Reducing The Cost of Containing:

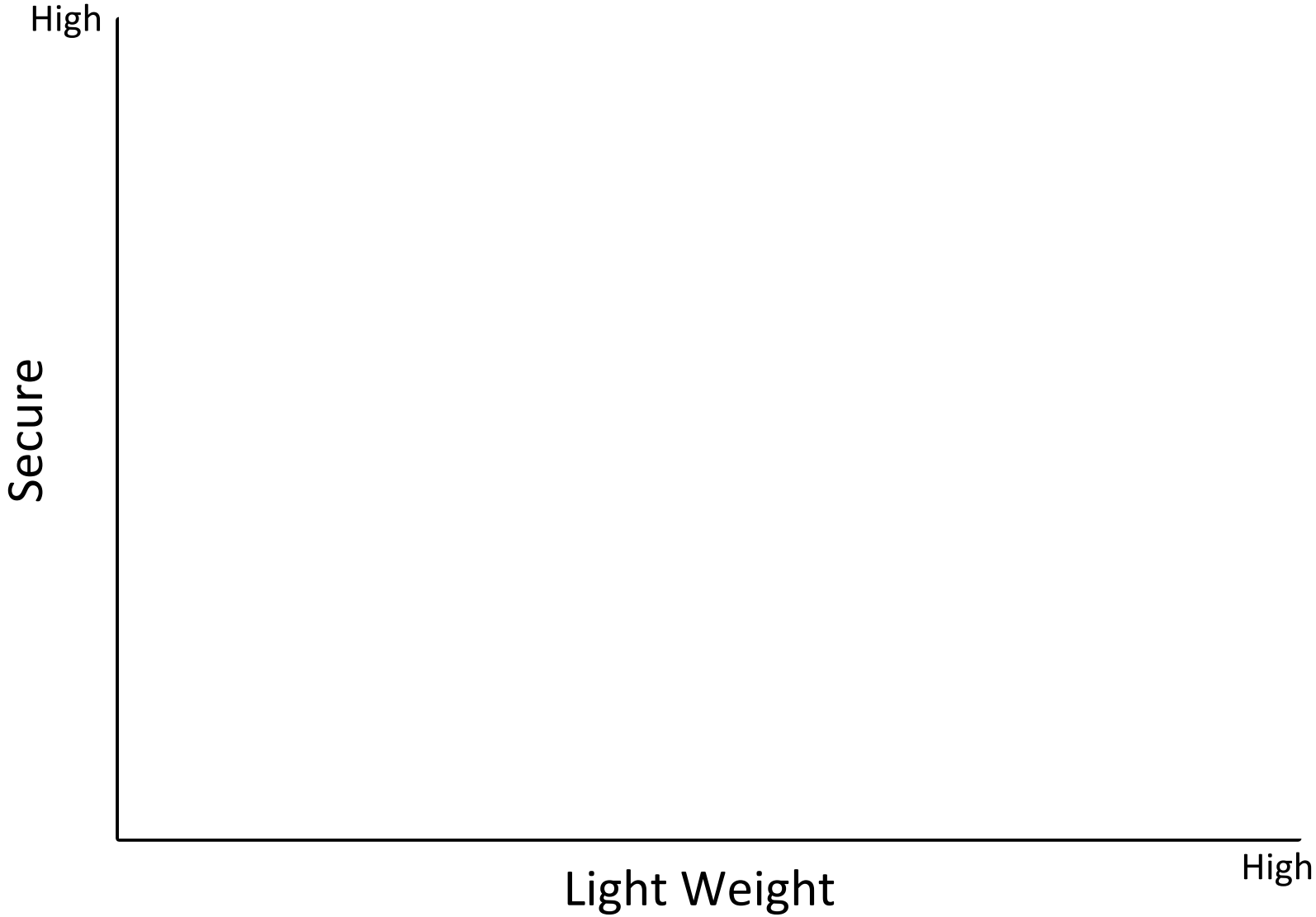
ContainerFS, A gVisor File System for Performance

Ethan G. Young, Shenghao Zou, Pengfei Zhu, Tyler Caraza-Harter, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

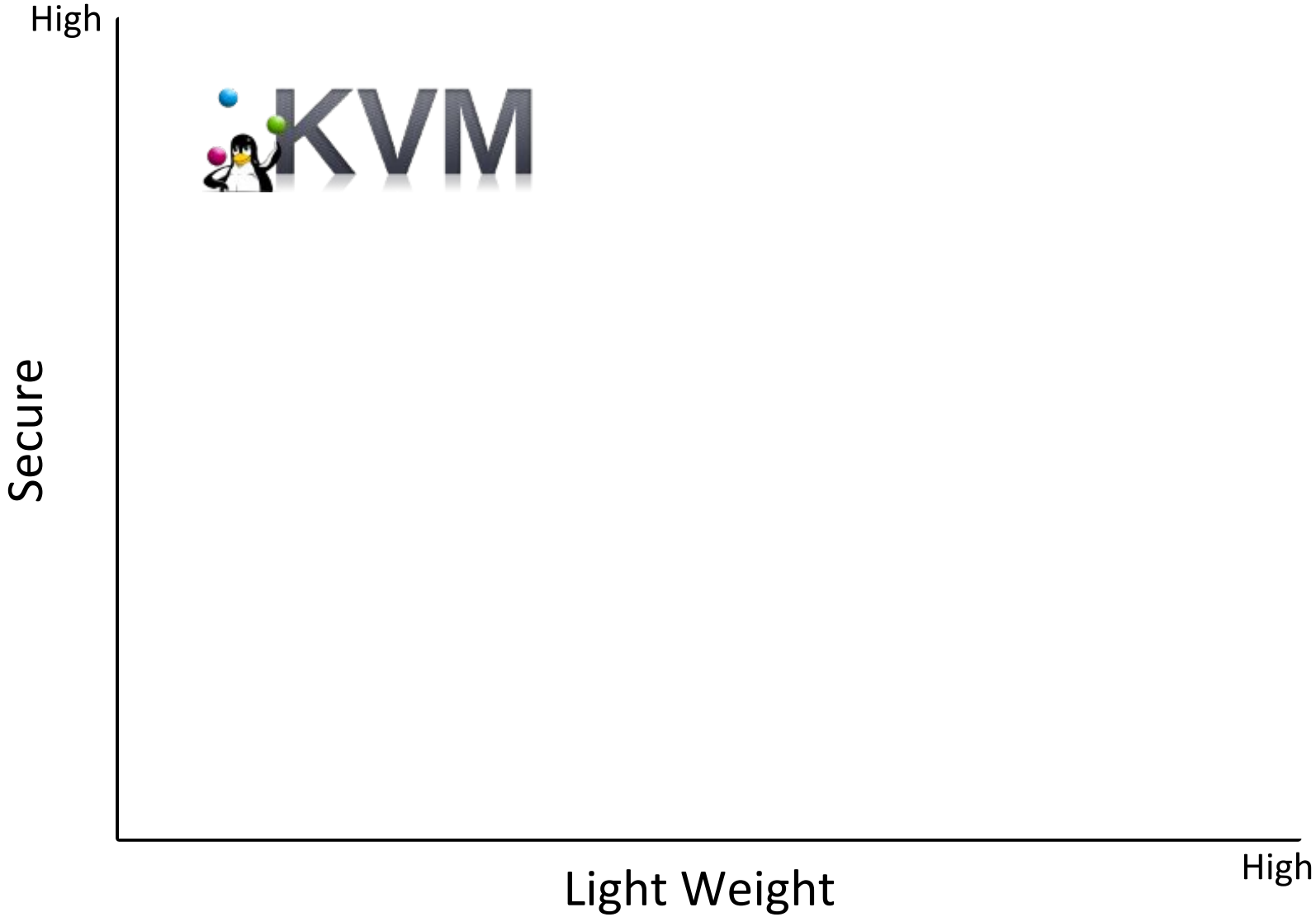
University of Wisconsin-Madison
December 5, 2019

gVisor

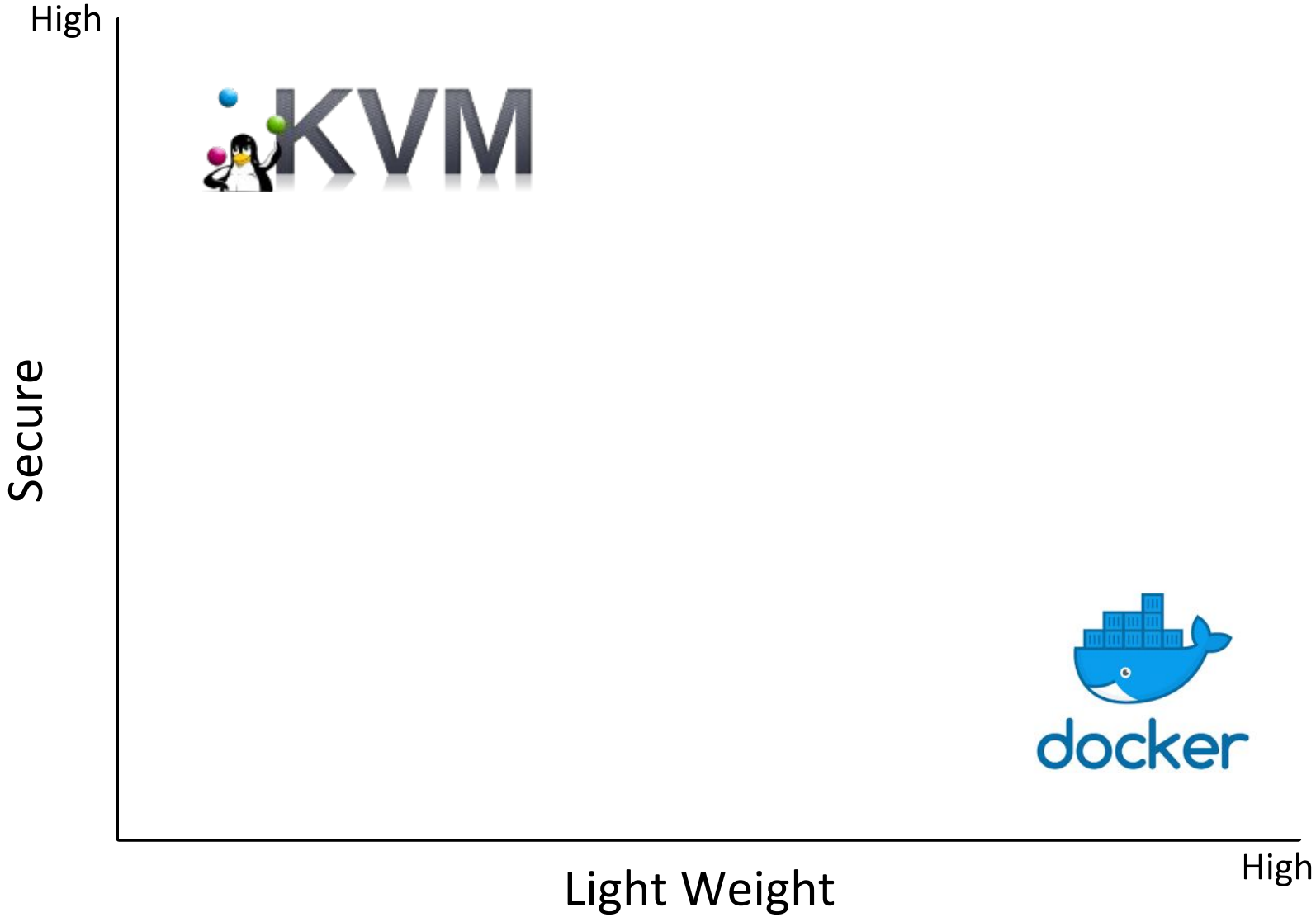
Developers wish to get the security of Virtual Machines and the light-weight characteristics of containers



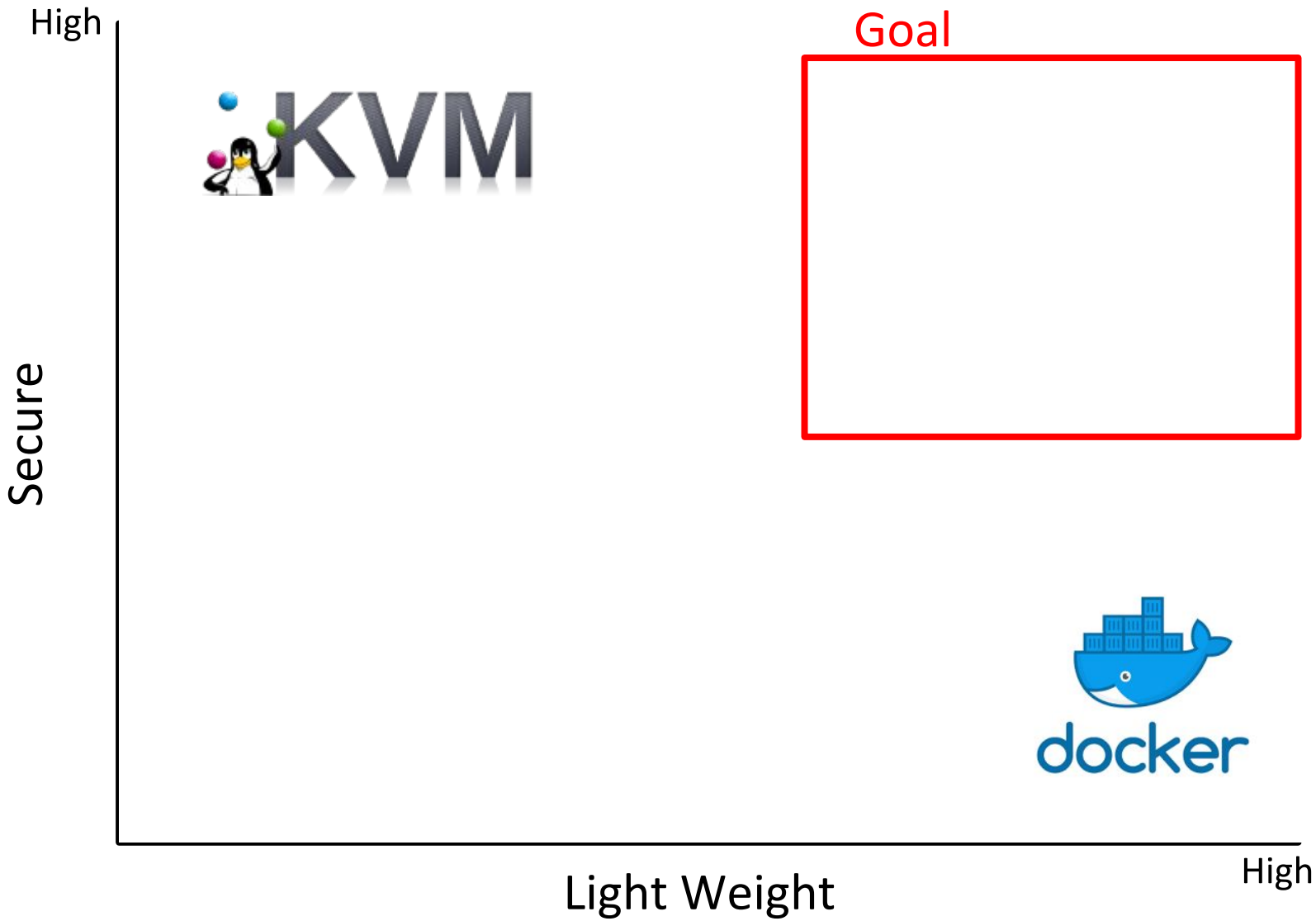
Developers wish to get the security of Virtual Machines and the light-weight characteristics of containers



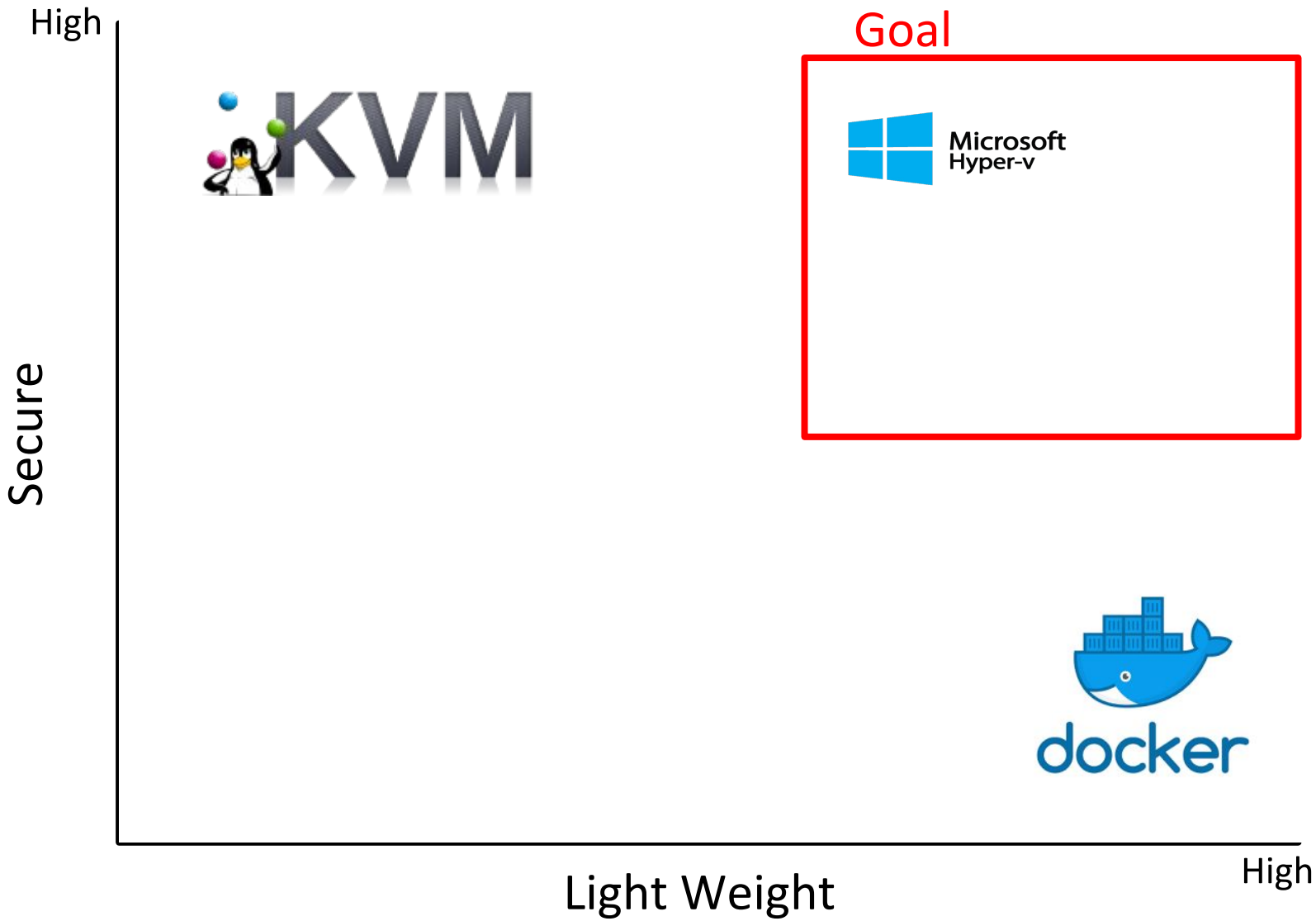
Developers wish to get the security of Virtual Machines and the light-weight characteristics of containers



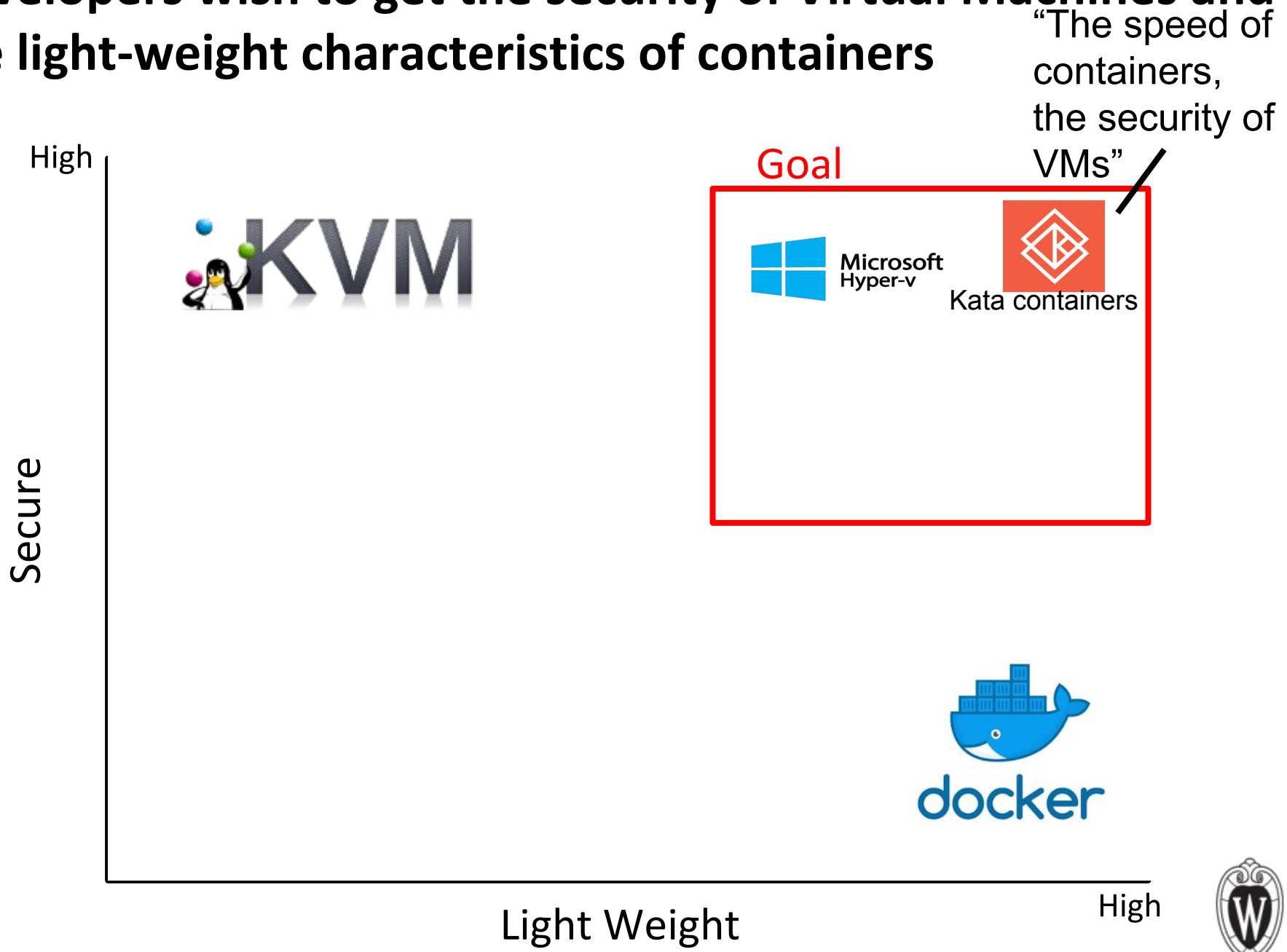
Developers wish to get the security of Virtual Machines and the light-weight characteristics of containers



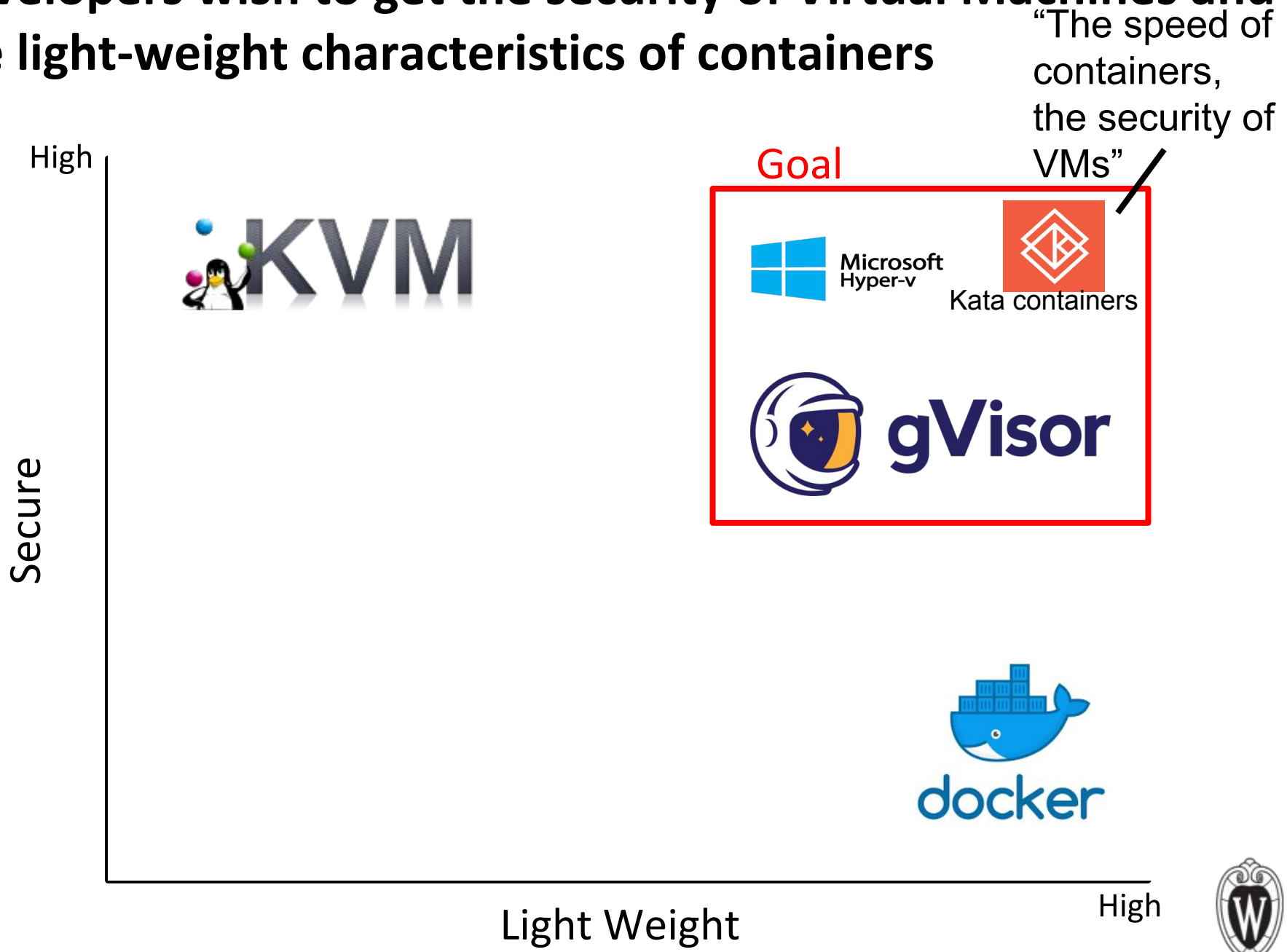
Developers wish to get the security of Virtual Machines and the light-weight characteristics of containers



Developers wish to get the security of Virtual Machines and the light-weight characteristics of containers



Developers wish to get the security of Virtual Machines and the light-weight characteristics of containers



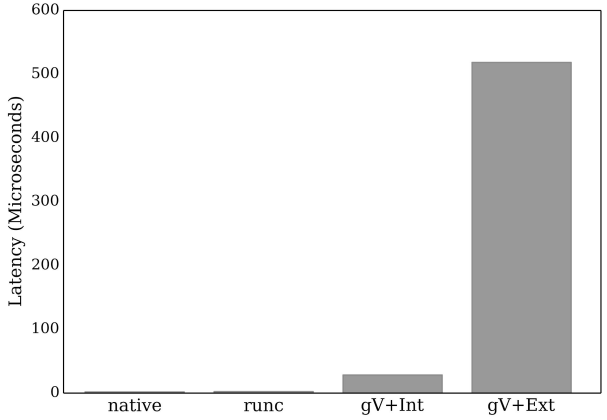
gVisor is an interesting case study because it backs Google's serverless platform and is open-source

gVisor

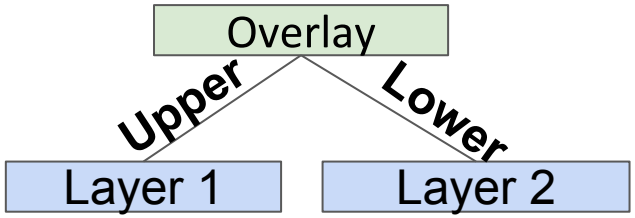
This talk analyzes gVisor, the platform backing Google Cloud Functions, and offers a file system to improve performance

gVisor

gVisor Architecture



gVisor Performance Analysis

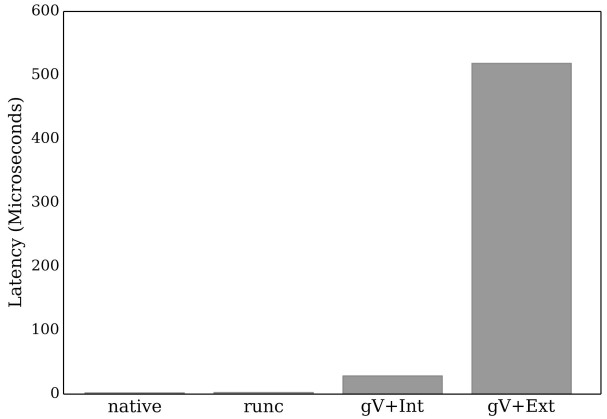


ContainerFS Overview

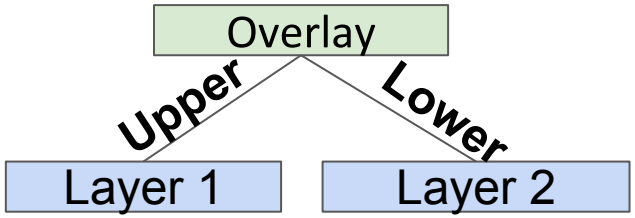
This talk analyzes gVisor, the platform backing Google Cloud Functions, and offers a file system to improve performance

gVisor

[gVisor Architecture]

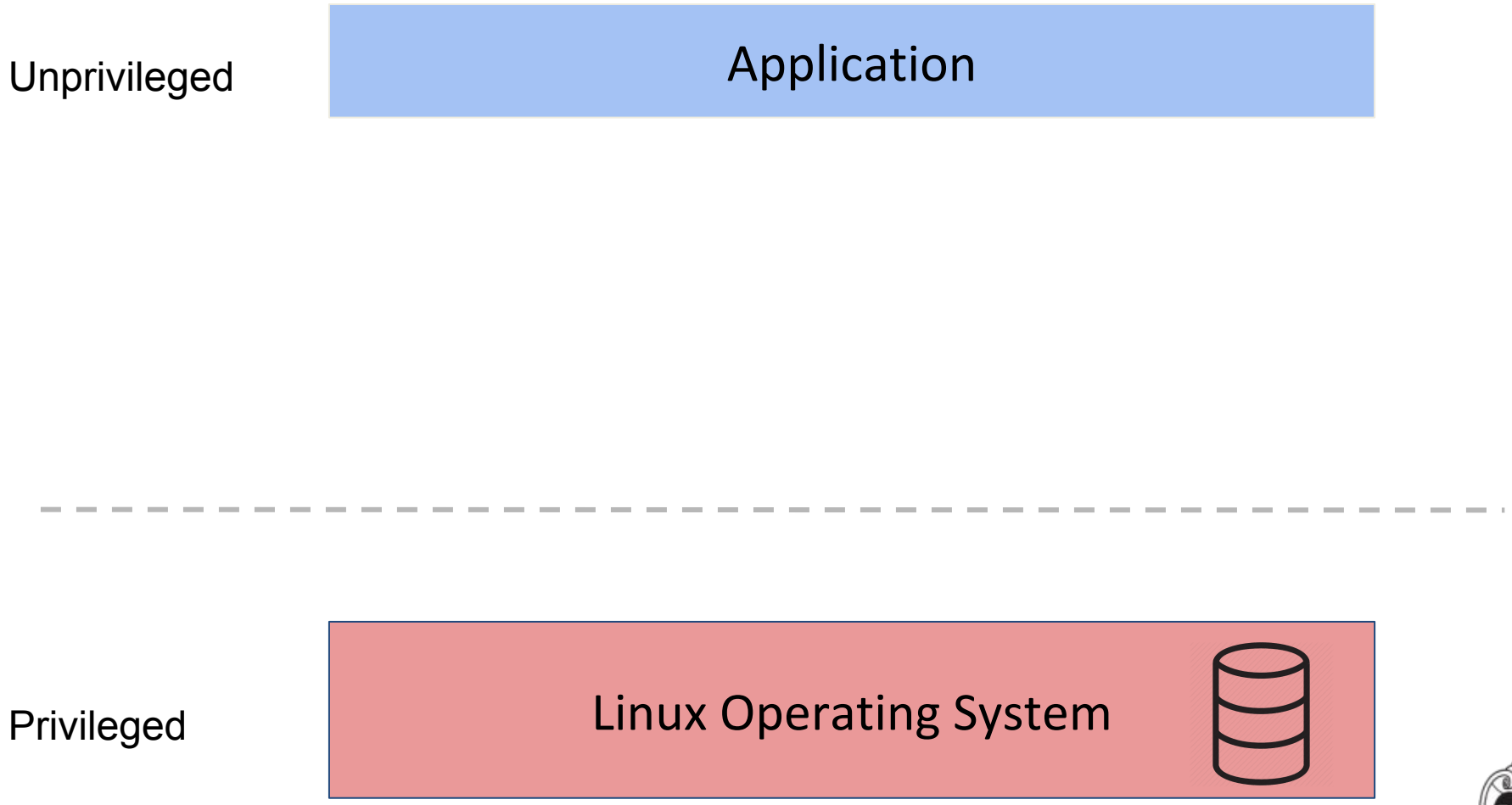


gVisor Performance Analysis

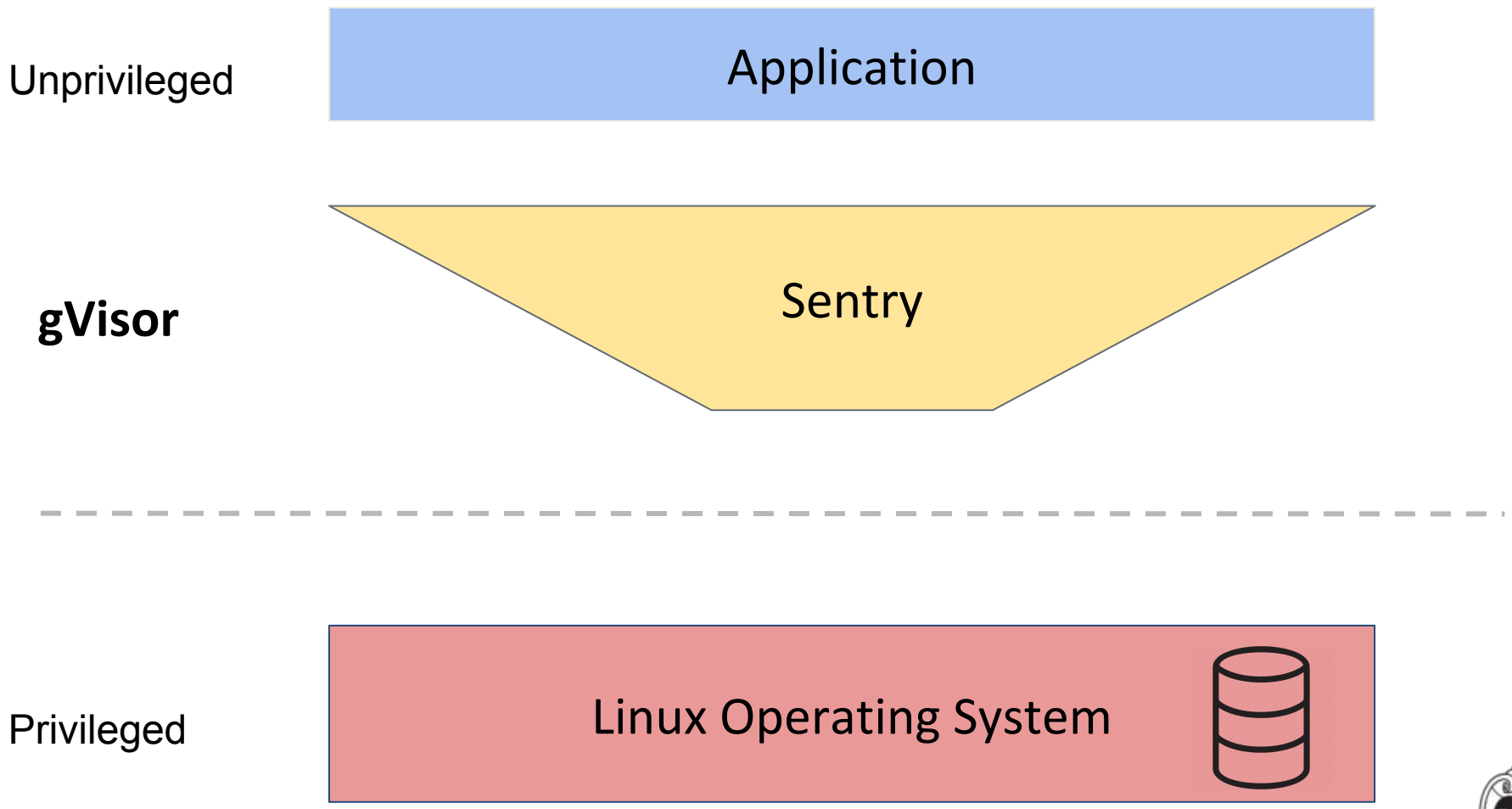


ContainerFS Overview

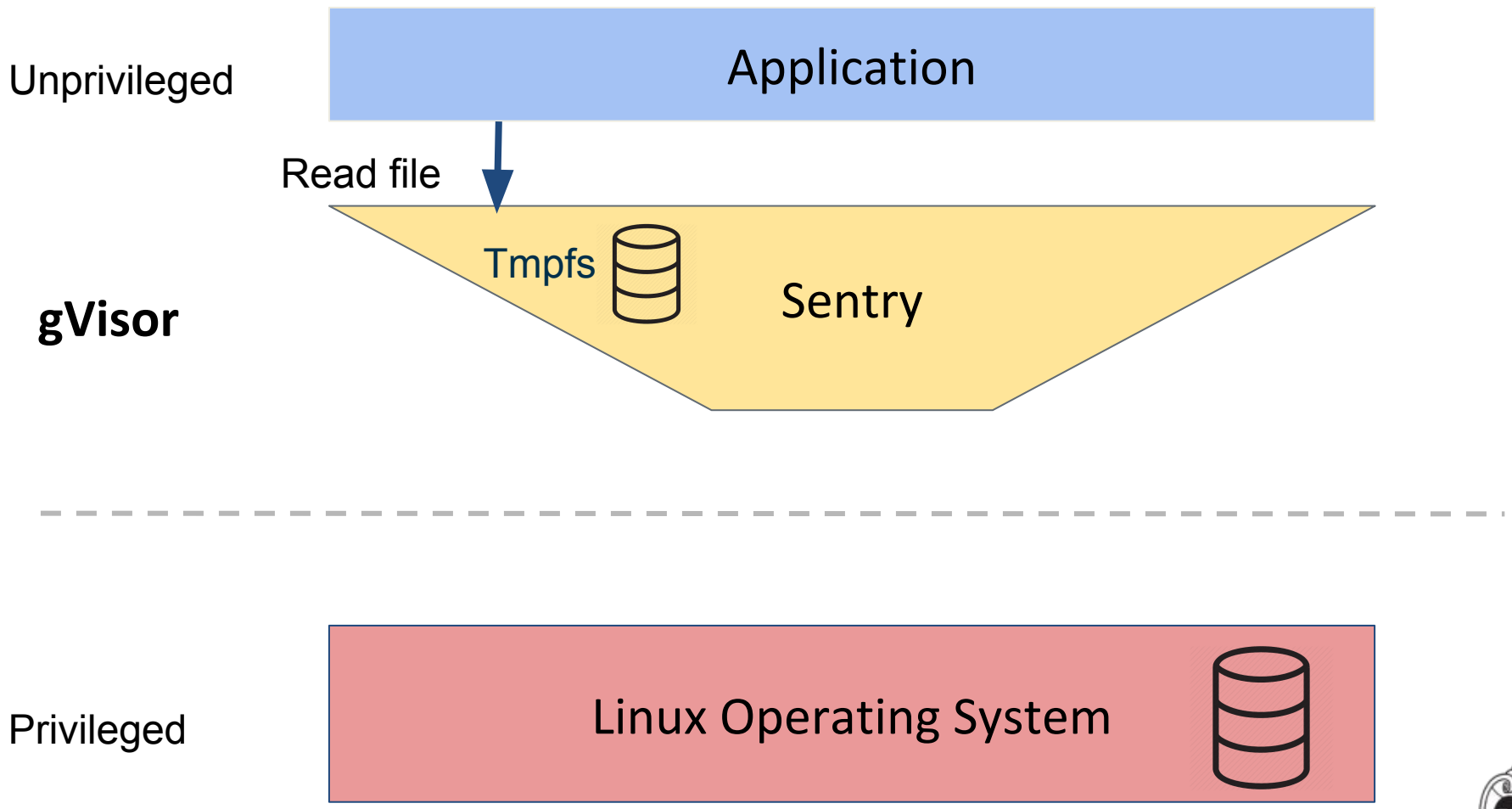
Even in a container, applications typically have direct access to the entire Linux system call interface



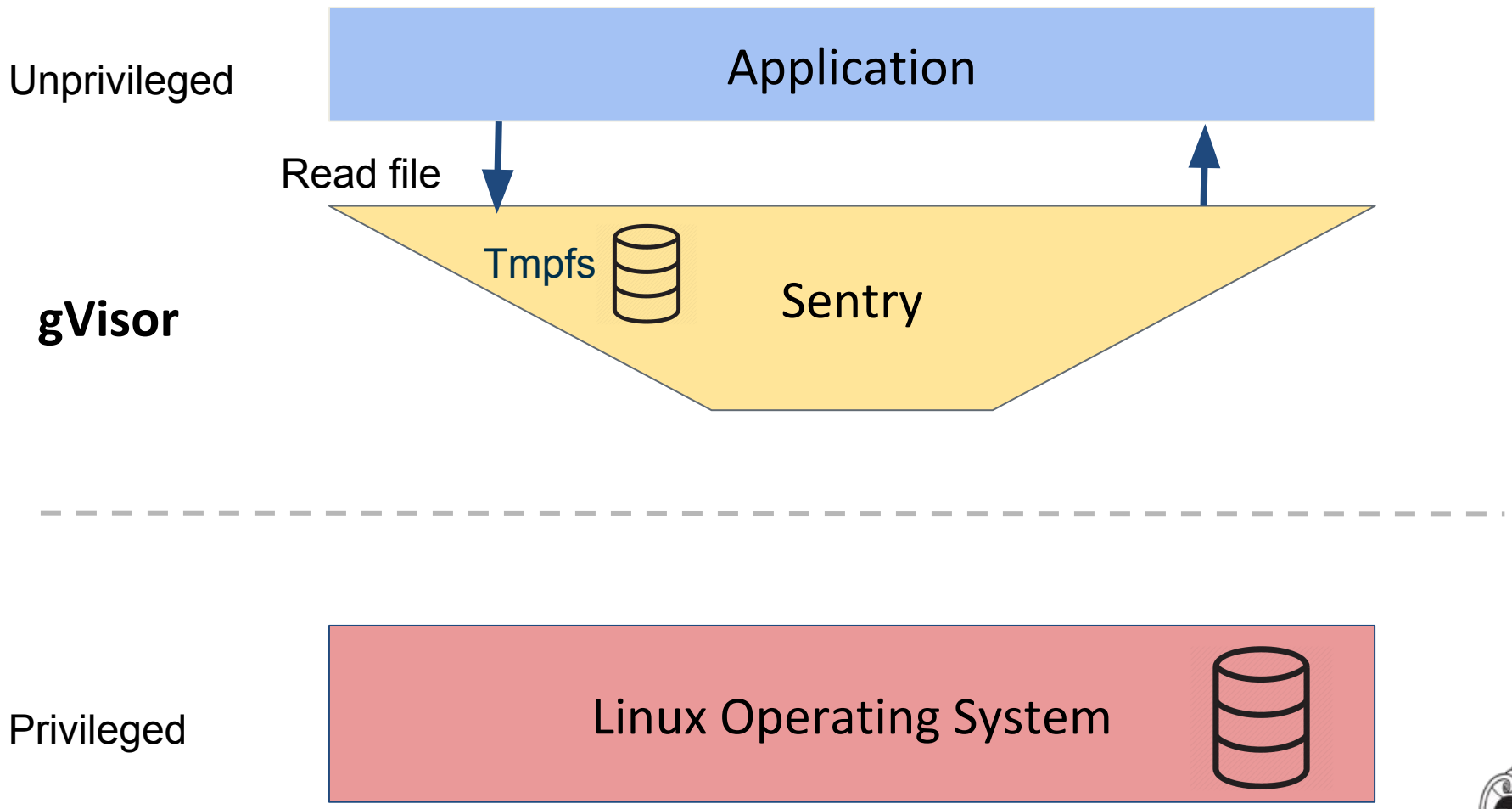
gVisor is able to intercept system calls and handle them without host intervention



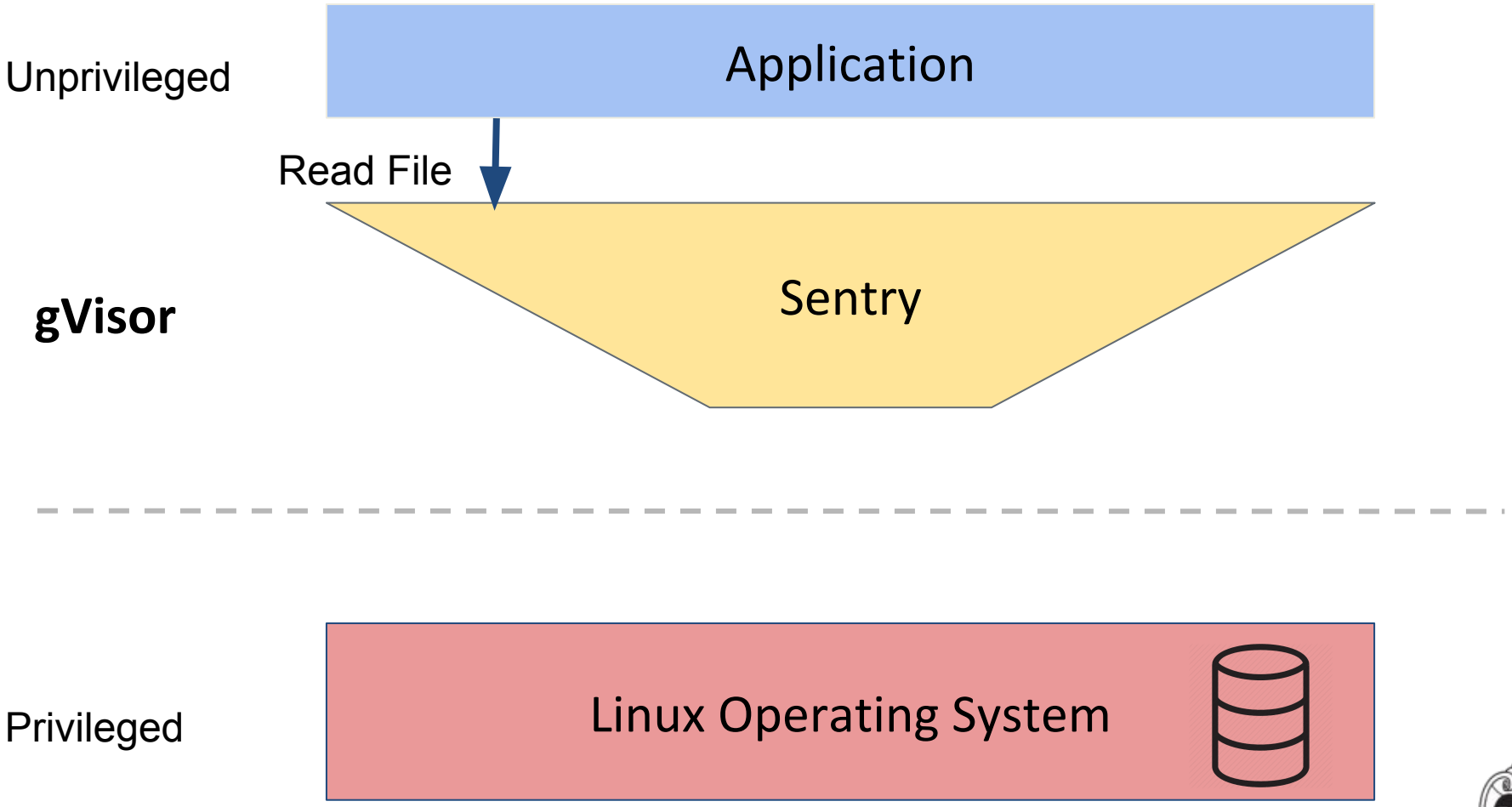
gVisor is able to intercept system calls and handle them without host intervention



