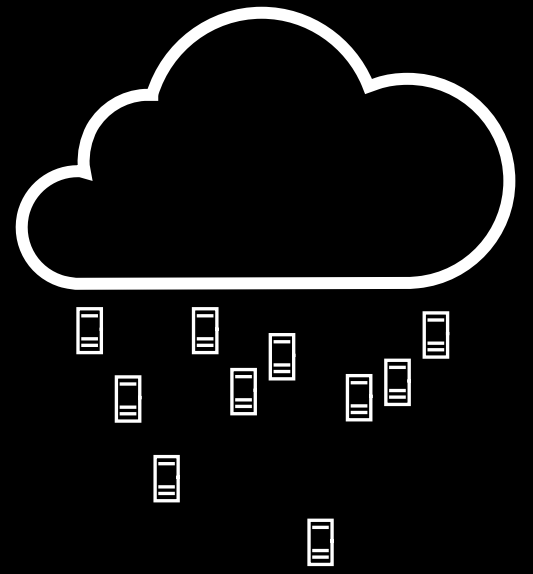
Protean: VM Allocation Service at Scale

Ori Hadary
Ishai Menache
Esaias E Greeff
Star Dorminey
Yang Chen
Thomas Moscibroda
Luke Marshall
Abhisek Pan
David Dion
Shailesh Joshi
Mark Russinovich



Microsoft
Azure / Research

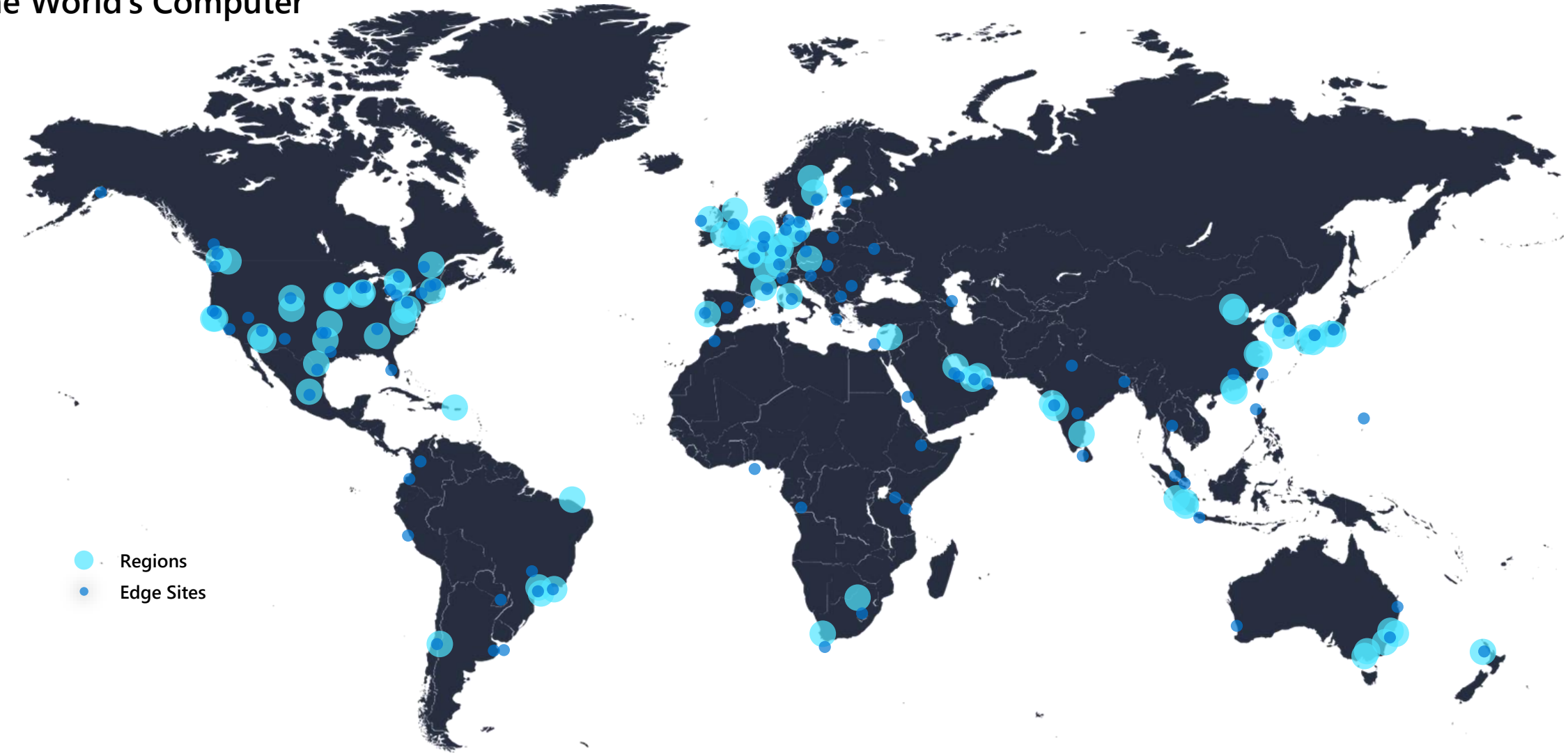
Protean

- Virtual machine allocation service for Microsoft Azure
- Allocates millions of VMs to millions of servers every day
- Runs at Zone Scale (100k machines)
- Critical for Azure Operations

