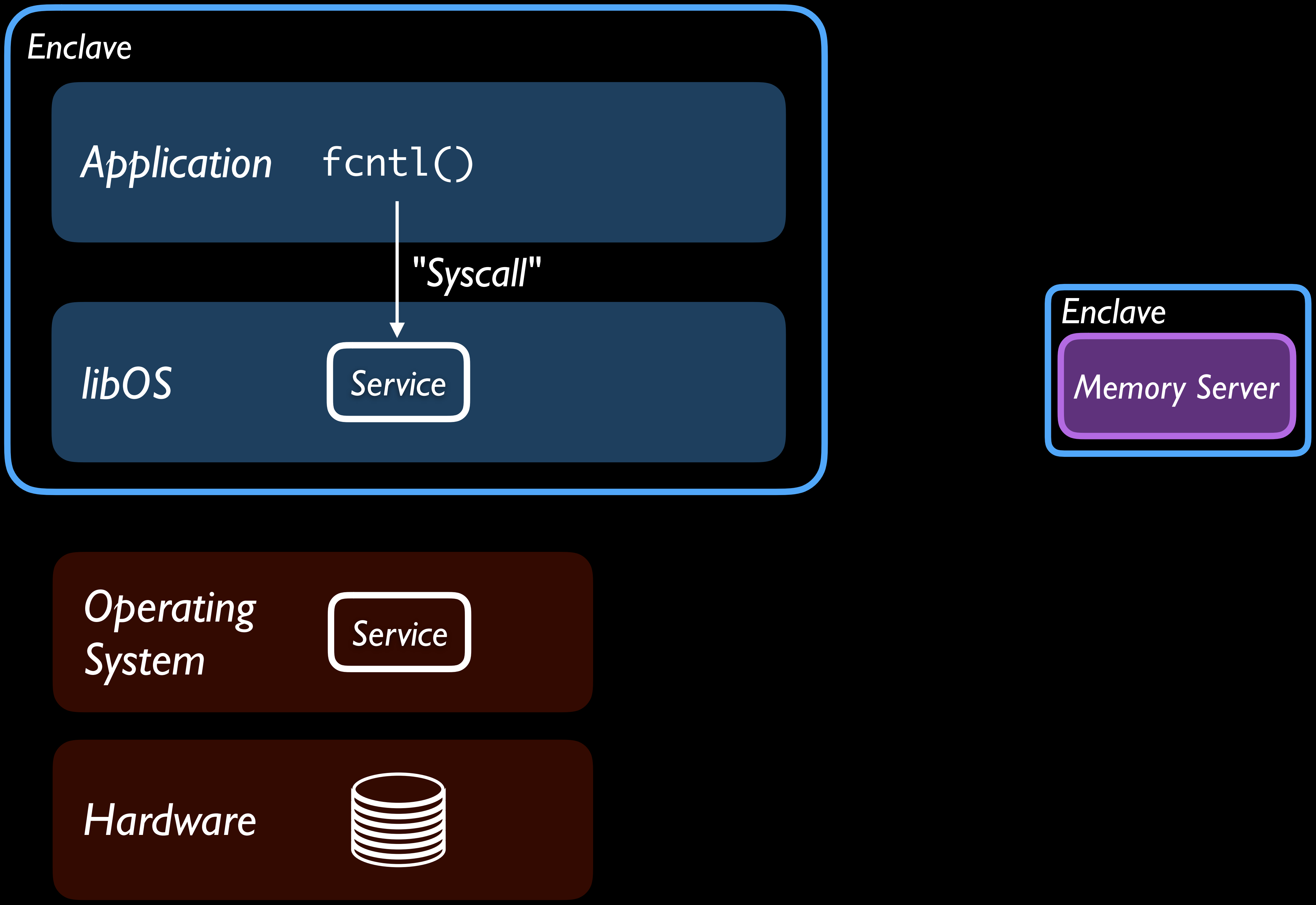
Conclaves Shared memory



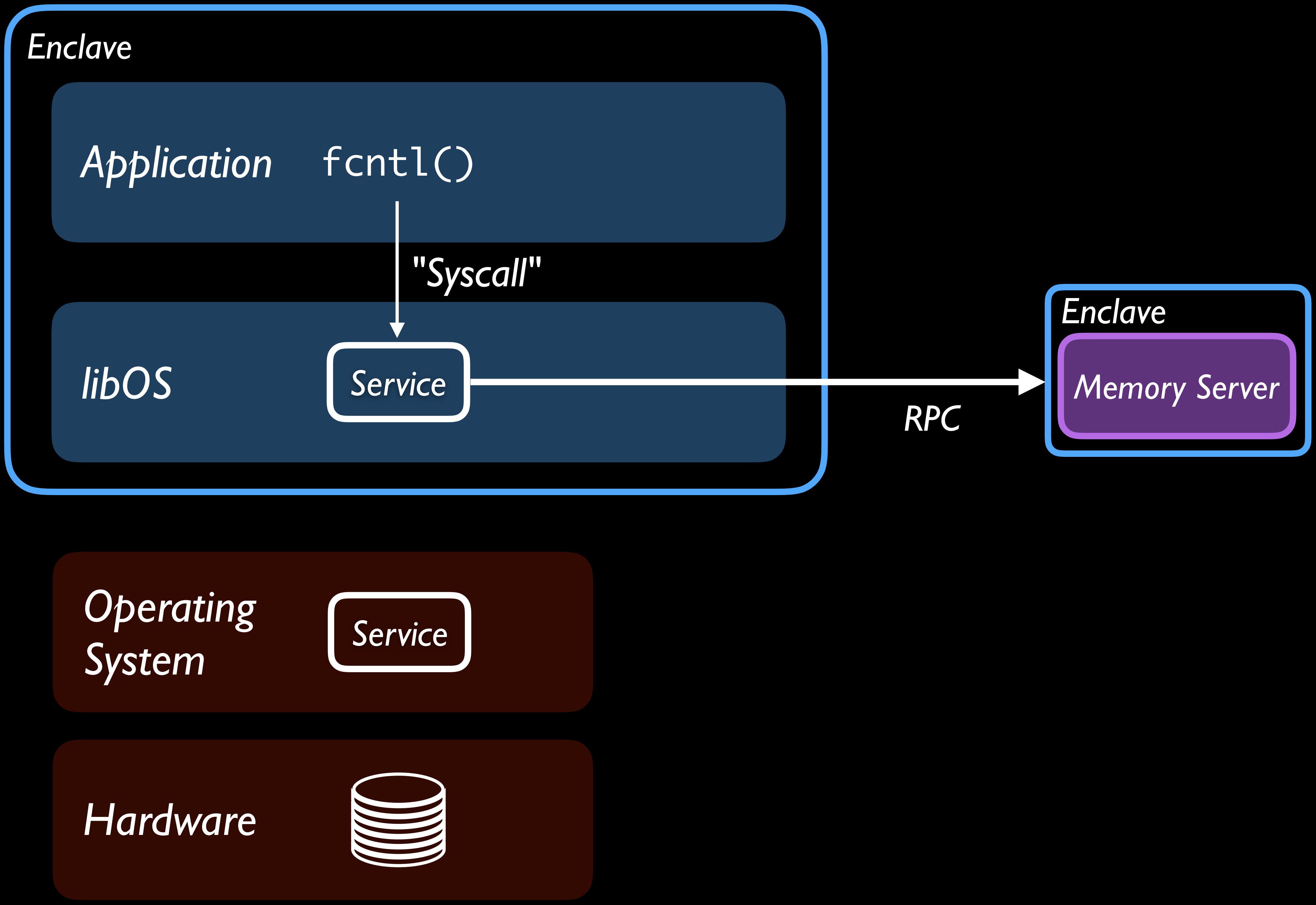
Conclaves Shared memory



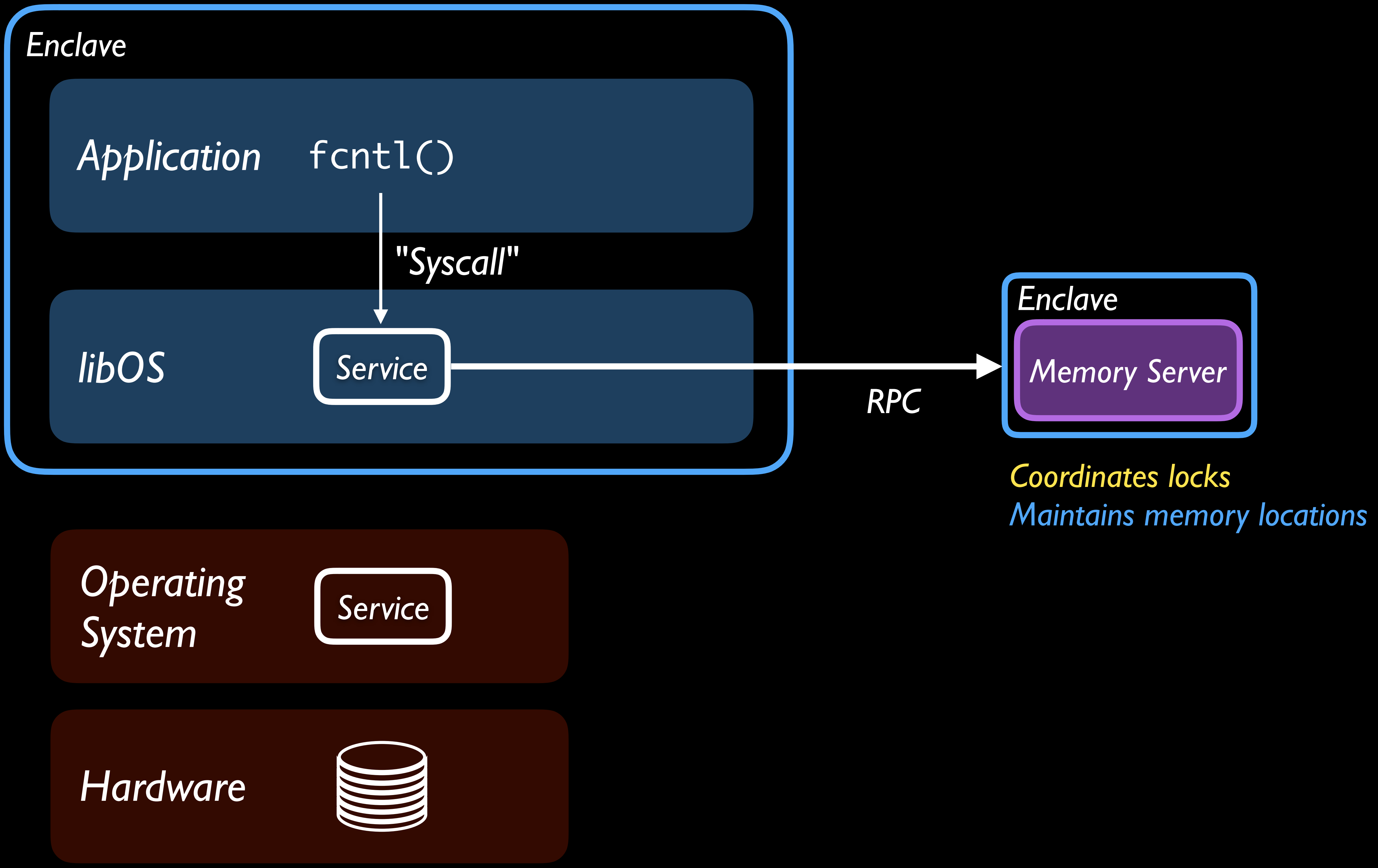
Conclaves Shared memory



Conclaves Shared memory

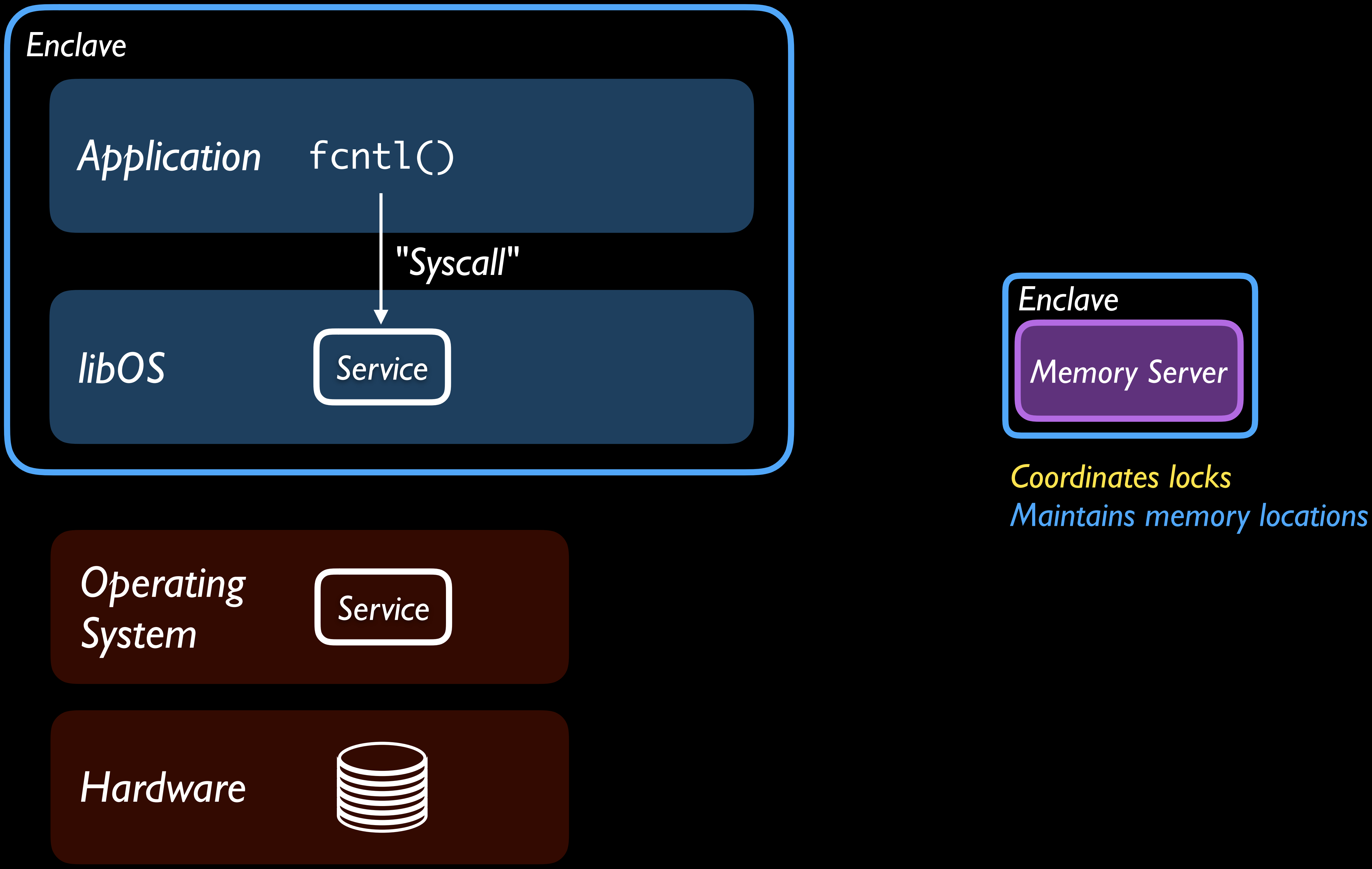


Conclaves Shared memory

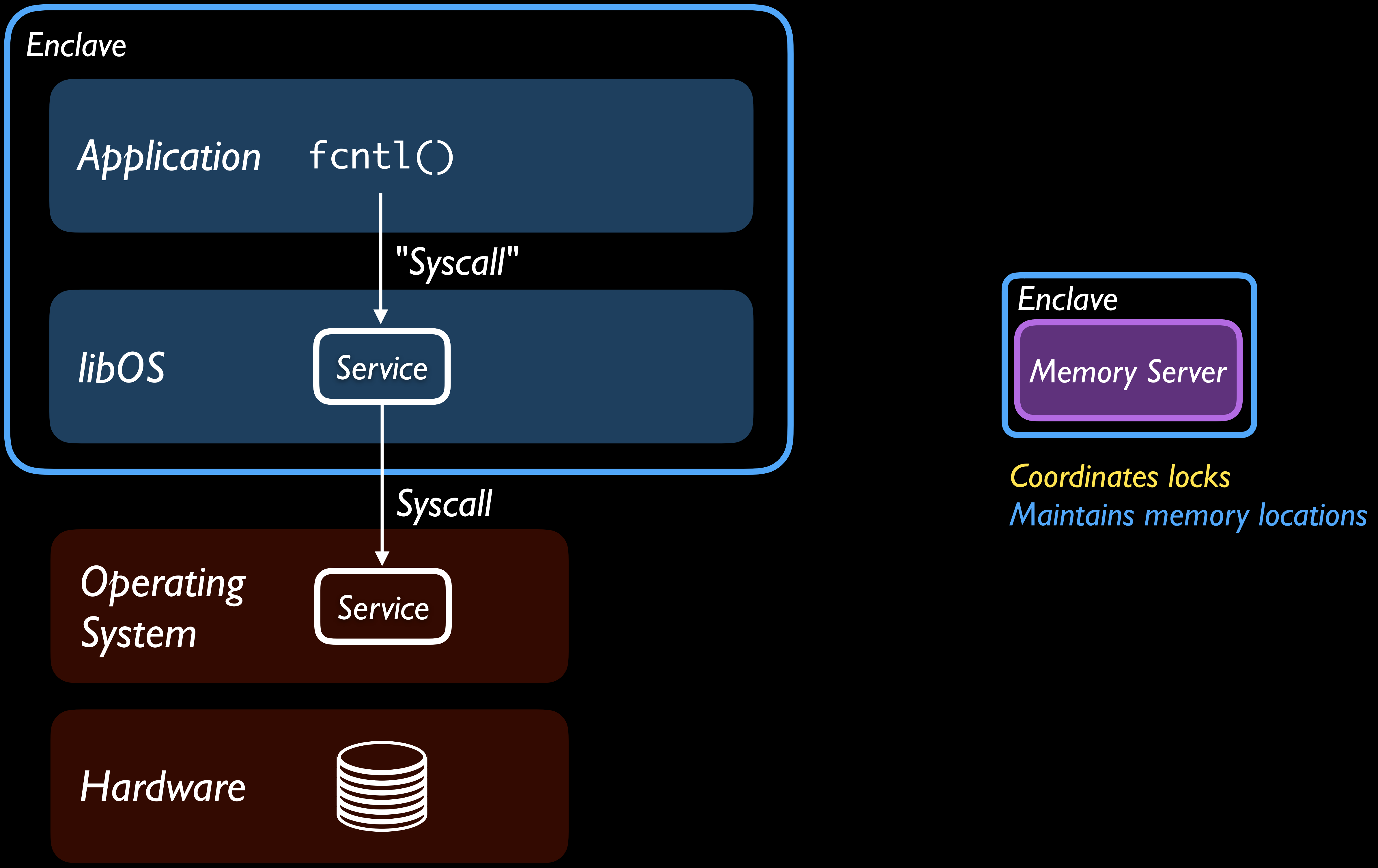


Conclaves

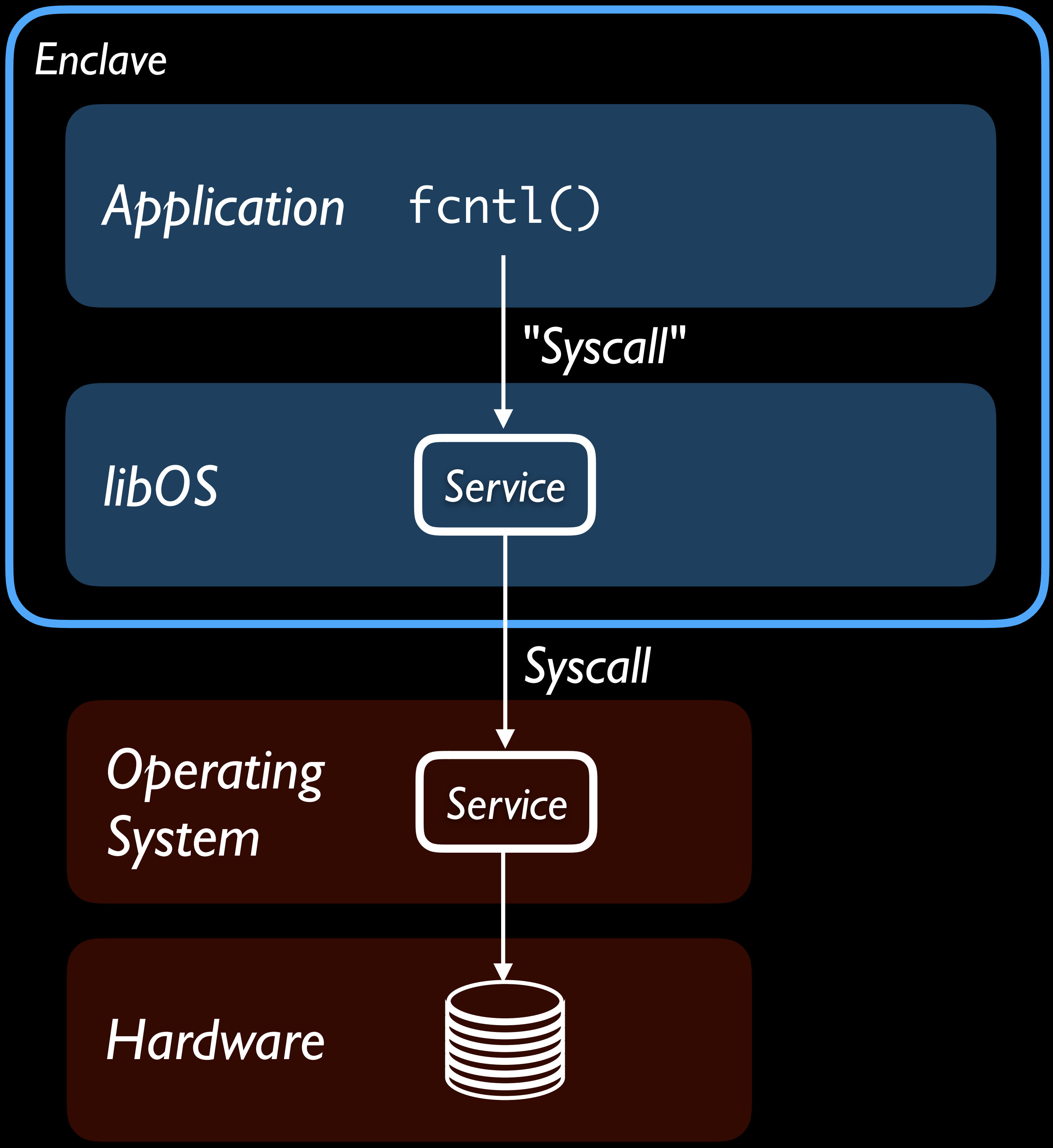
Shared memory

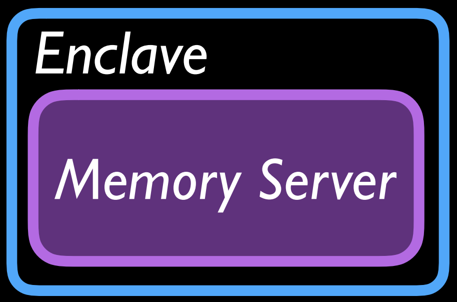


Conclaves Shared memory



Conclaves Shared memory

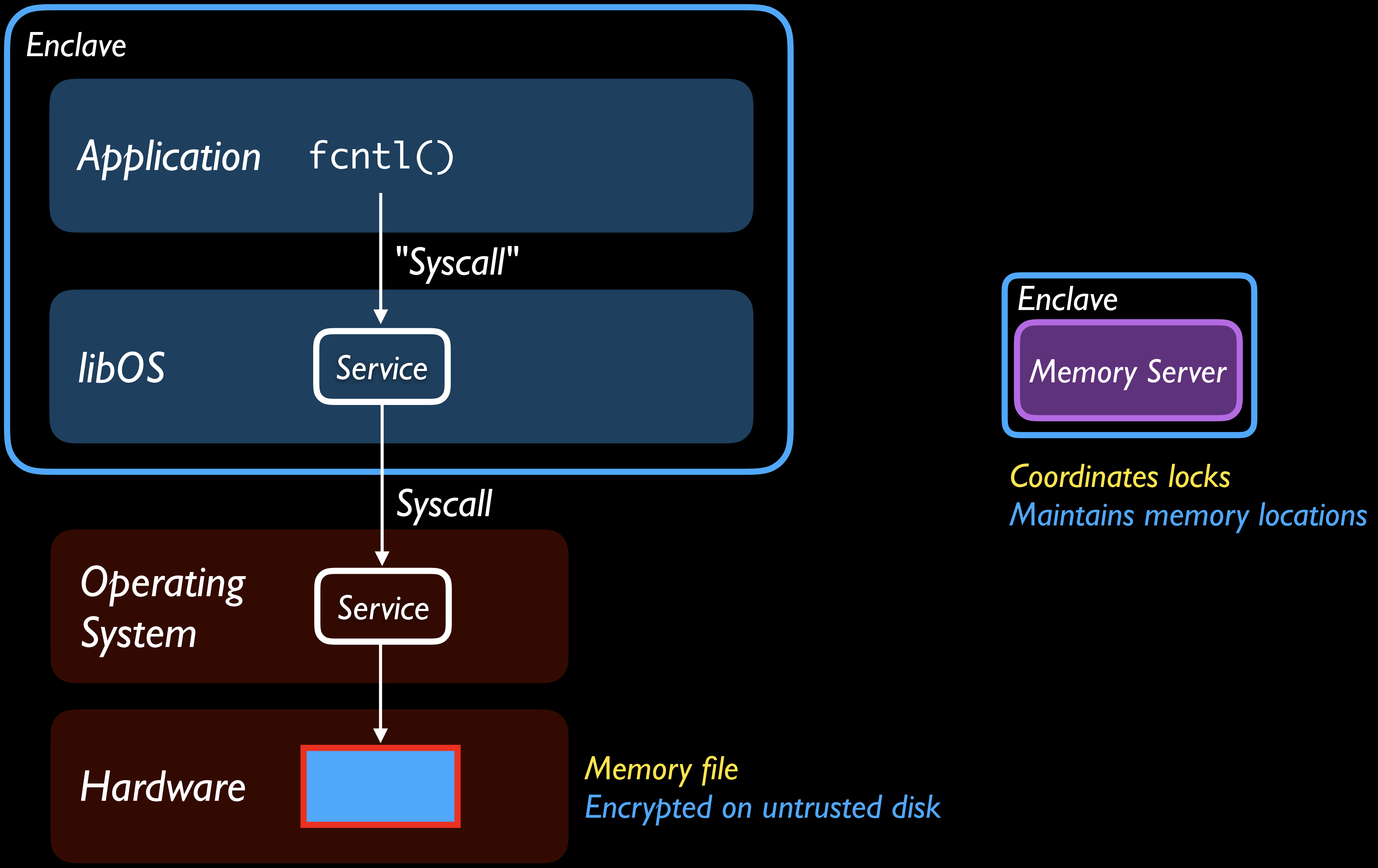




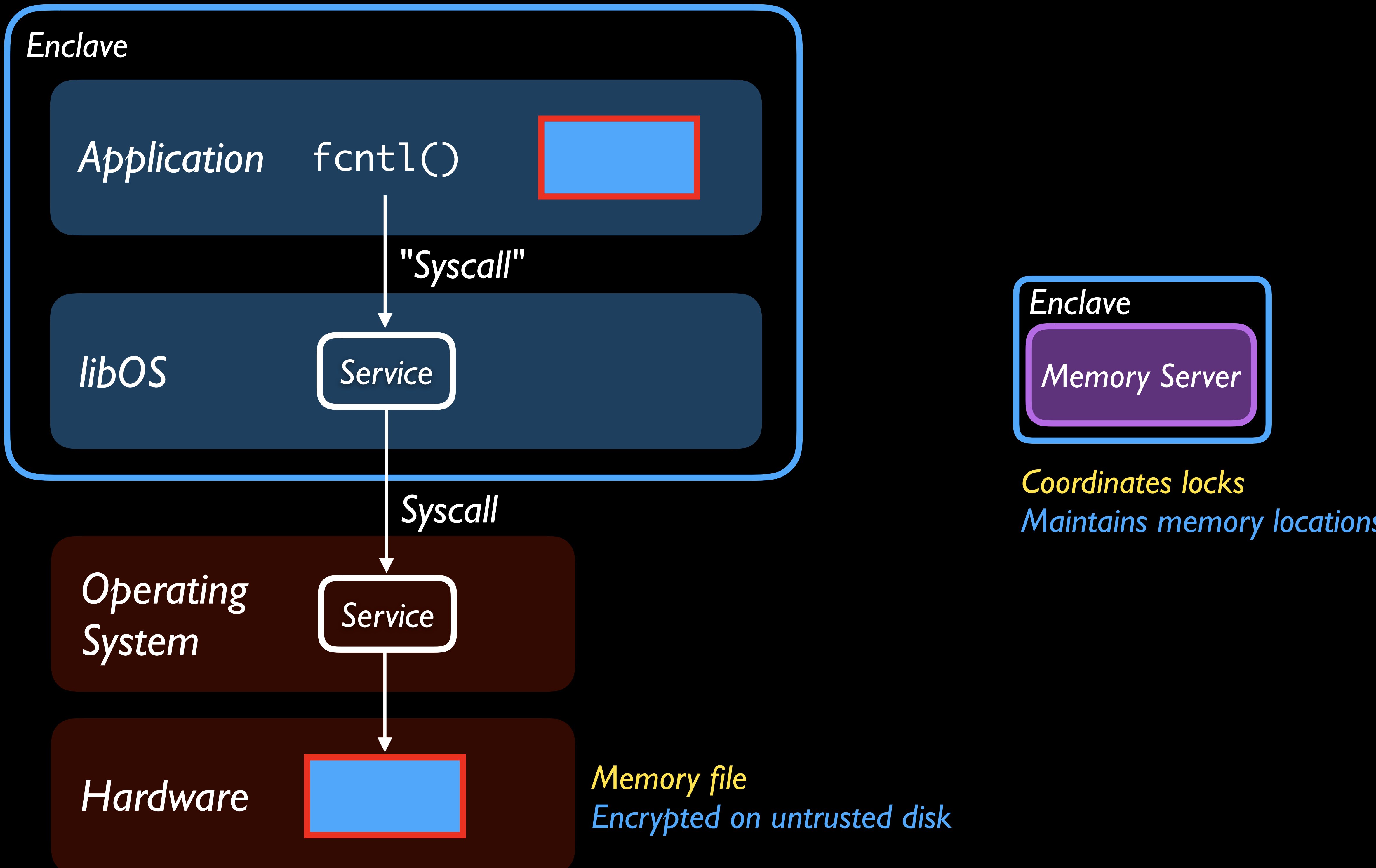
Coordinates locks

Maintains memory locations

