Cost-optimize your workloads on Amazon EKS

Borja Pérez Guasch **bpguasch@amazon.es**

Specialist Solutions Architect, Flexible Compute

Agenda

EC2 Spot & allocation strategies

Graviton

Amazon EKS & Karpenter

Kubecost

Q&A

EC2 Spot & allocation strategies

EC2 Spot knowledge check

Spot instance s
are based on custo
bidding?

Pricing for Spot is published by AWS and determined by long term supply and demand trends (slow and predictable)

Spot is for cu
ooking for "extre
cost" solutions

Spot is for customers looking to optimize their EC2 expenses for Spot-friendly workloads

Spot is tim d
difficult to imple

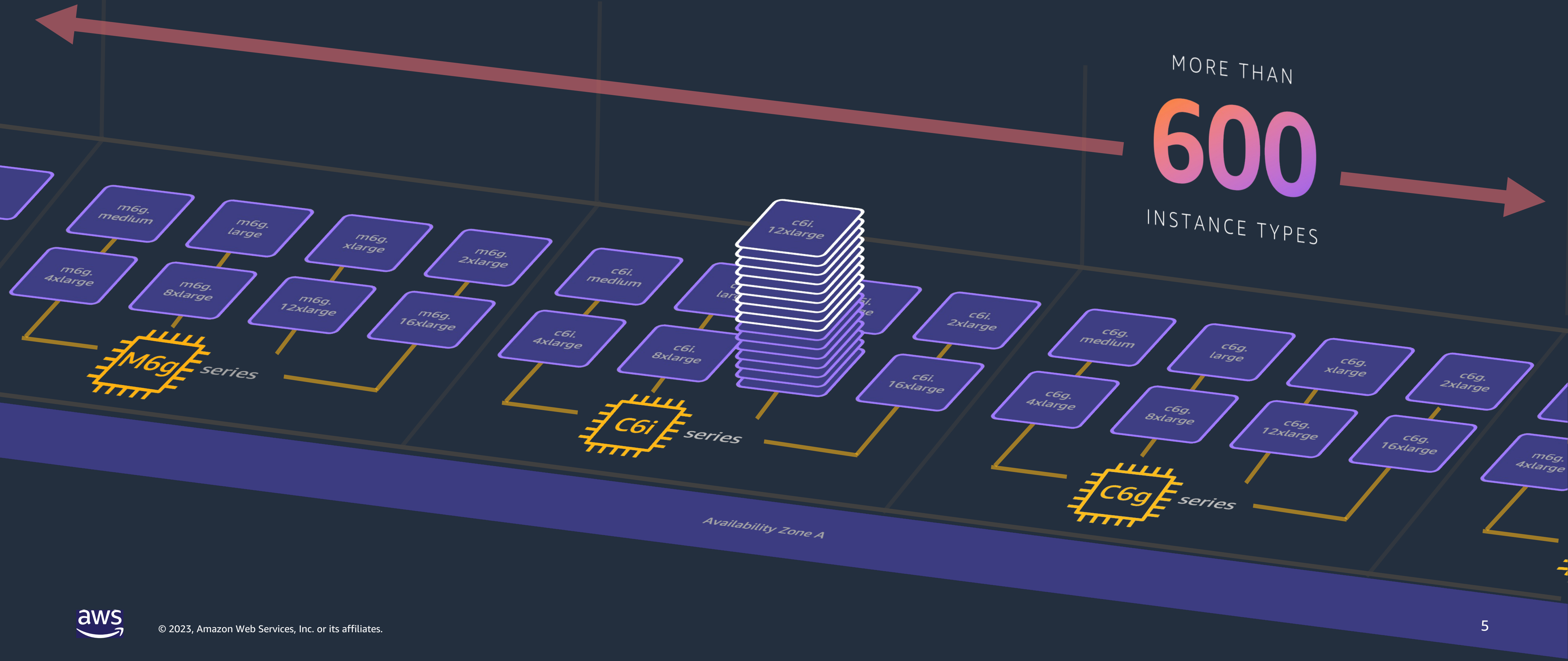
Integrations with AWS Services and our partners make it much easier to adopt Spot

Spot insta are
specific type
instances in some A
regions

Spot instances are the same instance types as On Demand and available in All Regions

Spot is inte for
non-productio
workloads

Spot is appropriate non-production and production workloads



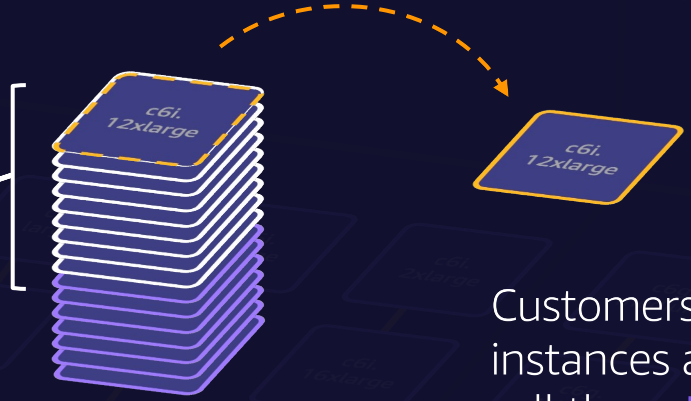
MORE THAN

600

INSTANCE TYPES

The picture represents an example of how a particular instance type-size could be in use and have spare capacity

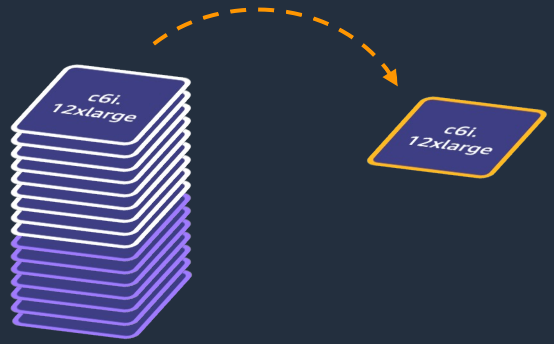
The white squares here represent the spare capacity of an instance type-size and the group of spare instances we call Spare Capacity Pools.



Customers can provision spare instances at a discounted rate – we call these EC2 Spot instances.

EC2 Spot instances offer up to a 90% discount compared to on-demand.

How Spot Instances Work ...



Spot instances

Spare EC2 Capacity

Provisioned from spare-capacity, uses same infrastructure as On-Demand (OD)

Pricing

Up to 90% off compared to On-Demand

Not bidding - Pricing is based on long-term supply and demand, smooth and predictable

Interruption

only interrupted if OD needs capacity

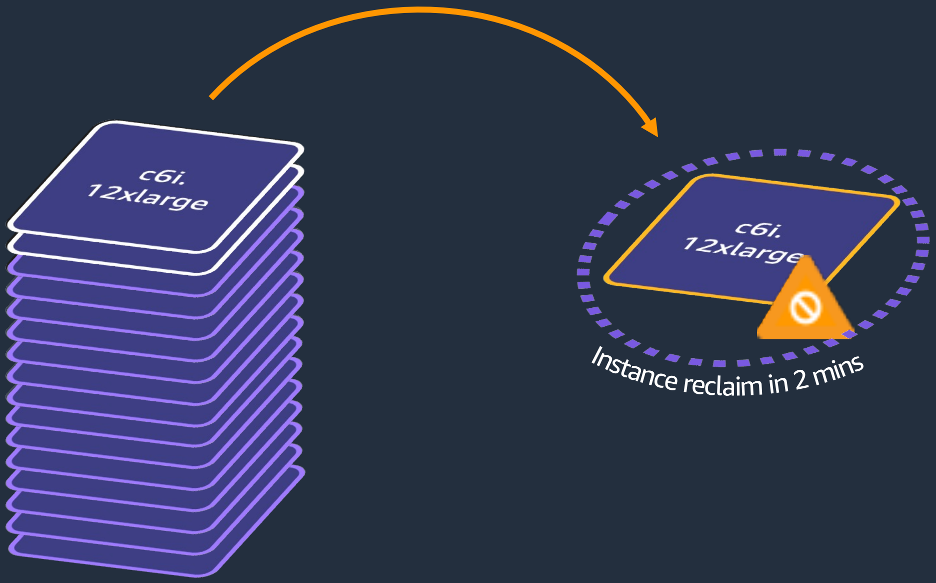
AWS can reclaim with 2-minute notice; issues two types of notifications to help handle interruptions

Diversification

and flexibility is key

Make use of different instance types, sizes, Availability Zones, and times Regions

Interruptions



By the nature of Spot as spare-capacity, a Spot instance can be interrupted if the instance is needed by On-Demand.