Azure – Scale, Diversity, Uncertainty

Microsoft Azure

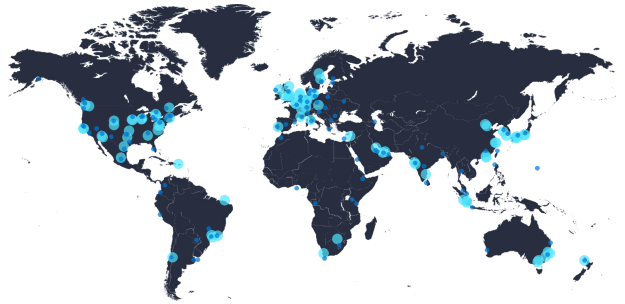
The World's Computer

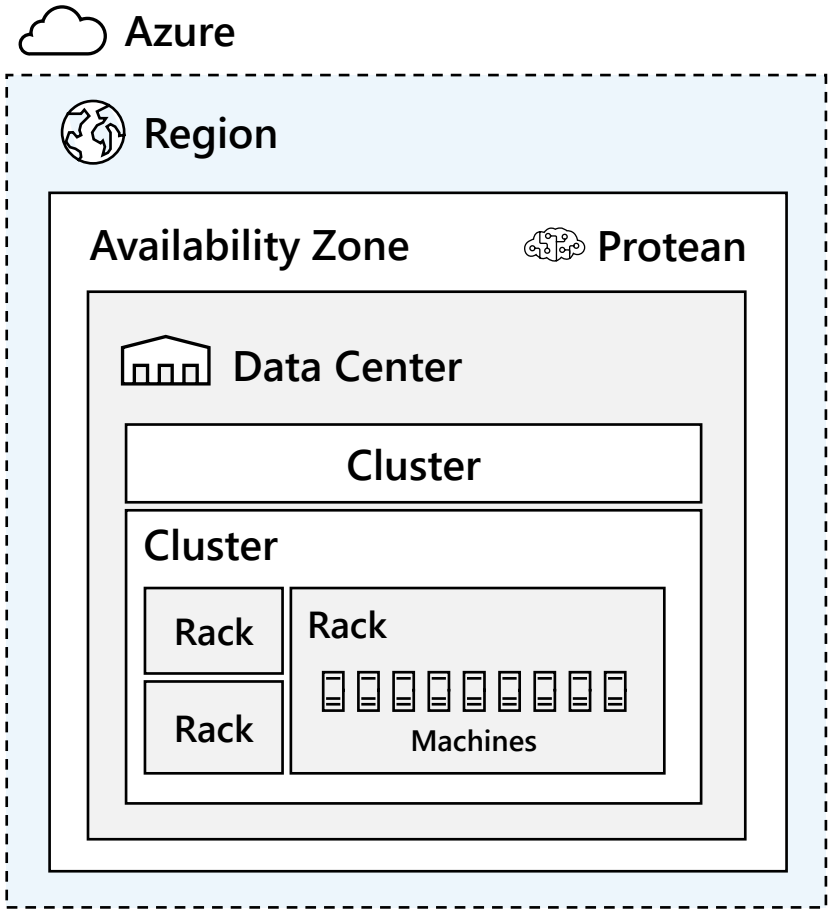


61 Regions **170+** Edge sites **2M+** Machines **5M+** New VMs / day

Topology

- Region
  - Multiple datacenters for redundancy and availability
  - 1 – 3 availability zones
- Availability Zone
  - Unique physical location
  - Independent power, cooling and networking
  - 100k+ machines
  - One or more DCs housing 100+ clusters
- Cluster
  - Homogenous set of 1000 – 3000 machines





