AWS re:Invent

NOV. 29 – DEC. 3, 2021 | LAS VEGAS, NV

DOP309

Amazon Builders' Library: Operational Excellence at Amazon

David Yanacek (he/him)
Sr. Principal Engineer
AWS Serverless

Key takeaways

We treat ops as an investment, not a cost

We align incentives for operational ownership with builders

We examine our operations together, regularly

What this talk isn't

Region build

Agenda: 5 stories

The ops win

The retrospective

The ops meeting

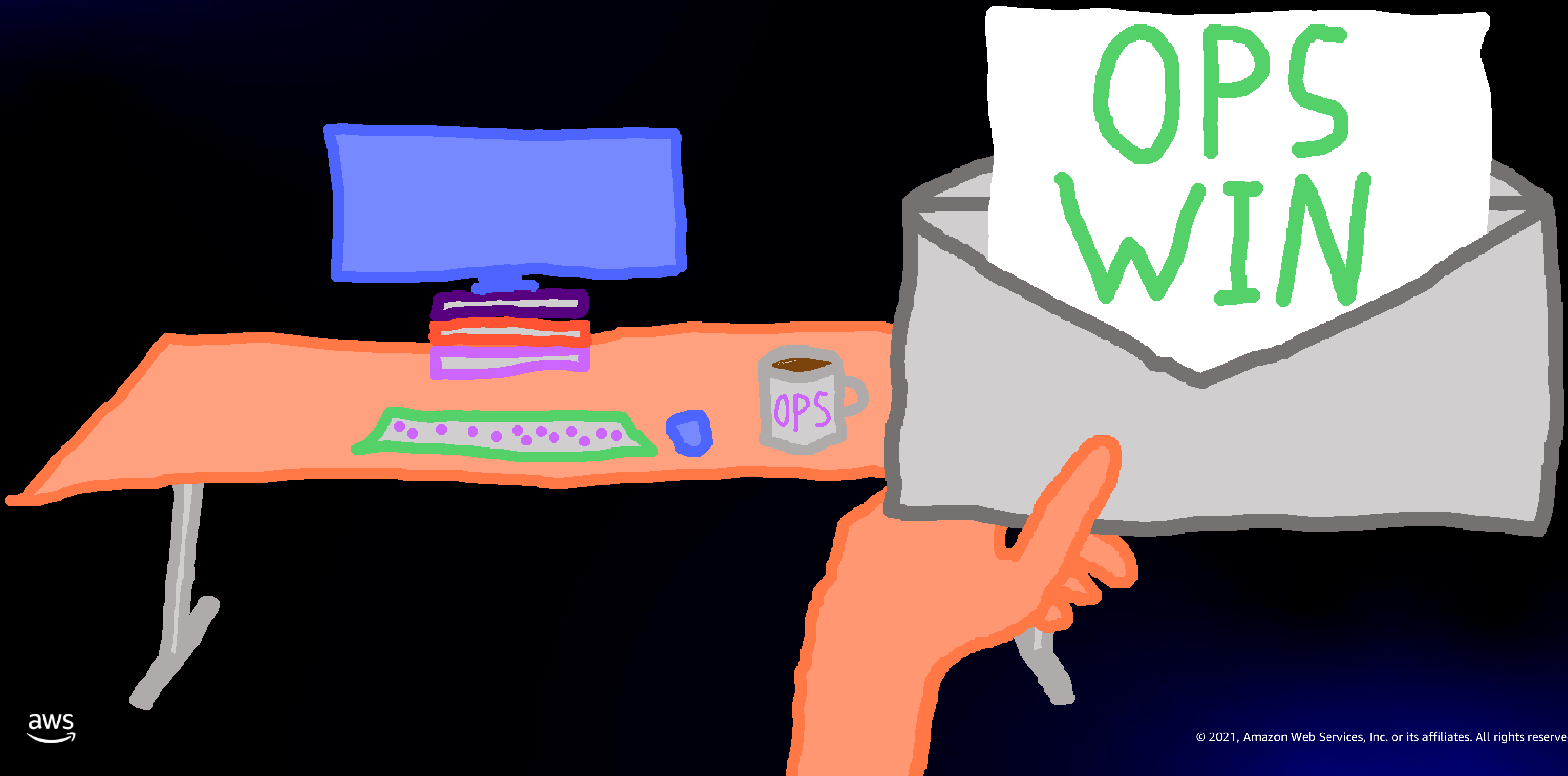
The investment in agility

The architectural choice

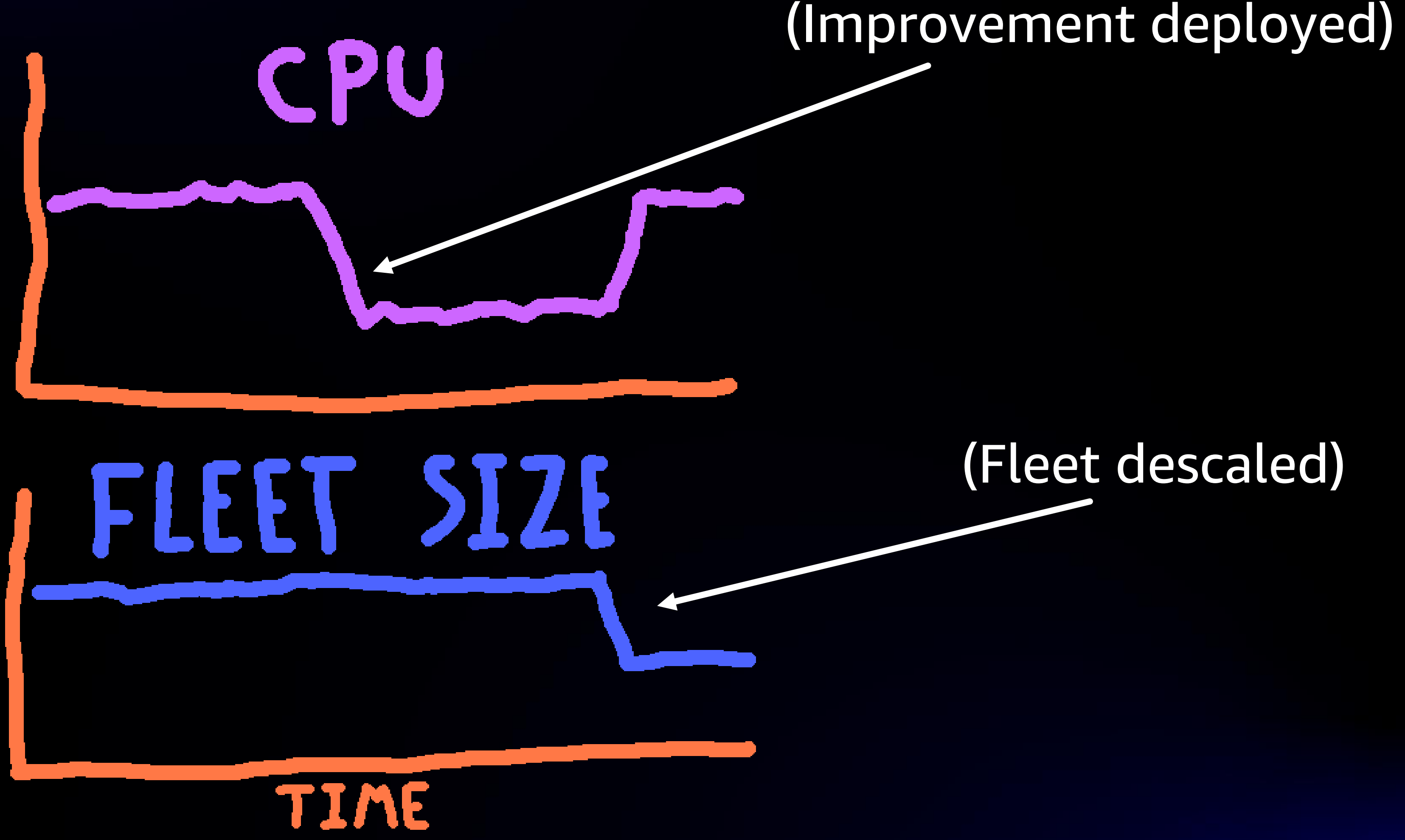
Story 1: The ops win

OPS

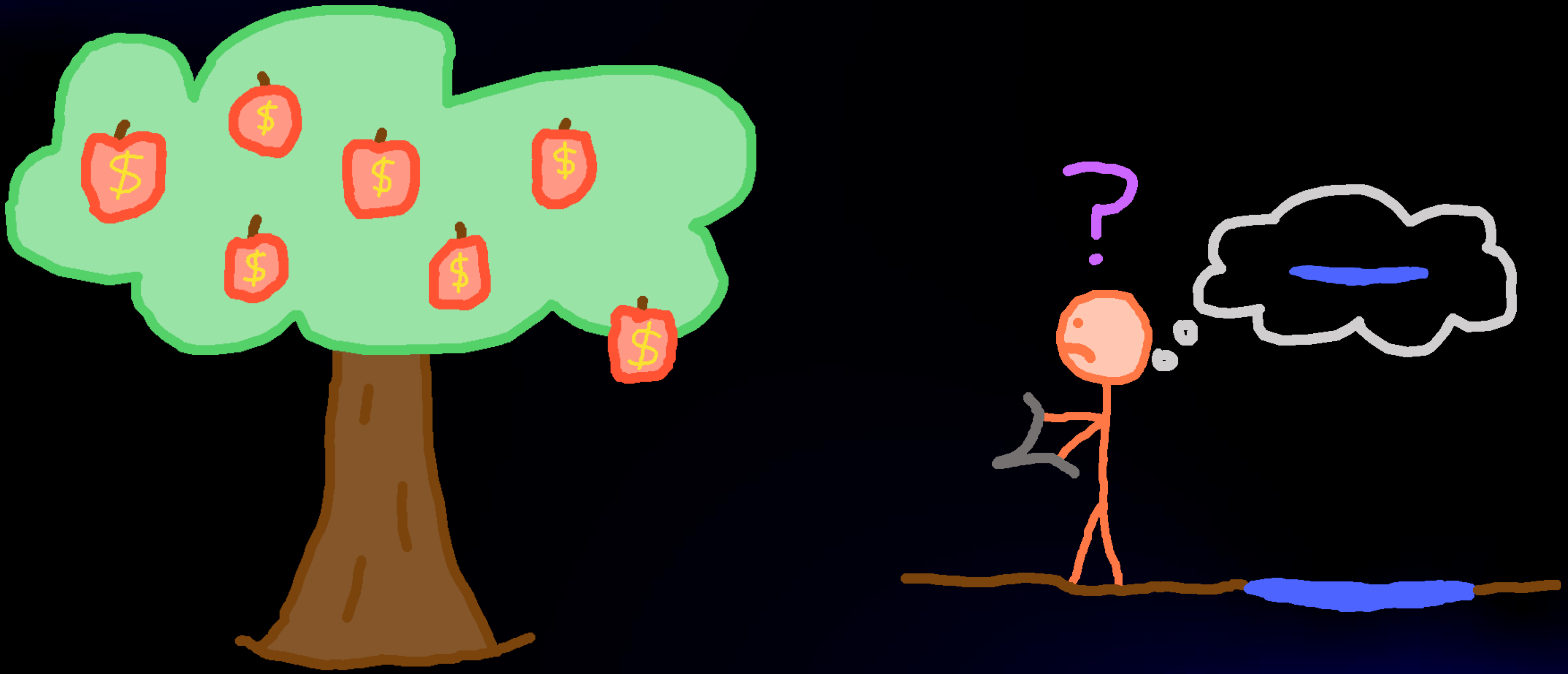
OPS


OPS WIN
OPS

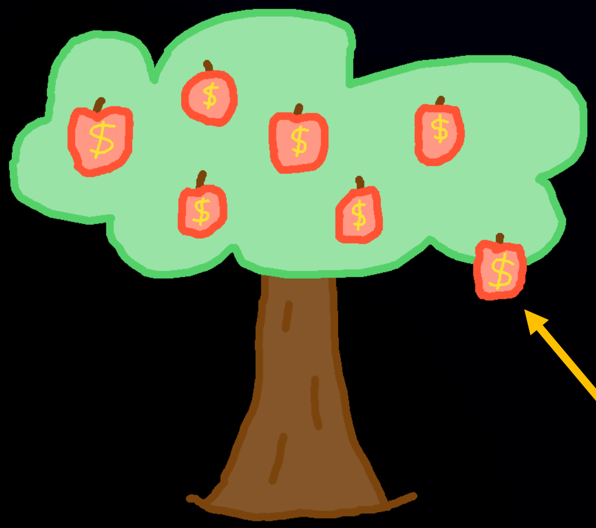
Deploying an efficiency improvement

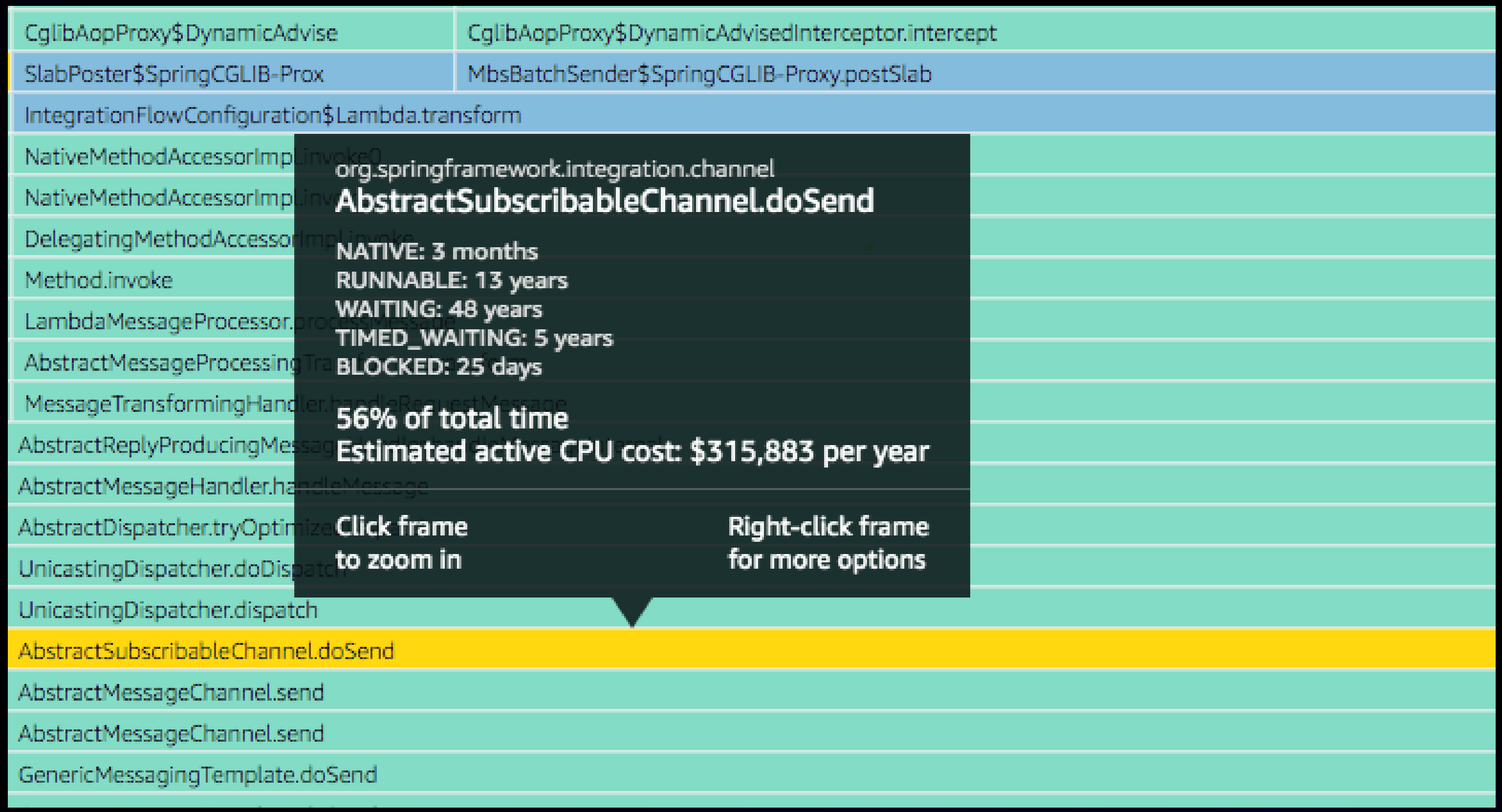


Find easy optimizations in code



Using a profiler to find low-hanging fruit





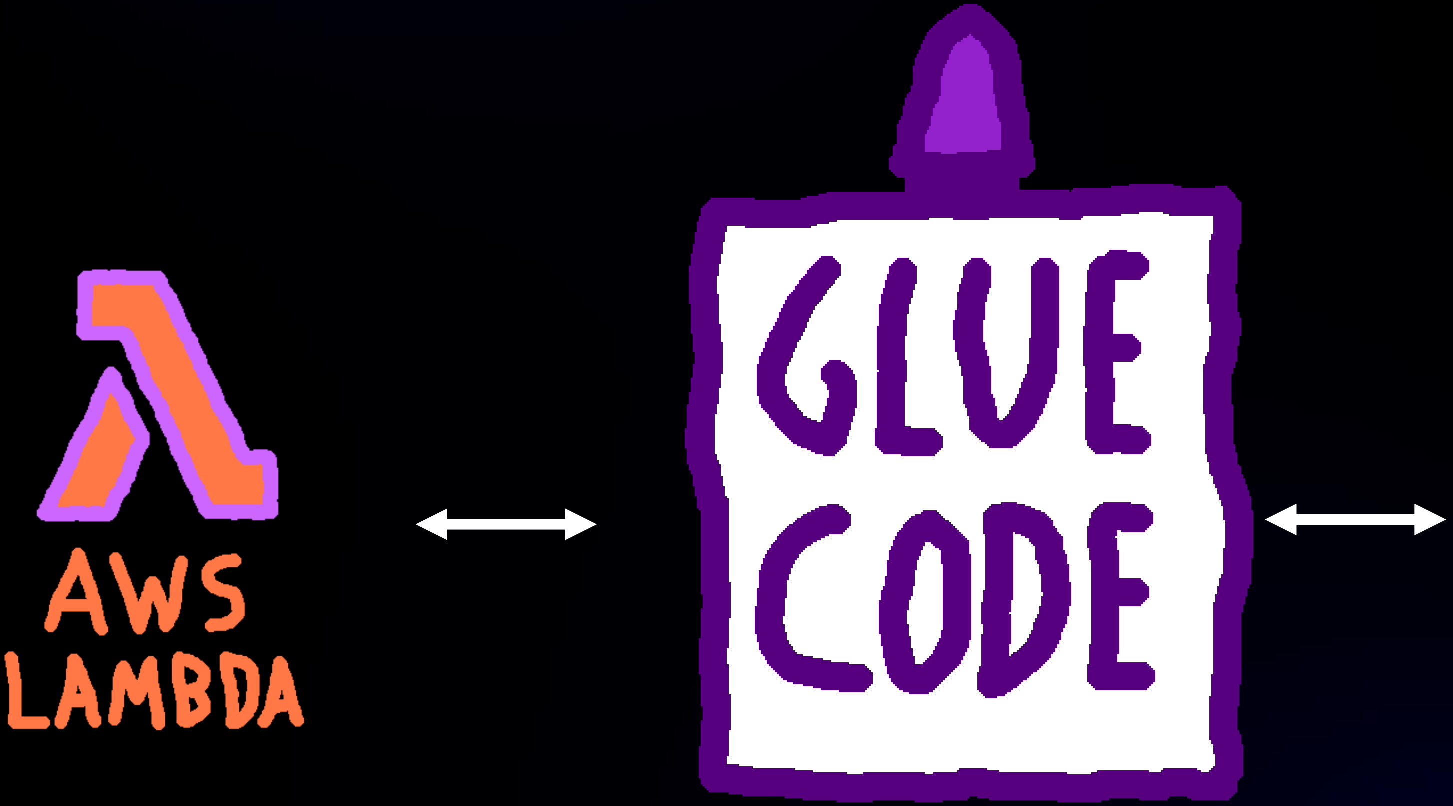
```
CglibAopProxy$DynamicAdvise        CglibAopProxy$DynamicAdvisedInterceptor.intercept
SlabPoster$SpringCGLIB-Prox        MbsBatchSender$SpringCGLIB-Proxy.postSlab
IntegrationFlowConfiguration$Lambda.transform
NativeMethodAccessorImpl.invoke0
NativeMethodAccessorImpl.inv
DelegatingMethodAccessorImpl.invoke
Method.invoke
LambdaMessageProcessor.pro
AbstractMessageProcessingTra
MessageTransformingHandler.handleRequestMessage
AbstractReplyProducingMessag
AbstractMessageHandler.handleMessage
AbstractDispatcher.tryOptimize
UnicastingDispatcher.doDispatch
UnicastingDispatcher.dispatch
AbstractSubscribableChannel.doSend
AbstractMessageChannel.send
AbstractMessageChannel.send
GenericMessagingTemplate.doSend
```