AWS provides two types of notifications to enable you to handle the response in an automated way:

EC2 instance rebalance recommendation *(proactive)*

- Spot instance is at elevated risk of interruption
- Built in support for AWS integrations such as EC2 Auto Scaling and EKS Managed Node Groups

Spot instance termination notice *(reactive)*

- Interruption of instance will happen in 2 minutes, adjust your workload appropriately
- Built in support for AWS integrations such as EC2 Auto Scaling and EKS Managed Node Groups

Historically **95%** of the Spot instances launched in the last 3 months completed without interruption

Diversification across instances reduces interruptions

Diversification + Flexibility is key…

1. Instance Flexible (Type + Size!)

- Use as many instance types as possible that suits the workload
- Multiple instance types are key to resilient clusters
- Attribute based instance selection helps you choose ranges of instance types

2. Availability Zone Flexible

- Capacity exists differently across availability zones, and multiplies potential capacity based on how many AZs are used

3. Region Flexible

- Different capacity exists across regions
- HPC customers and high production Spot users may span regions

4. Time Flexible

- Capacity can differ based on time/region usage, it is sometimes worth exploring running workloads at different times to utilize spare capacity

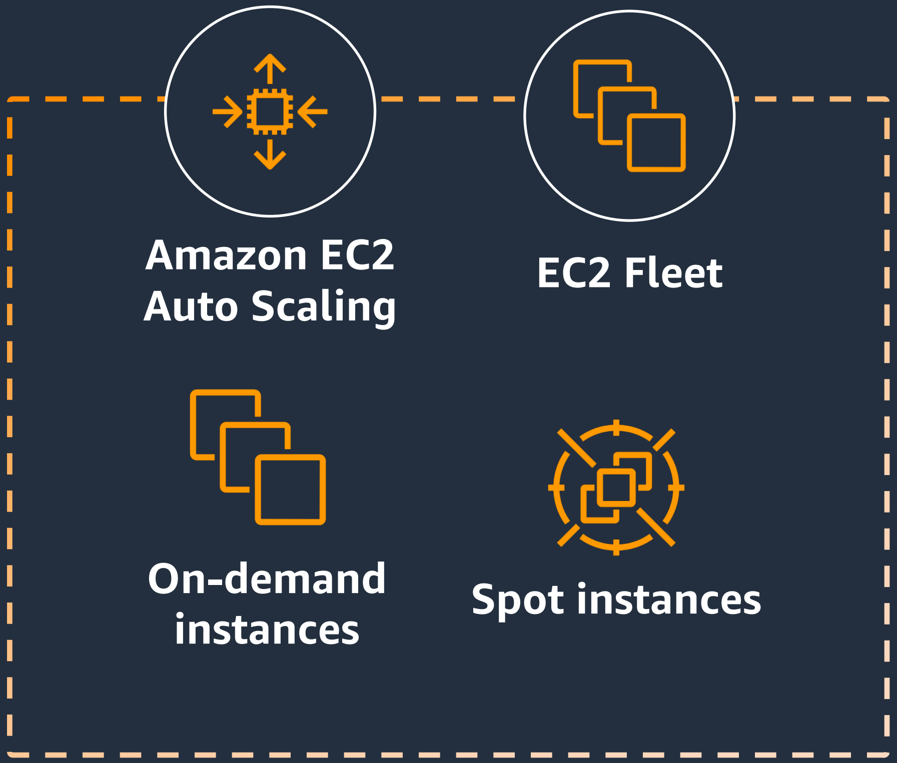
Use the right EC2 Spot allocation strategy

On-demand allocation

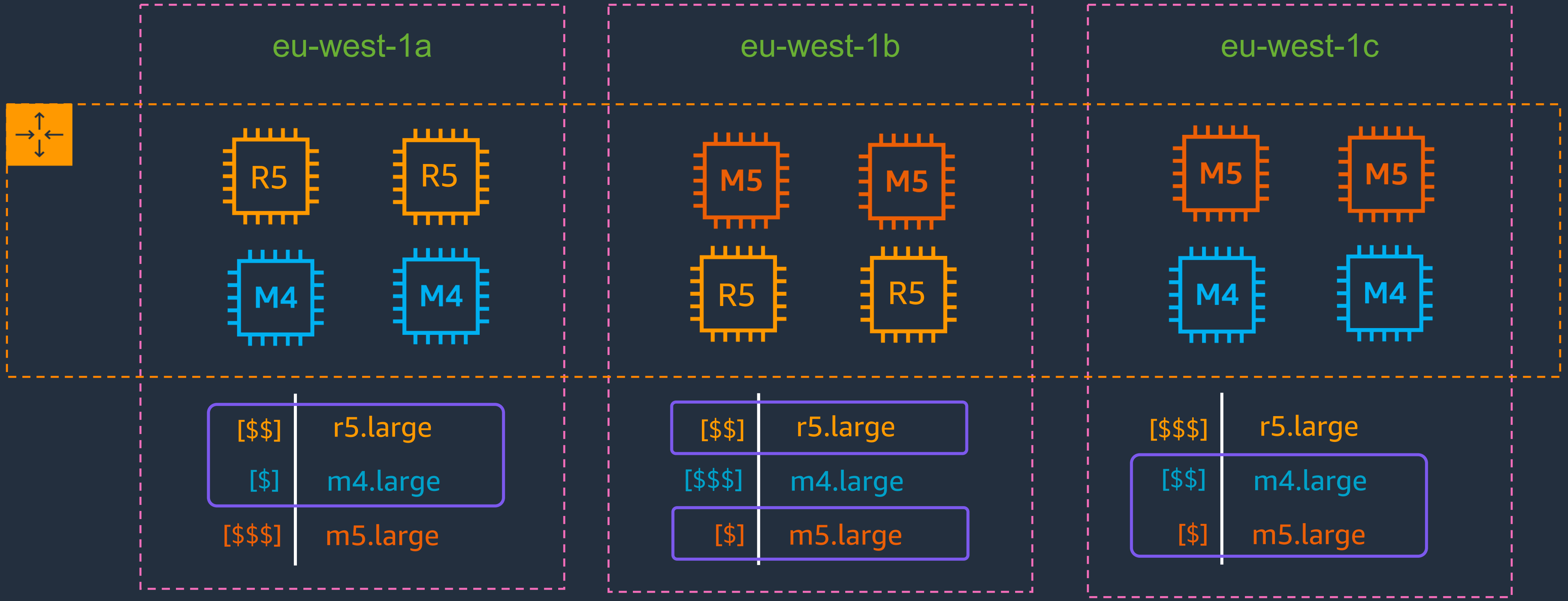
- Prioritized
- Lowest-price (default for mixed instance groups)

