FEARLESS AWS LAMBDAS

@johnchapin  symphonia.io

http://bit.ly/symph-qcon-fearless

john@symphonia.io

WHAT IS LAMBDA?
LAMBDA PERFORMANCE
LAMBDA + JAVA
LAMBDA + THE JVM
CHOOSING THE JVM

WHAT IS LAMBDA?

Serverless


O'REILLY
What is Serverless?
Understanding the Latest Advances in Cloud and Service-Based Architecture
Mike Roberts & John Chapin
Free e-book download!

Serverless traits

1. No management of long-lived host or application instances
2. Self auto-scaling/provisioning, based on load
3. Costs based on precise usage, zero usage = zero cost
4. Performance capability defined in terms other than host size/count
5. Implicit high availability

Backend as a Service (BaaS)

Functions as a Service (FaaS)

FaaS attributes

Runs user code

Event driven

Auto scaled/provisioned

Precise costs

Lambda platform

Java, Javascript, Python, C#

Integrated with other AWS services

Scales up in milliseconds

$ per GB/sec, 100ms increments

Lambda history

November 2014 - Node.js, 1GB, 60s

June 2015 - Java, 1.5GB

July 2015 - API Gateway

October 2015 - Python, 300s

November 2016 - C#, Lambda@Edge

Serverless ecosystem

API Gateway

DynamoDB

S3

Kinesis

SQS

SNS

Step Functions

X-Ray

Runtime environment

128MB to 1.5GB memory

2 virtual CPUs

500MB /tmp

STDOUT, STDERR to Cloudwatch Logs

Behind the scenes

Containers on EC2

Created on demand

Reaped when idle, old, or obsolete

LAMBDA PERFORMANCE

Performance challenges

Layers of abstraction

Diverse components

Error handling and retries

Event batching

Scaling

Benchmark Lambda

Java 8

Minimal dependencies

2 threads, 500 iterations of fib(30)

