

10 deploys per day

Dev & ops cooperation at Flickr

John Allspaw & Paul Hammond  
Velocity 2009

3 billion photos

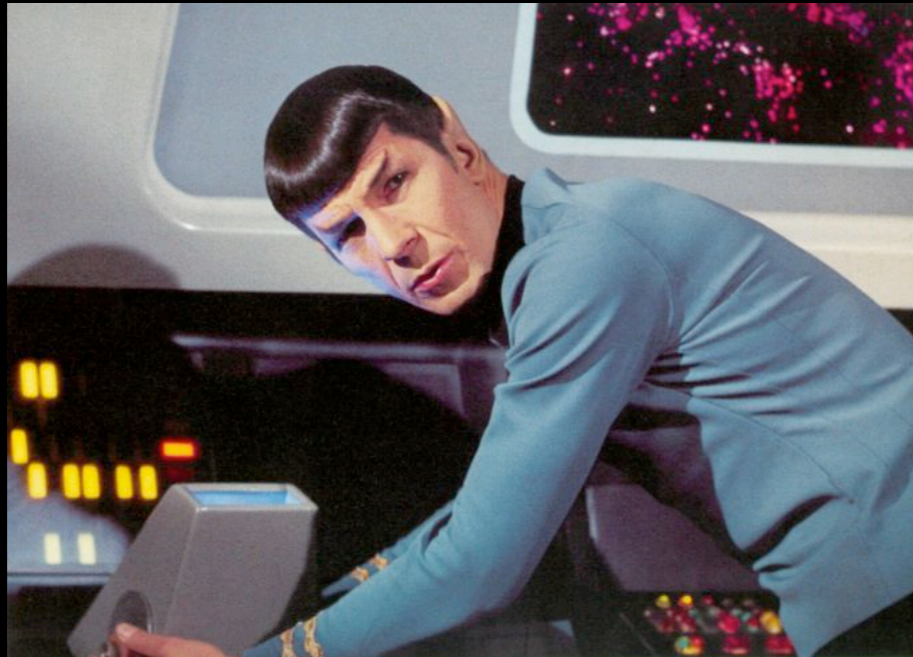
40,000 photos per second



# Dev *versus* Ops

“It’s not my machines,  
it’s your code!”

“It’s not my code,  
it’s your machines!”



Little bit weird  
Sits closer to the boss  
Thinks too hard



Pulls levers & turns knobs  
Easily excited  
Yells a lot in emergencies

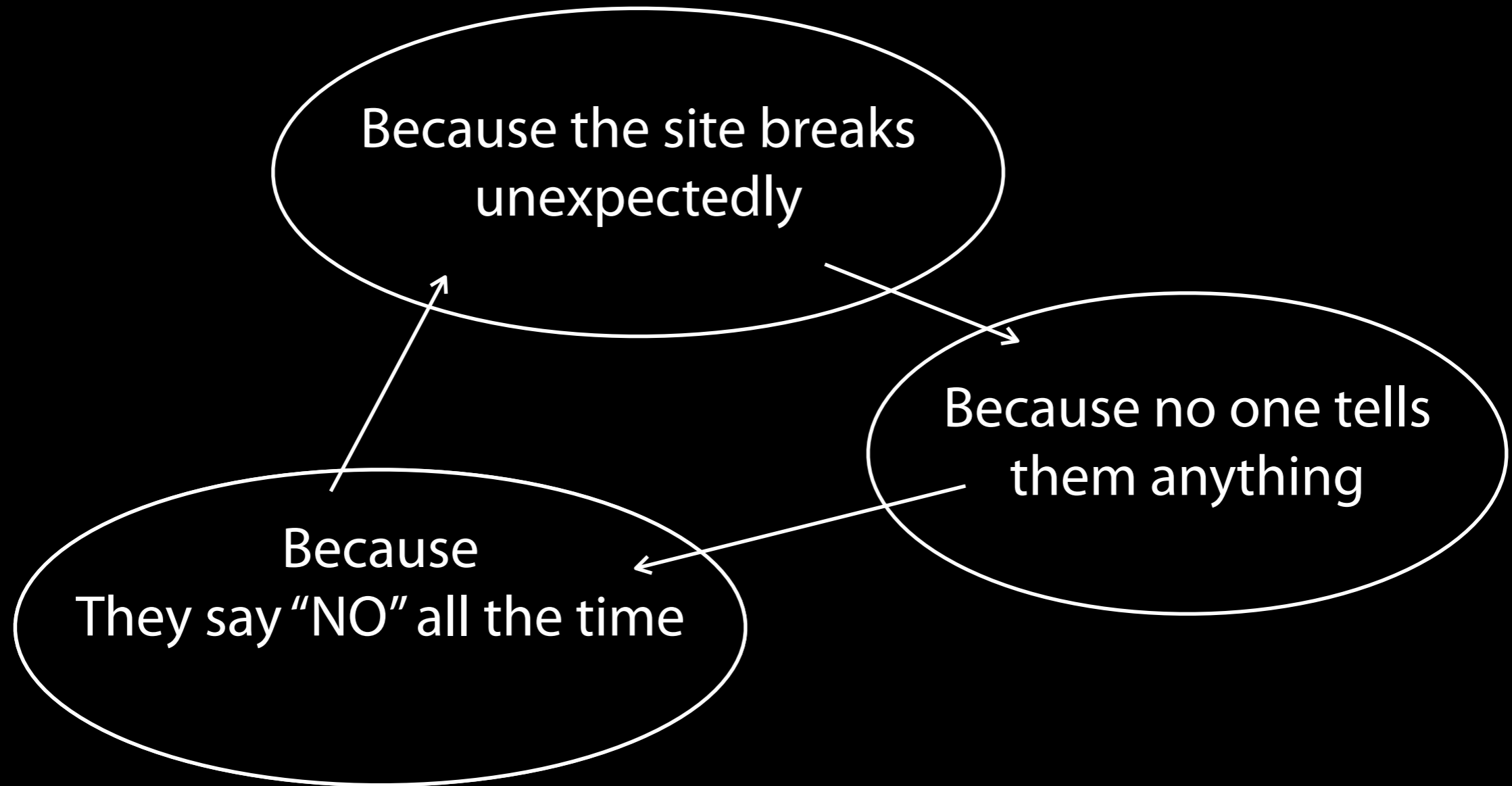




Says "No" all the time  
Afraid that new fangled things will break the site  
Fingerpointy



# Ops stereotype





# Traditional thinking

Dev's job is to add new features  
Ops' job is to keep the site stable and fast

Ops' job is **NOT** to keep the site stable and fast

Ops' job is to **enable** the business

(this is dev's job too)

The business **requires** change

But change is the root cause of most outages!

Discourage change in the interests of stability

or

Allow change to happen as often as it needs to

Lowering risk of change  
through **tools** and **culture**





Dev and Ops

Ops who think like devs  
Devs who think like ops

“But that’s me!”

You can **always** think more like them

Tools

# 1. Automated infrastructure

If there is only one thing you do...

Chef

CFengine

BCfg2

FAI

1. Automated infrastructure

If there is only one thing you do...

