Serverless Computation with OpenLambda

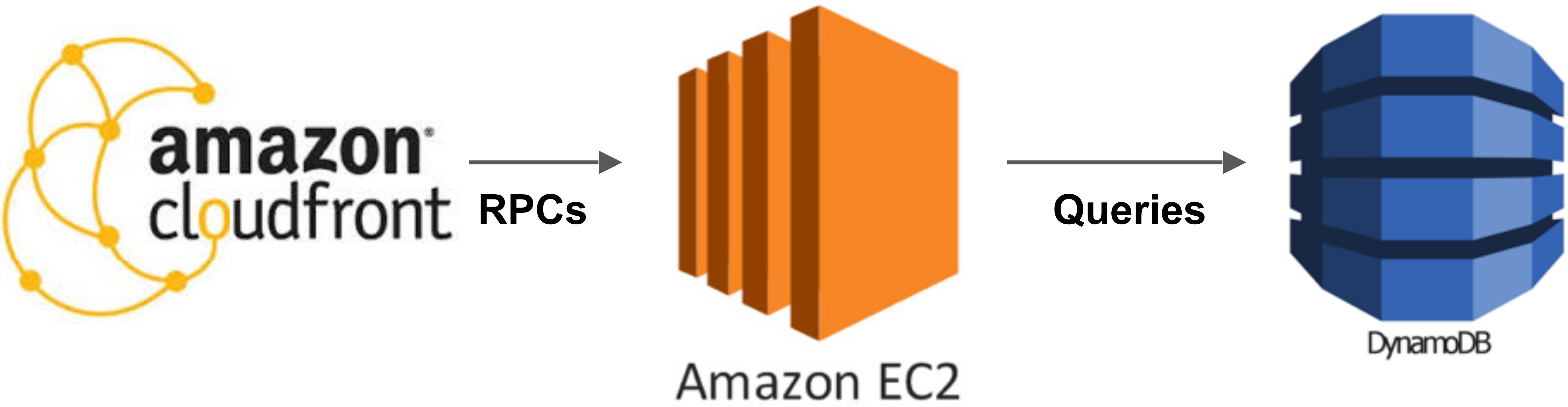
Web development in the cloud

CDN: static content
(e.g., JavaScript)

Compute: dynamic logic
(e.g., Python)

Storage:
application data

Web development in the cloud



CDN: static content
(e.g., JavaScript)

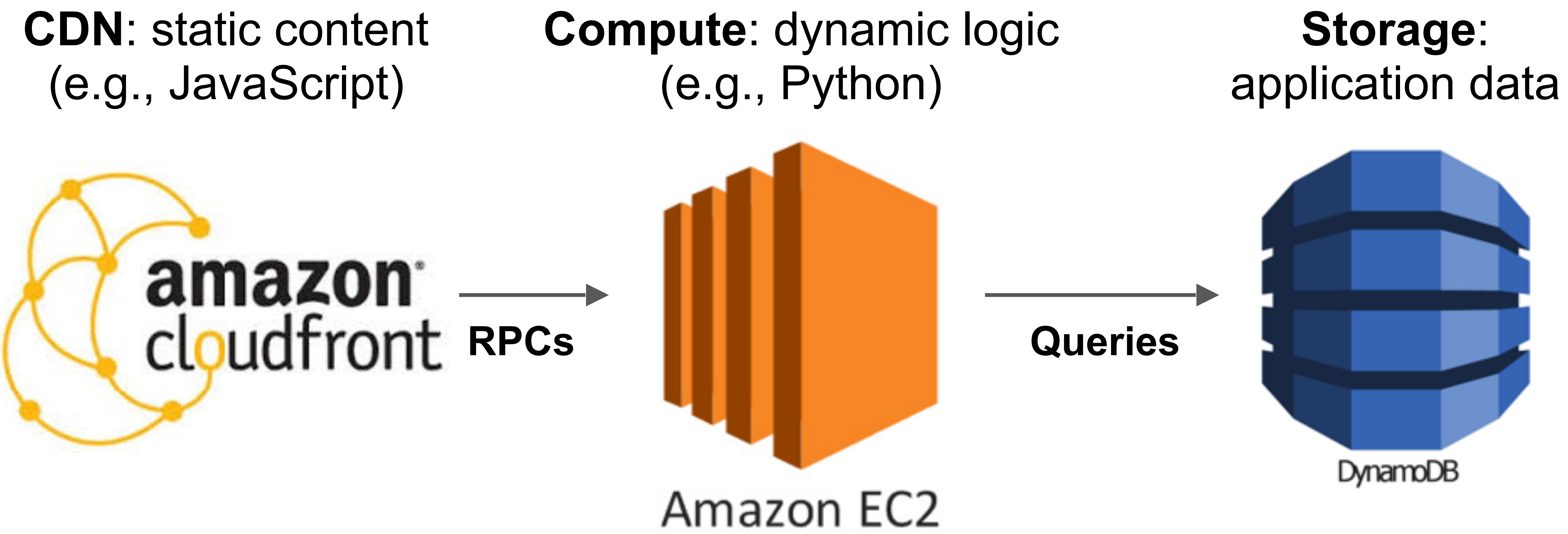
RPCs

Compute: dynamic logic
(e.g., Python)

Queries

Storage:
application data

Web development in the cloud



CDN: static content (e.g., JavaScript)

Compute: dynamic logic (e.g., Python)

Storage: application data

RPCs

Queries

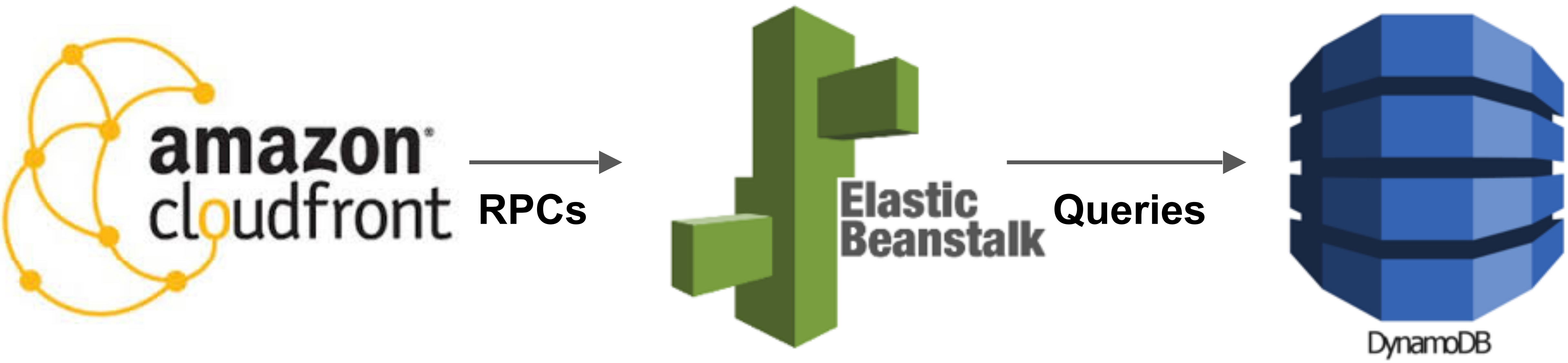
compute is evolving

Web development in the cloud

CDN: static content (e.g., JavaScript)

Compute: dynamic logic (e.g., Python)

Storage: application data



RPCs

Queries

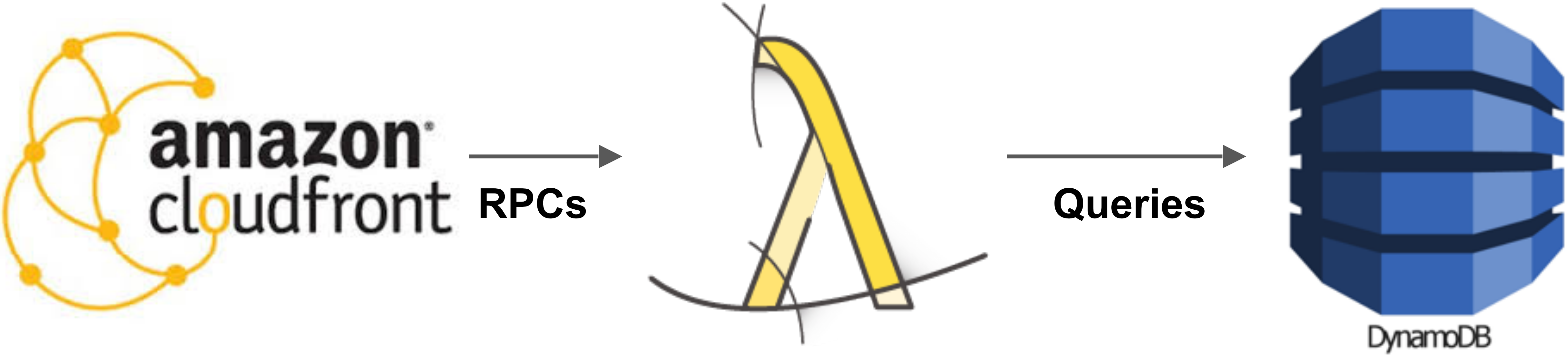
compute is evolving

Web development in the cloud

CDN: static content (e.g., JavaScript)

Compute: dynamic logic (e.g., Python)

Storage: application data



RPCs

Queries

AWS Lambda

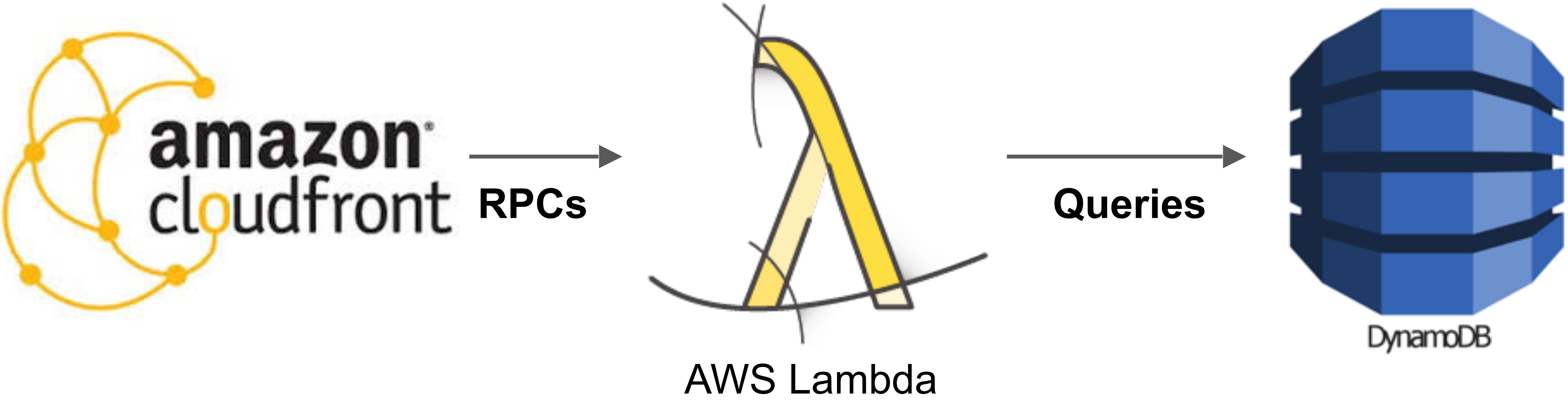
compute is evolving

Web development in the cloud

CDN: static content (e.g., JavaScript)

Compute: dynamic logic (e.g., Python)

Storage: application data



RPCs

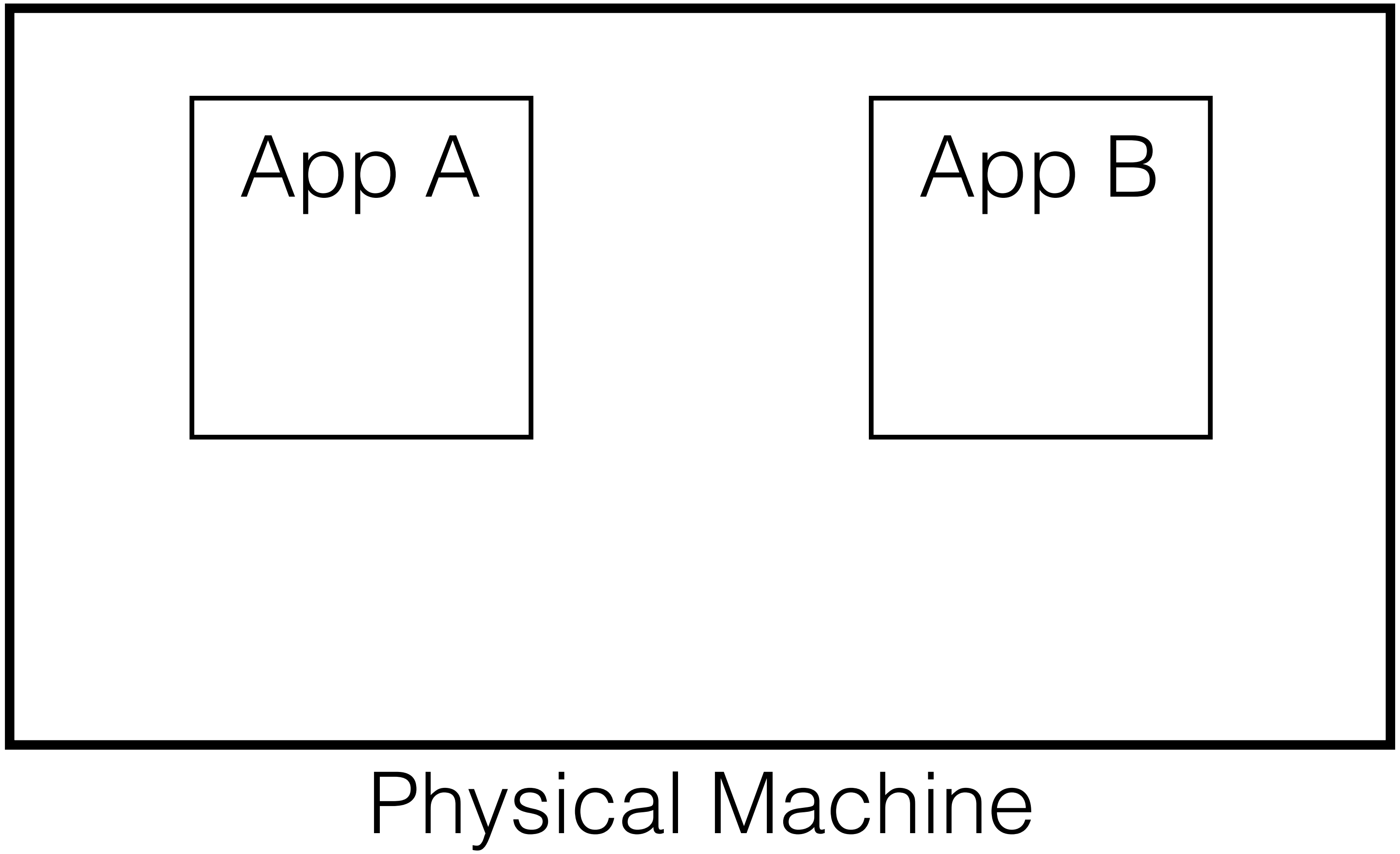
Queries

AWS Lambda

claim: prior to the Lambda model, cloud compute was neither elastic nor pay-as-you-go

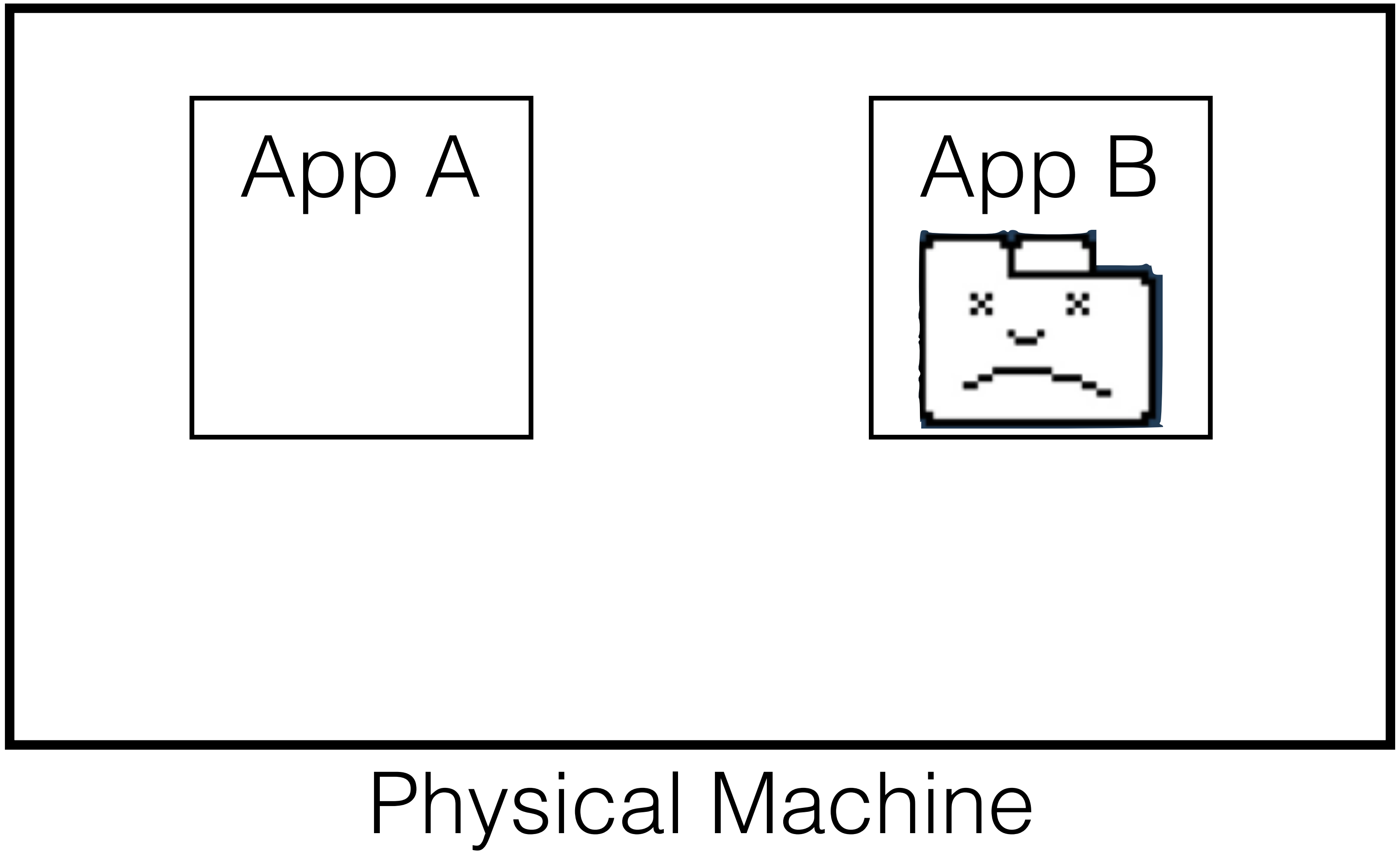
What do we expect from a cloud computing platform?

