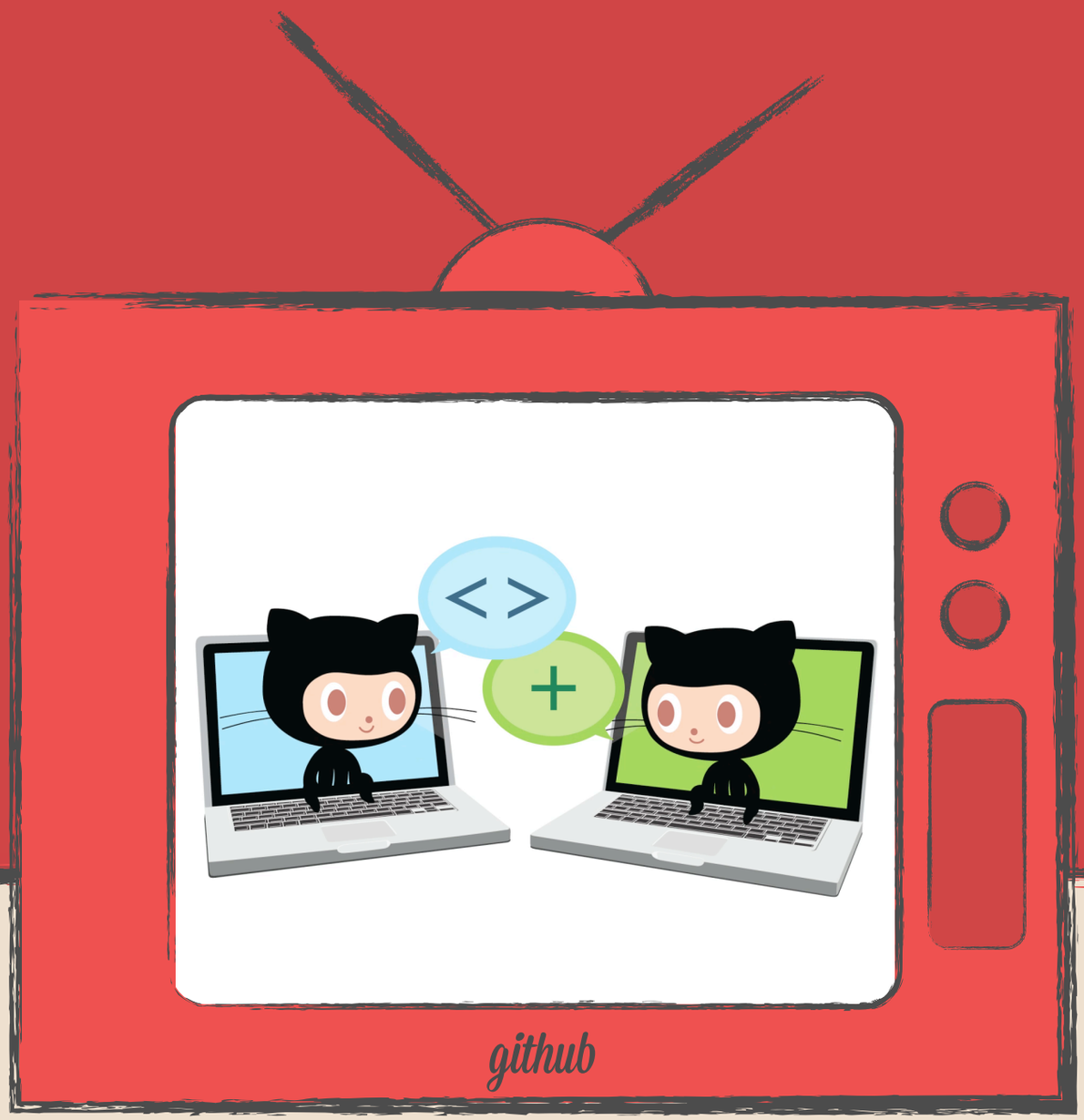
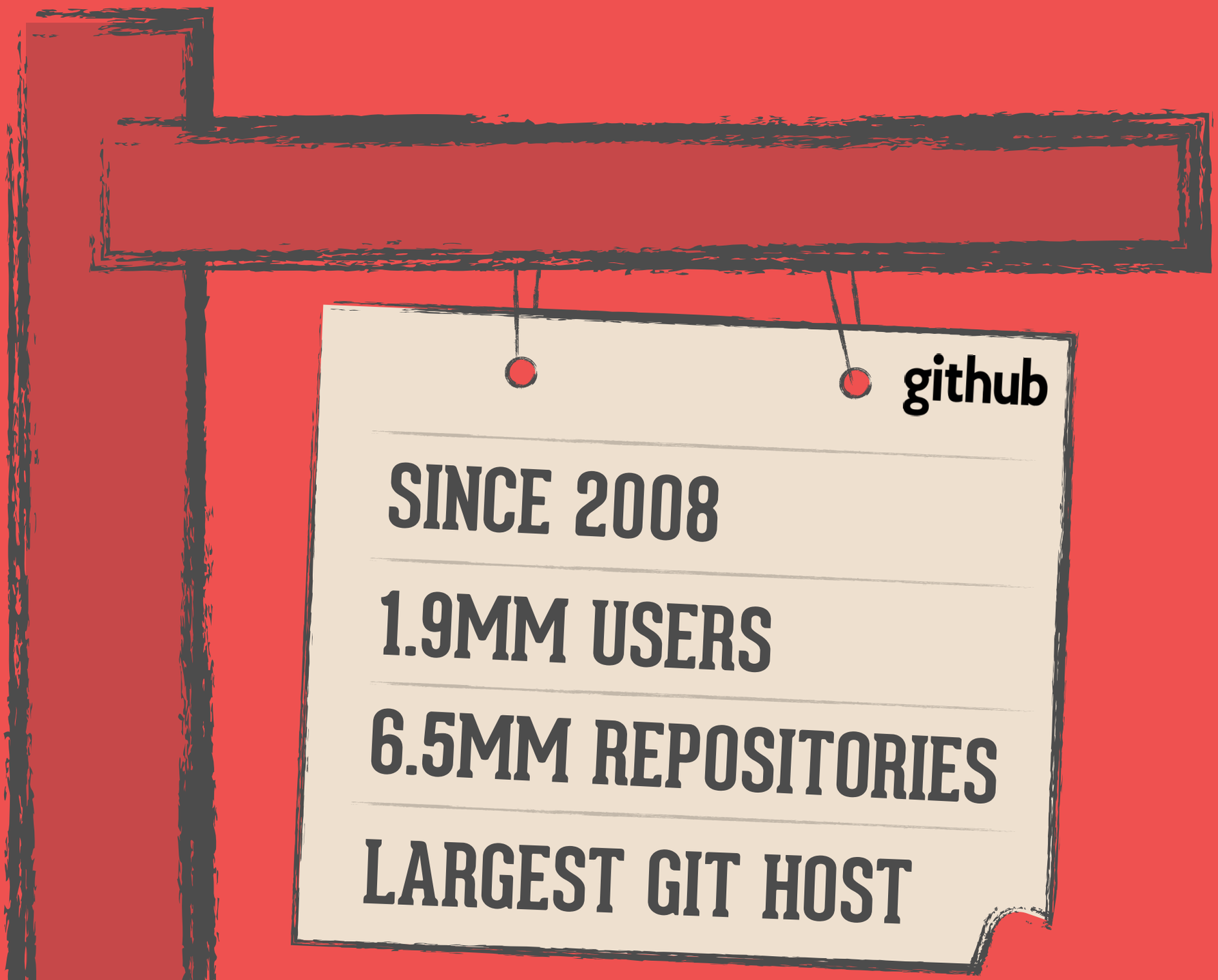
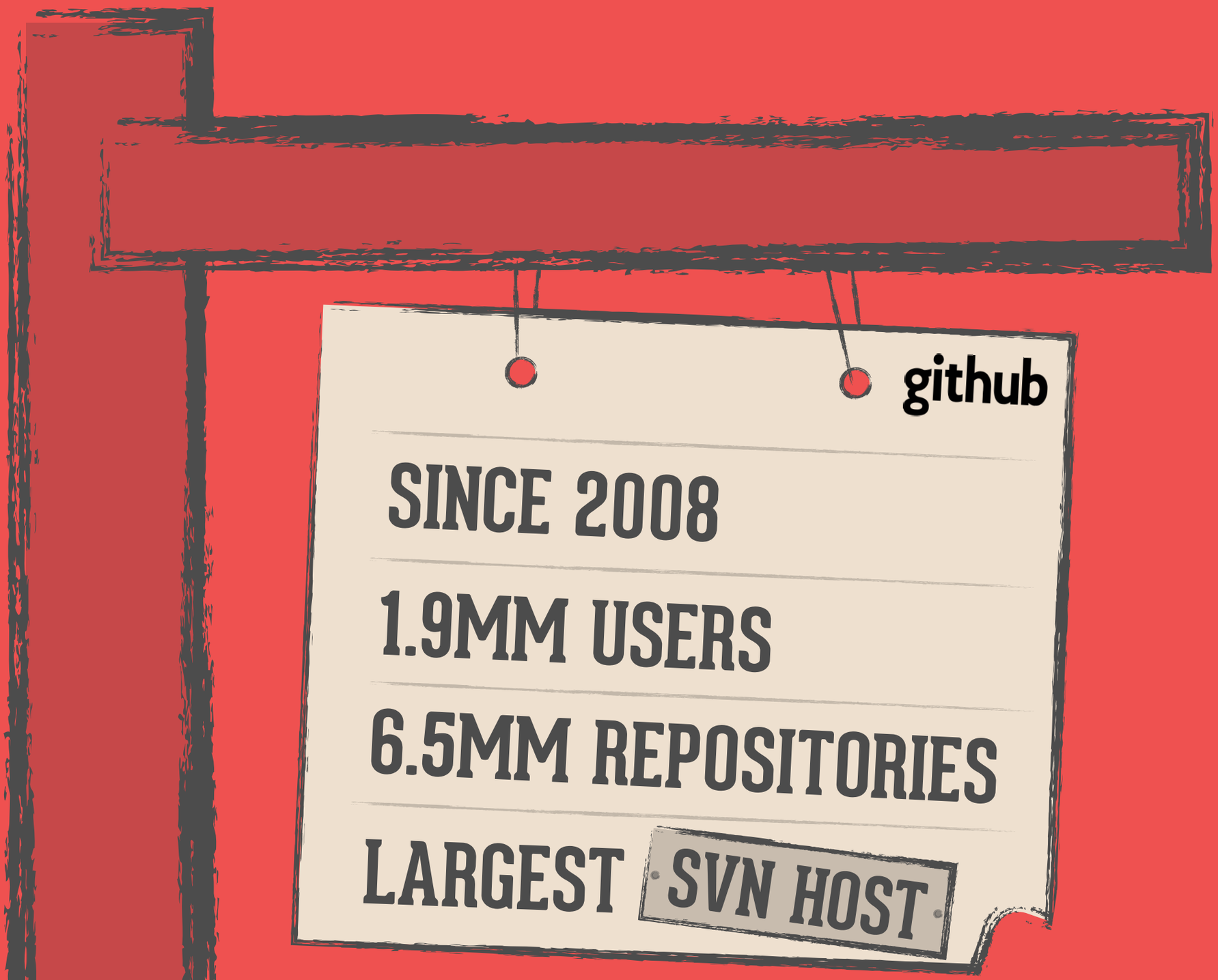