System Imager

Puppet

Cobbler



Role &  
configuration  
management

OS imaging



## 2. Shared version control



Everyone knows where to look

# 3. One step build

# Staging

## Last SVN-triggered precompile of the templates

```
Recompile started at 20:43:45 on 2009-06-19
saving prev version..ok (3 ms)
exporting svn.....ok (3202 ms)
changing paths.....ok (1 ms)
grabbing old files...ok (6 ms)
creating folders.....ok (317 ms)
compiling config.....
    host: stage-local
    colo: mud
    prod: 1
.....ok (349 ms)
precompiling.....ok (29.816 sec)
sync strings.....
    added: 14
    deleted: 1
.....ok (16.065 sec)
cleaning up.....ok
```

## Last Stage:

```
checking precompile..ok
exporting svn.....ok (6.034 sec)
fetching versions....ok (4225 ms)
changing paths.....ok (2 ms)
creating folders.....ok (0 ms)
grabbing snapshot....ok (16489 ms)
baking help form.....ok (2776 ms)
baking autosuggest...ok (298 ms)
fetching generated...ok (6 ms)
push to staging.....ok (8525 ms)
asset foldup.....ok (280 ms)
rewriting CSS paths..ok (2745 ms)
baking output.js.....ok (672 ms)
calcing js md5s.....ok (210 ms)
```

Completed in 42.911 sec

[Perform Staging](#)

If the staging version looks wonky, you can ask the Magi-cal Pixie to recompile all the templates.



[Oh Magi-cal Pixie, Make Everything New](#)

### 3. One step build and deploy

# Deploy Log

Before staging or deploying anything that might block deploy, you must check and update the deploy log.

```
[2009-06-18 13:12:13] [mygrant] DEPLOY OK
[2009-06-18 13:08:15] [mygrant] site staged
[2009-06-18 13:05:25] [mygrant] NO DEPLOY PLEASE
[2009-06-18 11:29:23] [aaron] site staged
[2009-06-18 11:19:26] [harmes] site deployed (changes...)
[2009-06-18 11:16:44] [harmes] starting deploy...
[2009-06-18 11:15:15] [harmes] site staged
[2009-06-18 10:20:26] [harmes] site deployed (changes...)
[2009-06-18 10:15:00] [harmes] starting deploy...
[2009-06-18 10:11:16] [harmes] site staged
```

# Deployment

When the staging version is ready, click the button below.

WARNING: This sync's the staging version to the live servers. In theory [this is what will change](#), but you might want to test it maybe?

## Last deploy:

```
pulling site from staging host...ok (10.096 sec)
syncing to ramdisk in mud.....ok (5.992 sec)
syncing to ramdisk in re2.....ok (7.443 sec)
stage 1.....ok (25.322 sec)
stage 2.....ok (42.108 sec)
stage 3.....ok (45.952 sec)
```

Completed in 2 min, 16 sec

# Deployment

When the staging version is ready, click the button below.

WARNING: This sync's the staging version to the live servers. In theory [this is what will change](#), but you might want to test it maybe?

## Active Deploy:

```
pulling site from staging host...ok (4636 ms)
syncing to ramdisk in mud.....ok (6.049 sec)
syncing to ramdisk in re2.....ok (7.349 sec)
stage 1.....ok (50.25 sec)
stage 2.....ok (1 min, 37 sec)
stage 3.....
```

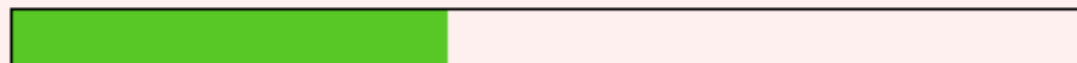
### Phase 1



### Phase 2



### Phase 3



Waiting for 151 hosts

Running for 2 minutes & 58 seconds

I'm Feeling Lucky

```
[2009-06-22 16:03:57] [harmes] site deployed (changes...)
```

# Who? When? What?



[Back to Dashboard](#)[Status](#)[Changes](#)[Workspace](#)[Build Now](#)[Delete Project](#)[Configure](#)[Subversion Polling Log](#)

## Build History (trend)

- [#31 Jun 3, 2009 3:06:12 PM](#)
- [#30 Jun 2, 2009 7:03:12 PM](#)
- [#29 Jun 2, 2009 7:00:13 PM](#)
- [#28 Jun 2, 2009 6:59:13 PM](#)
- [#27 Jun 2, 2009 6:51:12 PM](#)
- [#26 Jun 2, 2009 6:50:12 PM](#)
- [#25 Jun 2, 2009 6:09:12 PM](#)
- [#24 Jun 2, 2009 6:07:12 PM](#)
- [#23 Jun 2, 2009 5:57:12 PM](#)
- [#22 May 29, 2009 5:58:12 PM](#)
- [#21 May 29, 2009 4:41:12 PM](#)
- [#20 May 29, 2009 2:15:12 PM](#)

## Project tools\_statsd

[add description](#)[Workspace](#)[Last Successful Artifacts](#)

- [flickr\\_statsd-20090603.1244066774.T51204.tgz](#) 

[Recent Changes](#)

