AWS re:Invent

SVS406-R

Asynchronous processing best practices with AWS Lambda

Cecilia Deng
Software Development Manager
AWS Lambda
Amazon Web Services

AWS re:Invent

aws

Agenda

Why should you use async

Managing async with AWS Lambda

- Amazon Simple Queue Service (Amazon SQS) event sources
- Stream event sources
- Async Invoke

AWS Lambda async invoke: under the hood

Why async

Why async

Immediate relief from latency

- Caller does not have to wait
- Caller is then free to do other work

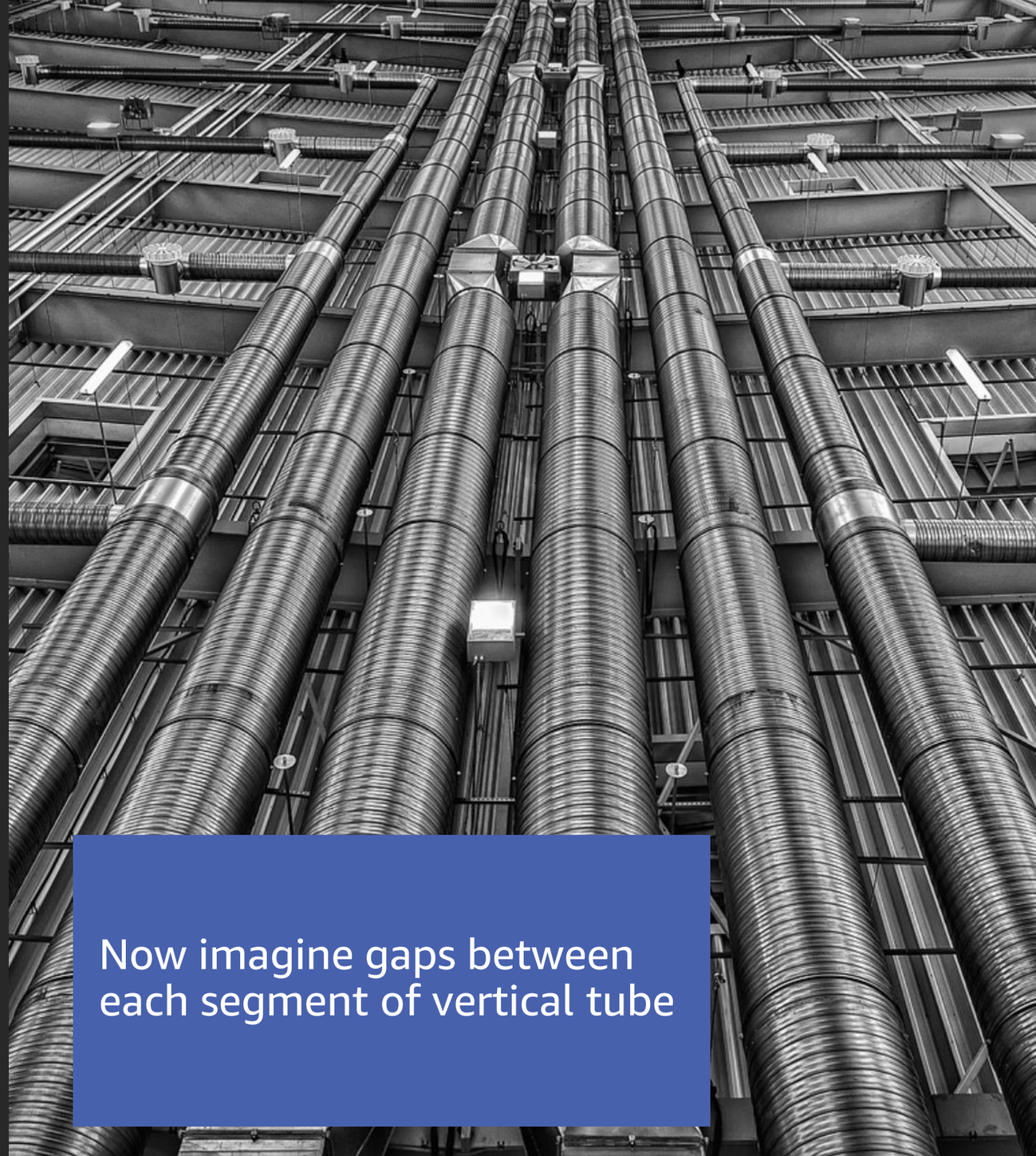
Resiliency

- Allowance for handling errors
- Allowance for handling throttles
- Allowance for variances in transmission duration

Why buffers

No buffers means no allowances to wait

Global synchronized timing becomes critical to avoiding loss



Now imagine gaps between each segment of vertical tube

“The road to hell is paved with good intentions.”

Henry G. Bohn

A Hand-Book of Proverbs, 1855

AWS re:Invent

aws

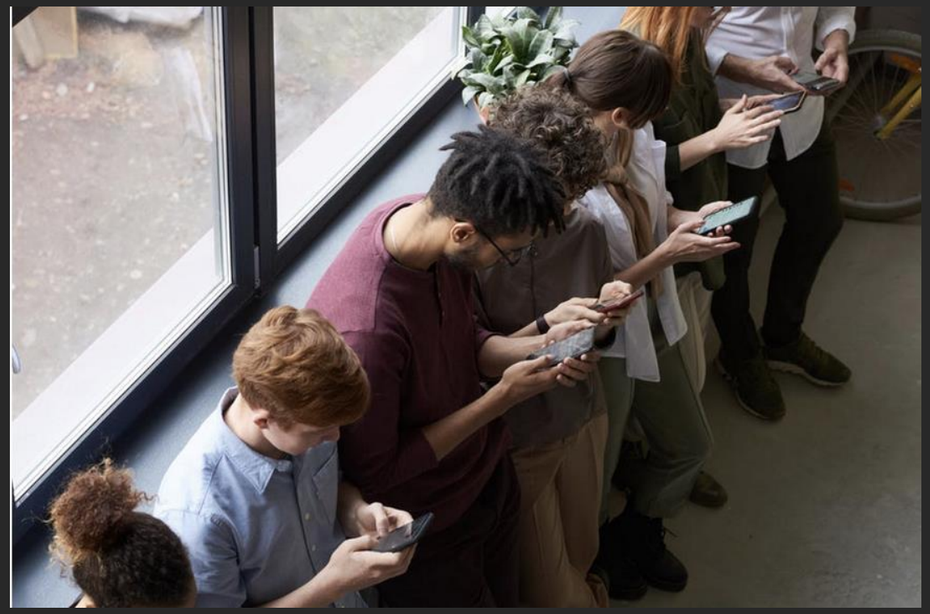
Tradeoffs

Latency

- Handling errors and throttles takes time
- End-to-end latency increases and varies

Backlogs

- Downstream bottlenecks and backpressure lead to more waiting
- Buffers fill up = backlog



Managing async with AWS Lambda

AWS re:Invent

aws

Async with AWS Lambda

Lambda async invoke

- The buffer is managed for you

Amazon SQS event source to Lambda

- You manage the buffer (Amazon SQS)

Stream event source to Lambda

- You manage the buffer (Amazon Kinesis or Amazon DynamoDB stream)
- Each shard is like a FIFO queue

Friendly backlogs

- Shock absorbs traffic spikes
- Shock absorbs error/throttle spikes
- Does not last long
- Acknowledges fairness

