Functional Infrastructures

Toni Batchelli, @disclojure

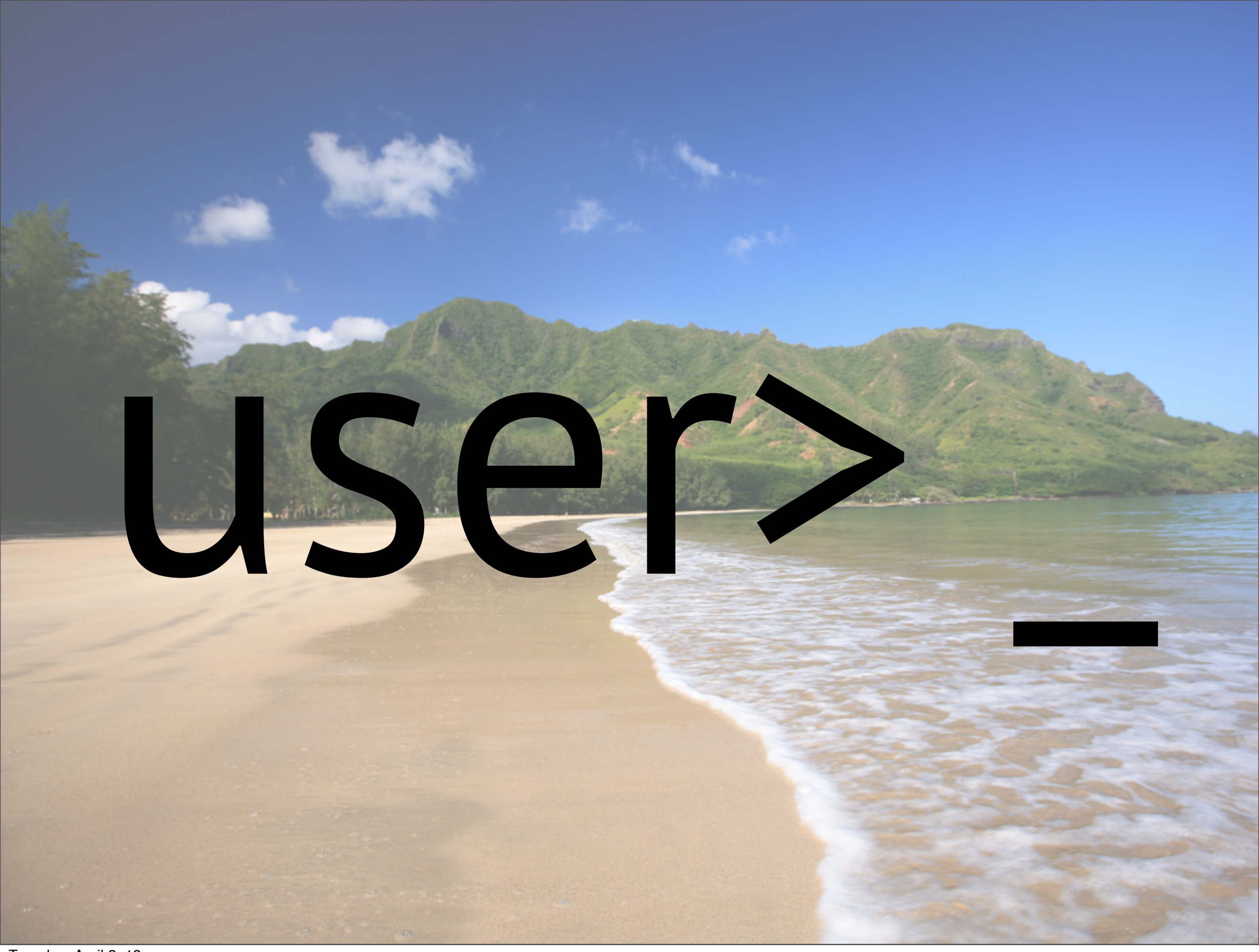
PalletOps

Clojure/West 2013

or...

