# DISBAND THE DEPLOYMENT ARMY

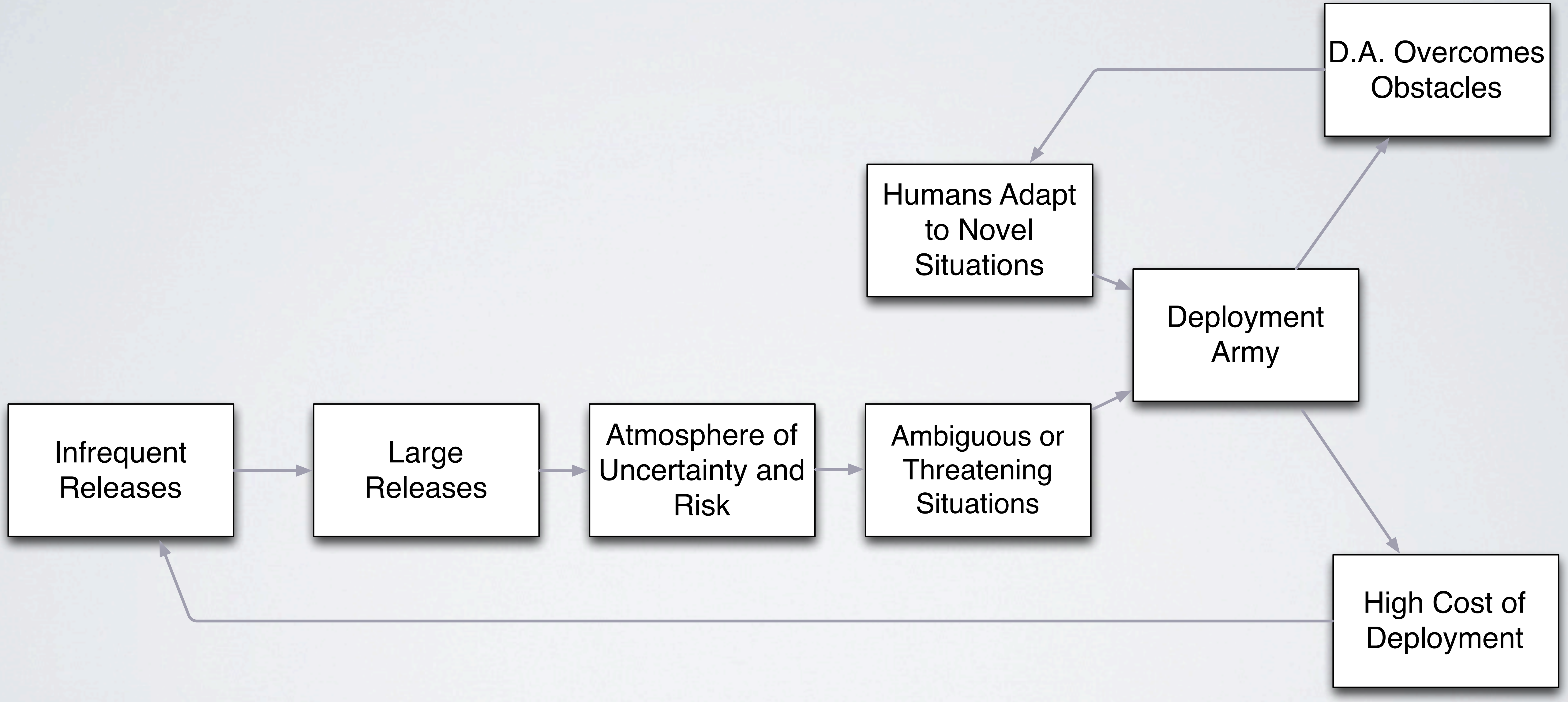
**Michael Nygard**

***Relevance, Inc.***

# Deployment Army

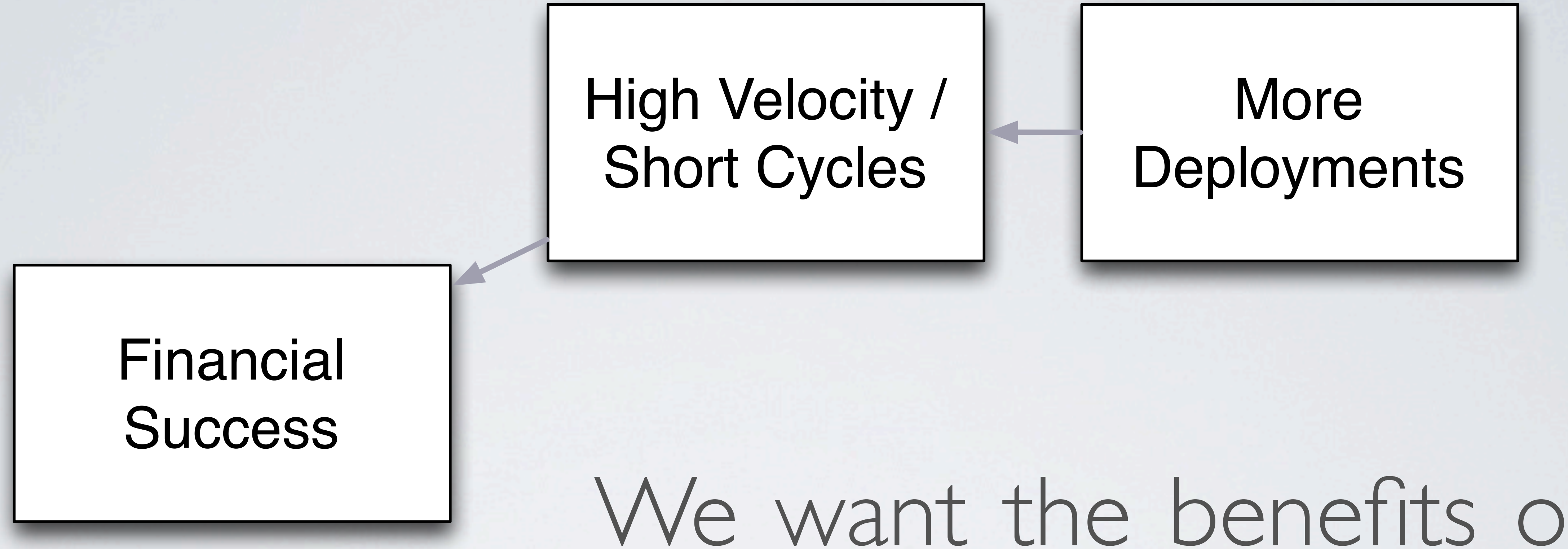
# Army Deployments

(Hint: It's An Antipattern)


D.A. Overcomes Obstacles
Humans Adapt to Novel Situations
Deployment Army
Infrequent Releases
Large Releases
Atmosphere of Uncertainty and Risk
Ambiguous or Threatening Situations
High Cost of Deployment

Financial Success

We want the benefits of agile development and continuous integration all the way to production.



High Velocity / Short Cycles

More Deployments

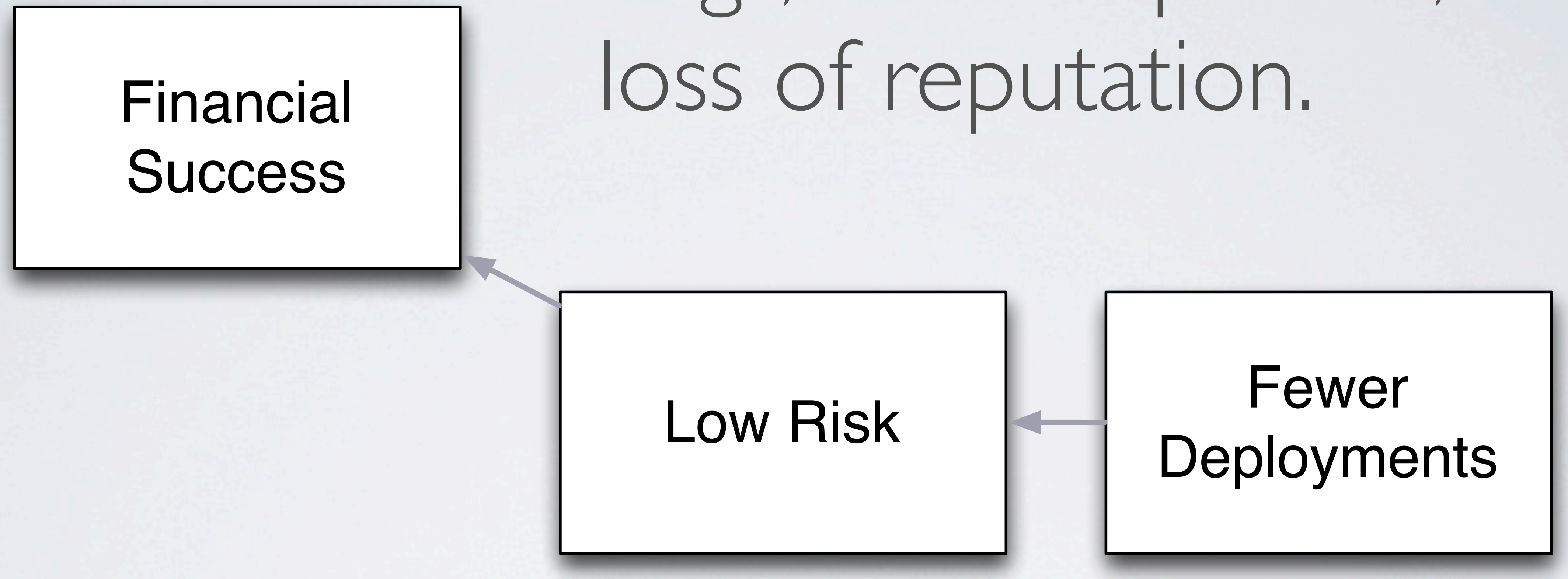
Financial Success

We want the benefits of agile development and continuous integration all the way to production.

Financial Success

We must avoid financial losses due to downtime, bugs, noncompliance, and loss of reputation.