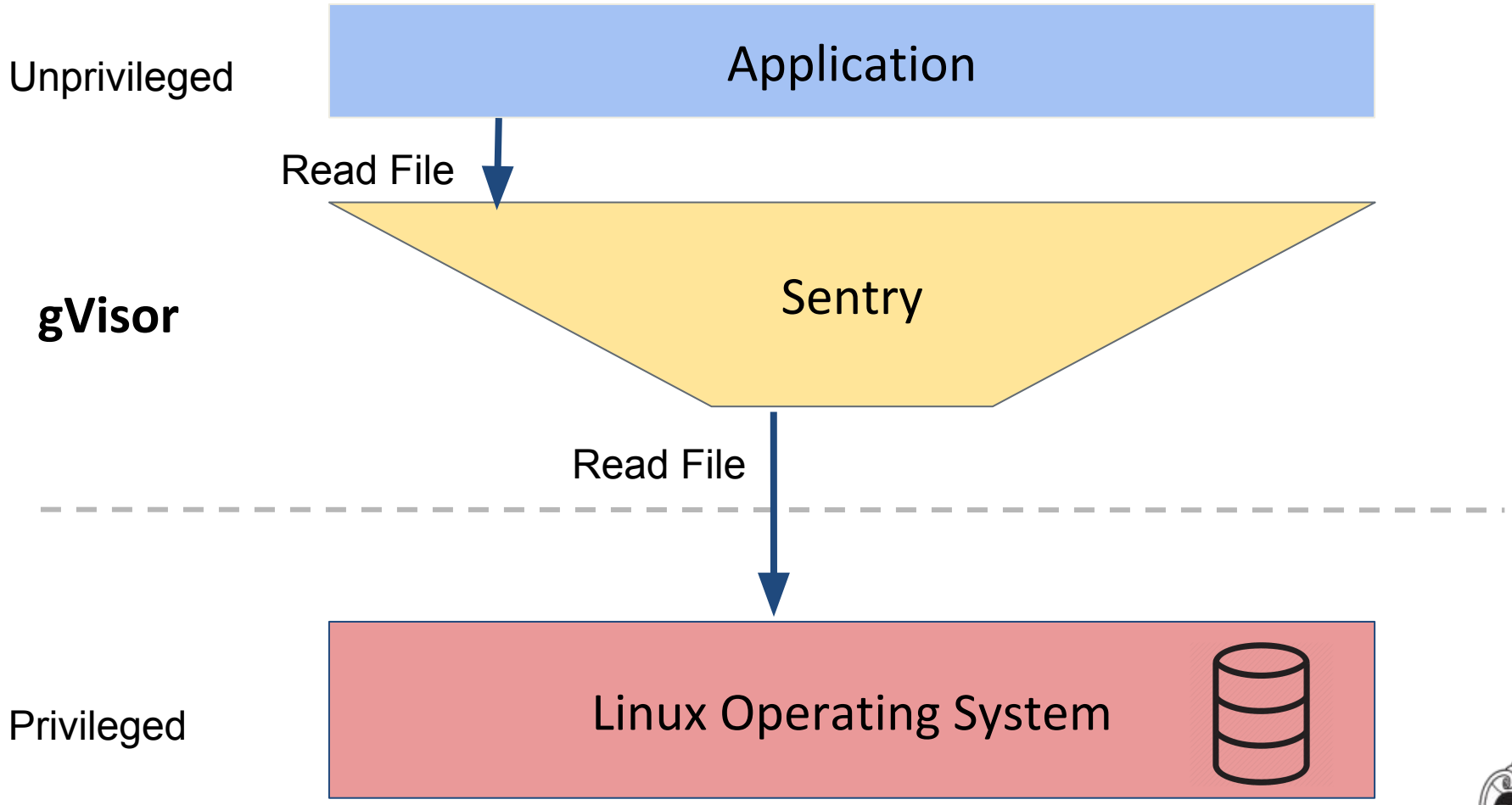
gVisor is able to intercept system calls and handle them without host intervention



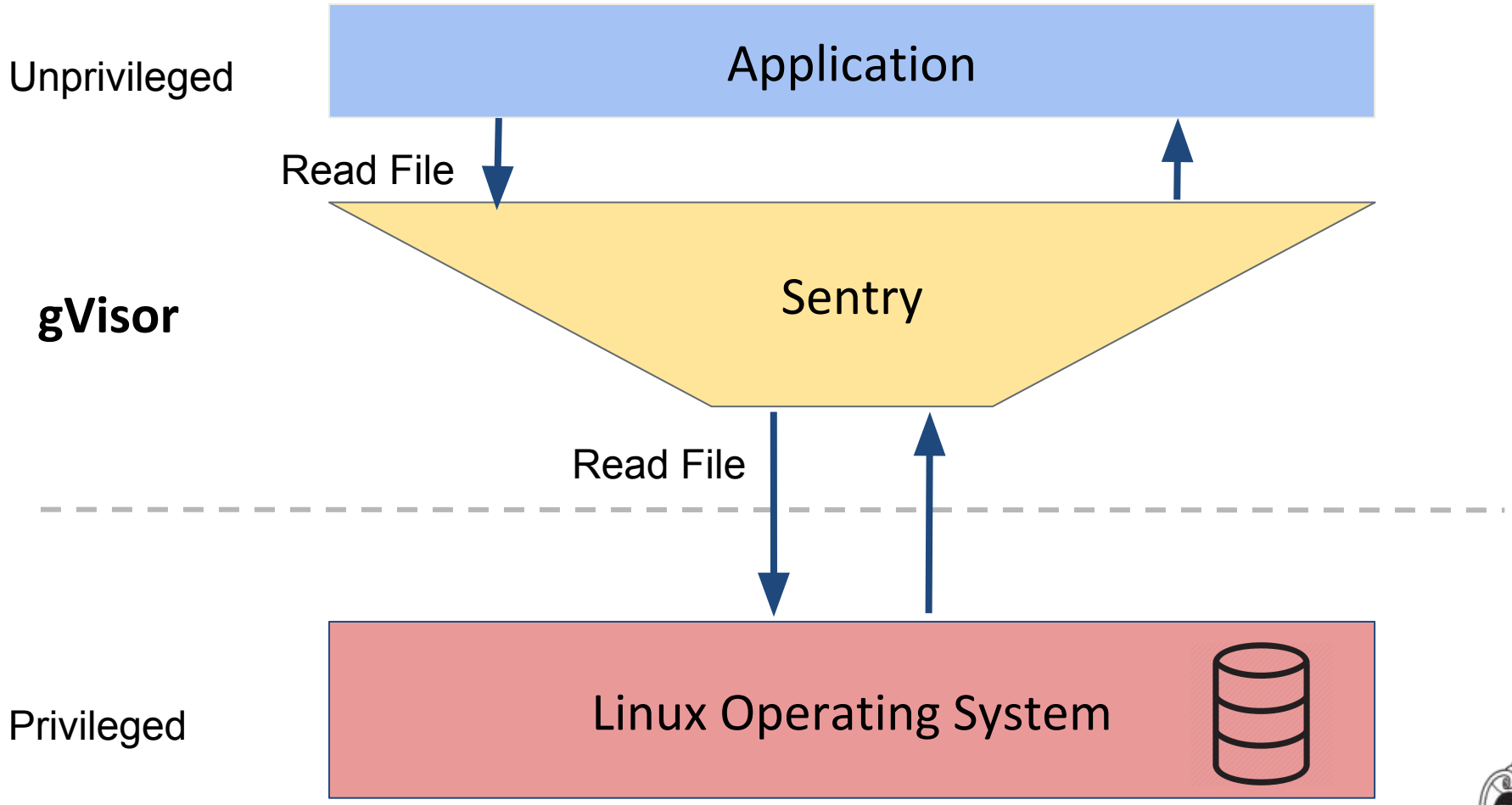
gVisor is able to forward a subset of system calls to the host



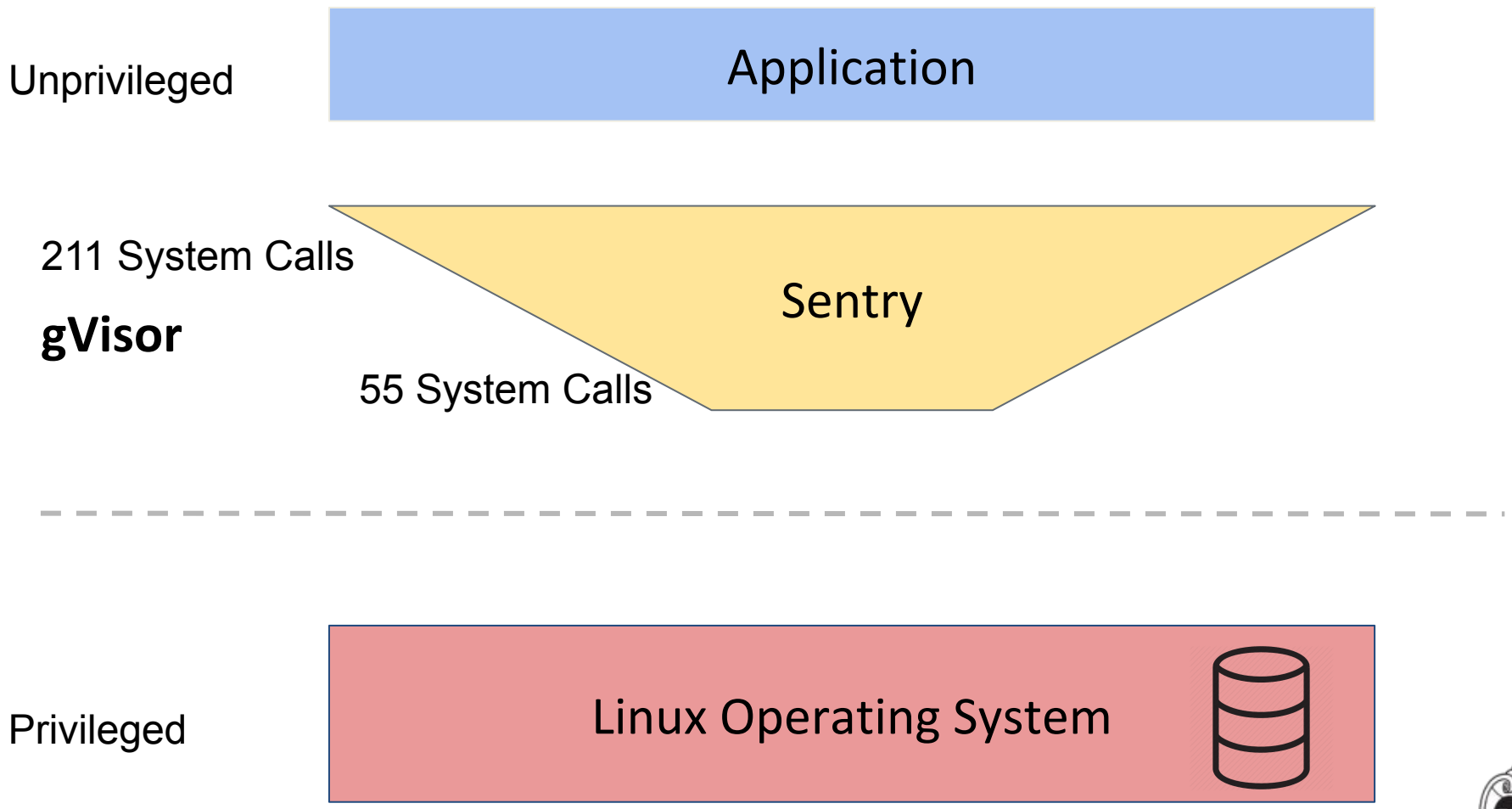
gVisor is able to forward a subset of system calls to the host



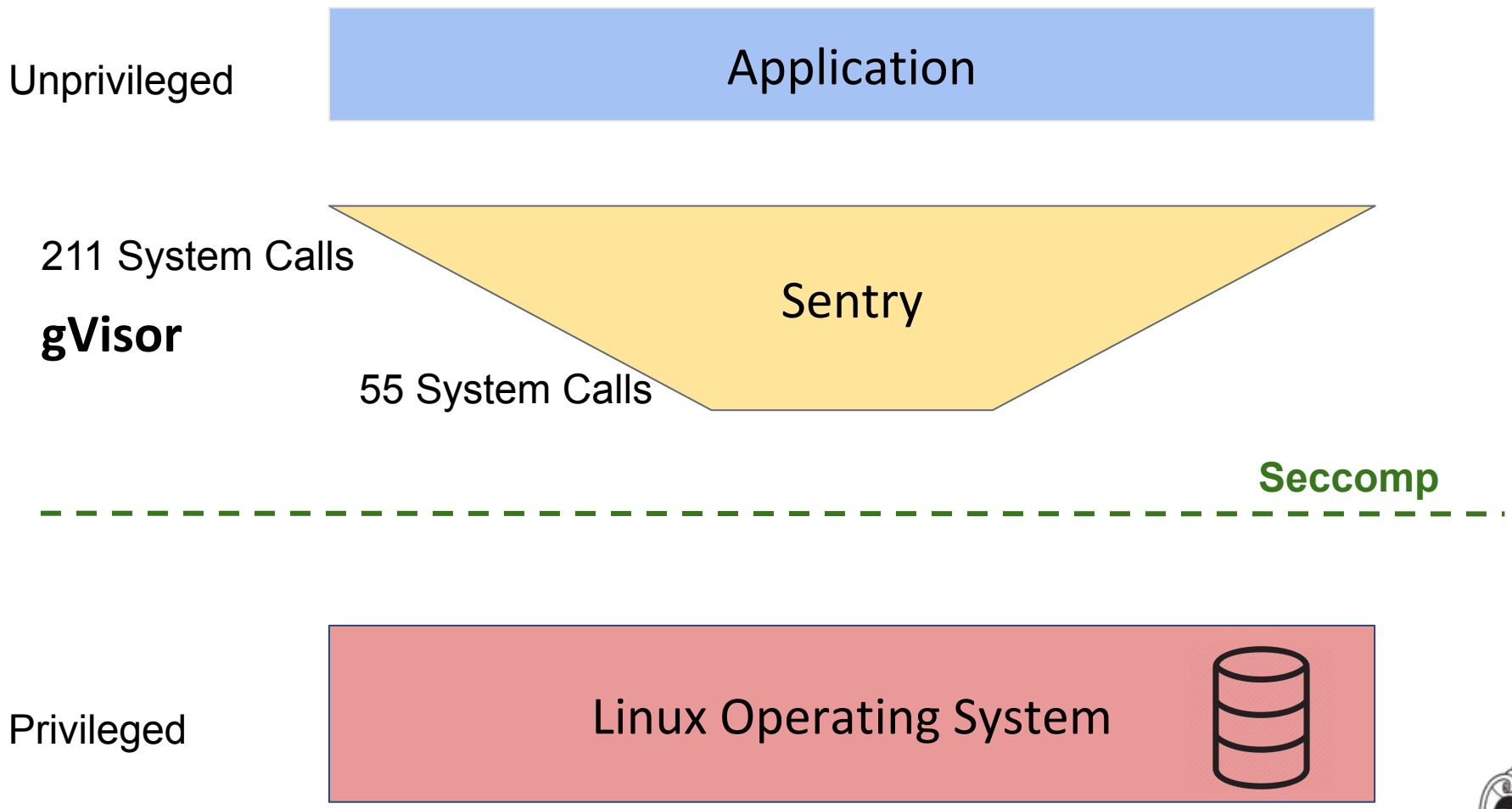
gVisor is able to forward a subset of system calls to the host



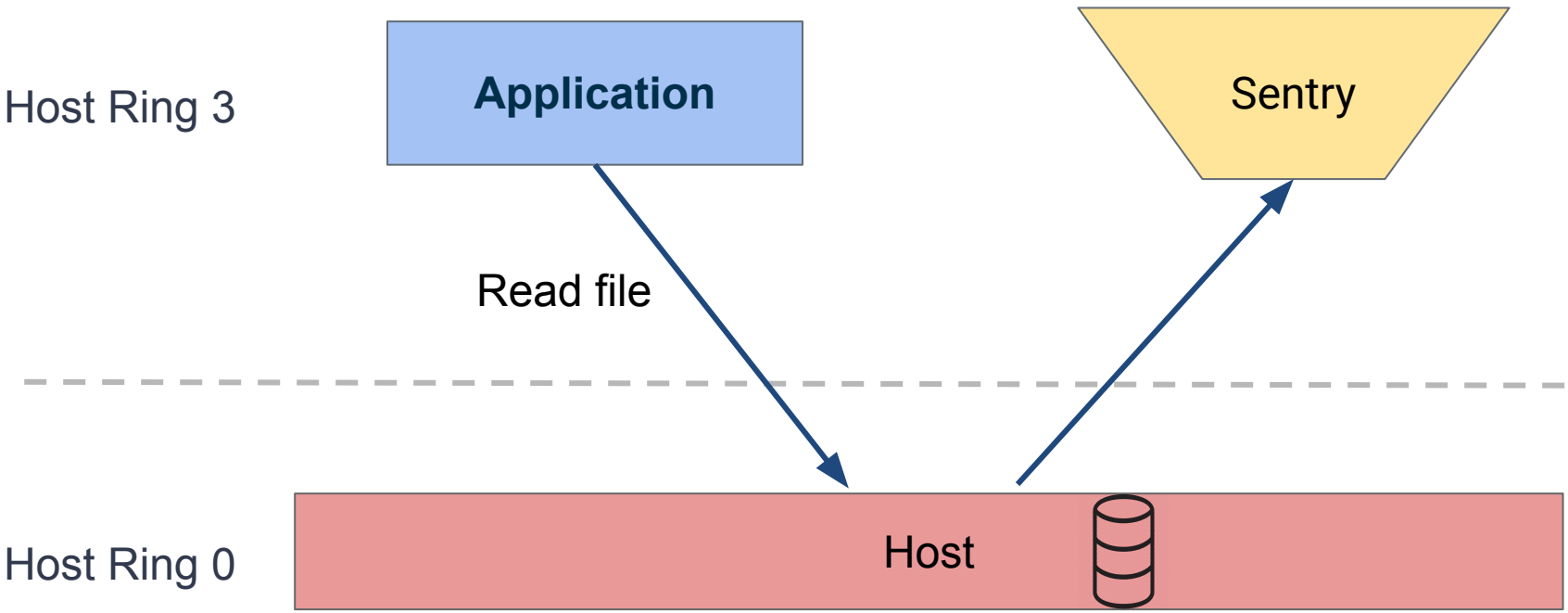
gVisor drastically reduces the attack surface of an application to the host operating system



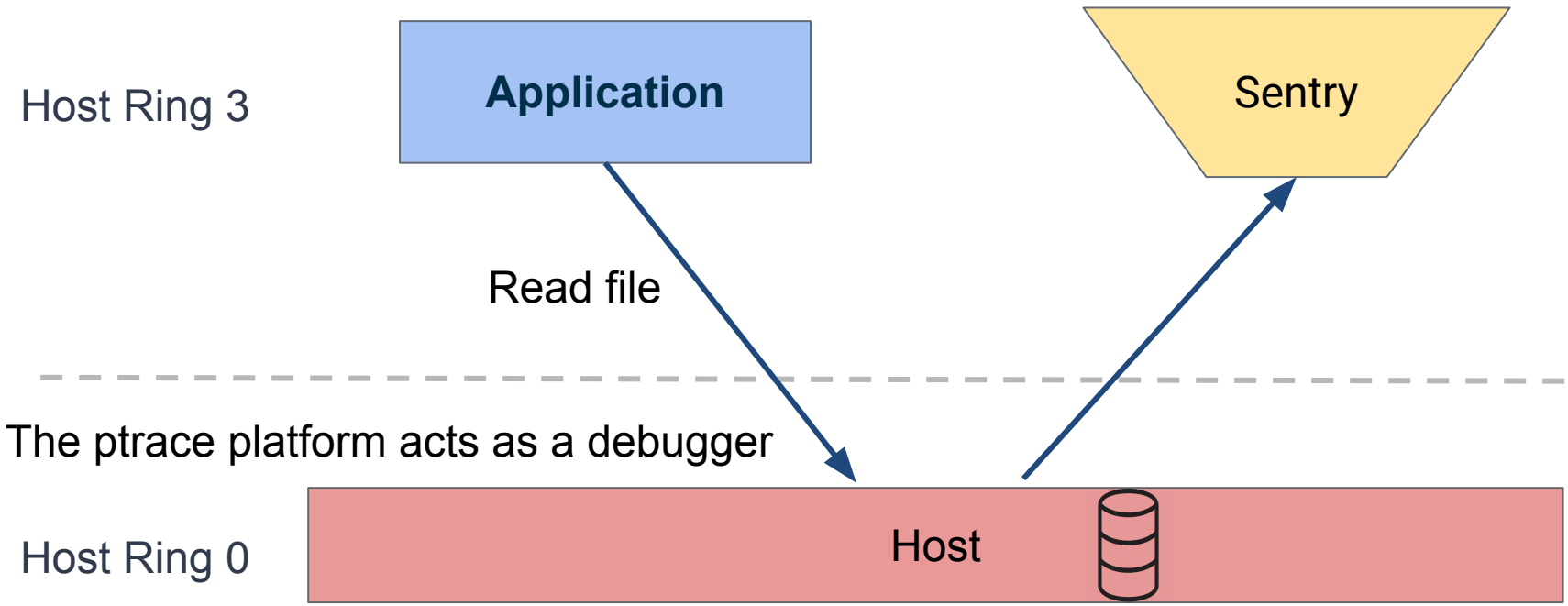
gVisor drastically reduces the attack surface of an application to the host operating system



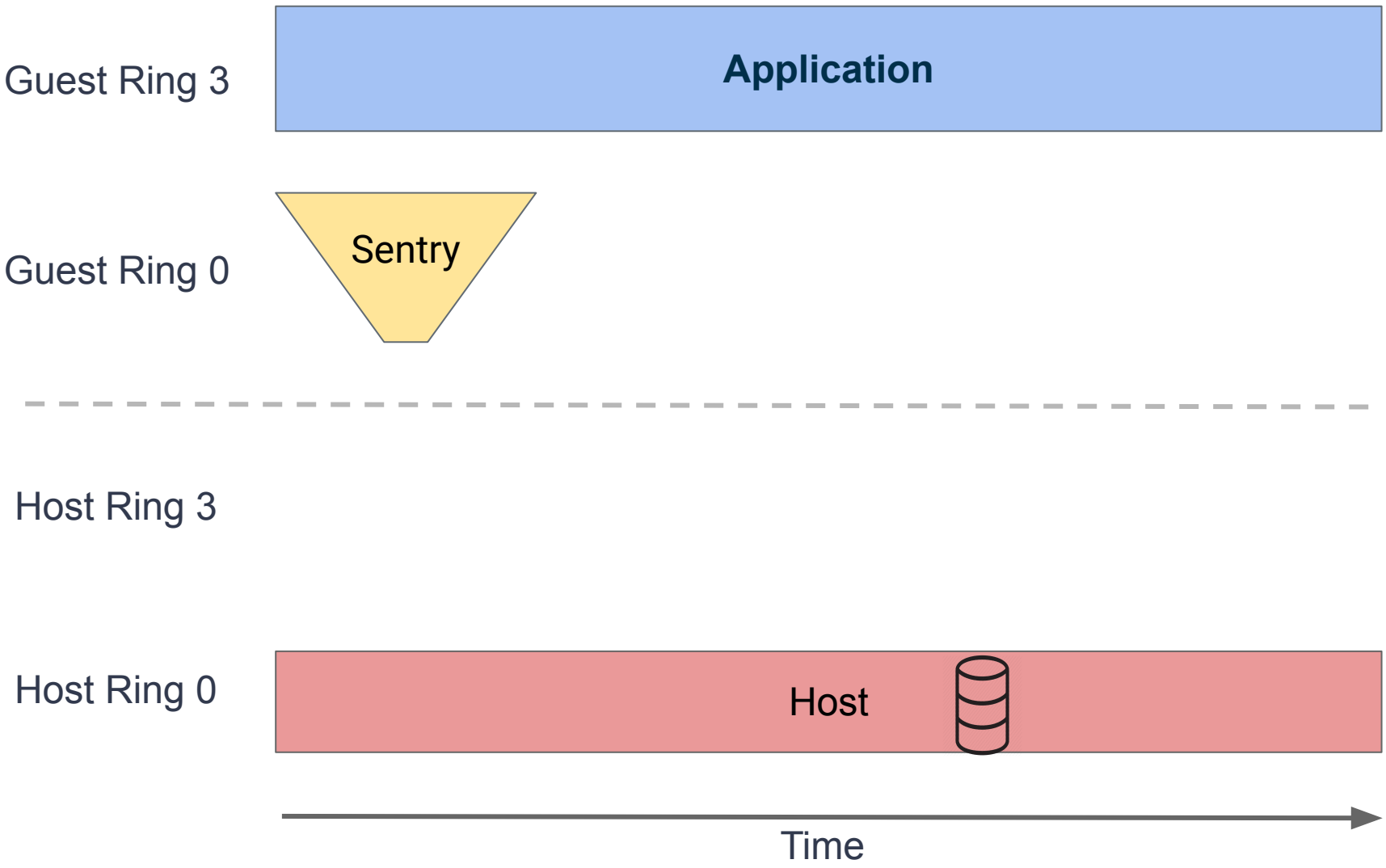
To intercept a system call, gVisor utilizes one of two platforms (ptrace or KVM) to transfer control to sentry



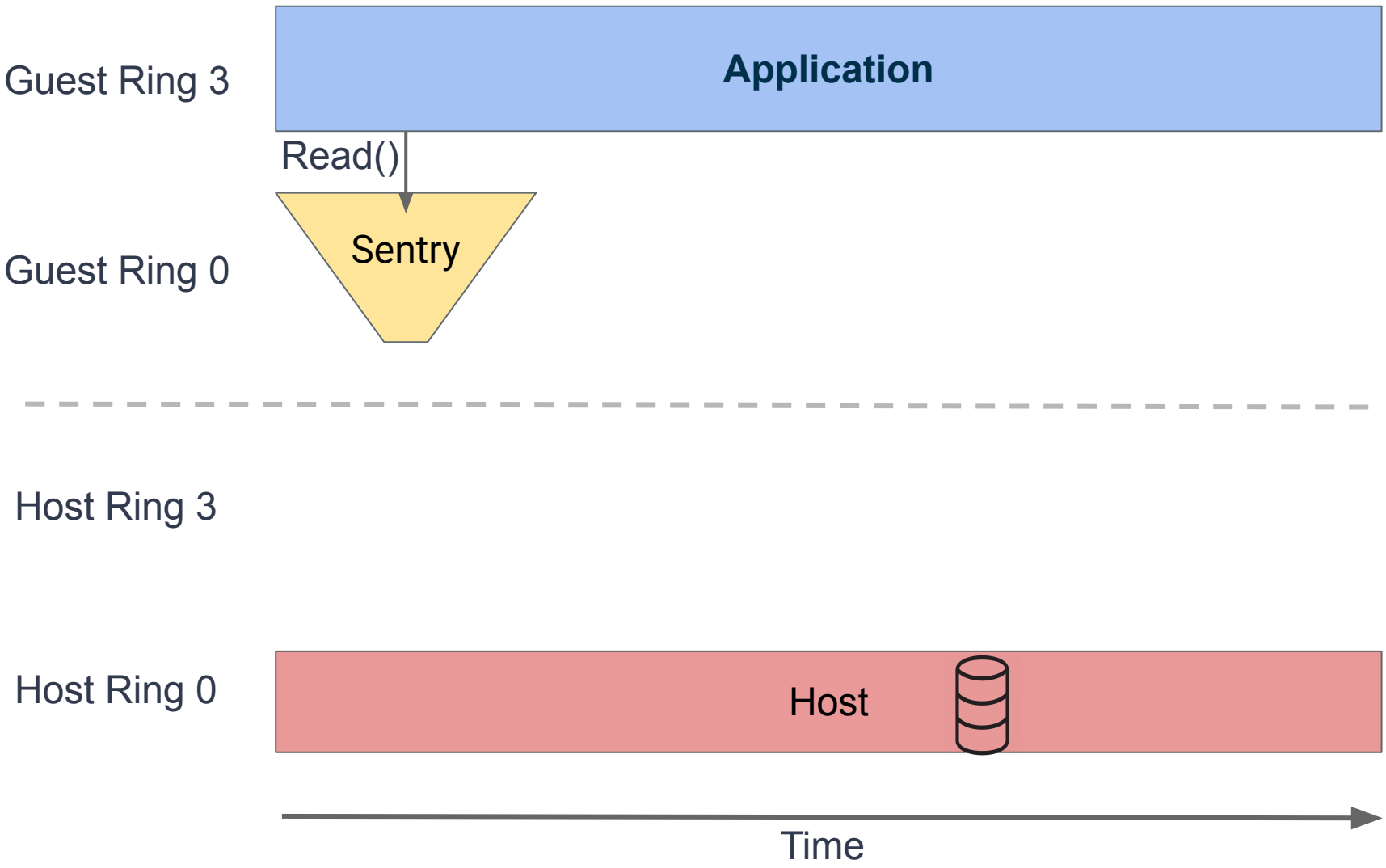
To intercept a system call, gVisor utilizes one of two platforms (ptrace or KVM) to transfer control to sentry



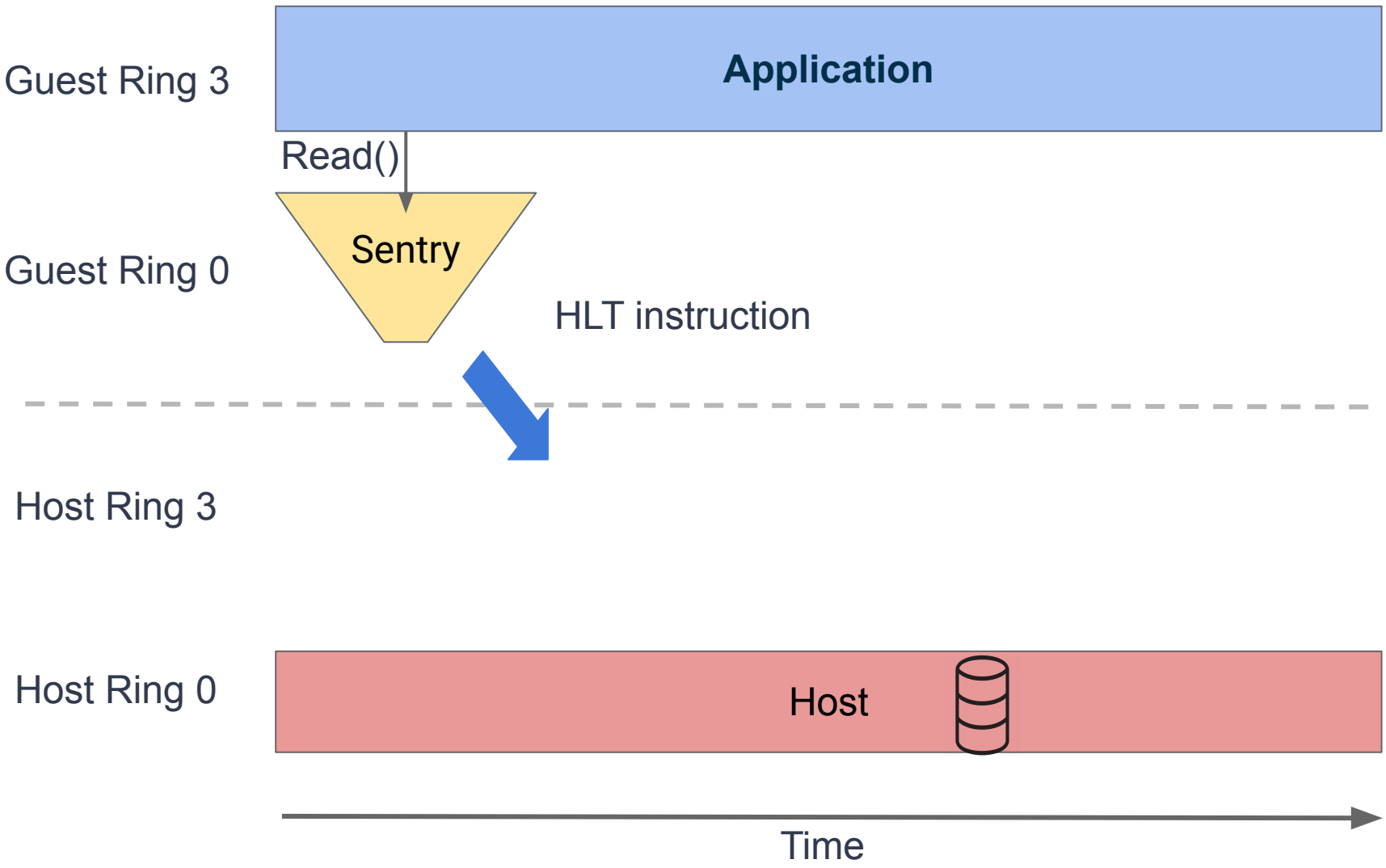
Issuing a system call to the host requires the sentry to switch from non-root ring0 to root ring3 in KVM mode



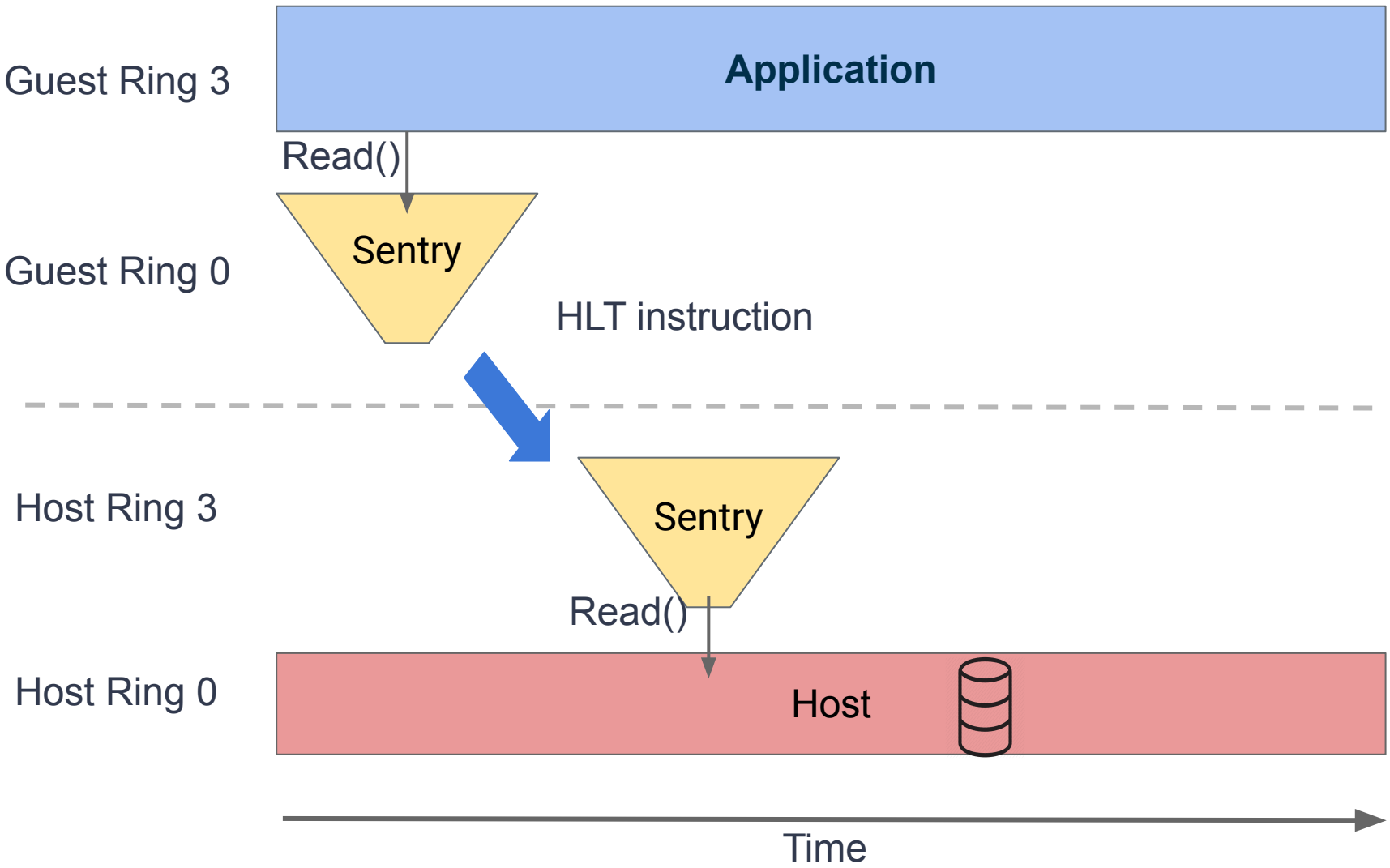
Issuing a system call to the host requires the sentry to switch from non-root ring0 to root ring3 in KVM mode



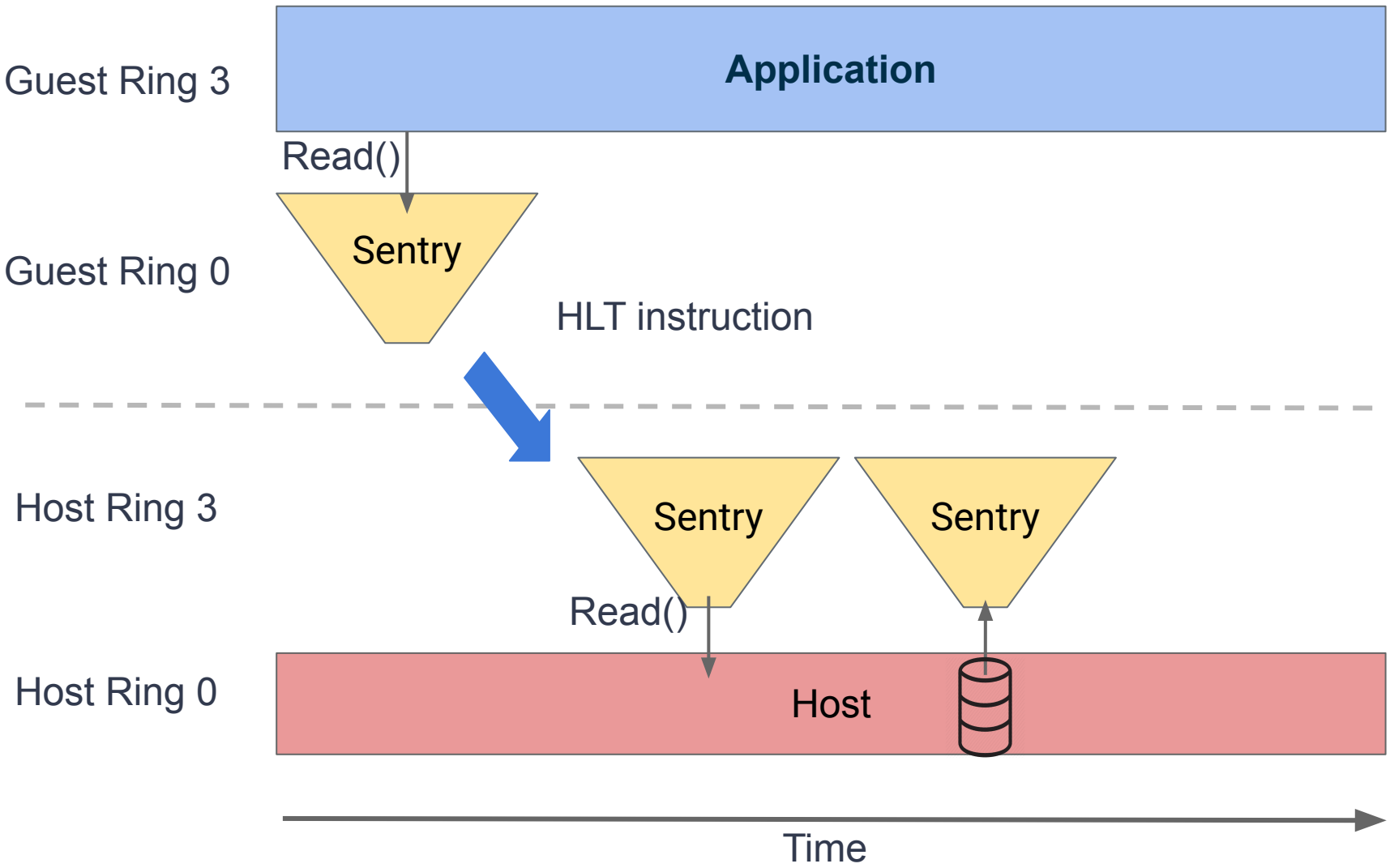
Issuing a system call to the host requires the sentry to switch from non-root ring0 to root ring3 in KVM mode



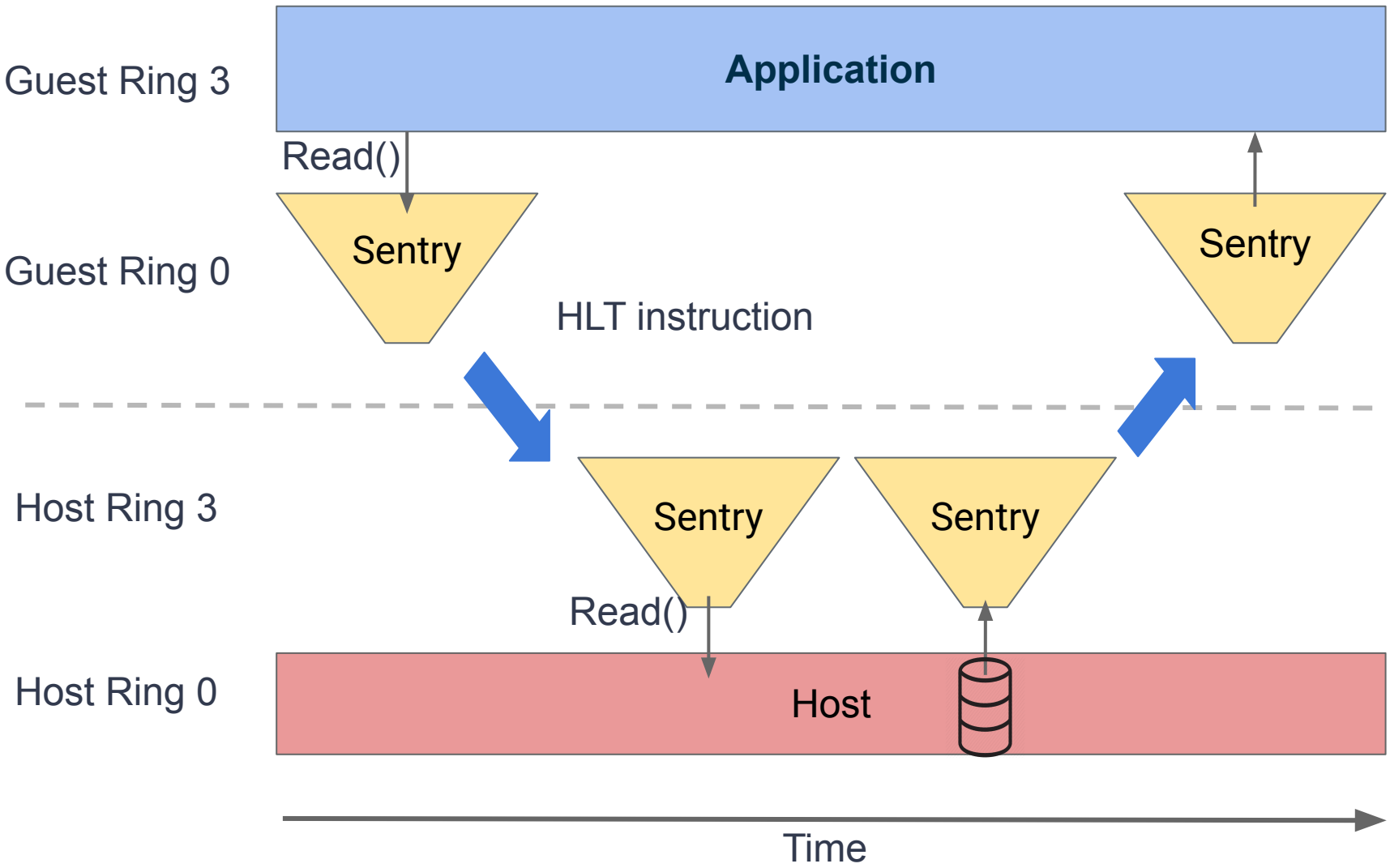
Issuing a system call to the host requires the sentry to switch from non-root ring0 to root ring3 in KVM mode



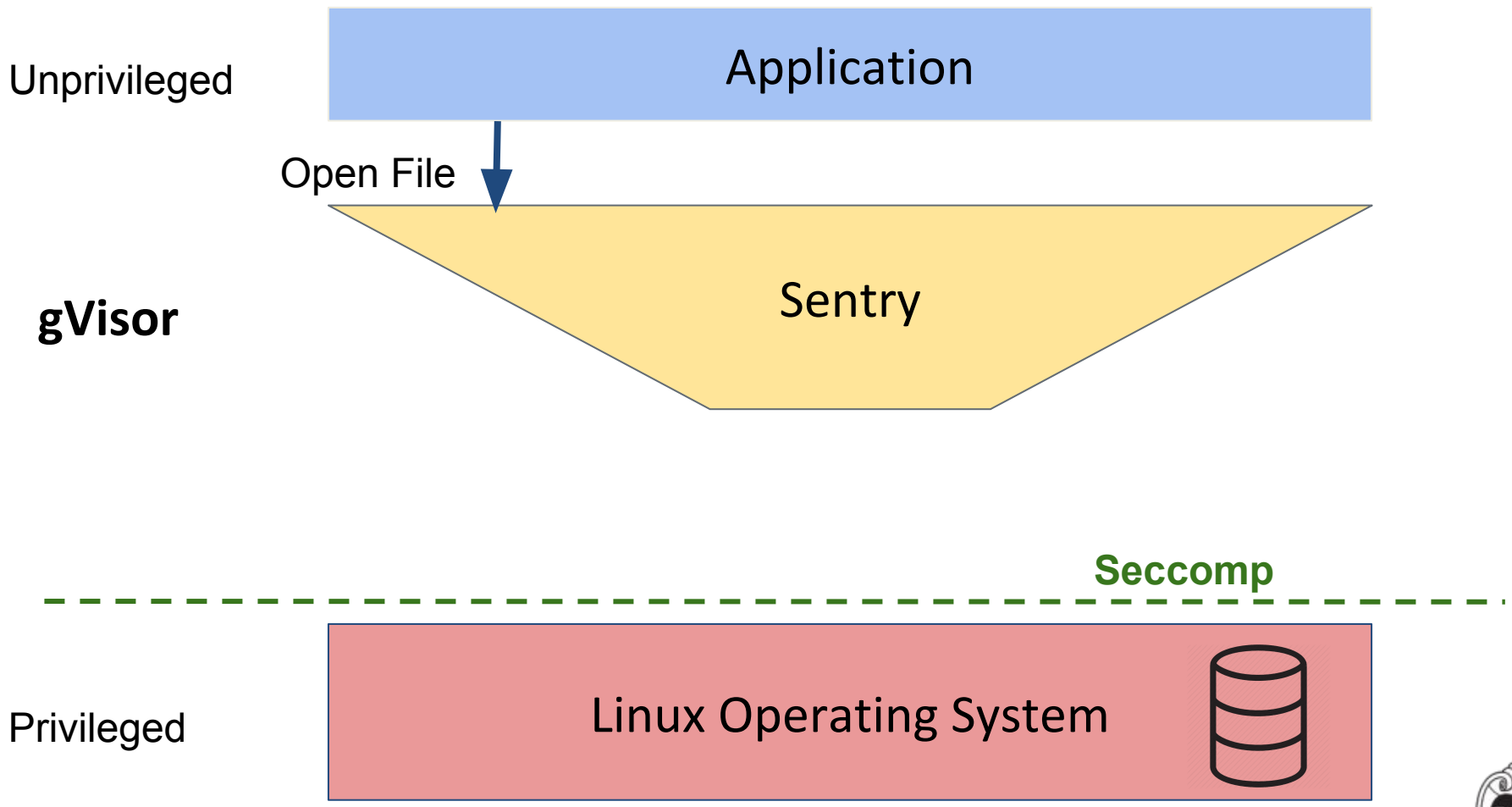
Issuing a system call to the host requires the sentry to switch from non-root ring0 to root ring3 in KVM mode



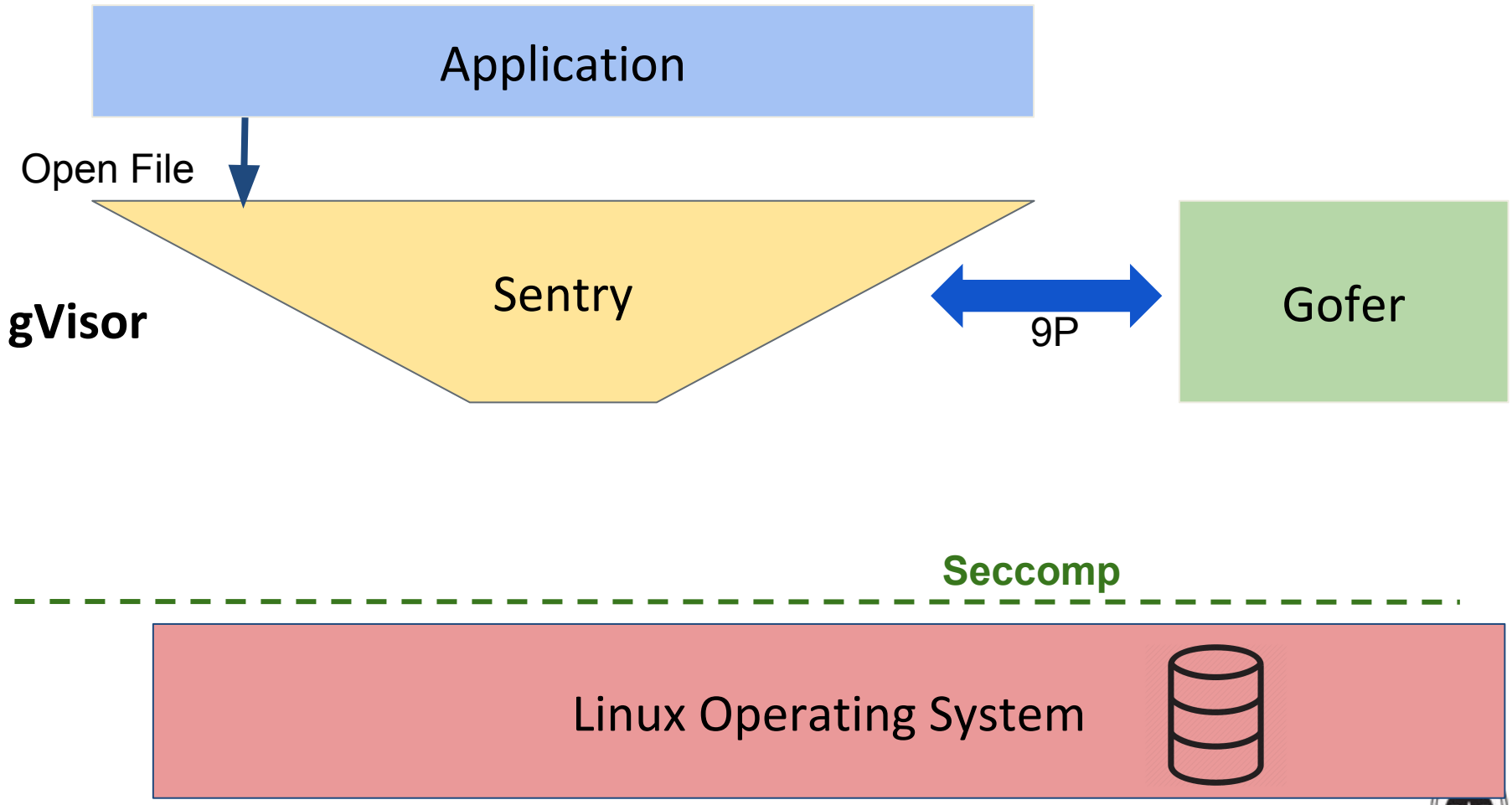
Issuing a system call to the host requires the sentry to switch from non-root ring0 to root ring3 in KVM mode



