Prophecy: Using History for High-Throughput Fault Tolerance

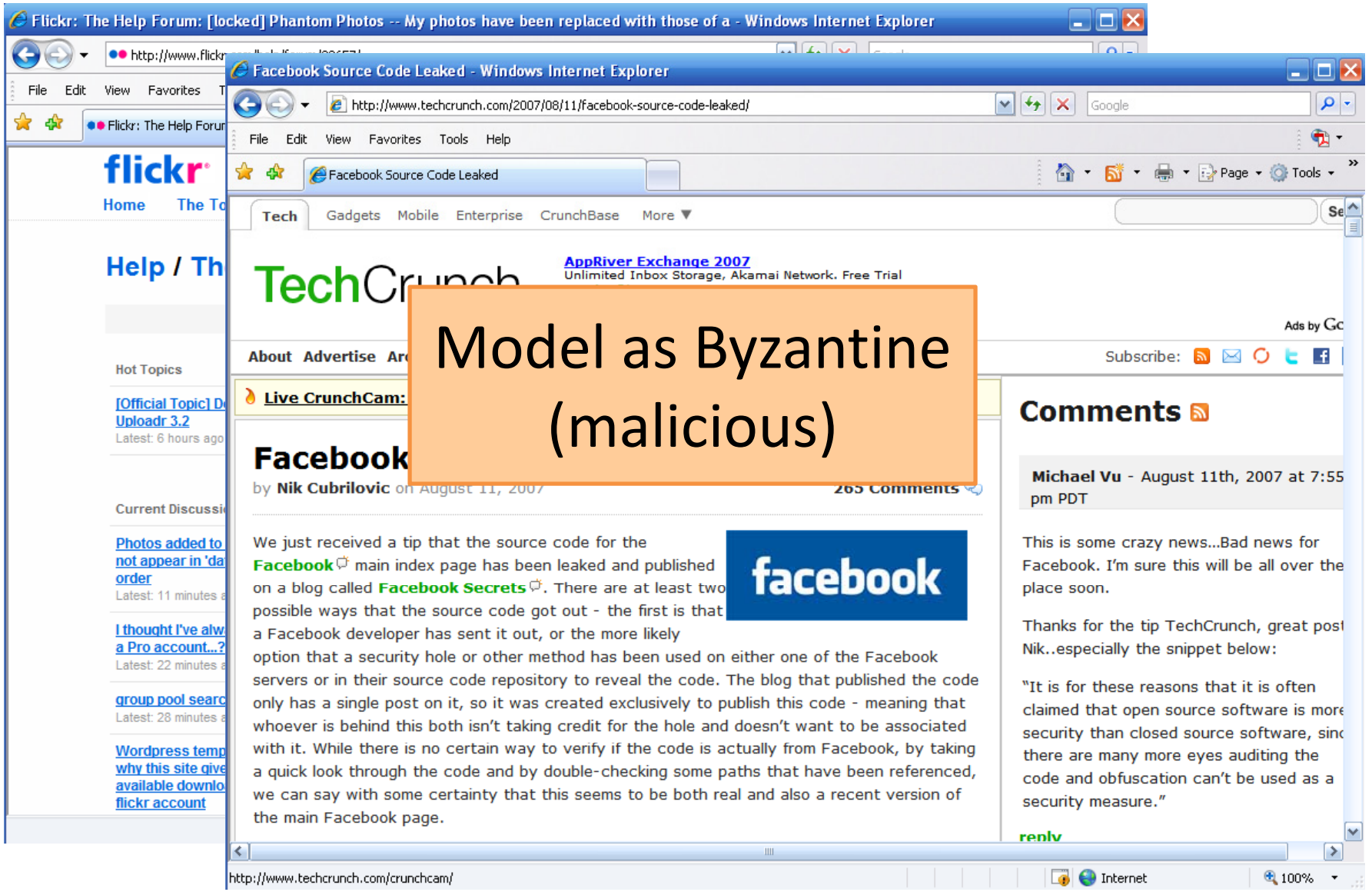
Siddhartha Sen

Joint work with Wyatt Lloyd and Mike Freedman

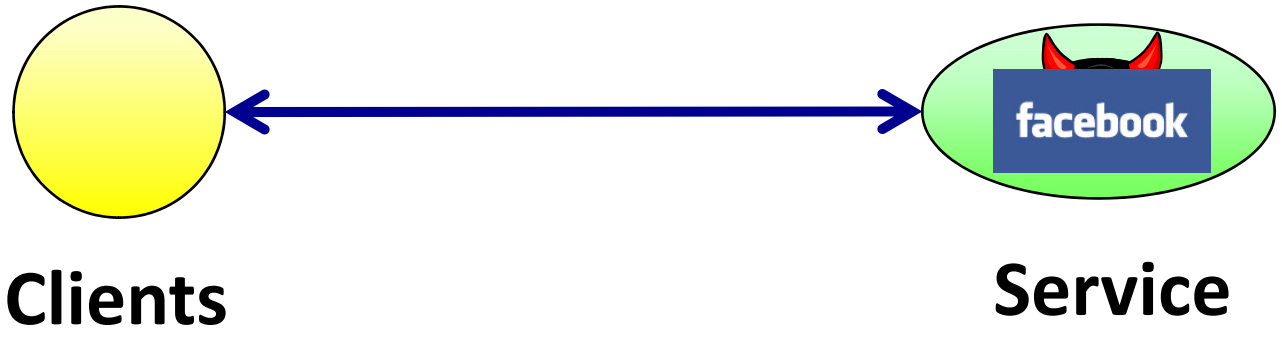
Princeton University

Non-crash failures happen

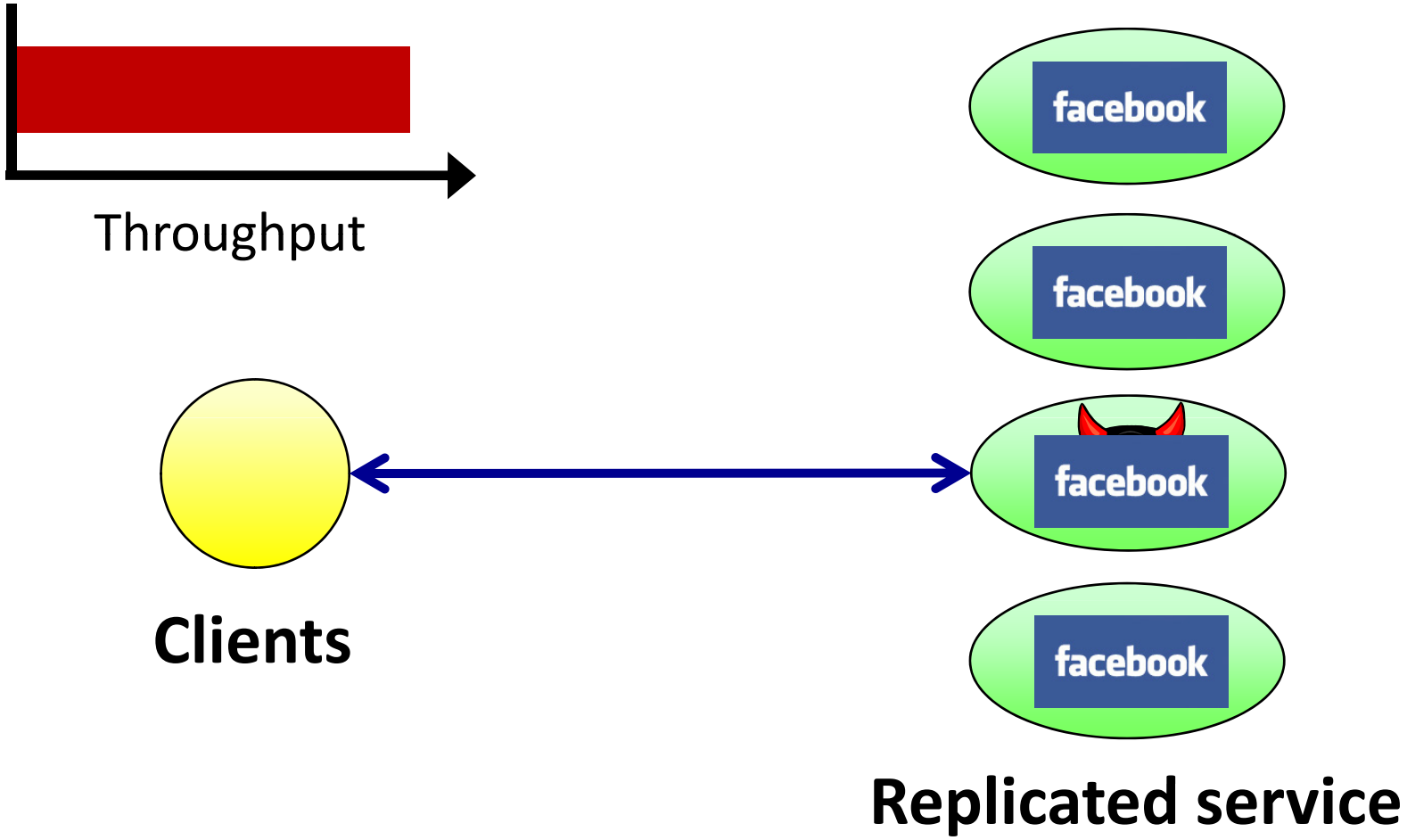
Non-crash failures happen



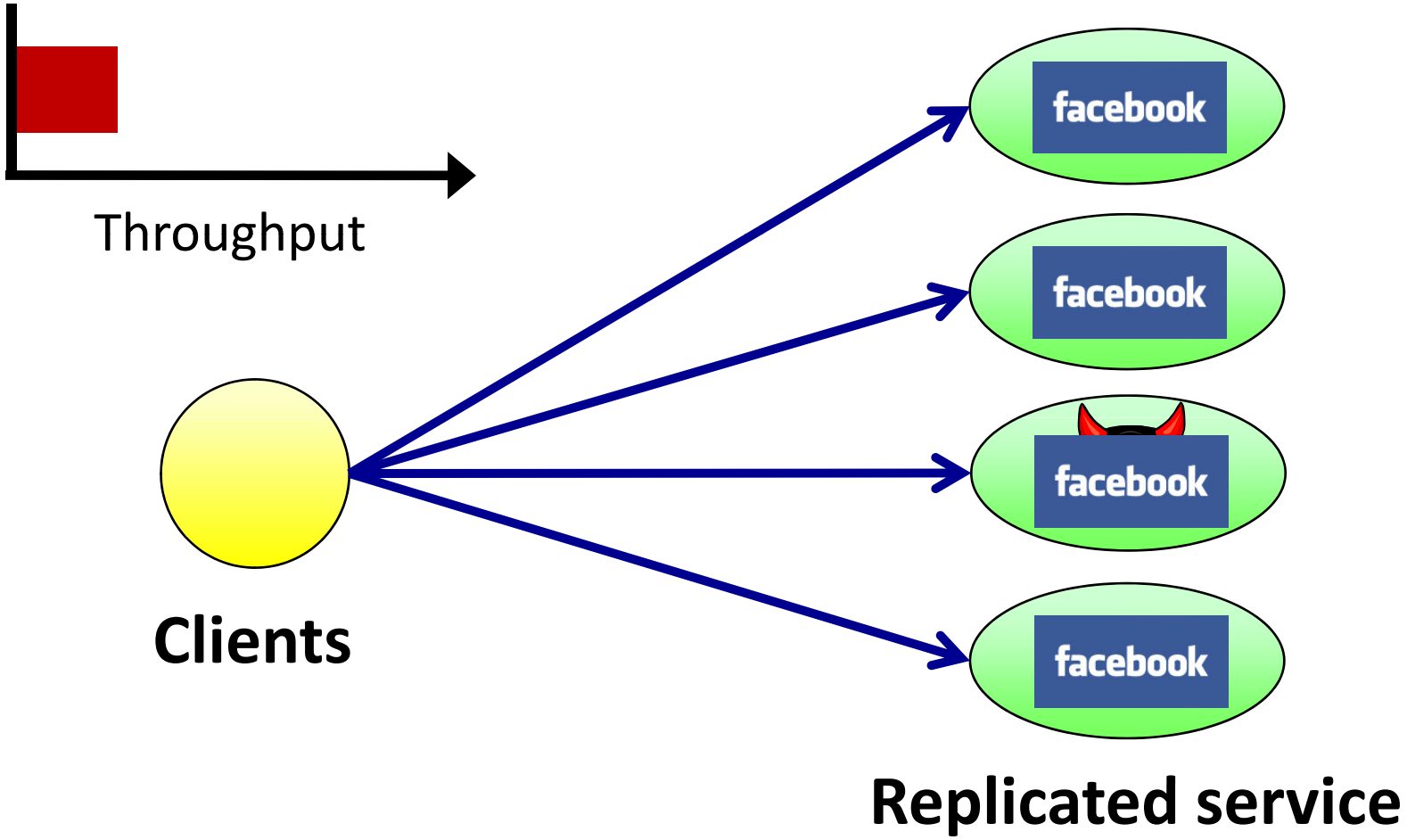
Mask Byzantine faults



Mask Byzantine faults

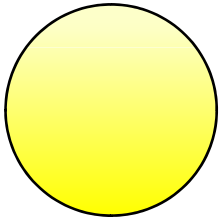


Mask Byzantine faults

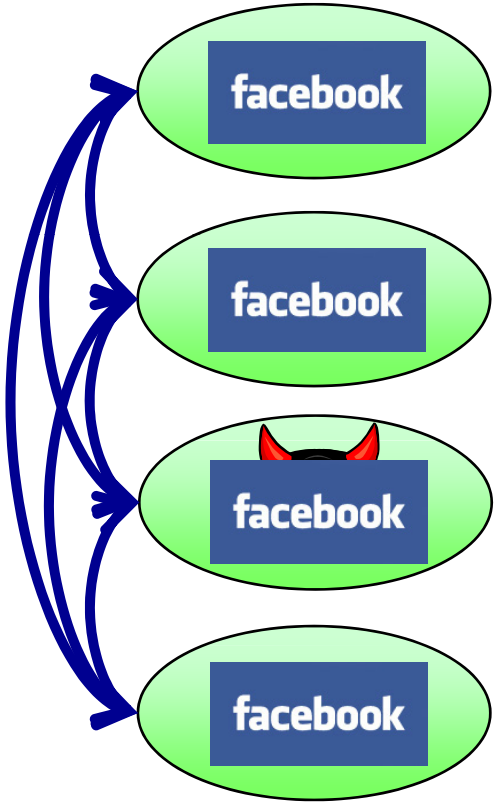


Mask Byzantine faults

Throughput

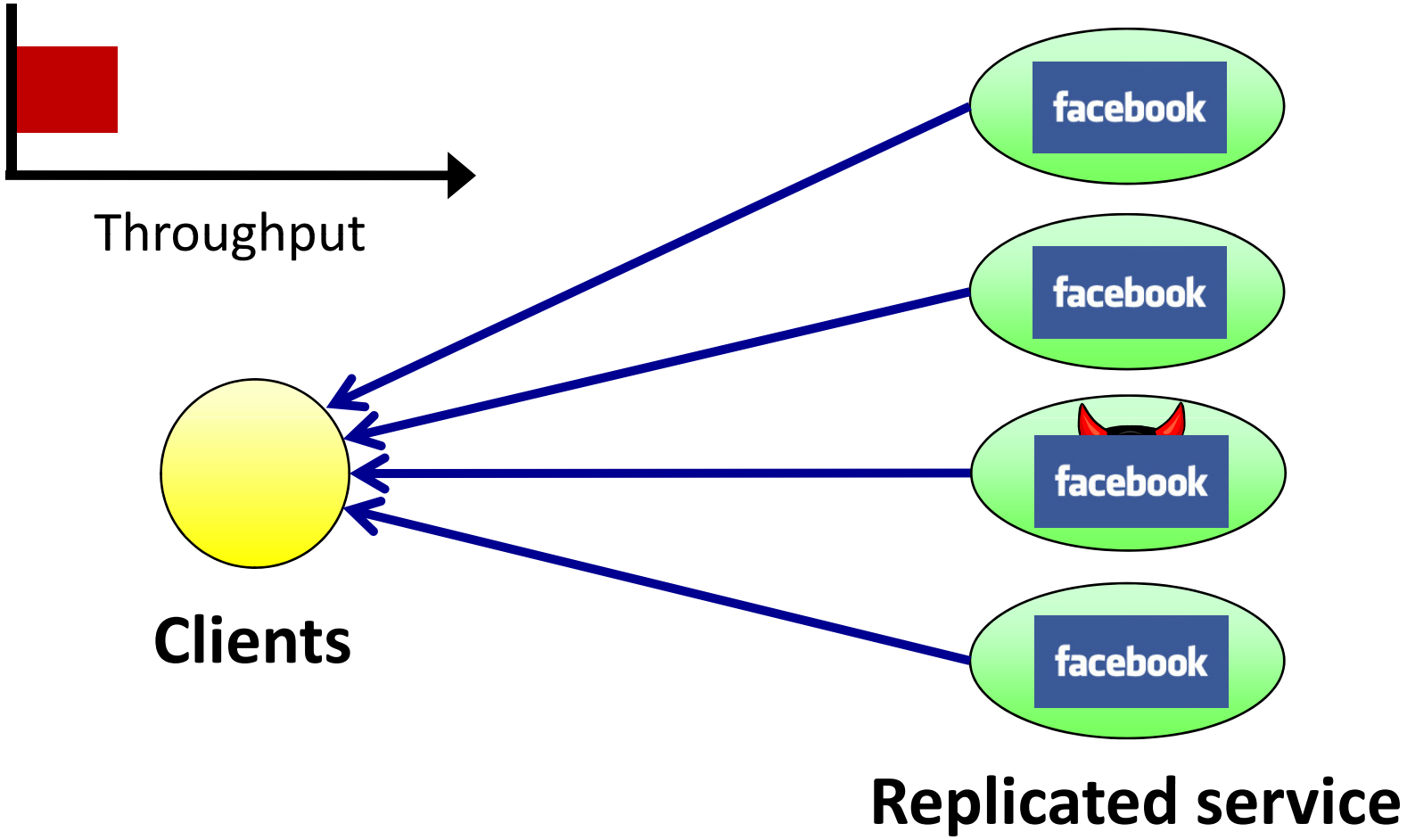


Clients

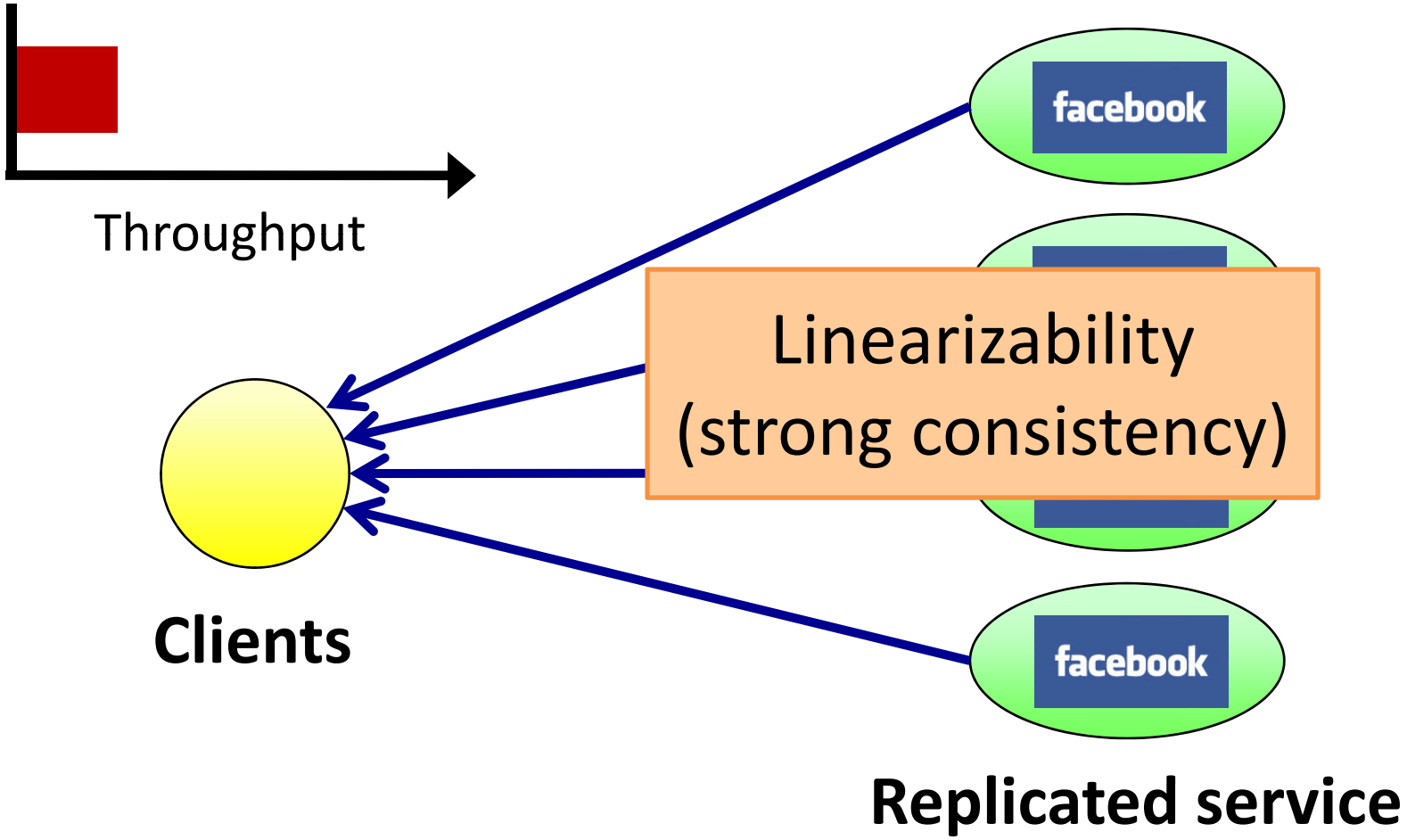


Replicated service

Mask Byzantine faults



Mask Byzantine faults



Byzantine fault tolerance (BFT)

- Low throughput
- Modifies clients
- Long-lived sessions

Prophecy

- High throughput + good consistency
- No free lunch:
  - Read-mostly workloads
  - Slightly weakened consistency

Byzantine fault tolerance (BFT)

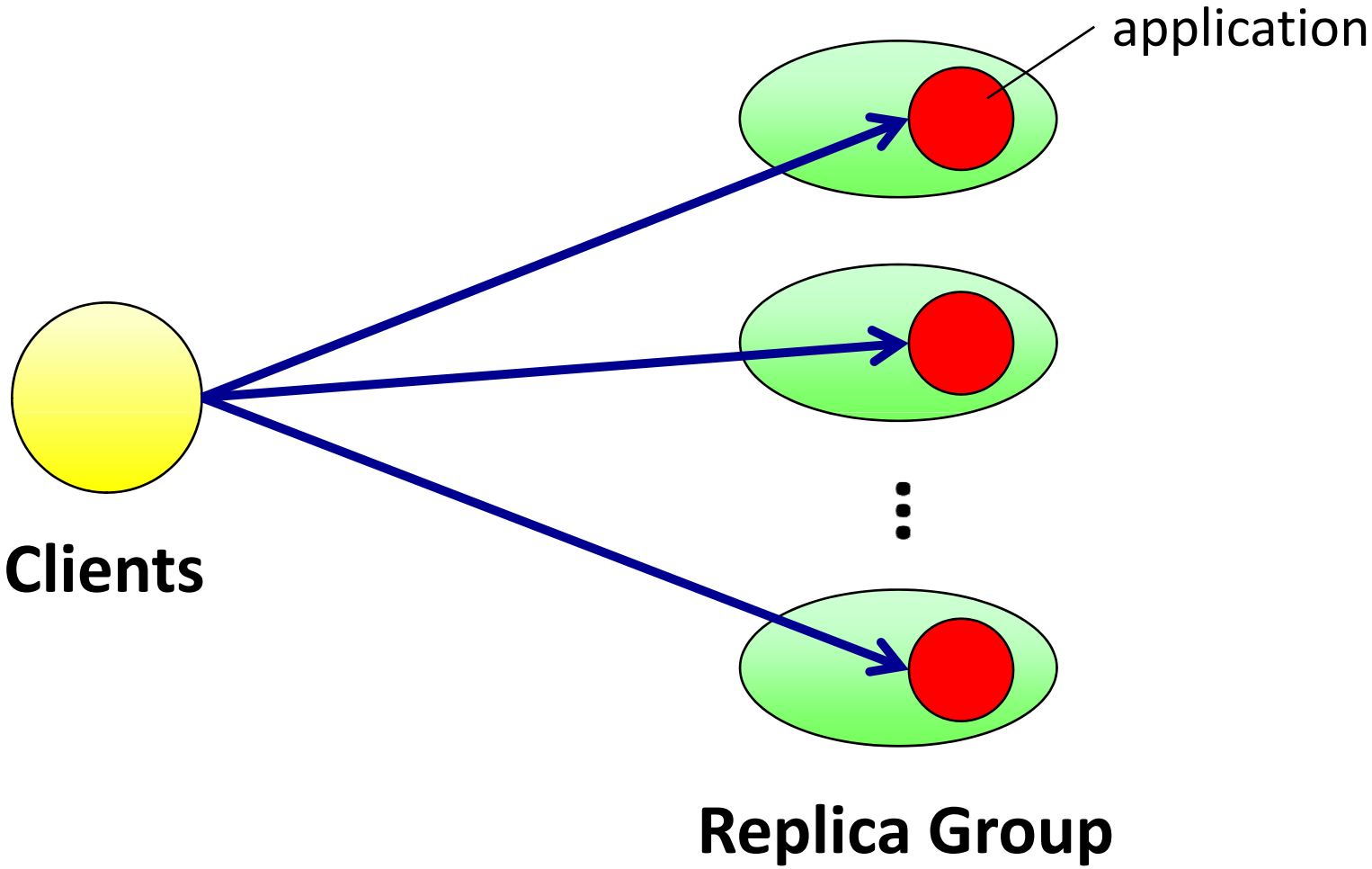
- Low throughput

D-Prophecy

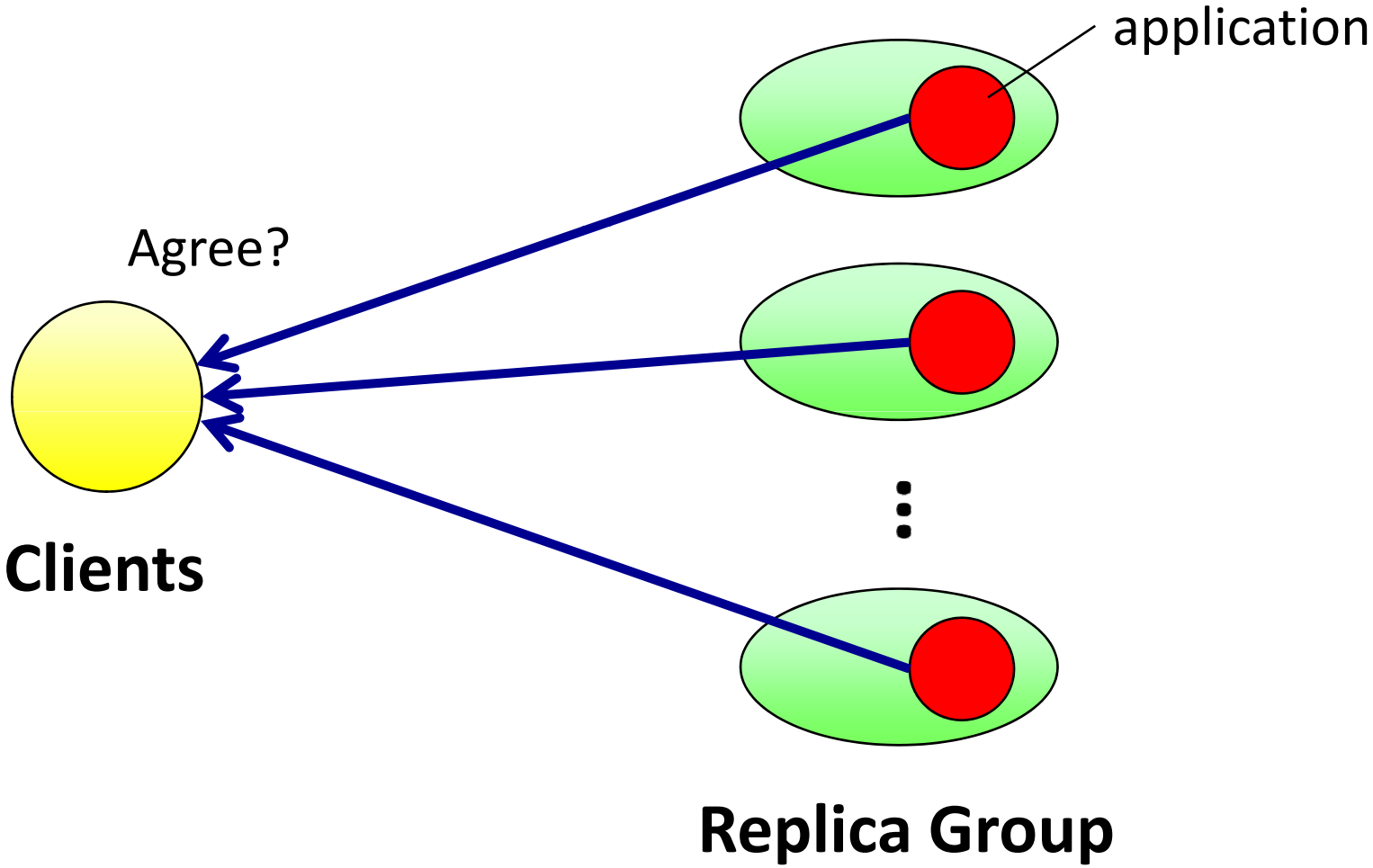
- Modifies clients
- Long-lived sessions