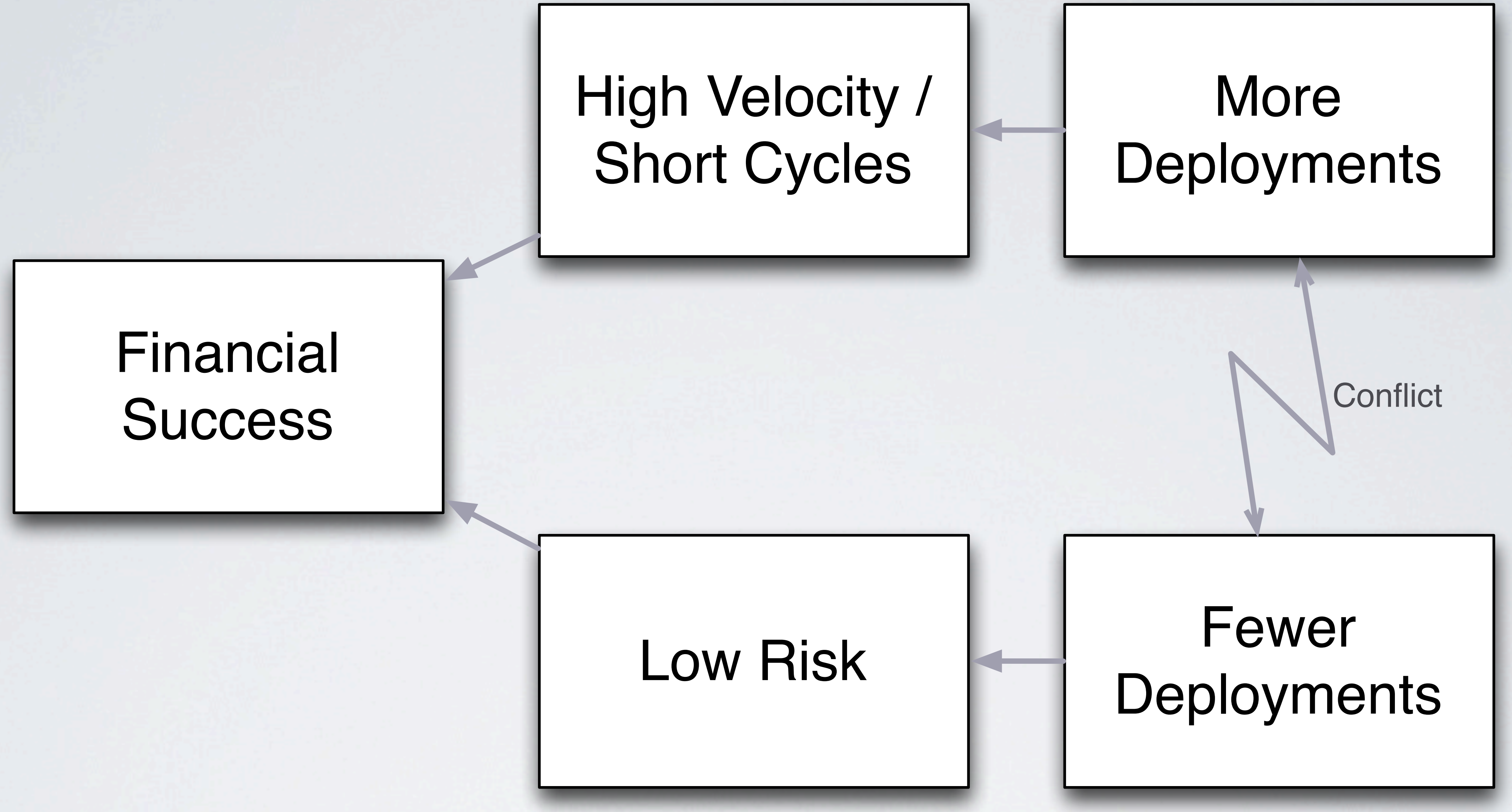
We must avoid financial losses due to downtime, bugs, noncompliance, and loss of reputation.



Financial Success

Low Risk

Fewer Deployments



One goal.
Two conflicting demands.

# Understanding Risk

# Understanding Risk

Expected losses from undesirable events.

# Exposure

Annual Loss Expectancy (ALE)

$$\text{Loss} = N_{\text{events}} \times P_{\text{error}} \times C_{\text{event}}$$

# Exposure Example: Bug In Checkout

| | | |
|---|---|---|
| $P_{error}$ | Occurance | 1 time in $10^8$ |
| $N_{events}$ | Checkouts / Year | $5.25 \times 10^8$ |
| $C_{event}$ | Average Lost Order | €25 |
| Loss | Total losses per annum | €131.40 |

# Categories Of Risk

# Categories Of Risk

1. Compliance Risk

2. Technical Risk

# Categories Of Risk

1. **Compliance Risk**

2. Technical Risk

# Compliance Risk

Certification

Regulatory approval

Third party testing

Commonly seen in:

- Banking / finance
- Health care
- Aviation
- Consumer Electronics

# Managing Compliance Risk

Deliver continuously to certification environment.

Rapidly detect noncompliant changes

Reduce time between validation cycles

# Categories Of Risk

1. Compliance Risk

**2. Technical Risk**

# Managing Technical Risk

# Managing Technical Risk

$$Loss = N_{events} \times P_{error} \times C_{event}$$

$N_{events}$ will increase.

We can decrease $P_{error}$ and $C_{event}$.

# Reducing Risk Exposure

| | Batch | Continuous |
|---|---|---|
| $N_{events}$ | Low | **High** |
| $P_{error}$ | High | ? |
| $C_{event}$ | High | ? |

