Tributary: spot-dancing for elastic services with latency SLOs

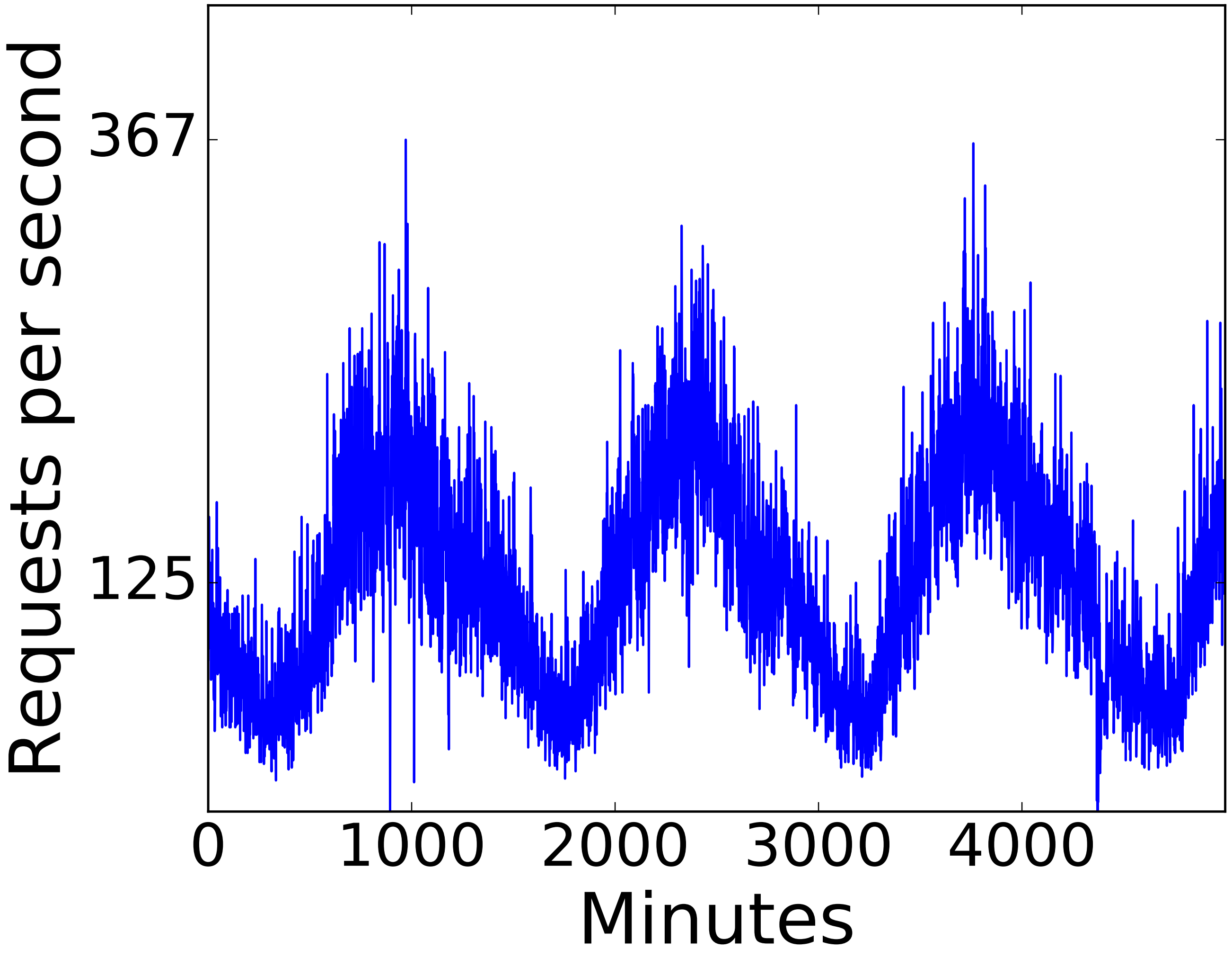
Aaron Harlap, Andrew Chung,
Alexey Tumanov*, Greg Ganger, Phil Gibbons

Carnegie Mellon University

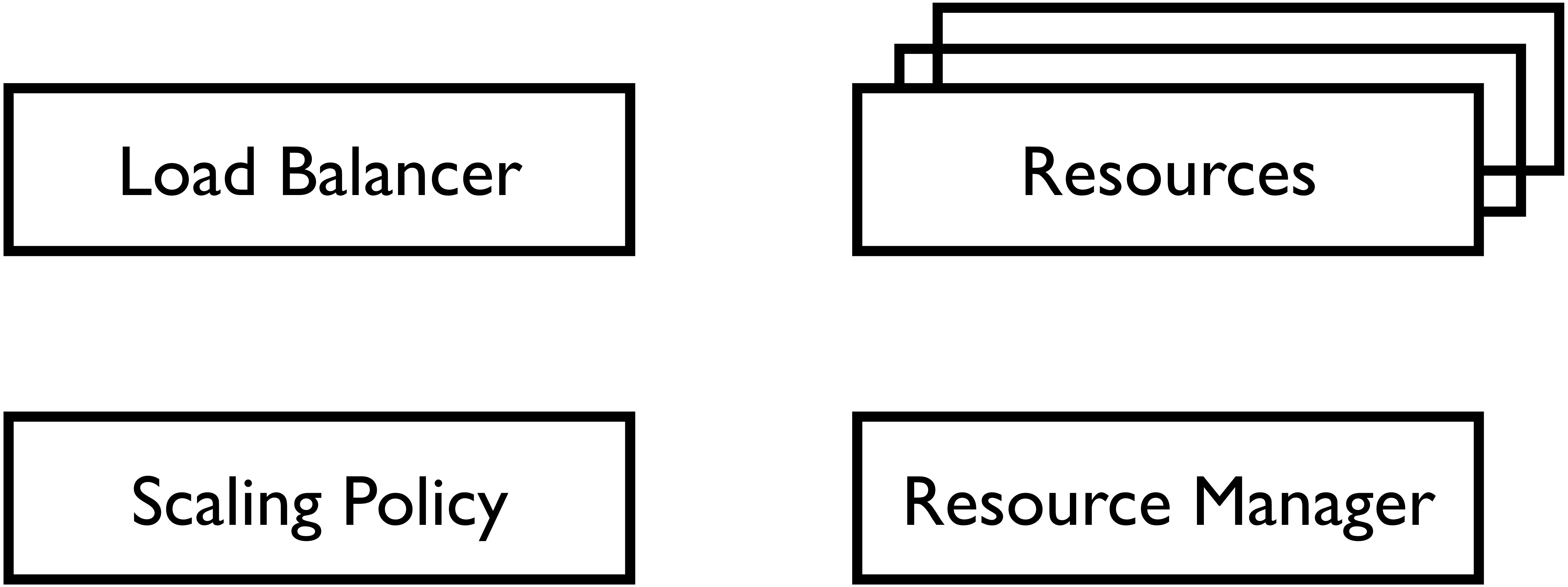
* UC Berkeley

Services with SLOs

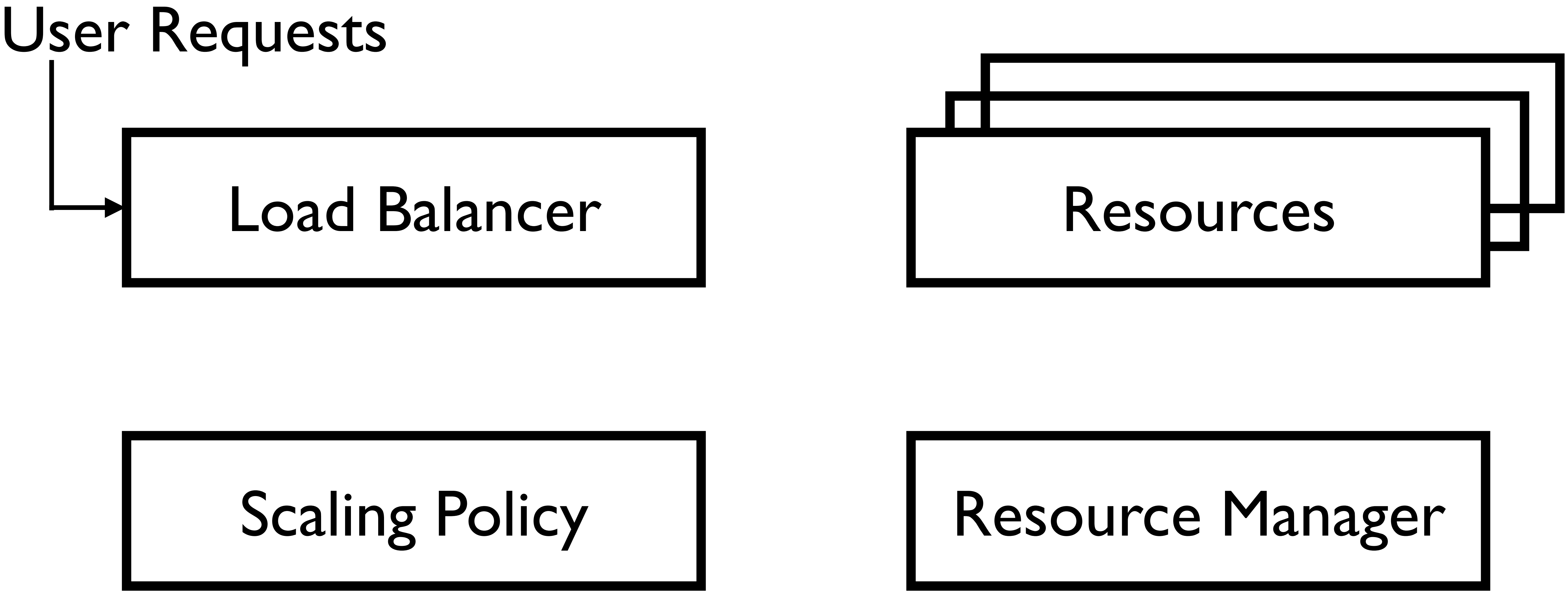
- Time varying client workloads
  - handled with elastically sized resources
- How are they sized?
  - decide how many resources are needed
  - add/release resources