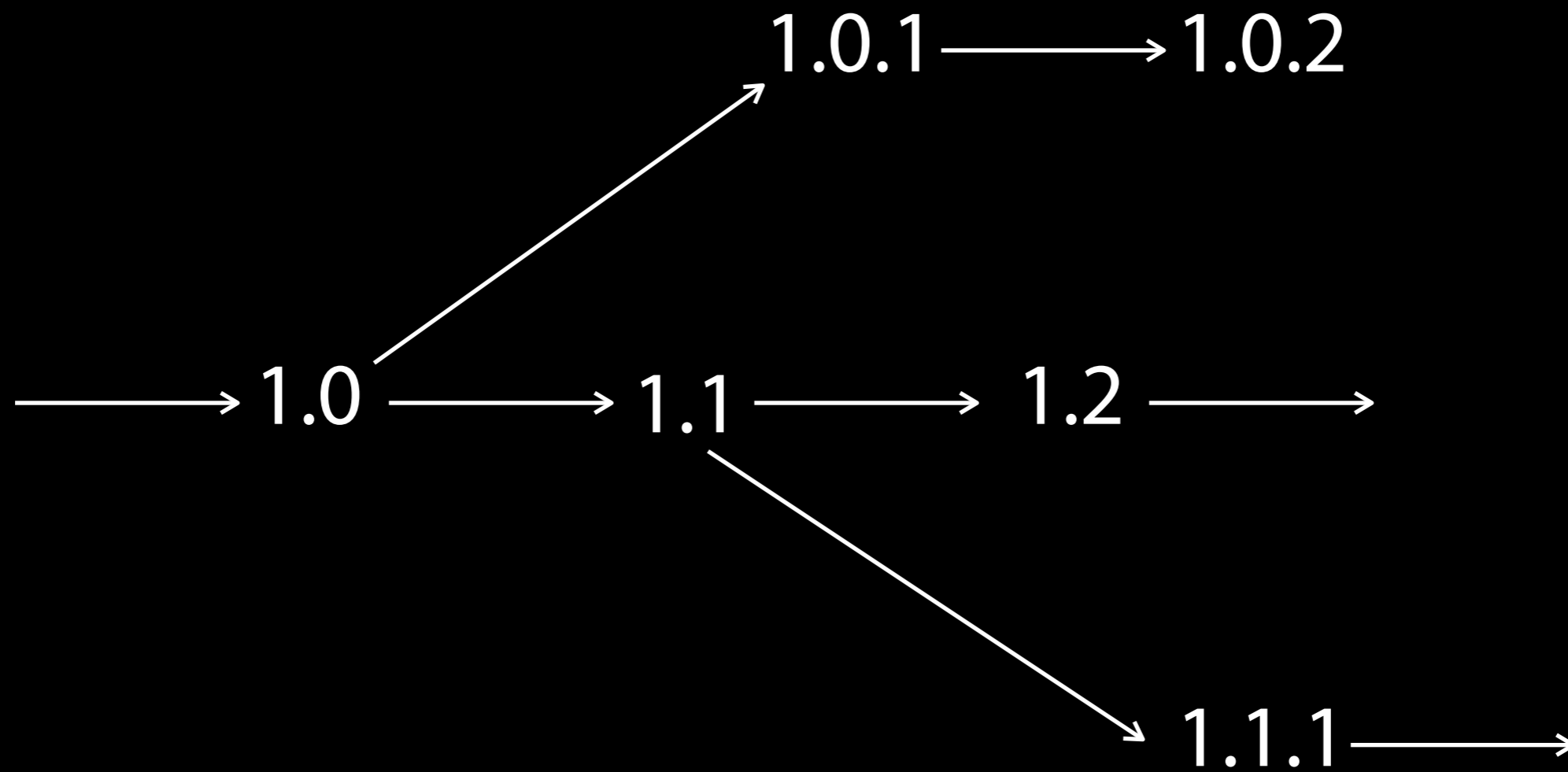
## Permalinks

- [Last build \(#31\), 19 days ago](#)
- [Last stable build \(#31\), 19 days ago](#)
- [Last successful build \(#31\), 19 days ago](#)
- [Last failed build \(#10\), 1 mo 5 days ago](#)



Small frequent changes

# 4. Feature flags (aka branching in code)



Desktop software

————→ r2301 —————→ r2302 —————→ r2306 —————→

Web software

Always ship trunk



Everyone knows exactly where to look

