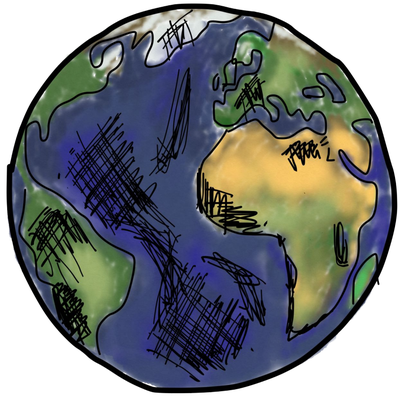
THE REVOLUTION WILL BE CONTAINERIZED:

ARCHITECTING THE INTELLIGENT APPS OF TOMORROW

William Benton • willb@redhat.com • @willb
Red Hat, Inc.

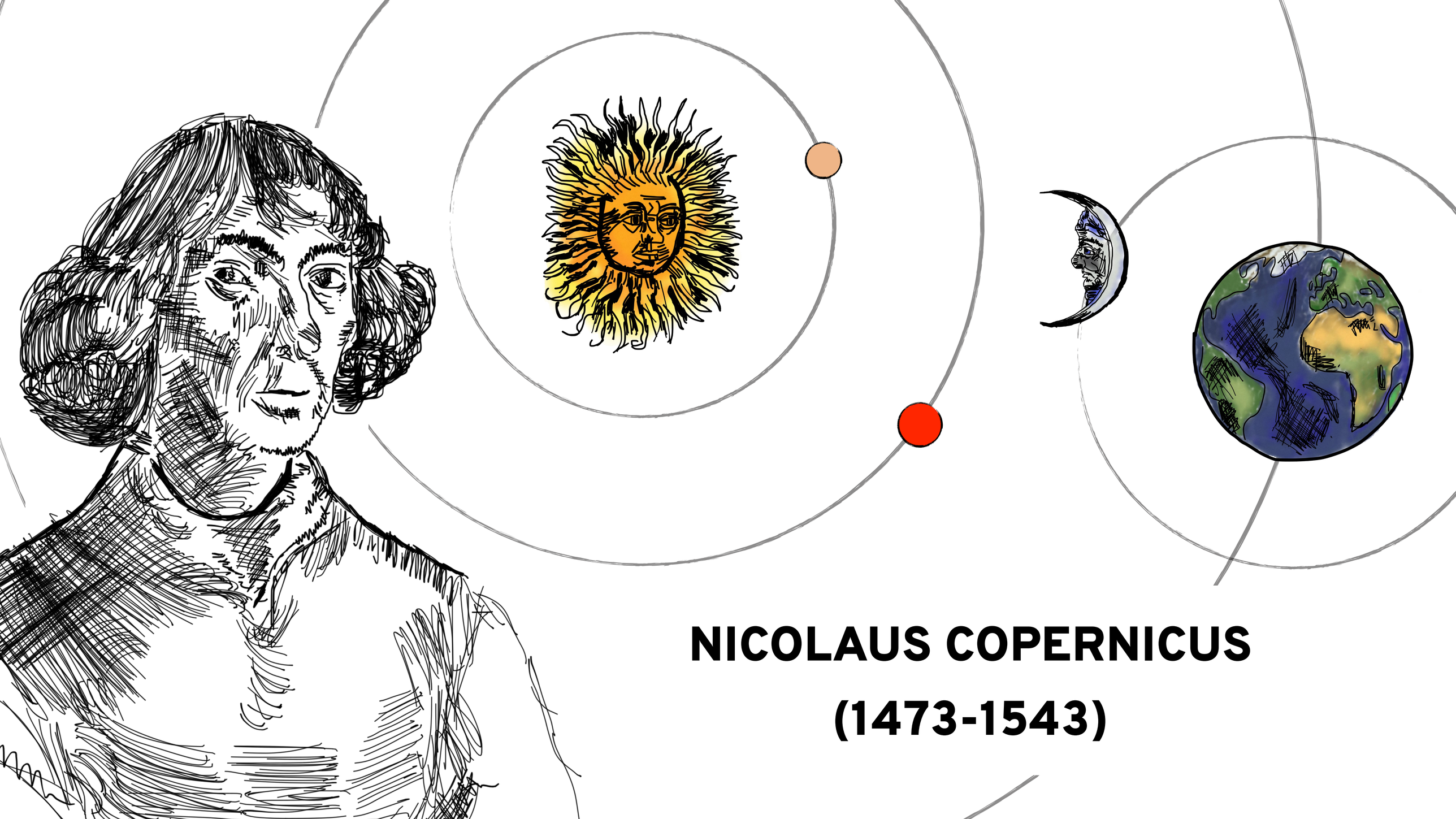


PTOLEMY of ALEXANDRIA

(ca. 100–170)

PTOLEMY of ALEXANDRIA

(ca. 100–170)

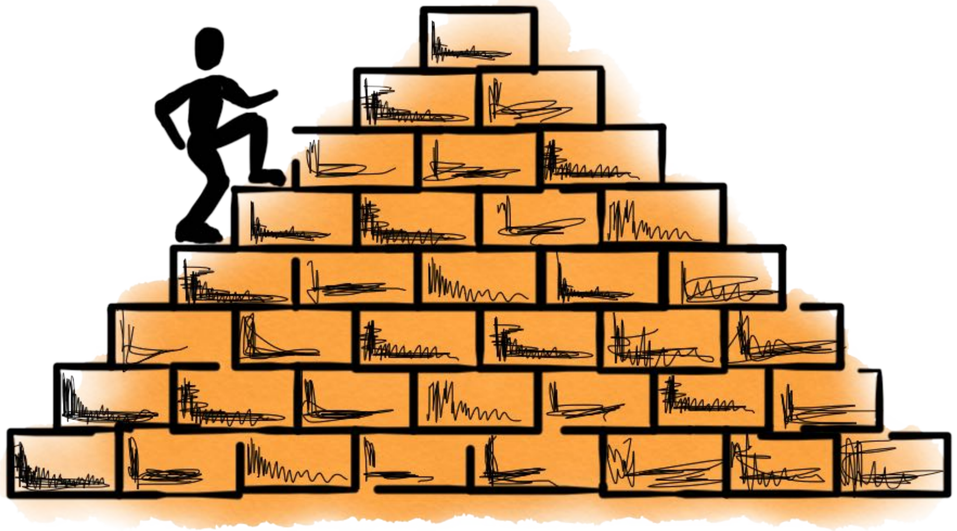


NICOLAUS COPERNICUS

(1473-1543)

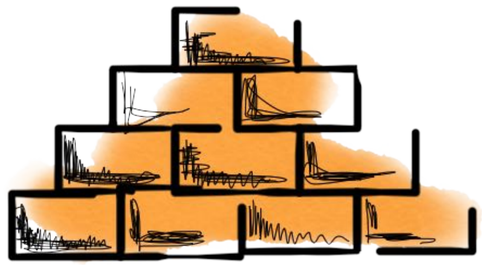
THOMAS KUHN

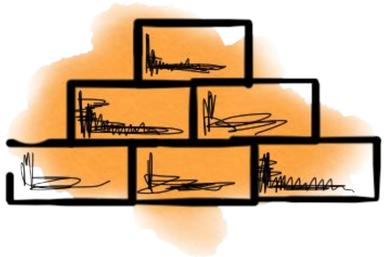
(1922–1996)



THOMAS KUHN

(1922–1996)

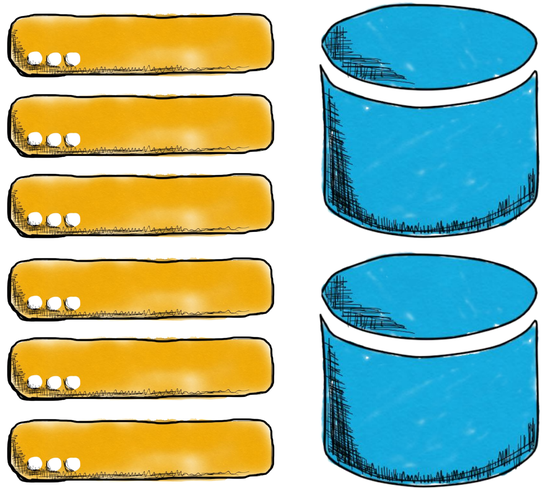




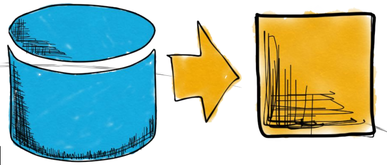
THOMAS KUHN

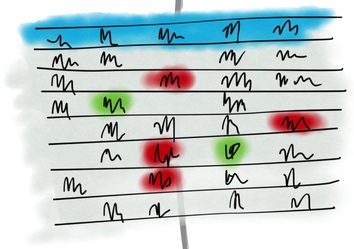
(1922–1996)

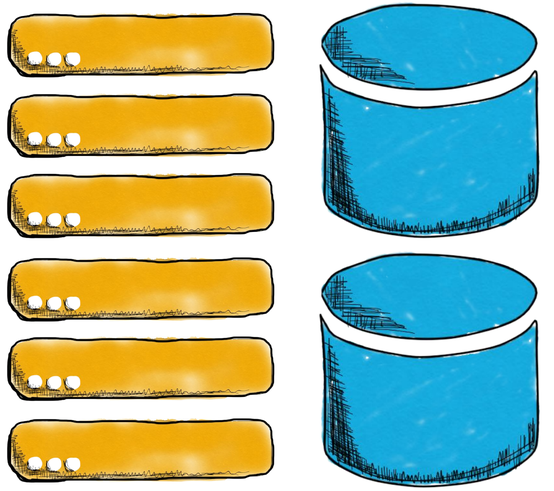
The cluster-centric model

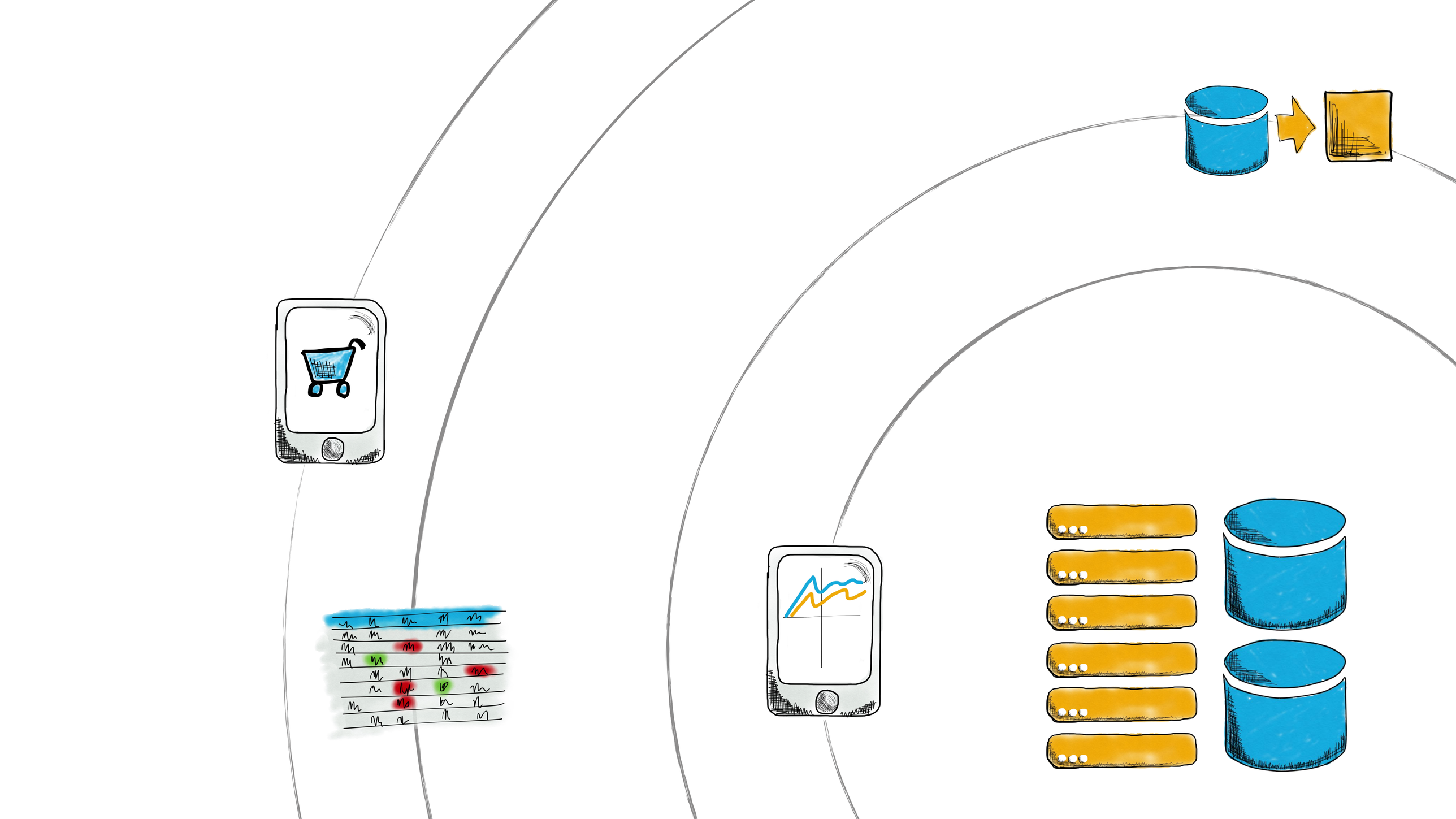


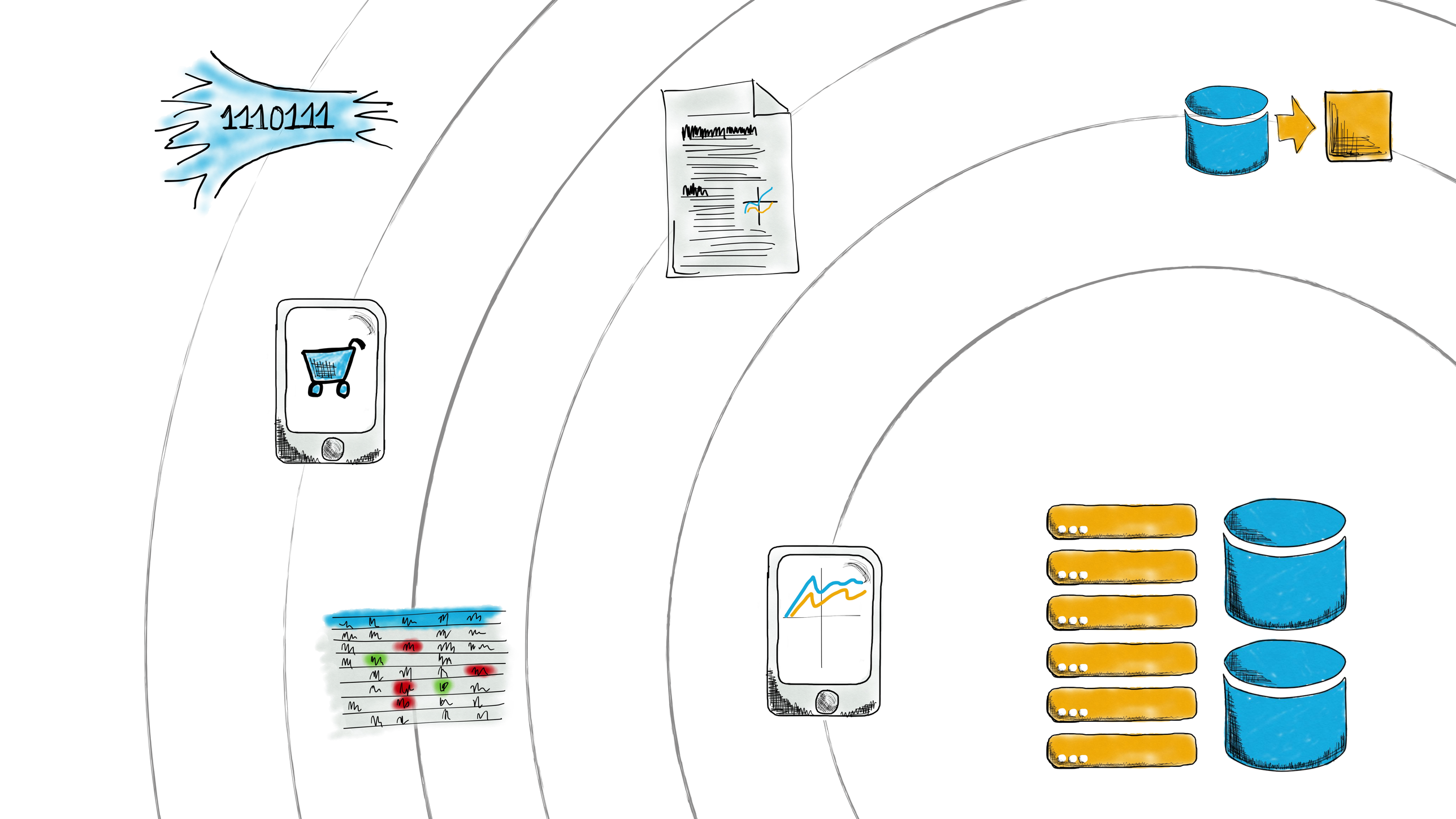
The cluster-centric model





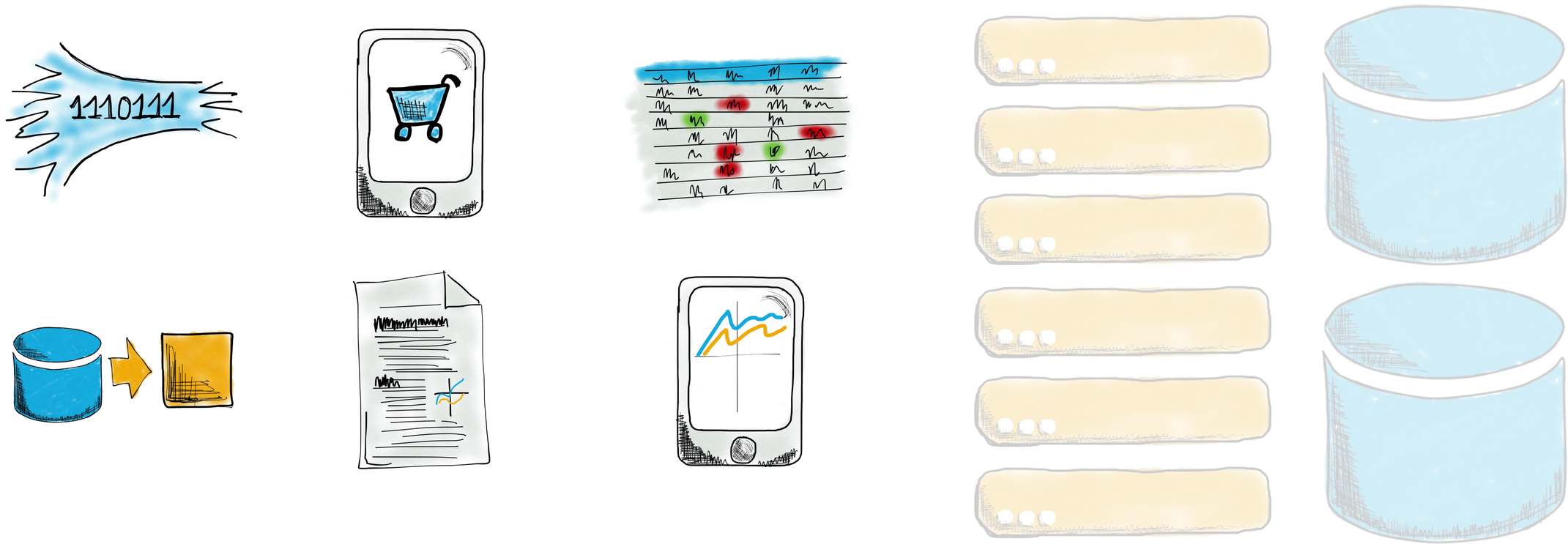





1110111

Towards an app-centric model

Towards an app-centric model



Forecast

Motivating containers

Architectures for intelligent applications

Practical concerns

Where to go from here

CONTAINERS: WHAT AND WHY

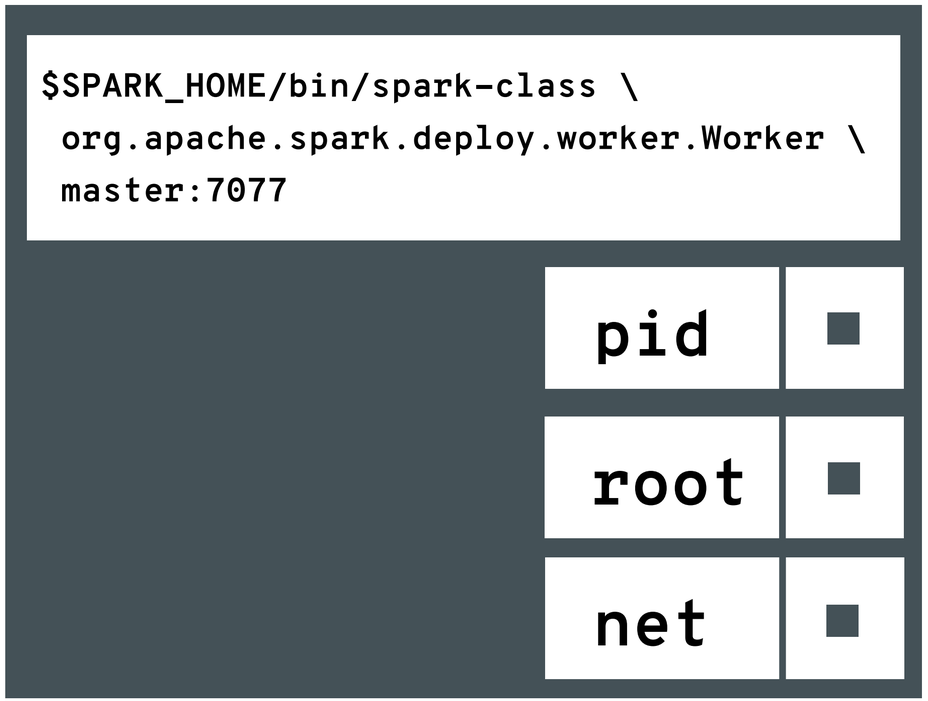
What is a container?

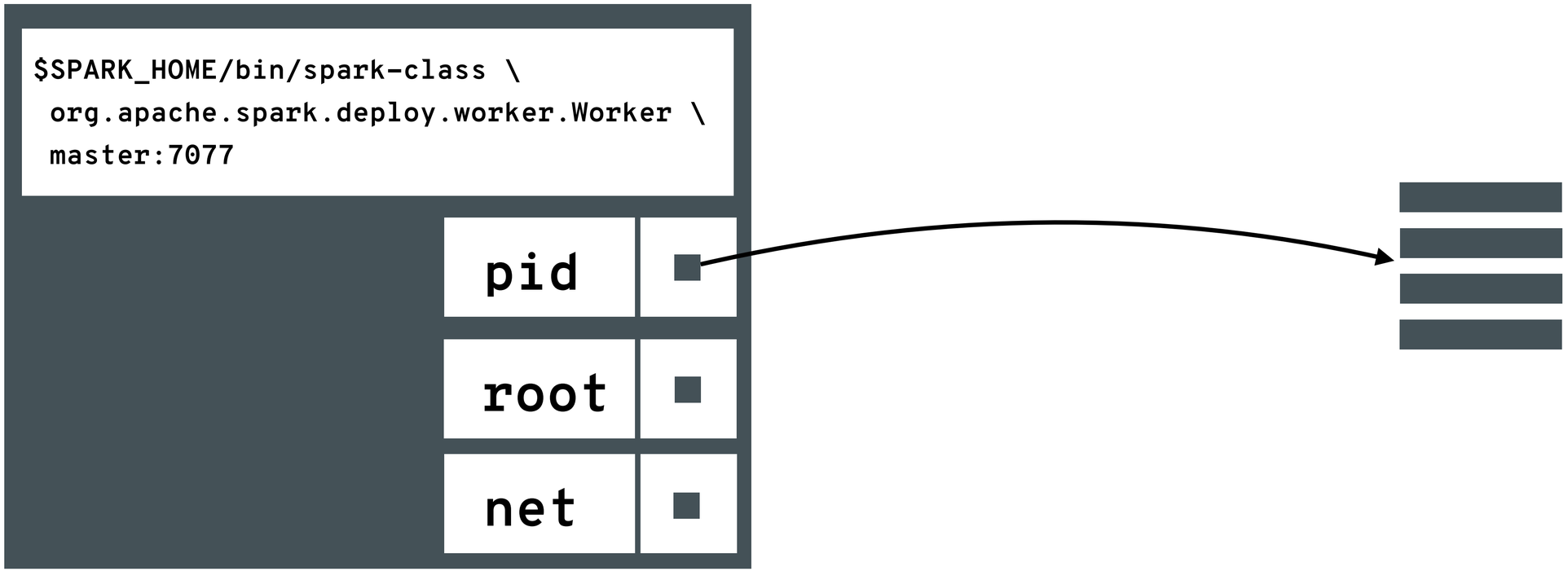
What is a container?

