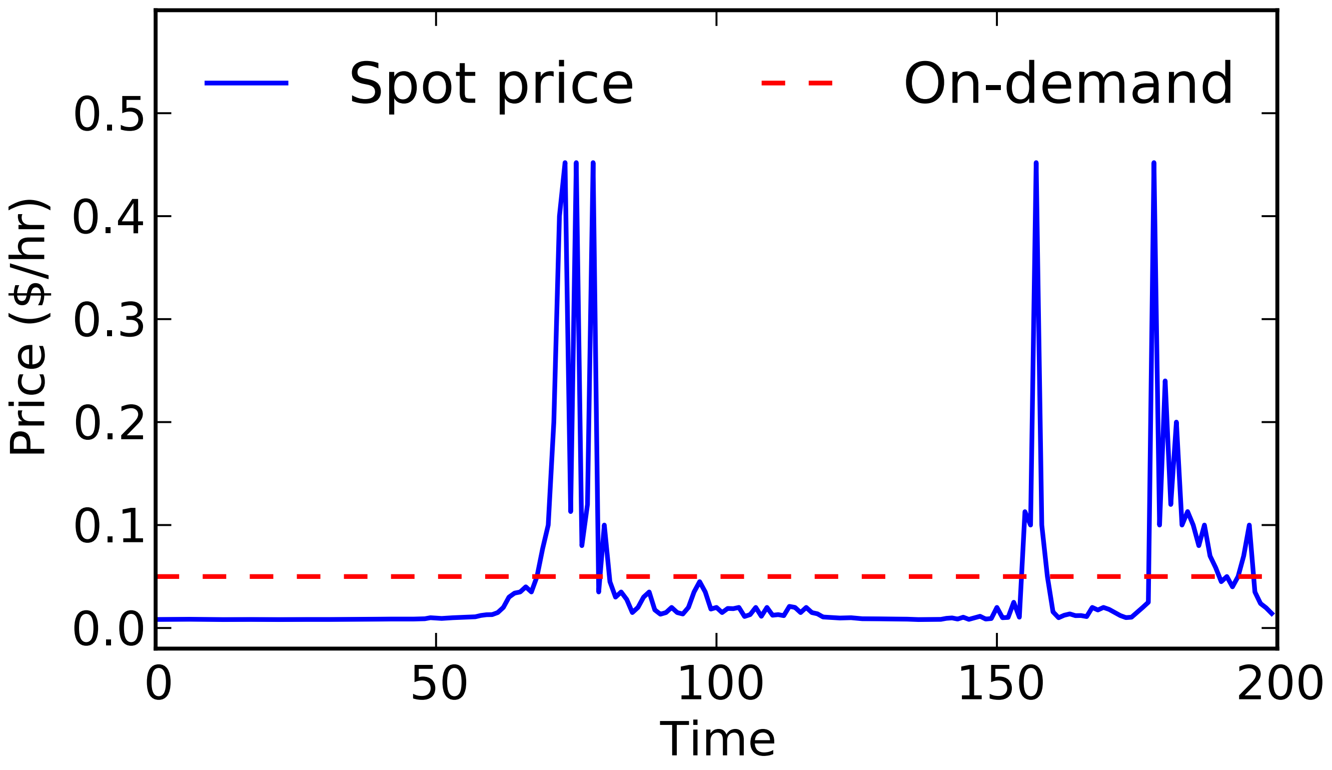
Transient Cloud Computing

Prateek Sharma

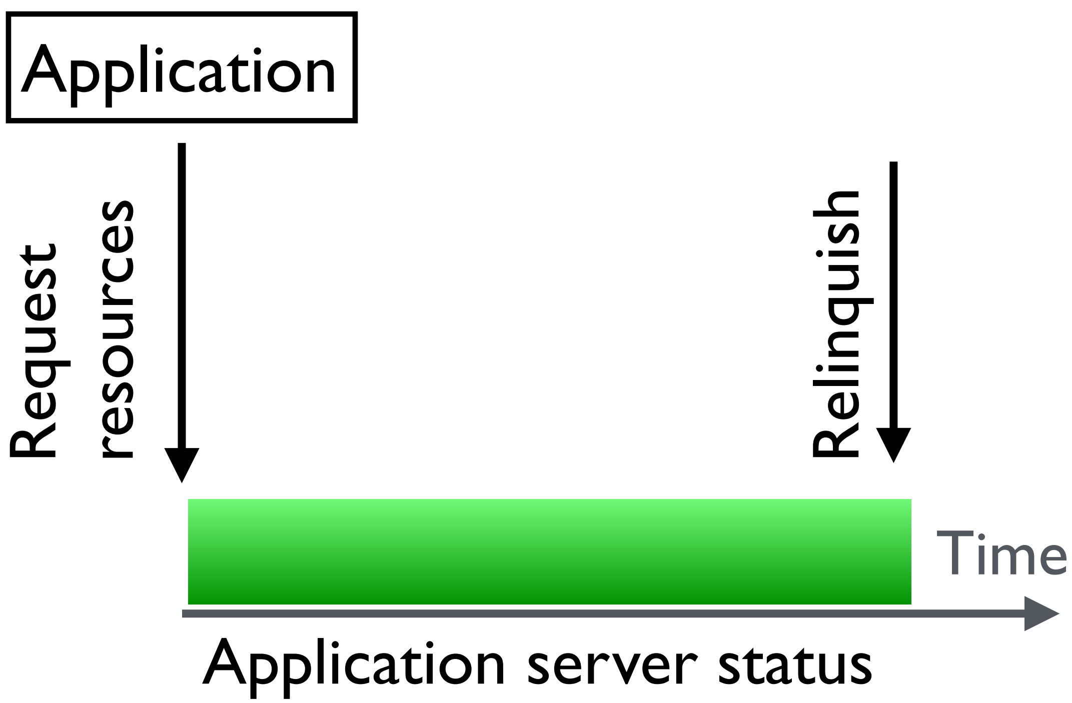
Cloud VM Pricing

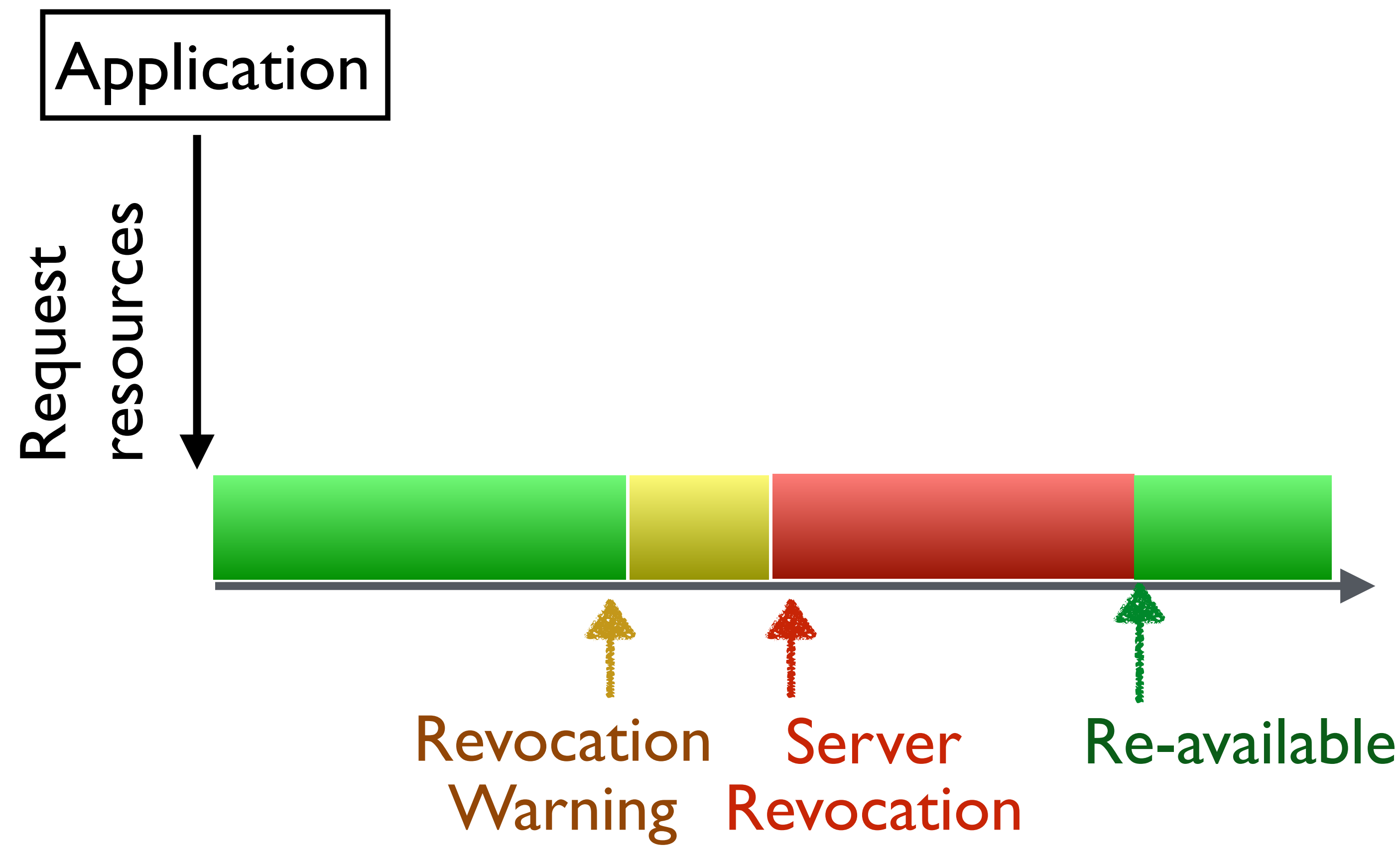
- Conventional **on-demand** instances: fixed per-hour/second pricing
- **Reserved** instances: Long-term lease (1/3 years), cheaper than on-demand
- **Transient** instances: Price and availability varies over time

- Classic example: EC2 spot instances
- Price set by continuous second-price auction
- If price > user's bid, the instance is terminated after 2 minutes



New Paradigm: Transient Computing





Conventional resources

- Continuous availability

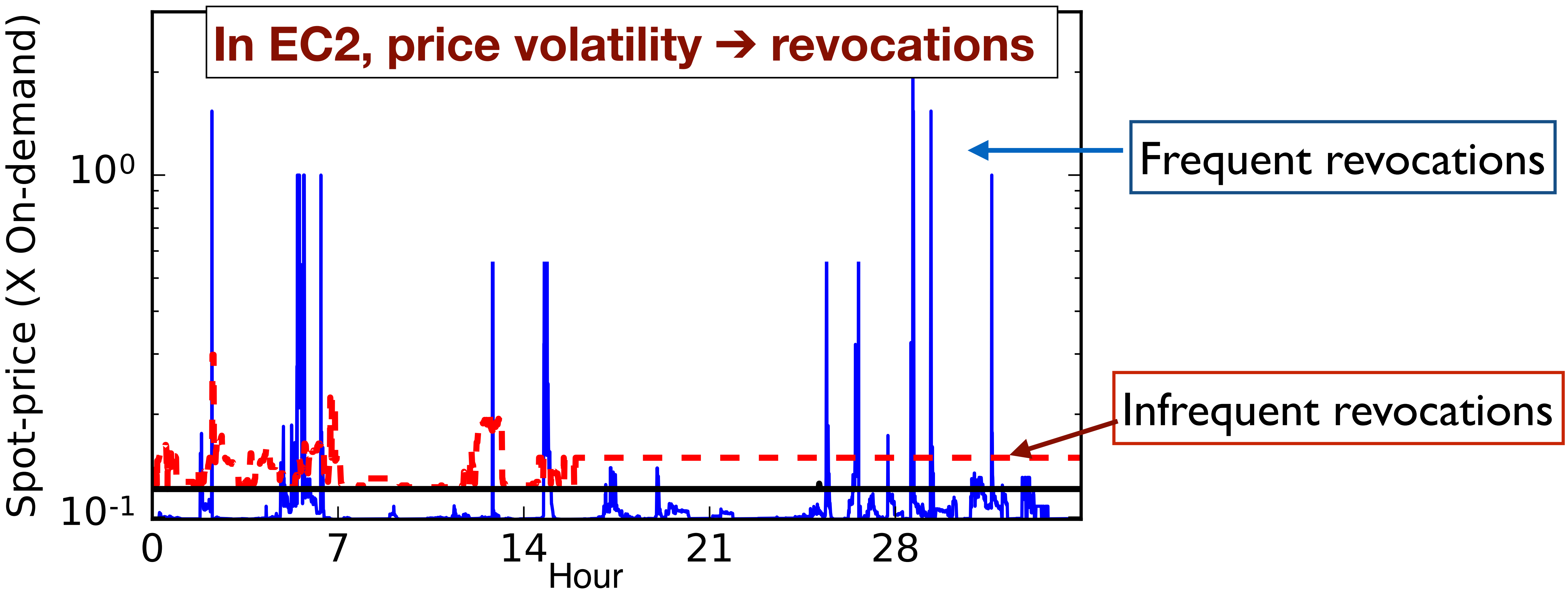
Transient Resource Availability

- Access unilaterally revoked
- Applications face disruption
  - Cannot assume continuous availability

