



# Achieving Rapid Response Times in Large Online Services

Jeff Dean

Google Fellow

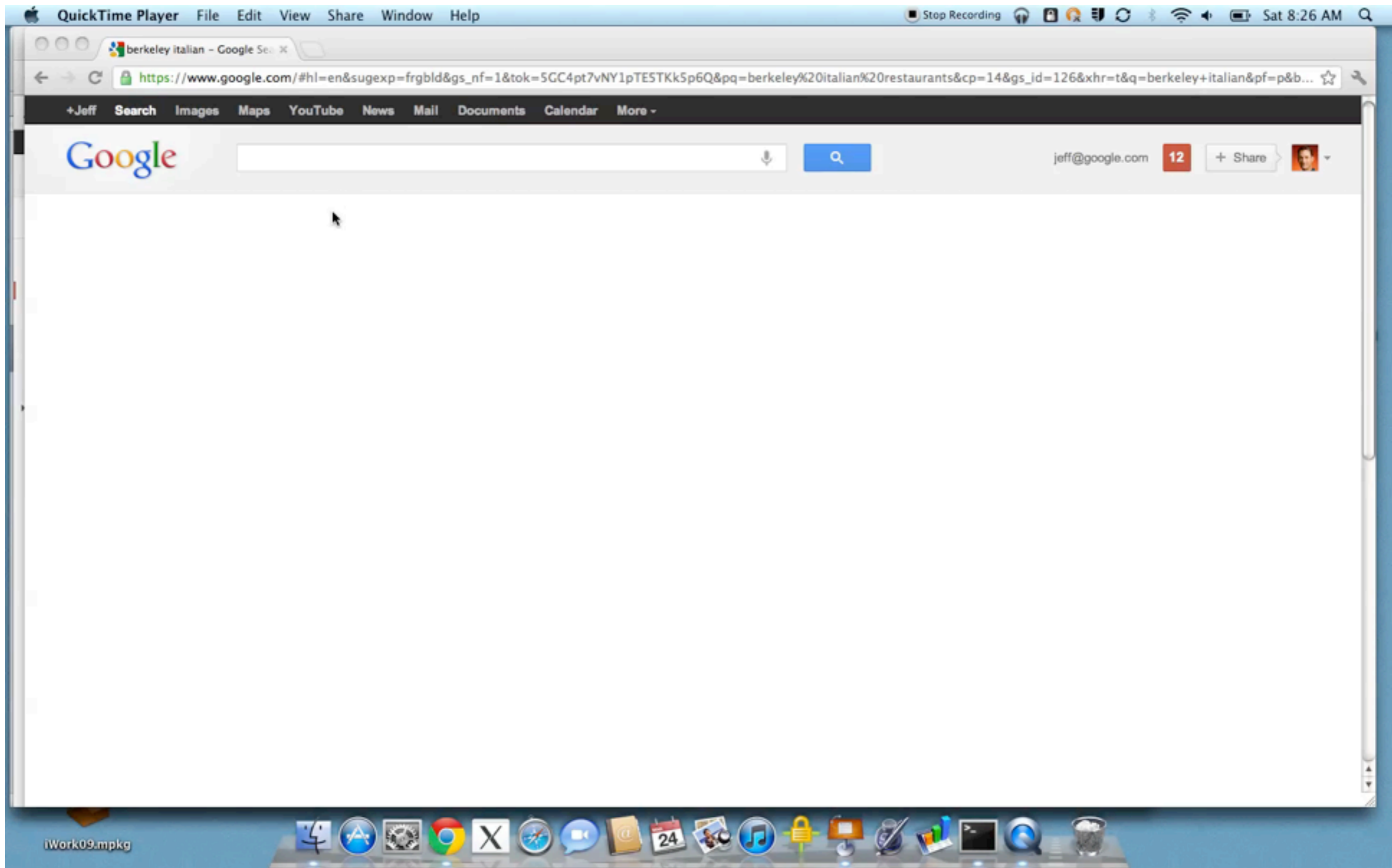
[jeff@google.com](mailto:jeff@google.com)

# Faster Is Better

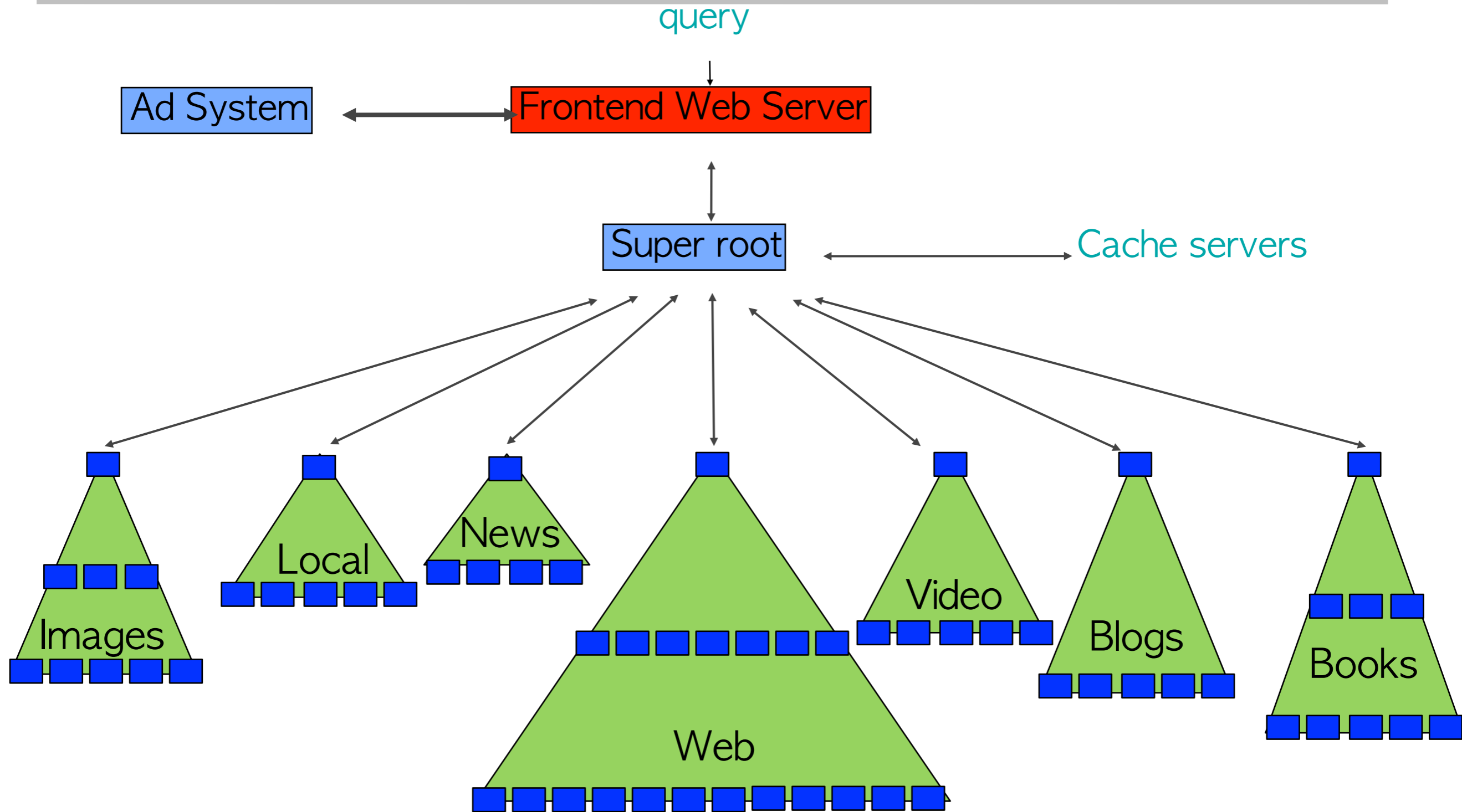
---



# Faster Is Better



# Large Fanout Services



# Why Does Fanout Make Things Harder?

---

- Overall latency  $\geq$  latency of slowest component
  - small blips on individual machines cause delays
  - touching more machines increases likelihood of delays
- Server with 1 ms avg. but 1 sec 99%ile latency
  - touch 1 of these: 1% of requests take  $\geq 1$  sec
  - touch 100 of these: 63% of requests take  $\geq 1$  sec



# One Approach: Squash All Variability

---

- Careful engineering all components of system
- Possible at small scale
  - dedicated resources
  - complete control over whole system
  - careful understanding of all background activities
  - less likely to have hardware fail in bizarre ways
- System changes are difficult
  - software or hardware changes affect delicate balance

**Not tenable at large scale: need to share resources**



# Shared Environment

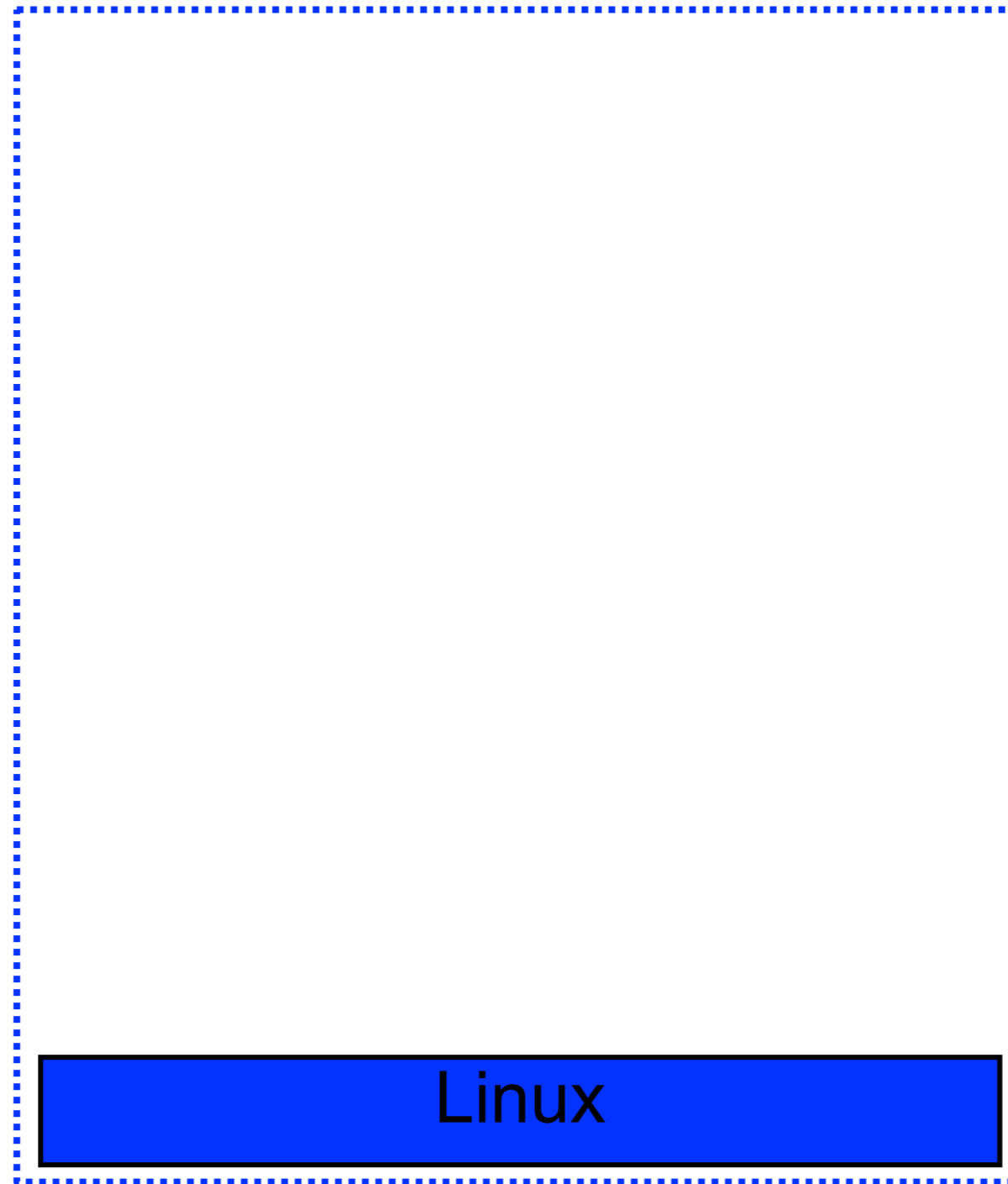
---

- Huge benefit: greatly increased utilization
- ... but hard to predict effects increase variability
  - network congestion
  - background activities
  - bursts of foreground activity
  - not just your jobs, but everyone else's jobs, too
- Exacerbated by large fanout systems



