AWS re:Invent

SVS407-R

Architecting and operating resilient serverless systems at scale

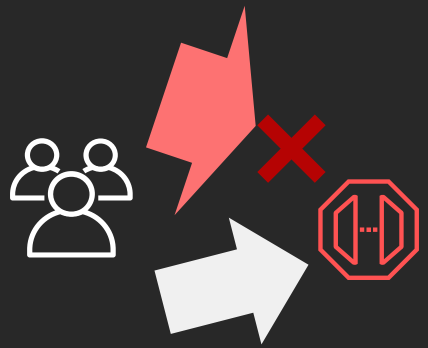
David Yanacek

Principal Engineer
AWS Lambda
Amazon Web Services

AWS re:Invent

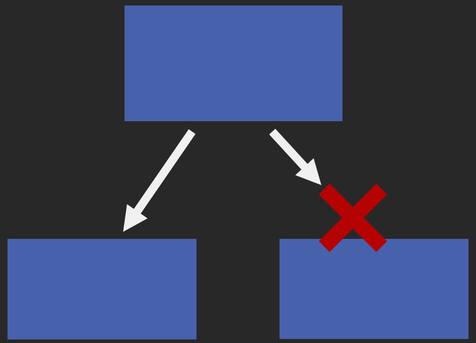
aws

Table of contents



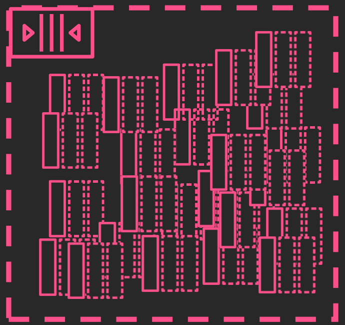
Load shedding

Avoid brownout by rejecting excess load



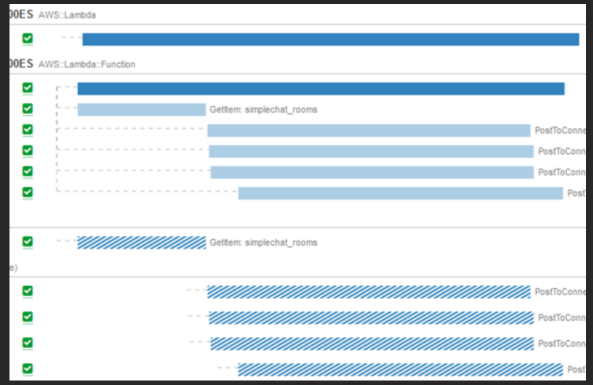
Dependency isolation

Prevent one dependency from affecting unrelated functionality



Avoiding queue backlogs

Prevent a backlog from extending recovery time



Operating

Quickly diagnose and mitigate issues

Structure

Theory

Serverless
resiliency tools

Motivation for
serverless

AWS under
the hood

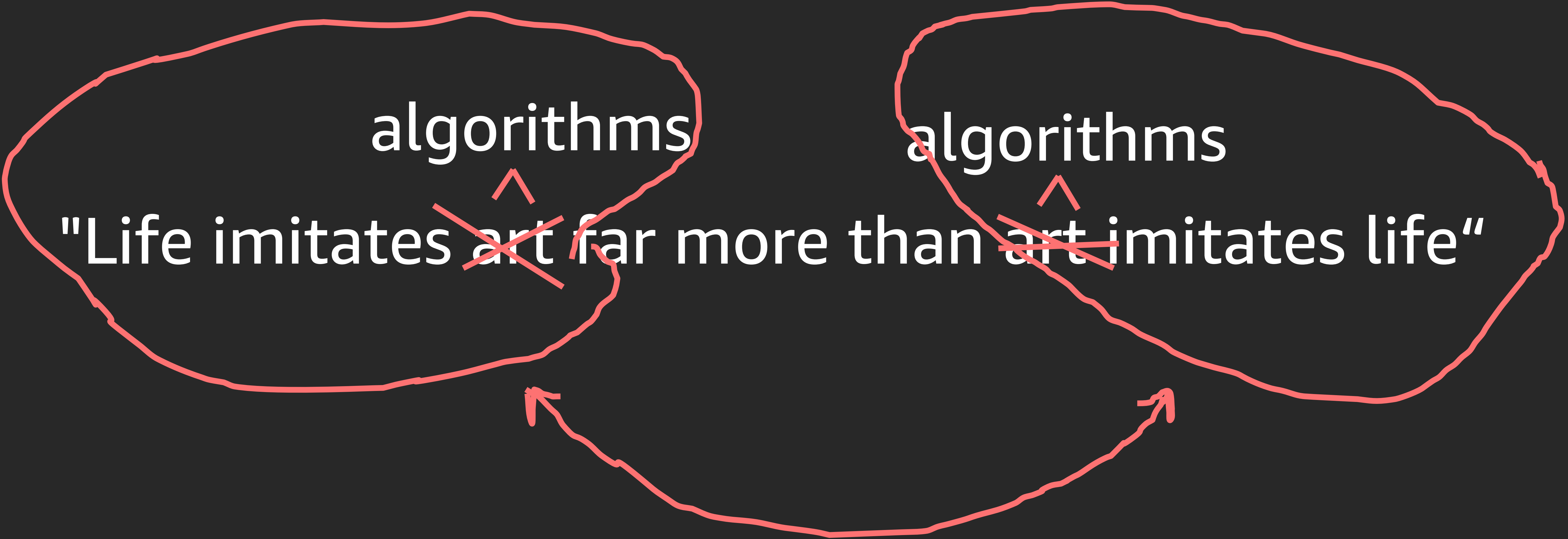
Overload, illustrated

"Life imitates art far more than art imitates life“

- Oscar Wilde, 1889

algorithms algorithms

"Life imitates ~~art~~ far more than ~~art~~ imitates life“



algorithms algorithms

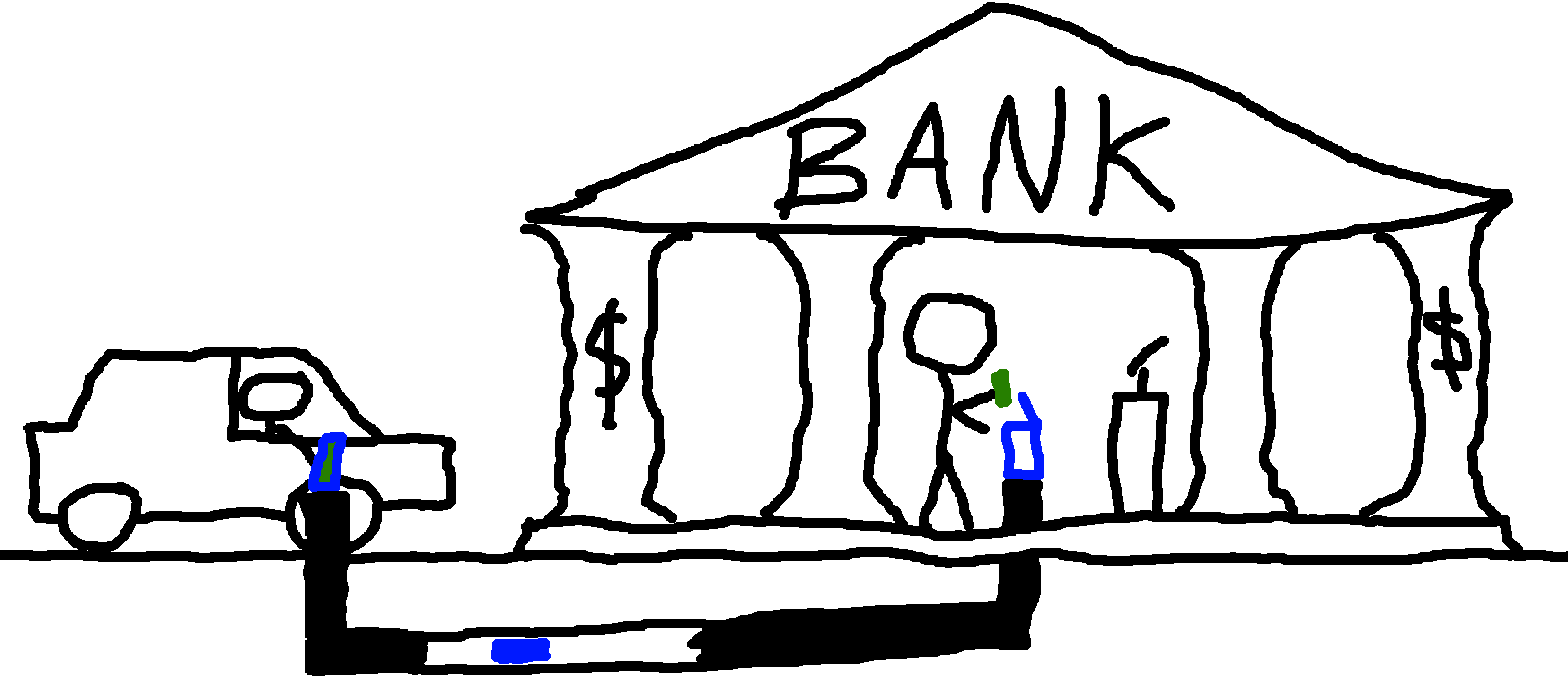
"Life imitates ~~art~~ far more than ~~art~~ imitates life“

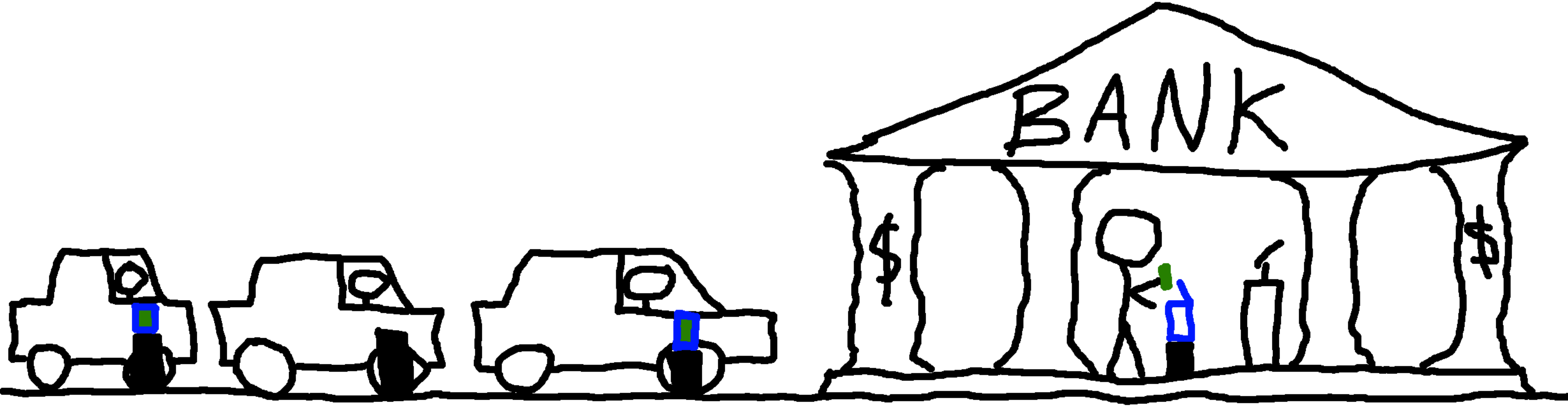
“Algorithms imitate life far more than life imitates algorithms“

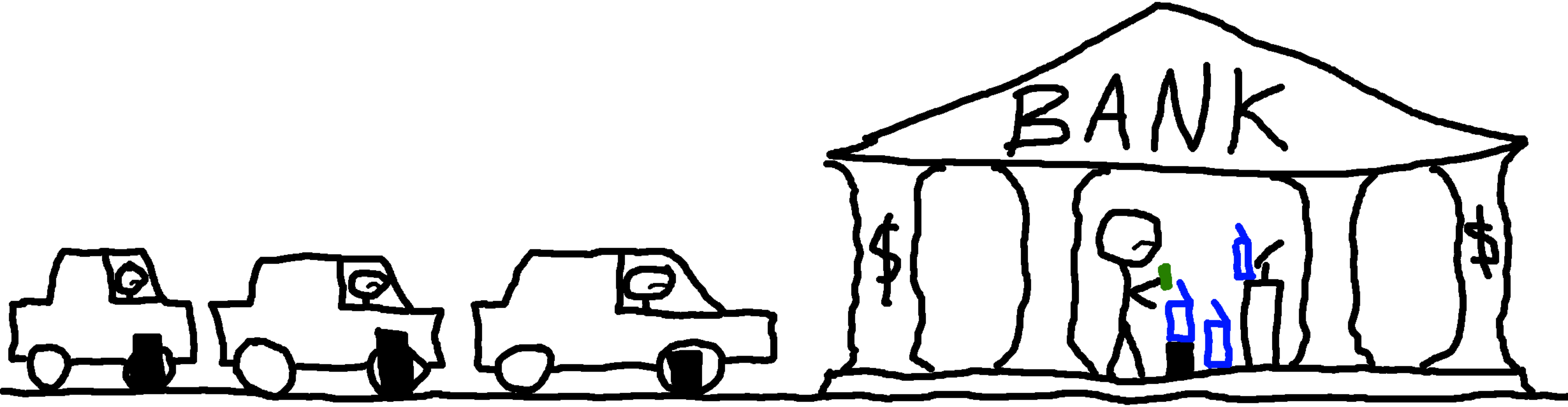
- Somebody, probably

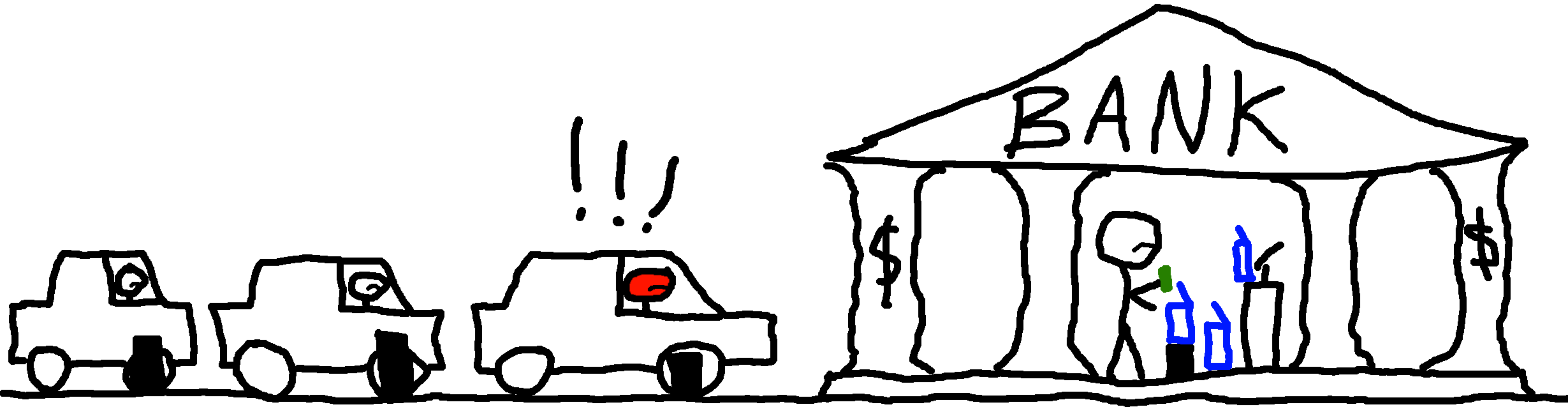
Back in high school…

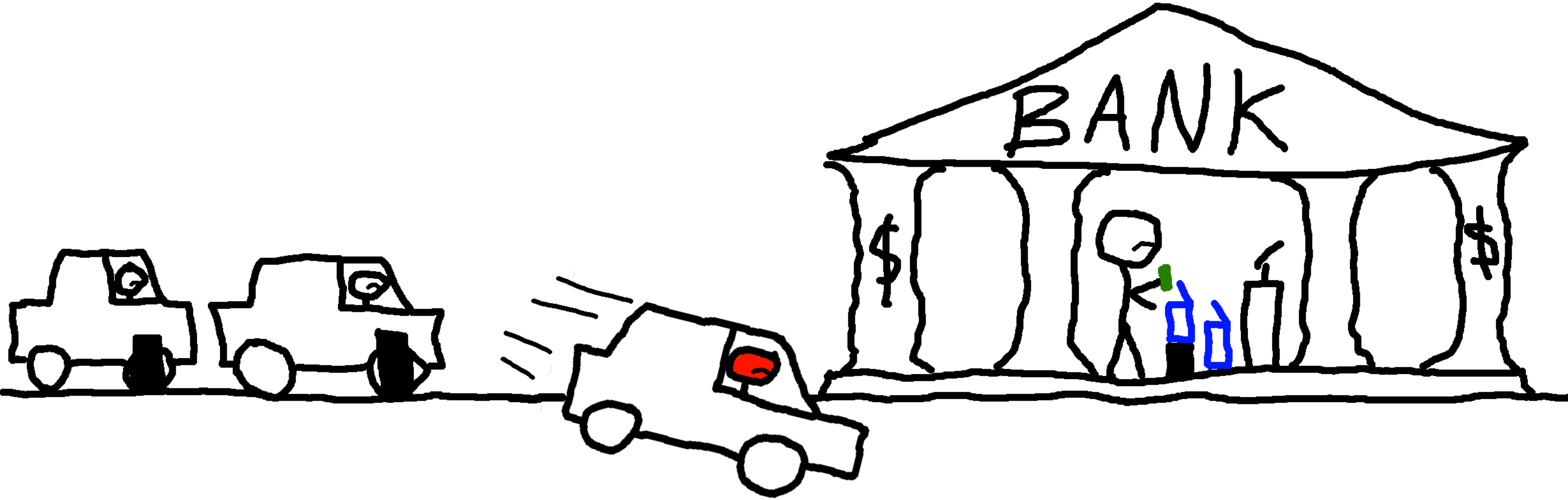
BANK
$
$


BANK
$
$


BANK
$
$


BANK
$
$

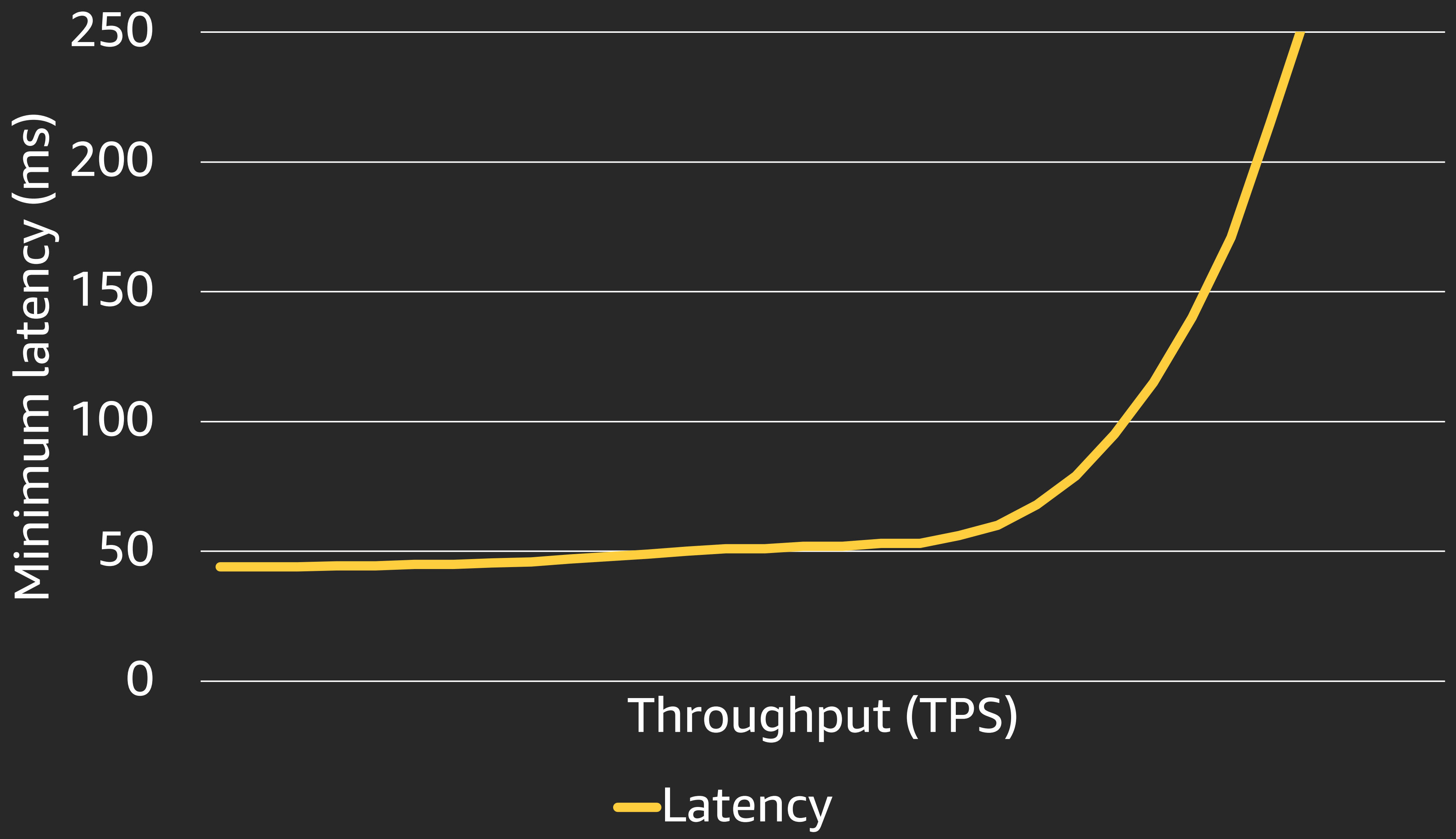

!!!
BANK
$
$

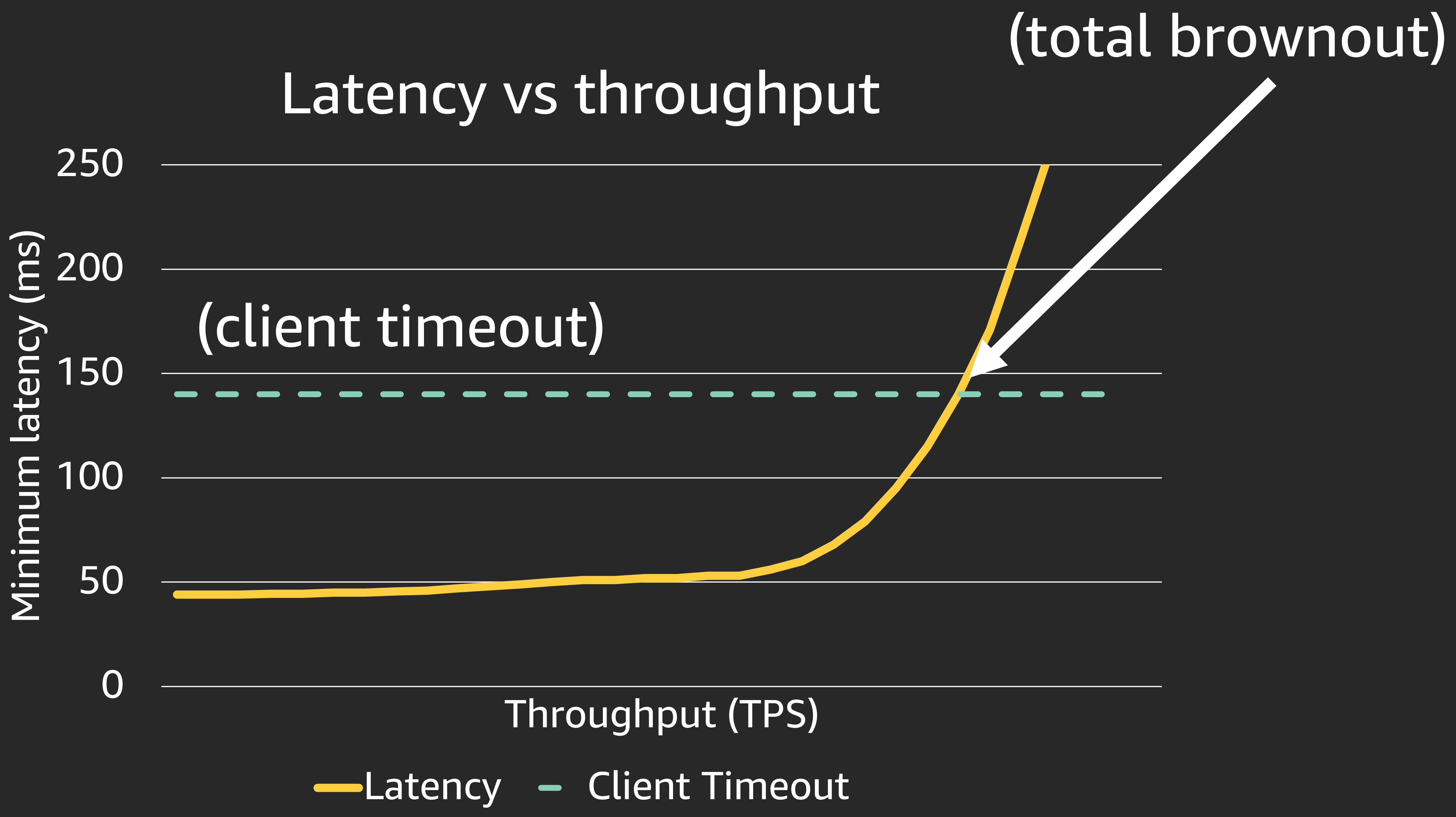

BANK
$
$

Amdahl’s Law, Universal Scalability Law

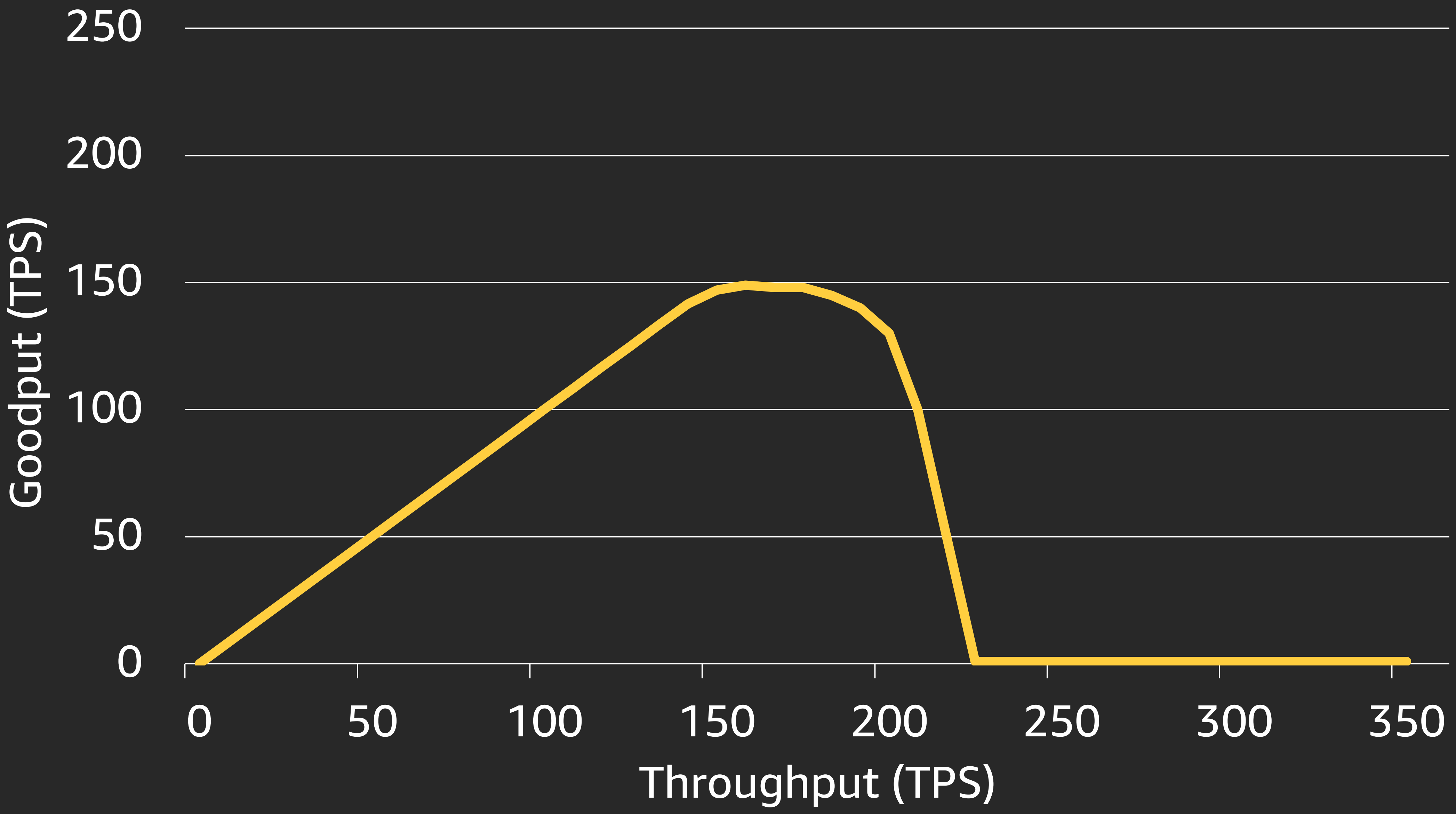
You can parallelize a system up to the point where contention becomes the bottleneck

Latency vs throughput

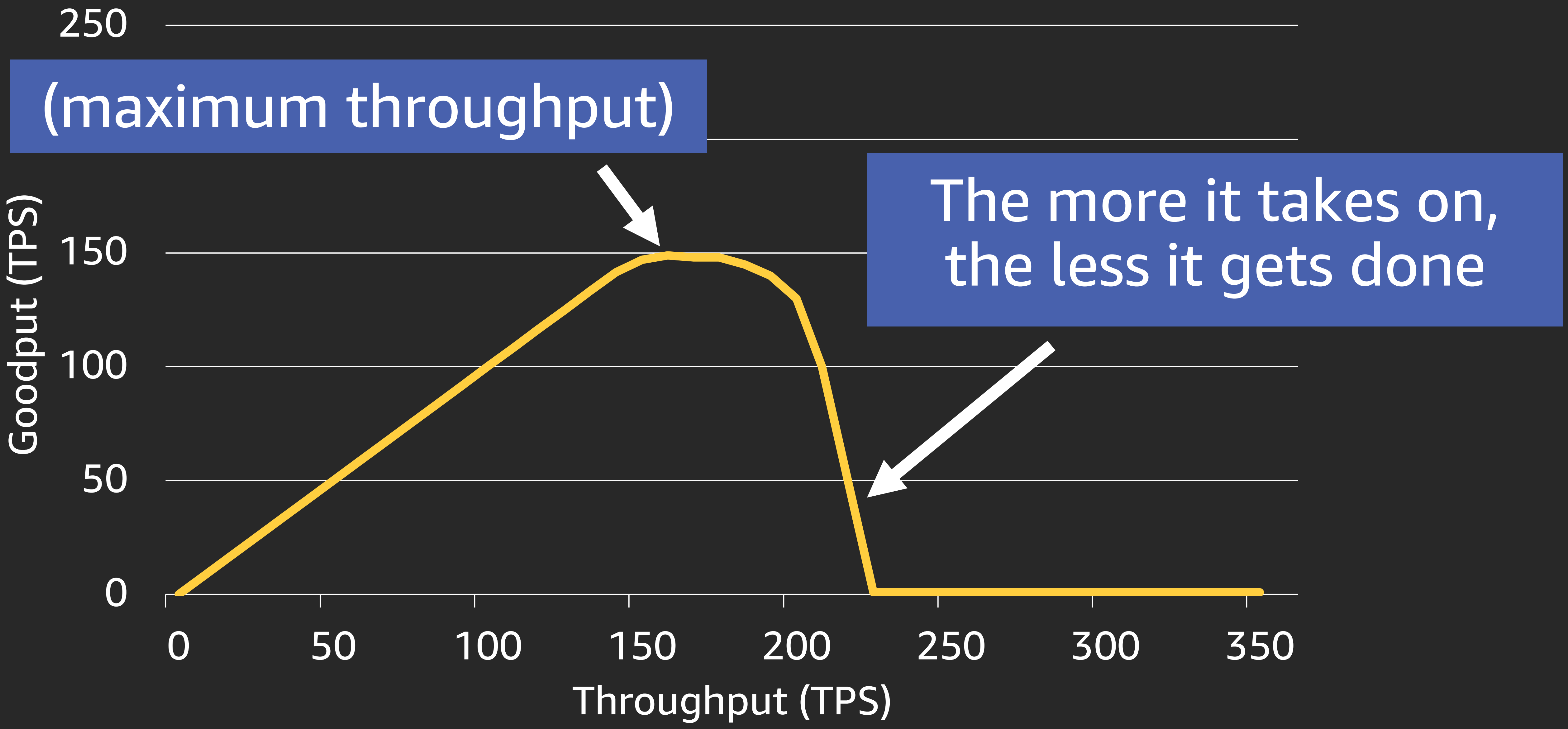




Goodput vs throughput



Goodput vs throughput

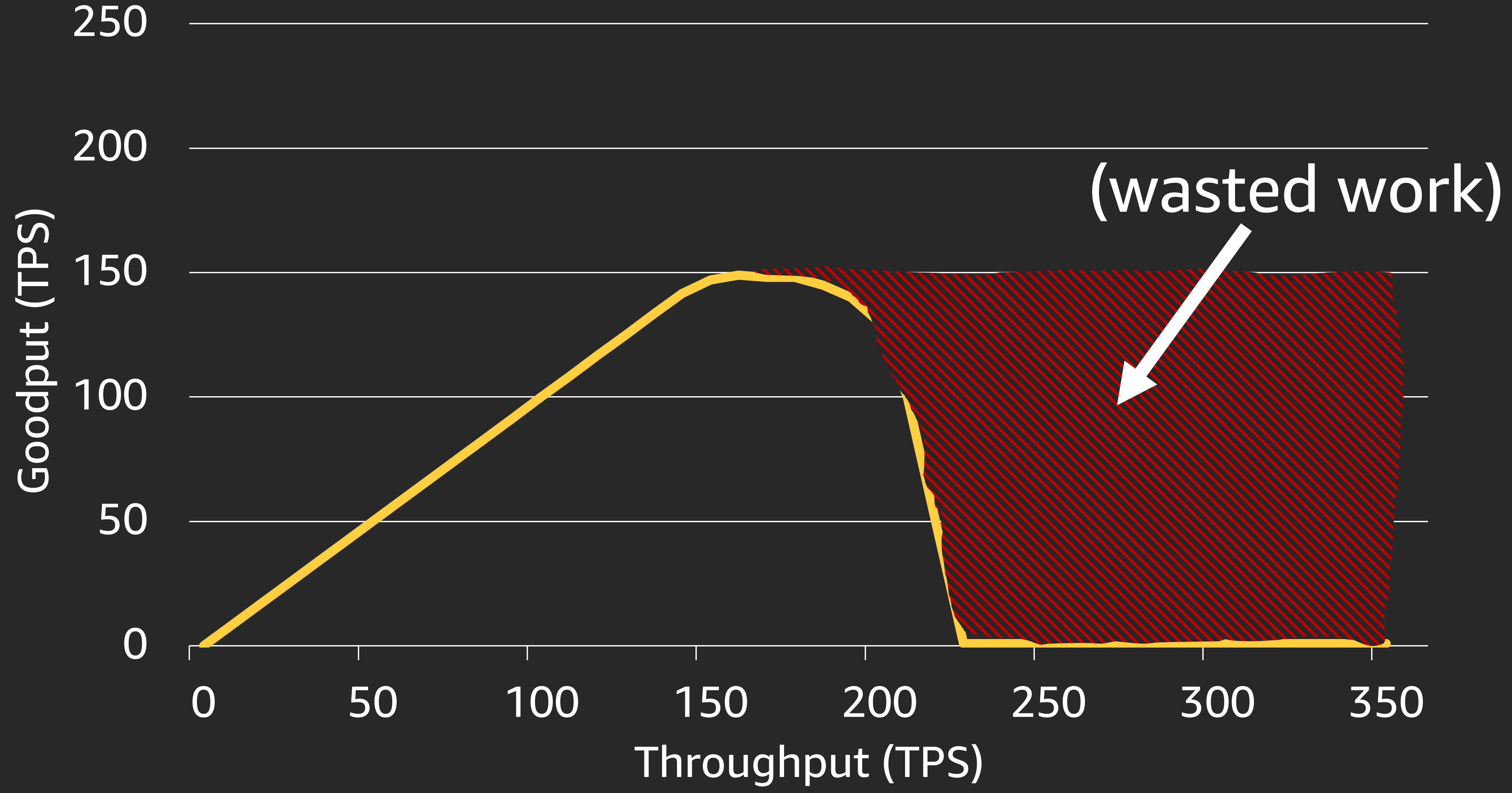


(maximum throughput)