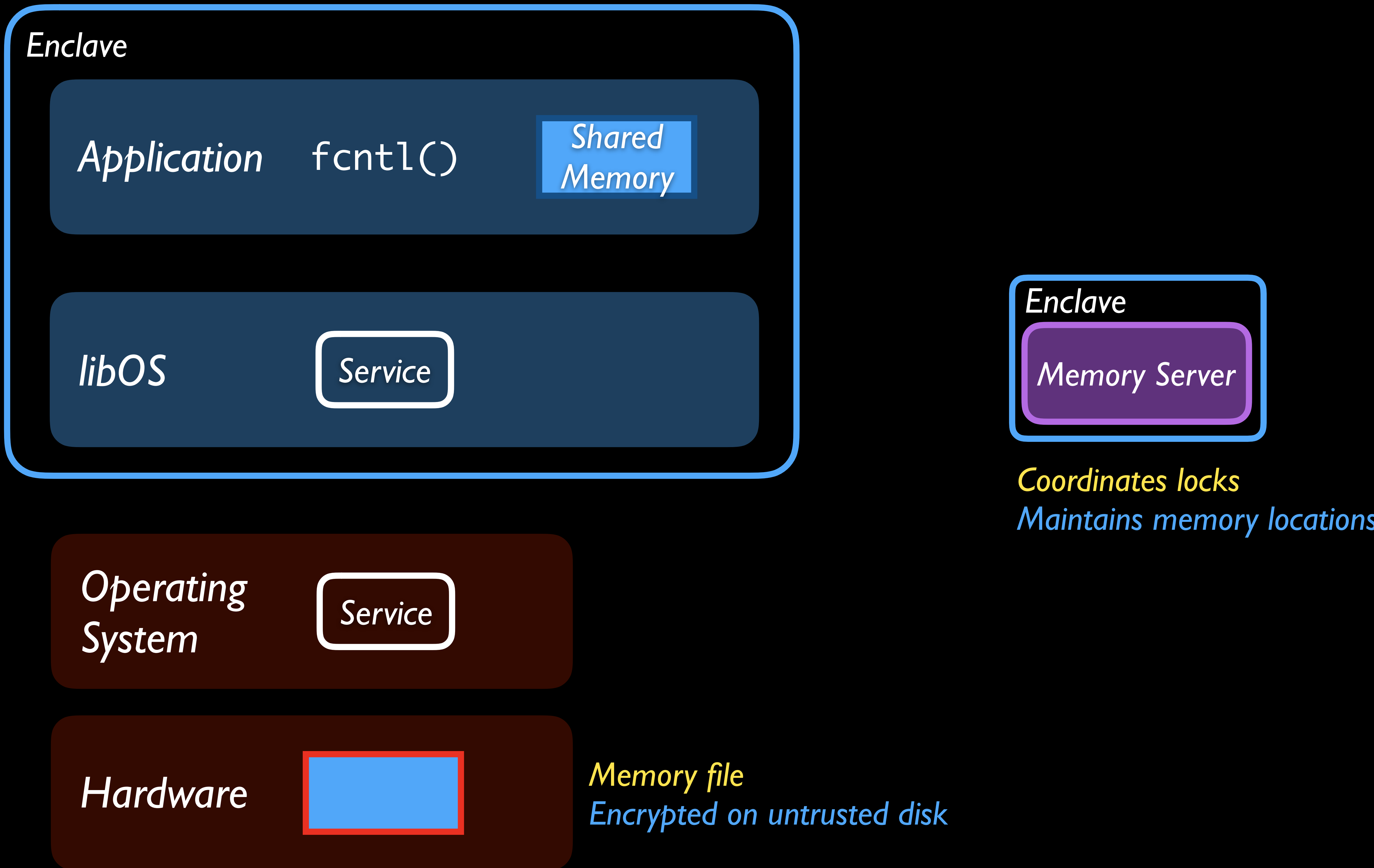
Conclaves Shared memory



Conclaves Shared memory

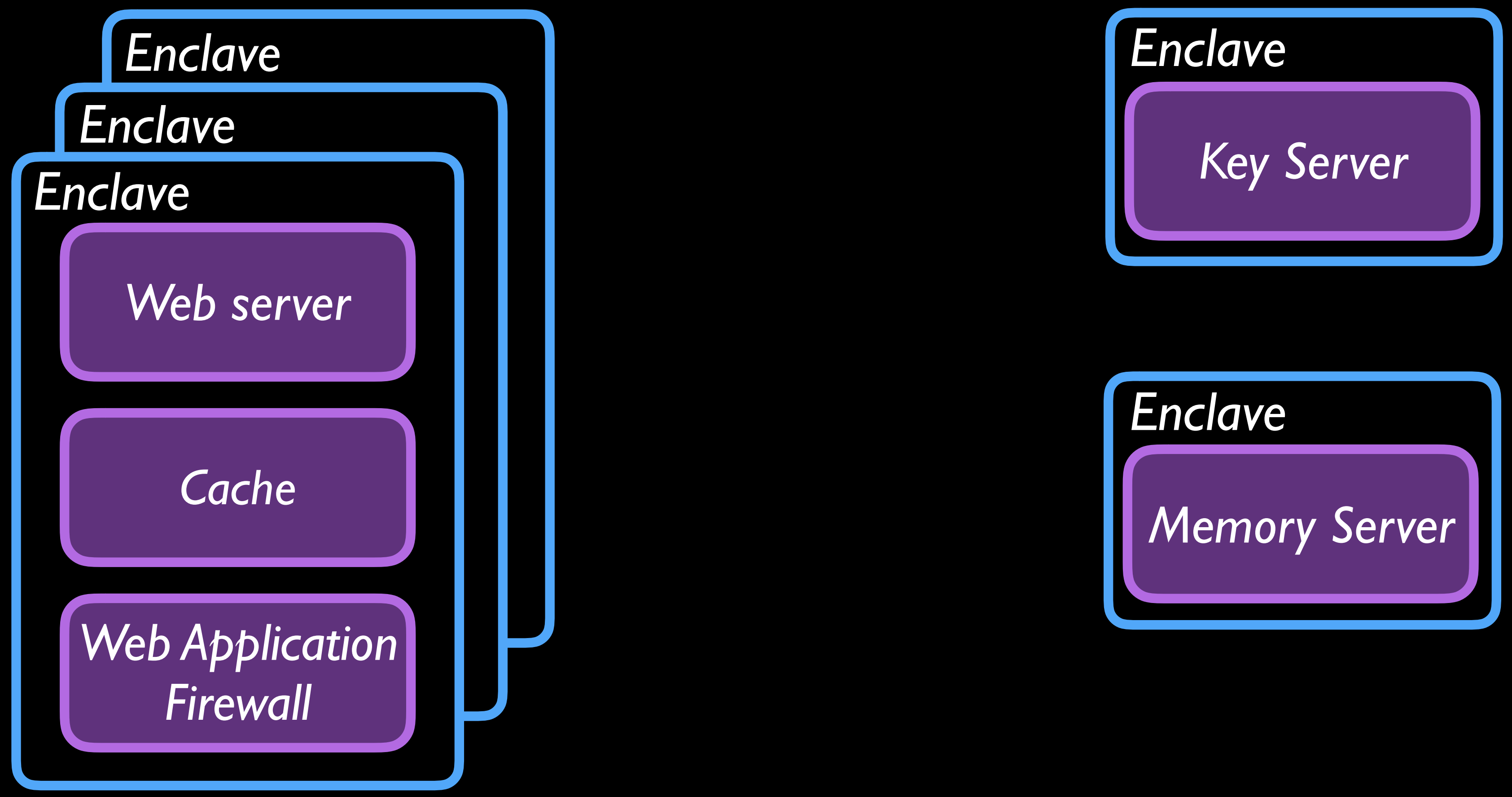


Conclaves Shared memory



Phoenix The first truly *keyless CDN*

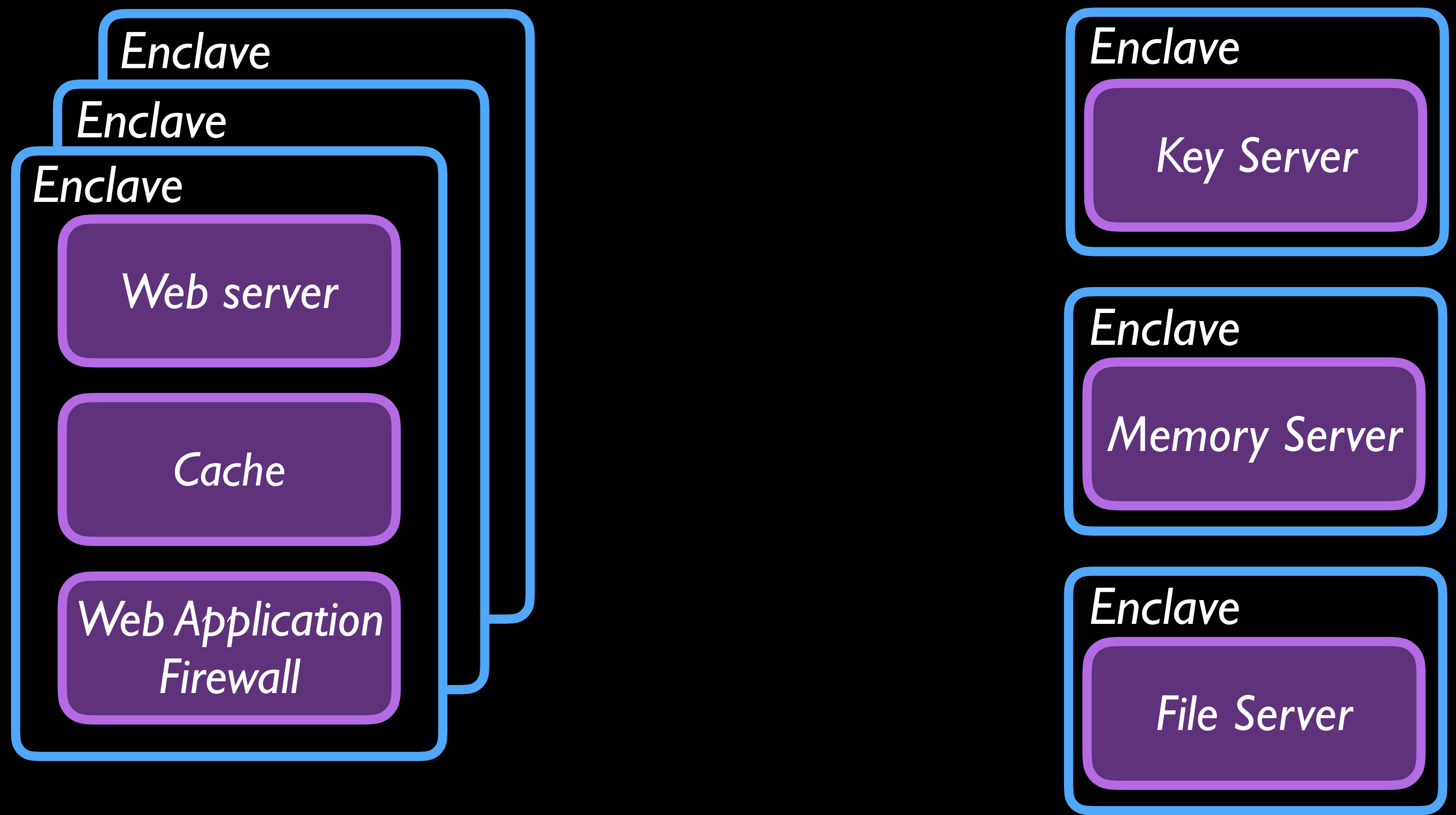
Conclaves *Containers of enclaves*



Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

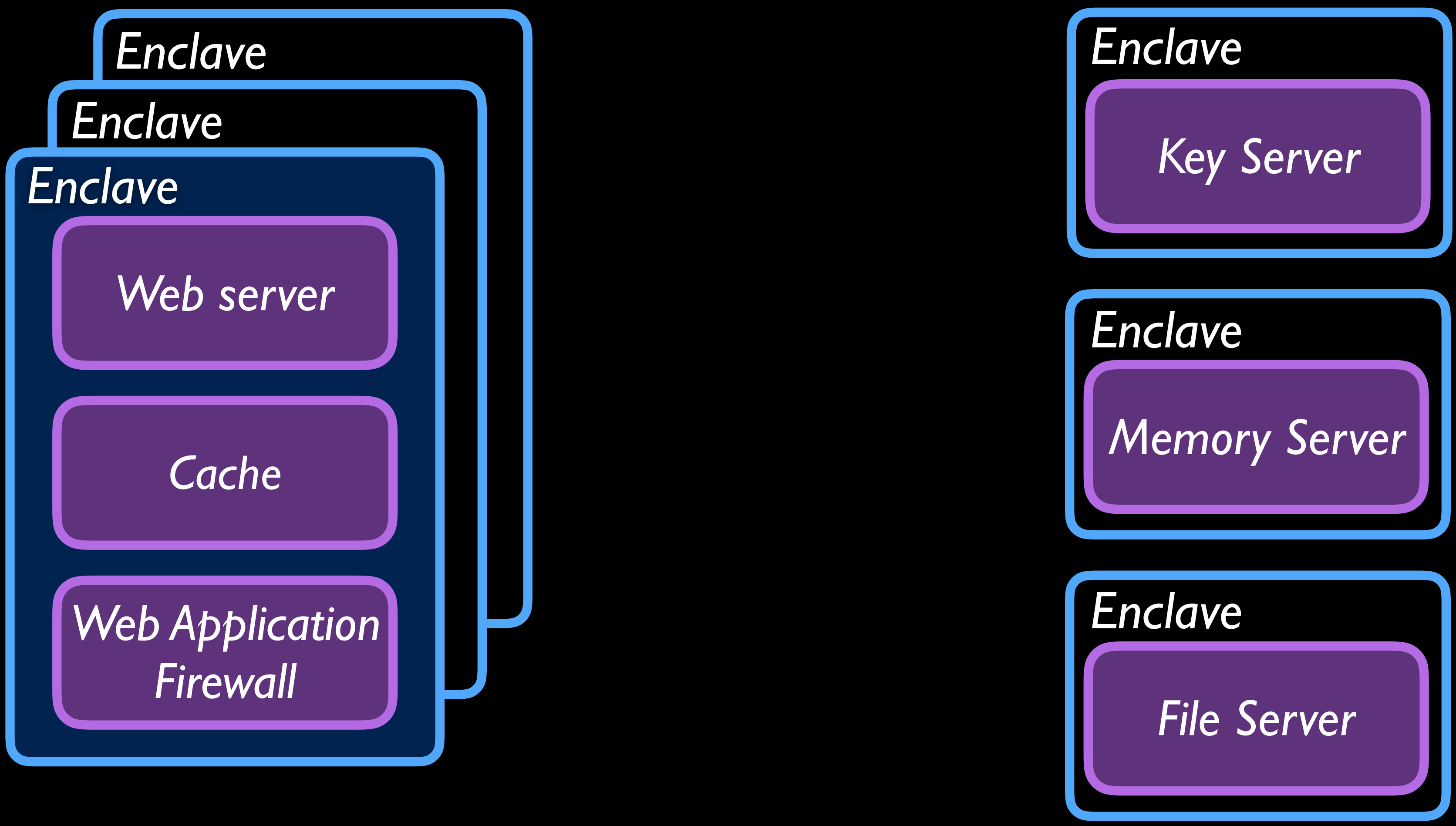
Conclaves *Cont*ainers of en*claves*



Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

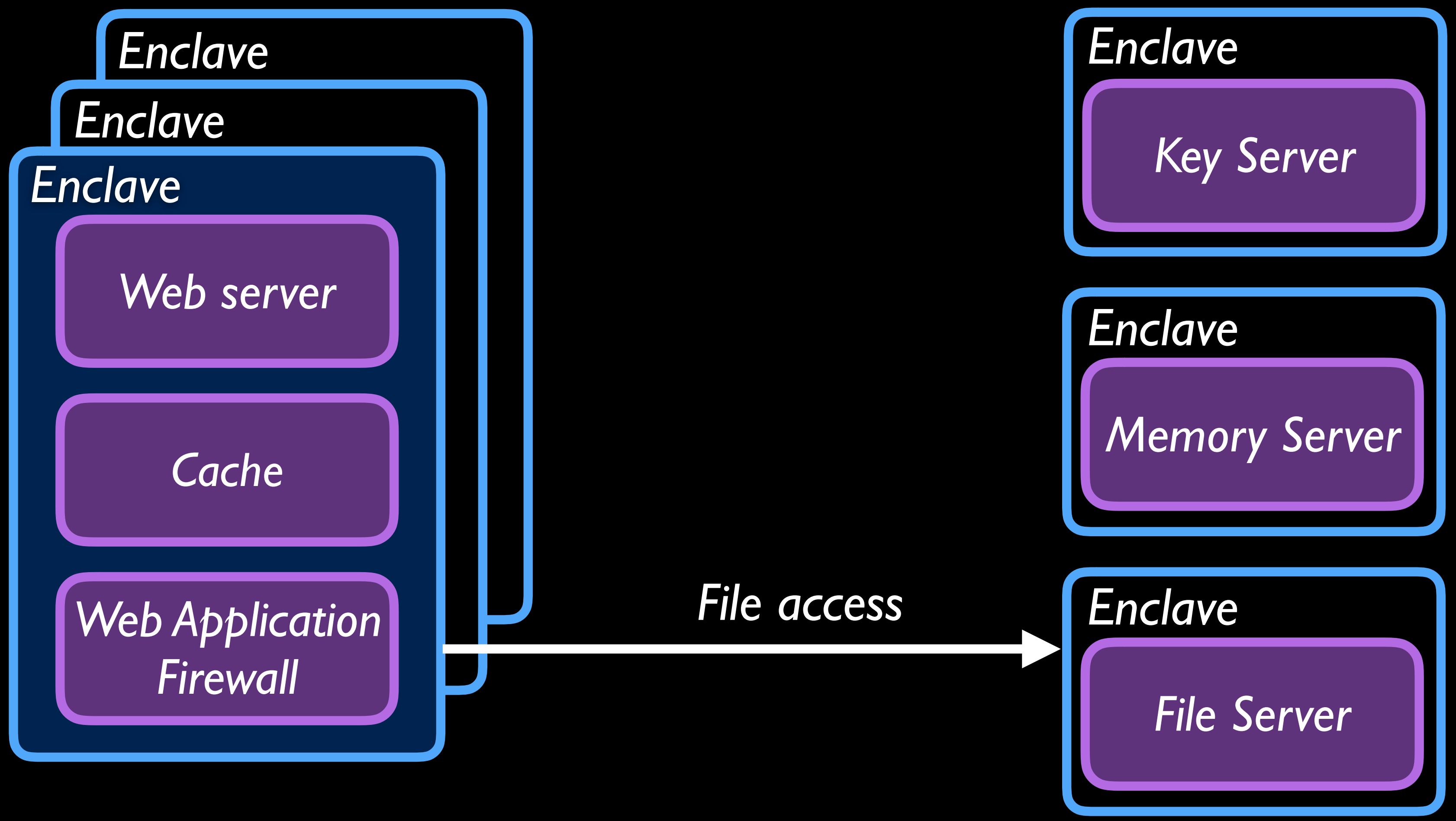
Conclaves *Containers of enclaves*



Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