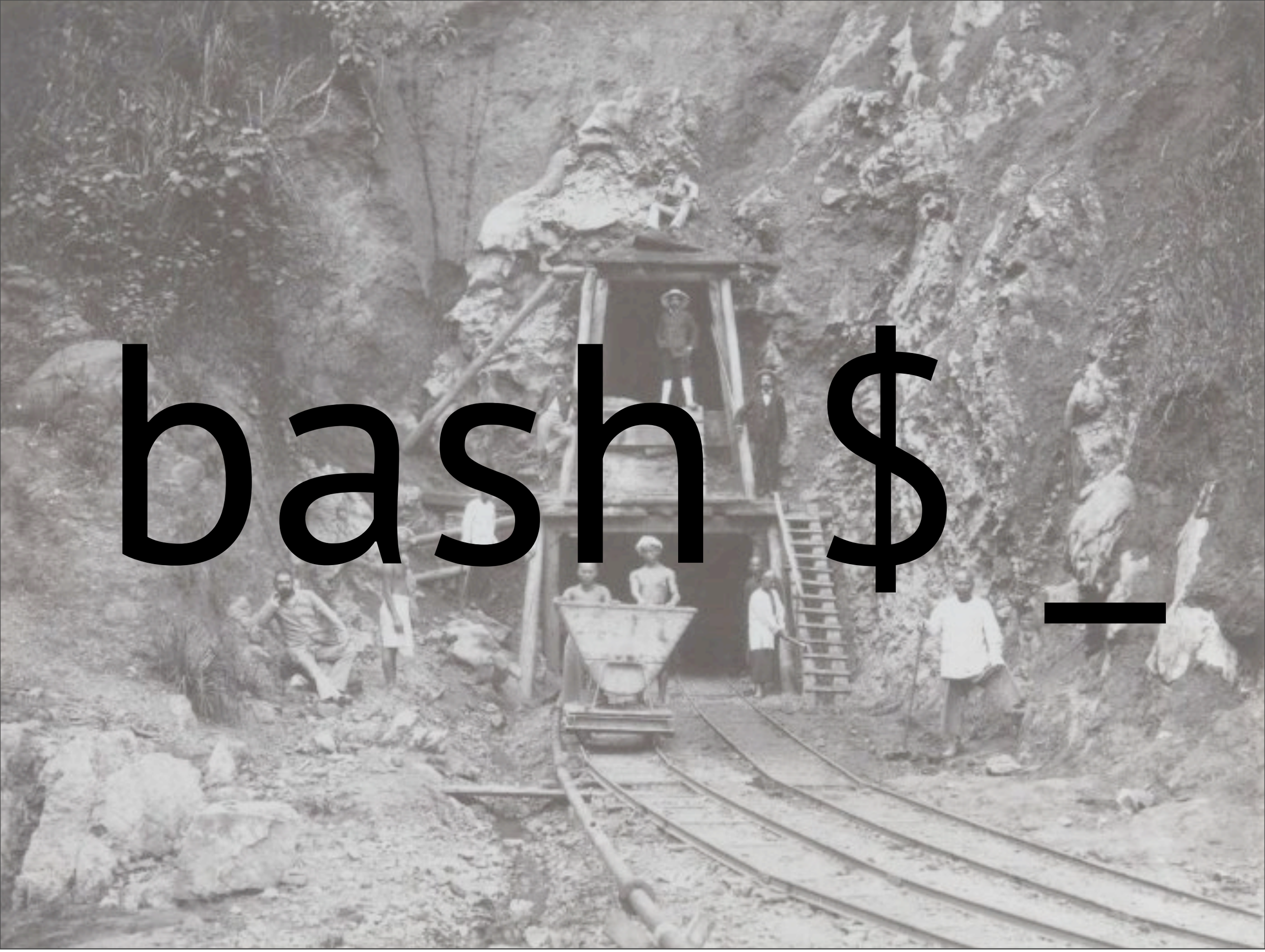
It's all fn until you hit production

user>


user>

$$b = f(a)$$

bash $ _


bash $ _

$$b \cong f(a)?$$

$$b=rand(a)$$

Infrastructure

Automation

$$\text{desired}_{\text{system}} = f(\text{current}_{\text{system}})$$

Bottom Up:

Ops' Perspective

One box

- Concrete: known hardware, OS family and OS version, package version, tooling
- Stateful (erk!)

- Scripts
- Save/Clone Image

PalletOps

Many boxes

- Some configuration is shared
- Certain uniformity is desired

- Scripts+Templates
- Configuration Data Base
- Transport
- Declarative actions
- Image Save/Clone

Many boxes

- Some configuration is shared
- Certain uniformity is desired

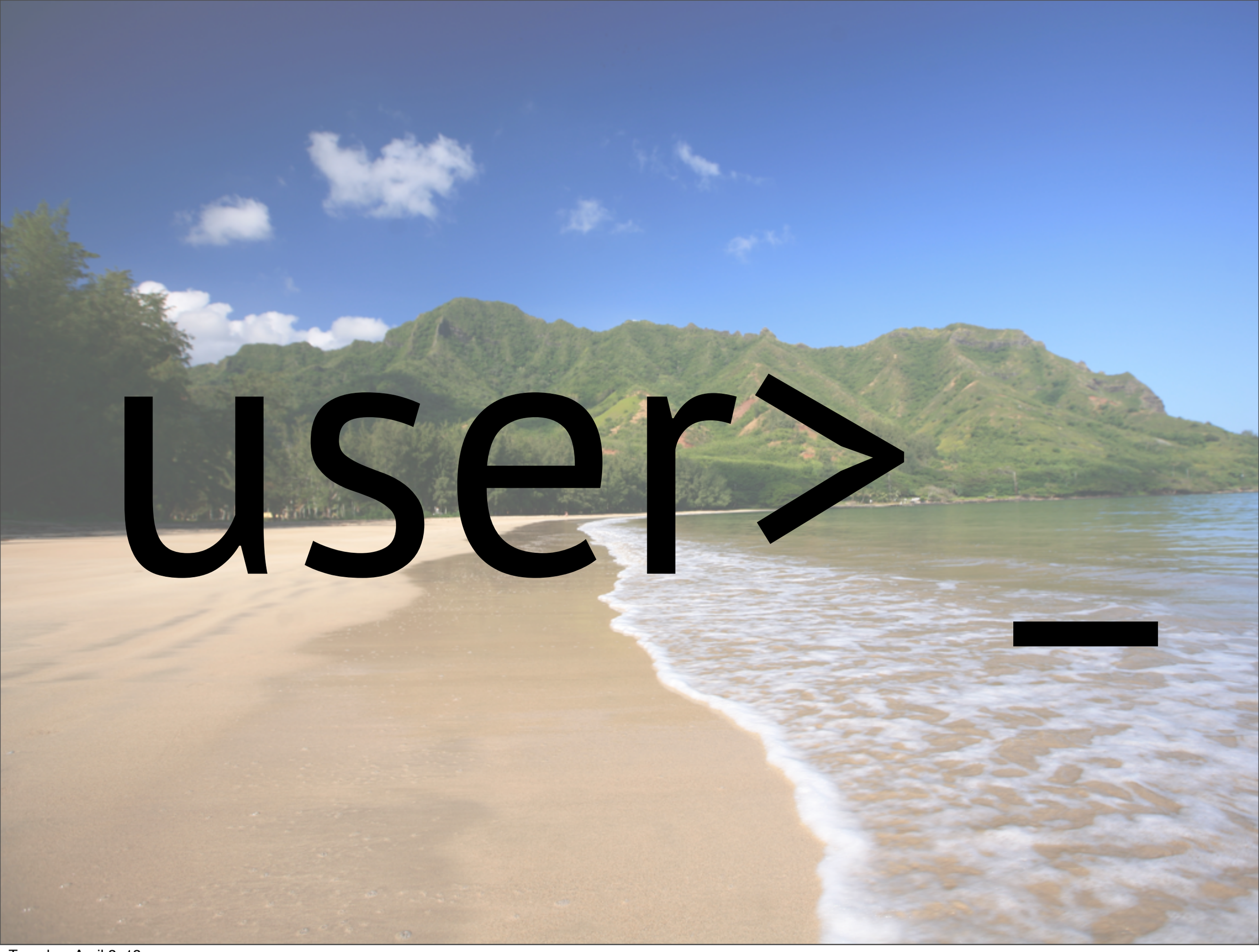
- Scripts+Templates
- Configuration Data Base
- Transport
- Declarative actions
- Image Save/Clone

DevOps

PalletOps

Top Down:

