



Prophecy: Using History for High-Throughput Fault Tolerance

Siddhartha Sen

Joint work with Wyatt Lloyd and Mike Freedman

Princeton University

Non-crash failures happen

The screenshot shows a Windows Internet Explorer browser window with the address bar displaying <http://www.techcrunch.com/2007/08/11/facebook-source-code-leaked/>. The page content includes the TechCrunch logo, navigation links, and a main article titled "Facebook Source Code Leaked" by Nik Cubrilovic, dated August 11, 2007. The article text begins with "We just received a tip that the source code for the Facebook main index page has been leaked and published on a blog called Facebook Secrets. There are at least two possible ways that the source code got out - the first is that a Facebook developer has sent it out, or the more likely option that a security hole or other method has been used on either one of the Facebook servers or in their source code repository to reveal the code. The blog that published the code only has a single post on it, so it was created exclusively to publish this code - meaning that whoever is behind this both isn't taking credit for the hole and doesn't want to be associated with it. While there is no certain way to verify if the code is actually from Facebook, by taking a quick look through the code and by double-checking some paths that have been referenced, we can say with some certainty that this seems to be both real and also a recent version of the main Facebook page." A blue Facebook logo is visible on the right side of the article text. To the right of the article is a "Comments" section with a comment from Michael Vu dated August 11th, 2007 at 7:55 pm PDT, which reads: "This is some crazy news...Bad news for Facebook. I'm sure this will be all over the place soon." Below the comment is a quote: "It is for these reasons that it is often claimed that open source software is more security than closed source software, since there are many more eyes auditing the code and obfuscation can't be used as a security measure." The browser's taskbar at the bottom shows the address bar with <http://www.techcrunch.com/crunchcam/>, the Internet icon, and a 100% zoom level.

Non-crash failures happen

Model as Byzantine
(malicious)

Facebook
by Nik Cubrilovic on August 11, 2007 265 Comments

We just received a tip that the source code for the **Facebook** main index page has been leaked and published on a blog called **Facebook Secrets**. There are at least two possible ways that the source code got out - the first is that a Facebook developer has sent it out, or the more likely option that a security hole or other method has been used on either one of the Facebook servers or in their source code repository to reveal the code. The blog that published the code only has a single post on it, so it was created exclusively to publish this code - meaning that whoever is behind this both isn't taking credit for the hole and doesn't want to be associated with it. While there is no certain way to verify if the code is actually from Facebook, by taking a quick look through the code and by double-checking some paths that have been referenced, we can say with some certainty that this seems to be both real and also a recent version of the main Facebook page.

Comments

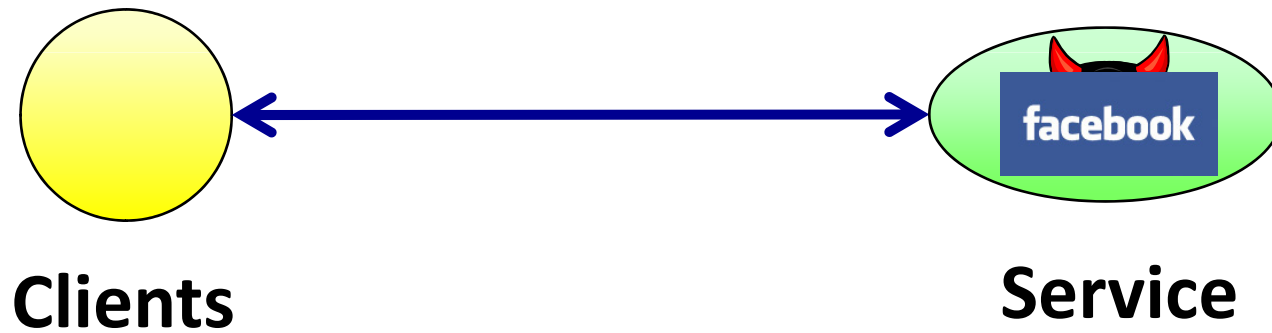
Michael Vu - August 11th, 2007 at 7:55 pm PDT

This is some crazy news...Bad news for Facebook. I'm sure this will be all over the place soon.

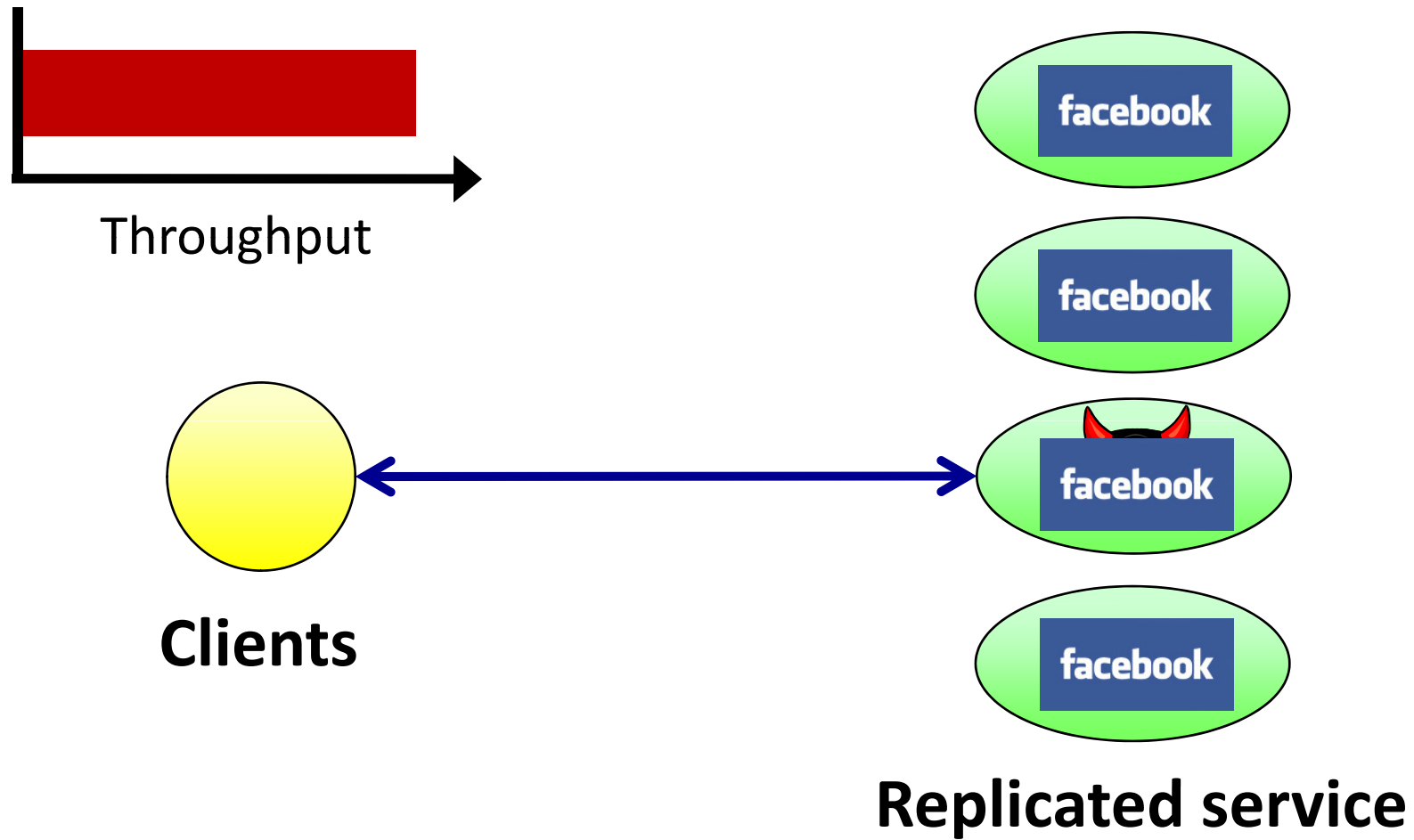
Thanks for the tip TechCrunch, great post Nik..especially the snippet below:

"It is for these reasons that it is often claimed that open source software is more security than closed source software, since there are many more eyes auditing the code and obfuscation can't be used as a security measure."