The more it takes on, the less it gets done

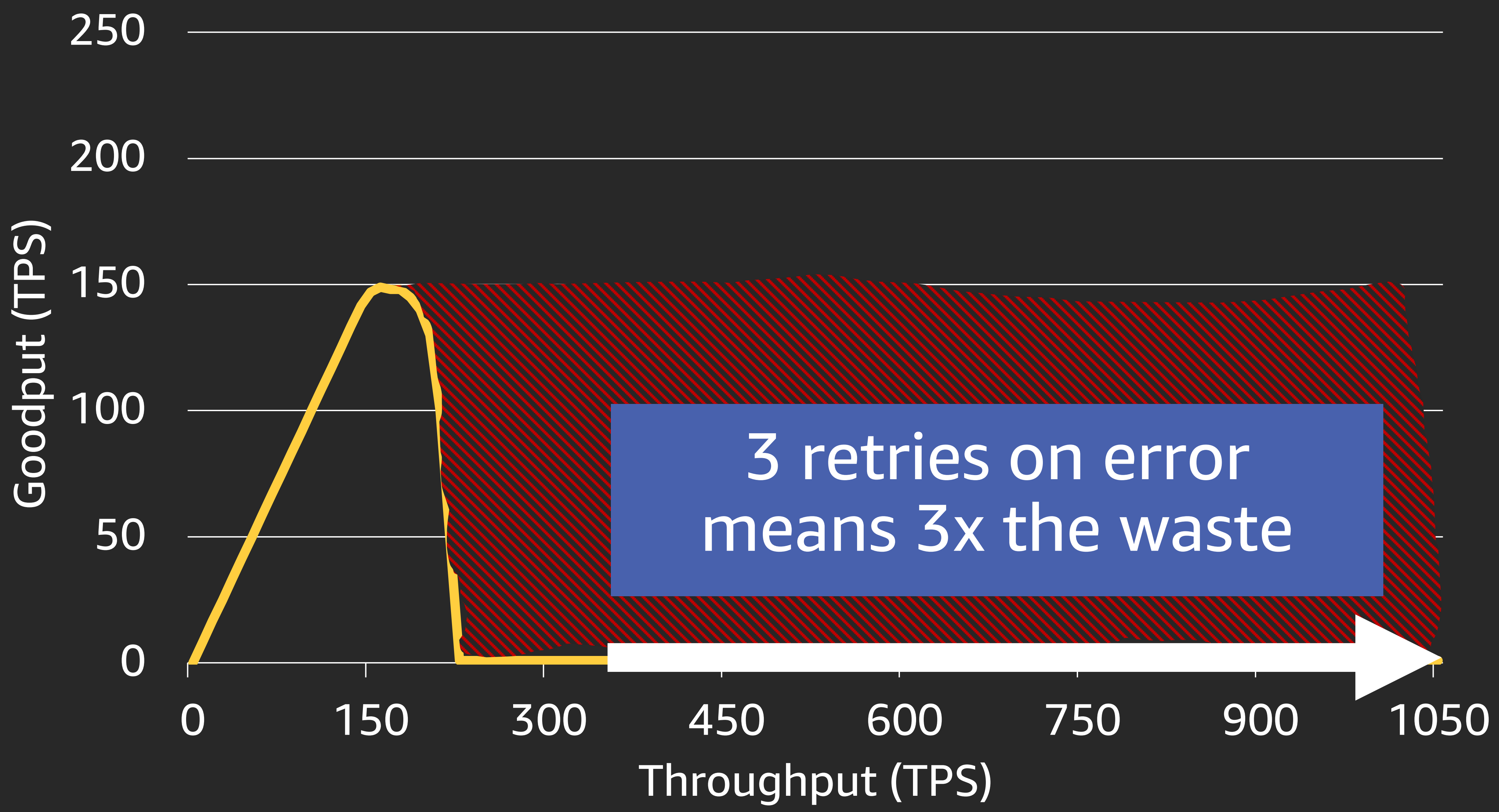
Servers are too optimistic

Goodput vs throughput



Clients retry

Goodput vs throughput

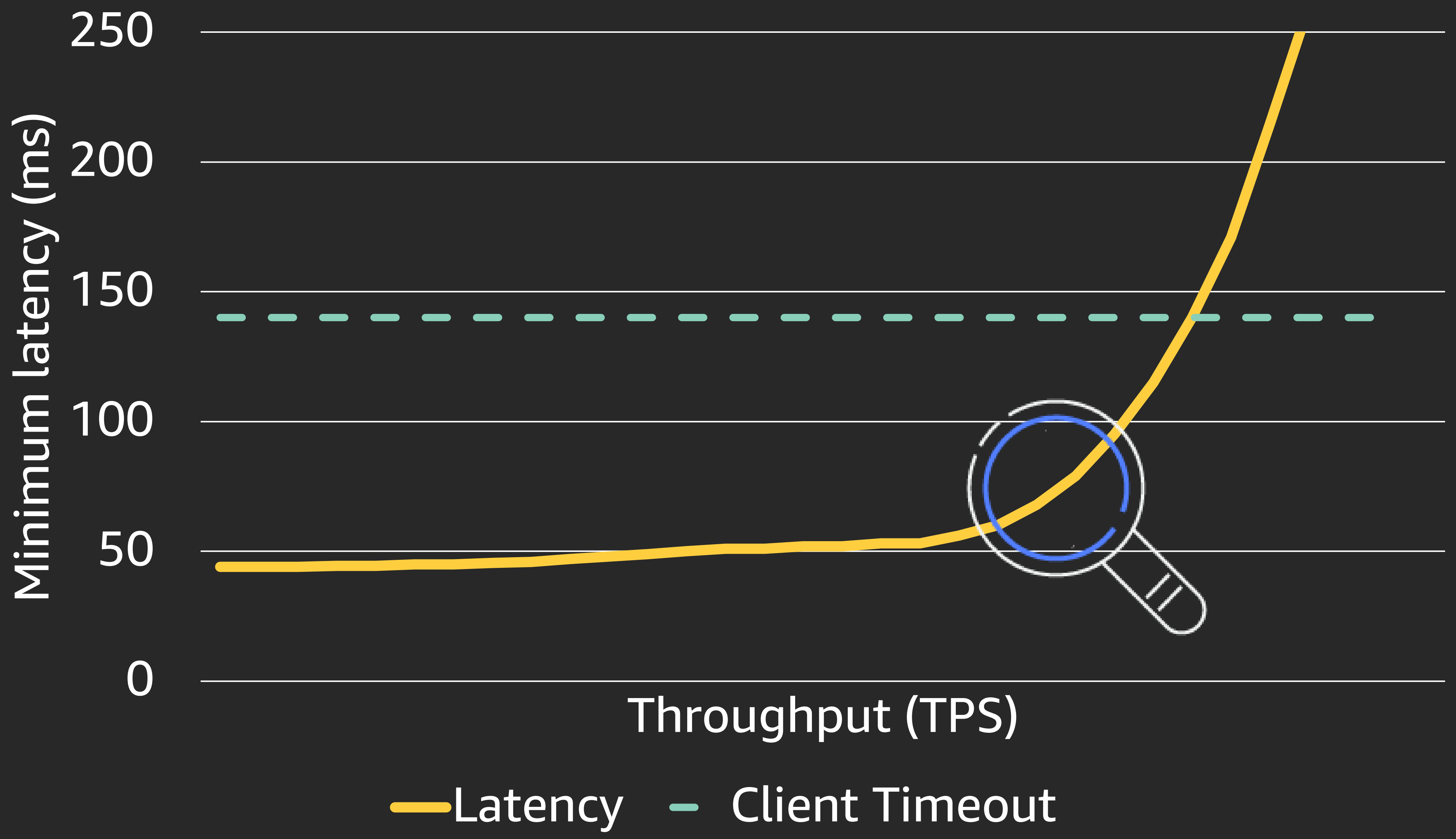


3 retries on error means 3x the waste

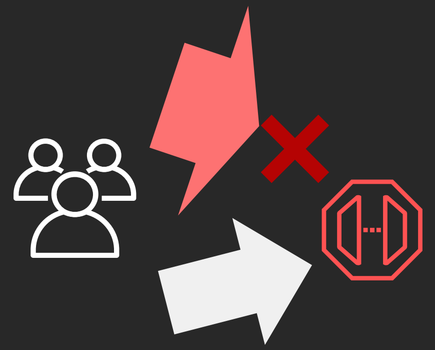
Amazon EC2 Auto Scaling

Understand tipping points

Latency vs throughput

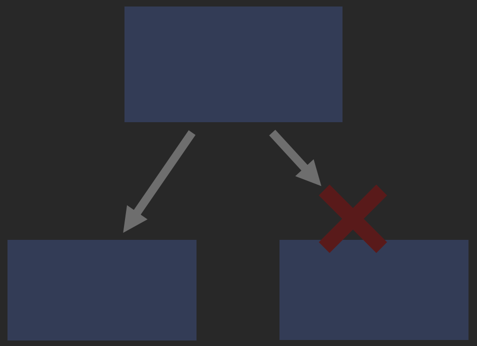


Chapter one



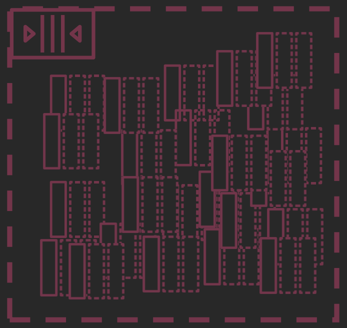
Load shedding

Avoid brownout by rejecting excess load



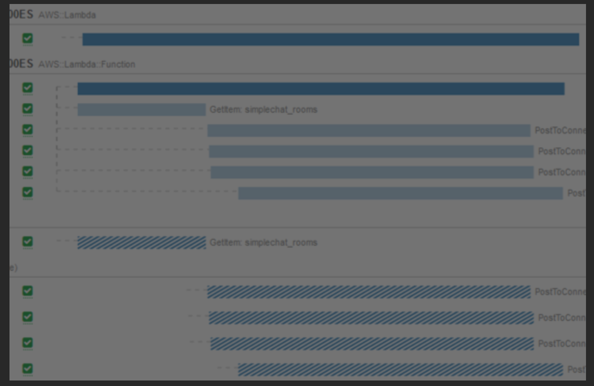
Dependency isolation

Prevent one dependency from affecting unrelated functionality



Avoiding queue backlogs

Prevent a backlog from extending recovery time

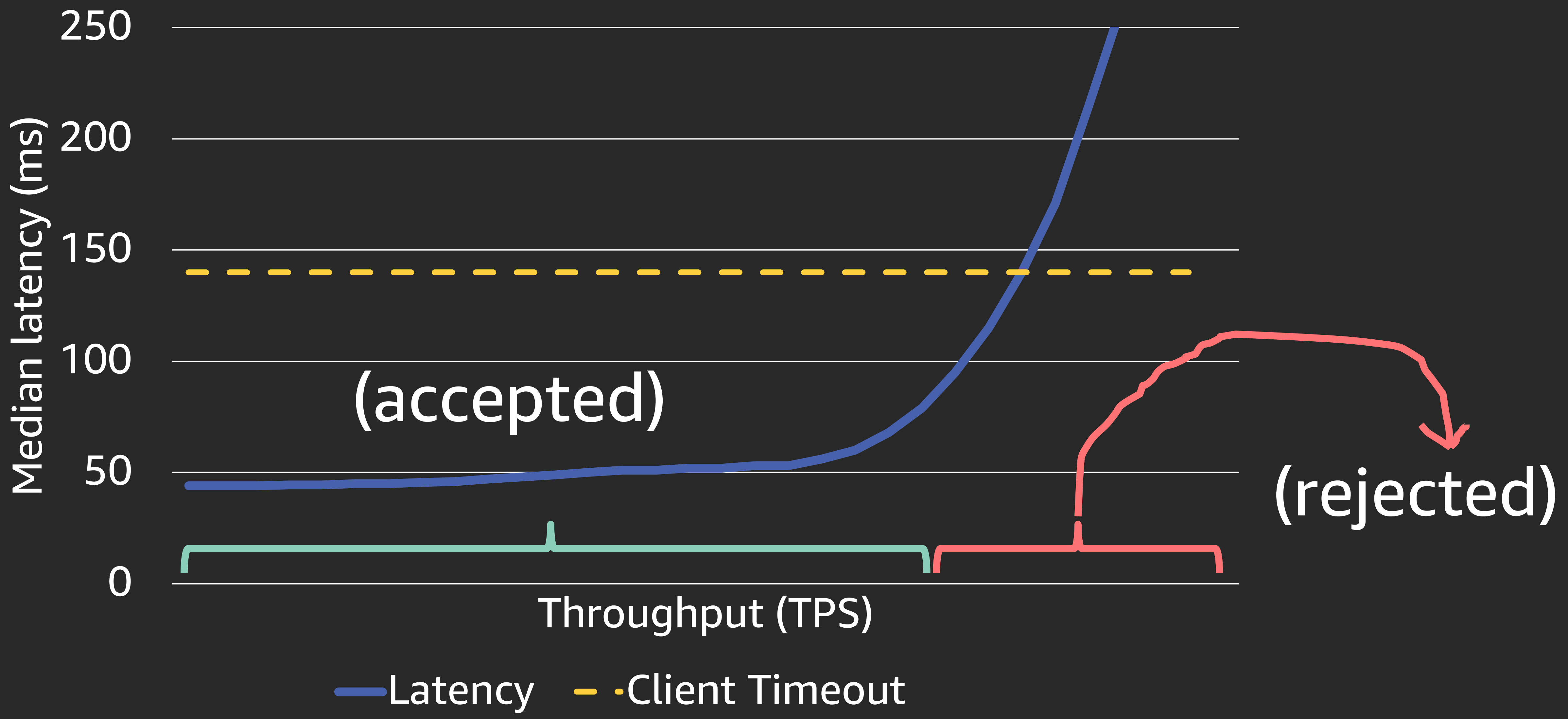


Operating

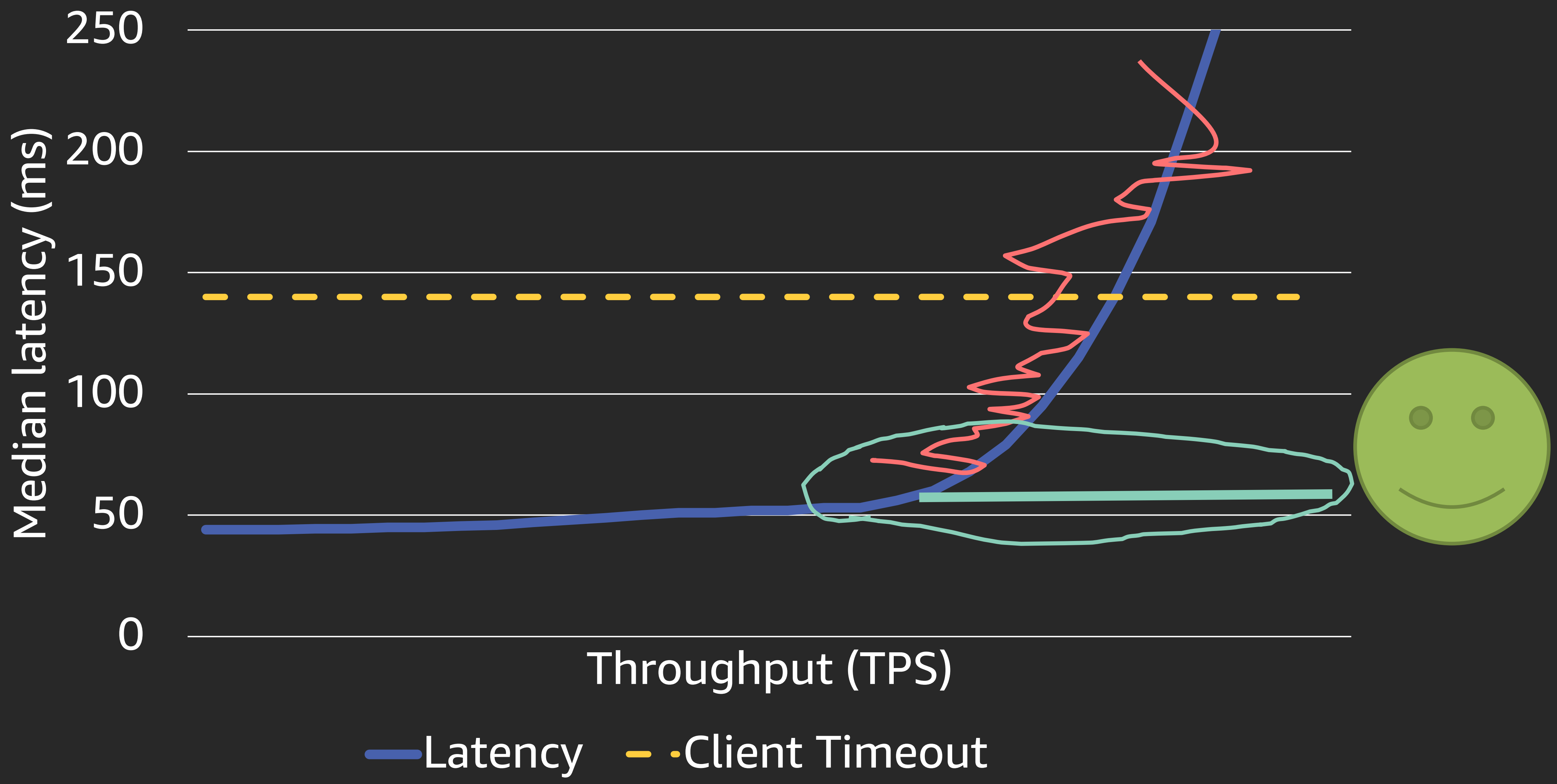
Quickly diagnose and mitigate issues

Cheaply reject excess work

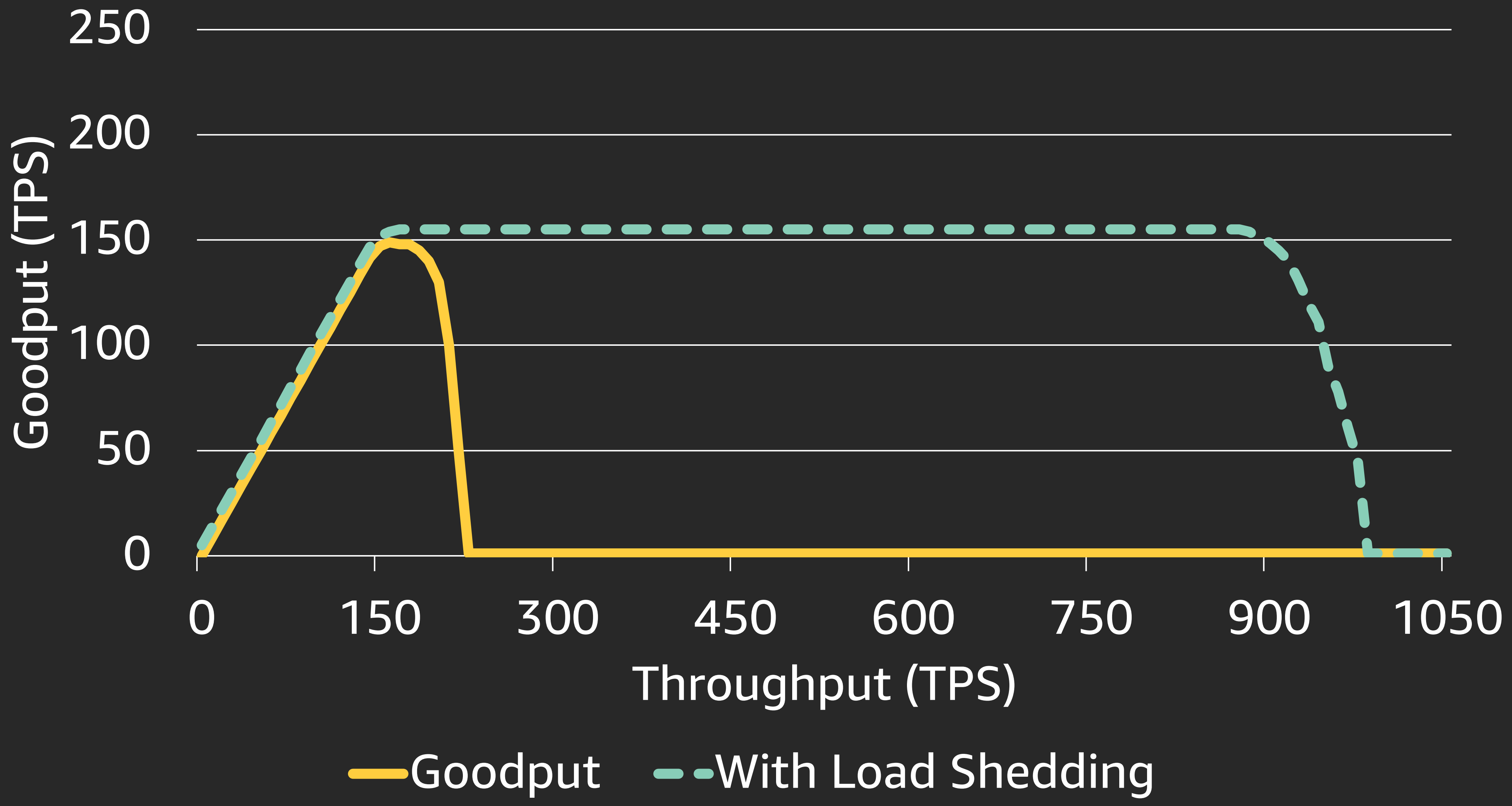
Latency vs throughput



Latency vs throughput

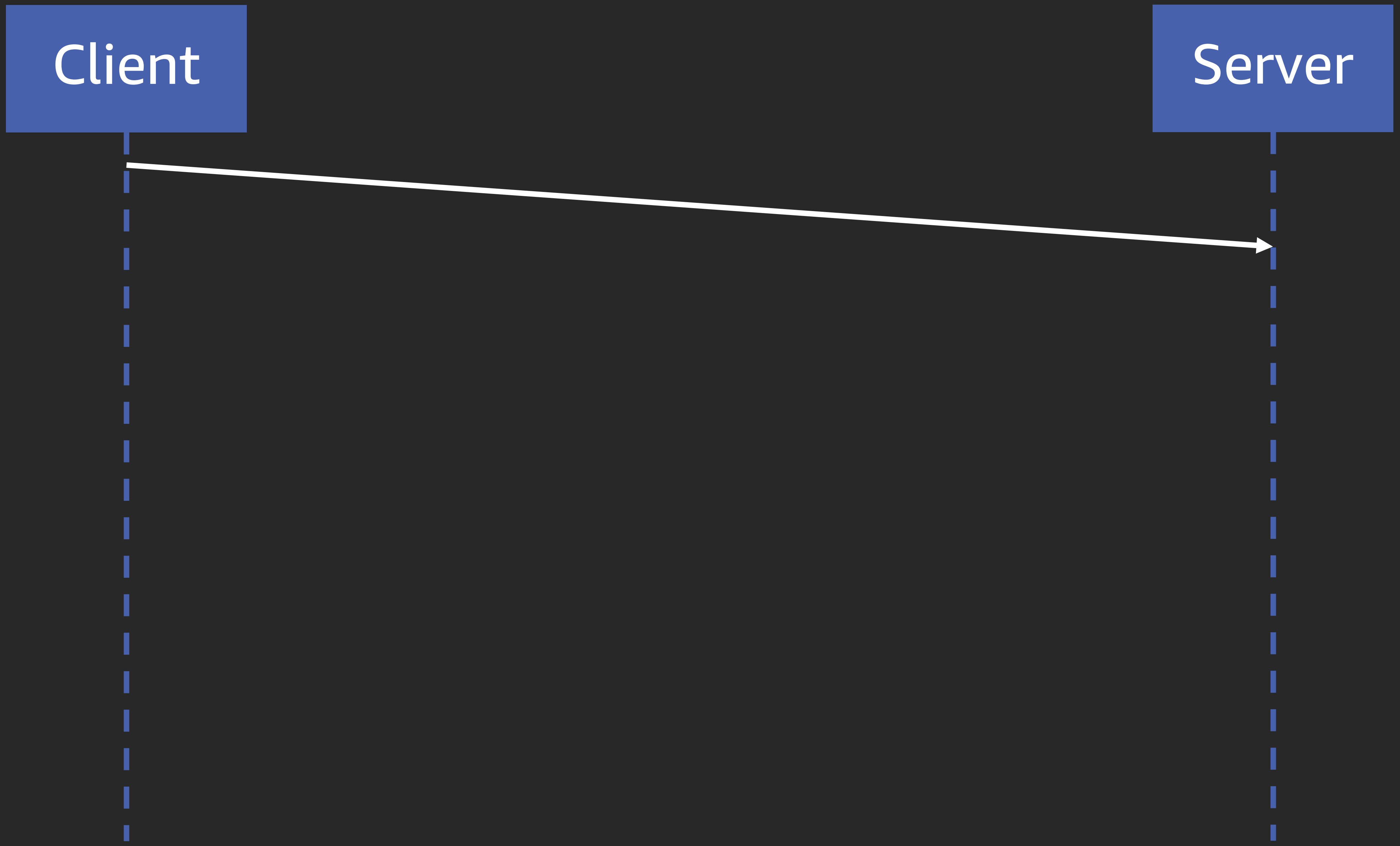


Goodput vs throughput, with and without load shedding

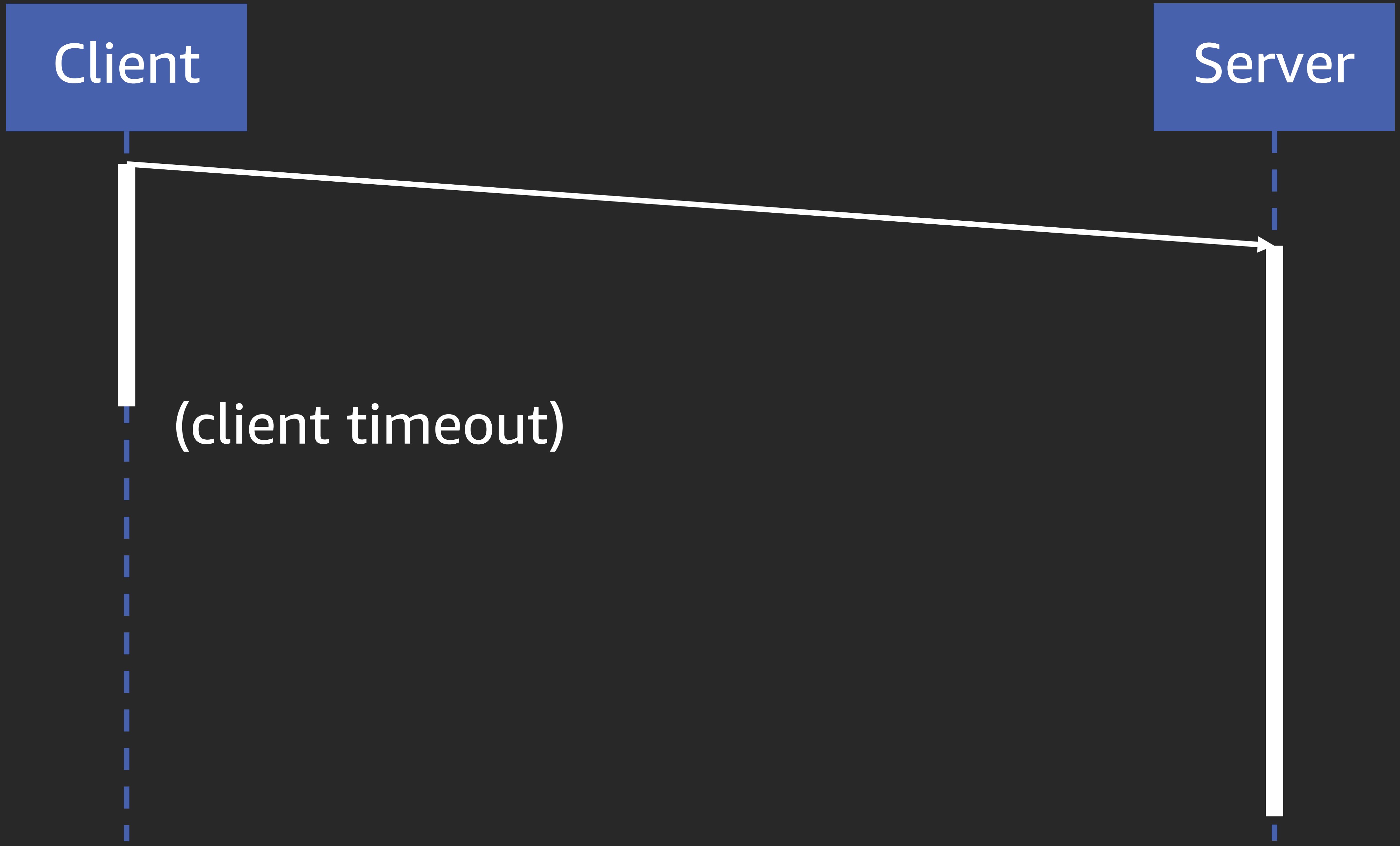


Don’t waste work

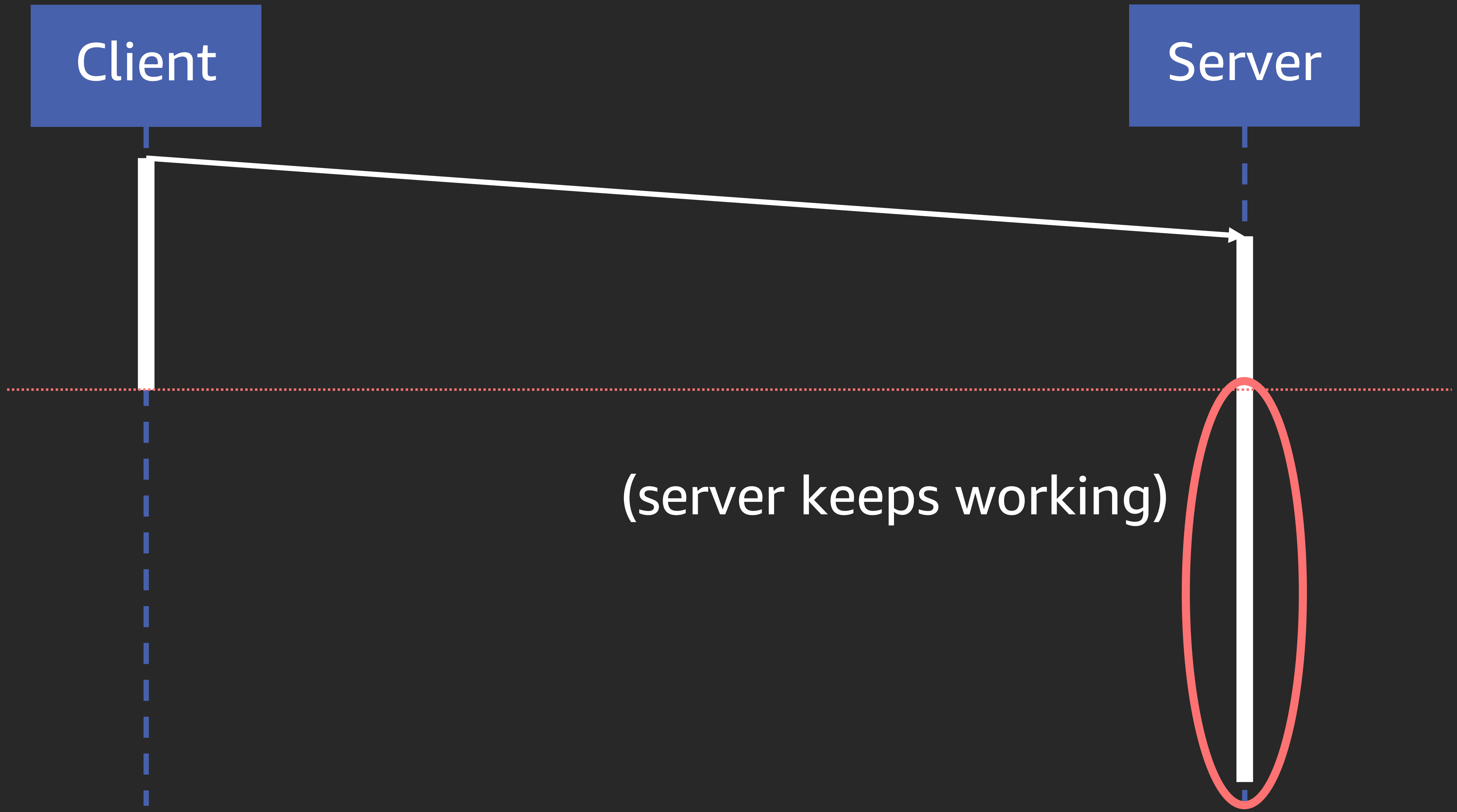
Timeout wasting work



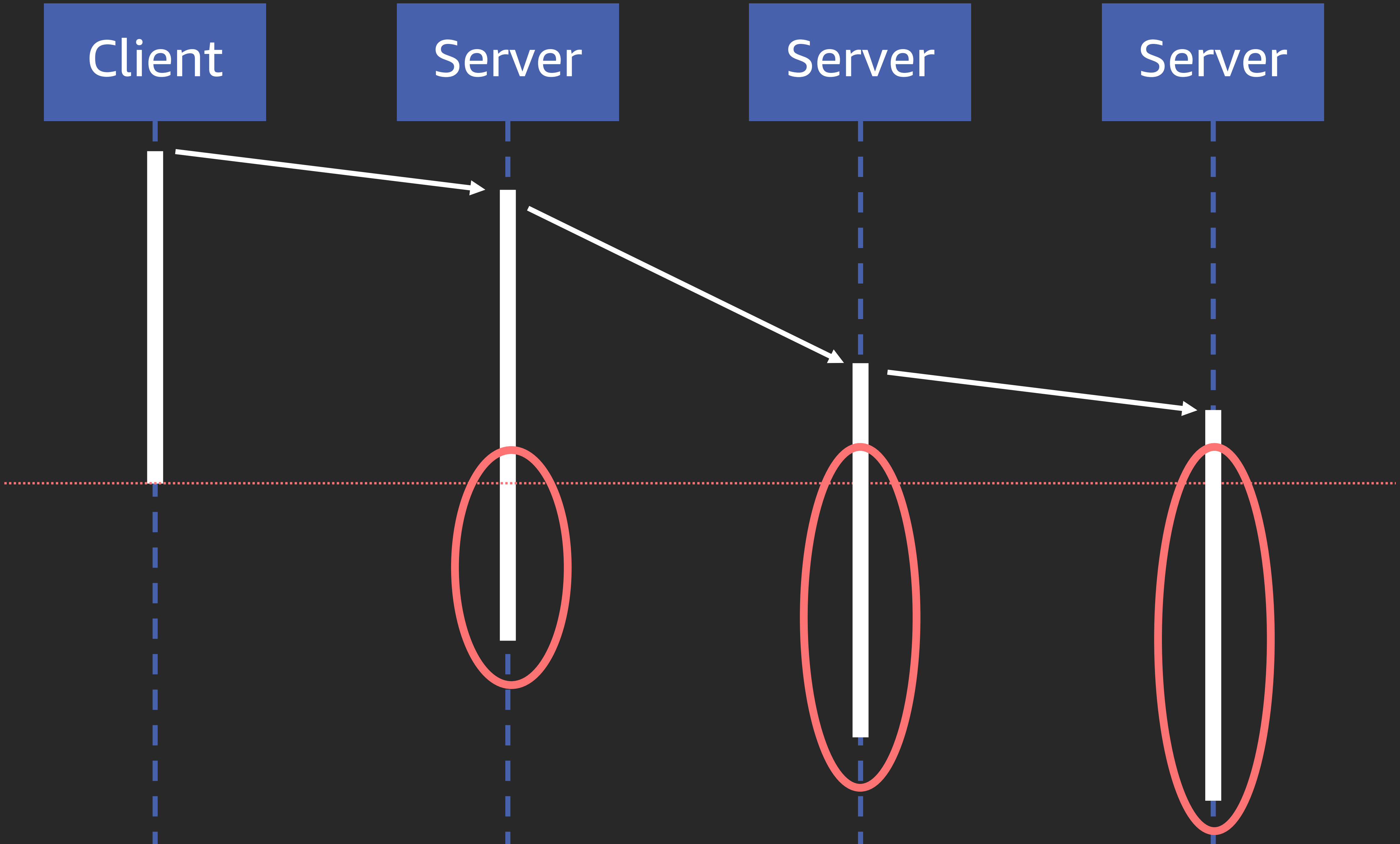
Timeout wasting work



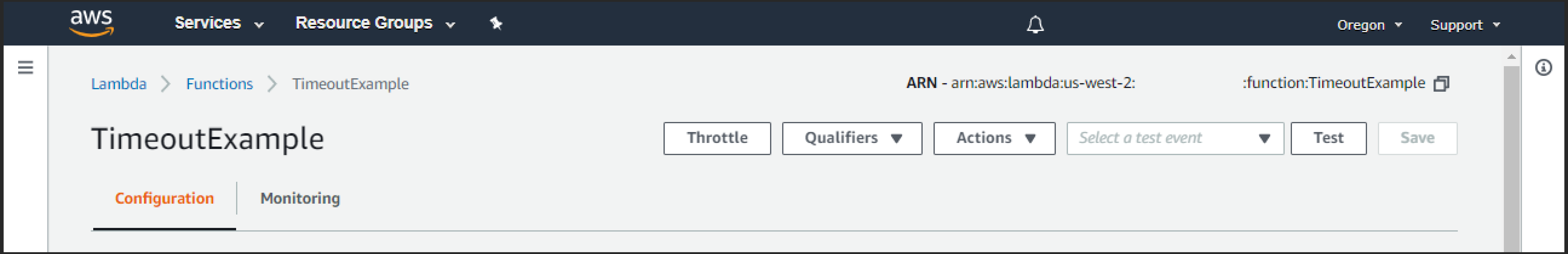
Timeout wasting work



Layers of waste

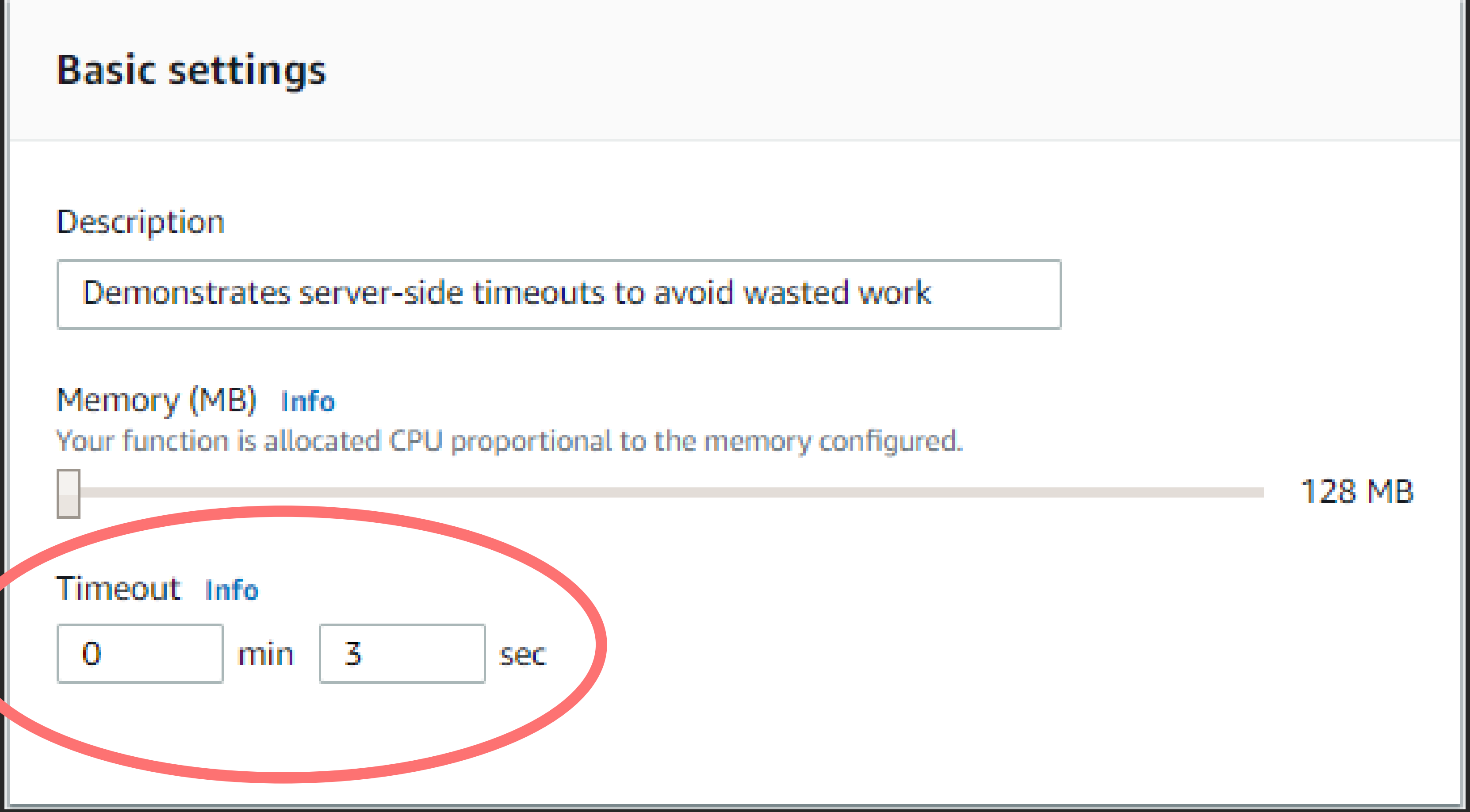


Server(less?)-side timeouts in Lambda



ARN - arn:aws:lambda:us-west-2: :function:TimeoutExample

TimeoutExample



Basic settings

Description

Demonstrates server-side timeouts to avoid wasted work

Memory (MB) Info

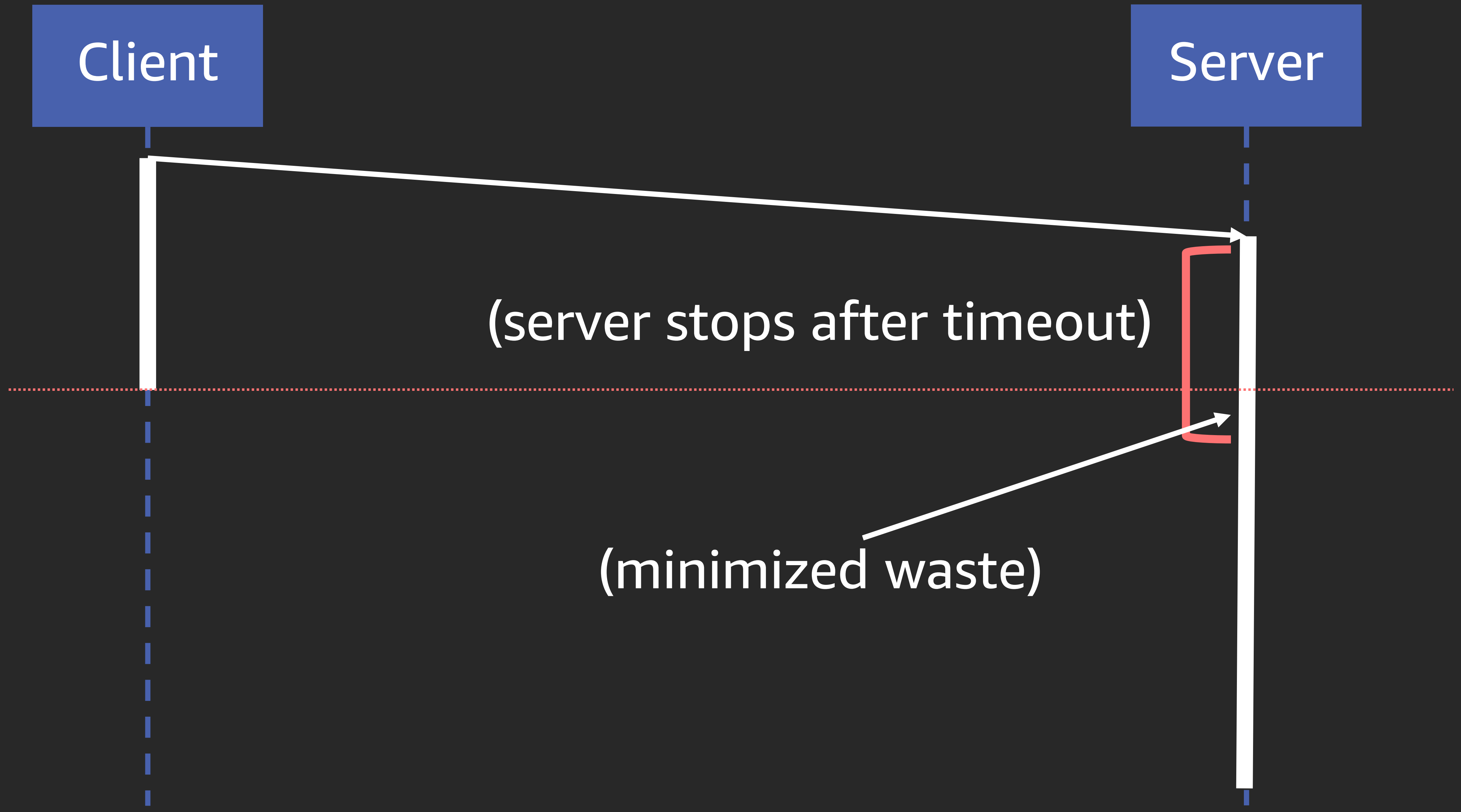
Your function is allocated CPU proportional to the memory configured.

128 MB

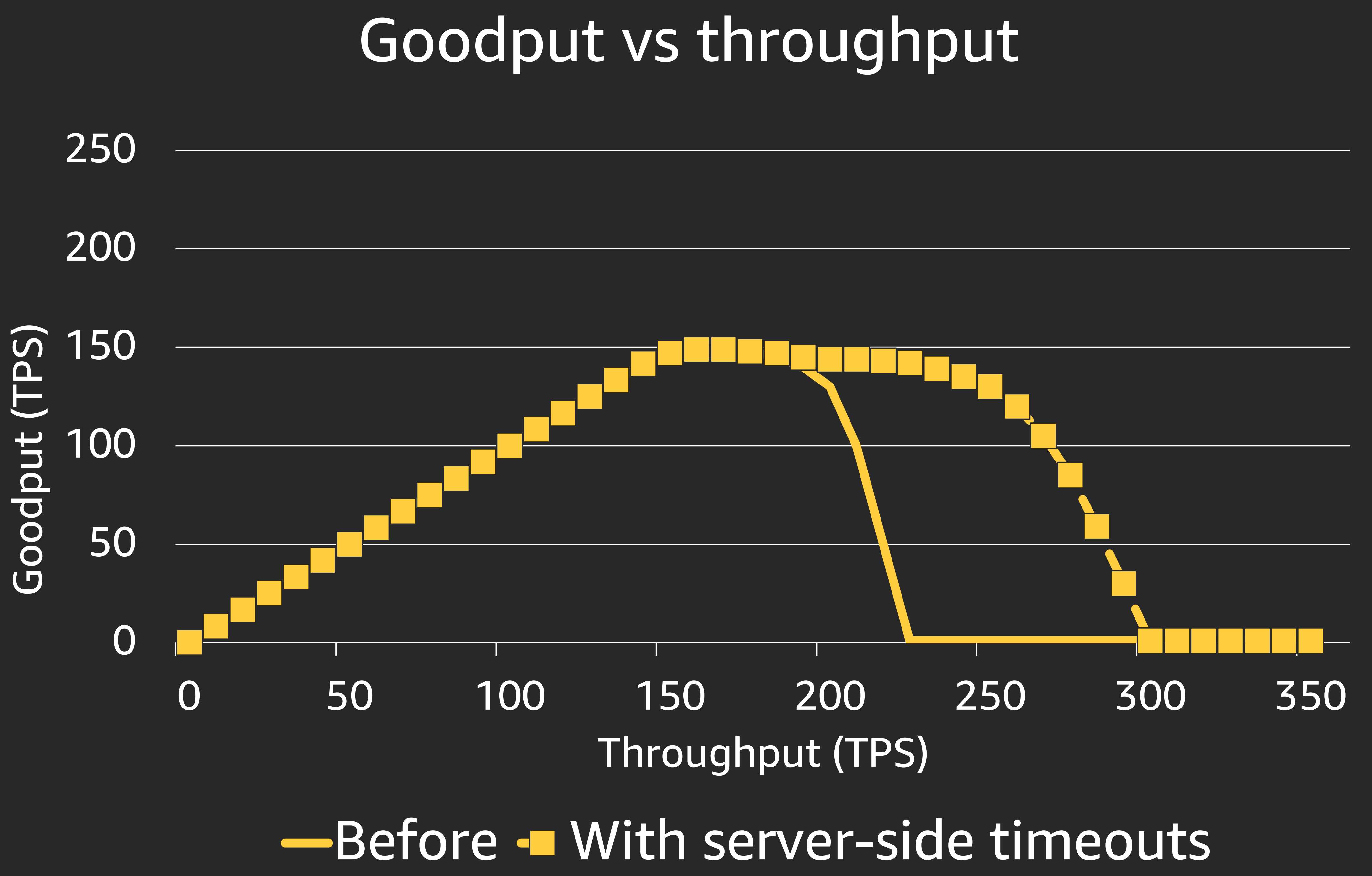
Timeout Info

0 min 3 sec

Server-side timeouts



Better, still not great



(Un?)Intended consequences?

Expensive requests?

Slow dependency?