Dev's Perspective

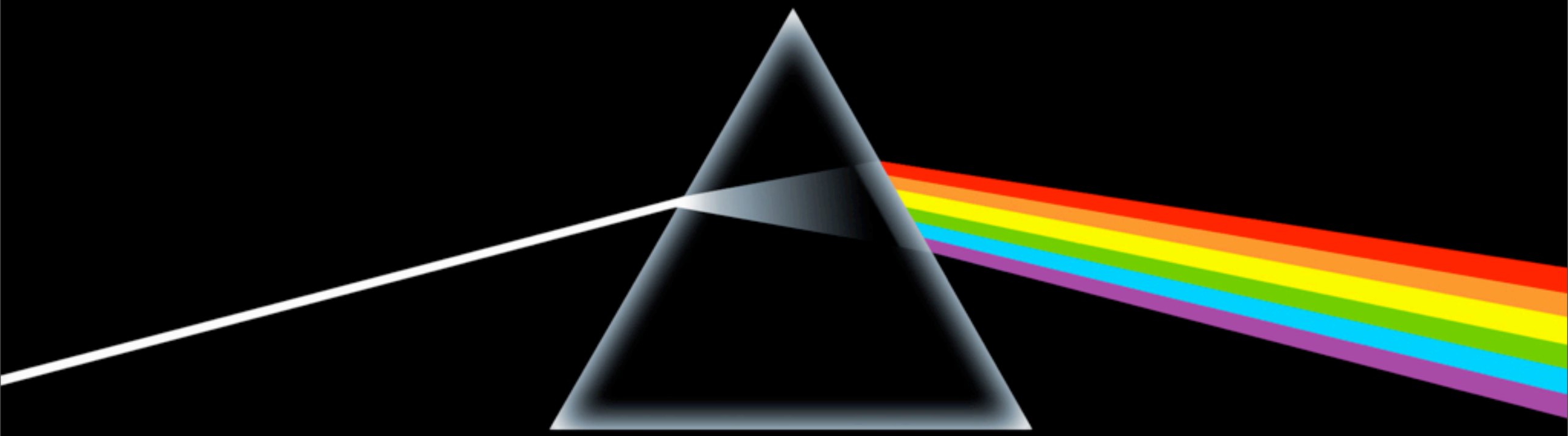

user>

Dev's view

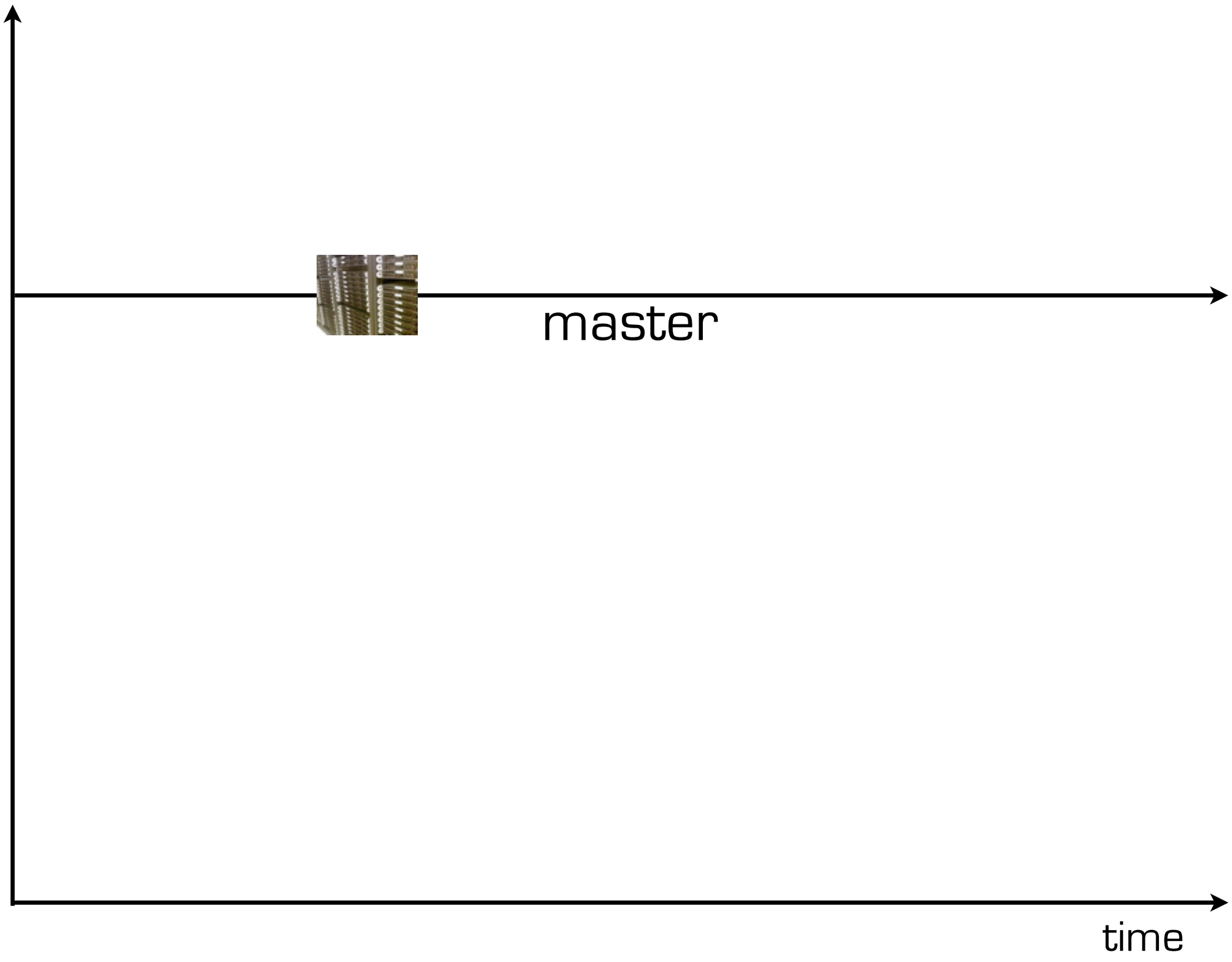
- **OS & Hardware don't matter**
- **Size doesn't matter**
- **Build environments**
  - Dev, Test, Stage & Production
- **Same code everywhere**
- **Development branches**