Big goal: sharing and isolation



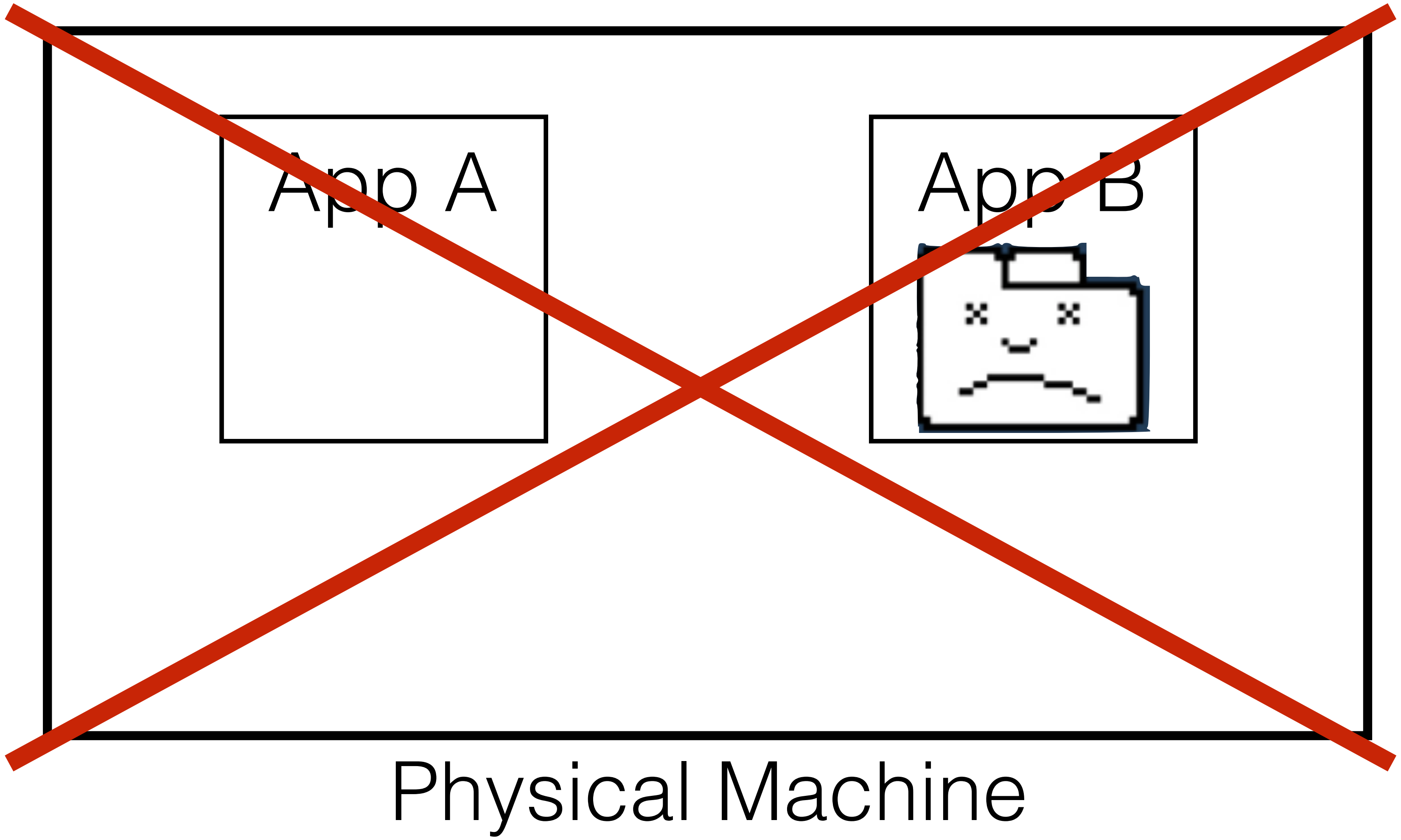
want: multitenancy

Big goal: sharing and isolation



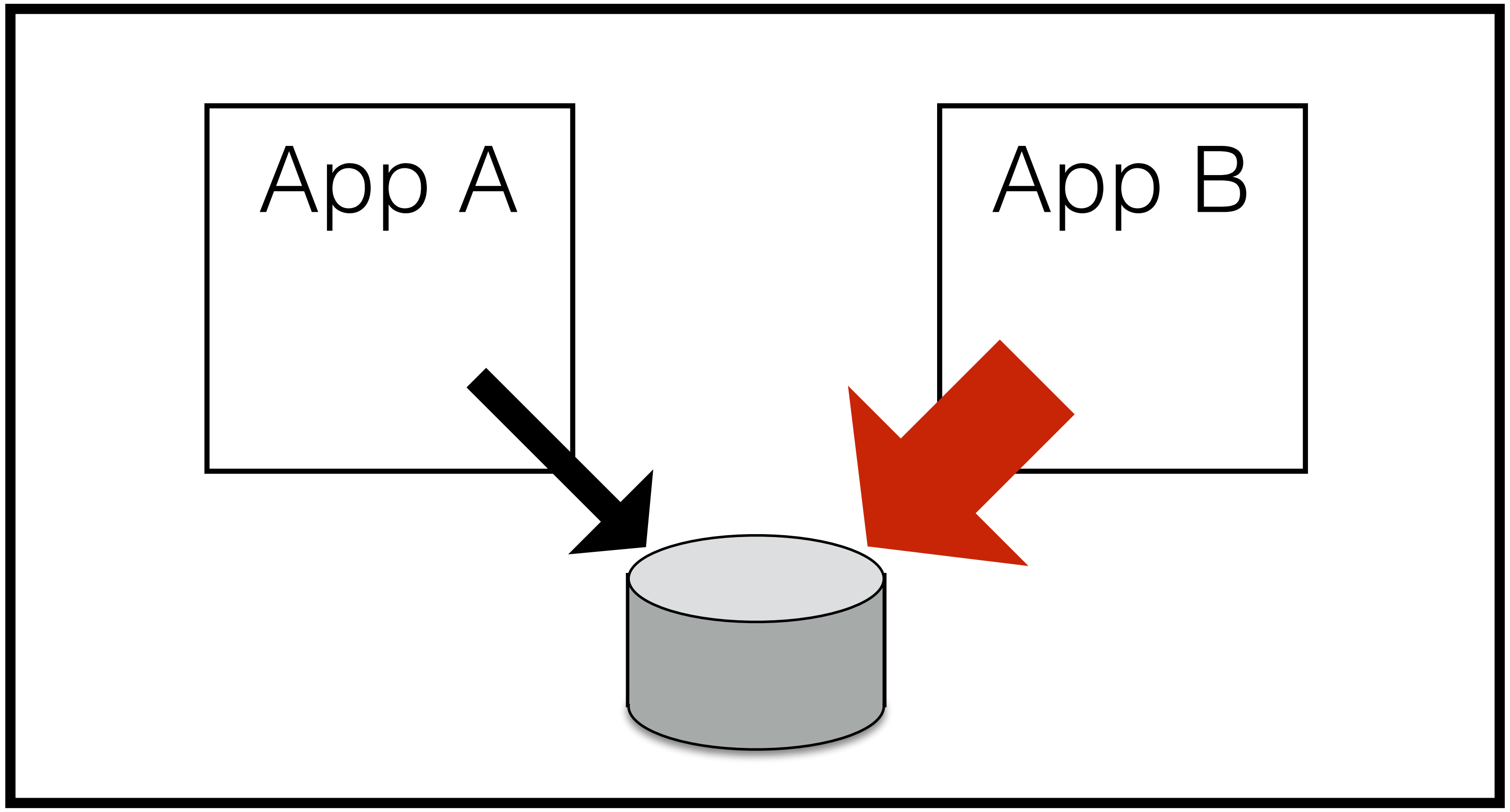
don’t want: crashes

Big goal: sharing and isolation



don’t want: crashes

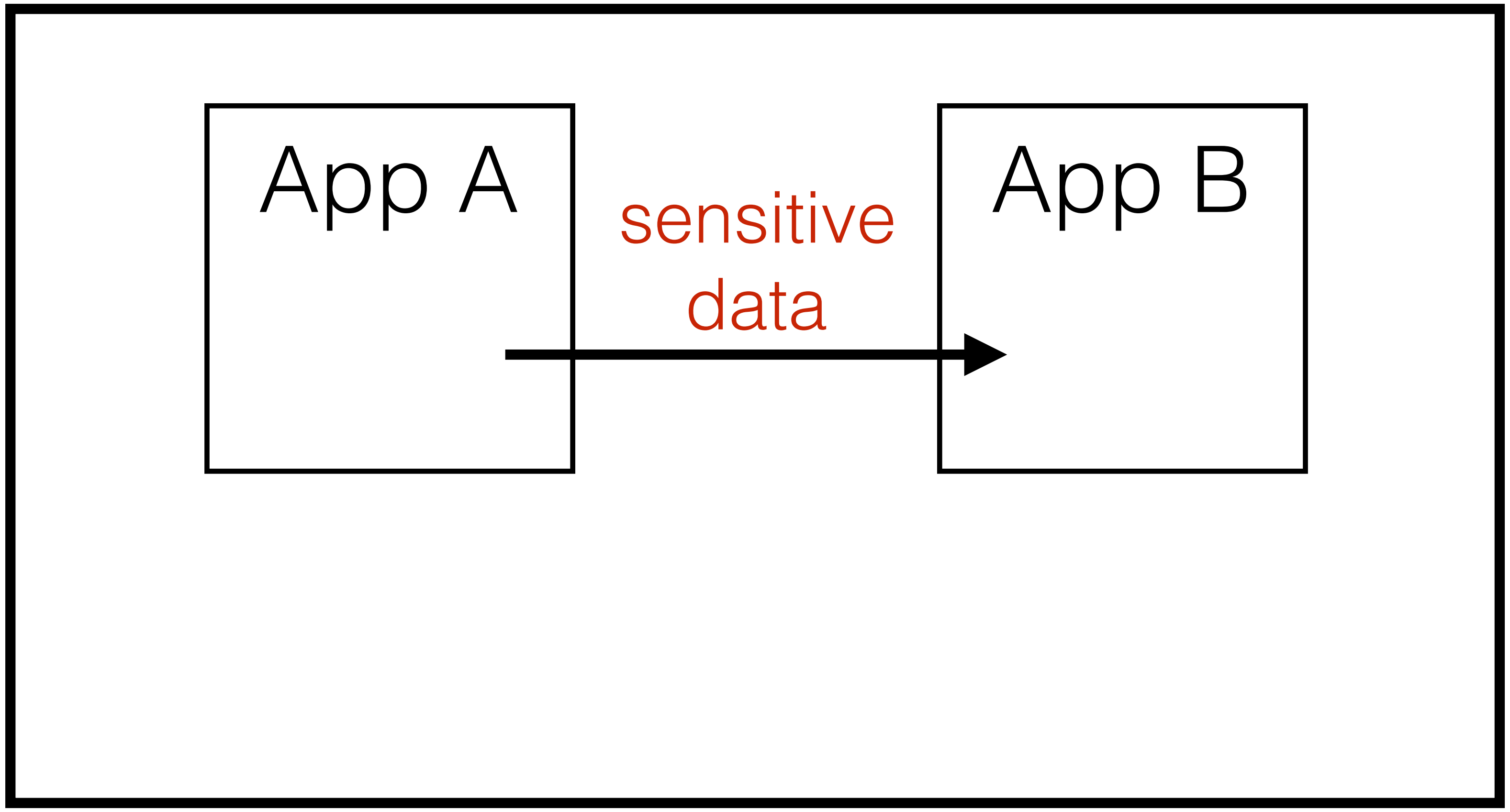
Big goal: sharing and isolation



Physical Machine

don’t want: unfairness

Big goal: sharing and isolation



Physical Machine

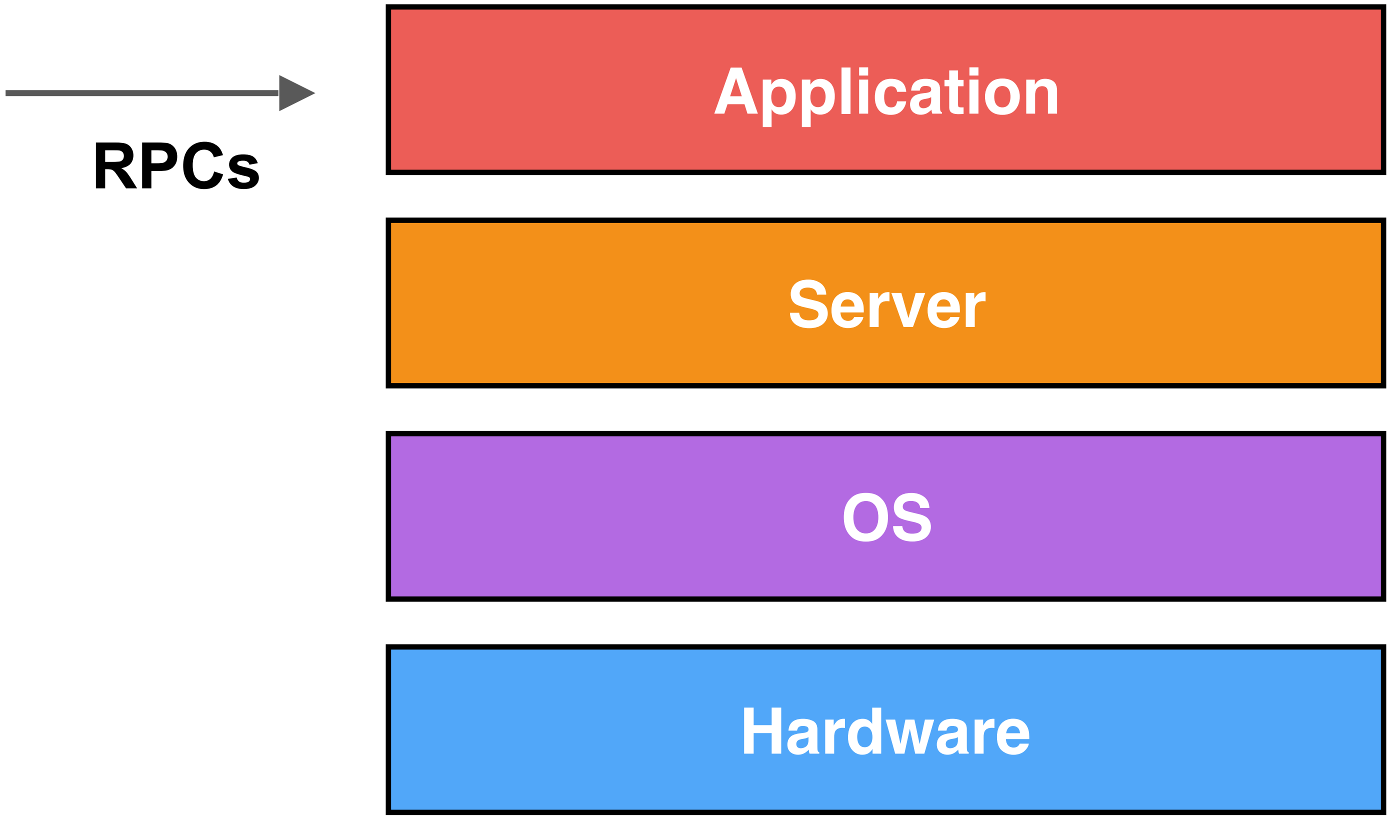
don’t want: leaks

Solution: Virtualization

namespaces and **scheduling** provide illusion of private resources

But what to virtualize?

Web application without virtualization



RPCs

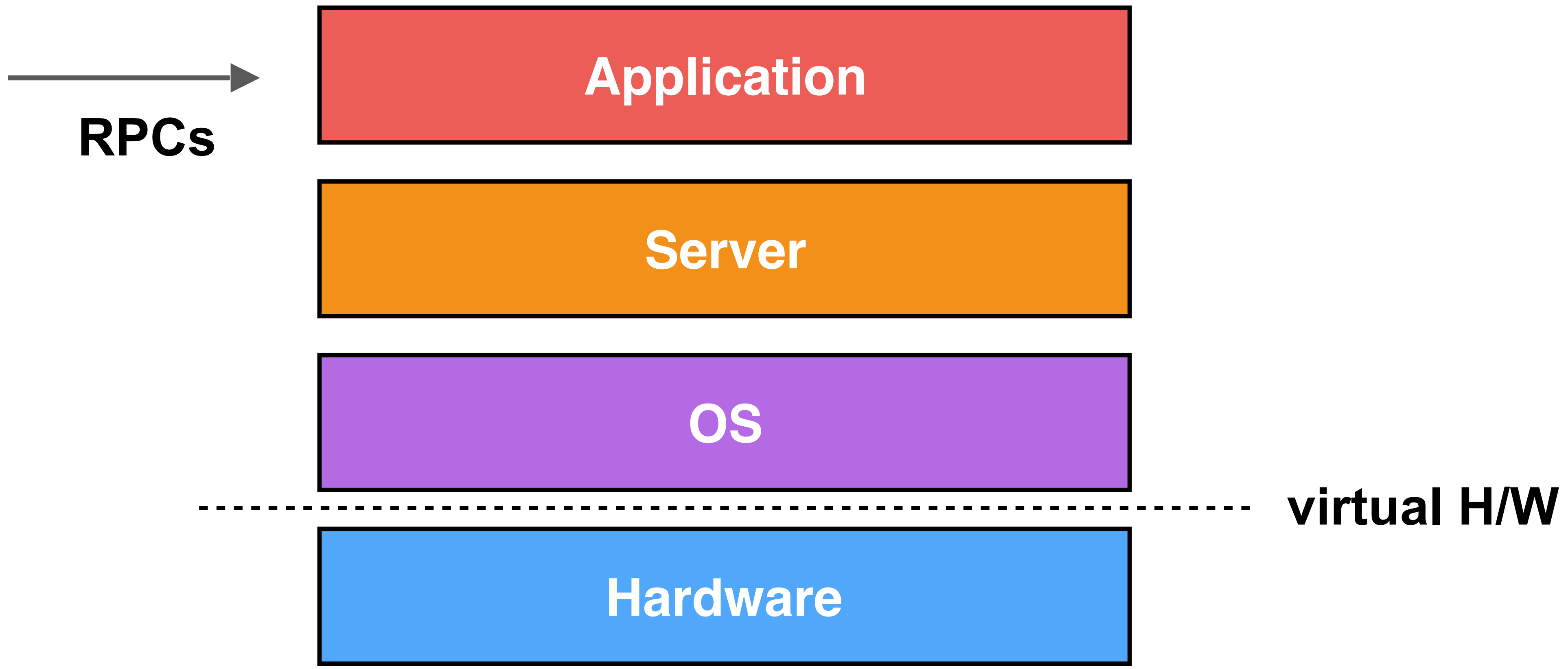
Application

Server

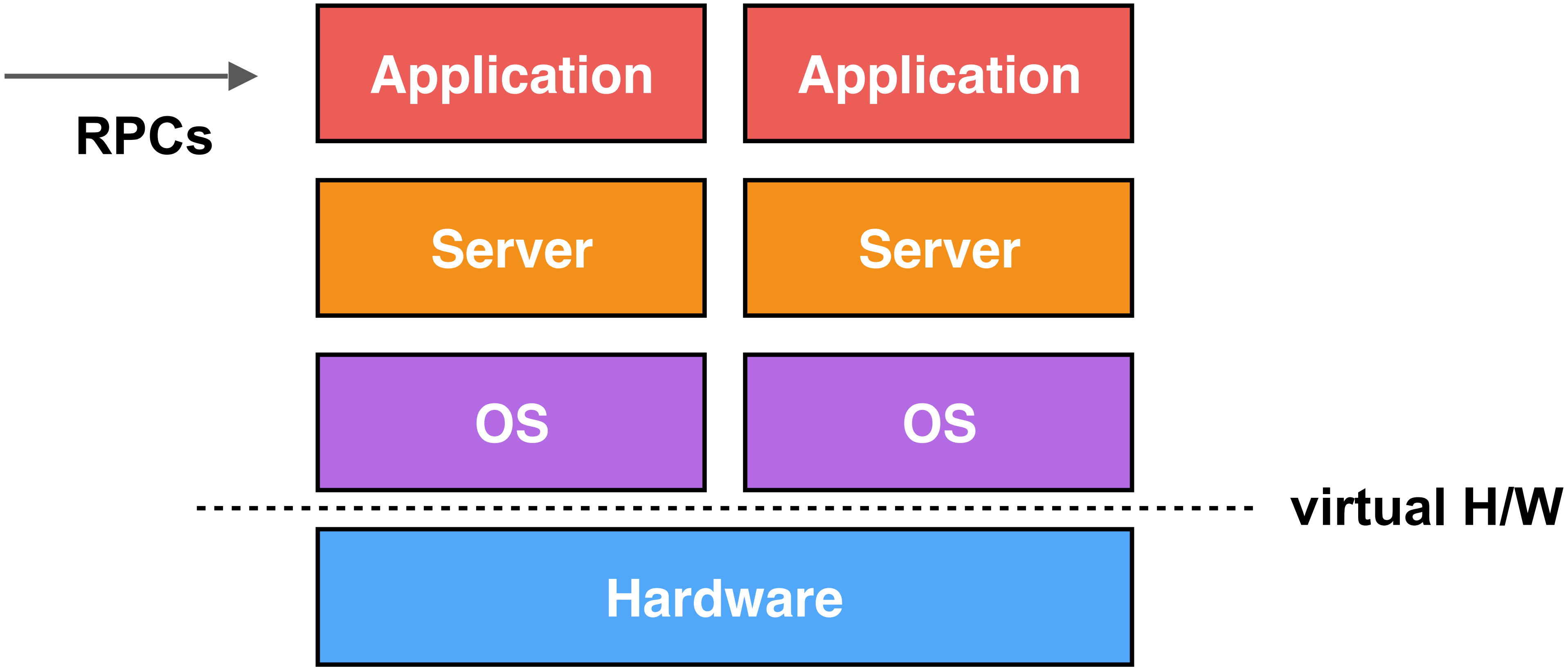
OS

Hardware

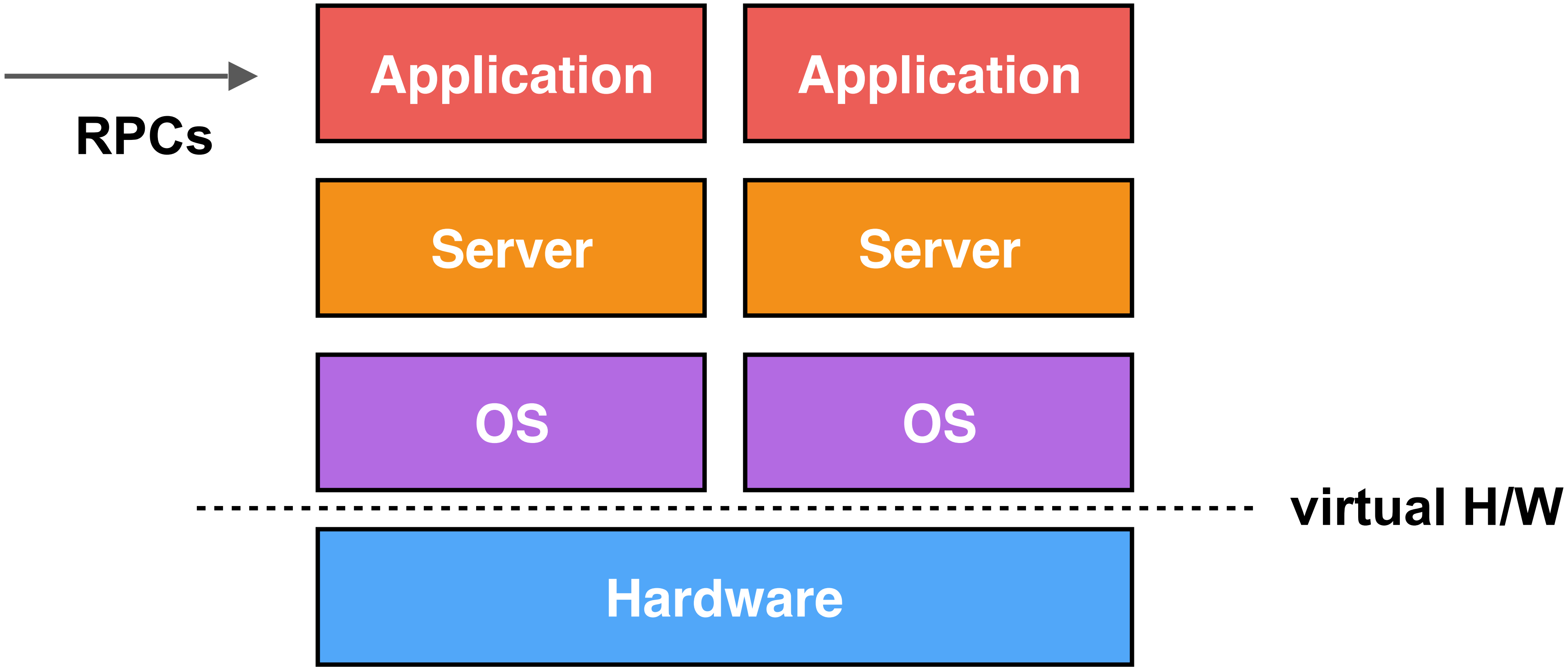
1st generation: virtual machines



1st generation: virtual machines



1st generation: virtual machines



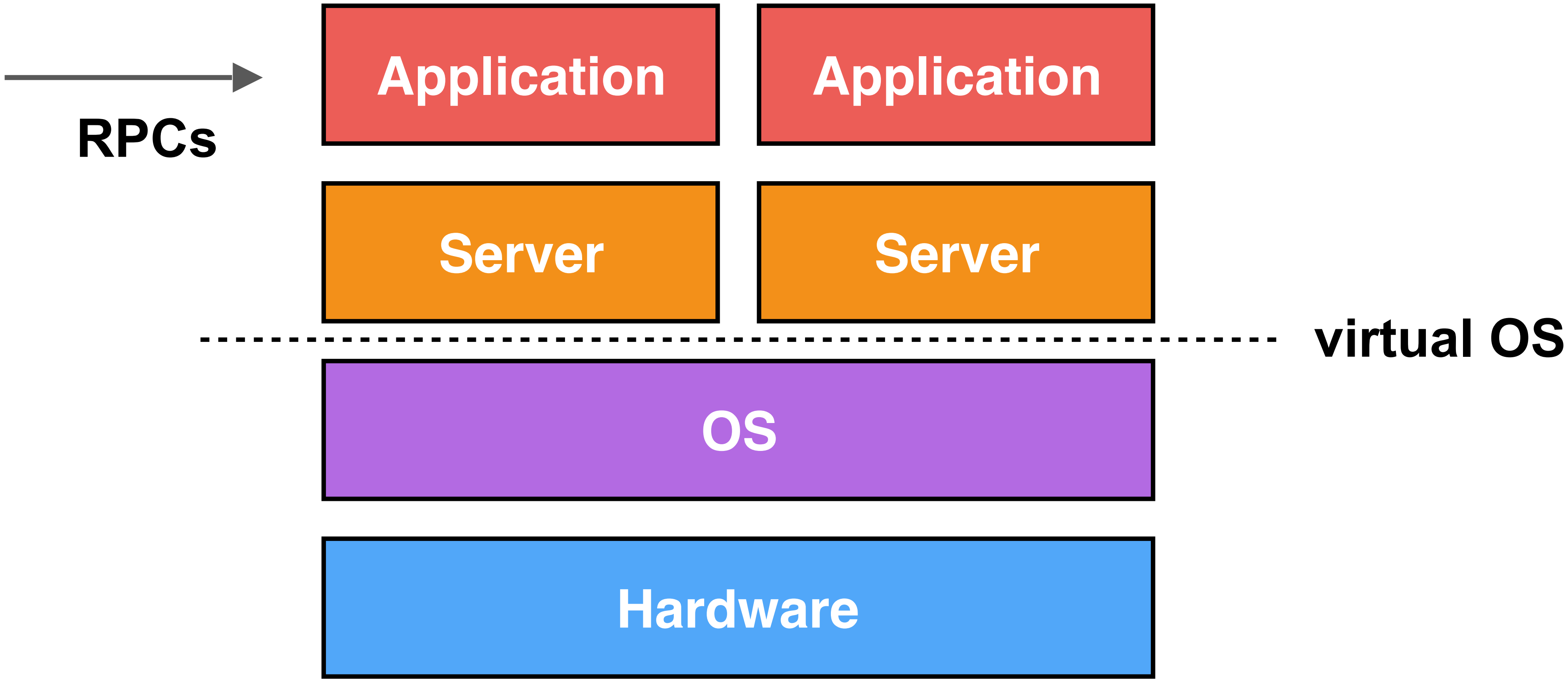
advantages:

- very flexible
- use any OS

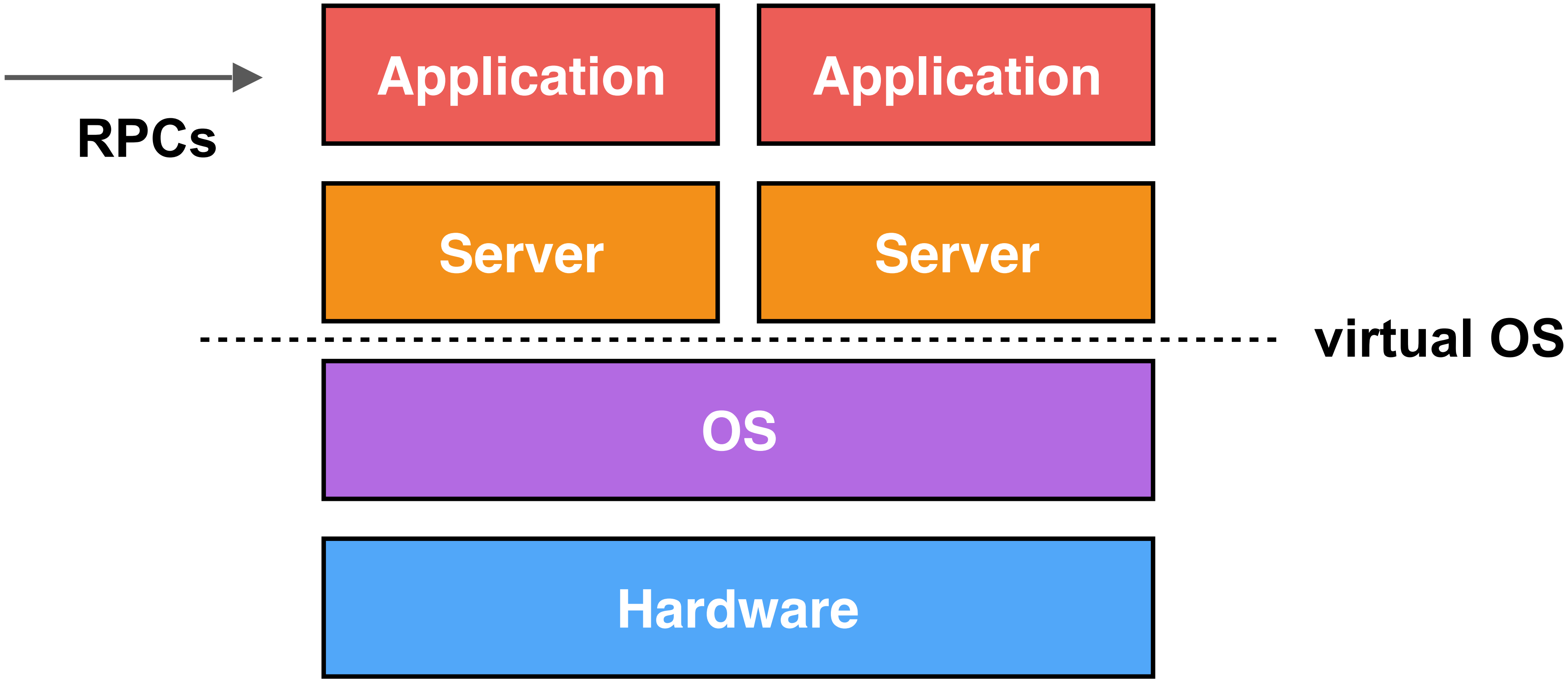
problems:

- interposition
- is RAM used? (ballooning)
- redundancy (e.g., FS journal)

2nd generation: containers



2nd generation: containers



advantages:

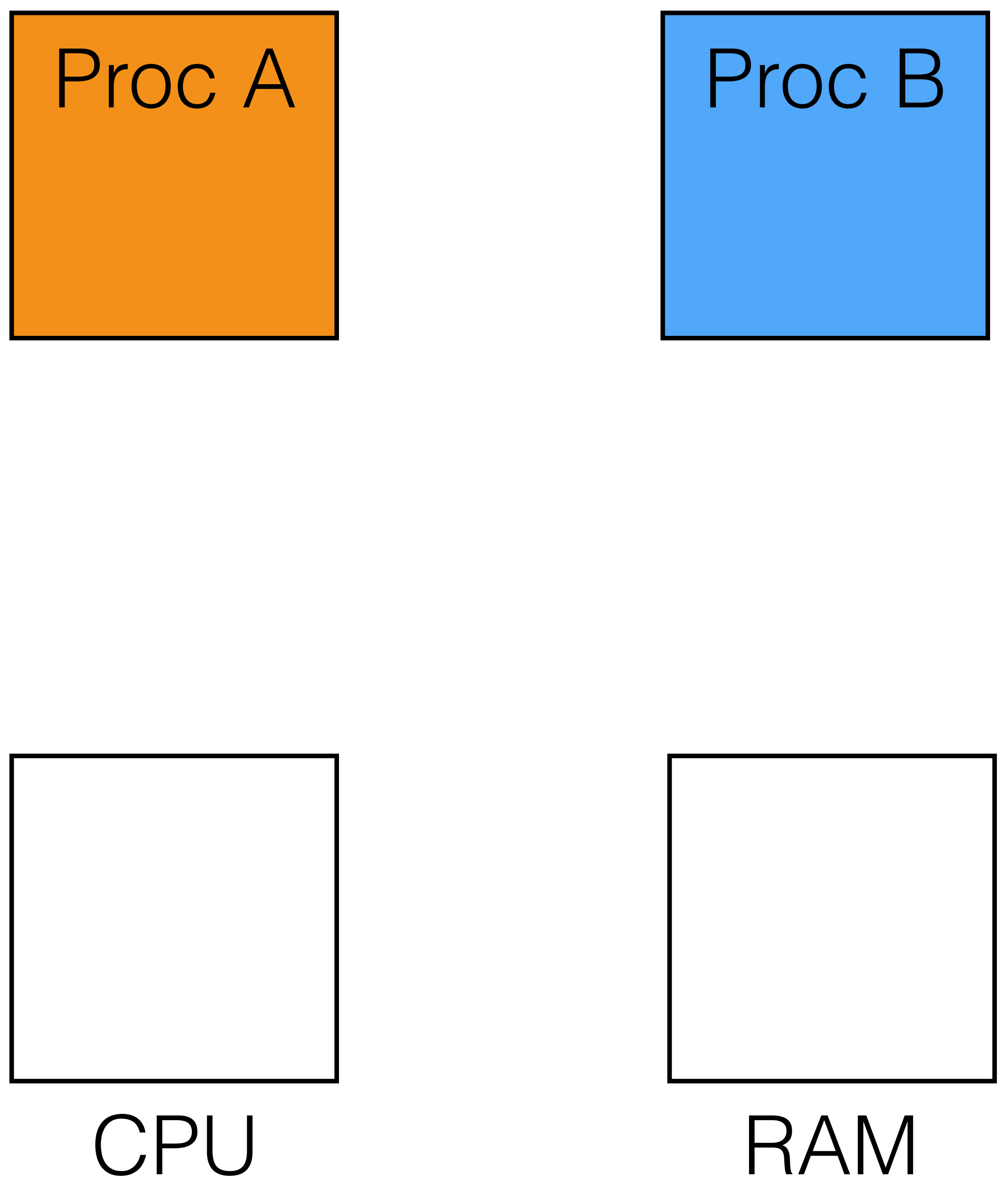
- centralized view
- init H/W once

problems:

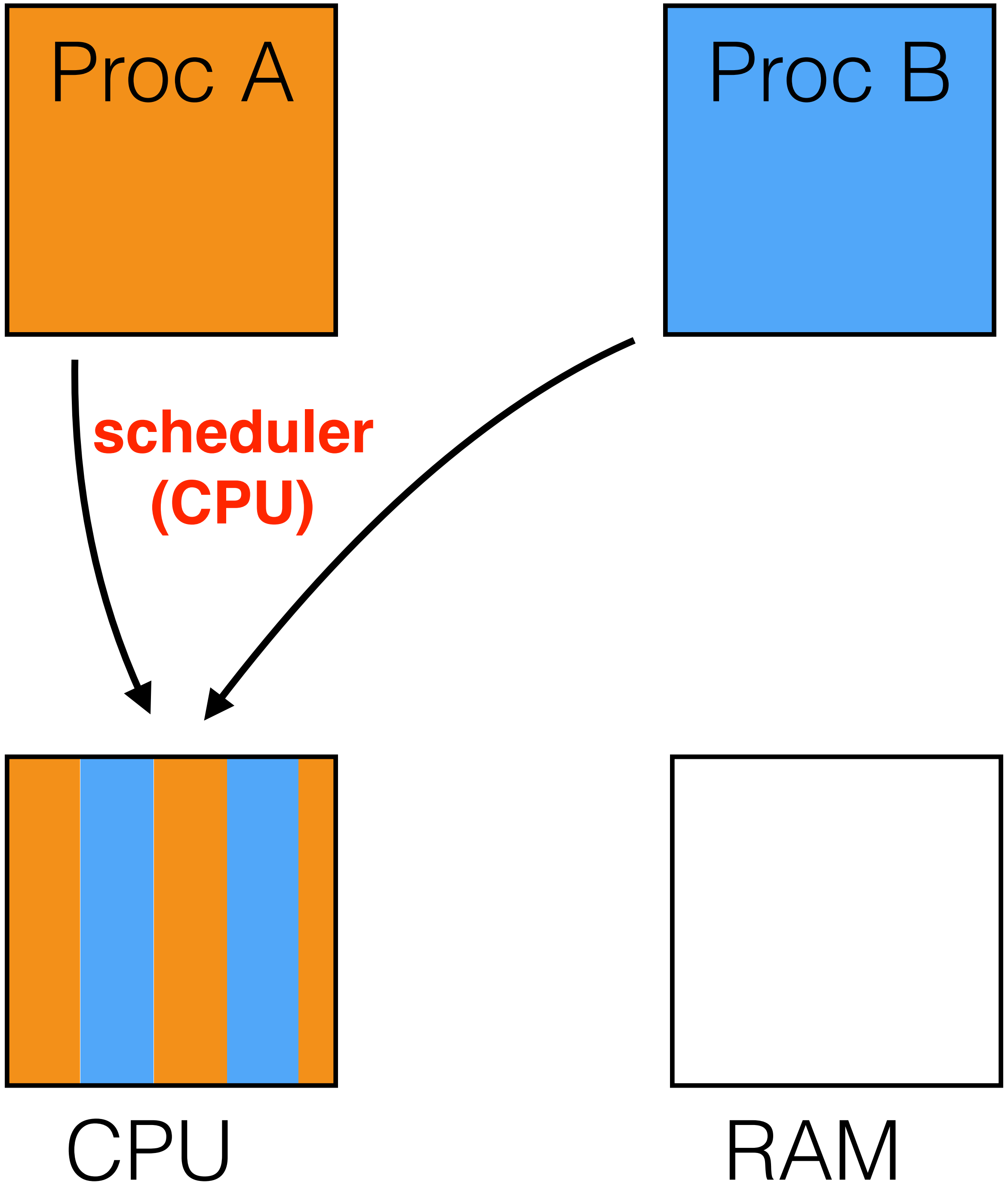
- large deployment bundle
- server spinup

How should we virtualize the OS?