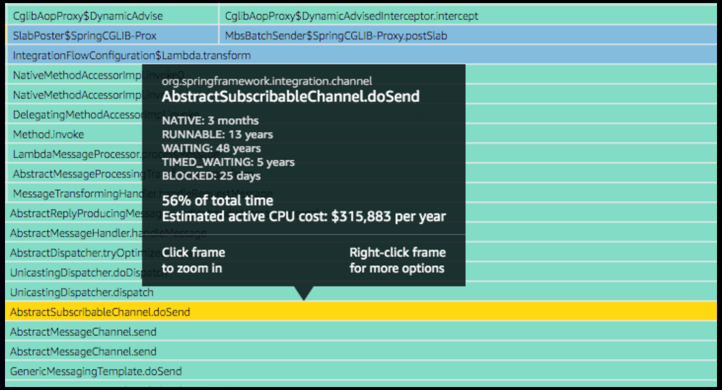
org.springframework.integration.channel
AbstractSubscribableChannel.doSend

NATIVE: 3 months
RUNNABLE: 13 years
WAITING: 48 years
TIMED_WAITING: 5 years
BLOCKED: 25 days

56% of total time
Estimated active CPU cost: $315,883 per year

Click frame to zoom in | Right-click frame for more options





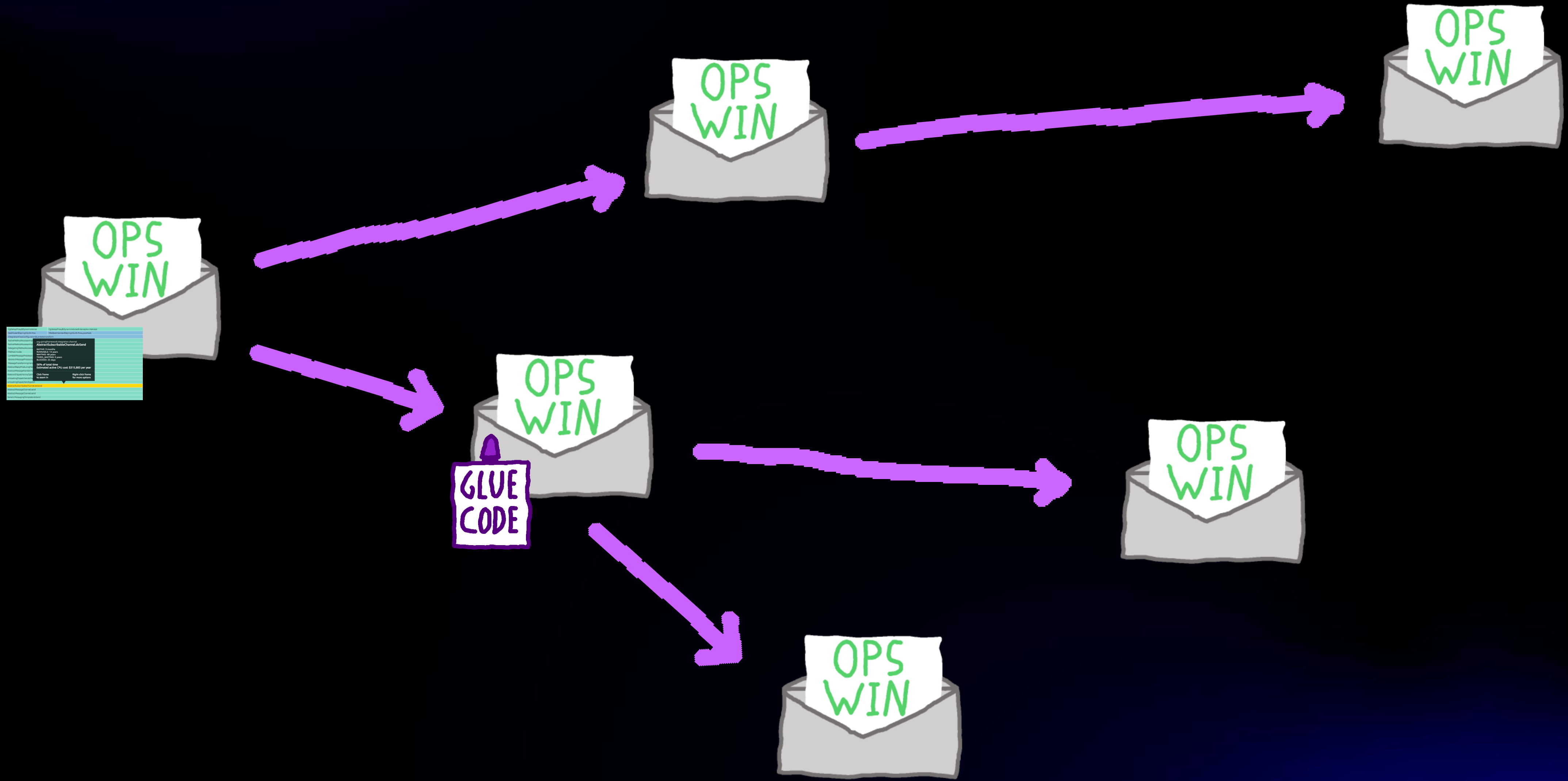
```
CglibAopProxy$DynamicAdvise      CglibAopProxy$DynamicAdvisedInterceptor.intercept
SlabPoster$SpringCGLIB-Prox      MbsBatchSender$SpringCGLIB-Proxy.postSlab
IntegrationFlowConfiguration$Lambda.transform
NativeMethodAccessorImp
NativeMethodAccessorImp
DelegatingMethodAccesso
Method.invoke
LambdaMessageProcessor.
AbstractMessageProcessing
MessageTransformingHand
AbstractReplyProducingMe
AbstractMessageHandler.h
AbstractDispatcher.tryOpti
UnicastingDispatcher.doDis
UnicastingDispatcher.dispatch
AbstractSubscribableChannel.doSend
AbstractMessageChannel.send
AbstractMessageChannel.send
GenericMessagingTemplate.doSend
```

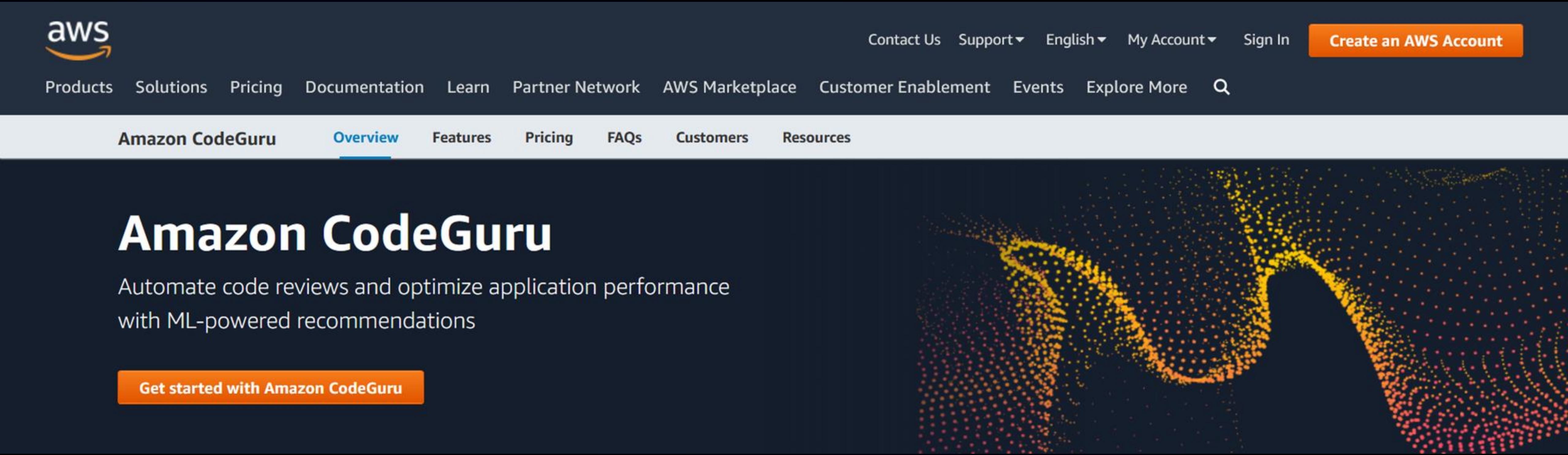
```
org.springframework.integration.channel
AbstractSubscribableChannel.doSend

NATIVE: 3 months
RUNNABLE: 13 years
WAITING: 48 years
TIMED_WAITING: 5 years
BLOCKED: 25 days

56% of total time
Estimated active CPU cost: $315,883 per year

Click frame              Right-click frame
to zoom in               for more options
```

```
CglibAopProxy$DynamicAdvise          CglibAopProxy$DynamicAdvisedInterceptor.intercept
SlabPoster$SpringCGLIB-Prox          MbsBatchSender$SpringCGLIB-Proxy.postSlab
IntegrationFlowConfiguration$Lambda.transform
NativeMethodAccessorImpl.invoke0
NativeMethodAccessorImpl.invoke
DelegatingMethodAccessorImpl.invoke
Method.invoke
LambdaMessageProcessor.process
AbstractMessageProcessingTr
MessageTransformingHandler.handleRequestMessage
AbstractReplyProducingMes
AbstractMessageHandler.handleMessage
AbstractDispatcher.tryOptimize
UnicastingDispatcher.doDispatch
UnicastingDispatcher.dispatch
AbstractSubscribableChannel.doSend
AbstractMessageChannel.send
AbstractMessageChannel.send
GenericMessagingTemplate.doSend
```

```
org.springframework.integration.channel
AbstractSubscribableChannel.doSend
NATIVE: 3 months
RUNNABLE: 13 years
WAITING: 48 years
TIMED_WAITING: 5 years
BLOCKED: 25 days
56% of total time
Estimated active CPU cost: $315,883 per year
Click frame          Right-click frame
to zoom in           for more options
```

The ripple effect





Amazon CodeGuru

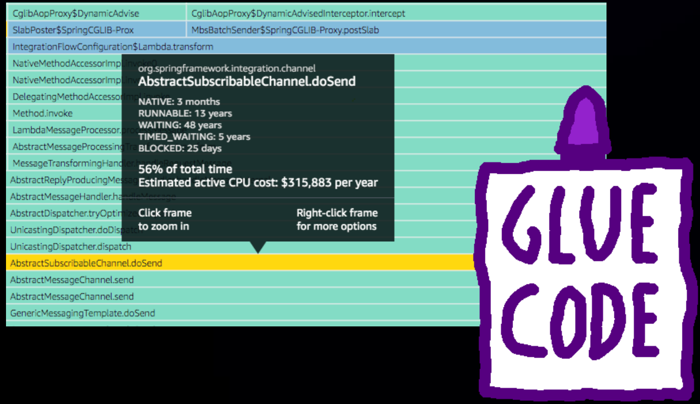
Automate code reviews and optimize application performance with ML-powered recommendations

The ripple effect

Amazon CodeGuru

Automate code reviews and optimize application performance with ML-powered recommendations

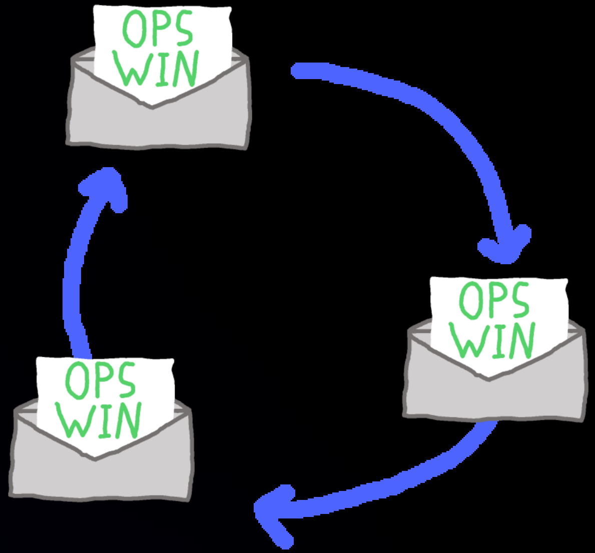
Takeaways: The ops win



```
CglibAopProxy$DynamicAdvise         CglibAopProxy$DynamicAdvisedInterceptor.intercept
SlabPoster$SpringCGLIB-Prox         MbsBatchSender$SpringCGLIB-Proxy.postSlab
IntegrationFlowConfiguration$Lambda.transform
NativeMethodAccessorImpl
NativeMethodAccessorImpl
DelegatingMethodAccesso
Method.invoke
LambdaMessageProcessor.
AbstractMessageProcessing
MessageTransformingHand
AbstractReplyProducingMes
AbstractMessageHandler.ha
AbstractDispatcher.tryOpti
UnicastingDispatcher.doDisp
UnicastingDispatcher.dispatch
AbstractSubscribableChannel.doSend
AbstractMessageChannel.send
AbstractMessageChannel.send
GenericMessagingTemplate.doSend
```

```
org.springframework.integration.channel
AbstractSubscribableChannel.doSend

NATIVE: 3 months
RUNNABLE: 13 years
WAITING: 48 years
TIMED_WAITING: 5 years
BLOCKED: 25 days

56% of total time
Estimated active CPU cost: $315,883 per year

Click frame          Right-click frame
to zoom in           for more options
```