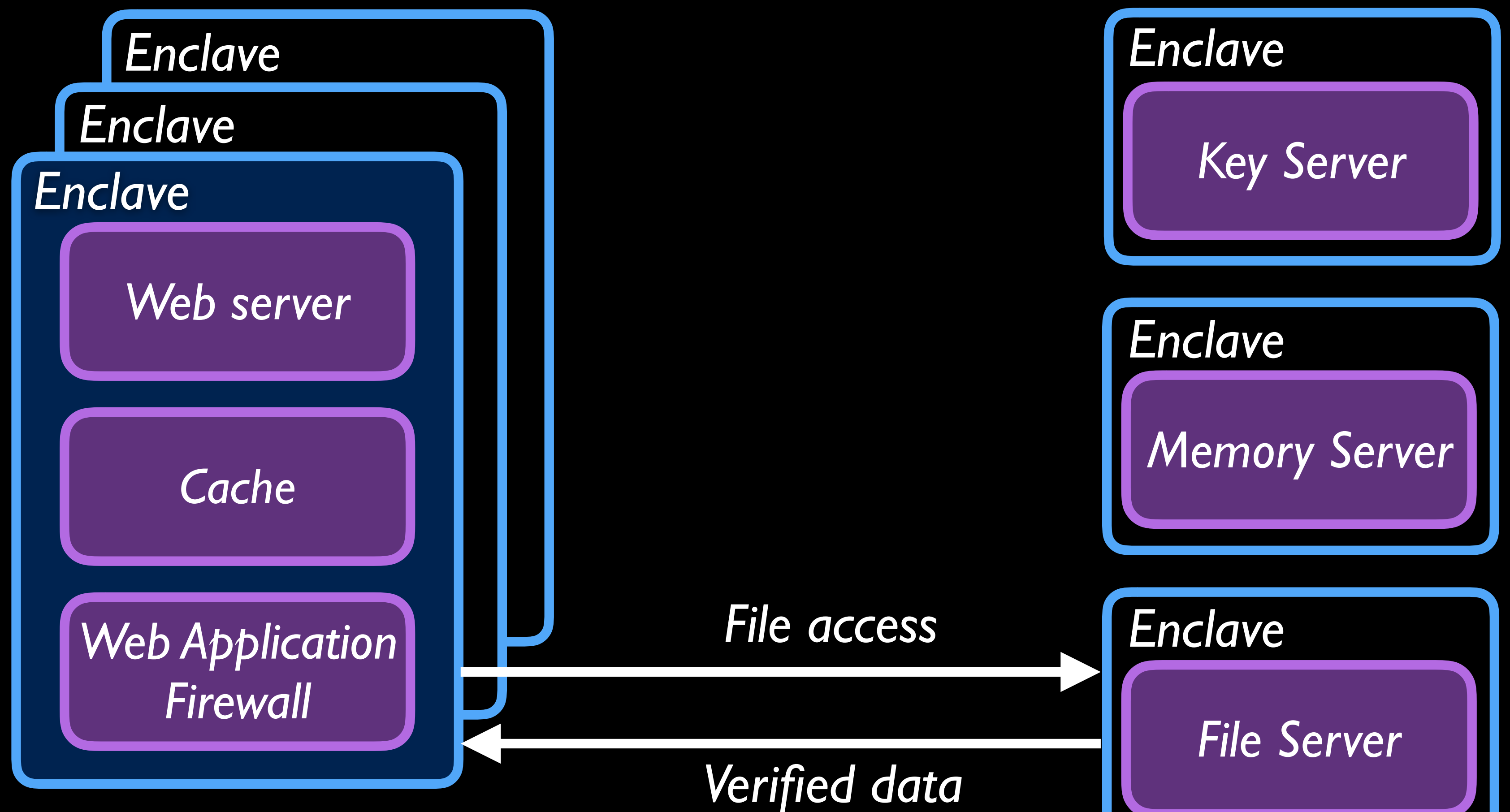
Conclaves *Containers of enclaves*



Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

Phoenix The first truly *keyless CDN*

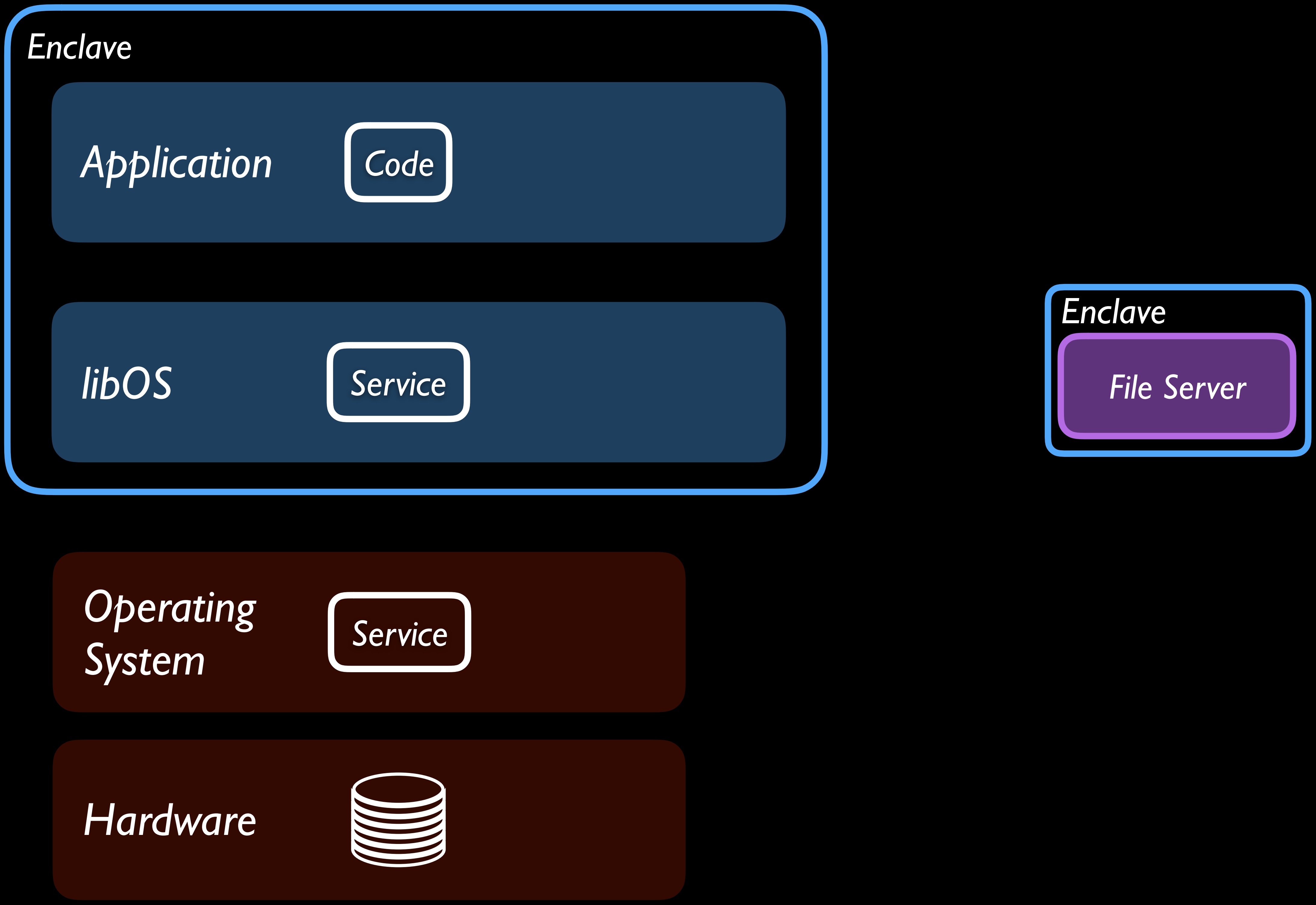
Conclaves *Containers of enclaves*

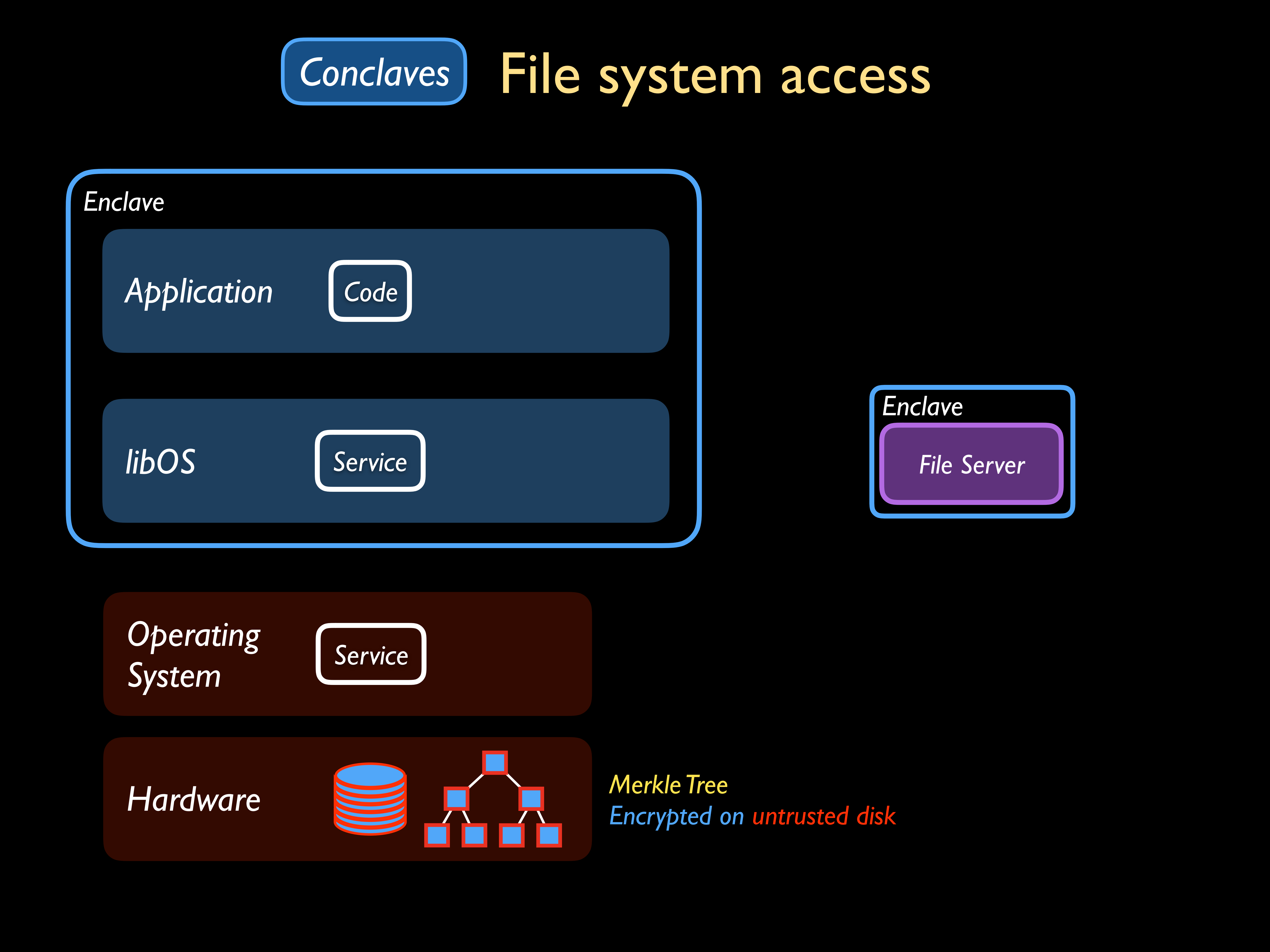


Insight: Treat enclaves like a *distributed system*
Implement services using *kernel servers*

Conclaves

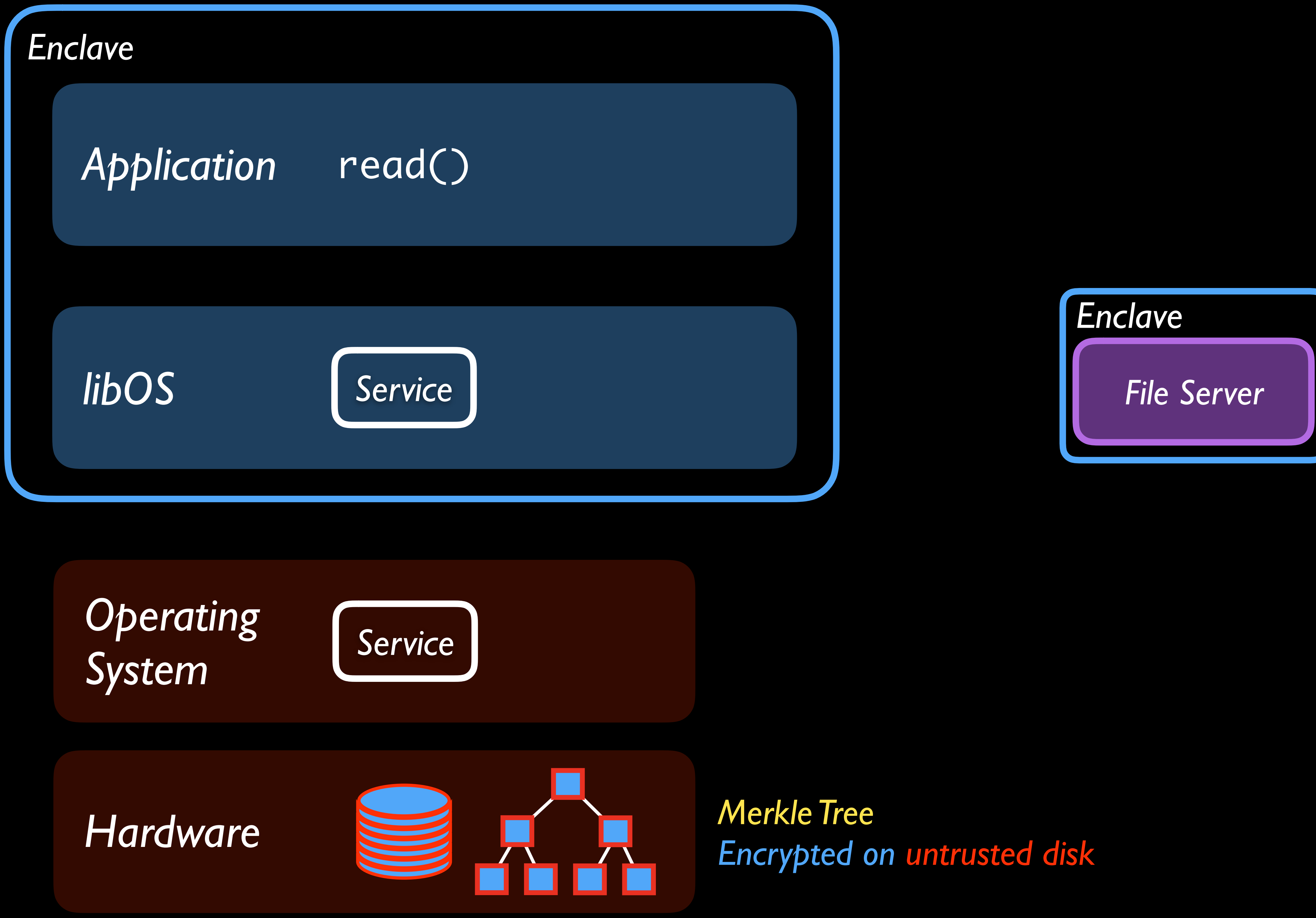
File system access




Conclaves
File system access
Enclave
Application
Code
libOS
Service
Enclave
File Server
Operating System
Service
Hardware
Merkle Tree
Encrypted on untrusted disk