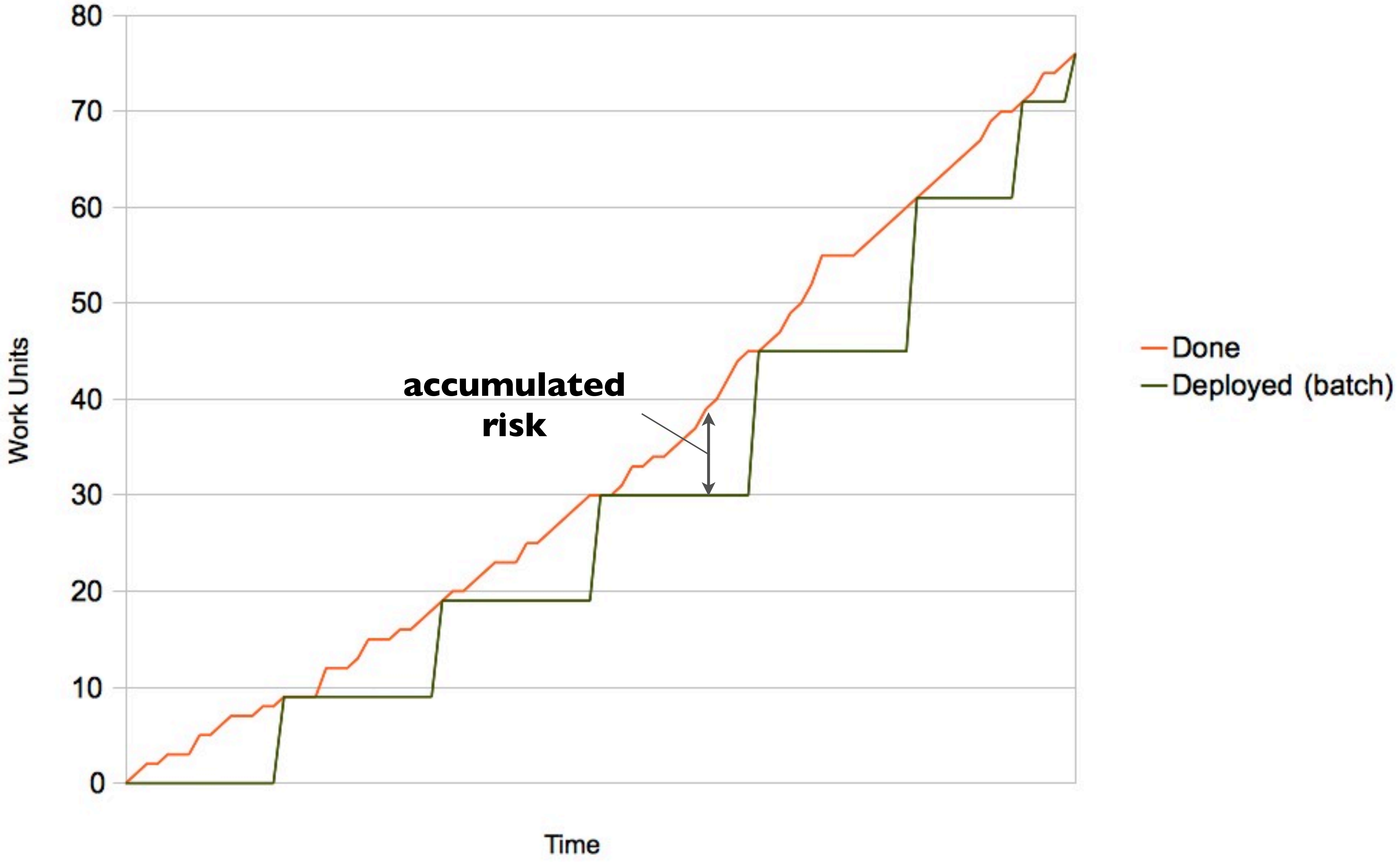
# Sources Of $P_{error}$

Defects in code

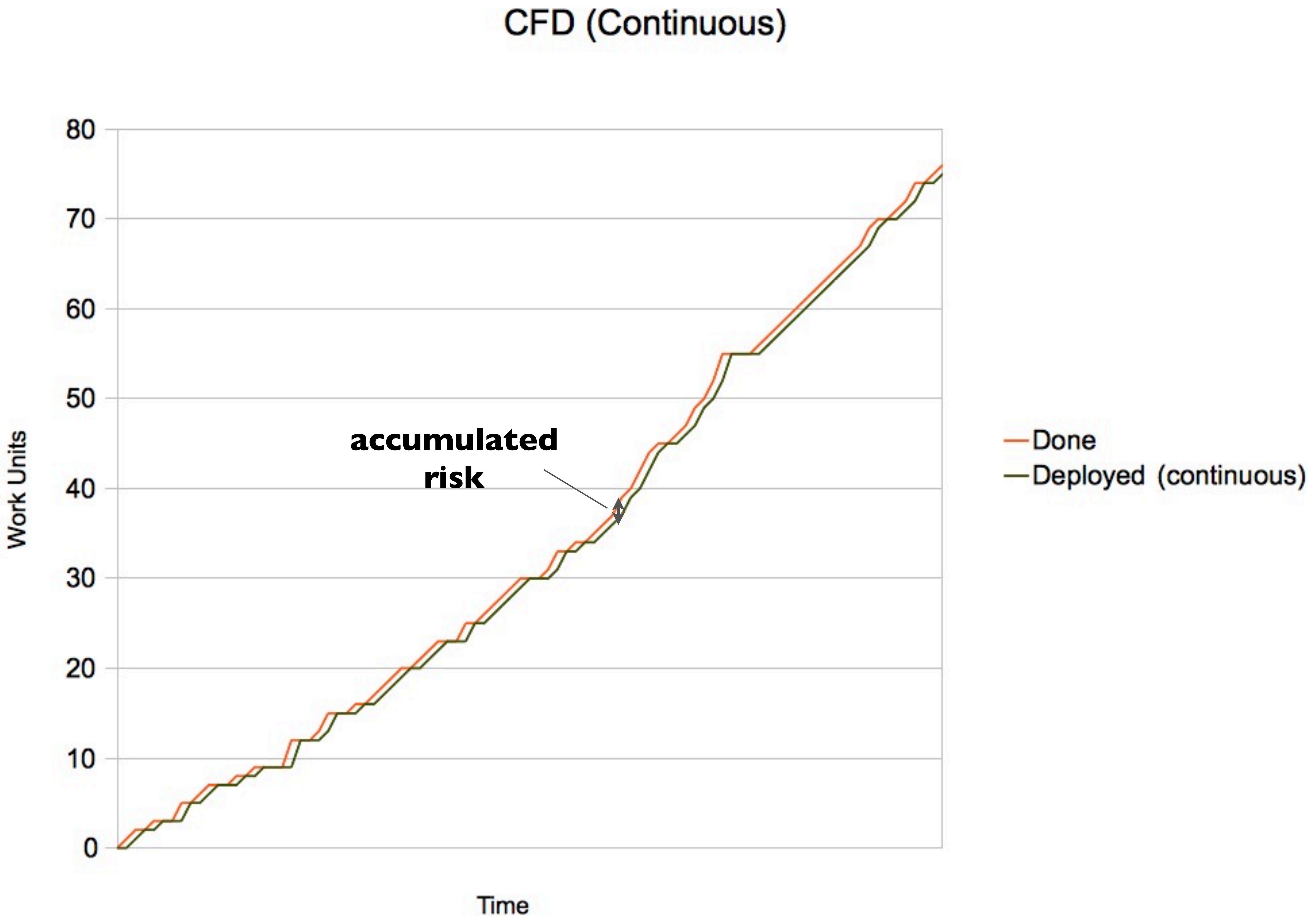
Errors in assembly or packaging

Errors executing changes

## CFD (Batched)

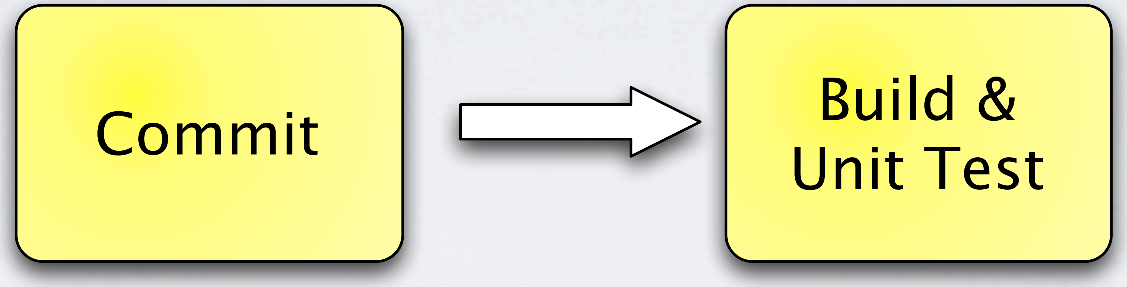


**accumulated risk**

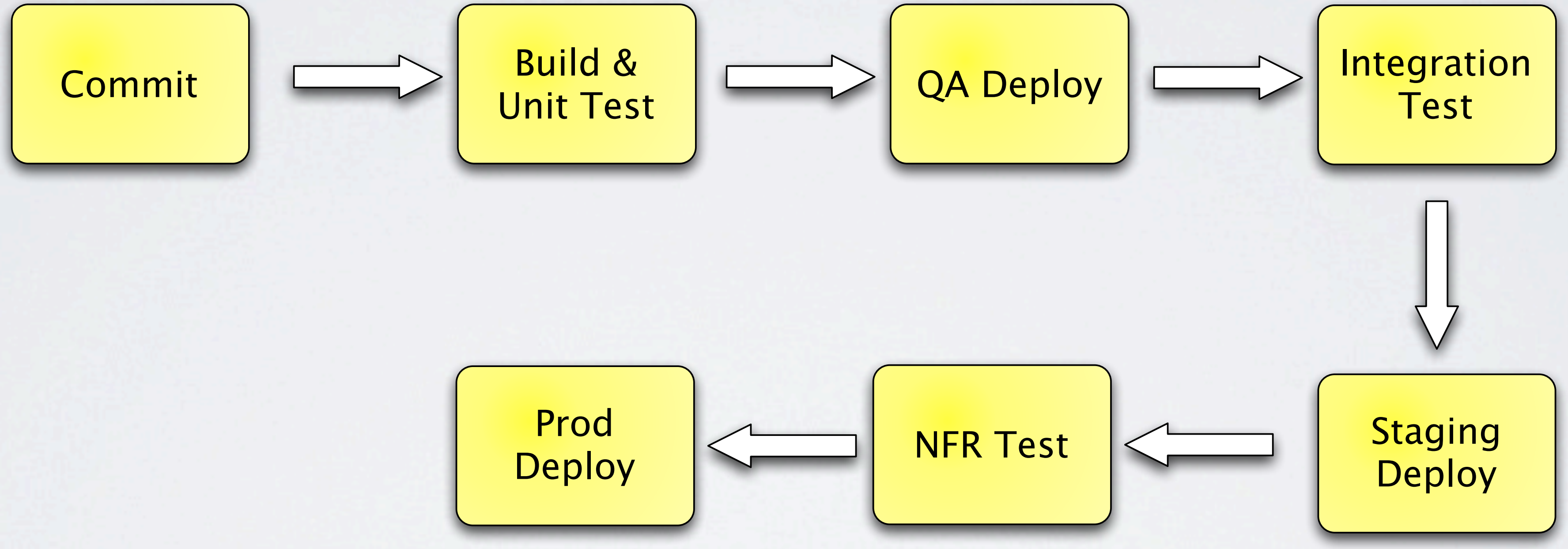

CFD (Continuous)

