Towards a Redundancy-Aware Network Stack for Data Centers

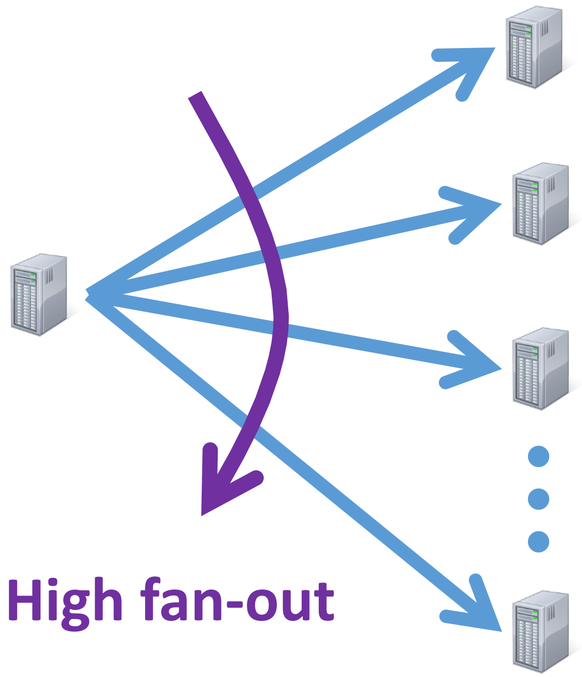
Ali Musa Iftikhar (Tufts)

Fahad R. Dogar (Tufts)

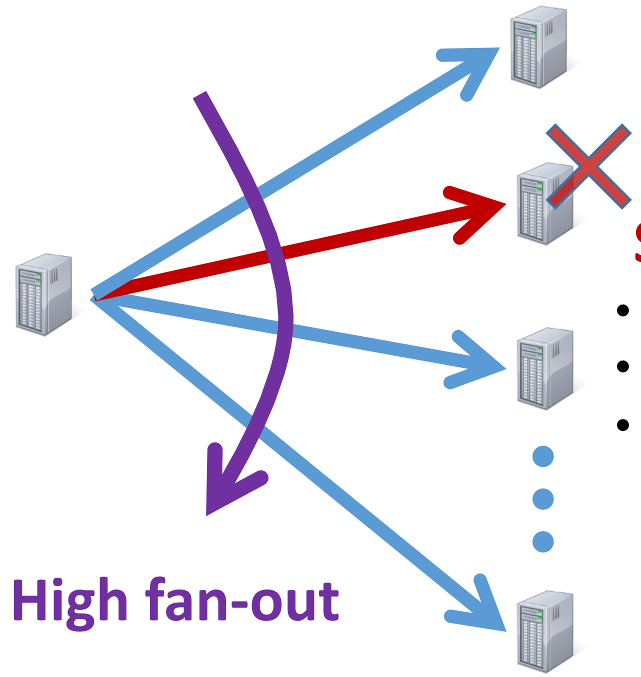
Ihsan Ayyub Qazi (LUMS)

The Problem of Tail Latency in Data Centers!

The Problem of Tail Latency in Data Centers!



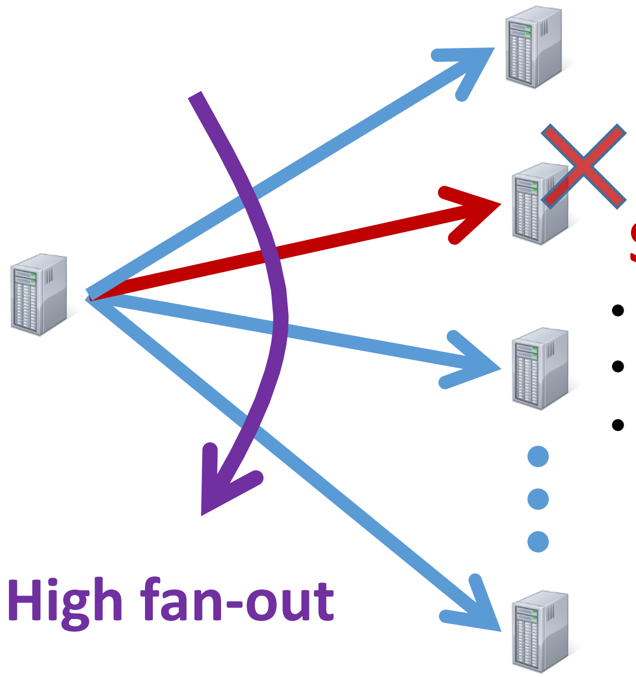
The Problem of Tail Latency in Data Centers!



Straggler

- **Load imbalance**
- **Background tasks**
- **Failures, etc.**

The Problem of Tail Latency in Data Centers!

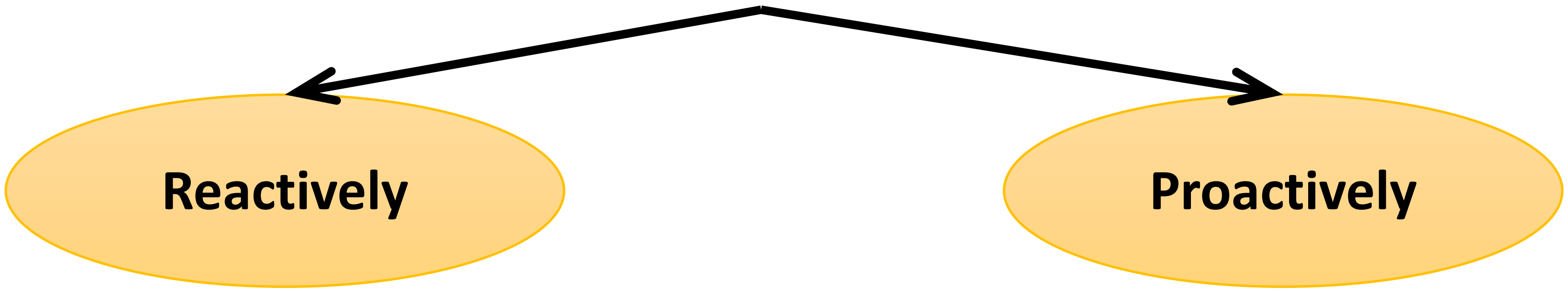


Straggler

- Load imbalance
- Background tasks
- Failures, etc.

High fan-out + Stragglers → Long tail latency

How to avoid stragglers?



How to avoid stragglers?

Reactively

Proactively

Hopper (SIGCOMM'15)
C3 (NSDI'15)
Sinbad(SIGCOMM'13)

PRO: low overhead

CON: requires straggler detection (slow and inaccurate)

How to avoid stragglers?

Reactively

Hopper (SIGCOMM'15)
C3 (NSDI'15)
Sinbad(SIGCOMM'13)

PRO: low overhead | **CON**: requires straggler detection (slow and inaccurate)

Proactively

Dolly (NSDI'13)
Low latency via redundancy (CoNext'13)

PRO: fast and accurate | **CON**: requires determining threshold load (non-trivial)

How to avoid stragglers?

Reactively

Hopper (SIGCOMM'15)
C3 (NSDI'15)
Sinbad(SIGCOMM'13)

Proactively

Dolly (NSDI'13)
Low latency via redundancy (CoNext'13)

PRO: low overhead

CON: requires straggler detection (slow and inaccurate)

PRO: fast and accurate

CON: requires determining threshold load (non-trivial)

Can we achieve the benefits of both without their limitations?