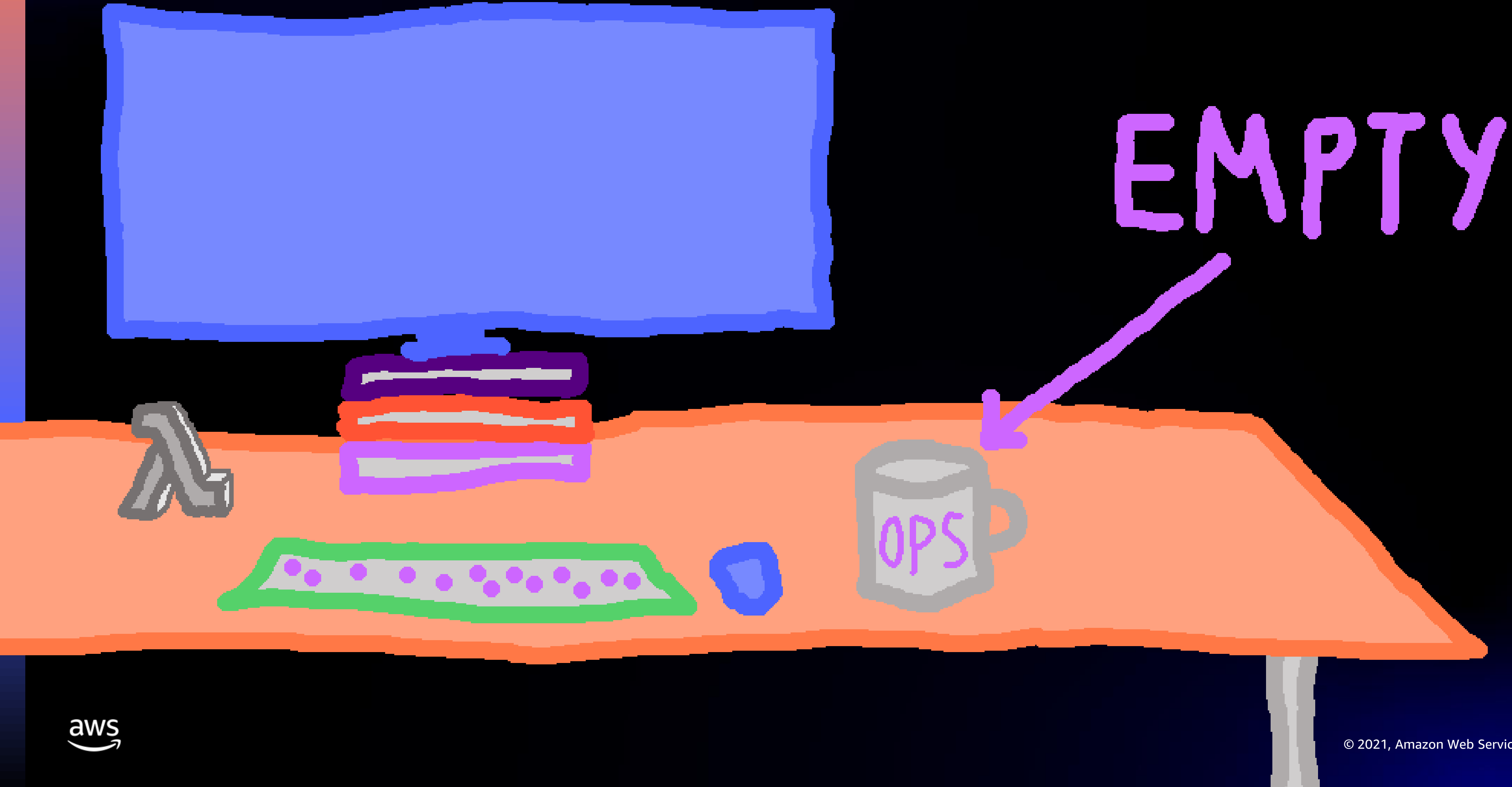
Share best practices



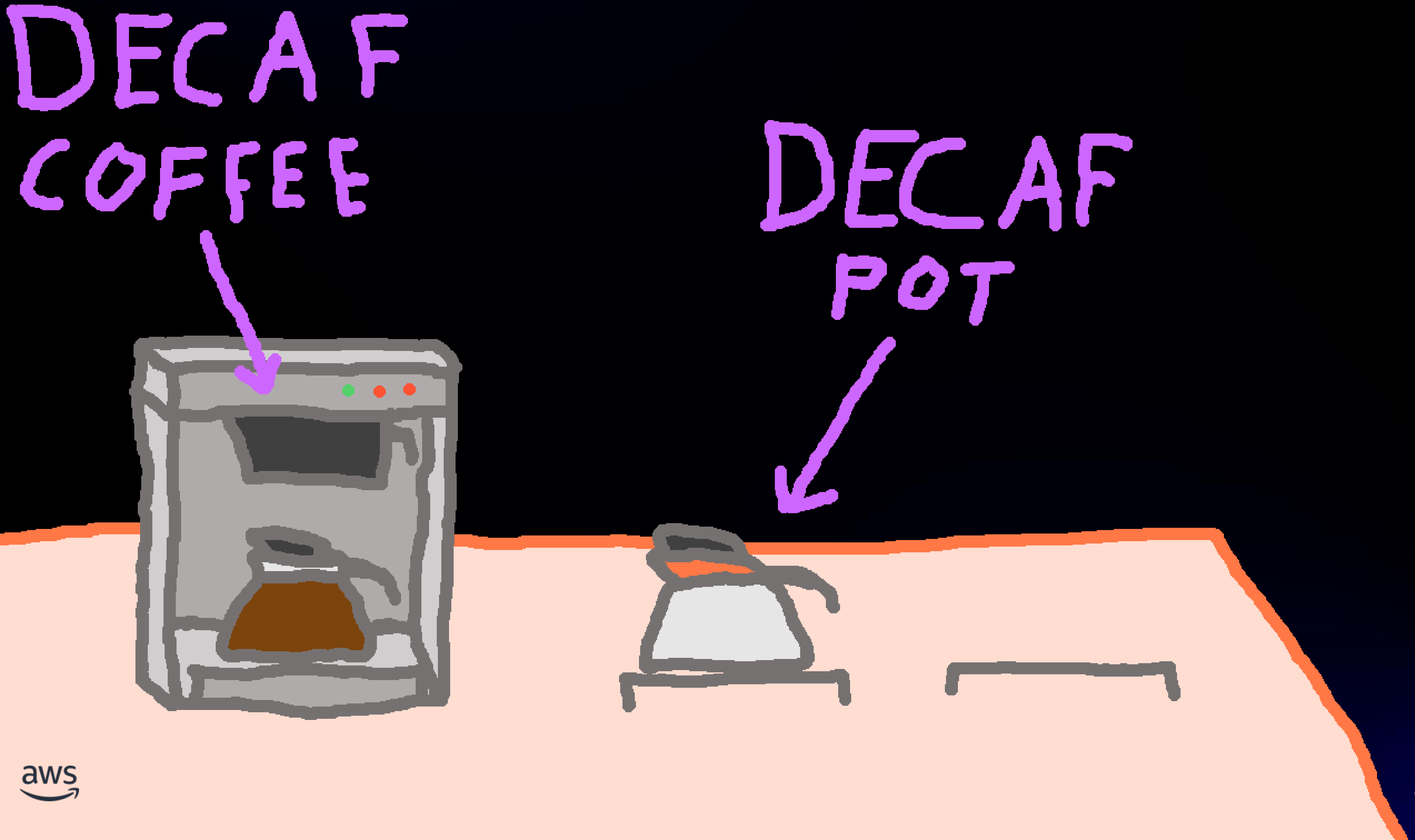
Reinforce ops culture

Story 2: The retrospective

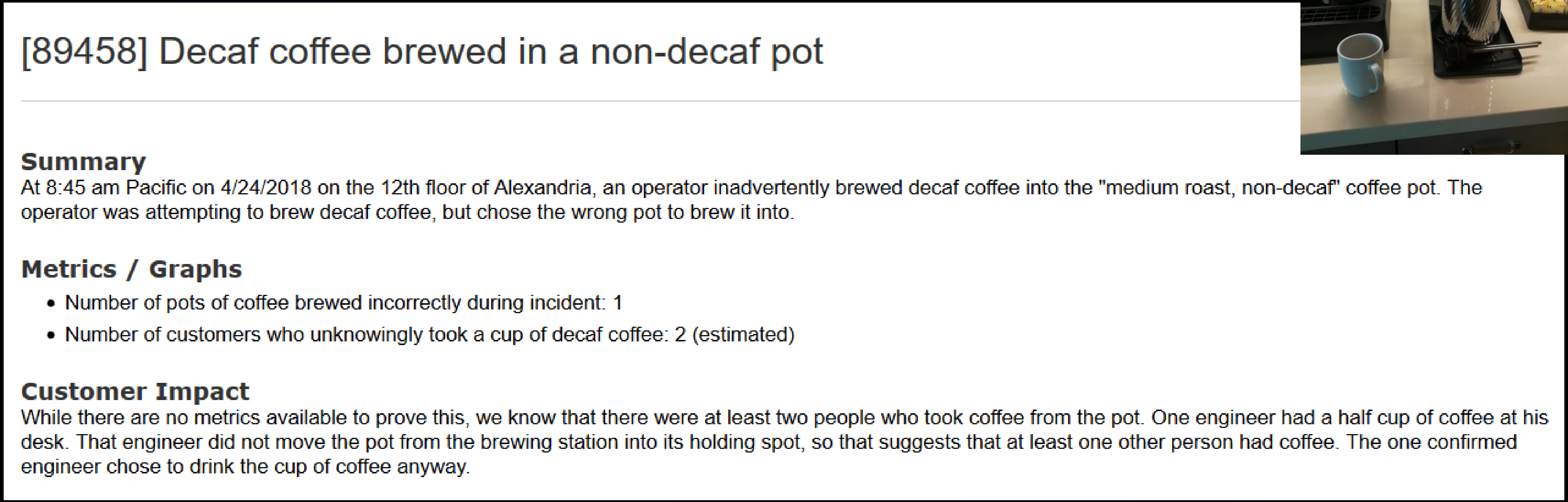
Early one morning . . .



Wrong coffee, wrong pot



COE: Correction of error



[89458] Decaf coffee brewed in a non-decaf pot

Summary
At 8:45 am Pacific on 4/24/2018 on the 12th floor of Alexandria, an operator inadvertently brewed decaf coffee into the "medium roast, non-decaf" coffee pot. The operator was attempting to brew decaf coffee, but chose the wrong pot to brew it into.

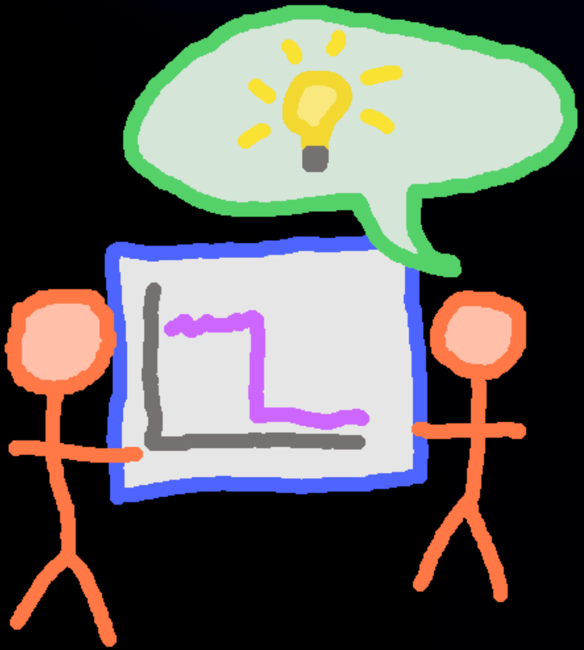
Metrics / Graphs

- Number of pots of coffee brewed incorrectly during incident: 1
- Number of customers who unknowingly took a cup of decaf coffee: 2 (estimated)

Customer Impact
While there are no metrics available to prove this, we know that there were at least two people who took coffee from the pot. One engineer had a half cup of coffee at his desk. That engineer did not move the pot from the brewing station into its holding spot, so that suggests that at least one other person had coffee. The one confirmed engineer chose to drink the cup of coffee anyway.

Amazon's approach to failing successfully
https://www.youtube.com/watch?v=yQiRli2ZPxU

Goals of retrospectives



Improve systems

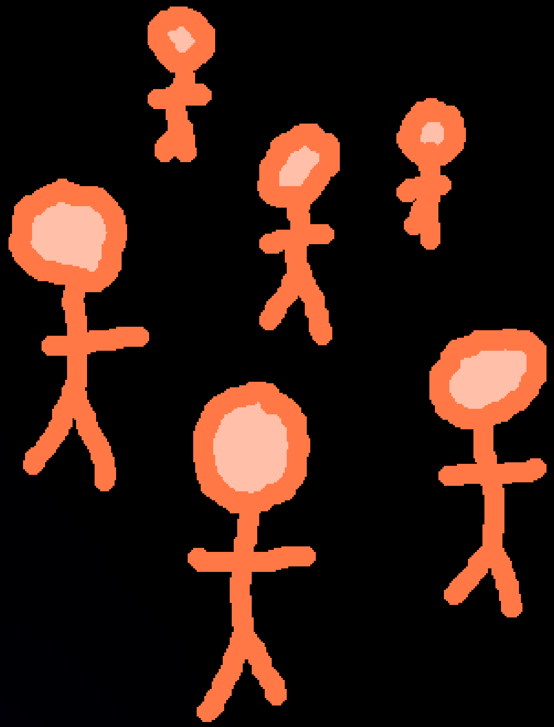
Teach others

Improve tools

Summary

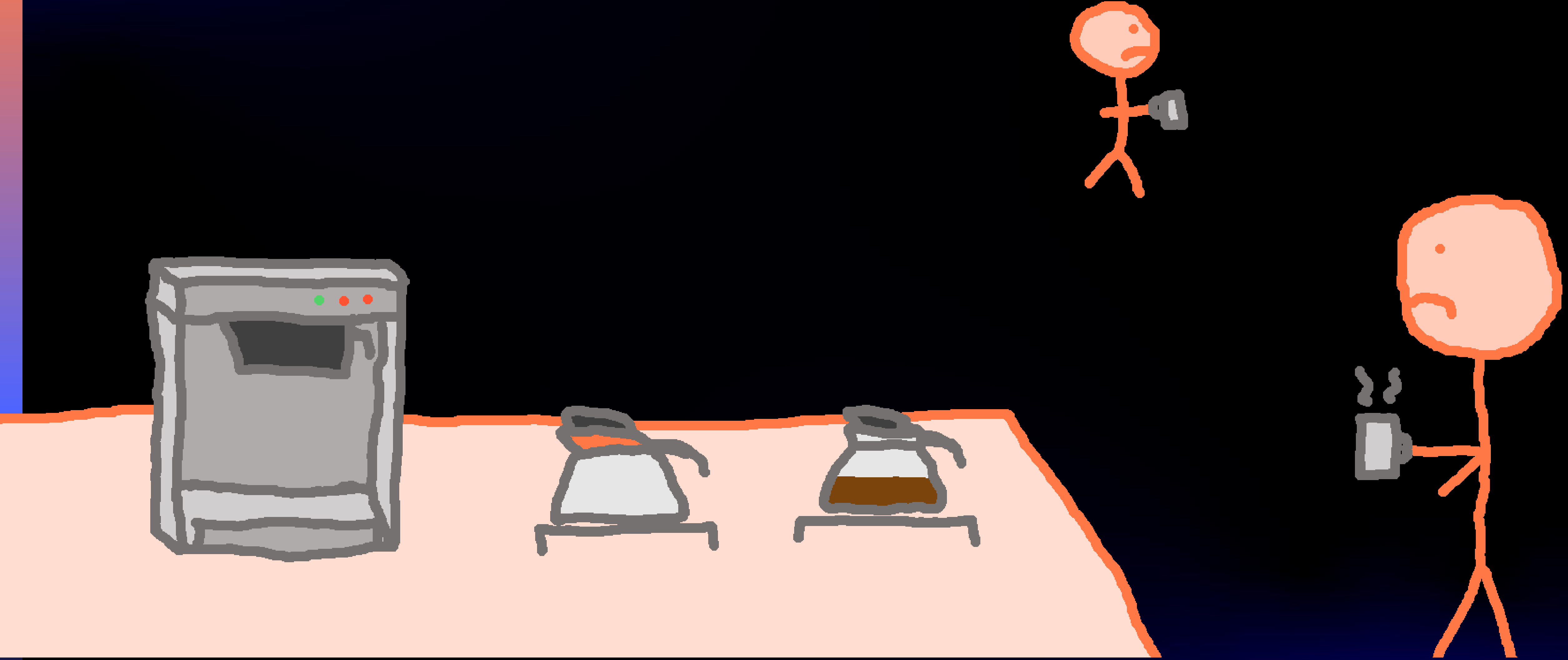
Non-punitive
(no names!)

3 paragraphs



Understandable by anyone
(no jargon!)

Customer impact

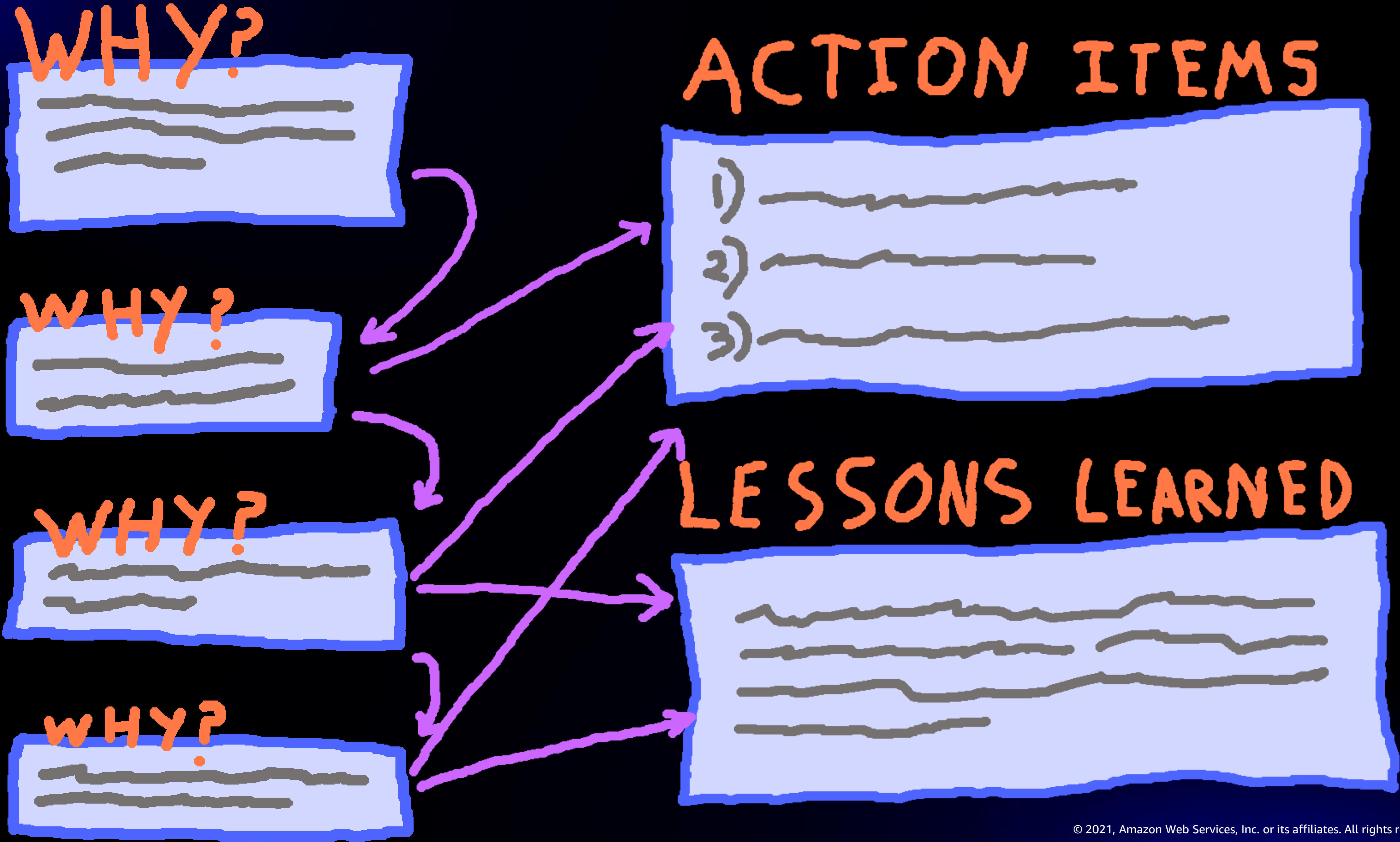


Graphs and timeline

BREW METRICS

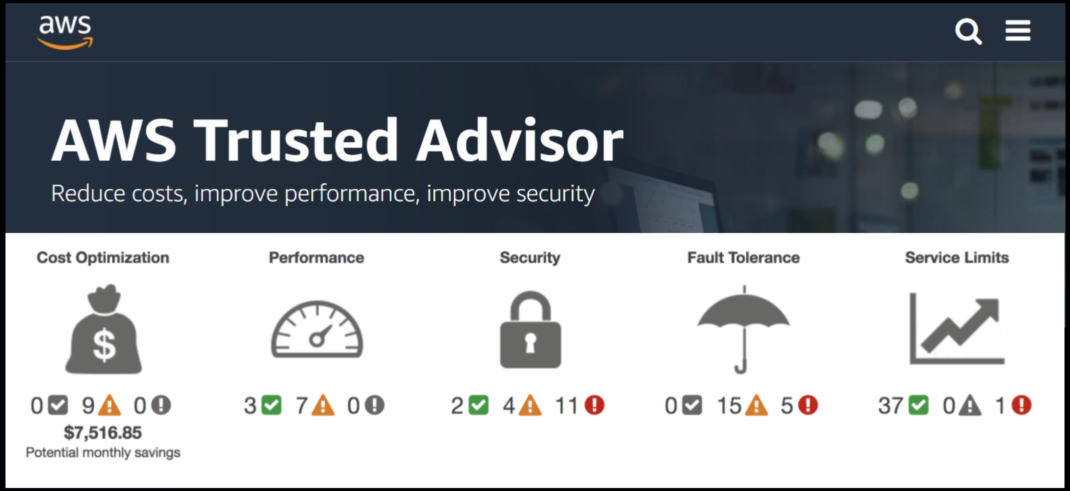
Timeline

08:29 – Operator accidentally selects “decaf” coffee packet
08:30 – Operator selects “regular” coffee pot
08:31 – Operator loads coffee and starts brewing
08:32 – Operator goes back to desk
08:36 – Coffee finishes brewing
08:?? – Someone removes carafe from coffee maker and takes a cup
08:45 – Operator returns and takes the first cup
09:25 – Operator returns for another cup
09:26 – Operator realizes that the coffee tastes different than normal
09:27 – Investigation begins
09:29 – Original coffee packet retrieved, confirmed decaf
09:30 – Decaf label transferred to regular carafe



aws

Sharing and improving broadly



AWS Trusted Advisor

Reduce costs, improve performance, improve security

Cost Optimization	Performance	Security	Fault Tolerance	Service Limits
0 ✓ 9 ⚠ 0 !	3 ✓ 7 ⚠ 0 !	2 ✓ 4 ⚠ 11 !	0 ✓ 15 ⚠ 5 !	37 ✓ 0 ⚠ 1 !
$7,516.85 Potential monthly savings				

AWS Well-Architected

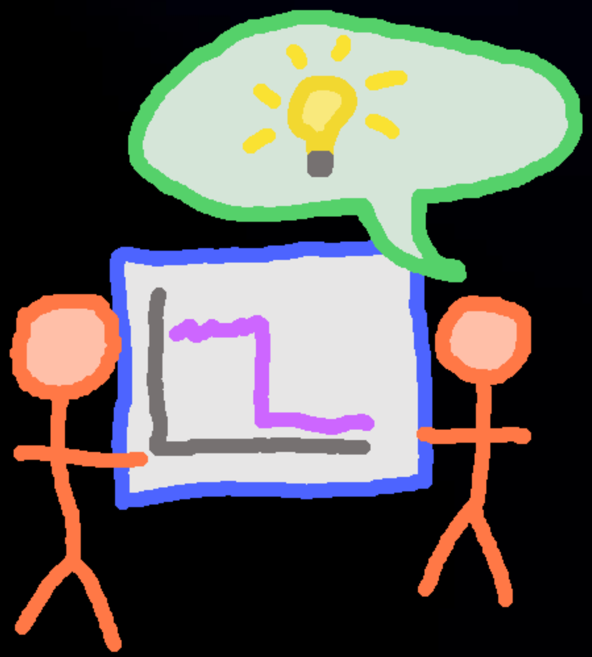
Learn, measure, and build using architectural best practices

Developer Center / ...