Befriending backlogs

Prioritize workloads and

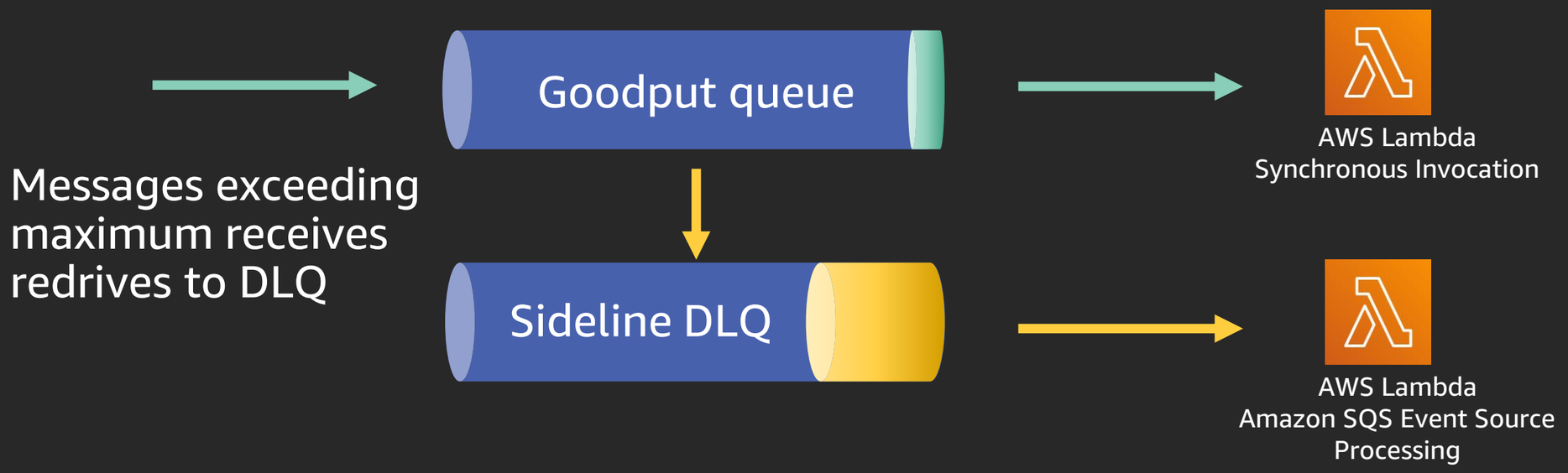
- Delay some workloads
- Throw away some workloads

Increase egress rate with

- More frequent processing
- Parallel processing

Prioritizing workloads: Amazon SQS event sources

Set the DLQ to hold lower priority workloads. This can be processed as another event source to Lambda that will not interfere with the goodput queue.



Prioritizing workloads: stream event sources

New controls allows you to configure criteria for which a stream event source record can be skipped. Use the existing event source mapping APIs.

```
"DestinationConfig”: {
        “OnFailure”: {
                “Destination”: string
        }
}
"MaximumRecordAgeInSeconds": number,
"MaximumRetryAttempts": number,
“BisectOnFunctionError”: boolean
```

New Features!

Prioritizing workloads: stream event sources

```
"DestinationConfig”: {
        “OnFailure”: {
                “Destination”: string
        }
}
"MaximumRecordAgeInSeconds": number,
"MaximumRetryAttempts": number,
“BisectOnFunctionError”: boolean
```

New Features!

DestinationConfig OnFailure: Choose a destination for unsuccessful invocations

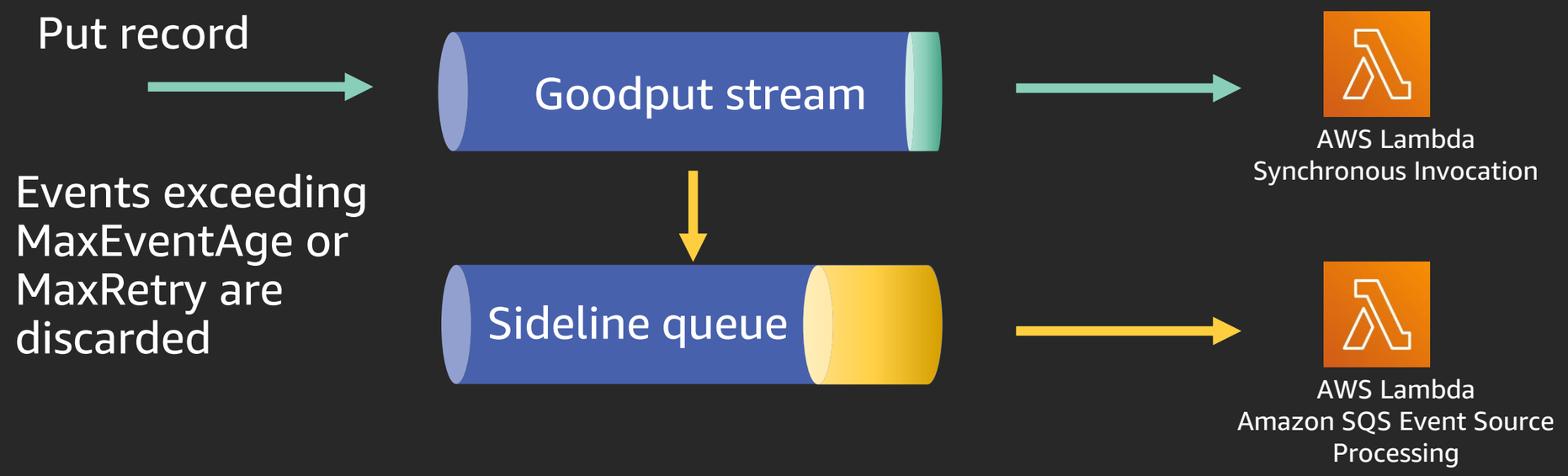
MaximumRecordAgeInSeconds: Set a TTL (time to live) between 60 seconds and 6 hours, after which a backlogged invocation will be discarded

MaximumRetryAttempts: Set max retry between 0 and 2 attempts

BisectOnFunctionError: Allow record batches to be split and retried for invoke

Prioritizing workloads: stream event sources

Lambda stream event source workloads



Set OnFailure Destination to be an Amazon SQS queue that can hold lower priority workloads. This can be processed by a separate Lambda function.

Prioritizing workloads: async invoke

EventInvokeConfig allows you to configure criteria for which a Lambda asynchronous invocation should be discarded. These can be set through a new set of APIs.

```
”FunctionEventInvokeConfig": {
        "DestinationConfig”: {
                “OnFailure”: {
                        “Destination”: string
                }
        },
        "MaximumEventAgeInSeconds": number,
        "MaximumRetryAttempts": number
}
```

New Features!

Prioritizing workloads: async invoke

```
”FunctionEventInvokeConfig": {
        "DestinationConfig”: {
                “OnFailure”: {
                        “Destination”: string
                }
        },
        "MaximumEventAgeInSeconds": number,
        "MaximumRetryAttempts": number
}
```

New Features!

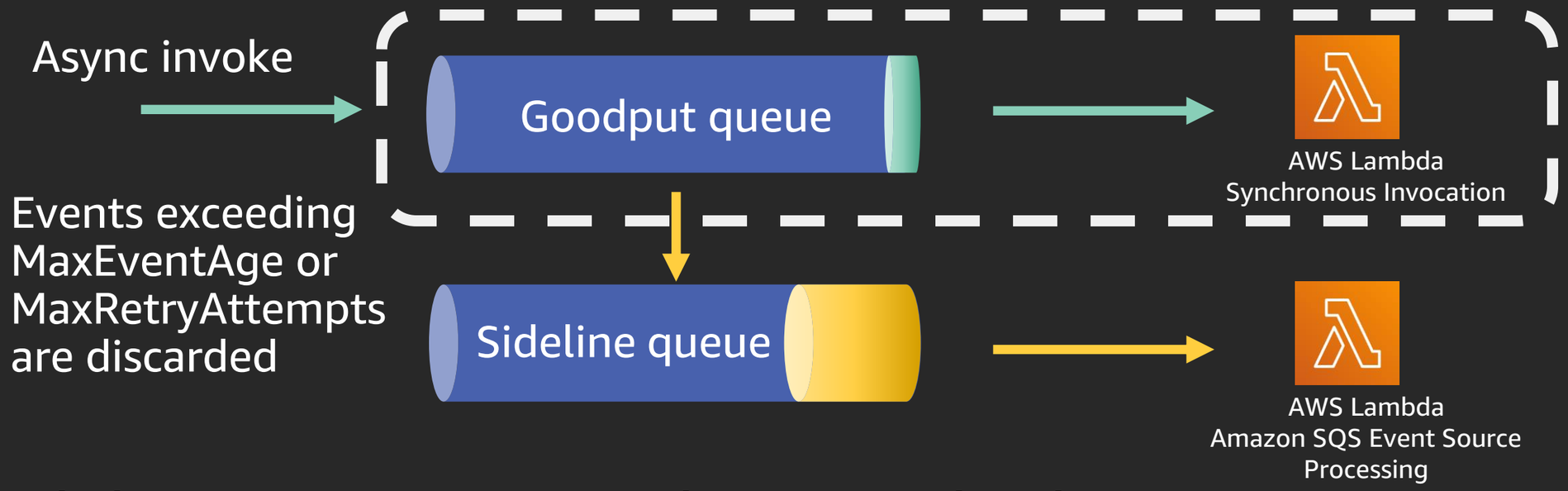
DestinationConfig OnFailure: Choose a destination for unsuccessful invocations

MaximumEventAgeInSeconds: Set a TTL (time to live) between 60 seconds and 6 hours, after which a backlogged invocation will be discarded

MaximumRetryAttempts: Set max retry between 0 and 2 attempts

Prioritizing workloads: async invoke

Set OnFailure destination to be an Amazon SQS queue that can hold lower priority workloads. This can be processed by a separate Lambda function.