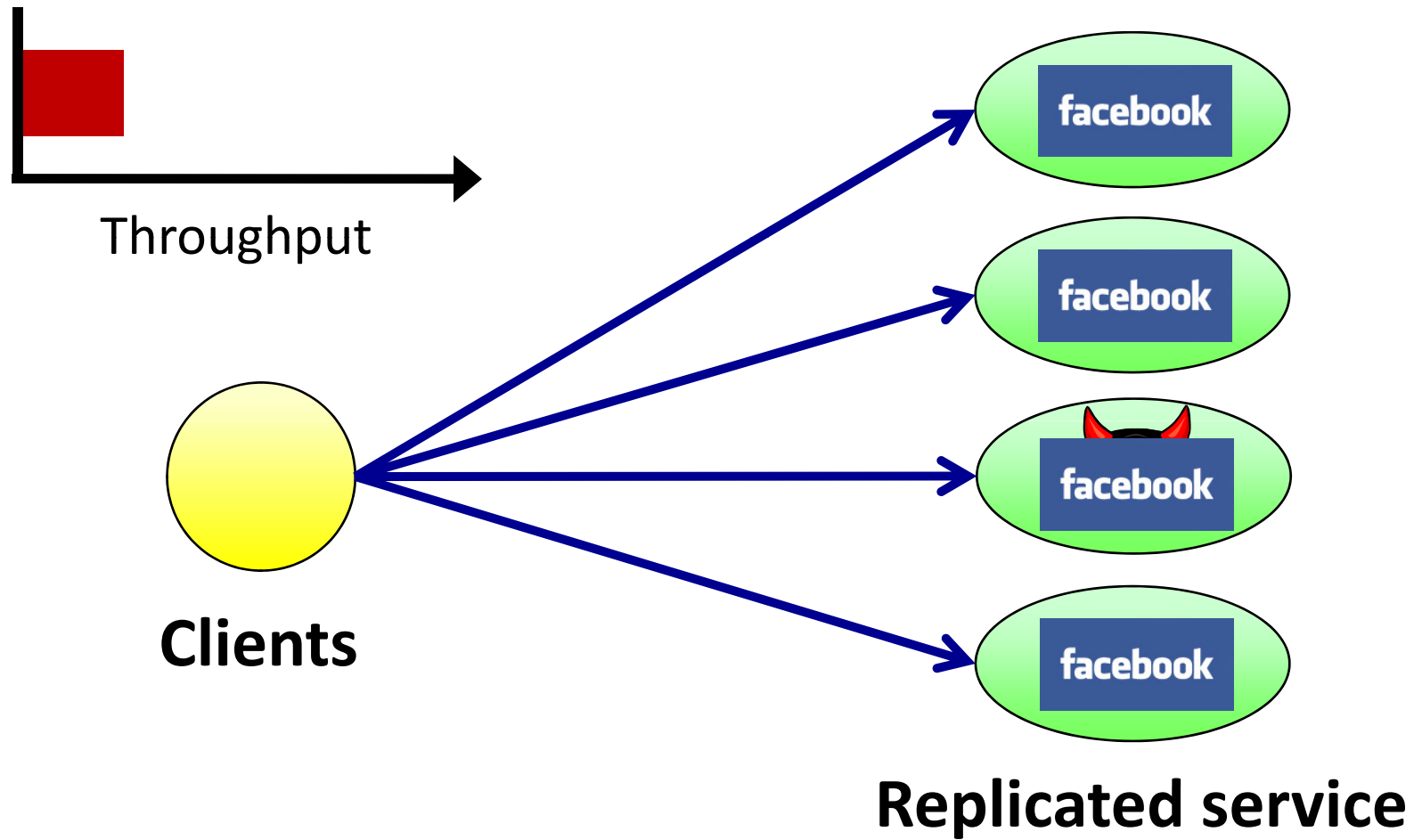
Mask Byzantine faults



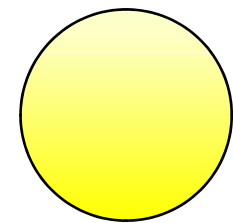
Mask Byzantine faults



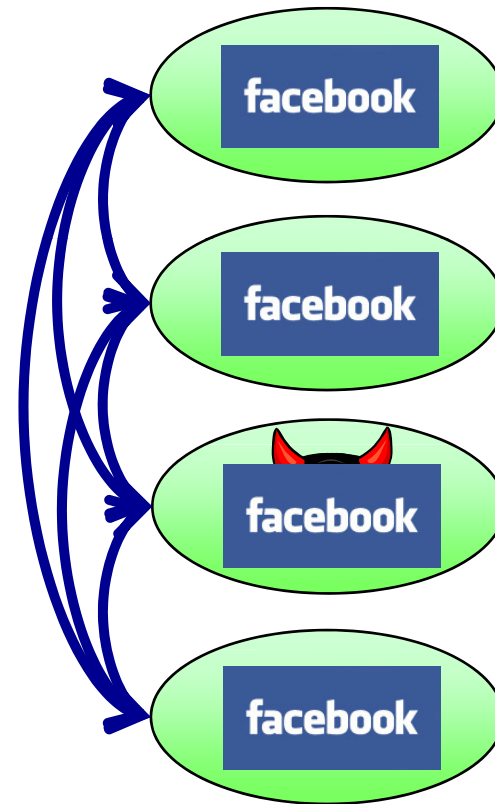
