

6.1800 Spring 2024

Lecture #7: Performance + Other Concerns


performance, with a deep dive into storage and filesystems



Microsoft Says It Has Created a New State of Matter to Power Quantum Computers

Microsoft's new "topological qubit" is not based on a solid, liquid or gas. It is another phase of matter that many experts did not think was possible.

<https://www.nytimes.com/2025/02/19/technology/microsoft-quantum-computing-topological-qubit.html>

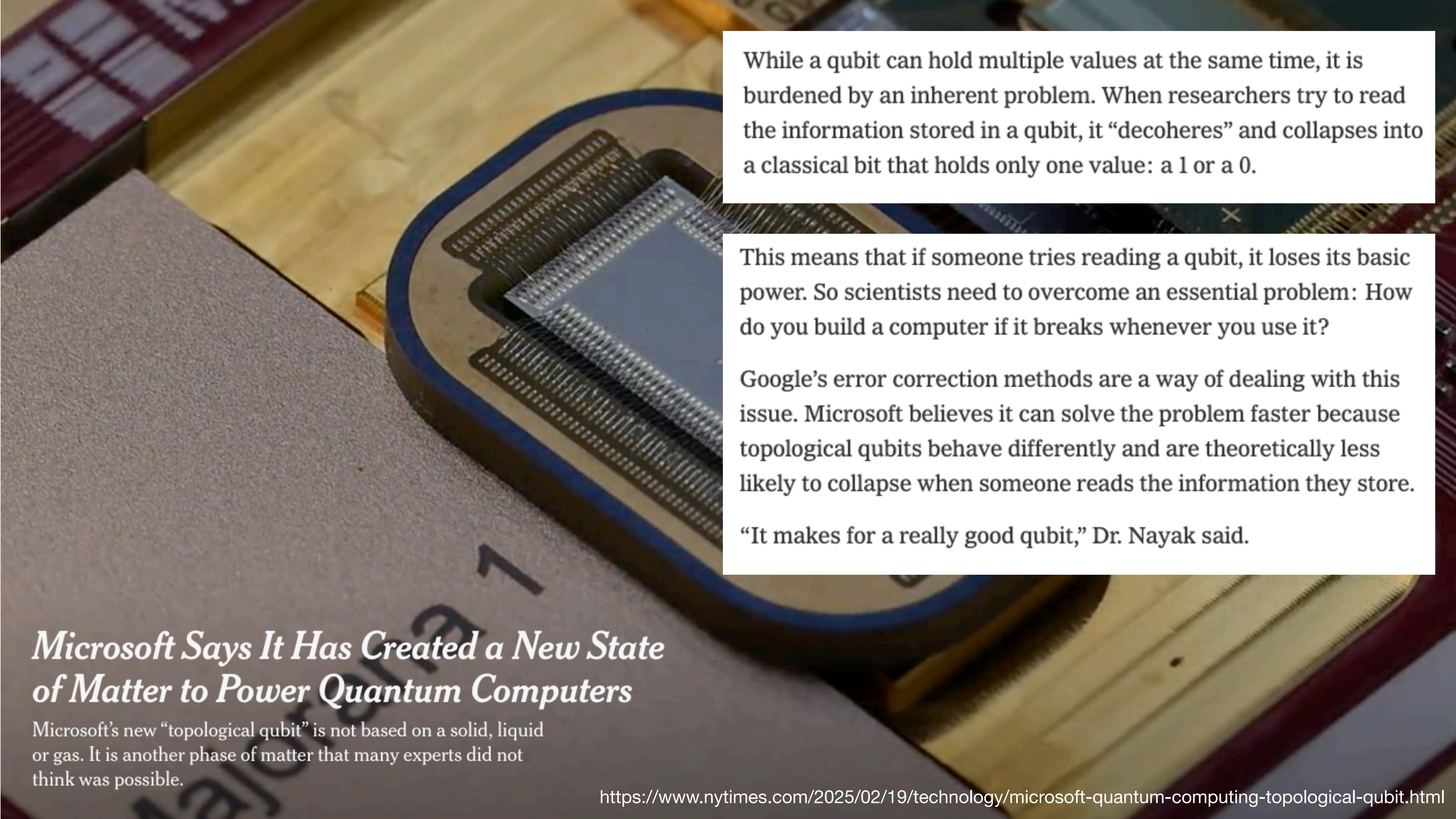


While a qubit can hold multiple values at the same time, it is burdened by an inherent problem. When researchers try to read the information stored in a qubit, it “decoheres” and collapses into a classical bit that holds only one value: a 1 or a 0.

Microsoft Says It Has Created a New State of Matter to Power Quantum Computers

Microsoft’s new “topological qubit” is not based on a solid, liquid or gas. It is another phase of matter that many experts did not think was possible.

<https://www.nytimes.com/2025/02/19/technology/microsoft-quantum-computing-topological-qubit.html>



While a qubit can hold multiple values at the same time, it is burdened by an inherent problem. When researchers try to read the information stored in a qubit, it “decoheres” and collapses into a classical bit that holds only one value: a 1 or a 0.

This means that if someone tries reading a qubit, it loses its basic power. So scientists need to overcome an essential problem: How do you build a computer if it breaks whenever you use it?

Google’s error correction methods are a way of dealing with this issue. Microsoft believes it can solve the problem faster because topological qubits behave differently and are theoretically less likely to collapse when someone reads the information they store.

“It makes for a really good qubit,” Dr. Nayak said.

Microsoft Says It Has Created a New State of Matter to Power Quantum Computers

Microsoft’s new “topological qubit” is not based on a solid, liquid or gas. It is another phase of matter that many experts did not think was possible.

operating systems enforce modularity on a single machine using **virtualization**

in order to enforce modularity + have an effective operating system, a few things need to happen

- | | | |
|---|--------|--|
| 1. programs shouldn't be able to refer to (and corrupt) each others' memory |→ | virtual memory |
| 2. programs should be able to communicate with each other |→ | bounded buffers
(virtualize communication links) |
| 3. programs should be able to share a CPU without one program halting the progress of the others |→ | threads
(virtualize processors) |

operating systems enforce modularity on a single machine using **virtualization**

in order to enforce modularity + have an effective operating system, a few things need to happen

- | | | |
|---|--------|--|
| 1. programs shouldn't be able to refer to
(and corrupt) each others' memory |→ | virtual memory |
| 2. programs should be able to
communicate with each other |→ | bounded buffers
(virtualize communication links) |
| 3. programs should be able to share a CPU without one program halting the progress of the others |→ | threads
(virtualize processors) |

you've also seen virtualization as a technique for running multiple operating systems on the same physical hardware

operating systems enforce modularity on a single machine using **virtualization**

in order to enforce modularity + have an effective operating system, a few things need to happen

- | | | |
|---|--------|--|
| 1. programs shouldn't be able to refer to (and corrupt) each others' memory |→ | virtual memory |
| 2. programs should be able to communicate with each other |→ | bounded buffers
(virtualize communication links) |
| 3. programs should be able to share a CPU without one program halting the progress of the others |→ | threads
(virtualize processors) |

today: performance more generally, with a focus on storage, and how the abstractions that an operating system provides impact our systems

performance issues have influenced a lot of the system designs you've seen so far

performance issues have influenced a lot of the system designs you've seen so far

latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

performance issues have influenced a lot of the system designs you've seen so far

latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

performance issues have influenced a lot of the system designs you've seen so far

latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context

performance issues have influenced a lot of the system designs you've seen so far

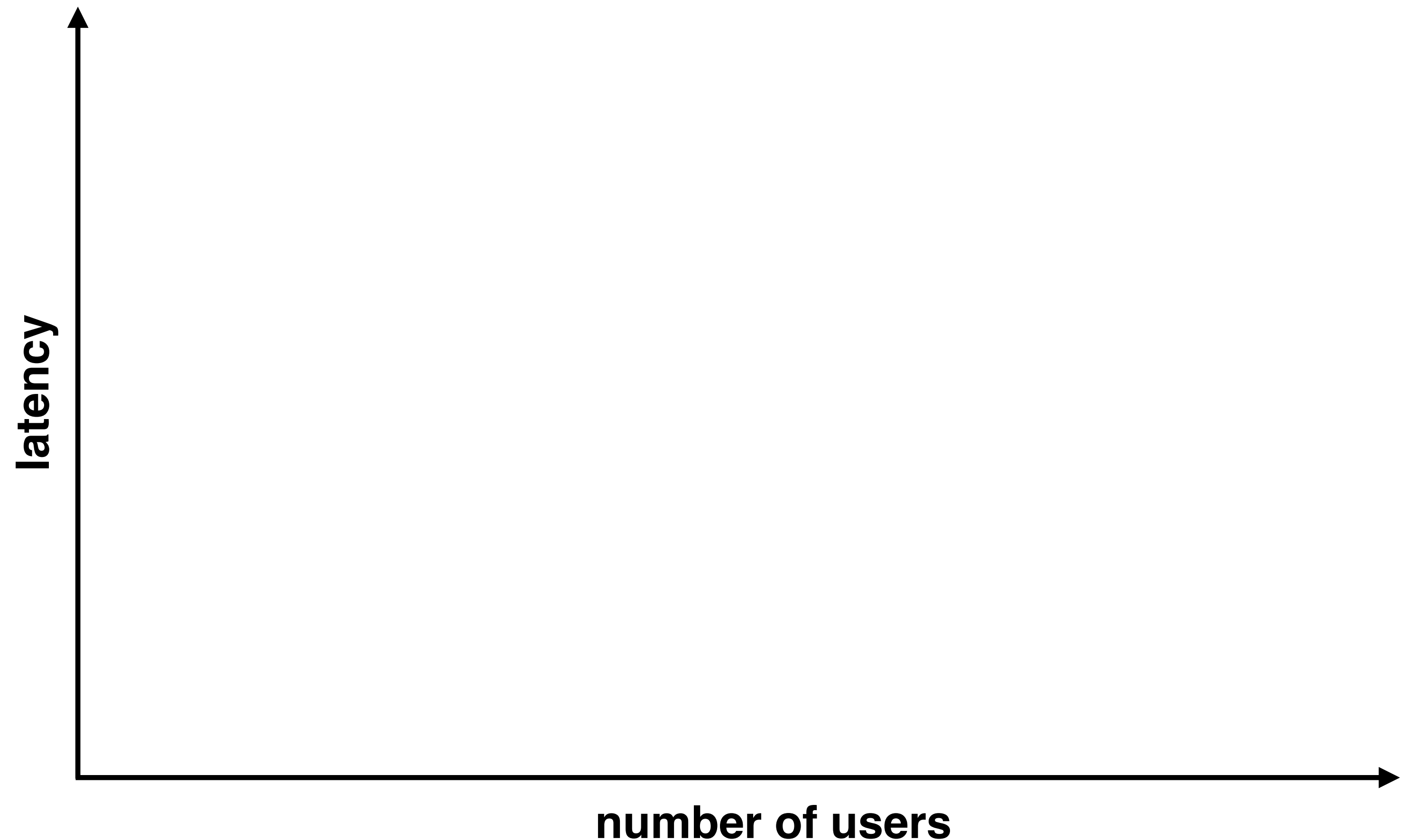
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



performance issues have influenced a lot of the system designs you've seen so far

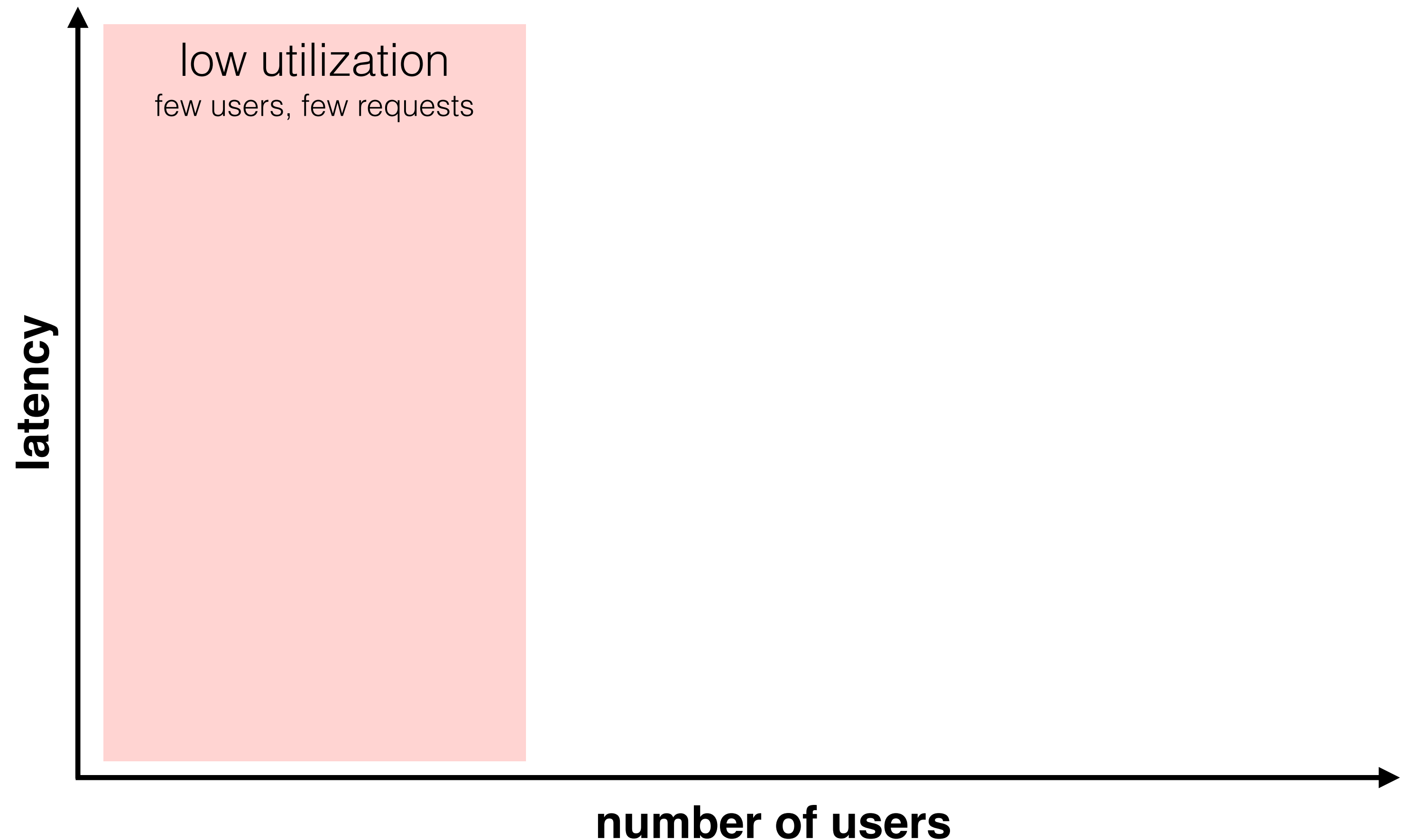
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



performance issues have influenced a lot of the system designs you've seen so far

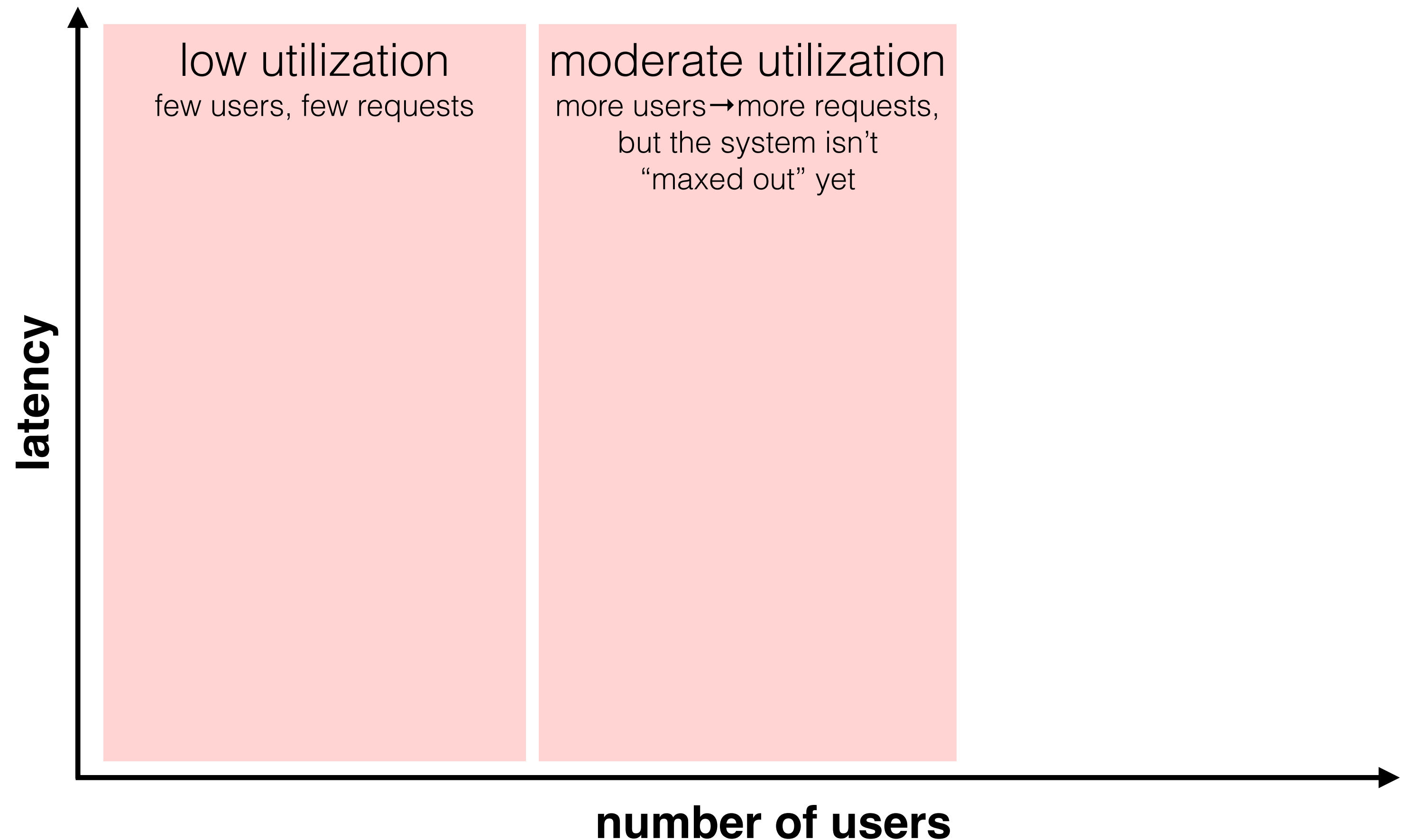
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



performance issues have influenced a lot of the system designs you've seen so far