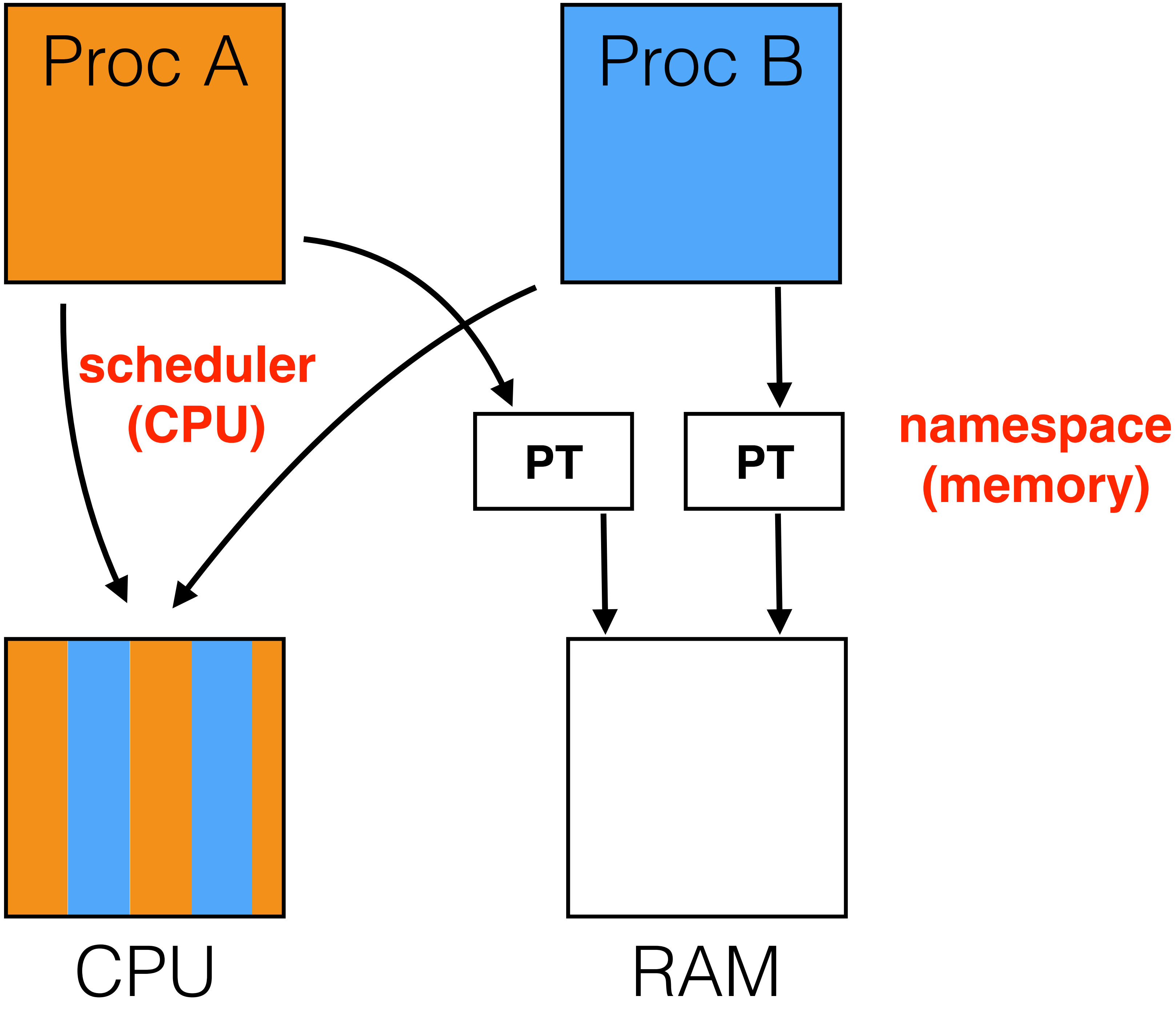
Operating systems have long provided process virtualization



Operating systems have long provided process virtualization



Operating systems have long provided process virtualization



OS virtualization

Operating systems have long virtualized CPU and memory

But many resources have not been historically virtualized:

- file system mounts
- network
- host names
- IPC queues
- process IDs
- user IDs

OS virtualization

Operating systems have long virtualized CPU and memory

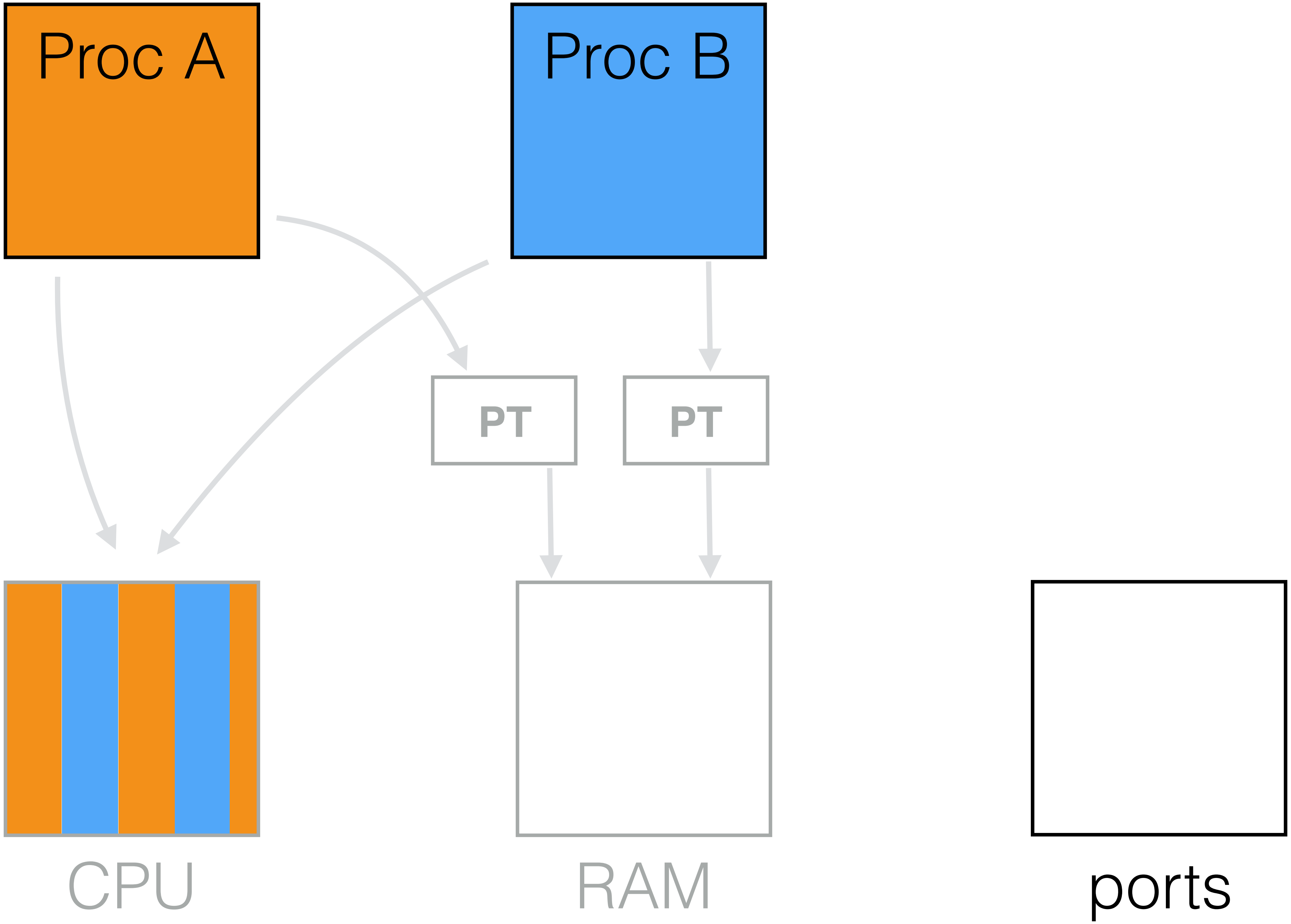
But many resources have not been historically virtualized:

- file system mounts
- network
- host names
- IPC queues
- process IDs
- user IDs

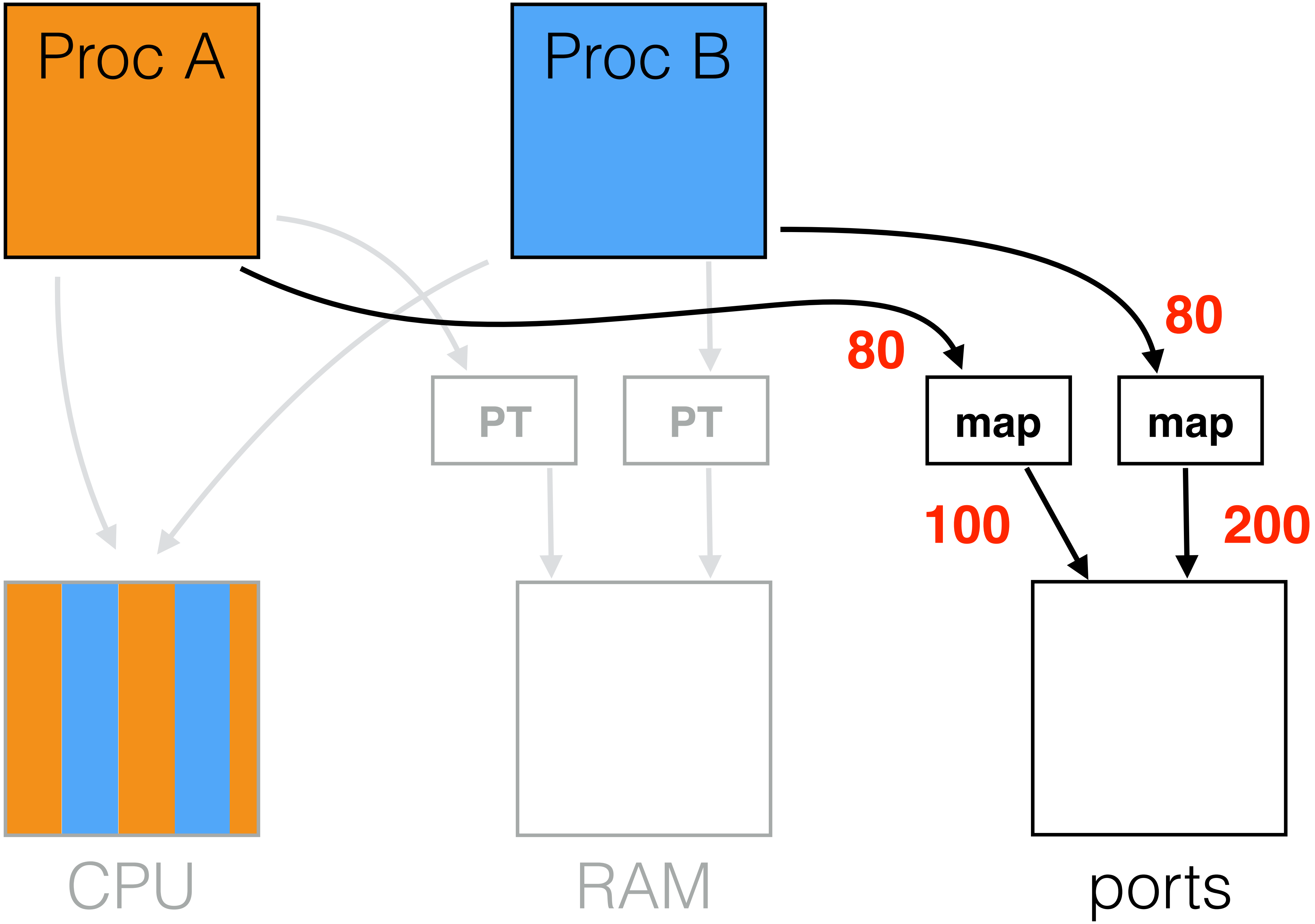
New namespaces are collectively called “containers”

- lightweight, like virtual memory
- old idea rebranded (Plan 9 OS)

Containers should be fast and simple



Containers should be fast and simple



Theory and practice

Theory: containers are lightweight

- just like starting a process!

Theory and practice

Theory: containers are lightweight

- just like starting a process!

Practice: container startup is slow

- **25 second** startup time [1]

> “ *task startup latency (the time from job submission to a task running) is an area that has received and continues to receive significant attention. It is highly variable, with the median typically about 25 s. **Package installation** takes about 80% of the total: one of the known bottlenecks is **contention for the local disk** where packages are written.* ”

[1] Large-scale cluster management at Google with Borg. http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf

Theory and practice

Theory: containers are lightweight

- just like starting a process!

Practice: container startup is slow

- **25 second** startup time [1]

> “ *task startup latency (the time from job submission to a task running) is an area that has received and continues to receive* ***significant attention****. It is highly variable, with the median typically about 25 s.* ***Package installation*** *takes about 80% of the total: one of the known bottlenecks is* ***contention for the local disk*** *where packages are written.* ”

[1] Large-scale cluster management at Google with Borg.
http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf

Theory and practice

Theory: containers are lightweight

- just like starting a process!

Practice: container startup is slow

- **25 second** startup time [1]

Startup time matters

- flash crowds
- load balance
- interactive development

[1] Large-scale cluster management at Google with Borg.
http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf

How to minimize startup latency?

Strategy: share as much as possible!

- Containers only share H/W and OS

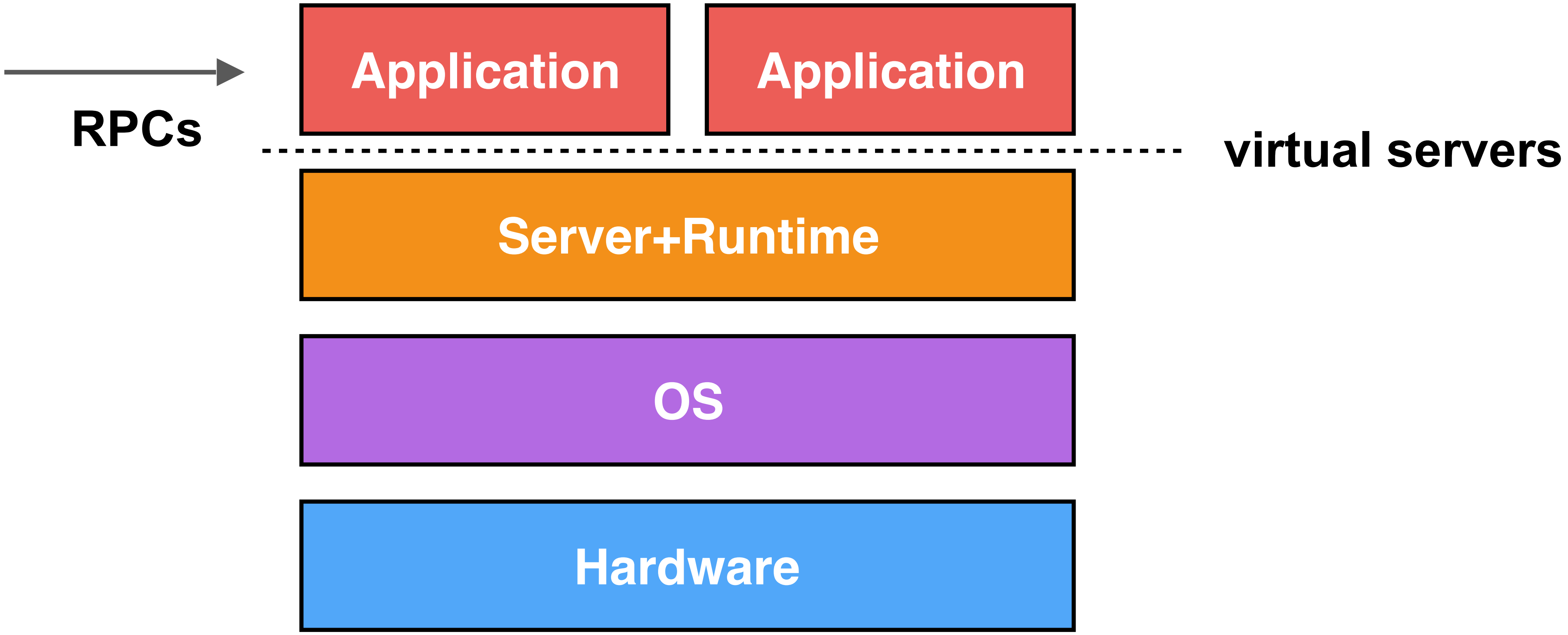
Servers

- Shouldn't need to spin up

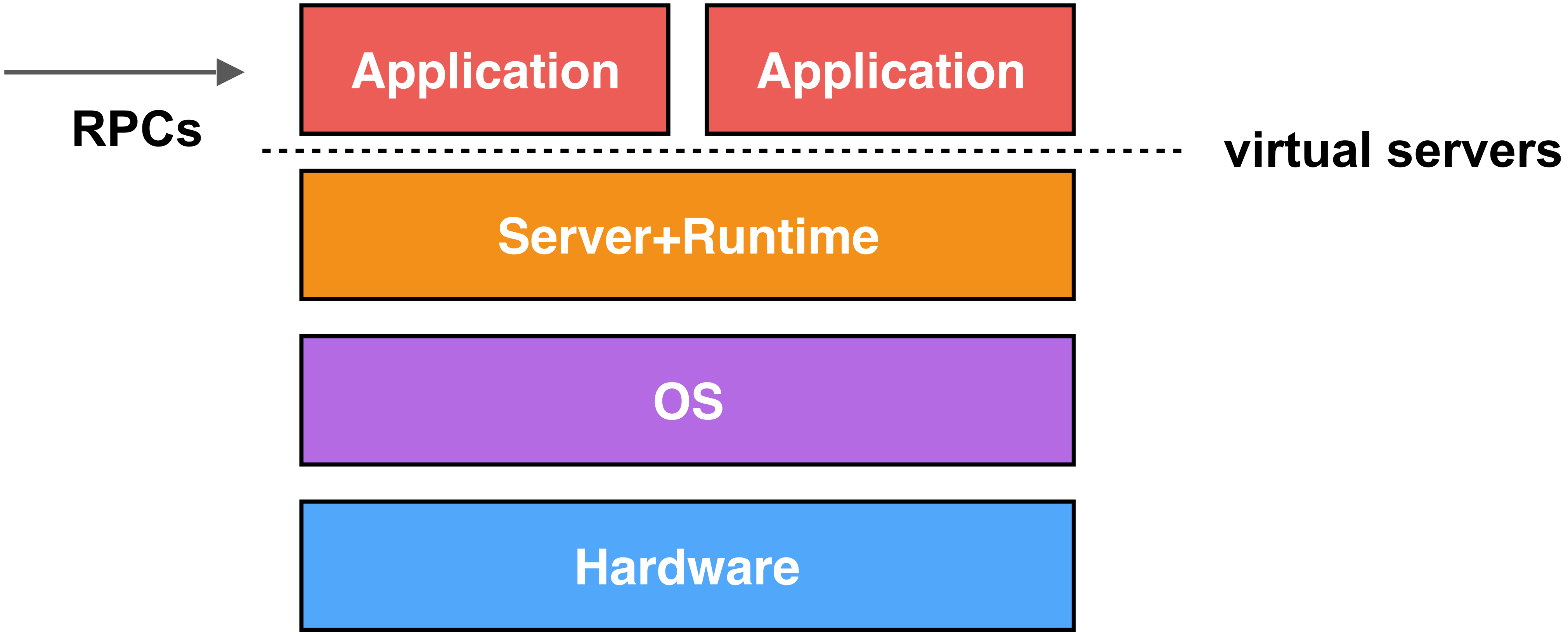
Runtimes

- Interpreter (e.g., Python) and packages
- Should already be in memory

3rd generation: Lambdas

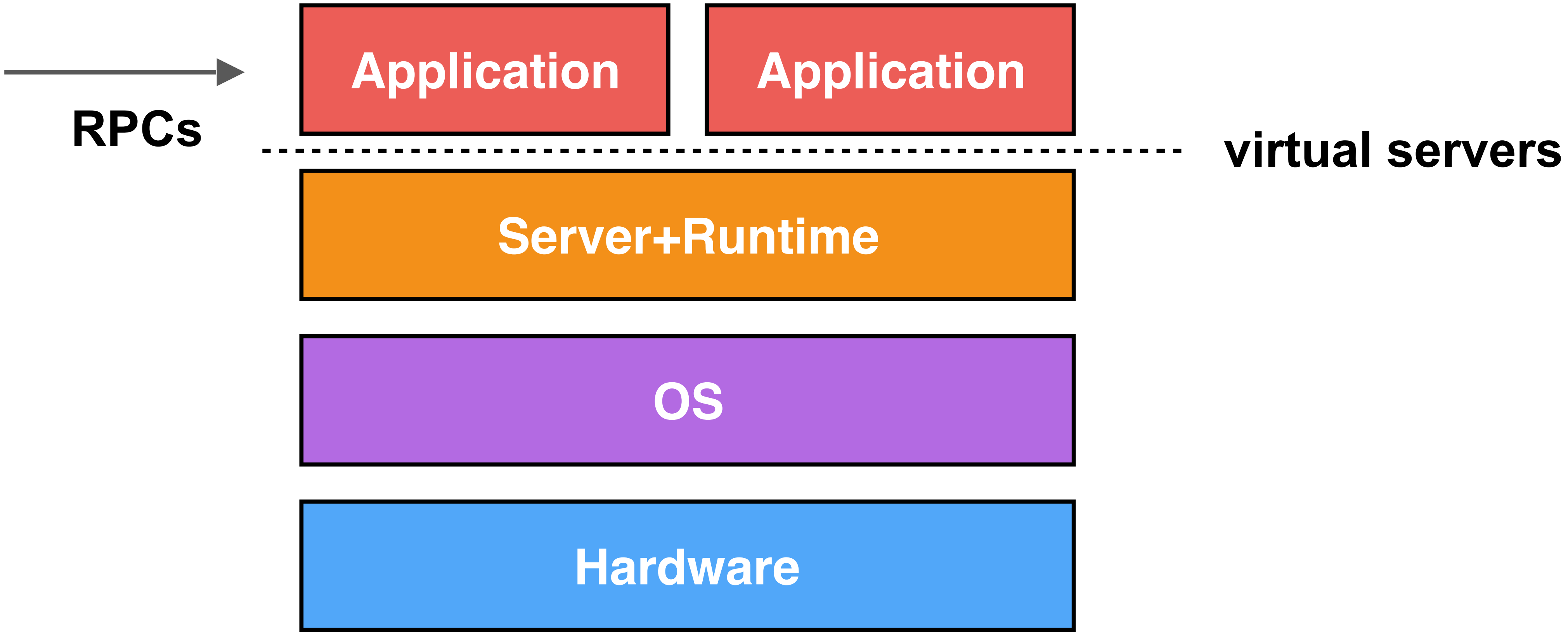


3rd generation: Lambdas



serverless computing

3rd generation: Lambdas



advantages:

- fast startup
- share memory

problems:

- not flexible

Outline

Emerging compute models

Containers vs. Lambdas

Application building

OpenLambda: code overview

Plan projects: discussion

What is it like to develop applications in containers?

A sad story in the cloud

Original app: **EES** (Engineering Equation Solver)

- Desktop application, costs $600
- Iterative equation solver for mechanical eng
- Very compute intensive
- Written in Fortran, very buggy

A sad story in the cloud

Original app: **EES** (Engineering Equation Solver)

- Desktop application, costs $600
- Iterative equation solver for mechanical eng
- Very compute intensive
- Written in Fortran, very buggy

Our app: **EESIER**

- Web application, pay-as-you-go
- Handle compute load bursts with auto-scaling in Google AppEngine

Google AppEngine

Container-based cloud service

Programming model

- Write application as a **web server**
- handle **RPC** calls from JavaScript frontend (e.g., AJAX)

Autoscaling

- Start new server instances as dictated by specified rules

EESIER code

```
from flask import Flask, request
app = Flask(__name__)
import solver

@app.route('/', methods=['GET', 'POST'])
def handle():
    equations = request.form.get('eqs')
    // solve

...
```

RPC handler of server

10s of seconds of compute

Experience

Plan: let students use EESIER instead of EES for H/W

- How to scale?
- How to minimize monetary cost?

Experiment: 10s of concurrent requests

- Starting new servers took minutes
- Not enough are started
- After a burst, you keep paying

Experience

Plan: let students use EESIER instead of EES for H/W

- How to scale?
- How to minimize monetary cost?

Experiment: 10s of concurrent requests

- Starting new servers took minutes
- Not enough are started
- After a burst, you keep paying

Conclusion: AppEngine is

- Not elastic
- Not pay-as-you-go

Experience

Plan: let students use EESIER instead of EES for H/W

- How to scale?
- How to minimize monetary cost?

Experiment: 10s of concurrent requests

- Starting new servers took minutes
- Not enough are started
- After a burst, you keep paying

Conclusion: AppEngine is

- Not elastic
- Not pay-as-you-go

Is AWS Elastic Beanstalk better?