Mask Byzantine faults



Mask Byzantine faults

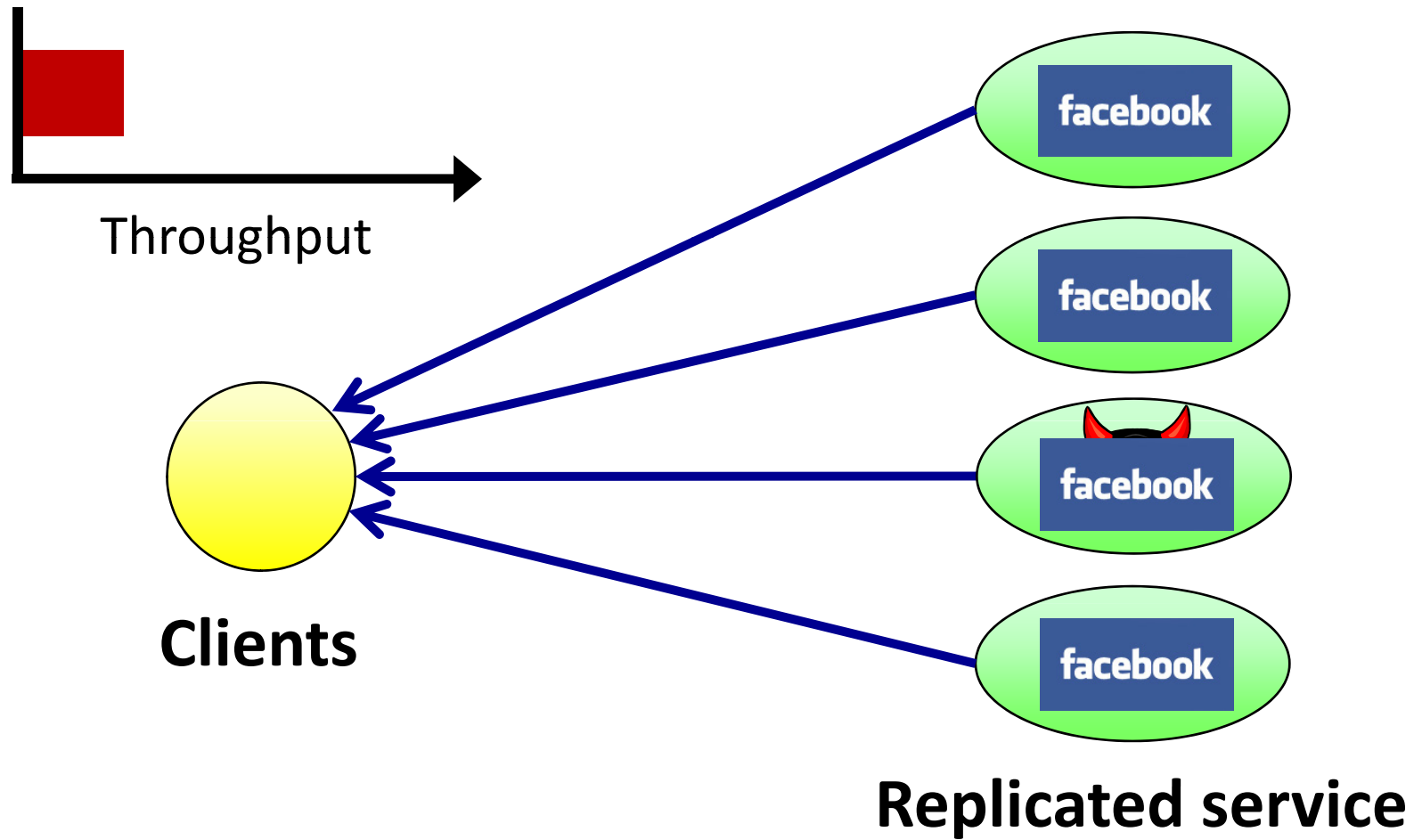


Clients

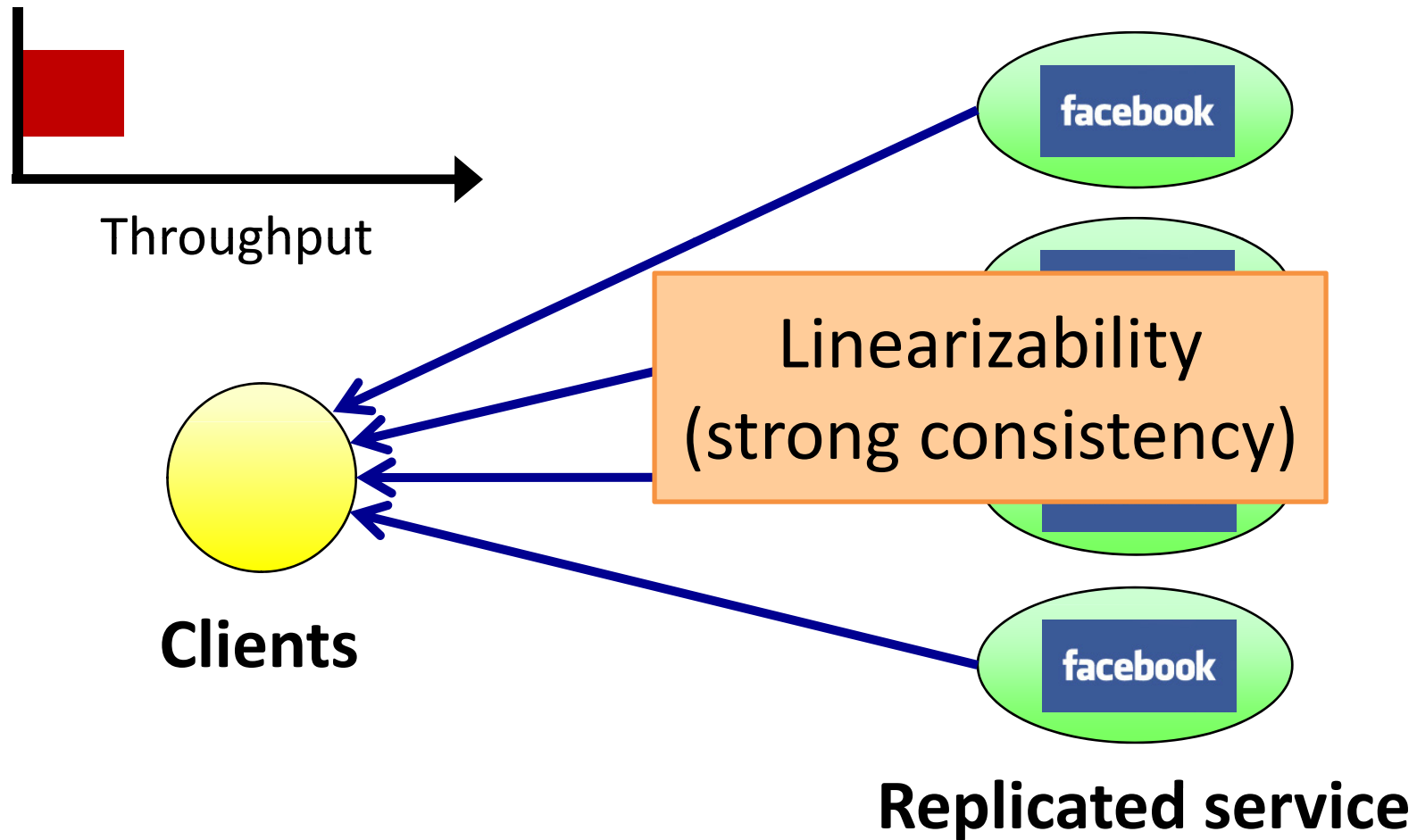


Replicated service

