@BigDana

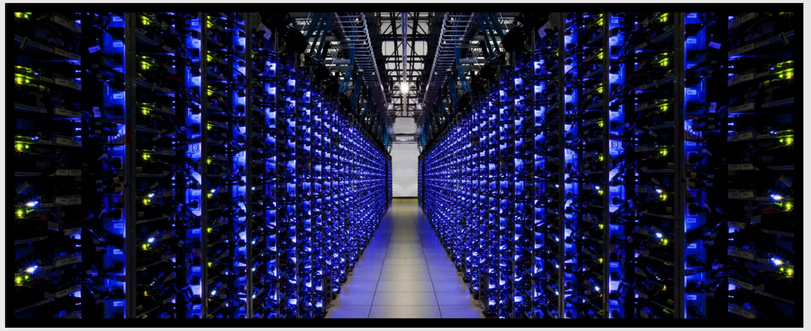
WITH GREAT SCALABILITY COMES GREAT RESPONSIBILITY

Dana Engebretson

Queen of the Amazons

@BigDana

@BigDana



@BigDana

@BigDana

Dana Engebretson
Performance Engineer
SPS Commerce

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

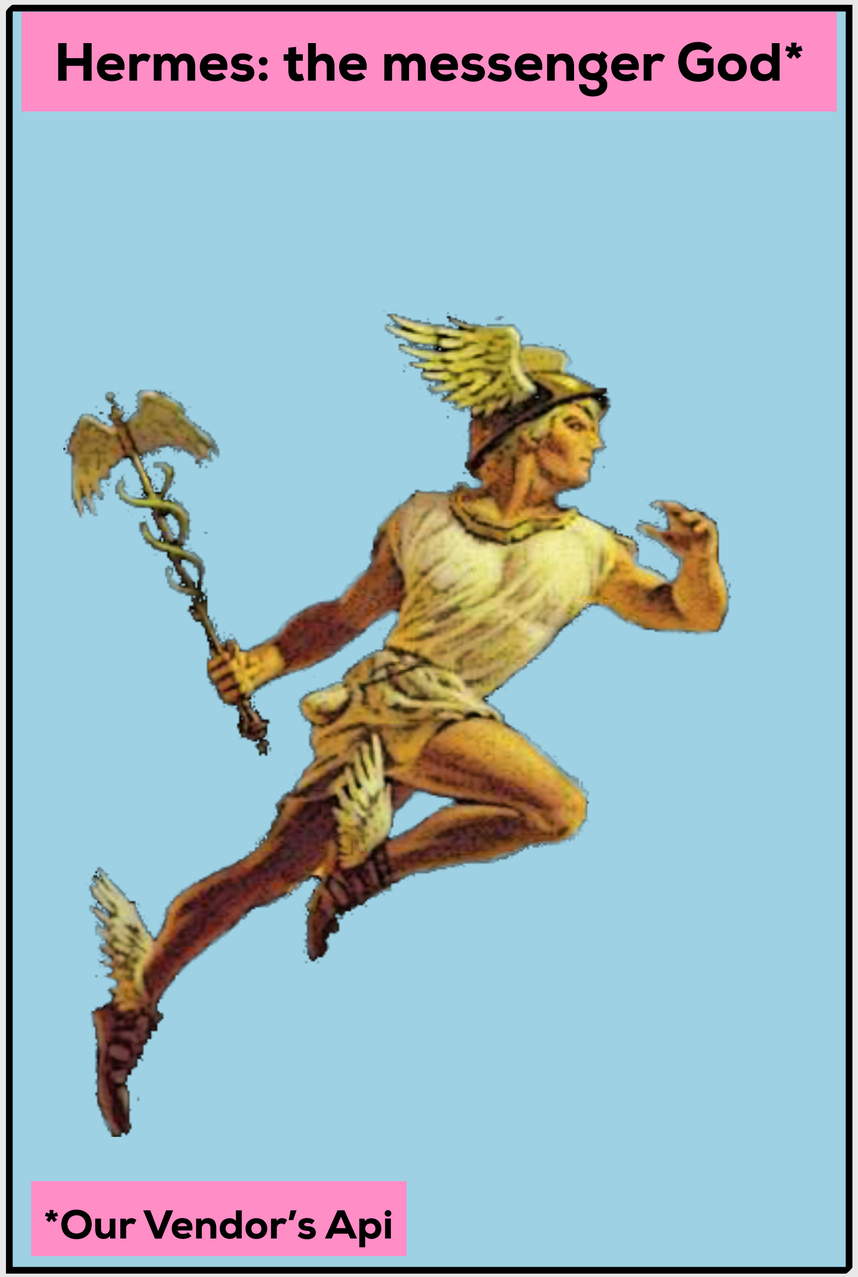
Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

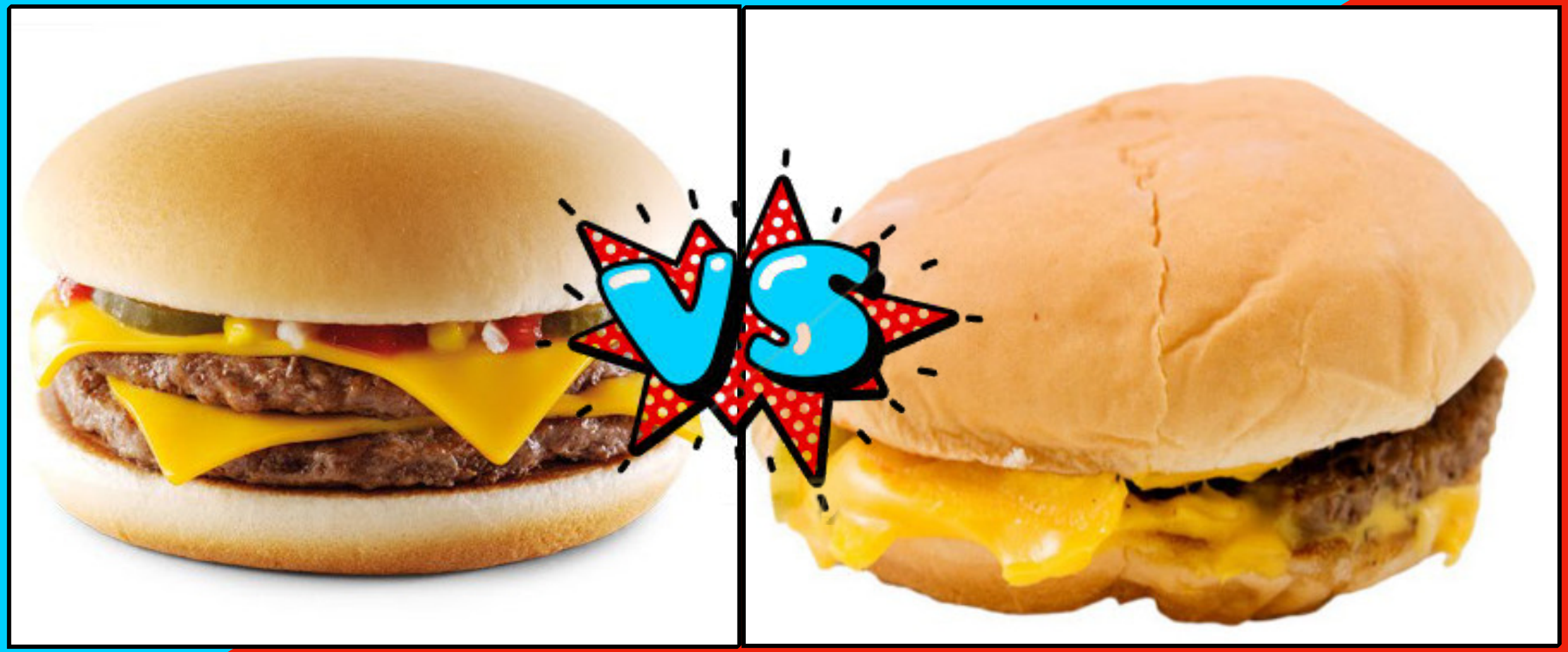
@BigDana

@BigDana


The Oracle*
*one of our monitoring vendors


Hermes: the messenger God*
*Our Vendor's Api

@BigDana



@BigDana

@BigDana

Hermes: the messenger God*

FETCH ME ALL MY DATA FOR ALL MY AMAZON WARRIORS PLEASE!

*Our Vendor's Api

@BigDana

Hermes: the messenger God*

*Our Vendor's Api

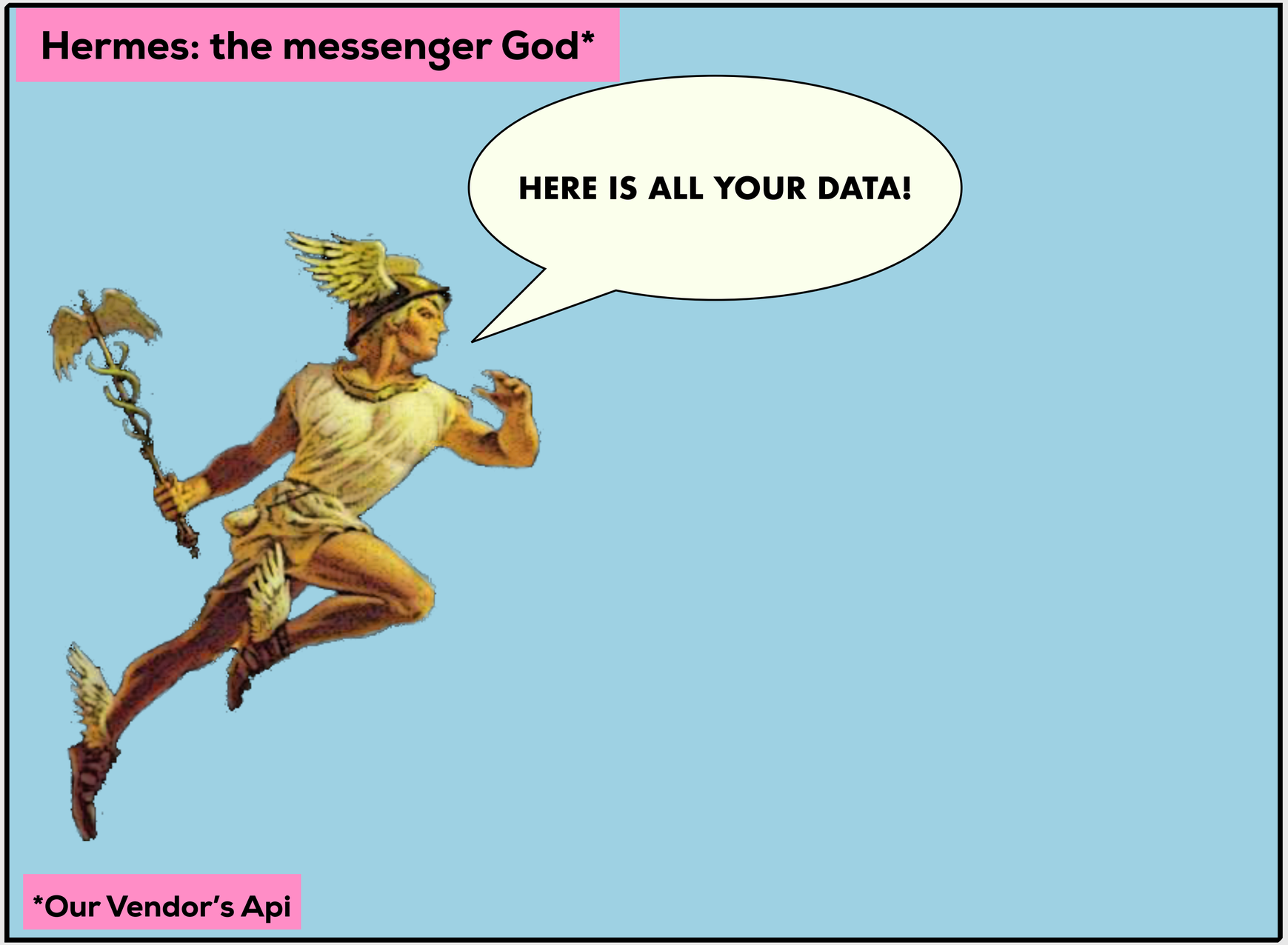
@BigDana

Hermes: the messenger God*

*Our Vendor's Api

@BigDana


The Oracle*
*one of our monitoring vendors


Hermes: the messenger God*
HERE IS ALL YOUR DATA!
*Our Vendor's Api

@BigDana

@BigDana

Hermes: the messenger God*

FETCH ME ALL MY AMAZON WARRIOR TROOPS* PLEASE!

*Our Vendor's Api

*grouped by micro-service

@BigDana


The Oracle*
*one of our monitoring vendors


Hermes: the messenger God*
*Our Vendor's Api

@BigDana


The Oracle*
*one of our monitoring vendors


Hermes: the messenger God*
*Our Vendor's Api

@BigDana


The Oracle*
*one of our monitoring vendors


Hermes: the messenger God*
HERE ARE YOUR AMAZON WARRIOR TROOPS!
troop 3
*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*

troop 1 | troop 2 | troop 3

*Our Vendor's Api

@BigDana

Hermes: the messenger God*

troop 1 | troop 2 | troop 3

FOR TROOP 1, FETCH ME MY AMAZON WARRIORS PLEASE!

*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*

troop 1 | troop 2 | troop 3

*Our Vendor's Api

@BigDana

Hermes: the messenger God*

troop 1 | troop 2 | troop 3

*Our Vendor's Api

@BigDana

Hermes: the messenger God*

HERE ARE YOUR AMAZON WARRIORS FOR TROOP 1!

troop 2

troop 3

amazon warrior 1

amazon warrior 2

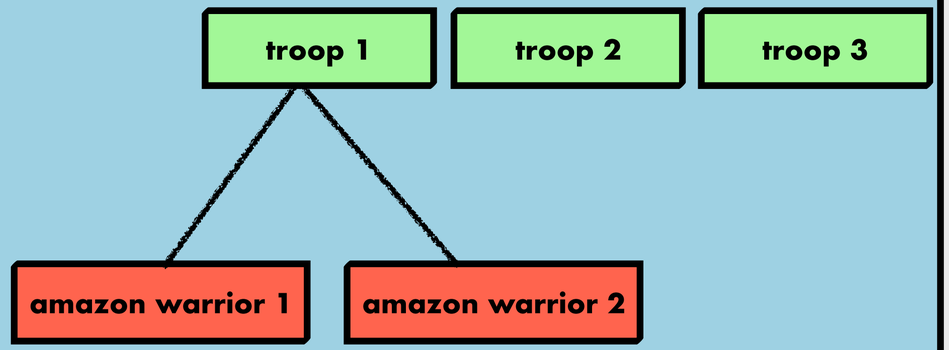
*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

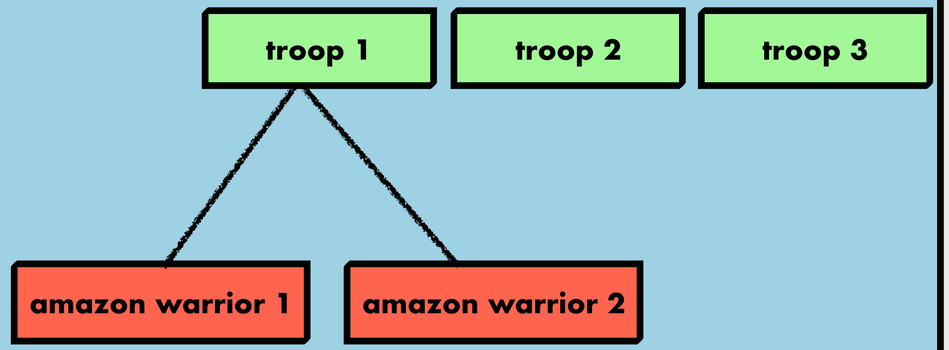
Hermes: the messenger God*



*Our Vendor's Api

@BigDana

Hermes: the messenger God*



FOR AMAZON WARRIOR 1, FETCH ME MY DATA GROUPS PLEASE!

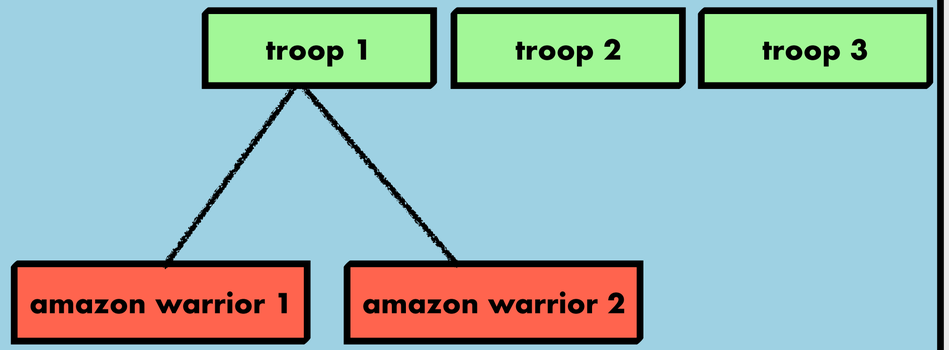
*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*



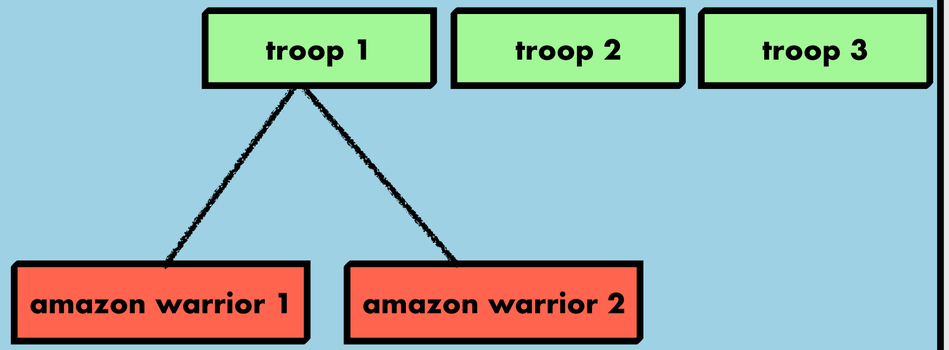
*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*



*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*

HERE ARE YOUR DATA GROUPS FOR AMAZON WARRIOR 1!

troop 3

amazon warrior 1

amazon warrior 2

data group 1

data group 2

data group 3

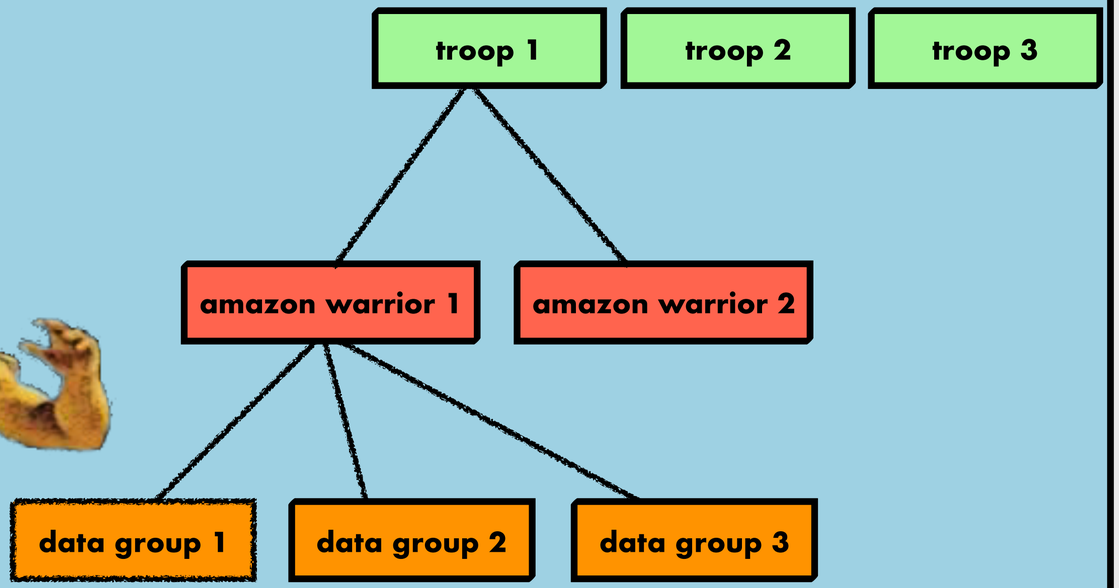
*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

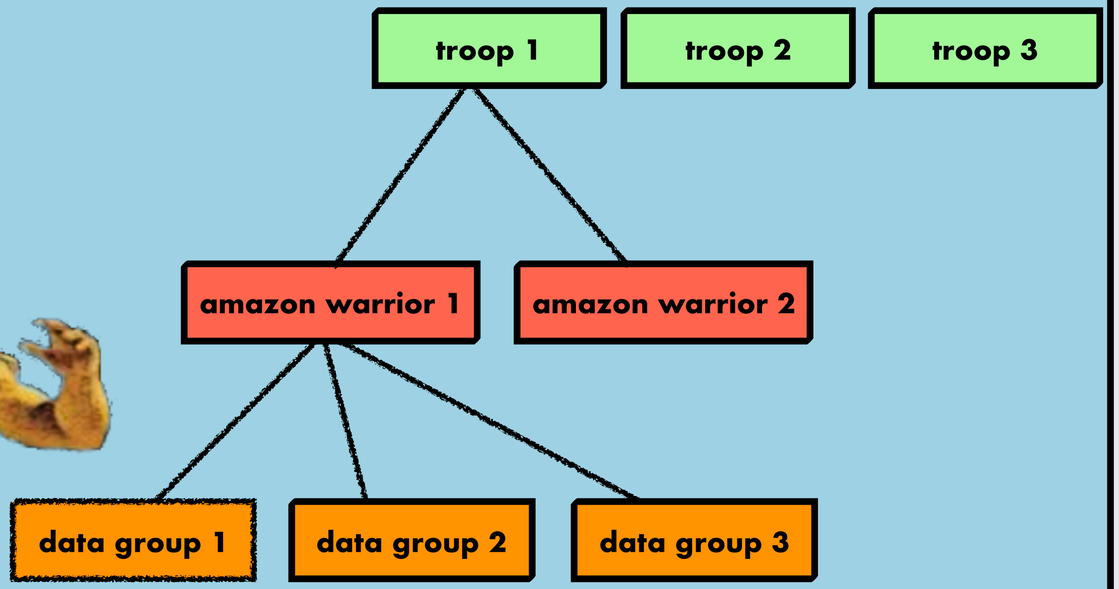
Hermes: the messenger God*



*Our Vendor's Api

@BigDana

Hermes: the messenger God*



FOR AMAZON WARRIOR 1,
AND DATA GROUP 1,
FETCH ME THE DATA PLEASE!

