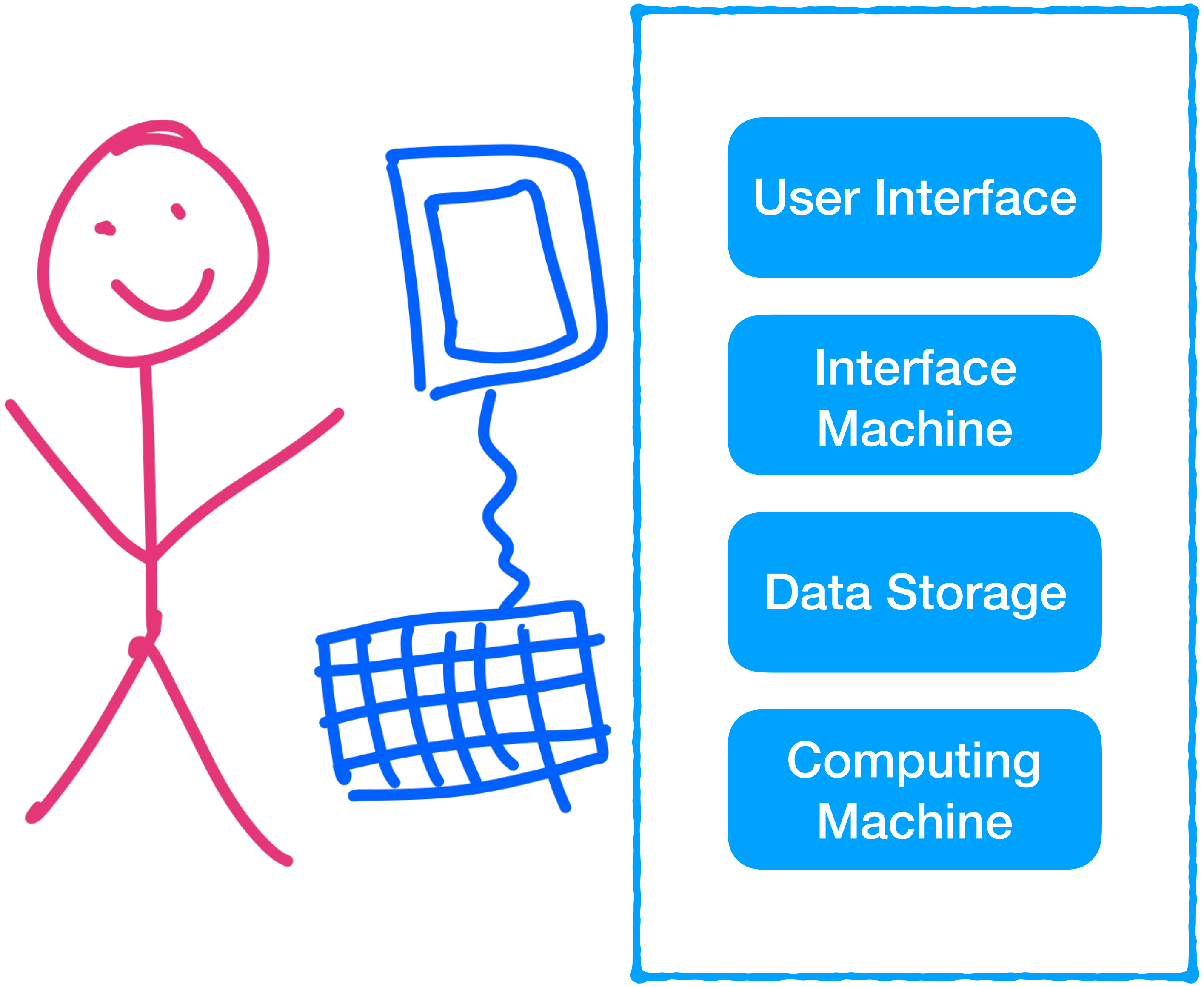
Deploying Algorithms

Biostatistics 140.711

Running Your Algorithm



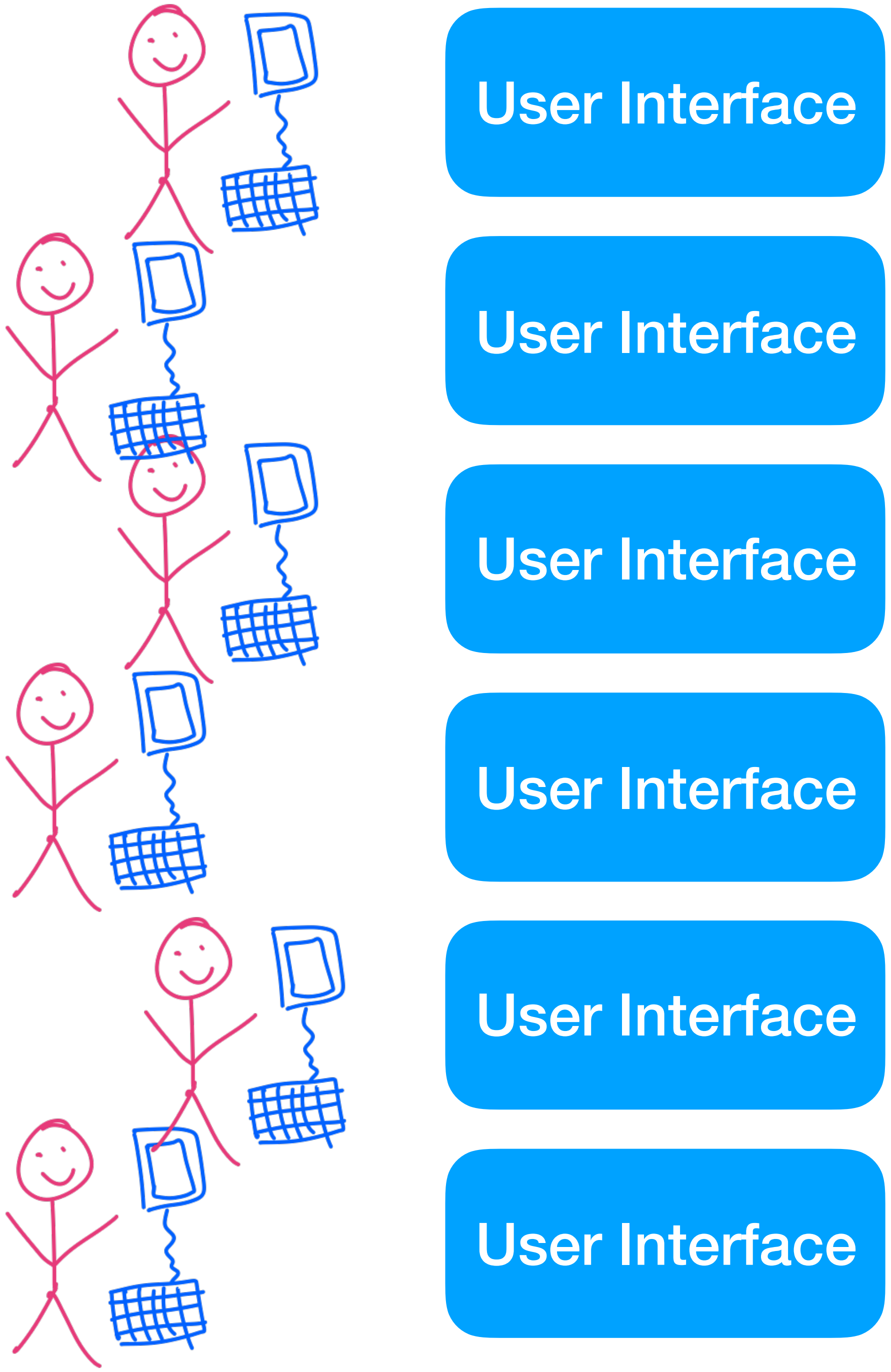
User Interface

Interface Machine

Data Storage

Computing Machine

Deploying Your Algorithm



User Interface