Elastic Beanstalk

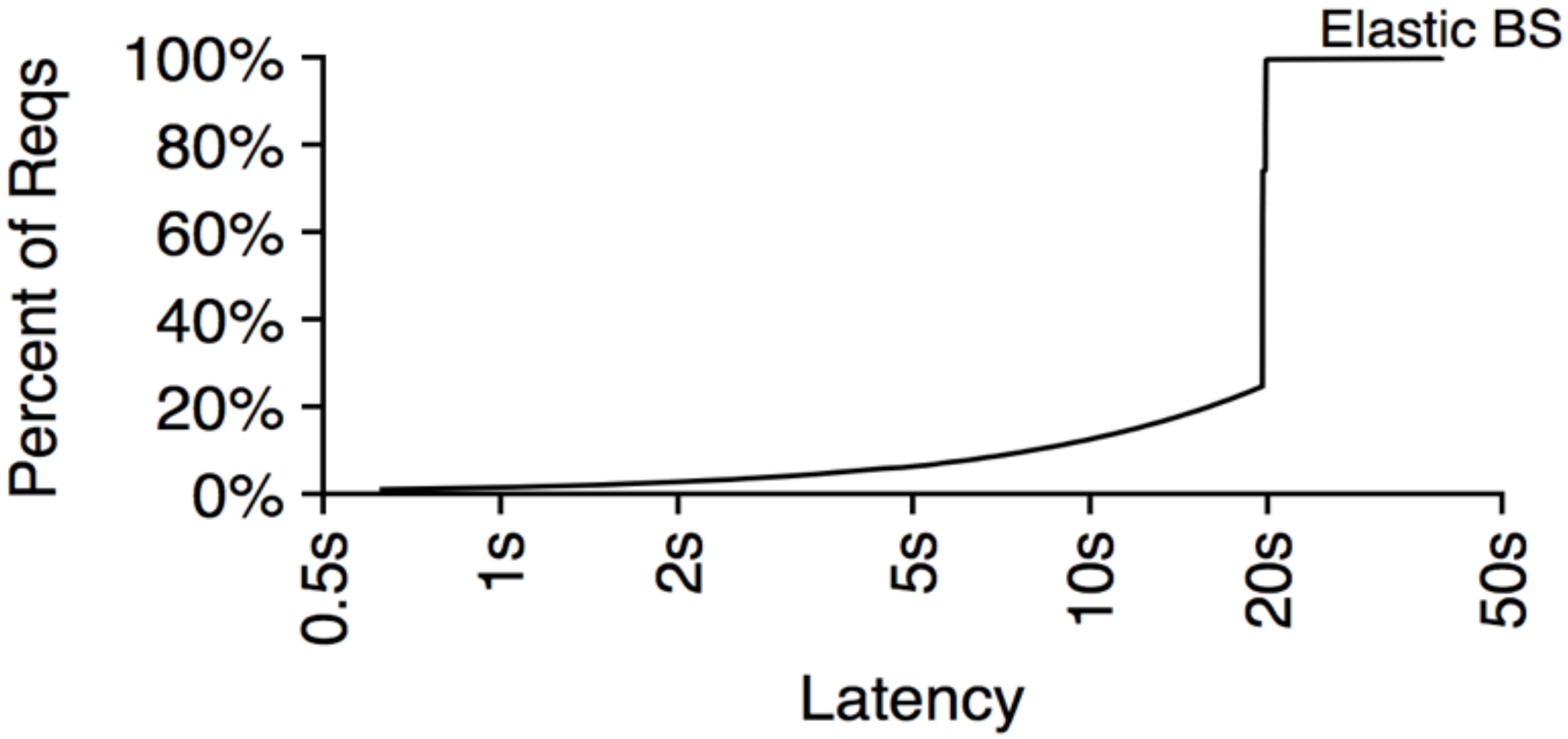
Also container based

More sophisticated autoscaling rules

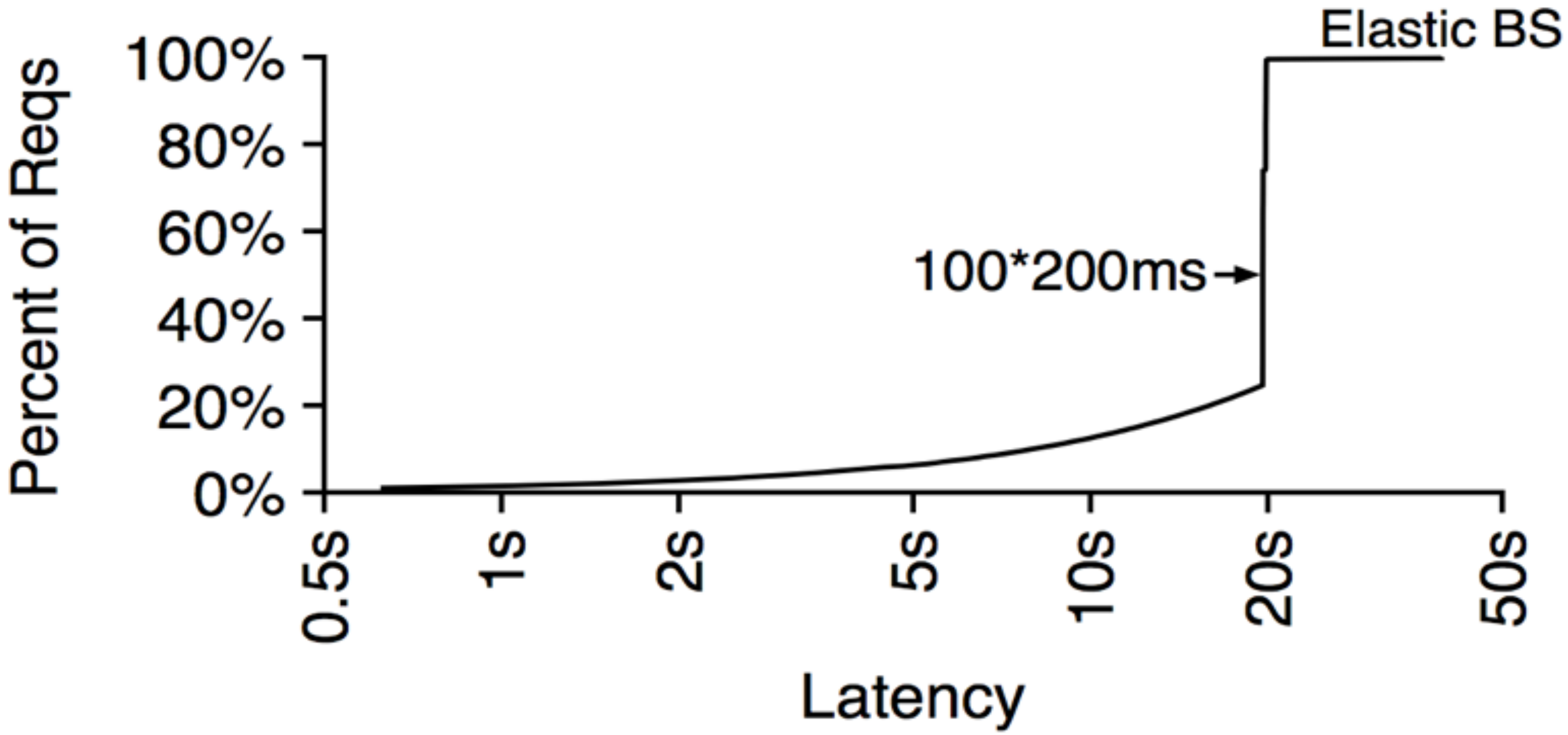
Experiment

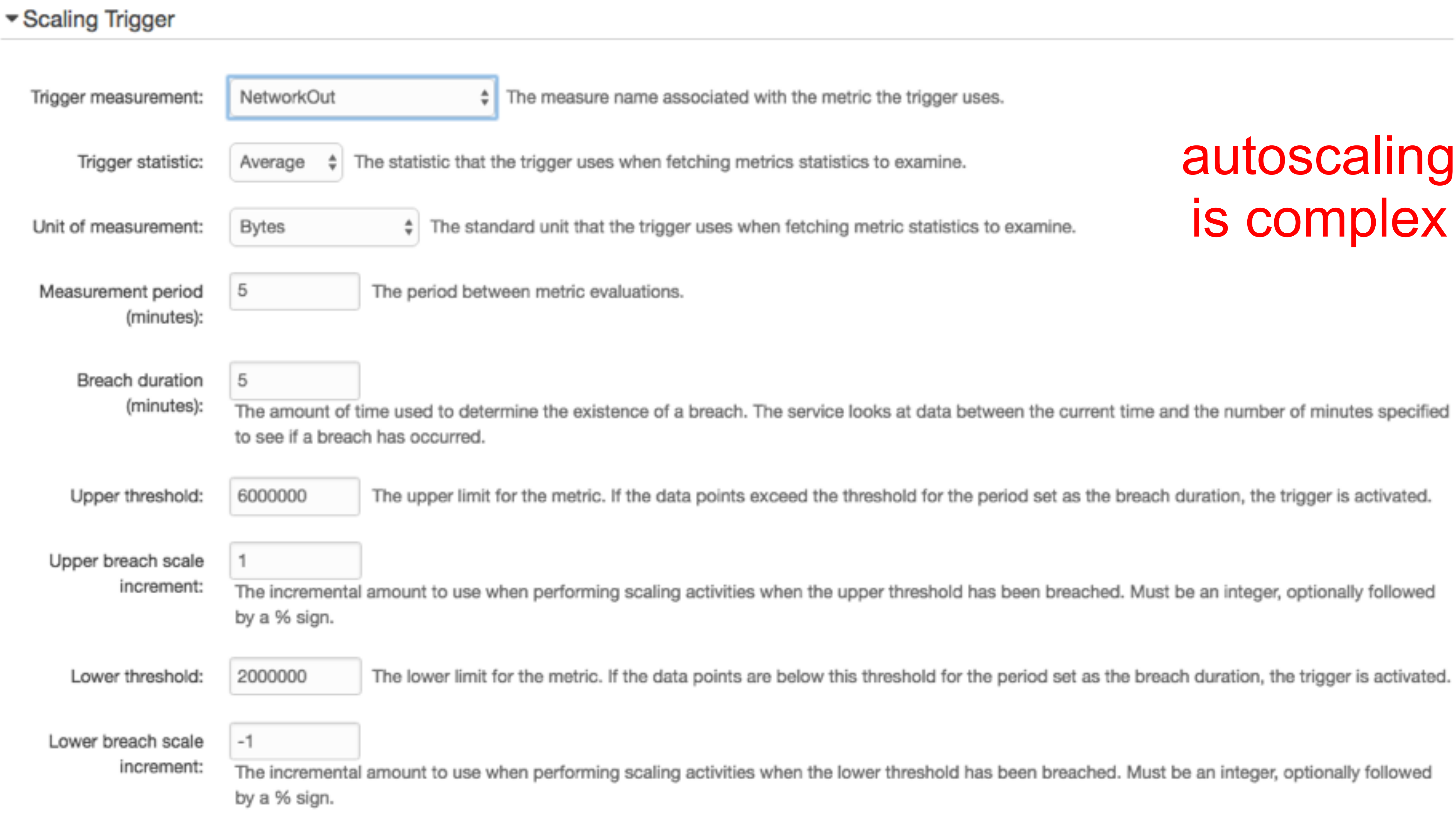
- Maintain **100 concurrent requests**
- Spin **200ms** per request
- Run for **1 minute**

Elastic Beanstalk



Elastic B***s****


Elastic BS
100%
80%
60%
40%
20%
0%
Percent of Reqs
100*200ms
0.5s
1s
2s
5s
10s
20s
50s
Latency

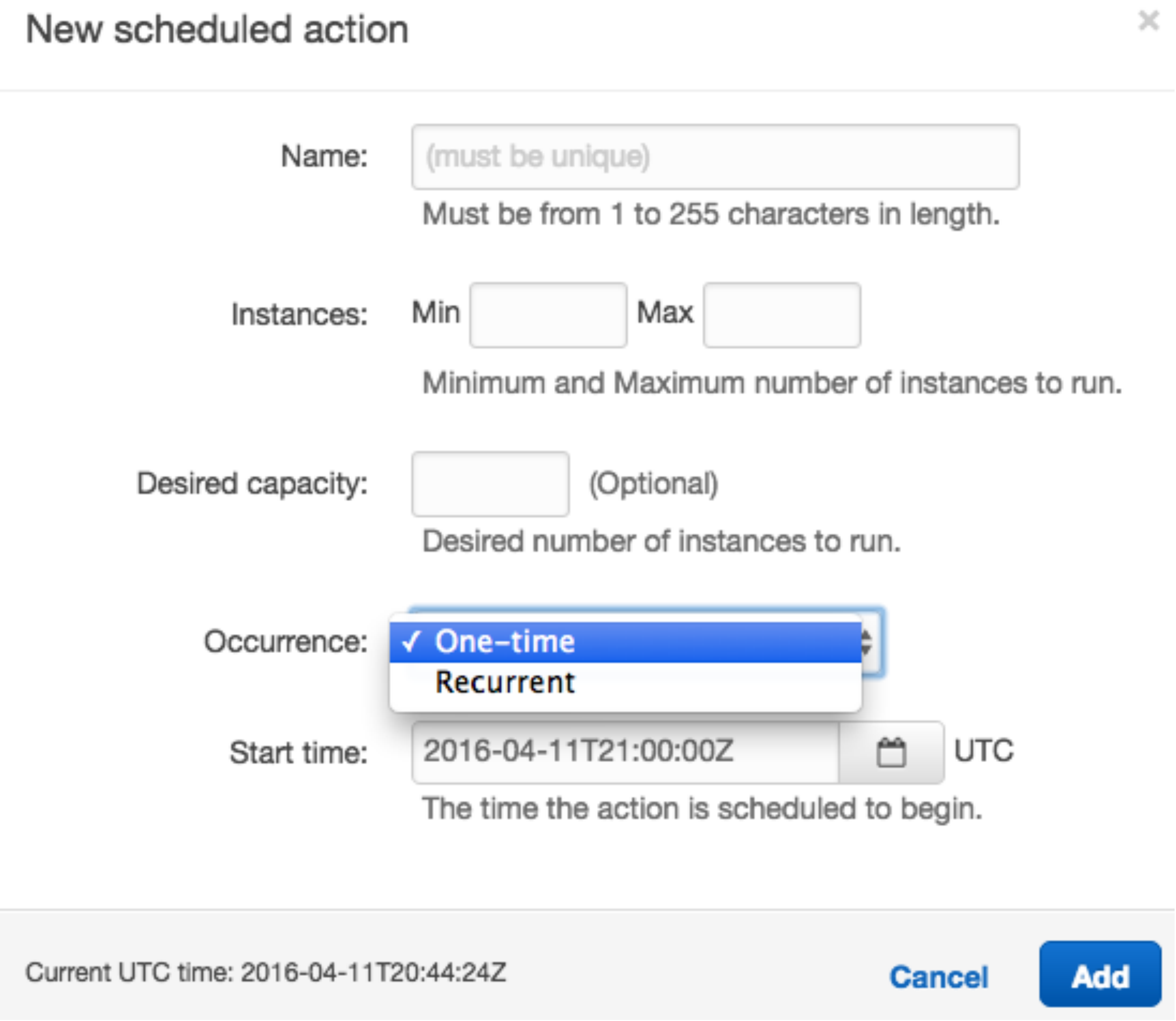


Scaling Trigger

Trigger measurement:	NetworkOut	The measure name associated with the metric the trigger uses.
Trigger statistic:	Average	The statistic that the trigger uses when fetching metrics statistics to examine.
Unit of measurement:	Bytes	The standard unit that the trigger uses when fetching metric statistics to examine.
Measurement period (minutes):	5	The period between metric evaluations.
Breach duration (minutes):	5	The amount of time used to determine the existence of a breach. The service looks at data between the current time and the number of minutes specified to see if a breach has occurred.
Upper threshold:	6000000	The upper limit for the metric. If the data points exceed the threshold for the period set as the breach duration, the trigger is activated.
Upper breach scale increment:	1	The incremental amount to use when performing scaling activities when the upper threshold has been breached. Must be an integer, optionally followed by a % sign.
Lower threshold:	2000000	The lower limit for the metric. If the data points are below this threshold for the period set as the breach duration, the trigger is activated.
Lower breach scale increment:	-1	The incremental amount to use when performing scaling activities when the lower threshold has been breached. Must be an integer, optionally followed by a % sign.

autoscaling is complex

“Autoscaling” is very manual



New scheduled action

Name: (must be unique)
Must be from 1 to 255 characters in length.

Instances: Min Max
Minimum and Maximum number of instances to run.

Desired capacity: (Optional)
Desired number of instances to run.

Occurrence: ✓ One-time
Recurrent

Start time: 2016-04-11T21:00:00Z UTC
The time the action is scheduled to begin.

Current UTC time: 2016-04-11T20:44:24Z

Why should it take minutes (or even seconds) to execute scripts that are 1000s of LOC?

Lambda model

Run user handlers in response to events

- web requests (RPC handlers)
- database updates (triggers)
- scheduled events (cron jobs)

Lambda model

Run user handlers in response to events

- web requests (RPC handlers)
- database updates (triggers)
- scheduled events (cron jobs)

Design principle: share as much as possible!

Lambda model

Run user handlers in response to events

- web requests (RPC handlers)
- database updates (triggers)
- scheduled events (cron jobs)

Design principle: share as much as possible!

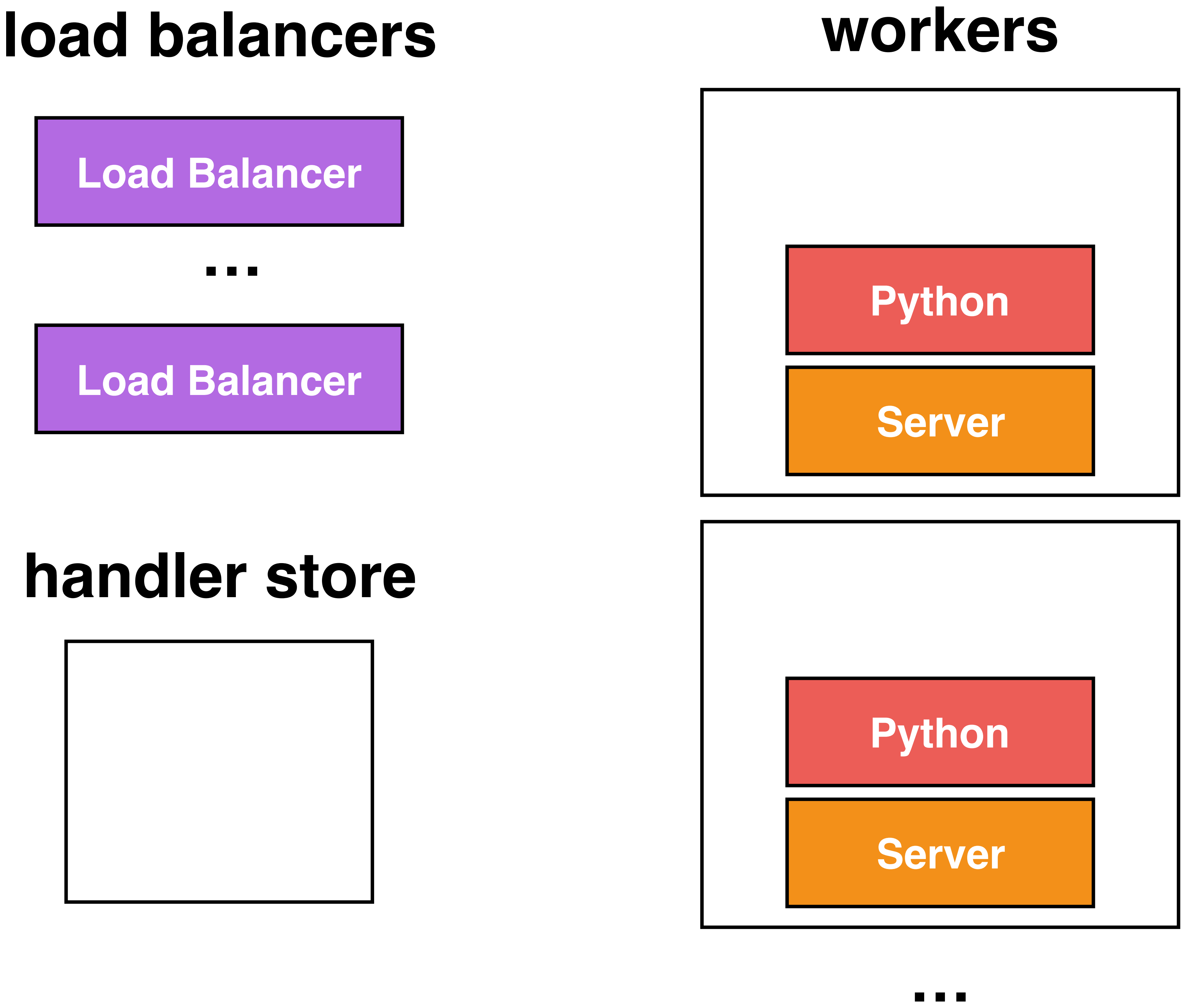
Share server pool between customers

- Any worker can execute any handler
- No spinup time
- Less switching

Encourage specific runtime (C#, Node.JS, Python)

- Minimize network copying
- Code will be in resident in memory

Architecture



load balancers

Load Balancer

...

Load Balancer

handler store

workers

Python

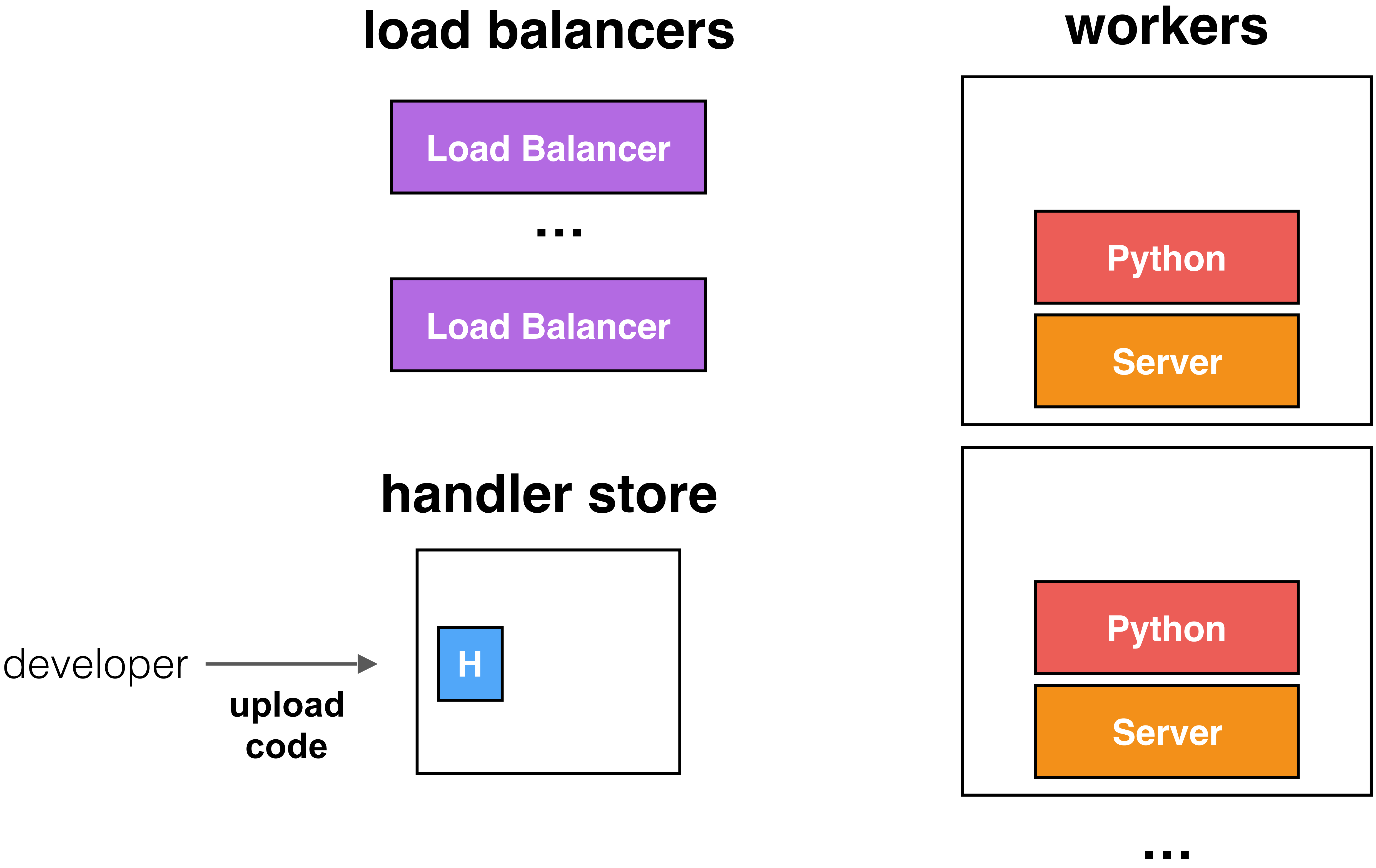
Server

Python

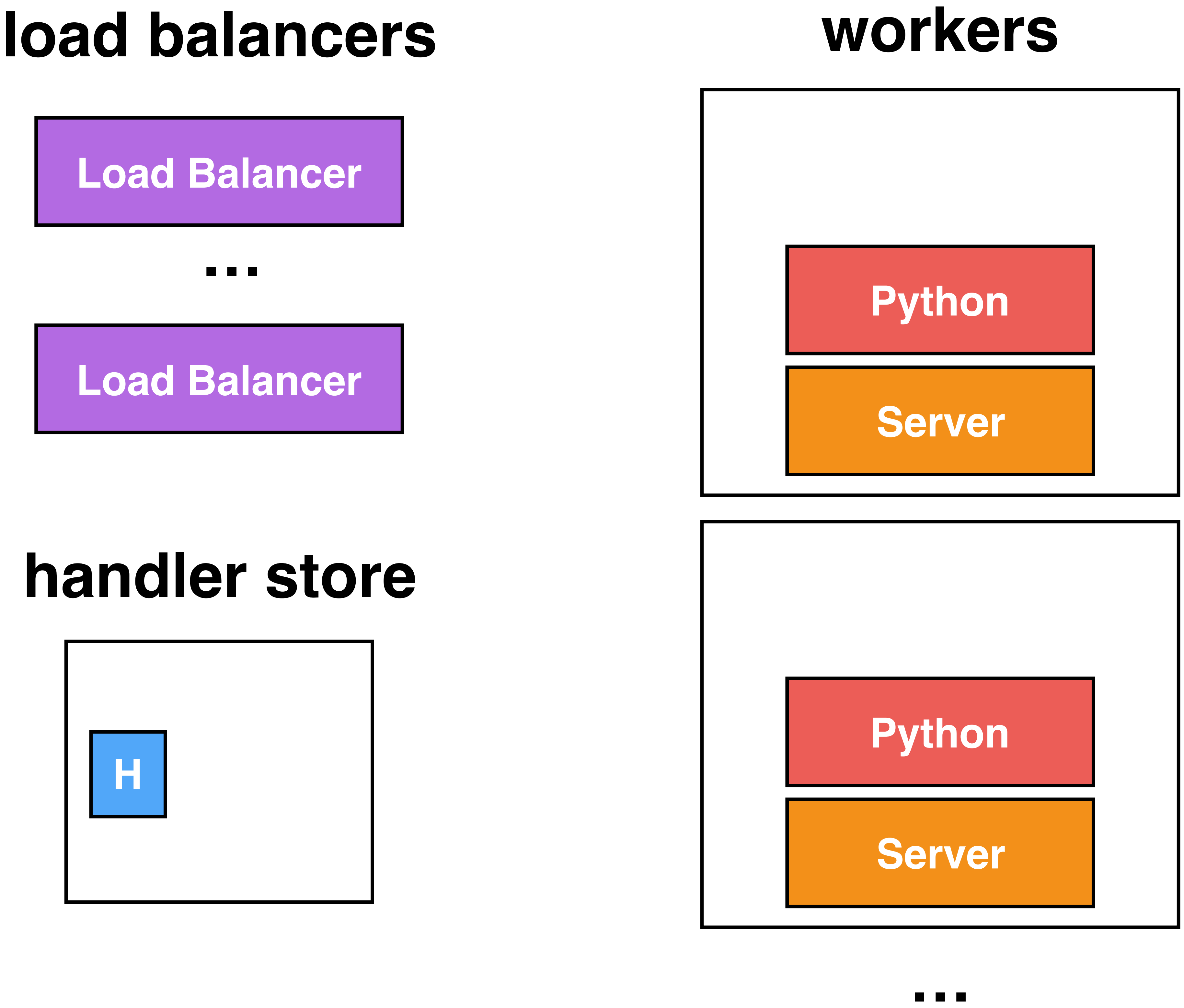
Server

...

Architecture



Architecture



load balancers

Load Balancer

...

Load Balancer

handler store

H

workers

Python

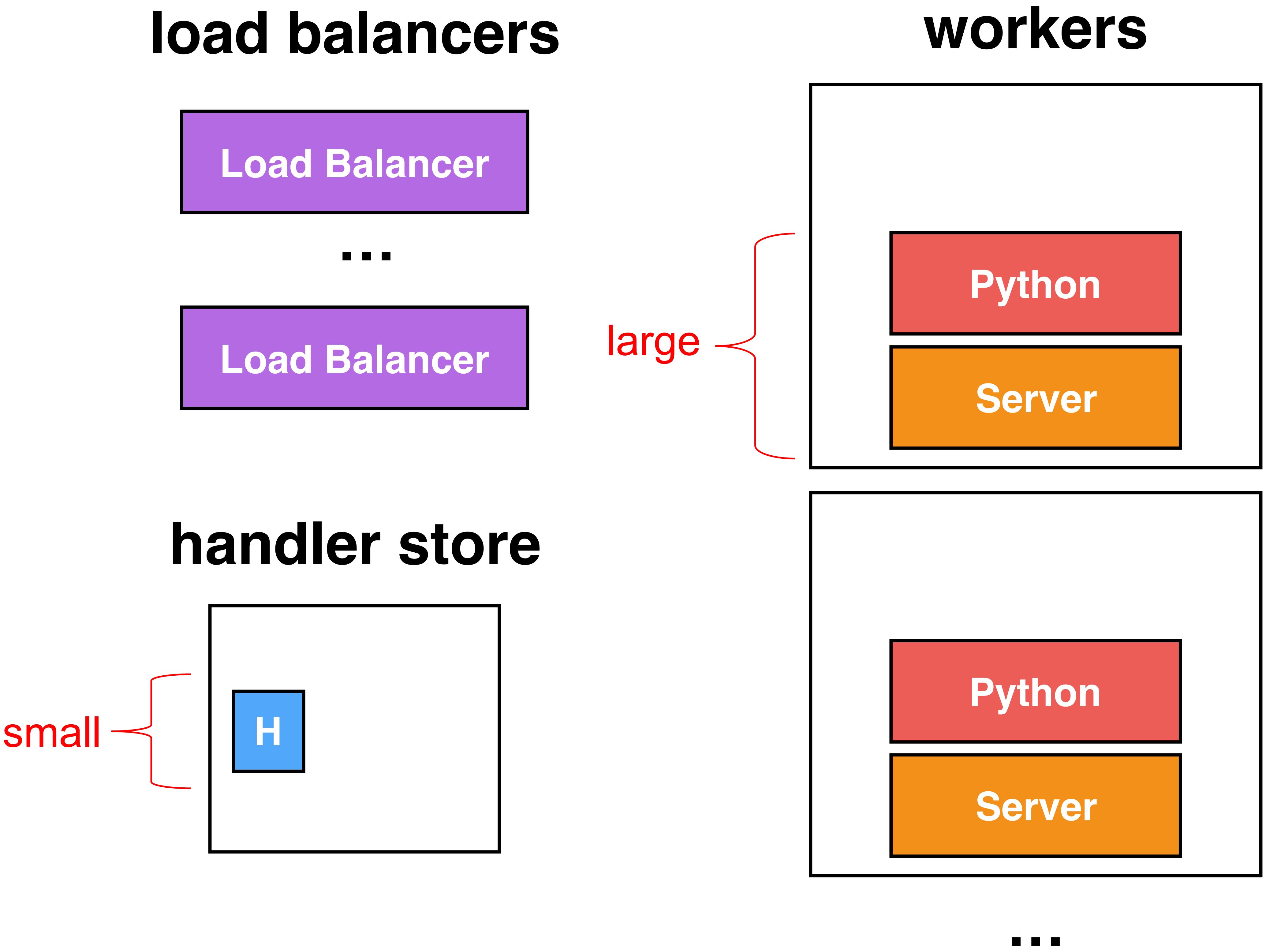
Server

Python

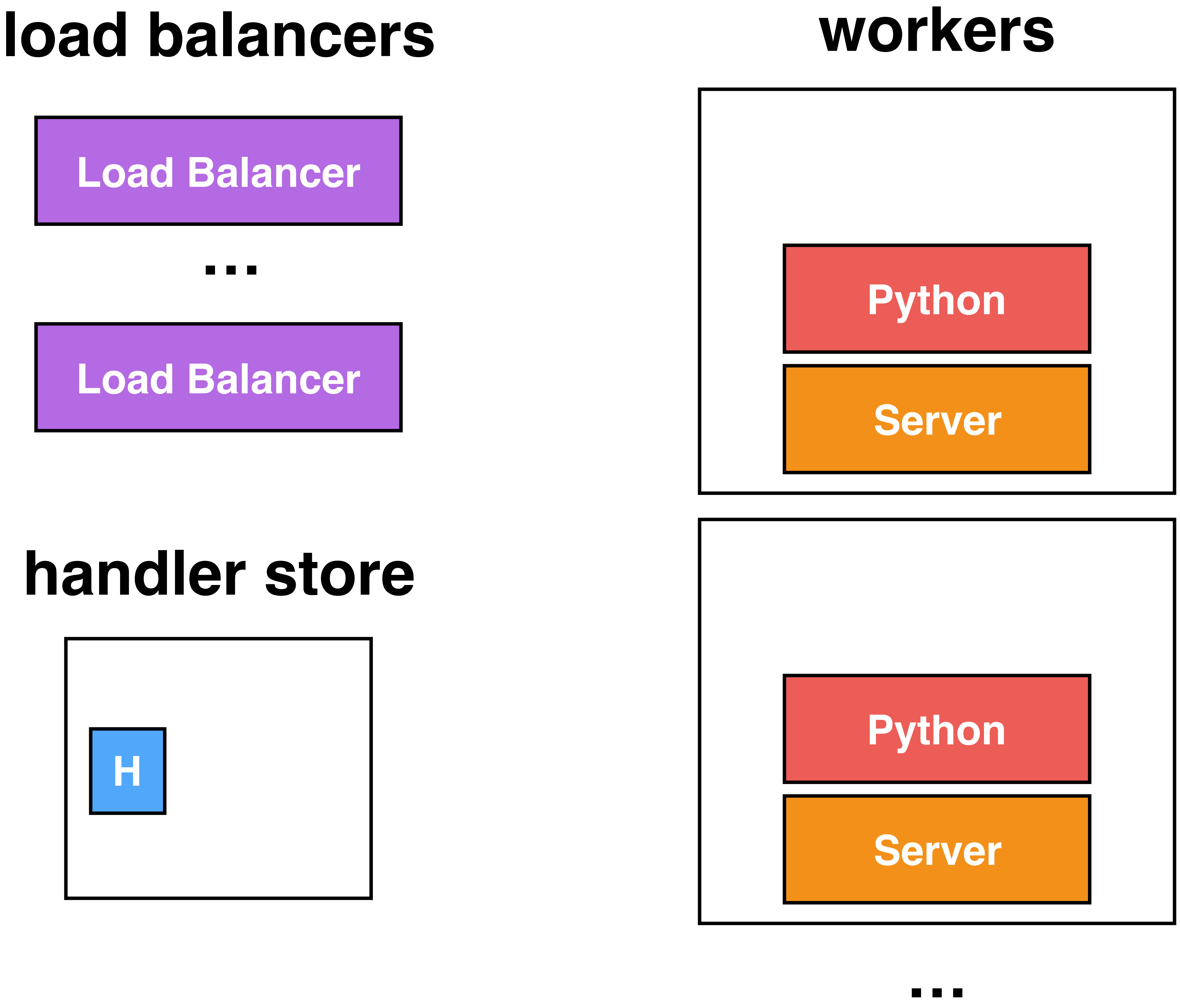
Server

...