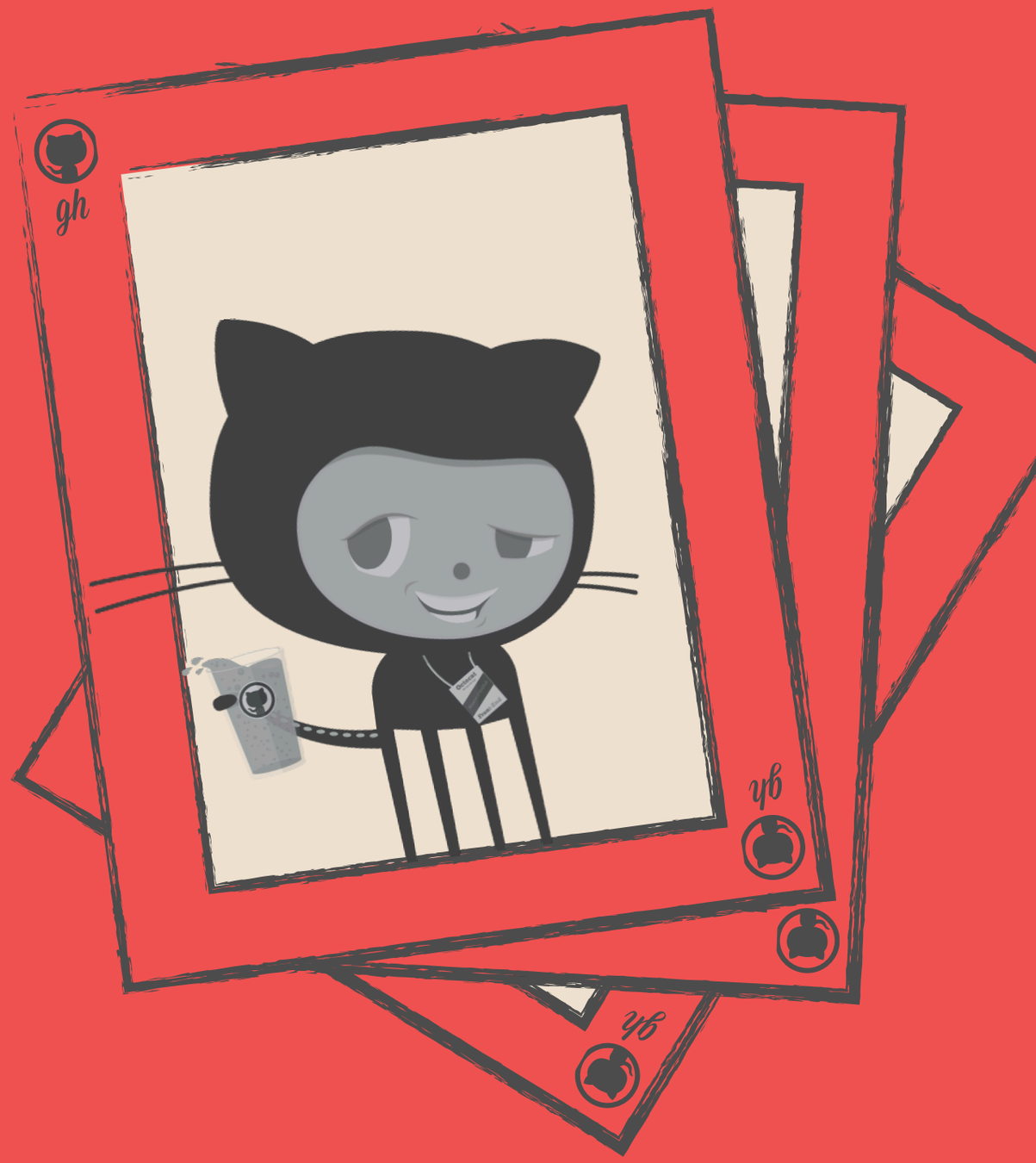
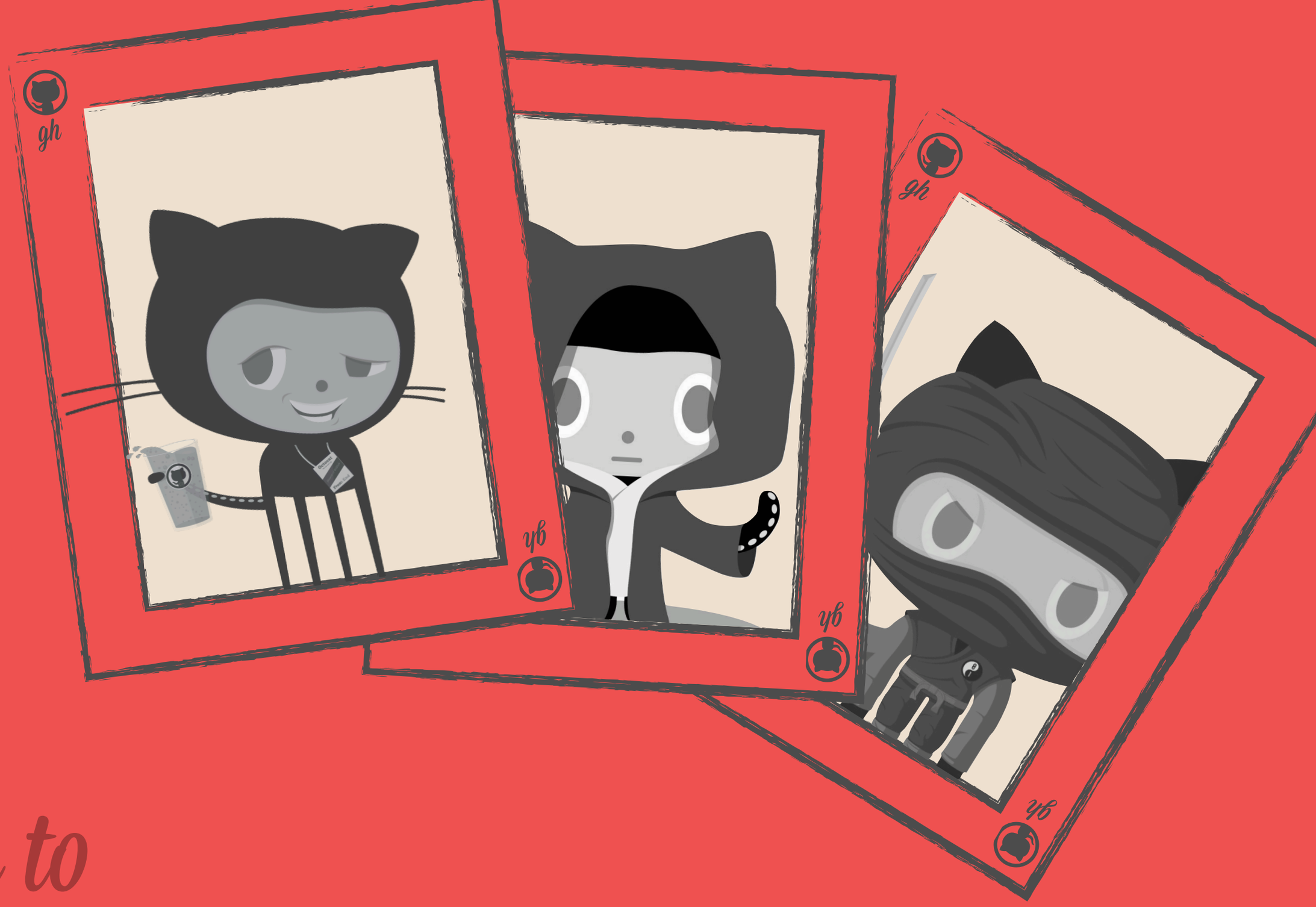
HOW *to* BUILD *a* GITHUB