The Amazon Builders' Library

How Amazon builds and operates software

Takeaways: The retrospective



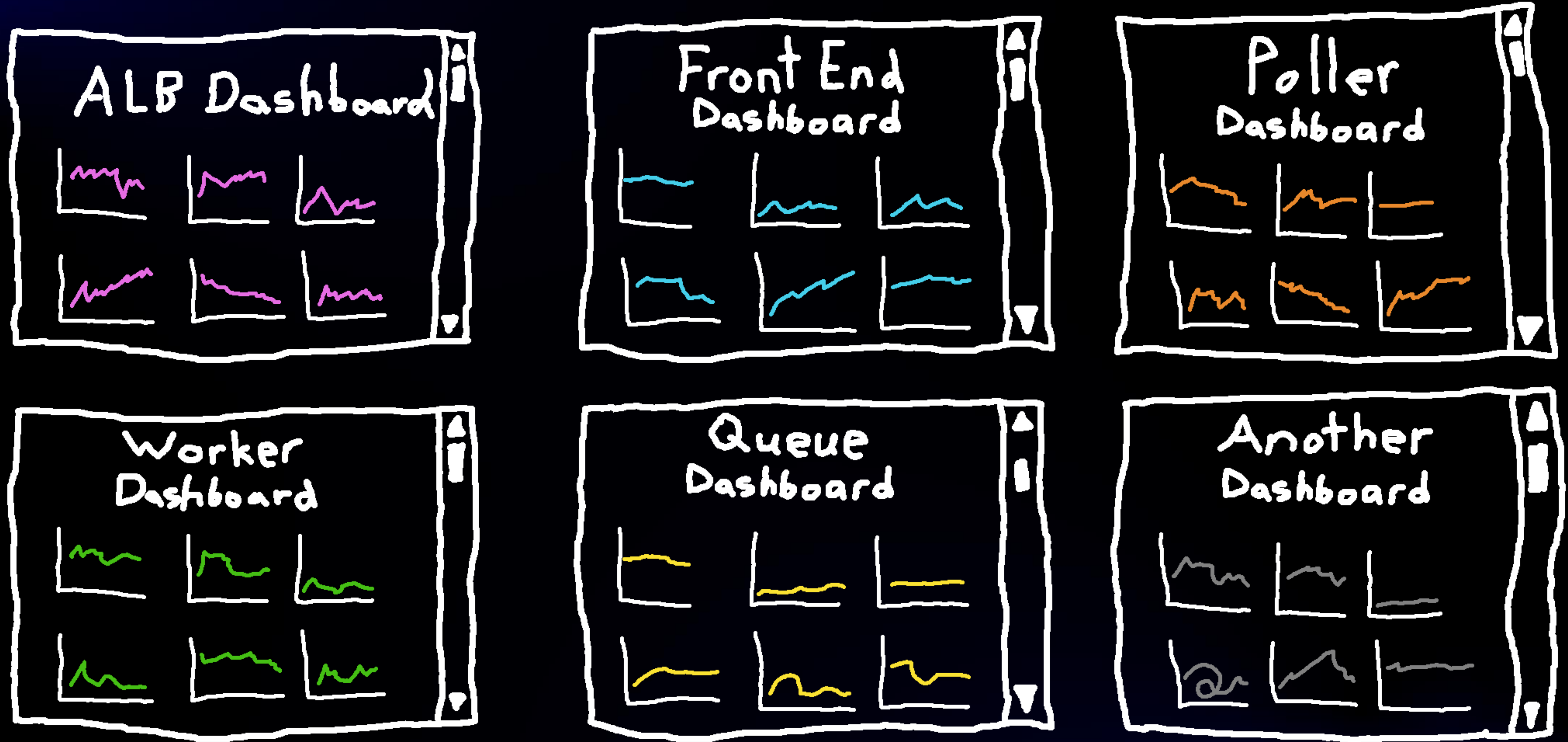
Improve systems

Teach others

Improve tools

Story 3: The ops meeting

Dashboards!



https://aws.amazon.com/builders-library/building-dashboards-for-operational-visibility/

Morning metrics

Weekly team operations meeting

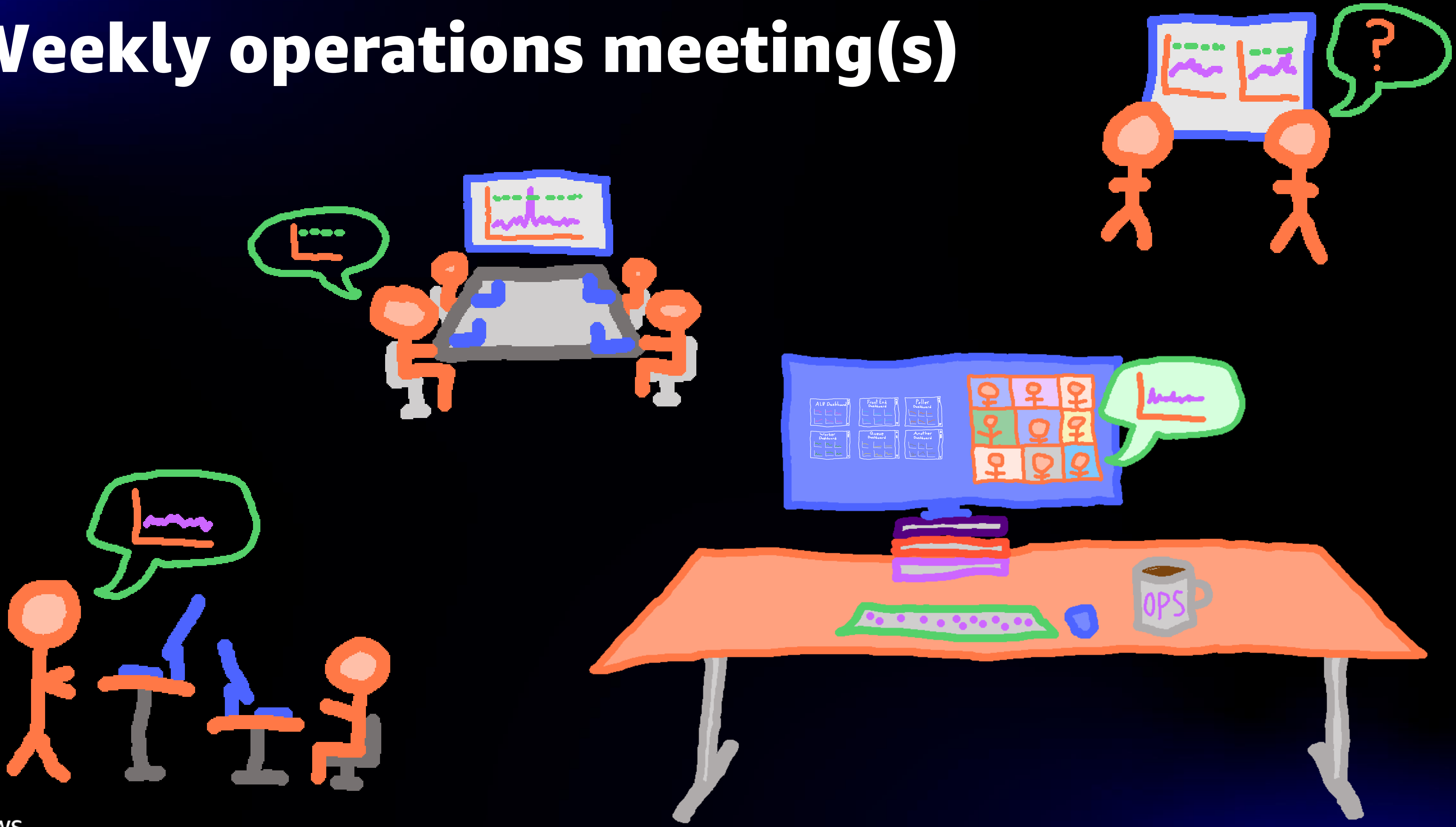
Weekly ops agenda

Wins

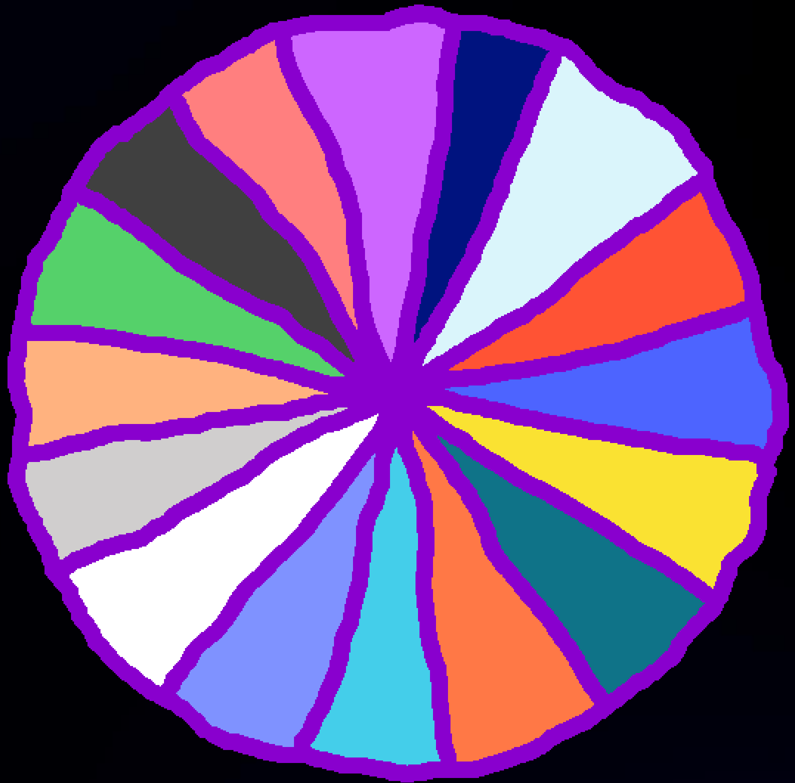
Retrospectives

Dashboards

Weekly operations meeting(s)