# Shared Environment

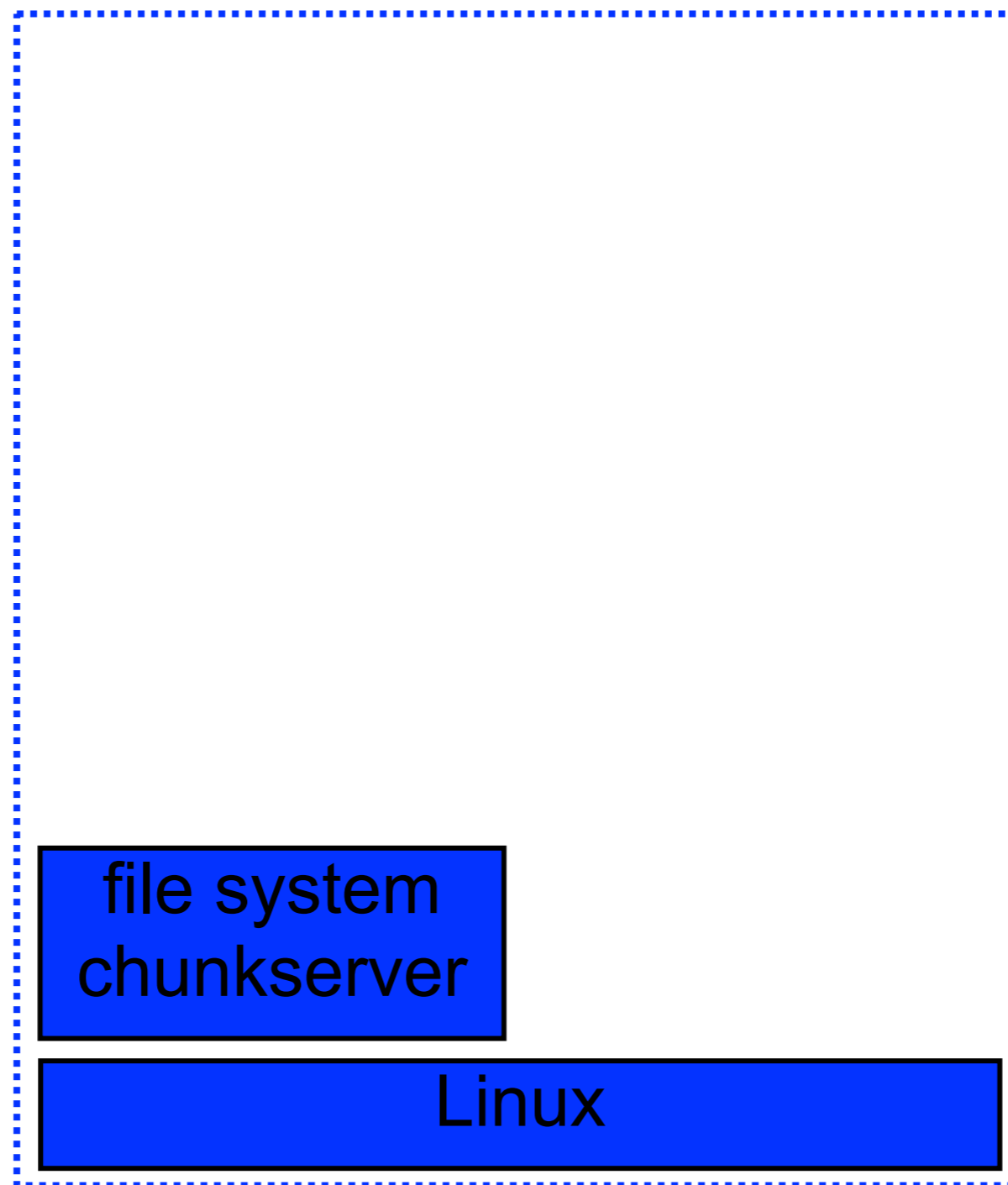
---





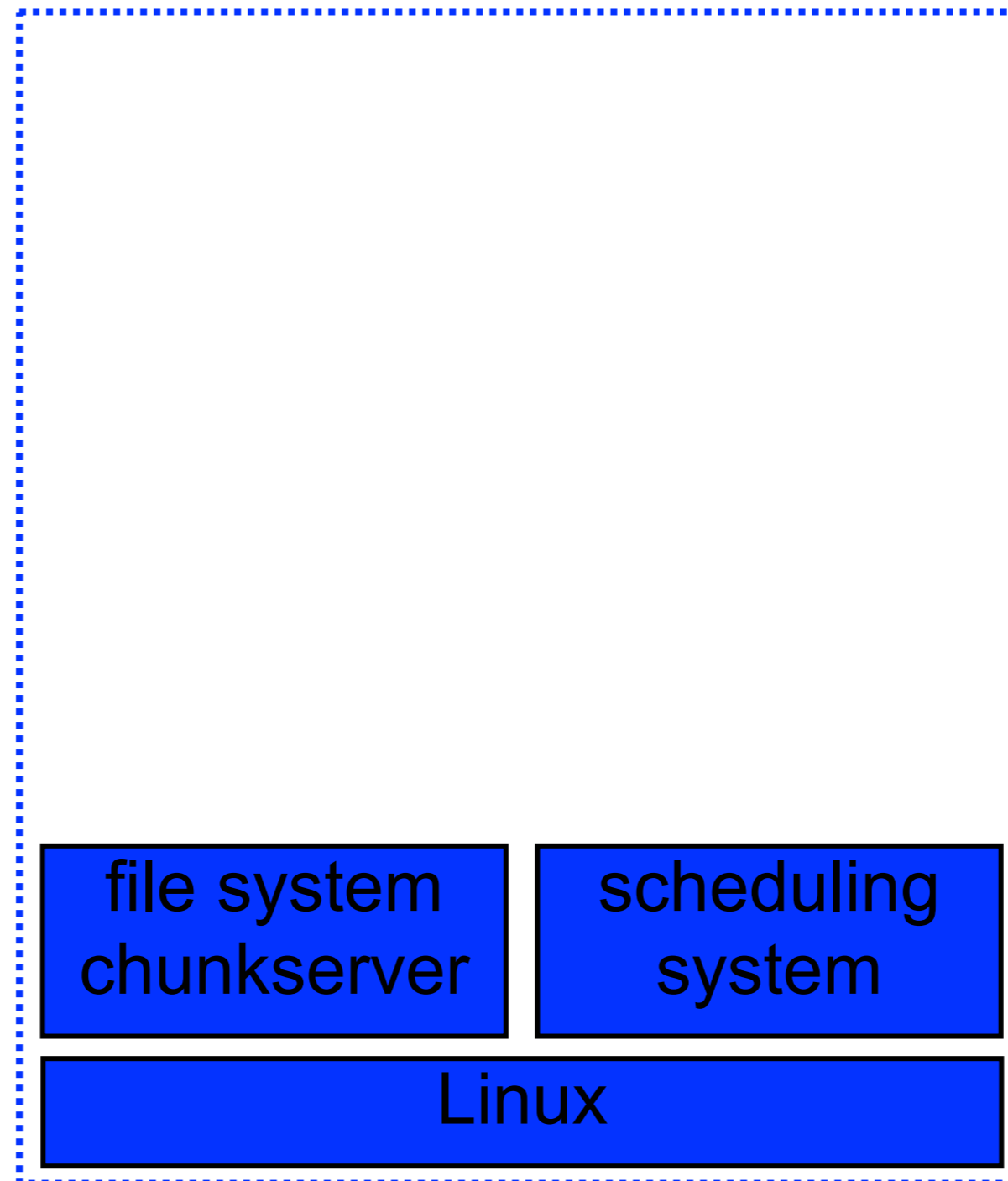
# Shared Environment

---



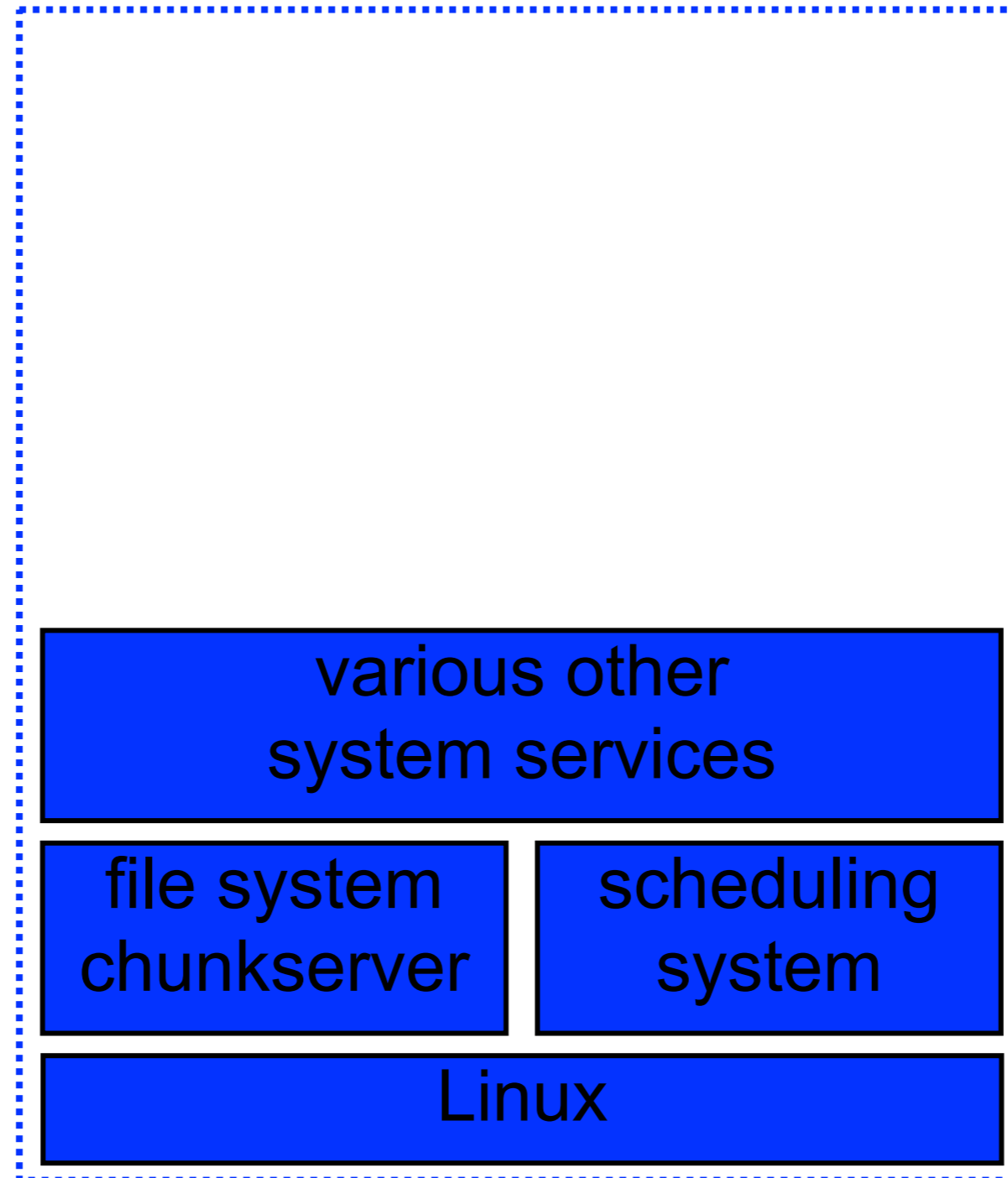
# Shared Environment

---



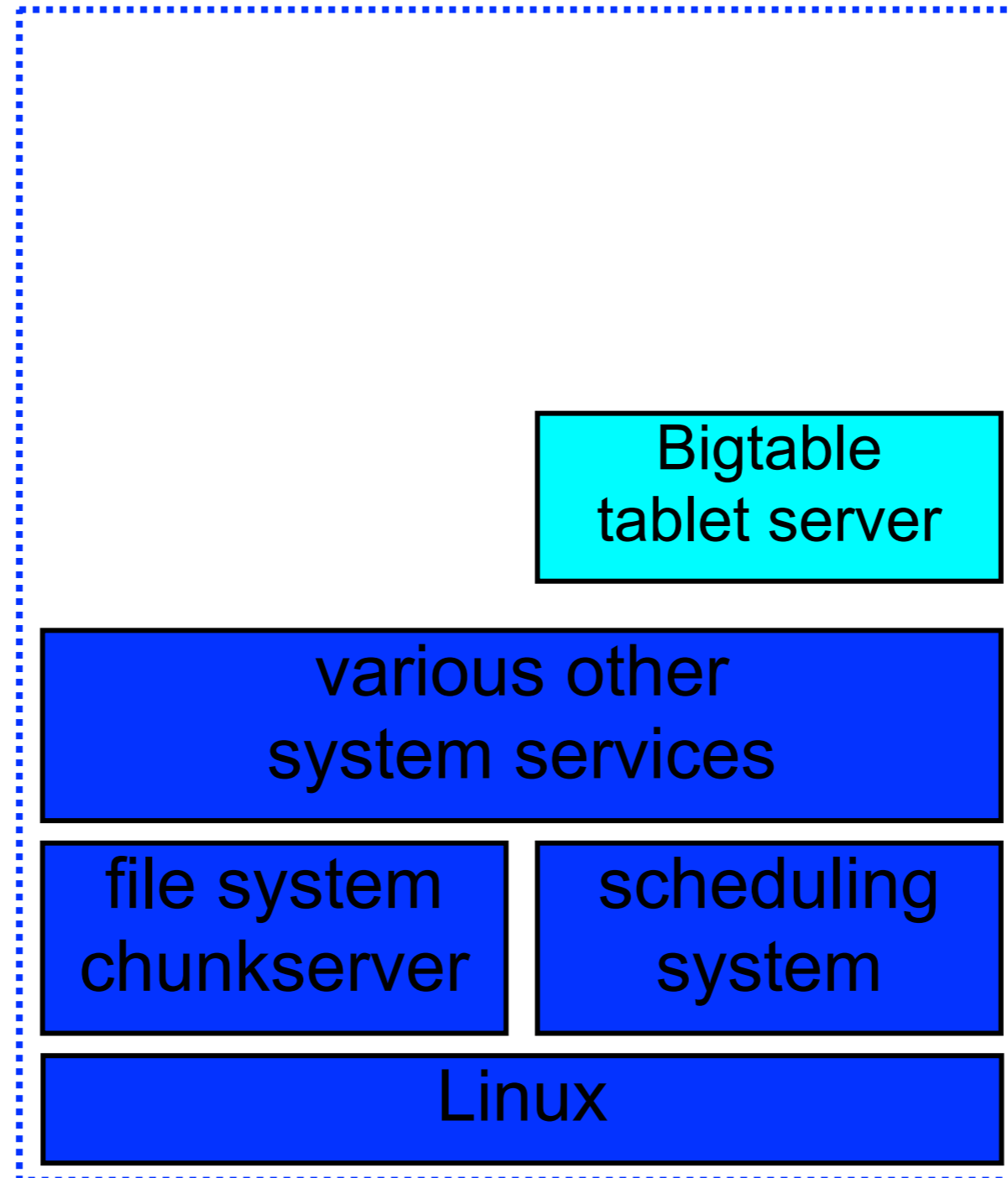
# Shared Environment

---



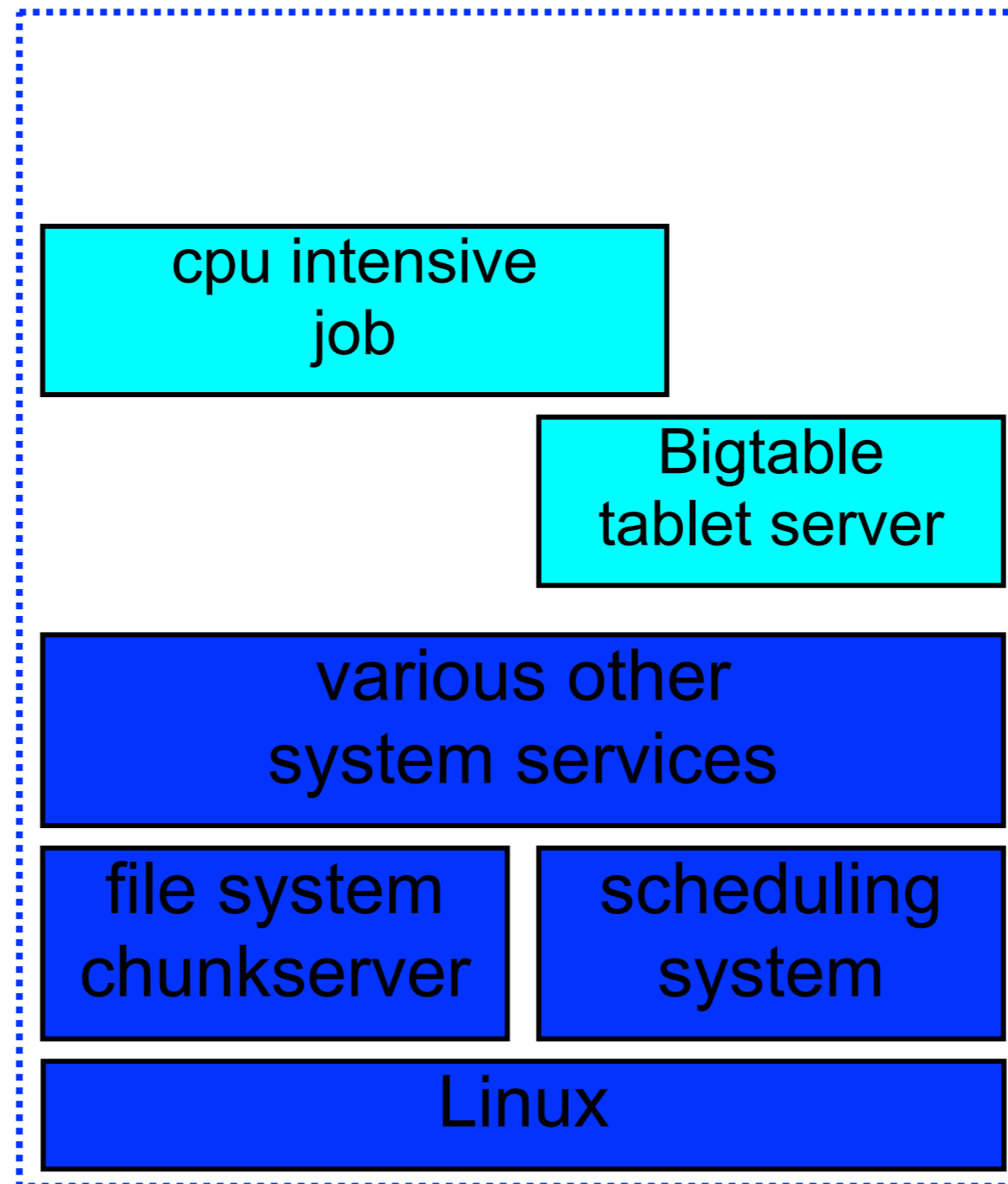
# Shared Environment

---



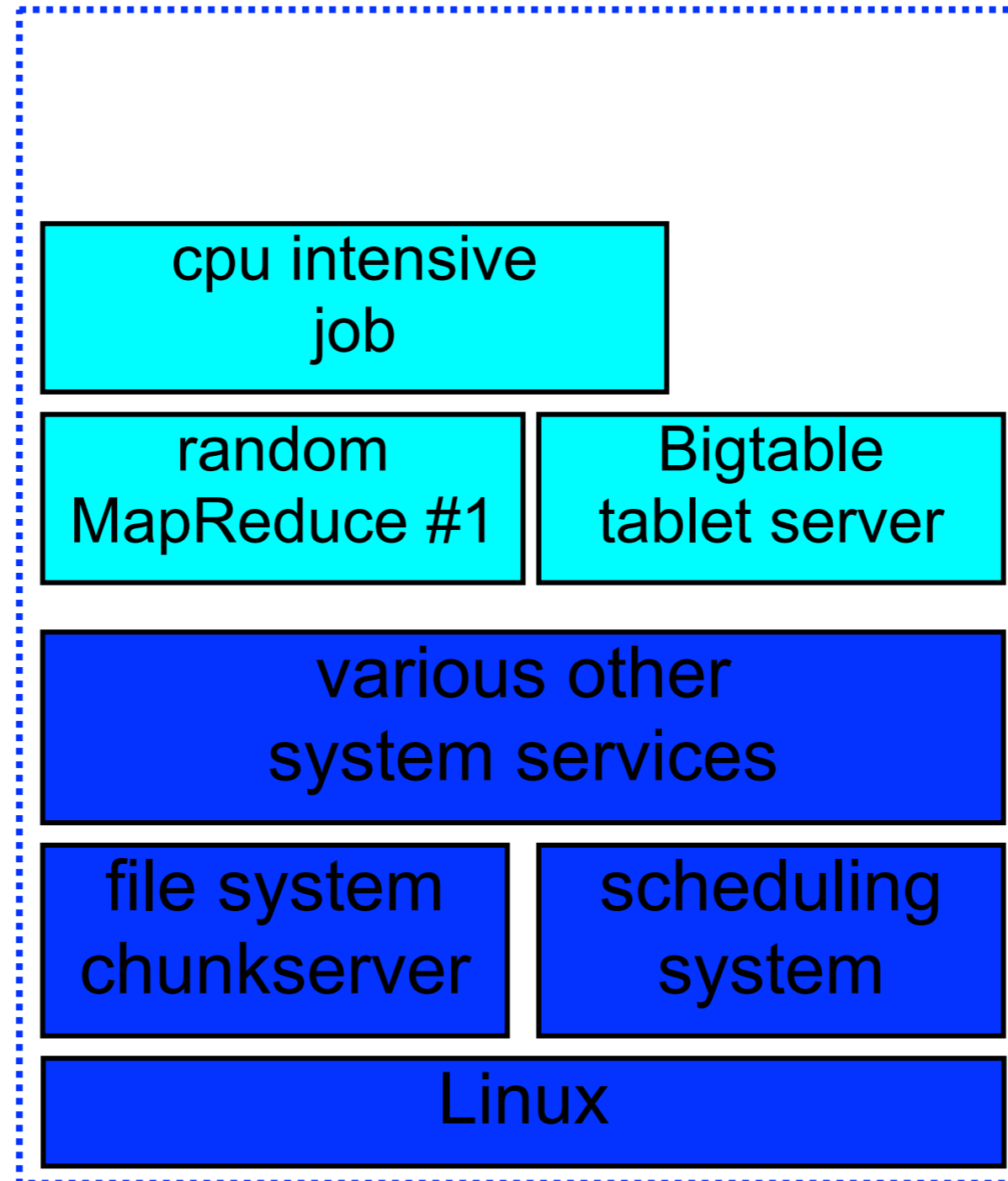
# Shared Environment

---



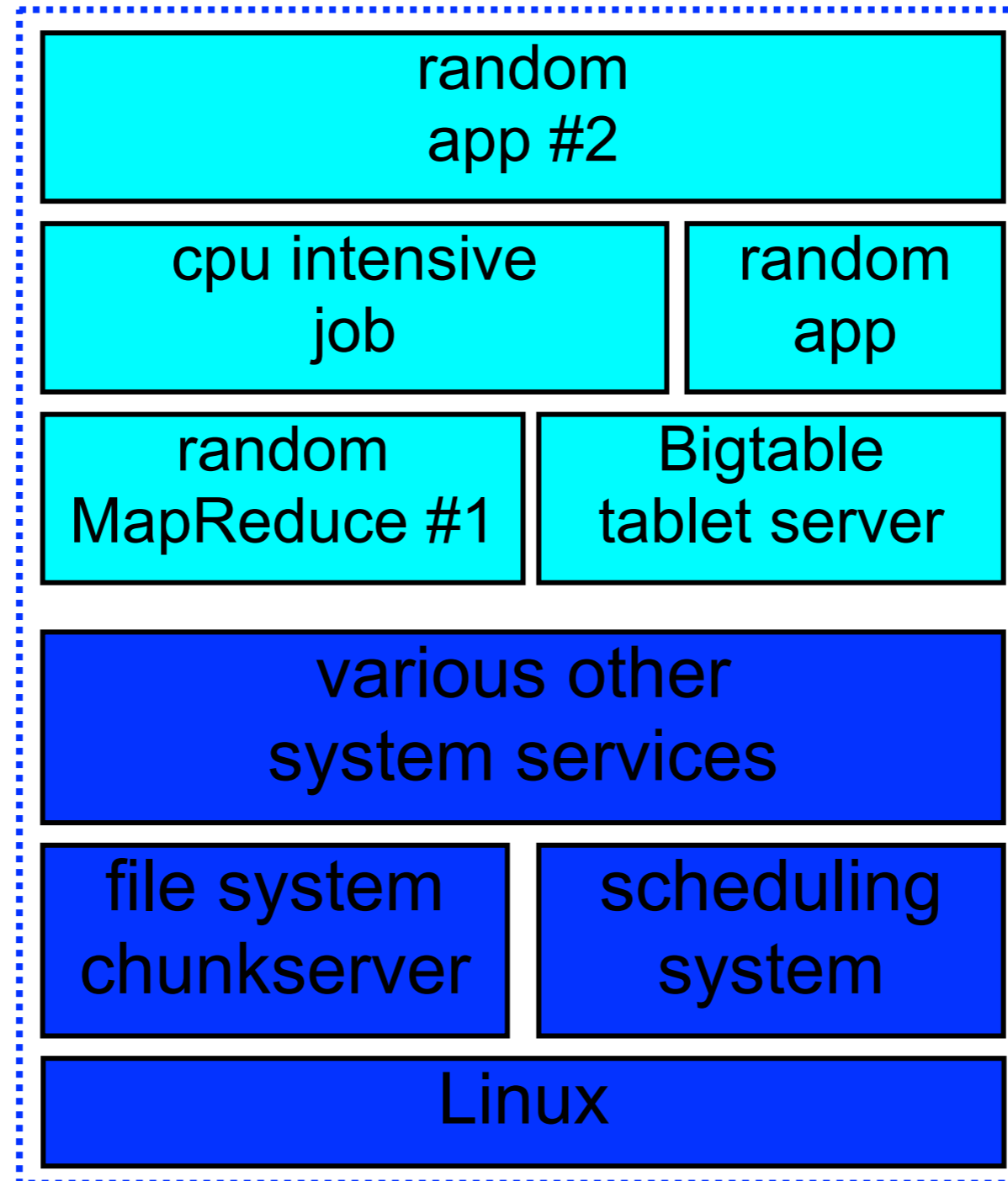
# Shared Environment

---



# Shared Environment

---



# Basic Latency Reduction Techniques

---

- Differentiated service classes
  - prioritized request queues in servers
  - prioritized network traffic
- Reduce head-of-line blocking
  - break large requests into sequence of small requests
- Manage expensive background activities
  - e.g. log compaction in distributed storage systems
  - rate limit activity
  - defer expensive activity until load is lower





# Synchronized Disruption

---

- Large systems often have background daemons
  - various monitoring and system maintenance tasks
- Initial intuition: randomize when each machine performs these tasks
  - actually a very bad idea for high fanout services
    - at any given moment, at least one or a few machines are slow
- Better to actually synchronize the disruptions
  - run every five minutes “on the dot”
  - one synchronized blip better than unsynchronized



# Tolerating Faults vs. Tolerating Variability

---

- Tolerating faults:
  - rely on extra resources
    - RAIDed disks, ECC memory, dist. system components, etc.
  - ***make a reliable whole out of unreliable parts***
- Tolerating variability:
  - use these same extra resources
  - ***make a predictable whole out of unpredictable parts***
- Times scales are very different:
  - variability: 1000s of disruptions/sec, scale of **milliseconds**
  - faults: 10s of failures per day, scale of **tens of seconds**



# Latency Tolerating Techniques

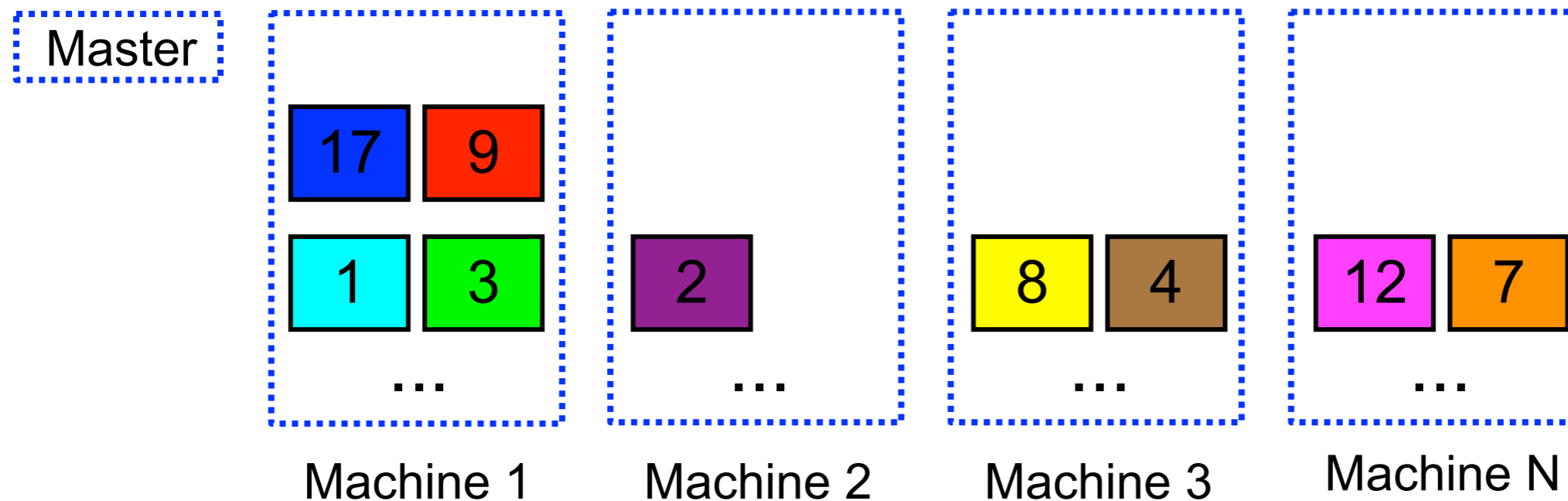
---

- **Cross request adaptation**
  - examine recent behavior
  - take action to improve latency of future requests
  - typically relate to balancing load across set of servers
  - time scale: 10s of seconds to minutes
- **Within request adaptation**
  - cope with slow subsystems in context of higher level request
  - time scale: right now, while user is waiting



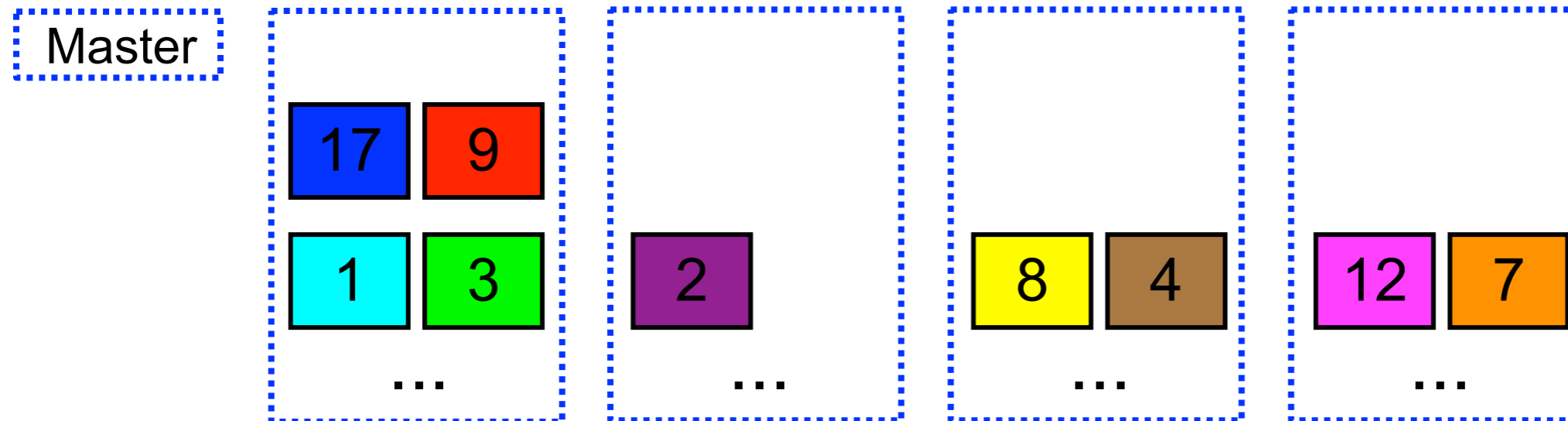
# Fine-Grained Dynamic Partitioning

- Partition large datasets/computations
  - more than 1 partition per machine (often 10-100/machine)
  - e.g. BigTable, query serving systems, GFS, ...



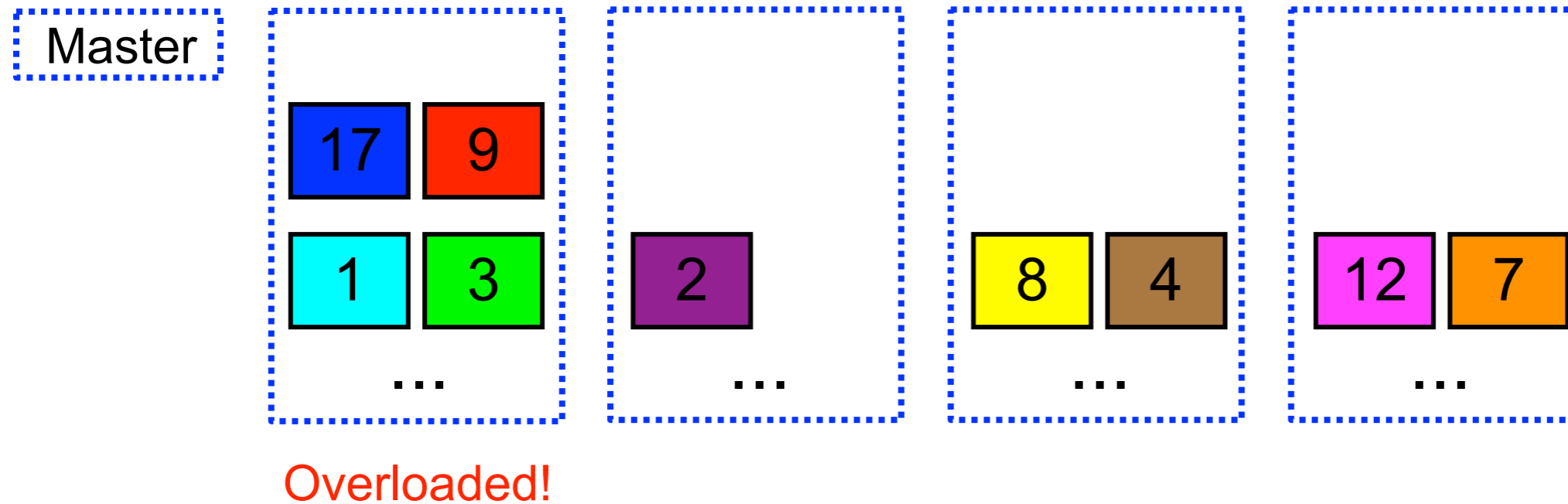
# Load Balancing

- Can shed load in few percent increments
  - prioritize shifting load when imbalance is more severe



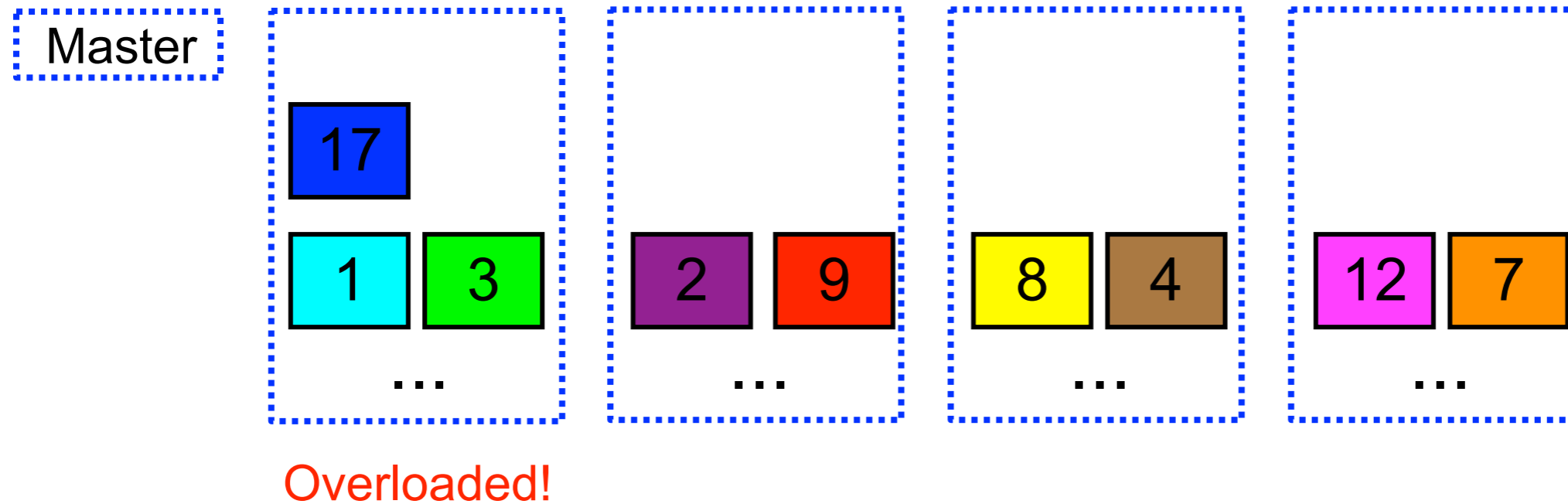
# Load Balancing

- **Can shed load in few percent increments**
  - prioritize shifting load when imbalance is more severe