accumulated risk

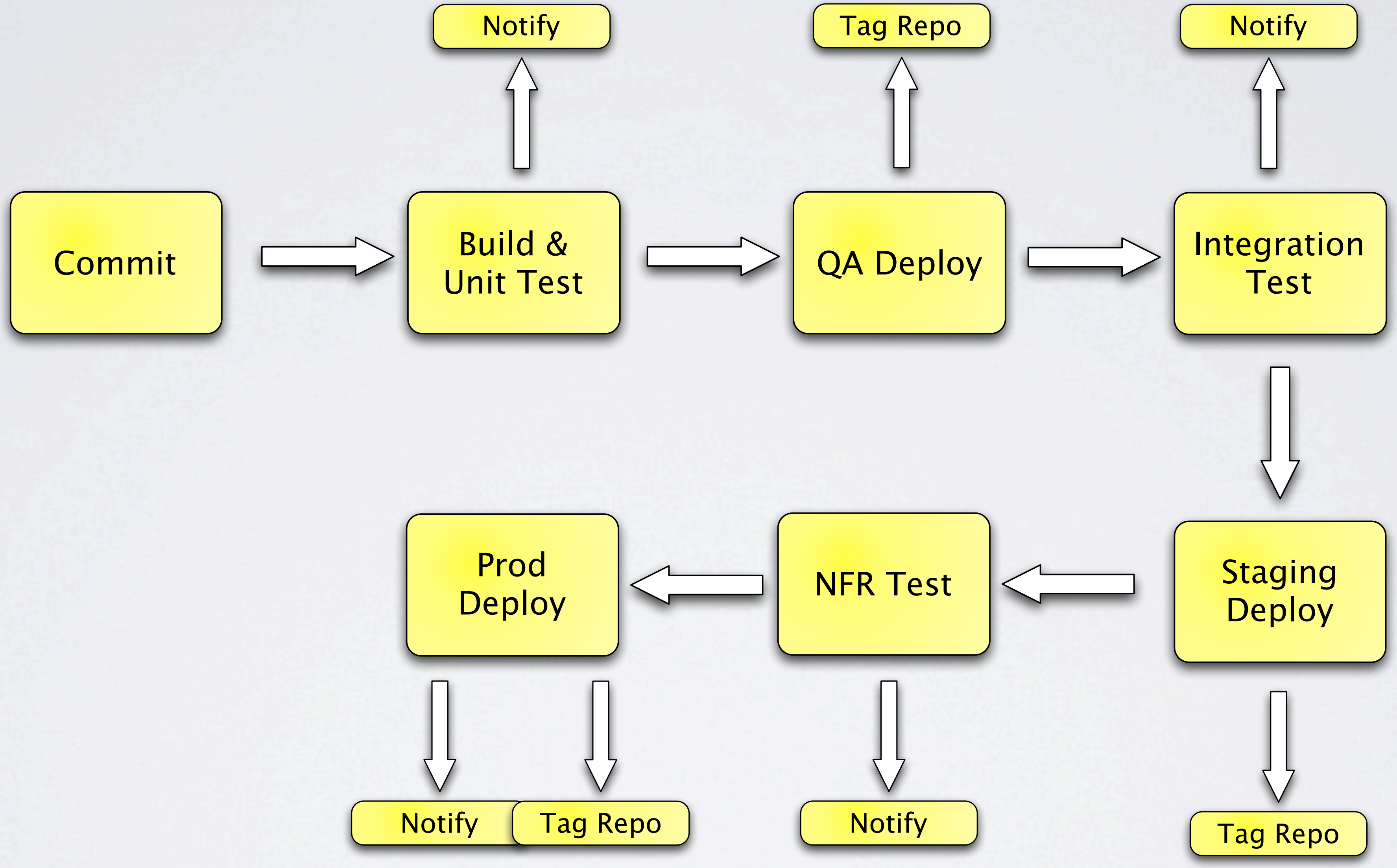
# Build Pipelines



# Build Pipelines



# Build Pipelines



# Reducing Build & Assembly Errors

Fast tests

Clean build servers from VCS

Clean build applications from VCS

Promote binaries, not sources

Deploy the same way everywhere

# Essential Practices

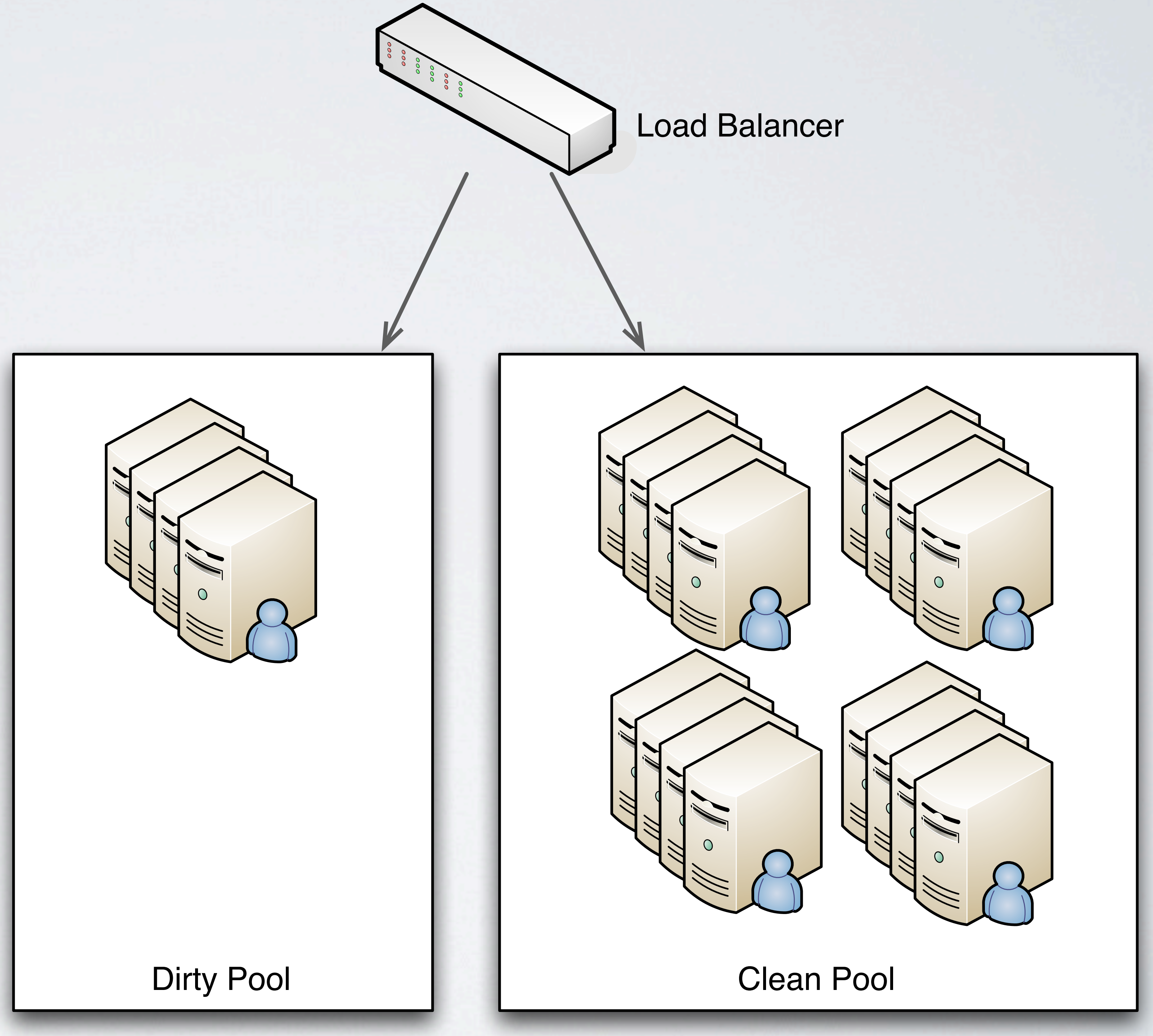
- Fast tests
- Never commit new work on a broken build
- Fail the build on slow tests
- Fail the build on violations: architecture, coding standards
- Involve Ops in creating deployment scripts
- Deploy from head of trunk