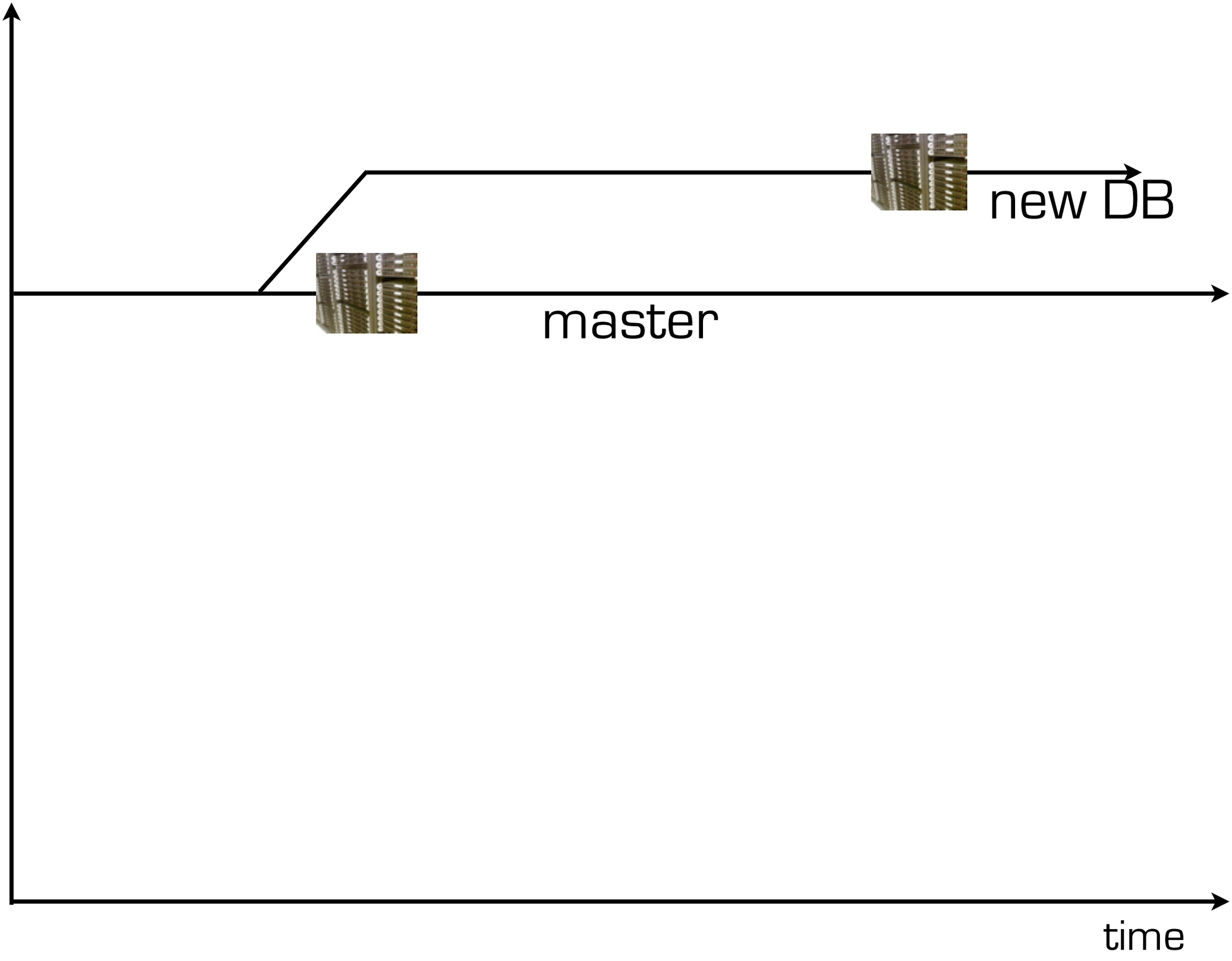
Dev's OTHER view

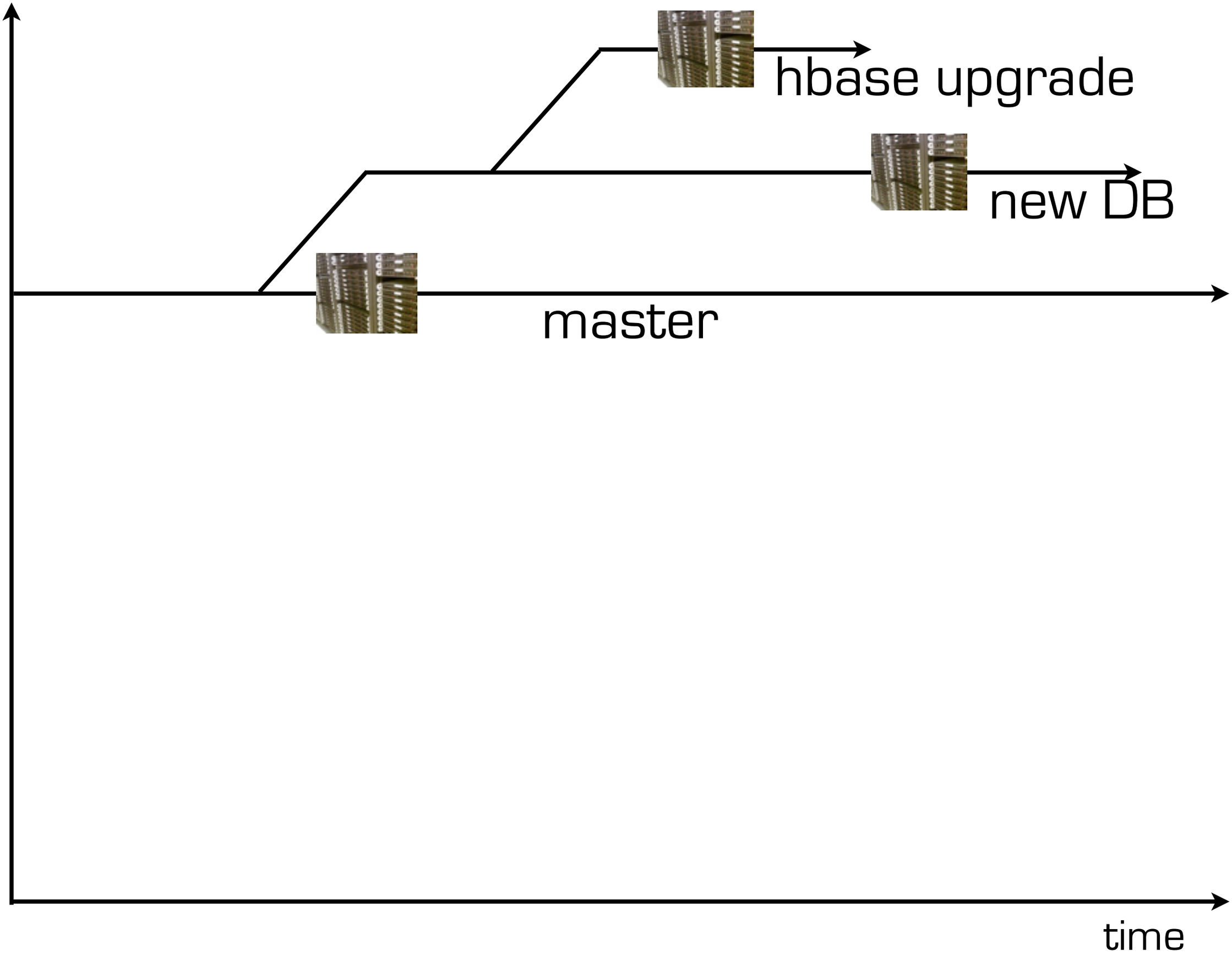
- Test on ubuntu 10.04 and Centos 6.3
- Prepare changes for new version of HBase
- Test recovery with hot stand-by DB
- Benchmark horizontal scalability
- Swap MySQL for MongoDB
- How do firewall rules affect our distributed system?