Overview

- **Duplicate-Aware Scheduling Framework**

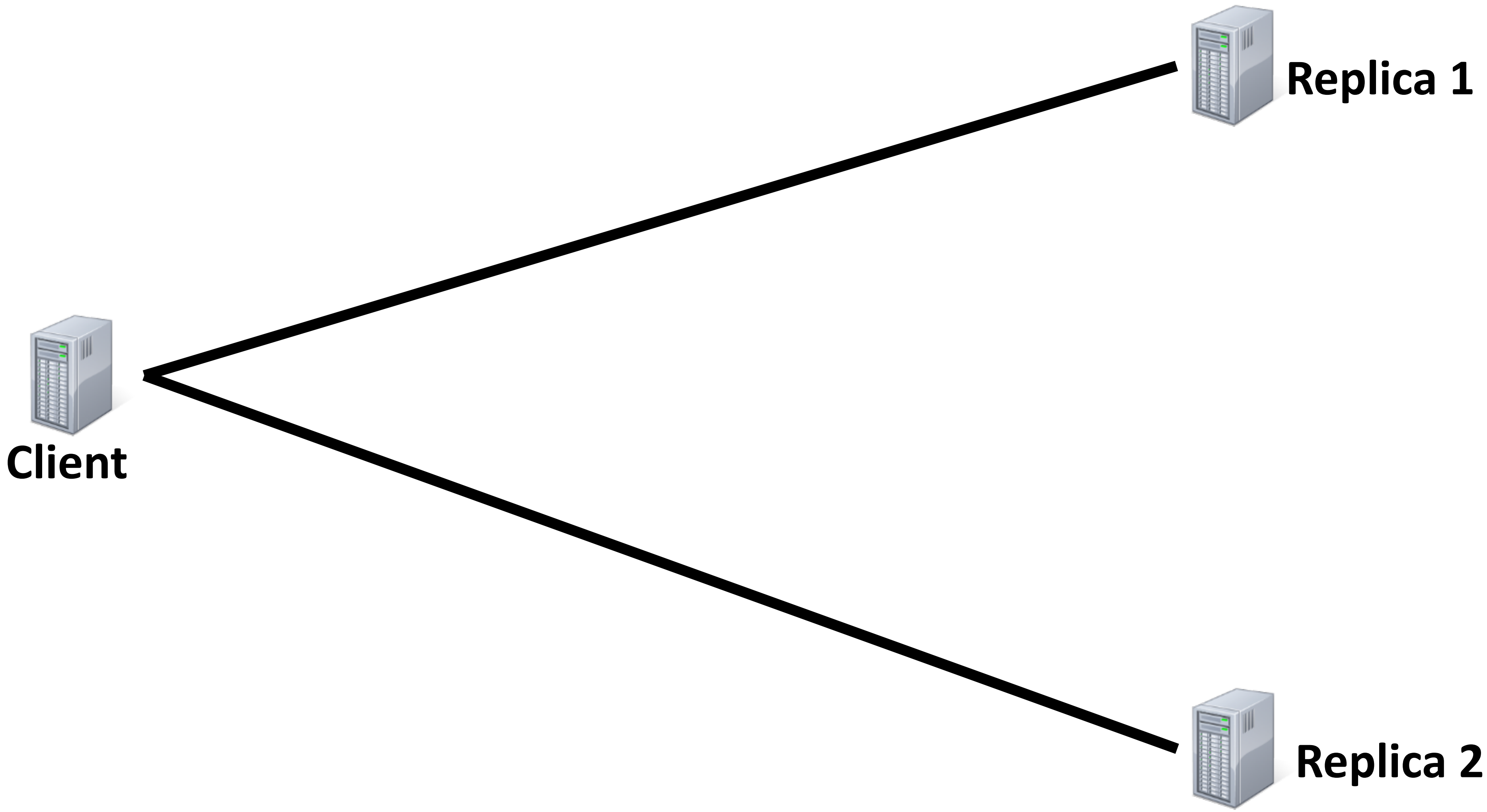
  Generic framework

- **Redundancy-Aware Network Stack**

  New network stack for DC

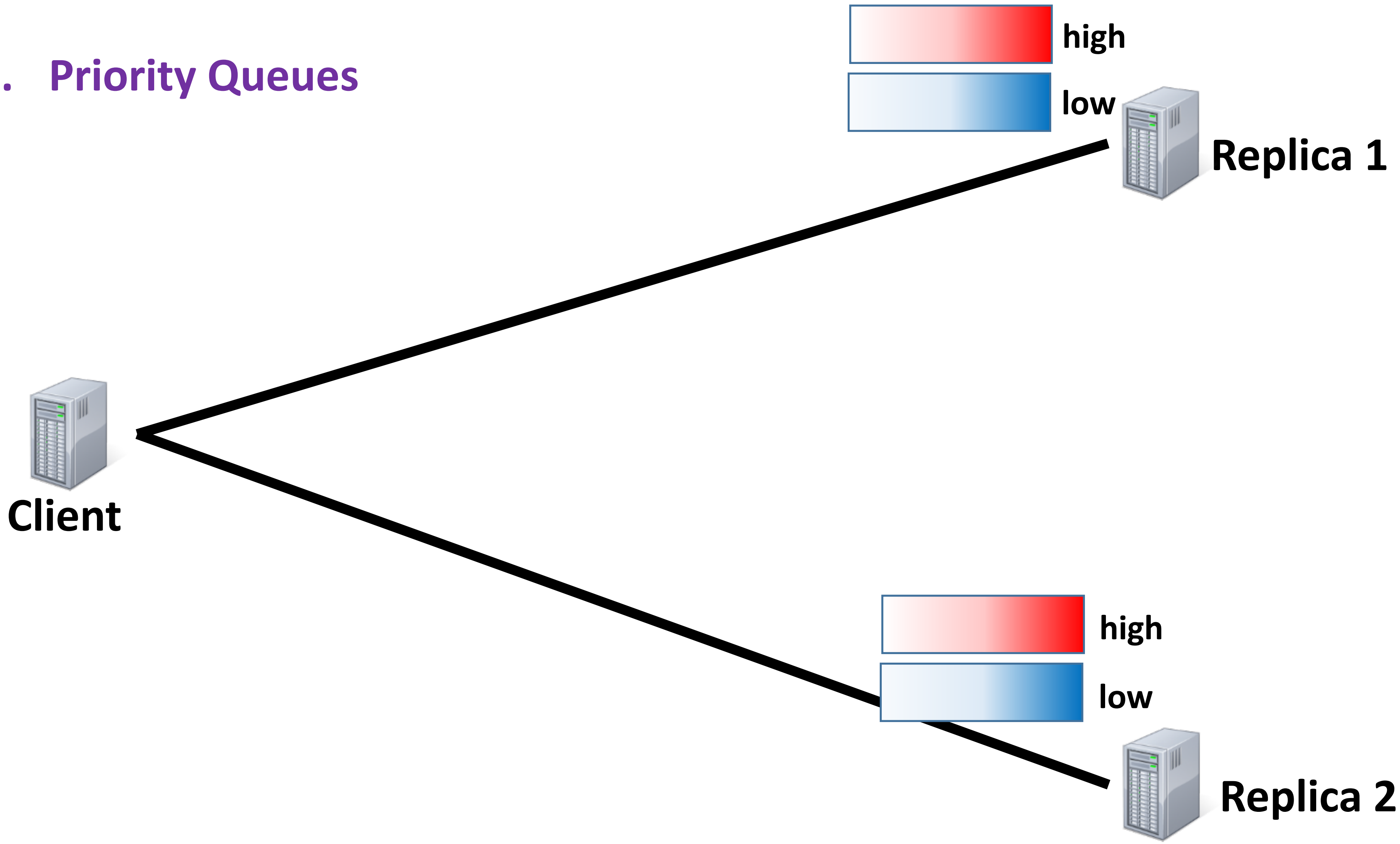
- **Preliminary Results**

Duplicate-aware scheduling



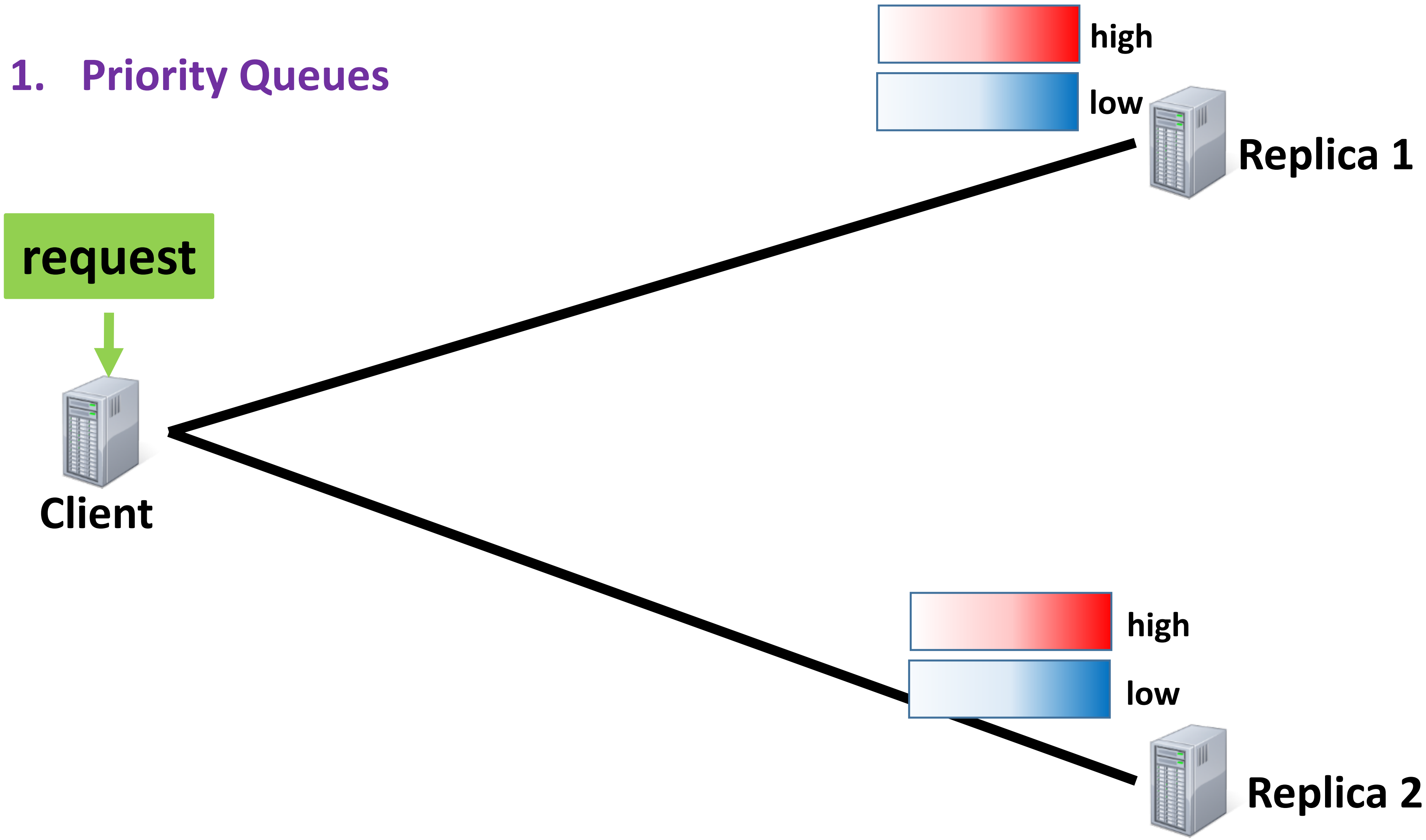
Duplicate-aware scheduling

1. **Priority Queues**



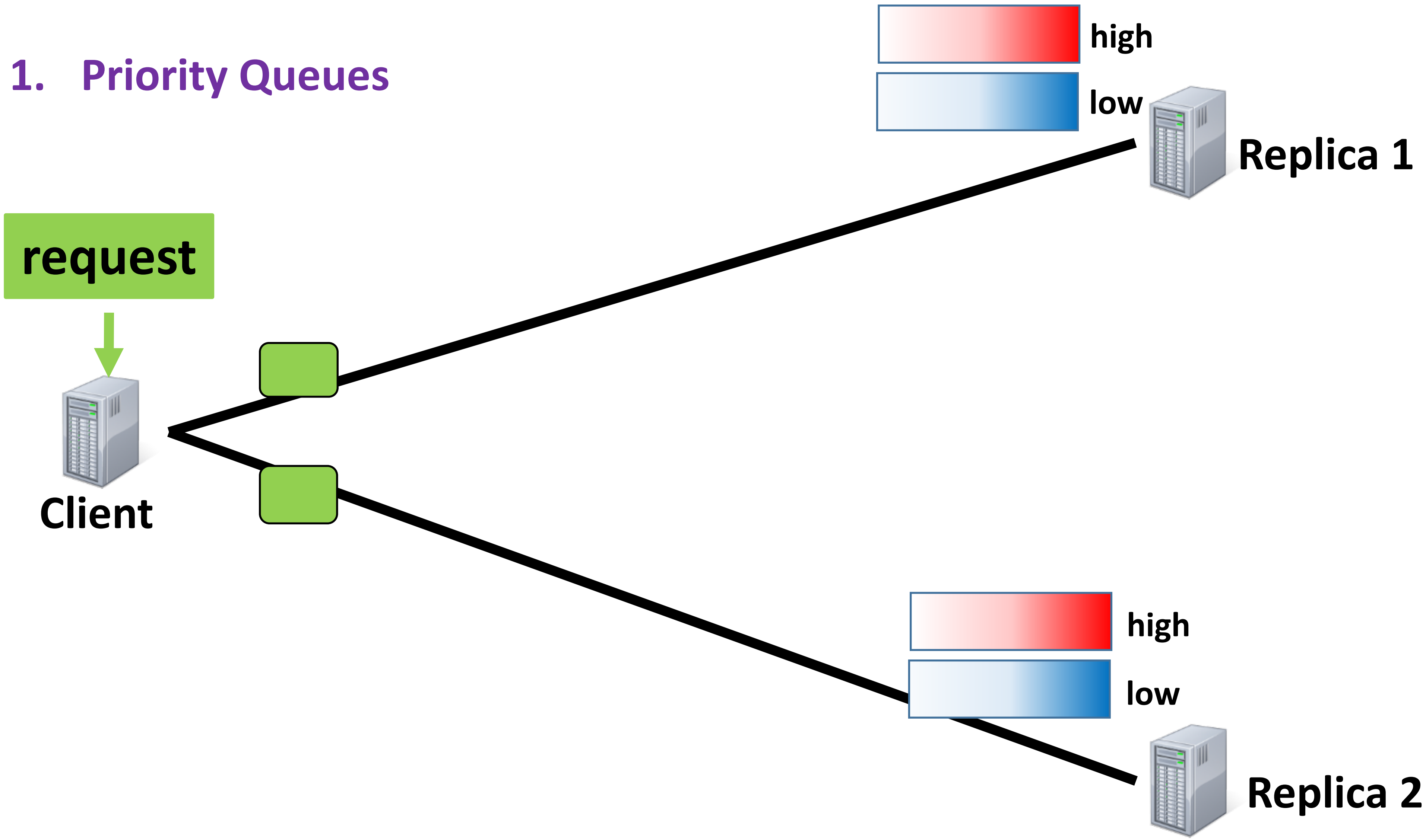
Duplicate-aware scheduling

1. Priority Queues



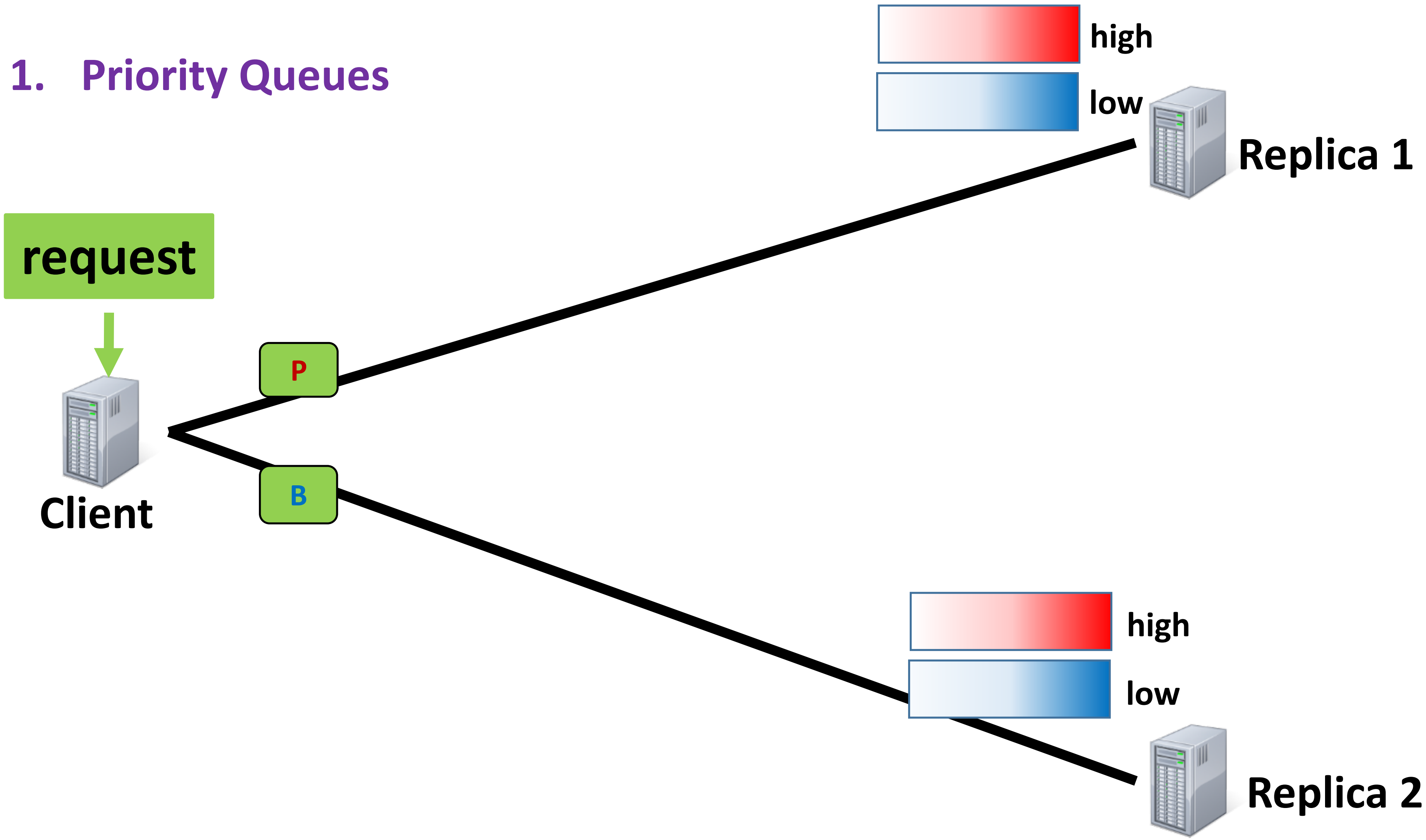
Duplicate-aware scheduling

1. **Priority Queues**



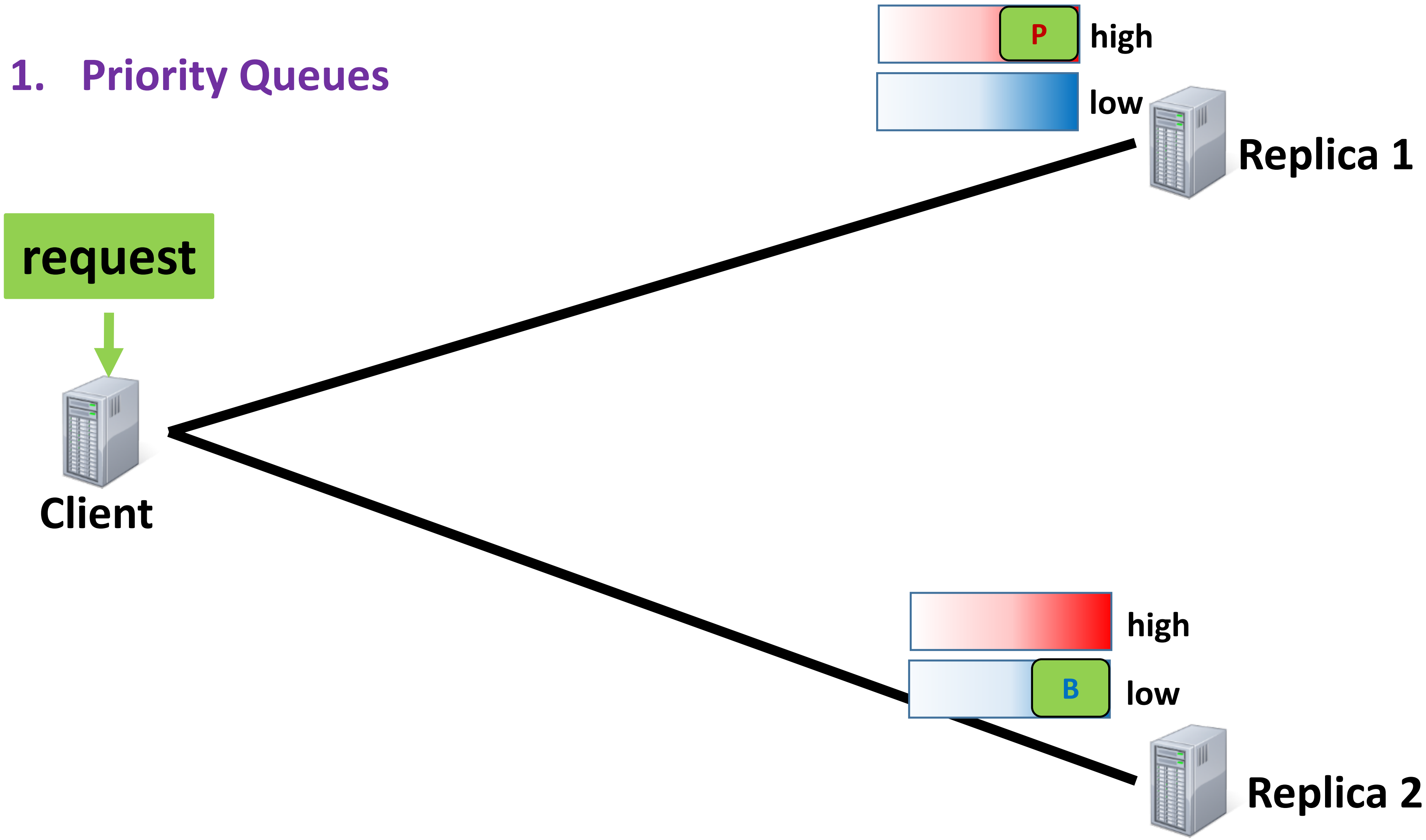
Duplicate-aware scheduling

1. **Priority Queues**



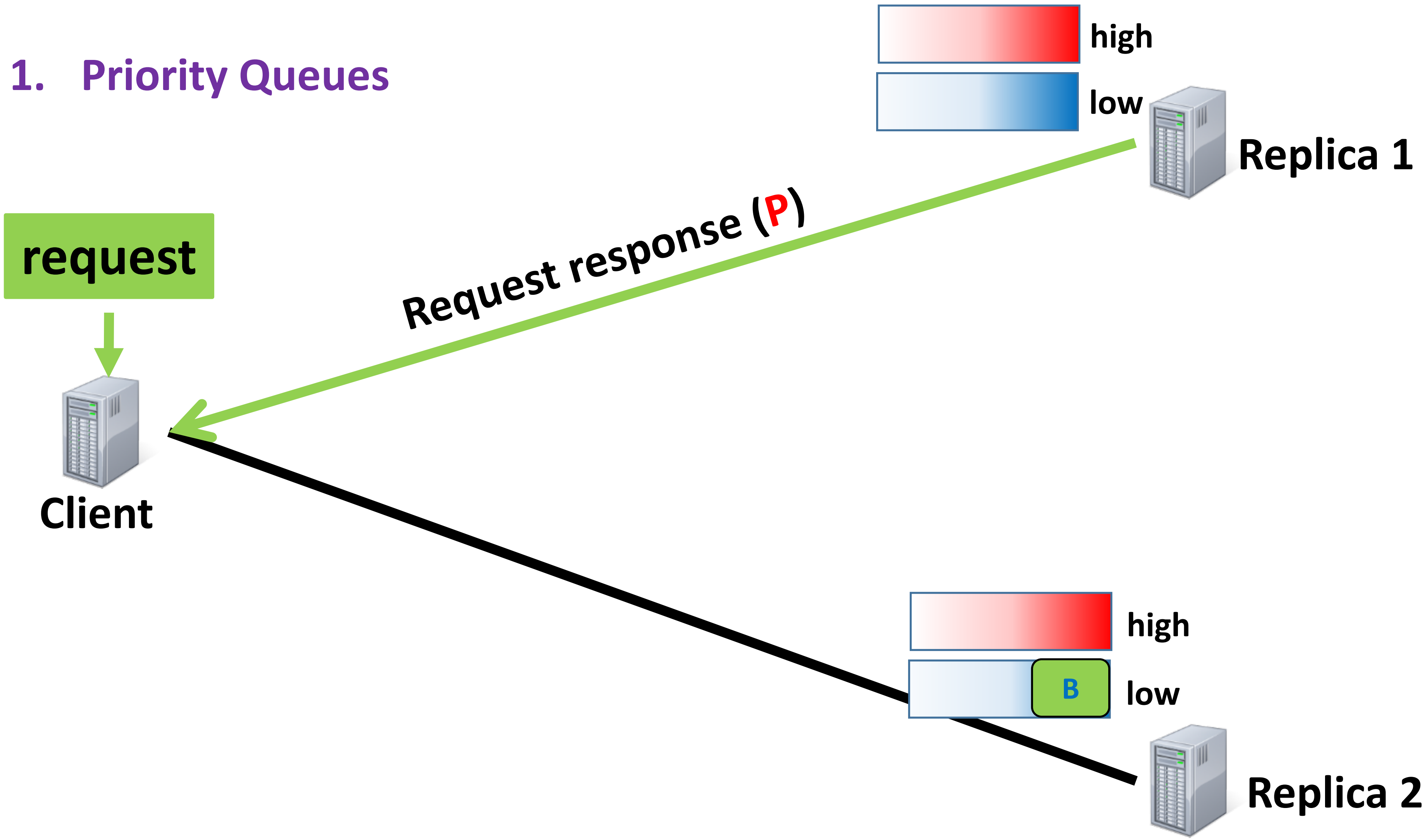
Duplicate-aware scheduling

1. **Priority Queues**



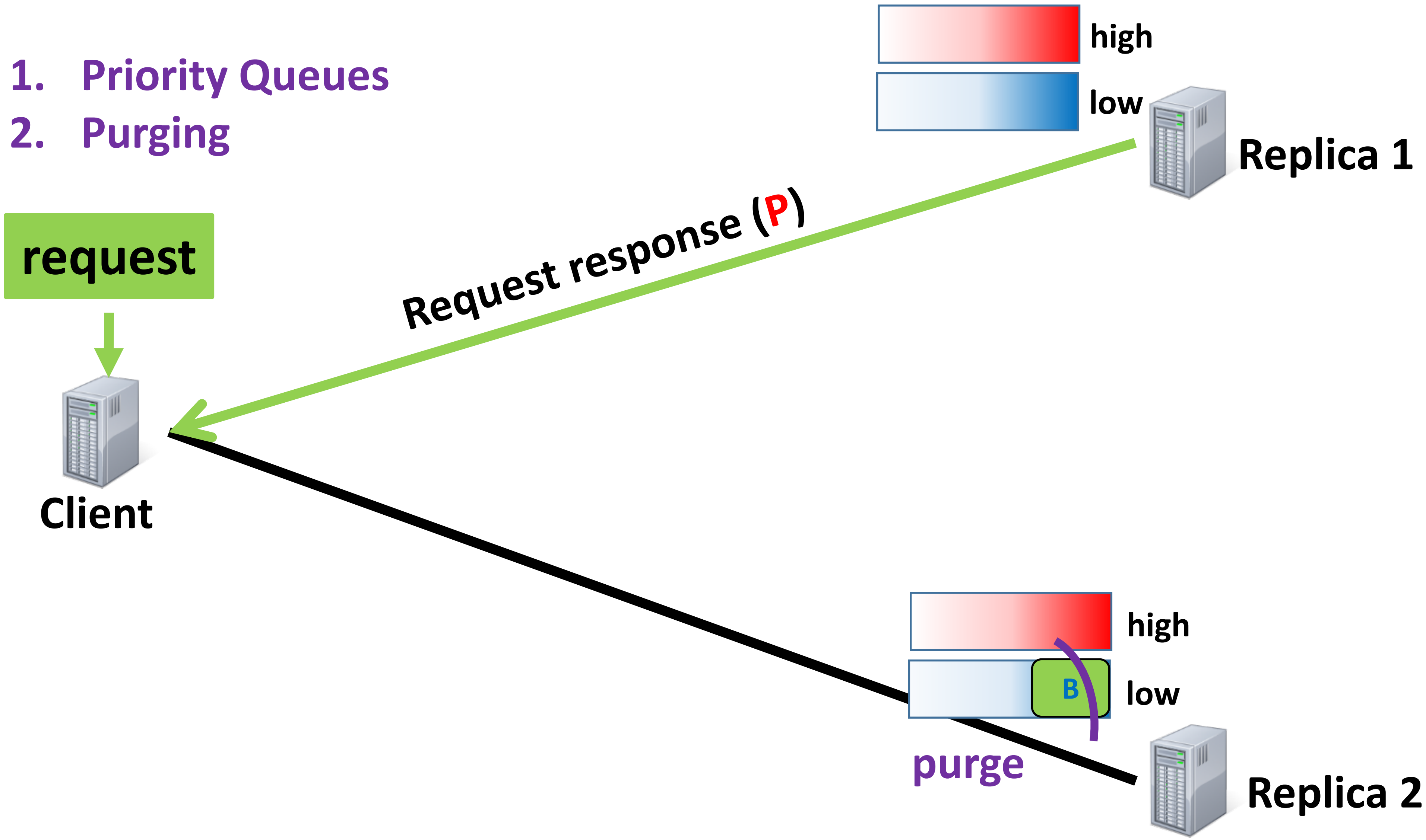
Duplicate-aware scheduling

1. **Priority Queues**