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

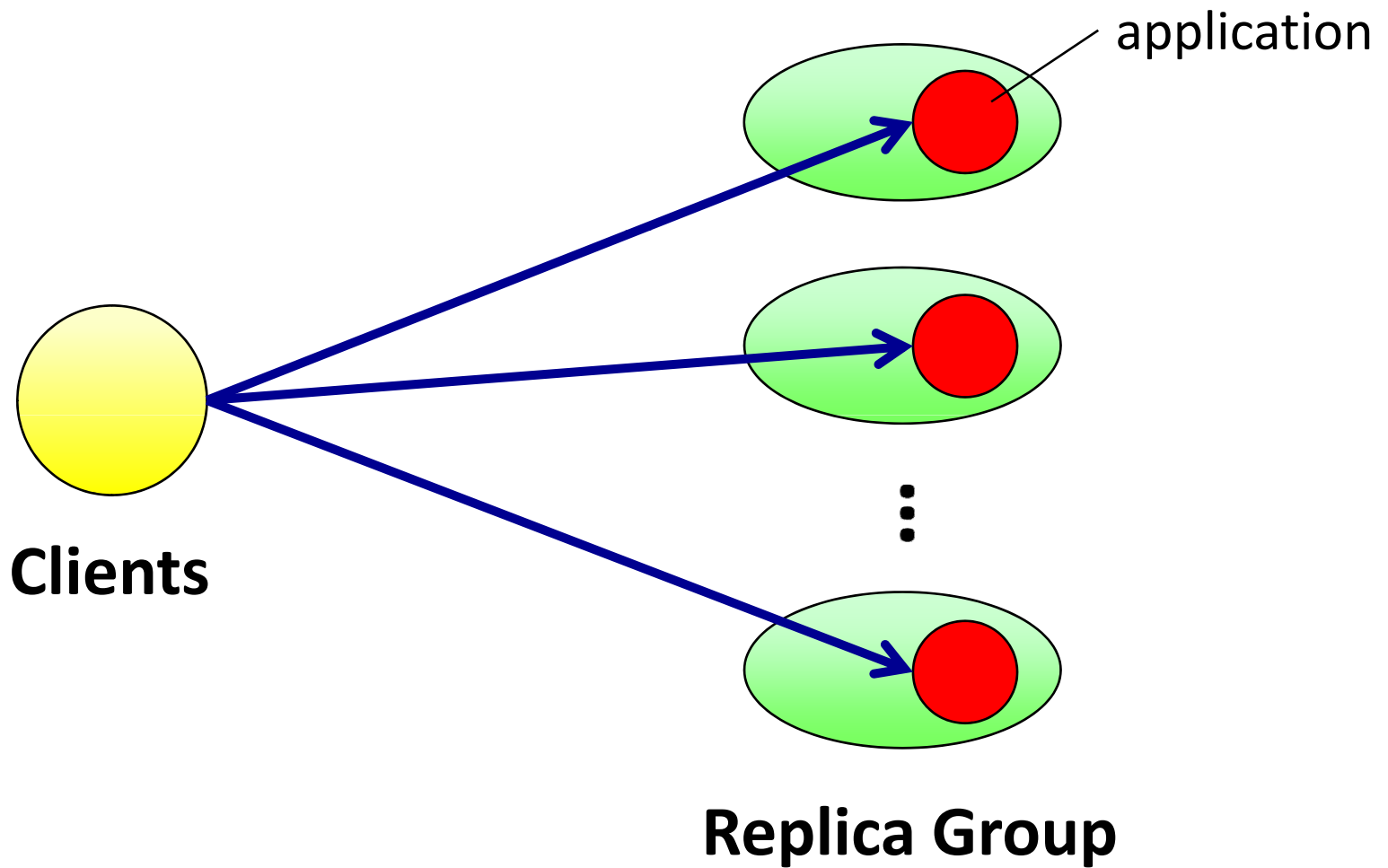
D-Prophecy

- Modifies clients

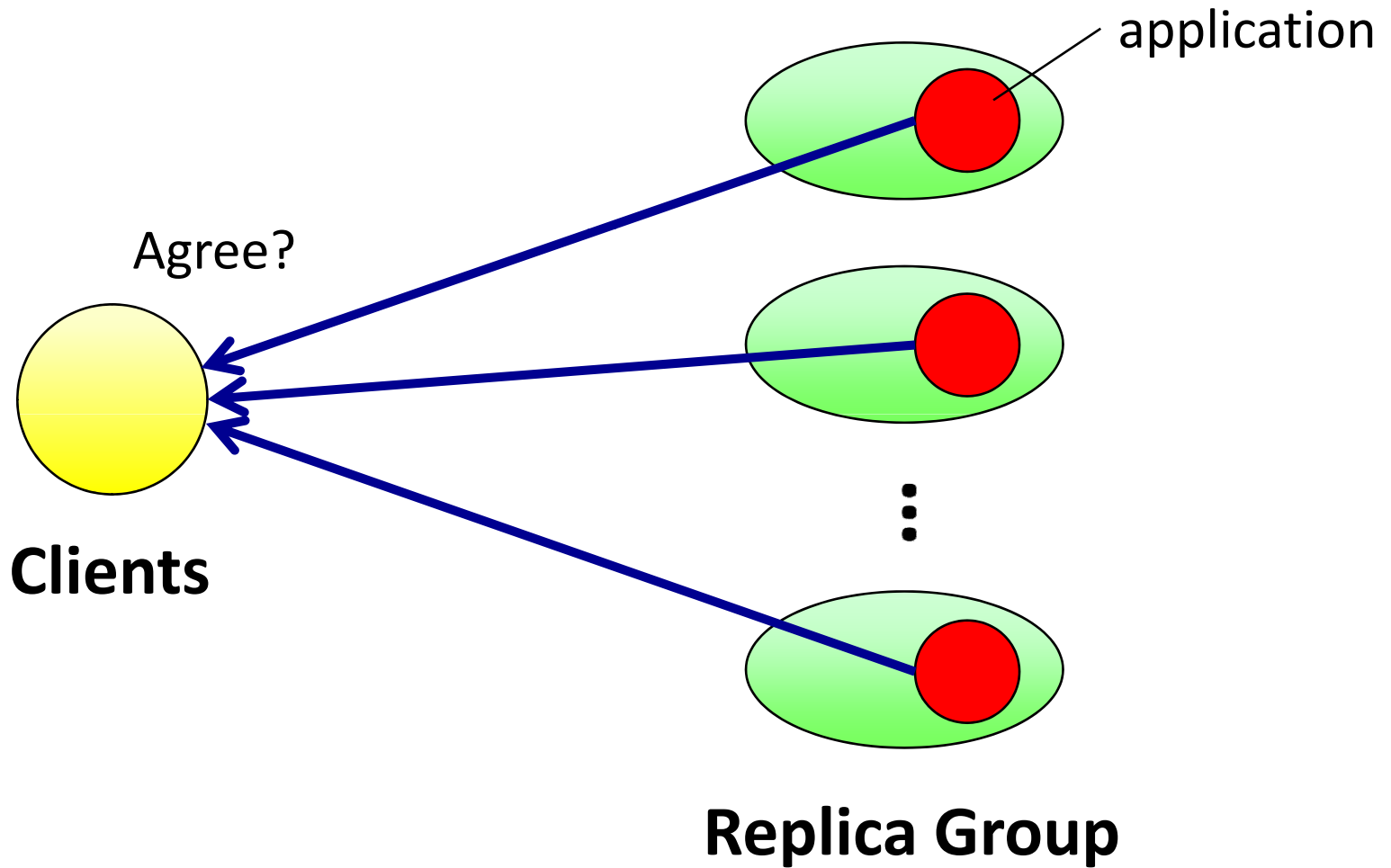
Prophecy