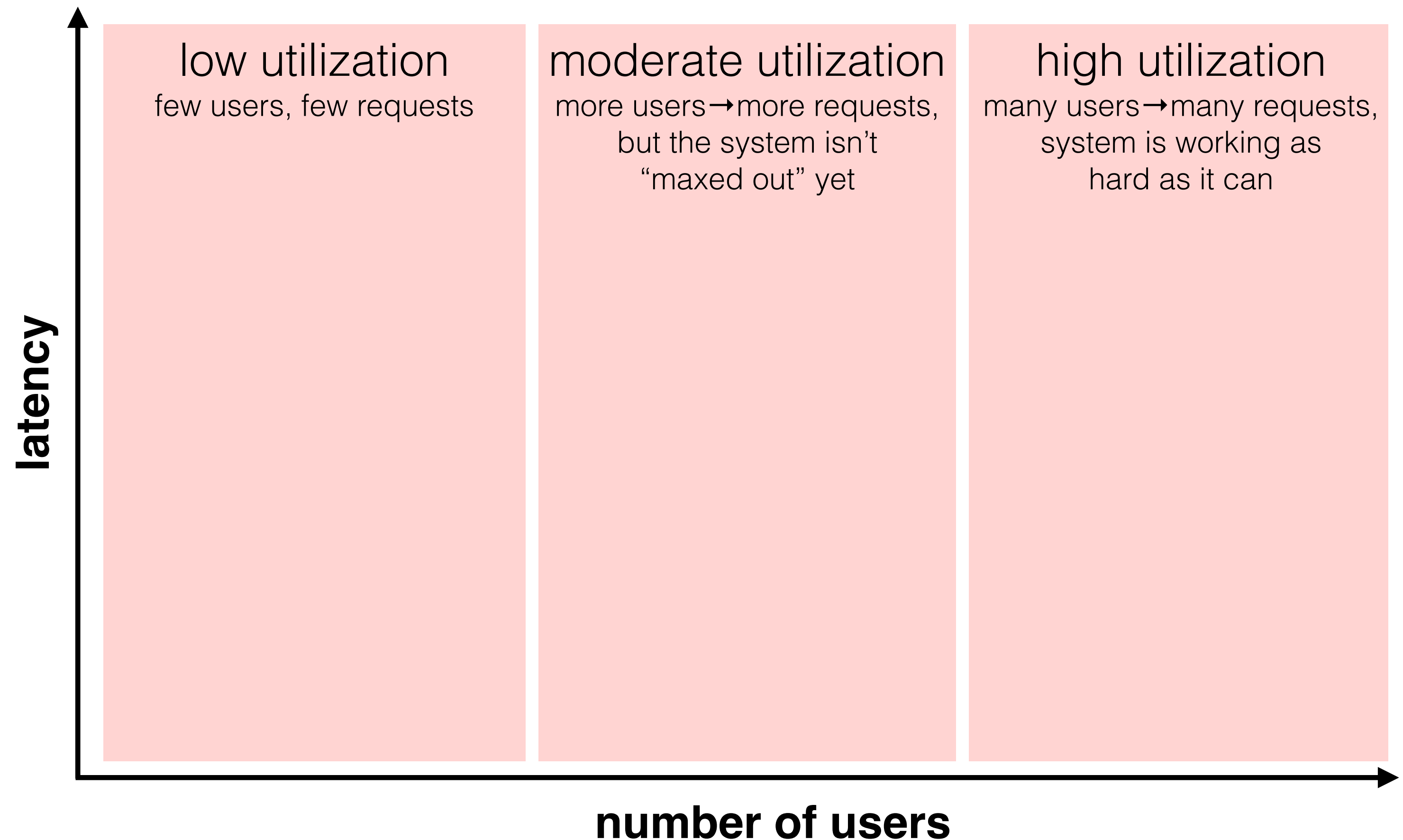
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



performance issues have influenced a lot of the system designs you've seen so far

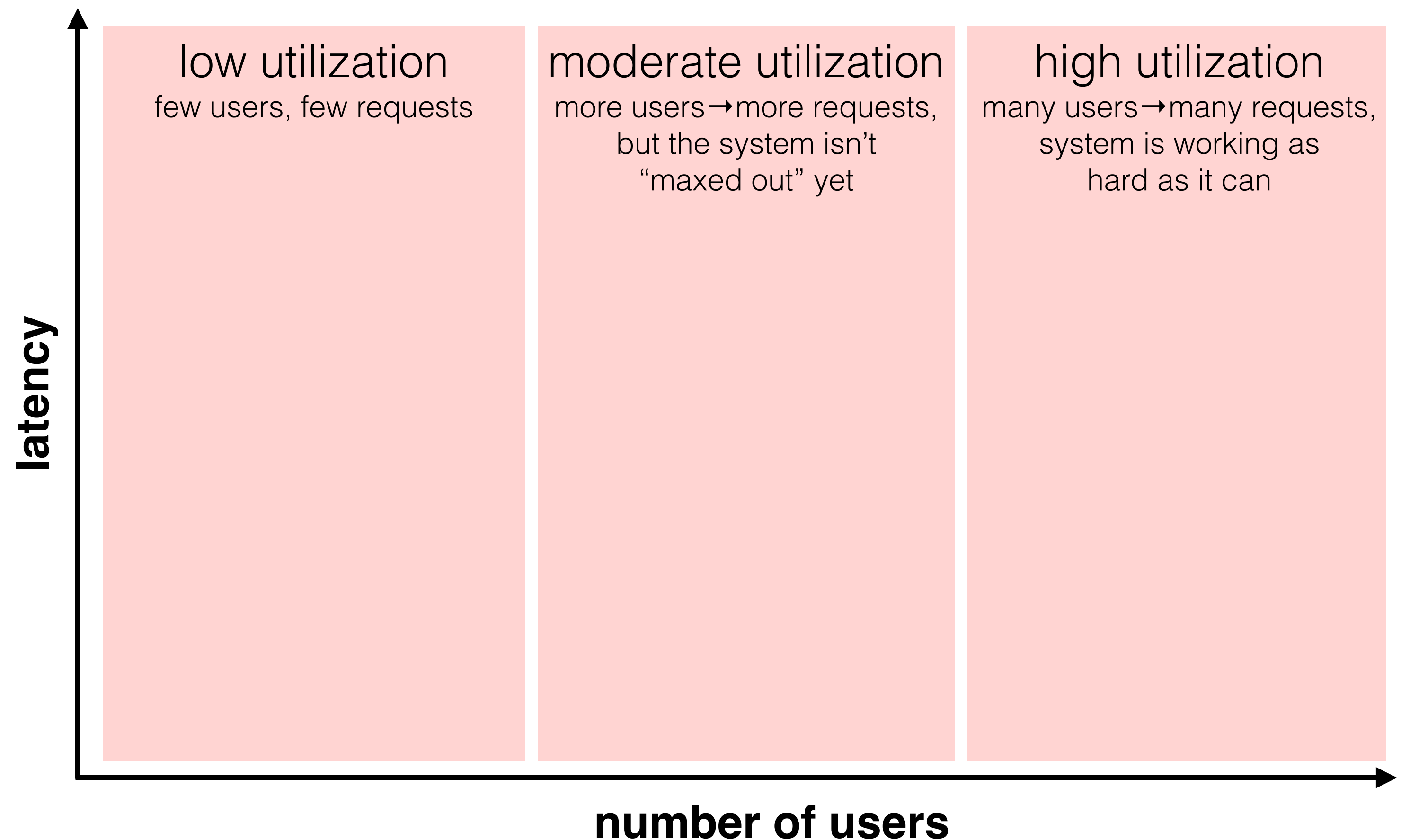
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



question: how do you expect latency to change as the system progresses through these stages?

performance issues have influenced a lot of the system designs you've seen so far

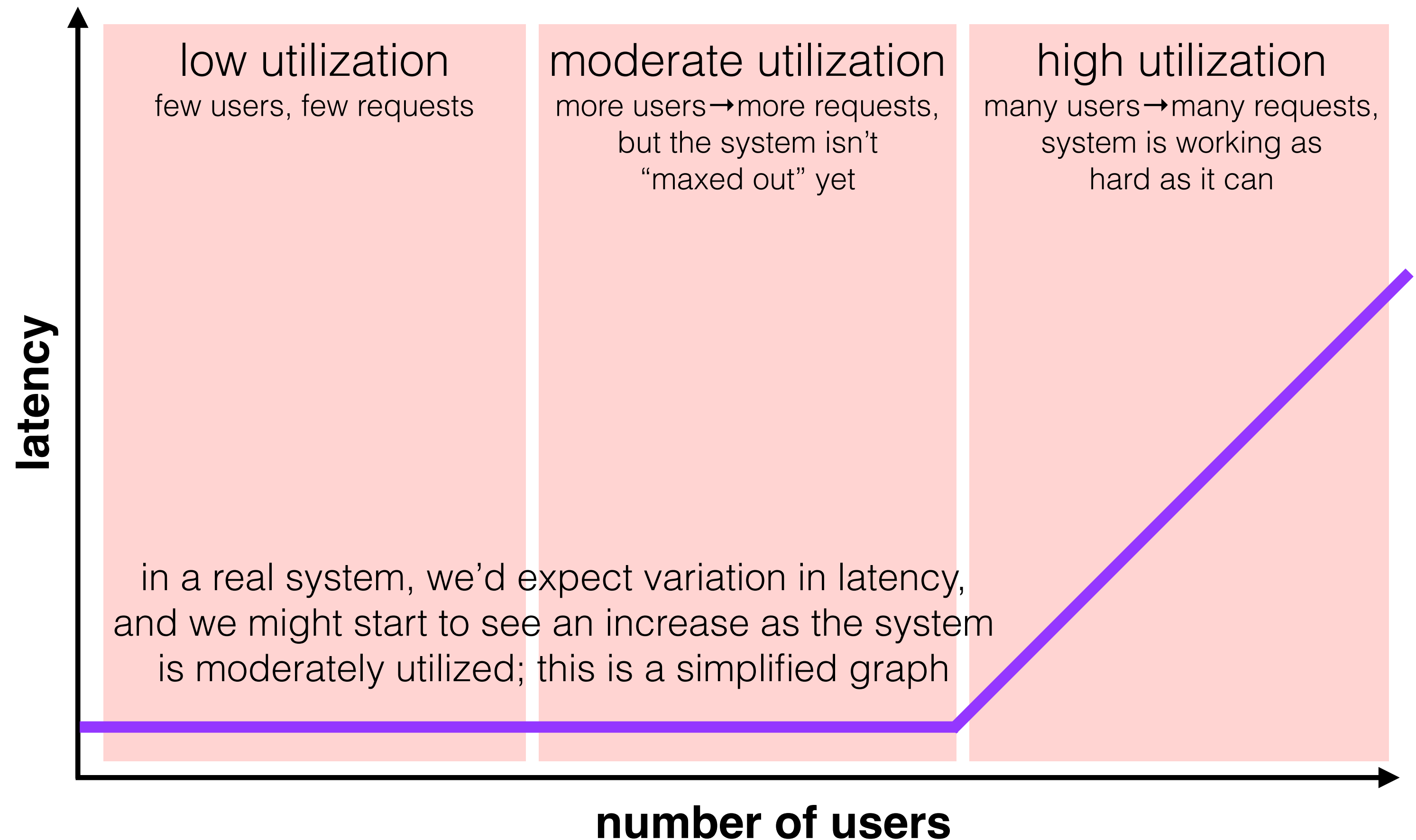
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



question: how do you expect latency to change as the system progresses through these stages?

performance issues have influenced a lot of the system designs you've seen so far

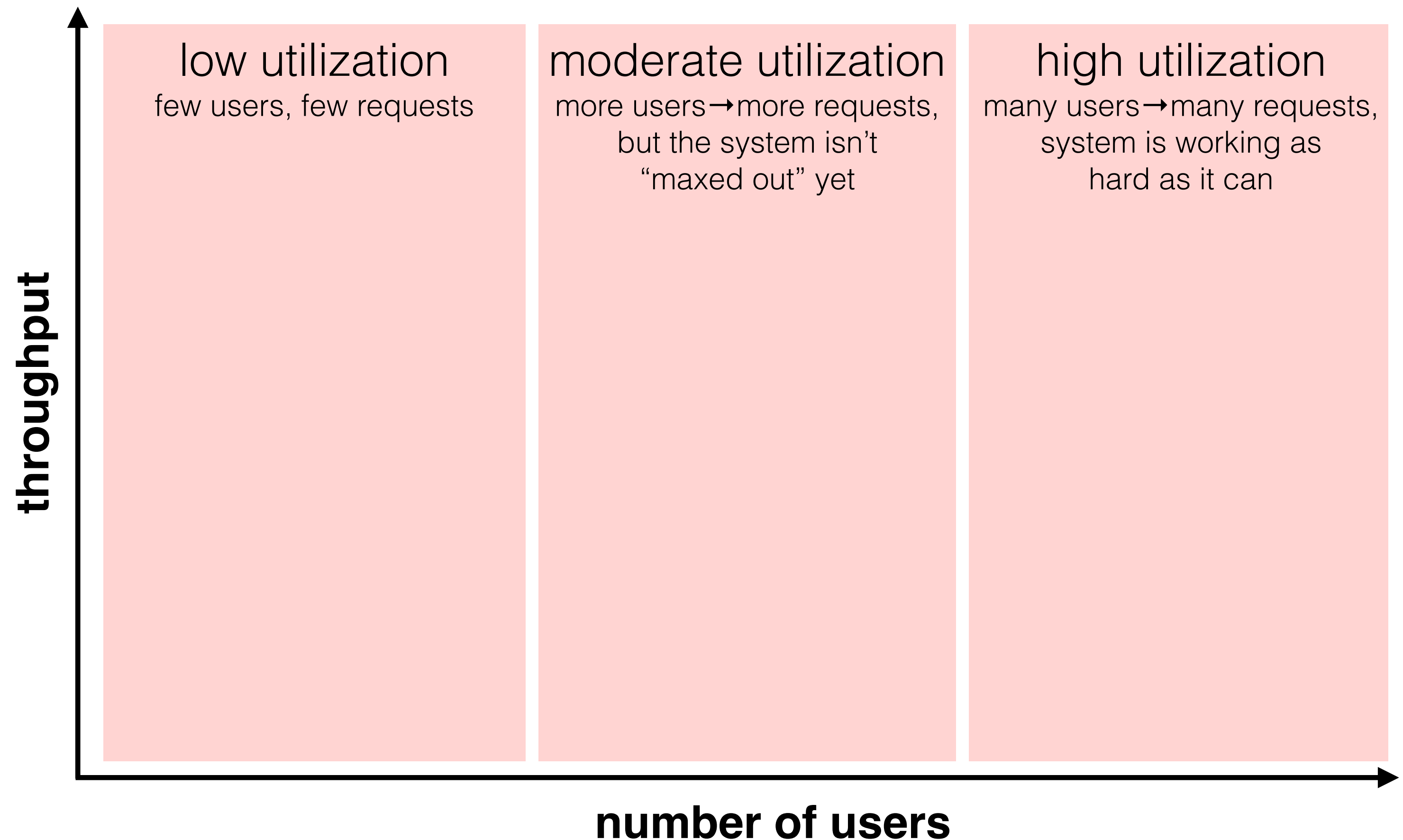
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



performance issues have influenced a lot of the system designs you've seen so far