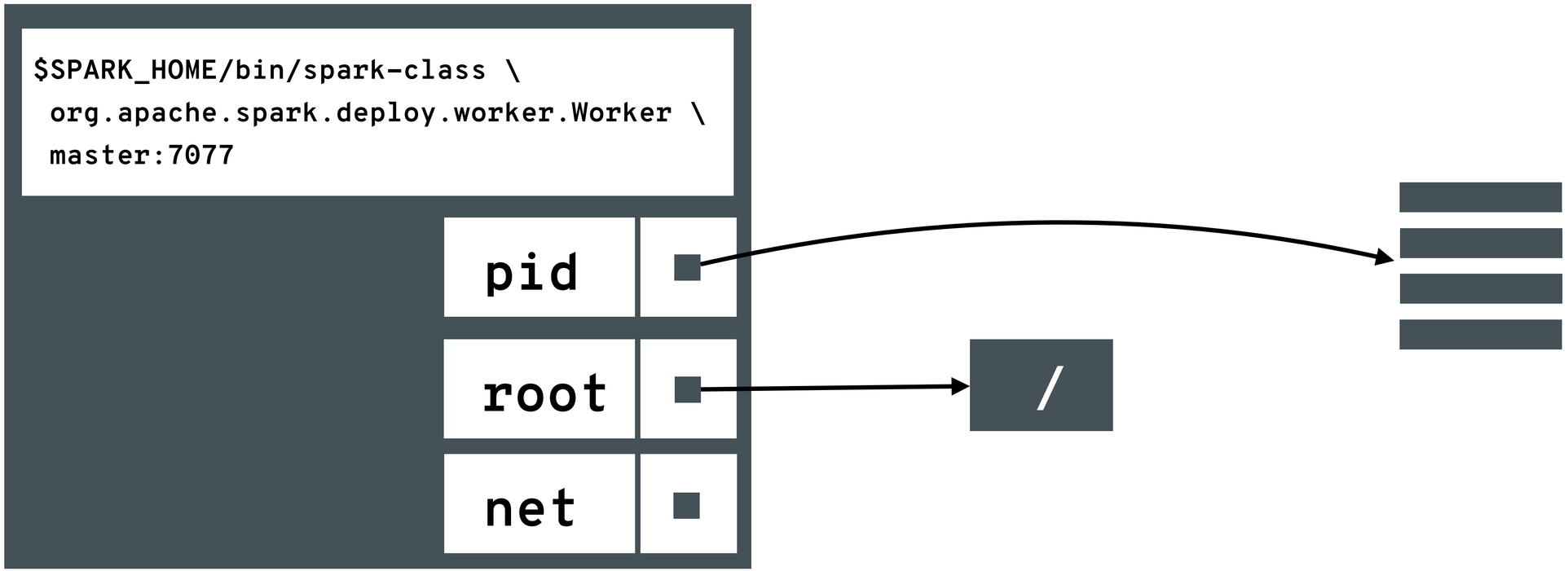
...a lightweight VM?

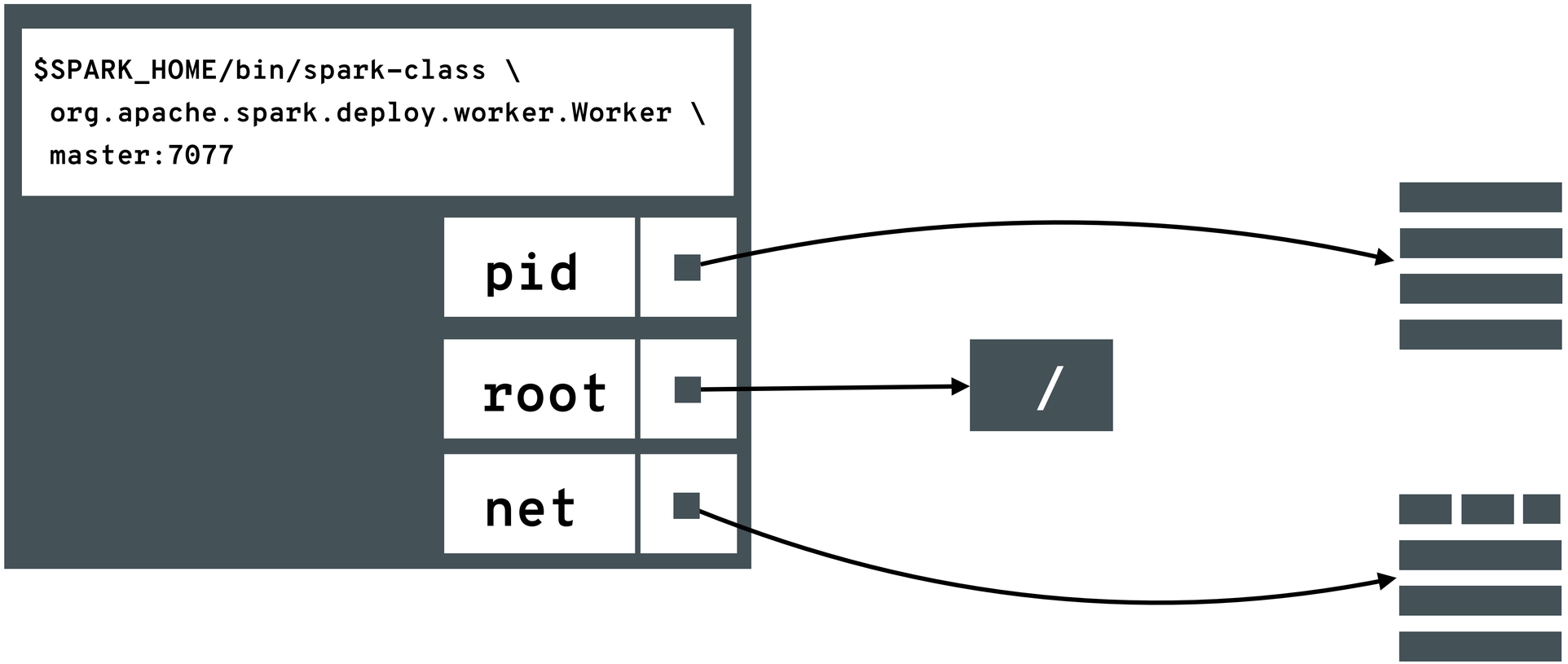
...a way to totally isolate applications?

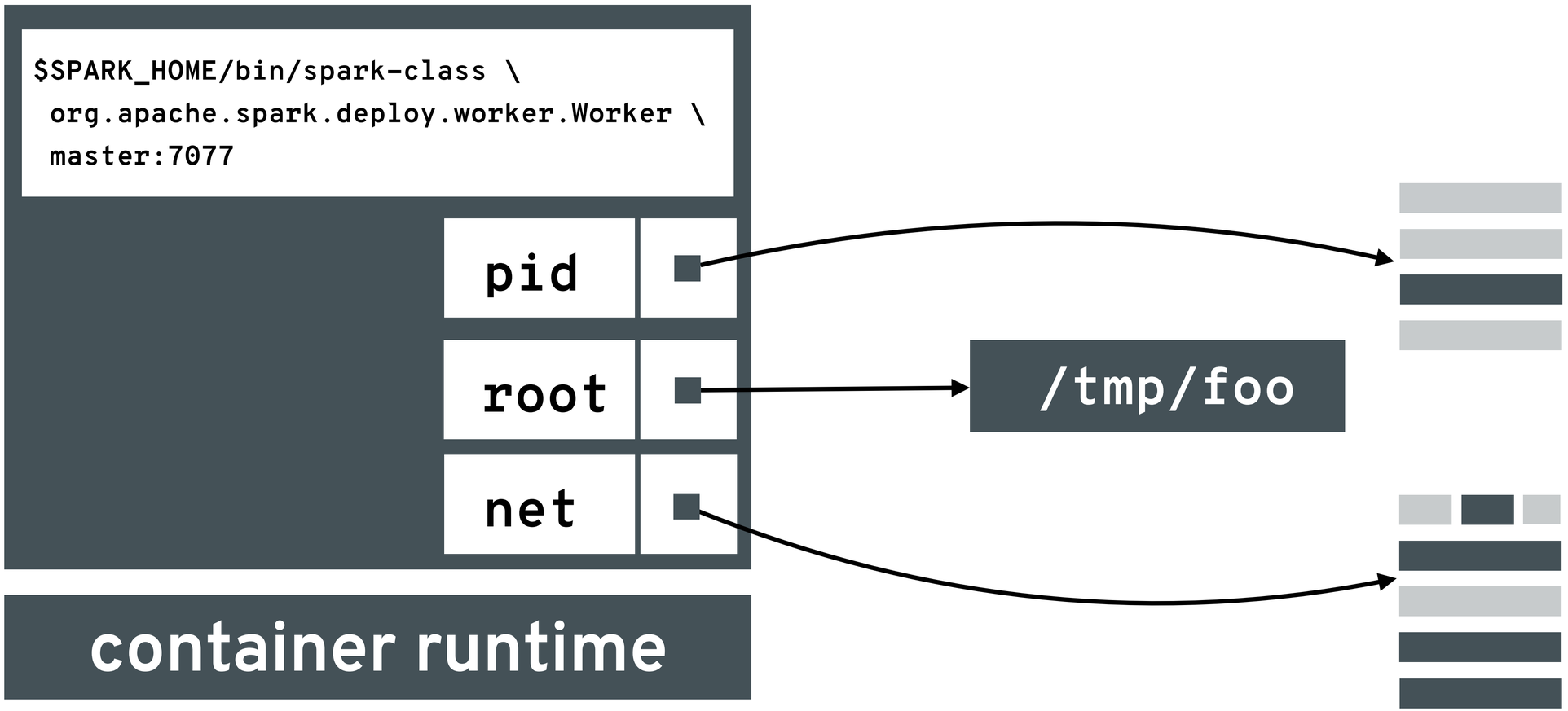
...a packaging format for a container runtime or orchestration platform?

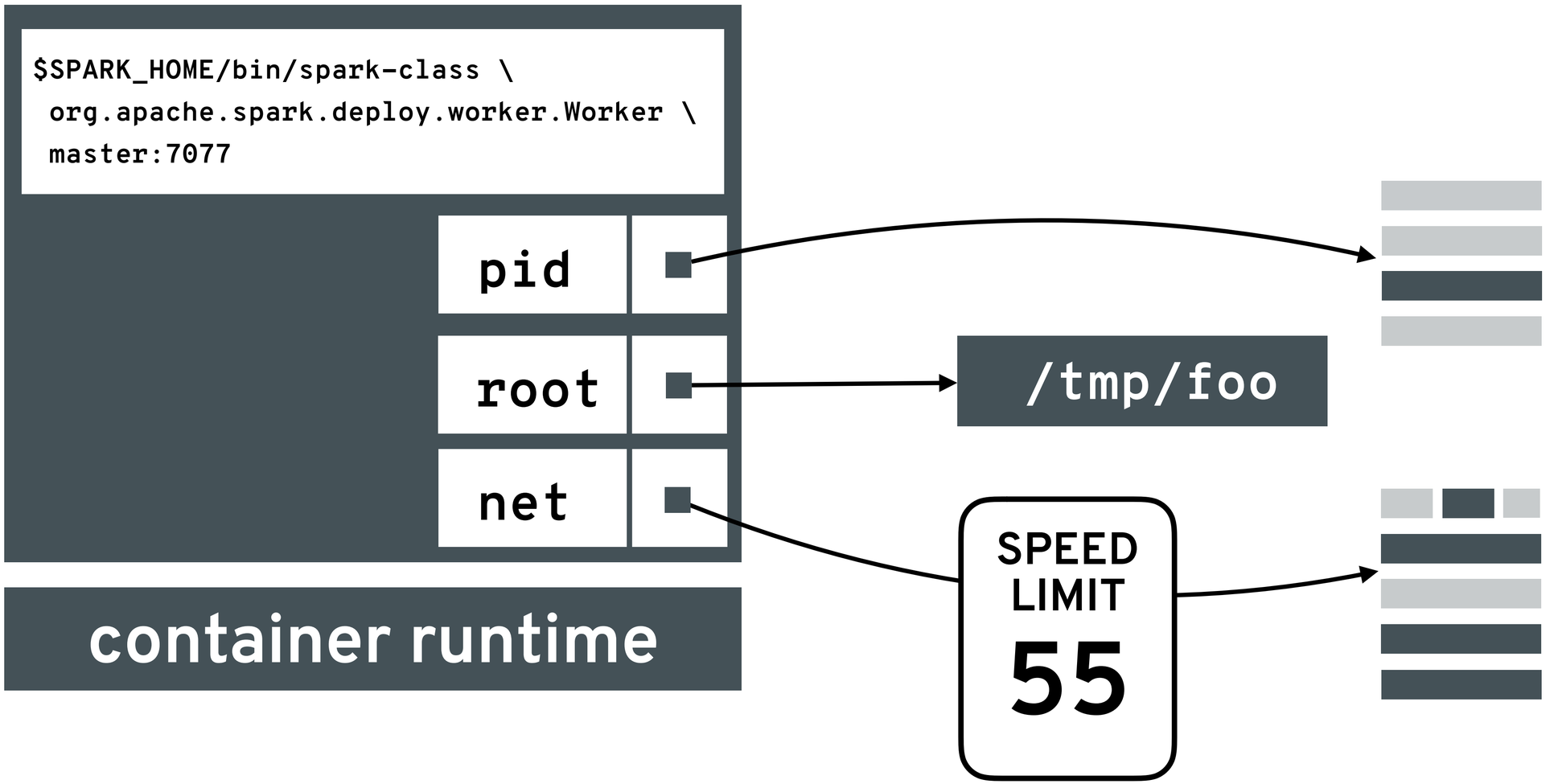

$SPARK_HOME/bin/spark-class \
org.apache.spark.deploy.worker.Worker \
master:7077
pid
root
net

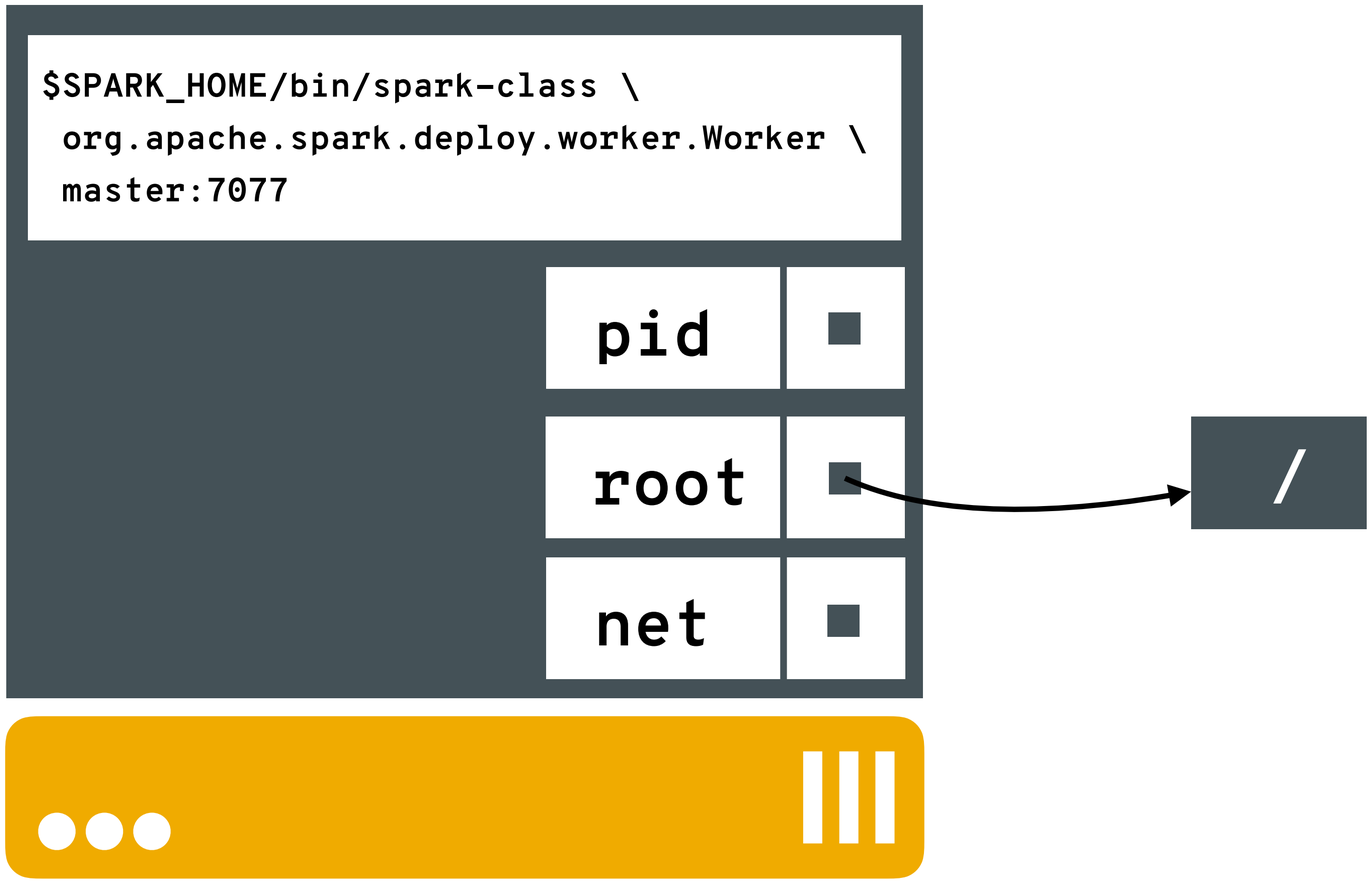

$SPARK_HOME/bin/spark-class \
org.apache.spark.deploy.worker.Worker \
master:7077
pid
root
net

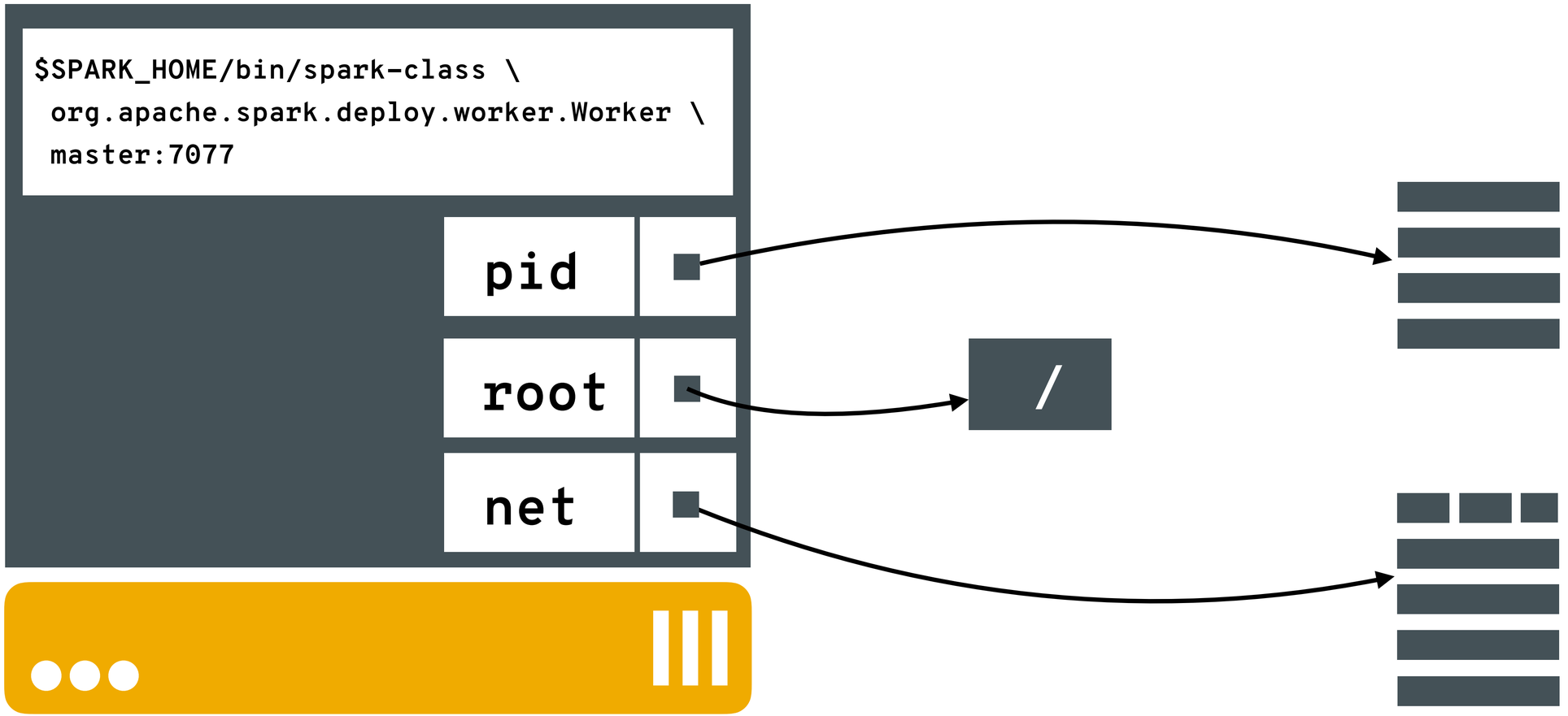

$SPARK_HOME/bin/spark-class \
org.apache.spark.deploy.worker.Worker \
master:7077
pid
root
net
/


$SPARK_HOME/bin/spark-class \
org.apache.spark.deploy.worker.Worker \
master:7077
pid
root
net
/


$SPARK_HOME/bin/spark-class \
org.apache.spark.deploy.worker.Worker \
master:7077
pid
root
net
/tmp/foo
container runtime


$SPARK_HOME/bin/spark-class \
org.apache.spark.deploy.worker.Worker \
master:7077
pid
root
net
/tmp/foo
SPEED
LIMIT
55
container runtime


$SPARK_HOME/bin/spark-class \
org.apache.spark.deploy.worker.Worker \
master:7077
pid
root
net
/


$SPARK_HOME/bin/spark-class \
org.apache.spark.deploy.worker.Worker \
master:7077
pid
root
net
/

What is a container?

...a lightweight VM?

...a way to totally isolate applications?

...a packaging format for a container runtime or orchestration platform?

What is a container?

...a lightweight means to address some of the same use cases as VMs.

...a way to totally isolate applications?

...a packaging format for a container runtime or orchestration platform?

What is a container?

...a lightweight means to address **some** of the same use cases as VMs.

...a way to provide **reasonable**, not **exhaustive** application isolation.

...a packaging format for a container runtime or orchestration platform?

What is a container?

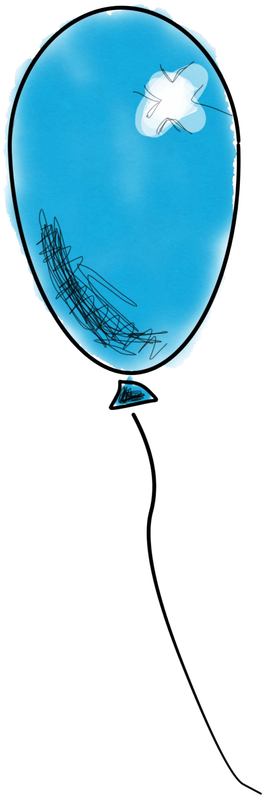
...a lightweight means to address **some** of the same use cases as VMs.

...a way to provide **reasonable**, not **exhaustive** application isolation.

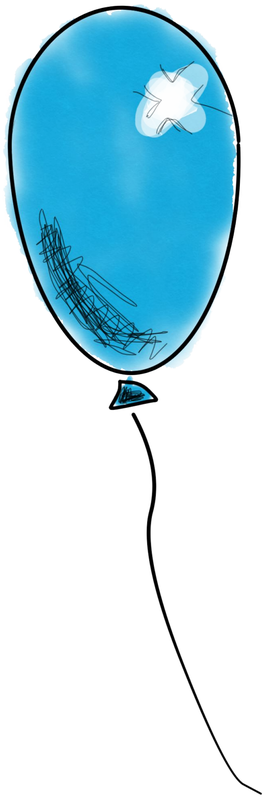
...really, just **any Linux process** with some special settings!