Spot allocation

- Lowest-price
- Capacity-optimized
- Price-capacity-optimized (recommended) New



Allocation Strategy: lowest-price

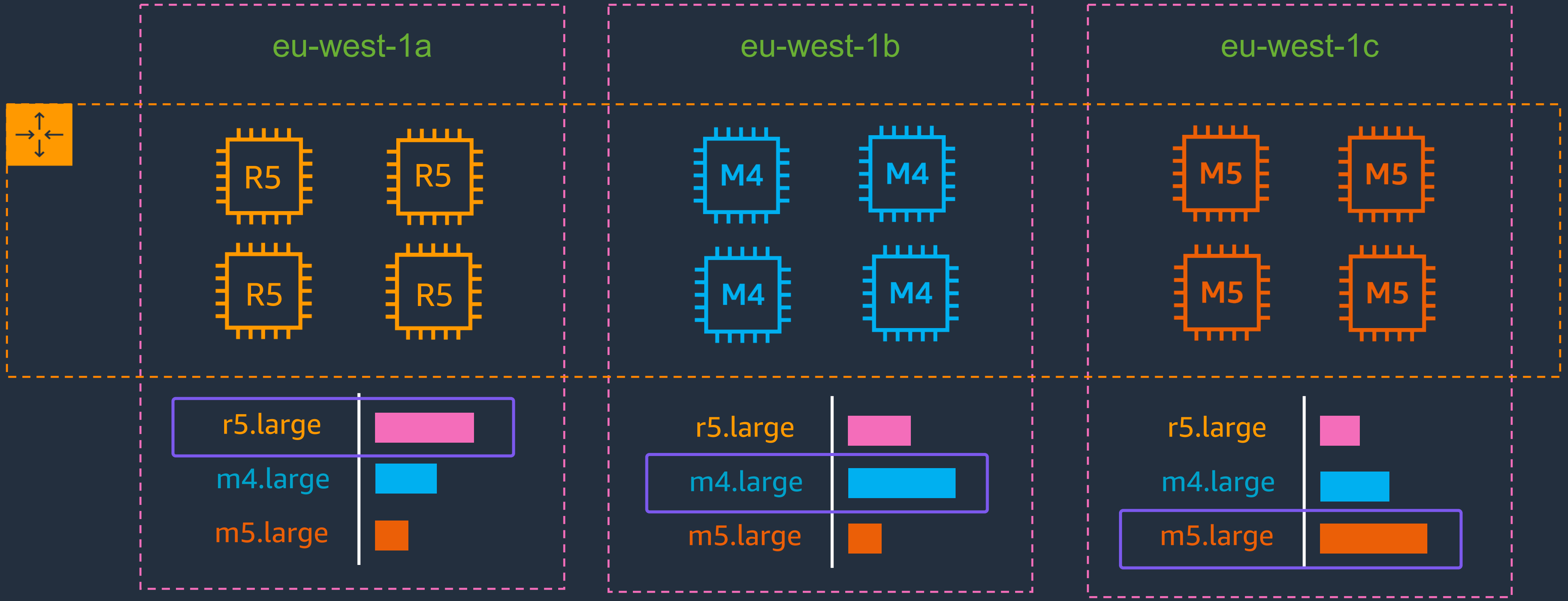


SpotAllocationStrategy: lowest-price SpotInstancePools:2 (default)

Overrides: [“r5.large”, “m4.large”, ”m5.large”]

Desired capacity: 12 OnDemandBaseCapacity: 0 OnDemandPercentageAboveCapacity: 0

Allocation Strategy: capacity-optimized

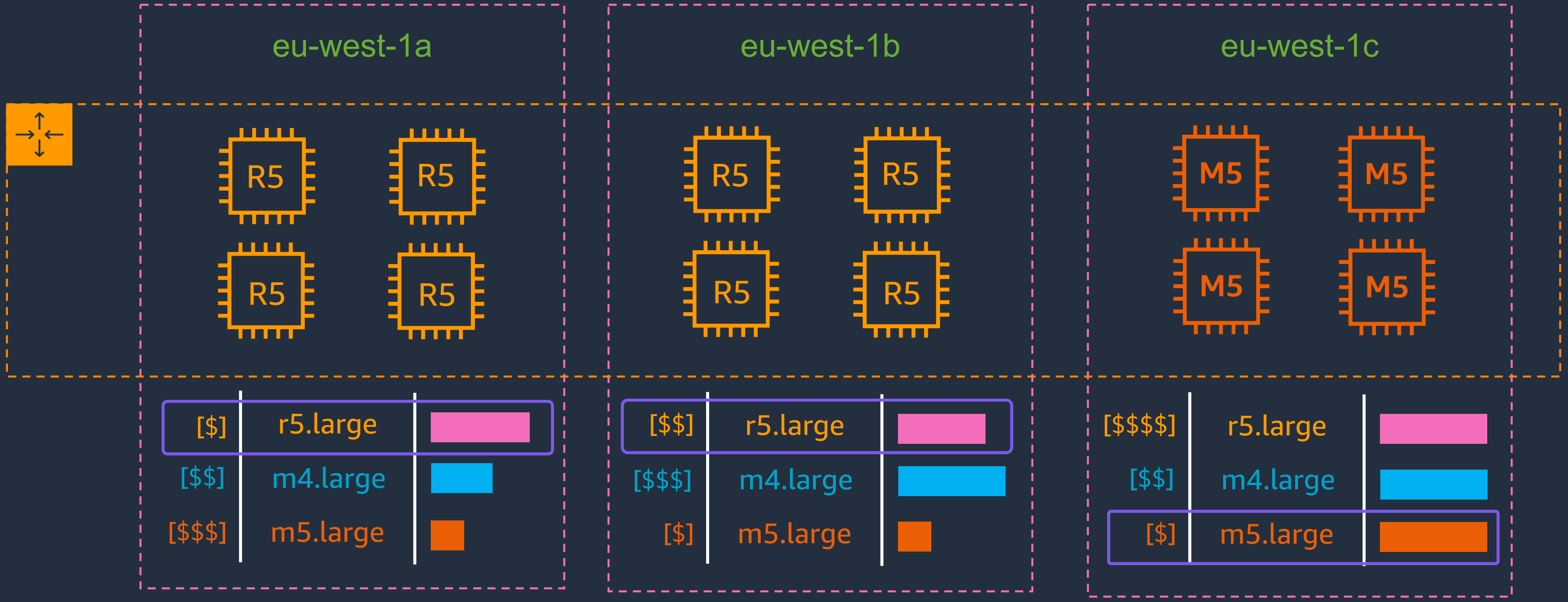


SpotAllocationStrategy: capacity-optimized

Overrides: [“r5.large”, “m4.large”, ”m5.large”]

Desired capacity: 12 OnDemandBaseCapacity: 0 OnDemandPercentageAboveCapacity: 0

Allocation Strategy: price-capacity-optimized



`SpotAllocationStrategy: price-capacity-optimized`

`Overrides: [“r5.large”, “m4.large”, ”m5.large”]`

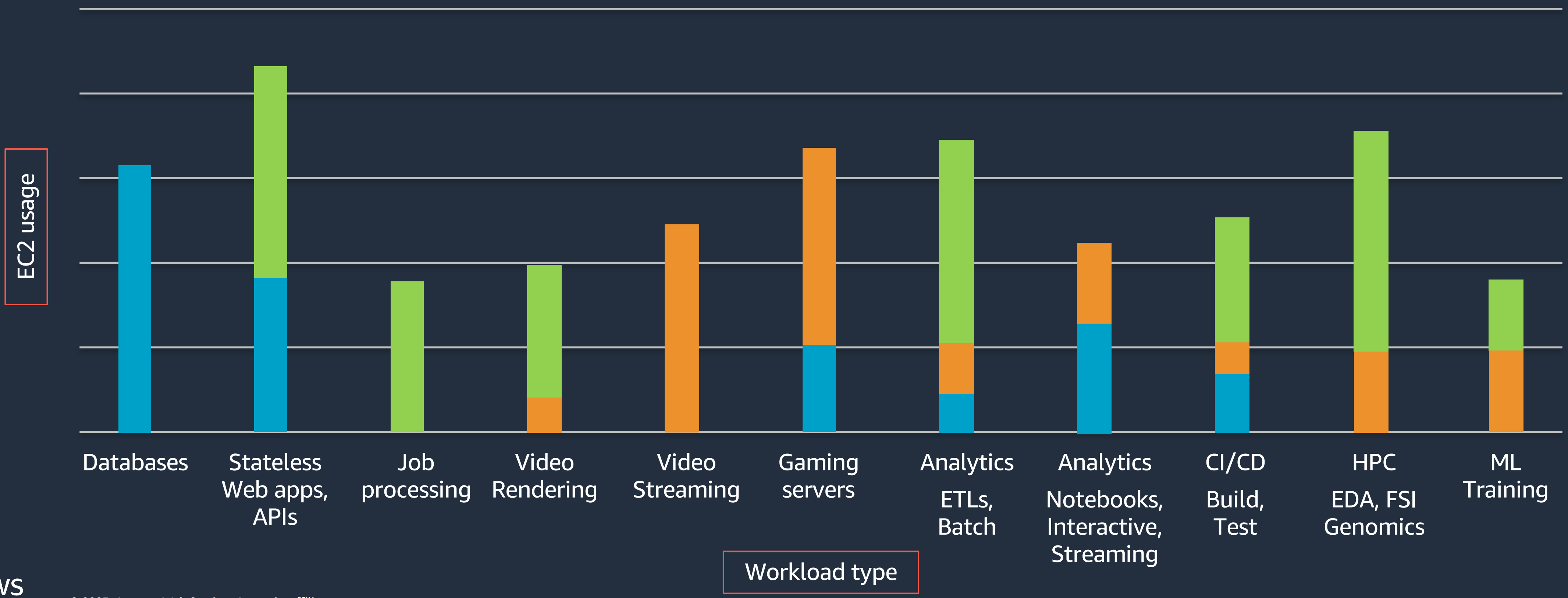
`Desired capacity: 12  OnDemandBaseCapacity: 0   OnDemandPercentageAboveCapacity: 0`