Elastic Service Architecture



Elastic Service Architecture



User Requests

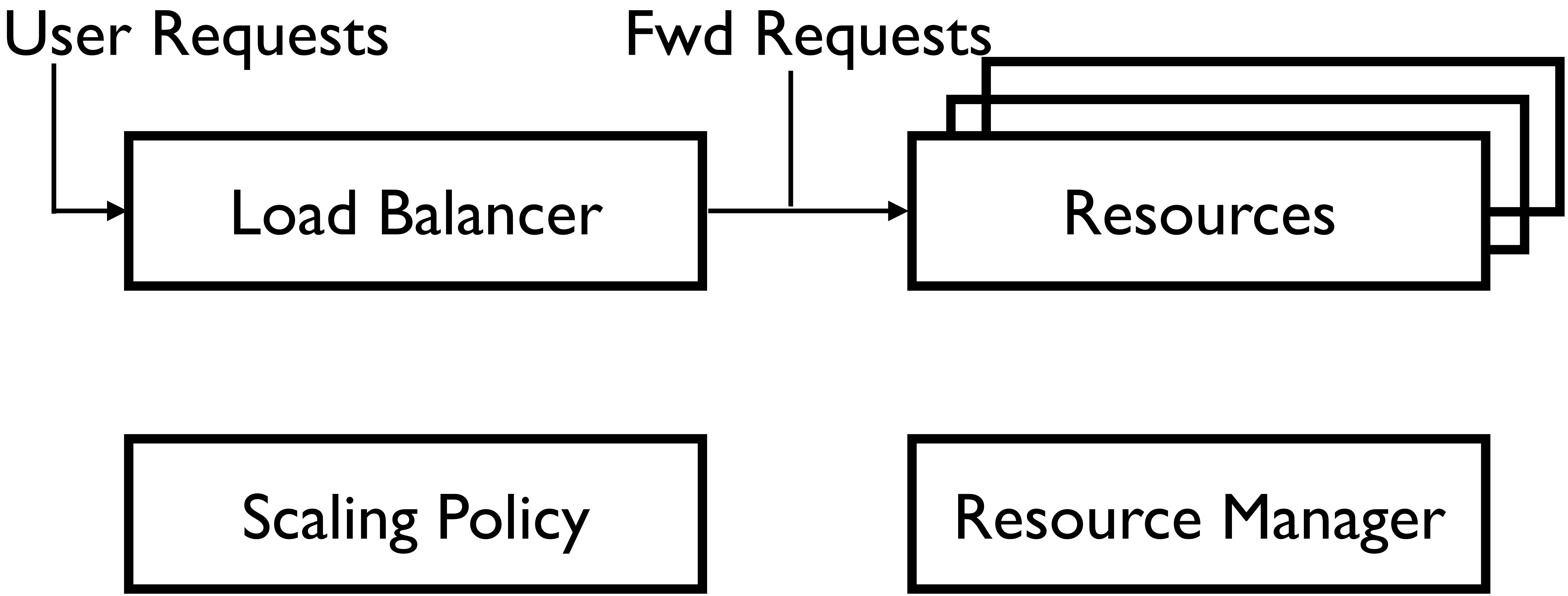
Load Balancer

Resources

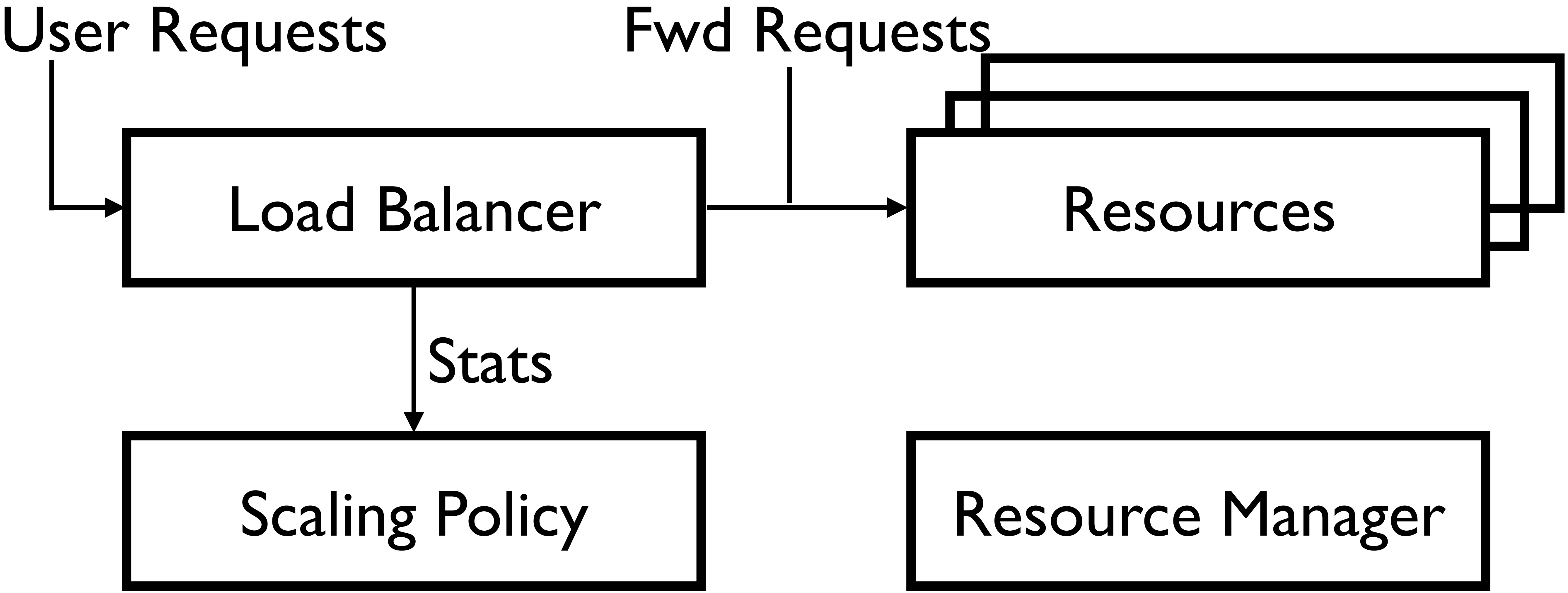
Scaling Policy

Resource Manager

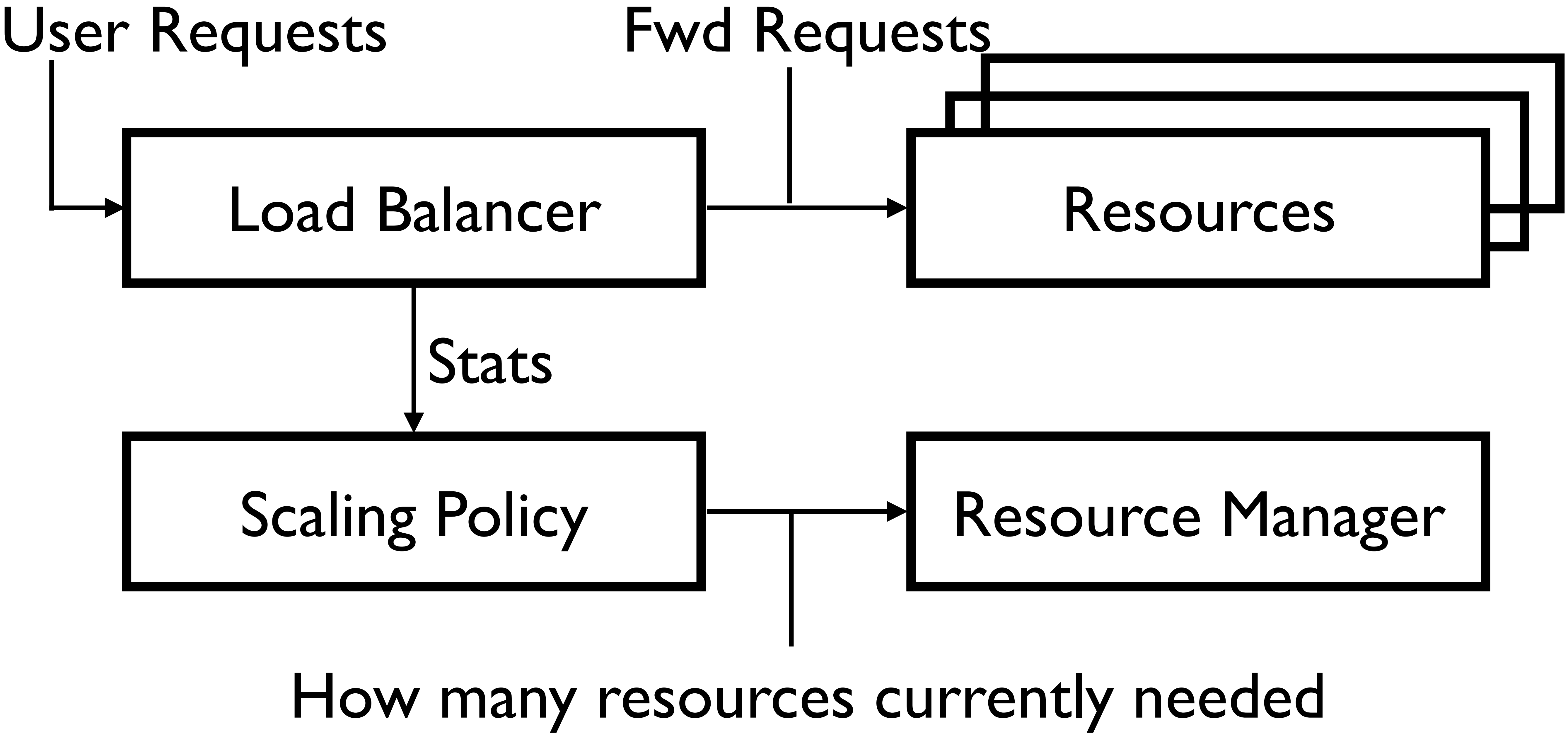
Elastic Service Architecture



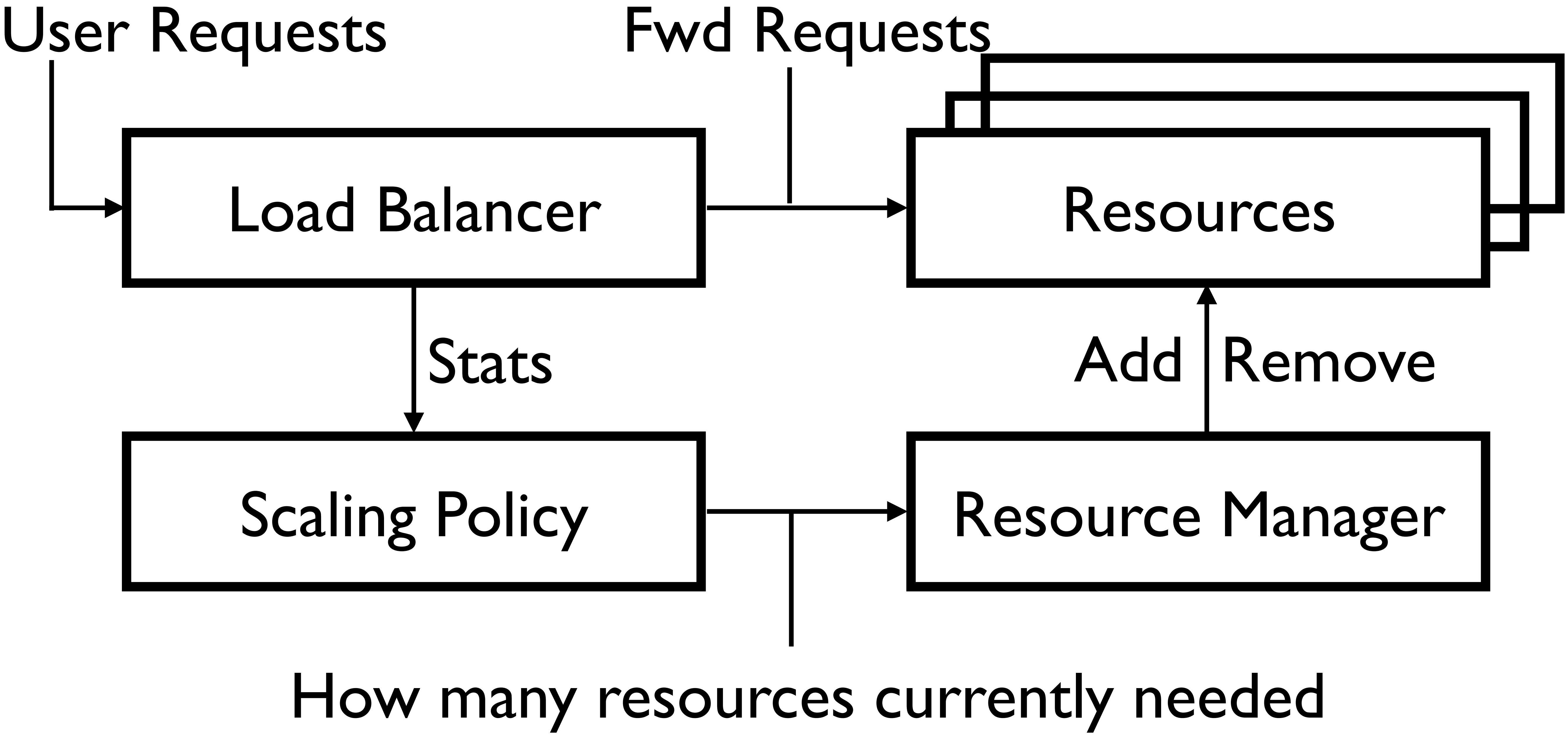
Elastic Service Architecture



Elastic Service Architecture



Elastic Service Architecture

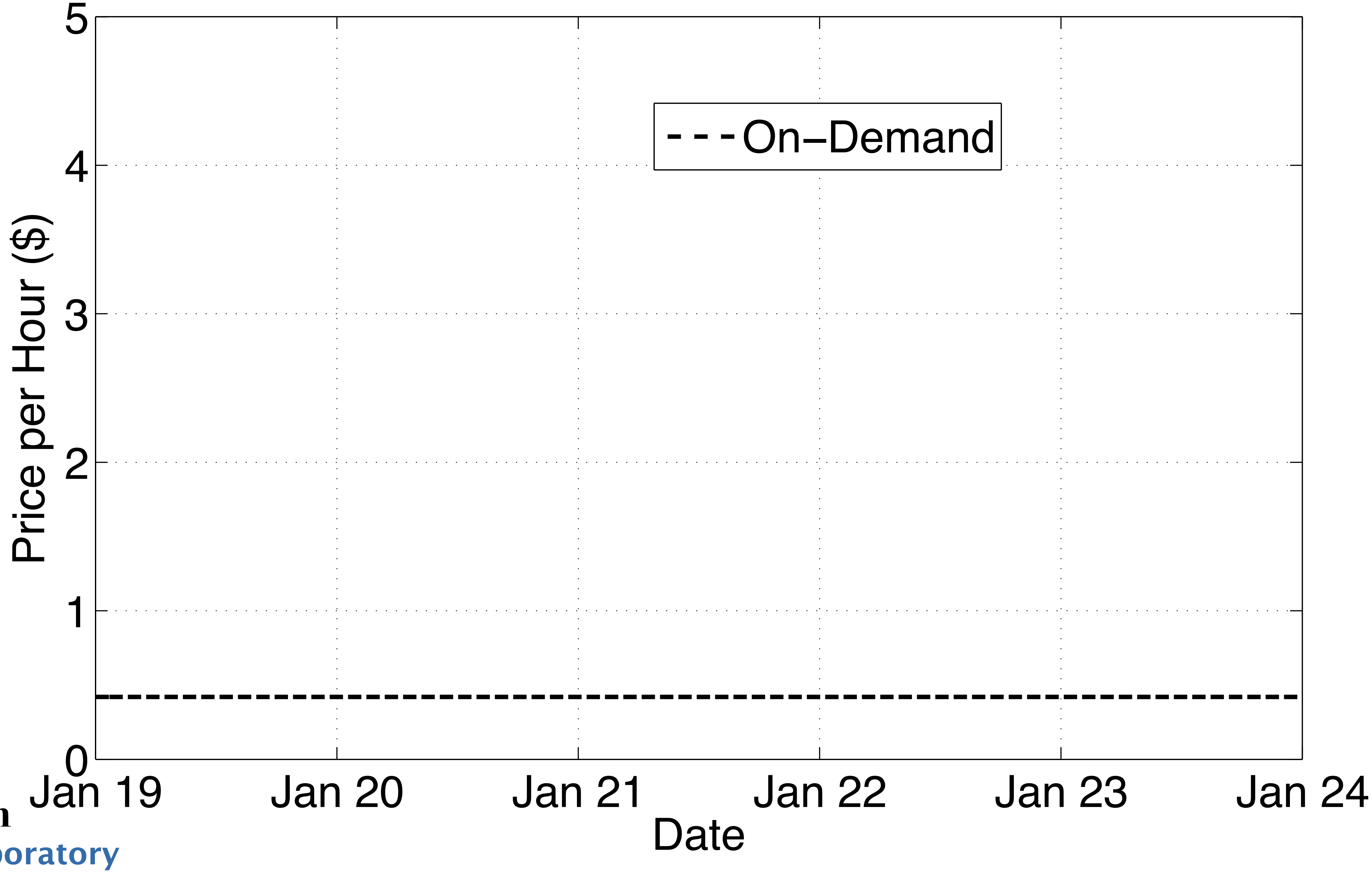


Why Tributary?

- CSPs offer cheaper resources that come with potential of being taken away
  - GCE preemptible instances
  - AWS EC2 spot instances
- Preemptions are bad for services w/ SLOs