# Feature flags

```
#php
if ($cfg['enable_feature_video']){
    ...
}
```

```
{* smarty *}
{if $cfg.enable_feature_beehive}
    ...
{/if}
```



Members Only

Private betas

# Bucket testing





Dark launches

STOP

Free  
contingency  
switches

# 5. Shared metrics

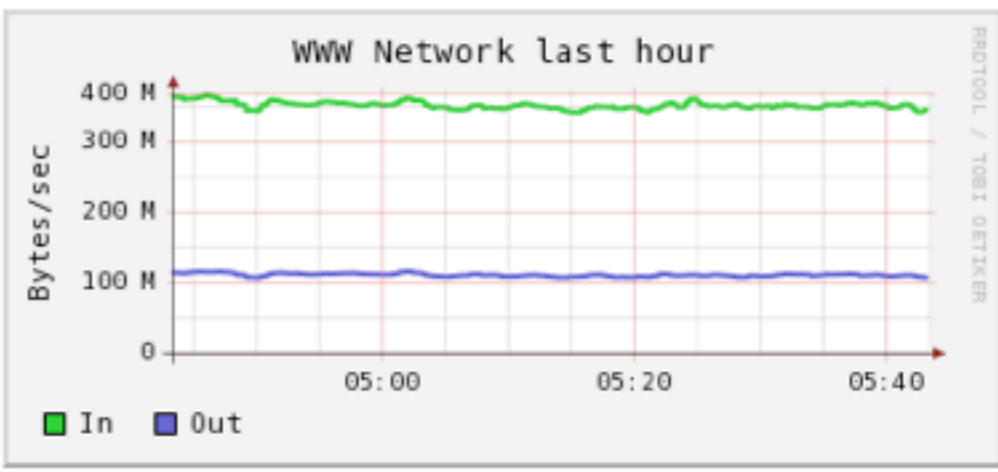
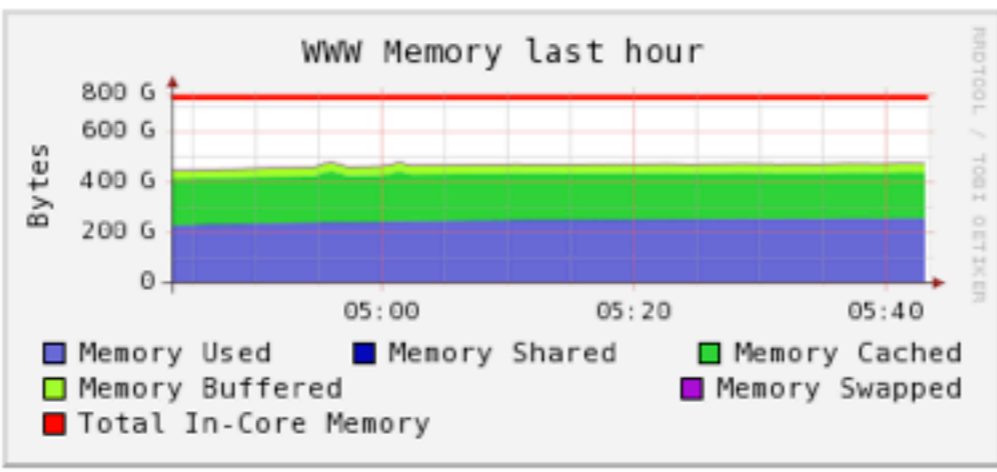
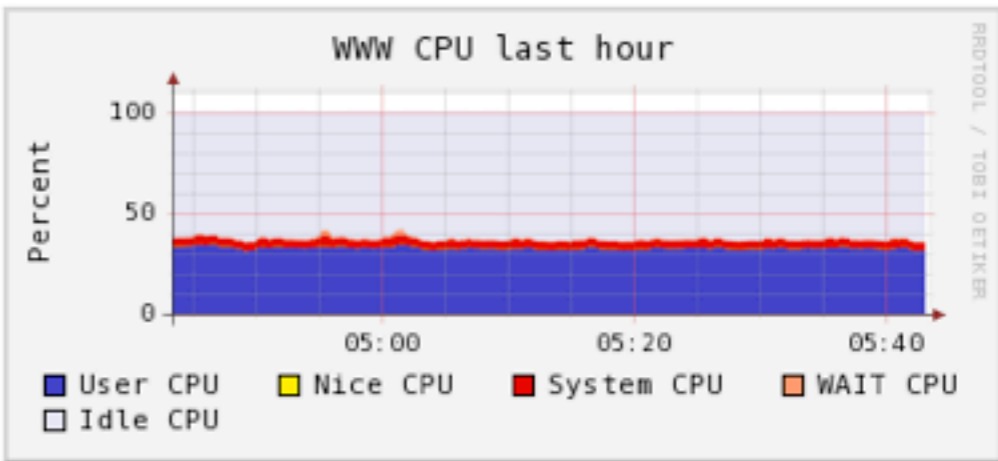
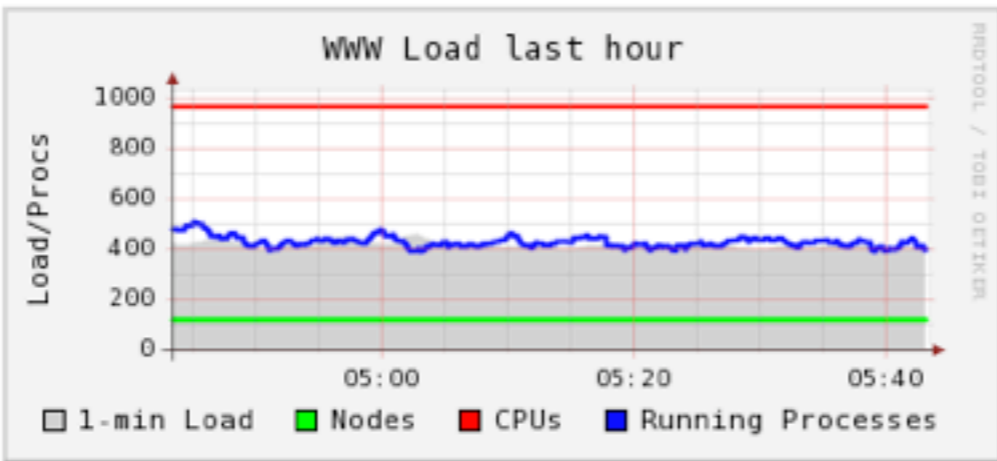
custom graph