Lambda async invocations that exceed either MaximumEventAgeInSeconds or MaximumRetryAttempts will first honor OnFailure destinations before being discarded

Networking parallels

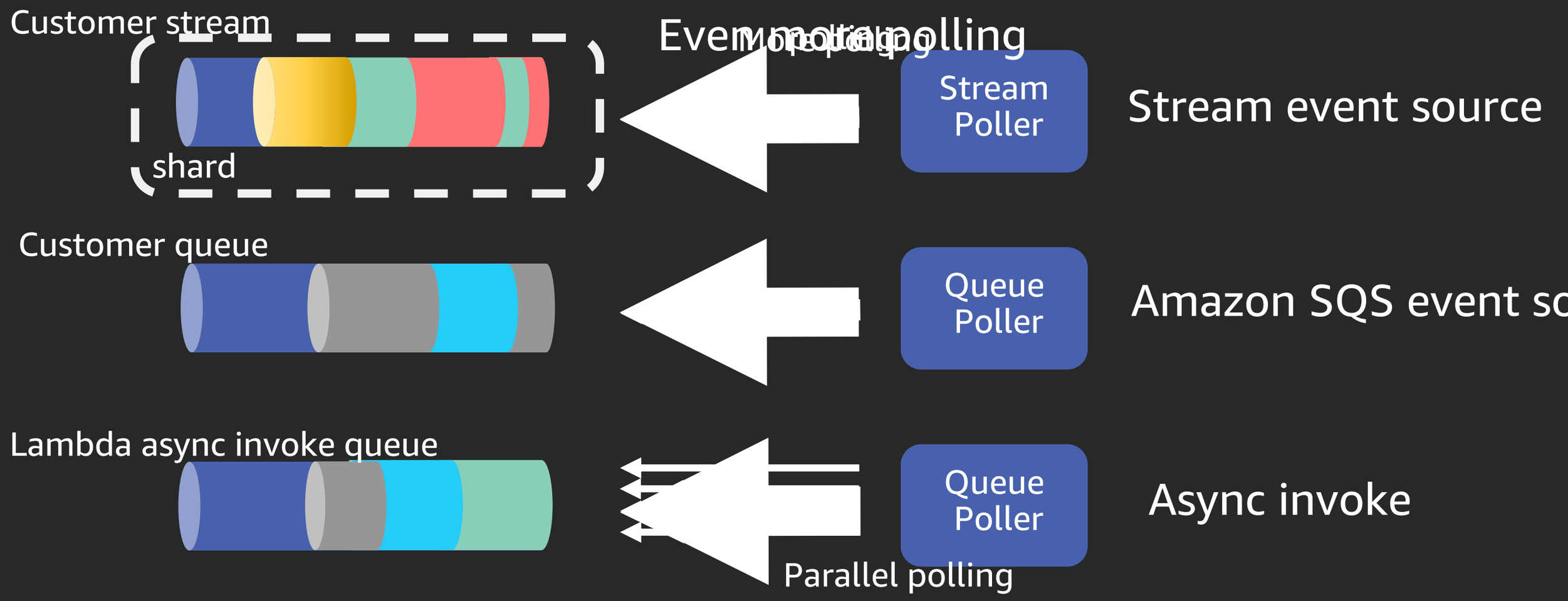
Bufferbloat is the existence of excessively large (bloated) buffers in systems, particularly network communication systems.

Goodput is the actual payload throughput of a link, stripped of all the traffic that is retried and other overhead.

Here we have some organic packets that can become less useful over time

Beefier processing

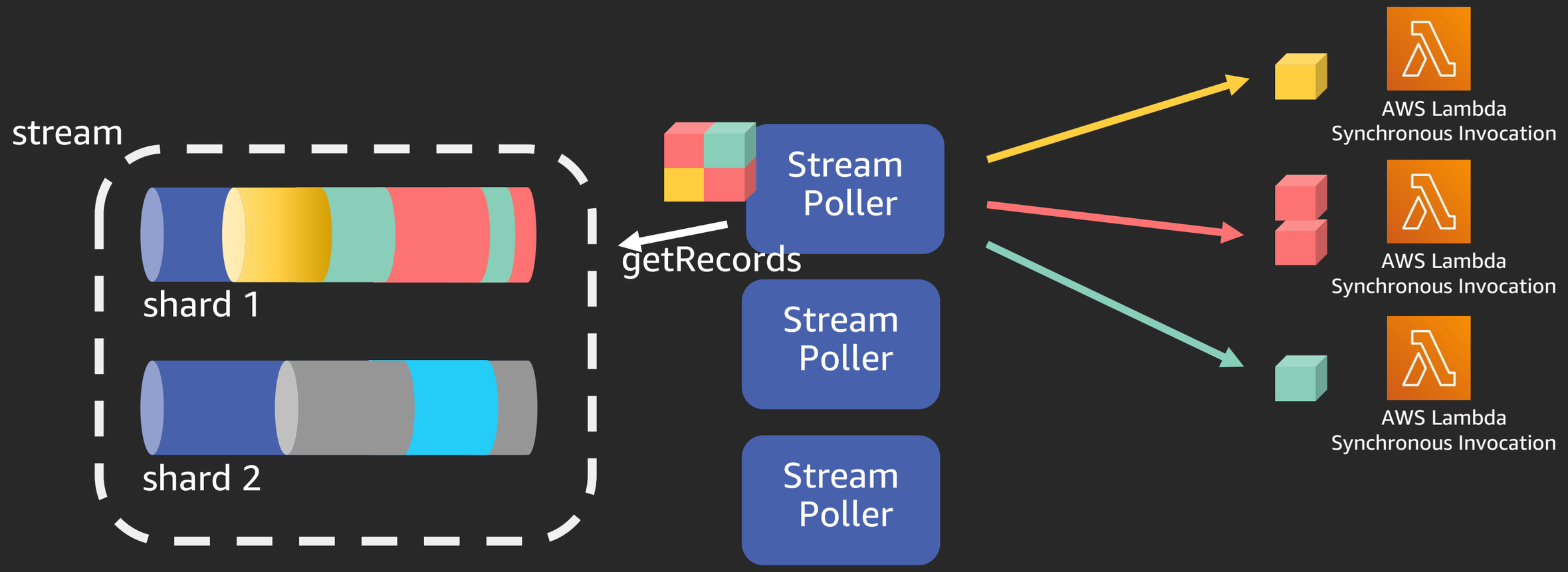
Under the hood: Lambda auto scales processing power (number of parallel threads) in response to incoming traffic



Parallel processing

New Feature!

The Parallelization Factor for stream event sources allows customers to set how many Lambdas a single shard can invoke simultaneously with partition key order guaranteed

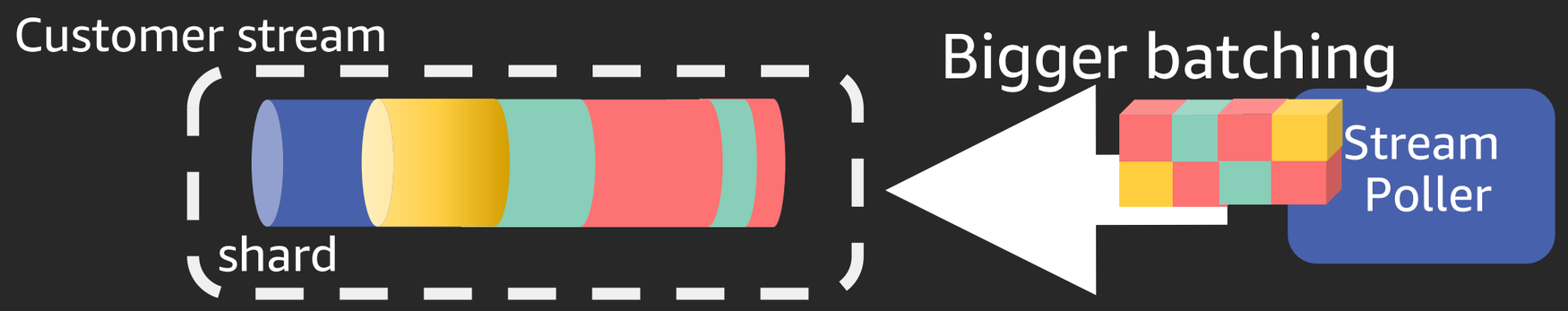


Networking parallels

Head-of-line blocking (**HOL blocking**) is a performance-limiting phenomenon that occurs when a line of packets is held up by the first packet.