Prophecy

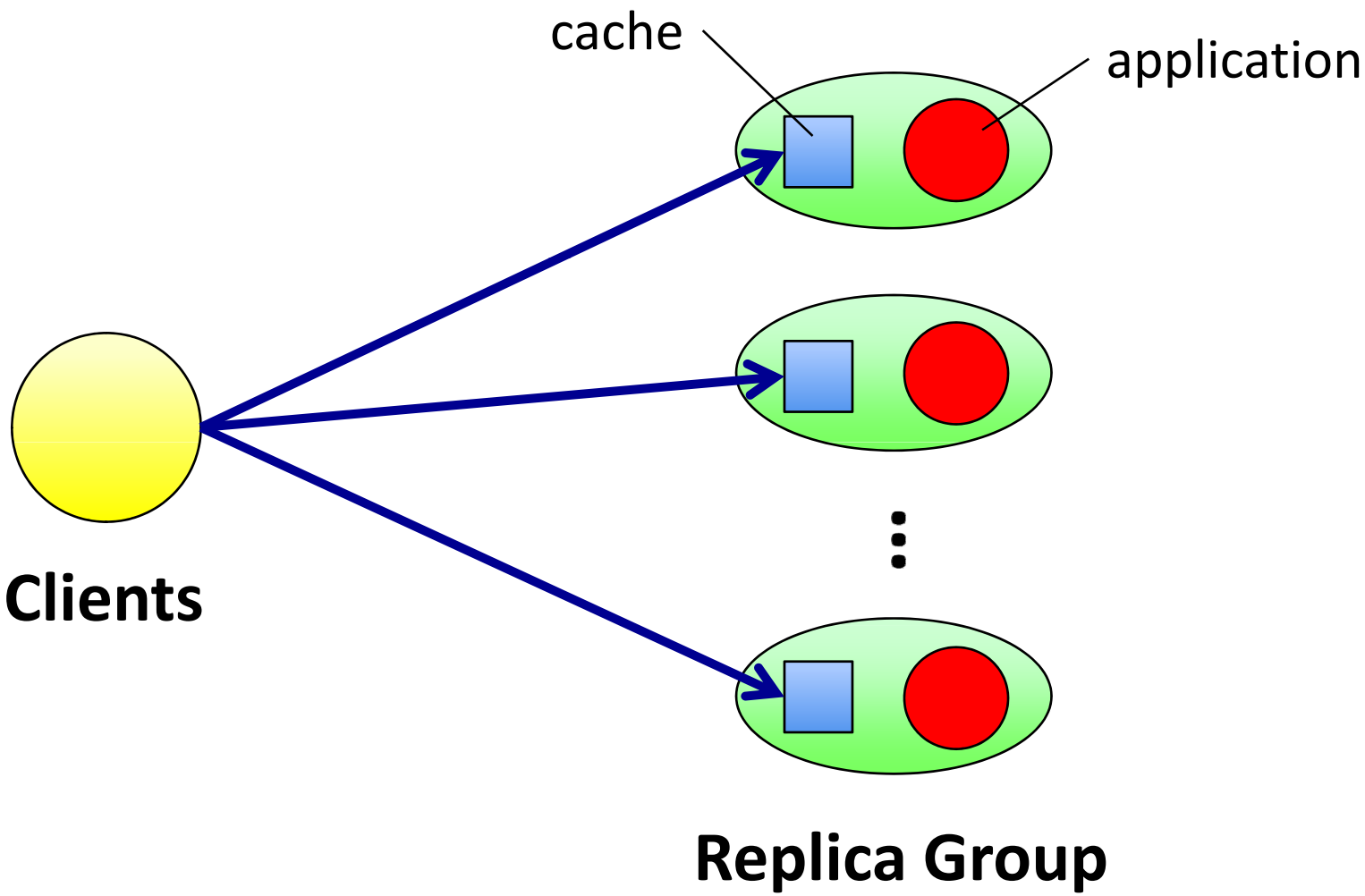
Traditional BFT reads



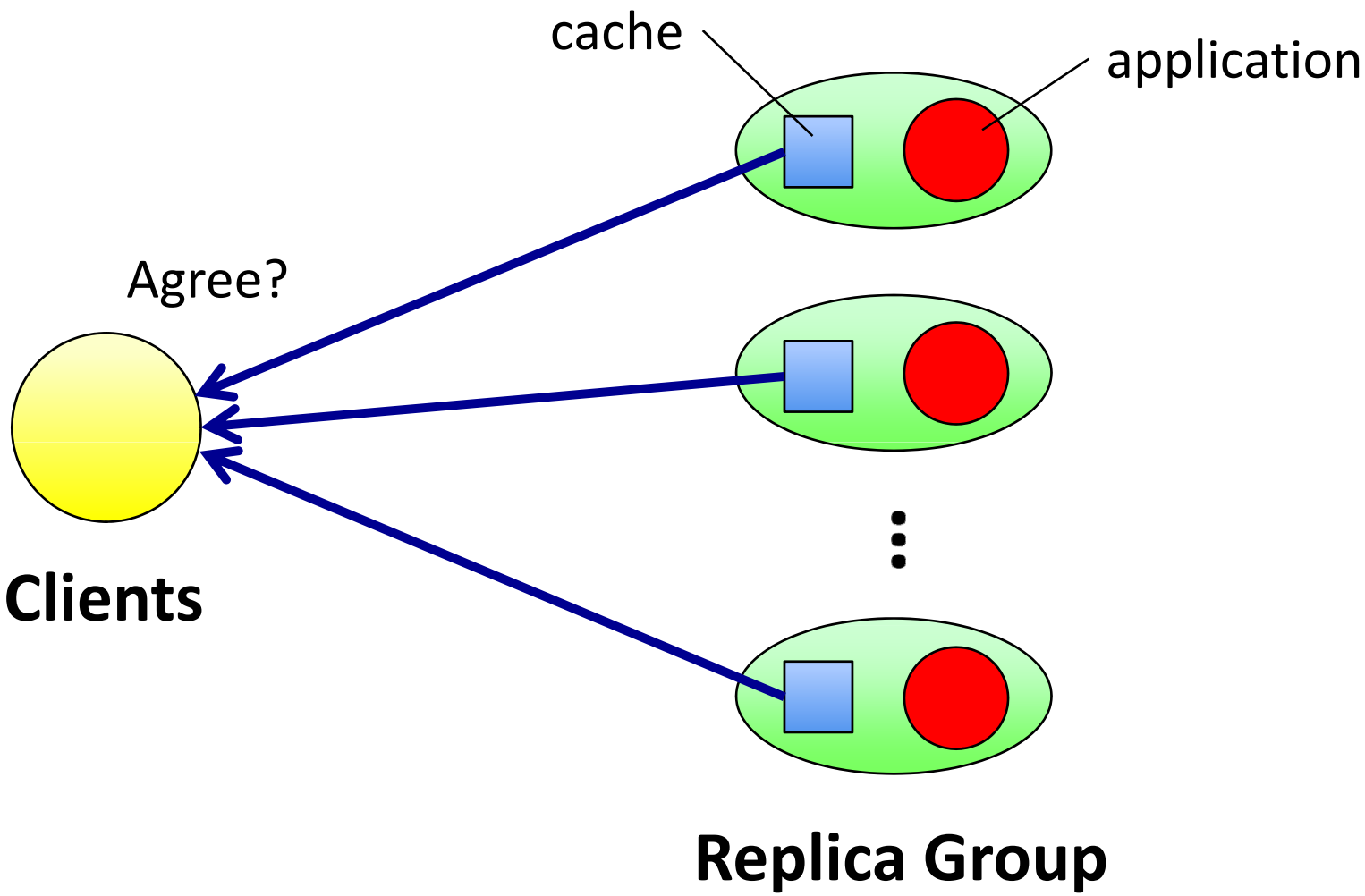
Traditional BFT reads



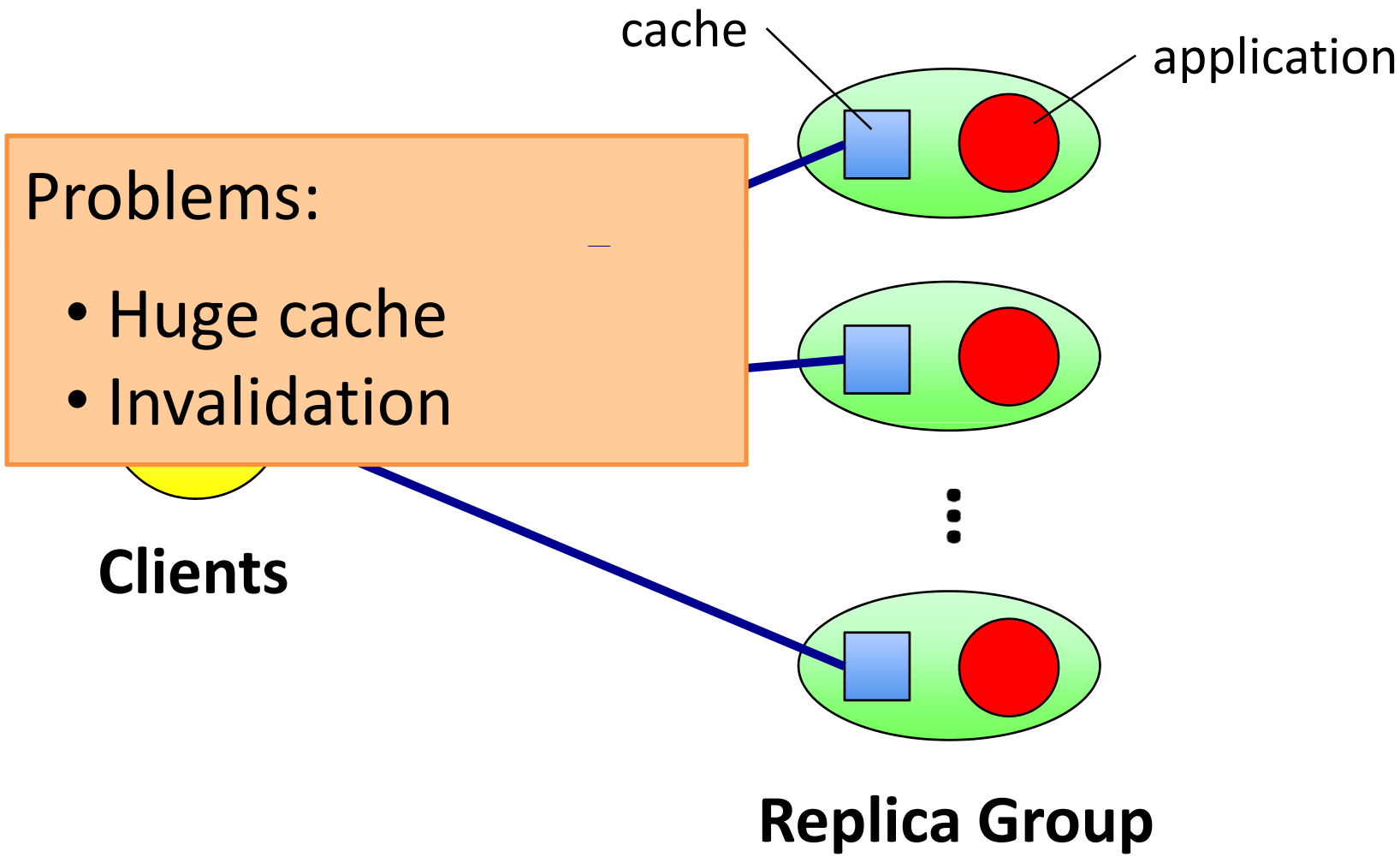
A cache solution



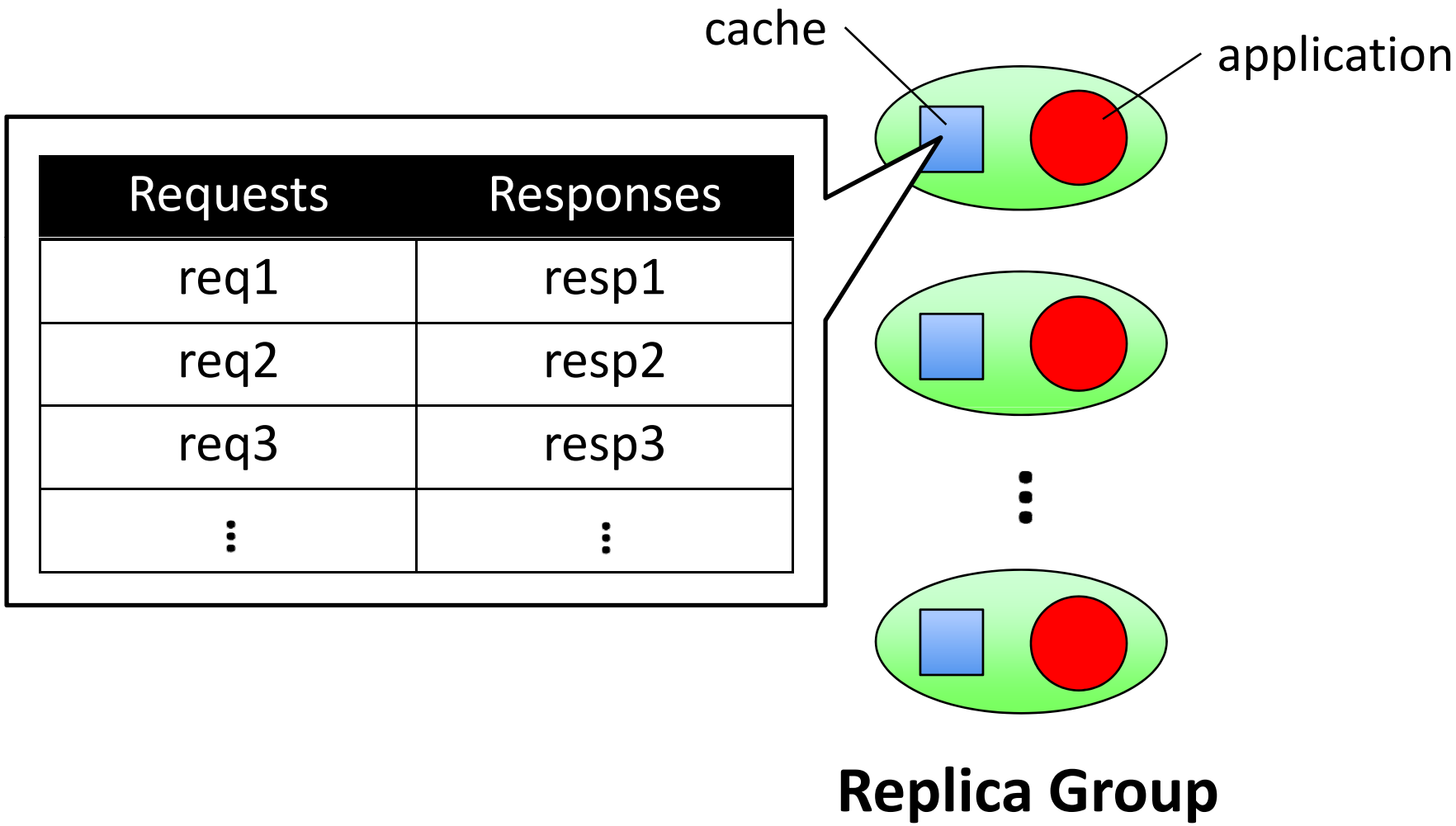
A cache solution



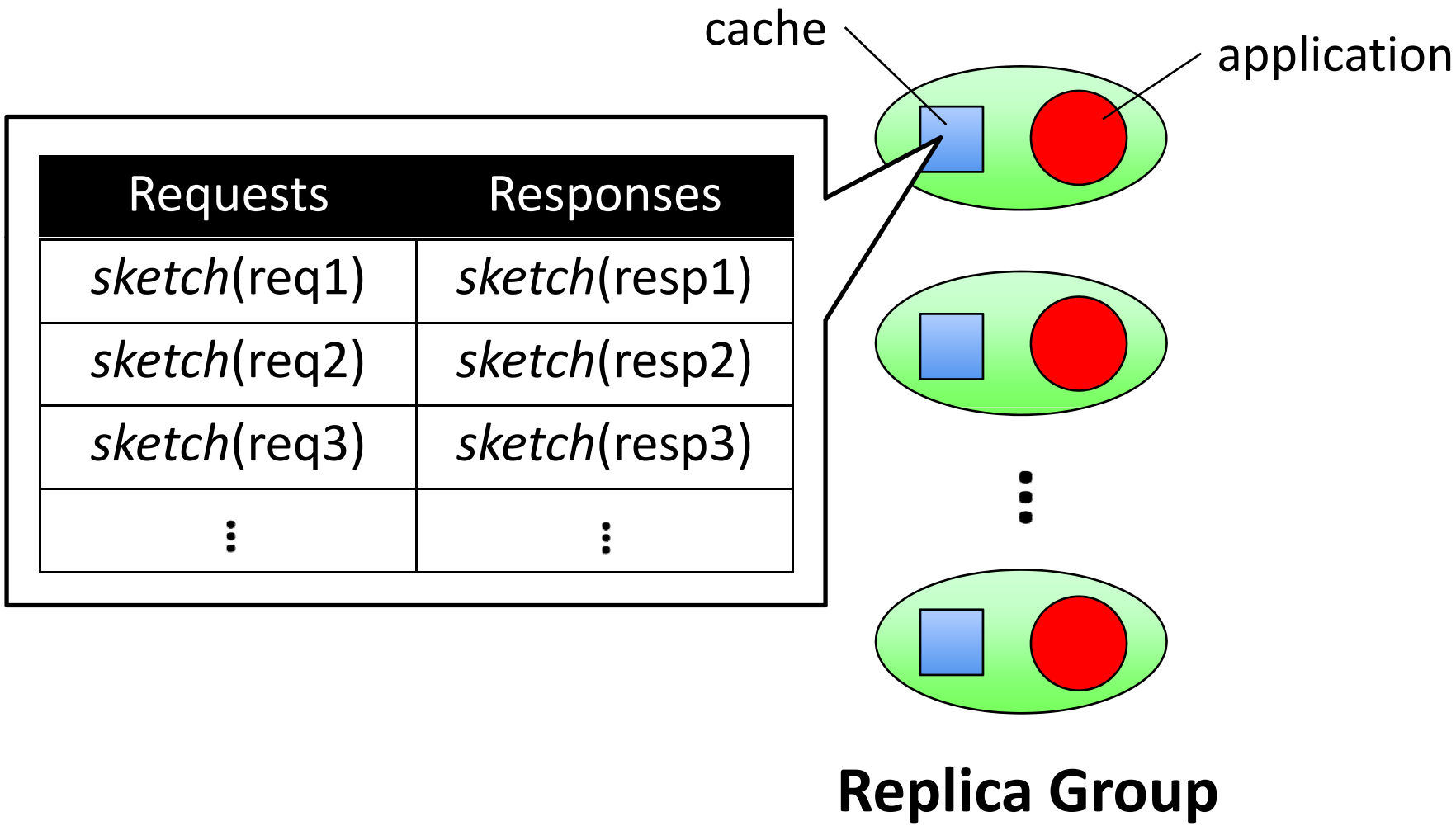
A cache solution



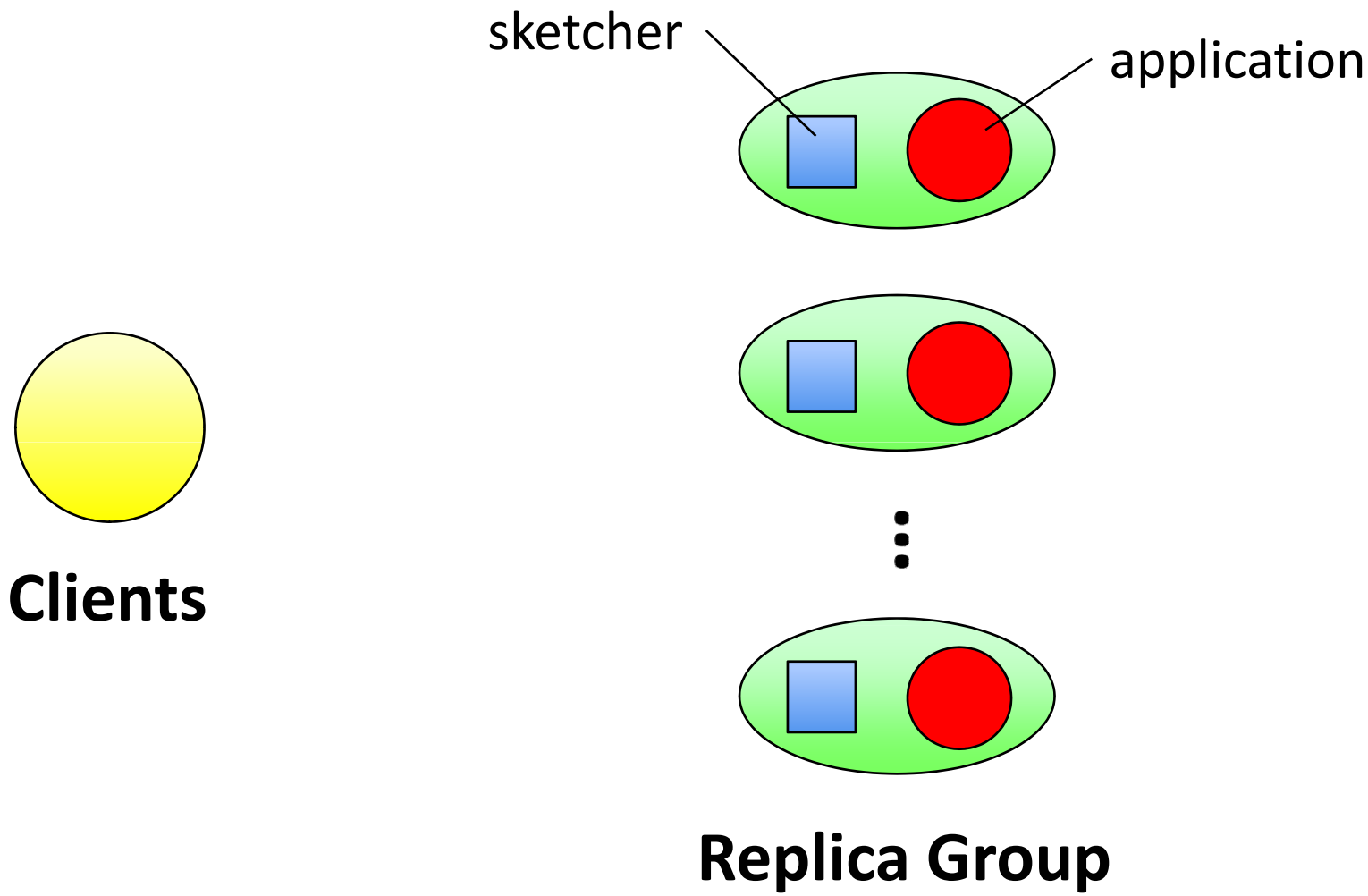
A compact cache



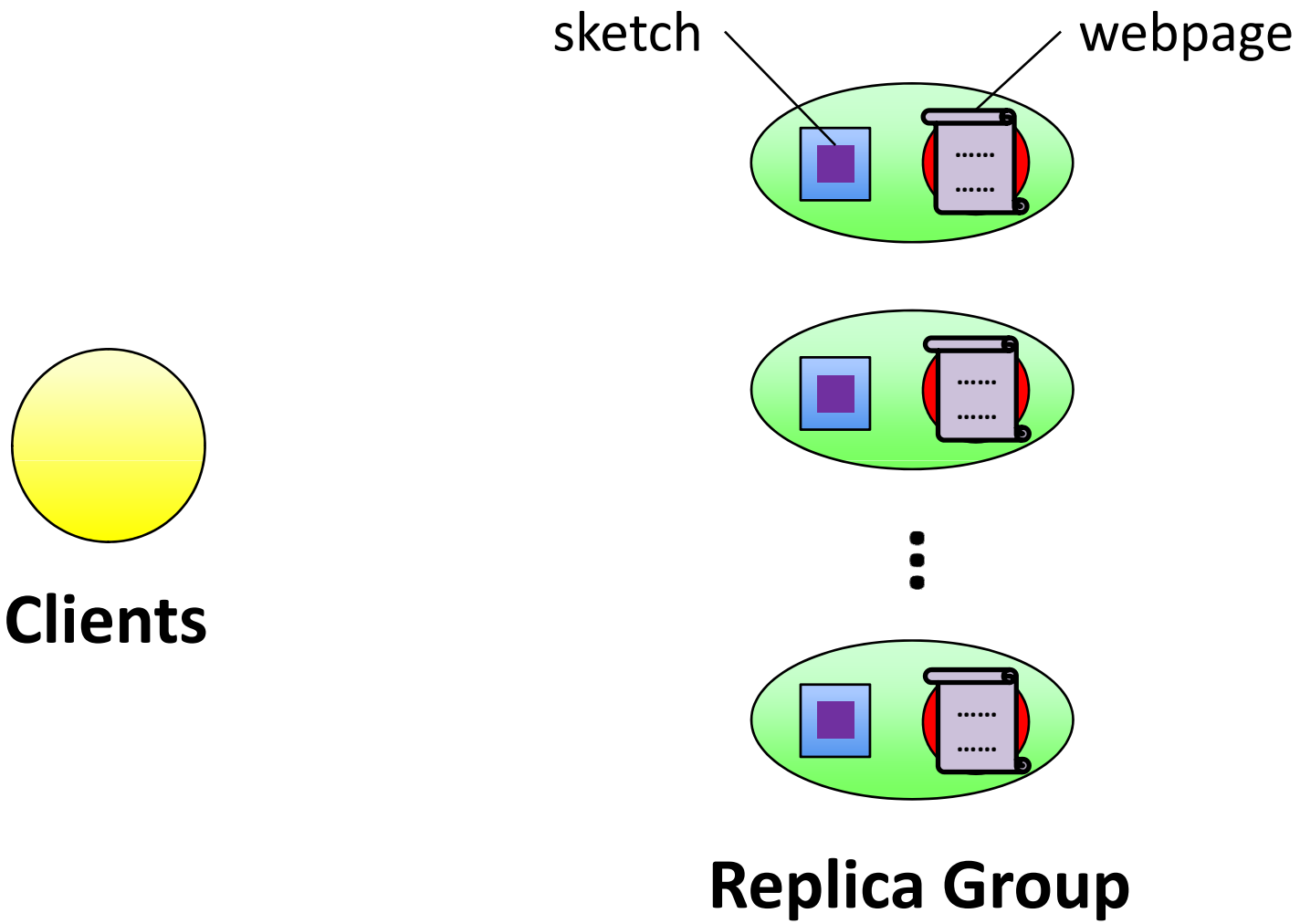
A compact cache



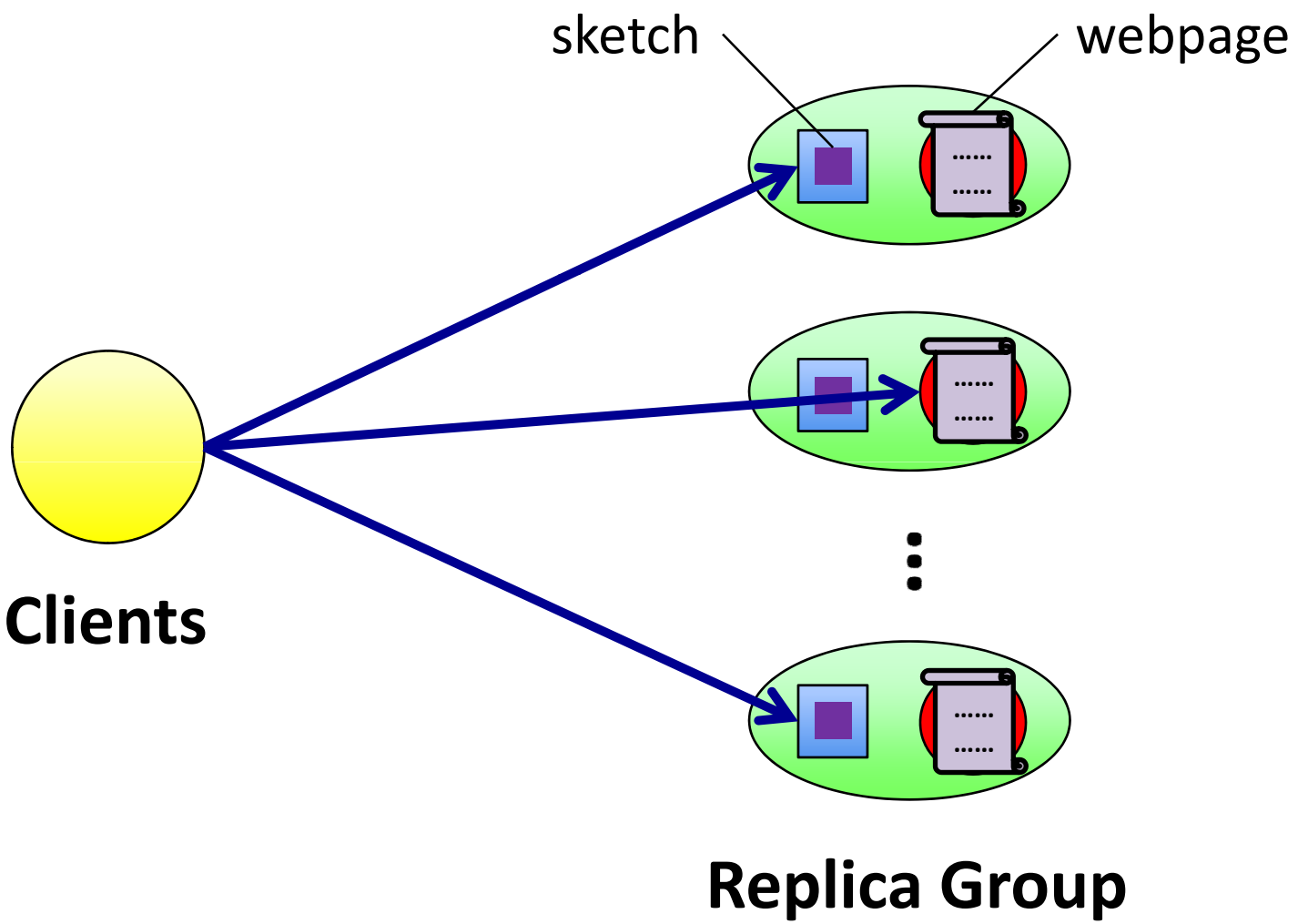
A sketcher



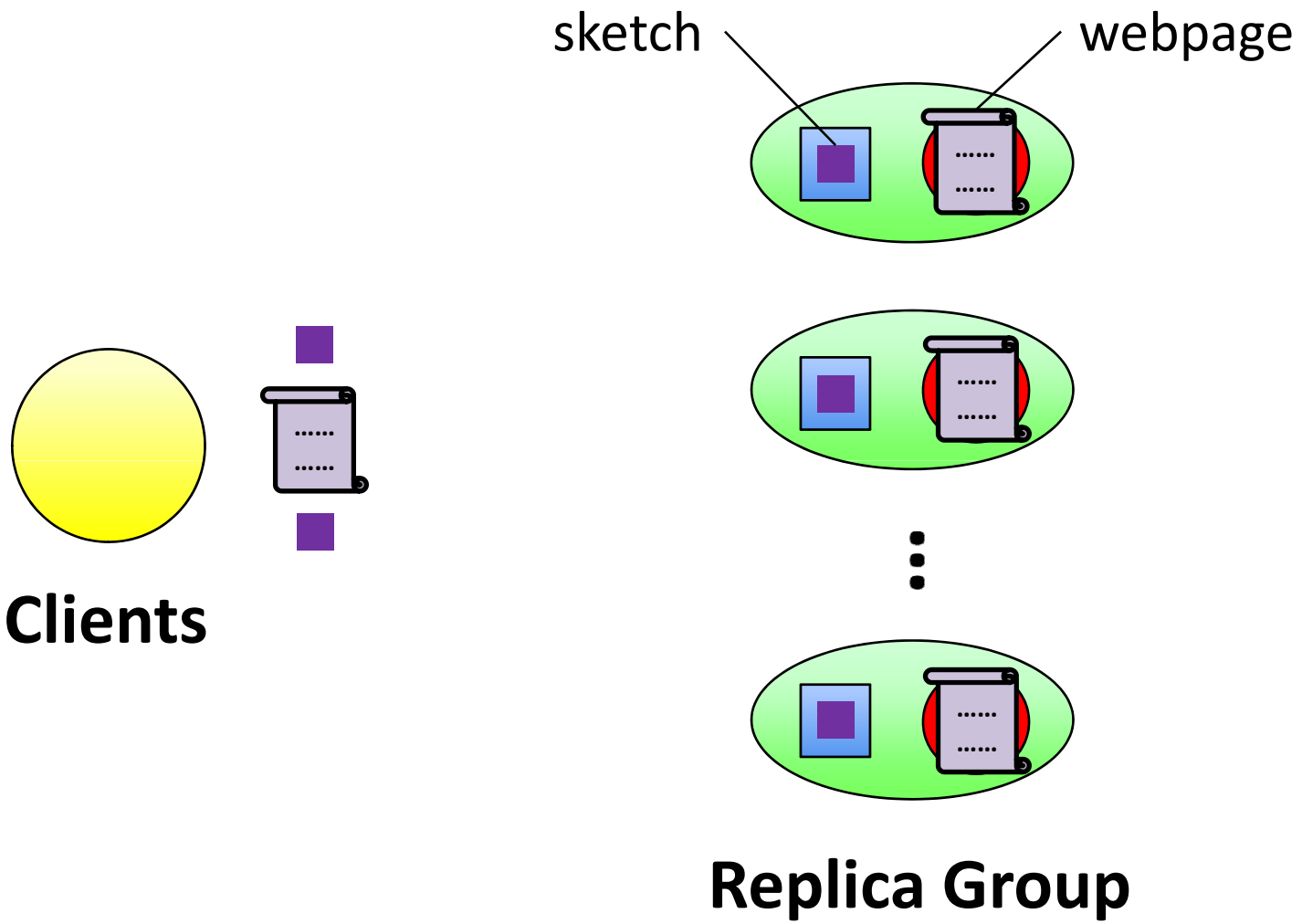
Executing a read



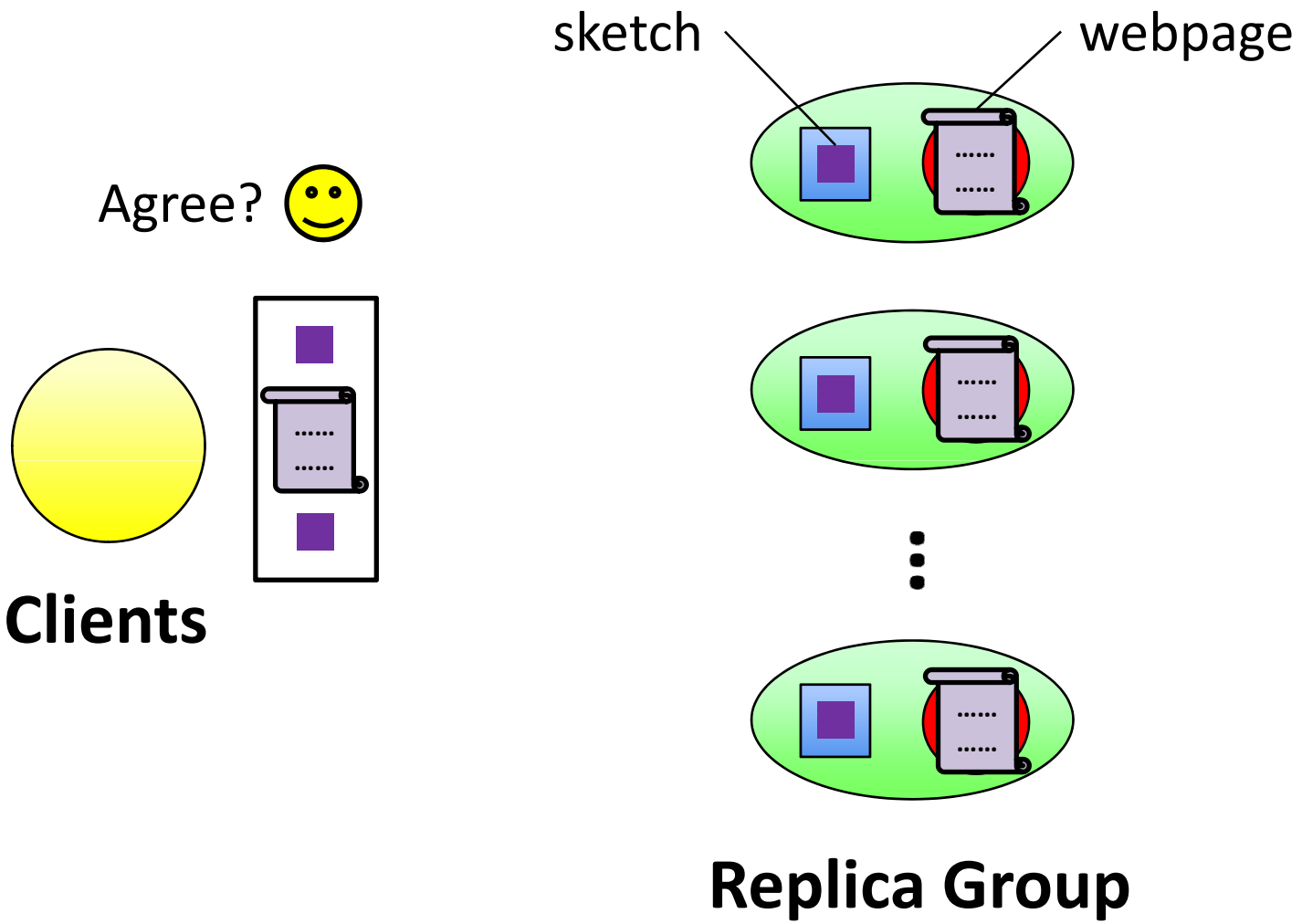
Executing a read



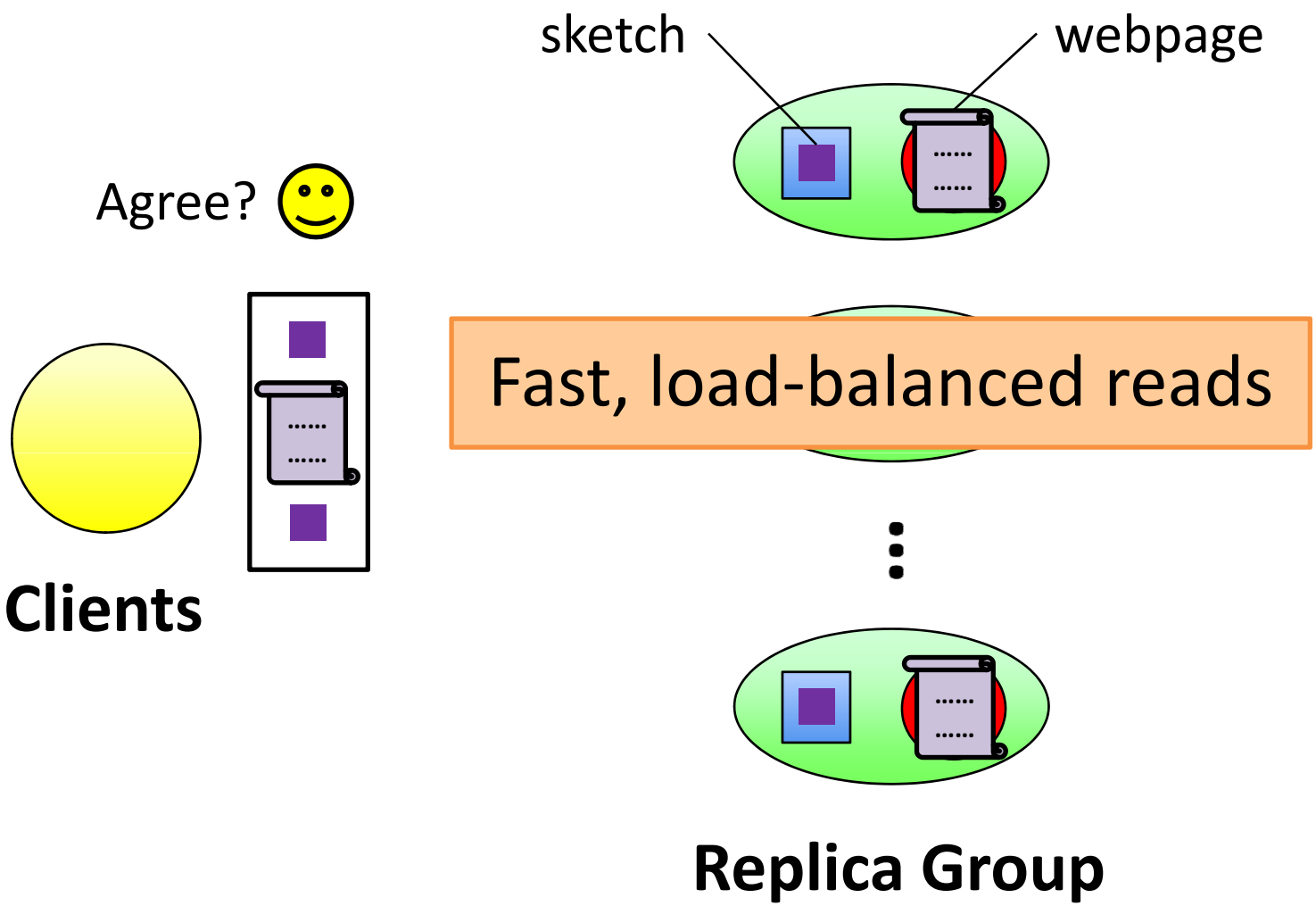