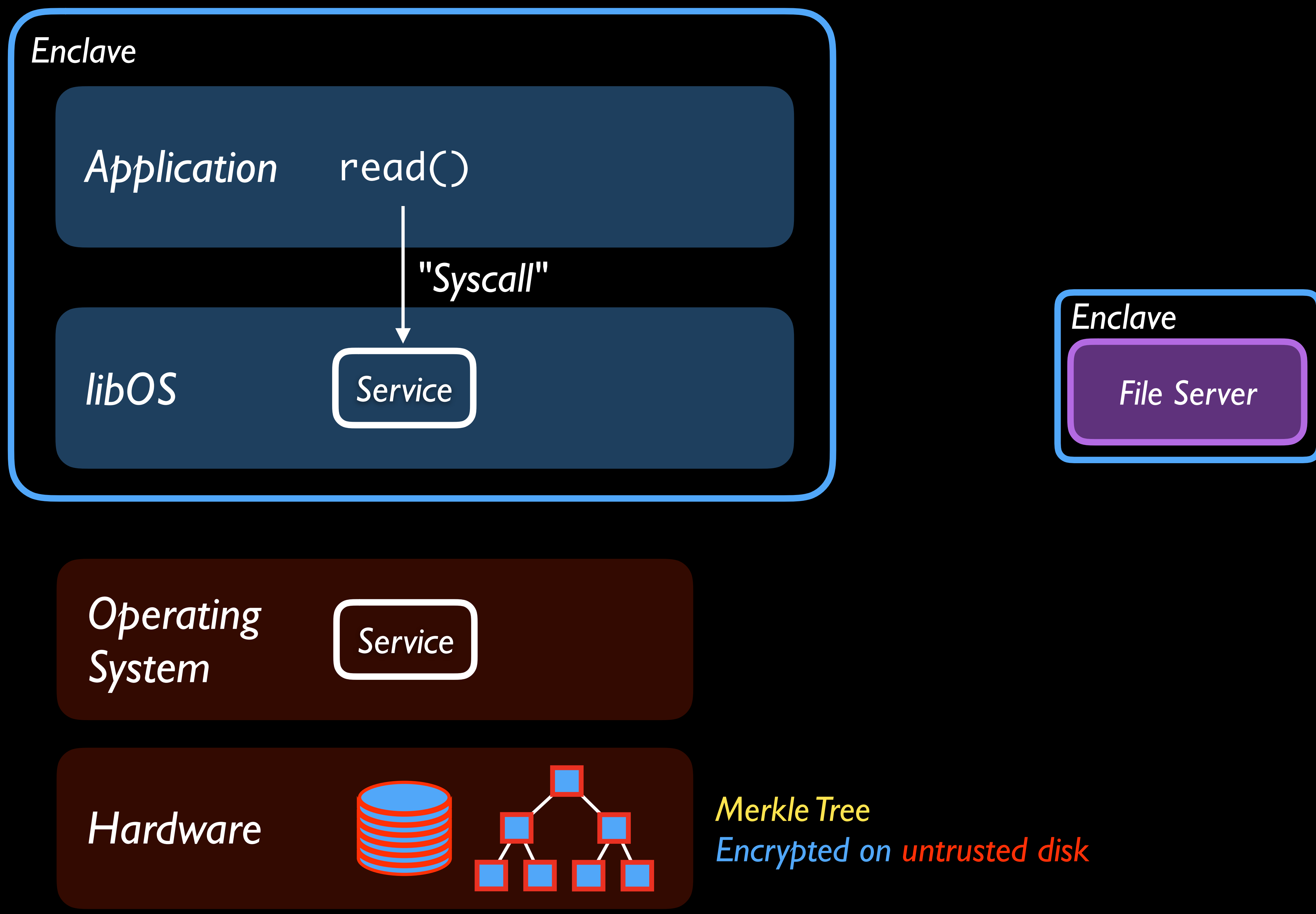
Conclaves

File system access



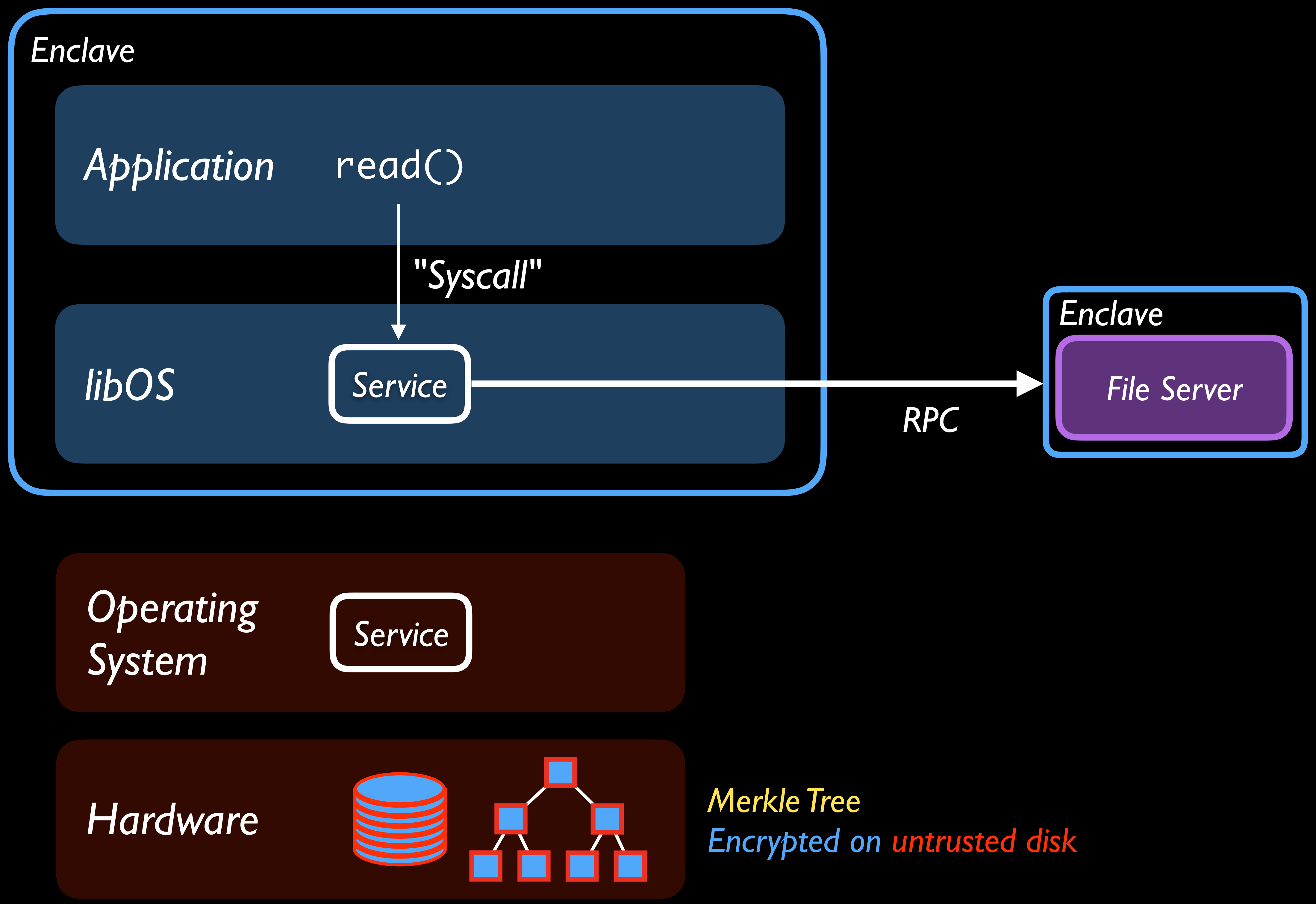
Conclaves

File system access

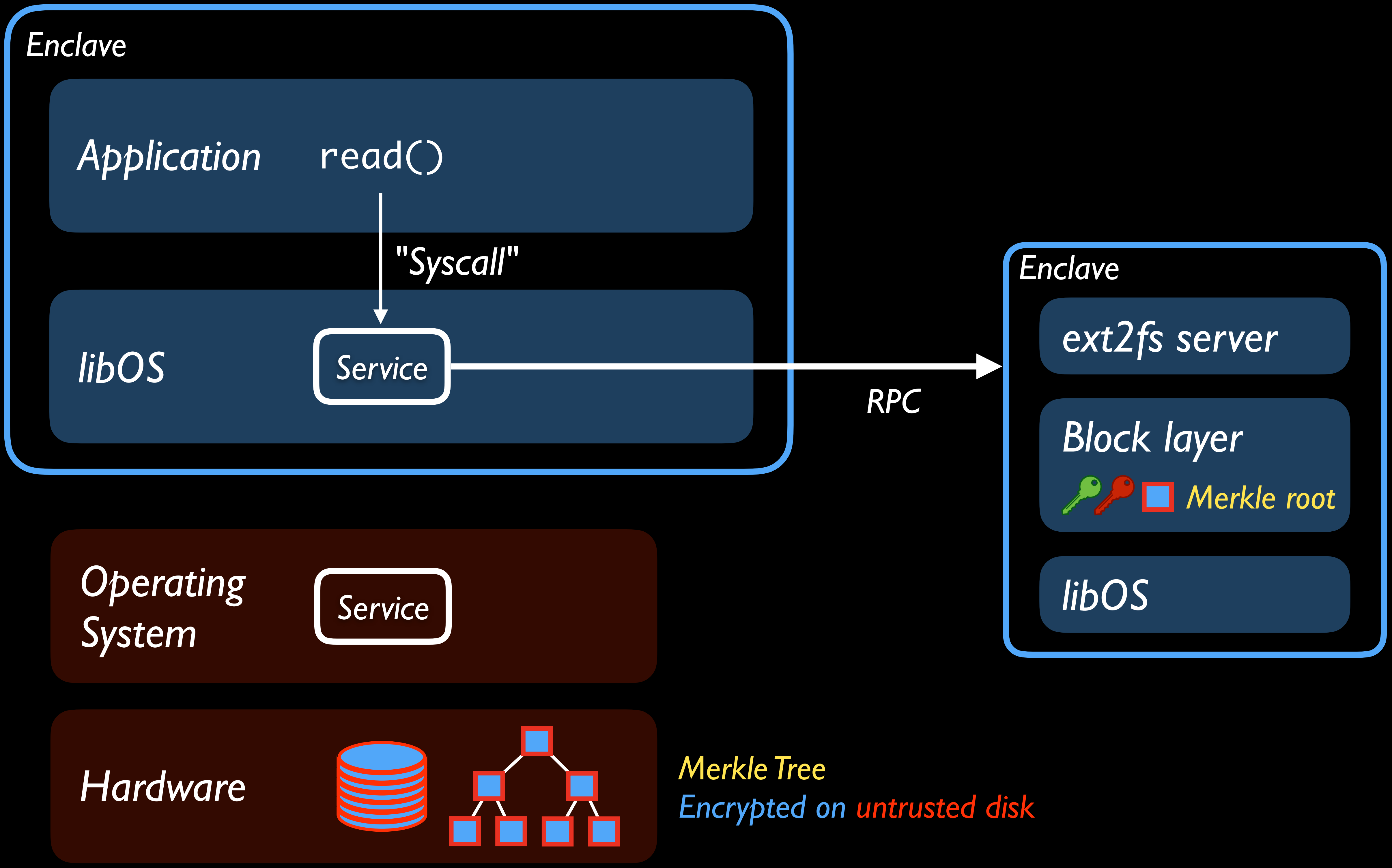


Conclaves

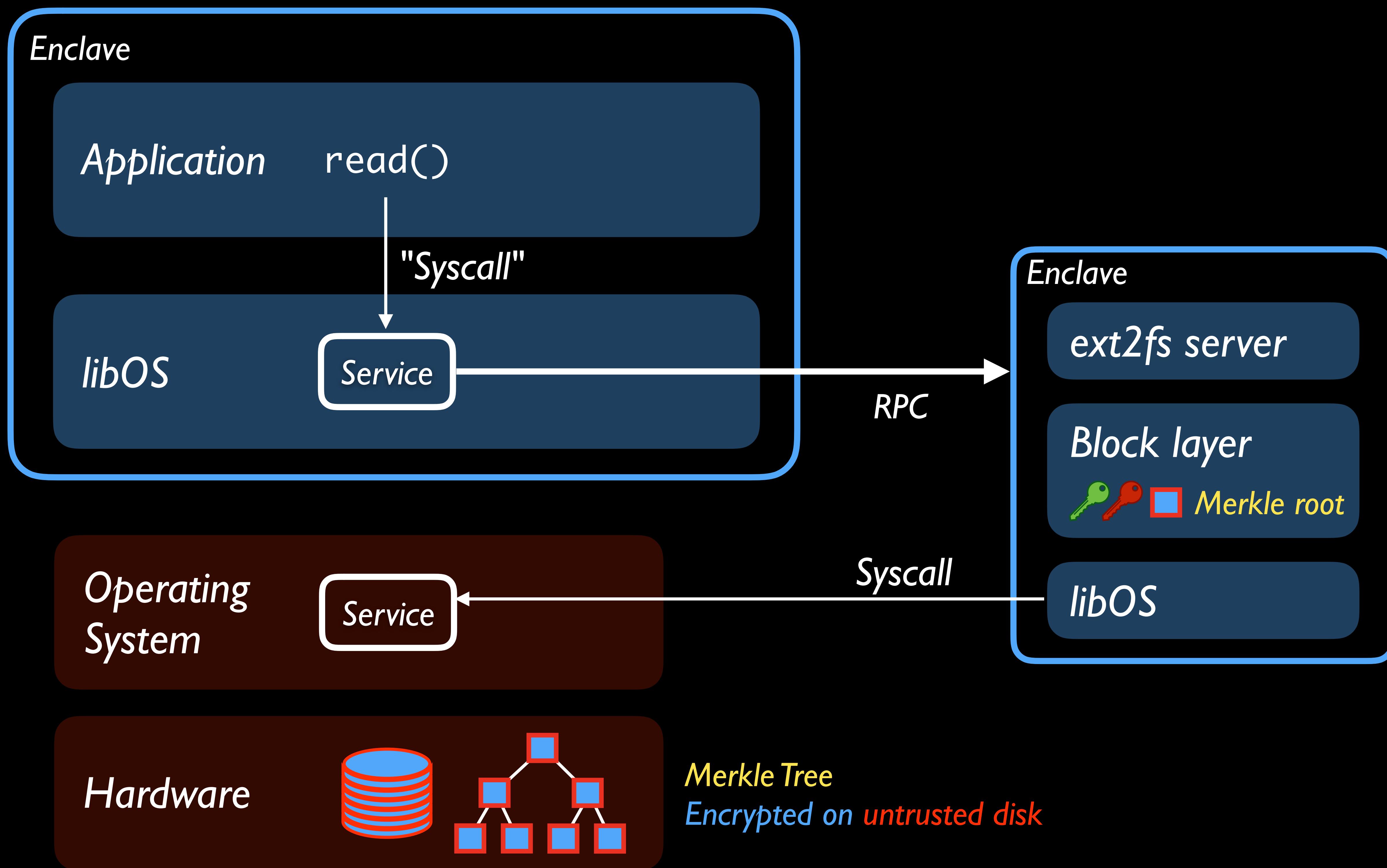
File system access



Conclaves File system access

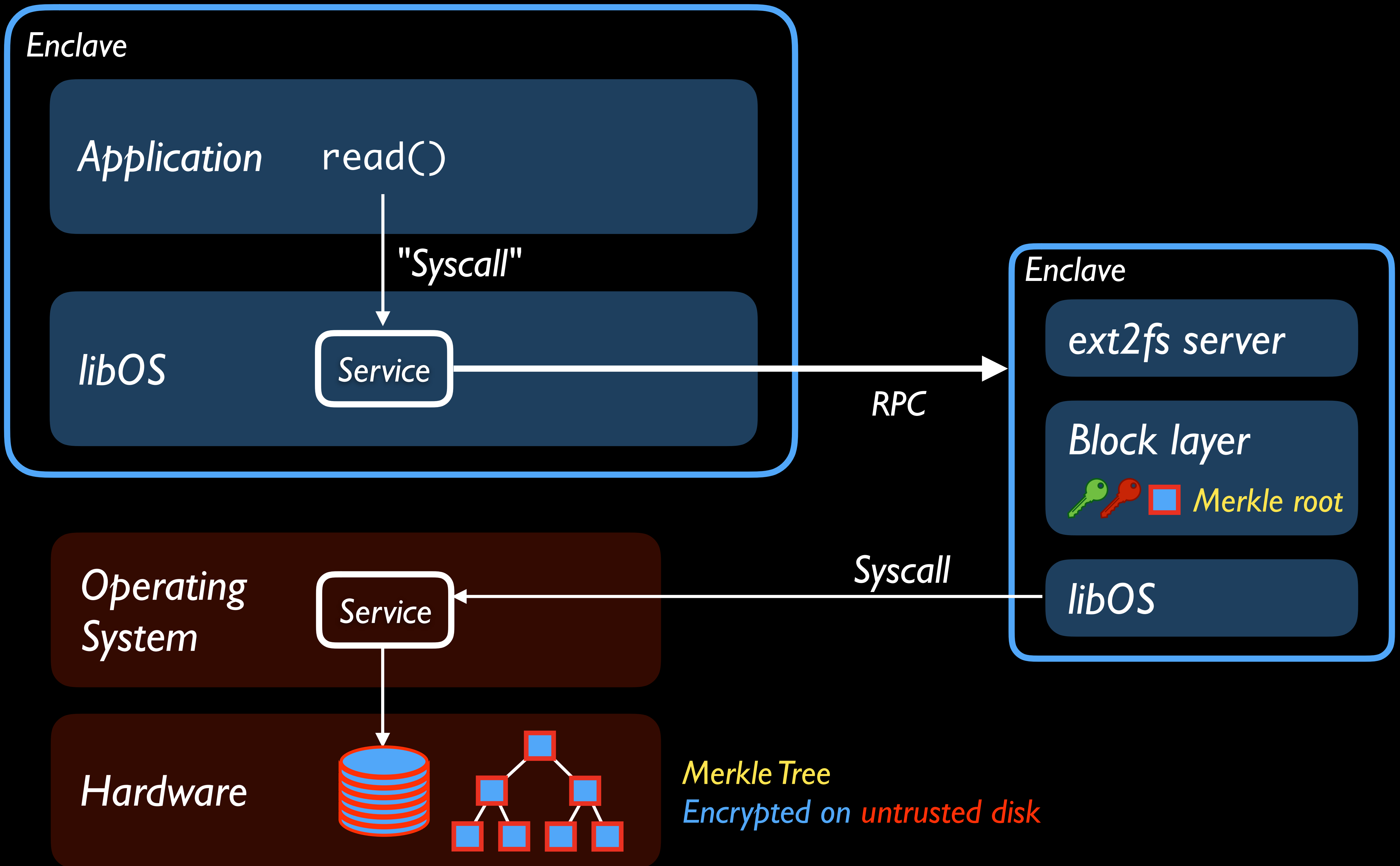


Conclaves File system access



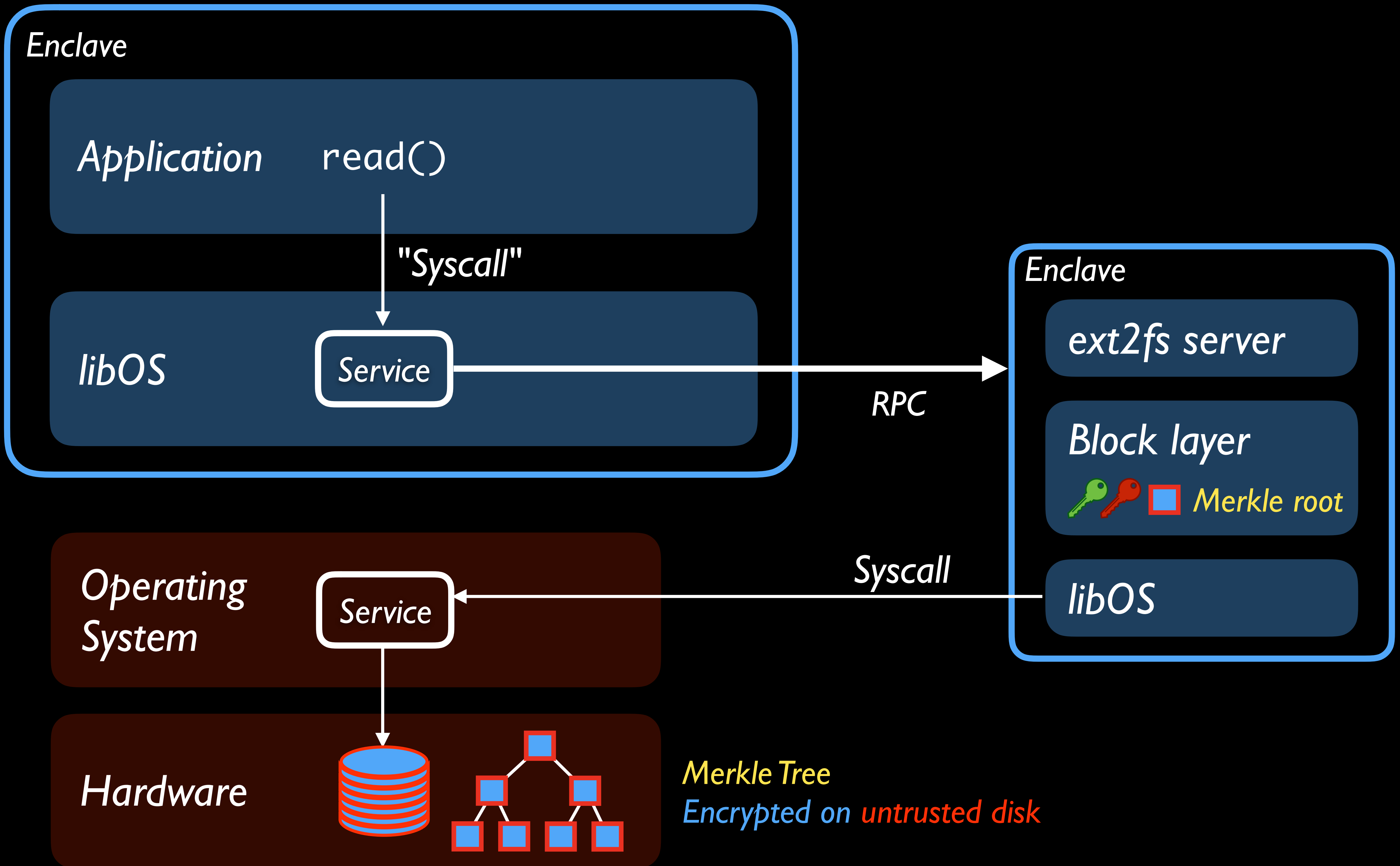
Conclaves

File system access



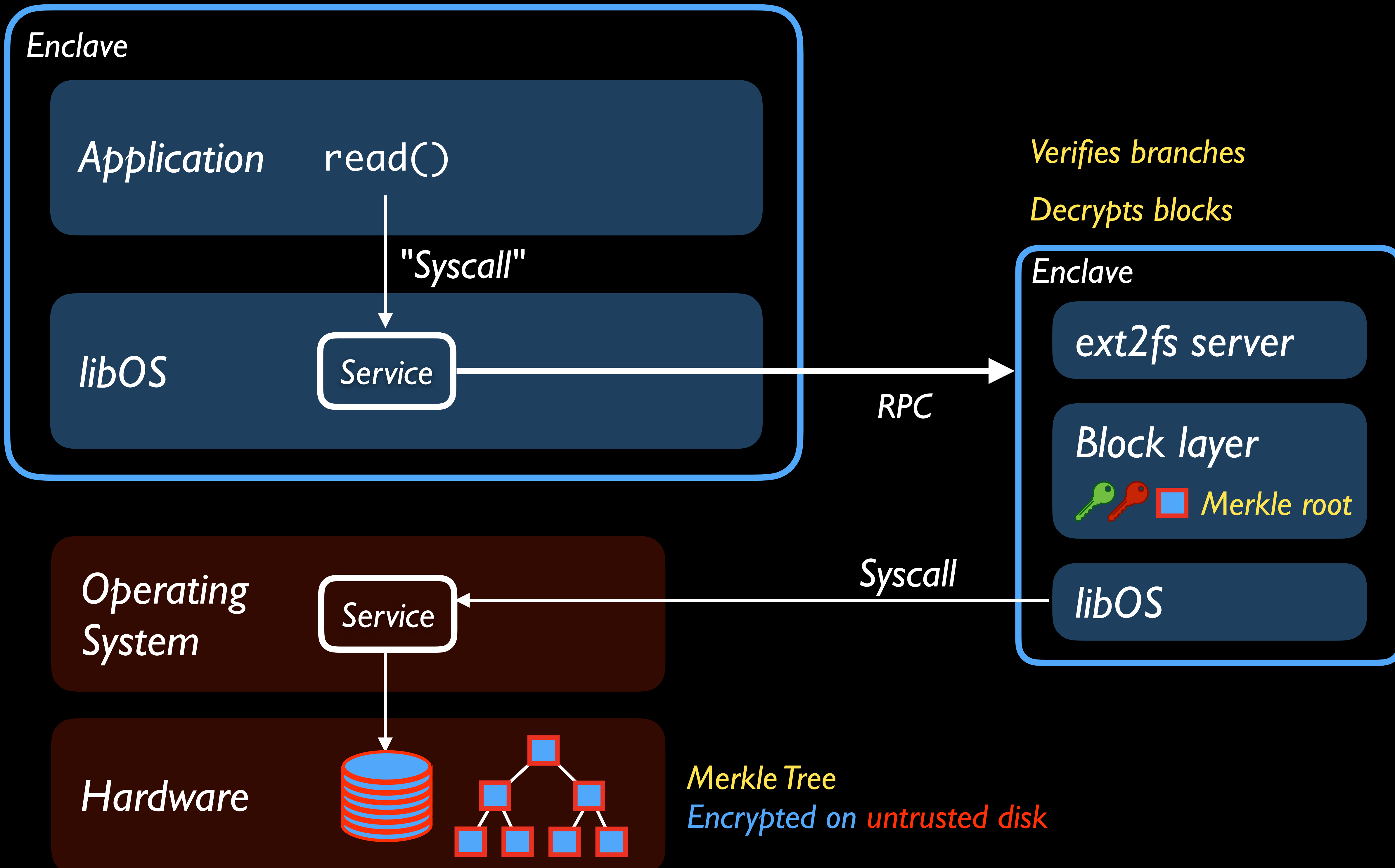