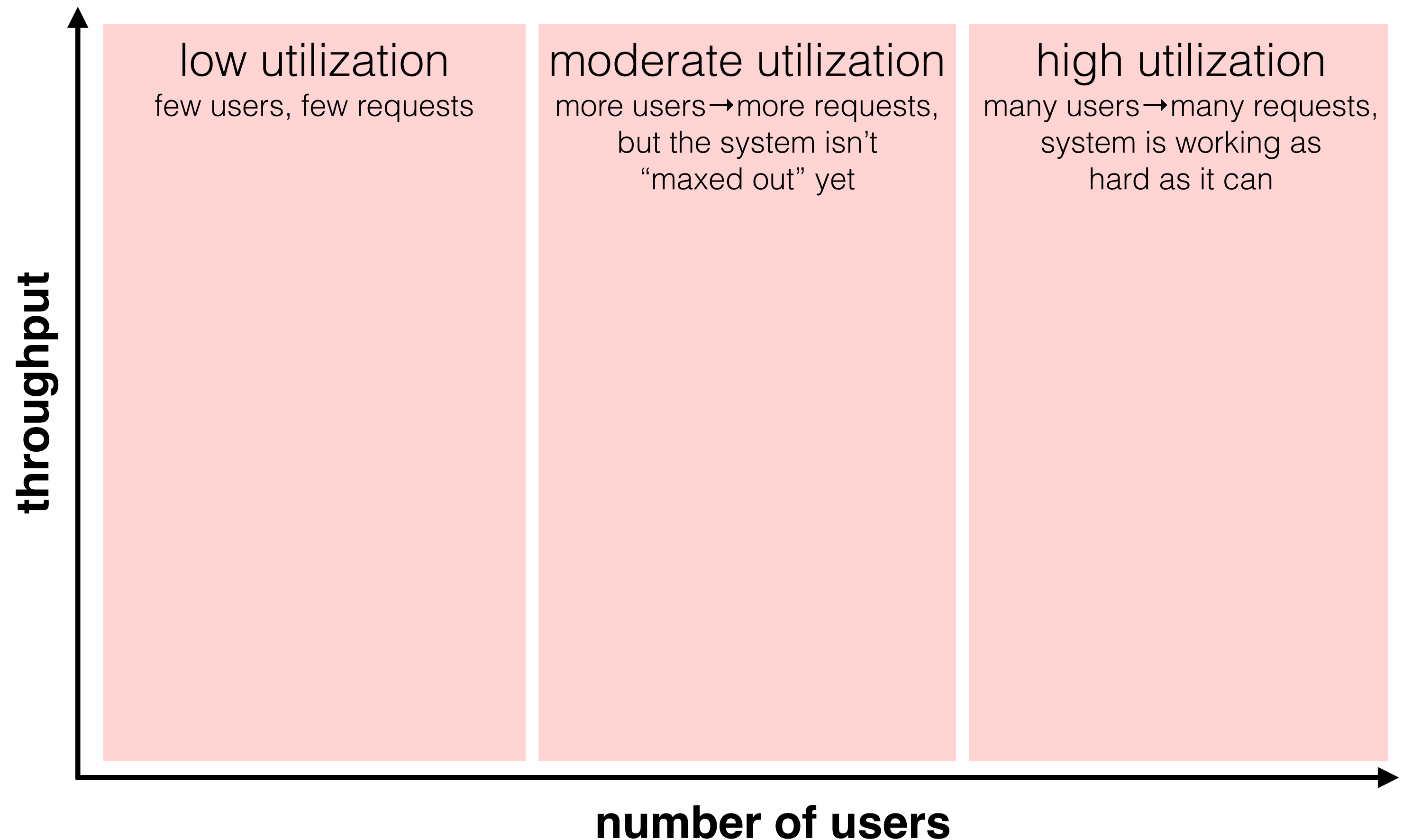
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



question: how do you expect throughput to change as the system progresses through these stages?

performance issues have influenced a lot of the system designs you've seen so far

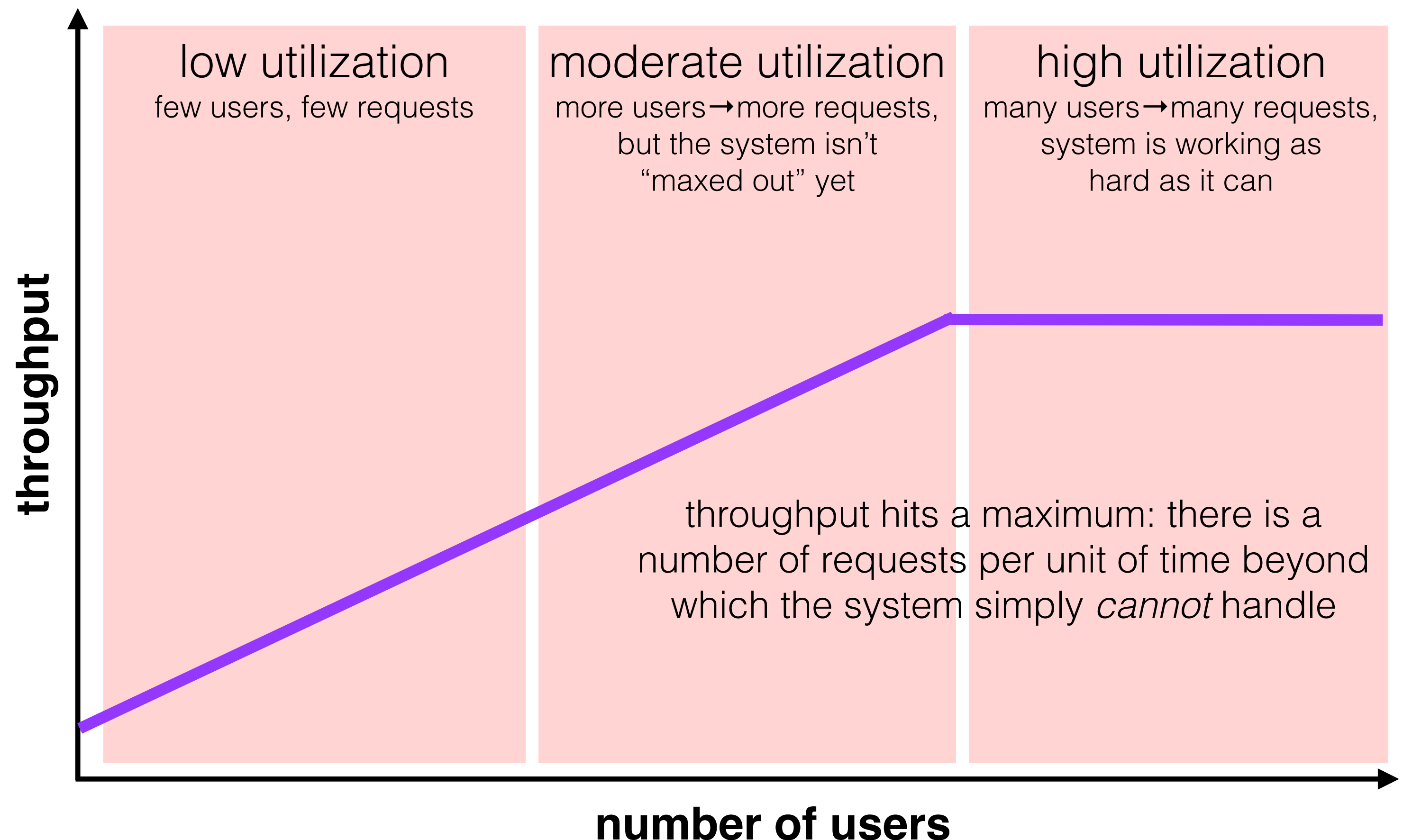
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context



question: how do you expect throughput to change as the system progresses through these stages?

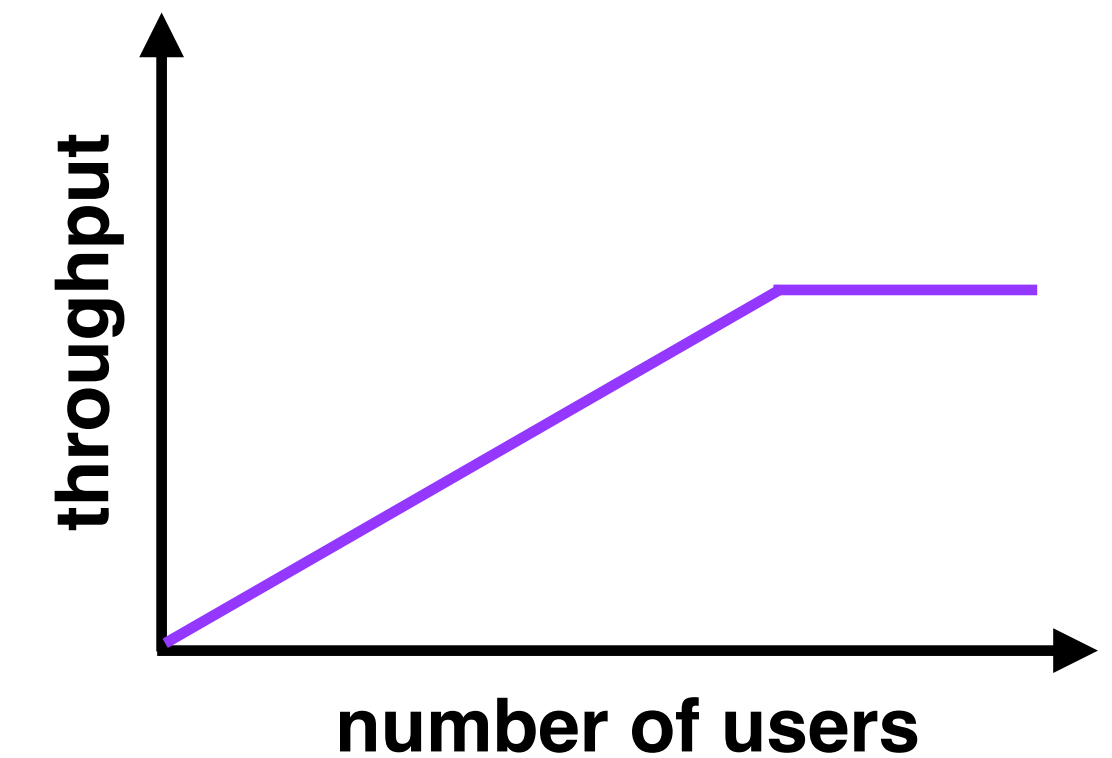
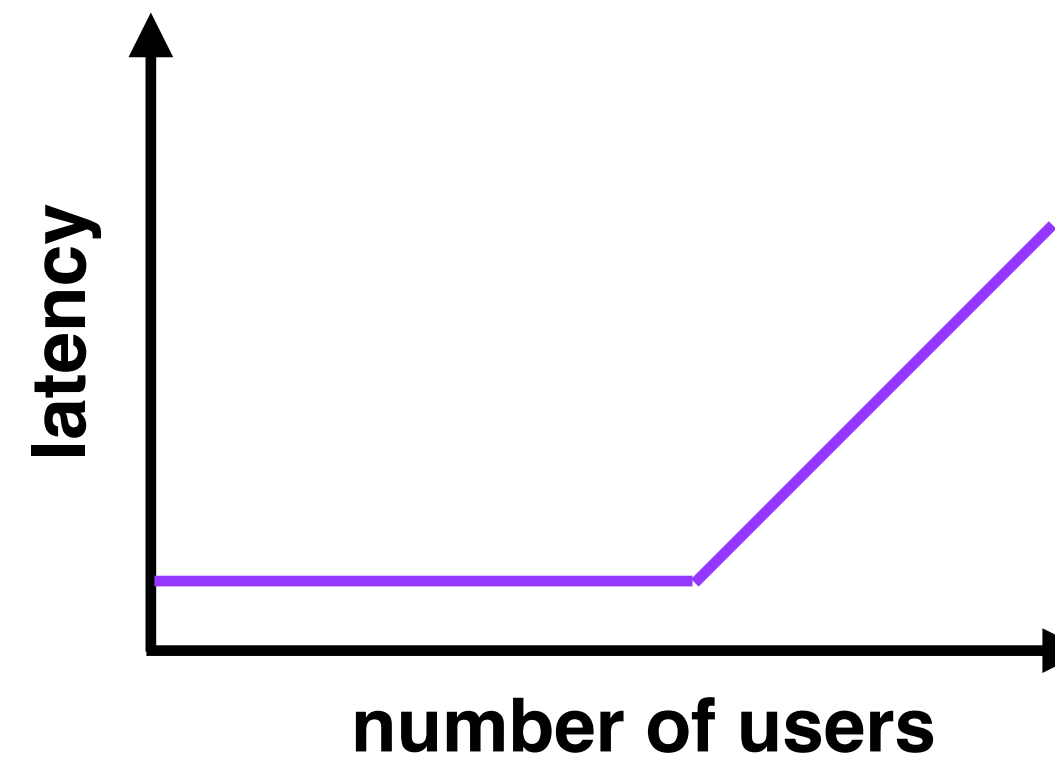
performance issues have influenced a lot of the system designs you've seen so far

latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?



utilization: what fraction of resources are being utilized? this puts our performance measurements in context

performance issues have influenced a lot of the system designs you've seen so far

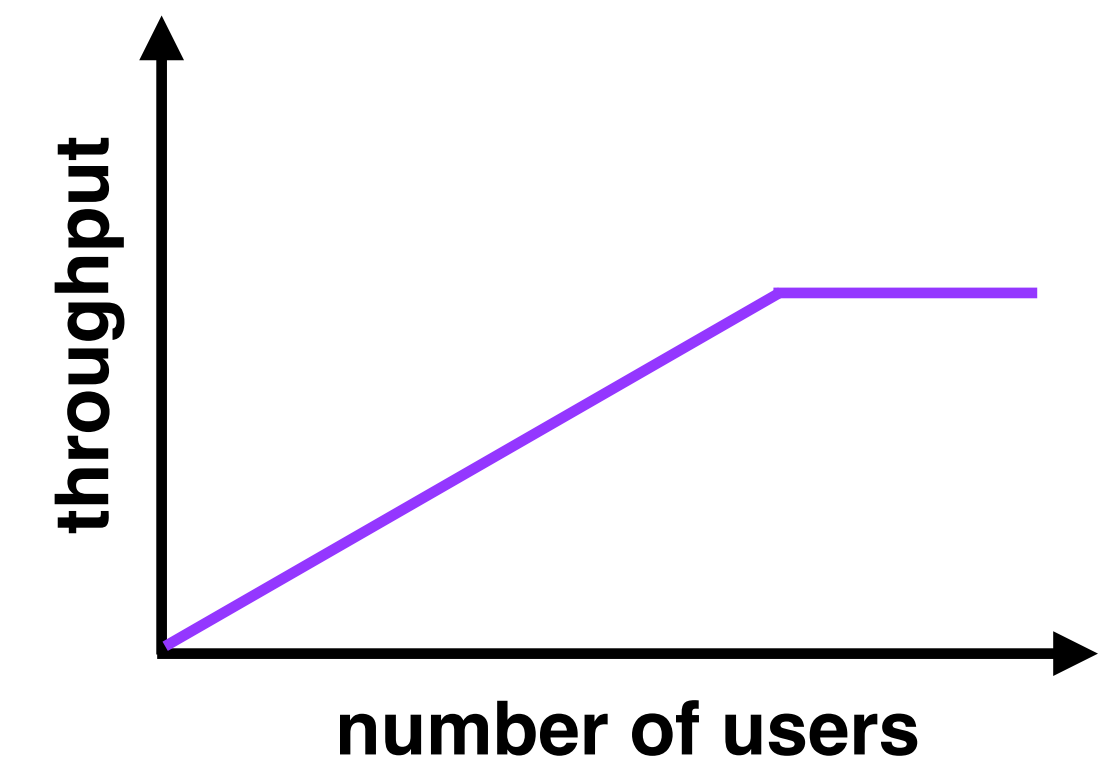
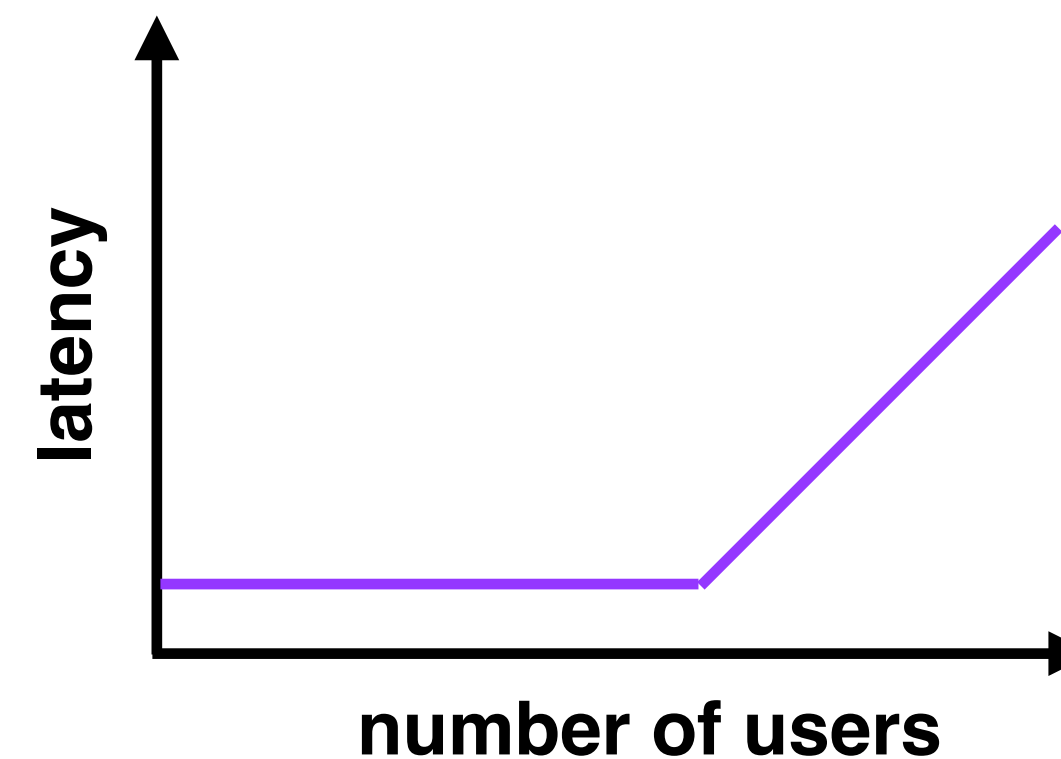
latency: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of data in an OS?