- Long-lived sessions

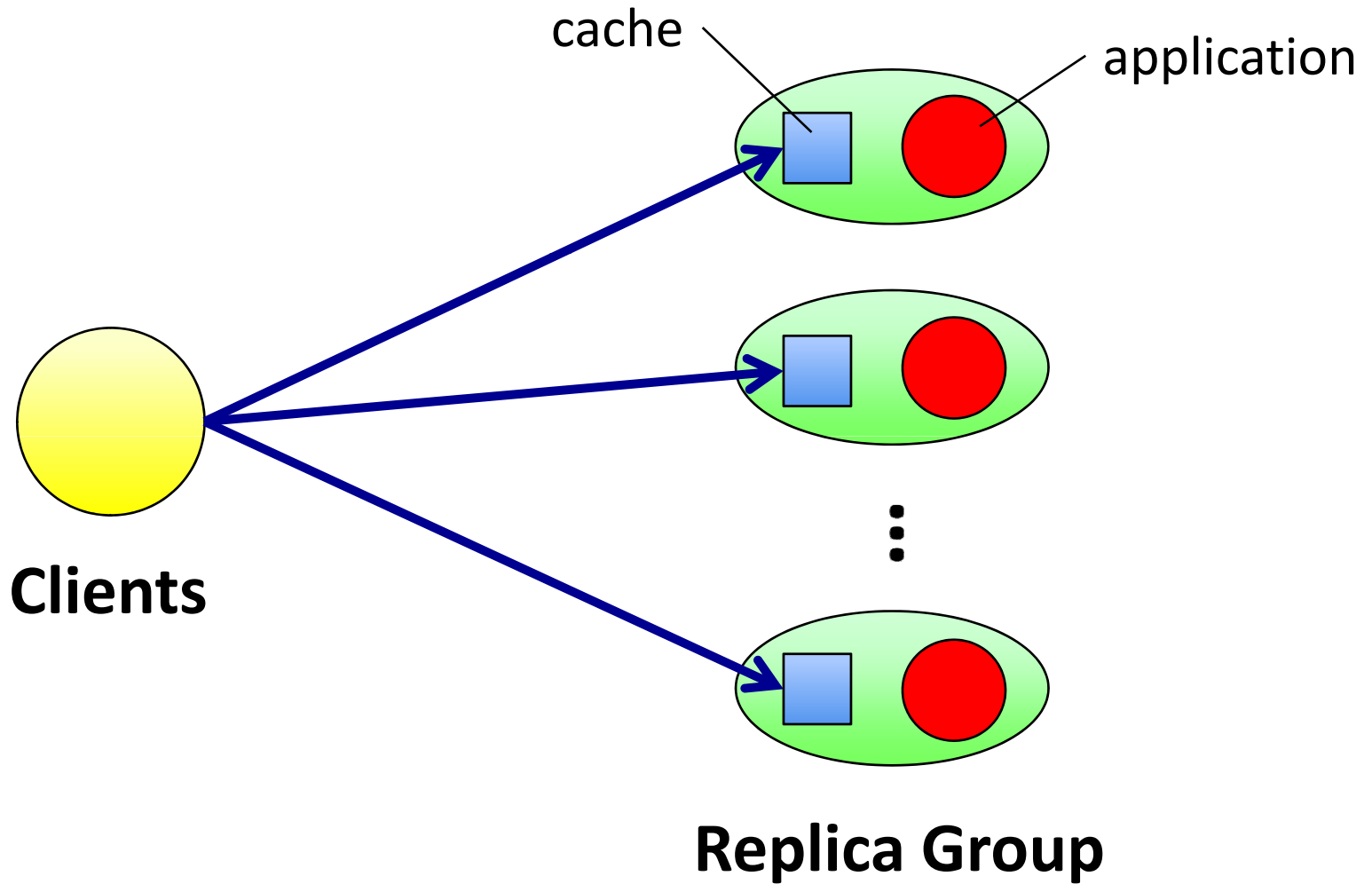
Traditional BFT reads



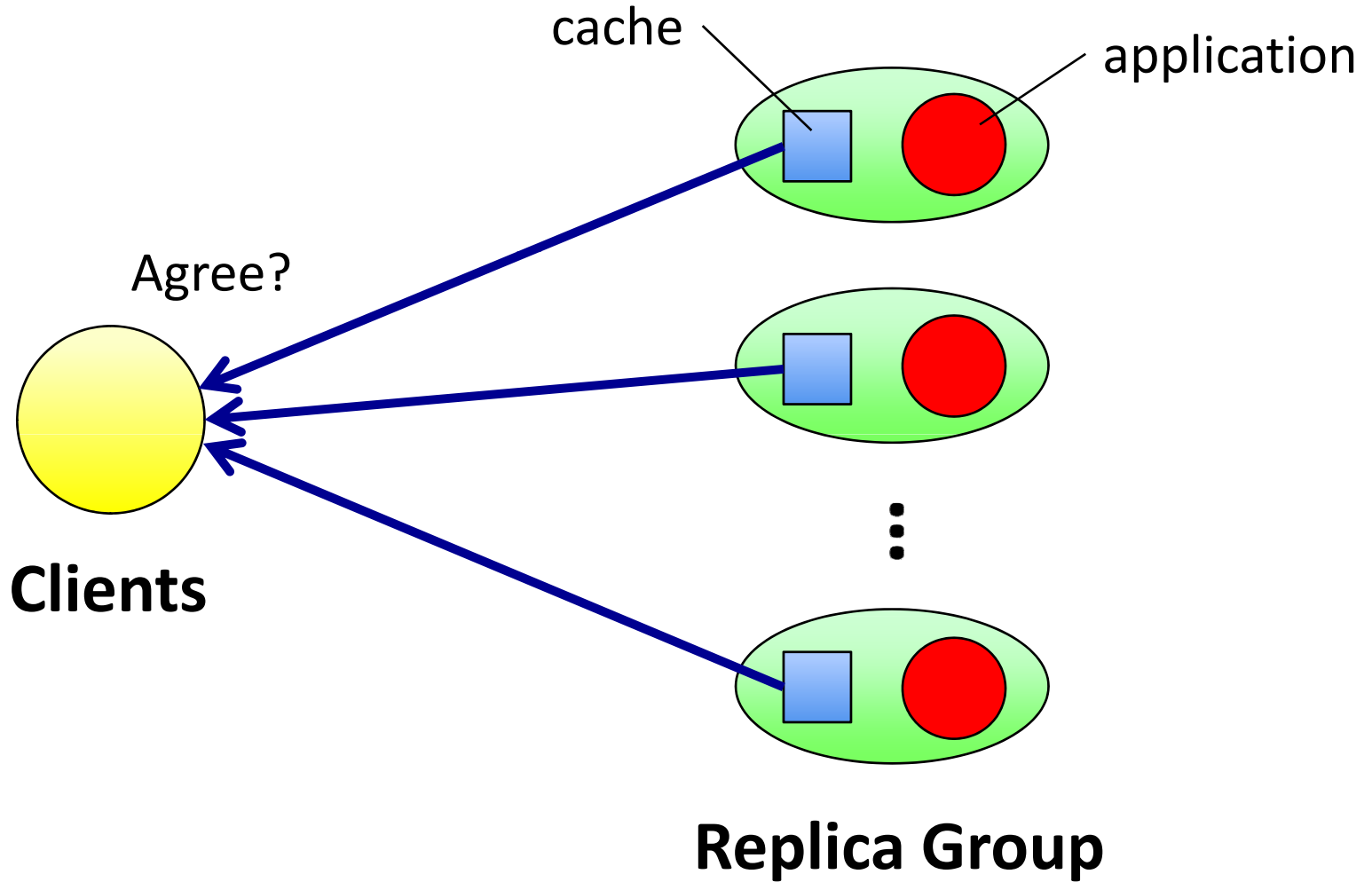
Traditional BFT reads



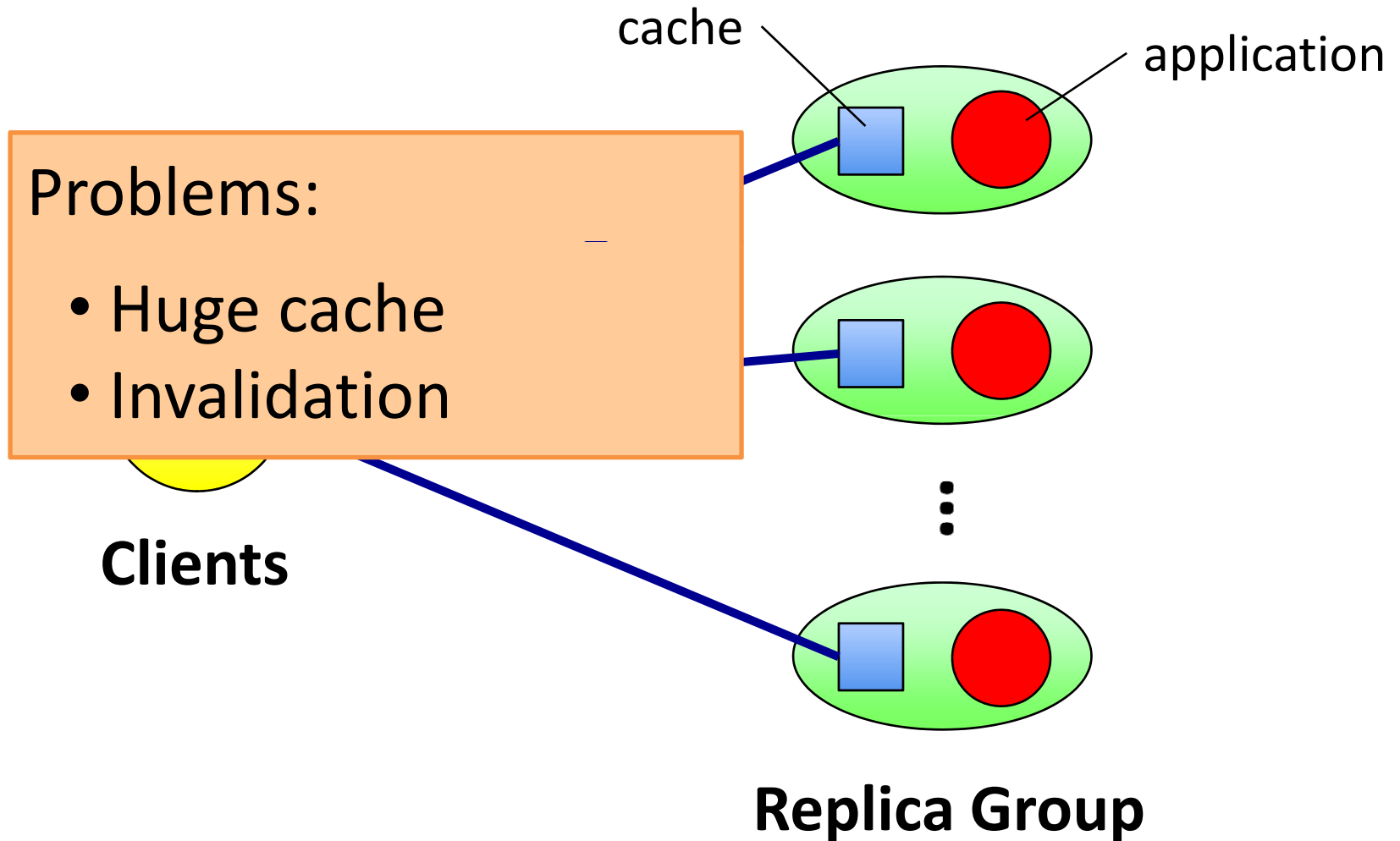