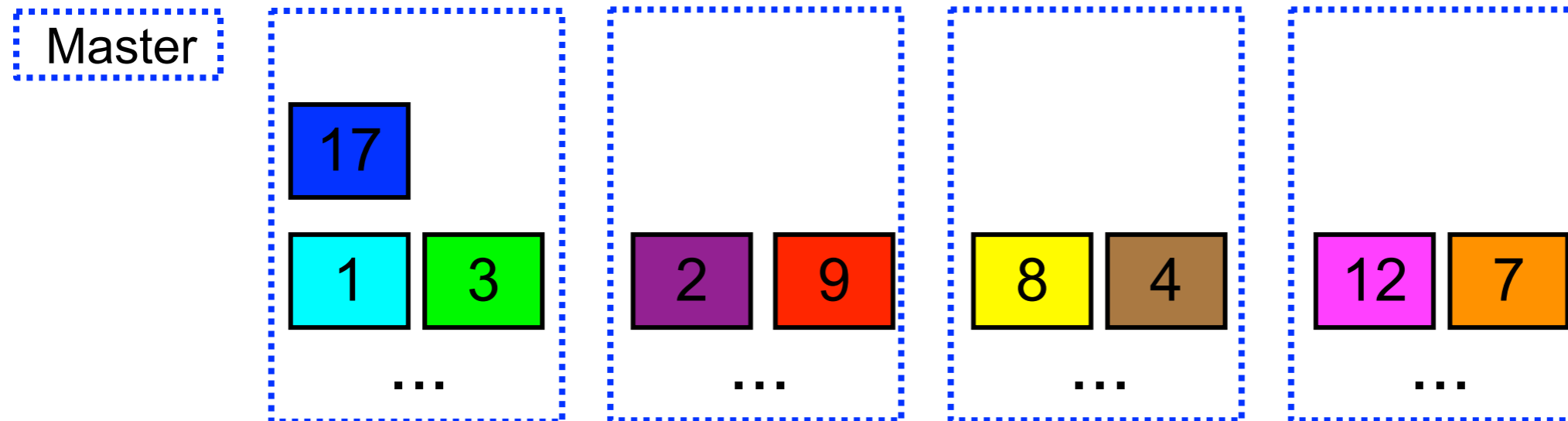
# Load Balancing

- **Can shed load in few percent increments**
  - prioritize shifting load when imbalance is more severe



# Load Balancing

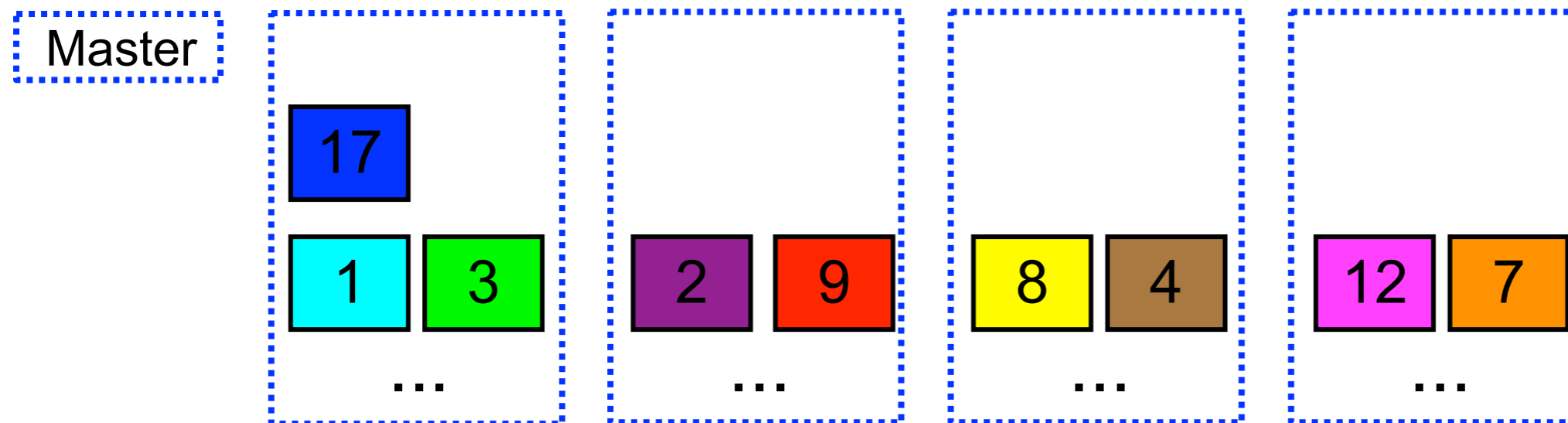
- **Can shed load in few percent increments**
  - prioritize shifting load when imbalance is more severe





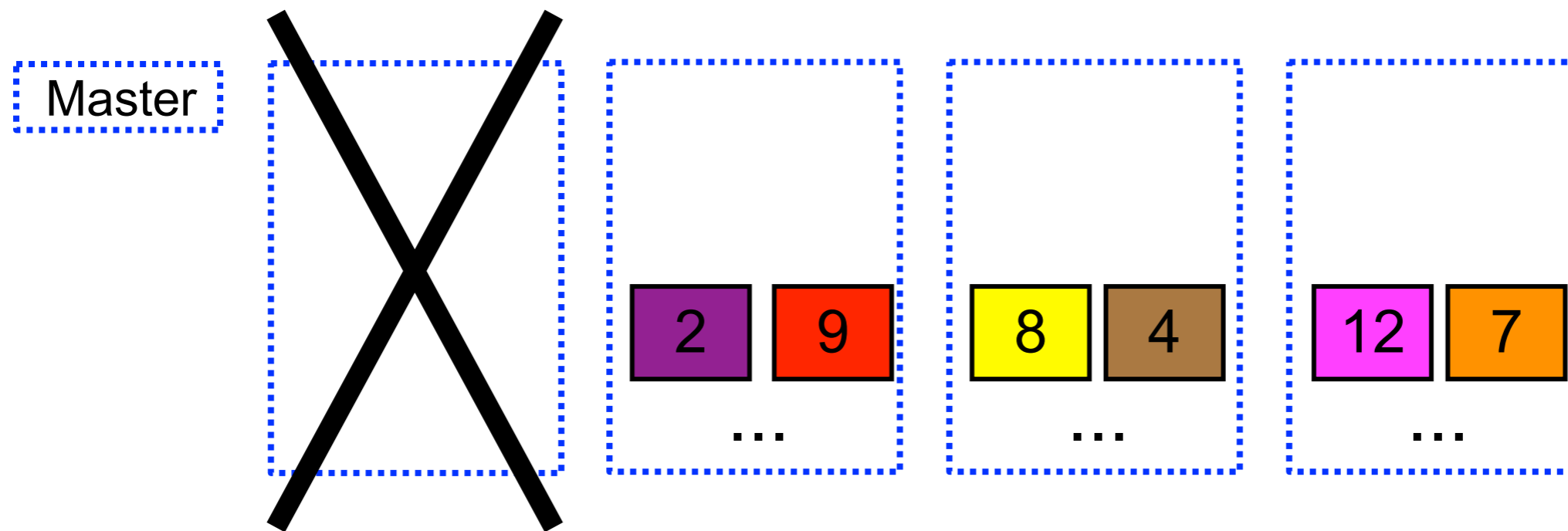
# Speeds Failure Recovery

- Many machines each recover one or a few partition
  - e.g. BigTable tablets, GFS chunks, query serving shards



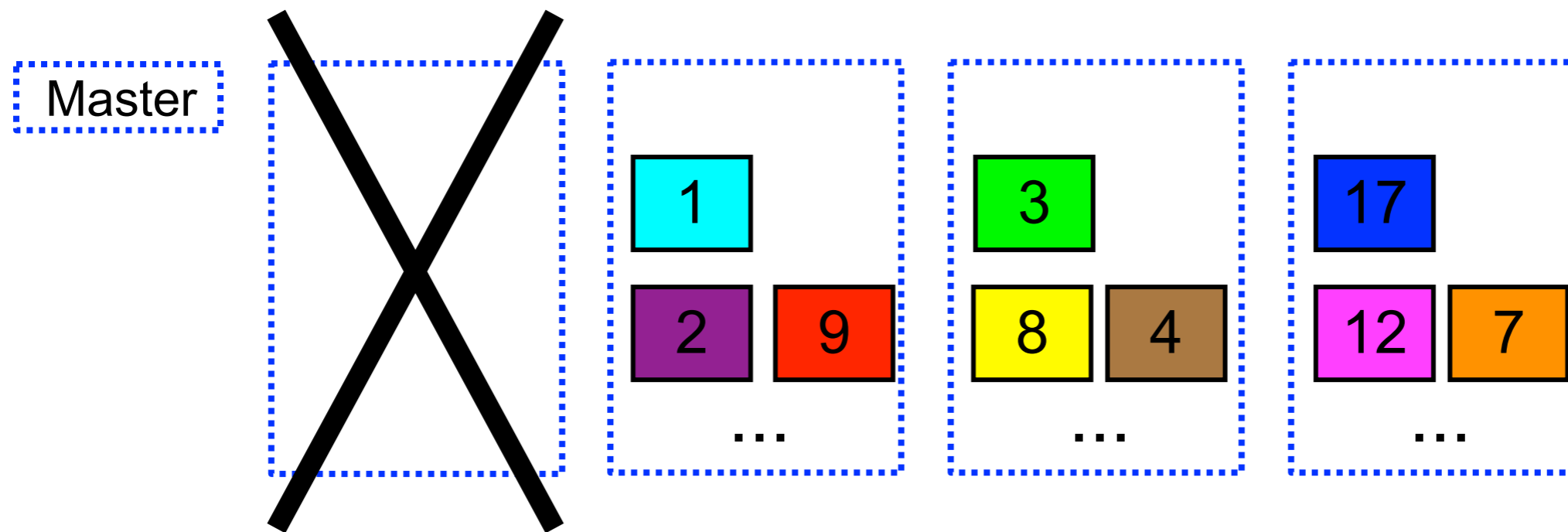
# Speeds Failure Recovery

- Many machines each recover one or a few partition
  - e.g. BigTable tablets, GFS chunks, query serving shards



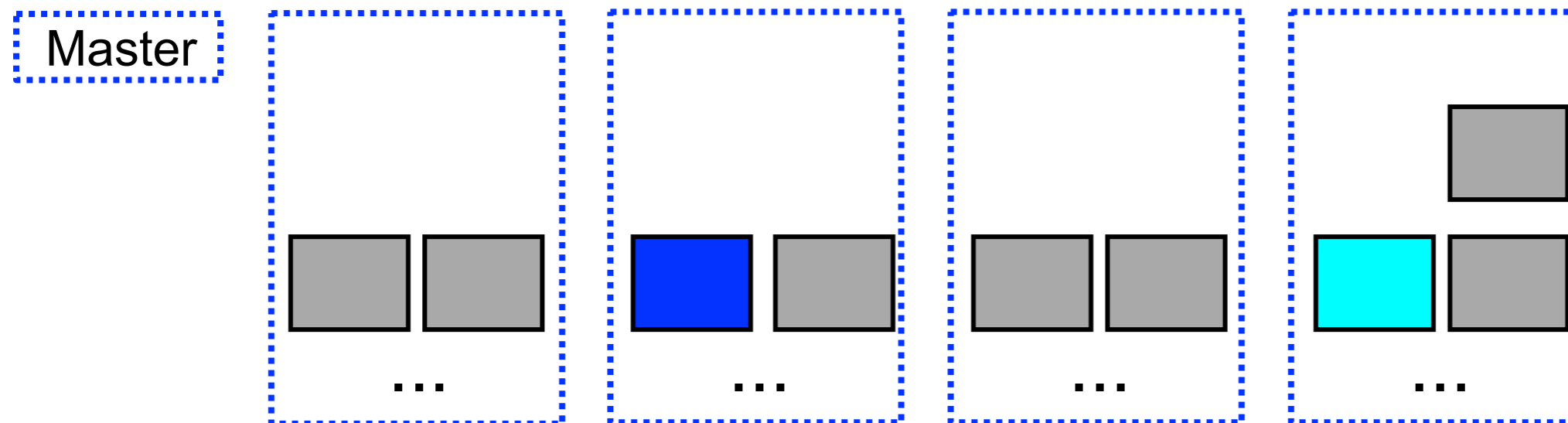
# Speeds Failure Recovery

- Many machines each recover one or a few partition
  - e.g. BigTable tablets, GFS chunks, query serving shards



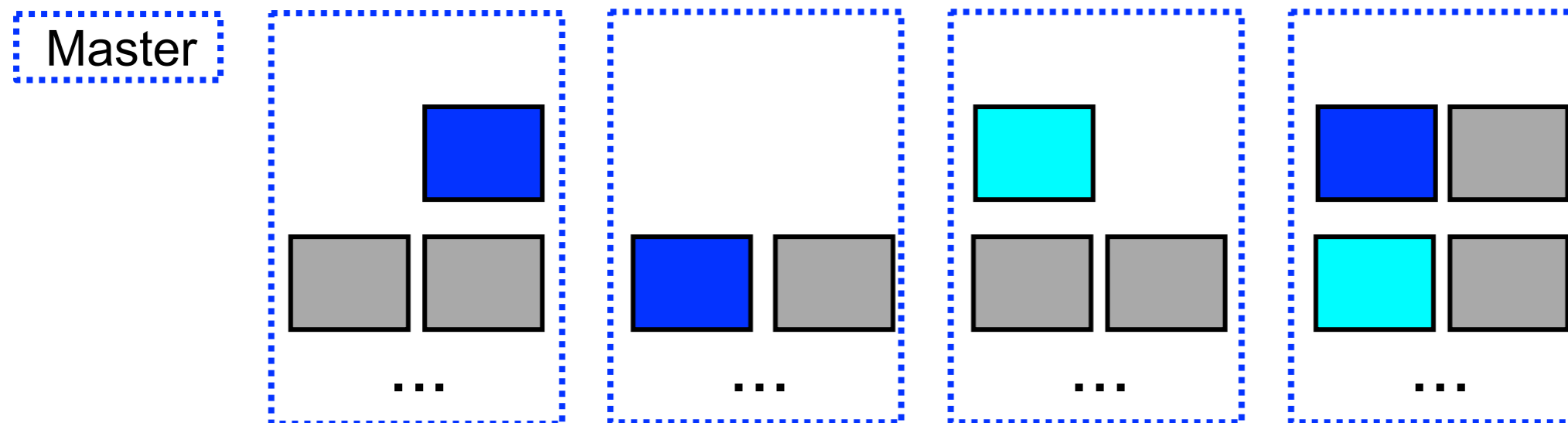
# Selective Replication

- Find heavily used items and make more replicas
  - can be static or dynamic
- Example: Query serving system
  - **static**: more replicas of important docs
  - **dynamic**: more replicas of Chinese documents as Chinese query load increases



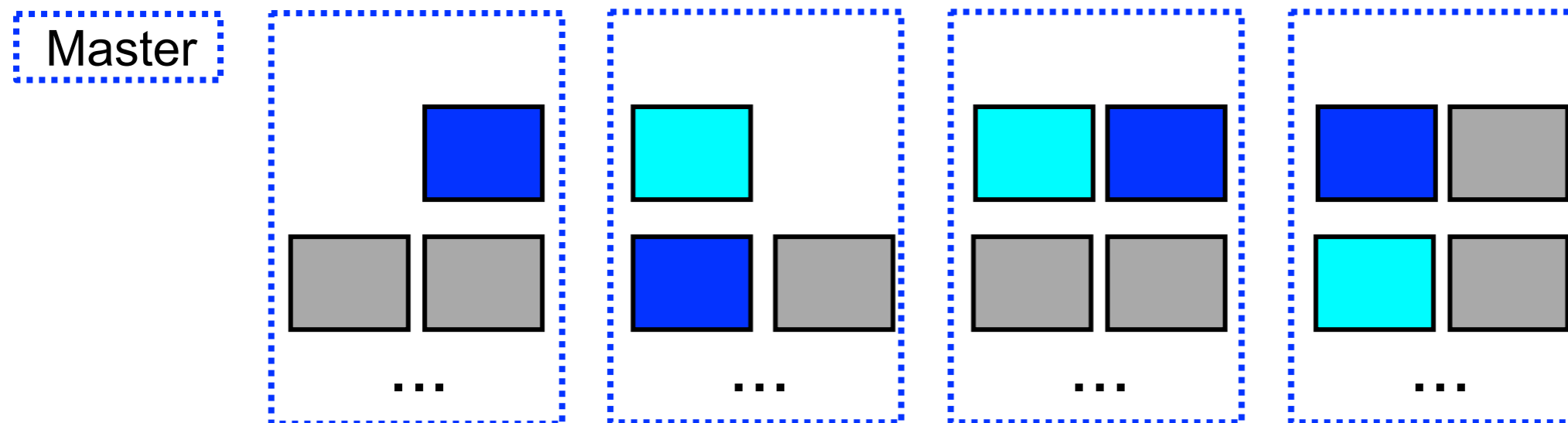
# Selective Replication

- Find heavily used items and make more replicas
  - can be static or dynamic
- Example: Query serving system
  - **static**: more replicas of important docs
  - **dynamic**: more replicas of Chinese documents as Chinese query load increases



# Selective Replication

- Find heavily used items and make more replicas
  - can be static or dynamic
- Example: Query serving system
  - **static**: more replicas of important docs
  - **dynamic**: more replicas of Chinese documents as Chinese query load increases



# Latency-Induced Probation

---

- Servers sometimes become slow to respond
  - could be data dependent, but...
  - often due to interference effects
    - e.g. CPU or network spike for other jobs running on shared server
- **Non-intuitive: remove capacity under load to improve latency (?!)**
- Initiate corrective action
  - e.g. make copies of partitions on other servers
  - continue sending shadow stream of requests to server
    - keep measuring latency
    - **return to service when latency back down for long enough**



# Handling Within-Request Variability

---

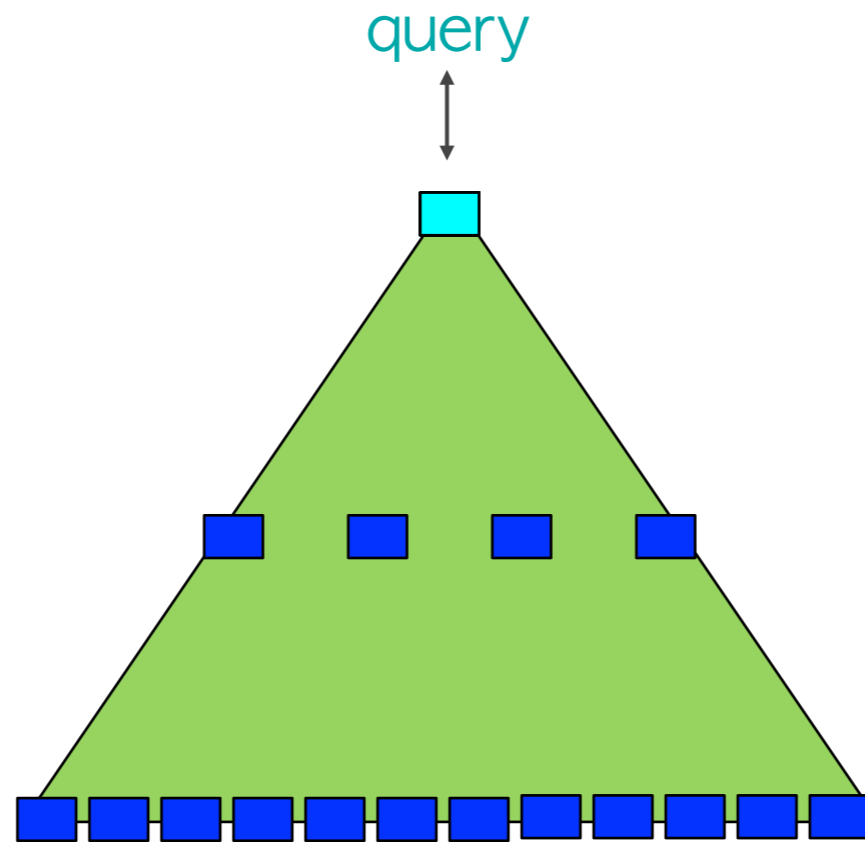
- Take action within single high-level request
- Goals:
  - reduce overall latency
  - don't increase resource use too much
  - keep serving systems safe