throughput: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

utilization: what fraction of resources are being utilized? this puts our performance measurements in context

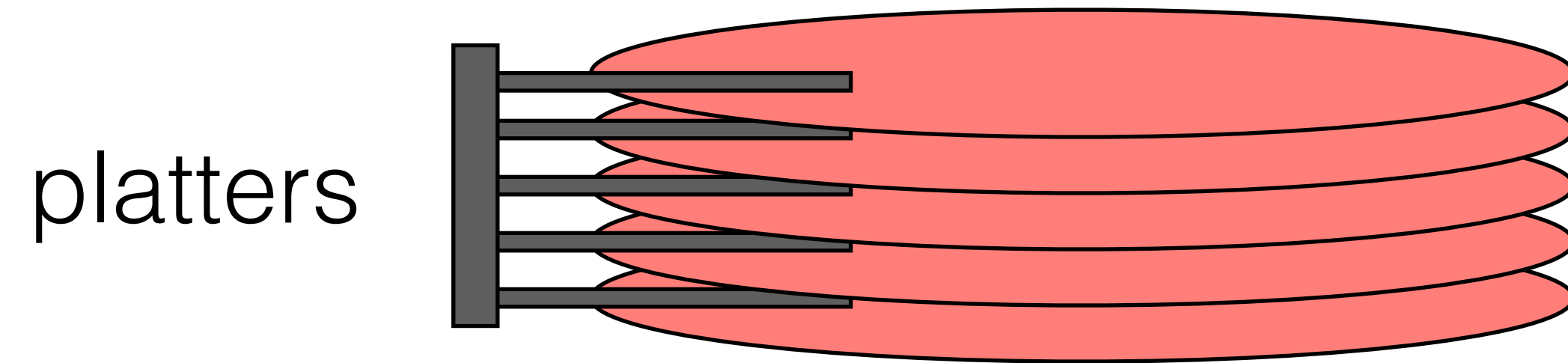


our general approach to improving performance is to measure our systems to find a **bottleneck**, and then to relax the bottleneck with general techniques such as caching, parallelism, etc.

we'll make this concrete with an example: performance in reading/writing to a file

the **disk** is often the main bottleneck in reading/writing stored data

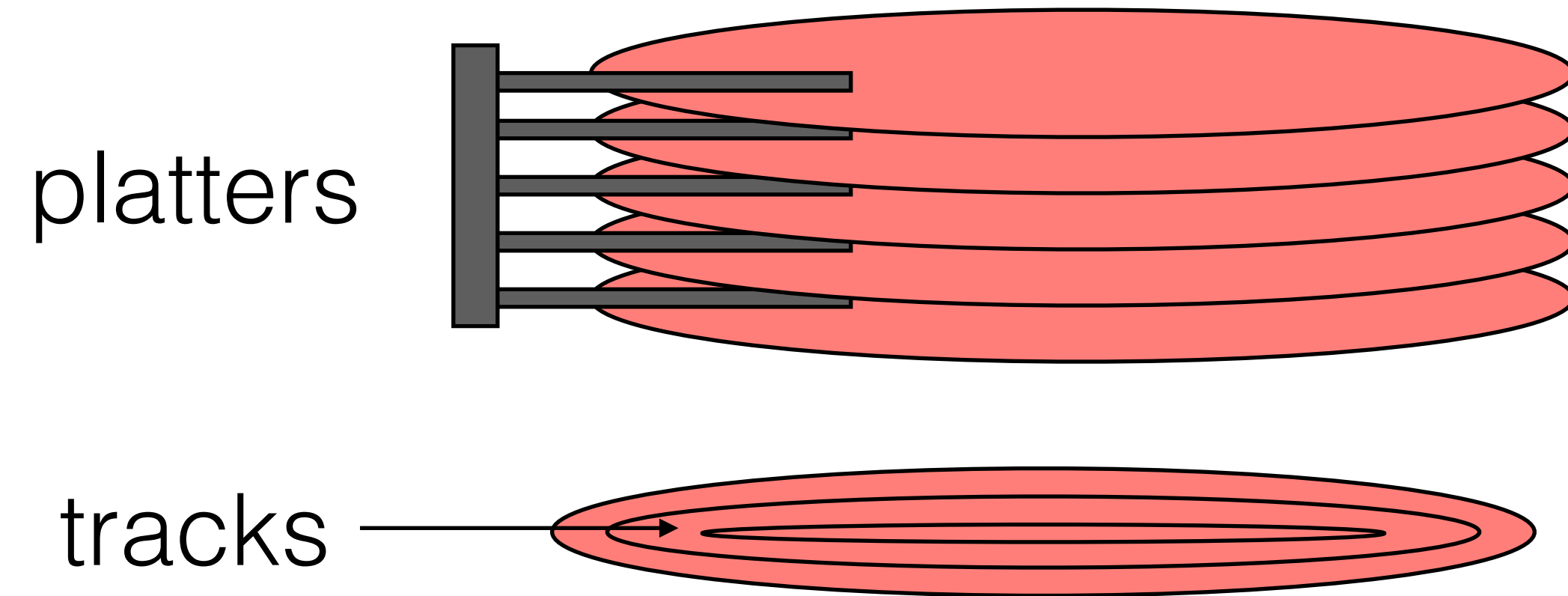
the **disk** is often the main bottleneck in reading/writing stored data



hard disk drives (HDDs)

common in datacenters

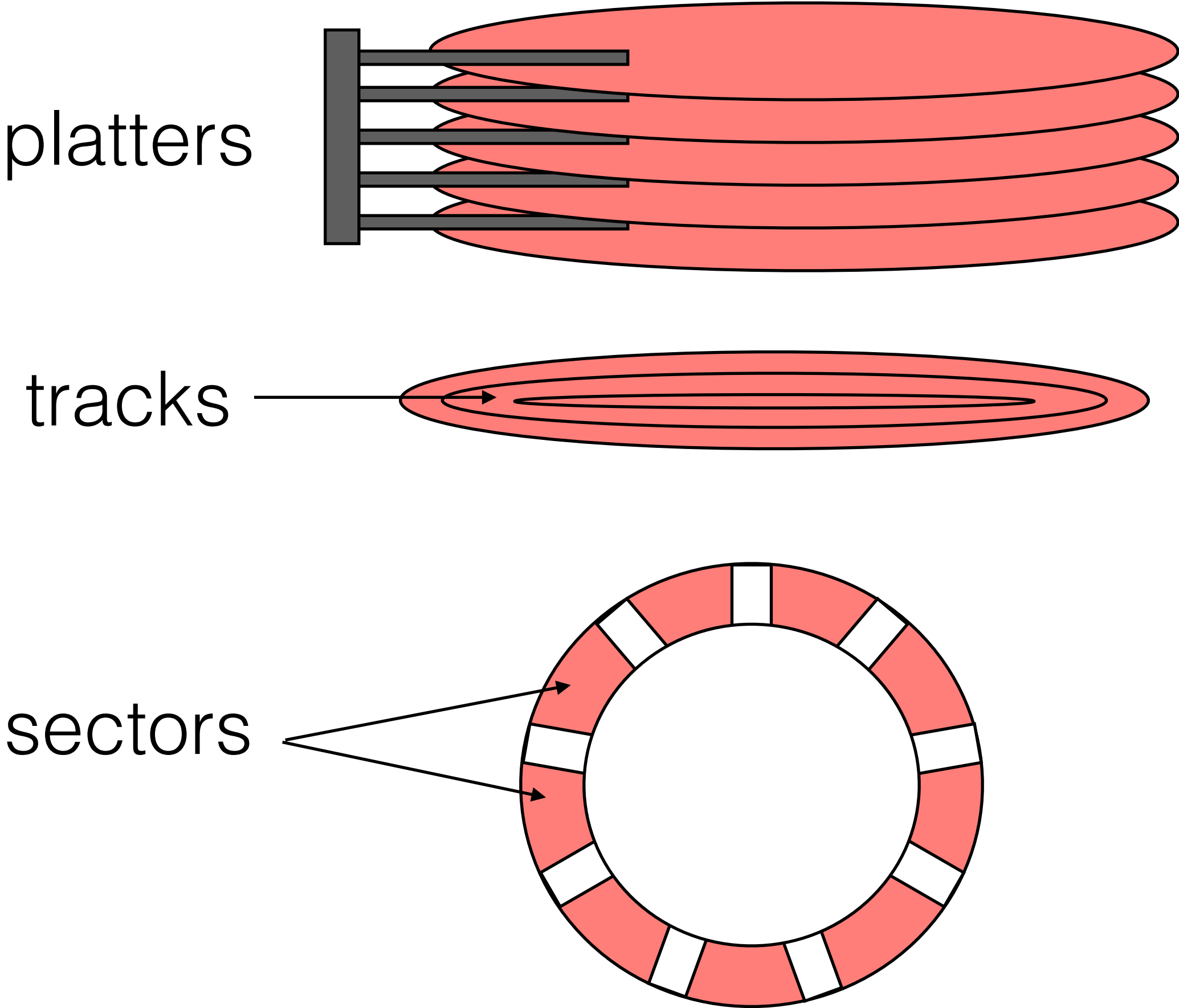
the **disk** is often the main bottleneck in reading/writing stored data



hard disk drives (HDDs)

common in datacenters

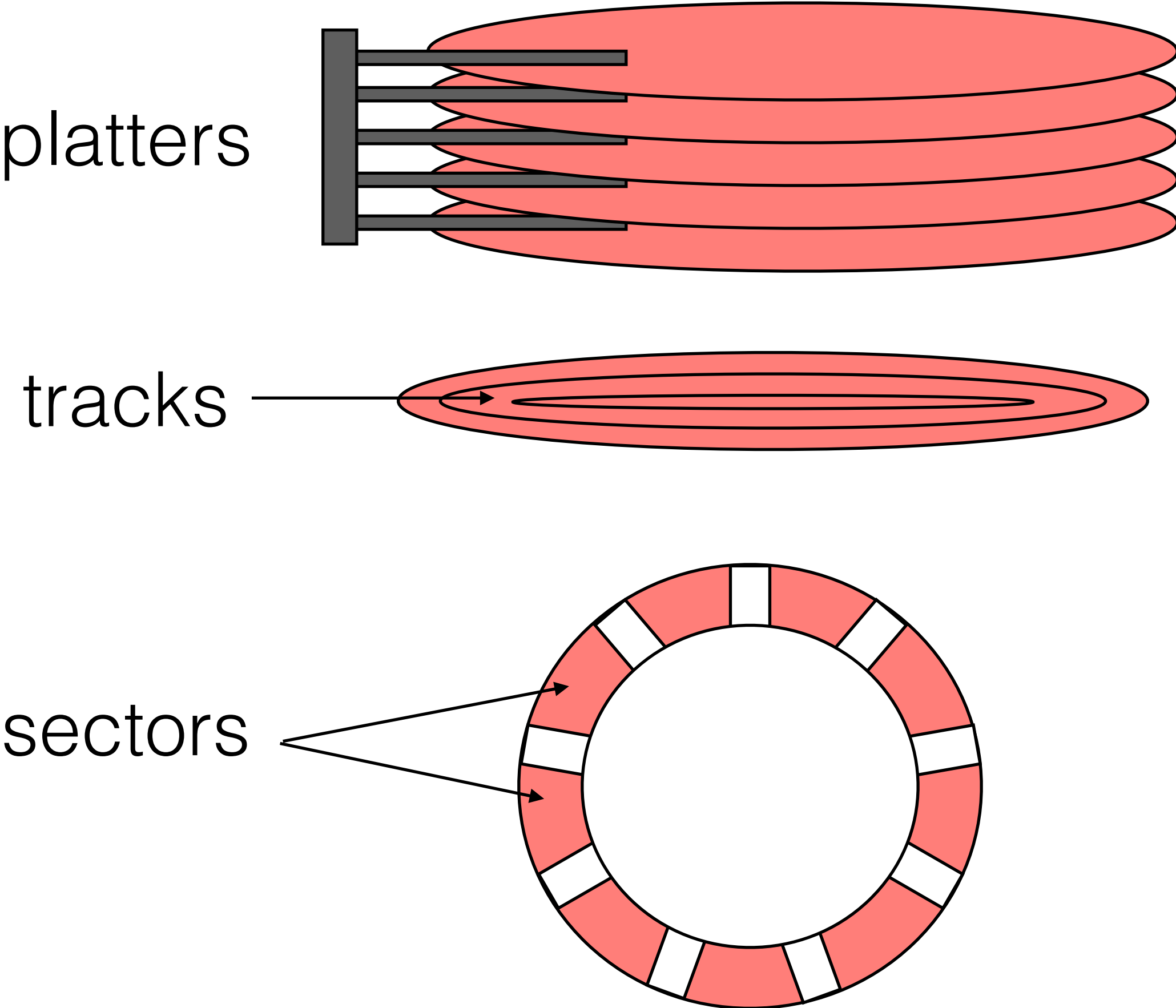
the **disk** is often the main bottleneck in reading/writing stored data



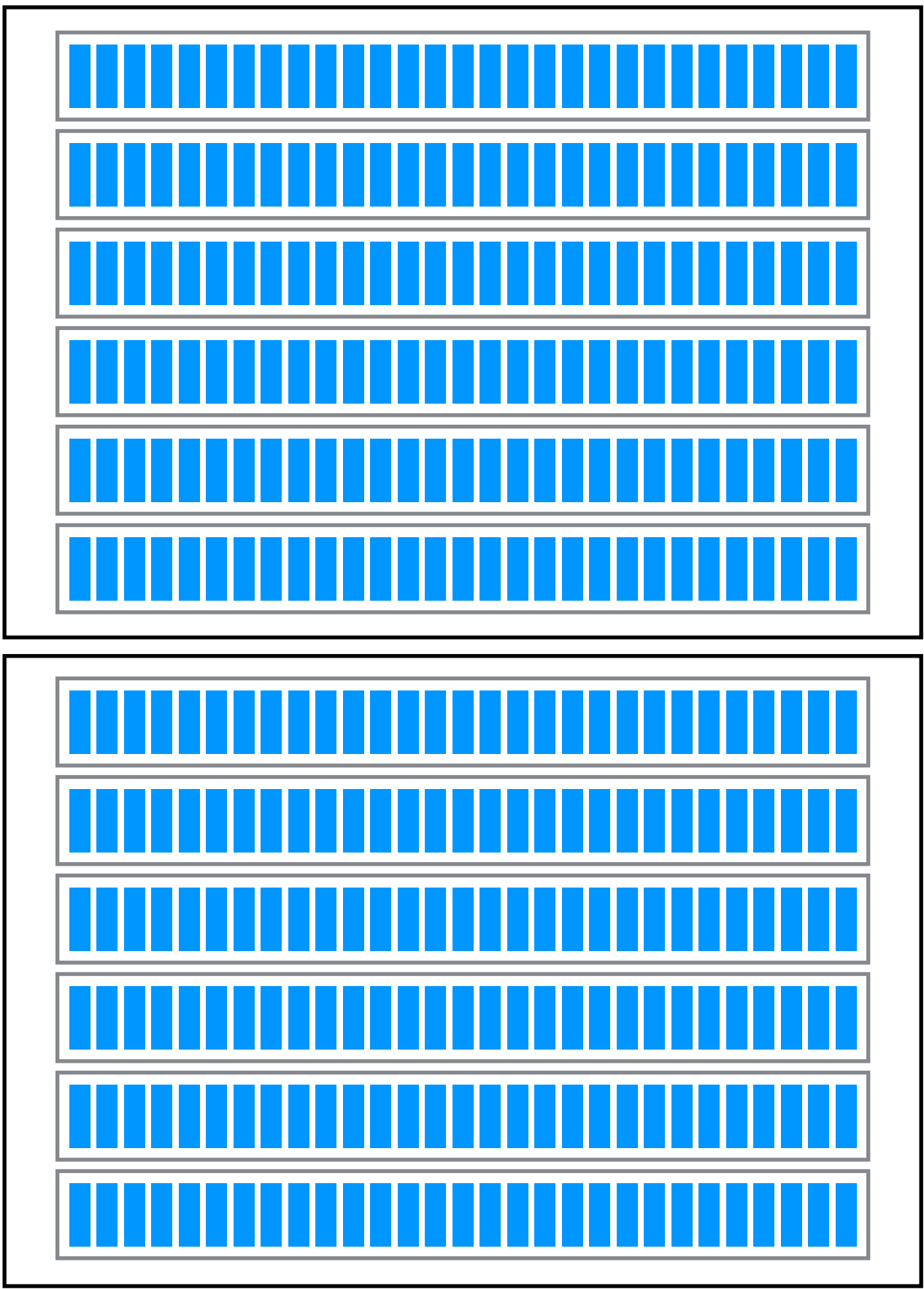
hard disk drives (HDDs)

