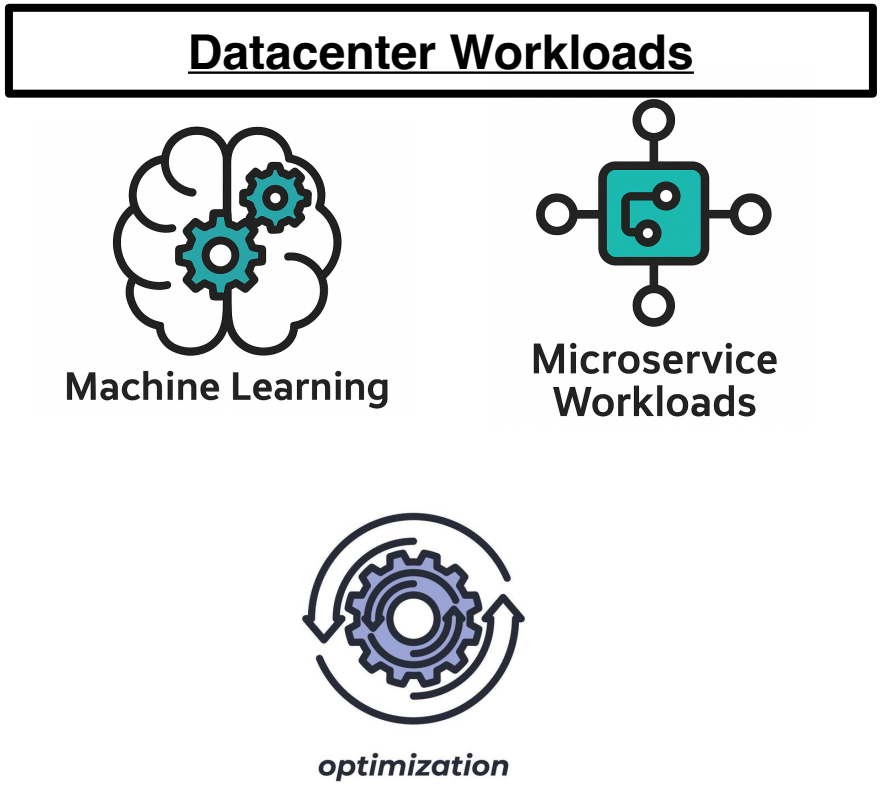
Modeling the Interactions between Core Allocation and Overload Control in μs-Scale Network Stacks

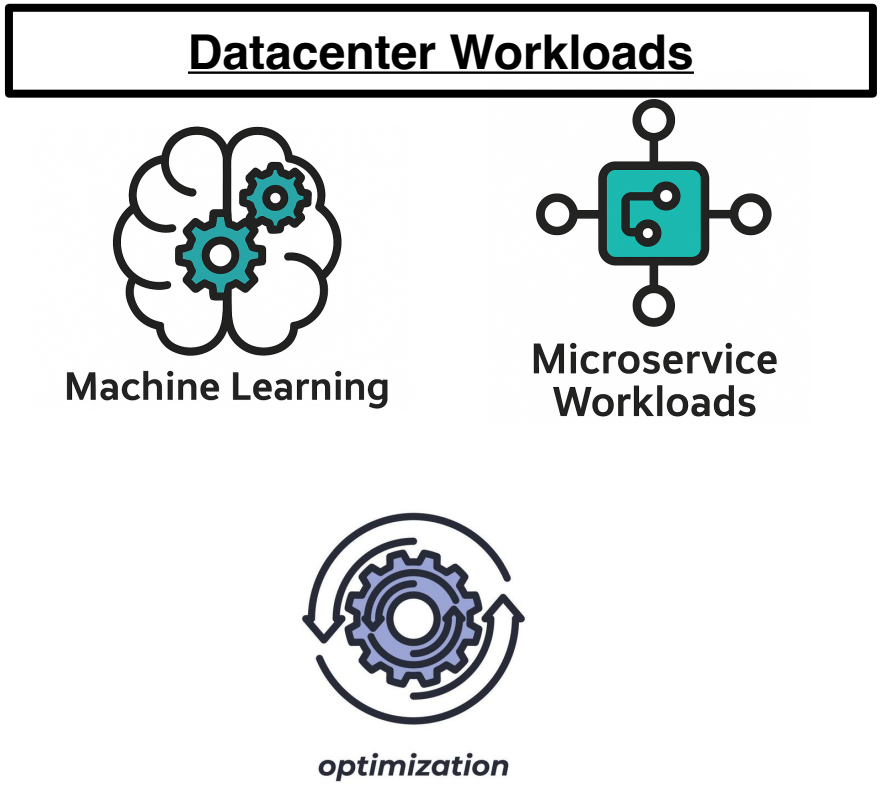
Jehad Hussien **Pratyush Sahu** Eric Stuhr Ahmed Saeed

Current Trends in Datacenter Workloads

Current Trends in Datacenter Workloads



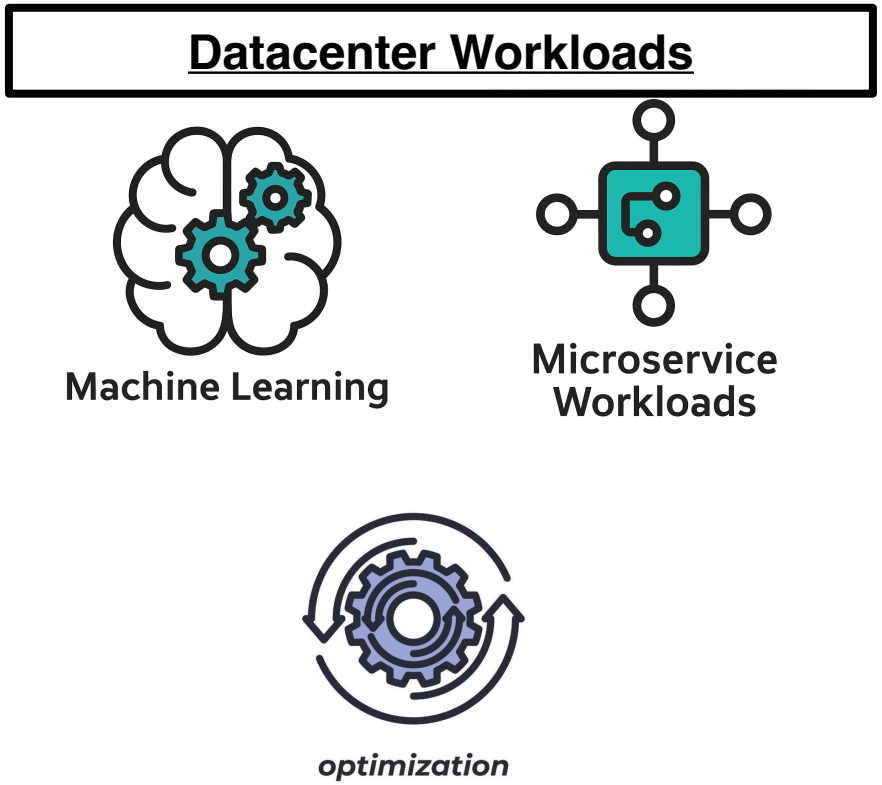
Current Trends in Datacenter Workloads



Service Level Objectives (SLO)

1. 99% “GET” RPCs completed in less than 10 µs (latency SLO)
2. Maintain an average of 1 million requests per second (throughput SLO)

Current Trends in Datacenter Workloads



Service Level Objectives (SLO)

1. 99% “GET” RPCs completed in less than 10 µs (latency SLO)
2. Maintain an average of 1 million requests per second (throughput SLO)

High Utilization

10-20% CPUs utilized in current datacenter servers leading to wastage of CPU cycles and energy

Multiple Controllers in a Single Server

Multiple Controllers in a Single Server

Core Allocation

Multiple Controllers in a Single Server

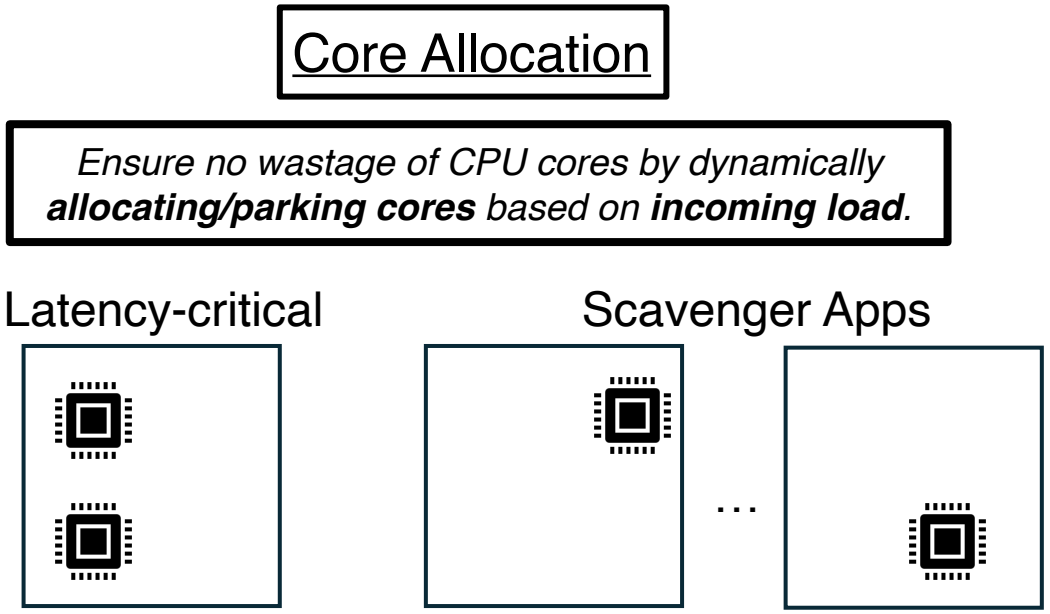
Core Allocation

*Ensure no wastage of CPU cores by dynamically **allocating/parking cores** based on **incoming load**.*

Multiple Controllers in a Single Server

Core Allocation

*Ensure no wastage of CPU cores by dynamically **allocating/parking cores** based on **incoming load**.*



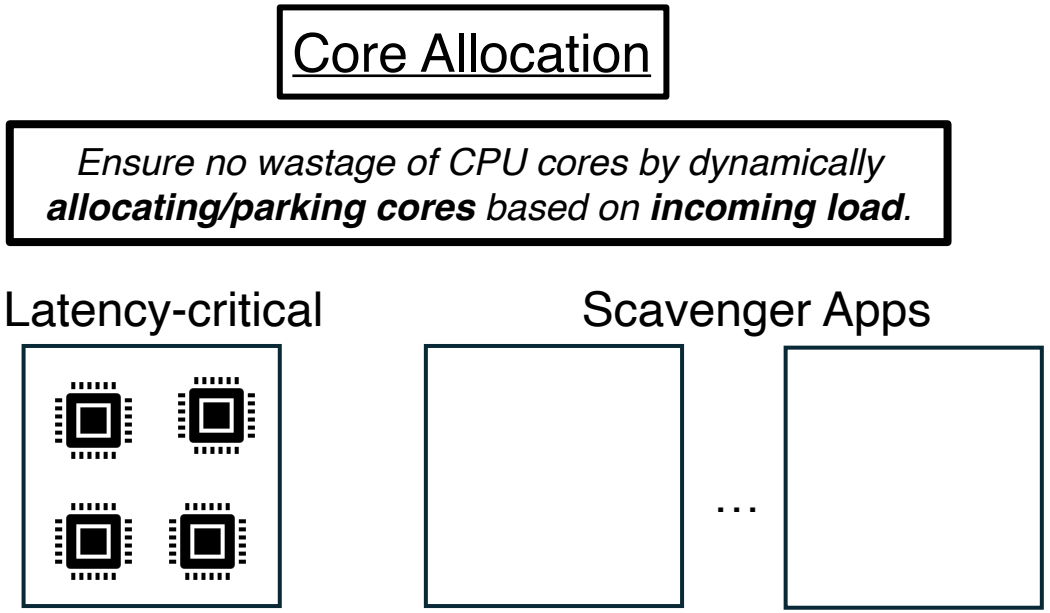
Low Load

High Load

Multiple Controllers in a Single Server

Core Allocation

*Ensure no wastage of CPU cores by dynamically **allocating/parking cores** based on **incoming load**.*



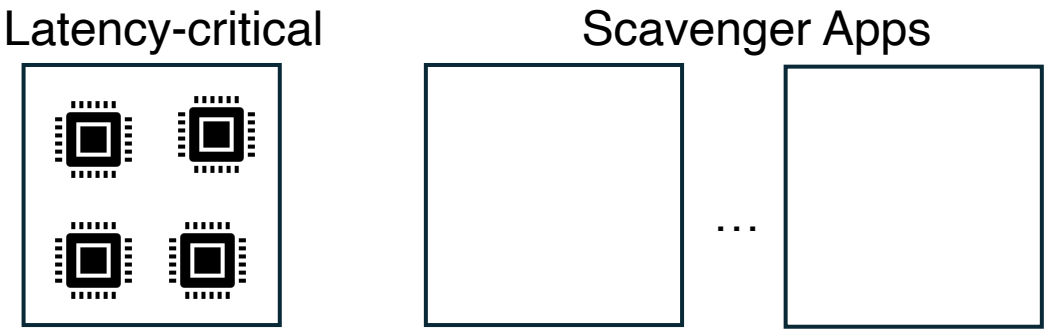
Low Load

High Load

Multiple Controllers in a Single Server

Core Allocation

*Ensure no wastage of CPU cores by dynamically **allocating/parking cores** based on **incoming load**.*



Low Load

High Load

Credit-based Overload Controller

*Avoid Congestion Collapse and ensure high performance by **controlling admission of requests***

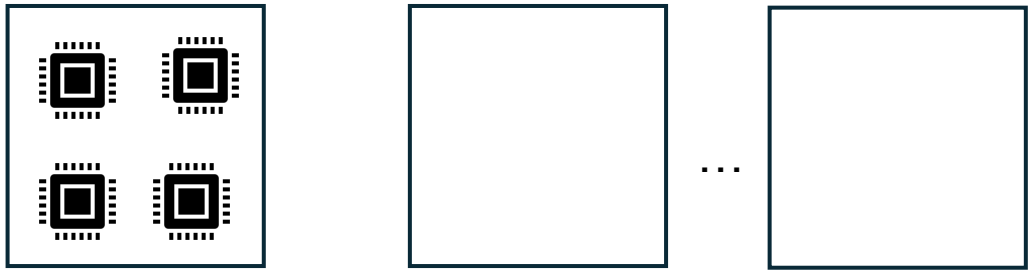
Multiple Controllers in a Single Server

Core Allocation

*Ensure no wastage of CPU cores by dynamically **allocating/parking cores** based on **incoming load**.*

Latency-critical

Scavenger Apps



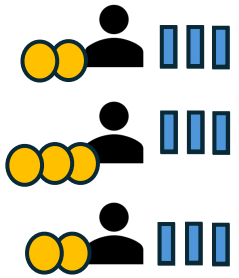
Low Load

High Load

Credit-based Overload Controller

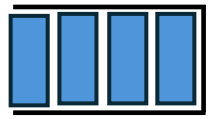
*Avoid Congestion Collapse and ensure high performance by **controlling admission of requests***

Credits =

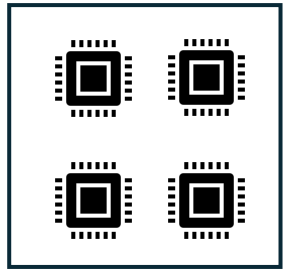


Server

Incoming Queue



Server Application



Low Load

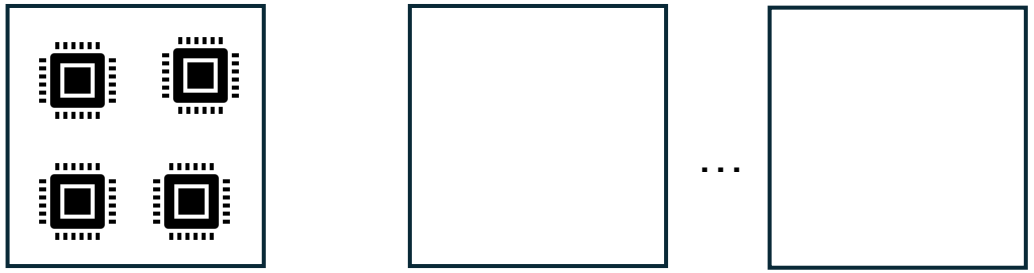
Multiple Controllers in a Single Server

Core Allocation

*Ensure no wastage of CPU cores by dynamically **allocating/parking cores** based on **incoming load**.*

Latency-critical

Scavenger Apps



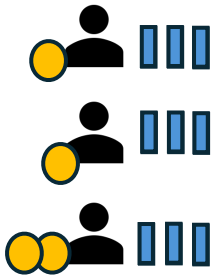
Low Load

High Load

Credit-based Overload Controller

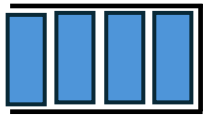
*Avoid Congestion Collapse and ensure high performance by **controlling admission of requests***

Credits =

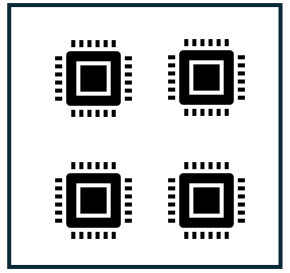


Server

Incoming Queue



Server Application

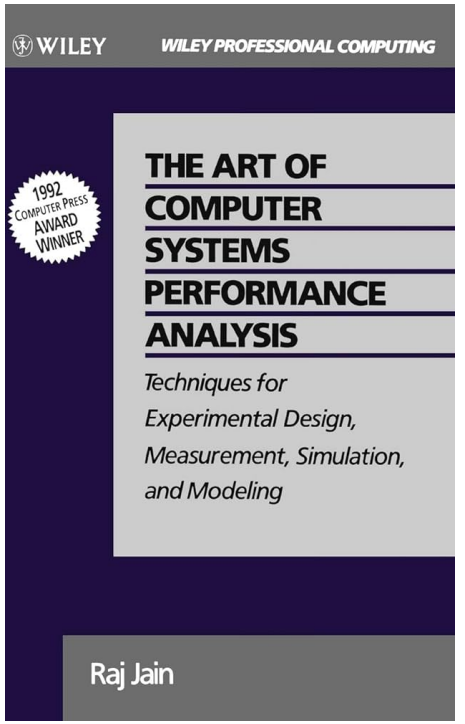


Low Load

High Load

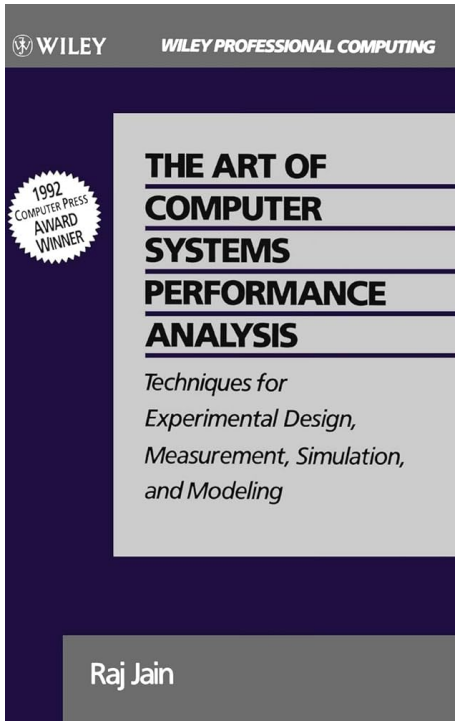
How do we model these controllers?