# Environment Requirements

- Package repository
- Tag the repo
- Logging
- Metrics

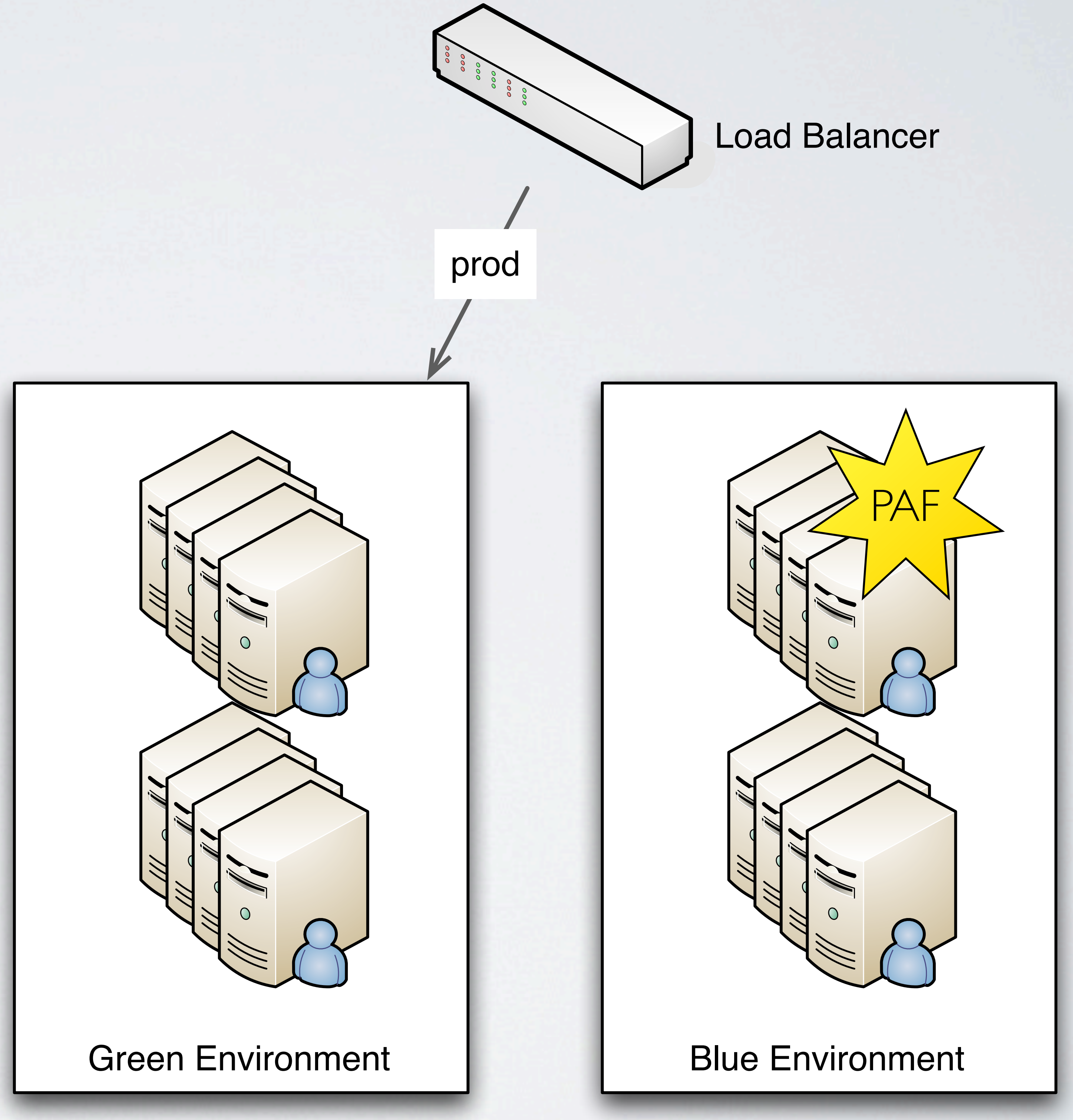
# Making Deployments Safe

Canary Releasing



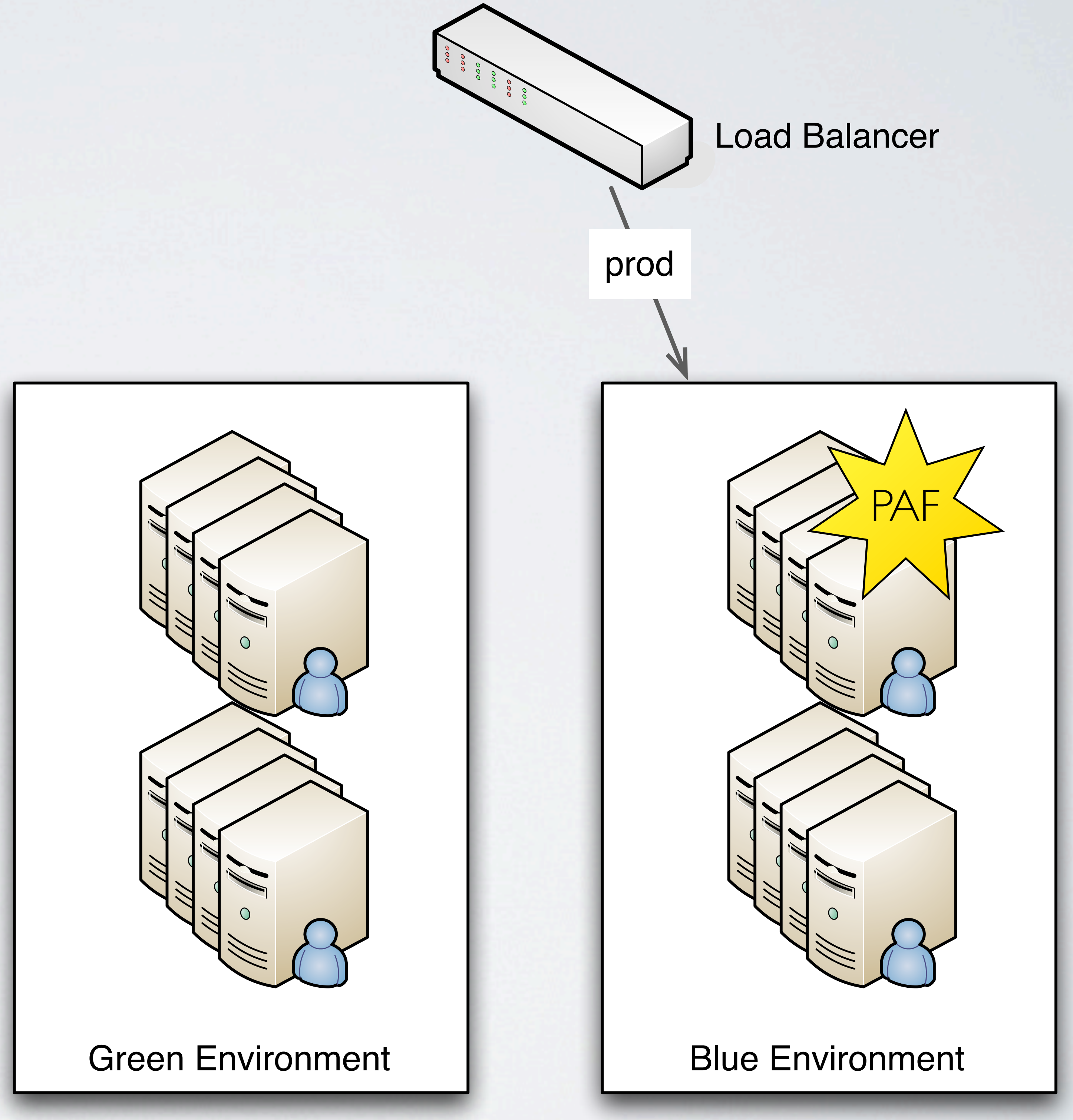
# Making Deployments Safe

Blue/Green Deployments

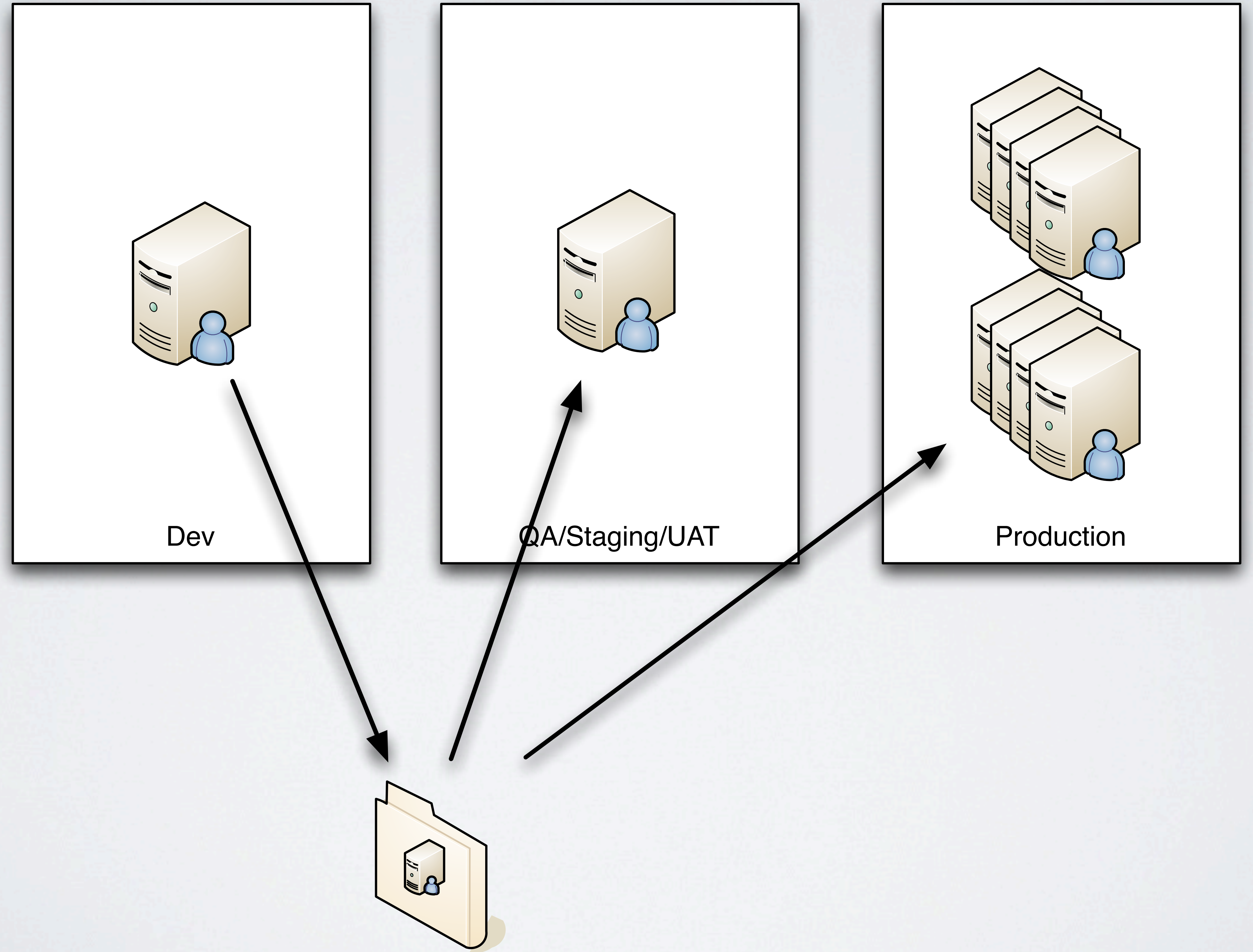


# Making Deployments Safe

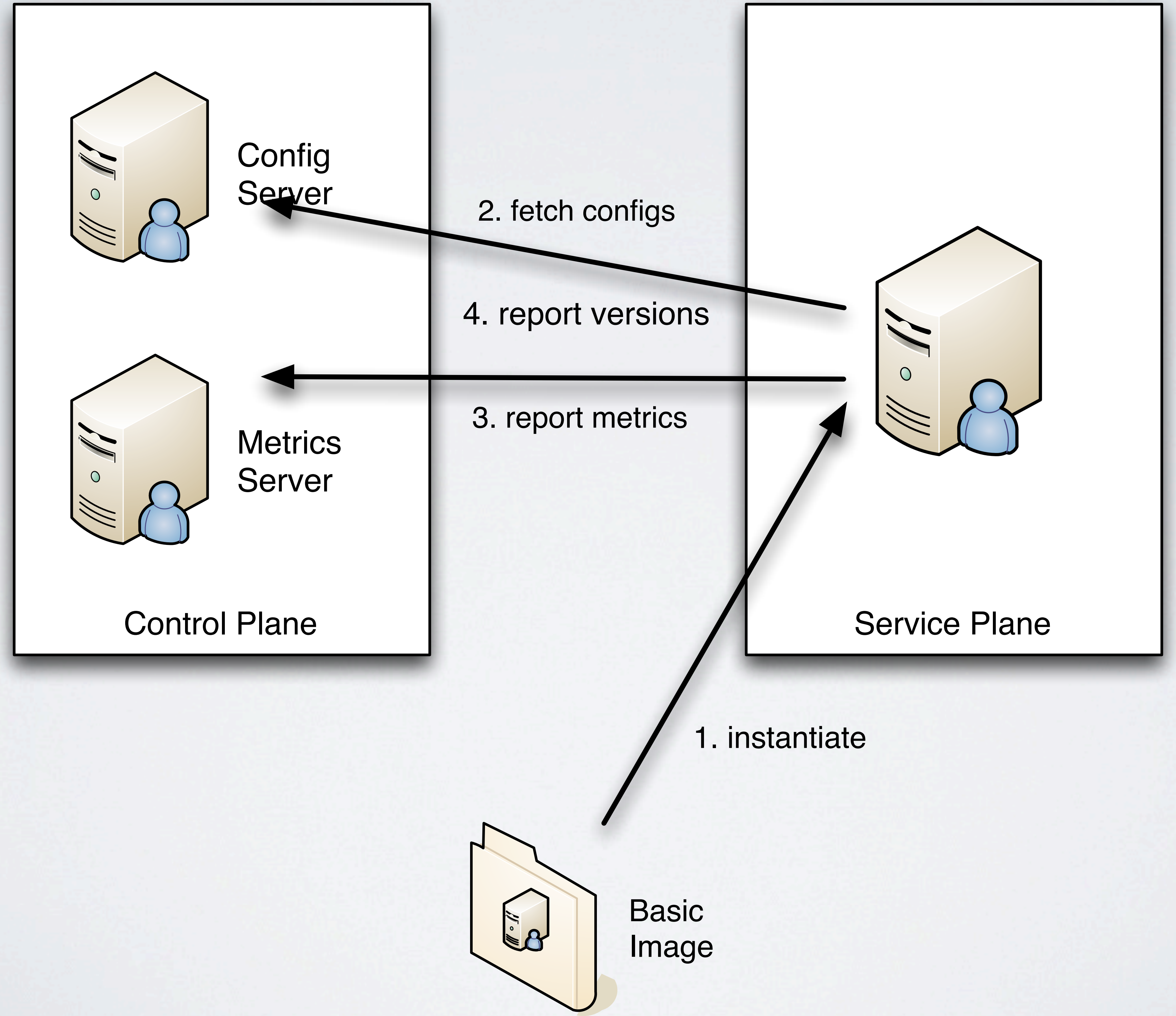
## Blue/Green Deployments



# Making Deployments Safe



# Making Deployments Predictable



# Making Deployments Safe

All of these *require* zero downtime deployments.