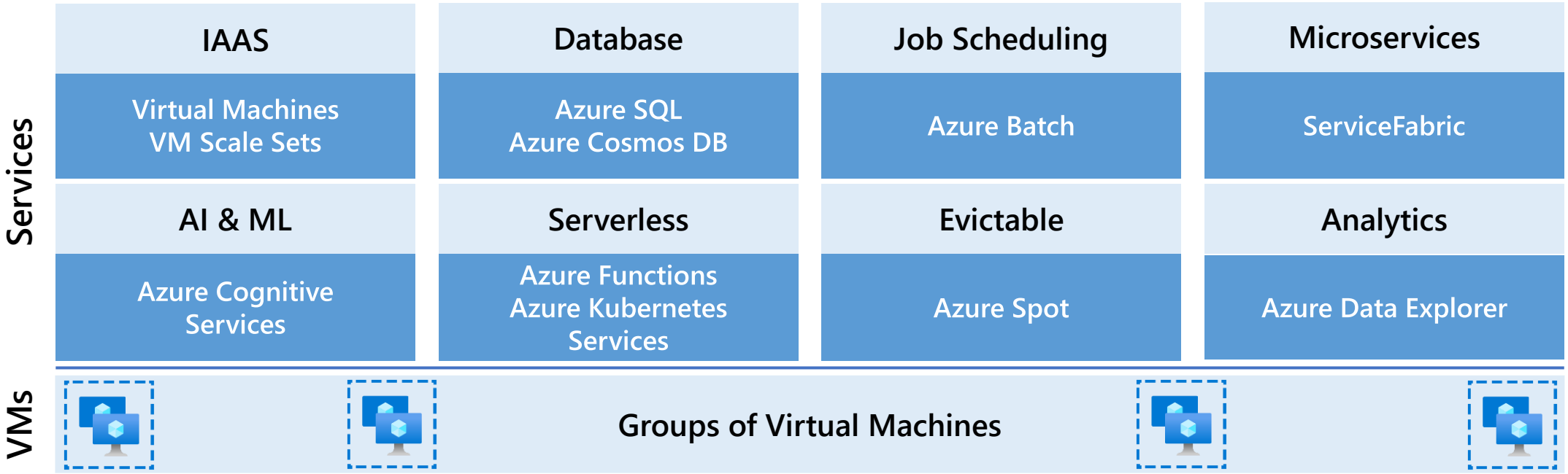
Hardware diversity

- Hardware generations
- Machine configurations
  - CPU / Memory optimized
  - Accelerators with high-performance interconnects
- Disaggregated architectures
- Containerized Datacenters