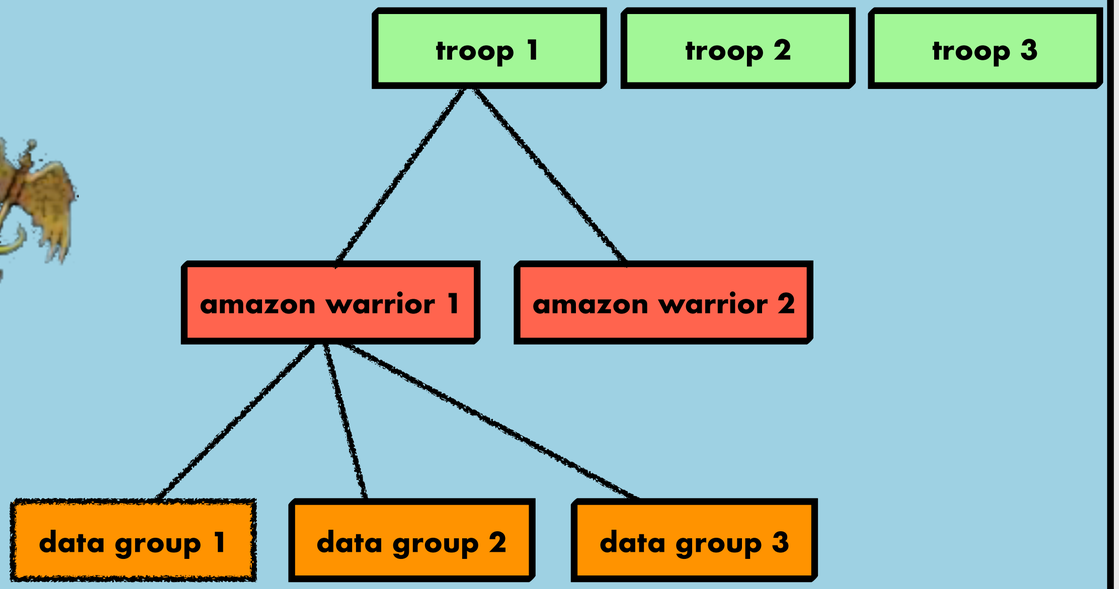
*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*



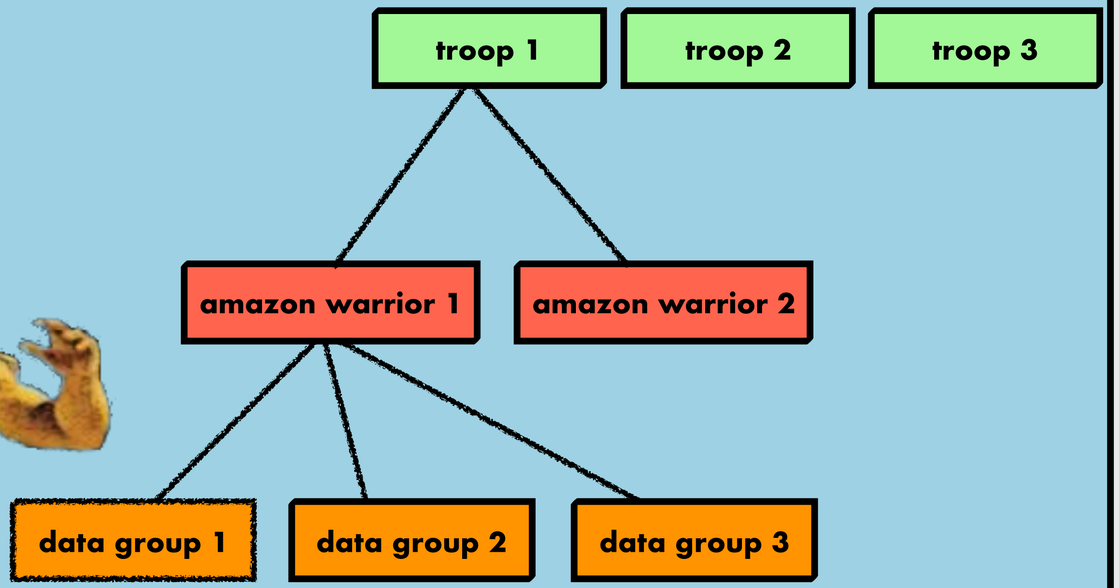
*Our Vendor's Api

@BigDana

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*

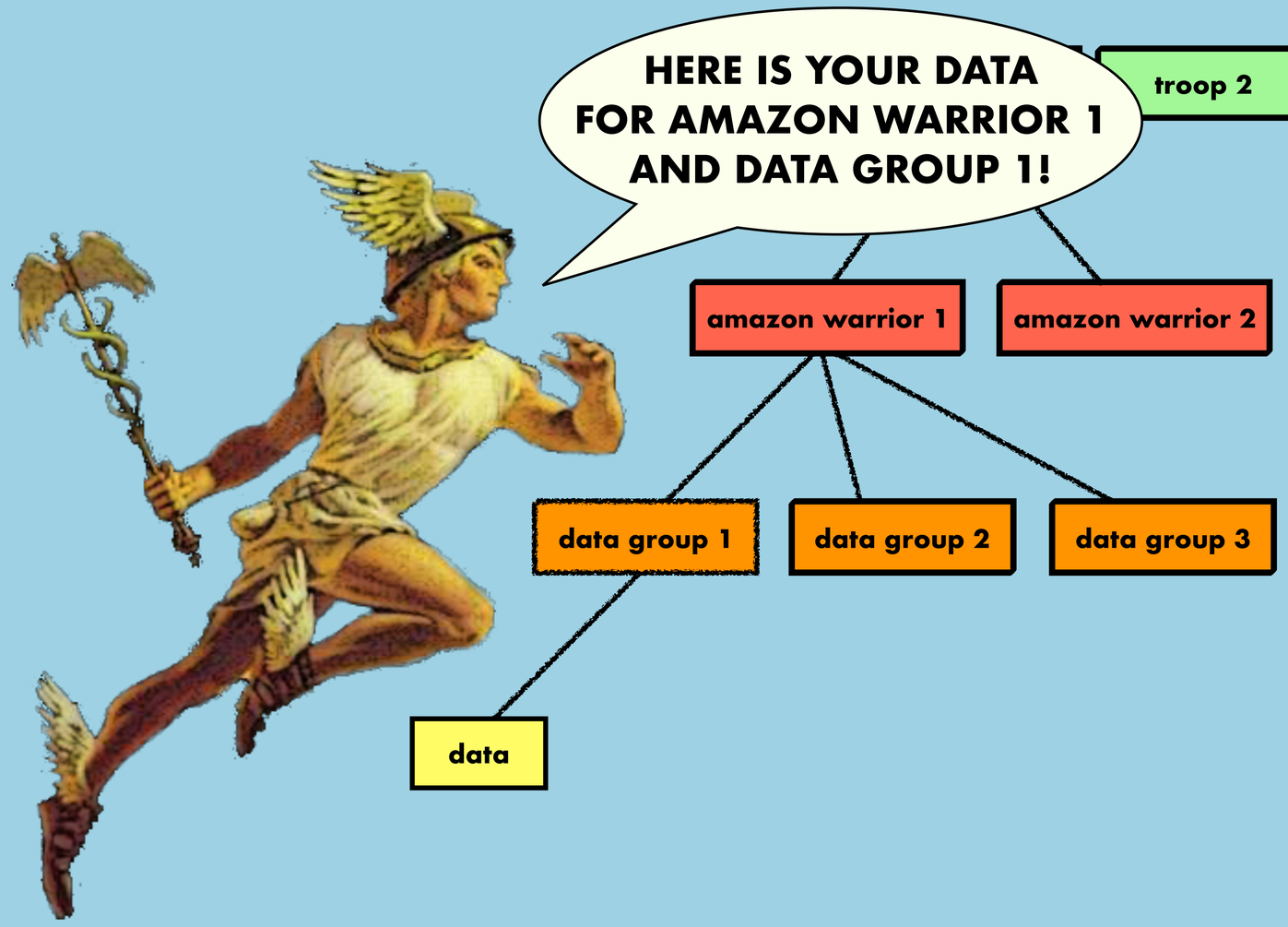


*Our Vendor's Api

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*



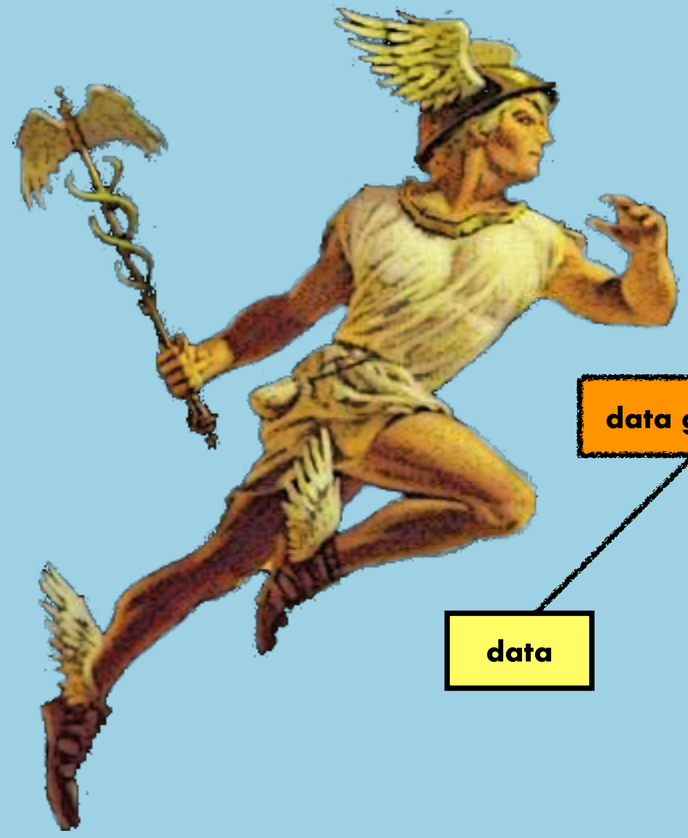
*Our Vendor's Api

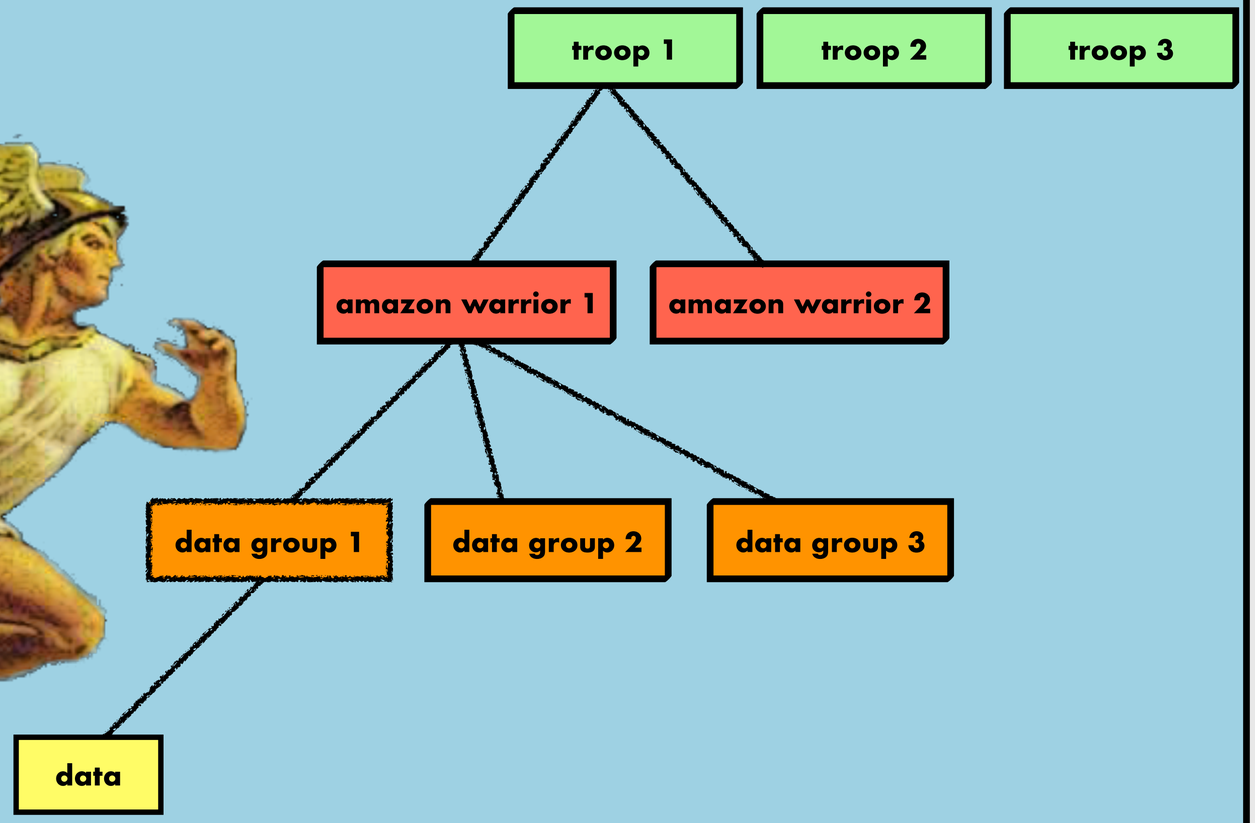
@BigDana

The Oracle*

*one of our monitoring vendors

Hermes: the messenger God*

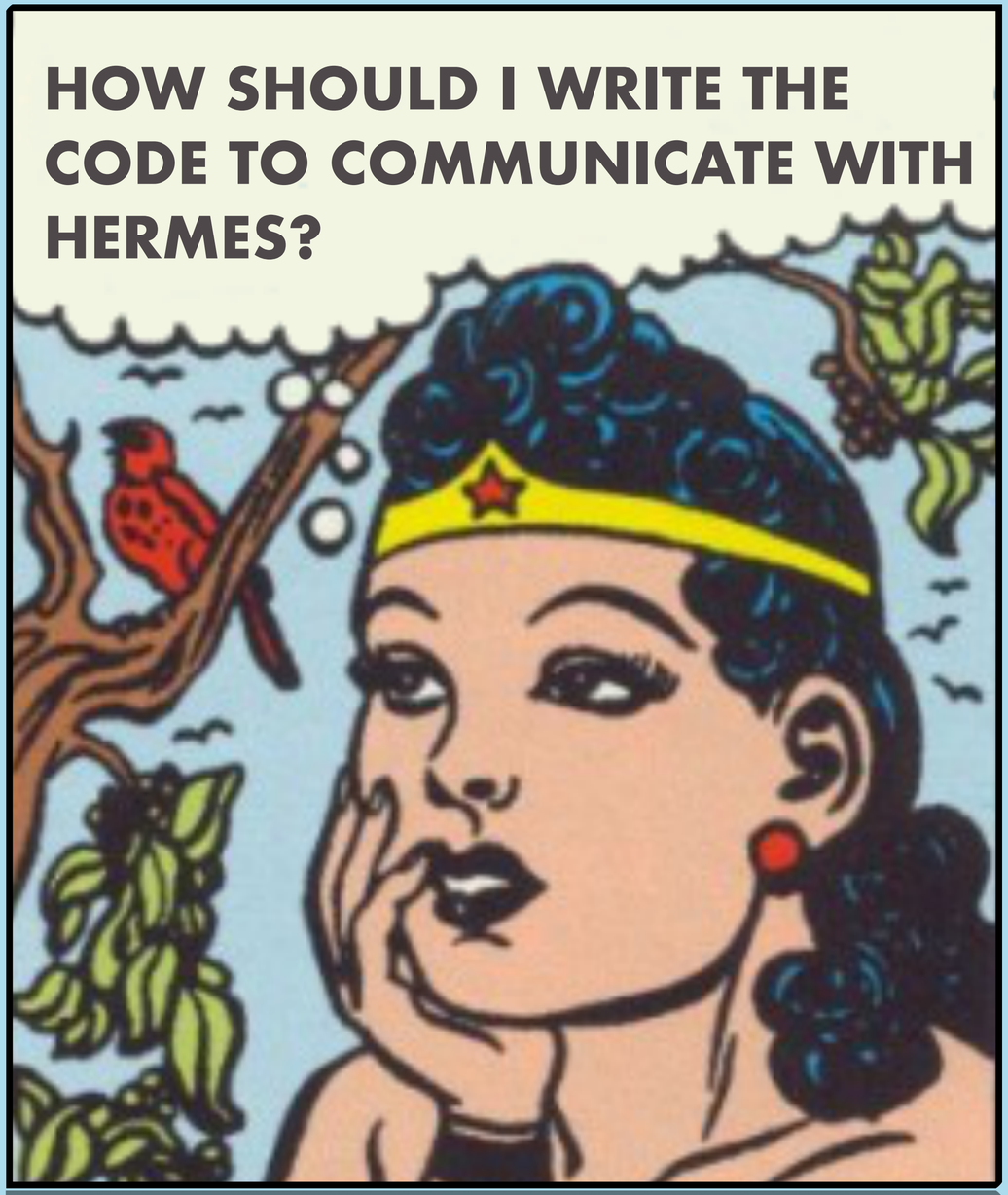




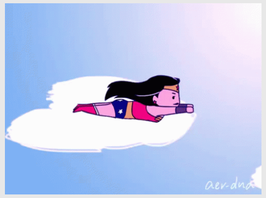
*Our Vendor's Api

@BigDana

35,000-65,000 API CALLS!



Building a Data Pipeline



Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Initial Solution:

Python Multiprocessing on an EC2 instance - continuously running

Easy for me to implement

not cheap

horizontal scaling seemed overkill

on-going maintenance
and management

Initial Solution:

Python Multiprocessing on an EC2 instance - intermittently running

Easy for me to implement

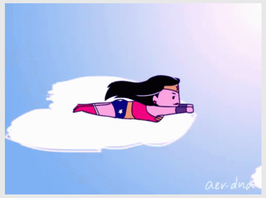
cheaper

still not cheap enough

horizontal scaling seemed overkill

on-going maintenance
and management

Building a Data Pipeline



Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Revision:

Python Multiprocessing on ~~an EC2 instance~~ a Spot instance

significantly cheaper

You need to configure how to restart where the previous process left off

Should I try Serverless?

Owning your own server : Owning your own car

Owning your own car: Owning your own server

The Cloud: Leasing a car

@BigDana

Owning your own server : Owning your own car

@BigDana

The Cloud : Leasing a car

Spot Instances: Renting a car

Owning your own server: Owning your own car

The Cloud: Leasing a car

Lambdas: car2go

Spot Instances: Renting a car

@BigDana

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

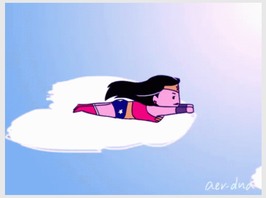
Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline



Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

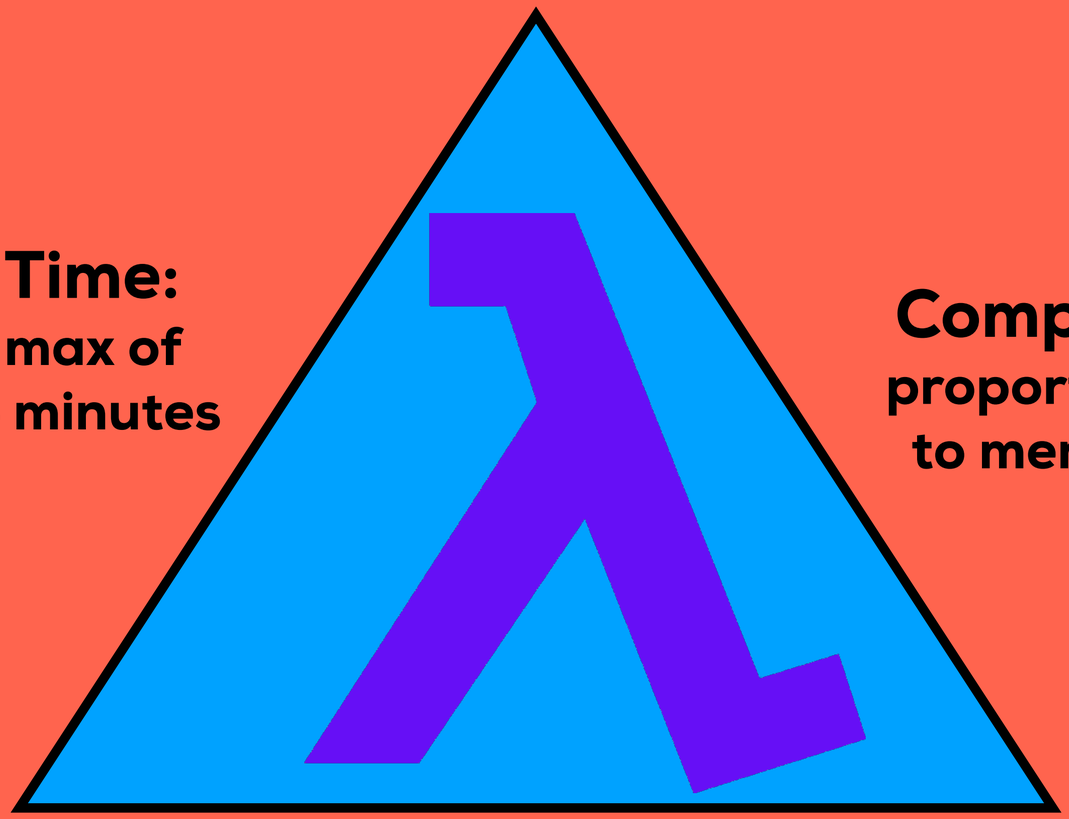
Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Lambda: a Function as a Service

@BigDana

Lambda: a Function as a Service

Time:
max of
5 minutes



Compute:
proportional
to memory

Memory:
max of 1536 MB

@BigDana

Lambda: Function as a Service

Compute

Dependency
Zip File size

250 MB uncompressed
code/dependencies

Memory

Time

```
21  build:
22      docker build --no-cache -t flowers .
23
24  dev:
25      docker run -it -v `pwd`:/code flowers bash
26
27  libs:
28      conda install --yes --file /code/serverless/get_data_lambda/requirements.txt -v
29      mkdir -p serverless/get_data_lambda/libs
30      cp -a /opt/conda/pkgs/. /code/serverless/get_data_lambda/libs
31
32  zip:
33      cp /code/serverless/get_data_lambda/get_data.py /code/serverless/get_data_lambda/libs/
34      rm -rf /code/tmpdir
35      mkdir -p /code/tmpdir
36      cp -r /code/serverless/get_data_lambda/libs/* /code/tmpdir
37      cd /code/tmpdir
38      zip deployment.zip *
```

```
21  build:
22      docker build --no-cache -t flowers .
23
24  dev:
25      docker run -it -v `pwd`:/code flowers bash
26
27  libs:
28      conda install --yes --file /code/serverless/get_data_lambda/requirements.txt -v
29      mkdir -p serverless/get_data_lambda/libs
30      cp -a /opt/conda/pkgs/. /code/serverless/get_data_lambda/libs
31
32  zip:
33      cp /code/serverless/get_data_lambda/get_data.py /code/serverless/get_data_lambda/libs/
34      rm -rf /code/tmpdir
35      mkdir -p /code/tmpdir
36      cp -r /code/serverless/get_data_lambda/libs/* /code/tmpdir
37      cd /code/tmpdir
38      zip deployment.zip *
```

```
21  build:
22      docker build --no-cache -t flowers .
23
24  dev:
25      docker run -it -v `pwd`:/code flowers bash
26
27  libs:
28      conda install --yes --file /code/serverless/get_data_lambda/requirements.txt -v
29      mkdir -p serverless/get_data_lambda/libs
30      cp -a /opt/conda/pkgs/. /code/serverless/get_data_lambda/libs
31
32  zip:
33      cp /code/serverless/get_data_lambda/get_data.py /code/serverless/get_data_lambda/libs/
34      rm -rf /code/tmpdir
35      mkdir -p /code/tmpdir
36      cp -r /code/serverless/get_data_lambda/libs/* /code/tmpdir
37      cd /code/tmpdir
38      zip deployment.zip *
```

```
21  build:
22      docker build --no-cache -t flowers .
23
24  dev:
25      docker run -it -v `pwd`:/code flowers bash
26
27  libs:
28      conda install --yes --file /code/serverless/get_data_lambda/requirements.txt -v
29      mkdir -p serverless/get_data_lambda/libs
30      cp -a /opt/conda/pkgs/. /code/serverless/get_data_lambda/libs
31
32  zip:
33      cp /code/serverless/get_data_lambda/get_data.py /code/serverless/get_data_lambda/libs/
34      rm -rf /code/tmpdir
35      mkdir -p /code/tmpdir
36      cp -r /code/serverless/get_data_lambda/libs/* /code/tmpdir
37      cd /code/tmpdir
38      zip deployment.zip *
```