github


github
SINCE 2008
1.9MM USERS
6.5MM REPOSITORIES
LARGEST GIT HOST


github
SINCE 2008
1.9MM USERS
6.5MM REPOSITORIES
LARGEST SVN HOST


gh
gh
gh



going to

SHOW YOU OUR CARDS

there is no

MAGIC BULLET

FOUR STAGES OF GROWTH

automate **EVERYTHING**

the **happiness**

LOST

@ HOLMAN

YO QUIT READING THIS SHIT

NO
FORKING

how

DID WE GIT HERE

1809:

PERL INVENTED

1814:

COMPUTERS INVENTED

1814-2004: ANARCHY AND CHAOS AND ZOMG EVERYONE'S DYING

2005:

VERSION CONTROL INVENTED

2007:

GLOBAL PEACE AND HAPPINESS ACHIEVED

github

...or something like that

TOM PRESTON-WERNER

OCTOBER 9, 2007

GRIT

git via ruby

GRIT

git via ruby

github's interface to git

object-oriented, read/write

open source


```
repo = Grit::Repo.new('/tmp/repository')
```

```
repo.commits
```

grit

shelling out to git is expensive

grit reimplements portions of git in ruby

native packfile and git object support

2x-100x speedup on low-level operations

grit

slowly reimplement grit for speed

allows for **incremental improvements**

OCTOBER 19, 2007

git

LED TO GITHUB

TODAY

22 FILESERVER PAIRS

ADDING 2TB A MONTH

23TB OF REPO DATA

THE FOUR STAGES

of

GITHUB GROWTH

GITHUB: FOUR STAGES OF GROWTH

LOCAL

NETWORKED

NET-SHARD

GITRPC

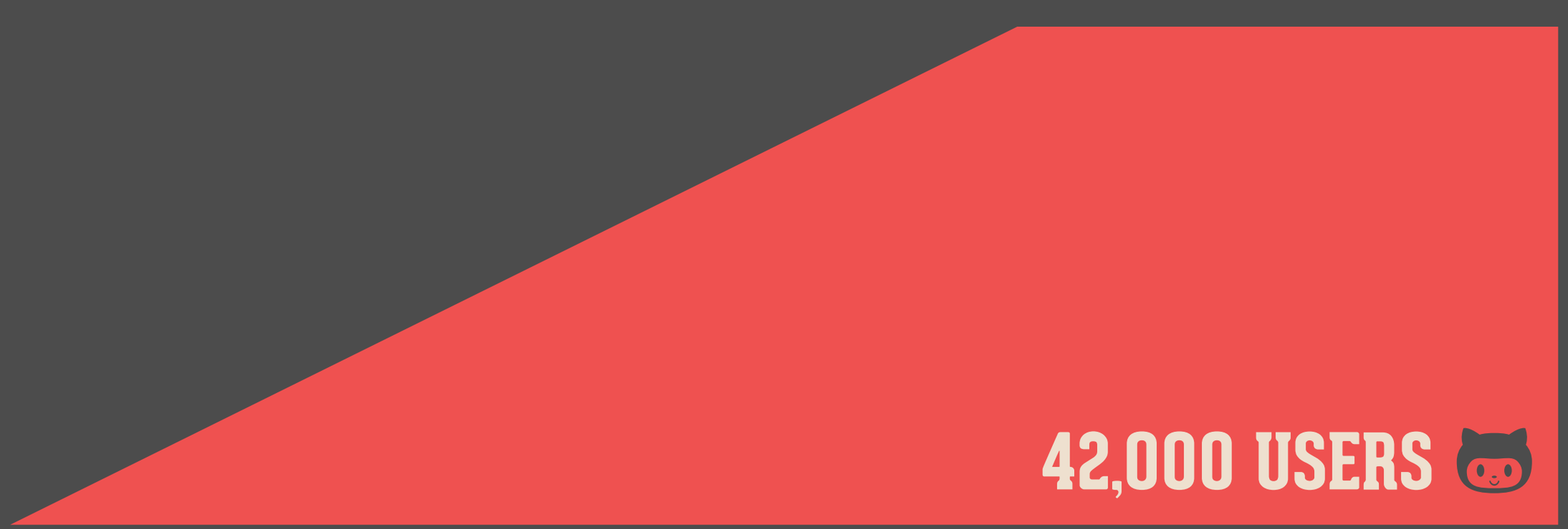
GITHUB: FOUR STAGES OF GROWTH