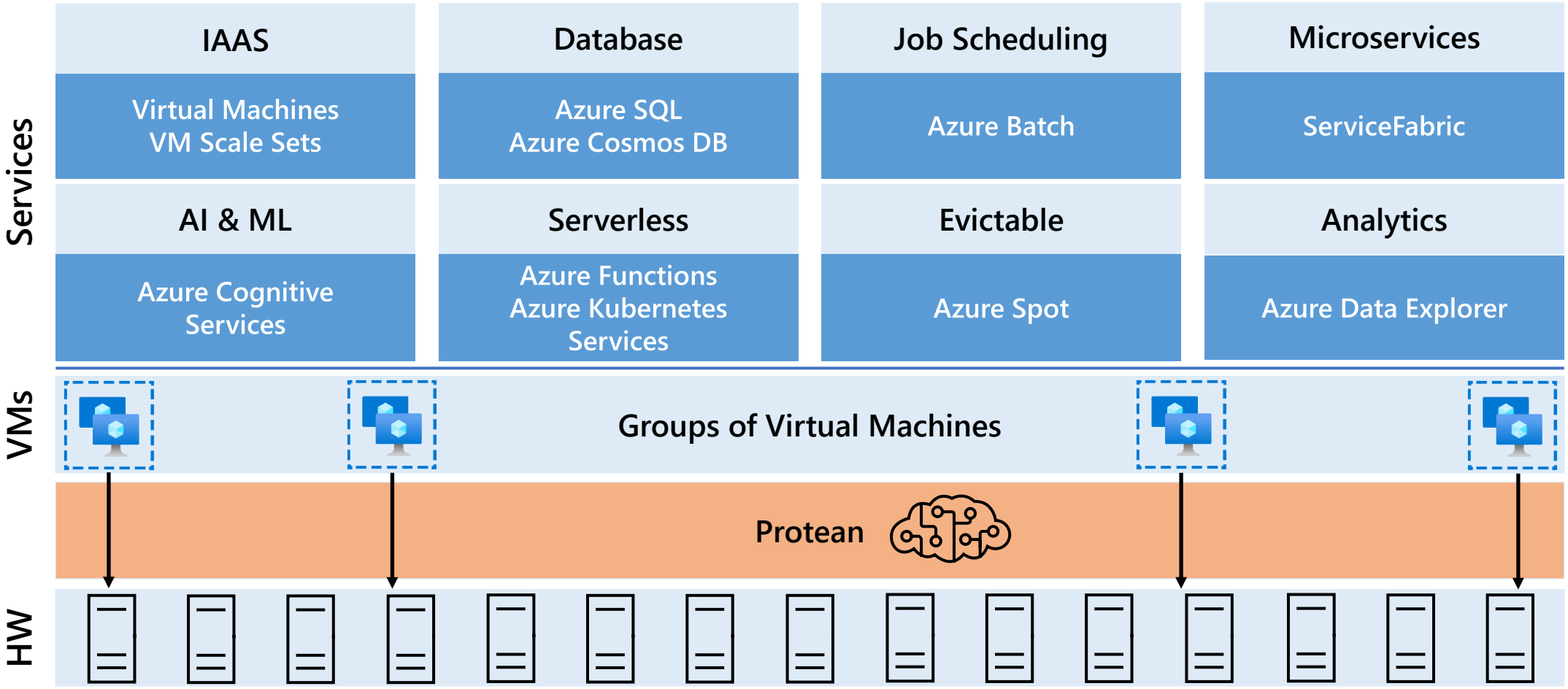
Azure Services

IAAS	Database	Job Scheduling	Microservices
Virtual Machines VM Scale Sets	Azure SQL Azure Cosmos DB	Azure Batch	ServiceFabric
AI & ML	**Serverless**	**Evictable**	**Analytics**
Azure Cognitive Services	Azure Functions Azure Kubernetes Services	Azure Spot	Azure Data Explorer