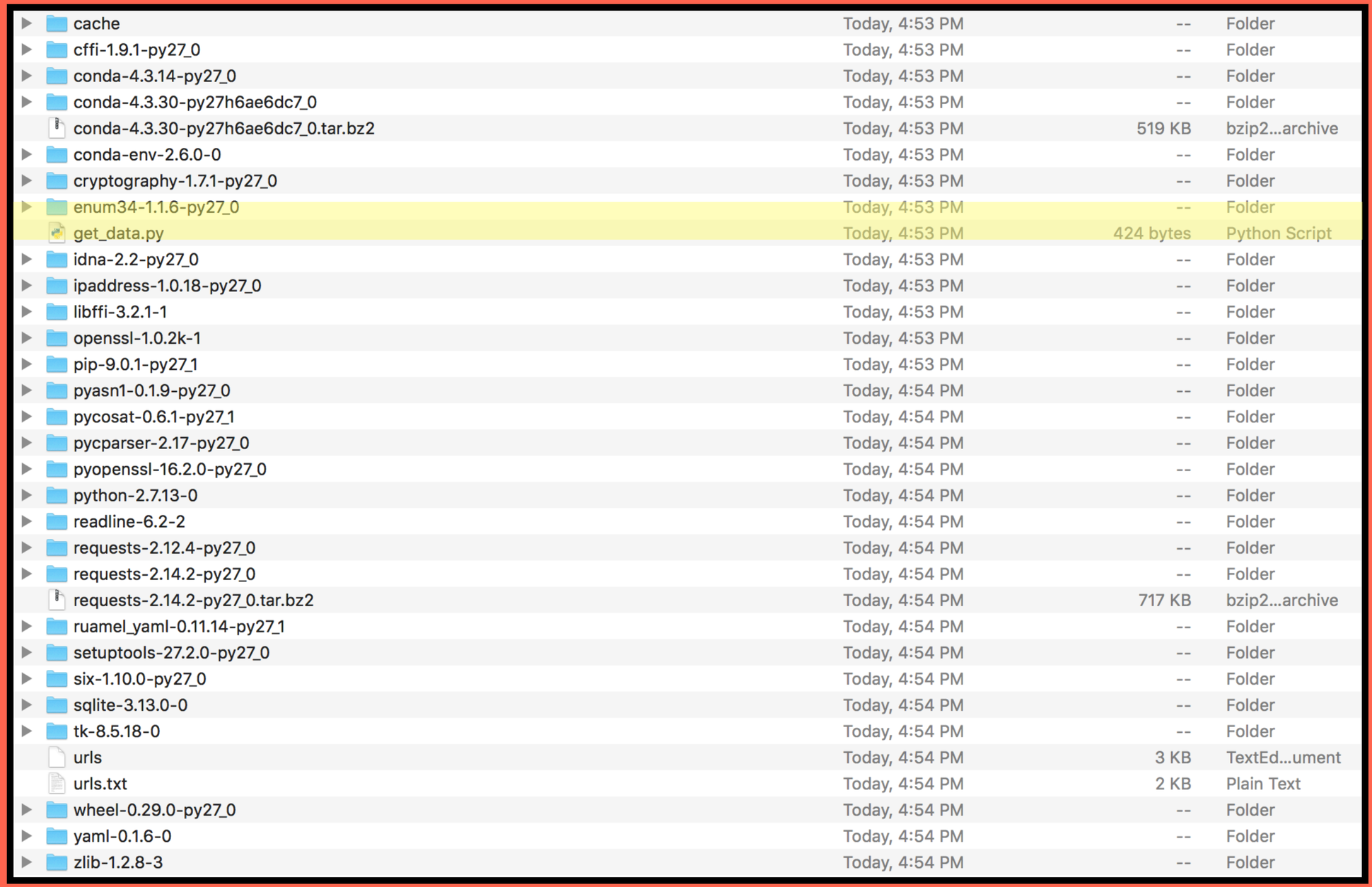
```
build:
    docker build --no-cache -t flowers .

dev:
    docker run -it -v `pwd`:/code flowers bash

libs:
    conda install --yes --file /code/serverless/get_data_lambda/requirements.txt -v
    mkdir -p serverless/get_data_lambda/libs
    cp -a /opt/conda/pkgs/. /code/serverless/get_data_lambda/libs

zip:
    cp /code/serverless/get_data_lambda/get_data.py /code/serverless/get_data_lambda/libs/
    rm -rf /code/tmpdir
    mkdir -p /code/tmpdir
    cp -r /code/serverless/get_data_lambda/libs/* /code/tmpdir
    cd /code/tmpdir
    zip deployment.zip *
```

cache	Today, 4:53 PM	--	Folder
cffi-1.9.1-py27_0	Today, 4:53 PM	--	Folder
conda-4.3.14-py27_0	Today, 4:53 PM	--	Folder
conda-4.3.30-py27h6ae6dc7_0	Today, 4:53 PM	--	Folder
conda-4.3.30-py27h6ae6dc7_0.tar.bz2	Today, 4:53 PM	519 KB	bzip2...archive
conda-env-2.6.0-0	Today, 4:53 PM	--	Folder
cryptography-1.7.1-py27_0	Today, 4:53 PM	--	Folder
enum34-1.1.6-py27_0	Today, 4:53 PM	--	Folder
get_data.py	Today, 4:53 PM	424 bytes	Python Script
idna-2.2-py27_0	Today, 4:53 PM	--	Folder
ipaddress-1.0.18-py27_0	Today, 4:53 PM	--	Folder
libffi-3.2.1-1	Today, 4:53 PM	--	Folder
openssl-1.0.2k-1	Today, 4:53 PM	--	Folder
pip-9.0.1-py27_1	Today, 4:53 PM	--	Folder
pyasn1-0.1.9-py27_0	Today, 4:54 PM	--	Folder
pycosat-0.6.1-py27_1	Today, 4:54 PM	--	Folder
pycparser-2.17-py27_0	Today, 4:54 PM	--	Folder
pyopenssl-16.2.0-py27_0	Today, 4:54 PM	--	Folder
python-2.7.13-0	Today, 4:54 PM	--	Folder
readline-6.2-2	Today, 4:54 PM	--	Folder
requests-2.12.4-py27_0	Today, 4:54 PM	--	Folder
requests-2.14.2-py27_0	Today, 4:54 PM	--	Folder
requests-2.14.2-py27_0.tar.bz2	Today, 4:54 PM	717 KB	bzip2...archive
ruamel_yaml-0.11.14-py27_1	Today, 4:54 PM	--	Folder
setuptools-27.2.0-py27_0	Today, 4:54 PM	--	Folder
six-1.10.0-py27_0	Today, 4:54 PM	--	Folder
sqlite-3.13.0-0	Today, 4:54 PM	--	Folder
tk-8.5.18-0	Today, 4:54 PM	--	Folder
urls	Today, 4:54 PM	3 KB	TextEd...ument
urls.txt	Today, 4:54 PM	2 KB	Plain Text
wheel-0.29.0-py27_0	Today, 4:54 PM	--	Folder
yaml-0.1.6-0	Today, 4:54 PM	--	Folder
zlib-1.2.8-3	Today, 4:54 PM	--	Folder



cache	Today, 4:53 PM	--	Folder
cffi-1.9.1-py27_0	Today, 4:53 PM	--	Folder
conda-4.3.14-py27_0	Today, 4:53 PM	--	Folder
conda-4.3.30-py27h6ae6dc7_0	Today, 4:53 PM	--	Folder
conda-4.3.30-py27h6ae6dc7_0.tar.bz2	Today, 4:53 PM	519 KB	bzip2...archive
conda-env-2.6.0-0	Today, 4:53 PM	--	Folder
cryptography-1.7.1-py27_0	Today, 4:53 PM	--	Folder
enum34-1.1.6-py27_0	Today, 4:53 PM	--	Folder
get_data.py	Today, 4:53 PM	424 bytes	Python Script
idna-2.2-py27_0	Today, 4:53 PM	--	Folder
ipaddress-1.0.18-py27_0	Today, 4:53 PM	--	Folder
libffi-3.2.1-1	Today, 4:53 PM	--	Folder
openssl-1.0.2k-1	Today, 4:53 PM	--	Folder
pip-9.0.1-py27_1	Today, 4:53 PM	--	Folder
pyasn1-0.1.9-py27_0	Today, 4:54 PM	--	Folder
pycosat-0.6.1-py27_1	Today, 4:54 PM	--	Folder
pycparser-2.17-py27_0	Today, 4:54 PM	--	Folder
pyopenssl-16.2.0-py27_0	Today, 4:54 PM	--	Folder
python-2.7.13-0	Today, 4:54 PM	--	Folder
readline-6.2-2	Today, 4:54 PM	--	Folder
requests-2.12.4-py27_0	Today, 4:54 PM	--	Folder
requests-2.14.2-py27_0	Today, 4:54 PM	--	Folder
requests-2.14.2-py27_0.tar.bz2	Today, 4:54 PM	717 KB	bzip2...archive
ruamel_yaml-0.11.14-py27_1	Today, 4:54 PM	--	Folder
setuptools-27.2.0-py27_0	Today, 4:54 PM	--	Folder
six-1.10.0-py27_0	Today, 4:54 PM	--	Folder
sqlite-3.13.0-0	Today, 4:54 PM	--	Folder
tk-8.5.18-0	Today, 4:54 PM	--	Folder
urls	Today, 4:54 PM	3 KB	TextEd...ument
urls.txt	Today, 4:54 PM	2 KB	Plain Text
wheel-0.29.0-py27_0	Today, 4:54 PM	--	Folder
yaml-0.1.6-0	Today, 4:54 PM	--	Folder
zlib-1.2.8-3	Today, 4:54 PM	--	Folder