Adapt your EC2 Purchasing Strategy to your Workload

USE RIs AND SAVINGS PLANS FOR KNOWN/ STEADY-STATE WORKLOADS

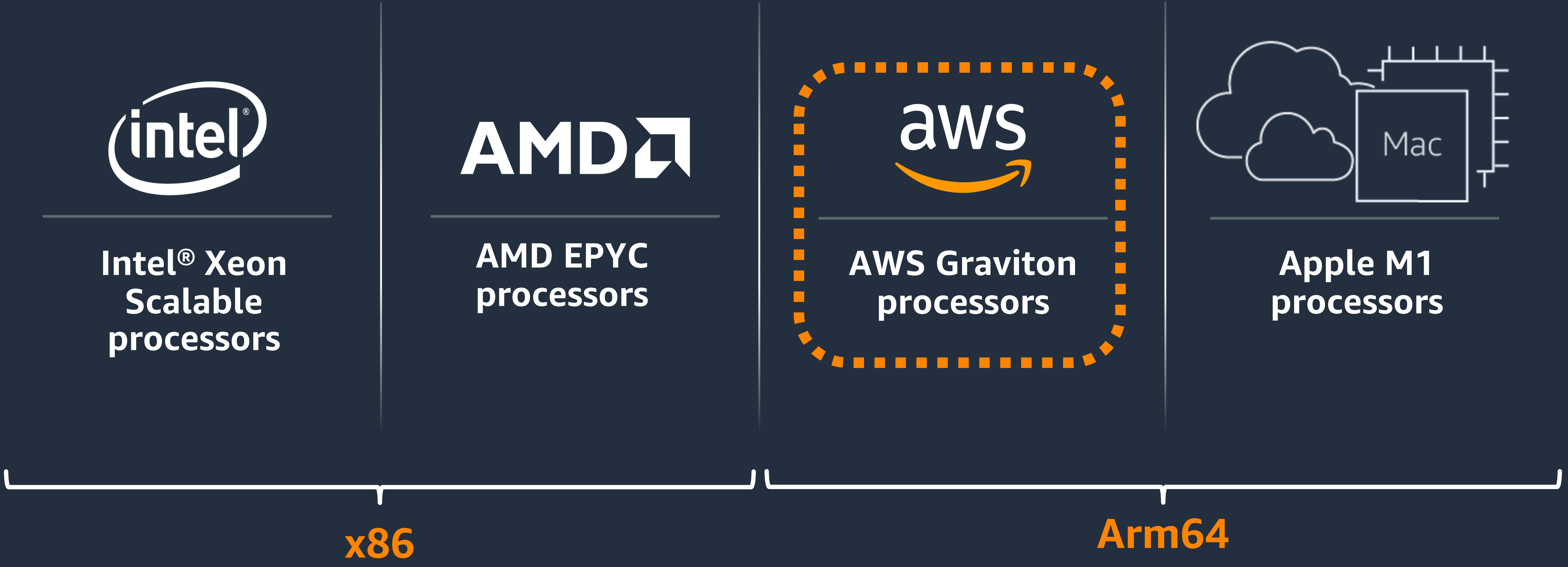
SCALE USING ON-DEMAND FOR NEW OR STATEFUL SPIKY WORKLOADS

SCALE USING SPOT INSTANCES FOR FLEXIBLE, FAULT-TOLERANT WORKLOADS



AWS Graviton

Broadest choice of processors



Intel® Xeon Scalable processors

AMD EPYC processors

AWS Graviton processors

Apple M1 processors

x86

Arm64

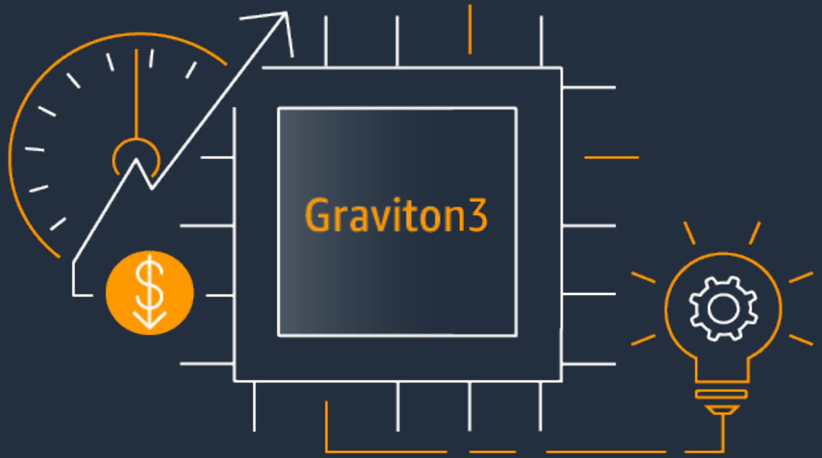
AWS Graviton

Up to 40% better price-performance for a broad spectrum of workloads

Up to 20% less expensive than comparable x86-based instances

Up to 60% more energy efficient vs. comparable x86-based instances

Graviton3 and Amazon EC2 C7g instances



Up to 25% better performance compared to Graviton2

Up to 2x higher floating-point performance, up to 2x faster cryptographic workload performance, and up to 3x better machine learning performance compared to Graviton2

First in the cloud to feature DDR5 memory

60% more energy efficient over comparable EC2 instances

C7g instances provide the best price performance for compute-intensive workloads in Amazon EC2

AWS Graviton: Broad workload applicability

Web and gaming servers

Open-source databases

High performance computing

In-memory caches

Media encoding

Electronic design automation

Analytics

Microservices

AWS managed services supporting Graviton

Databases

- Amazon DocumentDB
- Amazon Aurora
- Amazon RDS
- Amazon Elasticache
- Amazon MemoryDB
- Amazon Neptune