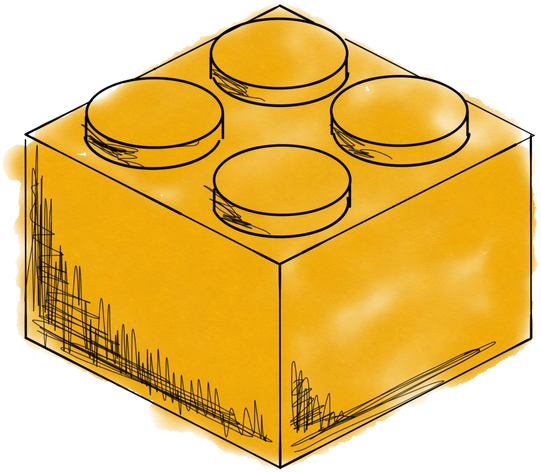
Microservice architectures

Microservice architectures

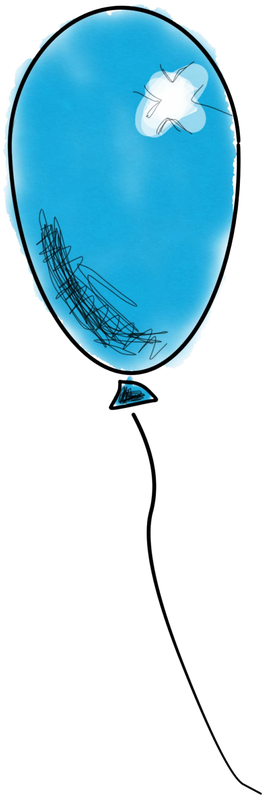


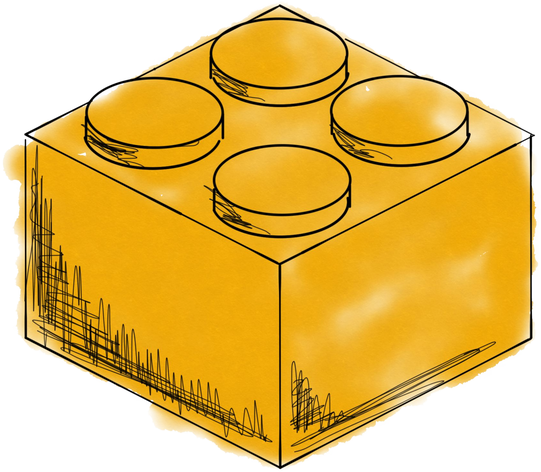
Microservice architectures

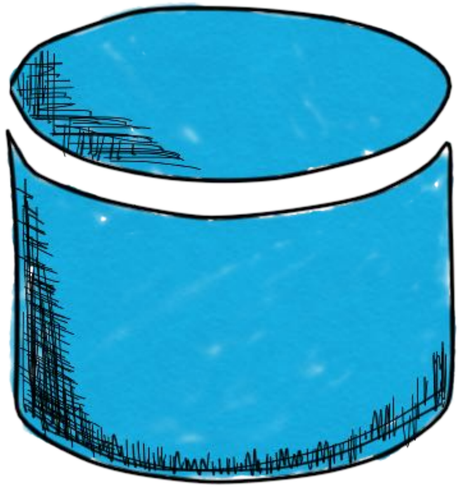




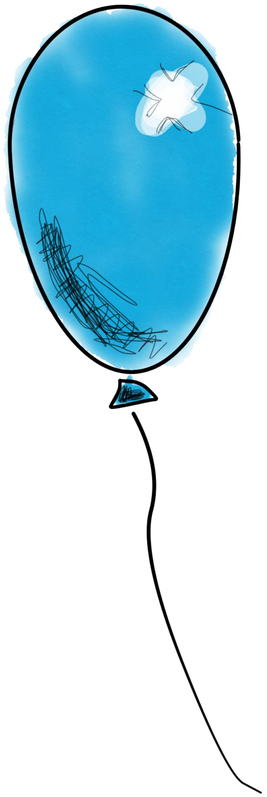
Microservice architectures

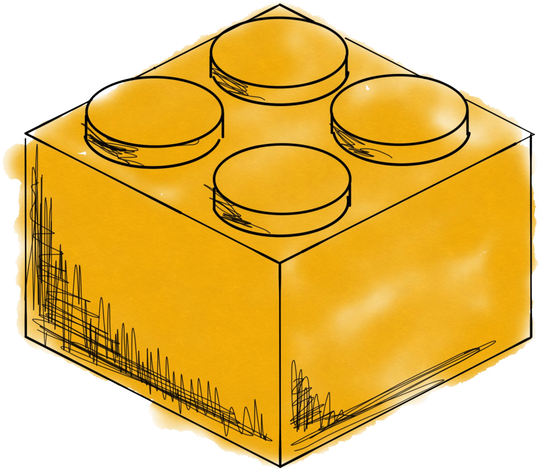






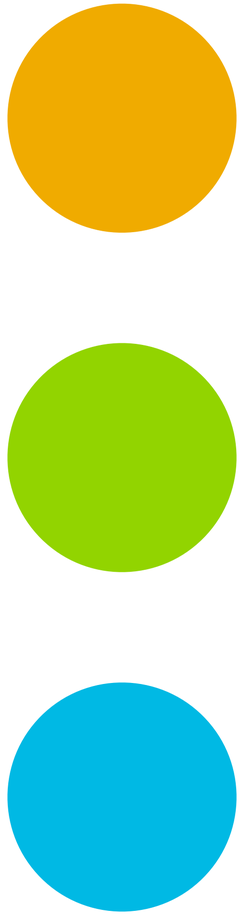
Microservice architectures



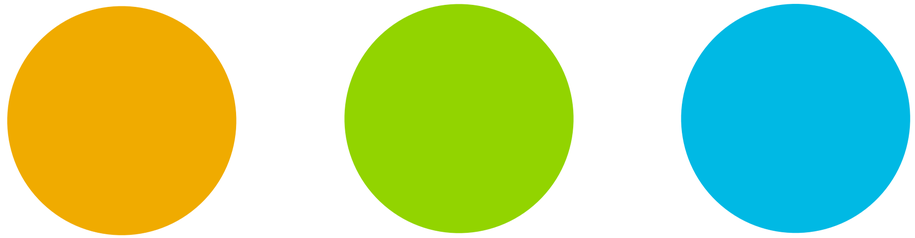


Microservices and **containers**
are a **natural fit** for one another!

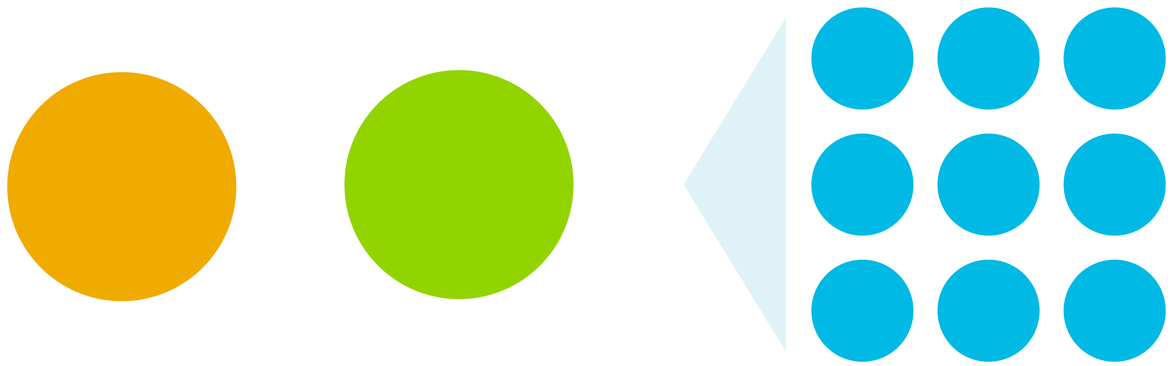
Microservices for operators



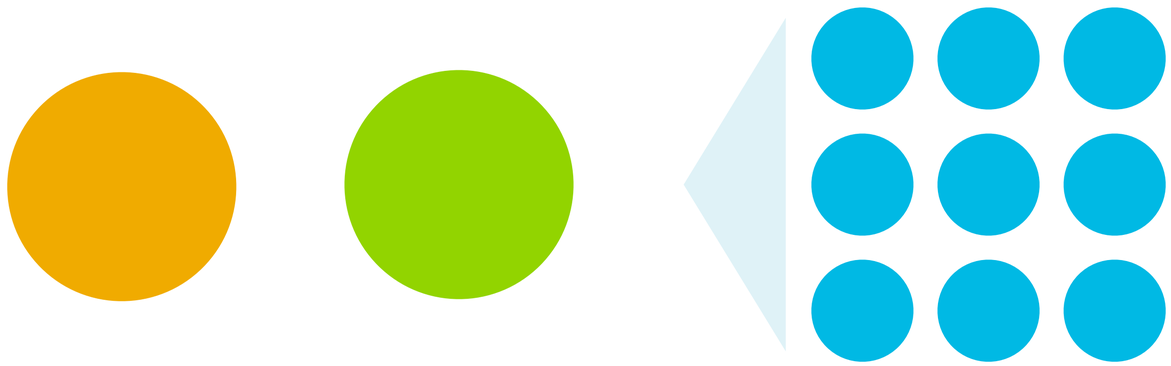
Microservices for operators



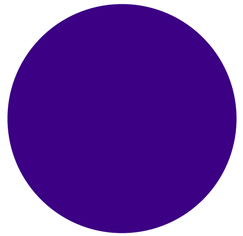
Microservices for operators



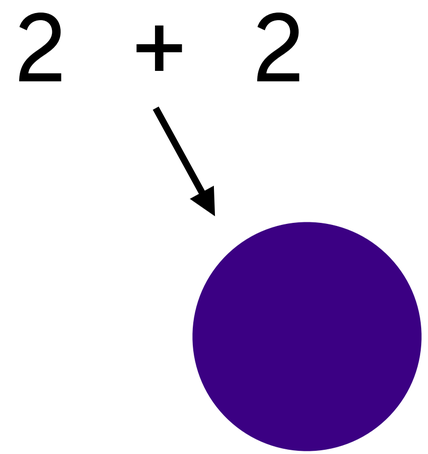
Microservices for operators



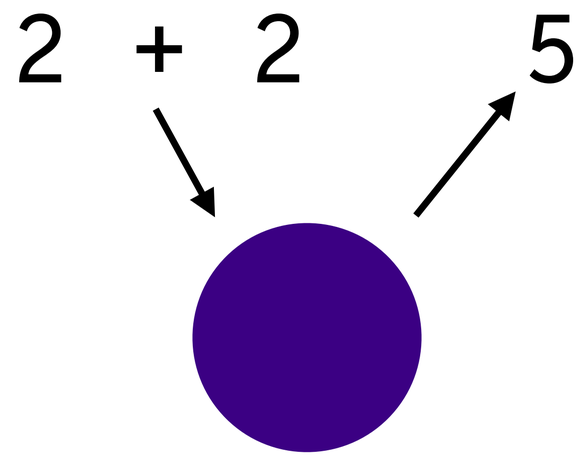
Microservices for developers



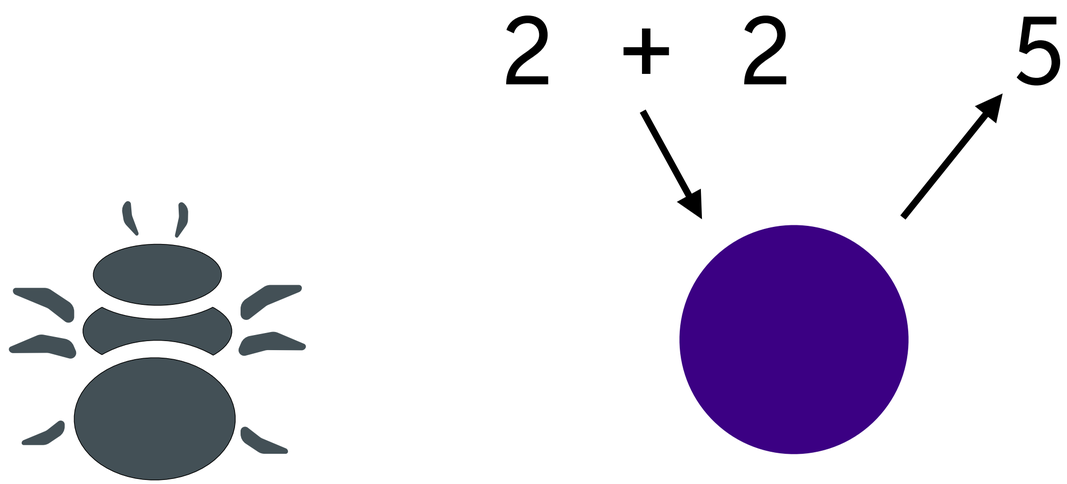
Microservices for developers



Microservices for developers



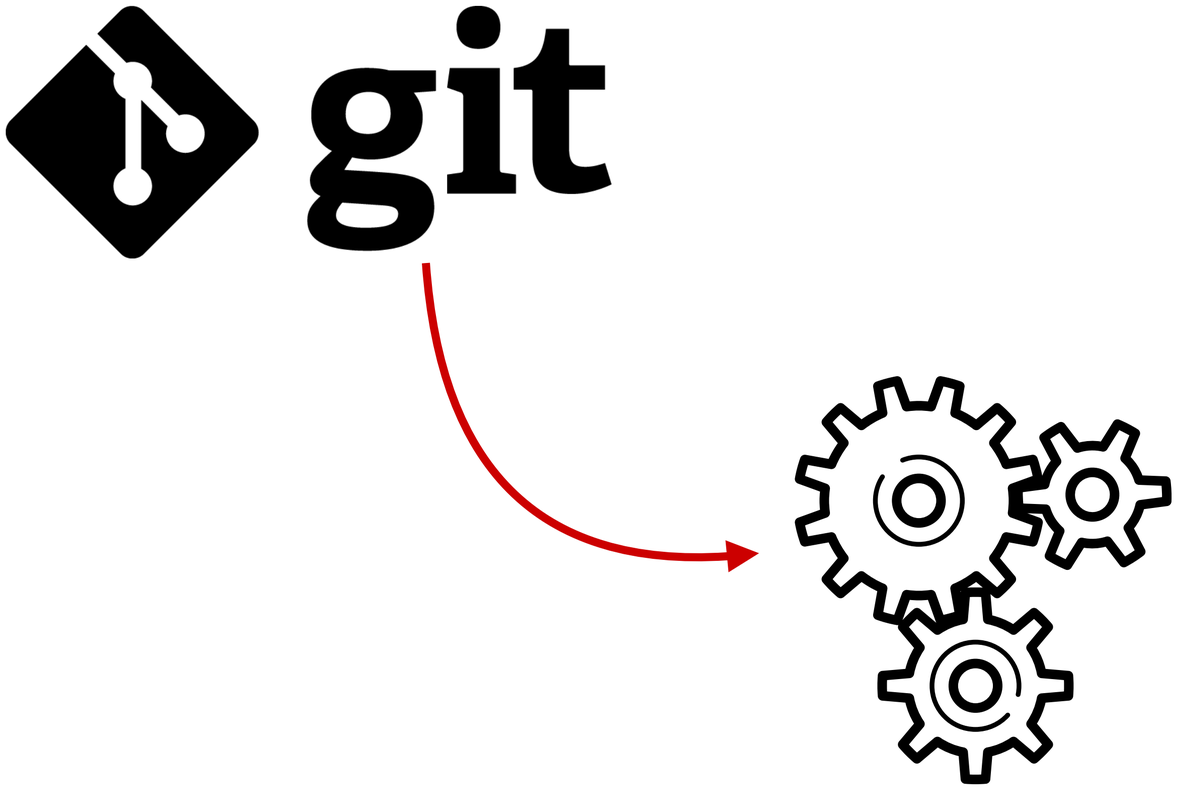
Microservices for developers



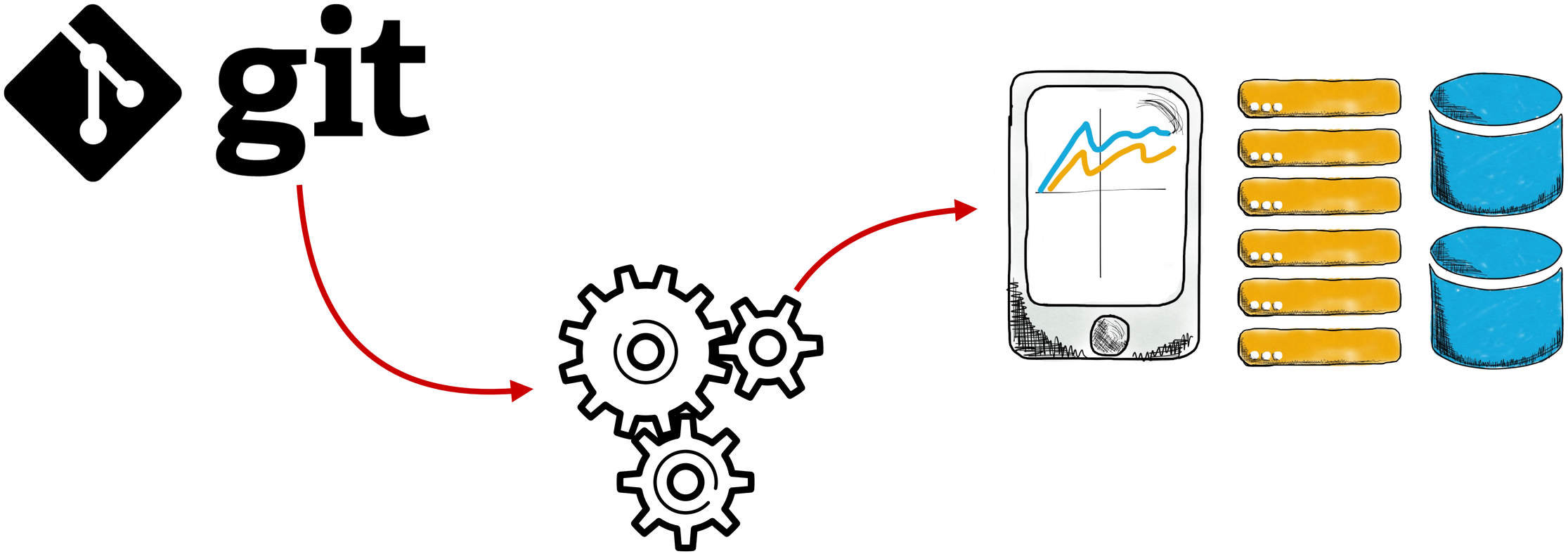
Microservices for developers

Microservices for developers

Microservices for developers

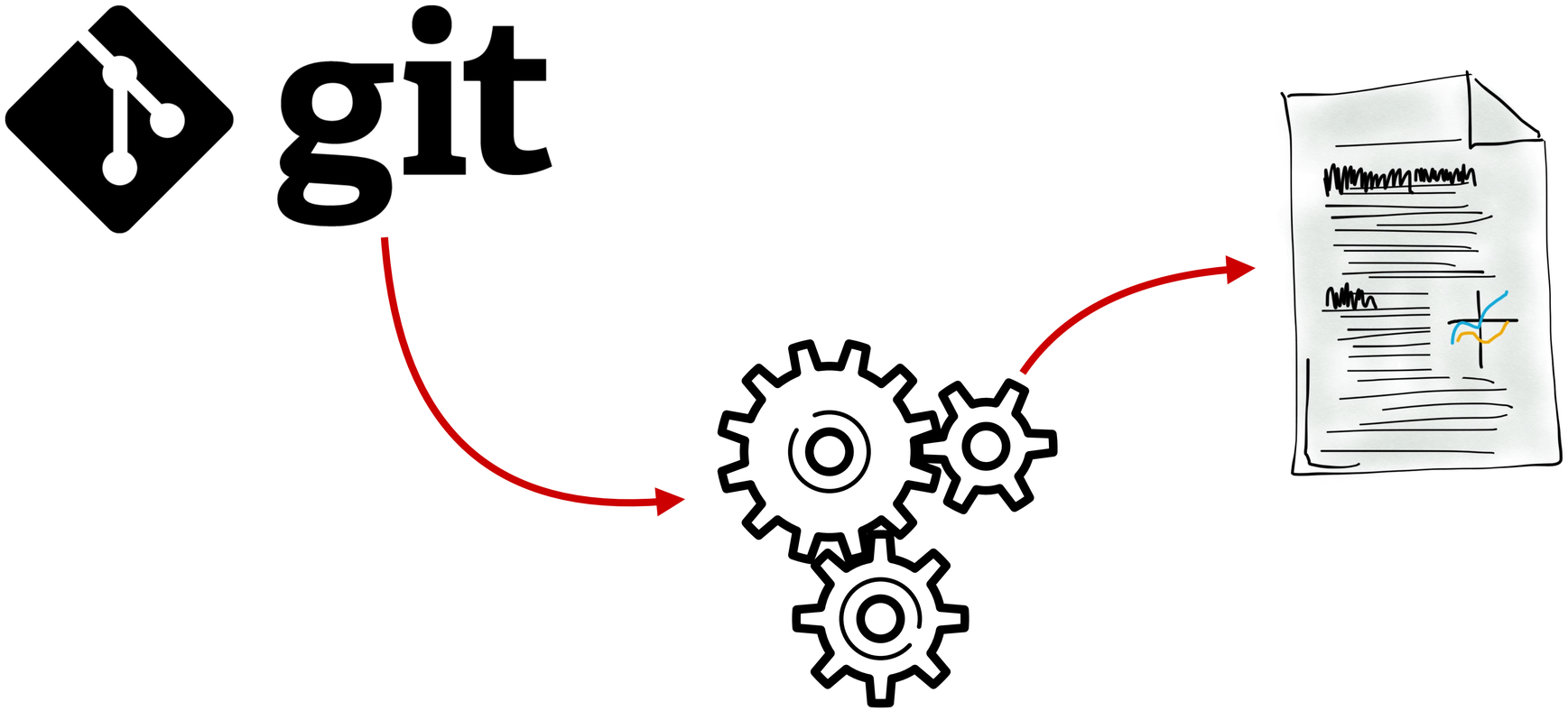


Microservices for developers



Microservices for developers

...and data scientists!



Cloud-native applications are...

Containerized

Dynamically-orchestrated

Microservice-oriented

Cloud-native applications are...

Containerized

Dynamically-orchestrated

Microservice-oriented

Cloud-native applications are...

Containerized

Dynamically-orchestrated

Microservice-oriented

Cloud-native applications are...

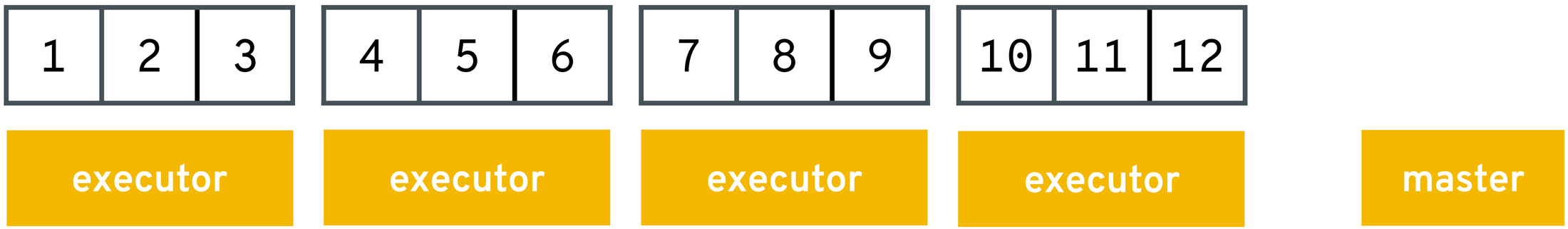
Containerized

Dynamically-orchestrated

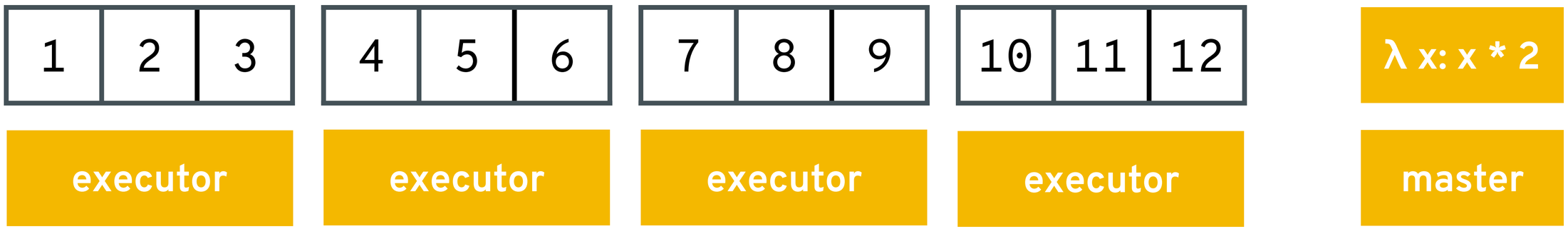
Microservice-oriented

Contemporary analytics and compute frameworks are most of the way there!