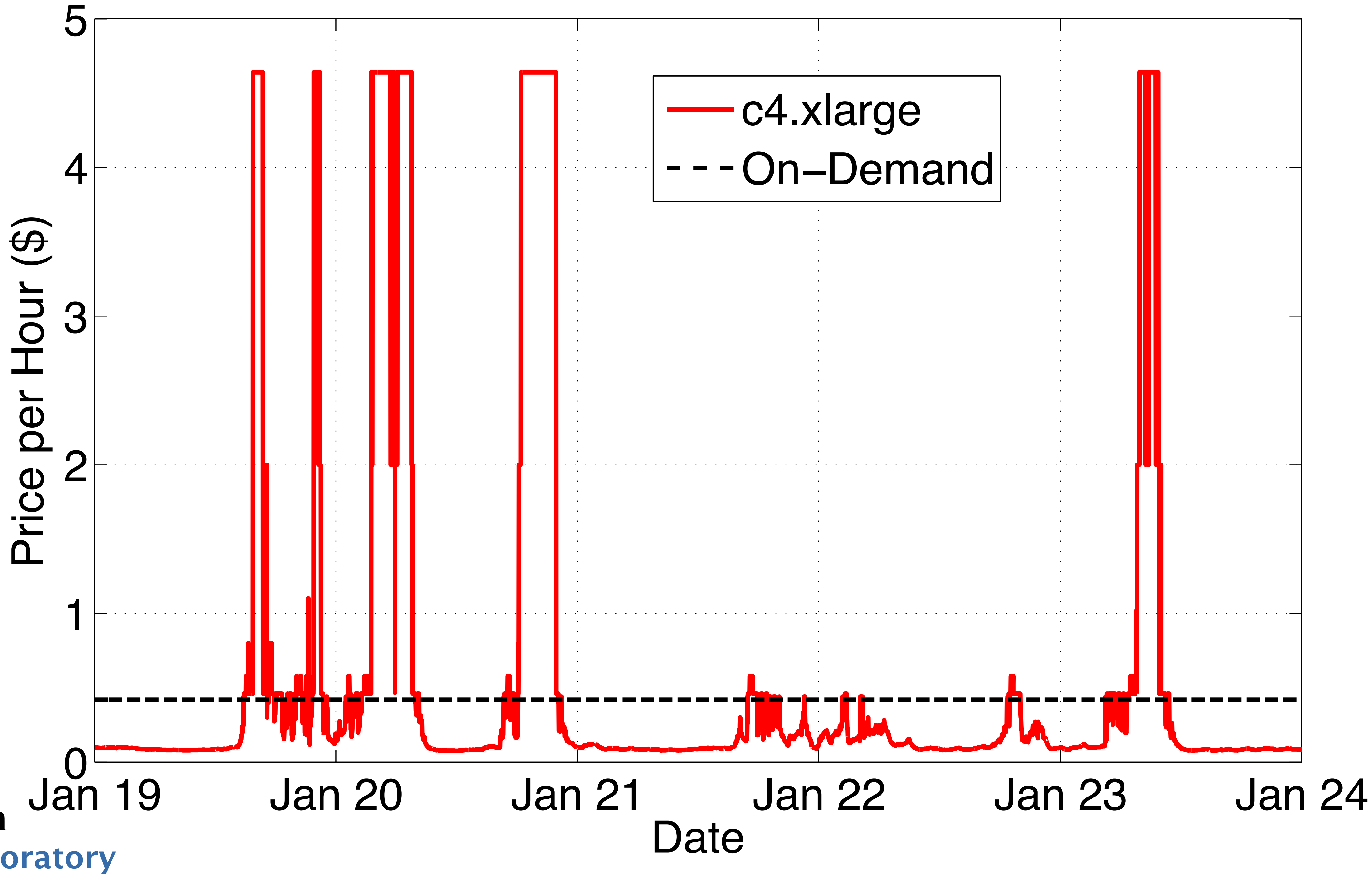
Transient resources much cheaper

- Often 75-85% cheaper to use Spot Instances



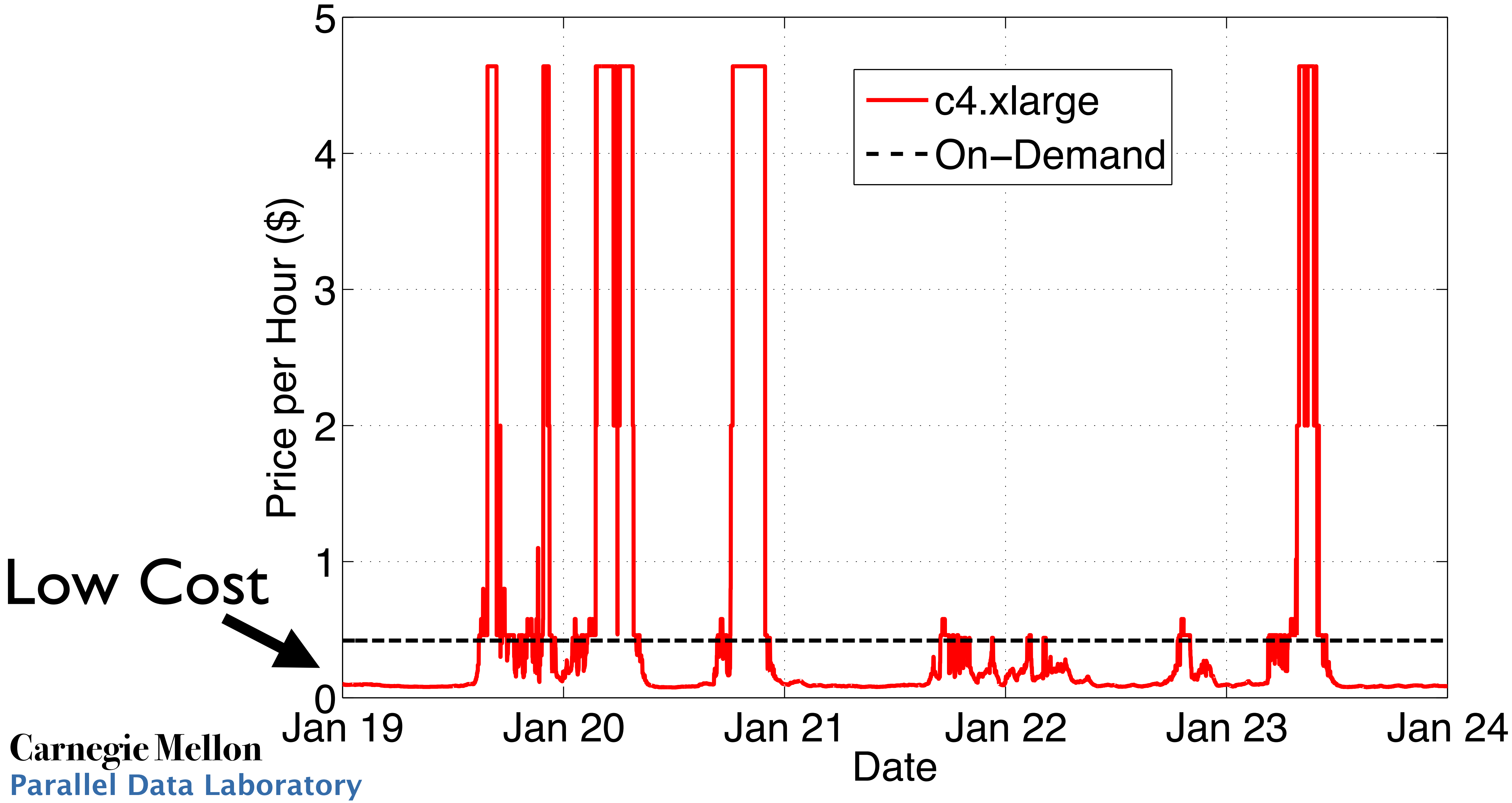
Transient resources much cheaper

- Often 75-85% cheaper to use Spot Instances



Transient resources much cheaper

- Often 75-85% cheaper to use Spot Instances



Spot Market Details

- Many different spot markets
  - each instance type, in each availability zone, in each datacenter
  - empirically, markets are uncorrelated
- If pre-empted, Amazon issues refund
  - during first hour only
- Aquire resource(machines) by specifying:
  - <spot market, bid price, number of machines>

Tributary Changes how we Aquire Resources

- Uses transient instead of reliable resources
  - while addressing bulk preemptions
- Uses resource from multiple spot markets
  - predicts allocation P[preemption]
  - tracks inter-market correlations
  - maintains diverse resource buffer

Tributary Components

- Predicting resource reliability

- Constructing resource footprint

Influencing P[preemption]

- User's bids influence P[preemption] of spot instances
  - bid delta = user bid price - spot market price
- Bigger Delta
  - lower P[preemption] and higher cost
- Smaller Delta
  - higher P[preemption] and lower cost

Predicting P[preemption]

- Predict P[preemption] as a function of bid deltas
- Extract features
    - calendrical
    - temporal
- Plug features into LSTM Model
    - models EC2 as a sequence of events

Constructing the Resource Footprint

- Need to achieve capacity to satisfy SLO of client workload
- Need sufficient diversity across markets

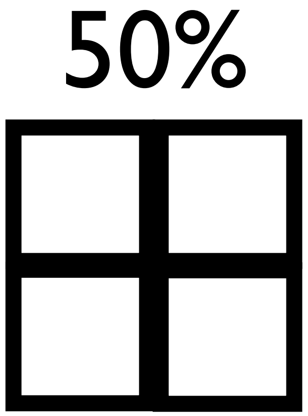
```
While expected request capacity < SLO:
    Add resource that increases expected cost the least
    and increases request capacity the most.
```

Computing Expected Request Capacity

- Compute probability of exactly 0 - N resources not pre-empted
- Accounts for spot market dependencies
- Encourages diversity

Computing Expected Request Capacity

- Compute probability of exactly 0 - N resources not pre-empted
- Accounts for spot market dependencies
- Encourages diversity

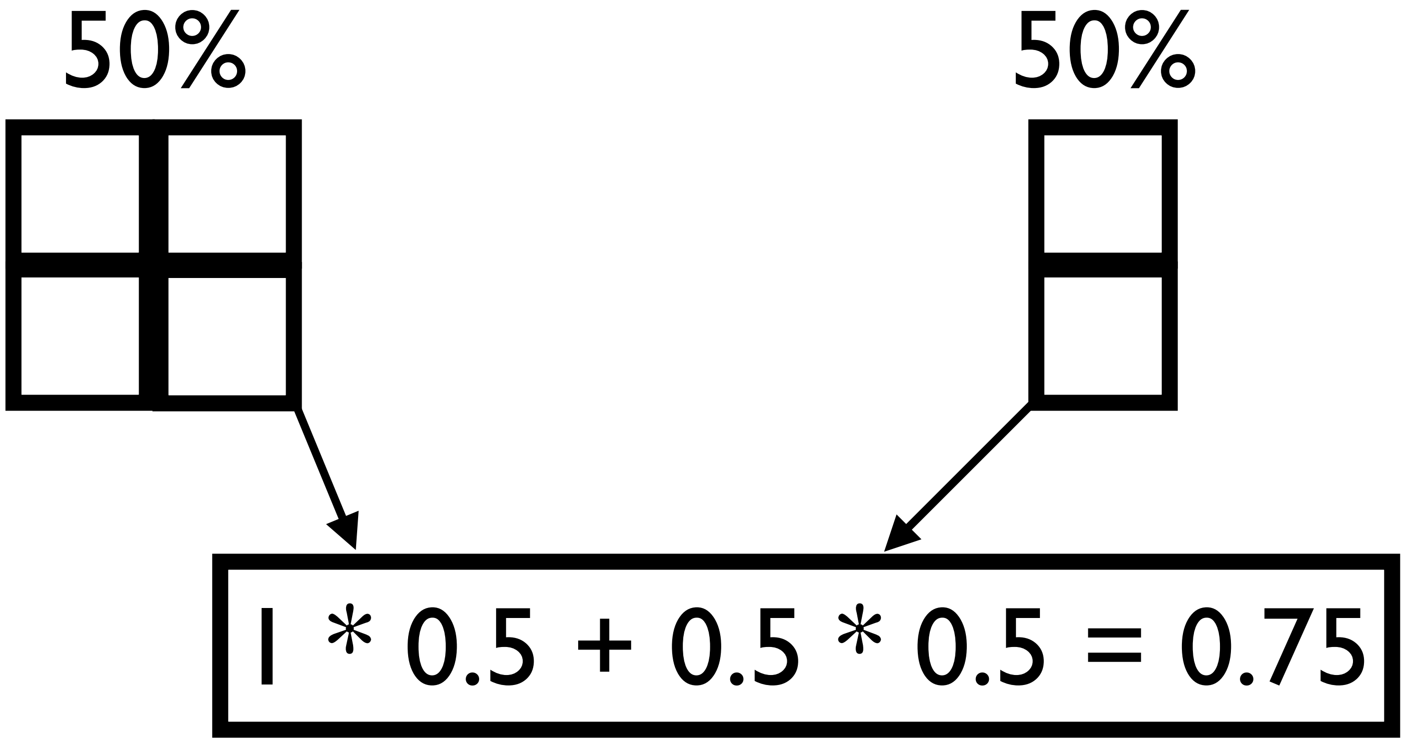


Computing Expected Request Capacity

- Compute probability of exactly 0 - N resources not pre-empted
- Accounts for spot market dependencies
- Encourages diversity

Computing Expected Request Capacity

- Compute probability of exactly 0 - N resources not pre-empted
- Accounts for spot market dependencies
- Encourages diversity