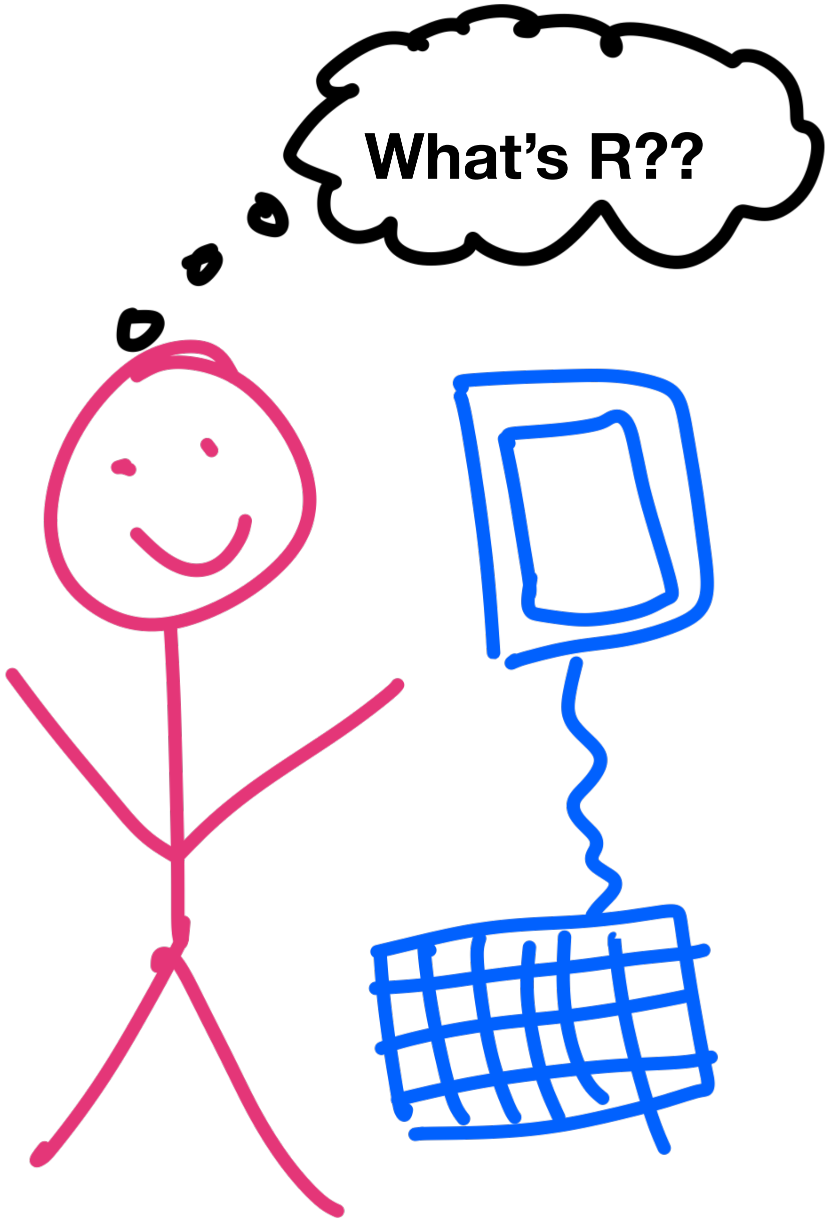
User Interface

User Interface

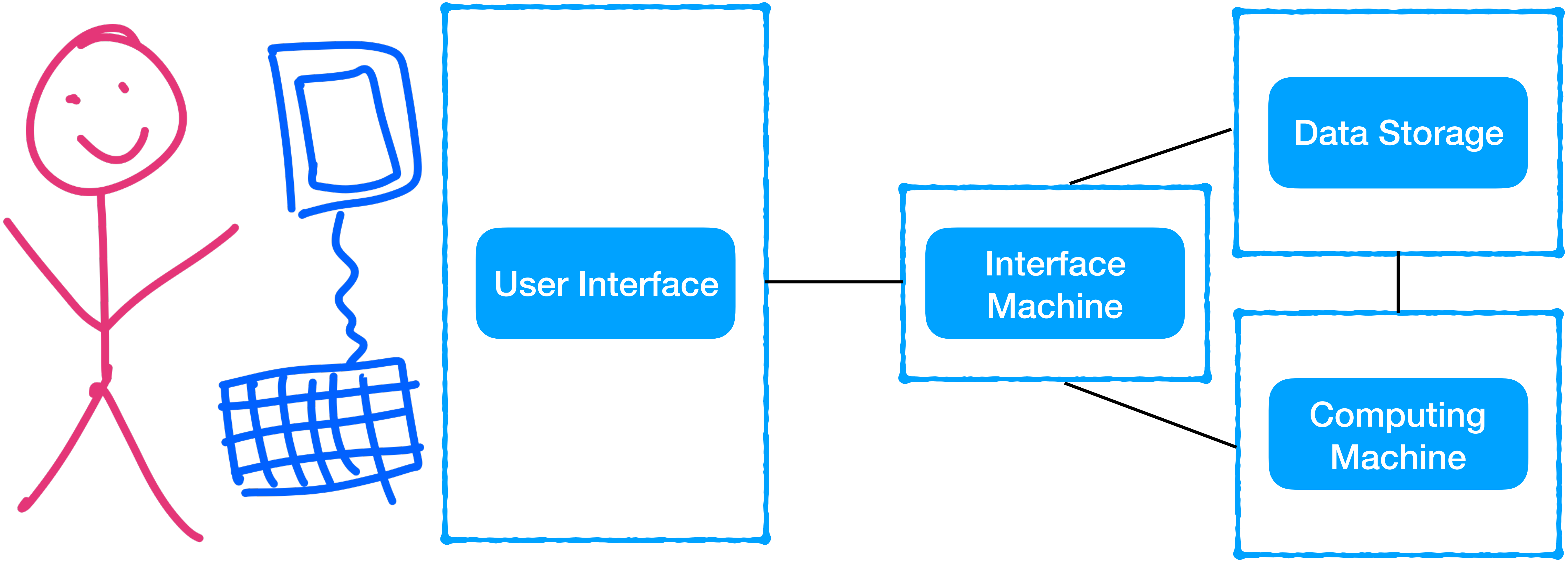
User Interface

User Interface

User Interface



Deploying Your Algorithm



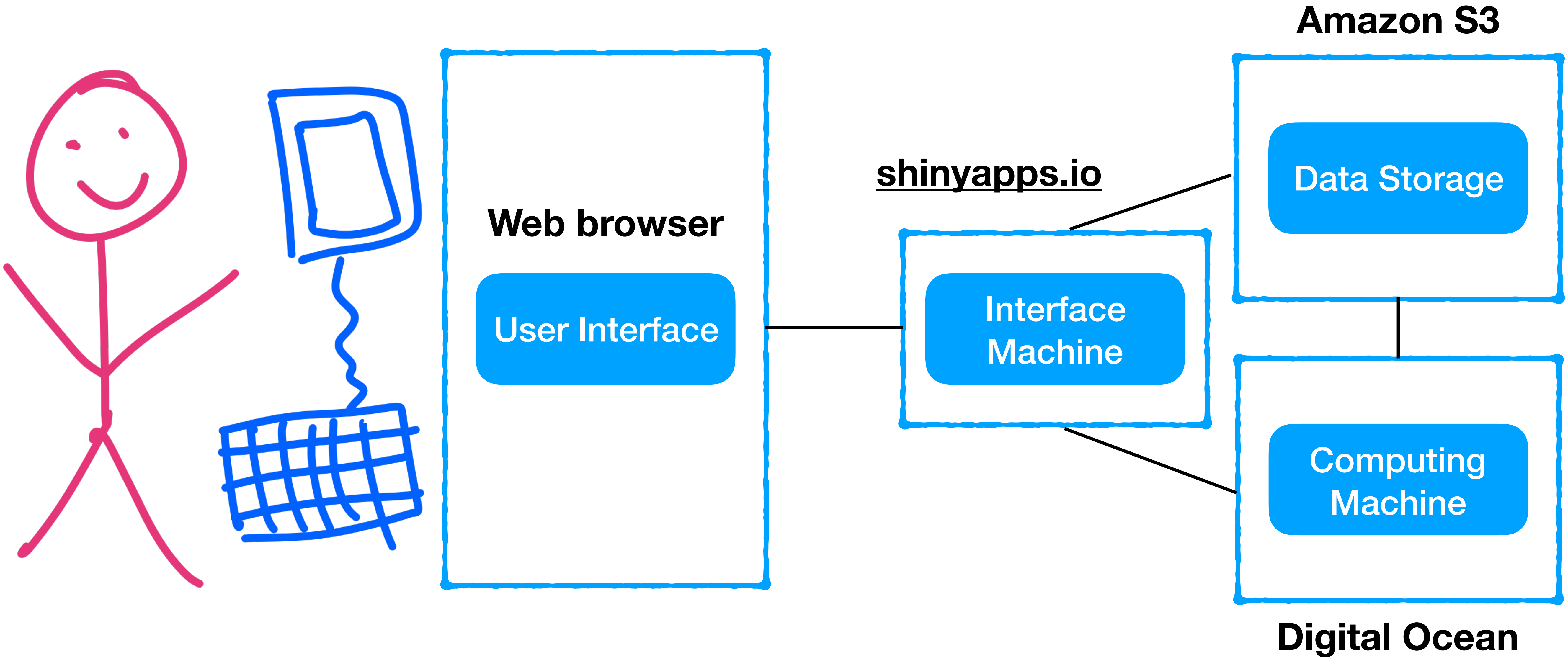
User Interface

Interface Machine