Executing a read



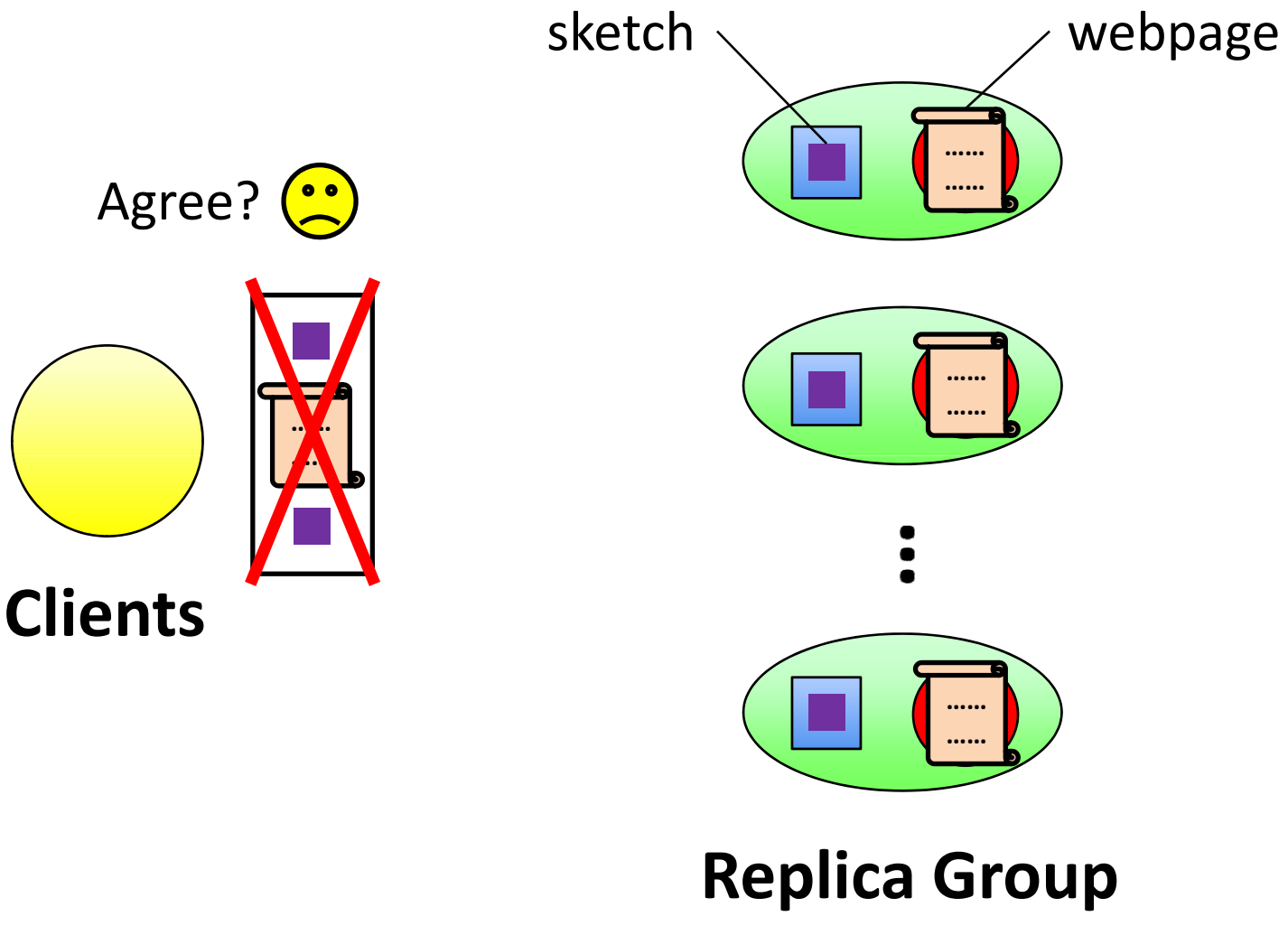
Executing a read



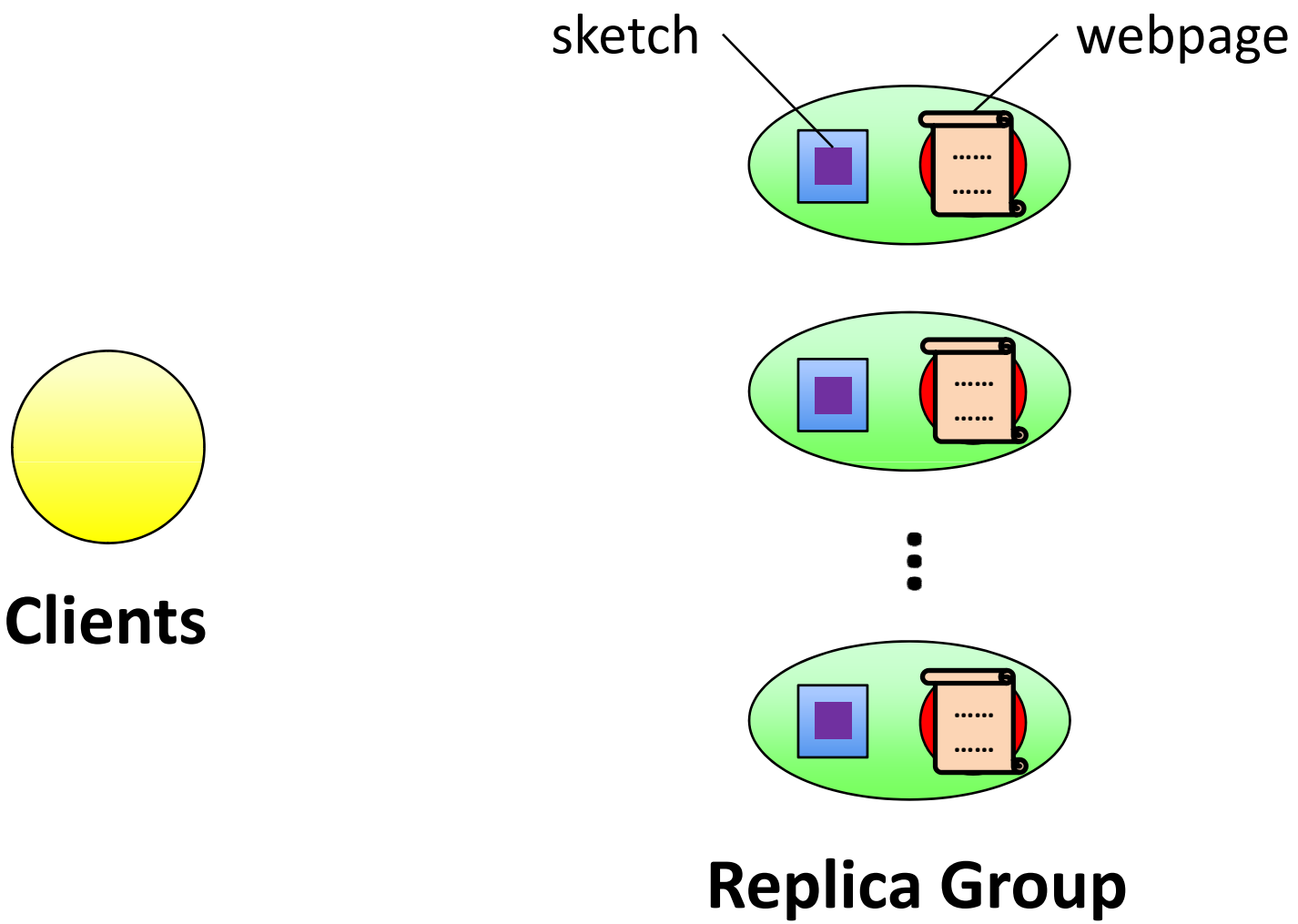
Executing a read



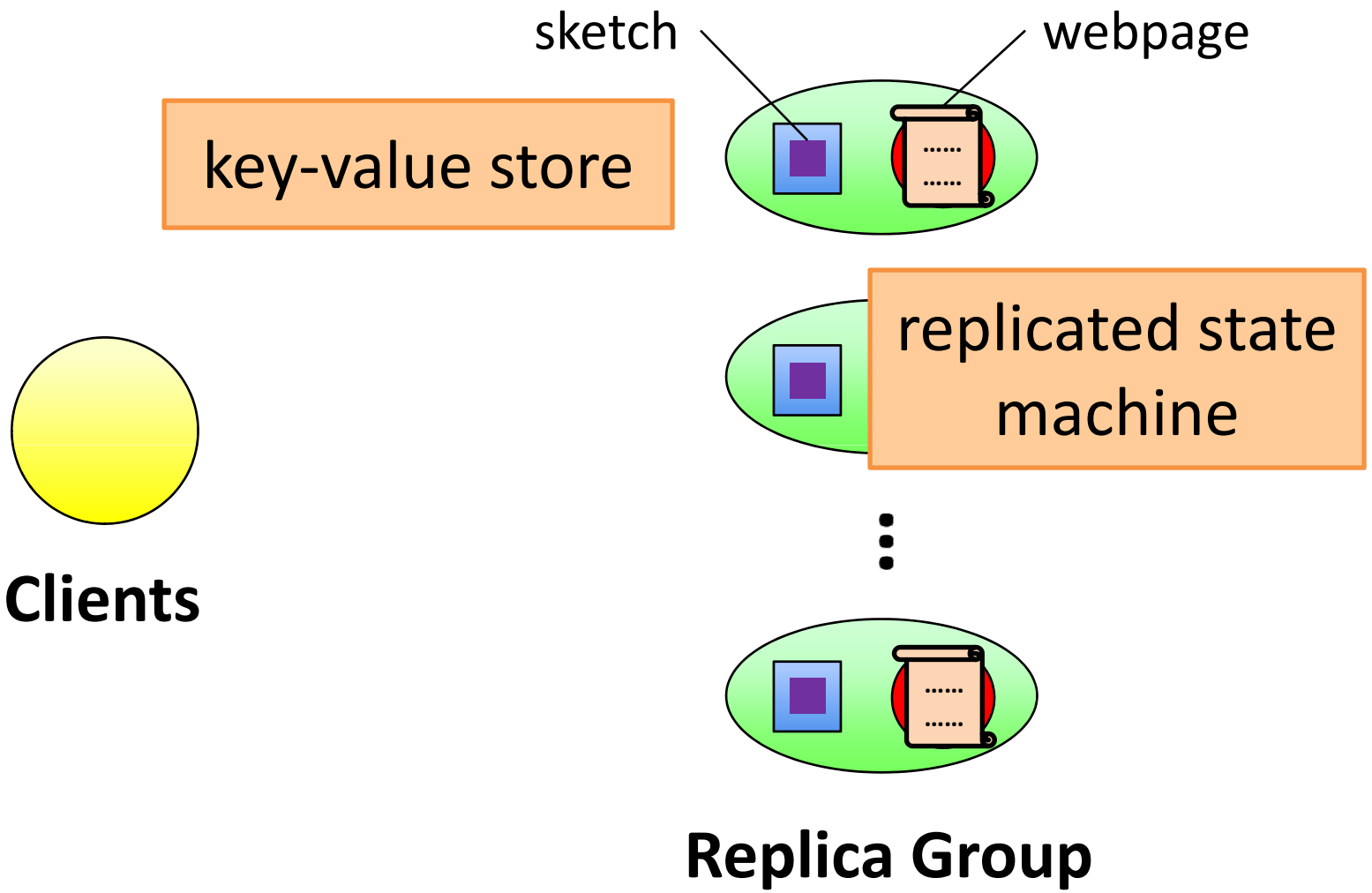
Executing a read



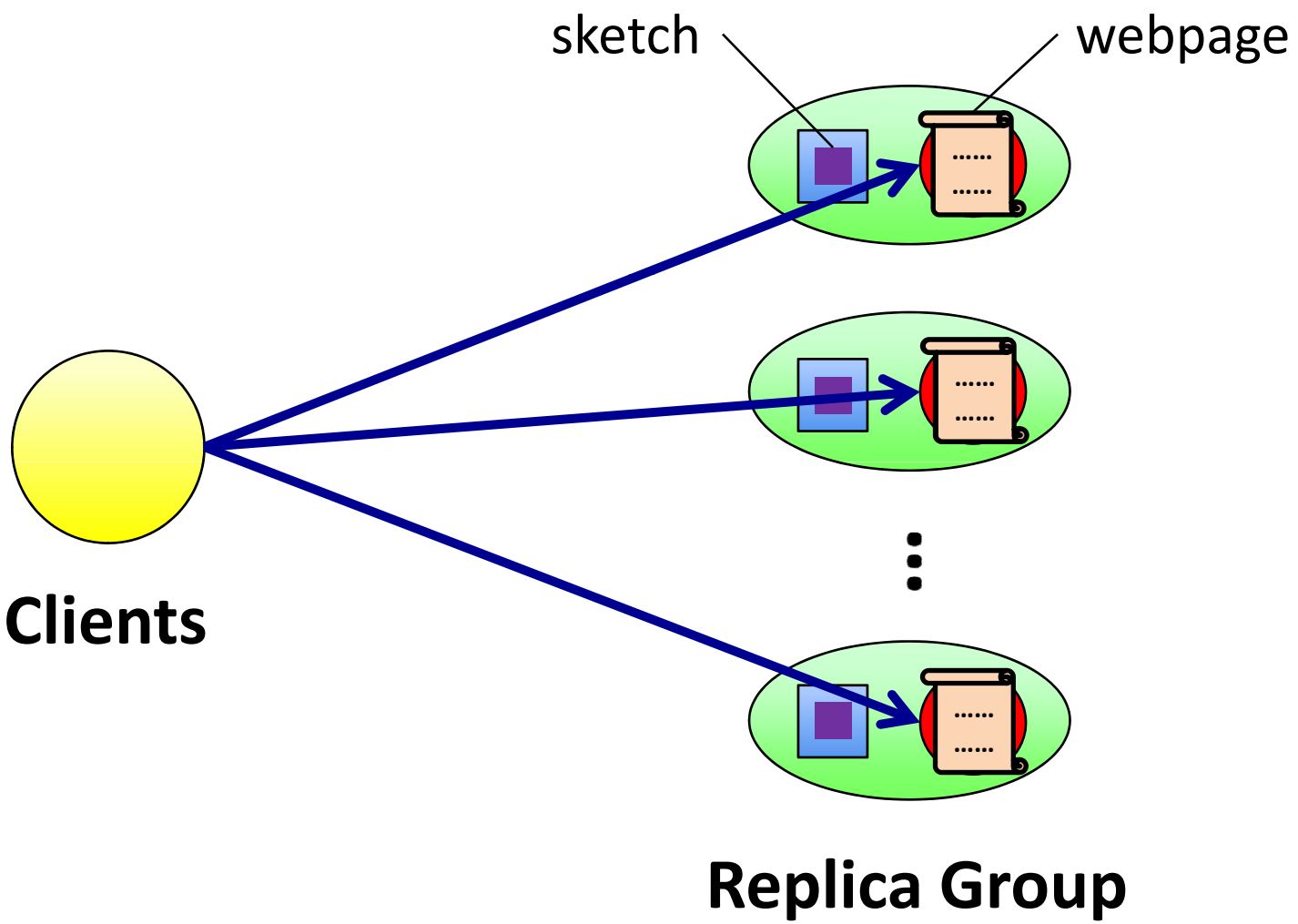
Executing a read



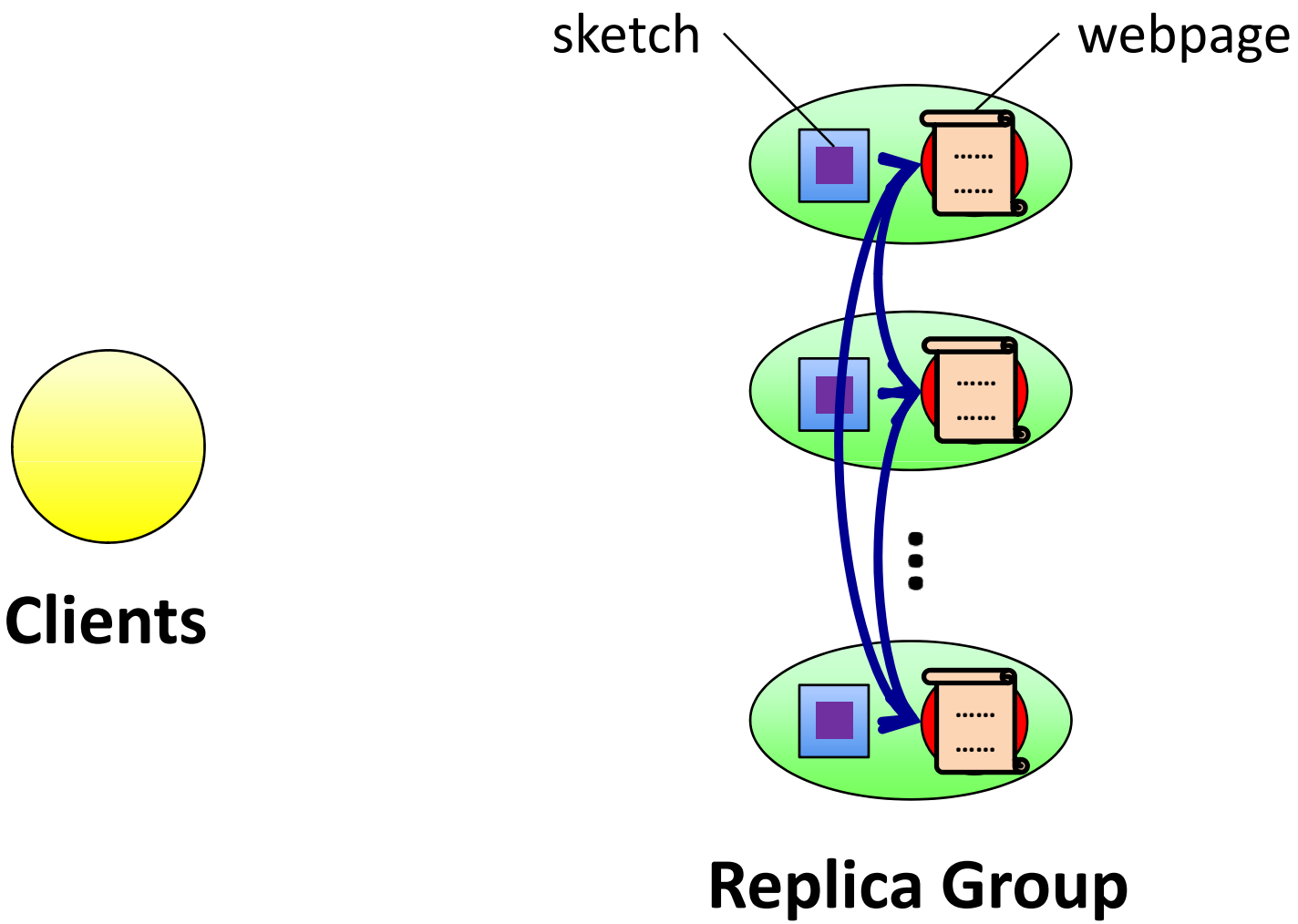
Executing a read



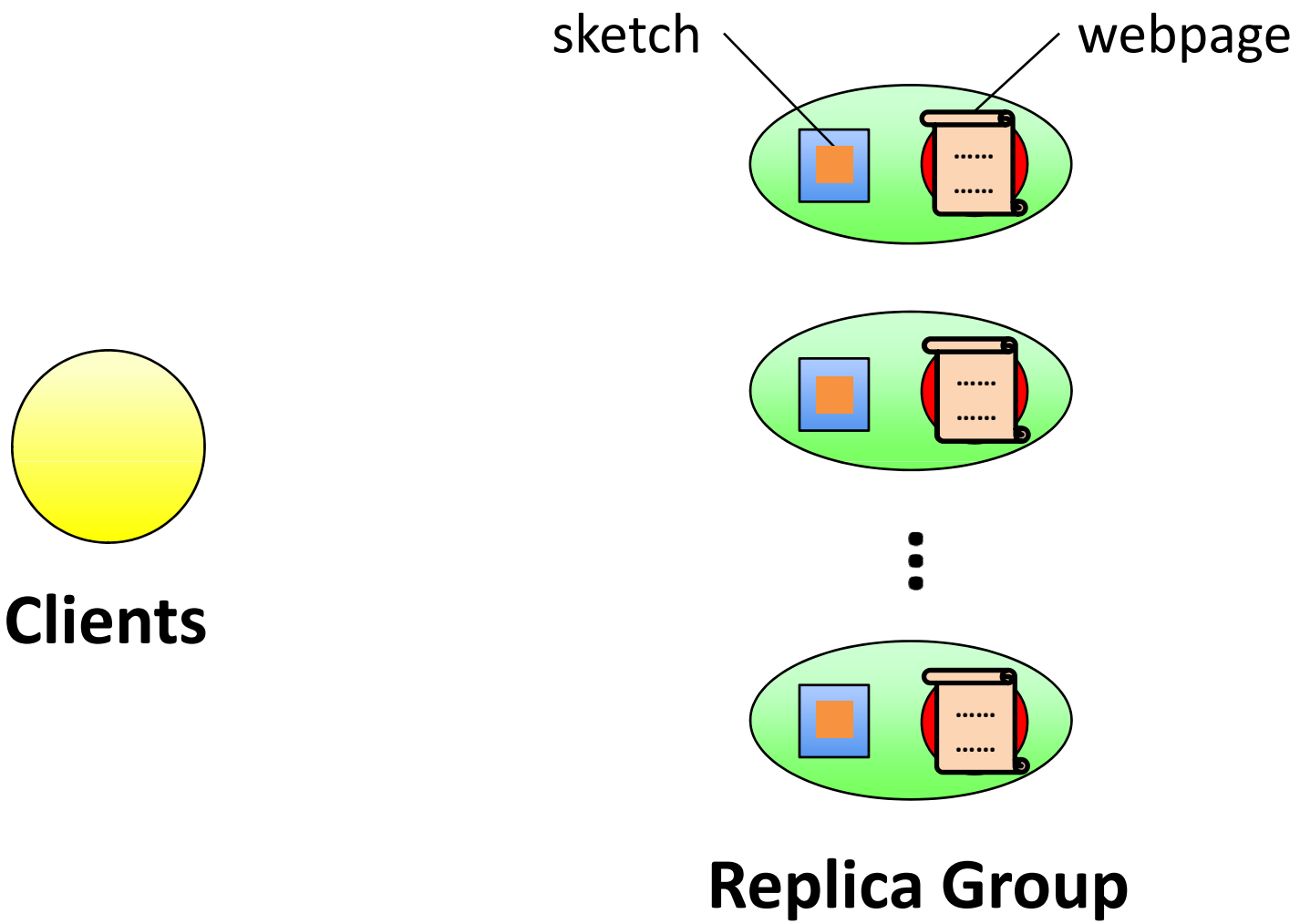
Executing a read



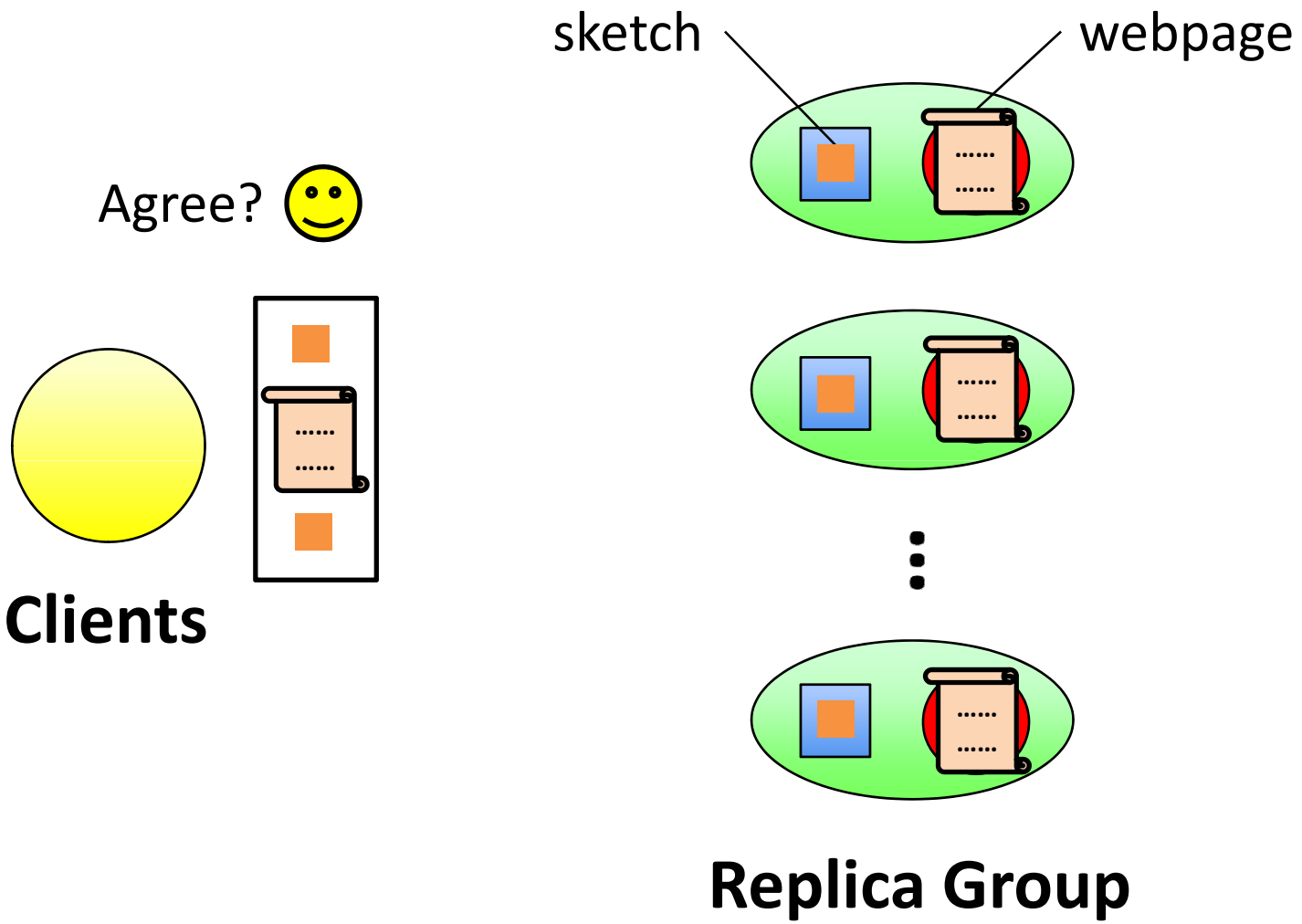
Executing a read



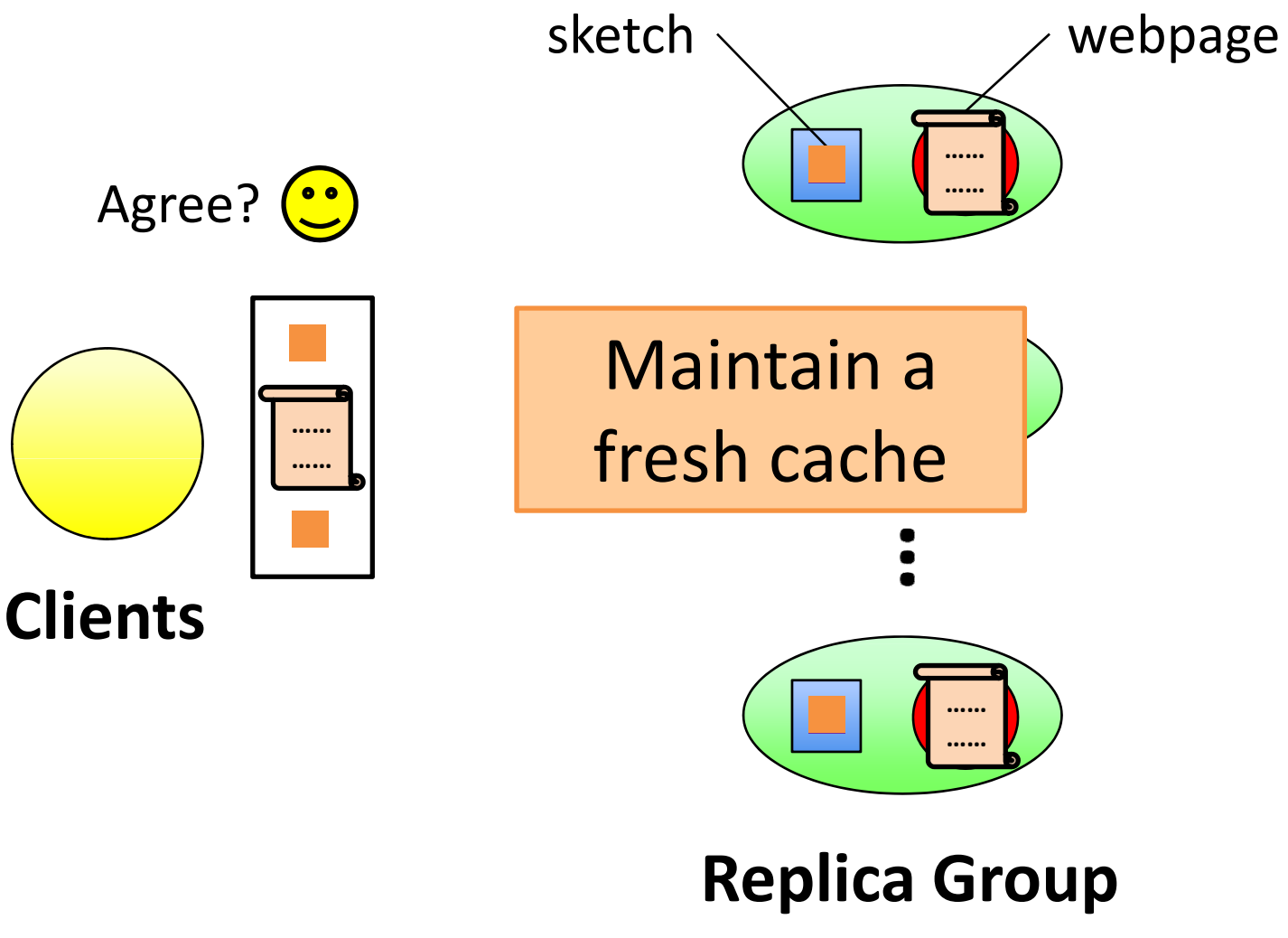
Executing a read



Executing a read



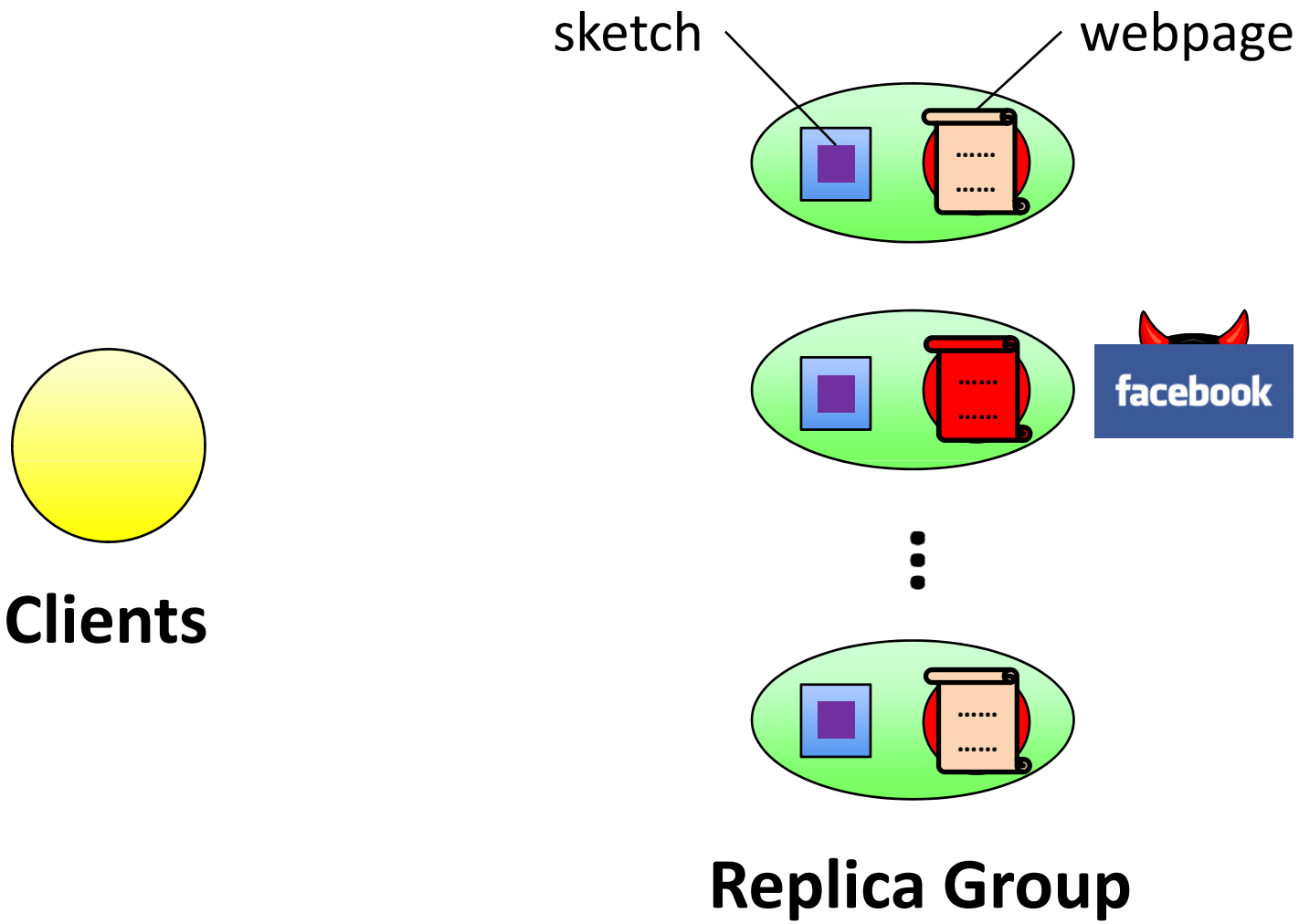
Executing a read



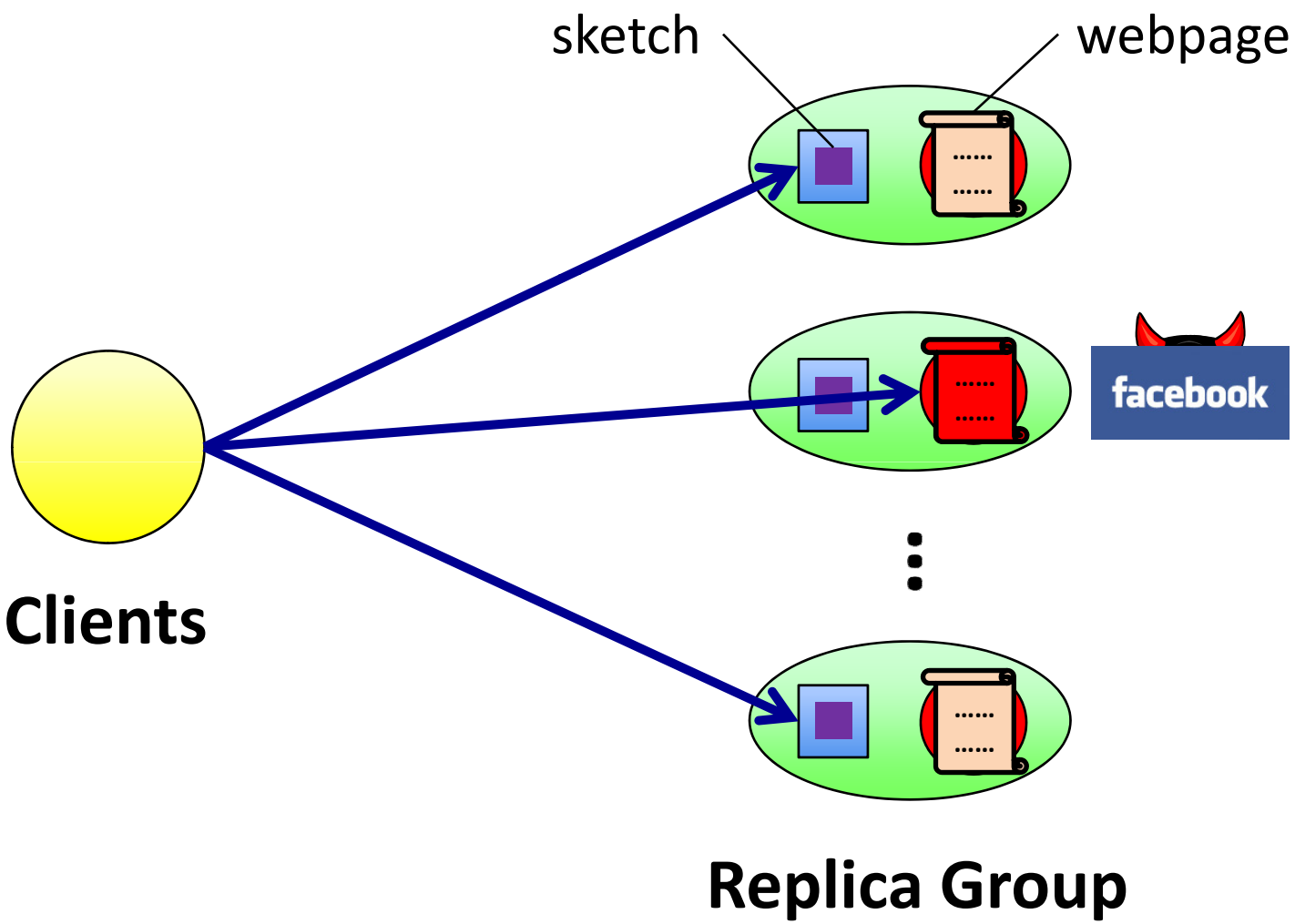
Did we achieve linearizability?

NO!