Conclaves

File system access

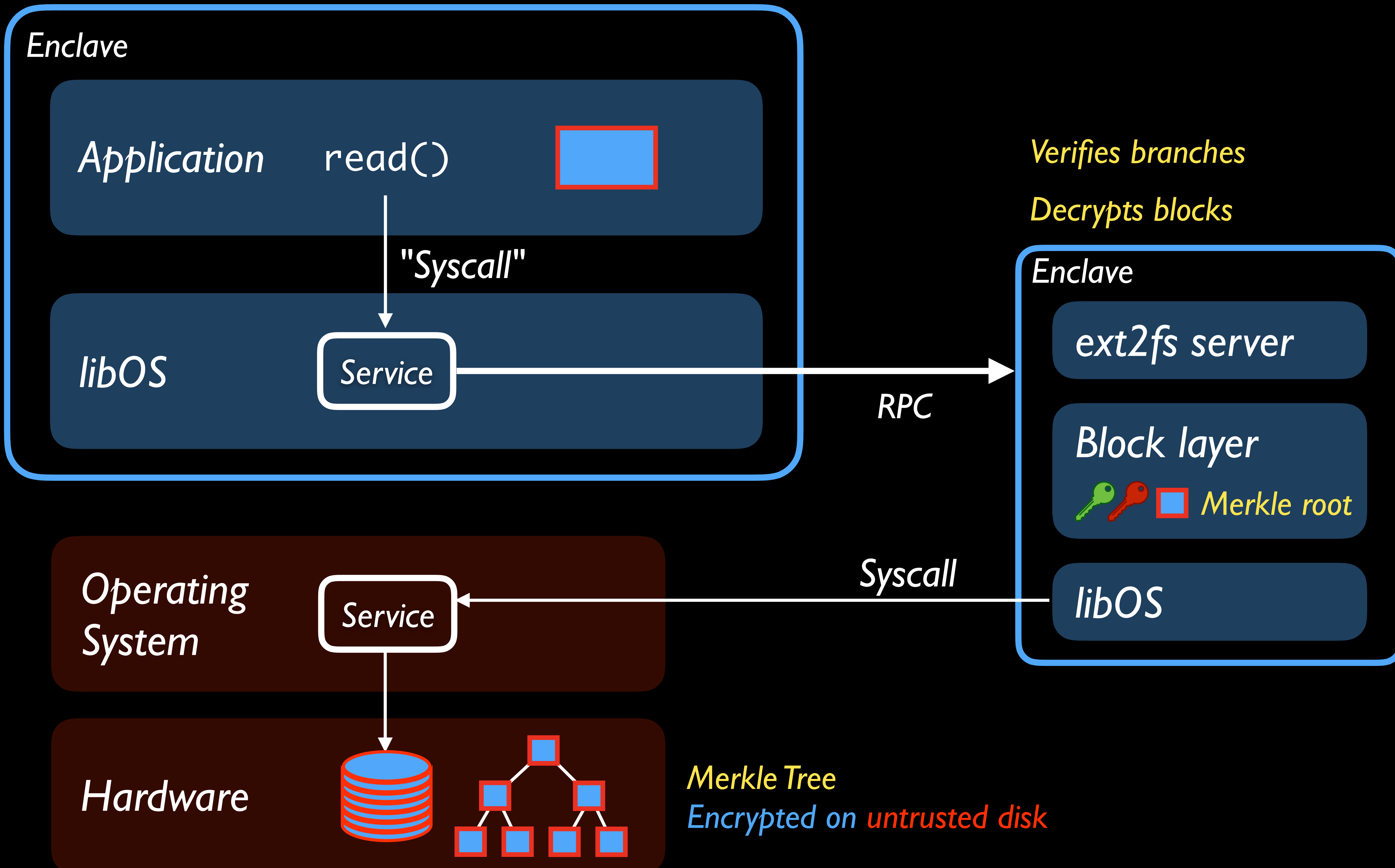


Conclaves

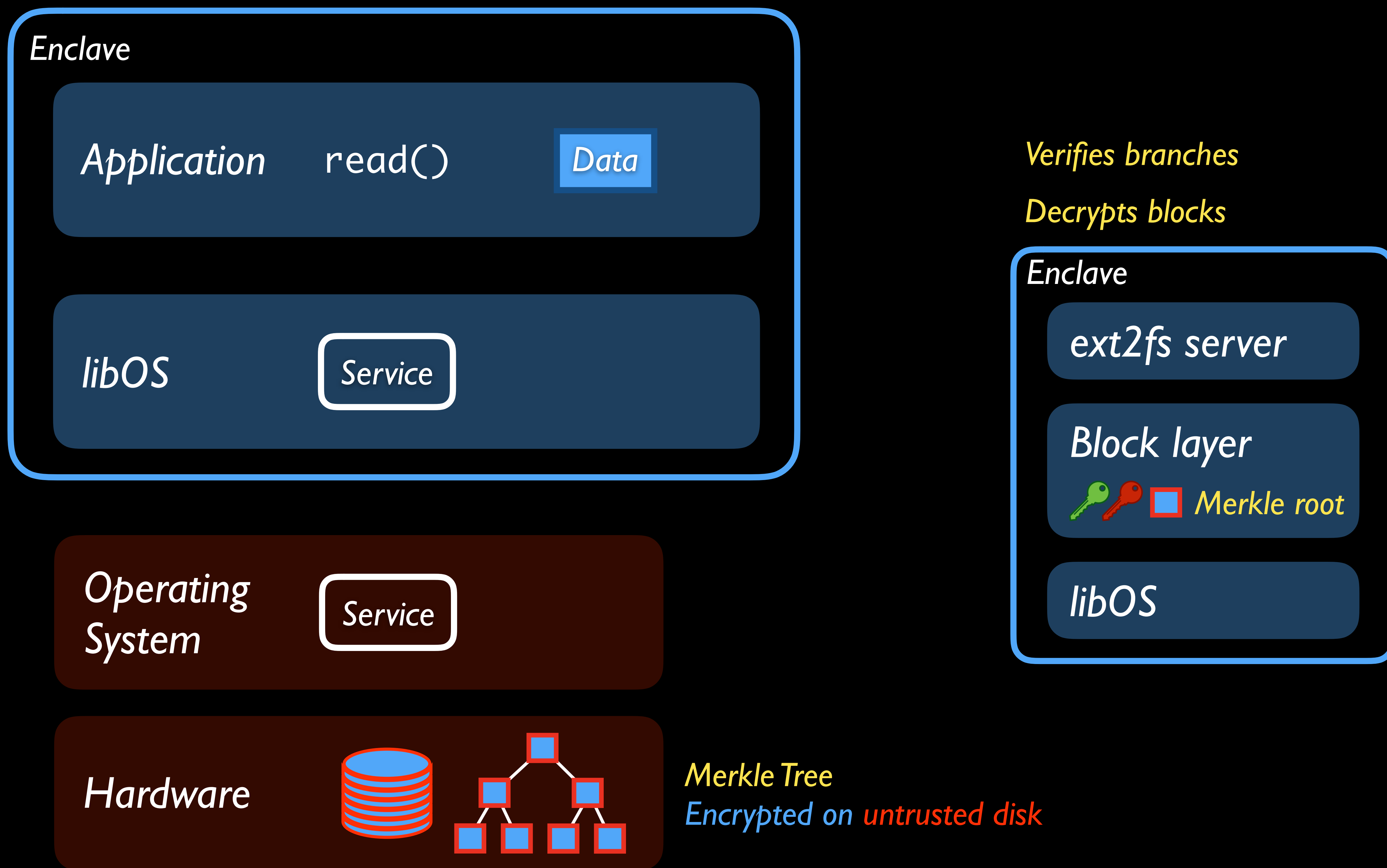
File system access



Conclaves File system access

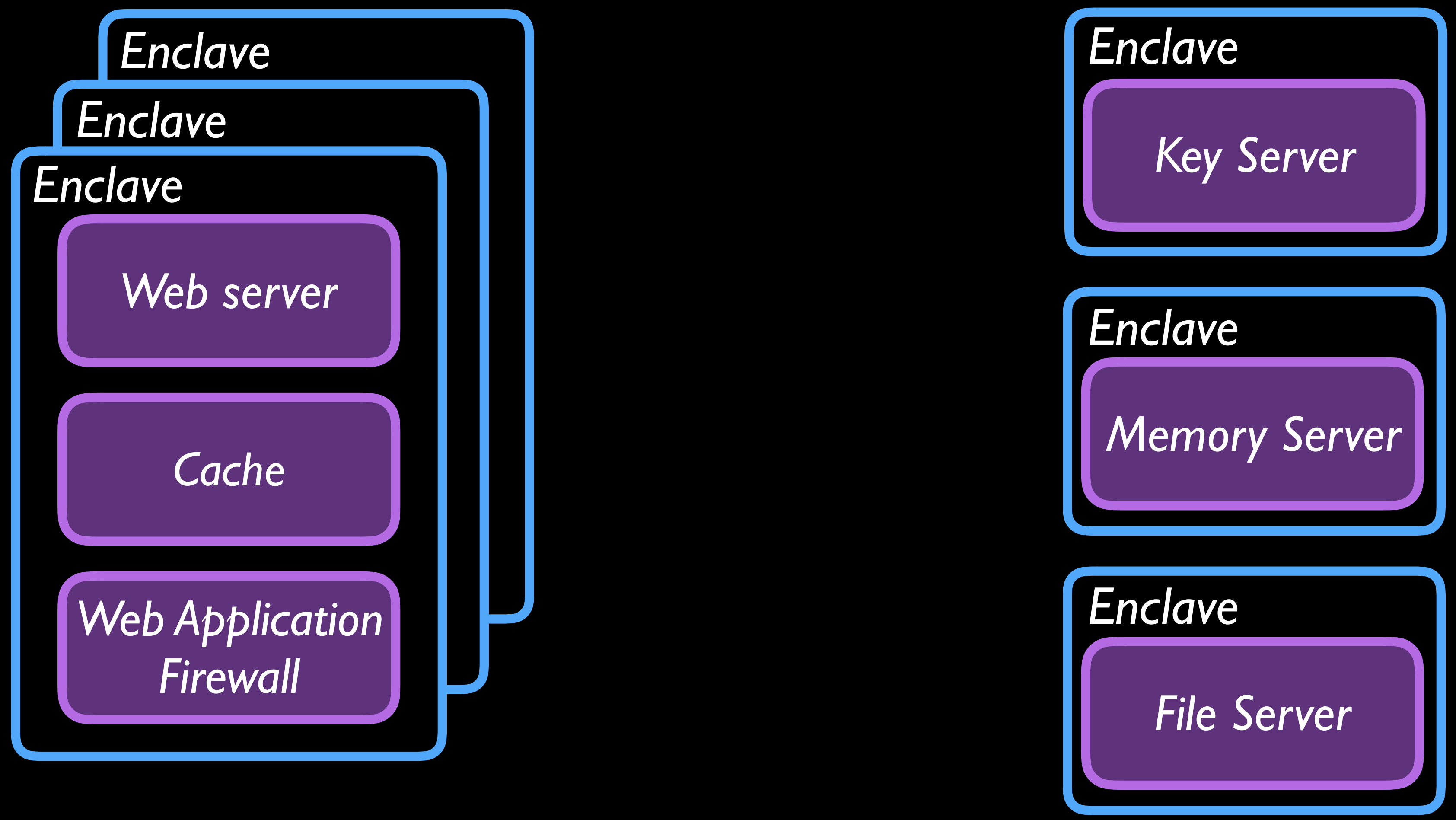


Conclaves File system access



Phoenix The first truly *keyless CDN*

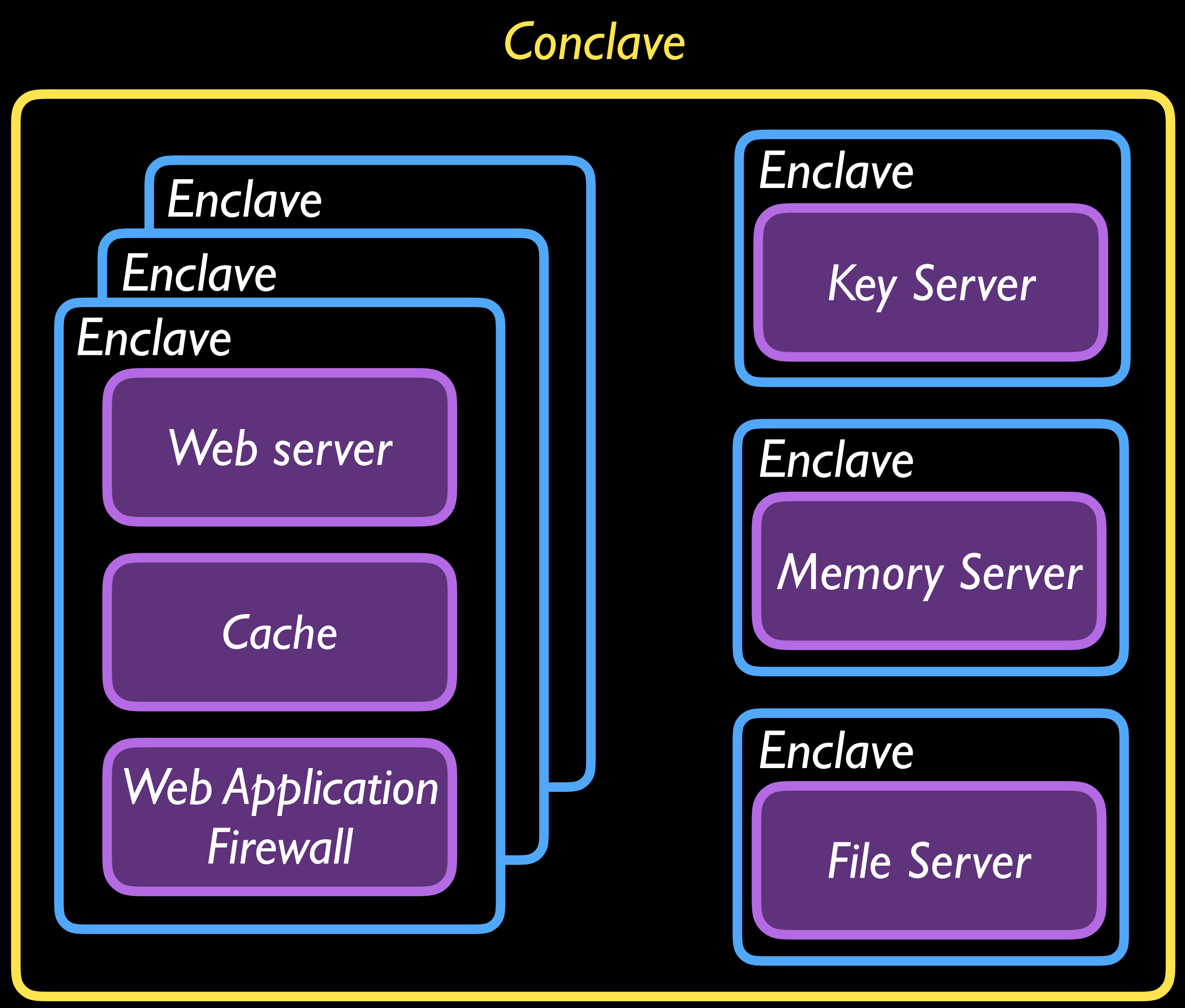
Conclaves *Containers of enclaves*



Execution environment is a distributed system of enclaves

Phoenix The first truly *keyless CDN*