A cache solution



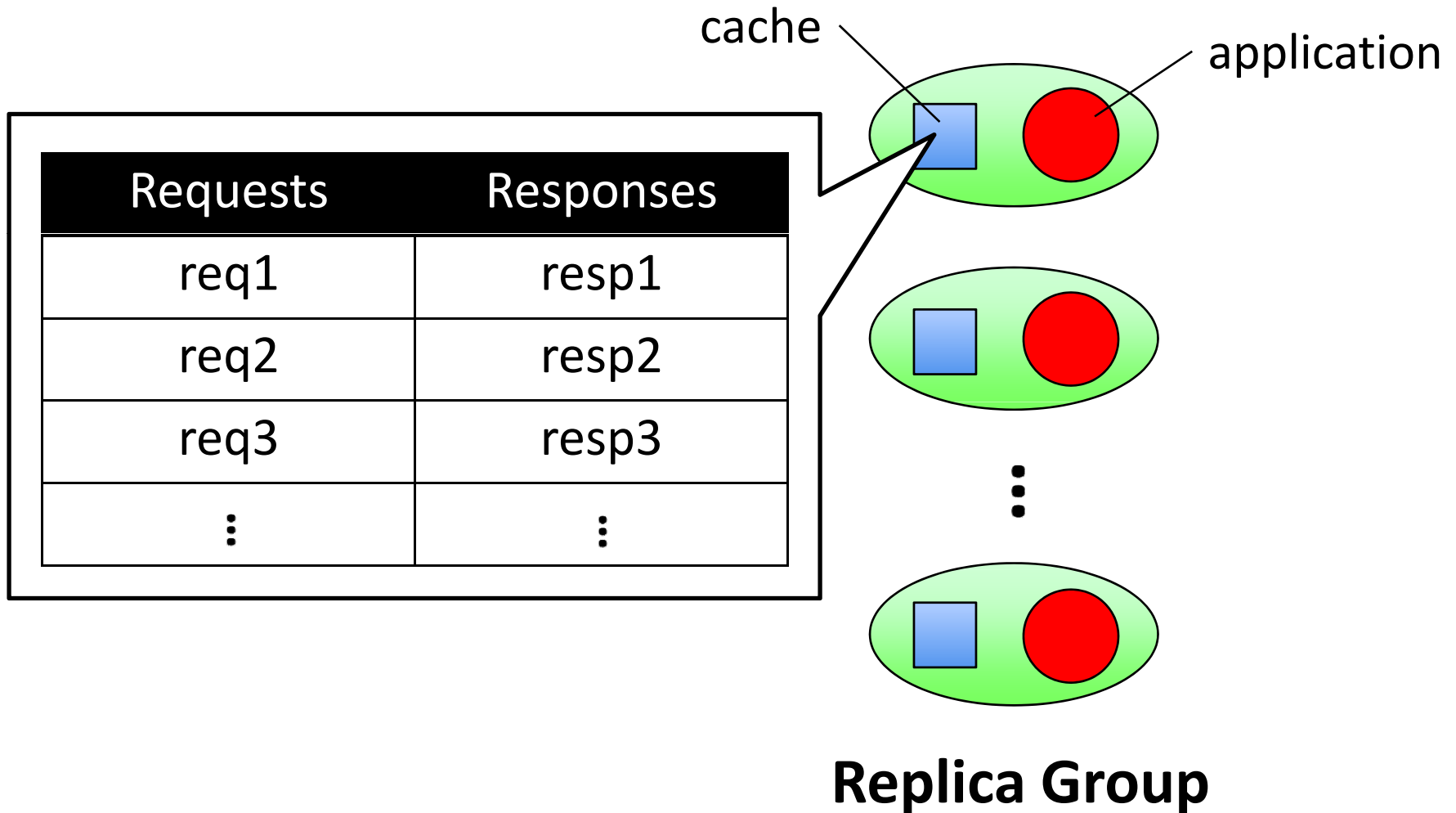
A cache solution



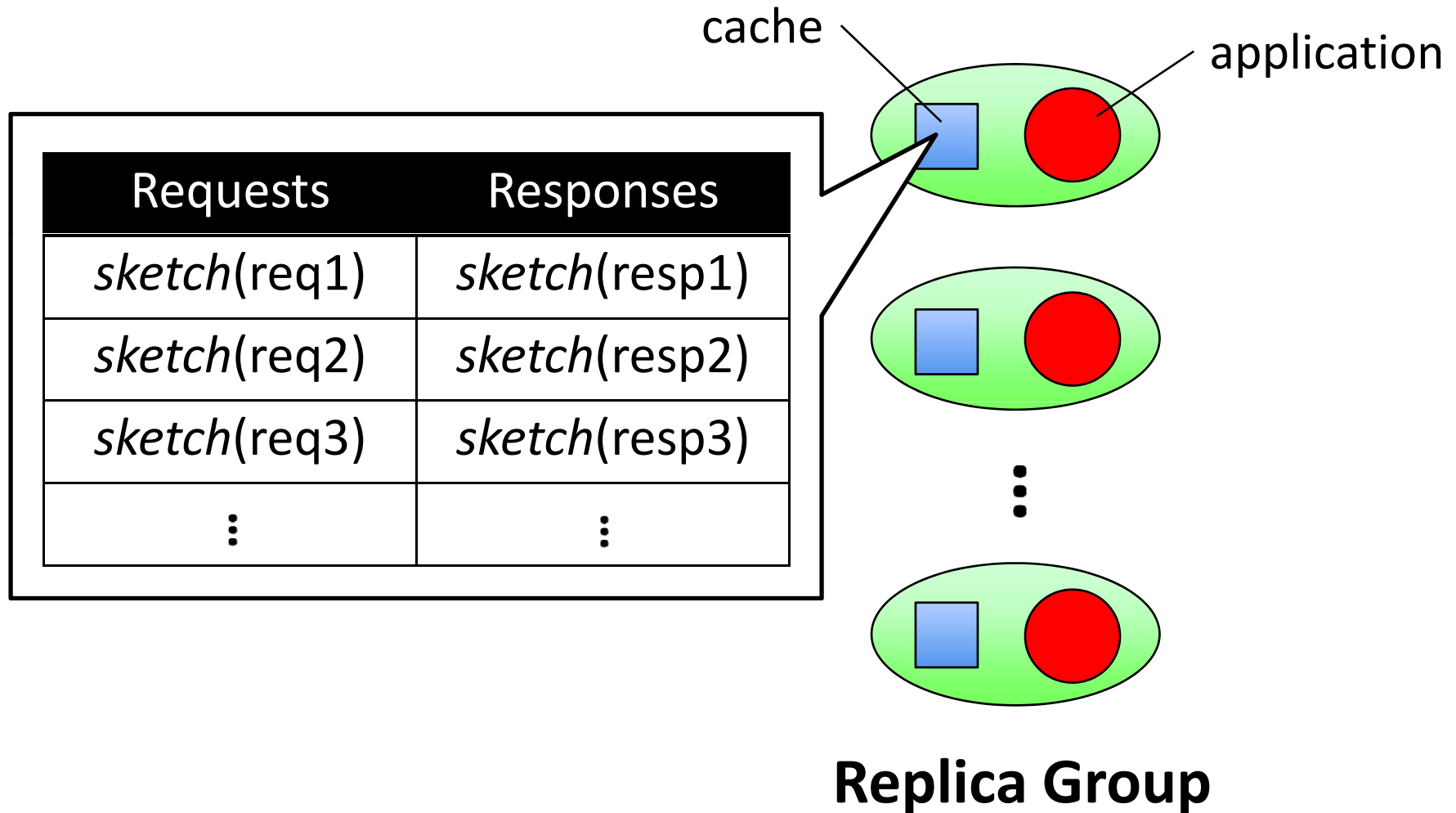
A cache solution