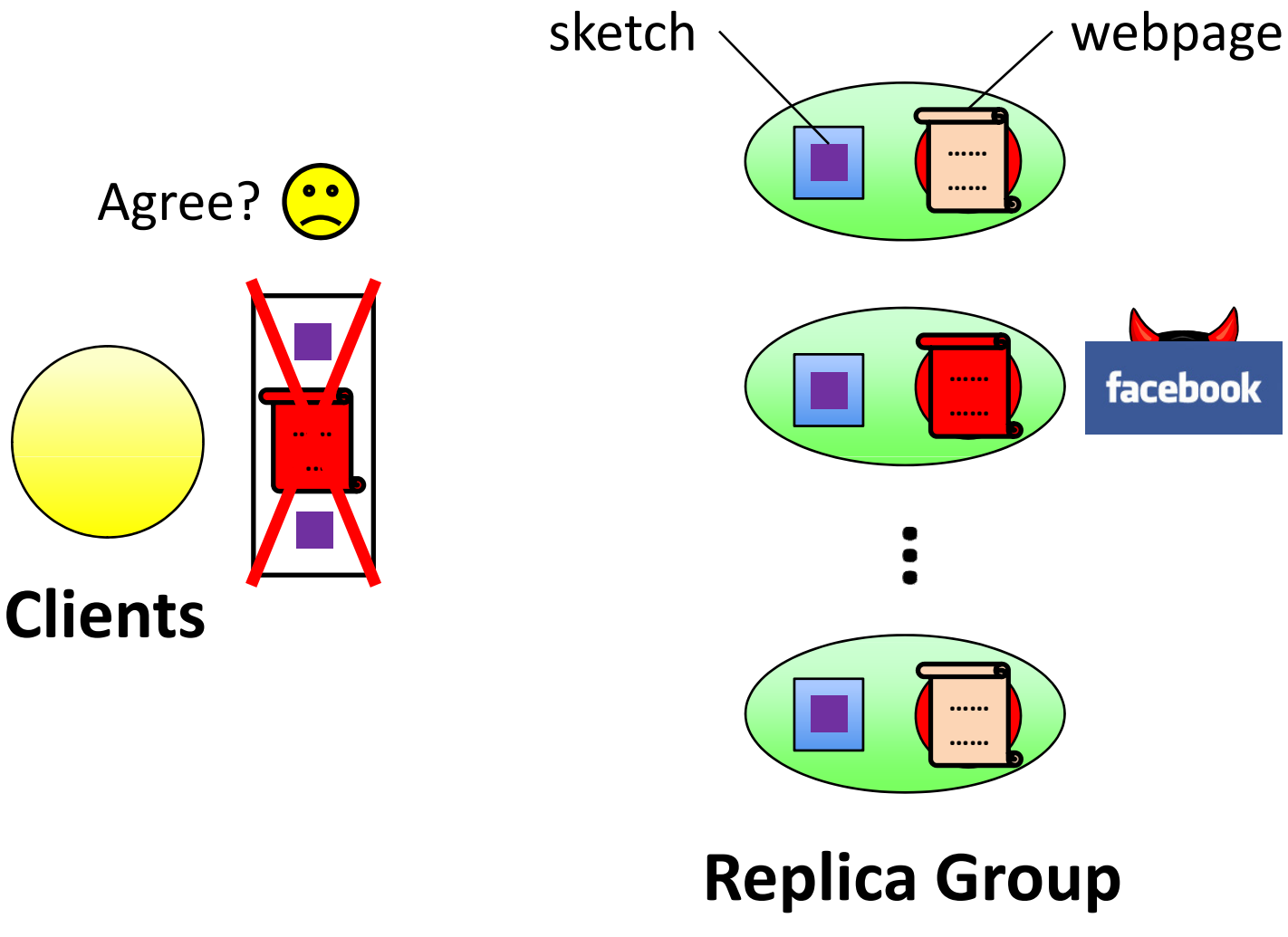
Executing a read



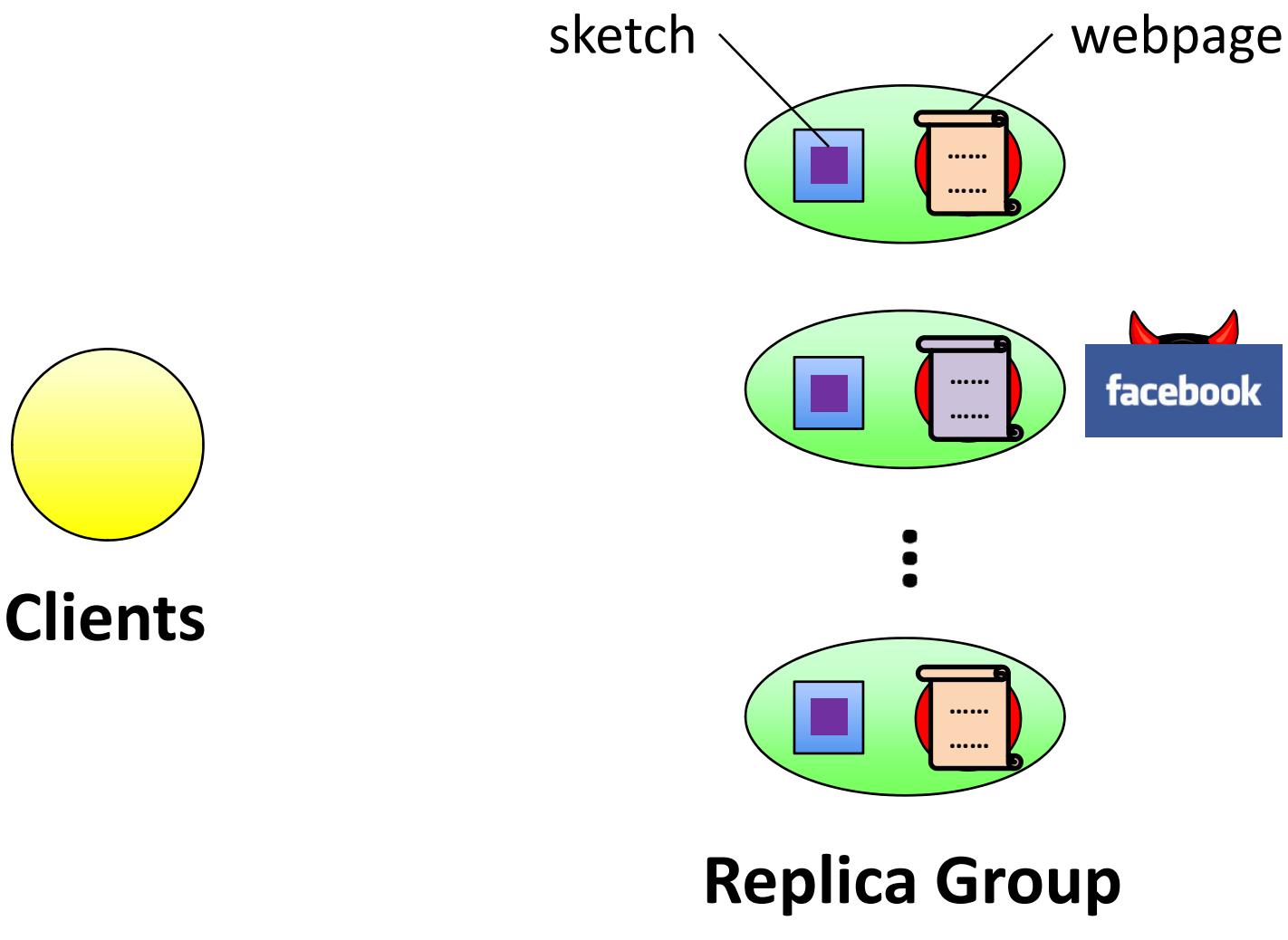
Executing a read



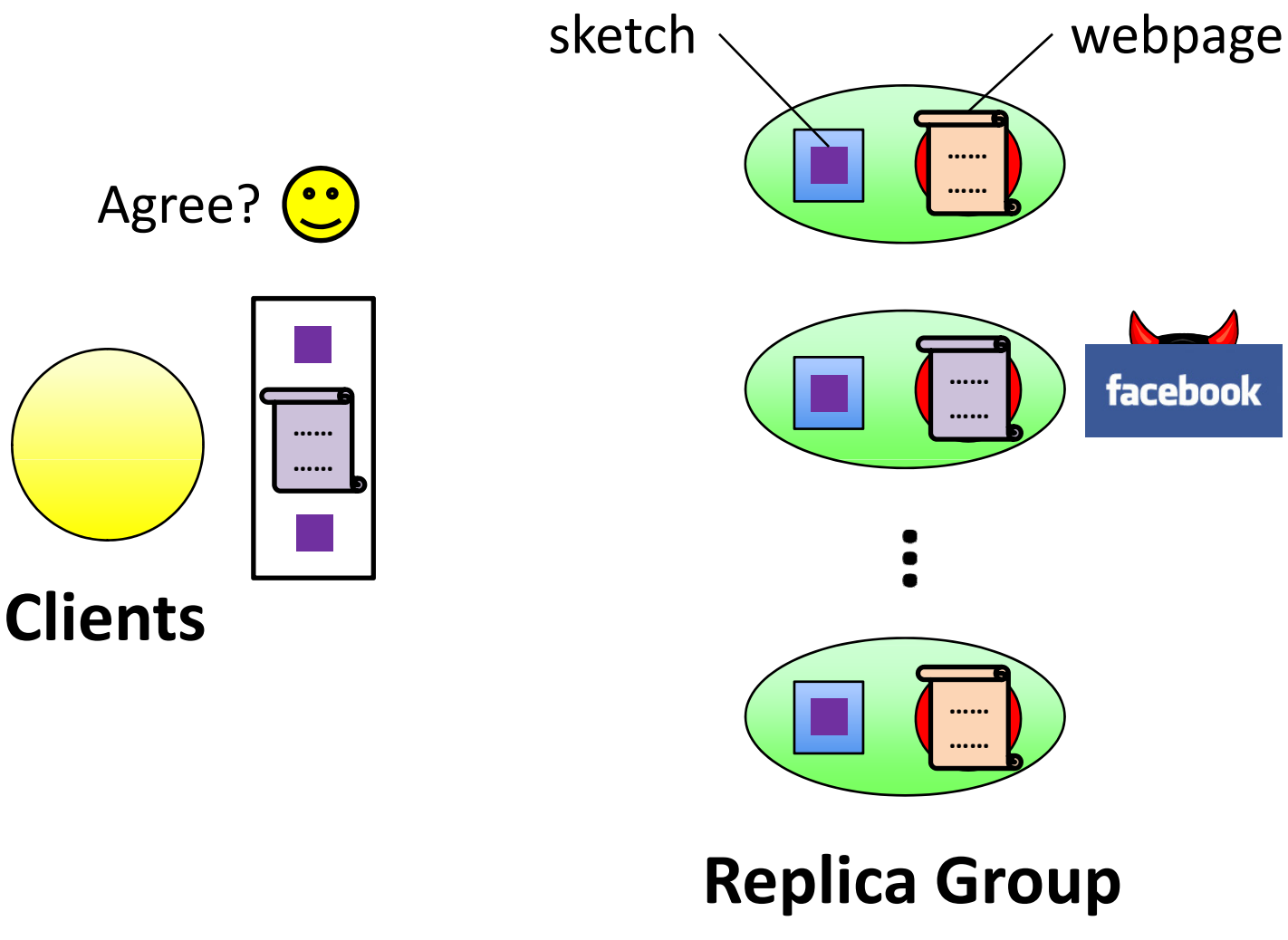
Executing a read



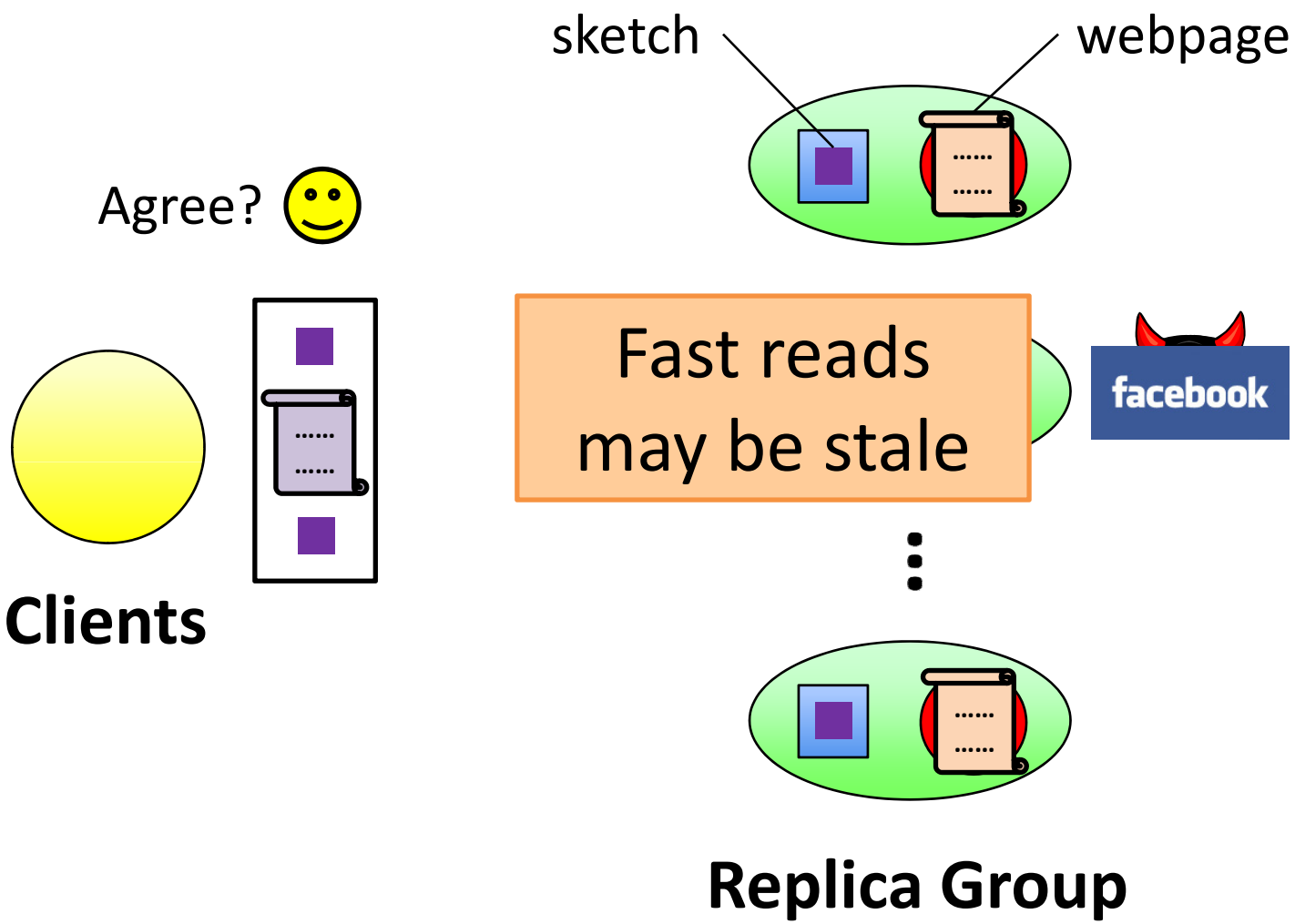
Executing a read



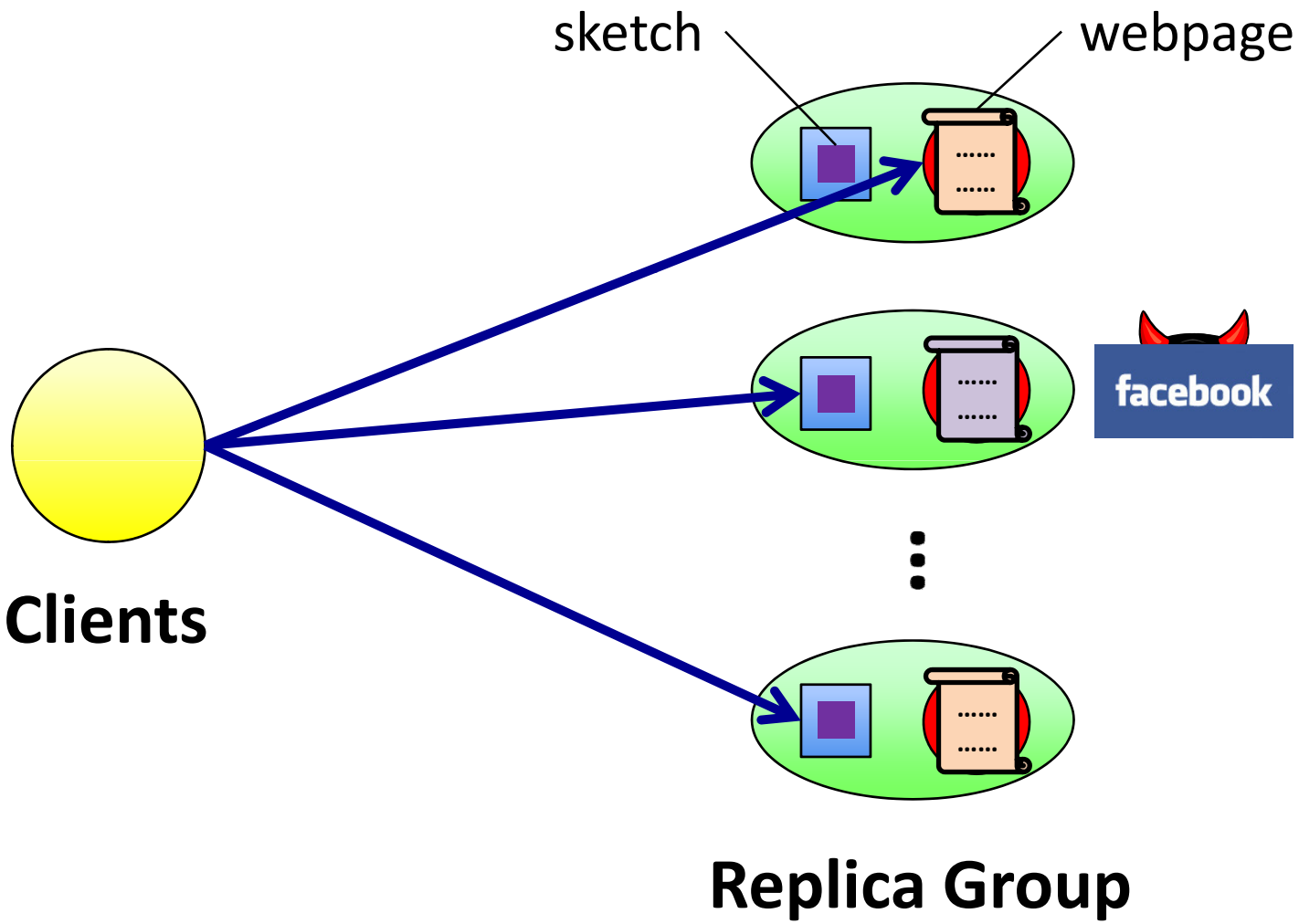
Executing a read



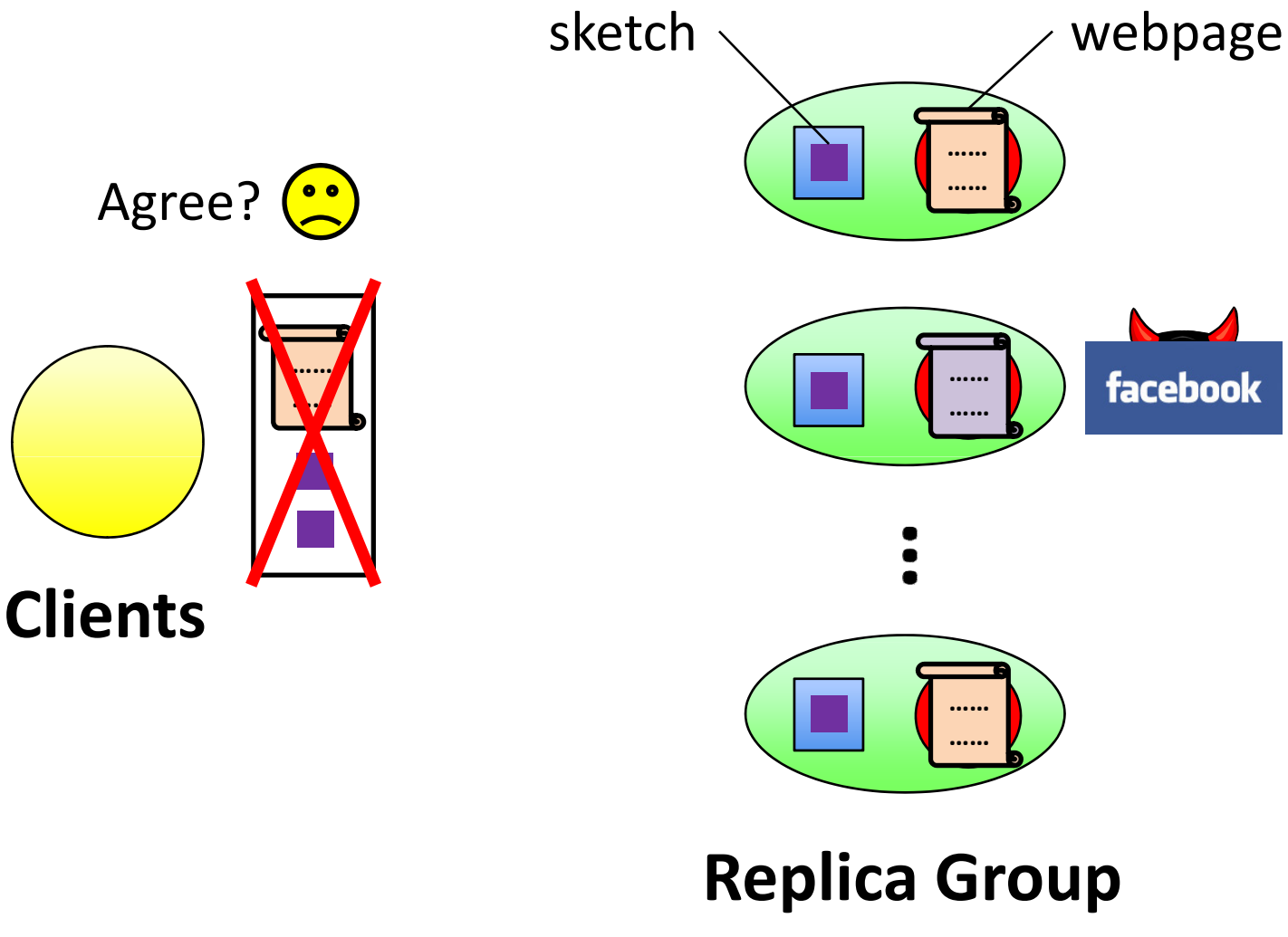
Executing a read



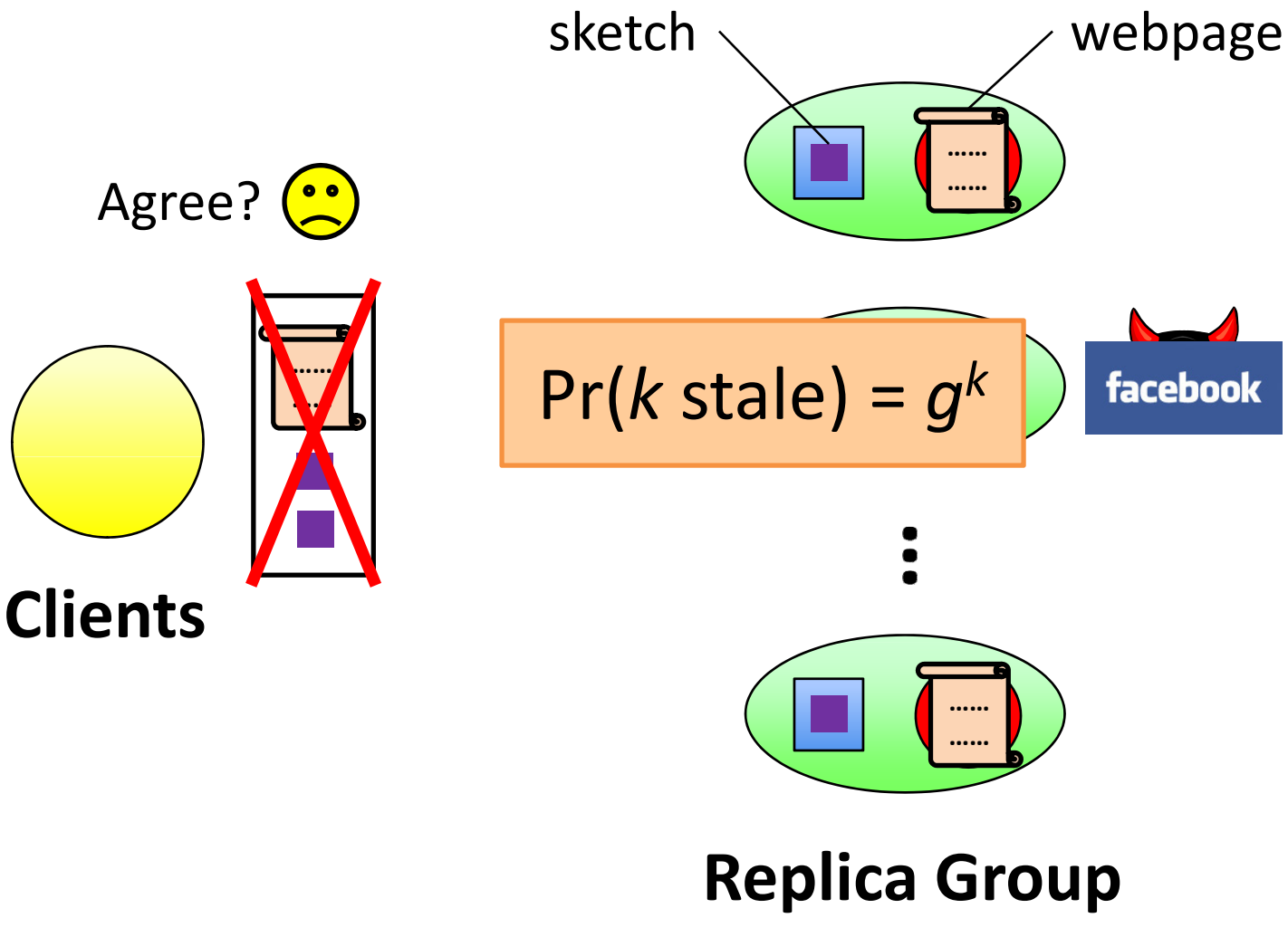
Load balancing



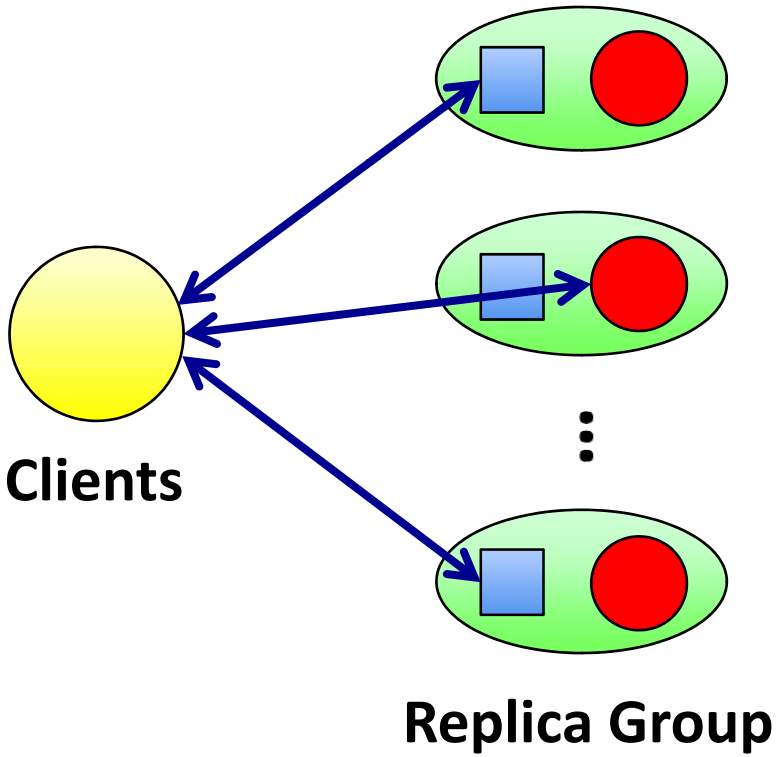
Load balancing



Load balancing



D-Prophecy vs. BFT



Traditional BFT:

- Each replica executes read
- Linearizability

D-Prophecy:

- One replica executes read
- “Delay-once” linearizability

Byzantine fault tolerance (BFT)

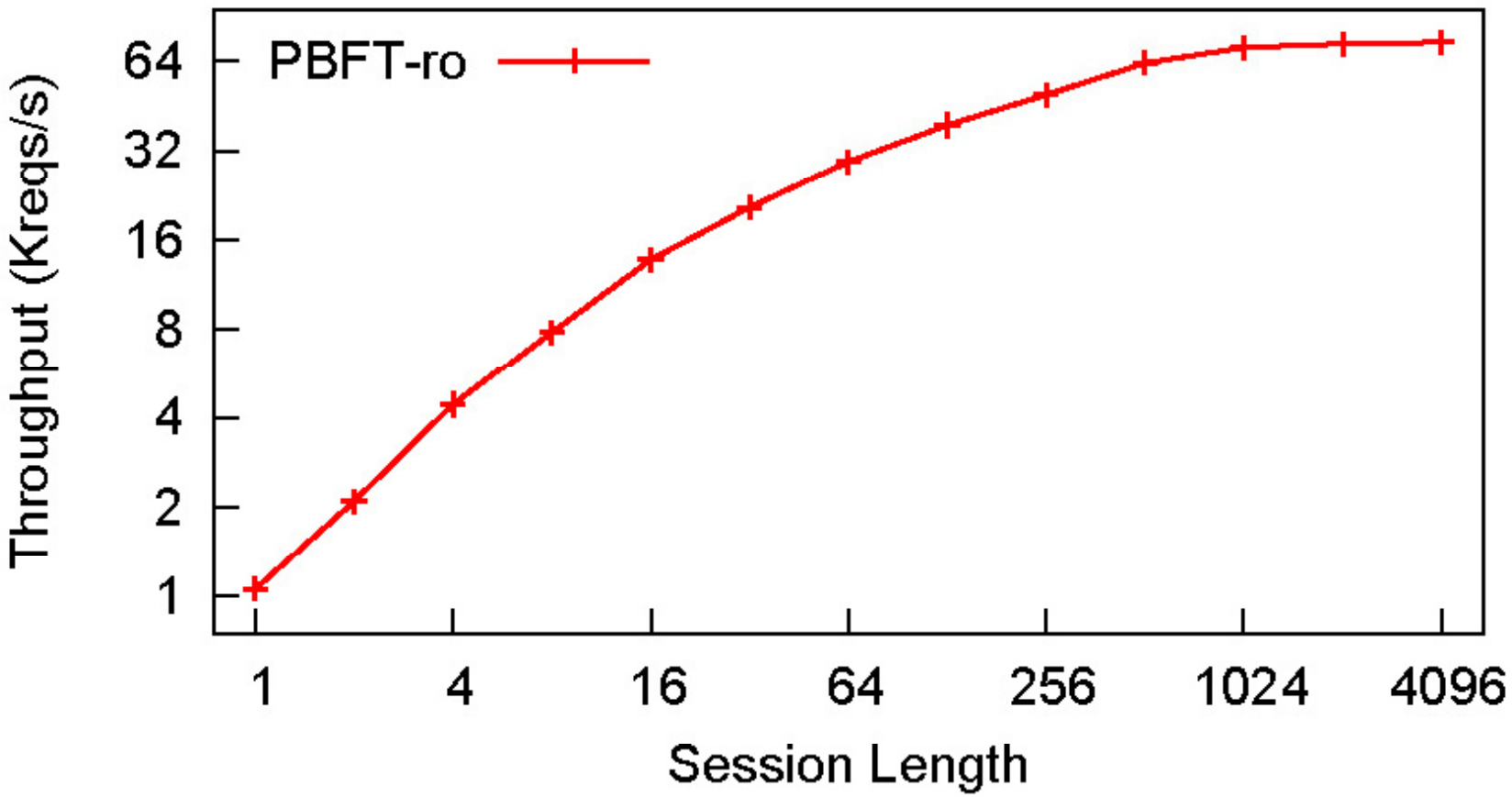
- ~~Low throughput~~

D-Prophecy

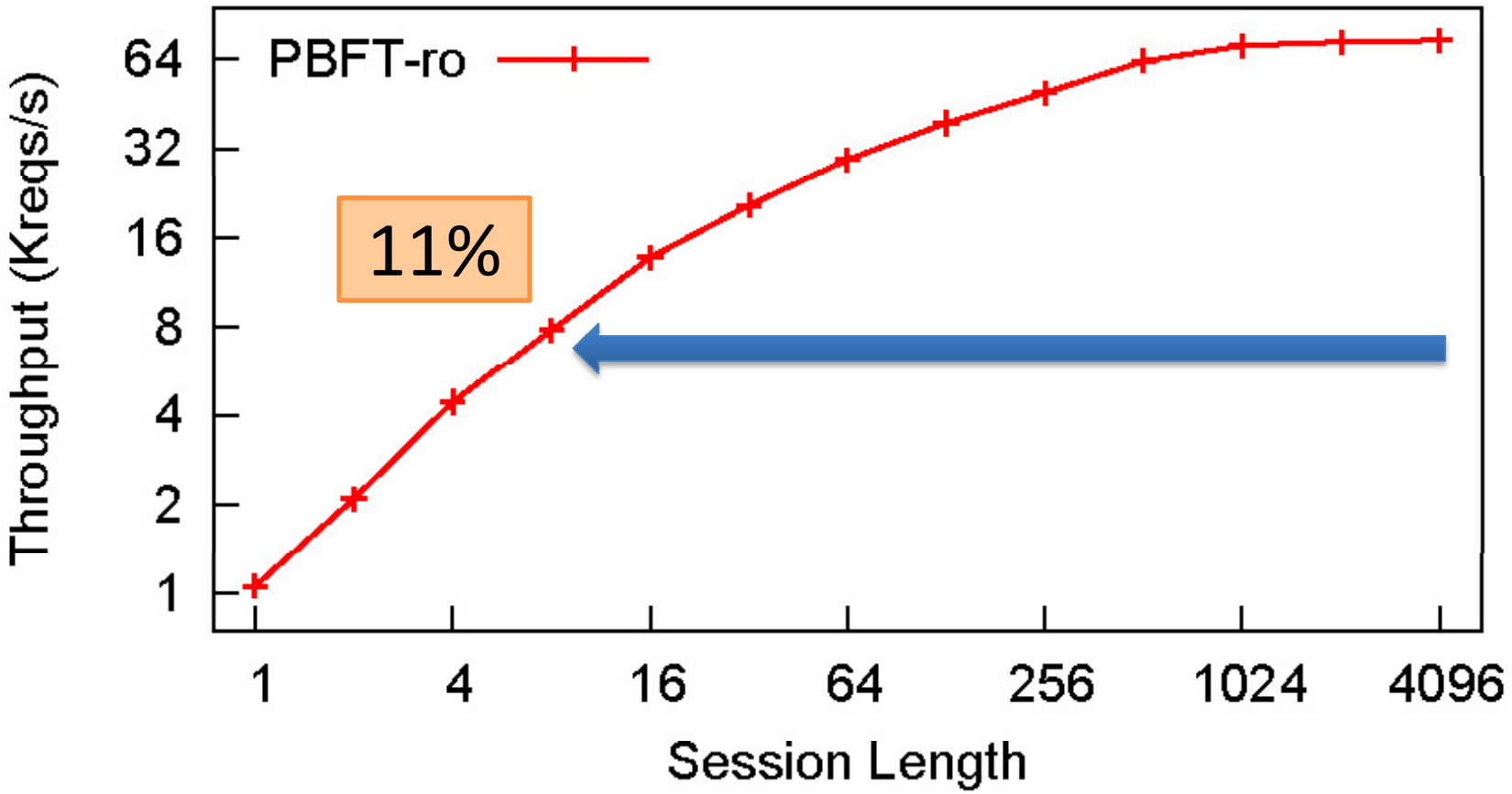
- Modifies clients
- Long-lived sessions