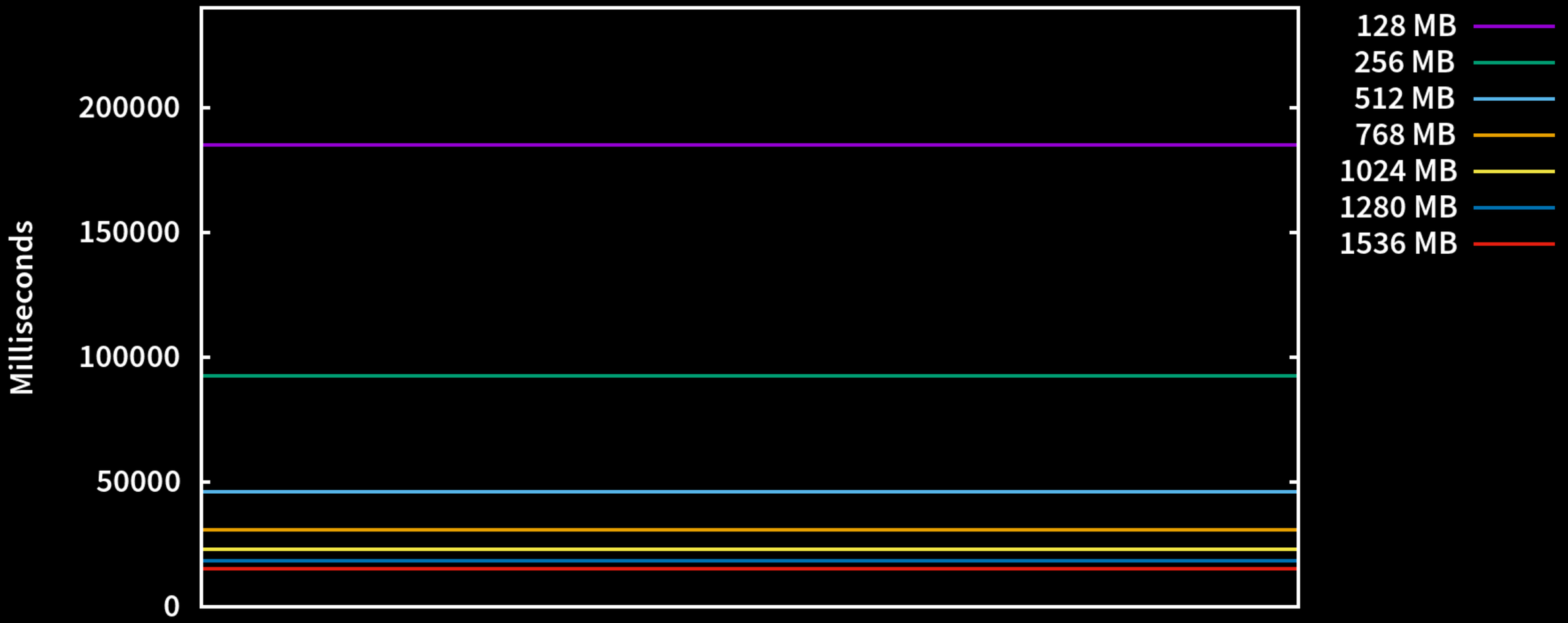
Various memory settings

Benchmark goal

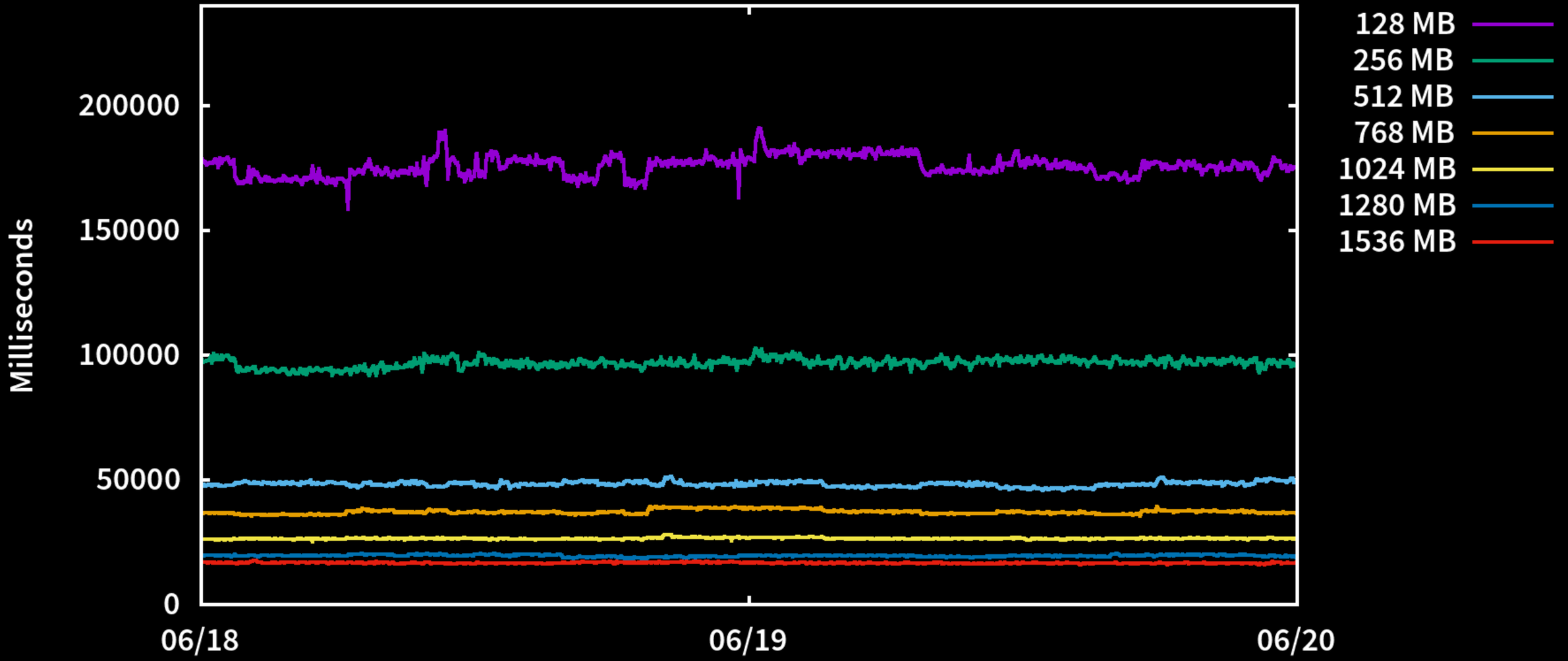
Show performance is proportional to memory setting