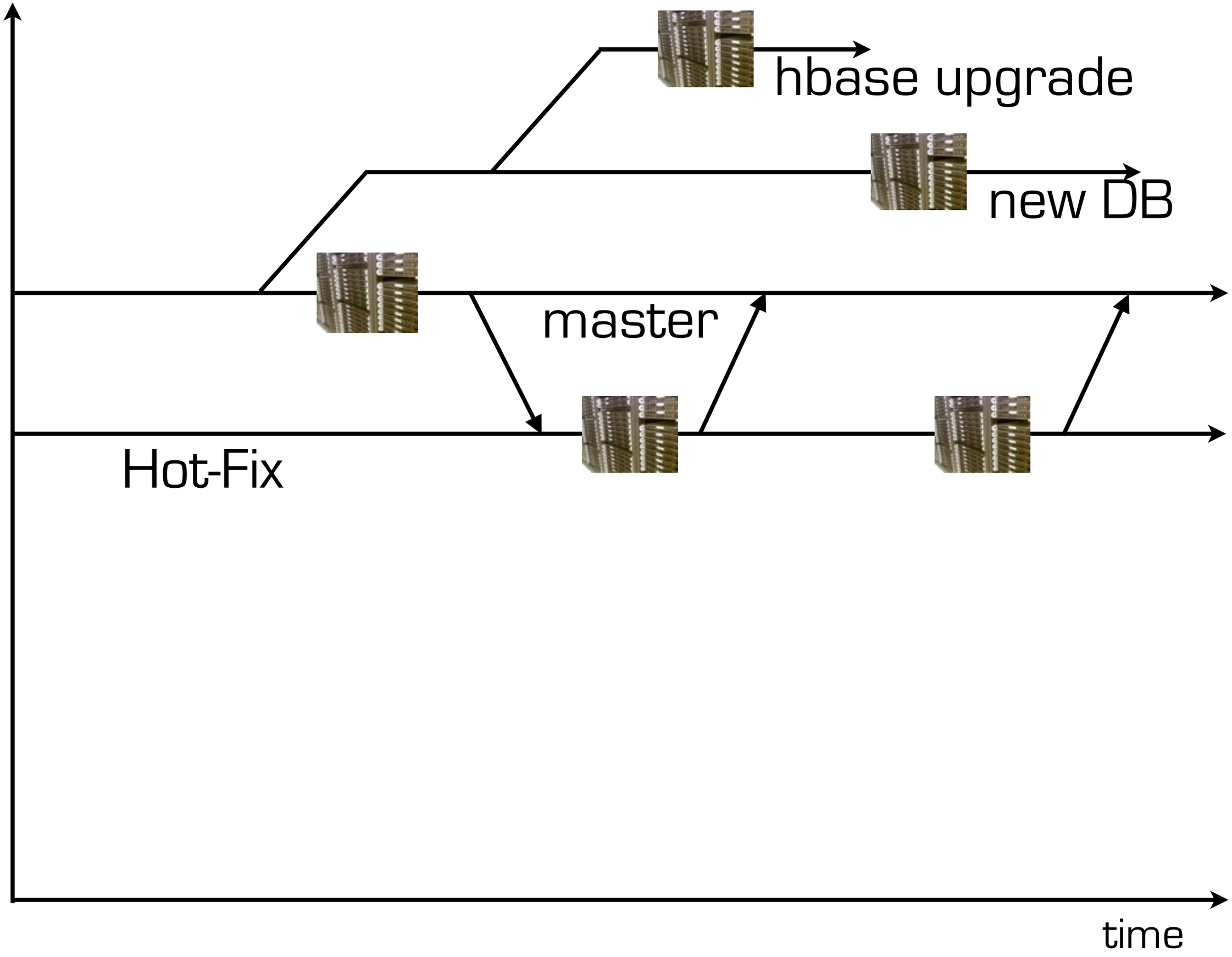


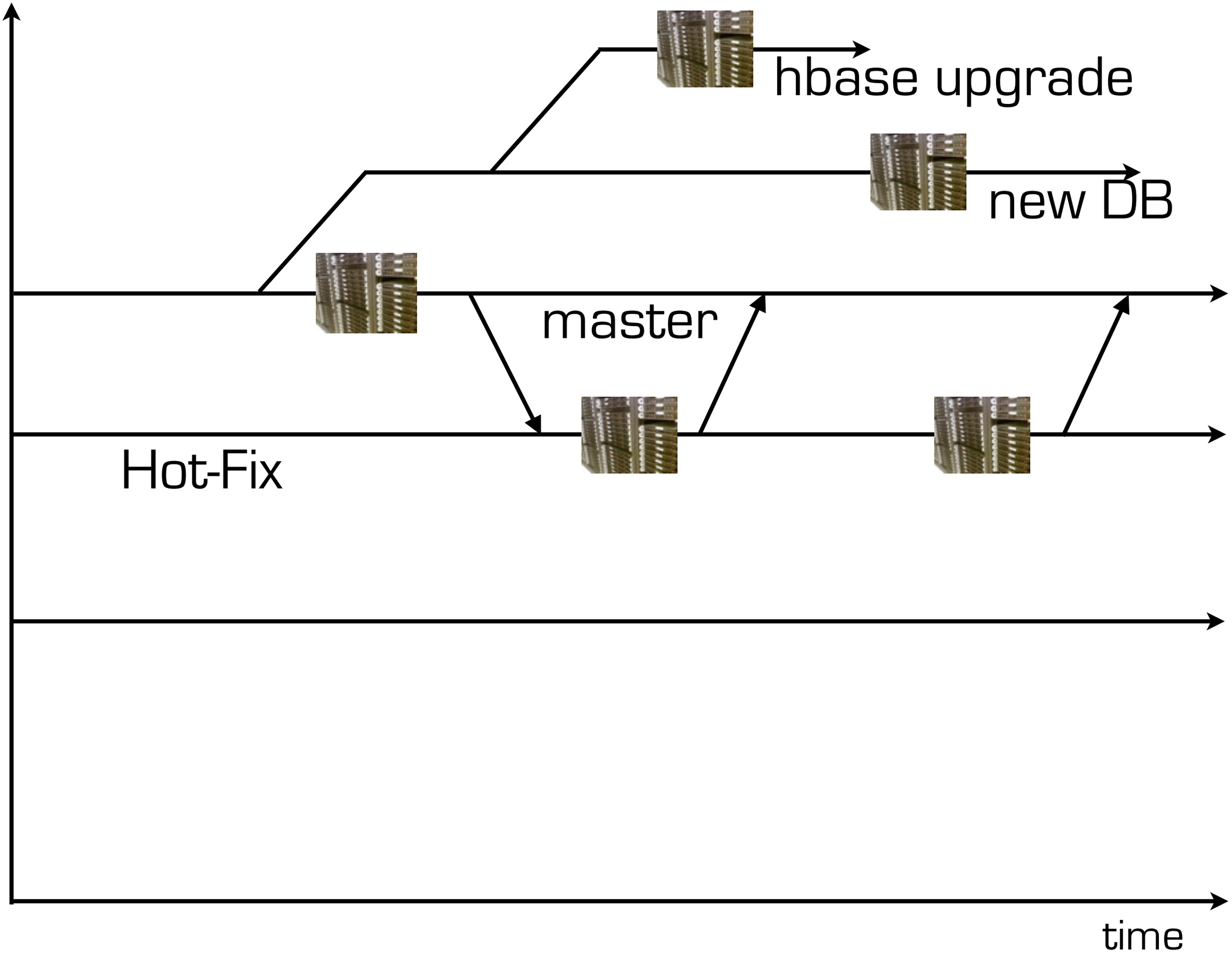
time

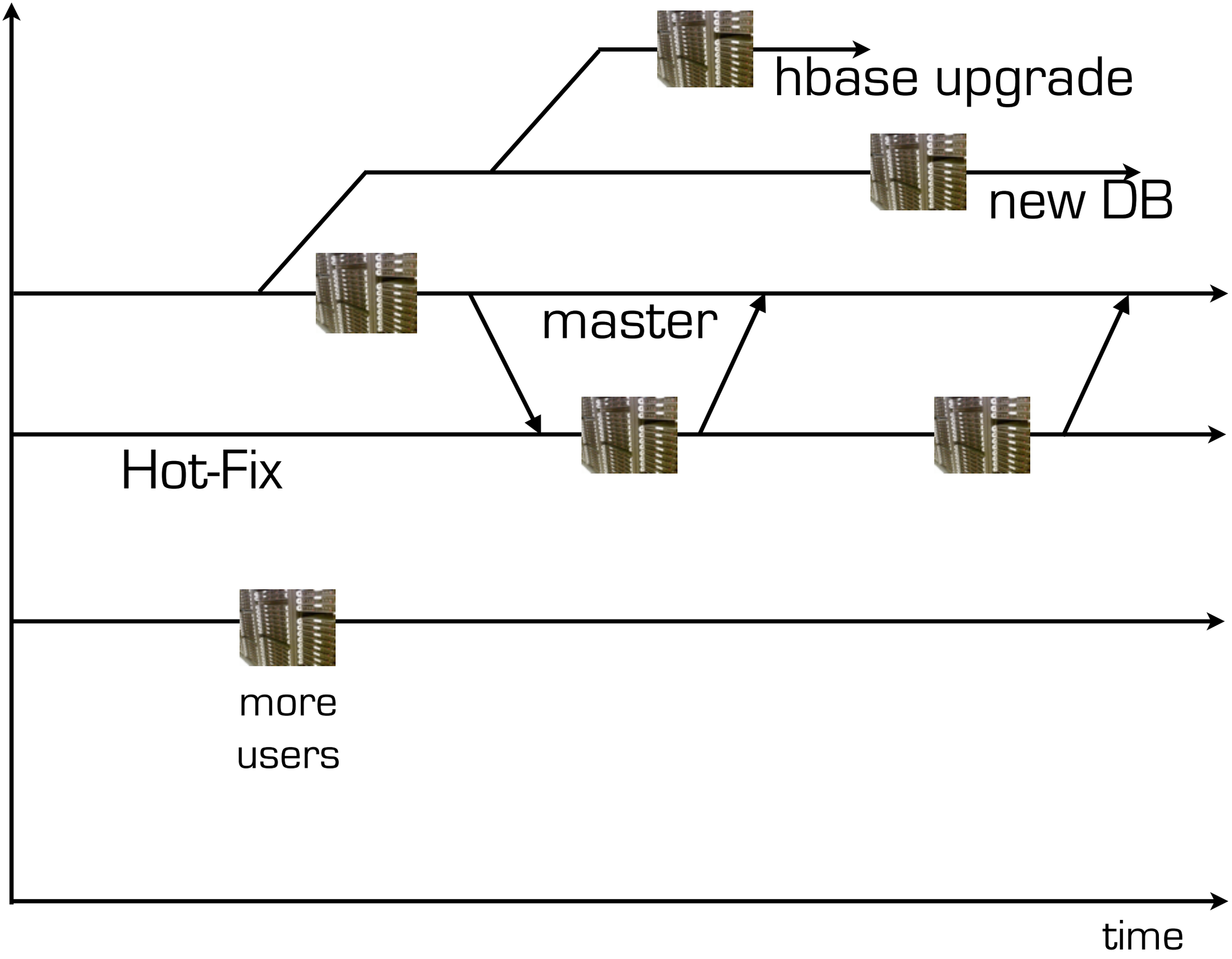

master
time

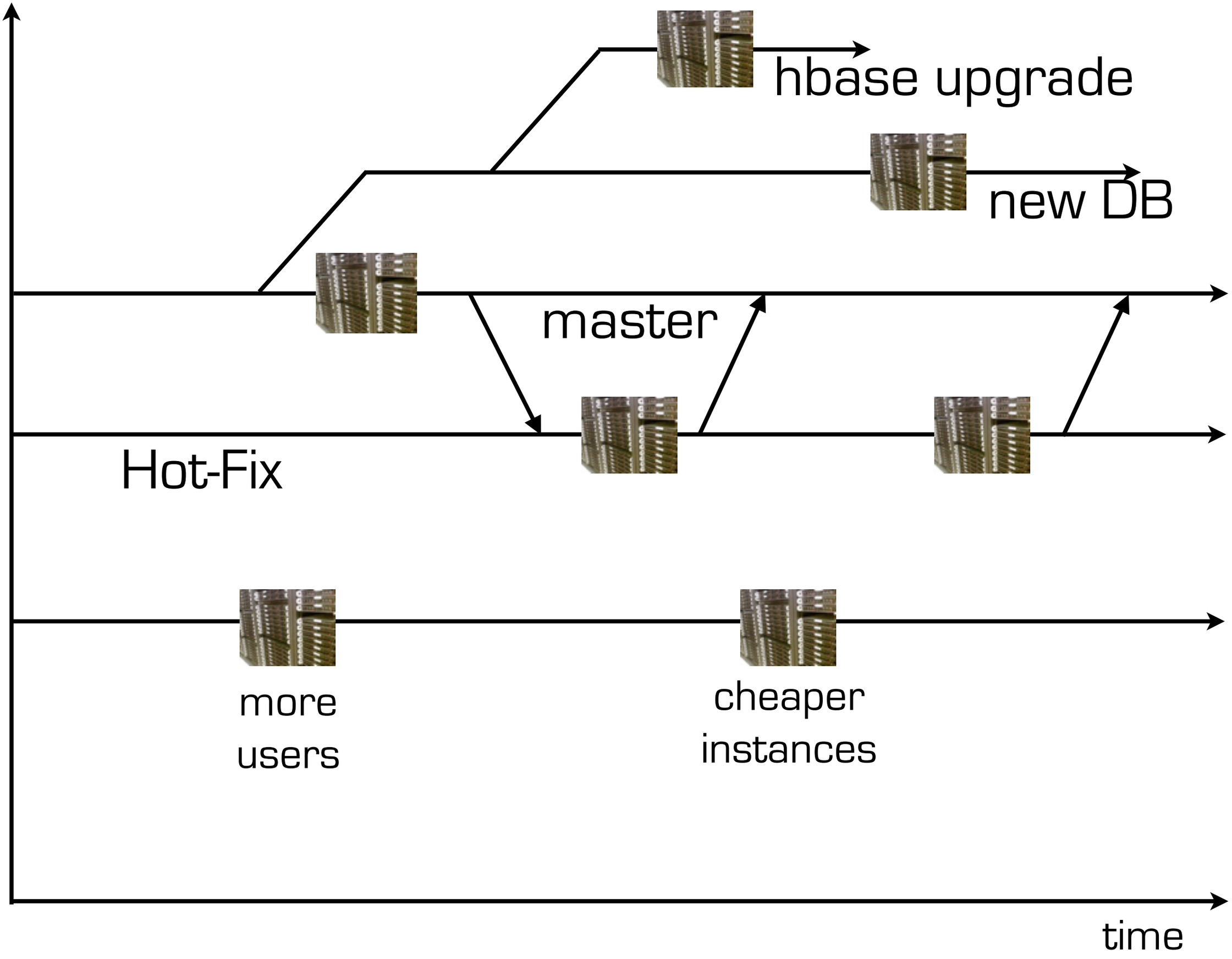

new DB