Platform Workload

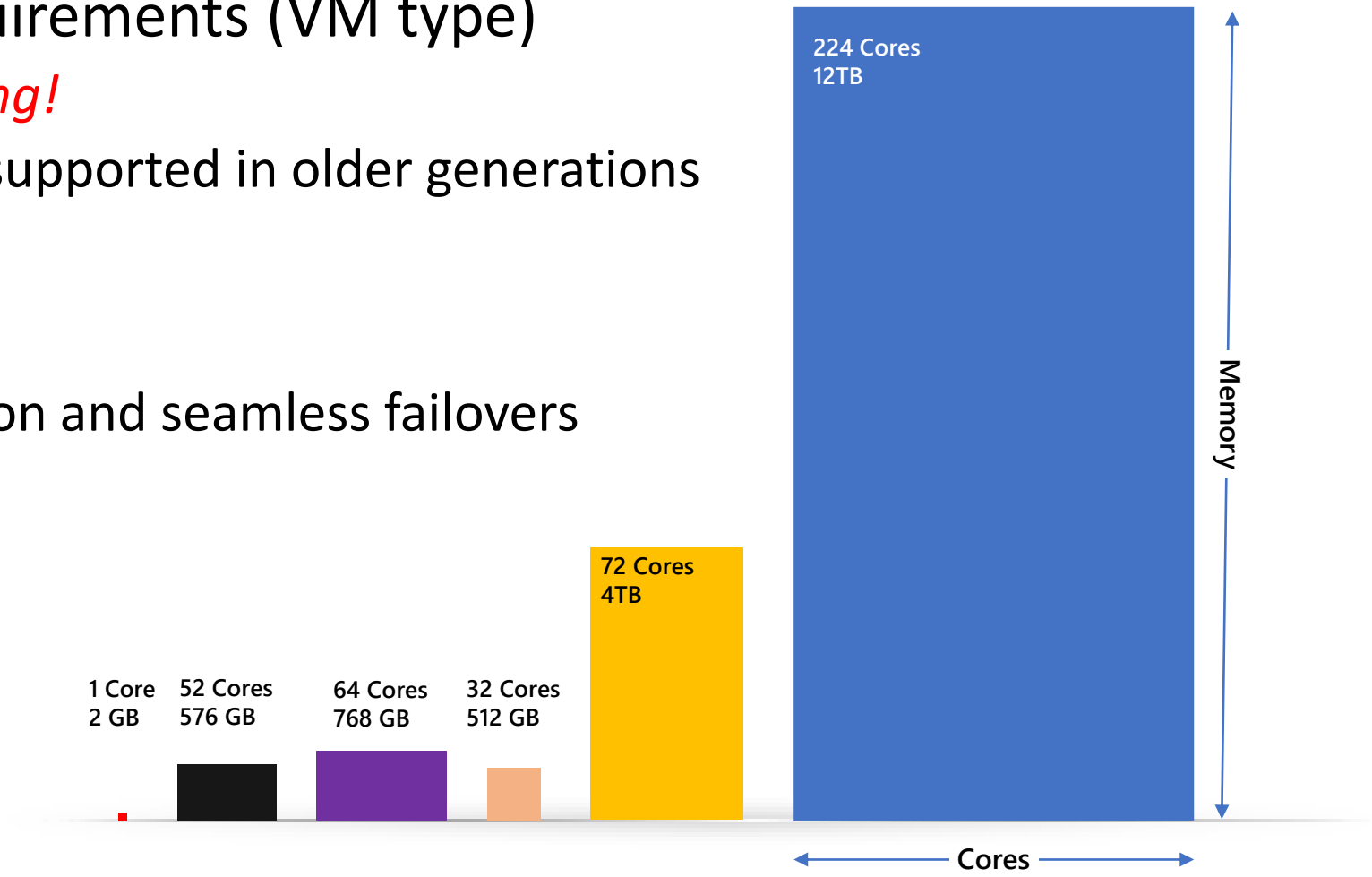


Machine-VM Assignment



Virtual Machines

- Discrete resource requirements (VM type)
  - *700+ types and growing!*
  - Newer VM Types not supported in older generations
- Priority
  - All-or-nothing allocation and seamless failovers
  - Preemptions



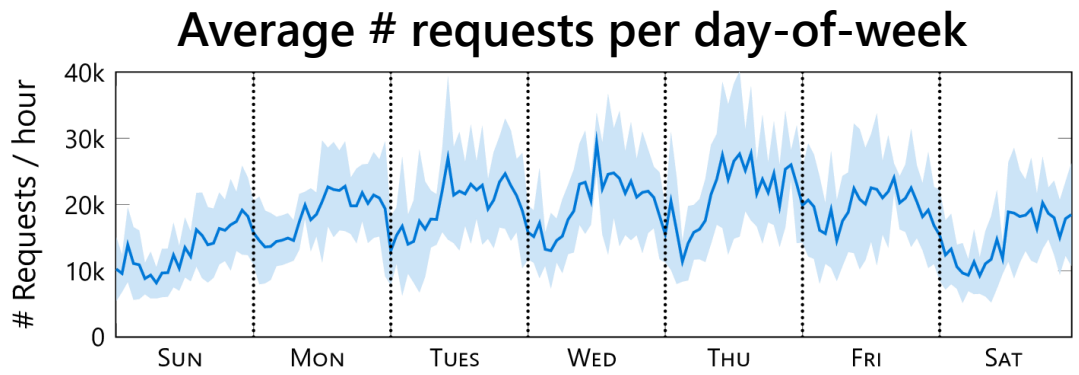
Distribution of VM Types

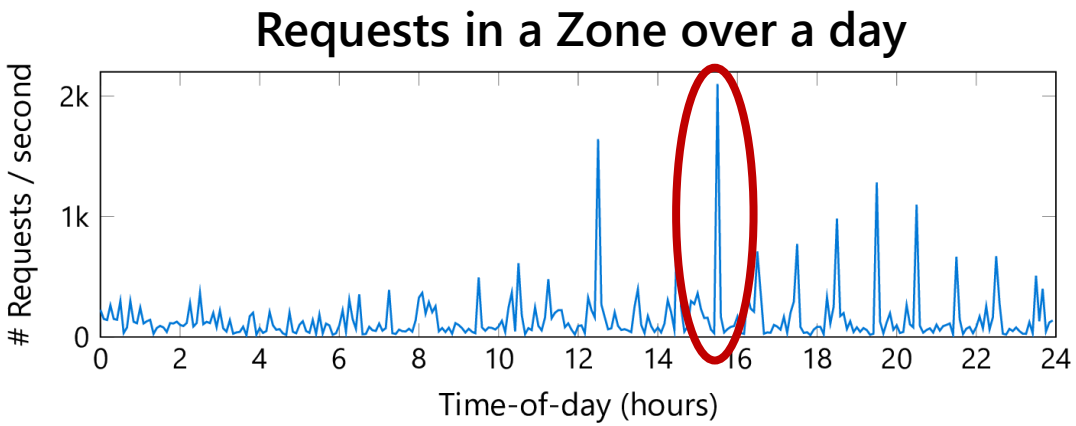
VM Type	Zone1 (%)	Zone2 (%)
A	4.6	0.1
B	3.5	3.6
C	6.5	12.9
D	0.7	8.4
E	1.9	3.7
F	3.2	4.4
G	0.6	3.1
H	0.8	2.2
I	2.4	7.4
J	23.7	31.6
K	21.3	2.1
L	3.5	0.4
M	0.0	2.2

VM Cores	Zone1 (%)	Zone2 (%)
1	17.1	27.0
2	37.4	52.4
4	32.0	10.5
8	8.9	4.5
⩾ 10	4.3	2.6
⩾ 20	0.3	3.1