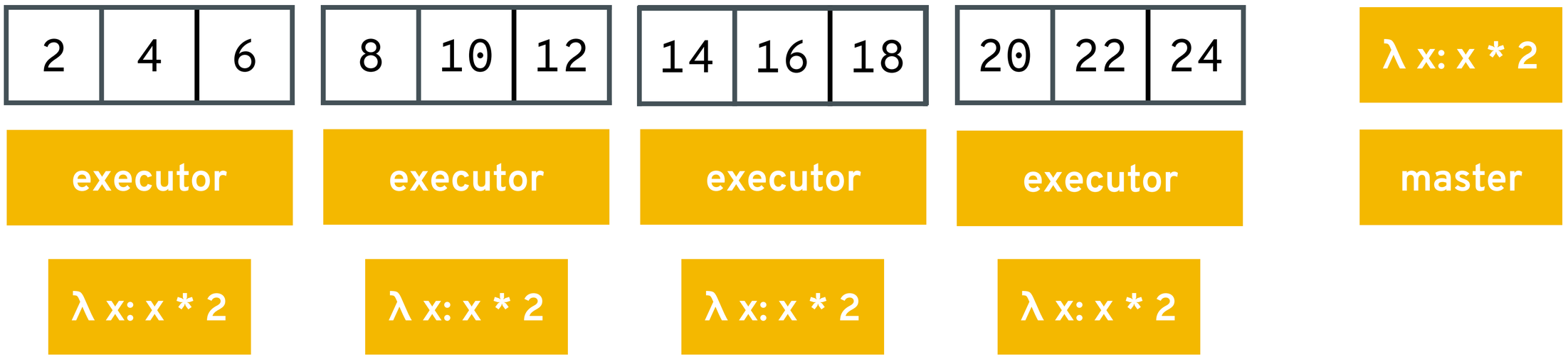
Microservices for analytics



Microservices for analytics

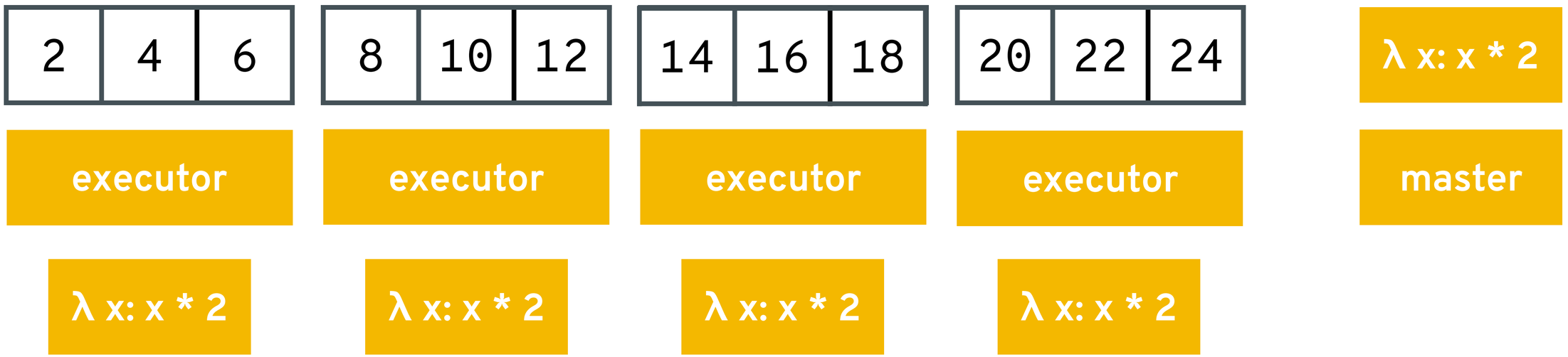


Microservices for analytics



Microservices for analytics

Microservices for analytics

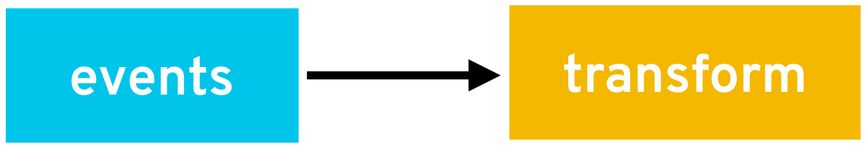


LEGACY ARCHITECTURES FOR ANALYTICS AND APPLICATIONS

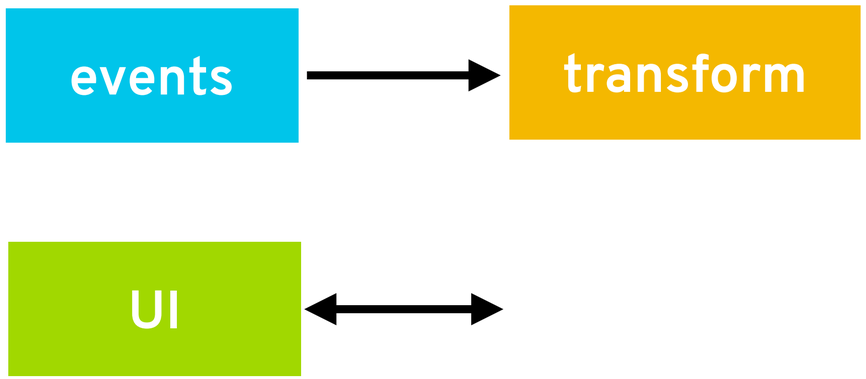
Transactions and analytics

events

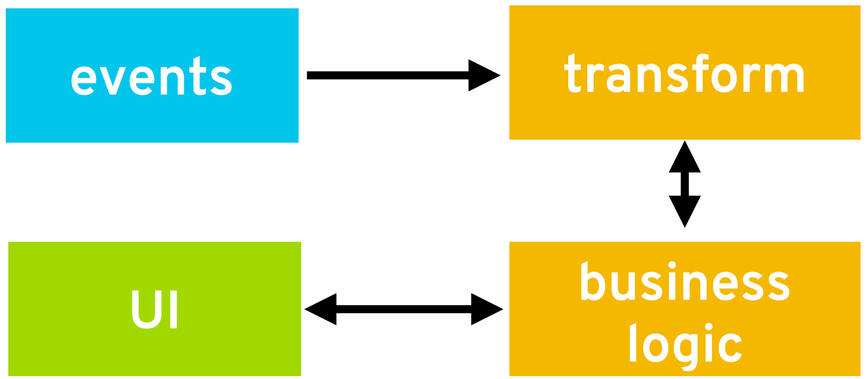
Transactions and analytics



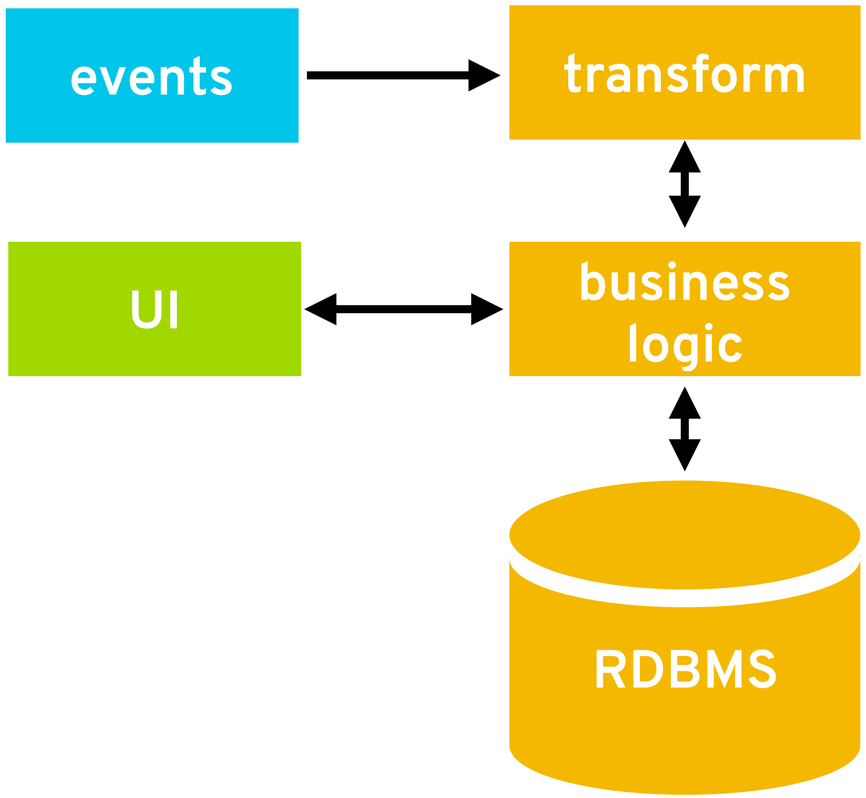
Transactions and analytics



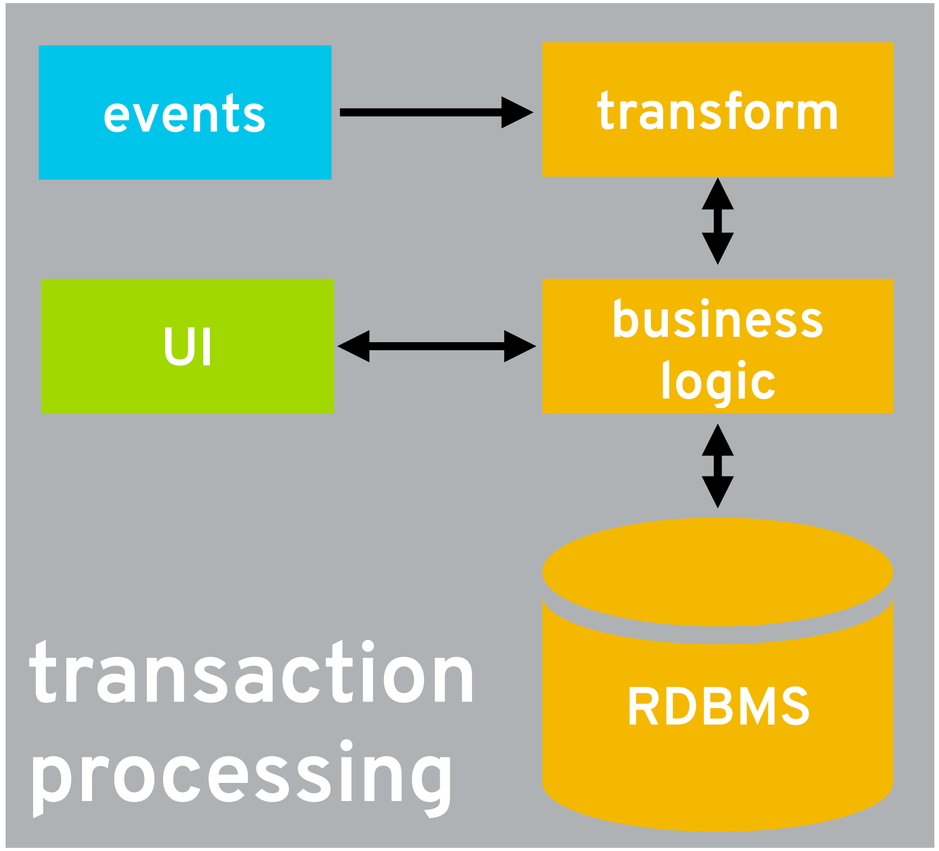
Transactions and analytics



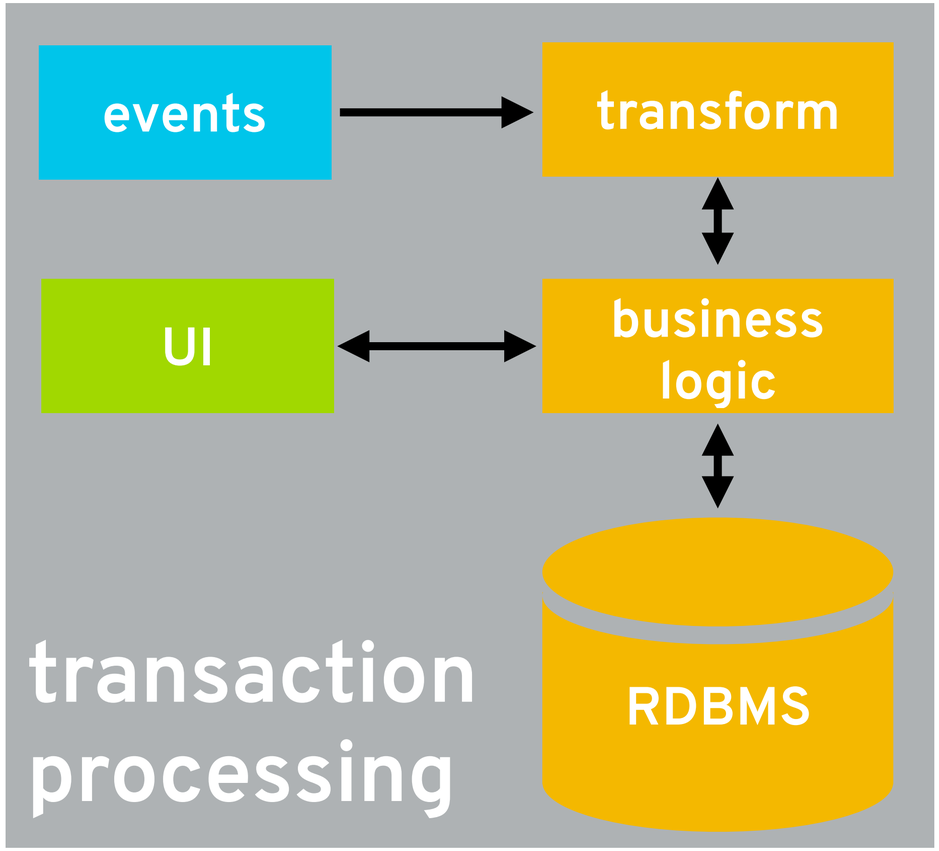
Transactions and analytics



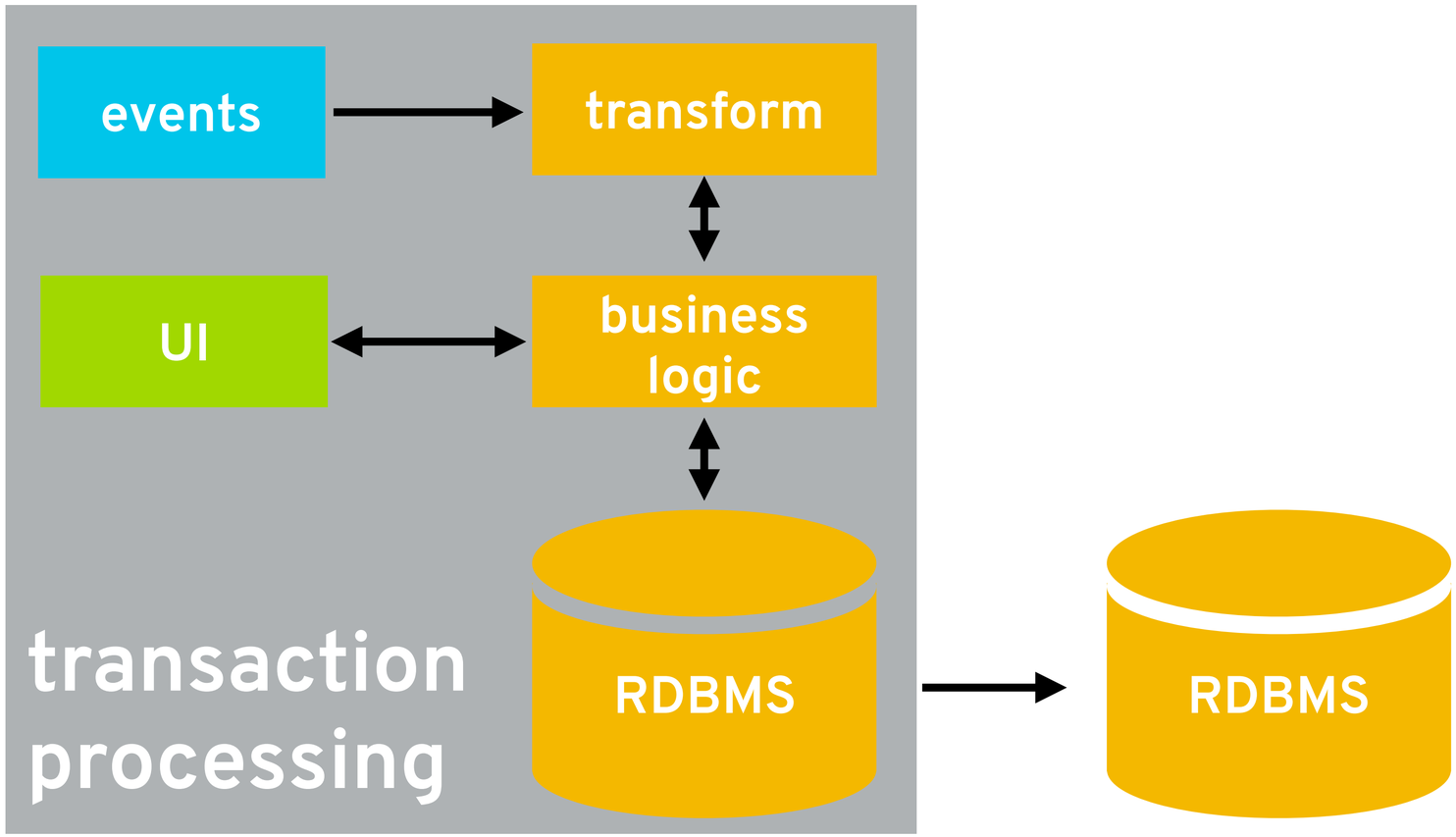
Transactions and analytics



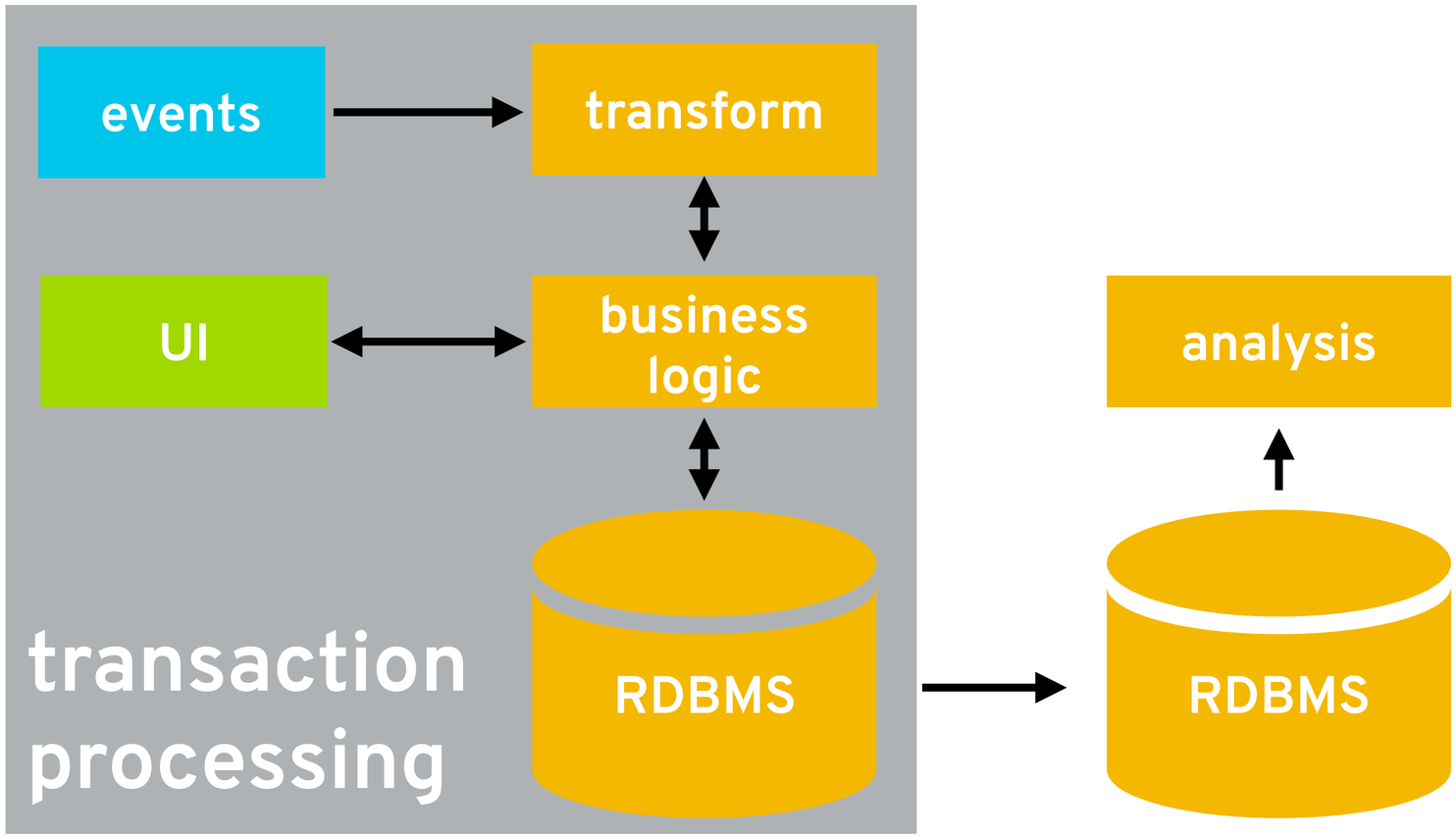
Transactions and analytics



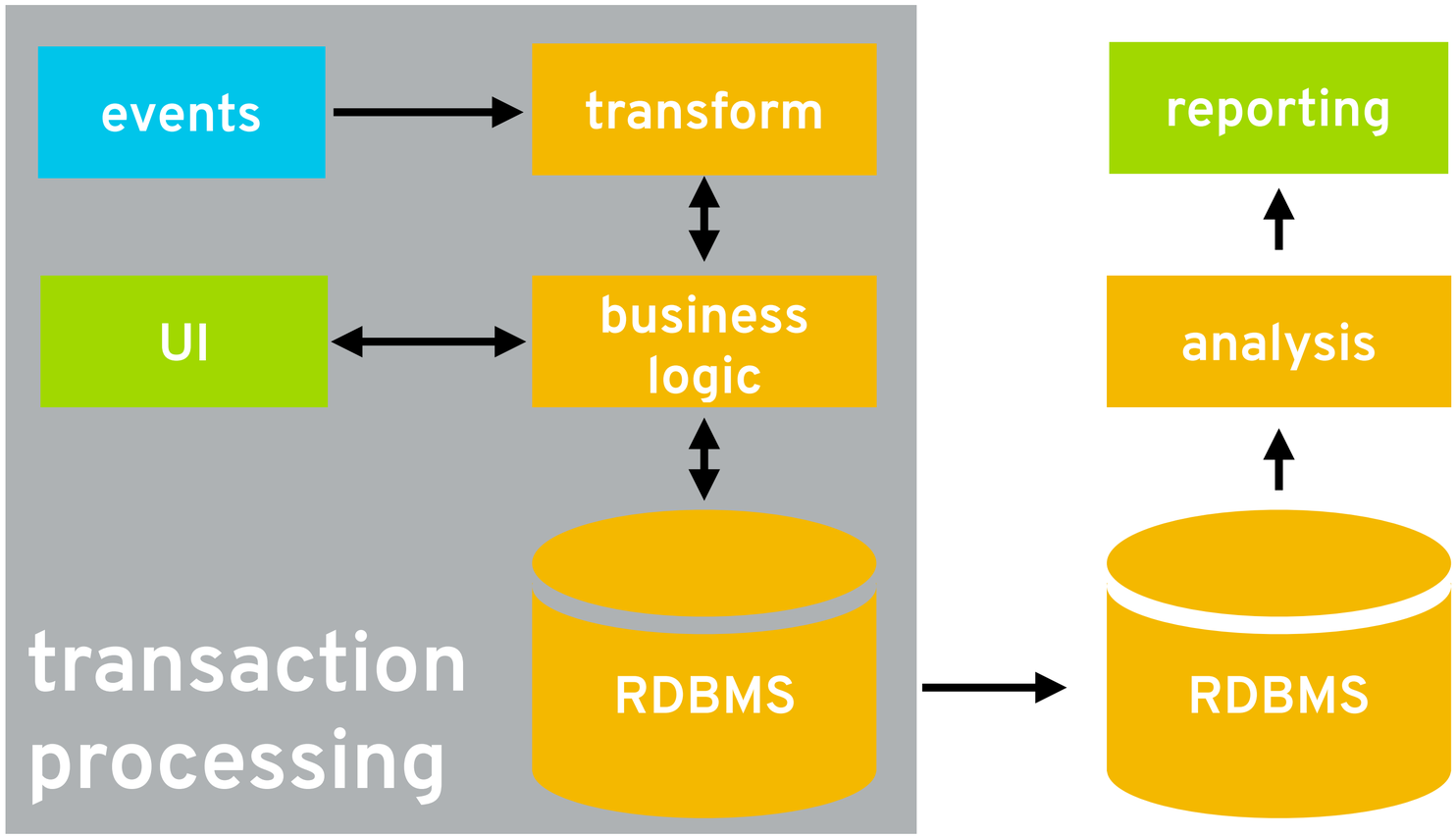
Transactions and analytics