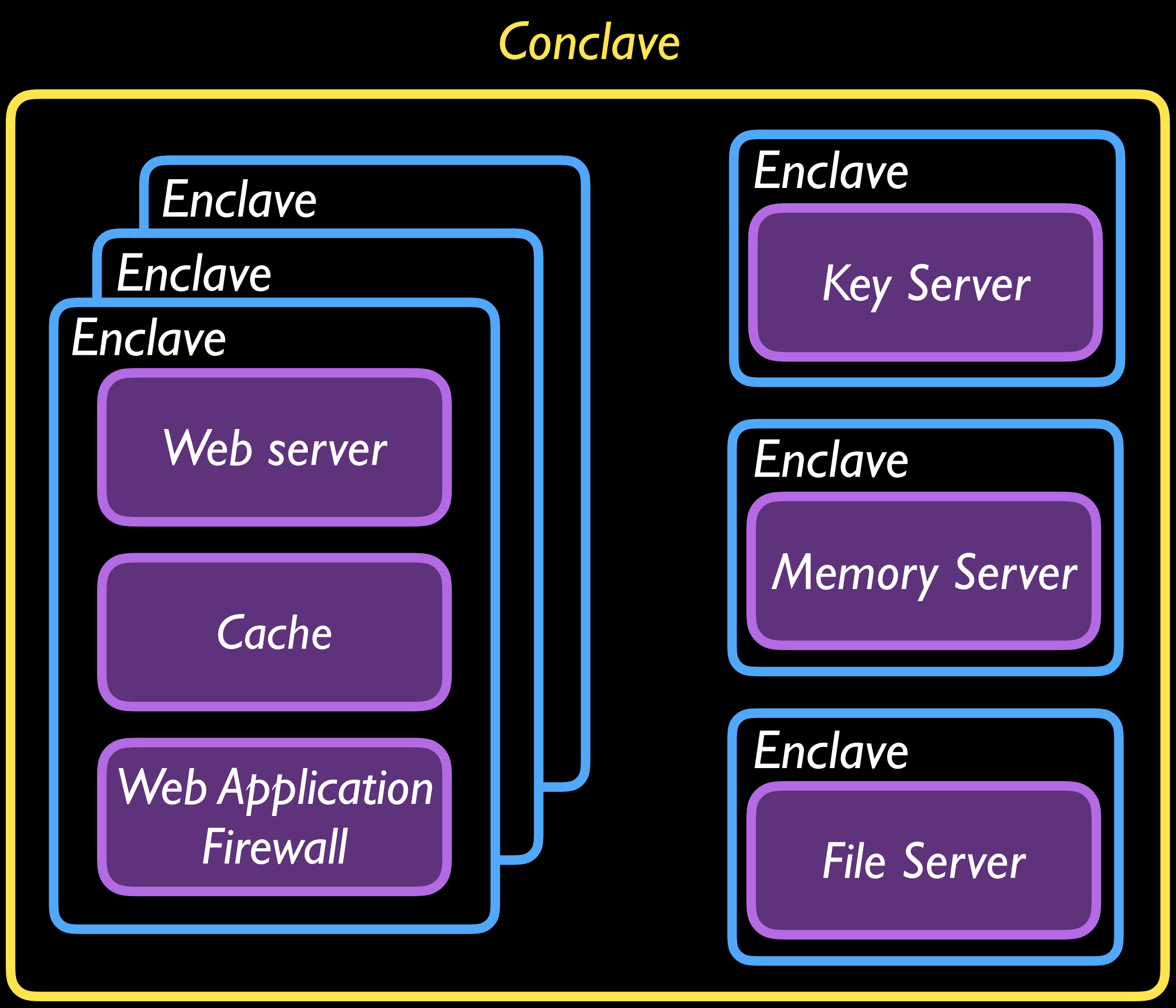
Conclaves *Containers of enclaves*



Execution environment is a distributed system of enclaves

Phoenix The first truly *keyless CDN*

Conclaves *Containers of enclaves*

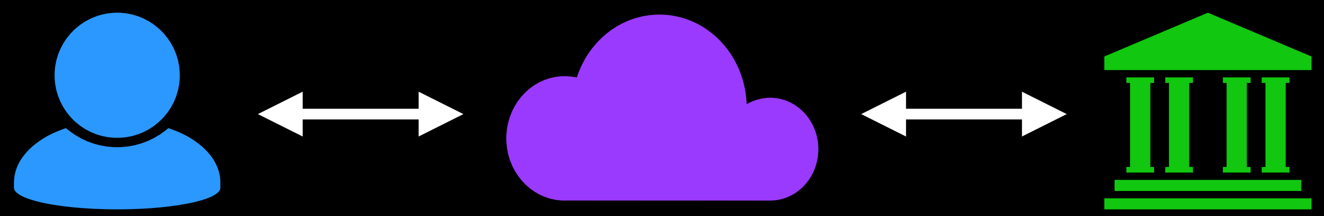


Conclaves supported services:

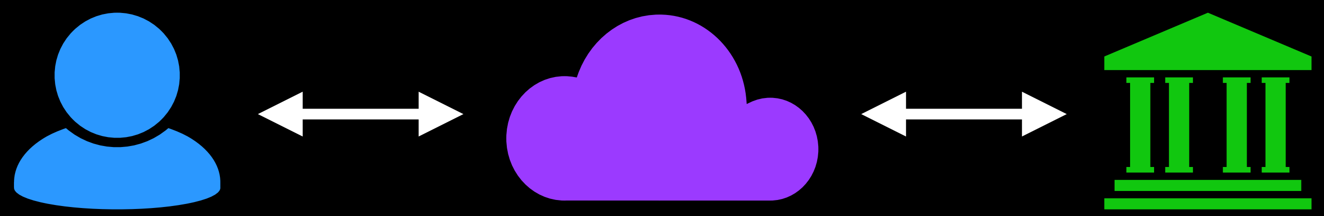
- ✓ *fork*
- ✓ *exec*
- ✓ *pipes, signals, semaphores*
- ✓ *Reading & writing files*
- ✓ *Shared memory*
- ✓ *Access to private keys*
- ✓ *Trusted time server*