Prophecy

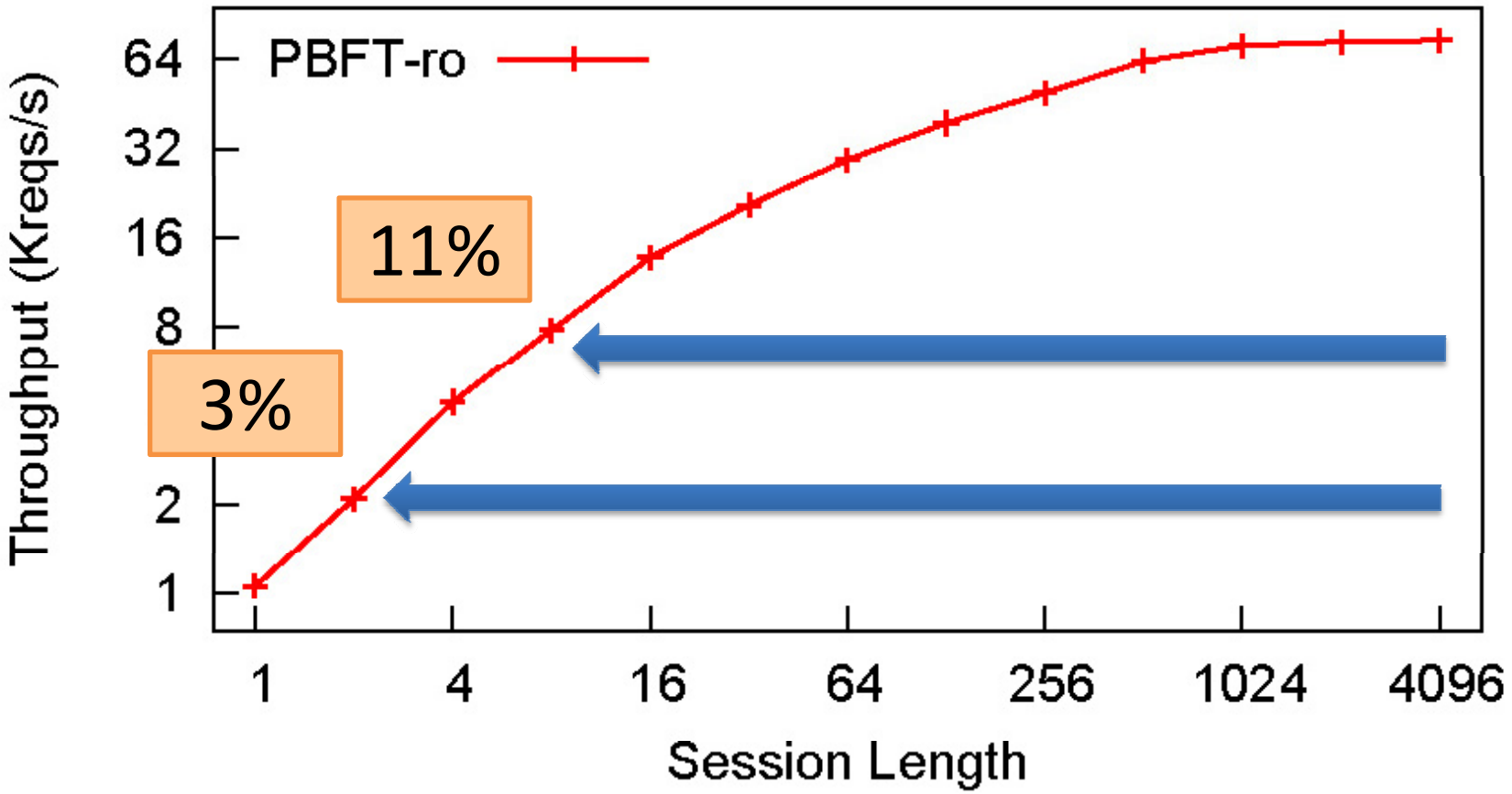
Key-exchange overhead



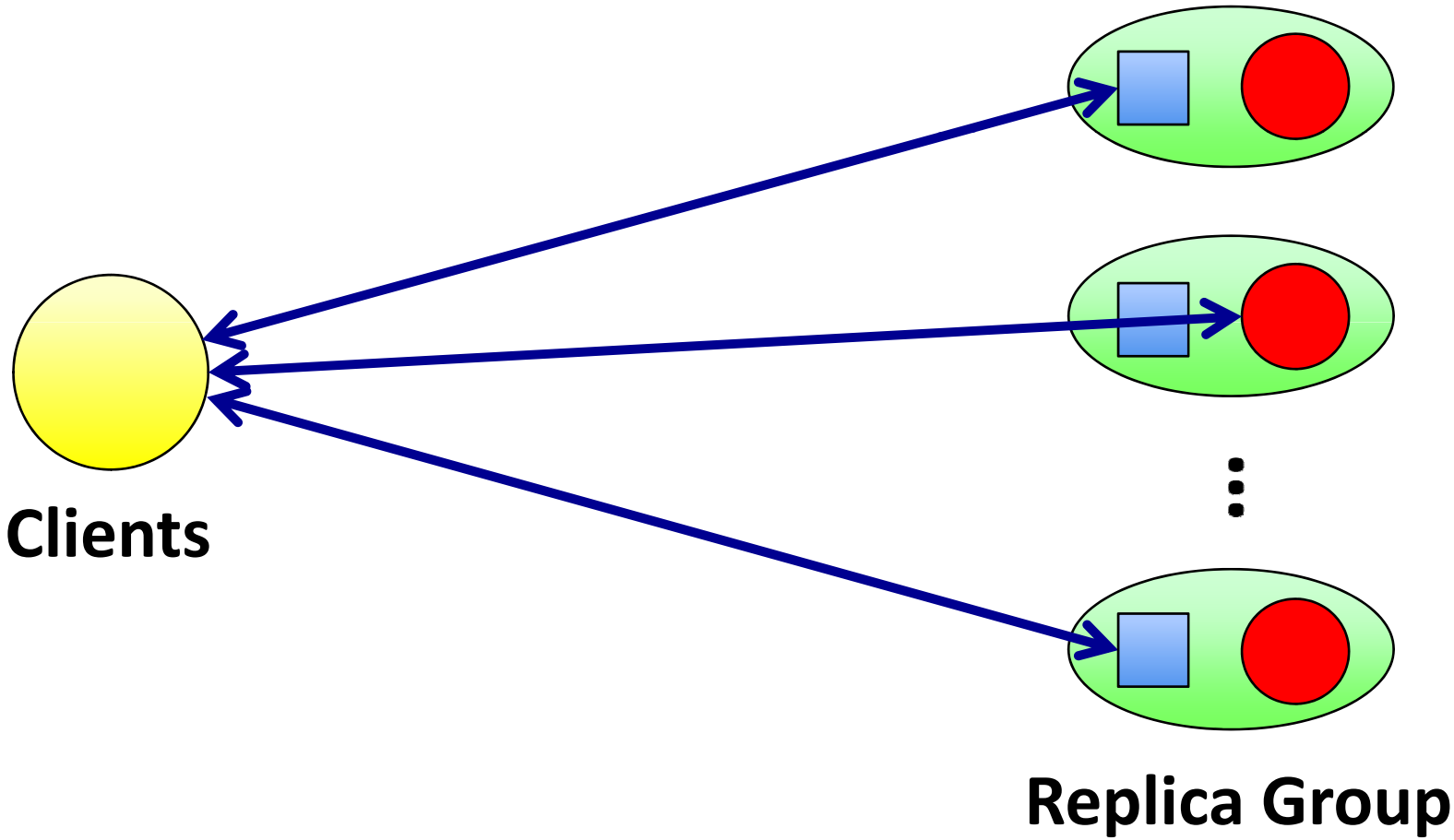
Key-exchange overhead



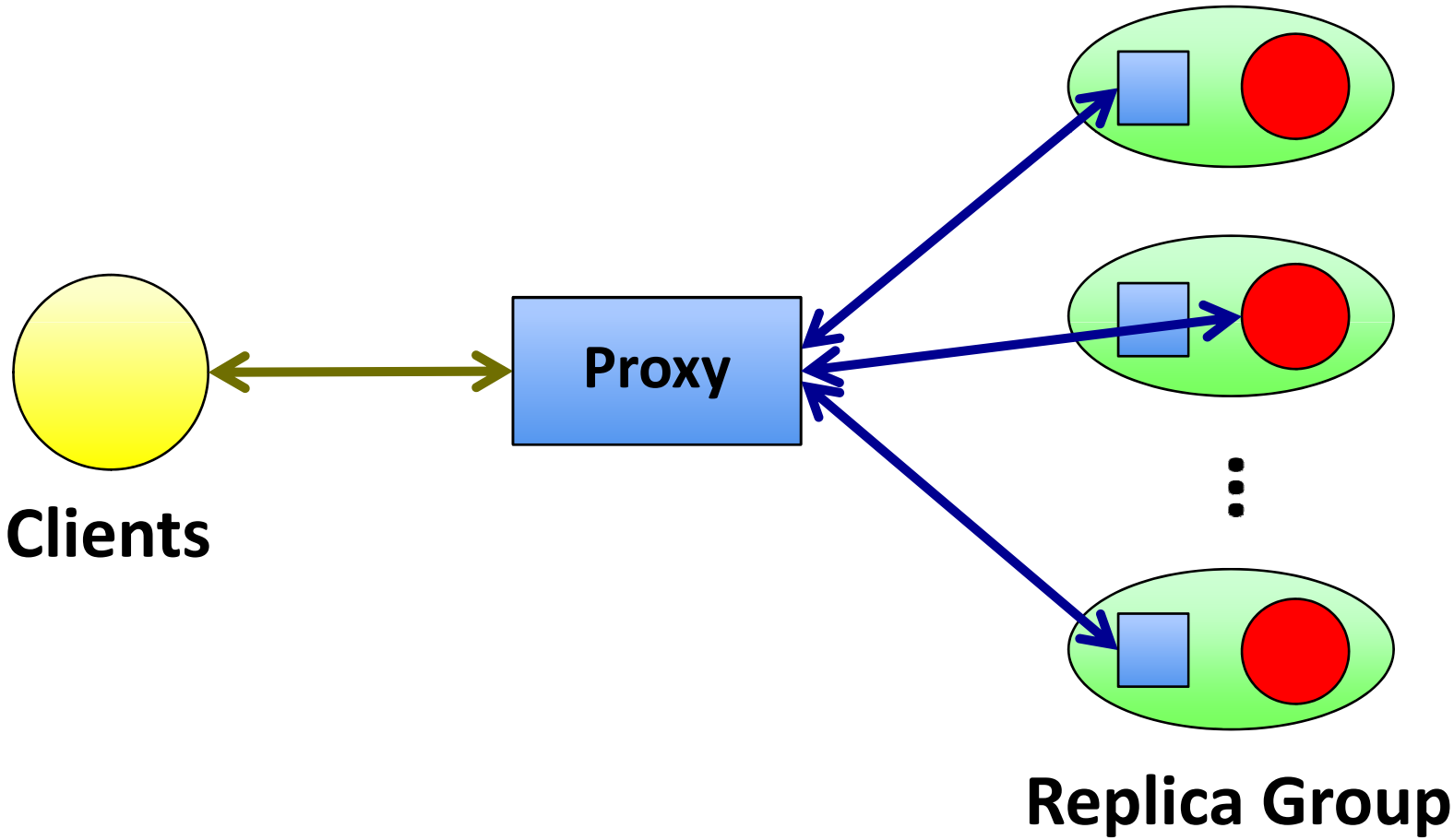
Key-exchange overhead



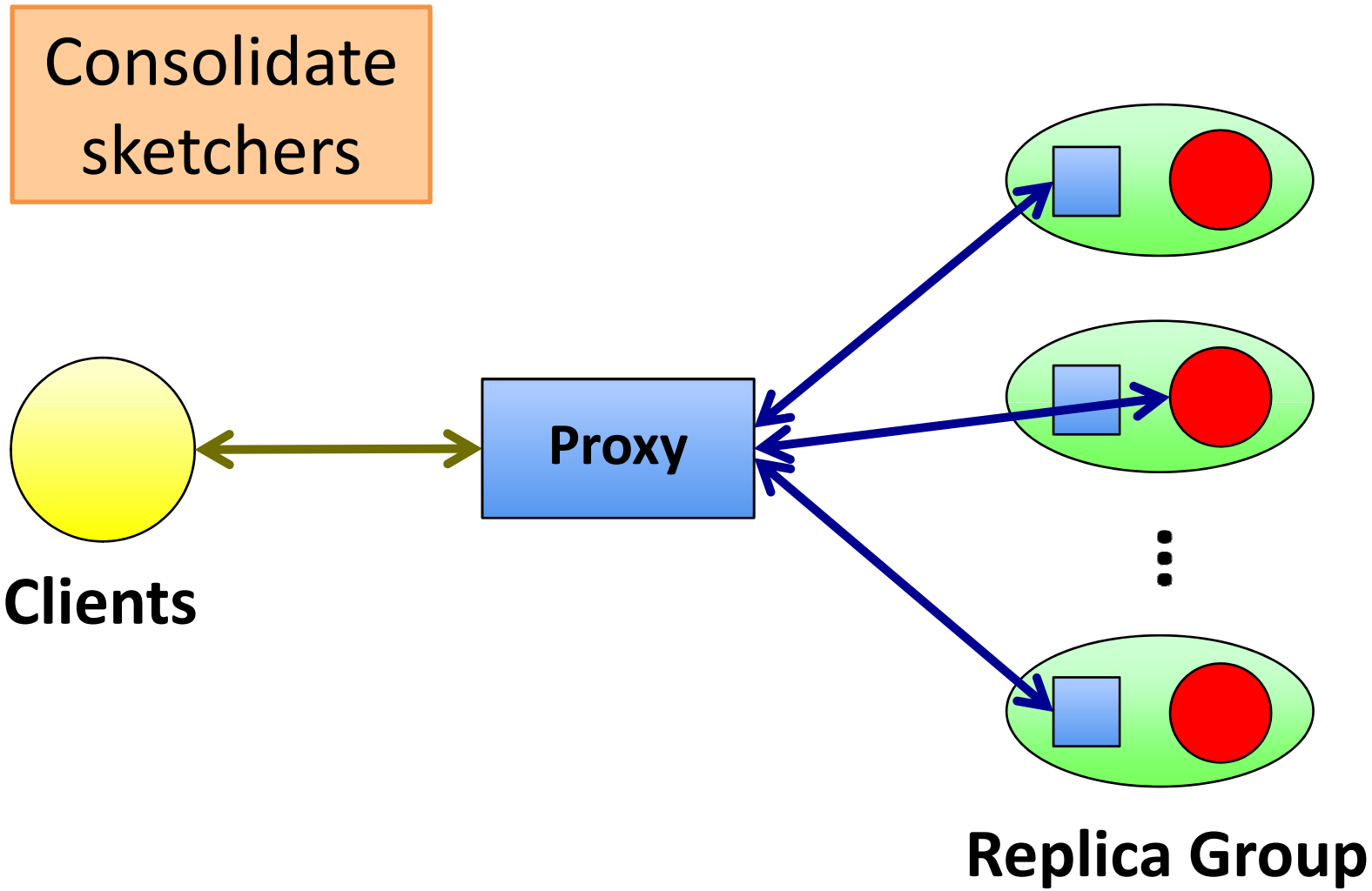
Internet services



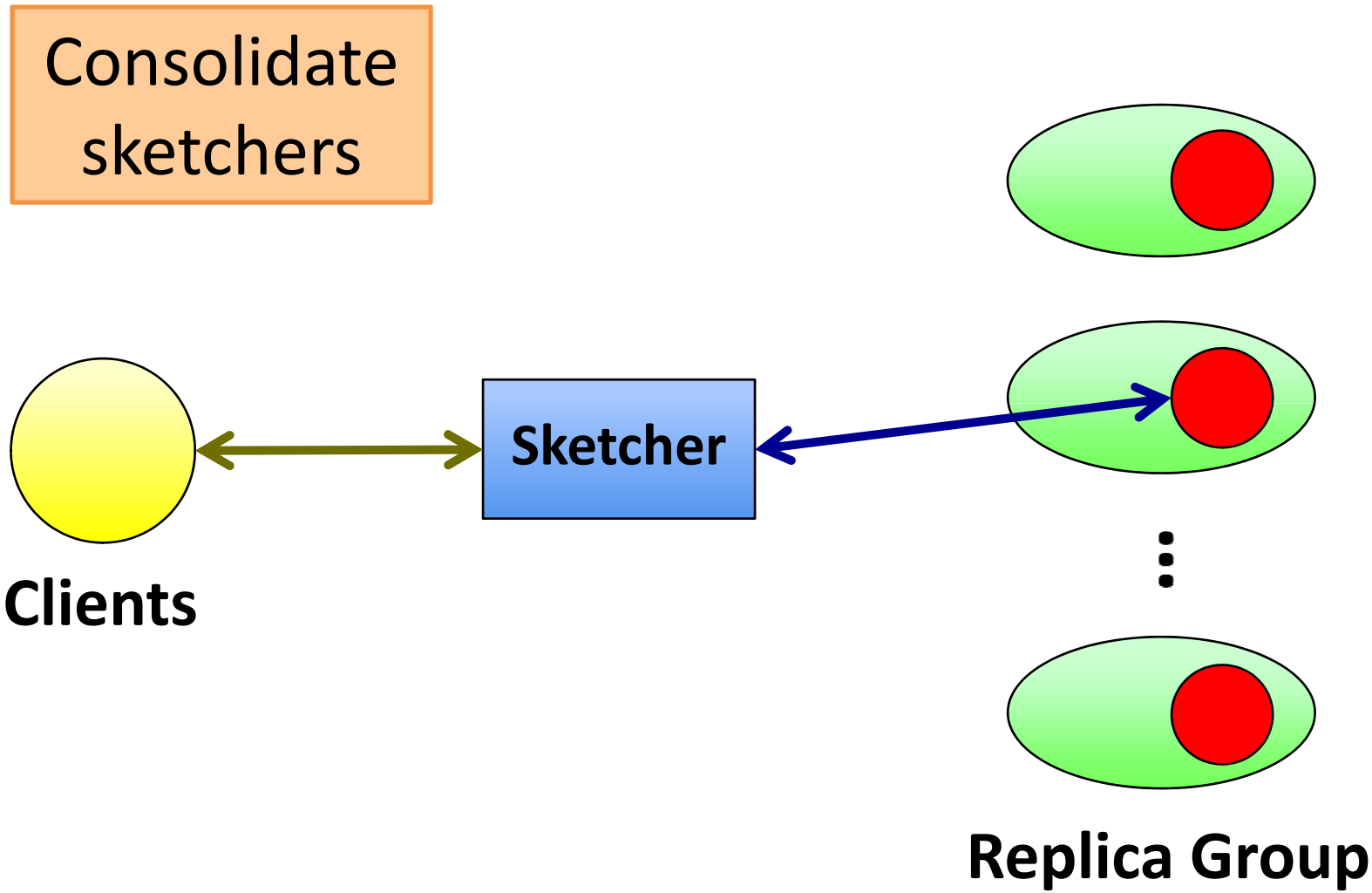
A proxy solution



A proxy solution

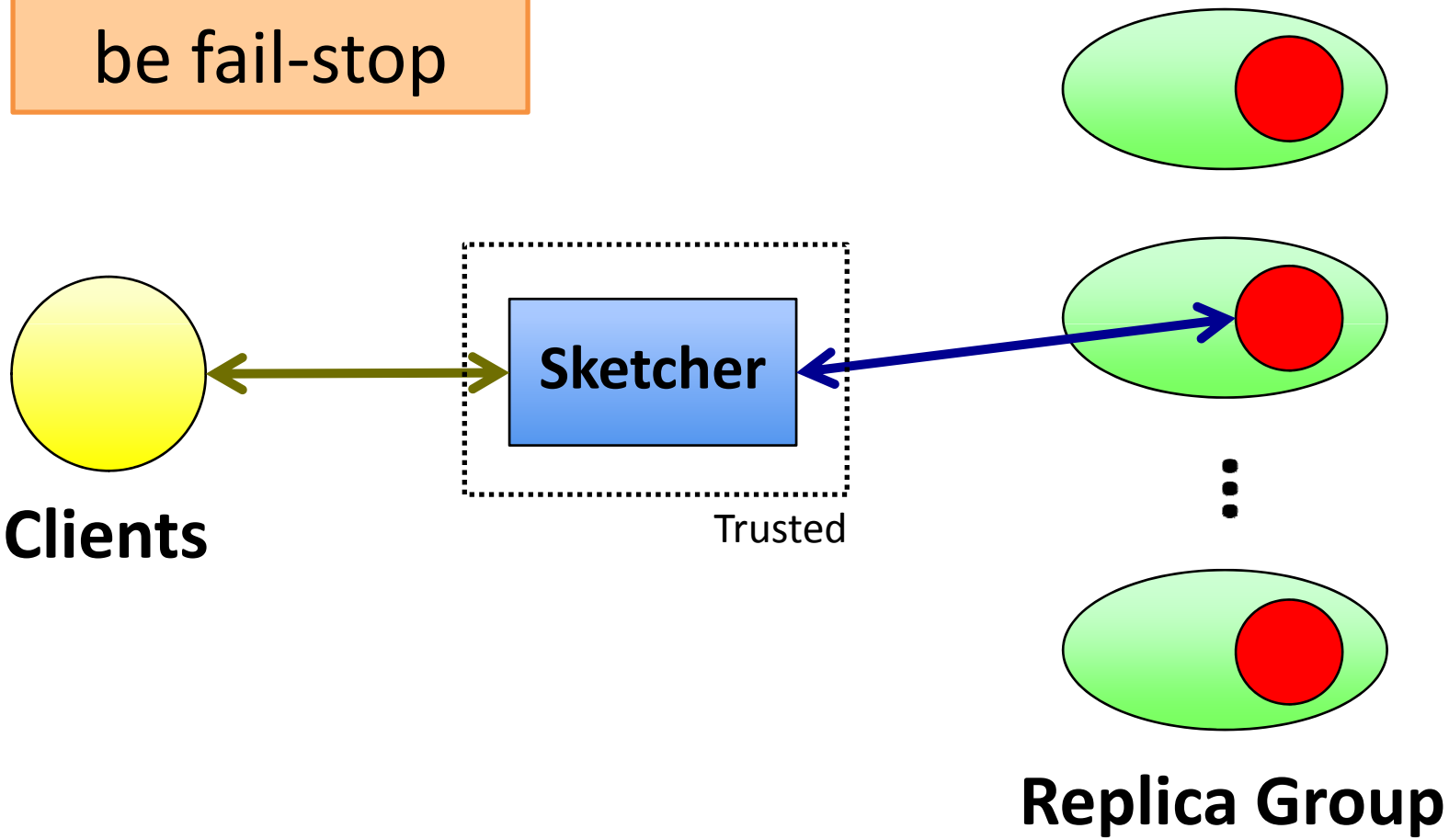


A proxy solution

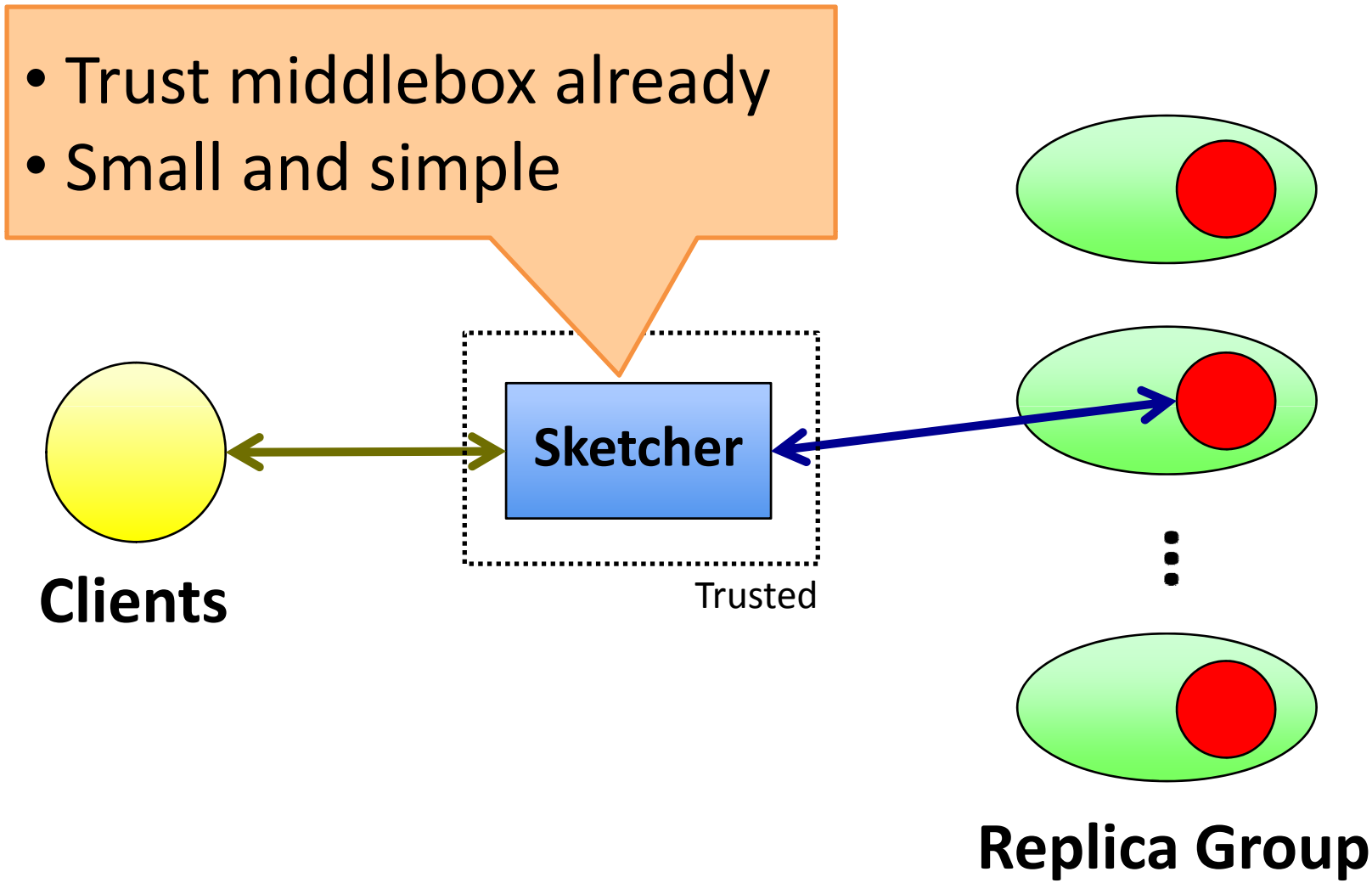


A proxy solution

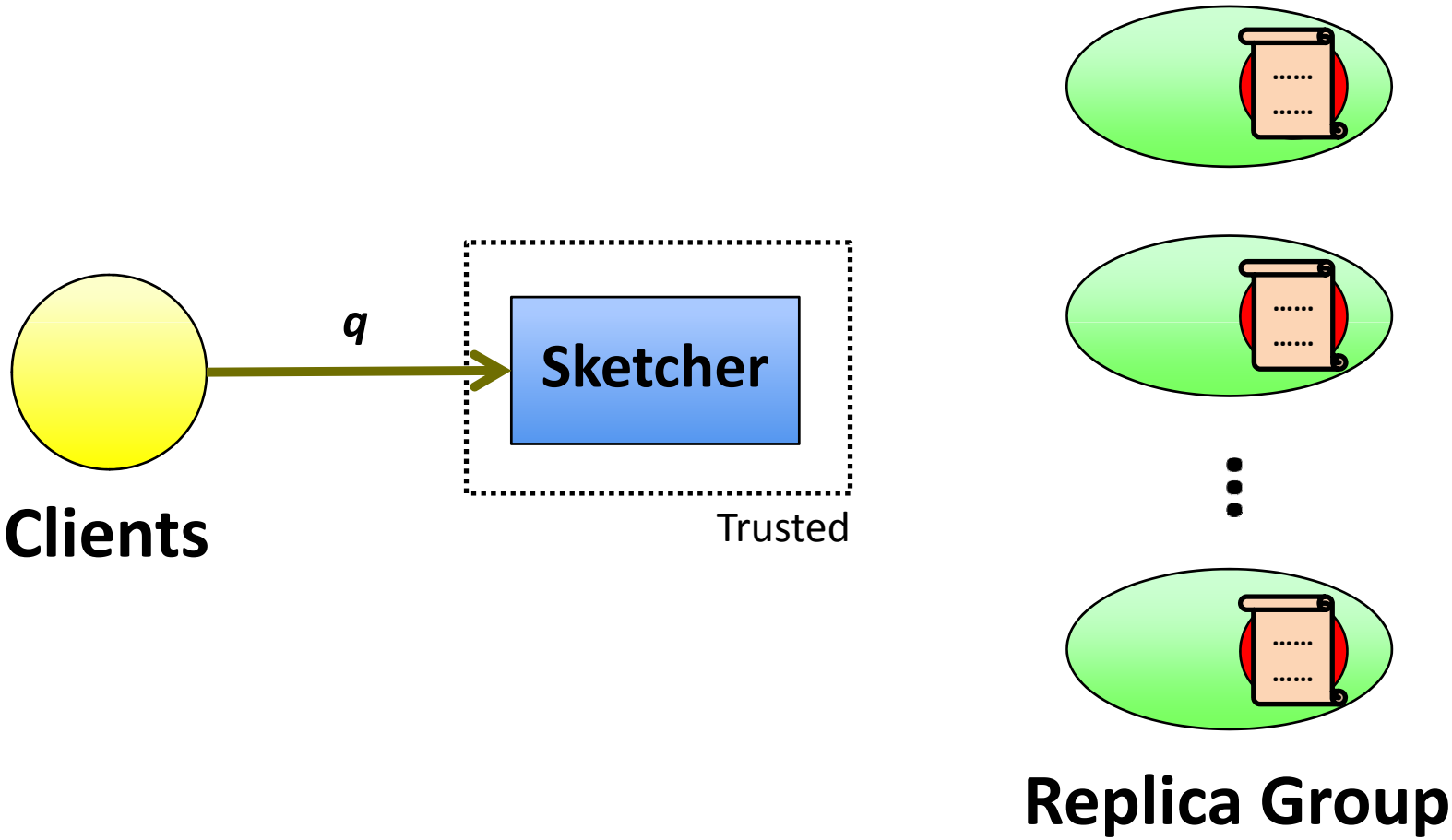
Sketcher must be fail-stop



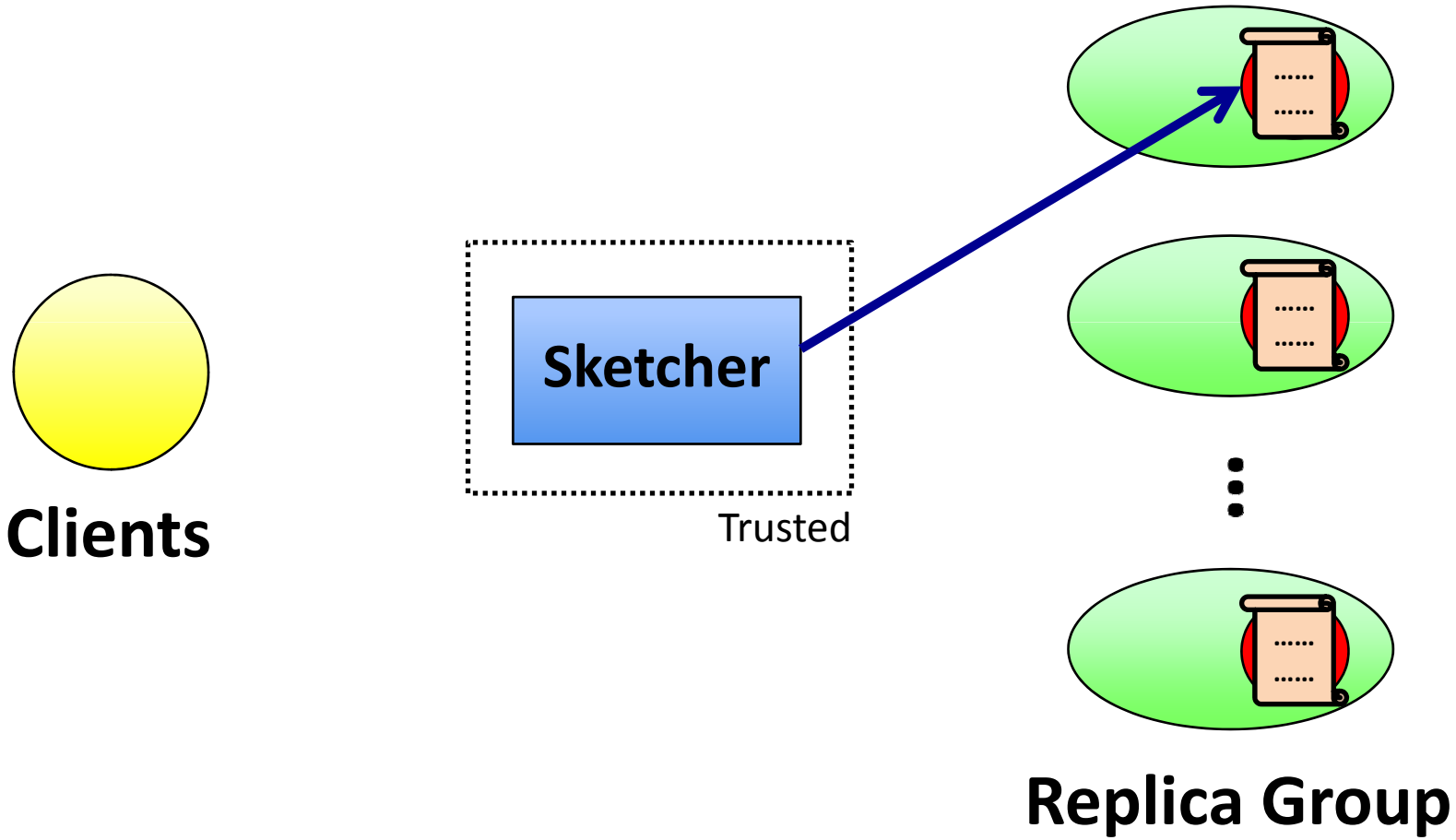
A proxy solution



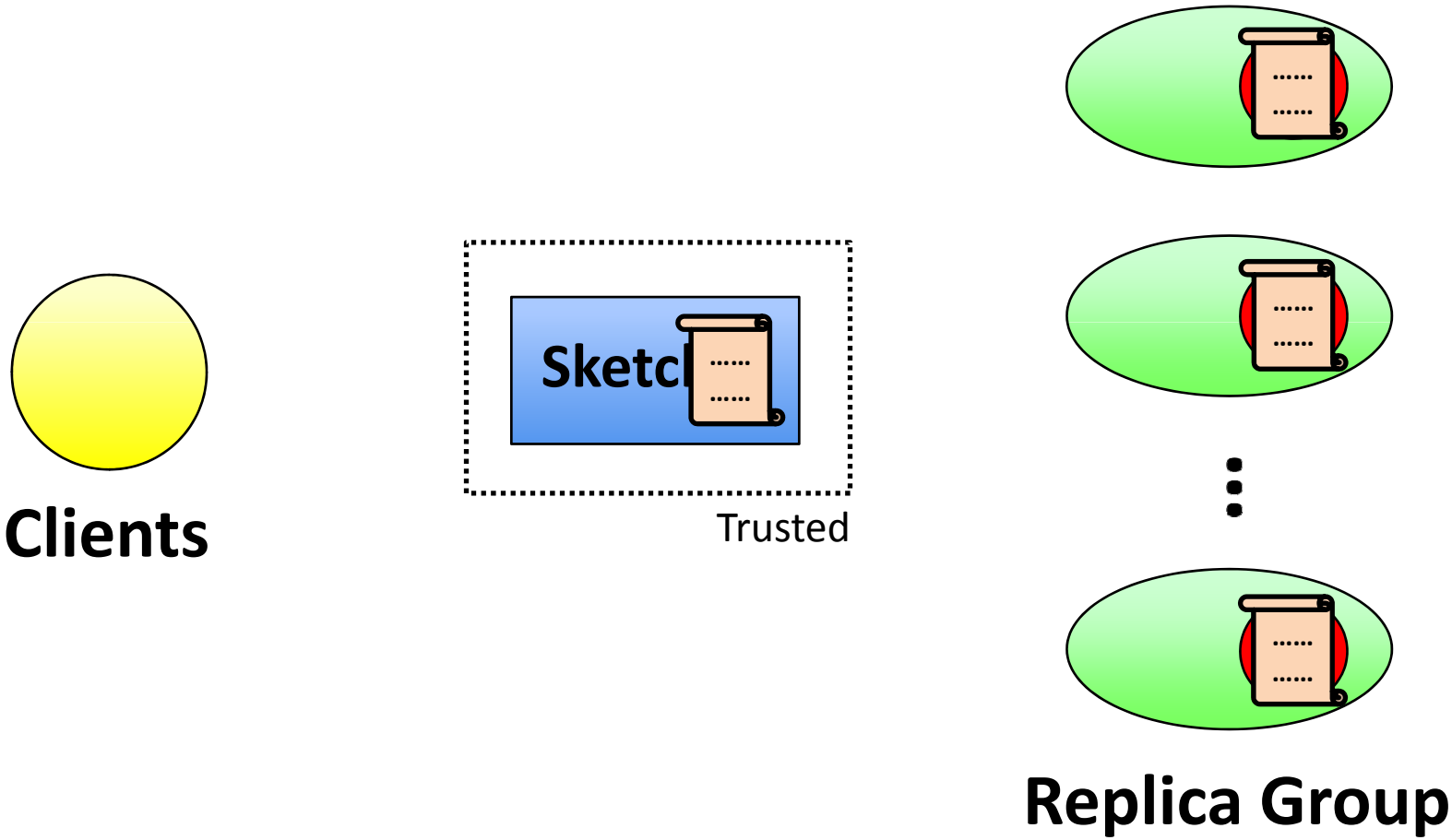
Executing a read



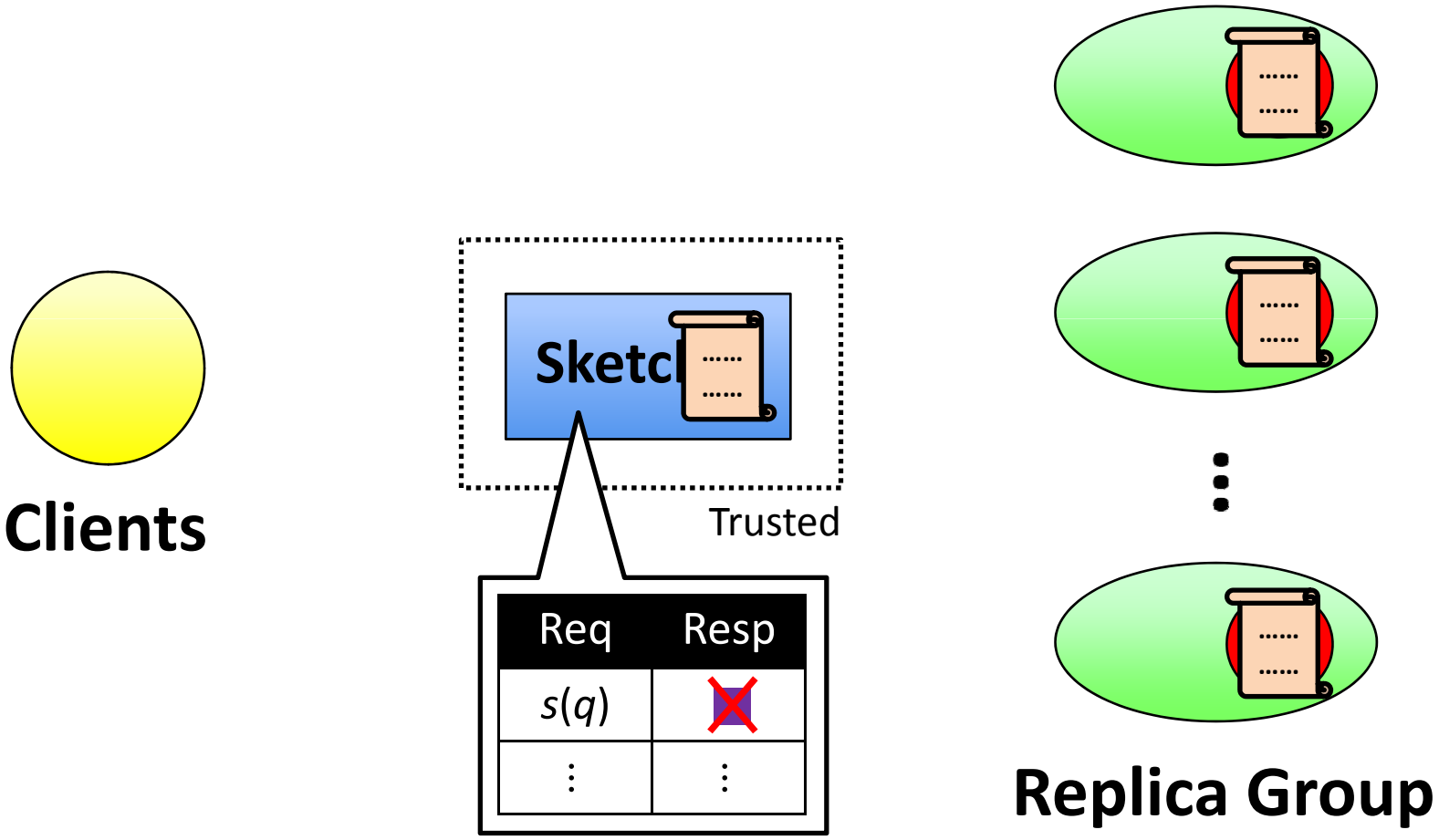
Executing a read



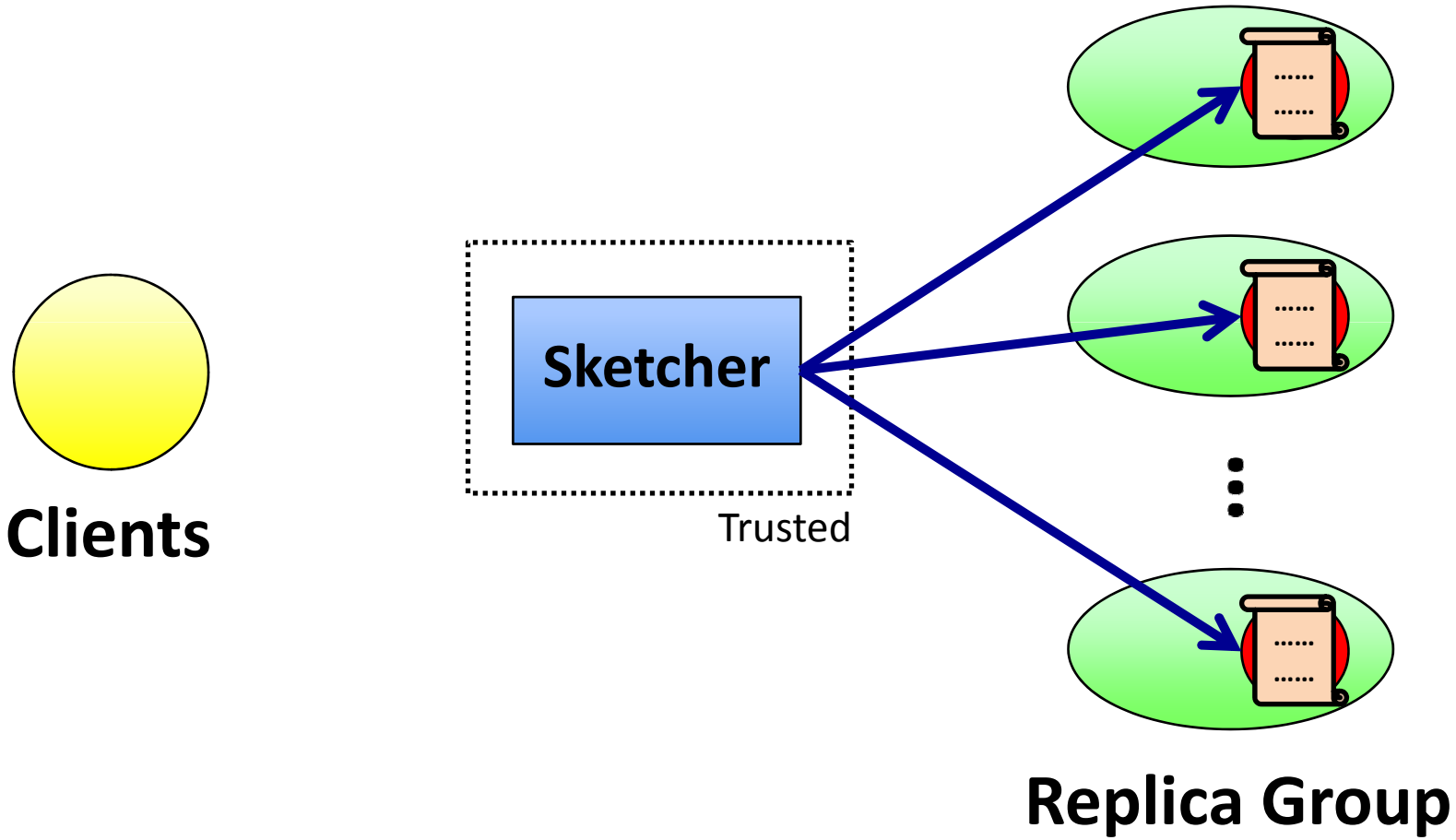
Executing a read



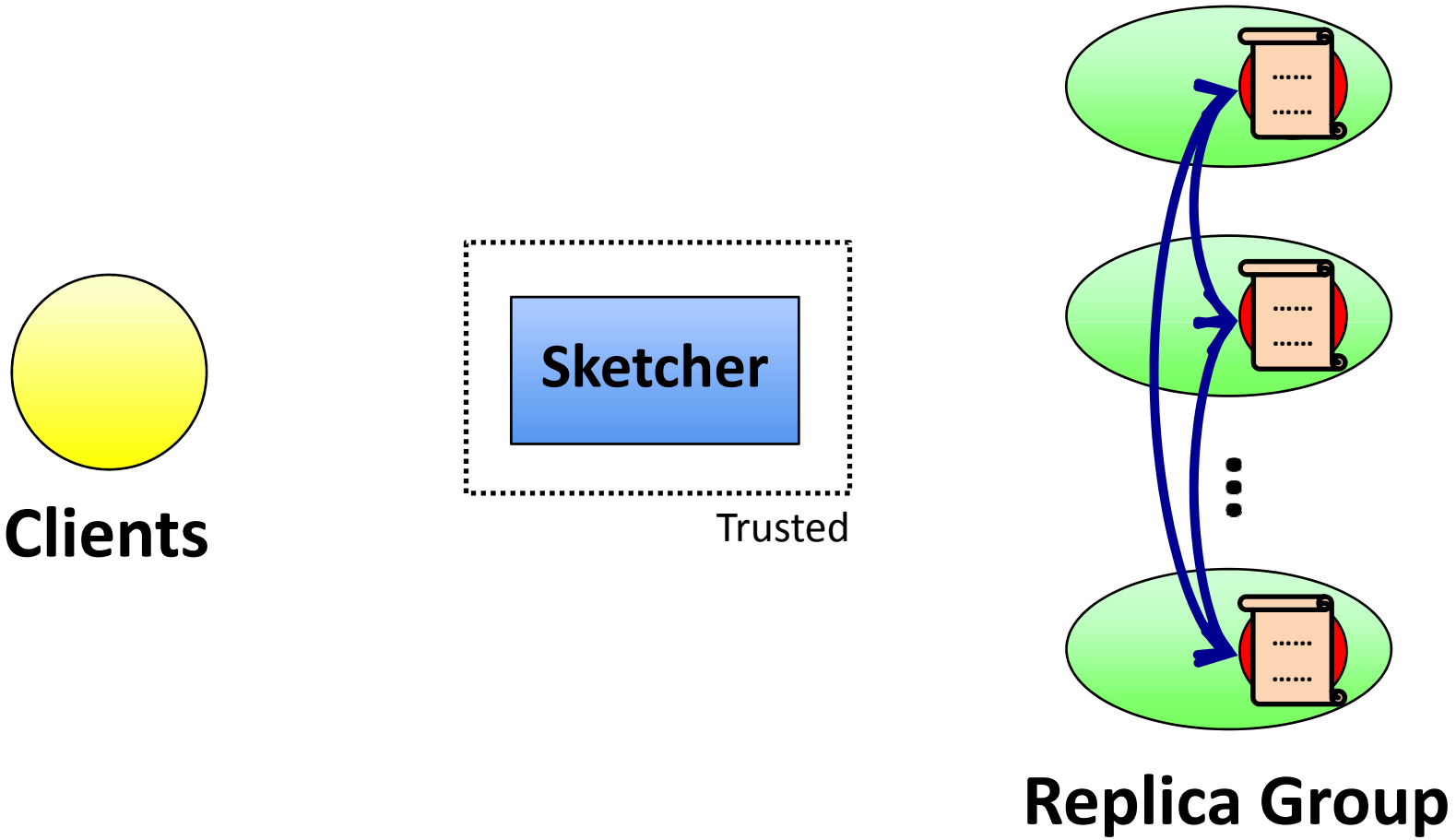
Executing a read



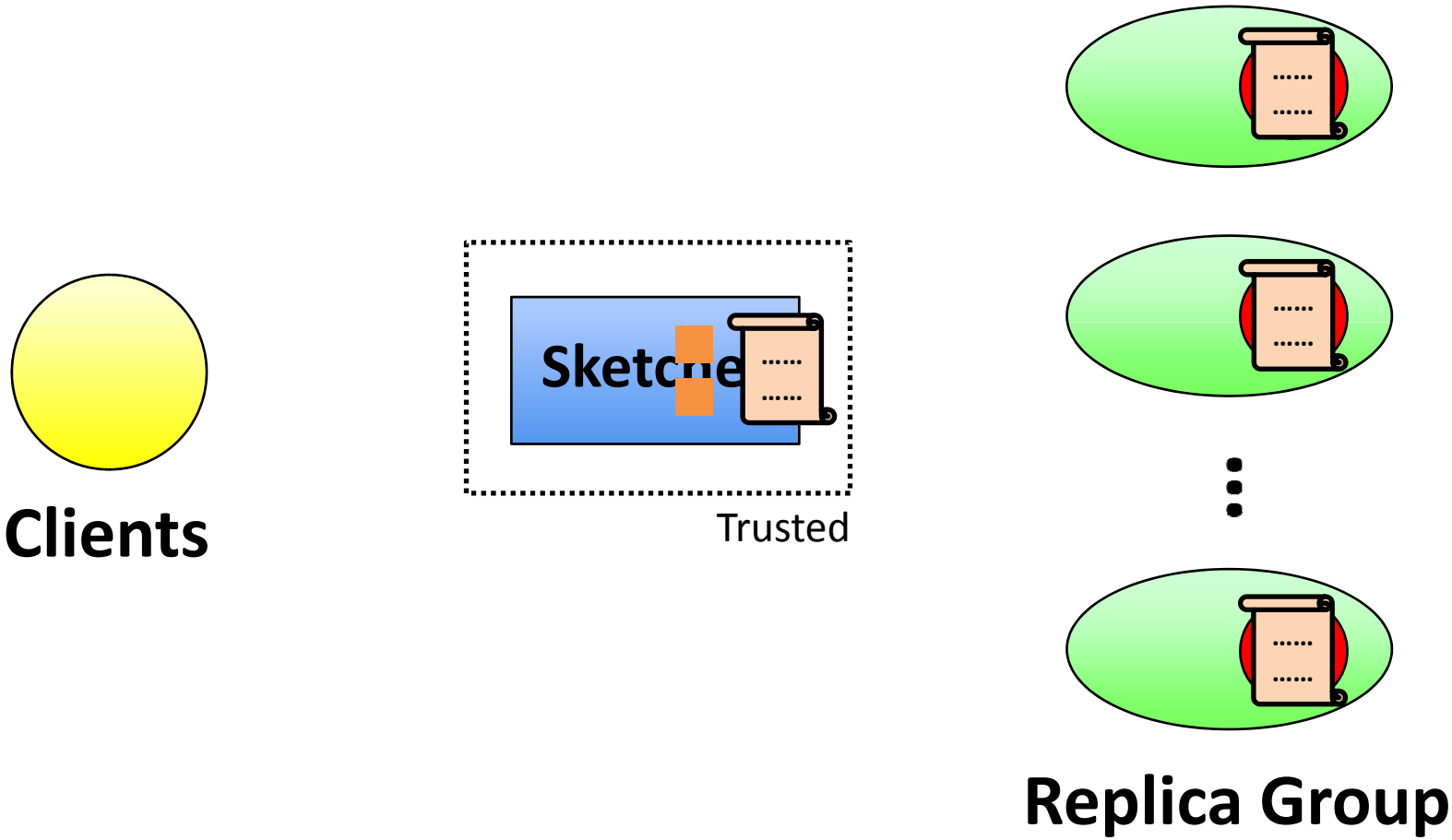
Executing a read



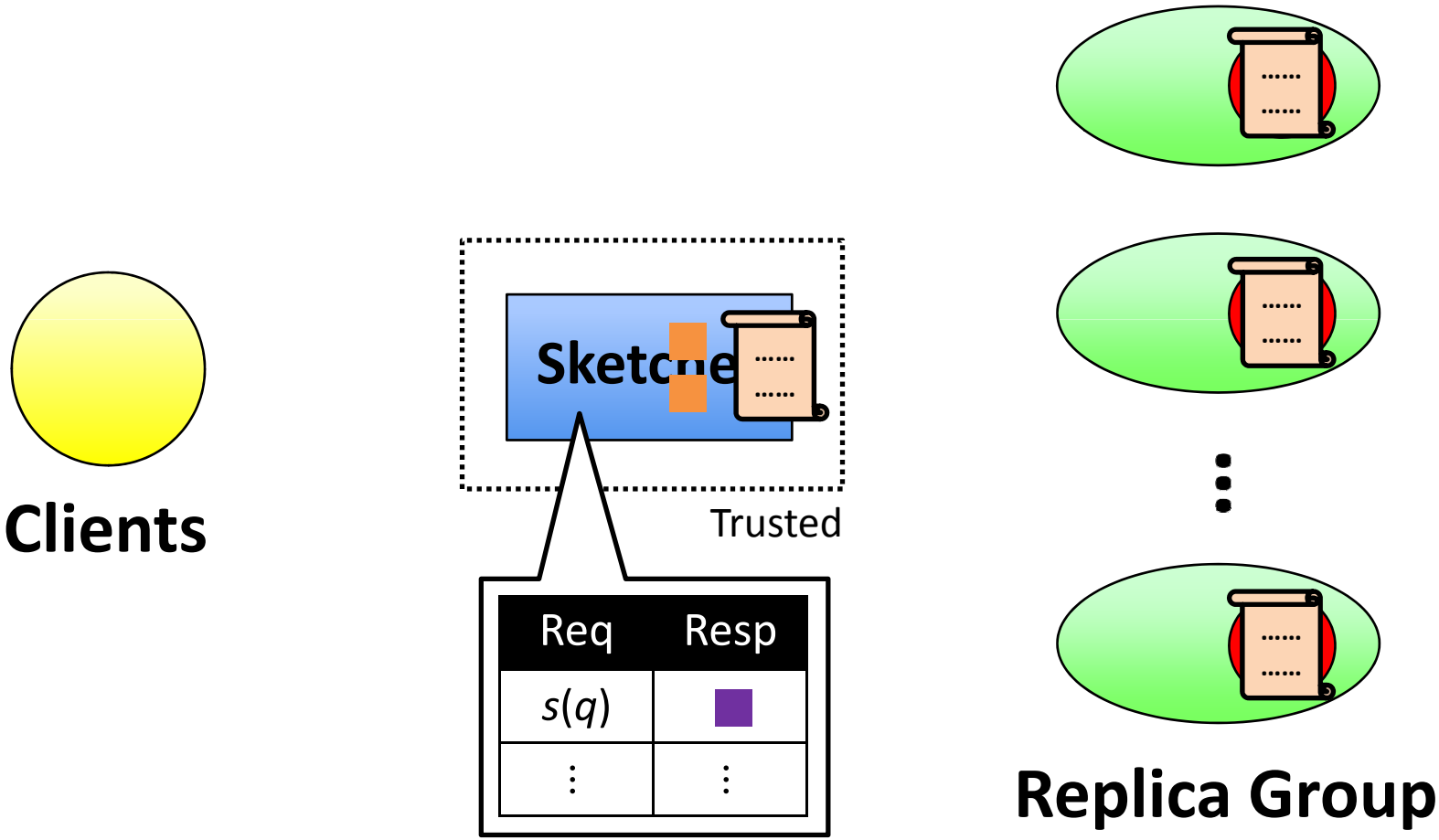
Executing a read



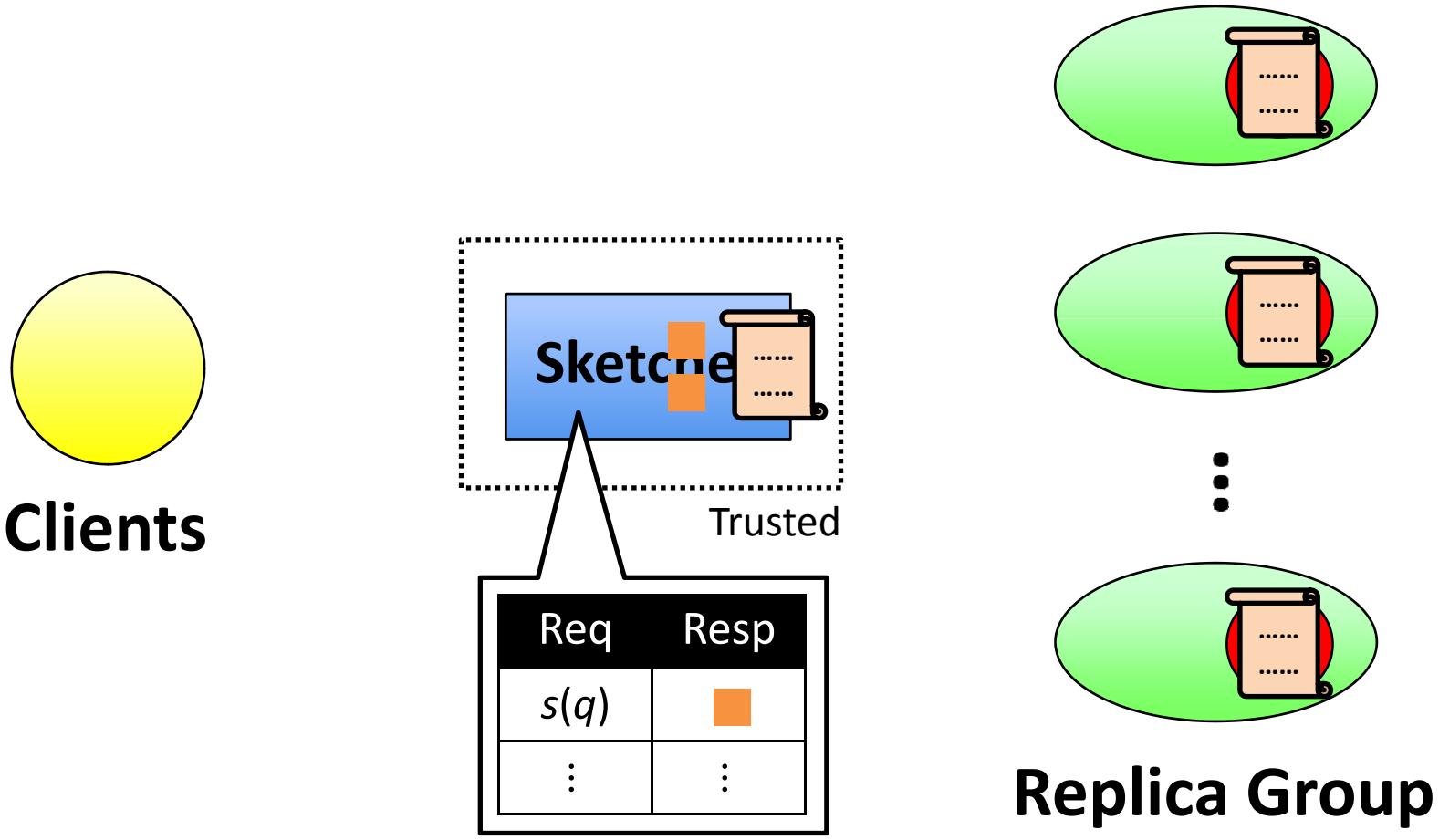
Executing a read



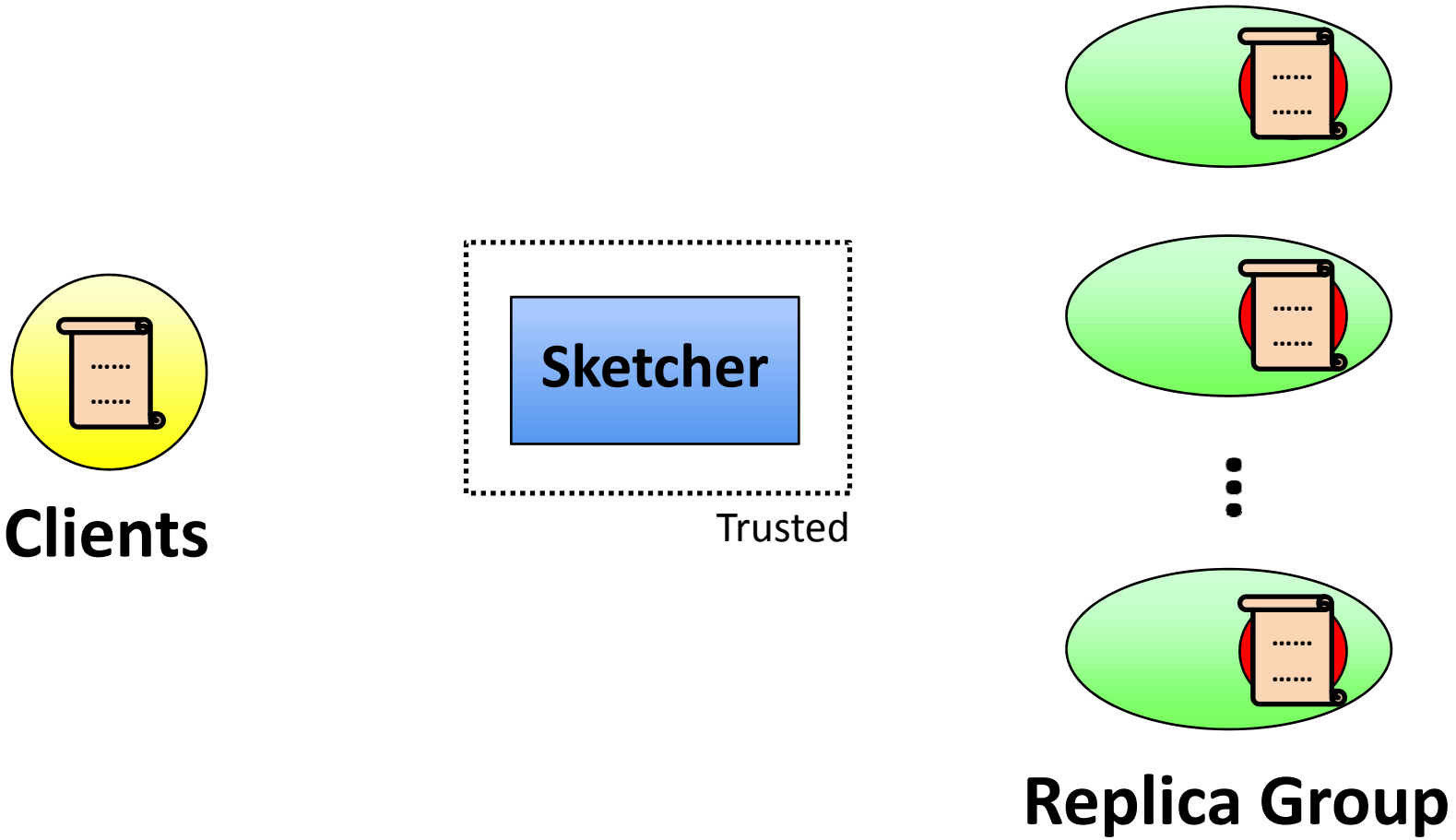
Executing a read



Executing a read

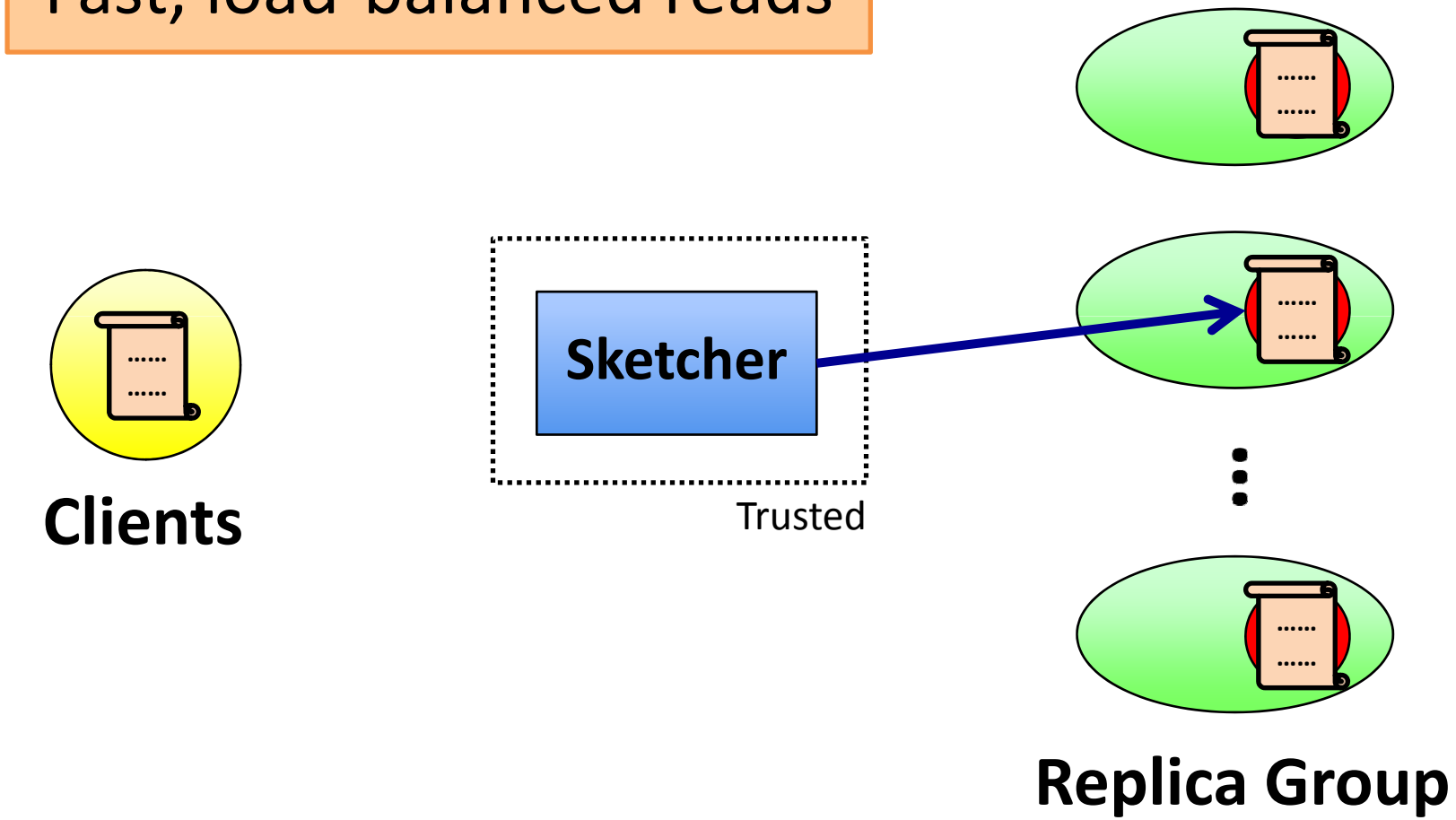


Prophecy



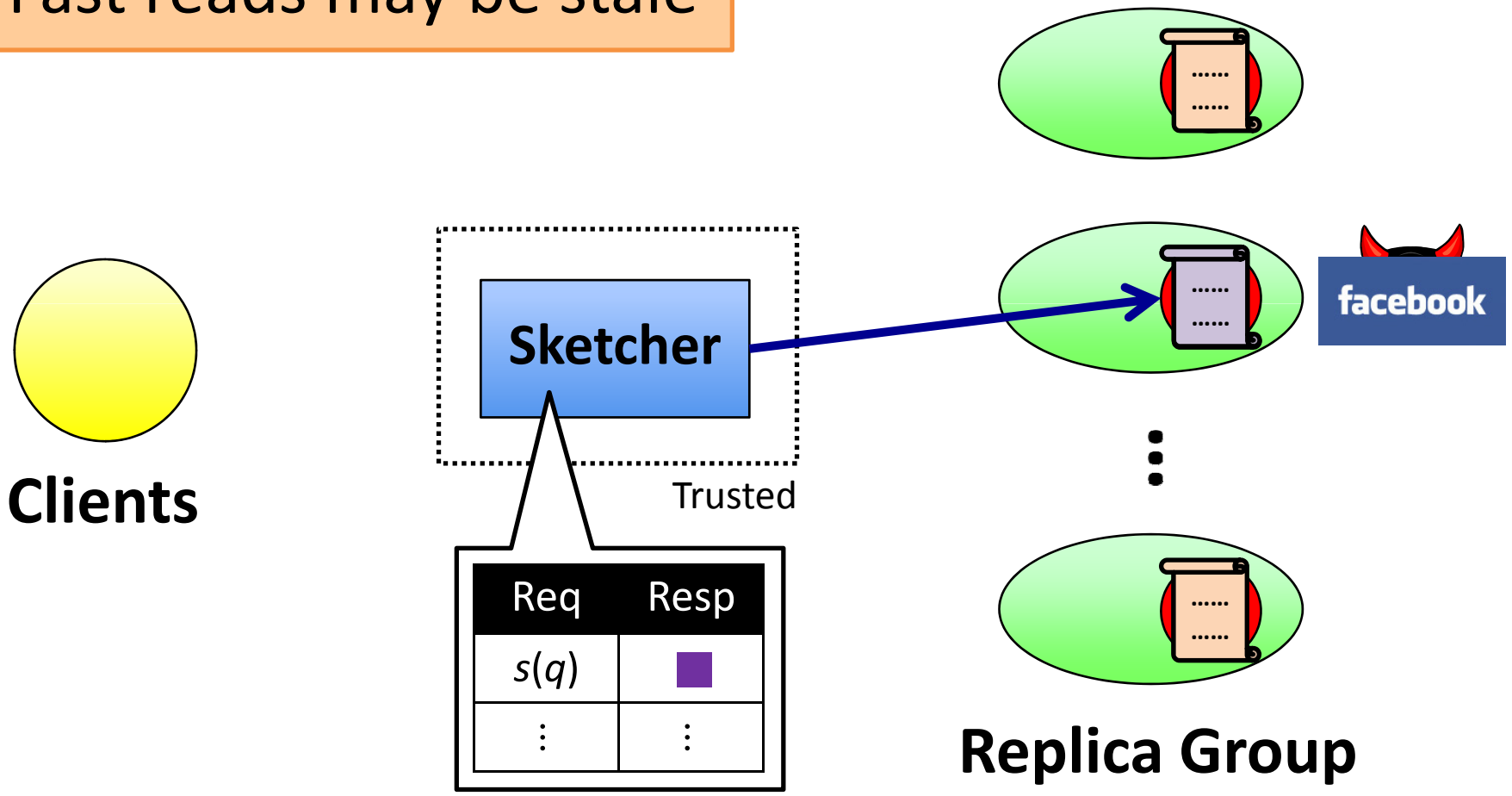
Prophecy

Fast, load-balanced reads



Prophecy

Fast reads may be stale