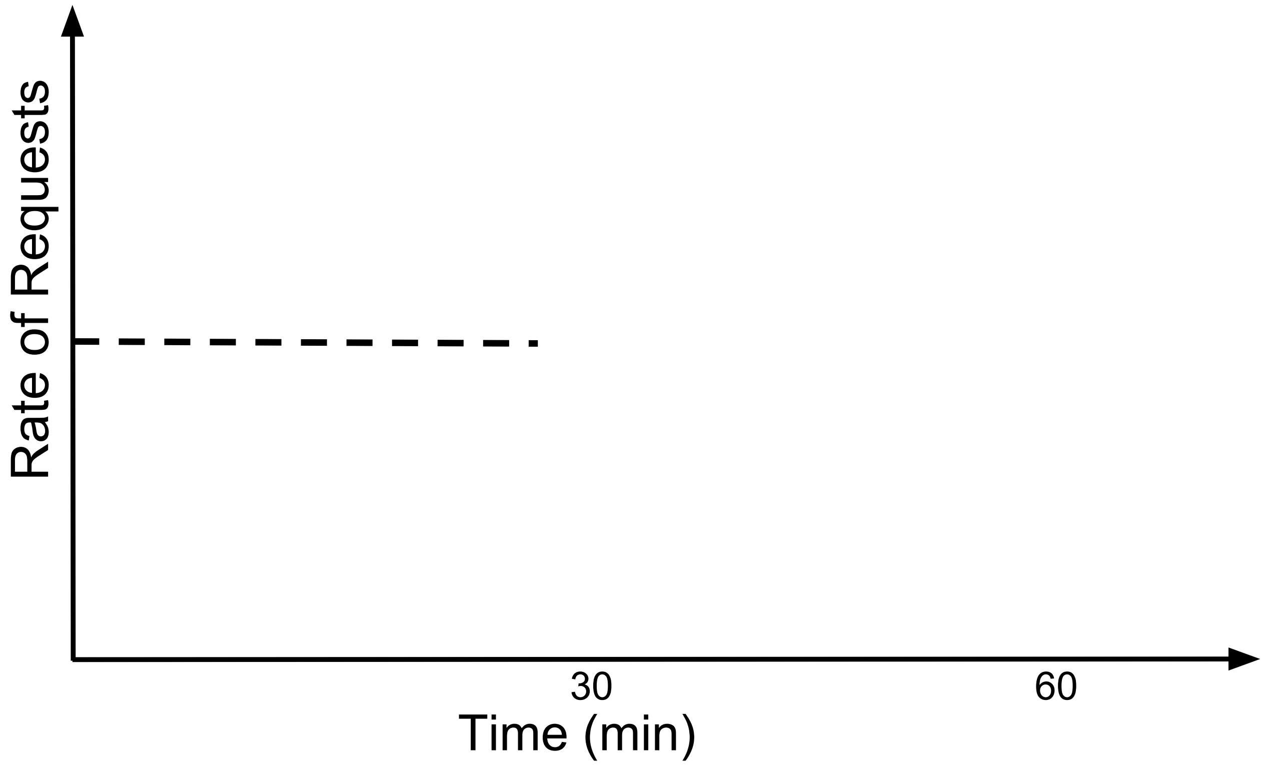


$$\boxed{1 * 0.5 + 0.5 * 0.5 = 0.75}$$

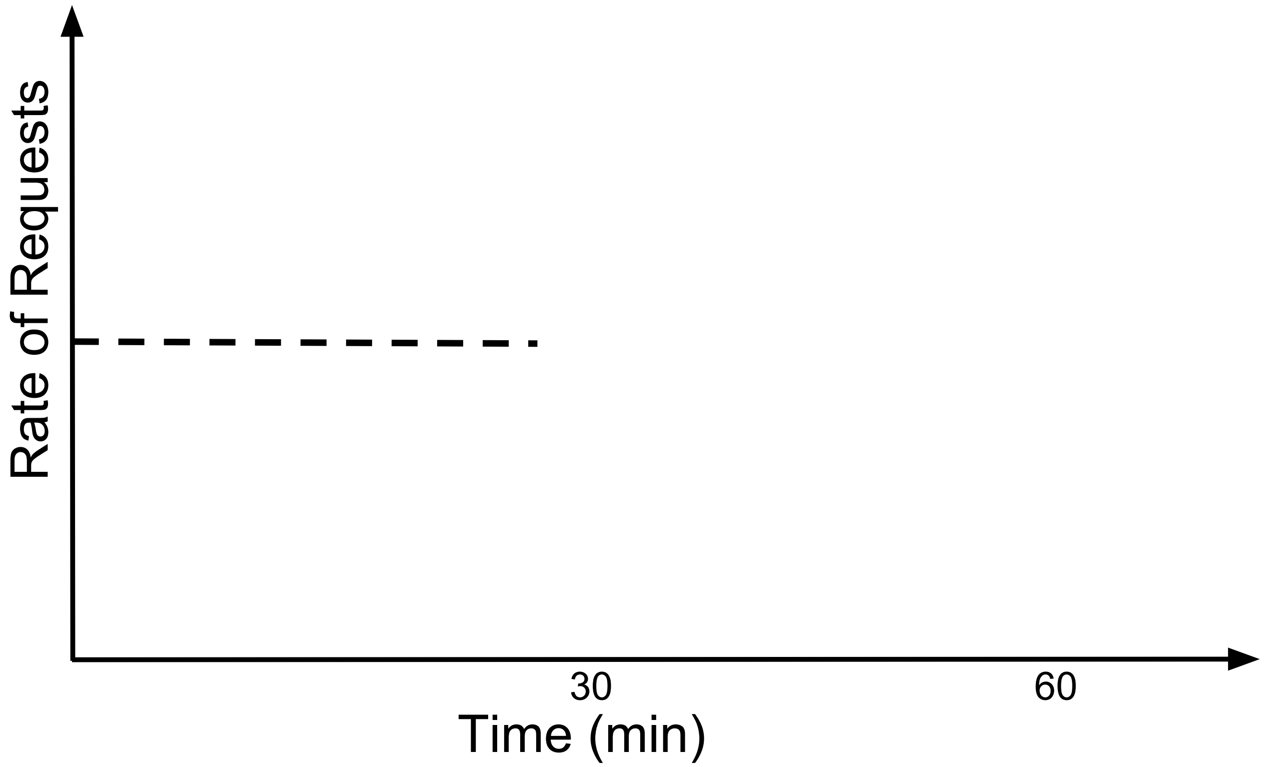
So Why Does this Work?

- Creates a diversified, oversized footprint
  - able to tolerate preemptions
  - little or no extra cost
- Handles unexpected workload spikes
  - handled via oversized natural resource buffers