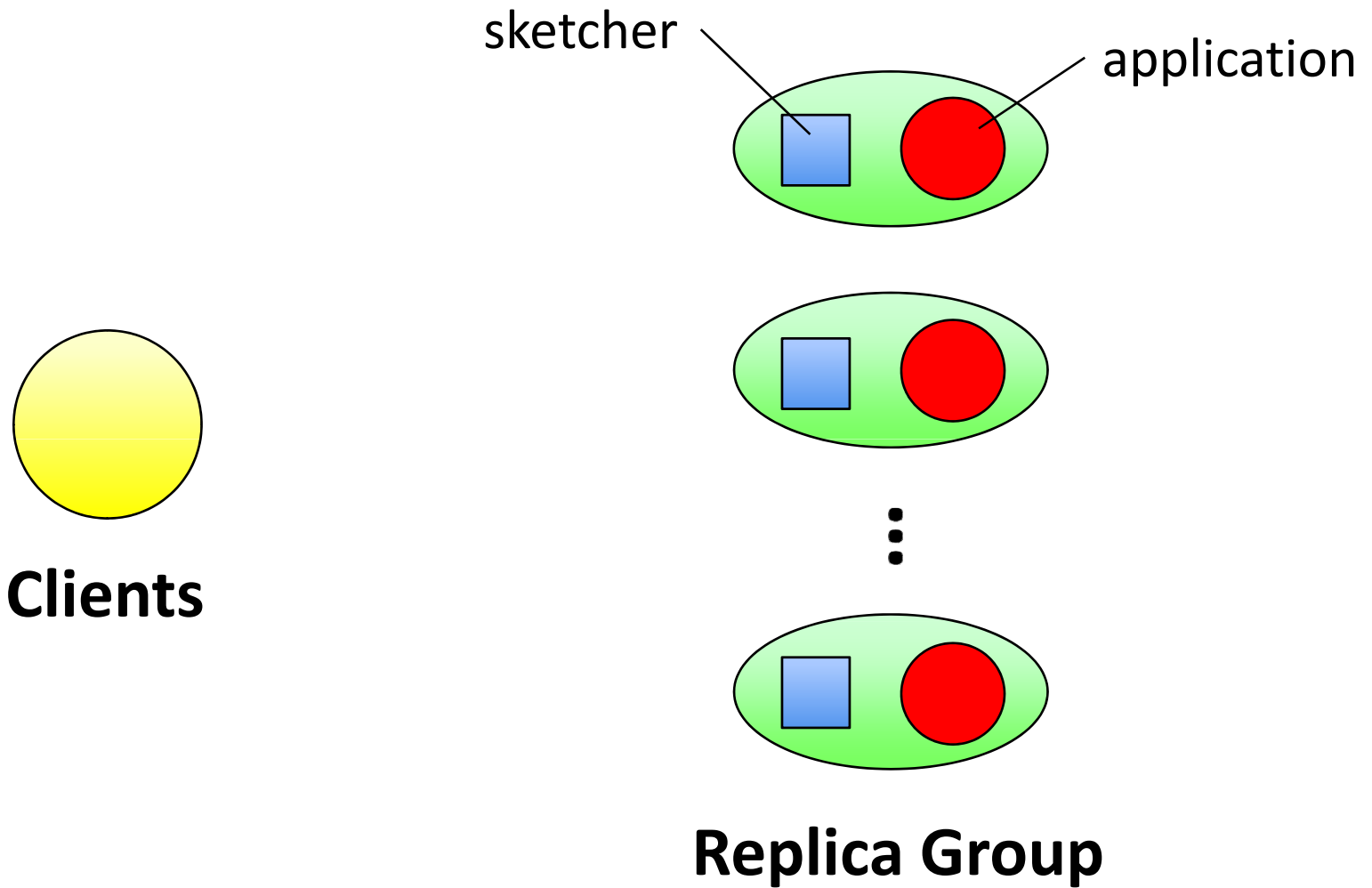
A compact cache



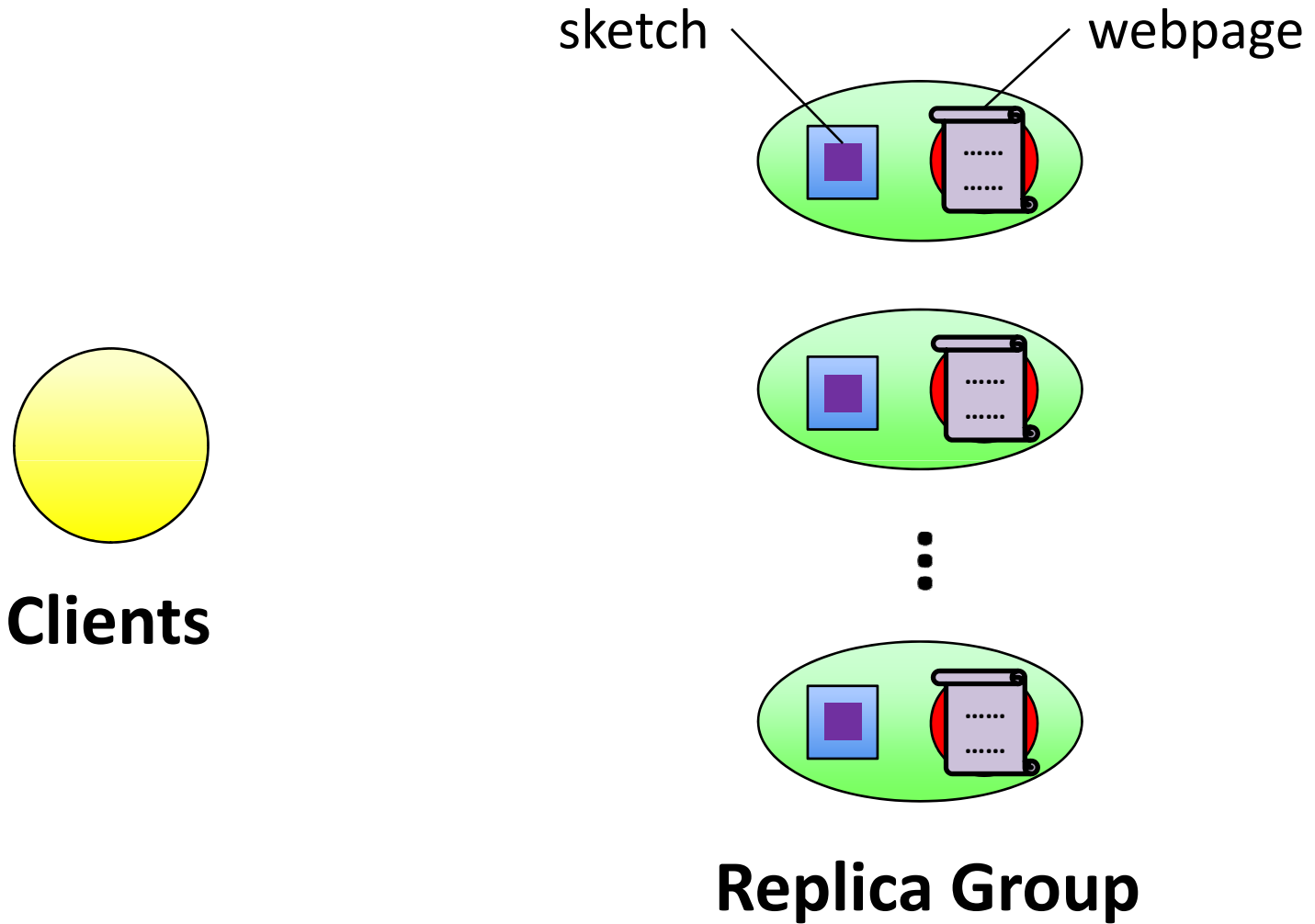
A compact cache



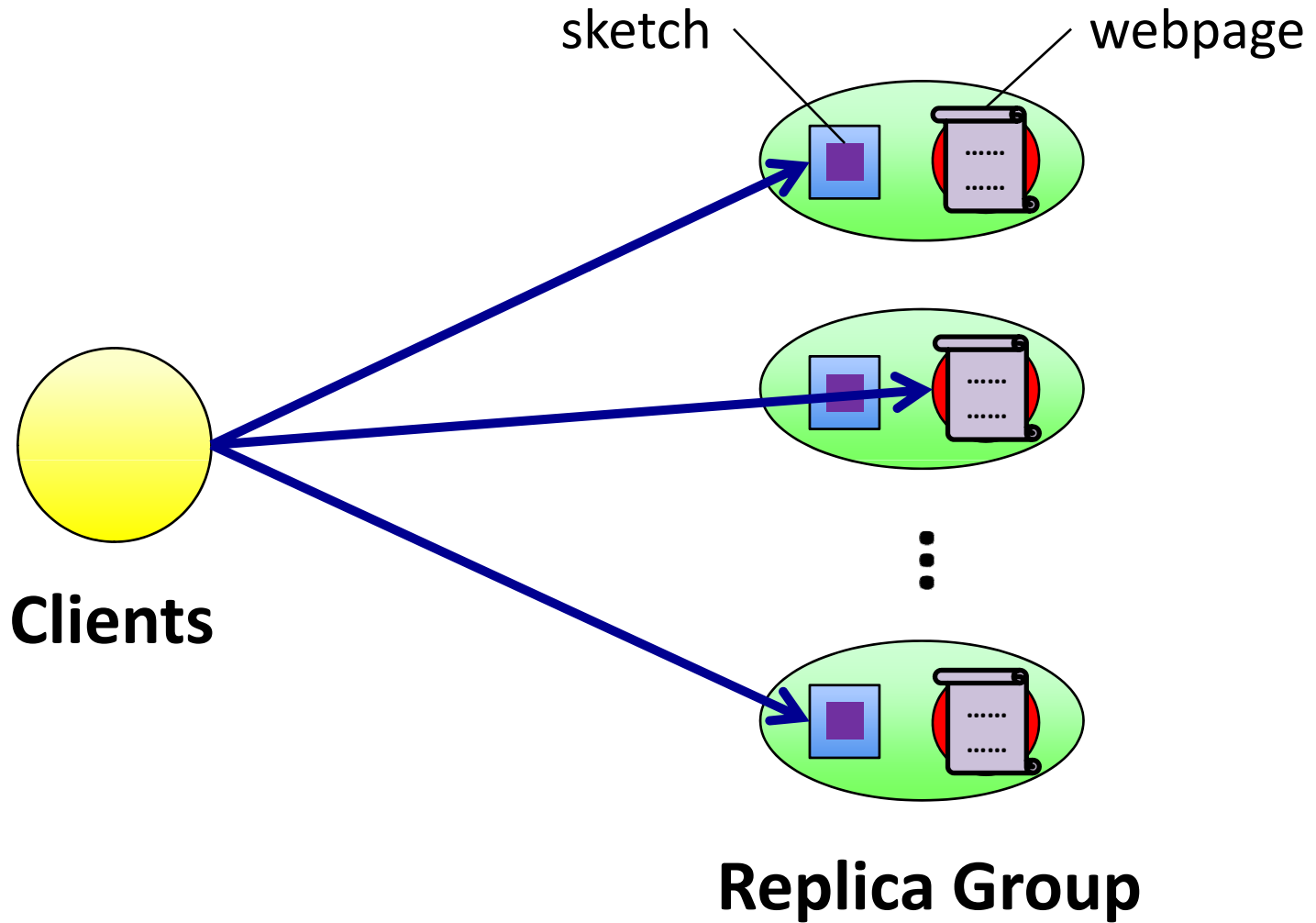
A sketcher