Bounded work

Input size validation

Pagination

Checkpointing

Checkpointing

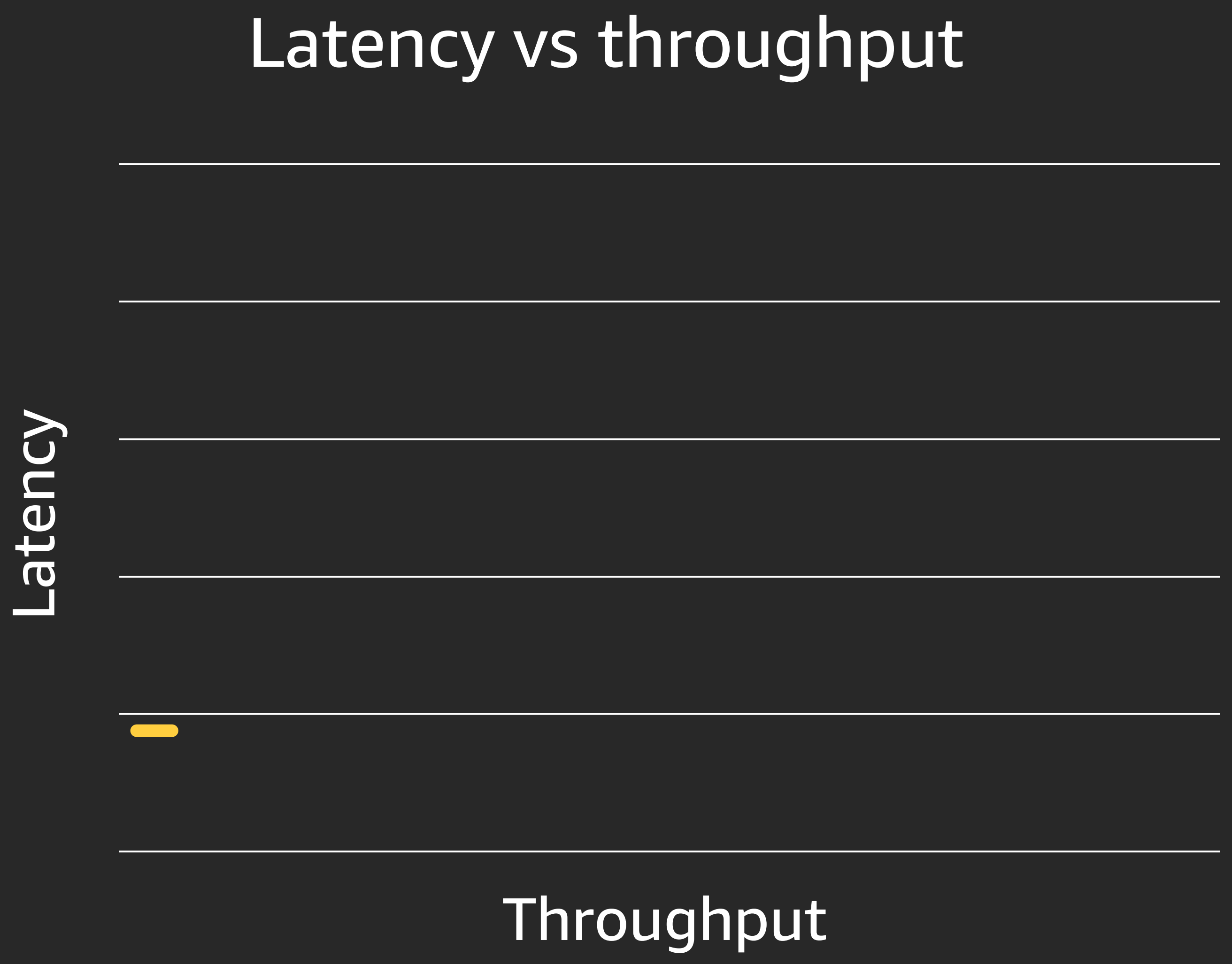
Checkpointing

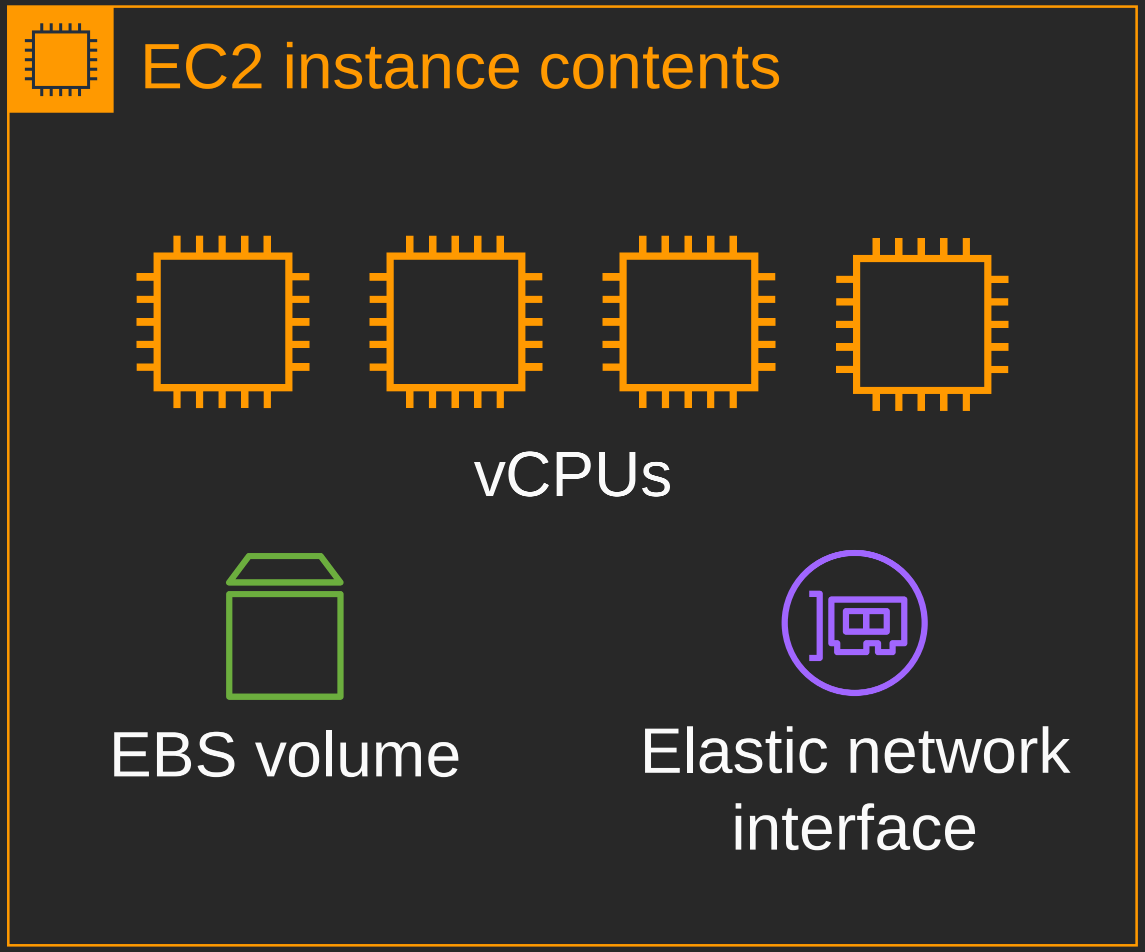
Checkpointing

Bounded work

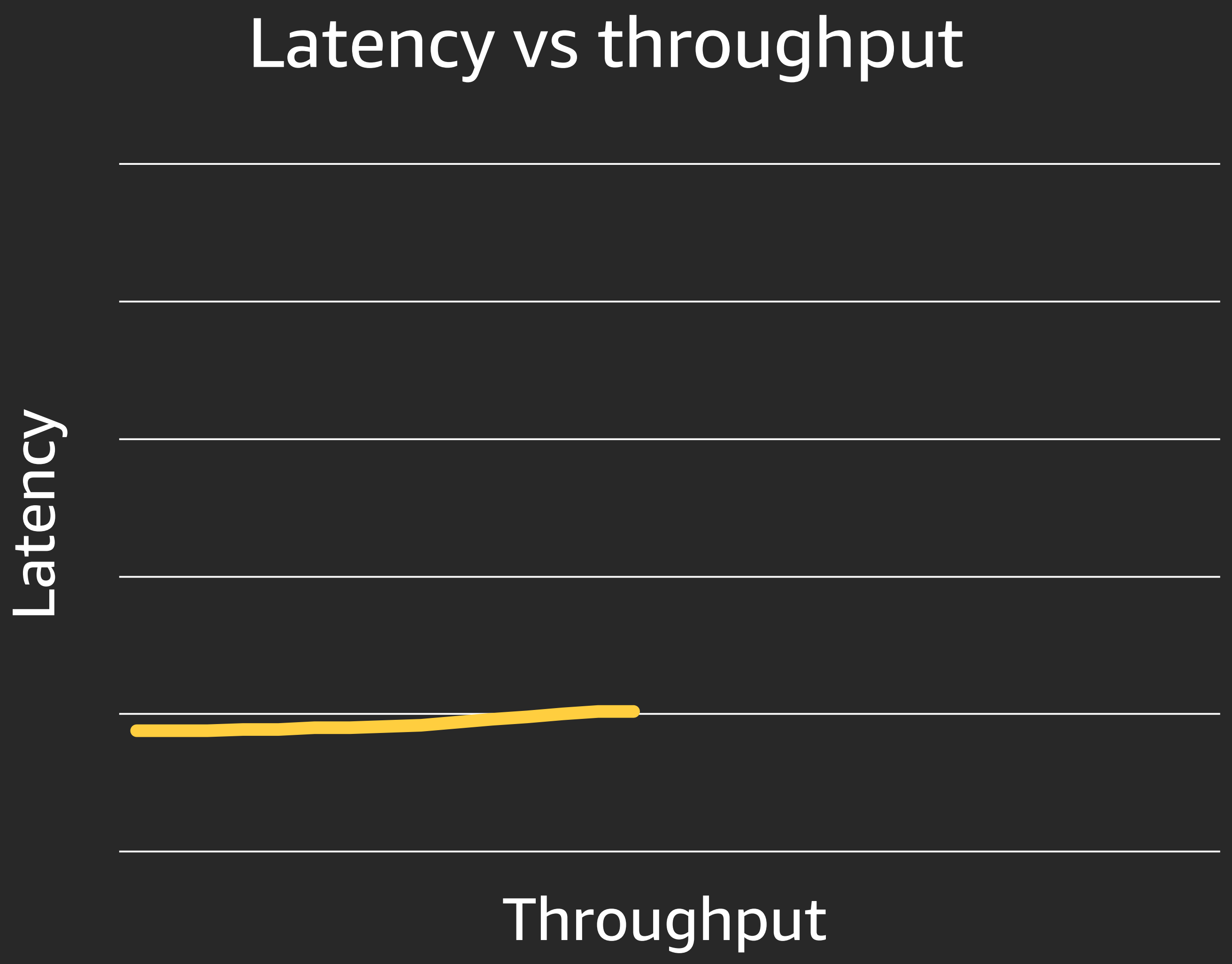
Don’t take on too much work

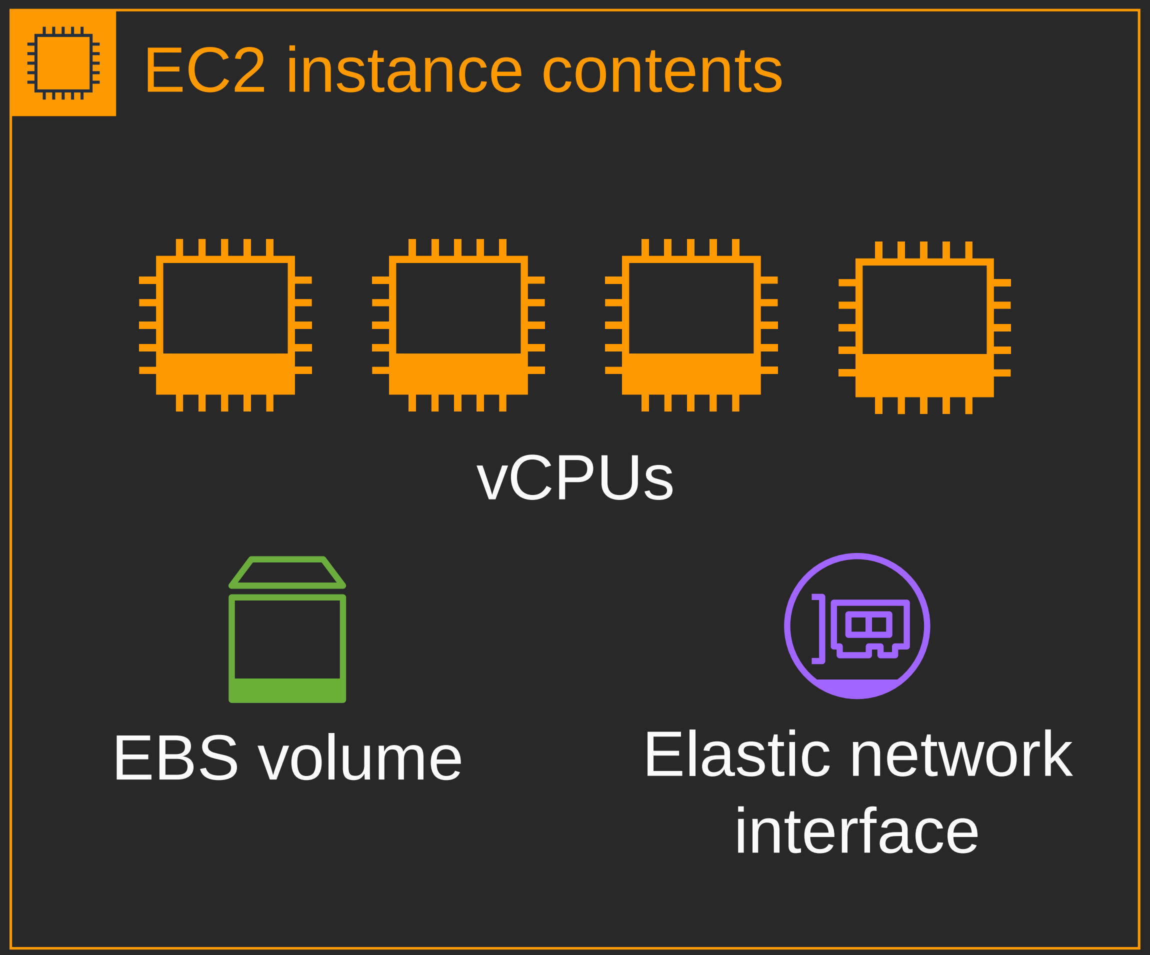
Degrading performance with load





Degrading performance with load



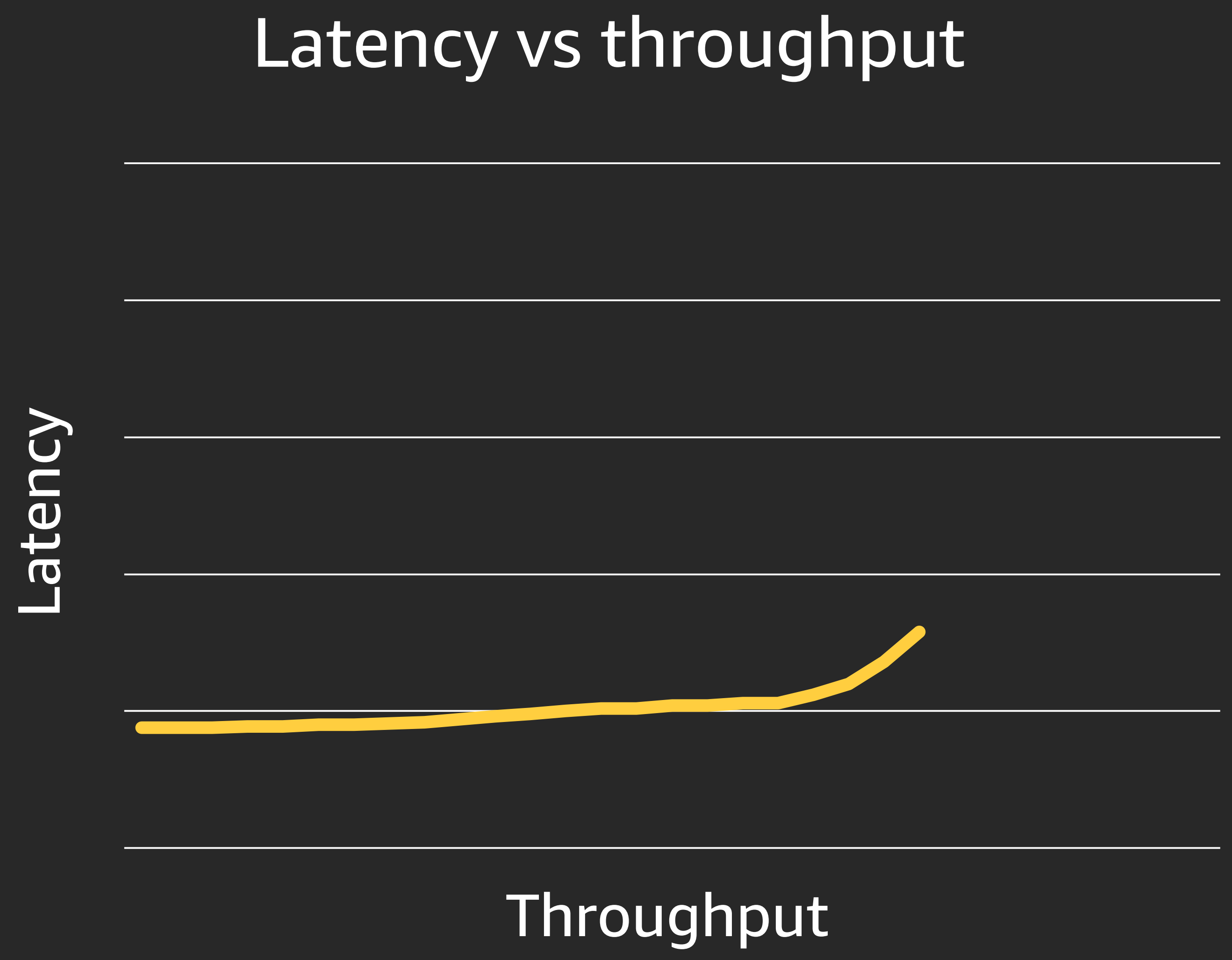

EC2 instance contents

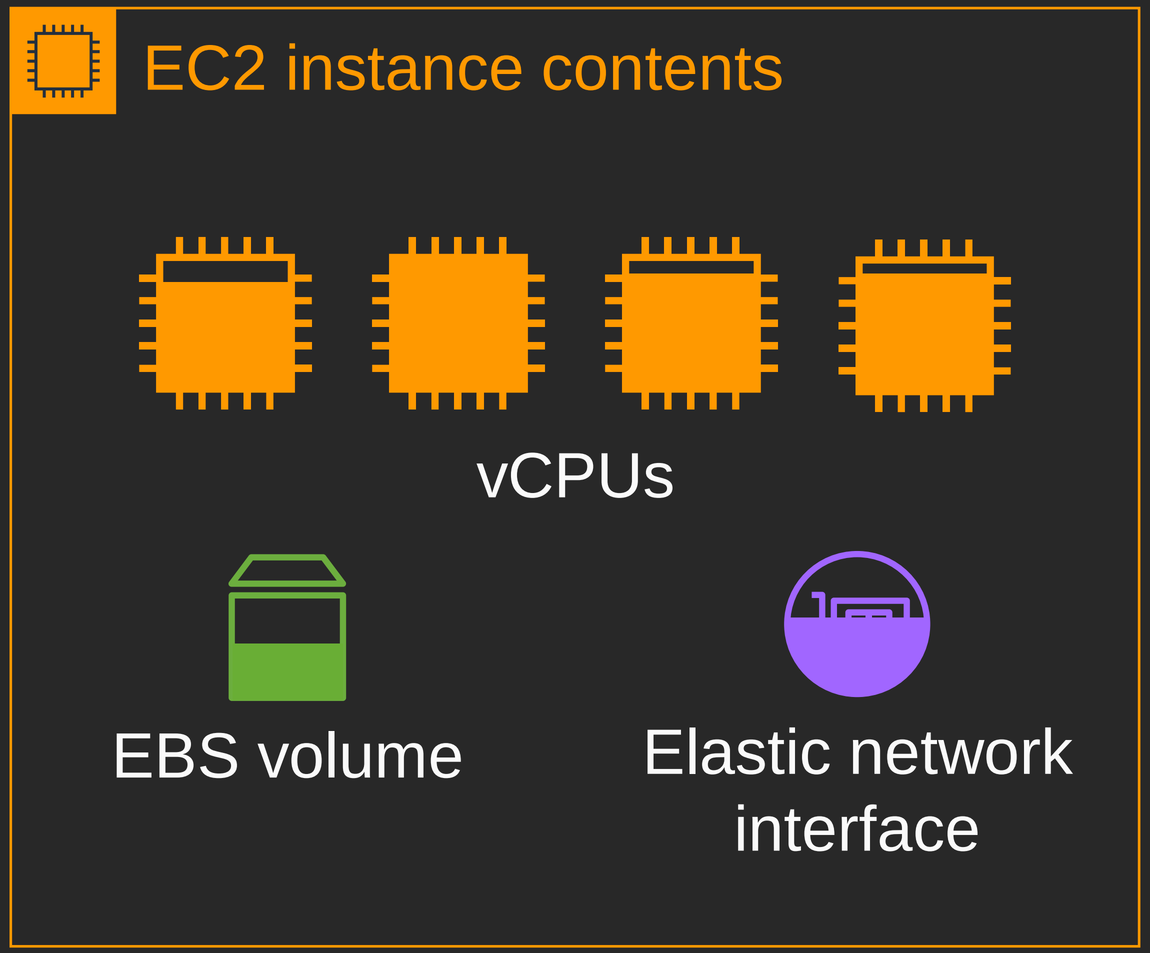
vCPUs

EBS volume

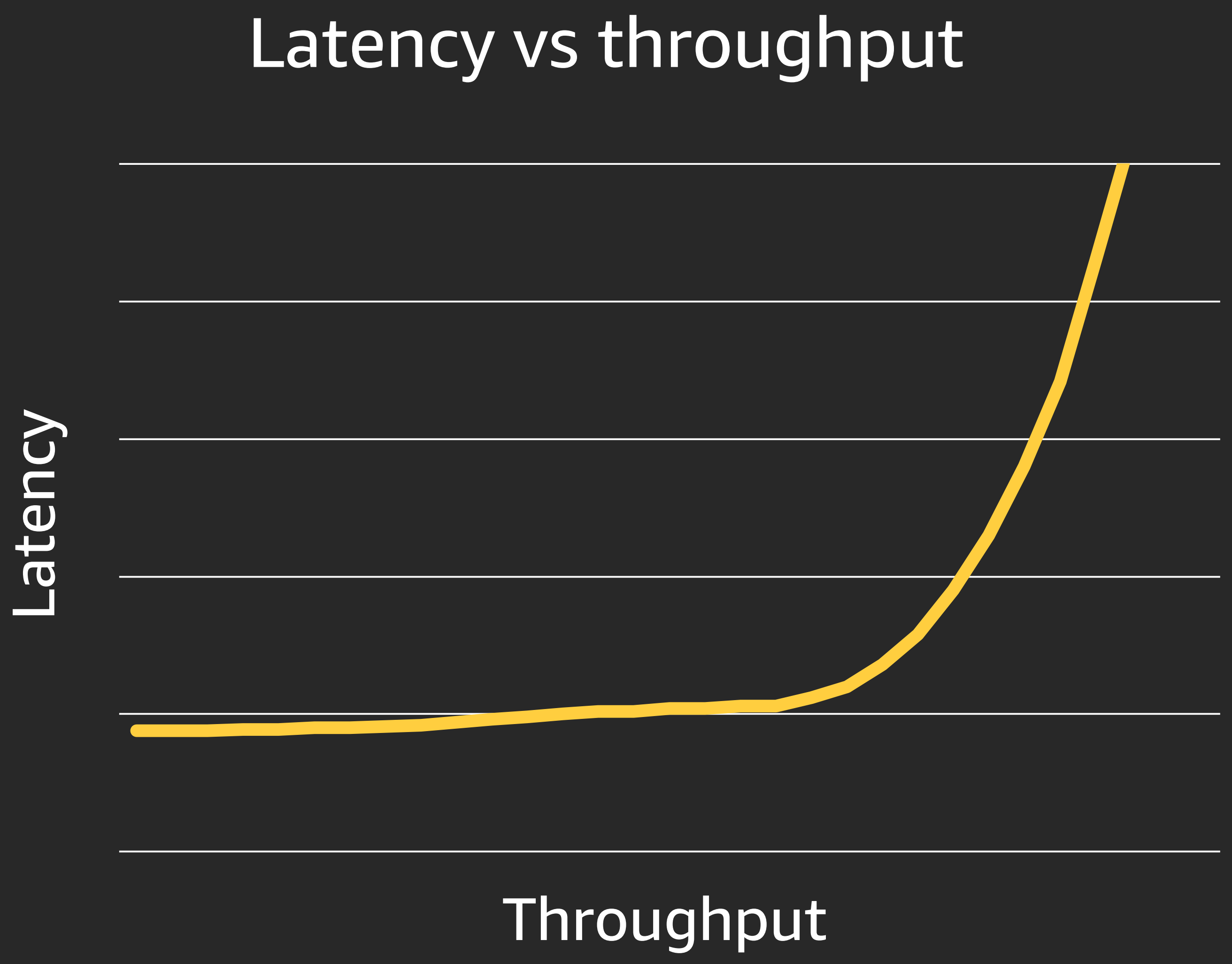
Elastic network interface

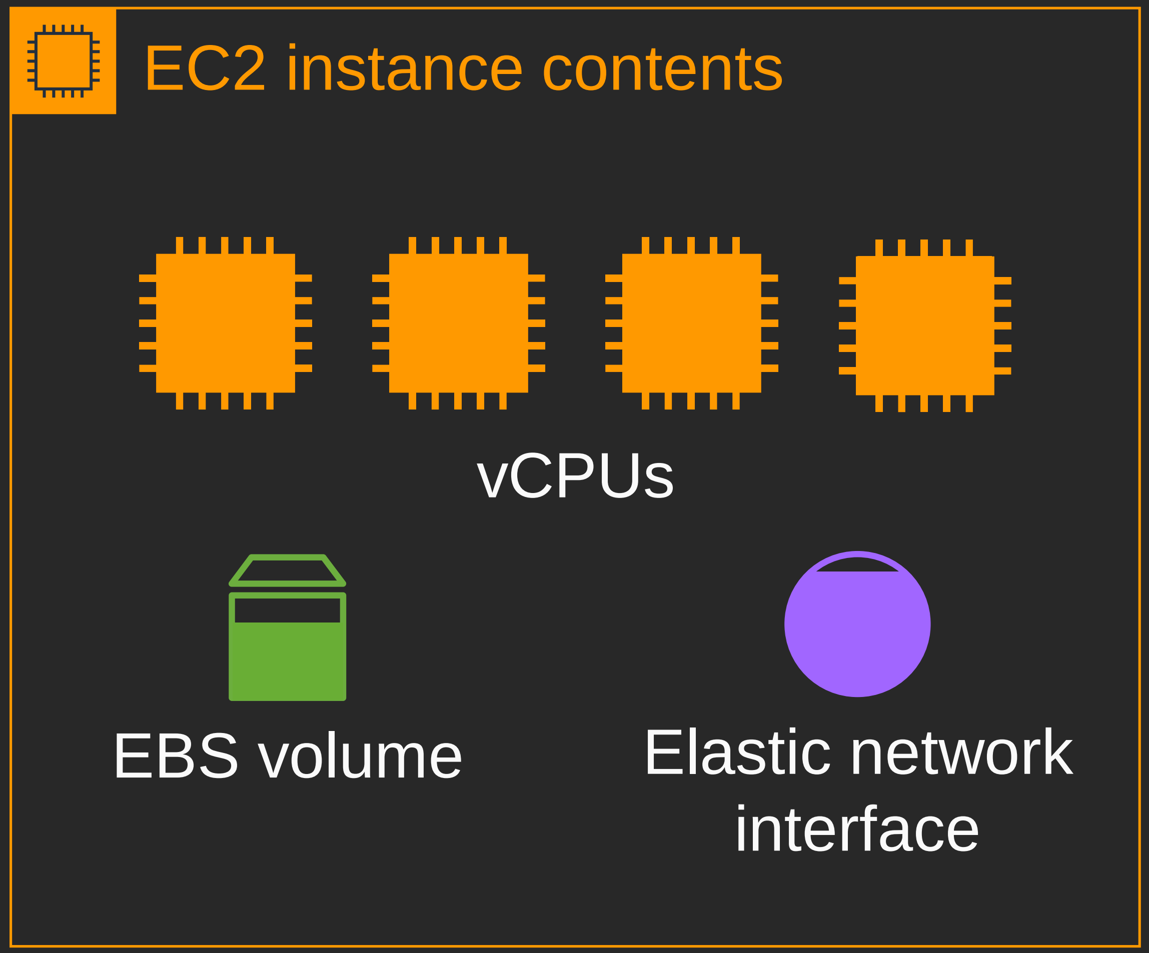
Degrading performance with load



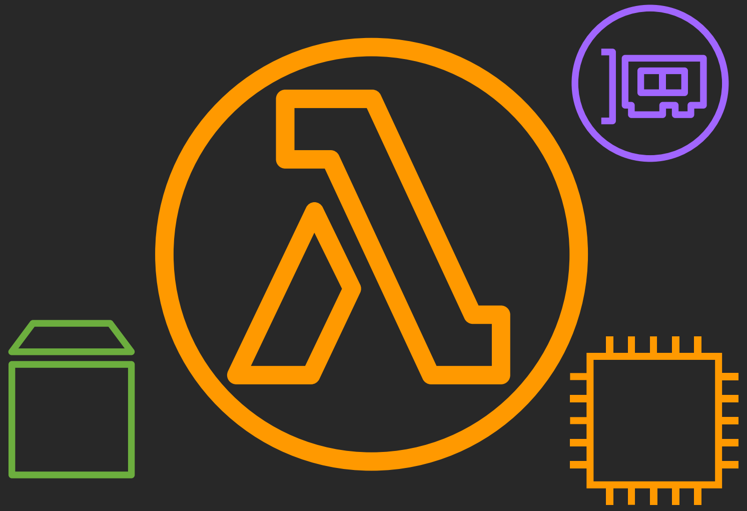


Degrading performance with load



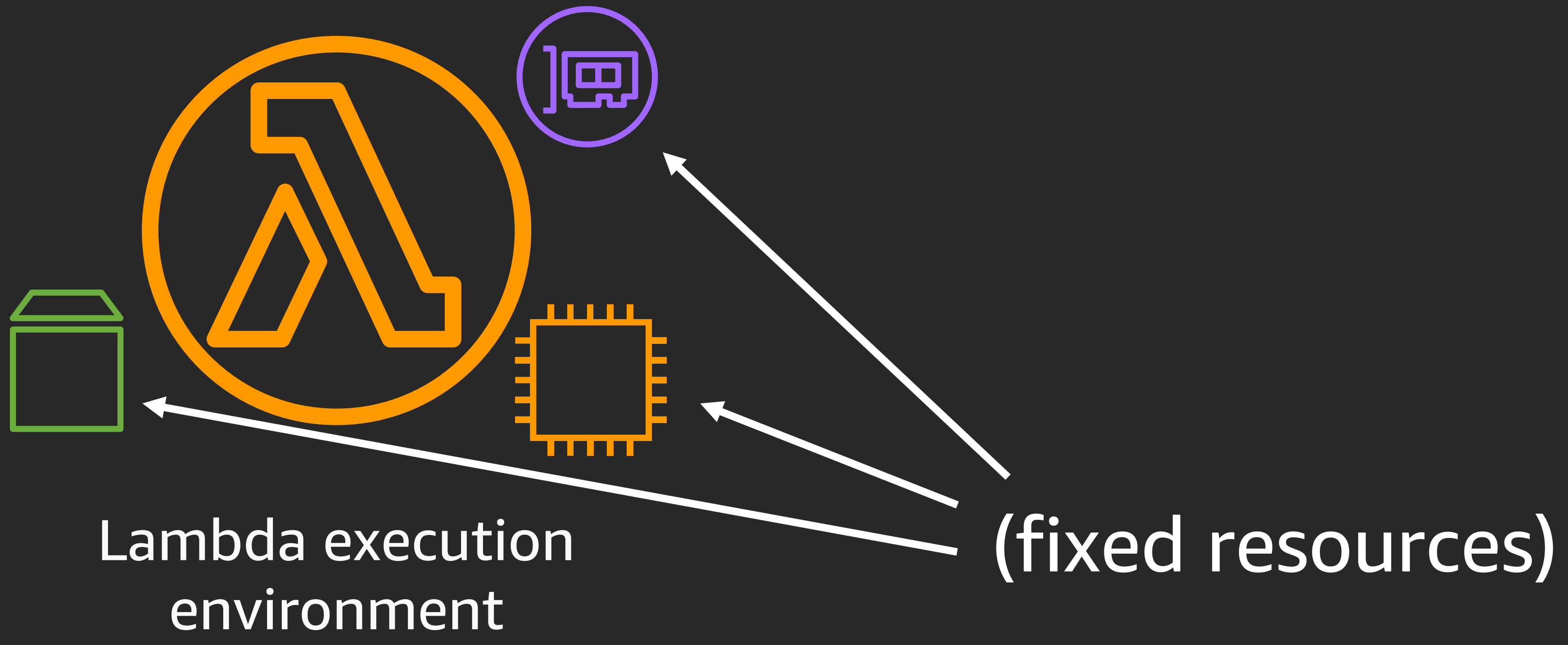


Fixed resources per unit of work

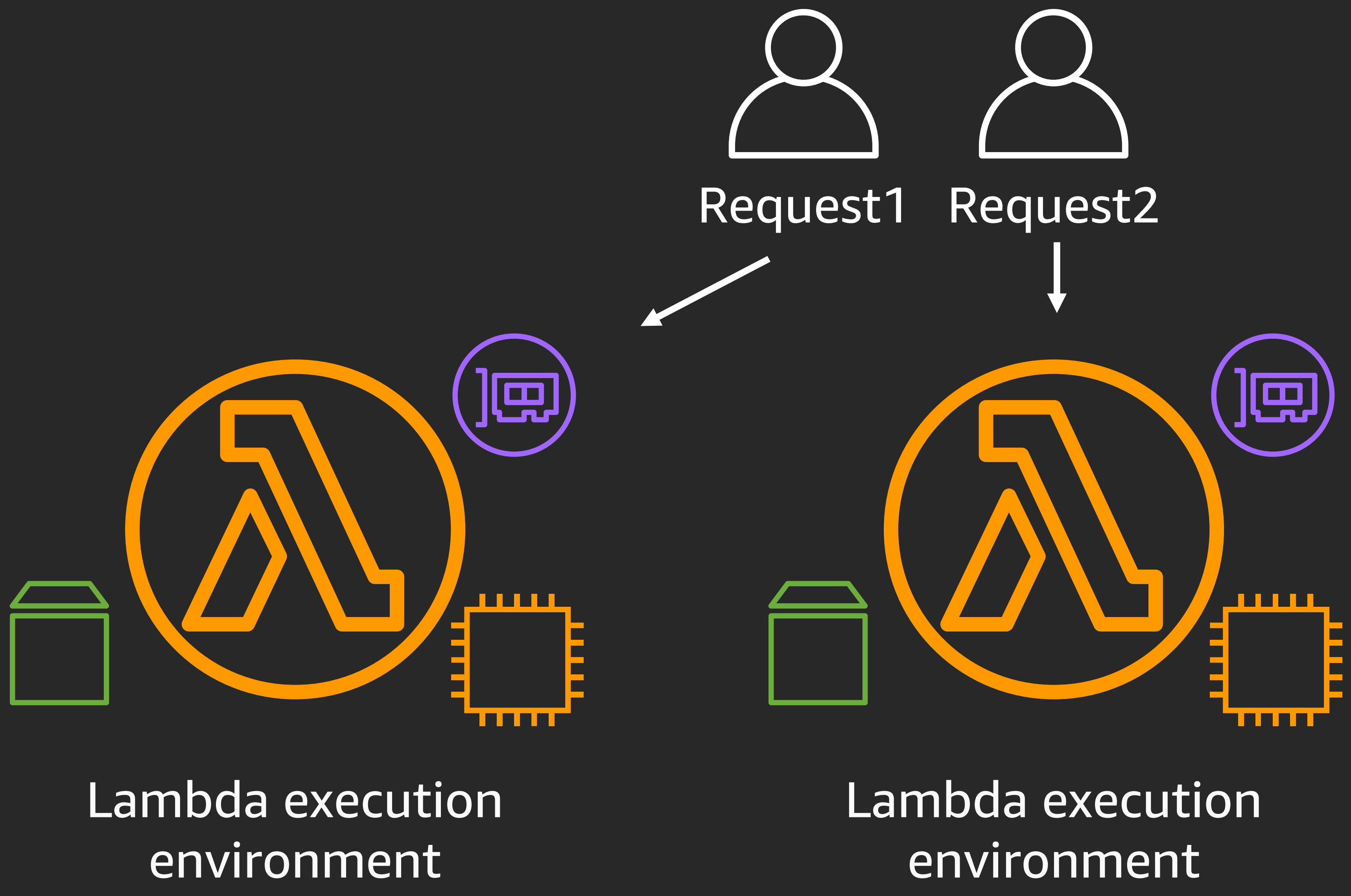


Lambda execution environment

Fixed resources per unit of work

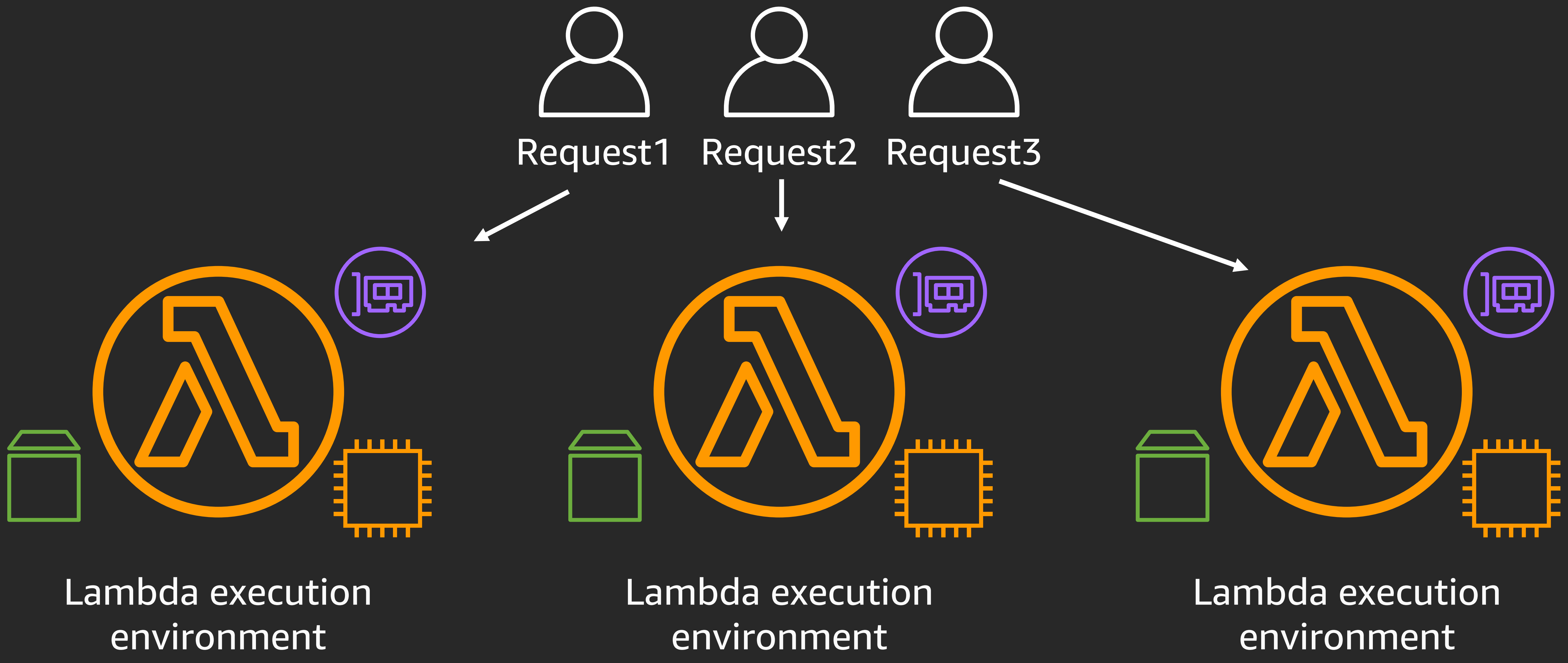


Fixed resources per unit of work



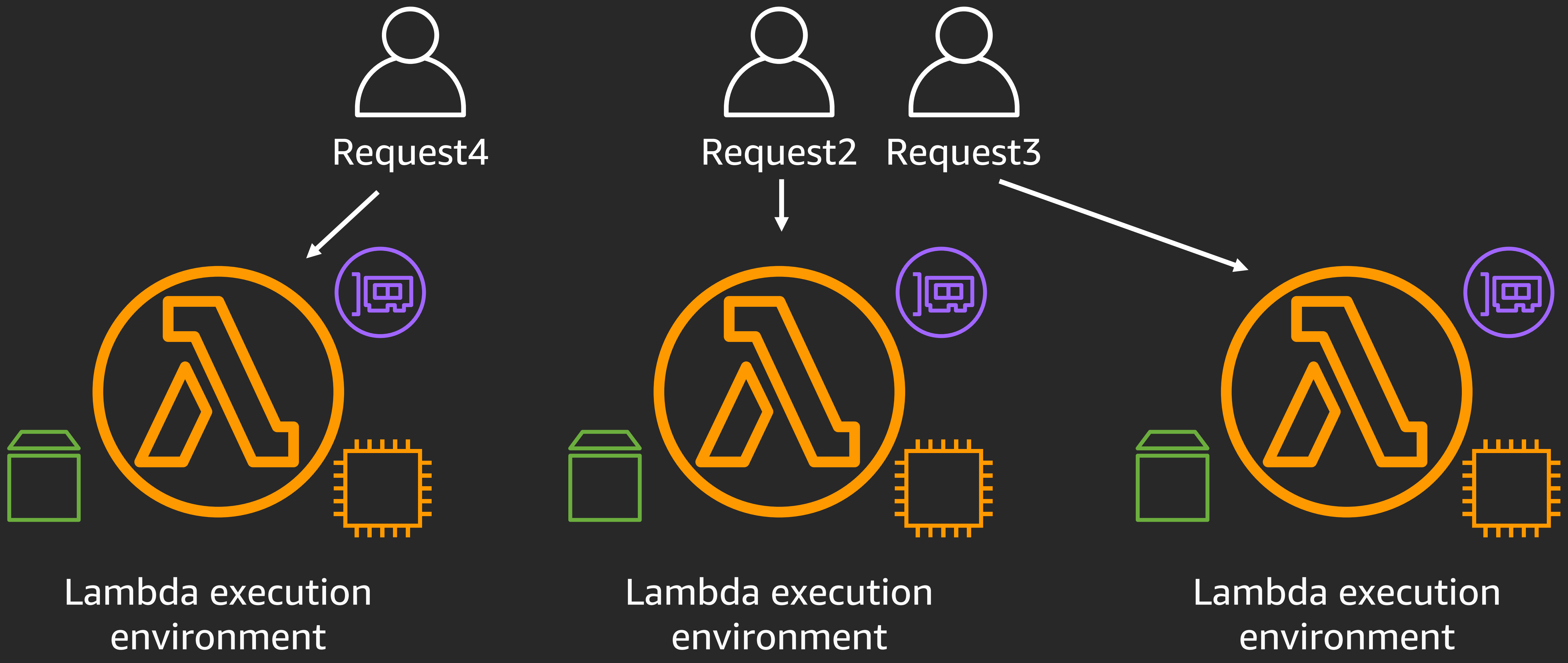
(only work on one thing at a time)

Fixed resources per unit of work


Request1
Request2
Request3
Lambda execution environment
Lambda execution environment
Lambda execution environment

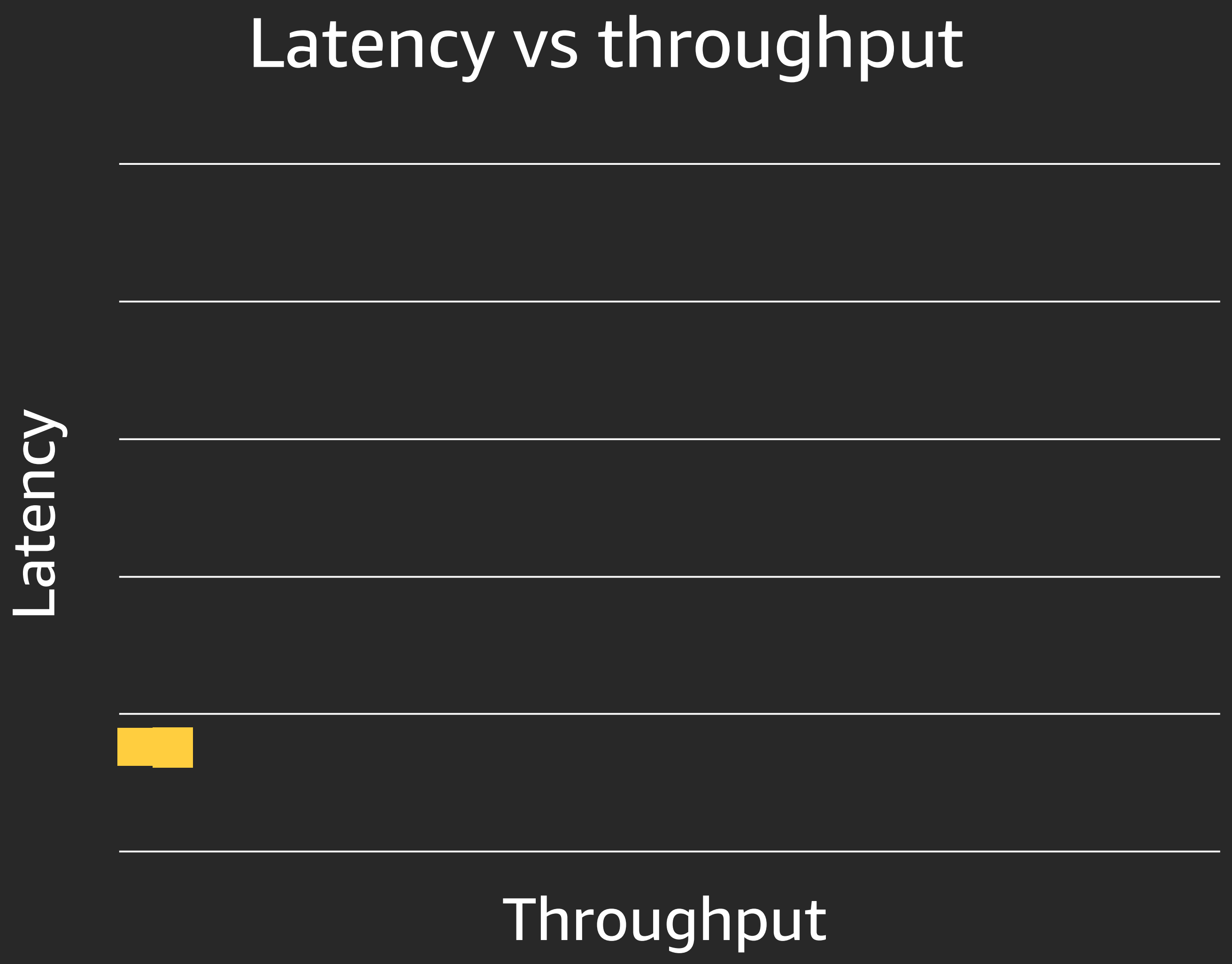
(only work on one thing at a time)

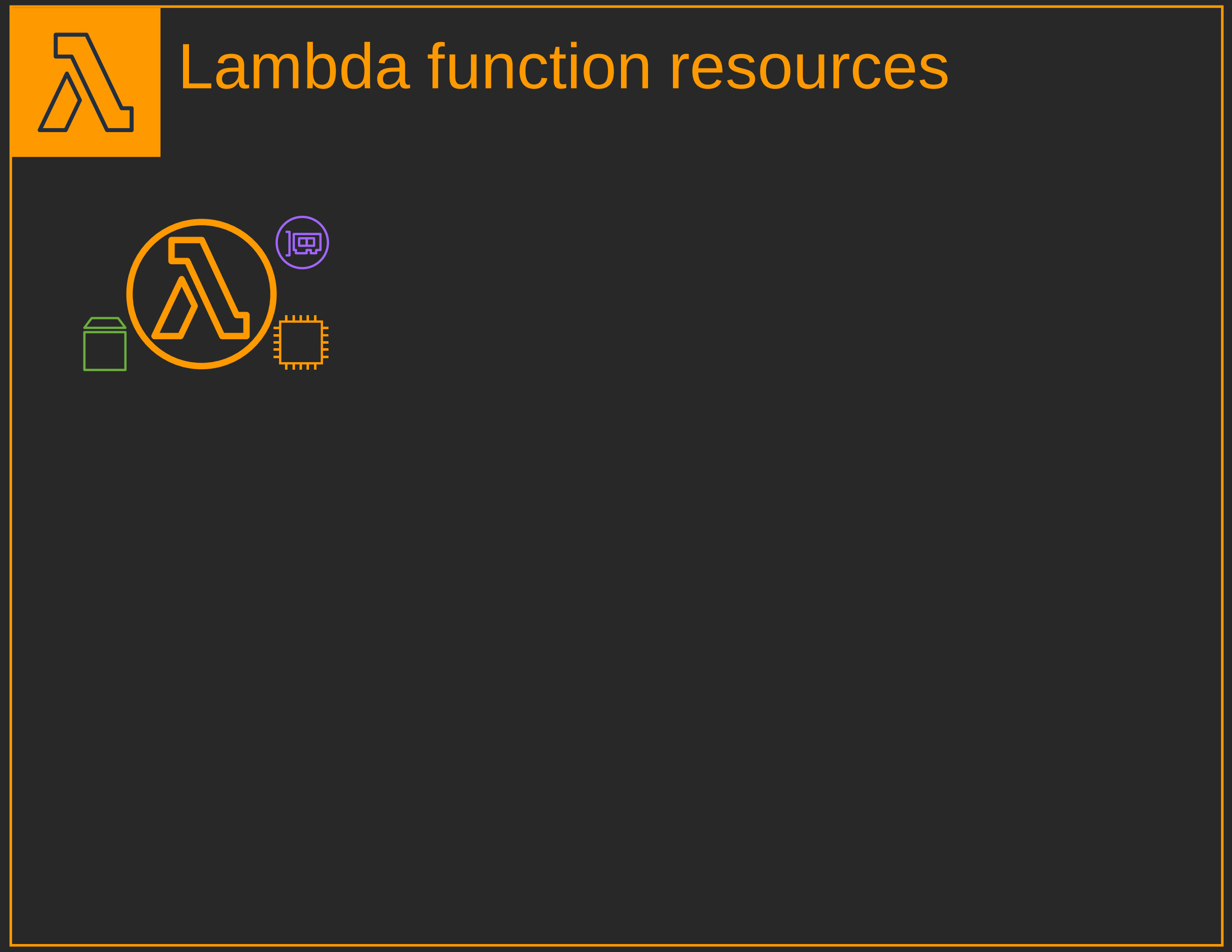
Fixed resources per unit of work


Request4
Request2
Request3
Lambda execution environment
Lambda execution environment
Lambda execution environment

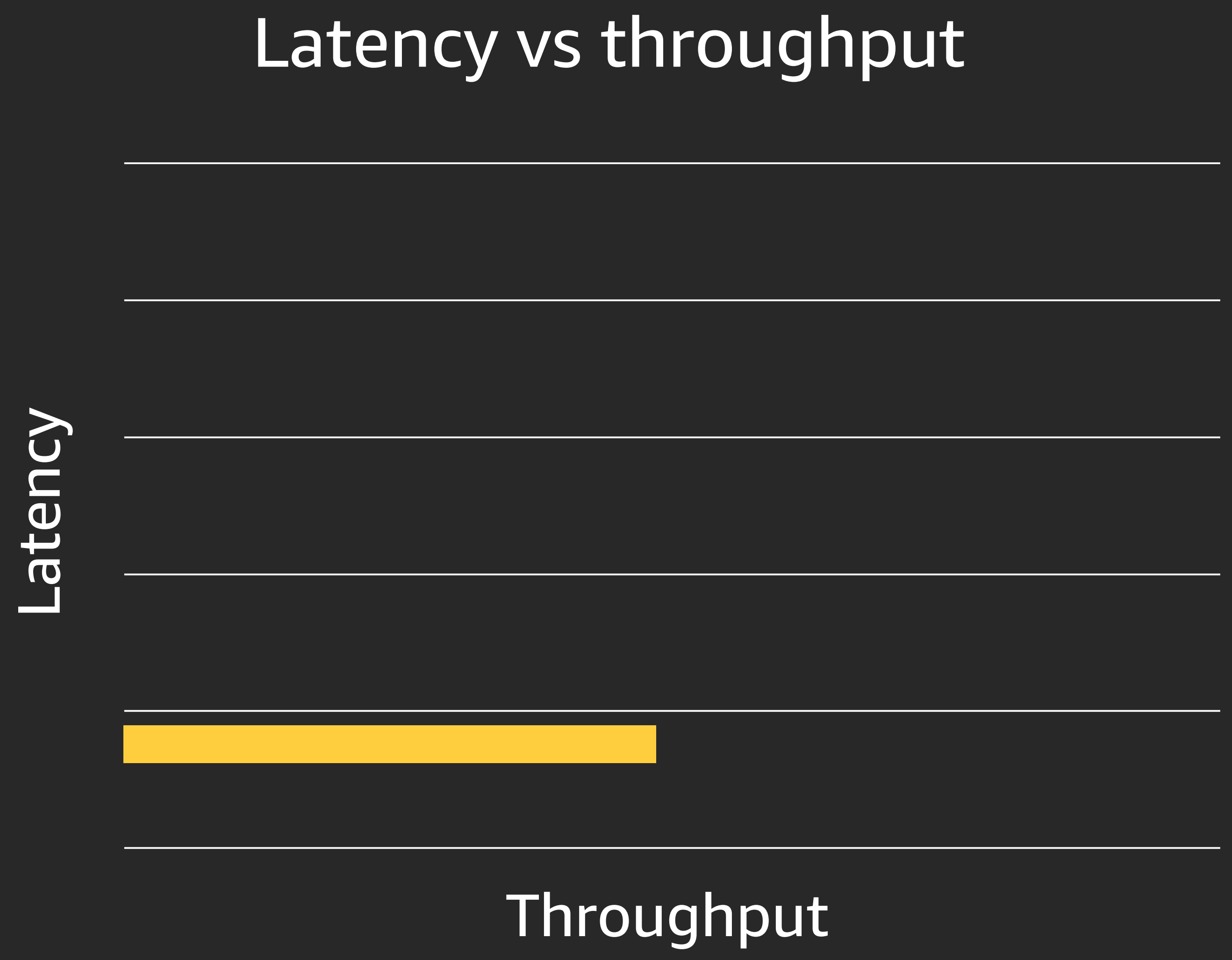
(only work on one thing at a time)

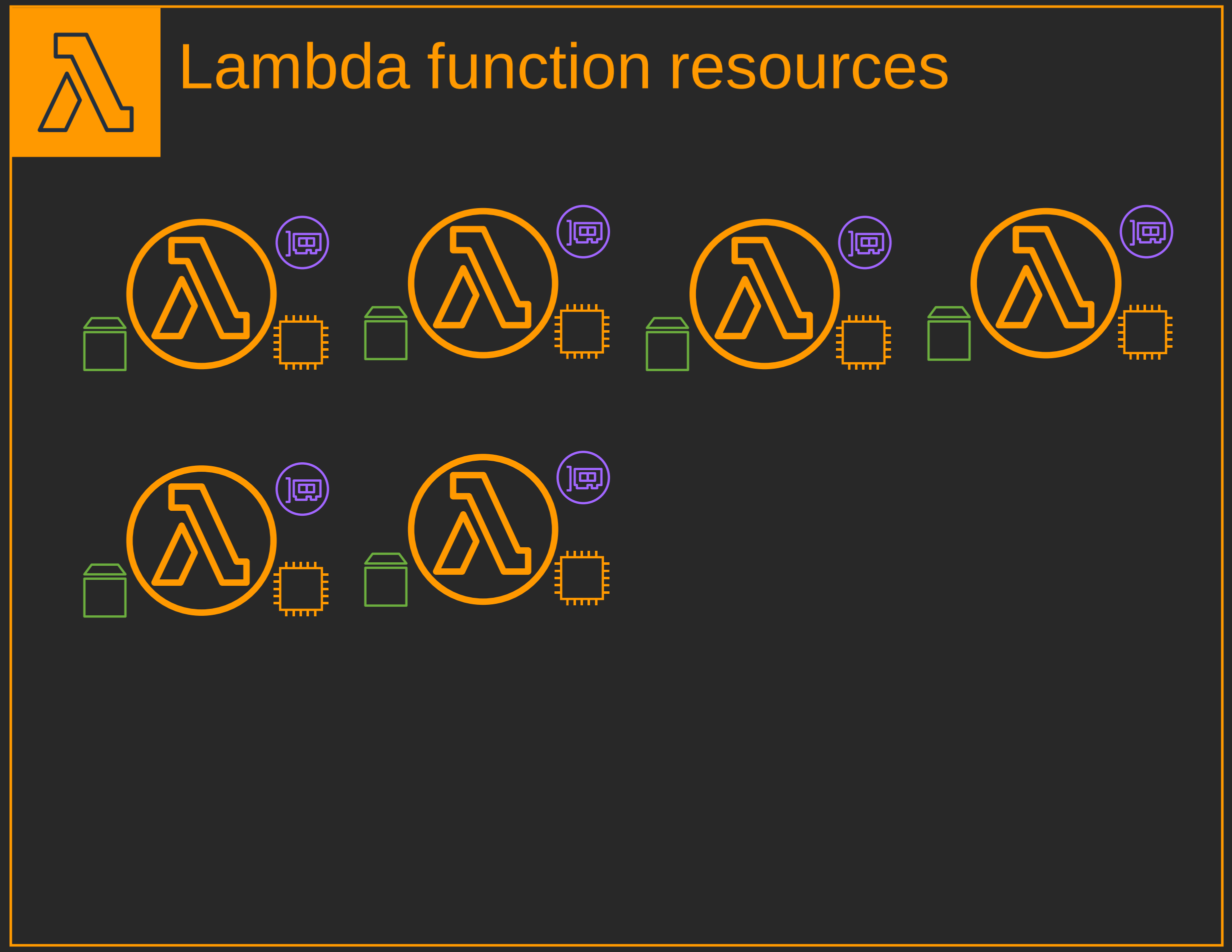
Workload isolation means predictable performance



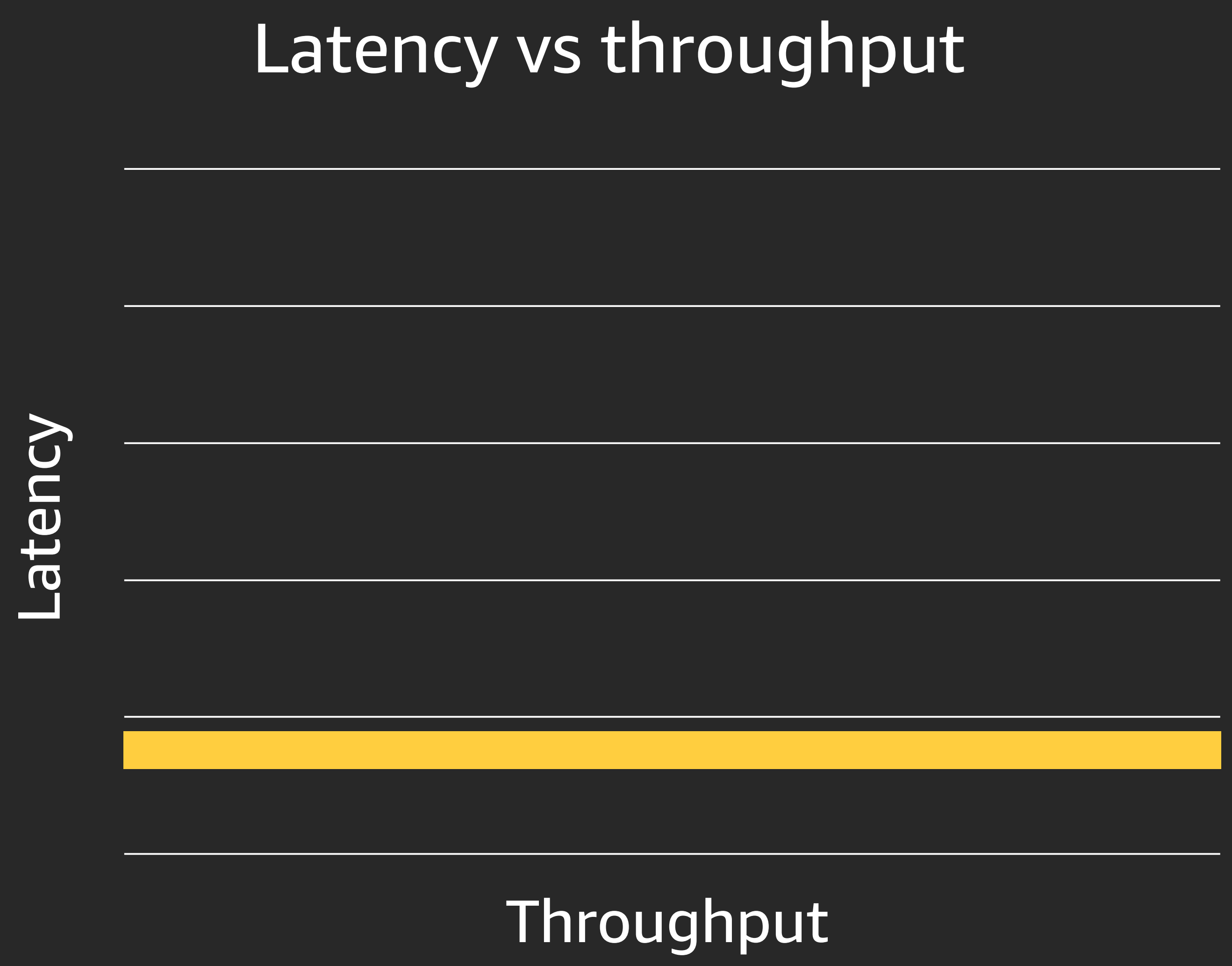


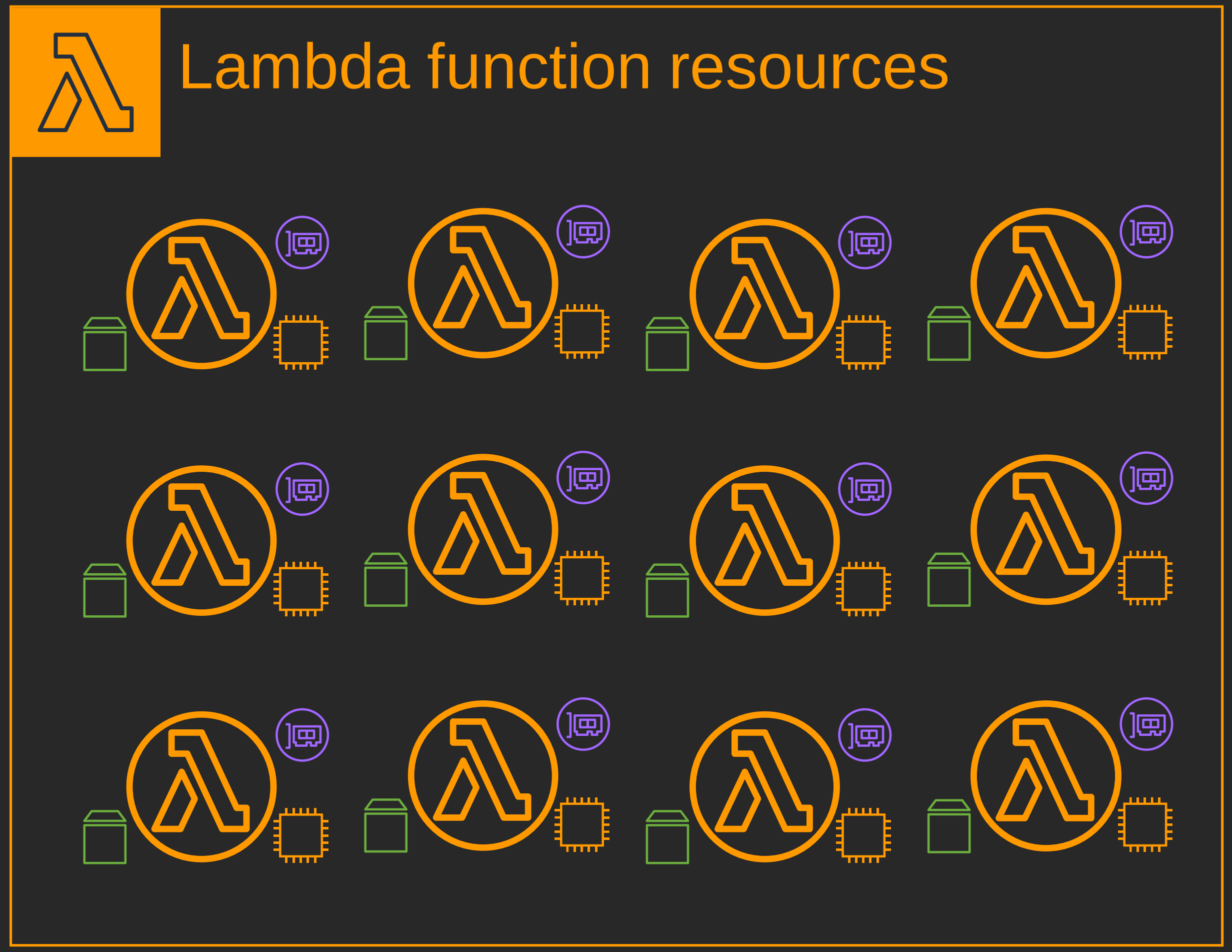
Workload isolation means predictable performance





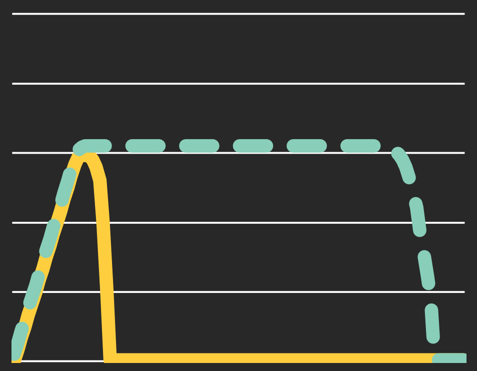
Workload isolation means predictable performance





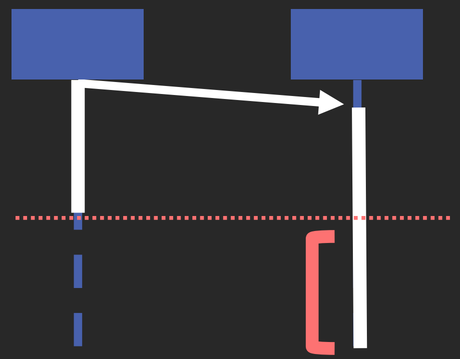
Don’t take on too much work

Load shedding recap



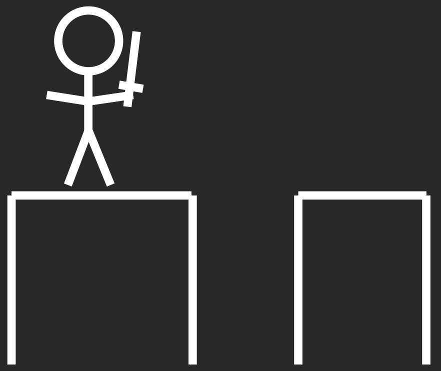
Reject excess load

Keep latency low for the work you take on



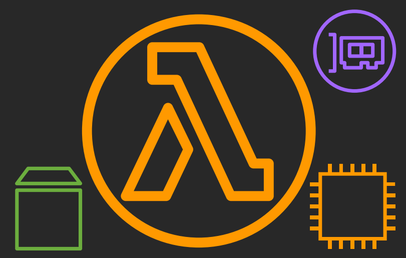
Don’t waste work

Wasting work reduces throughput



Do bounded work

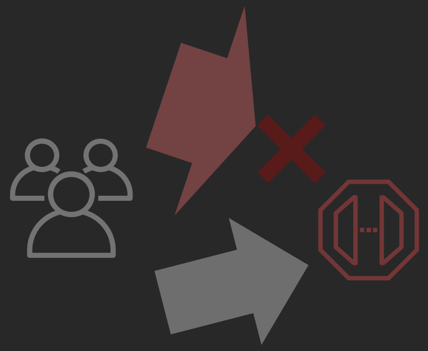
Make scaling more predictable and testable



Don’t take on extra work

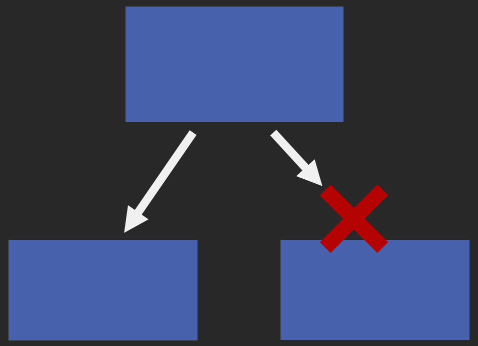
Give the same resources to each request

Chapter two



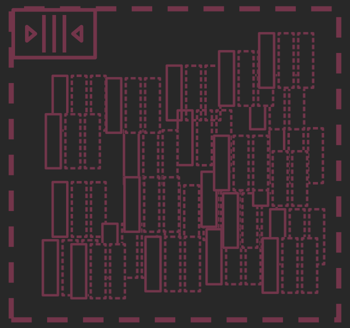
Load shedding

Avoid brownout by rejecting excess load



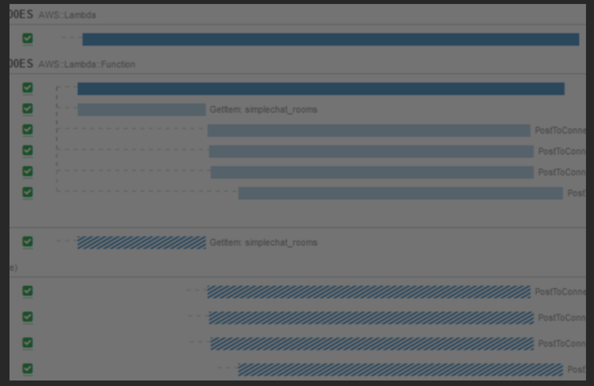
Dependency isolation

Prevent one dependency from affecting unrelated functionality



Avoiding queue backlogs

Prevent a backlog from extending recovery time



Operating

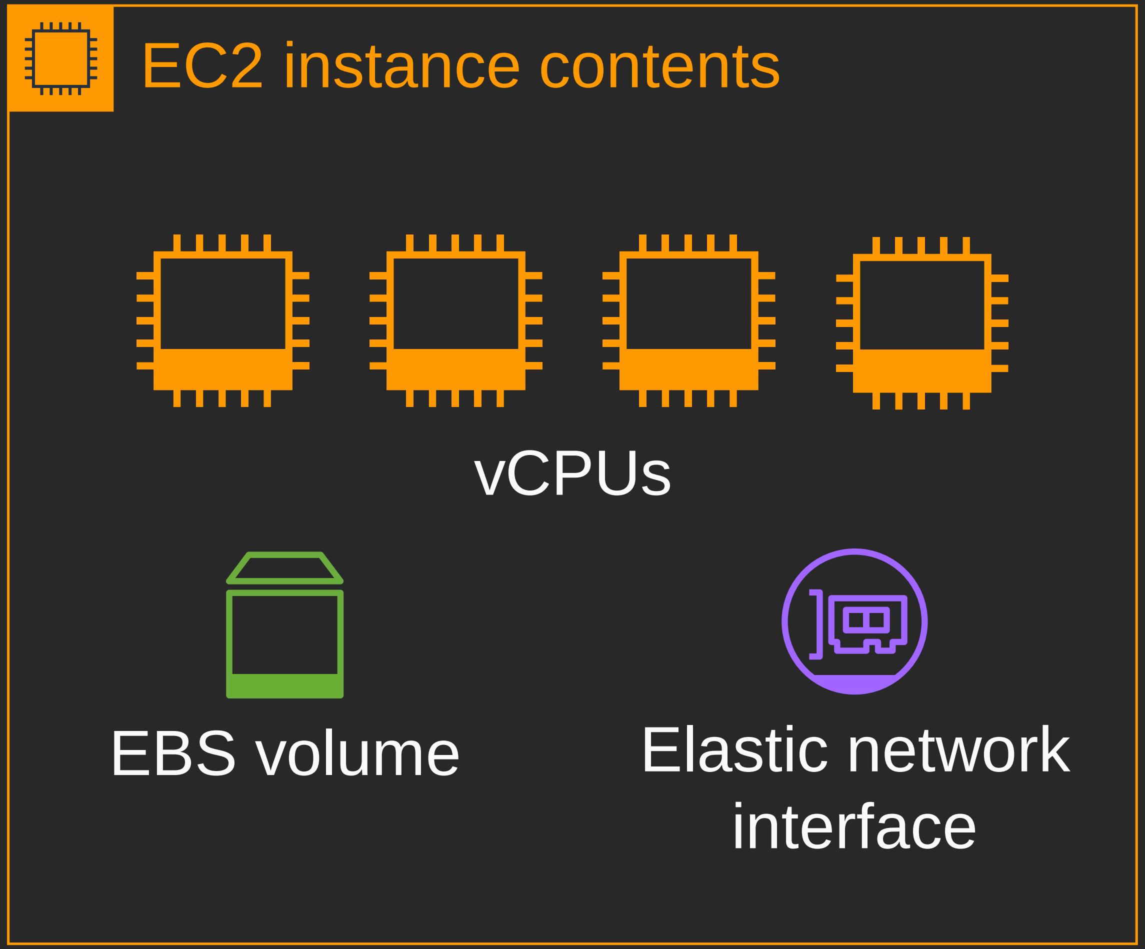
Quickly diagnose and mitigate issues

Running out of concurrency

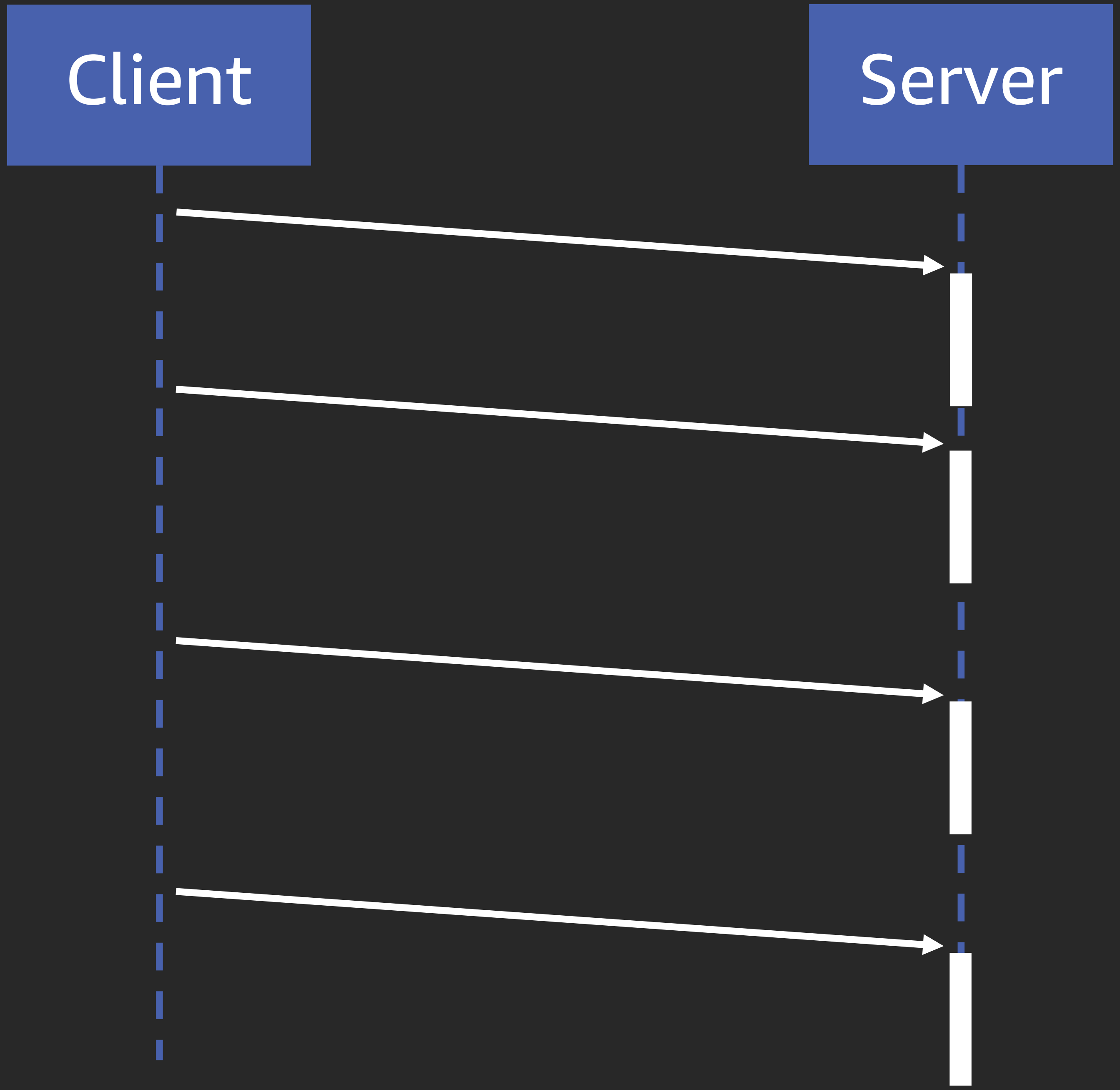
Application bottlenecks

Not pictured:

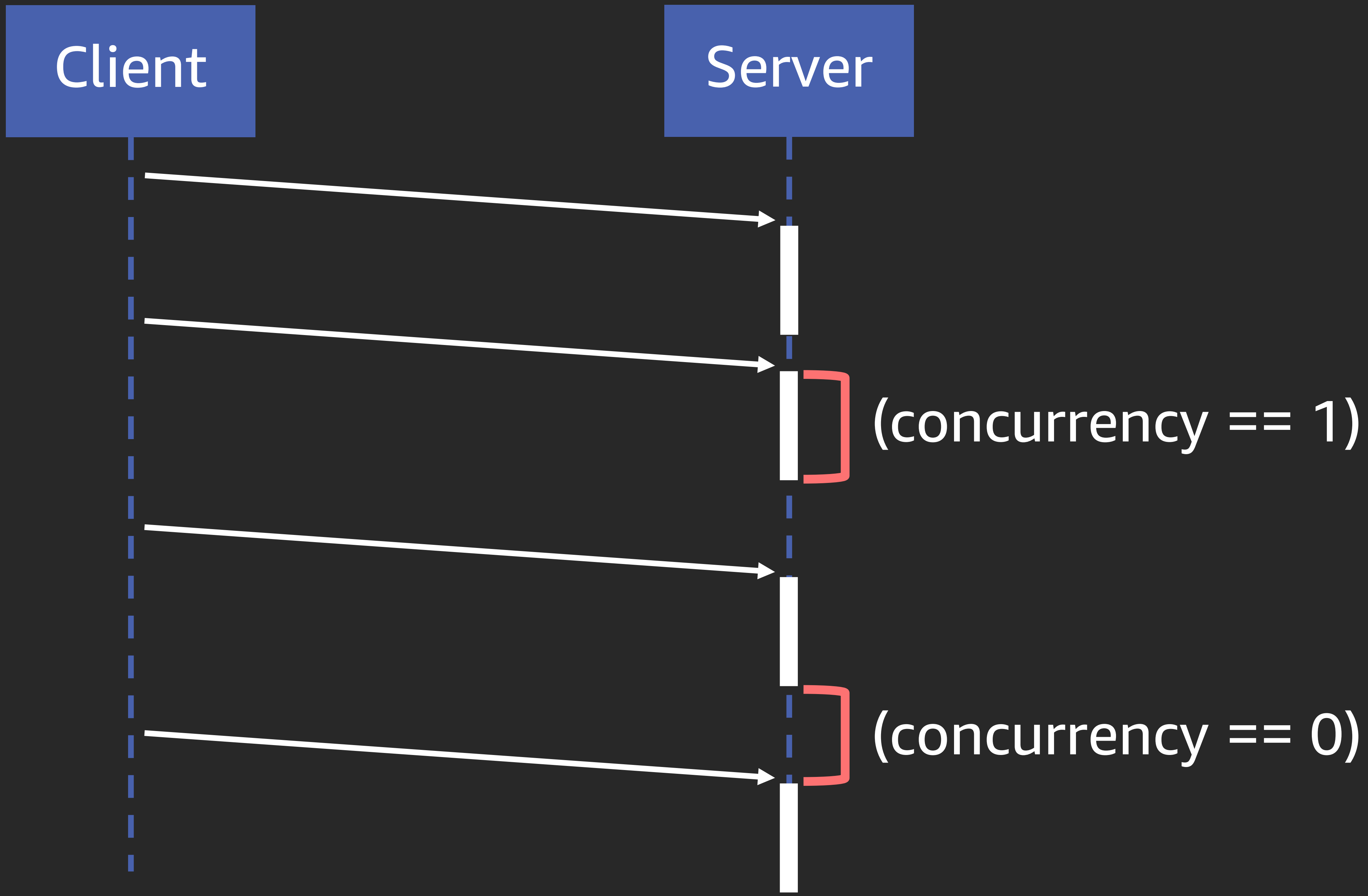
- Application thread pools
- File descriptors
- Ephemeral ports



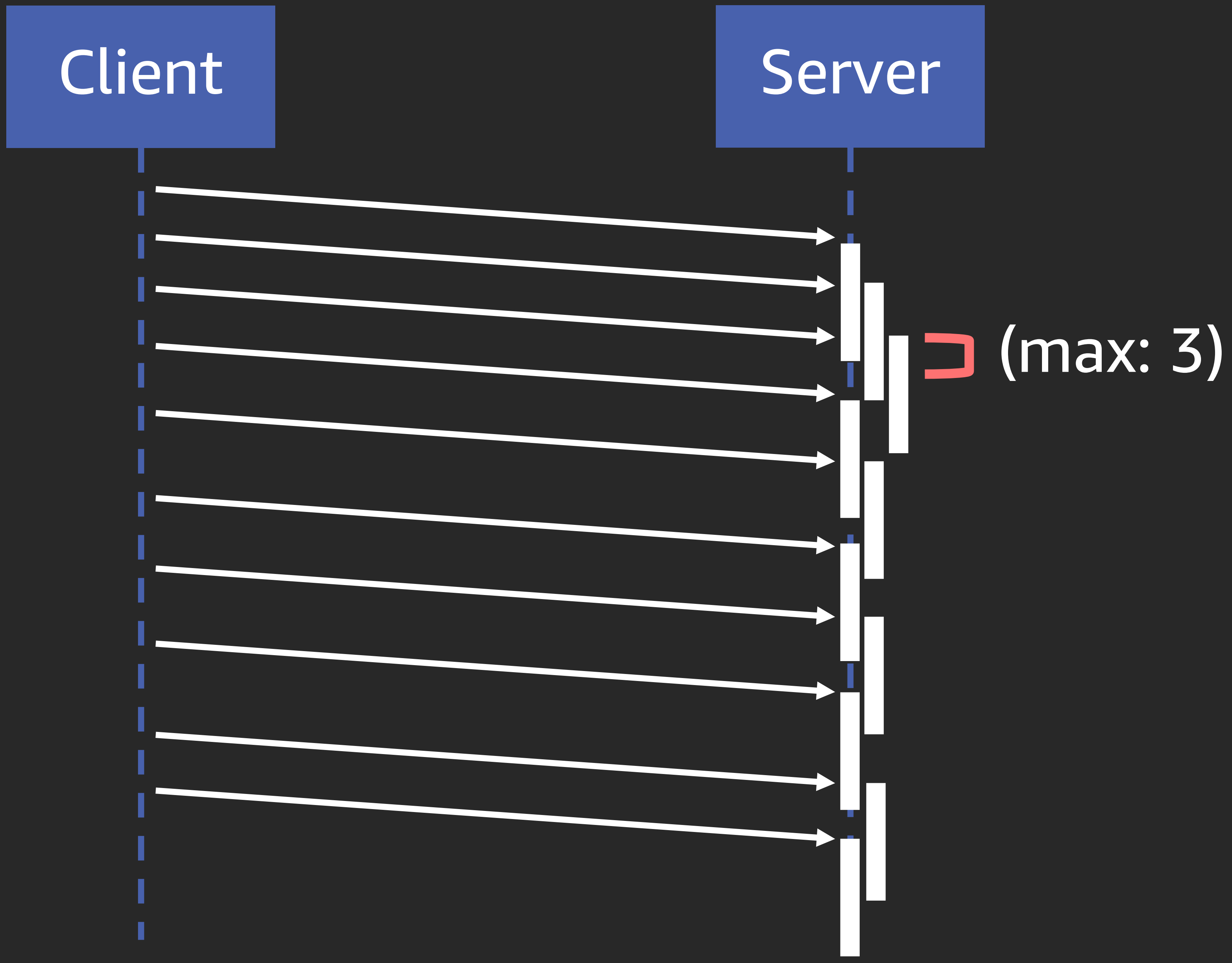
Concurrency, visualized



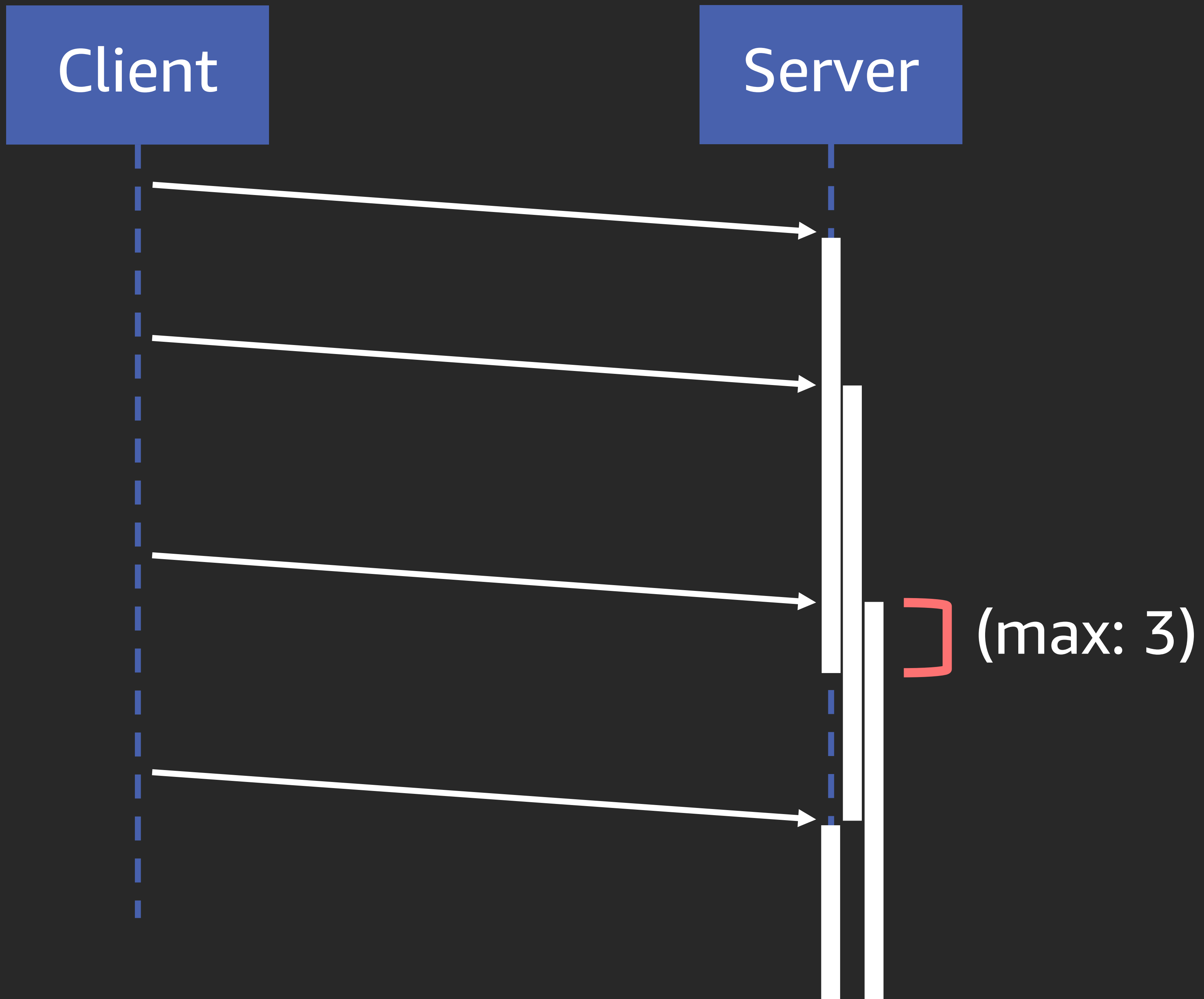
Concurrency with low request rate, low latency



Concurrency increases with arrival rate

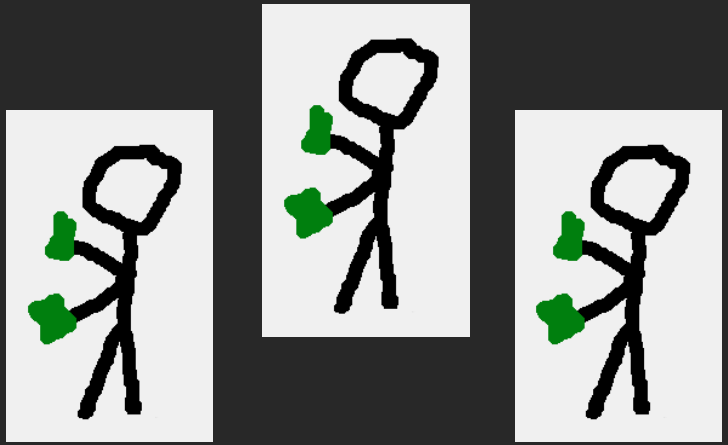


Concurrency increases with latency

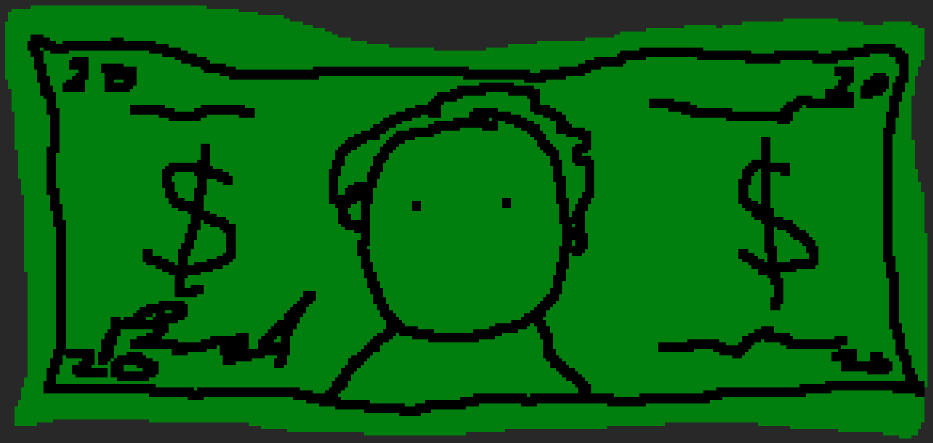


Little’s Law

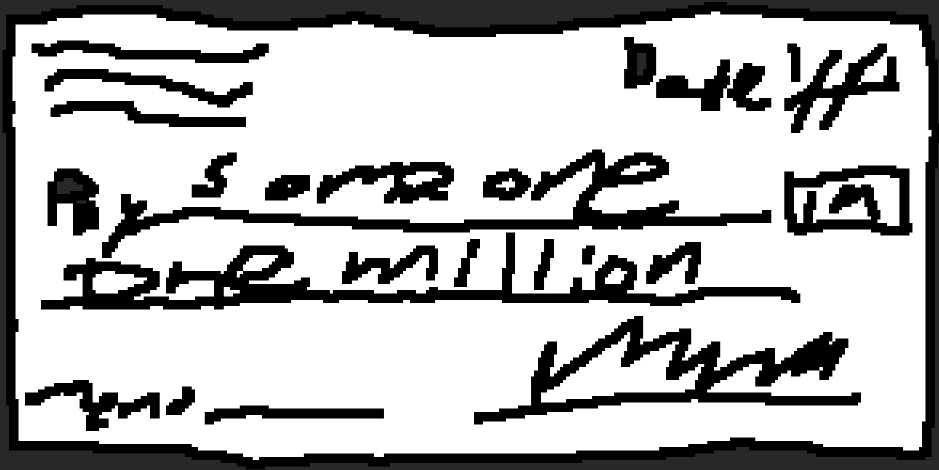
$$concurrency = arrival\,rate \times latency$$



$$\mathit{concurrency} = \mathit{arrival\ rate} \times \mathit{latency}$$

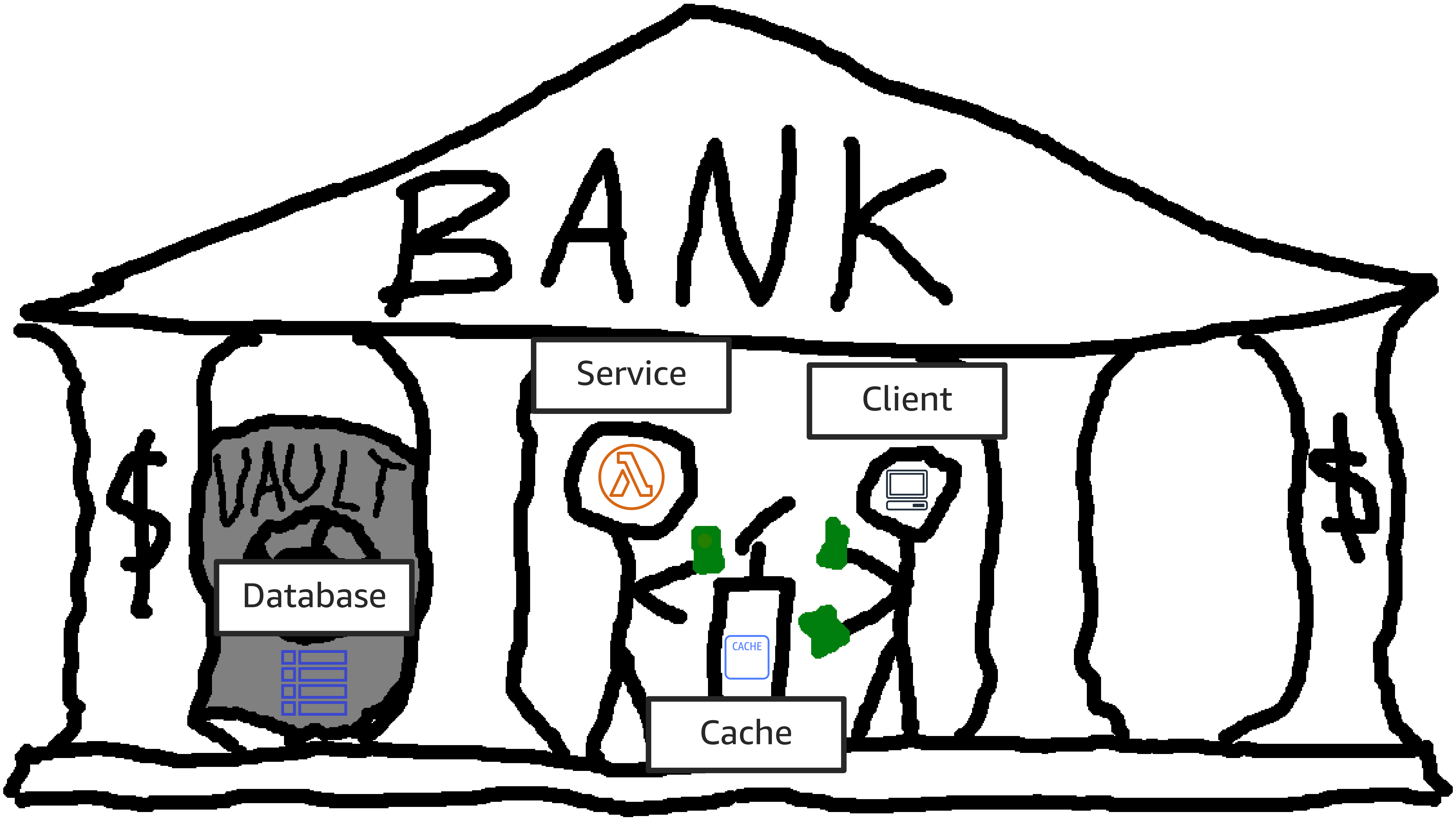


Cash



Check

BANK
$
VAULT
$


BANK
VAULT
Database
Service
Client
CACHE
Cache
$
$

BANK
VAULT
$
$
$

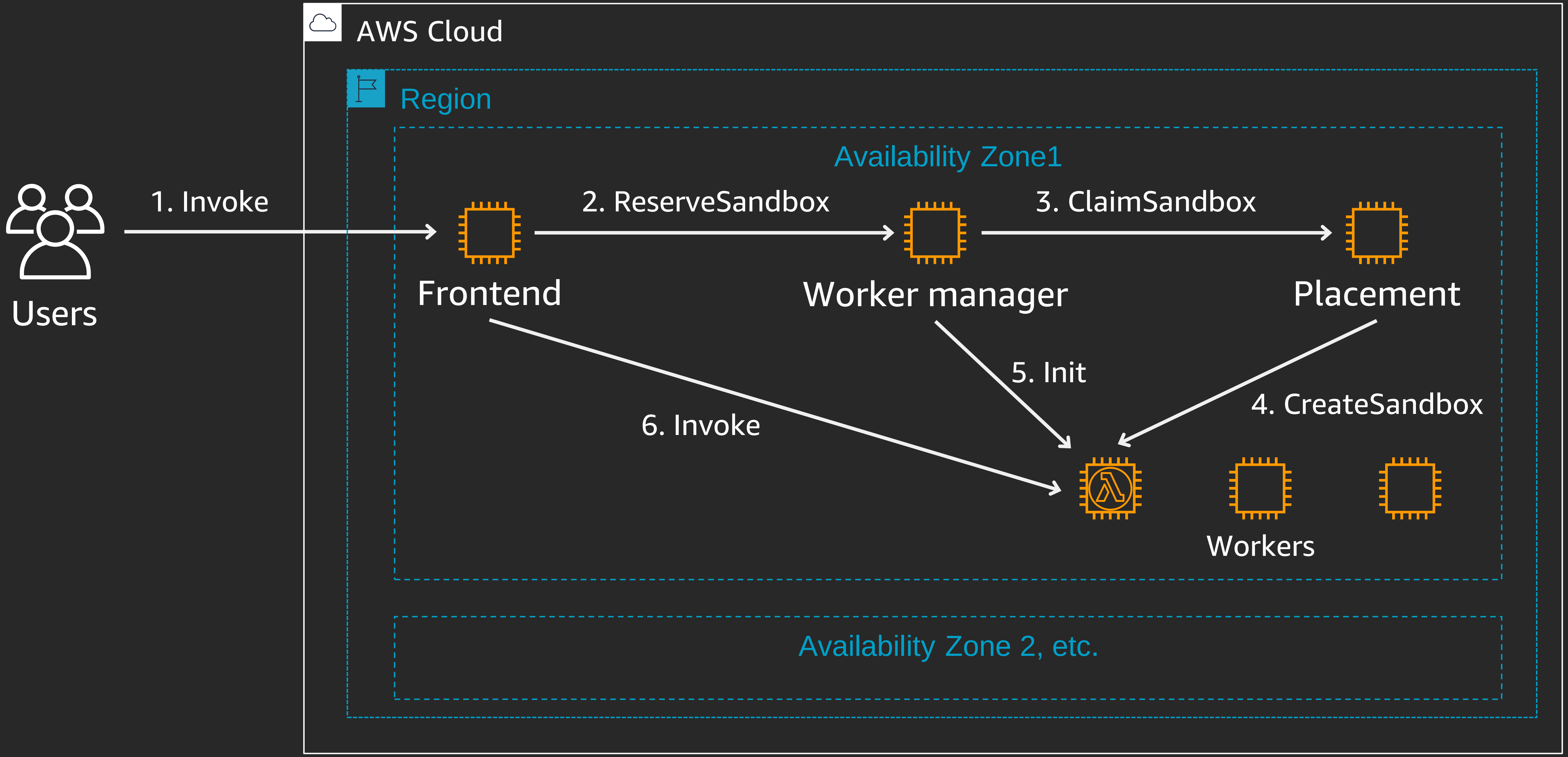
BANK
VAULT
$
$
$

Dependency isolation within AWS Lambda

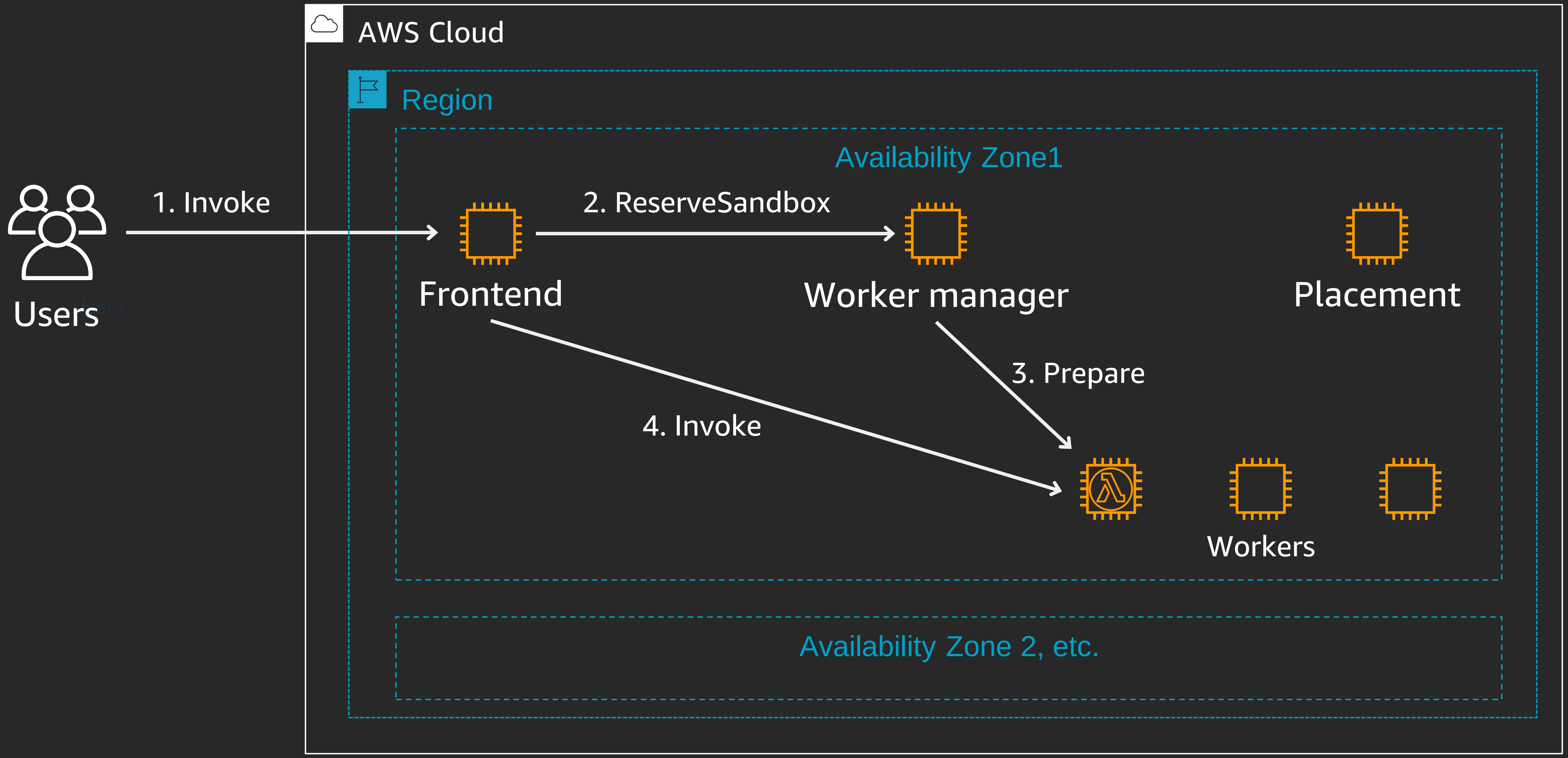
Lambda architecture*

*currently

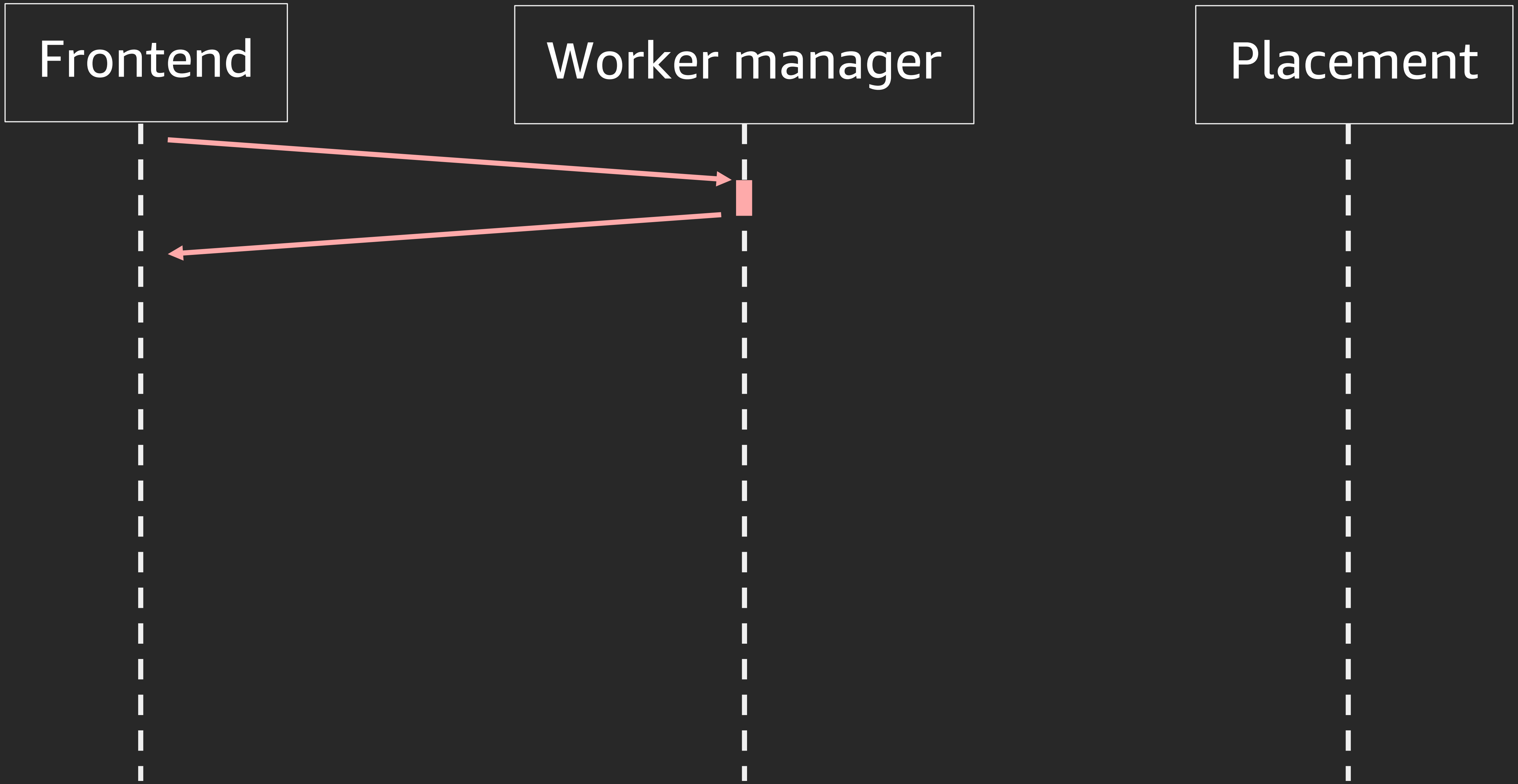
Lambda architecture: Cold invoke



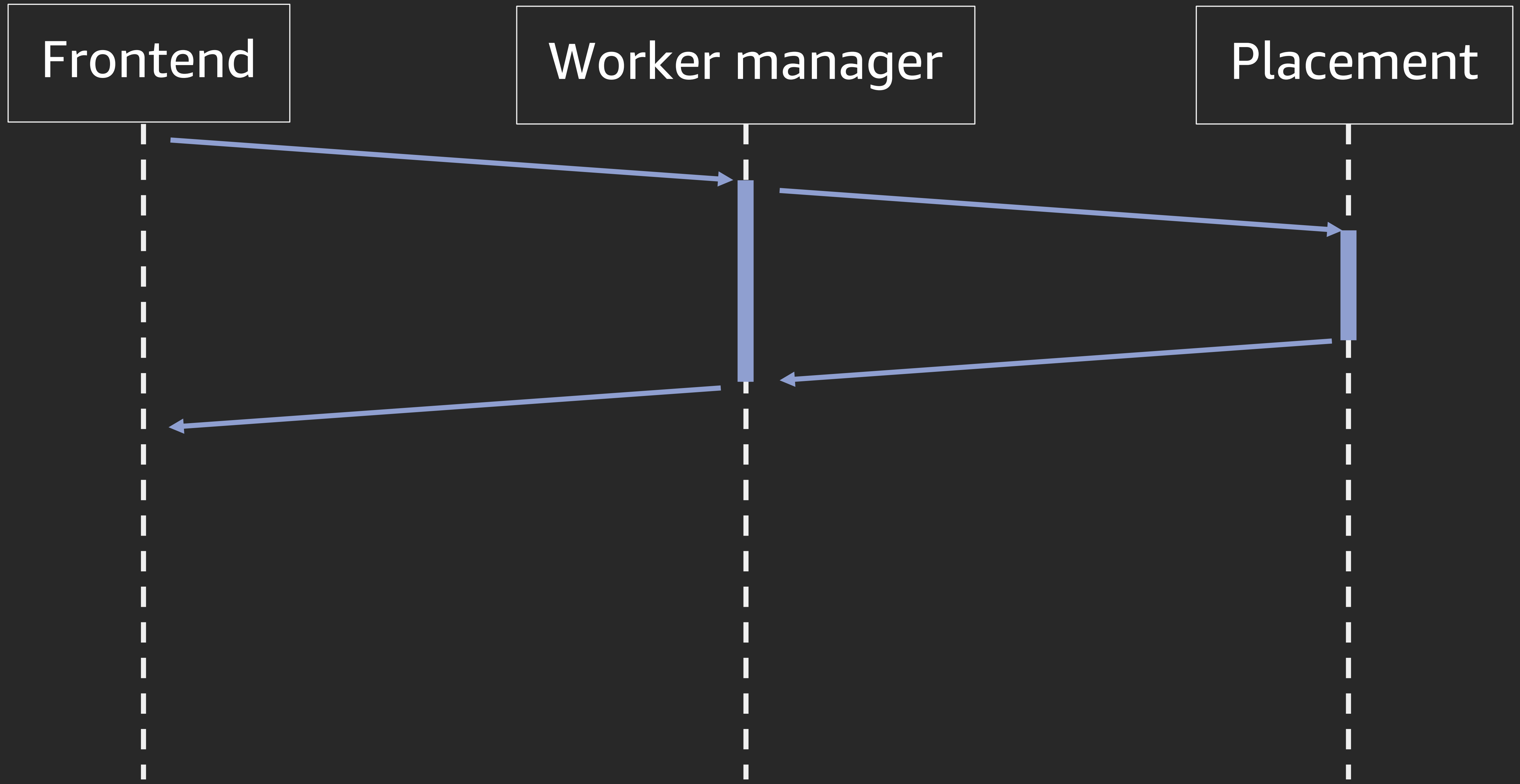
Lambda architecture: Warm invoke



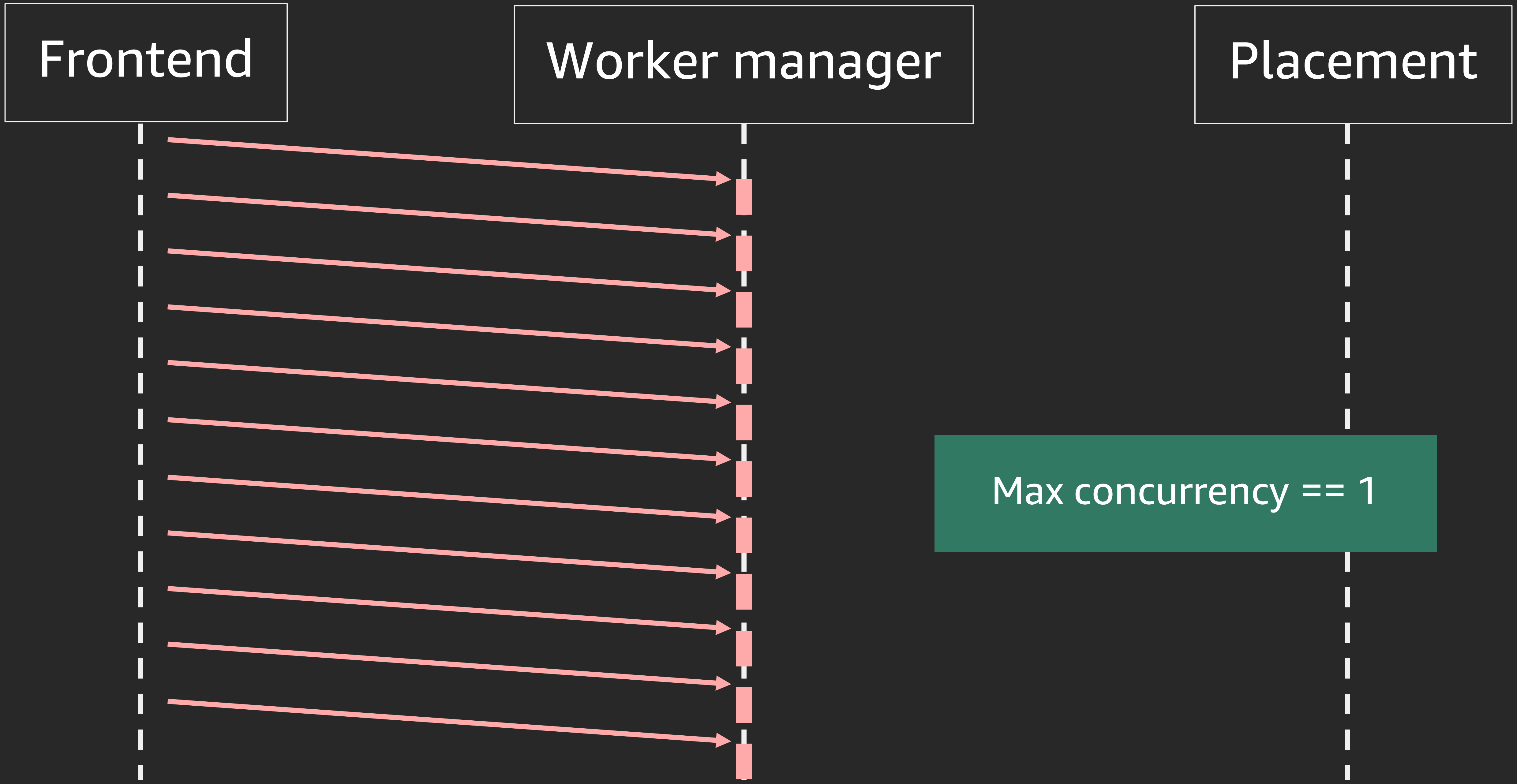
Warm invoke



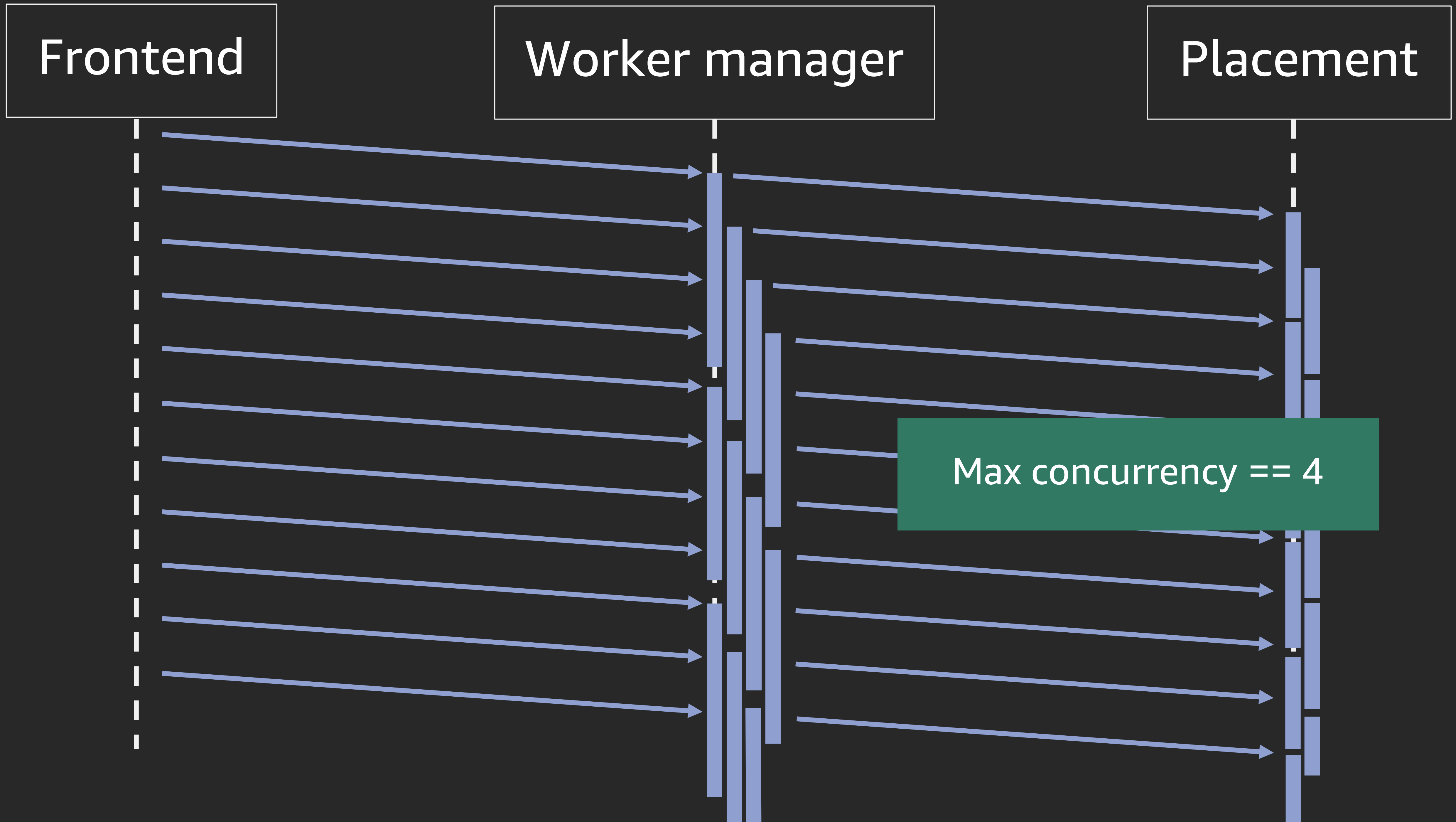
Cold invoke



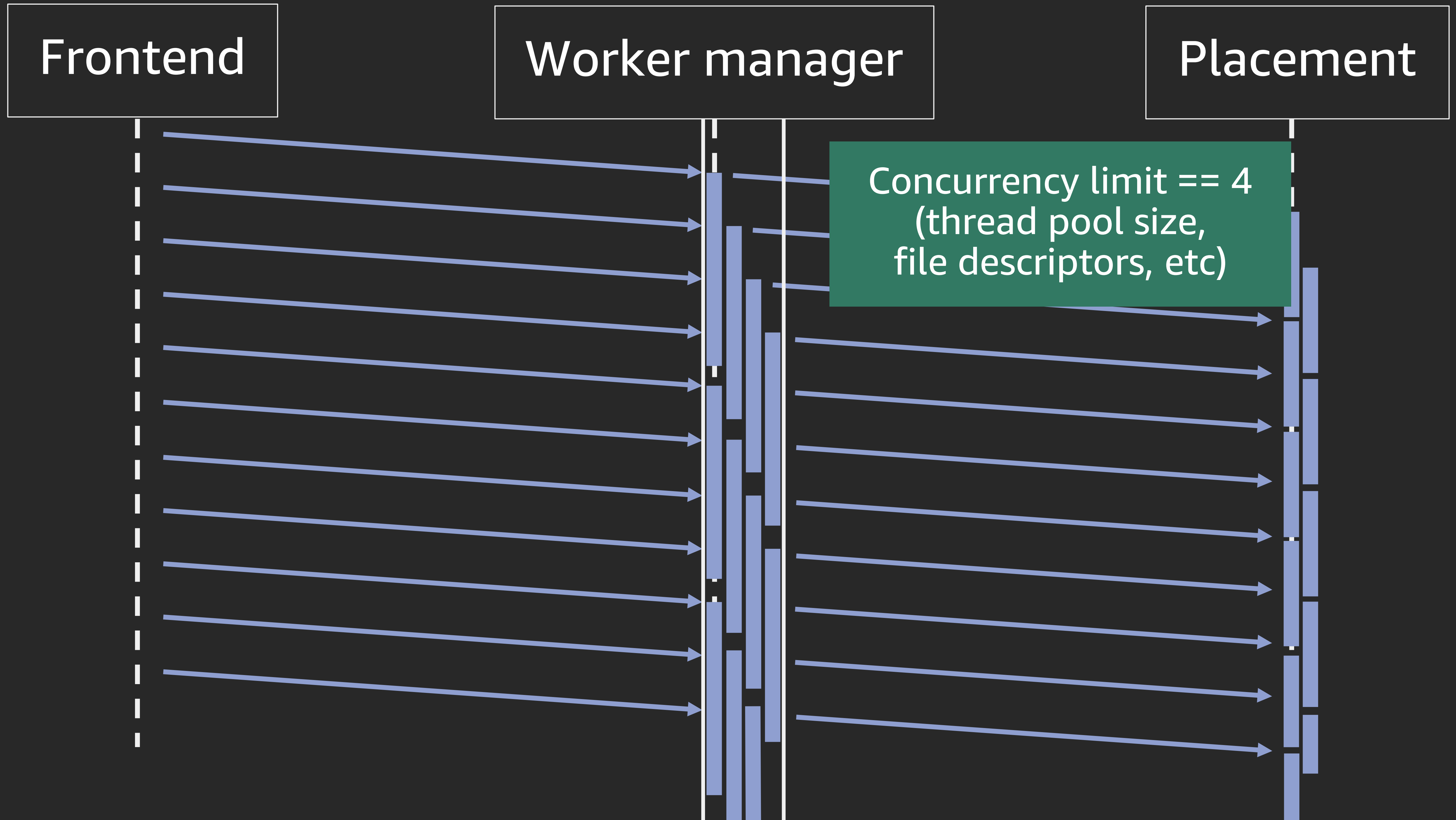
Concurrency from warm invokes



Flipping modes to cold invokes

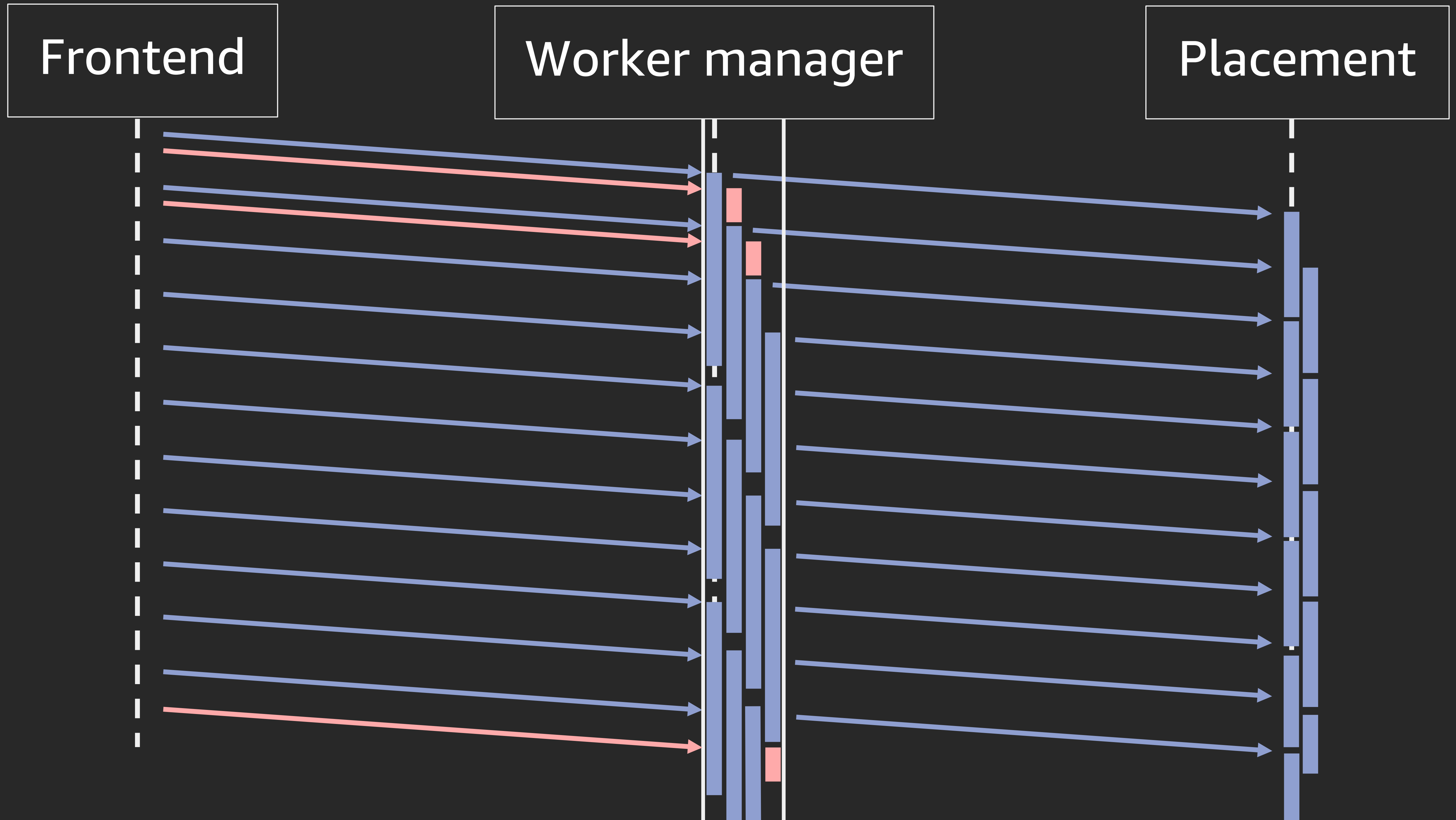


Concurrency limits

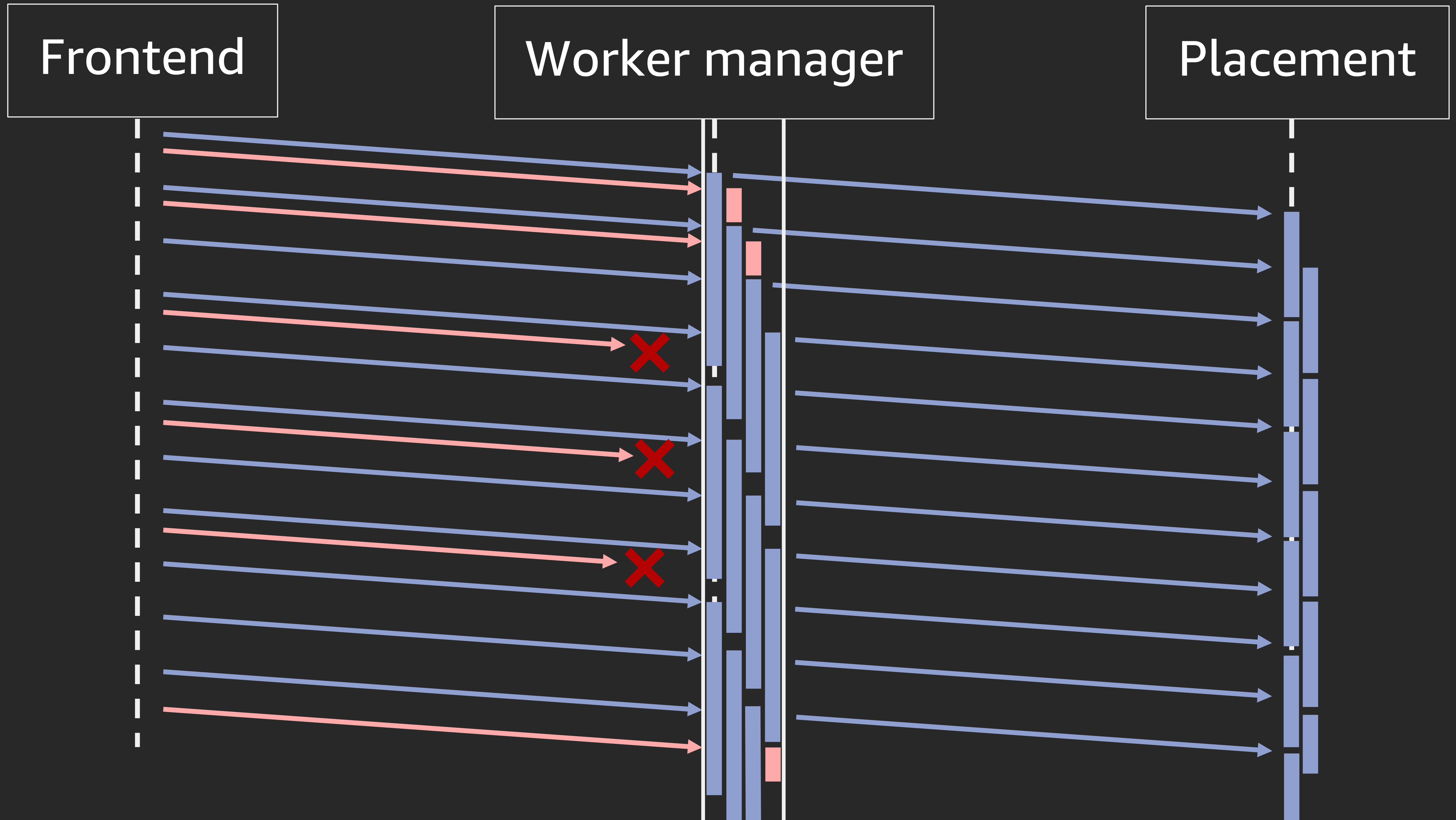


Concurrency limit == 4
(thread pool size,
file descriptors, etc)

Approaching concurrency limits



Exceeding concurrency limits



Worker manager behavior

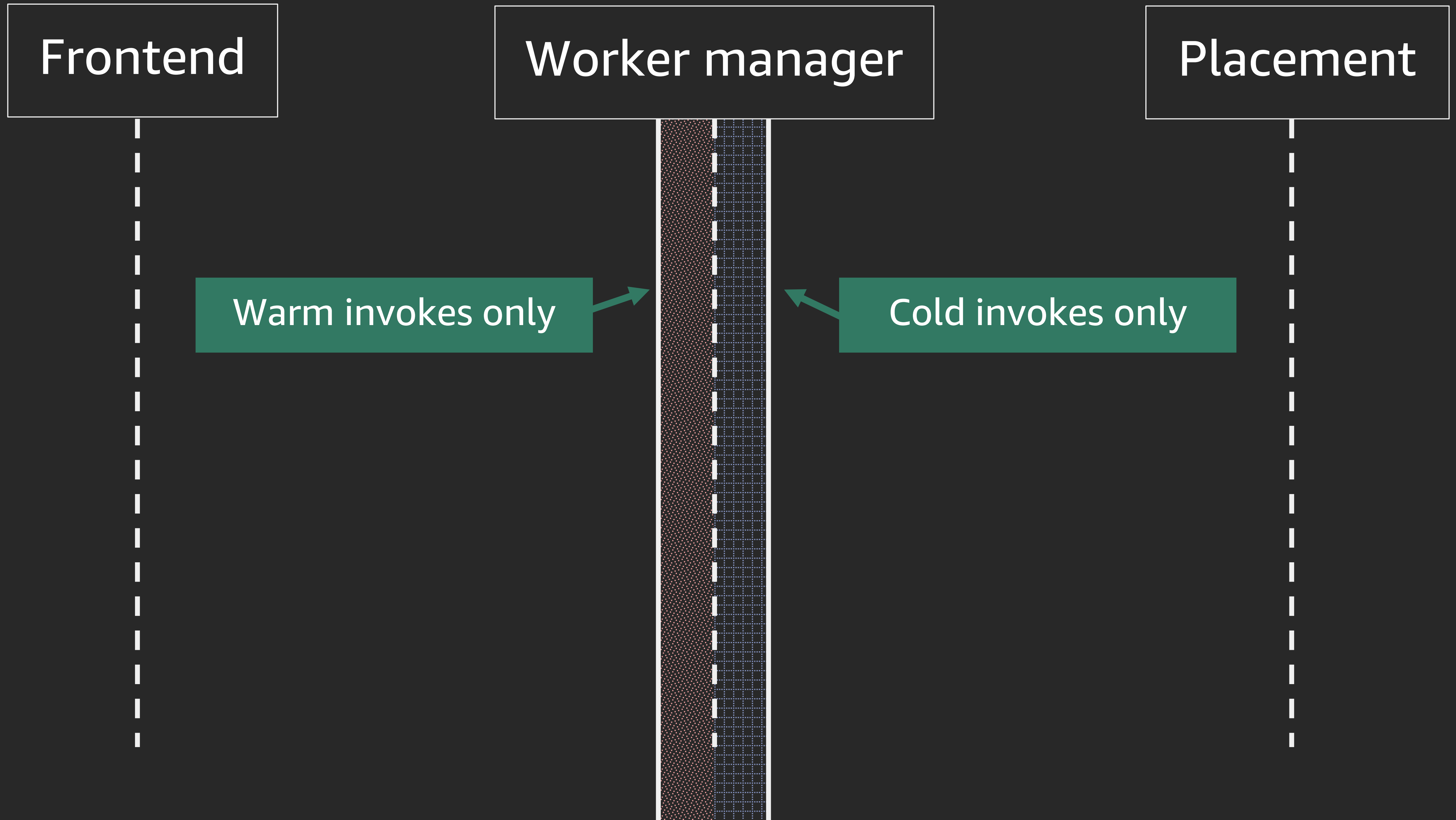
- Automatically scale
- Spread out increases in traffic to other worker managers
- Favor reusing existing sandboxes over creating new ones