Time for an Example

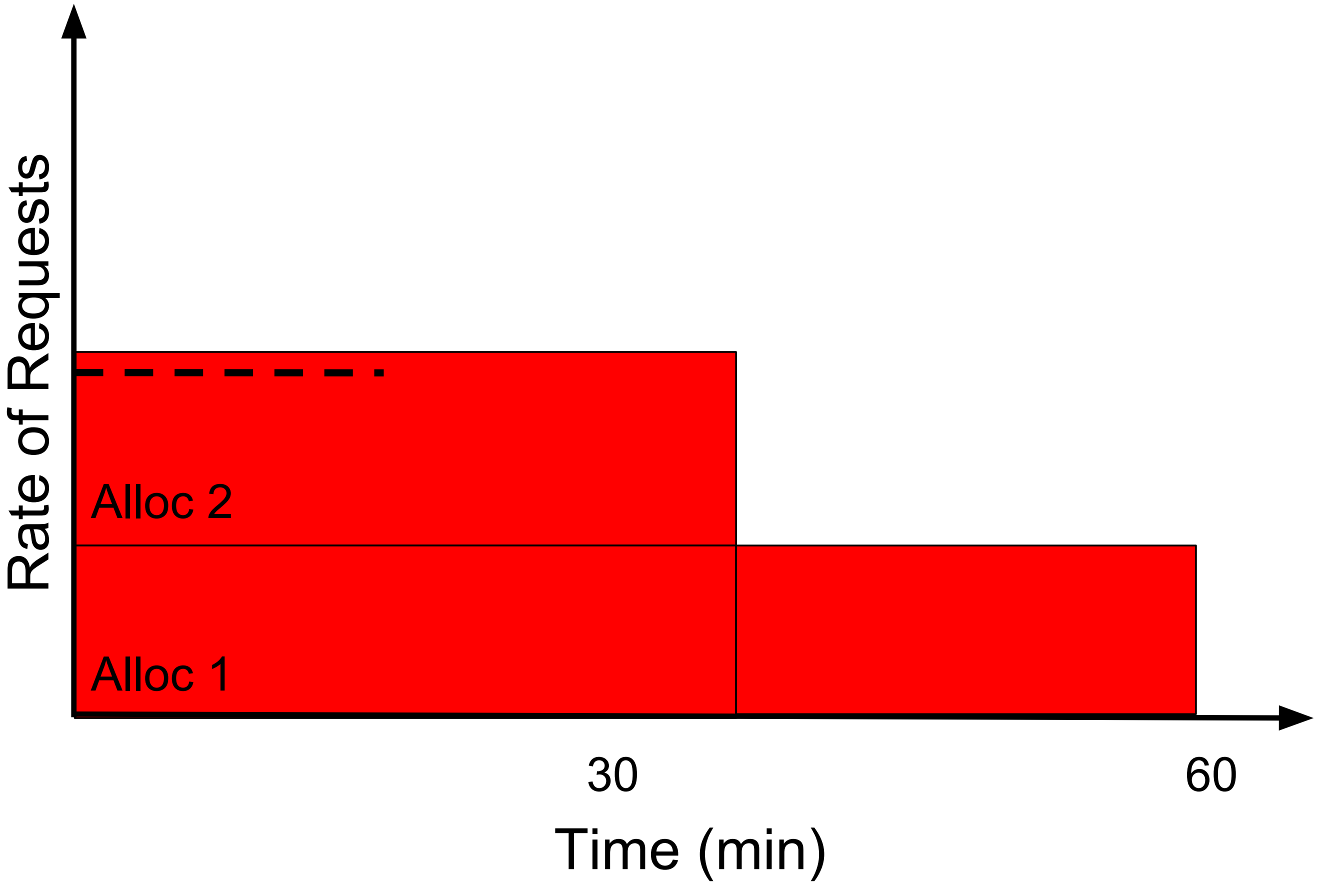


AutoScale

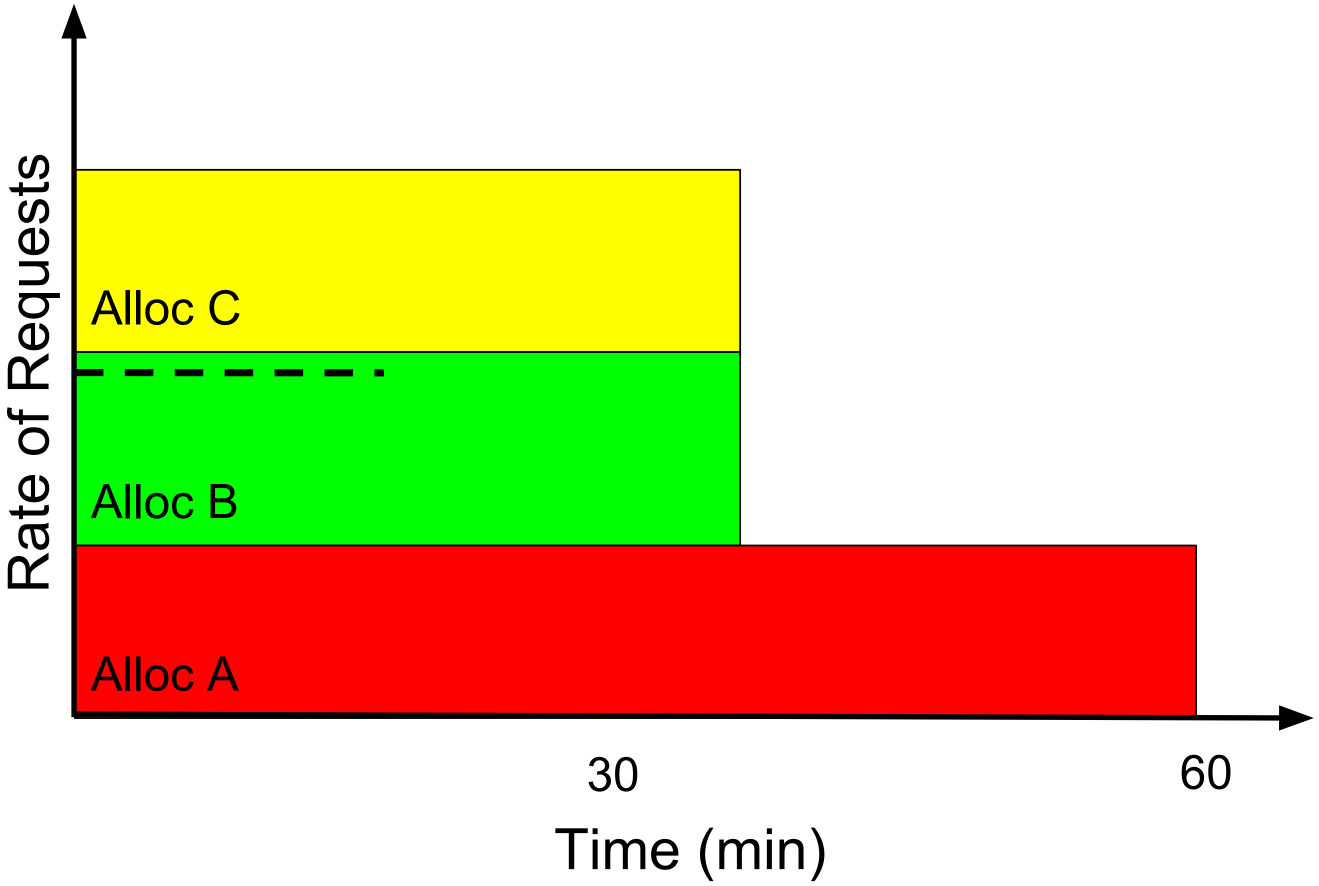


Tributary

Time for an Example

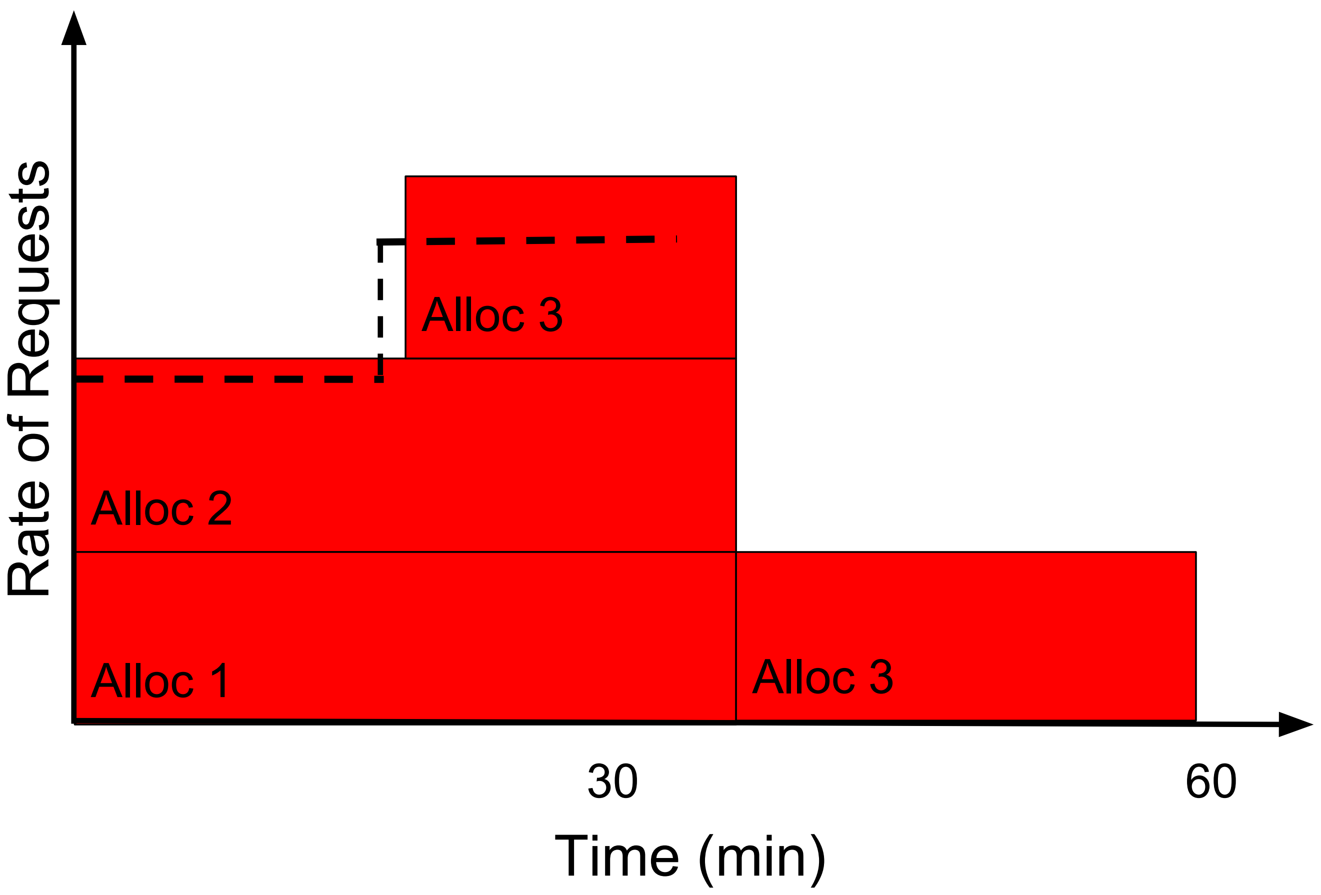


AutoScale

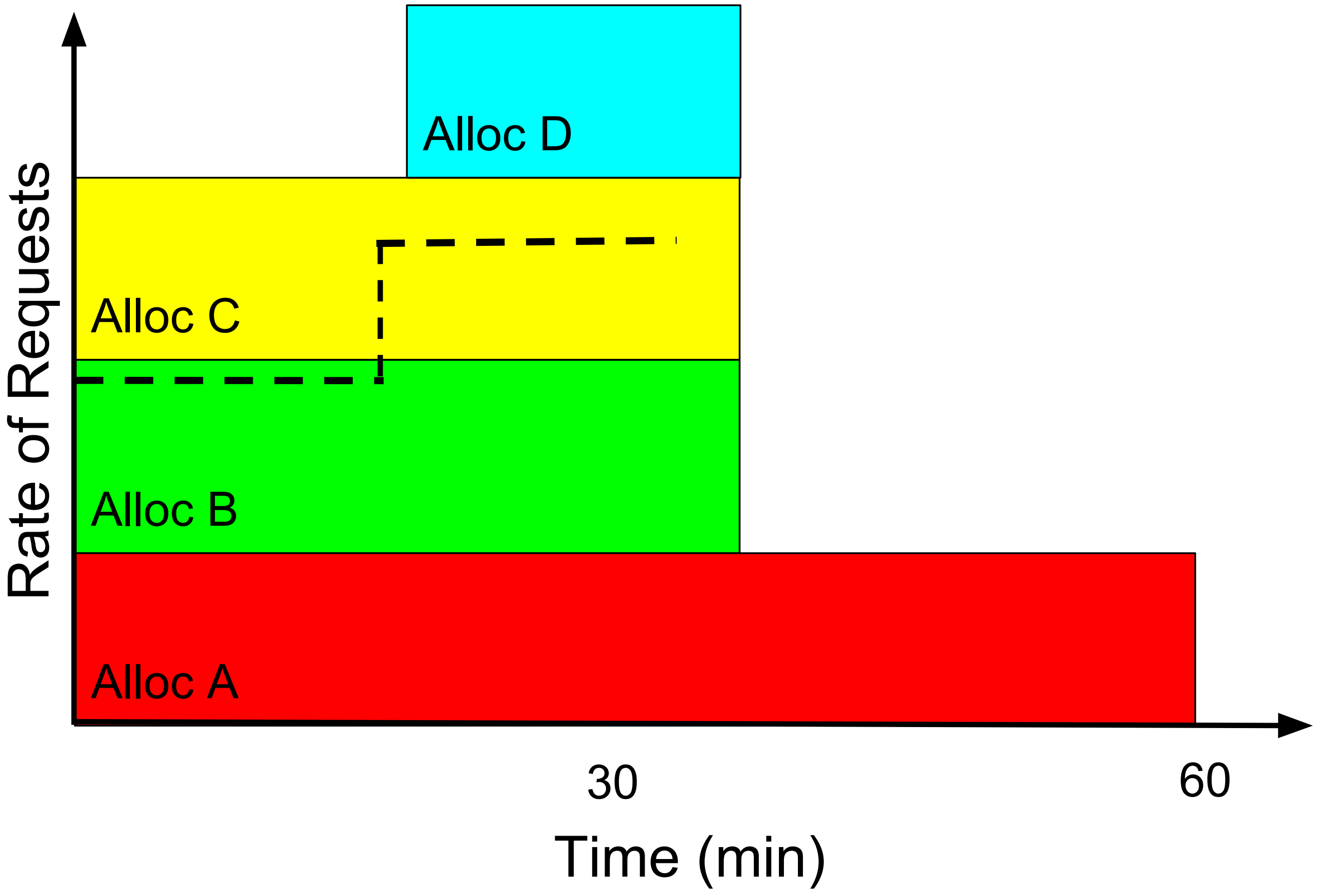


Tributary

Tributary Serves More Requests

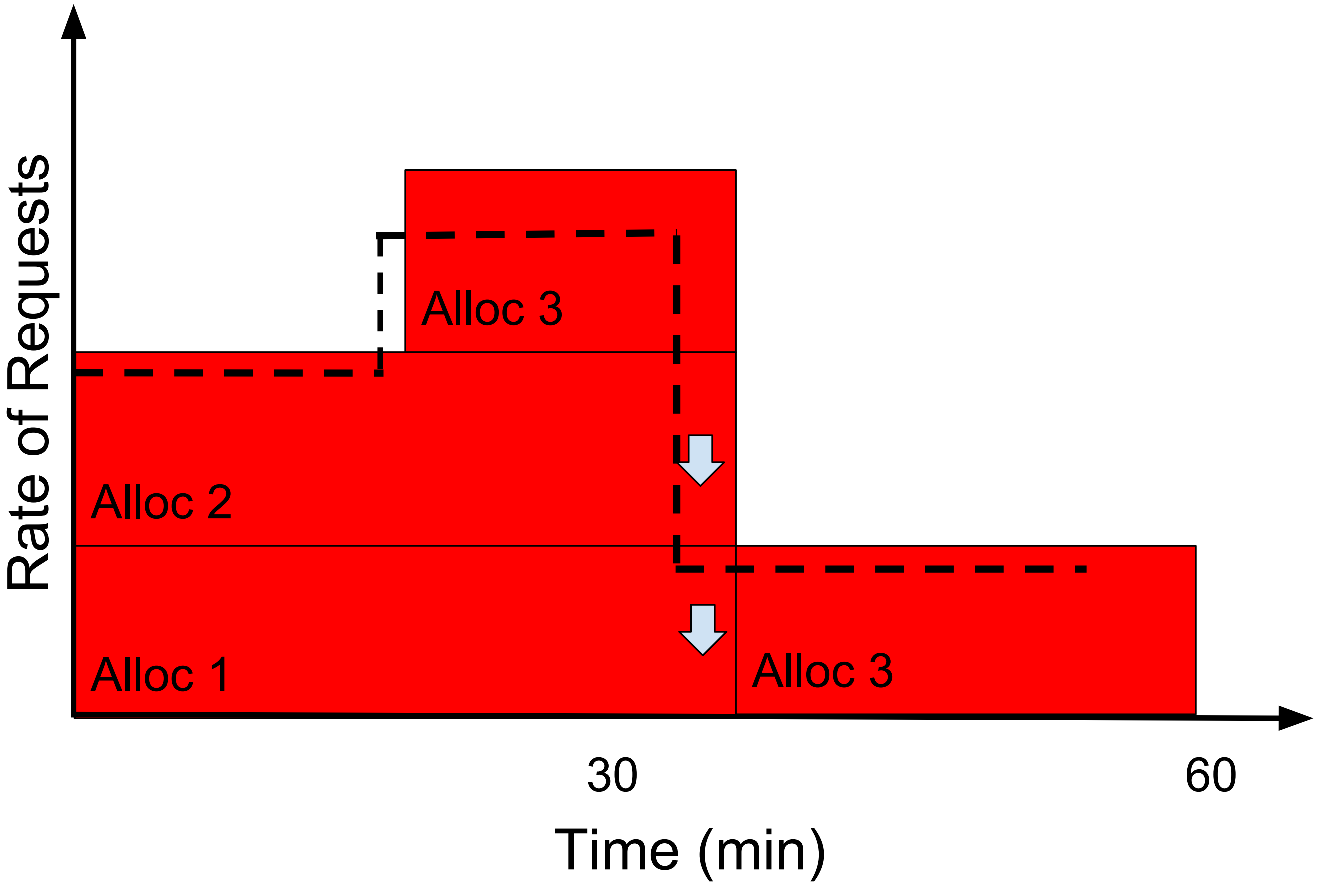


AutoScale