Data Storage

Computing Machine

Deploying Your Algorithm



Amazon S3

Web browser

User Interface

shinyapps.io

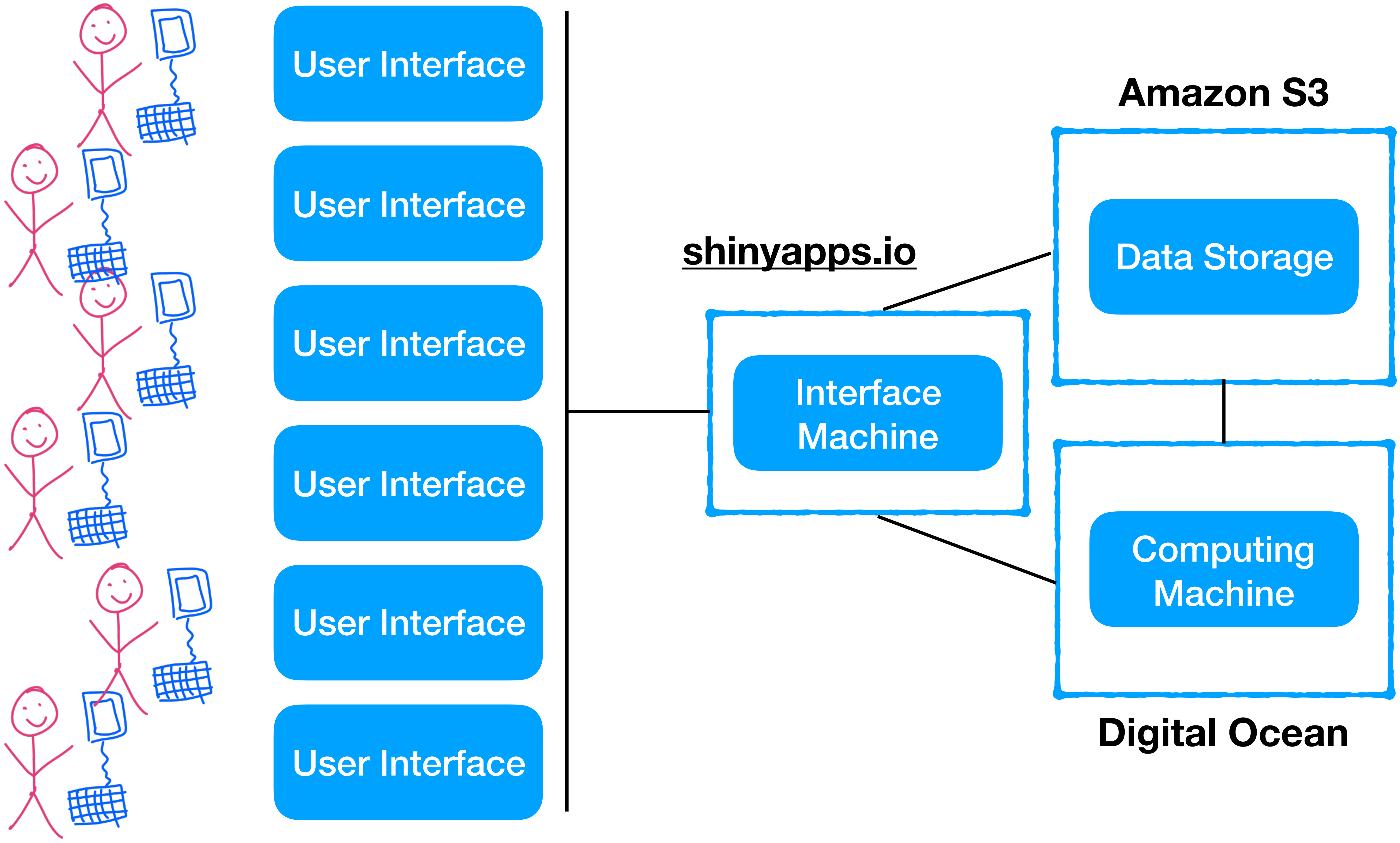
Interface Machine

Data Storage

Computing Machine

Digital Ocean

Deploying Your Algorithm