Worker manager behavior

- Automatically scale
- Spread out increases in traffic to other worker managers
- Favor reusing existing sandboxes over creating new ones

$$warm\ start > cold\ start$$

Isolating concurrency



Frontend

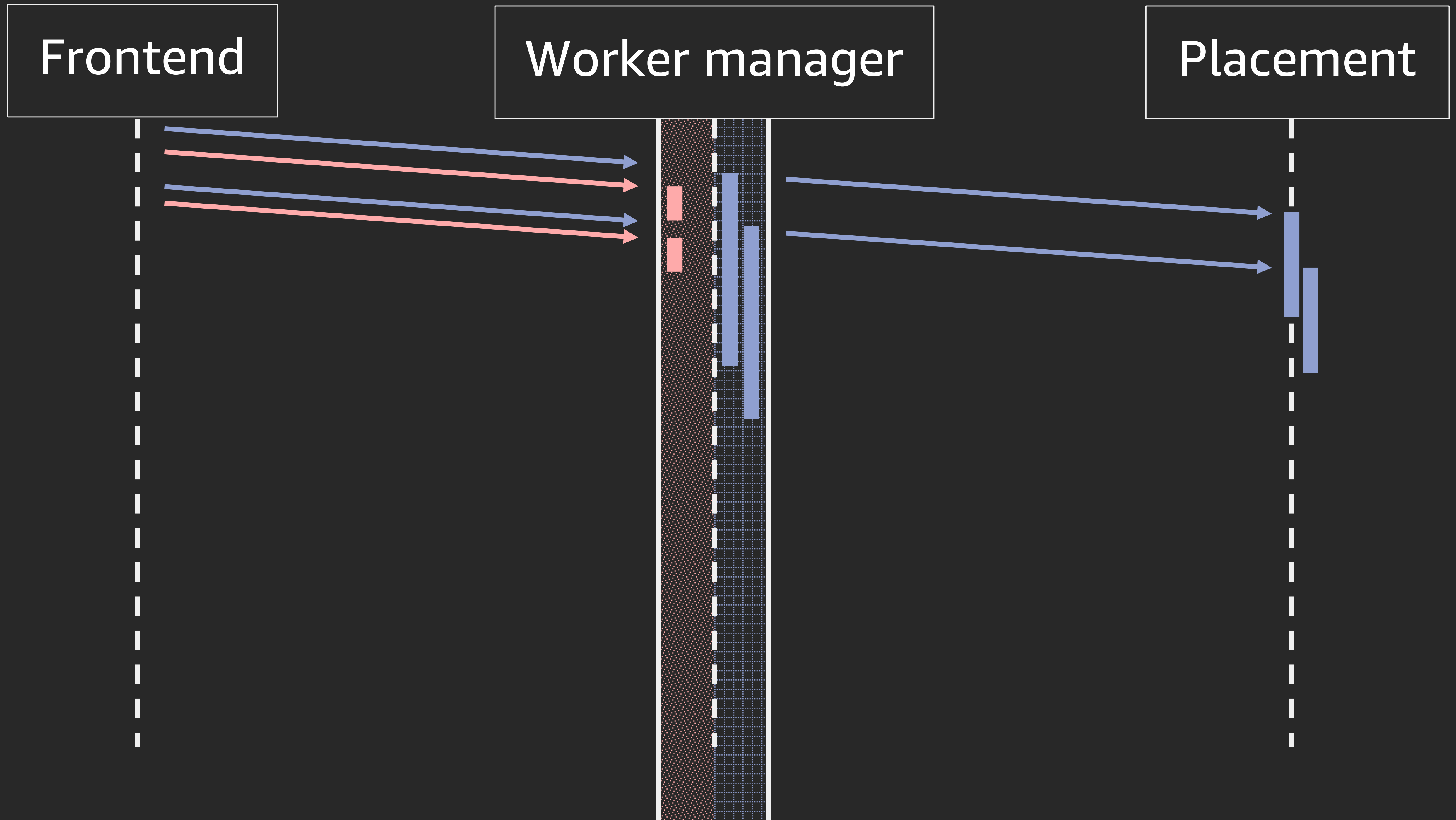
Worker manager

Placement

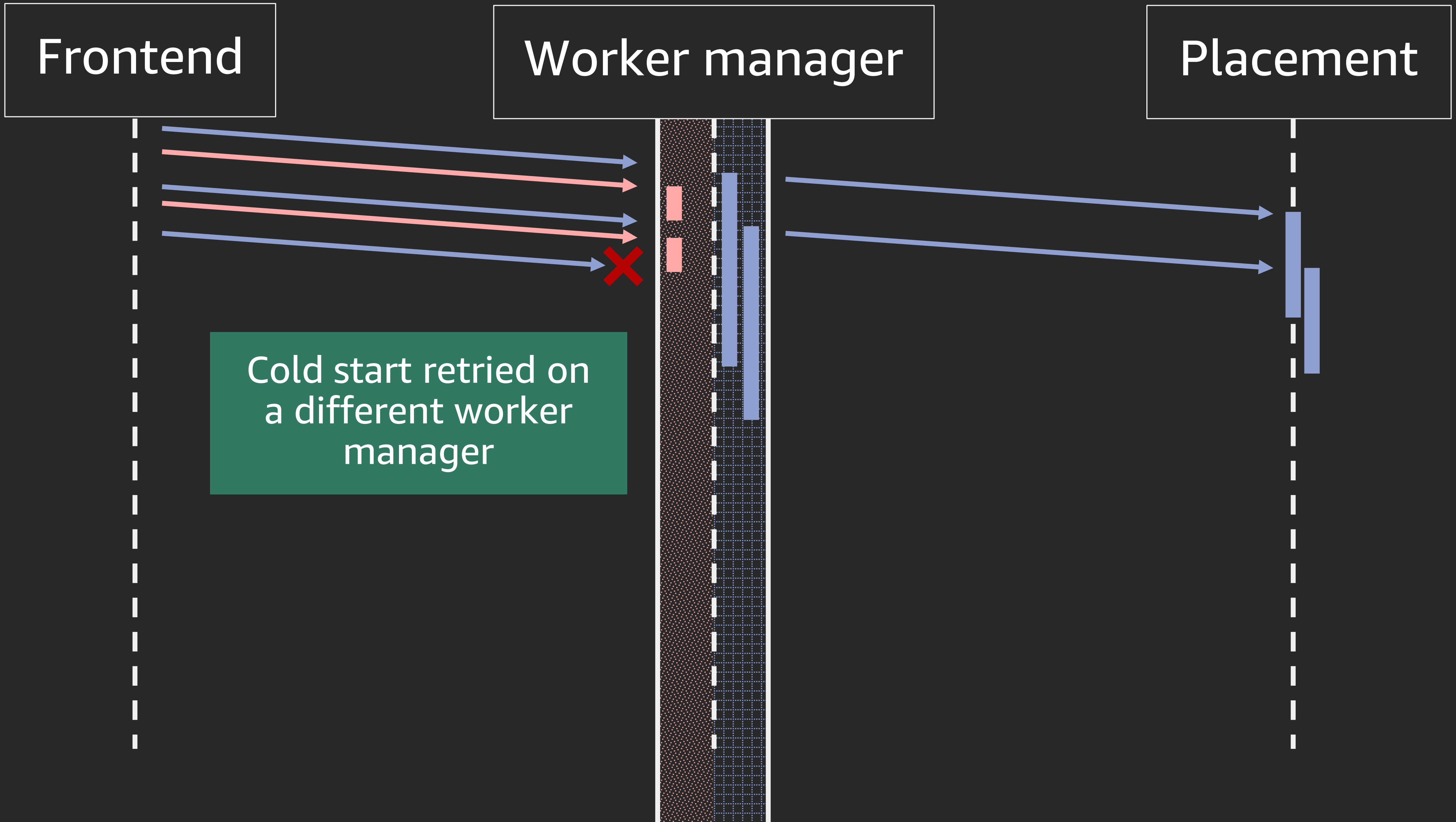
Warm invokes only

Cold invokes only

Isolating concurrency

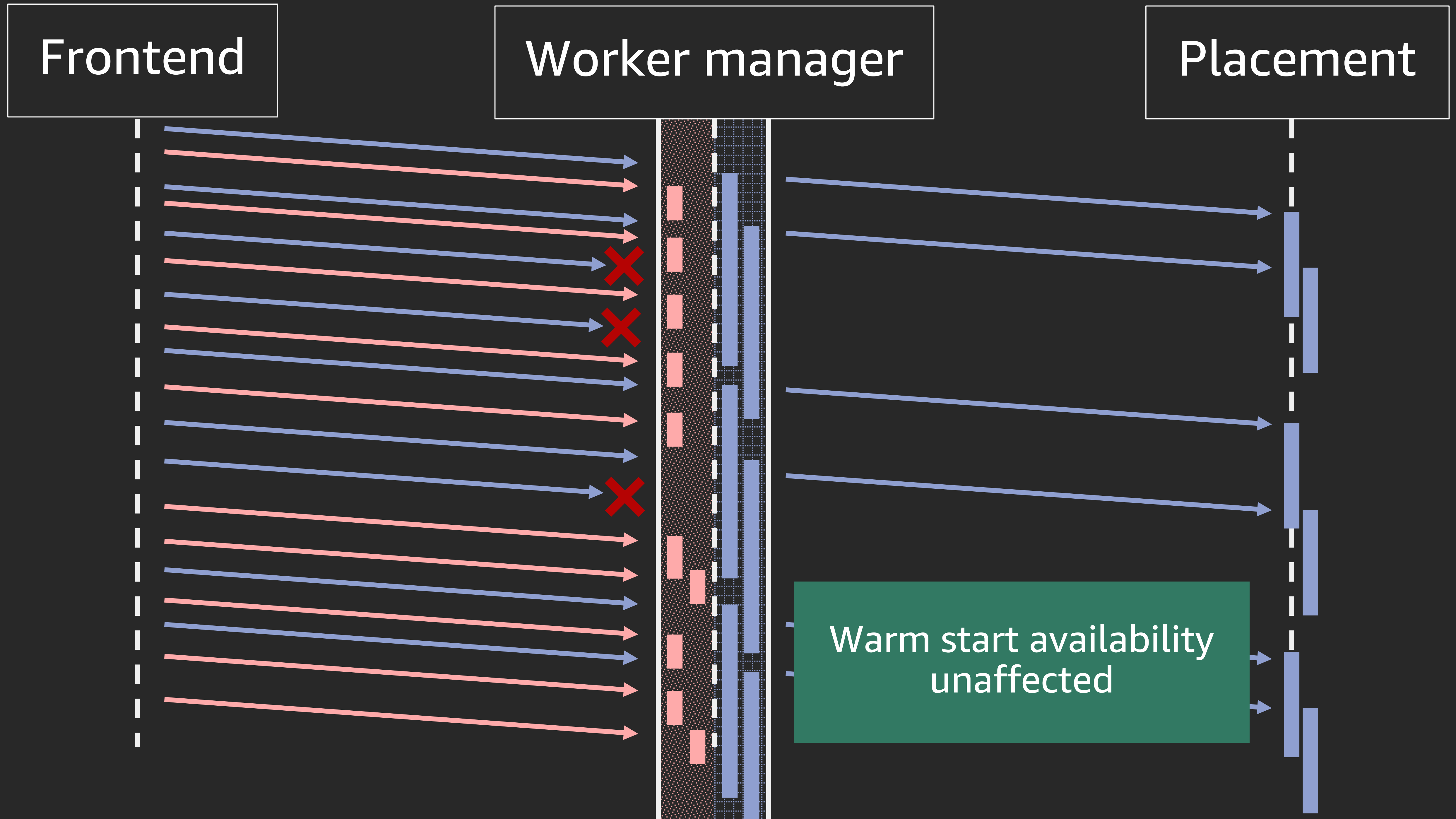


Isolating concurrency

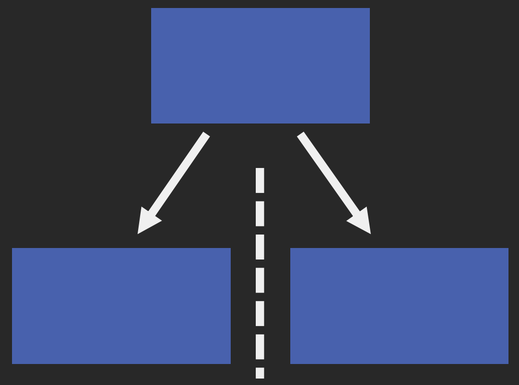


Cold start retried on a different worker manager

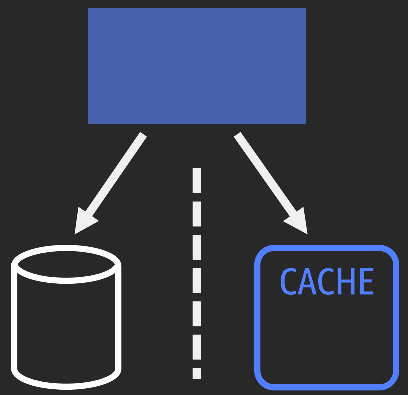
Isolating concurrency



Dependency isolation: Why?



Isolate
unrelated APIs



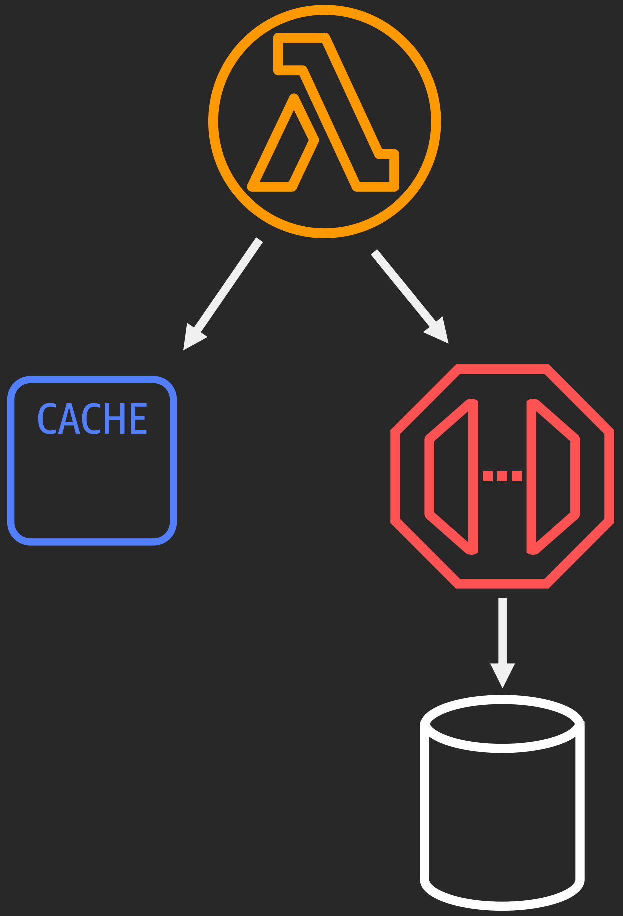
Protect against
modal behavior

API isolation: How?

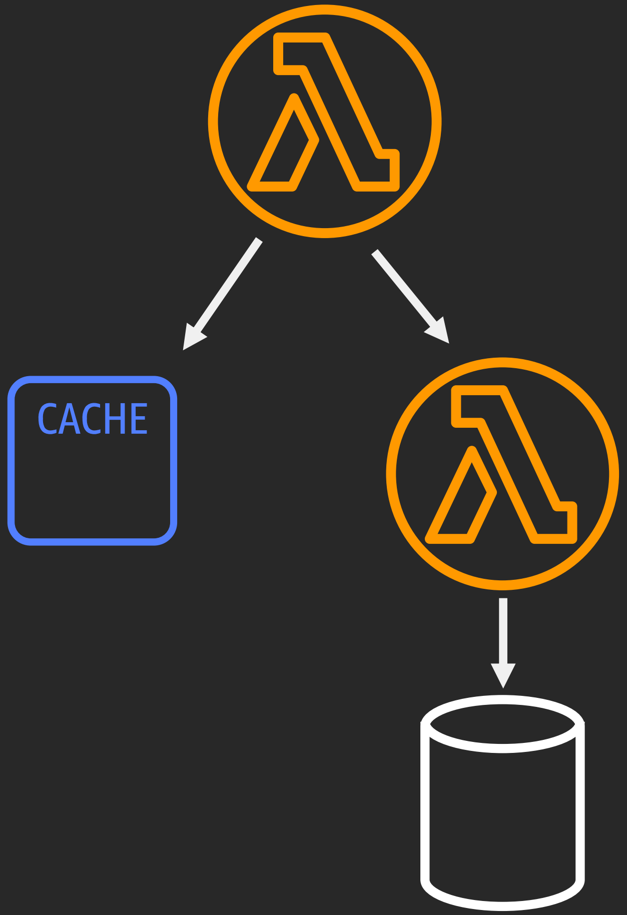
Amazon API Gateway
per-API throttling

Lambda per-function
concurrency controls

Modal behavior protection: How?



API Gateway throttling

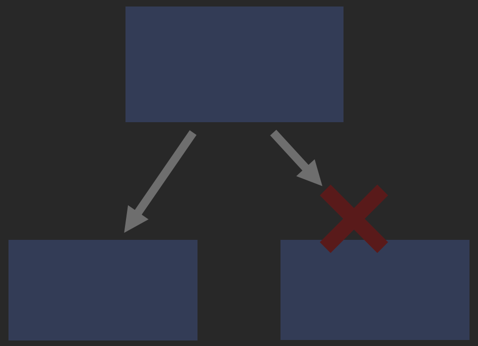


Per-function concurrency

Chapter three

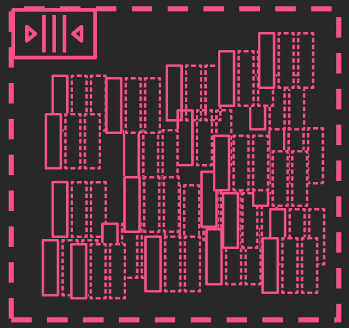
Load shedding

Avoid brownout by rejecting excess load



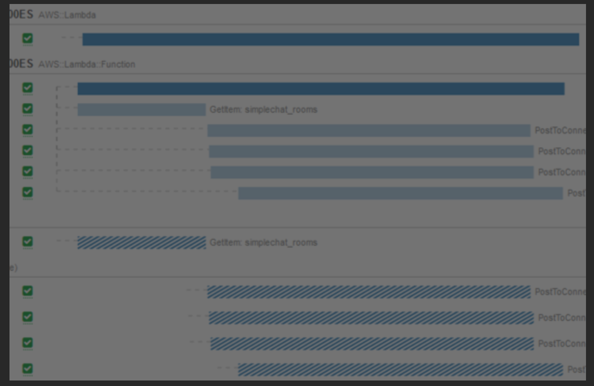
Dependency isolation

Prevent one dependency from affecting unrelated functionality



Avoiding queue backlogs

Prevent a backlog from extending recovery time



Operating

Quickly diagnose and mitigate issues

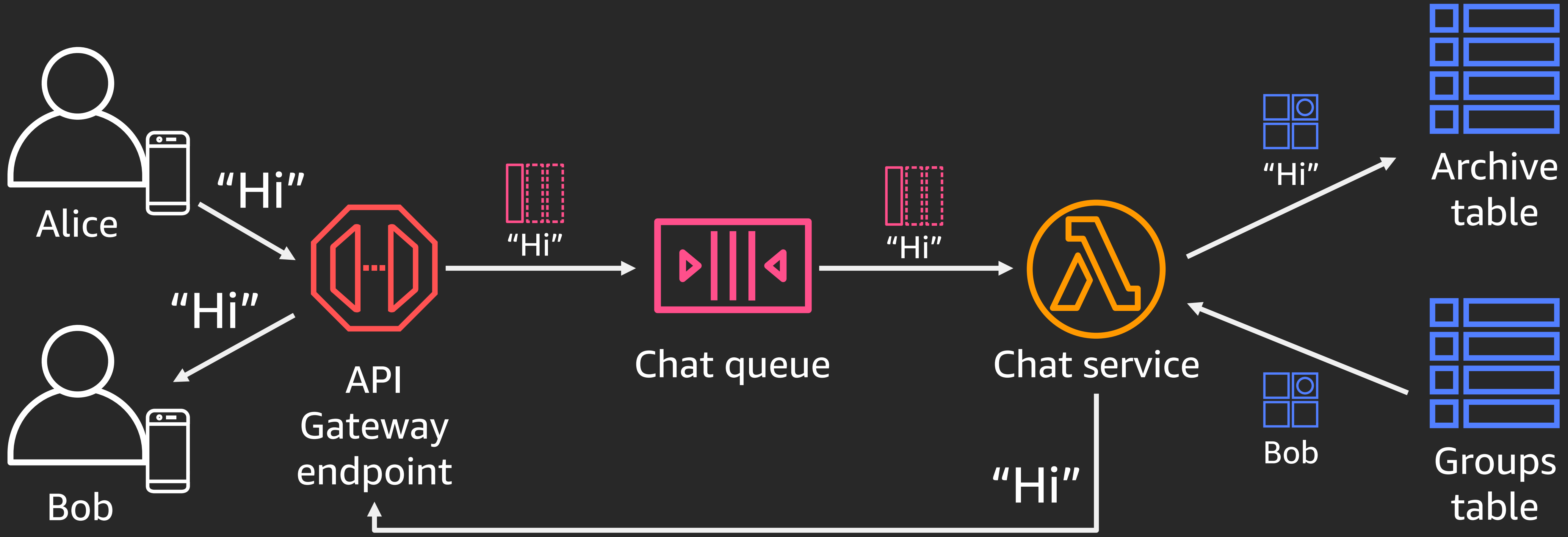
BANK
4:30
$
Store
Store
Store

BANK
5:00
$
Store
Store
Store
$
$
$

BANK
5:30
$
$
$
$
store
store
store

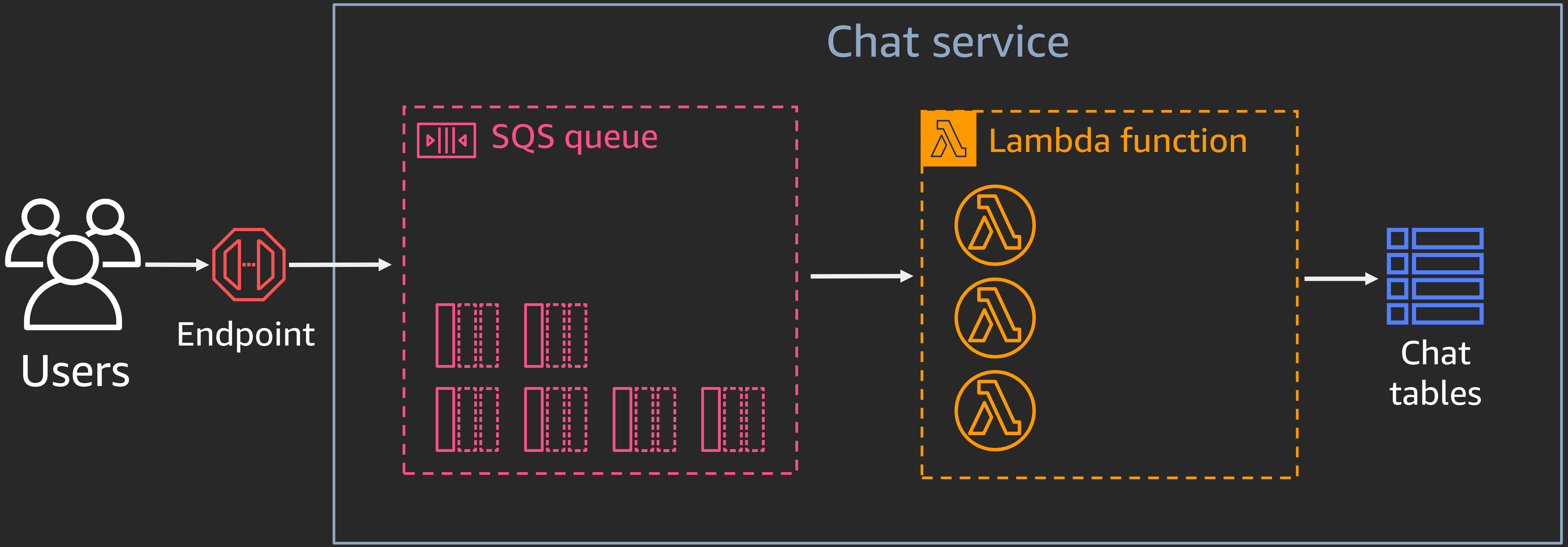
BANK
8:00
$
Store
Store
Store

Basic chat architecture

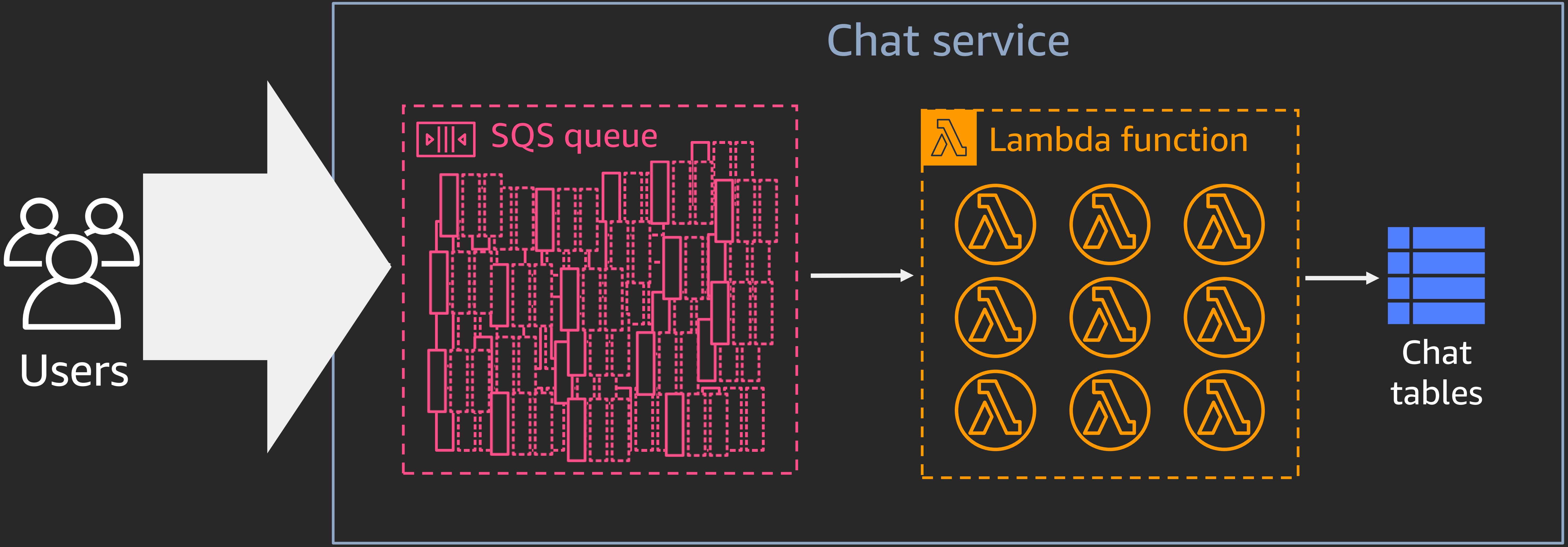


https://github.com/aws-samples/simple-websockets-chat-app

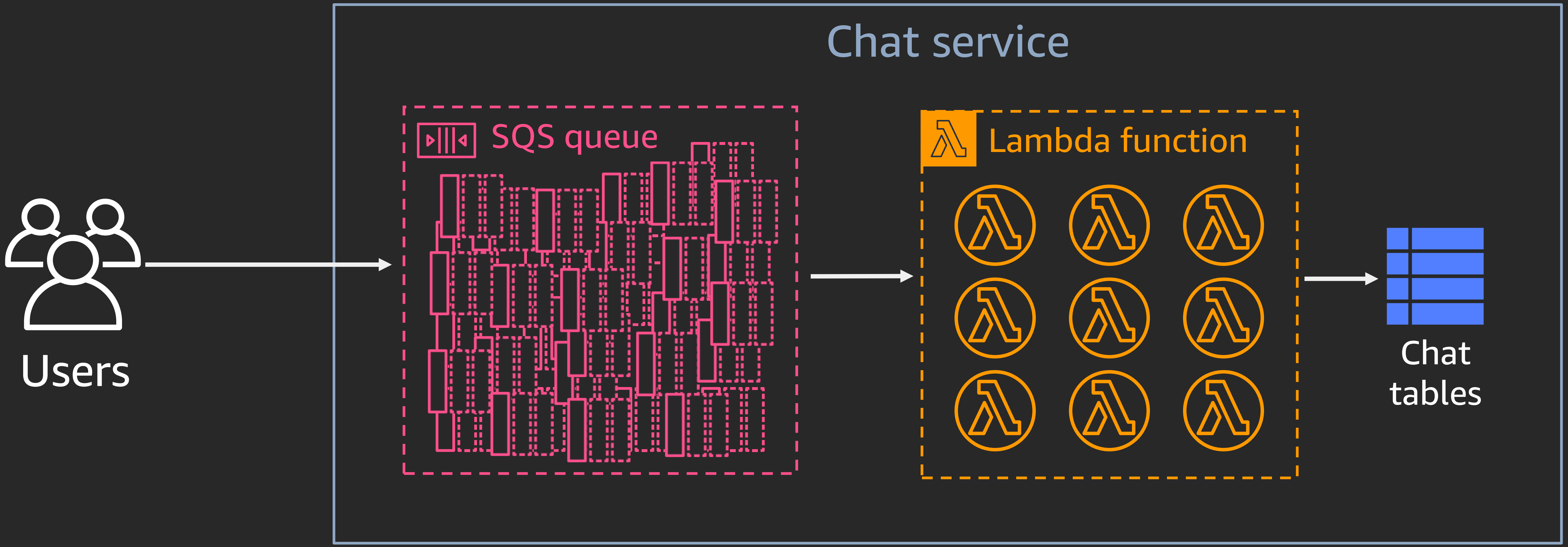
Business as usual



Surge in traffic

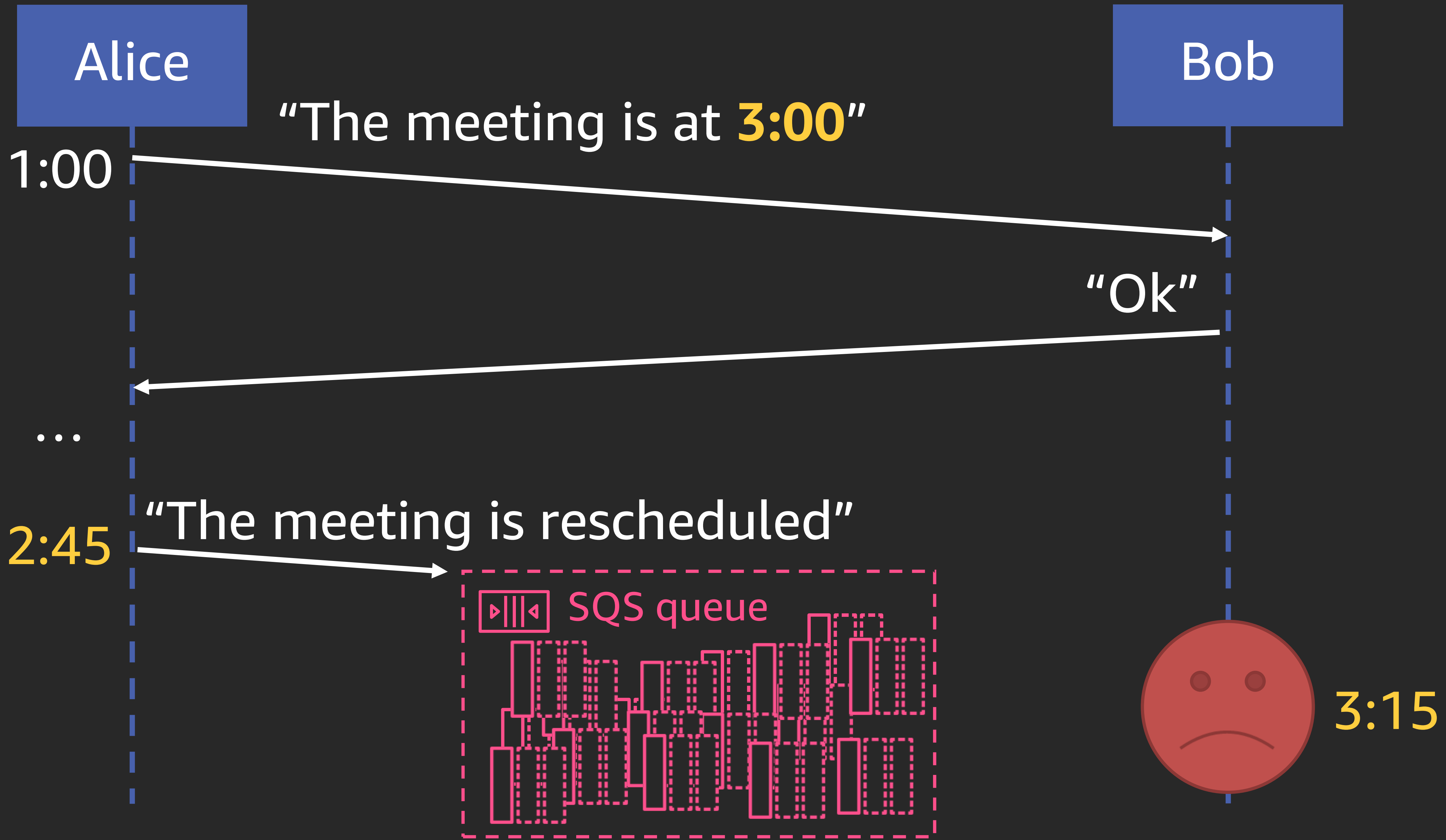


Surge subsides, backlog remains



Ideal customer experience?

(Un)Ideal customer experience?



Alice

Bob

1:00

“The meeting is at **3:00**”

“Ok”

…

2:45

“The meeting is rescheduled”

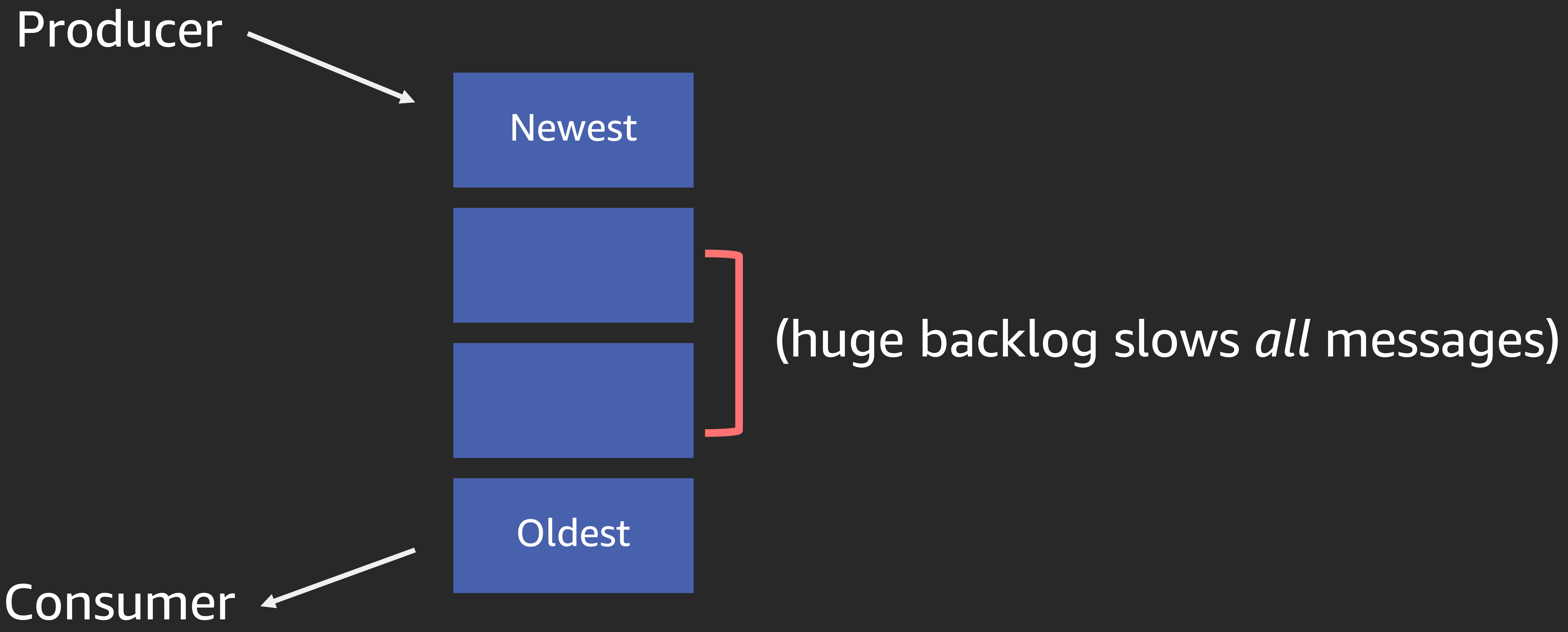
SQS queue

3:15

Queue backlogs are bad

FIFO

(First In, First Out)



Producer

Newest

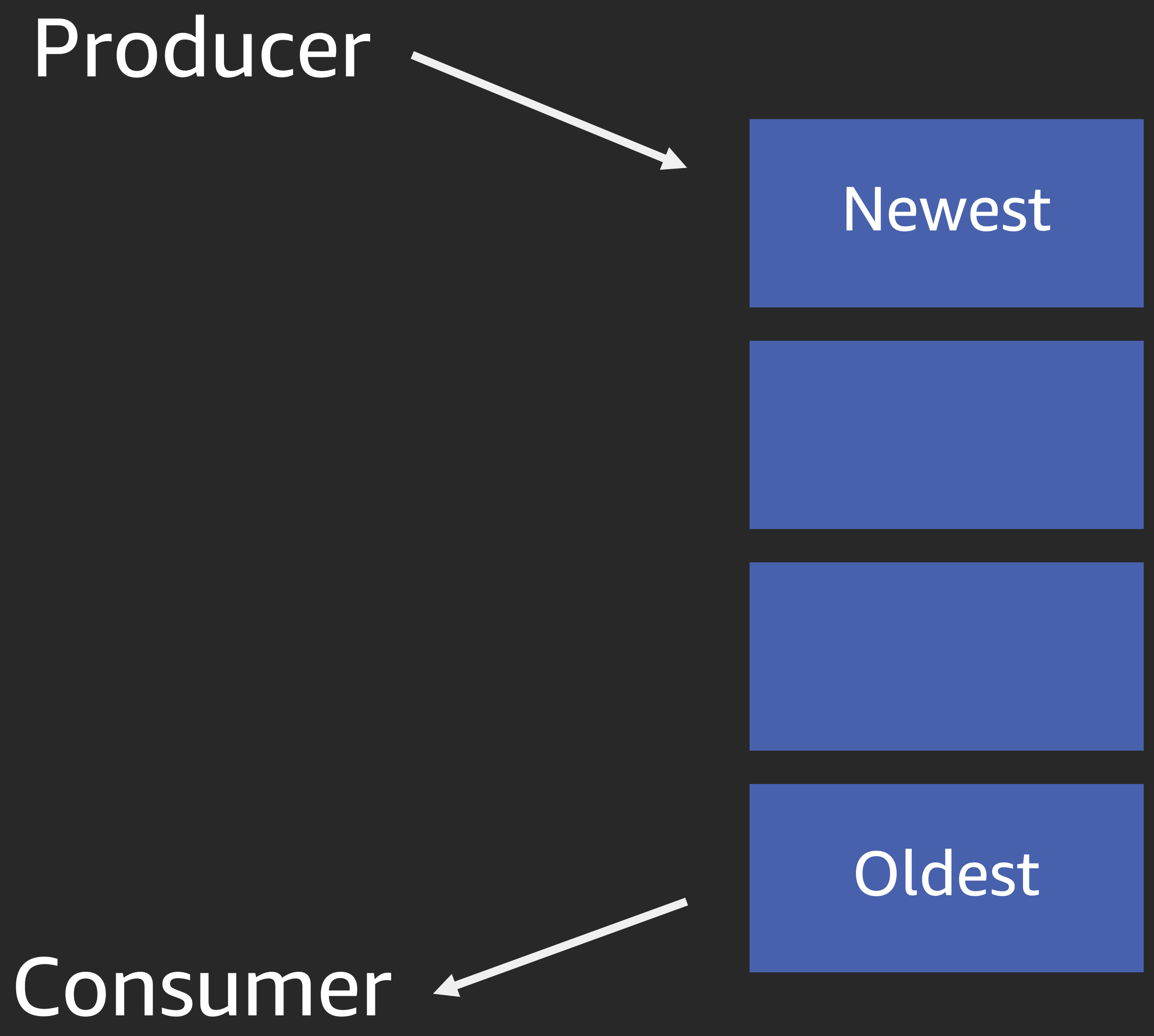
Oldest

(huge backlog slows *all* messages)

Consumer

FIFO

(First In, First Out)



Producer

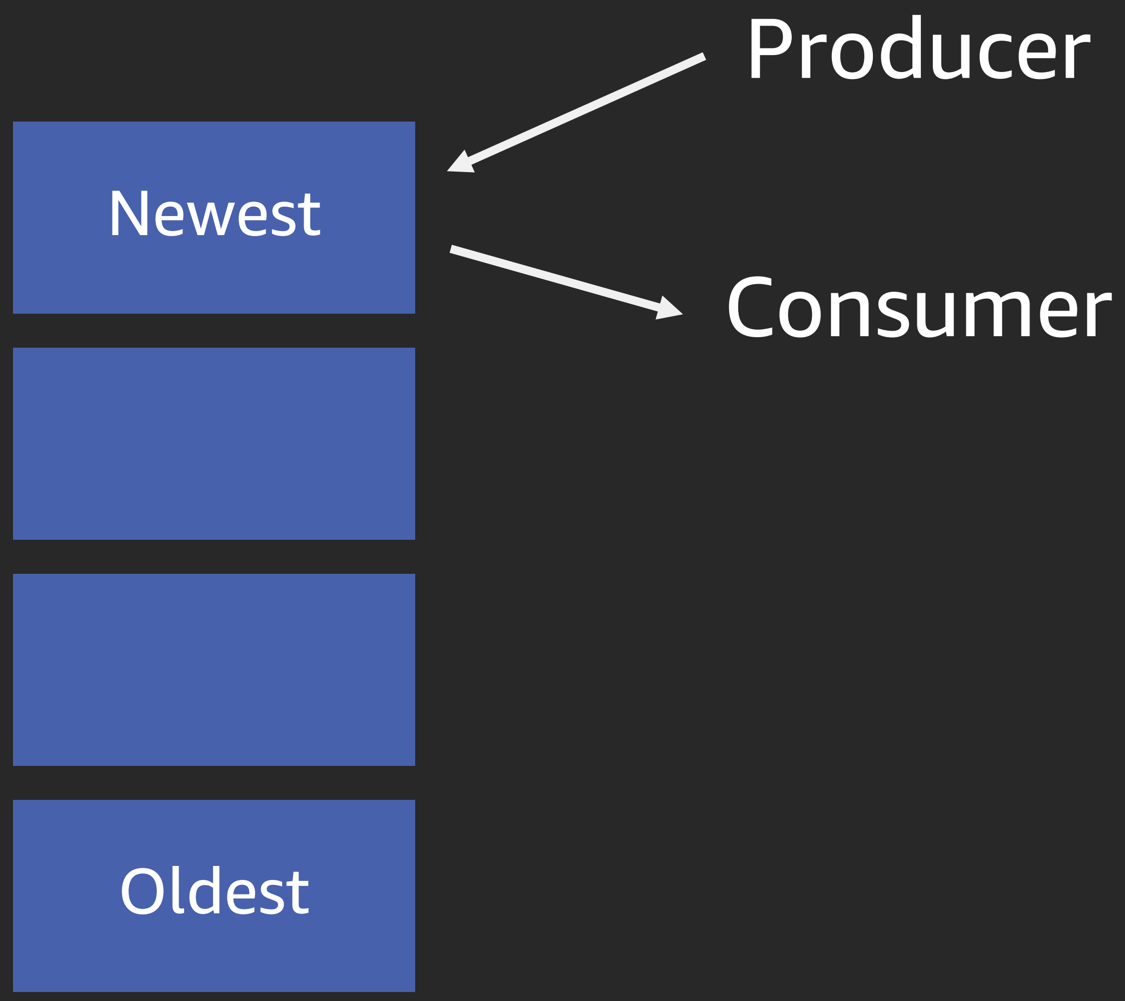
Newest

Oldest

Consumer

LIFO

(Last In, First Out)



Producer

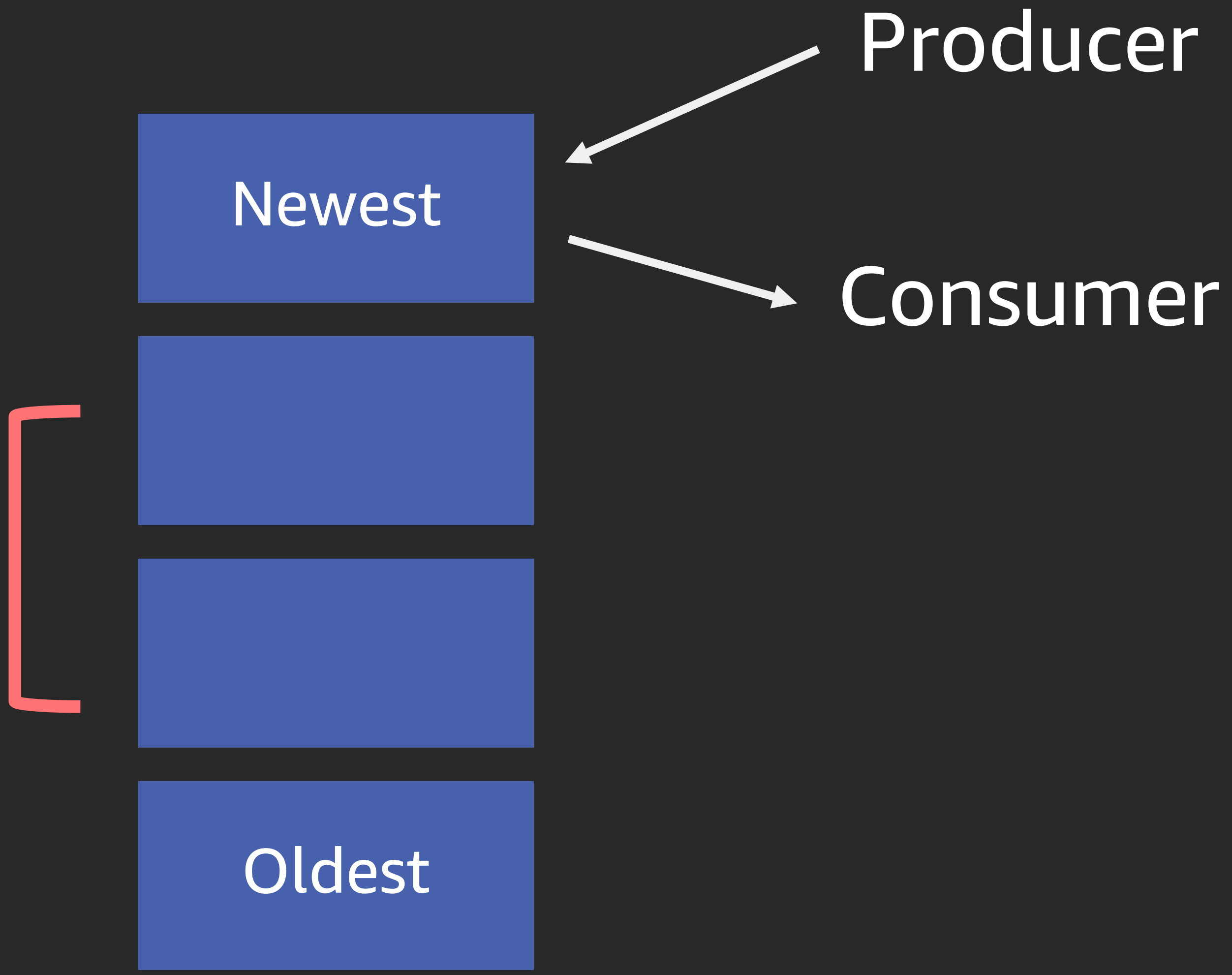
Newest

Consumer

Oldest

LIFO

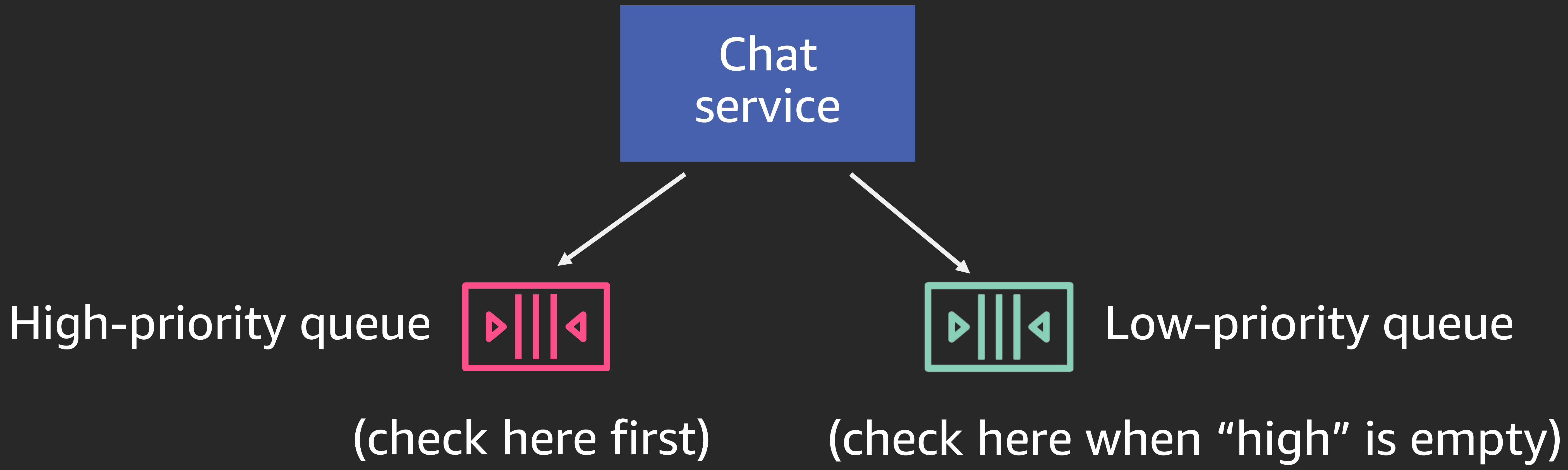
(Last In, First Out)



(huge backlog slows *old* messages)

Making FIFO behave LIFO

Priority queues



Chat service

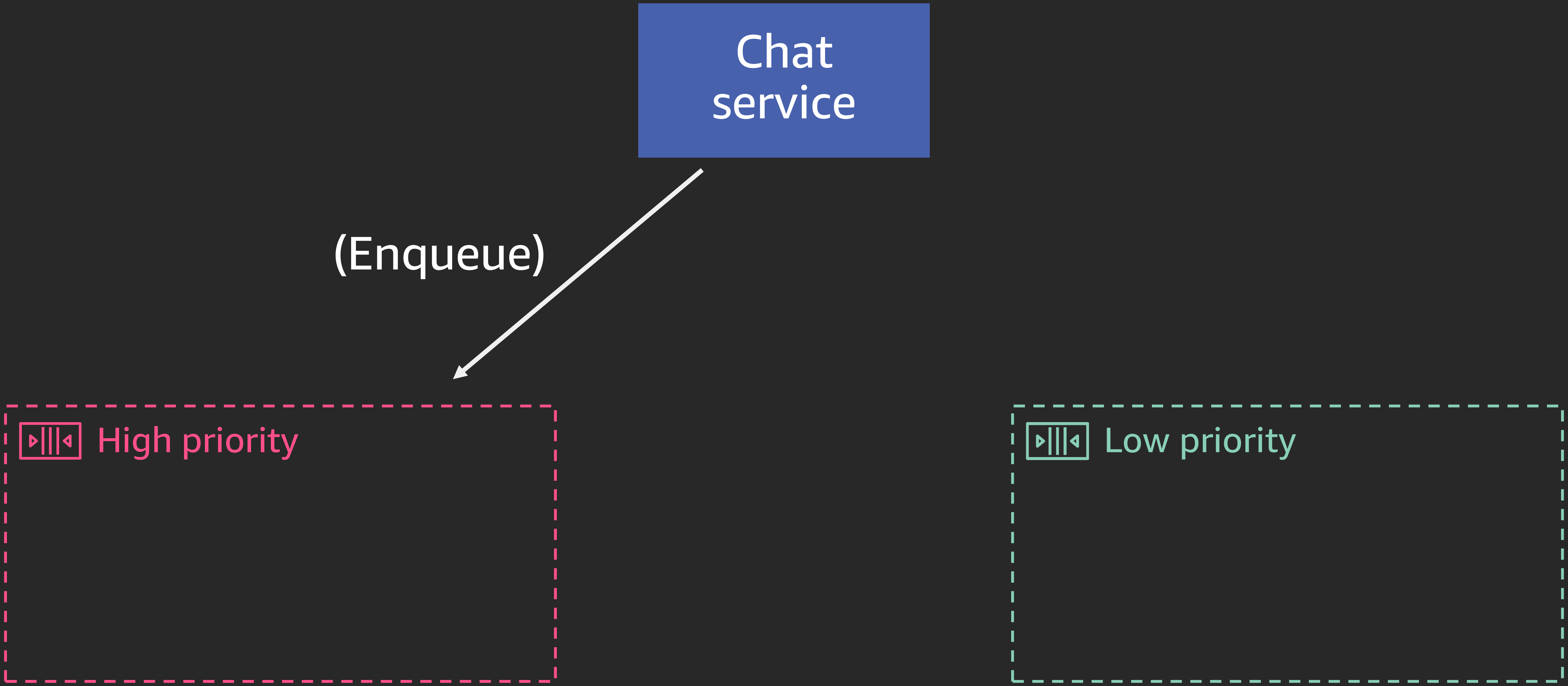
High-priority queue

Low-priority queue

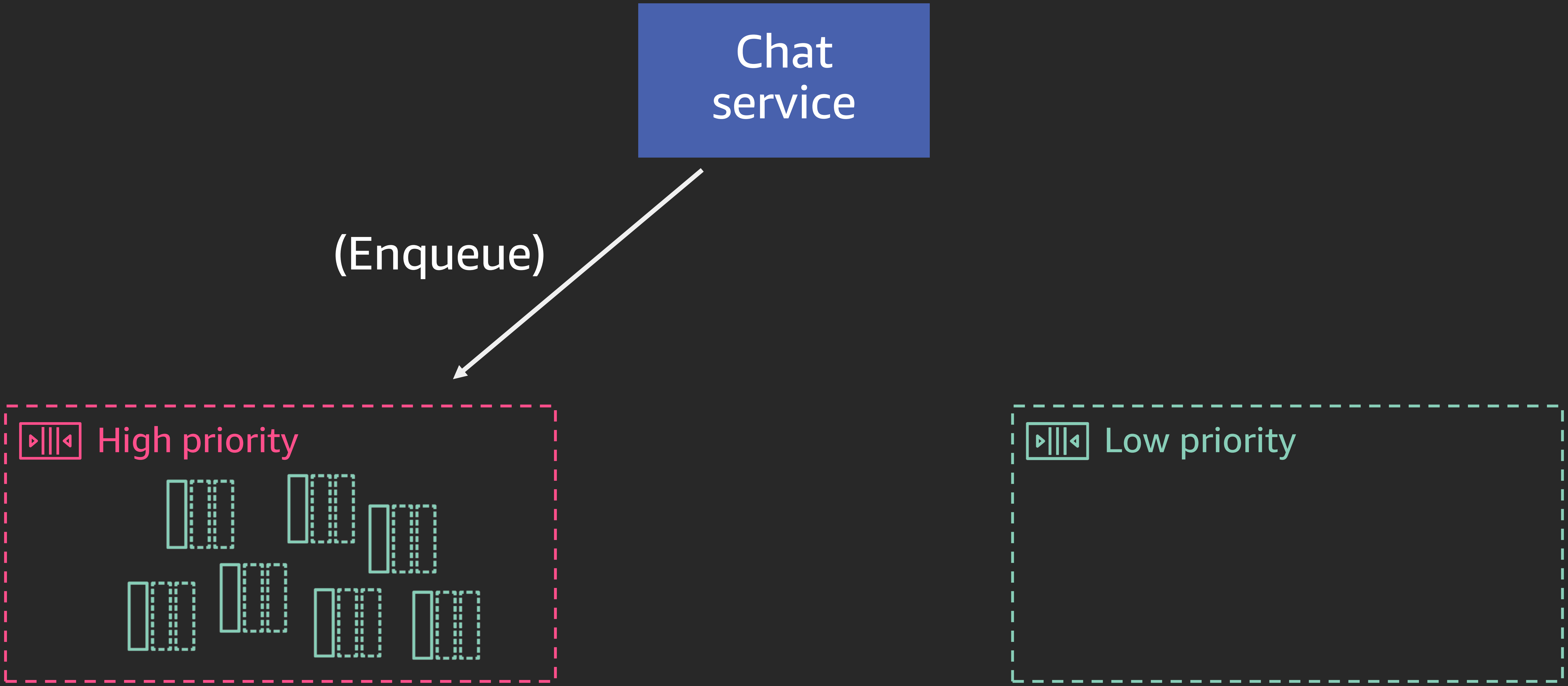
(check here first)