common in datacenters

the **disk** is often the main bottleneck in reading/writing stored data

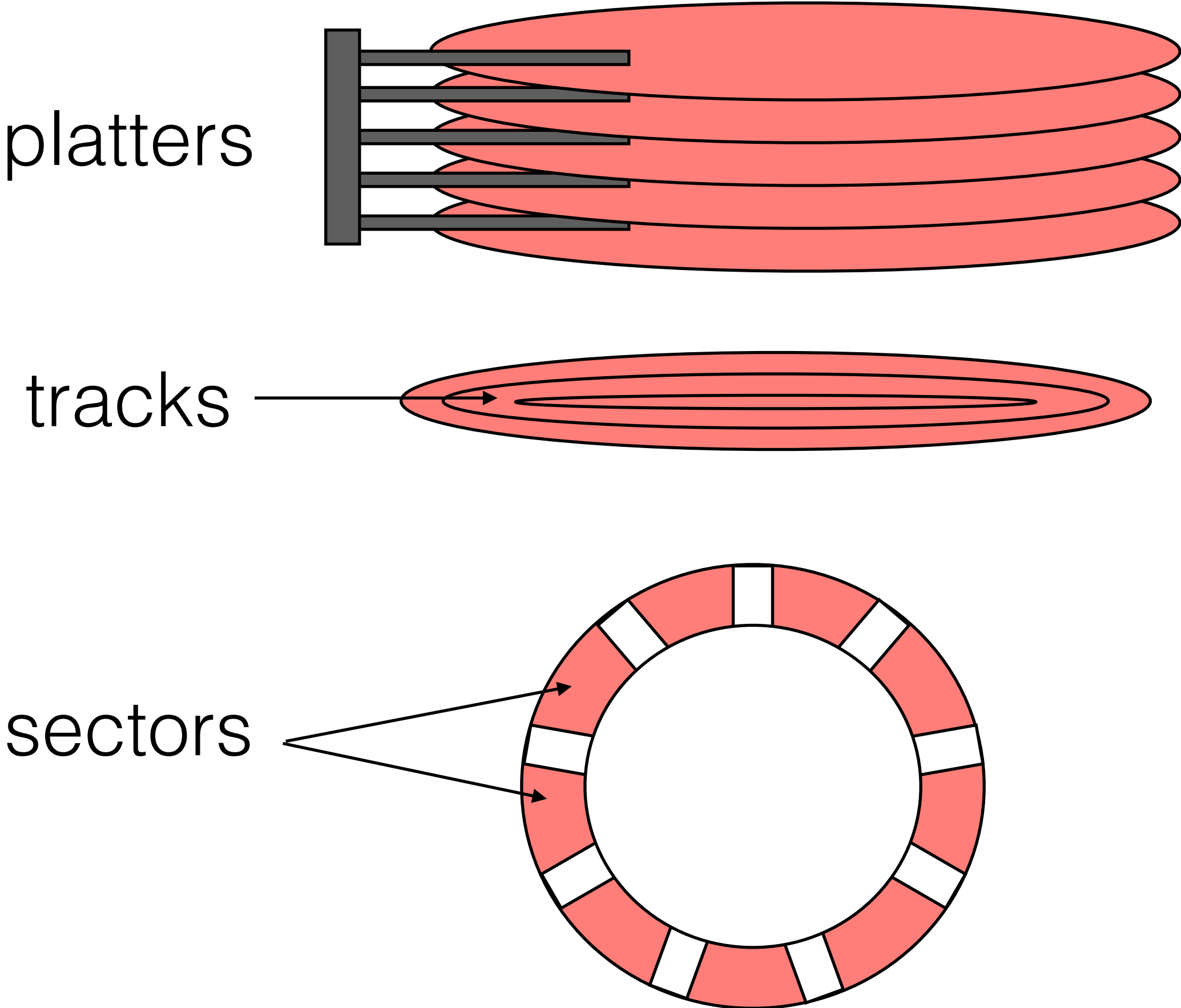


hard disk drives (HDDs)
common in datacenters

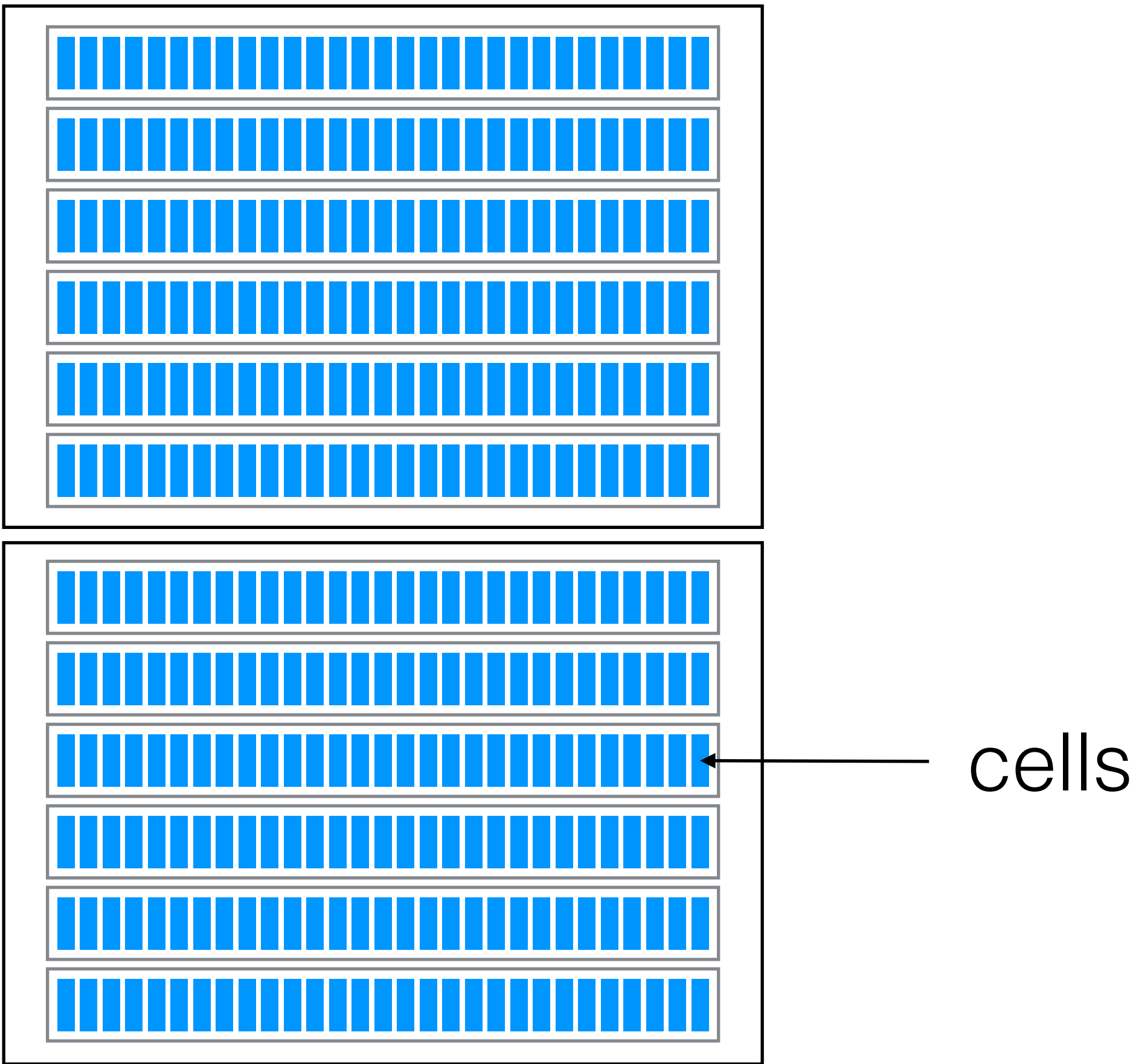


solid state drives (SSDs)
common in personal computers

the **disk** is often the main bottleneck in reading/writing stored data

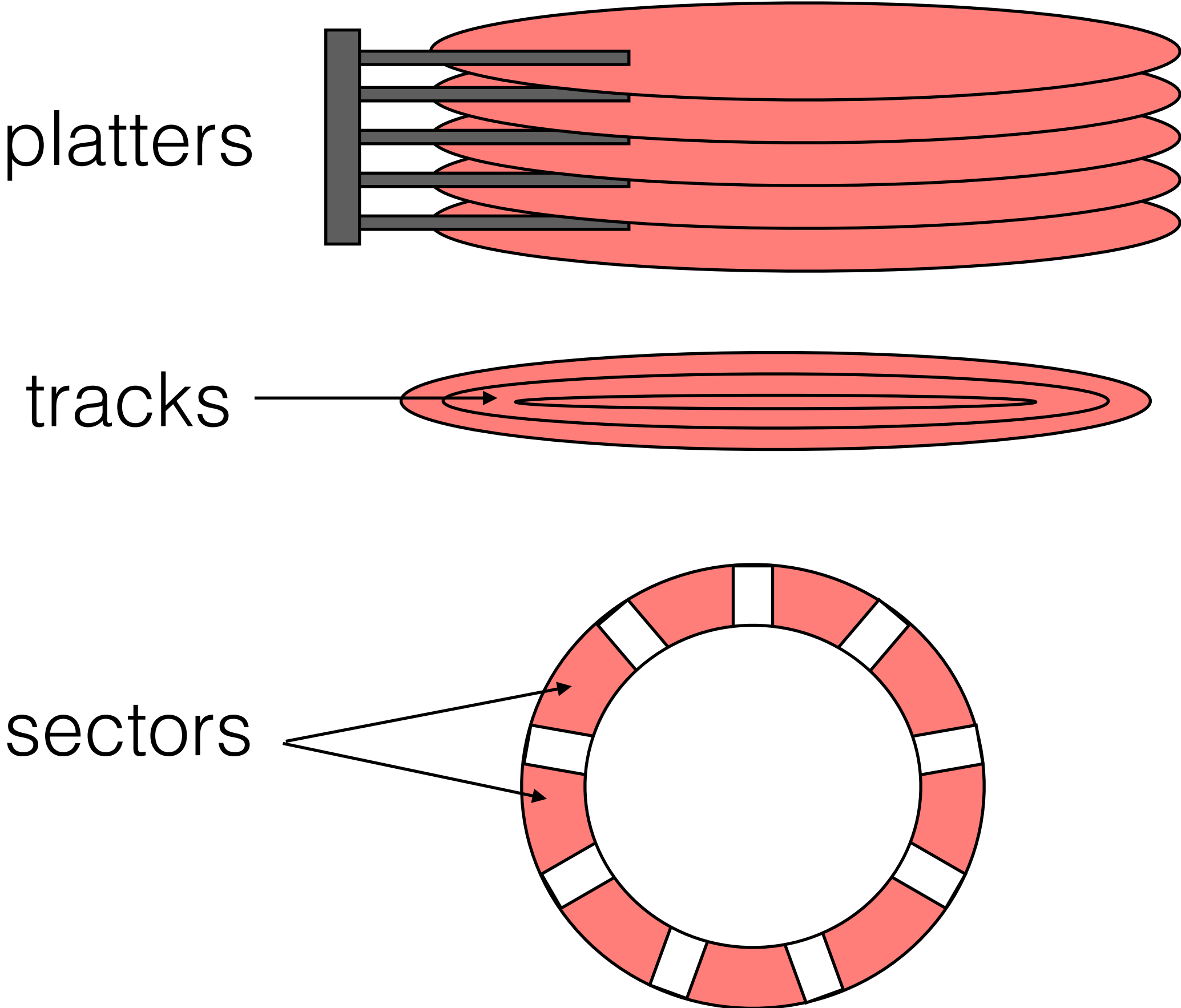


hard disk drives (HDDs)
common in datacenters



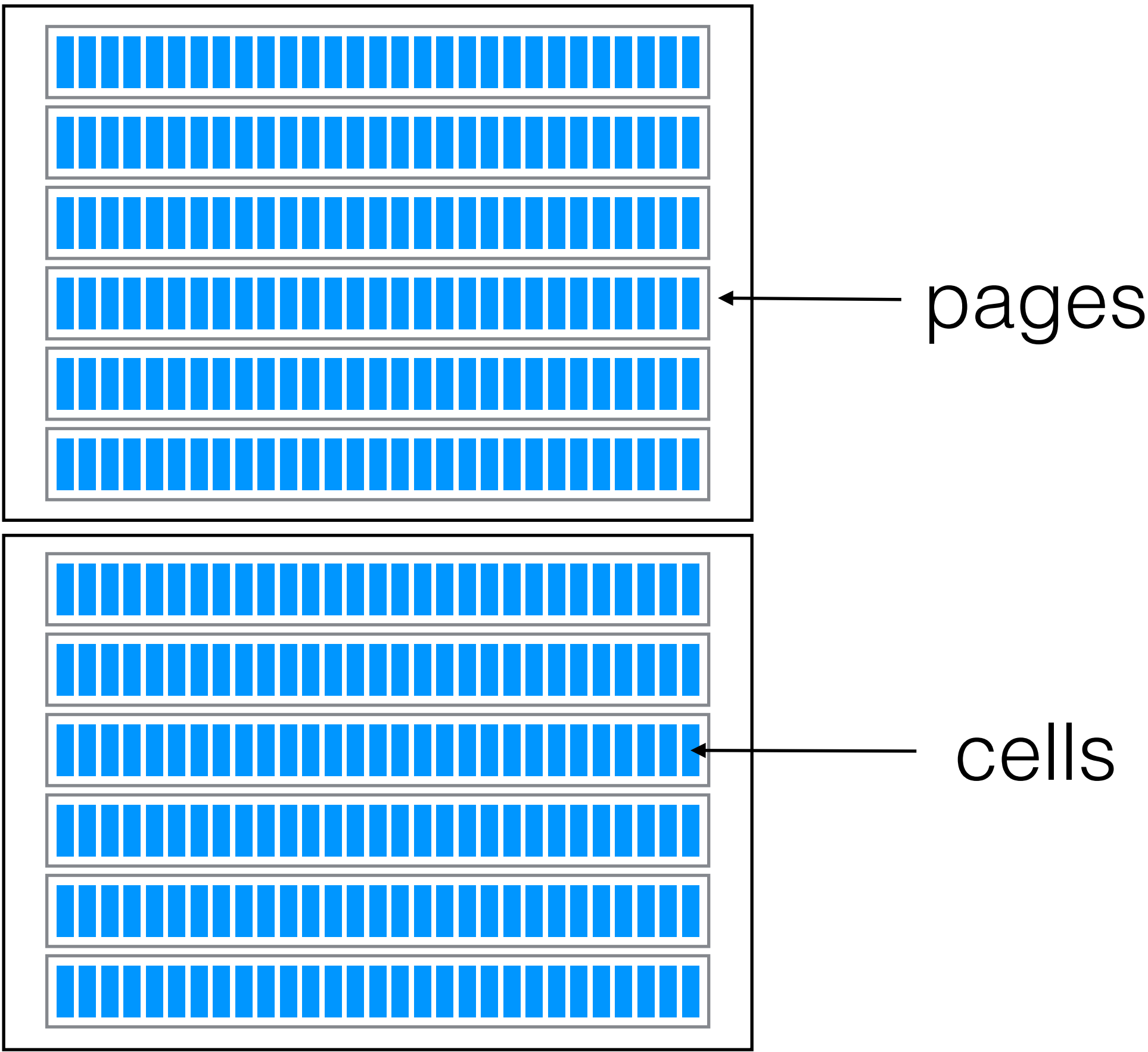
solid state drives (SSDs)
common in personal computers

the **disk** is often the main bottleneck in reading/writing stored data



hard disk drives (HDDs)