@BigDana

Lambda: Function as a Service

Compute

fastparquet
didn’t fit

Dependency Zip File size

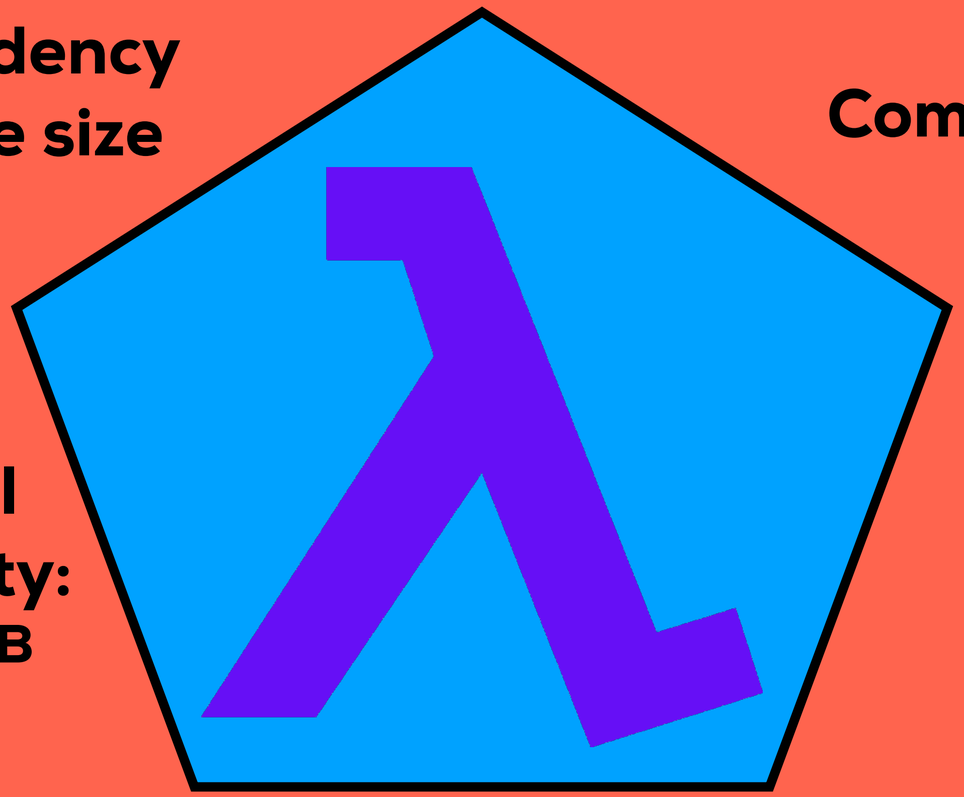
250 MB uncompressed code/dependencies

Memory

Time

@BigDana

Lambda: Function as a Service



Dependency
Zip File size

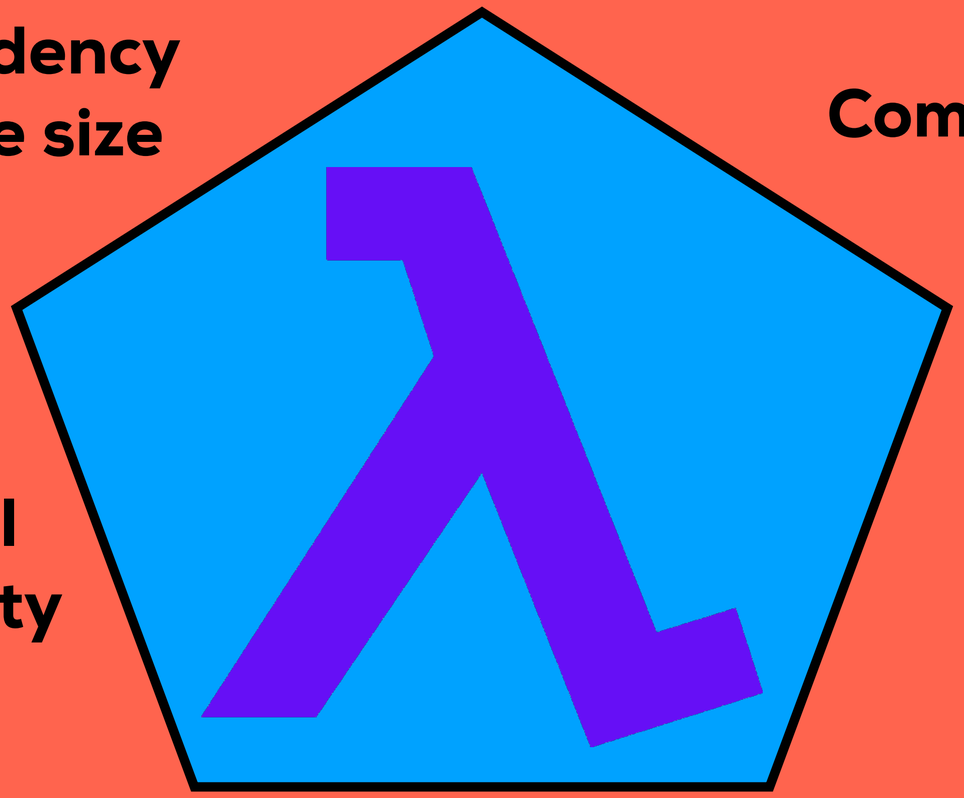
Compute

Ephemeral
Disk Capacity:
max of 512 MB

Memory

Time

Lambda: Function as a Service



Dependency
Zip File size

Compute

Ephemeral
Disk Capacity

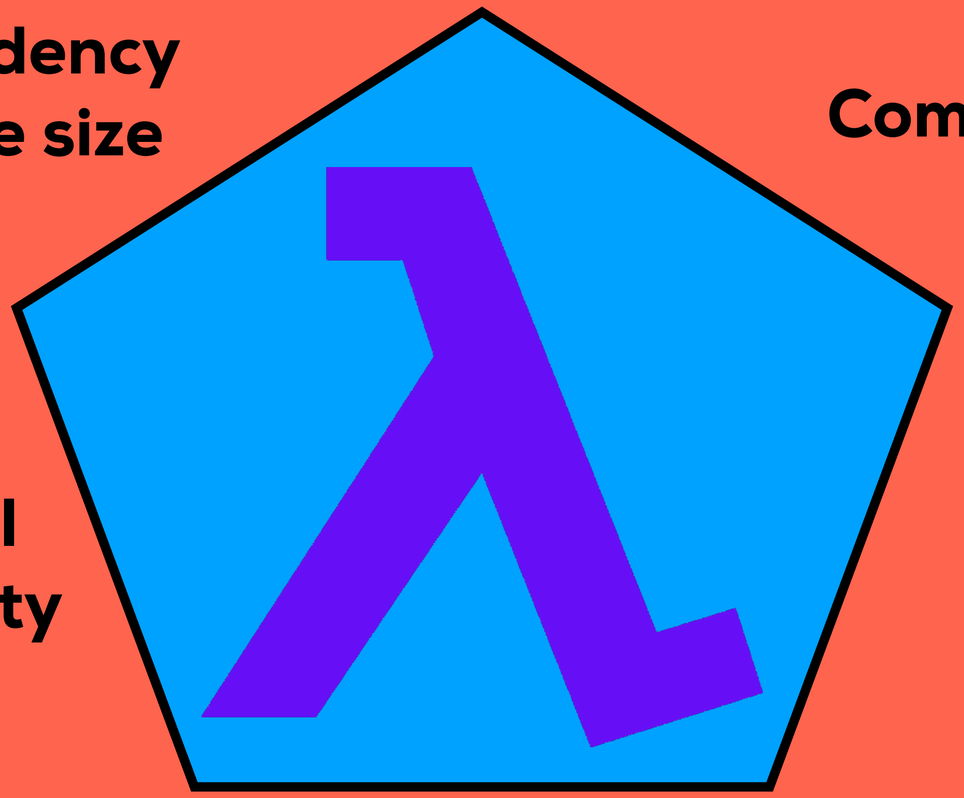
Memory

Time

concurrent executions:
default max of 1000 per account

@BigDana

Lambda: Function as a Service



Dependency
Zip File size

Compute

# of file
descriptors

# of processes
and threads

Ephemeral
Disk Capacity

Memory

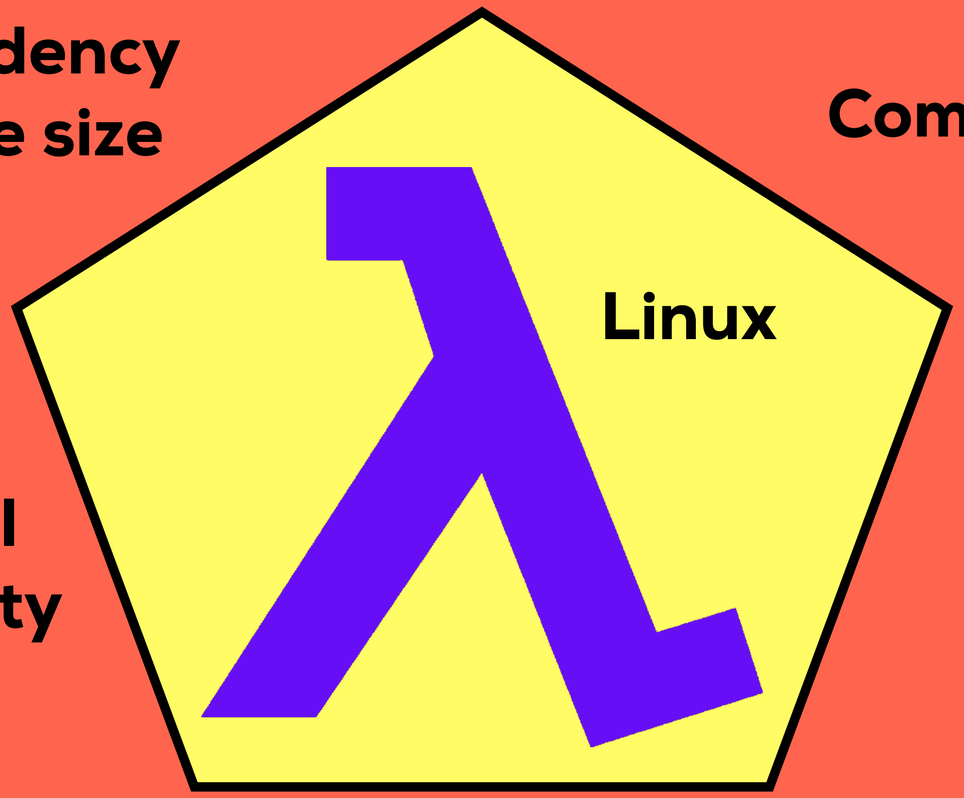
concurrent executions

Time

Invoke
request body
payload size

@BigDana

Lambda: Function as a Service



Dependency
Zip File size

Compute

# of file
descriptors

# of processes
and threads

Ephemeral
Disk Capacity

Memory

concurrent executions

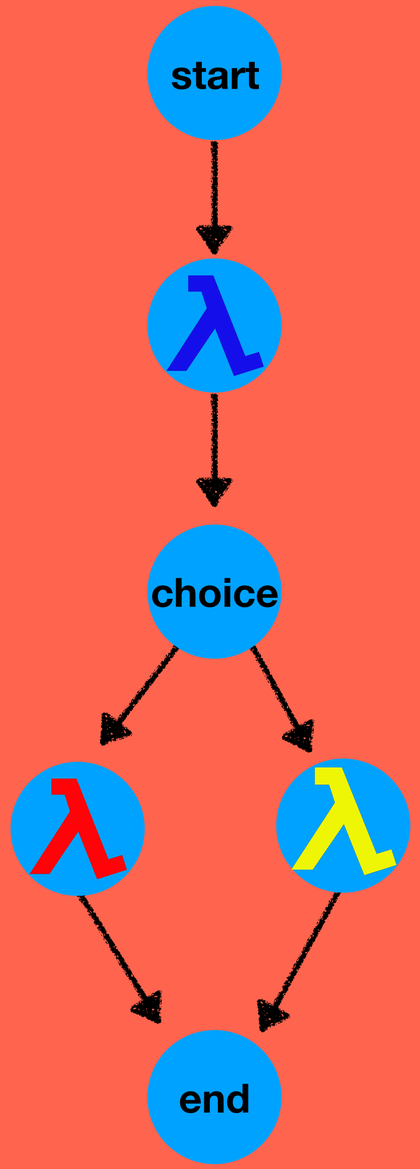
Time

Invoke
request body
payload size

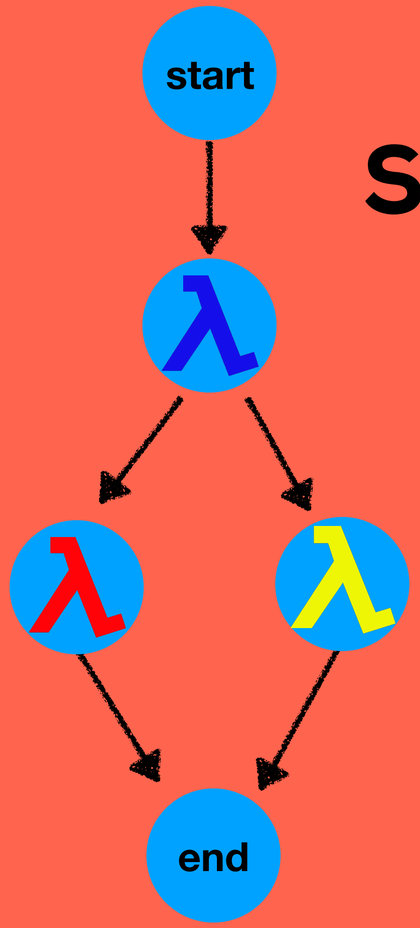
@BigDana

λ λλλλλλ
λλλλλλλ
λλλ λλλλλ
λλλ
λ

Step Functions

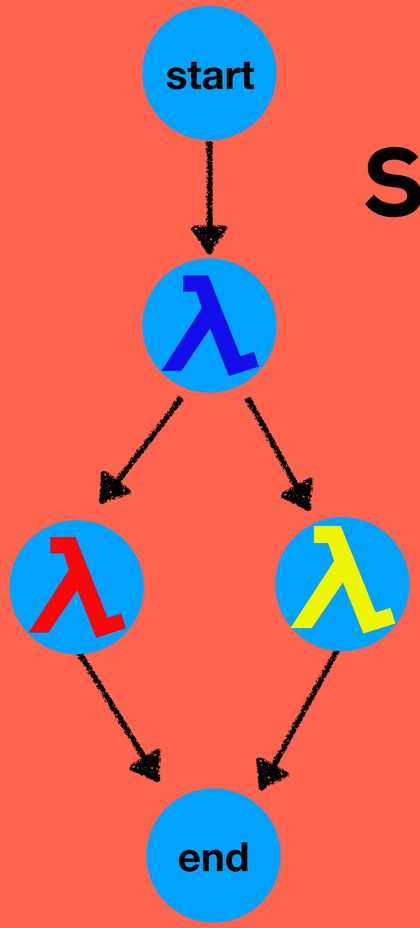


Choice State

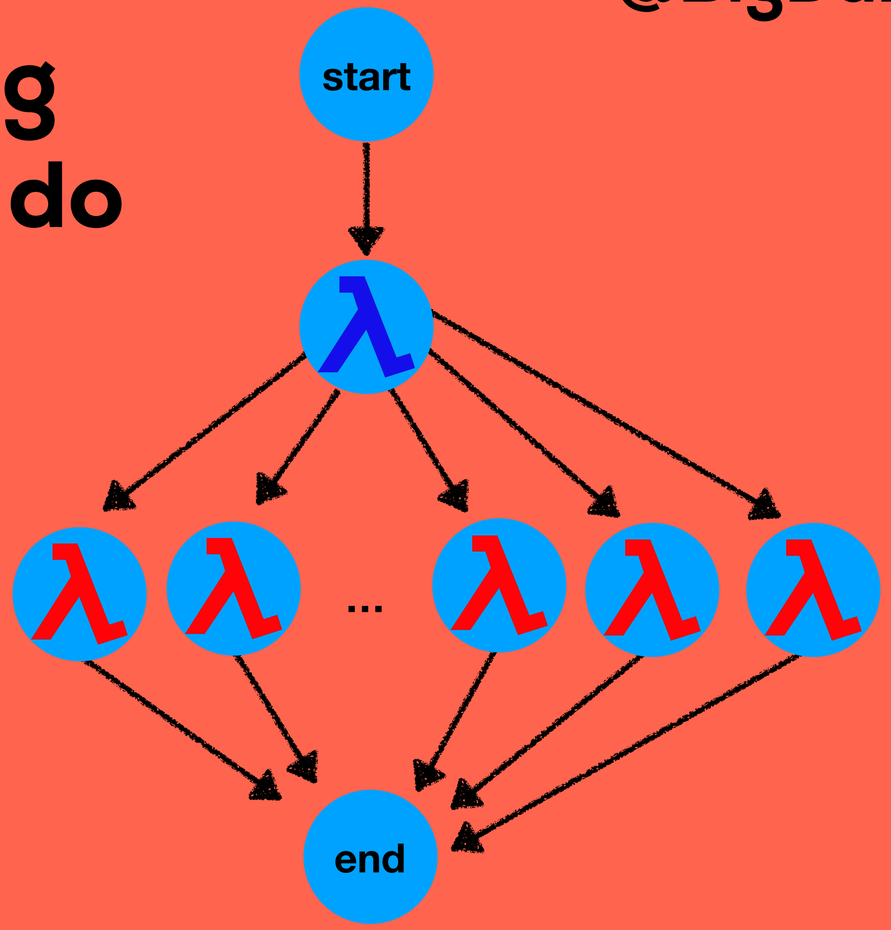


Parallel Processing Step Functions can do

Parallel Processing Step Functions can do



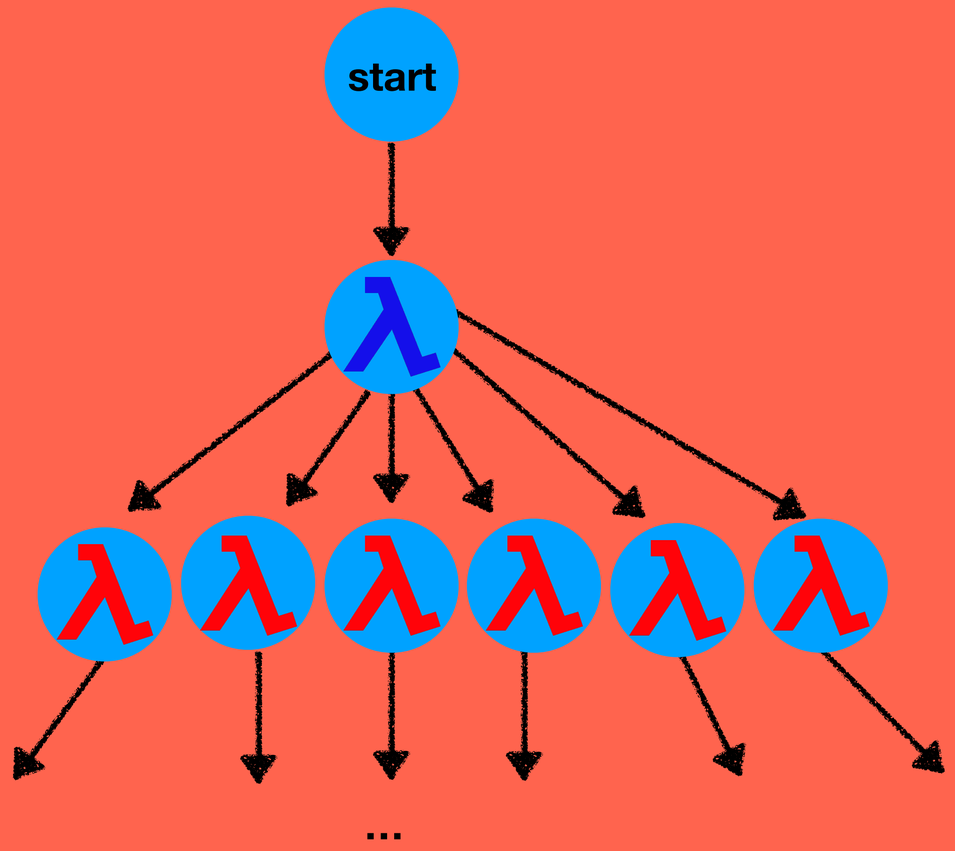
vs



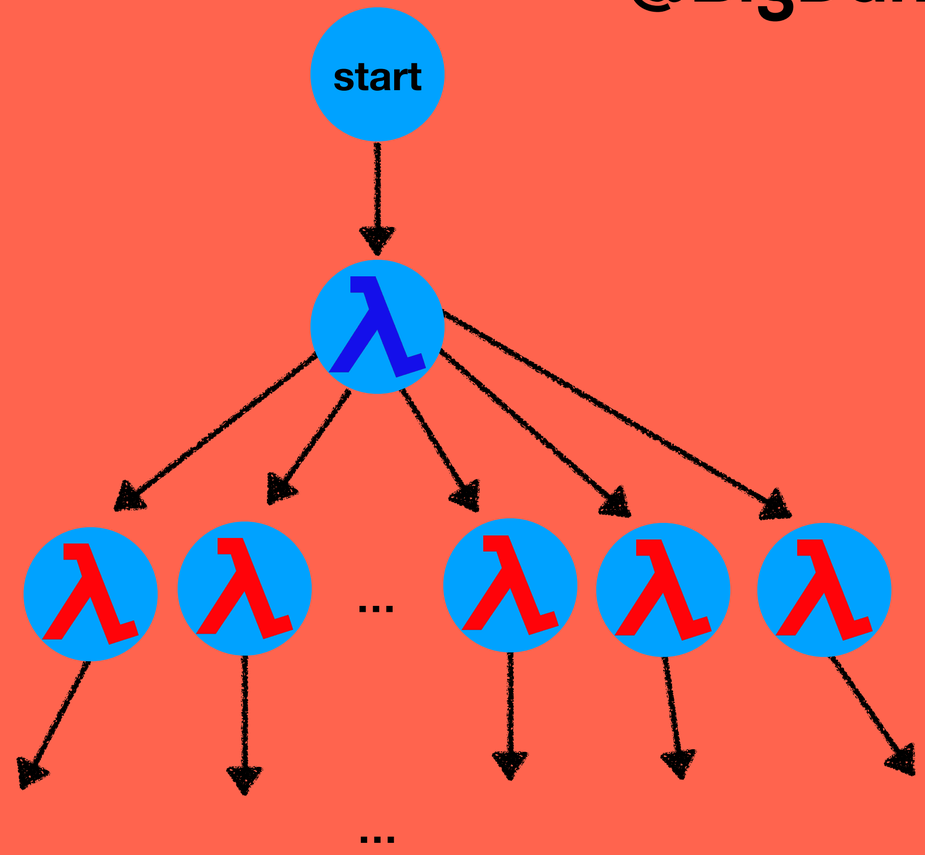
Dynamic Fan Out Parallel Processing

@BigDana

@BigDana

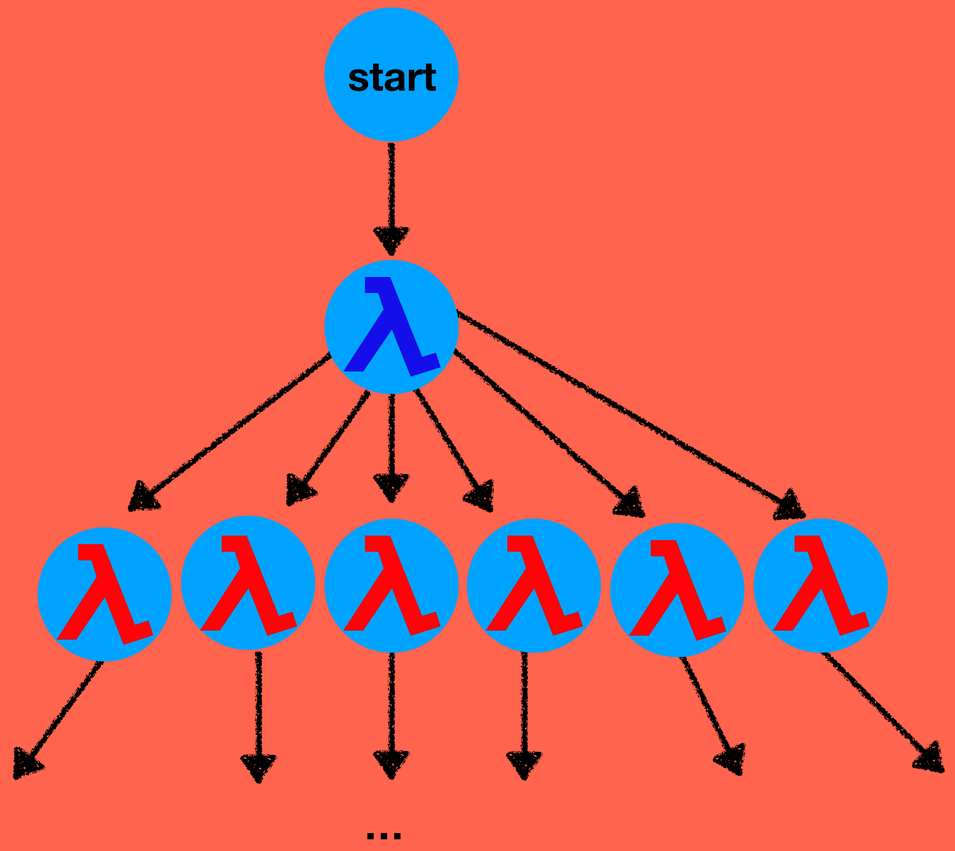


vs

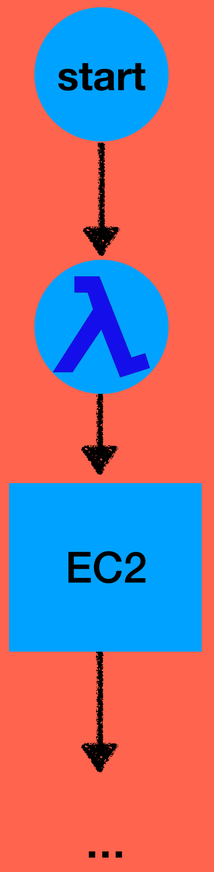


Defined Fan Out
Parallel Processing

Dynamic Fan Out
Parallel Processing



or



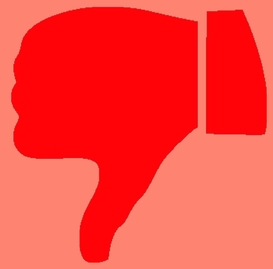
Defined Fan Out
Parallel Processing

send to an EC2 instance

Step Functions

easy to implement
workflow management for
lambdas and ec2 instances

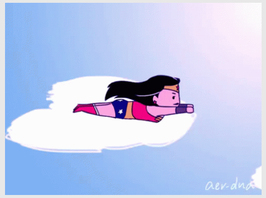
built-in back off policy
for retrying lambdas



doesn't yet natively
support a dynamic
one-to-many
fan out architecture
of lambdas


WONDER WOMAN'S EYELID MUSCLES LOOSEN THE TAPE-BUT ALSO HER EYELASHES.!
UN-UNH.! MY FEMININE VANITY WON'T LET ME PULL OUT MY EYELASHES.! I'LL HAVE TO ESCAPE BLINDFOLDED.!

Building a Data Pipeline



Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline

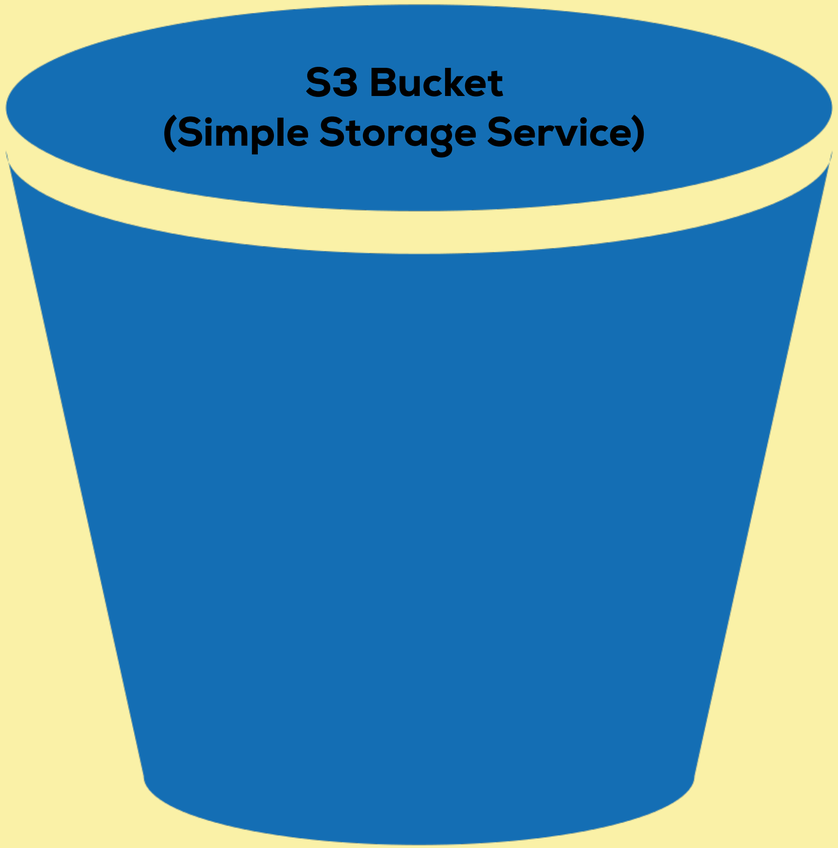
Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

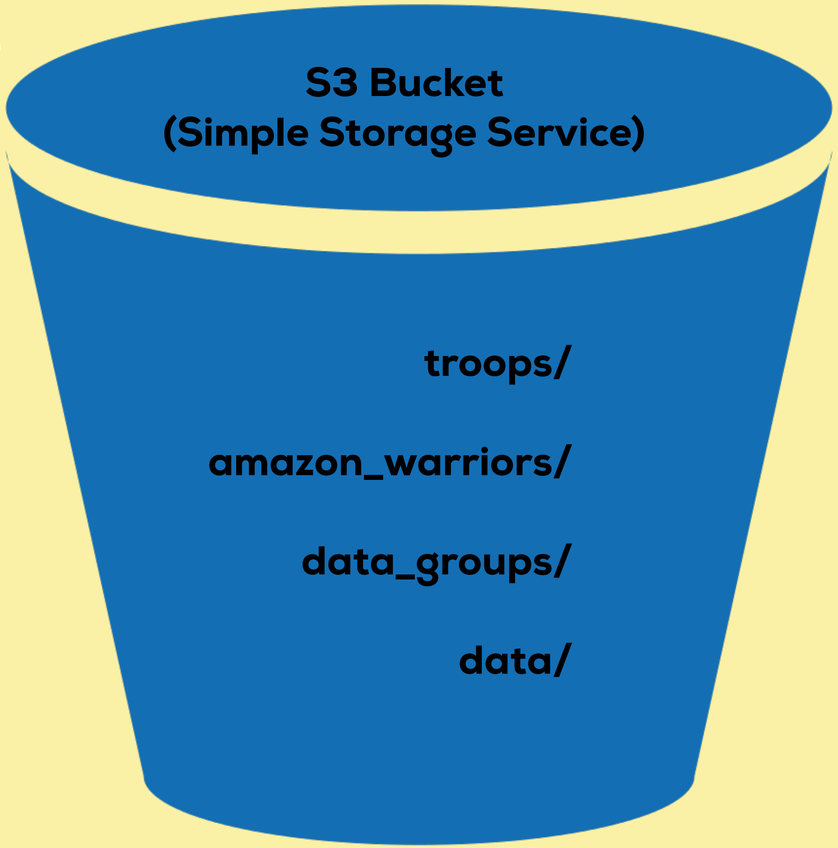
Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

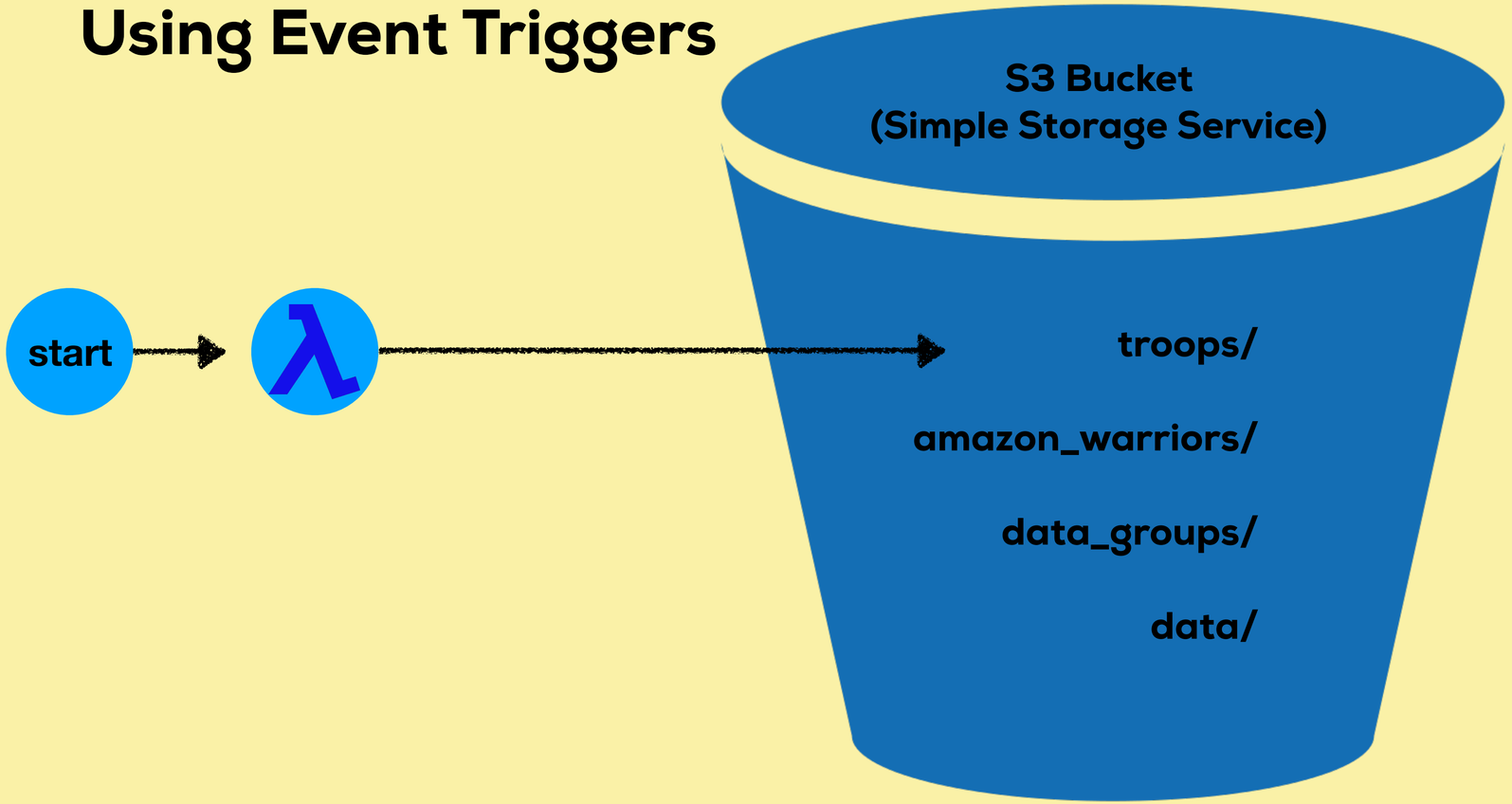
Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas



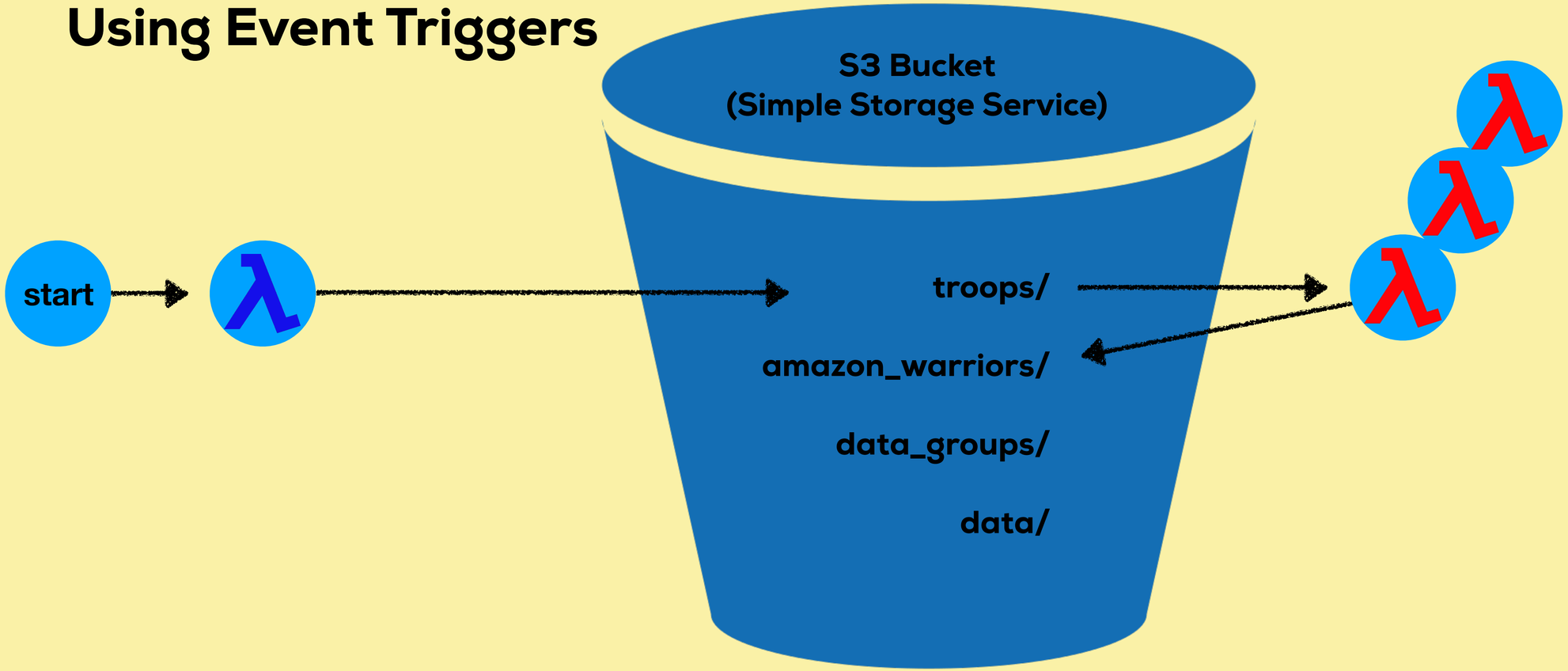
Using Event Triggers



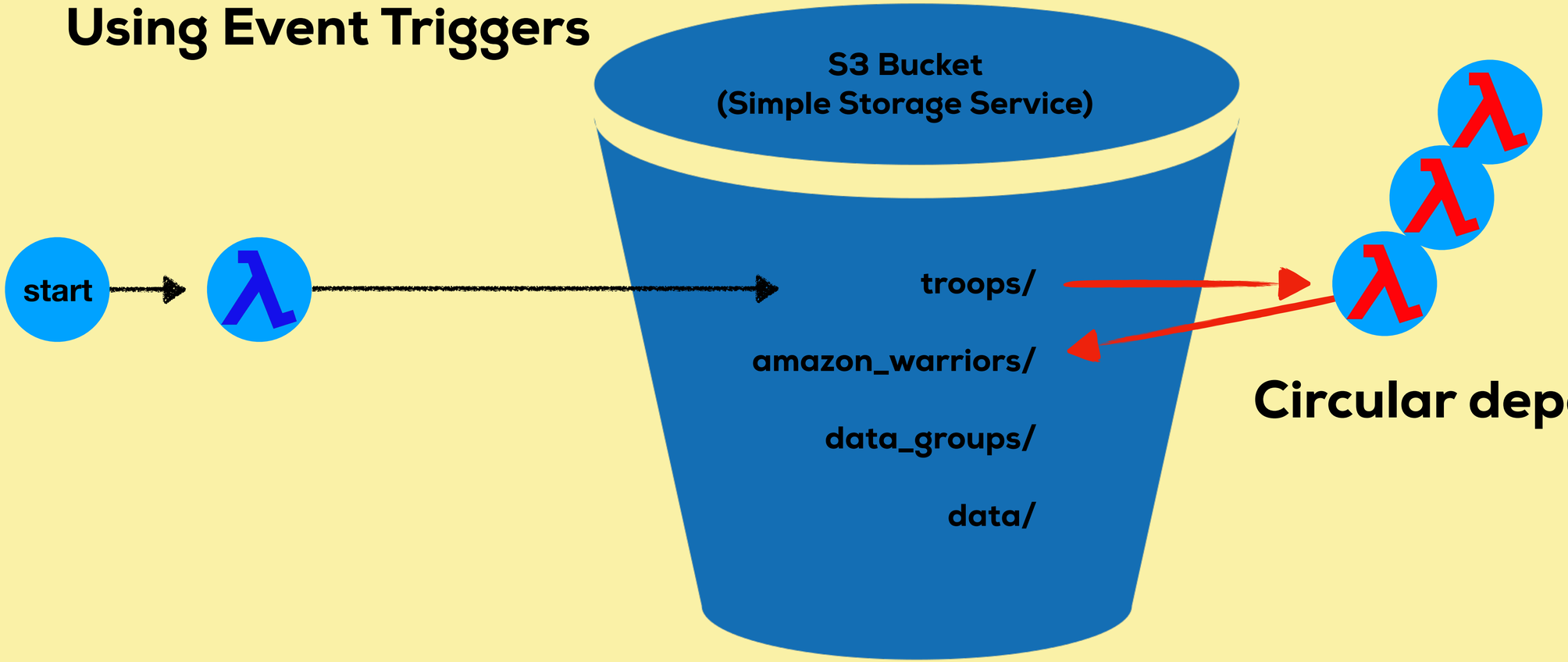
Using Event Triggers



Using Event Triggers

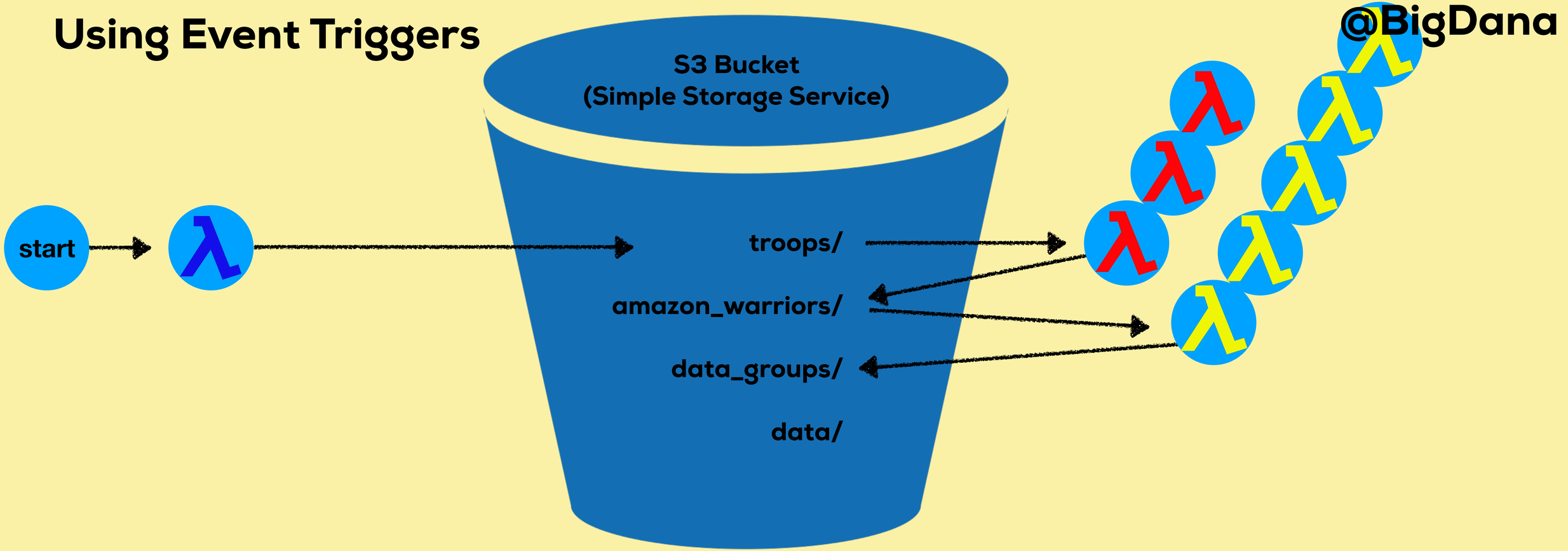


Using Event Triggers

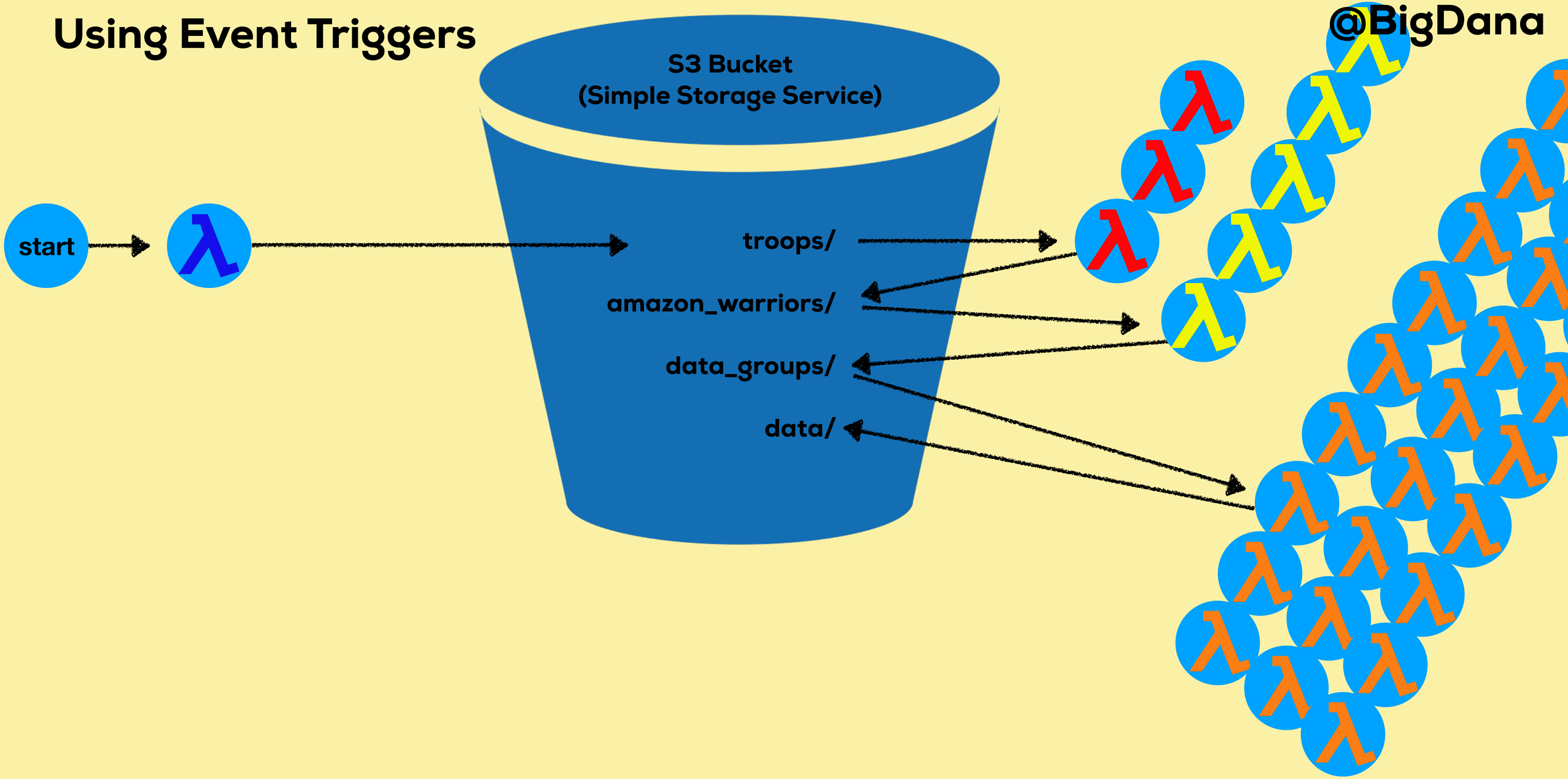


Circular dependency

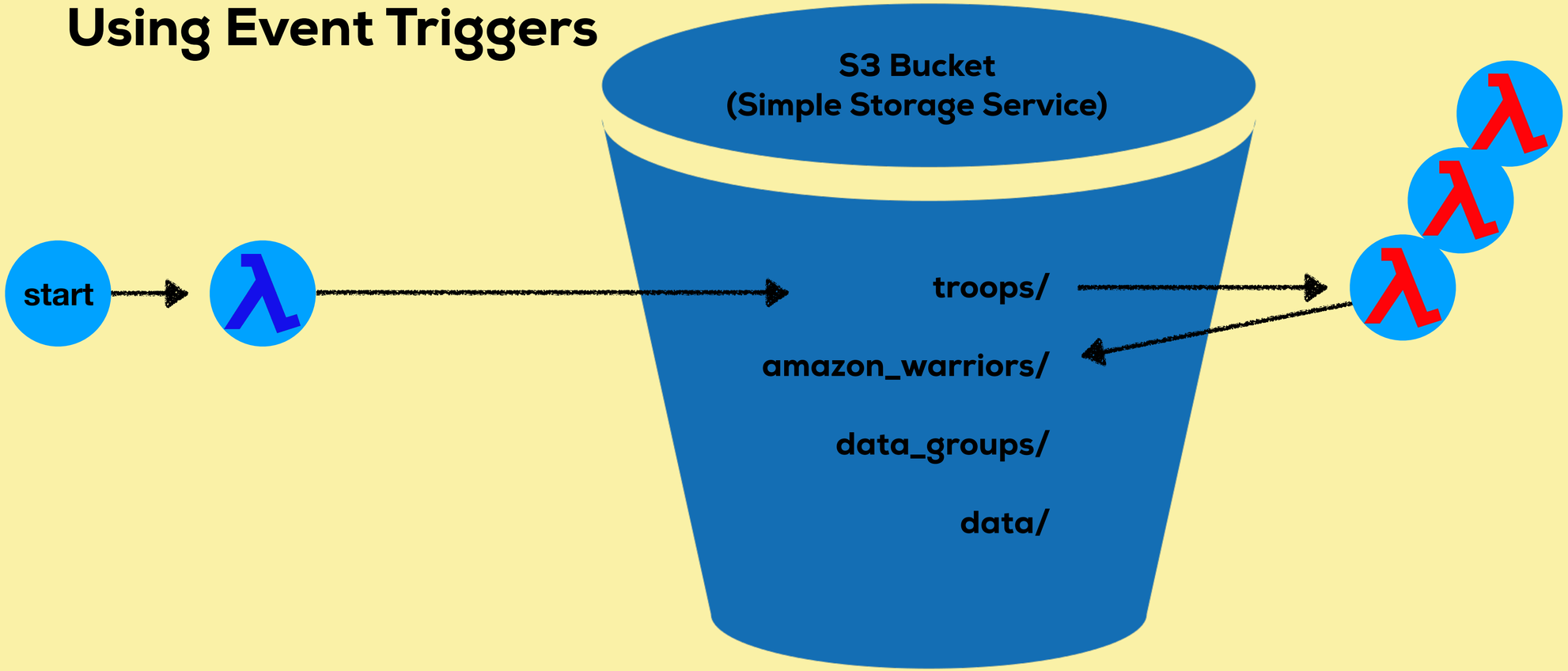
Using Event Triggers



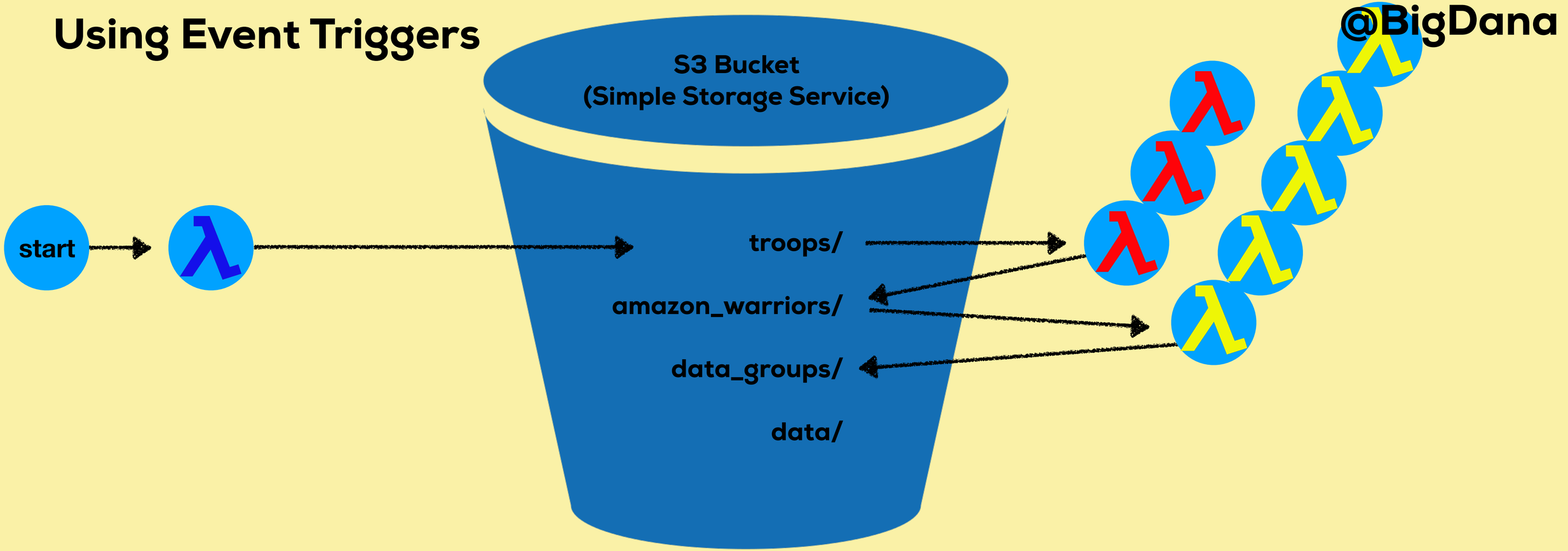
Using Event Triggers



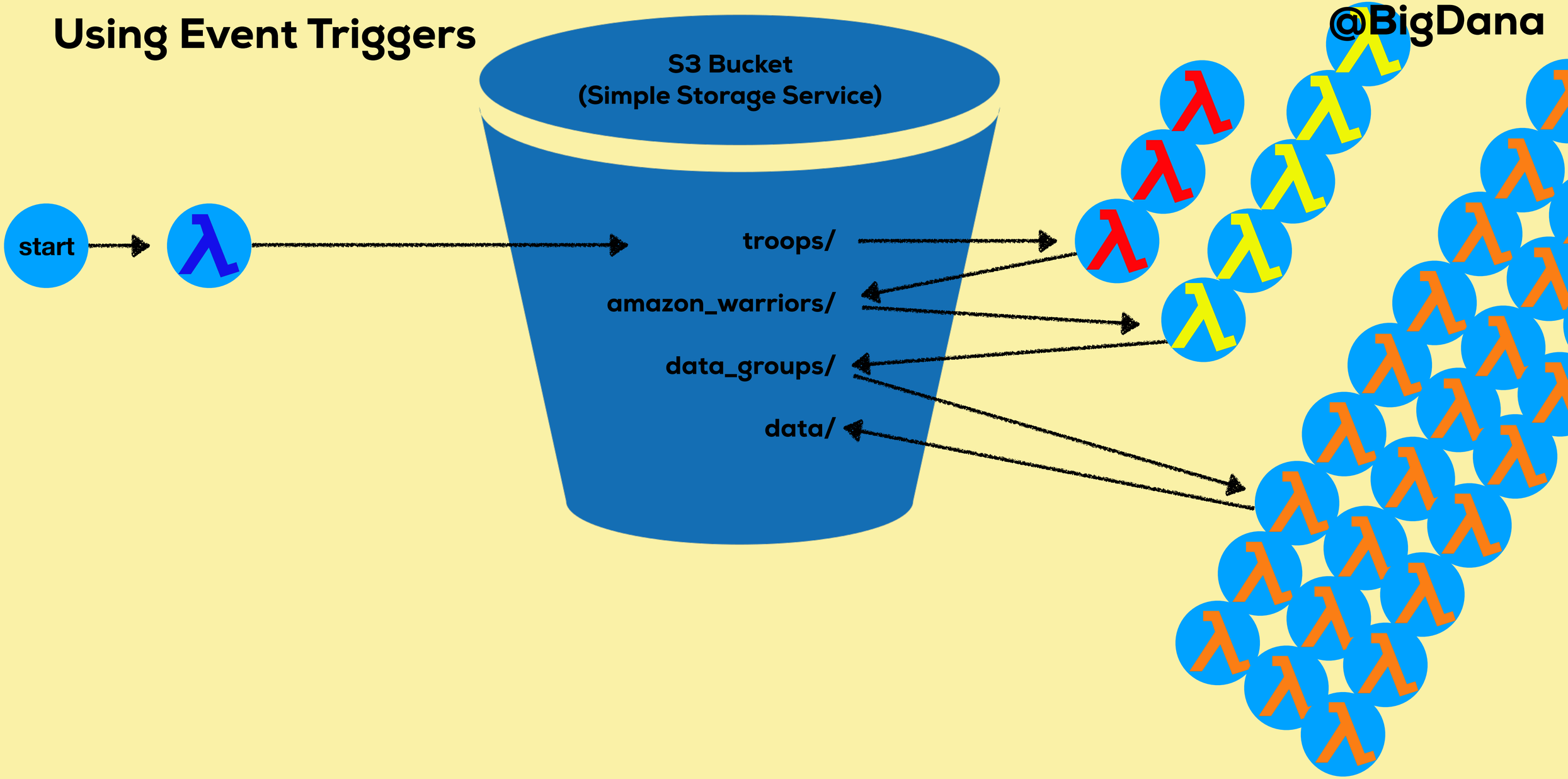
Using Event Triggers



Using Event Triggers



Using Event Triggers




Too Fast

Using Event Triggers with S3

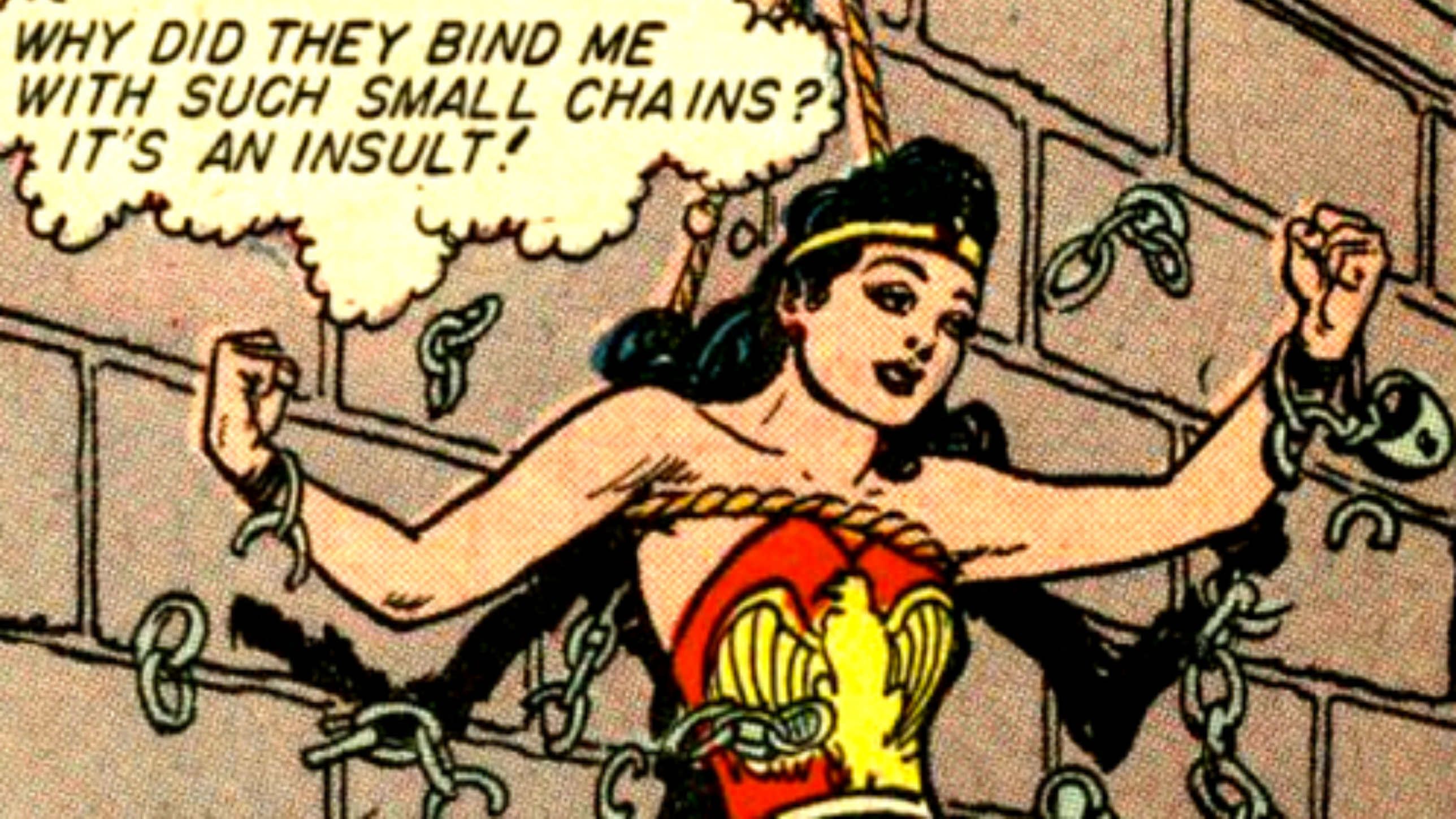
s3 is highly scalable and durable

Nested event triggers creates
super fast multiprocessing

You can’t control the rate
the lambdas get spawned

It will only retry a lambda 2 times
(can extend this manually)

You can lose in-flight tasks
if there is a system outage


WHY DID THEY BIND ME WITH SUCH SMALL CHAINS? IT'S AN INSULT!

@BigDana

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

Attempt 1: Lambdas Orchestrated by Step Functions

Attempt 2: Using Event Triggers with S3

Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

Building a Data Pipeline

Attempt 0: Python Multiprocessing on an EC2 Instance

Attempt 0.5: Python Multiprocessing on a Spot Instance

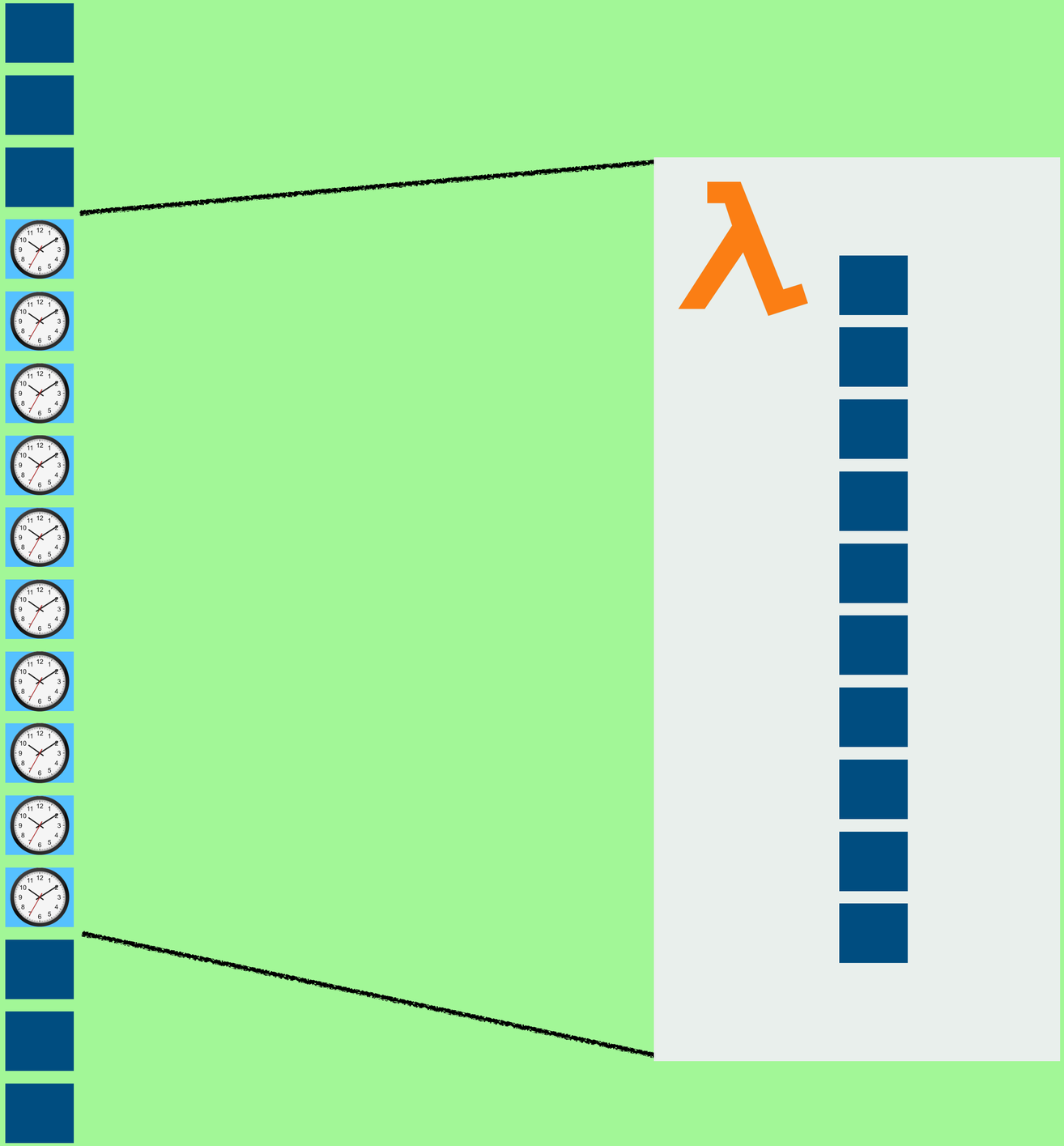
Attempt 1: Lambdas Orchestrated by Step Functions