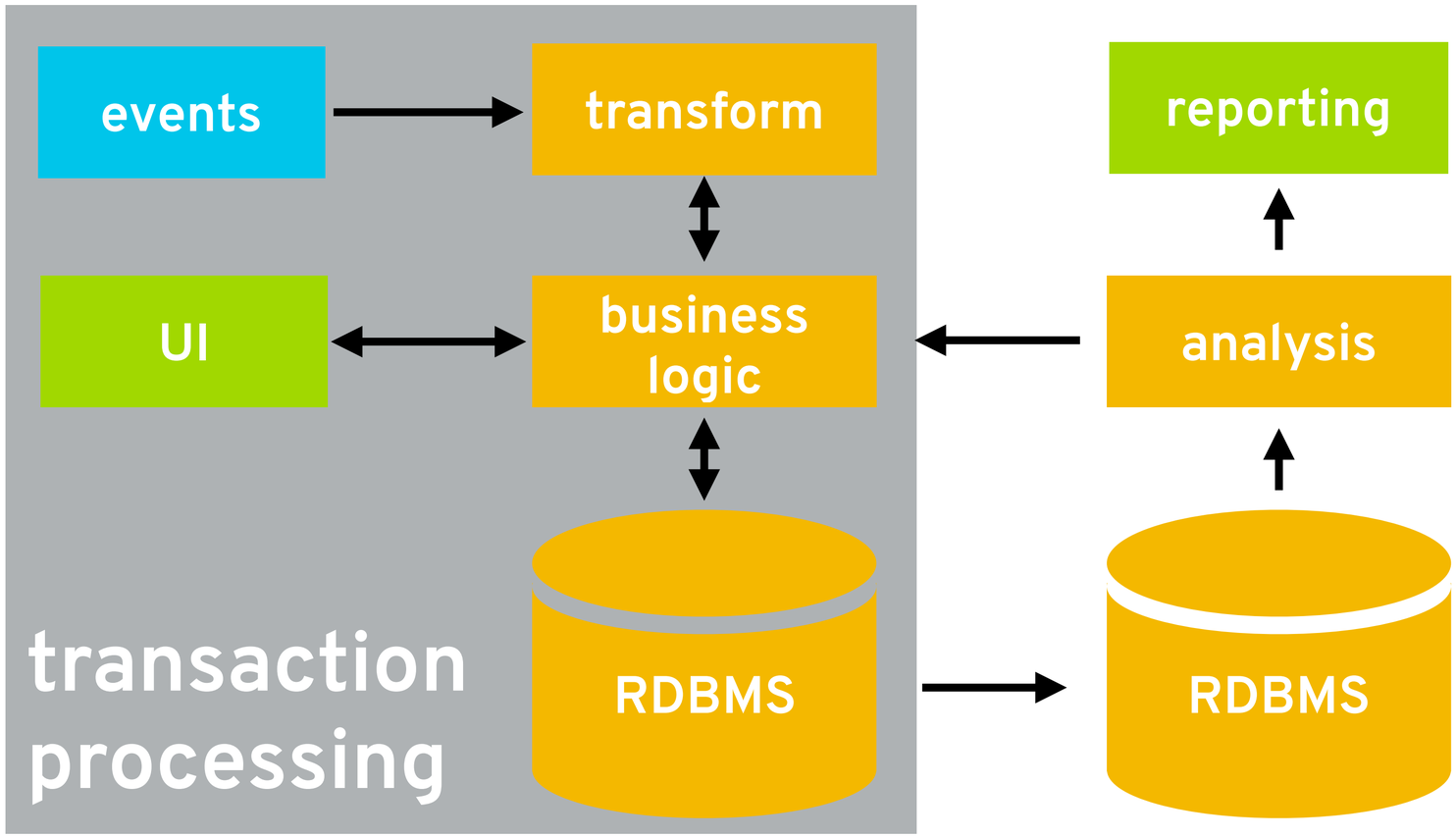
Transactions and analytics



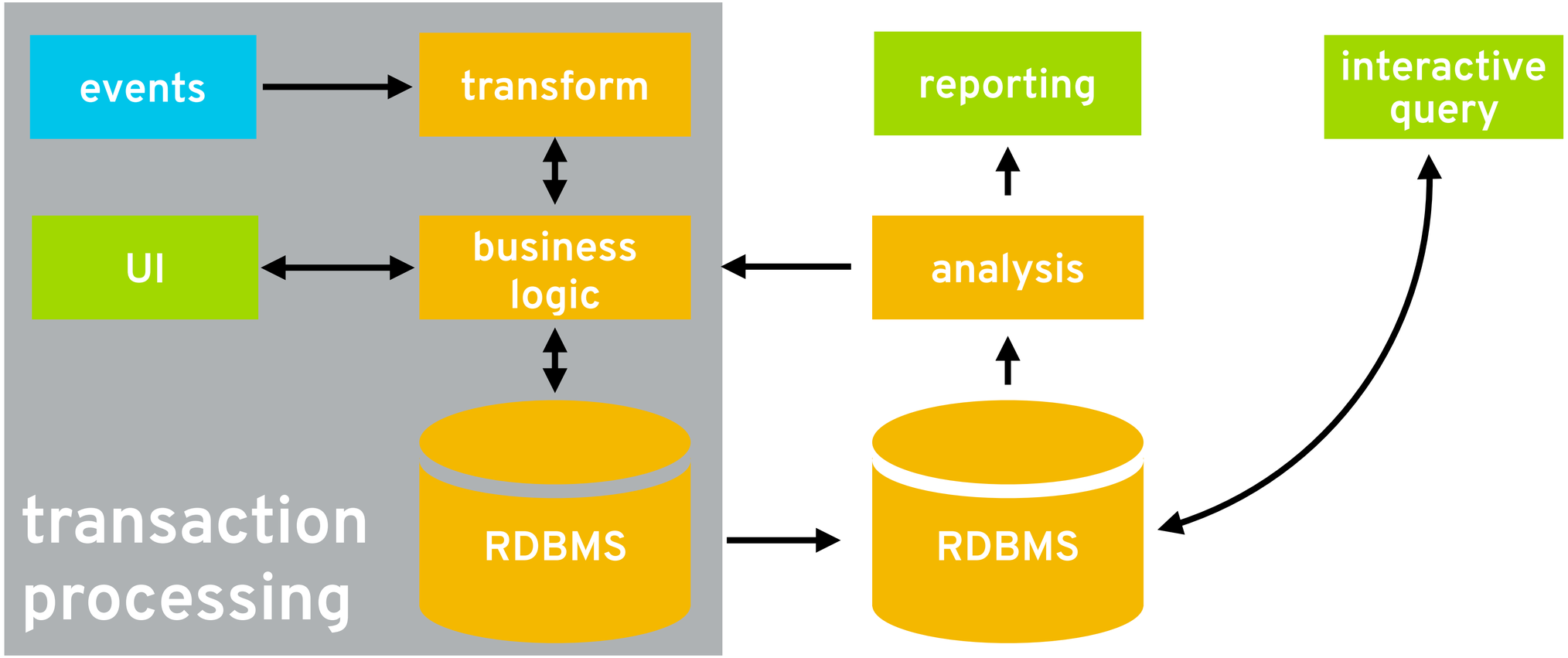
Transactions and analytics



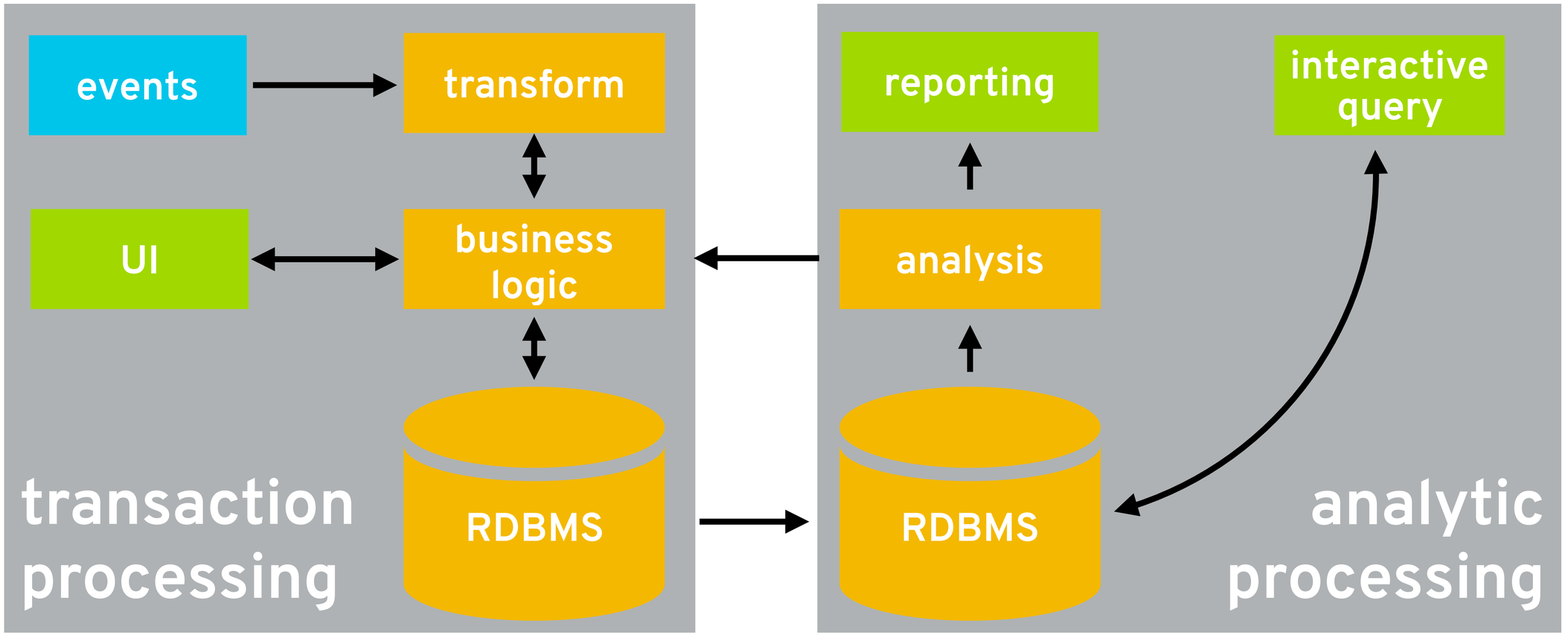
Transactions and analytics



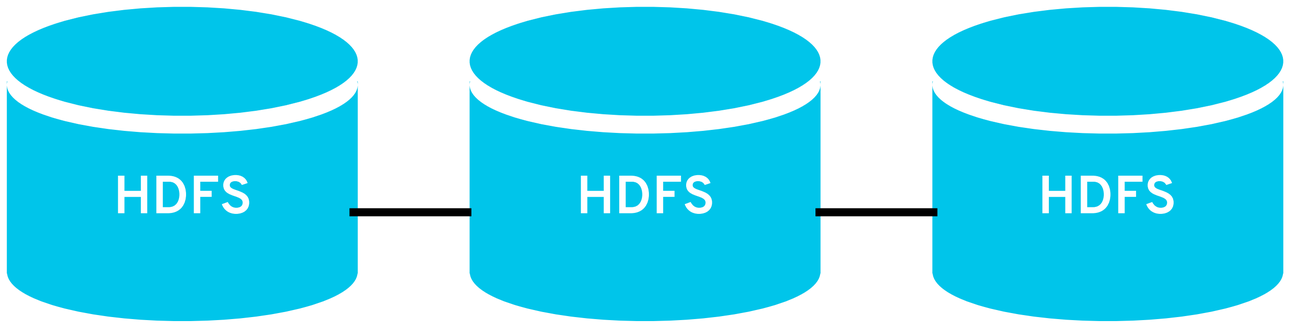
Transactions and analytics



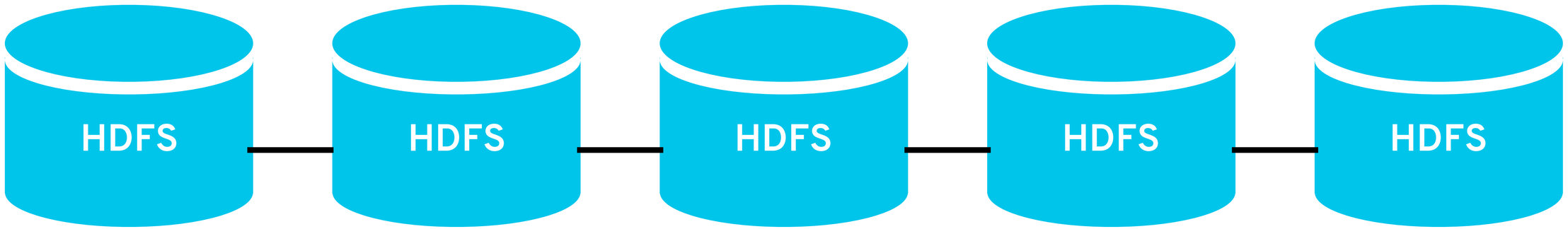
Transactions and analytics



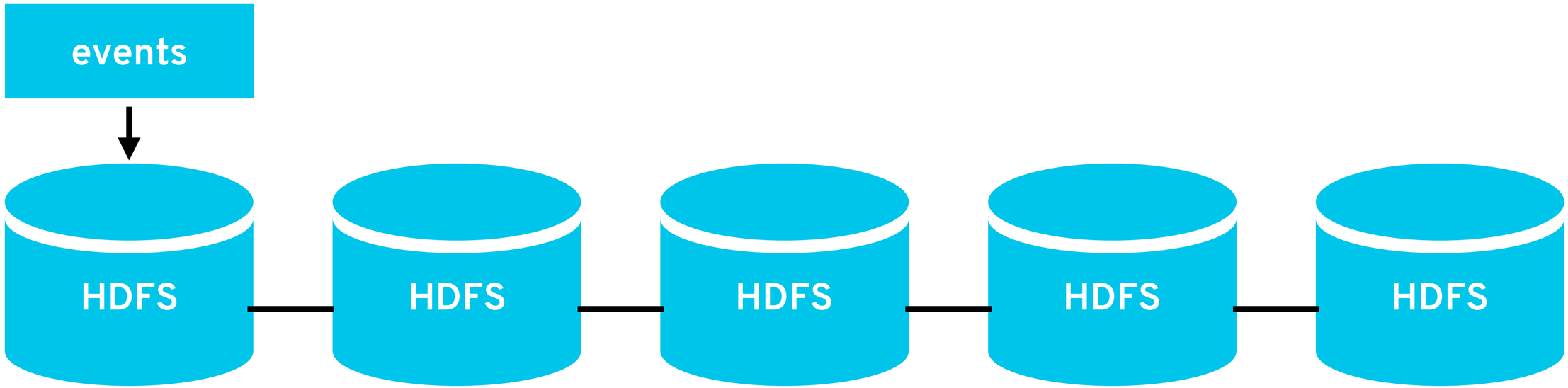
The “data lake”



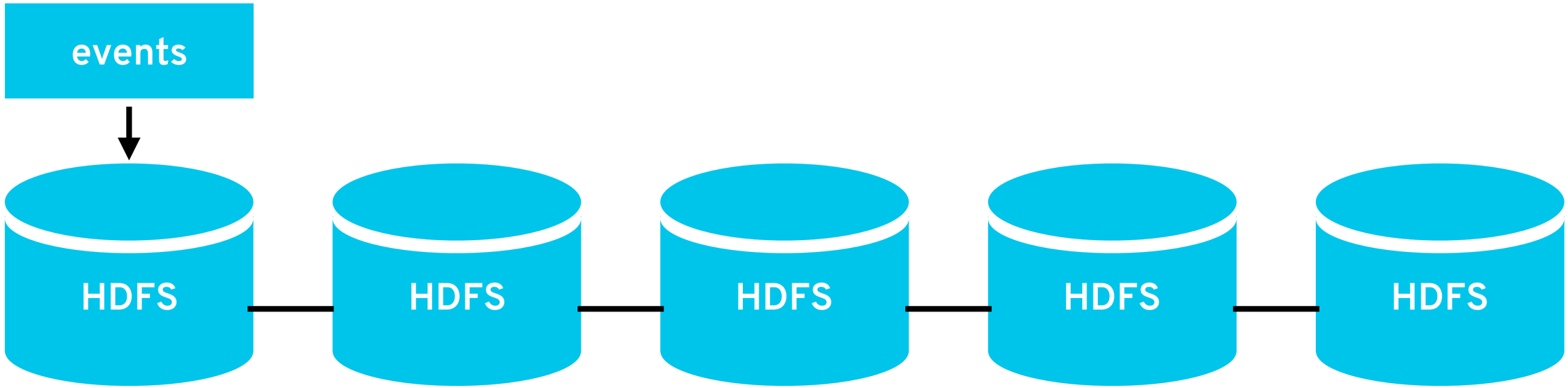
The “data lake”



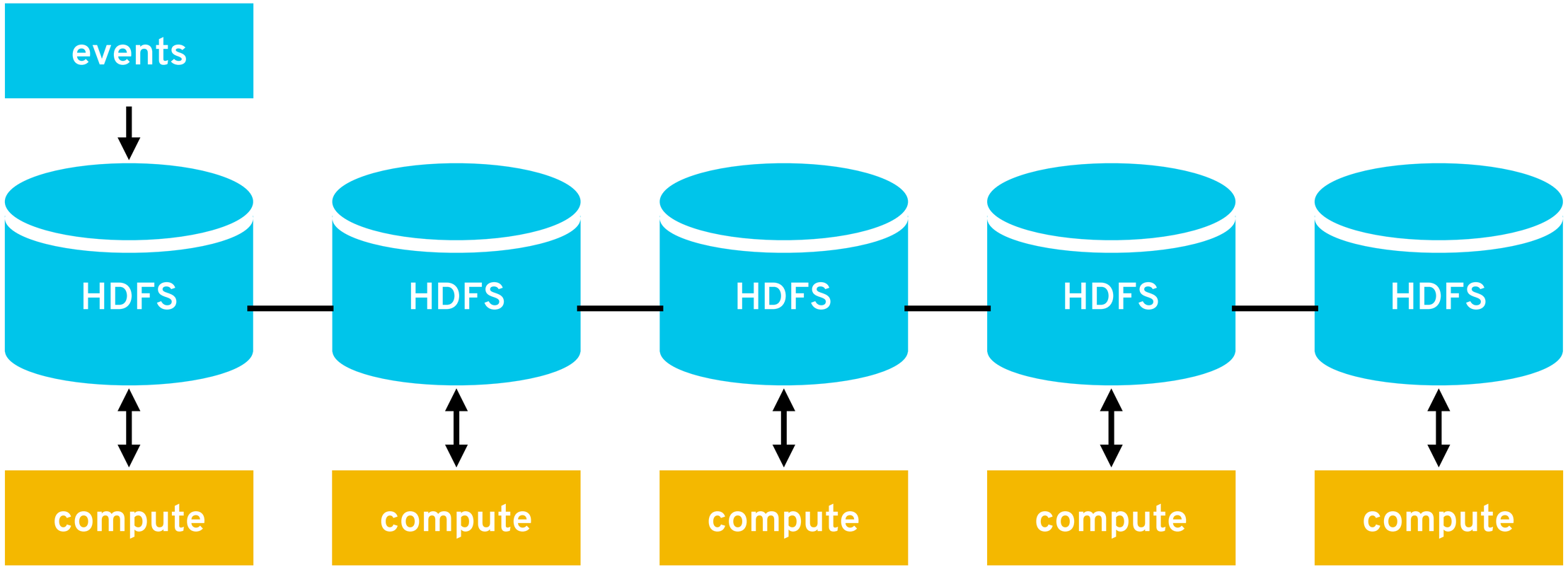
The “data lake”



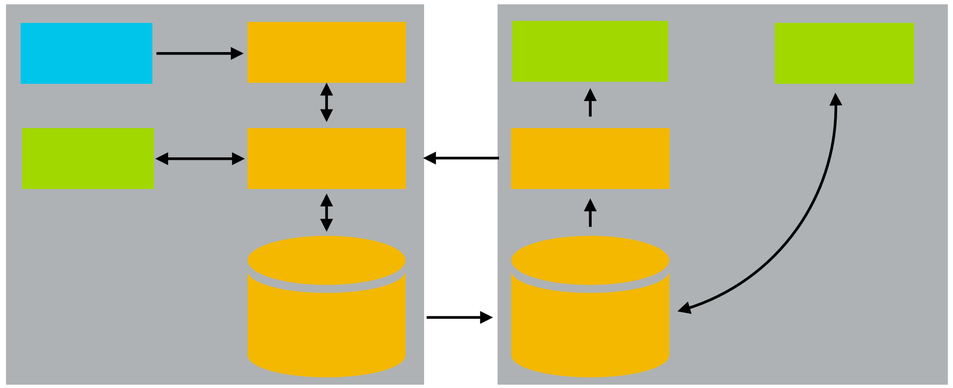
The “data lake”

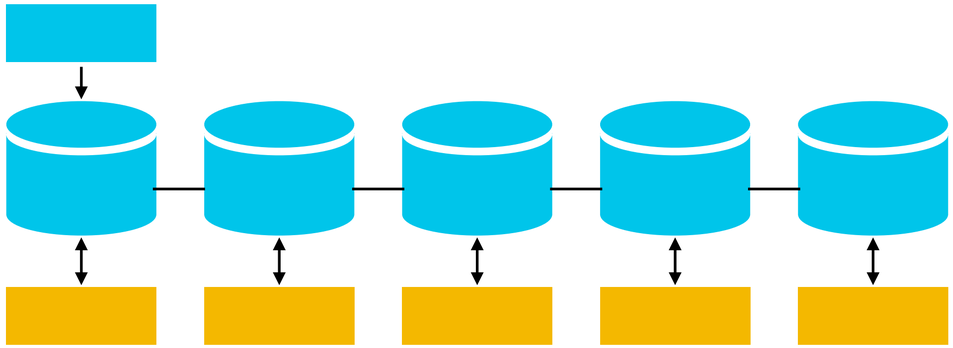


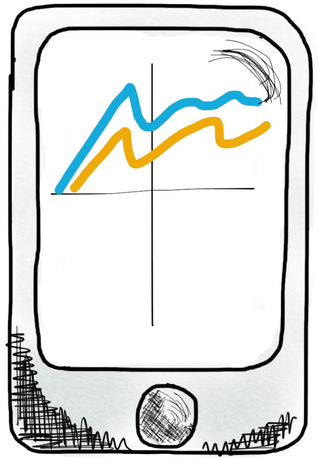
The “data lake”



Where do the applications go?



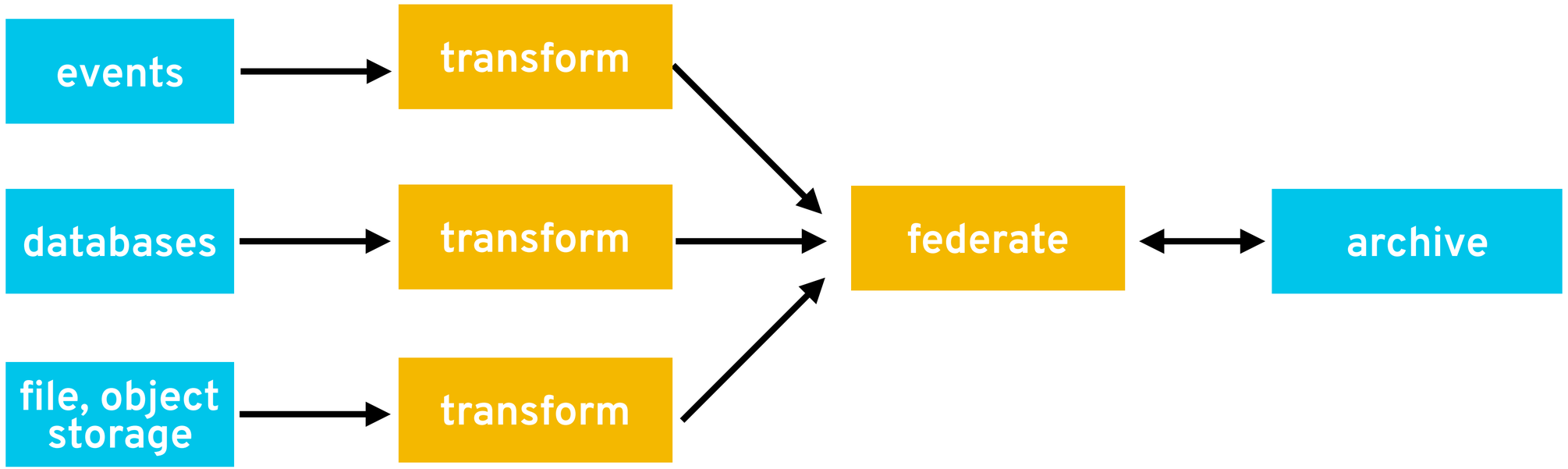




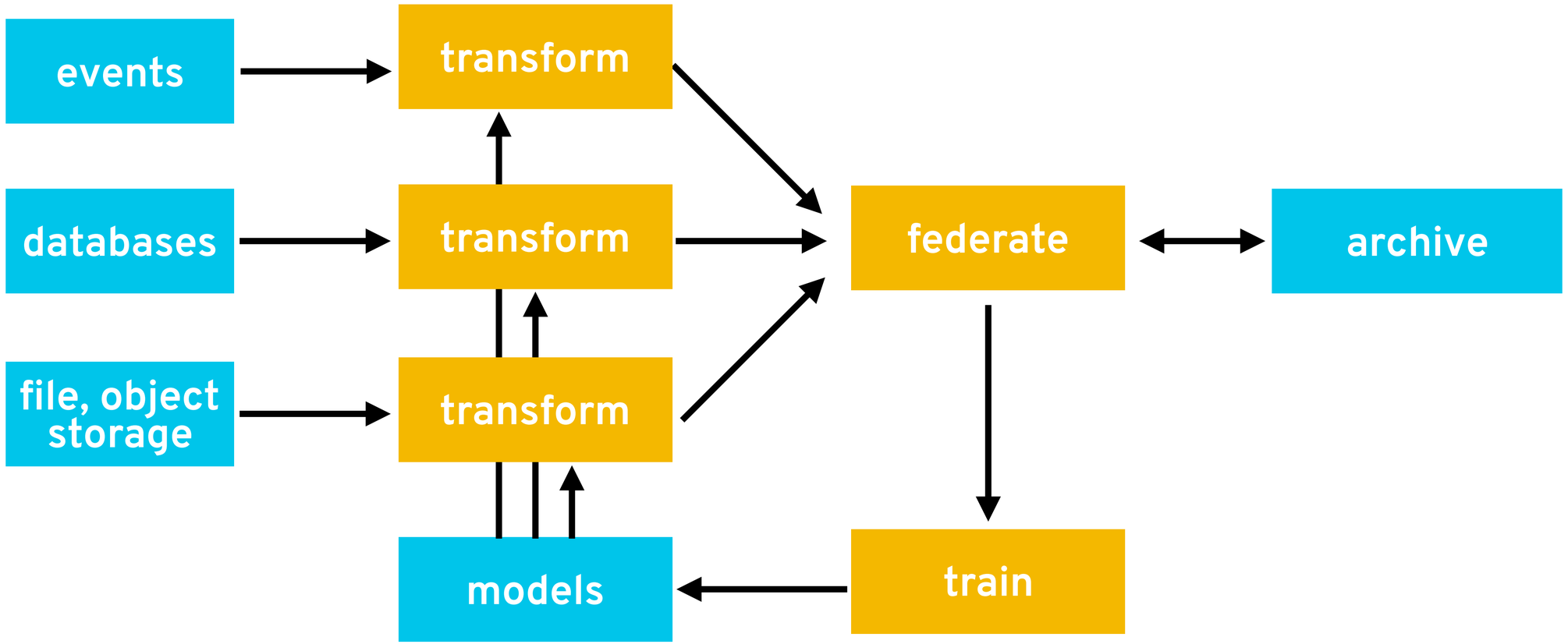
Architectures that **separate analytics from applications** make sense **only if** analytics is a **separate workload.**