LOCAL	NETWORKED	NET-SHARD	GITRPC
2008	2009	2010	2012

GITHUB: FOUR STAGES OF GROWTH

LOCAL | NETWORKED | NET-SHARD | GITRPC

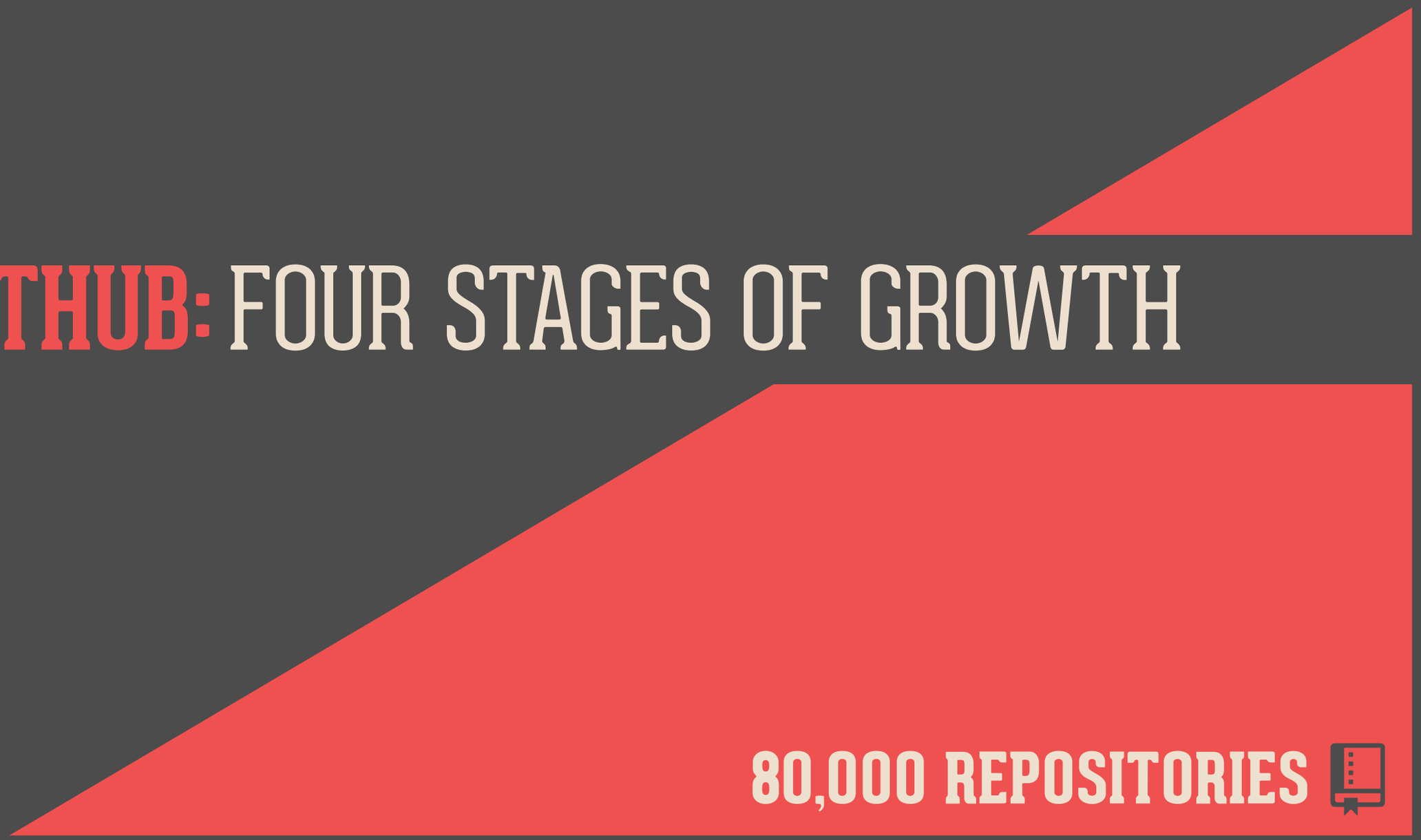
GITHUB: FOUR STAGES OF GROWTH



JAN 2008

DEC 2008

GITHUB: FOUR STAGES OF GROWTH

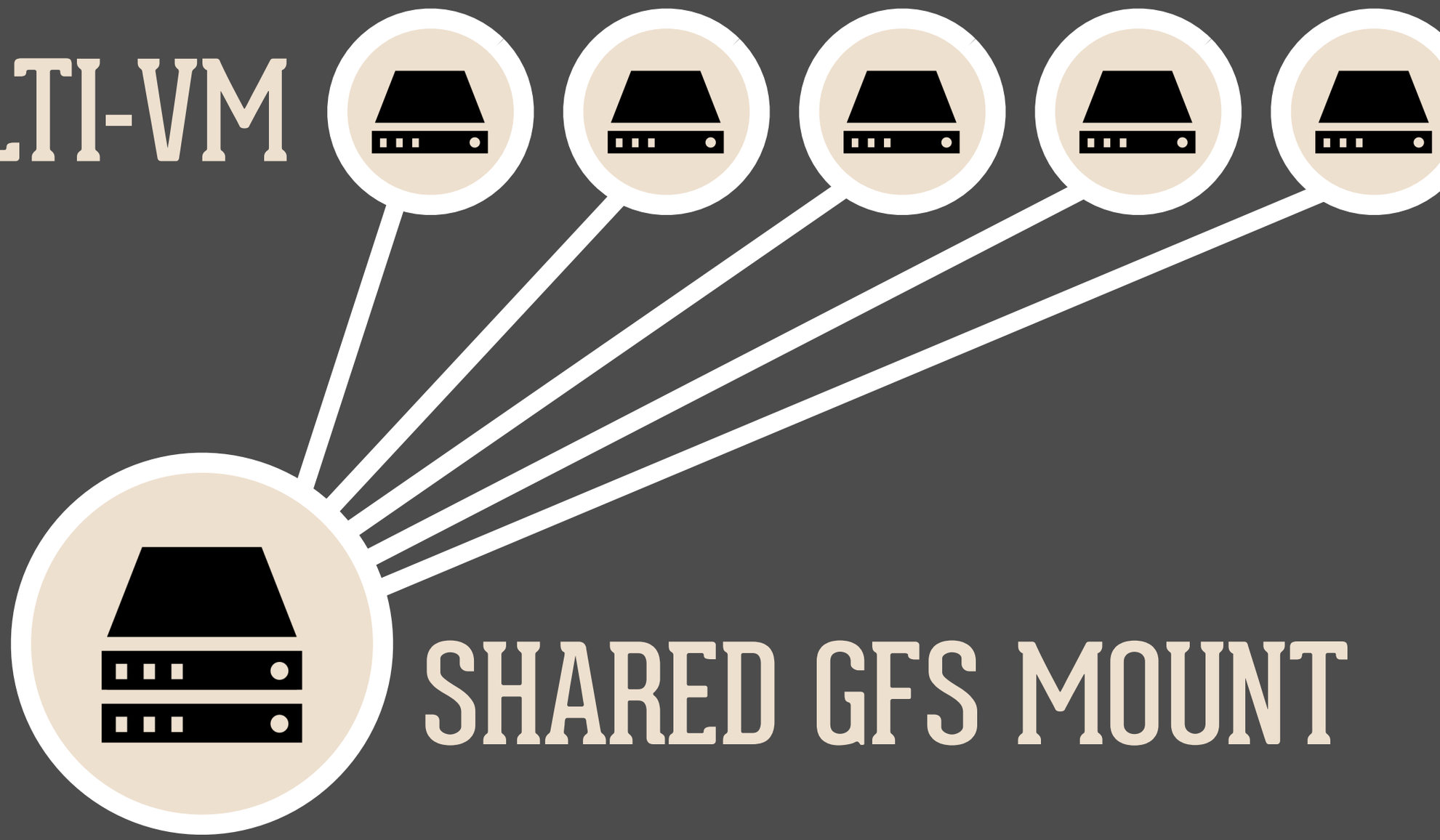


JAN 2008

DEC 2008

LOCAL

MULTI-VM



SHARED GFS MOUNT

LOCAL

MULTI-VM

WEB FRONTENDS

BACKGROUND WORKERS

LOCAL

MULTI-VM

SIMPLE ARCHITECTURE

HORIZONTALLY SCALABLE-ish

LOCAL

SHARED MOUNT ON EACH VM

ALLOWED LOCAL ACCESS VIA GRIT

SIMILAR PRODUCTION + DEVELOPMENT ACCESS

SHARED GFS MOUNT

LOCAL

SIMPLE APPROACH, COMMON GIT INTERFACE, QUICK TO BUILD AND SHIP

GITHUB: FOUR STAGES OF GROWTH

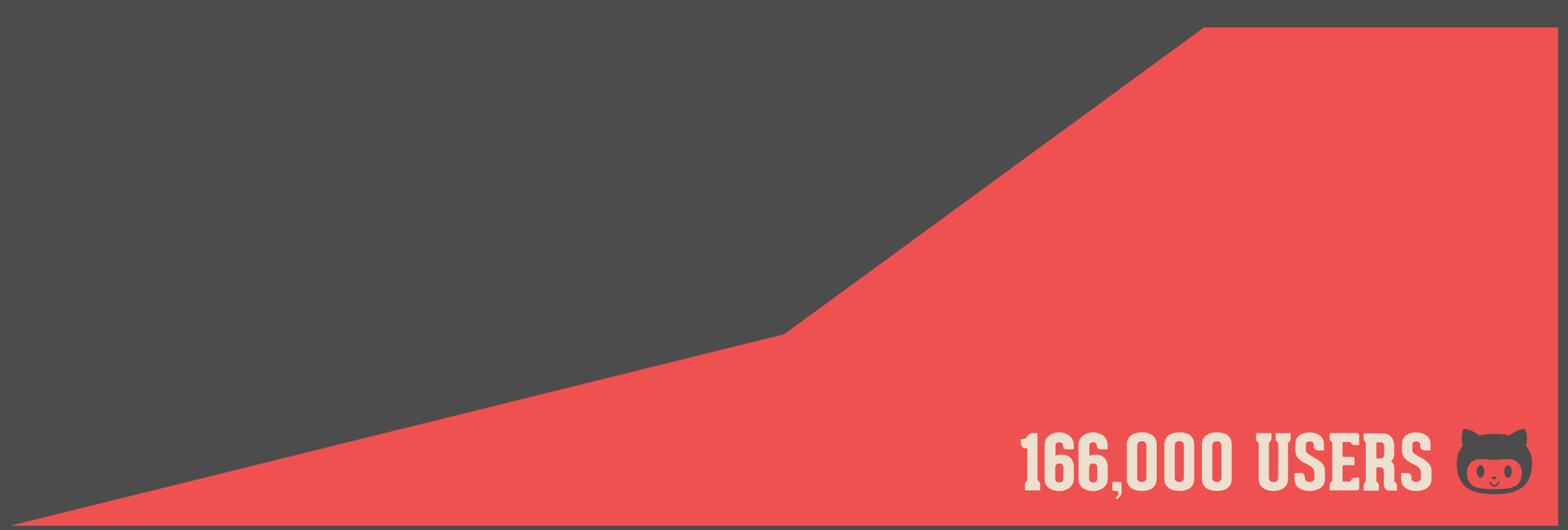
LOCAL

NETWORKED

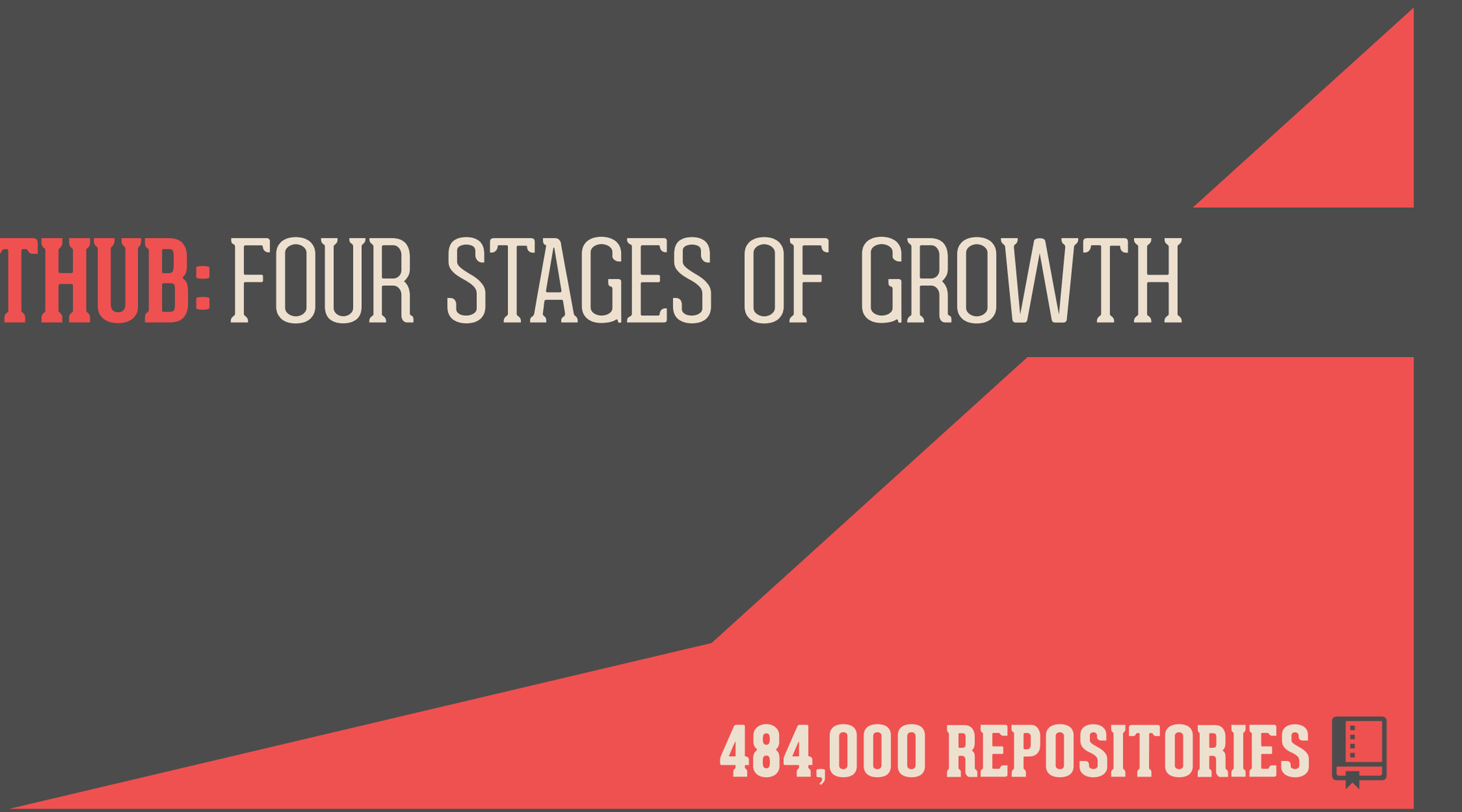
NET-SHARD

GITRPC

GITHUB: FOUR STAGES OF GROWTH



GITHUB: FOUR STAGES OF GROWTH



the problem:

GFS  is slow

performance degraded as repos added

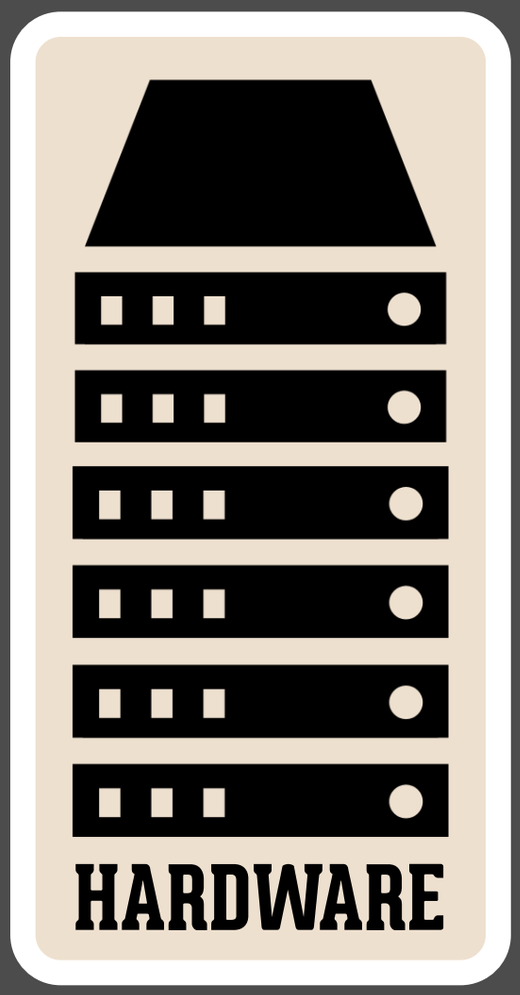
the problem:

we're  i/o-bound

read/write to disk needs to be **fast**

NETWORKED

THE PLAN



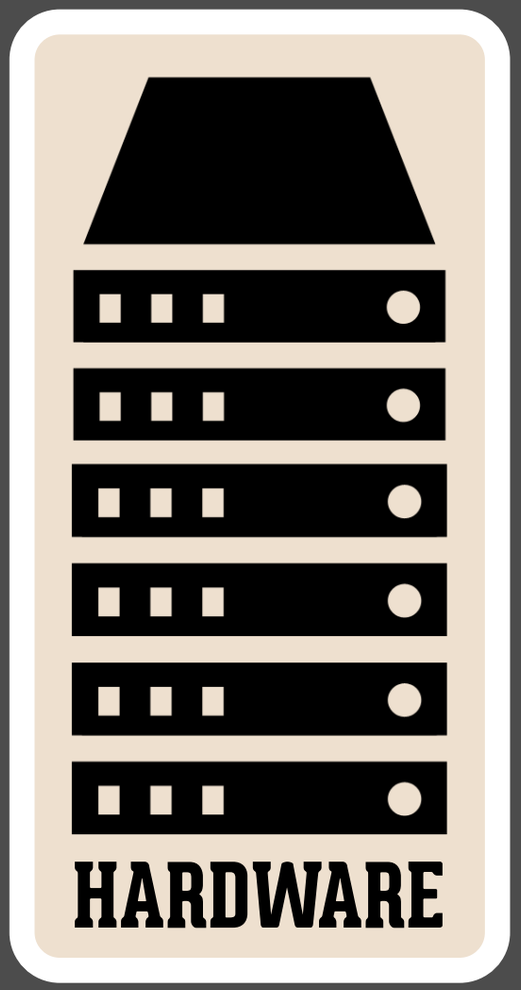
MOVE DATACENTERS

NETWORKED

bare metal servers

16 machines

6x RAM



machine roles

solid datacenter

got dat cloud

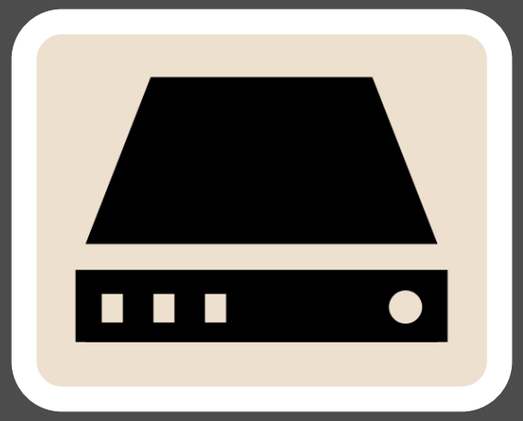
NETWORKED

LAUNCH:

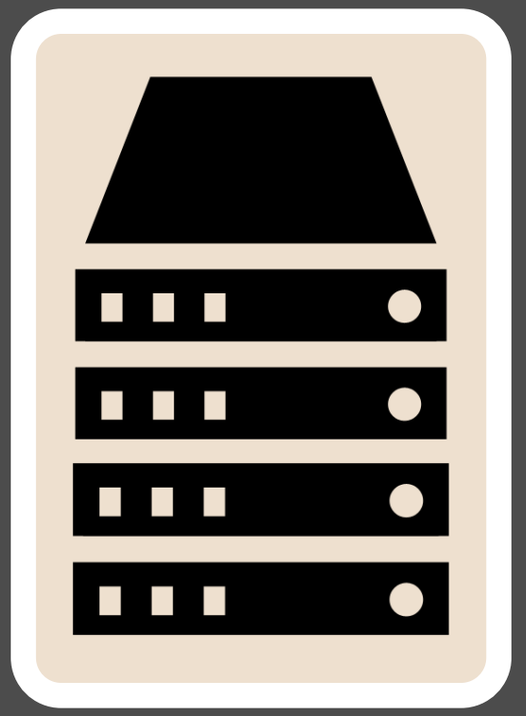
SERVER PAIRS

FRONTENDS FILESERVERS AUX DB

NETWORKED



GRIT IS LOCAL



NEEDS TO BE NETWORKED

NETWORKED

smoke service is run on each fs;
facilitates disk access

chimney routes the smoke,
stores routing table in **redis**

stub local **grit** calls, retain API
usage, but send over network

NETWORKED

server pairs offer **failover** via DRBD

real servers, real big **RAM allocations**

NETWORKED

LATENCY

networked routing adds **2-10ms** per request

optimize for the roundtrip

smoke contains smarter server-side logic

NETWORKED

LATENCY

smoke has custom git extension commands

git-distinct-commits

returns commits only contained on a given branch

calls to git-show-refs and git-rev-list

run all calls server-side in one roundtrip

NETWORKED

HORIZONTALLY-SCALABLE, LATENCY-CONSIDERATE, API-COMPATIBLE WITH GRIT

GITHUB: FOUR STAGES OF GROWTH

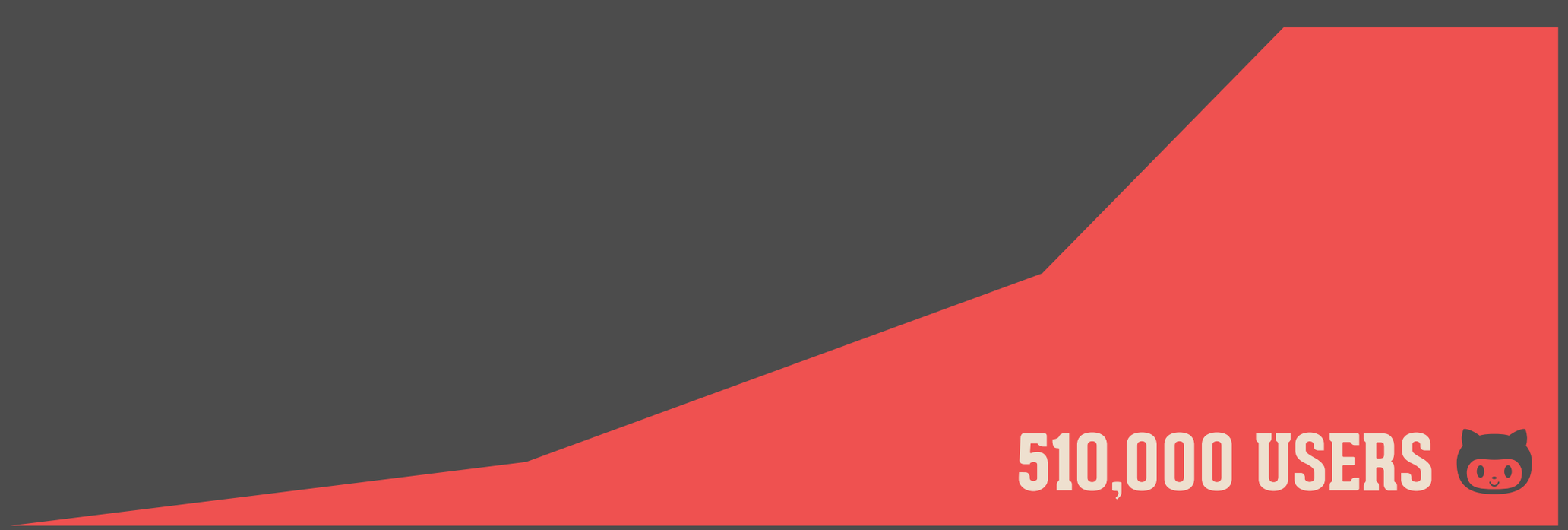
LOCAL

NETWORKED

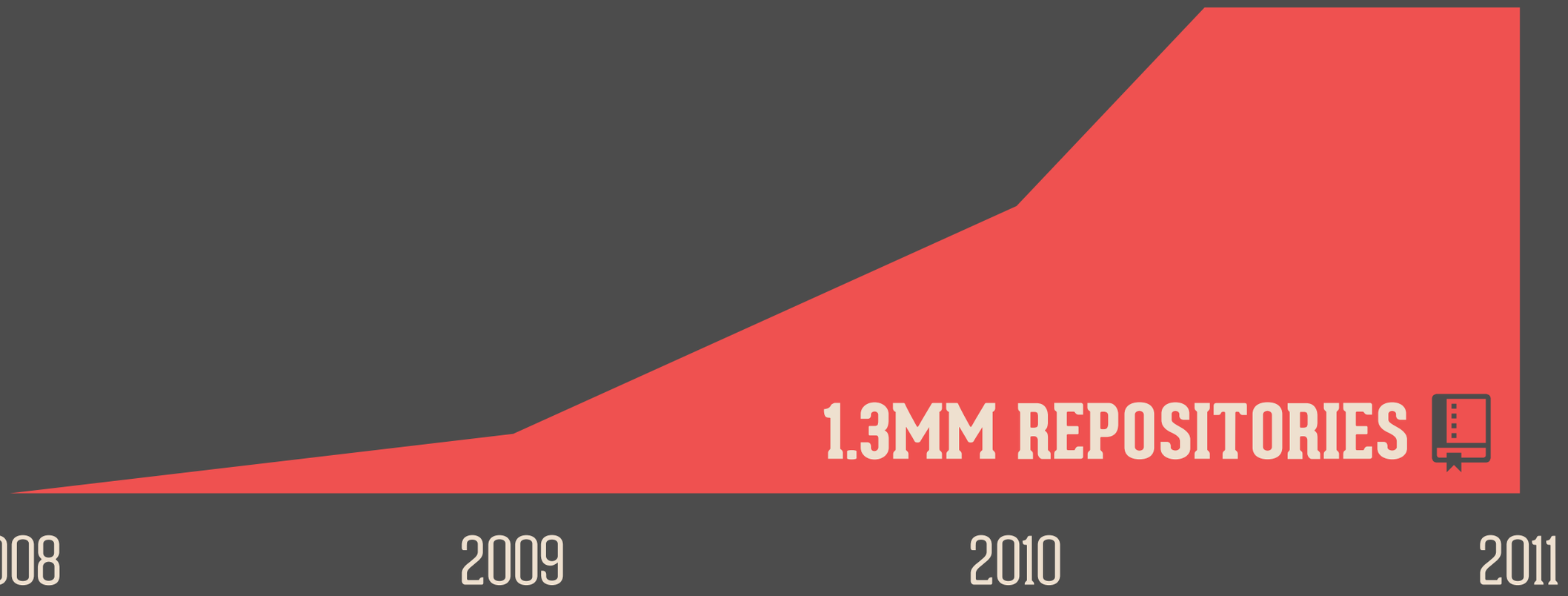
NET-SHARD

GITRPC

GITHUB: FOUR STAGES OF GROWTH



GITHUB: FOUR STAGES OF GROWTH

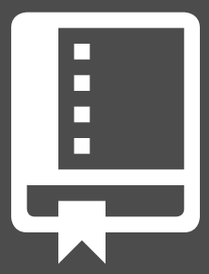


the problem:

data  duplication

each fork is a full project history

data duplication



i create a repo

fs5:/data/repositories/6/nw/6b/de/92/1/1.git

you fork my repo

fs7:/data/repositories/4/na/3b/dr/72/2/2.git

data duplication

1,000 commits

10MB

1,001 commits

10MB

20MB total disk

data duplication

1,000 commits

10MB

1 commit

1KB

GOAL:

10MB total disk

data duplication

75 MB repo

x 3.5k forks

~250 GB

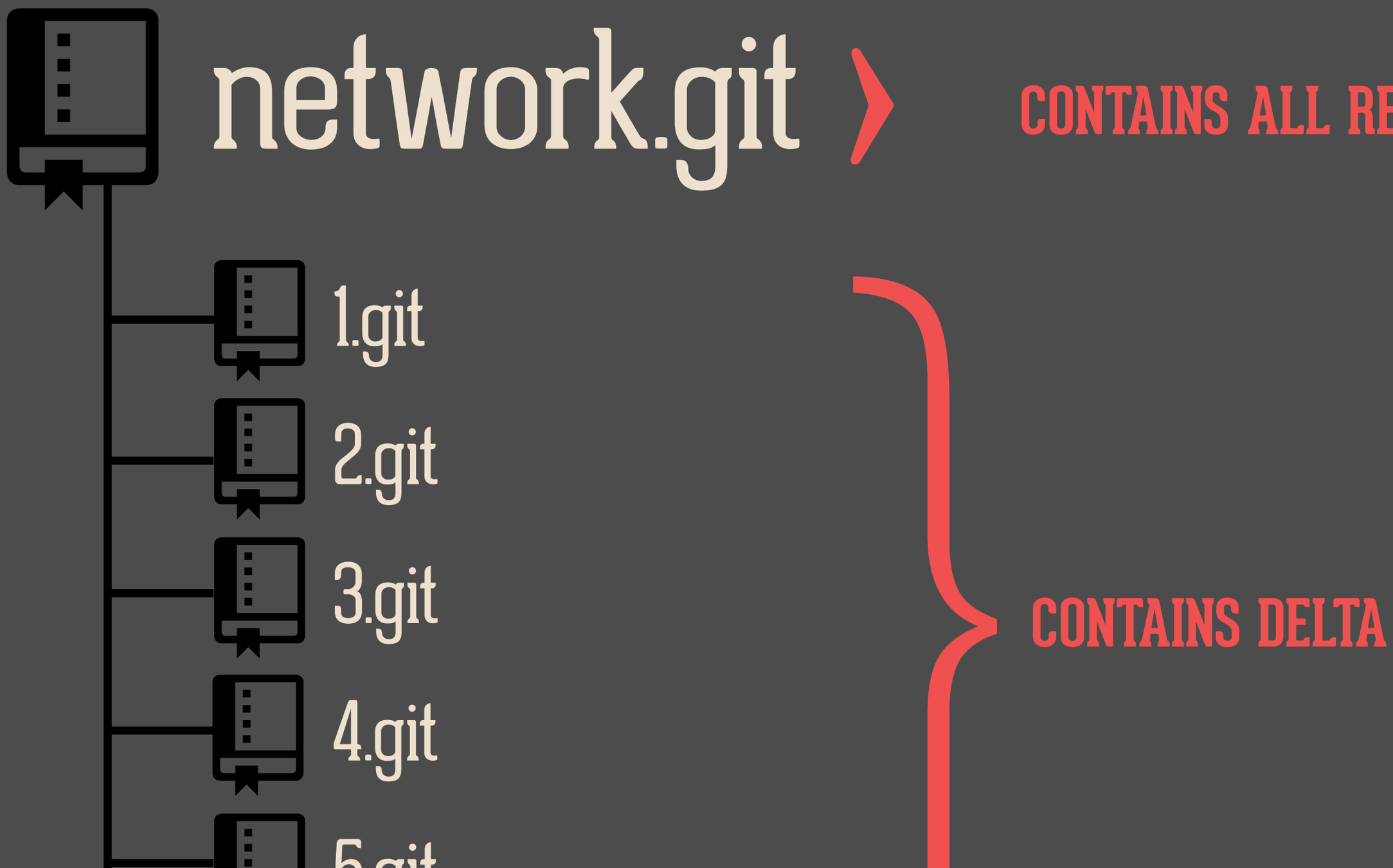
x 2 fs pairs + offsite backups

NET-SHARD

shard by repository **network**

(“forks”)