Diversity In Transient Cloud Servers

- Transient server revocations depend on server-type, location
  - >5,000 EC2 spot markets

Ex: {server-type: m4.small, region: us-east-1, data-center-zone: A, OS: Linux}



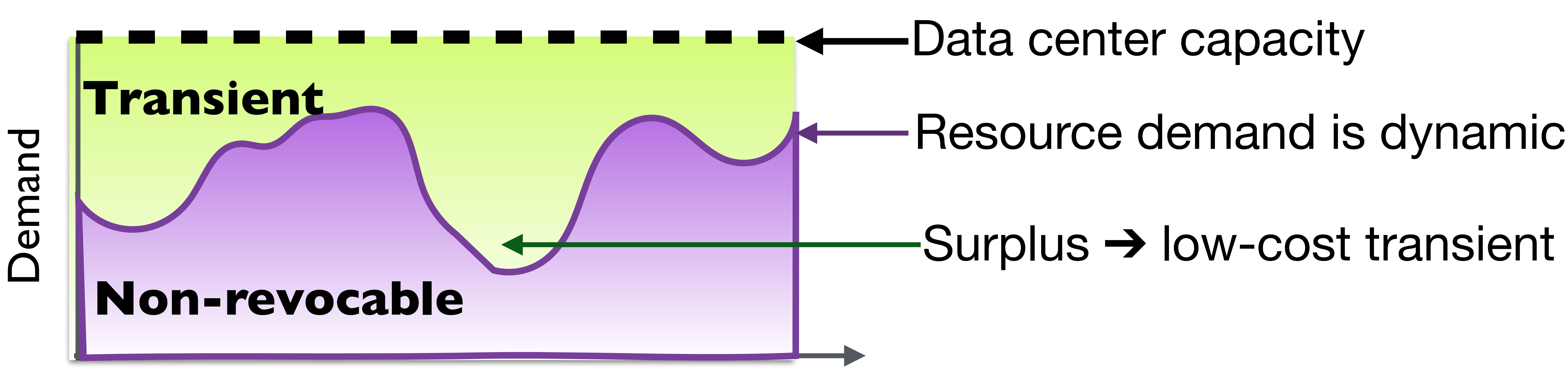
Transient cloud servers: diversity in demands and revocation frequency

Transiency Is Common In The Cloud

All major cloud providers offer transient servers

	Amazon EC2 Spot Instances	Google Cloud Preemptible VMs	Azure Batch VMs
Lifetime	2-48 hours	<24 hours	~12 hours
Discount	50-90%	70%	80%

- Revocations ➔ Servers have limited lifetimes
- Conventional cloud servers: non-revocable on-demand servers



Transient Server Characteristics

- **Frequent revocations**: MTTFs of hours/days, not years
  - Run applications without disruption/performance degradation?
- **Advance warning**: Not sudden fail-stop failures
  - How to mask revocations to reduce downtimes?
- **Heterogeneity**: Different price vs. availability tradeoffs
  - Resource management policies to manage revocation risk?

- **How can applications make effective use of transient resources?**
- **Can we design systems with transiency-specific mechanisms and policies?**

Transiency-driven System Design Challenges

Applications

1. Reduce impact of revocations on availability and performance
2. Reduce number and frequency of revocations
3. Abstractions for transient servers
4. Transient resource reclamation to avoid revocations

Transient Servers

Migrating Application State

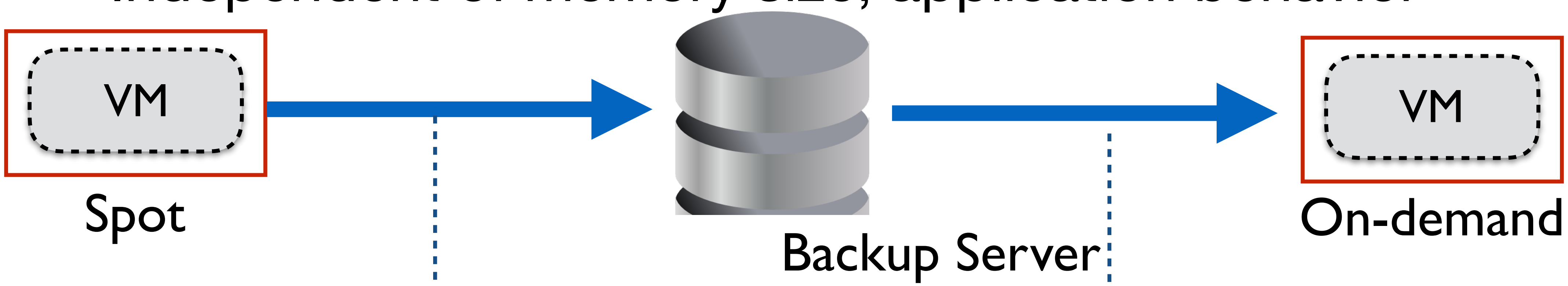
Basic idea:

Run on spot when possible. Migrate to on-demand when revoked.

- Existing technique: VM Live Migration
  - Migration may not complete within advance warning (2 mins)
  - Incomplete migrations result in state-loss and unavailability
- **Can we completely migrate VMs within warning period?**

SpotCheck VM Migration

- **Bounded-time VM Live Migration**
  - Ensures VMs migrate within a specified time duration
  - Independent of memory size, application behavior

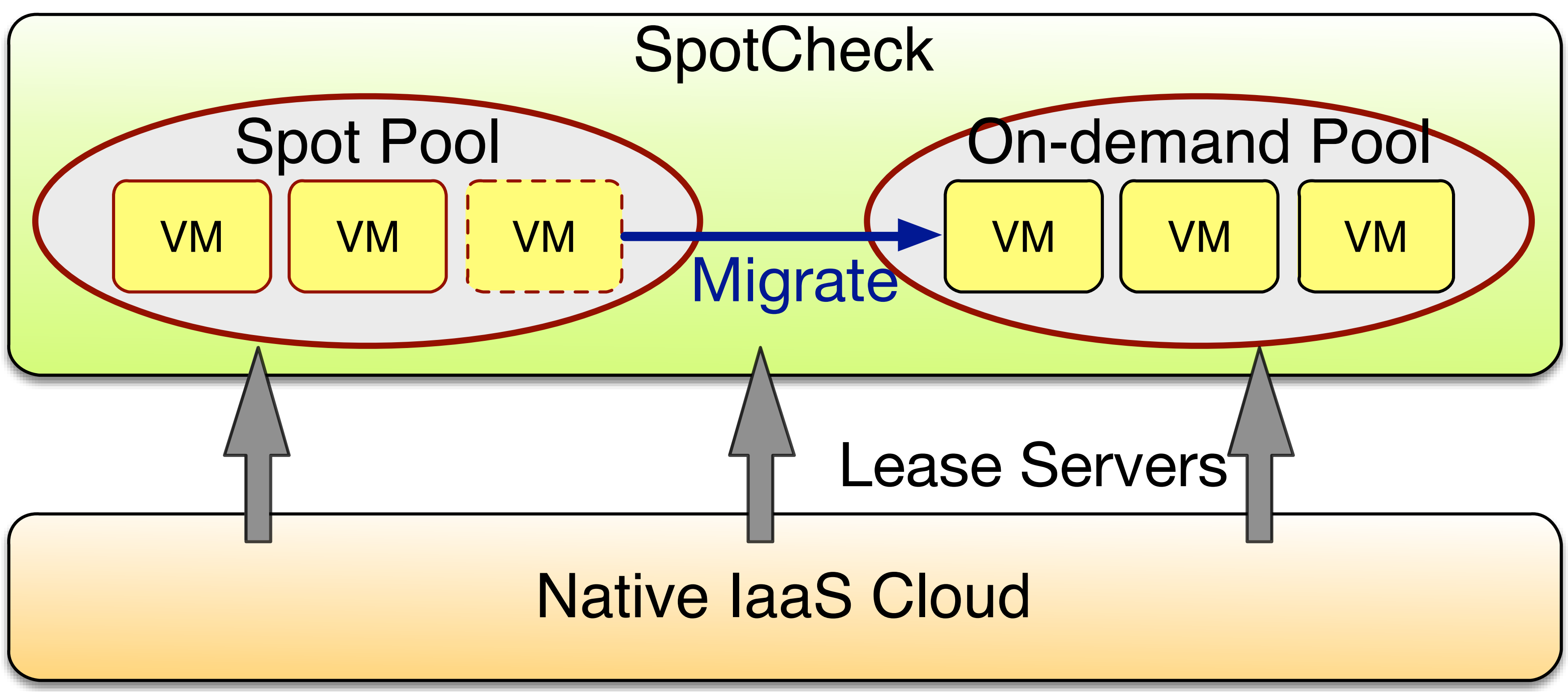


- Continuously checkpoint memory
- Residual dirty pages sent in bounded time

- Restore skeleton state immediately
  - VCPU state, page-tables,..
- Copy remaining pages on access

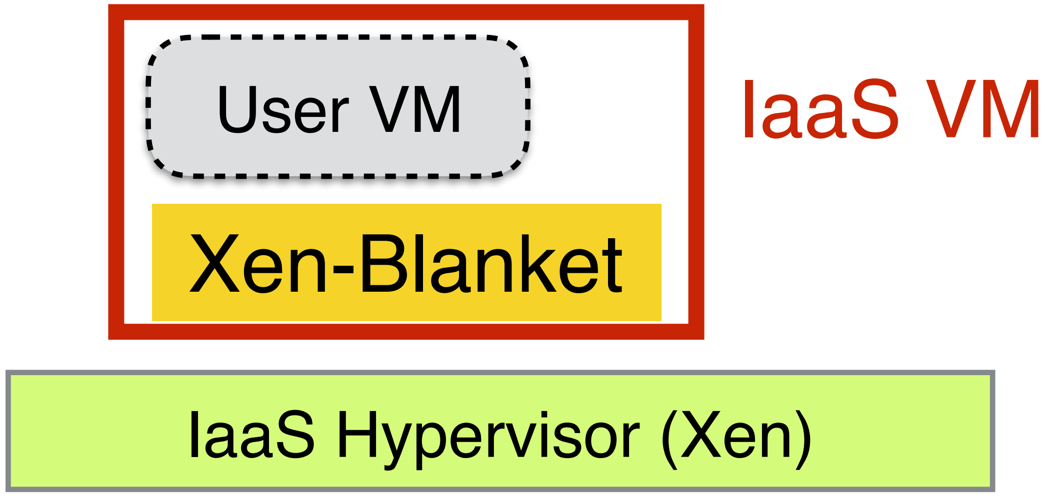
SpotCheck: A Derivative Cloud

- Derivative cloud: Cloud middleware derived from native cloud
- SpotCheck: Illusion of low-cost, non-revocable servers to run unmodified apps
- Multiplex spot and on-demand pools across multiple customers



Implementing SpotCheck

- How to migrate VM state in public clouds?
  - Migration and other hypervisor functionality not exposed
  - Solution: Nested Virtualization (Xen-Blanket)
  - Bonus: can run multiple nested VMs