(check here when “high” is empty)

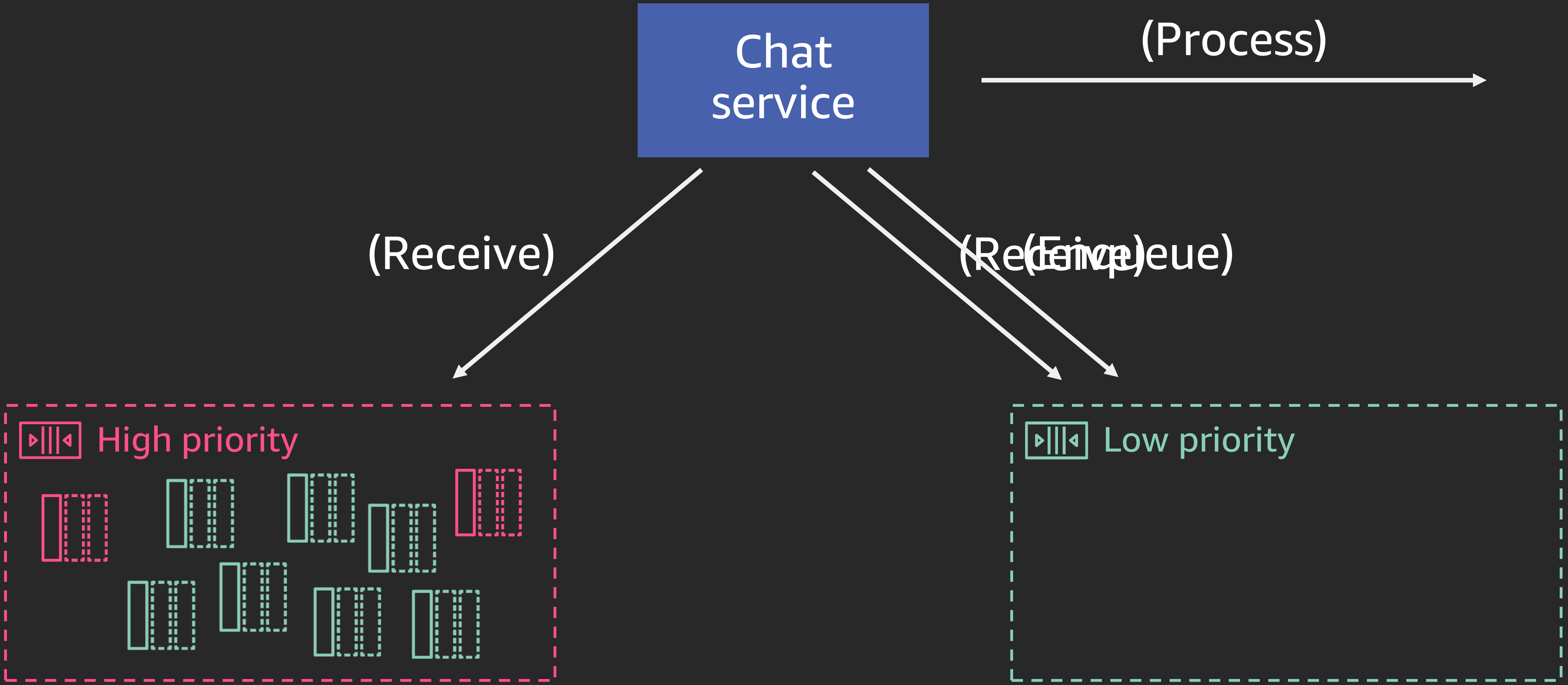
Priority is not known upfront



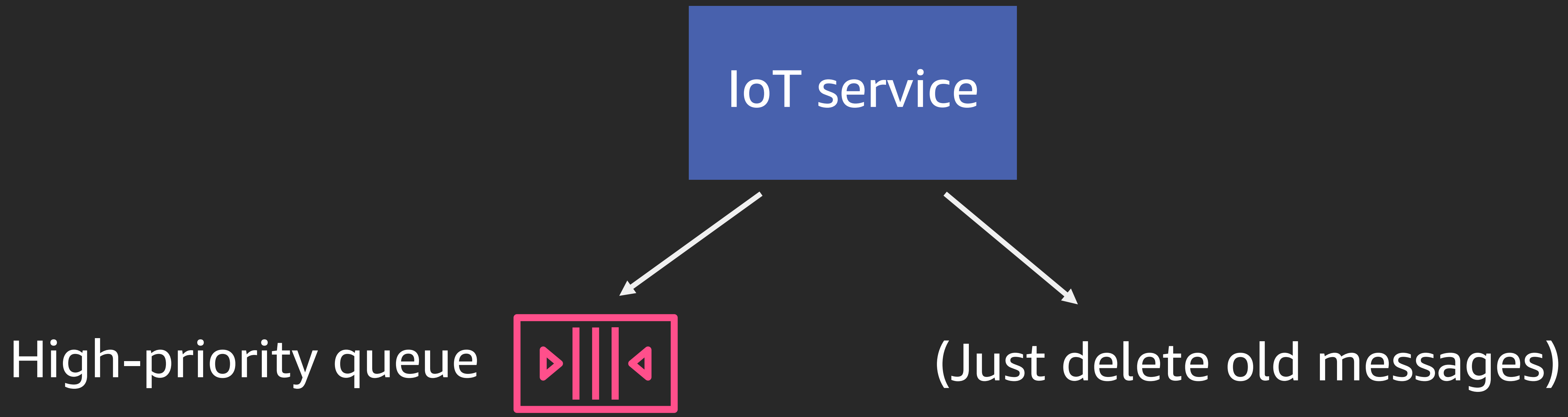
Priority is not known upfront



Move old messages to low-priority queue



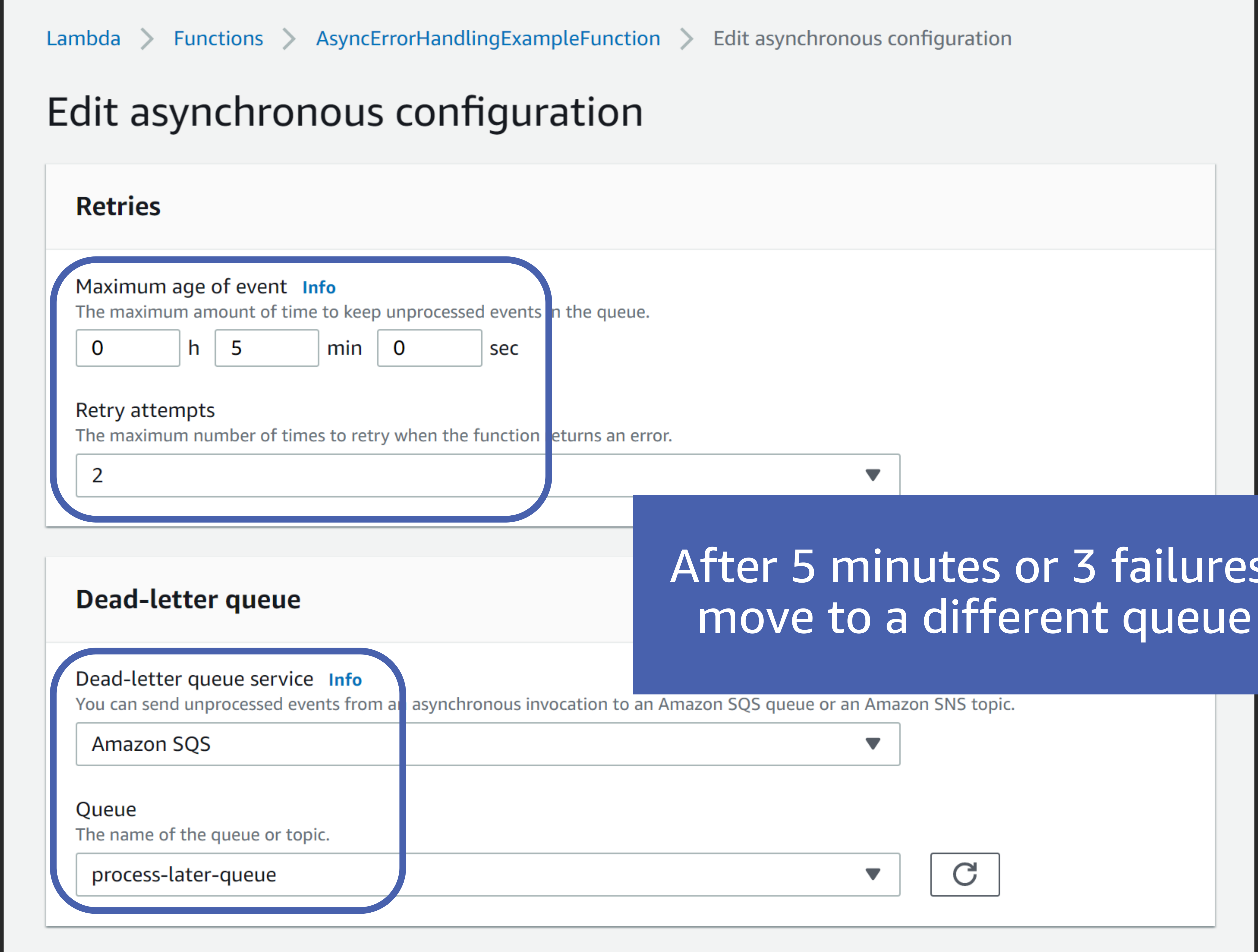
Time to Live (TTL)



IoT service

High-priority queue

(Just delete old messages)



Lambda > Functions > AsyncErrorHandlingExampleFunction > Edit asynchronous configuration

Edit asynchronous configuration

Retries

Maximum age of event Info
The maximum amount of time to keep unprocessed events in the queue.
0 h 5 min 0 sec

Retry attempts
The maximum number of times to retry when the function returns an error.
2 ▼

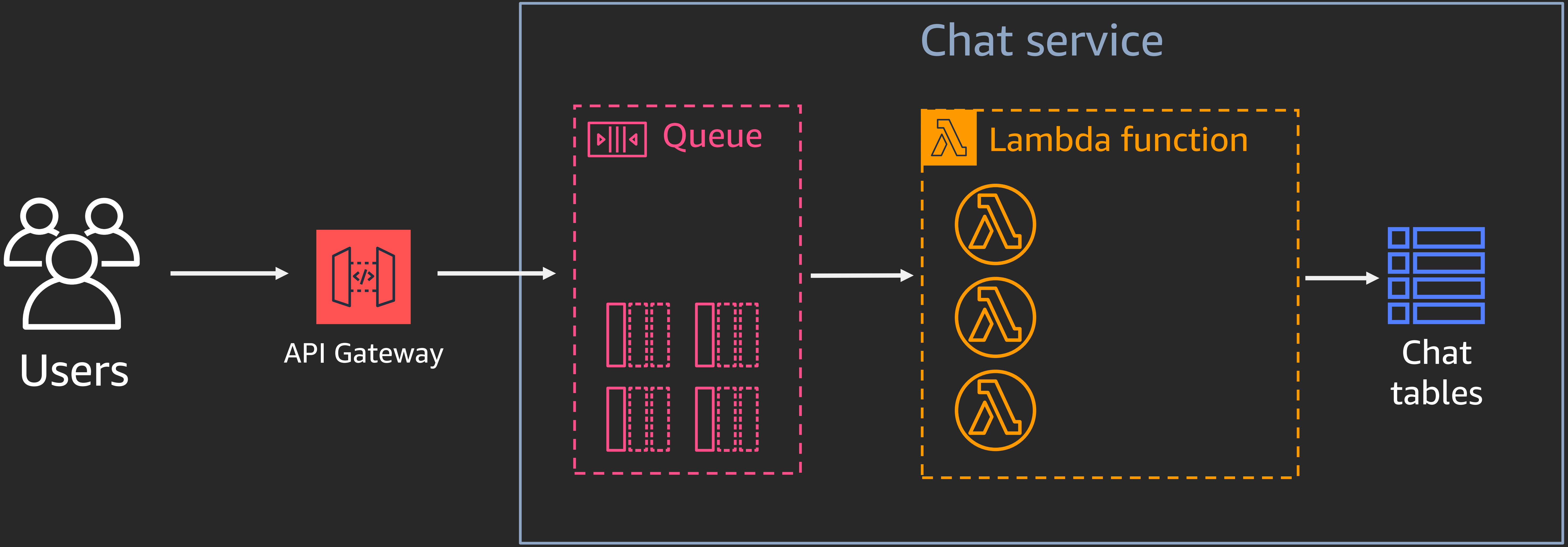
Dead-letter queue

Dead-letter queue service Info
You can send unprocessed events from an asynchronous invocation to an Amazon SQS queue or an Amazon SNS topic.
Amazon SQS ▼

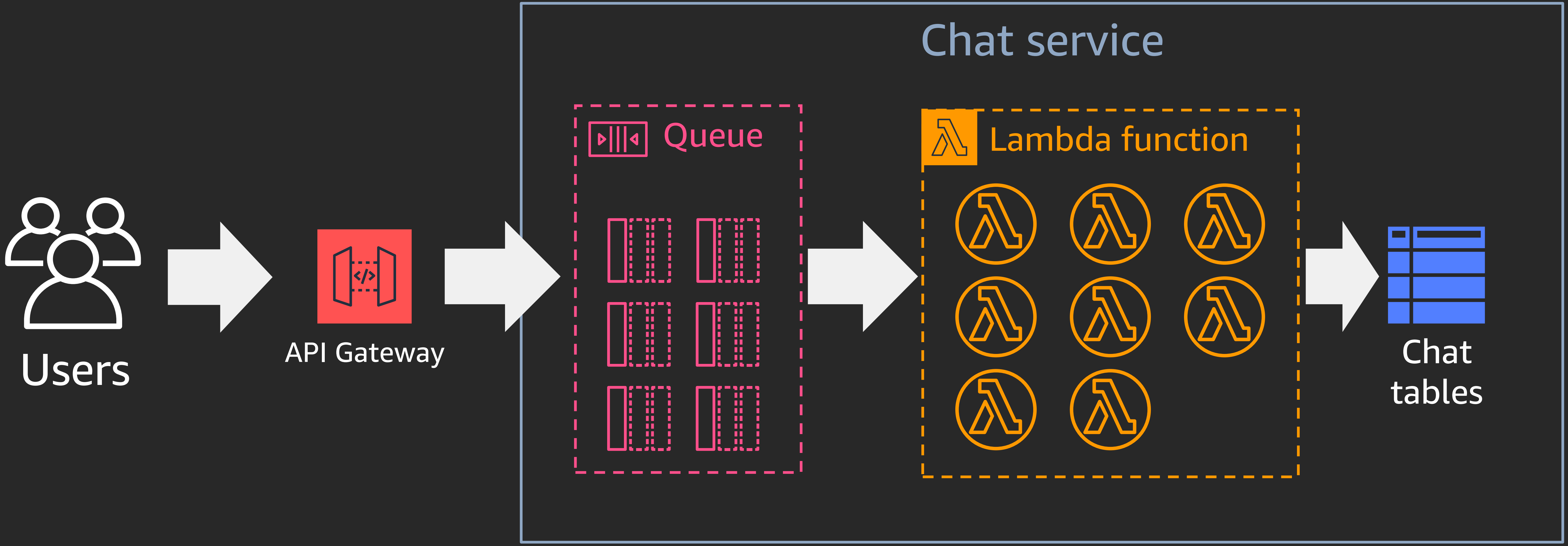
Queue
The name of the queue or topic.
process-later-queue ▼

After 5 minutes or 3 failures, move to a different queue

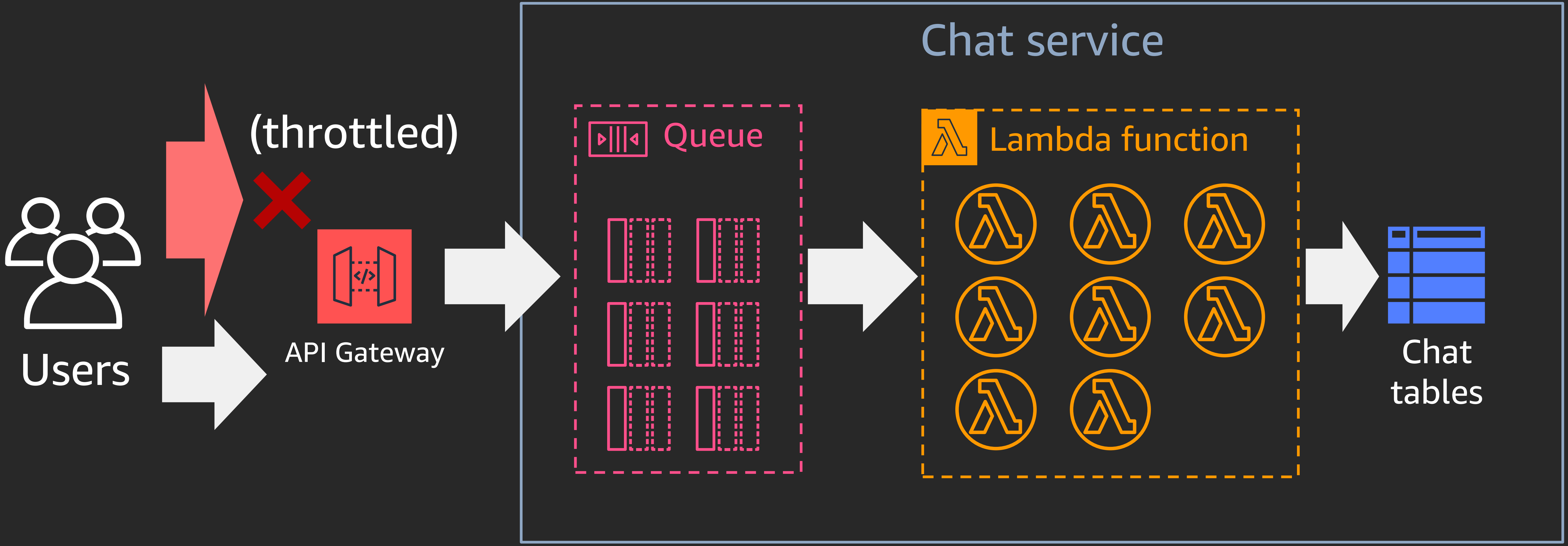
Backpressure (throttling)



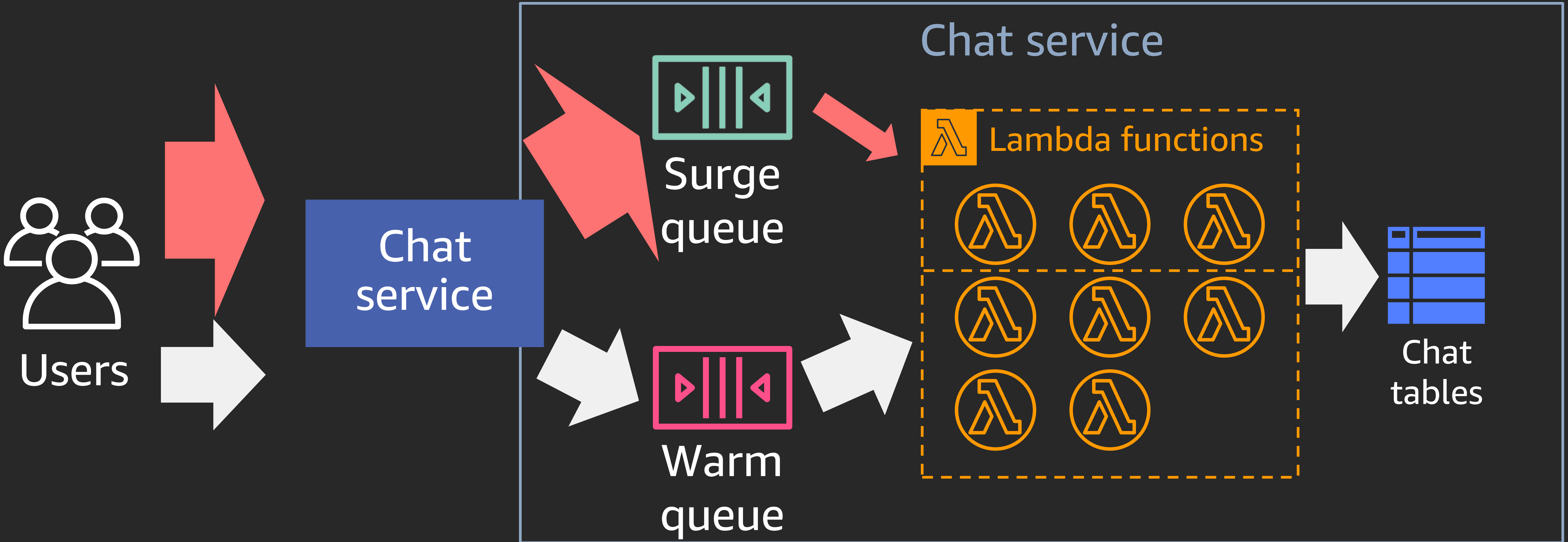
Backpressure (throttling)



Backpressure (throttling)



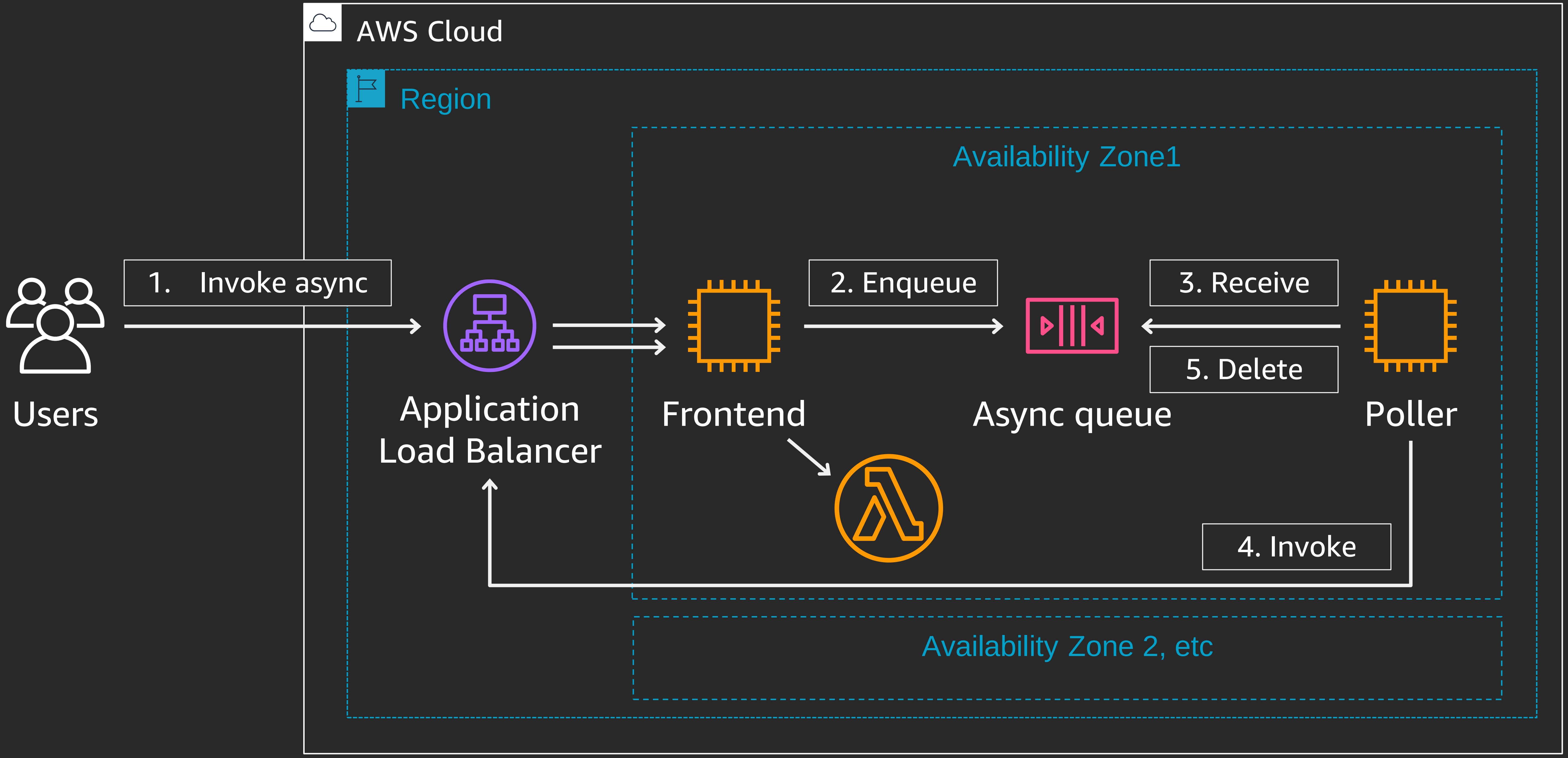
Priority queues + throttling: Best of both worlds?



Shuffle-sharding

Under the hood: Lambda async

Lambda async



AWS Cloud

Region

Availability Zone1

1. Invoke async

2. Enqueue

3. Receive

5. Delete

4. Invoke

Users

Application Load Balancer

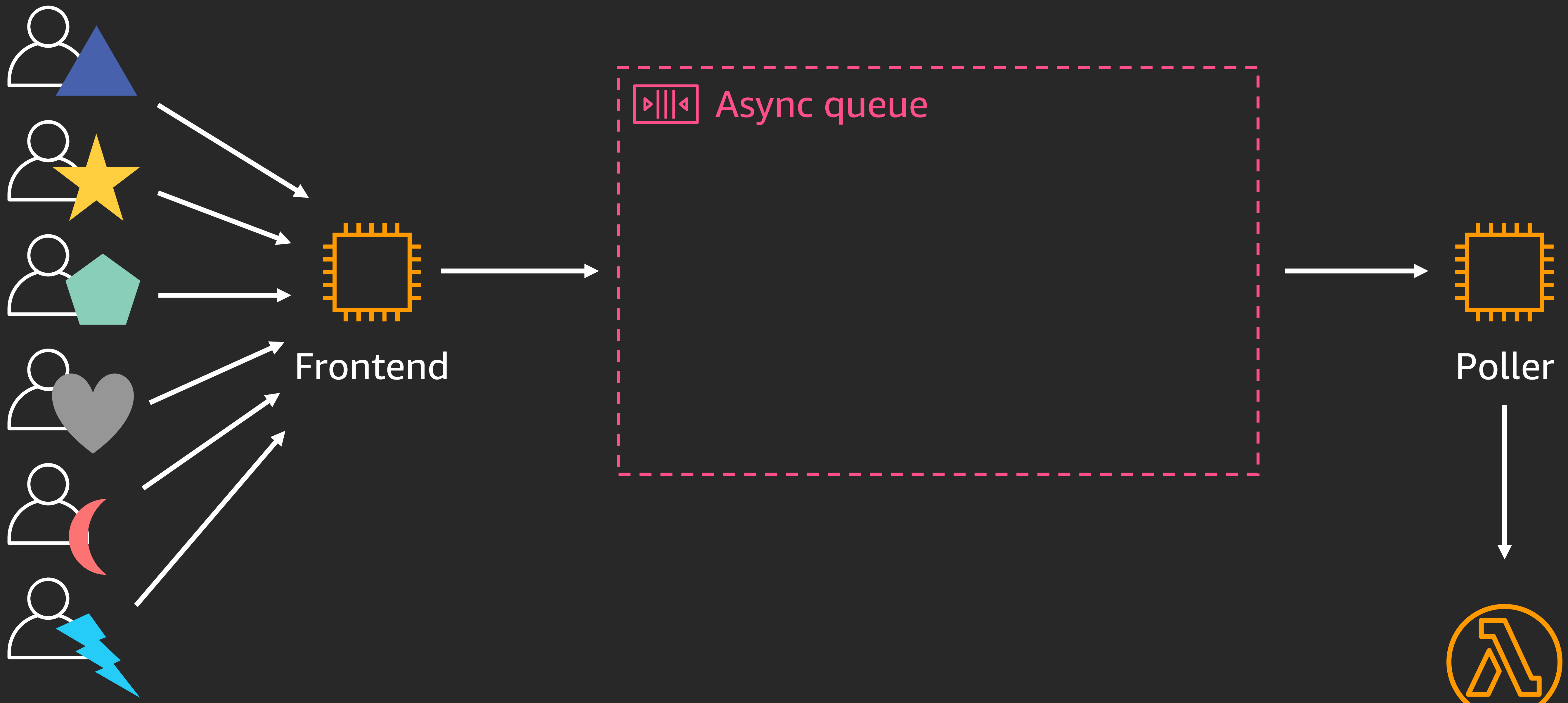
Frontend

Async queue

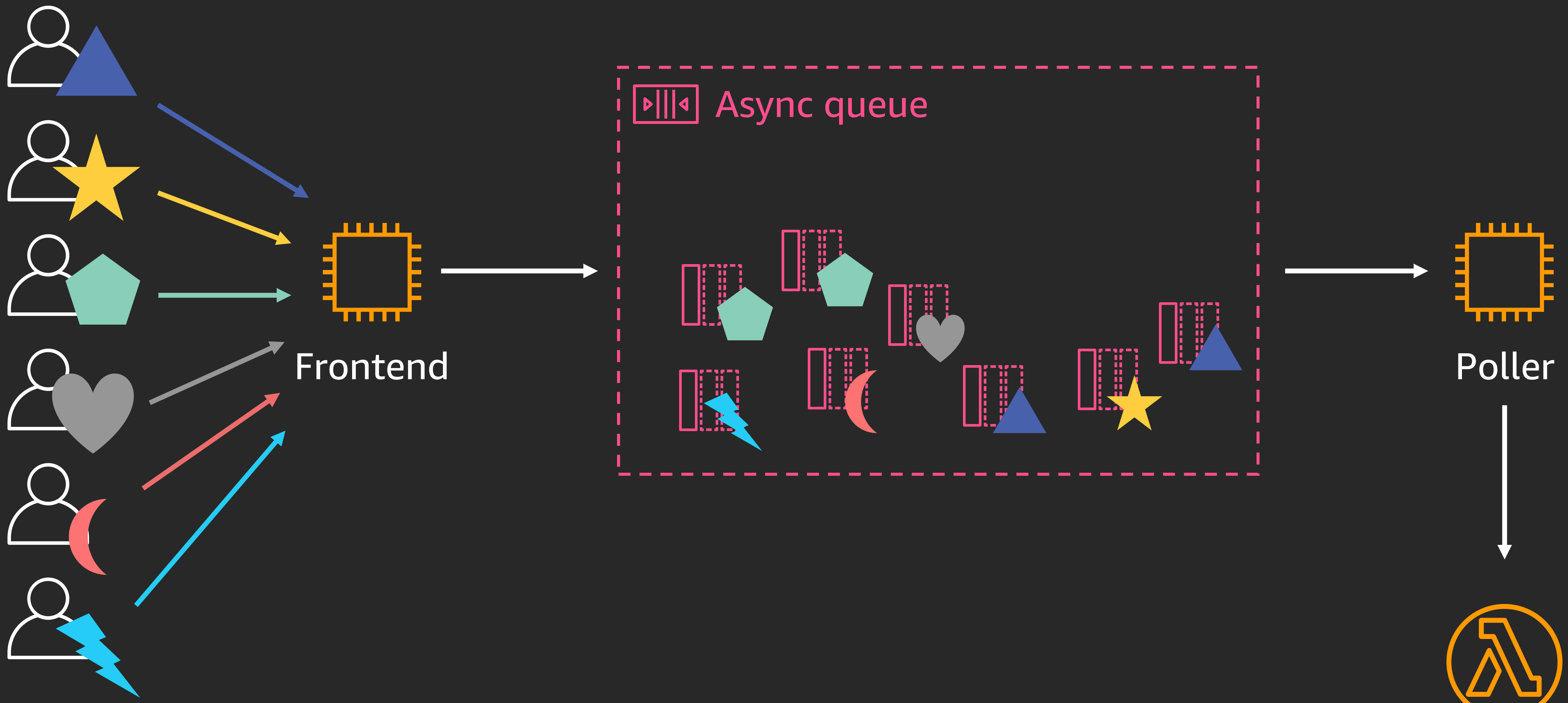
Poller

Availability Zone 2, etc

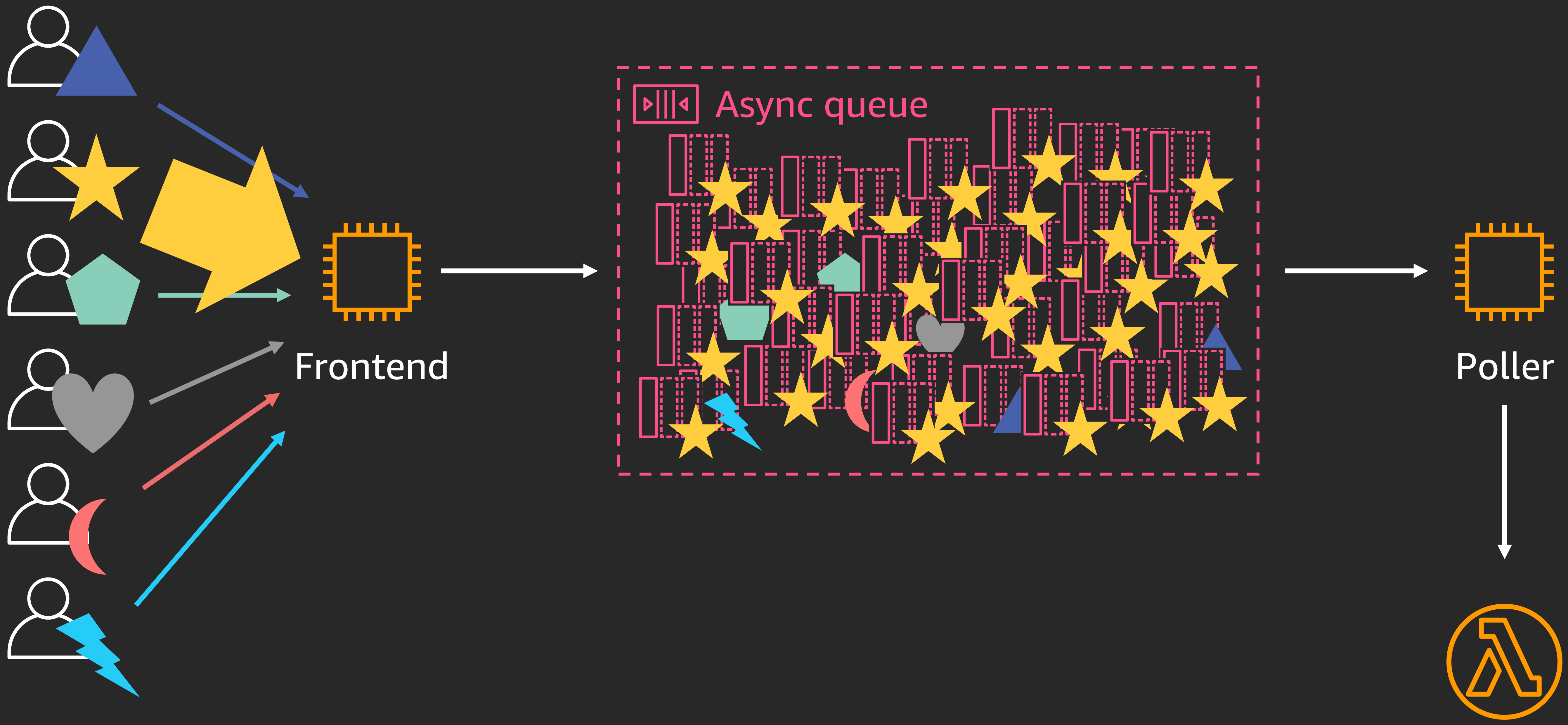
Lambda async



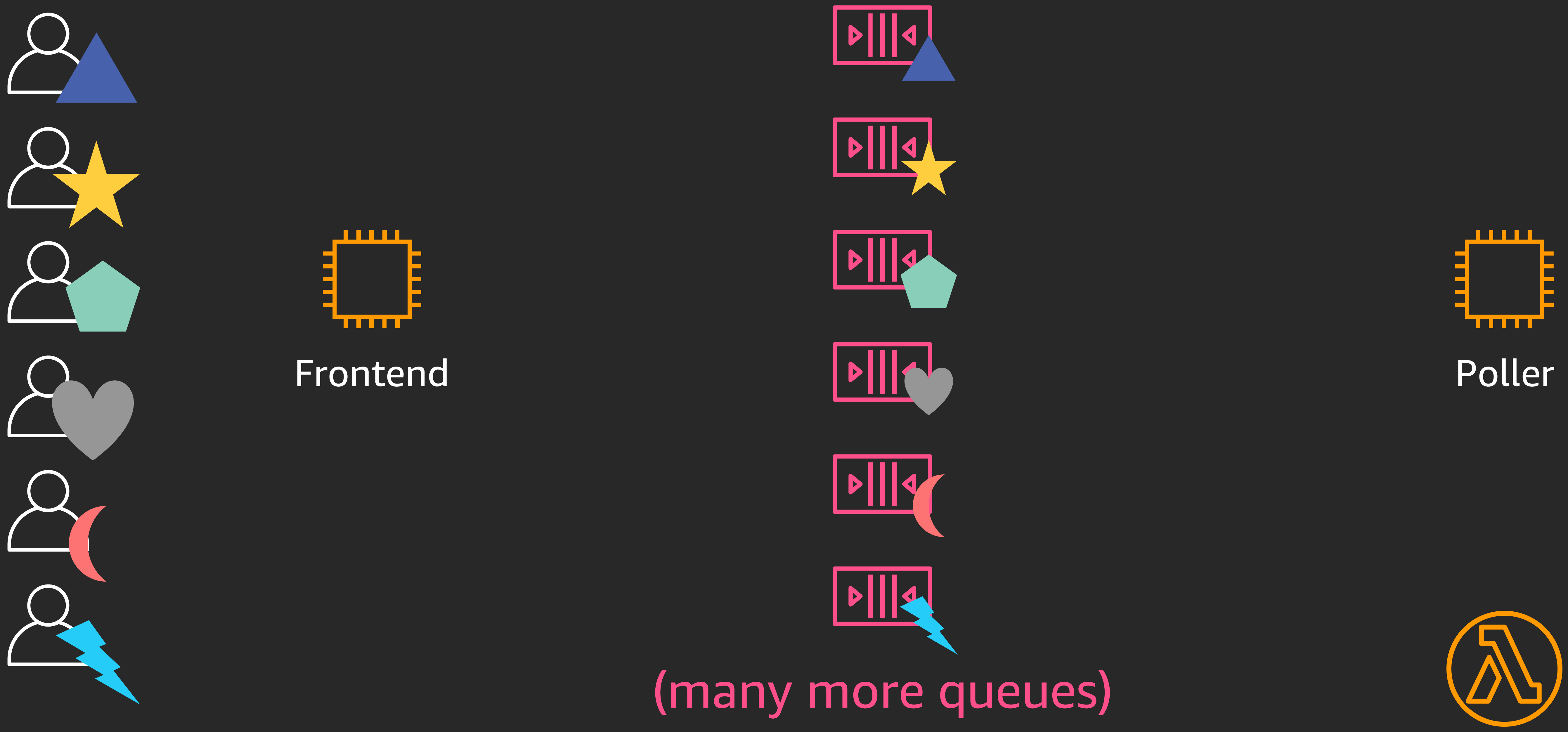
Lambda async



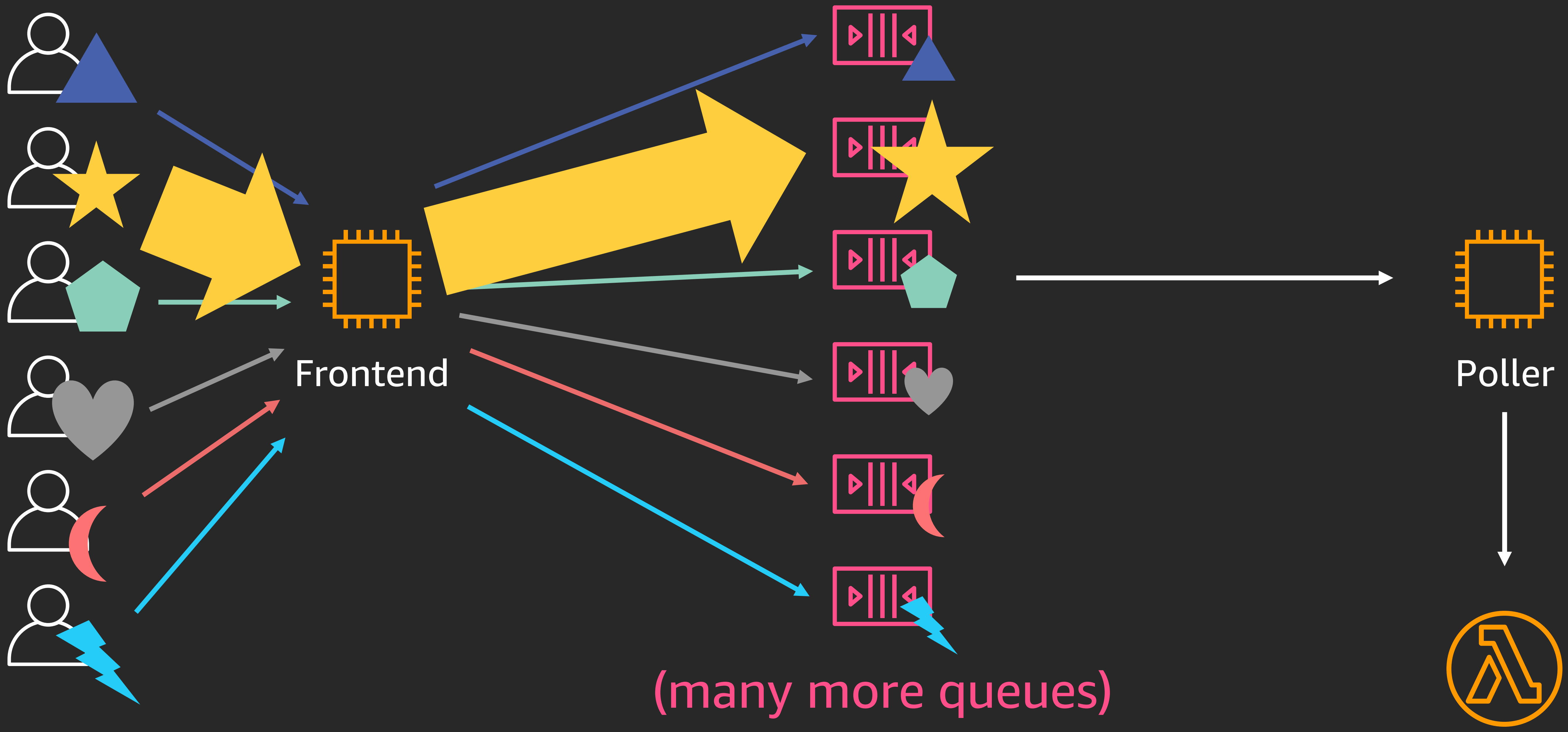
Lambda async (this is not what happens)



Queue per workload

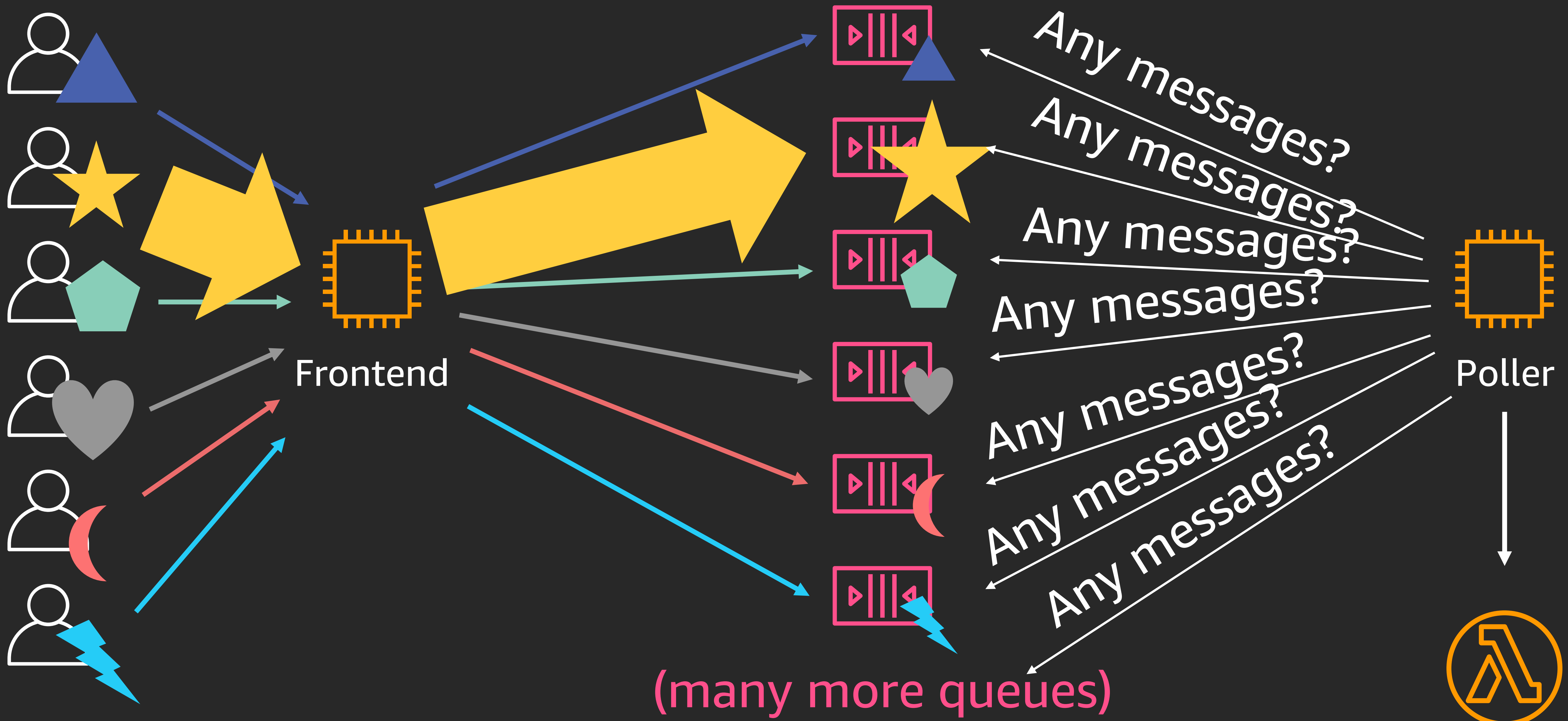


Queue per workload



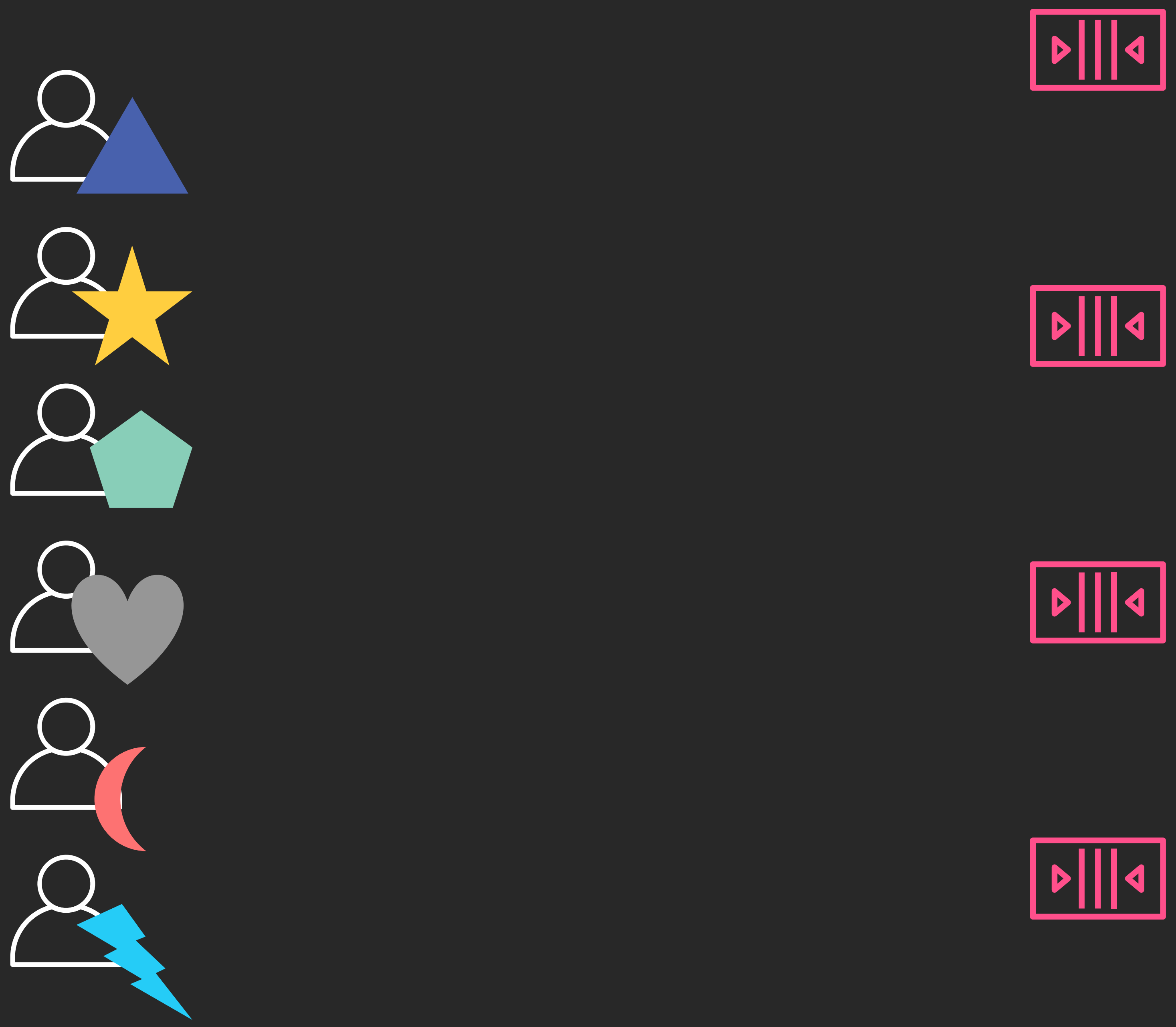
(many more queues)

Polling is not free



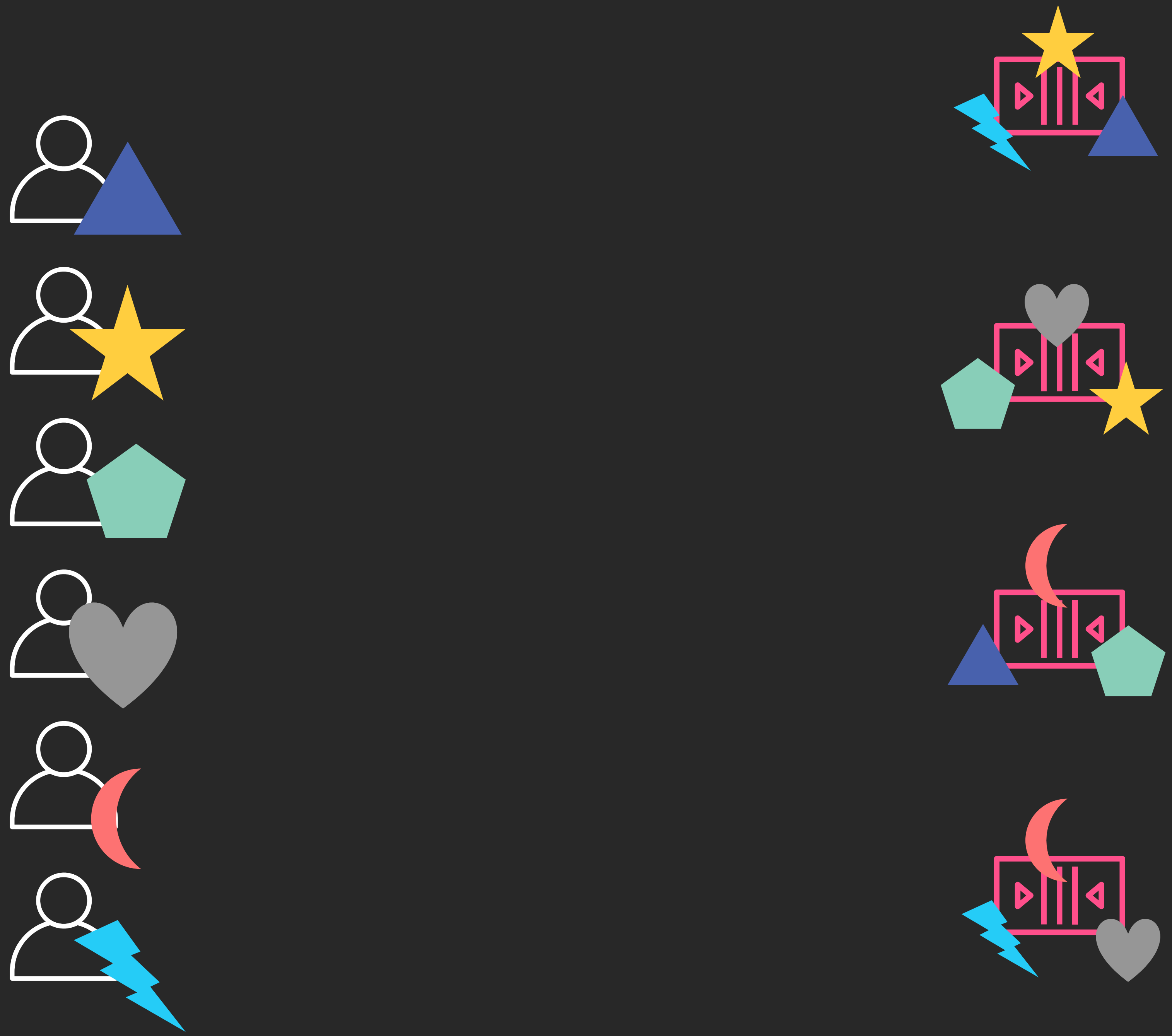
Shuffle-sharding

Shuffle-sharding



(fixed number of N queues)