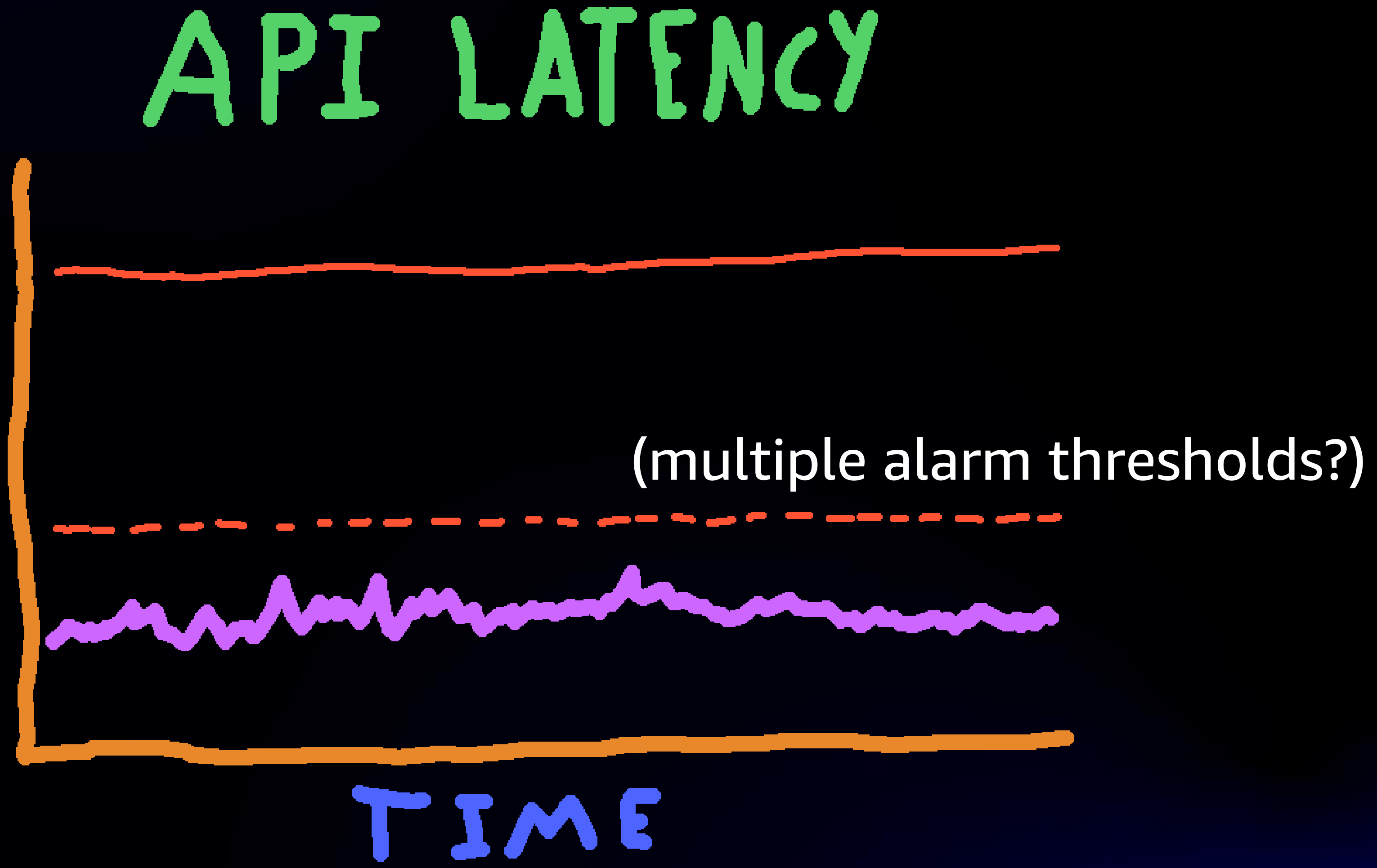


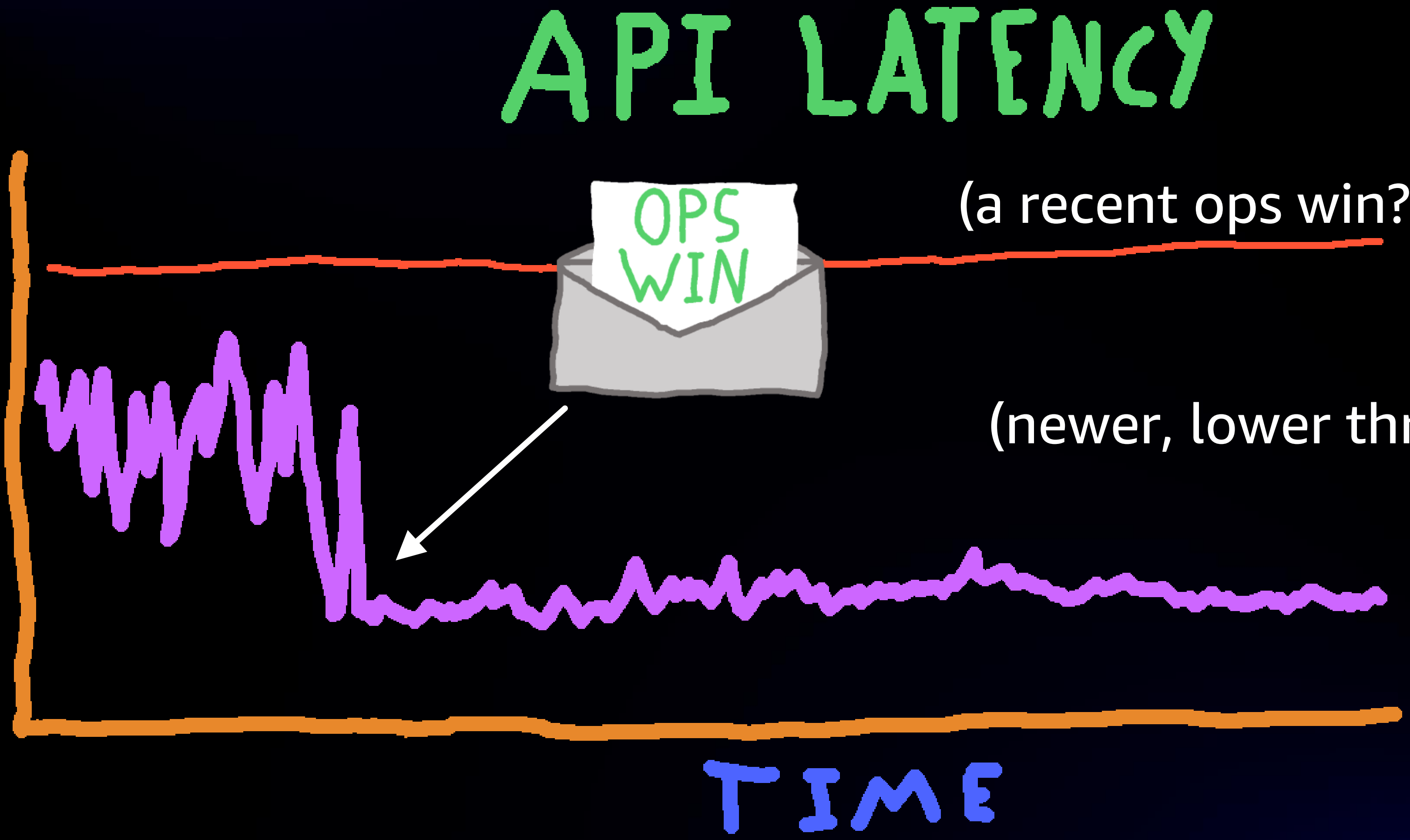
Spinning the wheel



https://github.com/aws/aws-ops-wheel

Feedback on dashboard metrics

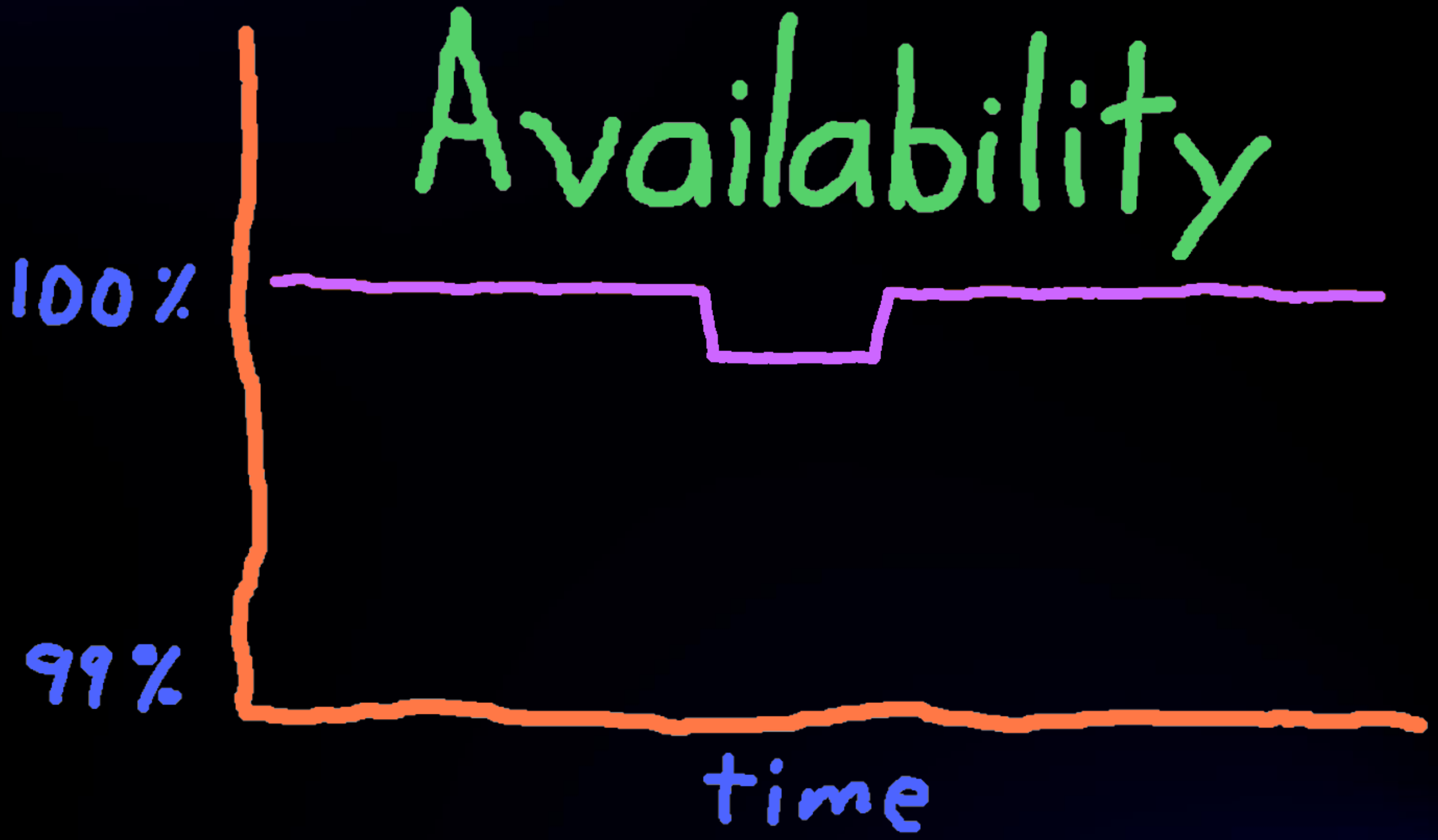




(a recent ops win?)

(newer, lower threshold?)

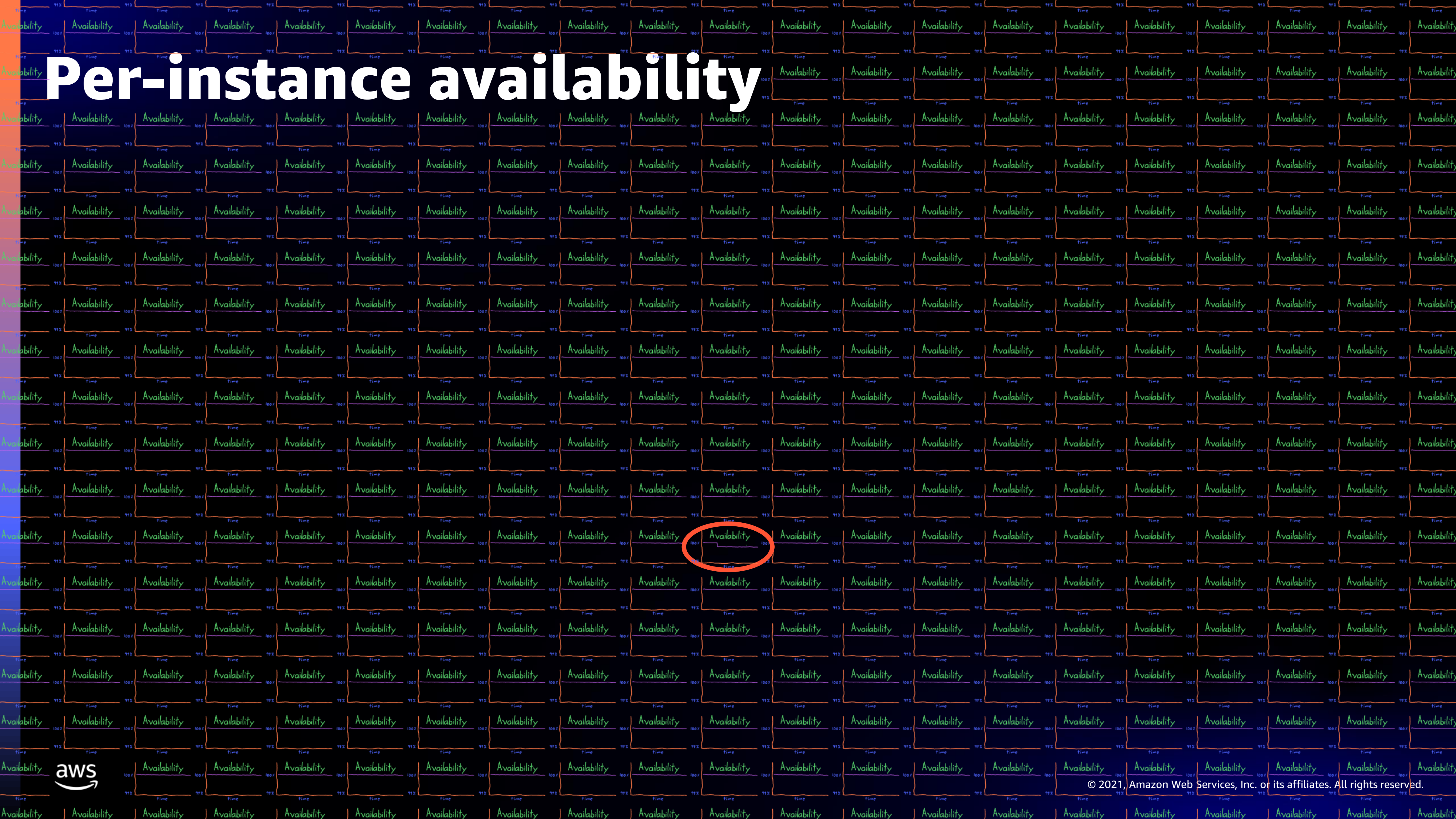
Overall availability



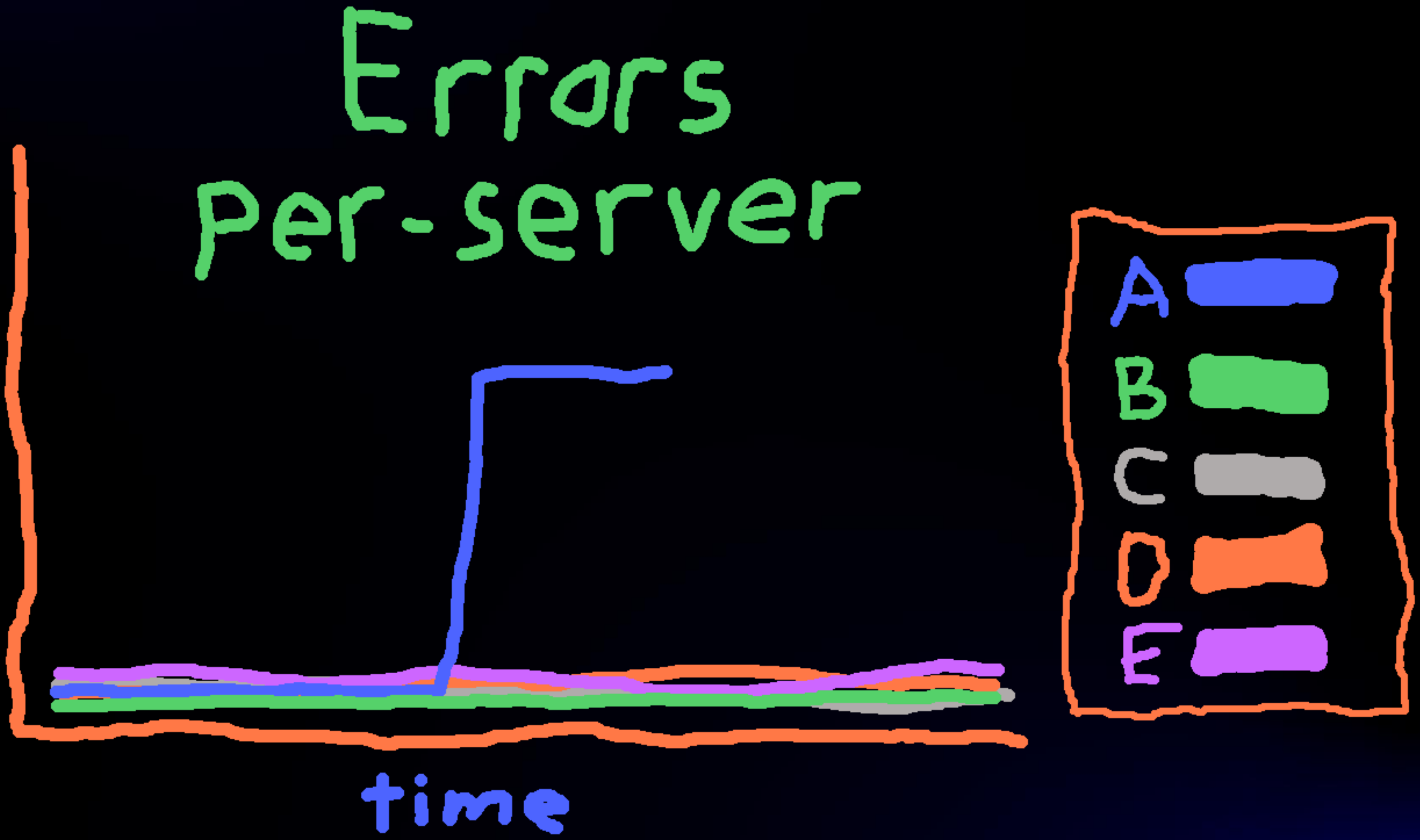
Per-instance availability

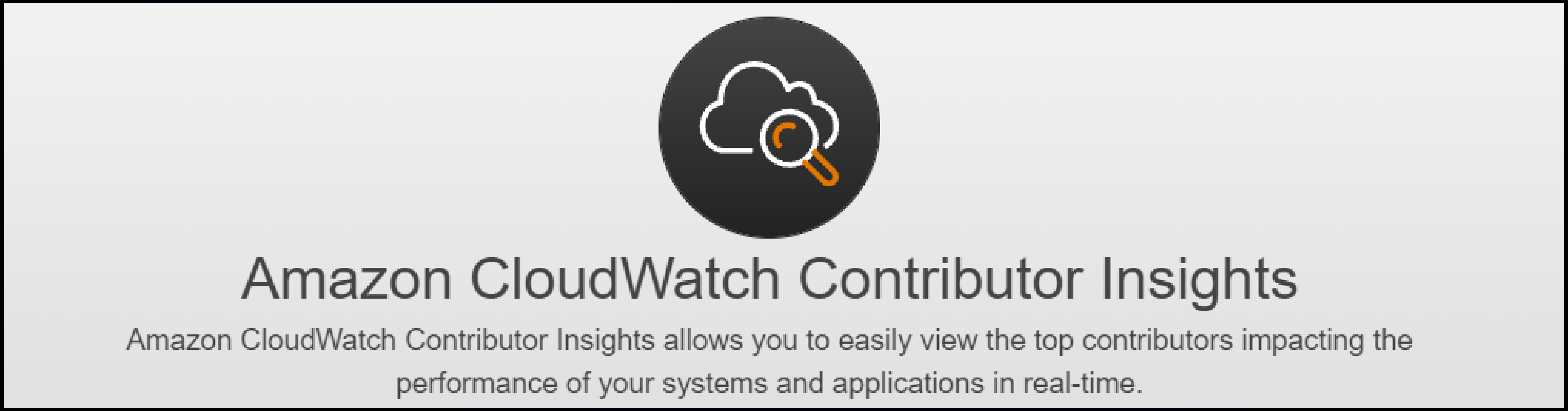
Availability
Time
100%
99%

Per-instance availability



Top-N instances by error count

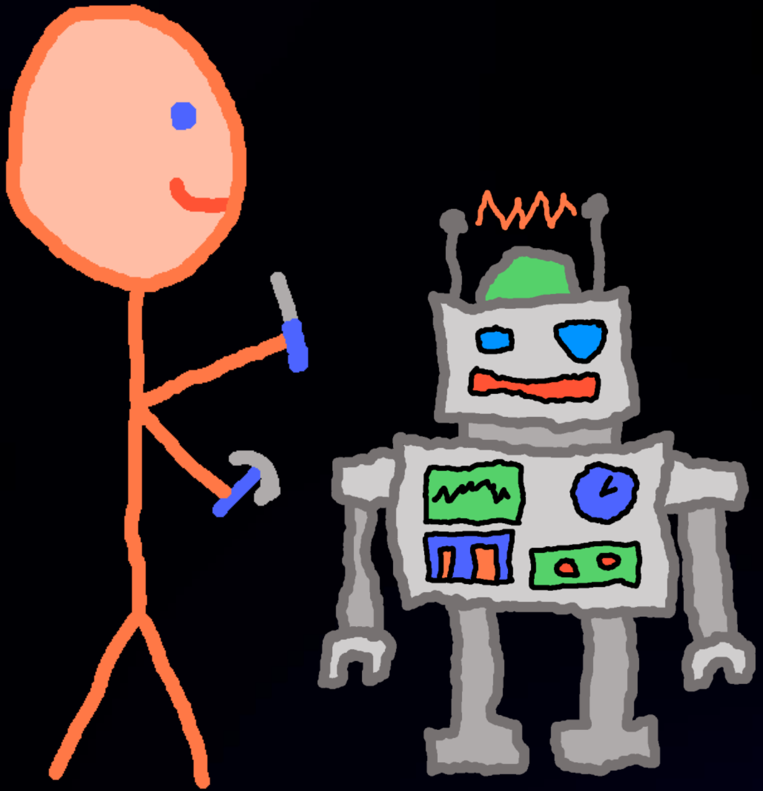




Amazon CloudWatch Contributor Insights

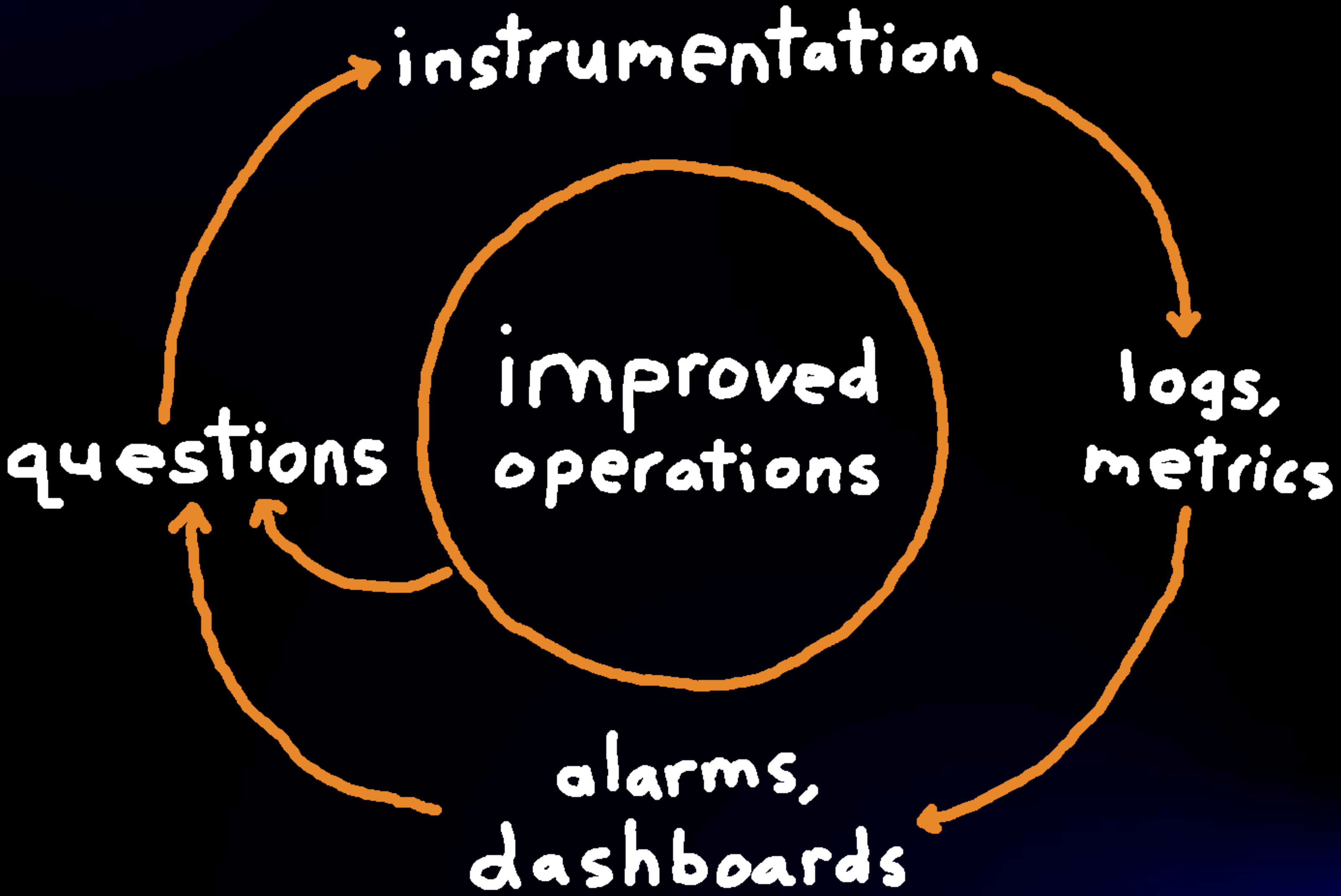
Amazon CloudWatch Contributor Insights allows you to easily view the top contributors impacting the performance of your systems and applications in real-time.