### Overview of WWW

CPU's Total:   
Hosts up:   
Hosts down:

Avg Load (15, 5, 1m):  
**41%, 42%, 41%**

Localtime:  
**2009-06-19 05:43**

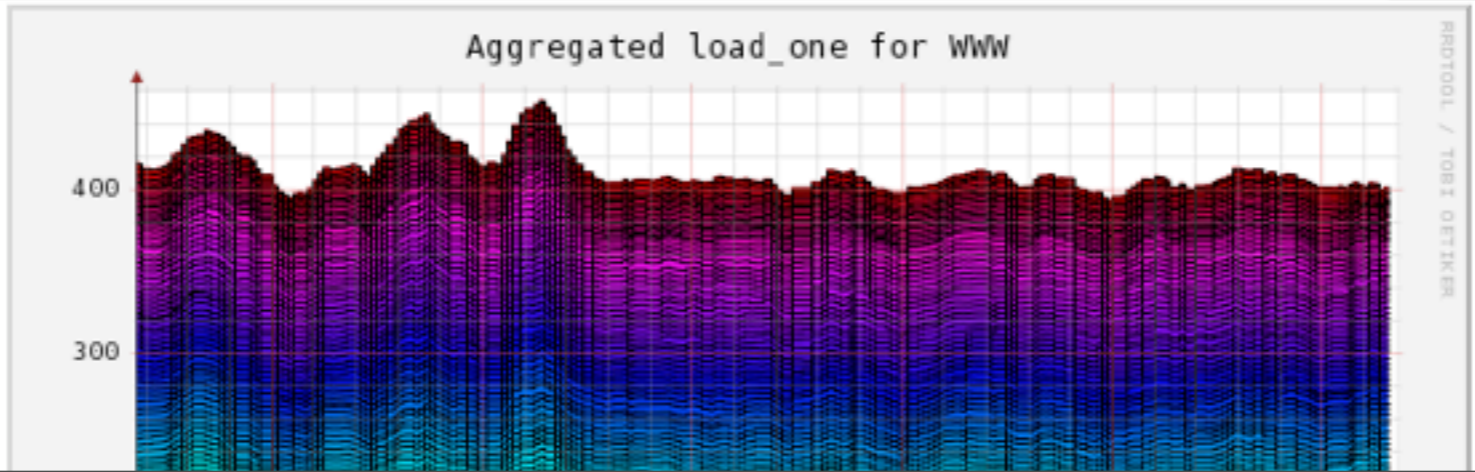


#### Cluster Load Percentages

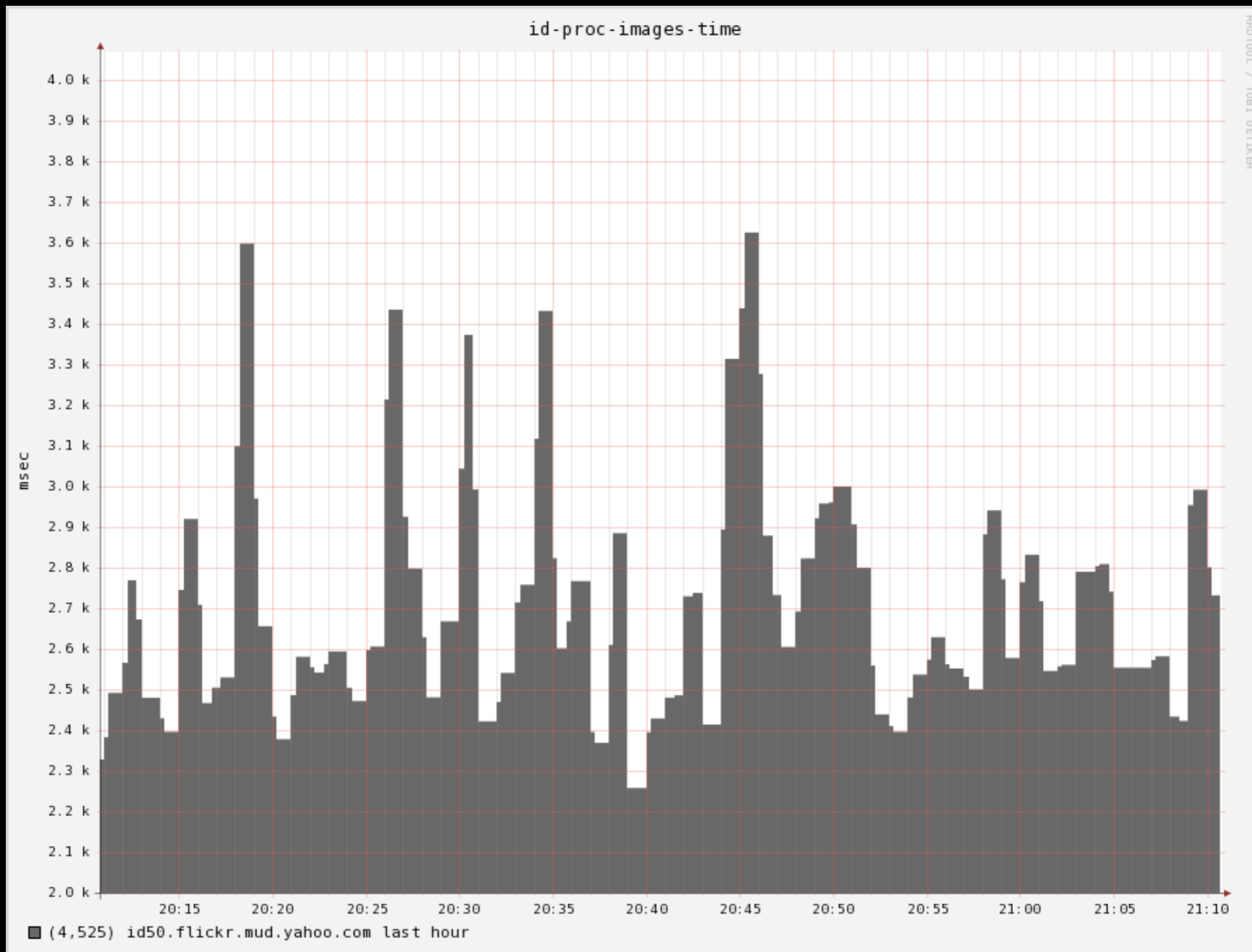


- 50-75 (9.92%)
- 25-50 (85.12%)
- 0-25 (4.96%)

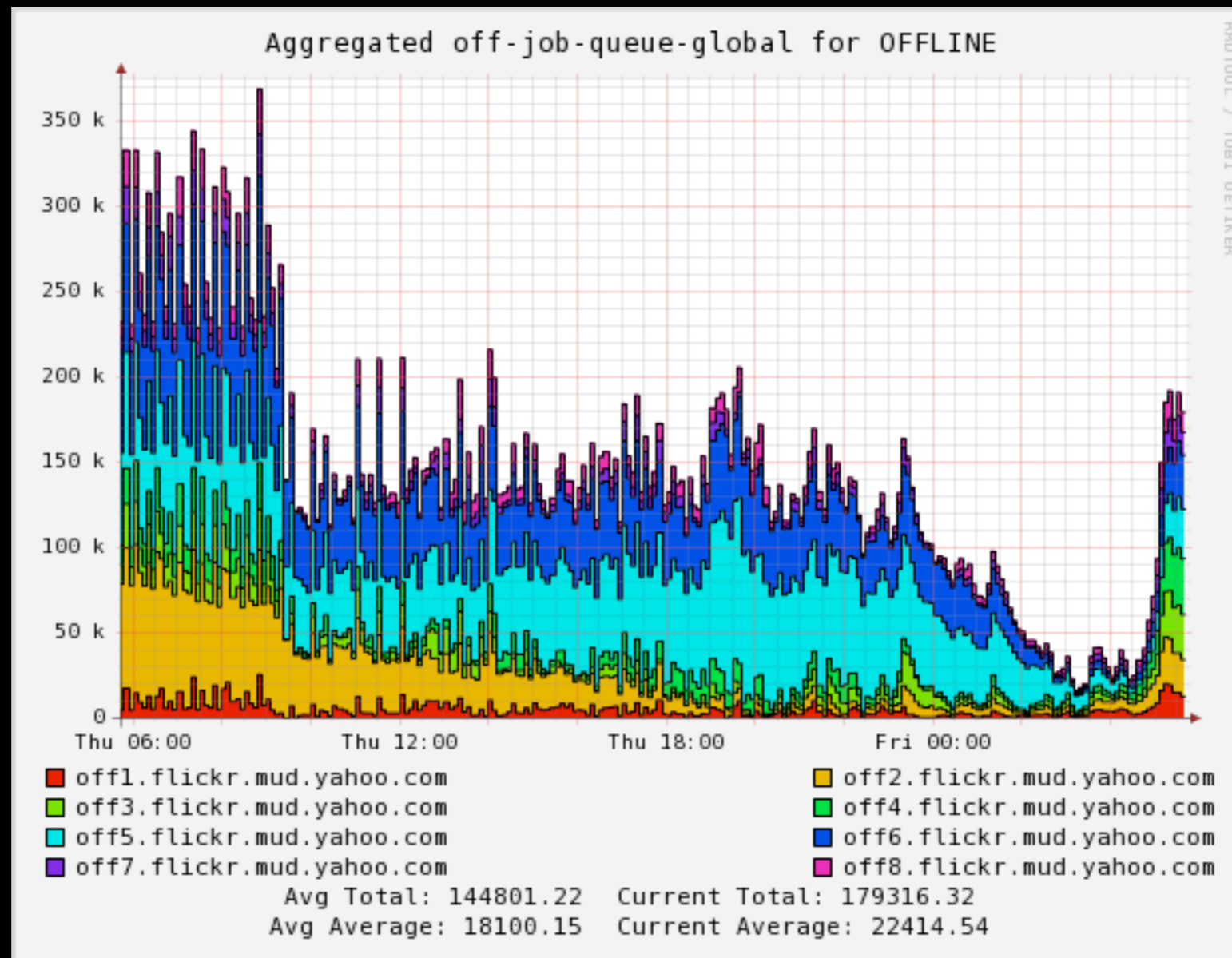
Show Hosts:  yes  no | WWW load\_one last hour sorted descending | Columns