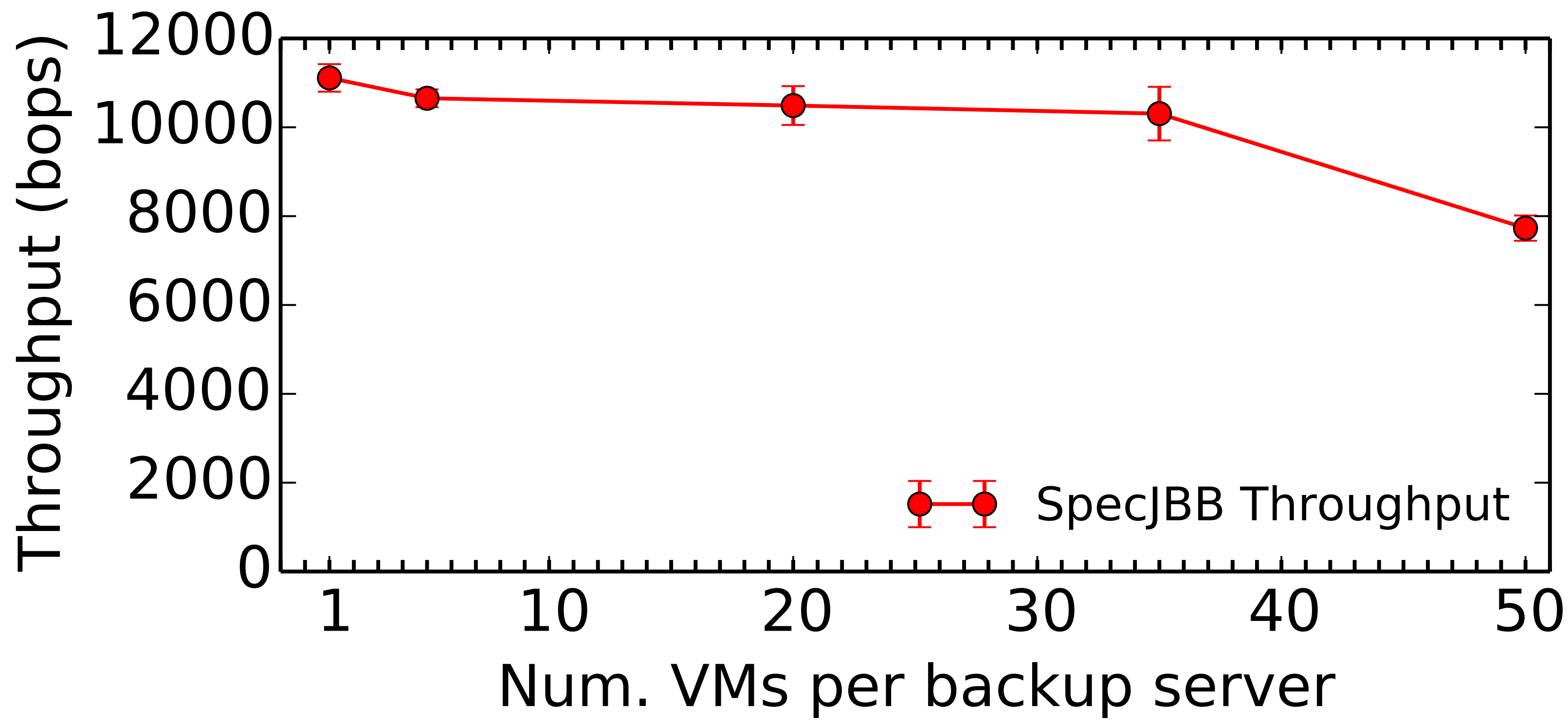
- Mitigating concurrent revocations:
  - Map customer VMs to different spot servers
  - Map VMs from different spot servers to a backup server

SpotCheck Application Performance

- Performance and cost overhead of continuous checkpointing is low

SpecJBB	0.015%
TPC-W	16.7%

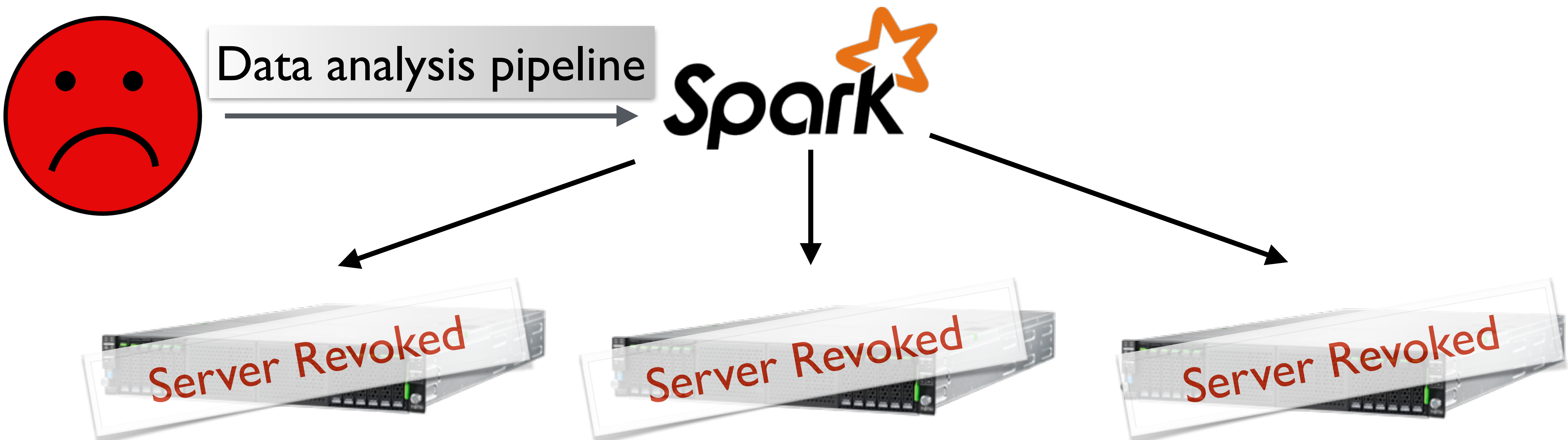
- Performance degradation due to continuous checkpointing is low



- Backup servers can support ~40 VMs
- Amortizes backup server cost

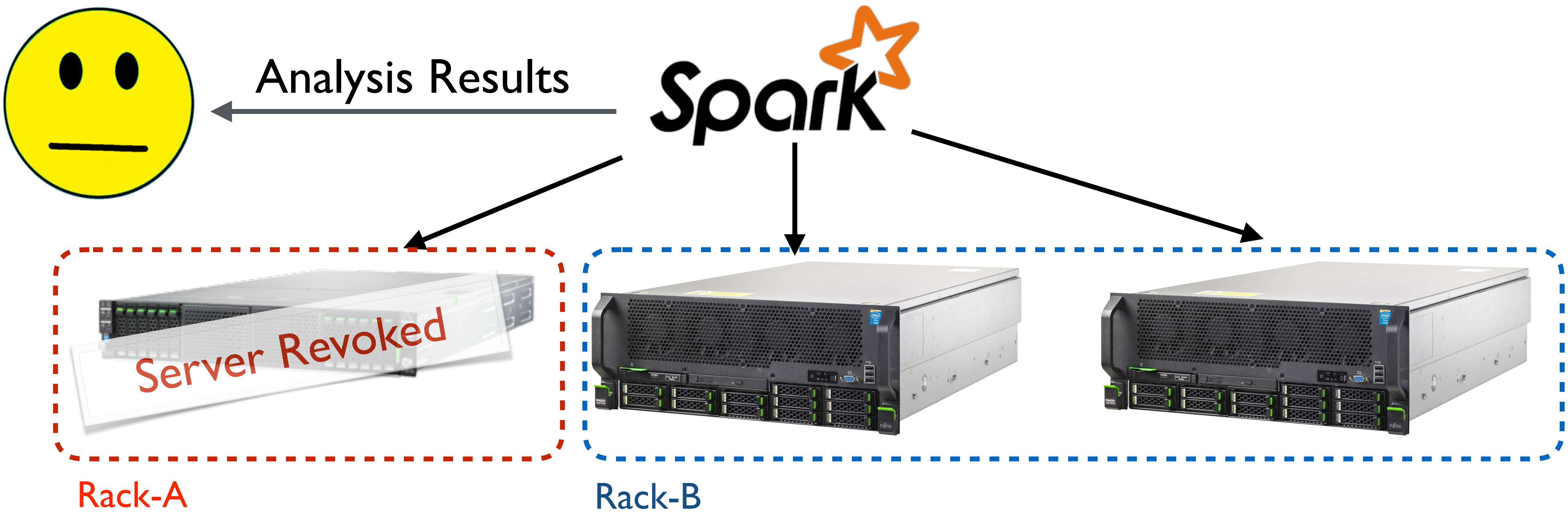
Challenges in Running Distributed Applications

SIGMETRICS '17



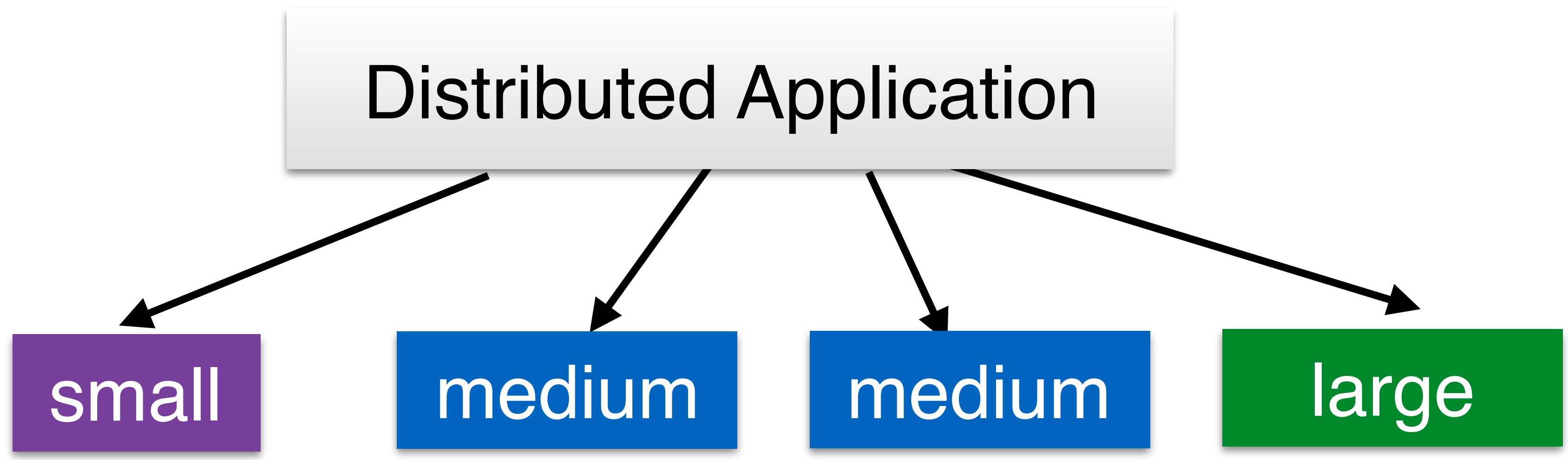
- All servers concurrently revoked ➔ complete resource starvation
- **Can we mitigate concurrent revocations for distributed apps?**

Key Idea: Select Heterogenous Servers



- Heterogenous servers: different configuration, data center racks/zones
- Many applications can tolerate partial failures, run in degraded mode
- Key: Uncorrelated revocations
  - Can we use heterogenous server selection in the cloud?