Analytics

- Amazon OpenSearch
- Amazon EMR

Compute

- AWS Lambda
- AWS Fargate (for ECS)
- AWS Elastic Beanstalk

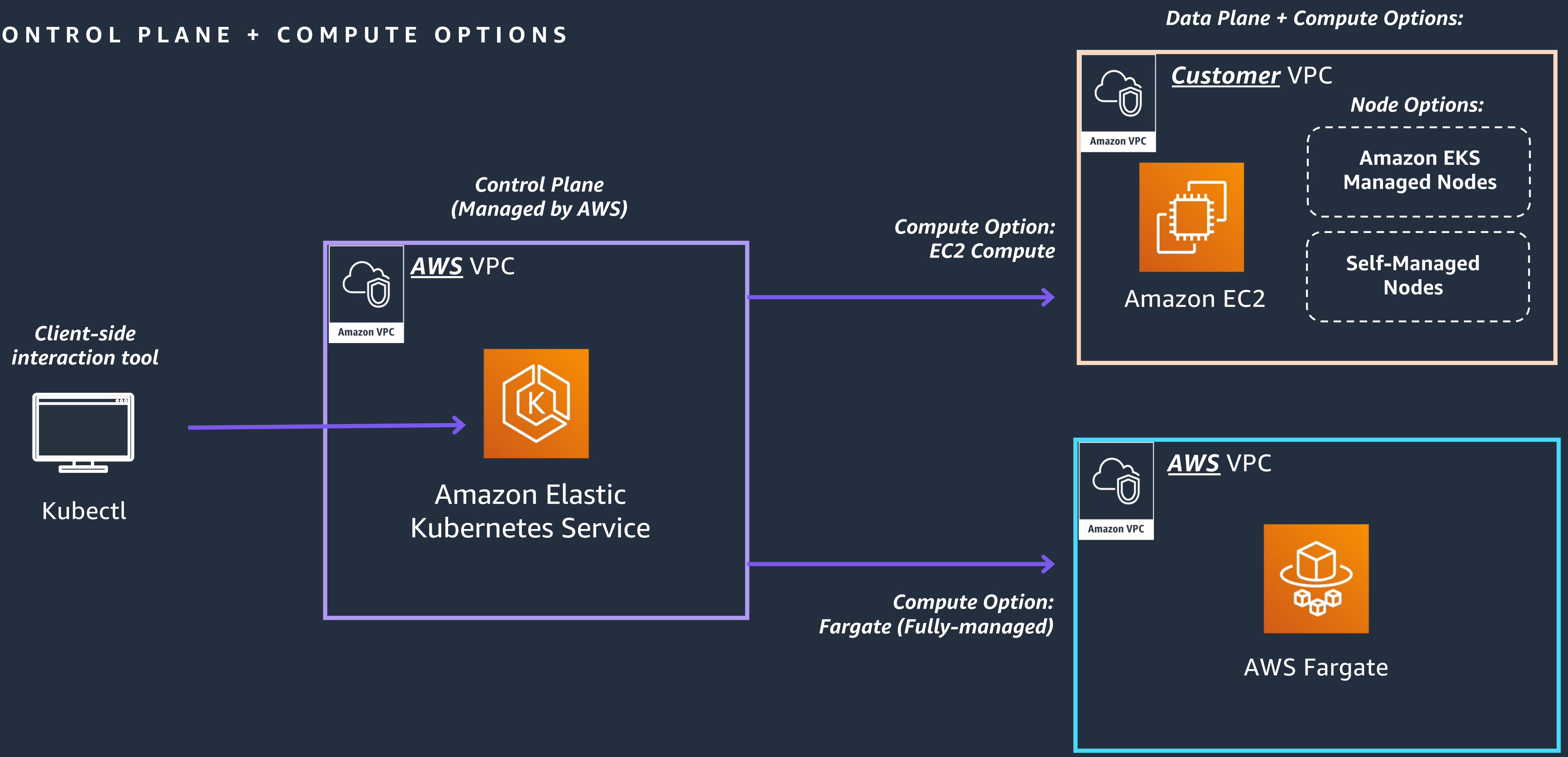
Machine Learning

- Amazon SageMaker

Amazon EKS & Karpenter

Amazon Elastic Kubernetes Service

CONTROL PLANE + COMPUTE OPTIONS



Client-side interaction tool

Kubectl

Control Plane (Managed by AWS)

AWS VPC

Amazon Elastic Kubernetes Service

Compute Option: EC2 Compute

Compute Option: Fargate (Fully-managed)

Data Plane + Compute Options:

Customer VPC

Amazon EC2

Node Options:

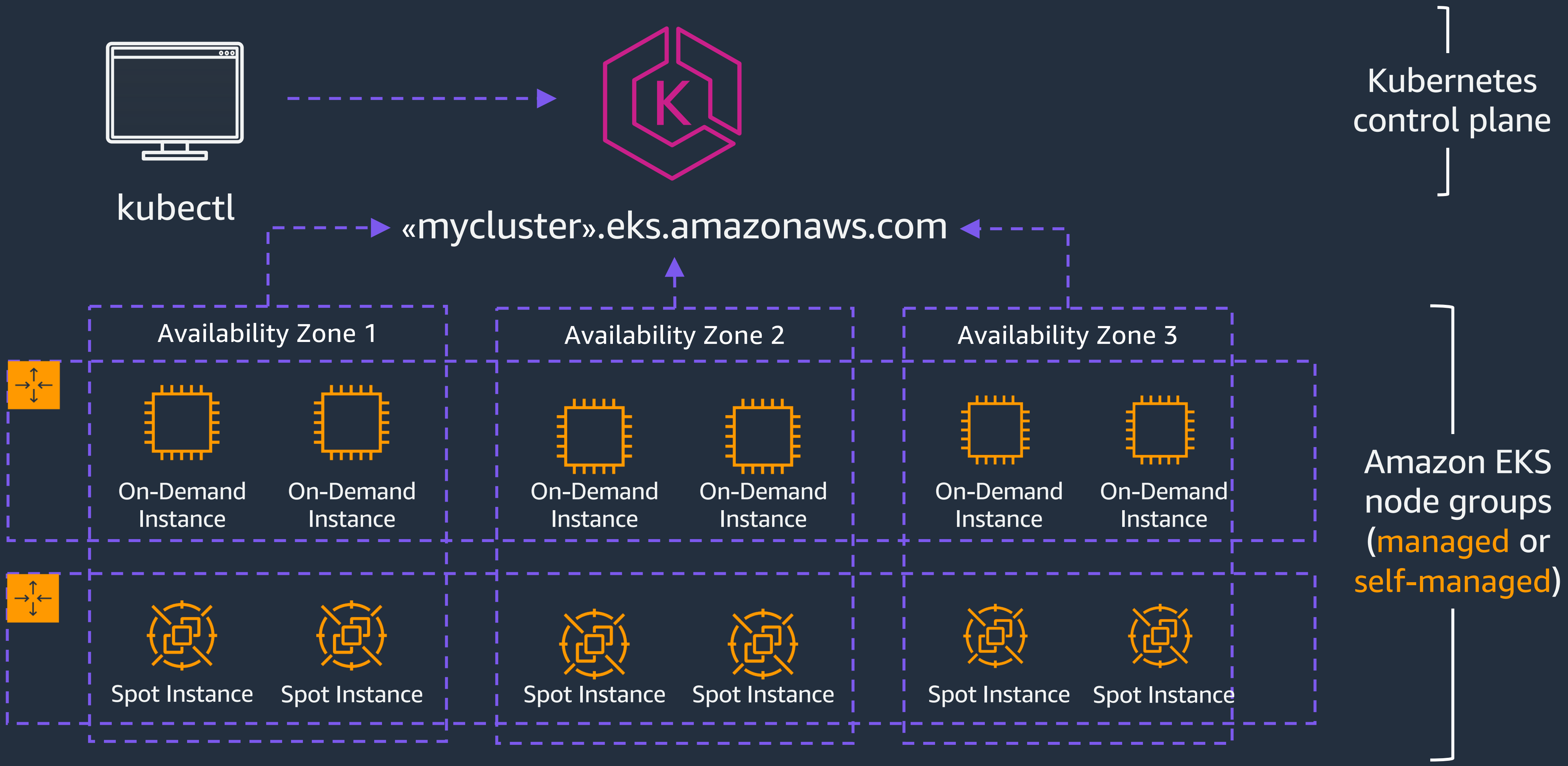
Amazon EKS Managed Nodes

Self-Managed Nodes

AWS VPC

AWS Fargate

EKS Cluster Architecture and Node Provisioning



Karpenter

EKS CLUSTER NODE PROVISIONER

Application First Infrastructure

Node provisioning based on Pod requirements

Default infrastructure provisioning

Simplified Configuration

Single configuration with On-demand and Spot purchasing options and diverse instance types