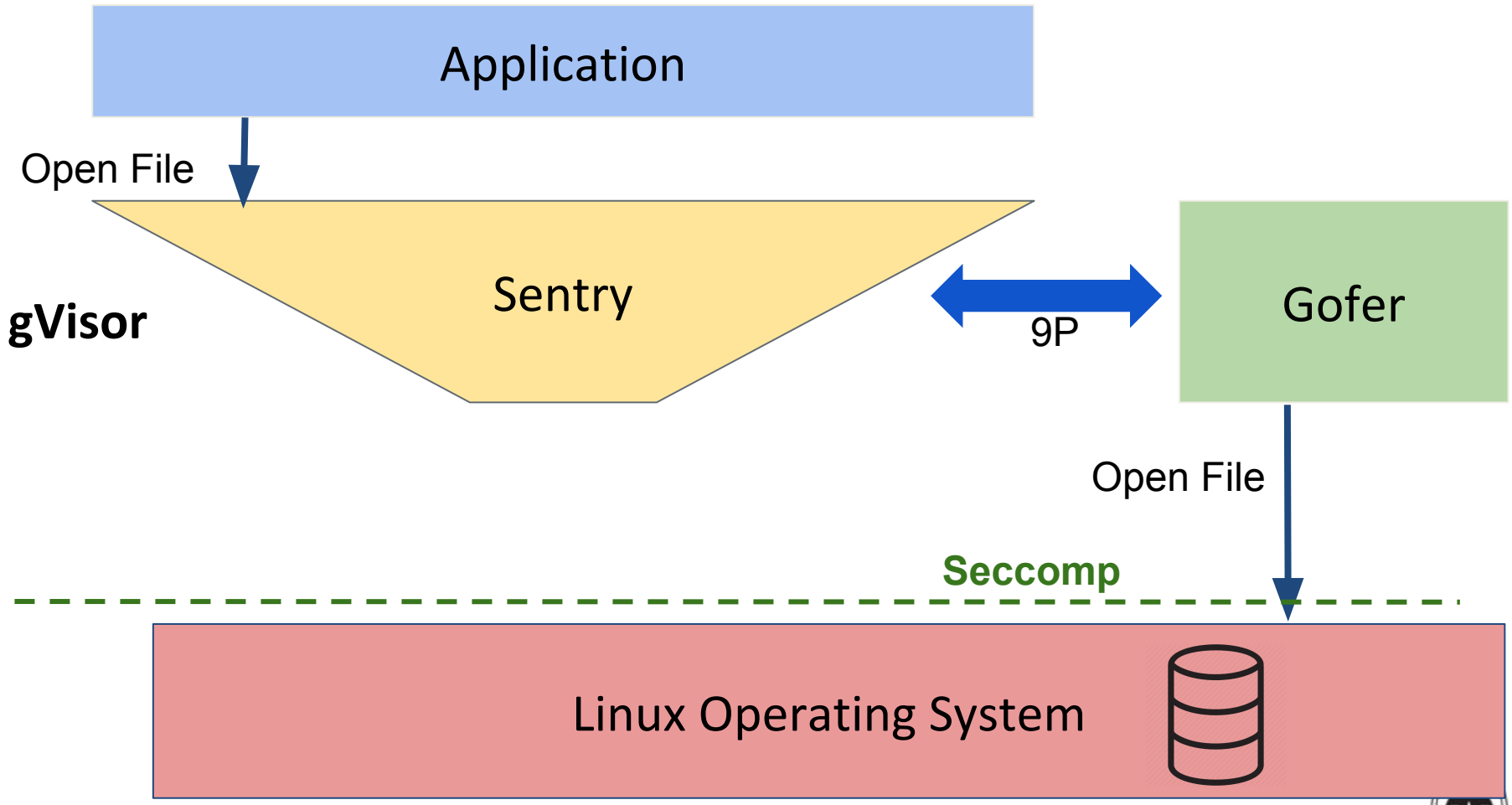
gVisor further protects the host by servicing open system calls with a separate process (the Gofer)



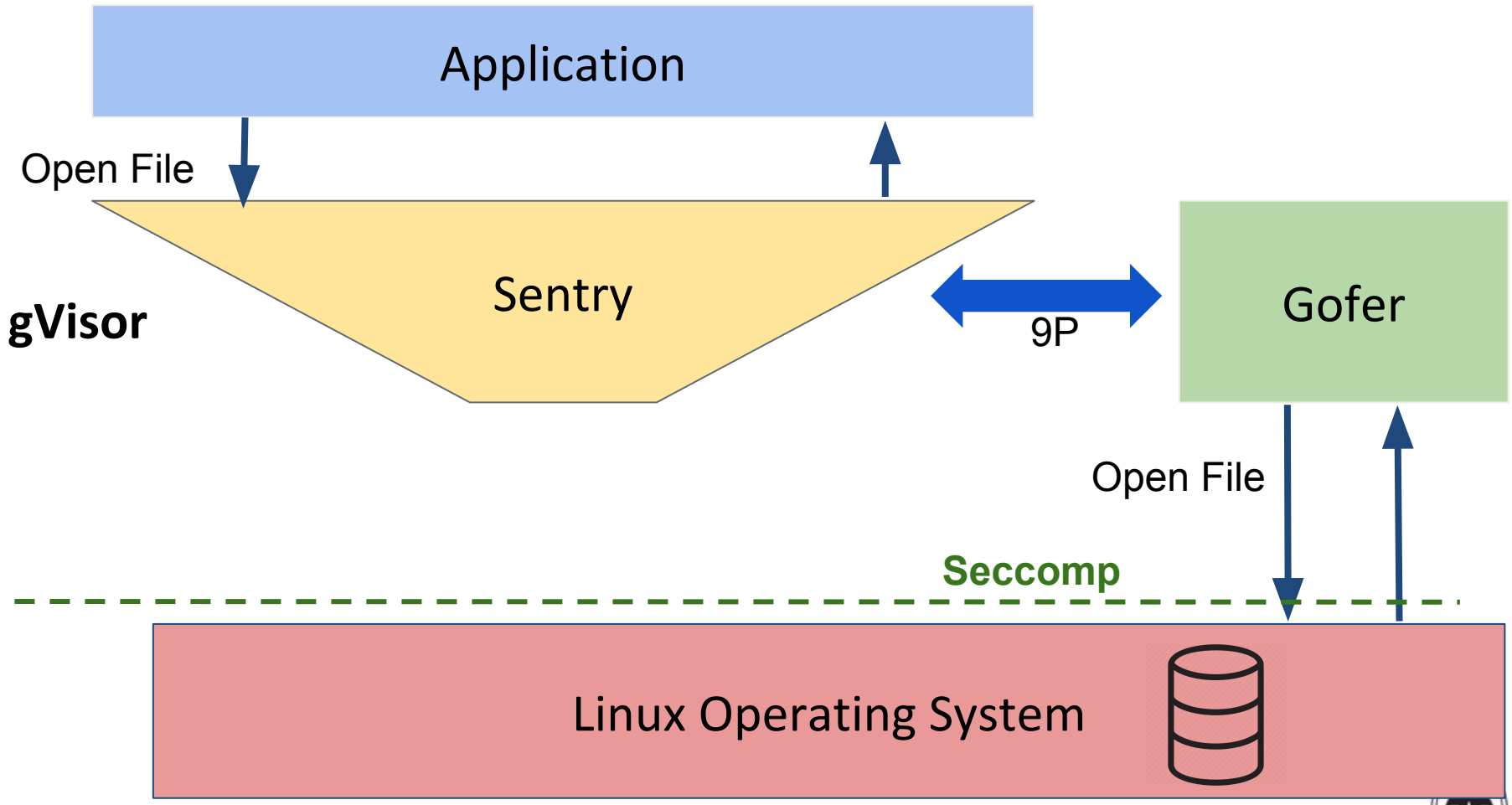
gVisor further protects the host by servicing open system calls with a separate process (the Gofer)



gVisor further protects the host by servicing open system calls with a separate process (the Gofer)



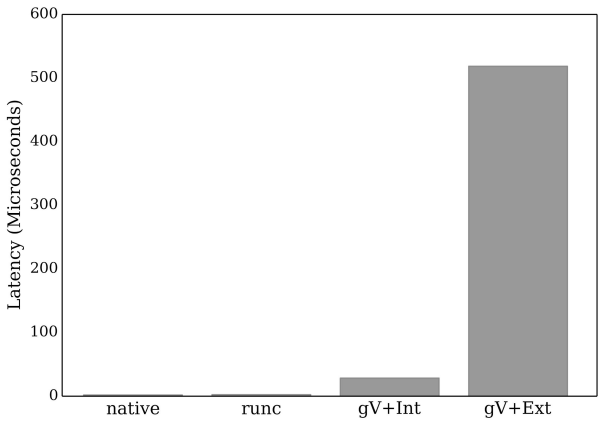
gVisor further protects the host by servicing open system calls with a separate process (the Gofer)



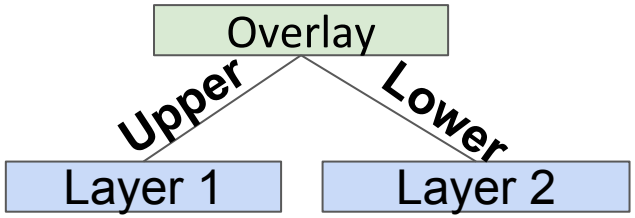
This talk analyzes gVisor, the platform backing Google Cloud Functions, and offers a file system to improve performance

gVisor

gVisor Architecture



gVisor Performance Analysis



ContainerFS Overview

Our experiments used CloudLab machines and were conducted using bare-metal, docker, and gVisor

- CloudLab machine specifications specified in paper

Our experiments used CloudLab machines and were conducted using bare-metal, docker, and gVisor

- CloudLab machine specifications specified in paper
- **Experiments include..**

Our experiments used CloudLab machines and were conducted using bare-metal, docker, and gVisor

- CloudLab machine specifications specified in paper

- **Experiments include..**
  - native -> Bare metal

Our experiments used CloudLab machines and were conducted using bare-metal, docker, and gVisor

- CloudLab machine specifications specified in paper

- **Experiments include..**
  - native -> Bare metal
  - Runc -> Docker’s runtime

Our experiments used CloudLab machines and were conducted using bare-metal, docker, and gVisor

- CloudLab machine specifications specified in paper

- **Experiments include..**
  - native -> Bare metal
  - Runc -> Docker's runtime
  - gV+ptrace -> gVisor's ptrace platform

Our experiments used CloudLab machines and were conducted using bare-metal, docker, and gVisor

- CloudLab machine specifications specified in paper

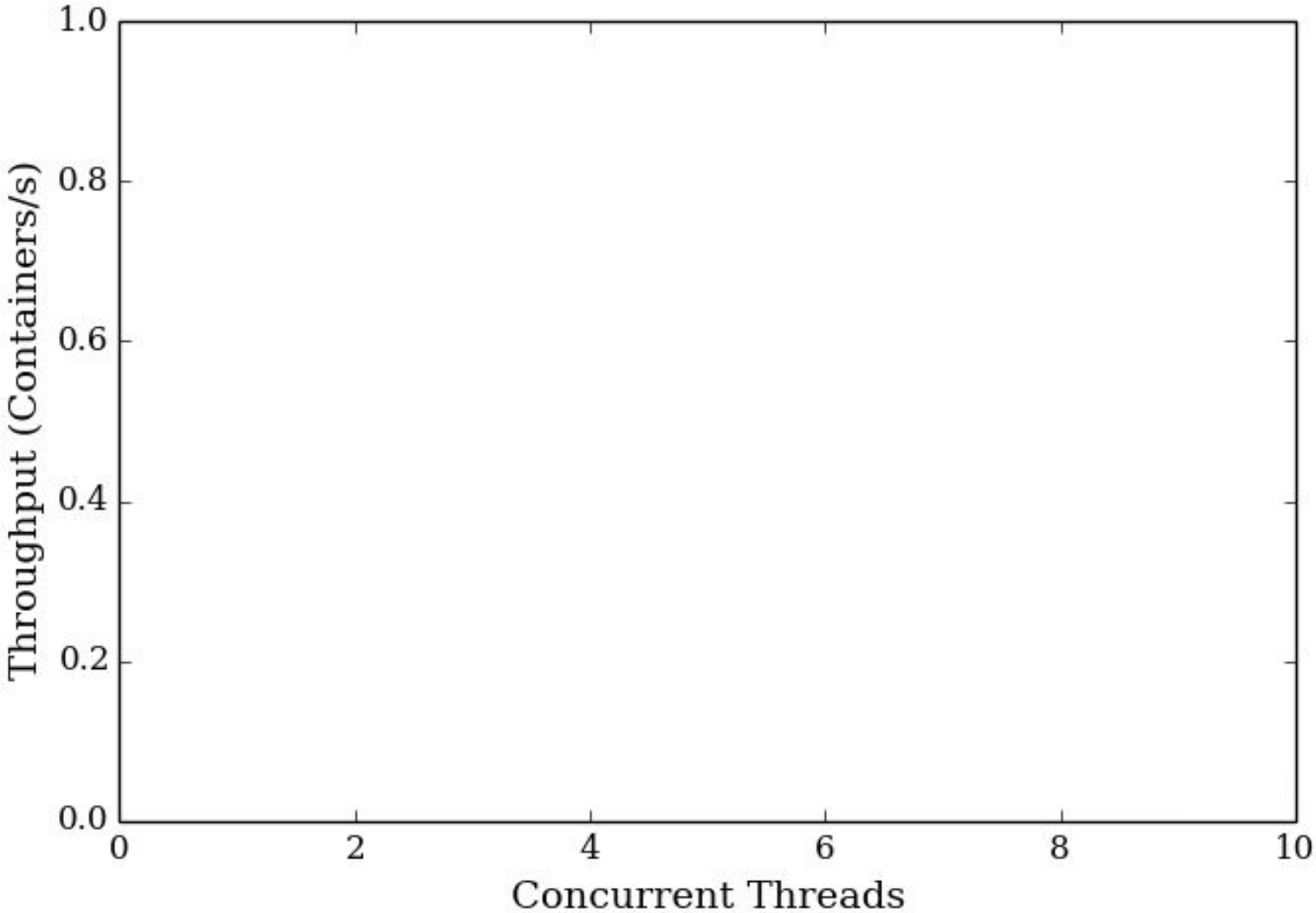
- **Experiments include..**
  - native -> Bare metal
  - Runc -> Docker's runtime
  - gV+ptrace -> gVisor's ptrace platform
  - gV+KVM -> gVisor's KVM platform

Research Questions - Container lifecycle, networking, system calls, memory, storage, Python

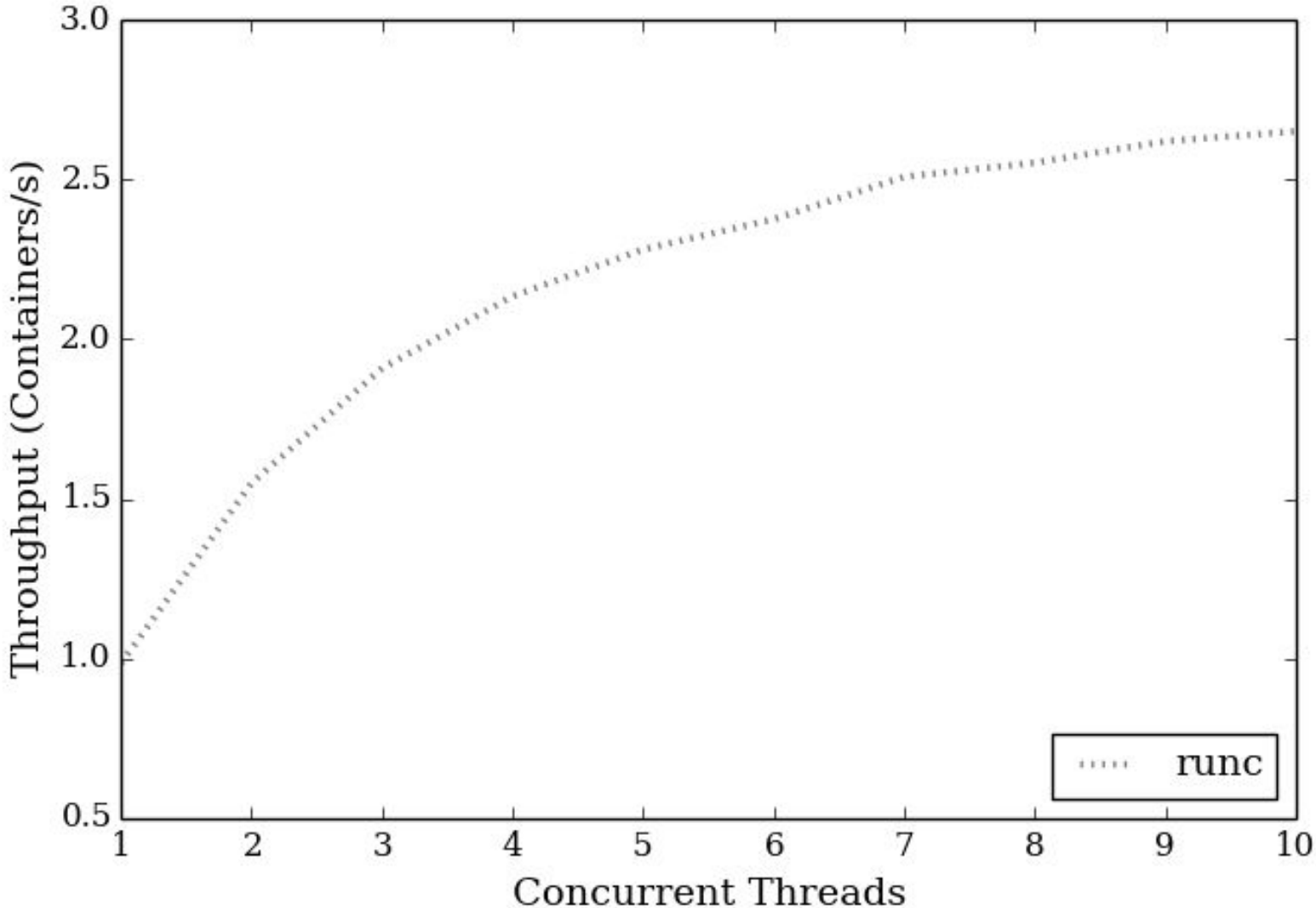
Research Questions - Container lifecycle, networking, system calls, memory, storage, Python

- How quickly can gVisor start and stop containers?

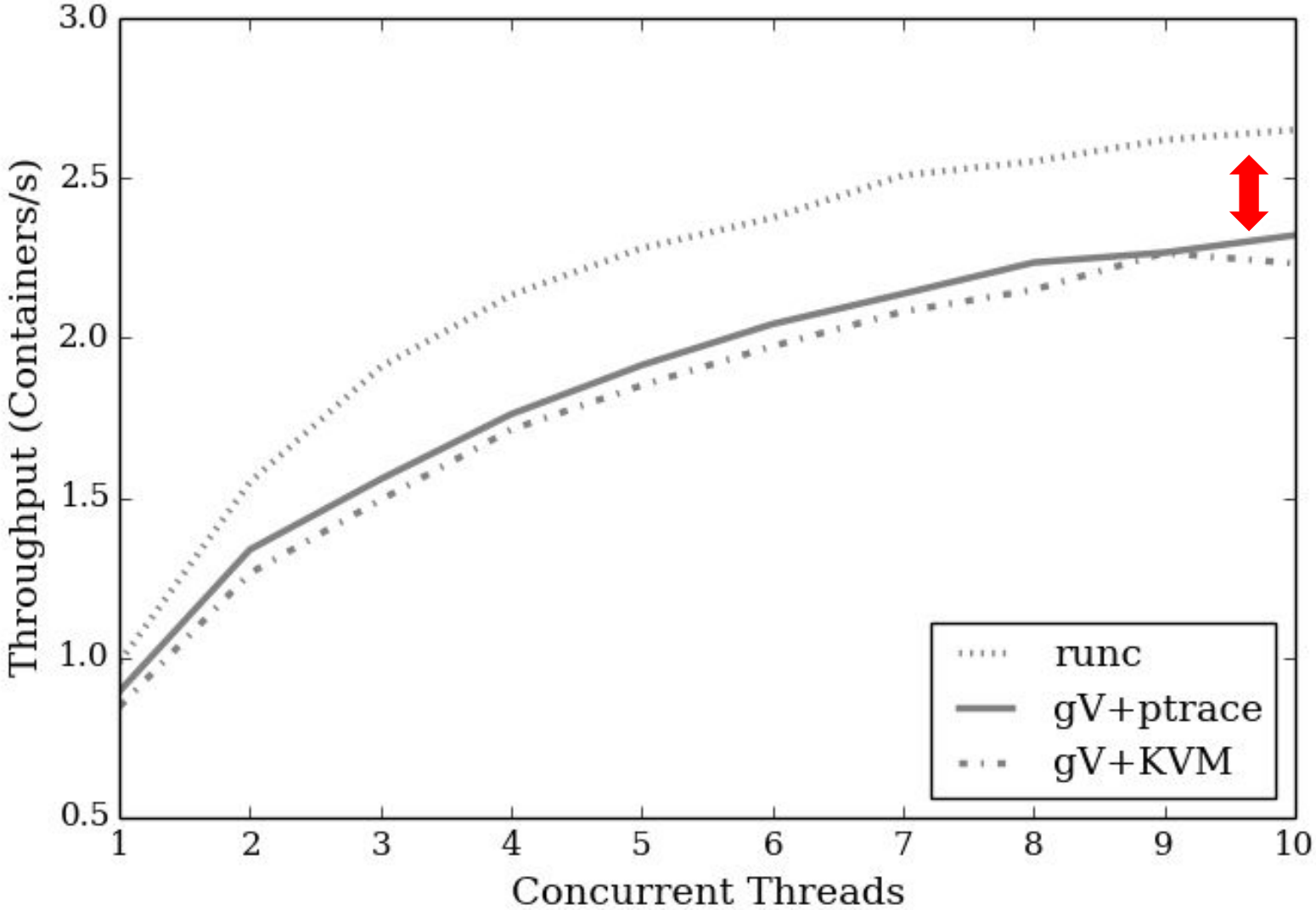
Runc is able to spinup and teardown concurrently executing containers faster than both gVisor platforms



Runc is able to spinup and teardown concurrently executing containers faster than both gVisor platforms



Runc is able to spinup and teardown concurrently executing containers faster than both gVisor platforms



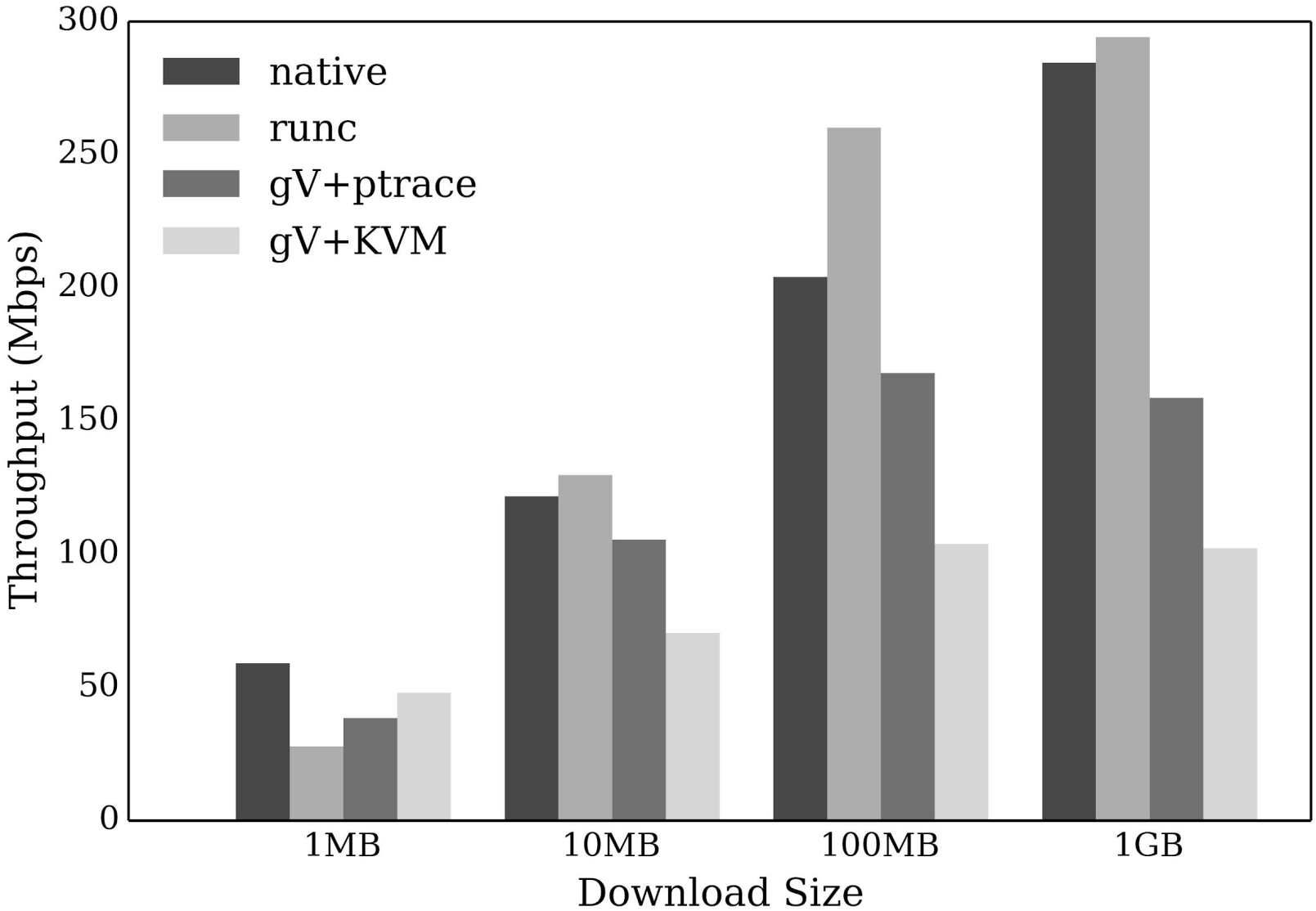
Research Questions - Container lifecycle, networking, system calls, memory, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*

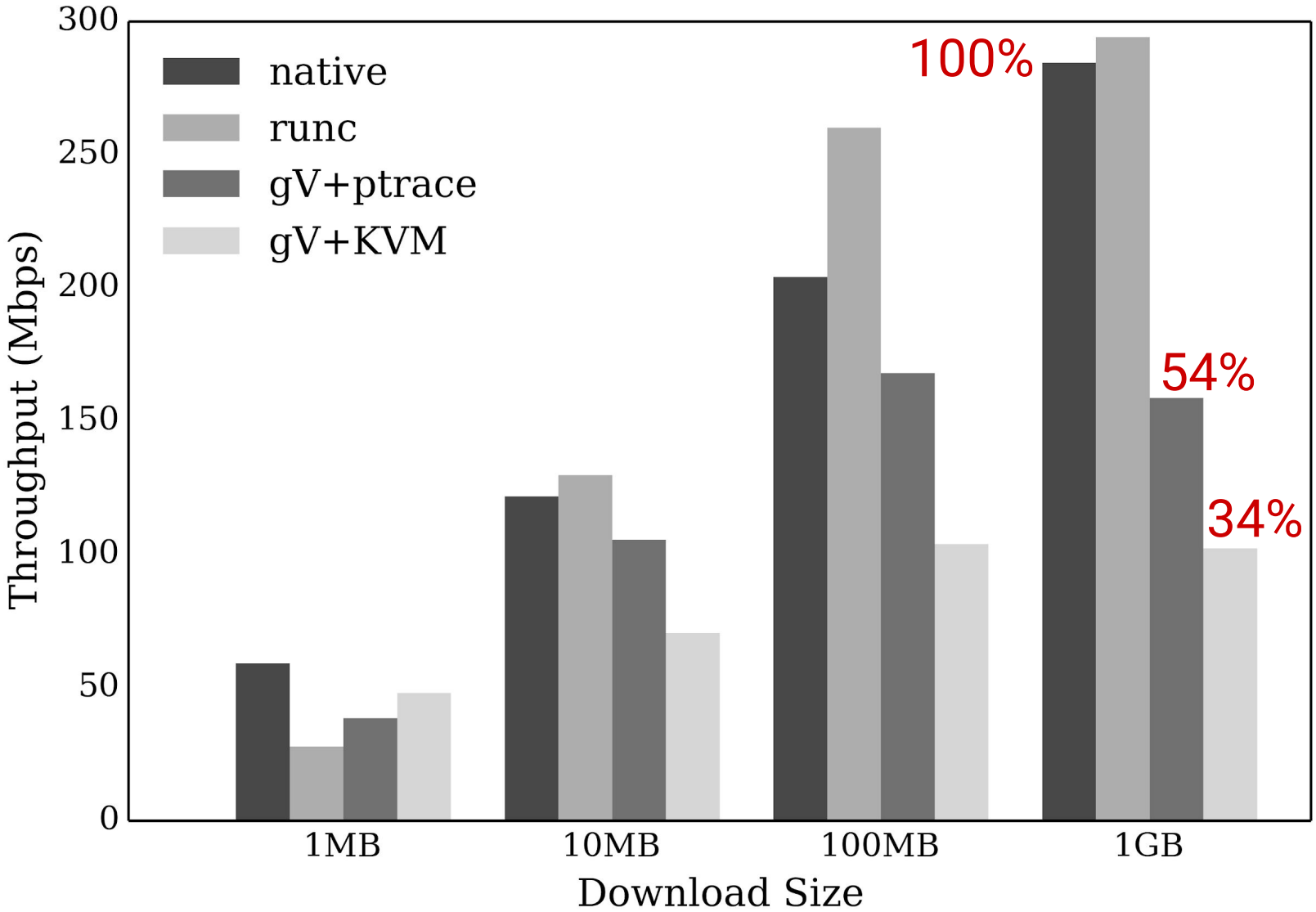
Research Questions - Container lifecycle, **networking**, system calls, memory, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?

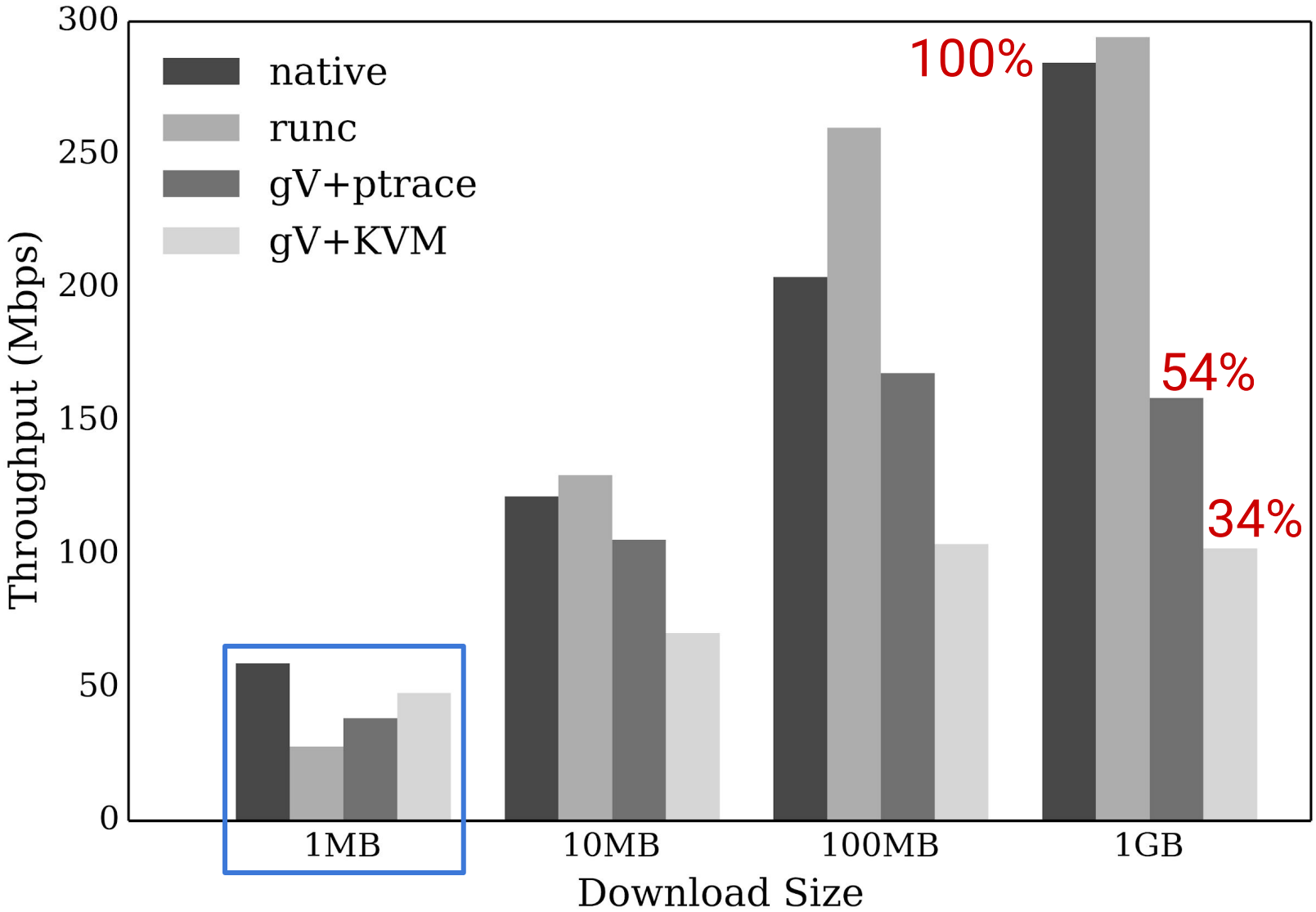
gVisor+KVM achieves 34% of the other system's performance but is competitive at small download sizes



gVisor+KVM achieves 34% of the other system's performance but is competitive at small download sizes



gVisor+KVM achieves 34% of the other system's performance but is competitive at small download sizes



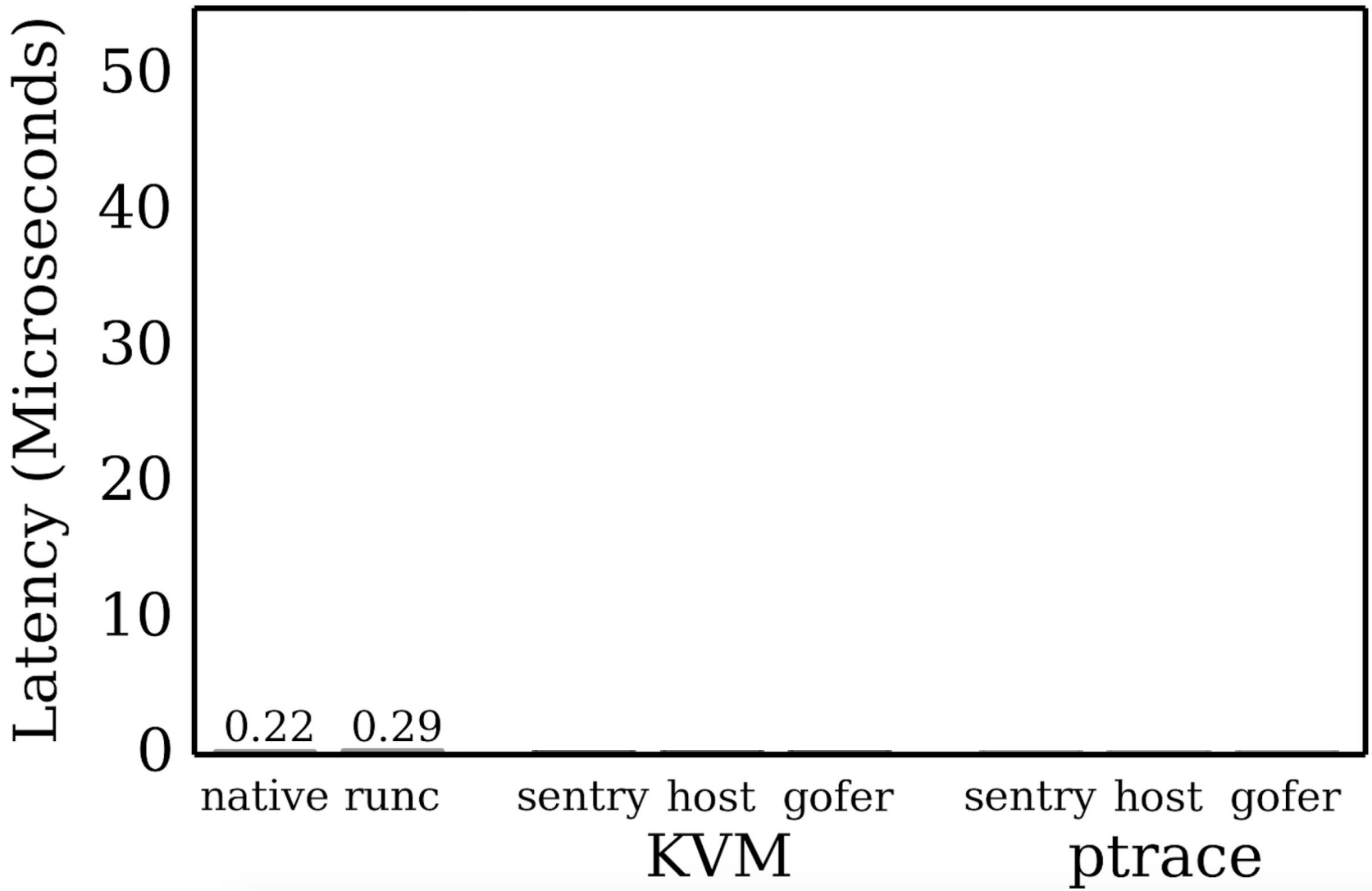
Research Questions - Container lifecycle, **networking**, system calls, memory, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*

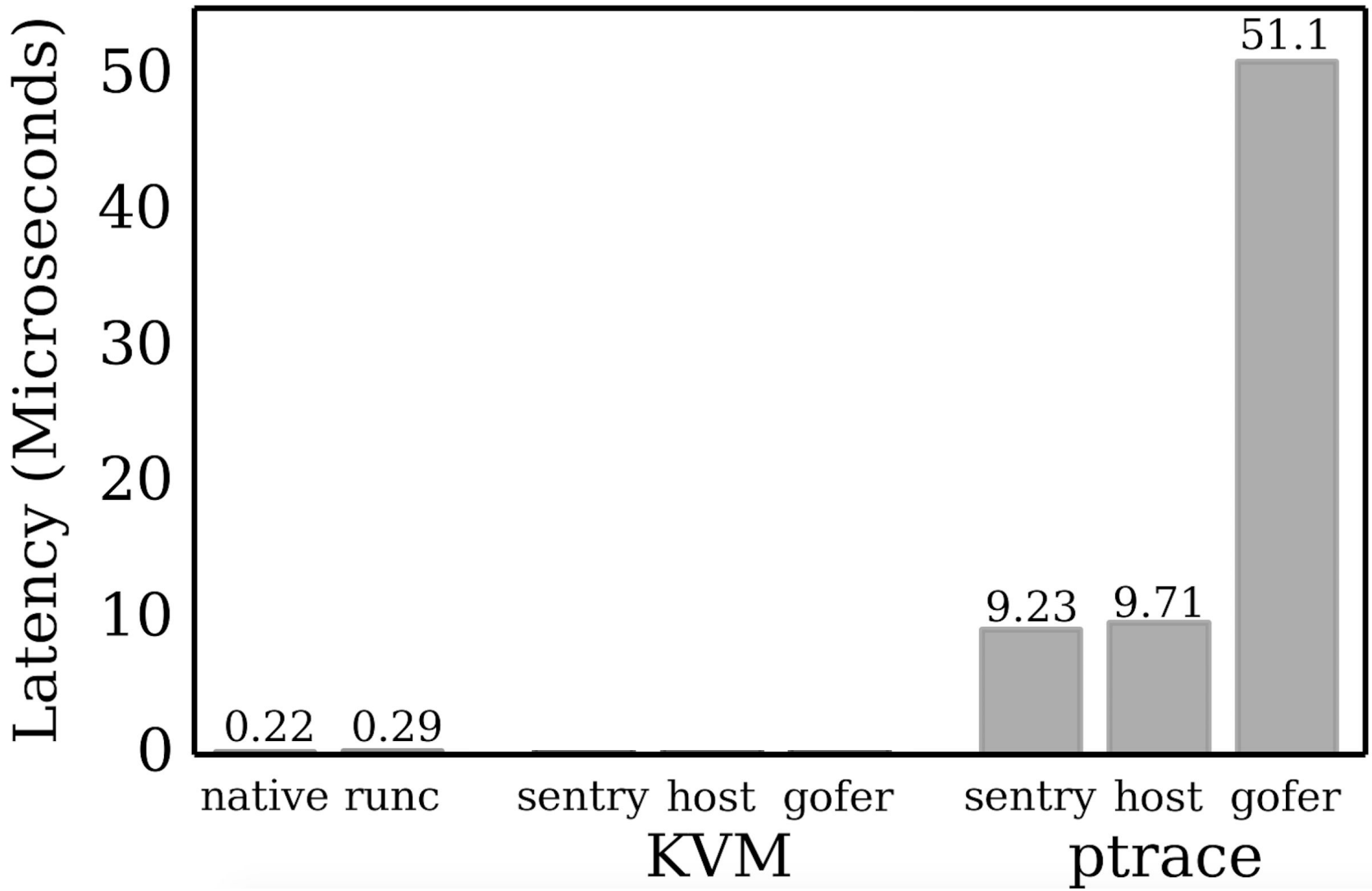
Research Questions - Container lifecycle, networking, **system calls**, memory, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*
- What are the overheads of servicing system calls?

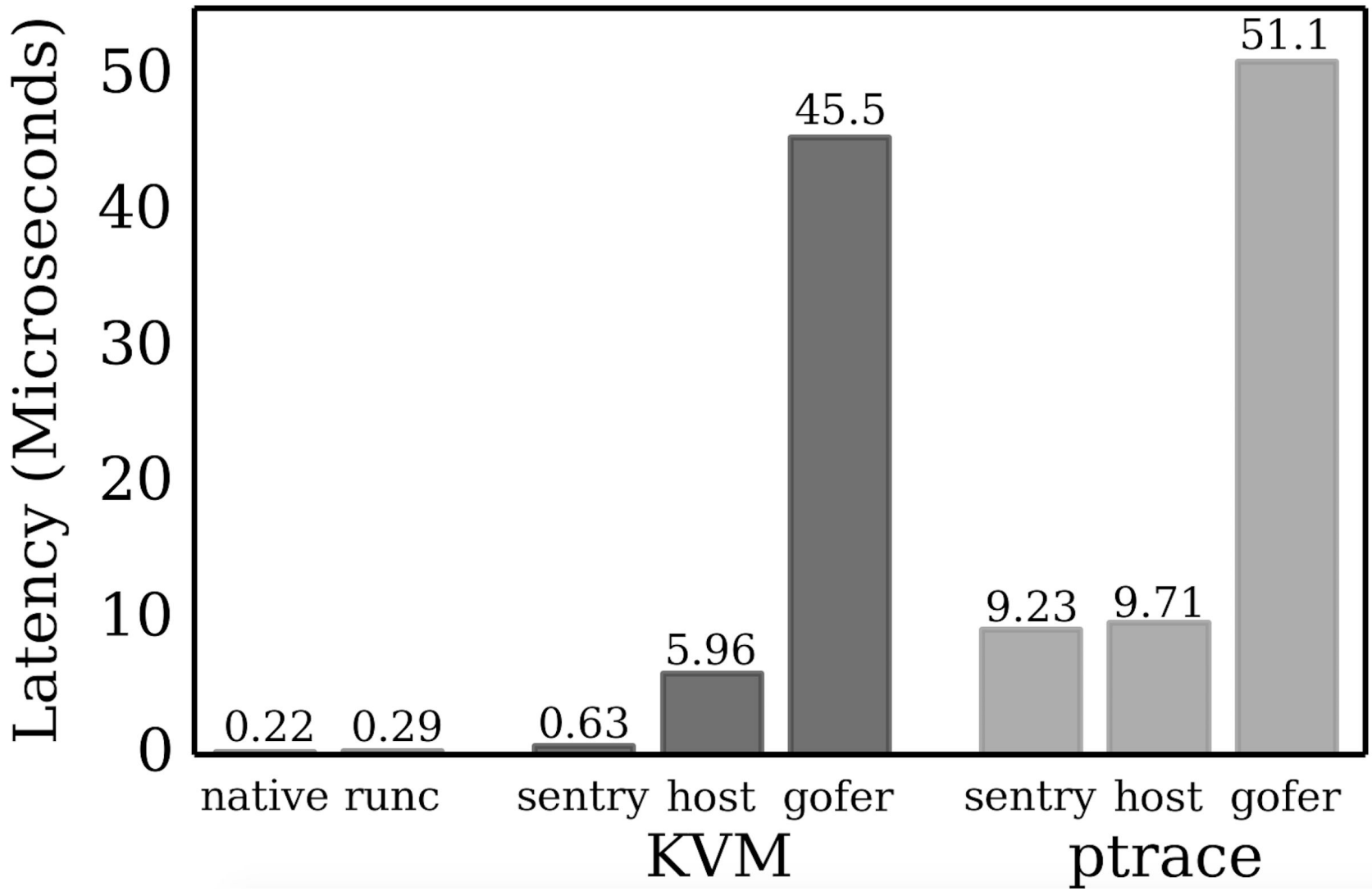
Handling system calls incurs high latencies on both the ptrace and KVM platforms, especially when using the gofer



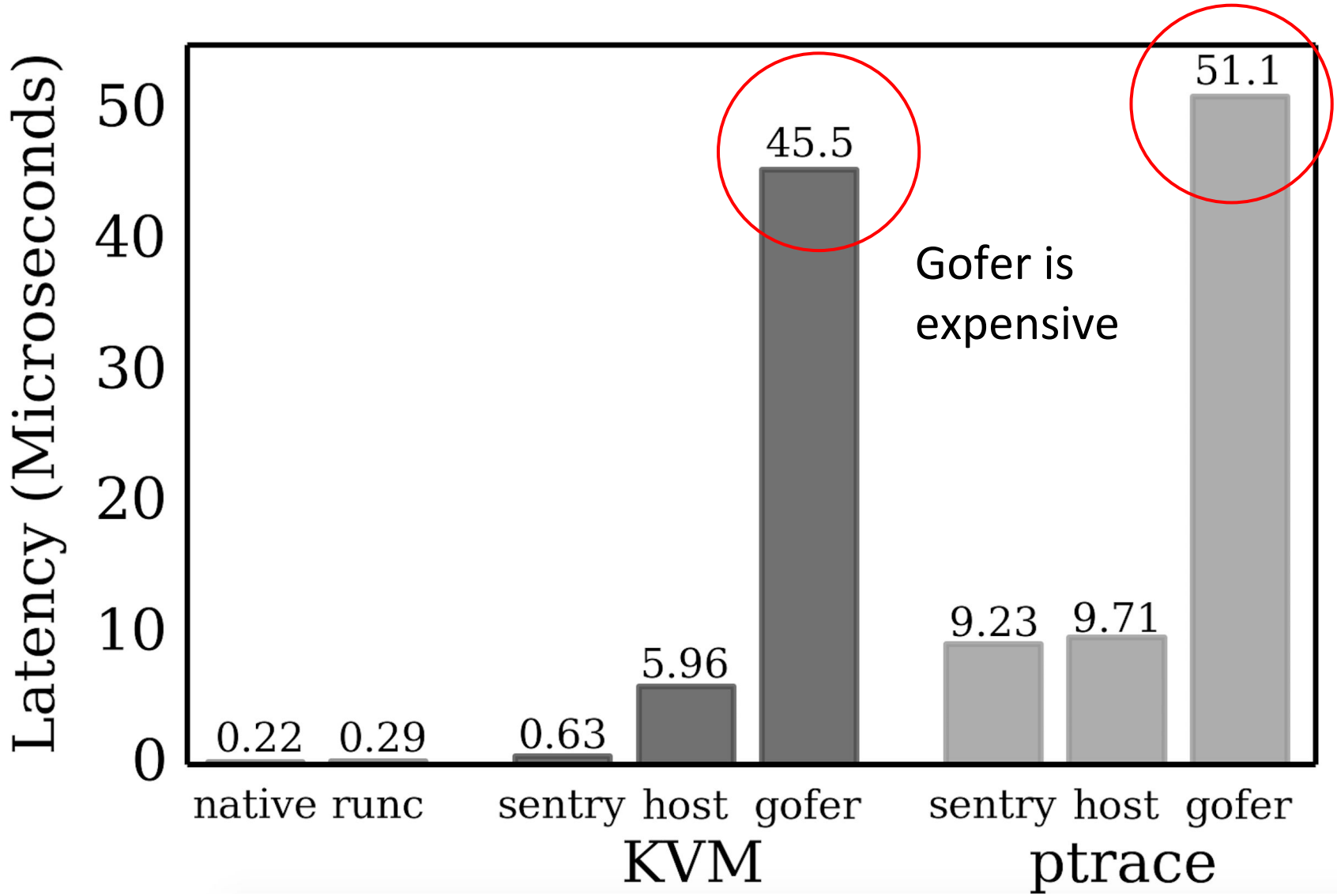
Handling system calls incurs high latencies on both the ptrace and KVM platforms, especially when using the gofer



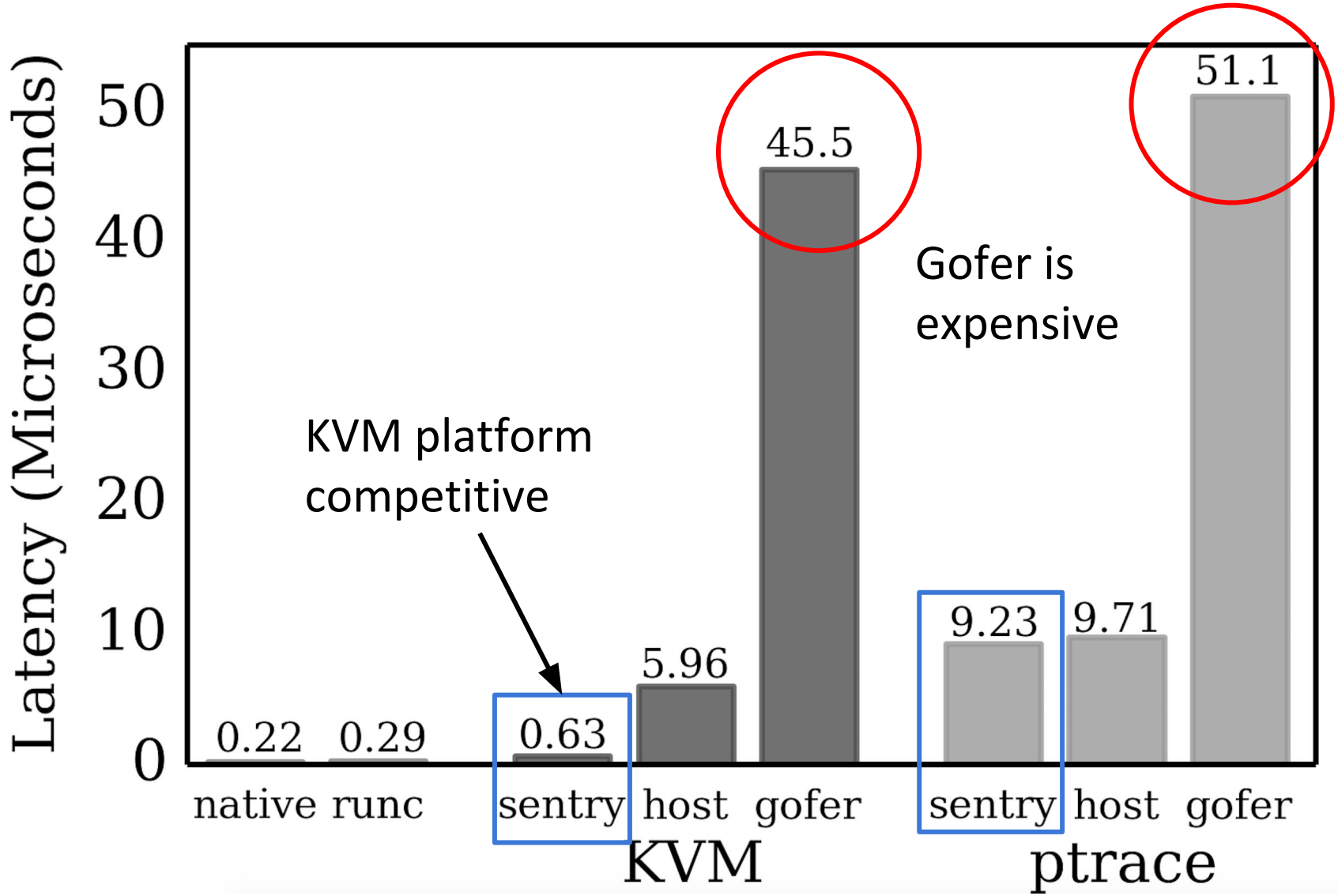
Handling system calls incurs high latencies on both the ptrace and KVM platforms, especially when using the gofer



Handling system calls incurs high latencies on both the ptrace and KVM platforms, especially when using the gofer



Handling system calls incurs high latencies on both the ptrace and KVM platforms, especially when using the gofer



Research Questions - Container lifecycle, networking, **system calls**, memory, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*
- What are the overheads of servicing system calls?
  - *gVisor introduces overheads for system calls*

Research Questions - Container lifecycle, networking, system calls, **memory**, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*
- What are the overheads of servicing system calls?
  - *gVisor introduces overheads for system calls*
- How is memory performance impacted when allocating memory in the sentry?