Delay-once linearizability

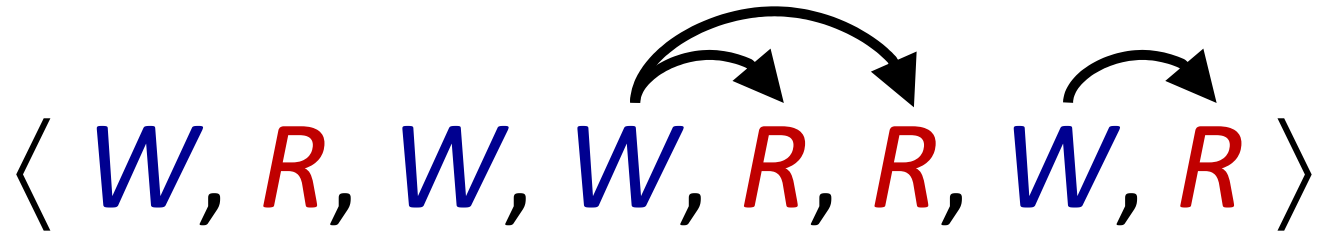
~~Delay-once~~ linearizability

~~Delay-once~~ linearizability

$\langle\, W, R, W, W, R, R, W, R \,\rangle$

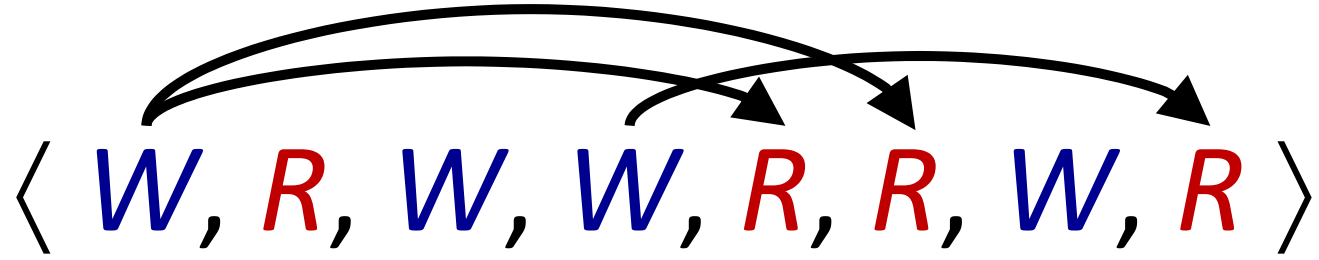
~~Delay-once~~ linearizability

Read-after-write property



Delay-once linearizability

~~Read-after-write property~~



Example application

- Upload embarrassing photos
  1. Remove colleagues from ACL
  2. Upload photos
  3. (Refresh)
- Weak may reorder
- Delay-once preserves order

Byzantine fault tolerance (BFT)

- ~~Low throughput~~
- ~~Modifies clients~~
- ~~Long-lived sessions~~

D-Prophecy

Prophecy

Implementation

- Modified PBFT
  - PBFT is stable, complete
  - Competitive with Zyzzyva et. al.
- C++, Tamer async I/O
  - Sketcher: ~2000 LOC
  - PBFT library: ~1140 LOC
  - PBFT client: ~1000 LOC

Evaluation

- Prophecy vs. proxied-PBFT
  - Proxied systems

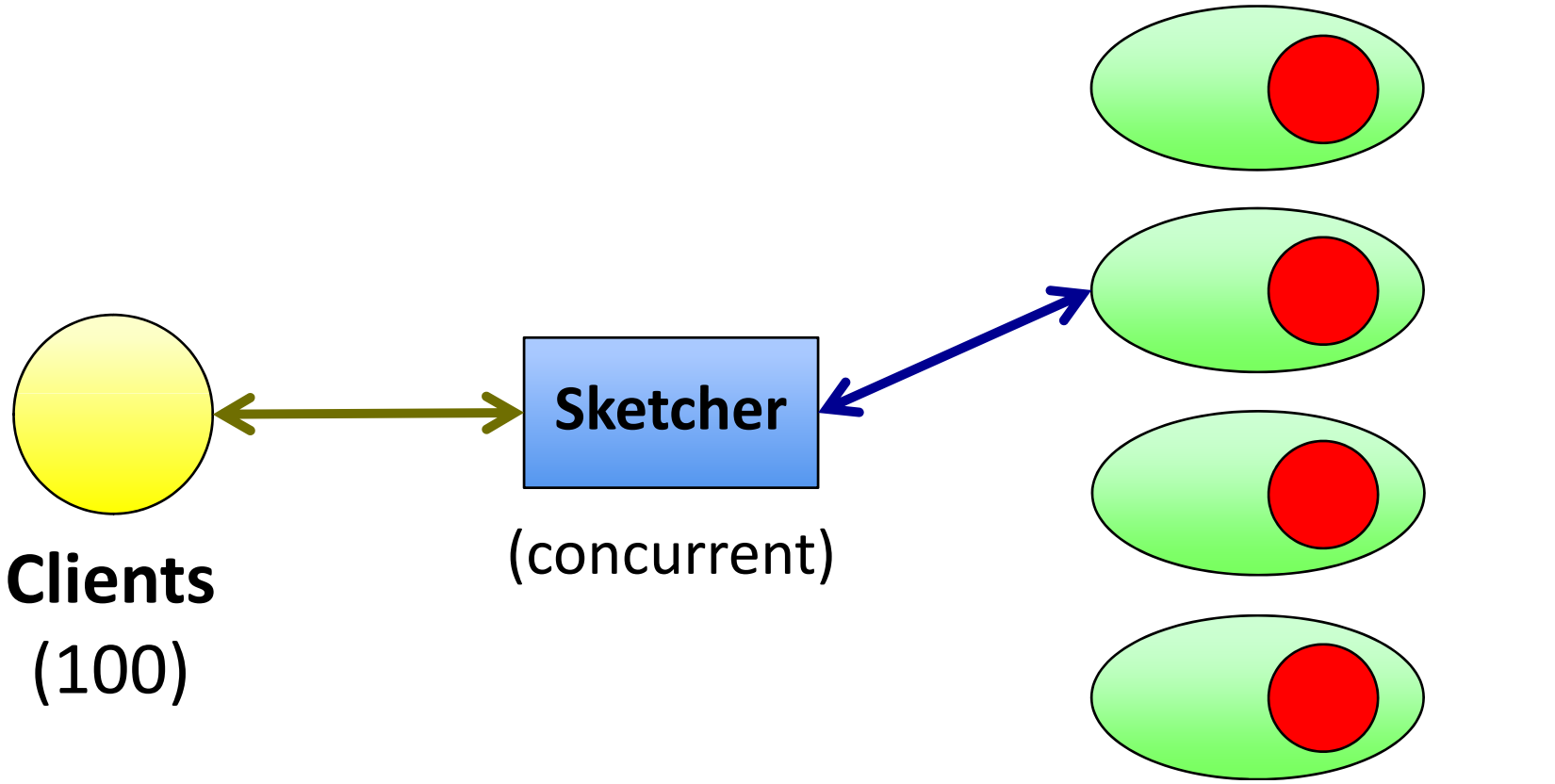
- D-Prophecy vs. PBFT
  - Non-proxied systems

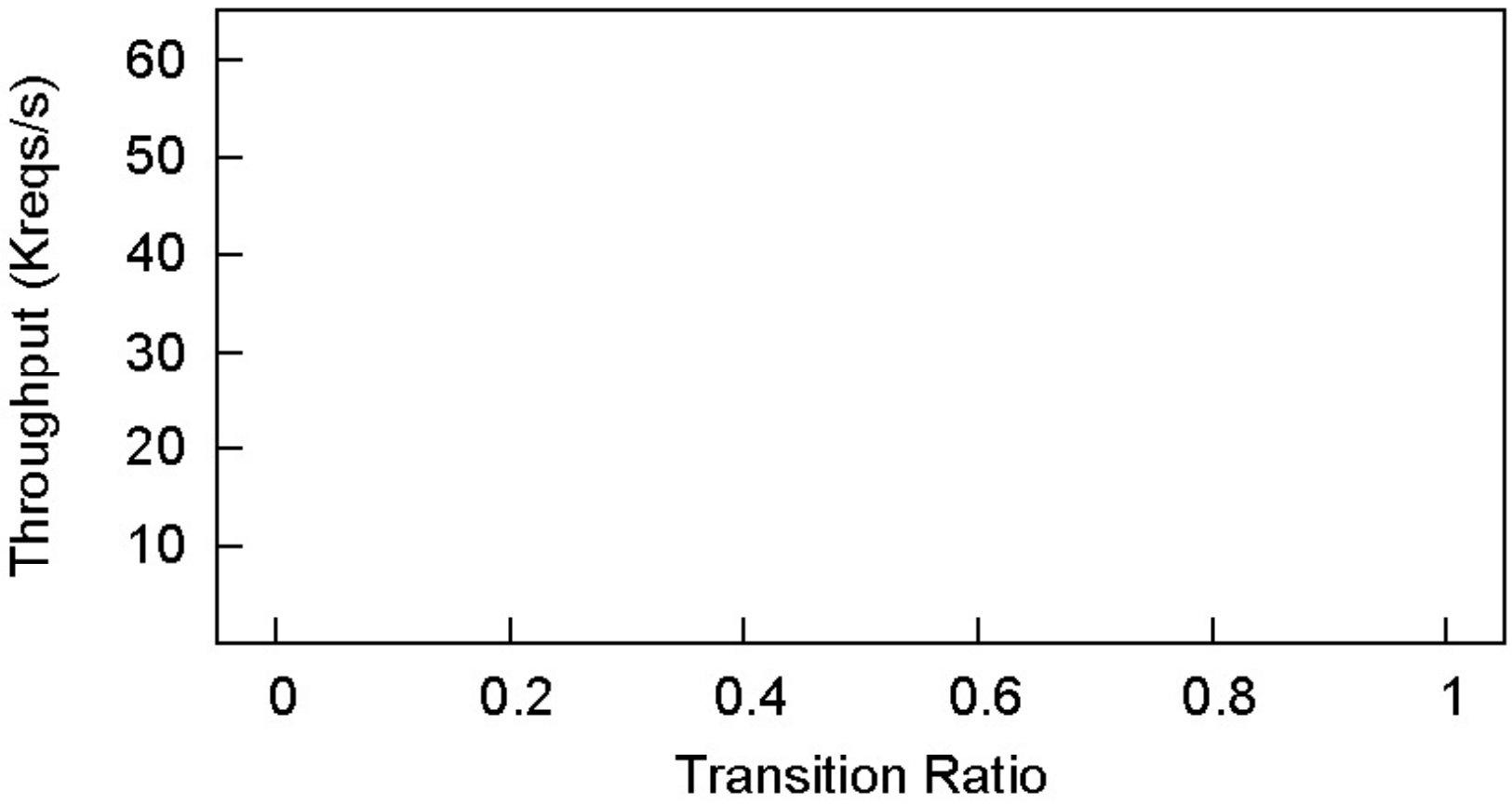
Evaluation

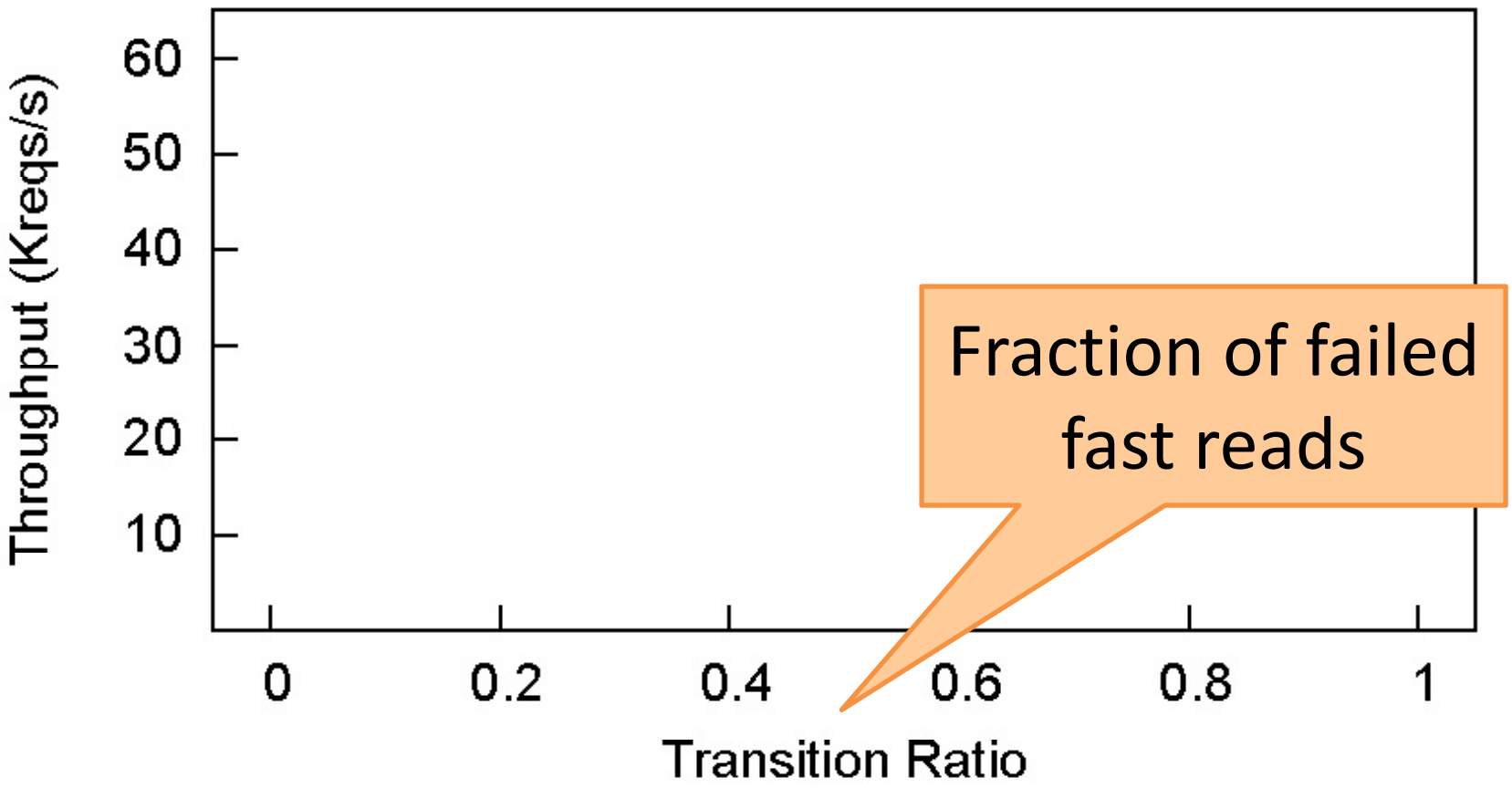
- Prophecy vs. proxied-PBFT
  - Proxied systems

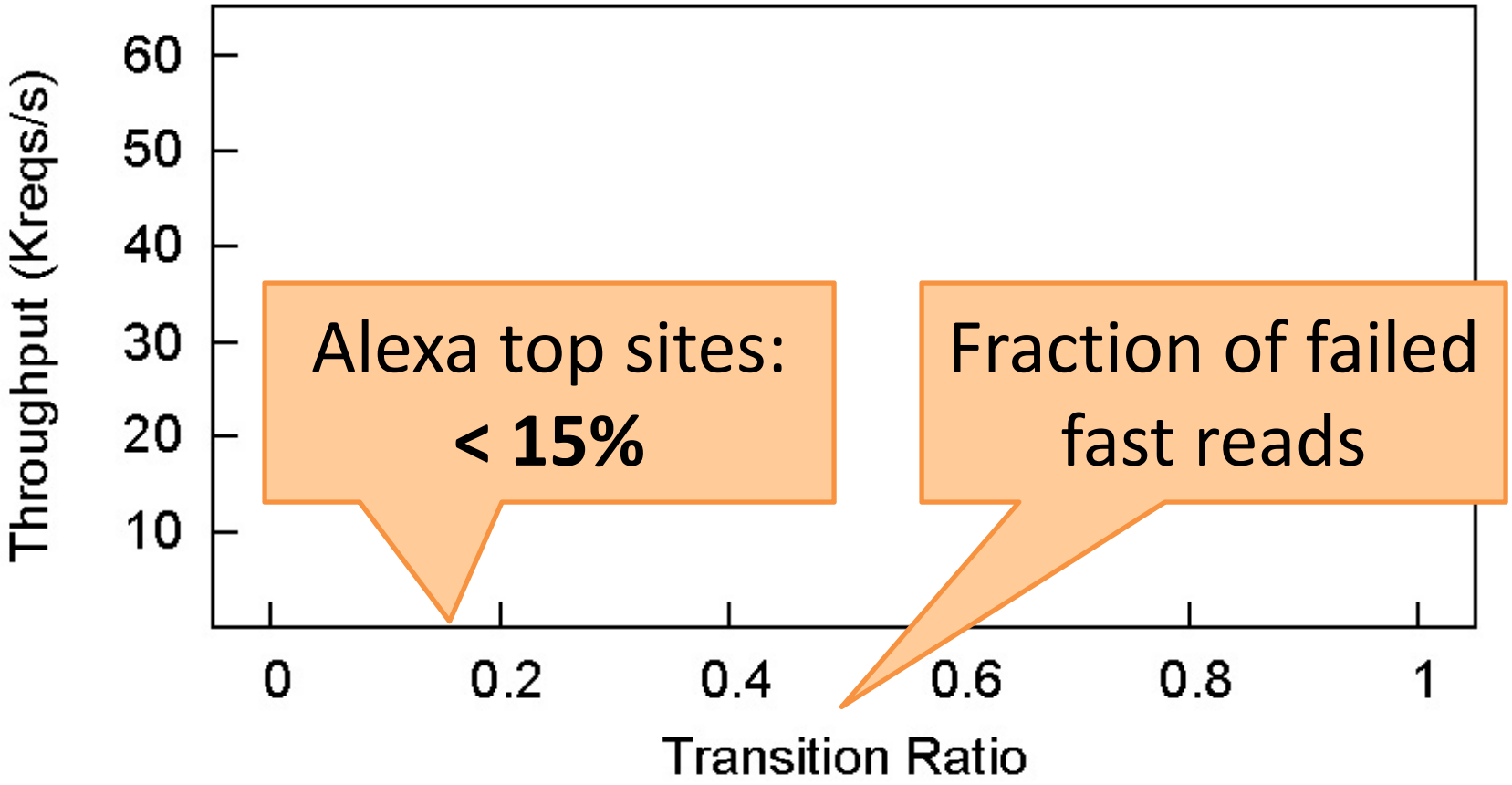
- We will study:
  - Performance on “null” workloads
  - Performance with real replicated service
  - Where system bottlenecks, how to scale