User Interface

User Interface

User Interface

User Interface

User Interface

User Interface

shinyapps.io

Interface Machine

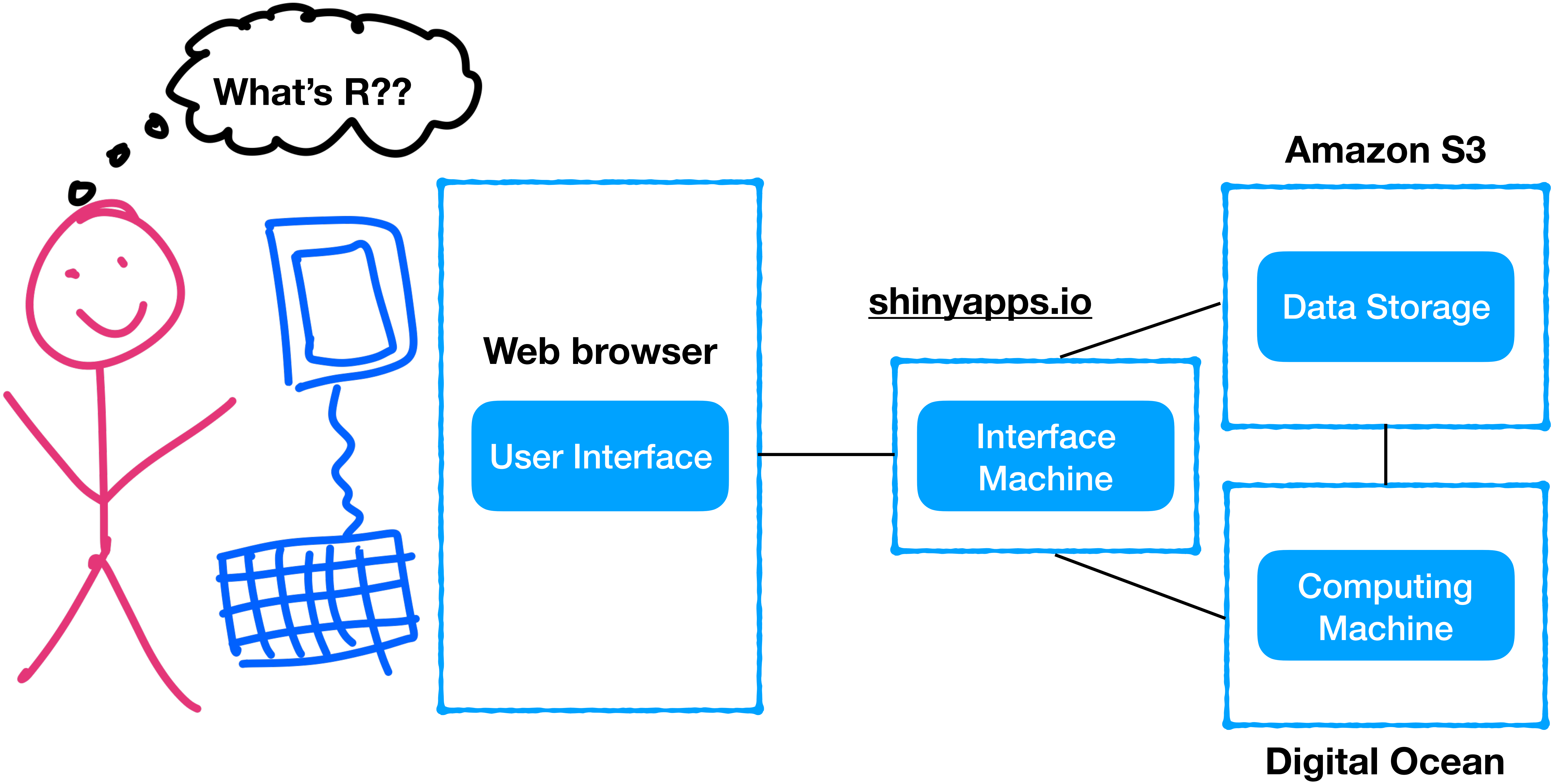
Amazon S3

Data Storage

Computing Machine

Digital Ocean

Deploying Your Algorithm



What's R??