Duplicate-aware scheduling

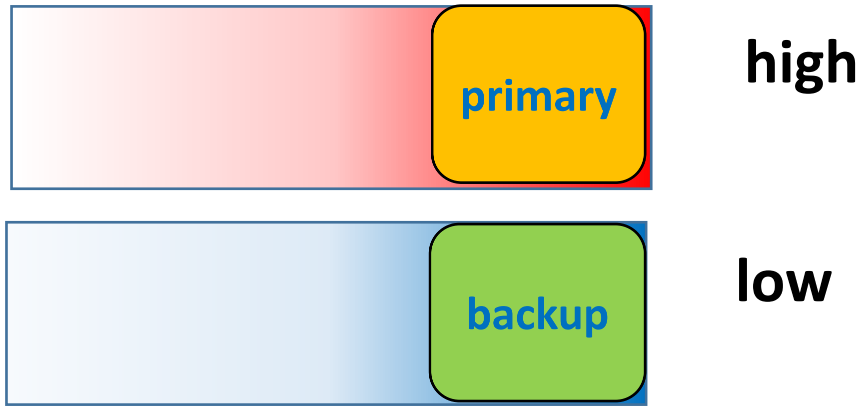
1. Priority Queues
2. Purging



Need for Priority Queuing

- **Duplication has an overhead!**

☹



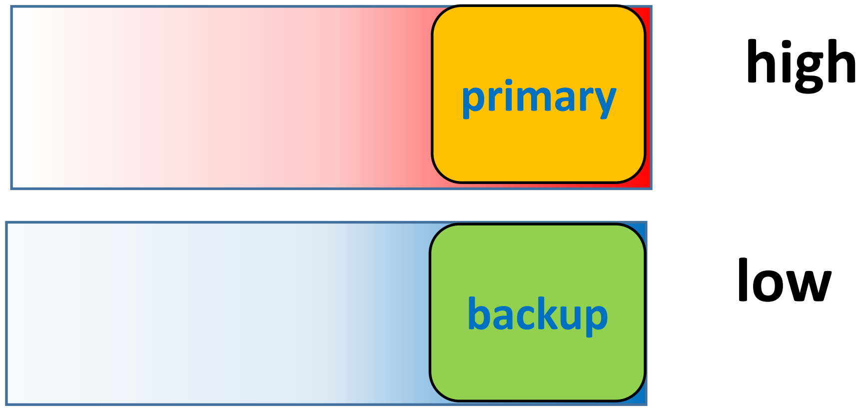
Need for Priority Queuing

- **Duplication has an overhead!**

☹

Properties required:

- ✓ **Strict priorities**
- ✓ **Work conservation**
- ✓ **Preemption**



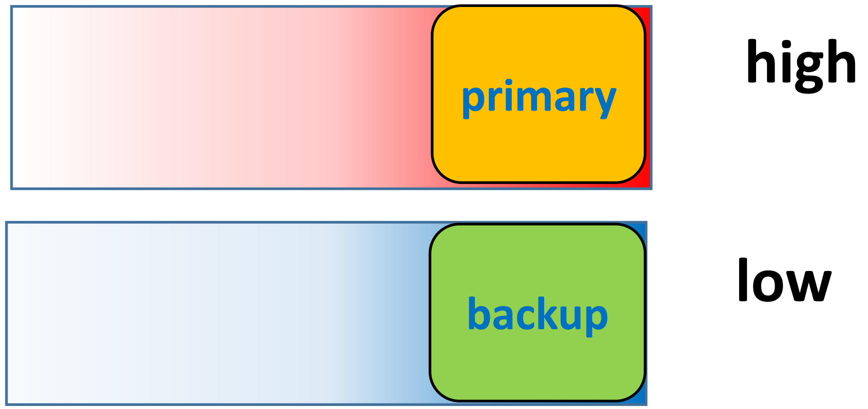
Need for Priority Queuing

- Duplication has an overhead!

☹

Properties required:

- ✓ Strict priorities
- ✓ Work conservation



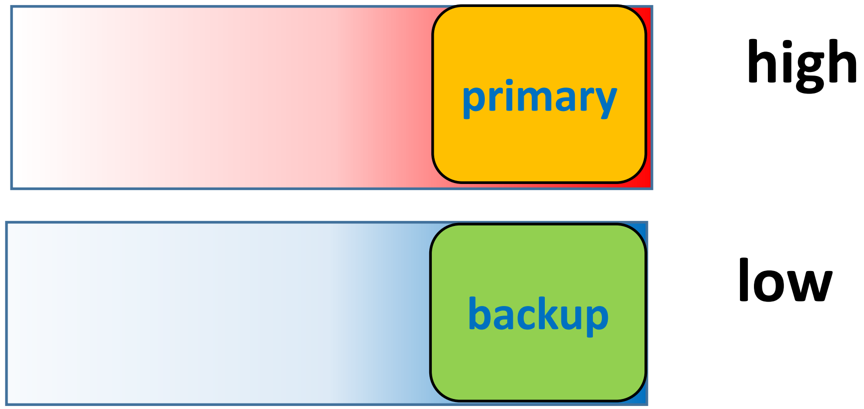
PQ makes the overhead of duplication low. ☺

Need for Priority Queuing

- **Duplication has an overhead!**

Properties required:

- ✓ **Strict priorities**
- ✓ **Work conservation**



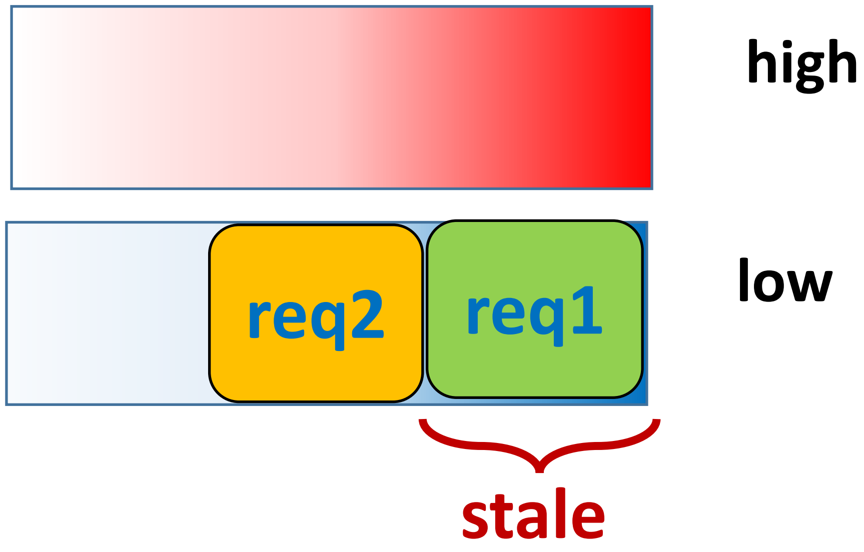
PQ makes the overhead of duplication <u>low</u>.