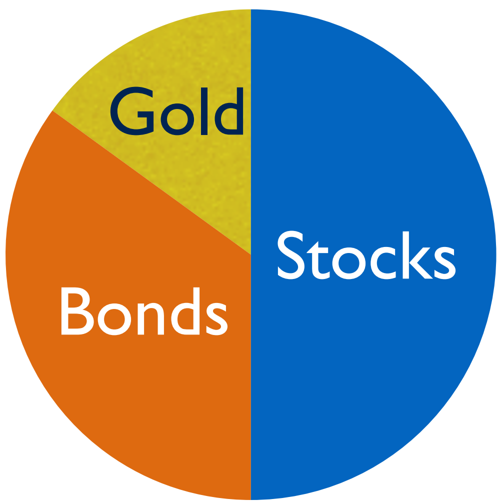
Heterogenous Server Selection

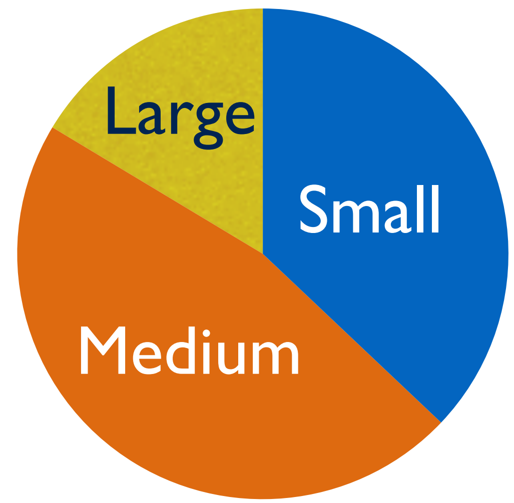


- Select a heterogenous collection of servers that:
  - Minimizes cost
  - Minimizes number and frequency of revocations
  - Minimizes fraction lost due to revocation ➔ Uncorrelated servers

Server Portfolios: Heterogenous Mix of Servers

Server selection is analogous to financial portfolio construction

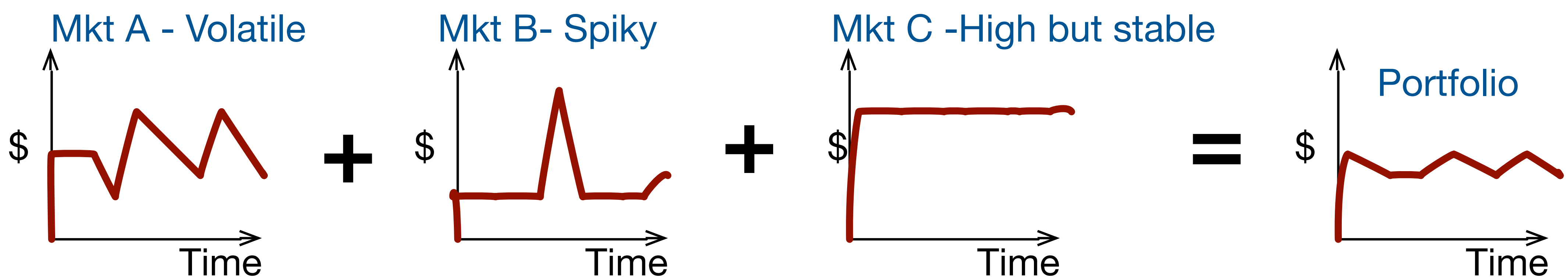




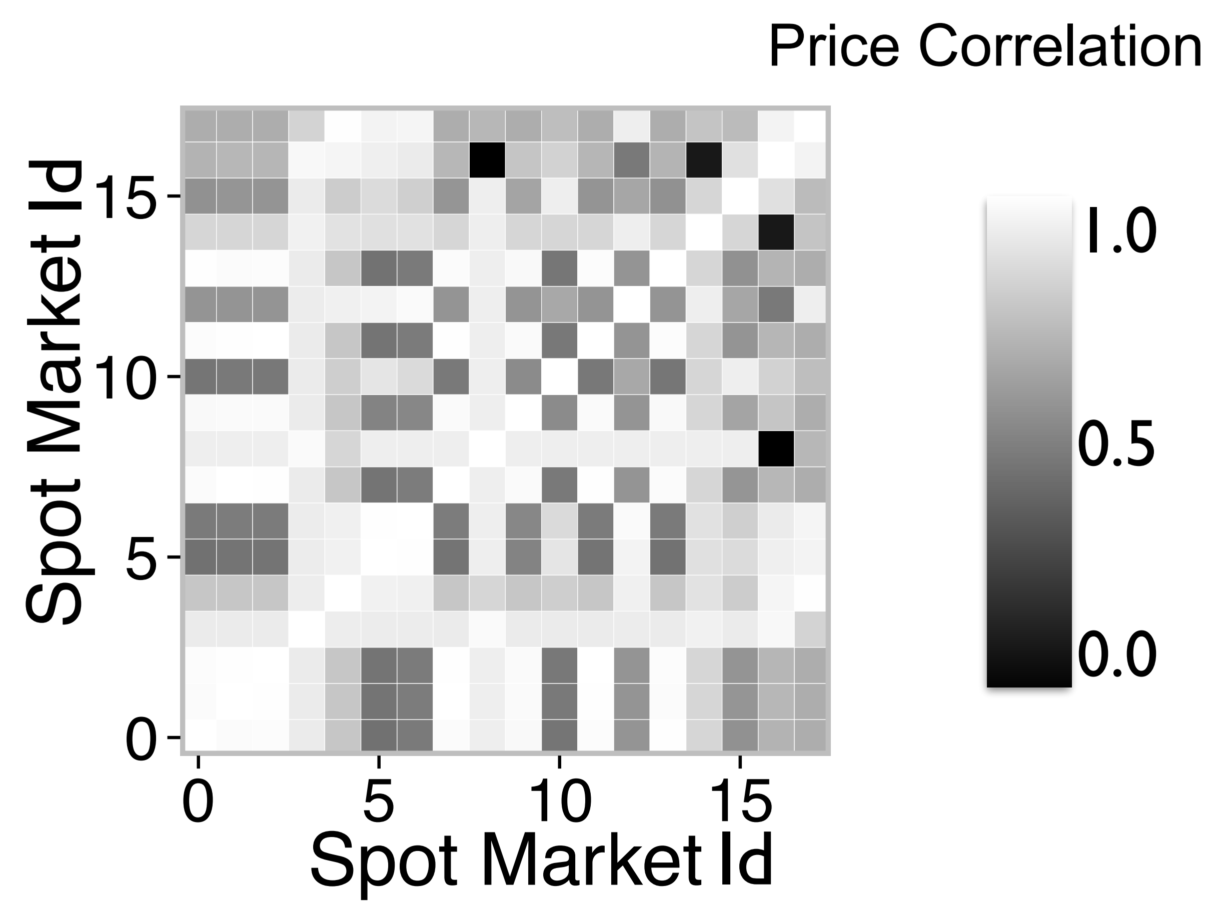
- Stocks, bonds to max returns and min risk → **Server types** to max savings and min revocation risk
- Reduce risk of large losses using uncorrelated assets → Reduce risk of **concurrent revocation** using uncorrelated servers
- Investors have different risk and reward preferences → **Applications** have different risk/reward preferences

Key Idea: Diversification

- Diversification reduces volatility due to individual markets



- Many EC2 markets have low correlations
- Diversification is a viable strategy



Model-driven Portfolio Construction

- **Based on Modern Portfolio Theory from finance (Markowitz 1953)**
- Objective: Maximize risk-adjusted returns

$$E[\text{Return}] - \alpha \cdot \text{Risk}$$

$$1 - \frac{E[\text{Spot-price}]}{\text{On-demand-price}}$$

Risk averseness parameter
$[0, \inf)$

Revocation risk
(variance in prices)

- Example output: portfolio allocation vector, x:

small	med	large
0	0.2	0.8

Portfolio Construction Optimization

$$E[\text{Return}] - \alpha \cdot \text{Risk}$$

$$1 - \frac{E[\text{Spot-price}]}{\text{On-demand-price}}$$

Risk averseness parameter

$[0, \inf)$

Revocation risk

(variance in prices)

x

x

c: Discount vector

x

Spot price traces

V: Covariance matrix

$$\frac{1}{n}\sum_{t=1}^{n}((A(t) - E[A])(B(t) - E[B]))$$

Spot price traces for markets A,B

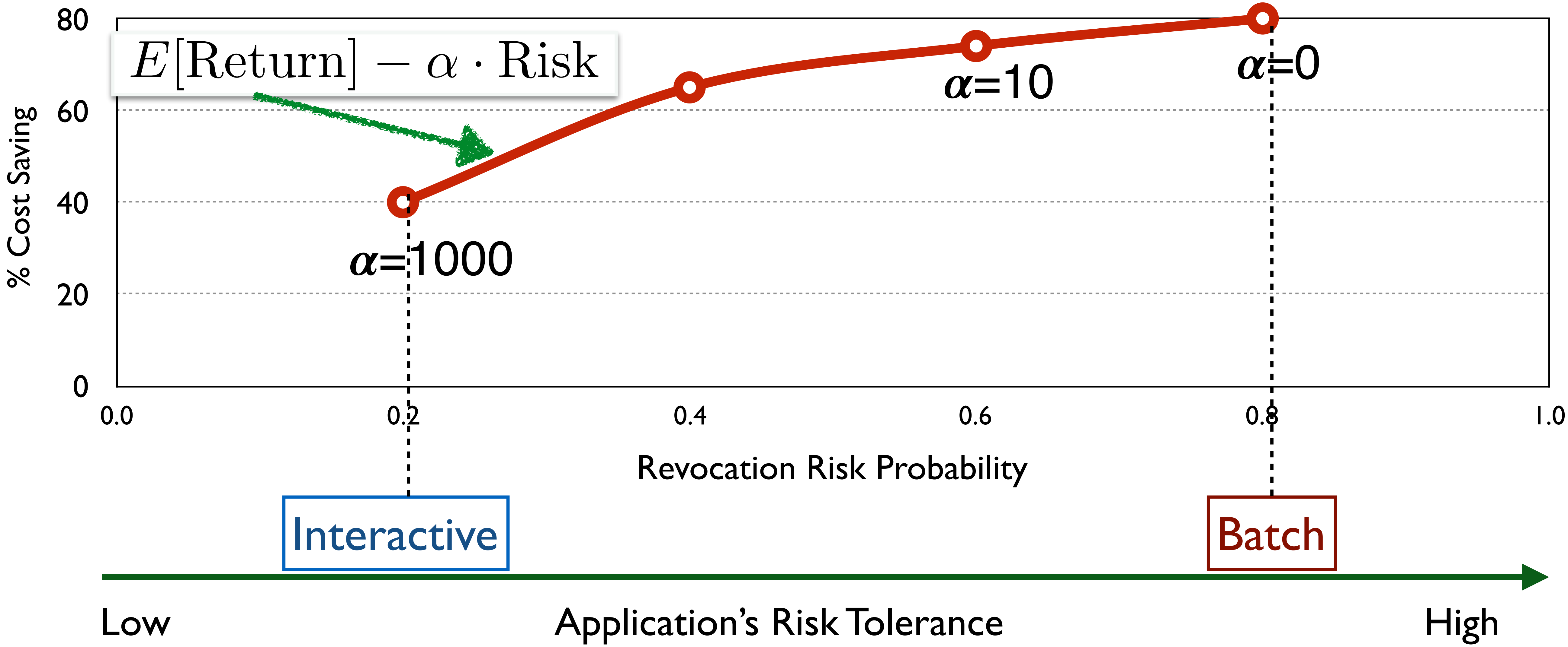
$$\text{Maximize:} \quad \mathbf{c}\mathbf{x}^T - \alpha\mathbf{x}\mathbf{V}\mathbf{x}^T$$

