TraceUpscaler :

Upscaling Traces to Evaluate Systems at High Load

Sultan Mahmud Sajal,

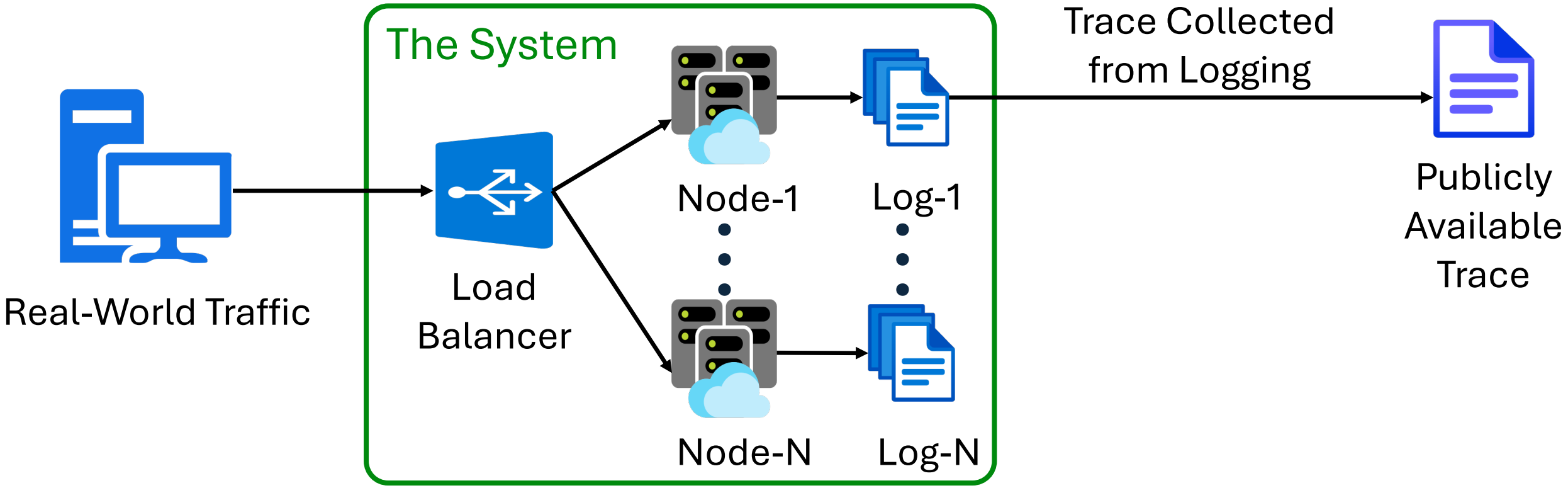
Timothy Zhu, Bhuvan Urgaonkar, Siddhartha Sen

PennState

Microsoft Research

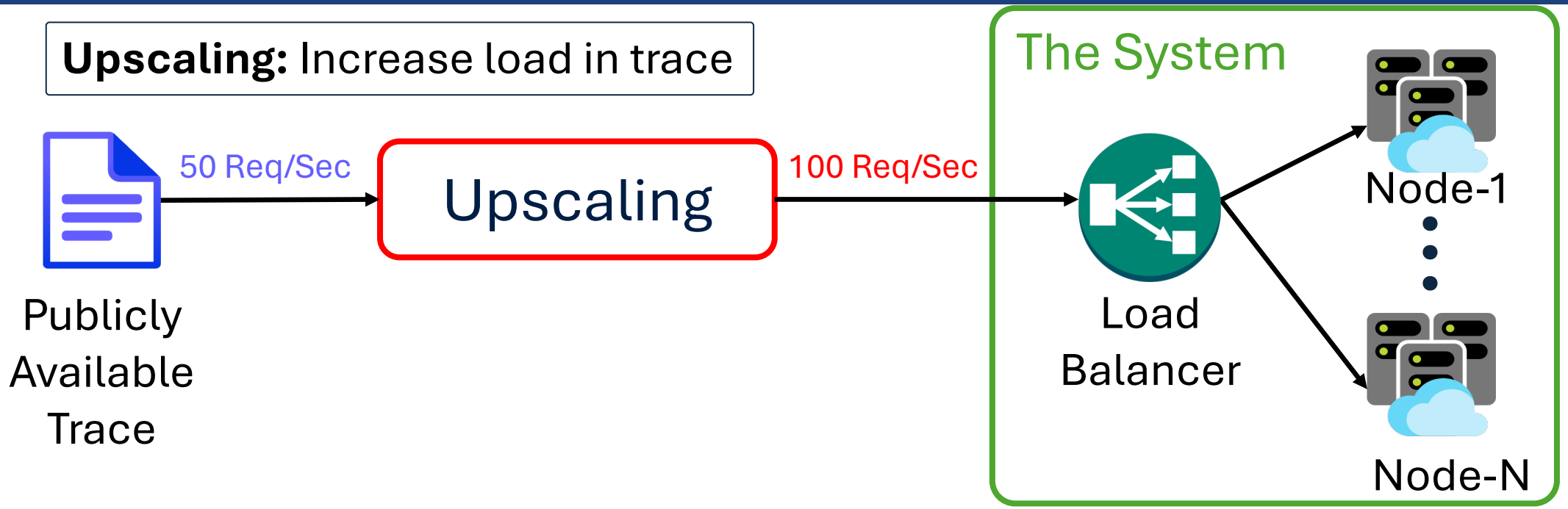
<EURO/SYS'24>

The Need for Upscaling



Load Too Low → Upscaling

Upscaling: Increase load in trace

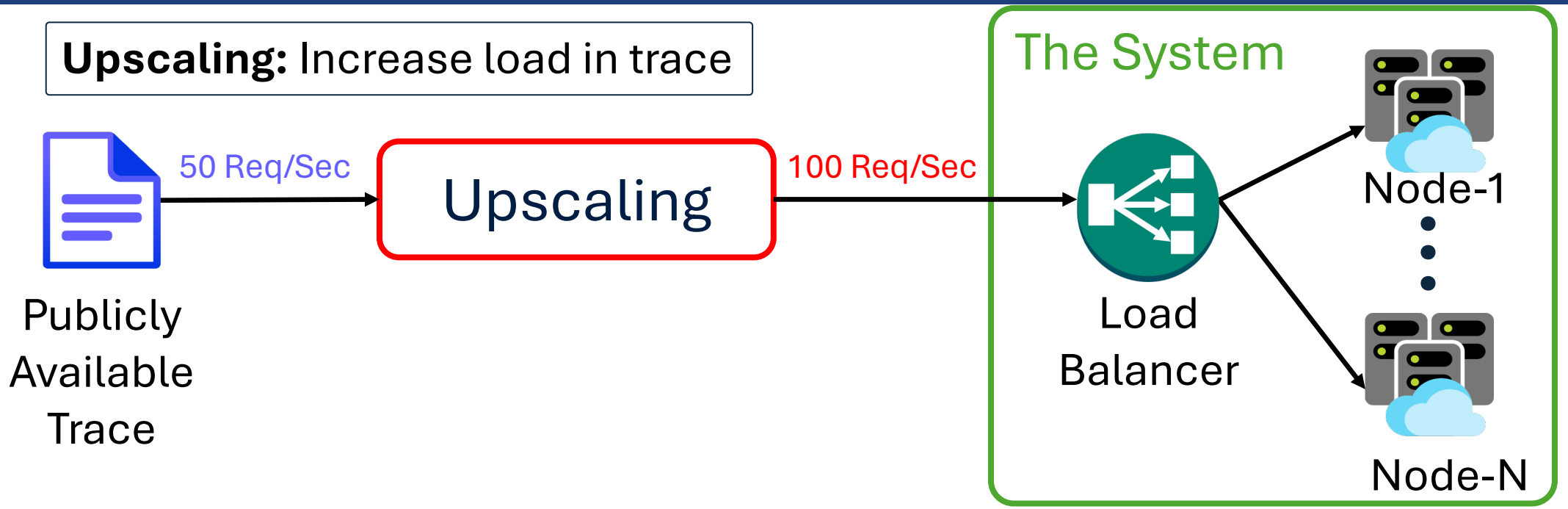


Use Case:

1. Test hypothetical increased load scenarios

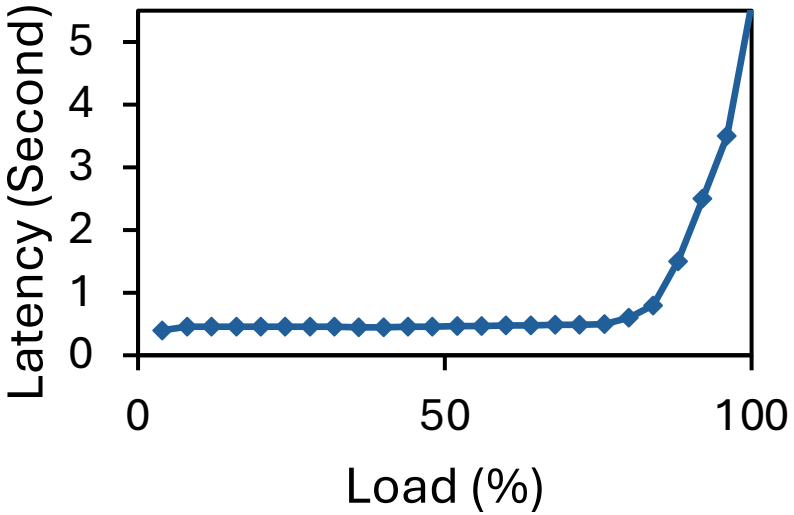
Load Too Low → Upscaling

Upscaling: Increase load in trace



Use Case:

1. Test hypothetical increased load scenarios
2. Characterize systems with Load vs Latency data



Load Too Low → Upscaling

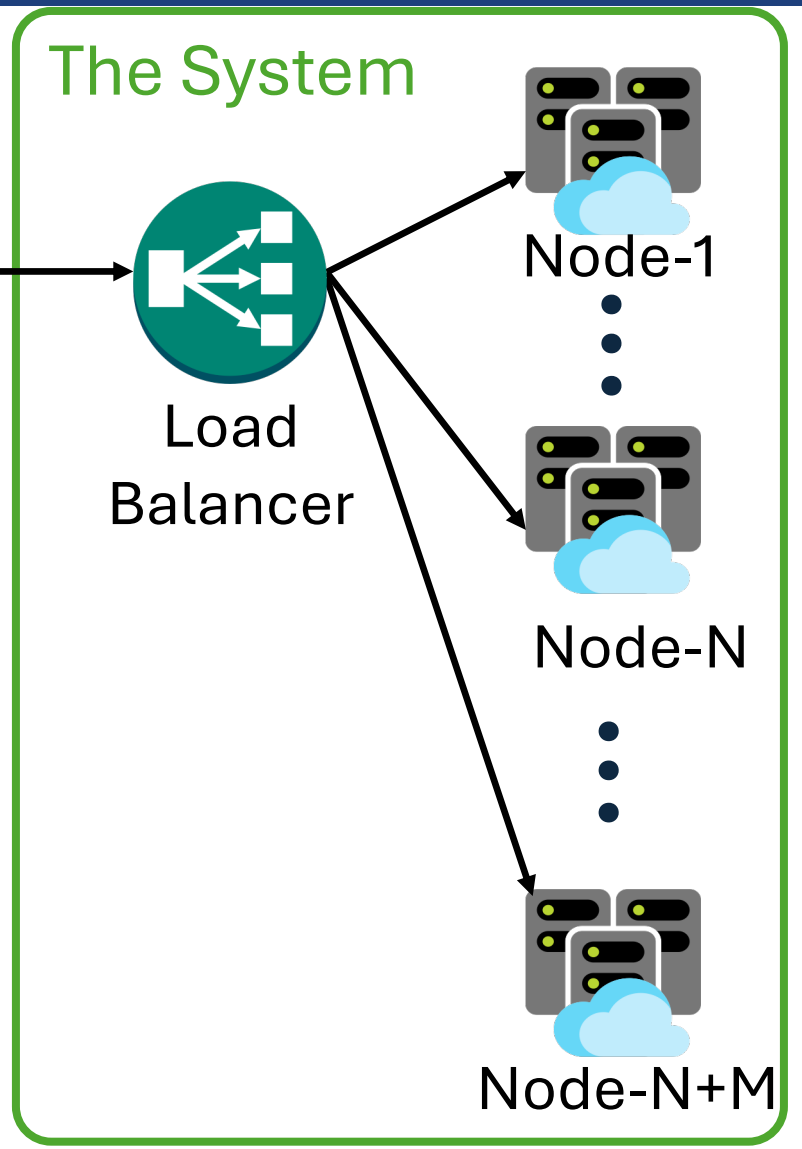
Upscaling: Increase load in trace

Publicly Available Trace

50 Req/Sec

Upscaling

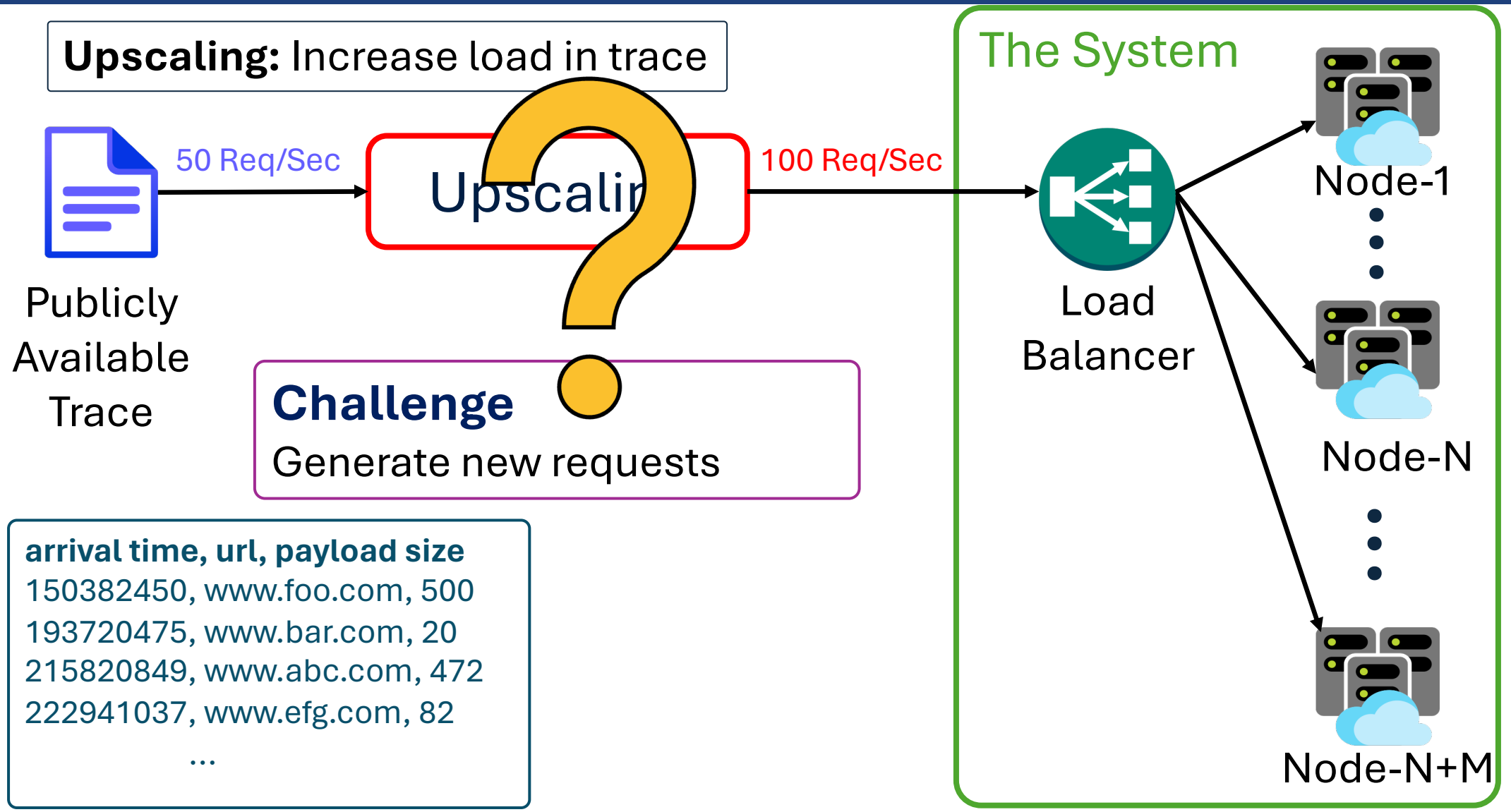
100 Req/Sec



Use Case:

1. Test hypothetical increased load scenarios
2. Characterize systems with Load vs Latency data
3. Evaluate larger systems

Load Too Low → Upscaling

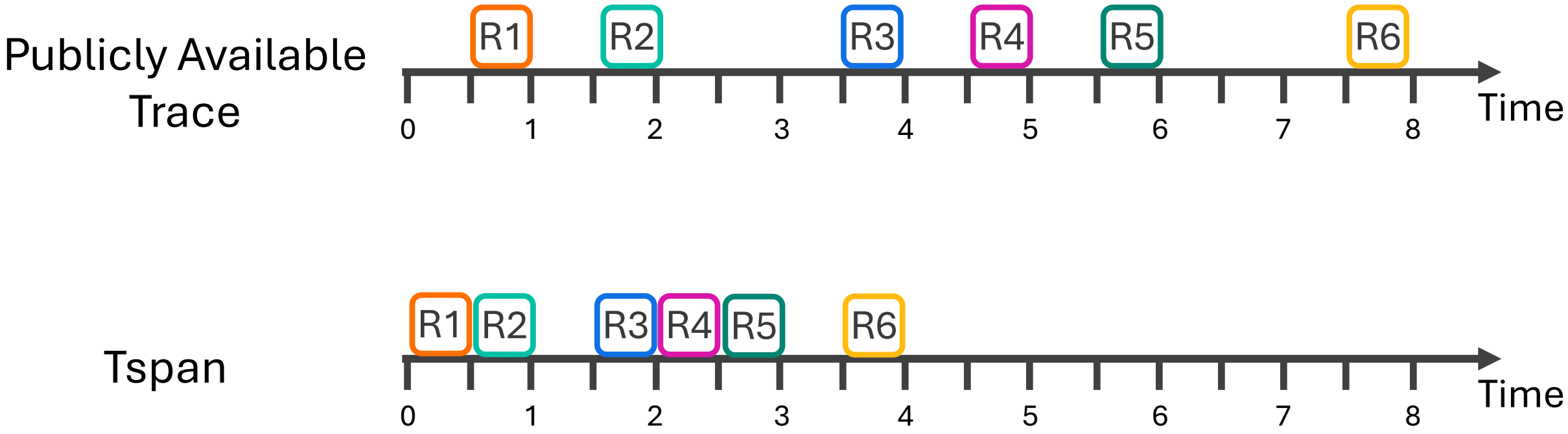


Timespan Scaling (Tspan) [FAST '14, SoCC '20, ATC '21]

Key Idea:

Divide timestamps by upscaling factor

Upscaling Factor, $f = 2$

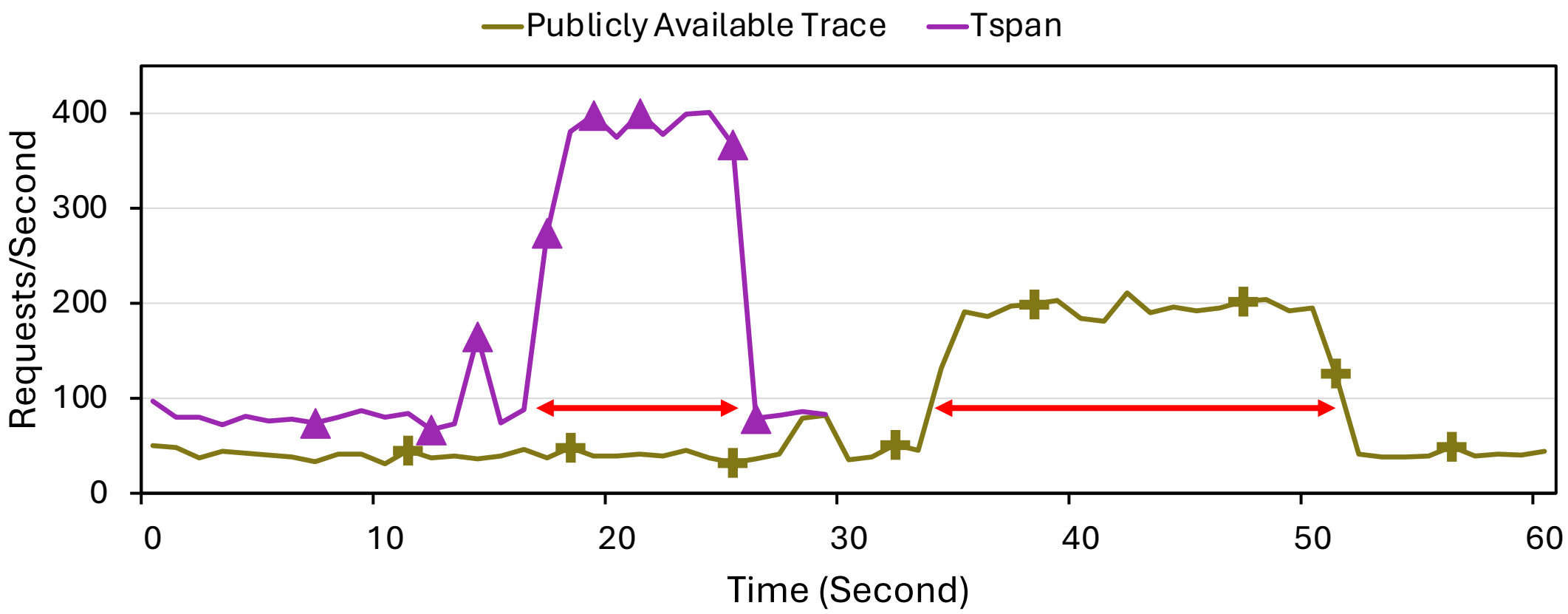


Timespan Scaling (Tspan) [FAST '14, SoCC '20, ATC '21]

Key Idea:

Divide timestamps by upscaling factor

Upscaling Factor, $f = 2$



Tspan distorts temporal pattern and overload characteristics

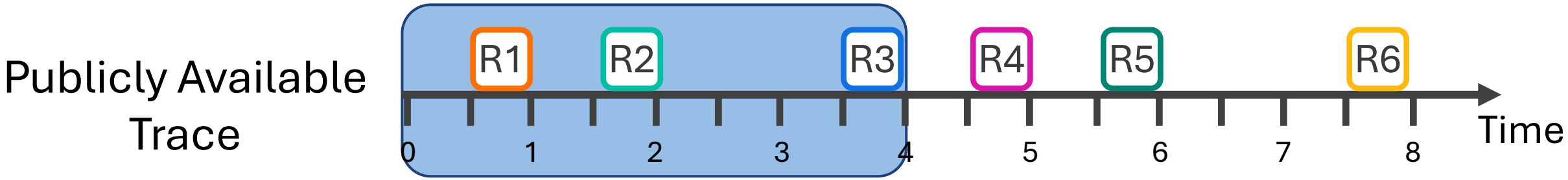
AverageRateScaling [WWW '22, SoCC '21, SoCC '18]

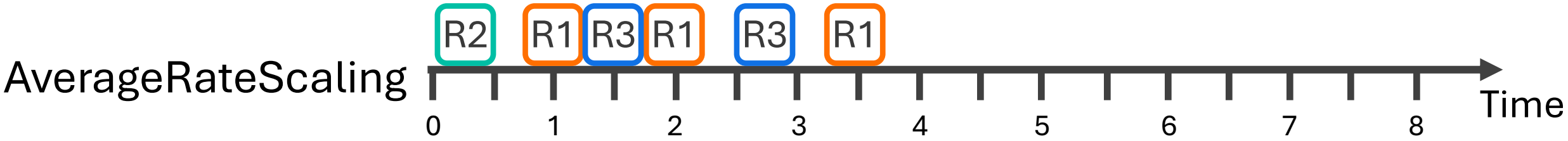
Key Idea:

- Select a time interval
- Upscale number of arrivals in the interval
- Generate random timestamps in the interval
- Sample from requests in the interval

Upscaling Factor, $f = 2$

Time Interval = 4





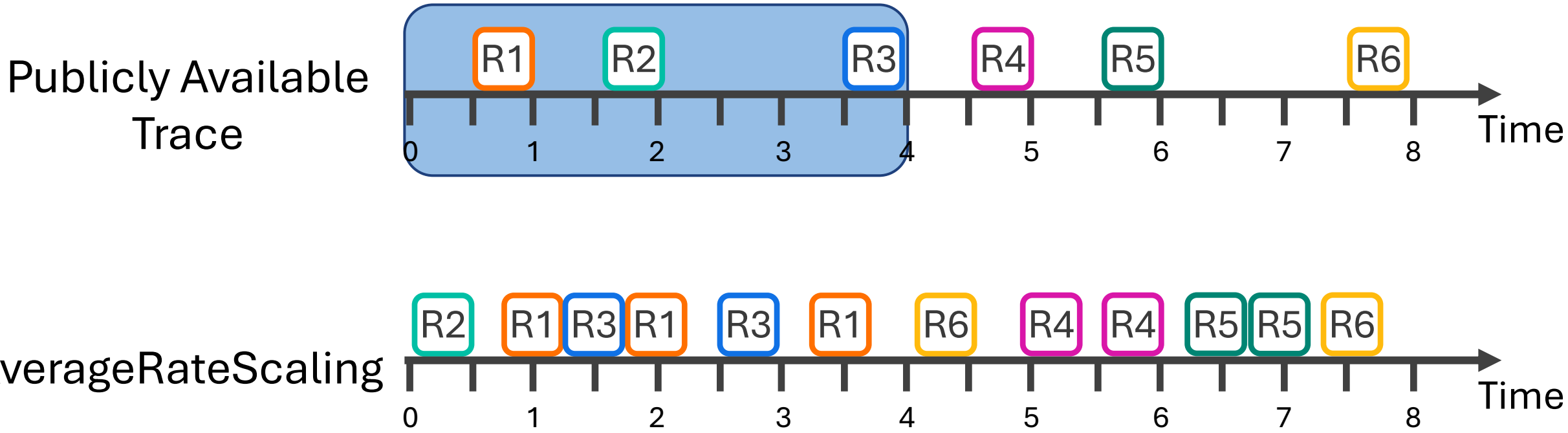
AverageRateScaling [WWW '22, SoCC '21, SoCC '18]

Key Idea:

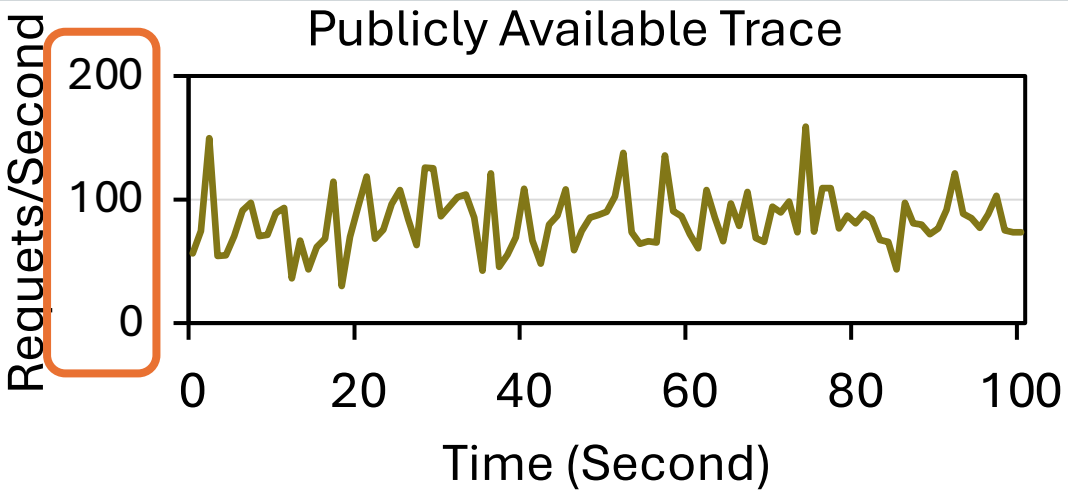
- Select a time interval
- Upscale number of arrivals in the interval
- Generate random timestamps in the interval
- Sample from requests in the interval