1.5GB *should* be 12x faster than 128MB

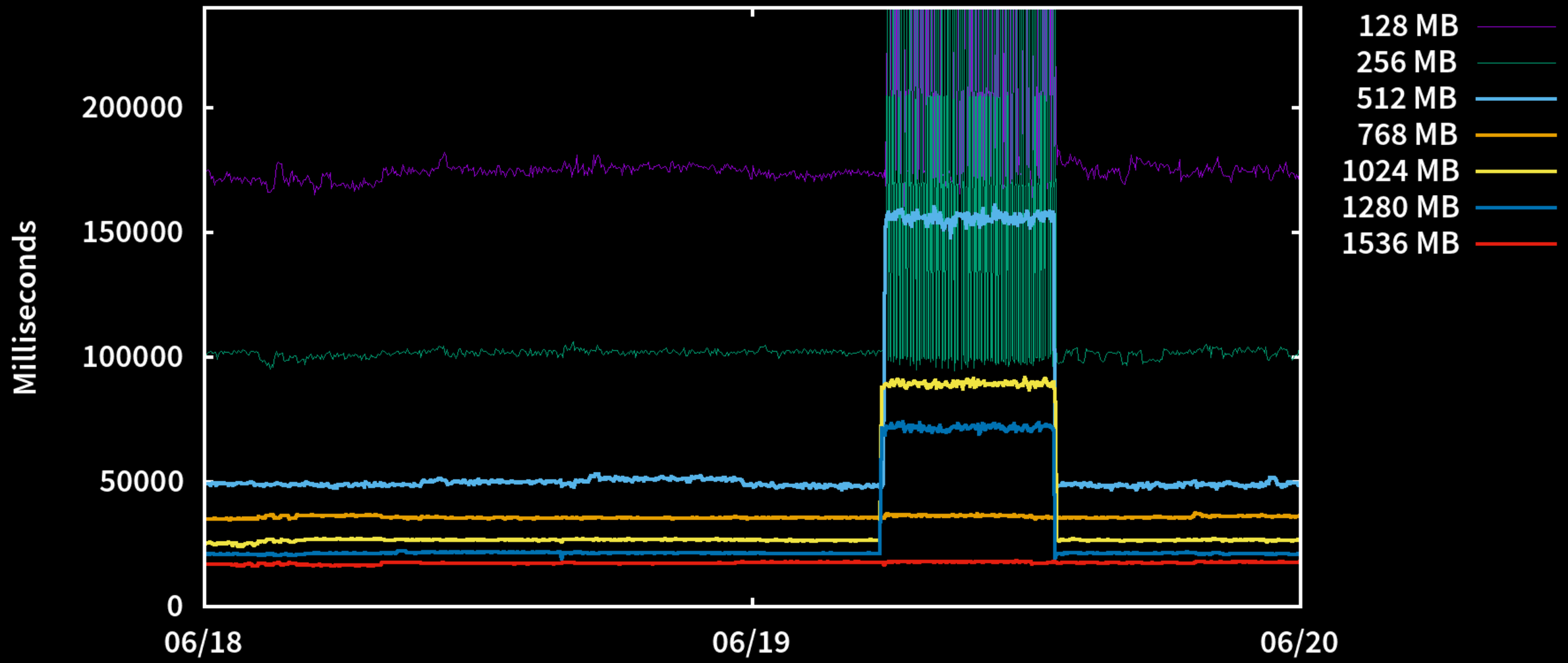
Expected performance