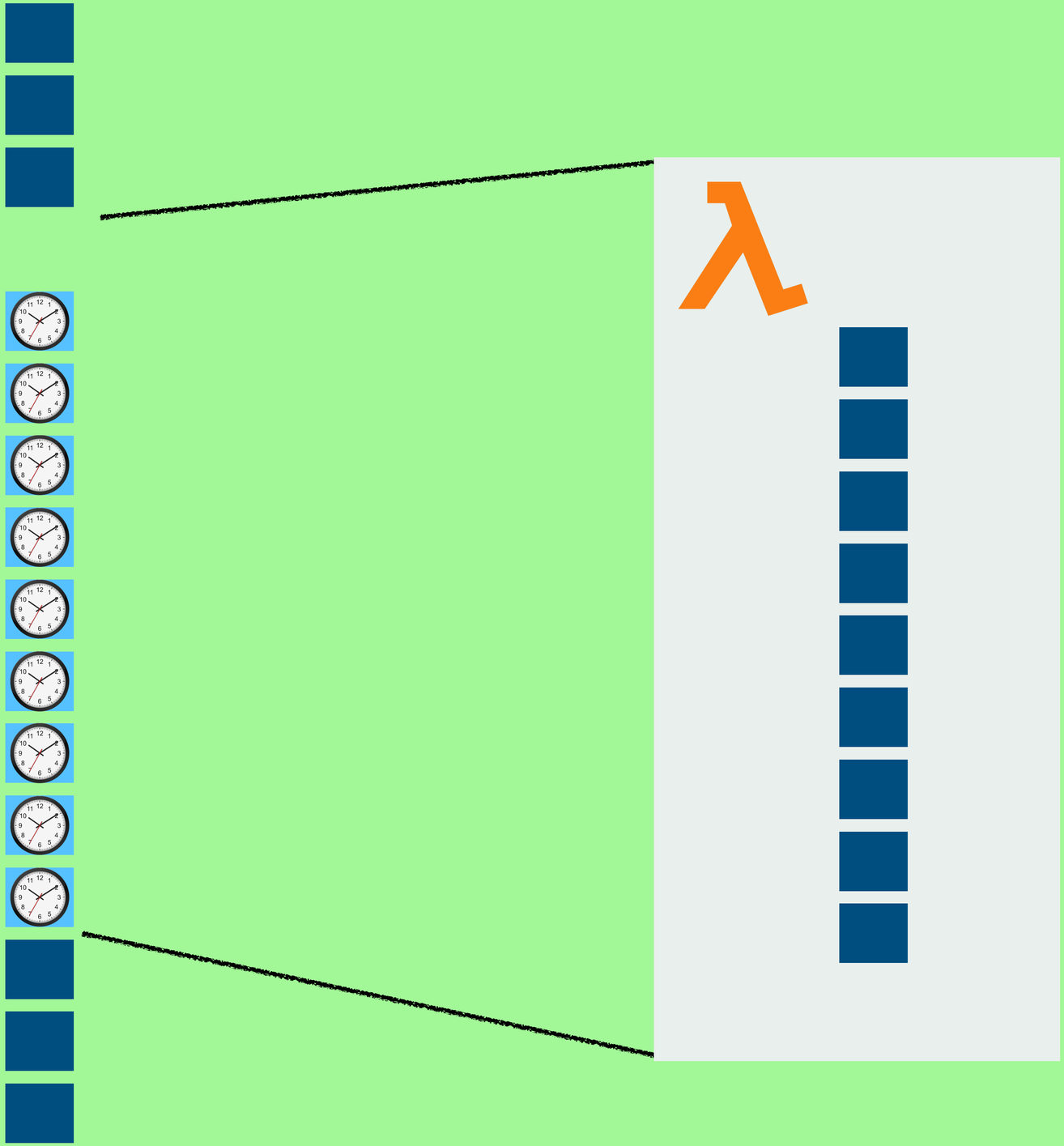
Attempt 2: Using Event Triggers with S3

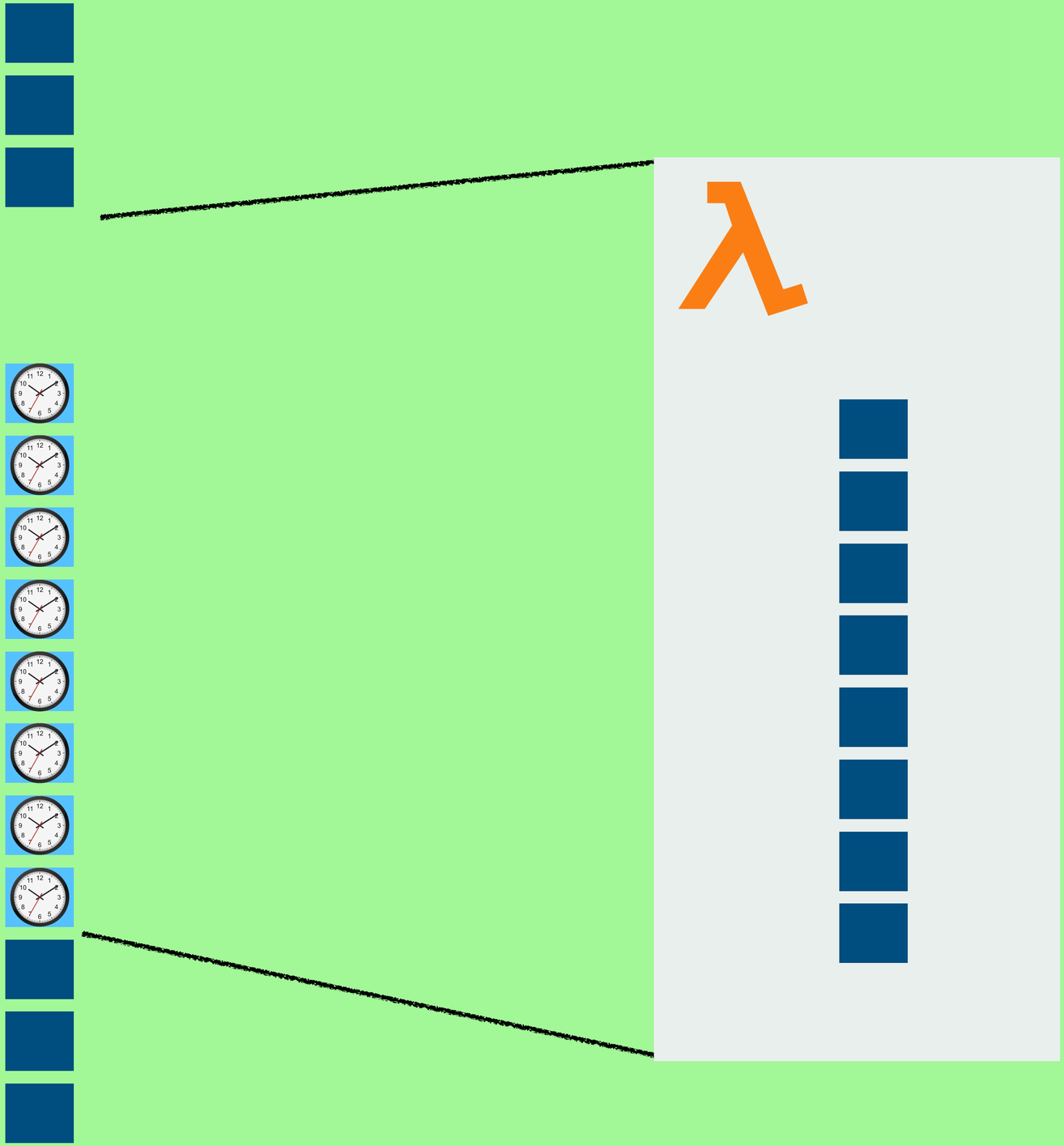
Attempt 3: Rate limiting with Queues and Cloudwatch triggered Lambdas

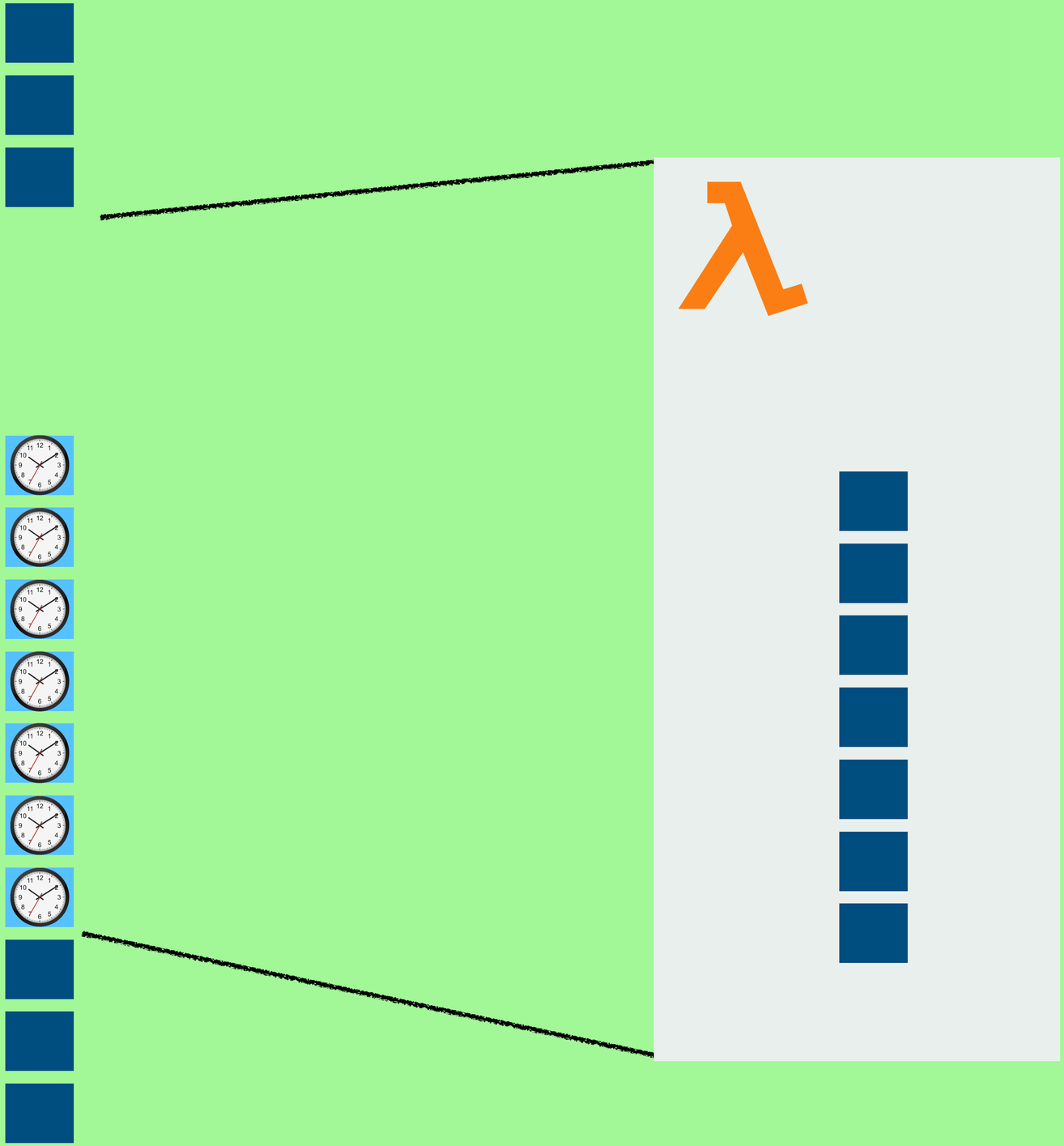
SQS Queues

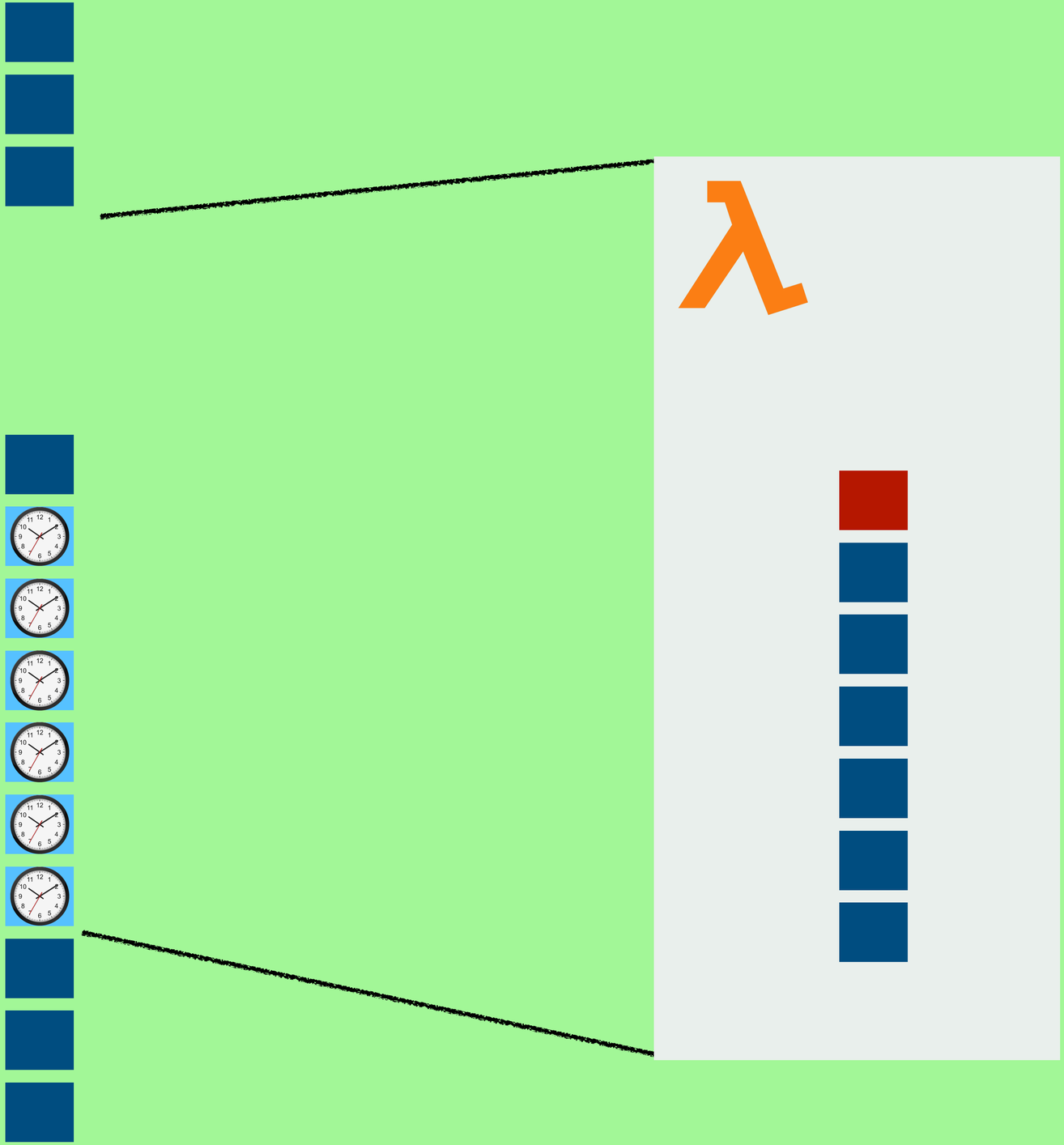
(Simple Queue Service)