Web browser

User Interface

shinyapps.io

Interface Machine

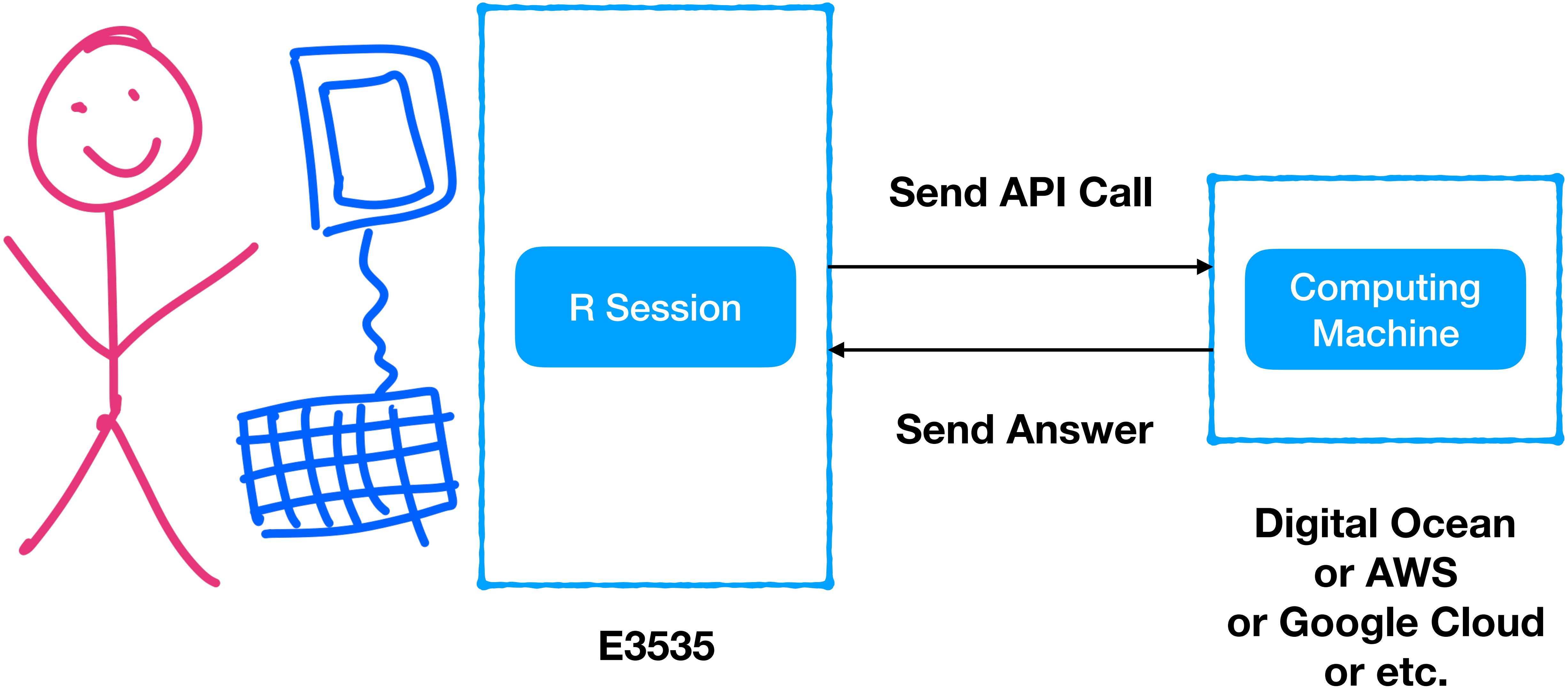
Amazon S3

Data Storage

Computing Machine

Digital Ocean

Deploying Your Algorithm



Deployment Strategy

- The **plumber** package converts an R function into a REST API

- Part of the “RStudio Universe” but does not require RStudio

- Can run on a server running R and provide a web interface

- Requires writing R code with a few extra markups (a little bit like roxygen2)

Ozone Prediction

- Given a temperature value, what level of ozone should we expect?
- This is a “hard problem” in general but is easily solved with regression
- We can provide a web interface that can take temperature as input and provide predicted ozone as output

Server: Ozone Prediction

```
## Predict Ozone Levels Given Temperature
library(splines)
library(datasets)
fit <- lm(Ozone ~ ns(Temp, 2), data = airquality)

#* Predict Ozone from Temperature
#* @param temp The temperature input
#* @get /ozone
ozone_predict <- function(temp) {
        ## Check input type
        temp <- as.numeric(temp)

        ## Make prediction from fitted model
        p <- predict(fit, data.frame(Temp = temp))

        ## Return predicted value
        as.numeric(p)
}
```

Client: Ozone Prediction

```r
library(jsonlite); library(curl); library(glue)

ozone_predict_remote <- function(temp) {
        ## Construct API URL
        cmd <- glue("http://67.205.166.80:8000/ozone?",
                    "temp={temp}")

        ## Open connection to the web server
        con <- curl(cmd)

        ## Read the answer from the server
        ans <- readLines(con, 1, warn = FALSE)

        ## Close server connection
        close(con)

        ## Convert answer from JSON and return
        fromJSON(ans)
}
```

Run Function

Sending More Data

- Typical web APIs are expecting lightweight inputs
- Not realistic to pass large data objects via URL strings
- For larger input data we need a different strategy where data can be stored/retrieved elsewhere
- The **aws.s3** package can be used to store/retrieve data from S3 (installed from GitHub)
    - remotes::install_github("cloudyr/aws.s3")

Confidence Intervals for the Median

- No closed-form solution
- Use the bootstrap to compute!
- Write a function called `confint_median(x, N)` that takes a vector of observations and a number of bootstrap iterations
- Return a vector containing the lower 2.5% and upper 97.5% of the median via the bootstrap