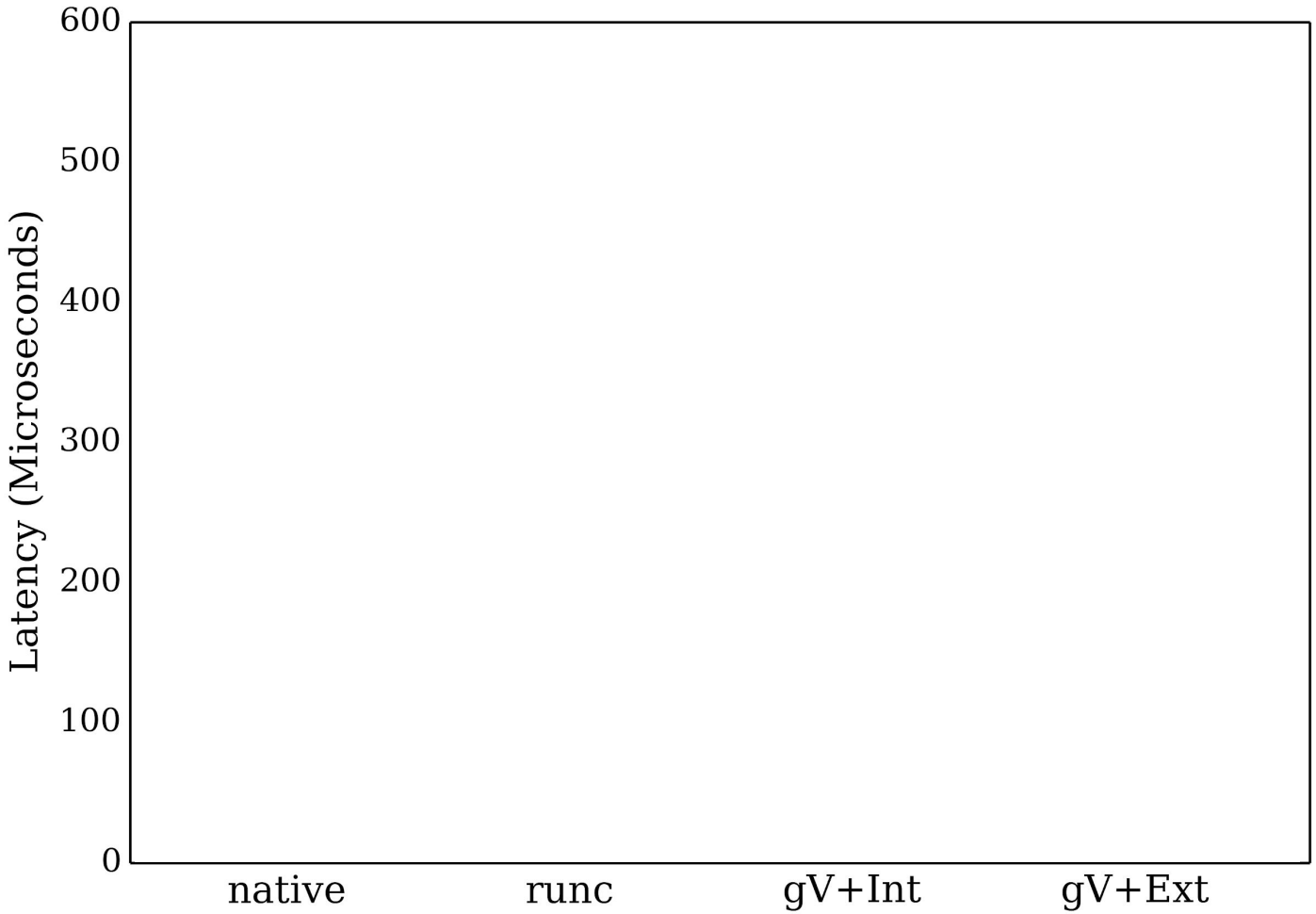
Research Questions - Container lifecycle, networking, system calls, **memory**, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*
- What are the overheads of servicing system calls?
  - *gVisor introduces overheads for system calls*
- How is memory performance impacted when allocating memory in the sentry?
  - *Answered in our paper*

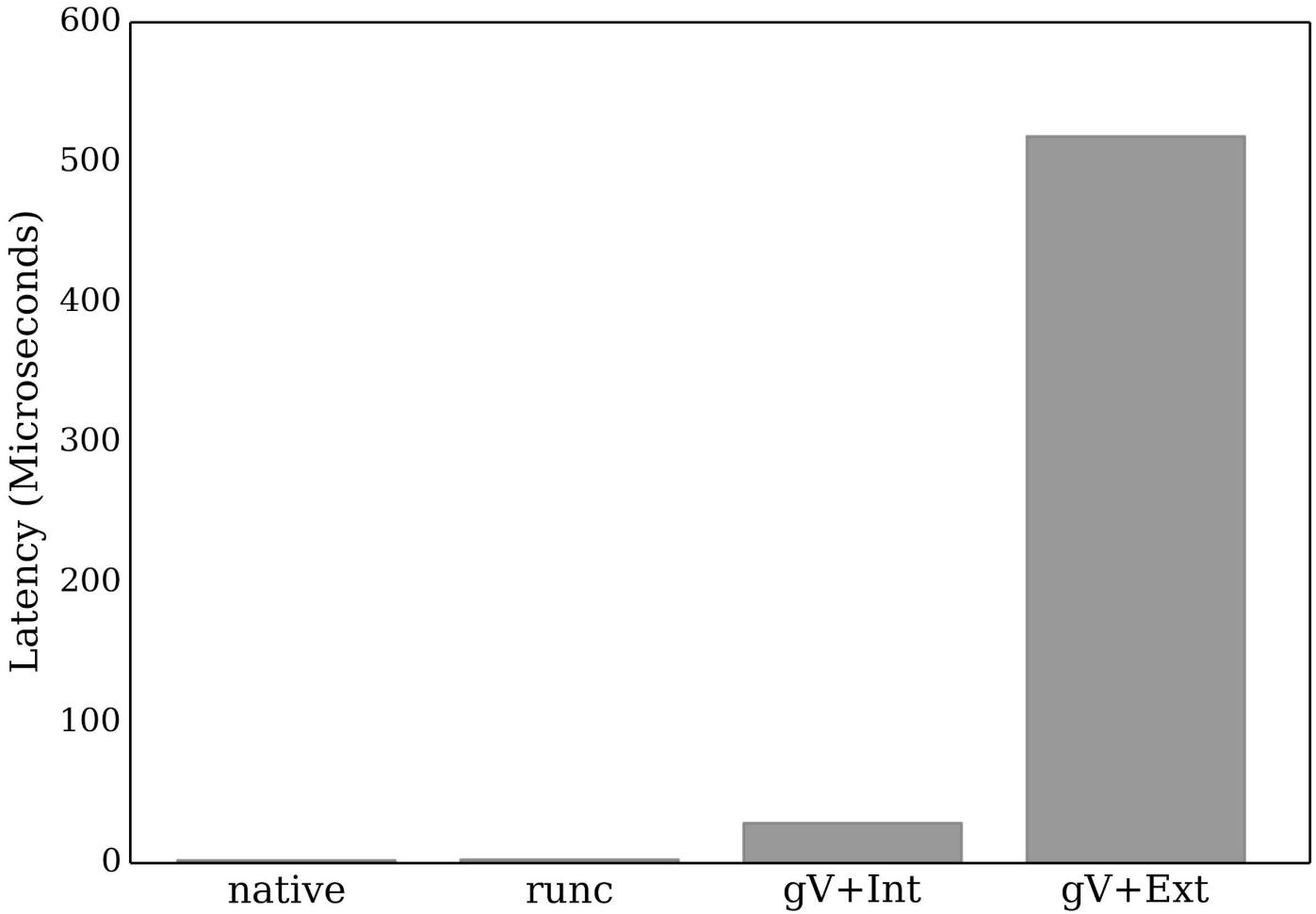
Research Questions - Container lifecycle, networking, system calls, memory, **storage**, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*
- What are the overheads of servicing system calls?
  - *gVisor introduces overheads for system calls*
- How is memory performance impacted when allocating memory in the sentry?
  - *Answered in our paper*
- What is the cost of opening files with the gofer and what are the benefits of embedding a file system in the sentry?

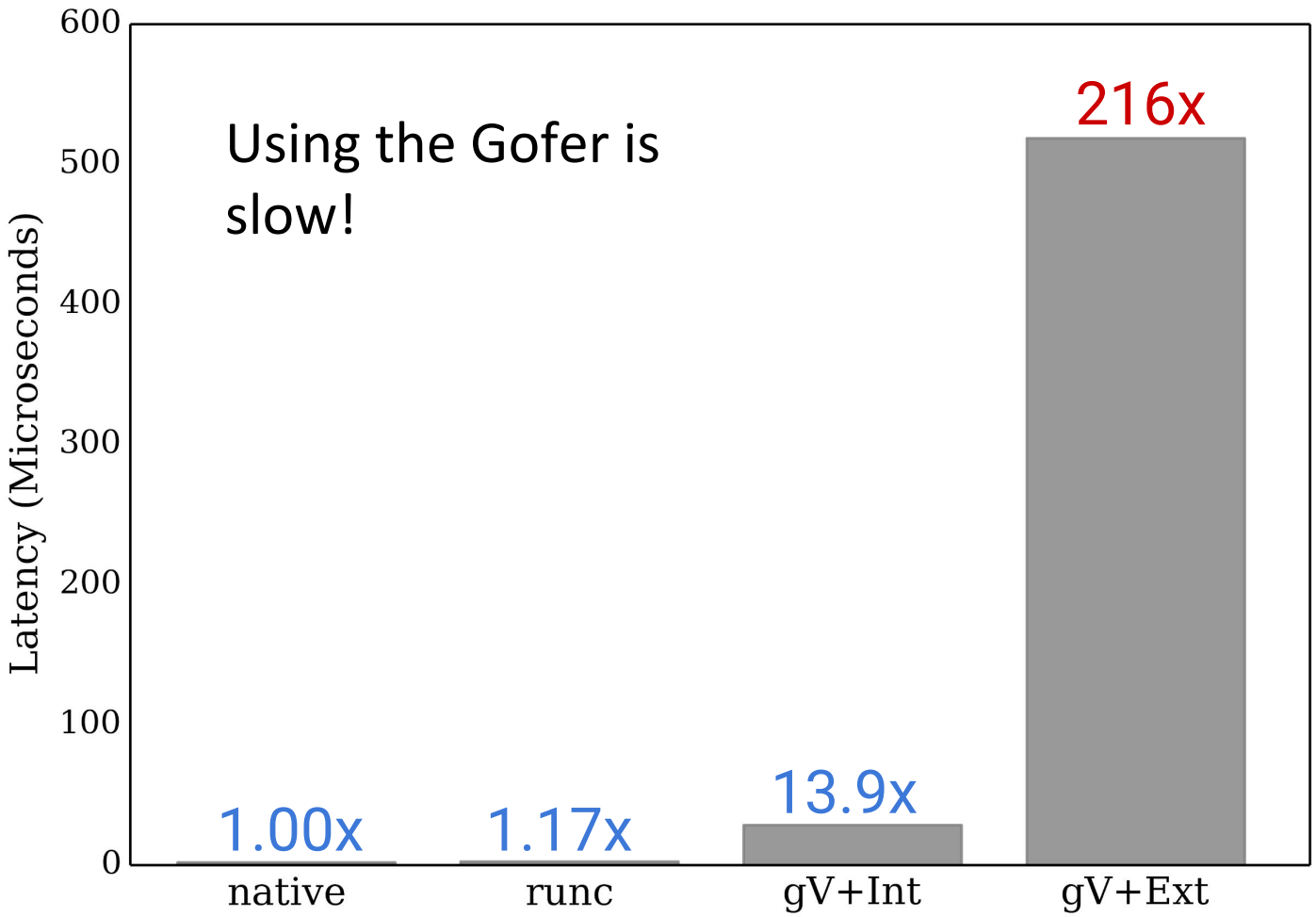
Opening and closing a file in gVisor is substantially slower than current platforms (native, runc)



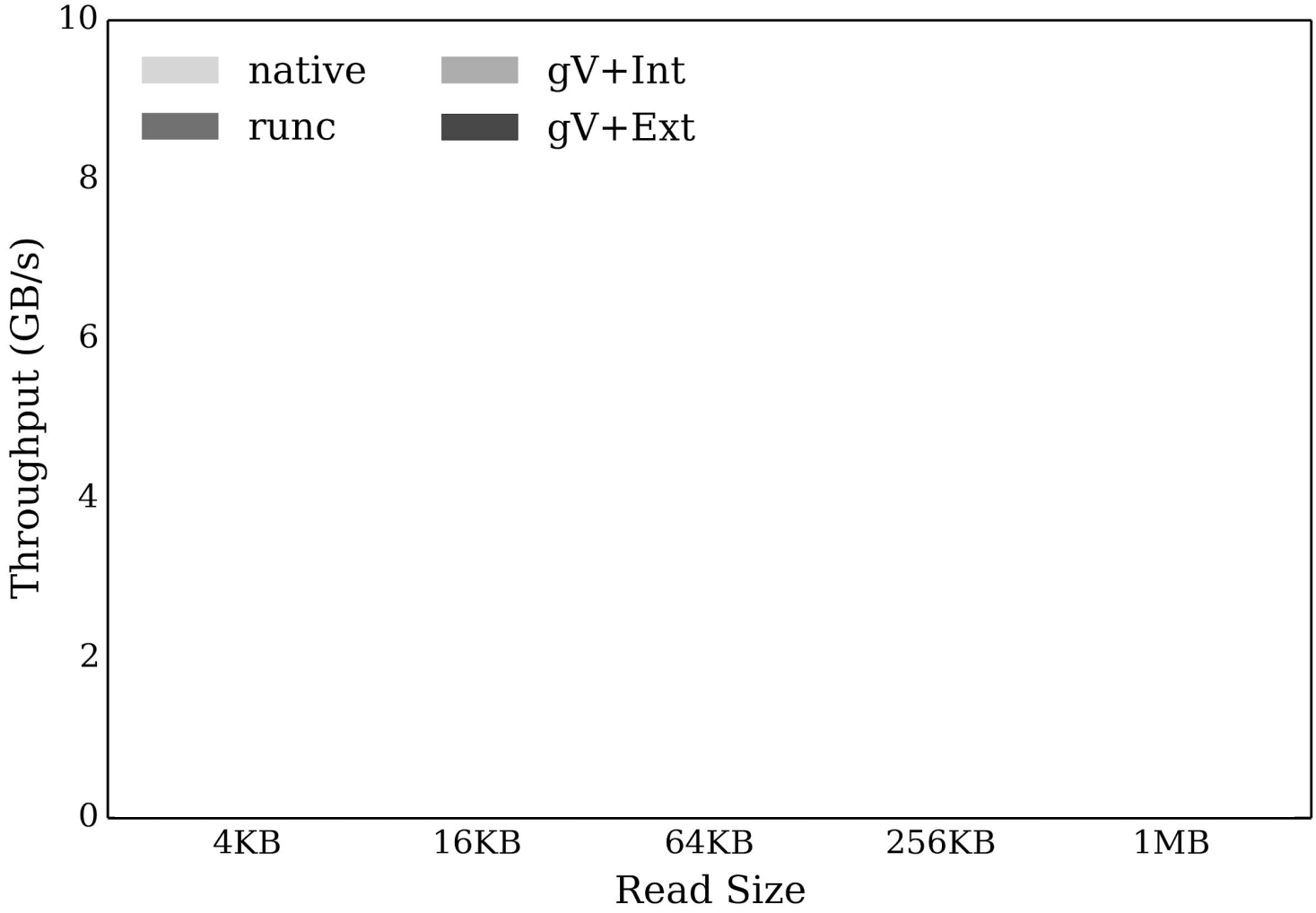
Opening and closing a file in gVisor is substantially slower than current platforms (native, runc)



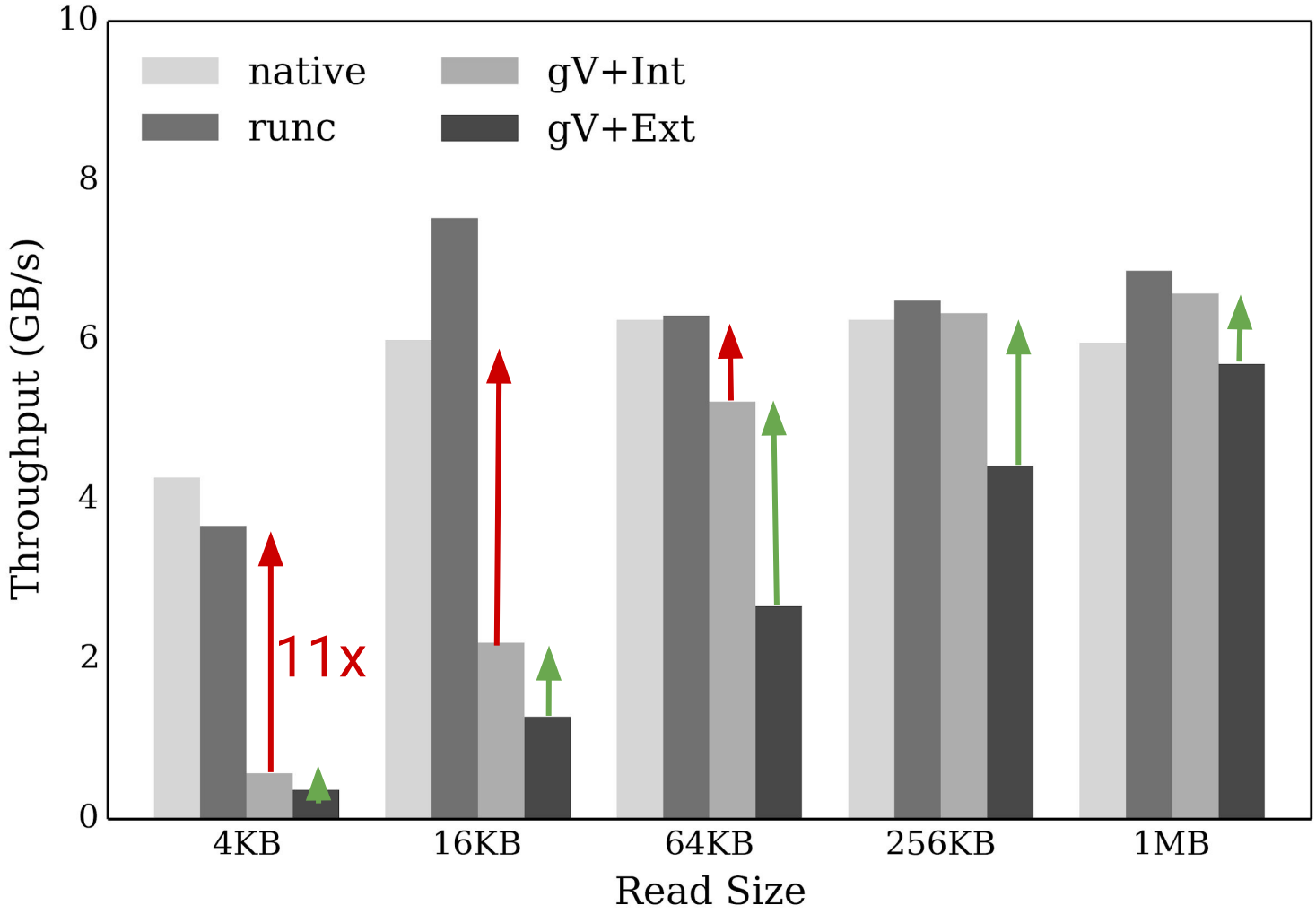
Opening and closing a file in gVisor is substantially slower than current platforms (native, runc)



gVisor performs poorly at small read sizes but begins to become competitive at large read sizes



gVisor performs poorly at small read sizes but begins to become competitive at large read sizes



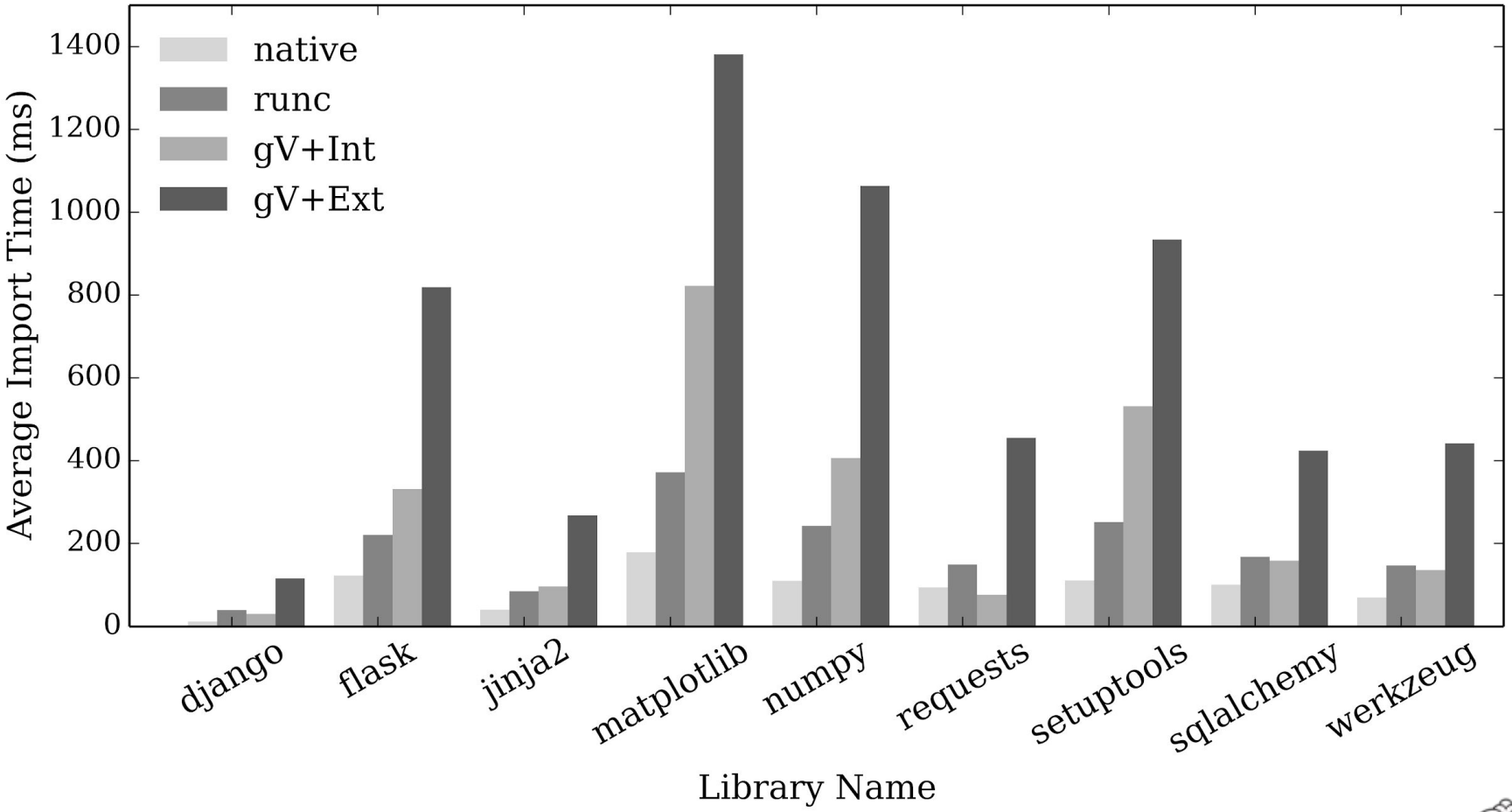
Research Questions - Container lifecycle, networking, system calls, memory, **storage**, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*
- What are the overheads of servicing system calls?
  - *gVisor introduces overheads for system calls*
- How is memory performance impacted when allocating memory in the sentry?
  - *Answered in our paper*
- What is the cost of opening files with the gofer and what are the benefits of embedding a file system in the sentry?
  - *Opening and closing files is 216x slower through the gofer*
  - *Reading small files is 11x slower, but scales well with size*

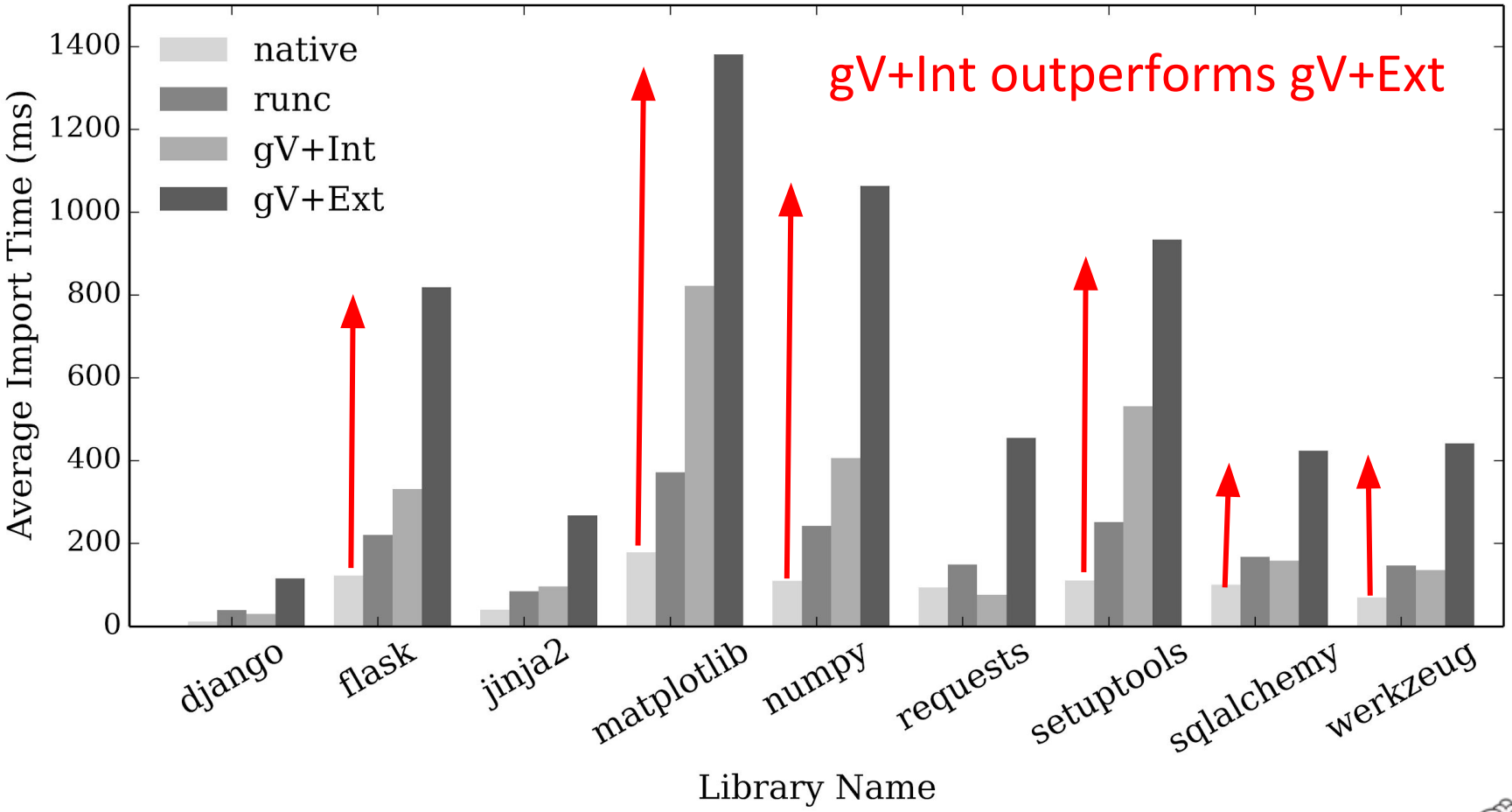
Research Questions - Container lifecycle, networking, system calls, memory, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*
- What are the overheads of servicing system calls?
  - *gVisor introduces overheads for system calls*
- How is memory performance impacted when allocating memory in the sentry?
  - *Answered in our paper*
- What is the cost of opening files with the gofer and what are the benefits of embedding a file system in the sentry?
  - *Opening and closing files is 216x slower through the gofer*
  - *Reading small files is 11x slower, but scales well with size*
- How are Python imports affected by gVisor?

Importing Python modules from an external tmpfs with gVisor is 2-4x slower than runc and native



Importing Python modules from an external tmpfs with gVisor is 2-4x slower than runc and native

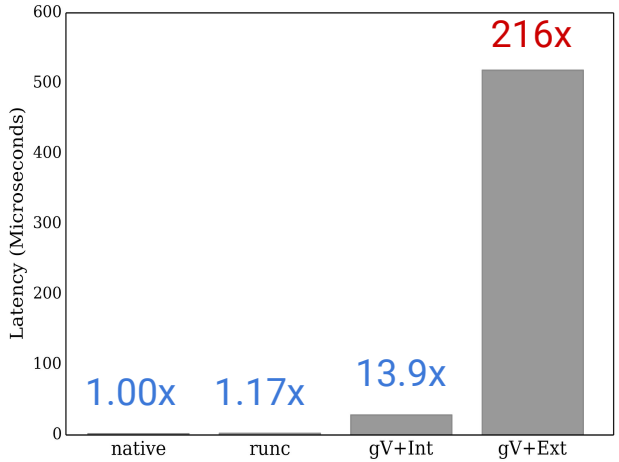


Research Questions - Container lifecycle, networking, system calls, memory, storage, Python

- How quickly can gVisor start and stop containers?
  - *gVisor scales poorly with increased concurrency*
- How does gVisor's user-space, Go-based network stack perform?
  - *Handles small downloads well, but scales poorly*
- What are the overheads of servicing system calls?
  - *gVisor introduces overheads for system calls*
- How is memory performance impacted when allocating memory in the sentry?
  - *Answered in our paper*
- What is the cost of opening files with the gofer and what are the benefits of embedding a file system in the sentry?
  - *Opening and closing files is 216x slower through the gofer*
  - *Reading small files is 11x slower, but scales well with size*
- How are Python imports affected by gVisor?
  - *Importing Python modules using gVisor is slow*

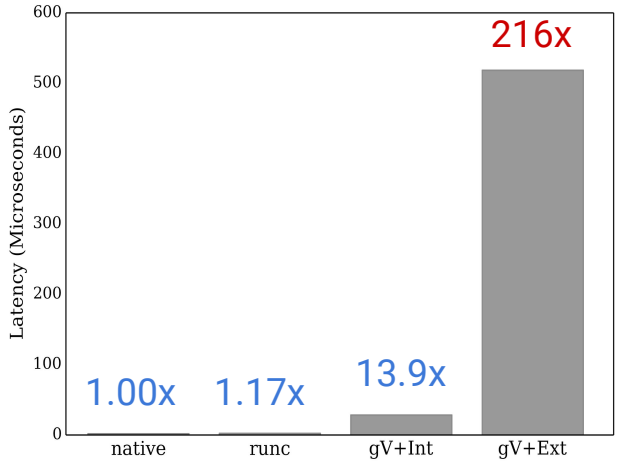
In summary, gVisor offers improved isolation at a cost to application performance

In summary, gVisor offers improved isolation at a cost to application performance

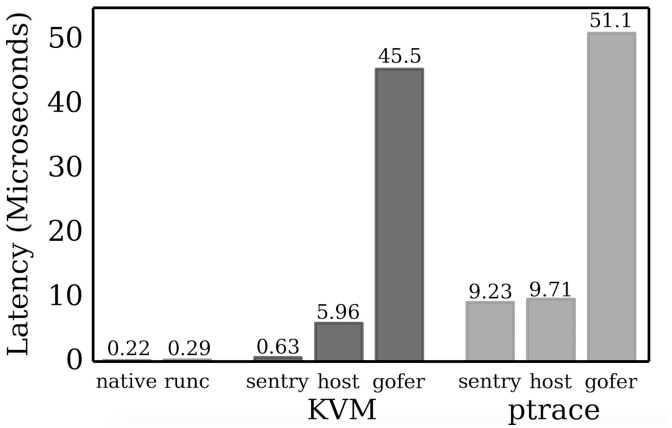


<u>Storage</u>: gVisor has high latency when operations use the gofer

In summary, gVisor offers improved isolation at a cost to application performance

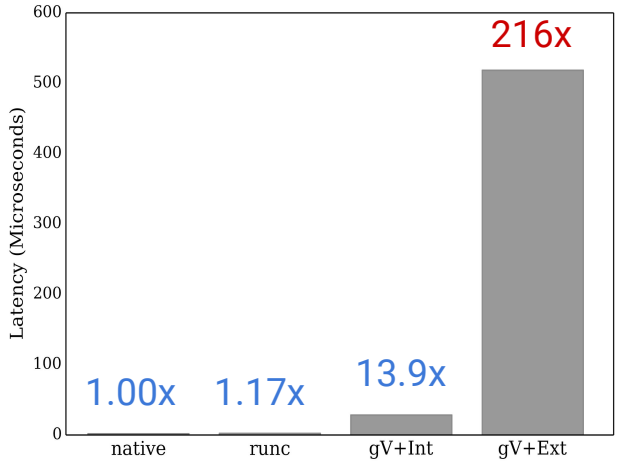


<u>Storage</u>: gVisor has high latency when operations use the gofer

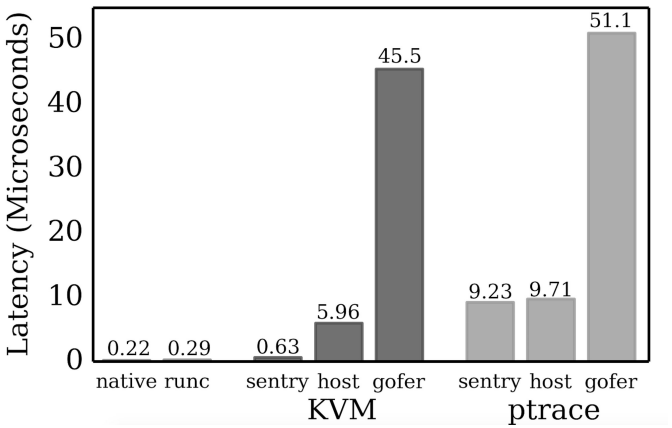


<u>System Calls</u>: gVisor incurs a cost for system call heavy workloads

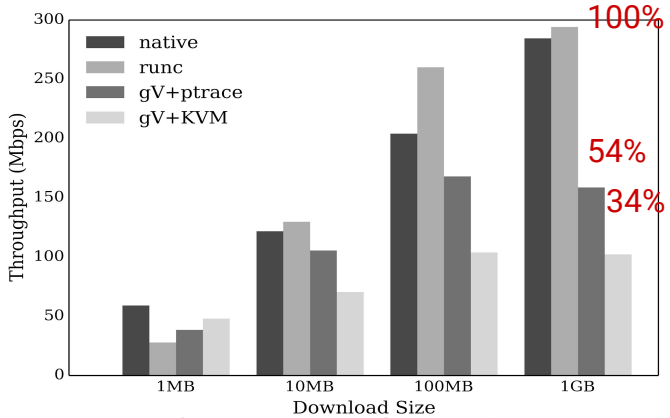
In summary, gVisor offers improved isolation at a cost to application performance



Storage: gVisor has high latency when operations use the gofer

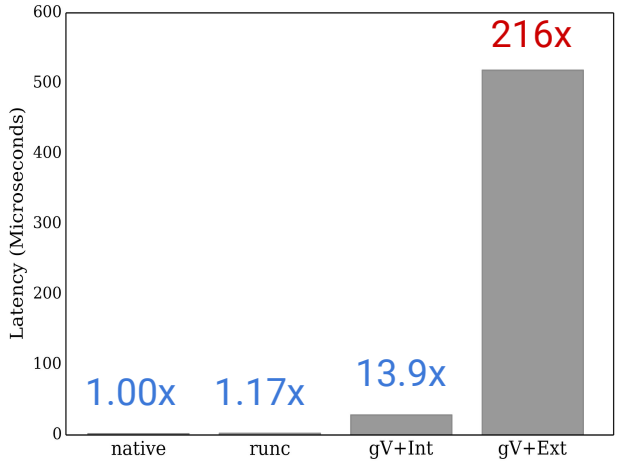


System Calls: gVisor incurs a cost for system call heavy workloads



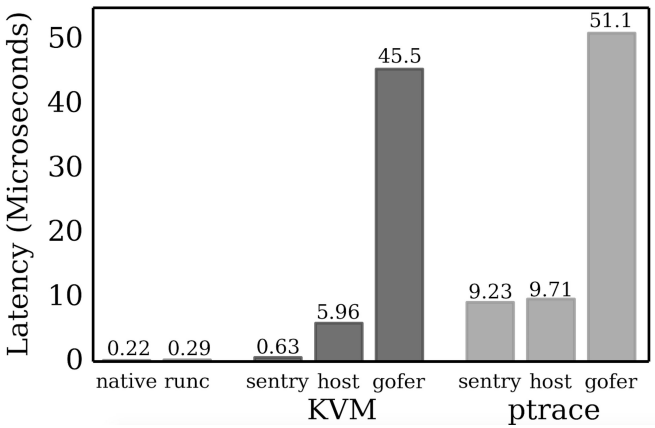
Network: gVisor has degraded download perf at large sizes

In summary, gVisor offers improved isolation at a cost to application performance

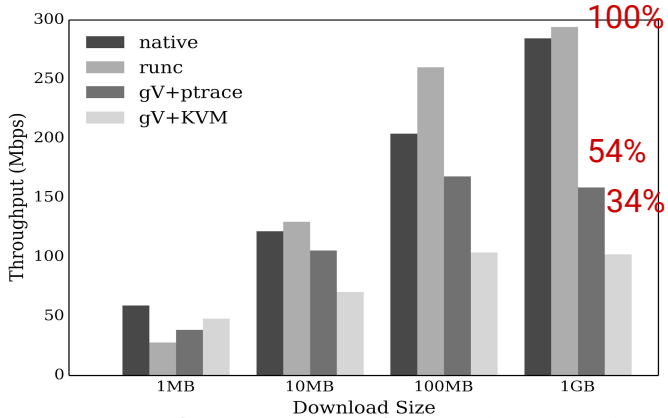


Storage: gVisor has high latency when operations use the gofer

Focus



System Calls: gVisor incurs a cost for system call heavy workloads

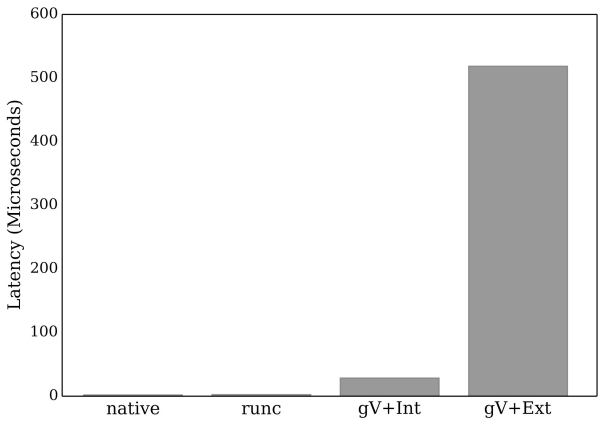


Network: gVisor has degraded download perf at large sizes

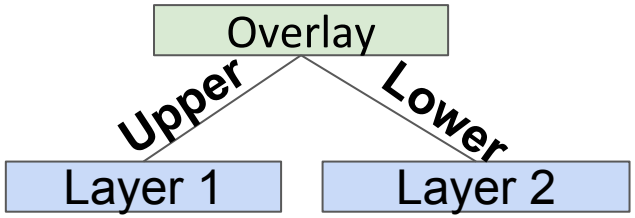
This talk analyzes gVisor, the platform backing Google Cloud Functions, and offers a file system to improve performance

gVisor

gVisor Architecture

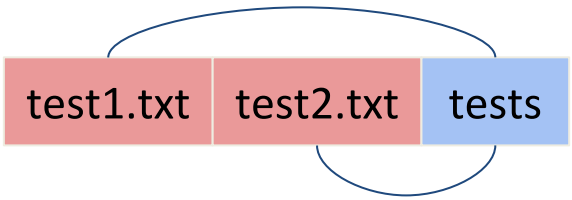


gVisor Performance Analysis

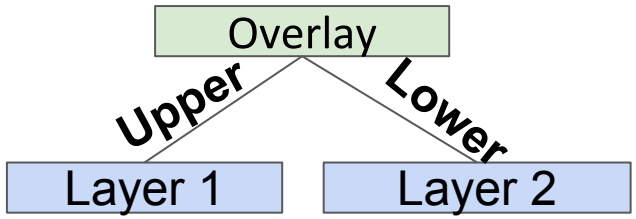


ContainerFS Overview

The second half of this talk explores the architecture of ContainerFS, a gVisor file system for performance



ImgFS Architecture

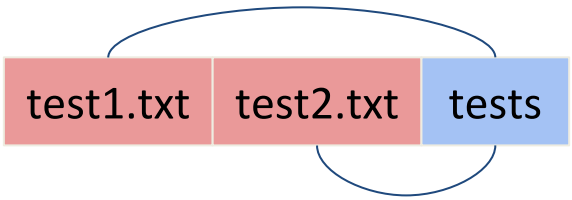


Mounting ImgFS Layers

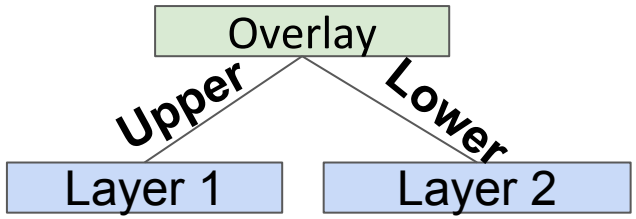
Layer	/	file2.txt
Layer 1	1	Used
Layer 2	2	

ContainerFS Traversal Algorithms

The second half of this talk explores the architecture of ContainerFS, a gVisor file system for performance



ImgFS Architecture

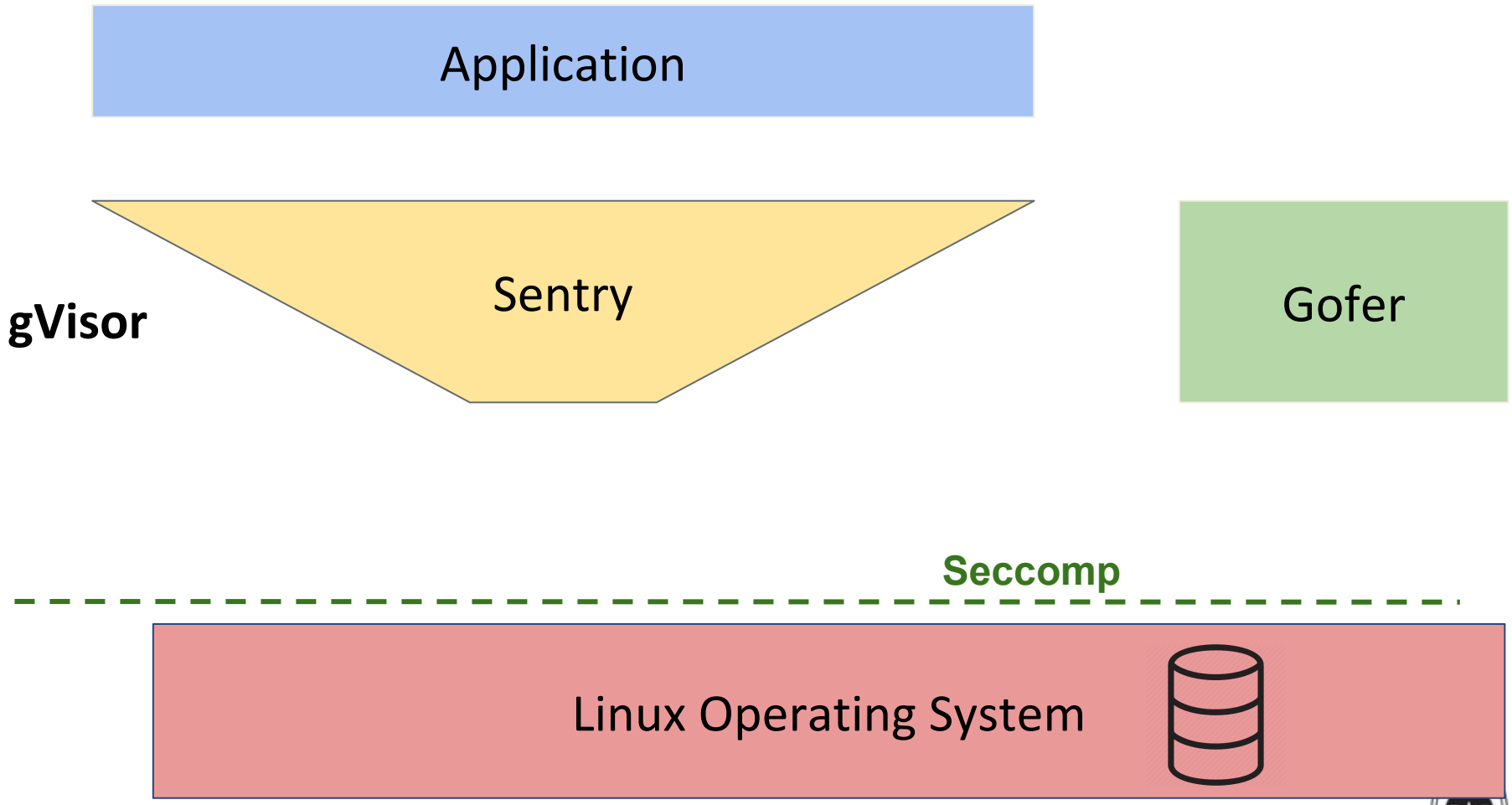


Mounting ImgFS Layers

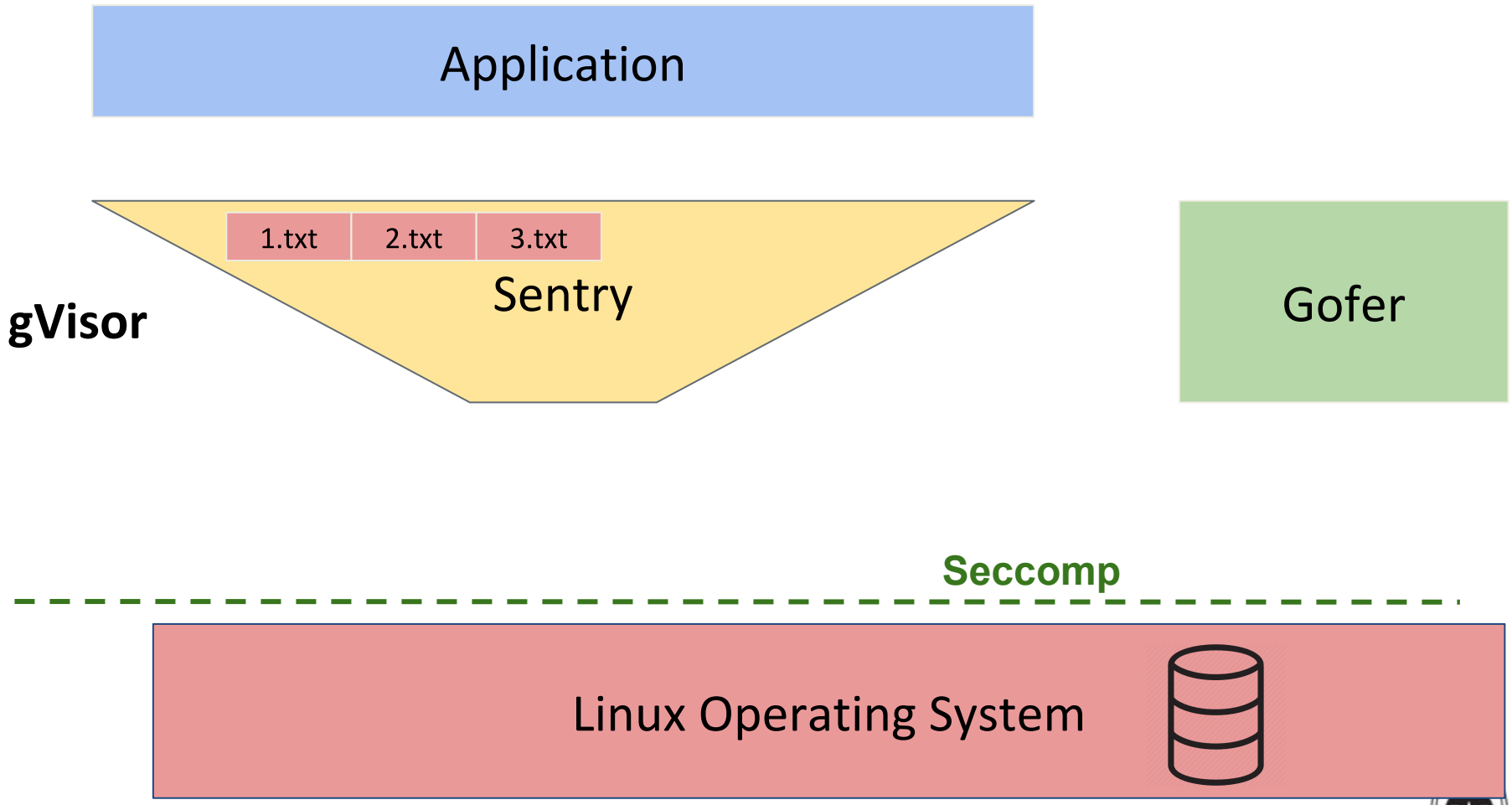
Layer	/	file2.txt
Layer 1	1	Used
Layer 2	2	