How do we model these controllers?



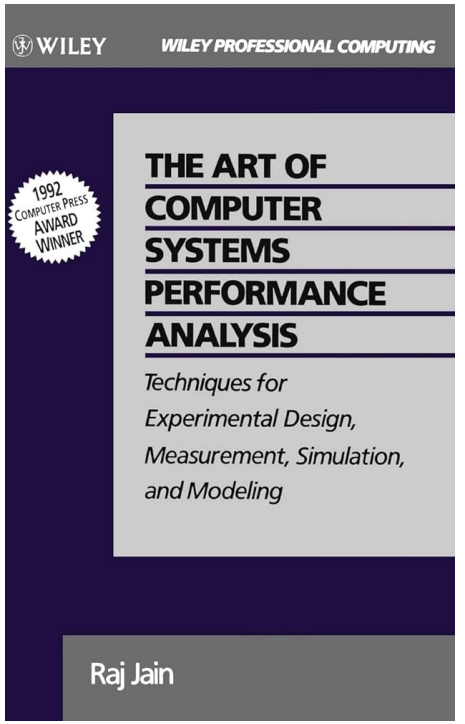
How do we model these controllers?

	Implementation	Simulation	Analytical



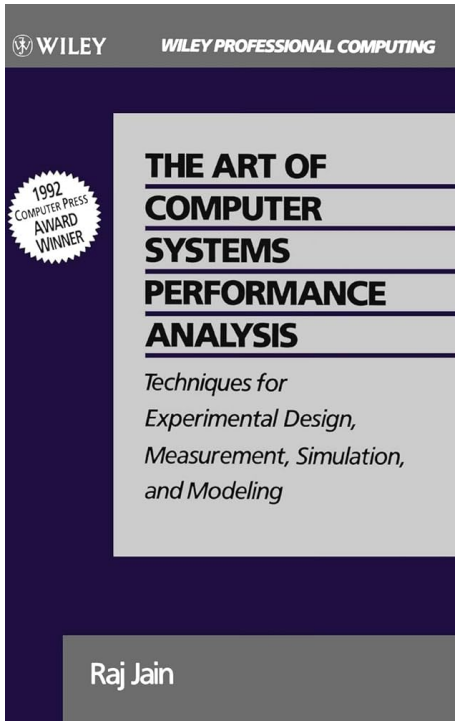
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance			



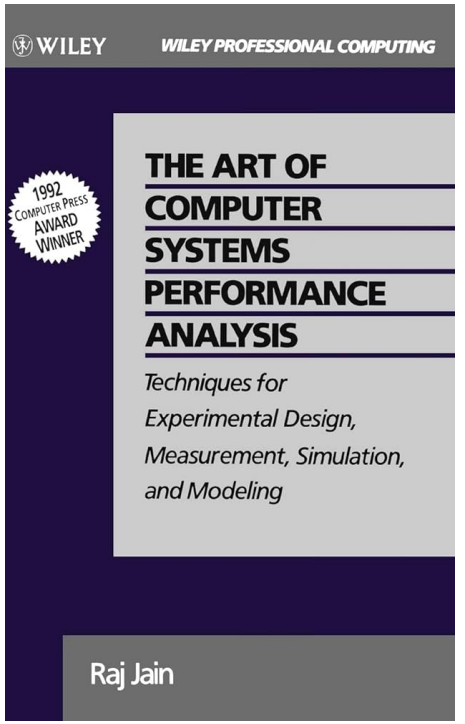
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical		



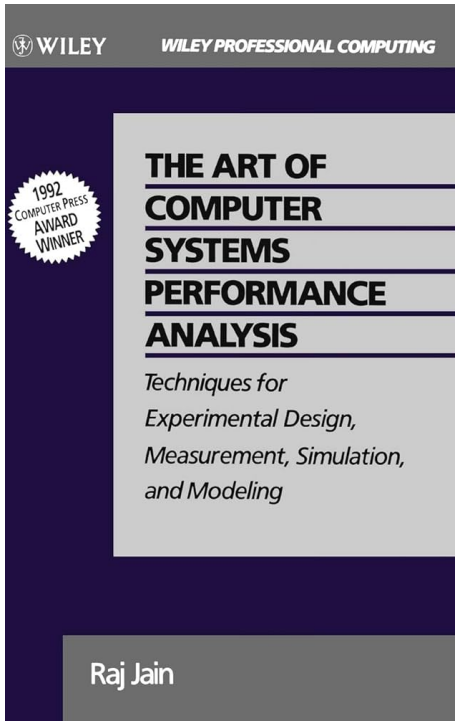
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	



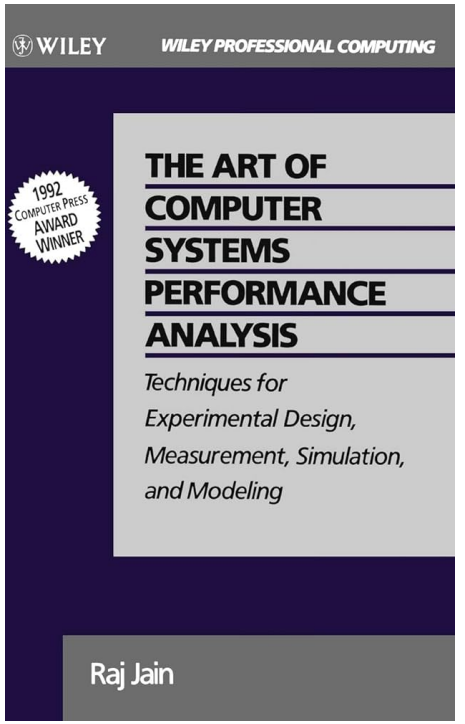
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed



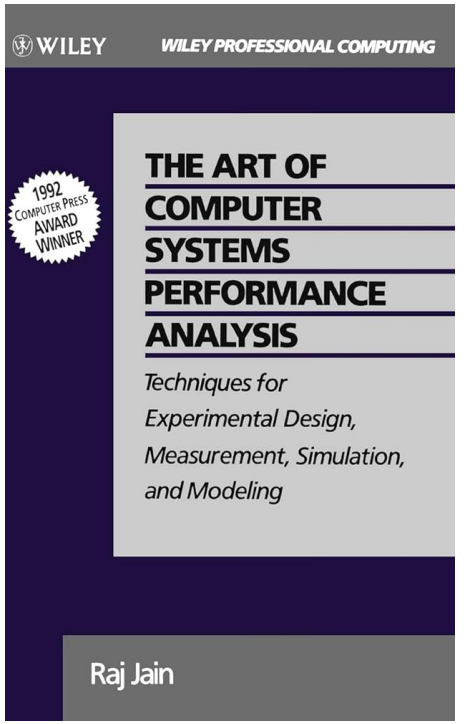
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed
Complexity and Extensibility			



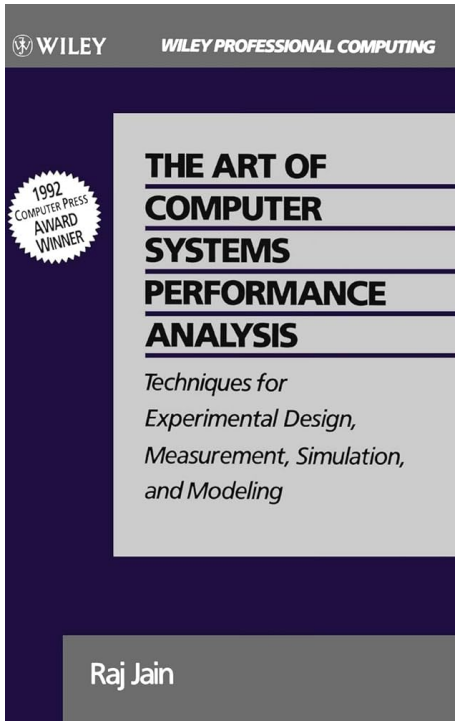
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed
Complexity and Extensibility	Hard		



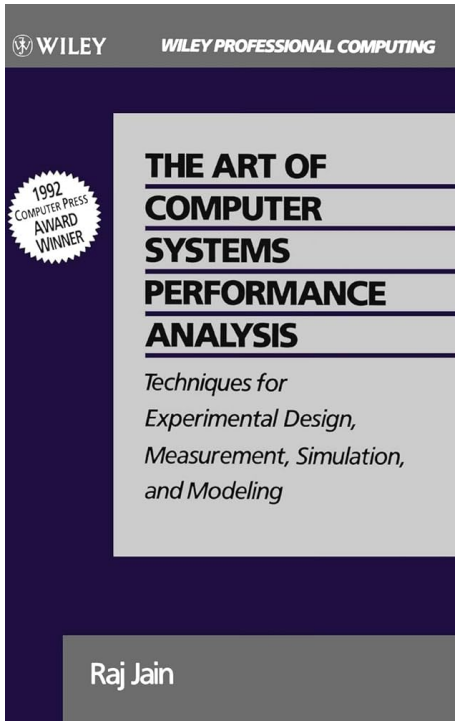
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed
Complexity and Extensibility	Hard	Easy	



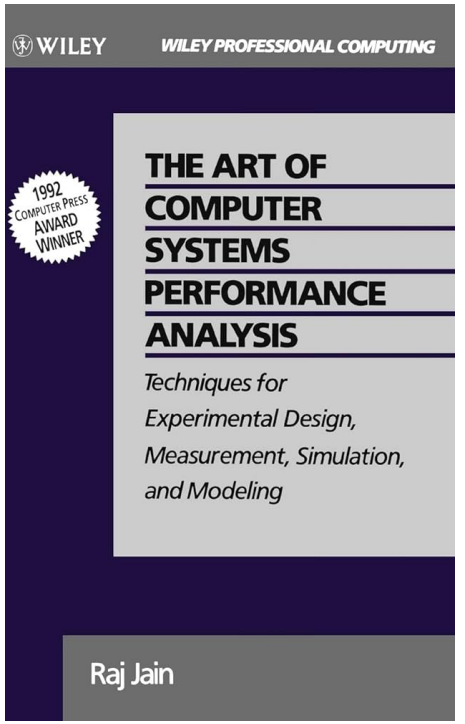
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed
Complexity and Extensibility	Hard	Easy	Hard



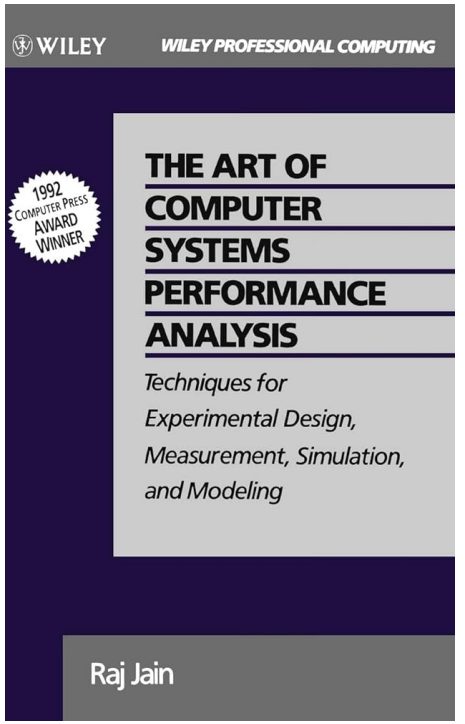
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed
Complexity and Extensibility	Hard	Easy	Hard
Realistic			



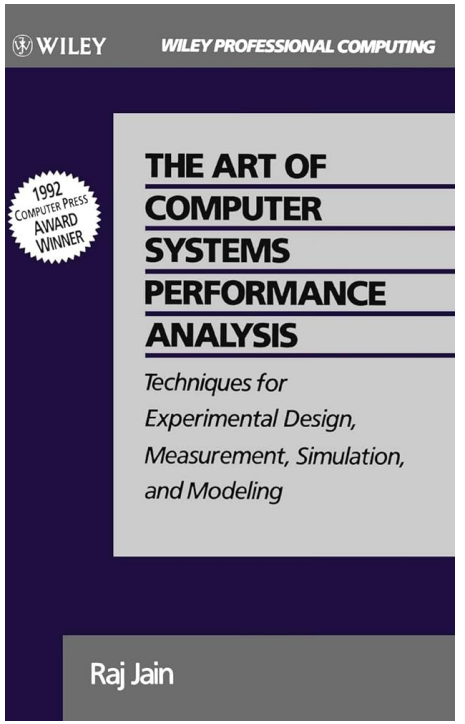
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed
Complexity and Extensibility	Hard	Easy	Hard
Realistic	Yes		



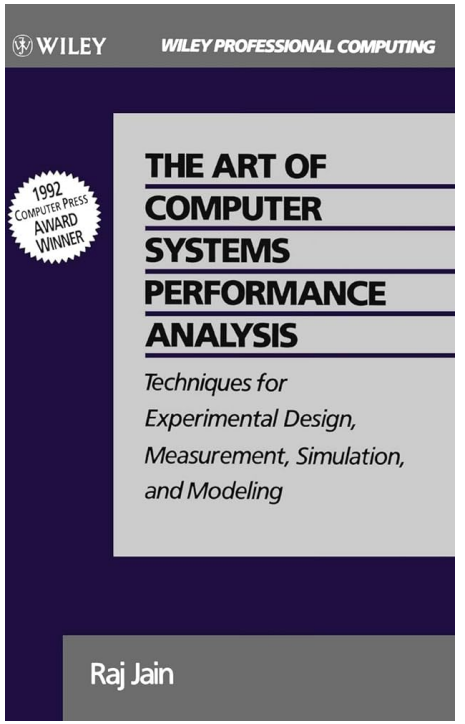
How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed
Complexity and Extensibility	Hard	Easy	Hard
Realistic	Yes	Simplified	



How do we model these controllers?

	Implementation	Simulation	Analytical
Proof of Performance	Only empirical	Statistical	Guaranteed
Complexity and Extensibility	Hard	Easy	Hard
Realistic	Yes	Simplified	Oversimplified



Performance Verification

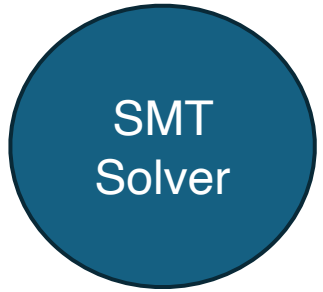
Performance Verification

Definition: Using finite model checking to verify performance properties of a system that is modeled as logical predicates

Performance Verification

Definition: Using finite model checking to verify performance properties of a system that is modeled as logical predicates

Use **Satisfiability Modulo Theories Solvers (SMT Solvers)** to find a possible scenario where a particular performance is met

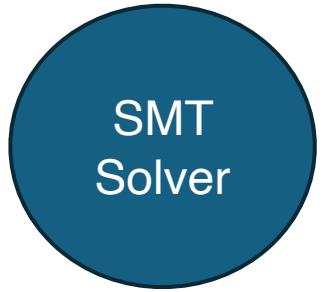


Performance Verification

Definition: Using finite model checking to verify performance properties of a system that is modeled as logical predicates

Use **Satisfiability Modulo Theories Solvers (SMT Solvers)** to find a possible scenario where a particular performance is met

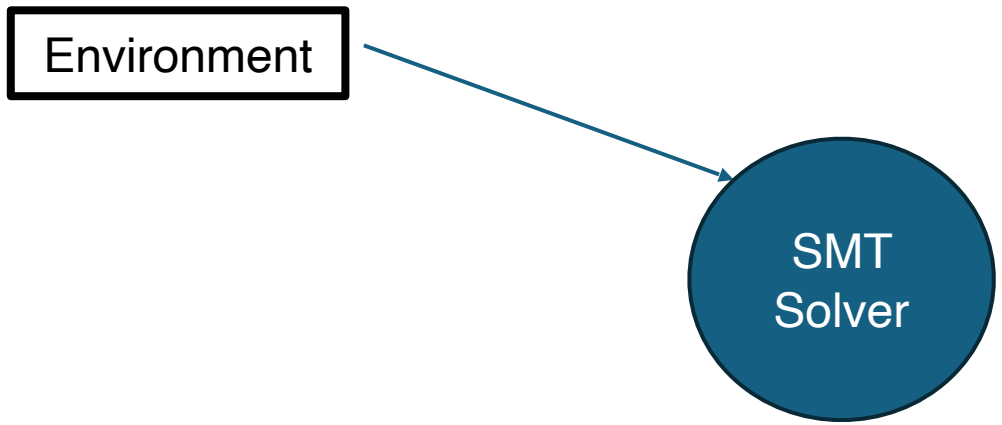
Represent **system**, the **controller algorithms**, and **performance query** as **Boolean constraints and first-order logic** over system state variables



Performance Verification

Definition: Using finite model checking to verify performance properties of a system that is modeled as logical predicates

Use **Satisfiability Modulo Theories Solvers (SMT Solvers)** to find a possible scenario where a particular performance is met

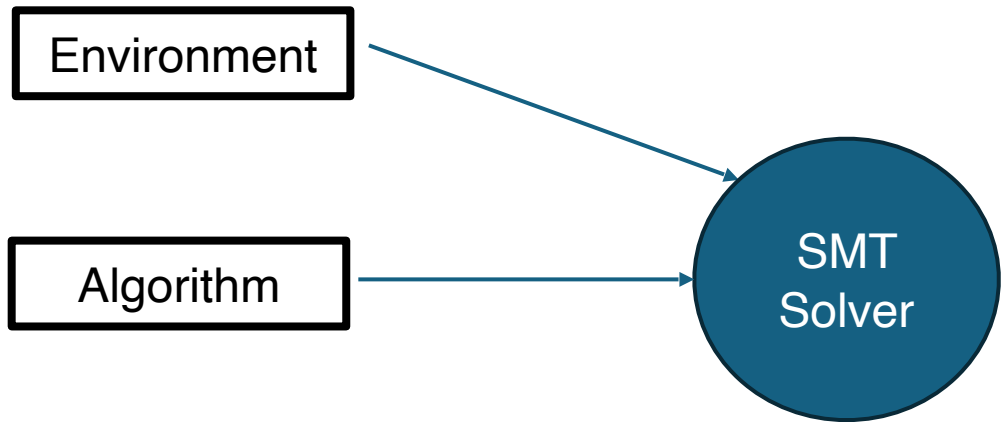


Represent **system**, the **controller algorithms**, and **performance query** as **Boolean constraints and first-order logic** over system state variables