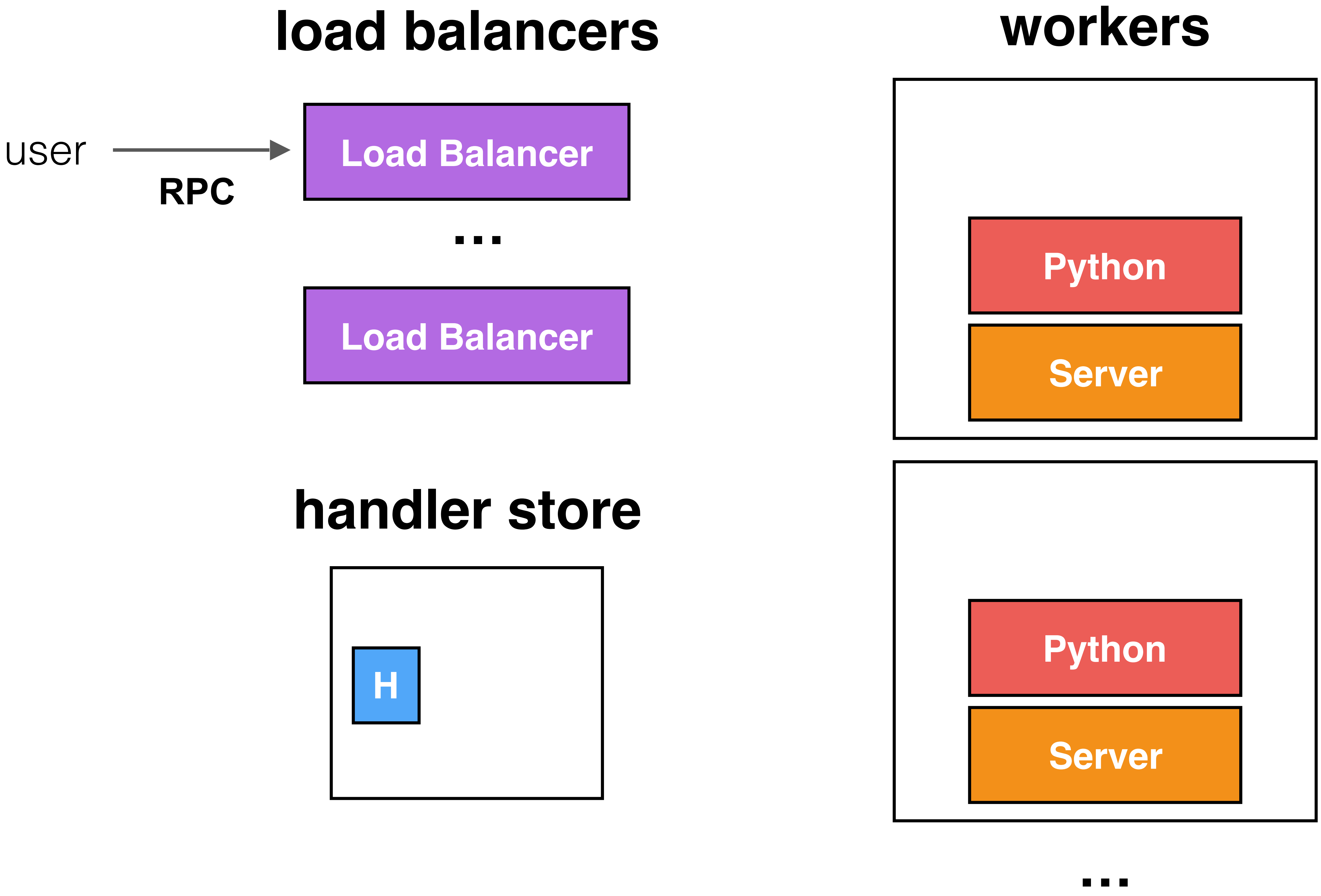
Architecture



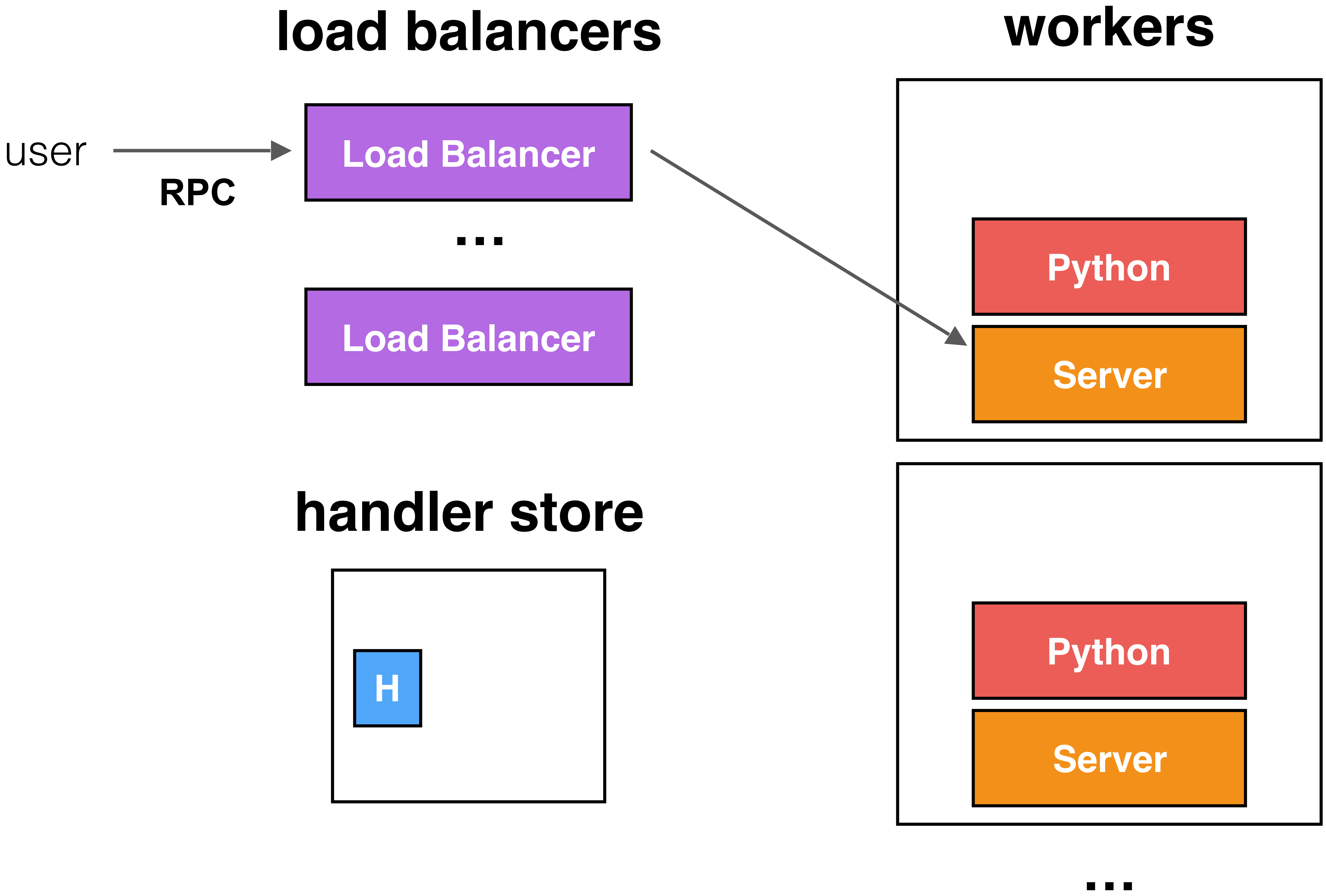
Architecture



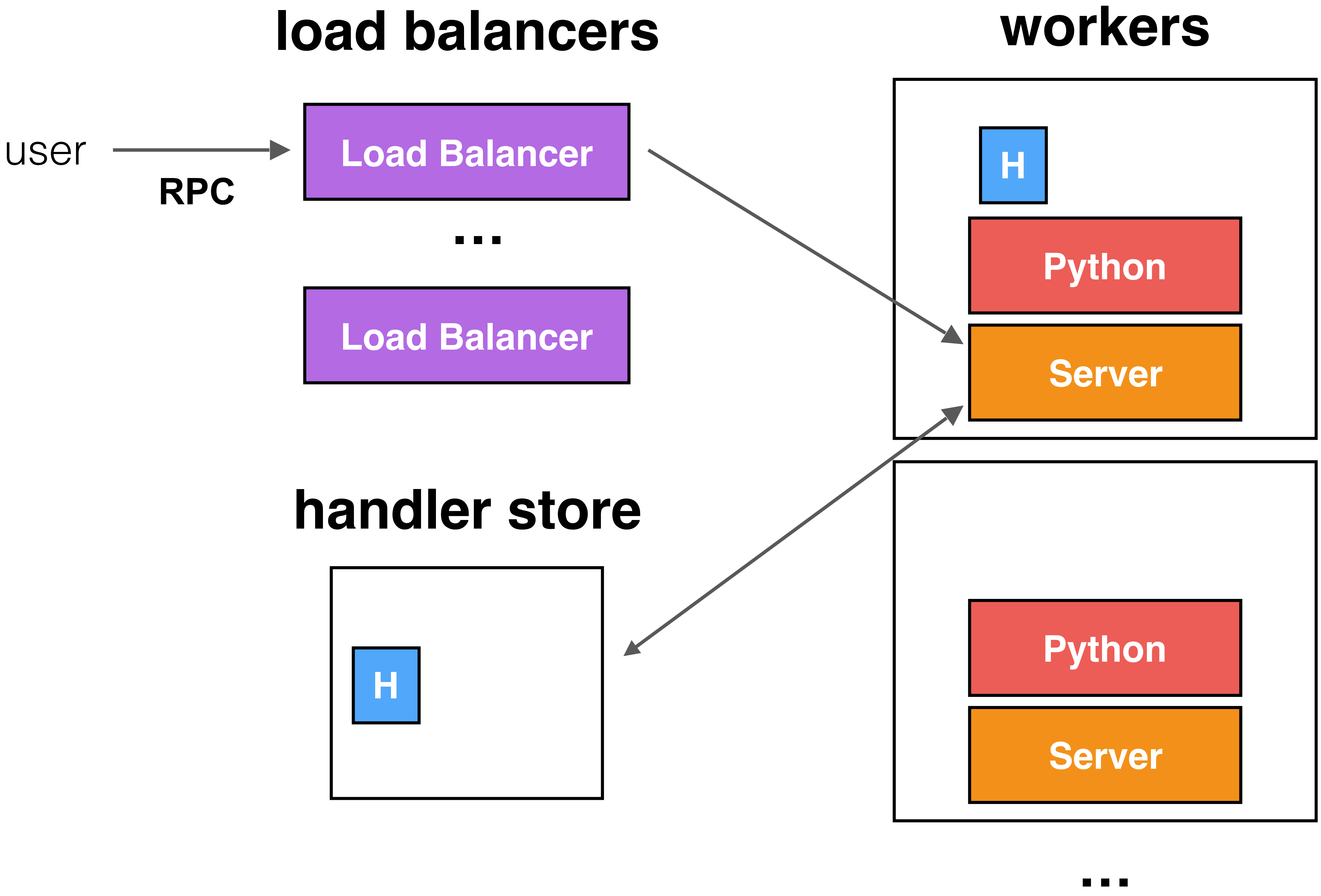
Architecture



Architecture



Architecture



load balancers

user

RPC

Load Balancer

...

Load Balancer

handler store

H

workers

H

Python

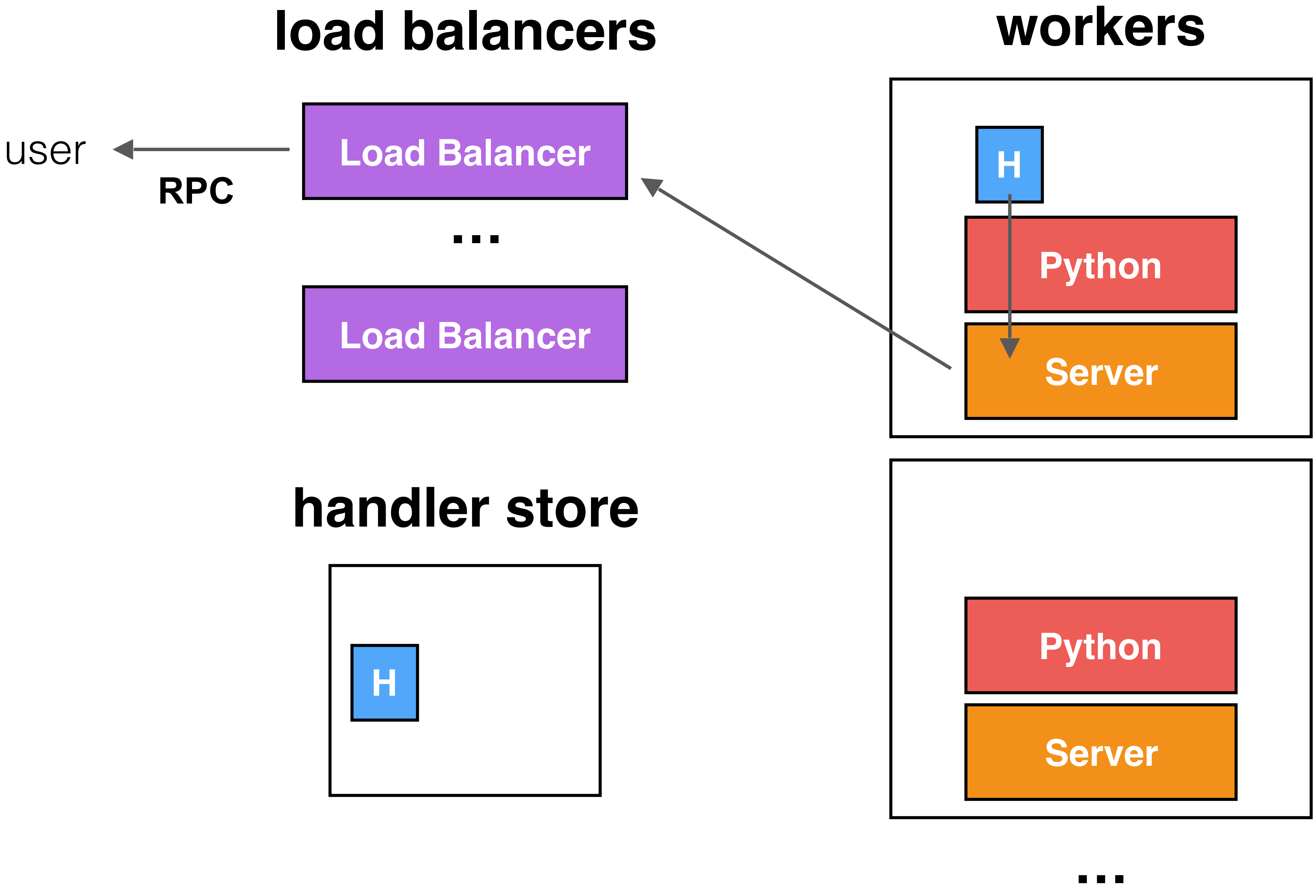
Server

Python

Server

...

Architecture



Lambda elasticity

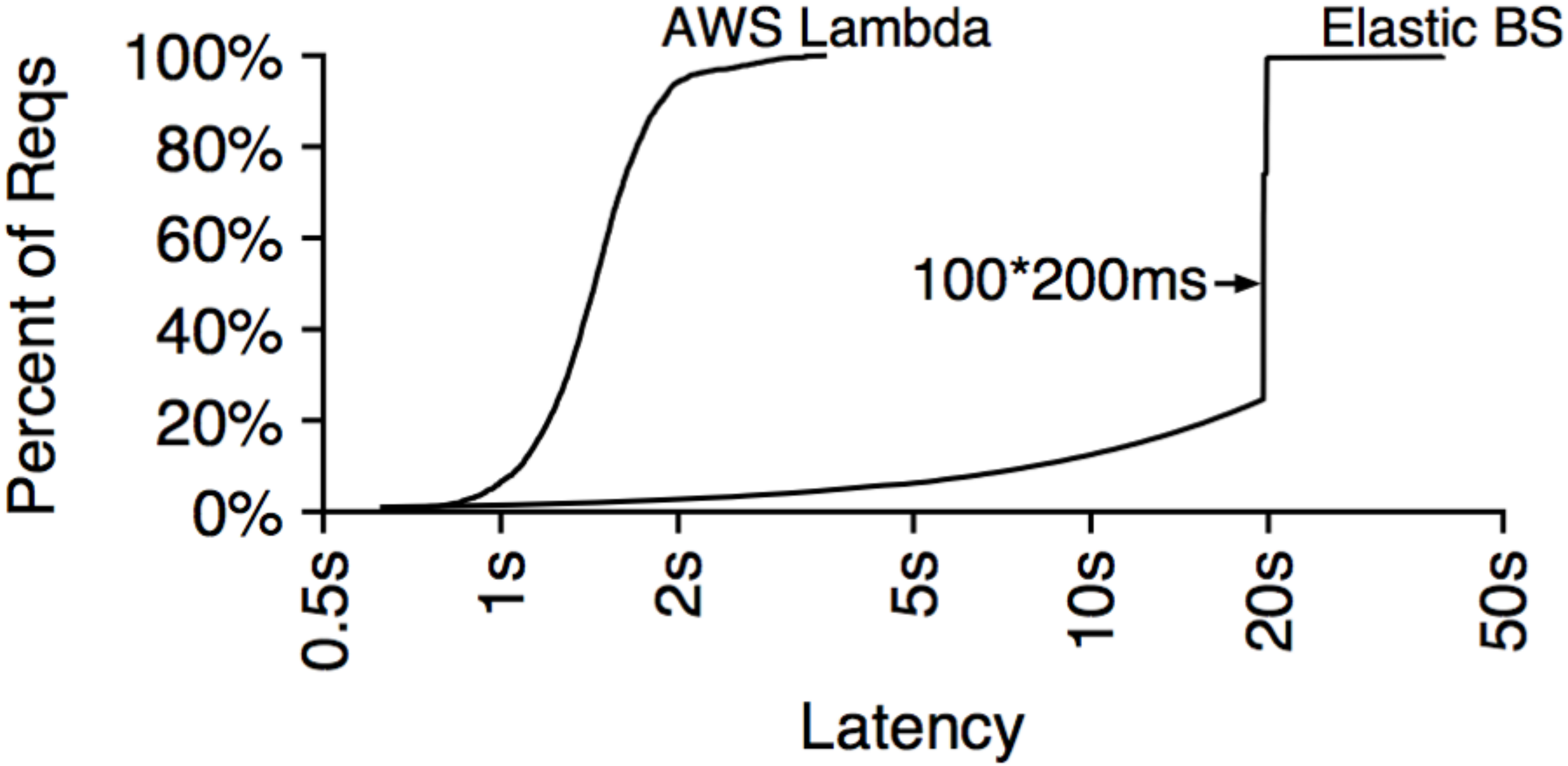
Fast scaling should be easy

- Handlers are small, so copying is cheap
- Servers already running

Repeat ElasticBS experiment

- Maintain **100 concurrent requests**
- Spin **200ms** per request
- Run for **1 minute**

Lambda elasticity



Charging

Pay per function invocation

- actually pay-as-you-go
- no charge for idle time between calls

AWS pricing scheme

- charge `actual_time * memory_cap`
- round up `actual_time` to nearest 100ms

Implementations

Public cloud

- Nov 2014: AWS Lambda
- Feb 2016: Google Cloud Functions (Alpha)
- Mar 2016: Azure Functions (Preview)

OpenLambda

- in progress, to be released June 20th, 2016
- goal: enable academic research on Lambdas

Outline

Emerging compute models

Containers vs. Lambdas

Application building

OpenLambda: code overview

Plan projects: discussion

Plan: everybody builds an application

Benefit 1: understanding

- learn about Lambdas
- identify pain points

Benefit 2: evaluation

- turn applications into benchmark suite
- measure improvement (latency, scalability) every week this summer

Application ideas

- Better chat
- Blog tool (with comments)
- Concert tickets
- Multiplayer game
- Nearby friends
- Calendar (with email reminders)
- Stock alert cron job
- Autocomplete
- Simple search engine
- Document conversion
- OCR service
- …

Features to explore

- Authentication (e.g., FB login)
- Cookies
- WebSockets
- DB triggers
- Different runtimes
- JavaScript event integration
- Lambdas calling other Lambdas
- Platforms (OpenLambda, AWS, Google, Azure)

Tips

- JQuery, AJAX
- curl, Postman
- Chrome tools
- CORS protocol (cross origin)
- others?

JavaScript

Suggestion: learn JQuery, AJAX:

```
data = {...};
$.ajax({
  url: “...”,
  type: "POST",
  data: JSON.stringify(data),
  contentType: "application/json",
  success: function(data) {
     ...
  },
  error: function(xhr, ajaxOptions, thrownError) {
     ...
  }
});
```

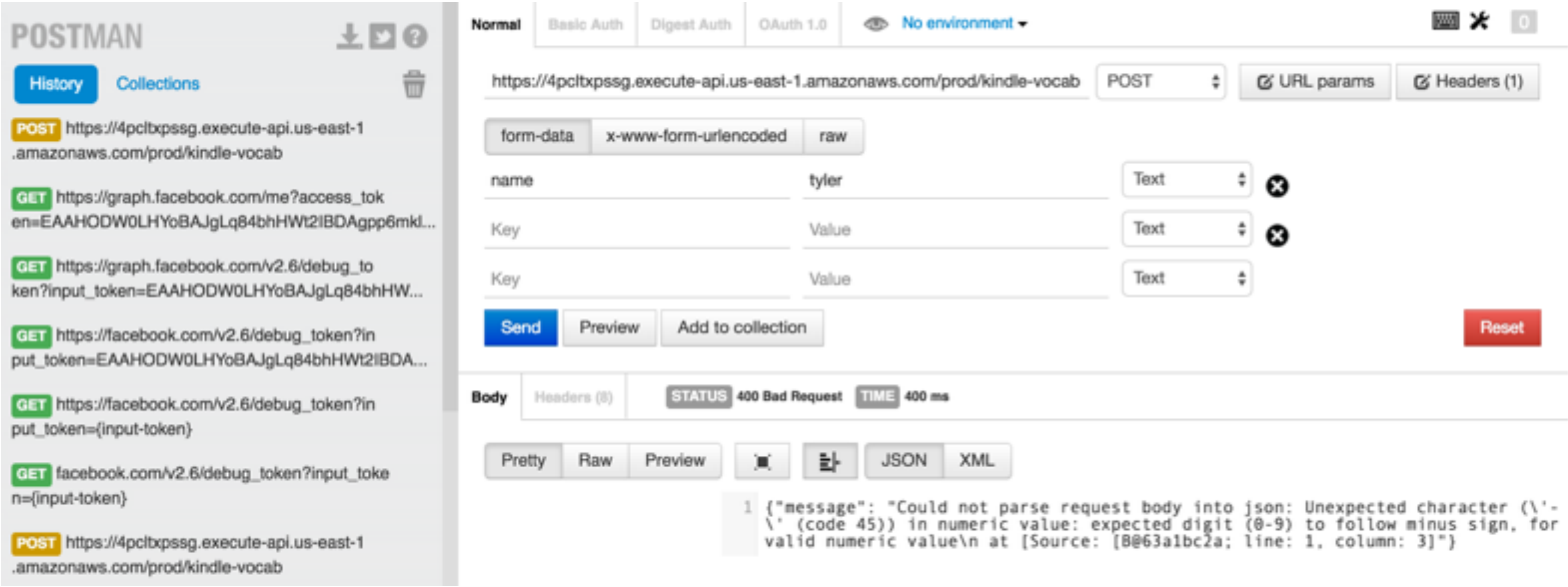
POSTing with curl

Issue command from terminal

```
curl -X POST 172.17.0.15:8080/runLambda/mylambda -d '{}'
```

POSTing with Postman

Chrome extension



POST https://4pcltxpssg.execute-api.us-east-1.amazonaws.com/prod/kindle-vocab

GET https://graph.facebook.com/me?access_token=EAAHODW0LHYoBAJgLq84bhHWt2IBDAgpp6mkl...

GET https://graph.facebook.com/v2.6/debug_token?input_token=EAAHODW0LHYoBAJgLq84bhHW...

GET https://facebook.com/v2.6/debug_token?input_token=EAAHODW0LHYoBAJgLq84bhHWt2IBDA...