NET-SHARD


network.git
CONTAINS ALL REFS
1.git
2.git
3.git
4.git
5.git
CONTAINS DELTA

NET-SHARD

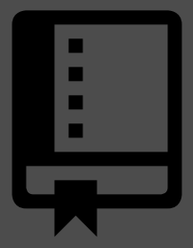
network.git

GIT ALTERNATES

store git object data externally to repository

we fetch refs into your fork, transparently

NET-SHARD



network.git

PRIVACY

potential leaking of refs cross-network

net-shard enabled on **all-public** and **all-private** repository networks only

NET-SHARD

network.git

DISK

halves disk usage

increase disk and kernel cache hits

NET-SHARD

network.git

MIGRATION

gradually transitioned repos to network.git

effectively feature-flagged by repo

NET-SHARD

SAVE DISK, IMPROVE PERFORMANCE

GITHUB: FOUR STAGES OF GROWTH

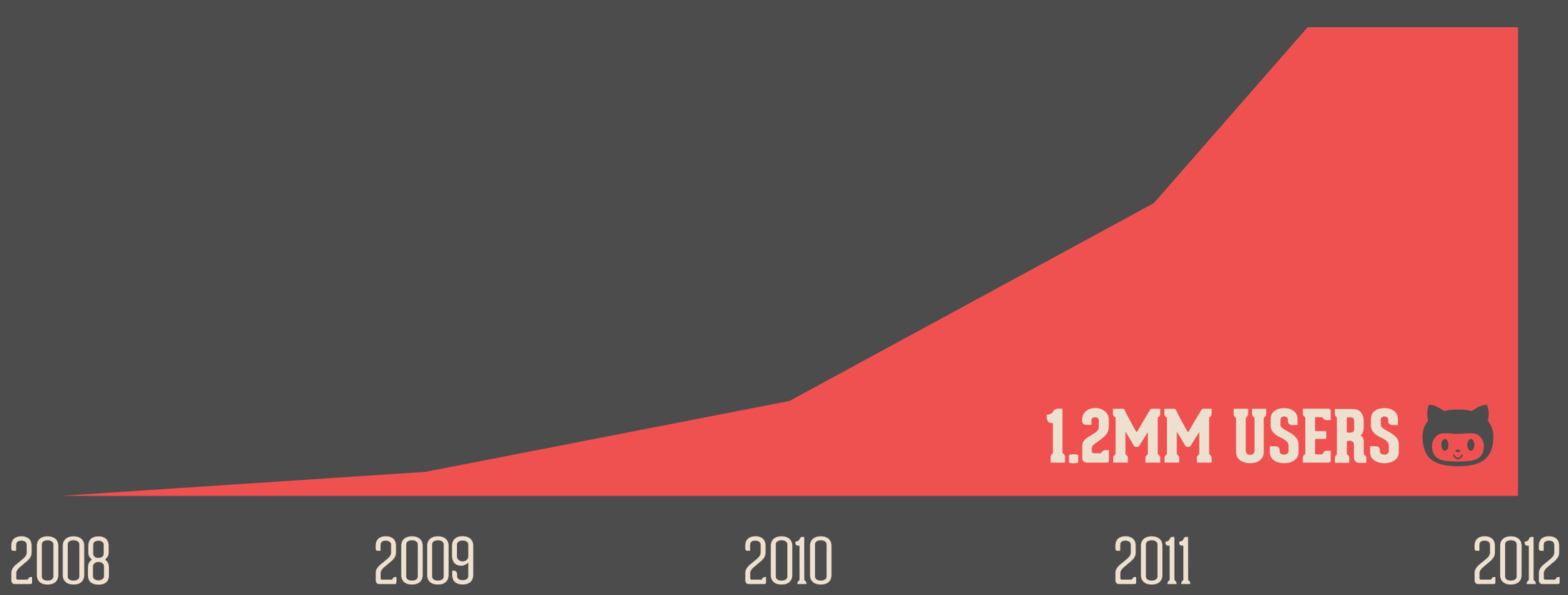
LOCAL

NETWORKED

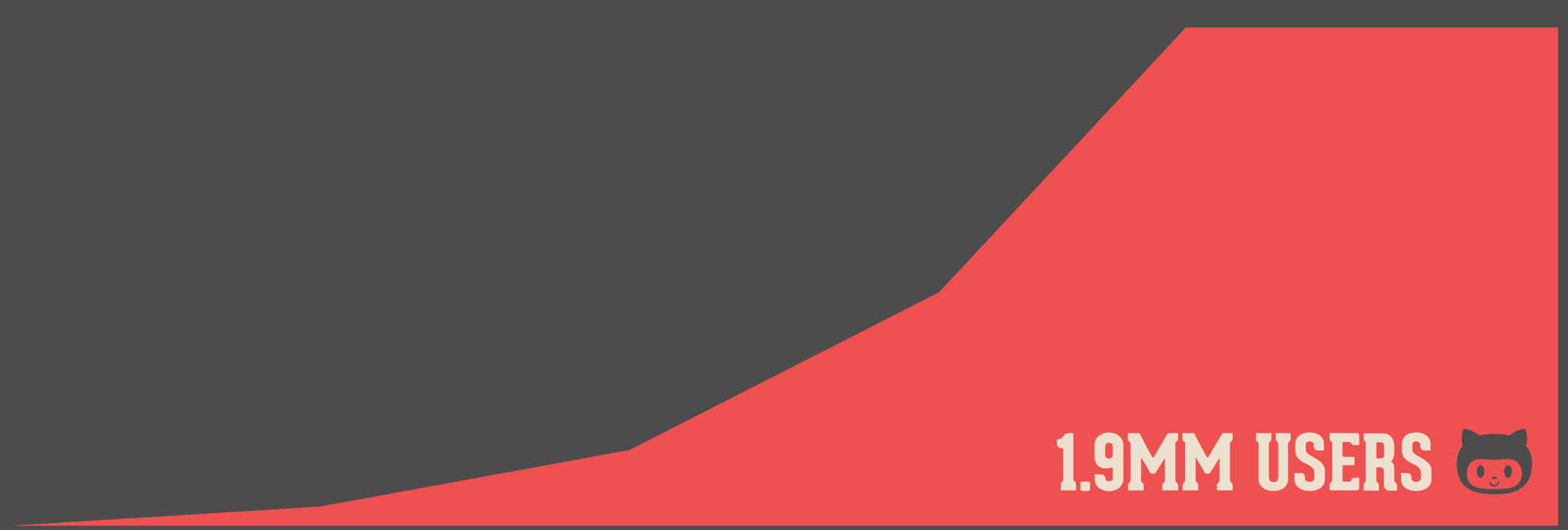
NET-SHARD

GITRPC

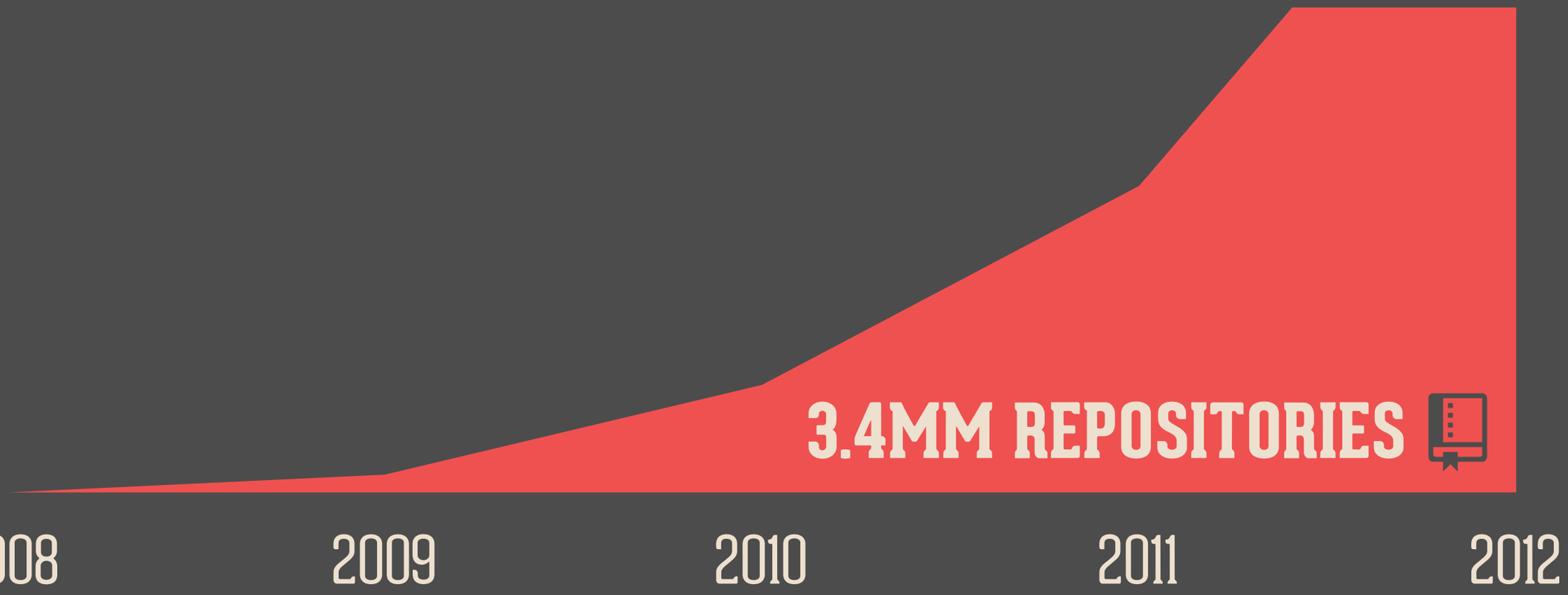
GITHUB: FOUR STAGES OF GROWTH



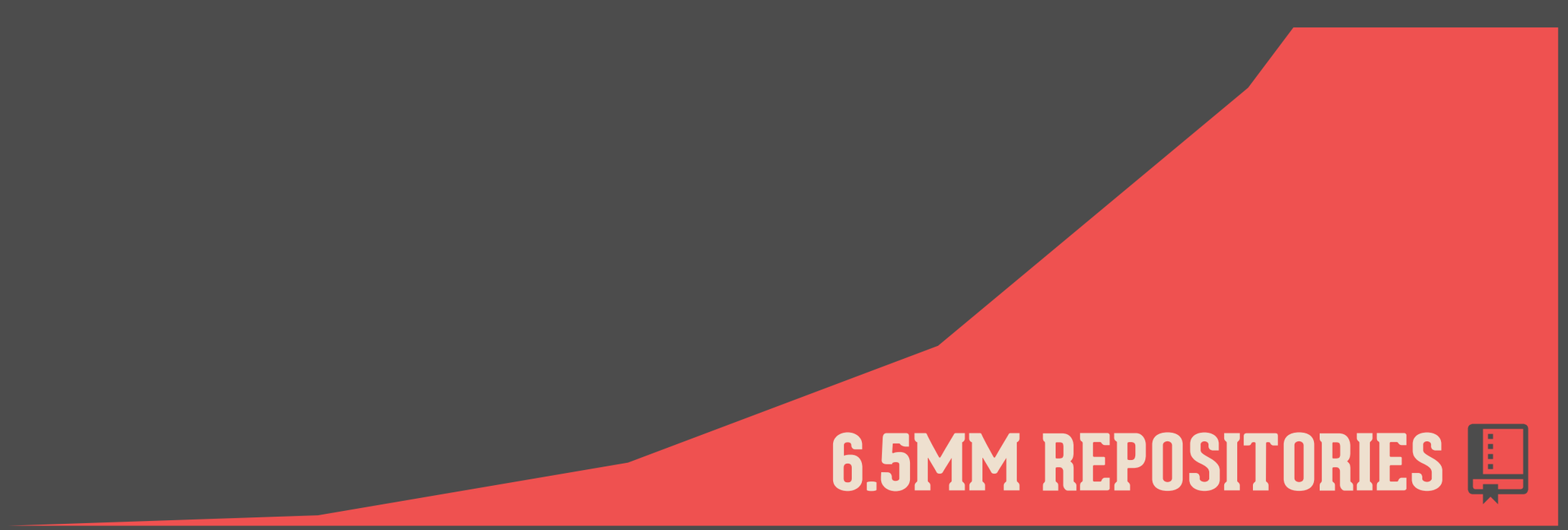
GITHUB: FOUR STAGES OF GROWTH



GITHUB: FOUR STAGES OF GROWTH



GITHUB: FOUR STAGES OF GROWTH



the problem:

GRIT

git via ruby

the problem:

local, ruby-based grit ended up
in a high-traffic distributed system

the problem:

inelegant code spread out everywhere

GITRPC

GitRPC

network-oriented library for git access

GITRPC

open source

github-sponsored project

fastest git implementation (C)

bindings for all major languages

used in our mac, windows clients

GITRPC

gitrpc (RUBY)

rugged (RUBY)

libgit2 (C)

GITRPC

LATENCY

like **smoke**, gitrpc aims to

reduce latency by reducing roundtrips

GITRPC

CACHING

operations cached on library level

yank out tons of app-level cache logic

GITRPC

MIGRATION

the move to gitrpc started this summer and will take months

gradually replace smoke and grit;
avoids a risky deploy

GITRPC

FAST AND STABLE NETWORKED GIT ACCESS

GITHUB: FOUR STAGES OF GROWTH

LOCAL

NETWORKED

NET-SHARD

GITRPC

identify

WHAT'S BROKEN

small

CHANGES, FAST DEVELOPMENT

real CODE BEATS IMAGINARY CODE

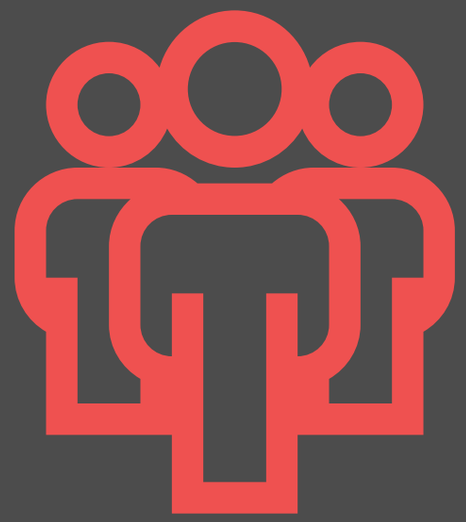
automate
automate
automate
automate
automate
automate

AUTOMATE EVERYTHING

automate
automate
automate
automate
automate
automate

SOFTWARE DEVELOPMENT

mr. manager



LOL DEVELOPERS

mr. manager

DEADLINES

MEETINGS

PRIORITIES

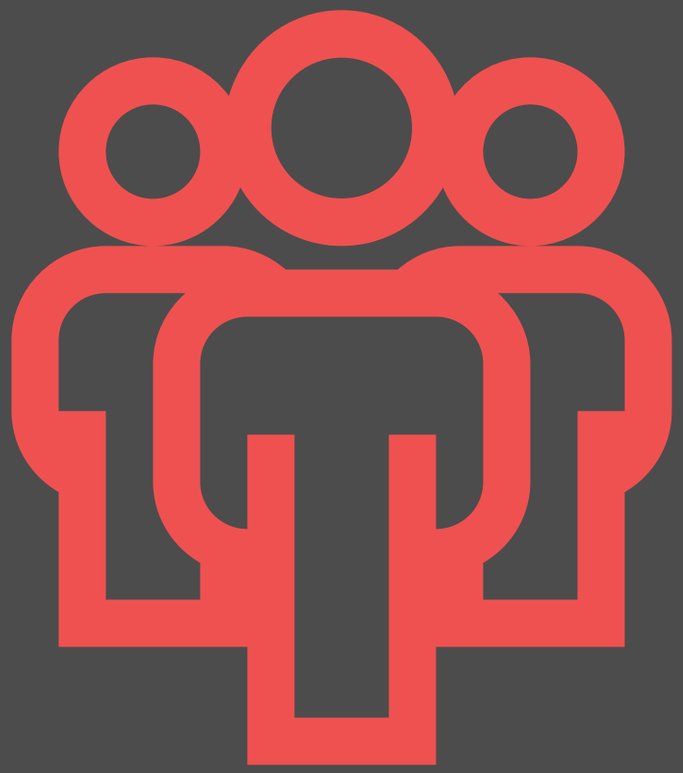
ESTIMATES

~~DEADLINES~~

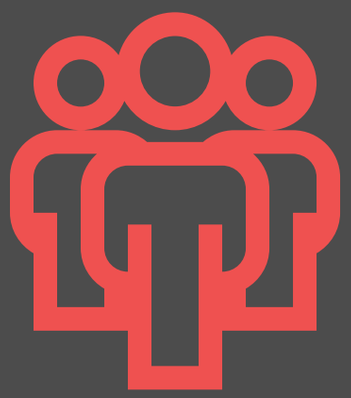
~~MEETINGS~~

~~PRIORITIES~~

~~ESTIMATES~~



EVERYONE *is* A MANAGER



AUTOMATE AWAY PAIN

DEVELOPMENT

DEPLOYMENT

RECOVERY

automate

DEVELOPMENT

DEVELOPMENT

RUN THIS IN EACH PROJECT:

```
> ./do-work
```

...AND YOU'RE DONE!