Track nodes using native Kubernetes labels

Diversify across Spot and On-Demand

Simplified diversification across purchase options

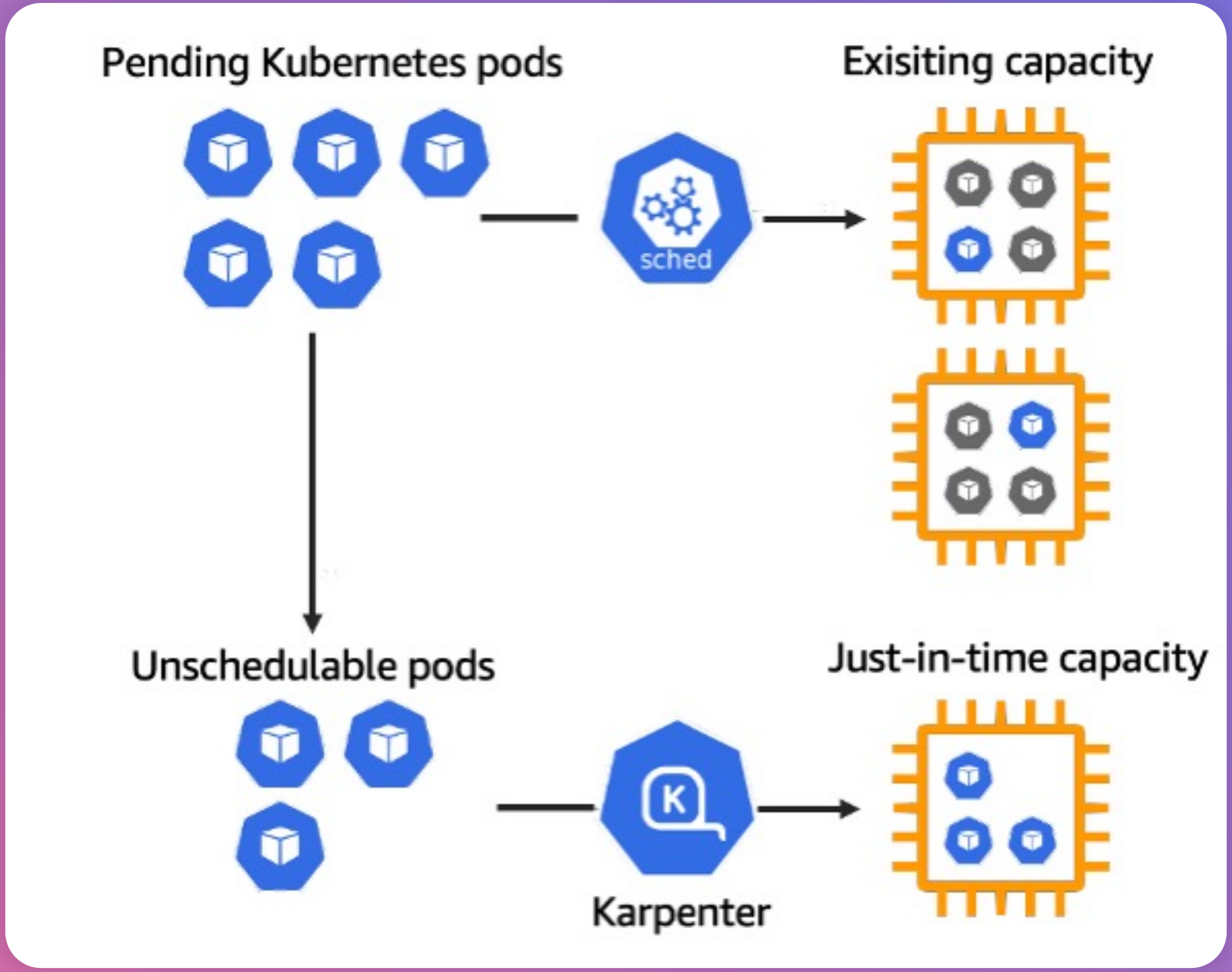
Launch template support

Custom configuration and custom AMIs for your Kubernetes nodes

Karpenter scale-up

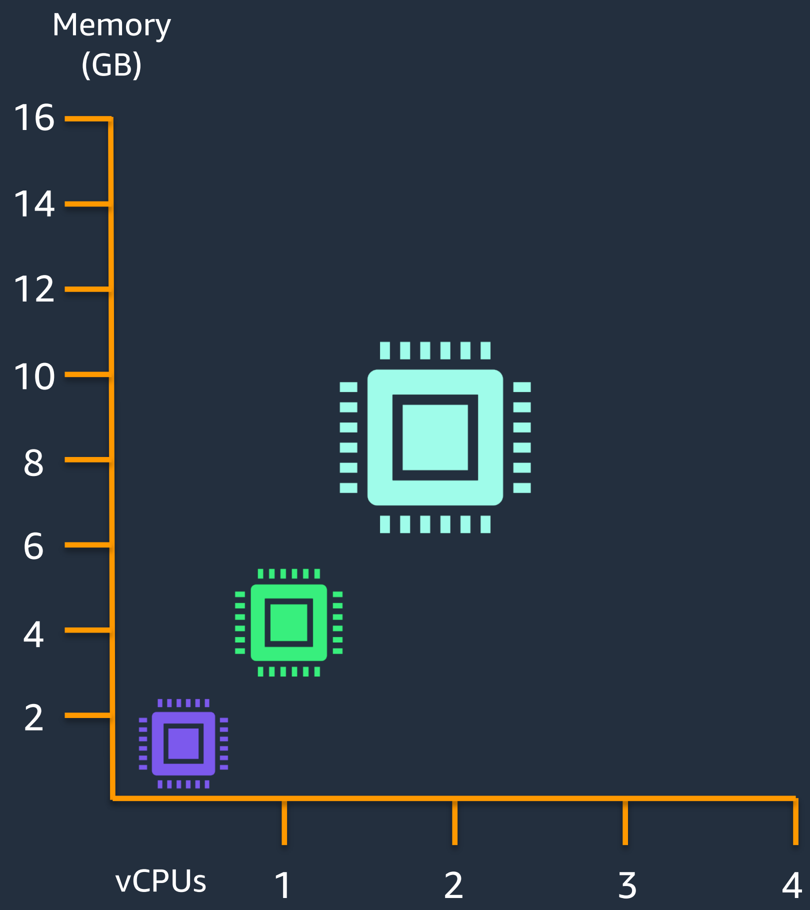
NODE PROVISIONING

- Kube Scheduler gets the first crack at scheduling pending pods. Tries to schedule on existing capacity
- Karpenter observes aggregate resource requests of unschedulable pods (set by kube scheduler) to make decisions on what instances to launch

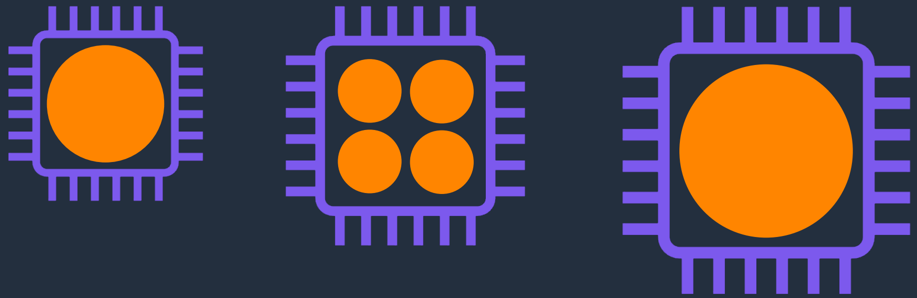


Karpenter bin-packing

NODE PROVISIONING



Online bin-packing while scaling up



Well-known labels

karpenter.sh/capacity-type=spot
kubernetes.io/arch=arm64
topology.Kubernetes.io/zone=us-west-2a
node.kuberenetes.io/instance-type=m5.large

Karpenter scale-up (continued)

NODE PROVISIONING

Instance types

Defaults to all instance types excluding metal and GPU

Diversify across sizes, families, generations CPUs

```
spec:
  requirements:
    - key: karpenter.sh/instance-type
      operator: NotIn
      values: [”m5.large"]
```

Purchase options

Defaults to on-demand

Combine Spot and on-demand

When included, Spot is prioritized

```
spec:
  requirements:
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["spot", "on-demand"]
```

Karpenter scale-up (continued)

NODE PROVISIONING

Architecture

Defaults to x86 instances (amd64)

Diversify across x86 and ARM

```
spec:
  requirements:
    - key: karpenter.sh/arch
      operator: In
      values: [”arm64”, “amd64”]
```

Availability zones

Defaults to all AZs

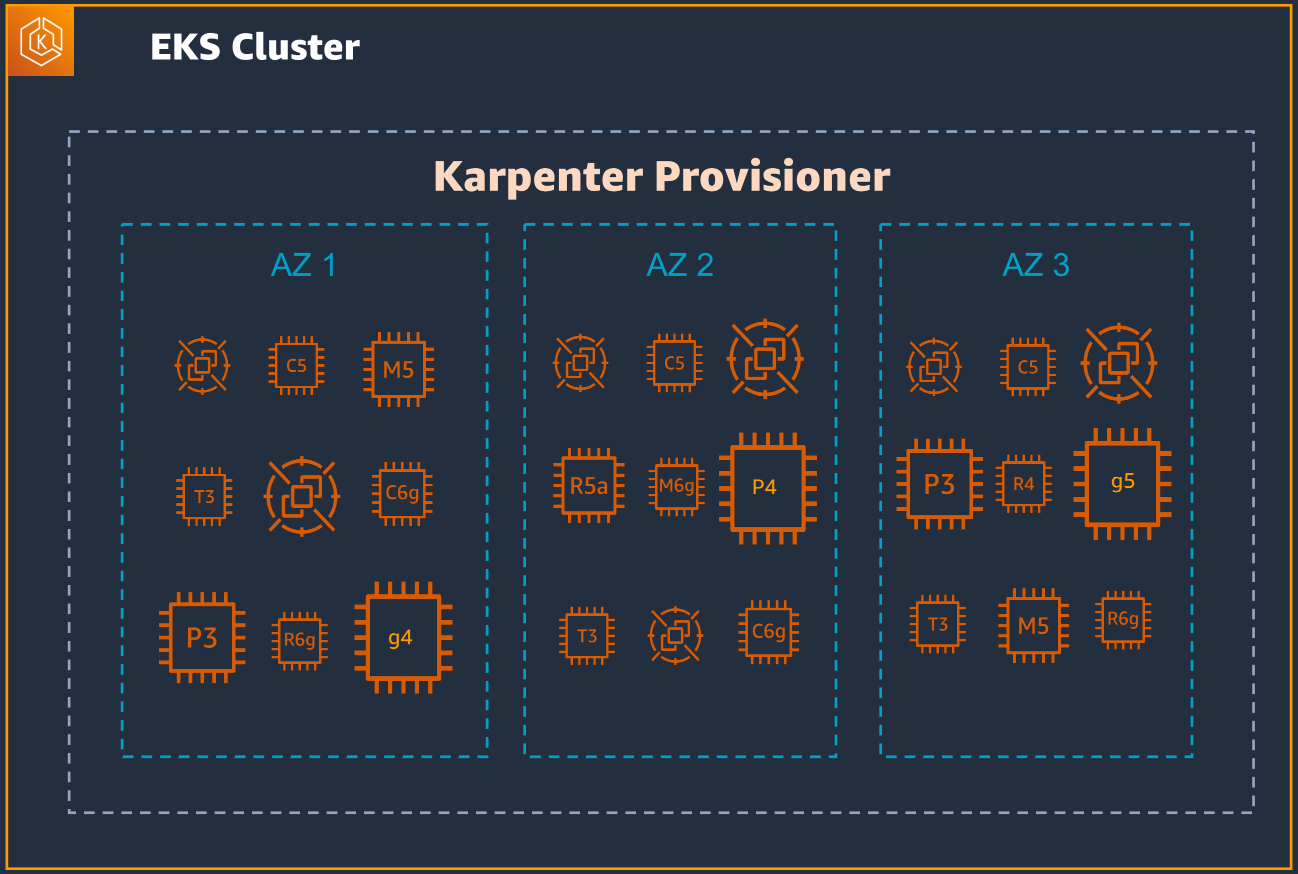
```
spec:
  requirements:
    - key: karpenter.sh/zone
      operator: In
      values: [”us-weast-2a"]
```