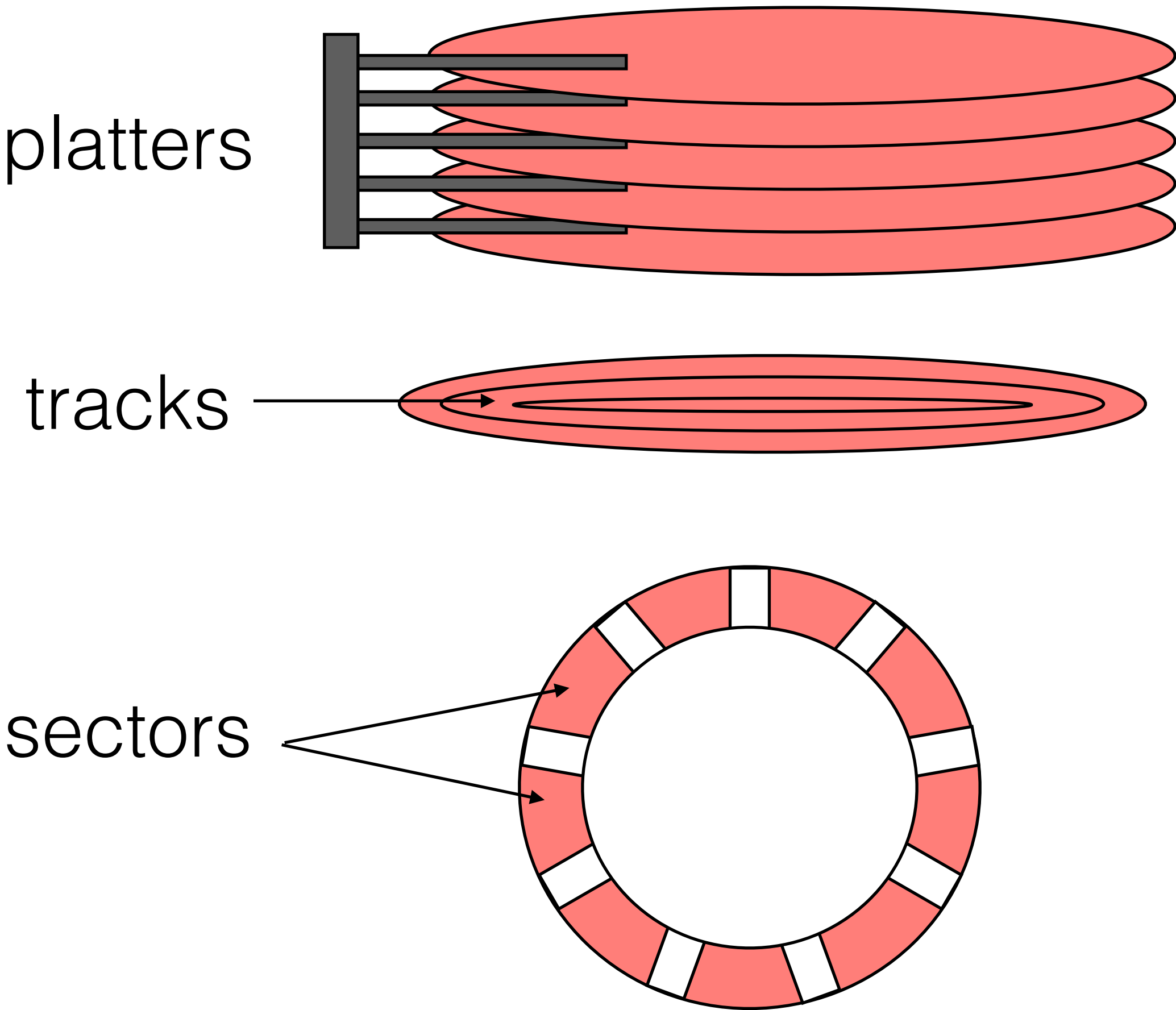
common in datacenters



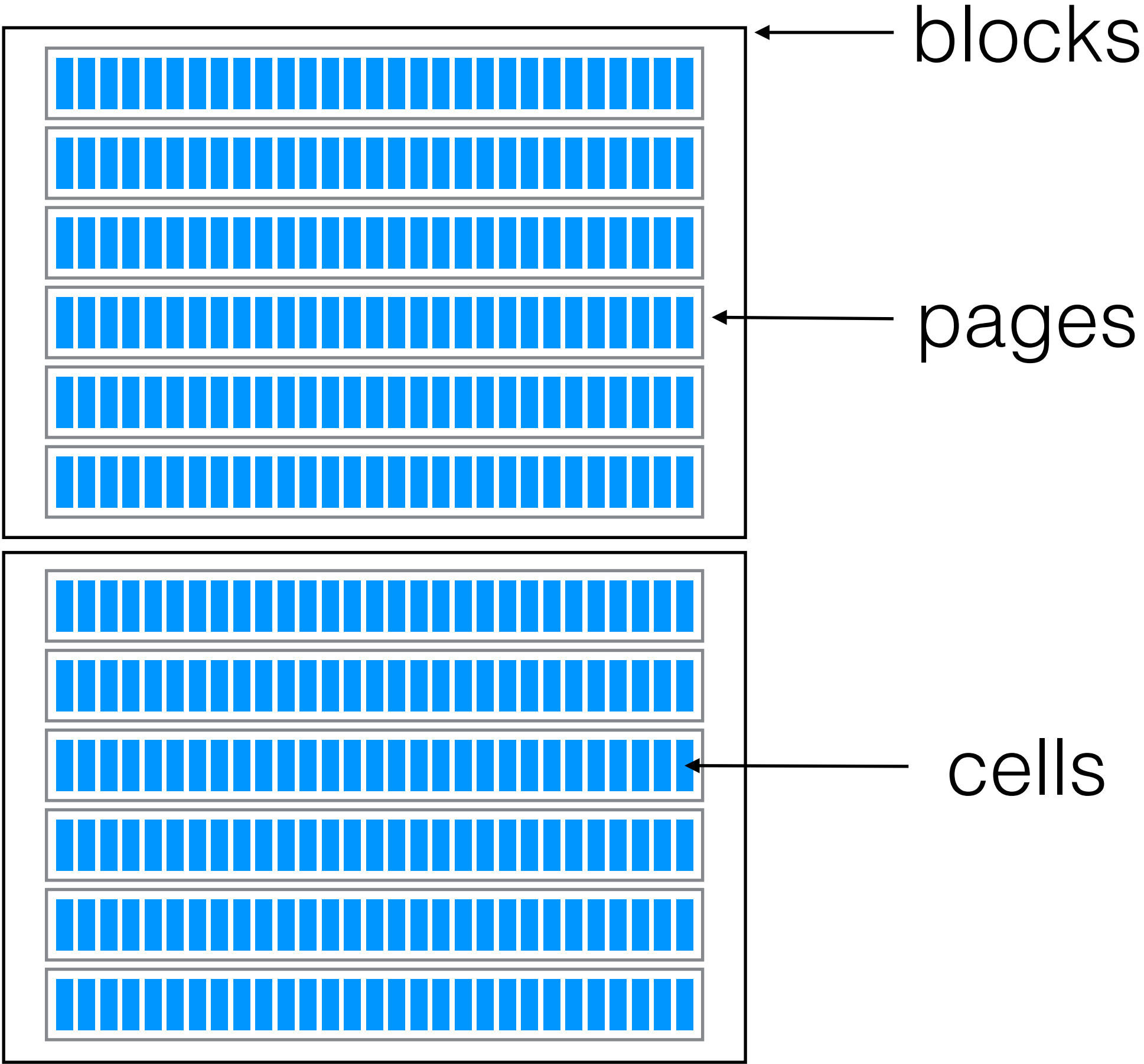
solid state drives (SSDs)

common in personal computers

the **disk** is often the main bottleneck in reading/writing stored data

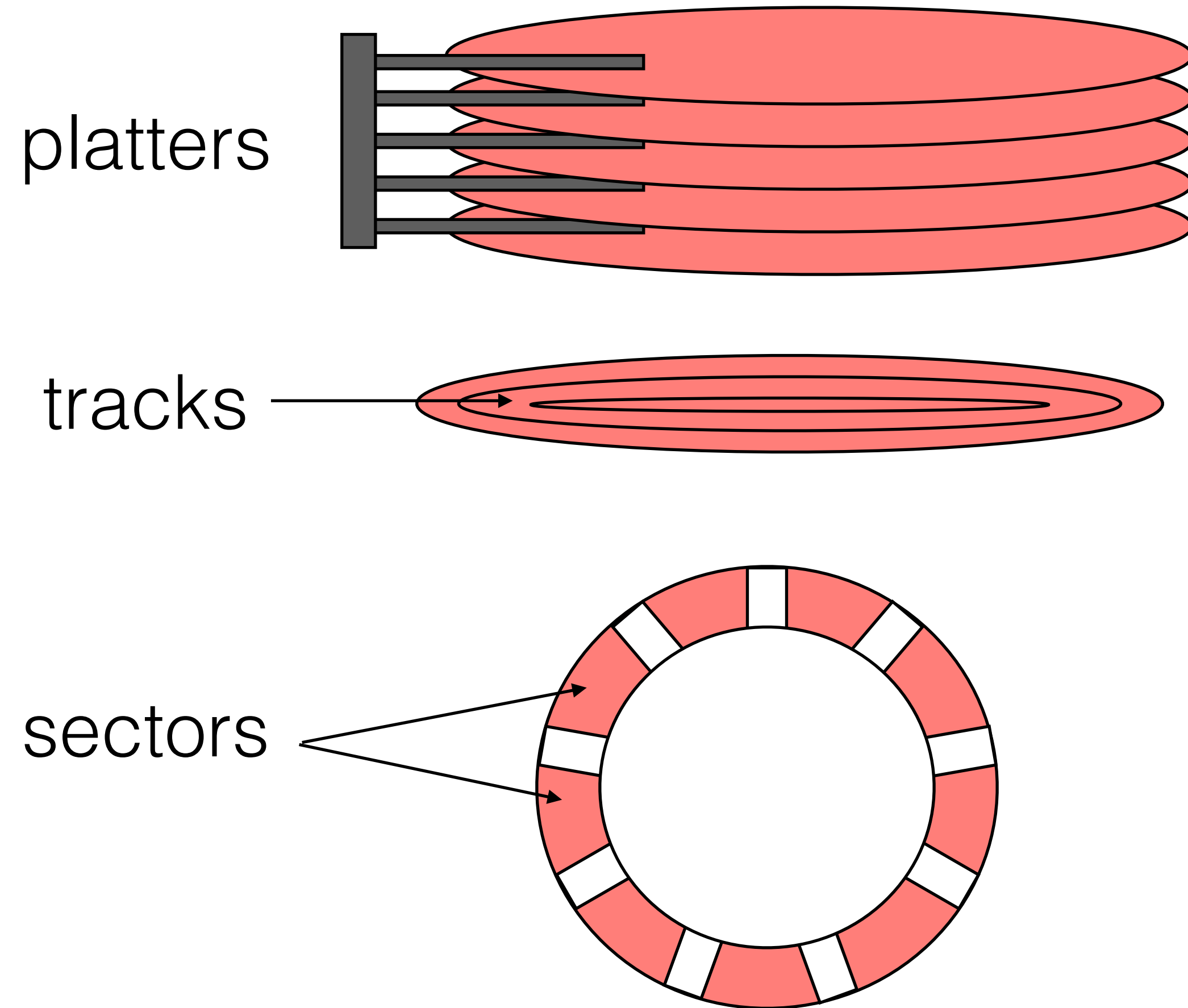


hard disk drives (HDDs)
common in datacenters



solid state drives (SSDs)
common in personal computers

the **disk** is often the main bottleneck in reading/writing stored data



hard disk drives (HDDs)

common in datacenters

example HDD specs (Hitachi 7K400)

capacity: 400GB

number of platters: 5

number of heads: 10

number of sectors per track: 567-1170

number of bytes per sector: 512

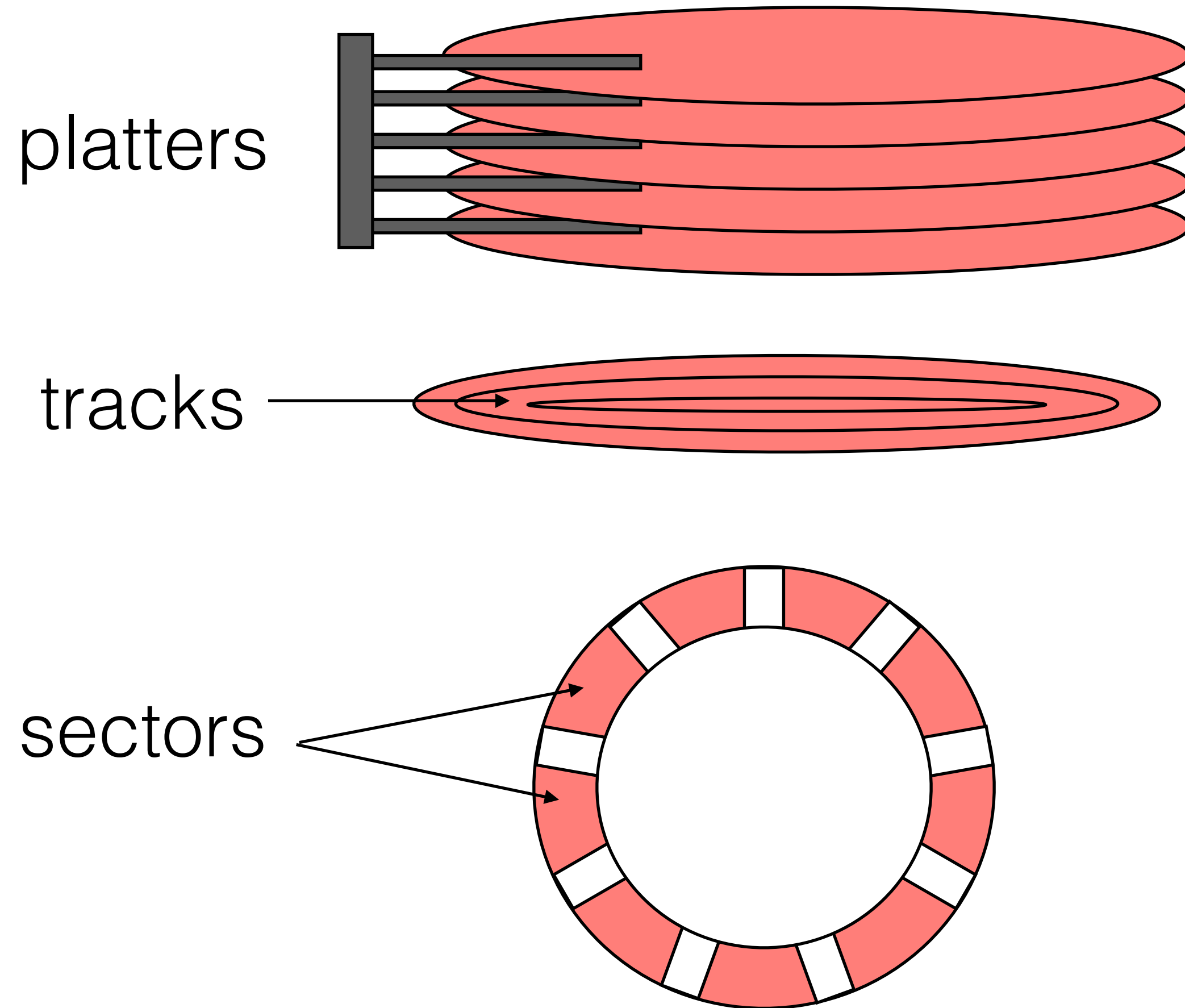
time for one revolution: 8.3ms

average read seek time: 8.2ms

average write seek time: 9.2ms

read/write throughput: 35-62MB/sec

the **disk** is often the main bottleneck in reading/writing stored data



hard disk drives (HDDs)

common in datacenters

example HDD specs (Hitachi 7K400)