master
time

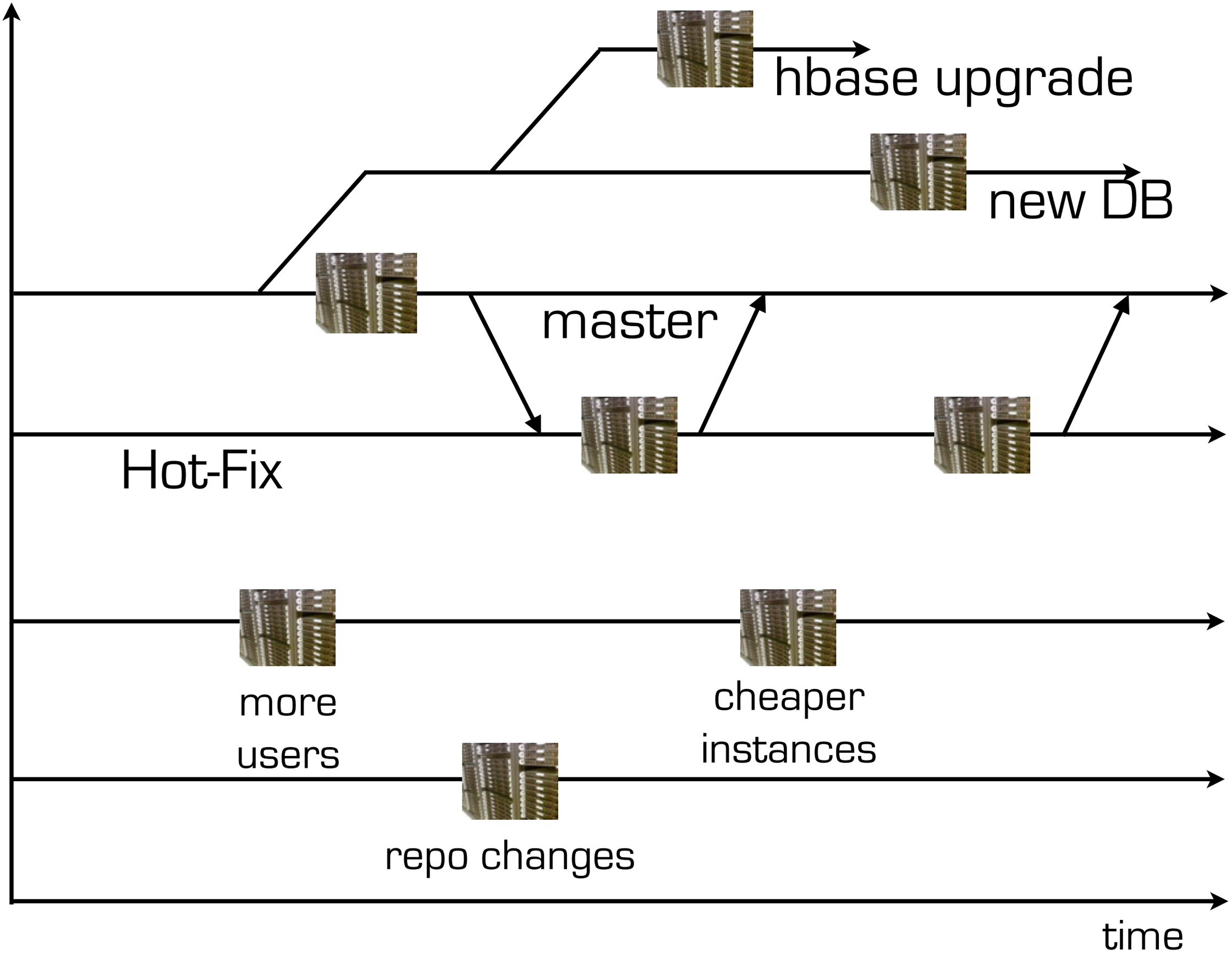

hbase upgrade
new DB
master
time


hbase upgrade
new DB
master
Hot-Fix
time


hbase upgrade
new DB
master
Hot-Fix
time


hbase upgrade
new DB
master
Hot-Fix
more
users
time


hbase upgrade
new DB
master
Hot-Fix
more
users
cheaper
instances
time


hbase upgrade
new DB
master
Hot-Fix
more
users
repo changes
cheaper
instances
time