- Large number of VM types with some popular types
- Most VMs use 1 – 4 cores

Variation in Demand





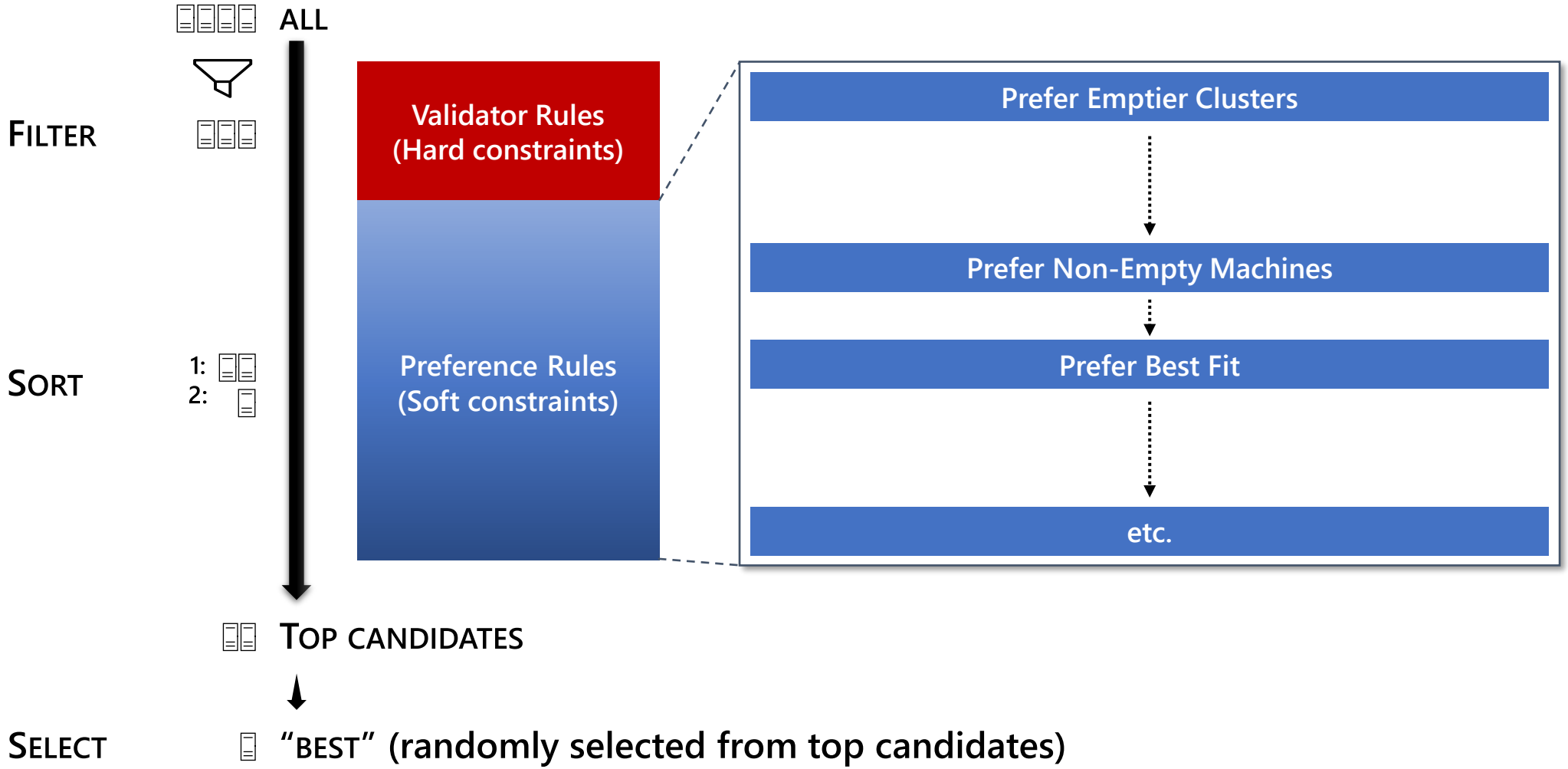
Diurnal patterns with intermittent spikes

Objectives

- Excellent customer experience
  - Prefer machines that can start VM faster
  - Minimize disruptions due to scheduled hardware maintenance
  - Low eviction rates
- Low fragmentation
  - *1% reduction can lead to $100M savings per year!*
  - Improves acceptance rates for large VMs
- Extensibility and Adaptability
  - Easy and safe to add new behavior, or tune existing behavior
- Interpretability
  - *Why did my VM creation fail?*

Handling Complexity

Rule-Based Allocation Logic



Validator Rules

```
interface IValidatorRule
{
   // Is machine (or cluster) x a valid candidate for placing VM v?
   bool IsValid(Node x, VM v);
}
```

Example:

`AreNodeResourcesValid`

- Does machine x have enough capacity to accommodate VM v?