Performance Verification

Definition: Using finite model checking to verify performance properties of a system that is modeled as logical predicates

Use **Satisfiability Modulo Theories Solvers (SMT Solvers)** to find a possible scenario where a particular performance is met



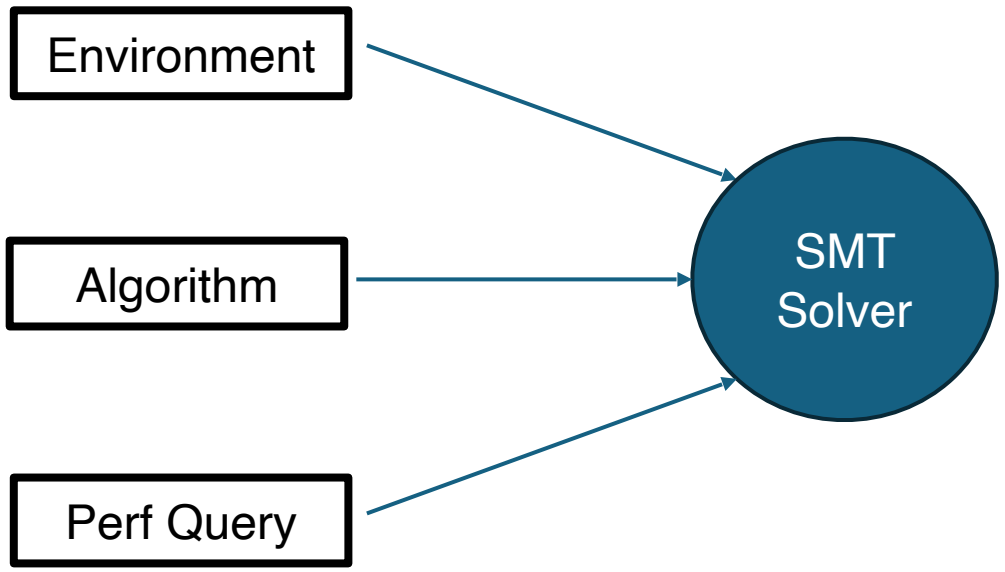
Represent **system**, the **controller algorithms**, and **performance query** as **Boolean constraints and first-order logic** over system state variables

Performance Verification

Definition: Using finite model checking to verify performance properties of a system that is modeled as logical predicates

Use **Satisfiability Modulo Theories Solvers (SMT Solvers)** to find a possible scenario where a particular performance is met

Represent **system**, the **controller algorithms**, and **performance query** as **Boolean constraints and first-order logic** over system state variables

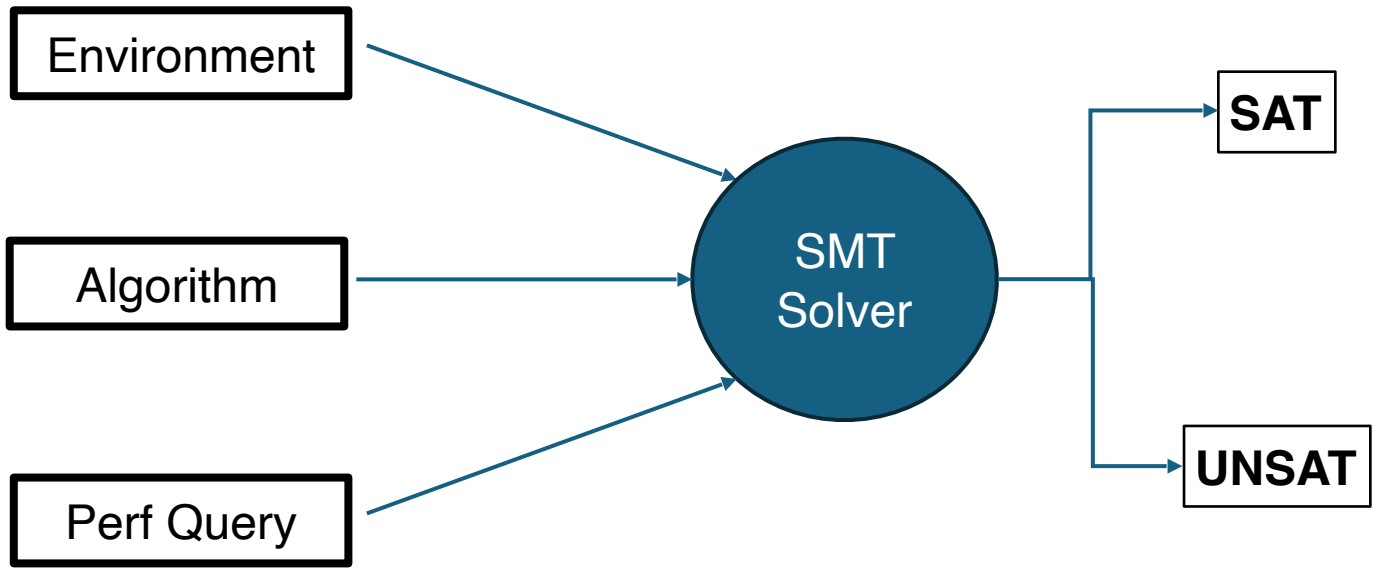


Performance Verification

Definition: Using finite model checking to verify performance properties of a system that is modeled as logical predicates

Use **Satisfiability Modulo Theories Solvers (SMT Solvers)** to find a possible scenario where a particular performance is met

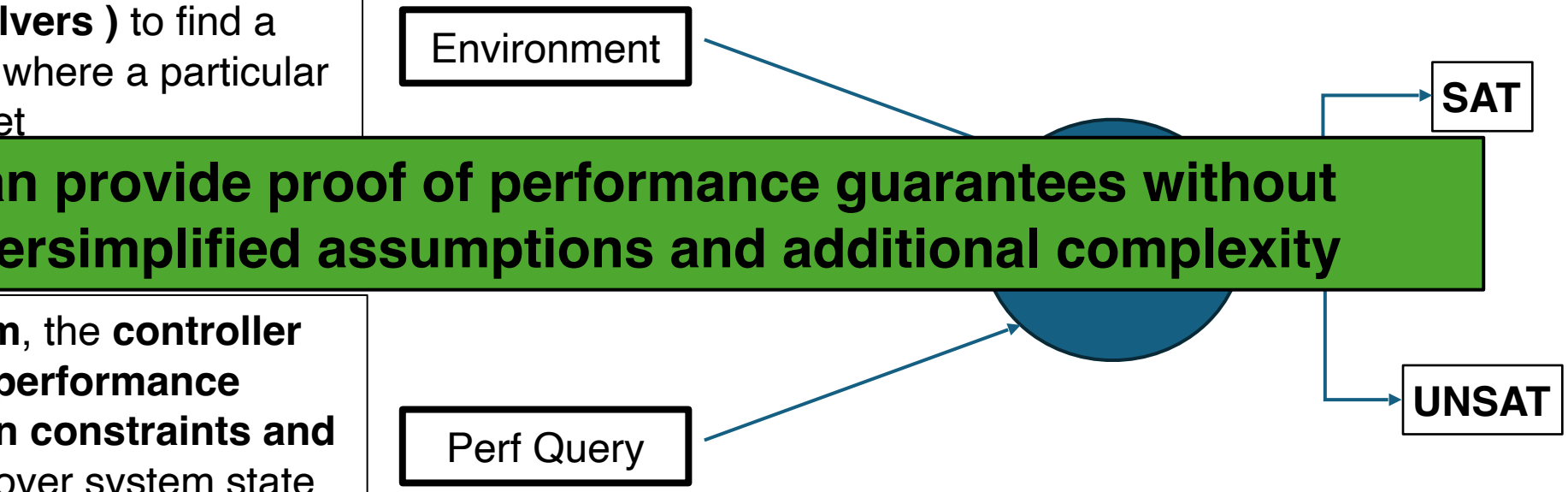
Represent **system**, the **controller algorithms**, and **performance query** as **Boolean constraints and first-order logic** over system state variables



Performance Verification

Definition: Using finite model checking to verify performance properties of a system that is modeled as logical predicates

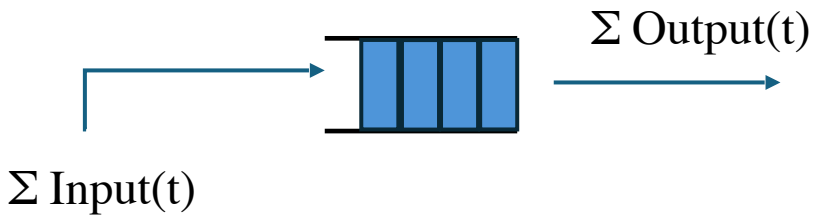
Use **Satisfiability Modulo Theories Solvers (SMT Solvers)** to find a possible scenario where a particular performance is met



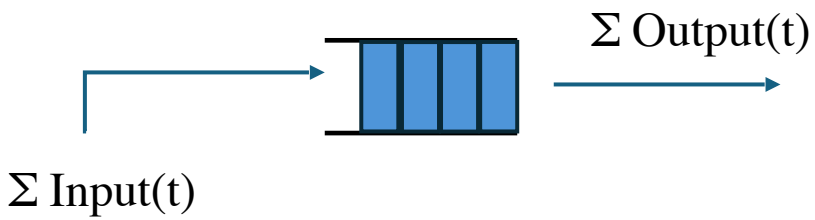
Represent **system**, the **controller algorithms**, and **performance query** as **Boolean constraints and first-order logic** over system state variables

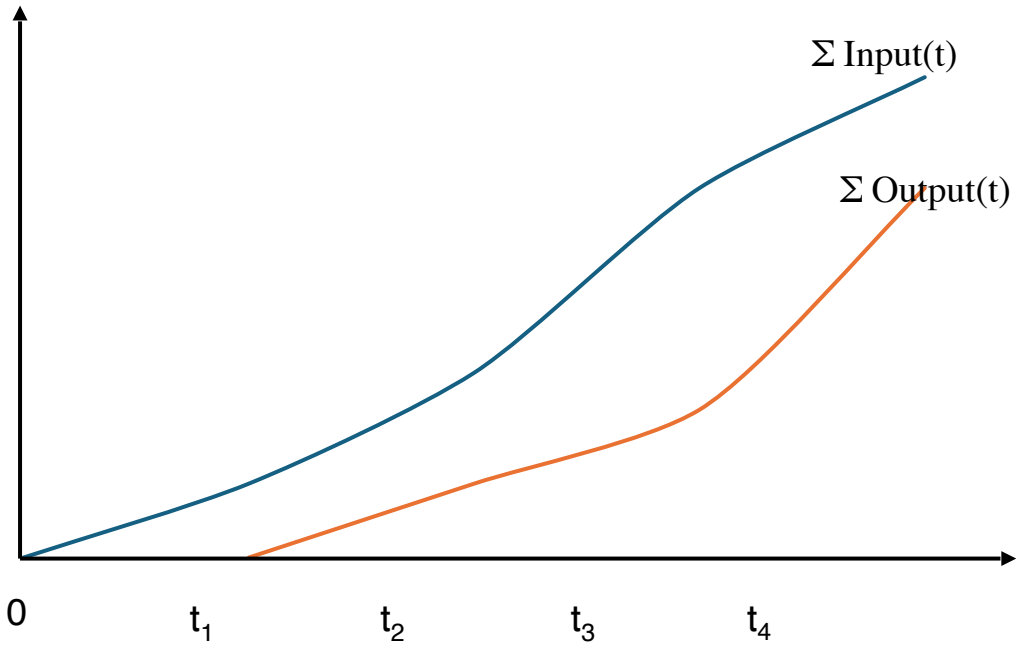
Model Abstraction and Discretization

Model Abstraction and Discretization

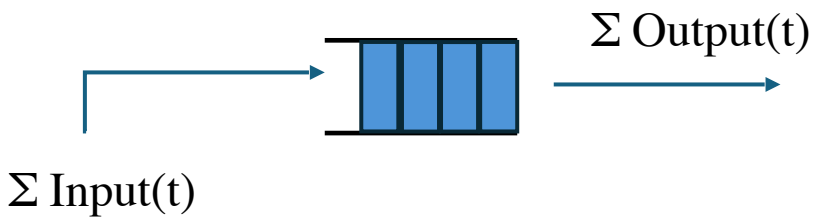


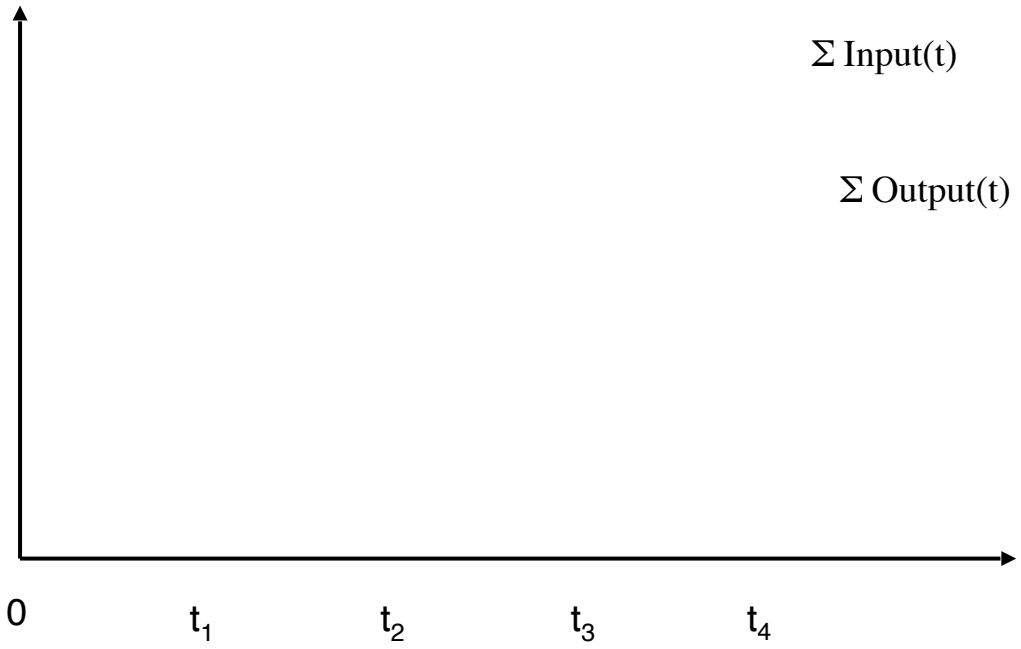
Model Abstraction and Discretization



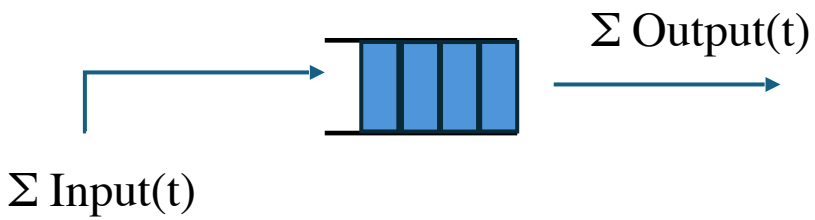


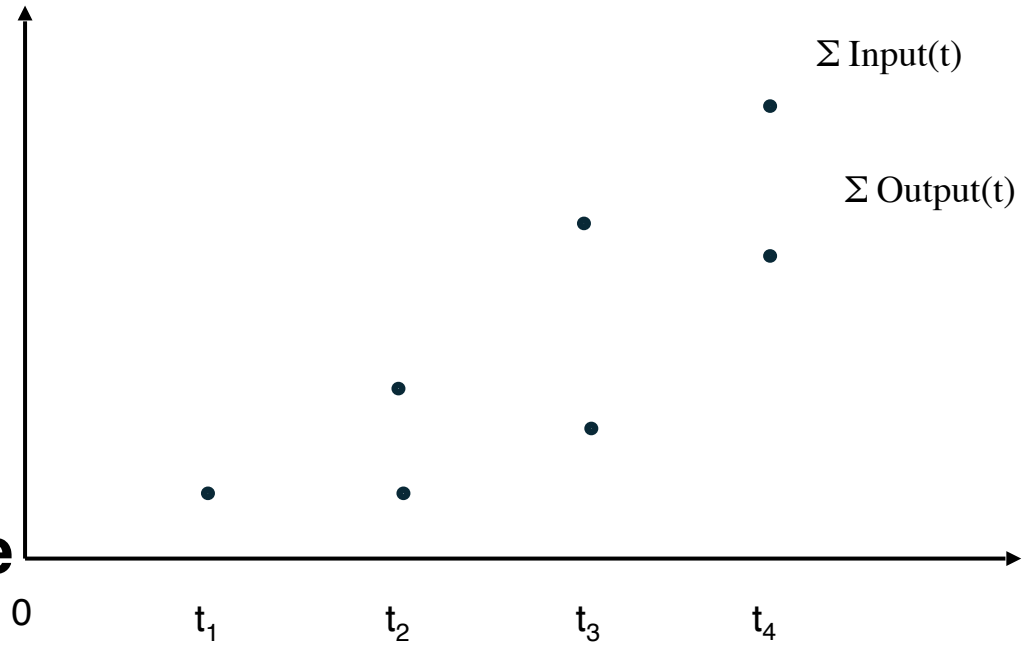
Model Abstraction and Discretization





Model Abstraction and Discretization

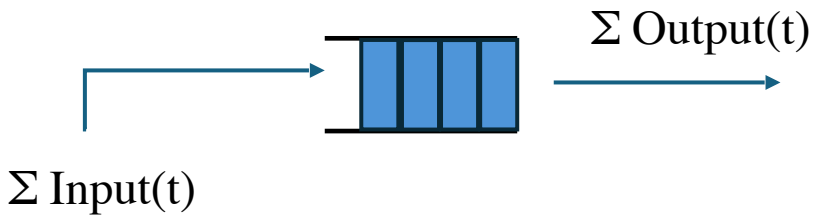


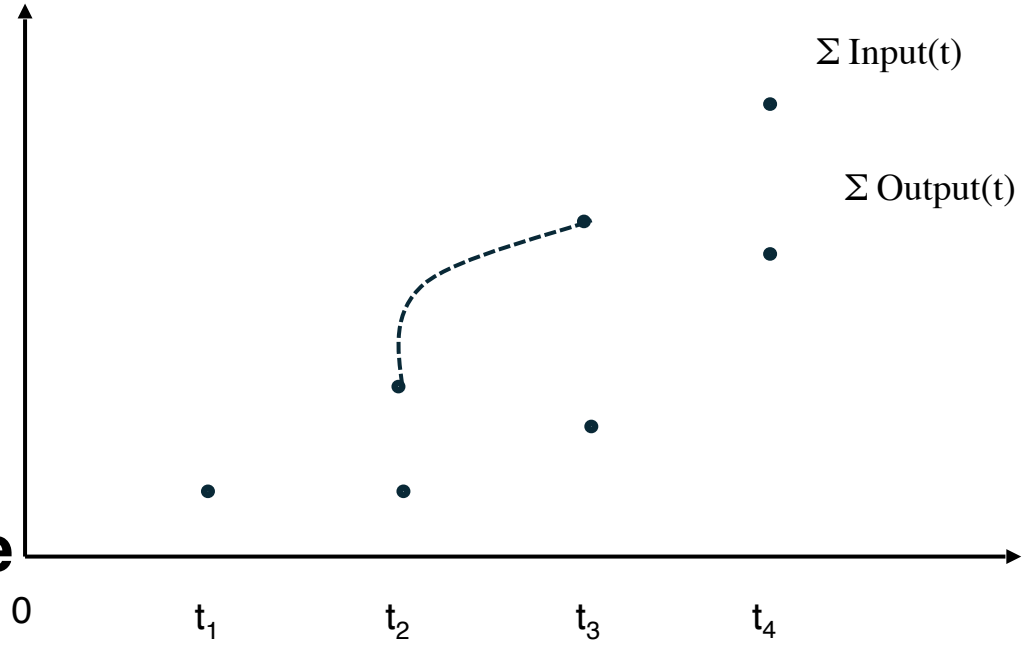


For every periodic timestep t_x

- Define Boolean constraints for **cumulative variables**
- Let solver pick **any valid** trace between two discrete timesteps