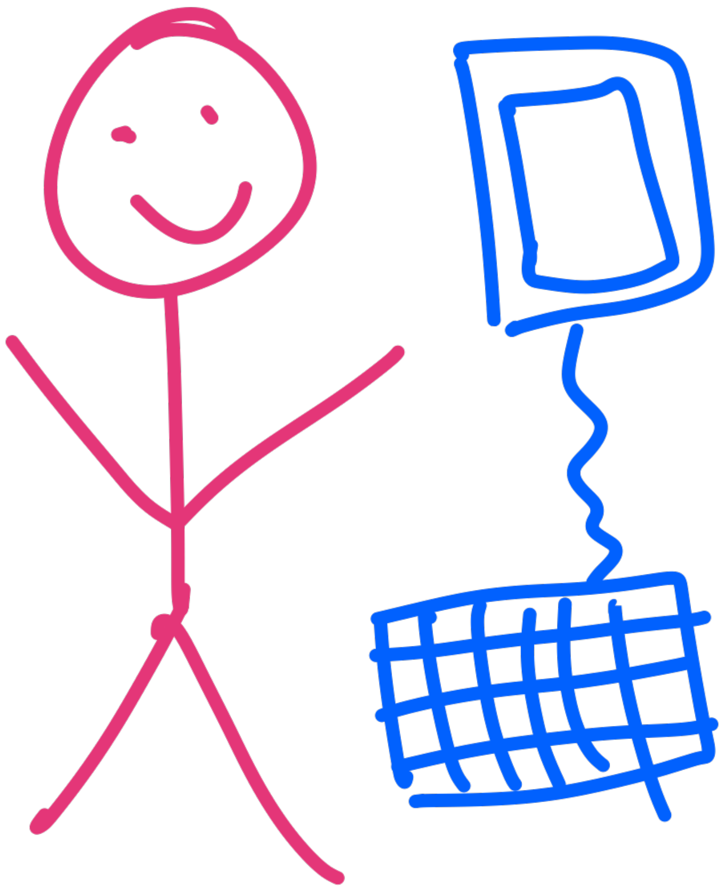
Confidence Intervals for the Median

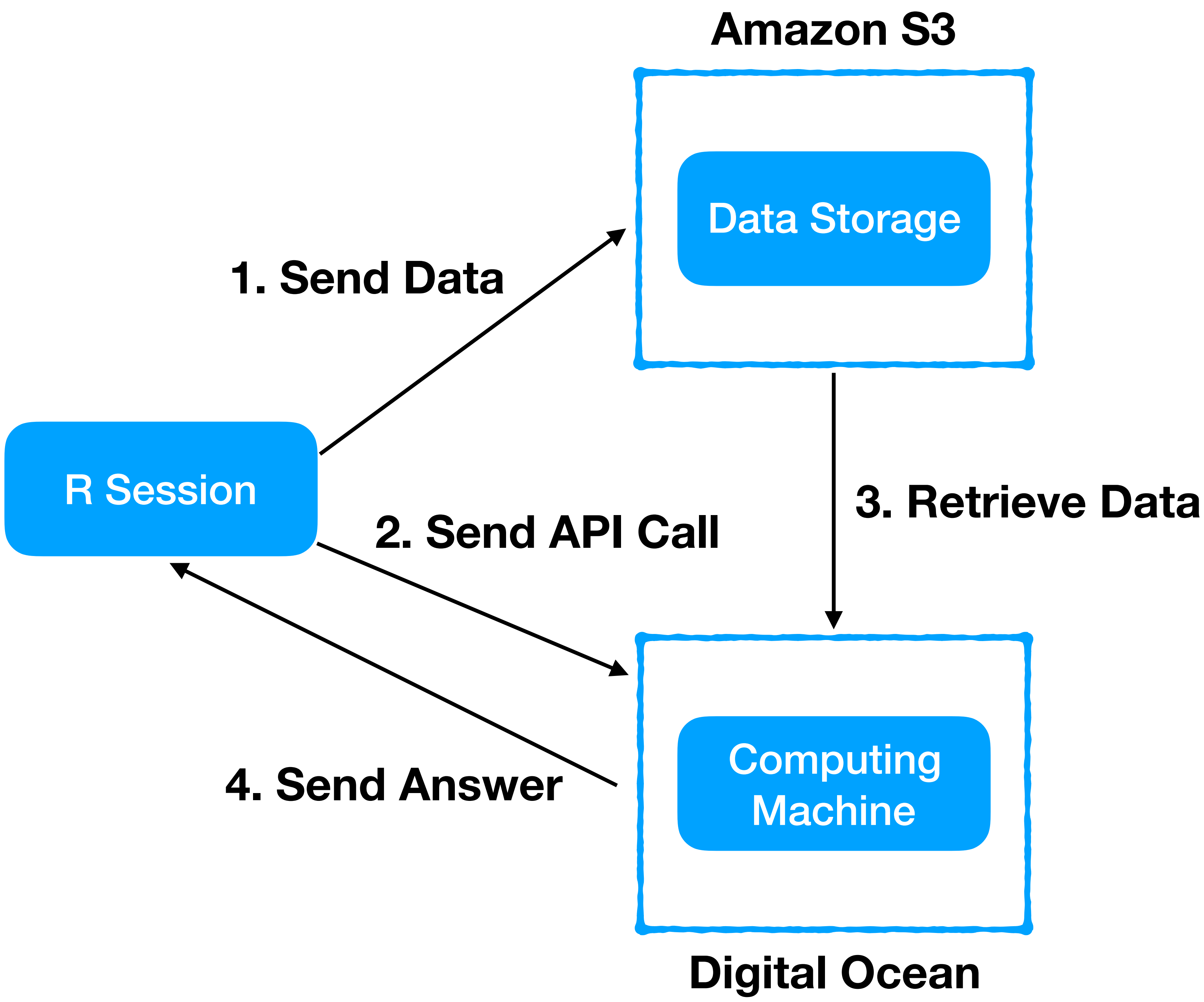
```
confint_median <- function(x, N = 1000) {
        ## Coerce to numeric
        x <- as.numeric(x)

        ## Remove missing values
        x <- x[!is.na(x)]

        if(length(x) == 0L)
                stop("no non-missing data values")
        nobs <- length(x)
        med <- replicate(N, {
                x.new <- sample(x, nobs, replace = TRUE)
                median(x.new)
        })
        quantile(med, c(0.025, 0.975))
}
```

Deploying This Function

- Storage
  - Copy input data from user to a the storage server
- Compute
  - Deploy the CI algorithm to the compute server
  - Read data from the storage server
  - Compute answer and return to the user