Preference Rules

```
interface IPreferenceRule
{
    // Which node (or cluster) between x and y is preferable for VM v?
    bool Compare(Node x, Node y, VM v);
}
```

Example:

`PreferNonEmptyMachines`

- Improve packing quality by preferring non-empty machines

Handling Multiple Preference Rules

- Strict prioritization requires “smoothing” of rules
  - Extreme discrimination by a rule makes weaker rules inconsequential
- Continuous score values are quantized into a few buckets.

Example:

`PreferBestFit`

- Multi-dimensional best-fit
- Assigns a weighted score S ∈ [0,1] to a machine
- Each resource is weighed by its global scarcity
- The score is quantized into N buckets

Buckets	PD (%)	Post-BestFit (%)
1	83.5	100.0
2	84.3	90.6
3	86.3	76.1
4	87.3	59.8
5	87.8	49.6
∞	89.1	8.3

Latency & Throughput at Scale

Throughput at Zone Scale

- Scale throughput as inventory and demand grows
  - Multiple allocation agents operating concurrently
  - Optimistic concurrency
  - Fine-grained conflict detection
  - Adaptive conflict reduction strategies during peak demand

Latency

- Keep allocation times bounded as inventory grows
  - Low-overhead step to pre-select eligible and high-quality clusters
    - Limits the number of machines used in machine selection step
  - **Multi-layered caching to expedite machine selection**

Motivations for Caching

- Requests exhibit temporal locality
  - Requests are similar, even considering all traits
- Inventory changes slowly
  - Primarily due to VM creates and deletes

Baseline Implementation

```
class AreNodeResourcesValid : IValidatorRule
{
   bool IsValid(Node x, VM v)
   {
      return AreNodeResourcesValid(x,v.Type);
   }
}
```

Storing Rule State

```
class AreNodeResourcesValid : IValidatorRule
{
   RuleState:
```

VM Type	Node	IsValid
A	N_1	true
B	N_1	true

```
   bool IsValid(Node x, VM v)
   {
      return RuleState[x][v.Type];
   }
}
```

Store state in long-lived rule object

Cache and reuse

Updating Rule State

```
class AreNodeResourcesValid : IValidatorRule
{
   RuleState:
```

VM Type	Node	IsValid
A	N_1	true
B	N_1	true

```
   bool IsValid(Node x, VM v)
   {
      return RuleState[x][v.Type];
   }

   void Update(Node[] modified)
   {
      foreach (Node x in modified)
         foreach (VMType t in RuleState.VMTypes)
            RuleState[x][t] = AreNodeResourcesValid(x,t);
   }
}
```

Update is called immediately before the rule object is used, with the latest state for modified machines

Further Improvements?

Splitting Rule State

```
class AreNodeResourcesValid : IValidatorRule
{
   RuleState:

   VMType T;

   bool IsValid(Node x)
   {
      return RuleState[x];
   }

   void Update(Node[] modified)
   {
      foreach (Node x in modified)
         RuleState[x] = AreNodeResourcesValid(x,T);
   }
}
```

Node	IsValid
N_1	true

Separate objects for each VM type

- r_1 = new AreNodeResourcesValid(A)
- r_2 = new AreNodeResourcesValid(B)

Reduces computation further

Object r_1 processes all requests of type A

Caching Complete Evaluation State

- Encapsulate and store entire evaluation state for a request
- Characterized by a vector of trait values
- Reused for all requests with the same combination of trait values

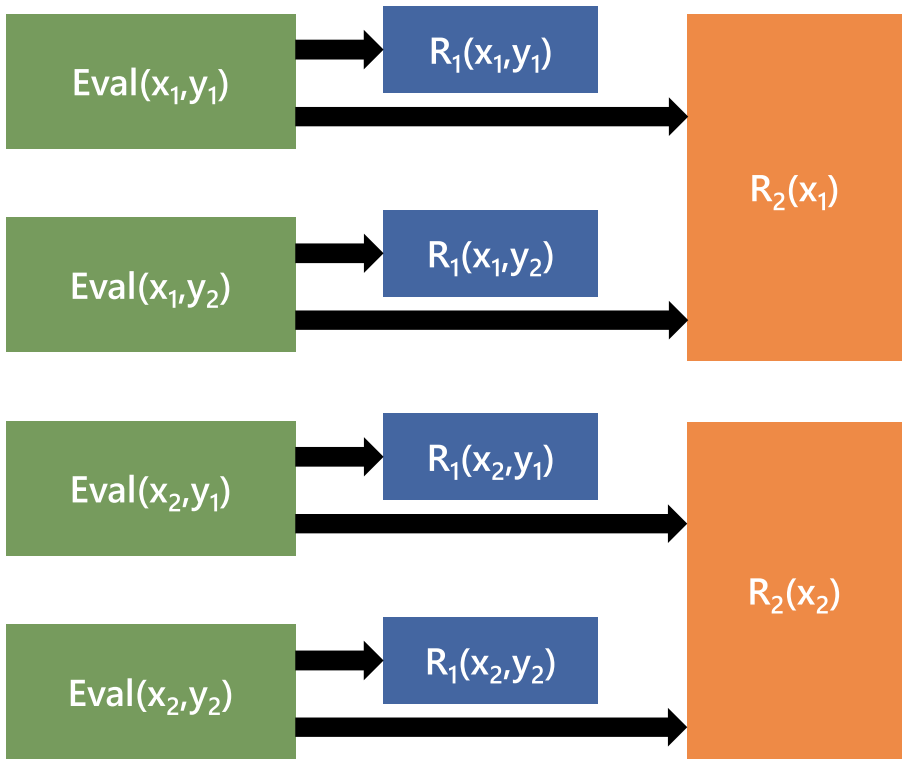
```
RuleEvaluationObject
{
   Dictionary<ITraitType,ITraitValue> identifier;

   OrderedList<Node> sortedNodes;

   OrderedList<IRule> ruleObjects;
}
```