# Application level metrics

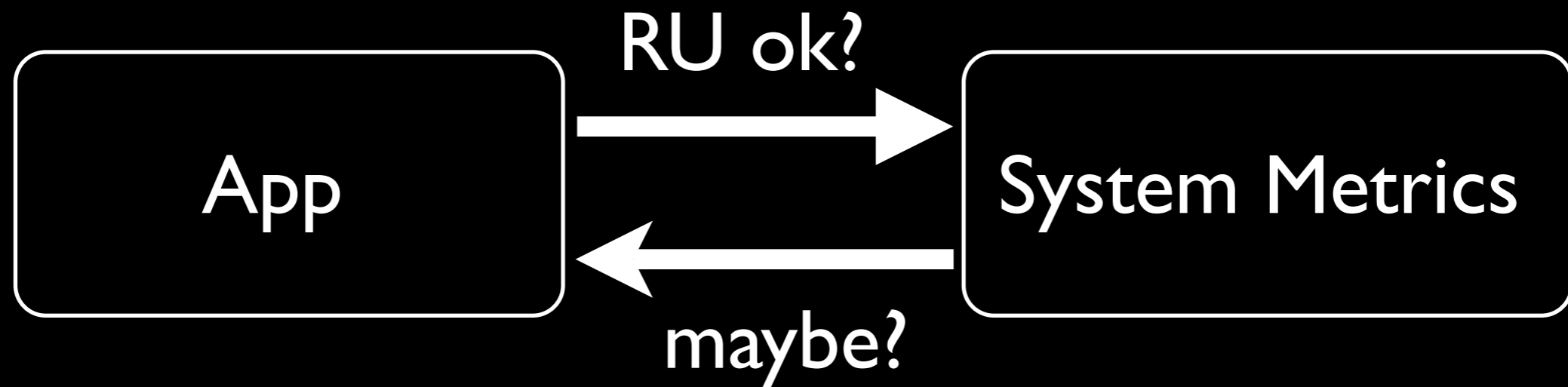


# Application level metrics

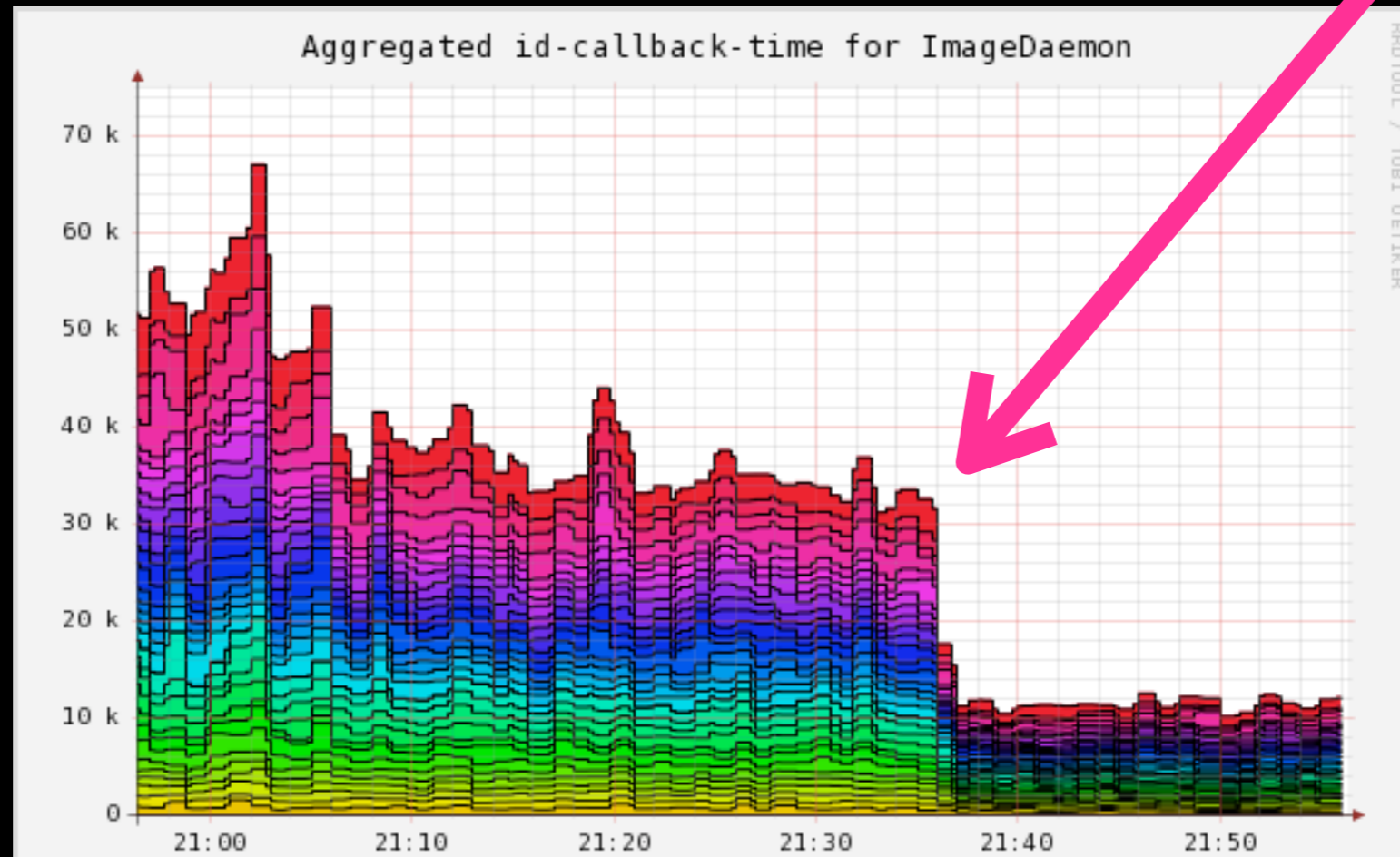




# Adaptive feedback loops

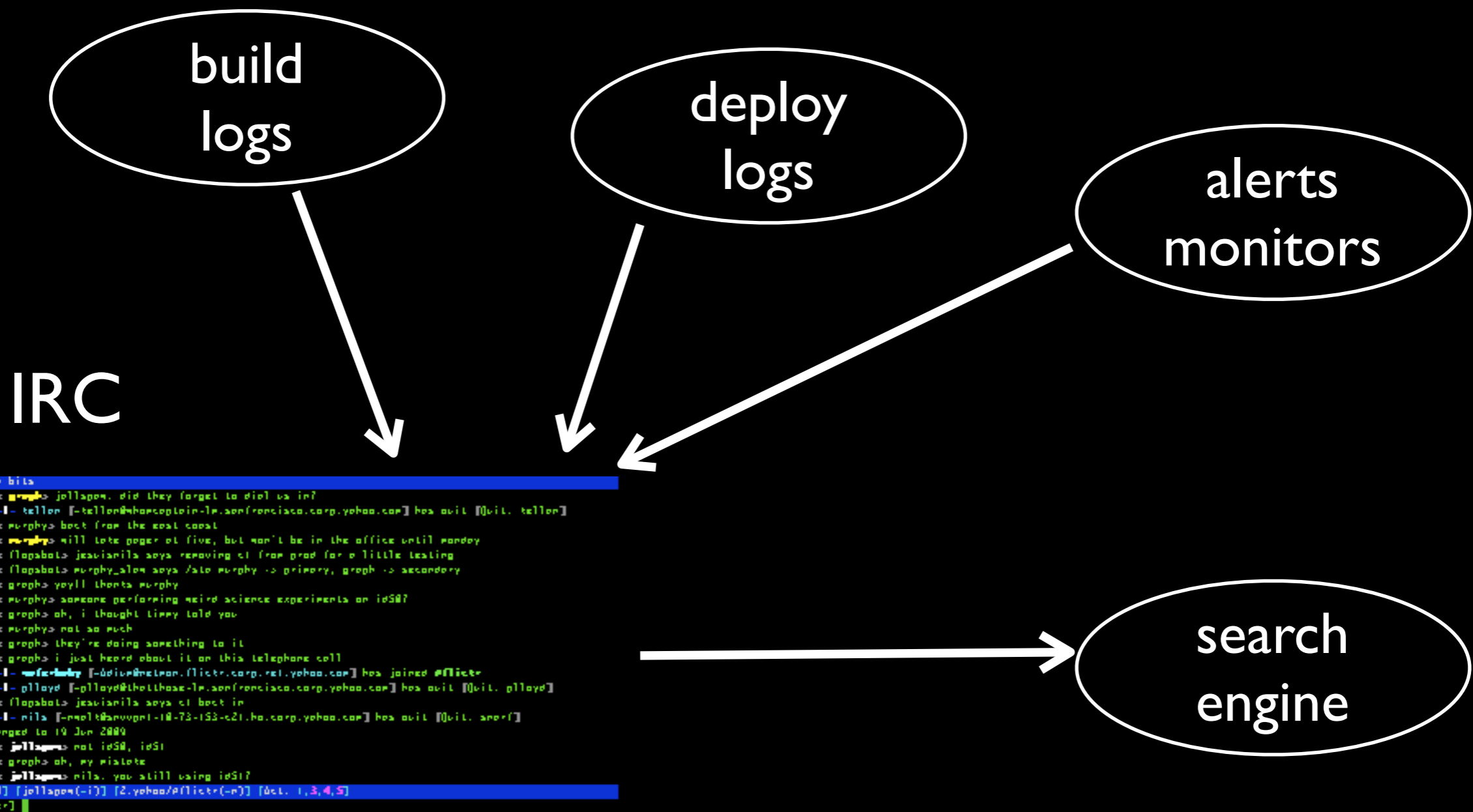


Cluster Report for Mon, 6 Apr 2009 21:56:45 +0000 Last Site Deploy: Mon, 6 Apr 2009 21:36:03 +0000



## 6. IRC and IM robots

# Dev, Ops, and Robots Having a conversation



Culture

# 1. Respect

If there is only one thing you do...