GET https://facebook.com/v2.6/debug_token?input_token={input-token}

GET facebook.com/v2.6/debug_token?input_token={input-token}

POST https://4pcltxpssg.execute-api.us-east-1.amazonaws.com/prod/kindle-vocab

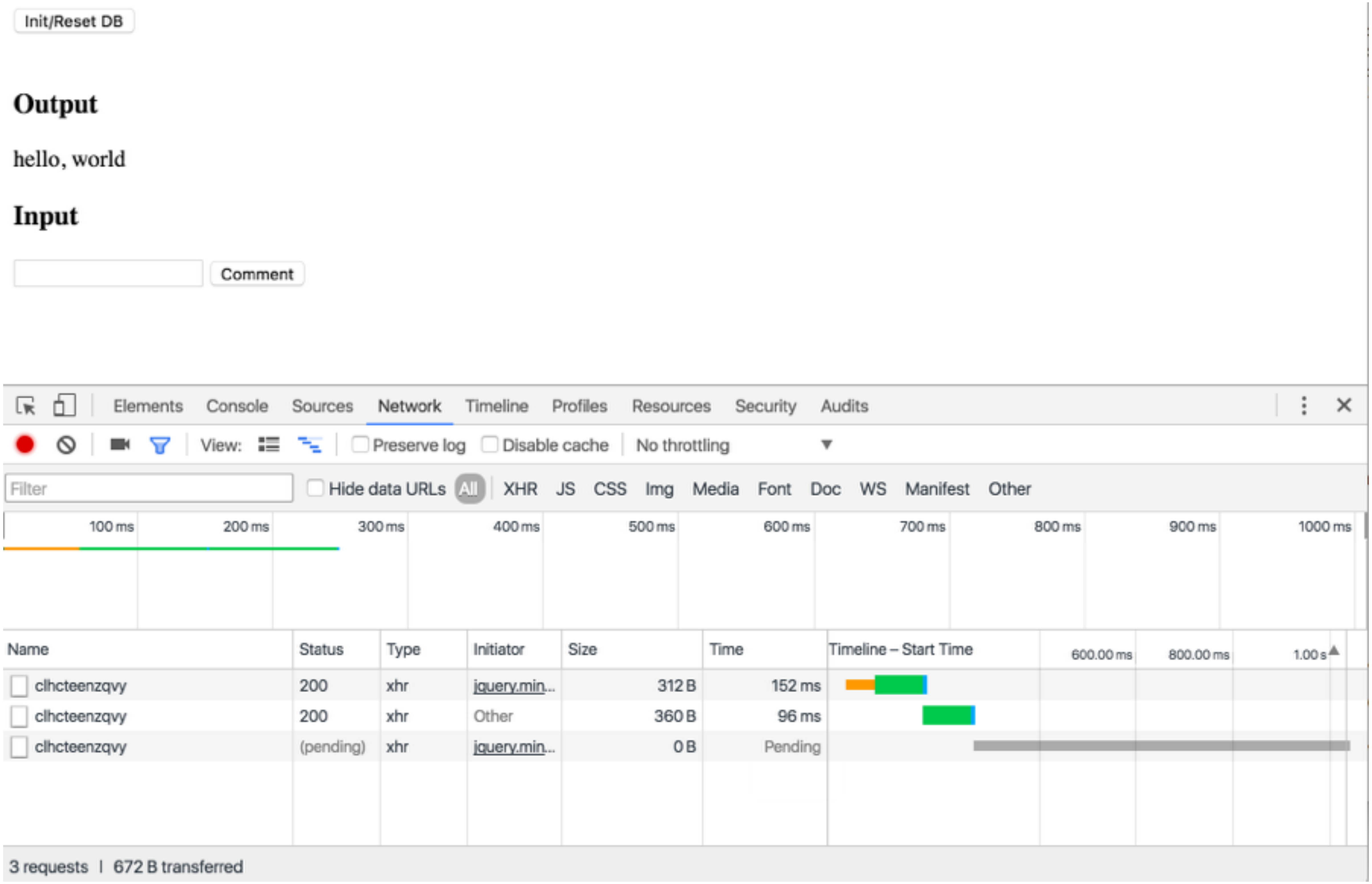
https://4pcltxpssg.execute-api.us-east-1.amazonaws.com/prod/kindle-vocab POST

name	tyler	Text
Key	Value	Text
Key	Value	Text

STATUS 400 Bad Request TIME 400 ms

```
{"message": "Could not parse request body into json: Unexpected character (\'-
\' (code 45)) in numeric value: expected digit (0-9) to follow minus sign, for
valid numeric value\n at [Source: [B@63a1bc2a; line: 1, column: 3]"}
```

Chrome



Output

hello, world

Input

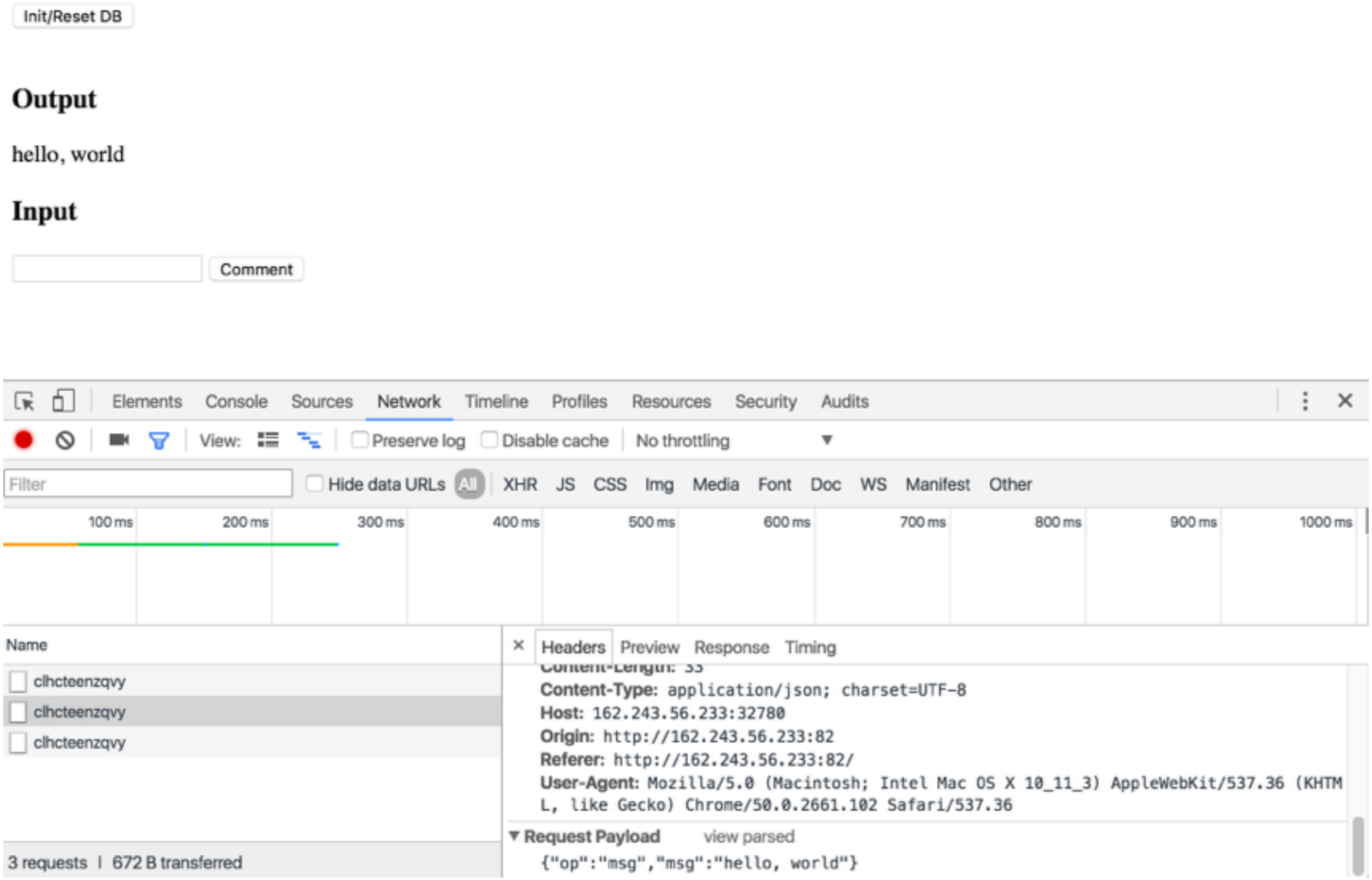
□ Preserve log □ Disable cache No throttling ▼

□ Hide data URLs

Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
□ clhcteenzqvy	200	xhr	jquery.min...	312 B	152 ms	
□ clhcteenzqvy	200	xhr	Other	360 B	96 ms	
□ clhcteenzqvy	(pending)	xhr	jquery.min...	0 B	Pending	

3 requests | 672 B transferred

Chrome


Init/Reset DB
Comment
Elements Console Sources Network Timeline Profiles Resources Security Audits
View:
□ Preserve log
□ Disable cache
No throttling
Filter
□ Hide data URLs
All XHR JS CSS Img Media Font Doc WS Manifest Other
100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms 900 ms 1000 ms
Name
Headers Preview Response Timing
view parsed

Output

hello, world

Input

- clhcteenzqvy
- clhcteenzqvy
- clhcteenzqvy

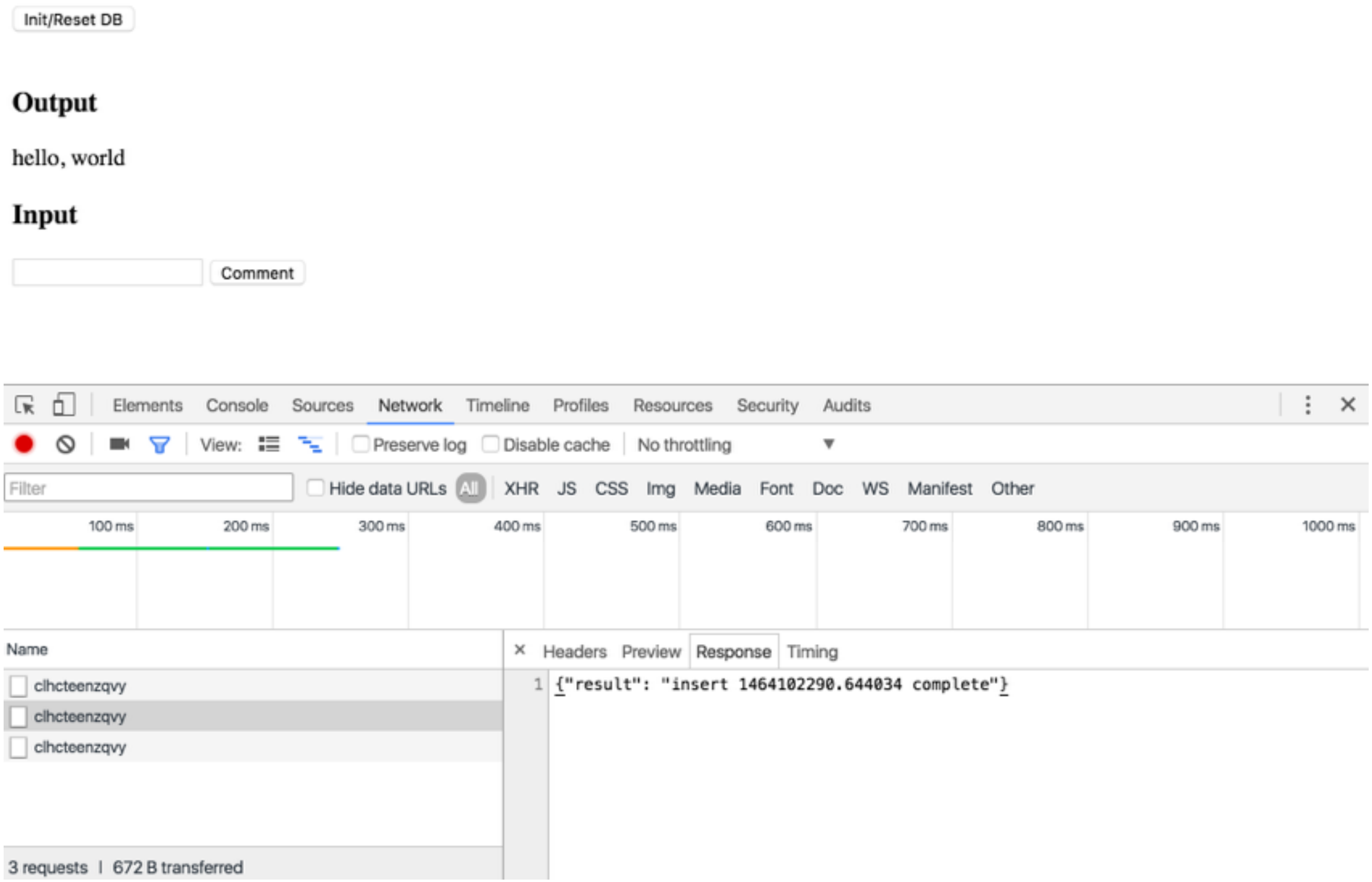
3 requests | 672 B transferred

```
Content-Length: 33
Content-Type: application/json; charset=UTF-8
Host: 162.243.56.233:32780
Origin: http://162.243.56.233:82
Referer: http://162.243.56.233:82/
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTM
L, like Gecko) Chrome/50.0.2661.102 Safari/537.36
```

▼ Request Payload

```
{"op":"msg","msg":"hello, world"}
```

Chrome



Output

hello, world

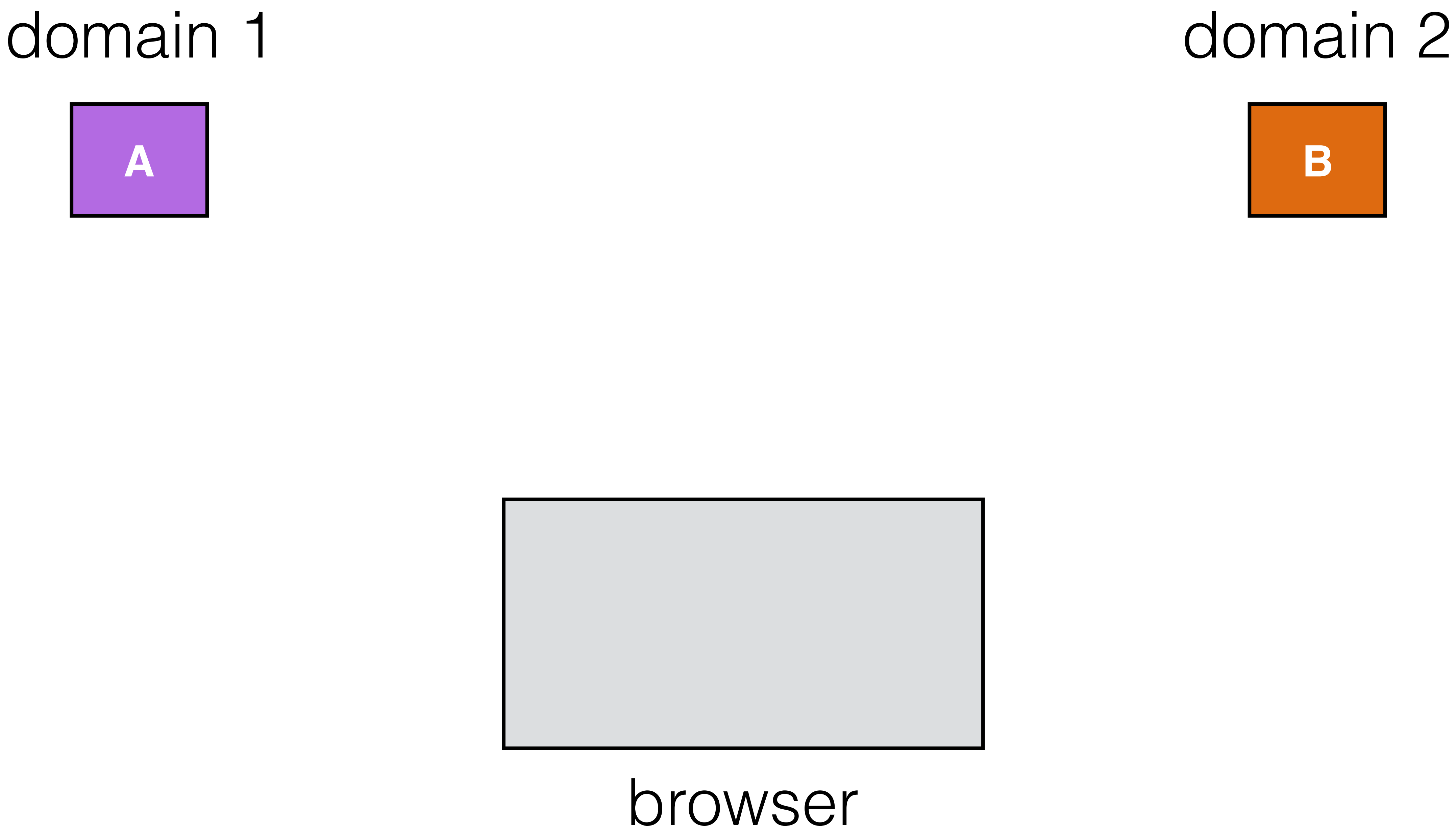
Input

□ clhcteenzqvy
□ clhcteenzqvy
□ clhcteenzqvy

```
{"result": "insert 1464102290.644034 complete"}
```

3 requests | 672 B transferred

CORS: cross-origin HTTP request



domain 1

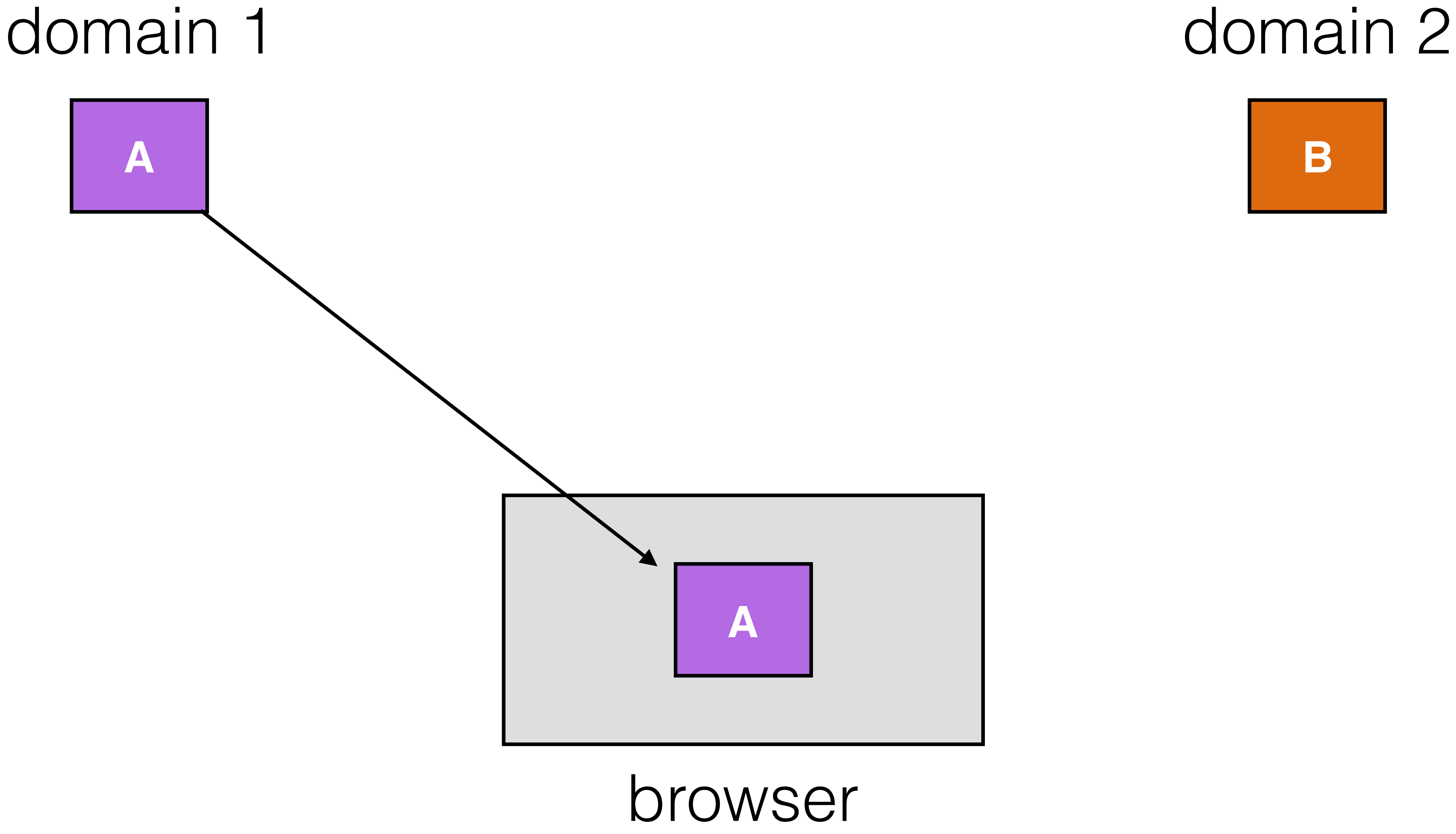
A

domain 2

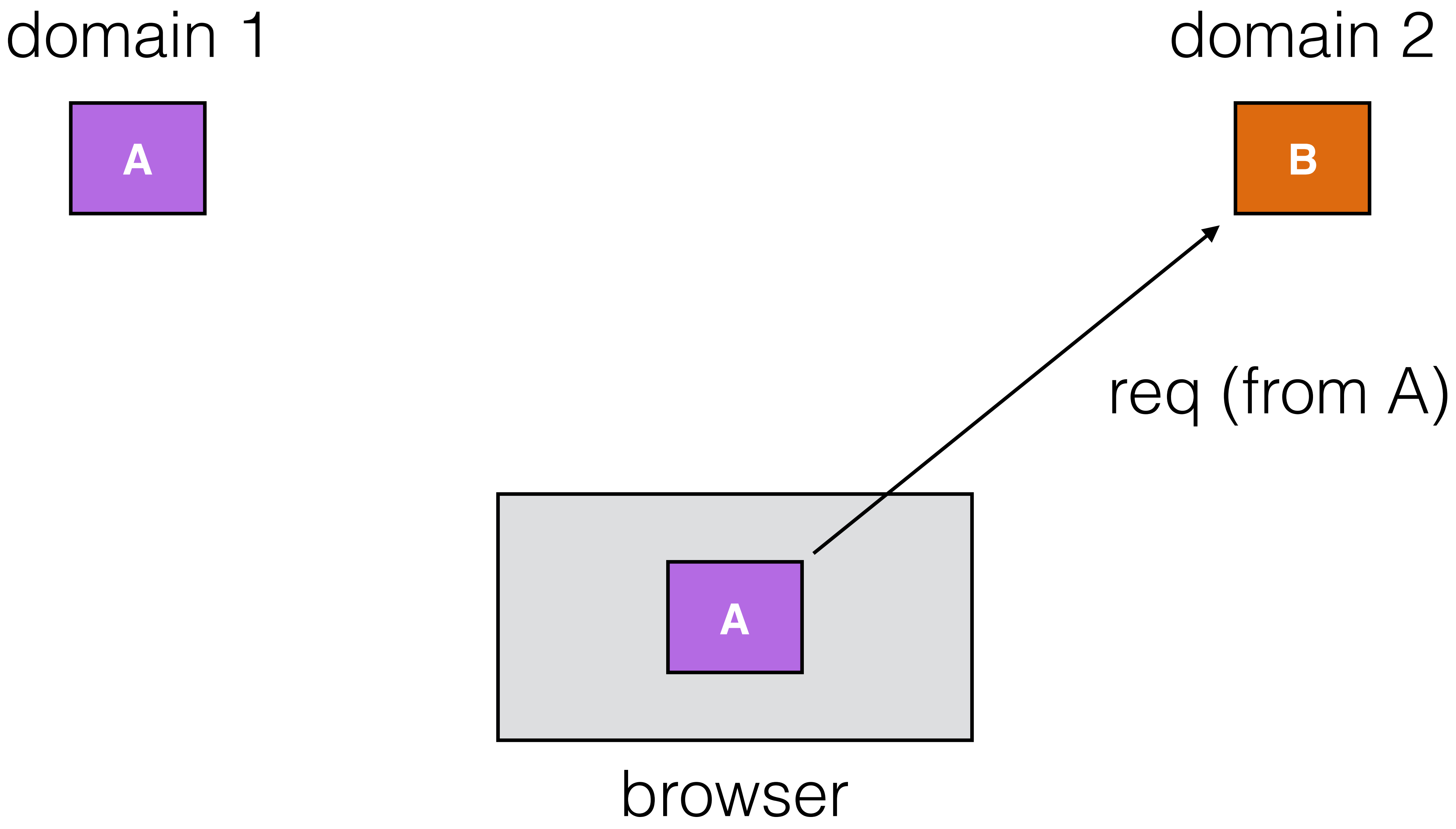
B

browser

CORS: cross-origin HTTP request

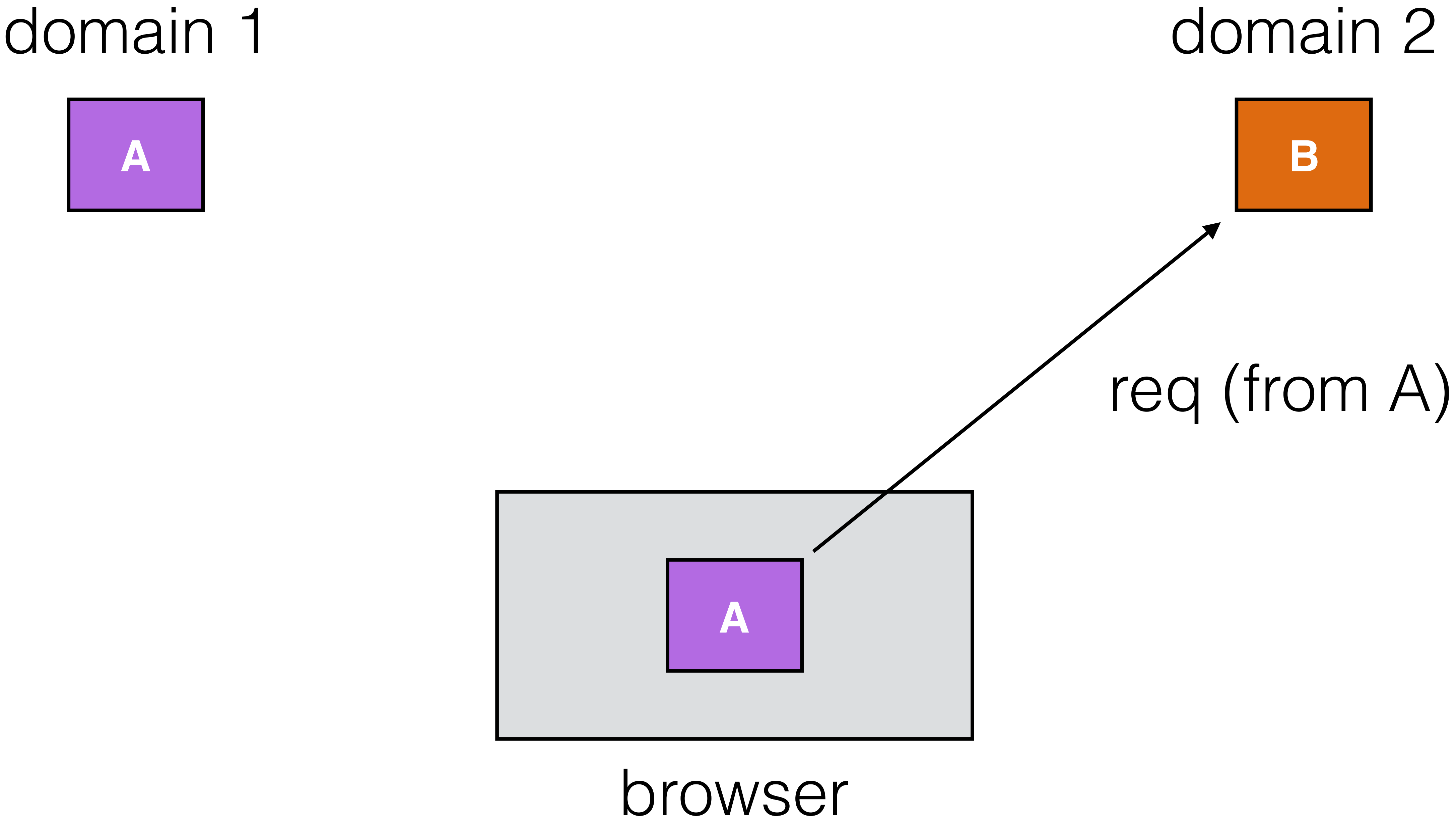


CORS: cross-origin HTTP request



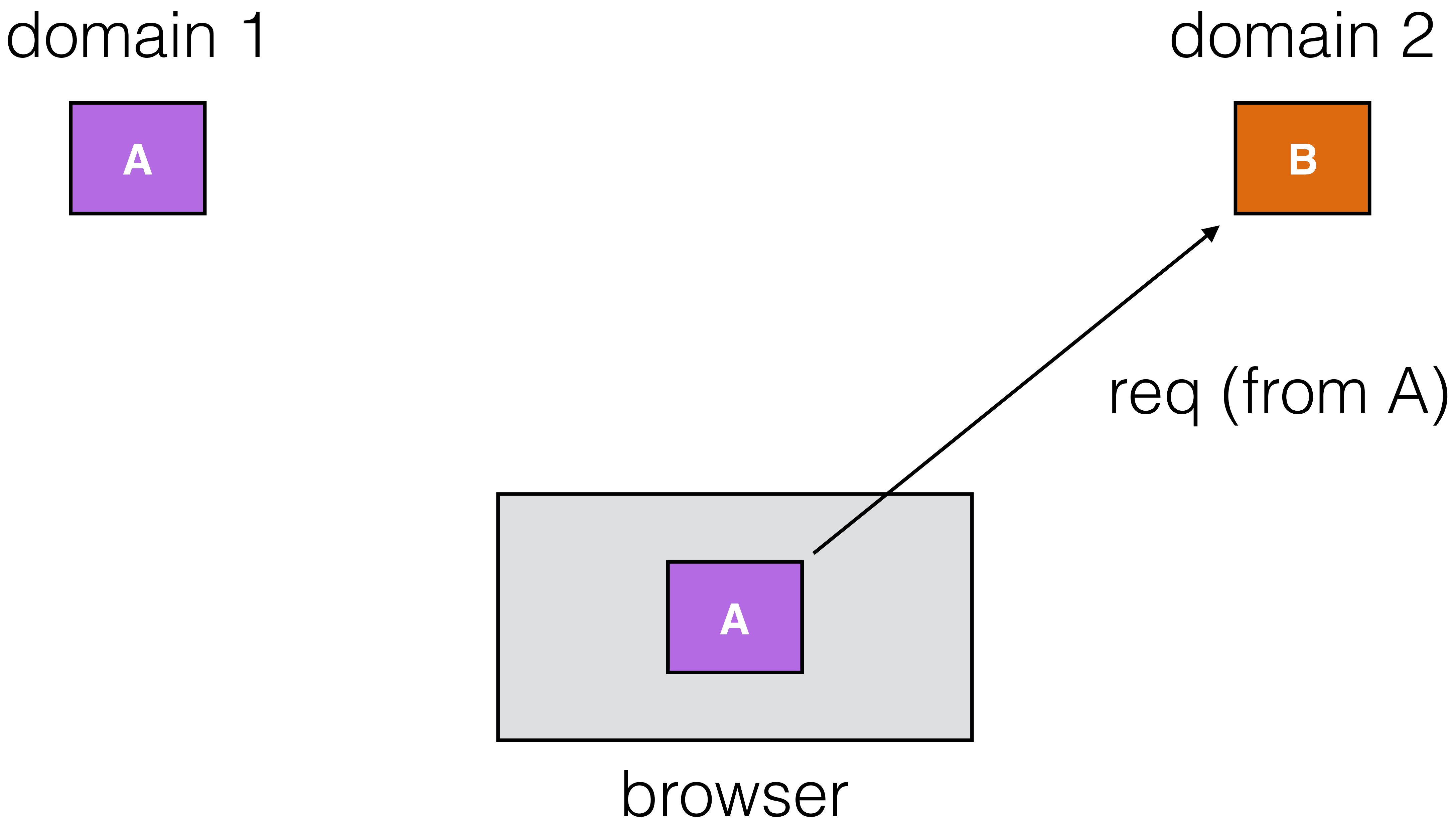
browser: is it OK for A content to request B content?