Virtual output queueing (VOQ) is a technique where, rather than keeping all traffic in a single queue, separate queues are maintained for each possible output location.

Beefier processing



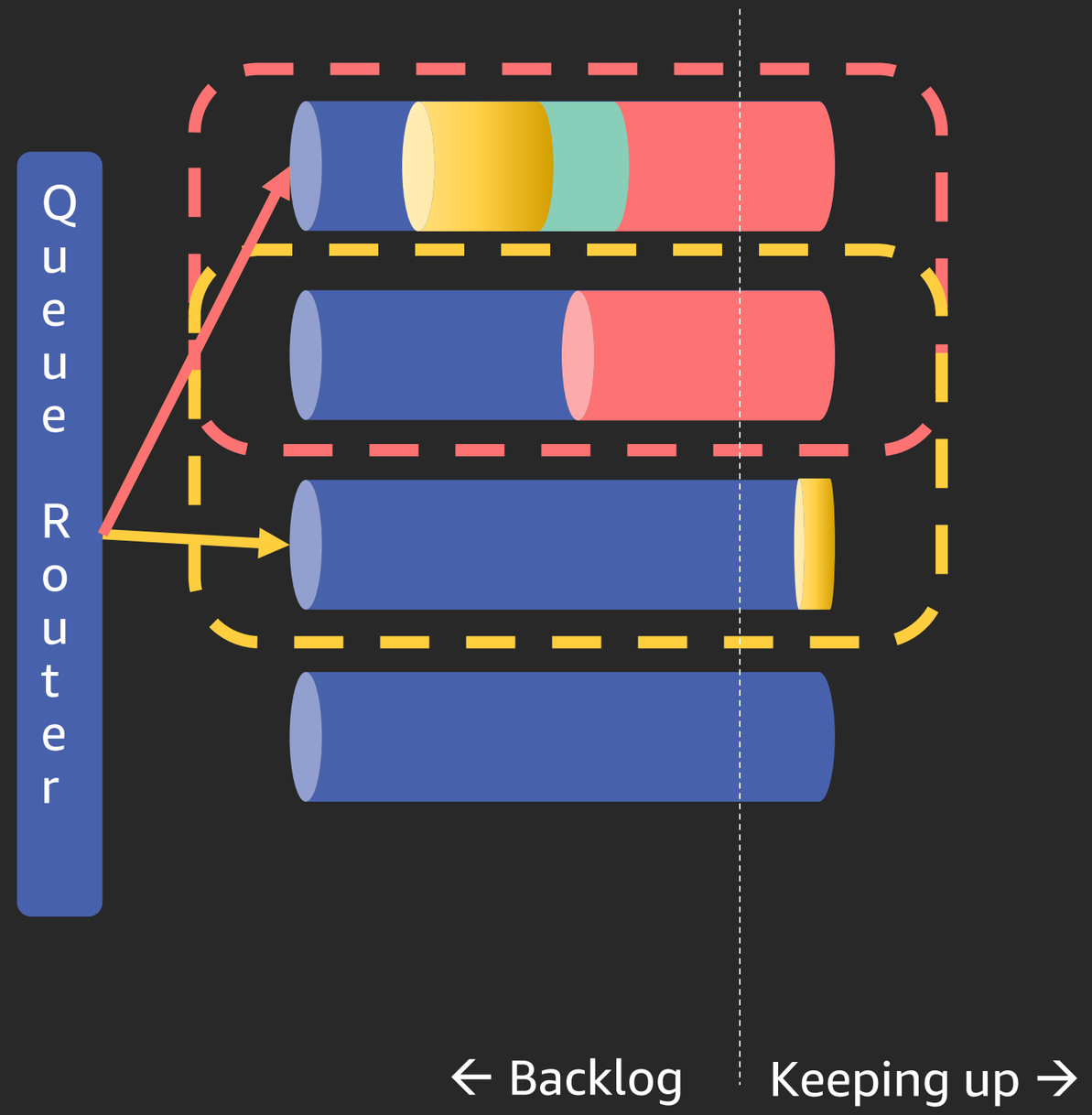
Set Custom Batch Window for your stream event source to wait up to 300 seconds to build a batch before invoking a function