$$\text{Subject to:} \quad \sum_{1}^{n} x_i = 1$$

$$\mathbf{x} \geq 0$$

Convex Quadratic

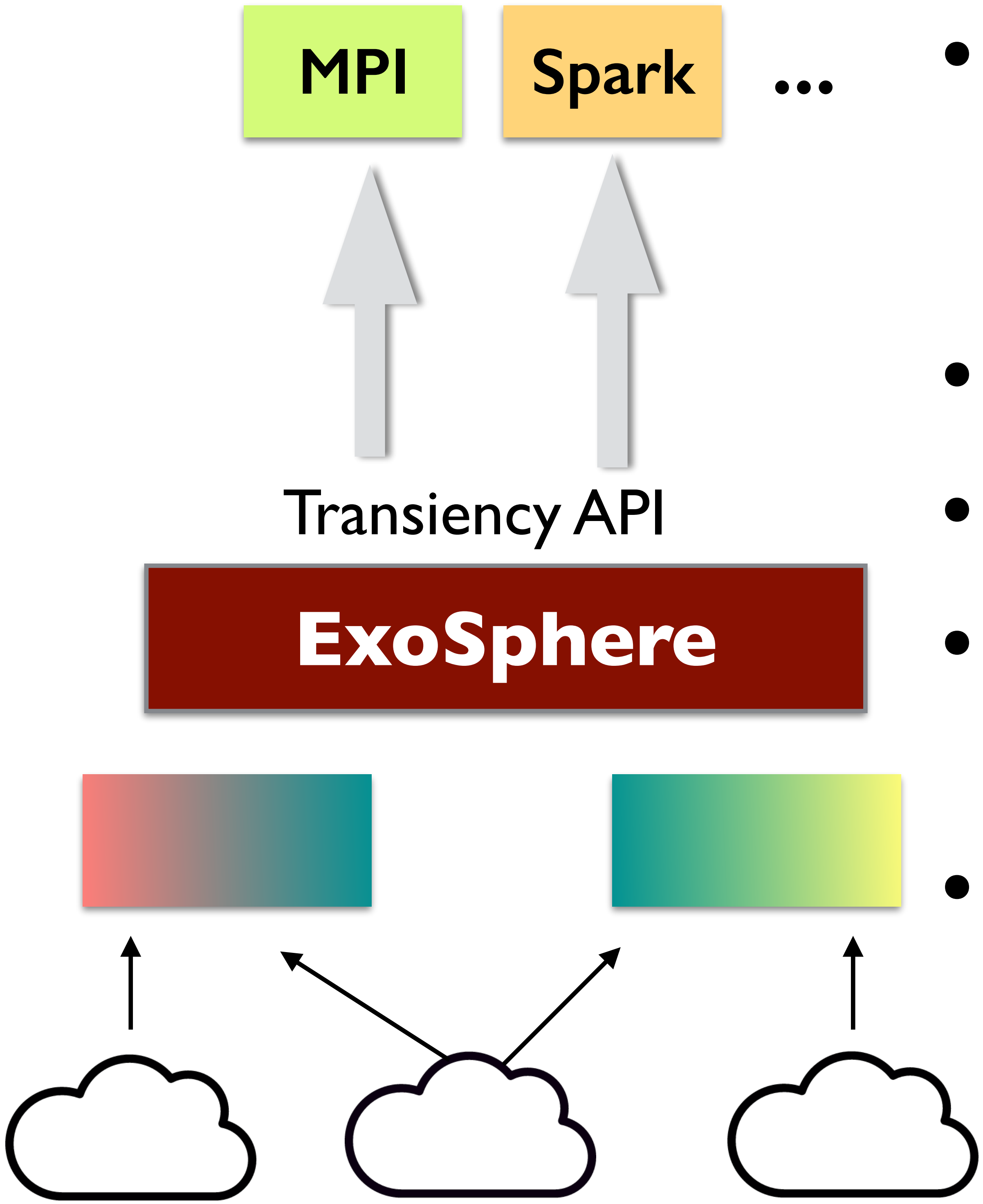
Risk-Return Tradeoffs With Portfolios



- Failures/delays are undesirable
- High diversification
  - Lower savings, low risk

- Failures/delays are tolerable
- Low diversification
  - High savings, higher risk

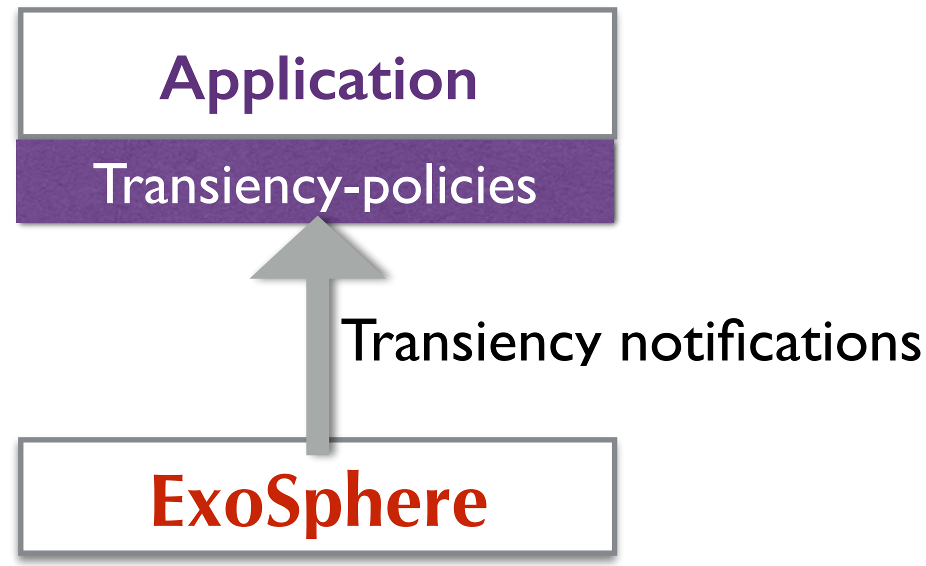
ExoSphere: Transiency-aware Cluster Management



- ExoSphere provides virtual clusters to run multiple applications
  - Spark, MPI, BOINC
- Transiency-aware Mesos (4K lines)
- Applications submit: (#CPUs, Mem, α)
- Applications can share cloud servers
  - Multiplexing → Reduced costs
- Applications get price, MTTF, revocation-warning notifications

Transiency Mitigation Policies With ExoSphere

- ExoSphere enables applications to implement custom policies
- Especially useful for fault-tolerance
  - Checkpoint application state and roll-back in case of revocations
  - Use existing *mechanisms* to implement policy in few lines of code



Young-Daly periodic checkpoint interval = $\sqrt{2 \times \text{Time to Checkpoint} \times \text{MTTF}}$

Checkpoint size, disk-speed   Provided by ExoSphere

When To Checkpoint?

- **Key idea: checkpoint periodically to minimize expected running time**
- Simplified Spark performance model on transient servers

$$E[T] = T + \frac{T}{\tau} \cdot \delta + \frac{T}{MTTR}\left(\frac{\tau}{2}\right)$$

Checkpointing overhead

Recovery

τ: Checkpointing interval

δ: Time required to save RDD

MTTR: Mean Time To Revocation

- **Minimize E[T] with respect to τ:**

$$\text{RDD checkpoint interval} \quad (\tau) = \sqrt{2 \cdot \delta \cdot MTTR}$$

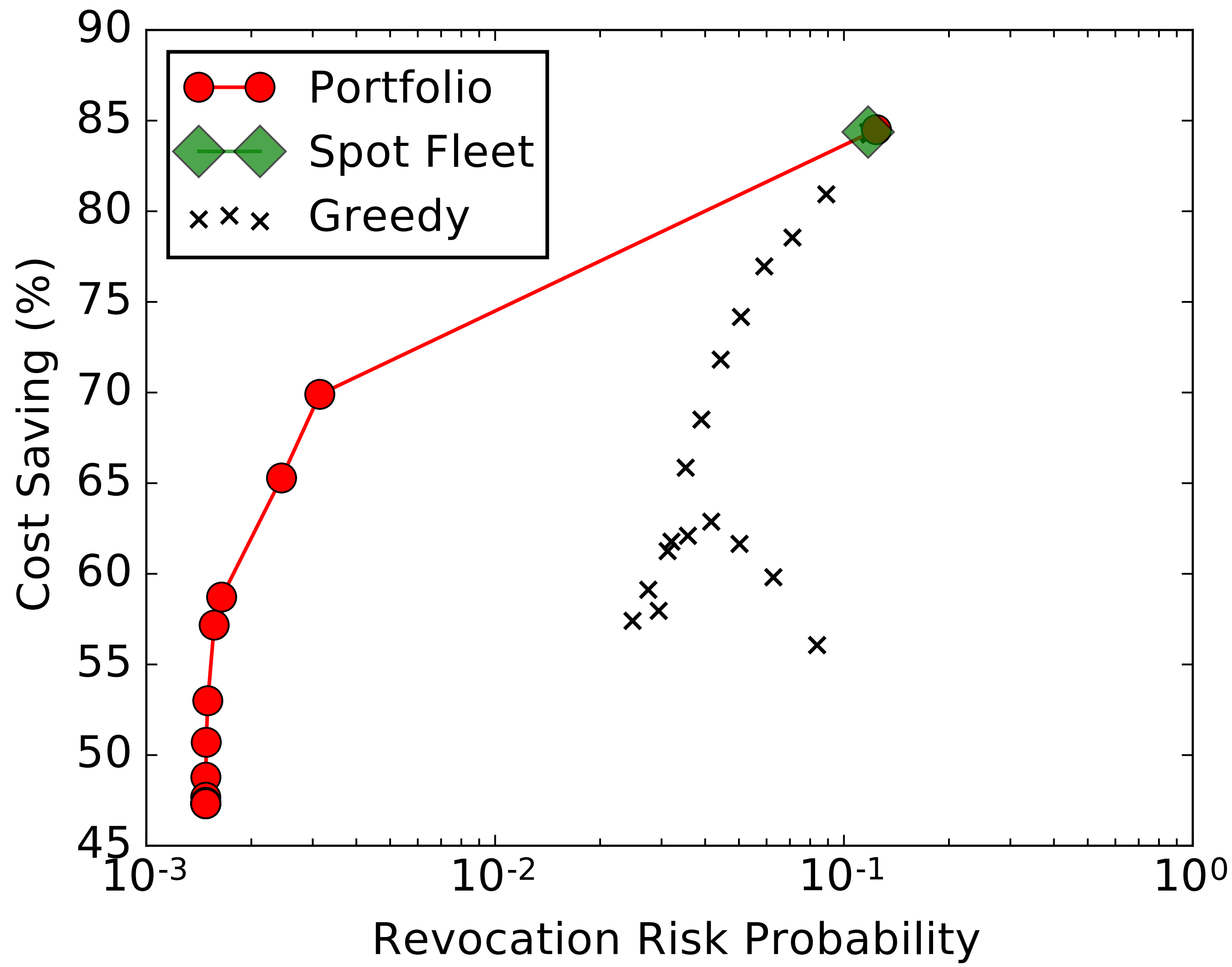
RDD size, write speed

Spot market price traces

Young-Daly 1974

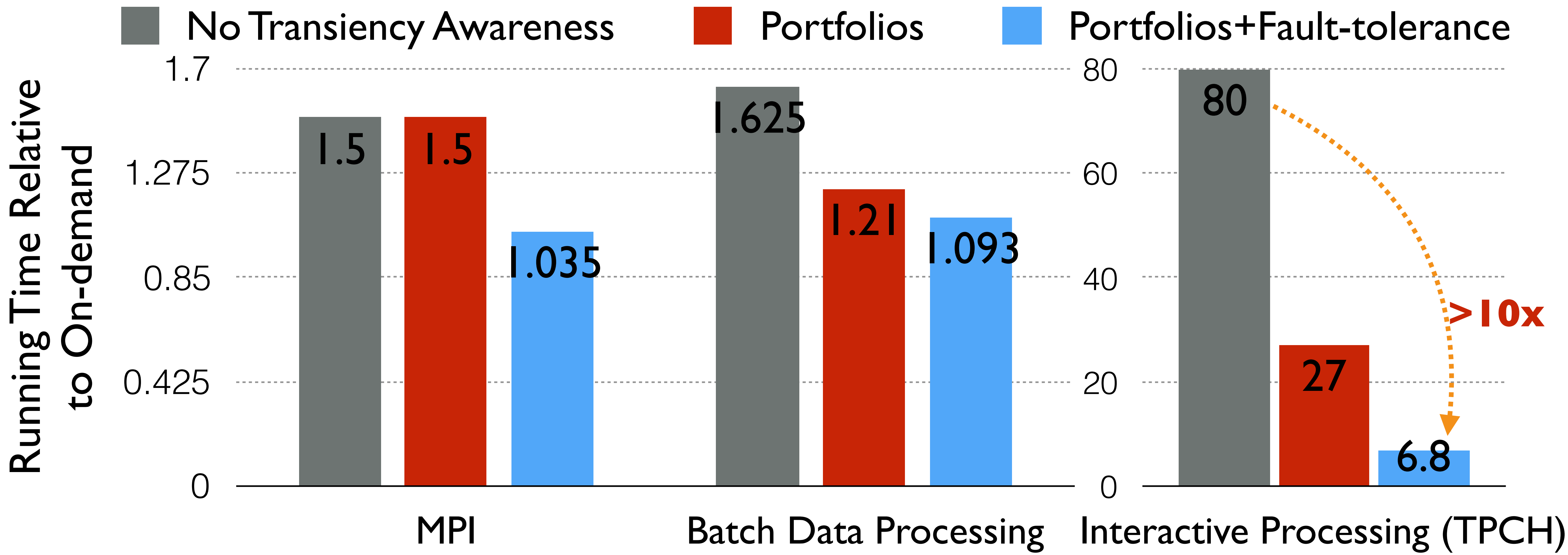
Effectiveness of Portfolios

Cost-risk comparison based on EC2 spot prices from March-Nov 2015



- 85% cost savings compared to on-demand
- ~100x reduction in revocation risk compared to existing approaches