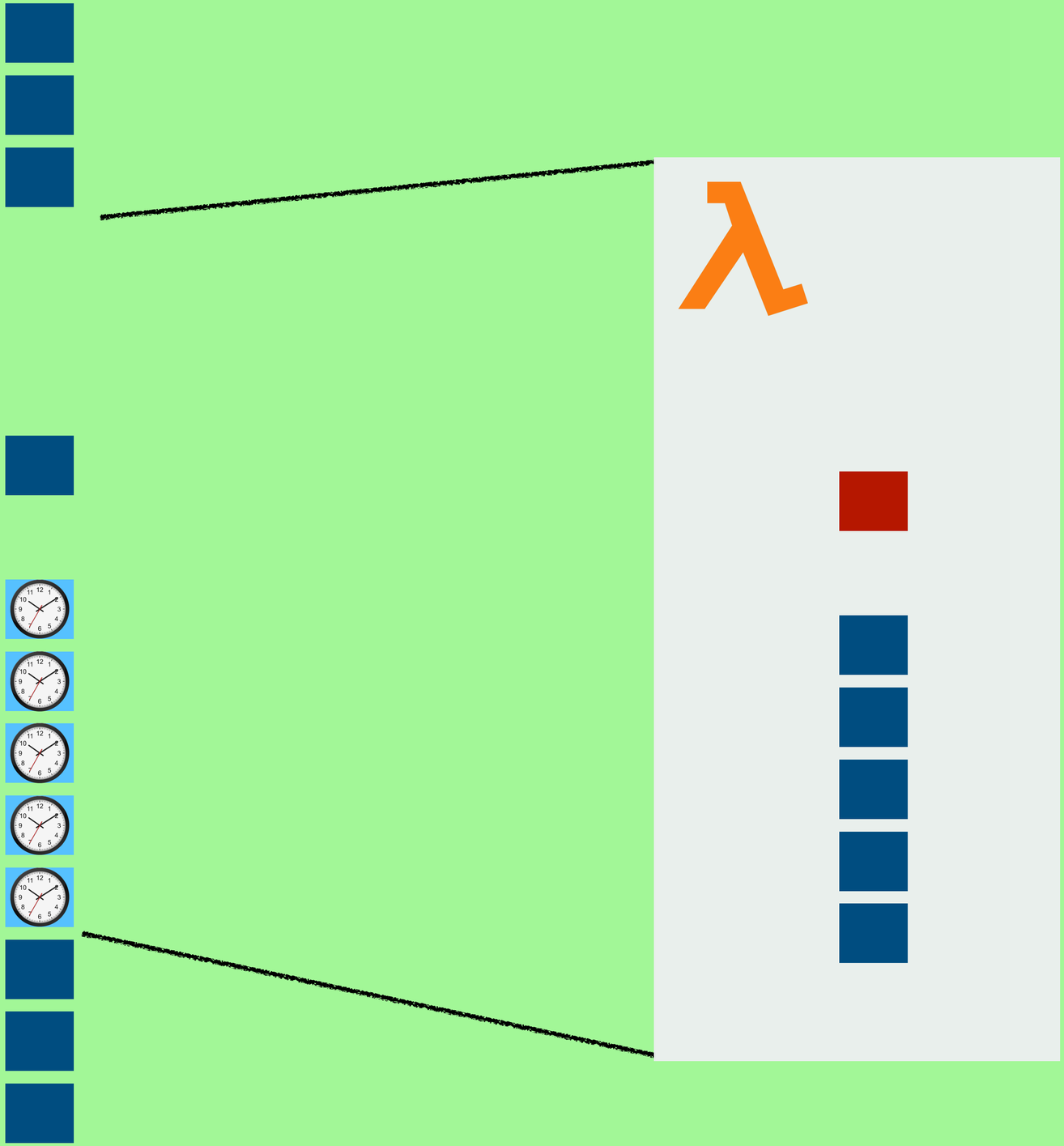


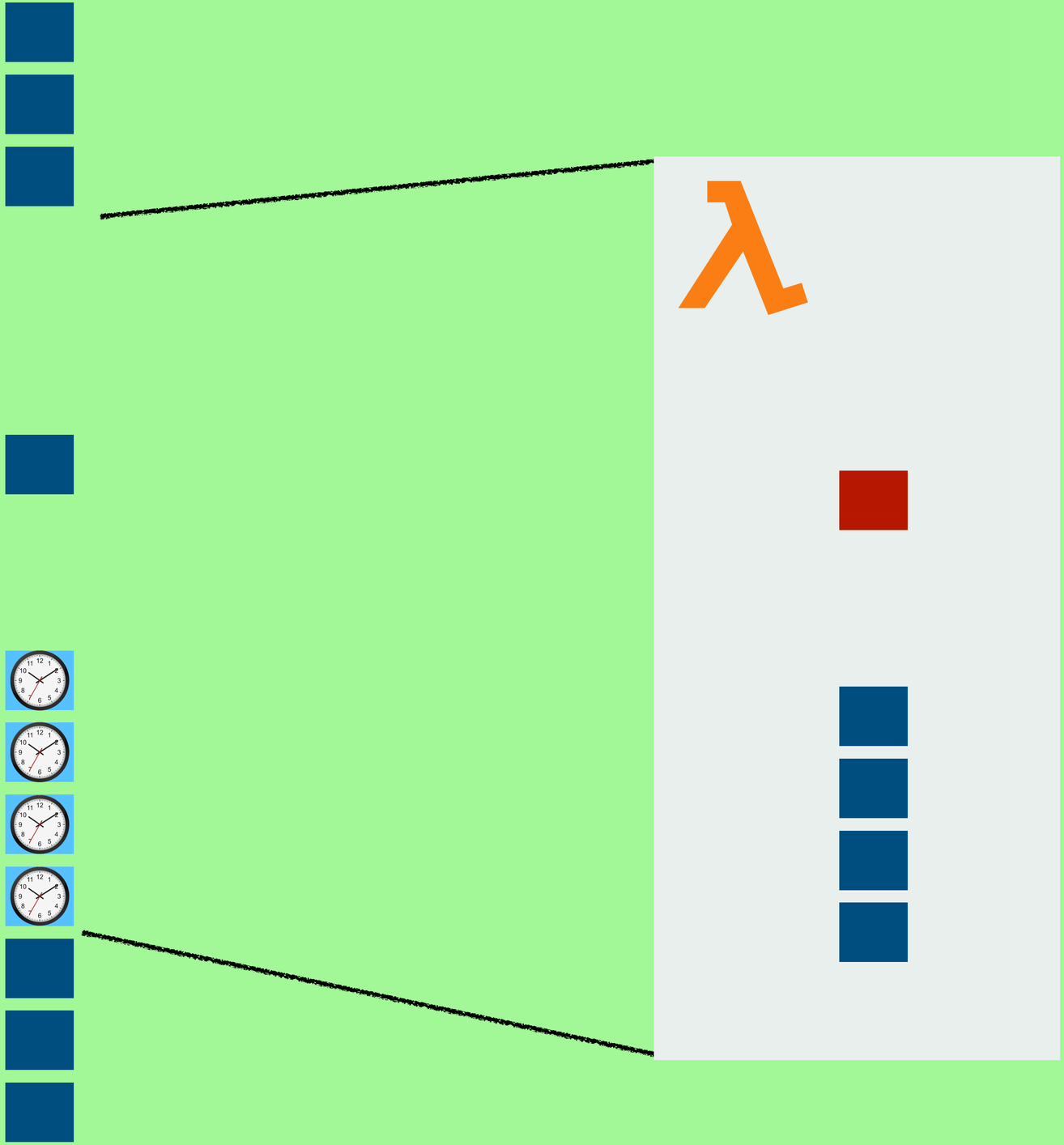


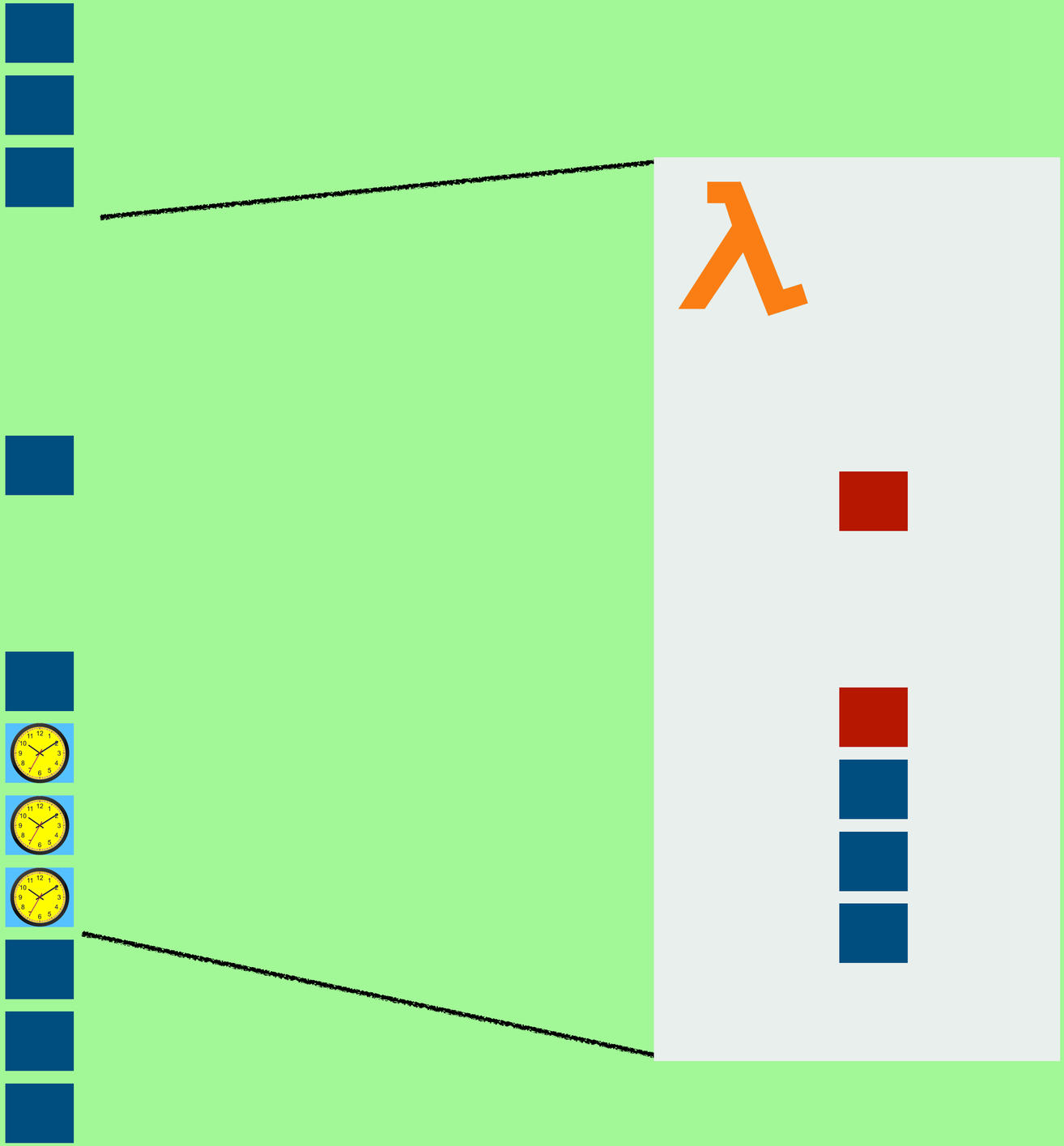


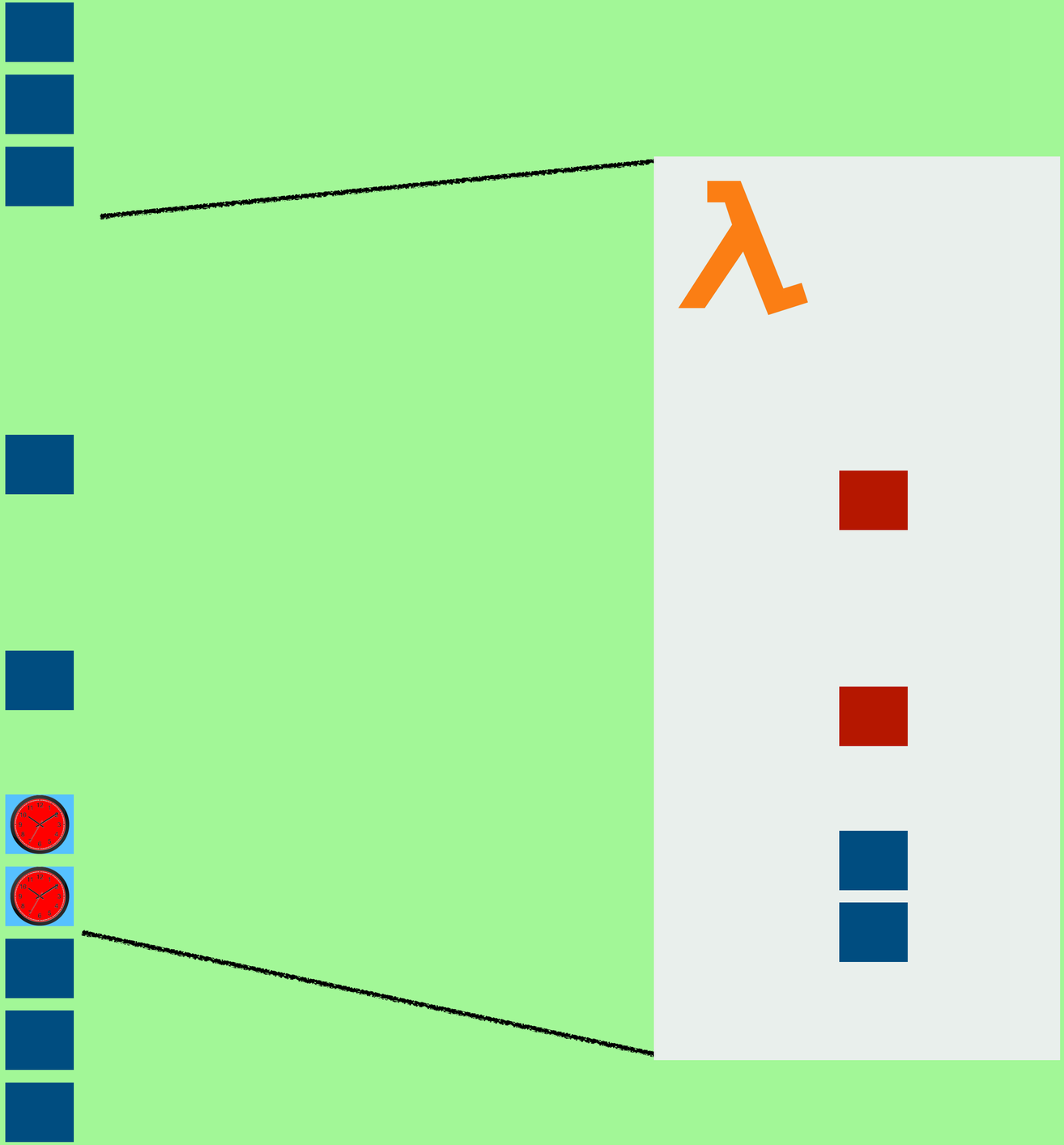


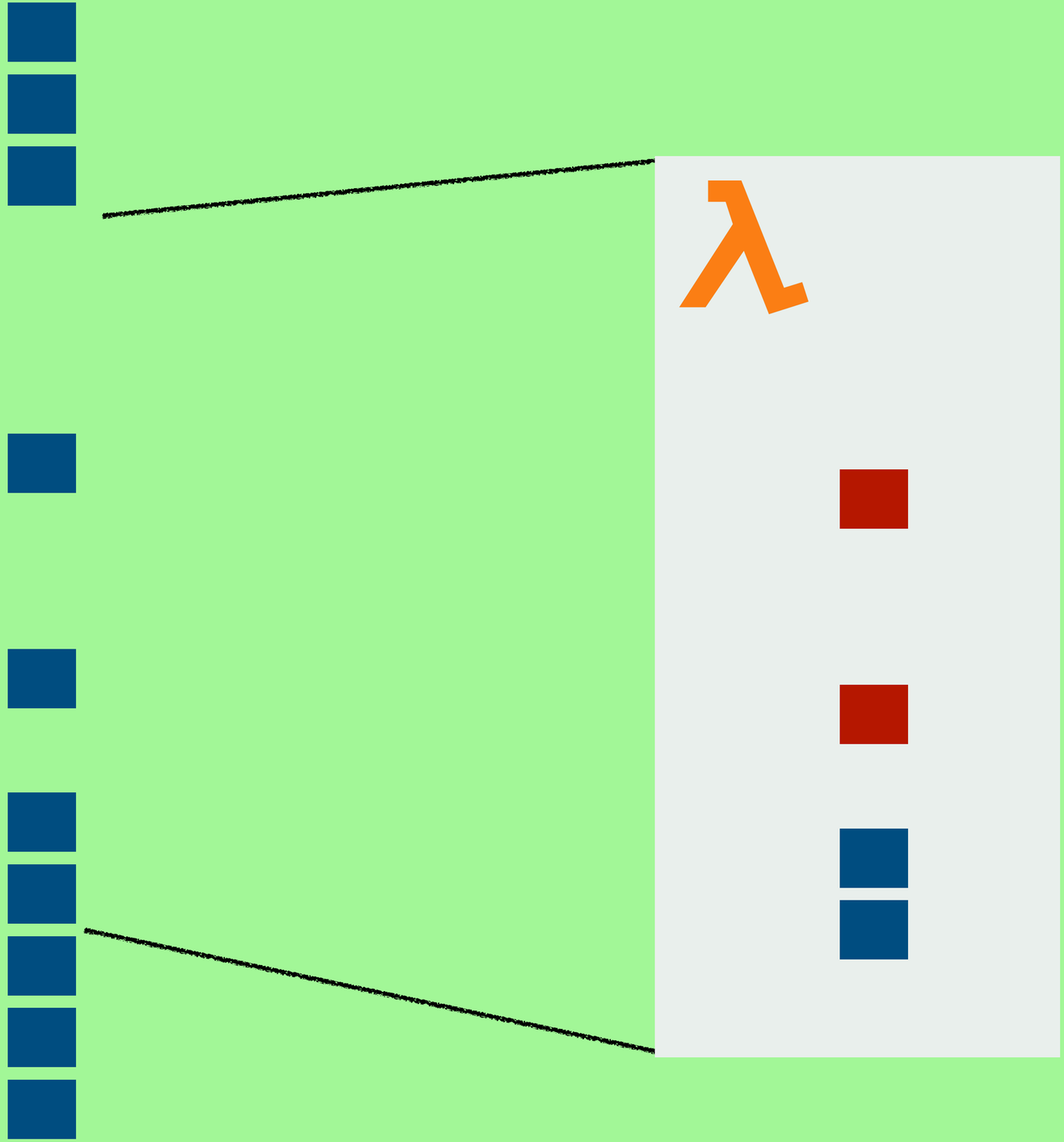


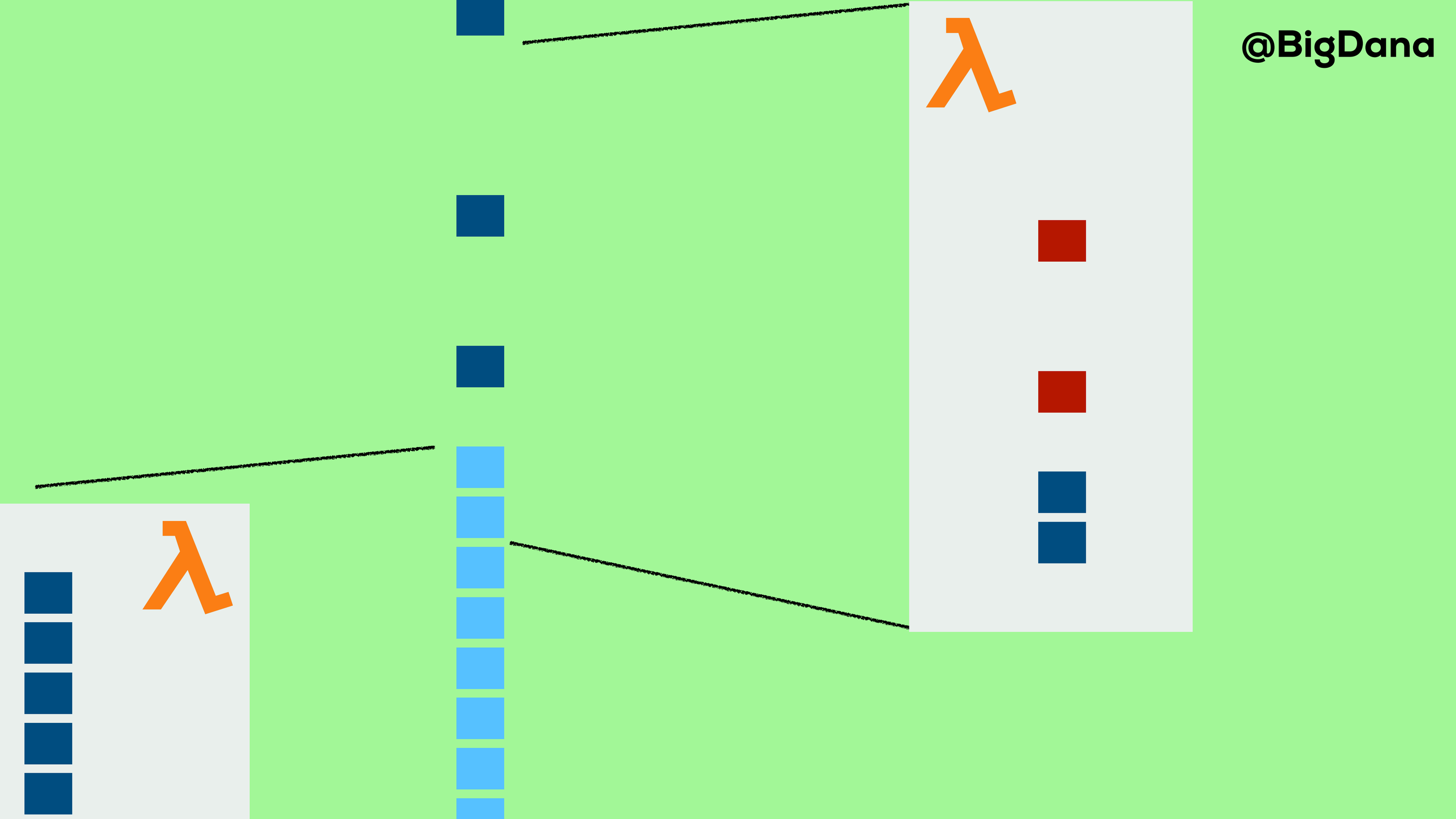


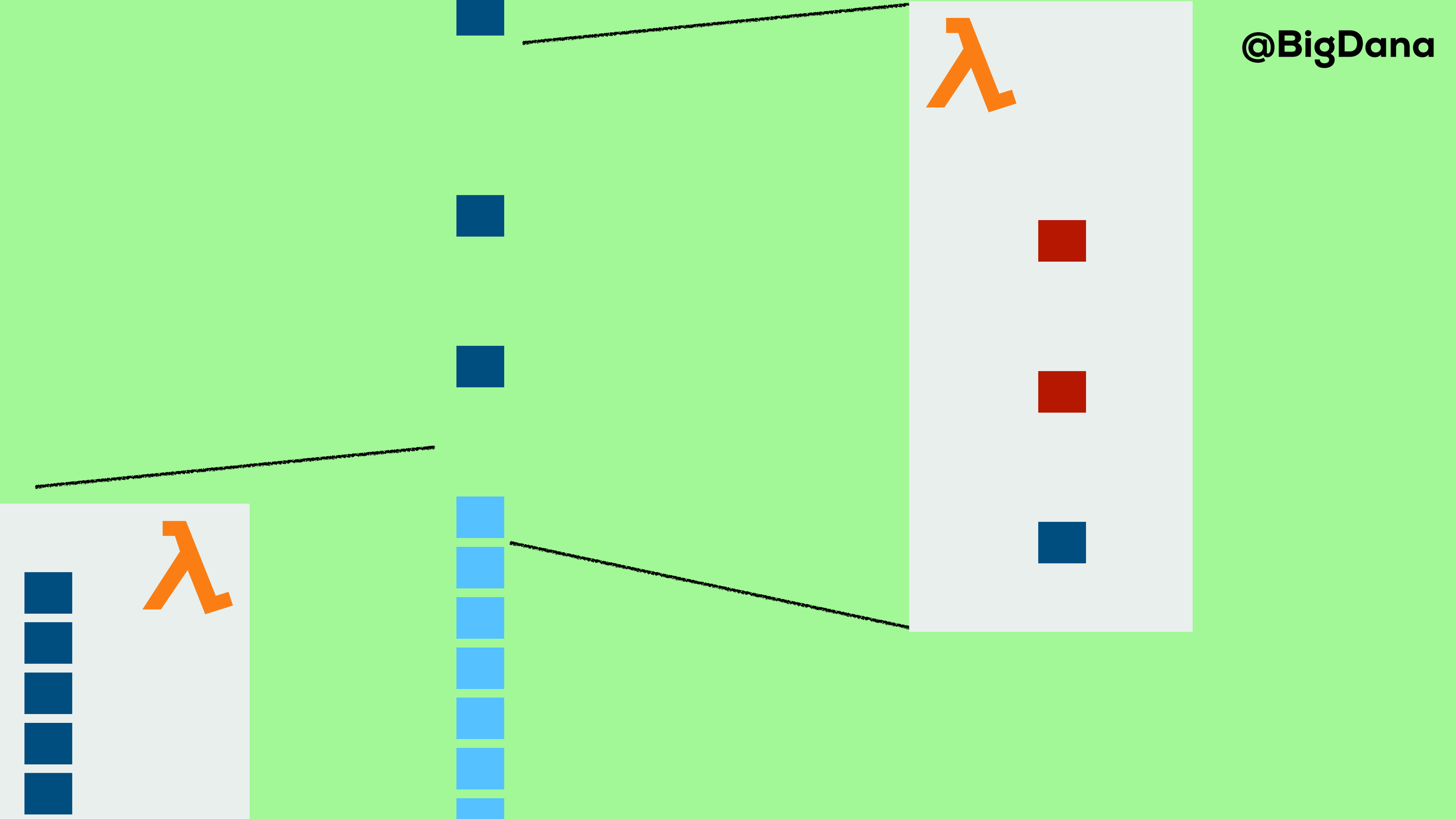


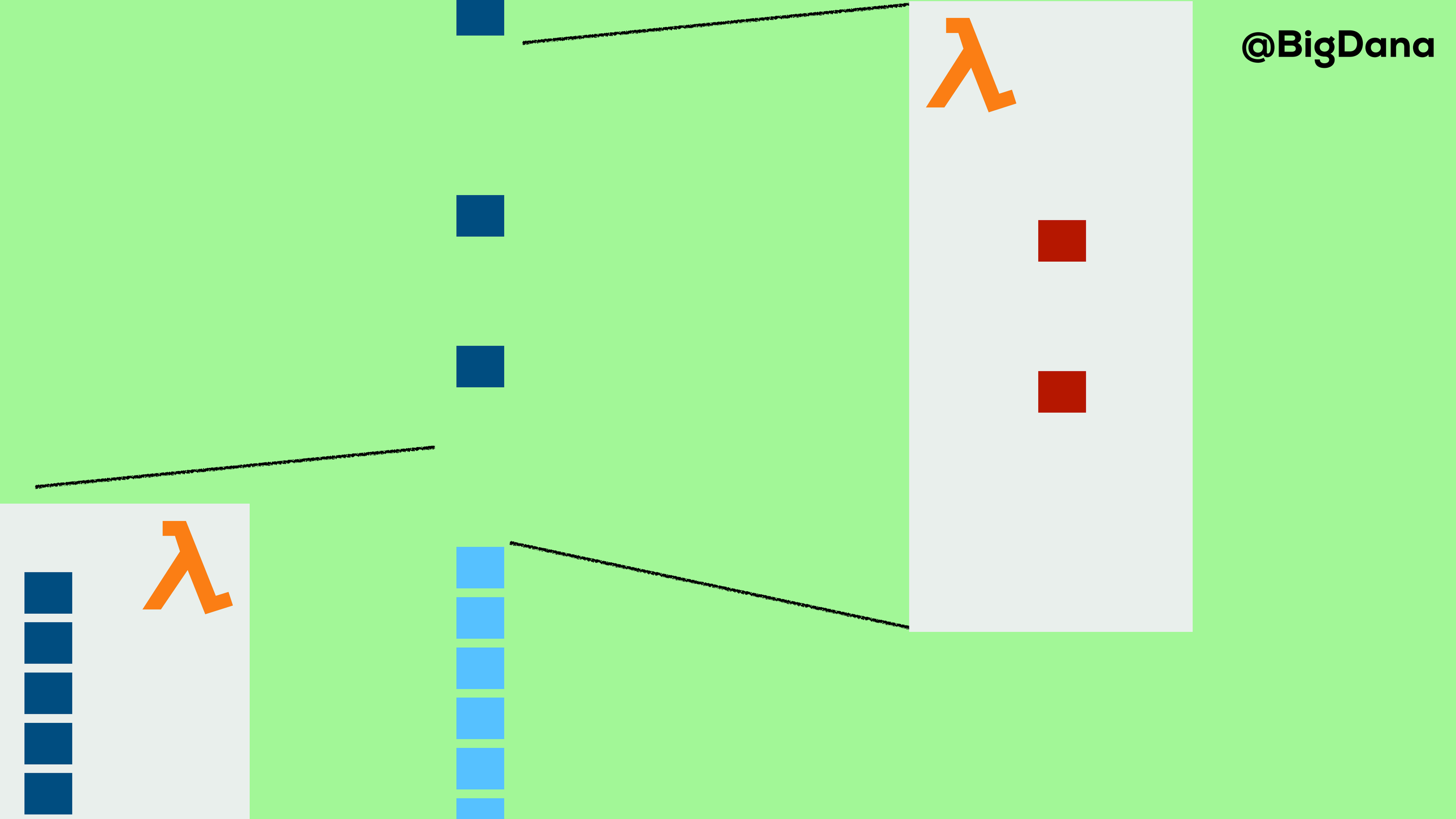


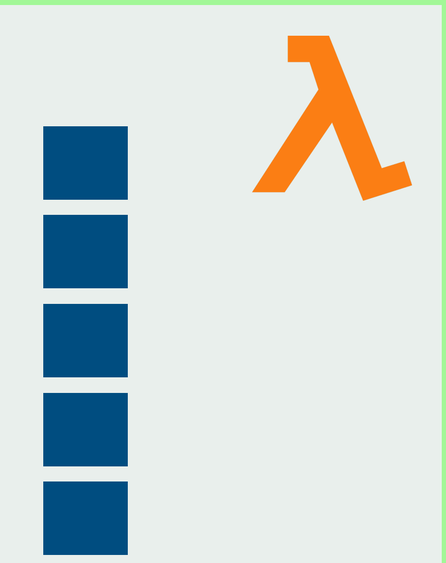





@BigDana


@BigDana

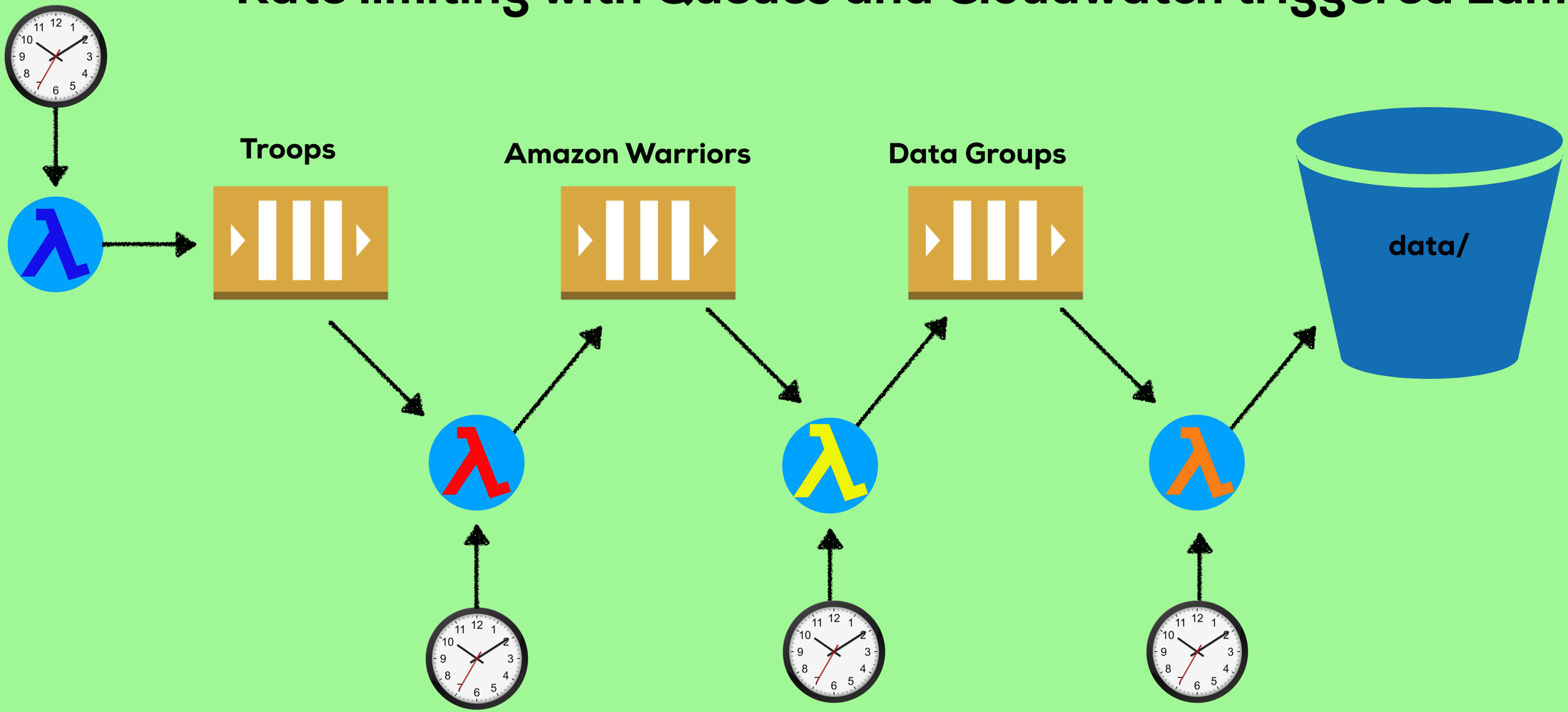

@BigDana



Cloudwatch Rules

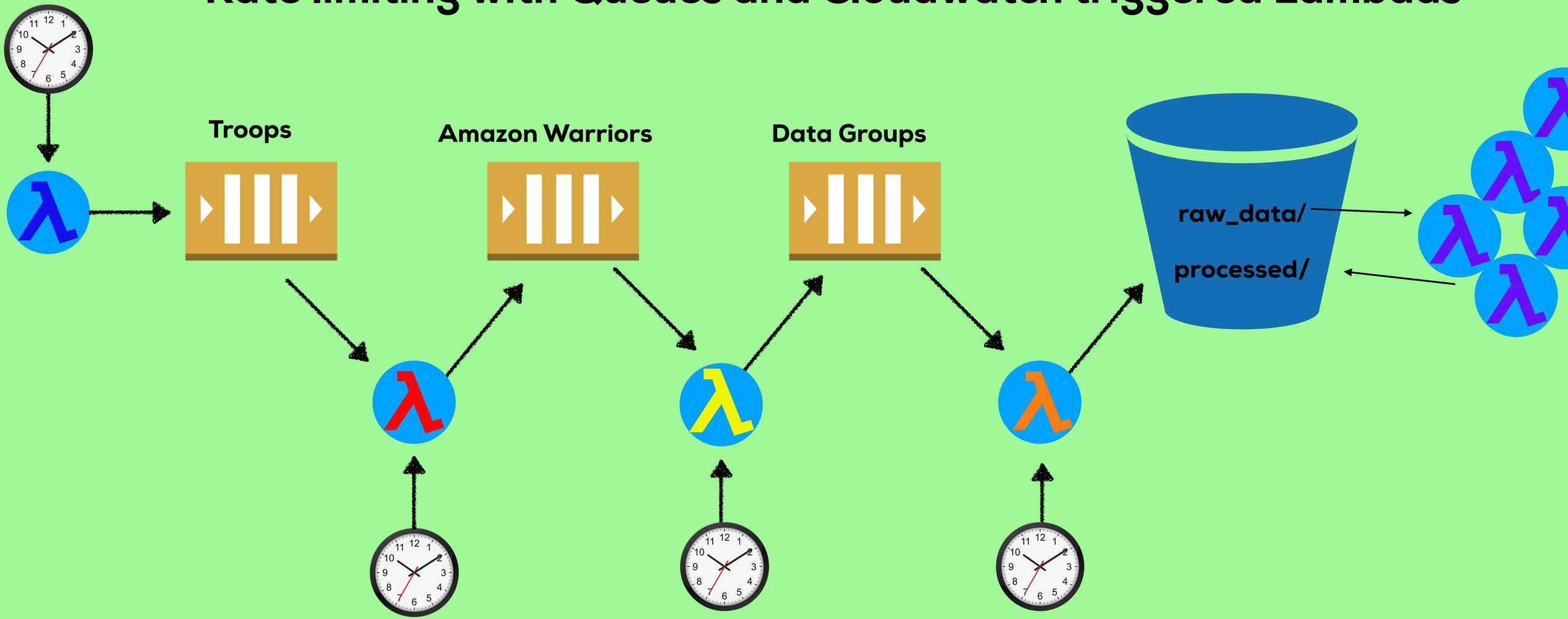
@BigDana

Rate limiting with Queues and Cloudwatch triggered Lambdas



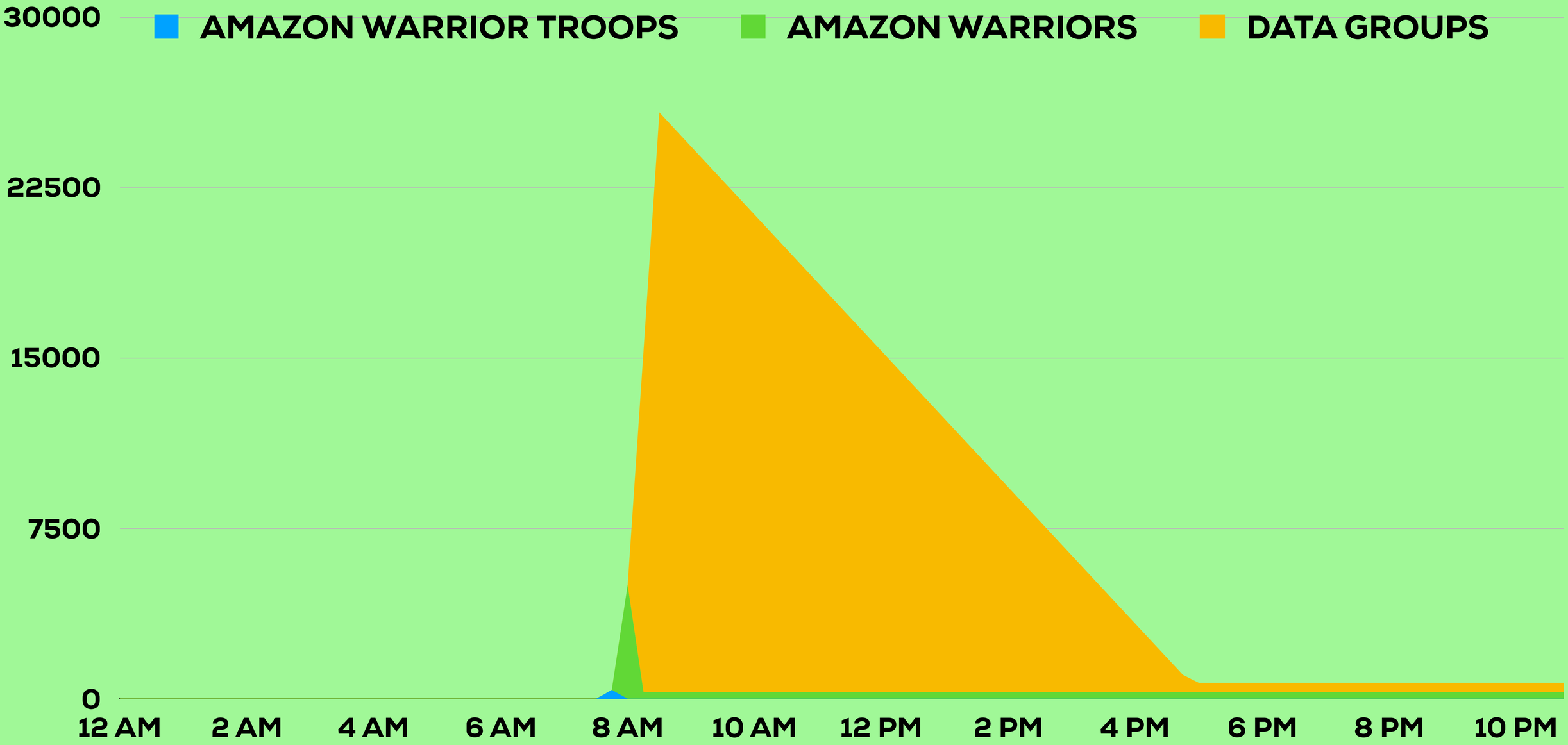
@BigDana

Rate limiting with Queues and Cloudwatch triggered Lambdas



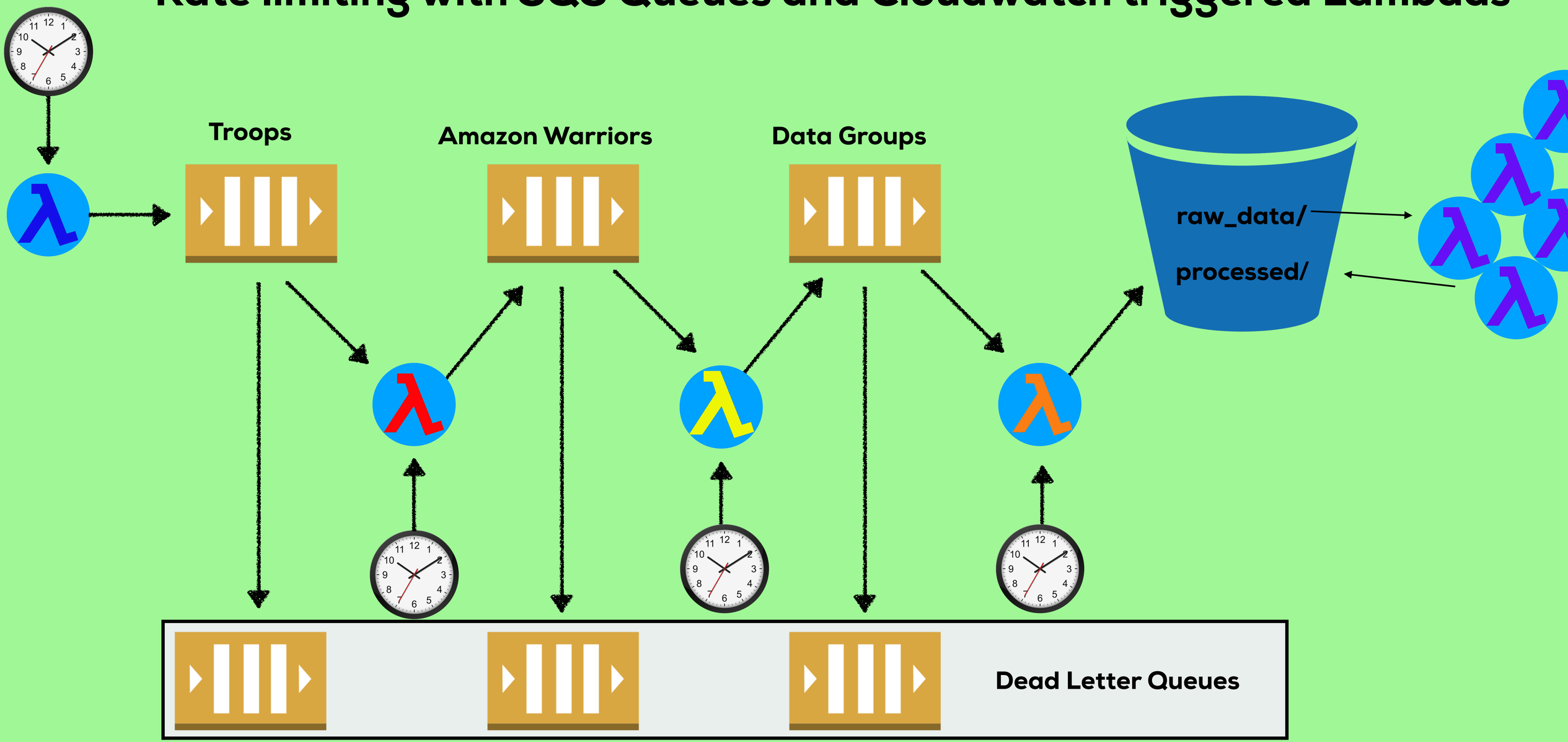
@BigDana

Queue Depth Over Time



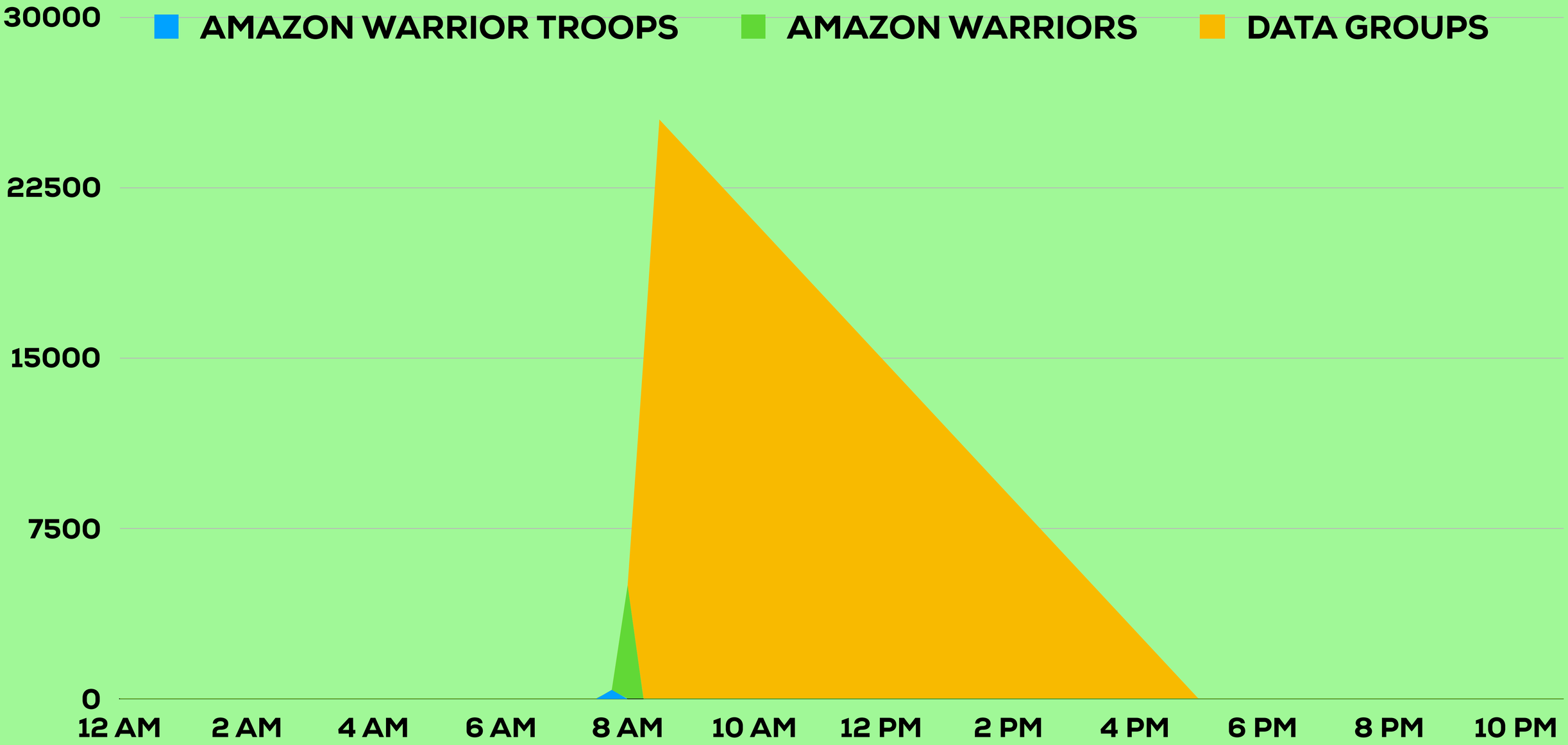
@BigDana

Rate limiting with SQS Queues and Cloudwatch triggered Lambdas



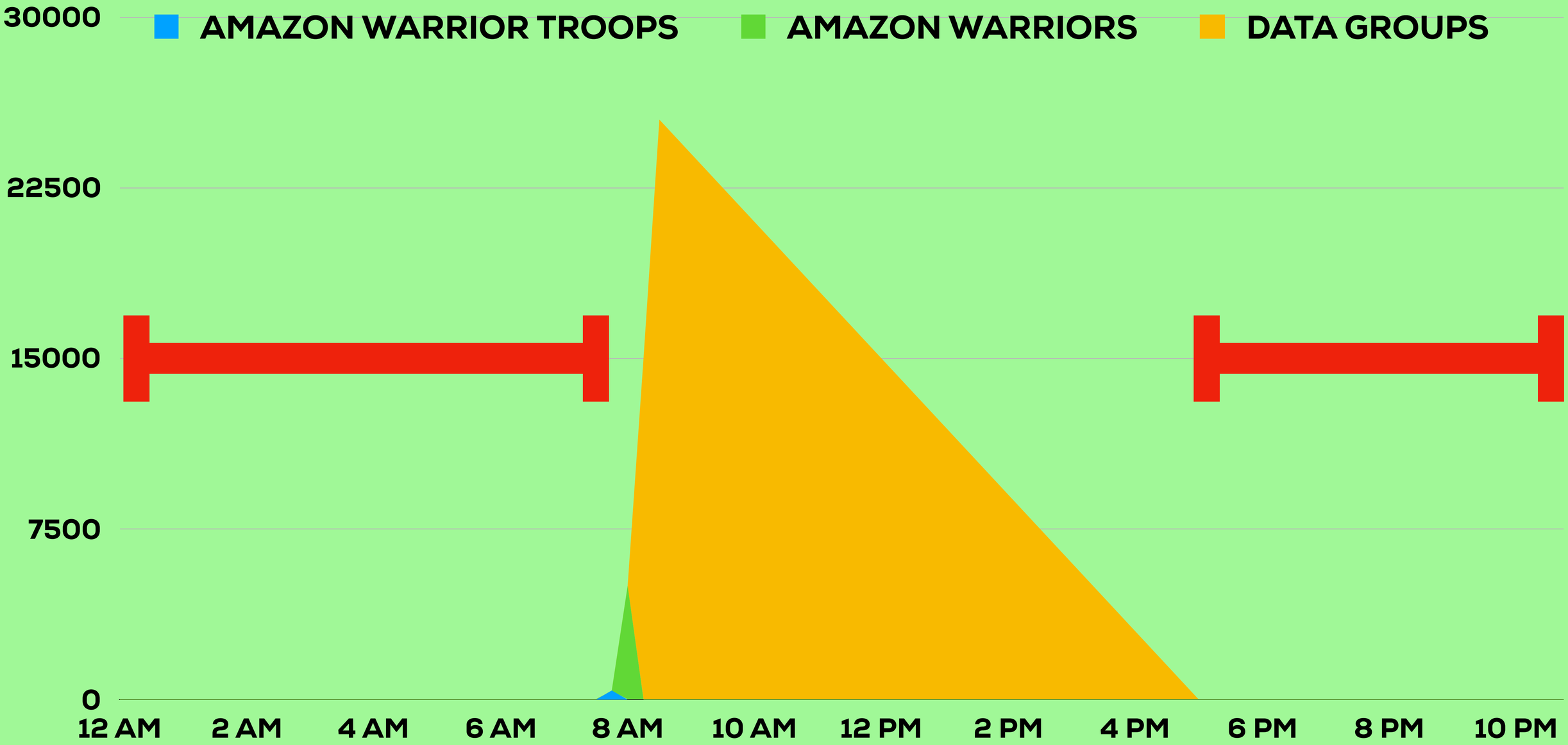
@BigDana

Queue Depth Over Time



@BigDana

Queue Depth Over Time

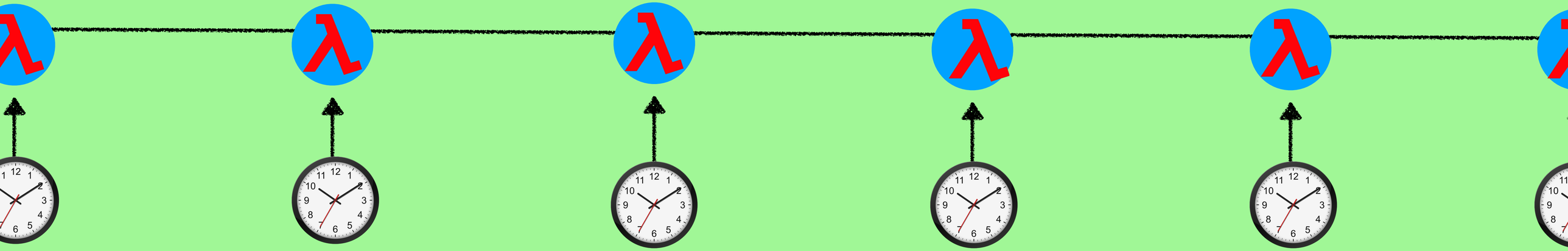


Rate limiting with Queues and Cloudwatch triggered Lambdas

```
18      while True:
19          try:
20              print('Reading up to 10 messages from device_queue...')
21              messages = device_queue.receive_messages(MaxNumberOfMessages=10)
22              got_messages = True
23          except botocore.exceptions.ClientError as e:
24              print('Error: failed to read from device_queue: {0}'.format(e))
25              got_messages = False
```

@BigDana

Rate limiting with Queues and Cloudwatch triggered Lambdas

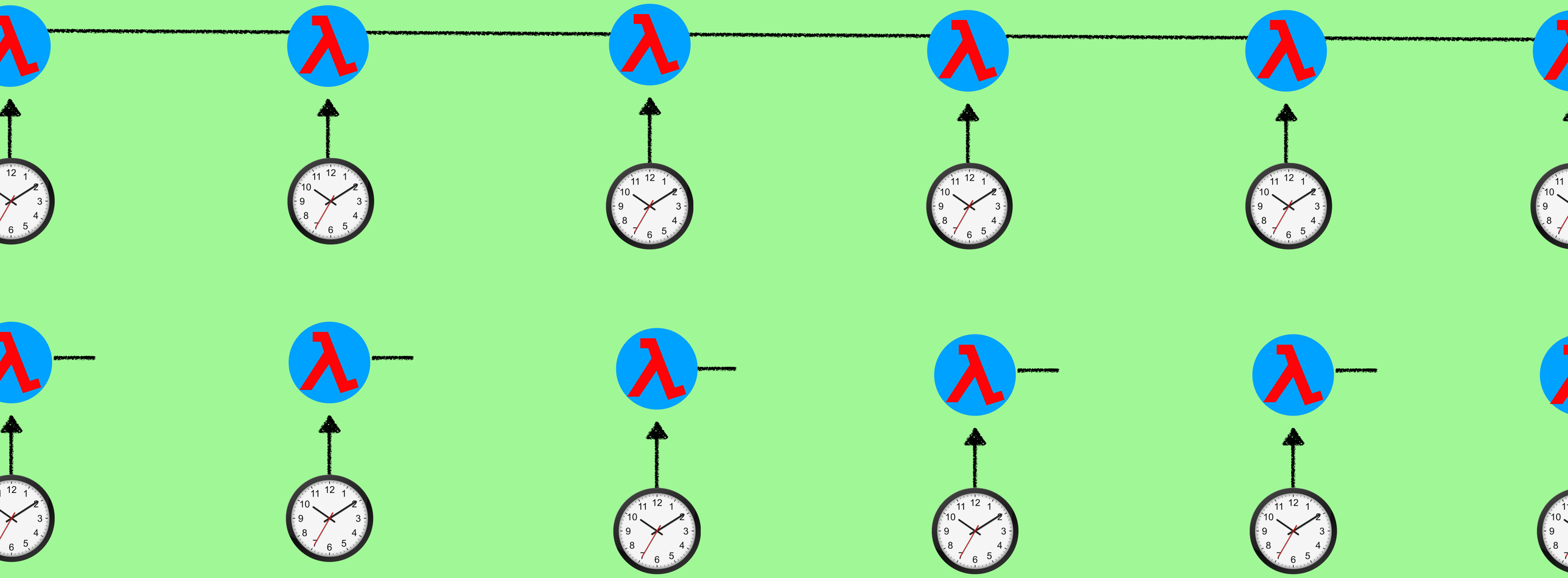


Rate limiting with Queues and Cloudwatch triggered Lambdas

```
18      while True:
19          try:
20              print('Reading up to 10 messages from device_queue...')