capacity: 400GB

number of platters: 5

number of heads: 10

number of sectors per track: 567-1170

number of bytes per sector: 512

time for one revolution: 8.3ms

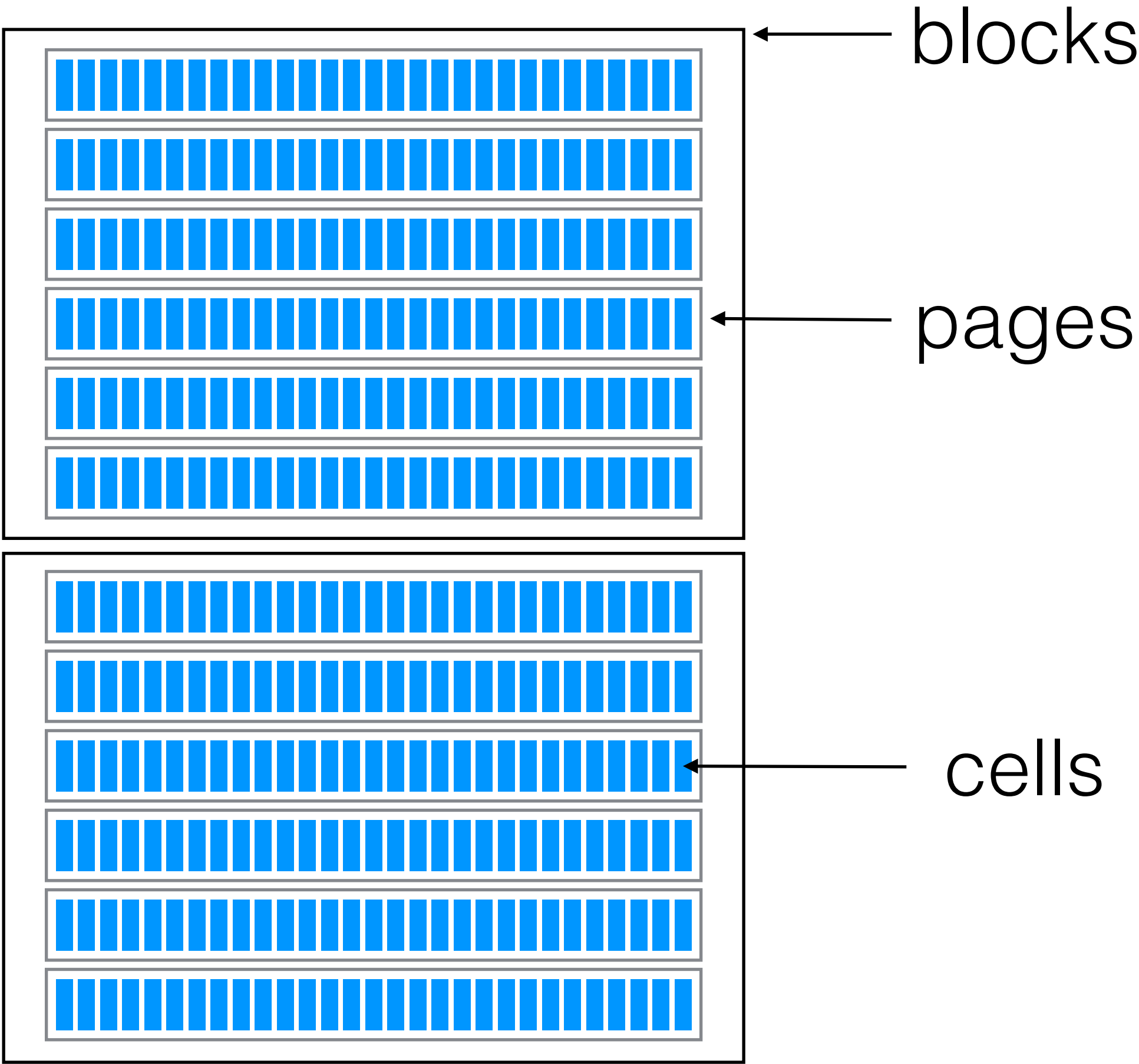
average read seek time: 8.2ms

average write seek time: 9.2ms

read/write throughput: 35-62MB/sec

**since so much time of reading/writing is spent
seeking, avoiding random access can improve
performance**

the **disk** is often the main bottleneck in reading/writing stored data



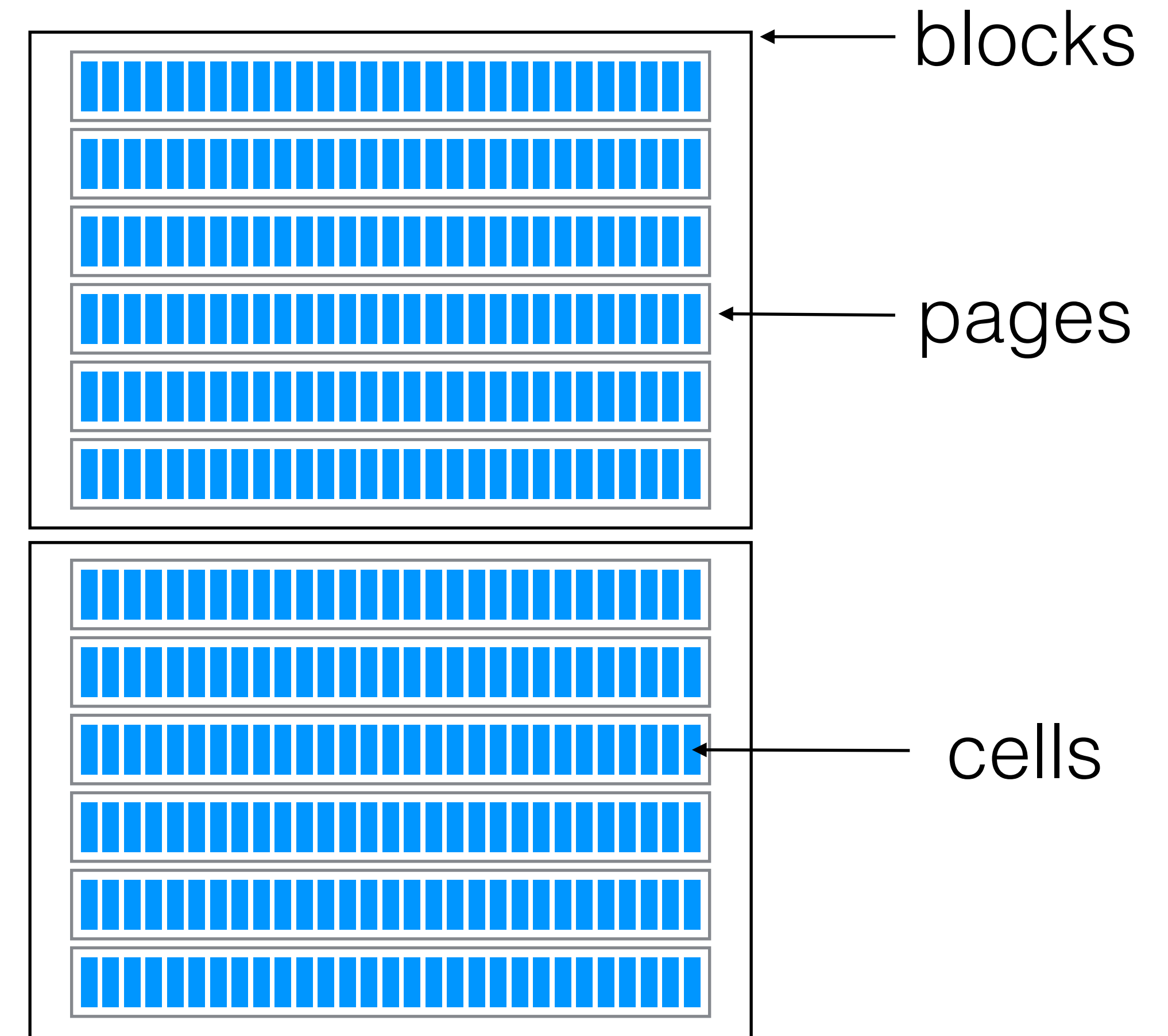
solid state drives (SSDs)

common in personal computers

the **disk** is often the main bottleneck in reading/writing stored data

since SSDs don't involve moving parts, disk seeks are not a concern (this is one of the reasons SSDs are so much faster than HDDs)

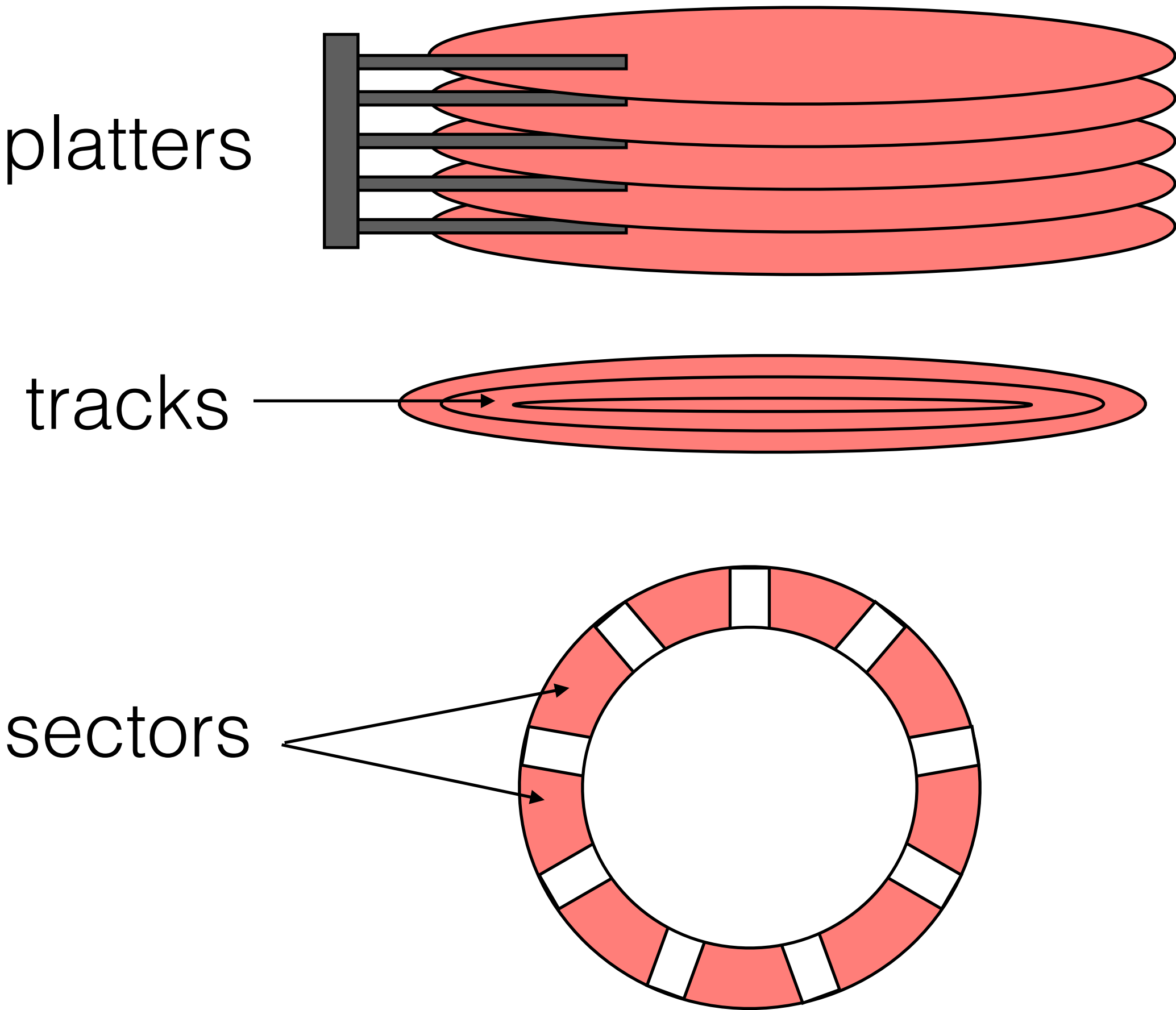
however, because of how writes are done, the SSD controller is careful about how it writes new data and makes changes to existing data



solid state drives (SSDs)

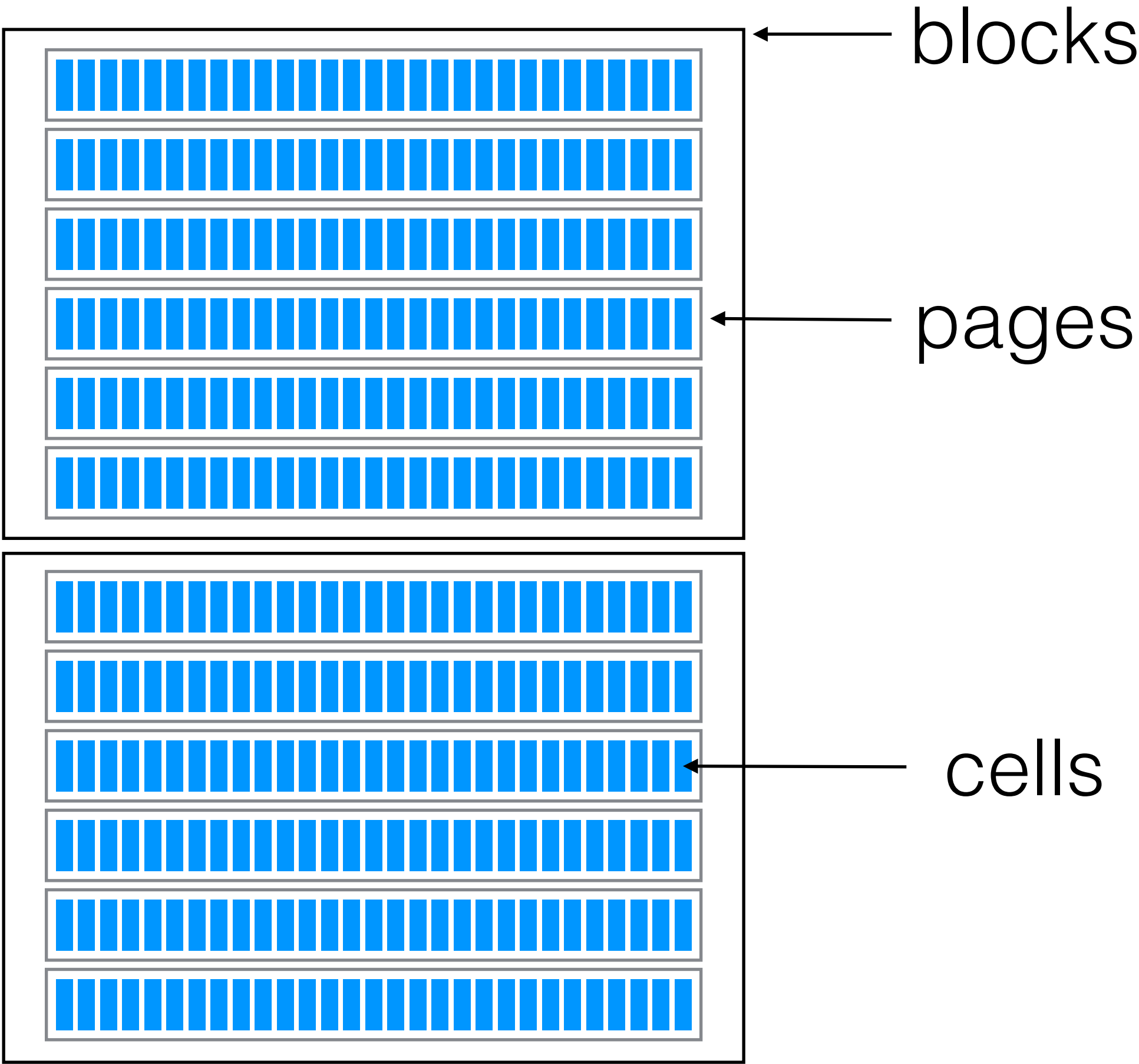
common in personal computers

the **disk** is often the main bottleneck in reading/writing stored data



hard disk drives (HDDs)