essential

Importance of Purging

- **Stale requests block new requests.**

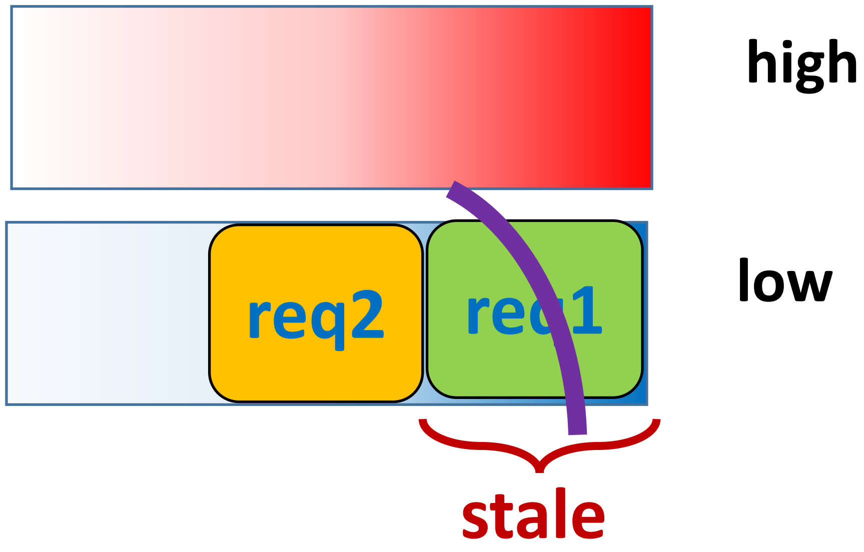
☹



Importance of Purging

- **Stale requests block new requests.**

☹

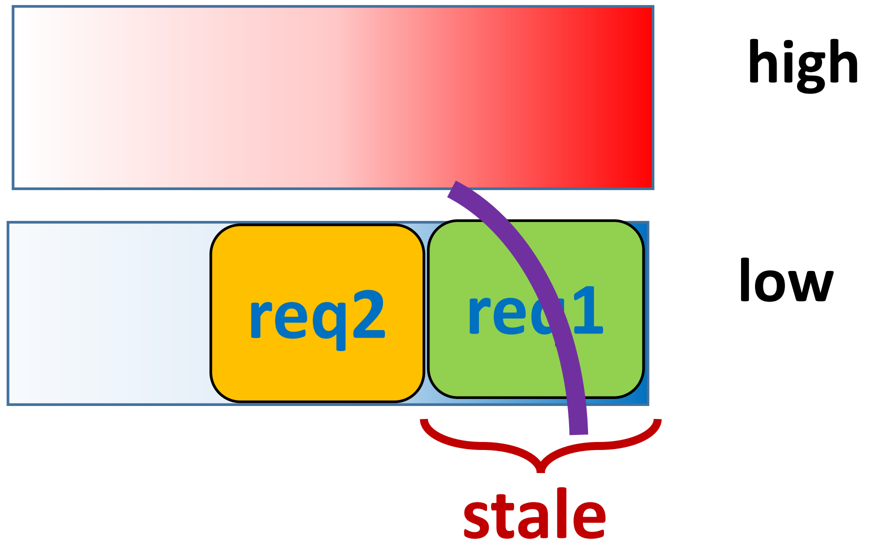


Purging makes the system more efficient! ☺

Importance of Purging

- **Stale requests block new requests.**

☹



Purging makes the system more <u>efficient</u>! ☺

optimization

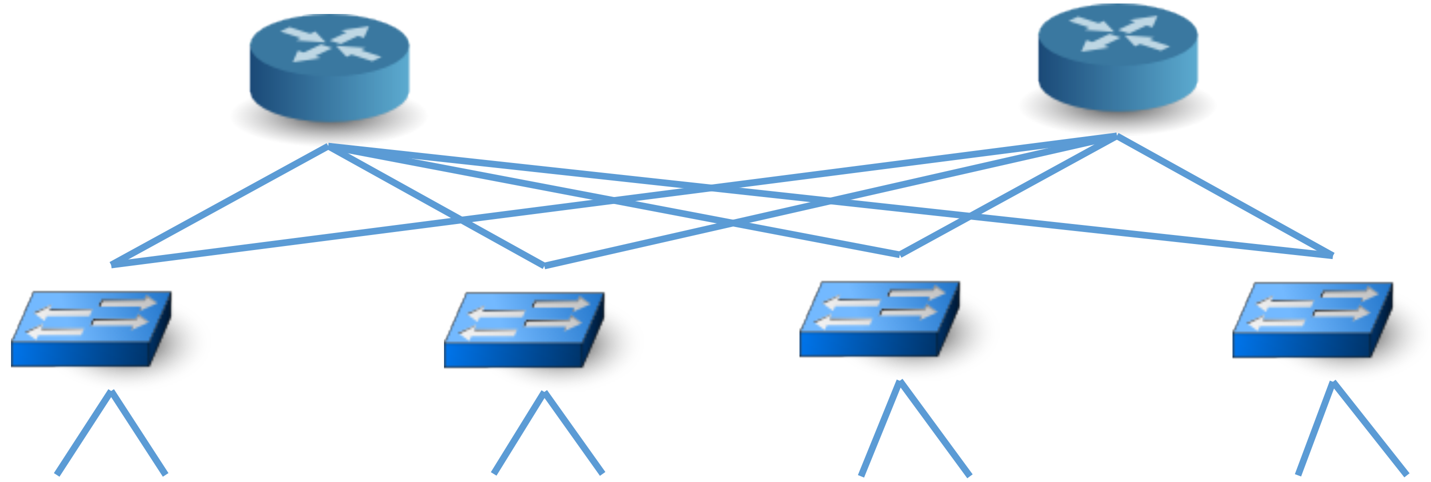
Realizing Duplicate-Aware Scheduling

at every **potential bottleneck** resource in a DC

Realizing Duplicate-Aware Scheduling

at every **potential bottleneck** resource in a DC

Network

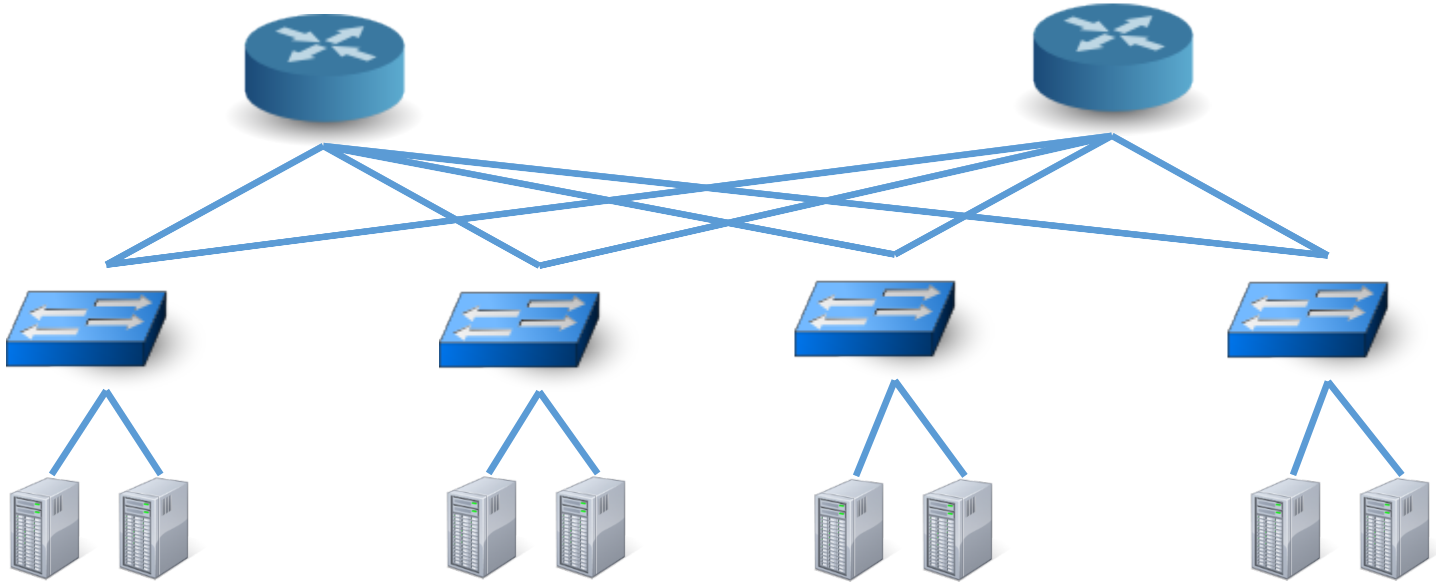


Realizing Duplicate-Aware Scheduling

at every **potential bottleneck** resource in a DC

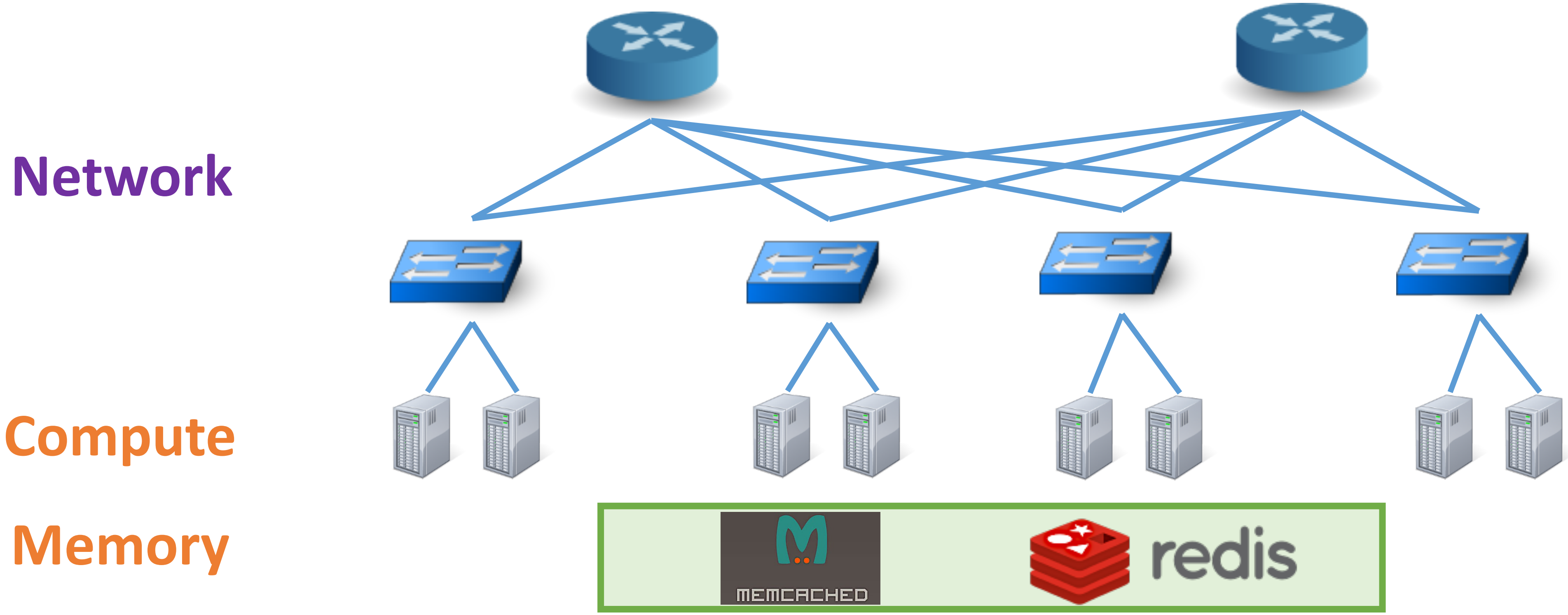
Network

Compute



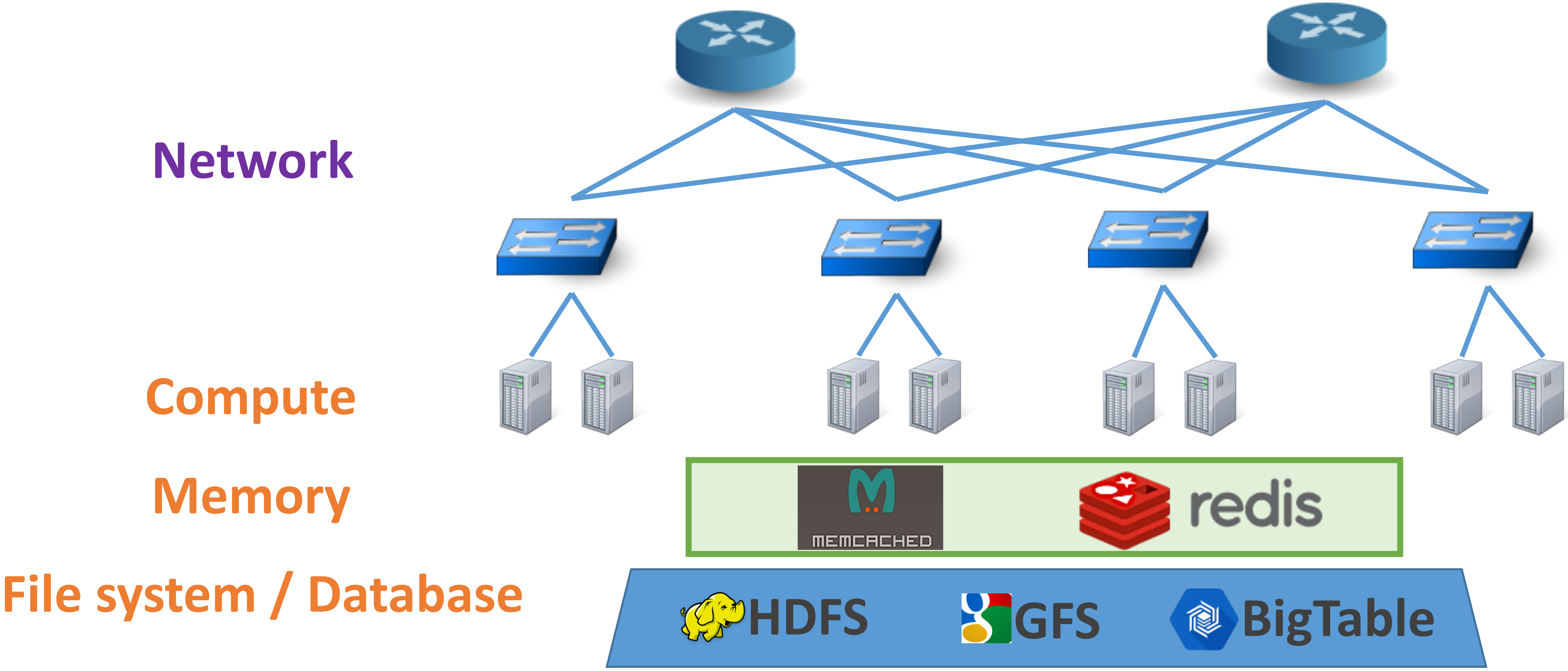
Realizing Duplicate-Aware Scheduling

at every **potential bottleneck** resource in a DC



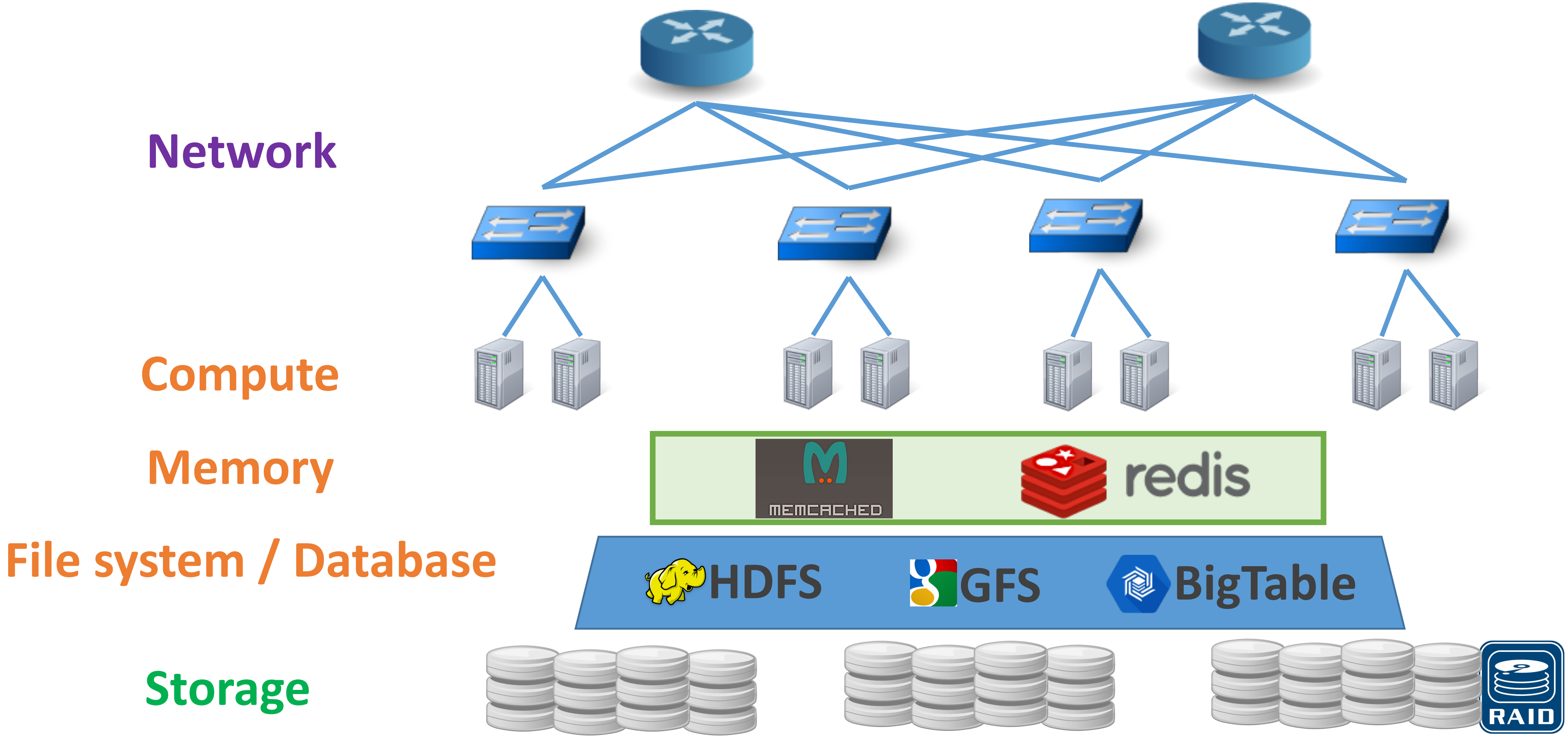
Realizing Duplicate-Aware Scheduling

at every **potential bottleneck** resource in a DC



Realizing Duplicate-Aware Scheduling

at every **potential bottleneck** resource in a DC



Realizing Duplicate-Aware Scheduling

at every **potential bottleneck** resource in a DC

challenges

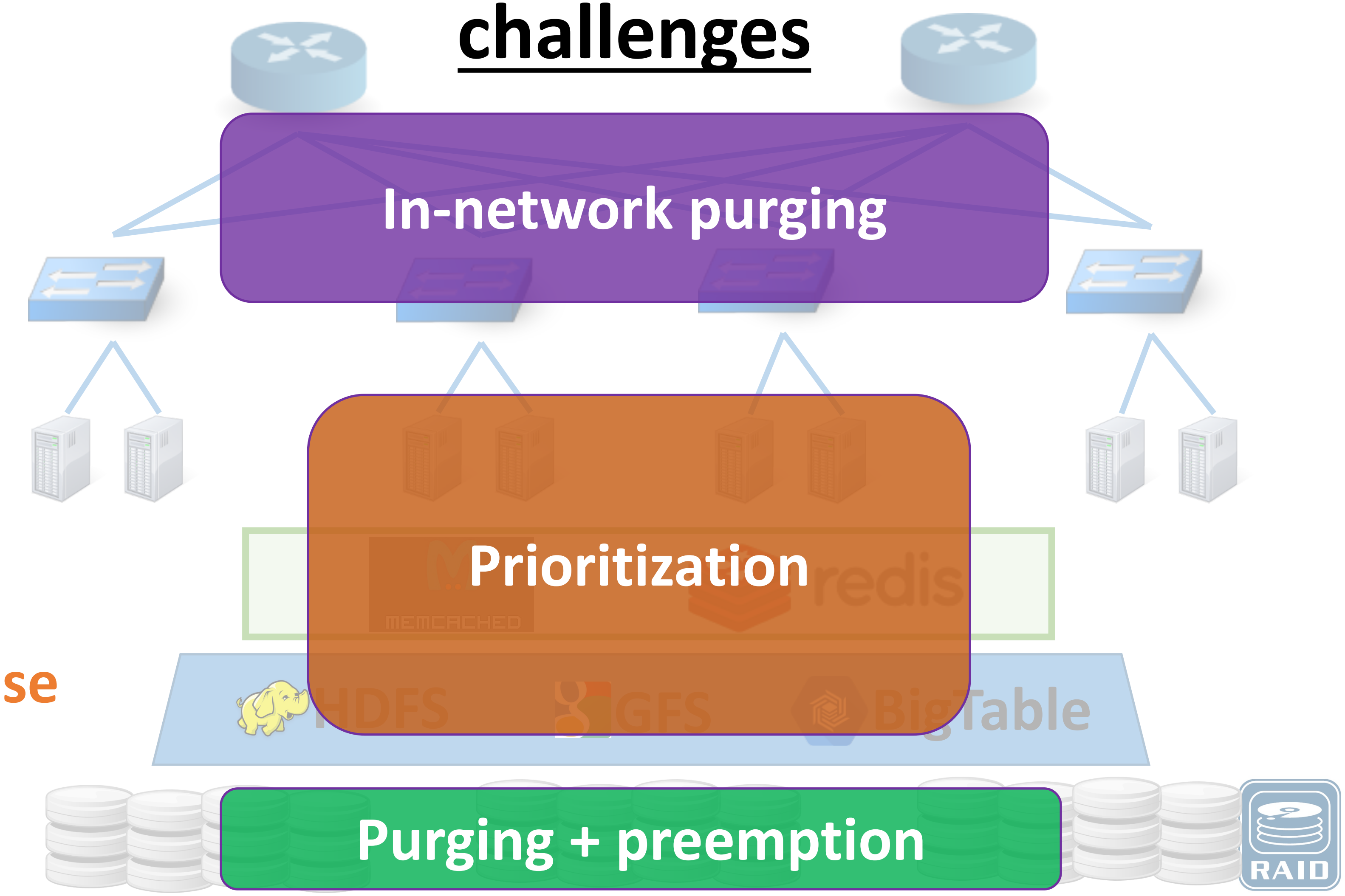
Network

Compute

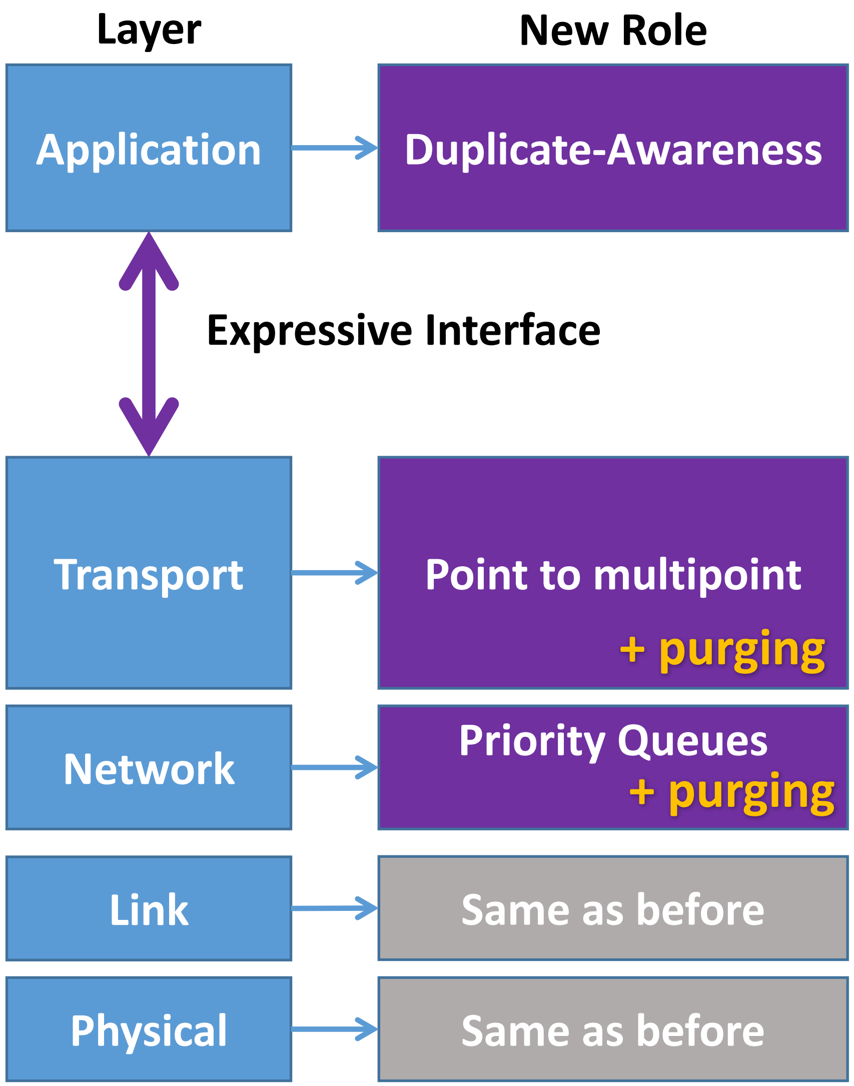
Memory

File system / Database

Storage



Redundancy Aware Network Stack (RANS)



Redundancy Aware Network Stack (RANS)

Layer	New Role
Application	Duplicate-Awareness
Expressive Interface	
Transport	Point to multipoint + purging
Network	Priority Queues + purging
Link	Same as before
Physical	Same as before

challenge

Applications need to be modified.