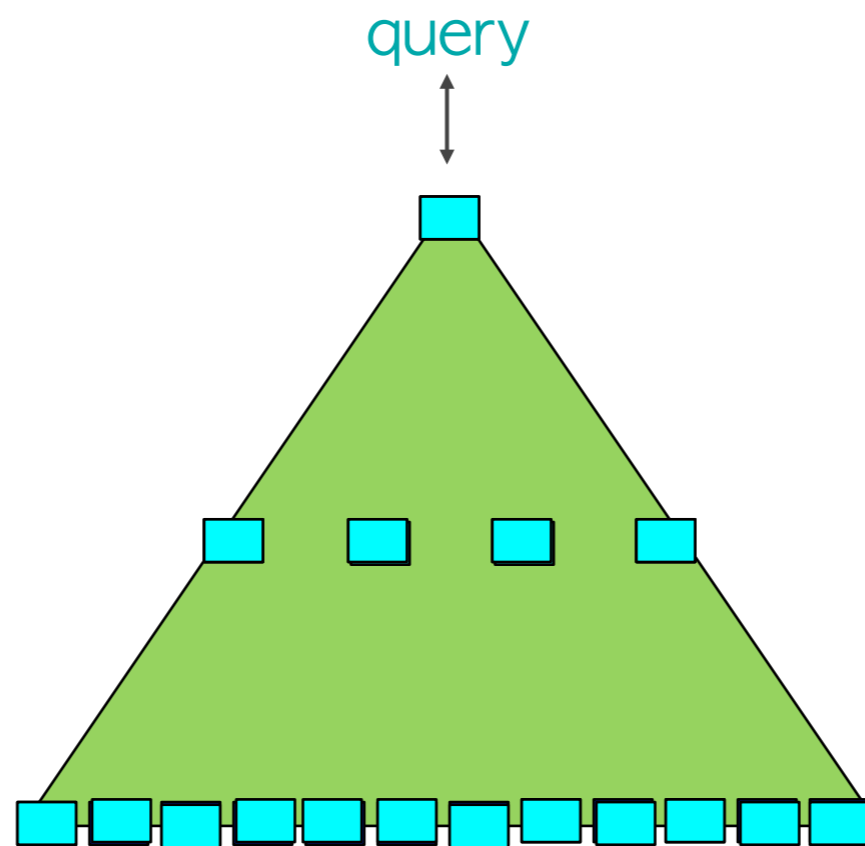
# Data Independent Failures

---



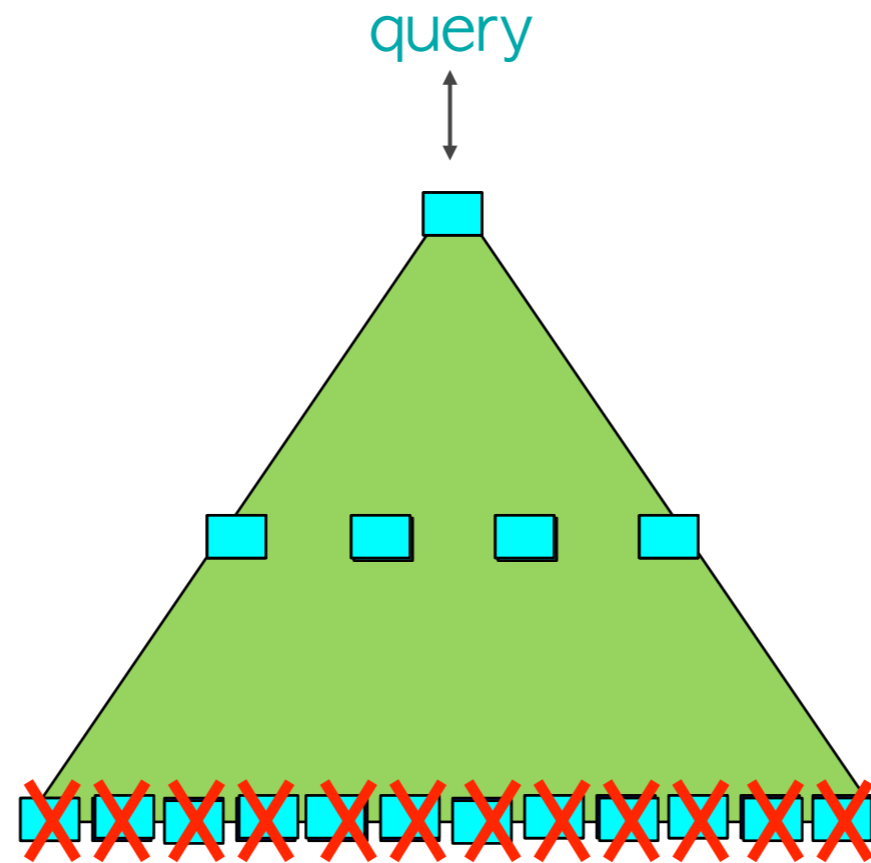
# Data Independent Failures

---



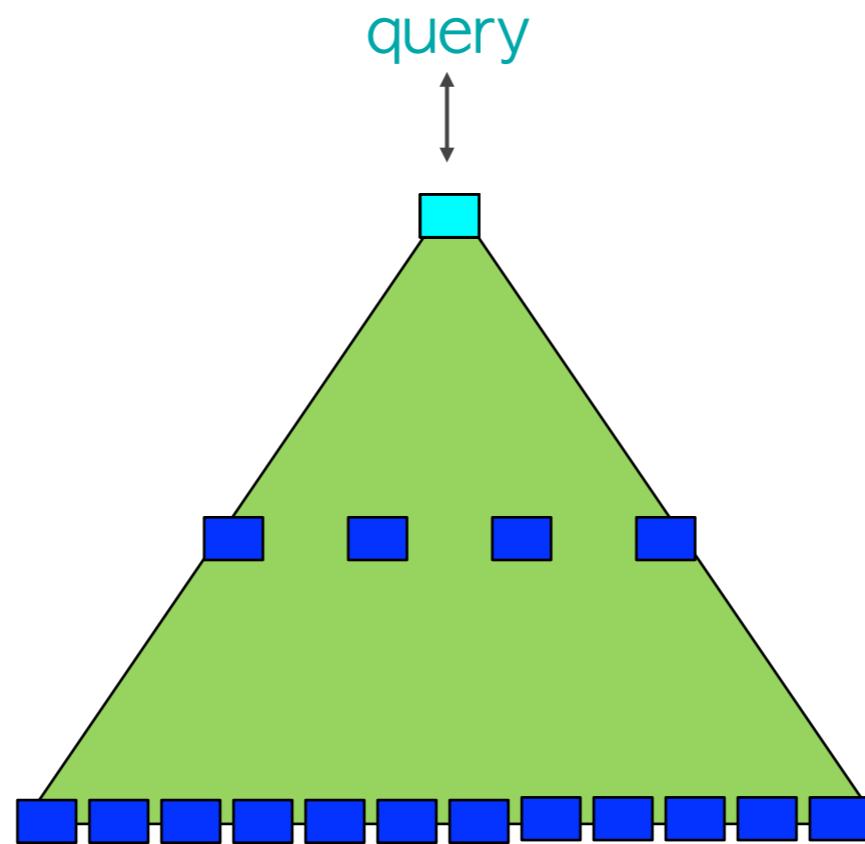
# Data Independent Failures

---



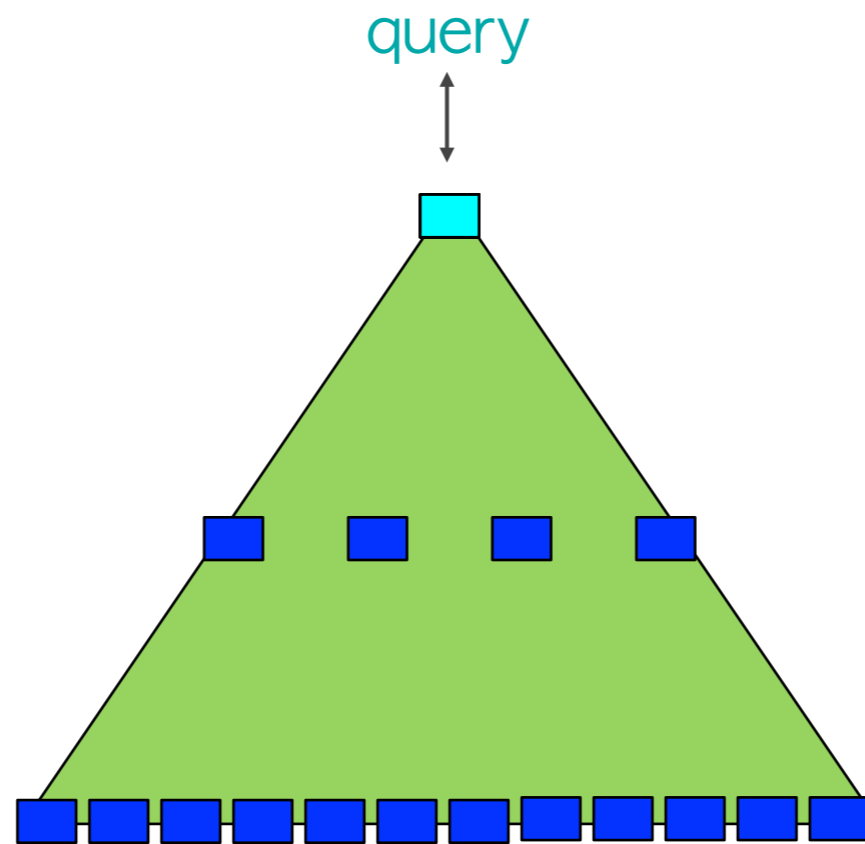
# Canary Requests (2)

---



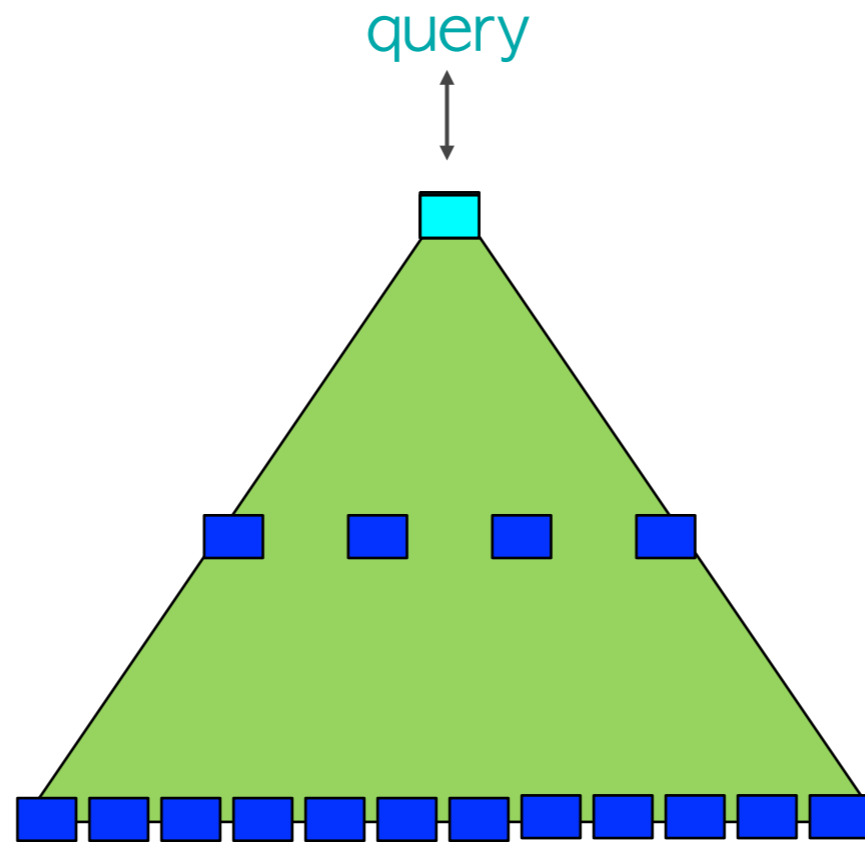
# Canary Requests (2)