loljk

DEVELOPMENT

YOU CAN AUTOMATE THE PAIN OF DEVELOPMENT

DEVELOPMENT

the SETUP

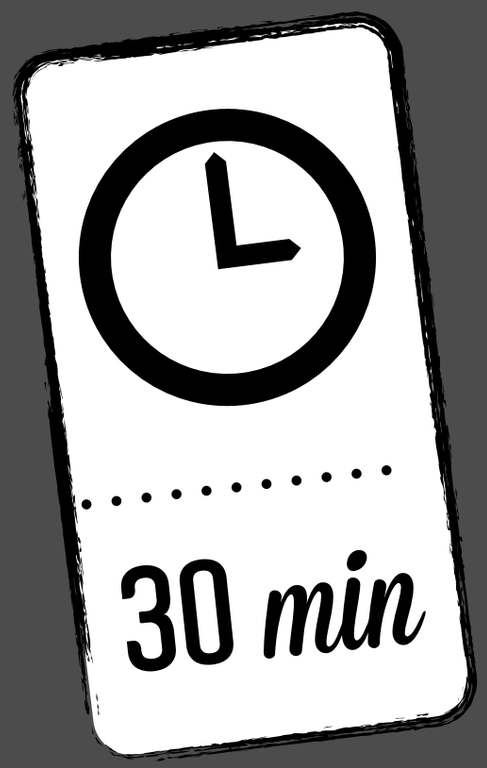
the SETUP

DEVELOPMENT

ONE-LINER INSTALLS ALL GITHUB DEVELOPMENT DEPENDENCIES

the SETUP

DEVELOPMENT



CLEAN MACHINE TO FULL DEVELOPMENT ENVIRONMENT

the SETUP

DEVELOPMENT

NEW EMPLOYEES

SHIP

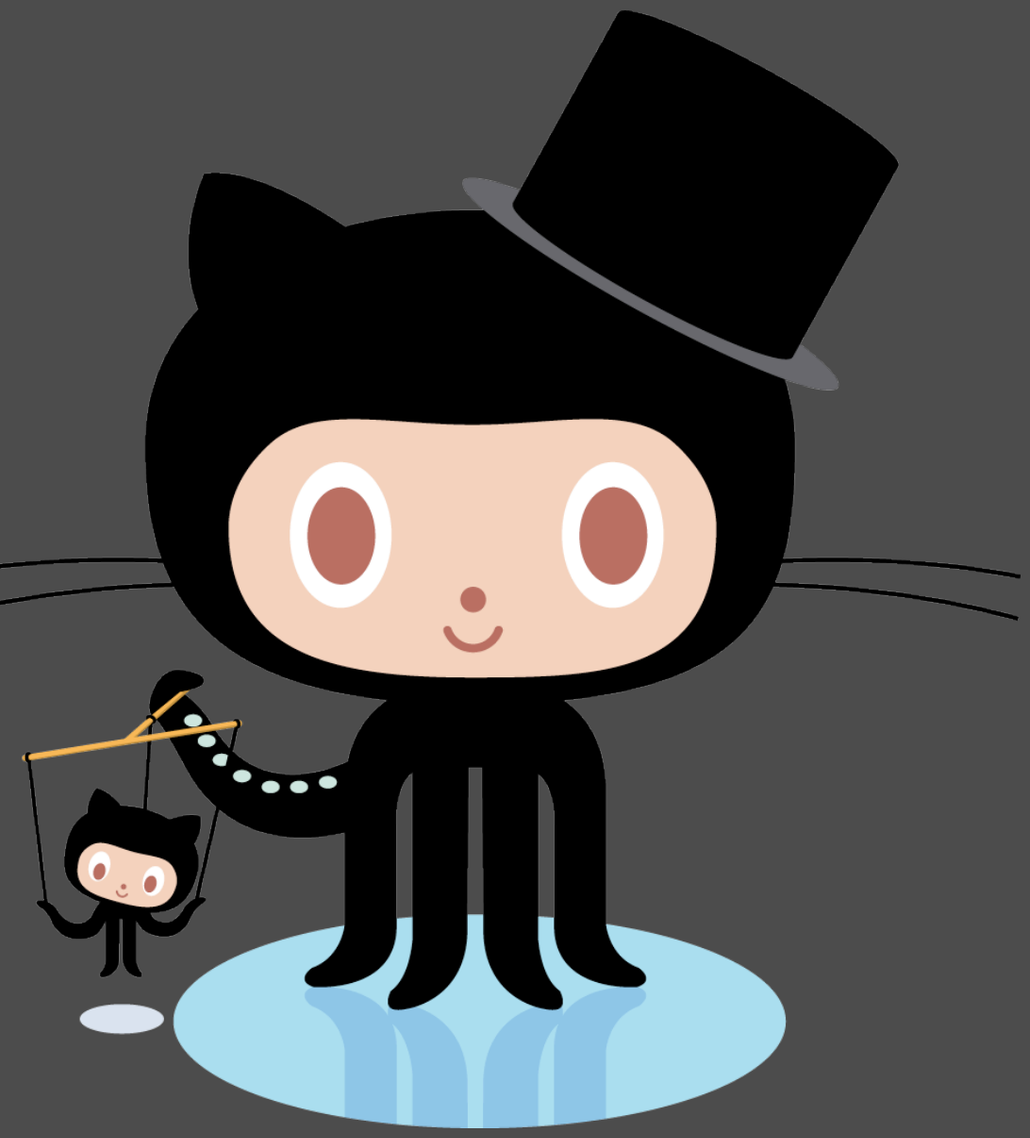
THEIR FIRST WEEK

the SETUP

DEVELOPMENT

PUPPET

HANDLES ALL DEPENDENCIES



automate

DEPLOYMENT

DEPLOYMENT

REAL BROGRAMMERS
DEPLOY WITH
NO FEAR

SO FUCK THAT

DEPLOYS SHOULD BE CAUTIOUS, COMMONPLACE, AND AUTOMATED

DEPLOYMENT

GITHUB DEPLOYS 20-40 TIMES A DAY

DEPLOYMENT

PUSH BRANCH

USUALLY OPEN A PULL REQUEST

HUBOT RUNS TESTS

IN ABOUT 200 SECONDS

DEPLOY BRANCH

EVERYWHERE · MACHINE CLASS · SPECIFIC SERVERS

DEPLOYMENT

DEPLOY LOCKING

CAN'T DEPLOY IF A BRANCH IS DEPLOYED

AUTODEPLOYS

PUSHED TO MASTER WITH GREEN TESTS? DEPLOY.

DEPLOYMENT

STAFF-ONLY FEATURE FLAGS

LIMITS EXPOSURE · REAL-WORLD · AVOIDS MERGES

automate

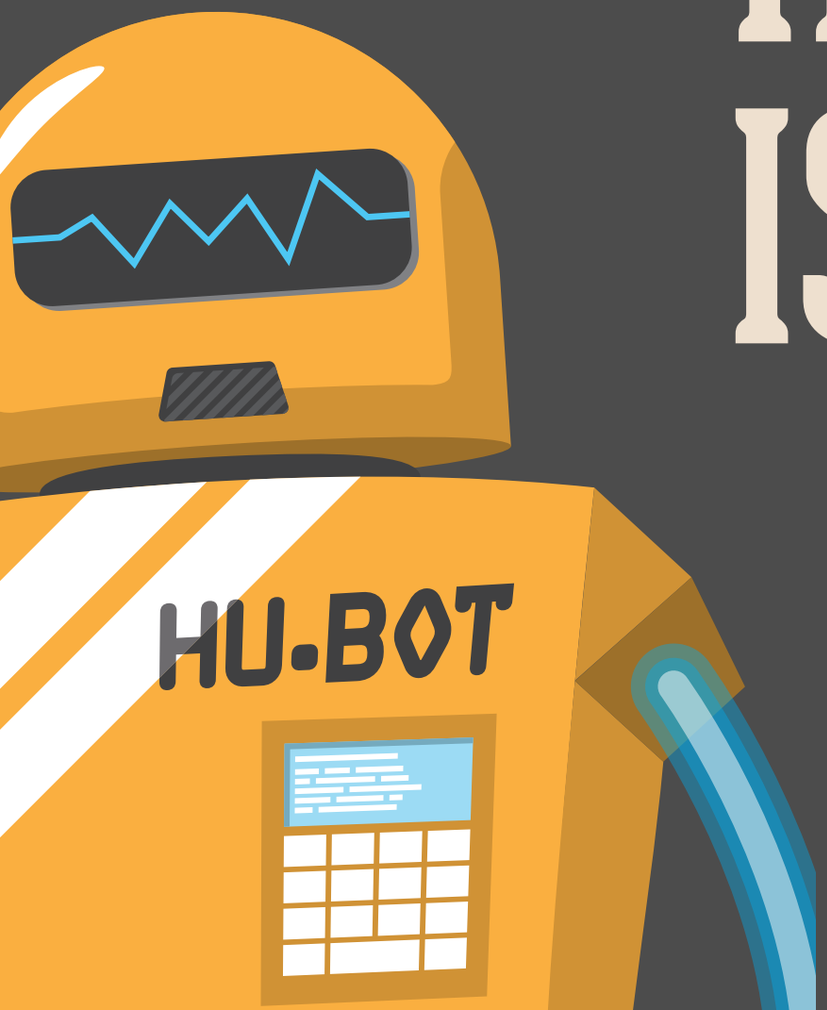
RECOVERY

RECOVERY

SOMETHING WILL ALWAYS BREAK

RECOVERY

HUBOT IS A SYSADMIN



RECOVERY

HUBOT LOAD
SERVER LOAD

HUBOT QUERIES
RUNNING DB QUERIES

HUBOT CONNS
ALL OPEN CONNECTIONS

RECOVERY

HUBOT RESTORE <REPO>

RESTORE A REPO FROM BACKUPS

HUBOT PUSH-LOG <REPO>

SEE RECENT PUSH LOGS TO A REPO

HUBOT GH-EACH <HOST> <COMMAND>

RUN COMMAND ON SPECIFIC HOSTS

RECOVERY

HIGH-LEVEL OVERVIEW IN MINUTES

SPEND MORE TIME FIXING AND LESS TIME INVESTIGATING

the

happiness

5

YEARS

ZERO

EMPLOYEES HAVE QUIT

108

EMPLOYEES

LEAVE	HIRE	RAMP-UP
2 WEEKS	1-2 MONTHS	1-3 MONTHS

LOSING AN EMPLOYEE CAN SET YOU BACK **HALF A YEAR**

remove

ANY REASON TO

LEAVE

TEST-FIRST ✓

BDD ✓

EMACS X
[just kidding]

TDD ✓

NONE OF THESE ✓

DESIGN-FIRST ✓

PAIR ✓
PROGRAMMING

WE CARE ABOUT THE WORK YOU DO, NOT ABOUT HOW YOU DO IT


LOCATION
HOURS
DIRECTION

HOURS

DIRECTION

LOCATION

2/3 GITHUB EMPLOYEES WORK REMOTELY

HOURS

DIRECTION

LOCATION

FAMILY RELOCATION, TRAVEL FREEDOM

LOCATION

CHOOSE YOUR SCHEDULE

HOURS

DIRECTION

CHOOSE YOUR VACATIONS

FRESH, CREATIVE EMPLOYEES

LOCATION

HOURS

DIRECTION

YOU

HACK ON THINGS
THAT INTEREST YOU

REDUCES BURNOUT

BE *flexible*

TOWARDS WORK/LIFE

github

basically,

MOVE FAST =
SMALL CHANGES

basically,

BE STABLE =

DEPLOY CONSTANTLY

basically,

HAPPY COMPANY = HAPPY EMPLOYEES

thanks

NO
FORKING

LOST

@HOLMAN

ZACHHOLMAN.COM/TALKS

YO QUIT READING THIS SHIT