AN ARCHITECTURE FOR ANALYTIC APPLICATIONS IN CONTAINERS

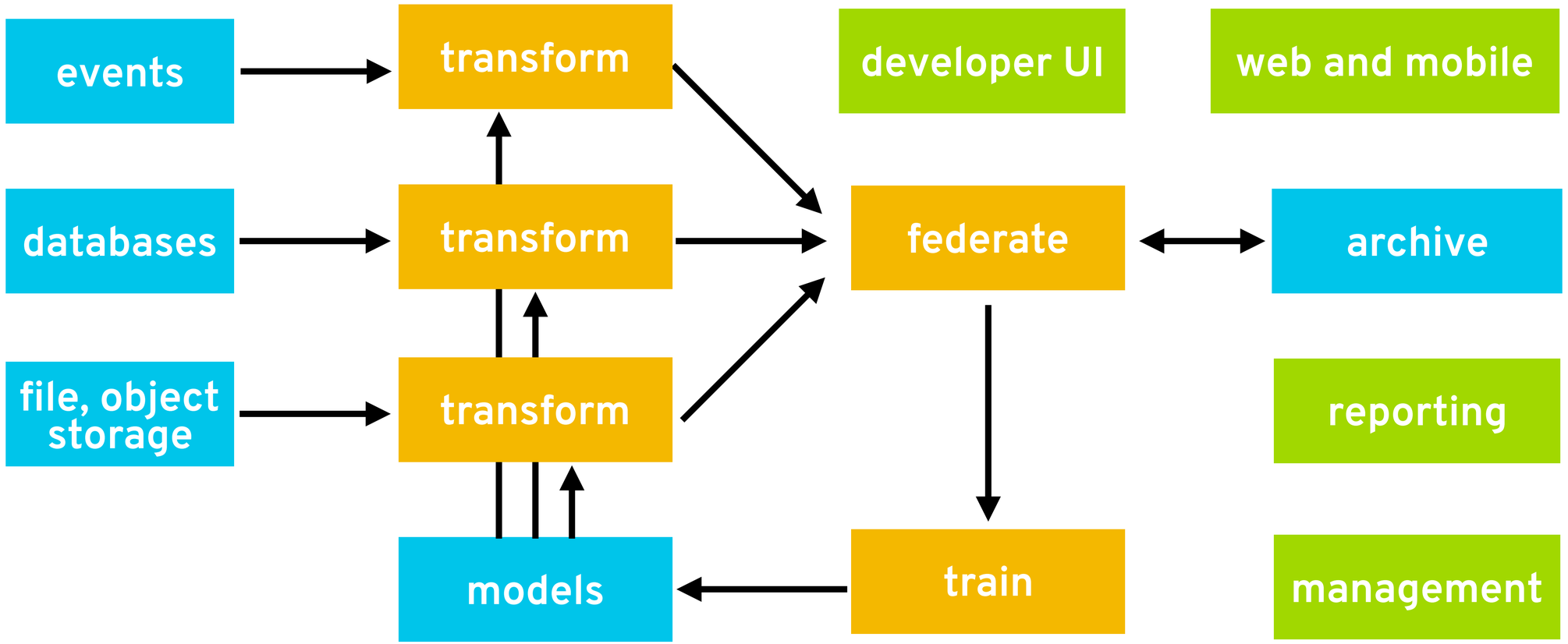
High-level app architecture



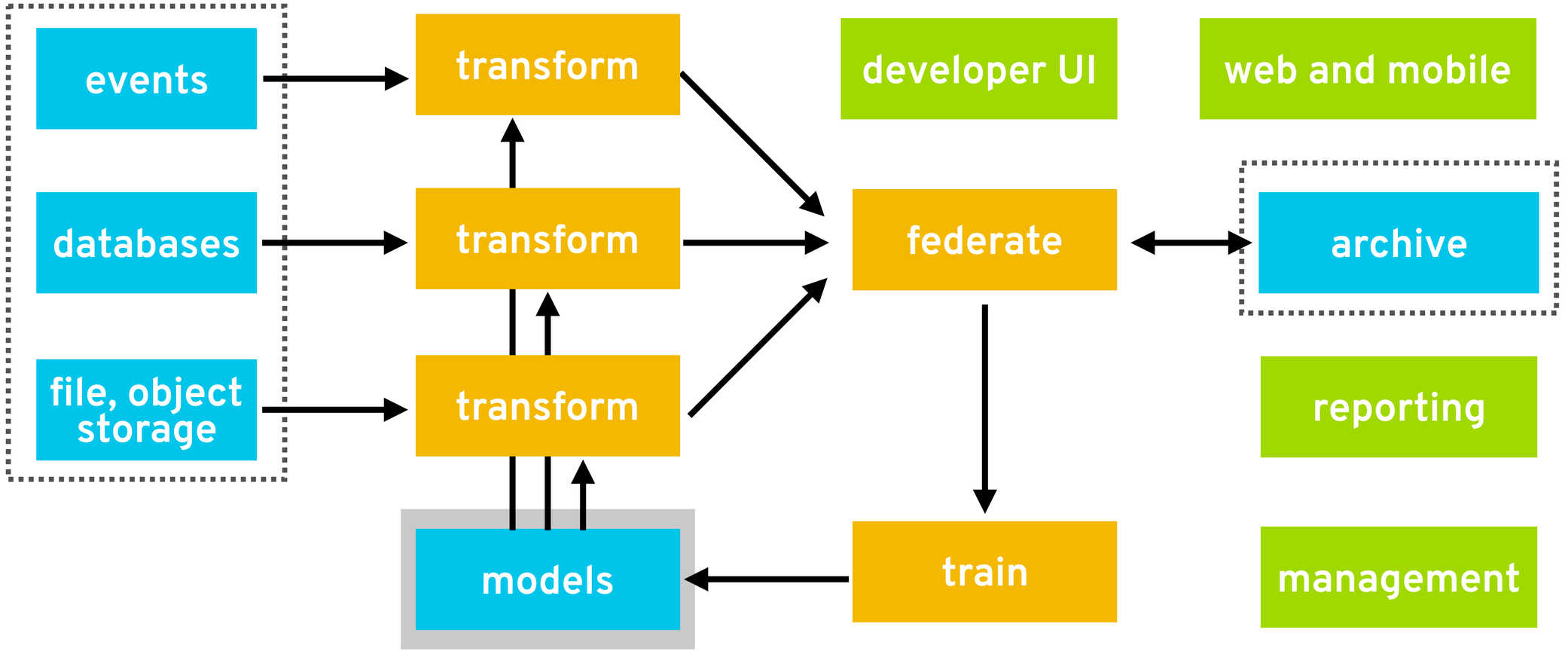
High-level app architecture



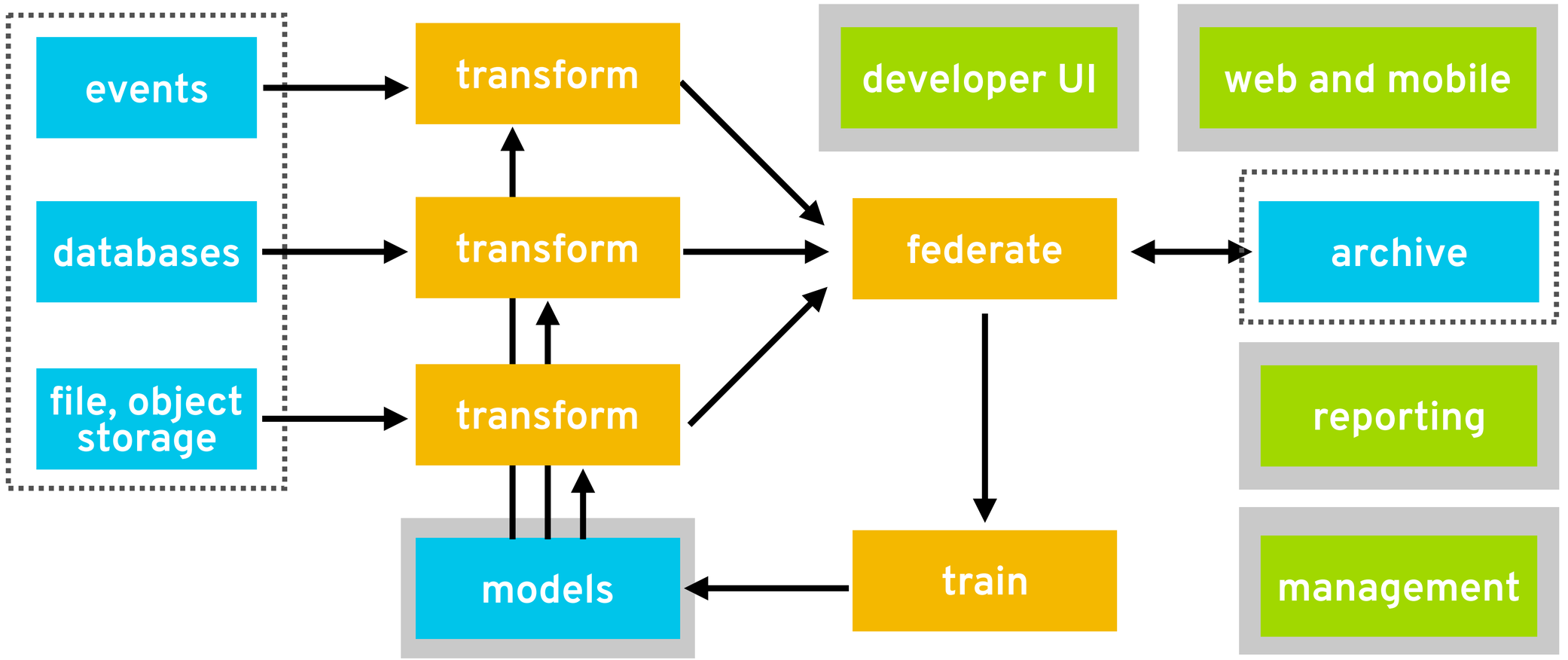
High-level app architecture



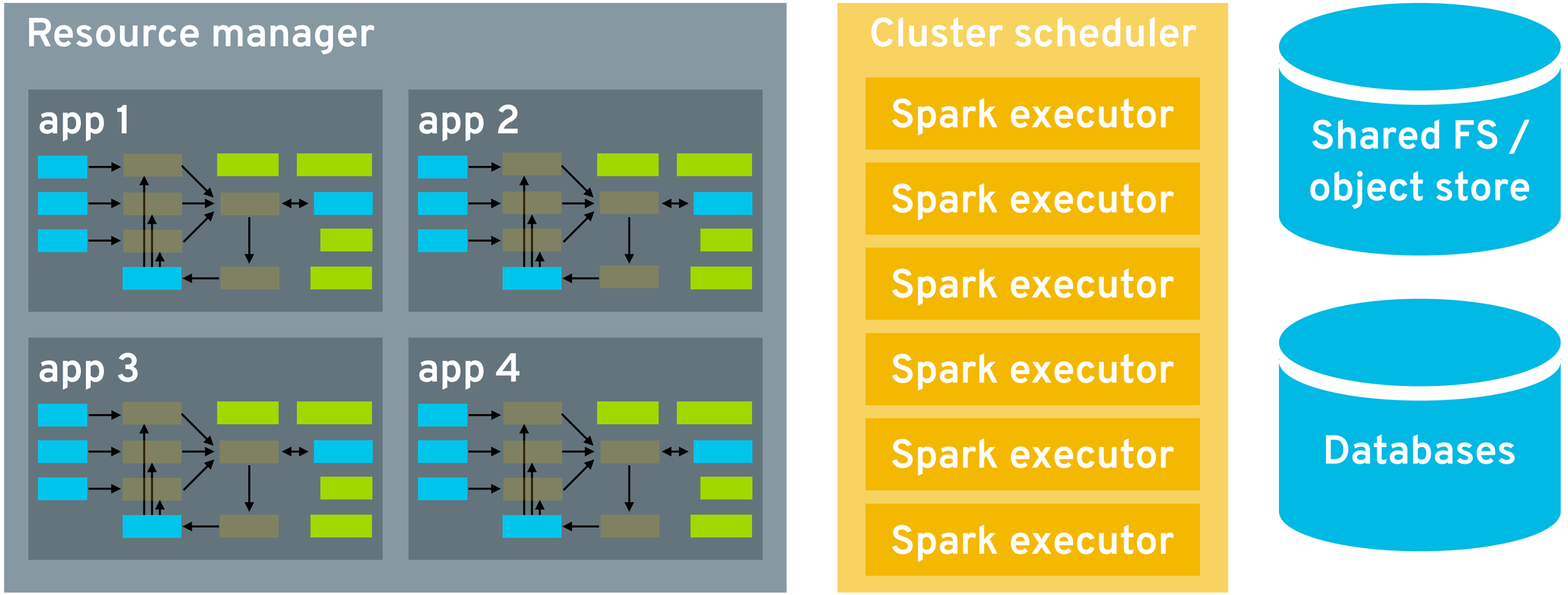
High-level app architecture



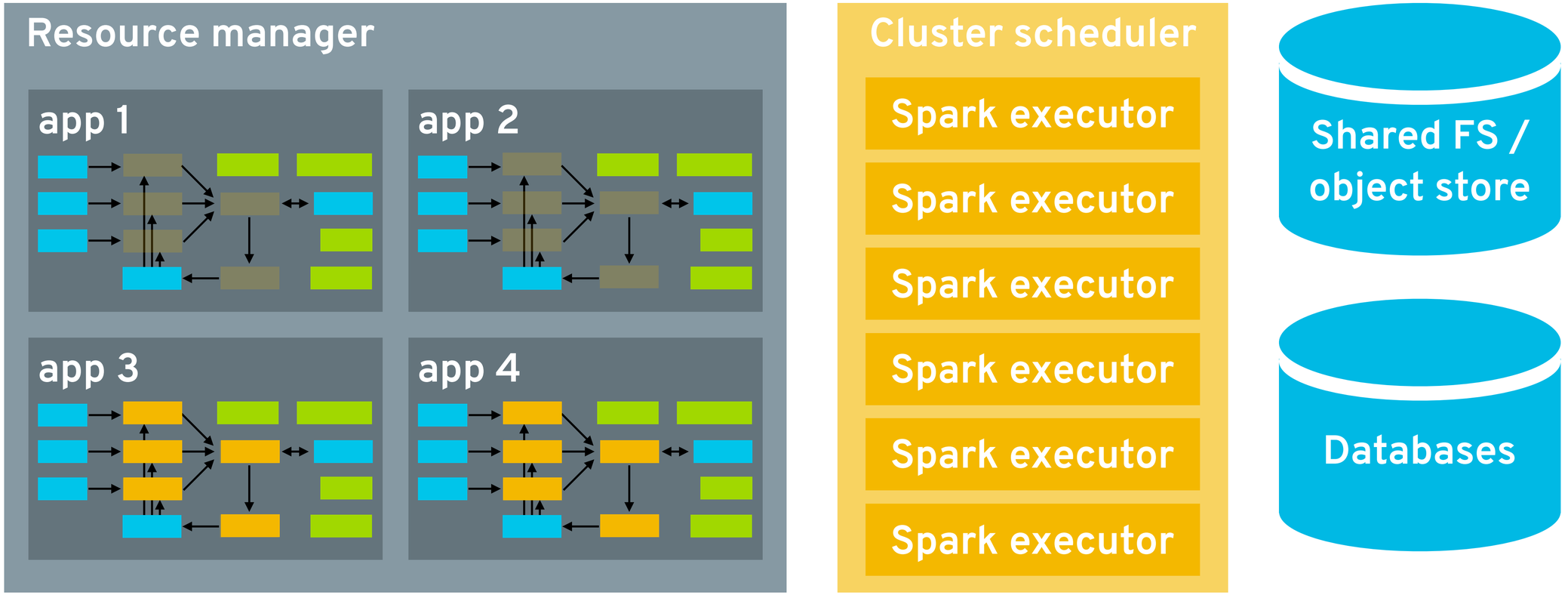
High-level app architecture



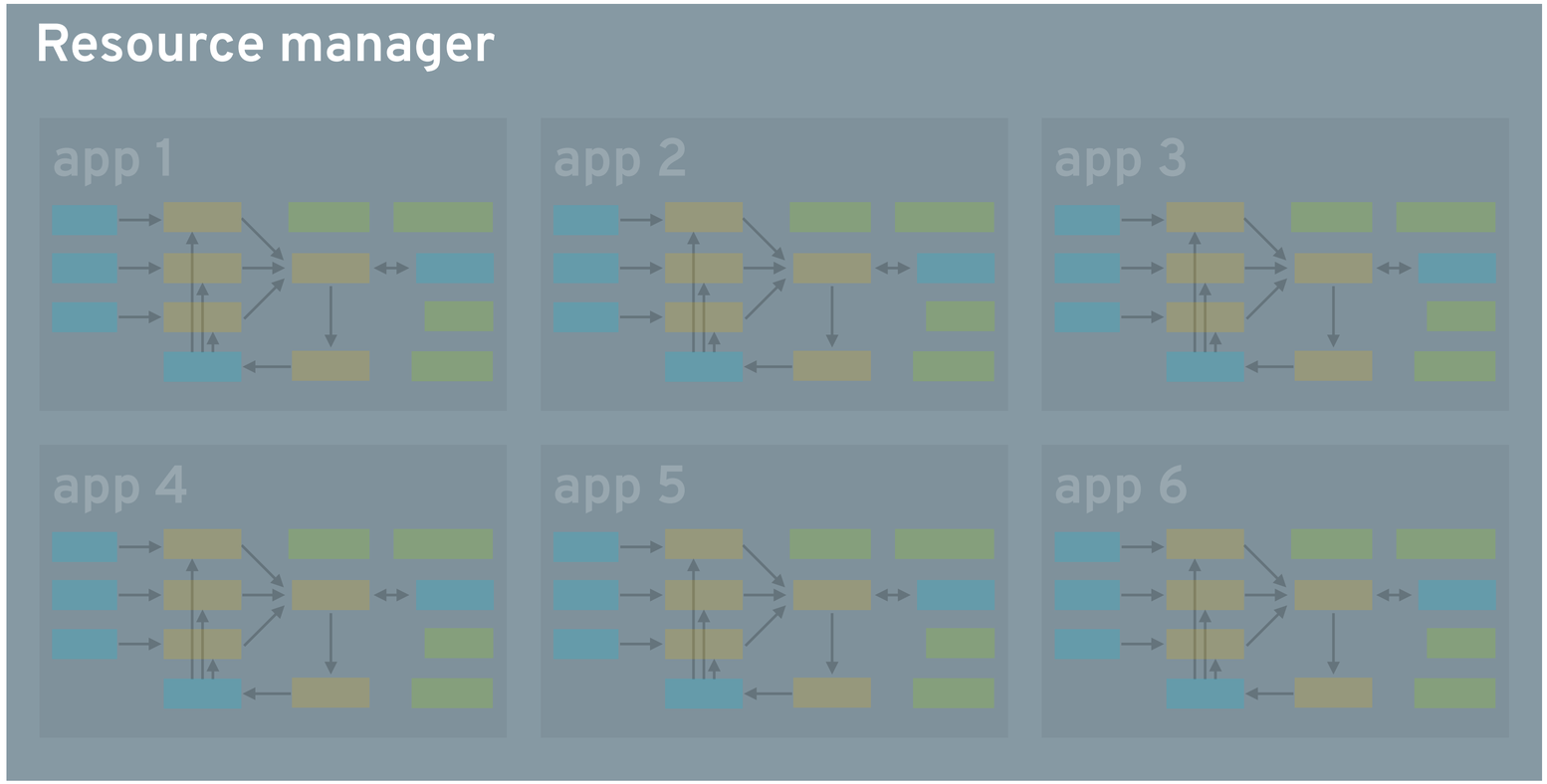
Multitenant compute clusters



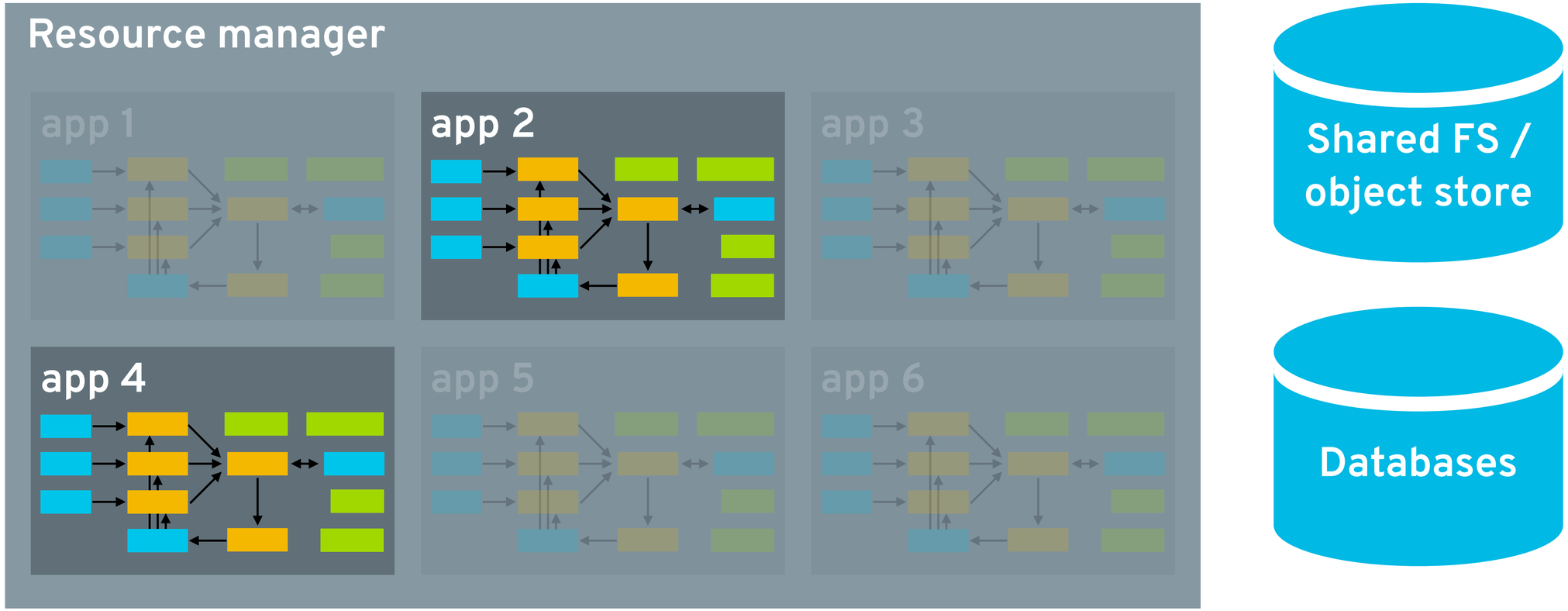
Multitenant compute clusters



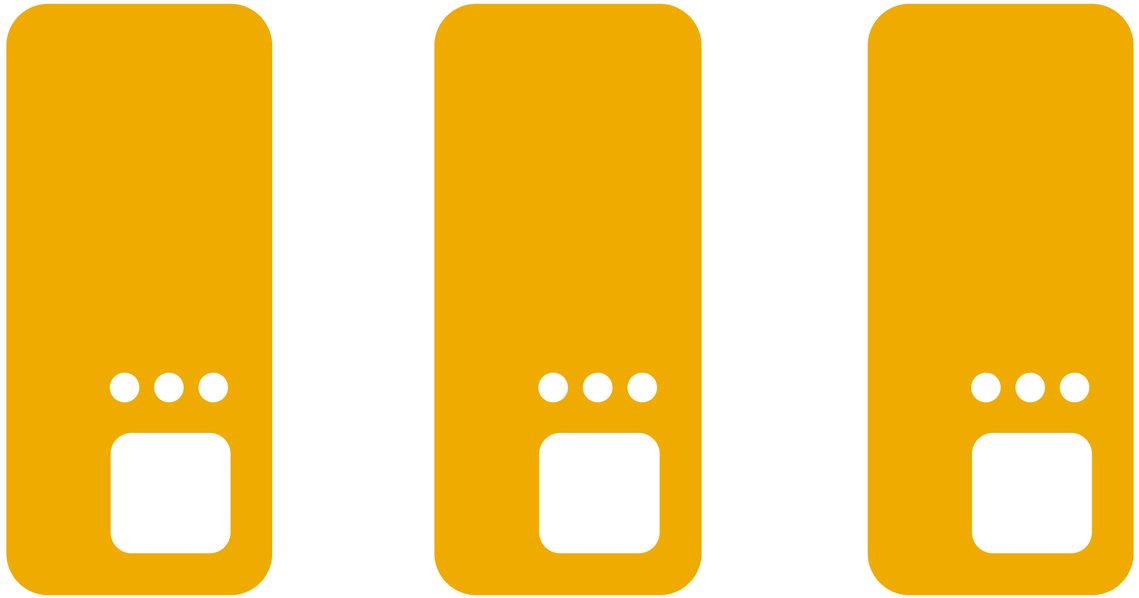
One cluster per application

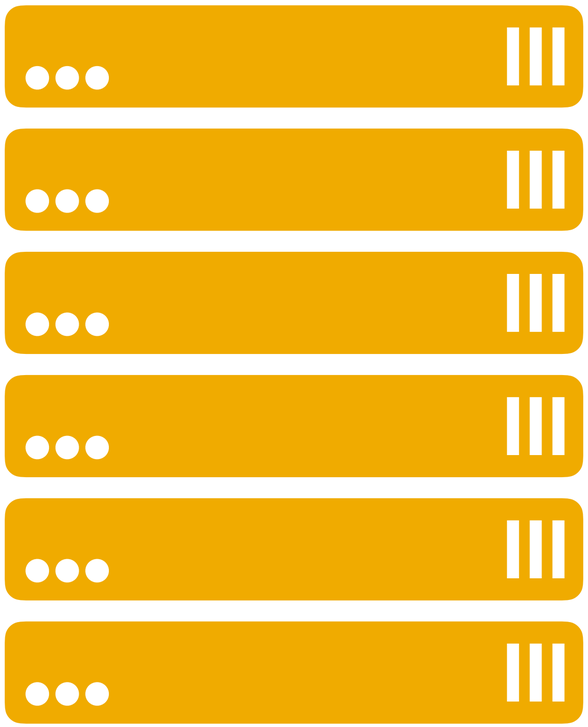


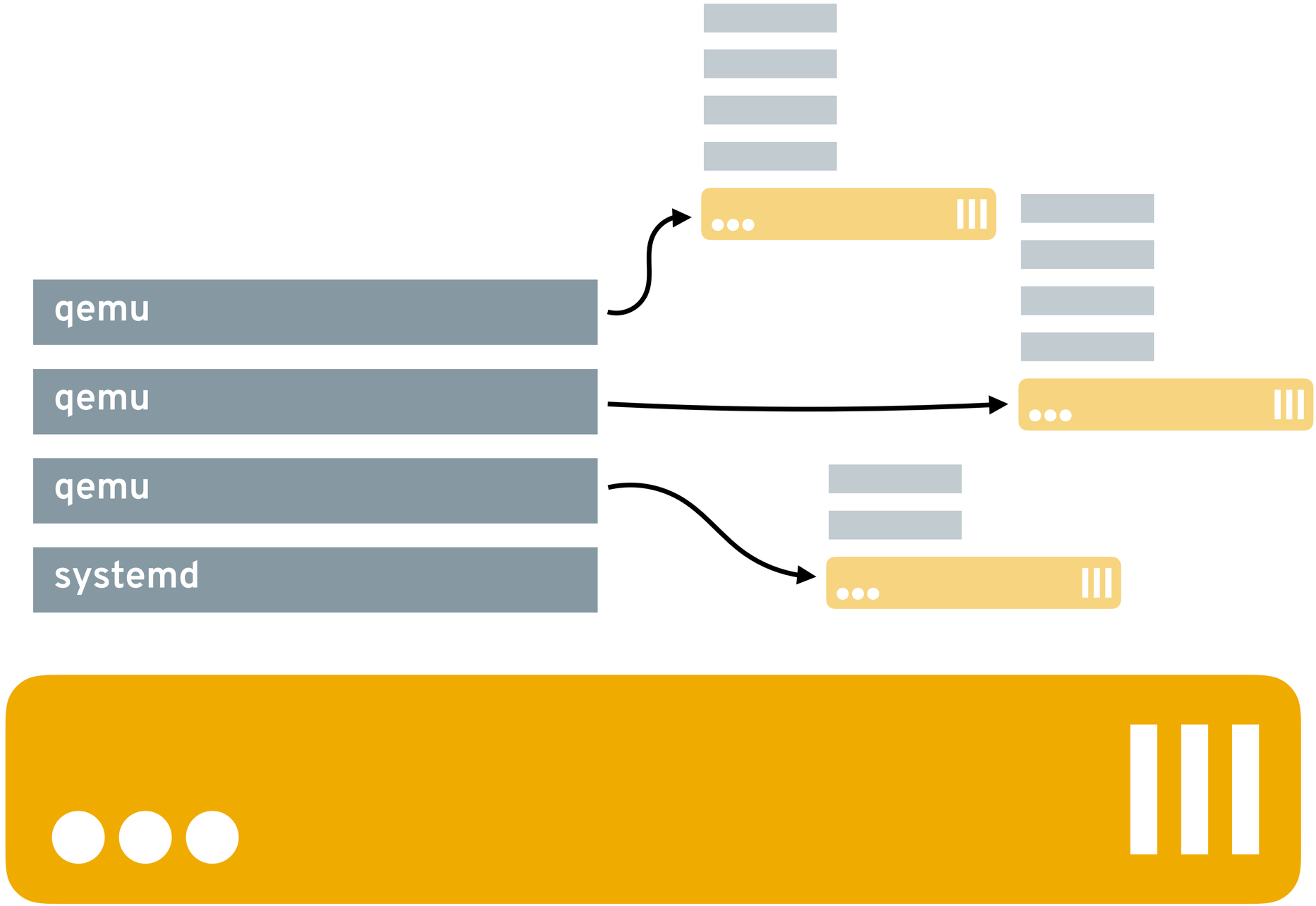