---

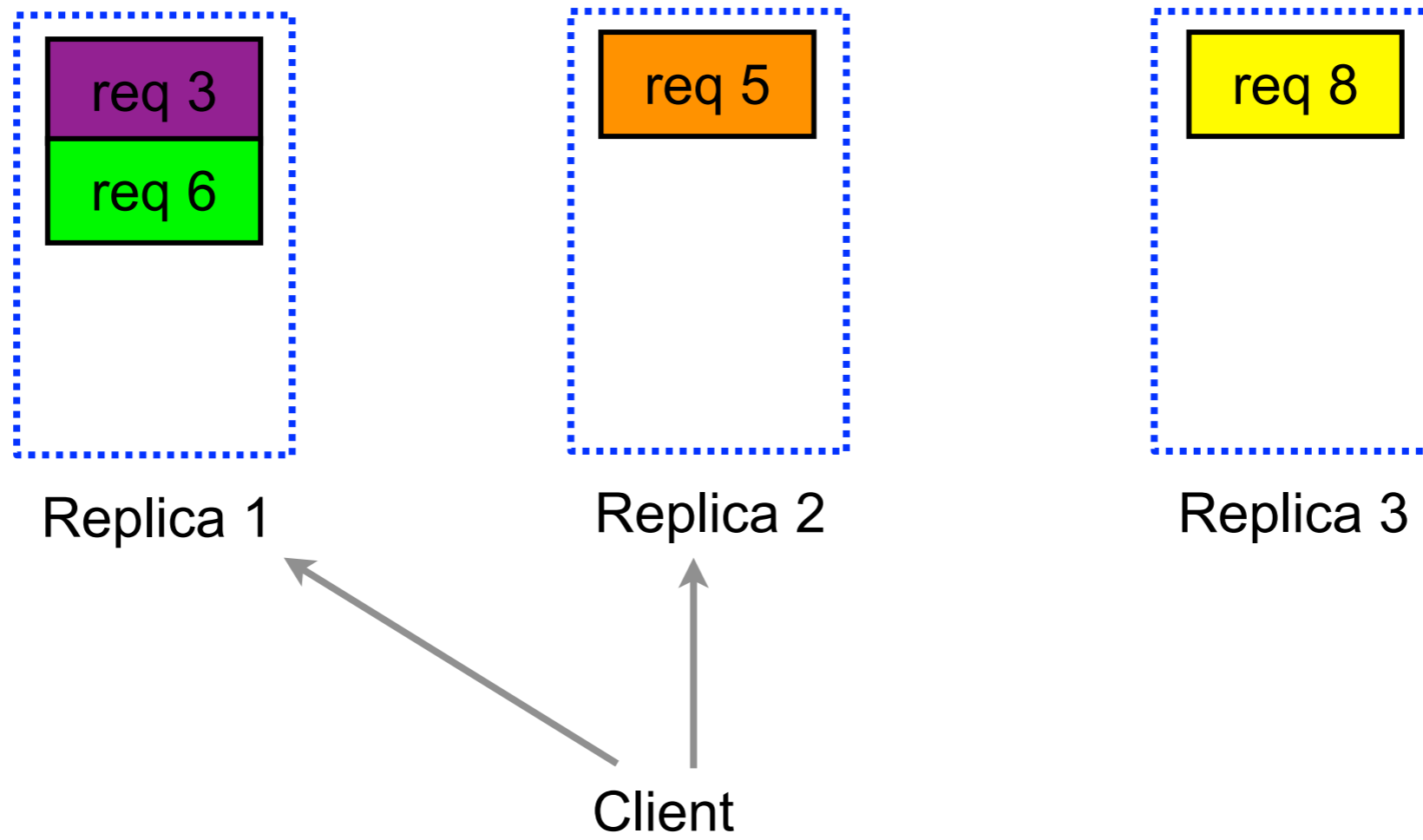


# Canary Requests (2)

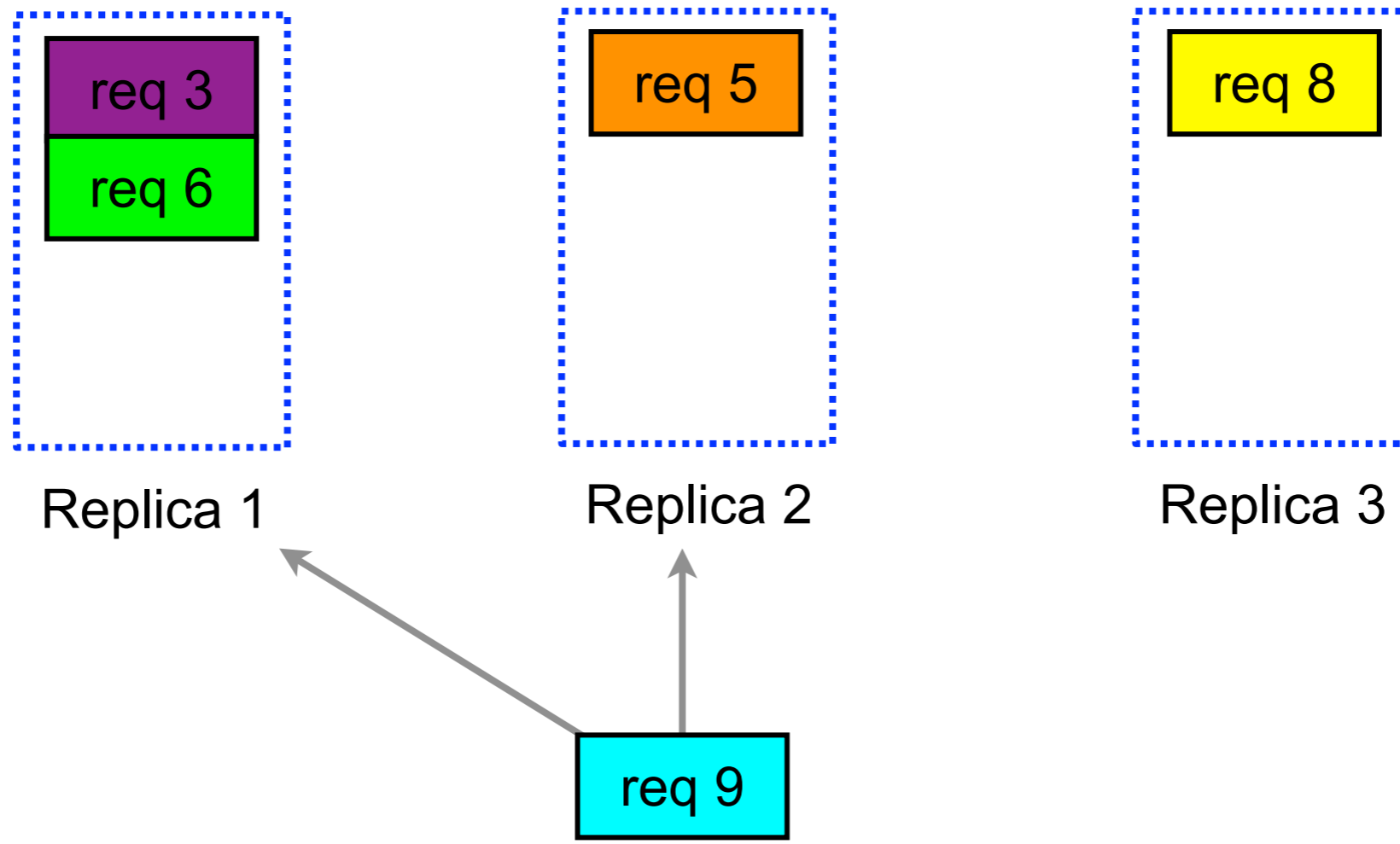
---



# Backup Requests

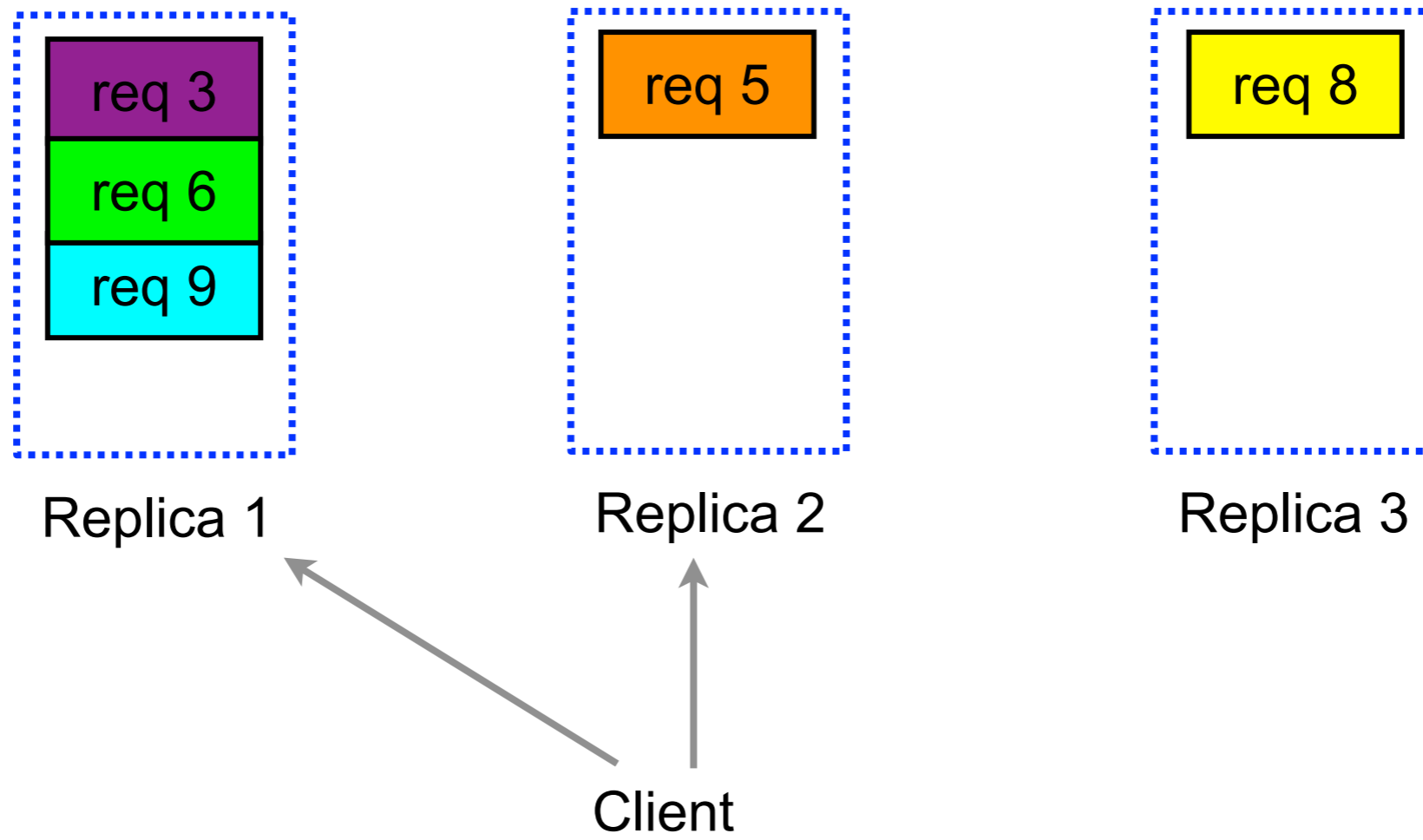


# Backup Requests

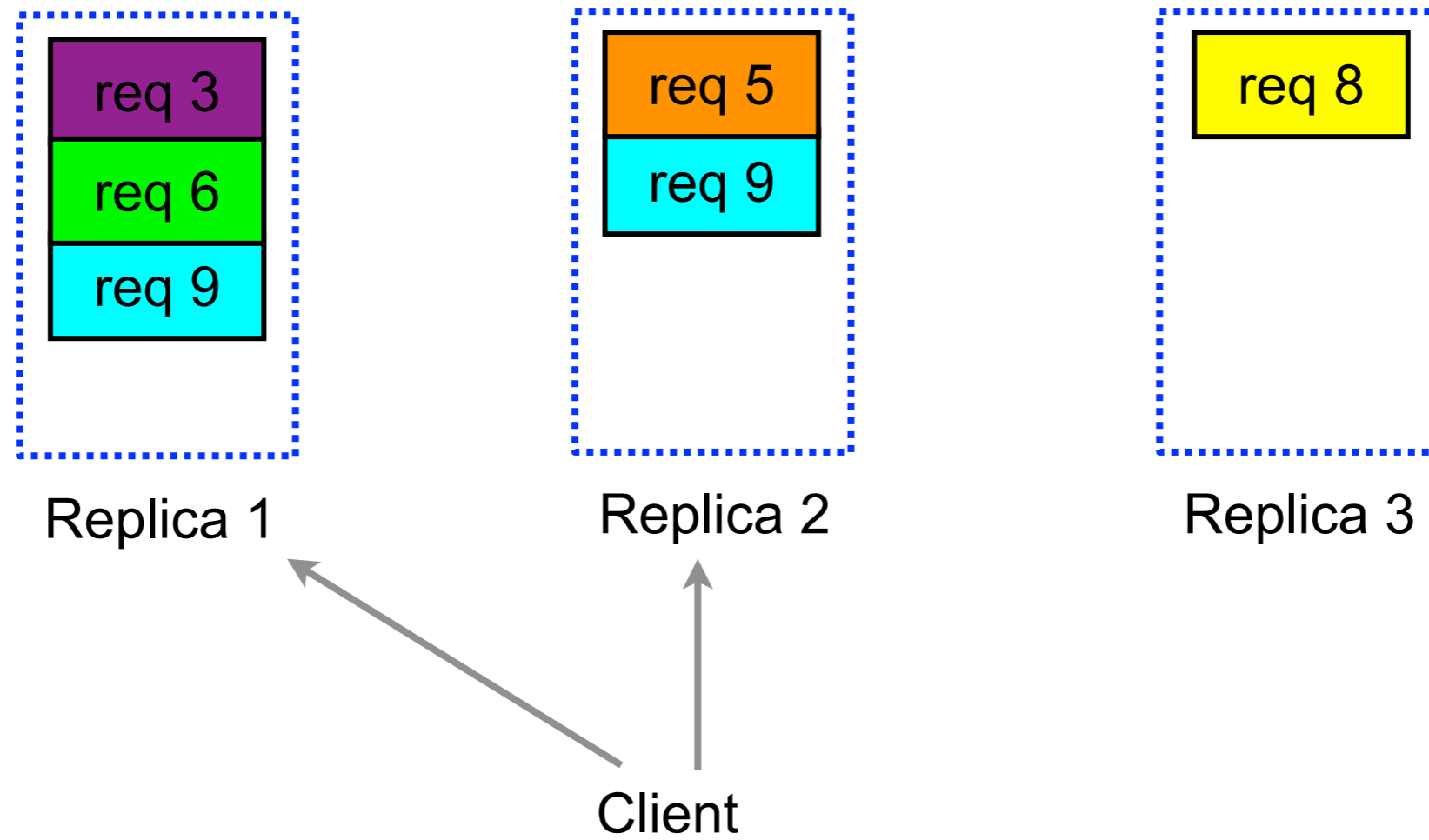




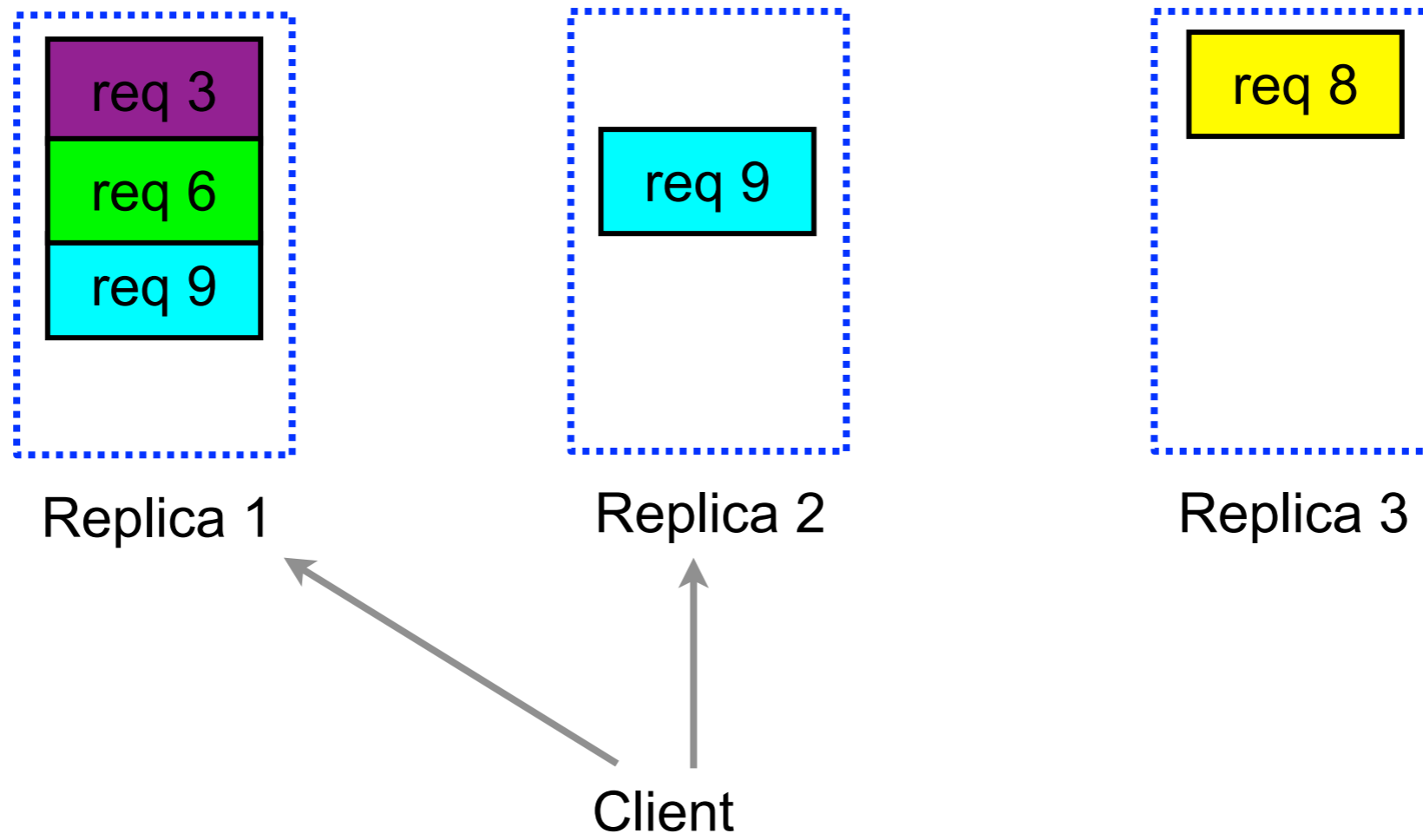
# Backup Requests



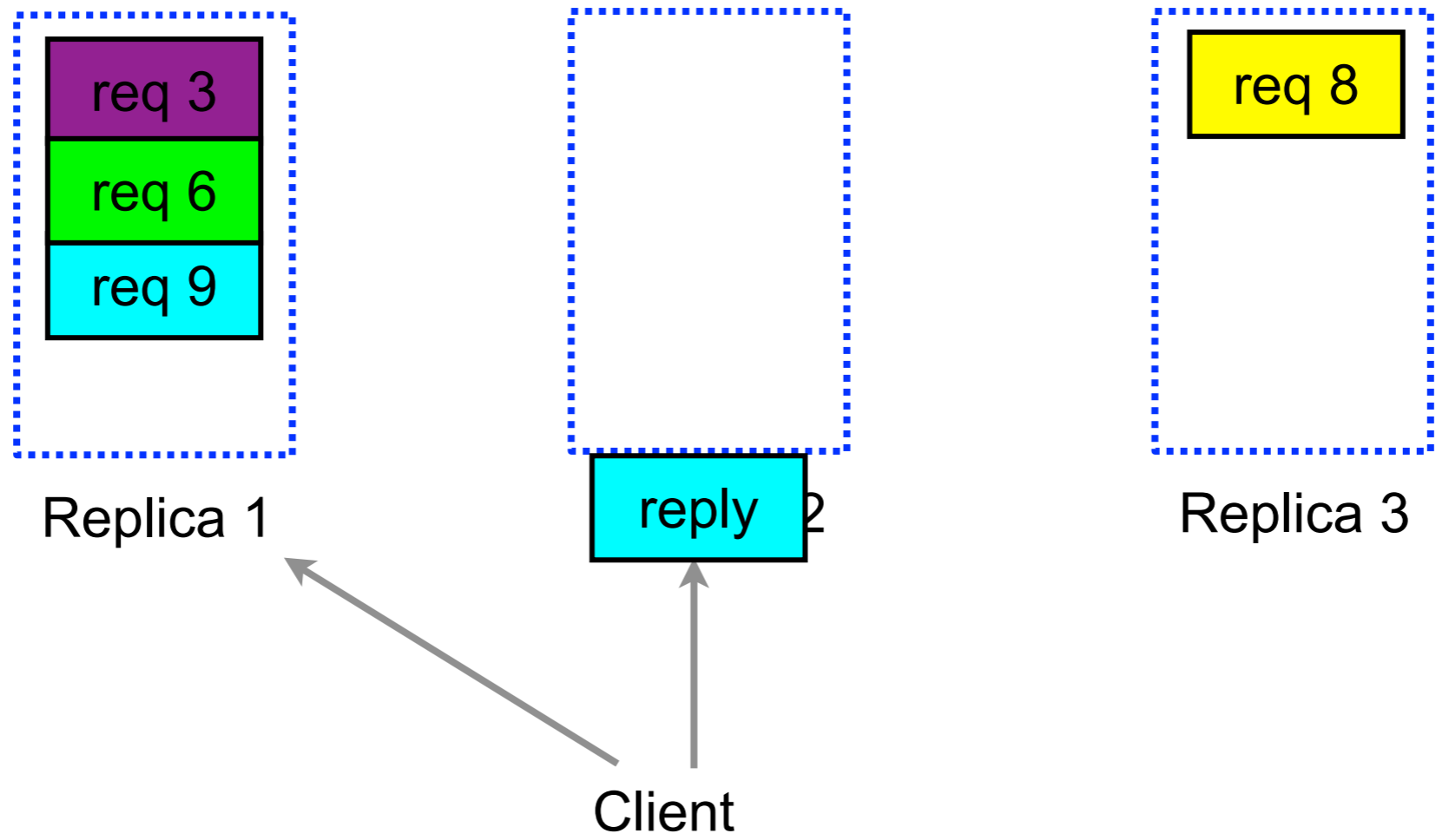
# Backup Requests



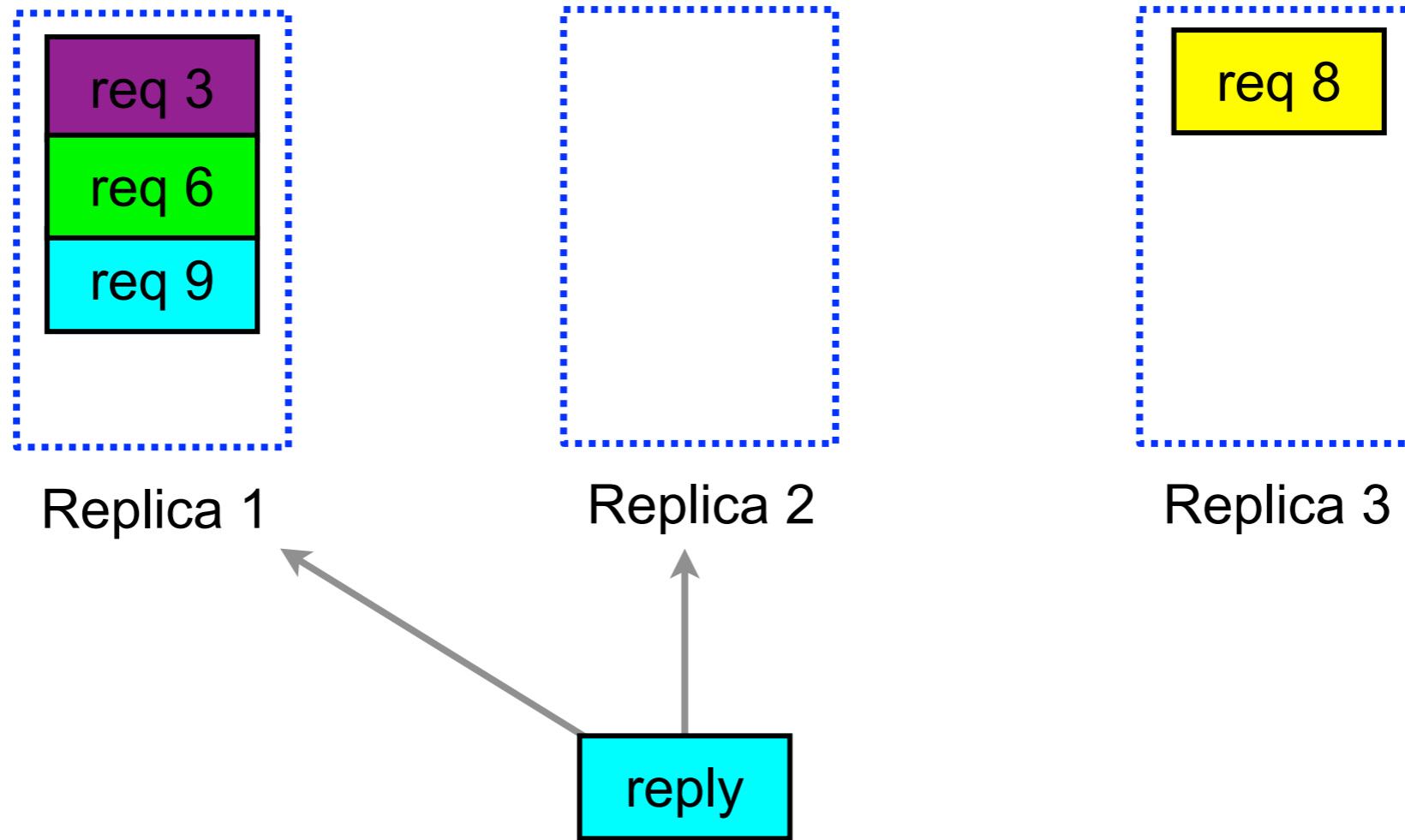
# Backup Requests



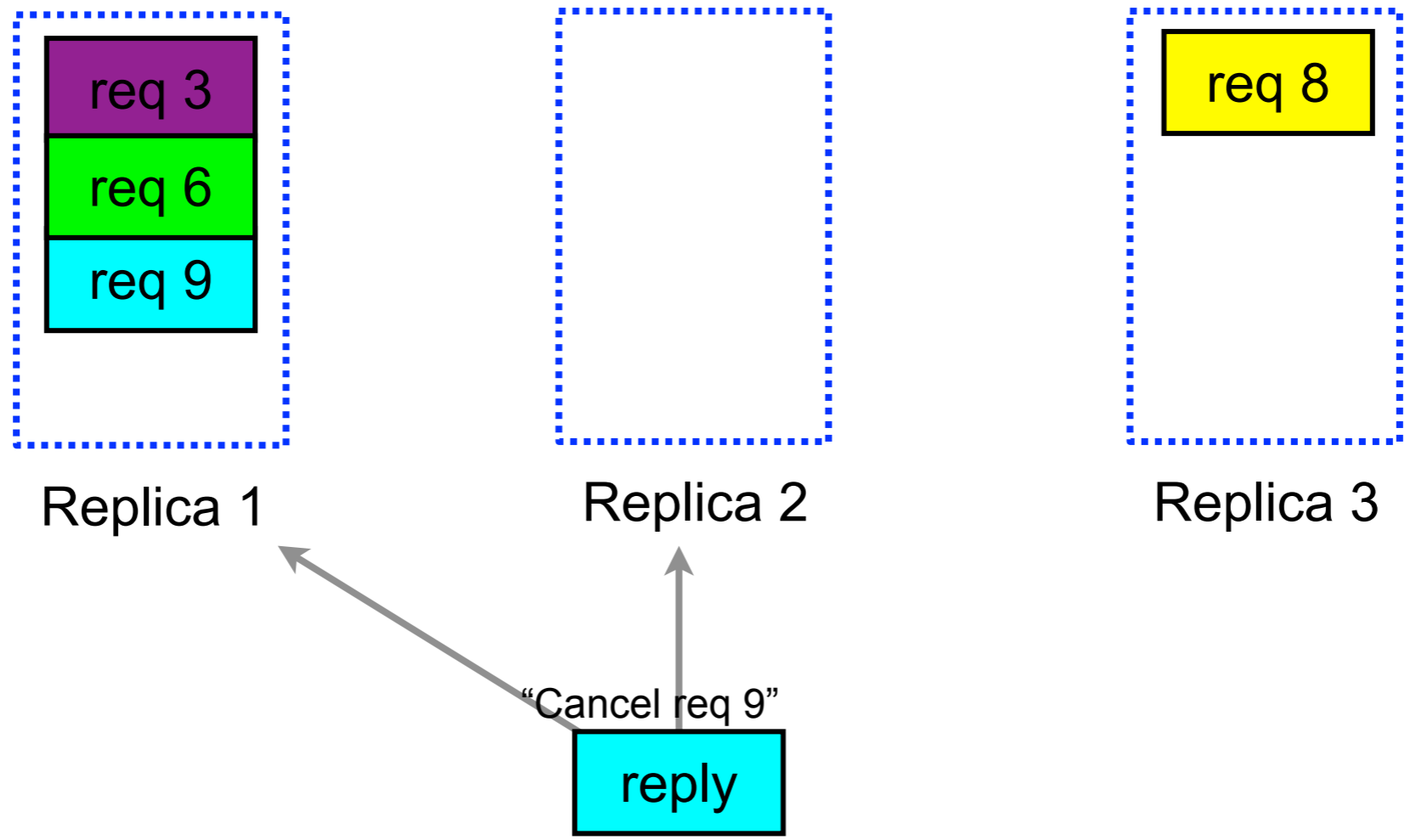
# Backup Requests



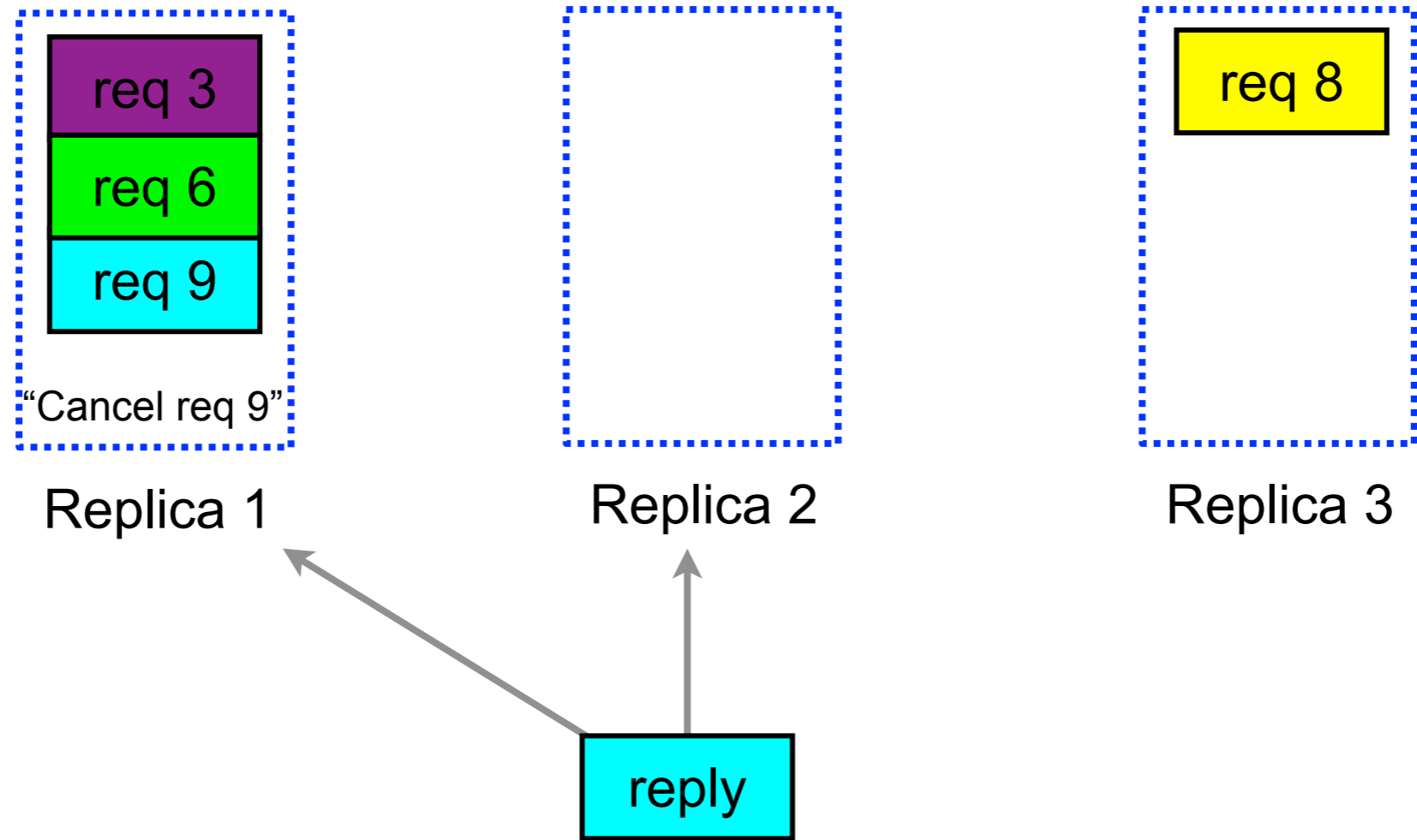
# Backup Requests



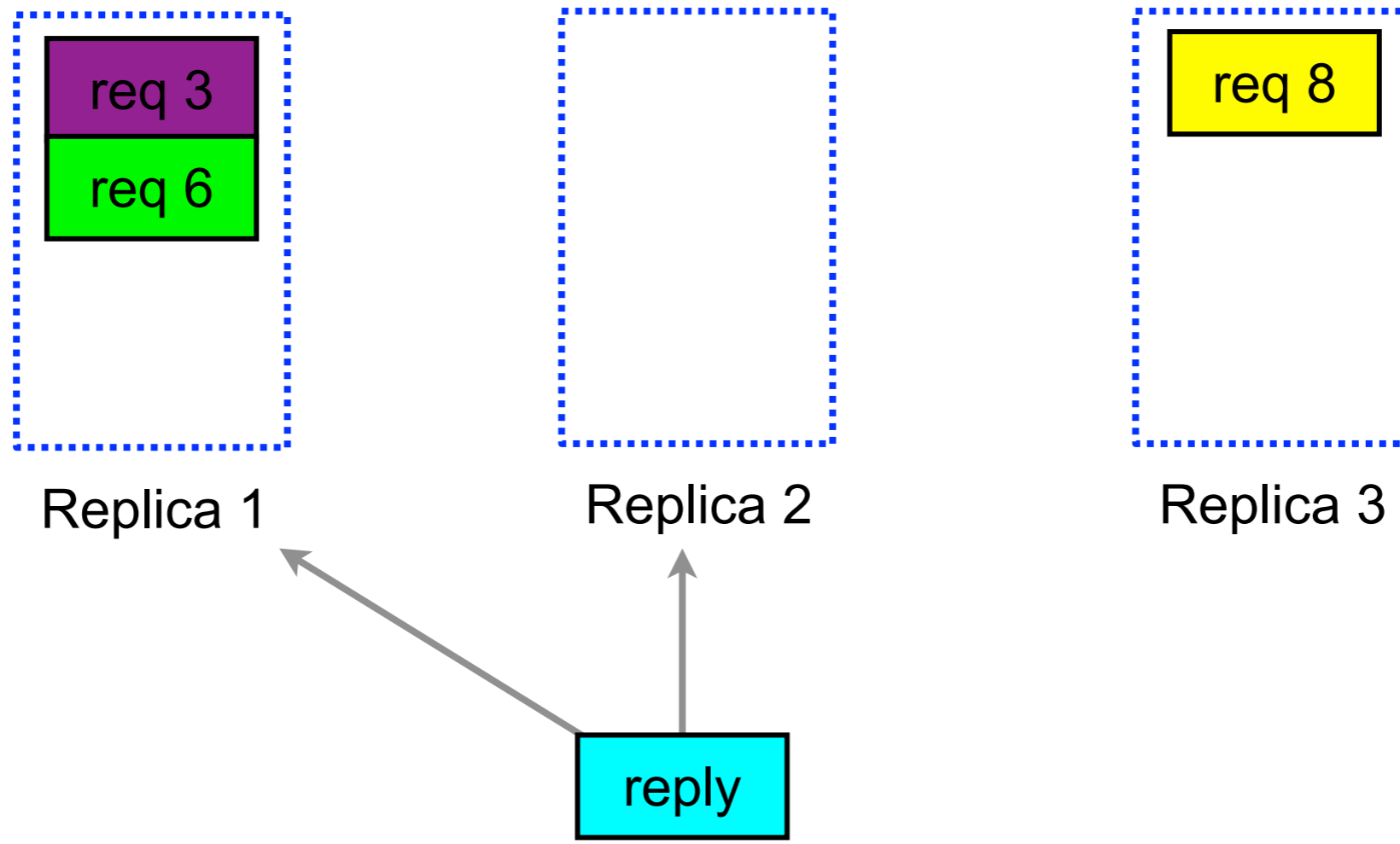
# Backup Requests



# Backup Requests



# Backup Requests





# Backup Requests Effects

---

- In-memory BigTable lookups
  - data replicated in two in-memory tables
  - issue requests for 1000 keys spread across 100 tablets
  - measure elapsed time until data for last key arrives



# Backup Requests Effects

---

- In-memory BigTable lookups
  - data replicated in two in-memory tables
  - issue requests for 1000 keys spread across 100 tablets
  - measure elapsed time until data for last key arrives

	Avg	Std Dev	95%ile	99%ile	99.9%ile
No backups	33 ms	1524 ms	24 ms	52 ms	994 ms
Backup after 10 ms	14 ms	4 ms	20 ms	23 ms	50 ms
Backup after 50 ms	16 ms	12 ms	57 ms	63 ms	68 ms



# Backup Requests Effects

---

- In-memory BigTable lookups
  - data replicated in two in-memory tables
  - issue requests for 1000 keys spread across 100 tablets
  - measure elapsed time until data for last key arrives

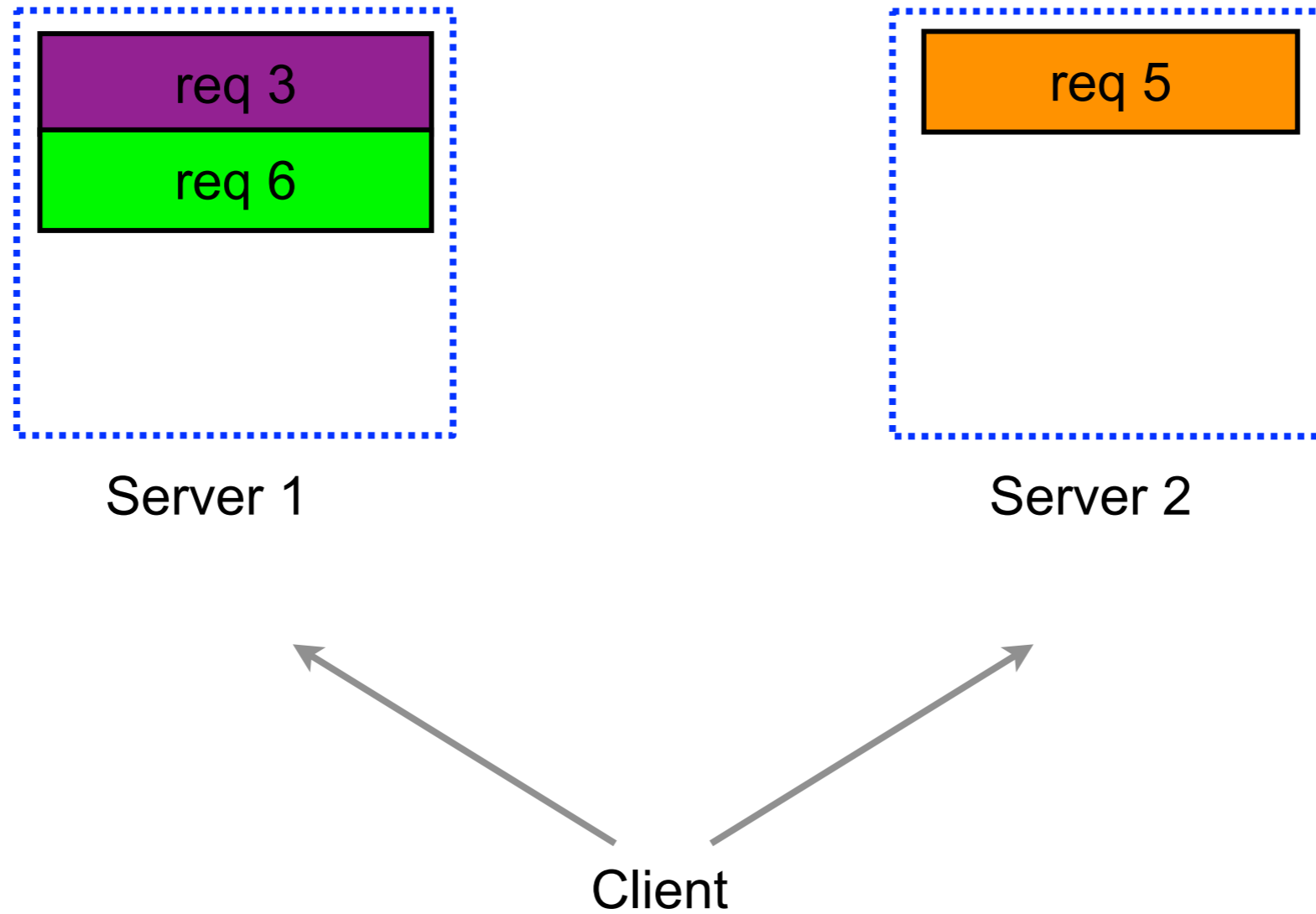
	Avg	Std Dev	95%ile	99%ile	99.9%ile
No backups	33 ms	1524 ms	24 ms	52 ms	994 ms
Backup after 10 ms	14 ms	4 ms	20 ms	23 ms	50 ms
Backup after 50 ms	16 ms	12 ms	57 ms	63 ms	68 ms