ContainerFS Traversal Algorithms

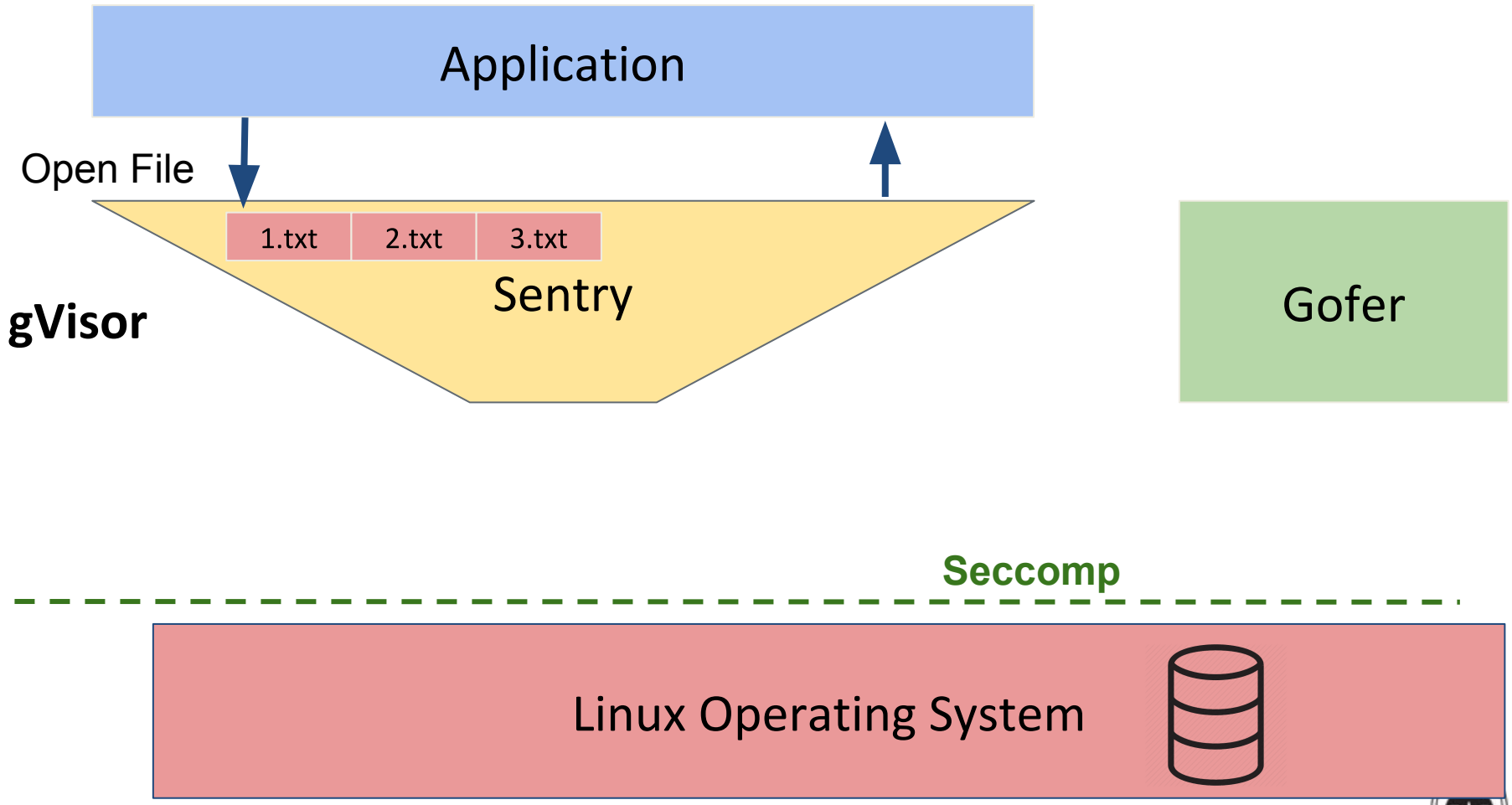
ImgFS preloads common files into the Sentry at startup to avoid involving the Gofer and host



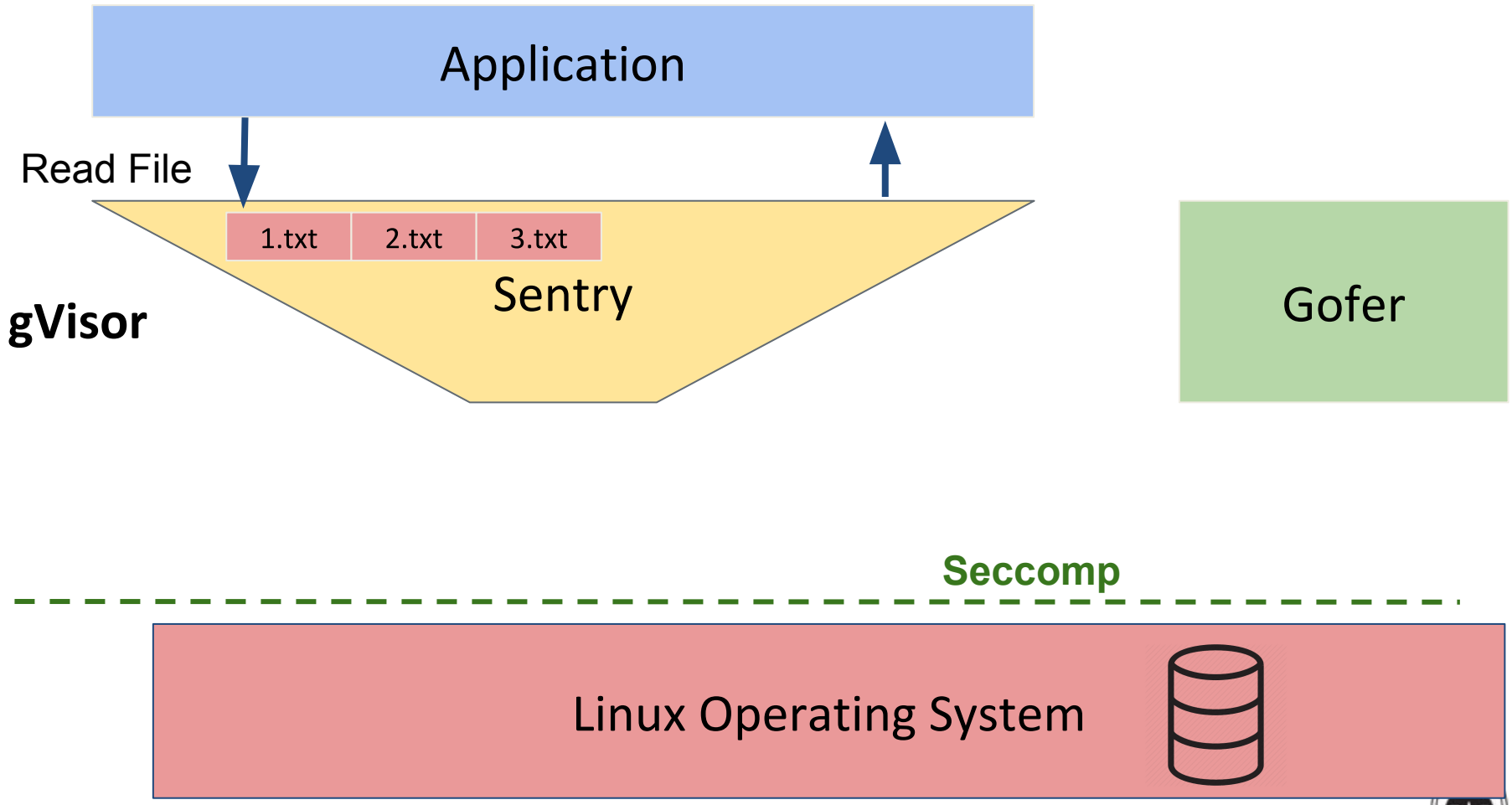
ImgFS preloads common files into the Sentry at startup to avoid involving the Gofer and host



ImgFS preloads common files into the Sentry at startup to avoid involving the Gofer and host

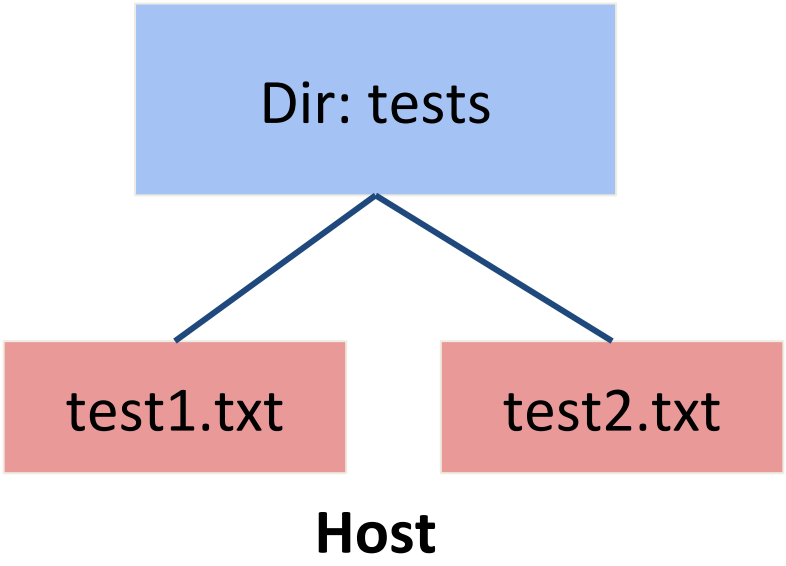


ImgFS preloads common files into the Sentry at startup to avoid involving the Gofer and host

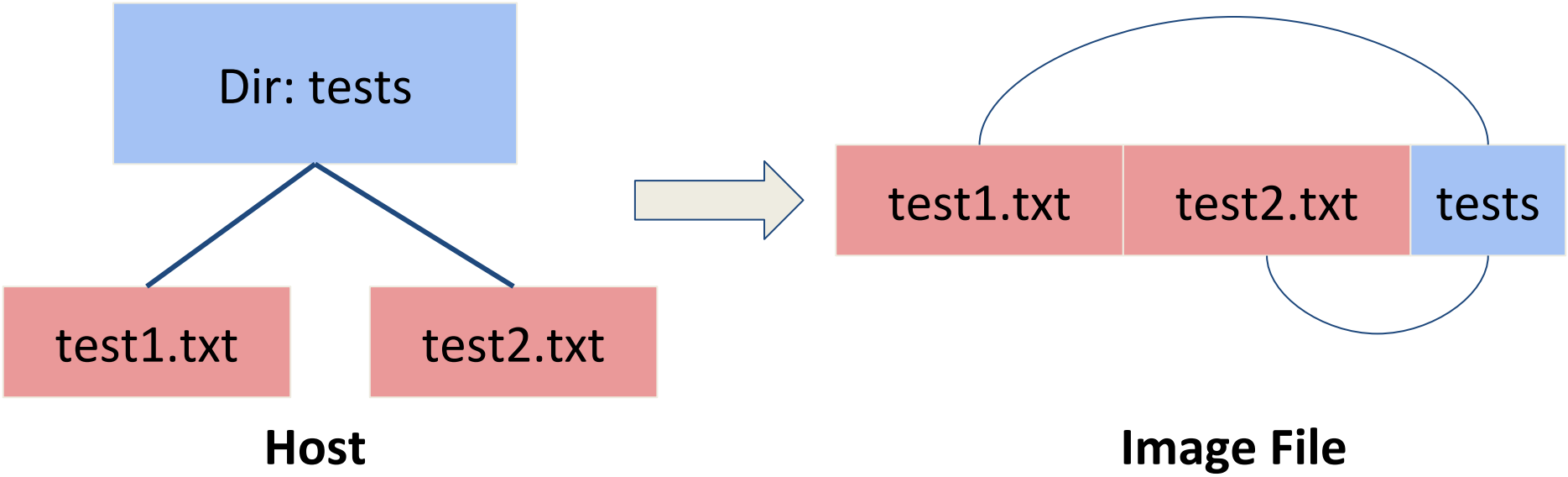


ImgFS uses an image file to encode the files and directory structure that will be preloaded into gVisor

ImgFS uses an image file to encode the files and directory structure that will be preloaded into gVisor

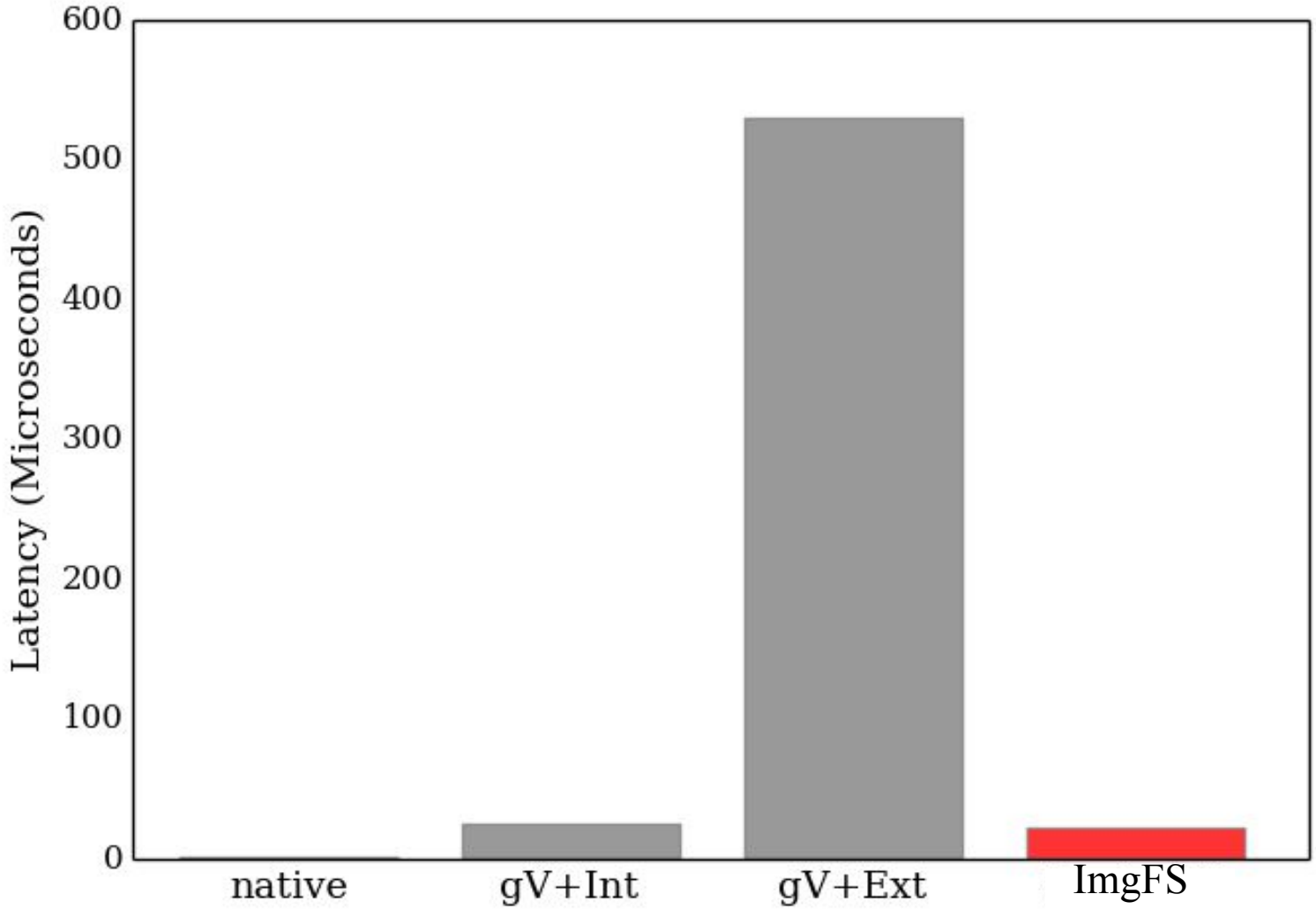


ImgFS uses an image file to encode the files and directory structure that will be preloaded into gVisor

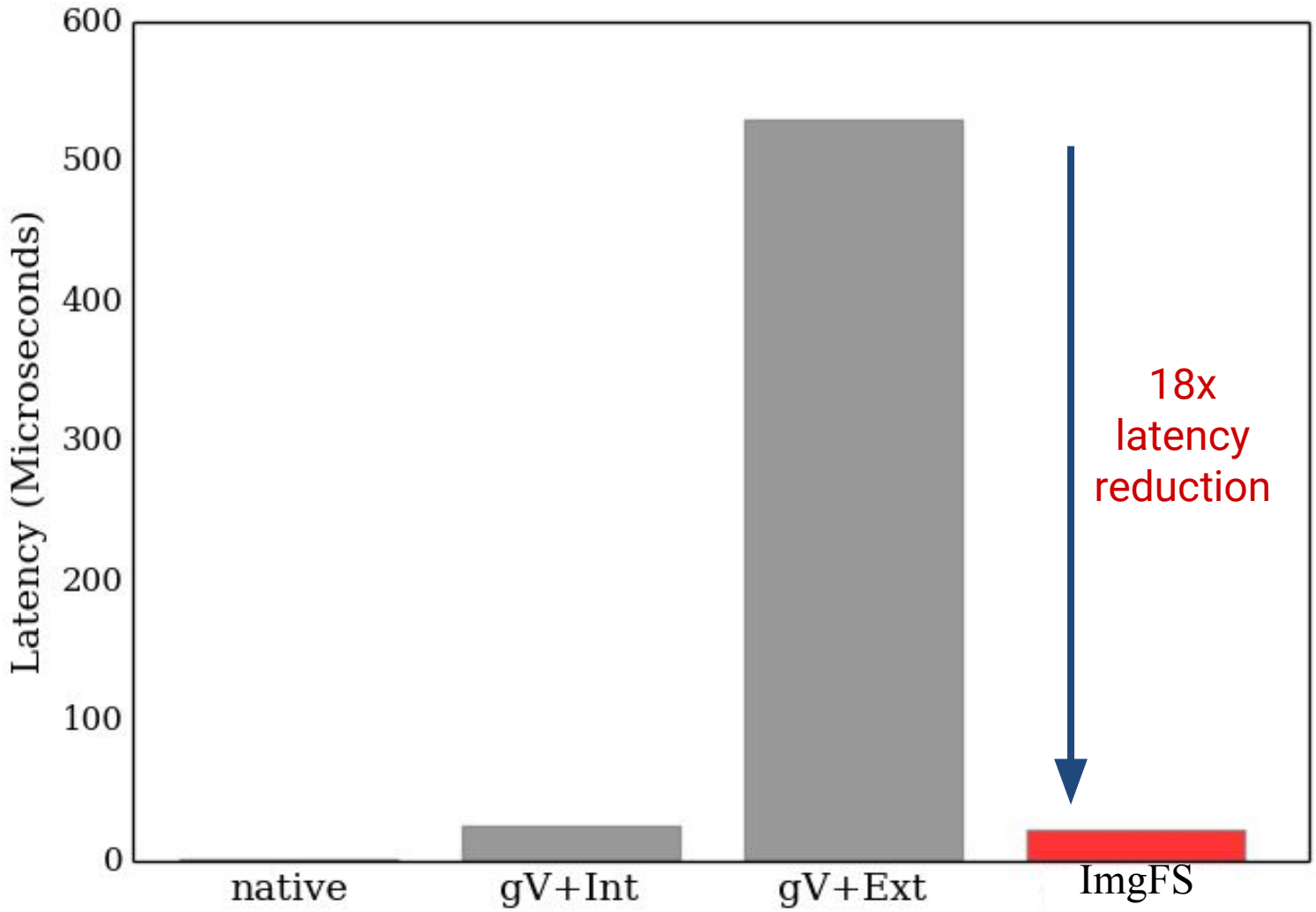


Serving the opening of files from the preloaded image file eliminates Gofer access latencies

Serving the opening of files from the preloaded image file eliminates Gofer access latencies

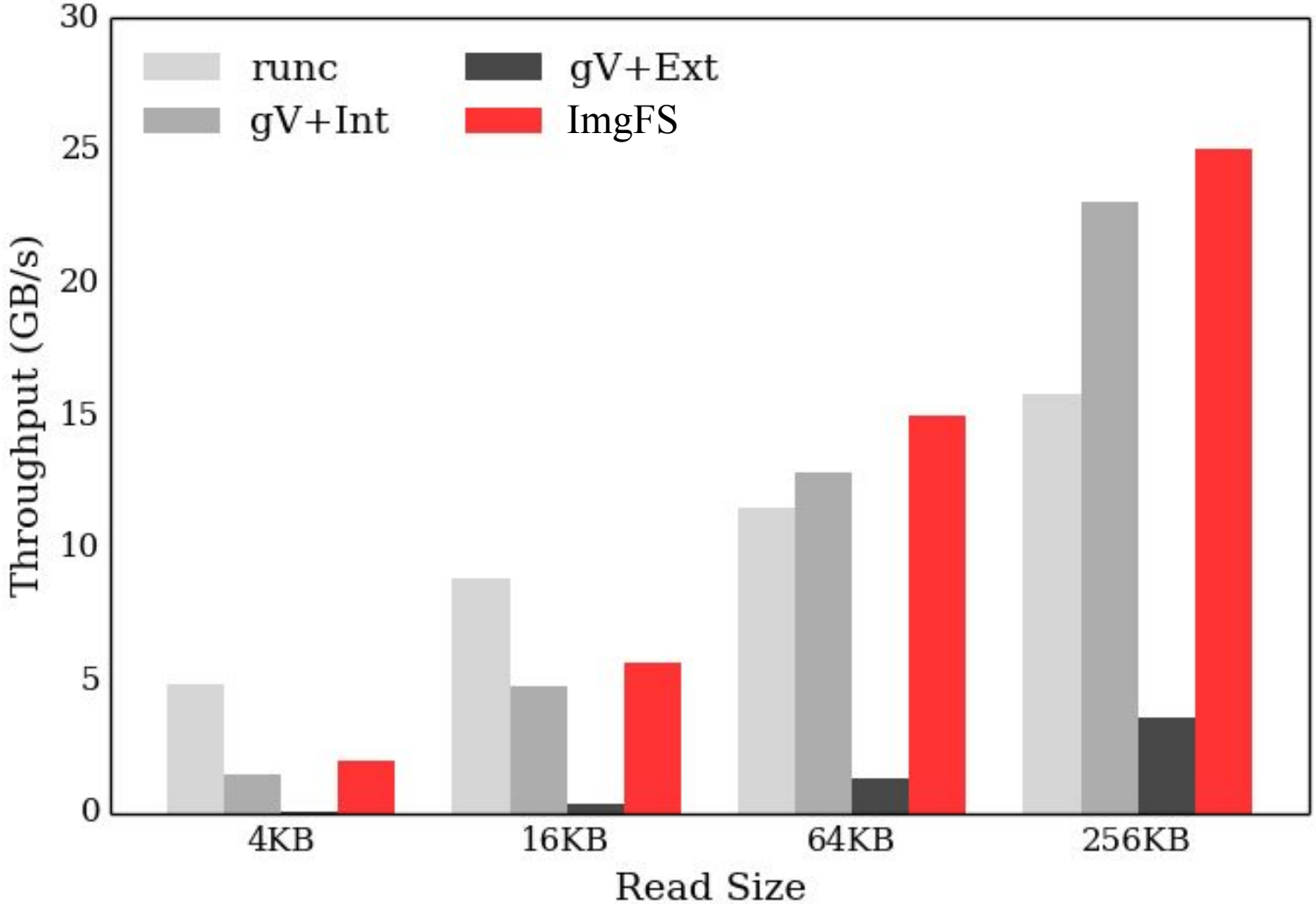


Serving the opening of files from the preloaded image file eliminates Gofer access latencies

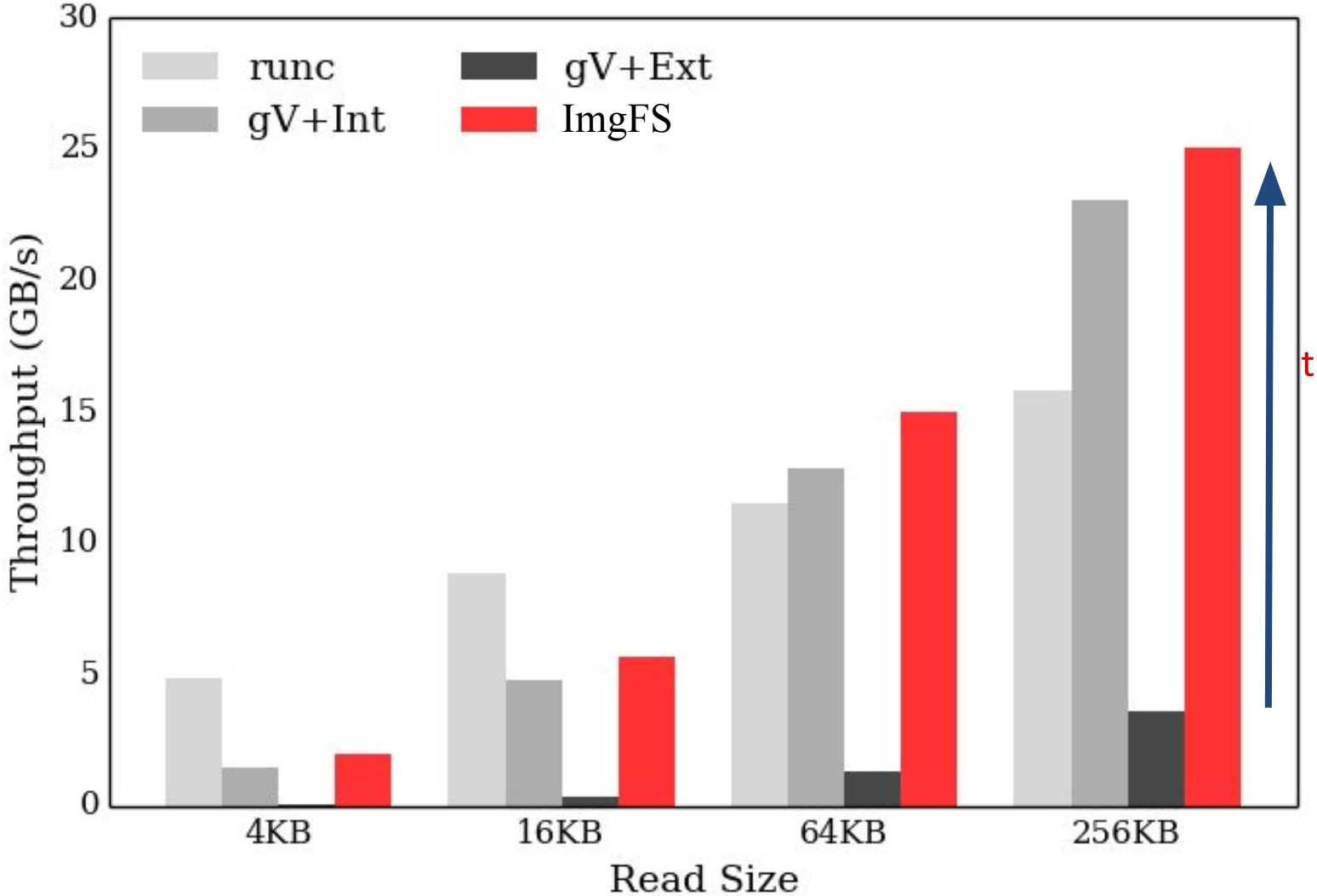


Preloading an image file into the Sentry significantly increases the achievable throughput for reading a file

Preloading an image file into the Sentry significantly increases the achievable throughput for reading a file

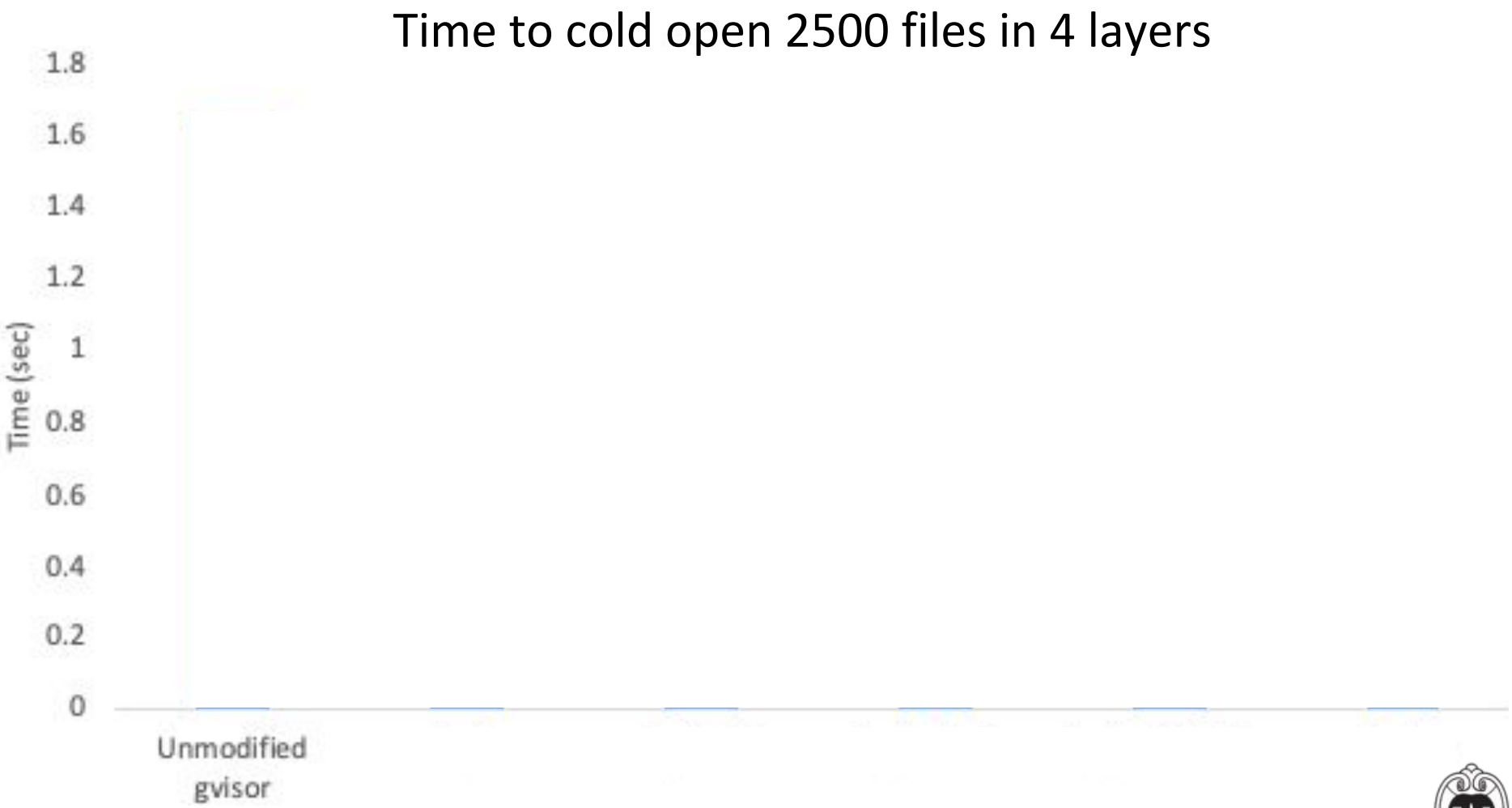


Preloading an image file into the Sentry significantly increases the achievable throughput for reading a file

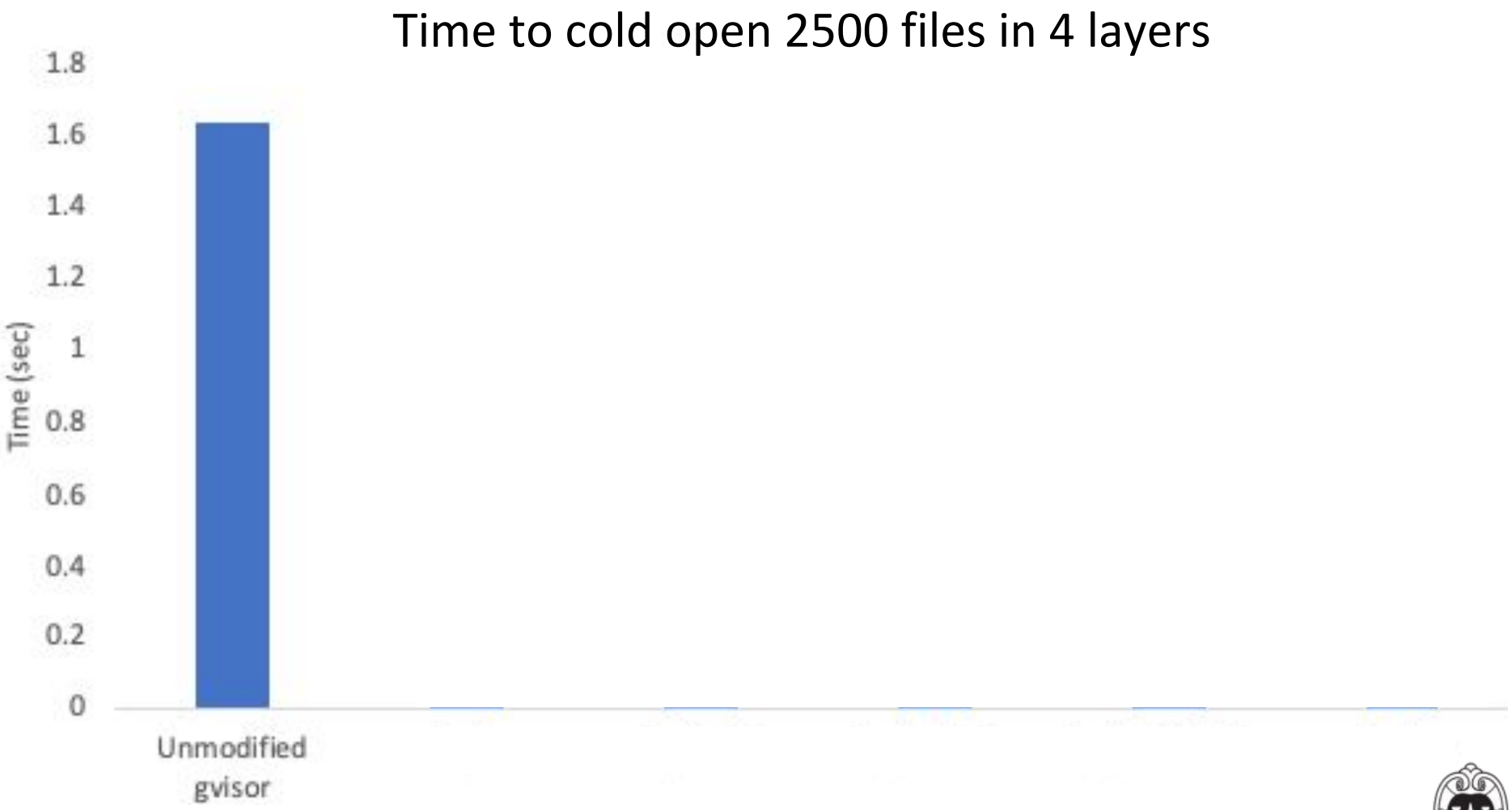


22x throughput increase

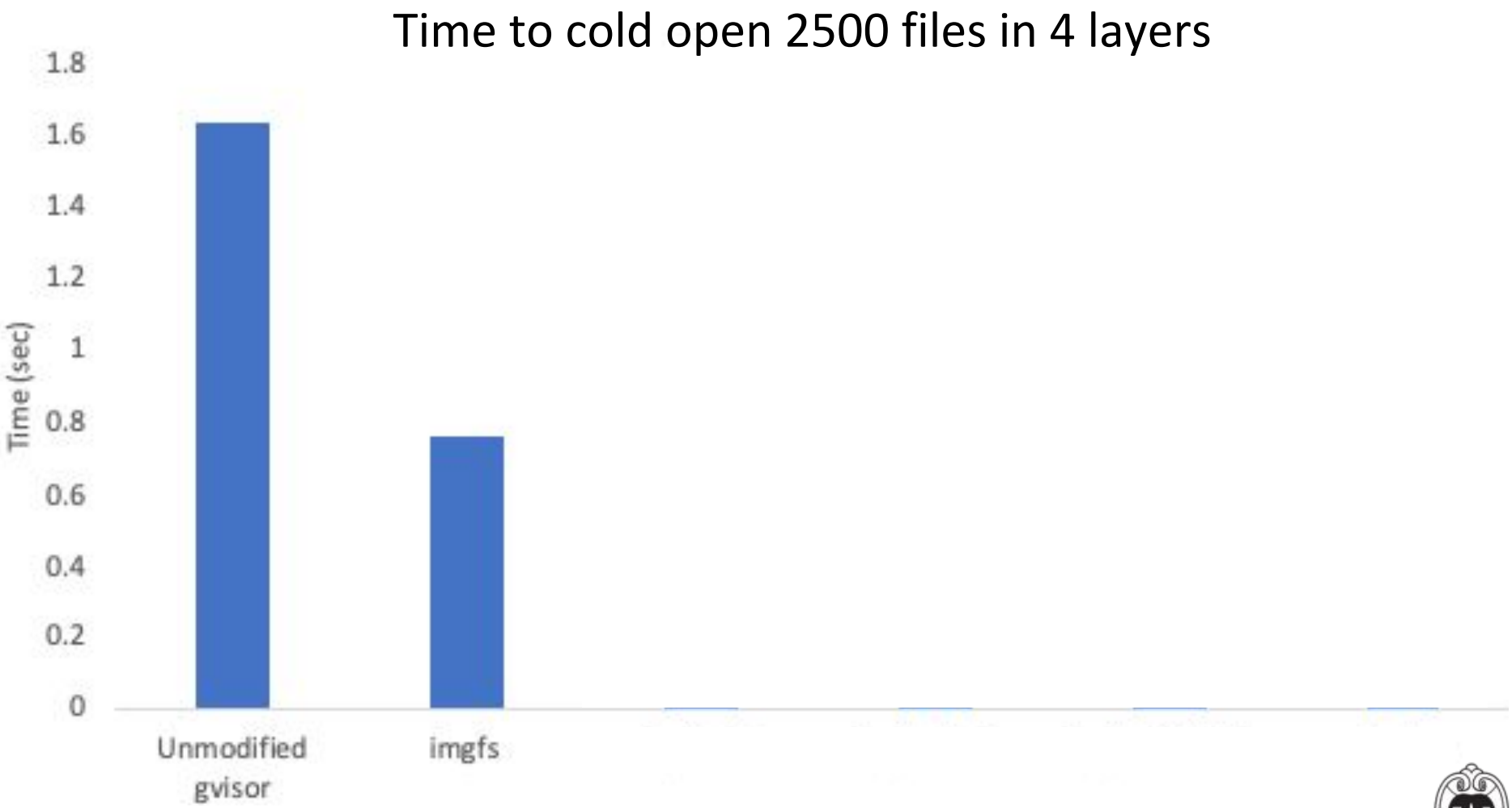
ImgFS decreases the latency for cold opens compared to unmodified gVisor



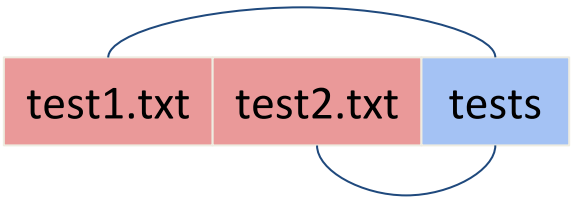
ImgFS decreases the latency for cold opens compared to unmodified gVisor



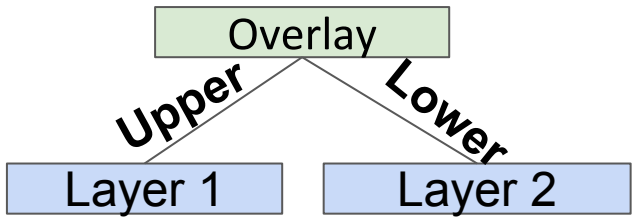
ImgFS decreases the latency for cold opens compared to unmodified gVisor



The second half of this talk explores the architecture of ContainerFS, a gVisor file system for performance



ImgFS Architecture



Mounting ImgFS Layers

Layer	/	file2.txt
Layer 1	1	Used
Layer 2	2	

ContainerFS Traversal Algorithms

Docker images are a stacked collection of file system layers made available to the container

Dockerfile

```
FROM alpine:3.4

RUN apk add curl
RUN apk add vim
```

Container File System

Docker images are a stacked collection of file system layers made available to the container

Dockerfile

```
FROM alpine:3.4

RUN apk add curl
RUN apk add vim
```

Container File System

FROM alpine:3.4

Contains all files for alpine image

Docker images are a stacked collection of file system layers made available to the container

Dockerfile

```
FROM alpine:3.4

RUN apk add curl
RUN apk add vim
```

Container File System

RUN apk add curl

FROM alpine:3.4

Contains all file differences

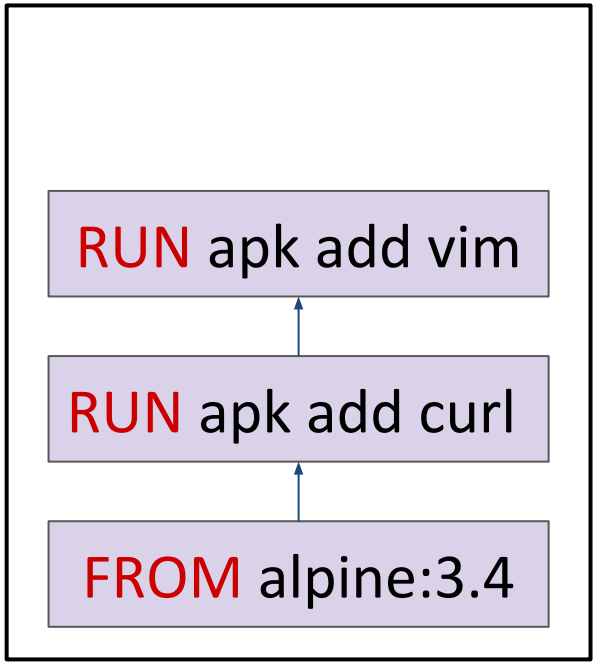
Docker images are a stacked collection of file system layers made available to the container

Dockerfile

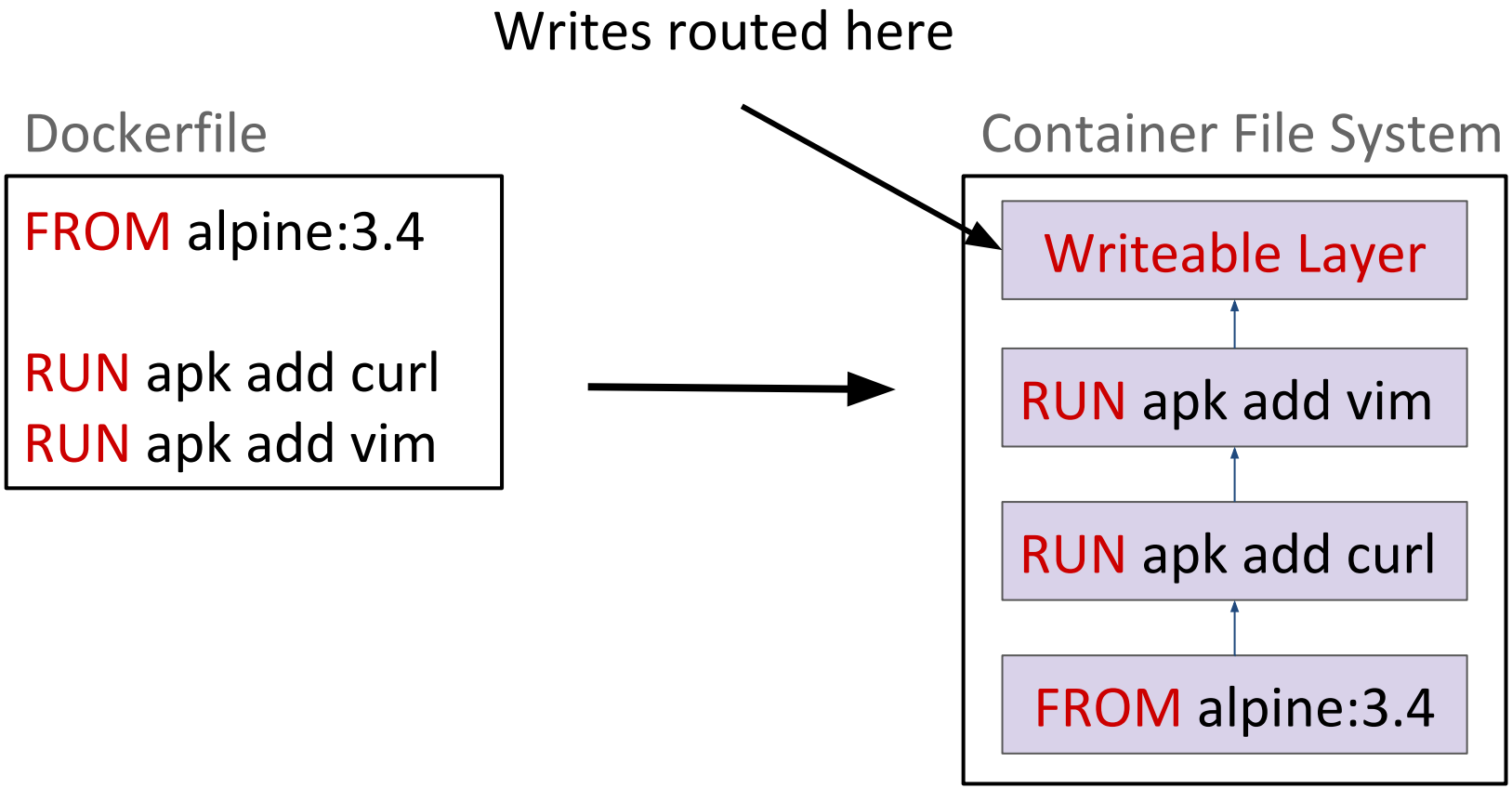
```
FROM alpine:3.4

RUN apk add curl
RUN apk add vim
```

Container File System

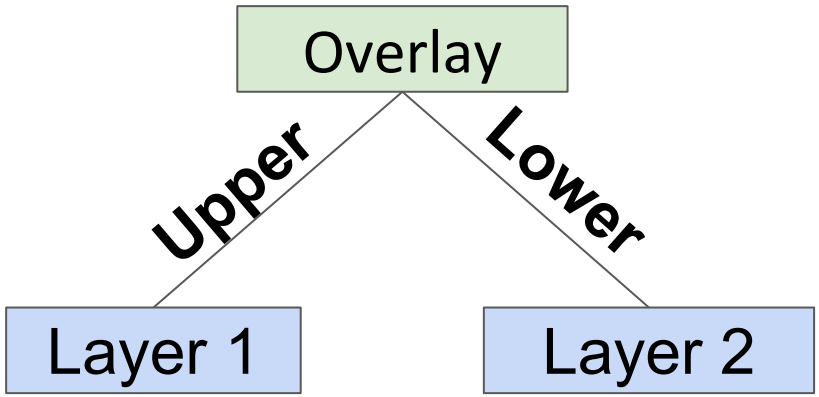


Docker images are a stacked collection of file system layers made available to the container

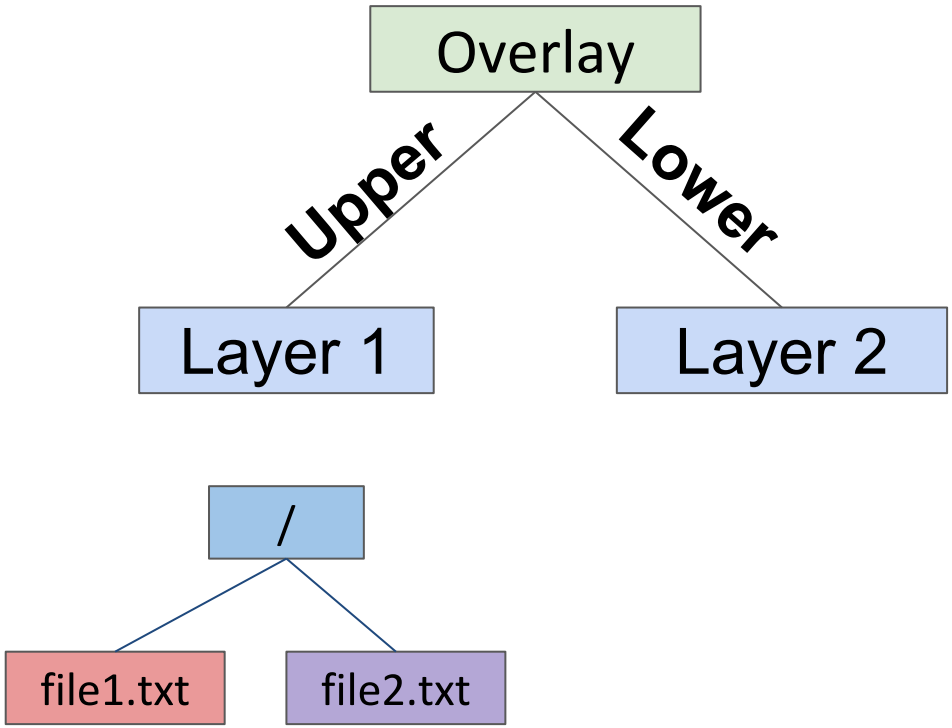


Linux' OverlayFS provides a mechanism creating a merged view of different file systems