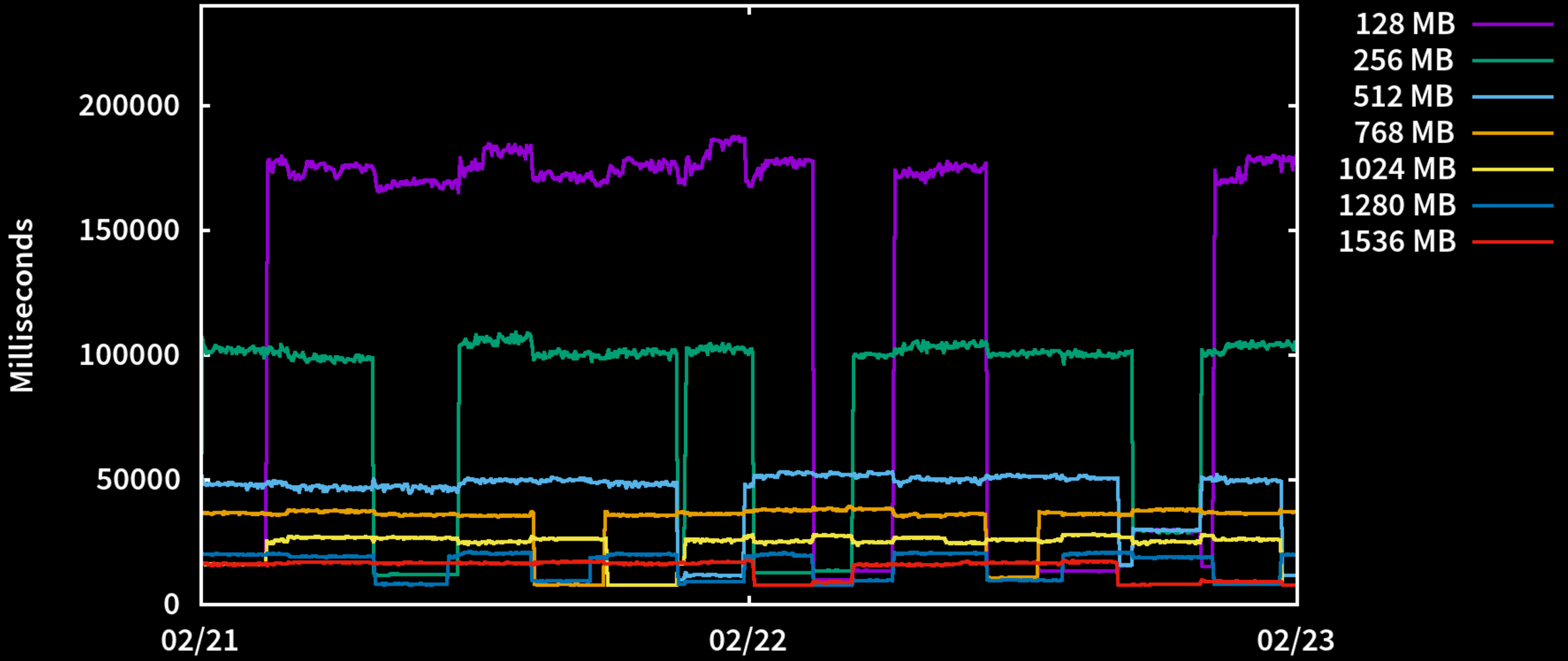
Measured performance (us-west-2)



US-EAST-1 strikes again!