- Modest increase in request load:
  - 10 ms delay: <5% extra requests; 50 ms delay: <1%



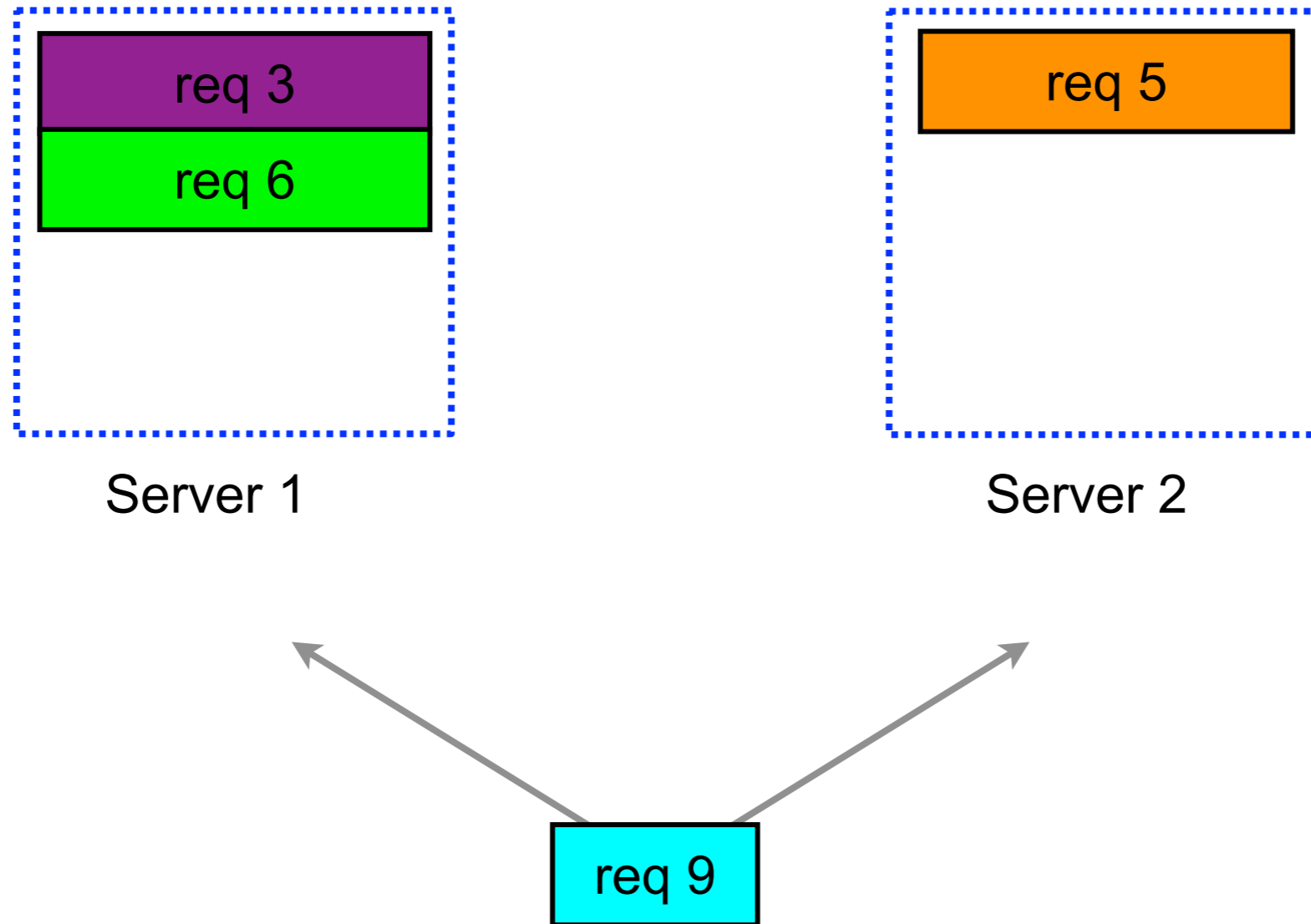
# Backup Requests w/ Cross-Server Cancellation

---



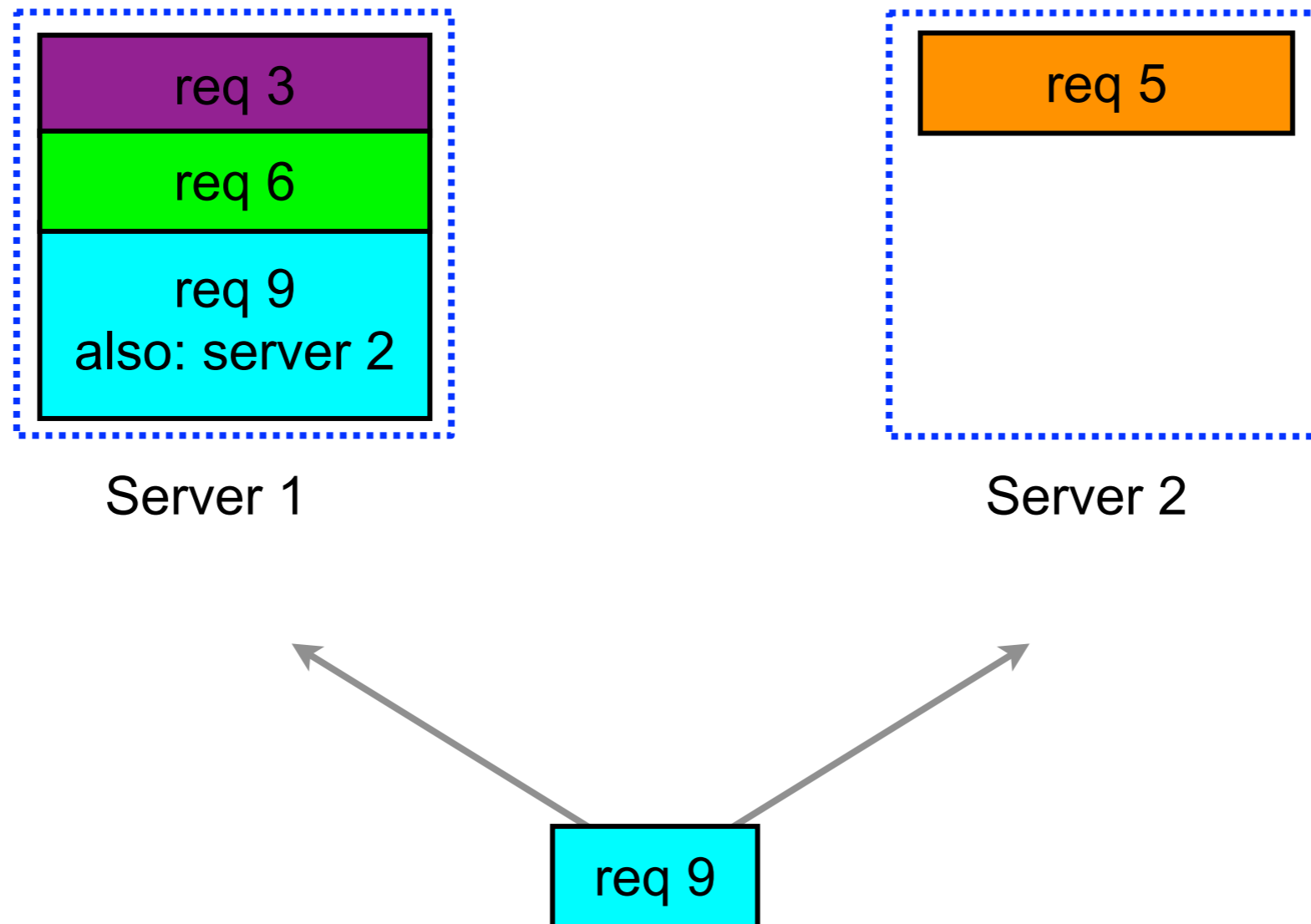
# Backup Requests w/ Cross-Server Cancellation

---



# Backup Requests w/ Cross-Server Cancellation

---

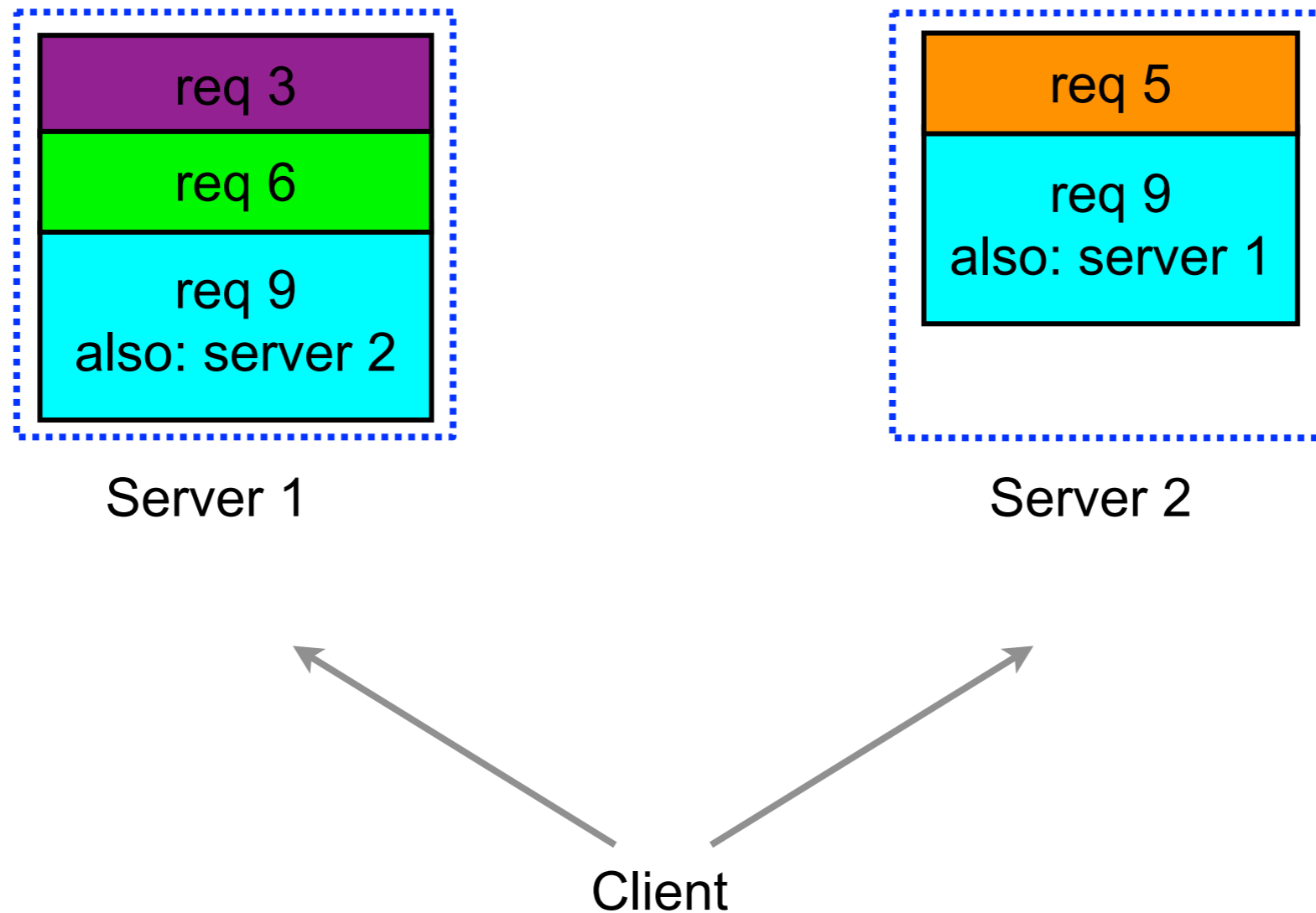


Each request identifies other server(s) to which request might be sent



# Backup Requests w/ Cross-Server Cancellation

---

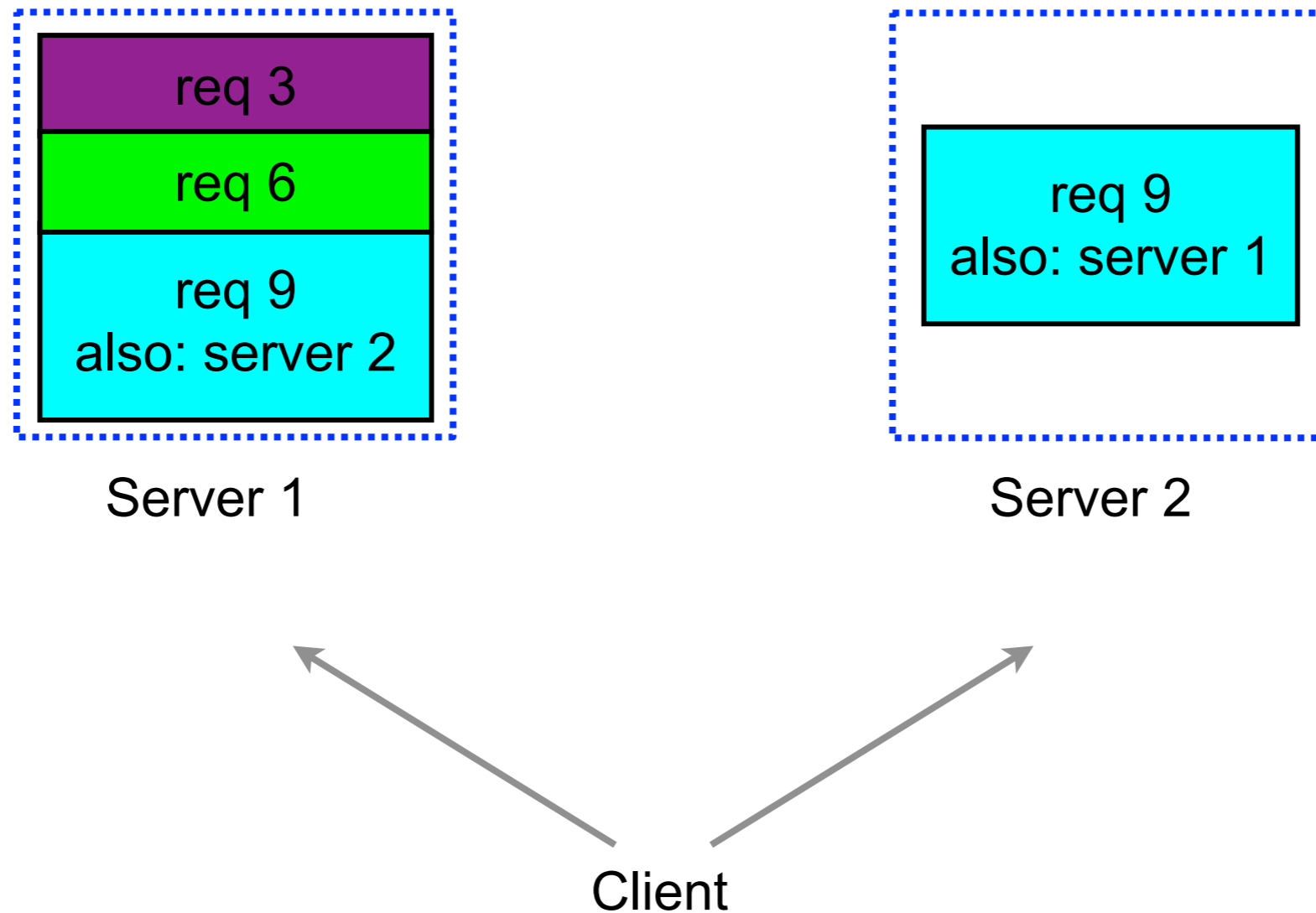


Each request identifies other server(s) to which request might be sent



# Backup Requests w/ Cross-Server Cancellation

---



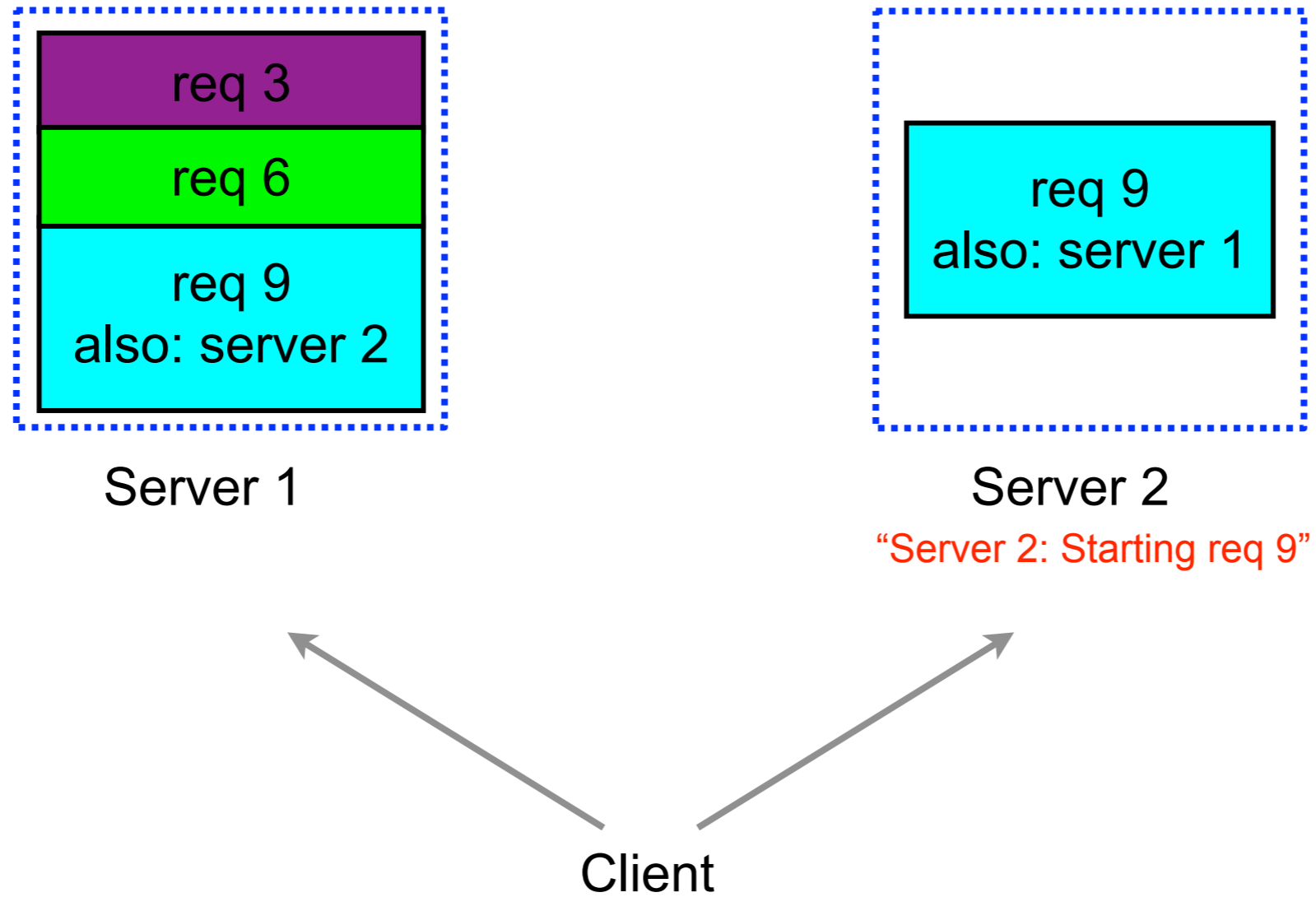
Each request identifies other server(s) to which request might be sent