Model Abstraction and Discretization

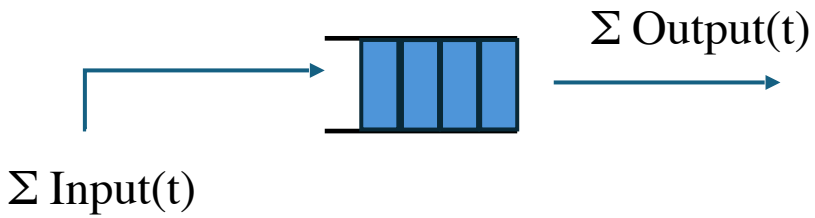


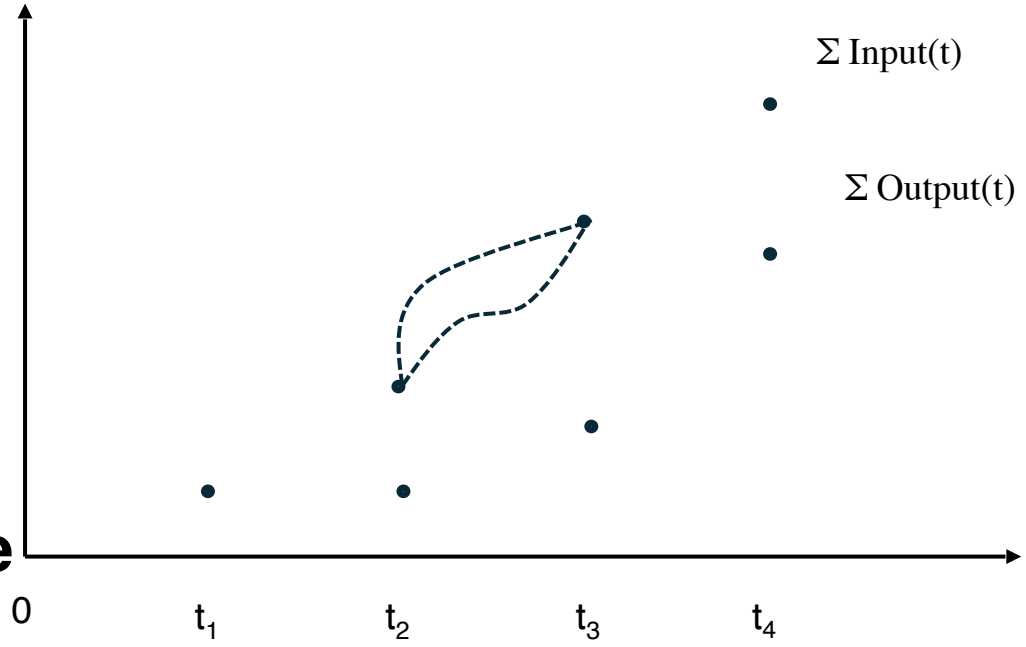


For every periodic timestep t_x

- Define Boolean constraints for **cumulative variables**
- Let solver pick **any valid** trace between two discrete timesteps

Model Abstraction and Discretization

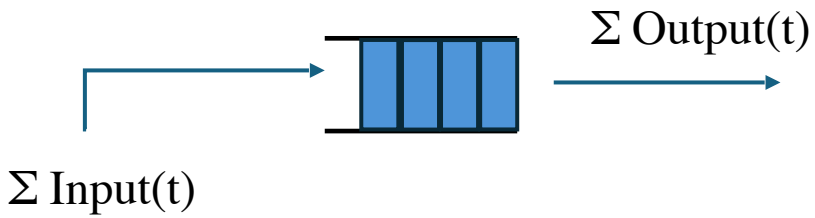


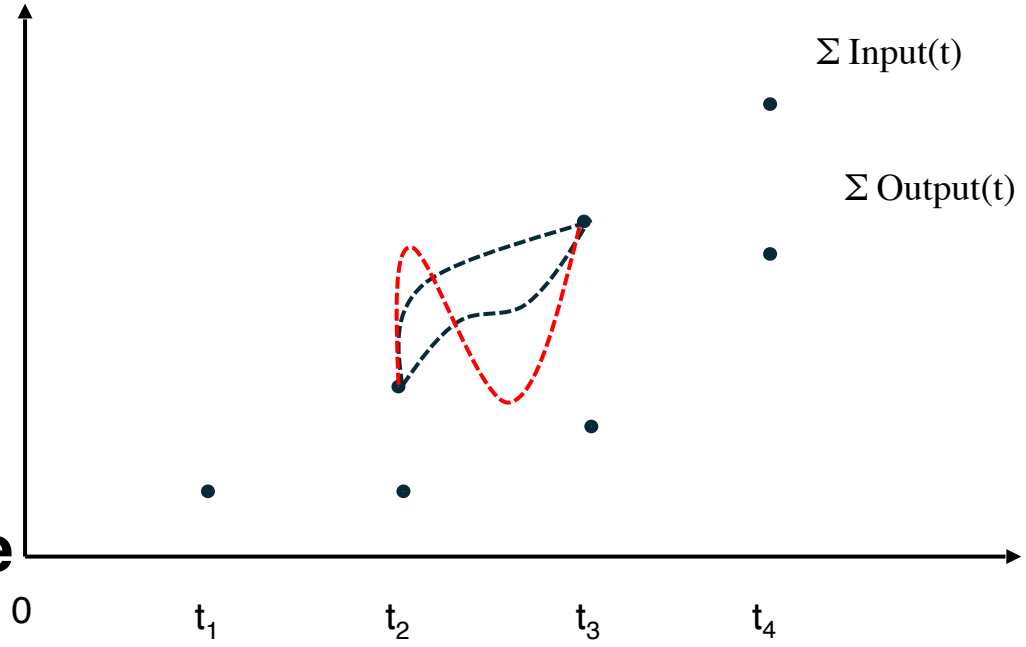


For every periodic timestep t_x

- Define Boolean constraints for **cumulative variables**
- Let solver pick **any valid** trace between two discrete timesteps

Model Abstraction and Discretization

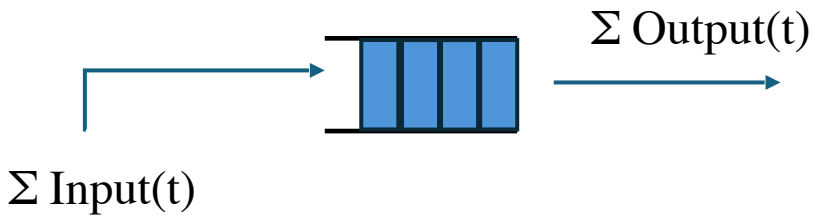


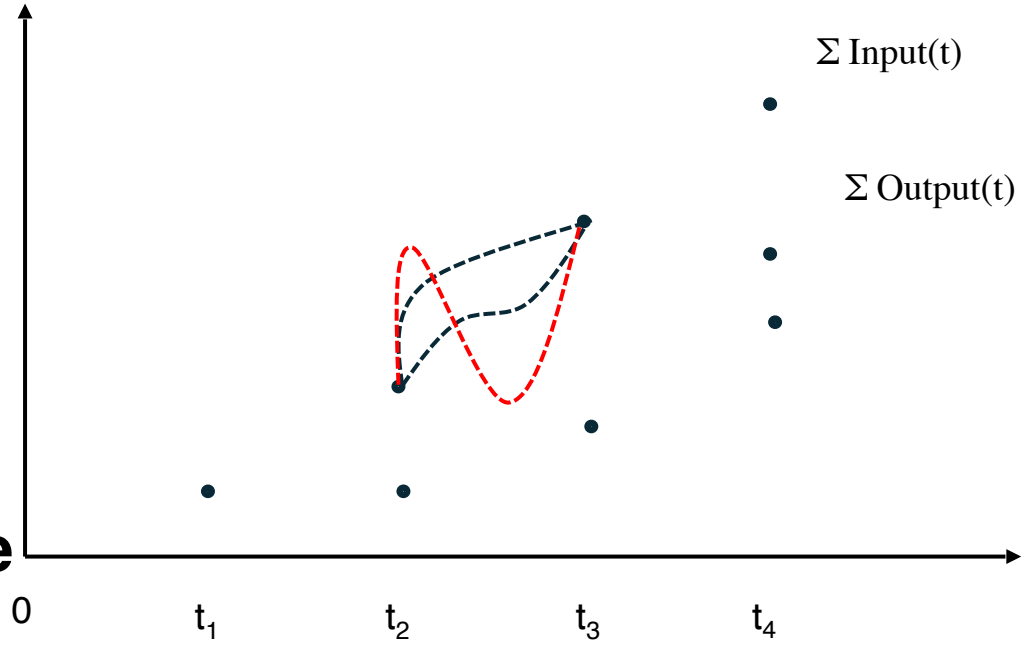


For every periodic timestep t_x

- Define Boolean constraints for **cumulative variables**
- Let solver pick **any valid** trace between two discrete timesteps

Model Abstraction and Discretization

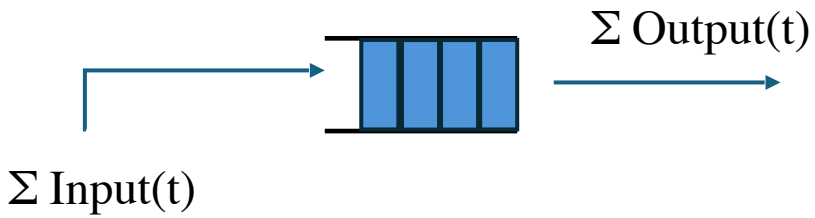


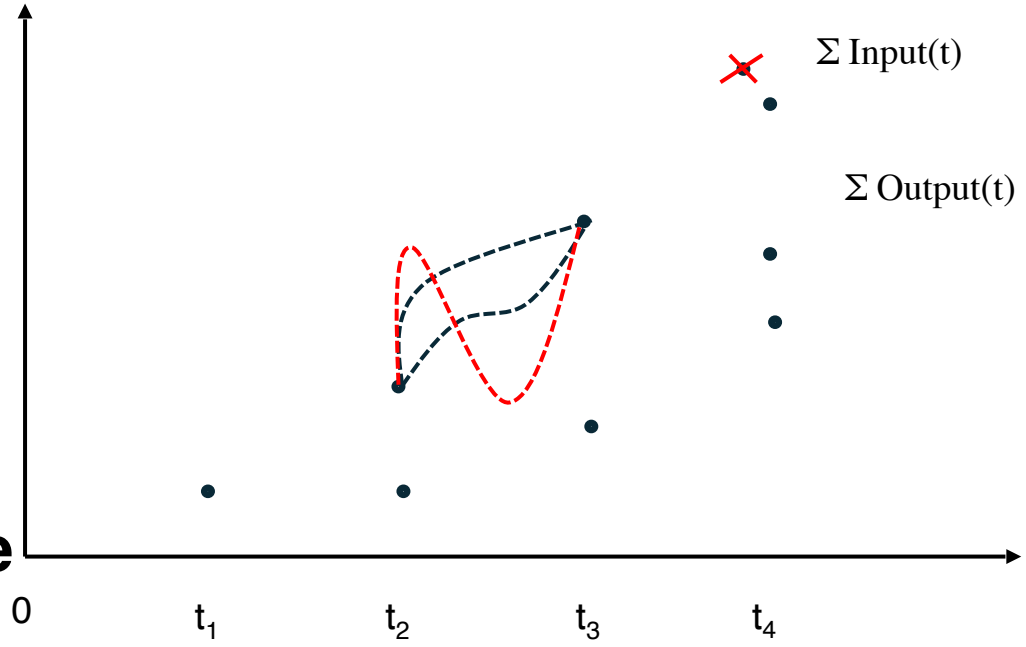


For every periodic timestep t_x

- Define Boolean constraints for **cumulative variables**
- Let solver pick **any valid** trace between two discrete timesteps

Model Abstraction and Discretization





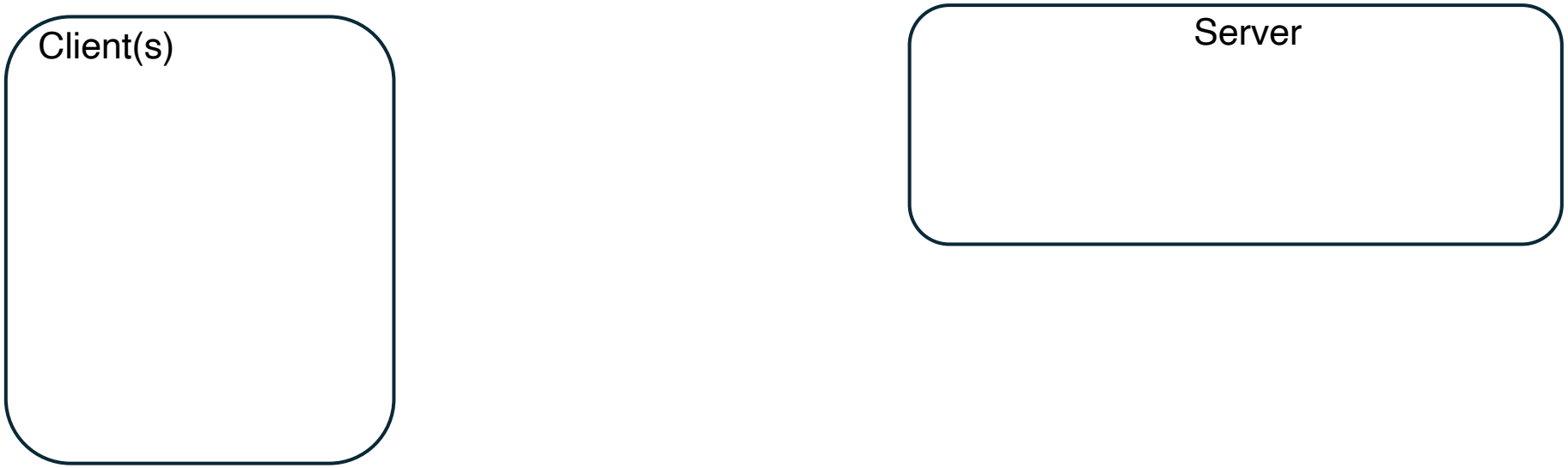
For every periodic timestep t_x

- Define Boolean constraints for **cumulative variables**
- Let solver pick **any valid** trace between two discrete timesteps

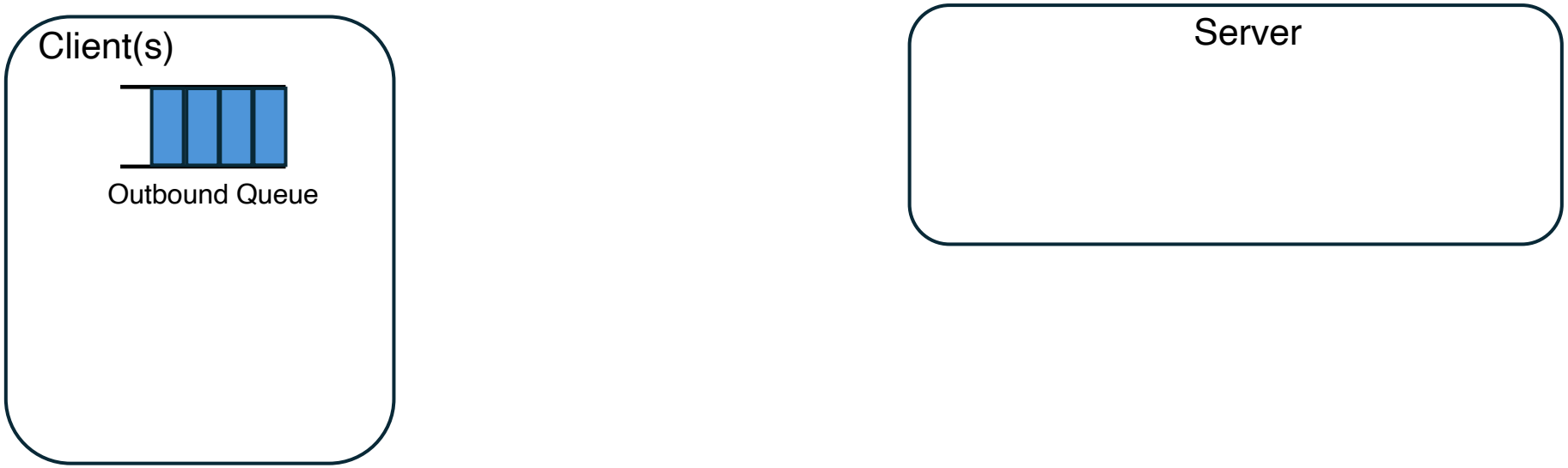
$$\text{Input}(t_x) \geq \text{Output}(t_x)$$