# Zero Downtime Deployments

- Database migrations
- Schema shims
- Versioned identifiers for assets
- Protocol versioning
- Endpoint versioning
- Decoupled architecture

# Reducing Risk Exposure

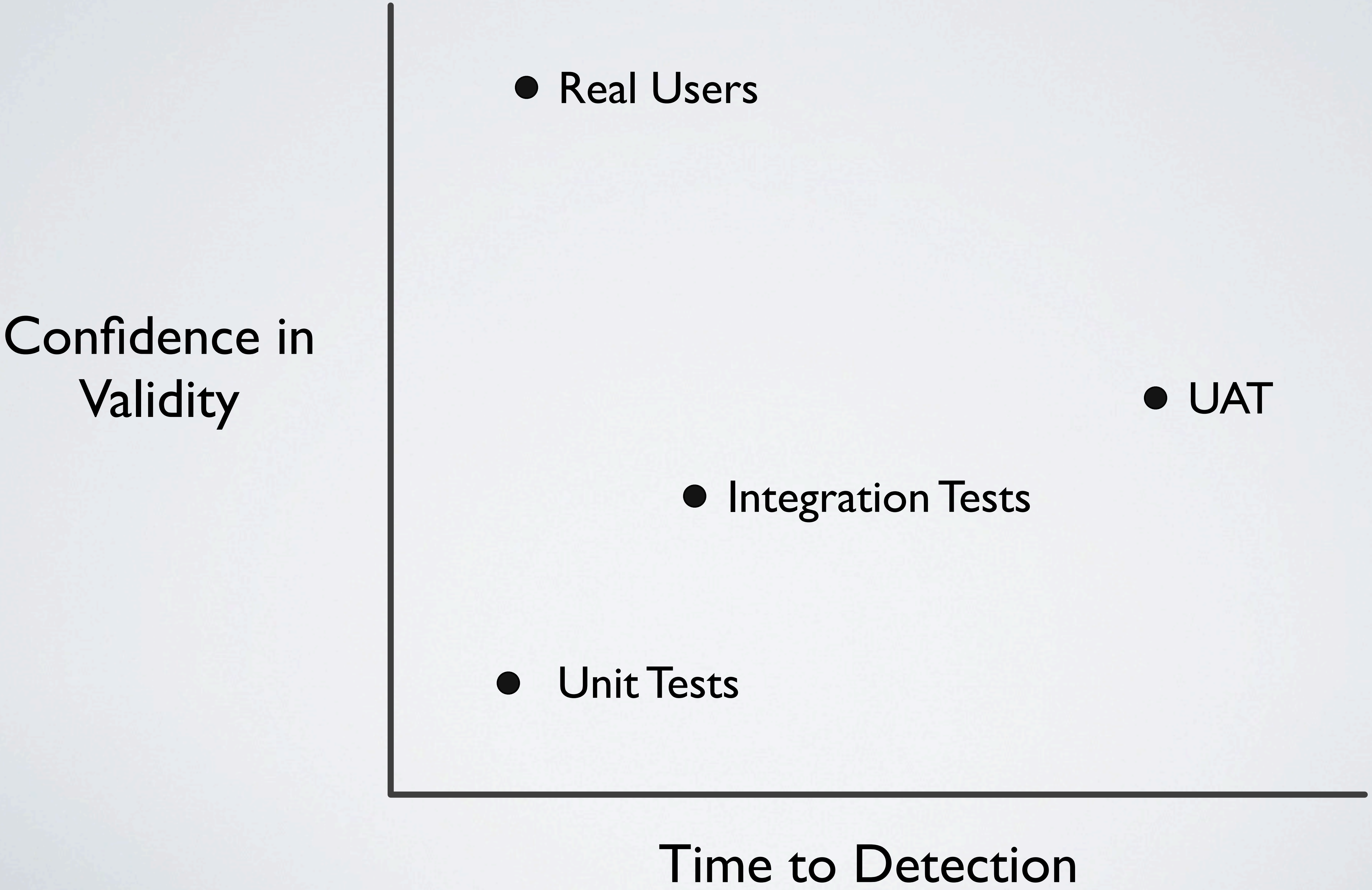
| | Batch | Continuous |
|---|---|---|
| $N_{events}$ | Low | High |
| $P_{error}$ | High | **Low** |
| $C_{event}$ | High | ? |

# Minimizing $C_{event}$

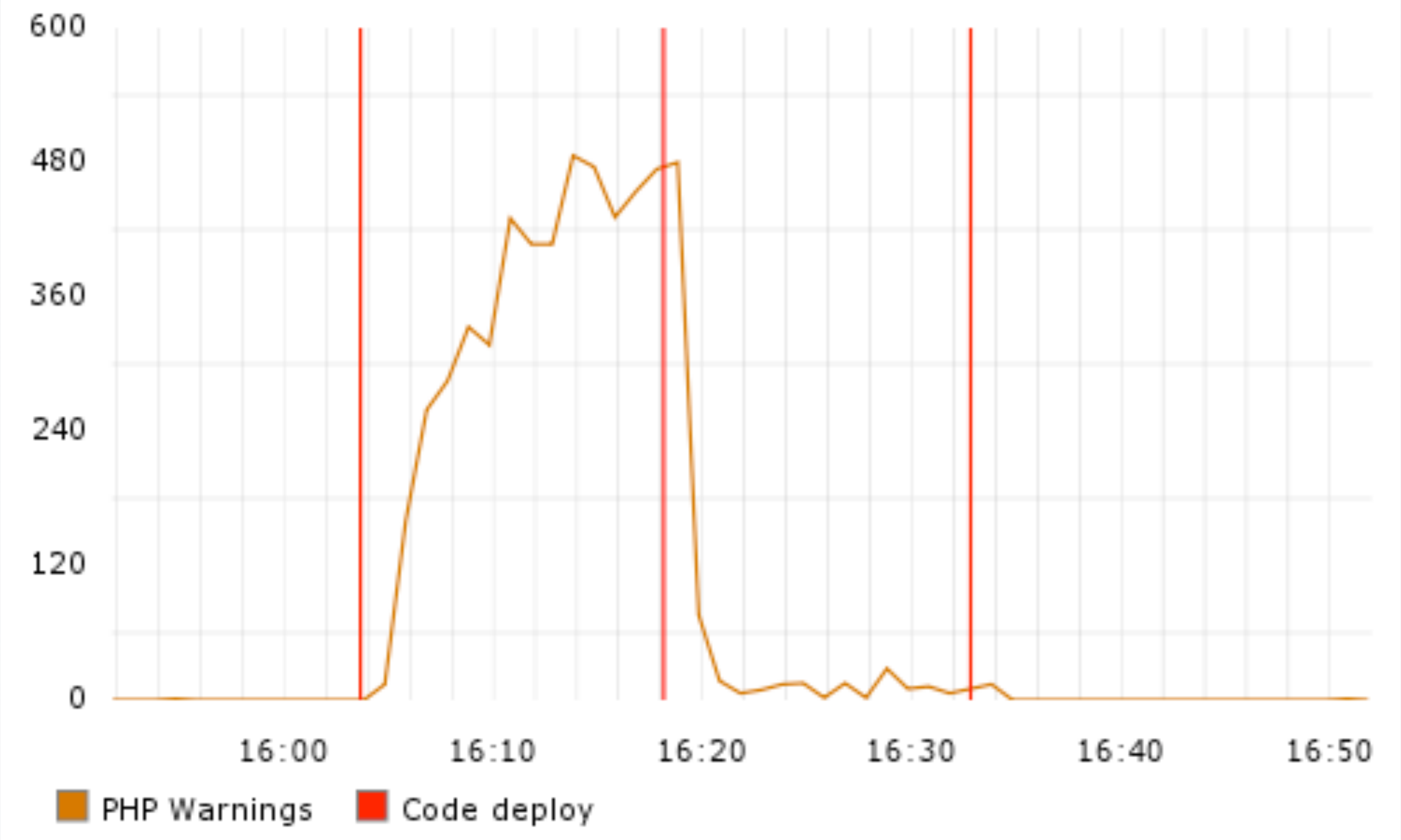
1. **Reduce time to detect (MTTD)**
2. Reduce time to correct (MTTR)
3. Reduce scope of impact

These are all things that batched deployment does badly.