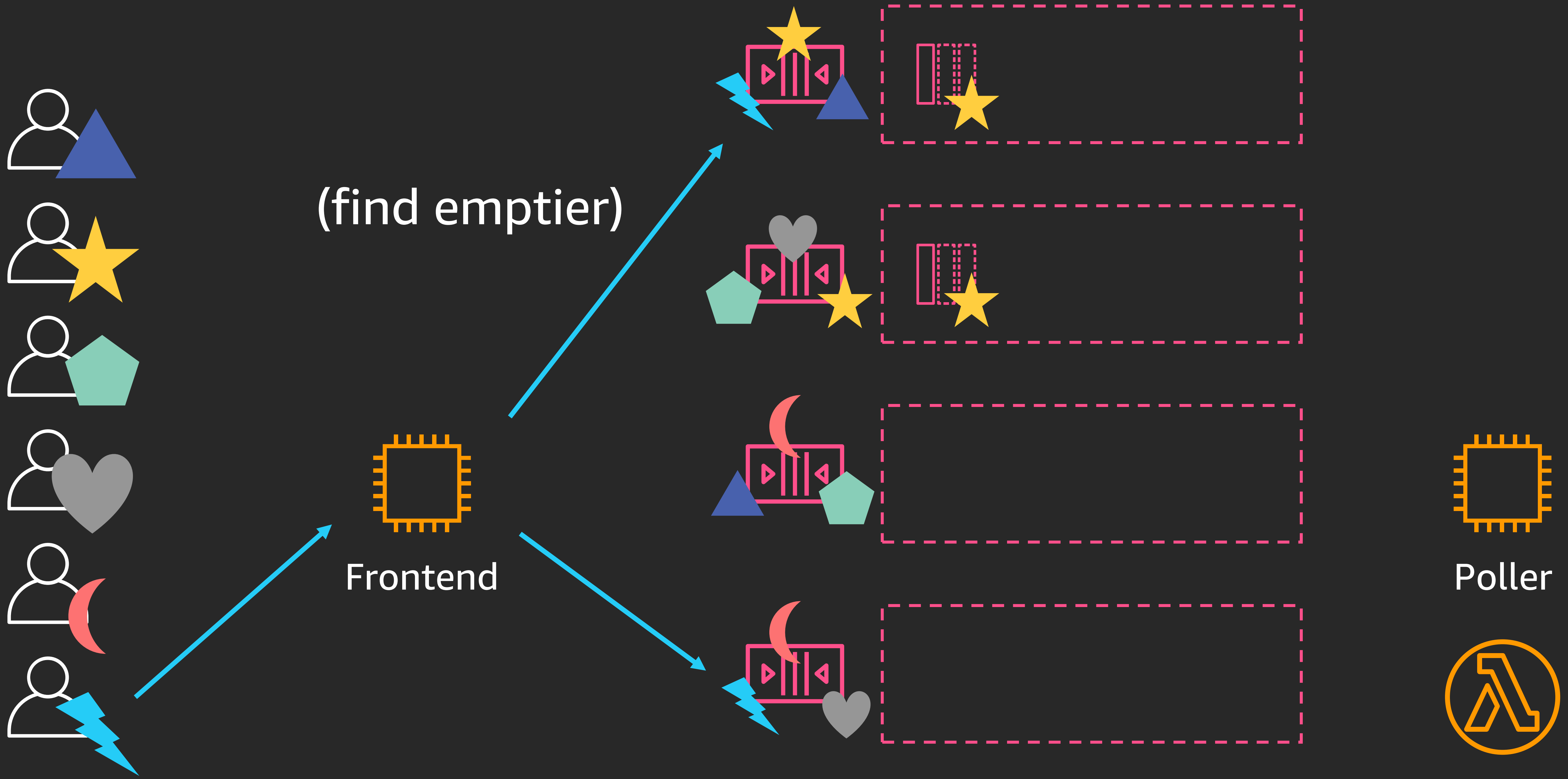
Shuffle-sharding



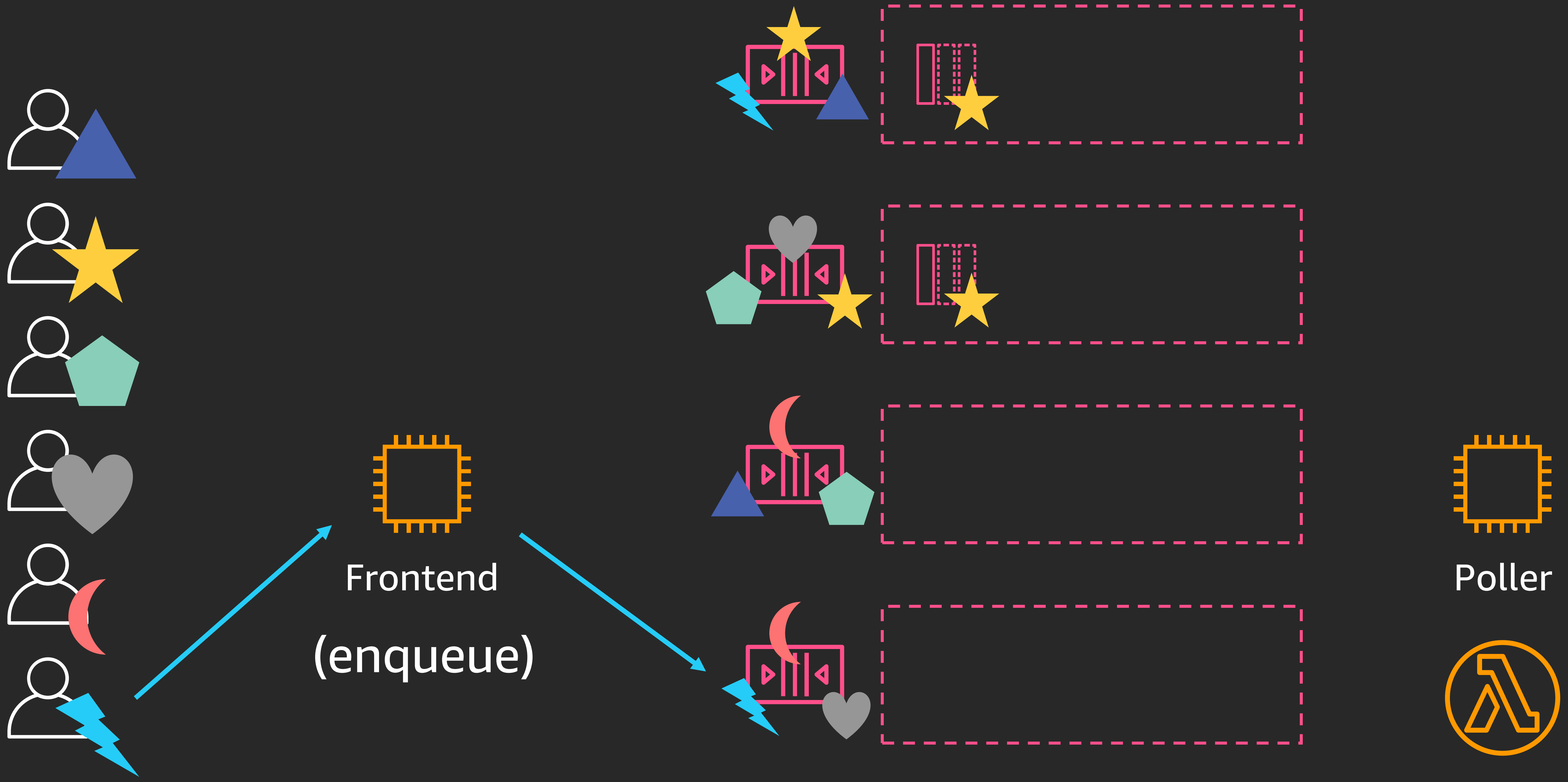
(fixed number of N queues)

(map each workload to K queues)

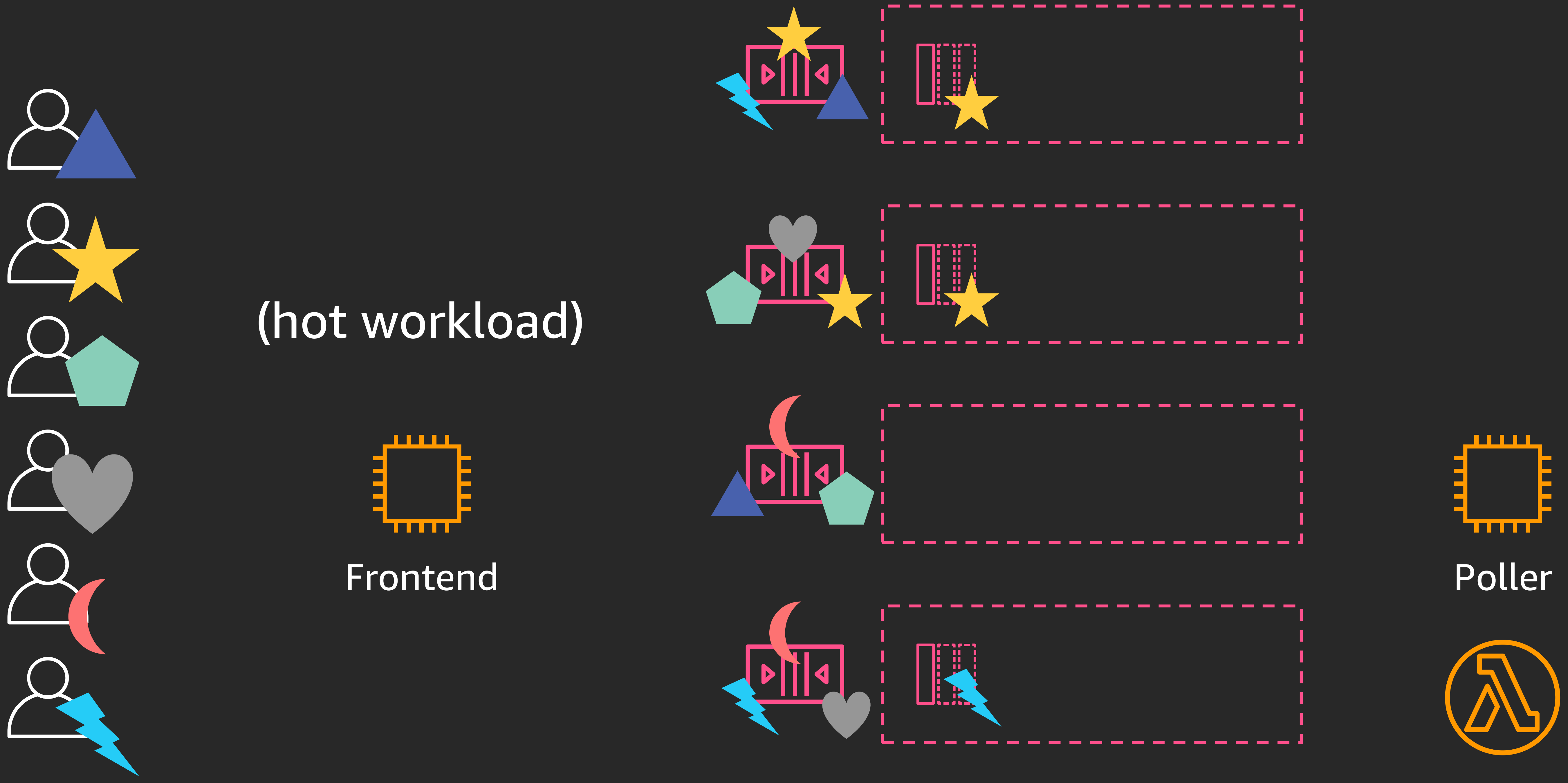
Shuffle-sharding: Enqueue



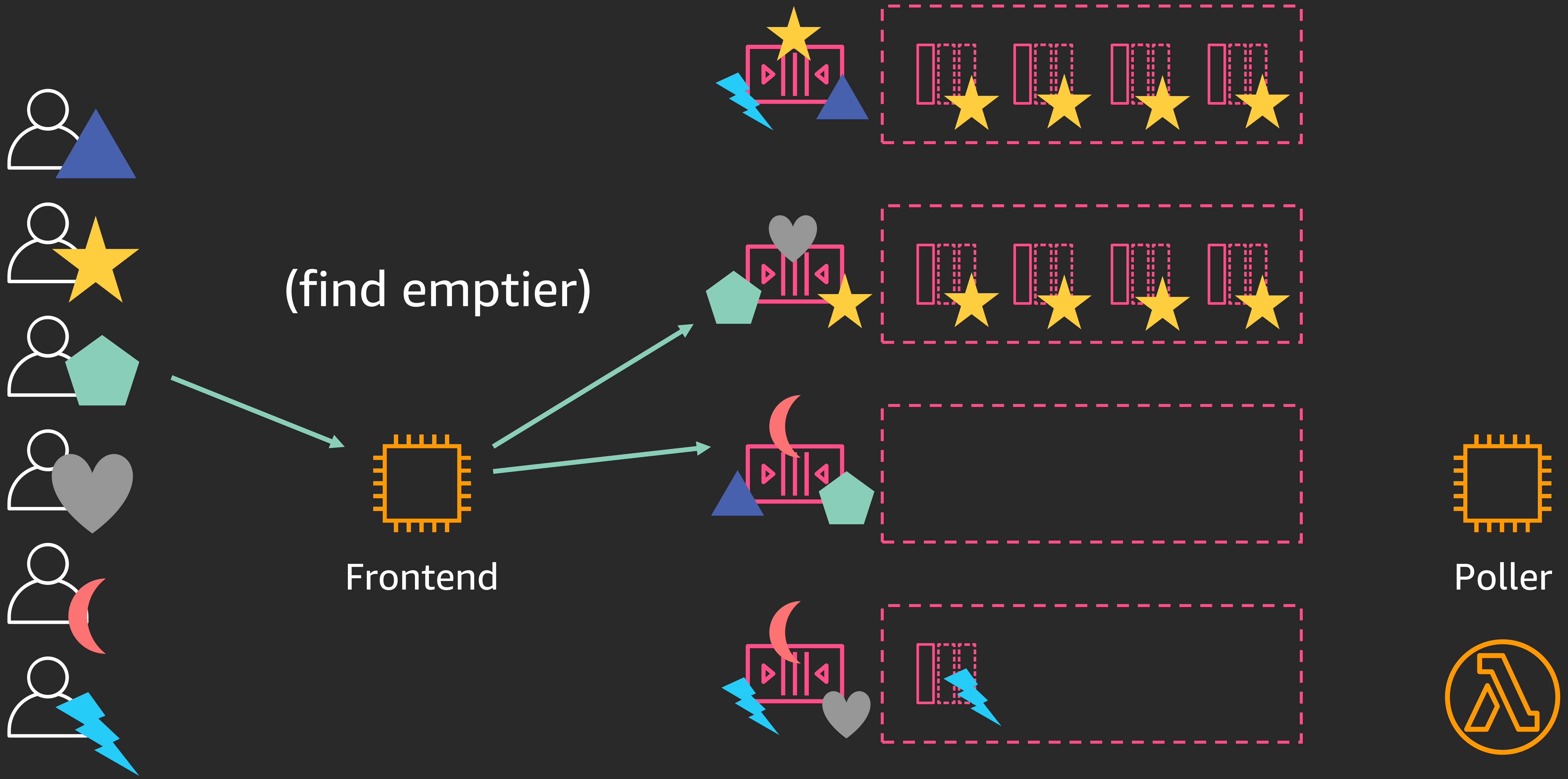
Shuffle-sharding: Enqueue



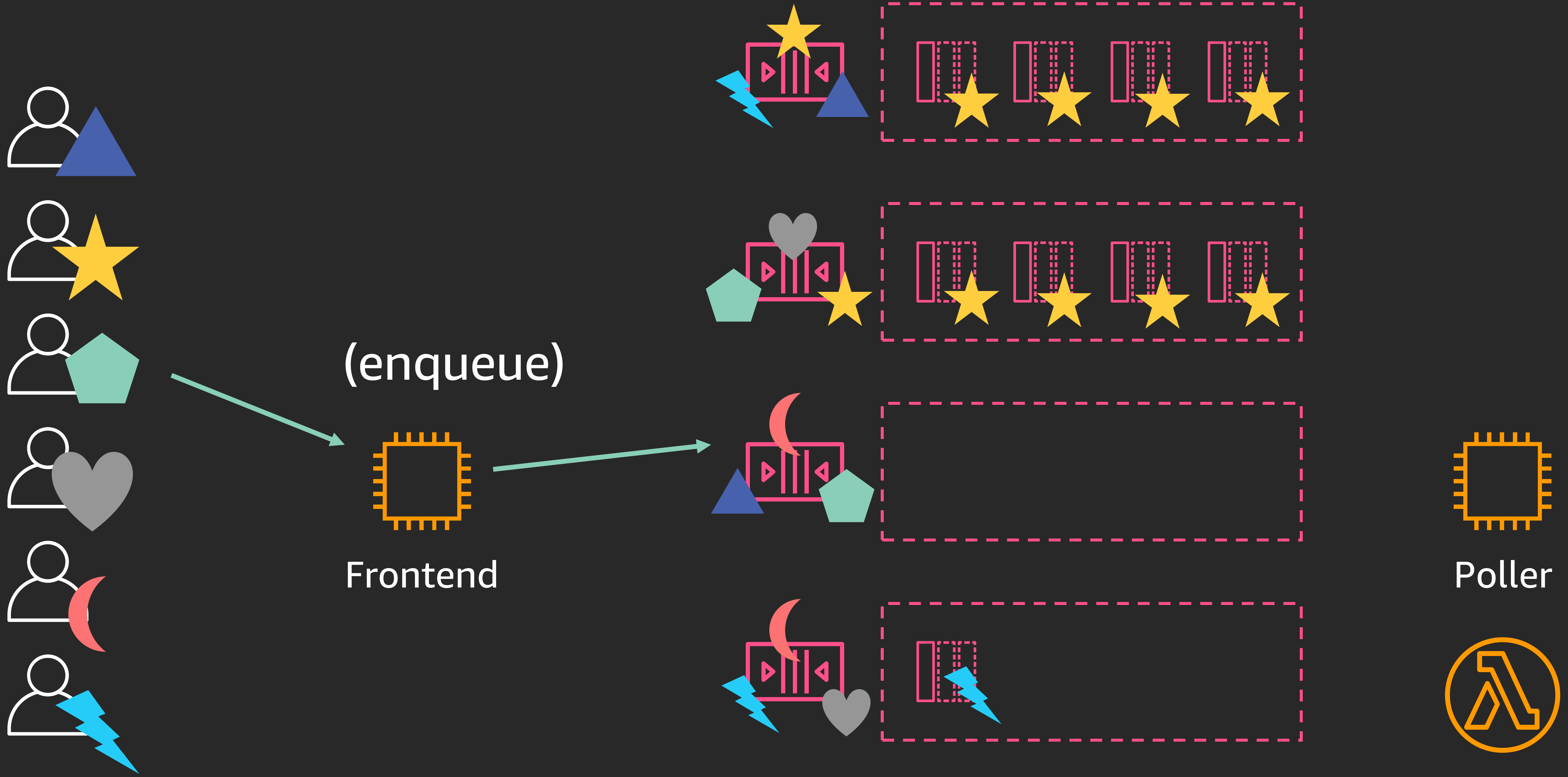
Shuffle-sharding: Busy workload



Shuffle-sharding: Busy workload



Shuffle-sharding: Busy workload



(enqueue)

Frontend

Poller

Shuffle-sharding: Magical resource isolation

$$\approx \frac{shardsize!}{|nodes|!}$$

Shuffle-sharding: Magical resource isolation

Nodes = 8
Shard size = 2

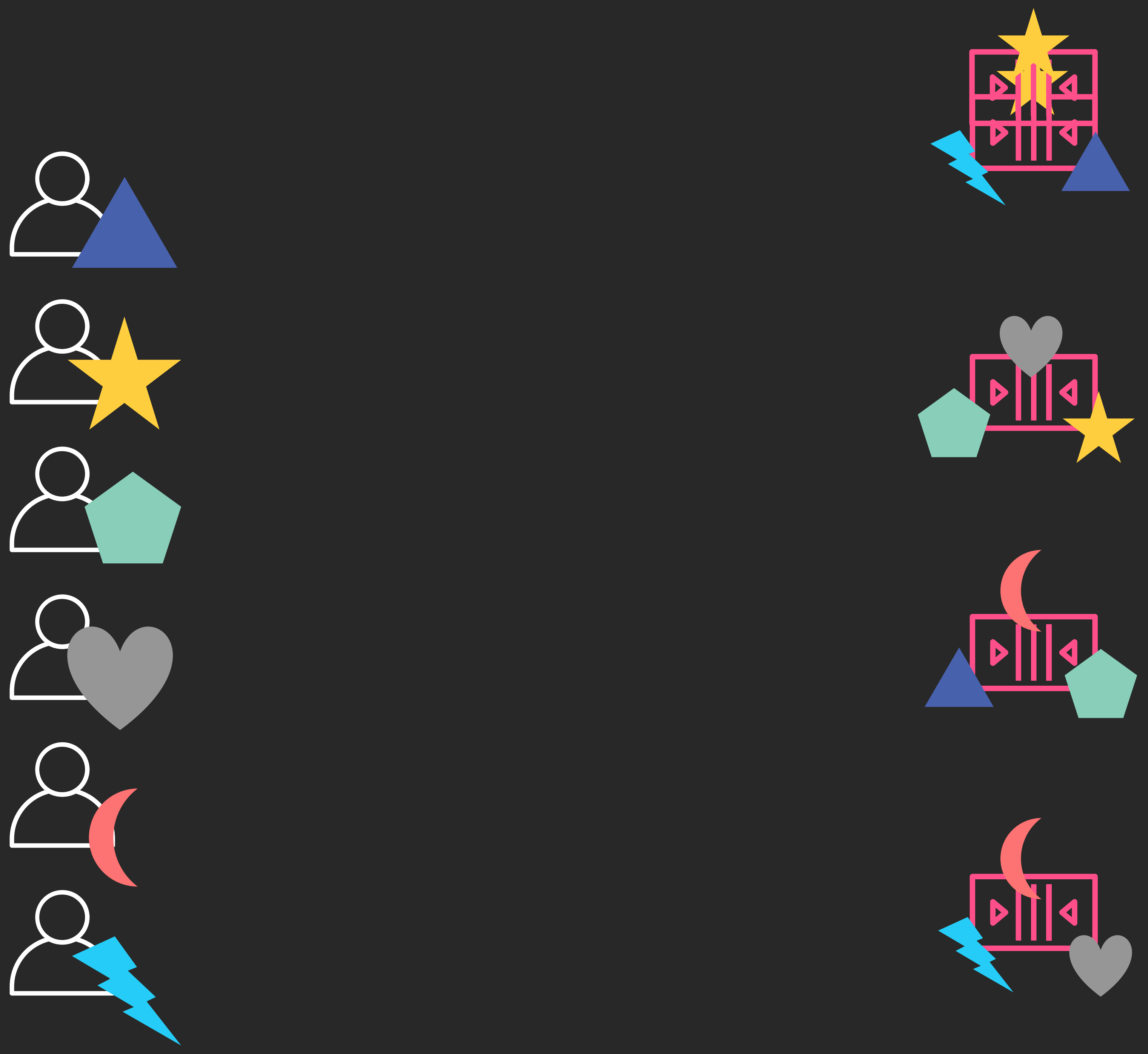
Overlap	% workloads
0	*53.6%*
1	*42.8%*
2	*3.6%*

Shuffle-sharding: Magical resource isolation

Nodes = 100
Shard size = 5

Overlap	% workloads
0	*77%*
1	*21%*
2	*1.8%*
3	*0.06%*
4	*0.0006%*
5	*0.0000013%*

Isolate busy workloads



(isolate busy workload)

Avoiding queue backlogs

- **Backlogs build quickly, introduce modes**

Avoiding queue backlogs

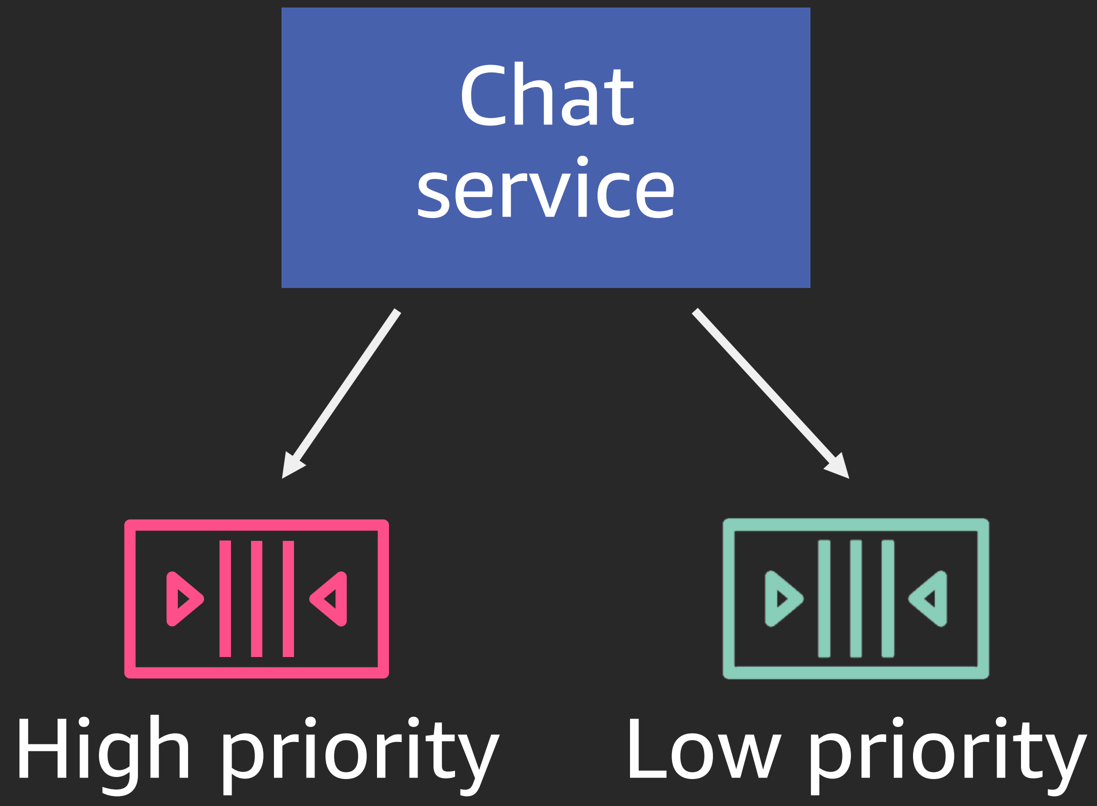
- Backlogs build quickly, introduce modes
- Auto Scaling and Lambda react quickly

Amazon EC2
Auto Scaling

AWS
Lambda

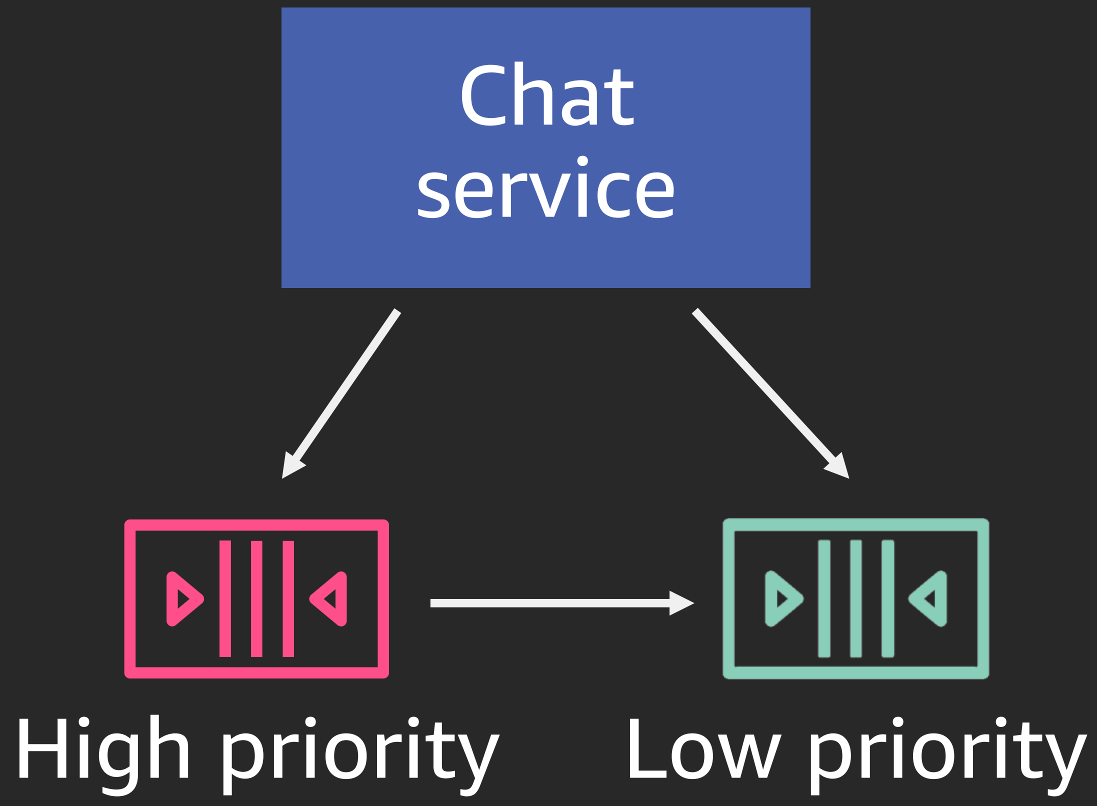
Avoiding queue backlogs

- Backlogs build quickly, introduce modes
- Auto Scaling and Lambda react quickly
- Priority queueing emulates LIFO



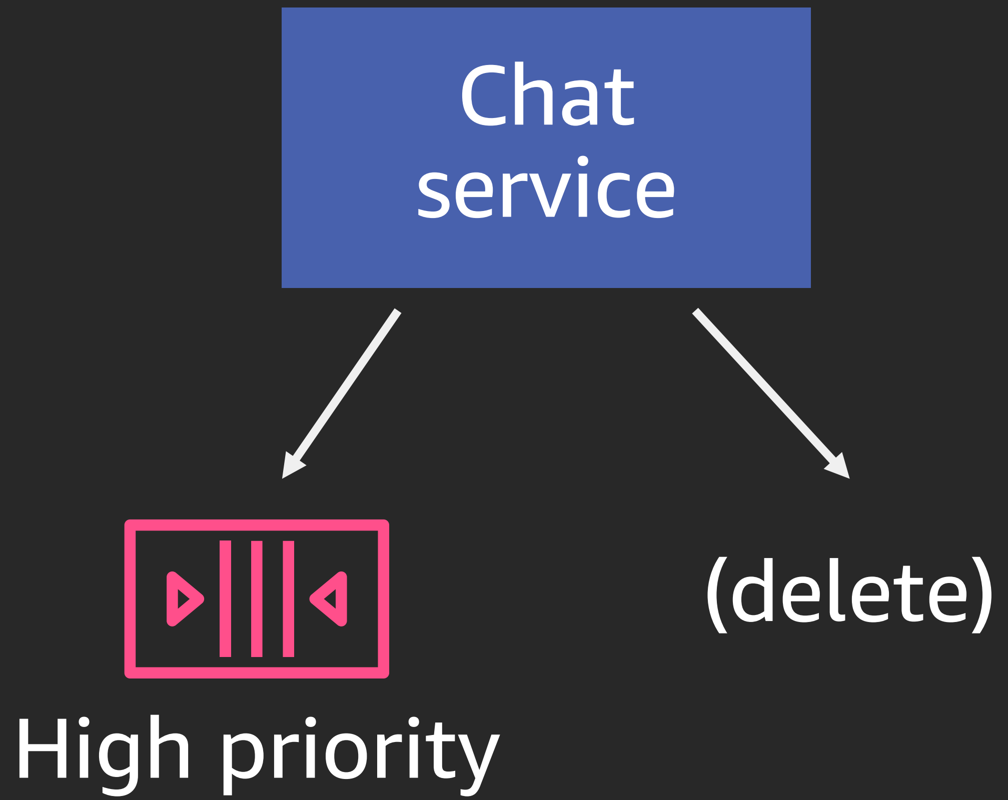
Avoiding queue backlogs

- Backlogs build quickly, introduce modes
- Auto Scaling and Lambda react quickly
- Priority queueing emulates LIFO
- Move old messages to low priority



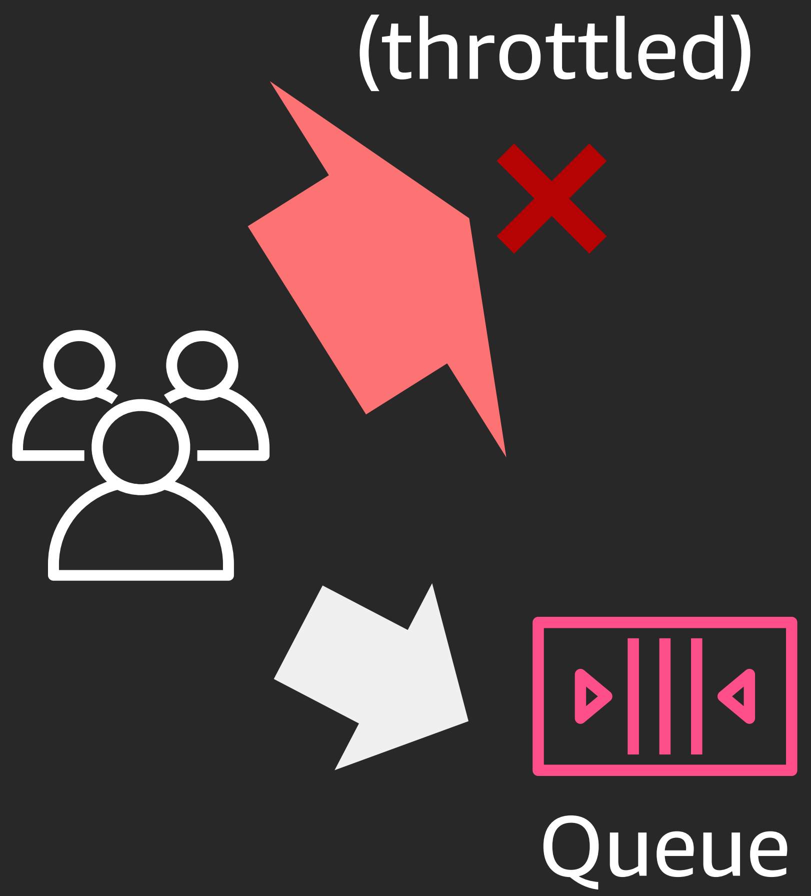
Avoiding queue backlogs

- Backlogs build quickly, introduce modes
- Auto Scaling and Lambda react quickly
- Priority queueing emulates LIFO
- Move old messages to low priority
- Message TTLs for stale information



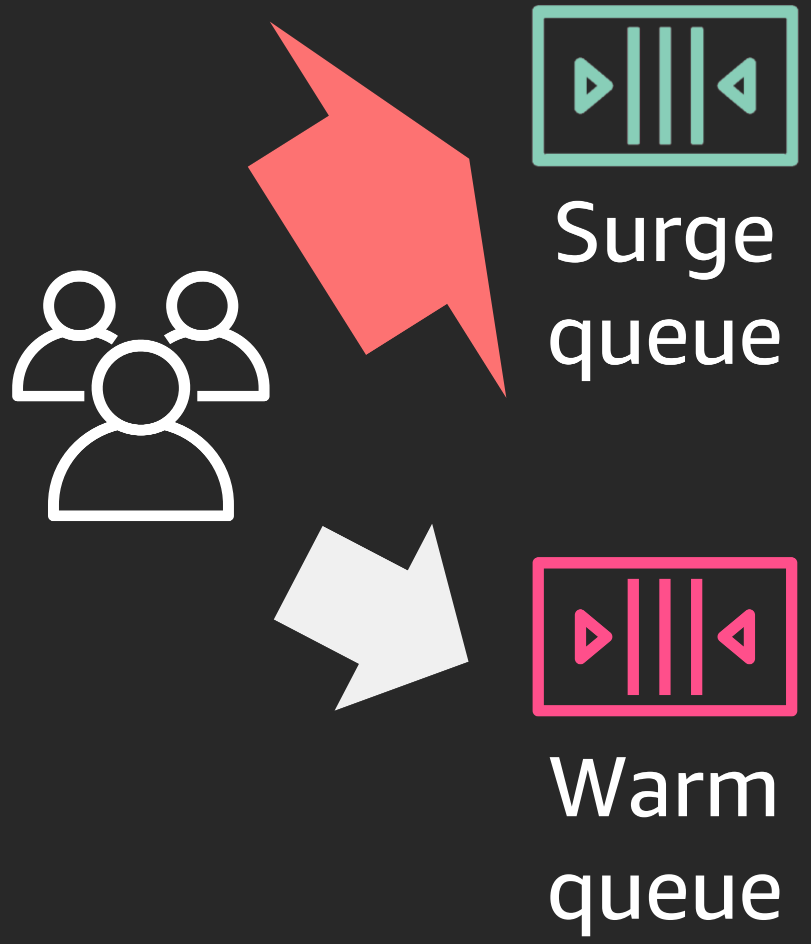
Avoiding queue backlogs

- Backlogs build quickly, introduce modes
- Auto Scaling and Lambda react quickly
- Priority queueing emulates LIFO
- Move old messages to low priority
- Message TTLs for stale information
- Apply backpressure



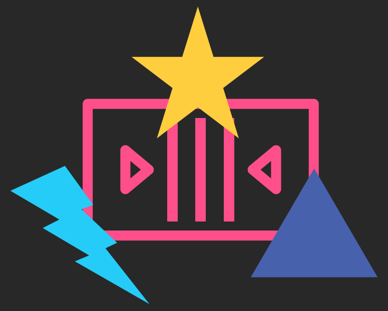
Avoiding queue backlogs

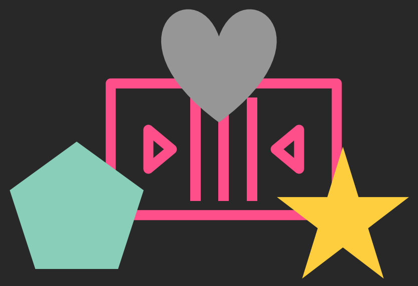
- Backlogs build quickly, introduce modes
- Auto Scaling and Lambda react quickly
- Priority queueing emulates LIFO
- Move old messages to low priority
- Message TTLs for stale information
- Apply backpressure
- Surge queue excess traffic



Avoiding queue backlogs

- Backlogs build quickly, introduce modes
- Auto Scaling and Lambda react quickly
- Priority queueing emulates LIFO
- Move old messages to low priority
- Message TTLs for stale information
- Apply backpressure
- Surge queue excess traffic
- Shuffle-sharding for isolation

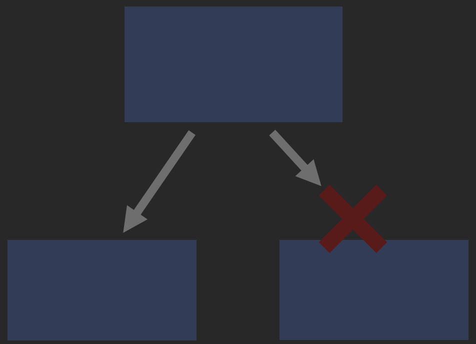




Chapter four

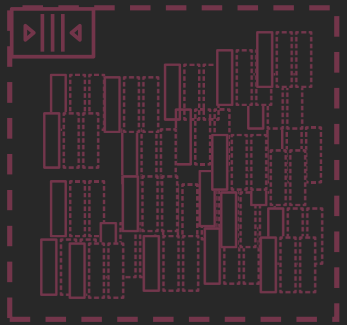
Load shedding

Avoid brownout by rejecting excess load



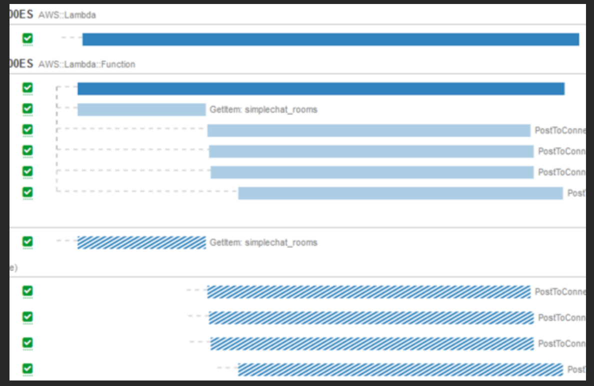
Dependency isolation

Prevent one dependency from affecting unrelated functionality



Avoiding queue backlogs

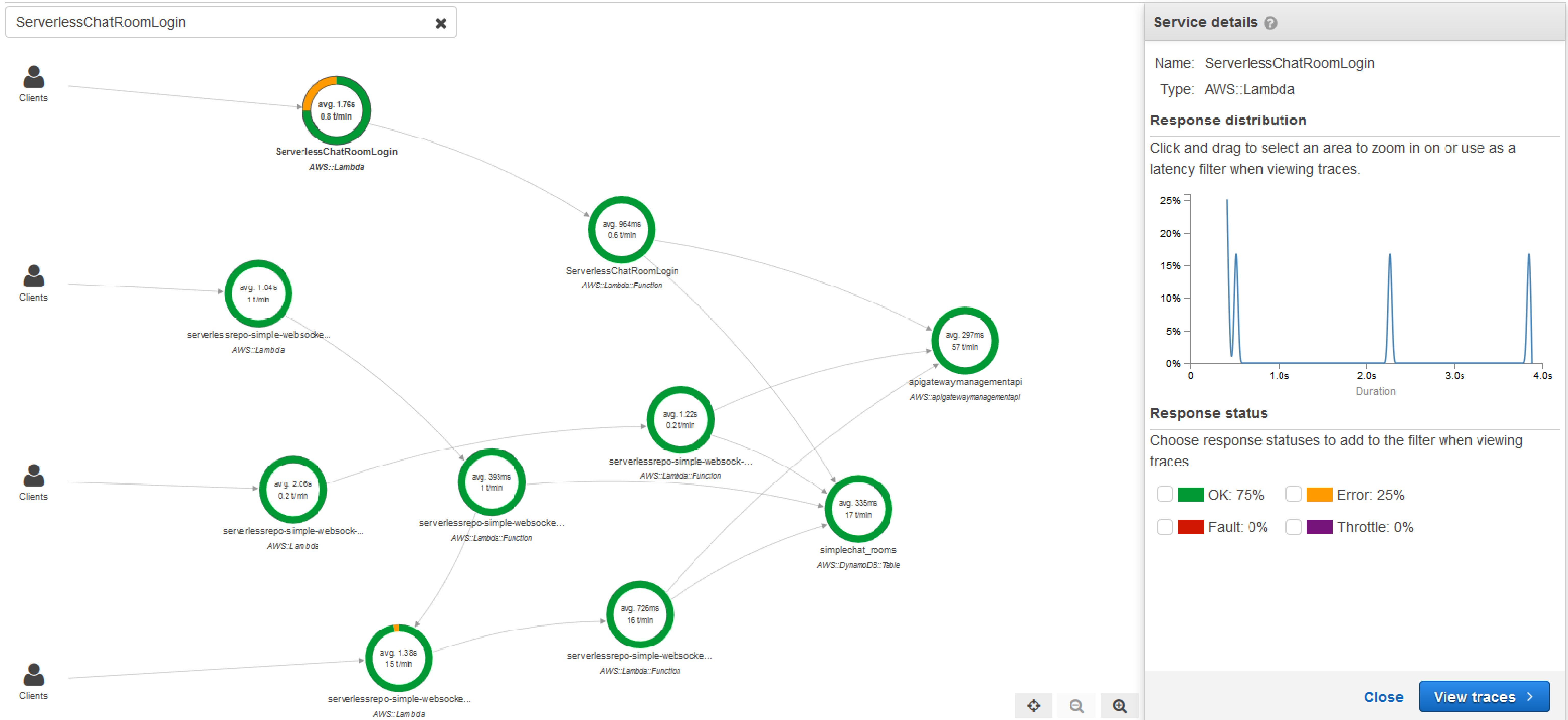
Prevent a backlog from extending recovery time



Operating

Quickly diagnose and mitigate issues

AWS X-Ray tracing



Service details

Name: ServerlessChatRoomLogin

Type: AWS::Lambda

Response distribution

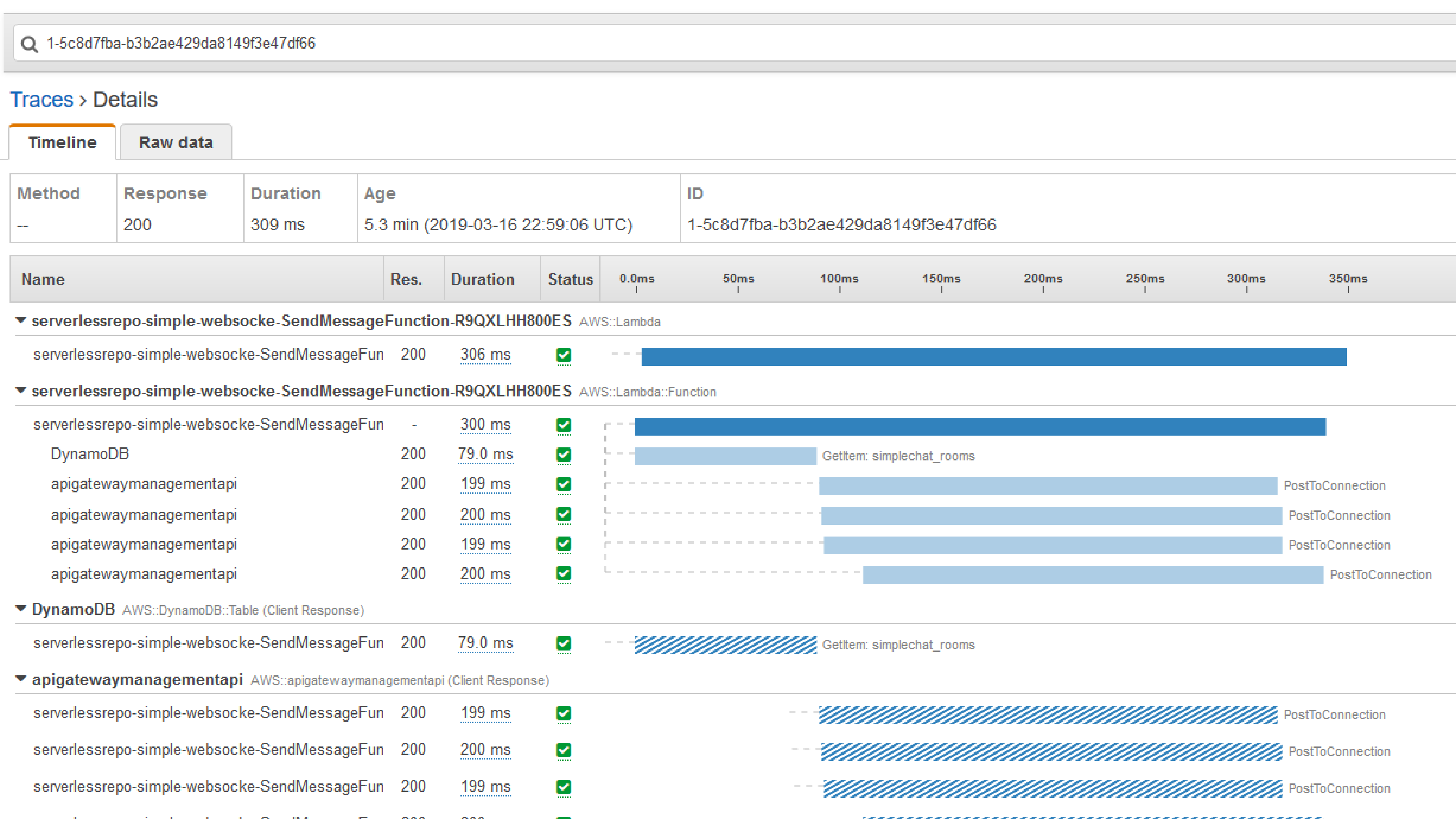
Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.

Response status

Choose response statuses to add to the filter when viewing traces.