System of Network Queues

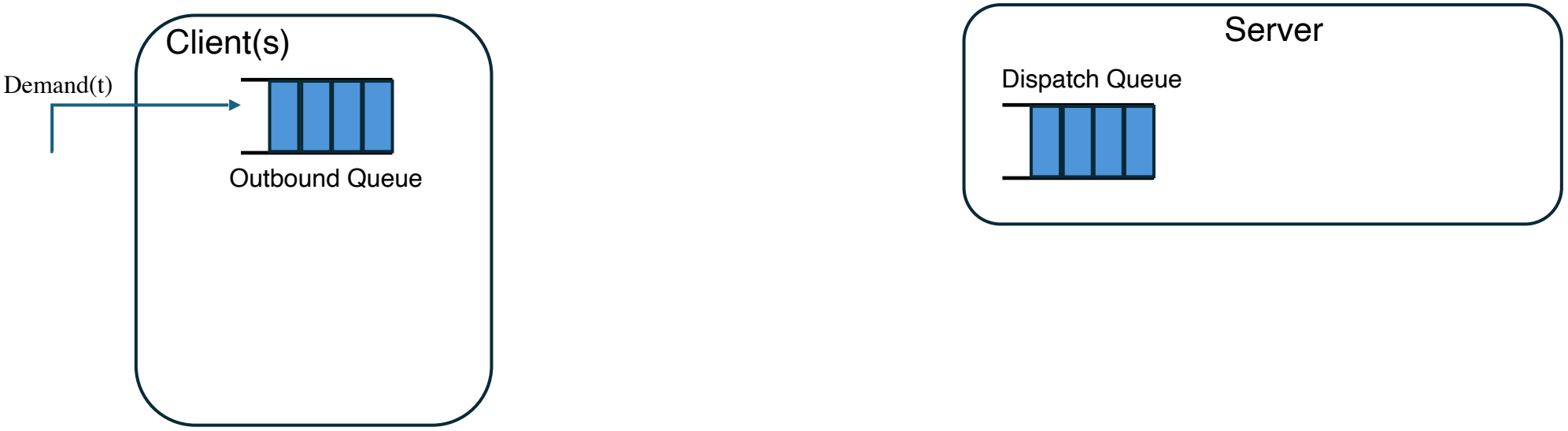
System of Network Queues



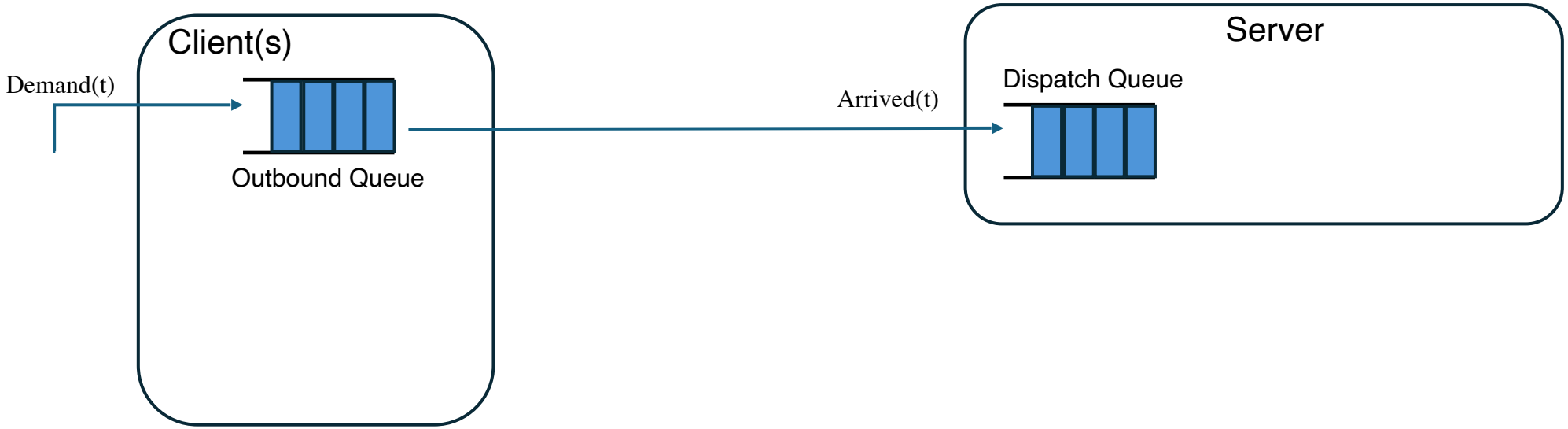
System of Network Queues



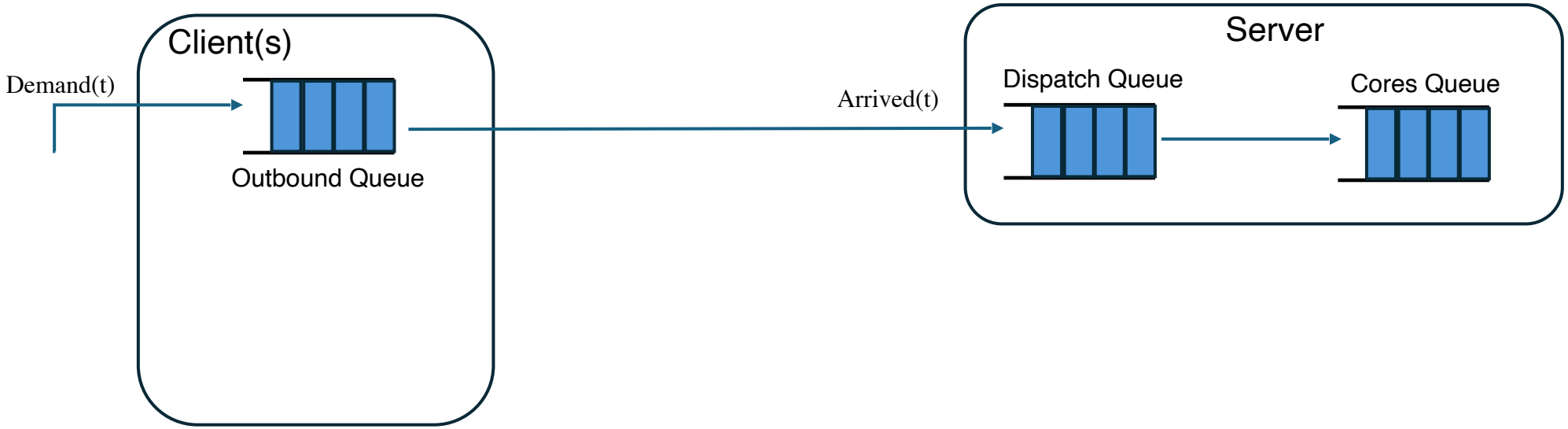
System of Network Queues



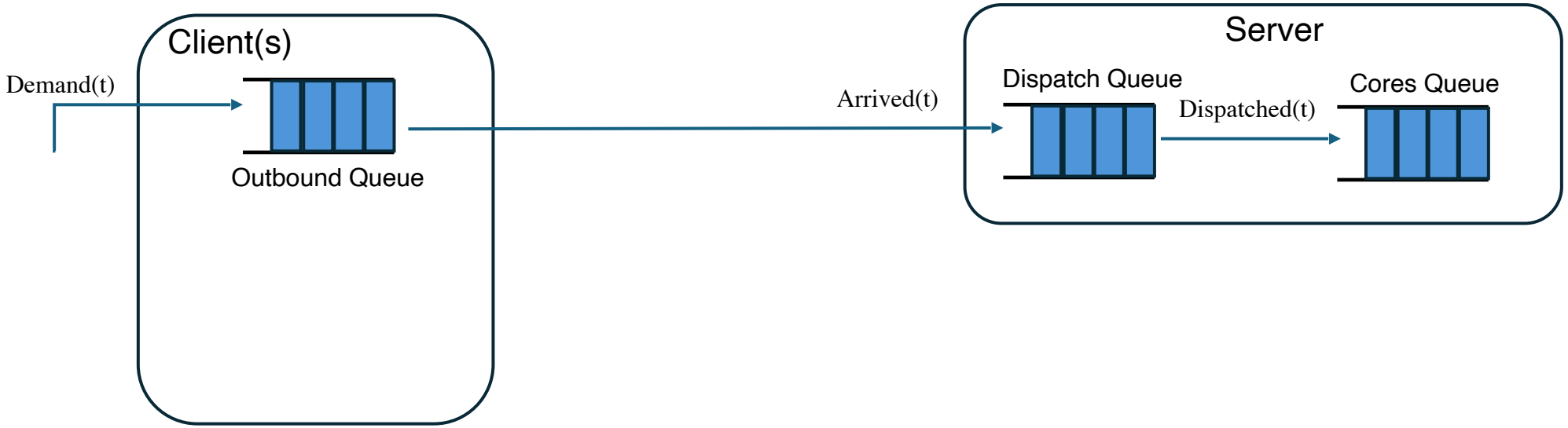
System of Network Queues



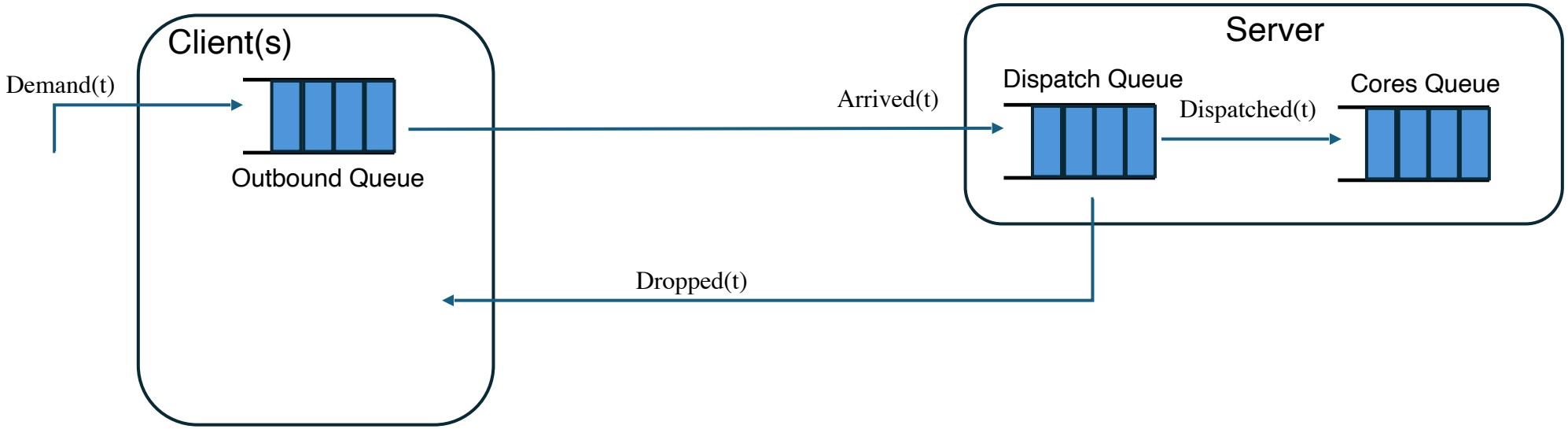
System of Network Queues



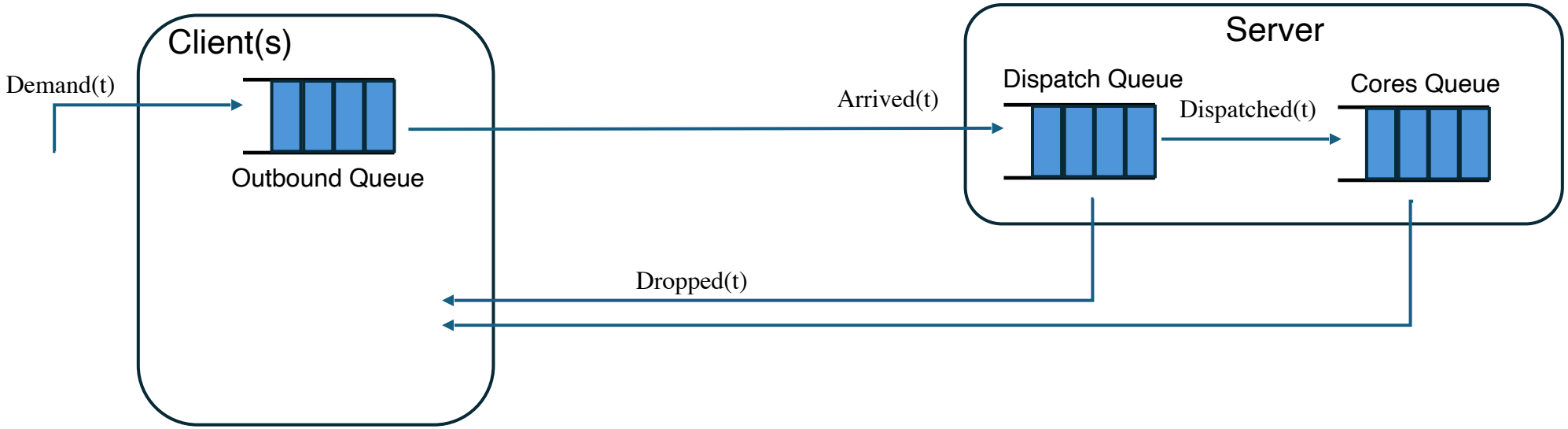
System of Network Queues



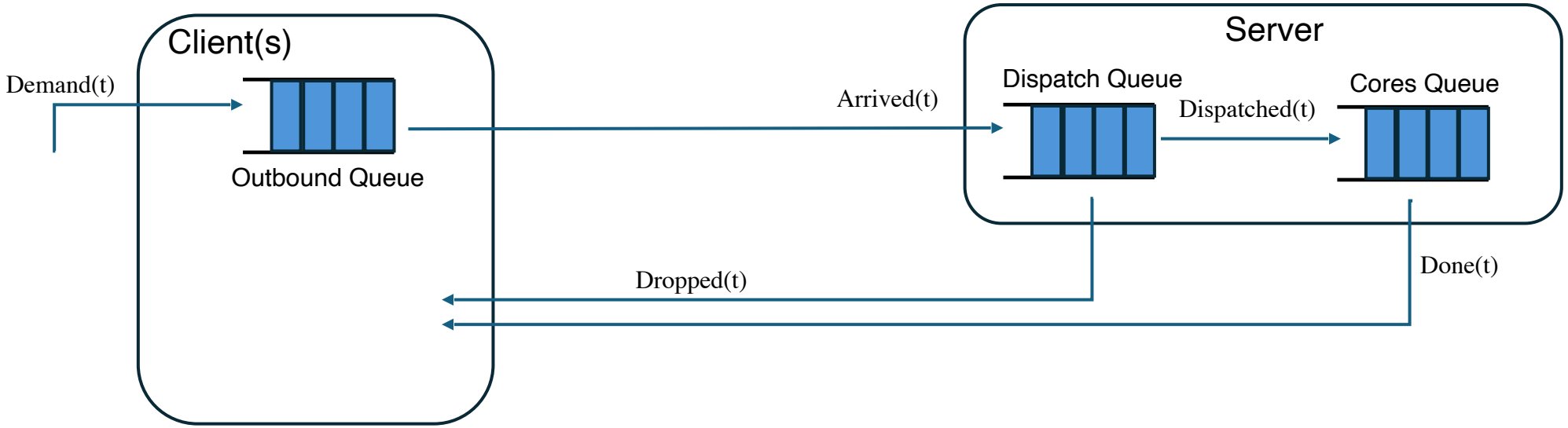
System of Network Queues