Provisioner CRD

- Custom resource to provision nodes with a set of attributes (taints, labels,
requirements, TTL)
- Single provisioner can manage compute
for multiple teams and workloads
- Can also have multiple provisioners for isolating compute for different needs

```yaml
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  consolidation:
    enabled: true
  requirements:
    # Include general purpose instance families
    - key: karpenter.k8s.aws/instance-family
      operator: In
      values: [c5, m5, r5]
    # Exclude small instance sizes
    - key: karpenter.k8s.aws/instance-size
      operator: NotIn
      values: [nano, micro, small, large]
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["on-demand", "spot"]
    - key: kubernetes.io/arch
      operator: In
      values: ["amd64", "arm64"]
  providerRef:
    name: default
```

Going large scale with Karpenter and Flexible Compute



Karpenter scale-in

NODE TERMINATION

Node TTL

Terminate empty nodes

Expire nodes to relaunch with new AMIs

```yaml
kind: Provisioner
metadata:
  name: default
spec:
  ttlSecondsAfterEmpty: 30
  ttlSecondsUntilExpired: 2592000
```

Consolidation

Attempts to reduce the overall cost of the nodes launched by that provisioner if workloads have changed

```yaml
kind: Provisioner
metadata:
  name: default
spec:
  consolidation:
    enabled: true
```

Karpenter consolidation

INTELLIGENT WORKLOAD'S RESOURCE CONSUMPTION REDUCTION

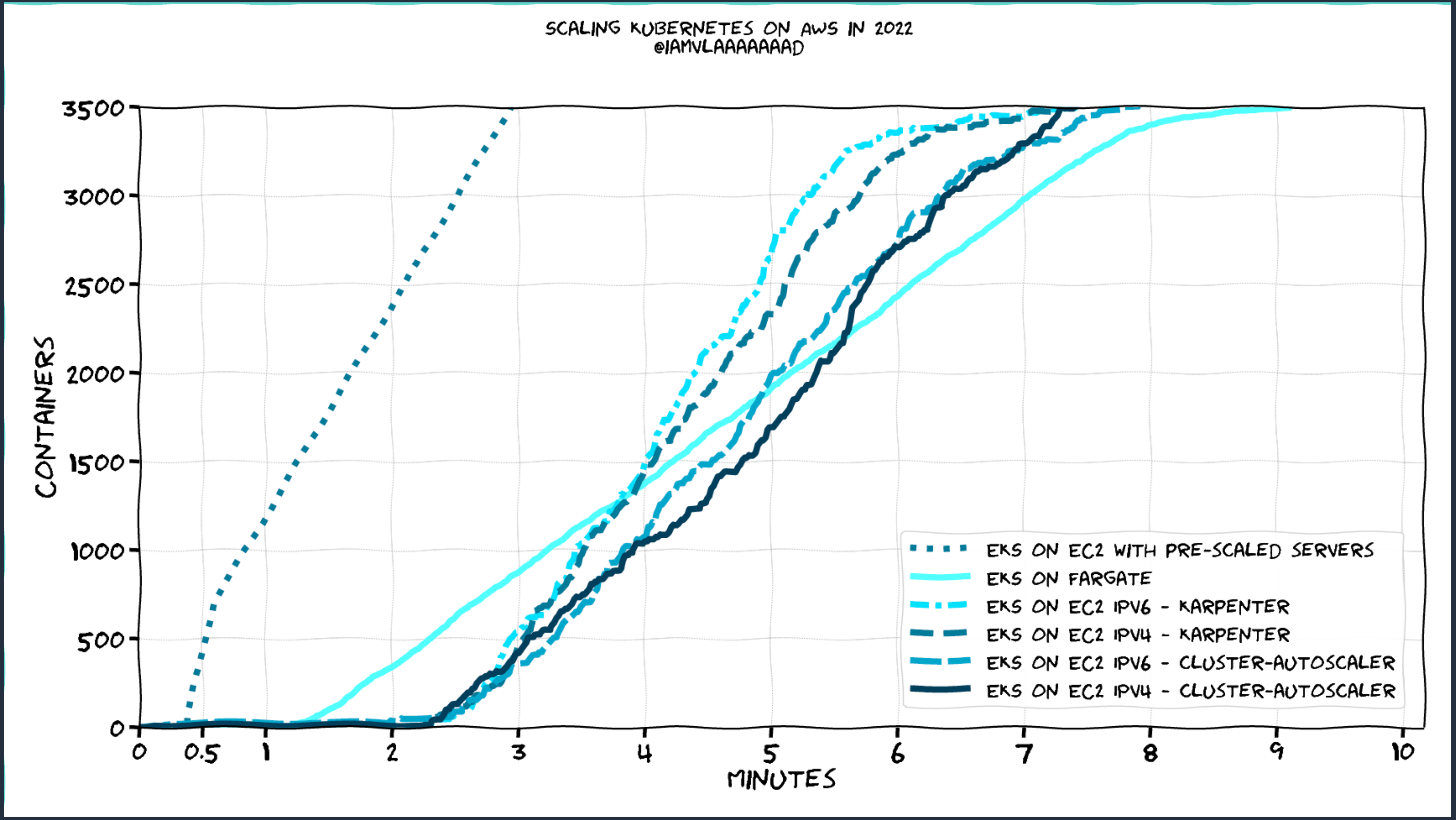
Deletes a node when…

It is empty or pods can run on free capacity of other nodes in the cluster

Replaces a node when…

Pods can run on a combination of free capacity of other nodes in the cluster + more efficient replacement node