Redundancy Aware Network Stack (RANS)

Layer	New Role
Application	Duplicate-Awareness
Transport	Point to multipoint + purging
Network	Priority Queues + purging
Link	Same as before
Physical	Same as before

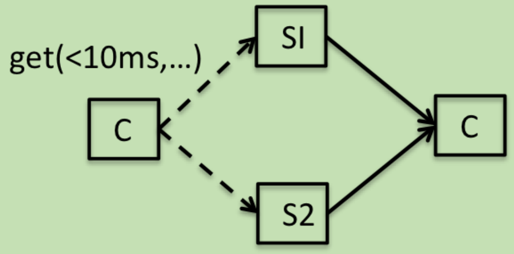
Expressive Interface

challenge

Applications need to be modified.

opportunity

Expressive interface allows rich communication b/w App and Transport.
E.g. DAG



Redundancy Aware Network Stack (RANS)

Layer	New Role
Application	Duplicate-Awareness
Transport	Point to multipoint + purging
Network	Priority Queues + purging
Link	Same as before
Physical	Same as before

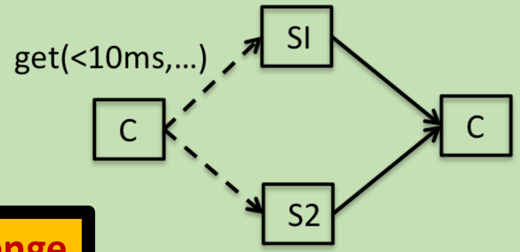
Expressive Interface

challenge

Applications need to be modified.

opportunity

Expressive interface allows rich communication b/w App and Transport.
E.g. DAG



challenge

Hard to implement per packet purging.

Redundancy Aware Network Stack (RANS)

Layer	New Role
Application	Duplicate-Awareness
Transport	Point to multipoint + purging
Network	Priority Queues + purging
Link	Same as before
Physical	Same as before

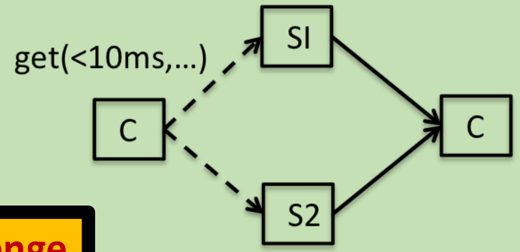
Expressive Interface

challenge

Applications need to be modified.

opportunity

Expressive interface allows rich communication b/w App and Transport.
E.g. DAG



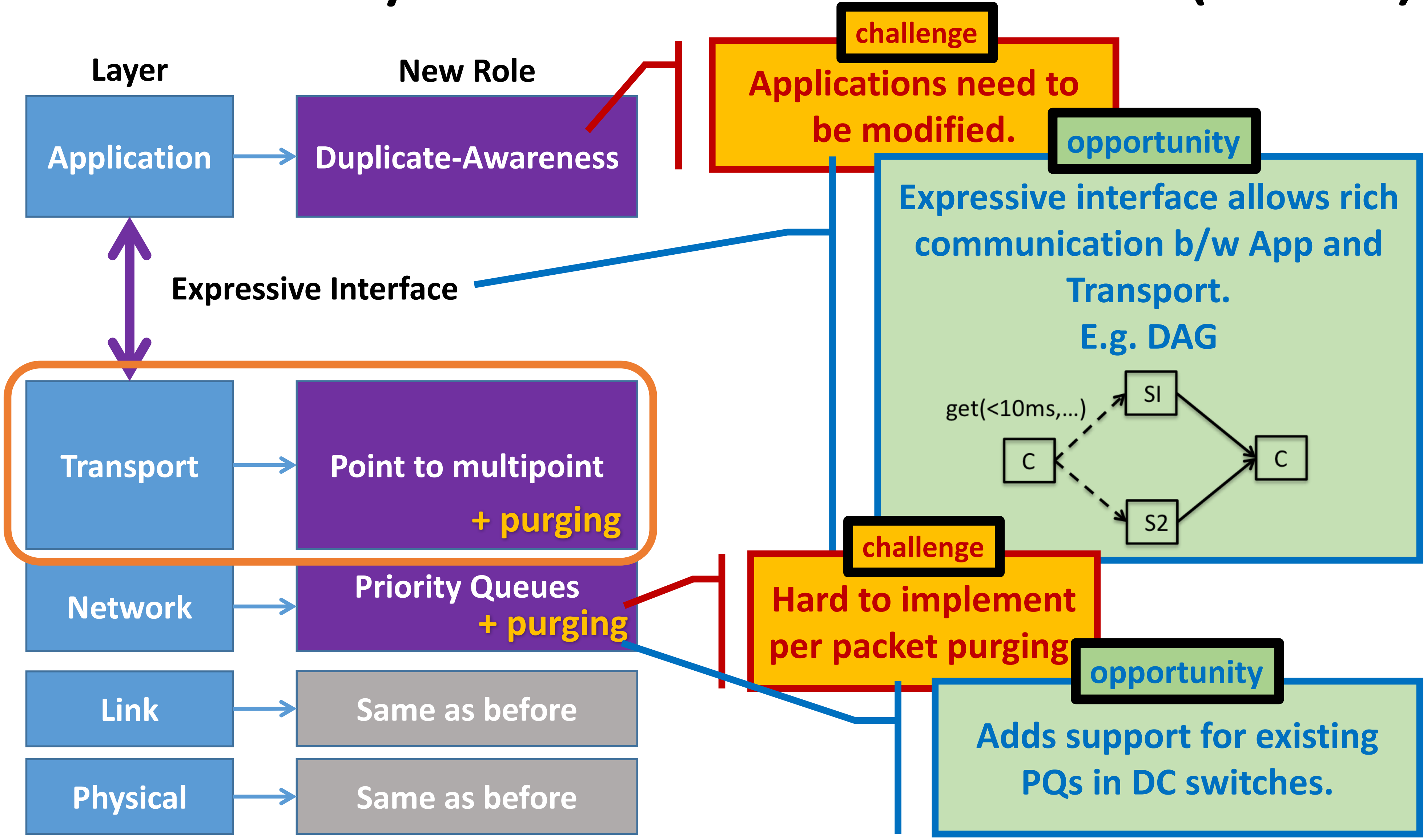
challenge

Hard to implement per packet purging

opportunity

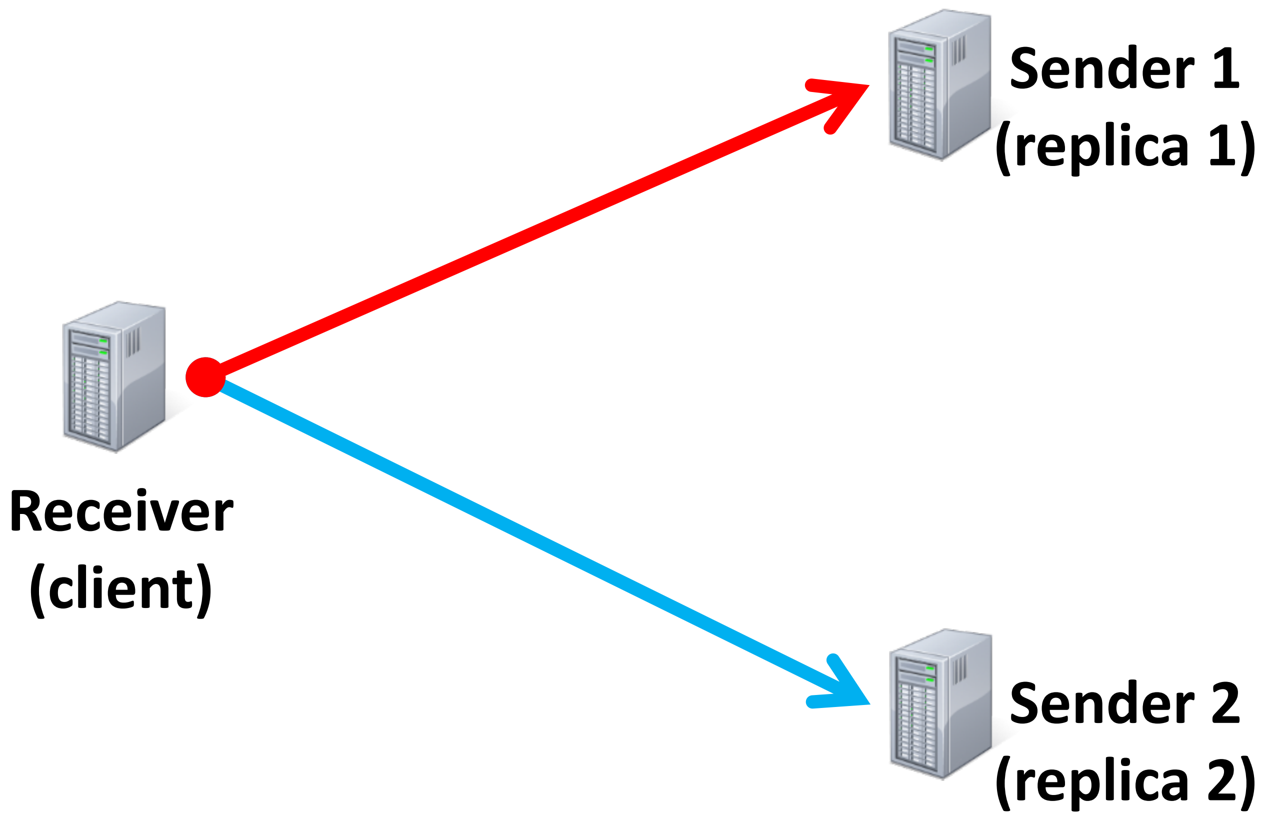
Adds support for existing PQs in DC switches.

Redundancy Aware Network Stack (RANS)



RANS Transport: Point to Multi-point

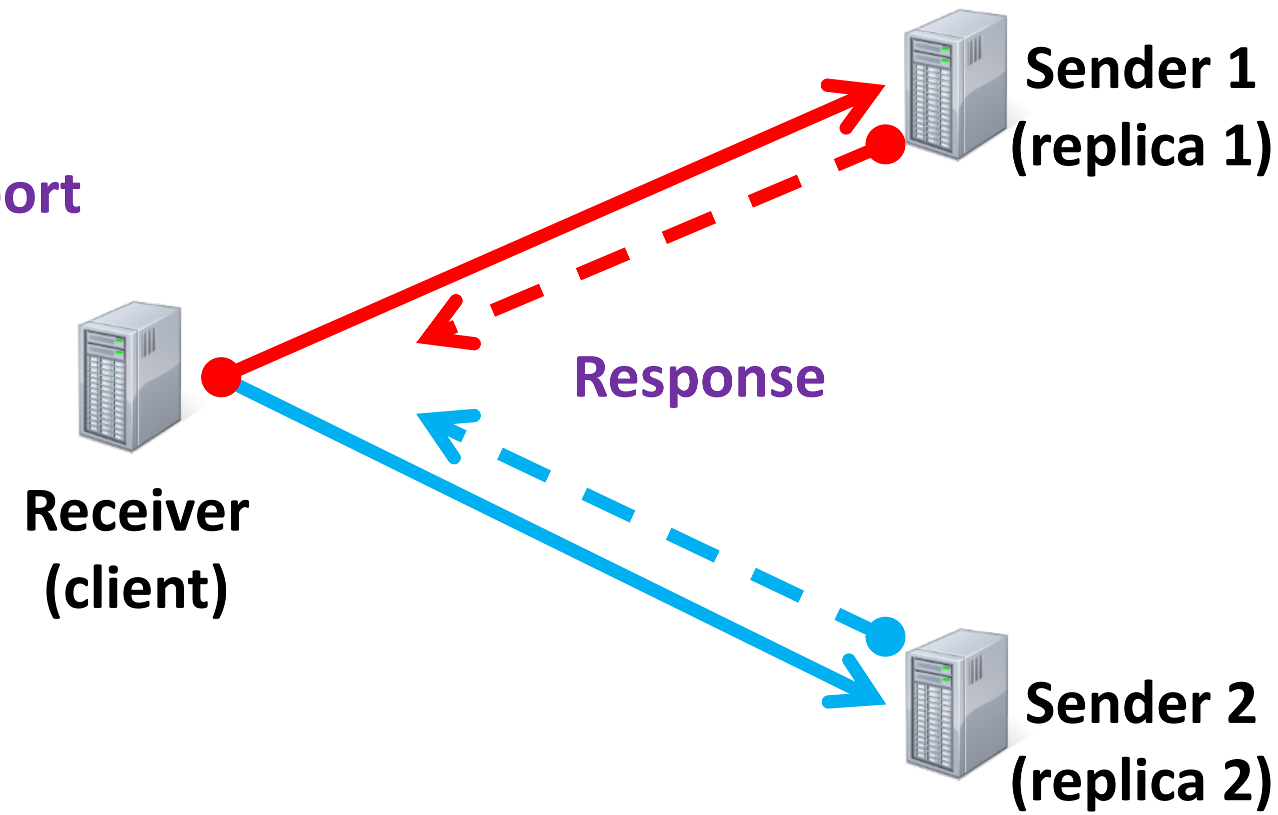
- Enables: Rich transport
  - ✓ Multipath
  - ✓ Multi-destination
    - e.g. Improved fault tolerance



RANS Transport: Byte Aggregation

- Opportunity: Receiver driven transport
  - ✓ Two or more response streams
  - ✓ Aggregate bytes at receiver side

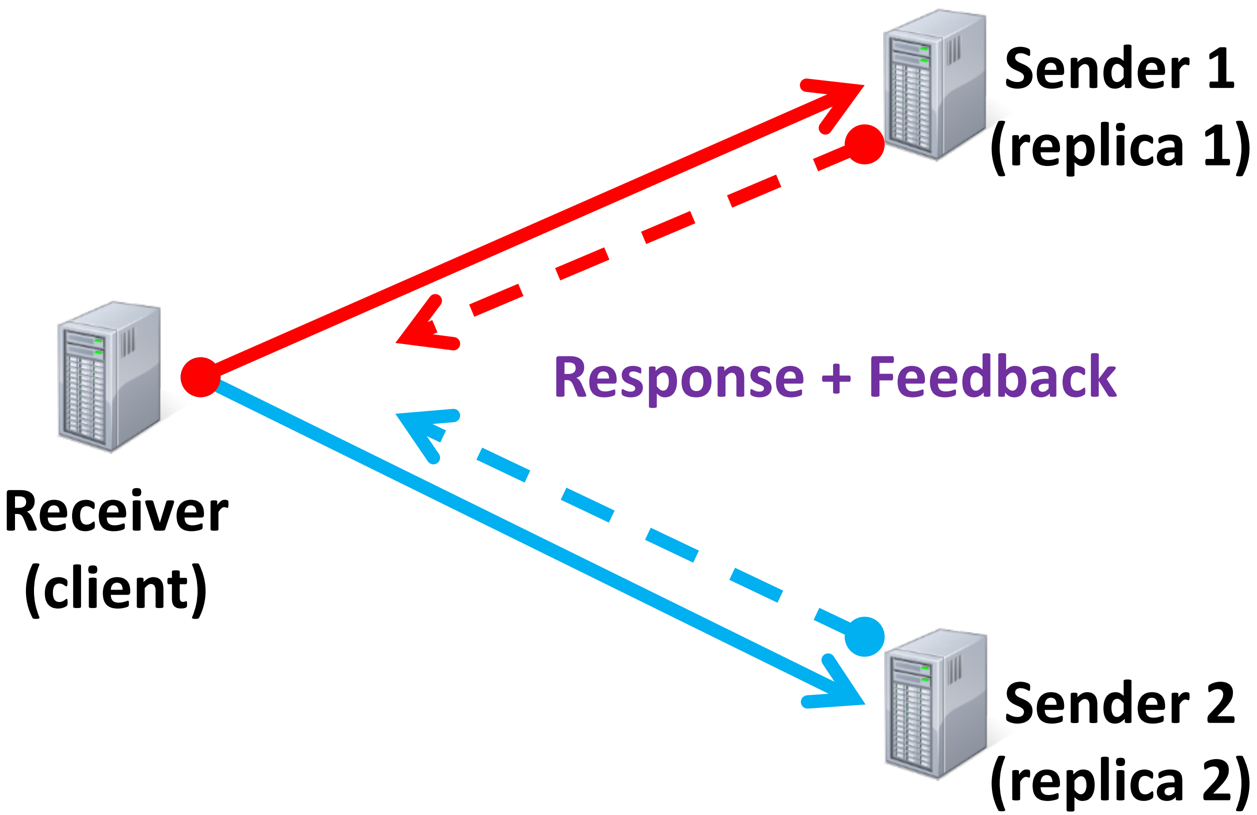
    e.g. More efficient congestion control (2x or more)