Automation



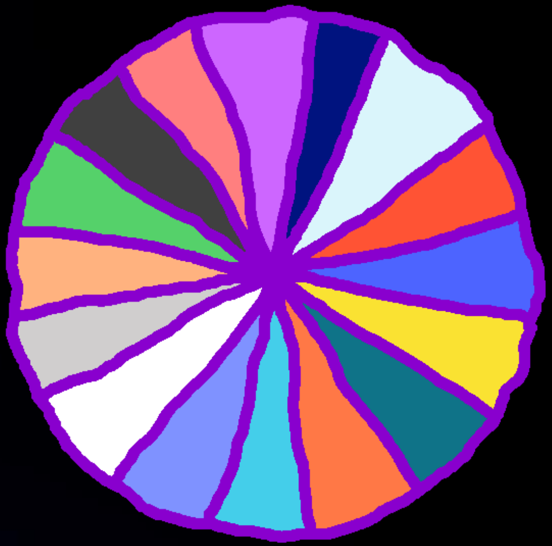
https://aws.amazon.com/builders-library/implementing-health-checks/

The cycle of monitoring



Takeaways: The ops meeting

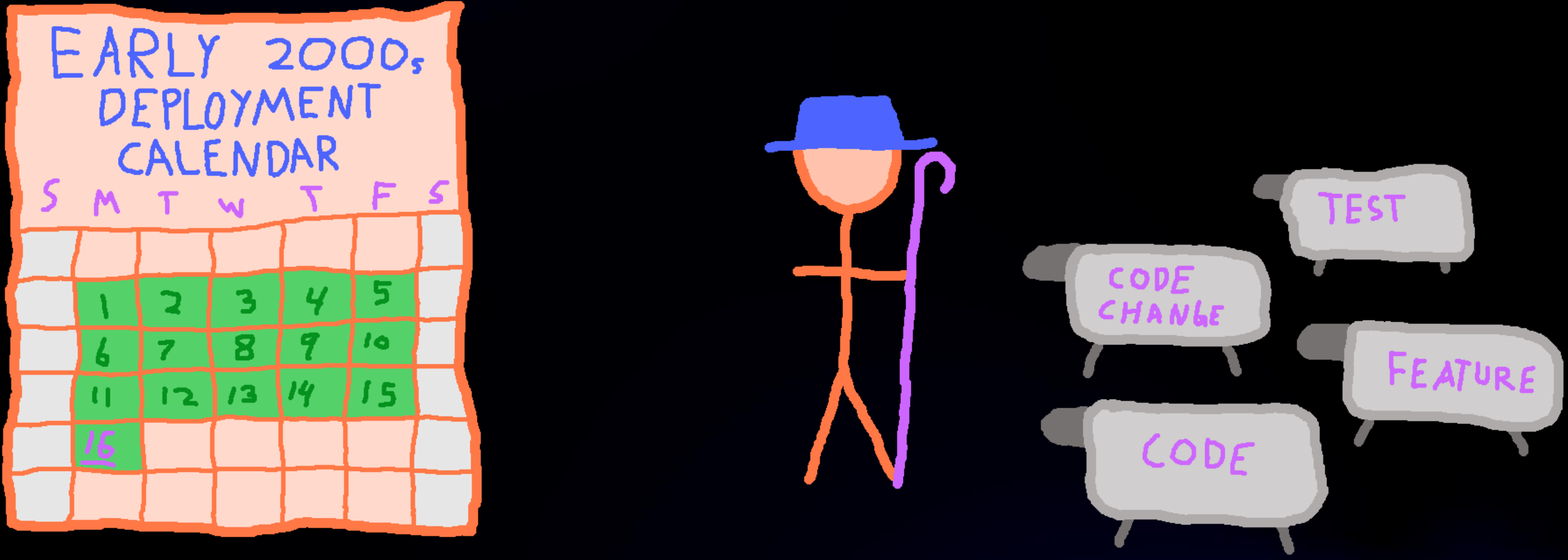
Learn from each other



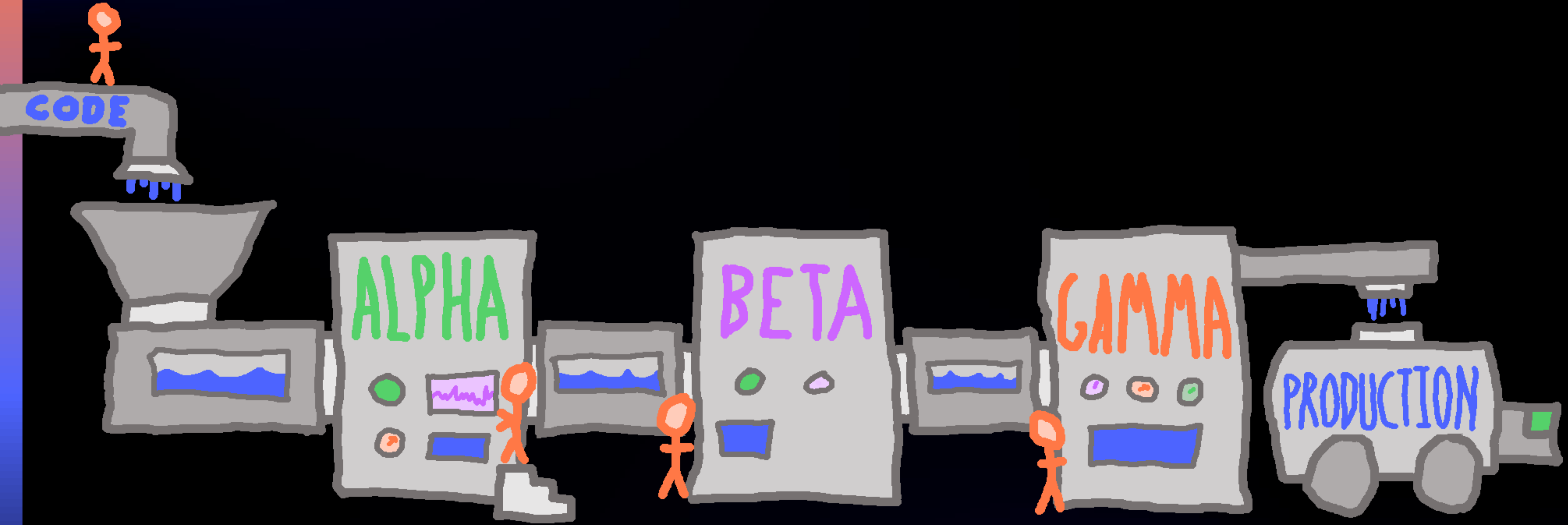
Practice regularly

Story 4: The investment in agility

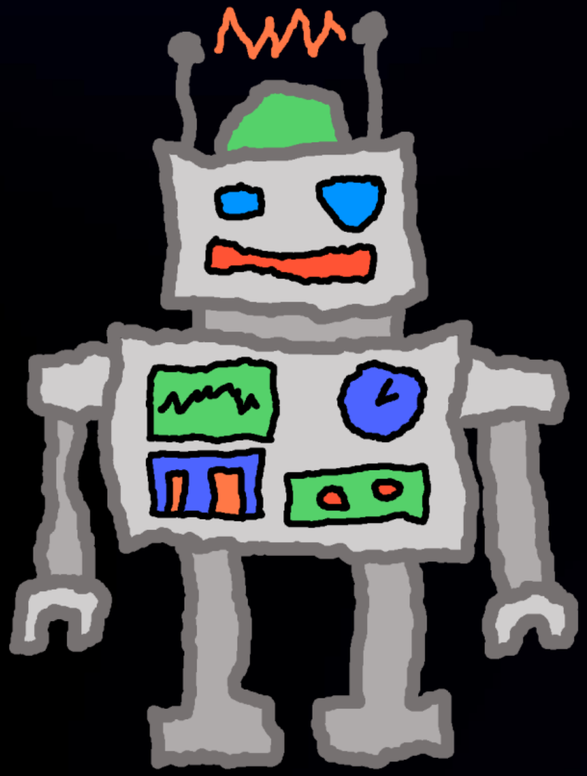
Sluggish deployment velocity

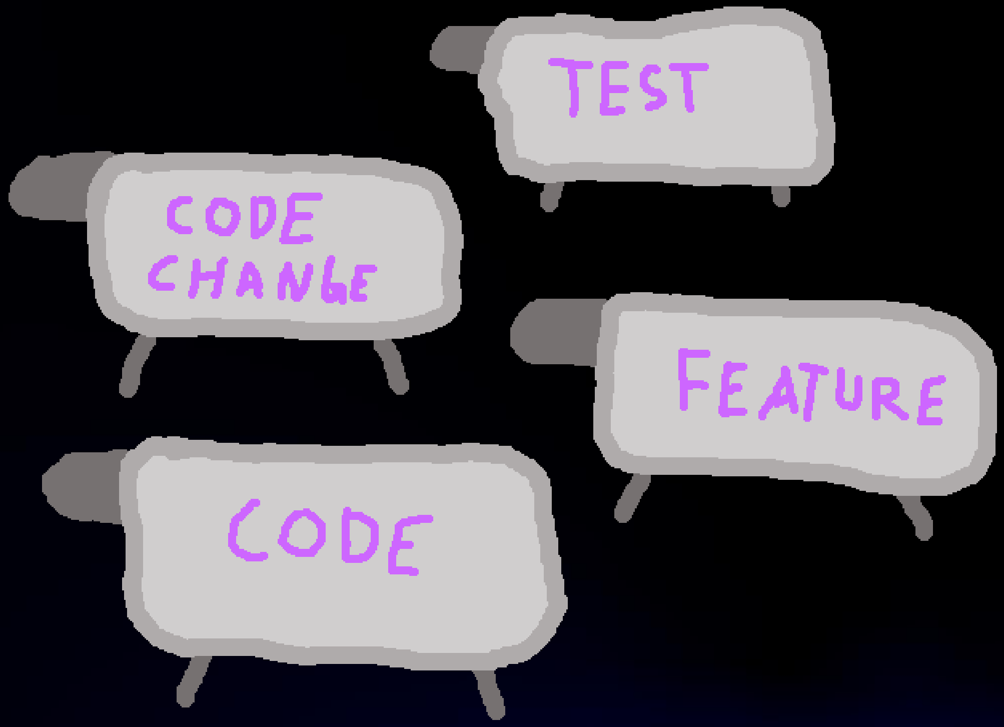


Deployment automation

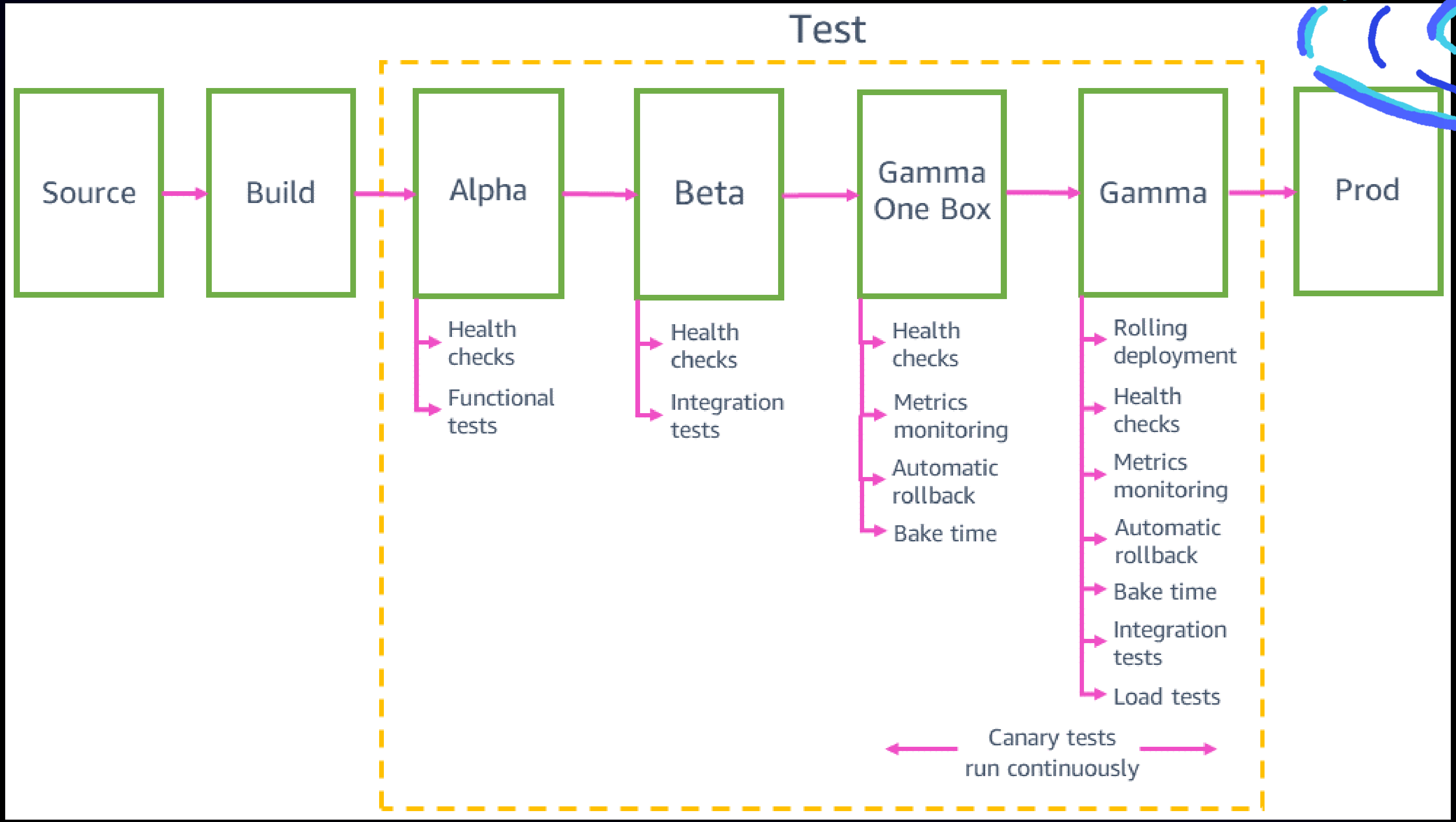


Ripple effect: Improved test automation



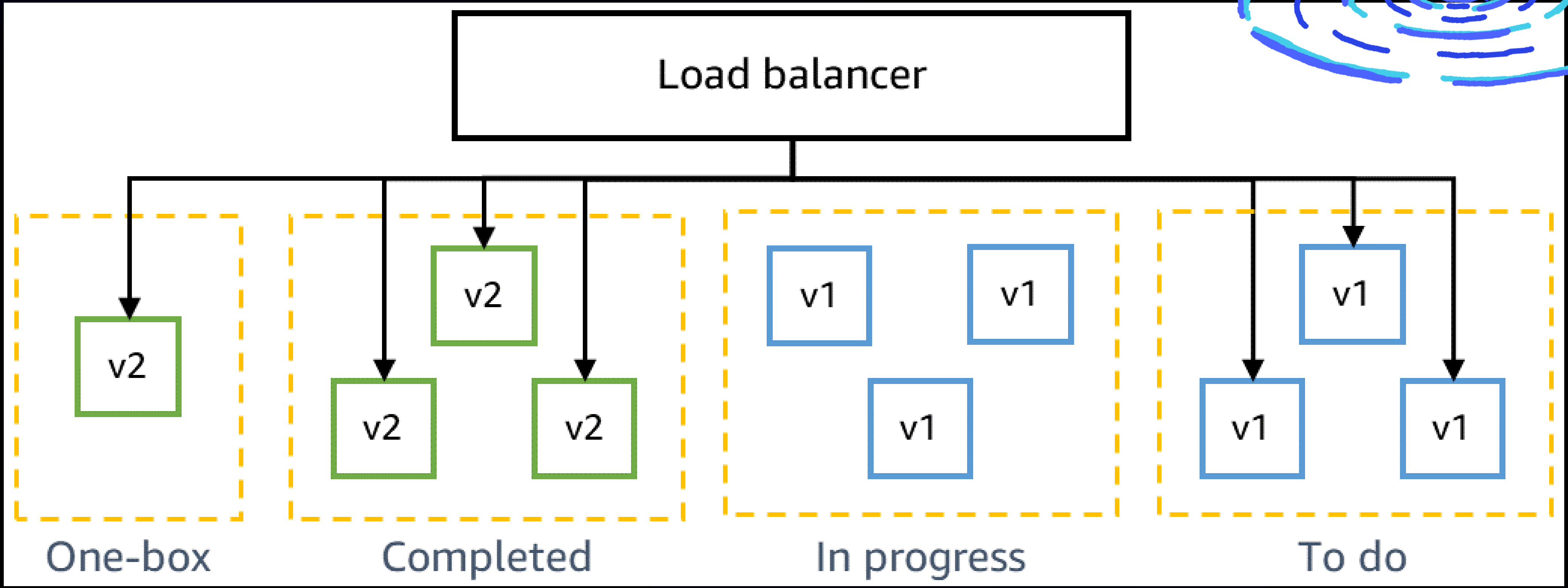


Ripple effect: Improved test automation



https://aws.amazon.com/builders-library/automating-safe-hands-off-deployments/

Ripple effect: Formalized phased deployment

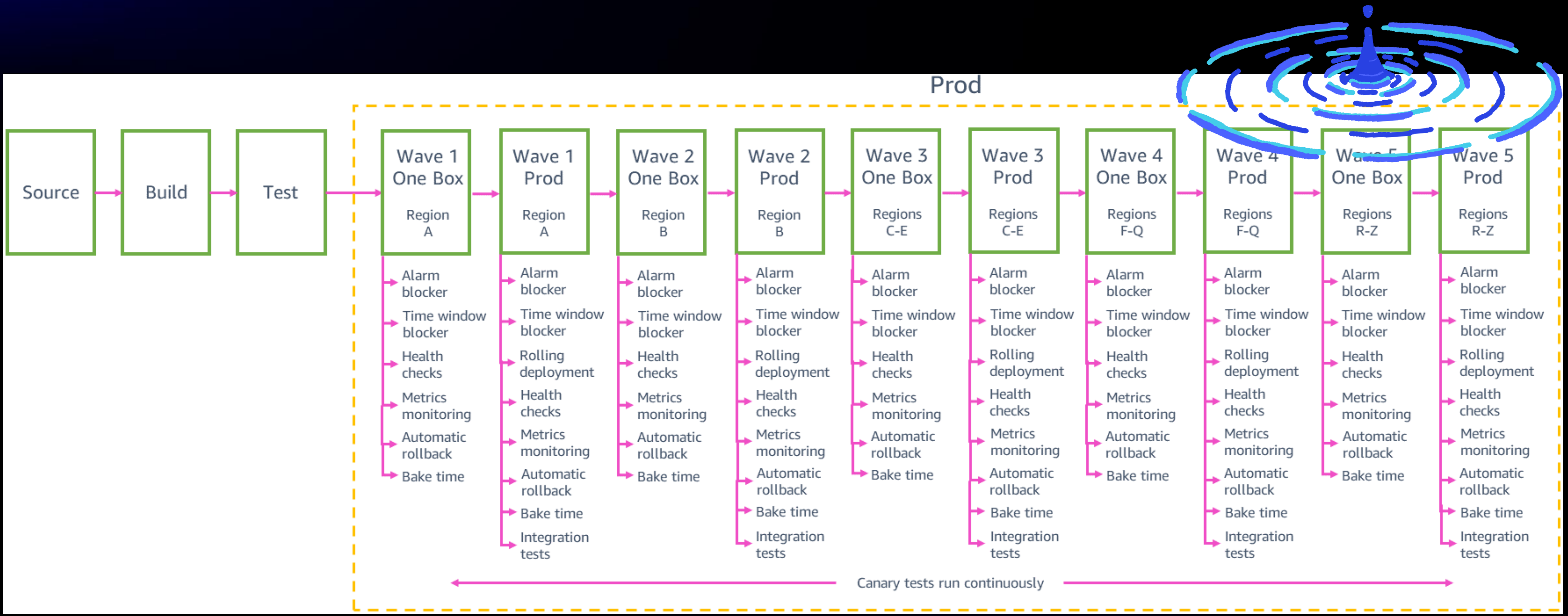


Ripple effect: Improved rollback automation

```
ALARM("FrontEndApiService_High_Fault_Rate") OR
ALARM("FrontEndApiService_High_P50_Latency") OR
ALARM("FrontEndApiService_High_P90_Latency") OR
ALARM("FrontEndApiService_High_P99_Latency") OR
ALARM("FrontEndApiService_High_Cpu_Usage") OR
ALARM("FrontEndApiService_High_Memory_Usage") OR
ALARM("FrontEndApiService_High_Disk_Usage") OR
ALARM("FrontEndApiService_High_Errors_In_Logs") OR
ALARM("FrontEndApiService_High_Failing_Health_Checks") OR
ALARM("BackendApiService_High_Severity") OR
ALARM("BackendWorkflows_High_Severity") OR
ALARM("Canaries_High_Severity")
```

https://aws.amazon.com/builders-library/ensuring-rollback-safety-during-deployments/