Execution environment is a distributed system of enclaves

Phoenix The first truly *keyless CDN*



Phoenix The first truly *keyless CDN*



Supports multi-tenancy

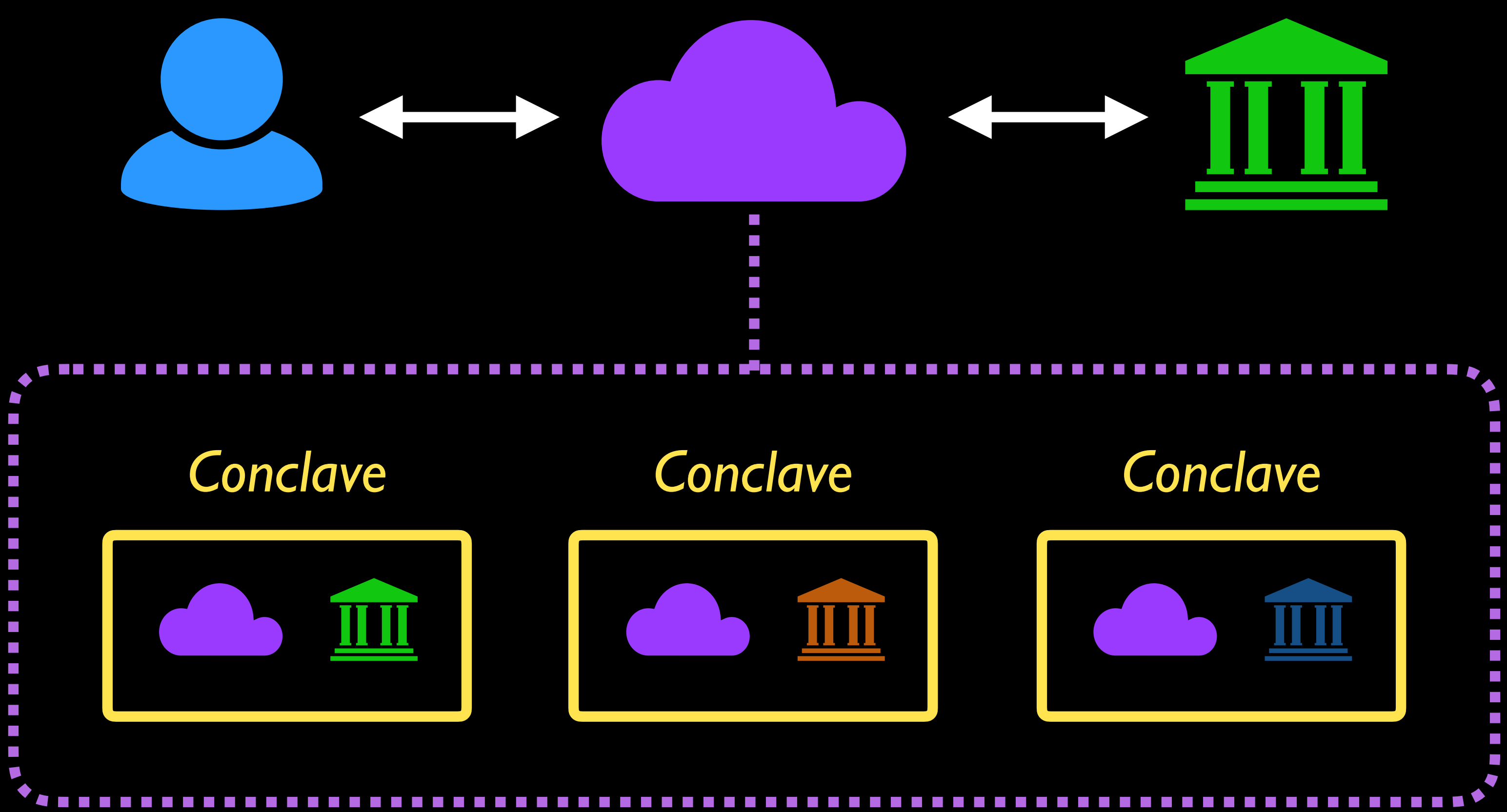
Both CDN and website can store private data

Other details in the paper

Websites delegate provisioning to CDNs

Phoenix supports many deployment configurations

Phoenix The first truly *keyless CDN*



Supports multi-tenancy

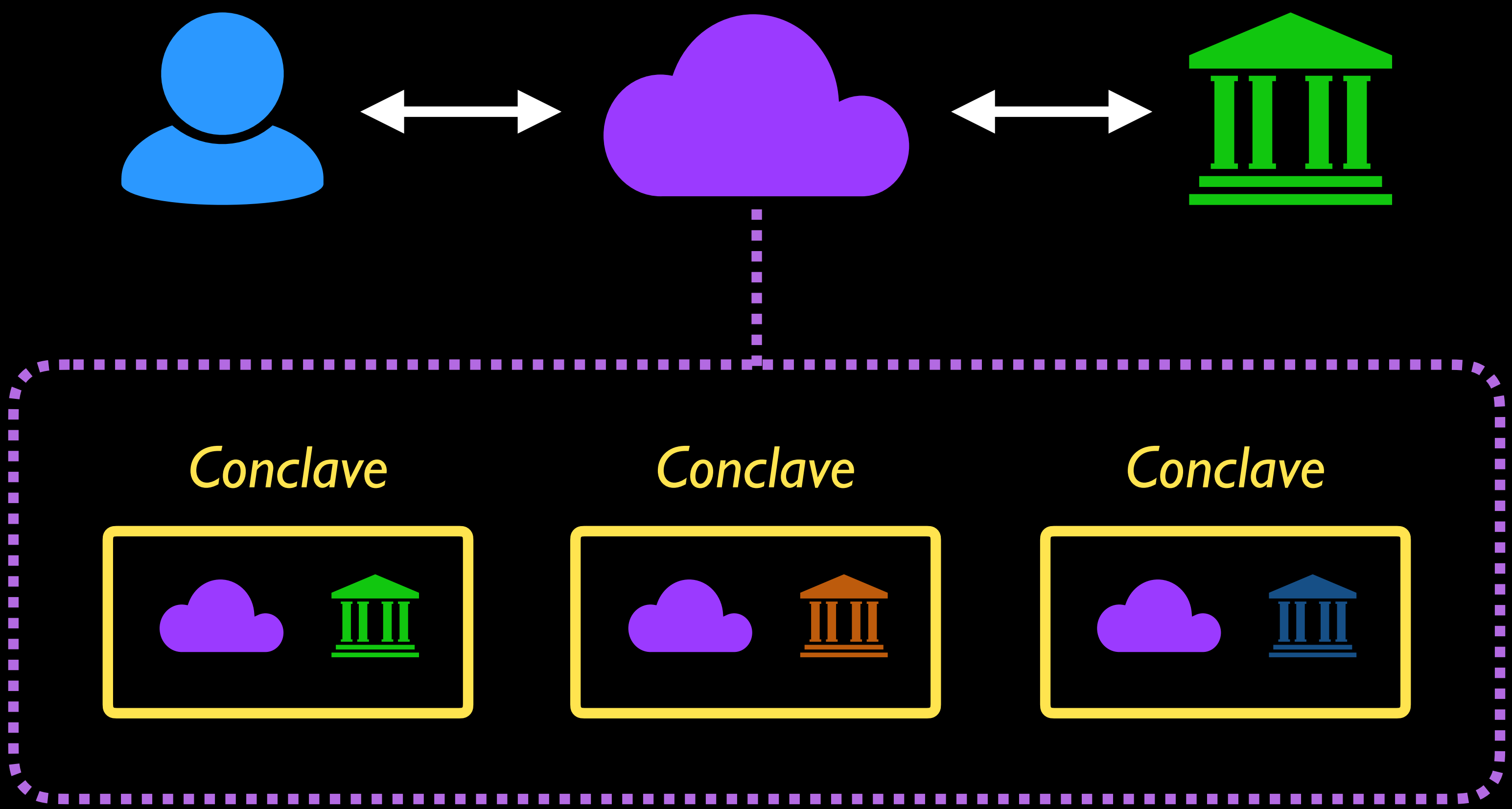
Both CDN and website can store private data

Other details in the paper

Websites delegate provisioning to CDNs

Phoenix supports many deployment configurations

Phoenix The first truly *keyless CDN*



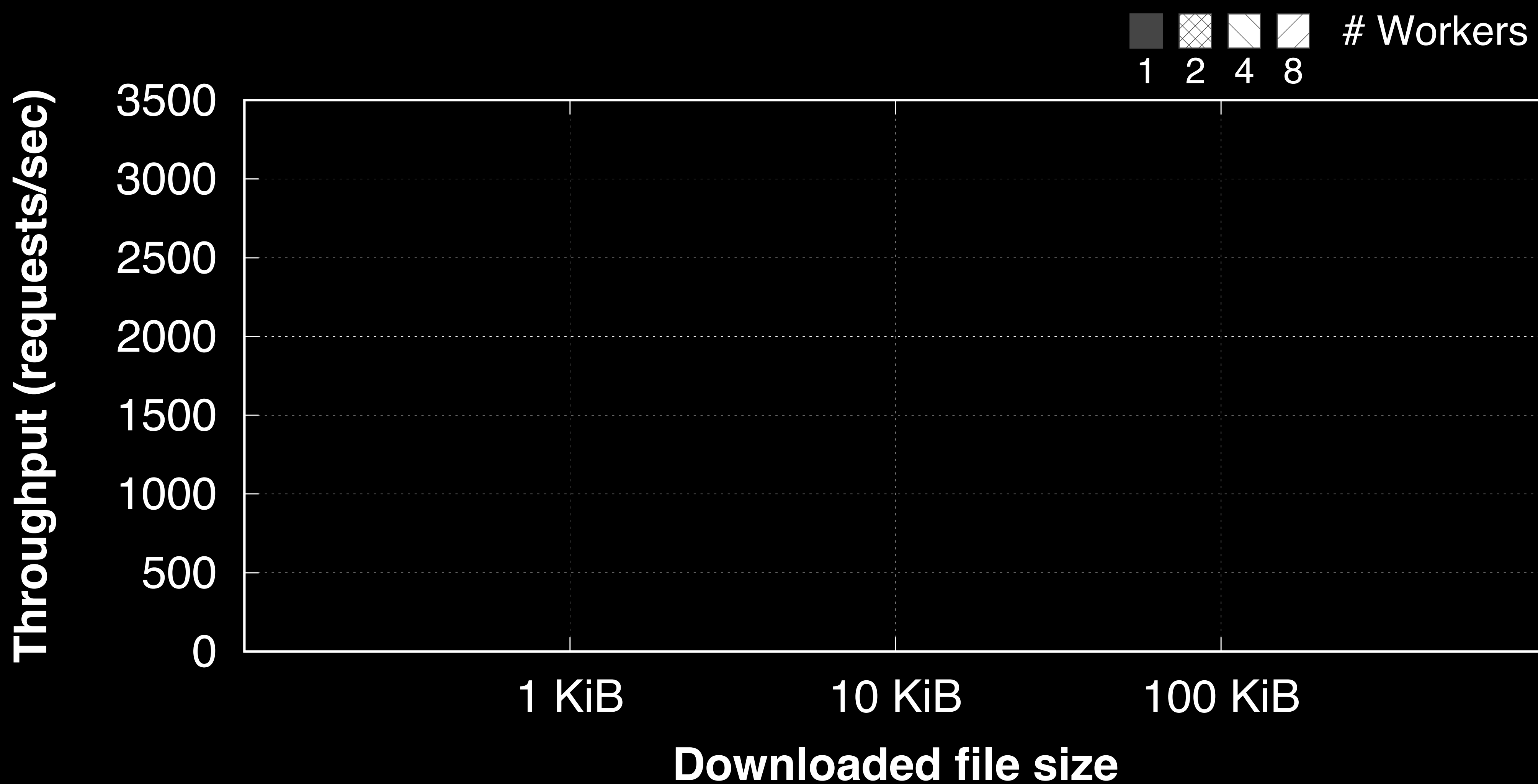
ARTIFACT EVALUATED usenix ASSOCIATION PASSED

Implemented on top of Graphene-SGX

Evaluated to understand *throughput* and *scalability*

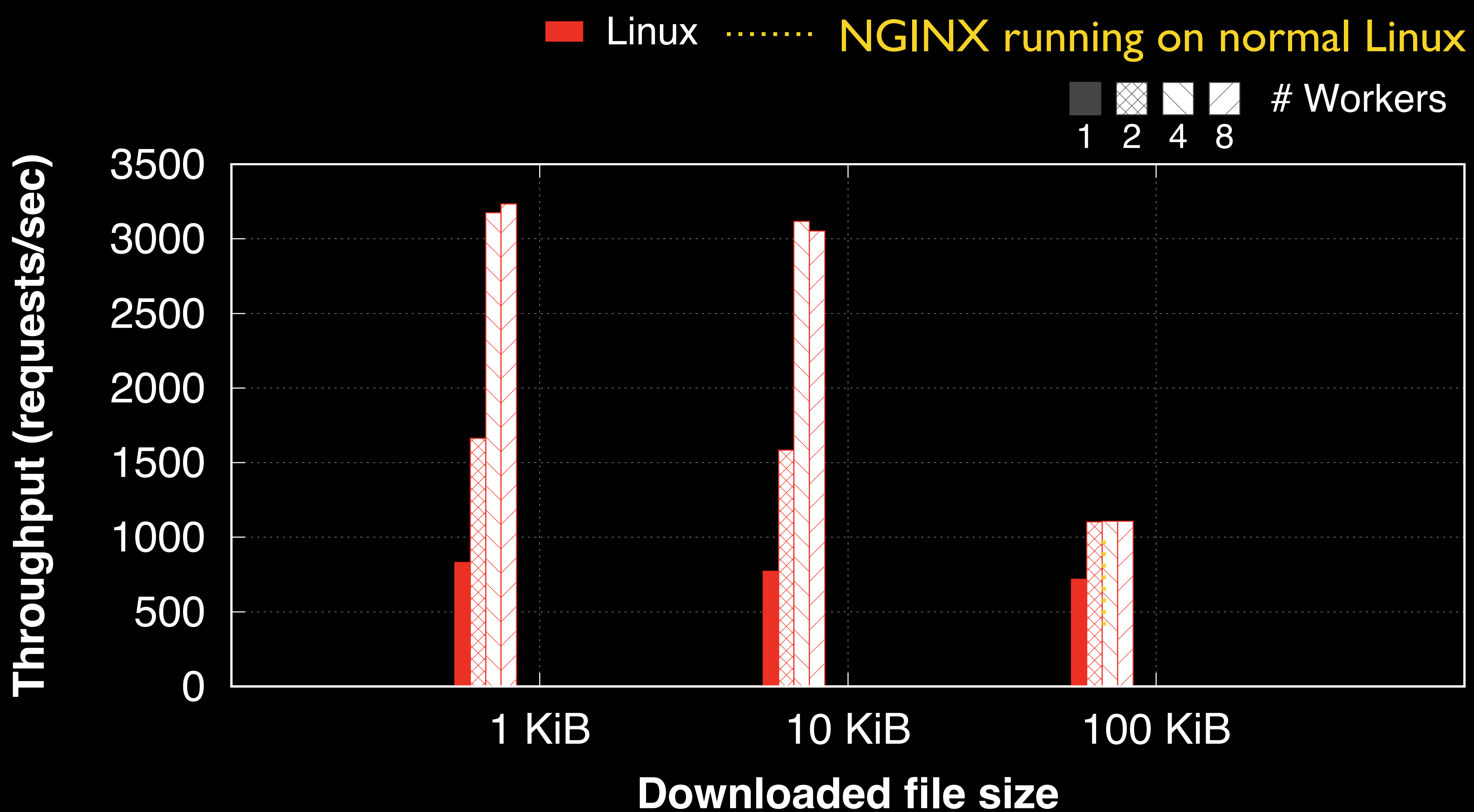
What is Phoenix's request throughput?

Fetch a file 10,000 times over non-persistent HTTPS connections from among 128 concurrent clients



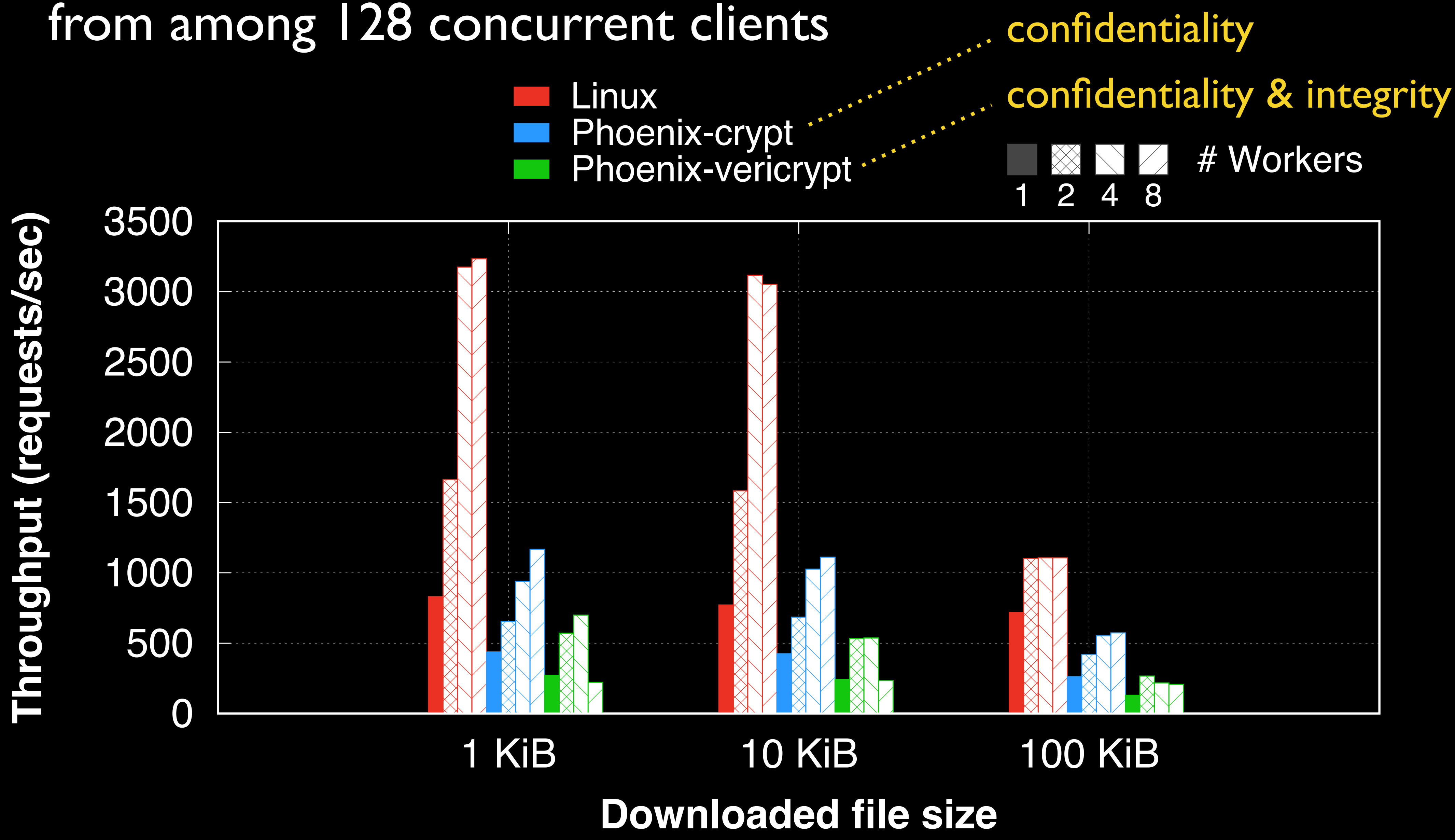
What is Phoenix's request throughput?