# Backup Requests w/ Cross-Server Cancellation

---

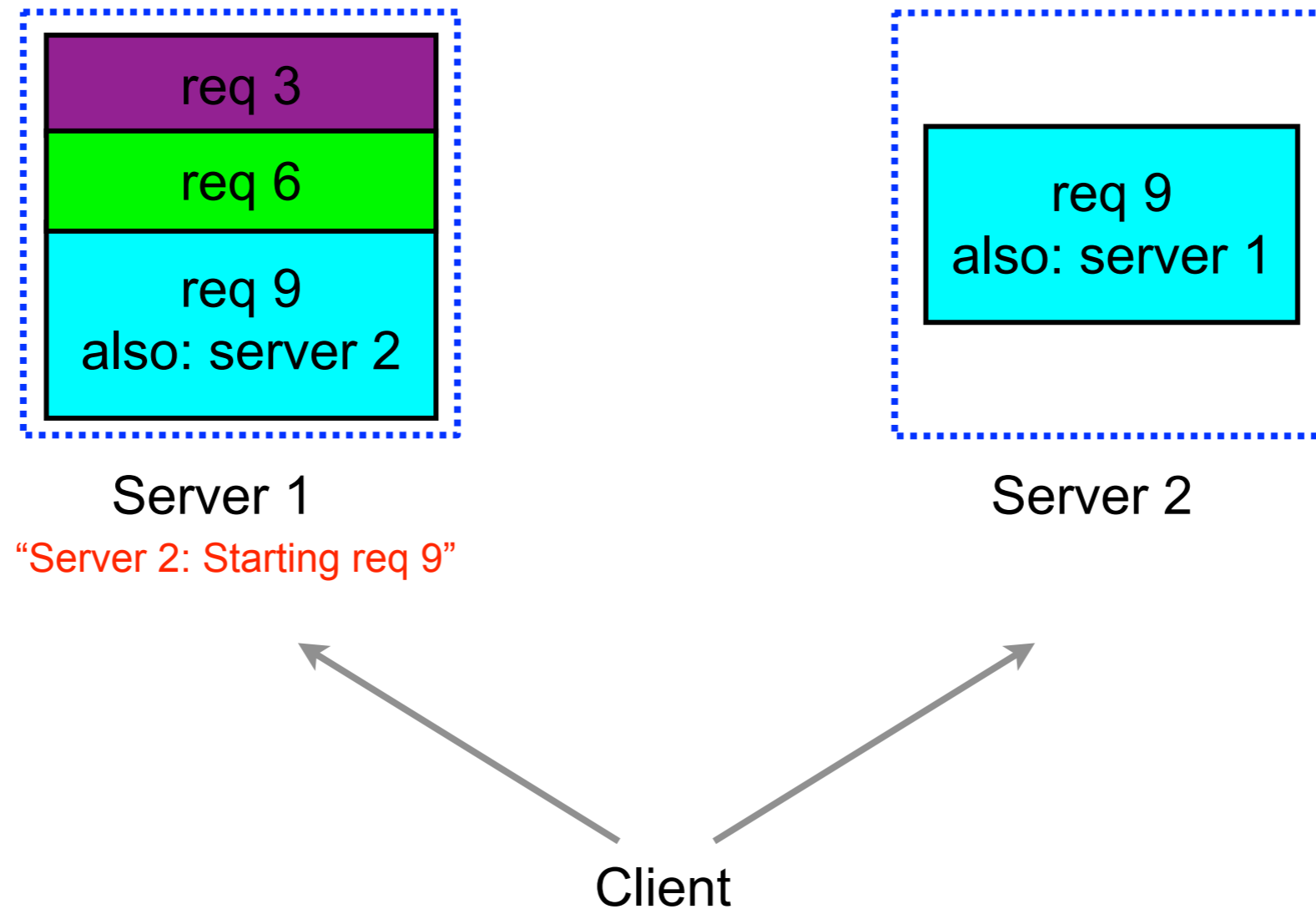


Each request identifies other server(s) to which request might be sent



# Backup Requests w/ Cross-Server Cancellation

---

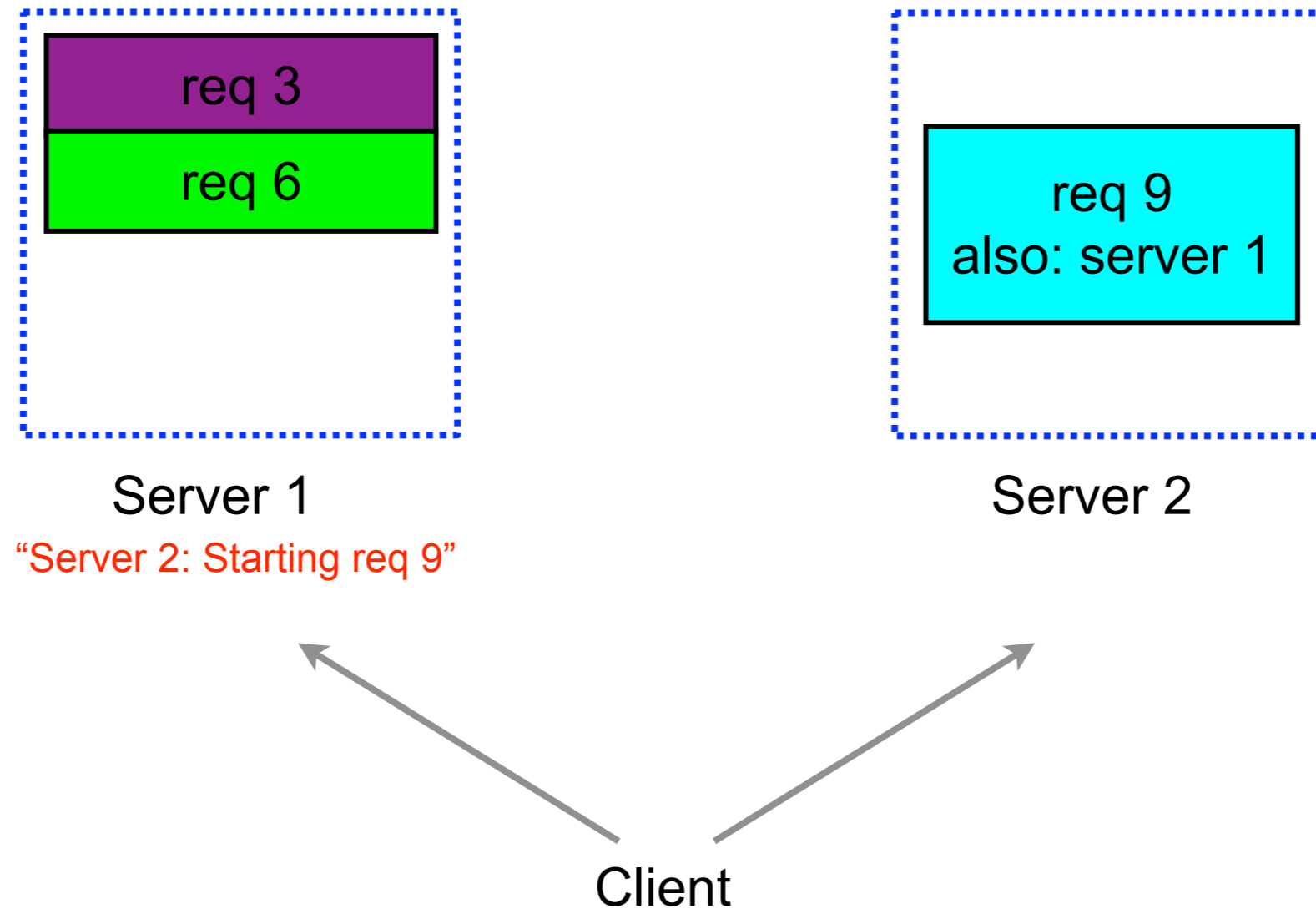


Each request identifies other server(s) to which request might be sent



# Backup Requests w/ Cross-Server Cancellation

---

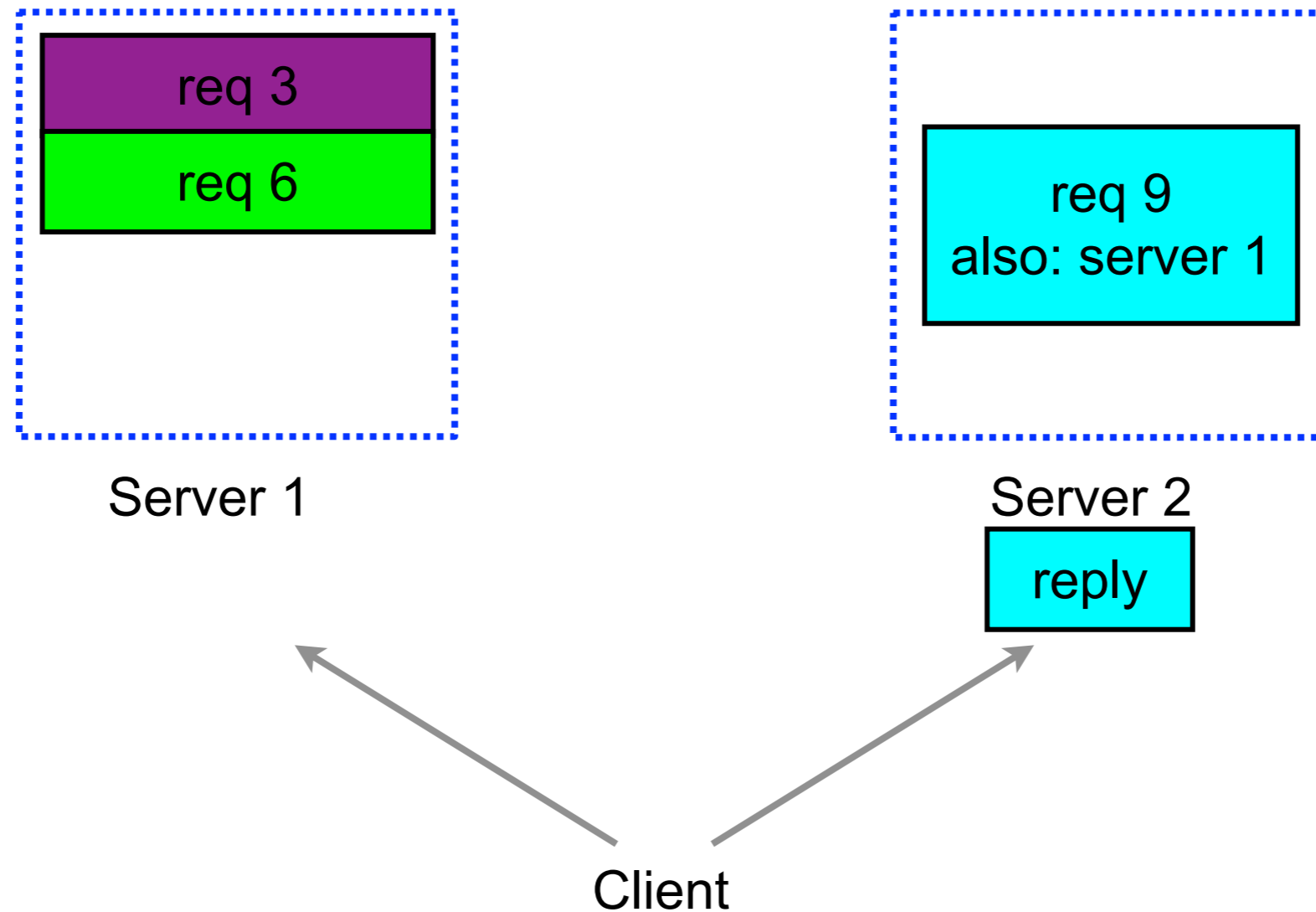


Each request identifies other server(s) to which request might be sent



# Backup Requests w/ Cross-Server Cancellation

---

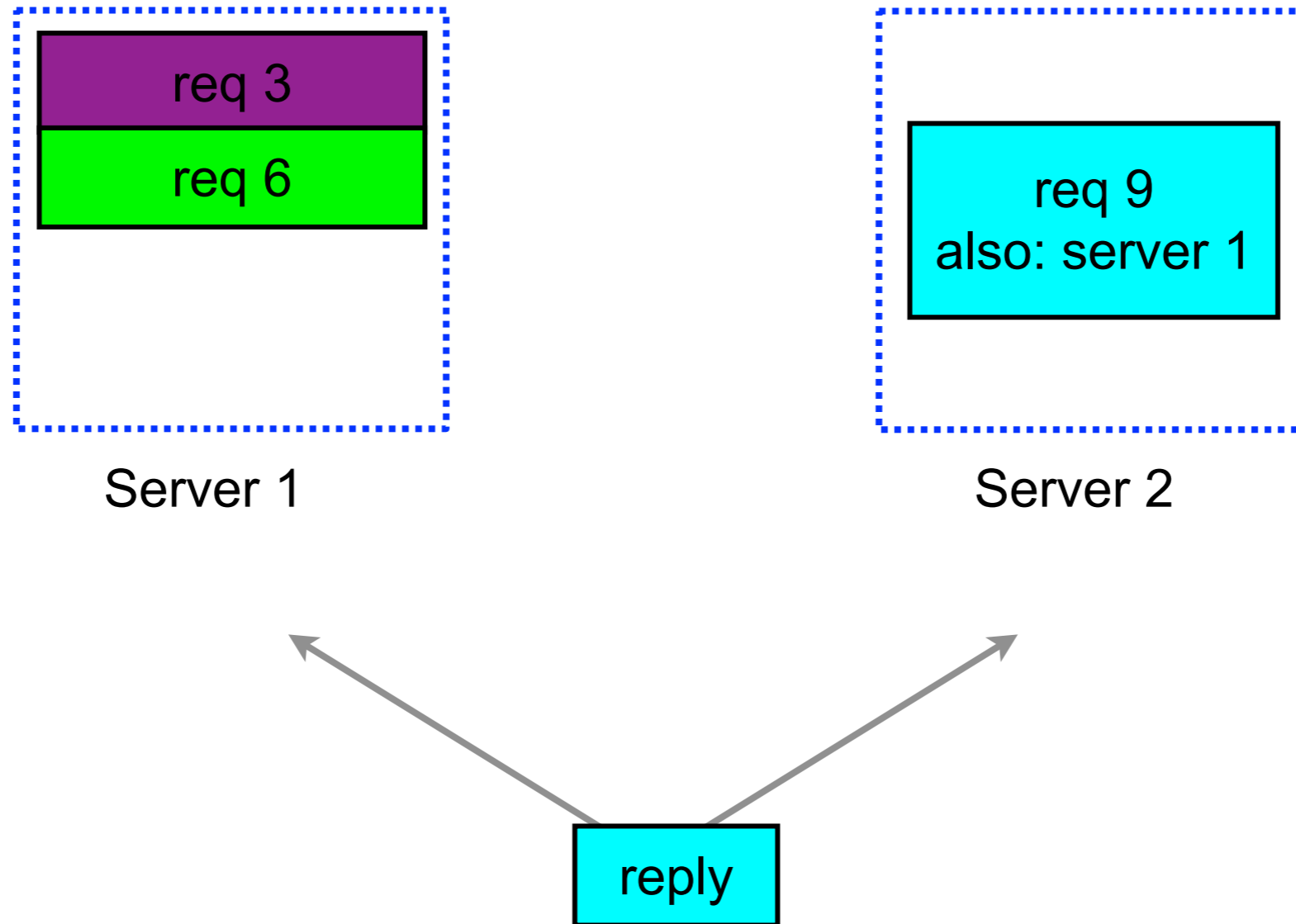


Each request identifies other server(s) to which request might be sent



# Backup Requests w/ Cross-Server Cancellation

---

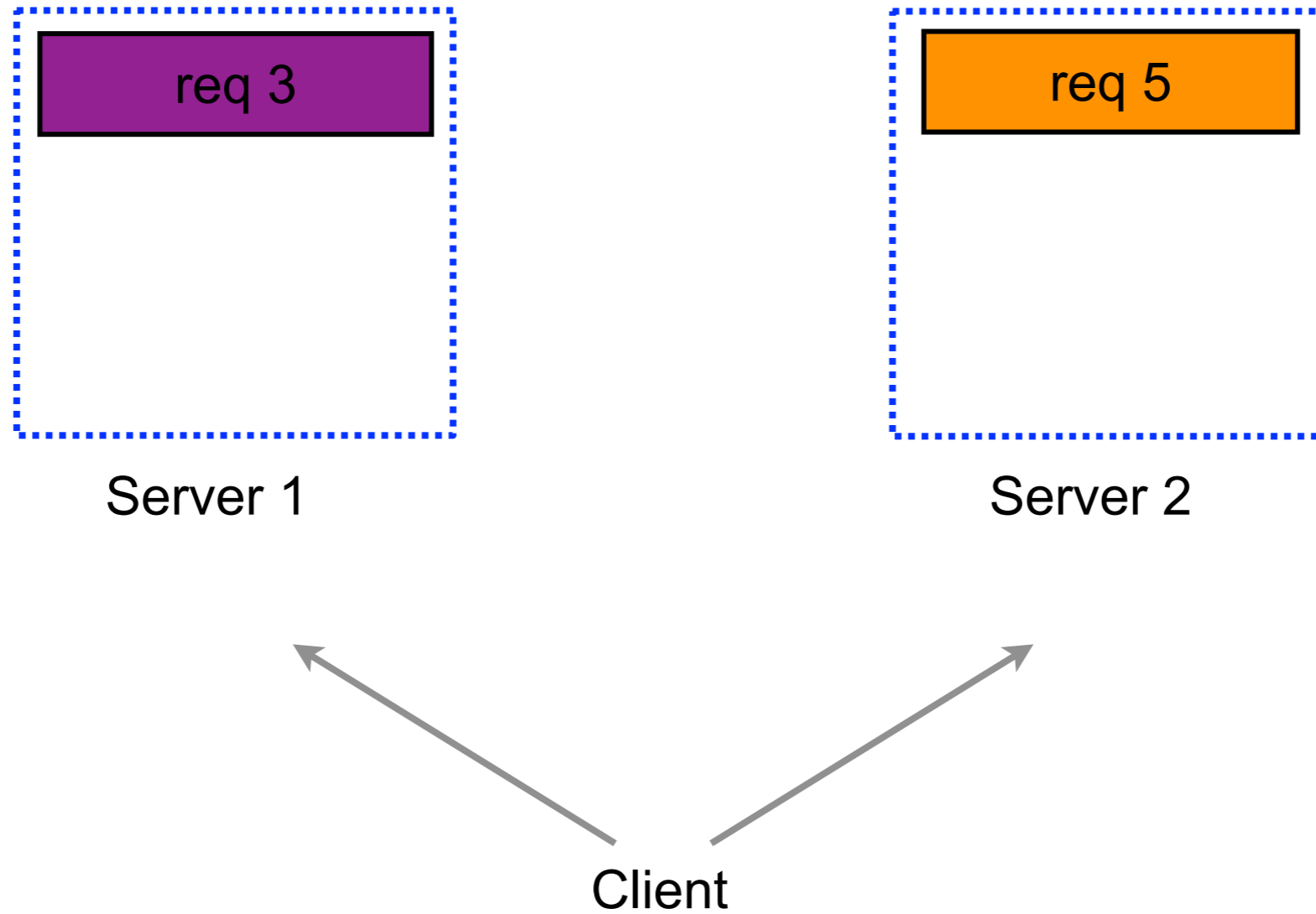


Each request identifies other server(s) to which request might be sent



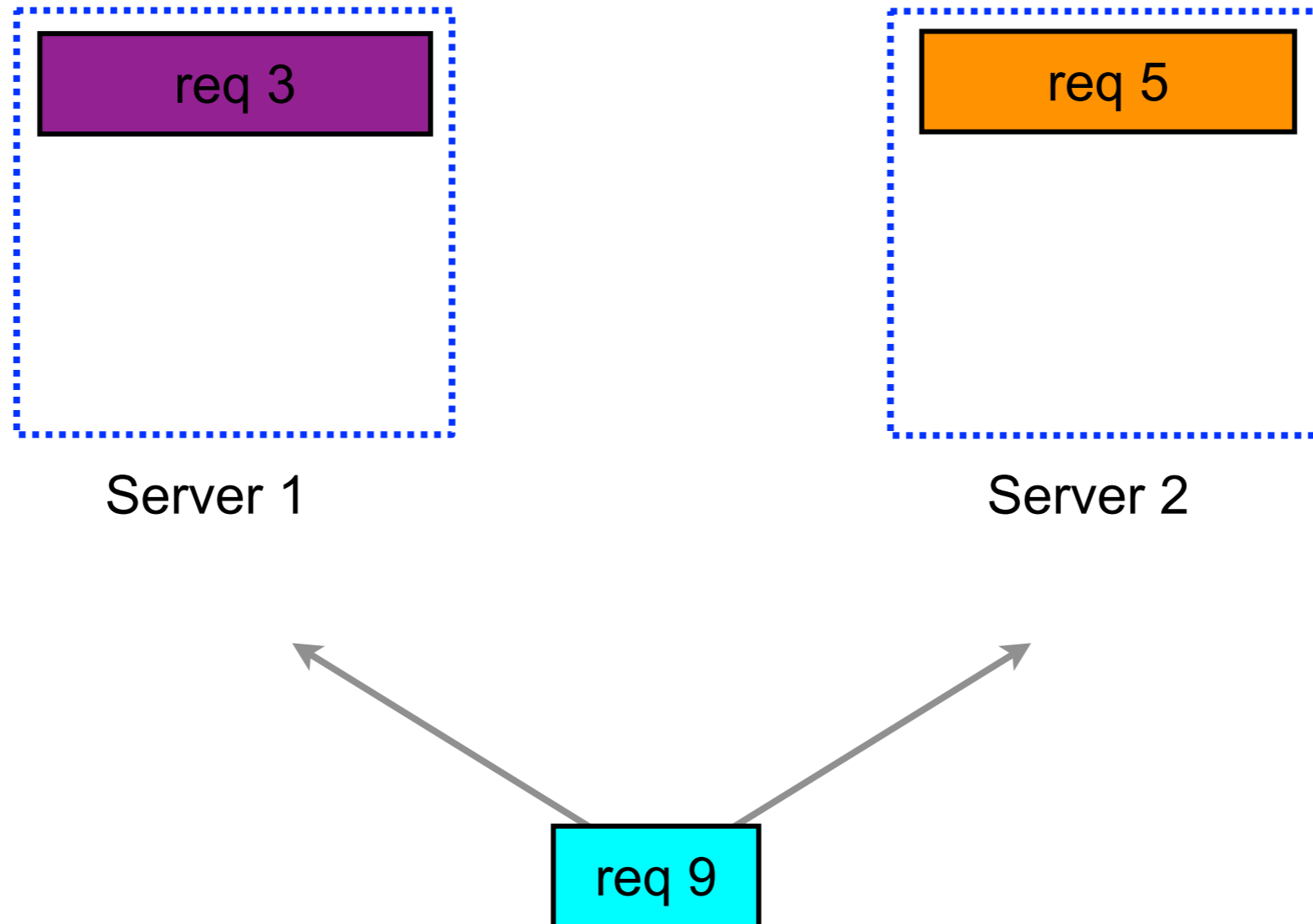
# Backup Requests: Bad Case

---



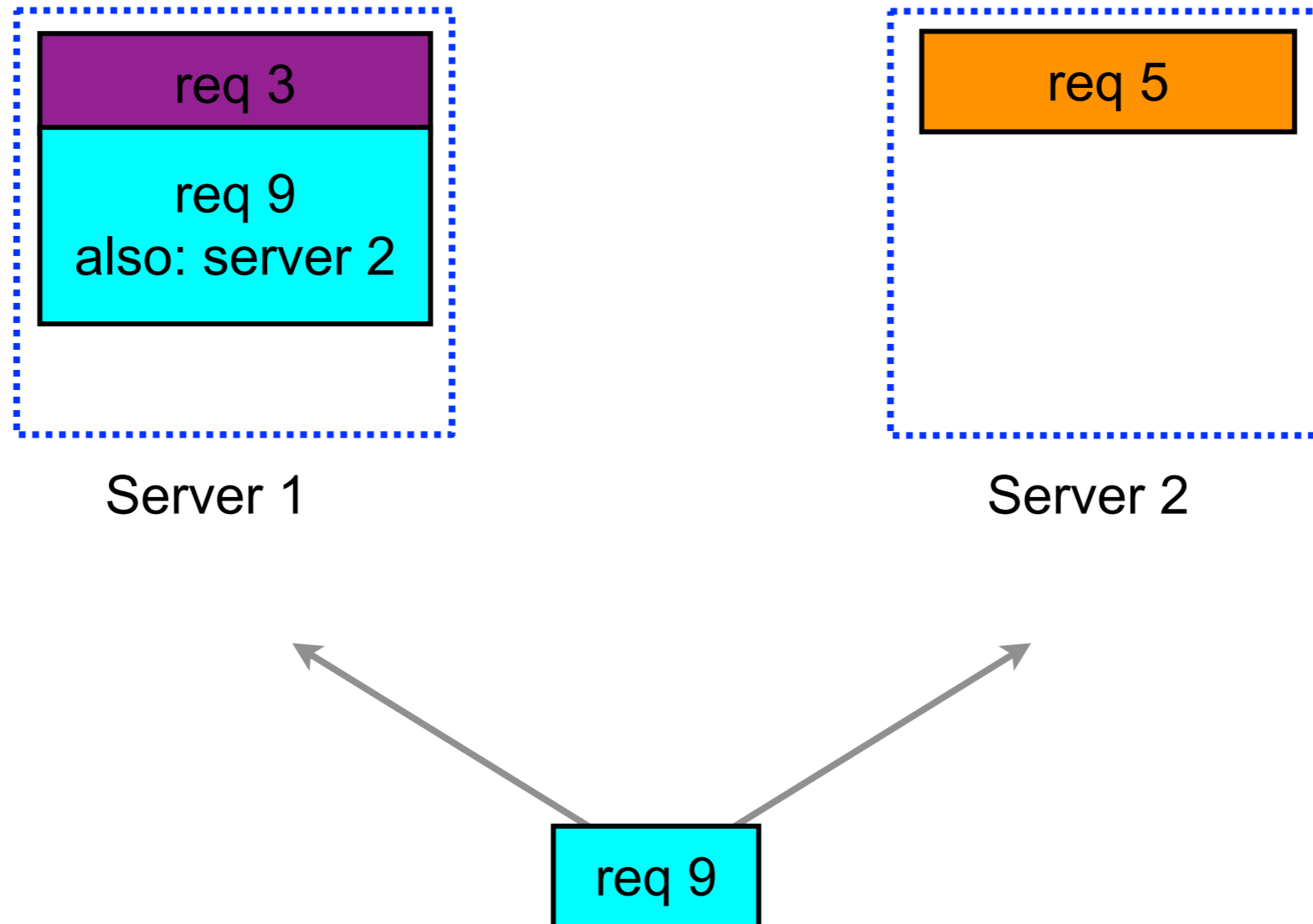
# Backup Requests: Bad Case

---



# Backup Requests: Bad Case

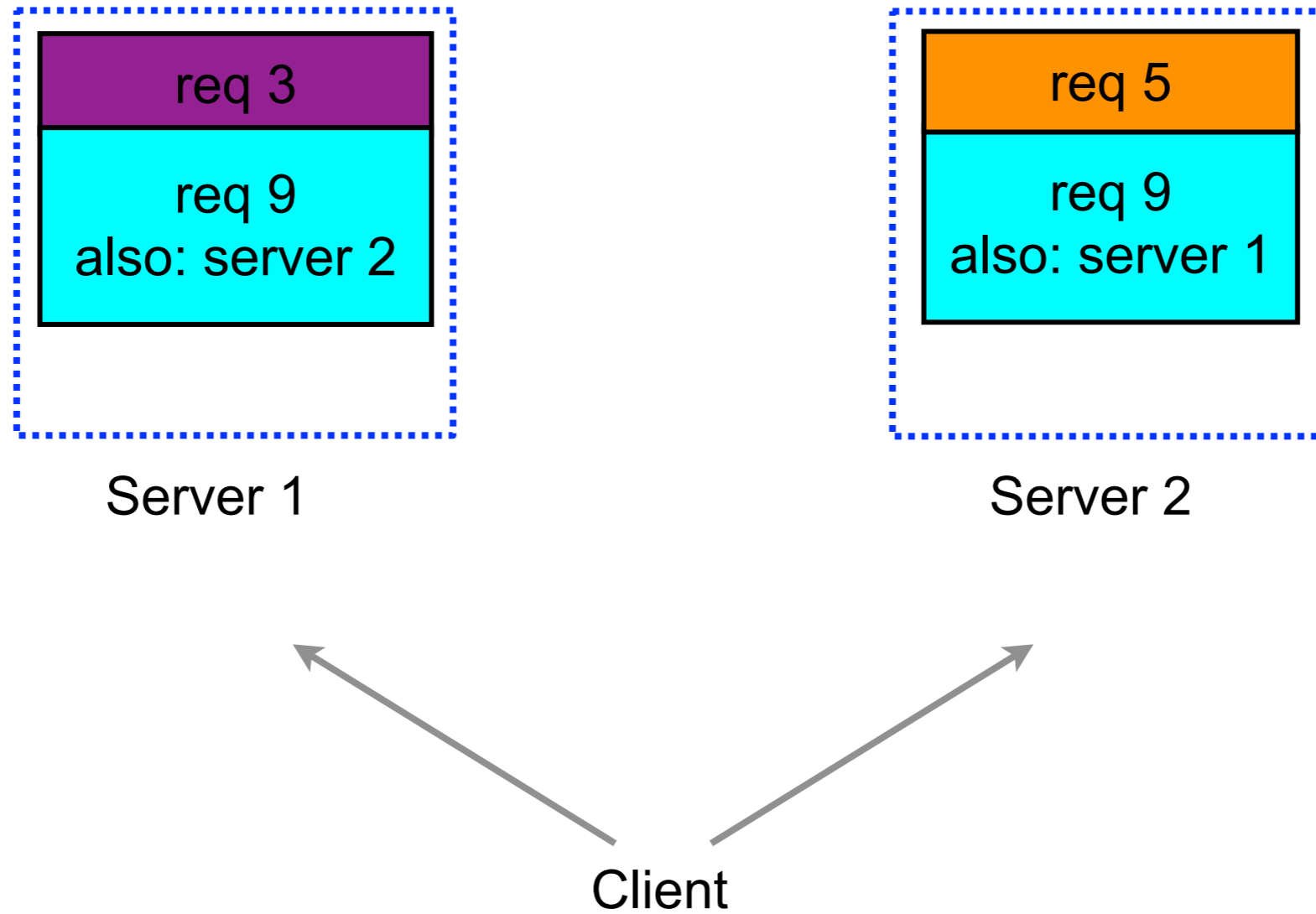
---





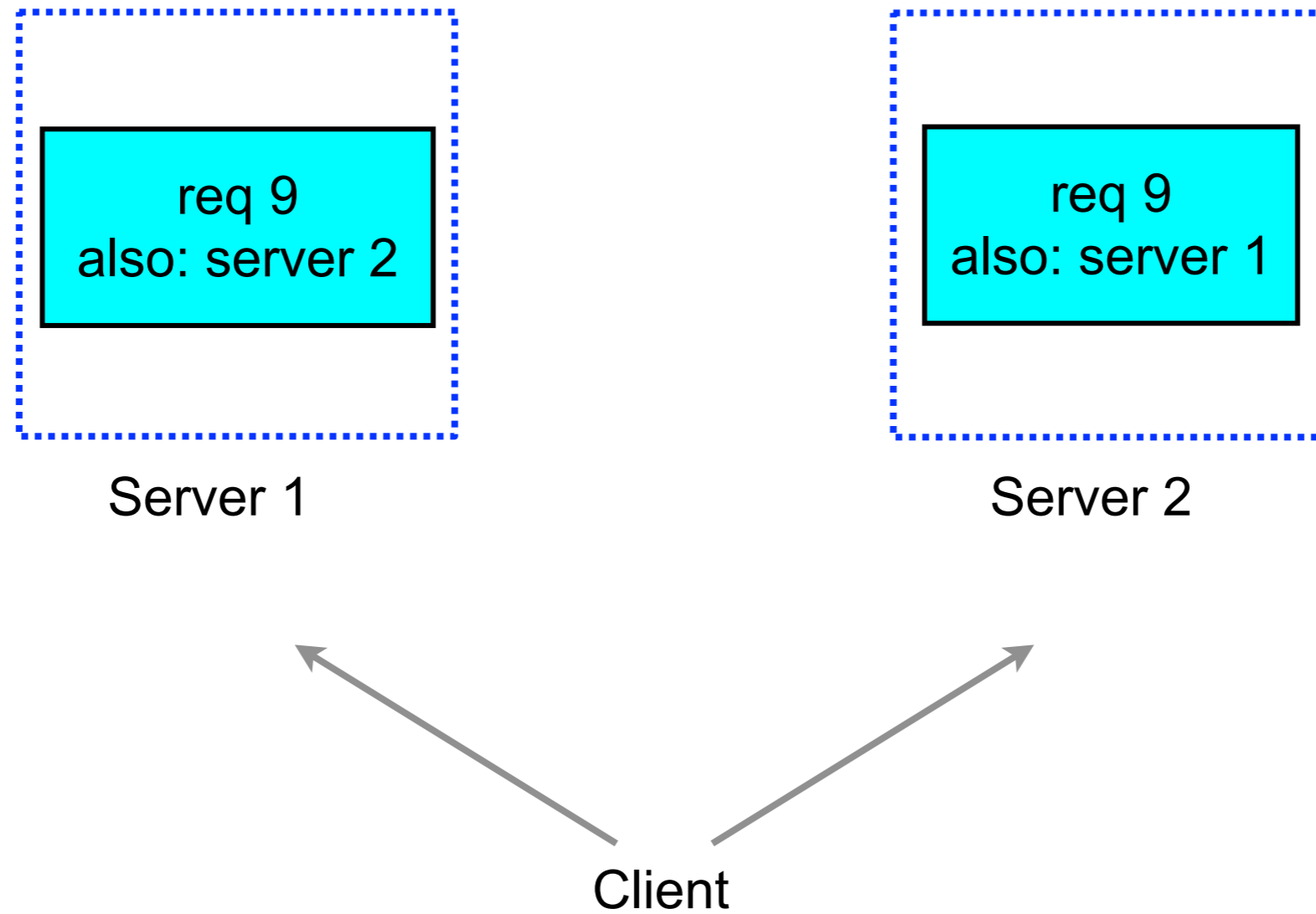
# Backup Requests: Bad Case

---



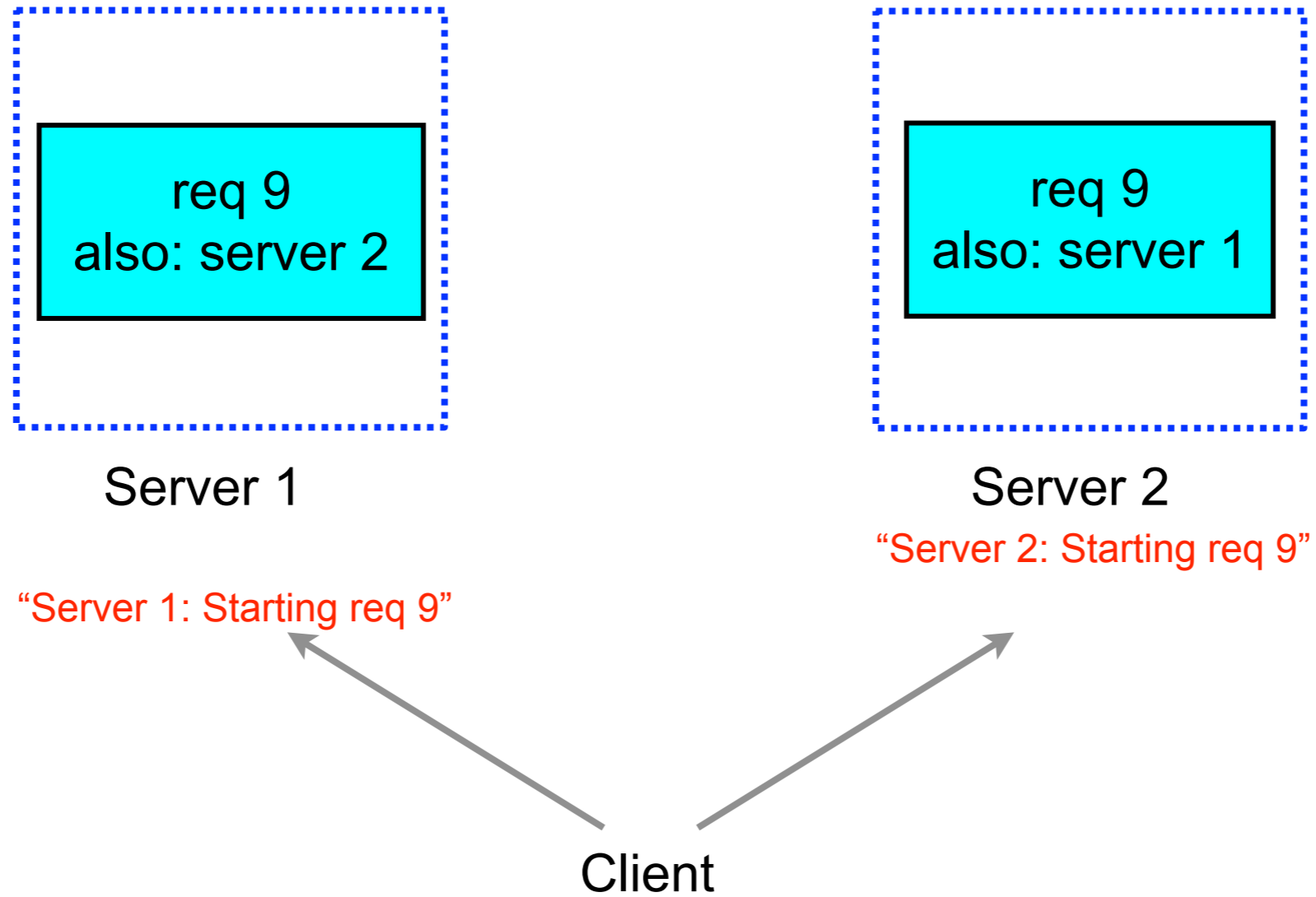
# Backup Requests: Bad Case

---



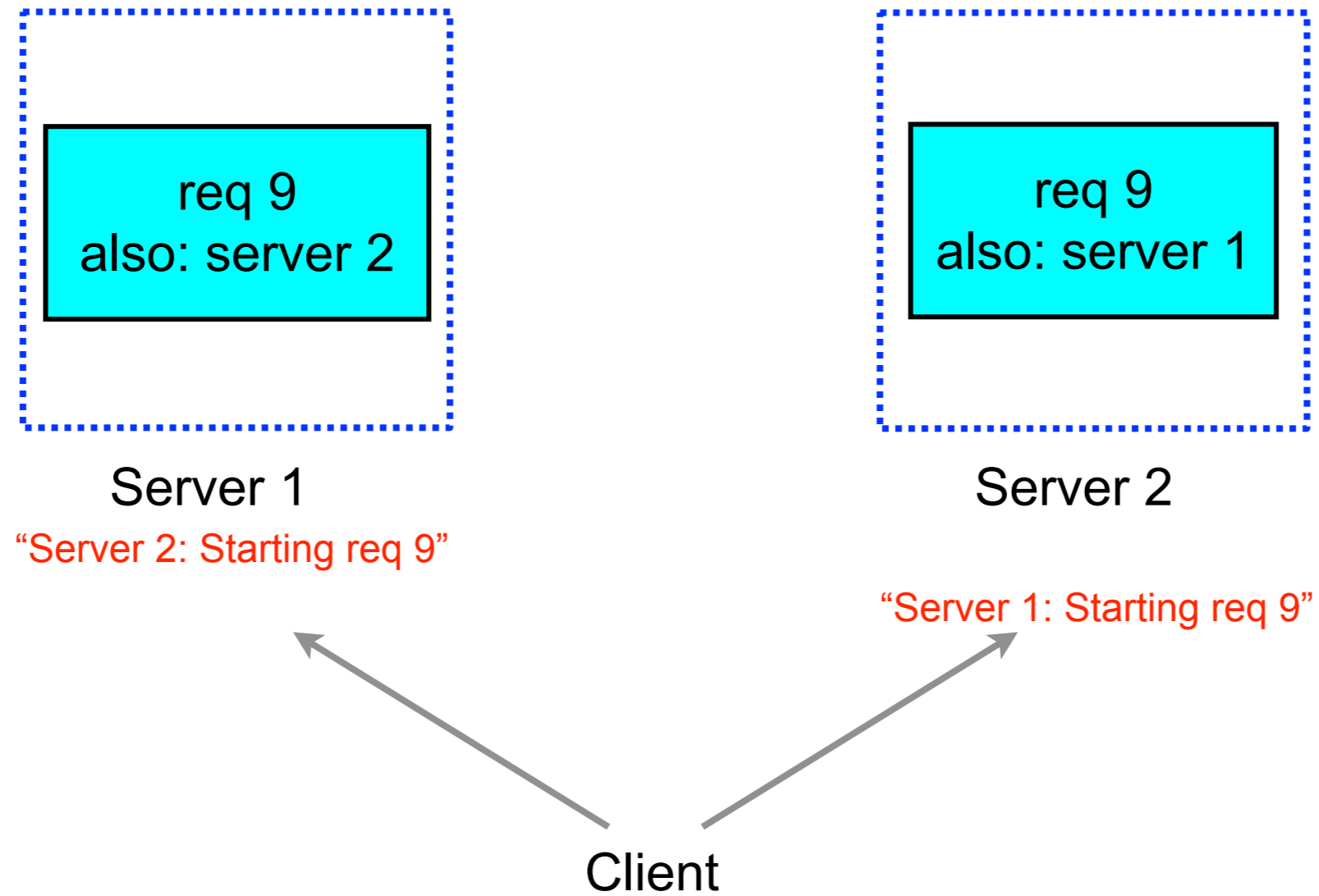
# Backup Requests: Bad Case

---



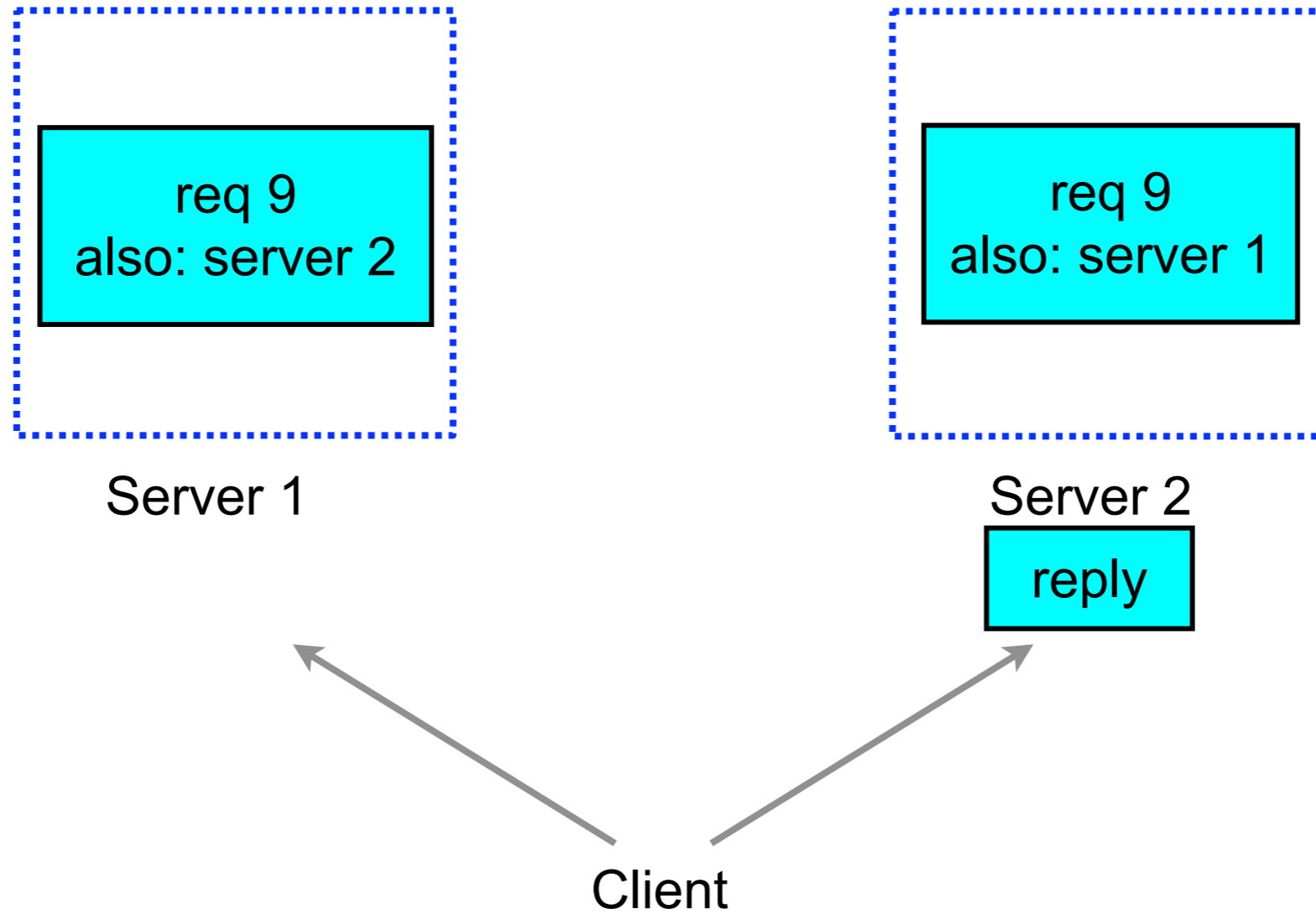
# Backup Requests: Bad Case

---



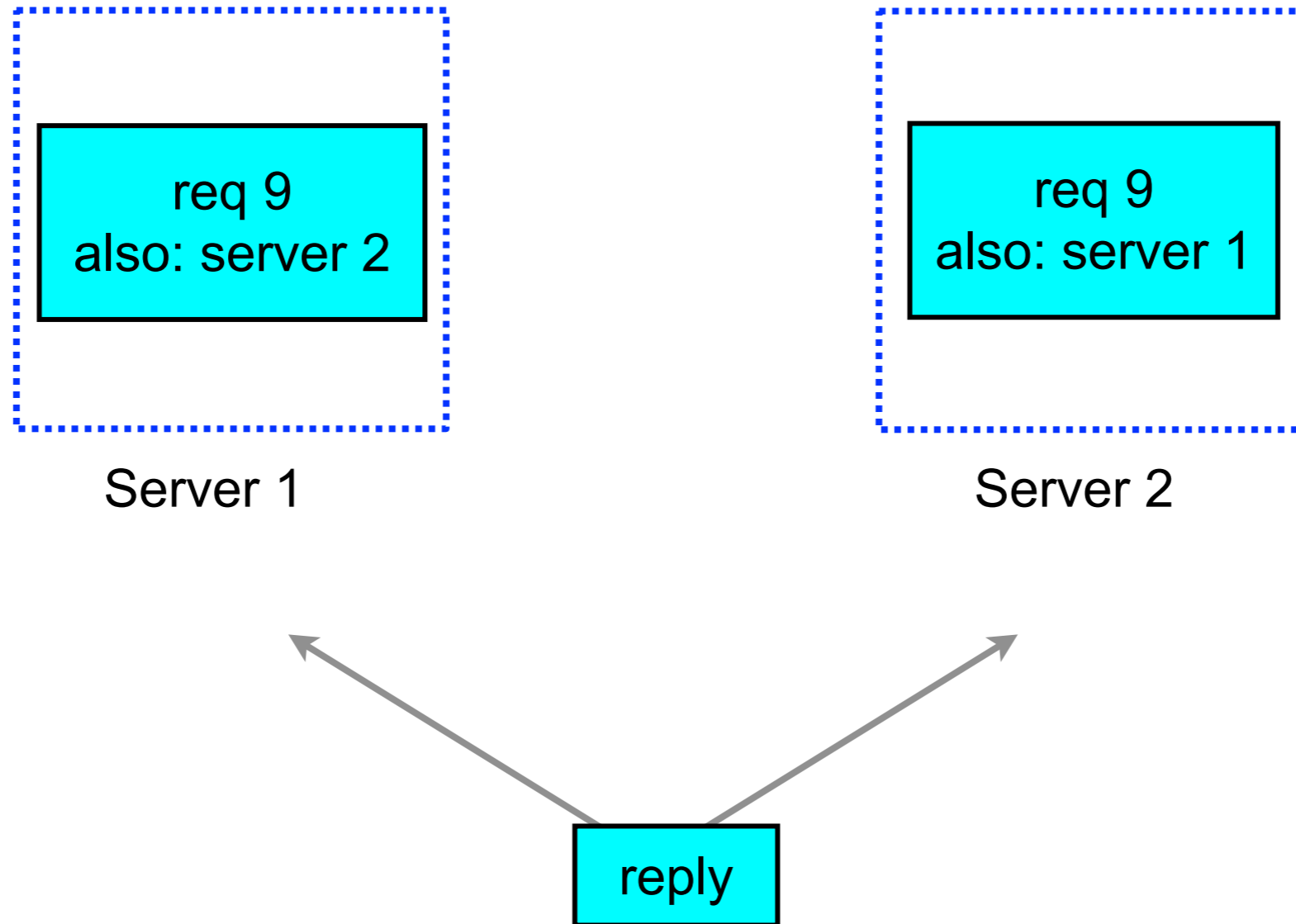
# Backup Requests: Bad Case

---



# Backup Requests: Bad Case

---



# Backup Requests w/ Cross-Server Cancellation

---

- Read operations in distributed file system client
  - send request to first replica
  - wait 2 ms, and send to second replica
  - servers cancel request on other replica when starting read
- Time for bigtable monitoring ops that touch disk



# Backup Requests w/ Cross-Server Cancellation

---

- Read operations in distributed file system client
  - send request to first replica
  - wait 2 ms, and send to second replica
  - servers cancel request on other replica when starting read
- Time for bigtable monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms





# Backup Requests w/ Cross-Server Cancellation

---

- Read operations in distributed file system client
  - send request to first replica
  - wait 2 ms, and send to second replica
  - servers cancel request on other replica when starting read
- Time for bigtable monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms

*Note: A red box highlights the 99%ile values (67 ms and 38 ms), and a grey arrow points from the 67 ms value to the 38 ms value with a red '-43%' label above it.*



# Backup Requests w/ Cross-Server Cancellation

---

- Read operations in distributed file system client
  - send request to first replica
  - wait 2 ms, and send to second replica
  - servers cancel request on other replica when starting read
- Time for bigtable monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms



# Backup Requests w/ Cross-Server Cancellation

- Read operations in distributed file system client
  - send request to first replica
  - wait 2 ms, and send to second replica