AWS Lambda async invoke: under the hood

AWS re:Invent

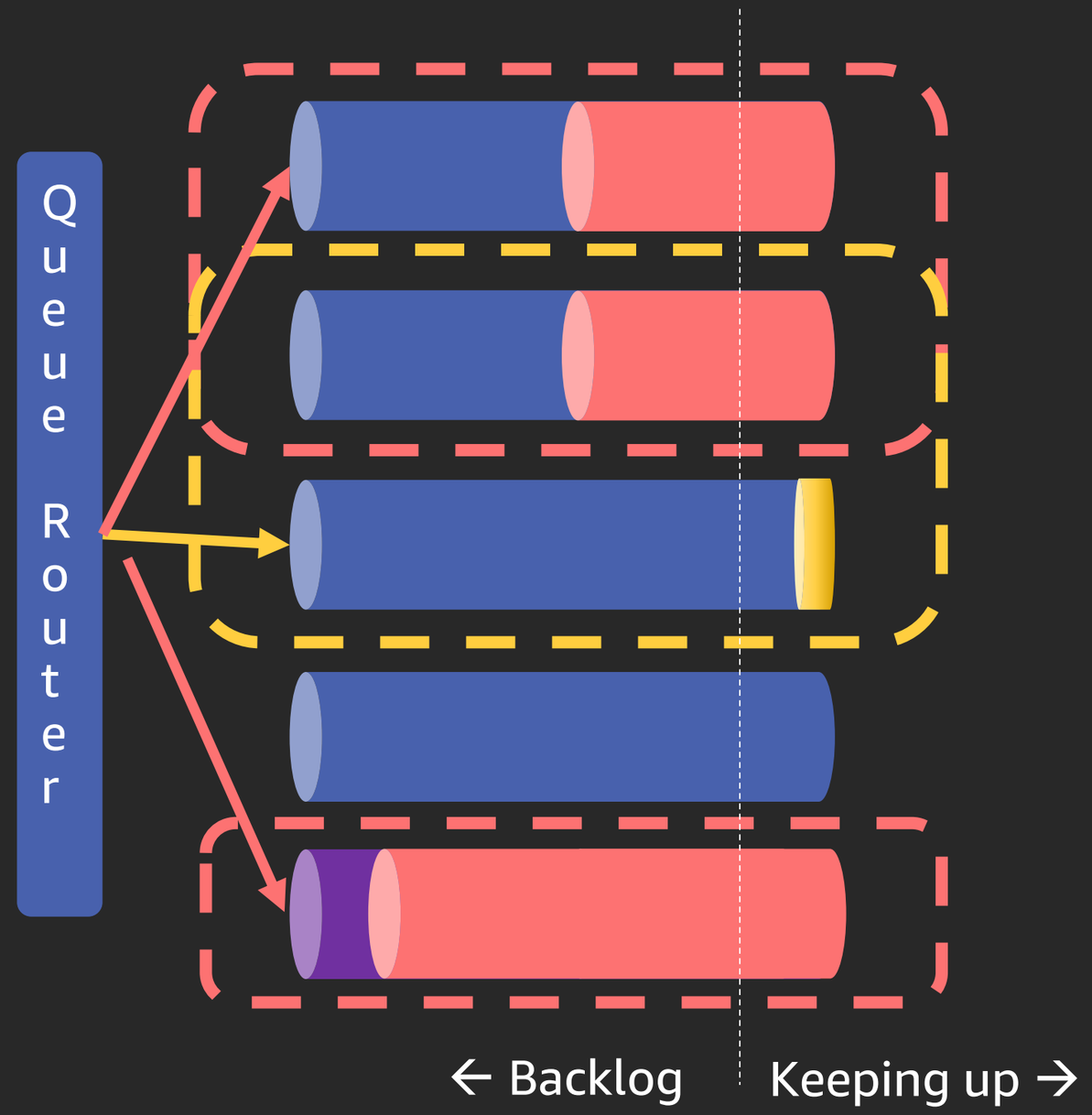
aws

Isolating backlogs



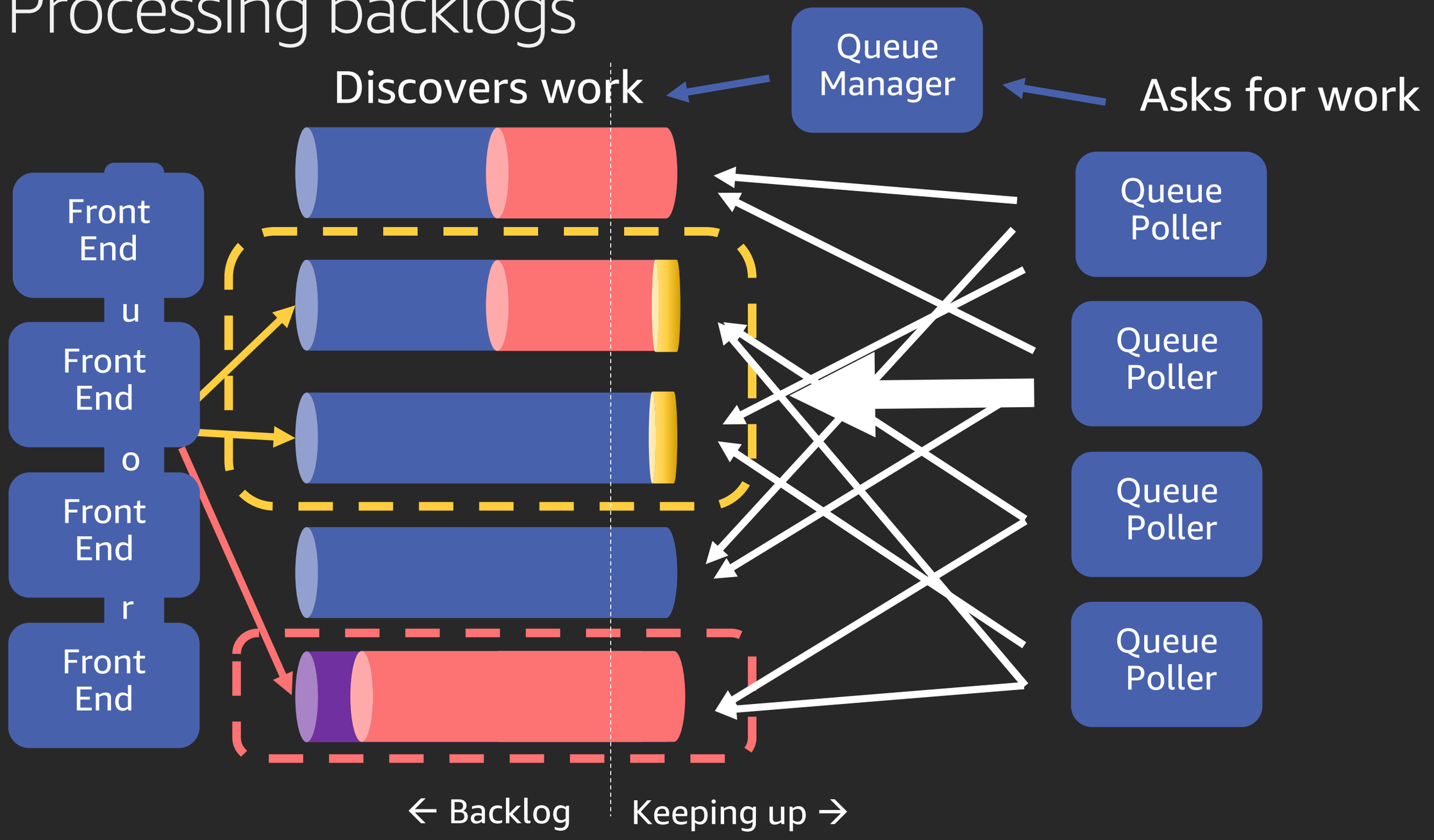
- Sharing queues can lead to interference
- Shuffle Sharding reduces scope of impact
- Best of N increases likelihood of isolation

Sidelining backlogs

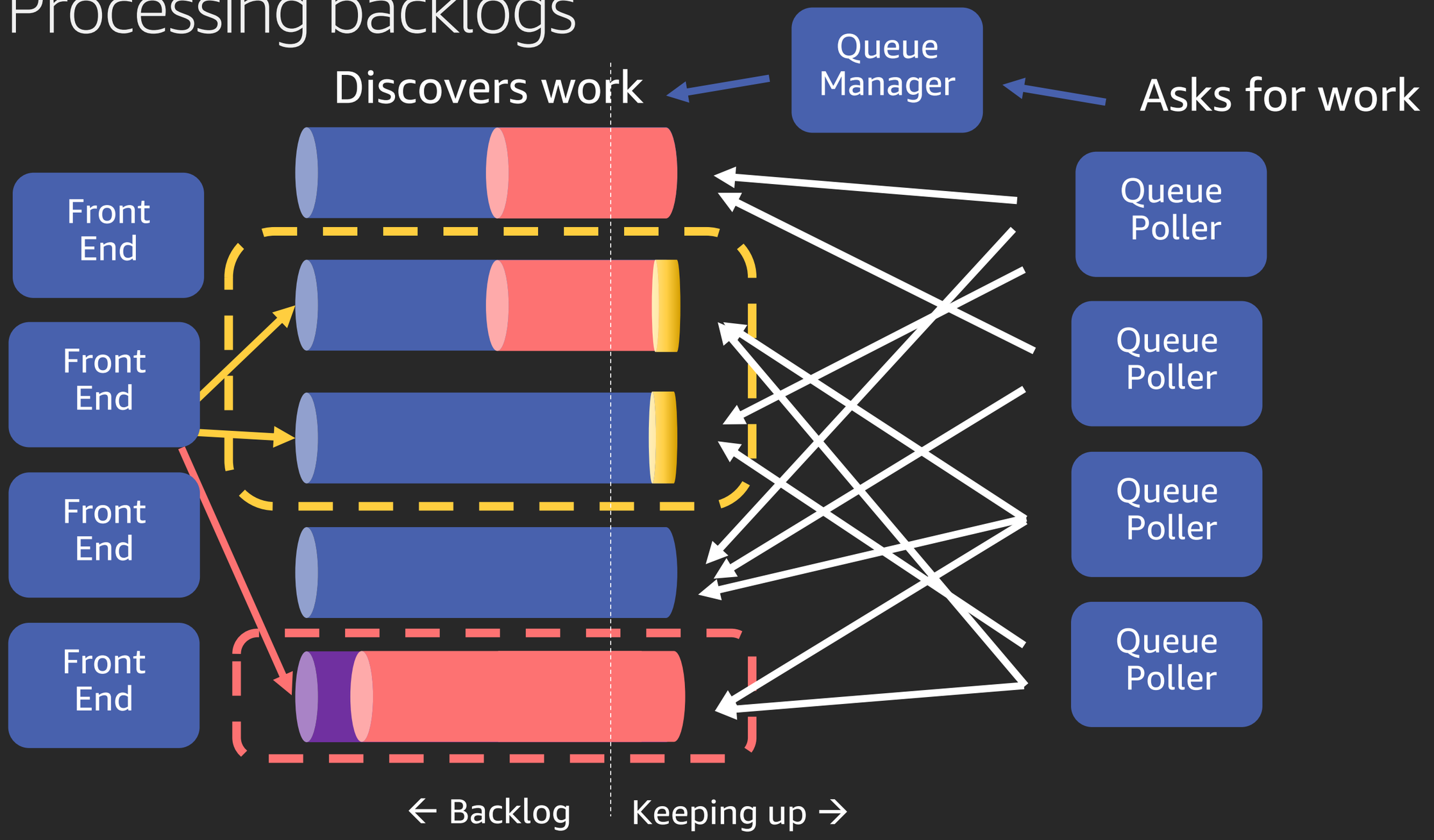


- Sidelining guarantees isolation
- Hybrid approach leverages probability while relying on determinism
- Sideline when backlog is determined to be sustained