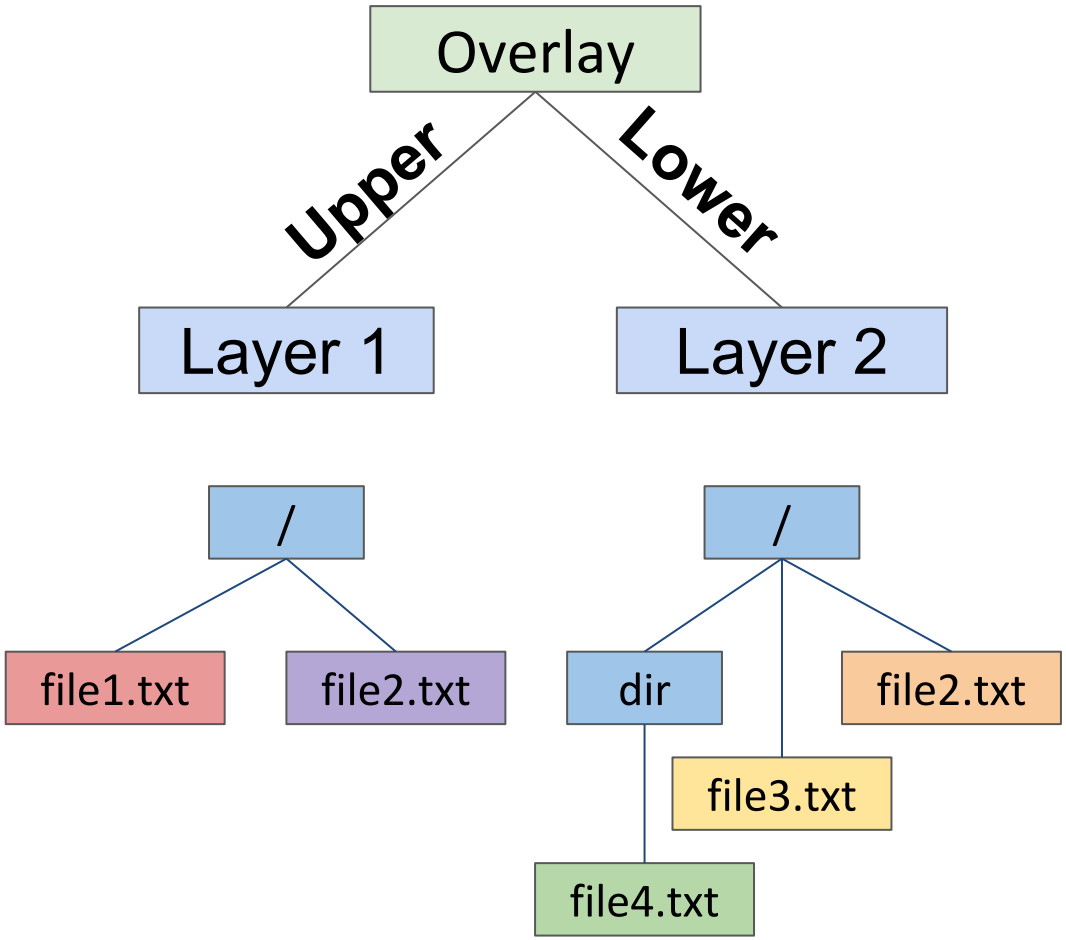
Linux' OverlayFS provides a mechanism creating a merged view of different file systems



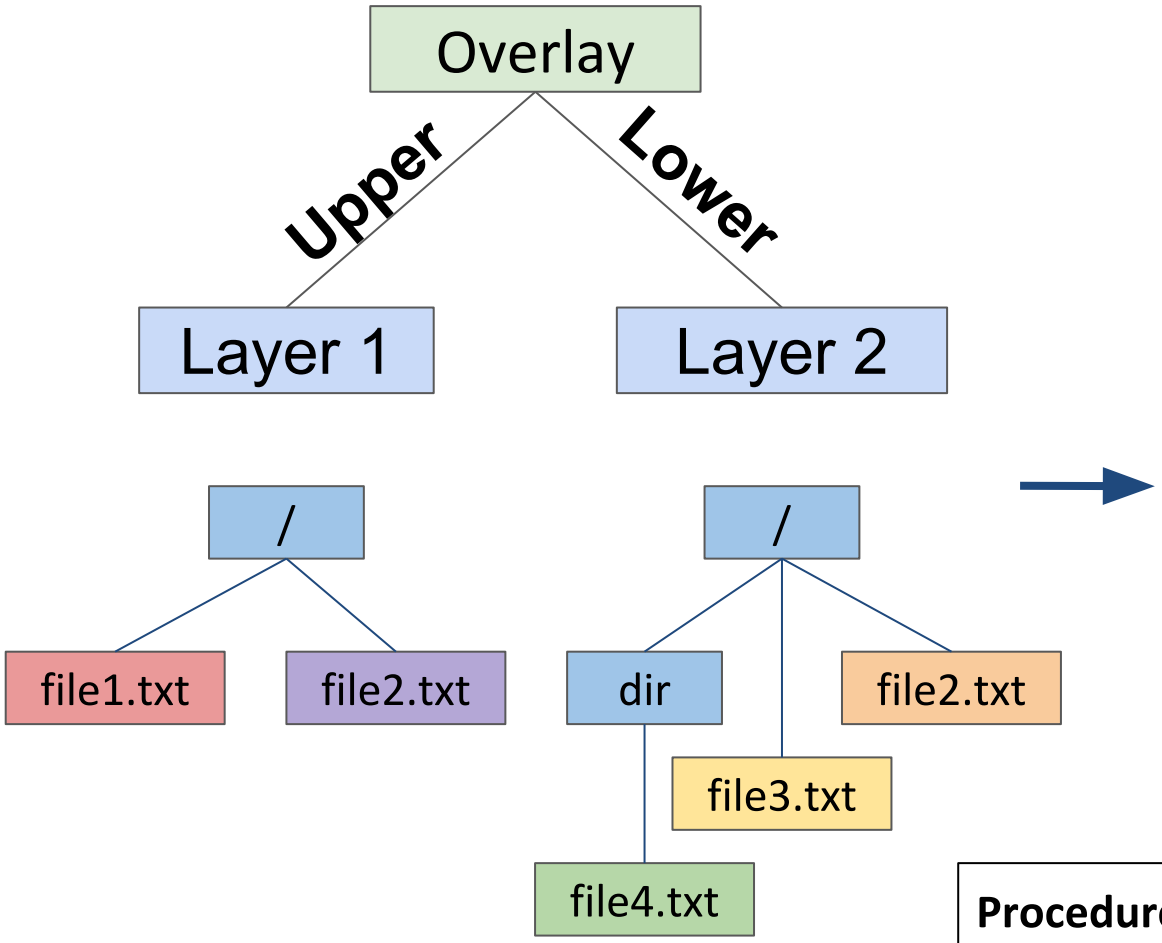
Linux' OverlayFS provides a mechanism creating a merged view of different file systems



Linux' OverlayFS provides a mechanism creating a merged view of different file systems



Linux' OverlayFS provides a mechanism creating a merged view of different file systems

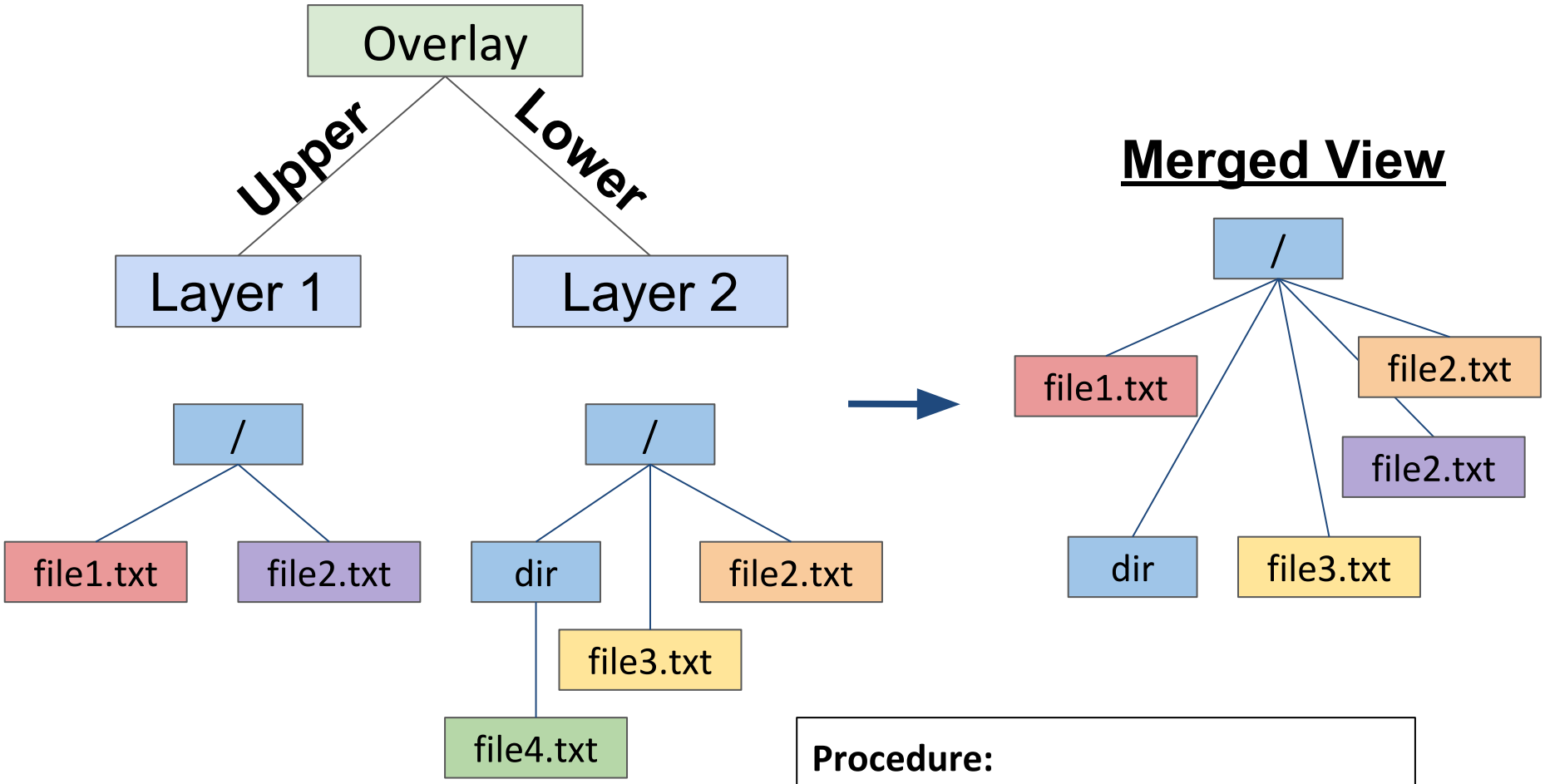


Merged View

Procedure:

- Recursively merge directory contents

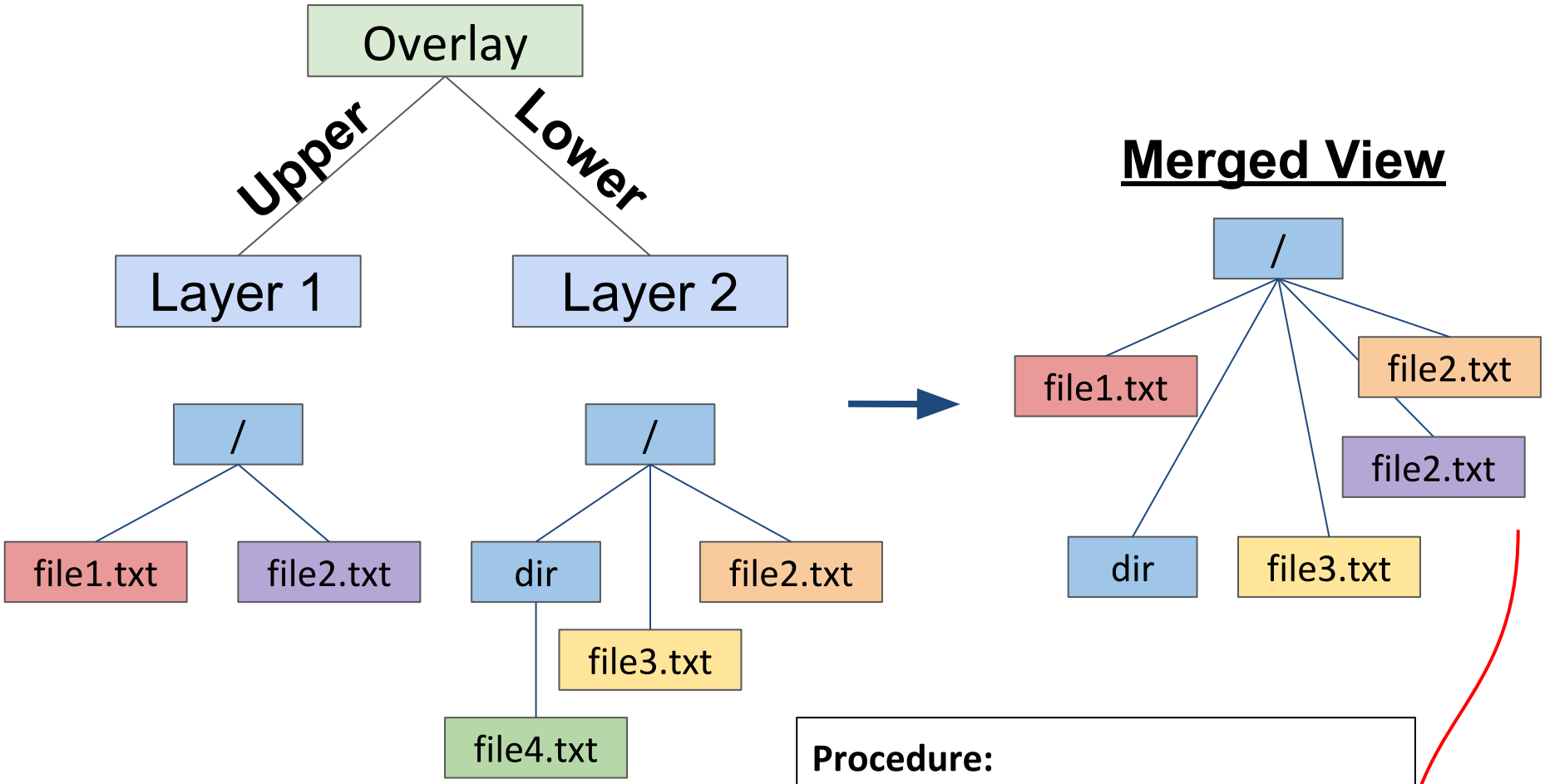
Linux' OverlayFS provides a mechanism creating a merged view of different file systems



Procedure:

- Recursively merge directory contents

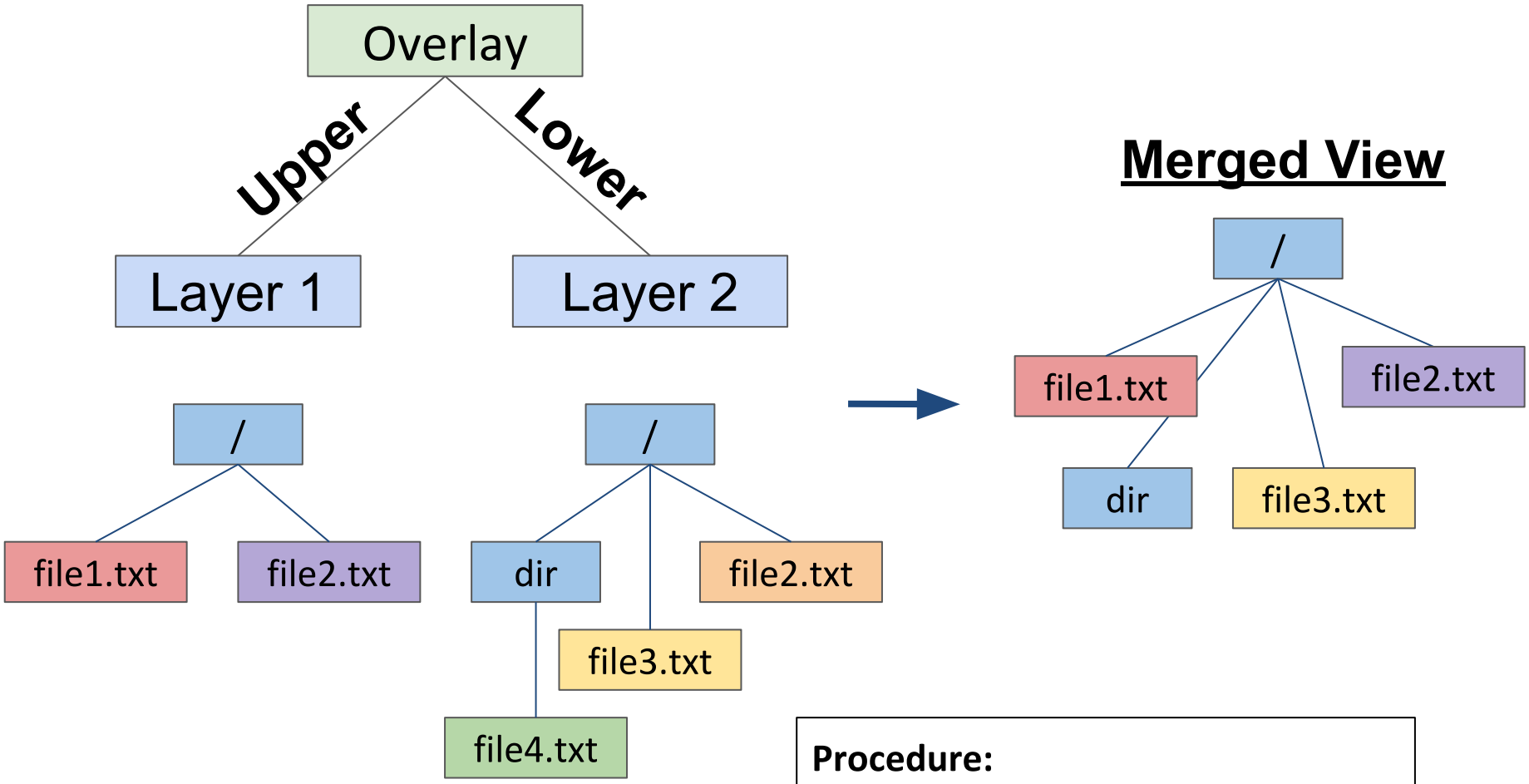
Linux' OverlayFS provides a mechanism creating a merged view of different file systems



Procedure:

- Recursively merge directory contents
- If same file in both, take upper

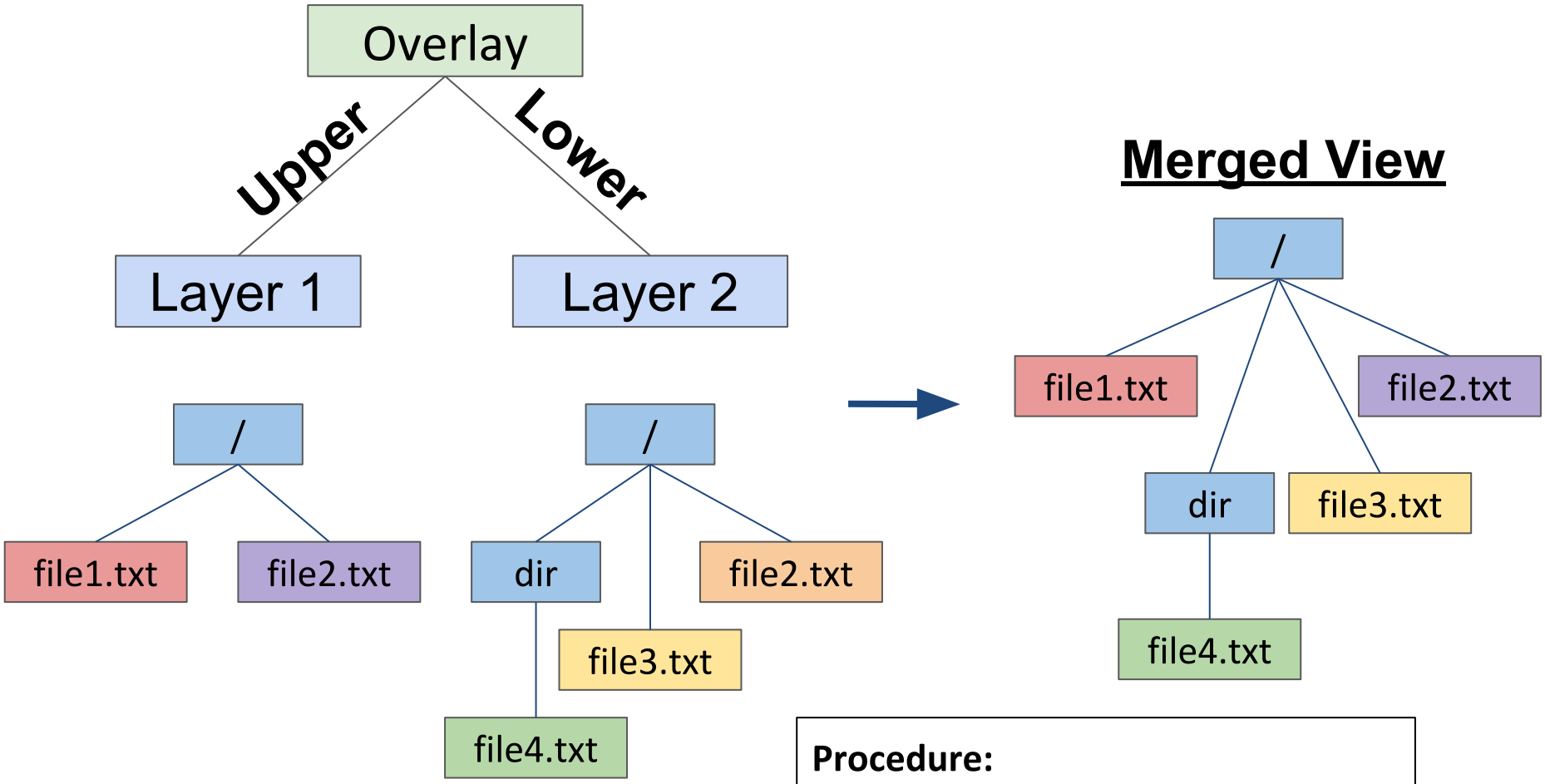
Linux' OverlayFS provides a mechanism creating a merged view of different file systems



Procedure:

- Recursively merge directory contents
- If same file in both, take upper

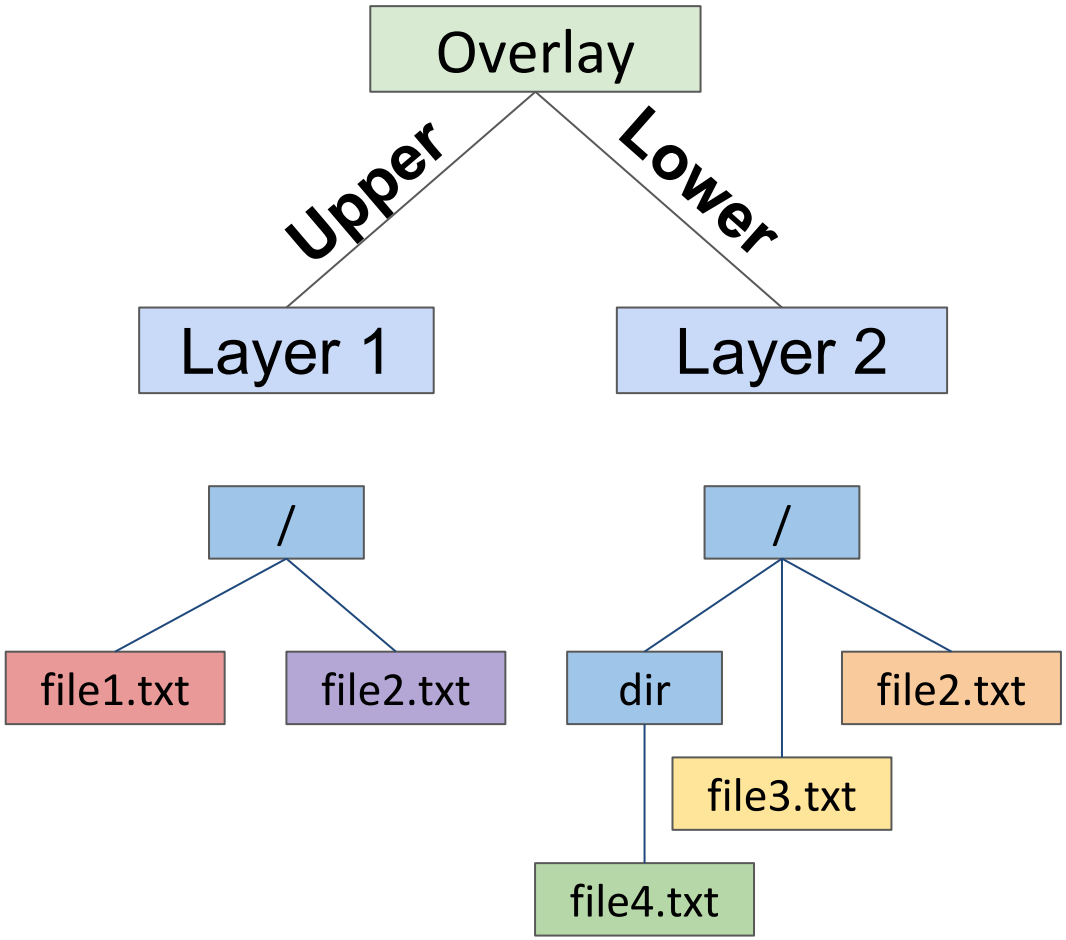
Linux' OverlayFS provides a mechanism creating a merged view of different file systems



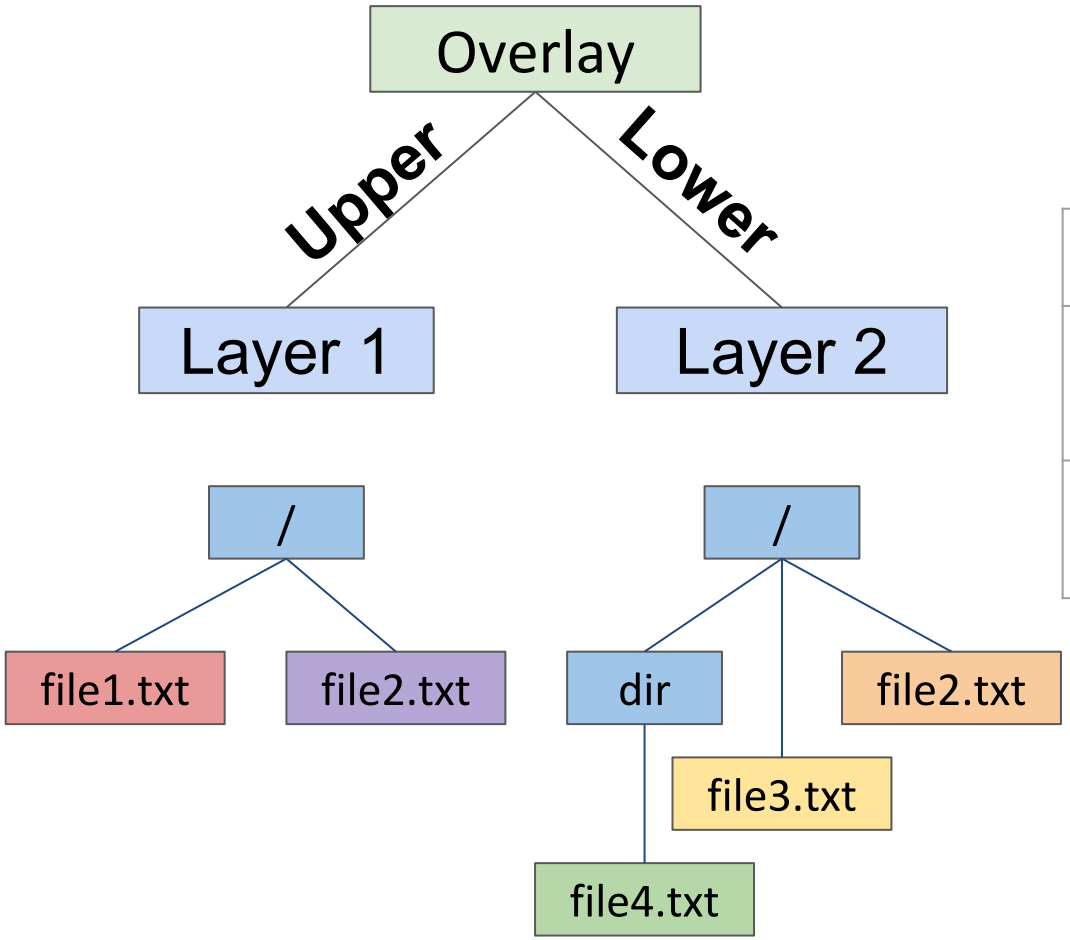
Procedure:

- Recursively merge directory contents
- If same file in both, take upper

OverlayFS resolves the path component by component in the upper FS then the lower FS

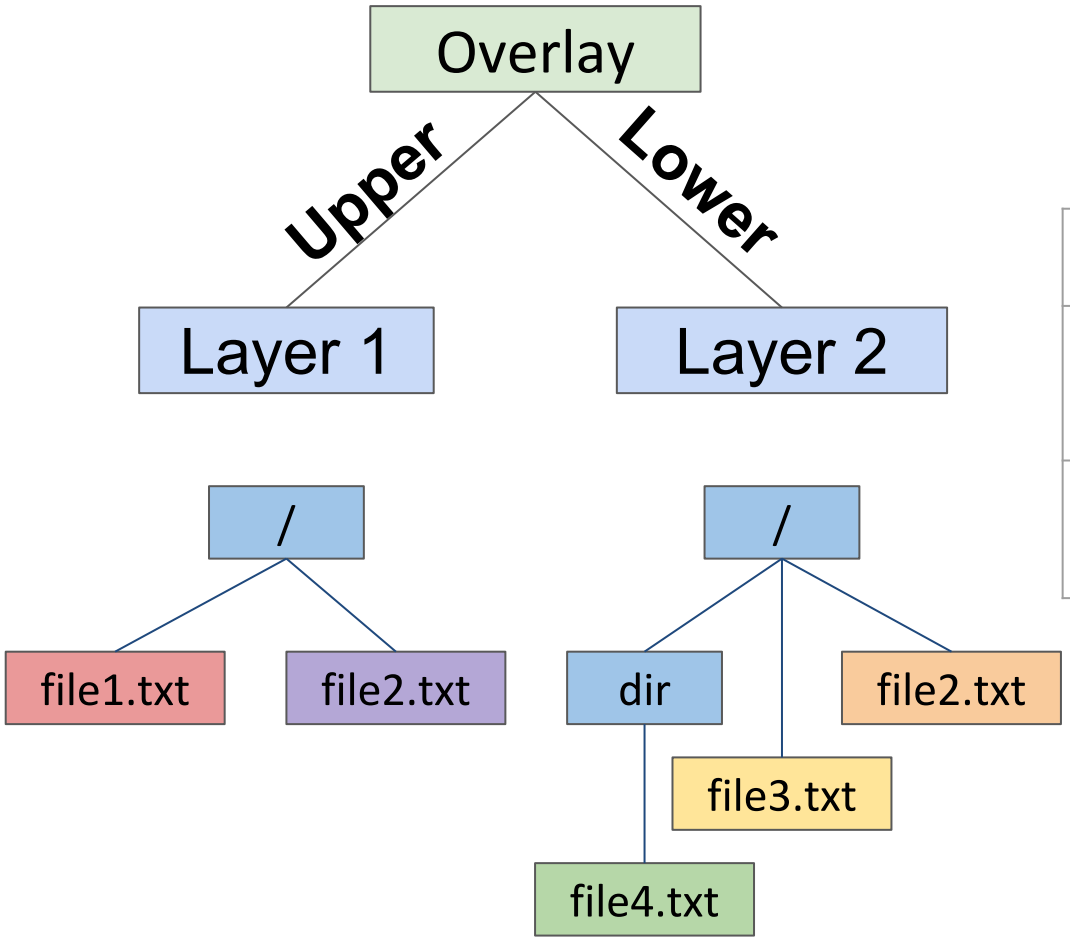


OverlayFS resolves the path component by component in the upper FS then the lower FS



Layer	/	dir/	file4.txt
Layer 1			
Layer 2			

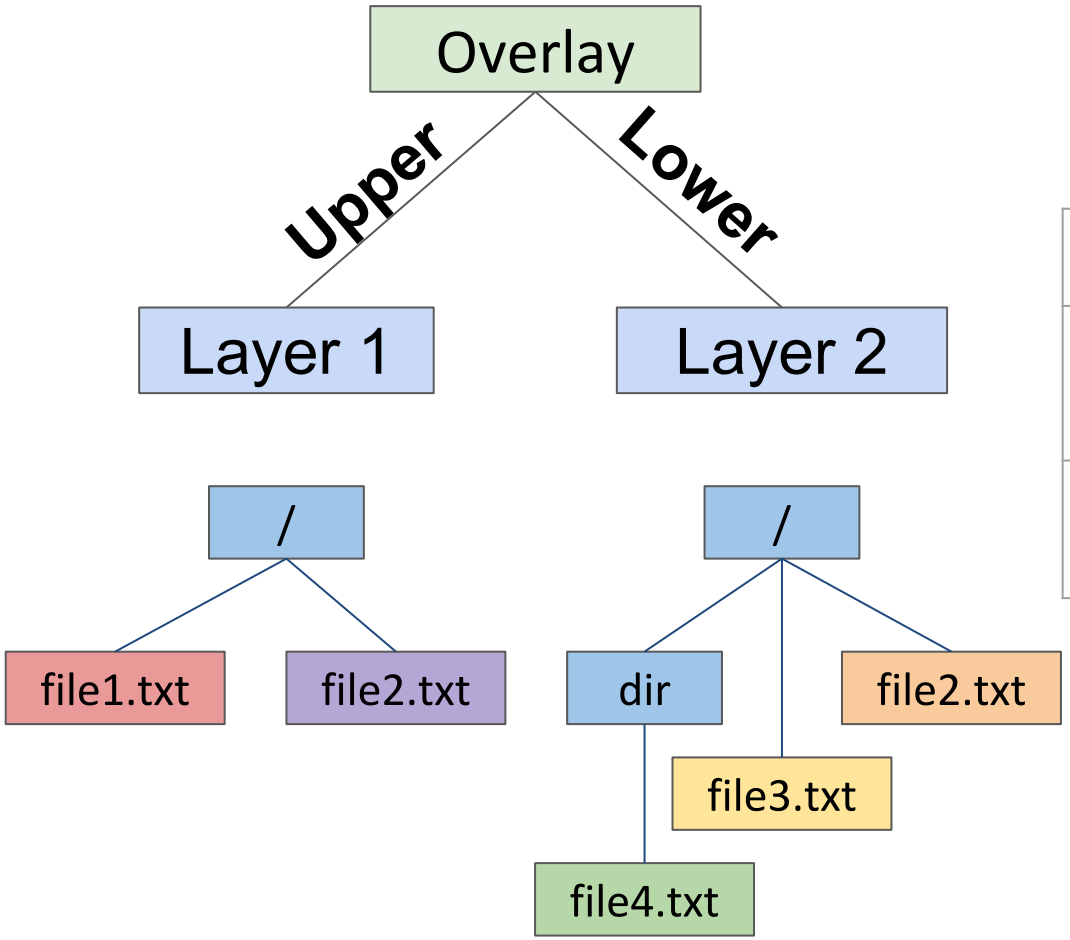
OverlayFS resolves the path component by component in the upper FS then the lower FS



Checking in '/' if 'dir/' exists

Layer	/	dir/	file4.txt
Layer 1			
Layer 2			

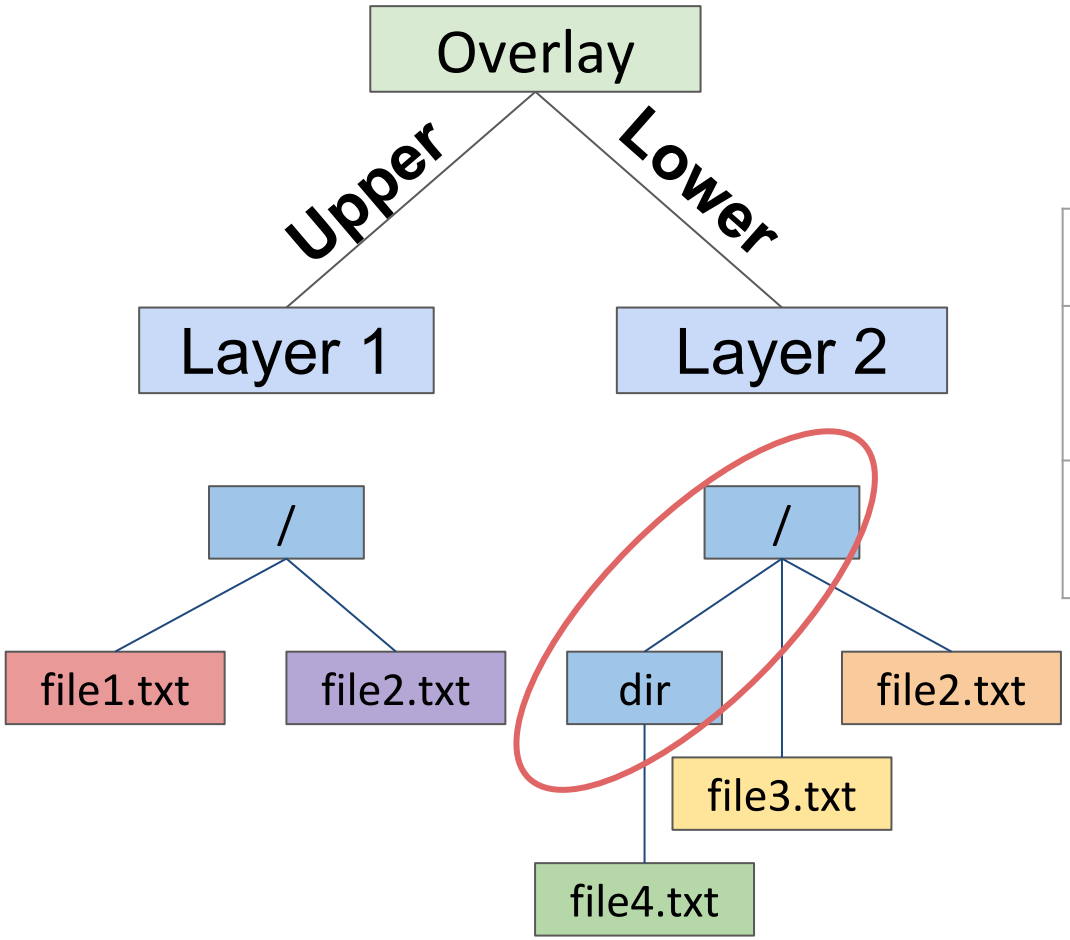
OverlayFS resolves the path component by component in the upper FS then the lower FS



Checking in '/' if 'dir/' exists

Layer	/	dir/	file4.txt
Layer 1	1		
Layer 2			

OverlayFS resolves the path component by component in the upper FS then the lower FS

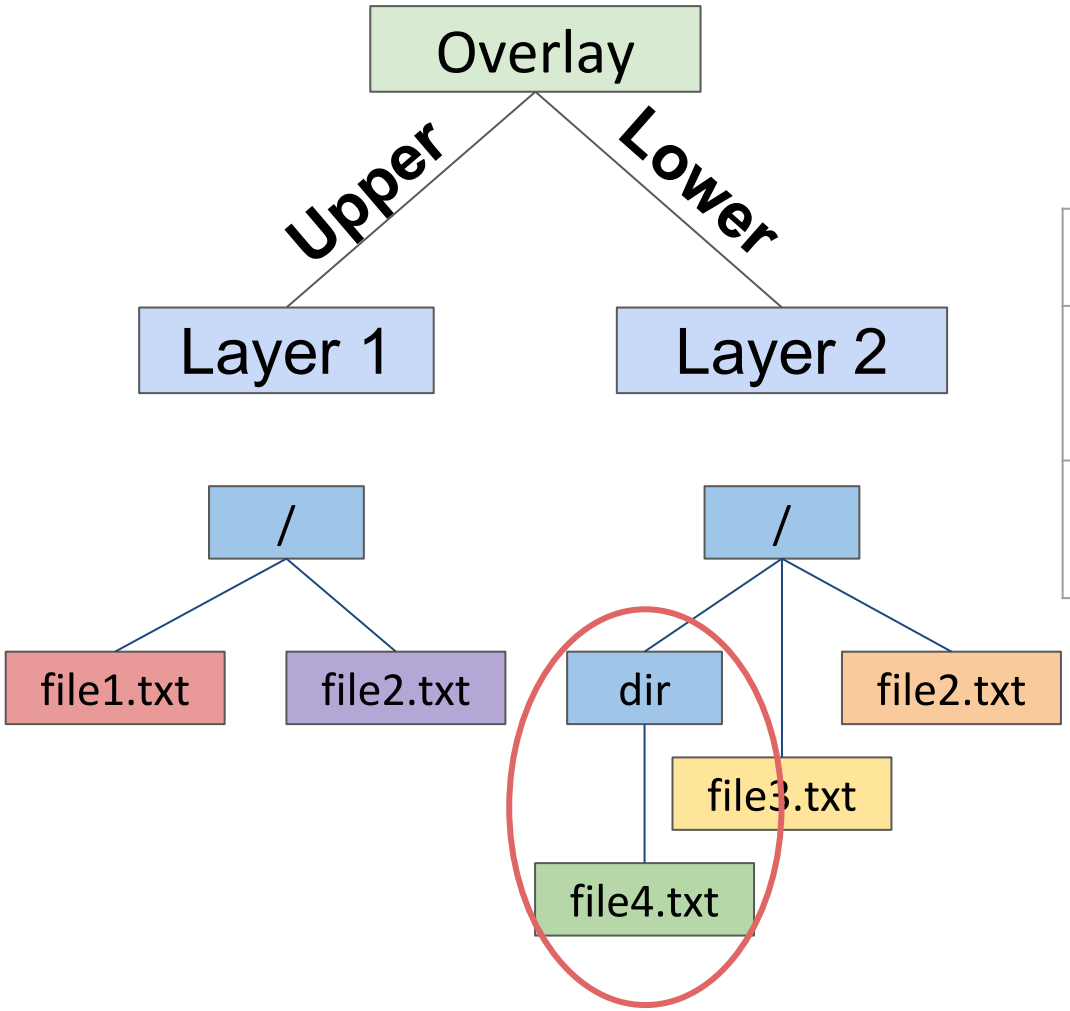


Checking in '/' if 'dir/' exists

Layer	/	dir/	file4.txt
Layer 1	1		
Layer 2	<u>2</u>		

Underlined means success and bold means file found

OverlayFS resolves the path component by component in the upper FS then the lower FS

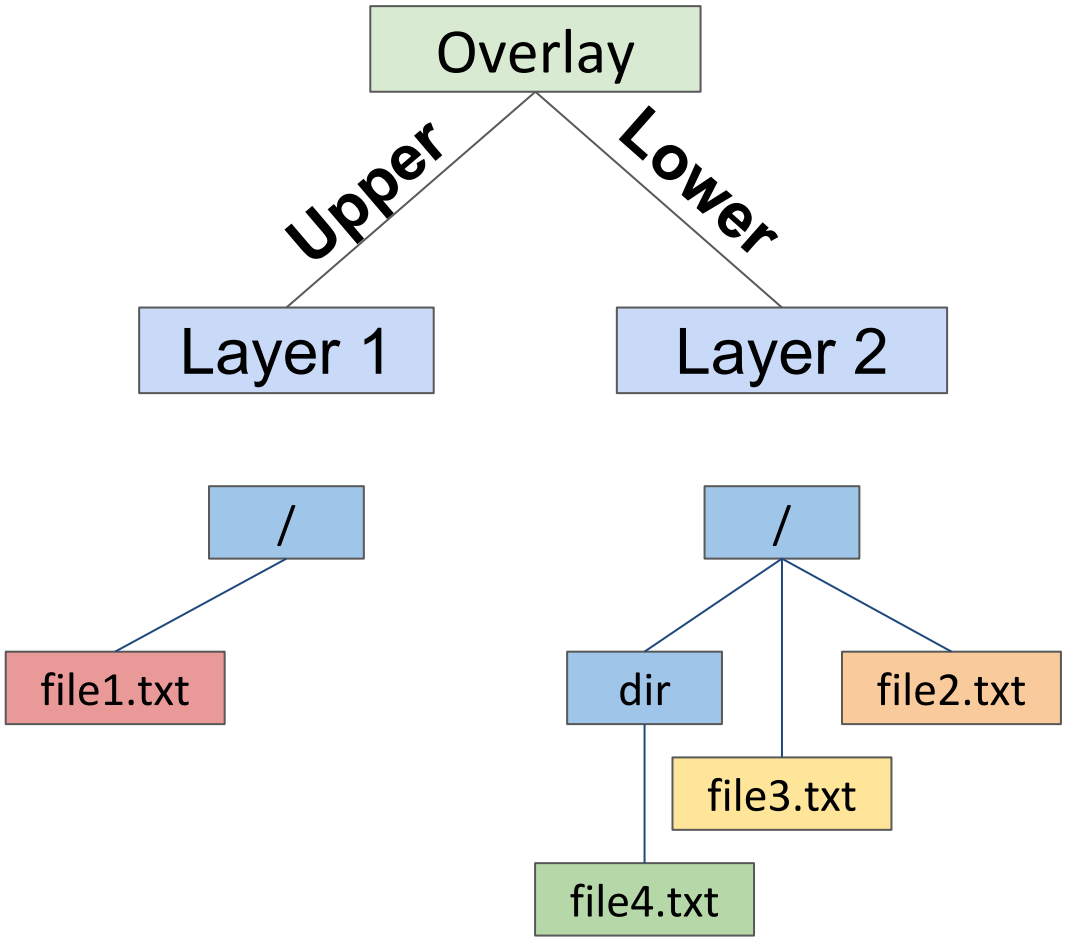


Checking in '/' if 'dir/' exists

Layer	/	dir/	file4.txt
Layer 1	1		
Layer 2	<u>2</u>	<u>**3**</u>	Used

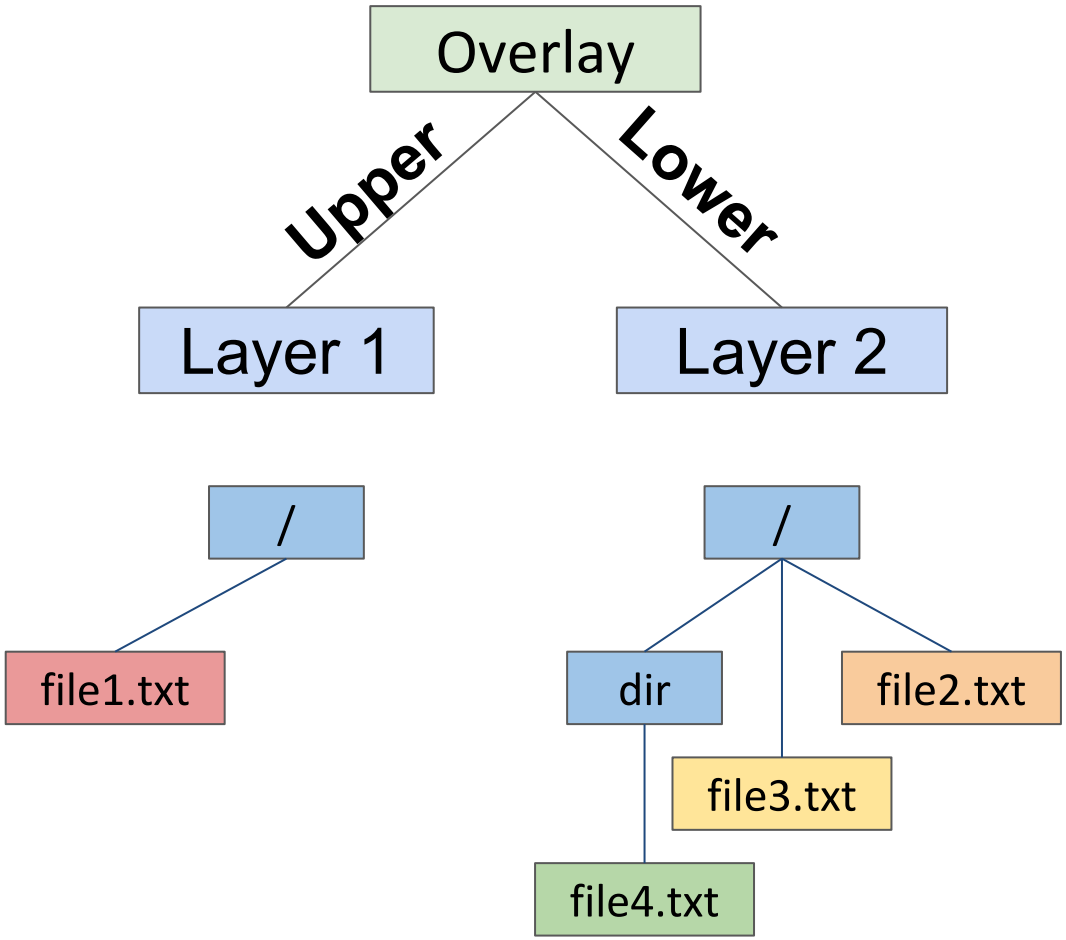
Underlined means success and bold means file found

File writes can only occur in the upper FS, thus file contents in lower FS must be "copied up"



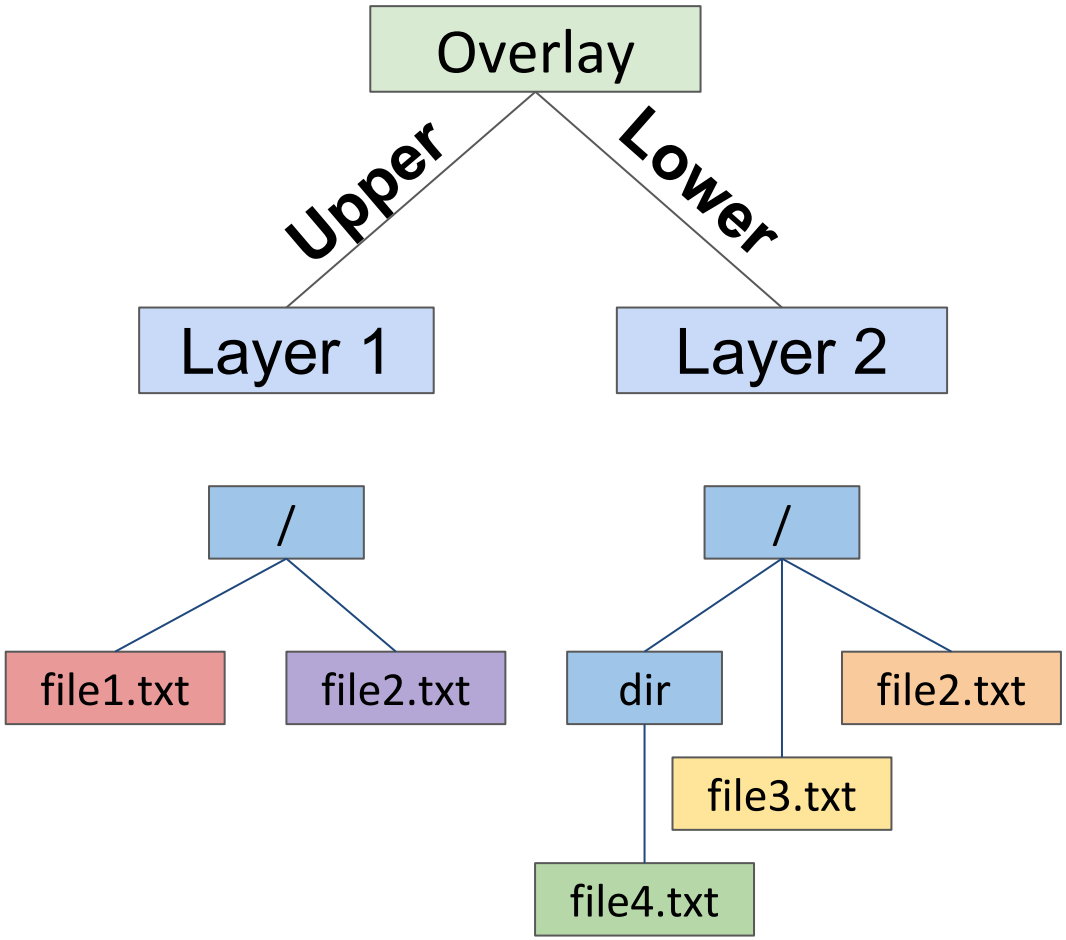
1) Write to /file2.txt

File writes can only occur in the upper FS, thus file contents in lower FS must be "copied up"