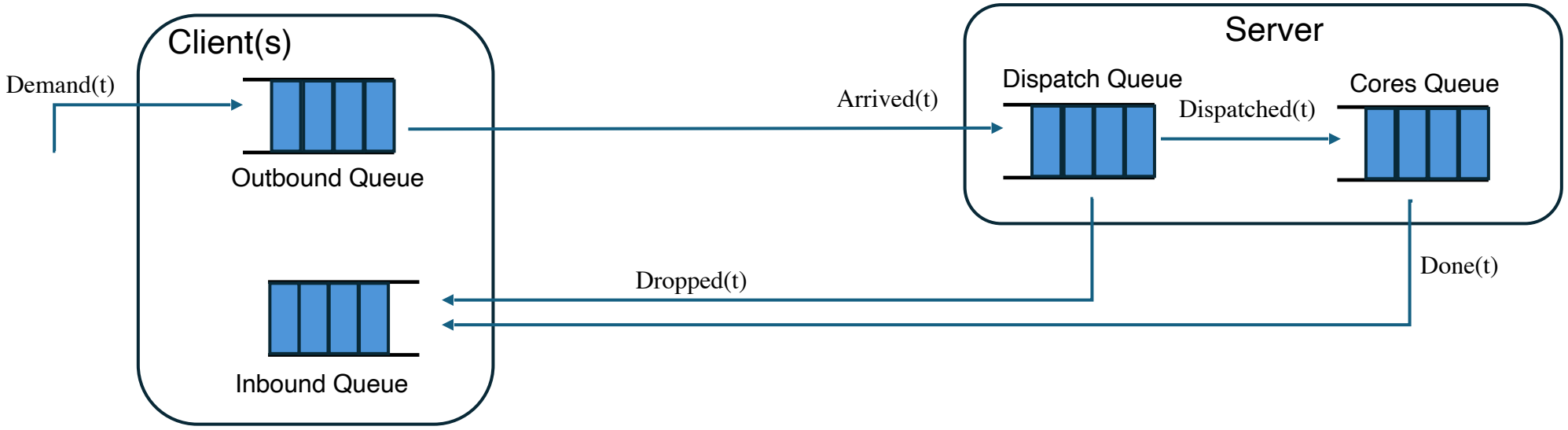
System of Network Queues



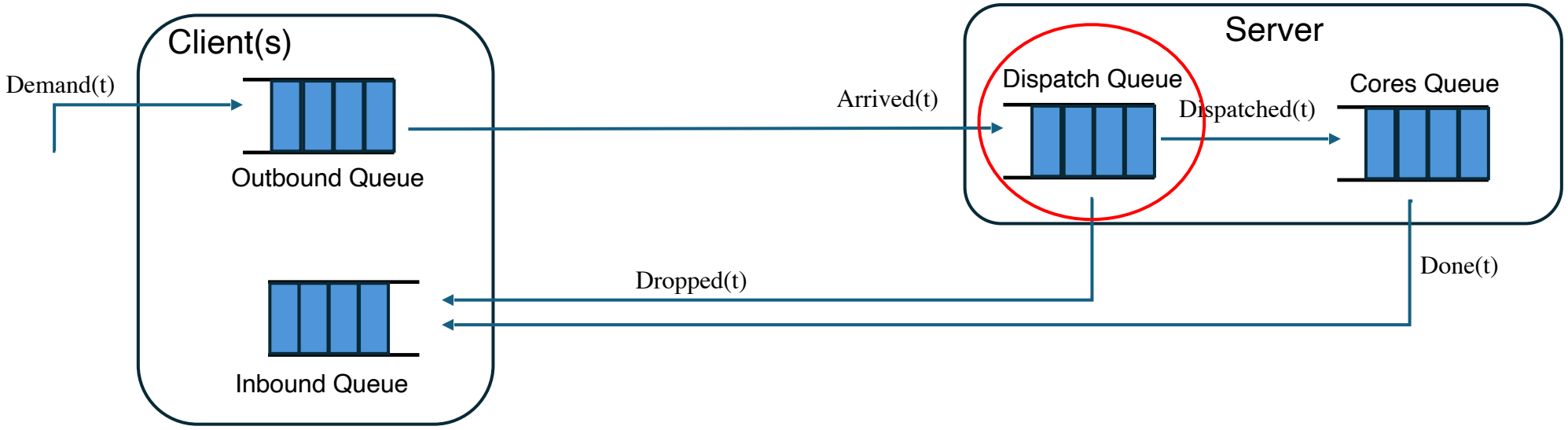
System of Network Queues



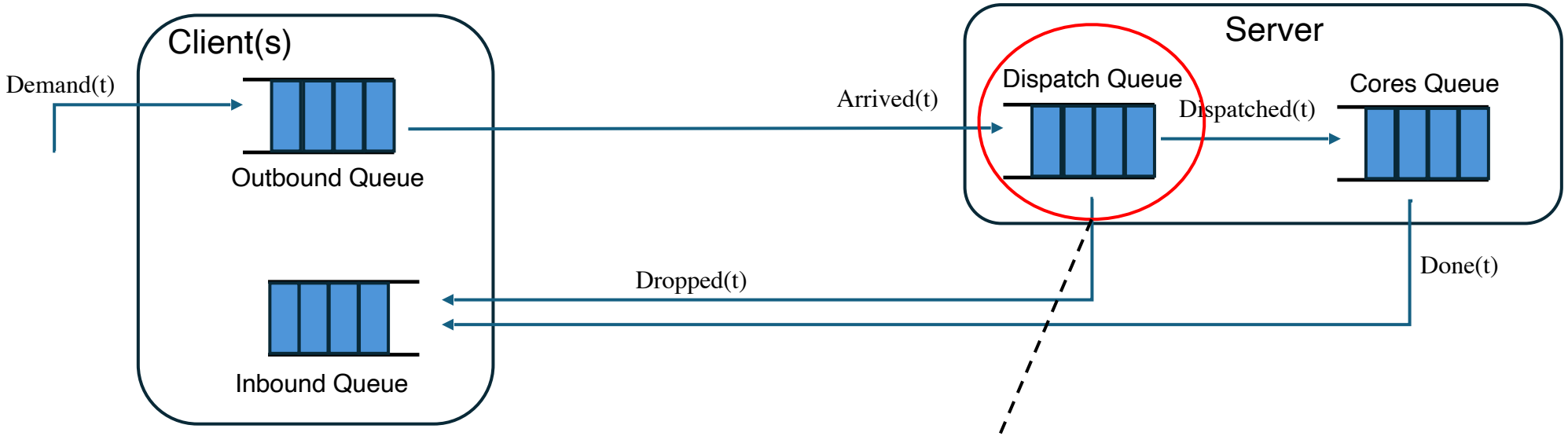
System of Network Queues



System of Network Queues



System of Network Queues

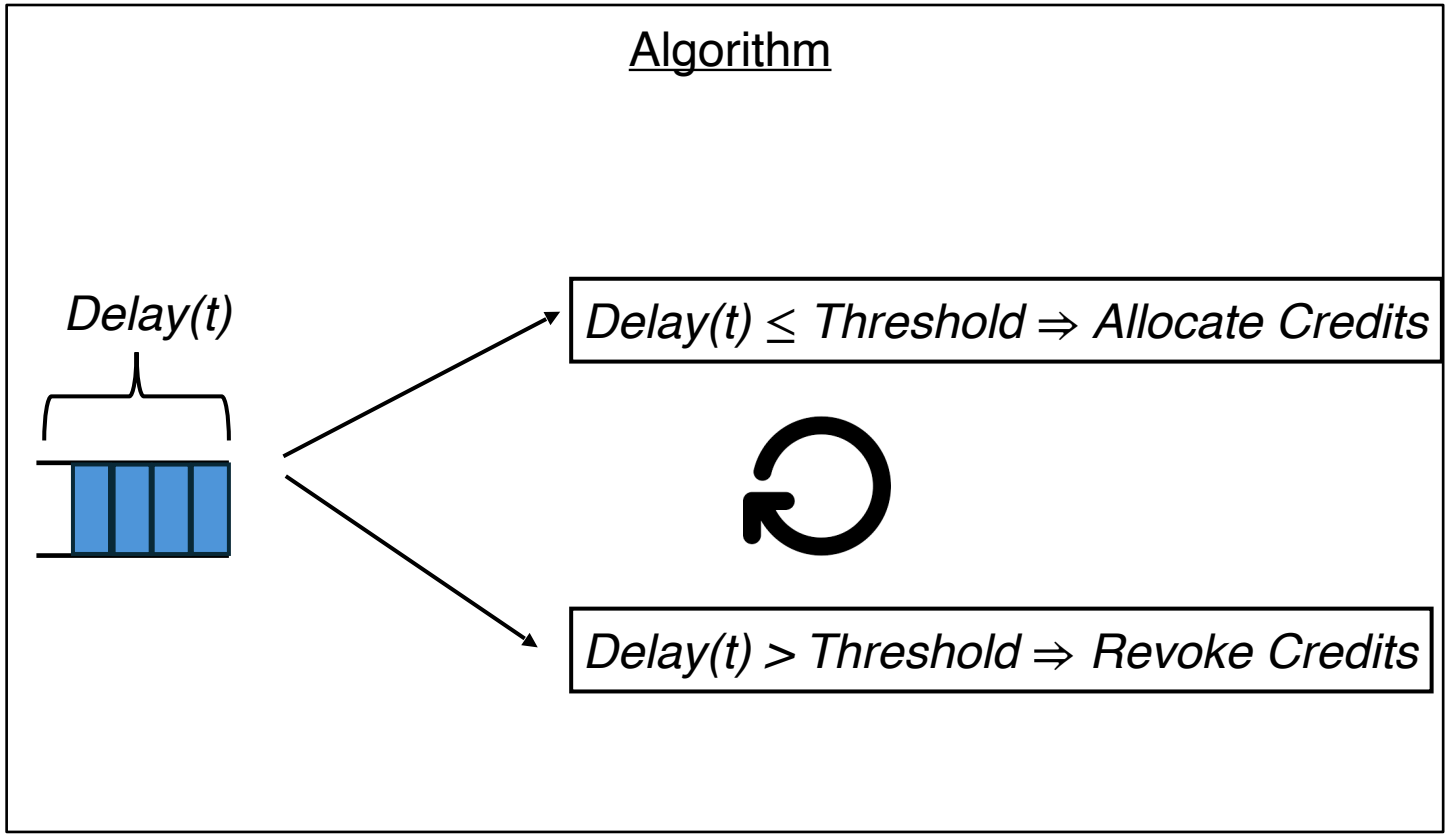


CoDel-like AQM Constraint on Dispatch Queue

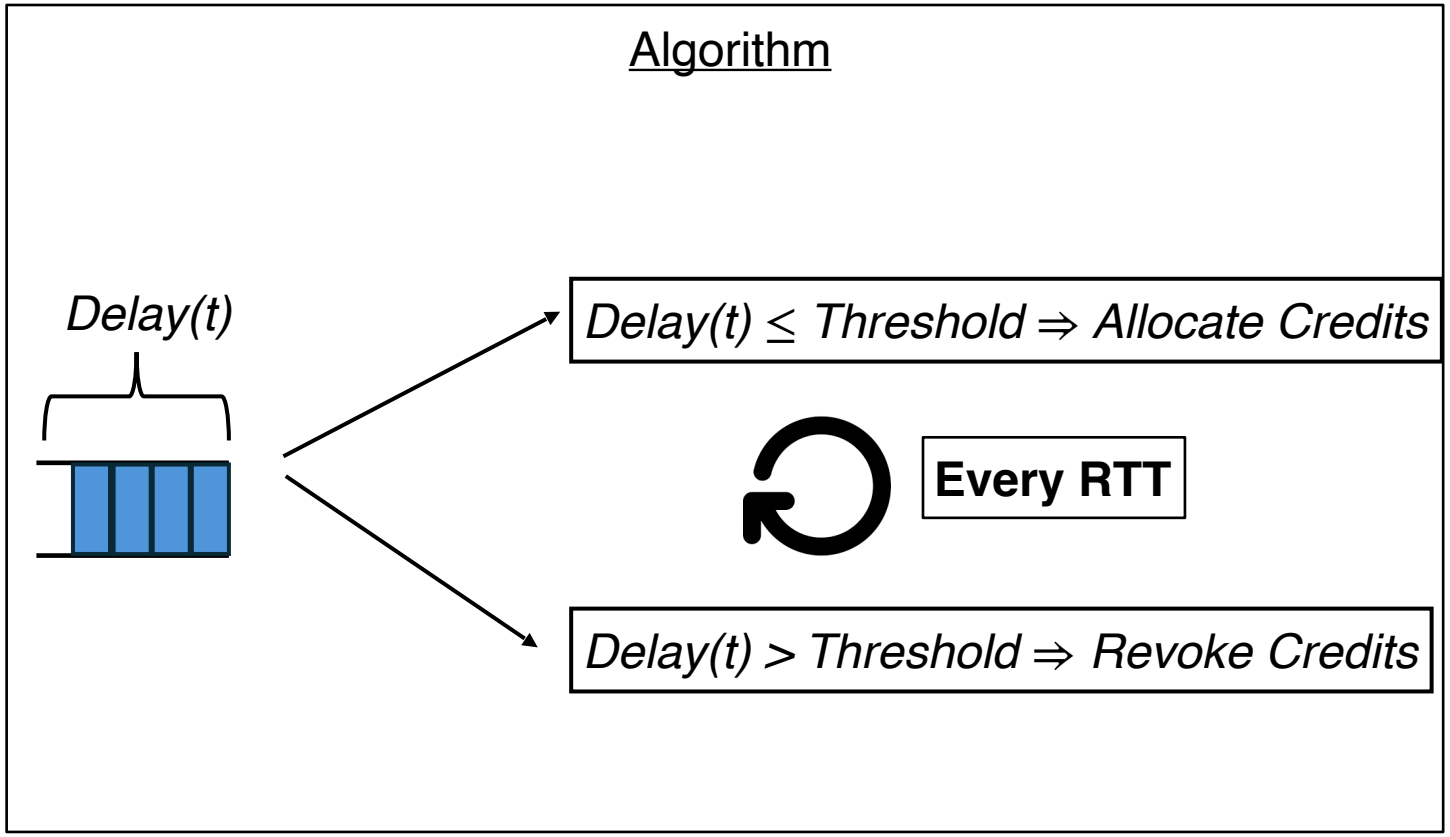
$$\frac{d}{dt}(\text{arrived(t)}) > 0 \bigwedge \frac{d}{dt}(\text{dropped(t)}) > 0 \Rightarrow \text{delay[t]} > \boldsymbol{Threshold}$$