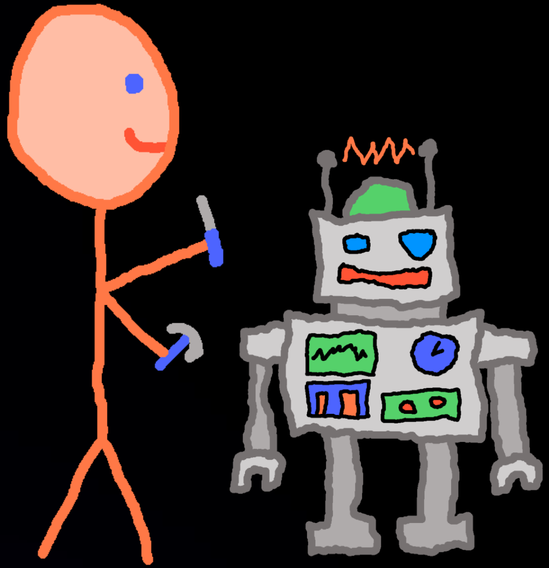
Ripple effect: Improved practices



https://aws.amazon.com/builders-library/going-faster-with-continuous-delivery/

Takeaways: The investment in agility

Continuous delivery has a ripple effect improving agility and quality

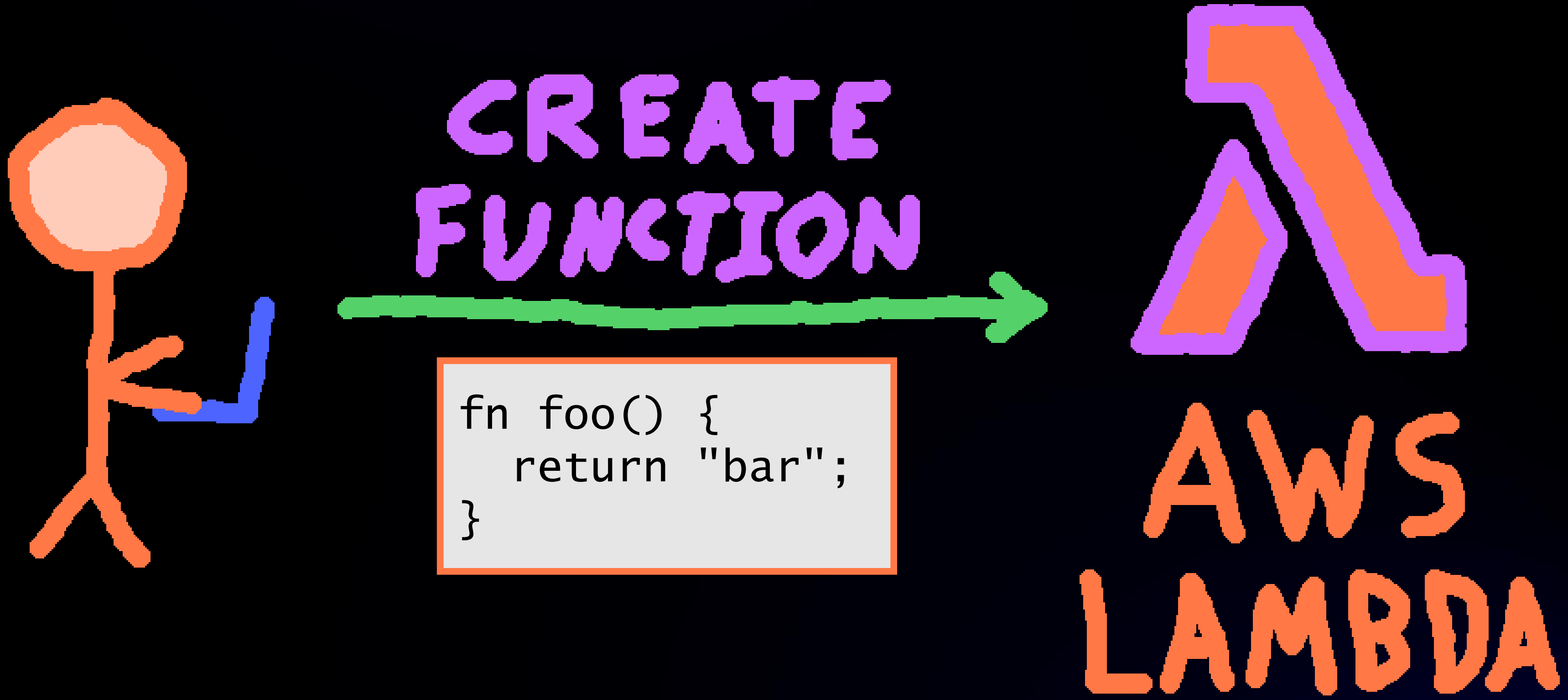


Operational excellence takes long-term investment

Story 5: The architectural choice

AWS Lambda functions

AWS Lambda functions

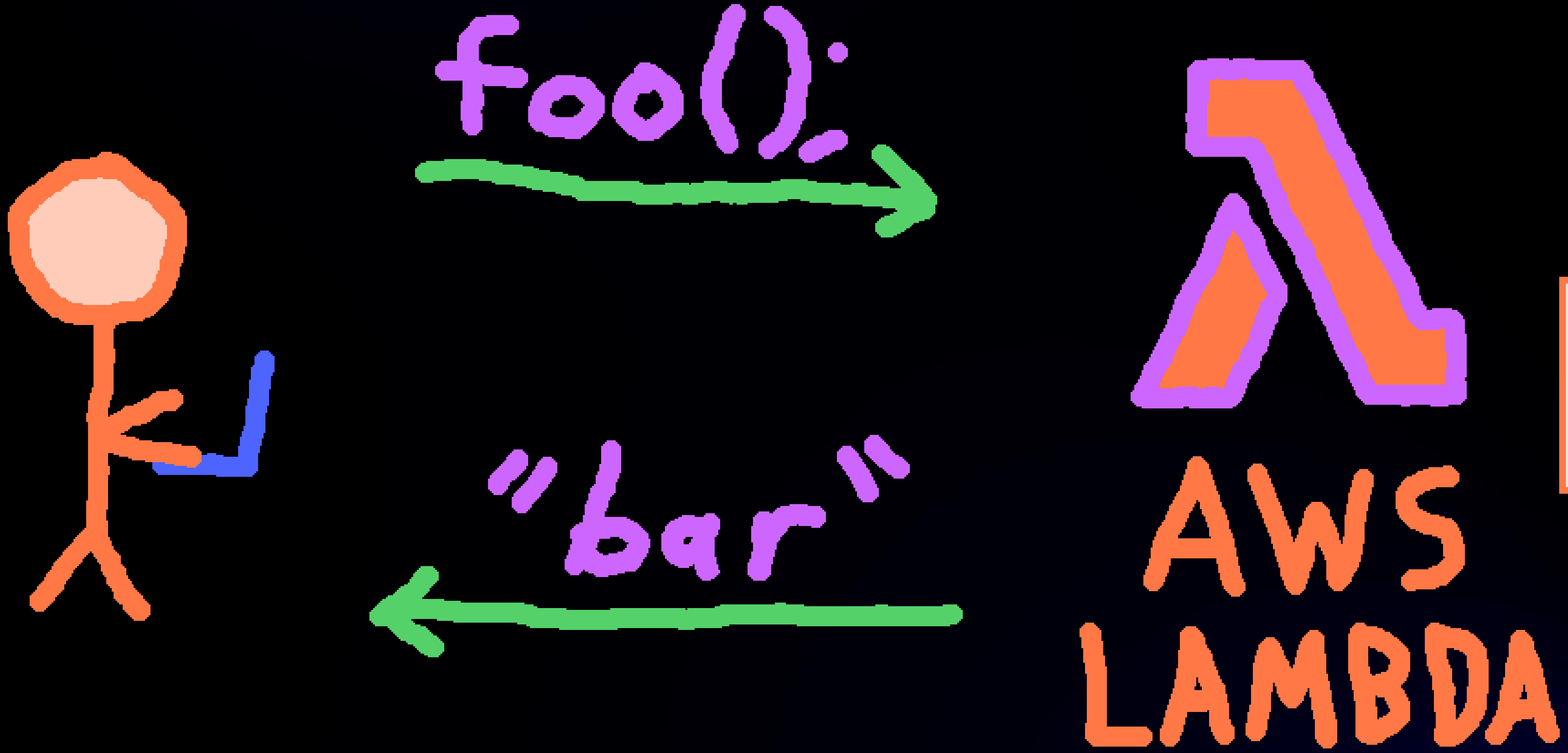


```
fn foo() {
  return "bar";
}
```

AWS Lambda functions

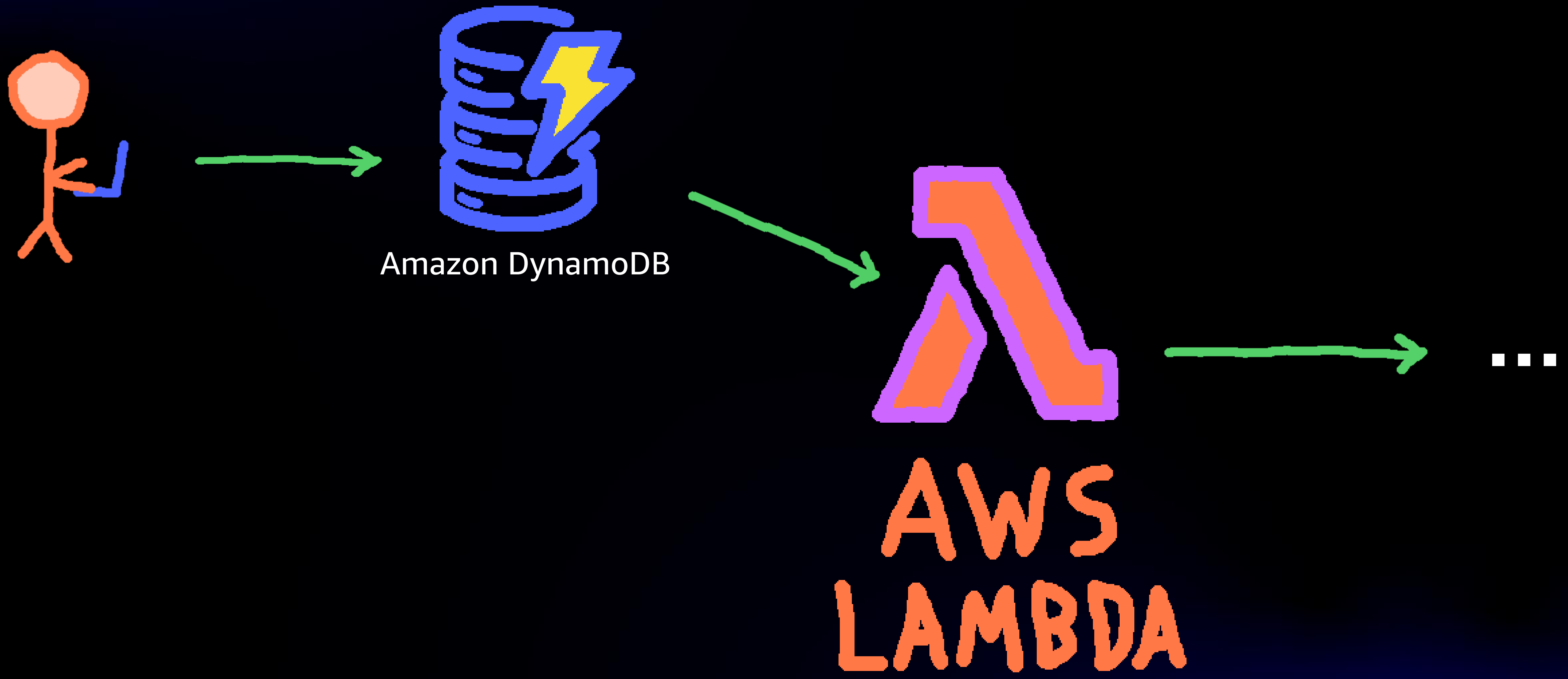
```
fn foo() {
  return "bar";
}
```

AWS Lambda functions

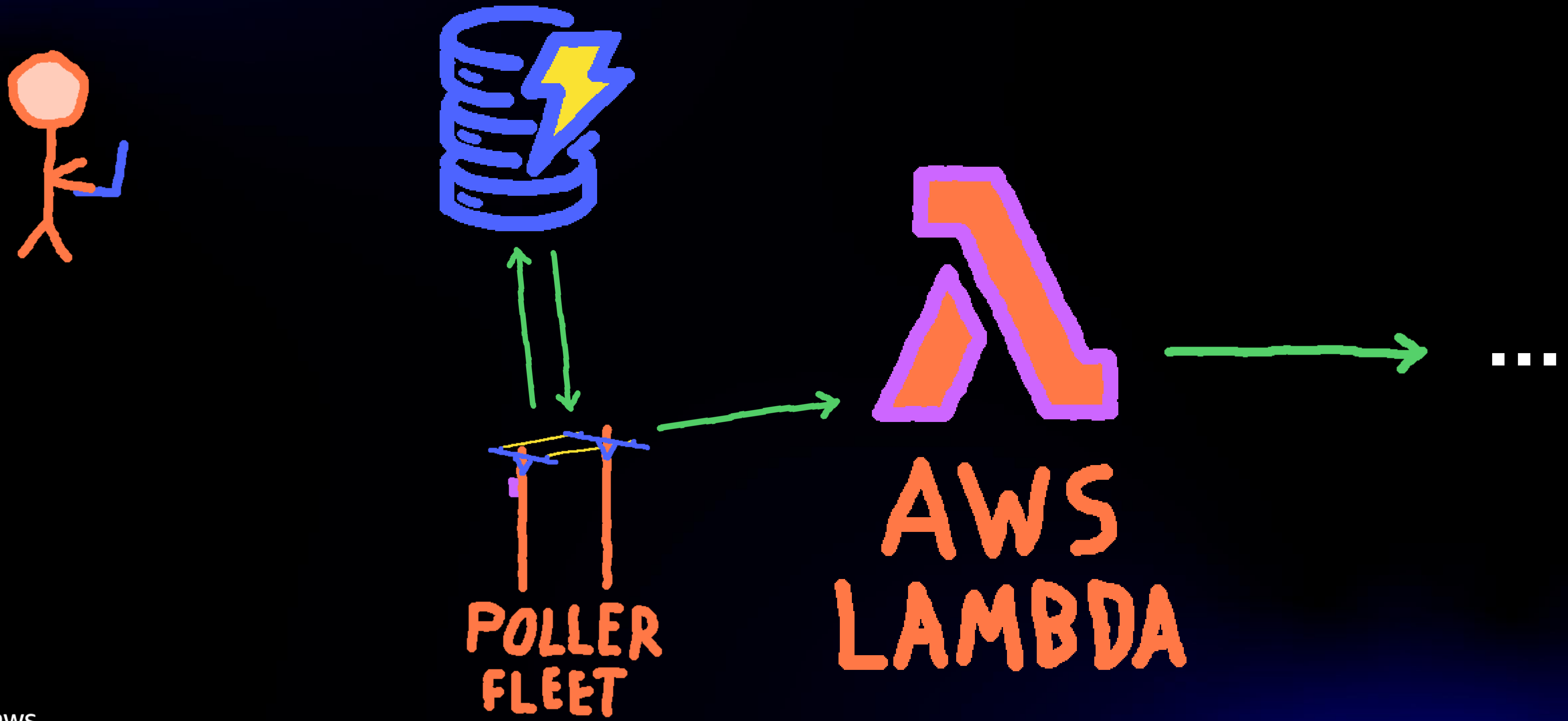


```
fn foo() {
  return "bar";
}
```

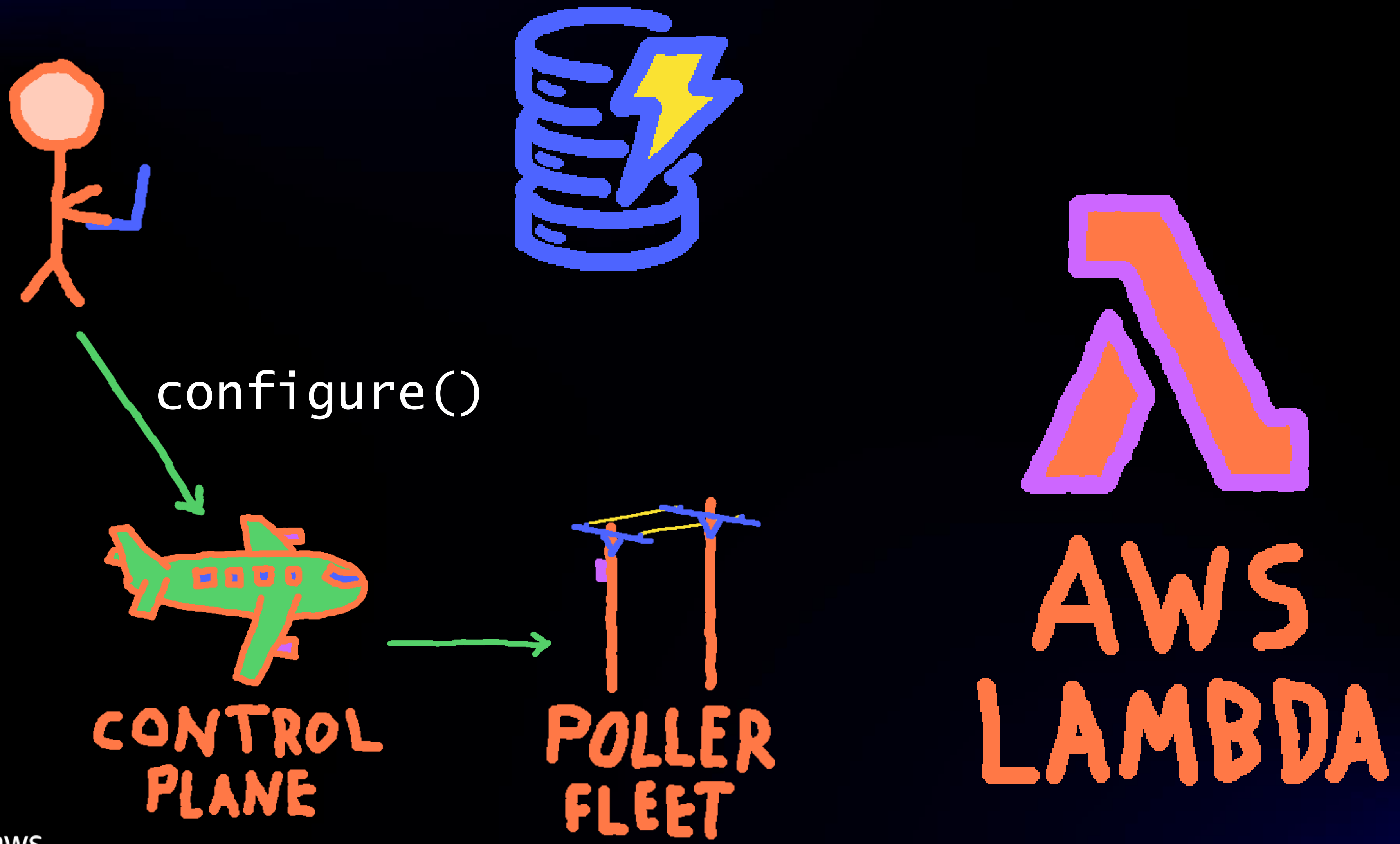
Event sources under the hood



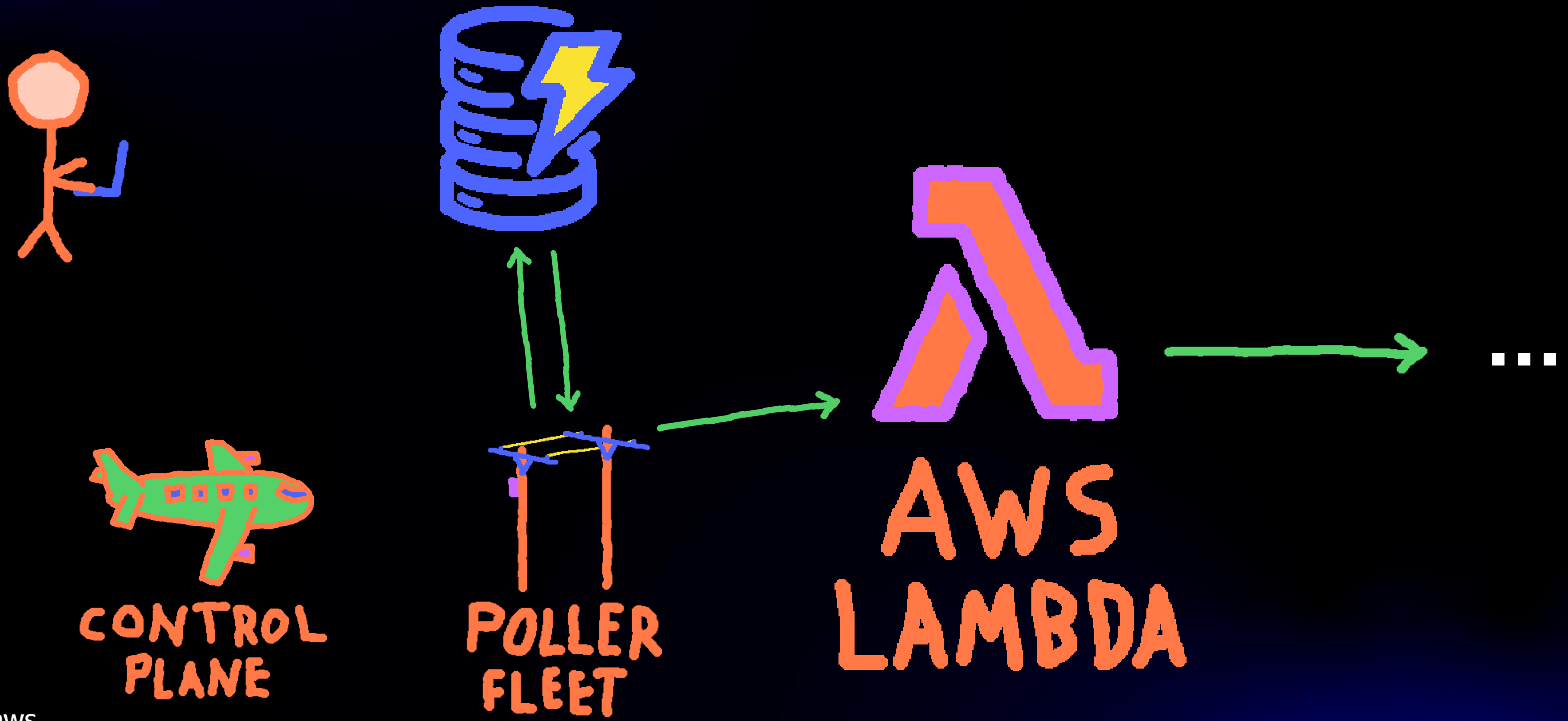
Event sources under the hood



One-time setup through a control plane



Data plane vs. control plane



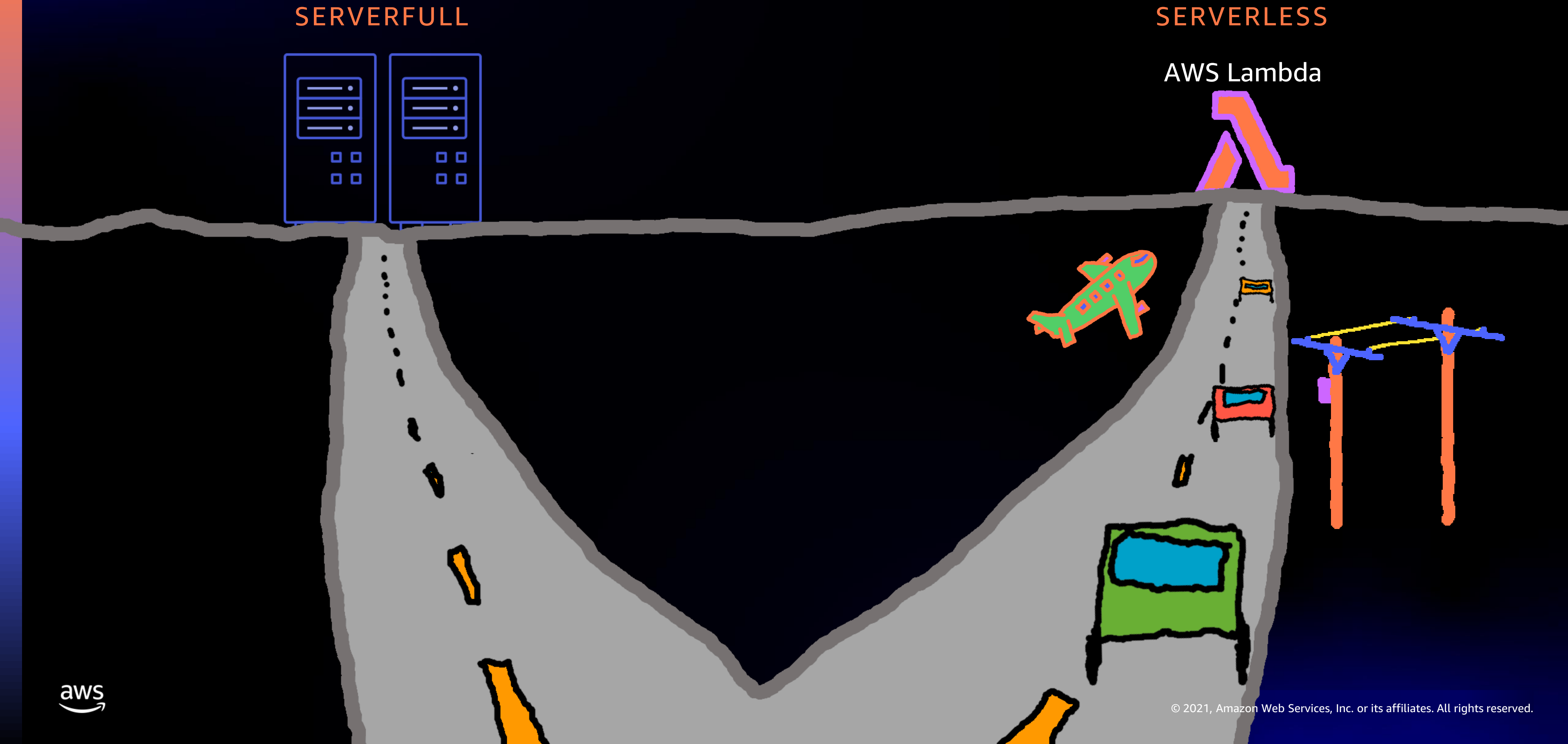
Event sources control plane APIs

CRUDL APIs for Event Source Mappings

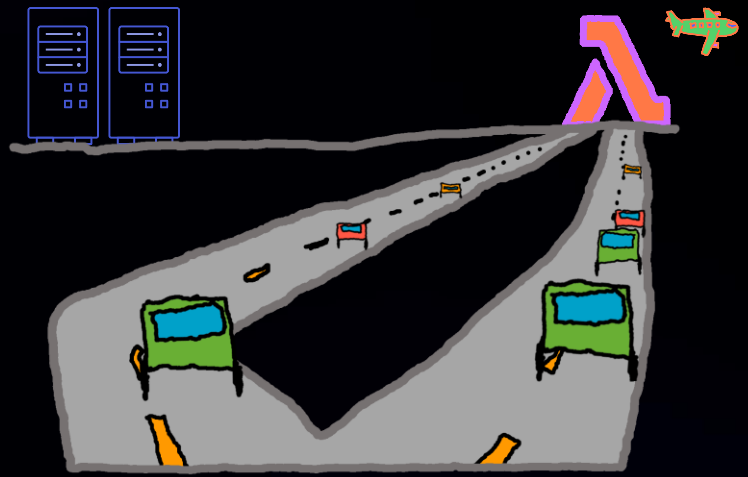
```
CreateEventSourceMapping()
GetEventSourceMapping()
UpdateEventSourceMapping()
DeleteEventSourceMapping()
ListEventSourceMapping()
```

"The road less traveled"

“The road less traveled”



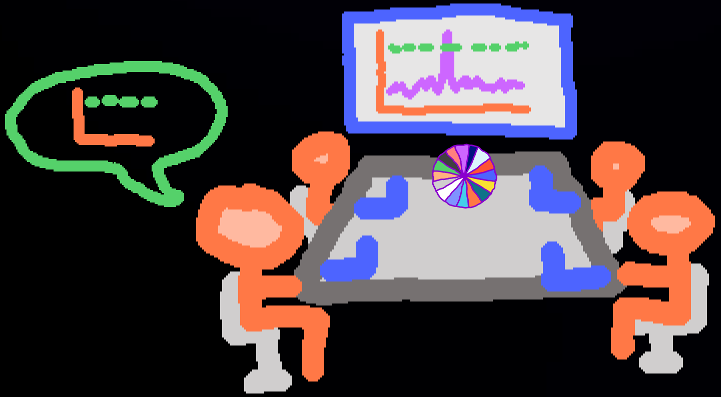