CORS: cross-origin HTTP request



browser: A must think so

CORS: cross-origin HTTP request



B must tell browser what domains are OK

CORS: cross-origin HTTP request

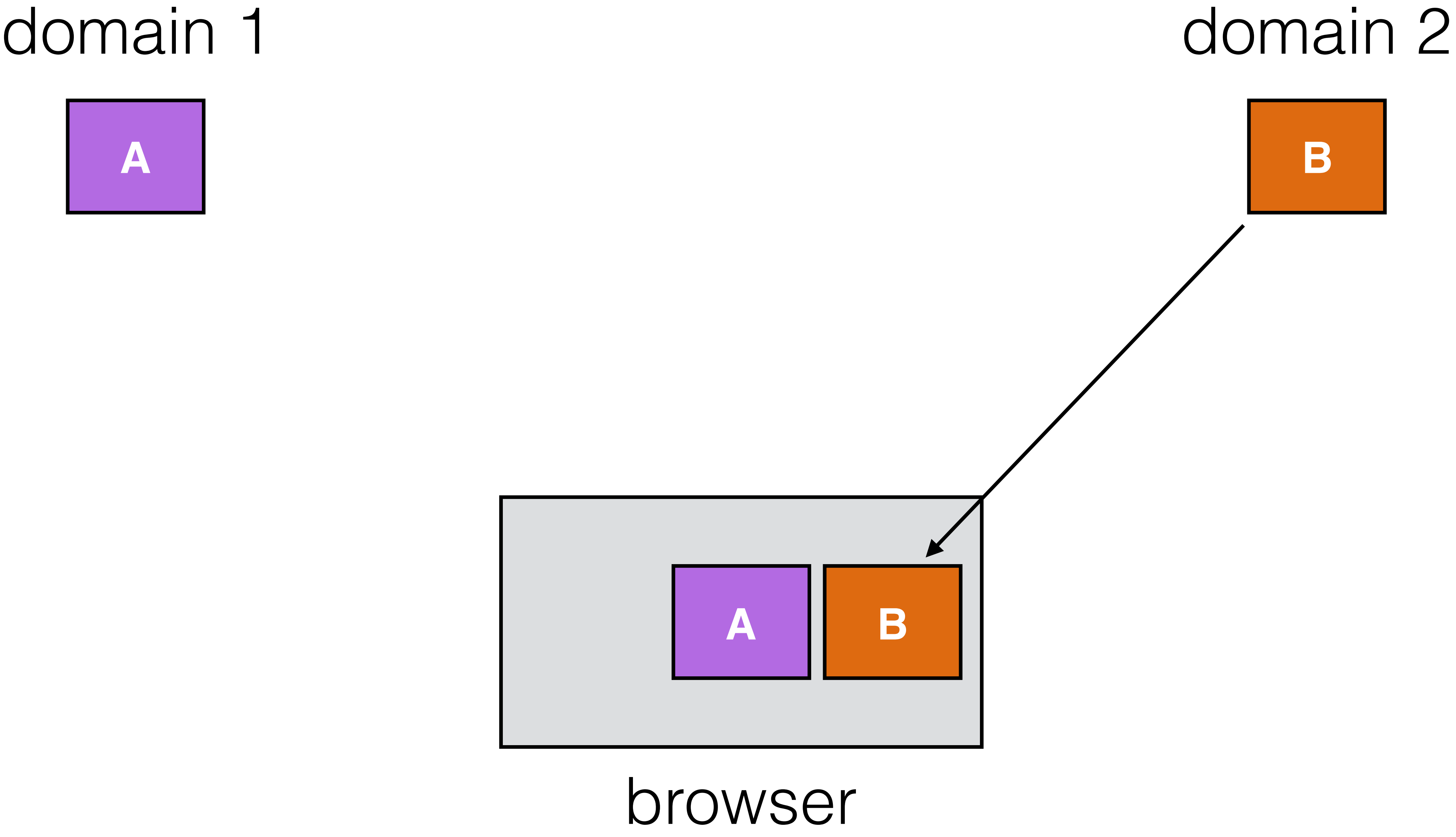
request

```
POST /runLambda/clhcteenzqvy HTTP/1.1
Host: 162.243.56.233:32780
Connection: keep-alive
Content-Length: 39
Accept: application/json, text/javascript, */*; q=0.01
Origin: http://162.243.56.233:82
User-Agent: Mozilla/5.0
Content-Type: application/json; charset=UTF-8
Referer: http://162.243.56.233:82/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
```

response

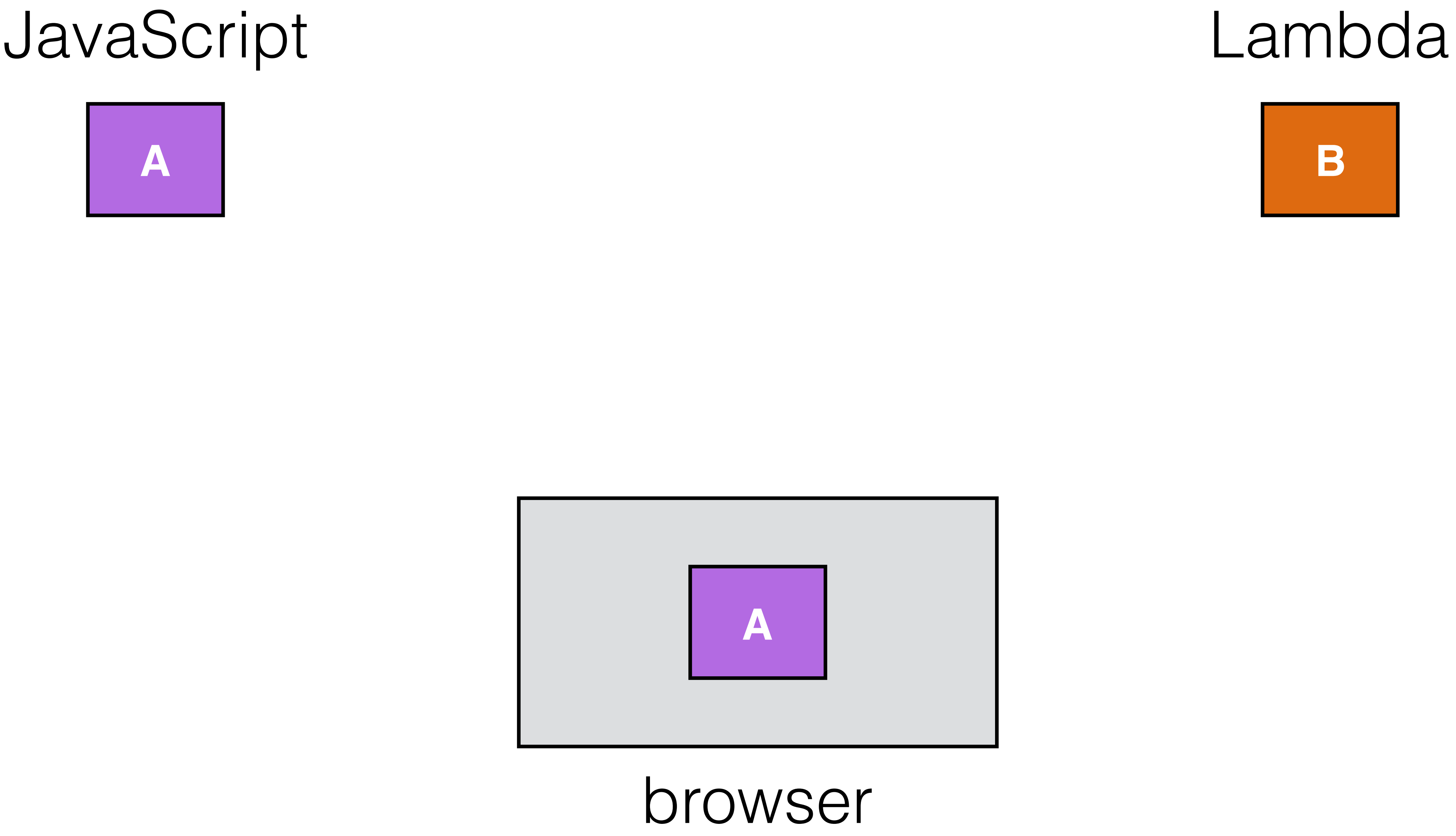
```
HTTP/1.1 200 OK
Access-Control-Allow-Headers: Content-Type, Content-Range, Content-Description
Access-Control-Allow-Methods: GET, PUT, POST, DELETE, OPTIONS
Access-Control-Allow-Origin: *
Date: Tue, 24 May 2016 17:39:30 GMT
Content-Length: 98
Content-Type: text/plain; charset=utf-8
```

CORS: cross-origin HTTP request



browser: B says it’s OK

CORS: cross-origin HTTP request



JavaScript

A

Lambda

B

A

browser

Outline

Emerging compute models

Containers vs. Lambdas

Application building

OpenLambda: code overview

Plan projects: discussion

Source code

https://github.com/tylerharter/open-lambda

- **worker**: Lambda server that executes handlers
- **nginx**: load balancer
- **lambda-generator**: old script for generating Python Lambdas
- **node**: container with worker, rethinkdb, and docker
- **util**: scripts for starting/stopping local cluster
- **applications**: OpenLambda applications
- **testing**: initial unit test environment

Source code

https://github.com/tylerharter/open-lambda

- **worker**: Lambda server that executes handlers
- **nginx**: load balancer
- **lambda-generator**: old script for generating Python Lambdas
- **node**: container with worker, rethinkdb, and docker
- **util**: scripts for starting/stopping local cluster
- **applications**: OpenLambda applications
- **testing**: initial unit test environment

Details

- golang
- receives web requests
- starts Lambda handlers inside docker containers

Source code

https://github.com/tylerharter/open-lambda

- **worker**: Lambda server that executes handlers
- **nginx**: load balancer
- **lambda-generator**: old script for generating Python Lambdas
- **node**: container with worker, rethinkdb, and docker
- **util**: scripts for starting/stopping local cluster
- **applications**: OpenLambda applications
- **testing**: initial unit test environment

Details

- C++
- schedule requests across workers
- no real changes
- skeleton policy: **modules/ngx_http_upstream_lambda_module.c**

Source code

https://github.com/tylerharter/open-lambda

- **worker**: Lambda server that executes handlers
- **nginx**: load balancer
- **lambda-generator**: old script for generating Python Lambdas
- **node**: container with worker, rethinkdb, and docker
- **util**: scripts for starting/stopping local cluster
- **applications**: OpenLambda applications
- **testing**: initial unit test environment

Details

- Python
- Bundles Lambda function inside Docker container (Alpine)
- To be replaced soon

Source code

https://github.com/tylerharter/open-lambda

- **worker**: Lambda server that executes handlers
- **nginx**: load balancer
- **lambda-generator**: old script for generating Python Lambdas
- **node**: container with worker, rethinkdb, and docker
- **util**: scripts for starting/stopping local cluster
- **applications**: OpenLambda applications
- **testing**: initial unit test environment

Details

- Docker container (name=lambda-node)
- Allows execution of cluster on one machine
- One container simulates one machine
- Contents: Docker, RethinkDB, Lambda worker
- Note: containers inside containers!

Source code

https://github.com/tylerharter/open-lambda

- **worker**: Lambda server that executes handlers
- **nginx**: load balancer
- **lambda-generator**: old script for generating Python Lambdas
- **node**: container with worker, rethinkdb, and docker
- **util**: scripts for starting/stopping local cluster
- **applications**: OpenLambda applications
- **testing**: initial unit test environment

Details

- Python
- util/start-local-cluster.py spins up cluster
- Each node described in util/cluster
- Each node is a “lambda-node” container

Source code

https://github.com/tylerharter/open-lambda

- **worker**: Lambda server that executes handlers
- **nginx**: load balancer
- **lambda-generator**: old script for generating Python Lambdas
- **node**: container with worker, rethinkdb, and docker
- **util**: scripts for starting/stopping local cluster
- **applications**: OpenLambda applications
- **testing**: initial unit test environment

Details

- Various applications and deployment scripts
- Looks at **util/cluster** to determine how to deploy
- Generates **config.json** so JavaScript knows where to issue RPCs

Source code

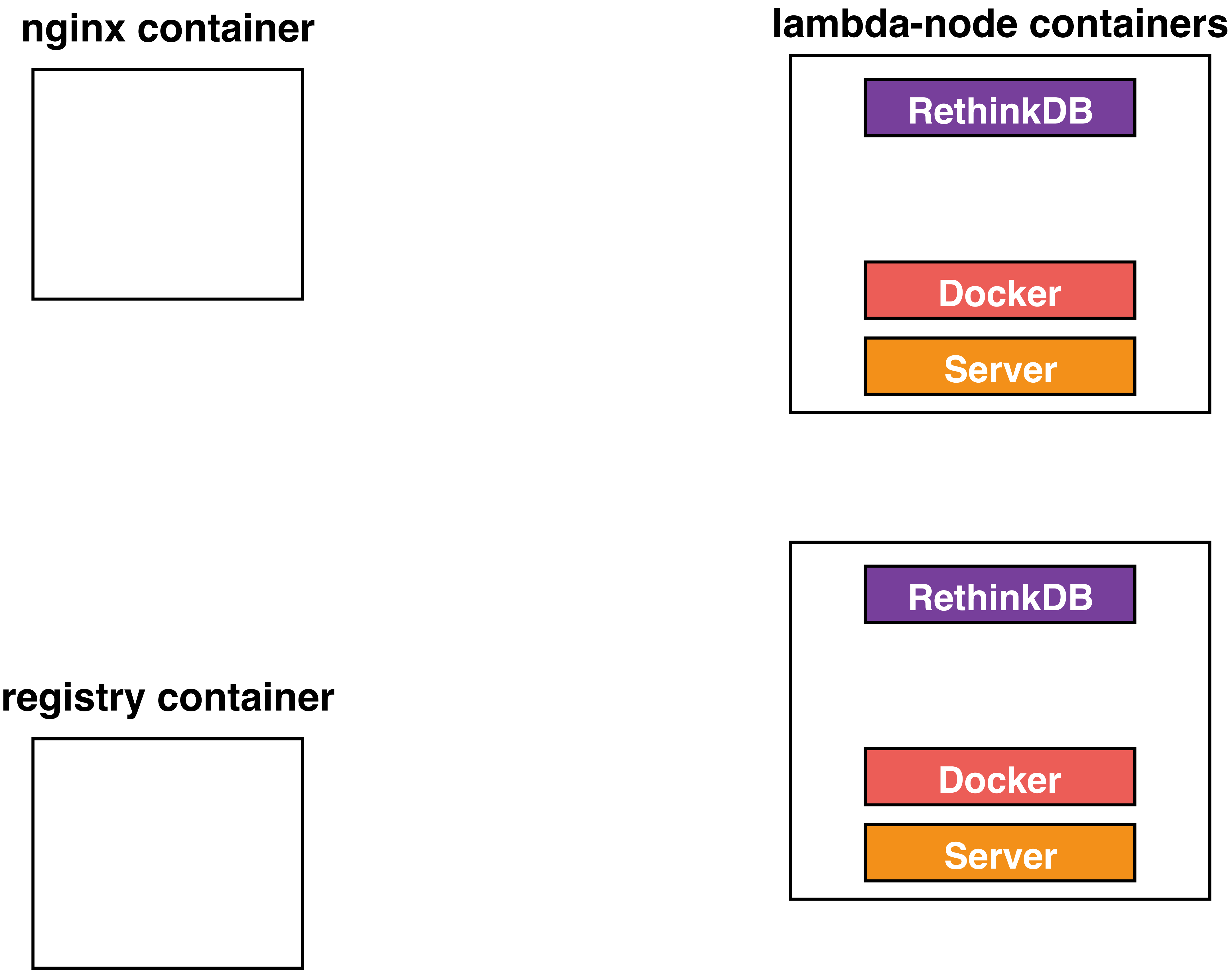
https://github.com/tylerharter/open-lambda

- **worker**: Lambda server that executes handlers
- **nginx**: load balancer
- **lambda-generator**: old script for generating Python Lambdas
- **node**: container with worker, rethinkdb, and docker
- **util**: scripts for starting/stopping local cluster
- **applications**: OpenLambda applications
- **testing**: initial unit test environment

Details

- Python
- Pushes simple Lambdas to Docker registry (localhost:5000)
- Go unit tests in worker depend on these
- Just run “make test” after starting a registry

Architecture (1 phys machine)



nginx container

lambda-node containers

RethinkDB

Docker

Server

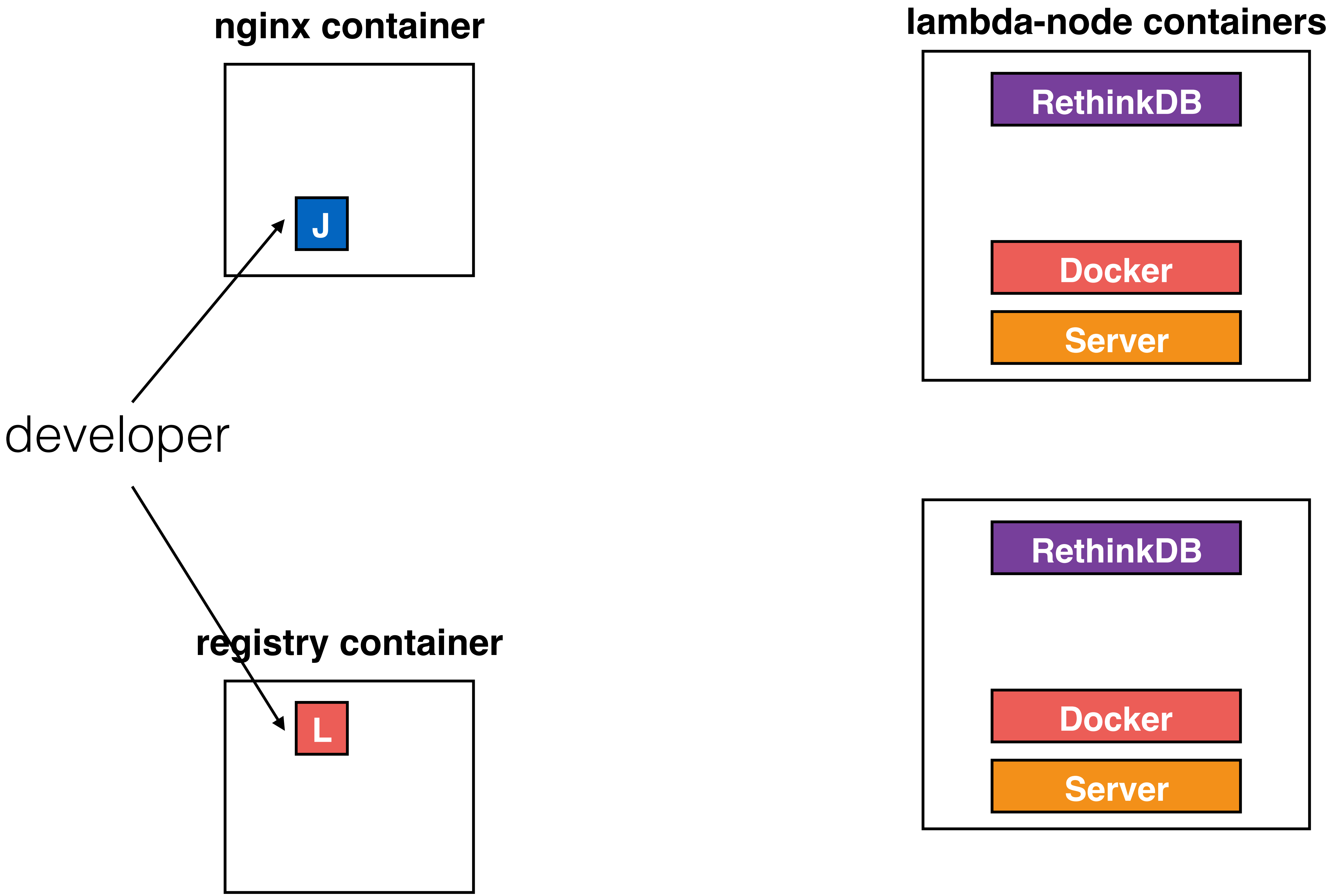
RethinkDB

Docker

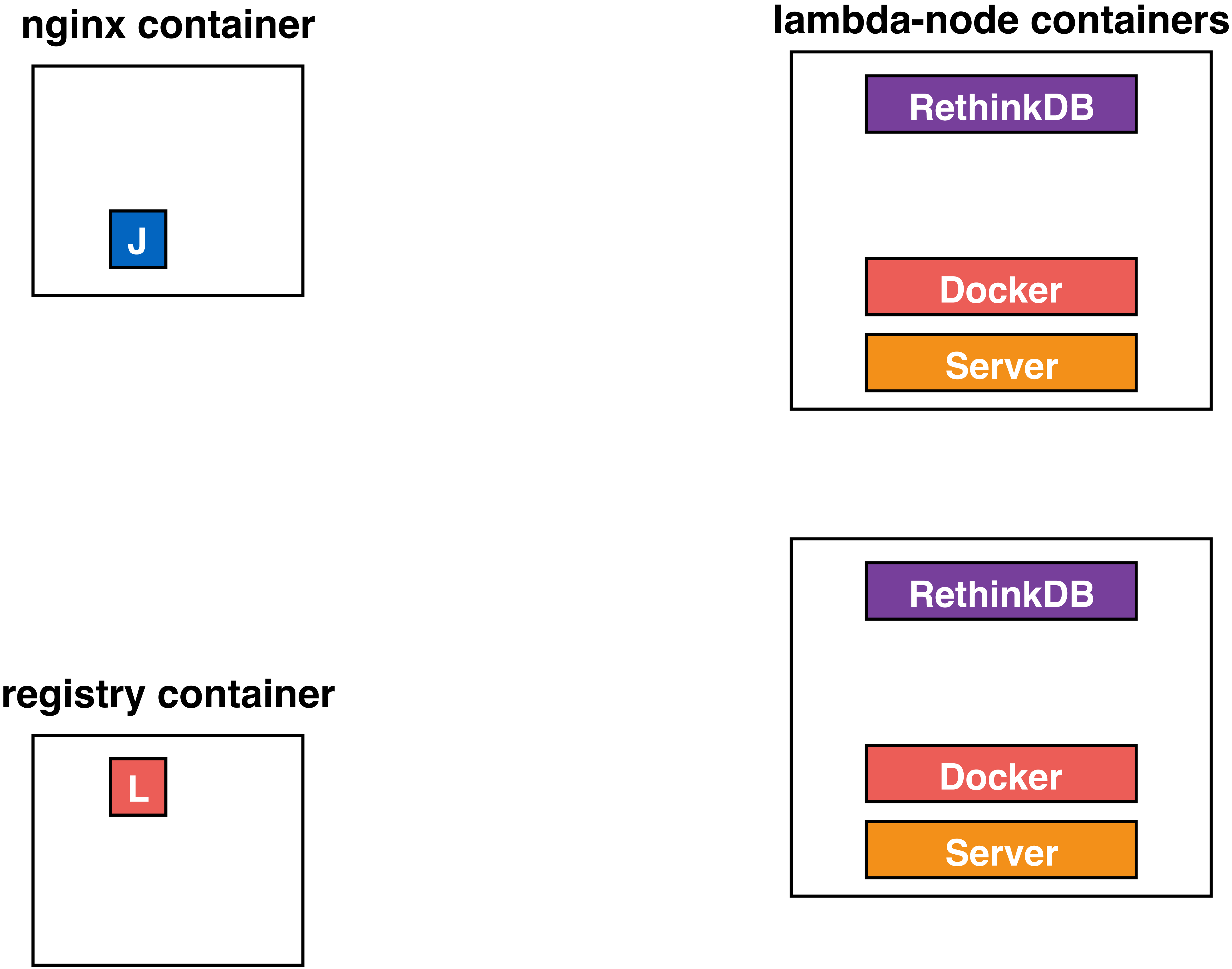
Server

registry container

Architecture (1 phys machine)



Architecture (1 phys machine)



nginx container

J

lambda-node containers

RethinkDB

Docker

Server

RethinkDB

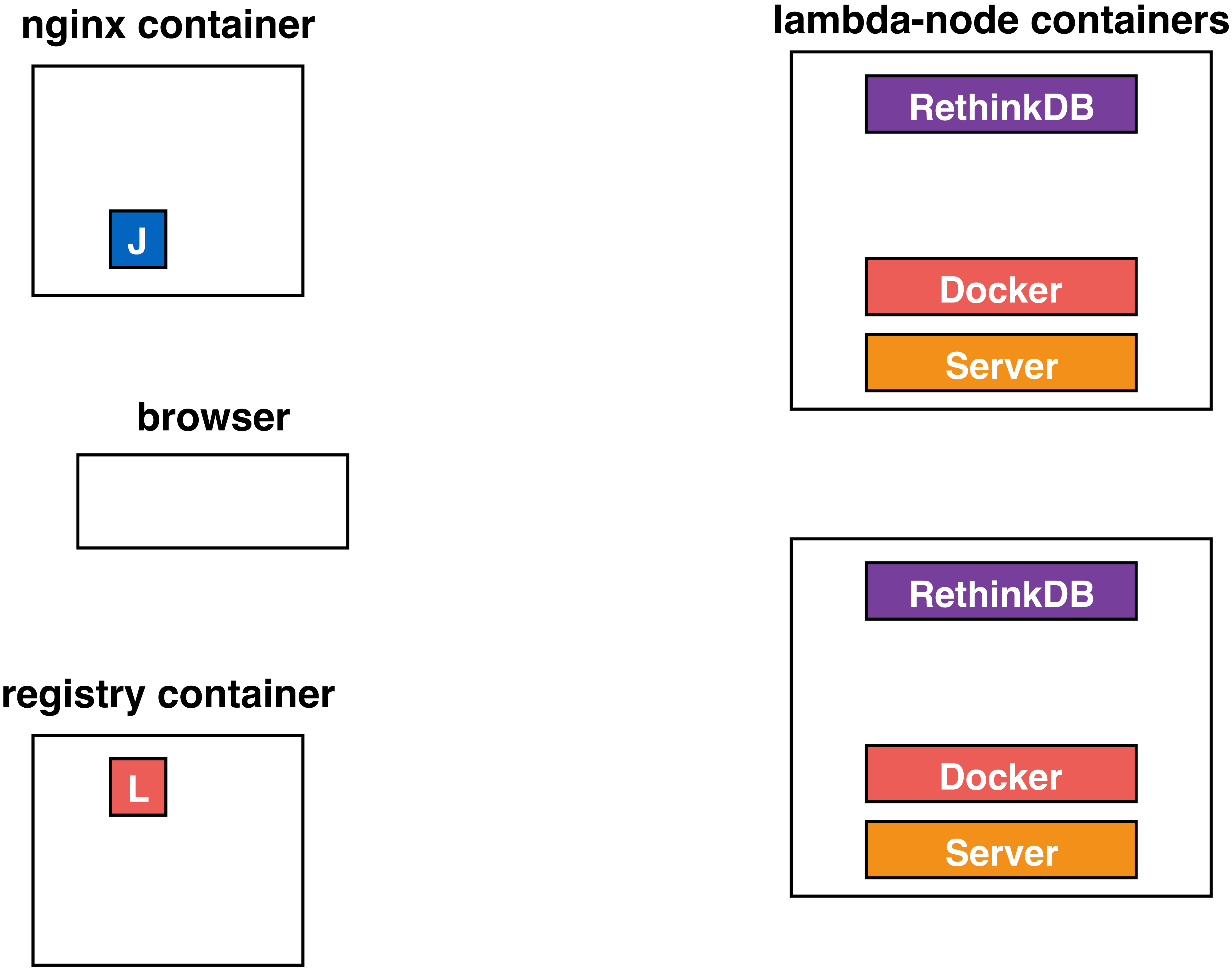
Docker

Server

registry container

L

Architecture (1 phys machine)