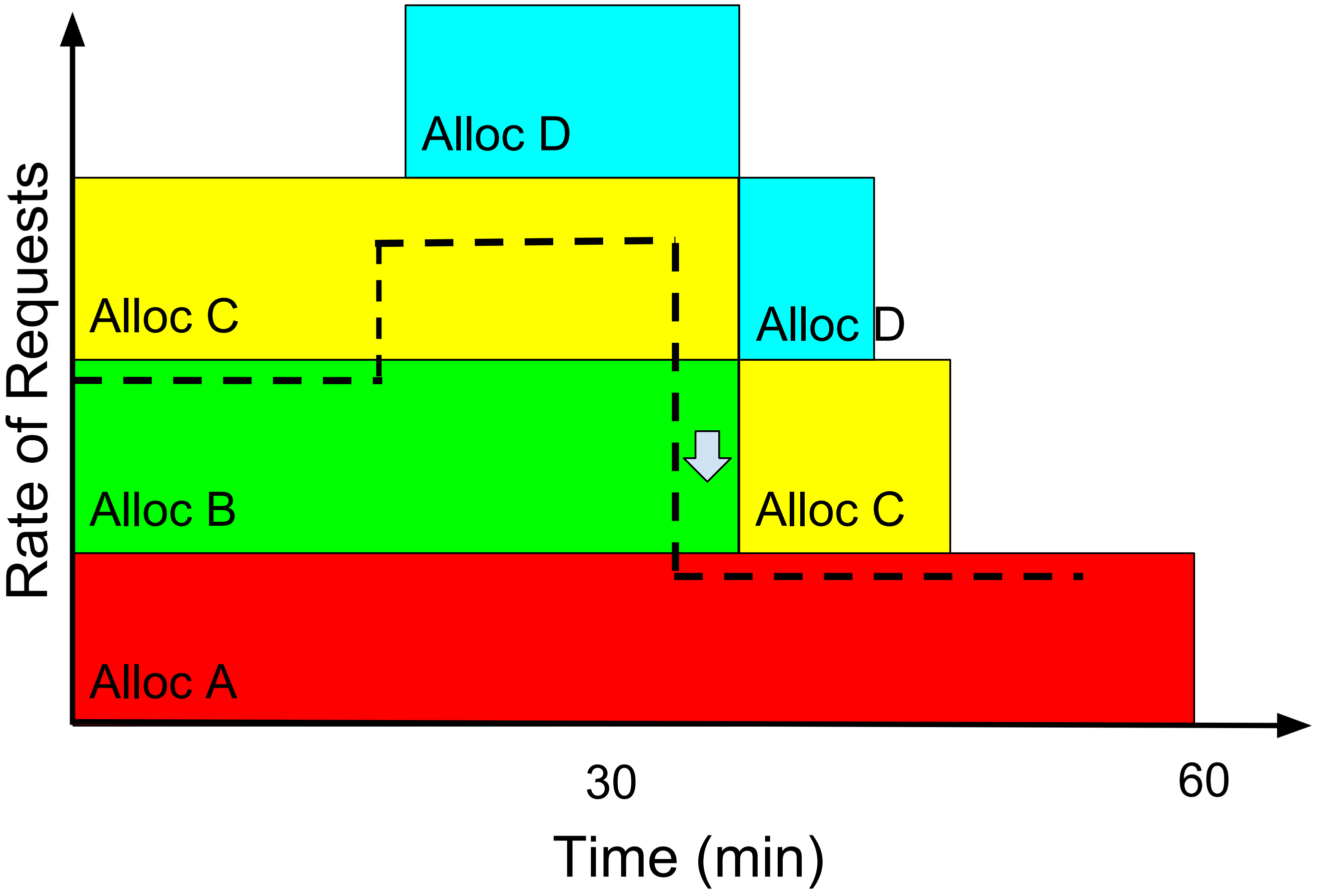


Tributary

Request Rate Decreases



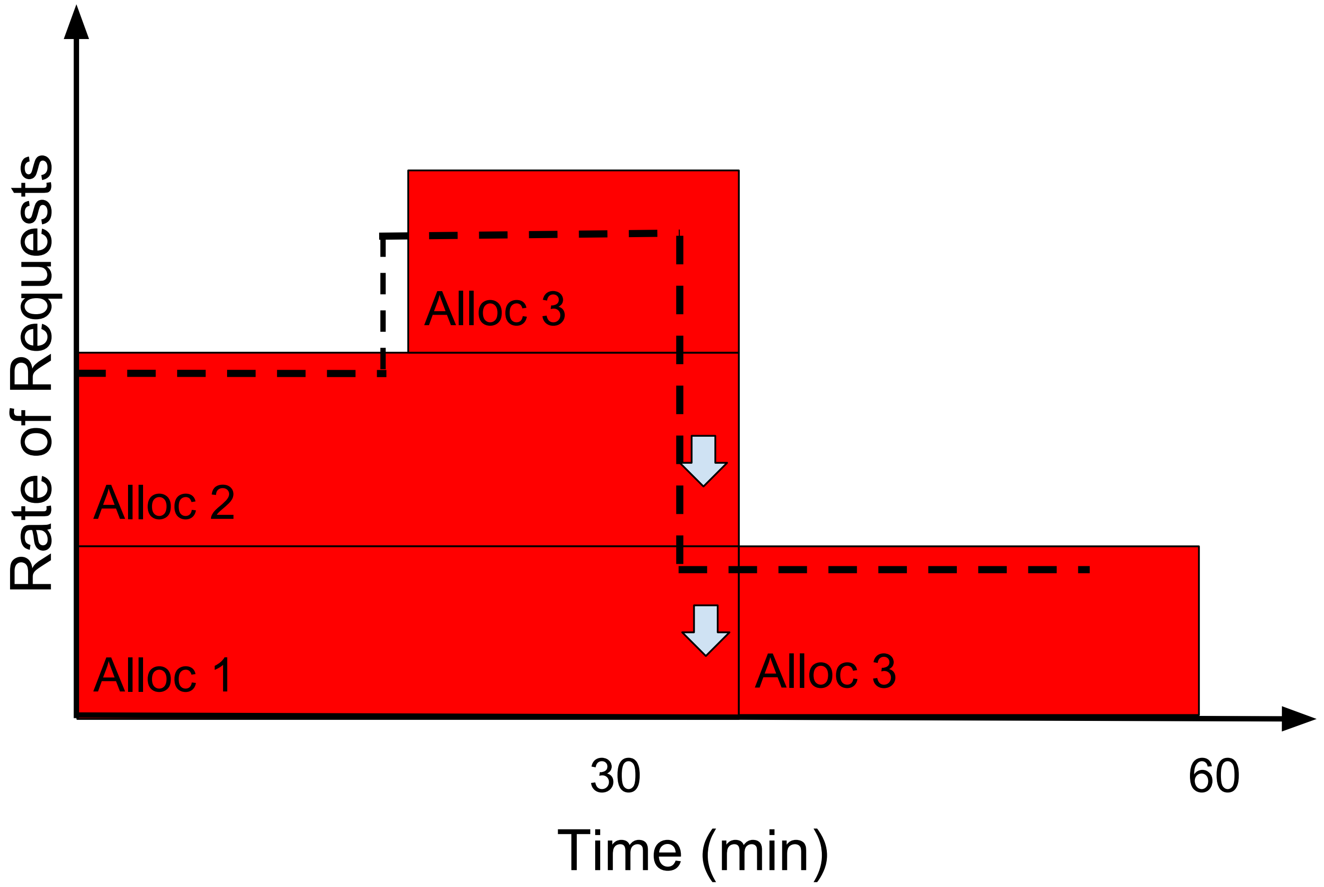
AutoScale



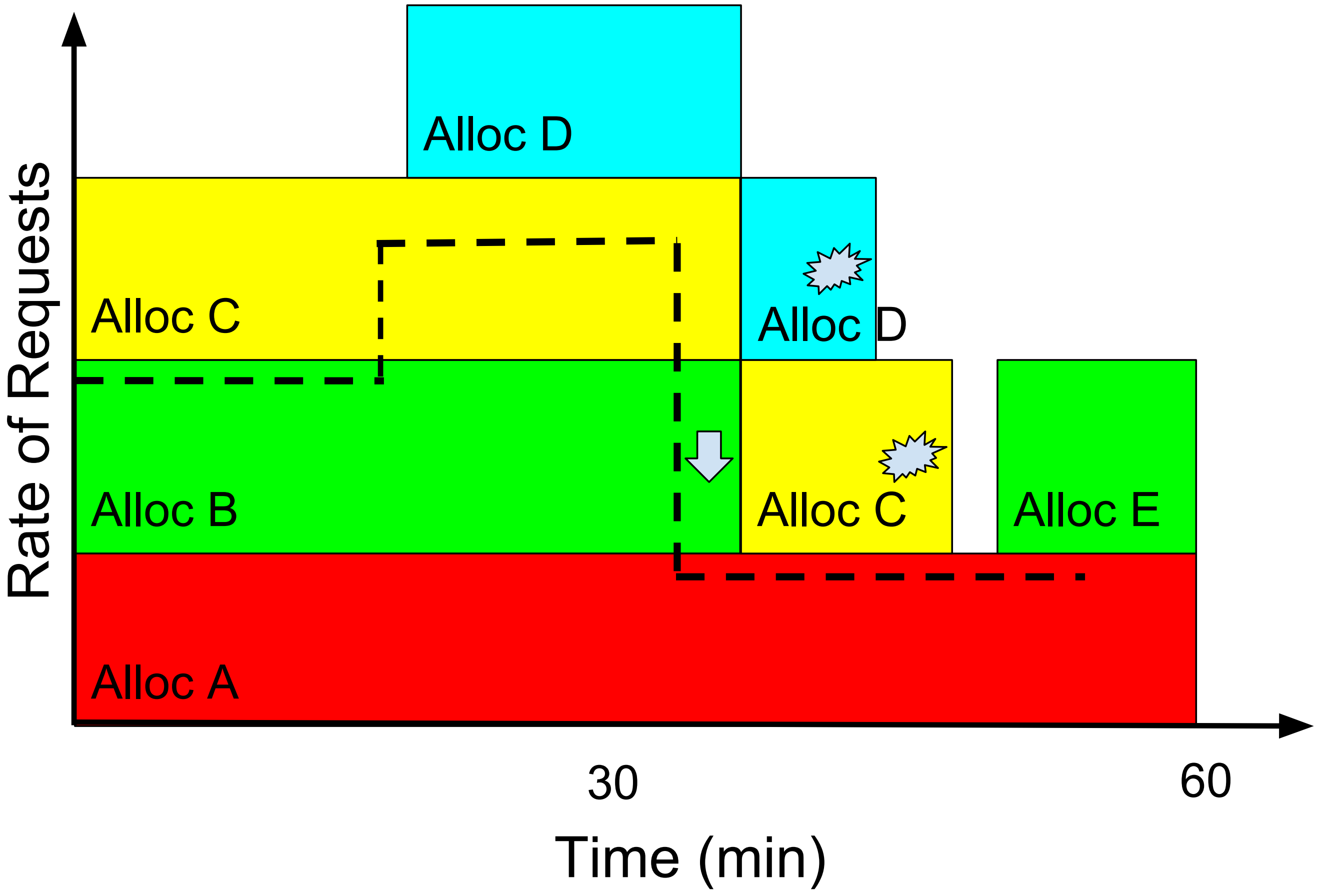
Tributary

Carnegie Mellon
Parallel Data Laboratory

Tributary’s Resources are Pre-empted



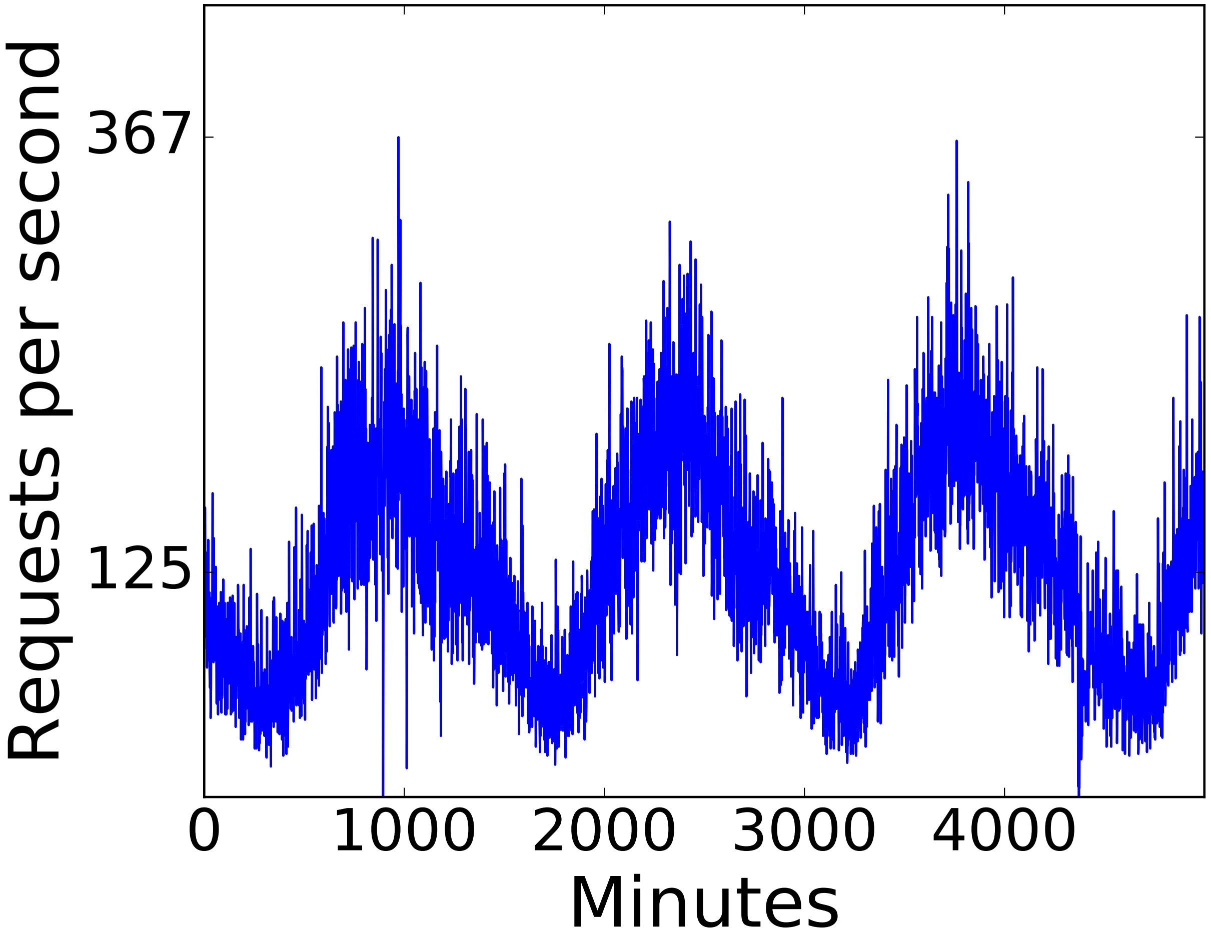
AutoScale



Tributary

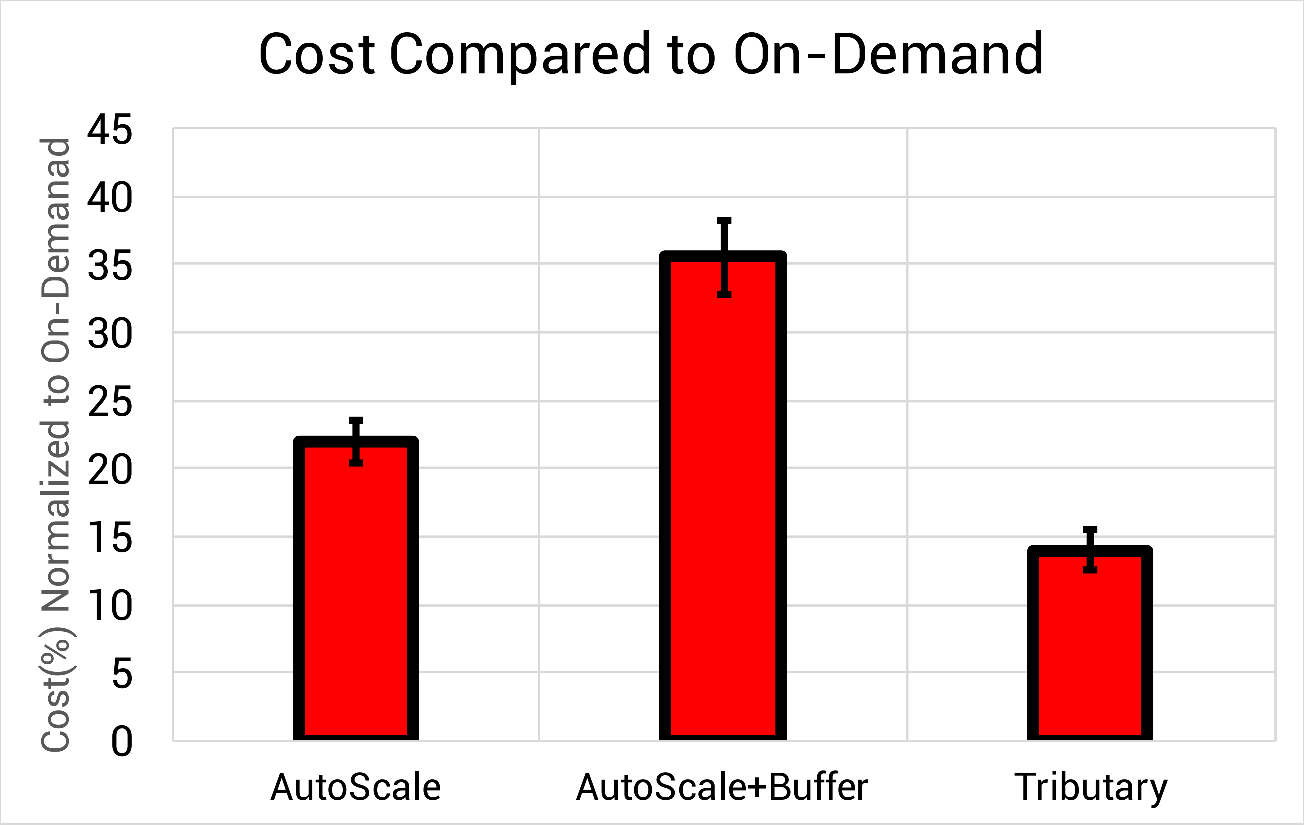