**Don't  
stereotype**  
(not all developers are lazy)

Respect other people's expertise,  
opinions and responsibilities



**COMMIT**  
~ **NO** ~  
**NUISANCE**

Don't just say "No"

Don't hide things



Developers: Talk to ops about the impact of your code:

- what metrics will change, and how?
- what are the risks?
- what are the signs that something is going wrong?
- what are the contingencies?

This means you need to **work this out** before talking to ops

## 2. Trust

Ops needs to trust dev to involve them on feature discussions

Dev needs to trust ops to discuss infrastructure changes

Everyone needs to trust that everyone else is doing their best for the business



**GENERAL INSTRUCTIONS**  
**BEHIND THIS DOOR**  
SACTS/2000/01

Shared runbooks & escalation plans

PERIPHERAL EQUIPMENT DISPLAY AND CONTROL

REEL RECORDER CONTROL

LOOP RECORDER CONTROL

INPUT-OUTPUT MODE STATUS AND CONTROL  
LOCATION

764

POWER CONTROL

MEMORY SELECTOR

MODE

TAPE RECORDER CONTROL

REPRODUCE LEVEL

RECORD LEVEL



SYSTEM DISPLAY AND CONTROL

ALARMS

GUIDANCE MODE STATUS AND CONTROL

DATA MEMORY MONITOR  
BASE VALUE

POWER OF TEN

COMPUTER WORD

WORD

55655623

TAPE RECORDER STATUS  
TAPE CONTROL ENABLE ALARMS

TDMG STATUS AND CONTROL



Provide knobs and levers

Ops: Be transparent,  
give devs access to systems





### 3. Healthy attitude about failure



Failure will happen

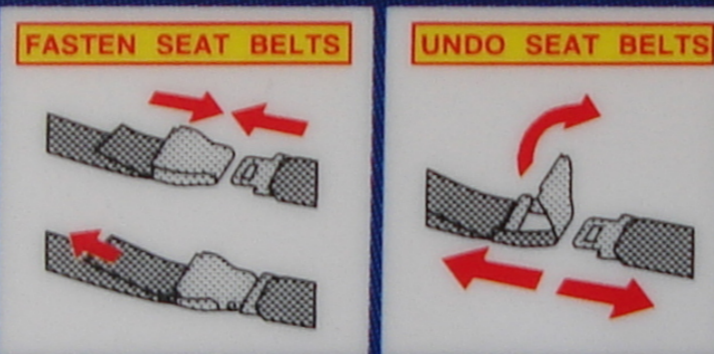
If you think you can prevent failure then you aren't developing your ability to respond