WTF? (Feb 2017, us-west-2)



Benchmark conclusions

Memory = consistency (mostly)

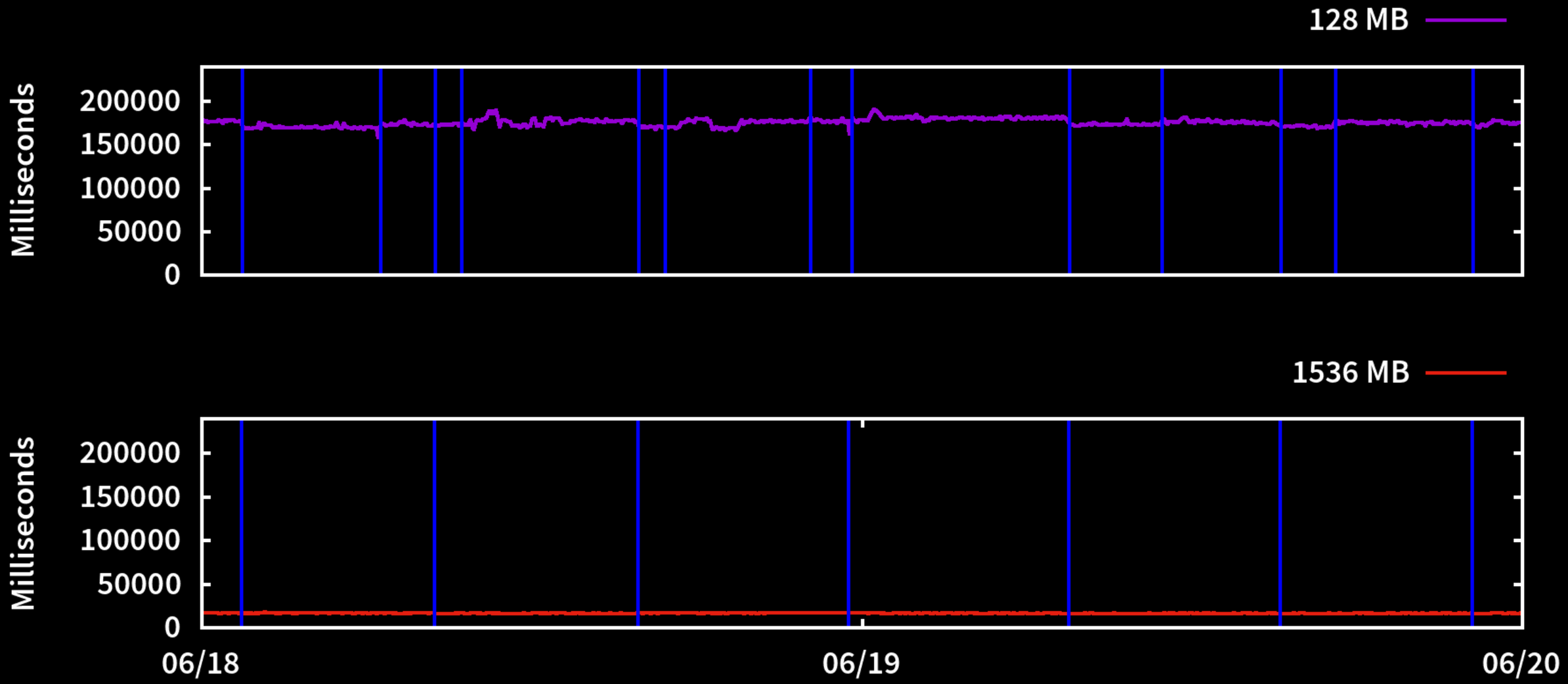
Benchmark over long periods of time

Benchmark in multiple regions

Cold starts

1. Platform receives event
2. Platform inits container
3. Container inits language runtime
4. Language runtime inits user code
5. Platform passes event to container

Cold starts (us-west-2)



Cold starts

2 days of data (us-west-2)