Basic setup

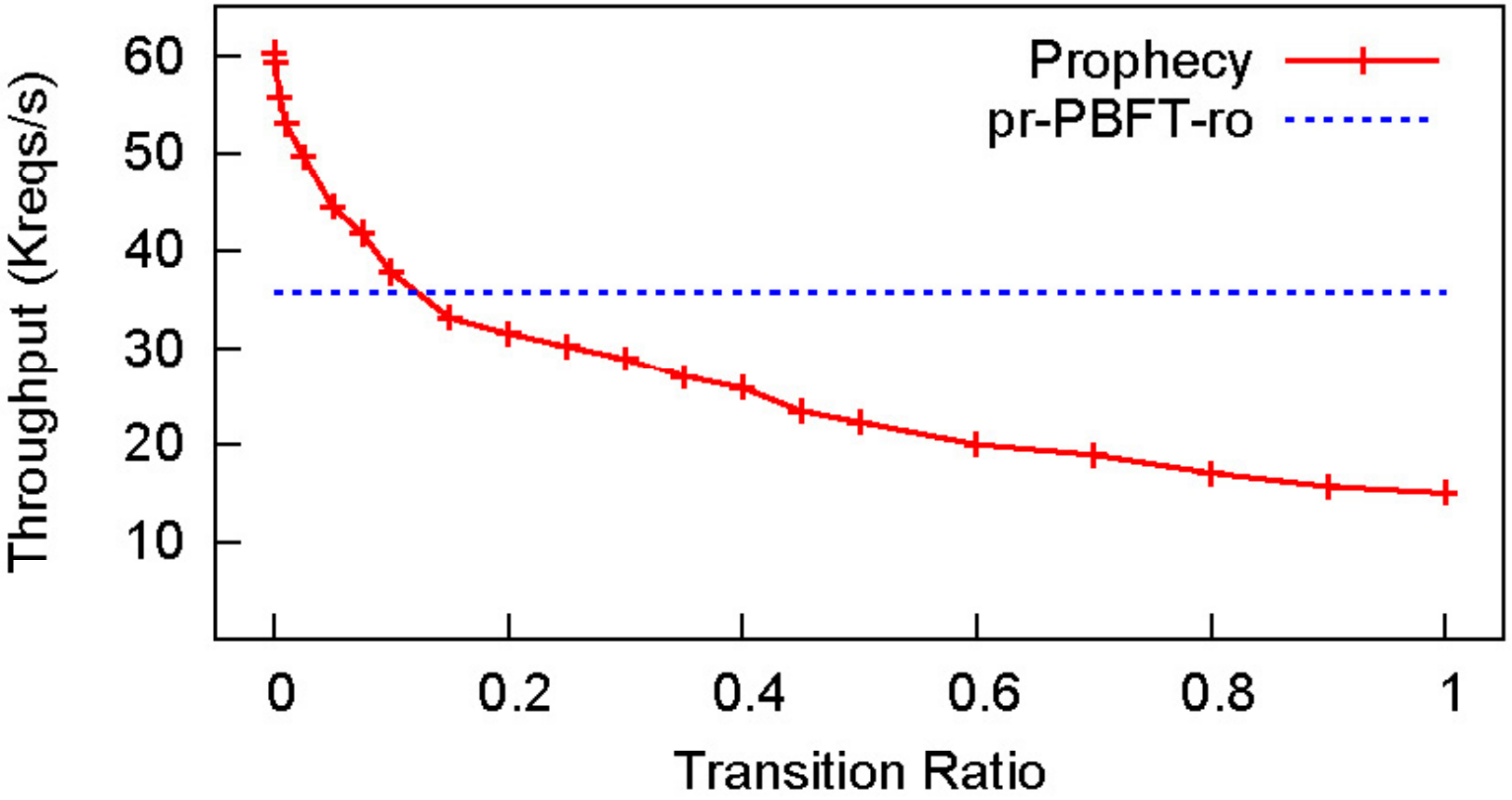



Throughput (Kreqs/s)
60
50
40
30
20
10
0
0.2
0.4
0.6
0.8
1
Transition Ratio

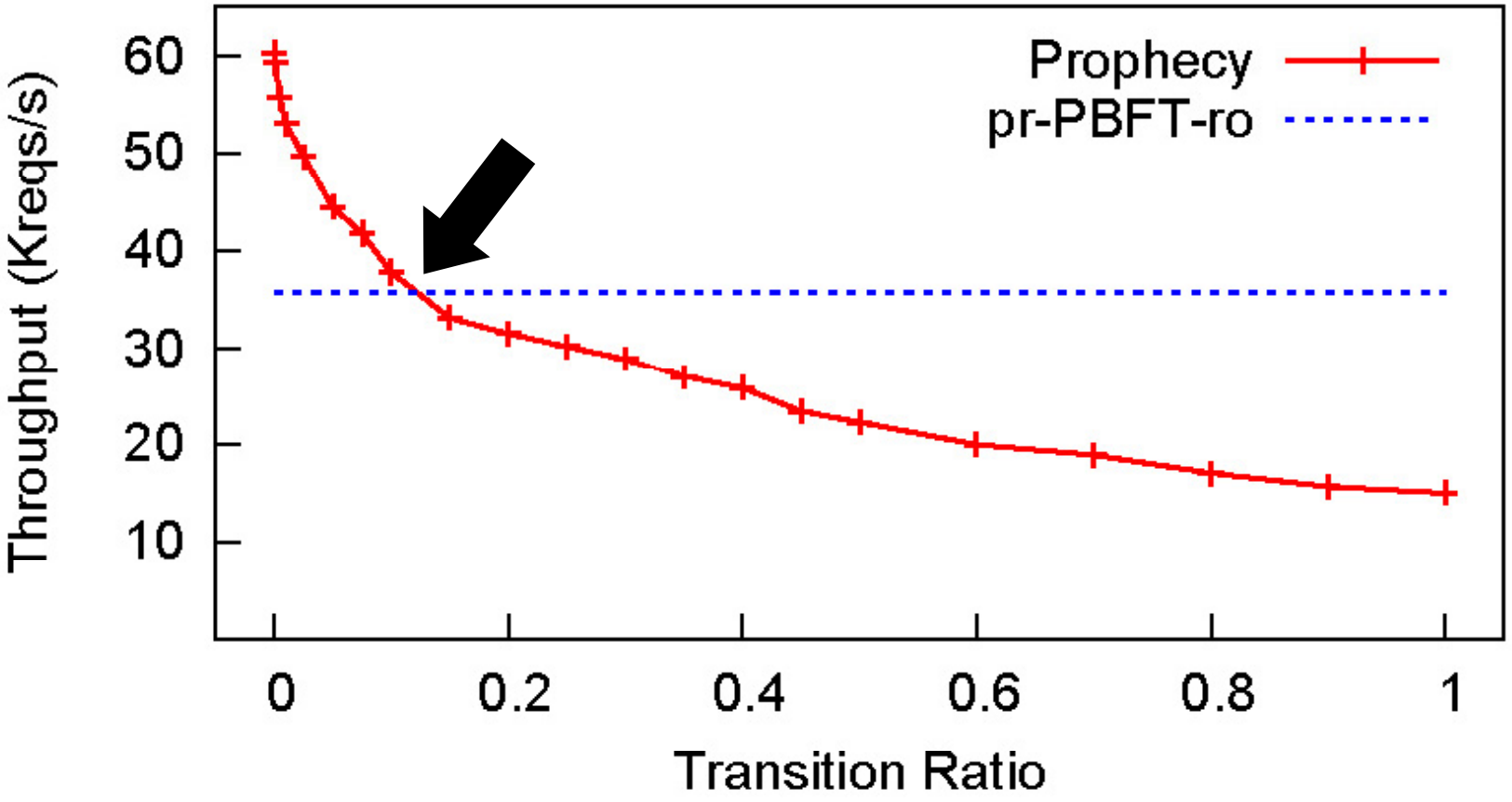

Throughput (Kreqs/s)
60
50
40
30
20
10
0
0.2
0.4
0.6
0.8
1
Transition Ratio
Fraction of failed fast reads


Throughput (Kreqs/s)
60
50
40
30
20
10
0
0.2
0.4
0.6
0.8
1
Transition Ratio
Alexa top sites:
< 15%
Fraction of failed
fast reads