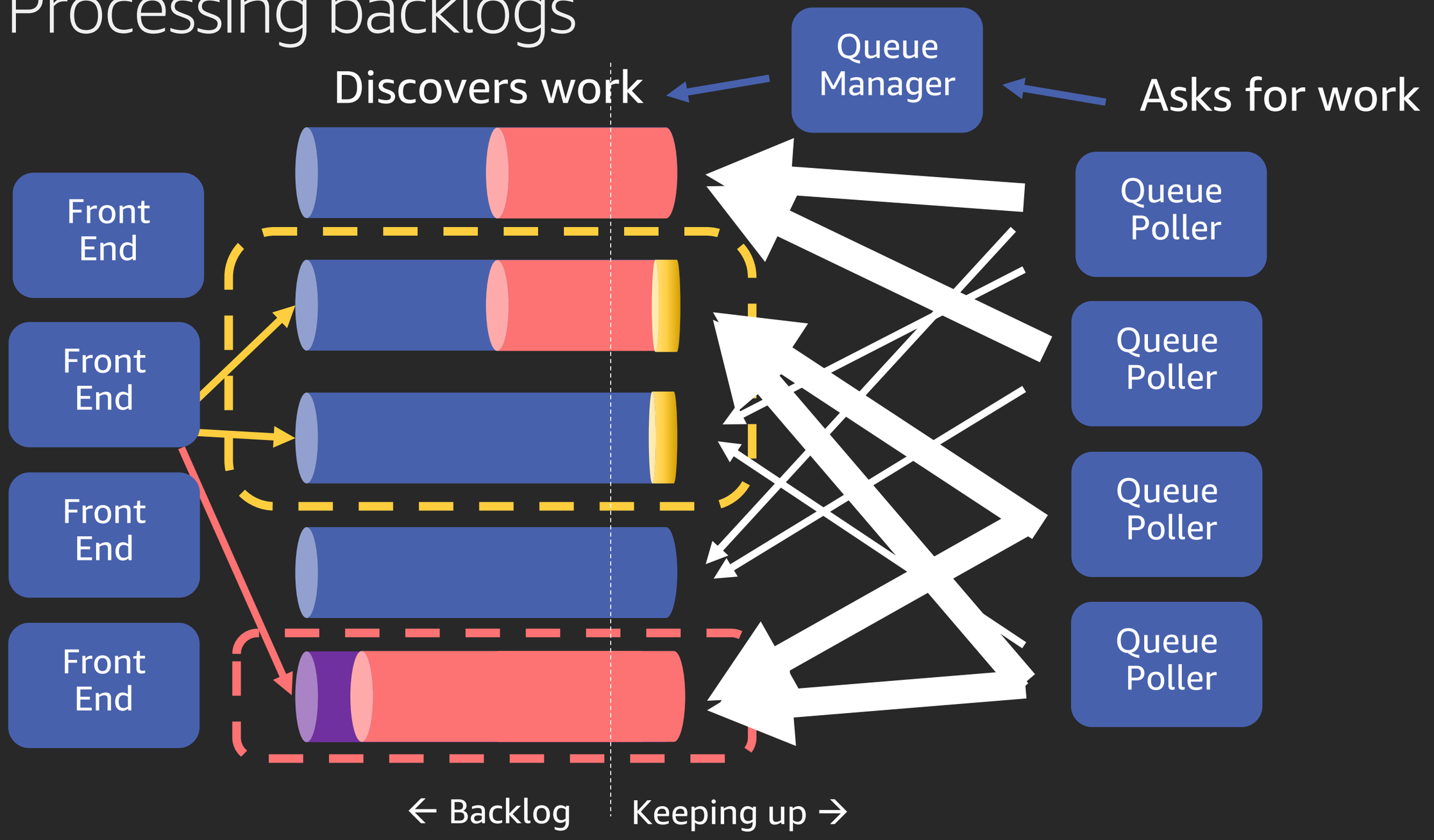
Processing backlogs



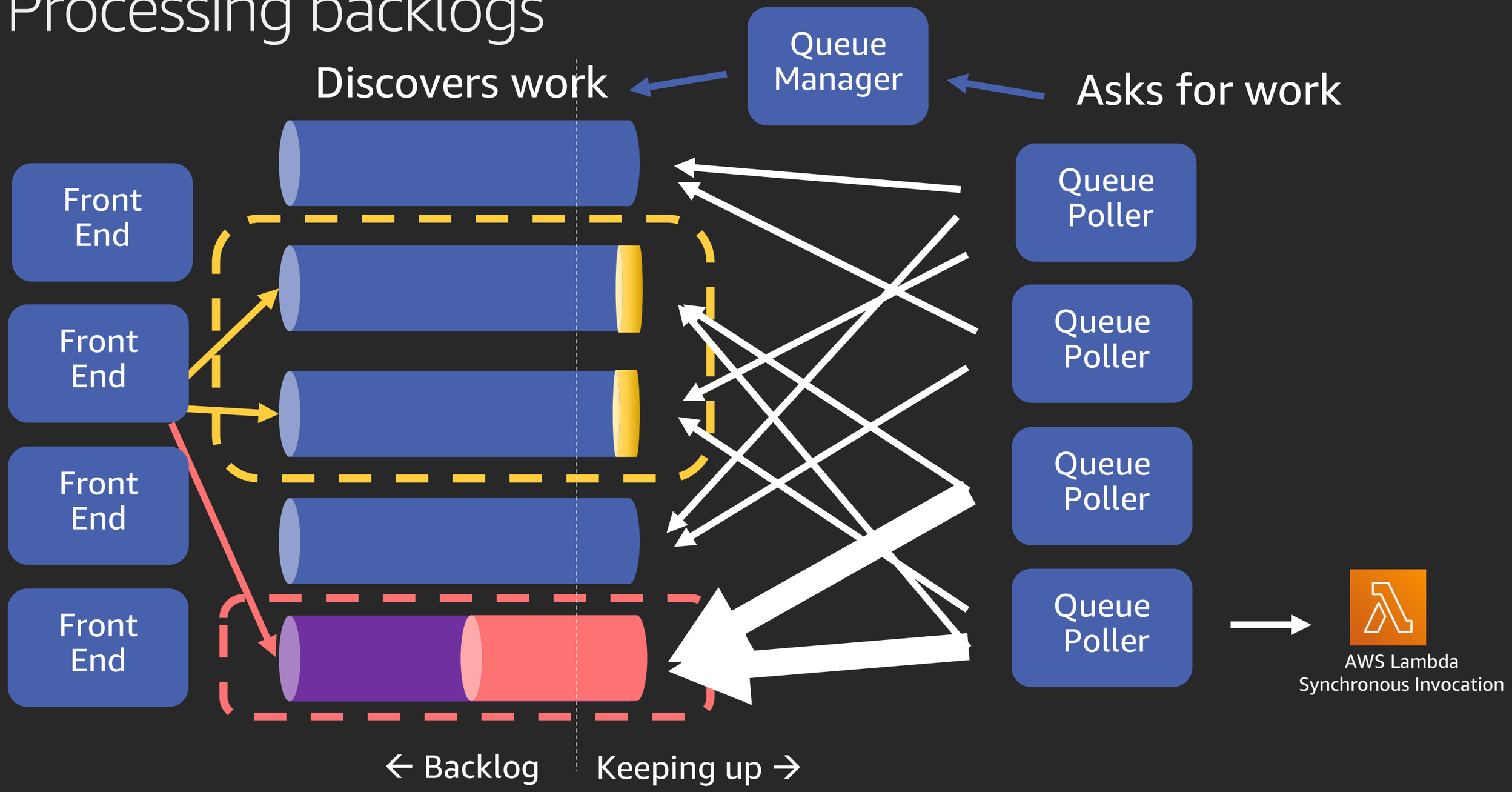
Processing backlogs



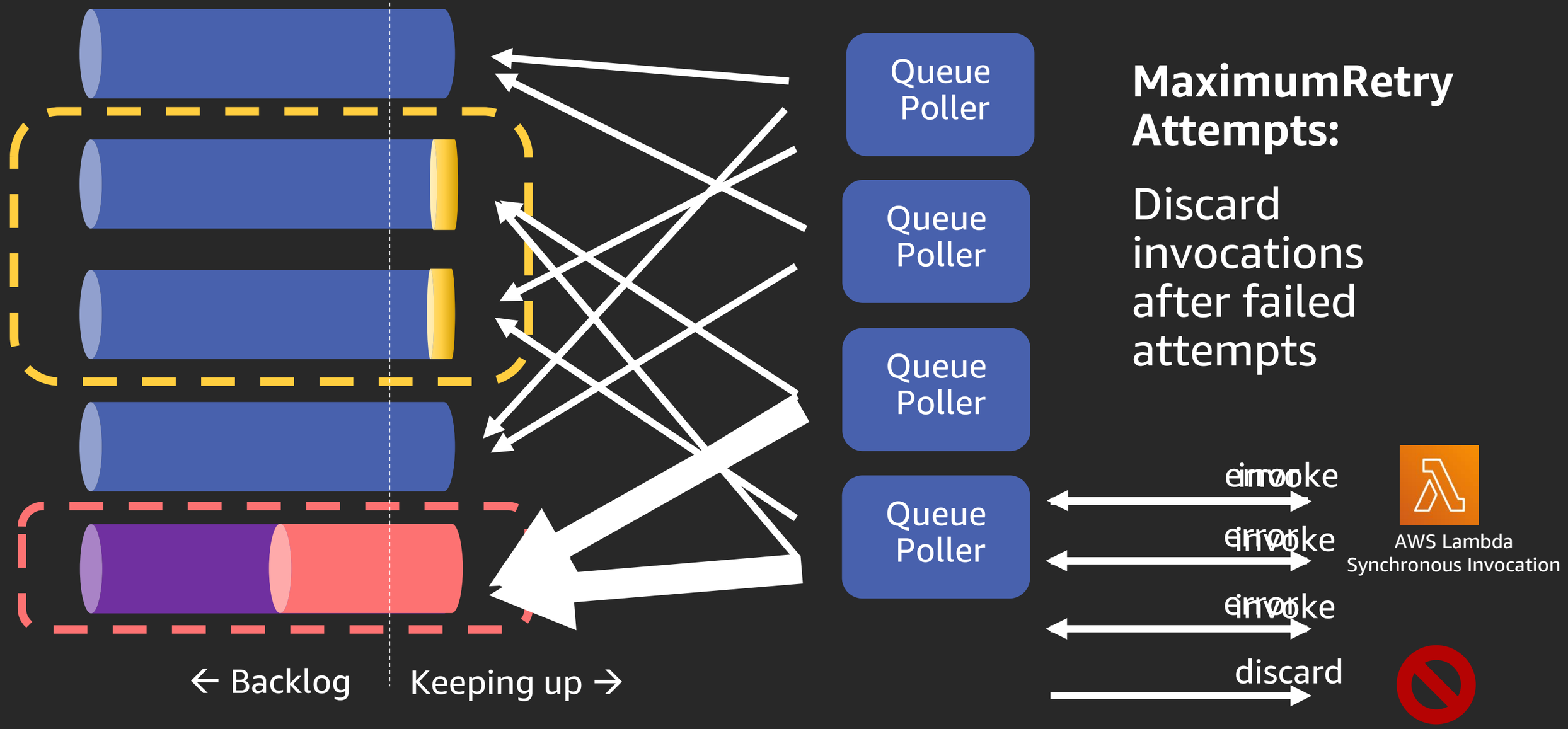