128MB - 1.8% of the time

1.5GB - 0.97% of the time

LAMBDA + JAVA

```
io.symphonia.Lambda::handler
```

```java
package io.symphonia;

public class Lambda {

    public String handler(String input) {
        return input;
    }

}
```

```
package io.symphonia;

public class Lambda {

    public void handler(int n) {
        // do something ...
    }

}
```

```java
package io.symphonia;

import java.io.InputStream;
import java.io.OutputStream;

public class Lambda {

    public void handler(InputStream input, OutputStream output) {
        // Read from the InputStream
        // Write to the OutputStream
    }

}
```

```java
package io.symphonia;

import com.amazonaws.services.lambda.runtime.CognitoIdentity;
import com.amazonaws.services.lambda.runtime.Context;

public class Lambda {

    public String handler(String input, Context context) {

        // Inspect Context
        CognitoIdentity identity = context.getIdentity();
        int remaining = context.getRemainingTimeInMillis();

        // do something ...

        return input;
    }

}
```

Handling input

Lambda runtime will deserialize JSON -> POJOs

Include event POJOs as source

Avoid large dependencies just for event classes

- AWS event types are scattered across libraries

Or, parse incoming JSON yourself (InputStream)

```java
package io.symphonia;

public class Lambda {

    public String handler(MyInputObject inputObject) {
        return inputObject.getField();
    }

    public static class MyInputObject {
        String field;

        public String getField() {
            return field;
        }

        public void setField(String field) {
            this.field = field;
        }
    }
}
```

```java
package io.symphonia;

import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import java.util.UUID;

public class Lambda {

    private AmazonS3 s3client;
    private String bucket = "myBucket";

    public Lambda() {
        s3client = AmazonS3ClientBuilder.defaultClient();
    }

    public void handler(String key) {
        s3client.putObject(bucket, key, UUID.randomUUID().toString());
    }
}
```

Deployment

Deployment artifact is a zip file

For Java, an uberjar

Use mvn-shade-plugin or similar

- Your code, plus your dependencies

Multi-Lambda applications

Multi-module Maven project

AWS Bill-of-materials (BOM)

- dependencyManagement for AWS SDKs

Serverless Application Model (SAM)

LAMBDA + THE JVM

Lambda’s JVM runtime

OpenJDK 1.8 Server VM

-XX:MaxHeapSize = 85% of configured Lambda memory

-XX:+UseSerialGC

-XX:+TieredCompilation

-Xshare:on

JVM cold starts

Class loading

Initialization

- Constructors, static initialization blocks

Alternative language runtime loading (Clojure!)

JIT compilation

The Lambda diet

Fewer classes = faster startup

- Ruthlessly cull dependencies

- AWS libraries can be bloated!

mvn dependency:tree, sbt dependencyStats

Cloudwatch Logs

System.out/err output goes to Cloudwatch Logs

One “log group” per Lambda (by default)

Within “log group”, one “log stream” per container

From Cloudwatch, can aggregate and/or forward

System.out.nope

System.out.println is bad for the normal reasons

Real logging is better

Lambda runtime can add RequestId to Log4J logs

aws-lambda-java-log4j uses Log4J 1.2 ☹️

lambda-logging

SLF4J + Logback 😀

Sane default configuration w/ AWS RequestId

Open Source (Apache 2 license)

- io.symphonia/lambda-logging “1.0.0”

- github.com/symphoniacloud/lambda-monitoring/

Cloudwatch Metrics

No built-in business metrics

Lambda platform metrics

- Errors, Duration, Invocations, Throttles

Naive metrics collection approach is dangerous!

- Cloudwatch has account-level API limits 🔥

Cloudwatch Metric Filters

Built into Cloudwatch! Scalable!

Scrape Cloudwatch Logs data using special (finicky) patterns

Generates and post Cloudwatch metrics

lambda-metrics

Codahale metrics and lambda-logging

Maven plugin builds Metric Filters to scrape logs, post to CW

Open Source (Apache 2 license)

- io.symphonia/lambda-metrics “1.0.0”
- github.com/symphoniacloud/lambda-monitoring

CHOOSING THE JVM

Ideal application

Latency tolerant

Regularly invoked

Computationally intensive

TL;DR

Use Lambda's JVM runtime

Reduce and amortize cold start impact

Benchmark extensively

Use real logging and metrics

http://bit.ly/symph-qcon-fearless

john@symphonia.io