  - servers cancel request on other replica when starting read
- Time for bigtable monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

-38%



# Backup Requests w/ Cross-Server Cancellation

---

- Read operations in distributed file system client
  - send request to first replica
  - wait 2 ms, and send to second replica
  - servers cancel request on other replica when starting read
- Time for bigtable monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

Backups cause about ~1% extra disk reads



# Backup Requests w/ Cross-Server Cancellation

---

- Read operations in distributed file system client
  - send request to first replica
  - wait 2 ms, and send to second replica
  - servers cancel request on other replica when starting read
- Time for bigtable monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms



# Backup Requests w/ Cross-Server Cancellation

---

- Read operations in distributed file system client
  - send request to first replica
  - wait 2 ms, and send to second replica
  - servers cancel request on other replica when starting read
- Time for bigtable monitoring ops that touch disk

Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

Backups w/big sort job gives same read latencies as no backups w/ idle cluster!



# Backup Request Variants

---

- Many variants possible:
- Send to third replica after longer delay
  - sending to two gives almost all the benefit, however.
- Keep requests in other queues, but reduce priority
- Can handle Reed-Solomon reconstruction similarly



# Tainted Partial Results

---

- Many systems can tolerate inexact results
  - information retrieval systems
    - search 99.9% of docs in 200ms better than 100% in 1000ms
  - complex web pages with many sub-components
    - e.g. okay to skip spelling correction service if it is slow
- Design to proactively abandon slow subsystems
  - set cutoffs dynamically based on recent measurements
    - can tradeoff completeness vs. responsiveness
  - important to **mark such results as tainted** in caches





# Hardware Trends

---

- Some good:
  - lower latency networks make things like backup request cancellations work better
- Some not so good:
  - plethora of CPU and device **sleep modes save power**, but **add latency variability**
  - higher number of “wimpy” cores => higher fanout => more variability
- **Software techniques can reduce variability despite increasing variability in underlying hardware**



# Conclusions

---

- **Tolerating variability**
  - important for large-scale online services
  - large fanout magnifies importance
  - makes services more responsive
  - saves significant computing resources
- **Collection of techniques**
  - **general good engineering practices**
    - prioritized server queues, careful management of background activities
  - **cross-request adaptation**
    - load balancing, micro-partitioning
  - **within-request adaptation**
    - backup requests, backup requests w/ cancellation, tainted results



# Thanks

---

- Joint work with Luiz Barroso and many others at Google
- Questions?