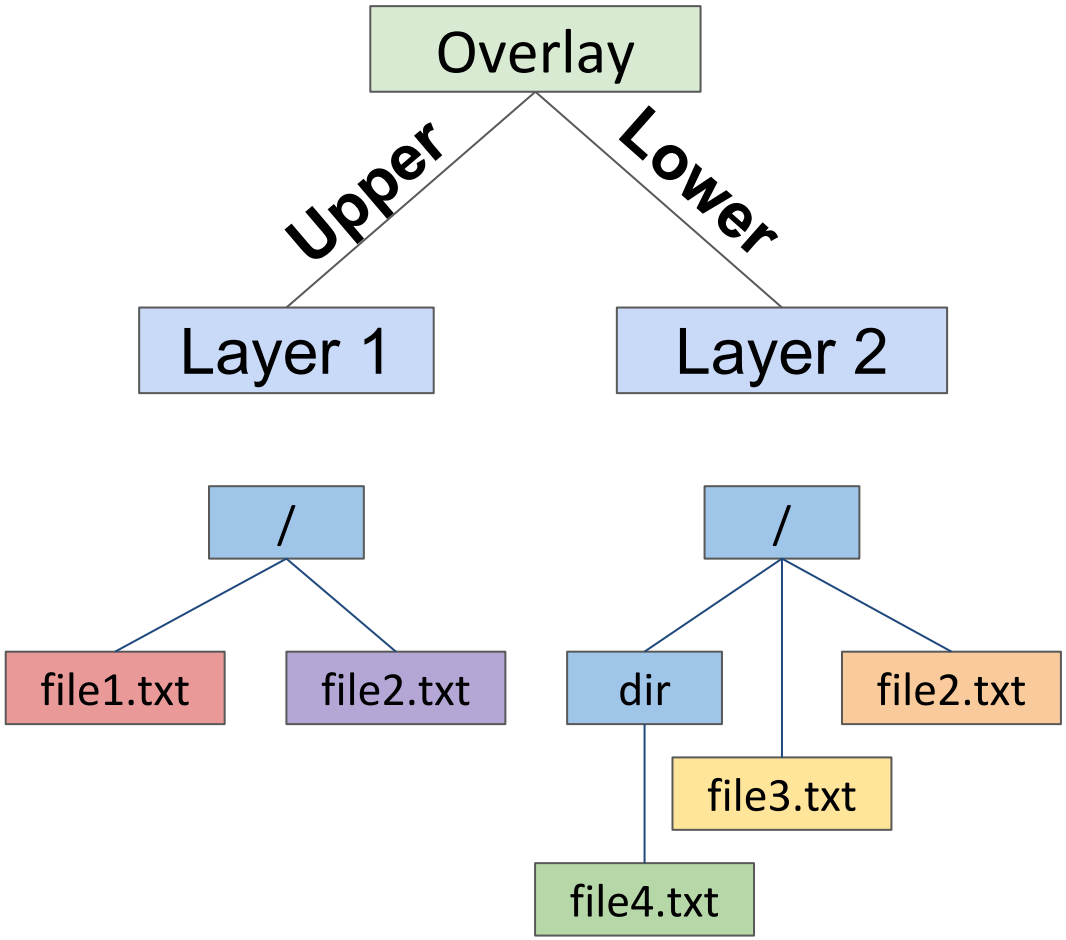
1) Write to /file2.txt
2) Resolve path

File writes can only occur in the upper FS, thus file contents in lower FS must be "copied up"



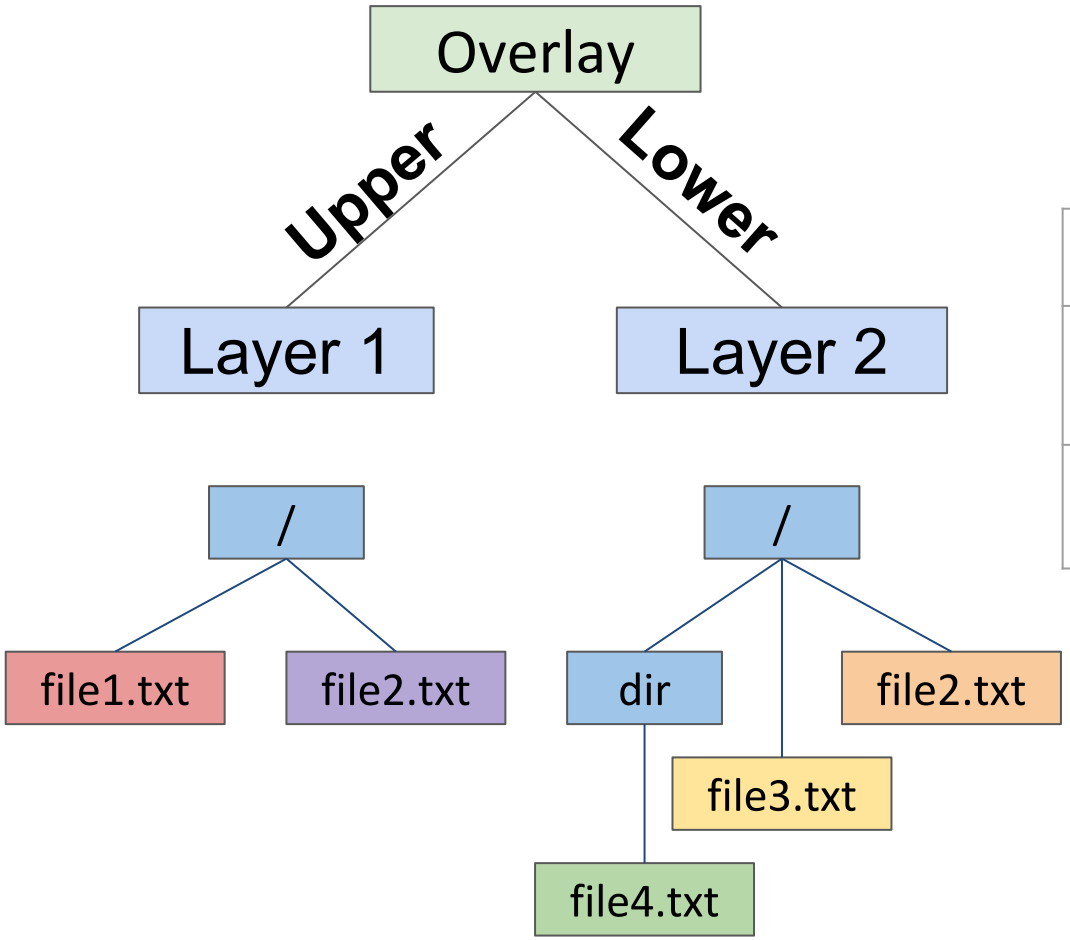
1) Write to /file2.txt
2) Resolve path
3) Copy up

File writes can only occur in the upper FS, thus file contents in lower FS must be "copied up"



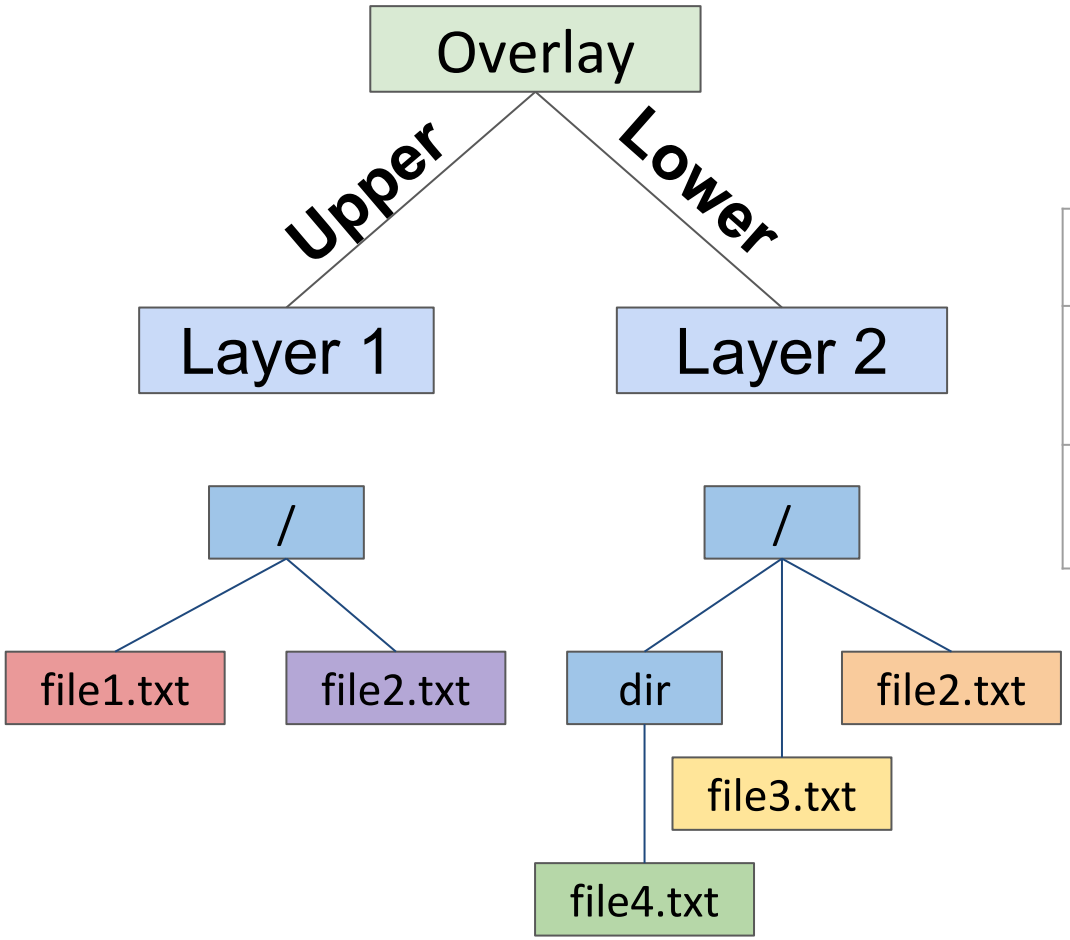
1) Write to /file2.txt
2) Resolve path
3) Copy up
4) Write to /file2.txt in upper

If a file exists in both upper (initially empty) and lower, we must retrieve the inode number from the lower



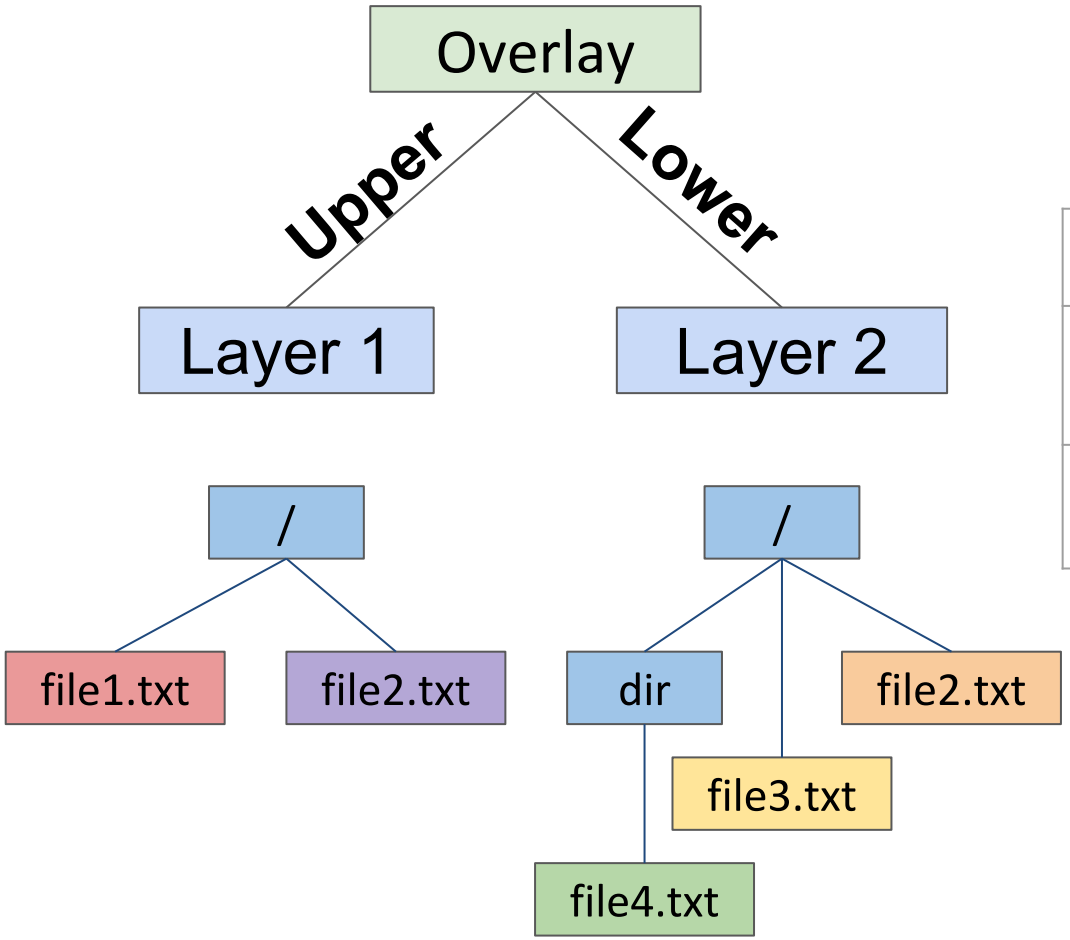
Layer	/	file2.txt
Layer 1		
Layer 2		

If a file exists in both upper (initially empty) and lower, we must retrieve the inode number from the lower



Layer	/	file2.txt
Layer 1	**1**	Used
Layer 2	**2**	

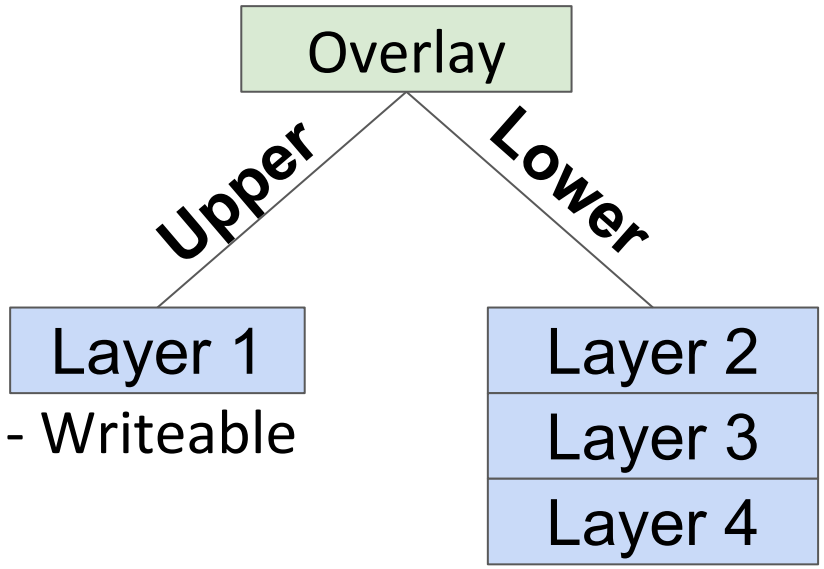
If a file exists in both upper (initially empty) and lower, we must retrieve the inode number from the lower



Layer	/	file2.txt
Layer 1	**1**	Used
Layer 2	**2**	

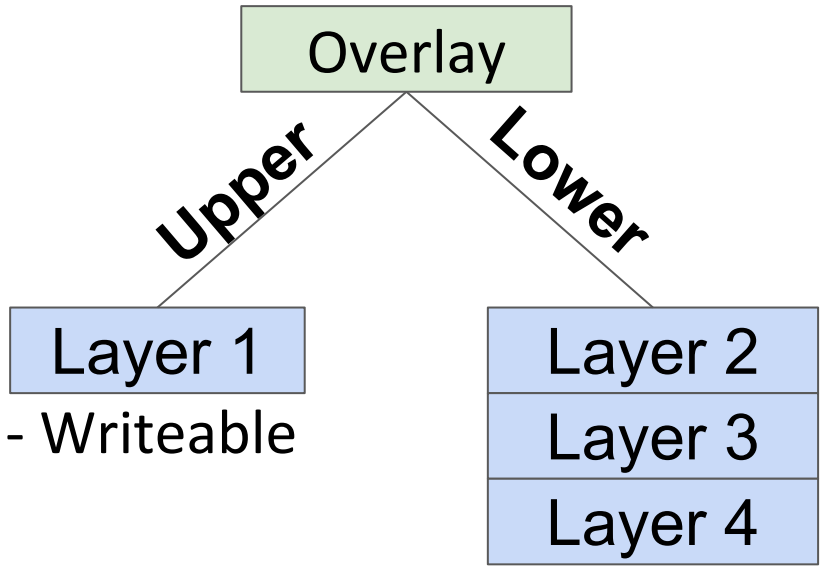
Retrieves inode number from lower because upper writeable

OverlayFS allows for multiple read-only lower layers, which are traversed from top to bottom



Layer	/	dir/	file4.txt
Layer 1			
Layer 2			
Layer 3			
Layer 4			

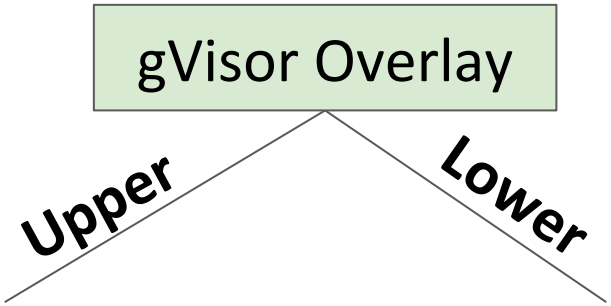
OverlayFS allows for multiple read-only lower layers, which are traversed from top to bottom



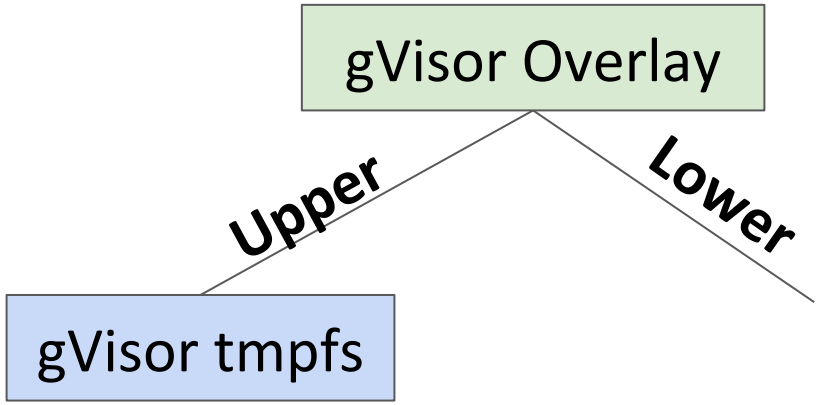
Layer	/	dir/	file4.txt
Layer 1	1	5	
Layer 2	2	6	
Layer 3	3	7	
Layer 4	4	**8**	Used

gVisor uses a combination of an internal Overlay and Linux' OverlayFS to access files in image layers

gVisor uses a combination of an internal Overlay and Linux' OverlayFS to access files in image layers

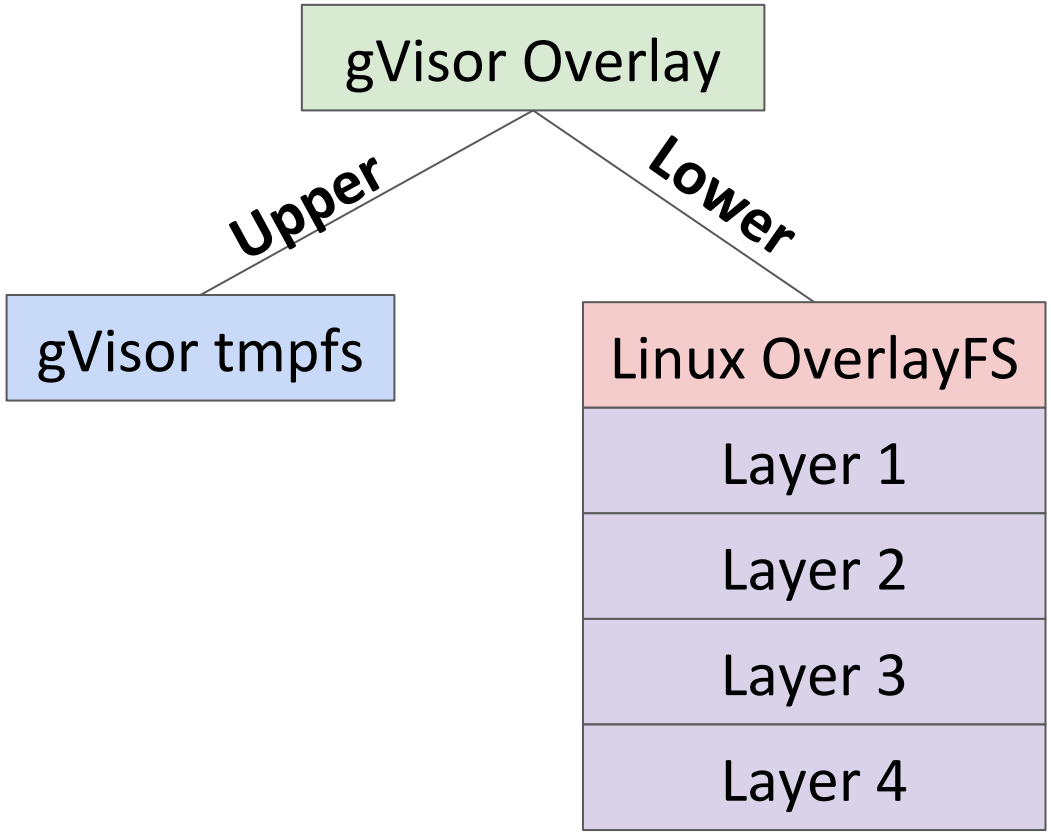


gVisor uses a combination of an internal Overlay and Linux' OverlayFS to access files in image layers



- Writeable
- Initially empty

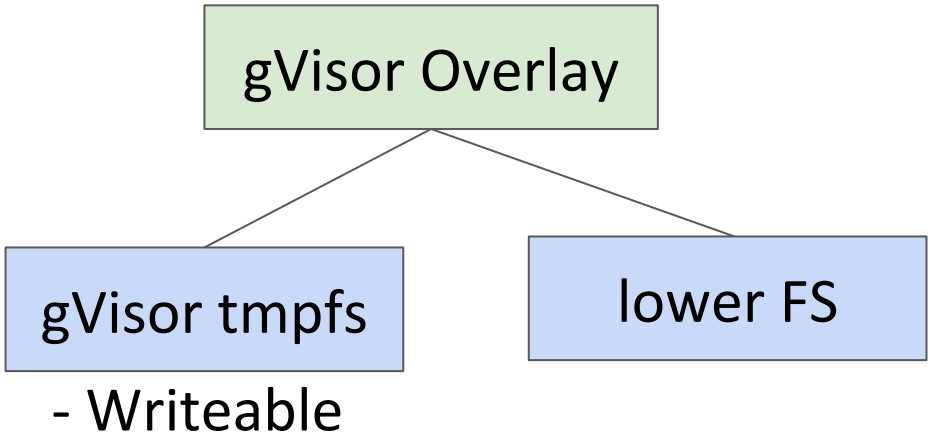
gVisor uses a combination of an internal Overlay and Linux' OverlayFS to access files in image layers

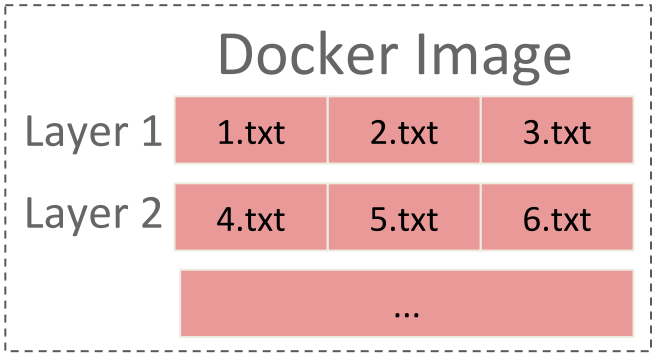


- Writeable
- Initially empty

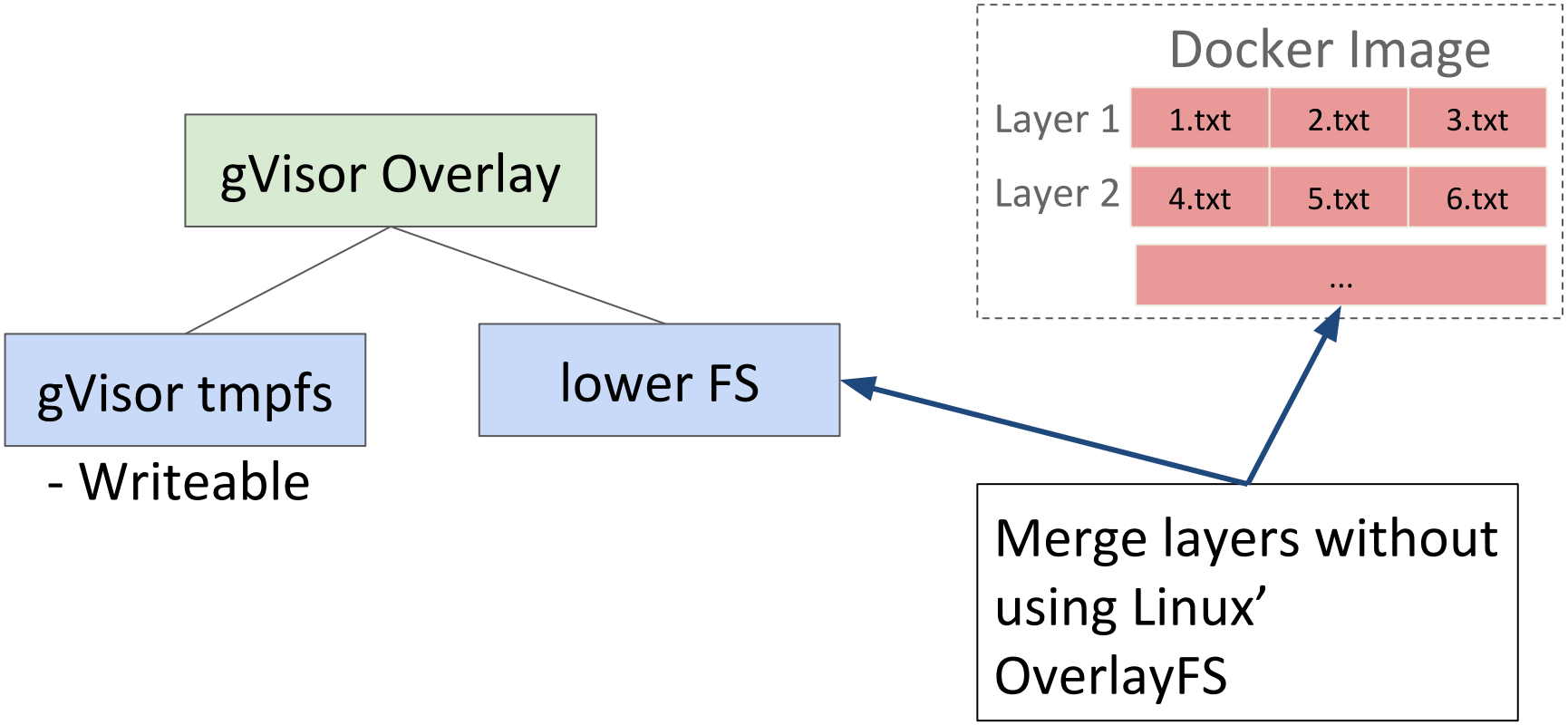
- Accessed via Gofer

In ContainerFS, Docker image layers are converted to image files and mounted within gVisor

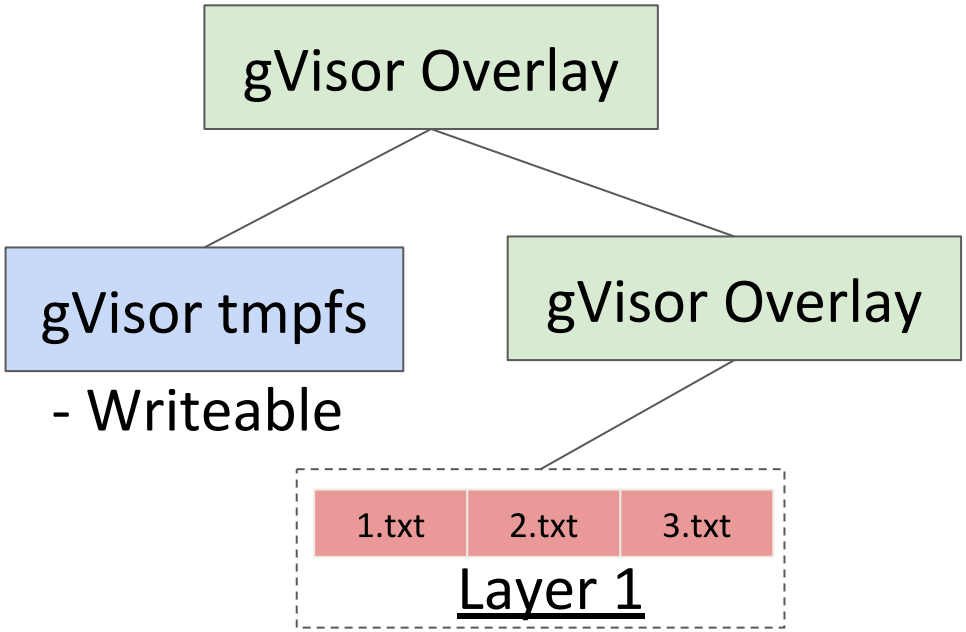


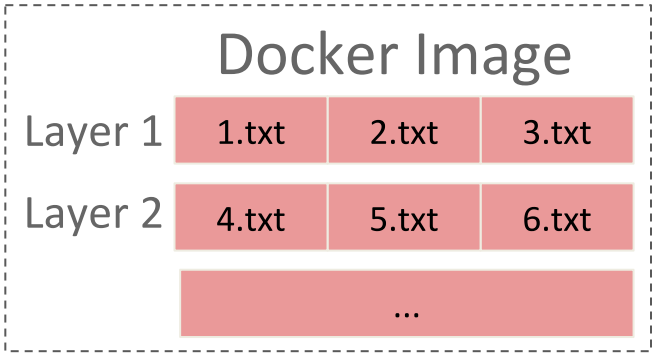


In ContainerFS, Docker image layers are converted to image files and mounted within gVisor

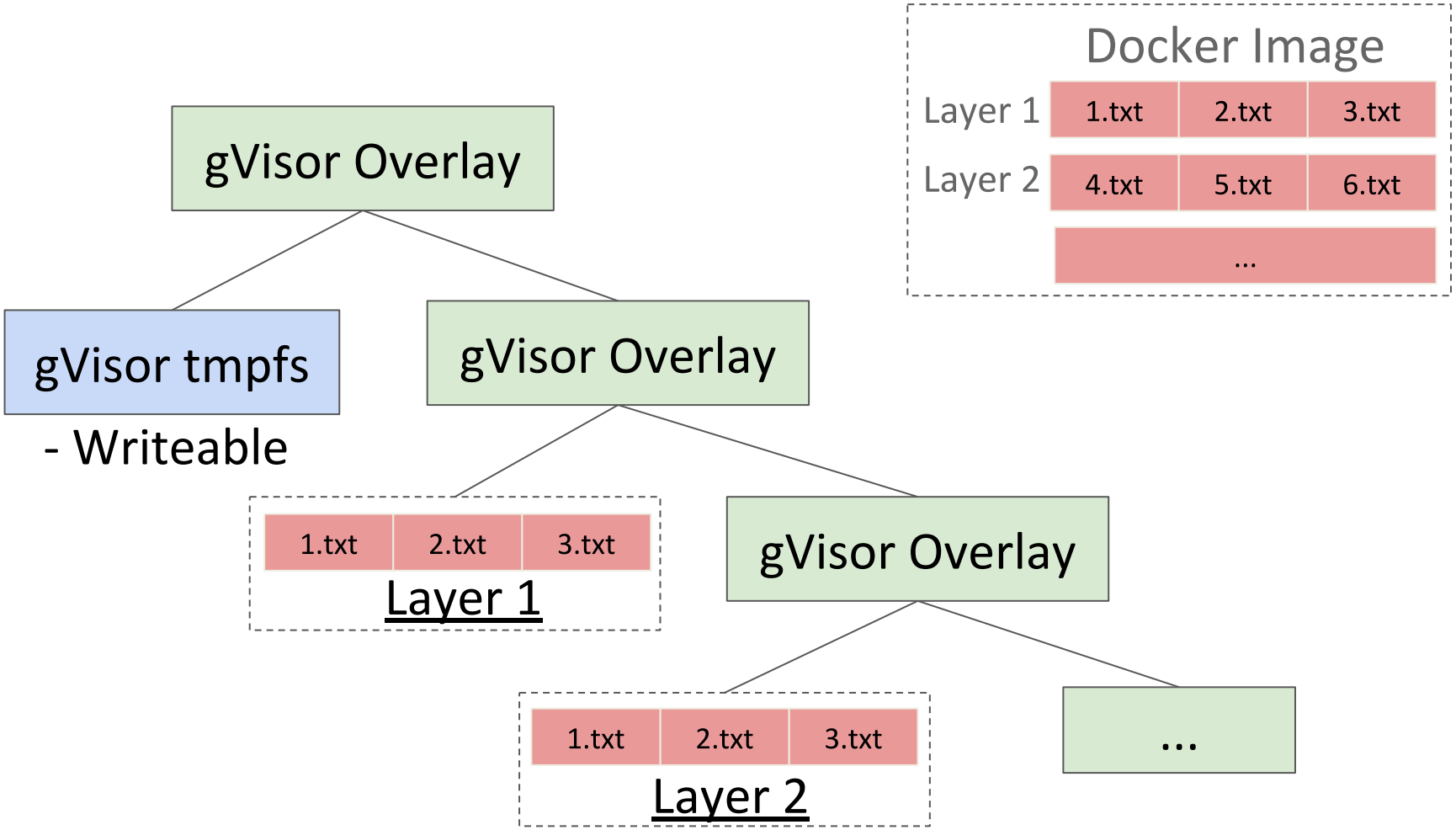


In ContainerFS, Docker image layers are converted to image files and mounted within gVisor





In ContainerFS, Docker image layers are converted to image files and mounted within gVisor



ContainerFS' Overlay tree is traversed identically to OverlayFS with multiple lower layers

Linux Overlay

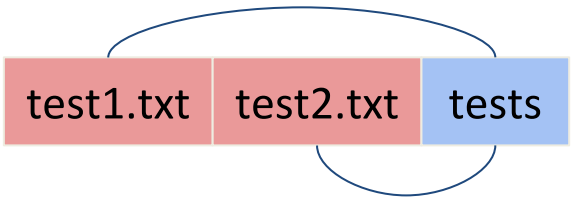
Layer	/	dir1	dir2	file.txt
img8	1	10	19	
img7	2	11	20	
img6	3	12	21	
img5	4	13	22	
img4	5	14	23	
img3	6	15	24	
img2	7	16	25	
img1	8	17	26	
img0	9	18	**27**	Used

→

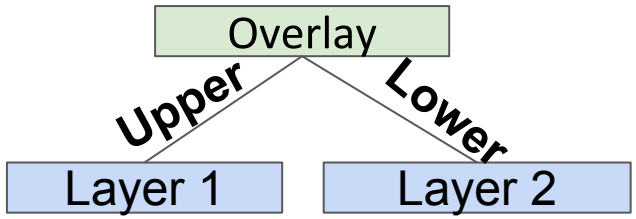
ContainerFS

Layer	/	dir1	dir2	file.txt
img8	1	10	19	
img7	2	11	20	
img6	3	12	21	
img5	4	13	22	
img4	5	14	23	
img3	6	15	24	
img2	7	16	25	
img1	8	17	26	
img0	9	18	**27**	Used

The second half of this talk explores the architecture of ImgFS, a gVisor file system for performance



ImgFS Architecture

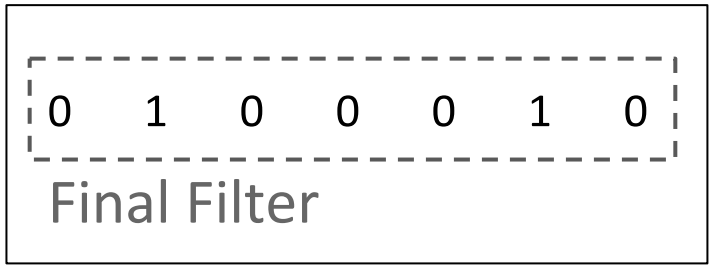


Mounting ImgFS Layers

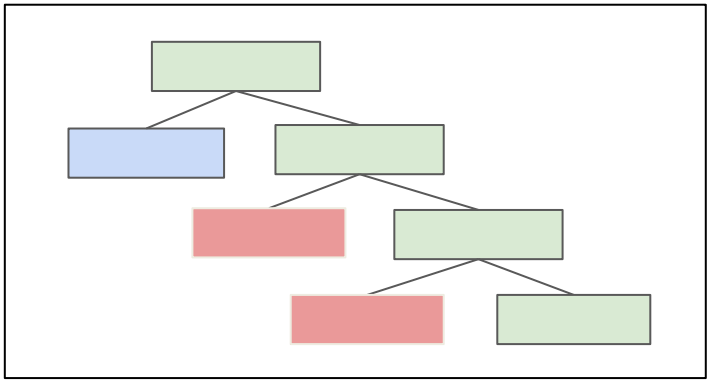
Layer	/	file2.txt
Layer 1	1	Used
Layer 2	2	

ContainerFS Traversal Algorithms

ContainerFS traverses the layers using two methods: Bloom Filter Traversal and Horizontal Traversal

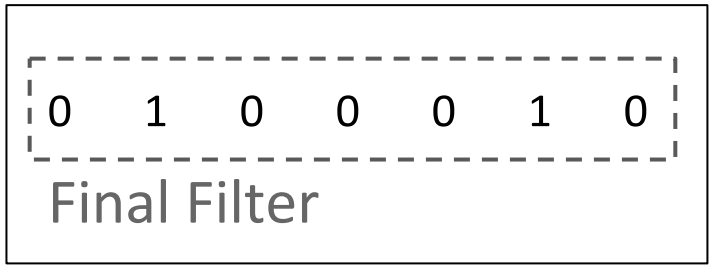


Bloom Filter Traversal

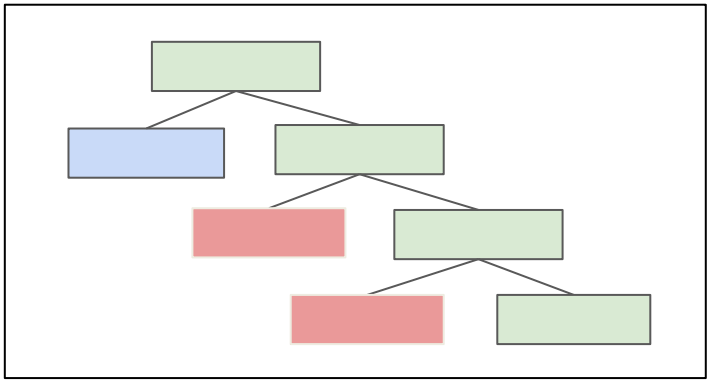


Horizontal Traversal

ContainerFS traverses the layers using two methods: Bloom Filter Traversal and Horizontal Traversal



Bloom Filter Traversal



Horizontal Traversal

A Bloom Filter is a probabilistic method for testing if an element definitely does not exist in a set

<u>Example</u>: Add element “test”

0	0	0	0	0	0	0

Initial Filter

A Bloom Filter is a probabilistic method for testing if an element definitely does not exist in a set

<u>Example</u>: Add element “test”

Set one bit from each hash

hash(“test”)
hash(“test”)
…

0	0	0	0	0	0	0

Initial Filter

0	1	0	0	0	1	0

Final Filter

A Bloom Filter is a probabilistic method for testing if an element definitely does not exist in a set

<u>Example</u>: Add element "test"

Set one bit from each hash

hash("test")
hash("test")
...

0	0	0	0	0	0	0

Initial Filter

0	1	0	0	0	1	0

Final Filter

<u>Example</u>: Test element <u>"init"</u>

0	0	0	0	0	0	0

Initial Filter

A Bloom Filter is a probabilistic method for testing if an element definitely does not exist in a set

Example: Add element "test"

Set one bit from each hash

hash("test")
hash("test")
...

0	0	0	0	0	0	0

Initial Filter

0	1	0	0	0	1	0

Final Filter

Example: Test element "init"

hash("init")
hash("init")
...

0	0	0	0	0	0	0

Initial Filter

0	1	0	1	0	0	0

Final Filter

A Bloom Filter is a probabilistic method for testing if an element definitely does not exist in a set

<u>Example</u>: Add element "test"

Set one bit from each hash

hash("test")
hash("test")
...

0 0 0 0 0 0 0

Initial Filter

0 1 0 0 0 1 0

Final Filter

Not matching, so definitely not in filter

<u>Example</u>: Test element "init"

hash("init")
hash("init")
...

0 0 0 0 0 0 0

Initial Filter

0 1 0 1 0 0 0

Final Filter

A Bloom Filter is a probabilistic method for testing if an element definitely does not exist in a set

<u>Example</u>: Add element “test”

Set one bit from each hash

hash(“test”)
hash(“test”)
…

0	0	0	0	0	0	0

Initial Filter

0	1	0	0	0	1	0

Final Filter

<u>Example</u>: Test element <u>“file”</u>

0	0	0	0	0	0	0

Initial Filter

A Bloom Filter is a probabilistic method for testing if an element definitely does not exist in a set

Example: Add element "test"

Set one bit from each hash

hash("test")
hash("test")
...

0	0	0	0	0	0	0

Initial Filter

0	1	0	0	0	1	0

Final Filter

Example: Test element "file"

hash("file")
hash("file")
...

0	0	0	0	0	0	0

Initial Filter

0	1	0	0	0	1	0

Final Filter

A Bloom Filter is a probabilistic method for testing if an element definitely does not exist in a set

<u>Example</u>: Add element "test"

Set one bit from each hash

hash("test")
hash("test")
...

0 0 0 0 0 0 0

Initial Filter

0 1 0 0 0 1 0

Final Filter

Matching, so maybe element in filter

<u>Example</u>: Test element <u>"file"</u>

hash("file")
hash("file")
...

0 0 0 0 0 0 0

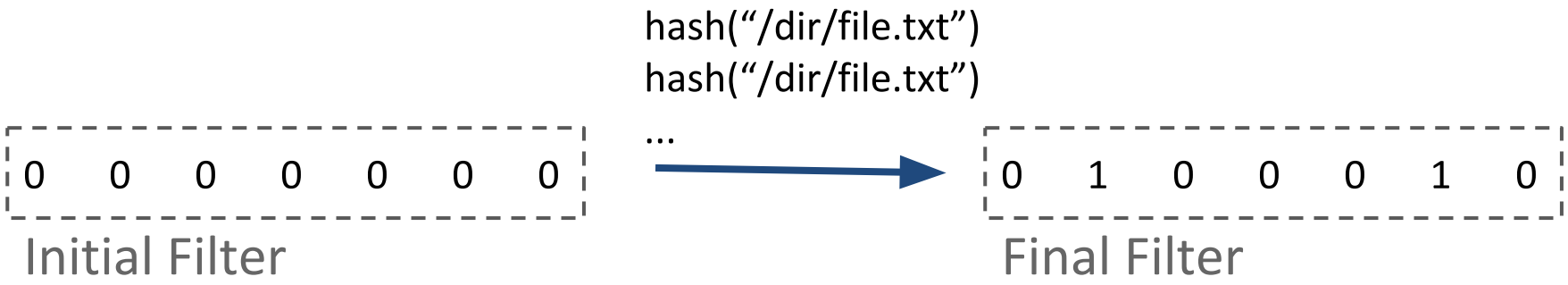
Initial Filter

0 1 0 0 0 1 0

Final Filter

Bloom Filters can be used to determine if a path exists in a layer

<u>Example</u>: Add element “/dir/file.txt”



Bloom Filters can be used to determine if a path exists in a layer

Example: Add element "/dir/file.txt"

hash("/dir/file.txt")
hash("/dir/file.txt")
...

0	0	0	0	0	0	0

Initial Filter

0	1	0	0	0	1	0

Final Filter

Example: Test element "/file.txt"

hash("/file.txt")
hash("/file.txt")
...

0	0	0	0	0	0	0

Initial Filter

0	1	0	1	0	0	0

Final Filter

Bloom Filters can be used to determine if a path exists in a layer

Example: Add element "/dir/file.txt"

hash("/dir/file.txt")
hash("/dir/file.txt")
...

0 0 0 0 0 0 0 → 0 1 0 0 0 1 0

Initial Filter Final Filter

Not matching, so path not in filter

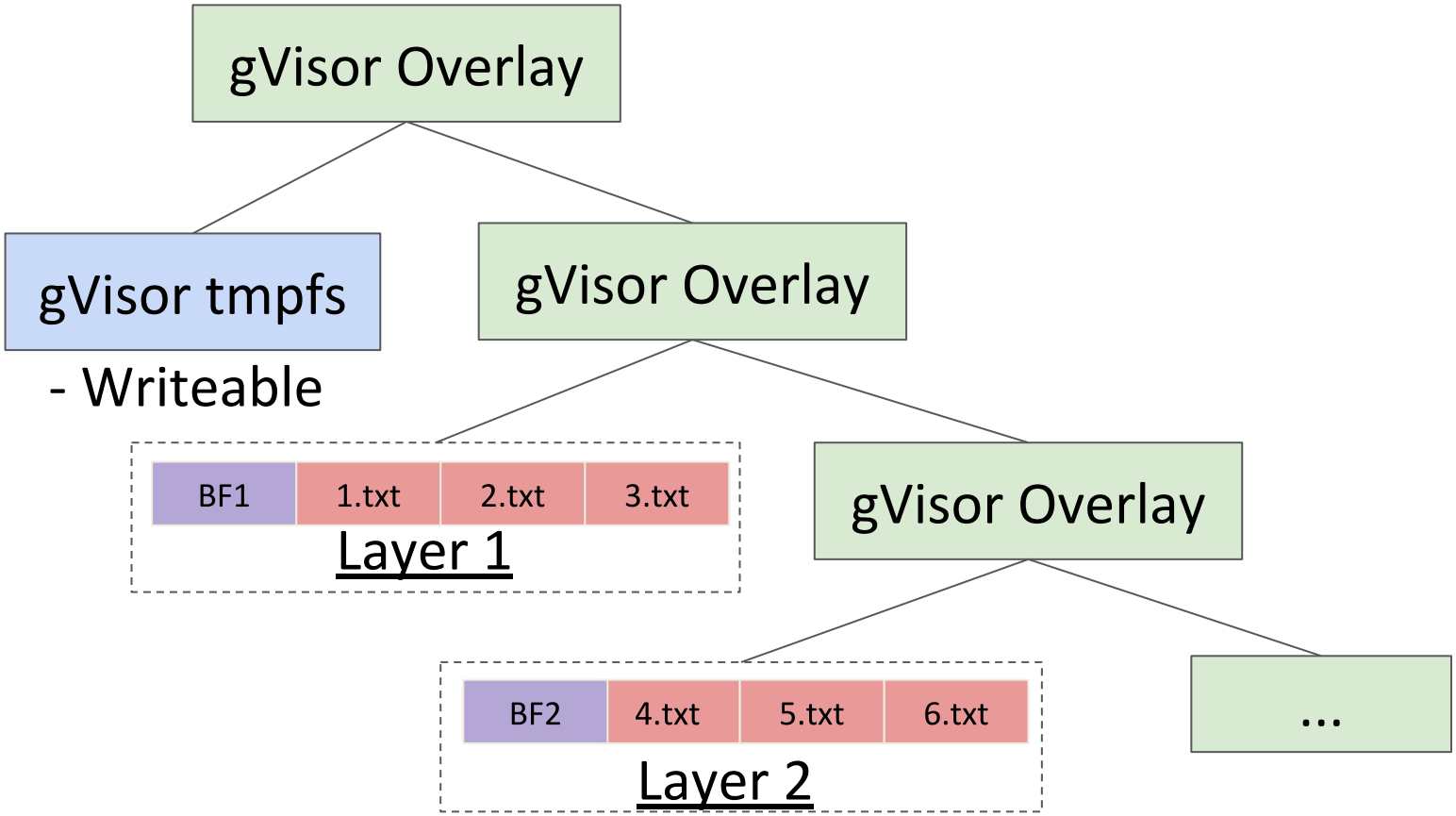
Example: Test element "/file.txt"

hash("/file.txt")
hash("/file.txt")
...