# Reduce Mean Time To Detection

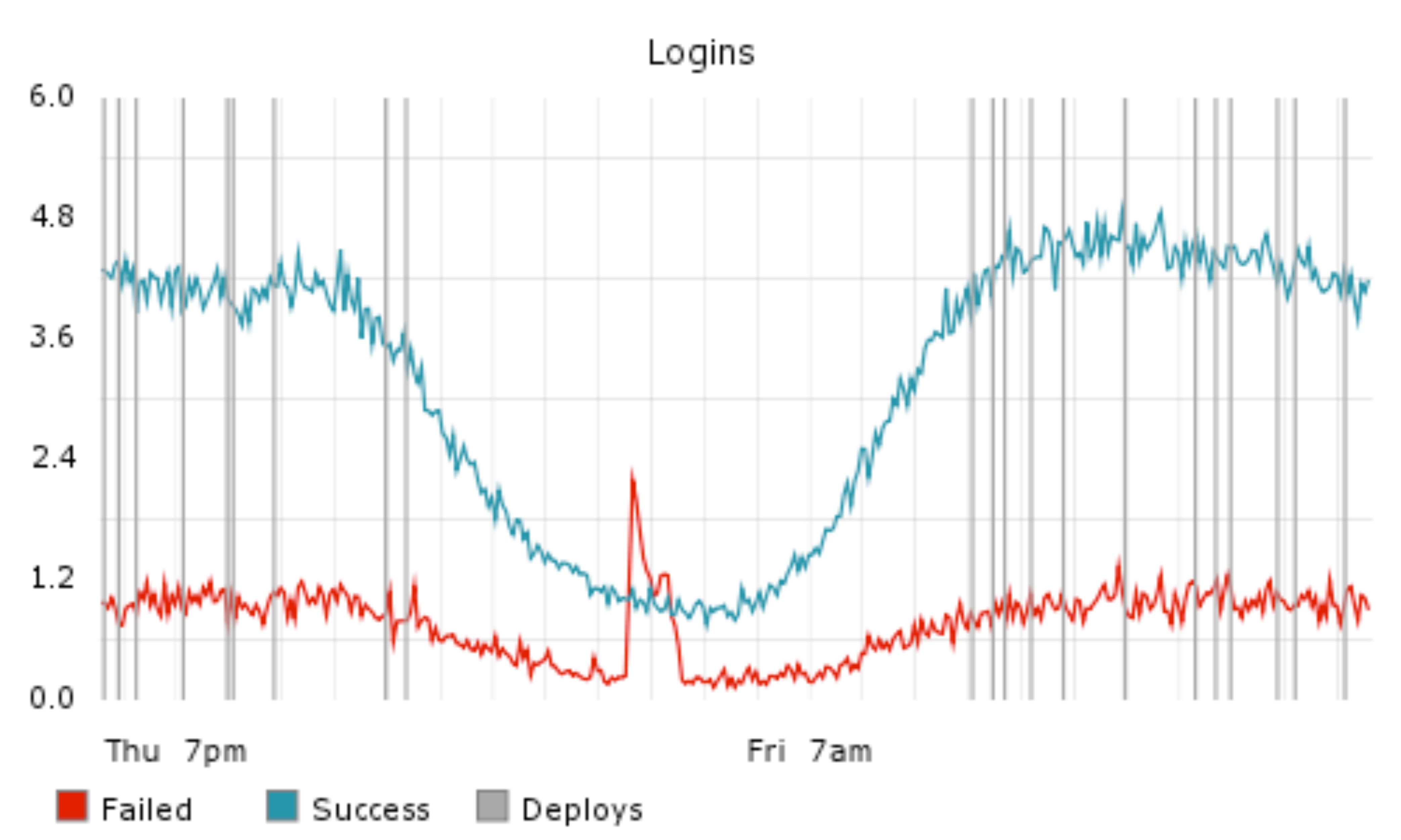


# Cult Of Charts



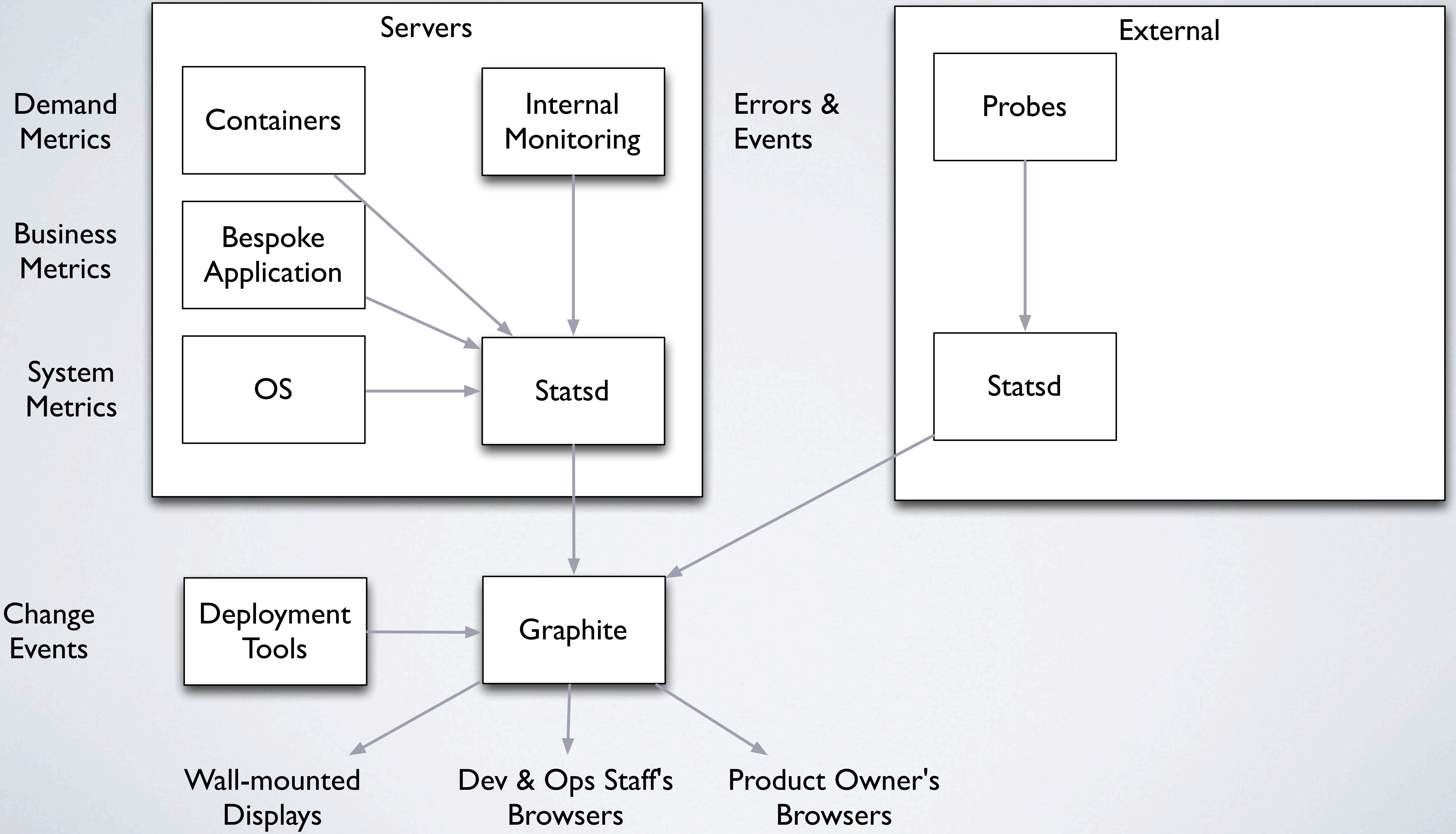
Mike Brittain, "Tracking Every Release"
http://codeascraft.etsy.com/2010/12/08/track-every-release/

# Cult Of Charts



Ian Mapless, "Measure Anything, Measure Everything"
http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/

# Charting Everything



# Minimizing $C_{event}$

1. Reduce time to detect (MTTD)
2. **Reduce time to correct (MTTR)**
3. Reduce scope of impact

# Factors In MTTR

1. Determine the problem
2. Fix the problem
3. Deploy the fix

# Minimizing $C_{event}$

1. Reduce time to detect (MTTD)
2. Reduce time to correct (MTTR)
3. **Reduce scope of impact**

# Scope Of Impact

How many users are exposed to the error?

Split testing

Dark launch

Feature flags

Remote control

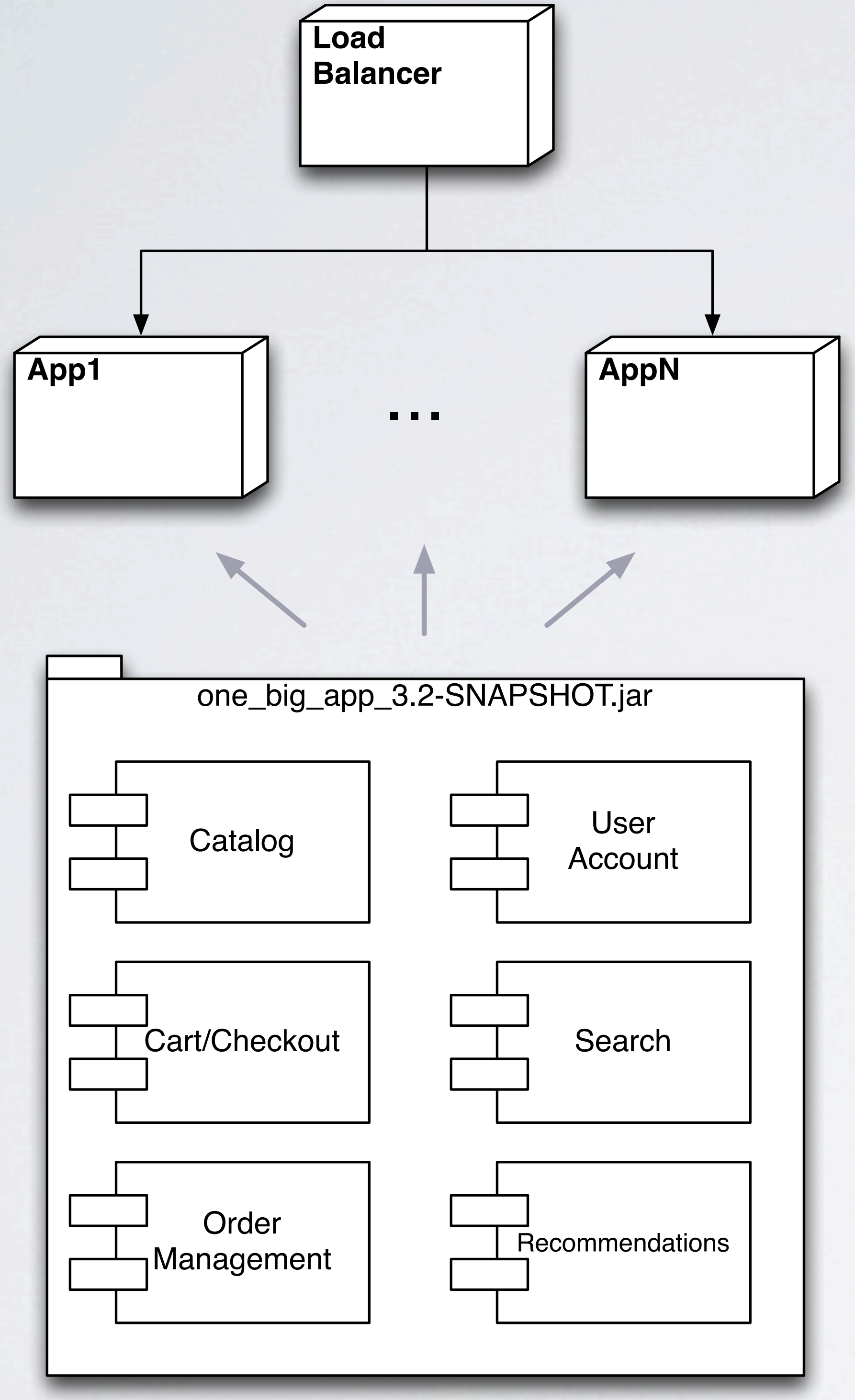
# Scope Of Impact

How far can the error propagate in the system?