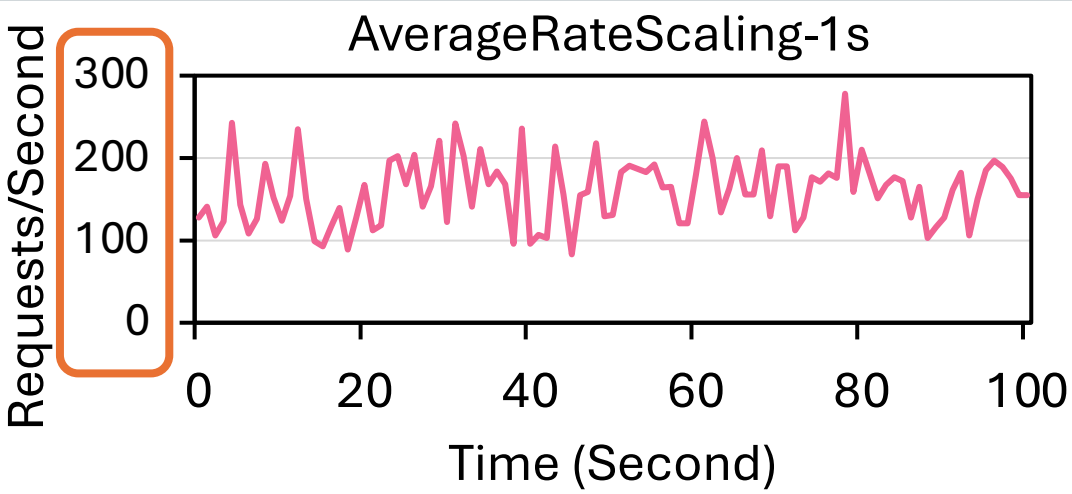
Upscaling Factor, $f = 2$

Time Interval = 4

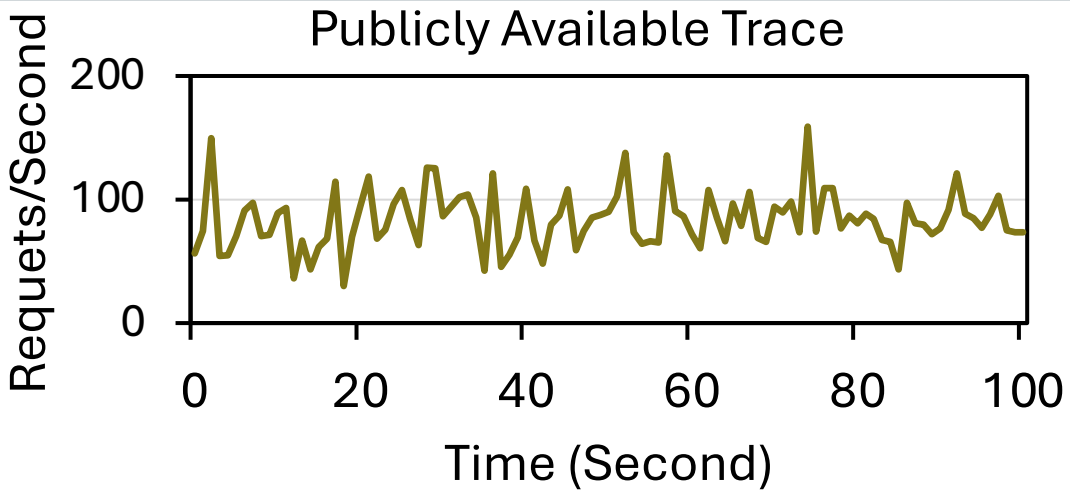


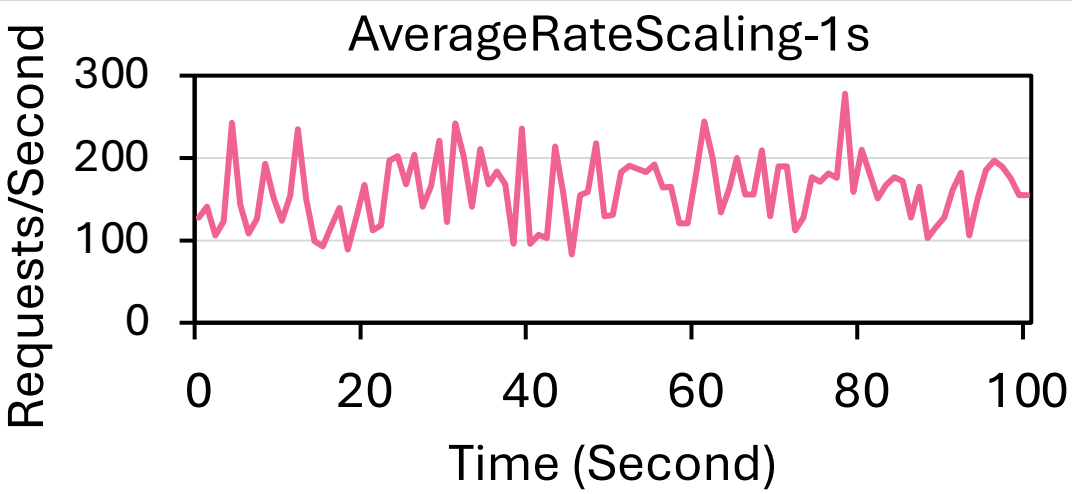
AverageRateScaling [WWW '22, SoCC '21, SoCC '18]

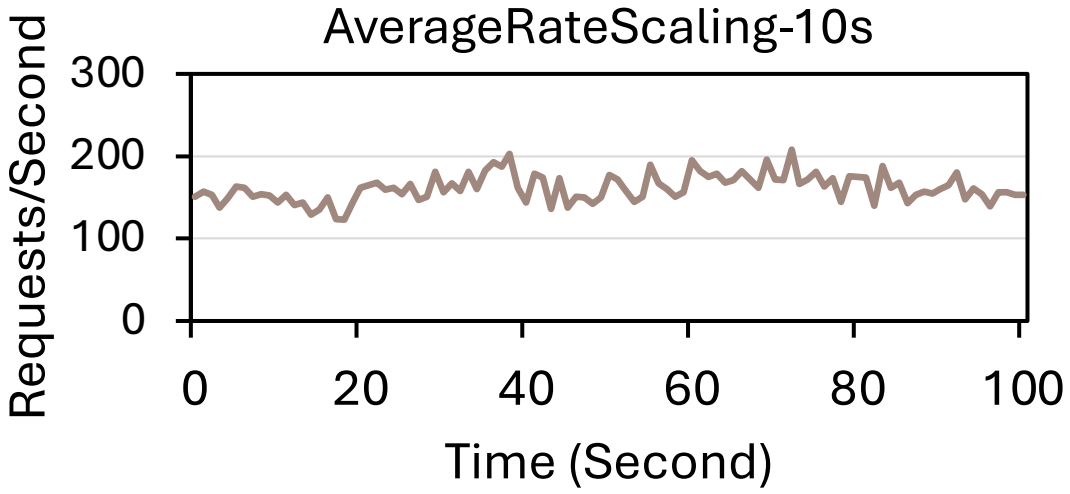


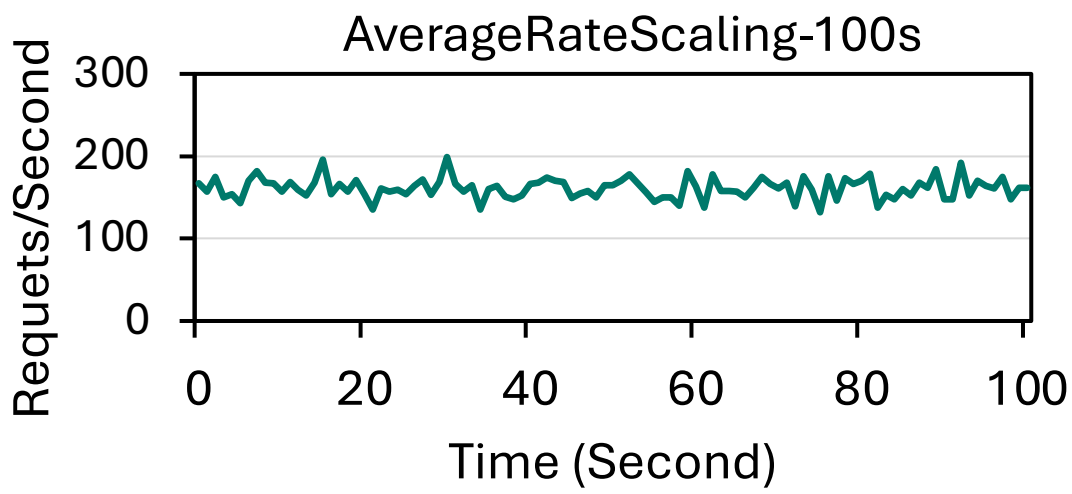


AverageRateScaling [WWW '22, SoCC '21, SoCC '18]



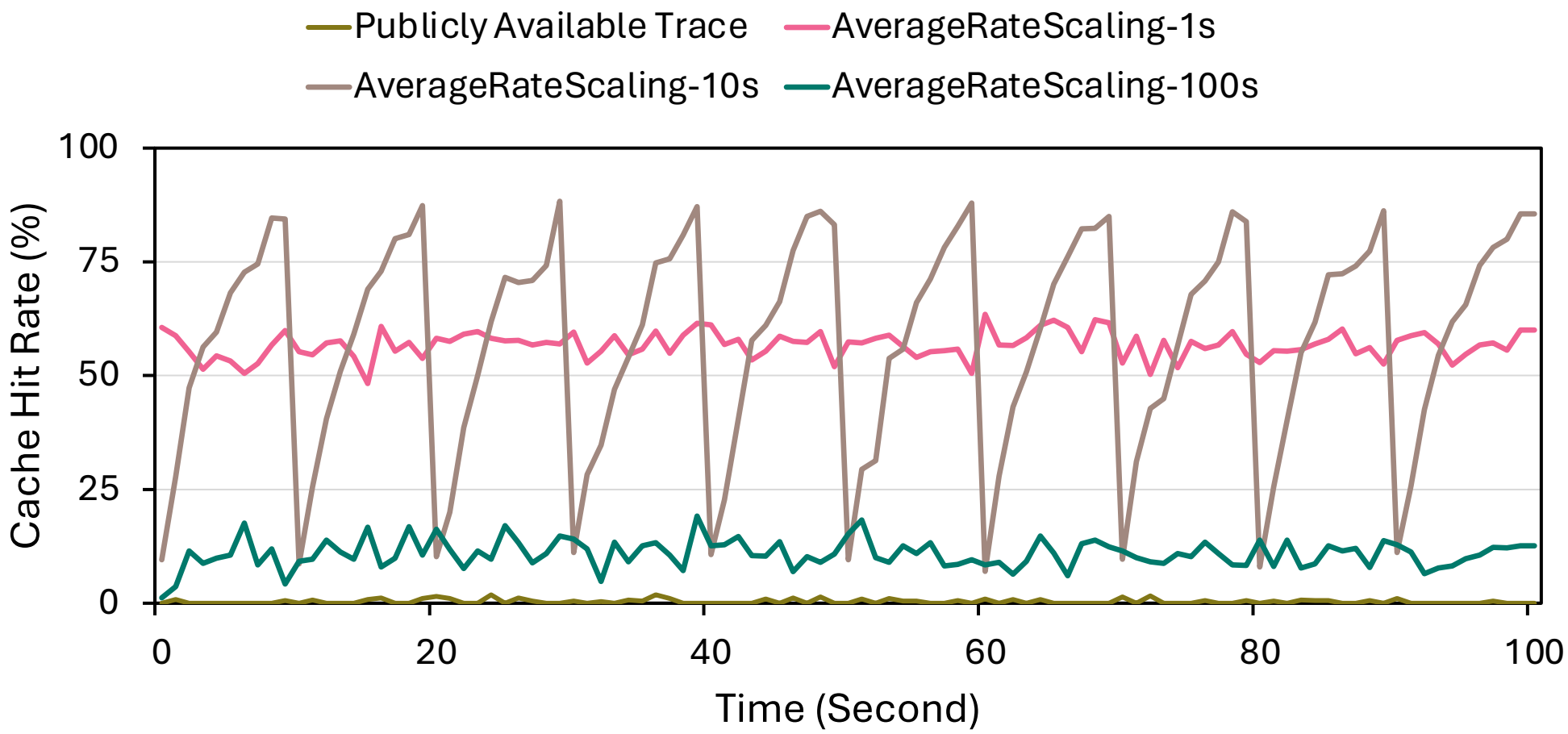






Too Long Interval Suppresses Short-Term Bursts

AverageRateScaling [WWW '22, SoCC '21, SoCC '18]