Moving Parts

(os-families × os-versions × node-types

× configurations × cross-node configs

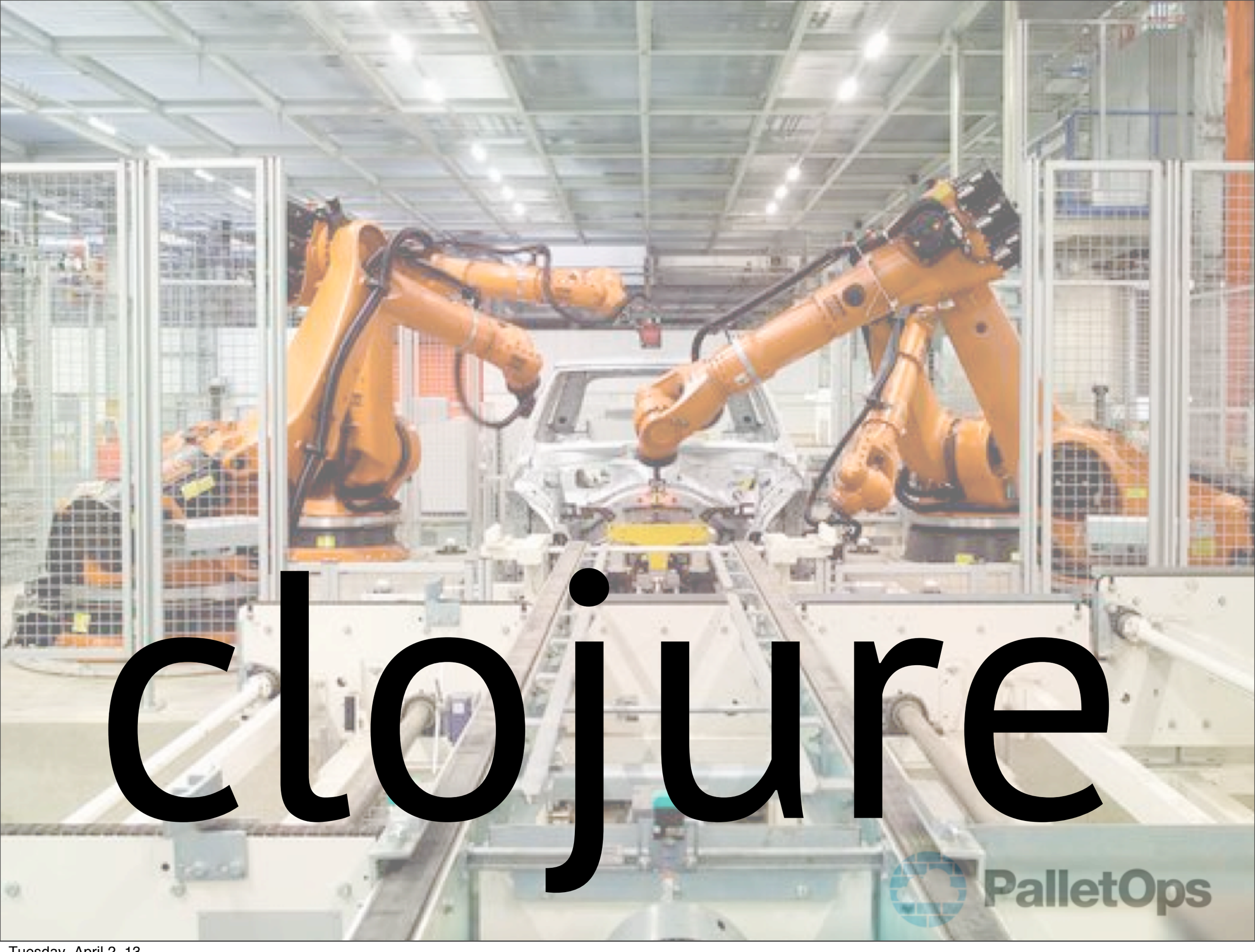
× branches × environments

× cluster shapes

× infrastructure providers ...)

× time

PalletOps


clojure
PalletOps

Pallet

- An Functional Infrastructure Automation platform written in Clojure
- 3+ years of development
- Brain child of Hugo Duncan
- Used in IaaS, PaaS, SaaS and elsewhere
- +30K lines of Clojure

Design Constraints

- Needs to work today
- Write once; run everywhere
- Native support for Cloud infrastructure
- Native support for Clusters
- A library, not a framework, not a server
- Build your own automation
- Build your own abstractions

PalletOps

Pallet

- **A Library**
  - Stateless
  - Embeddable
- **Provides abstractions over**
  - Cloud Providers
  - Hardware
  - OS
  - Configuration
  - Relationships
- **A companion library of Crates (modules) a number of services**

```
(def web-server-nodes
    (node-spec {:image {:os-family :ubuntu
                        :os-version "10.04"}}
               {:hardware {:cpu-count 12
                           :min-ram (* 64 1024)}})))

(def web-servers
    (group-spec “web-server”
        :node-spec node-spec
        :phases
            {:configure (plan-fn
                          (java/install)
                          (tomcat/install))})))

(def ec2 (compute-service :ec2 ...))

(converge {web-servers 5} :compute-service ec2)
```

```
(defn web-server-node [cpus ram os-family os-version]
  (node-spec
    {:image {:os-family os-family
             :os-version os-version}}
    {:hardware {:cpu-count cpus
                :min-ram (* ram 1024)}}
```

```
(defn web-servers [cpus ram os-family os-version]
    (group-spec
        :node-spec node-spec
        :phases
           {:configure (plan-fn
                         (java/install)
                         (tomcat/install))}))
```

```
(converge {(web-servers 12 32 :centos "6.3") 1}
          :compute-service ec2)

(def platforms [[:centos "6.3"]
                [:ubuntu "10.04"]
                [:rhel "7"]])

(def webservers-to-build
  (zipmap (map (fn [[os-family os-version]]
                 (web-servers 12 32
                              os-family os-version))
               platforms)
          (repeat 1)))

(converge webservers-to-build
          :compute-service ec2)

(converge webservers-to-build
          :compute-service virtualbox)
```

PalletOps

Pallet

- Serverless – your infrastructure in Git
- Build your abstractions on top of Pallet's abstractions
- Embed Pallet in your application
- Distribute automation as .jars
- Describe and build your infrastructure in a generative way
- Write once, run everywhere

PalletOps

DSLs

- **Bridging between very different worlds**
- **Pallet has a DSL named Stevedore for abstracting over scripting languages and tooling**
  - Clojure + Stevedore + OS family/version = correct script
  - Intertwine clojure code with shell scripts in a way similar to macros do with clojure.
  - Pallet itself is built on Stevedore, and all of it is OS independent
  - Script abstractions – create and reuse scripts inside stevedore

DEMO

Multi-methods

- **For when the same abstract concept has many concrete representations**
- **Open extensibility – very important!**
- **Pallet uses them for abstracting over OS family & version, at both script and plan levels, e.g.:**
  - Passwordless sudo is done differently in Centos 5.3 and Centos 6+
  - Java is installed different depending on the OS family and version
  - Java is installed differently if it's Oracle JDK or OpenJDK

PalletOps

Data

- Data allows to perform heavy lifting operations very simply
- Data is easy to test, inspect, debug, log
- Defer execution as much as possible
- Pallet internals are built around data manipulation
  - Coupling between components is data
  - All intermediate representation is data, until right before the execution

PalletOps

Data

- **Pallet represents abstractions as data structures: they compose well**
  - Abstractions built as maps
  - Infrastructure can be defined dynamically and programmatically
  - Mixins with different merging semantics
    - E.g. if you merge different specs with the same function, the actions are first joined and then reordered
  - environment, node, server, group, cluster...

PalletOps

Compilers

- **Powerful design pattern to bridge two different worlds**
- **In a compiler**
  - Start with your code and target architecture
  - Build abstract data representation of the code
  - Live in the data world for as long as you can, transforming this data (pipeline)
  - translate to data into OPCodes for the target CPU/Architecture

PalletOps

plan=lift(current system)

plan=lift(current system)

desired system =exec(plan)

DEMO

Finite State Machines

- **Provide a better understanding and management of external stateful resources**
- **Separate state from execution**
- **Pallet's main execution paths are controlled by state machines**
  - Separate side-effect free from side-effect all-the-things
  - Query the execution state
  - Write your own functions on existing primitives

PalletOps

Concurrency

- **Immutability in Clojure is priceless...**
  - ... for everything else, there are atoms and refs
- **99% of pallet is stateless**
  - functions are easy to test
  - we use threads at the target-node level
  - a session is kept in thread-local atoms
  - 100's of nodes

PalletOps

JVM

- Access to well tested, well understood and high-performance libraries
- dm;hc
- **Pallet uses 3 java libs:**
  - Multi-cloud support via jclouds (optional)
  - slf4j for logging
  - jsch – ssh support

JVM

- **.jar**
  - Embed Pallet in your code
  - Distribute pallet automation as jars
  - Build your domain on Pallet
  - Every developer can build their own infrastructure
  - Projects can distribute their infrastructure builder
  - Versioning, signing, etc...

$$b \cong f(a)?$$

Where are we now?

- Functional and programmatic infrastructure automation
- Works on most cloud providers and target OSs (as long as they're *nix)
- Build complex and flexible clusters
- Fast development paths
- Easy to build your domain abstractions on infrastructure

What are we doing with it?

- **Pallet Big Data – Automate everything Big Data**
  - Amazon EMR everywhere!
  - Build Clusters, keep them happy, run workloads
- **bricklayer – like git, but for builds**
  - Decouple build from server
  - Build evolves with code
- **lein pallet up – download project from git, run it everywhere**

PalletOps

Infrastructure Automation

~

Clojure Development

~

http://palletops.com