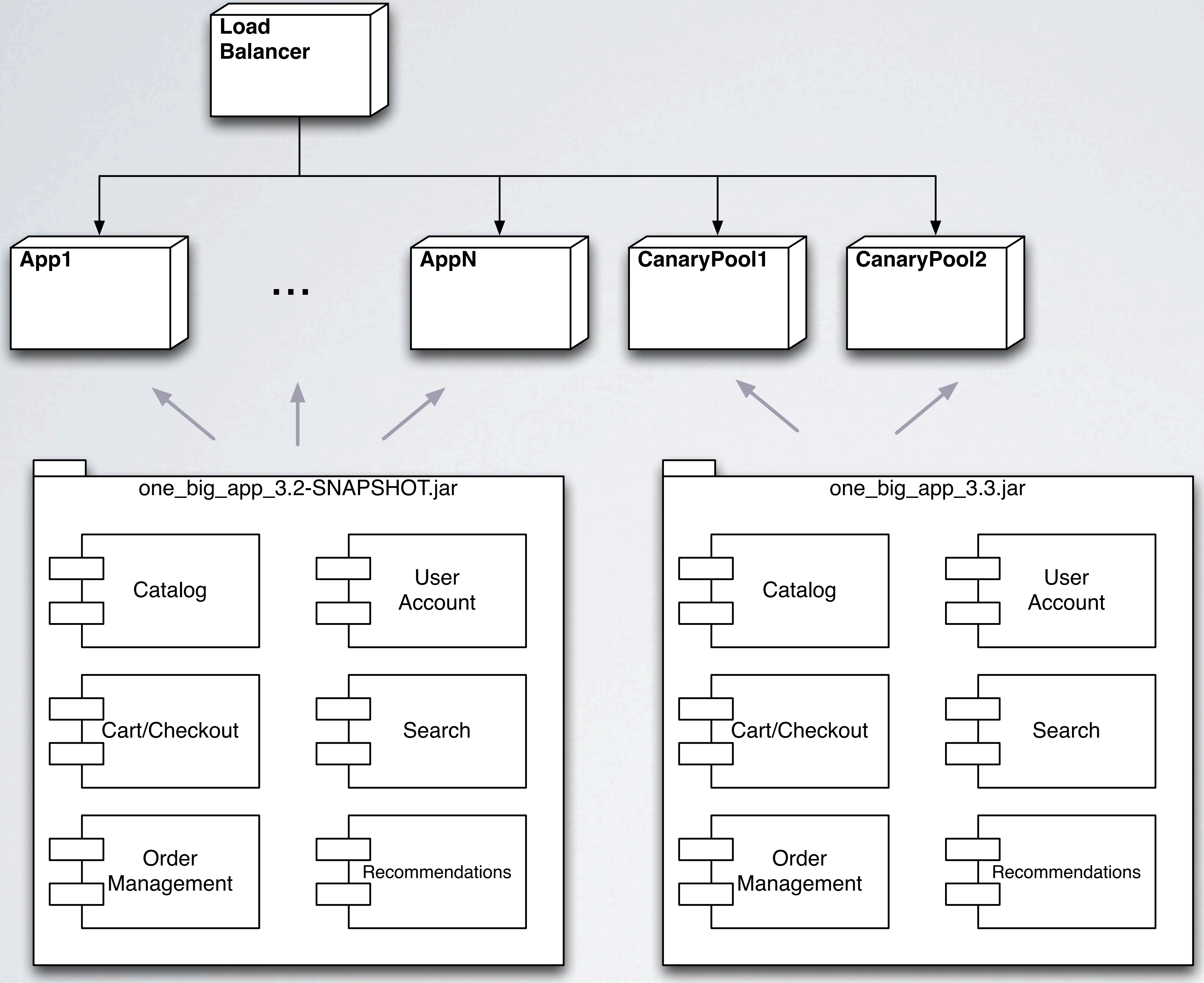
Decoupled architecture

Bulkheads

Circuit breakers

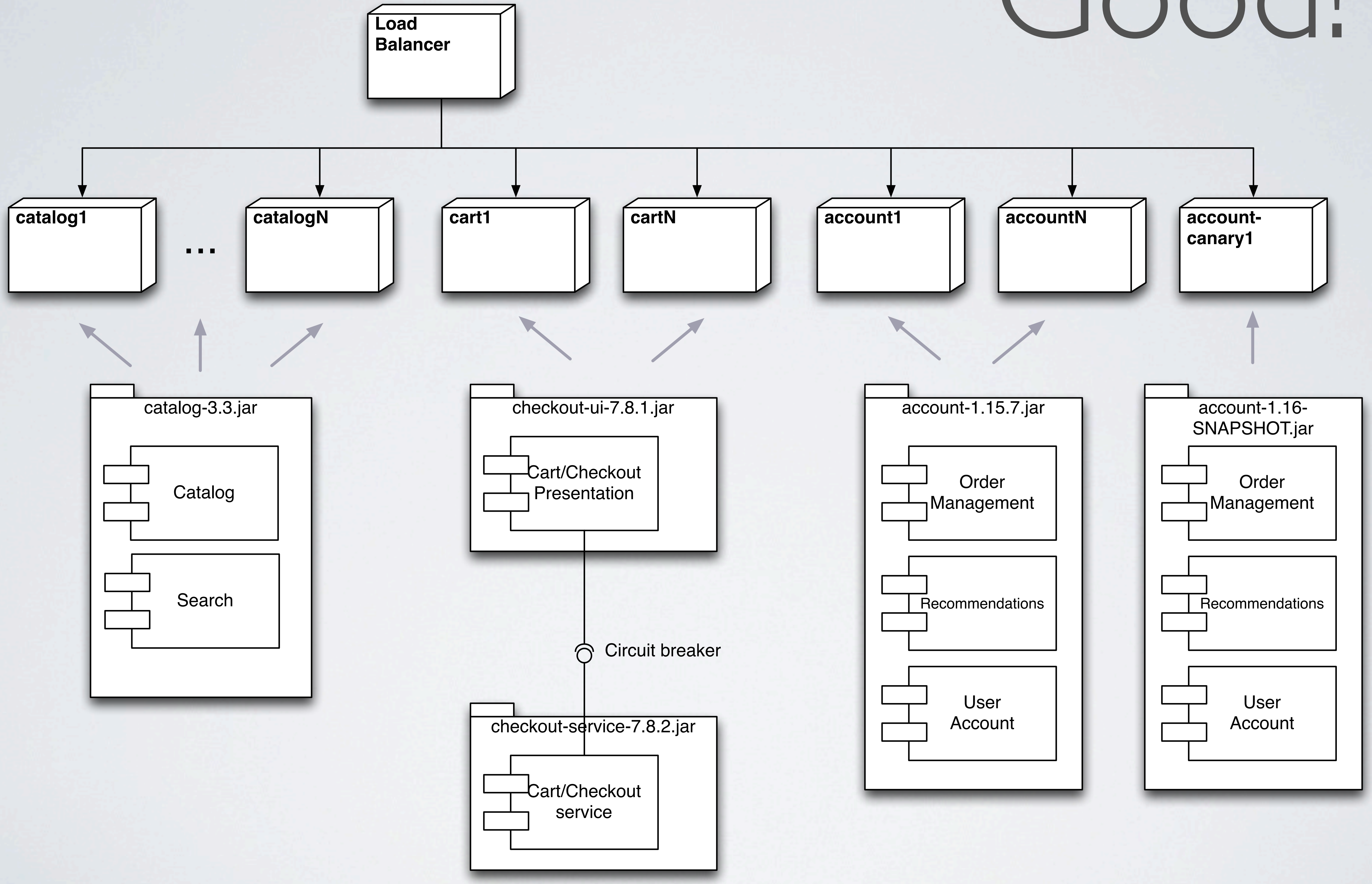

Load Balancer
App1
...
AppN
one_big_app_3.2-SNAPSHOT.jar
Catalog
User Account
Cart/Checkout
Search
Order Management
Recommendations

Doomed.


Load Balancer
App1
...
AppN
CanaryPool1
CanaryPool2
one_big_app_3.2-SNAPSHOT.jar
Catalog
User Account
Cart/Checkout
Search
Order Management
Recommendations
one_big_app_3.3.jar
Catalog
User Account
Cart/Checkout
Search
Order Management
Recommendations

# Better.

# Good!


Load Balancer
catalog1
...
catalogN
cart1
cartN
account1
accountN
account-canary1
catalog-3.3.jar
Catalog
Search
checkout-ui-7.8.1.jar
Cart/Checkout Presentation
Circuit breaker
checkout-service-7.8.2.jar
Cart/Checkout service
account-1.15.7.jar
Order Management
Recommendations
User Account
account-1.16-SNAPSHOT.jar
Order Management
Recommendations
User Account

(BTW: This is where PHP developers get to brag.)

# Reducing Risk Exposure

| | Batch | Continuous |
|---|---|---|
| $N_{events}$ | Low | High |
| $P_{error}$ | High | Low |
| $C_{event}$ | High | **Low** |

# Unsolved Problems

# Unsolved Problems

- Managing library dependencies.
- Managing service & protocol dependencies.
- Interfacing with ITIL processes.

# Questions To Think About

# Questions To Think About

If you could rebuild whole environments over a coffee break, how would your processes change?

# Questions To Think About

What's the smallest incremental change you could make toward CD?

# Questions To Think About

What could you automate that you're doing manually today?

# Questions To Think About

What monitors could add to improve your visibility?

# Questions To Think About

What could you do to remove organizational barriers that prevent you from doing CD today?

# DISBAND THE DEPLOYMENT ARMY

Michael T. Nygard
Relevance, Inc.
www.thinkrelevance.com

michael.nygard@thinkrelevance.com @mtnygard