nginx container

J

browser

registry container

L

lambda-node containers

RethinkDB

Docker

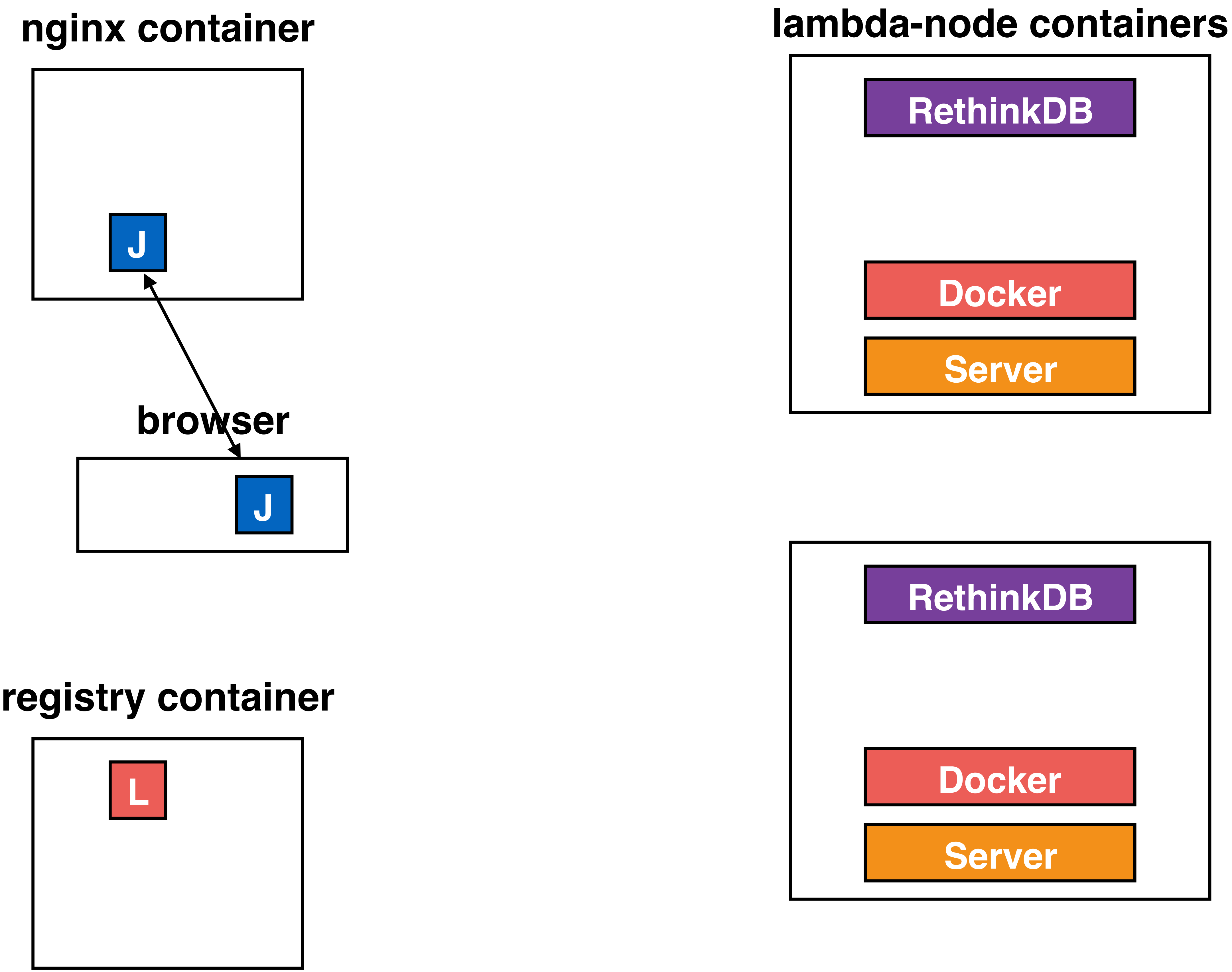
Server

RethinkDB

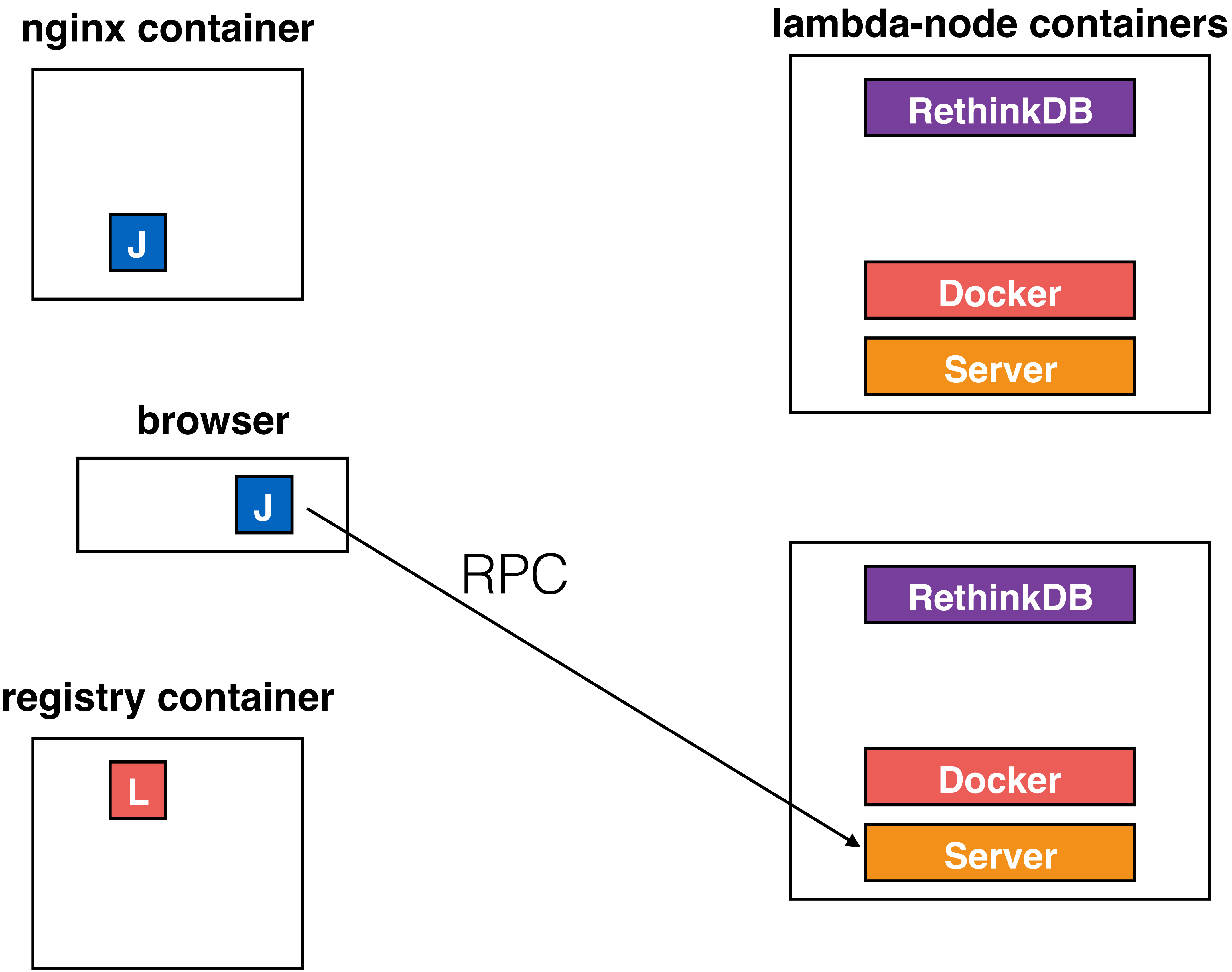
Docker

Server

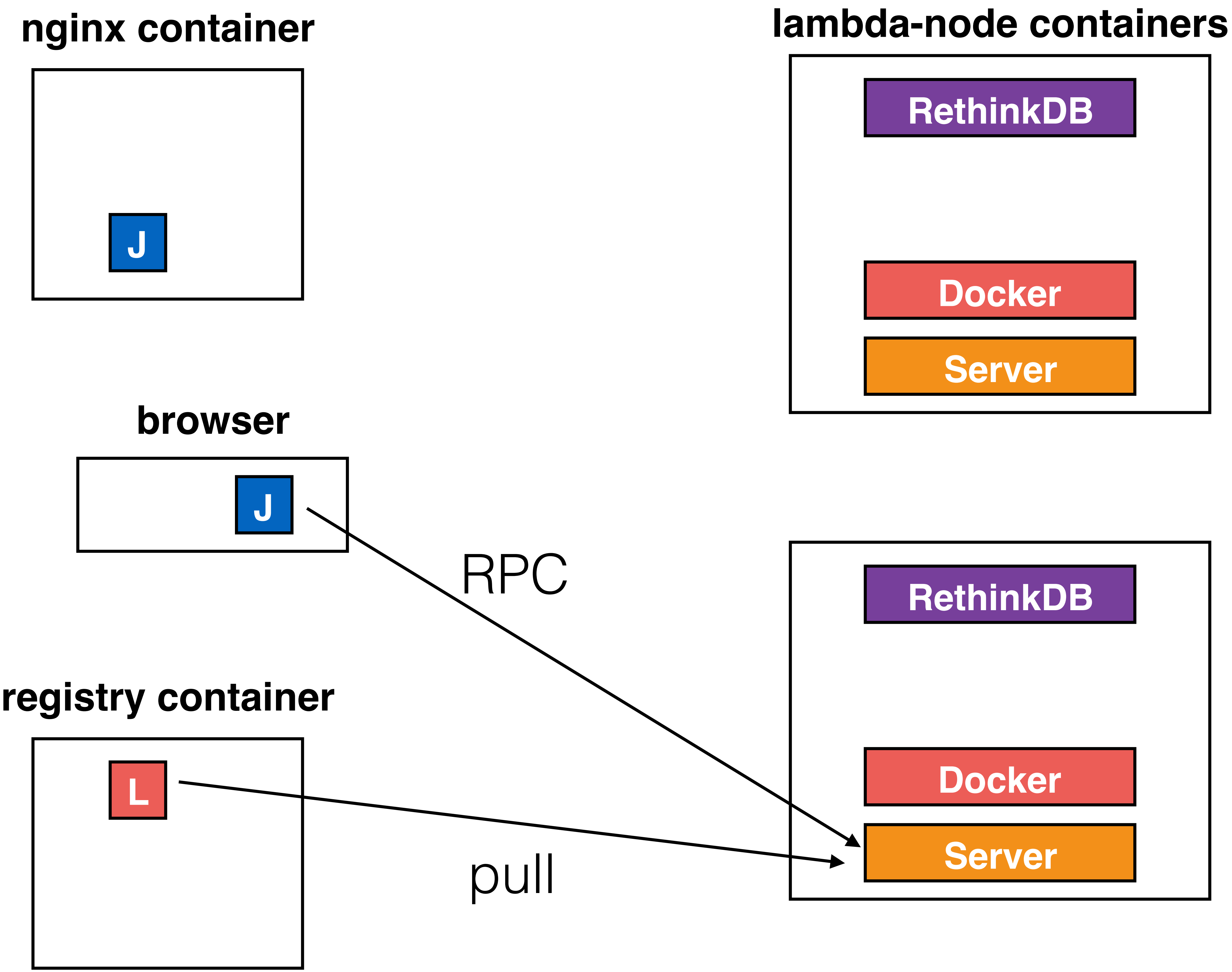
Architecture (1 phys machine)



Architecture (1 phys machine)



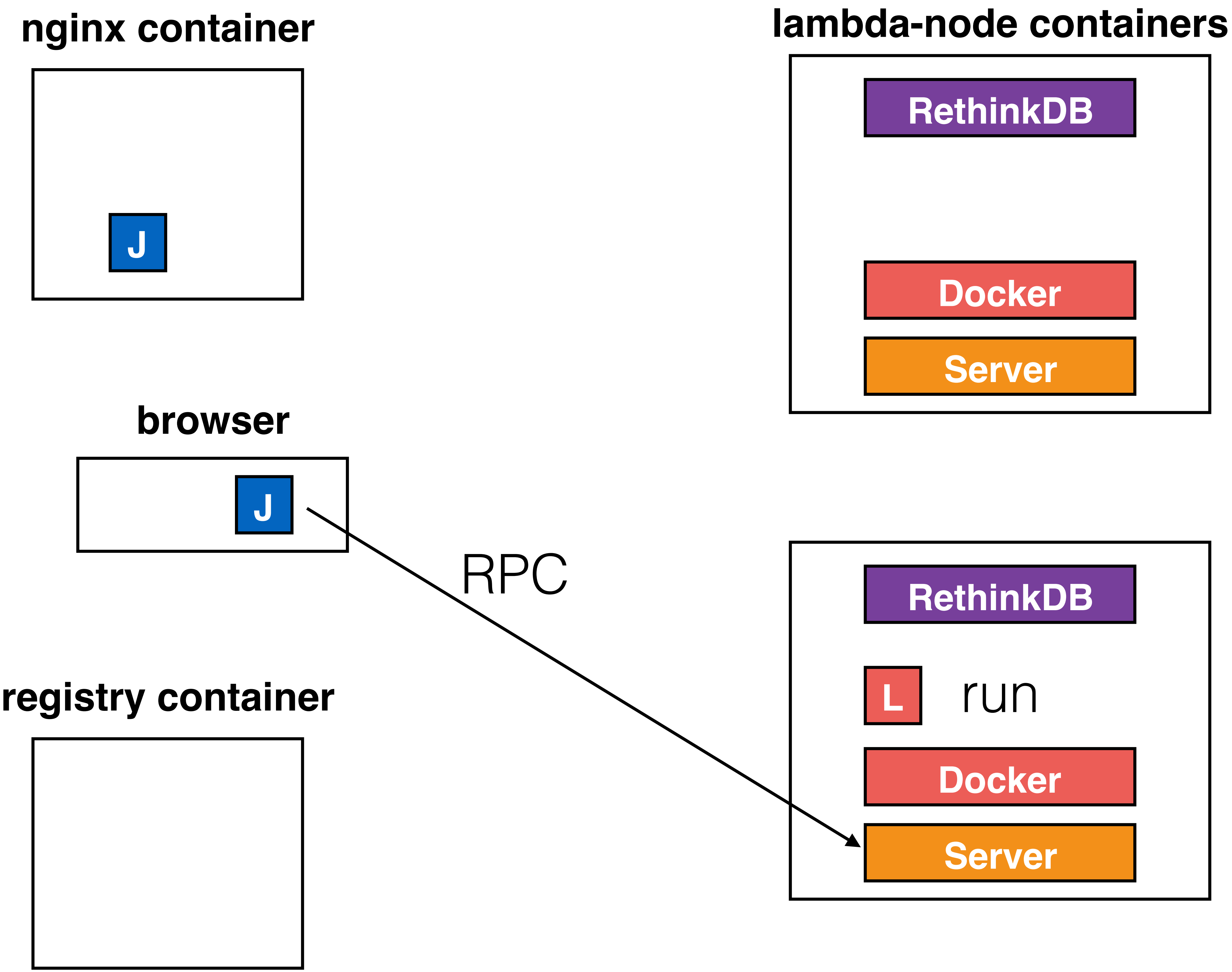
Architecture (1 phys machine)



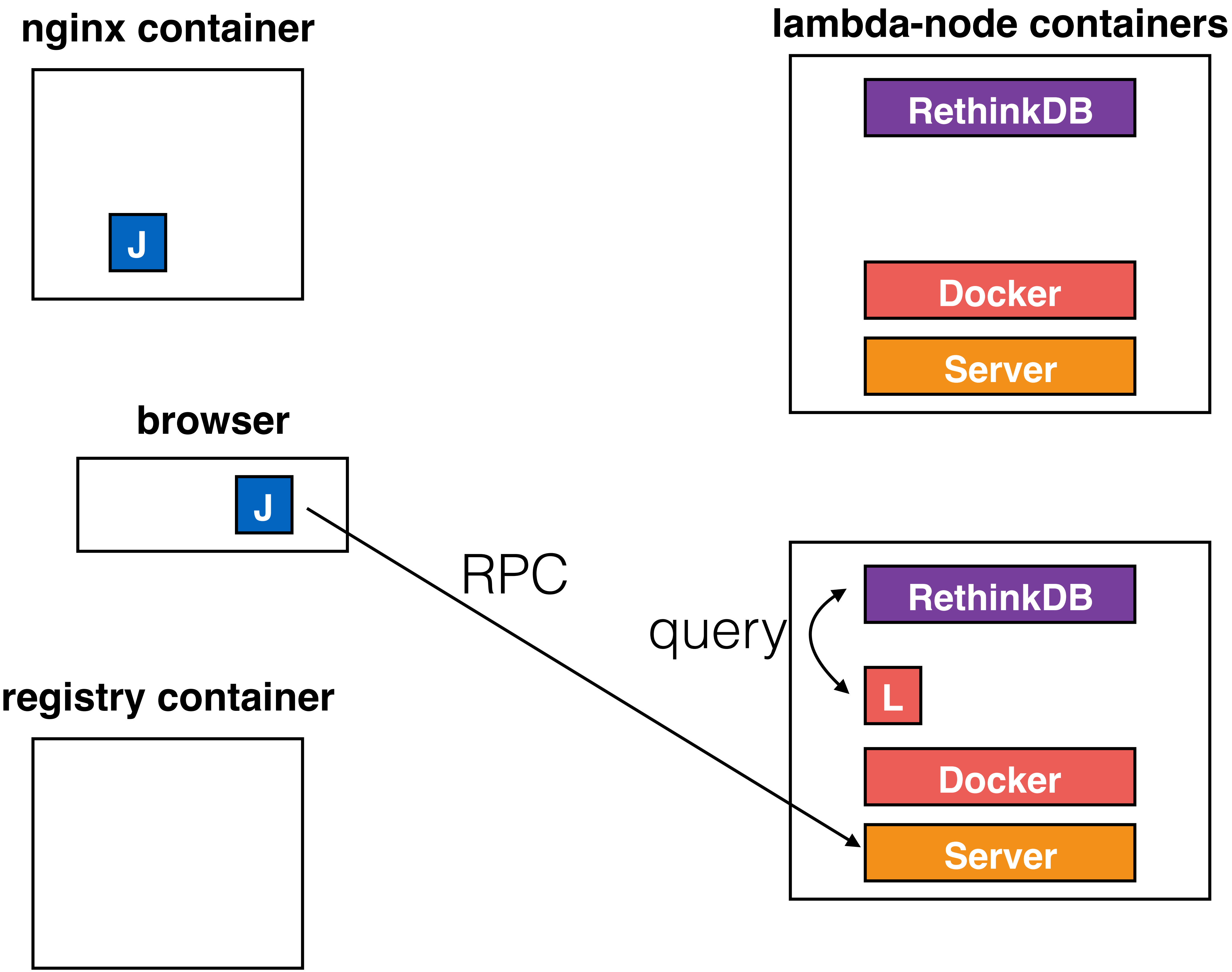
nginx container

lambda-node containers

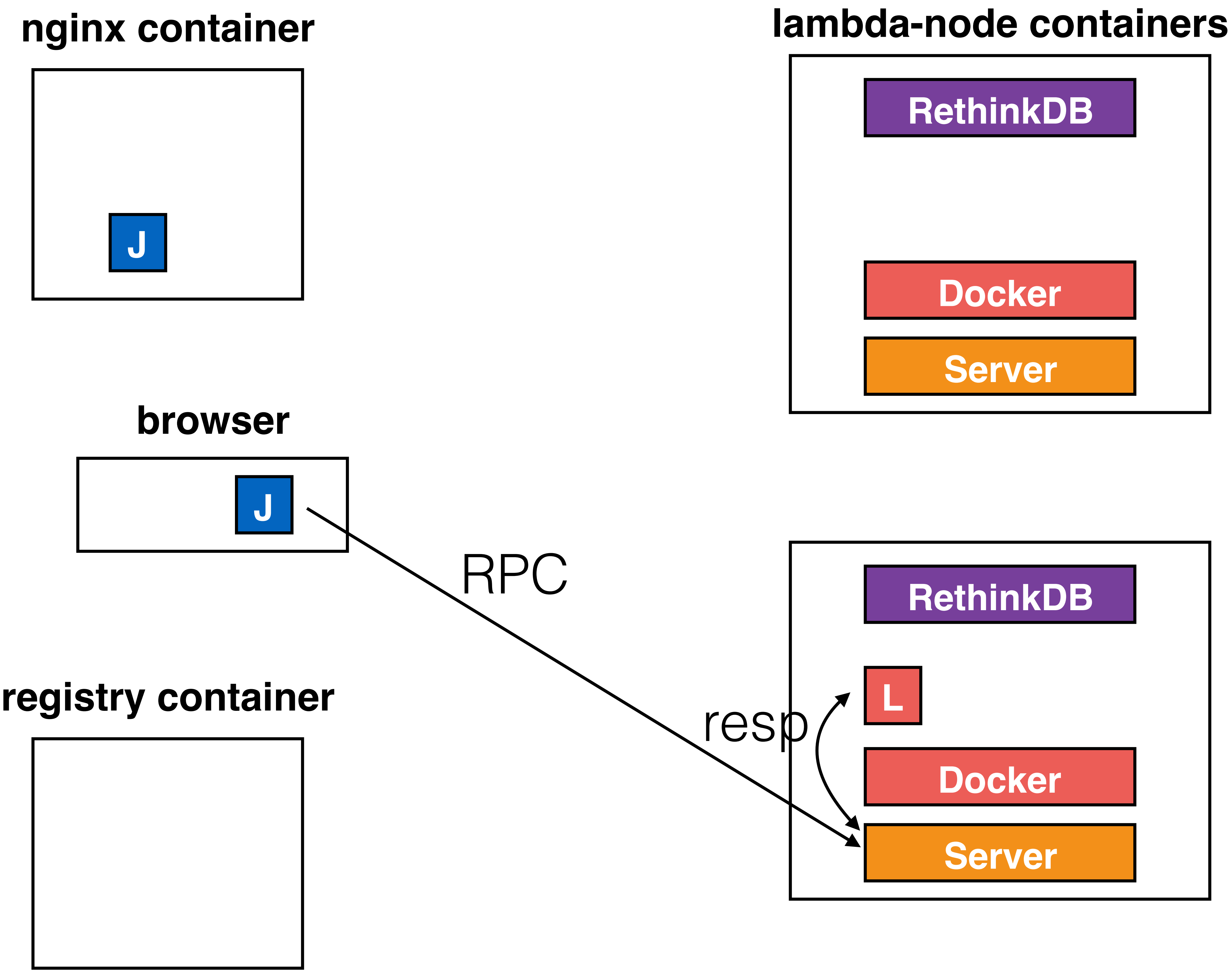
Architecture (1 phys machine)



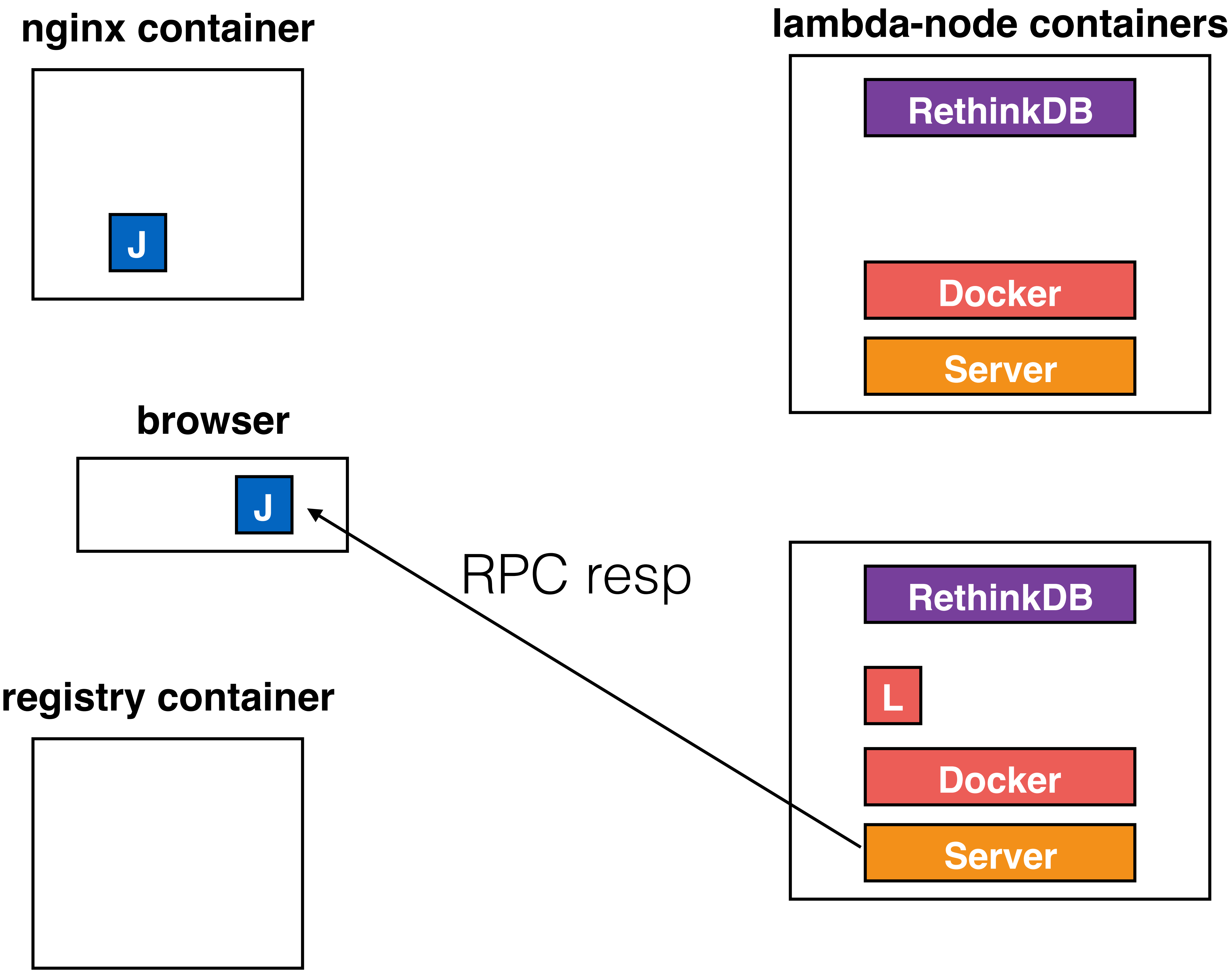
Architecture (1 phys machine)



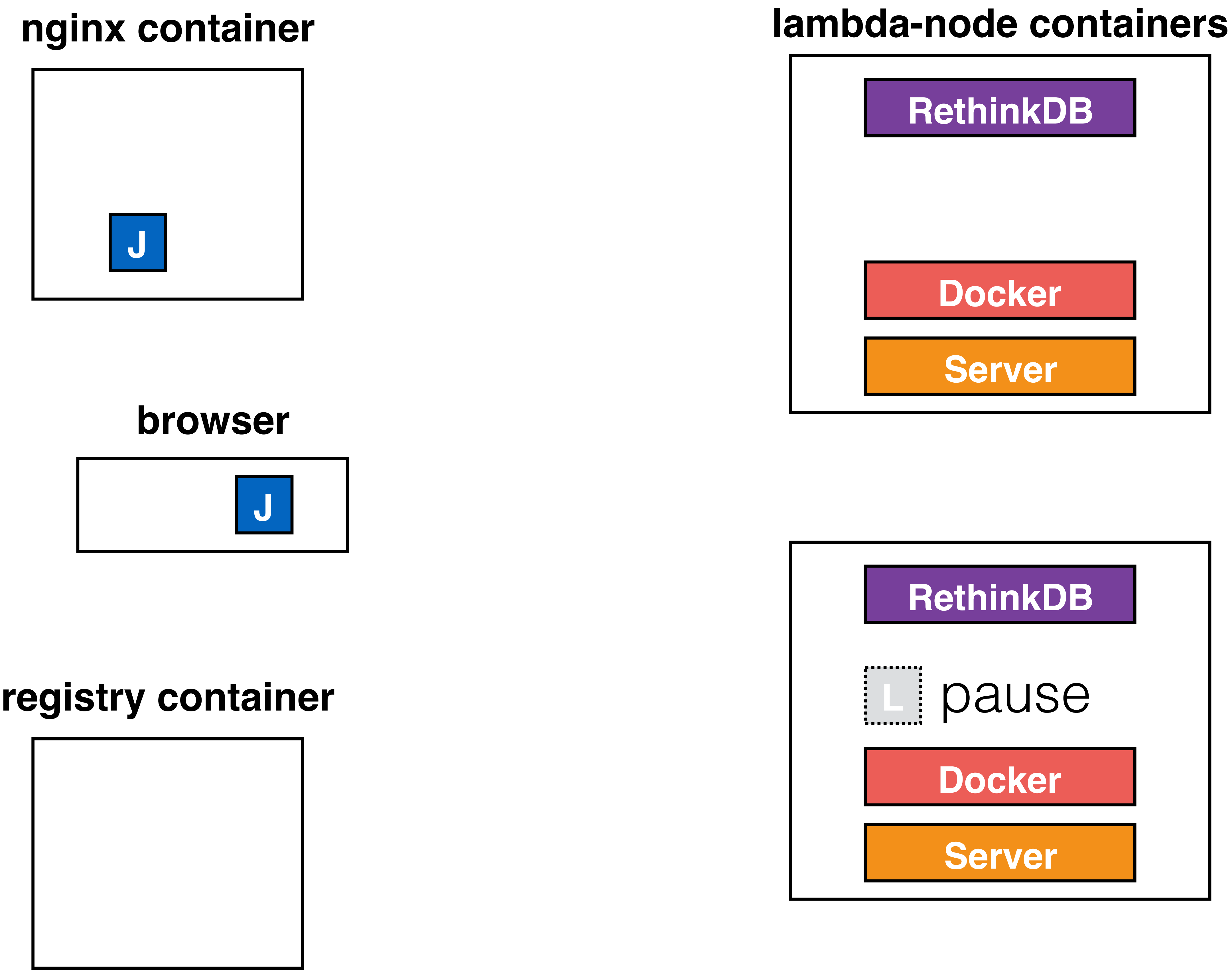
Architecture (1 phys machine)



Architecture (1 phys machine)



Architecture (1 phys machine)



nginx container

J

browser

J

registry container

lambda-node containers

RethinkDB

Docker

Server

RethinkDB

L pause

Docker

Server

Getting started

```
PROMPT> make
...

PROMPT> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL
SIZE
lambda-node         latest              e3c7c9b3680e        4 minutes ago       376.8 MB
ubuntu              trusty              d4751aa1c40a        2 weeks ago         188 MB

PROMPT> ./util/start-local-cluster.py
...

PROMPT> ./applications/pychat/setup.py
...

PROMPT> docker run -d -p 80:80 -v /root/git_co/open-lambda/applications/pychat/static:/
usr/share/nginx/html:ro nginx
...

PROMPT> docker run -d -p 5000:5000 registry:2
...

PROMPT> make test
...
```

Outline

Emerging compute models

Containers vs. Lambdas

Application building

OpenLambda: code overview

Plan projects: discussion