21              messages = device_queue.receive_messages(MaxNumberOfMessages=10)
22              got_messages = True
23          except botocore.exceptions.ClientError as e:
24              print('Error: failed to read from device_queue: {0}'.format(e))
25              got_messages = False
26
27          if len(messages) == 0:
28              print("No messages on the queue, exiting lambda...")
29              break
```

@BigDana

Rate limiting with Queues and Cloudwatch triggered Lambdas



SQS Queues:

highly scalable and resilient

can store failed tasks indefinitely

gives you control to rate limit the processing of the tasks in the queue

a small task can wait in line on a queue that already has many tasks

Lambdas triggered by Cloudwatch Rules

allows for quite specific schedules

fastest schedule is
every 1 minute

doesn’t scale responsively to
your queue depth

@BigDana

IT TAKES REAL CHARACTER TO ADMIT ONES FAILURES — AND NOT A LITTLE WISDOM TO TAKE YOUR PROFITS FROM DEFEAT

Takeaways

Lambdas are not yet suitable for quick tasks if they require large dependencies

Step Functions does not yet provide dynamic fan out parallel processing

Event triggers don’t yet allow you to rate limit

We can’t yet trigger a Lambda based off an SQS Queue depth

Lessons Learned

Expect Limits

Expect to Exceed those limits

Expect you will have to handle failures

Reflections for Managers

Invest in training new hires

Decouple the learning process of your build deploy pipeline and building serverless architecture

Victoria Pierce

@thesecondshade

June 5th 2017:
Joined SPS Commerce as a Site Reliability Engineer Intern

July 12th 2017:
Deployed her first lambda

As of November 15th 2017:
Maintains 6 serverless applications

@BigDana

Thank you!

Dana Engebretson
Performance Engineer
SPS Commerce

Questions?