Overload Control: **Breakwater**

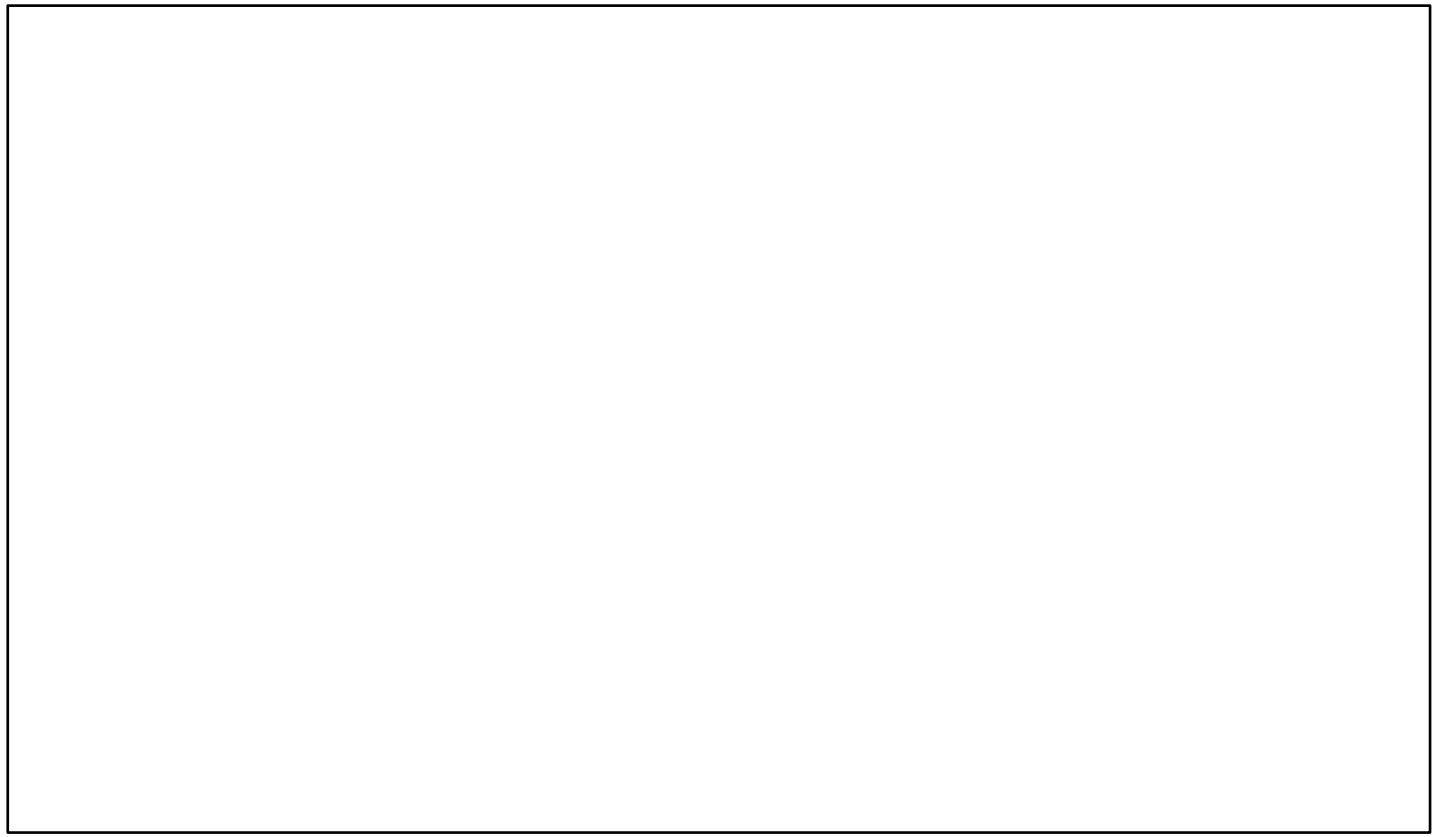
Overload Control: **Breakwater**



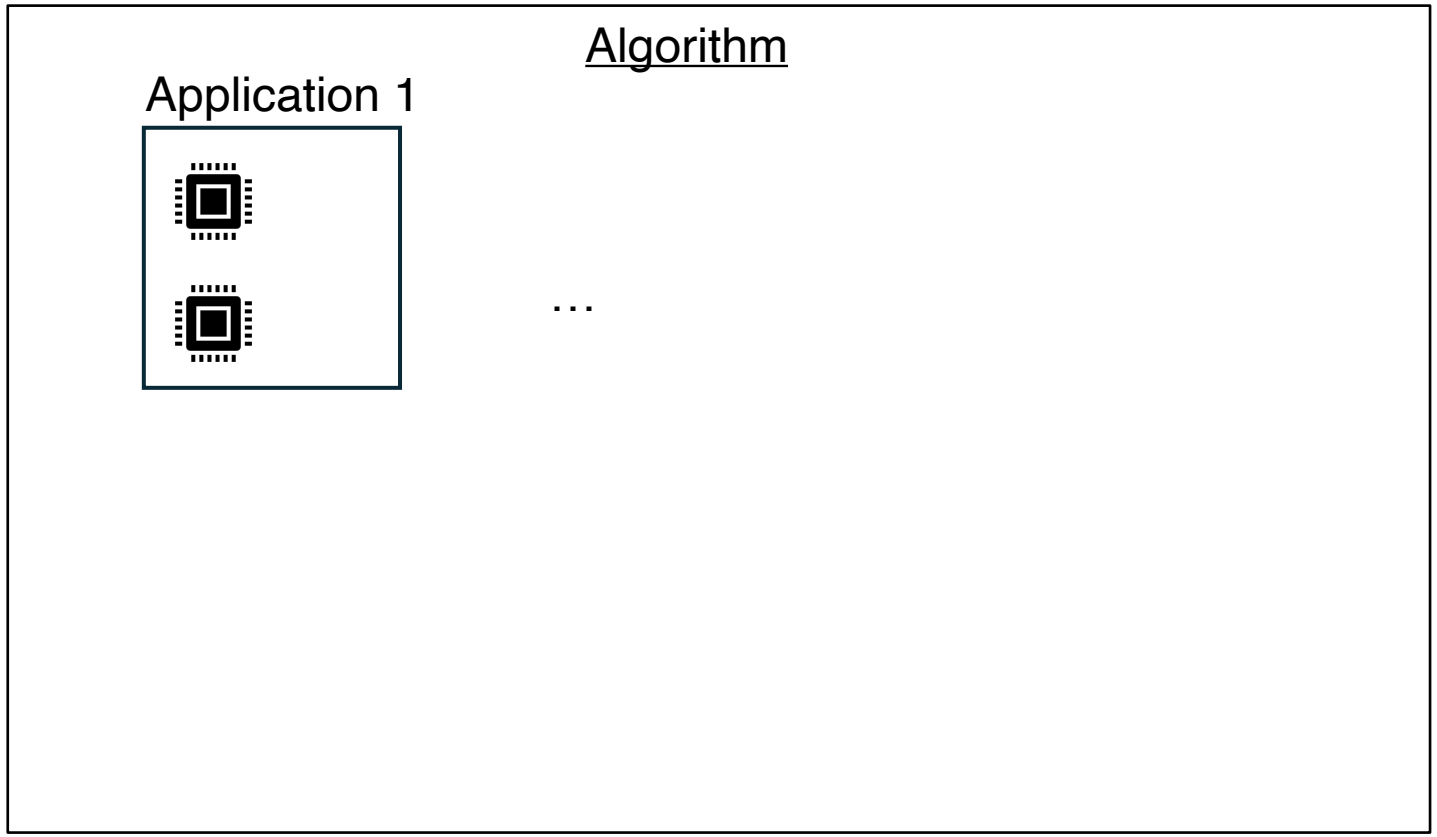
Overload Control: **Breakwater**



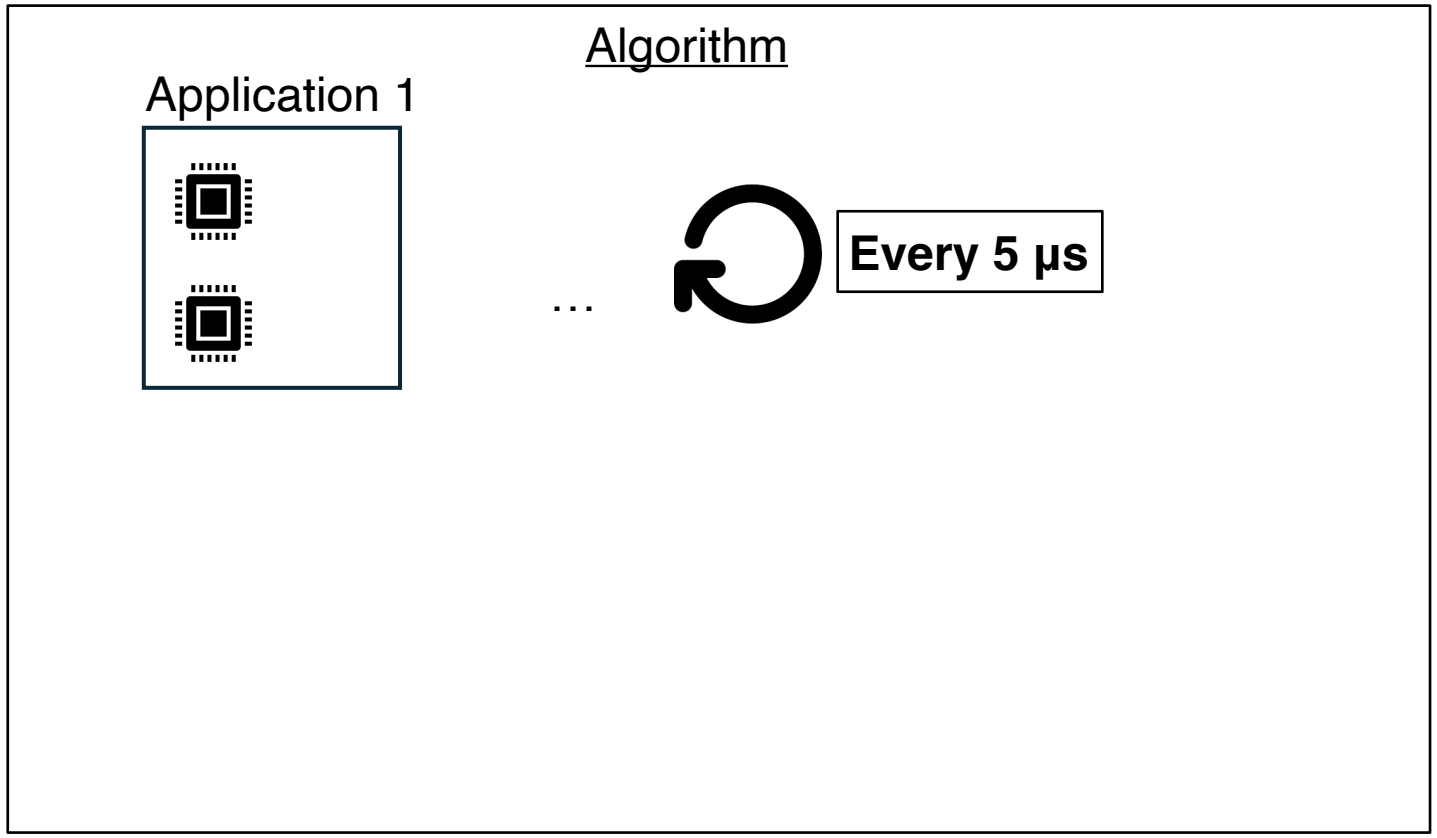
Core Allocation: **Shenango**



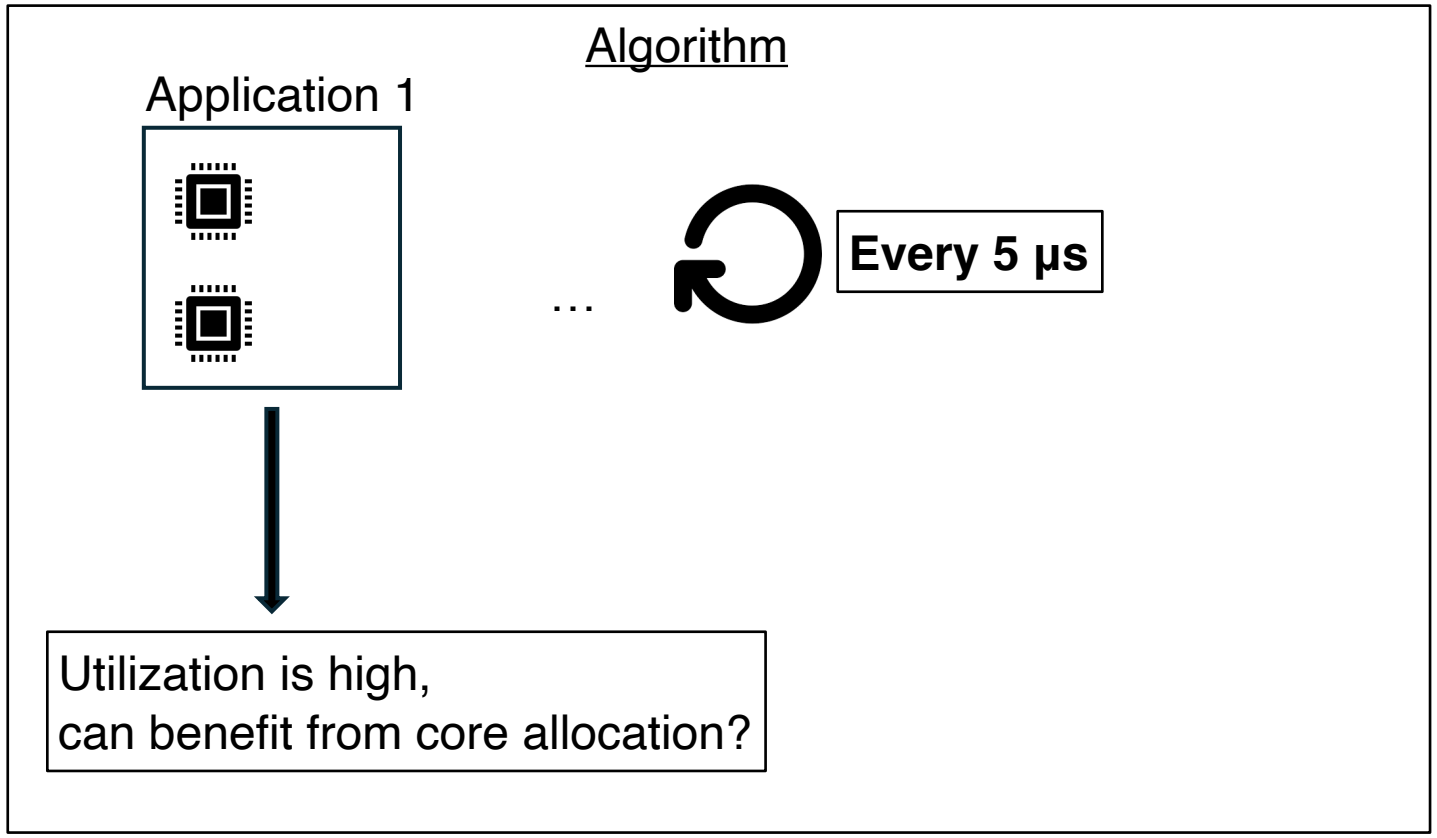
Core Allocation: **Shenango**



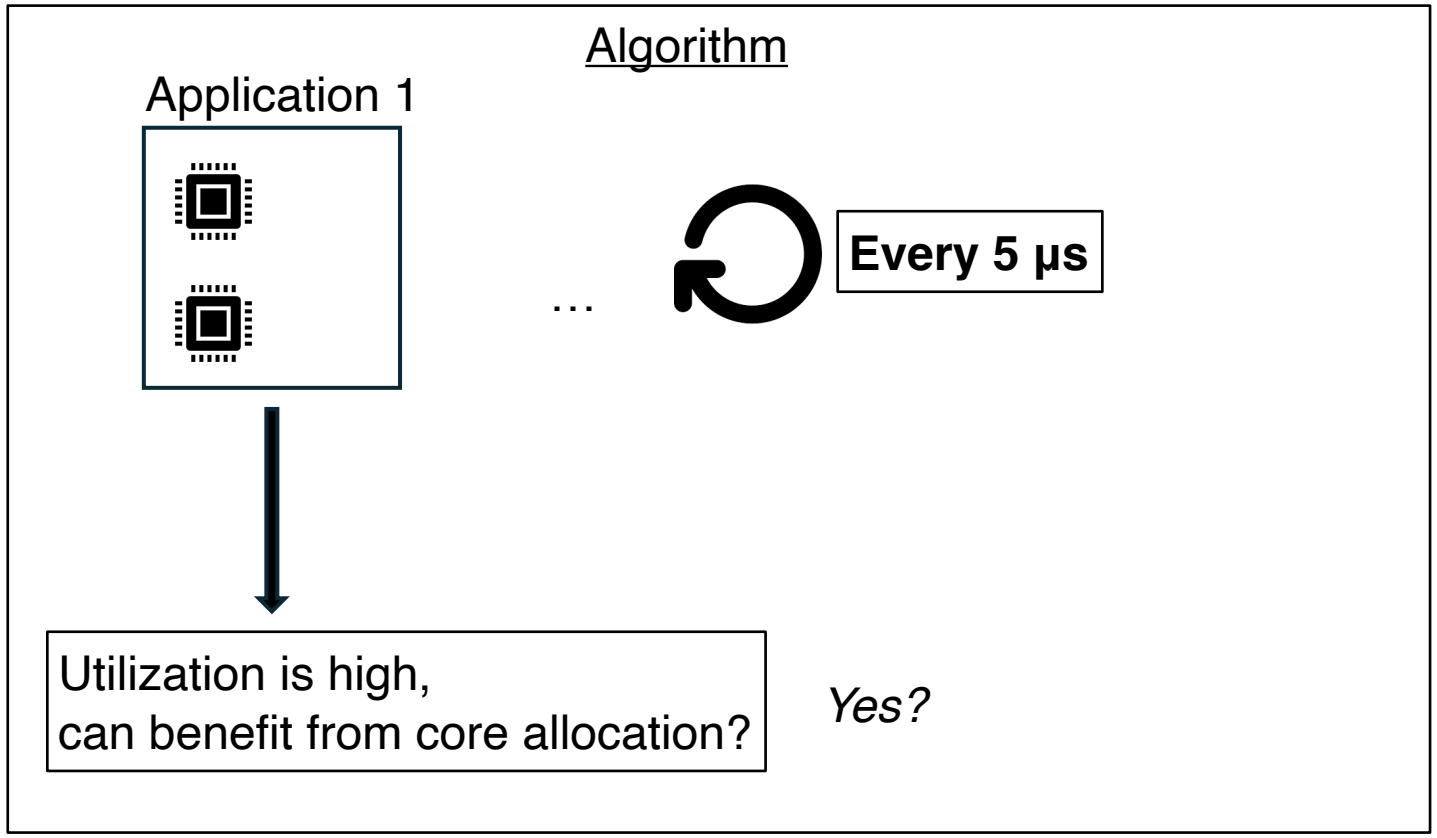
Core Allocation: **Shenango**



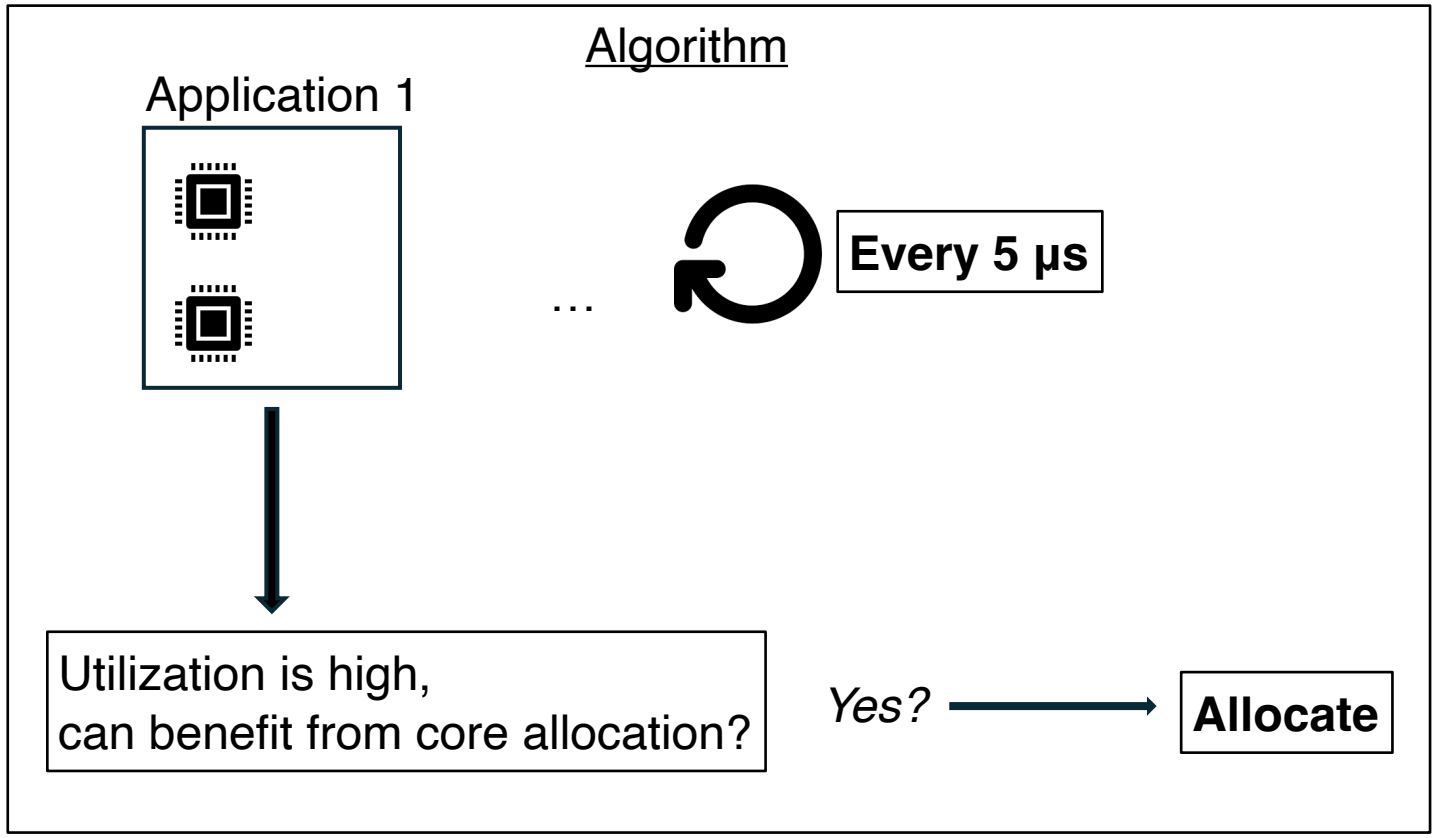
Core Allocation: **Shenango**



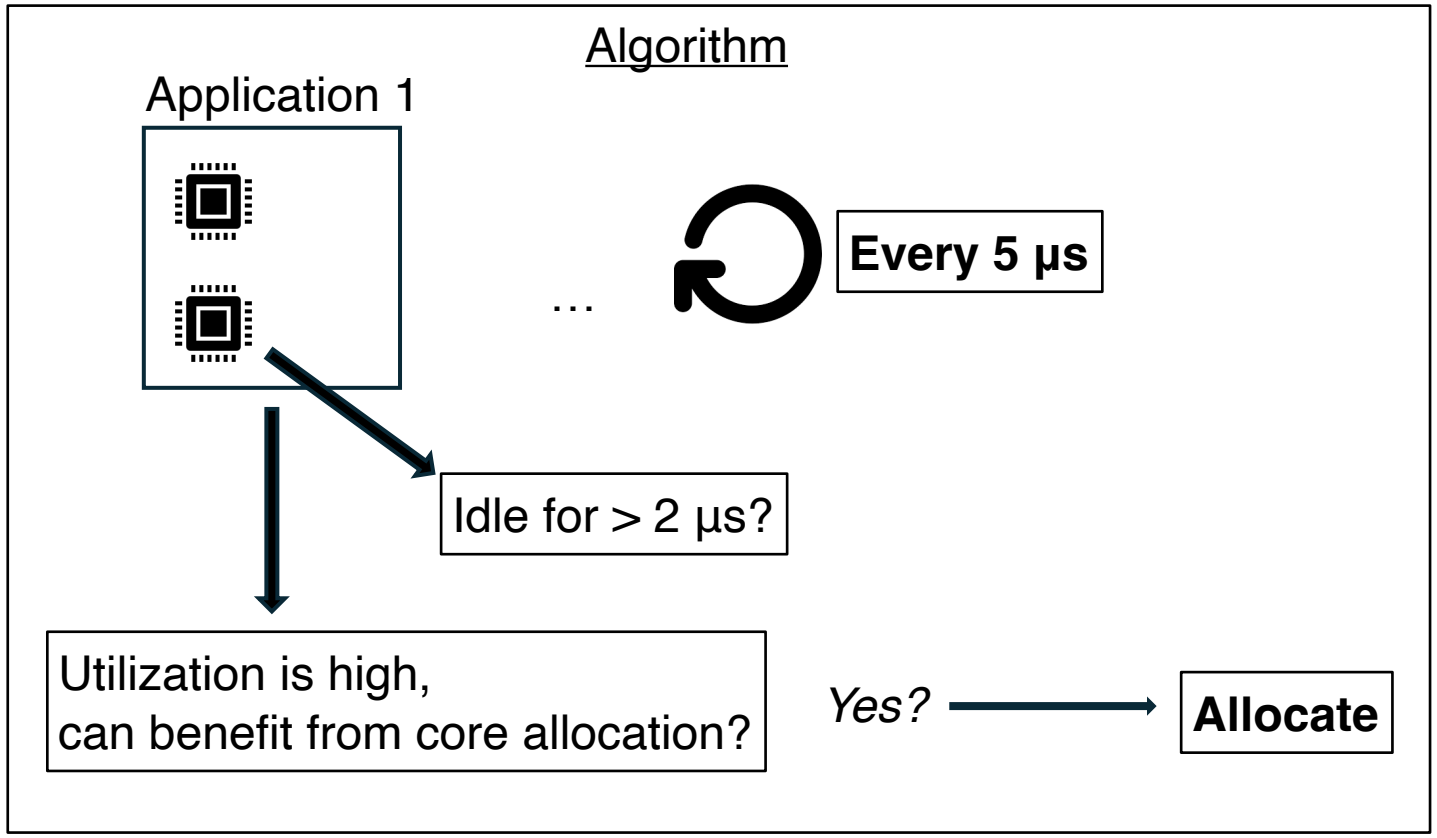
Core Allocation: **Shenango**



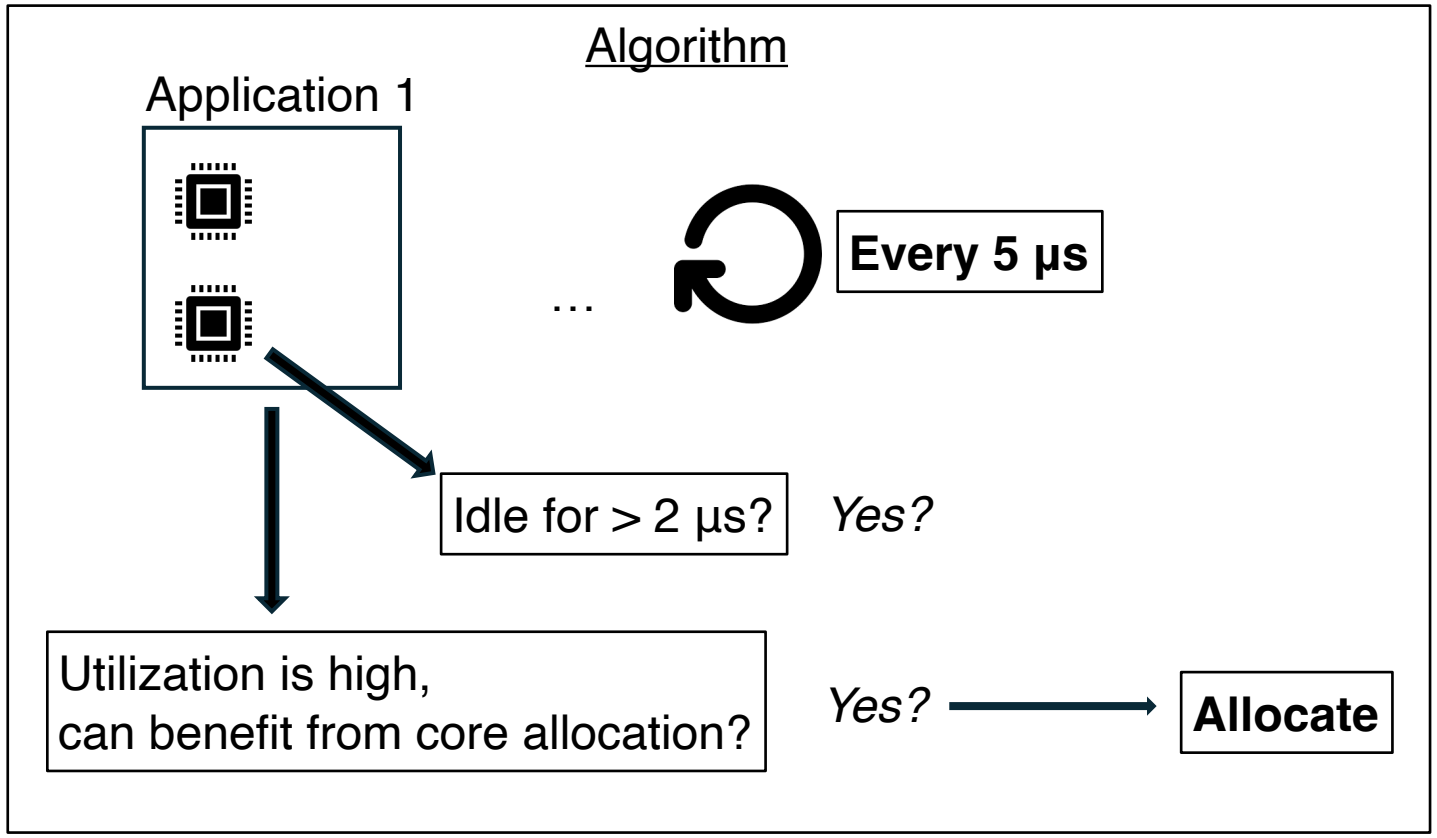
Core Allocation: **Shenango**



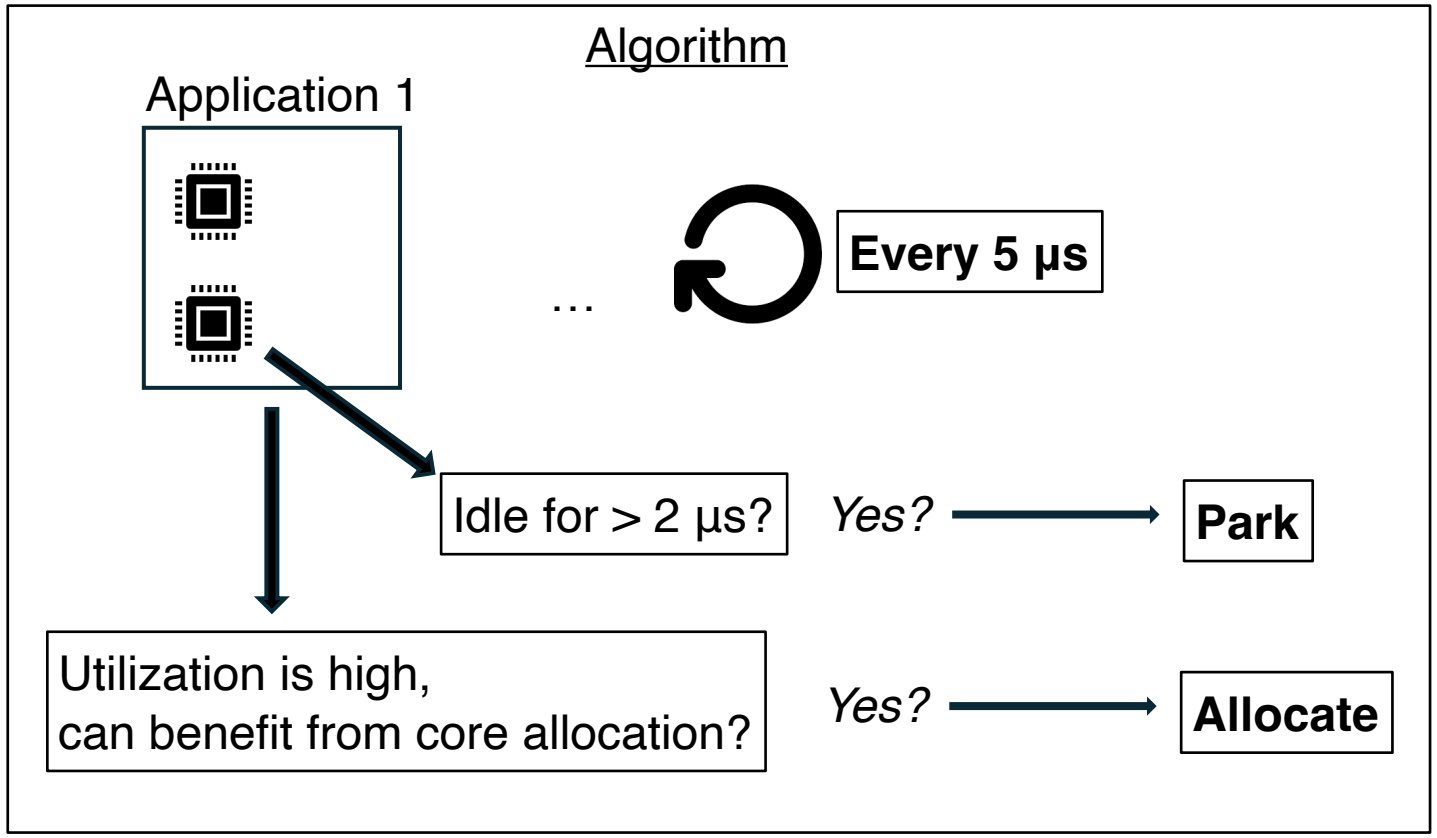
Core Allocation: **Shenango**



Core Allocation: **Shenango**



Core Allocation: **Shenango**



Methodology – Algorithms and Configuration

Methodology – Algorithms and Configuration

Compare 2 policies

- **Static + Breakwater:** Maximum cores are statically allocated for the application
- **Shenango + Breakwater:** Cores dynamically allocated based on Shenango's policy

Methodology – Algorithms and Configuration

Compare 2 policies

- **Static + Breakwater:** Maximum cores are statically allocated for the application
- **Shenango + Breakwater:** Cores dynamically allocated based on Shenango's policy

Basic Evaluation Configuration

- **Trace Duration:** 200 microseconds
- **RTT:** 20 microseconds
- **Maximum cores available for app:** 20
- **Service time of requests:** 1microsecond to 5 microseconds

Methodology - Performance Queries

Methodology - Performance Queries

Throughput Query: Given the load on a server, can **x** requests or lower be done over the entire trace **T**?

Methodology - Performance Queries

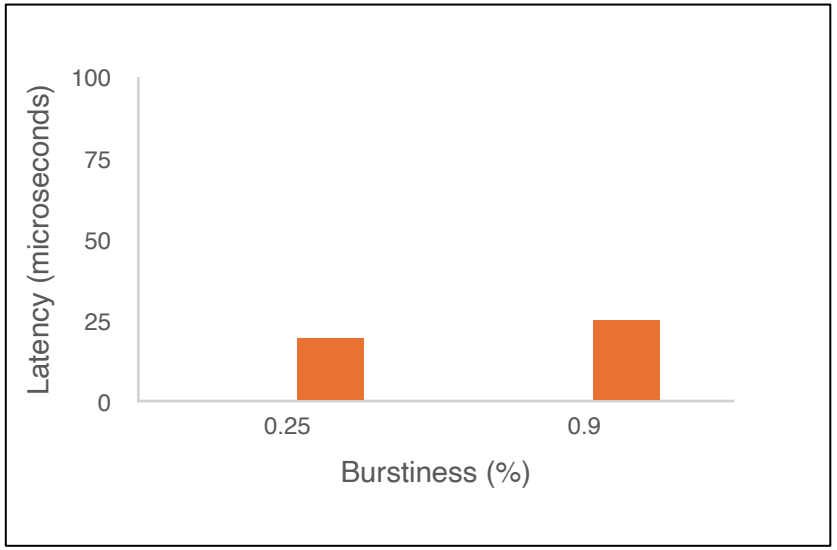
Throughput Query: Given the load on a server, can **x** requests or lower be done over the entire trace **T**?

Latency Query: Given the load on a server, can the maximum delay over the entire trace **T** be **y** microseconds or higher?

Latency Results

Latency Results

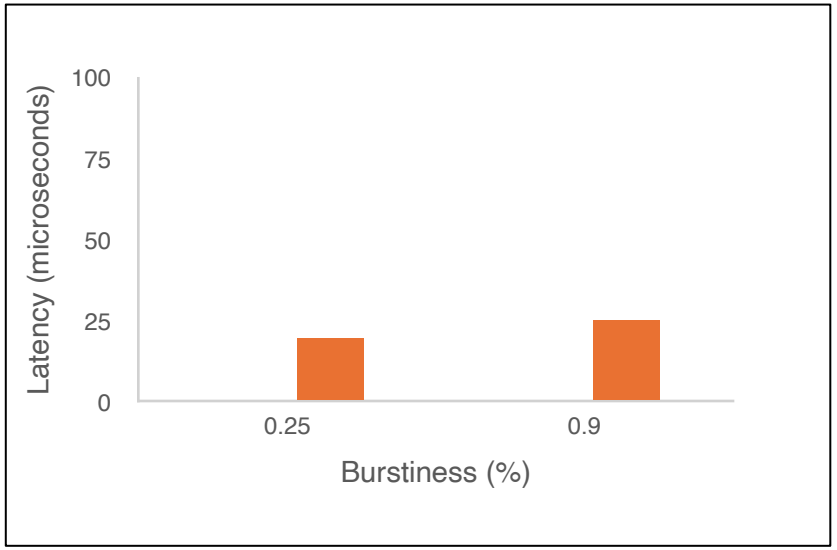