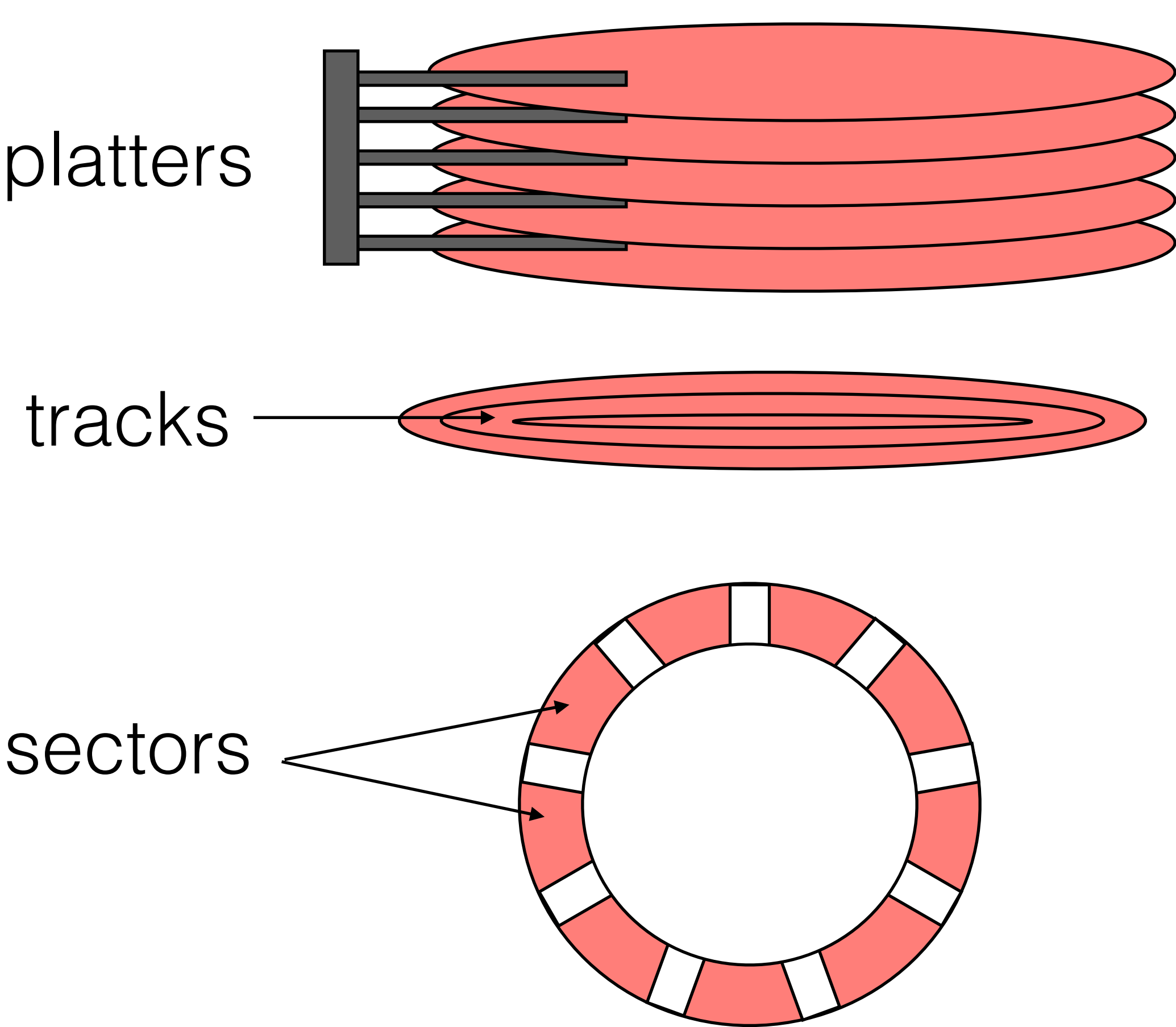
common in datacenters



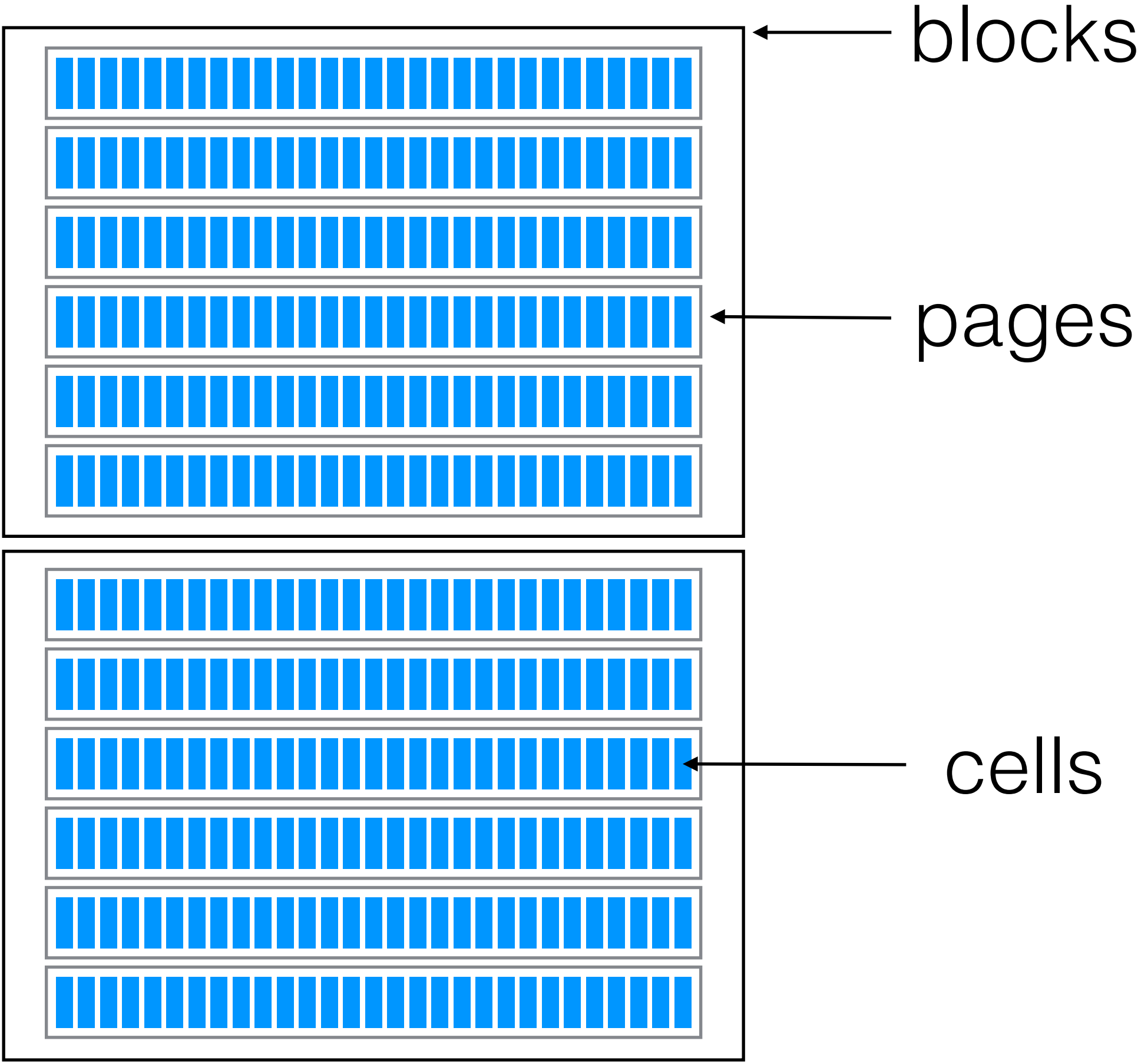
solid state drives (SSDs)

common in personal computers

so far, we have always imagined reading/writing data via the abstraction of a **filesystem**.
does that abstraction ever get in the way?



hard disk drives (HDDs)
common in datacenters



solid state drives (SSDs)
common in personal computers

so far, we have always imagined reading/writing data via the abstraction of a **filesystem**.
does that abstraction ever get in the way?

so far, we have always imagined reading/writing data via the abstraction of a **filesystem**.
does that abstraction ever get in the way?

first name	last name	building	room
katrina	lacurts	38	476
karen	sollins	32	G534
sam	madden	32	G983
...			

so far, we have always imagined reading/writing data via the abstraction of a **filesystem**.
does that abstraction ever get in the way?

first name	last name	building	room
katrina	lacurts	38	476
karen	sollins	32	G534
sam	madden	32	G983
...			

first name	last name	role
katrina	lacurts	lecture
karen	sollins	recitation
sam	madden	recitation
...		

so far, we have always imagined reading/writing data via the abstraction of a **filesystem**.
does that abstraction ever get in the way?

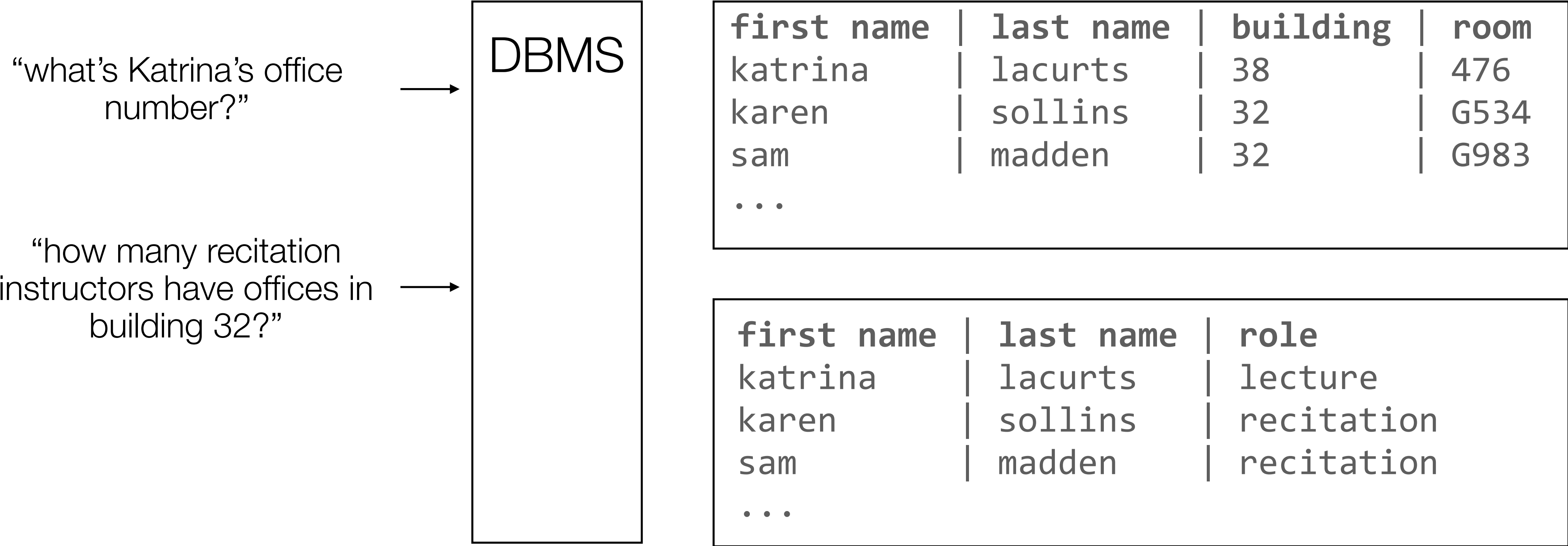
“what’s Katrina’s office
number?”

first name	last name	building	room
katrina	lacurts	38	476
karen	sollins	32	G534
sam	madden	32	G983
...			

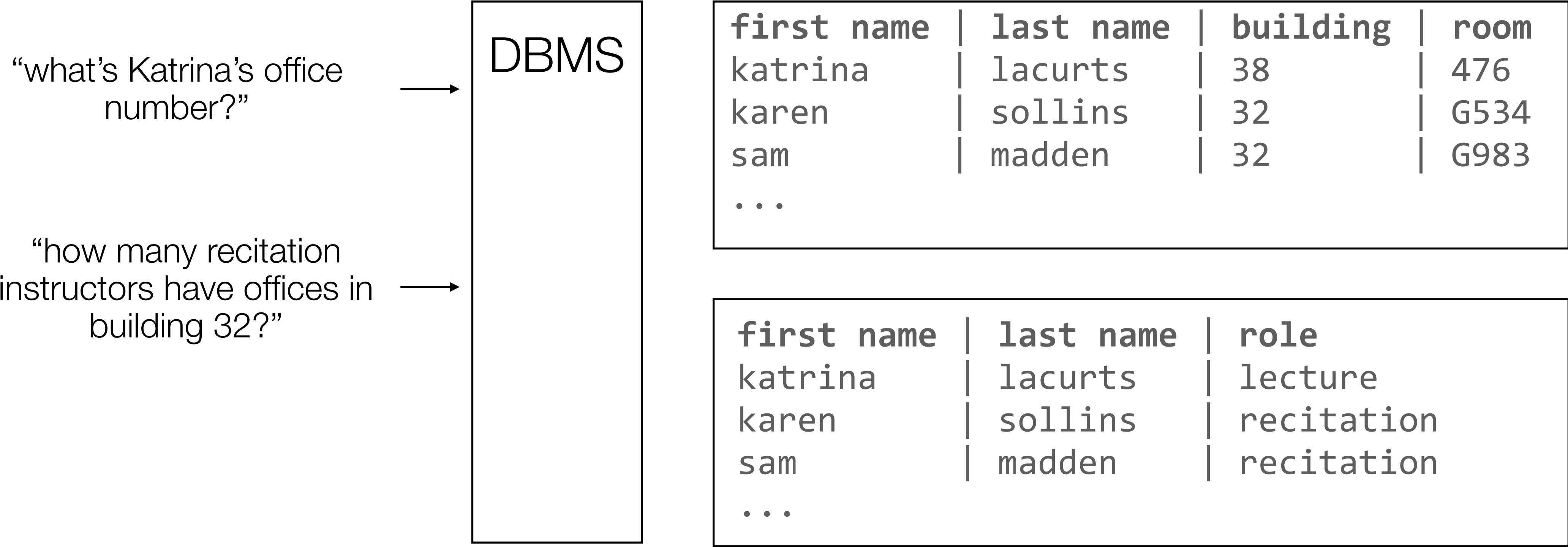
“how many recitation
instructors have offices in
building 32?”

first name	last name	role
katrina	lacurts	lecture
karen	sollins	recitation
sam	madden	recitation
...		

so far, we have always imagined reading/writing data via the abstraction of a **filesystem**.
does that abstraction ever get in the way?

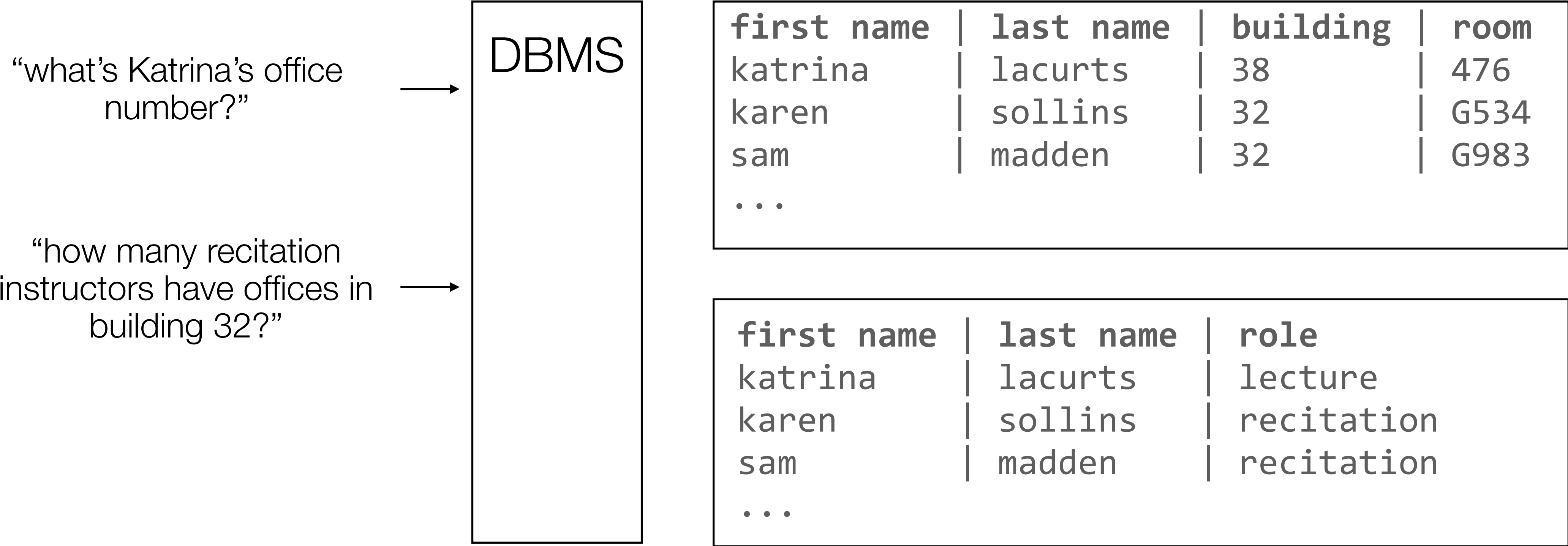


so far, we have always imagined reading/writing data via the abstraction of a **filesystem**.
does that abstraction ever get in the way?



how should the data be stored as files?
one file for everything? one file per table? per row? per column? per cell?

so far, we have always imagined reading/writing data via the abstraction of a **filesystem**.
does that abstraction ever get in the way?



the DBMS knows so much about the data and related queries that it can do a very good job at predicting which byte it needs next

it's in a good position to exploit block-level control over loading or evicting data to memory

performance is important throughout systems. we often measure throughput, latency, and utilization, and use techniques such as caching and batching to improve performance

performance is important throughout systems. we often measure throughput, latency, and utilization, and use techniques such as caching and batching to improve performance

tomorrow's recitation will continue this theme in the context of larger distributed systems

performance is important throughout systems. we often measure throughput, latency, and utilization, and use techniques such as caching and batching to improve performance

in reading/writing files, the **disk** is often the bottleneck. performance changes dramatically depending on the pattern of reads/writes (e.g., random vs. sequential access)

tomorrow's recitation will continue this theme in the context of larger distributed systems

performance is important throughout systems. we often measure throughput, latency, and utilization, and use techniques such as caching and batching to improve performance

tomorrow's recitation will continue this theme in the context of larger distributed systems

in reading/writing files, the **disk** is often the bottleneck. performance changes dramatically depending on the pattern of reads/writes (e.g., random vs. sequential access)

abstractions such as the **filesystem** work well in many places, but sometimes get in the way, especially when it comes to performance. **block-level control** can make sense for certain applications, such as **databases**

performance is important throughout systems. we often measure throughput, latency, and utilization, and use techniques such as caching and batching to improve performance

in reading/writing files, the **disk** is often the bottleneck. performance changes dramatically depending on the pattern of reads/writes (e.g., random vs. sequential access)

abstractions such as the **filesystem** work well in many places, but sometimes get in the way, especially when it comes to performance. **block-level control** can make sense for certain applications, such as **databases**

tomorrow's recitation will continue this theme in the context of larger distributed systems

block-level control isn't perfect for every type of database; some do just fine with filesystems