Application Performance In ExoSphere



- Portfolios+checkpointing: reduces transiency overhead to < 10%
- Risk intolerant interactive applications see significant performance benefit

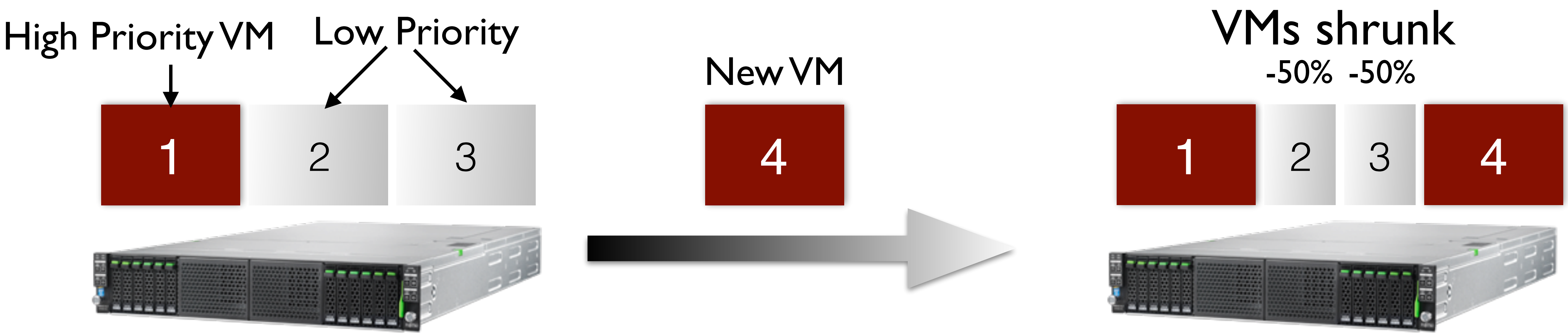
ExoSphere Summary

- Transient server selection based on portfolio modeling
- ExoSphere: system for portfolio based cluster management
- General framework supporting common transiency policies
- Cloud transient servers are an increasingly popular area:

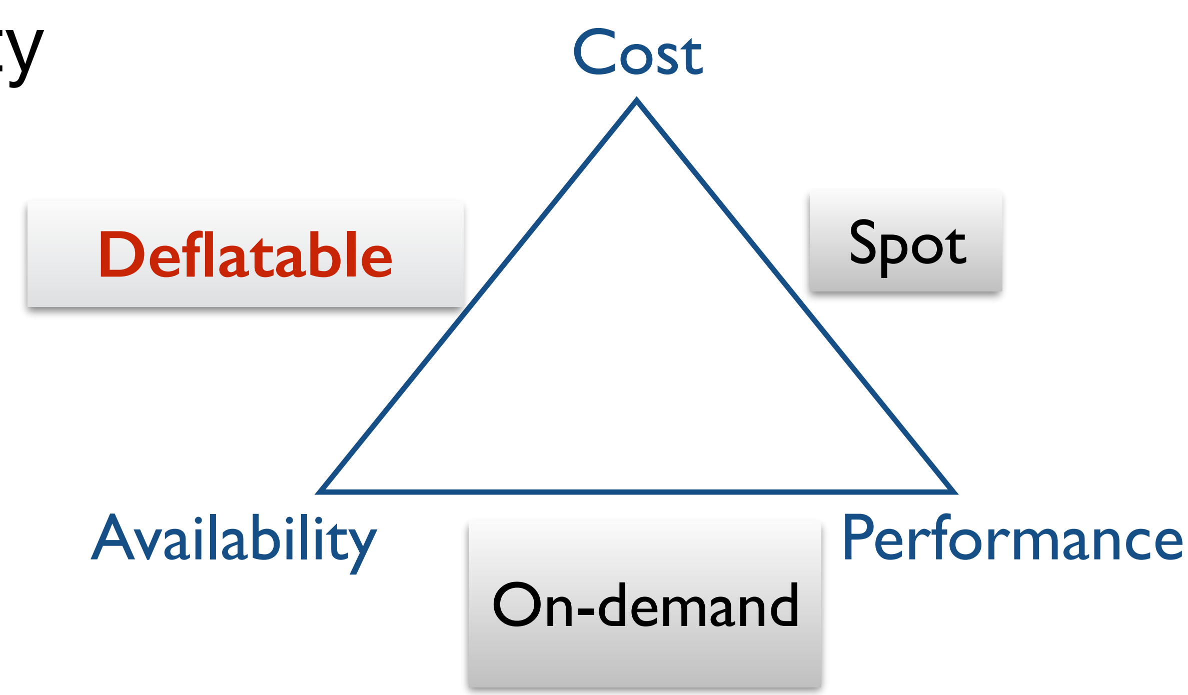
Homogenous server selection	OptiSpot, SpotOn [SoCC '15],…
Heterogenous server selection	Amazon SpotFleets, Tributary [ATC'18]
Application-specific techniques	Spark [TR-Spark-SoCC '16, HPDC'17], MPI [HPDC '14], ML [Proteus-EuroSys'17]

Reclaiming Resources Using Resource Deflation

- How to reclaim resources from low-priority VMs?
  - **Resource Deflation: Fractional resource reclamation.**



- For most applications: higher availability > performance degradation
- Trade-off higher availability for performance degradation

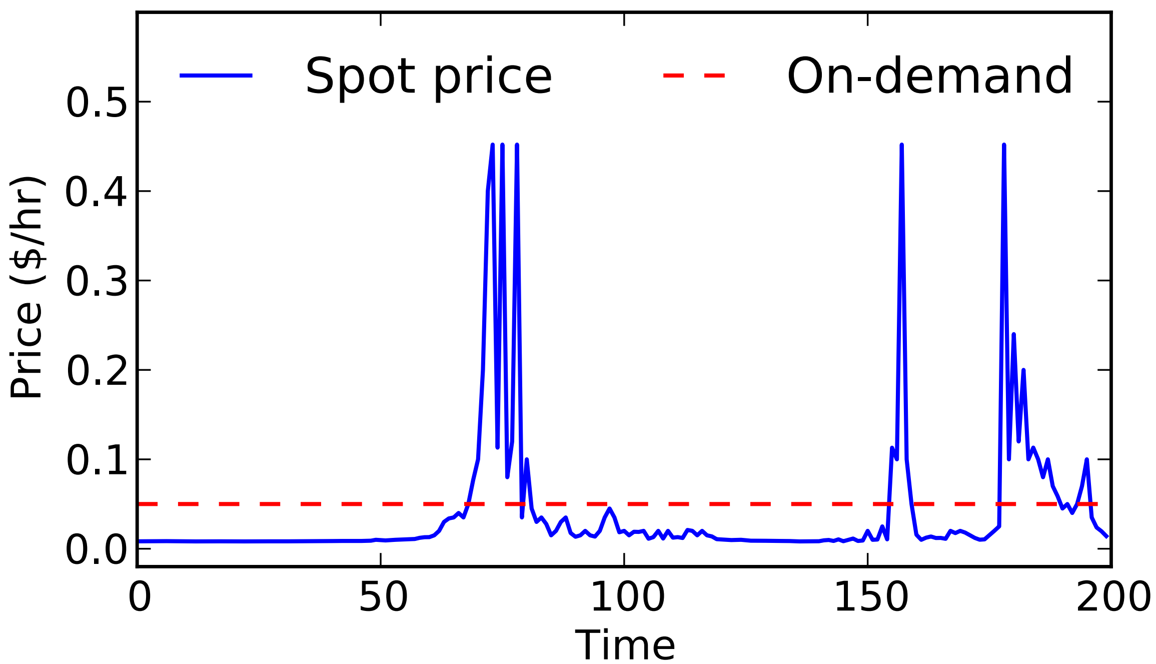


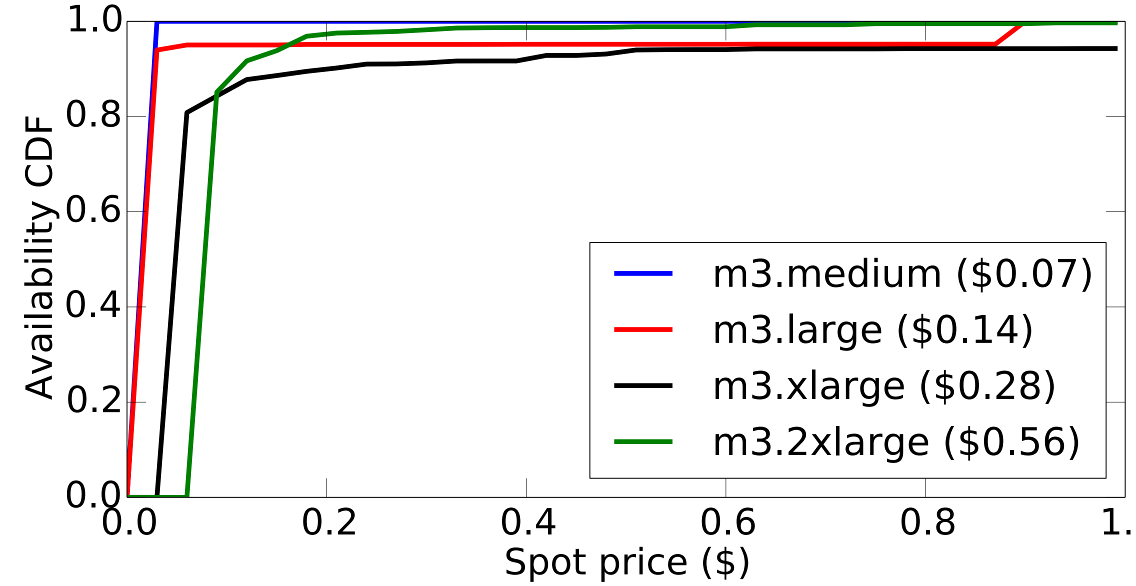
Thanks!

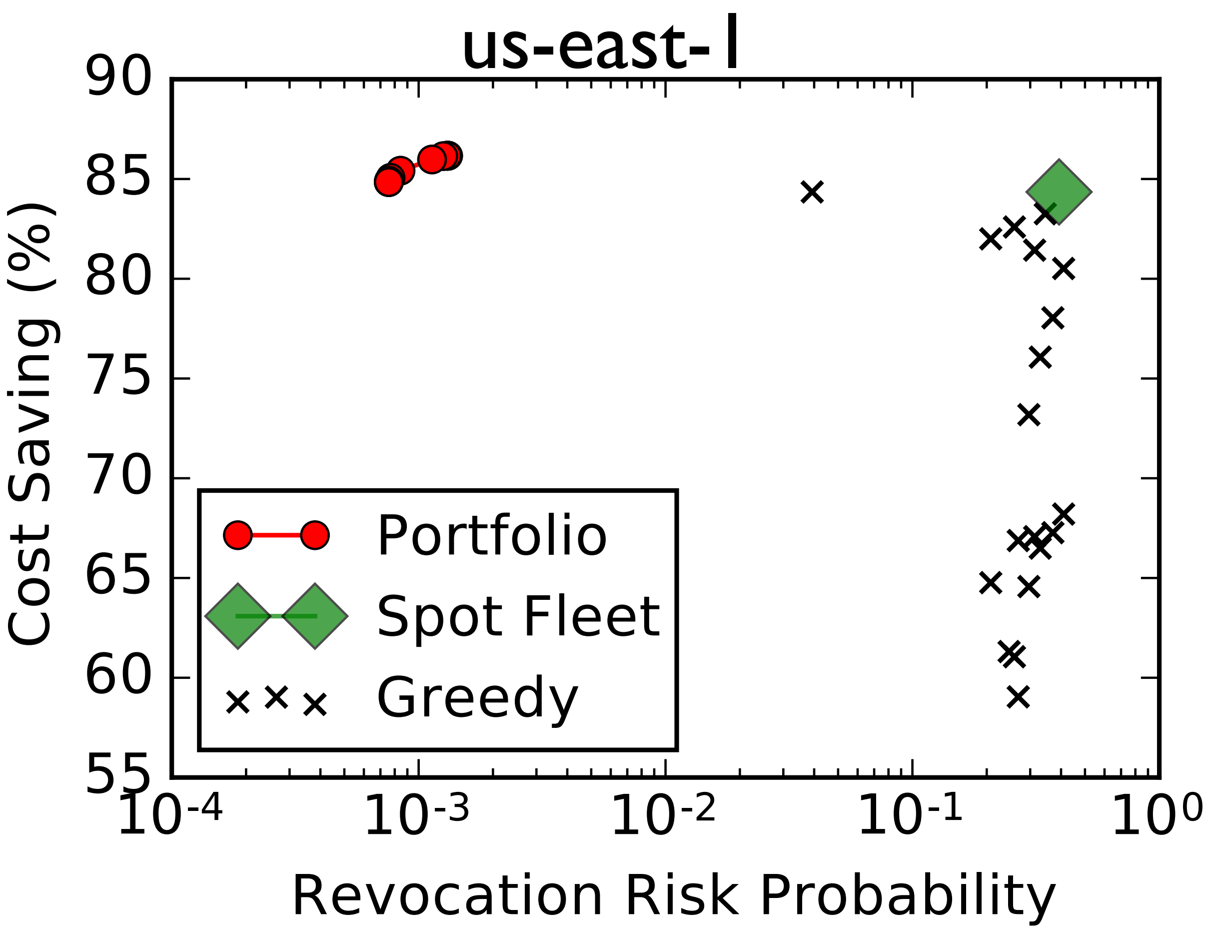
Questions?