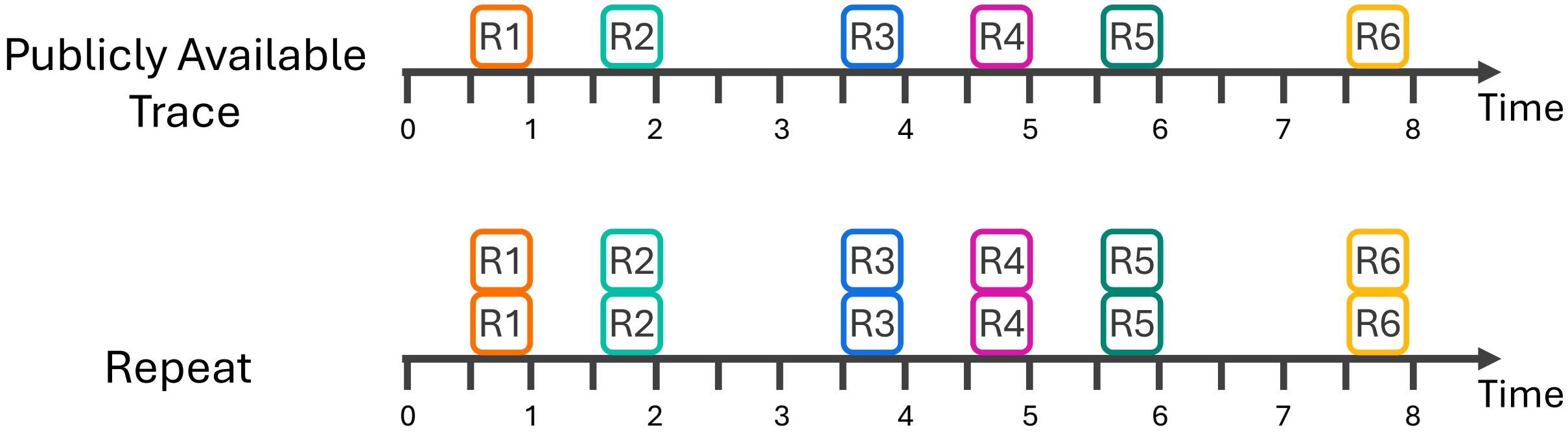
Too Short Interval Distort Cache Access Pattern

Repeat

Key Idea:

Repeat each request at exact timestamp

Upscaling Factor, $f = 2$

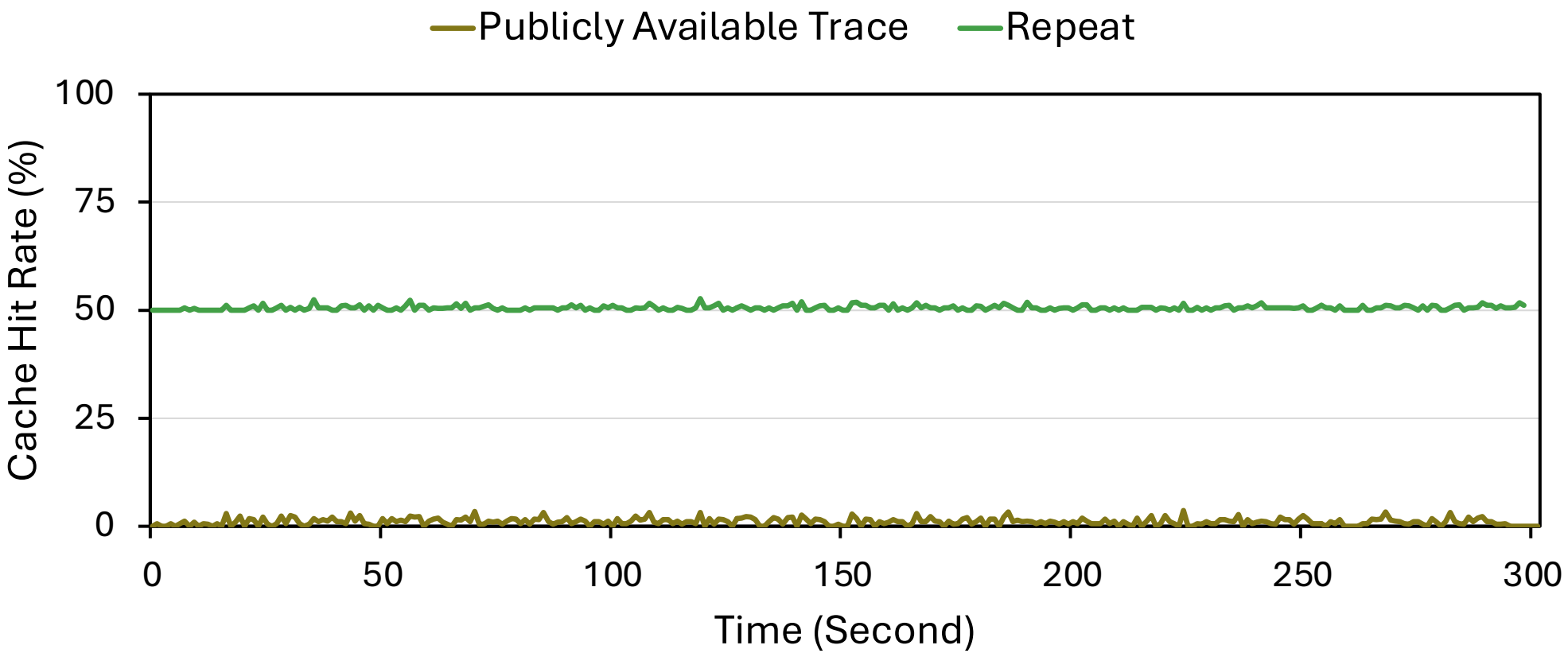


Repeat

Key Idea:

Repeat each request at exact timestamp

Upscaling Factor, $f = 2$



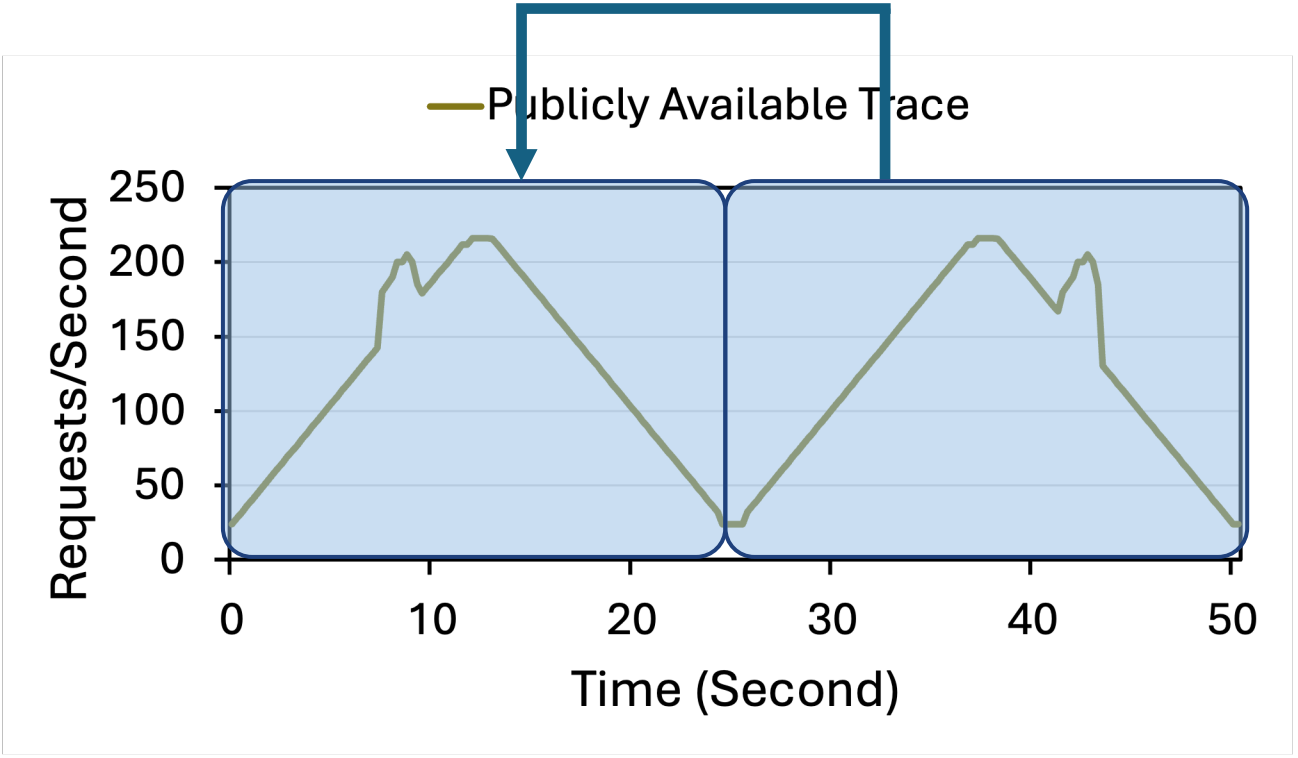
Repeat Distorts Cache Access Pattern

Fold

Key Idea:

Overlap *f* consecutive parts

Upscaling Factor, *f* = 2

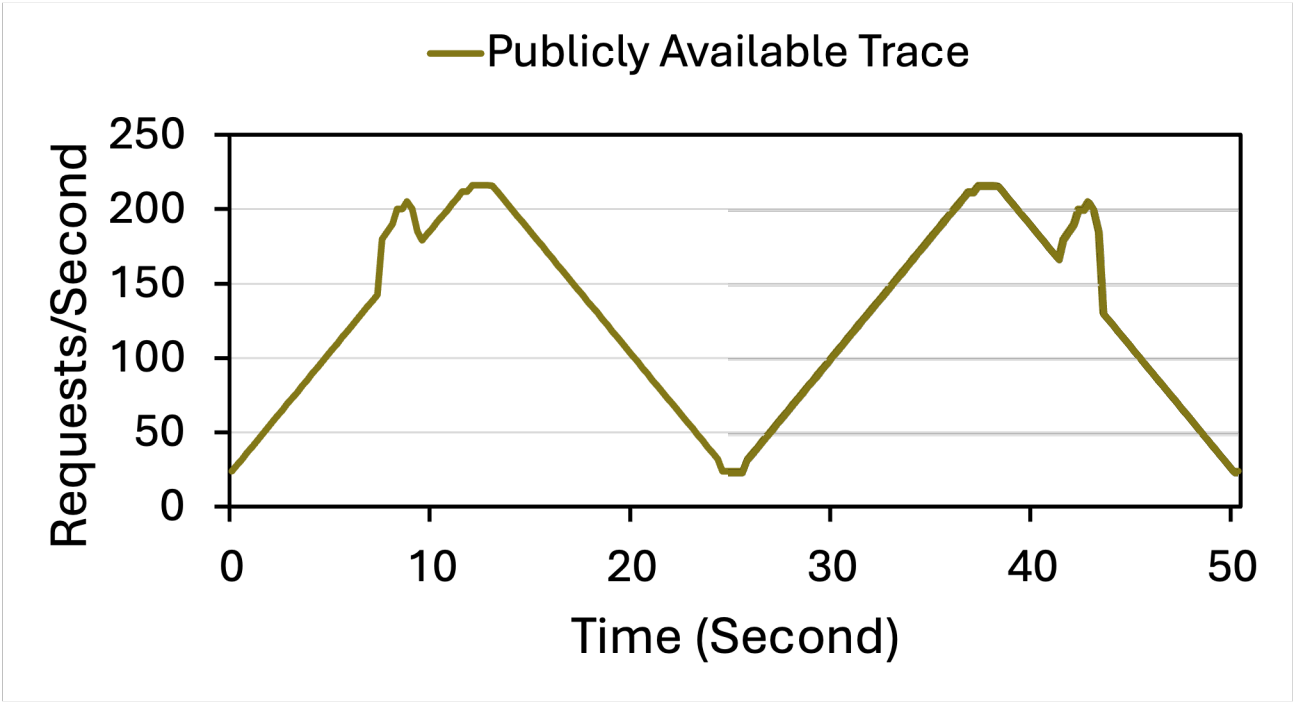


Fold

Key Idea:

Overlap *f* consecutive parts

Upscaling Factor, *f* = 2

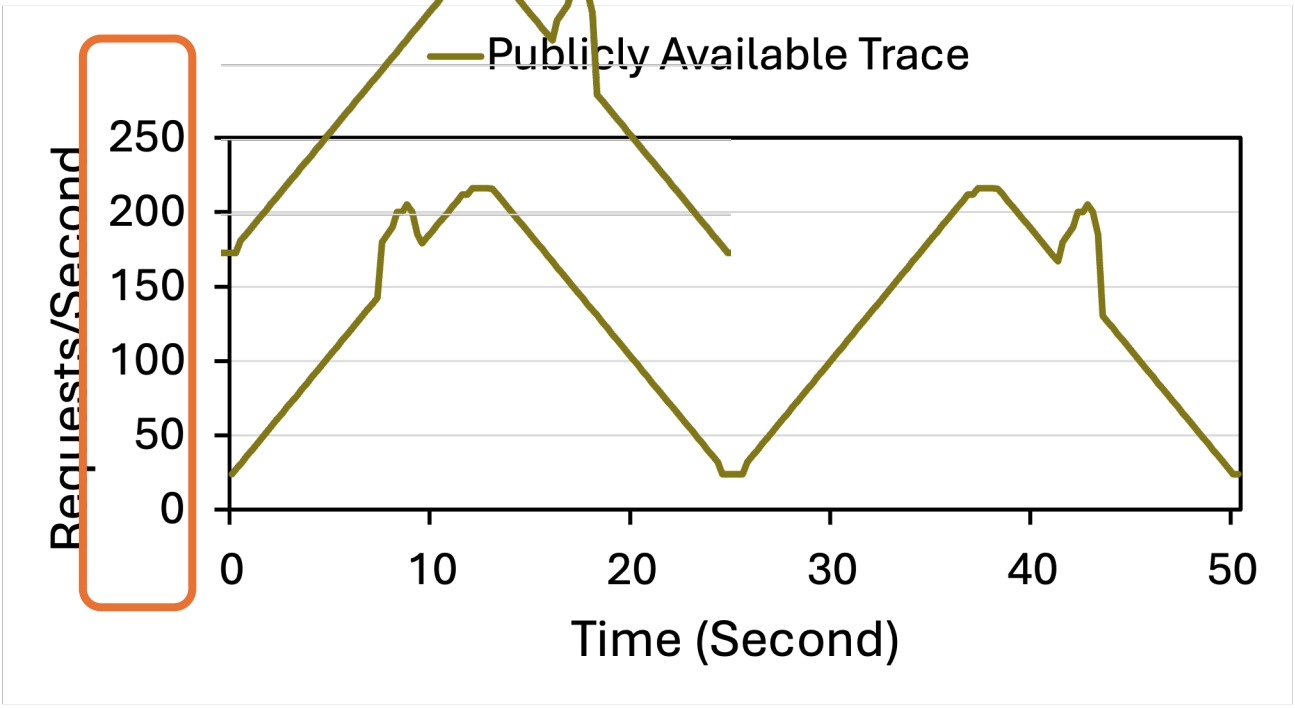


Fold

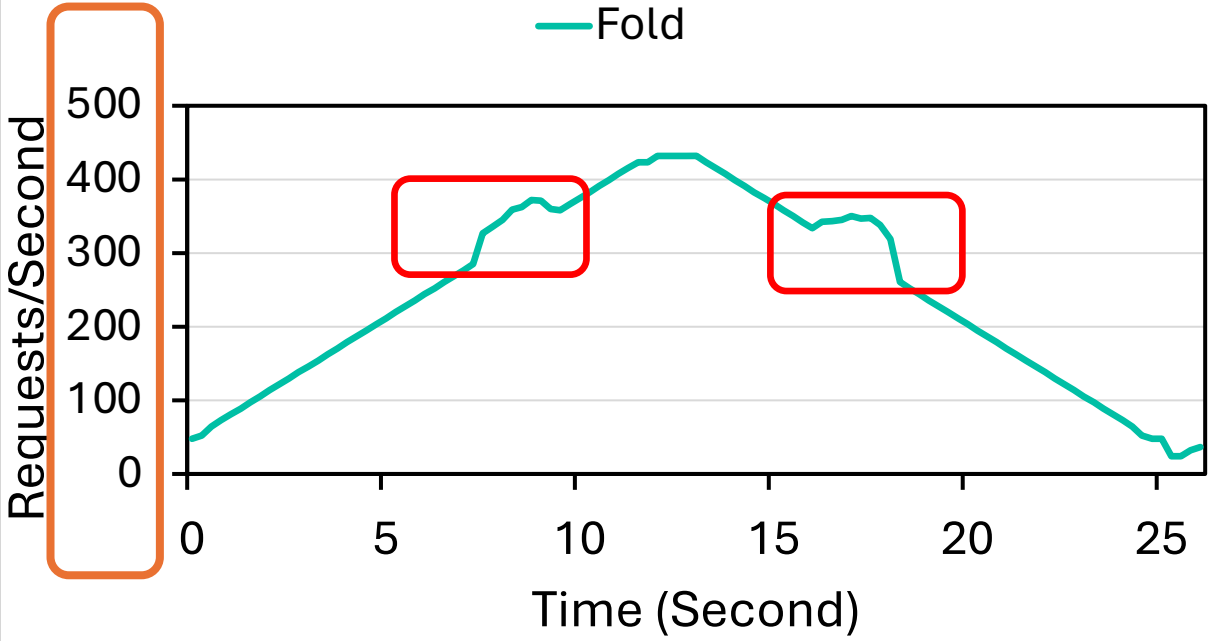
Key Idea:

Overlap *f* consecutive parts

Upscaling Factor, *f* = 2



Suppressed Burst



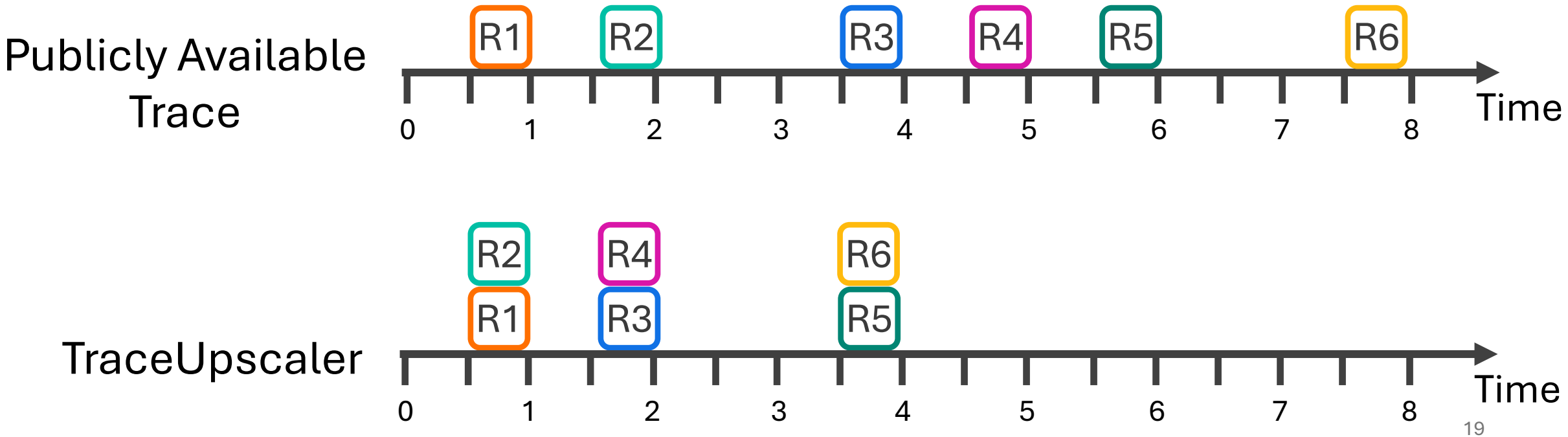
Fold Suppresses Short-Term Bursts

TraceUpscaler

Key Idea:

- Repeat arrival timestamps while using request parameters in the same order
  - Reuse arrival timestamps → preserve temporal pattern
  - Preserve relative order of requests → preserve cache access pattern