One cluster per application



PRACTICAL CONCERNS: SECURITY AND PERFORMANCE





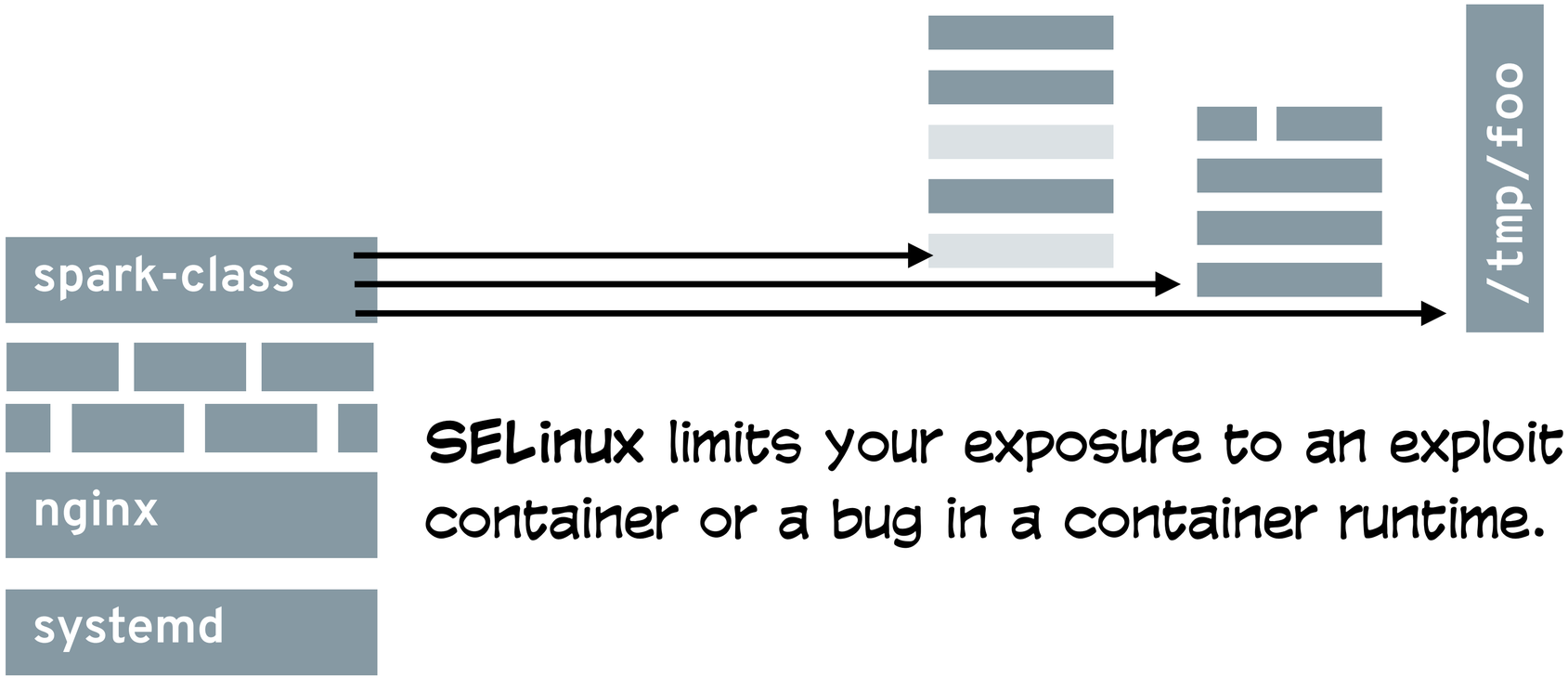

qemu
qemu
qemu
systemd


spark-class
mongodb
nginx
systemd
/tmp/foo
/tmp/bar
/tmp/blah


spark-class
mongodb
nginx
systemd

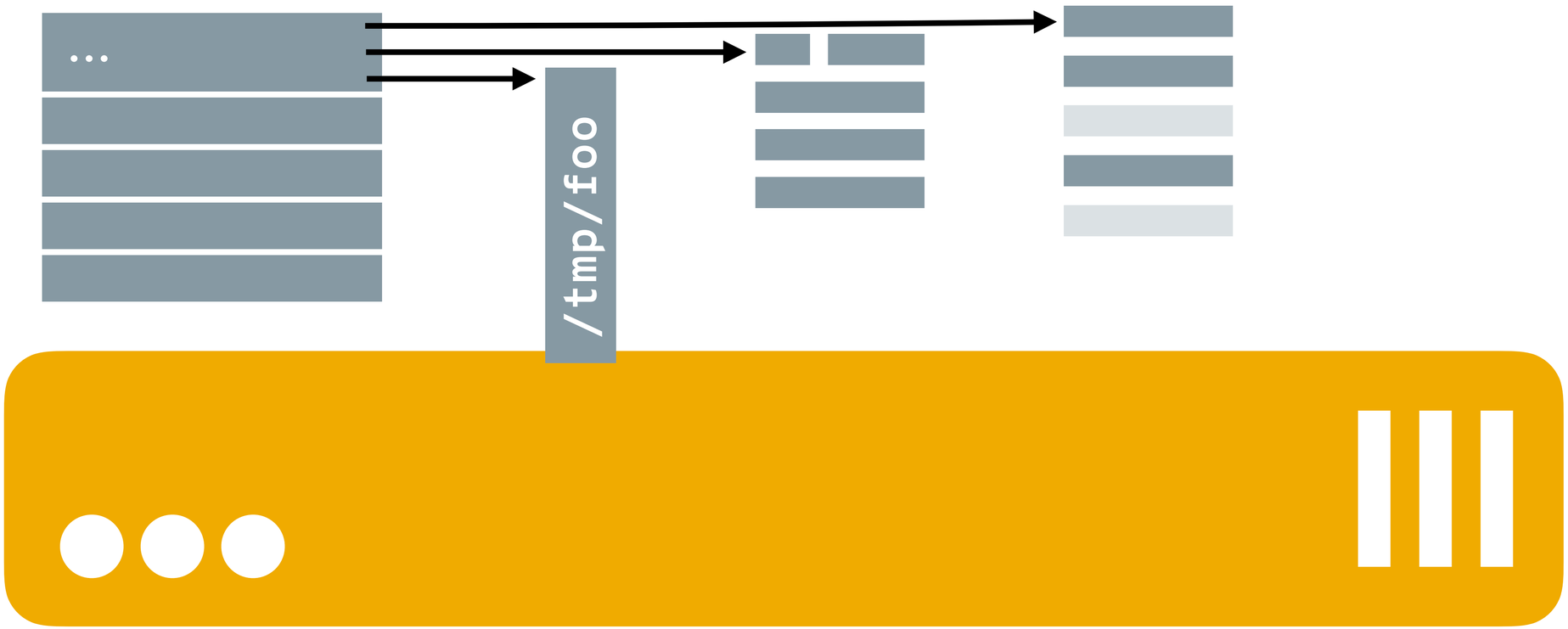

spark-class
/tmp/foo
nginx
systemd


spark-class
nginx
systemd
/tmp/foo

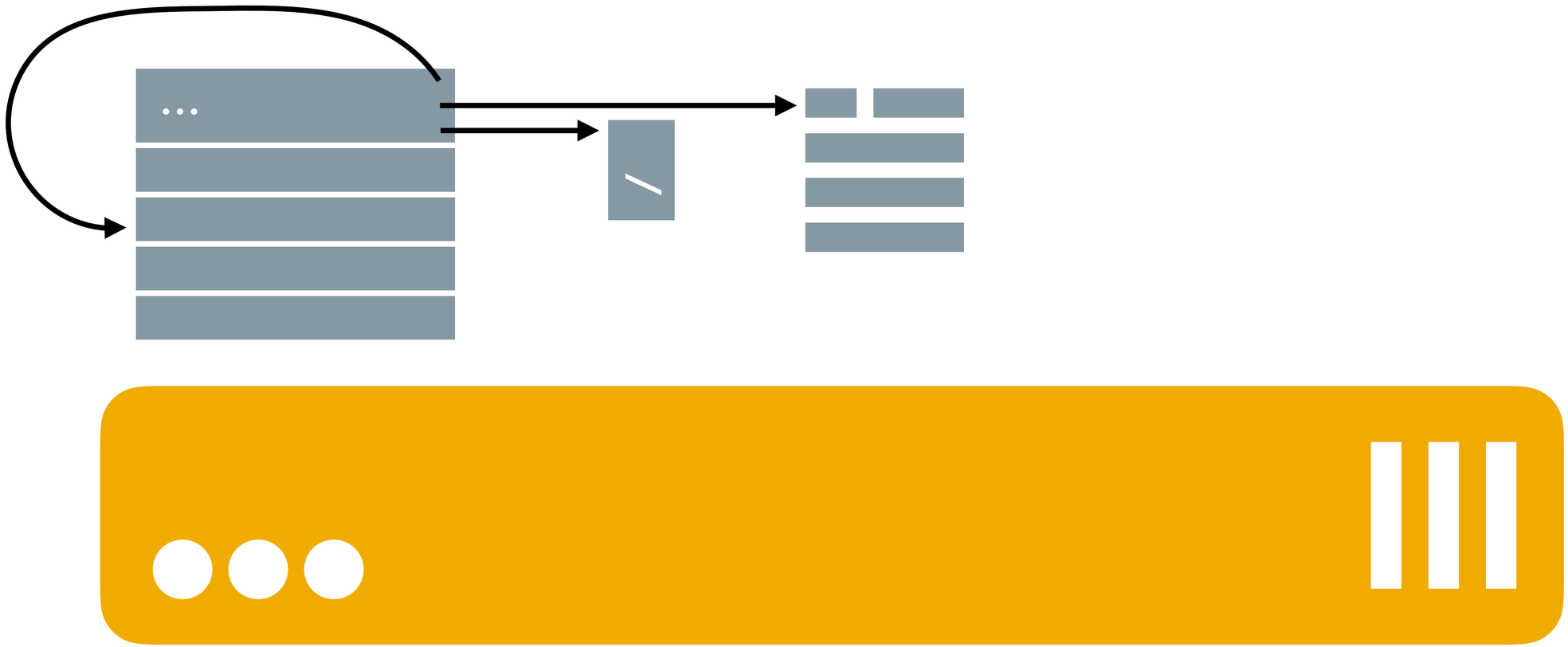


SELinux limits your exposure to an exploit in a container or a bug in a container runtime.

Root is root



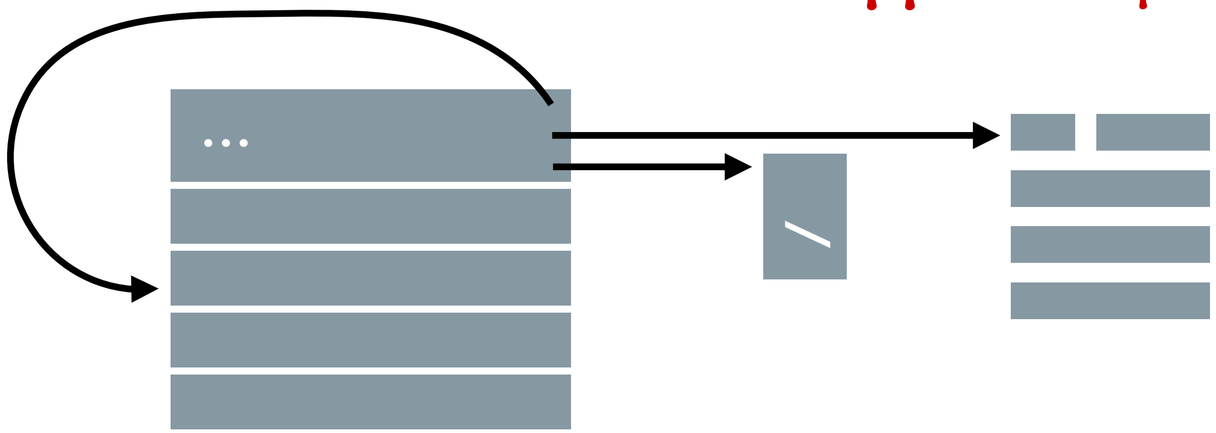
Root is root



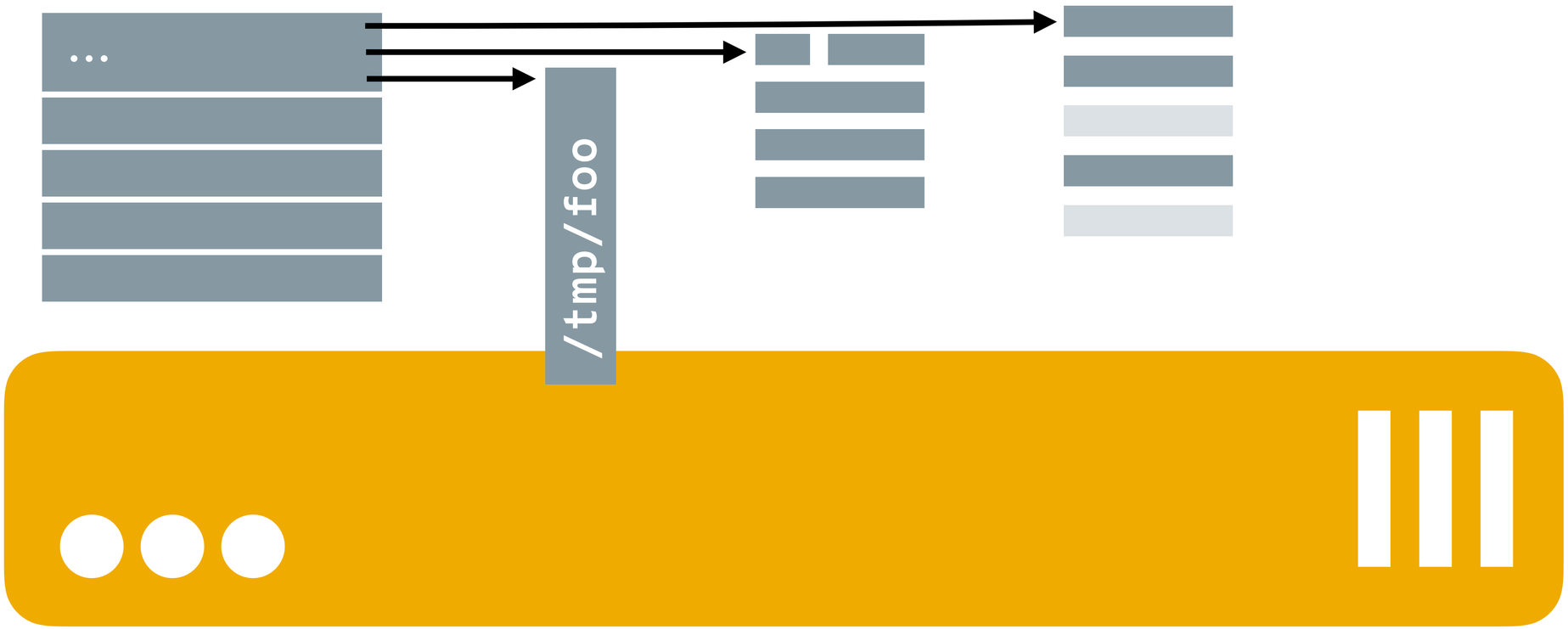
Root is root

...and you might not have a password file!

Use nss_wrapper to provide one.