RANS Transport: Priority Assignment

- Dynamic replica assignment
  - ✓ Fine grained monitoring of congestion window
  - ✓ Dynamically reprioritize flows
  - ✓ Feedback to Application

    e.g. Improved replica assignment

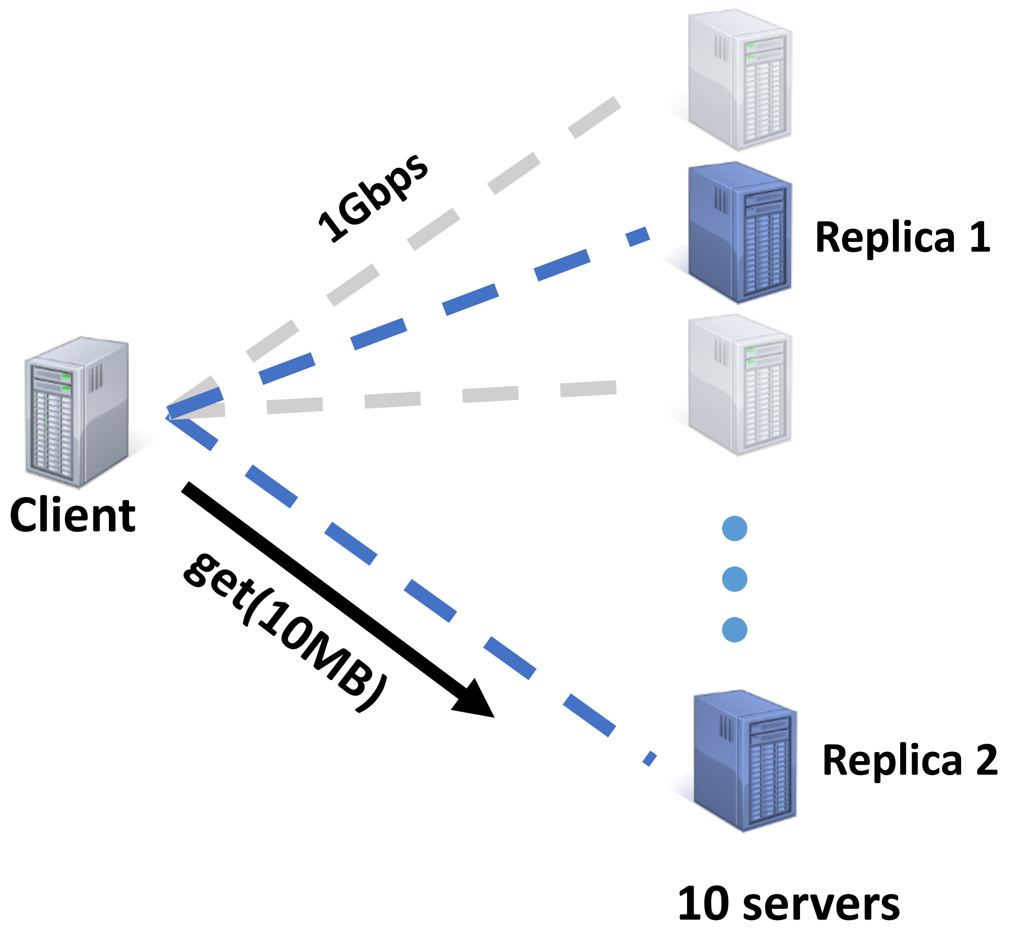


Overview

- Duplicate-Aware Scheduling Framework
- Redundancy-Aware Network Stack
- **Preliminary Results**

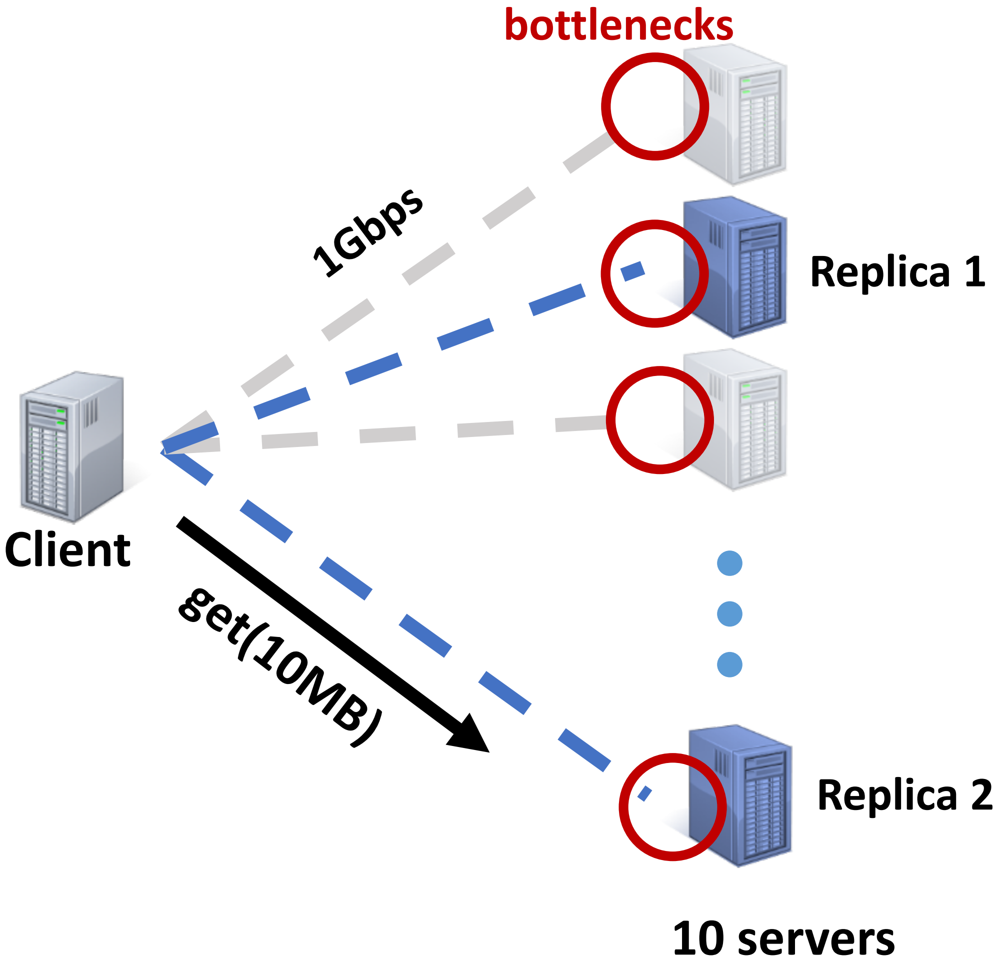
Preliminary Evaluation: ns-2 setup details

➢ **Storage scenario**



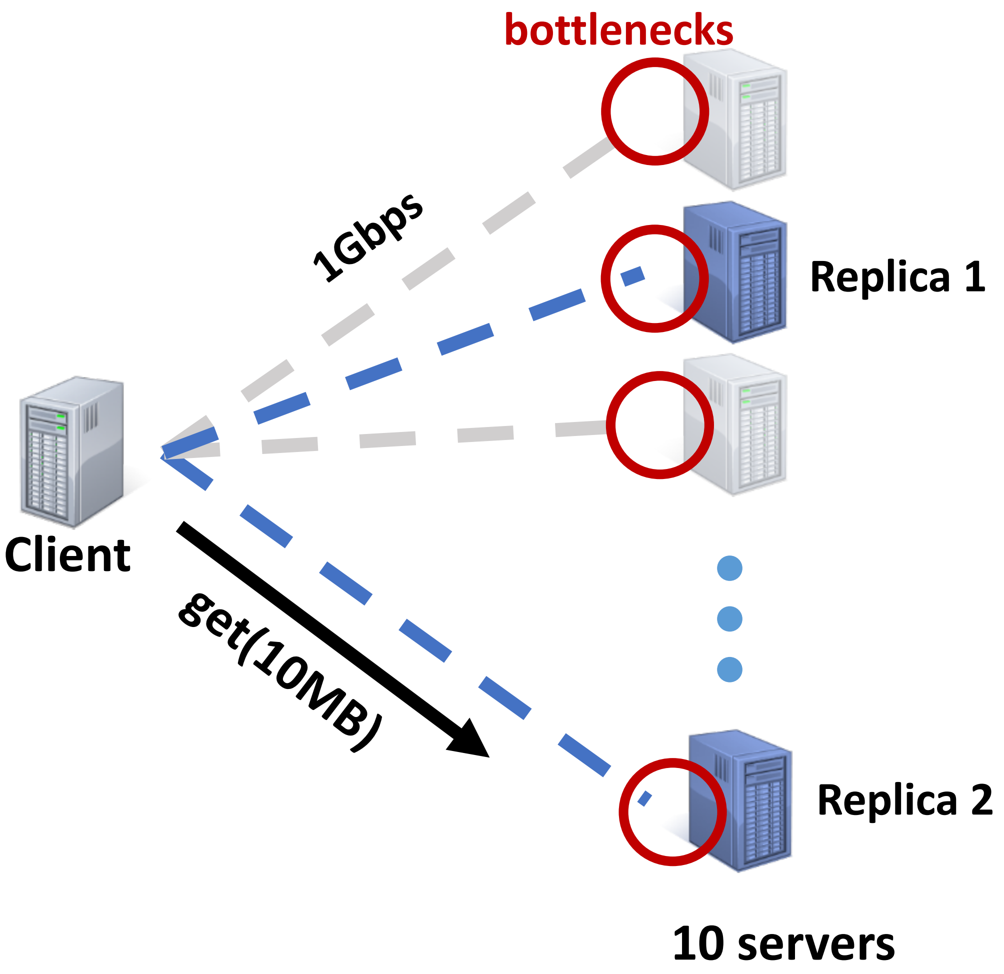
Preliminary Evaluation: ns-2 setup details

- Storage scenario



Preliminary Evaluation: ns-2 setup details

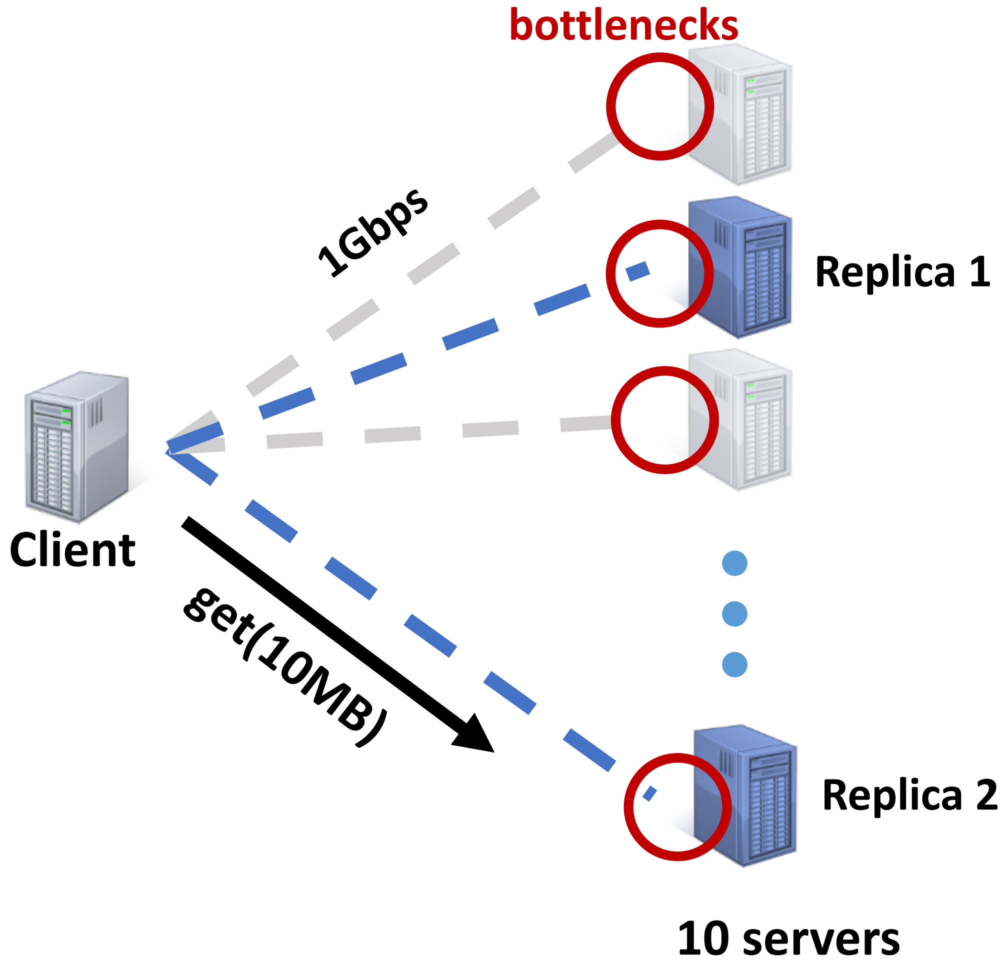
- Storage scenario



Traffic Details	
Total requests	20K
Arrival process	Poisson
Server & replica selection	Uniformly random

Preliminary Evaluation: ns-2 setup details

➢ **Storage scenario**

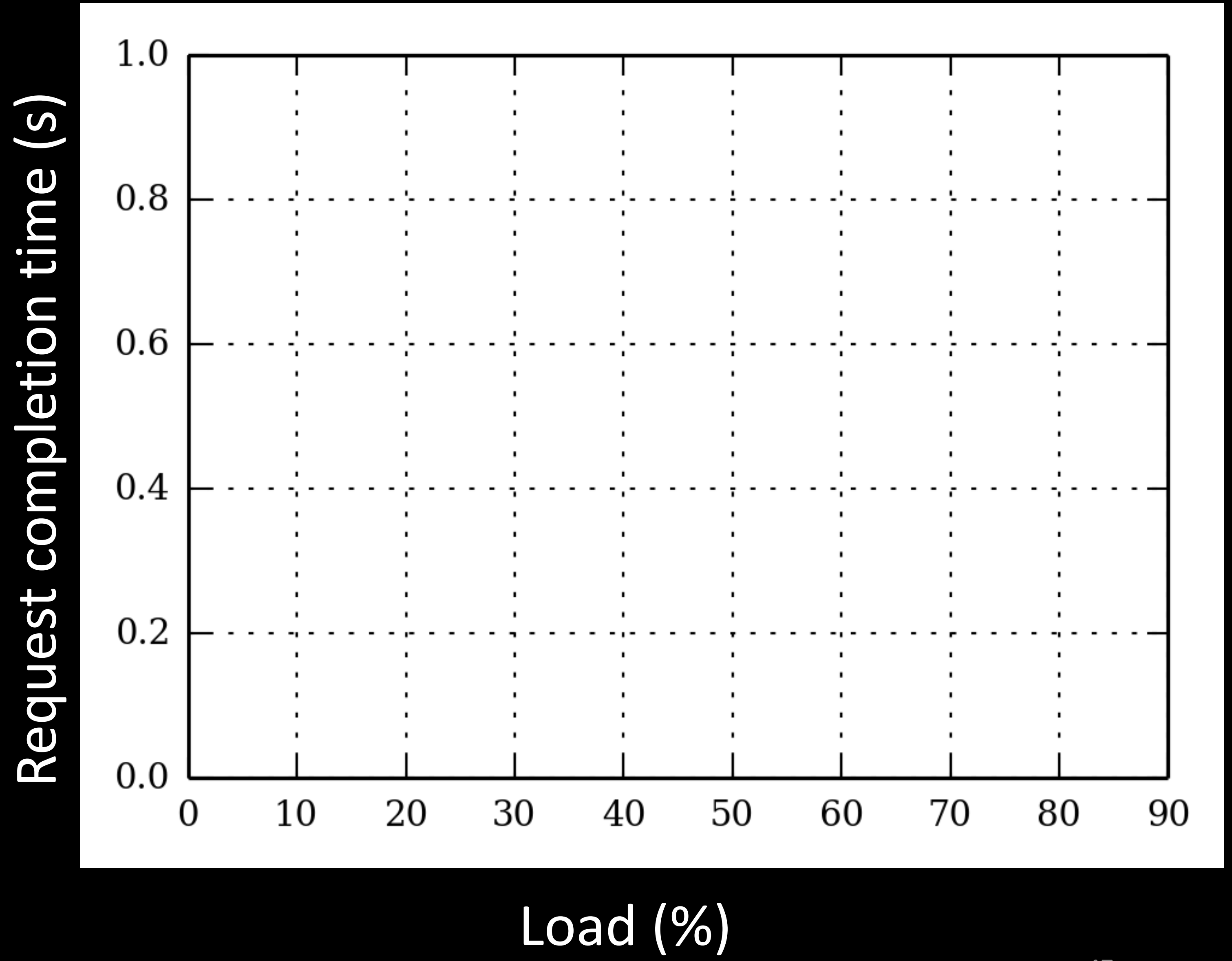


Traffic Details	
Total requests	20K
Arrival process	Poisson
Server & replica selection	Uniformly random

The only source of stragglers is load imbalance.