**RYANAIR.COM** Boeing 737-800 Safety Information  
*THE LOW FARES AIRLINE*

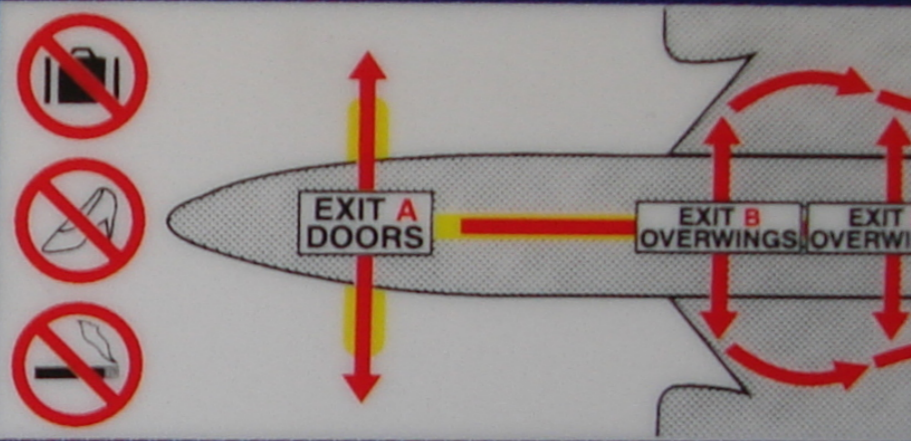
**EMERGENCY OXYGEN**



**SEAT BELTS**



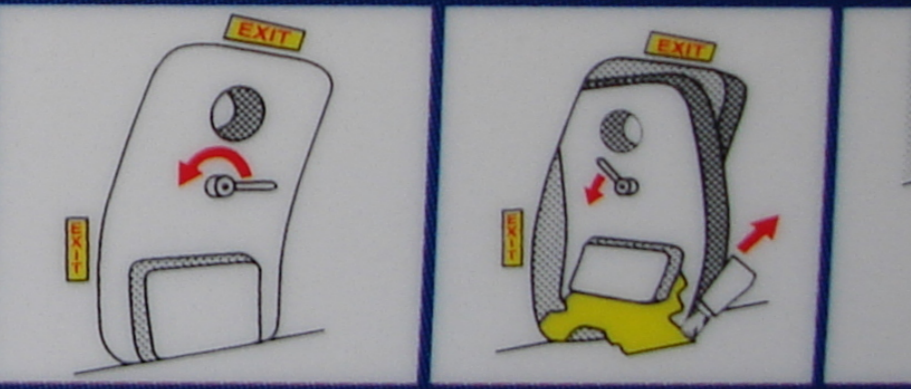
**EMERGENCY EXITS**



**LIFE VESTS**



**EXIT A DOORS**



**EMERGENCY LANDING**



**EXIT B - OVERWINGS**



RA815 REV. 0



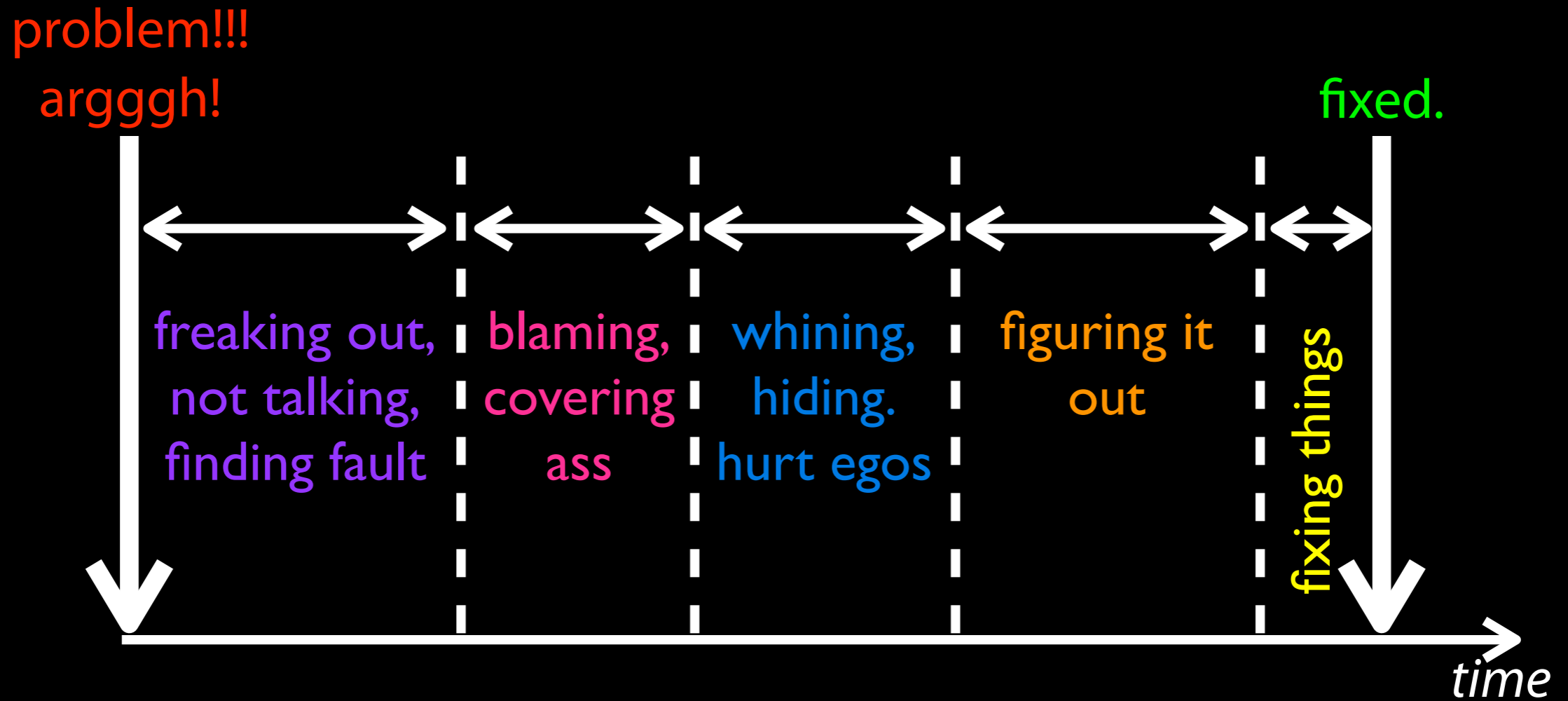
# Fire drills



# 4. Avoiding Blame

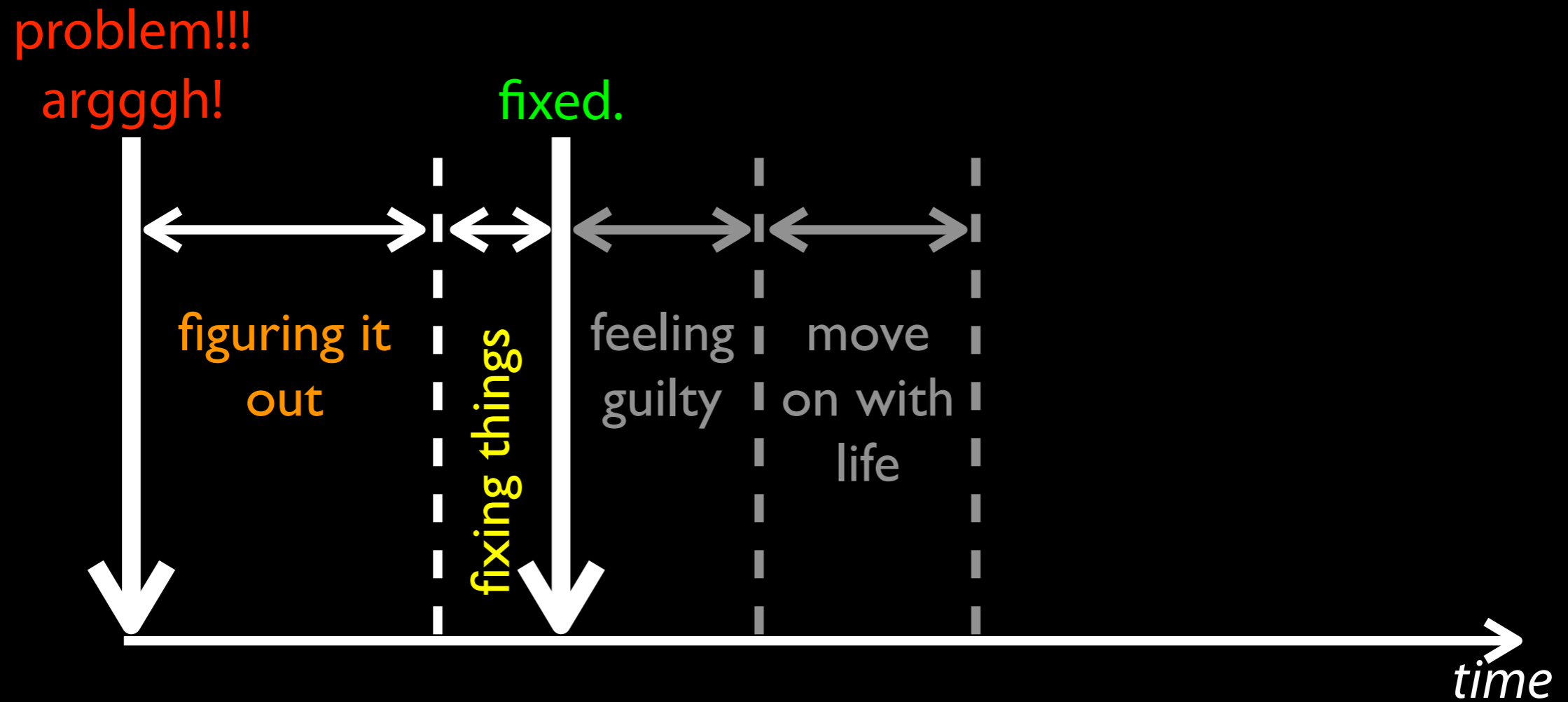
No fingerpointing

# Fingerpointyness





# Being productive





Developers: Remember that someone else will probably get woken up when your code breaks

Ops: provide  
constructive  
feedback on  
current aches  
and pains



1. Automated infrastructure
2. Shared version control
3. One step build and deploy
4. Feature flags
5. Shared metrics
6. IRC and IM robots

1. Respect
2. Trust
3. Healthy attitude about failure
4. Avoiding Blame

This is **not easy**

You could just carry on shouting at each other...

(Thank you)