Takeaways: The architectural choice



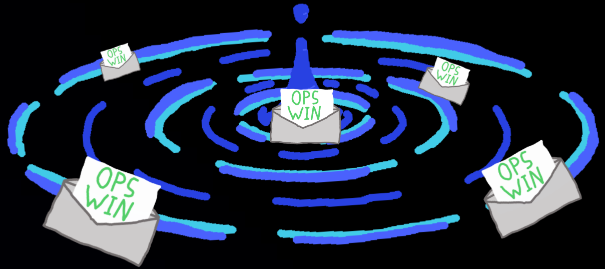
Developers gravitate toward tools that simplify operations

In conclusion

1. Culture

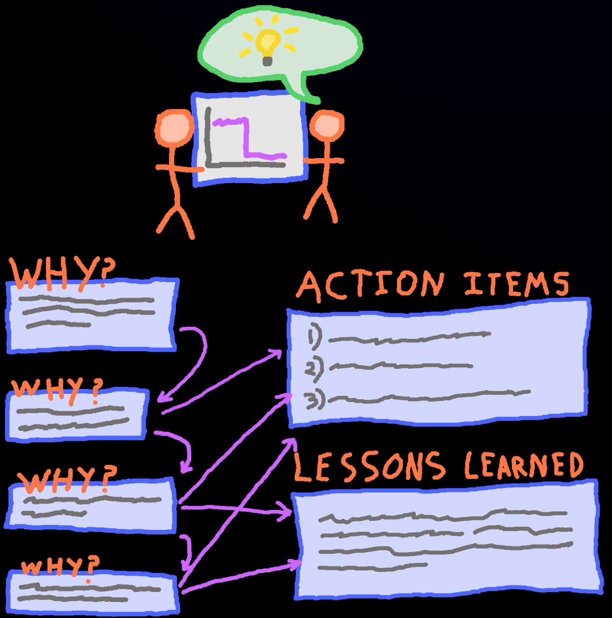


Weekly ops meetings

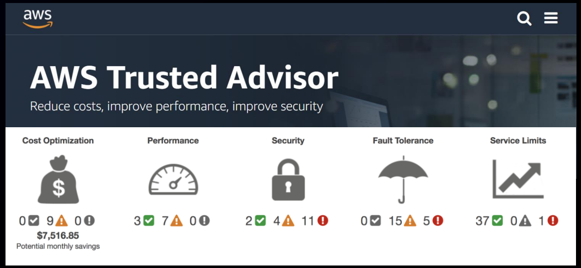


Cultural ripple effects

2. Closed loops



Retrospective



AWS Trusted Advisor

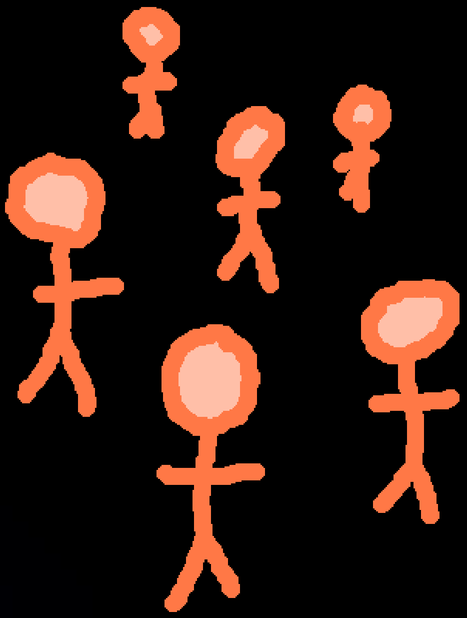
Reduce costs, improve performance, improve security

Cost Optimization	Performance	Security	Fault Tolerance	Service Limits
0 ✓ 9 ⚠ 0 ●	3 ✓ 7 ⚠ 0 ●	2 ✓ 4 ⚠ 11 ●	0 ✓ 15 ⚠ 5 ●	37 ✓ 0 ⚠ 1 ●
$7,516.85 Potential monthly savings				

AWS Well-Architected

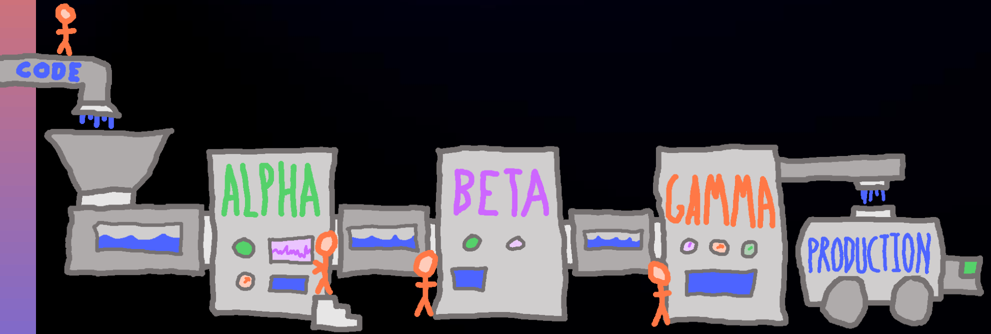
Learn, measure, and build using architectural best practices

Proactive tools

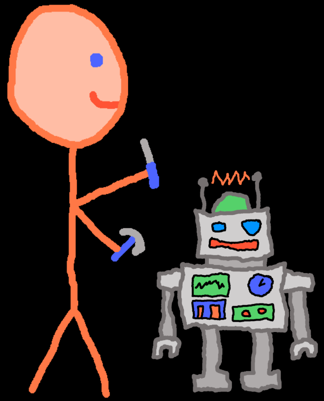


Incidents prevented

3. Operations as an investment

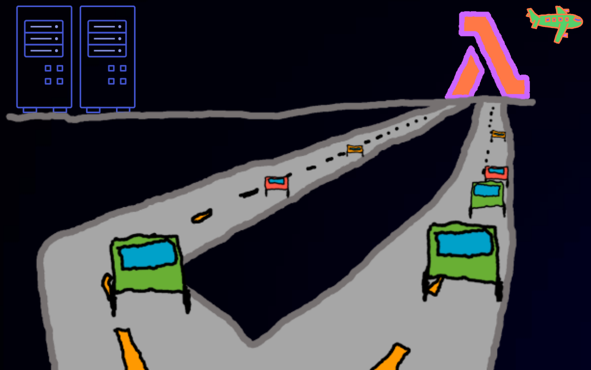


Continuous delivery



Automation of best practices

Architecting for operations



Practical takeaways

Create an ops win email list

Do a retrospective for an outage and share it broadly

Set up a recurring ops review meeting for your team or organization

Improve deployment automation and safety

Ask what operational tools or improvements teams would make with more time

Thank you!

David Yanacek

@dyanacek