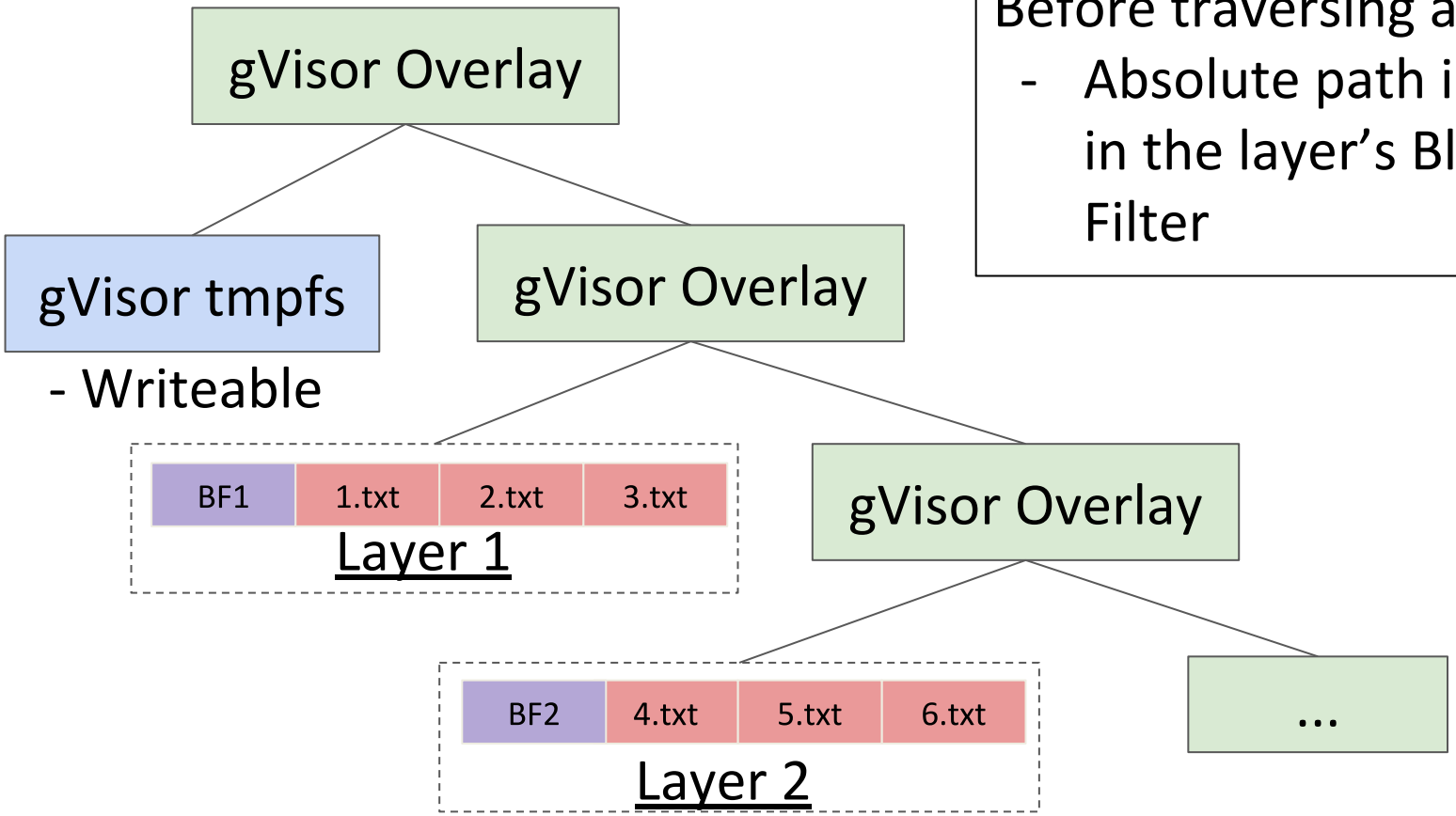
0 0 0 0 0 0 0 → 0 1 0 1 0 0 0

Initial Filter Final Filter

ContainerFS uses Bloom Filters preloaded with the absolute paths of all files and directories in layer



ContainerFS uses Bloom Filters preloaded with the absolute paths of all files and directories in layer



Before traversing a layer…
- Absolute path is tested in the layer's Bloom Filter

Bloom Filters eliminate unnecessary lookups in high layers

Without BF

Layer	/	dir1	dir2	file.txt
img8	1	10	19	
img7	2	11	20	
img6	3	12	21	
img5	4	13	22	
img4	5	14	23	
img3	6	15	24	
img2	7	16	25	
img1	8	17	26	
img0	9	18	**27**	Used

Bloom Filters eliminate unnecessary lookups in high layers

Without BF

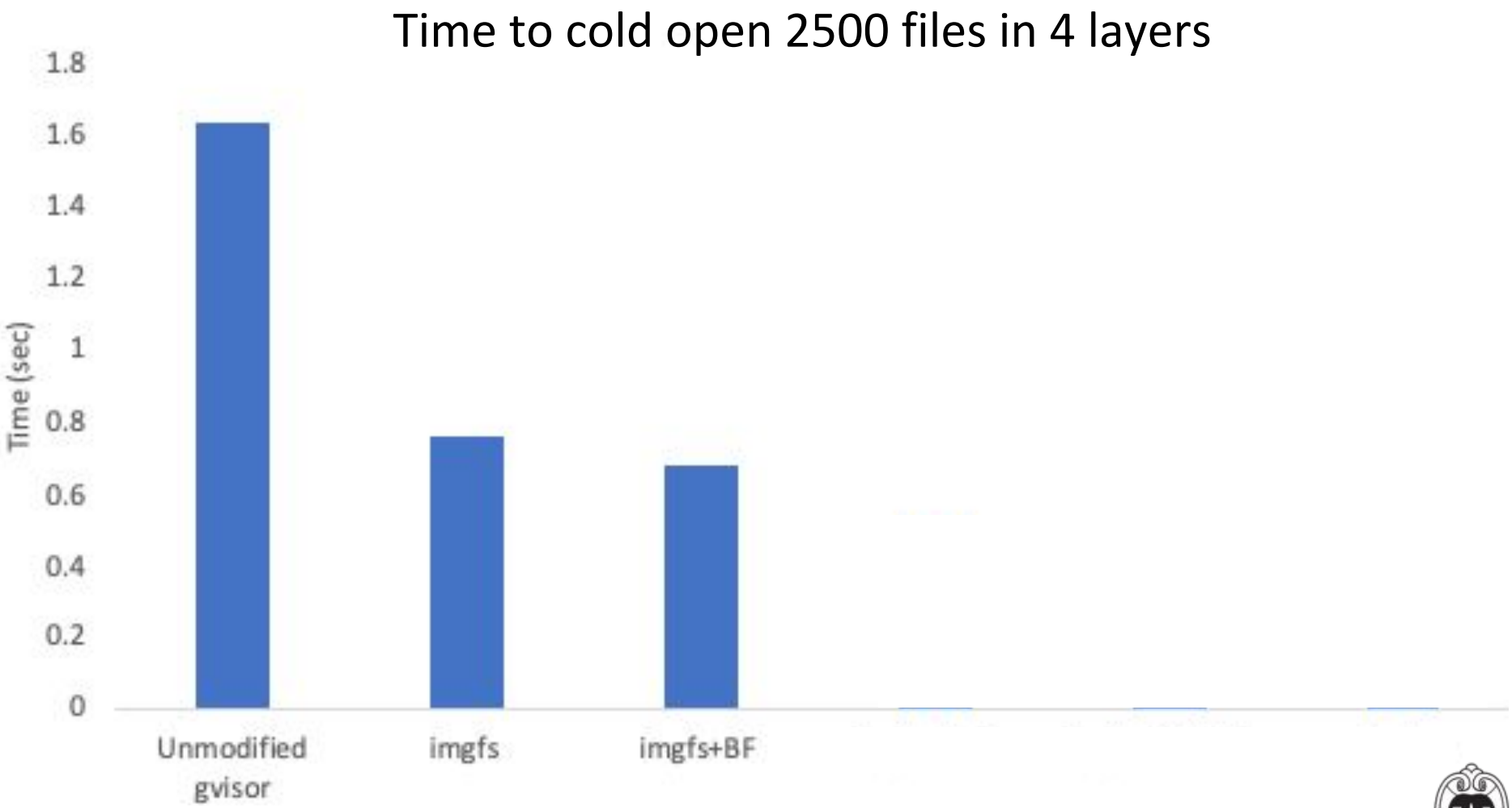
Layer	/	dir1	dir2	file.txt
img8	1	10	19	
img7	2	11	20	
img6	3	12	21	
img5	4	13	22	
img4	5	14	23	
img3	6	15	24	
img2	7	16	25	
img1	8	17	26	
img0	9	18	**27**	Used

→

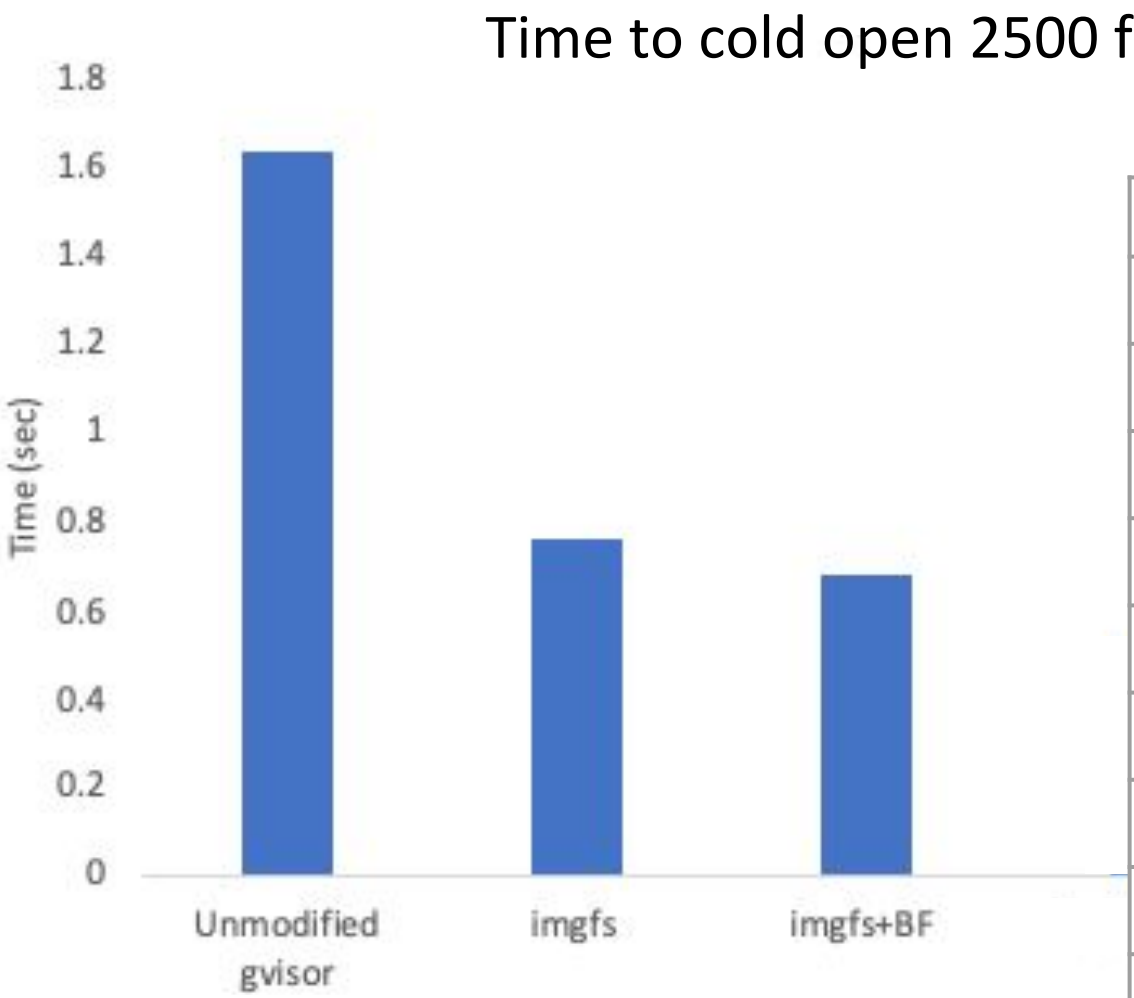
With BF

Layer	BF	/	dir1	dir2	file.txt
img8	1				
img7	2				
img6	3				
img5	4				
img4	5				
img3	6				
img2	7				
img1	8				
img0	9	11	12	**13**	Used

Bloom Filters increase ContainerFS' performance, but it is costly to check each layer for each path

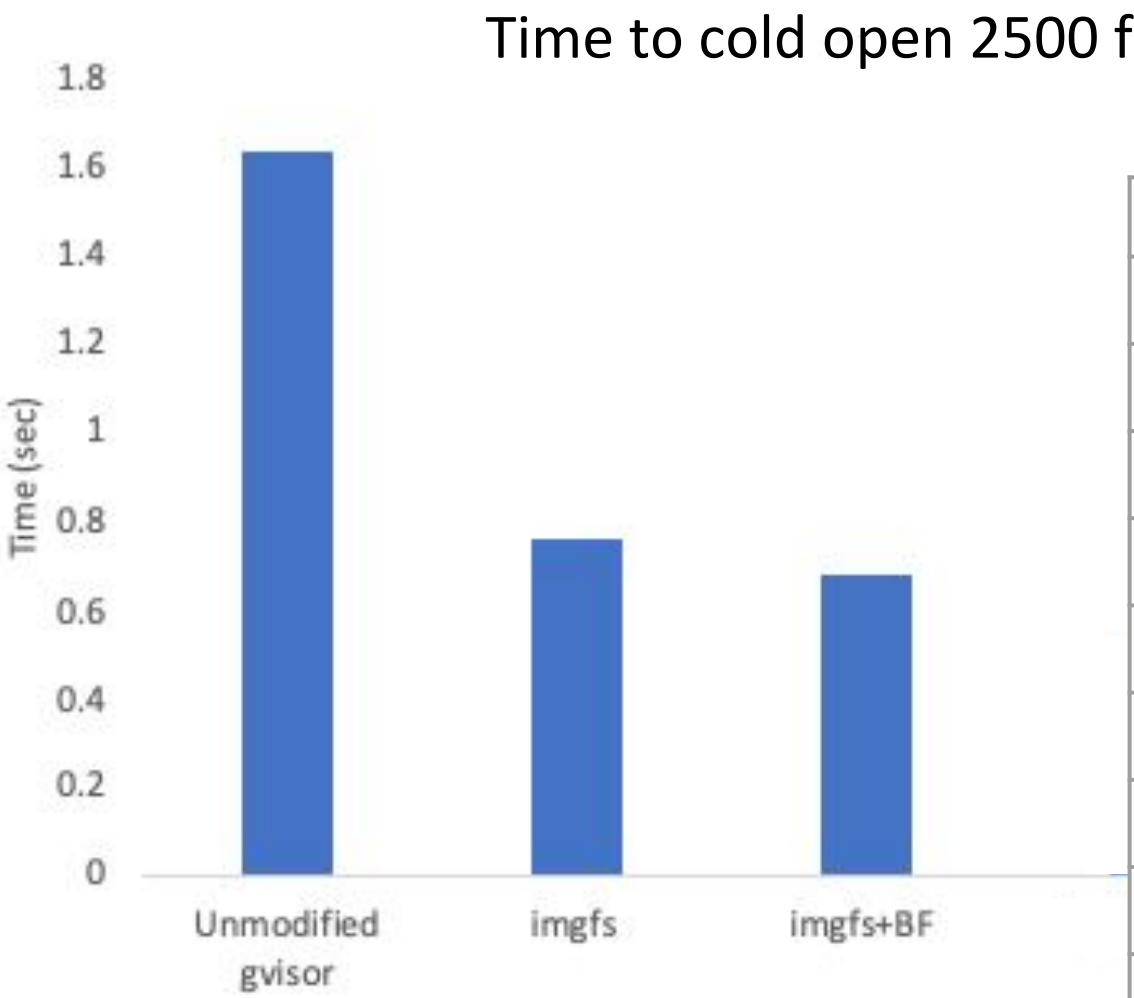


Bloom Filters increase ContainerFS' performance, but it is costly to check each layer for each path



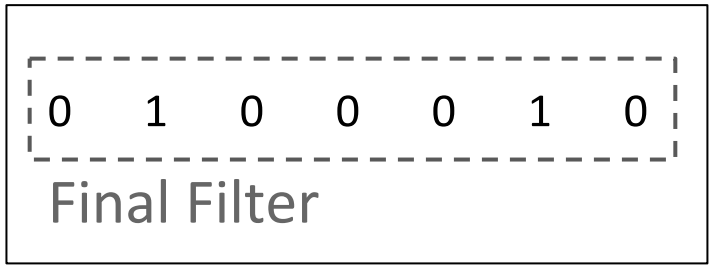
Layer	BF	/	dir1	dir2	file.txt
img8	1	11	12	**13**	Used
img7	2				
img6	3				
img5	4				
img4	5				
img3	6				
img2	7				
img1	8				
img0	9				

Bloom Filters increase ContainerFS' performance, but it is costly to check each layer for each path

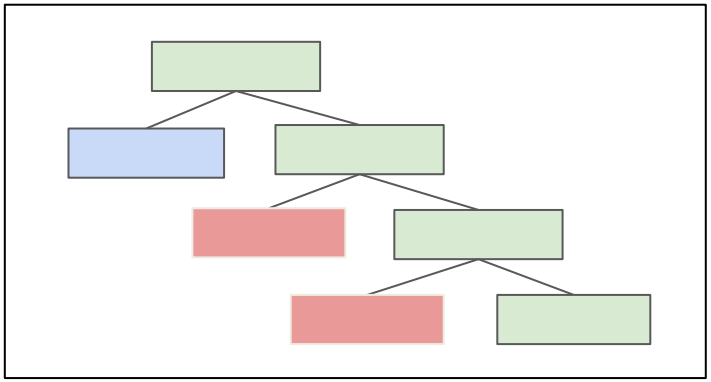


Layer	BF	/	dir1	dir2	file.txt
img8	1	11	12	**13**	Used
img7	2				
img6	3				
img5	4				
img4	5				
img3	6				
img2	7				
img1	8				
img0	9				

ContainerFS traverses the layers using two methods: Bloom Filter Traversal and Horizontal Traversal

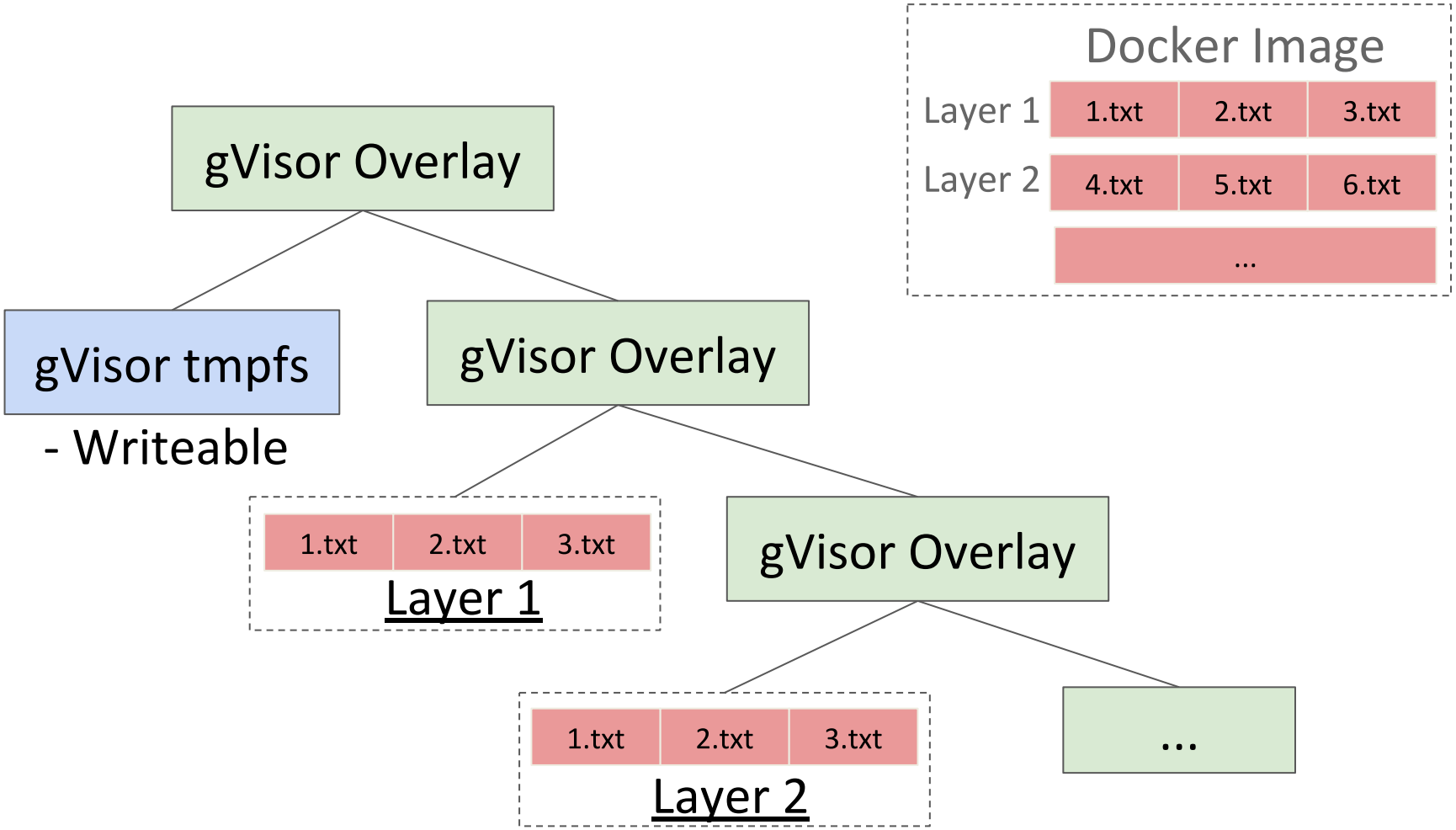


Bloom Filter Traversal



Horizontal Traversal

Horizontal Traversal: ContainerFS completely traverses a layer before moving to a lower layer



Horizontal Traversal eliminates unnecessary lookups in low layers

Without Horiz

Layer	/	dir1	dir2	file.txt
img8	1	10	**19**	Used
img7	2	11		
img6	3	12		
img5	4	13		
img4	5	14		
img3	6	15		
img2	7	16		
img1	8	17		
img0	9	18		

Horizontal Traversal eliminates unnecessary lookups in low layers

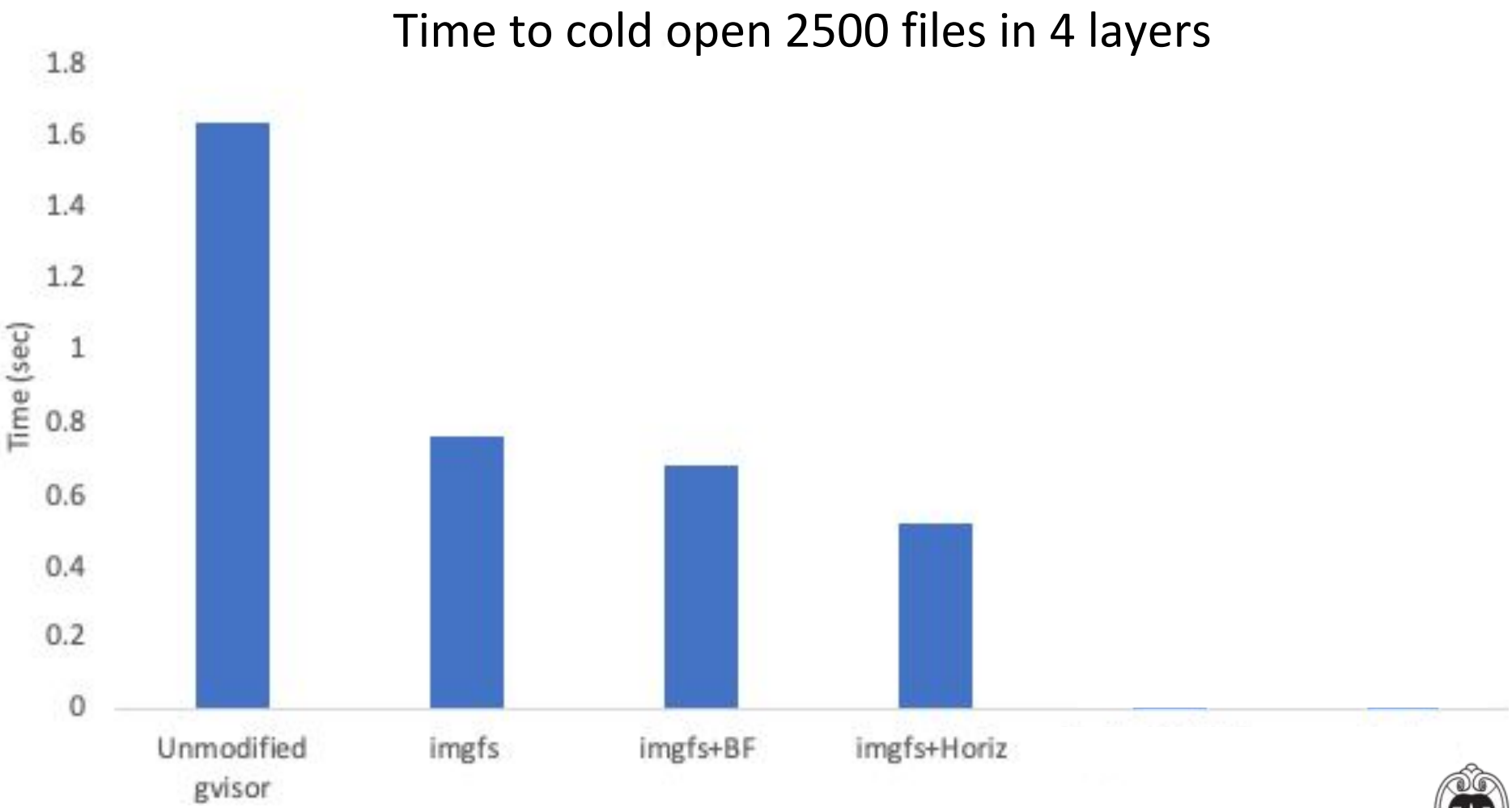
Without Horiz

Layer	/	dir1	dir2	file.txt
img8	1	10	**19**	Used
img7	2	11		
img6	3	12		
img5	4	13		
img4	5	14		
img3	6	15		
img2	7	16		
img1	8	17		
img0	9	18		

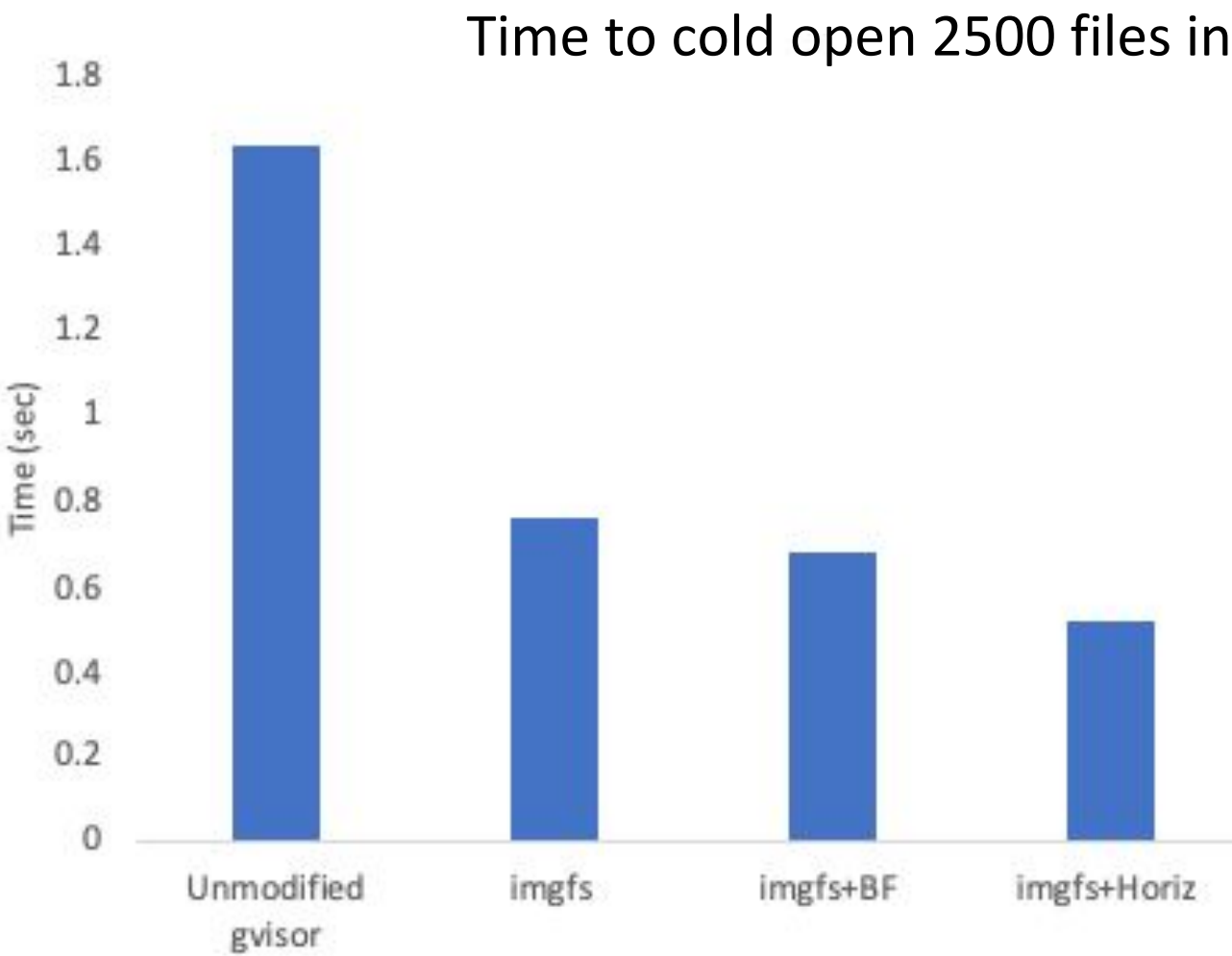
With Horiz

Layer	/	dir1	dir2	file.txt
img8	1	2	**3**	Used
img7				
img6				
img5				
img4				
img3				
img2				
img1				
img0				

Horizontal Traversal increases ContainerFS' cold open performance, but costly to check upper layers still



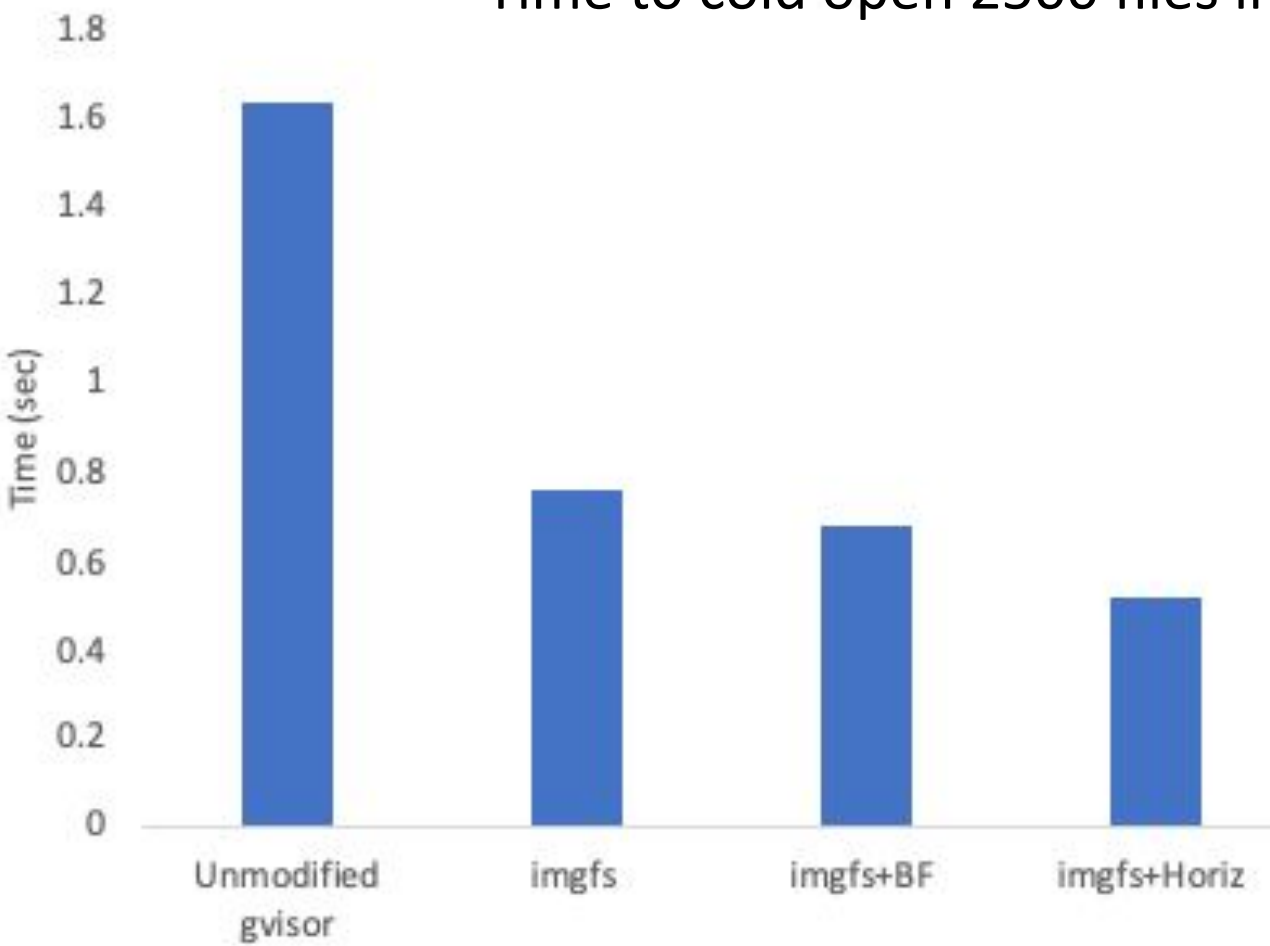
Horizontal Traversal increases ContainerFS' cold open performance, but costly to check upper layers still



Layer	/	dir1	dir2	file.txt
img8	1	5	9	
img7	2	6	10	
img6	3	7	11	
img5	4	8	**12**	Used
img4				
img3				
img2				
img1				
img0				

Horizontal Traversal increases ContainerFS' cold open performance, but costly to check upper layers still

Time to cold open 2500 files in 4 layers



Layer	/	dir1	dir2	file.txt
img8	1	5	9	
img7	2	6	10	
img6	3	7	11	
img5	4	8	**12**	Used
img4				
img3				
img2				
img1				
img0				

Combining Horizontal Traversal and a Bloom Filter per layer eliminates lookups at high and low layers

Without Either

Layer	/	dir1	dir2	file.txt
img8	1	10	19	
img7	2	11	20	
img6	3	12	21	
img5	4	13	22	
img4	5	14	**23**	Used
img3	6	15		
img2	7	16		
img1	8	17		
img0	9	18		

Combining Horizontal Traversal and a Bloom Filter per layer eliminates lookups at high and low layers

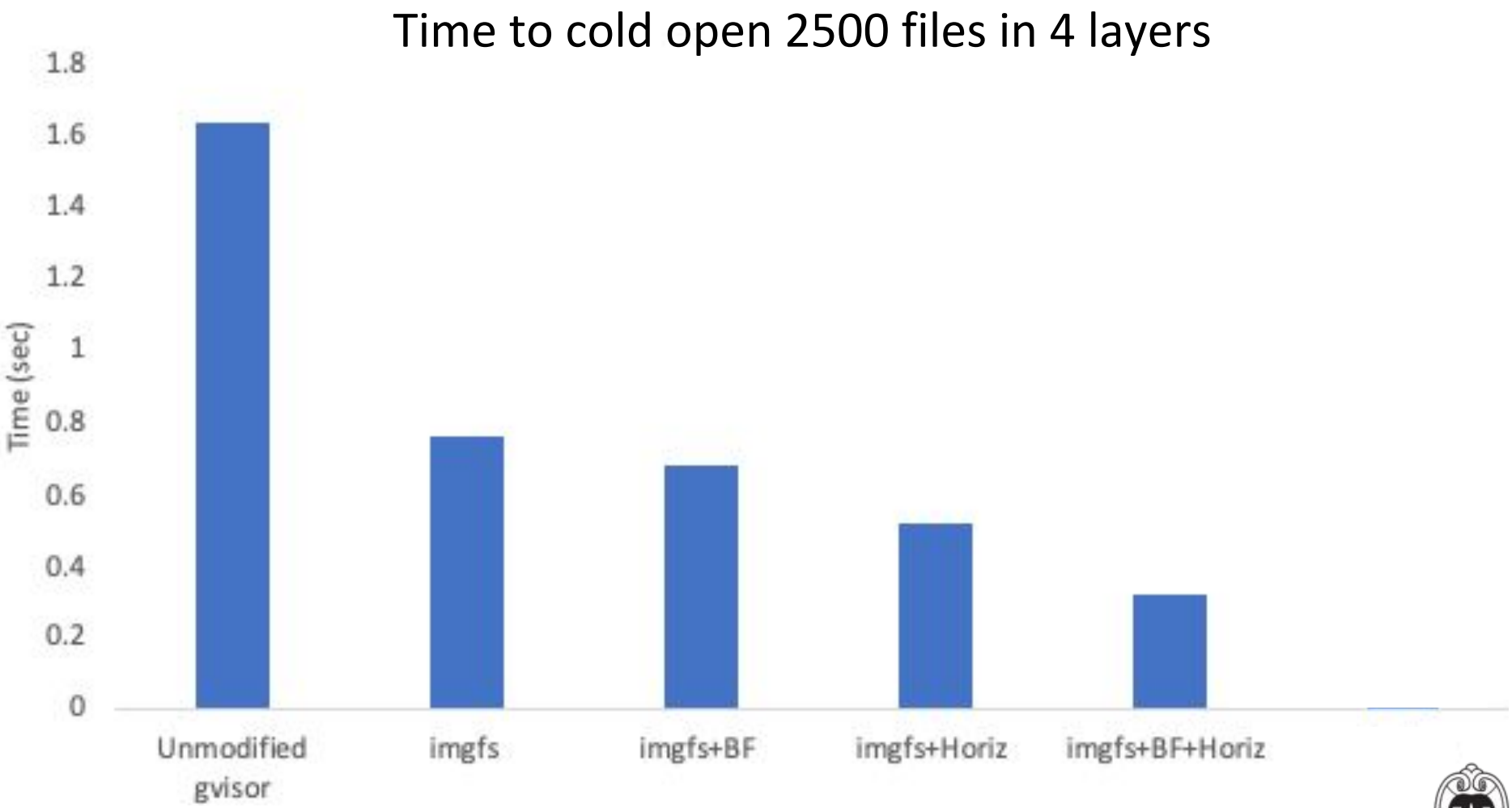
Without Either

Layer	/	dir1	dir2	file.txt
img8	1	10	19	
img7	2	11	20	
img6	3	12	21	
img5	4	13	22	
img4	5	14	**23**	Used
img3	6	15		
img2	7	16		
img1	8	17		
img0	9	18		

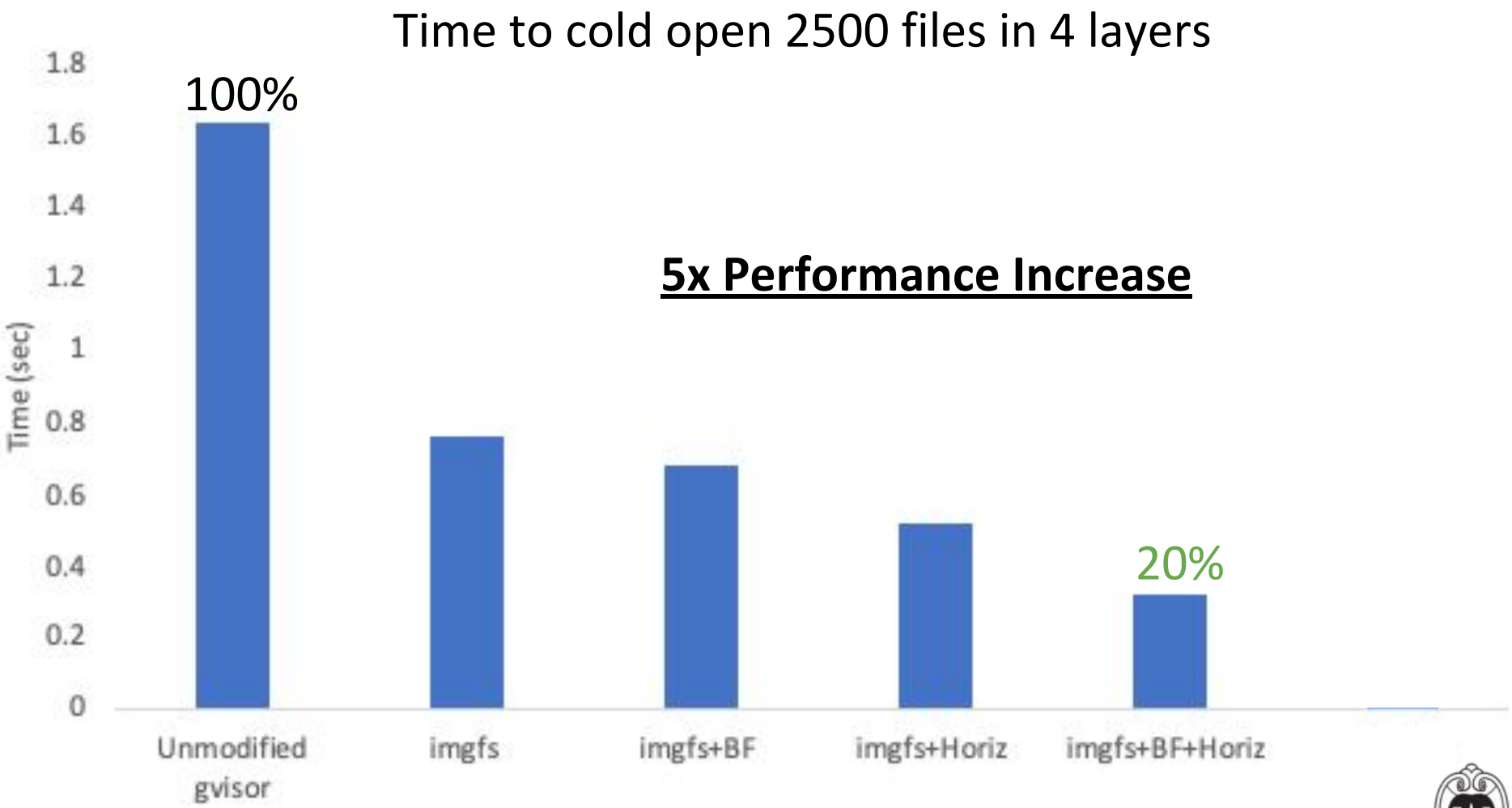
With Both

Layer	BF	/	dir1	dir2	file.txt
img8	1				
img7	2				
img6	3				
img5	4				
img4	5	6	7	**8**	Used
img3					
img2					
img1					
img0					

Combining Horizontal Traversal and a Bloom Filter per layer significantly increases performance



Combining Horizontal Traversal and a Bloom Filter per layer significantly increases performance



Related work includes Docker's build time merge algorithm and gVisor's future releases

Related work includes Docker's build time merge algorithm and gVisor's future releases

gVisor

gVisor will release a new VFS that delegates path resolution to the underlying FS

Related work includes Docker's build time merge algorithm and gVisor's future releases

gVisor will release a new VFS that delegates path resolution to the underlying FS

gVisor is releasing a ext{2,3,4} image reader for the Sentry

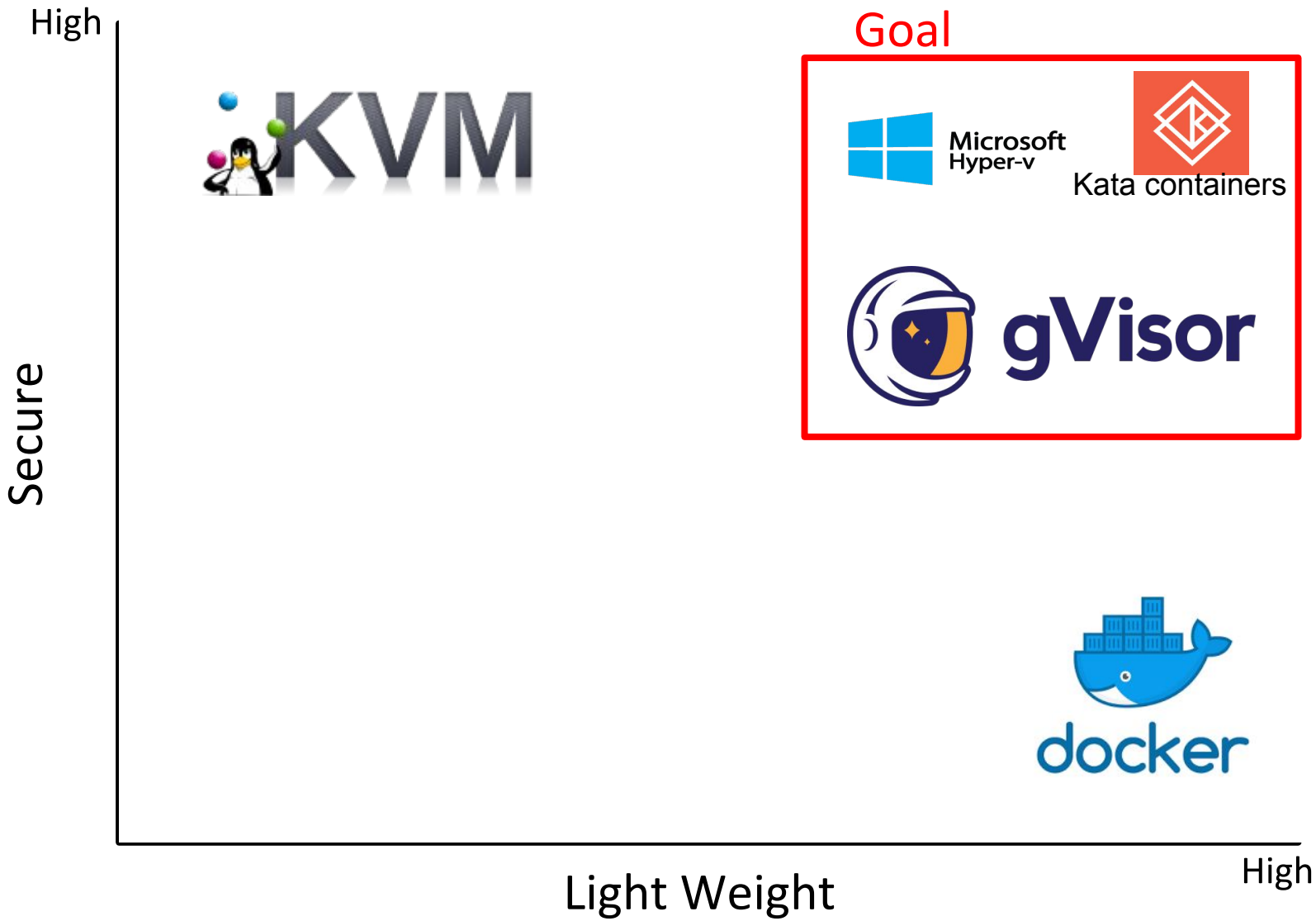
Related work includes Docker's build time merge algorithm and gVisor's future releases

gVisor will release a new VFS that delegates path resolution to the underlying FS

gVisor is releasing a ext{2,3,4} image reader for the Sentry

Docker recursively merges image layers into one overlay at build time

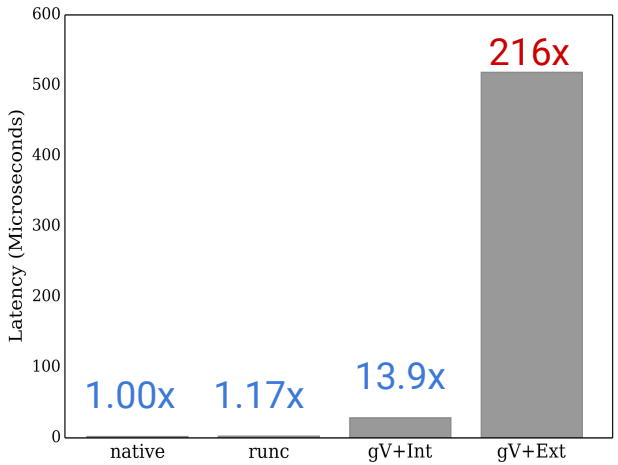
In summary, ContainerFS mitigates cold open file system latencies for gVisor



In summary, ContainerFS mitigates cold open file system latencies for gVisor

High

Secure



gVisor has high latency when operations utilize the gofer

Goal

Microsoft Hyper-v

Kata containers

gVisor

docker

High

Light Weight

In summary, ContainerFS mitigates cold open file system latencies for gVisor

High

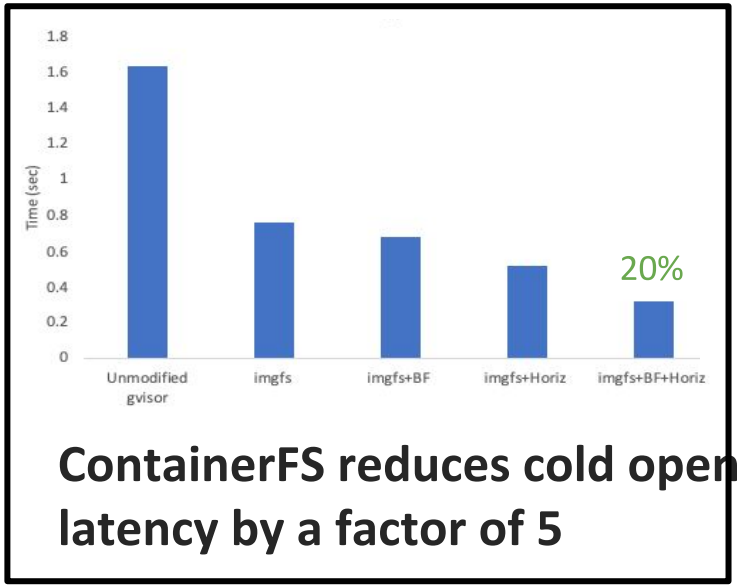
Secure

Goal

Microsoft Hyper-v

Kata containers

gVisor



ContainerFS reduces cold open latency by a factor of 5

docker

Light Weight

High

In summary, ContainerFS mitigates cold open file system latencies for gVisor

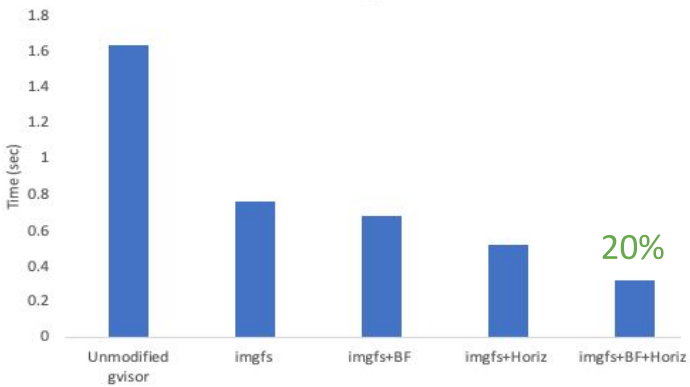
High

Secure

Goal

Microsoft Hyper-v

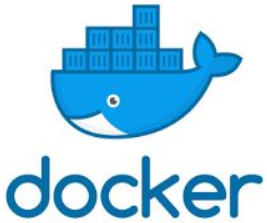
Kata containers



ContainerFS reduces cold open latency by a factor of 5

Questions?

Email:
eyoung8@wisc.edu



Light Weight

High