Running Your Algorithm





Amazon S3

Data Storage

1. Send Data

R Session

2. Send API Call

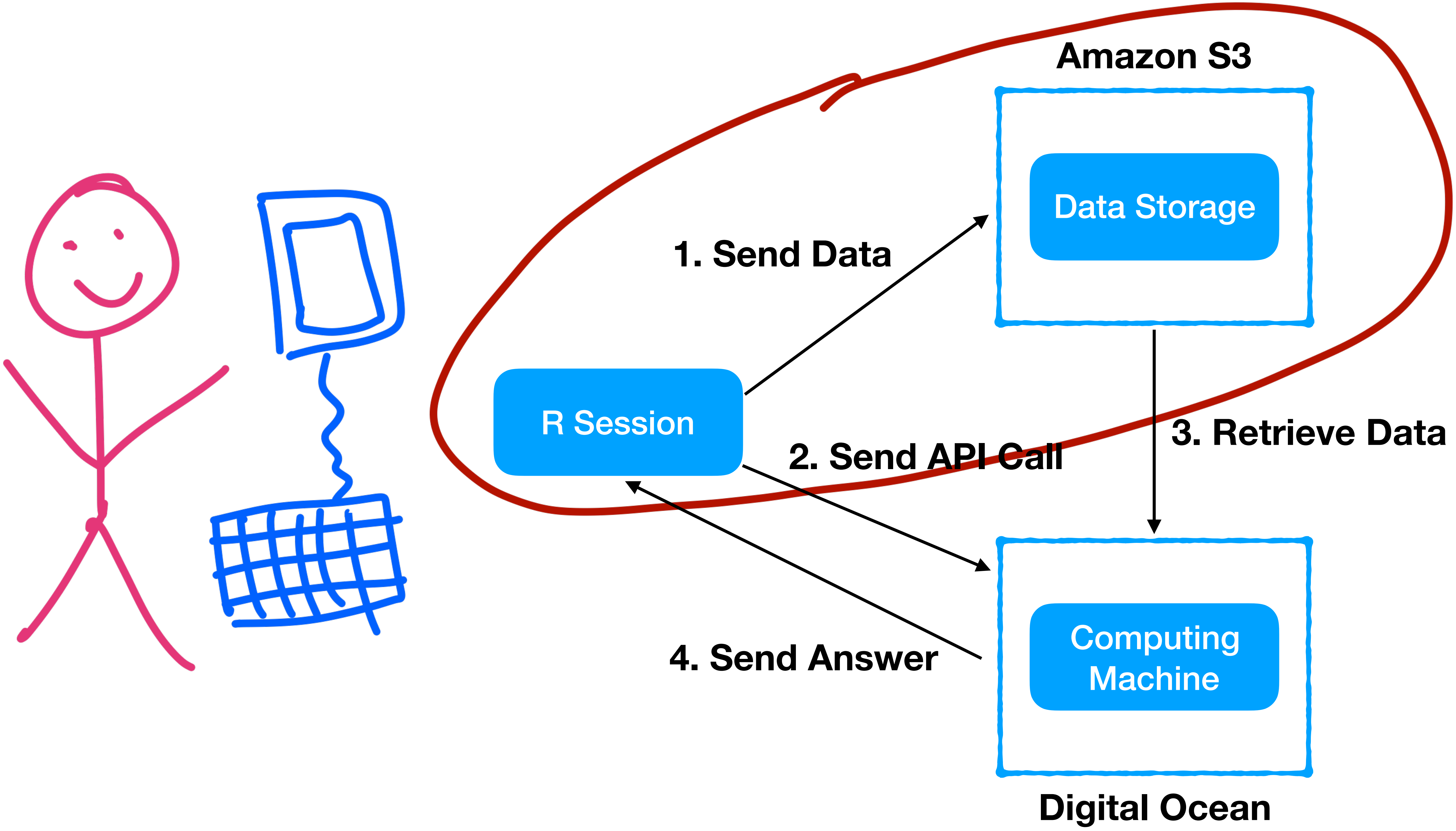
3. Retrieve Data

4. Send Answer

Computing Machine

Digital Ocean

Running Your Algorithm



Amazon S3

Data Storage

1. Send Data

R Session

2. Send API Call

3. Retrieve Data

4. Send Answer

Computing Machine

Digital Ocean

Save to AWS S3

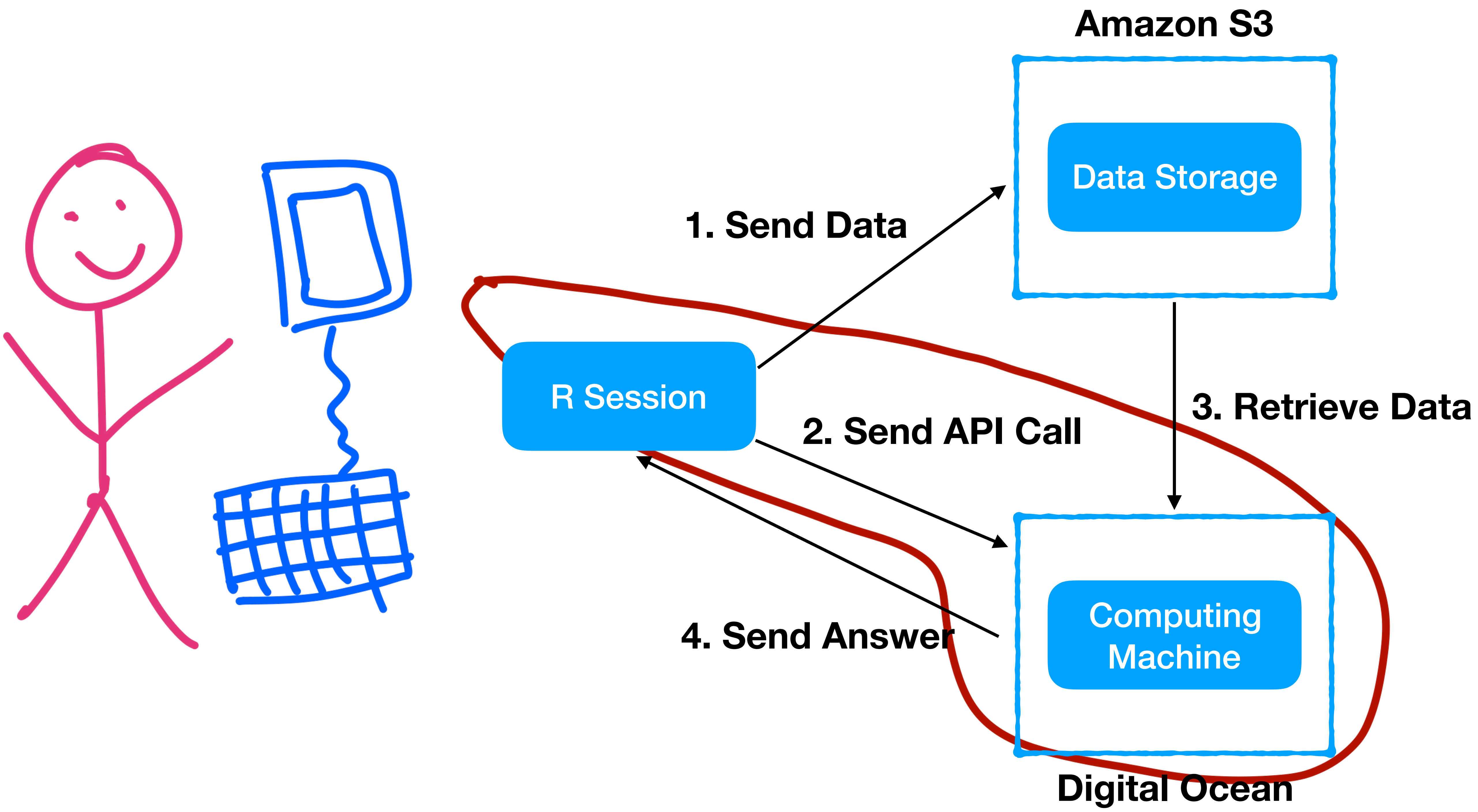
```r
median_CI <- function(x, N = 1000) {
        bucket <- "confint"
        key <- "xdata"
        val <- s3saveRDS(x, key, bucket)

        cmd <- glue("http://67.205.166.80:8000/confint?",
                    "key={key}&bucket={bucket}&N={N}")
        con <- curl(cmd)

        tryCatch({
                ans <- readLines(con, 1, warn = FALSE)
        }, finally = {
                ## Close server connection
                close(con)
        })
        ## Convert answer from JSON and return
        fromJSON(ans)
}
```