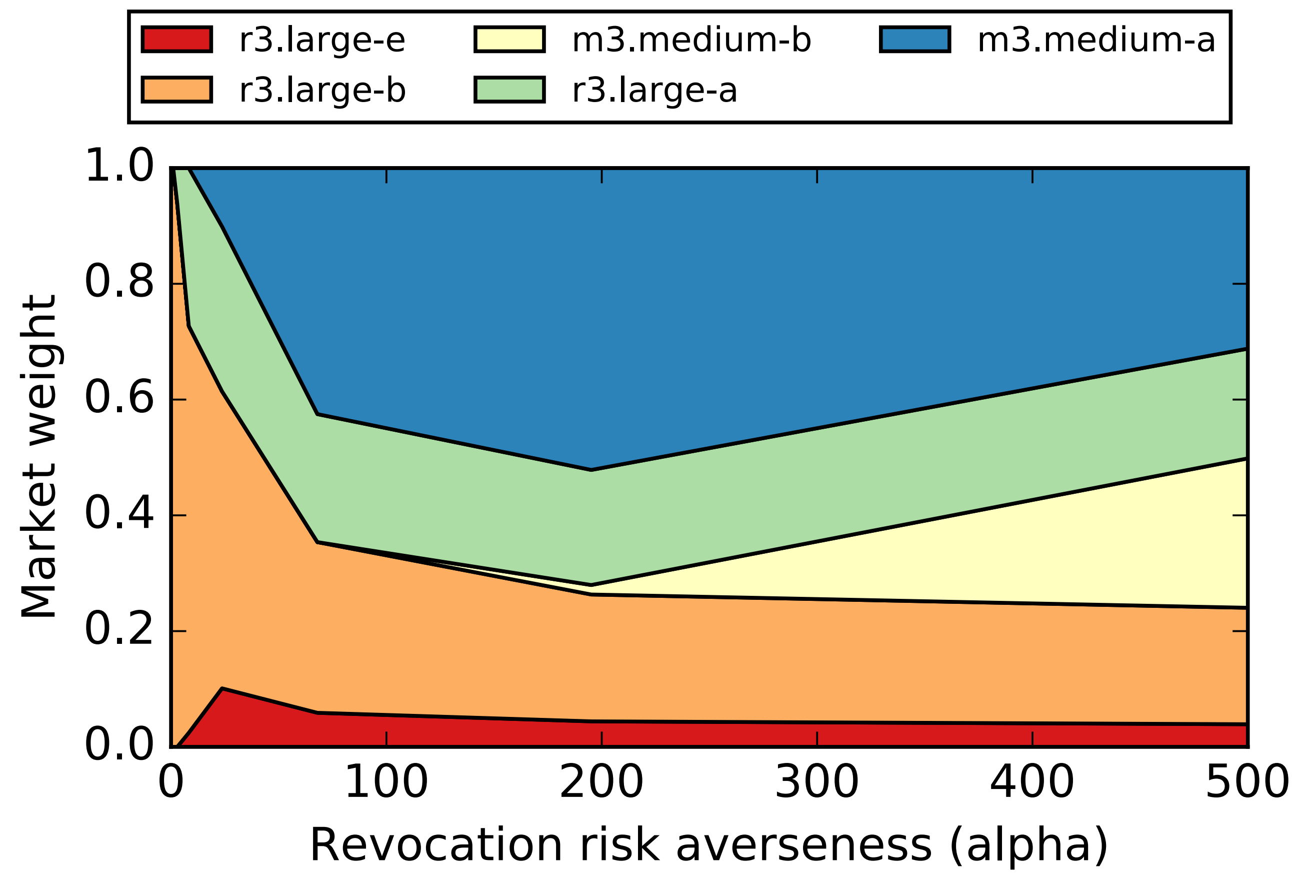
- **Transient Servers**
- SpotCheck
- Flint
- **ExoSphere**
- **Resource Deflation**