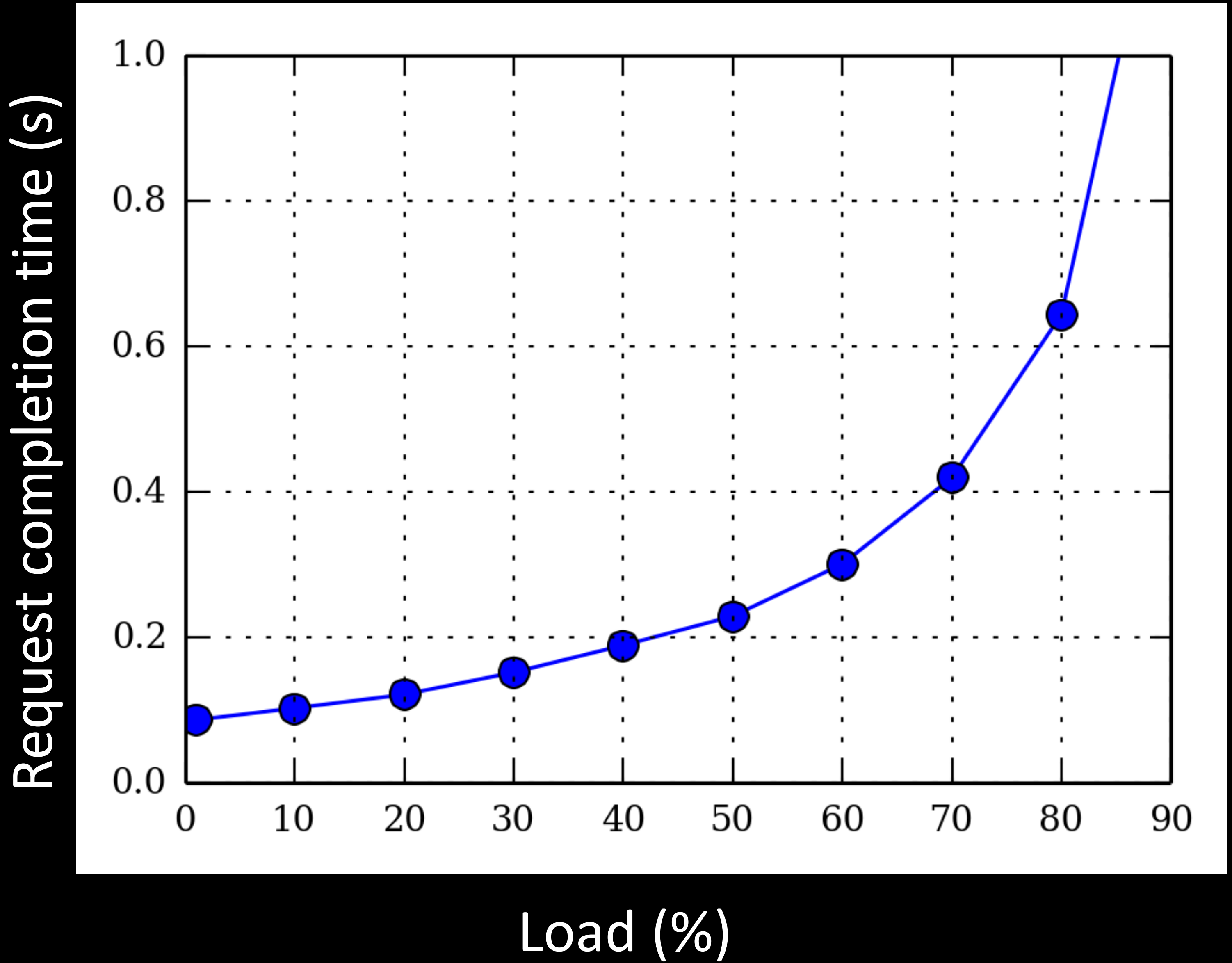
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
- + PQs
- + Purging
- + Byte Aggregation (RANS)

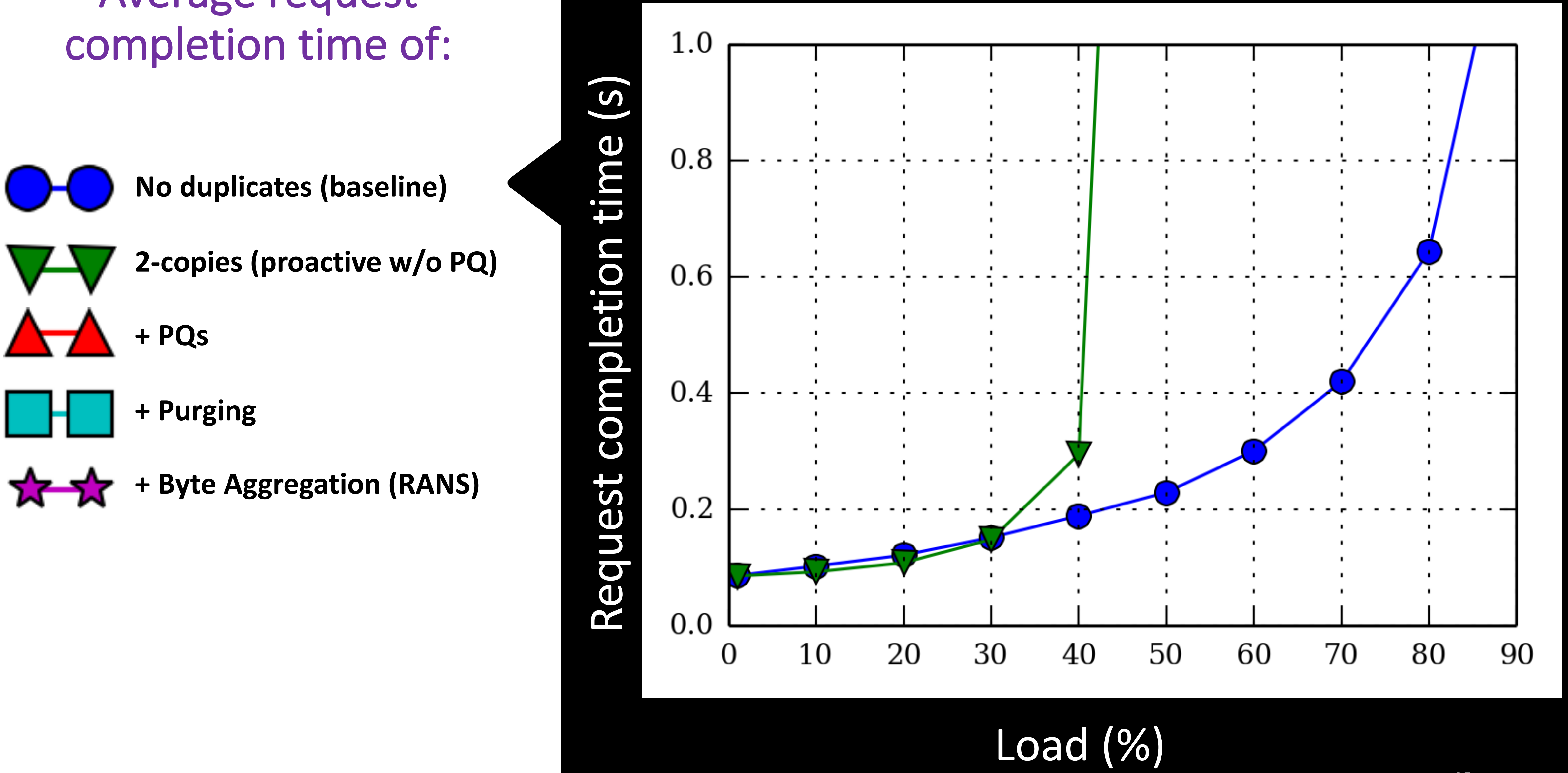


Average request completion time of:

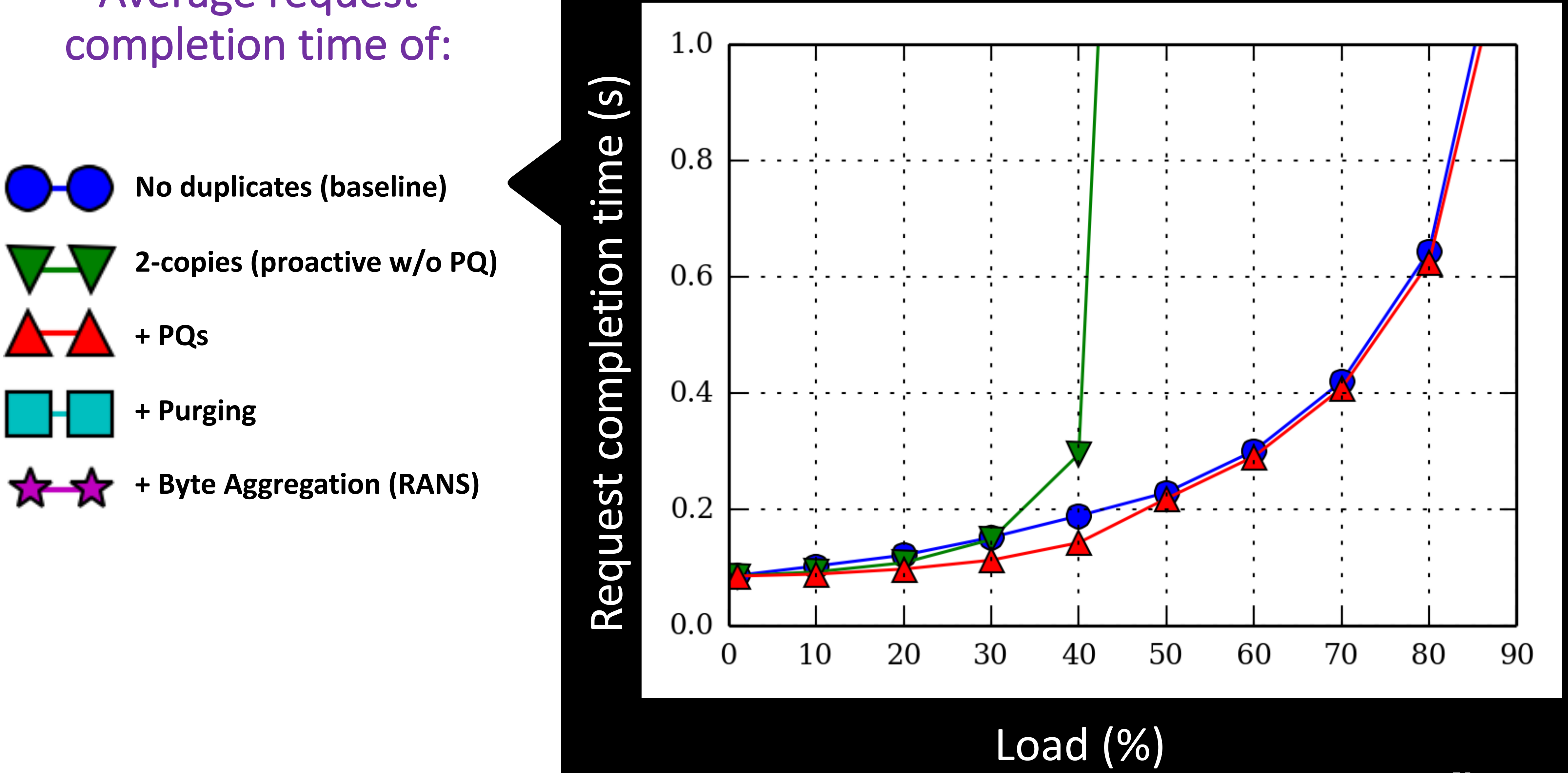
- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
- + PQs
- + Purging
- + Byte Aggregation (RANS)



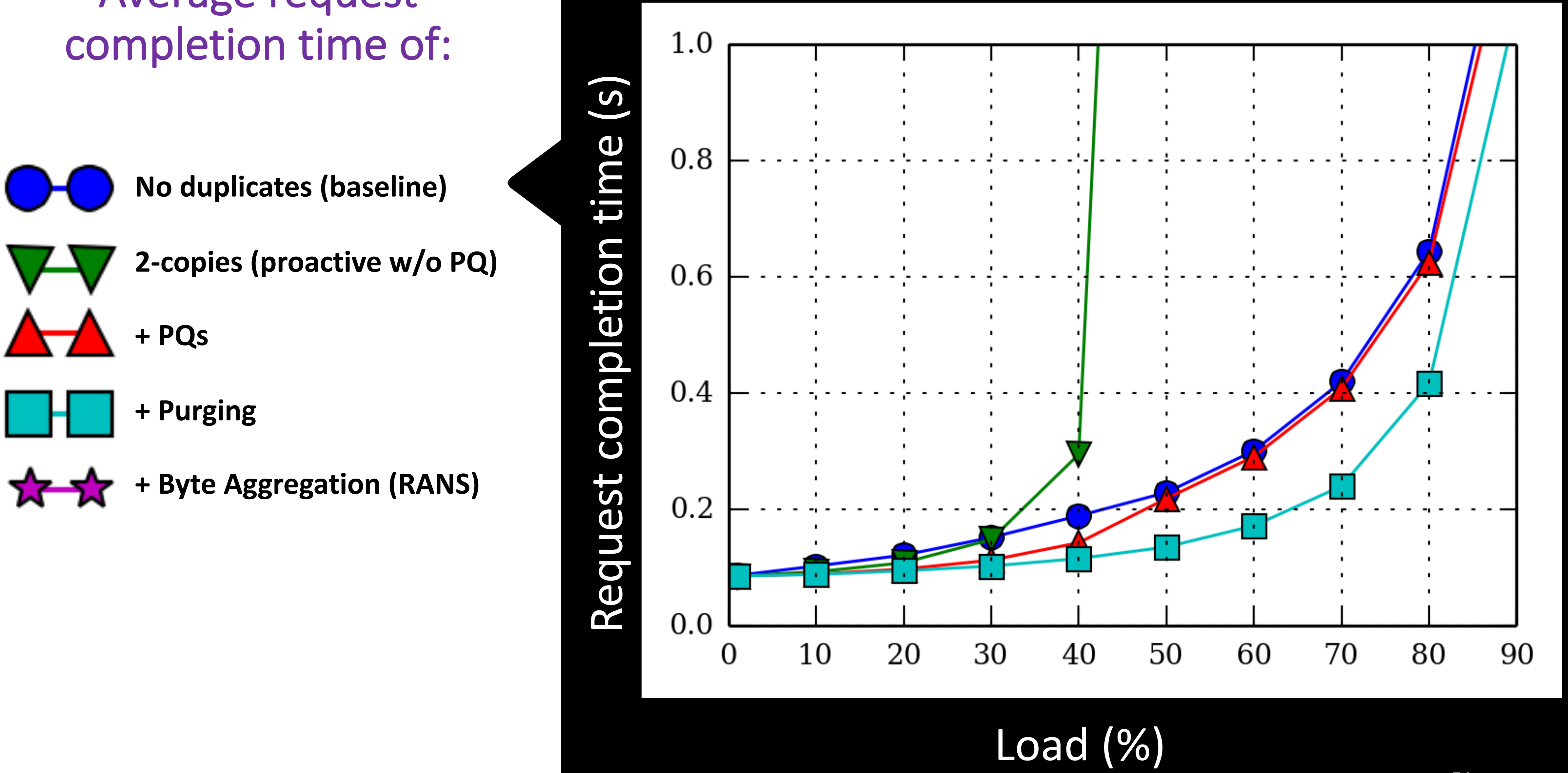
Average request completion time of:



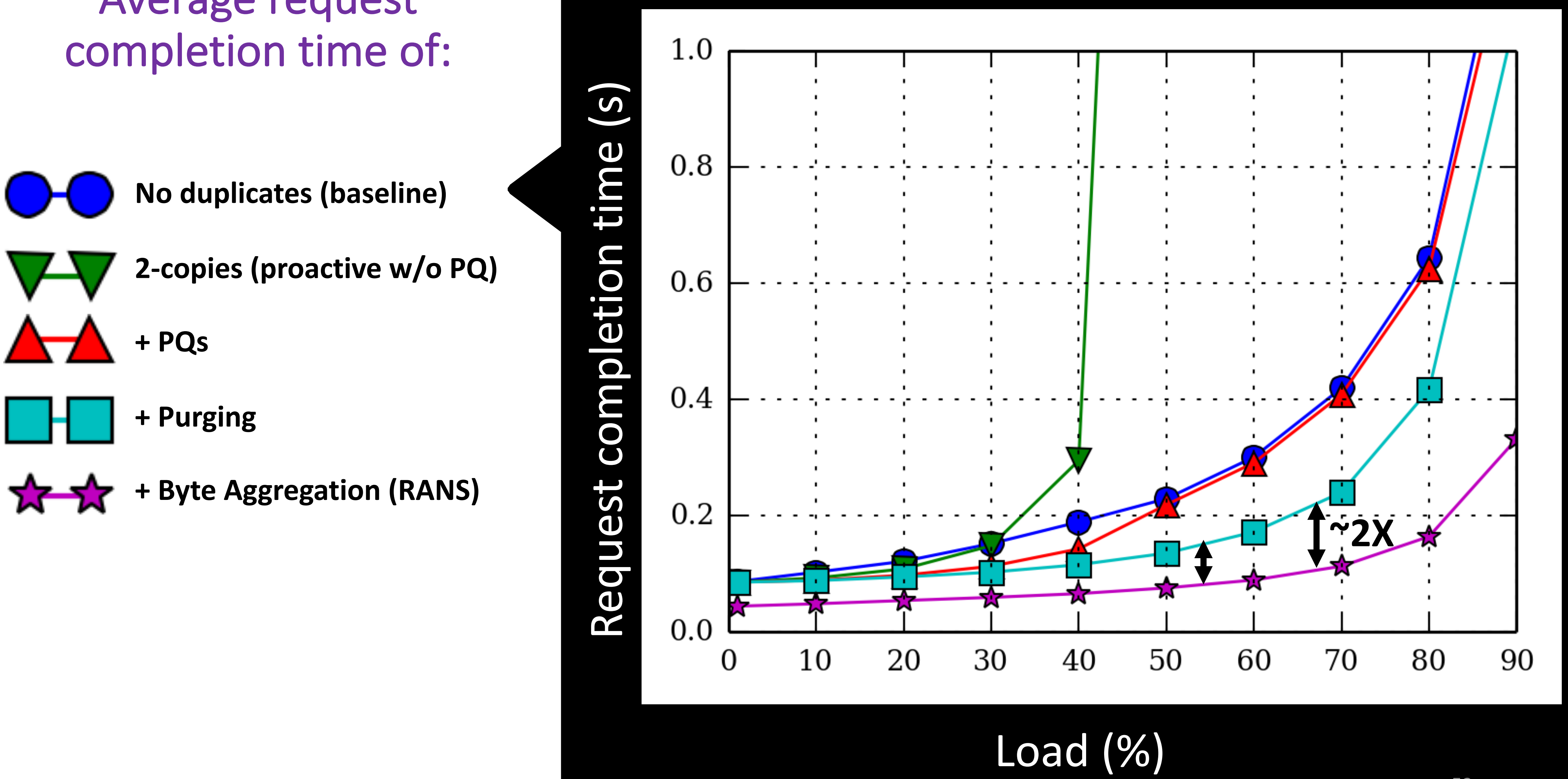
Average request completion time of:



Average request completion time of:



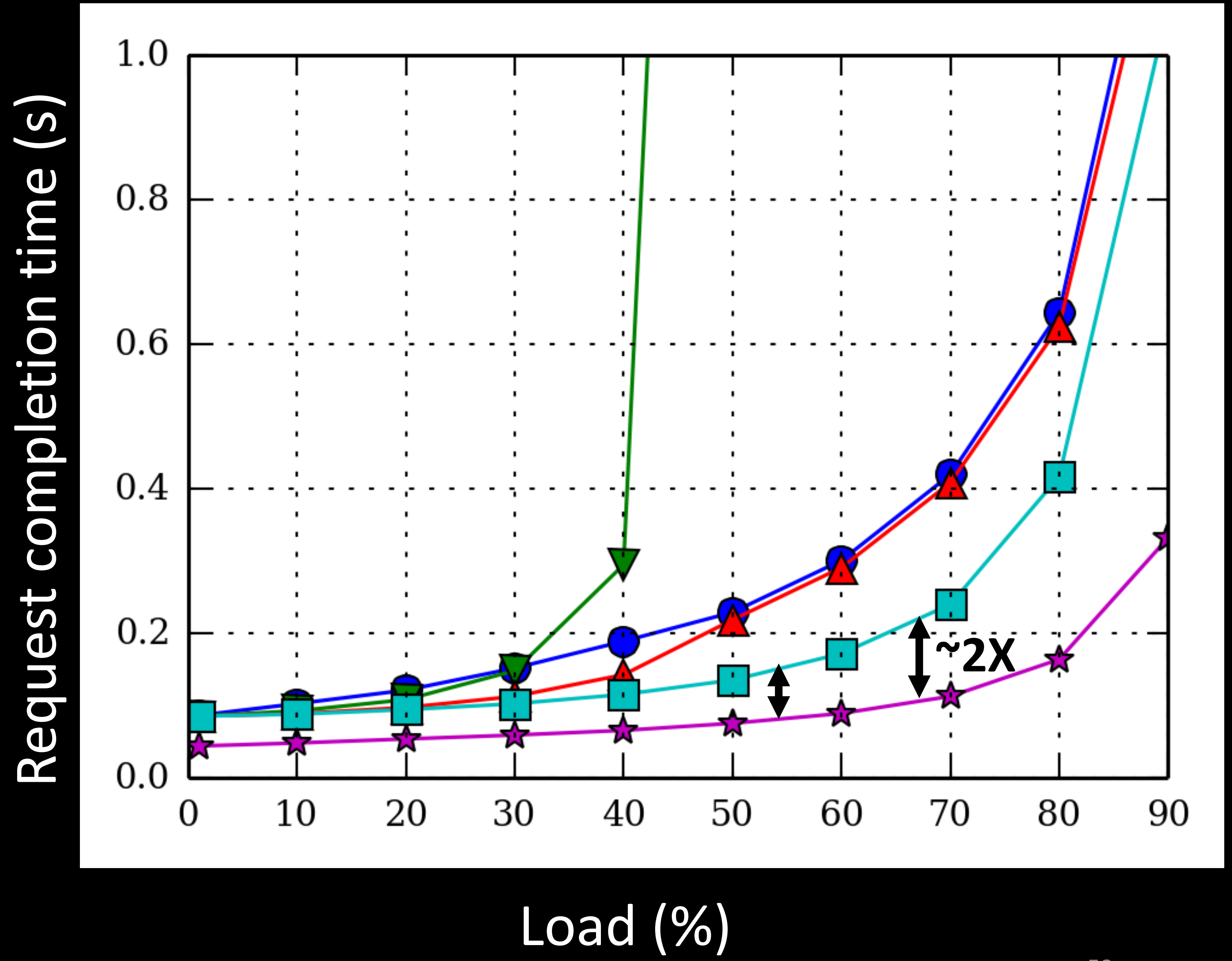
Average request completion time of:



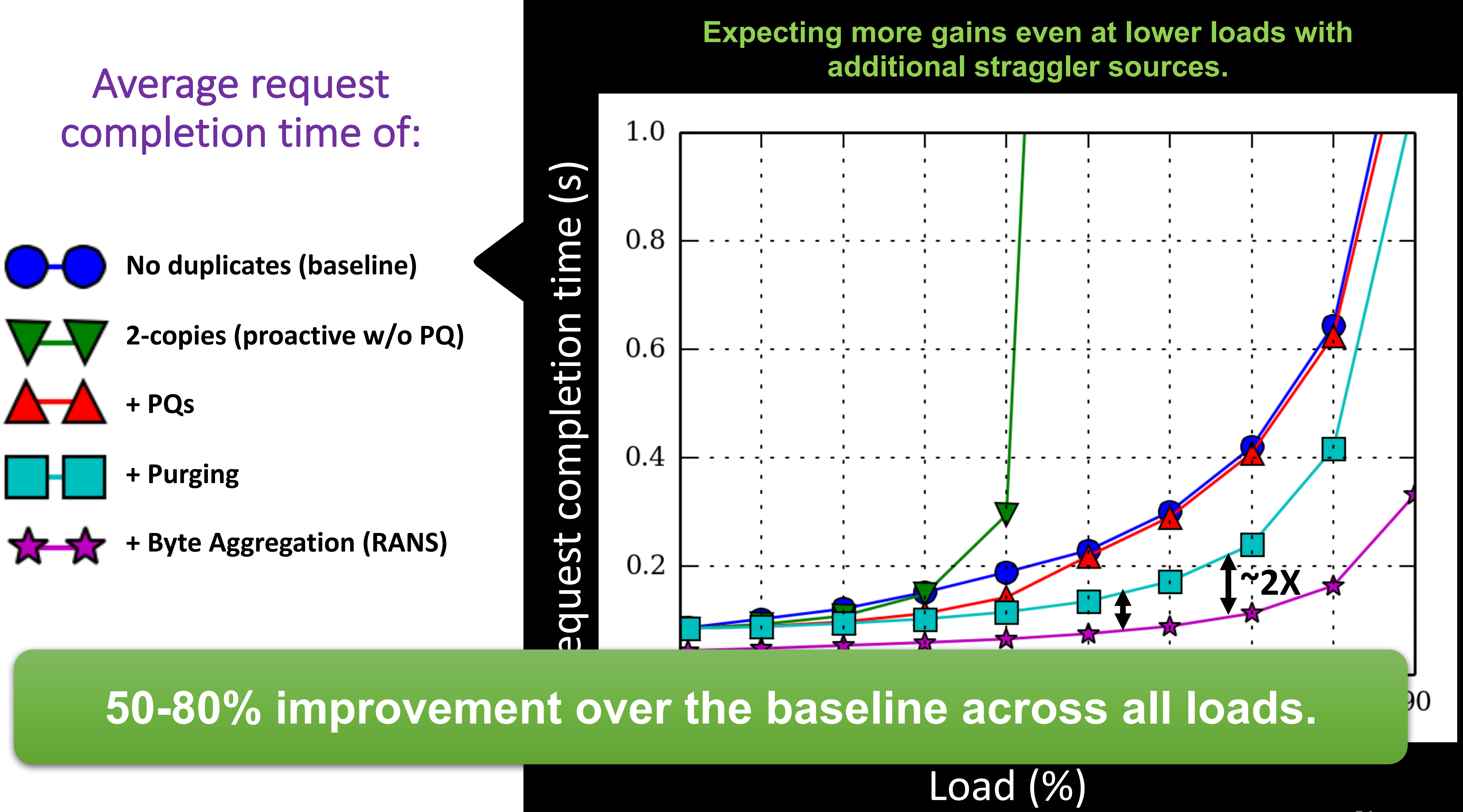
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
- + PQs
- + Purging
- + Byte Aggregation (RANS)

Expecting more gains even at lower loads with additional straggler sources.



Average request completion time of:



50-80% improvement over the baseline across all loads.

Summary & Future work

- **The Issue of Stragglers**
- **Duplicate-Aware Scheduling Framework**

  Simple yet challenging solution
- **RANS**

  A first step towards a duplicate-aware network
- **Implementing in HDFS and Cassandra**

RANS: Feedback and Discussion

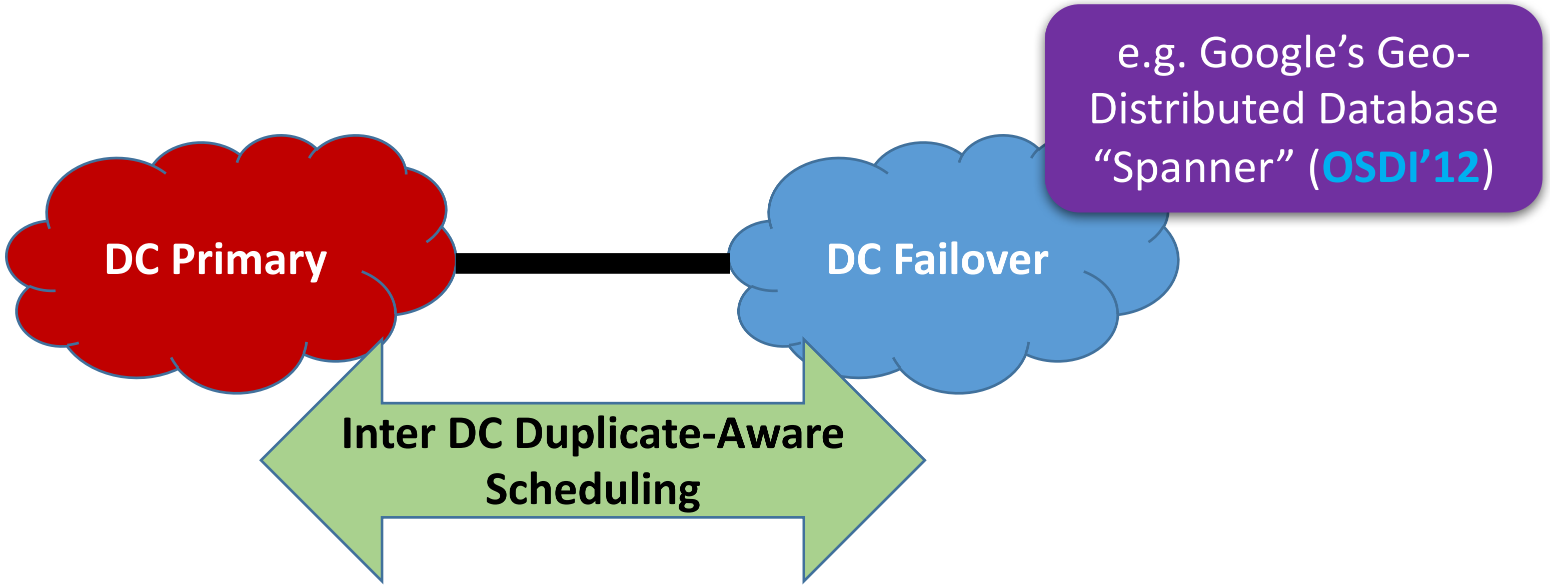
- **Ali Musa Iftikhar (musa@cs.tufts.edu)**
- **Fahad R. Dogar (fahad@cs.tufts.edu)**
- **Ihsan A. Qazi (ihsan.qazi@lums.edu.pk)**

Layer	**New Role**
Application	Duplicate-Awareness
Expressive Interface	
Transport	Point to multipoint Byte aggregation Priority assignment + purging
Network	Priority Queues + purging
Link	Same as before
Physical	Same as before

Possible questions – backup slide

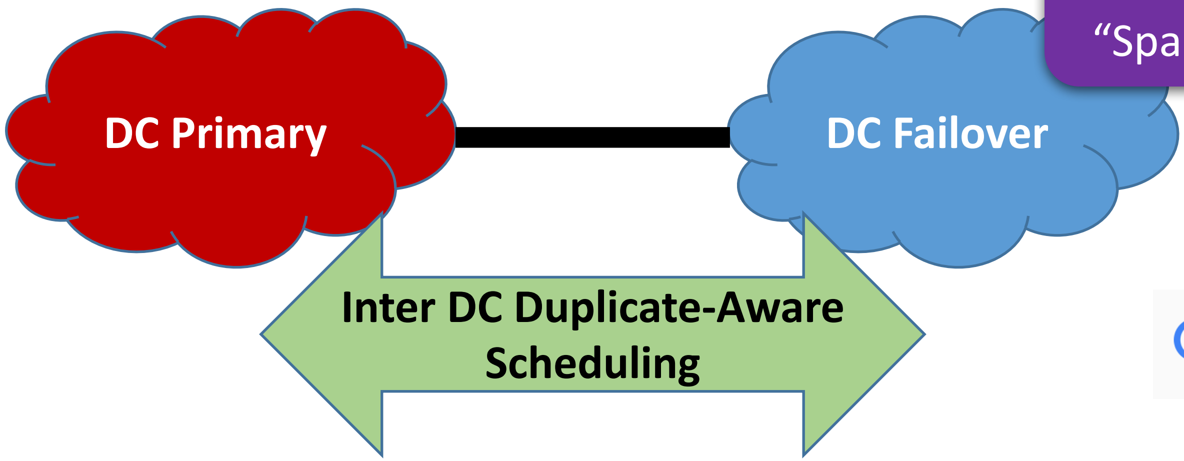
- Preemption overhead
  - Not really an issue in the network because packets are small.
- Packet purging
  - PFC (back pressure, build queues at the end hosts and purge them)
  - Drop the entire duplicate queue (easier than per-packet drops)
  - Recent trend towards programmable switches
- Gains with PQ
  - More gains with failures as stragglers (primary undergoes a failure)
  - Also more benefits with different resources
- Duplication overhead at client
  - Client is usually not the bottleneck
- Non-Idempotent requests
  - We are targeting the class of apps which have flexible end points and require at least once semantics
- Replicating only small packets and prioritizing them
  - Only beneficial with bursty small flows
  - HDFS have a typical chunk size b/w 64MB-128MB
- Quorum systems
  - RANS complements such systems, they can use this technique and send K out of N requests at high prio while N-K as backups
- Can’t just implement at the app and get the same benefits?
  - Network could be a bottleneck
  - Fine grained control, much more control
- Root causes of performance improvement
  - PQ avoids overheads
  - Now we can easily get the benefits of duplications like aggregation etc.
  - Purging will also at times purge primary making the system more efficient.

Food for thought



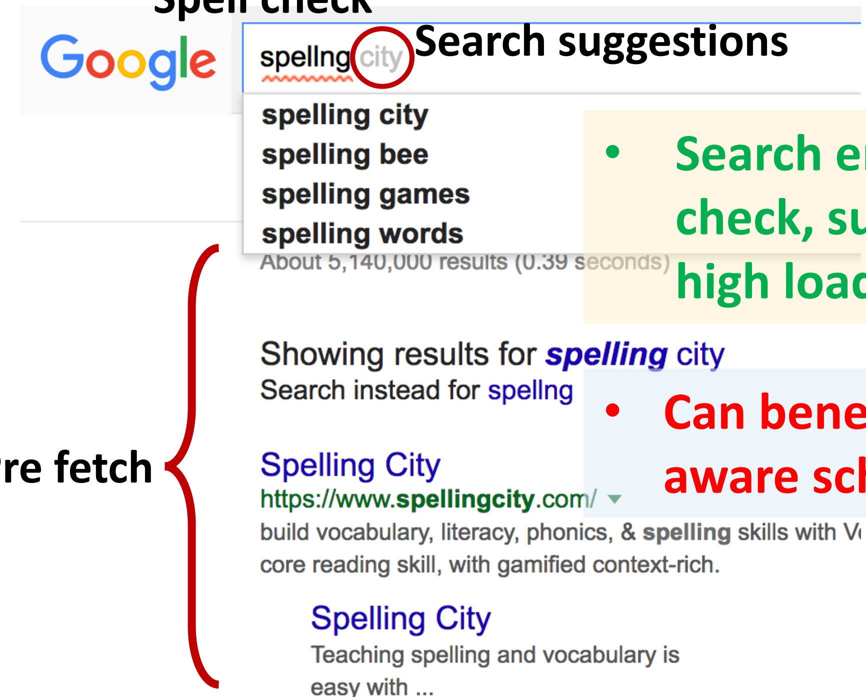
Food for thought

e.g. Google's Geo-Distributed Database "Spanner" (OSDI'12)



Spell check

Search suggestions



Pre fetch

- Search engines drop spell check, suggestions, etc. at high loads.
- Can benefit from duplicate-aware scheduling.

When RANS works best?

- Application fanout is high and stragglers are frequent.
- End-points are flexible and “at least once” semantics are sufficient.
- Client is not the bottleneck.
- Request sizes are small (or preemption overhead is minimal).