Processing backlogs



Processing backlogs



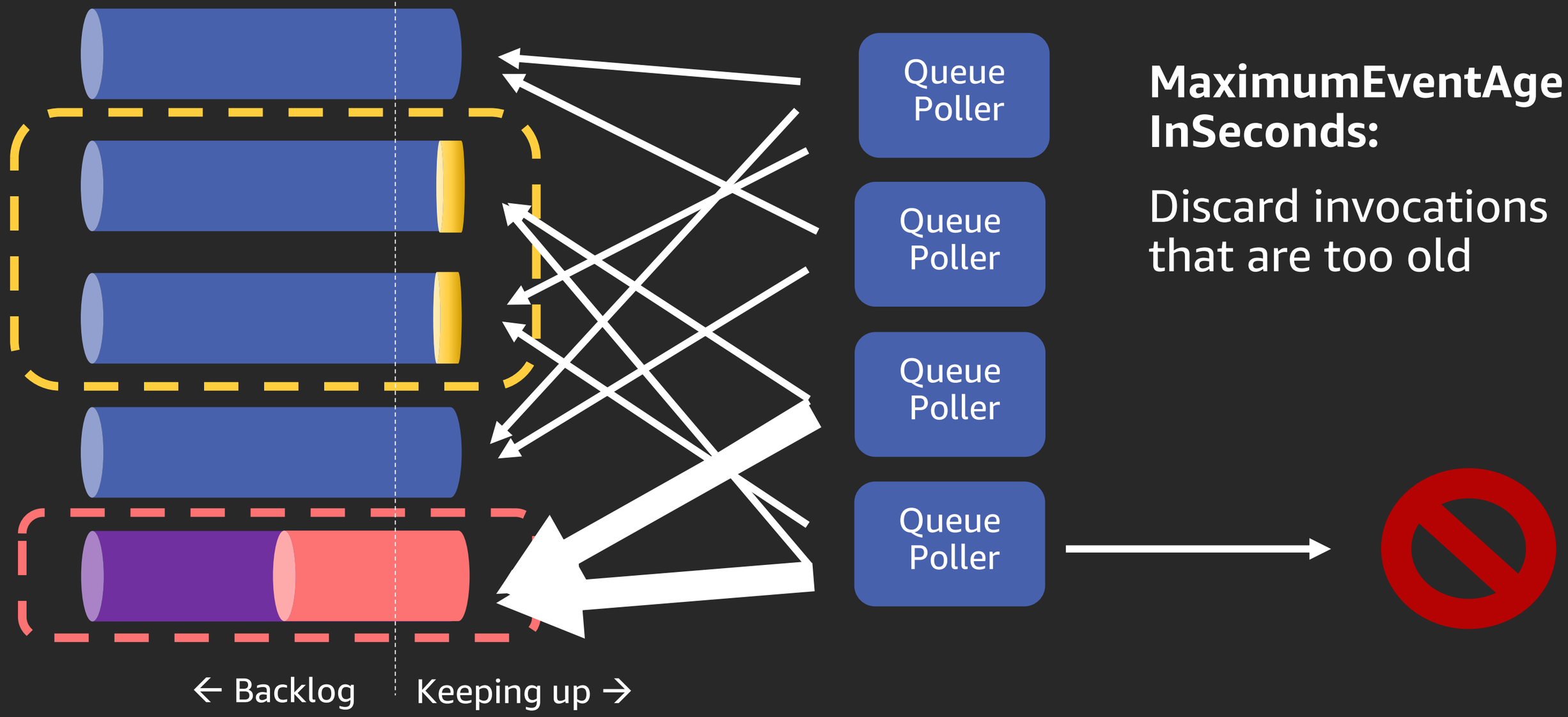
Prioritizing backlogs



MaximumRetry Attempts:

Discard invocations after failed attempts

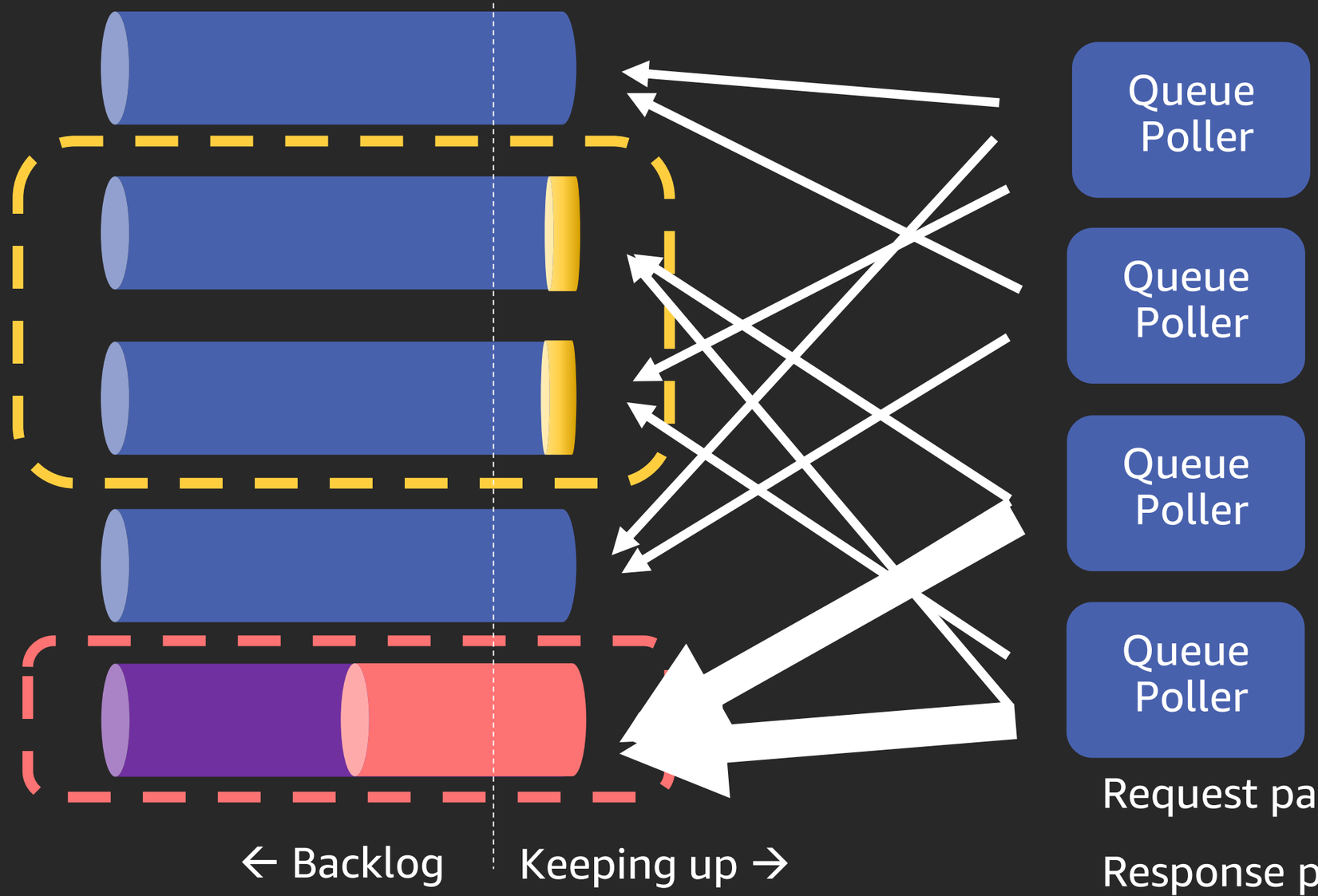
Prioritizing backlogs



MaximumEventAge InSeconds:

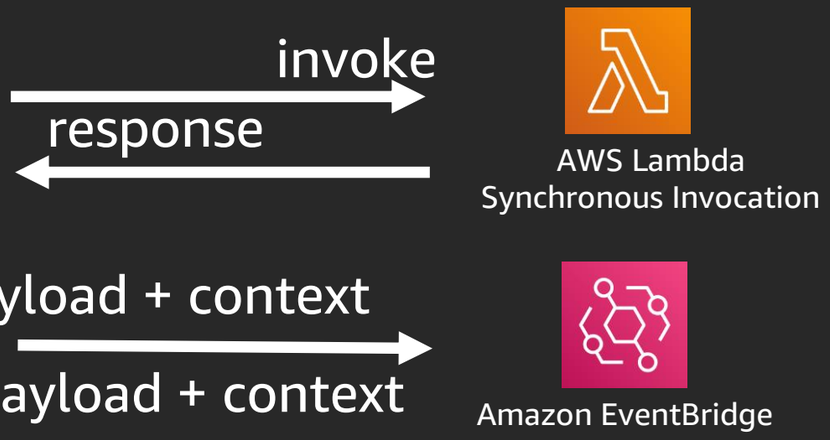
Discard invocations that are too old

Prioritizing backlogs



DestinationConfig:

Send invocation results to a destination



Visibility: destinations

OnSuccess lets you be notified in real time the results of your async invokes

What used to be fire and forget, becomes fire and be notified

Send hot fresh response payloads to your destination

Visibility: dwell times

Calculate dwell time from the event time on the payload

Example Amazon S3 record:


```
{
  "Records": [
    {
        "eventVersion":"2.0",
        "eventSource":"aws:s3",
        "awsRegion":"us-west-2",
        "eventTime":"1970-01-01T00:00:00.000Z",
        …
    }
  ]
}
```

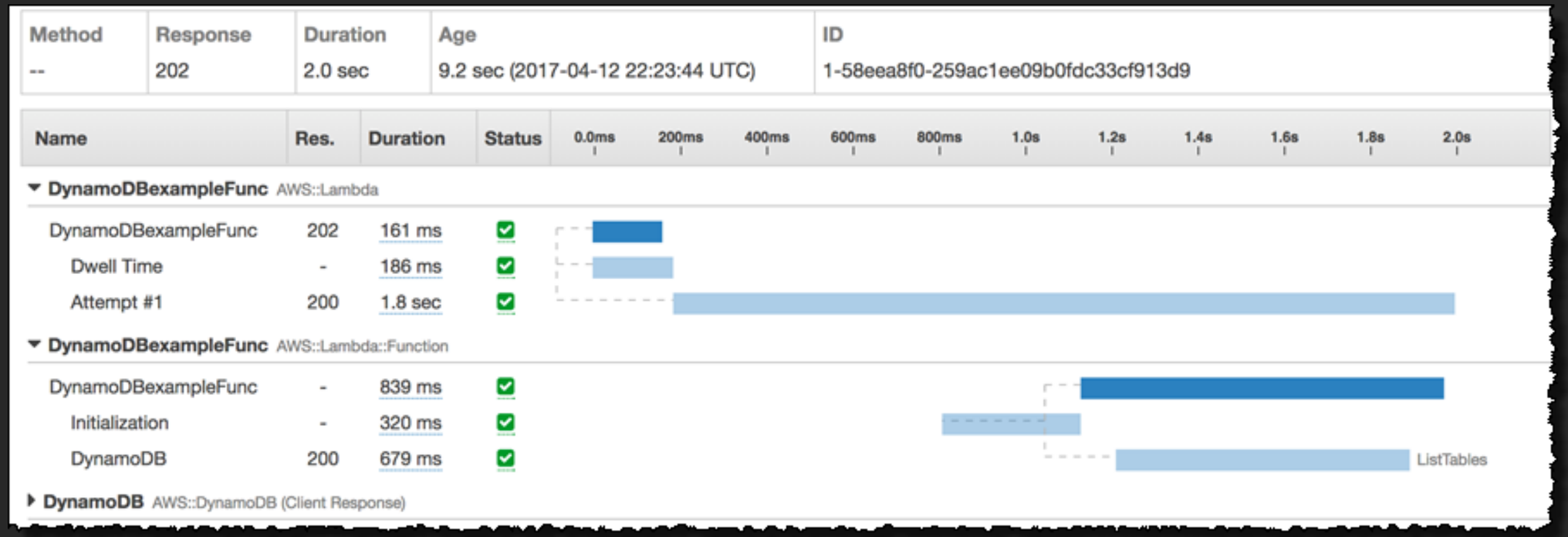
Visibility: dwell times

Calculate dwell time from the event time on the payload

```
…
cloudwatch = boto3.client('cloudwatch’)
dwelltime = getMsSince(event.Records[0].eventTime)
response = cloudwatch.put_metric_data(
  MetricData = [
        {
                'MetricName': 'KPIs’,
                'Dimensions': [
                        {
                                'Name’: ‘Function’,
                                'Value’: context.functionName
                        },
                ],
                'Unit': 'None’,
                'Value’: dwelltime
        },
  ],
Namespace = ‘MyApp’ )
…
```

Visibility: AWS X-Ray

Trace dwell time and number of attempts for a single invocation



Learn more

Related talks

[SVS326] [Managing events in your serverless application]

[SVS317] [Serverless stream processing pipeline best practices]

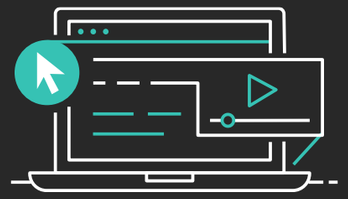
[API304] [Scalable serverless event-driven applications using Amazon Amazon SQS & AWS Lambda]

[SVS405] [A serverless journey: AWS Lambda under the hood]

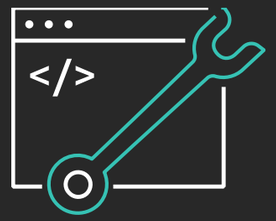
[SVS407] [Architecting and operating resilient serverless systems at scale]

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
Deep Dive: Lambda@Edge and Deep Dive on AWS Fargate



Additional digital and classroom trainings cover modern application development and computing

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

aws training and certification

Thank you!

Cecilia Deng

cicikendiggit@

AWS re:Invent

aws

Please complete the session survey in the mobile app.

AWS re:Invent

aws