Fetch a file 10,000 times over non-persistent HTTPS connections from among 128 concurrent clients

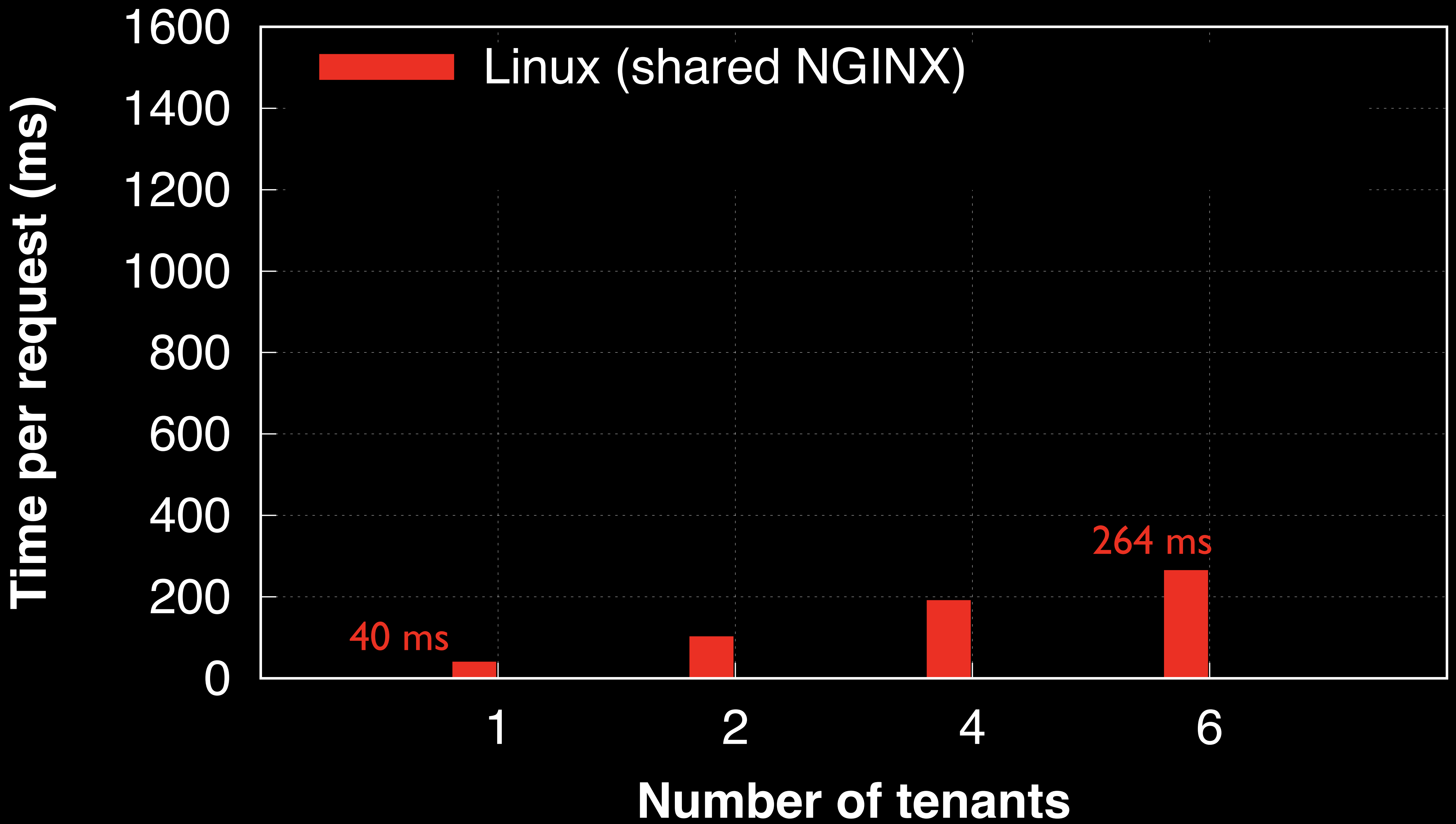


What is Phoenix's request throughput?

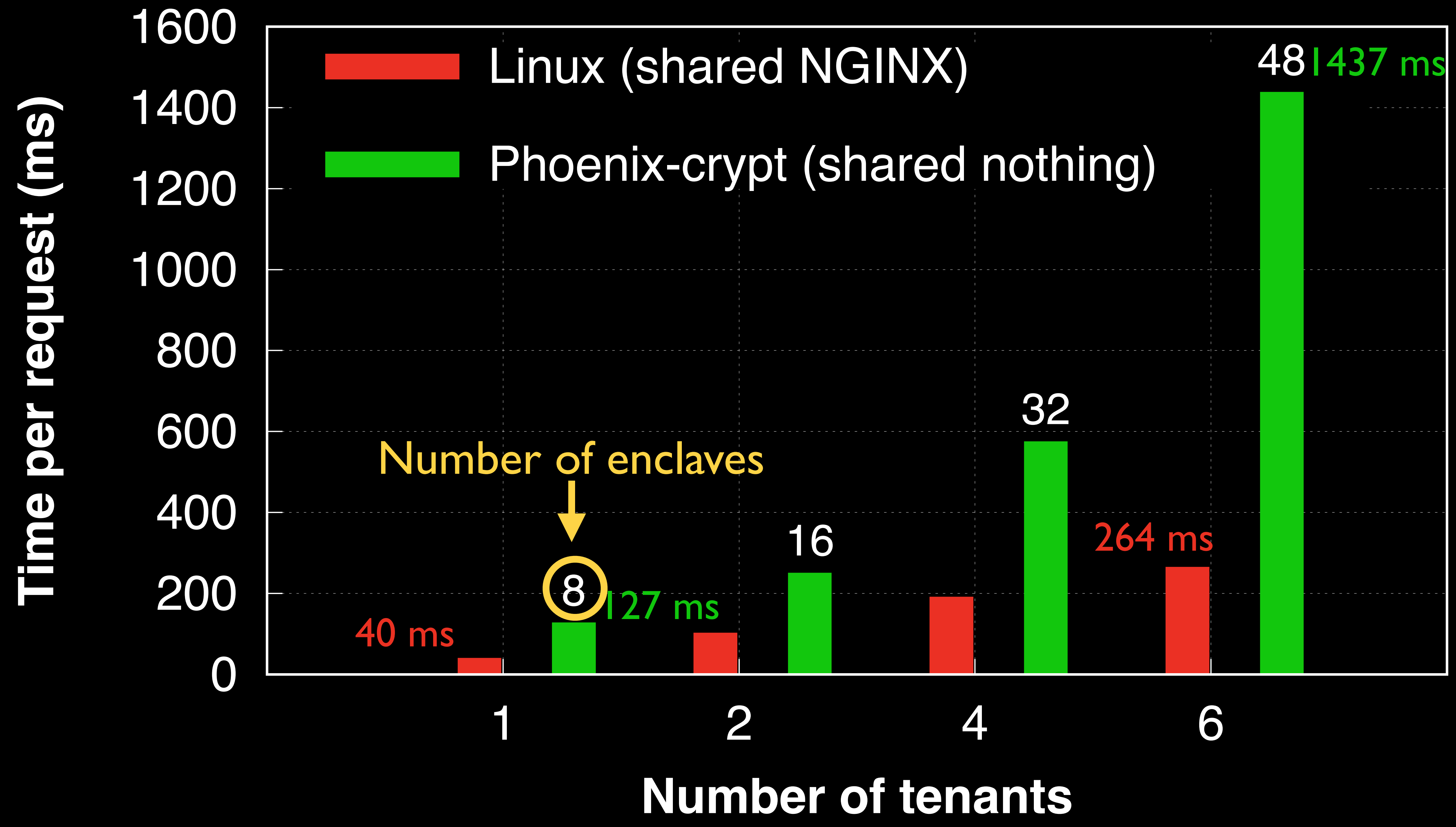
Fetch a file 10,000 times over non-persistent HTTPS connections from among 128 concurrent clients



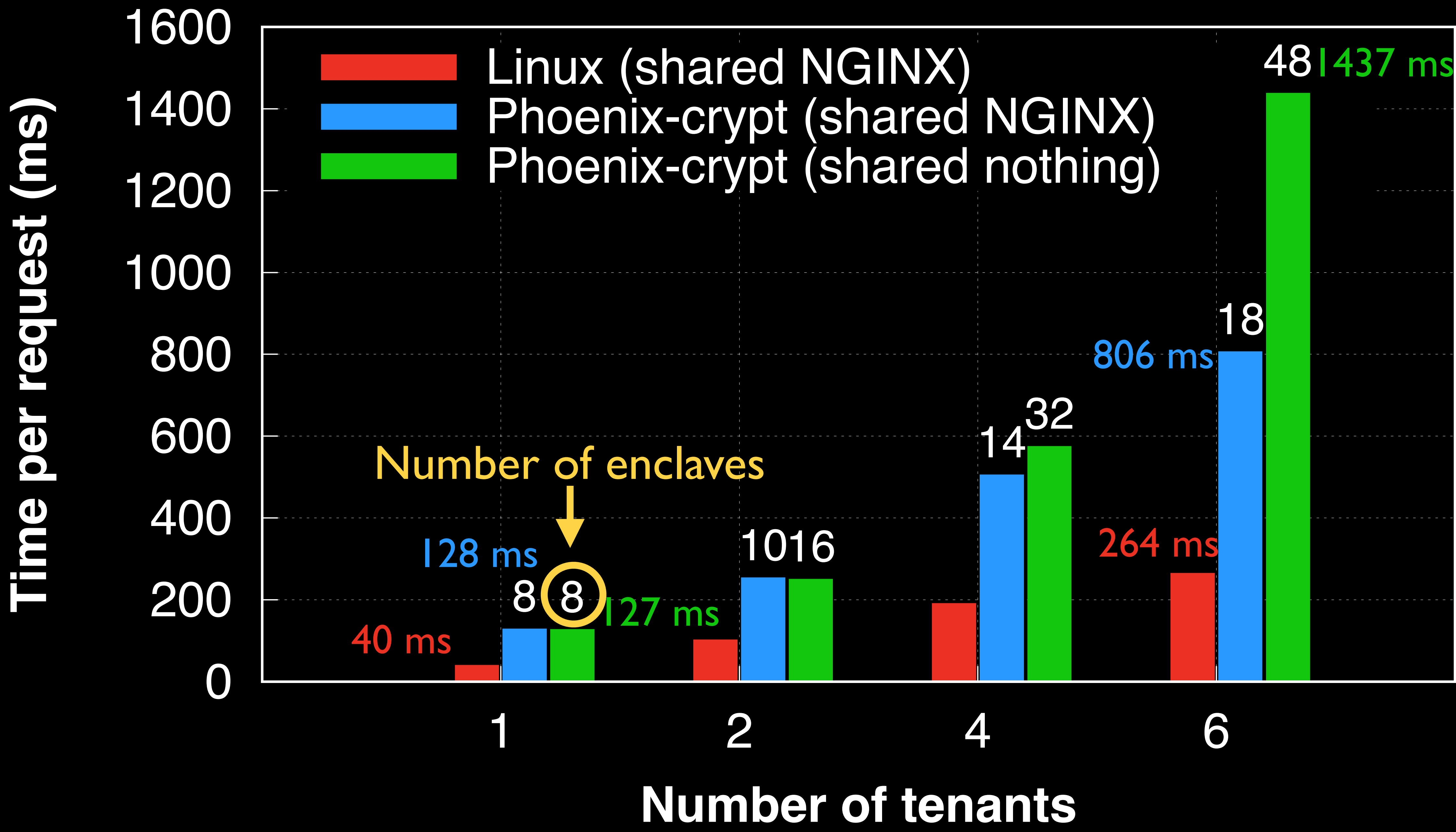
How does Phoenix scale to multiple tenants?



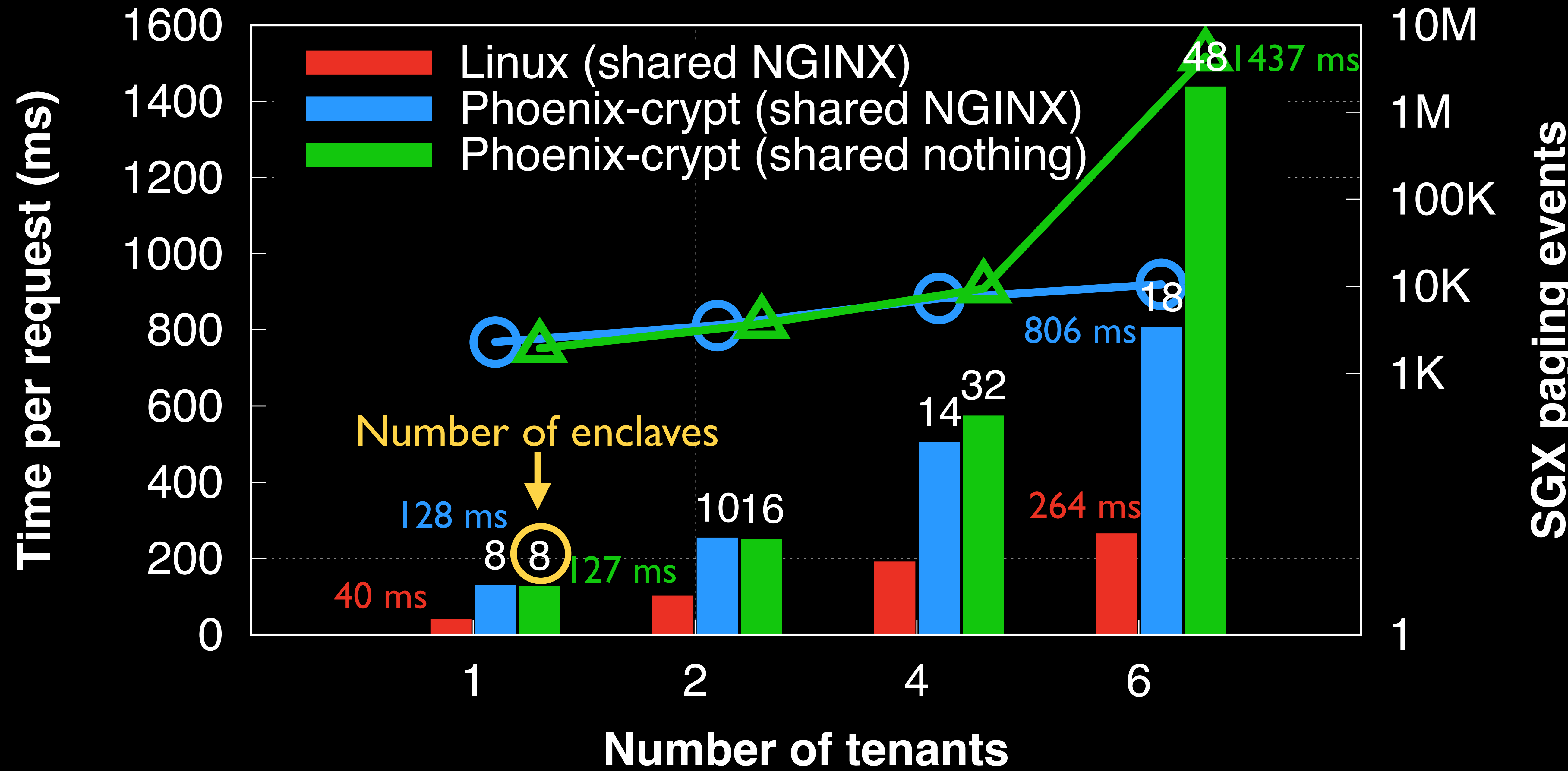
How does Phoenix scale to multiple tenants?



How does Phoenix scale to multiple tenants?



How does Phoenix scale to multiple tenants?



Other results

Benchmark overhead of running WAFs (ModSecurity) in SGX
(overhead about the same as in Linux)

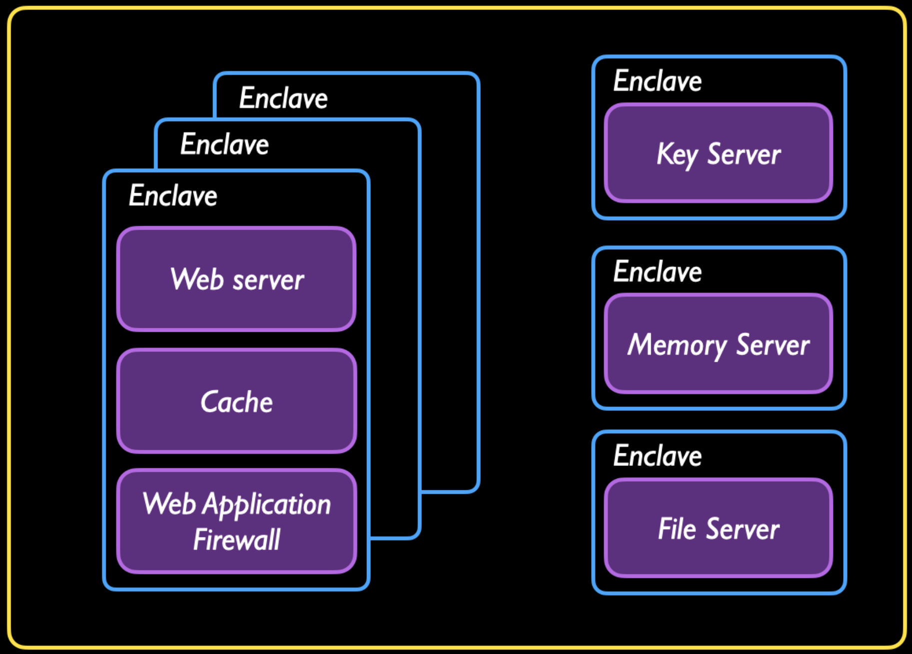
Perform detailed micro-benchmarks of each kernel server

Compare standard ocalls to exitless ocalls
(not always better)

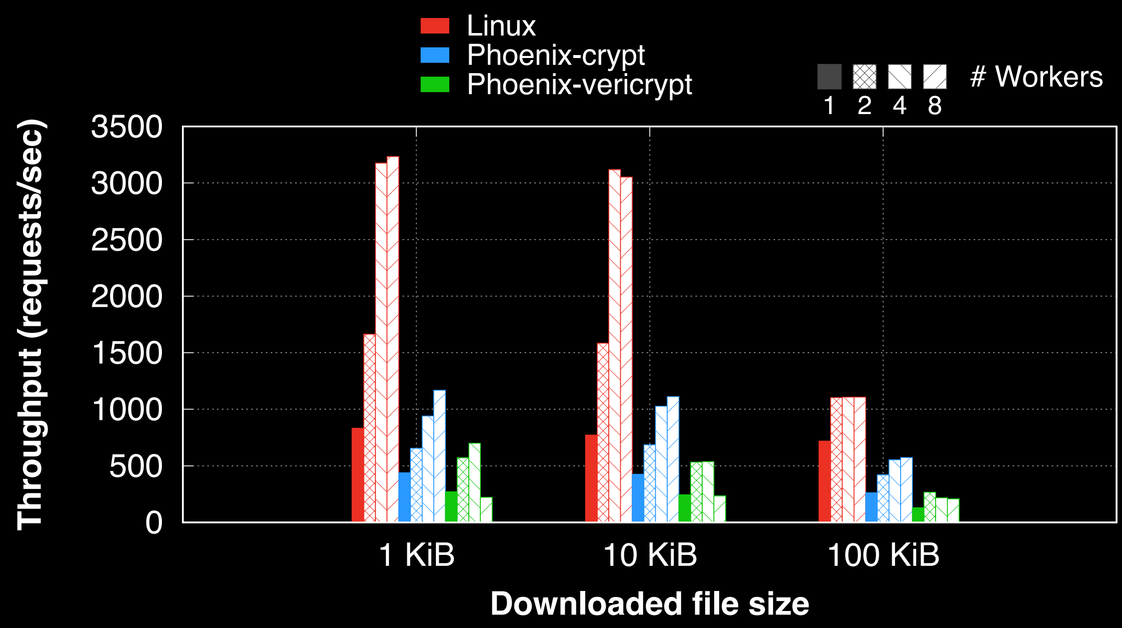
Phoenix — The first truly *keyless CDN*

Conclaves — *Containers of enclaves*

Run legacy apps in enclaves



Moderate performance overheads



https://phoenix.cs.umd.edu/

ARTIFACT EVALUATED

usenix ASSOCIATION

PASSED