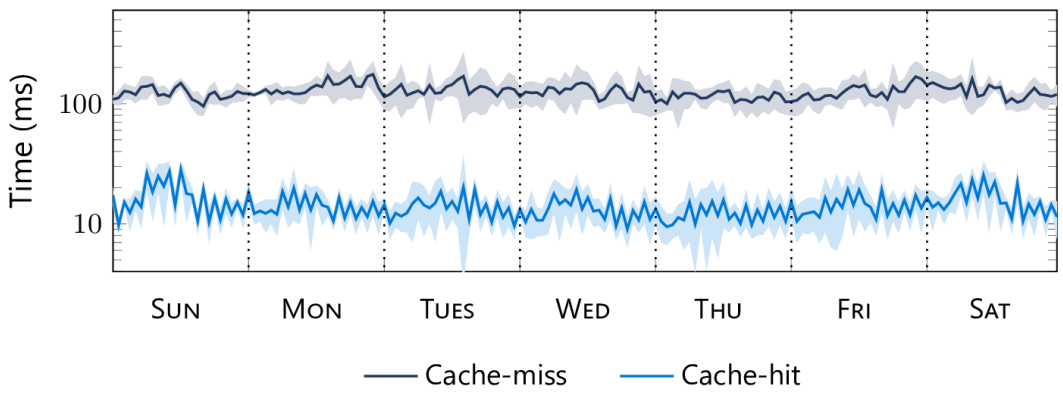
Request Traits: $X \in \{x_1, x_2\}$; $Y \in \{y_1, y_2\}$

Rules: R_1 depends on X and Y,
R_2 depends on X.

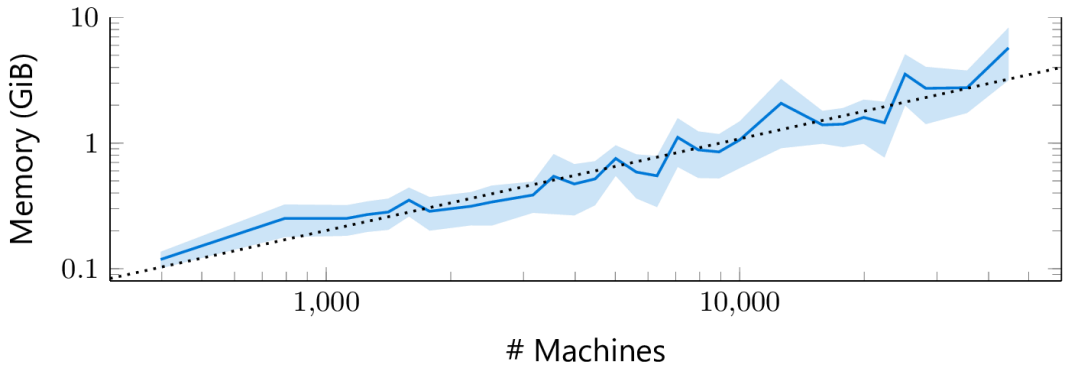


Cache Evaluation

Latency Improvement due to Cache Hits



Memory Footprint vs Inventory Size



Thank you!

Azure VM packing trace
https://github.com/Azure/AzurePublicDataset

Contact: protean_azure@microsoft.com

Team Protean:

Jason Chu
Eric Hao
John Miller
Kanishk Thareja
Brian Yan
Chris Cowdery
Ryan Hidalgo
Mukund Nigam
Karel Trueba
Dustin Dobransky
Valentina Li
Jason Seo
Yiran Wei