Reduced Complexity, Increased Performance at scale



https://www.vladionescu.me/posts/scaling-containers-on-aws-in-2022/

Takeaways

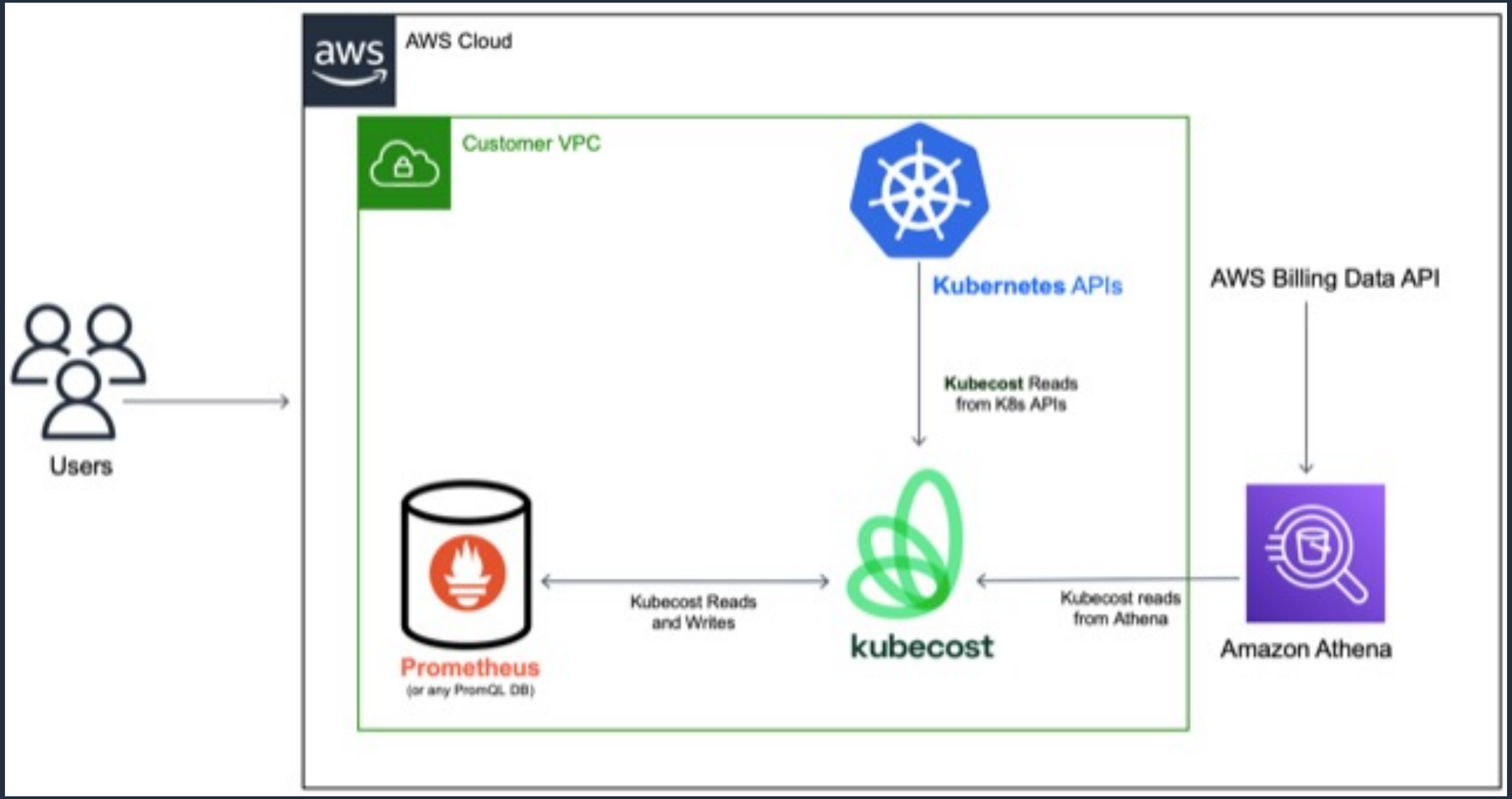
- Schedule pods to EC2 Spot Instances to optimize cost
- Use Provisioners to ensure you are scaling nodes using Spot best practices
- Use default Provisioner with diverse Instance Types and Availability Zones
- Use additional Provisioners for different compute constraints
- Control scheduling of your application Pods with Node Selector, topologySpreadConstraints, Taints, Tolerations and Provisioners

Kubecost

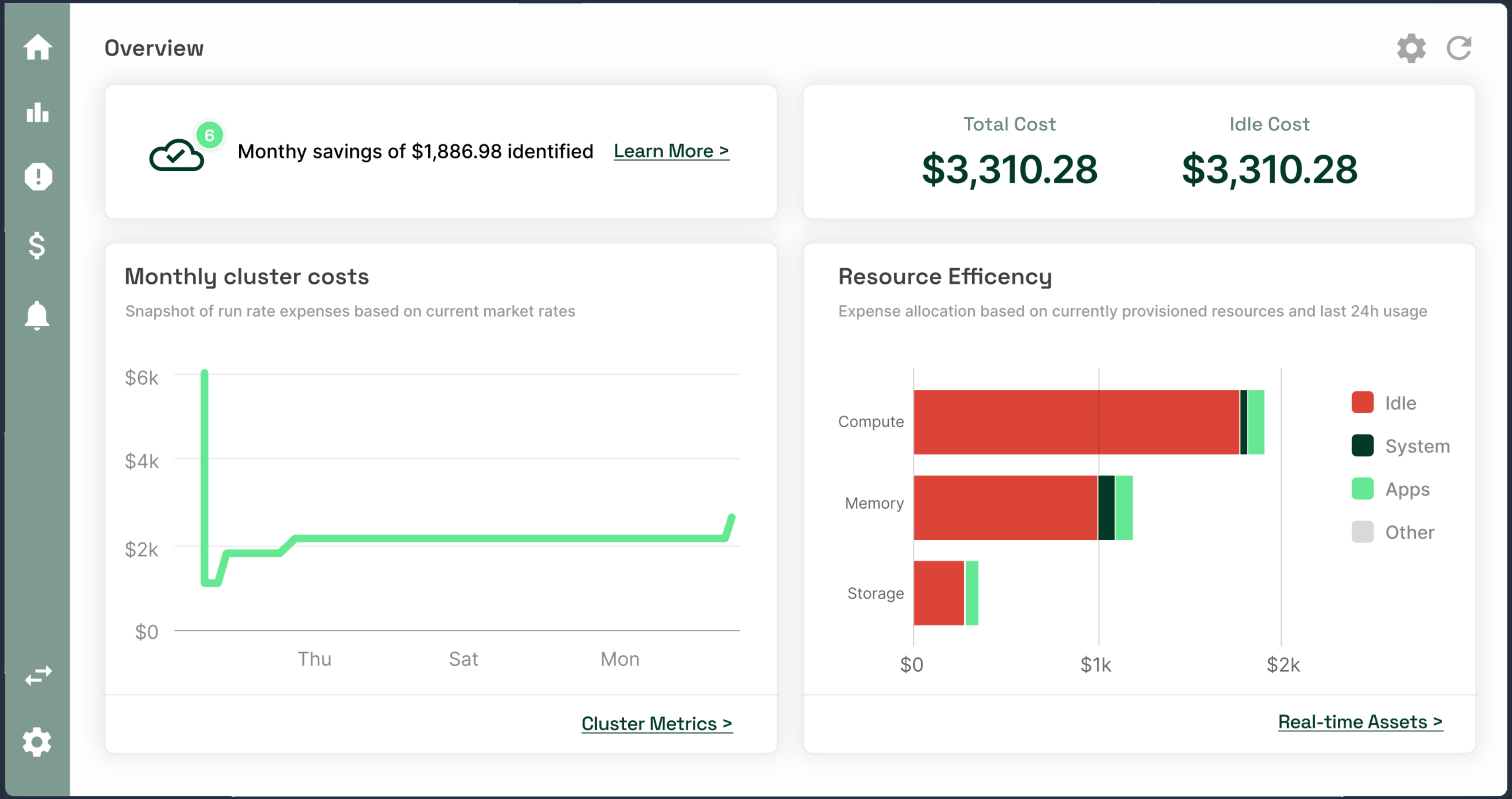
Kubecost: Real-time Cost Monitoring

- Open source tools like **Kubecost** enable real-time cost visibility for Kubernetes
- Evaluates cost and usage at deep granularity:
  - by Kubernetes service, deployment, namespace, label, statefulset, daemonset, pod, and container
- Cost and usage can be attributed to org concepts such as team or application
- Uses Prometheus metrics to determine usage by applications
- Makes recommendations for where to optimize resources

Kubecost: Real-time Cost Monitoring



Kubecost: Real-time Cost Monitoring



Overview

6 Monthy savings of $1,886.98 identified Learn More >

Total Cost $3,310.28

Idle Cost $3,310.28

Monthly cluster costs

Snapshot of run rate expenses based on current market rates

Cluster Metrics >

Resource Efficency

Expense allocation based on currently provisioned resources and last 24h usage

Real-time Assets >

Thank you!

Borja Pérez
bpguasch@amazon.es

Arpit Sapra
sapraa@amazon.com

SCAN ME

Additional resources

- Karpenter : https://karpenter.sh
- Karpenter Best Practices: https://aws.github.io/aws-eks-best-practices/karpenter/
- Karpenter workshop: https://ec2spotworkshops.com/karpenter.html
- Launch Blog: https://aws.amazon.com/blogs/aws/introducing-karpenter-an-open-source-high-performance-kubernetes-cluster-autoscaler/
- Blog post: https://aws.amazon.com/blogs/containers/using-amazon-ec2-spot-instances-with-karpenter/