Latency Results

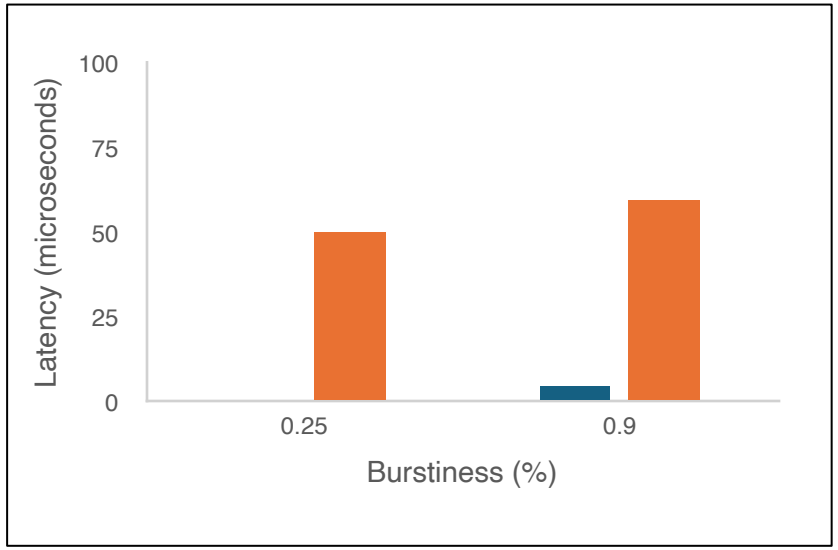


Low Load

Static+Breakwater Shenango+Breakwater

Latency Results



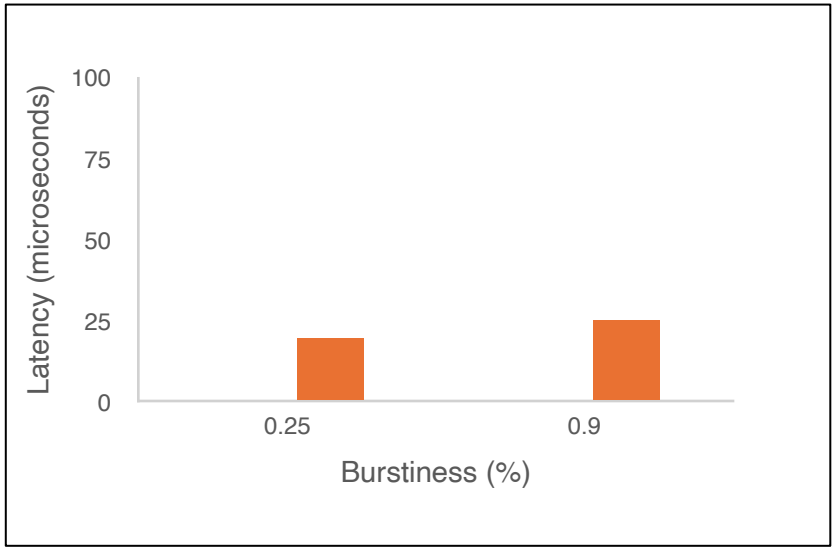


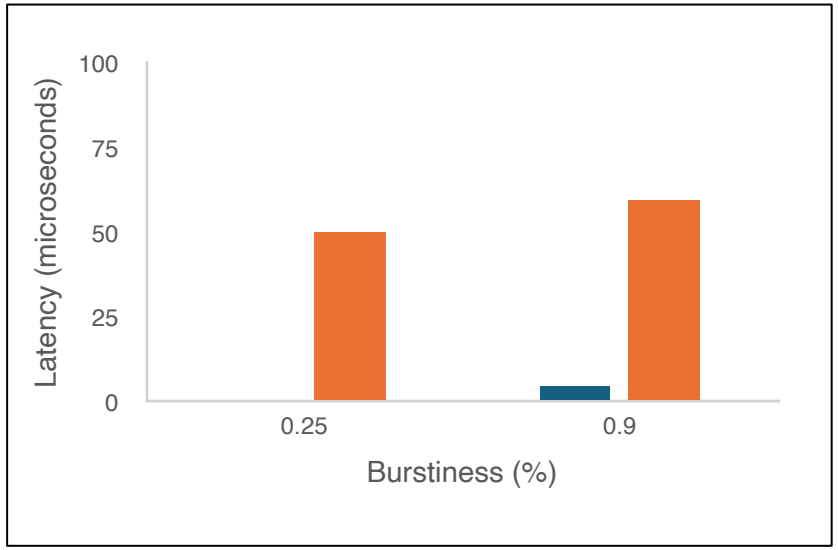
Low Load

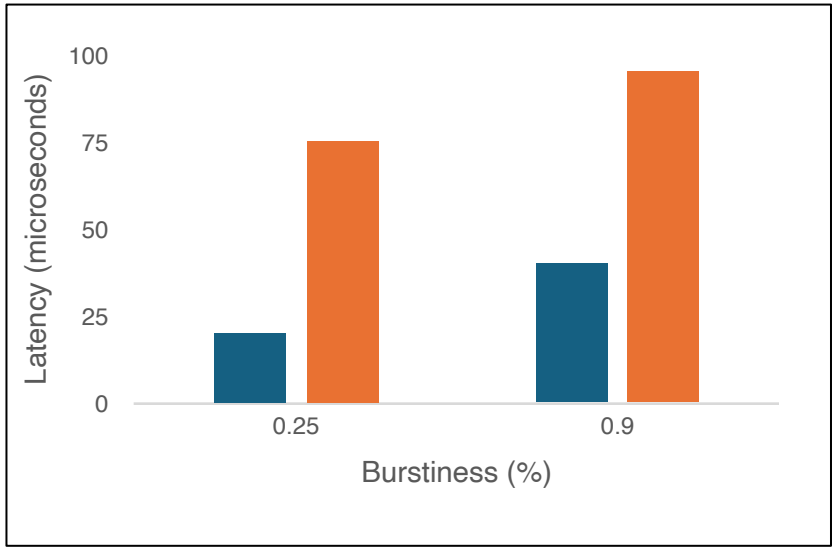
High Load

Static+Breakwater Shenango+Breakwater

Latency Results







Low Load

High Load

Overload

Static+Breakwater Shenango+Breakwater

Validation of Results

Validation of Results

- Use simulation with similar configurations to find traces that exhibit non-trivial worst-case behavior

Validation of Results

- Use simulation with similar configurations to find traces that exhibit non-trivial worst-case behavior
- Traces eventually found would be hard to get without knowledge of performance guarantees provided by our tool

Validation of Results

- Use simulation with similar configurations to find traces that exhibit non-trivial worst-case behavior
- Traces eventually found would be hard to get without knowledge of performance guarantees provided by our tool

CoreSync: Use partial proportionality to maintain credits available to clients based on number of cores allocated (ICNP 2025)

Conclusion

- Existing models are limited in analyzing performance tradeoffs in multi-controller servers in datacenters.
- We present a modular **performance verification** framework built on an SMT solver to provide non-trivial worst-case performance guarantees.
- Our tool is evaluated on *Shenango*, a core allocation algorithm, and *Breakwater*, an overload control algorithm, for establishing bounds on Throughput and Latency.

Conclusion

- Existing models are limited in analyzing performance tradeoffs in multi-controller servers in datacenters.
- We present a modular **performance verification** framework built on an SMT solver to provide non-trivial worst-case performance guarantees.
- Our tool is evaluated on *Shenango*, a core allocation algorithm, and *Breakwater*, an overload control algorithm, for establishing bounds on Throughput and Latency.

Thank You!