☐ 🟩 OK: 75% ☐ 🟧 Error: 25%

☐ 🟥 Fault: 0% ☐ 🟪 Throttle: 0%

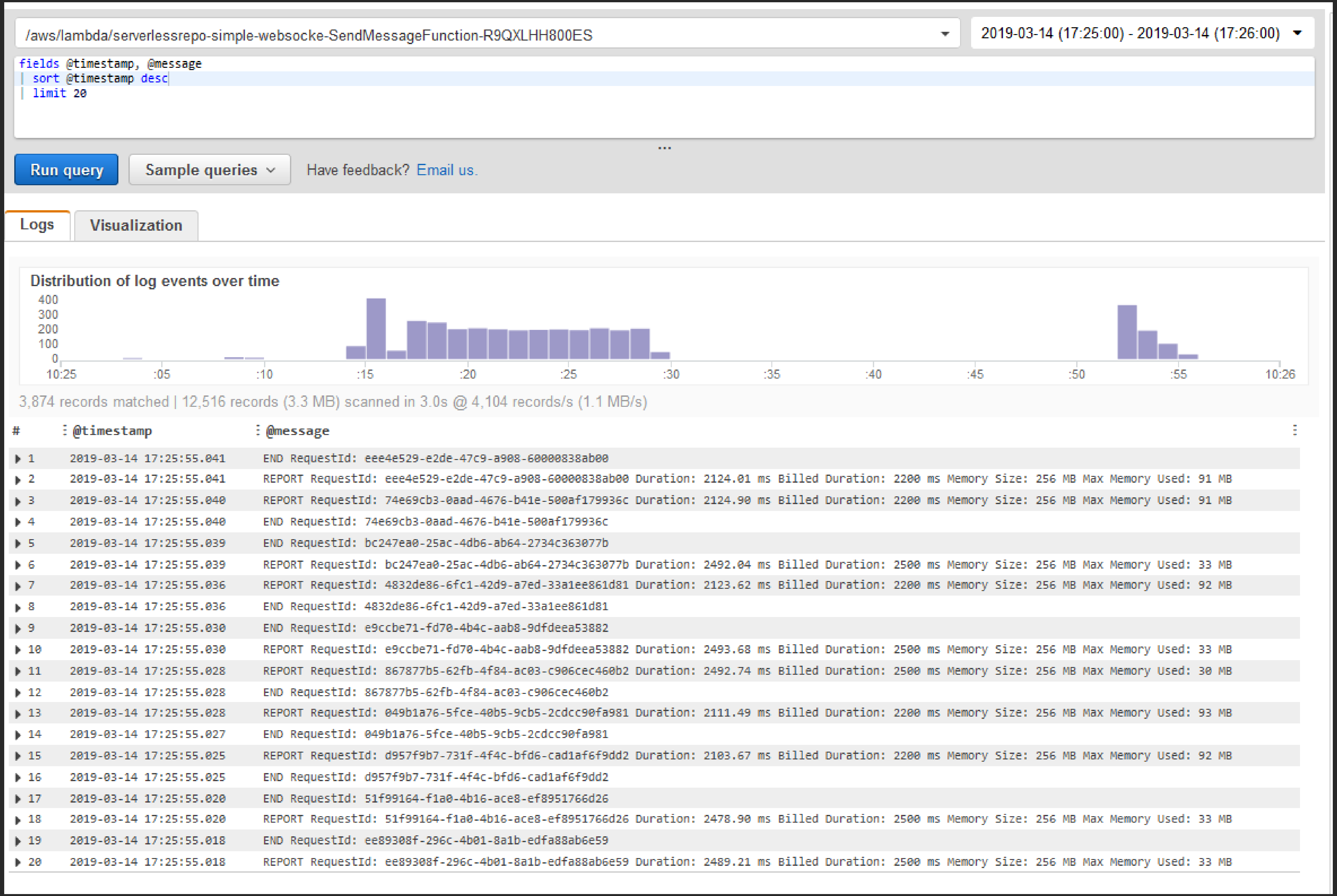


Traces › Details

Method	Response	Duration	Age	ID
--	200	309 ms	5.3 min (2019-03-16 22:59:06 UTC)	1-5c8d7fba-b3b2ae429da8149f3e47df66

Name	Res.	Duration	Status	0.0ms 50ms 100ms 150ms 200ms 250ms 300ms 350ms
serverlessrepo-simple-websocke-SendMessageFunction-R9QXLHH800ES AWS::Lambda				
serverlessrepo-simple-websocke-SendMessageFun	200	306 ms	☑	
serverlessrepo-simple-websocke-SendMessageFunction-R9QXLHH800ES AWS::Lambda::Function				
serverlessrepo-simple-websocke-SendMessageFun	-	300 ms	☑	
DynamoDB	200	79.0 ms	☑	GetItem: simplechat_rooms
apigatewaymanagementapi	200	199 ms	☑	PostToConnection
apigatewaymanagementapi	200	200 ms	☑	PostToConnection
apigatewaymanagementapi	200	199 ms	☑	PostToConnection
apigatewaymanagementapi	200	200 ms	☑	PostToConnection
DynamoDB AWS::DynamoDB::Table (Client Response)				
serverlessrepo-simple-websocke-SendMessageFun	200	79.0 ms	☑	GetItem: simplechat_rooms
apigatewaymanagementapi AWS::apigatewaymanagementapi (Client Response)				
serverlessrepo-simple-websocke-SendMessageFun	200	199 ms	☑	PostToConnection
serverlessrepo-simple-websocke-SendMessageFun	200	200 ms	☑	PostToConnection
serverlessrepo-simple-websocke-SendMessageFun	200	199 ms	☑	PostToConnection

Amazon CloudWatch insights



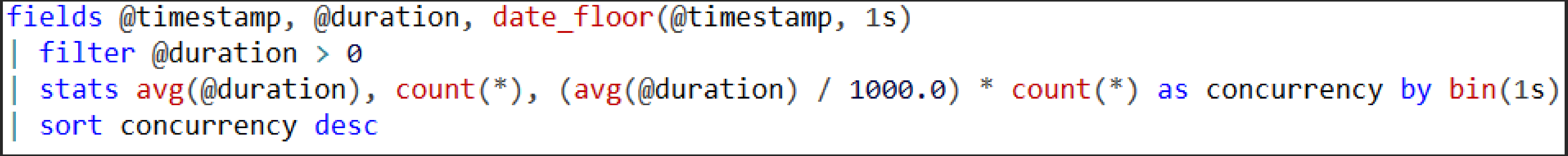
/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9QXLHH800ES
2019-03-14 (17:25:00) - 2019-03-14 (17:26:00)

```
fields @timestamp, @message
| sort @timestamp desc
| limit 20
```

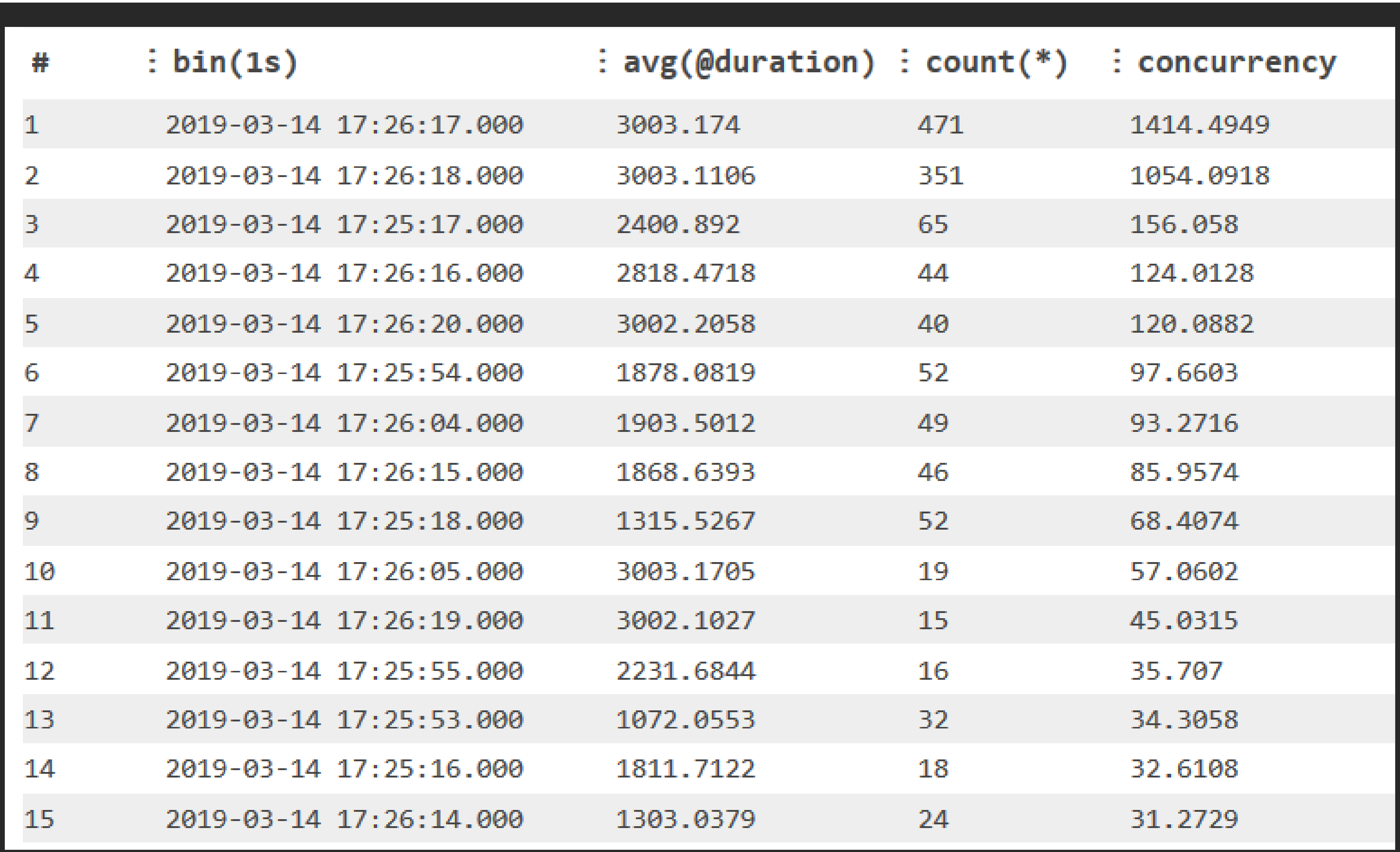
3,874 records matched | 12,516 records (3.3 MB) scanned in 3.0s @ 4,104 records/s (1.1 MB/s)

```
#     @timestamp               @message
1     2019-03-14 17:25:55.041  END RequestId: eee4e529-e2de-47c9-a908-60000838ab00
2     2019-03-14 17:25:55.041  REPORT RequestId: eee4e529-e2de-47c9-a908-60000838ab00 Duration: 2124.01 ms Billed Duration: 2200 ms Memory Size: 256 MB Max Memory Used: 91 MB
3     2019-03-14 17:25:55.040  REPORT RequestId: 74e69cb3-0aad-4676-b41e-500af179936c Duration: 2124.90 ms Billed Duration: 2200 ms Memory Size: 256 MB Max Memory Used: 91 MB
4     2019-03-14 17:25:55.040  END RequestId: 74e69cb3-0aad-4676-b41e-500af179936c
5     2019-03-14 17:25:55.039  END RequestId: bc247ea0-25ac-4db6-ab64-2734c363077b
6     2019-03-14 17:25:55.039  REPORT RequestId: bc247ea0-25ac-4db6-ab64-2734c363077b Duration: 2492.04 ms Billed Duration: 2500 ms Memory Size: 256 MB Max Memory Used: 33 MB
7     2019-03-14 17:25:55.036  REPORT RequestId: 4832de86-6fc1-42d9-a7ed-33a1ee861d81 Duration: 2123.62 ms Billed Duration: 2200 ms Memory Size: 256 MB Max Memory Used: 92 MB
8     2019-03-14 17:25:55.036  END RequestId: 4832de86-6fc1-42d9-a7ed-33a1ee861d81
9     2019-03-14 17:25:55.030  END RequestId: e9ccbe71-fd70-4b4c-aab8-9dfdeea53882
10    2019-03-14 17:25:55.030  REPORT RequestId: e9ccbe71-fd70-4b4c-aab8-9dfdeea53882 Duration: 2493.68 ms Billed Duration: 2500 ms Memory Size: 256 MB Max Memory Used: 33 MB
11    2019-03-14 17:25:55.028  REPORT RequestId: 867877b5-62fb-4f84-ac03-c906cec460b2 Duration: 2492.74 ms Billed Duration: 2500 ms Memory Size: 256 MB Max Memory Used: 30 MB
12    2019-03-14 17:25:55.028  END RequestId: 867877b5-62fb-4f84-ac03-c906cec460b2
13    2019-03-14 17:25:55.028  REPORT RequestId: 049b1a76-5fce-40b5-9cb5-2cdcc90fa981 Duration: 2111.49 ms Billed Duration: 2200 ms Memory Size: 256 MB Max Memory Used: 93 MB
14    2019-03-14 17:25:55.027  END RequestId: 049b1a76-5fce-40b5-9cb5-2cdcc90fa981
15    2019-03-14 17:25:55.025  REPORT RequestId: d957f9b7-731f-4f4c-bfd6-cad1af6f9dd2 Duration: 2103.67 ms Billed Duration: 2200 ms Memory Size: 256 MB Max Memory Used: 92 MB
16    2019-03-14 17:25:55.025  END RequestId: d957f9b7-731f-4f4c-bfd6-cad1af6f9dd2
17    2019-03-14 17:25:55.020  END RequestId: 51f99164-f1a0-4b16-ace8-ef8951766d26
18    2019-03-14 17:25:55.020  REPORT RequestId: 51f99164-f1a0-4b16-ace8-ef8951766d26 Duration: 2478.90 ms Billed Duration: 2500 ms Memory Size: 256 MB Max Memory Used: 33 MB
19    2019-03-14 17:25:55.018  END RequestId: ee89308f-296c-4b01-8a1b-edfa88ab6e59
20    2019-03-14 17:25:55.018  REPORT RequestId: ee89308f-296c-4b01-8a1b-edfa88ab6e59 Duration: 2489.21 ms Billed Duration: 2500 ms Memory Size: 256 MB Max Memory Used: 33 MB
```

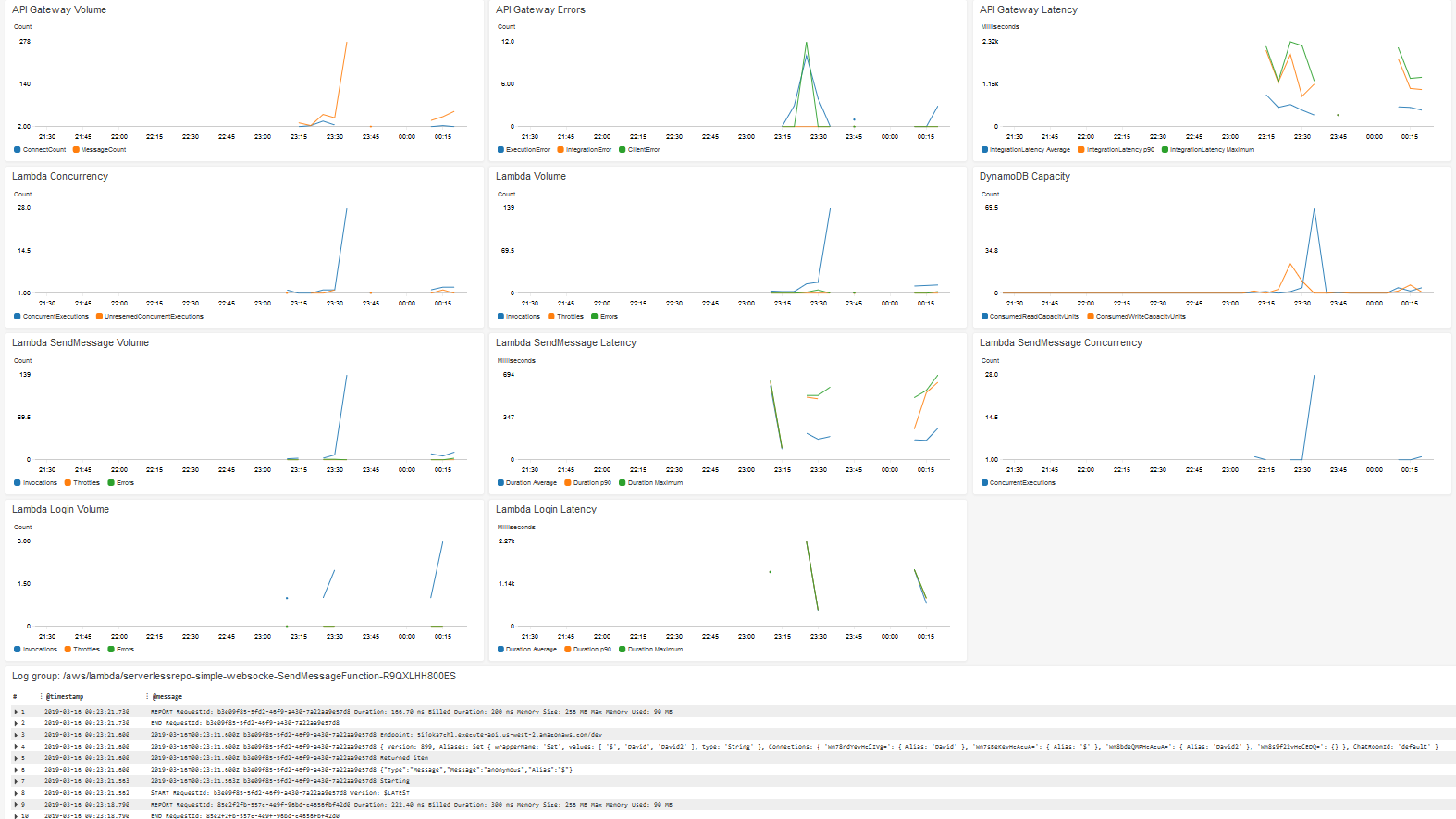
Amazon CloudWatch insights



```
fields @timestamp, @duration, date_floor(@timestamp, 1s)
| filter @duration > 0
| stats avg(@duration), count(*), (avg(@duration) / 1000.0) * count(*) as concurrency by bin(1s)
| sort concurrency desc
```



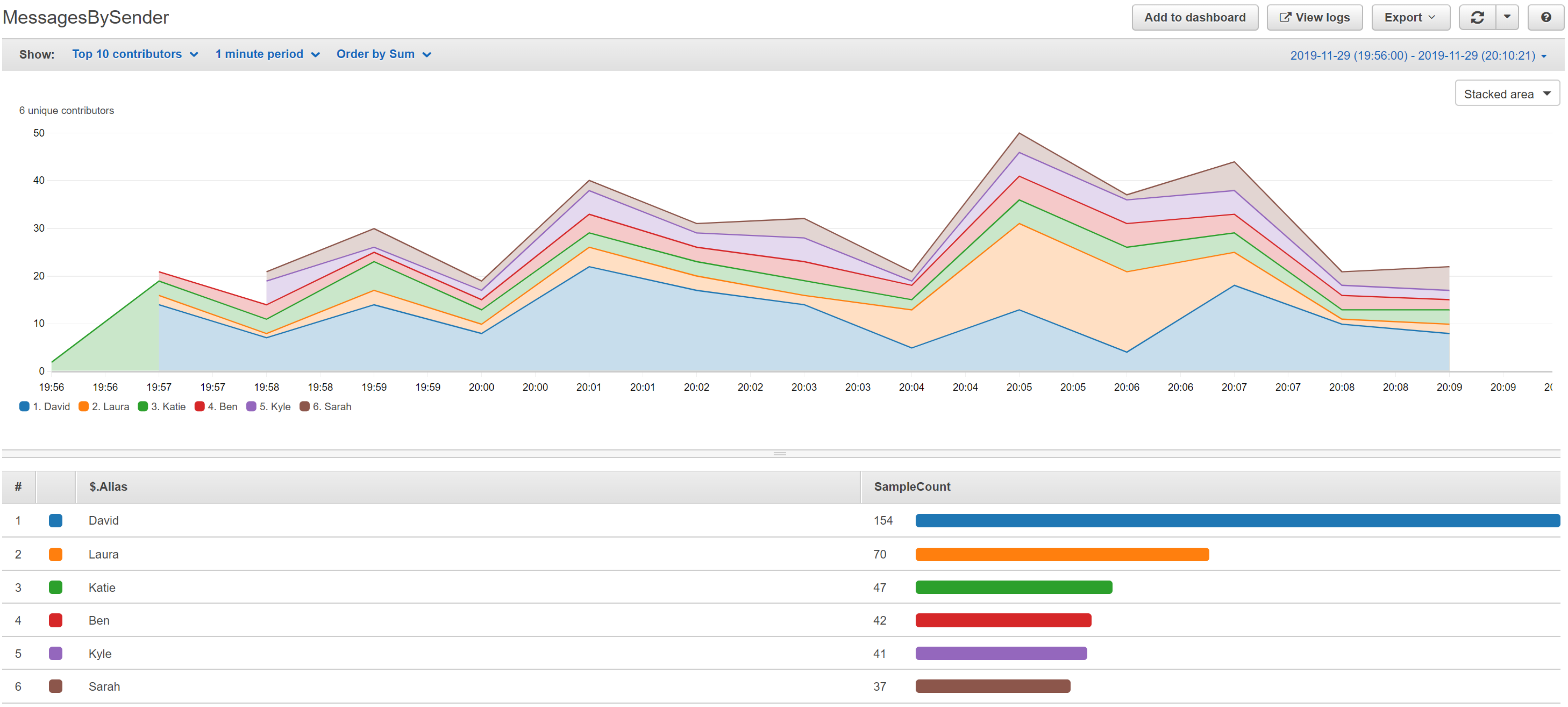
#	bin(1s)	avg(@duration)	count(*)	concurrency
1	2019-03-14 17:26:17.000	3003.174	471	1414.4949
2	2019-03-14 17:26:18.000	3003.1106	351	1054.0918
3	2019-03-14 17:25:17.000	2400.892	65	156.058
4	2019-03-14 17:26:16.000	2818.4718	44	124.0128
5	2019-03-14 17:26:20.000	3002.2058	40	120.0882
6	2019-03-14 17:25:54.000	1878.0819	52	97.6603
7	2019-03-14 17:26:04.000	1903.5012	49	93.2716
8	2019-03-14 17:26:15.000	1868.6393	46	85.9574
9	2019-03-14 17:25:18.000	1315.5267	52	68.4074
10	2019-03-14 17:26:05.000	3003.1705	19	57.0602
11	2019-03-14 17:26:19.000	3002.1027	15	45.0315
12	2019-03-14 17:25:55.000	2231.6844	16	35.707
13	2019-03-14 17:25:53.000	1072.0553	32	34.3058
14	2019-03-14 17:25:16.000	1811.7122	18	32.6108
15	2019-03-14 17:26:14.000	1303.0379	24	31.2729



Log group: /aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9QXLHH800ES

```
#     @timestamp                 @message
▸ 1   2019-03-16 00:23:21.730    REPORT RequestId: b3e09f85-5fd2-46f9-a430-7a22aa9e57d8 Duration: 166.70 ms Billed Duration: 200 ms Memory Size: 256 MB Max Memory Used: 90 MB
▸ 2   2019-03-16 00:23:21.730    END RequestId: b3e09f85-5fd2-46f9-a430-7a22aa9e57d8
▸ 3   2019-03-16 00:23:21.600    2019-03-16T00:23:21.600Z b3e09f85-5fd2-46f9-a430-7a22aa9e57d8 Endpoint: 51jpka7ch1.execute-api.us-west-2.amazonaws.com/dev
▸ 4   2019-03-16 00:23:21.600    2019-03-16T00:23:21.600Z b3e09f85-5fd2-46f9-a430-7a22aa9e57d8 { Version: 899, Aliases: Set { wrapperName: 'Set', values: [ '$', 'David', 'David2' ], type: 'String' }, Connections: { 'Wn7BrdYevHcCzvg=': { Alias: 'David' }, 'Wn7s86KevHcAcuA=': { Alias: '$' }, 'Wn8bdeQMPHcAcuA=': { Alias: 'David2' }, 'Wn8s9f22vHcCEDQ=': {} }, ChatRoomId: 'default' }
▸ 5   2019-03-16 00:23:21.600    2019-03-16T00:23:21.600Z b3e09f85-5fd2-46f9-a430-7a22aa9e57d8 Returned item
▸ 6   2019-03-16 00:23:21.600    2019-03-16T00:23:21.600Z b3e09f85-5fd2-46f9-a430-7a22aa9e57d8 {"Type":"Message","Message":"anonymous","Alias":"$"}
▸ 7   2019-03-16 00:23:21.563    2019-03-16T00:23:21.563Z b3e09f85-5fd2-46f9-a430-7a22aa9e57d8 Starting
▸ 8   2019-03-16 00:23:21.562    START RequestId: b3e09f85-5fd2-46f9-a430-7a22aa9e57d8 Version: $LATEST
▸ 9   2019-03-16 00:23:18.790    REPORT RequestId: 85e2f2fb-557c-4e9f-96bd-c4656fbf42d0 Duration: 222.40 ms Billed Duration: 300 ms Memory Size: 256 MB Max Memory Used: 90 MB
▸ 10  2019-03-16 00:23:18.790    END RequestId: 85e2f2fb-557c-4e9f-96bd-c4656fbf42d0
```

Amazon CloudWatch Contributor Insights



MessagesBySender

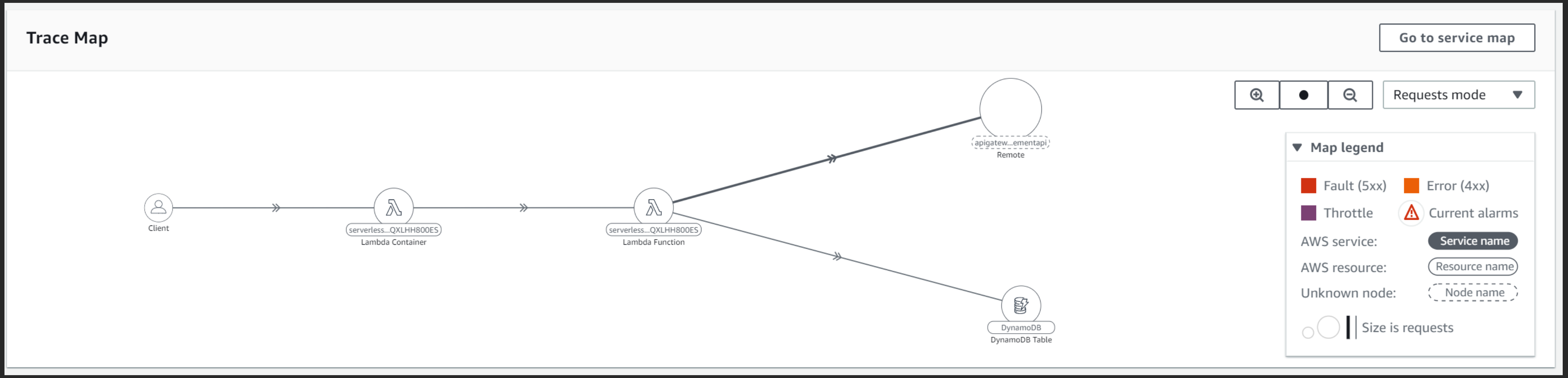
Show: Top 10 contributors · 1 minute period · Order by Sum

2019-11-29 (19:56:00) - 2019-11-29 (20:10:21)

6 unique contributors

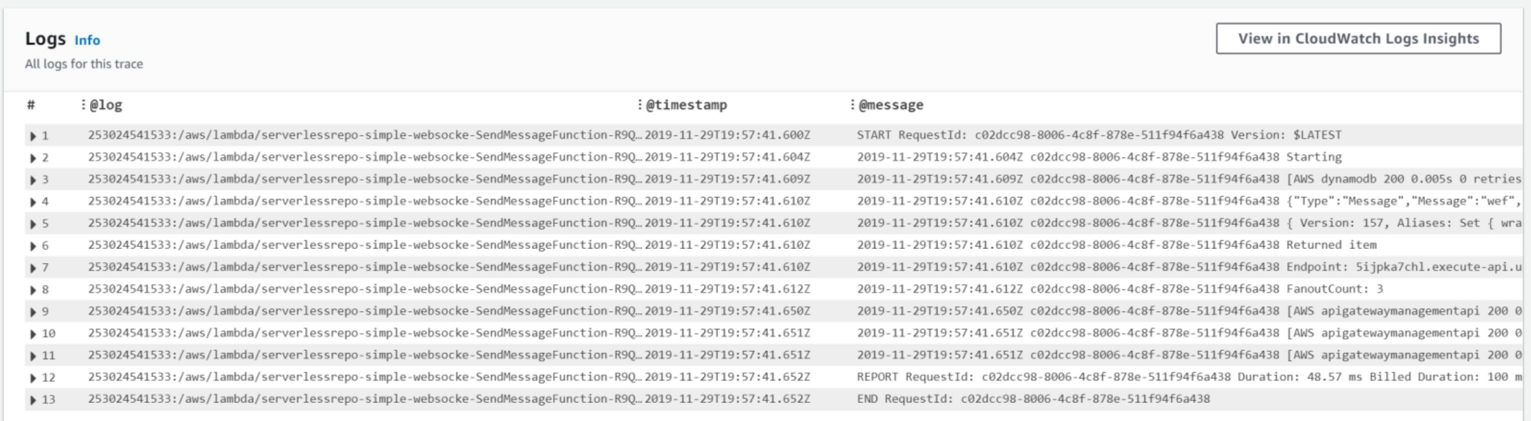
#		$.Alias	SampleCount
1	🟦	David	154
2	🟧	Laura	70
3	🟩	Katie	47
4	🟥	Ben	42
5	🟪	Kyle	41
6	🟫	Sarah	37

Amazon CloudWatch ServiceLens



Overall system health

Drill down to a specific request

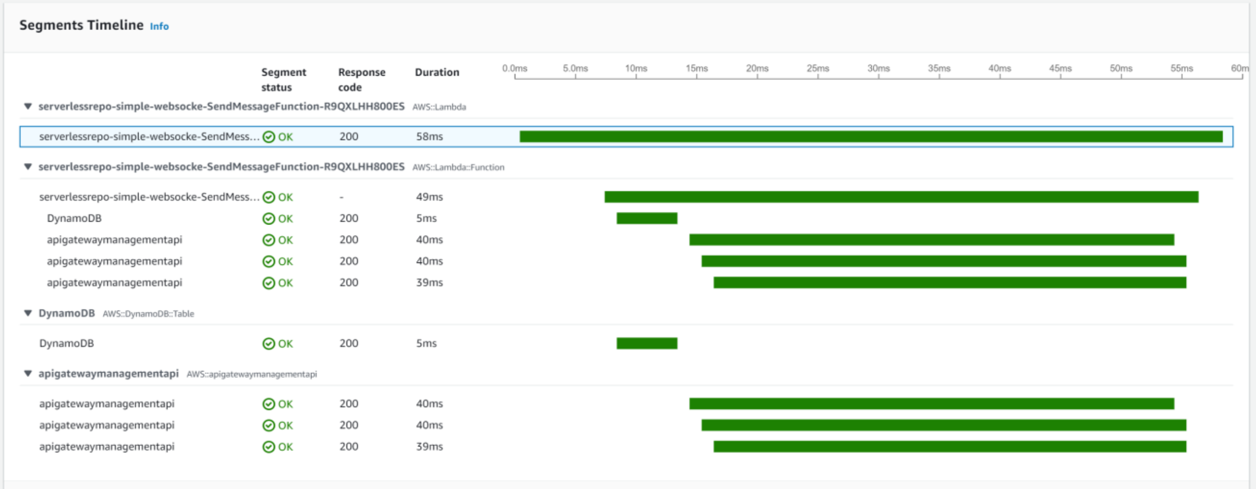


Logs Info

All logs for this trace

View in CloudWatch Logs Insights

```
#     ⋮@log                                                                  ⋮@timestamp                ⋮@message
▸ 1   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.600Z  START RequestId: c02dcc98-8006-4c8f-878e-511f94f6a438 Version: $LATEST
▸ 2   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.604Z  2019-11-29T19:57:41.604Z c02dcc98-8006-4c8f-878e-511f94f6a438 Starting
▸ 3   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.609Z  2019-11-29T19:57:41.609Z c02dcc98-8006-4c8f-878e-511f94f6a438 [AWS dynamodb 200 0.005s 0 retries
▸ 4   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.610Z  2019-11-29T19:57:41.610Z c02dcc98-8006-4c8f-878e-511f94f6a438 {"Type":"Message","Message":"wef",
▸ 5   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.610Z  2019-11-29T19:57:41.610Z c02dcc98-8006-4c8f-878e-511f94f6a438 { Version: 157, Aliases: Set { wra
▸ 6   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.610Z  2019-11-29T19:57:41.610Z c02dcc98-8006-4c8f-878e-511f94f6a438 Returned item
▸ 7   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.610Z  2019-11-29T19:57:41.610Z c02dcc98-8006-4c8f-878e-511f94f6a438 Endpoint: 5ijpka7chl.execute-api.u
▸ 8   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.612Z  2019-11-29T19:57:41.612Z c02dcc98-8006-4c8f-878e-511f94f6a438 FanoutCount: 3
▸ 9   253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.650Z  2019-11-29T19:57:41.650Z c02dcc98-8006-4c8f-878e-511f94f6a438 [AWS apigatewaymanagementapi 200 0
▸ 10  253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.651Z  2019-11-29T19:57:41.651Z c02dcc98-8006-4c8f-878e-511f94f6a438 [AWS apigatewaymanagementapi 200 0
▸ 11  253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.651Z  2019-11-29T19:57:41.651Z c02dcc98-8006-4c8f-878e-511f94f6a438 [AWS apigatewaymanagementapi 200 0
▸ 12  253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.652Z  REPORT RequestId: c02dcc98-8006-4c8f-878e-511f94f6a438 Duration: 48.57 ms Billed Duration: 100 m
▸ 13  253024541533:/aws/lambda/serverlessrepo-simple-websocke-SendMessageFunction-R9Q… 2019-11-29T19:57:41.652Z  END RequestId: c02dcc98-8006-4c8f-878e-511f94f6a438
```



Segments Timeline Info

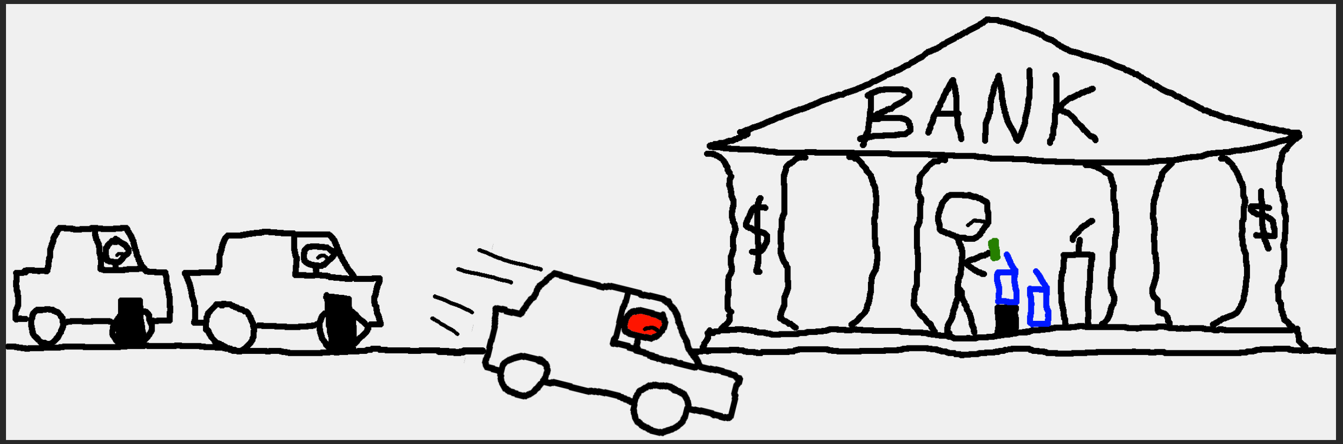
	Segment status	Response code	Duration
▼ **serverlessrepo-simple-websocke-SendMessageFunction-R9QXLHH800ES** AWS::Lambda			
serverlessrepo-simple-websocke-SendMess…	✓ OK	200	58ms
▼ **serverlessrepo-simple-websocke-SendMessageFunction-R9QXLHH800ES** AWS::Lambda::Function			
serverlessrepo-simple-websocke-SendMess…	✓ OK	-	49ms
DynamoDB	✓ OK	200	5ms
apigatewaymanagementapi	✓ OK	200	40ms
apigatewaymanagementapi	✓ OK	200	40ms
apigatewaymanagementapi	✓ OK	200	39ms
▼ **DynamoDB** AWS::DynamoDB::Table			
DynamoDB	✓ OK	200	5ms
▼ **apigatewaymanagementapi** AWS::apigatewaymanagementapi			
apigatewaymanagementapi	✓ OK	200	40ms
apigatewaymanagementapi	✓ OK	200	40ms
apigatewaymanagementapi	✓ OK	200	39ms

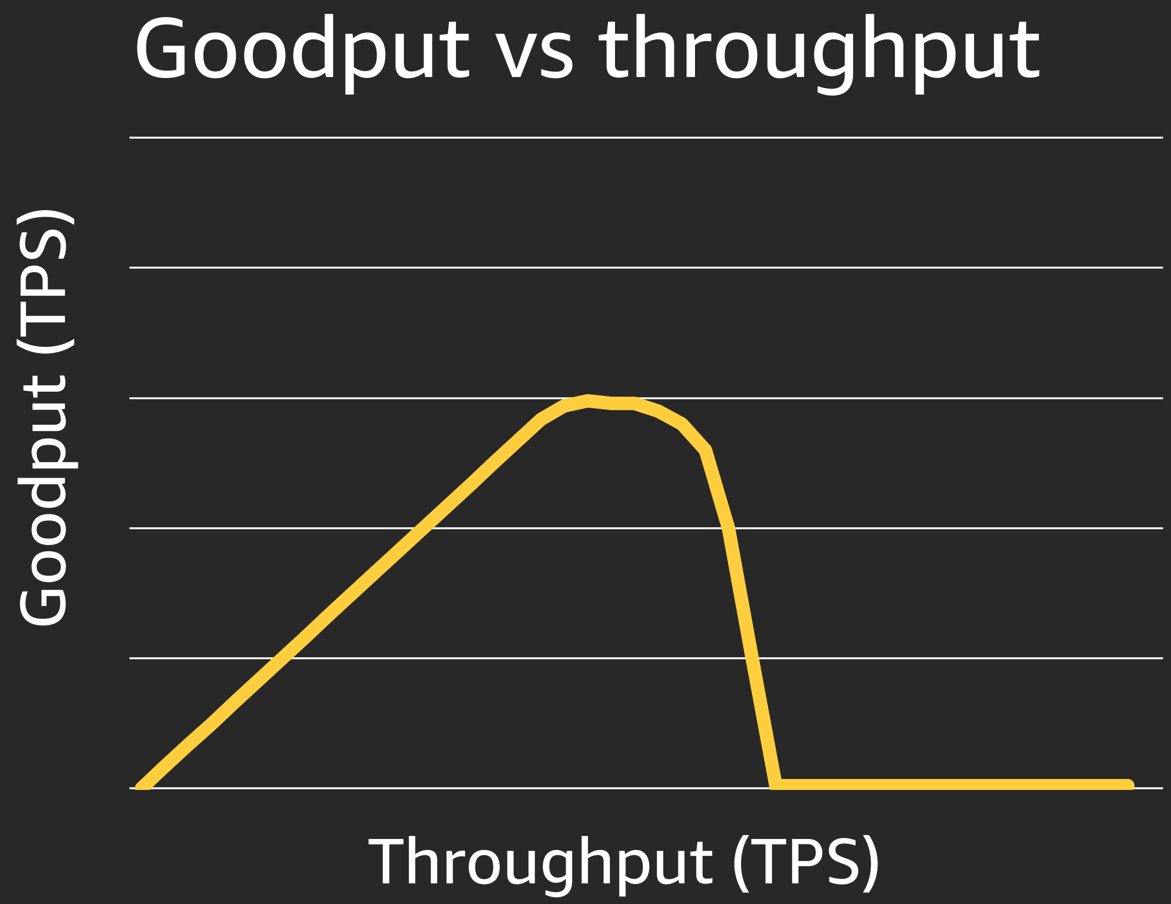
Recap: Resilient systems

AWS re:Invent

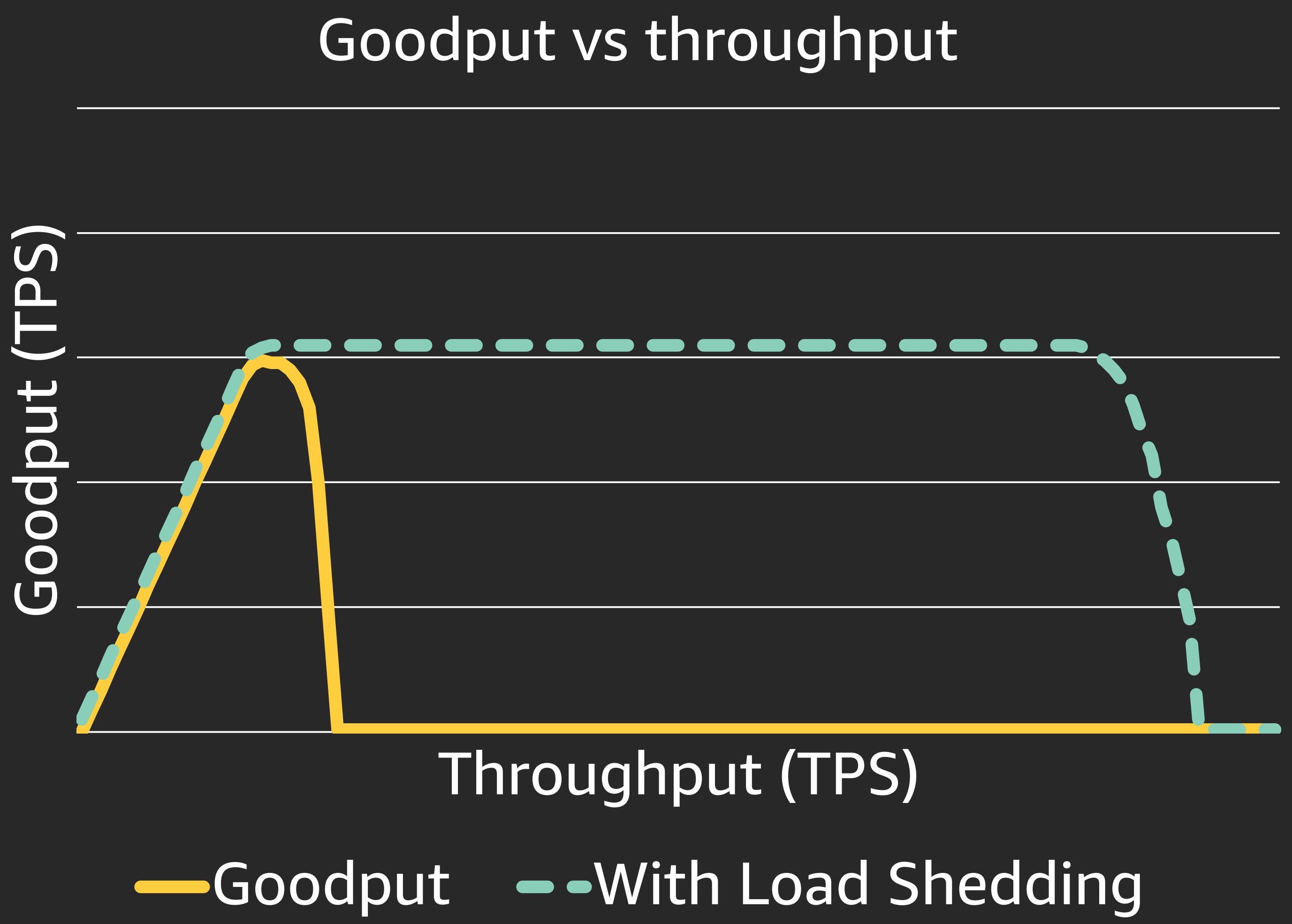
aws

Don’t take on too much work

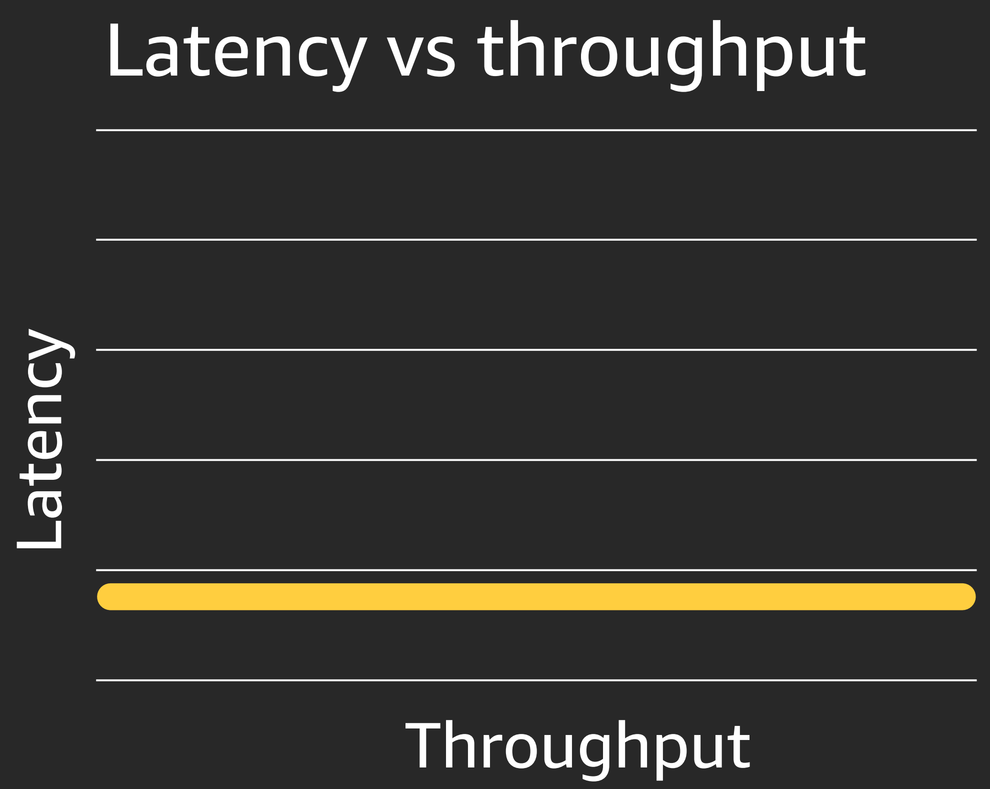


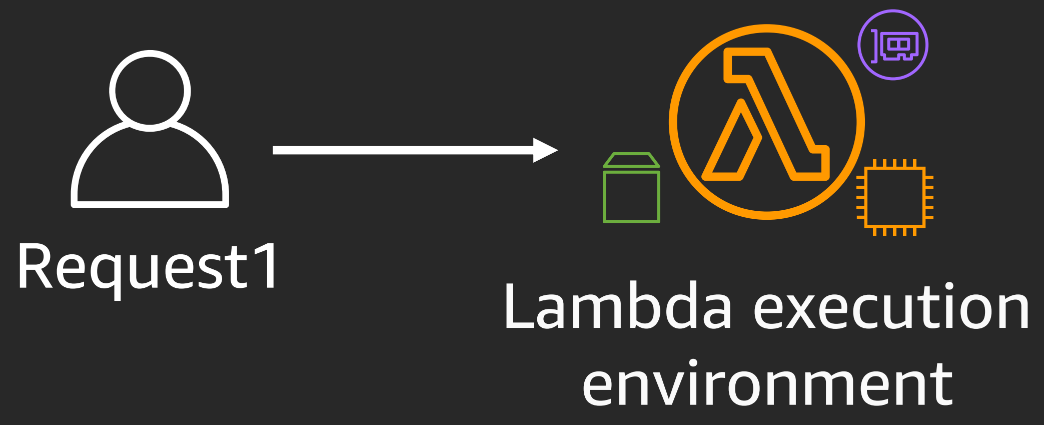


Reject excess work



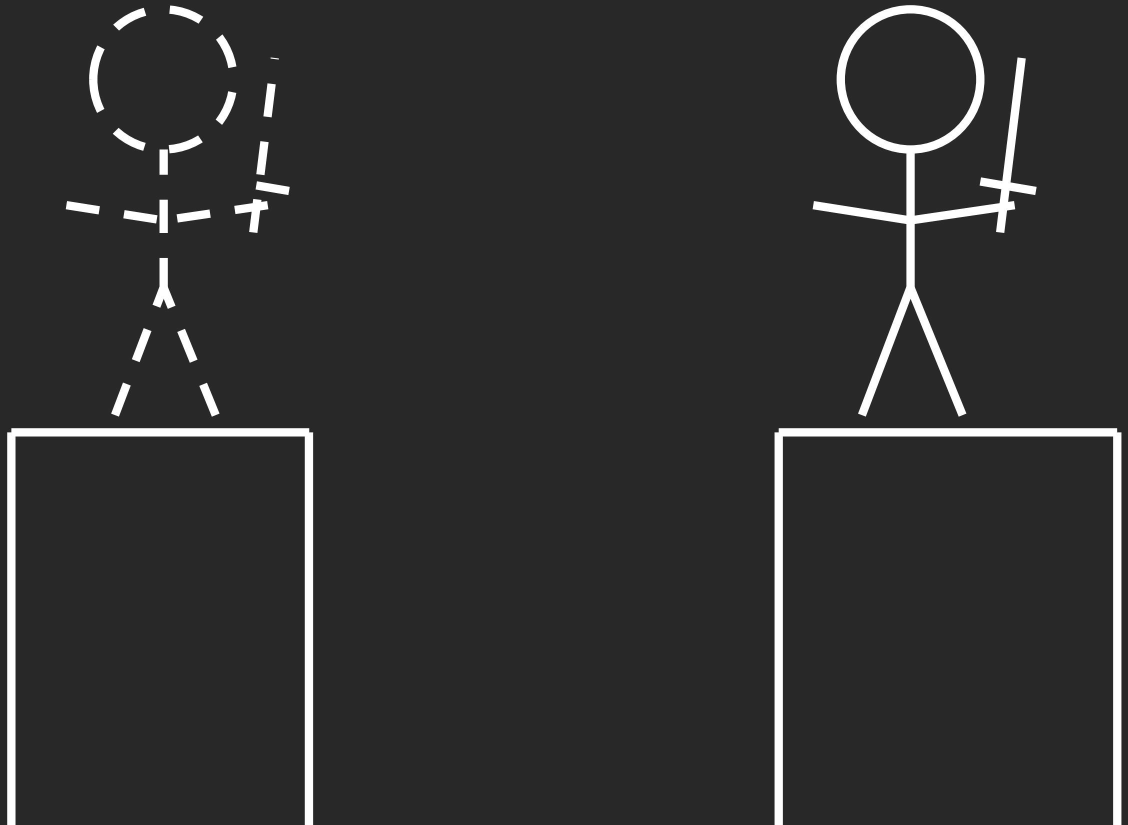
Workload isolation means predictable performance



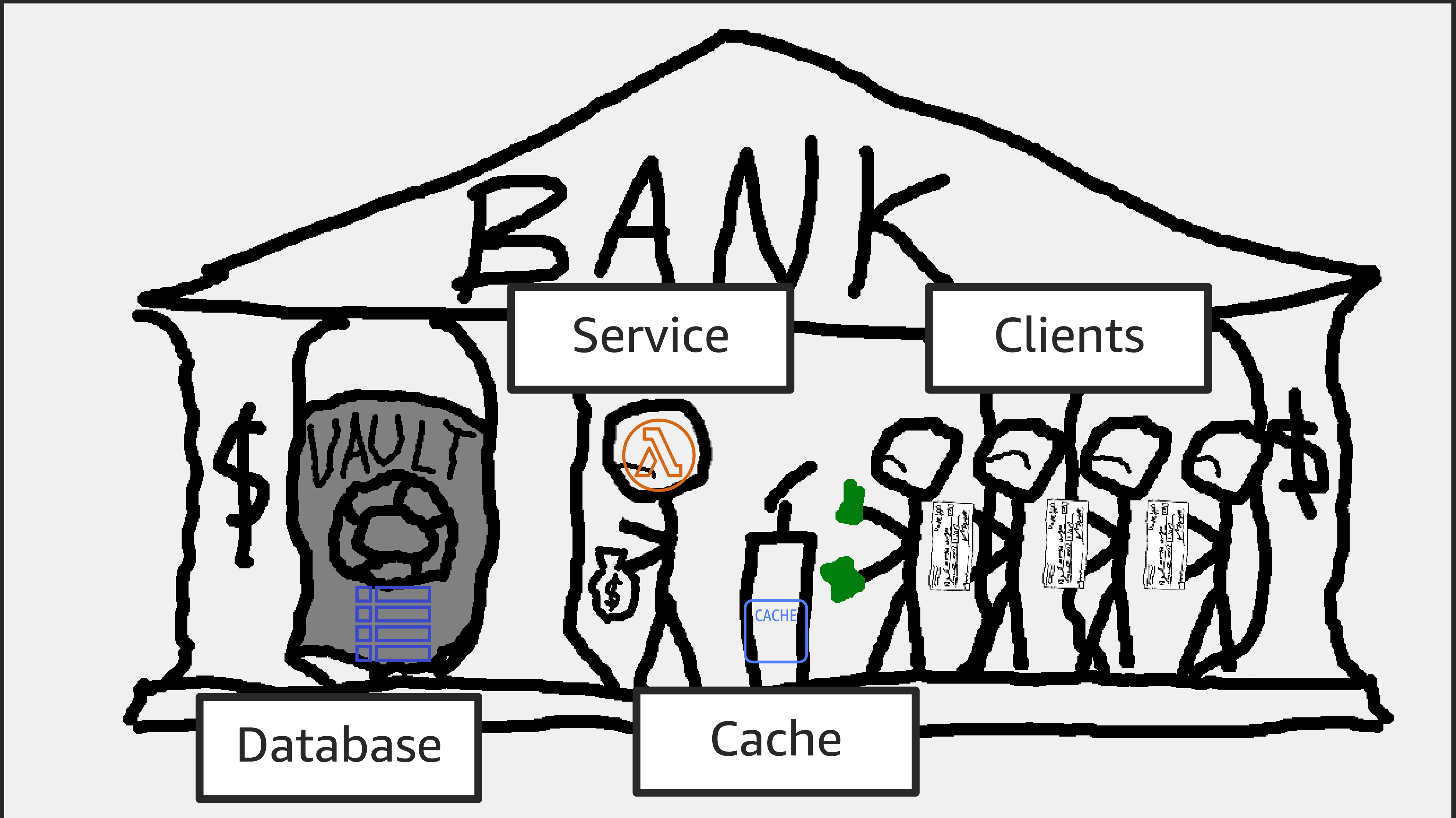


(only work on one thing at a time)

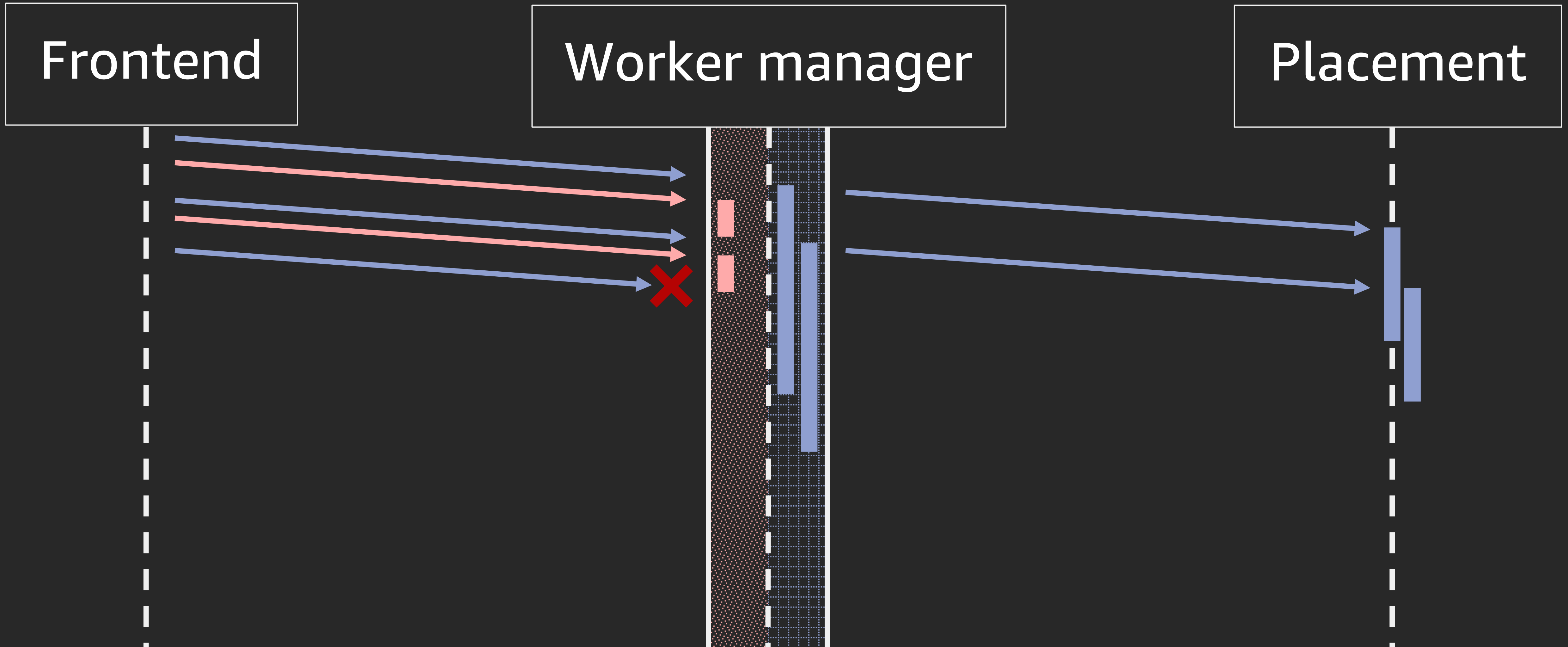
Bounded work



Compartmentalize dependencies



Compartmentalize dependencies



Compartmentalize dependencies

AWS Lambda

(per-function concurrency control)

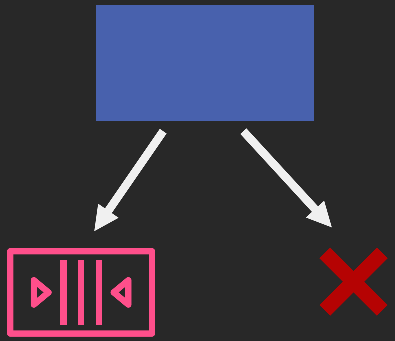
Amazon
API Gateway

(per-API throttling)

Watch out for queue backlogs

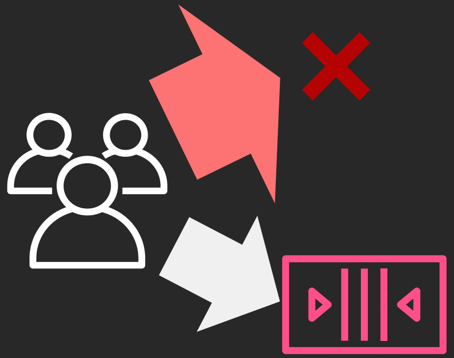
Avoid pile-ups

Queues are quick to fill, slow to drain



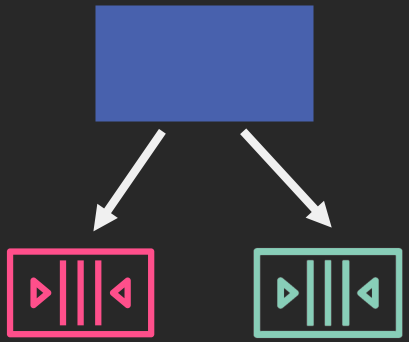
TTLs

Drop messages that are no longer relevant



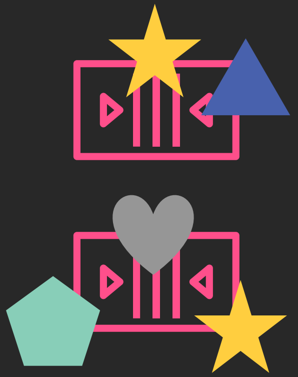
Backpressure

Prevent a backlog from becoming unmanageable



Priority queues

Shift the backlog into a low-priority queue



Shuffle-sharding

Isolate backlogs in unrelated workloads

Serverless == resiliency

Learn serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development

Free, on-demand courses on serverless, including

- Introduction to Serverless Development
- Getting into the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures

Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at https://aws.training

Thank you!

Please complete the session survey in the mobile app.

AWS re:Invent

aws