Construct API URL

Read from server

Amazon S3

Running Your Algorithm



Amazon S3

Data Storage

1. Send Data

R Session

2. Send API Call

3. Retrieve Data

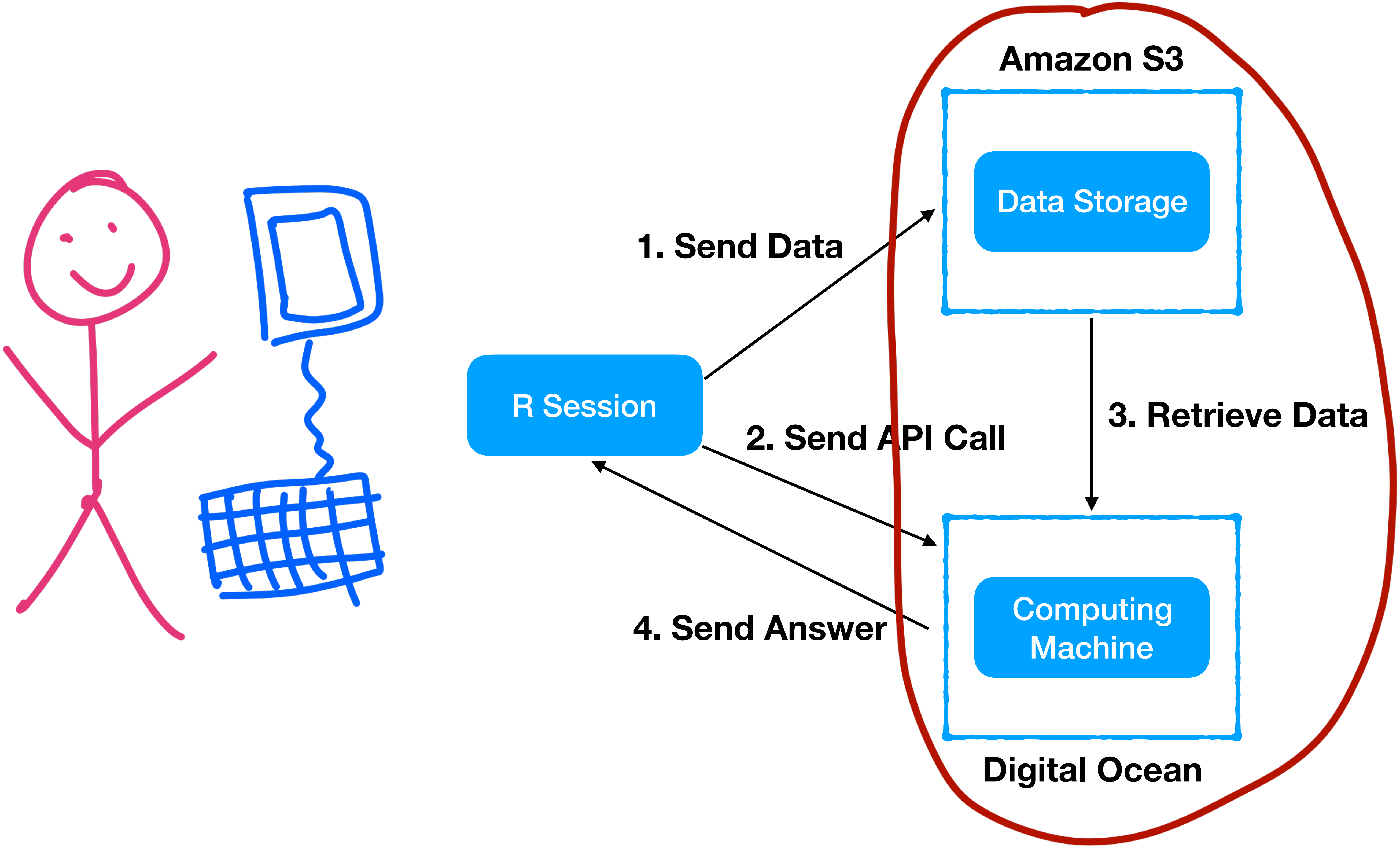
4. Send Answer

Computing Machine

Digital Ocean

API Demo

Running Your Algorithm



Amazon S3

Data Storage

1. Send Data

R Session

2. Send API Call

3. Retrieve Data

Computing Machine

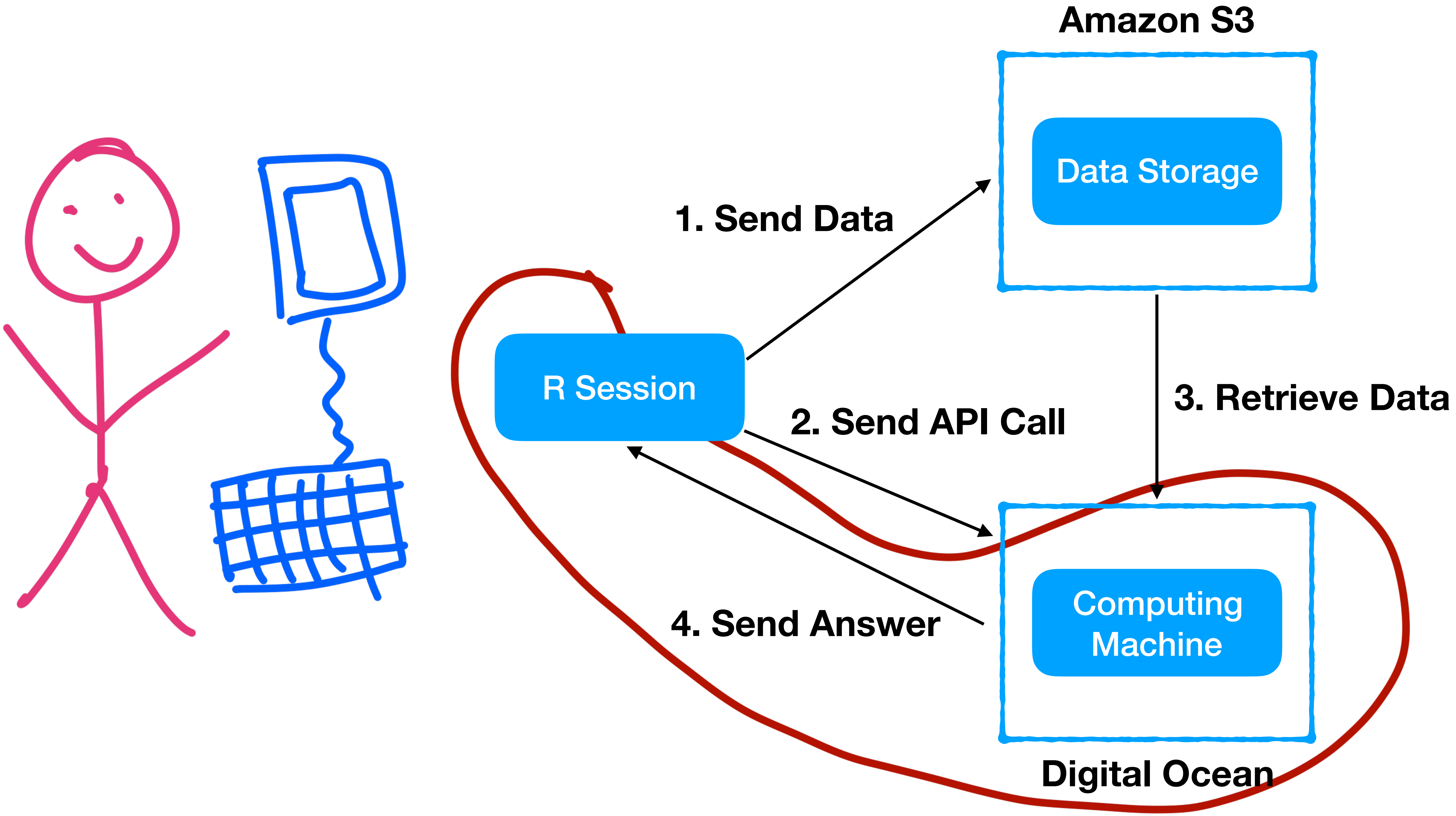
4. Send Answer

Digital Ocean

Retrieving Data from S3

```
#* Compute the 95% bootstrap confidence interval for the median
#* @param key the S3 key for the data
#* @param bucket the name of the bucket where the data live
#* @param N the number of bootstrap iterations
#* @get /confint
confint_median_compute <- function(key, bucket, N) {
        ## Make sure data is proper type
        key <- as.character(key)
        bucket <- as.character(bucket)
        N <- as.integer(N)

        ## Read data from S3
        x <- s3readRDS(key, bucket = bucket)

        ## Compute the confidence interval
        confint_median(x, N)
}
```

Running Your Algorithm



Amazon S3

Data Storage

1. Send Data

R Session

2. Send API Call

3. Retrieve Data

4. Send Answer

Computing Machine

Digital Ocean

Summary

- Creating APIs allows non-R-programmers access to your code in a standardized manner

- The plumber package translates R functions in to web API interfaces

- Amazon S3 can serve as a data intermediary if needed (via the aws.s3 package)

- Getting all the pieces to work right is tricky and everything will be different next month