ExoSphere Backup

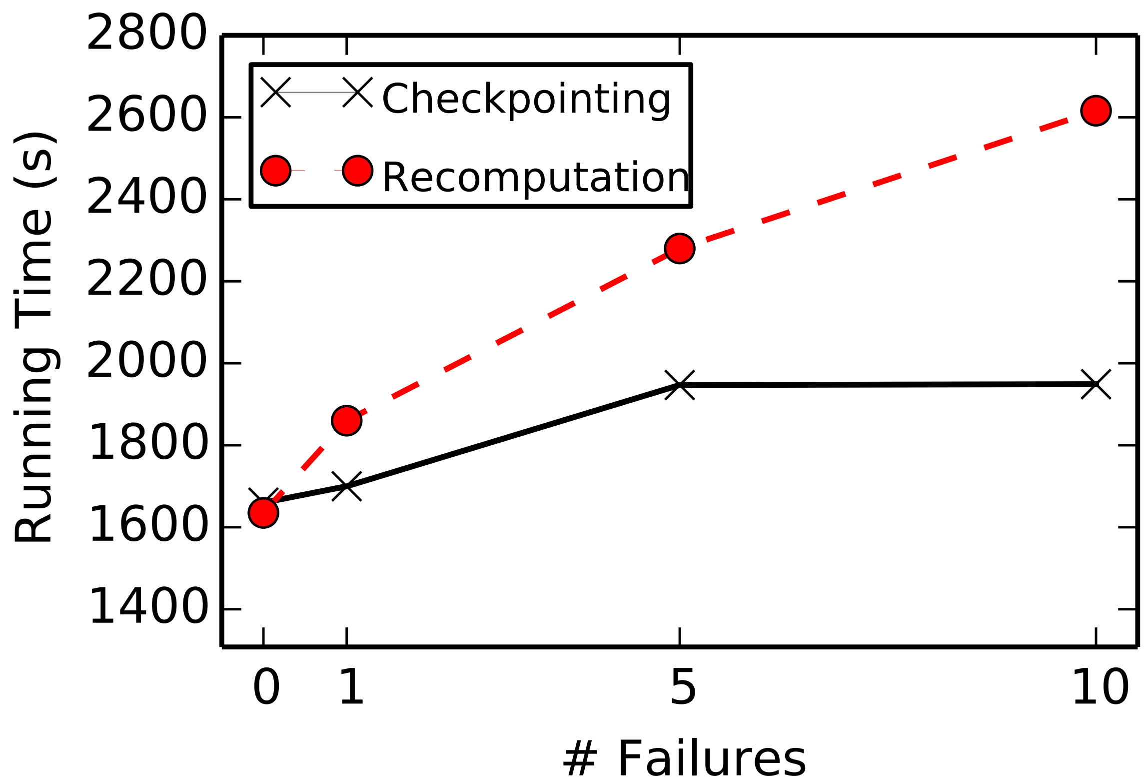


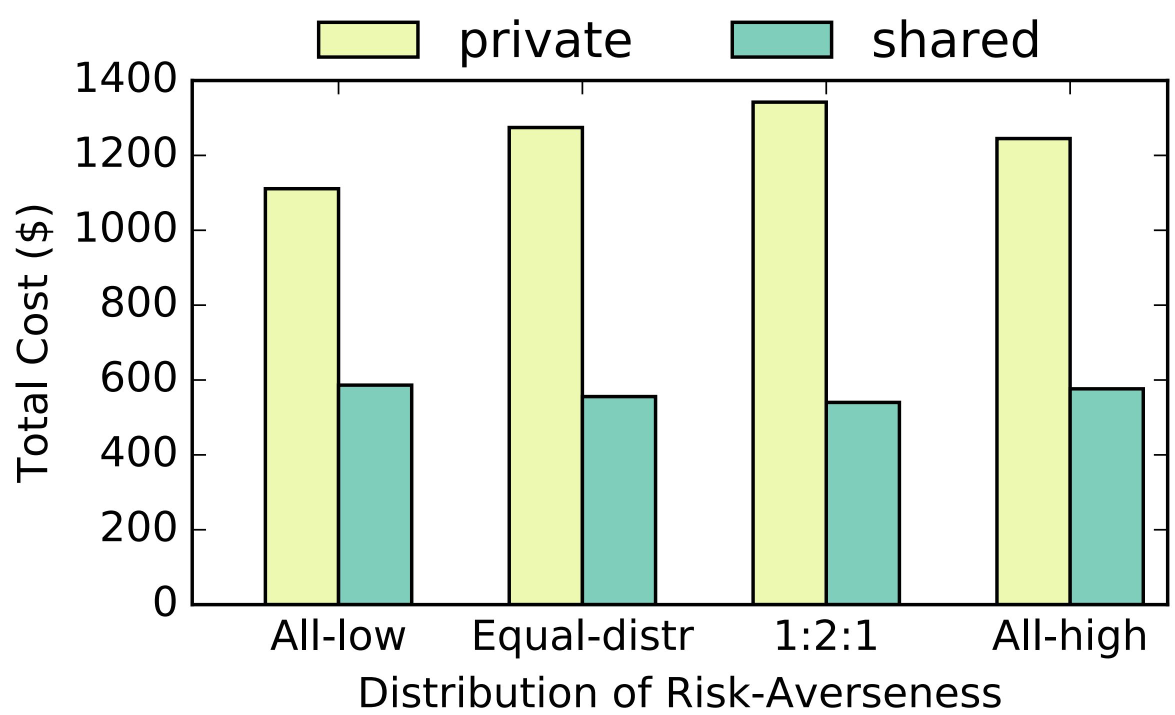


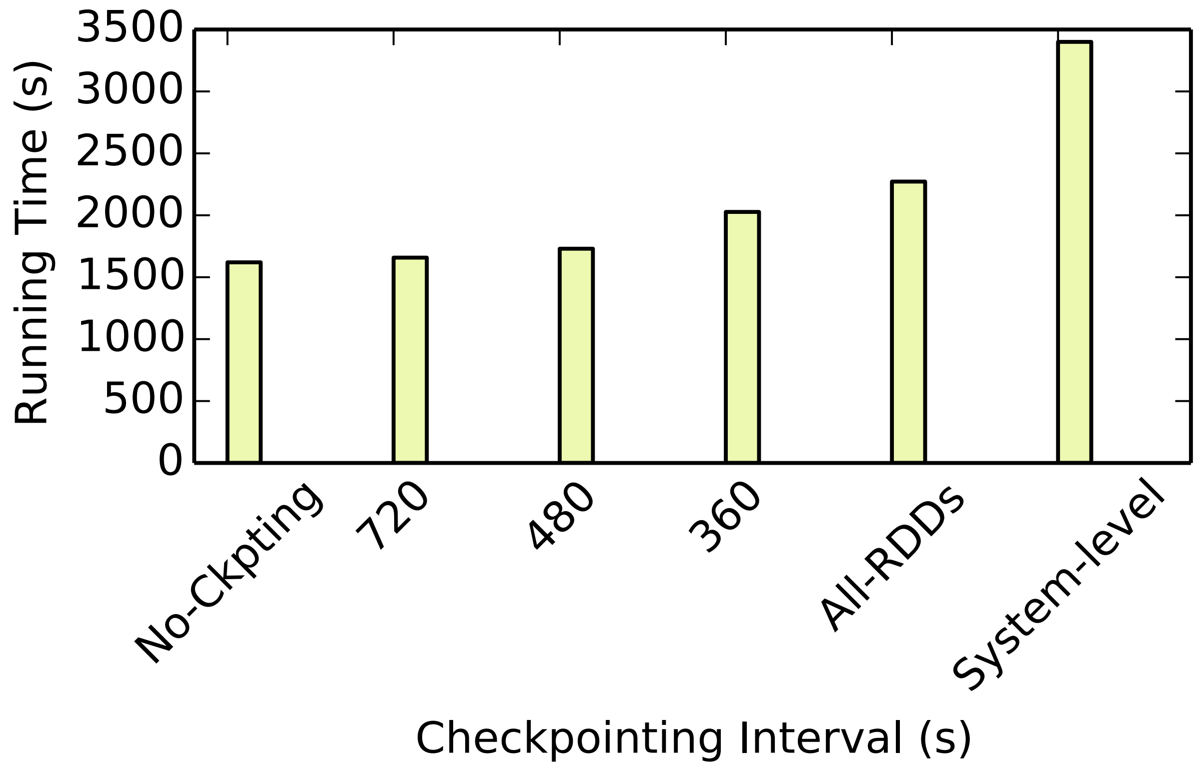




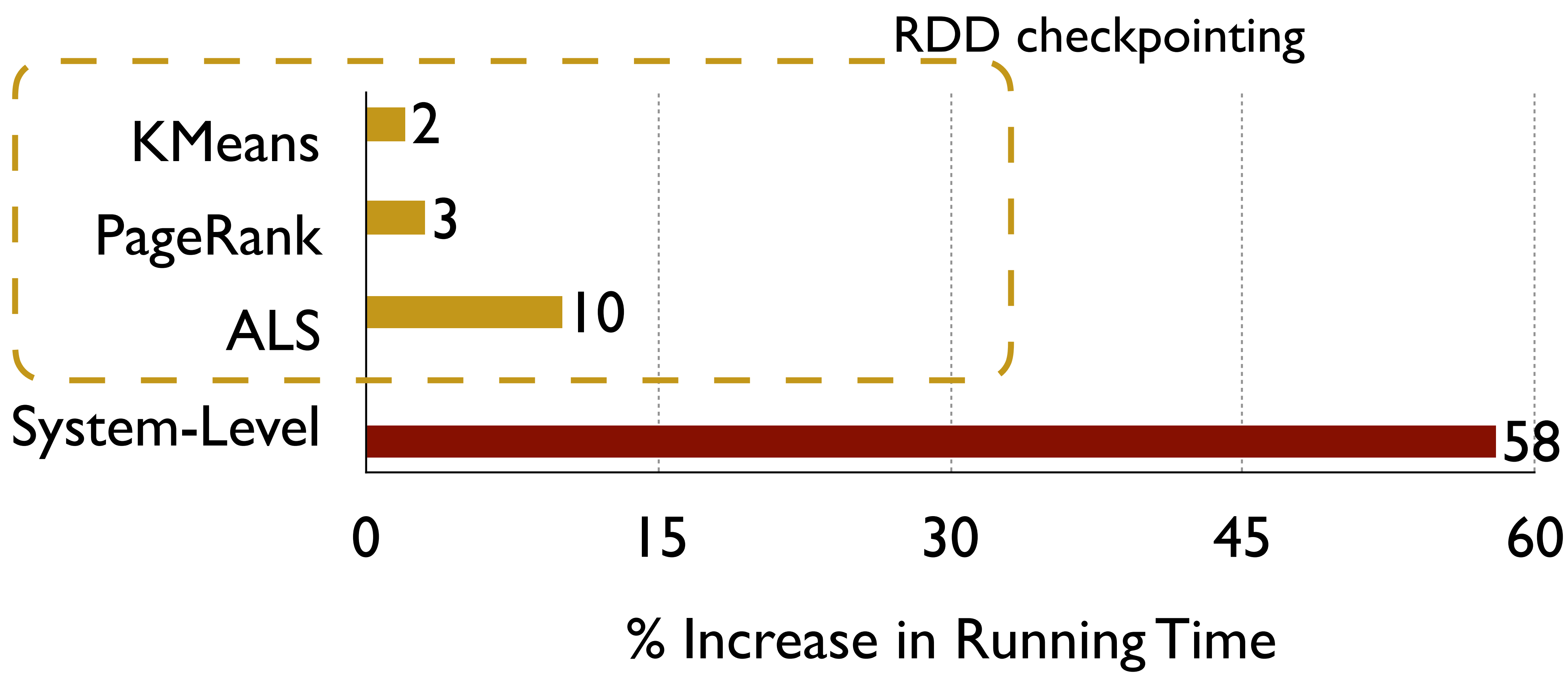
ExoSphere + Flint





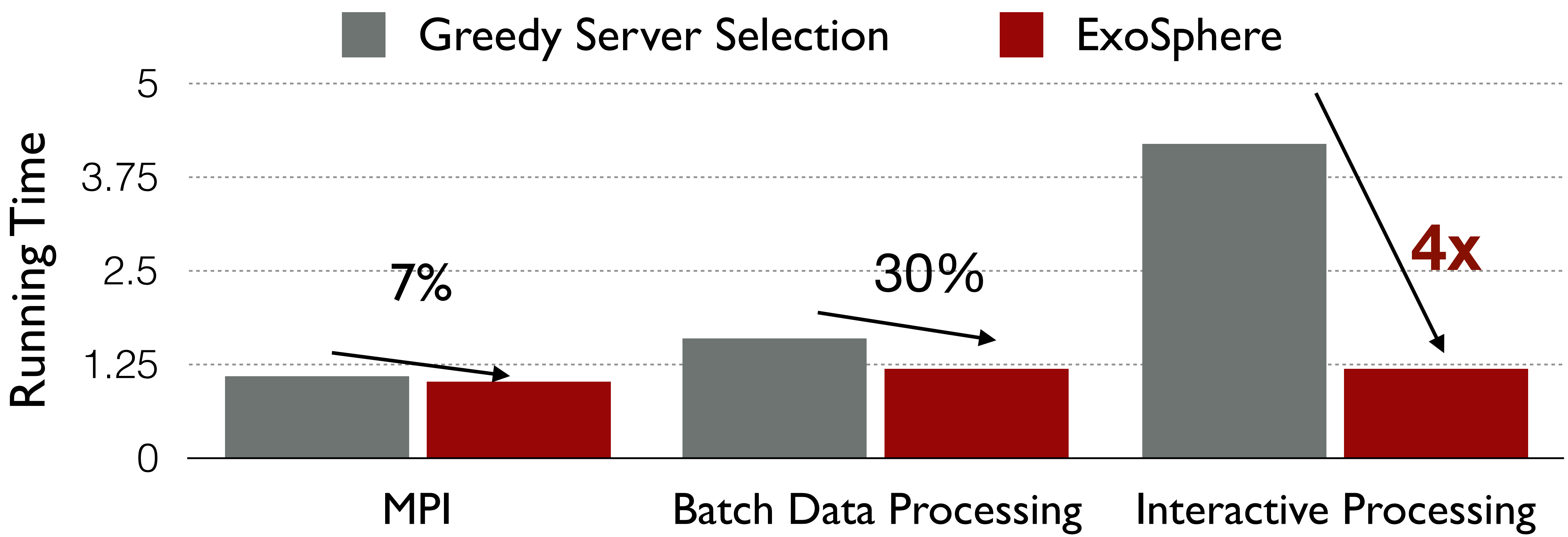


Checkpointing Overhead



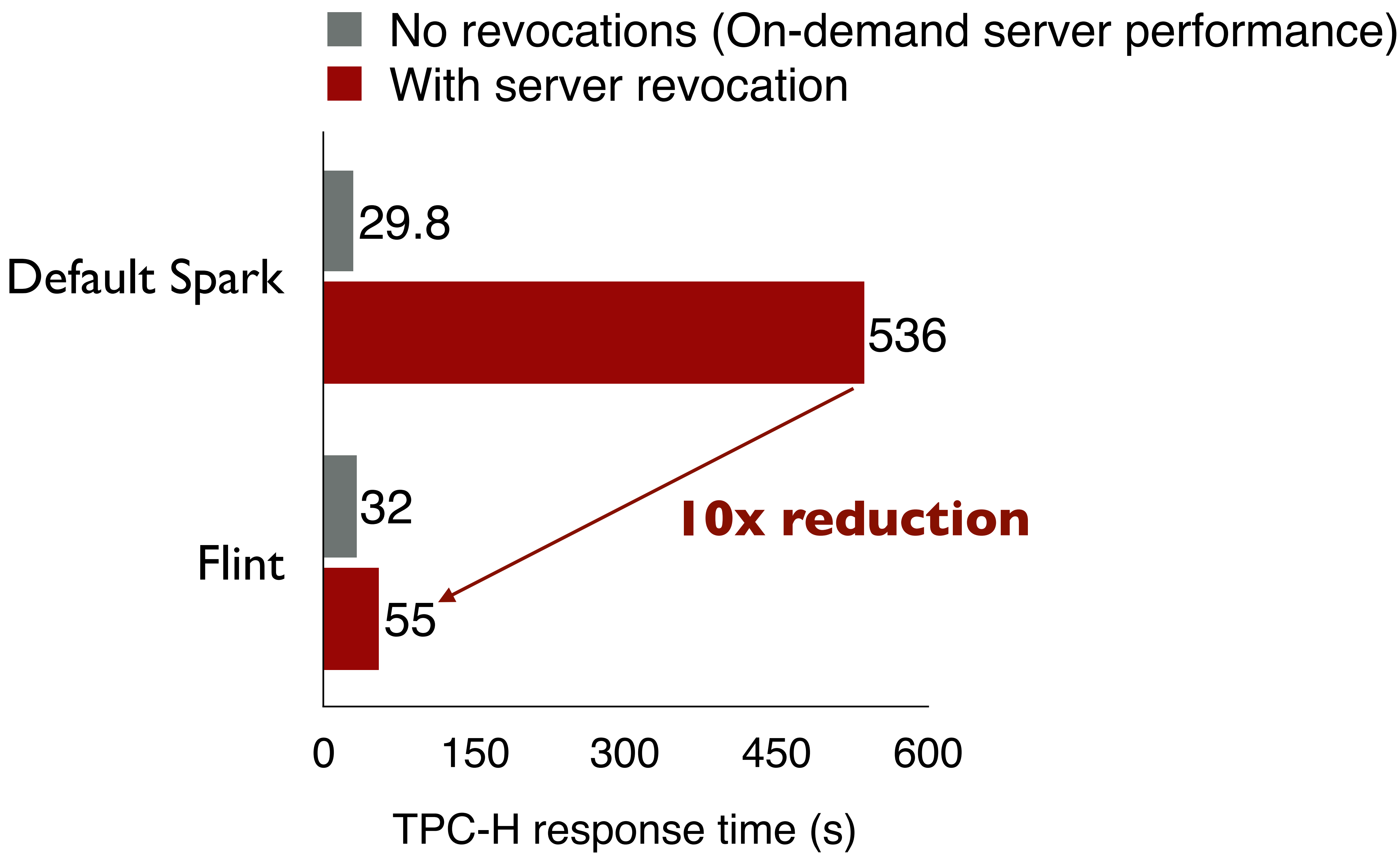
- RDD checkpointing: <10% performance overhead
- System-level checkpointing: high overhead of writing OS and cache

Application Performance



- Greedy selection only considers cost ➔ high revocation risk
- Risk intolerant applications see significant performance benefit

Interactive Data Processing Performance



Interactivity maintained even after revocation