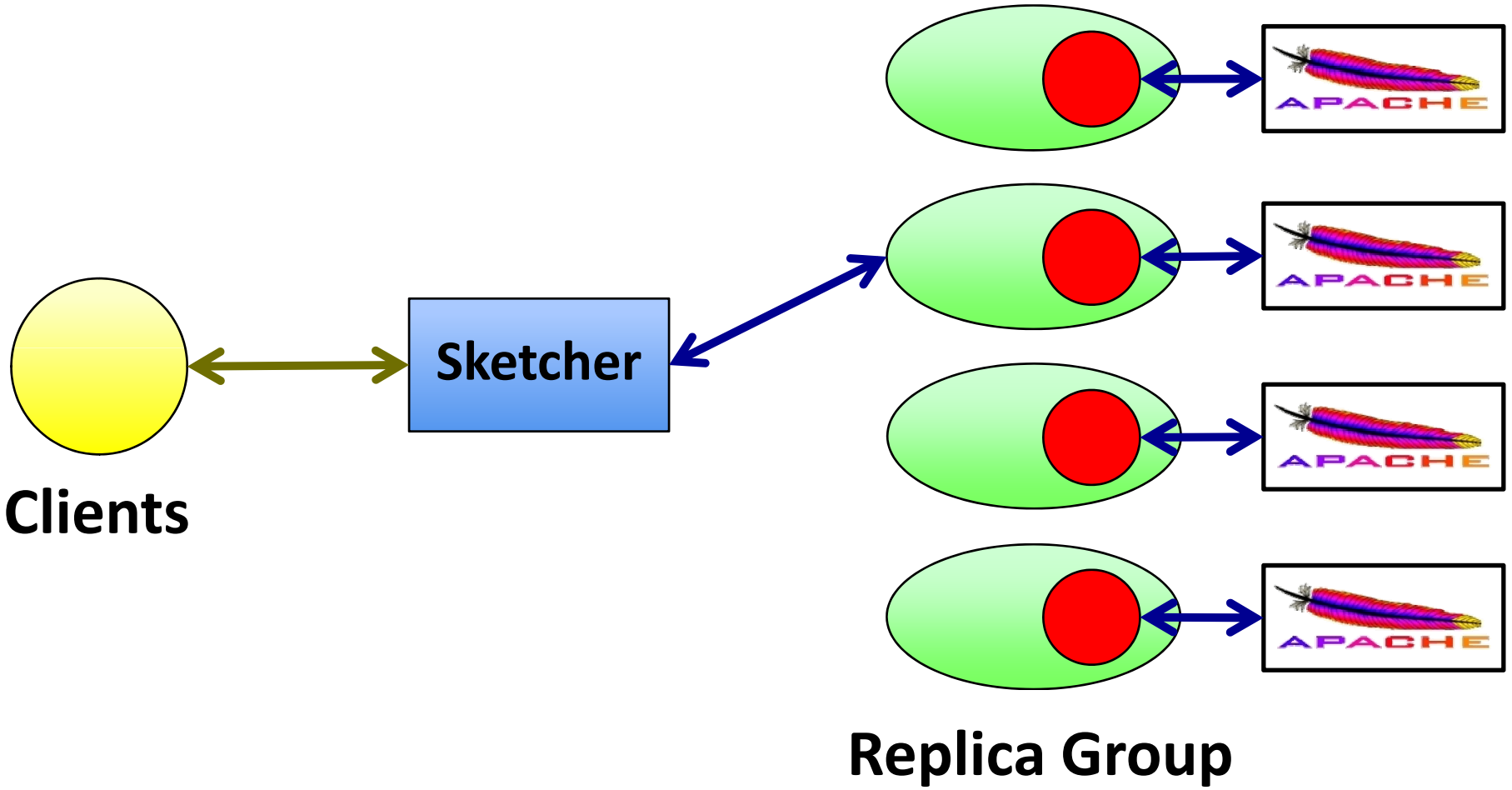
Small benefit on null reads



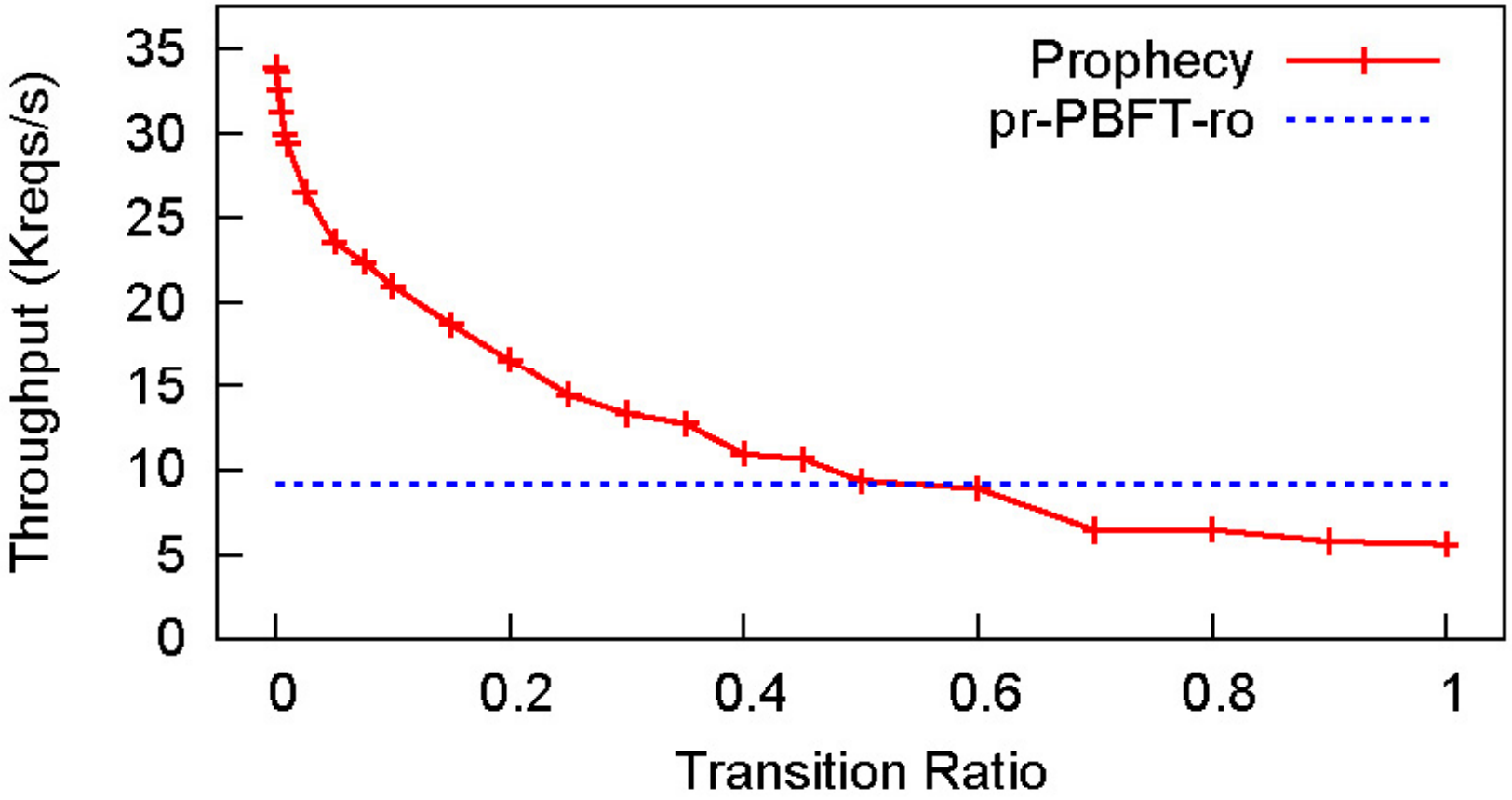
Small benefit on null reads



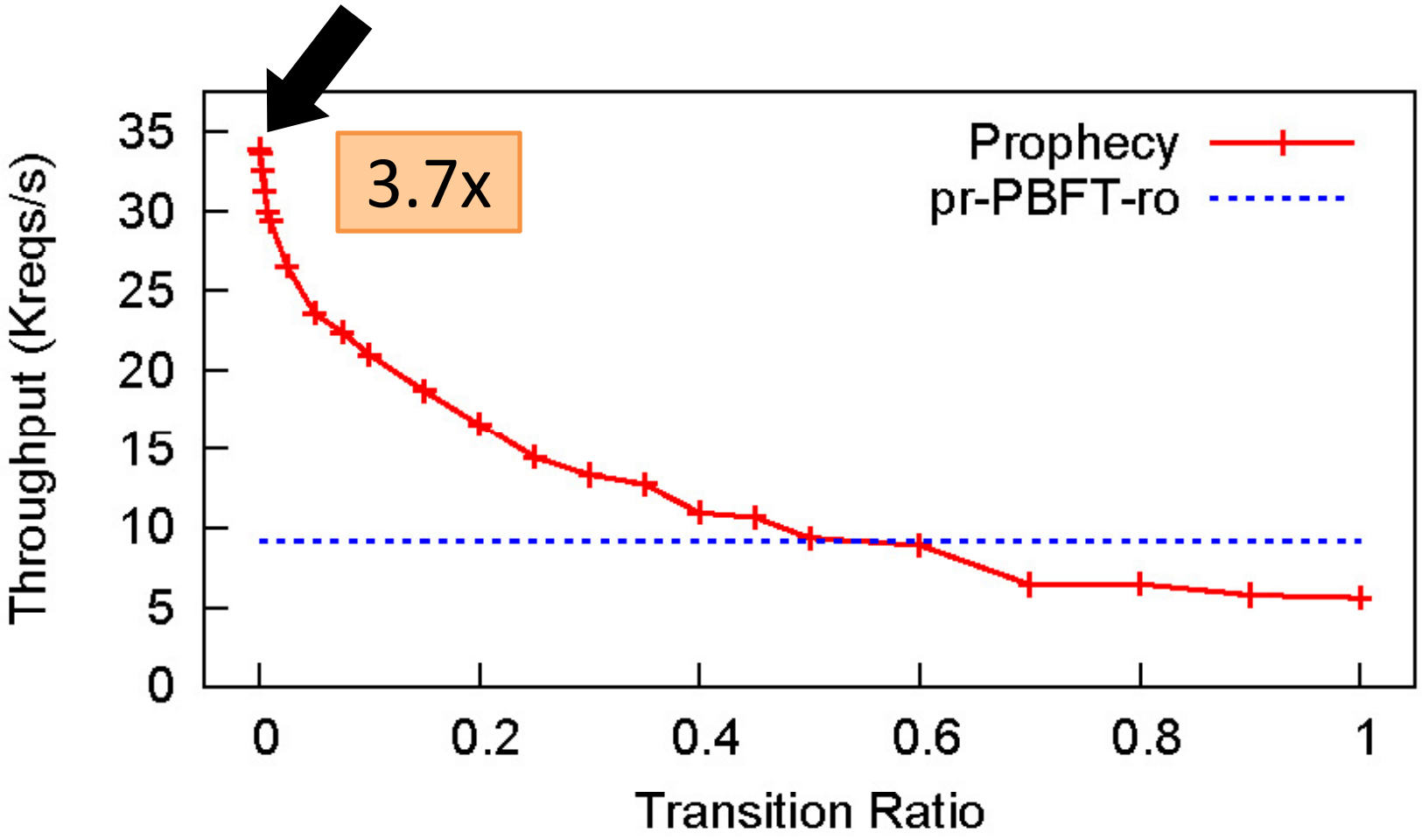
Apache webserver setup



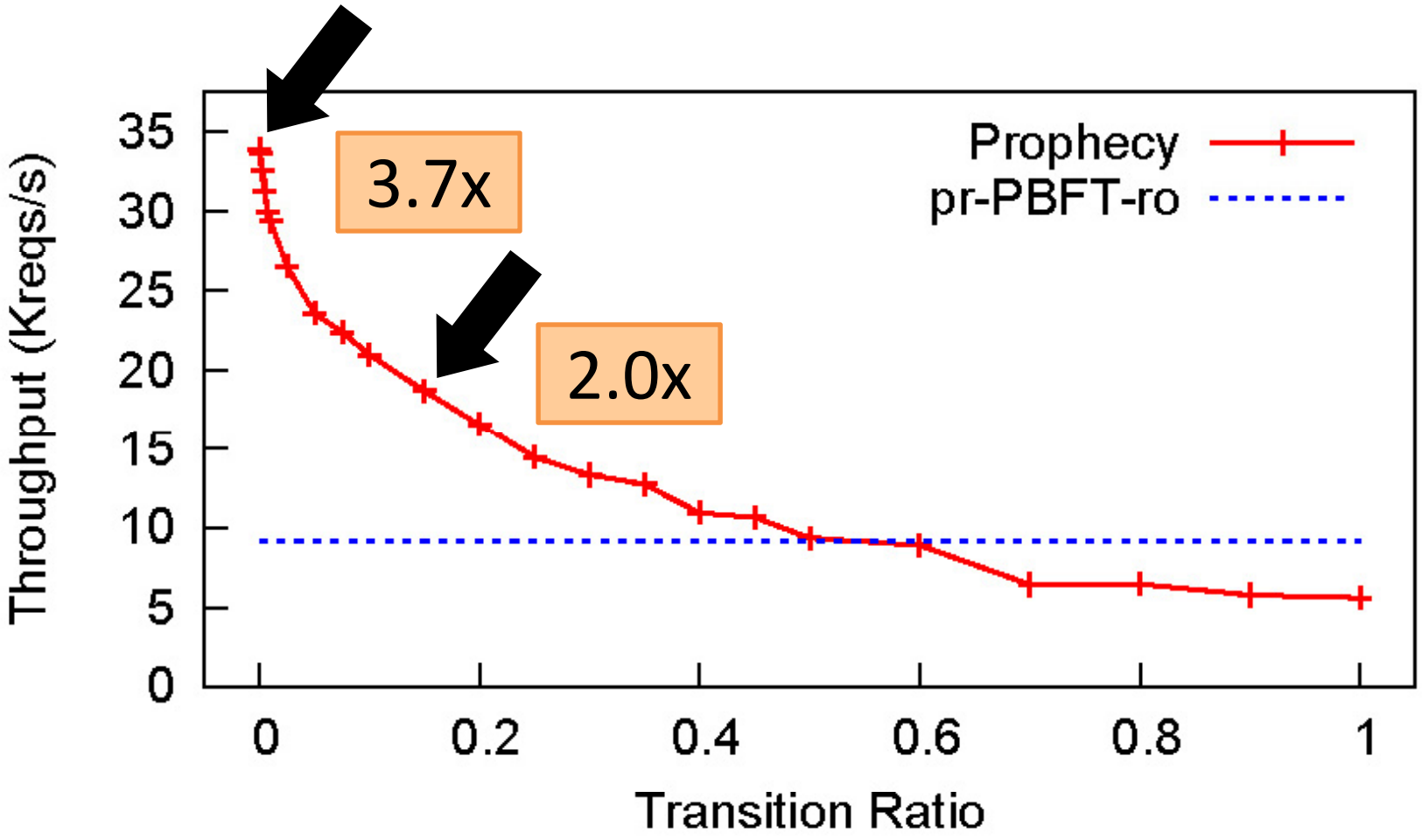
Large benefit on real workload



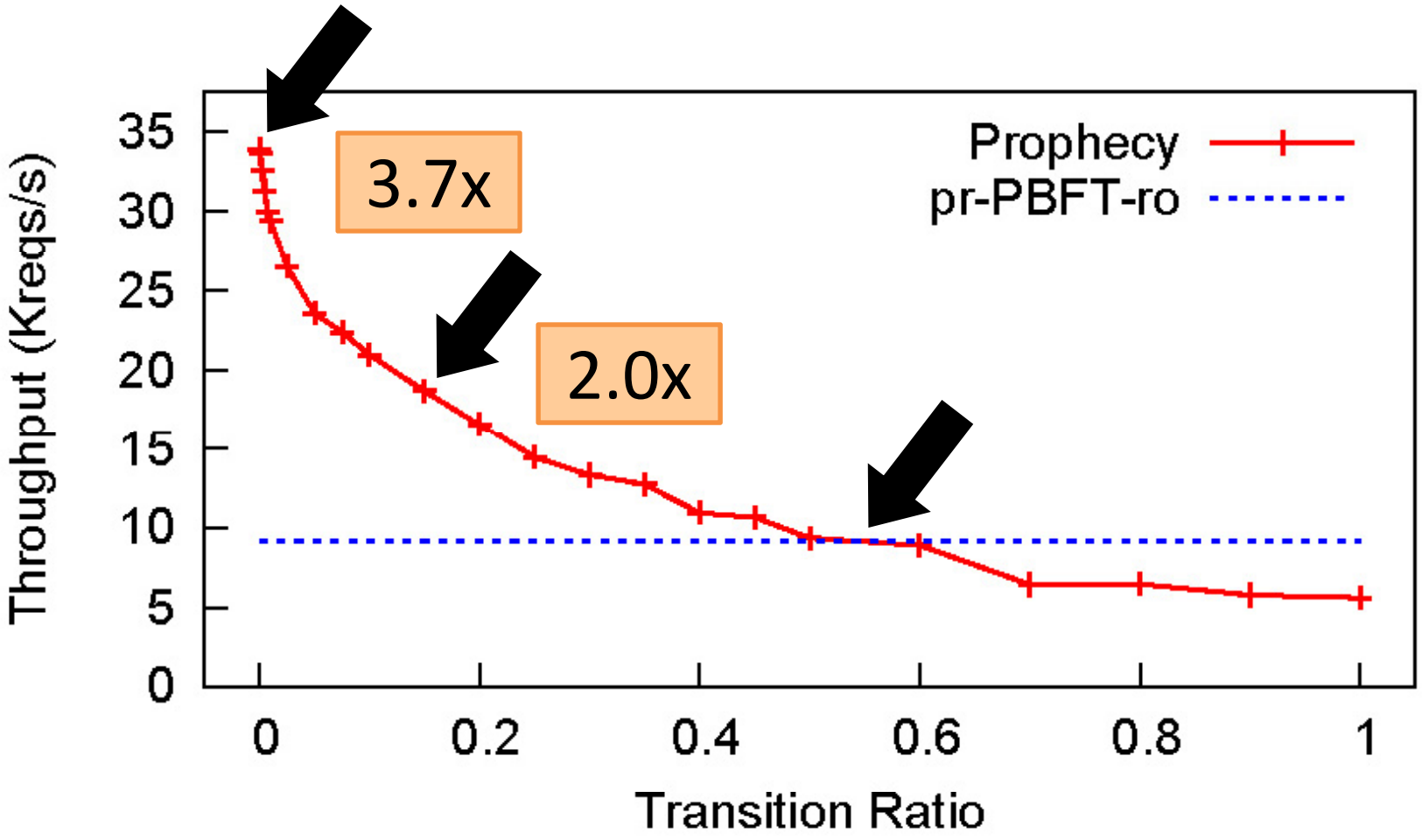
Large benefit on real workload



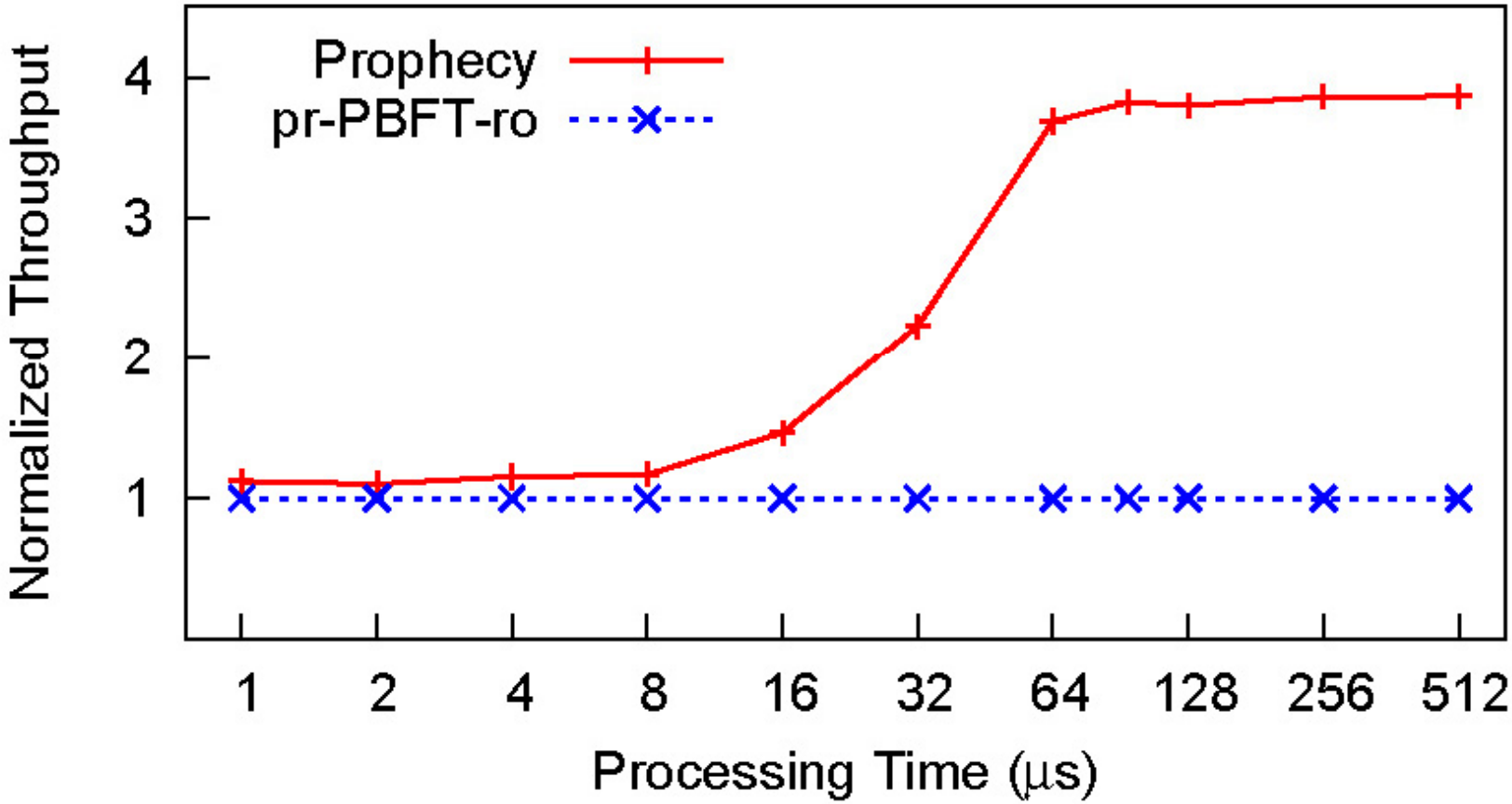
Large benefit on real workload



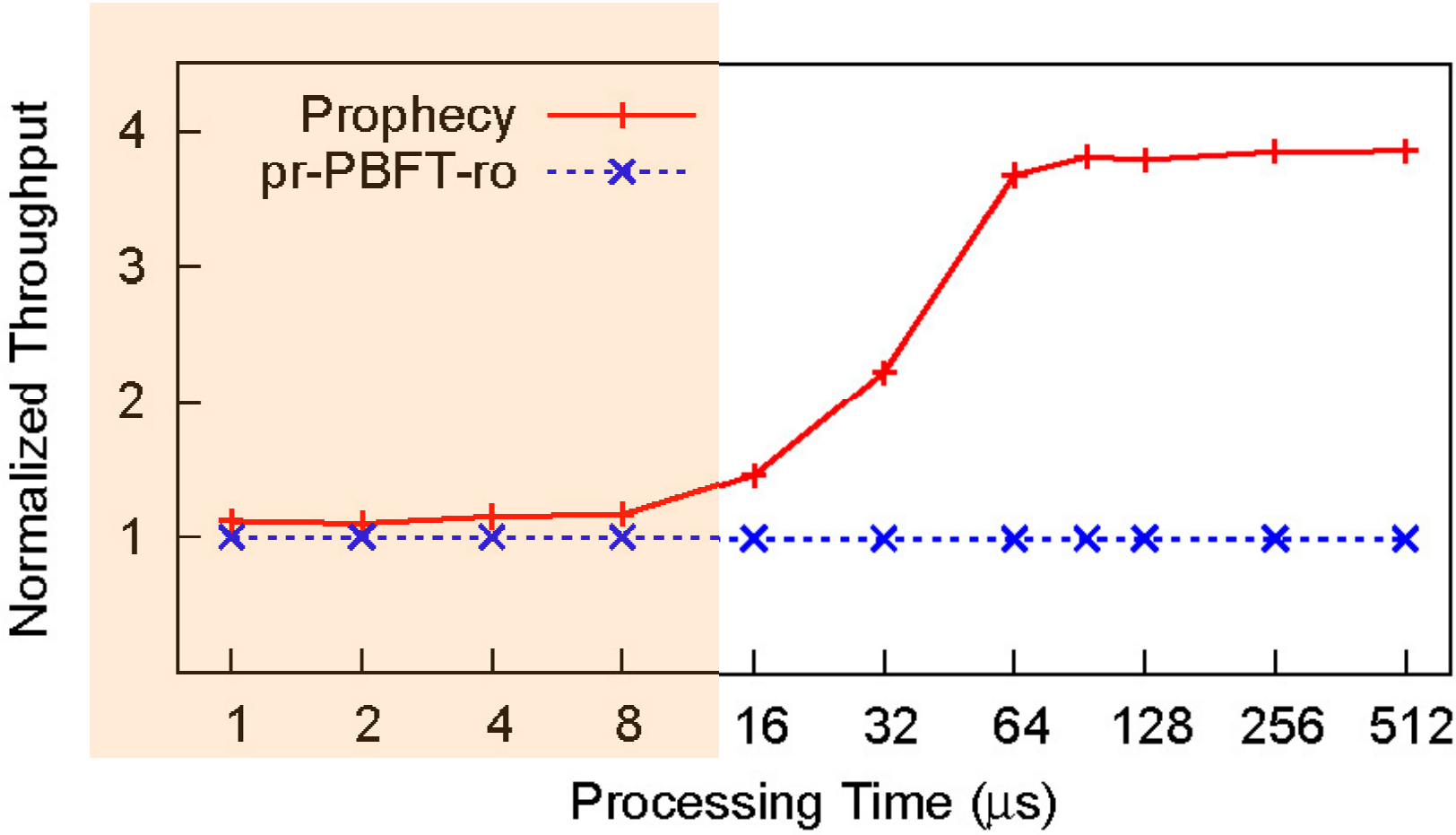
Large benefit on real workload



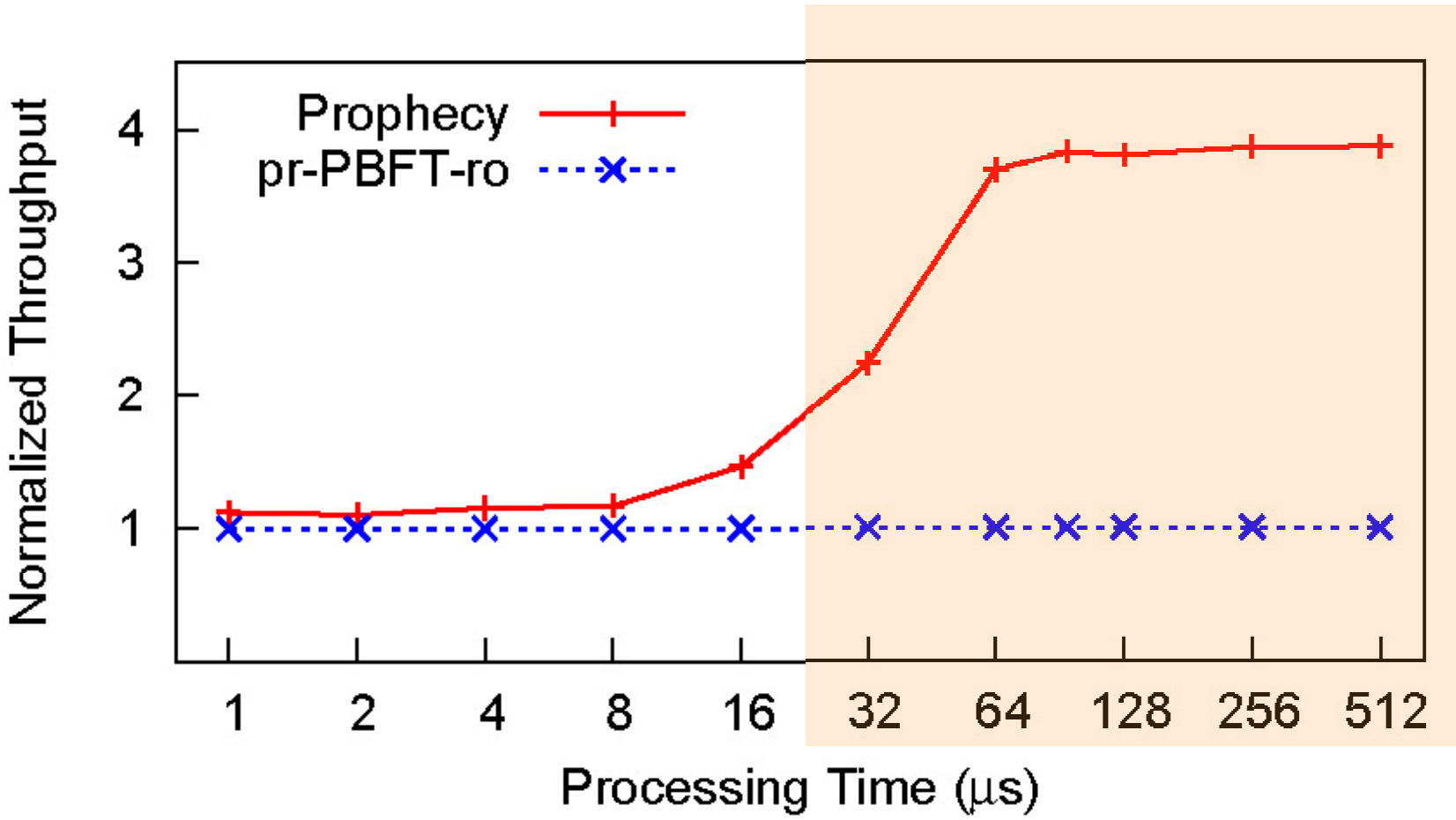
Benefit grows with work



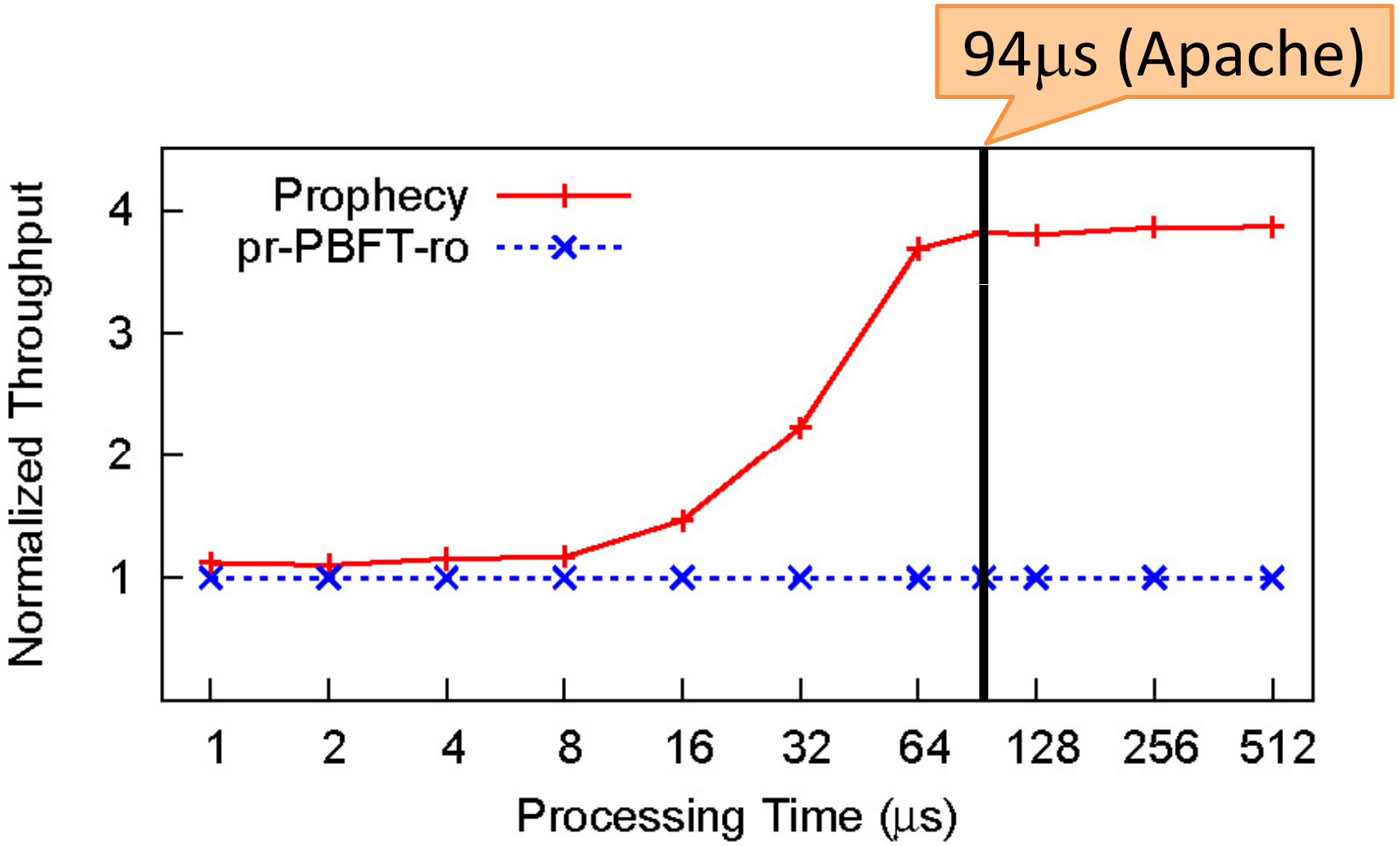
Benefit grows with work



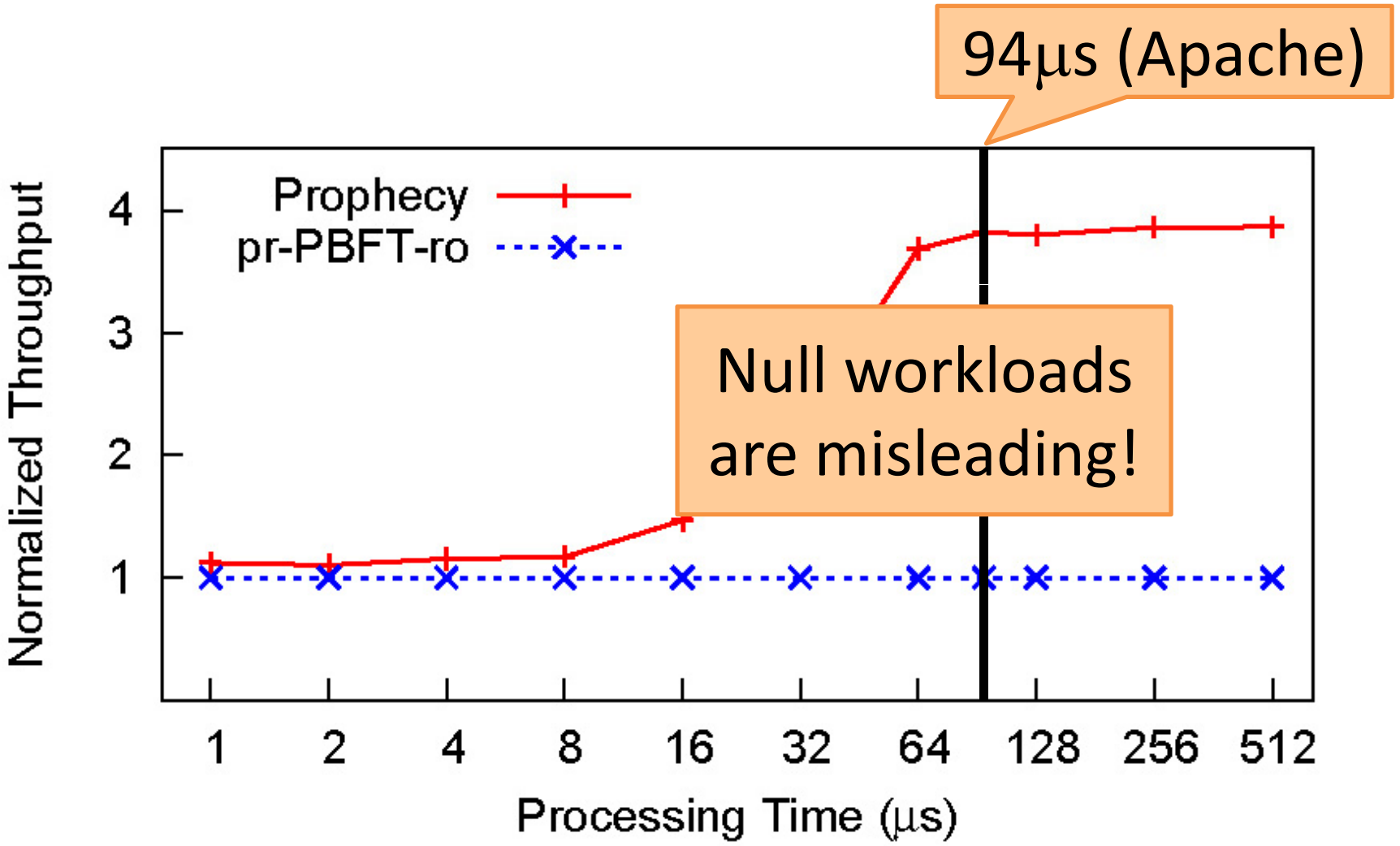
Benefit grows with work



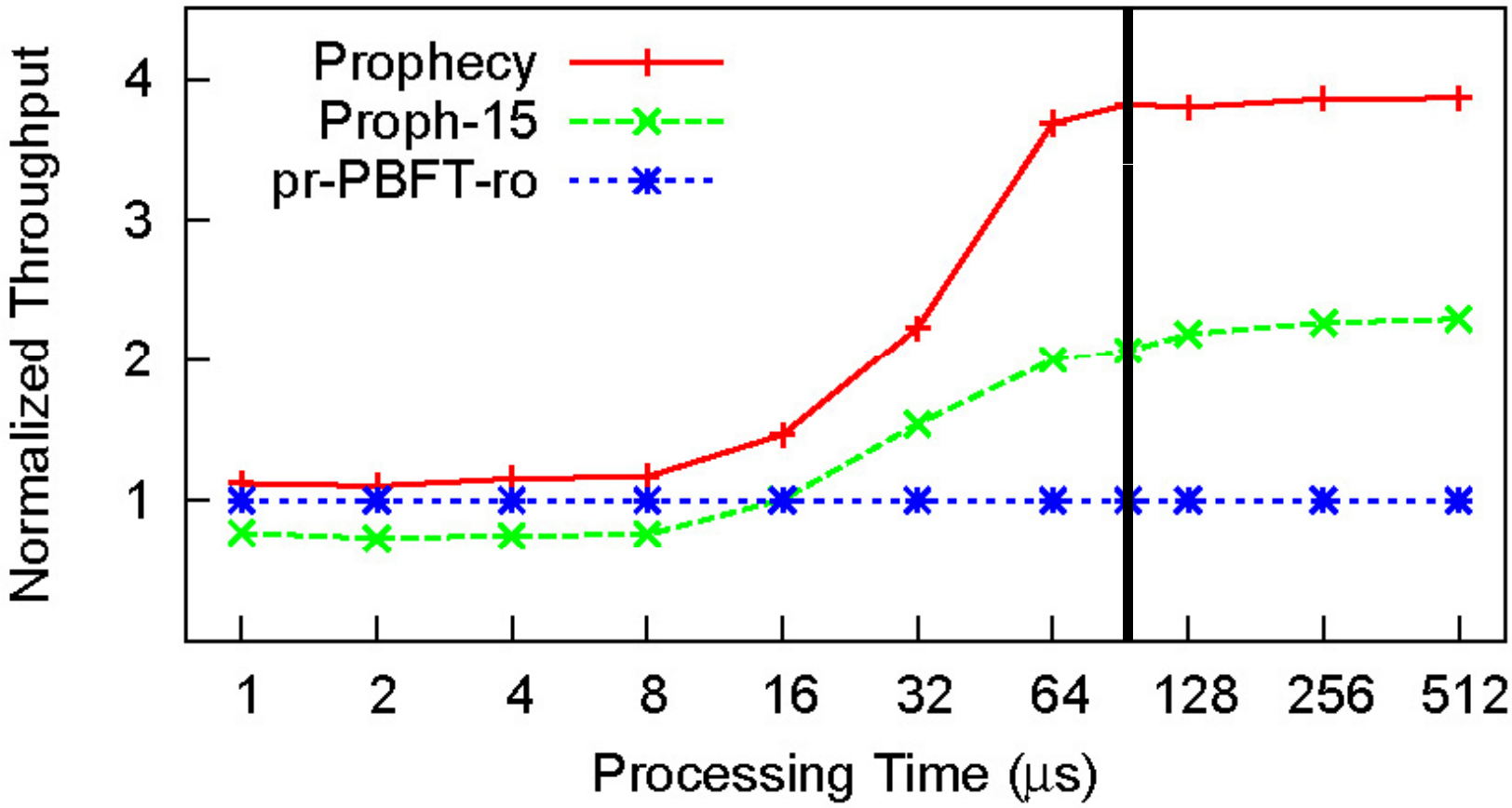
Benefit grows with work



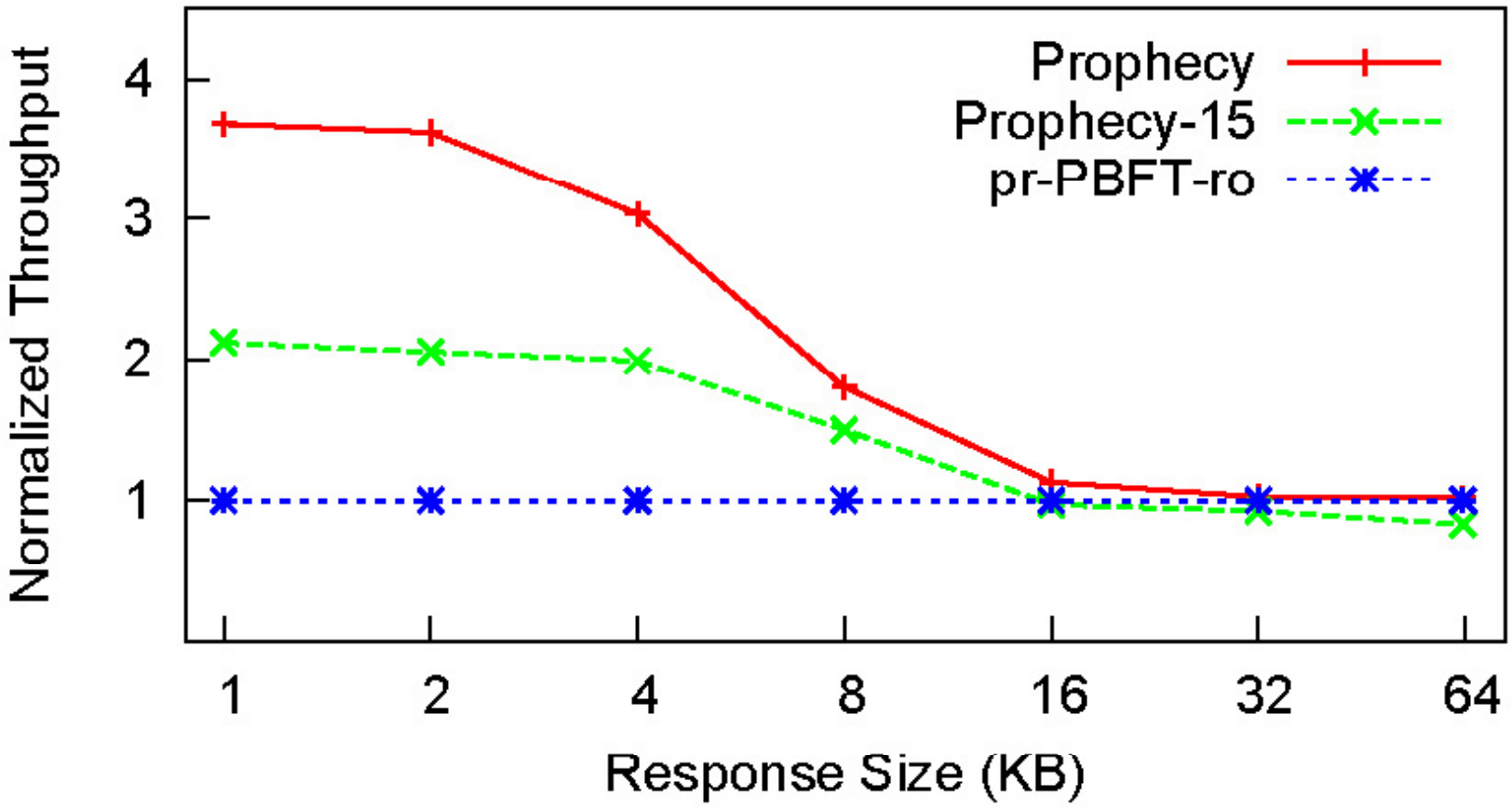
Benefit grows with work



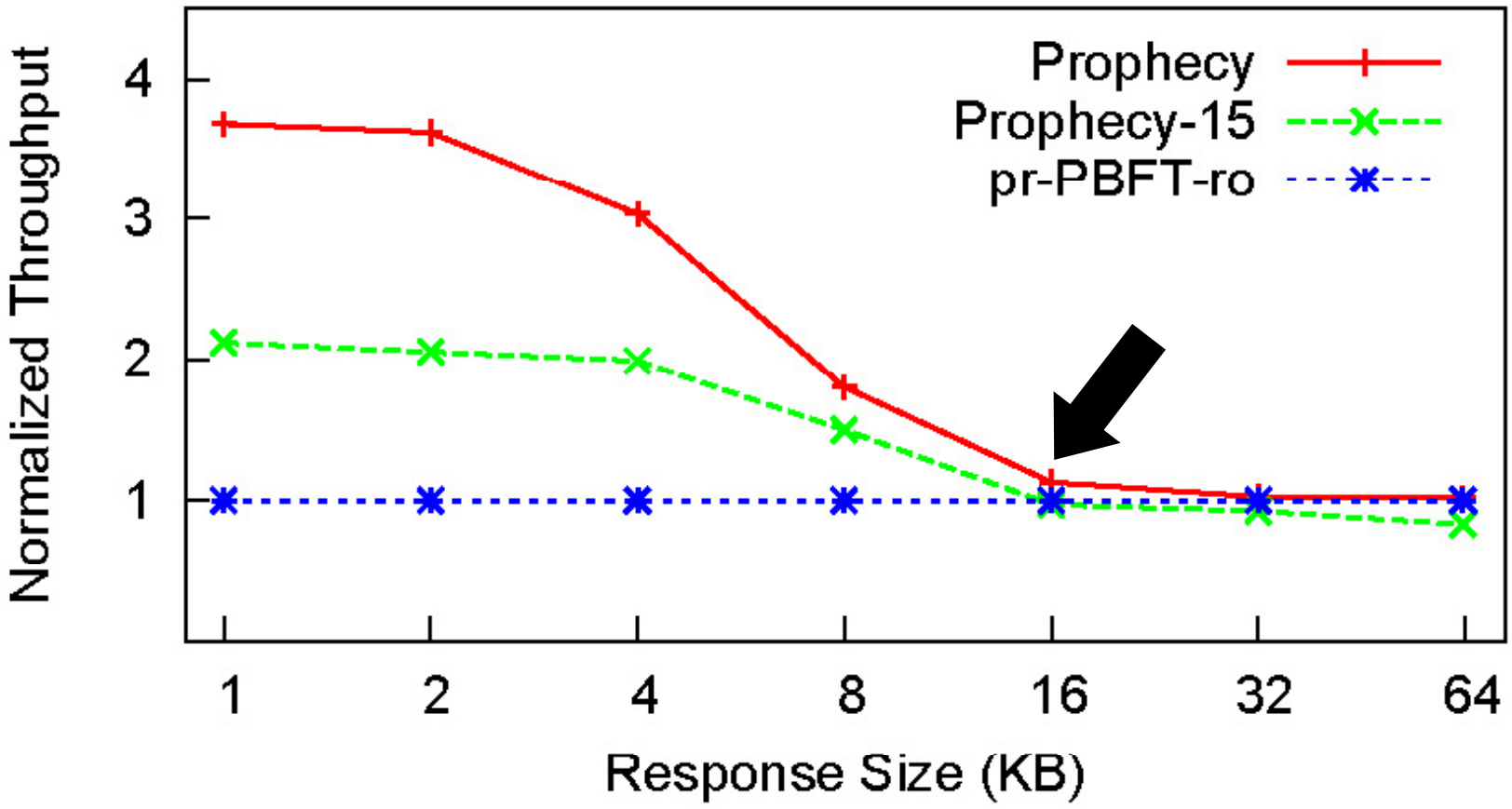
Benefit grows with work



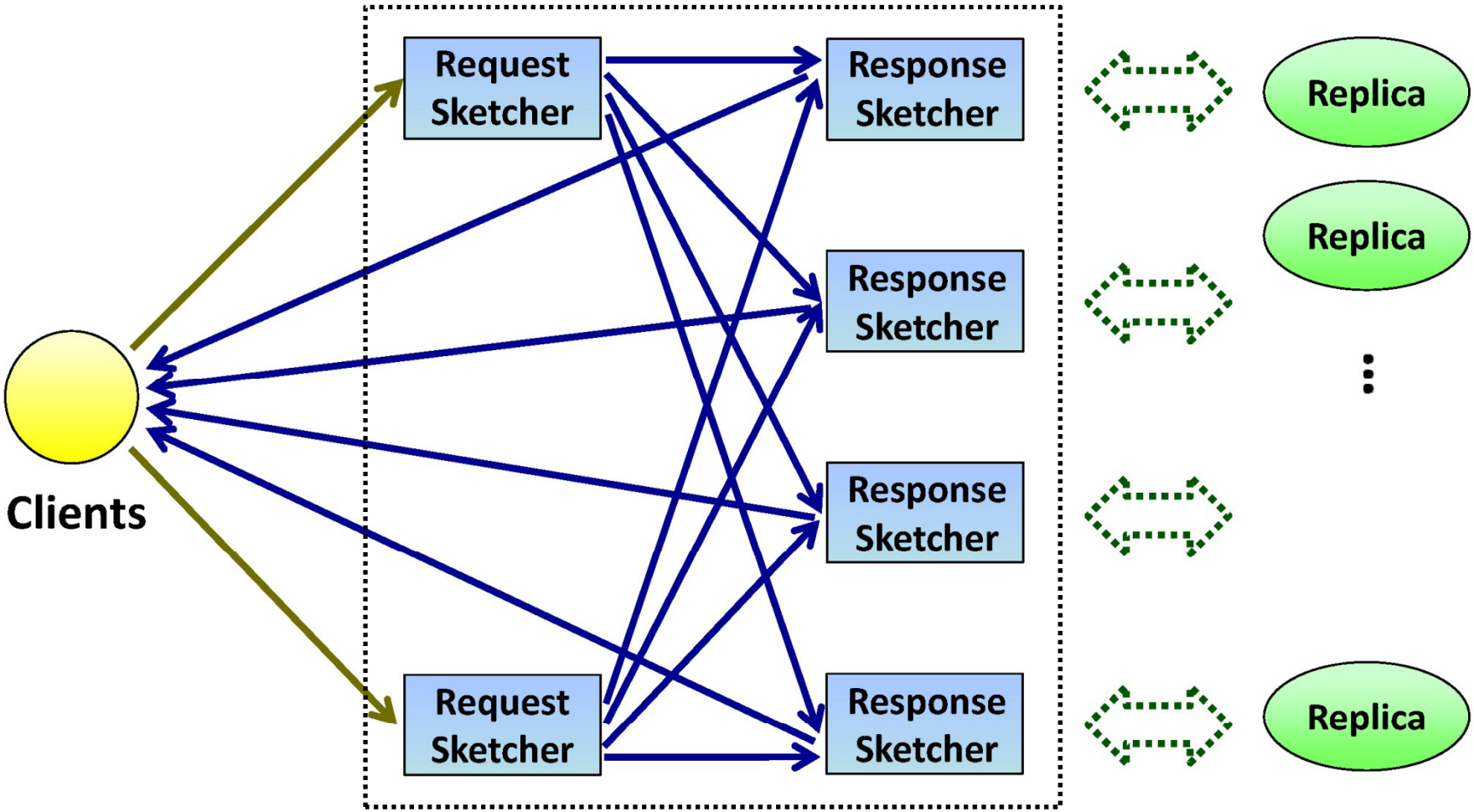
Single sketcher bottlenecks



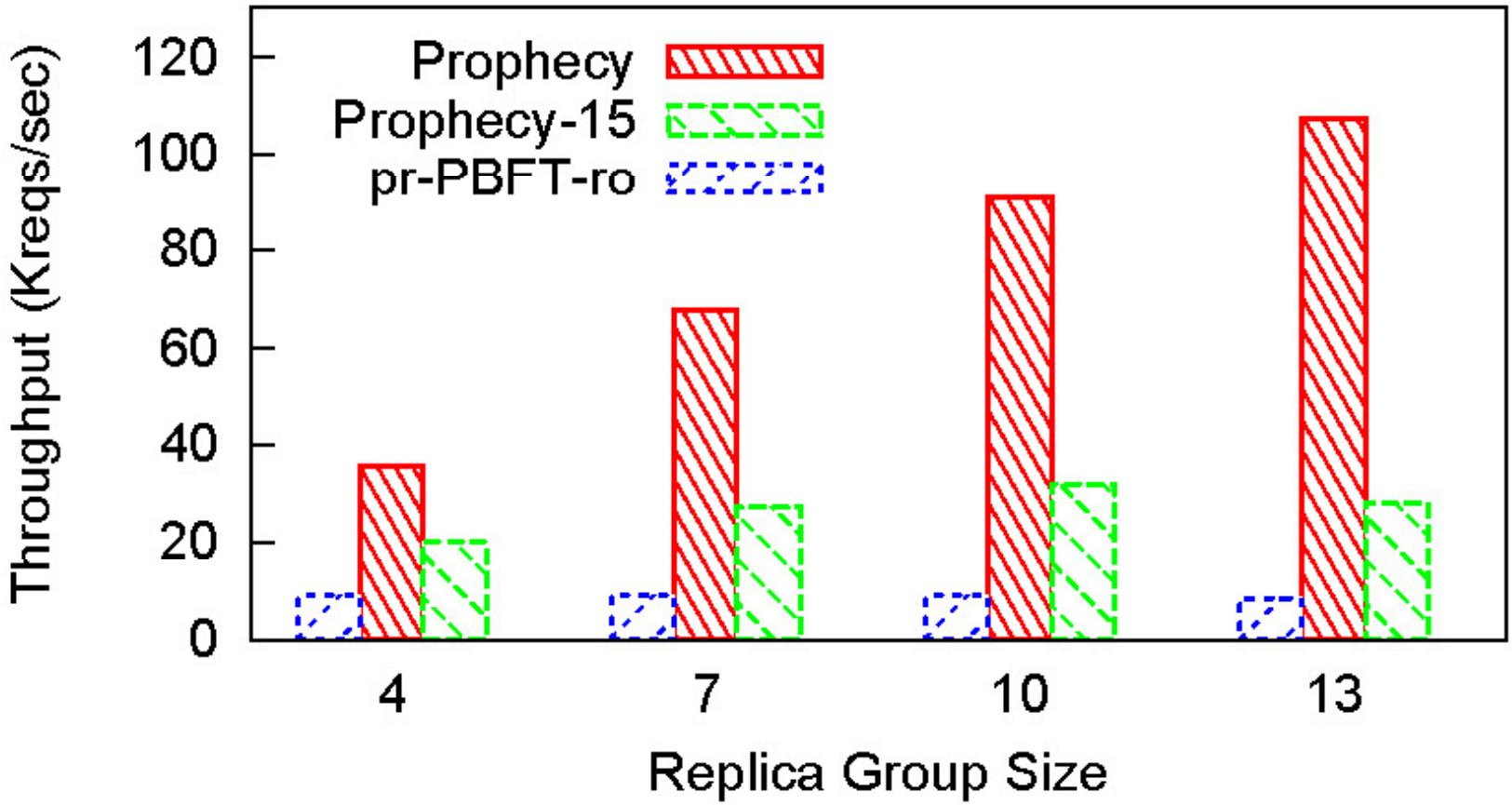
Single sketcher bottlenecks



Scaling out


Request Sketcher
Request Sketcher
Response Sketcher
Response Sketcher
Response Sketcher
Response Sketcher
Clients
Replica
Replica
Replica

Scales linearly with replicas



Summary

- Prophecy good for Internet services
  - Fast, load-balanced reads
- D-Prophecy good for traditional services
- Prophecy scales linearly while PBFT stays flat
- Limitations:
  - Read-mostly workloads (meas. study corroborates)
  - Delay-once linearizability (useful for many apps)

Thank You

Additional slides

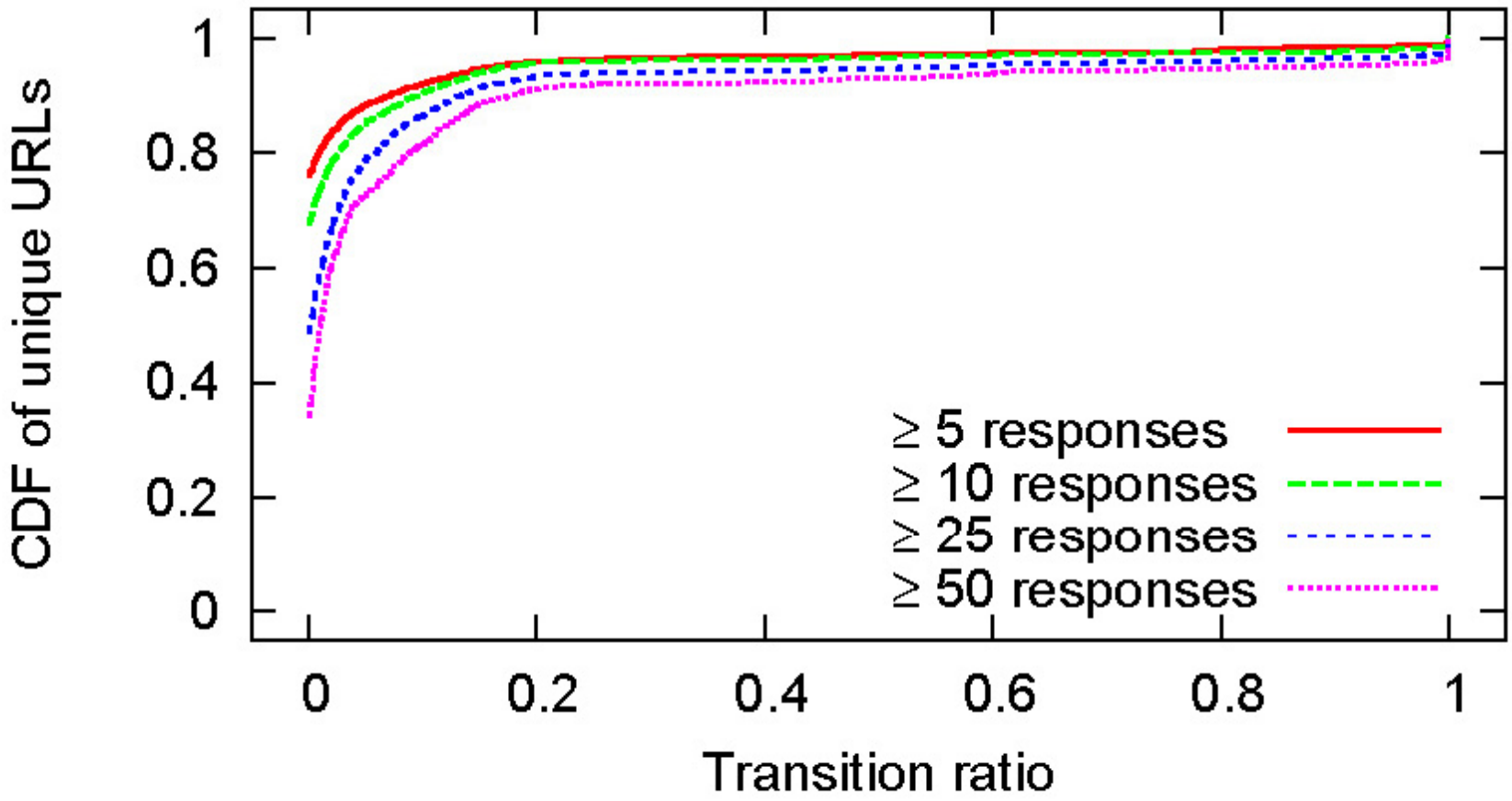
Transitions

- Prophecy good for read-mostly workloads
- Are transitions rare in practice?

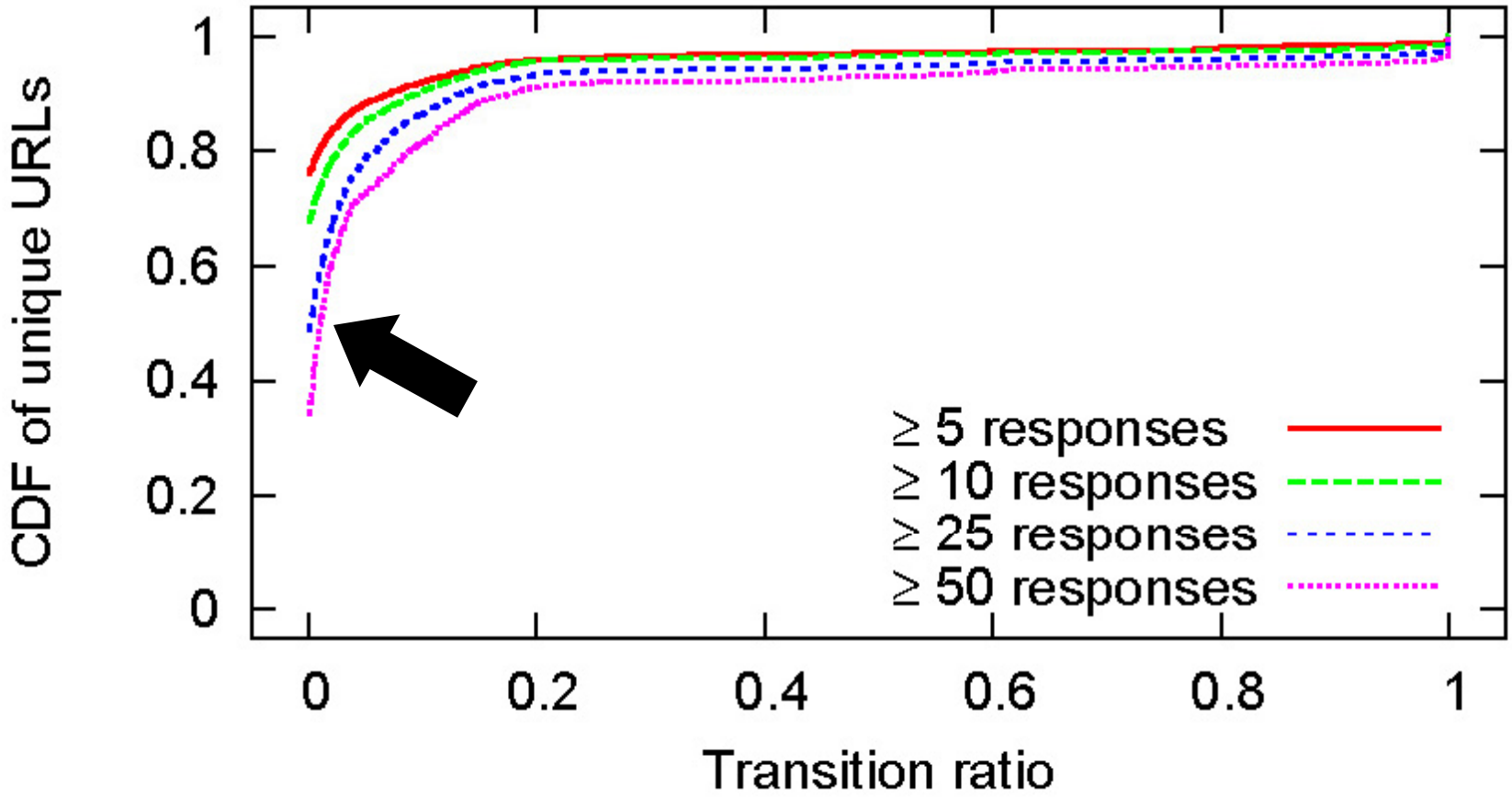
Measurement study

- Alexa top sites
- Access main page every 20 sec for 24 hrs

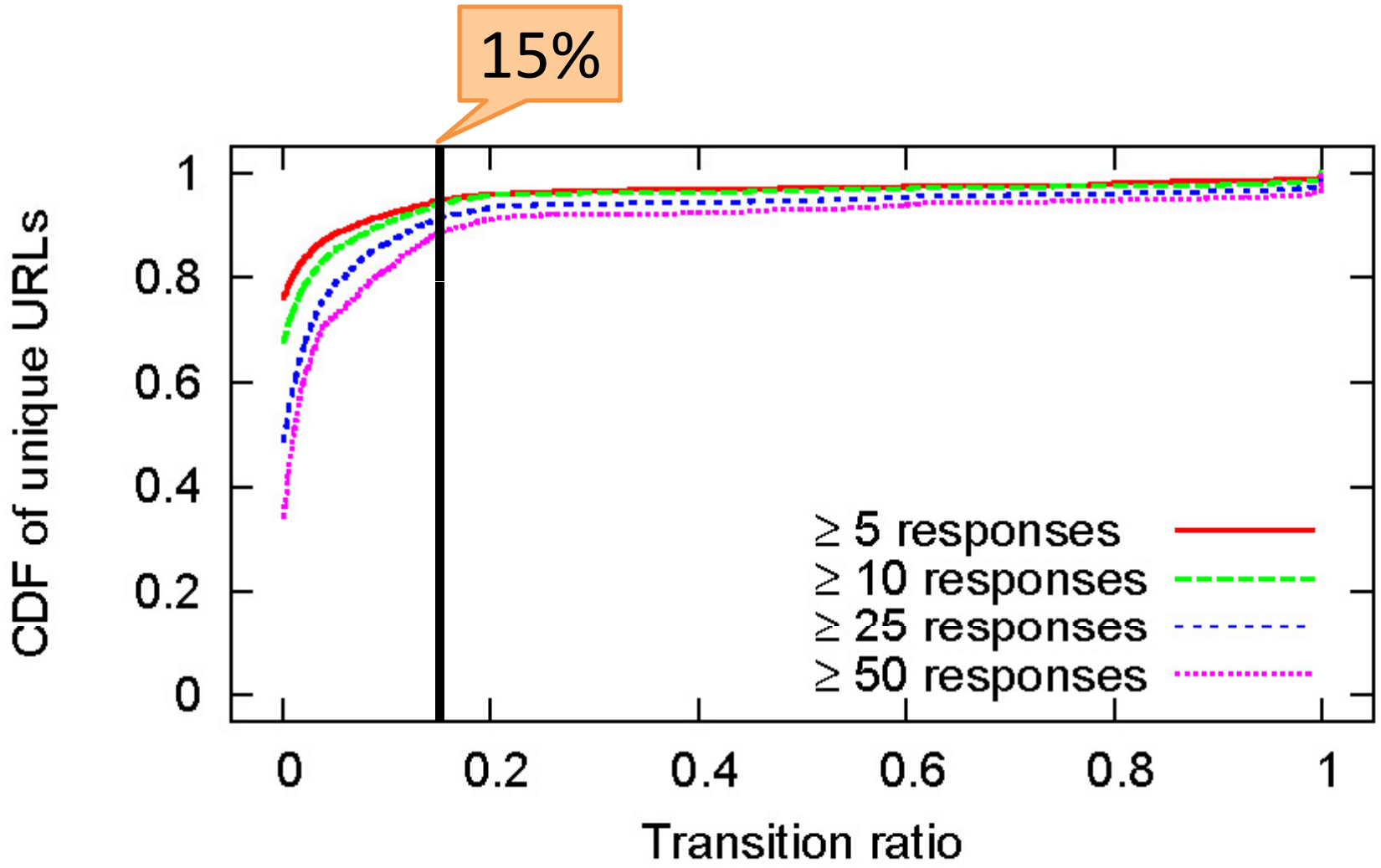
Mostly static content



Mostly static content



Mostly static content



Dynamic content

- Rabin fingerprinting on transitions
- 43% differ by single contiguous change
- Sampled 4000 of them, over half due to:
  - Load balancing directives
  - Random IDs in links, function parameters