Experimental Setup

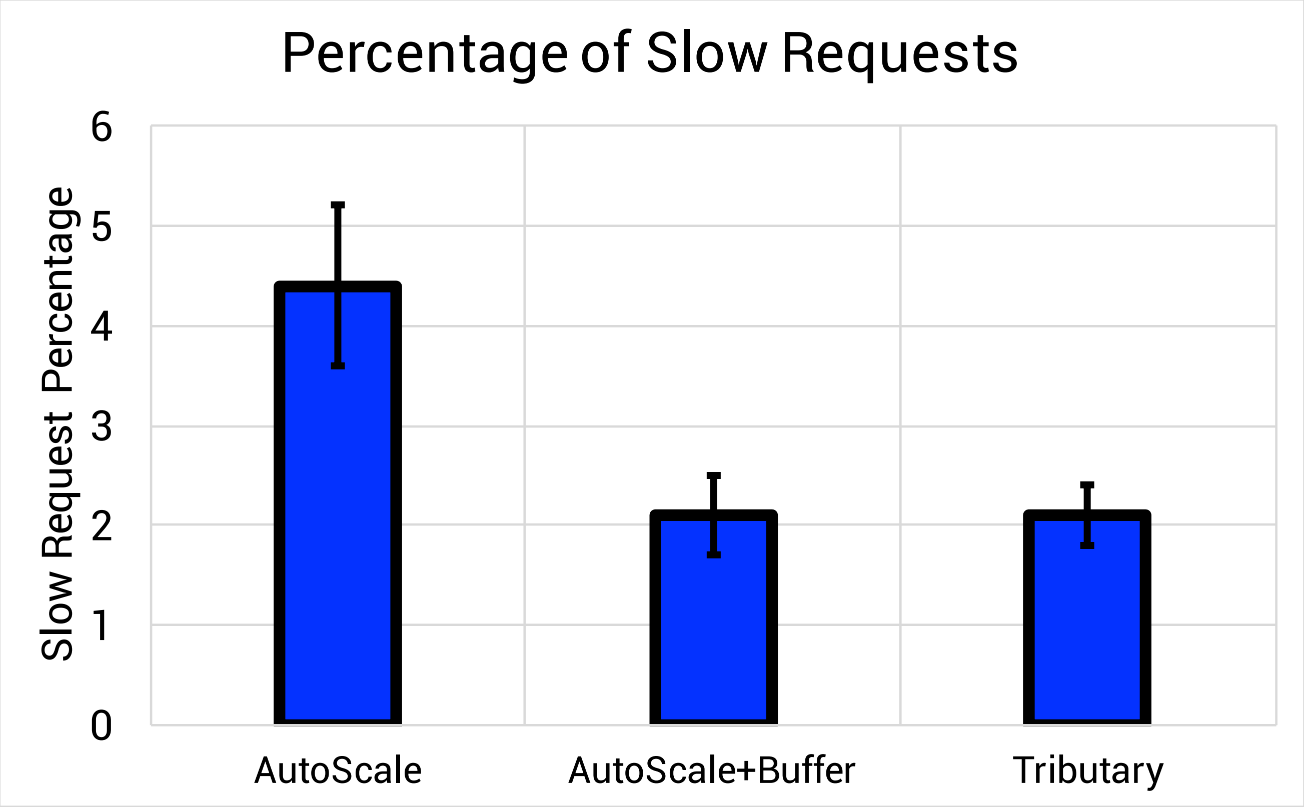
- 4 Traces Evaluated
  - show Clarknet
- 3 Scaling Policies
  - show reactive
- Comparisons
  - Autoscale on spot
  - Autoscale+Buffer on spot
  - Tributary



Comparing to AutoScale

- AWS AutoScale
  - AWS service that acquires cheapest spot instances





Other Interesting Results

- Across 4 traces Tributary reduces cost by 47-62%
- Outperformed recent research systems
  - ExoSphere [Sharma 2017]
  - Proteus [Harlap 2017]
- Only ~50% of cost saving come from preemptions

Conclusion

- Provides reliable service using transient resources
- Uses diversified buffers of resources
- Reduces cost by ~85% over on-demand