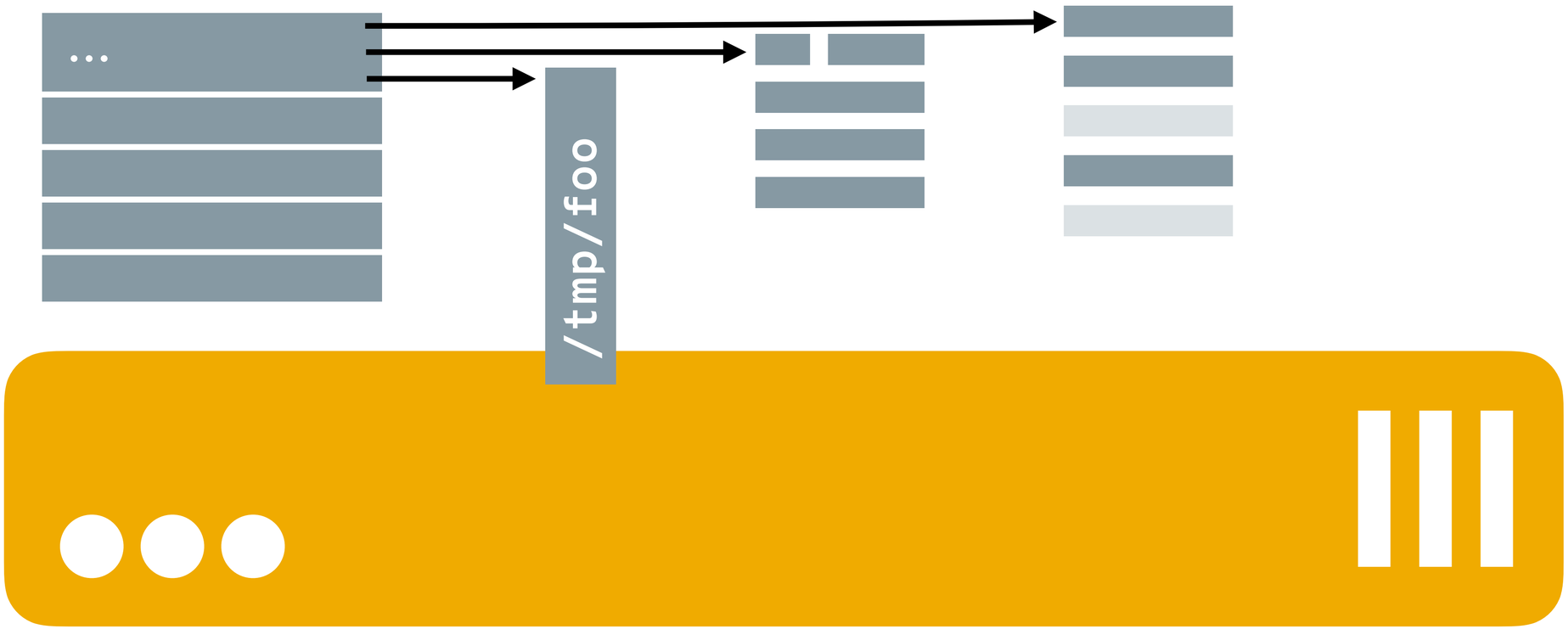


Denials of service



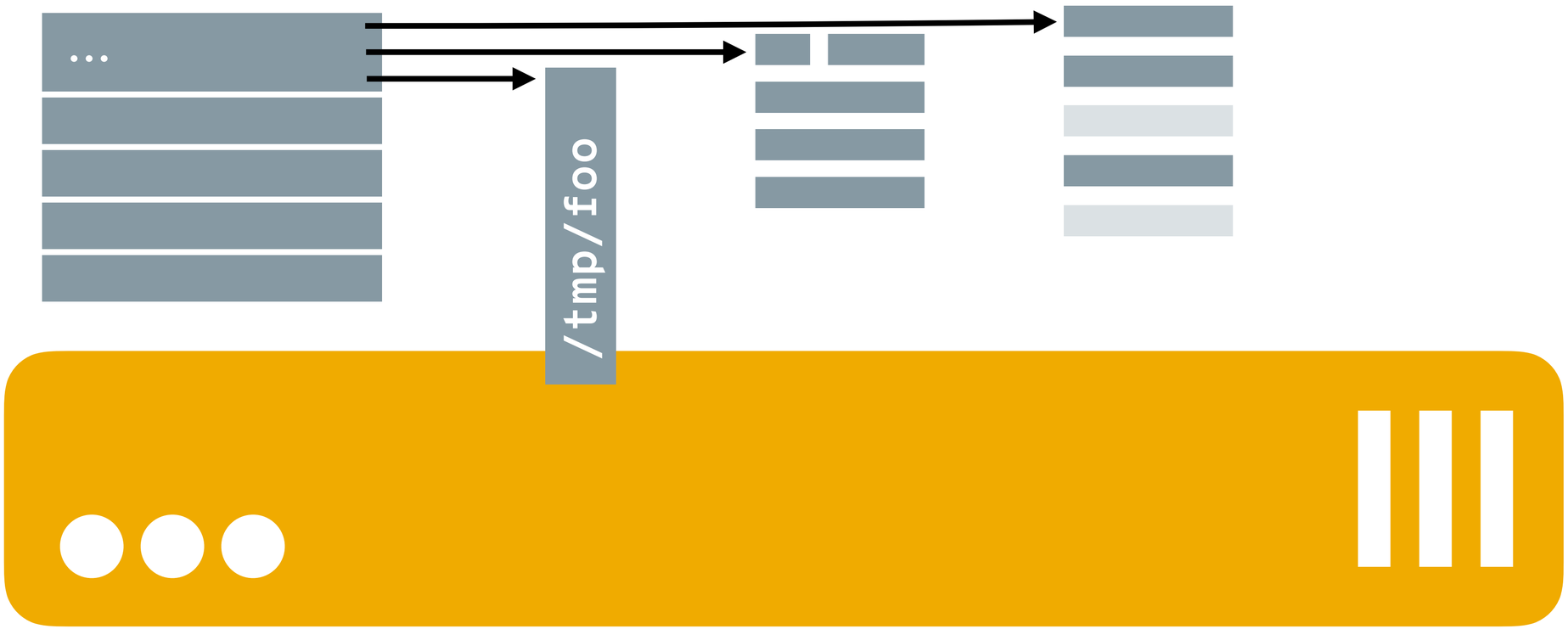
Denials of service

Kernel panics



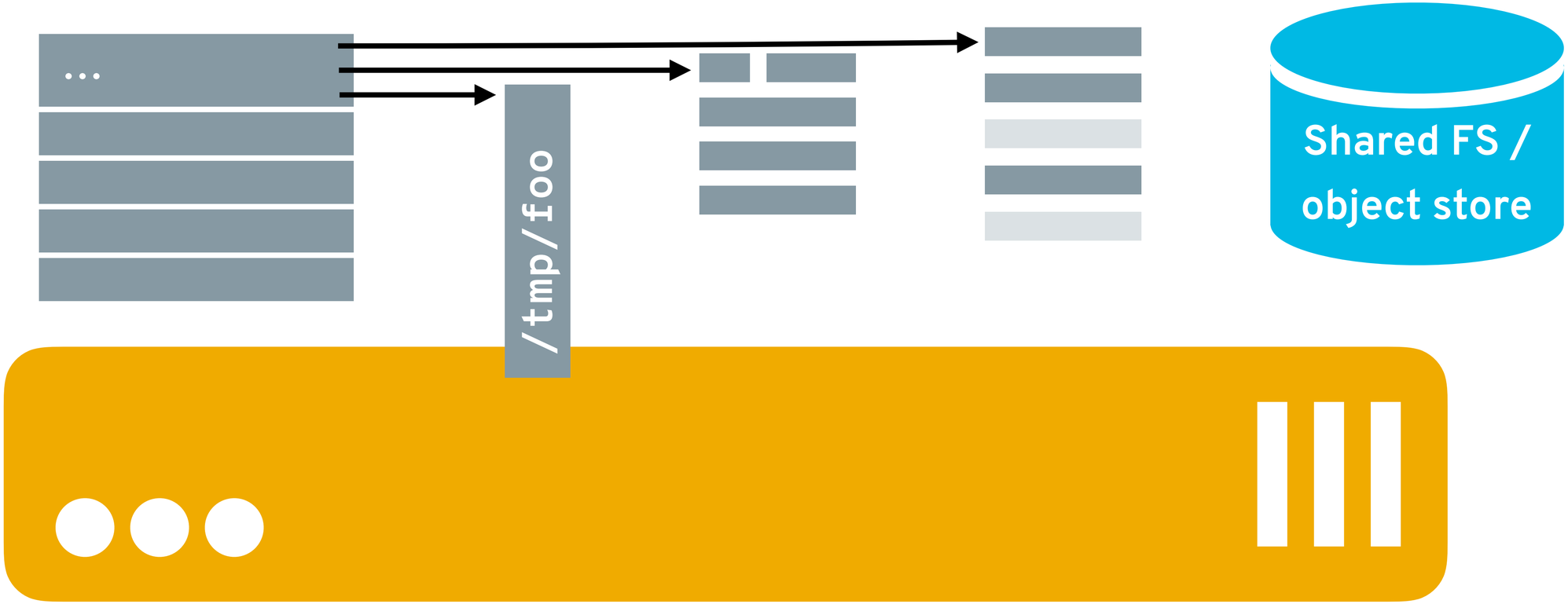
Kernel panics

Keeping secrets



Keeping secrets

```
ACCESS_KEY=…
SECRET_KEY=…
```



Keeping secrets

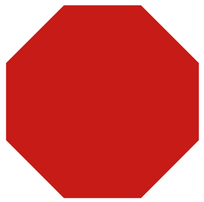
```
cat <<EOF > secret.txt
ACCESS_KEY=…
SECRET_KEY=…
EOF
git add secret.txt
```

Keeping secrets

```
cat <<EOF > secret.txt
ACCESS_KEY=…
SECRET_KEY=…
EOF
git add secret.txt
```

```
export ACCESS_KEY=…
export SECRET_KEY=…
```

Keeping secrets



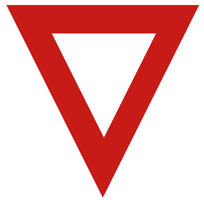
```
cat <<EOF > secret.txt
ACCESS_KEY=…
SECRET_KEY=…
EOF
git add secret.txt
```

```
export ACCESS_KEY=…
export SECRET_KEY=…
```

```
kubectl create secret \
 generic mysecrets \
  --from-file=… \
  --from-file=…
```

Keeping secrets

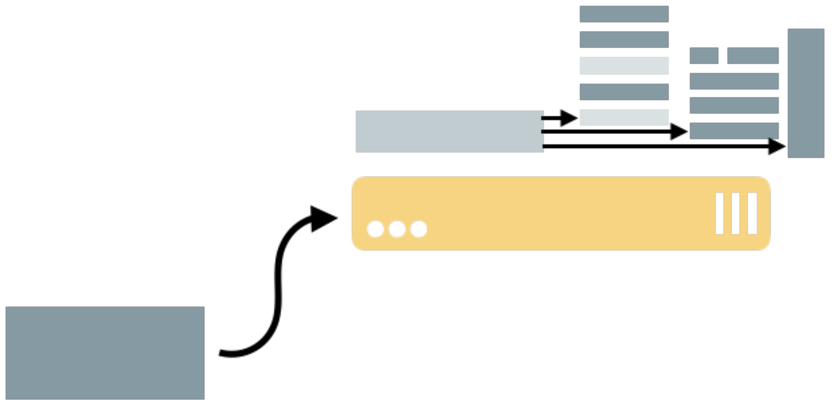
```
cat <<EOF > secret.txt
ACCESS_KEY=…
SECRET_KEY=…
EOF
git add secret.txt
```



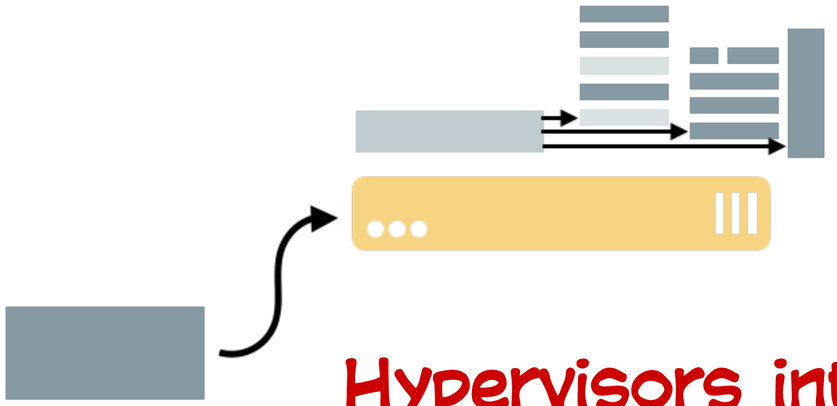
```
export ACCESS_KEY=…
export SECRET_KEY=…
```

```
kubectl create secret \
 generic mysecrets \
  --from-file=… \
  --from-file=…
```

Potential performance pitfalls

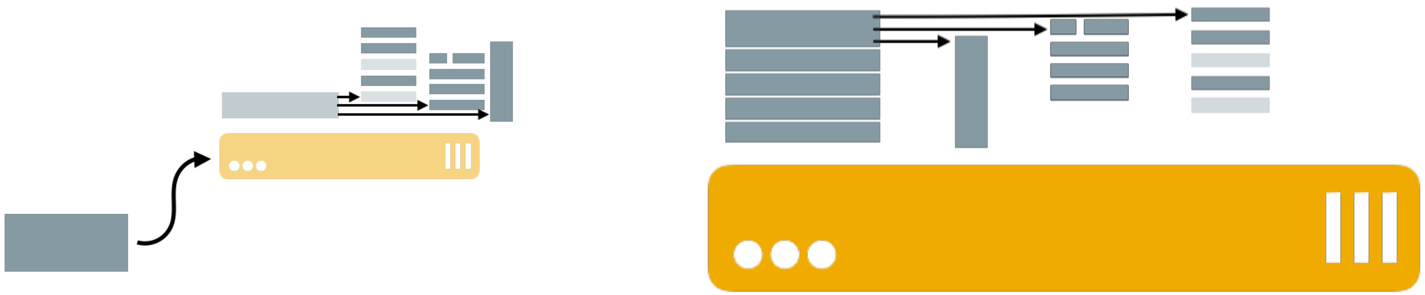


Potential performance pitfalls

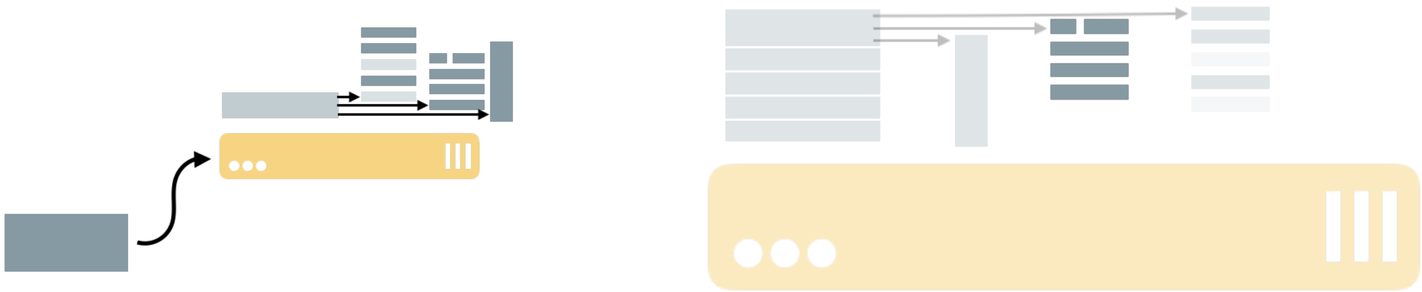


Hypervisors introduce overhead.
Use more lightweight isolation mechanisms to preserve performance.

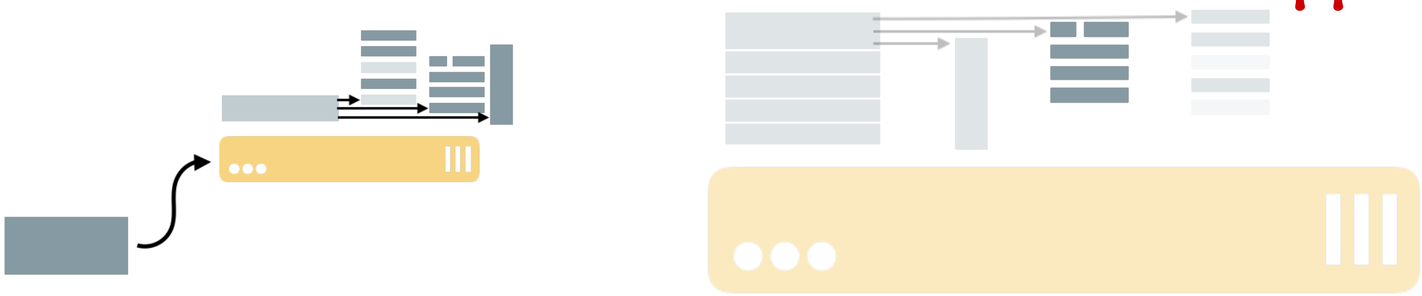
Potential performance pitfalls



Potential performance pitfalls

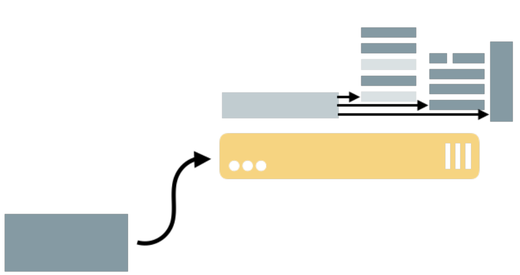


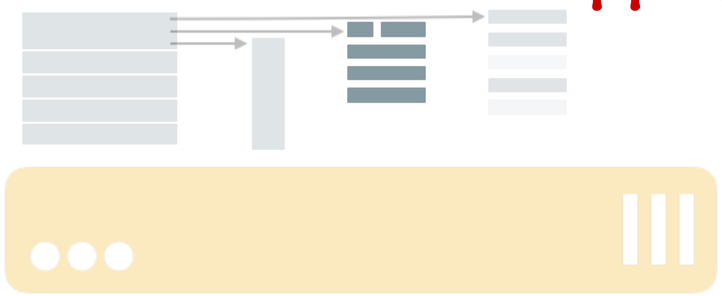
Potential performance pitfalls



Virtualized networking likely has minimal impact on overall application performance!

Potential performance pitfalls





Virtualized networking likely has minimal impact on overall application performance!

...but measure the performance of your I/O configuration!

Potential performance pitfalls

Potential performance pitfalls

Potential performance pitfalls

Quotas mean some ubiquitous techniques can have surprising performance impact. Consider in particular GC configuration and disk buffer cache use.

Potential performance pitfalls

If you use a recent build of OpenJDK 8 or 9,
you have a container-aware JVM!

(If not, set limits manually.)

Container-aware JVM features

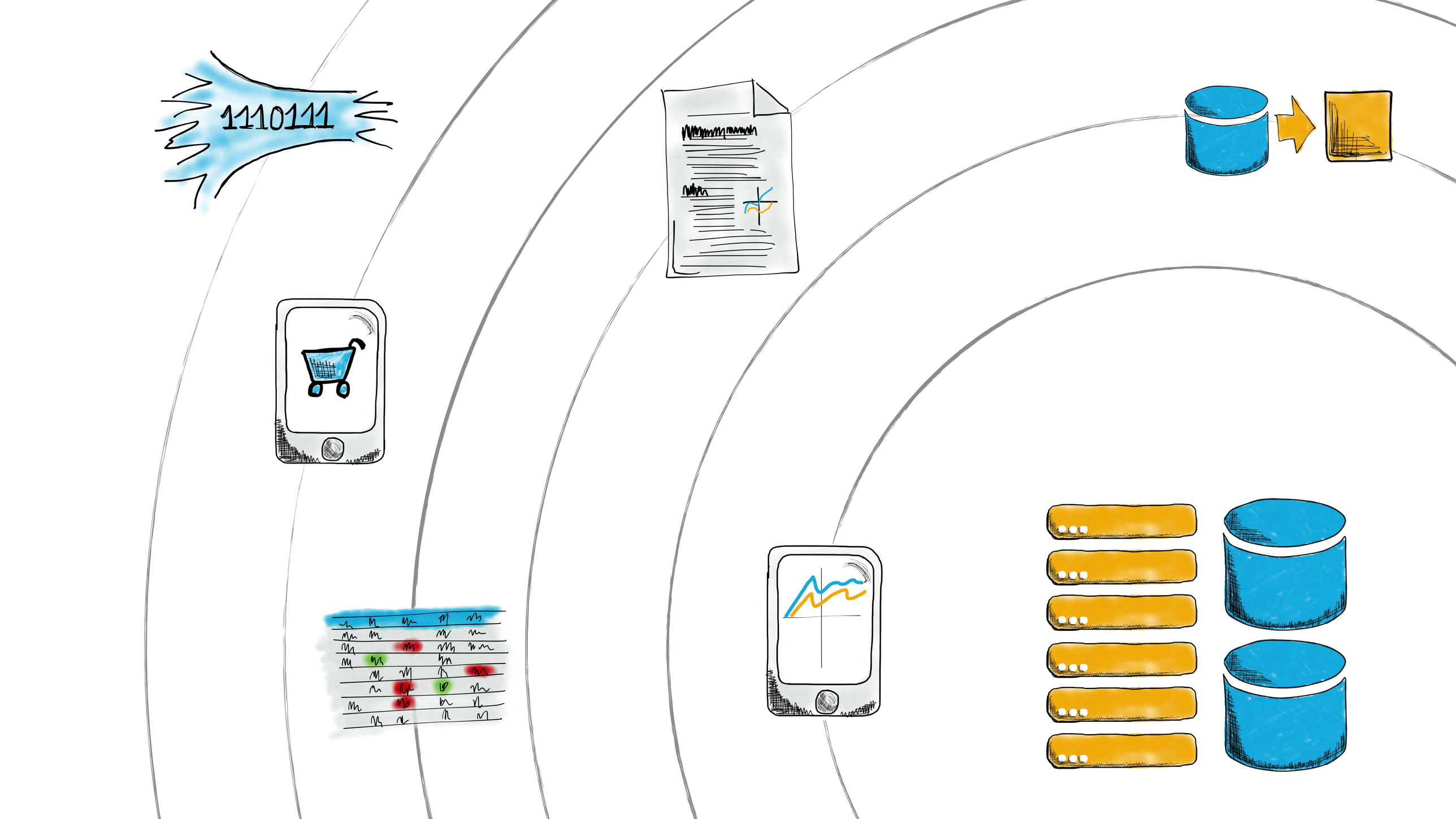
Do the right thing for memory...

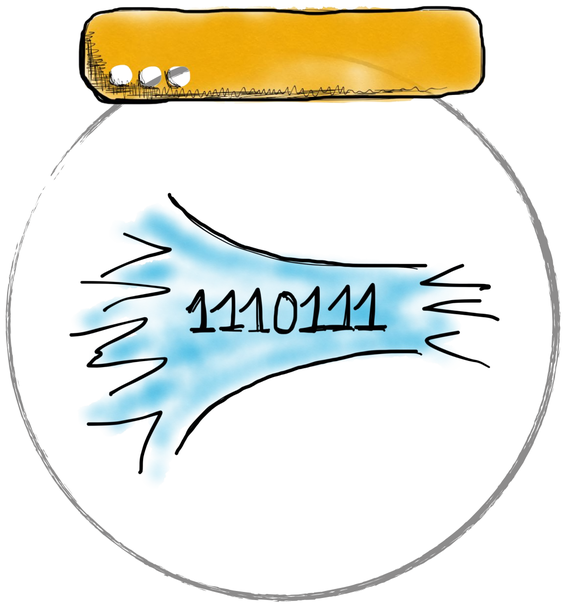
```
java -XX:+UnlockExperimentalVMOptions \
     -XX:+UseCGroupMemoryLimitForHeap
```

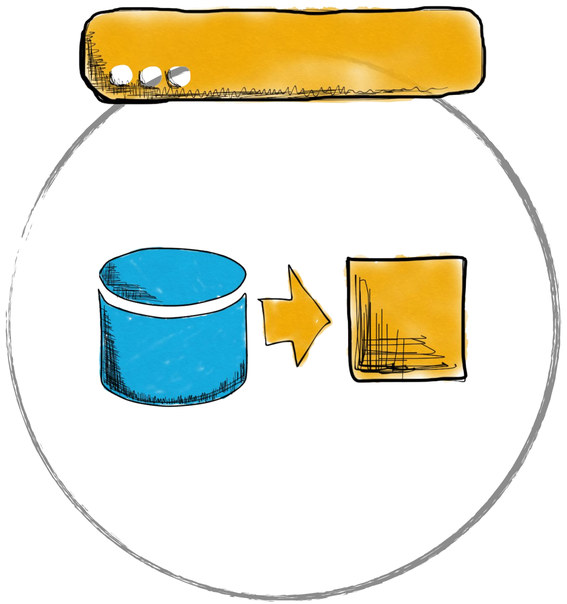
...and for CPU!

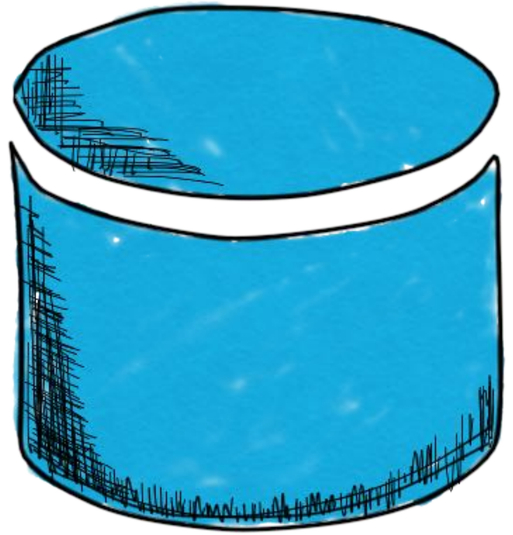
```
Runtime.getRuntime().availableProcessors()
```

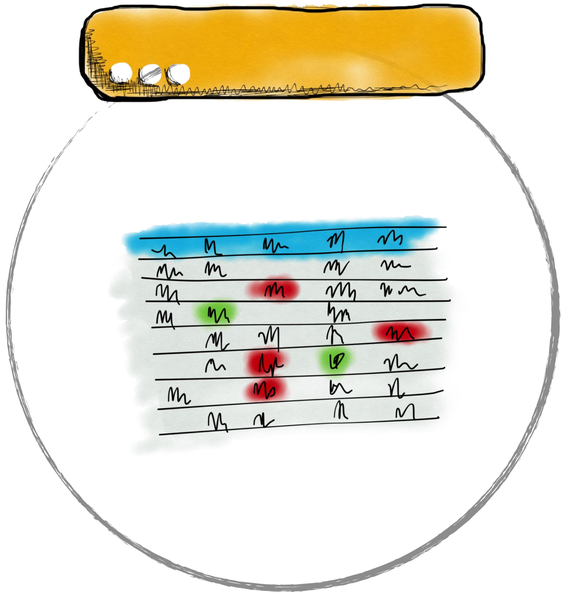
CONCLUSIONS AND WHERE TO GO FROM HERE

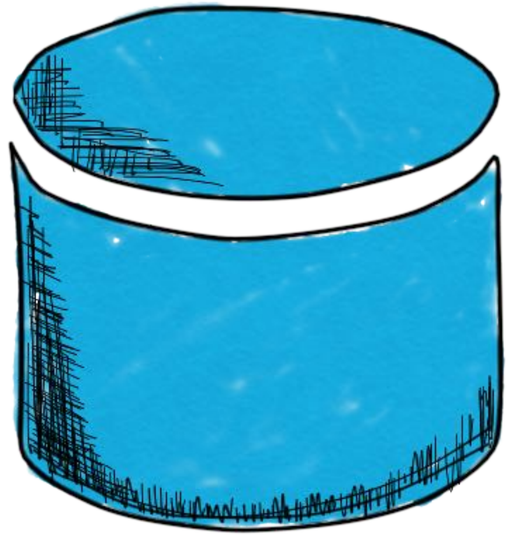

1110111


1110111









Architectural takeaways

Many frameworks are already **cloud-native**

Use a **single compute cluster per app**

Storage lives outside containers and is **accessed through service interfaces**

Correctness takeaways

Arbitrary code isn't safe just because it's in a container, so **don't run code in containers** as root

Use SELinux to minimize exposure to error and malice

Avoid ad hoc mechanisms for configuring secrets

Ephemeral user IDs may **confuse your framework**

Performance takeaways

Don’t use hypervisors for isolation

Virtualized networking is **probably not a concern**

Optimizations that work **well outside of containers** may **have surprising consequences inside containers!**

How to get started

Visit **https://radanalytics.io** for tooling, a containerized Spark distribution, and example applications

Stay in touch!

THANKS!

also: WE ARE HIRING

willb@redhat.com
https://chapeau.freevariable.com
https://radanalytics.io