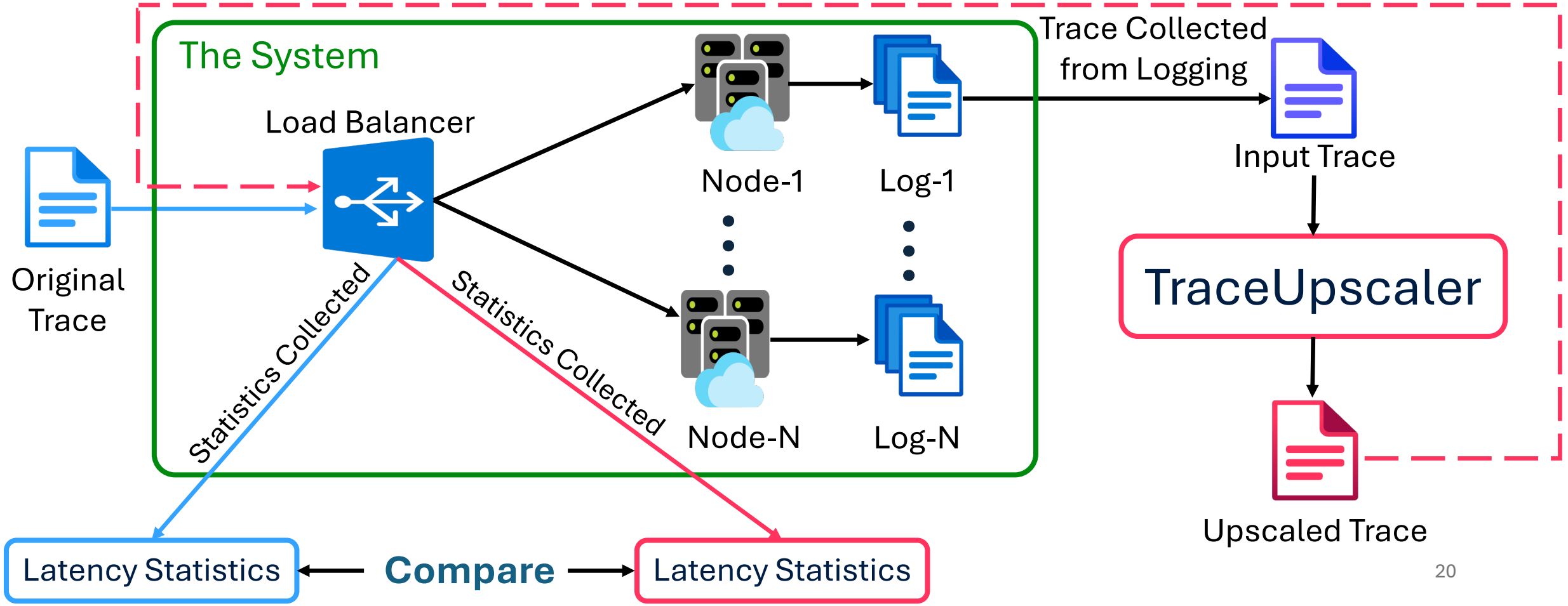
Upscaling Factor, $f = 2$



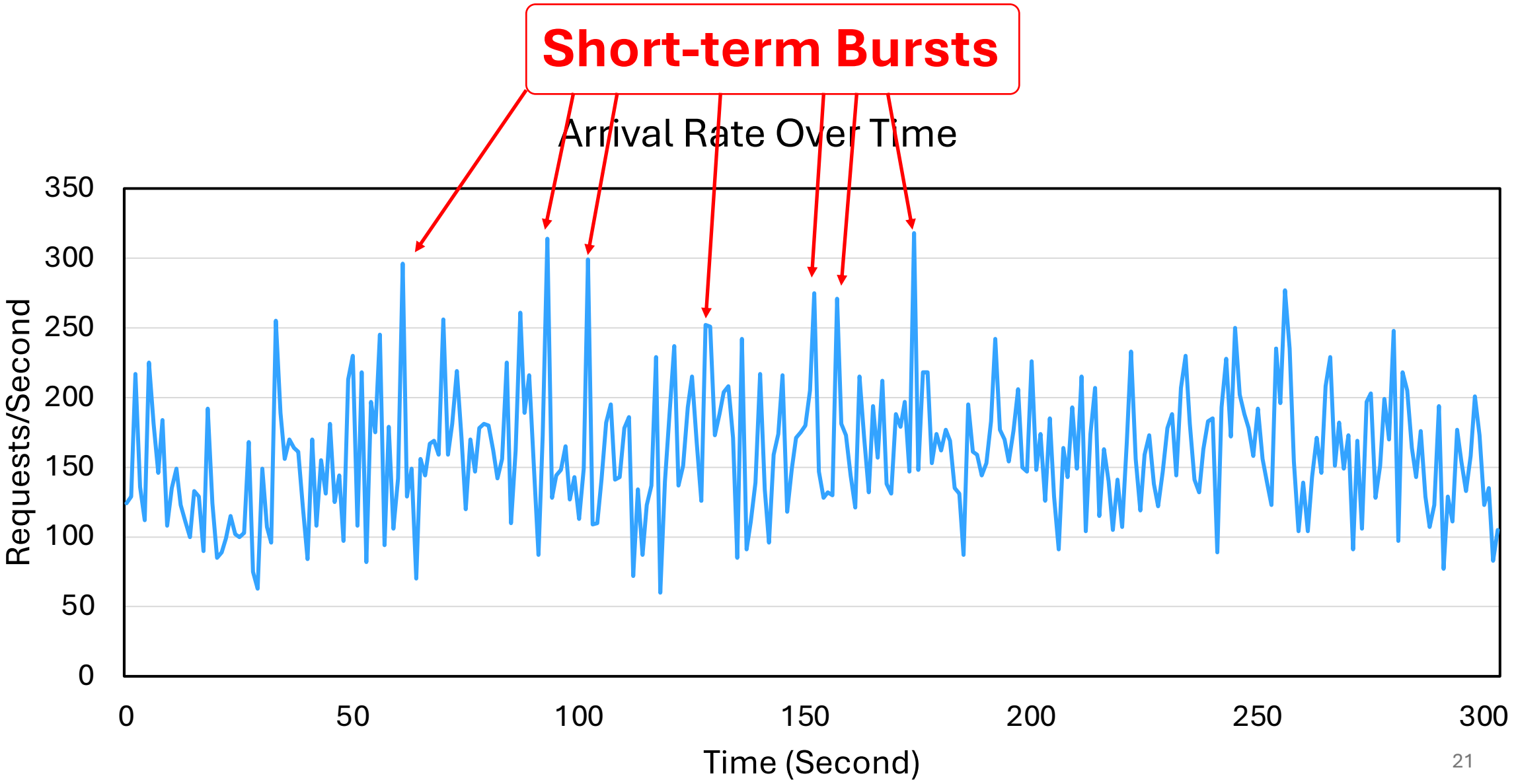
Evaluation of Upscaling Techniques

No ground truth → no way to evaluate upscaled trace

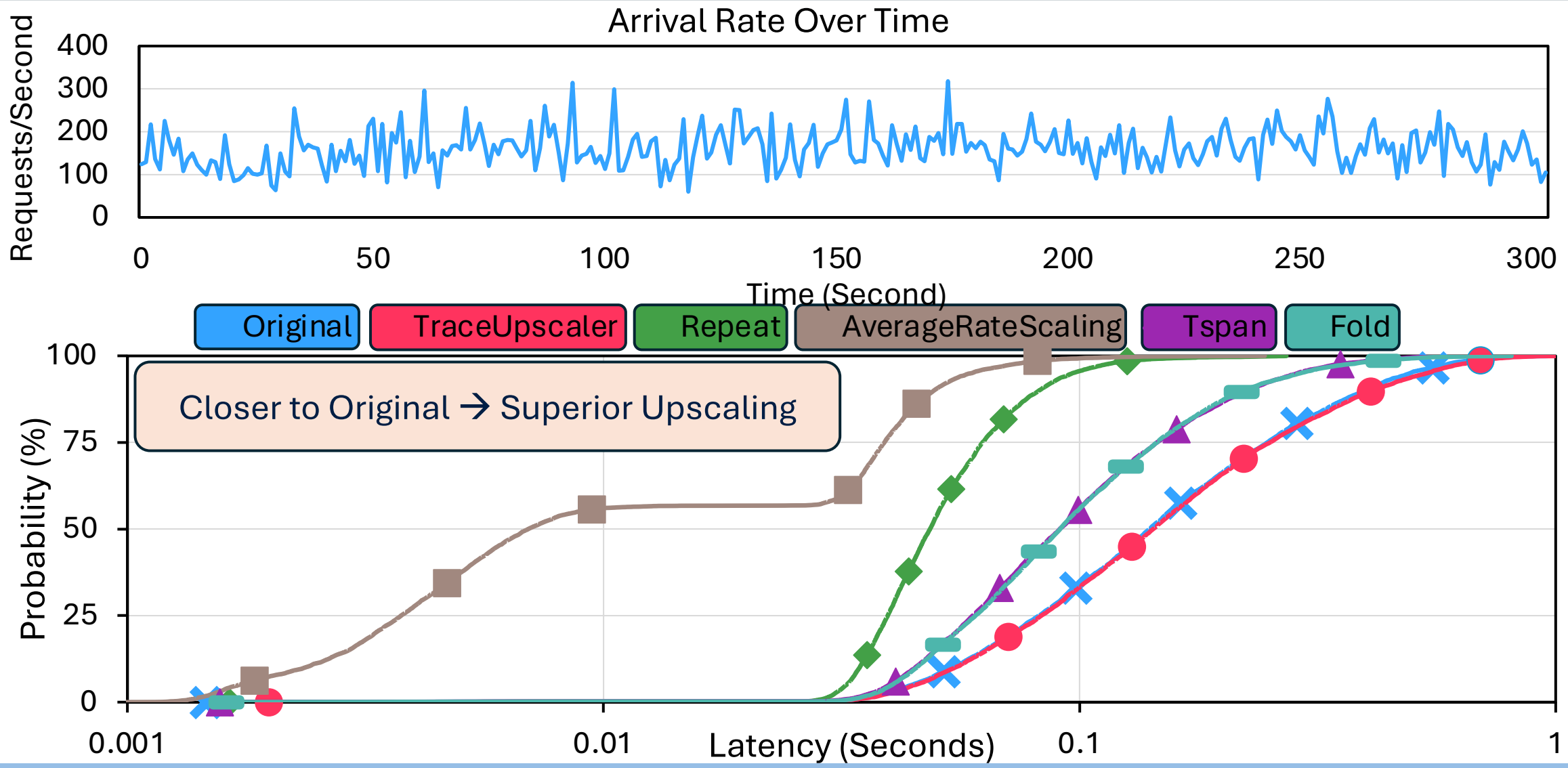
Reframe trace upscaling problem to trace reconstruction problem



Results with Real-World Arrival Times

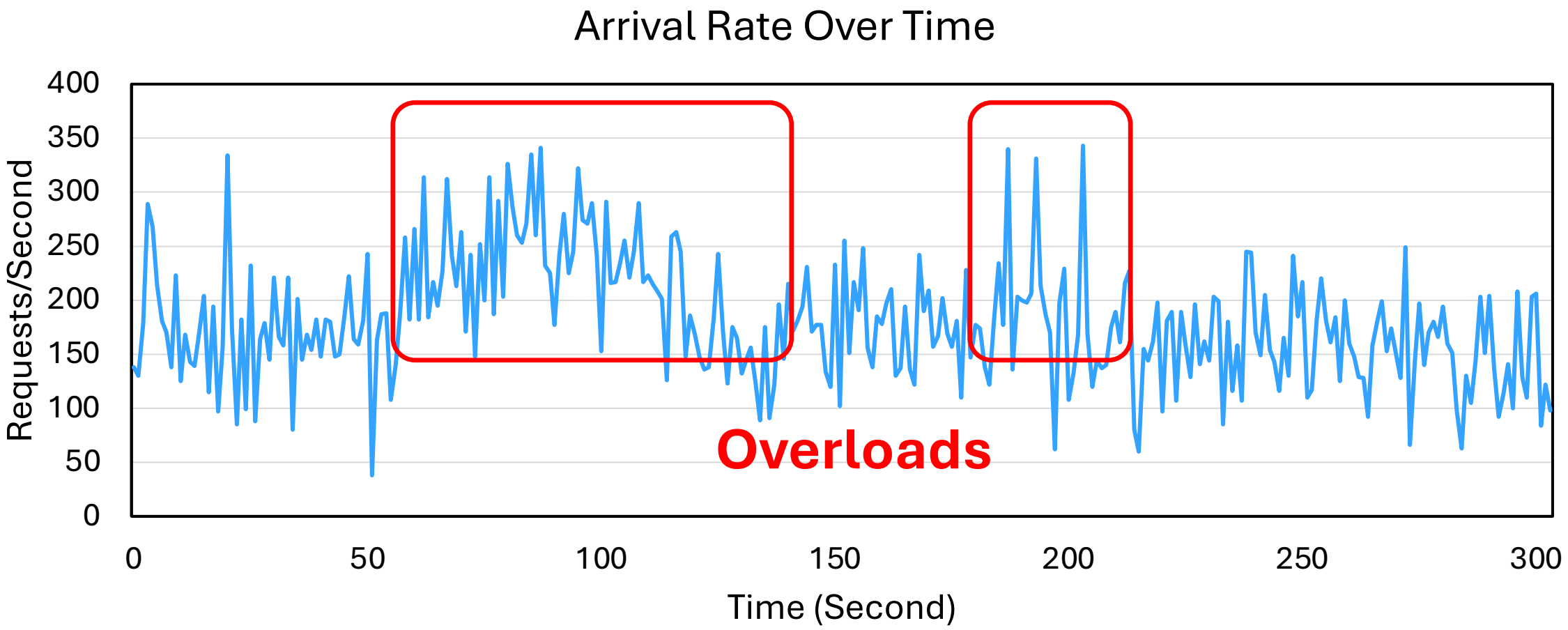


Results with Real-World Arrival Times

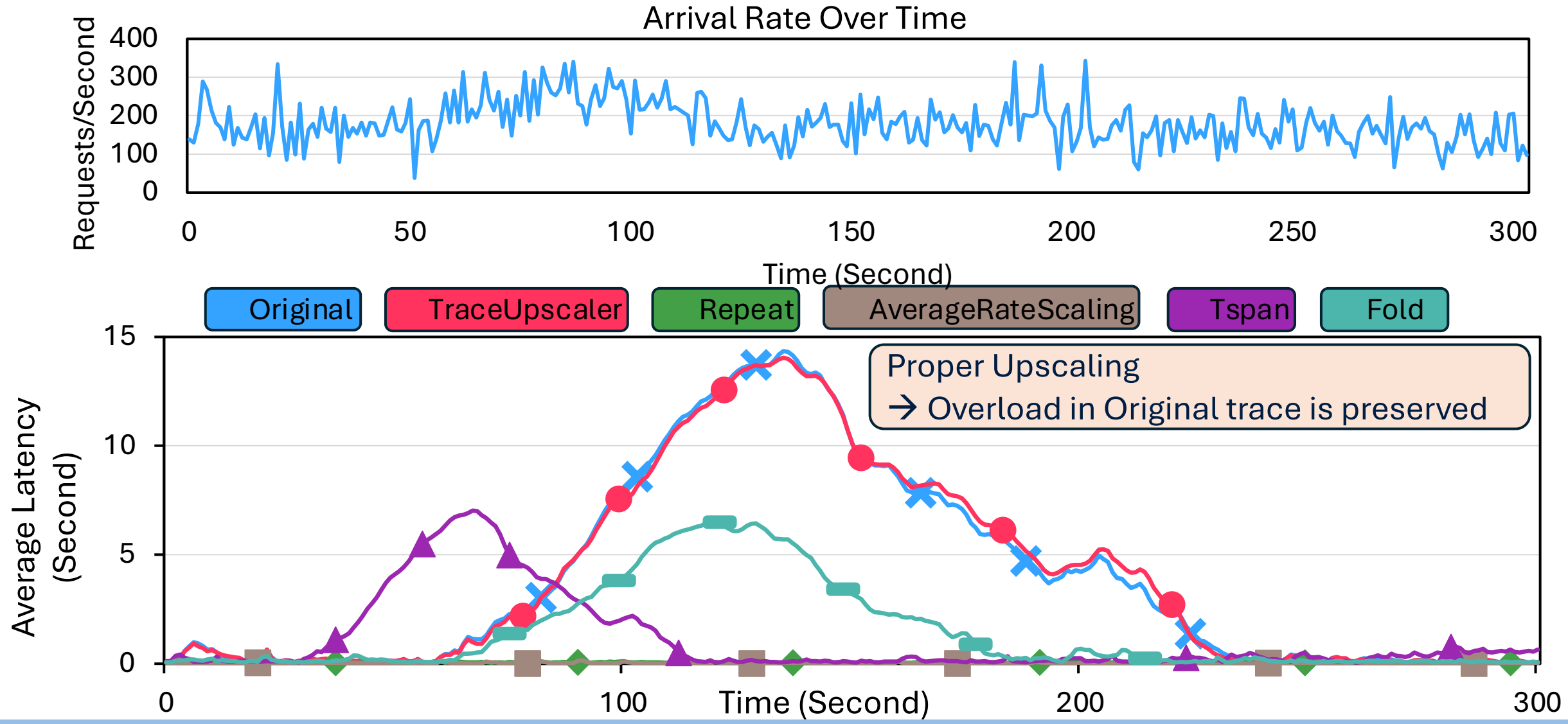


TraceUpscaler preserves latency distribution

Representing Overloads from Real-World



Representing Overloads



TraceUpscaler correctly represents overload in Original trace

Conclusion

- **Upscaling**: Increase load in trace for testing under high load
- **TraceUpscaler**: Decouple arrival timestamps from request parameters
  - Reuse timestamps → preserve temporal pattern
  - Reuse request parameters → preserve cache access pattern
- Experiments with real-world and synthetic data show TraceUpscaler's superiority
- Open-sourced code: github.com/smsajal/TraceUpscaler

PennState

Microsoft Research

<EURO/SYS'24>