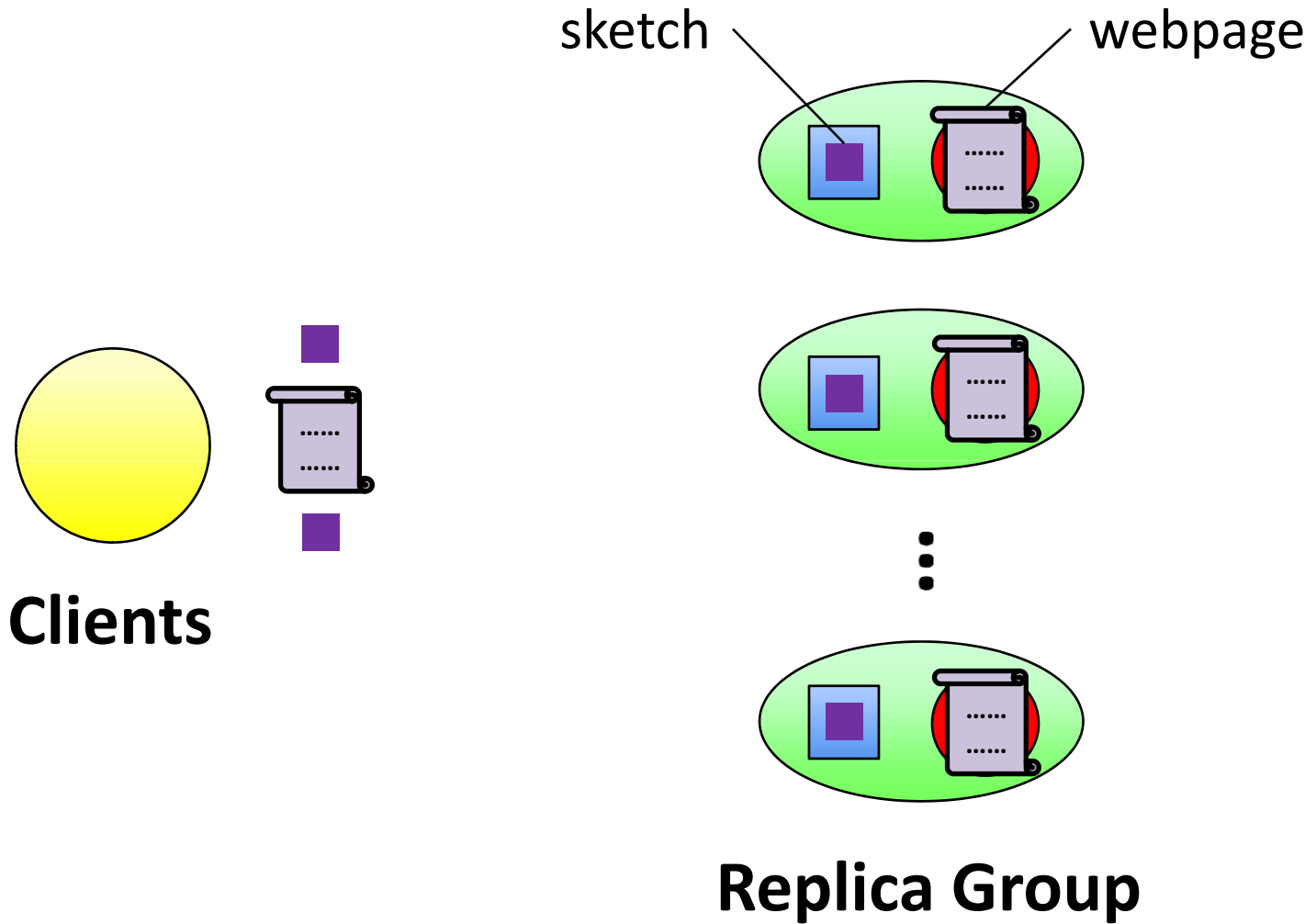
Executing a read



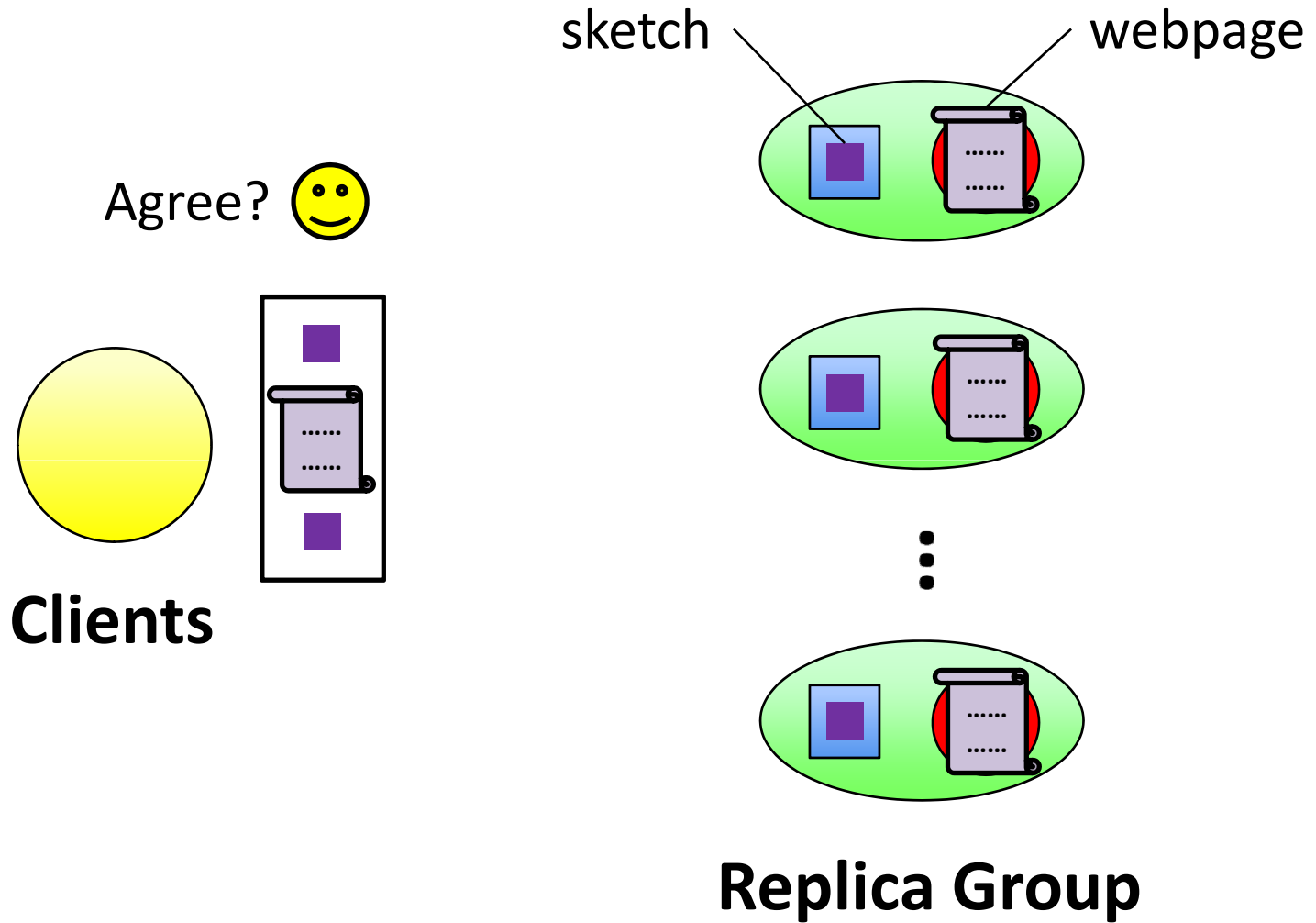
Executing a read



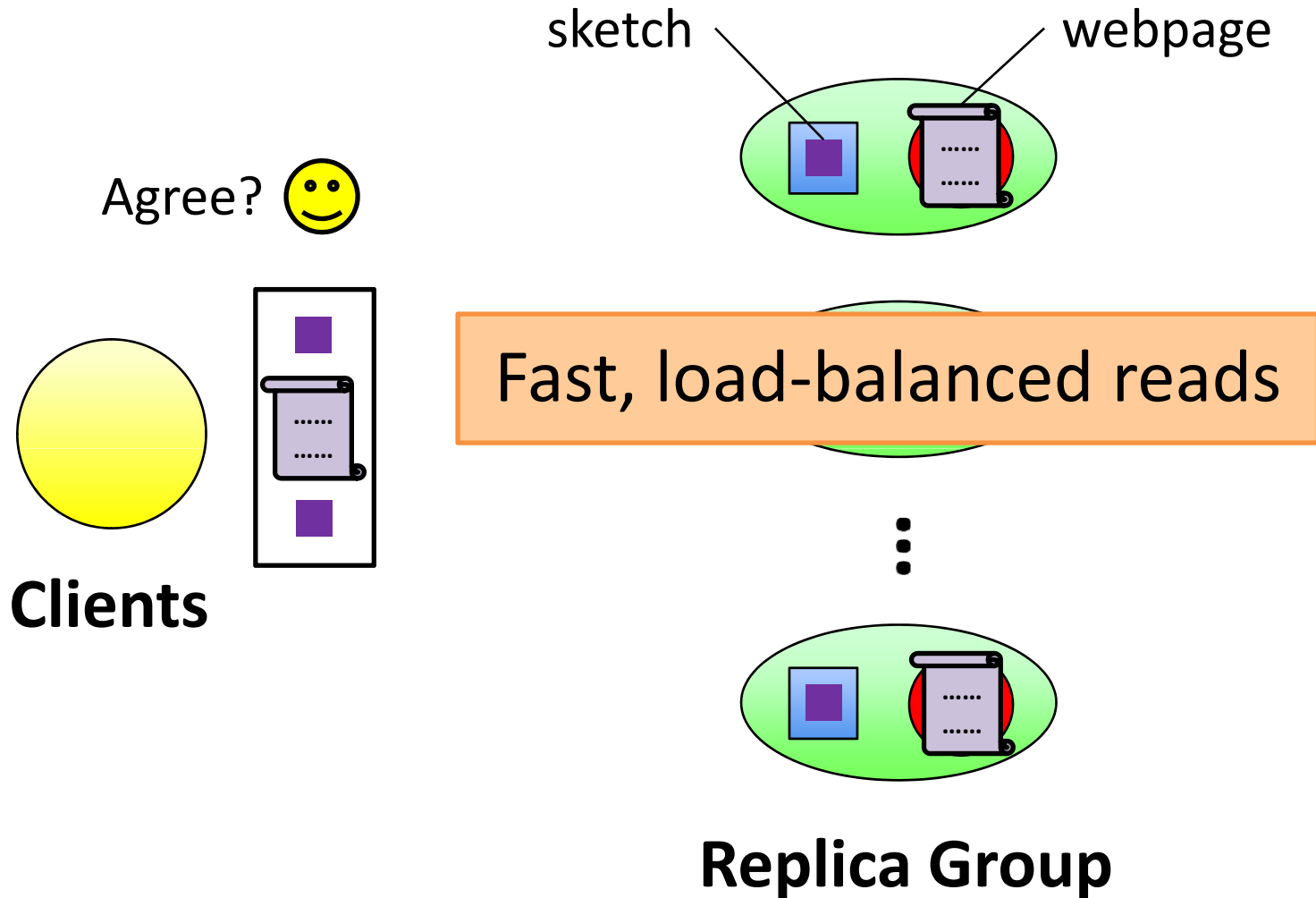
Executing a read



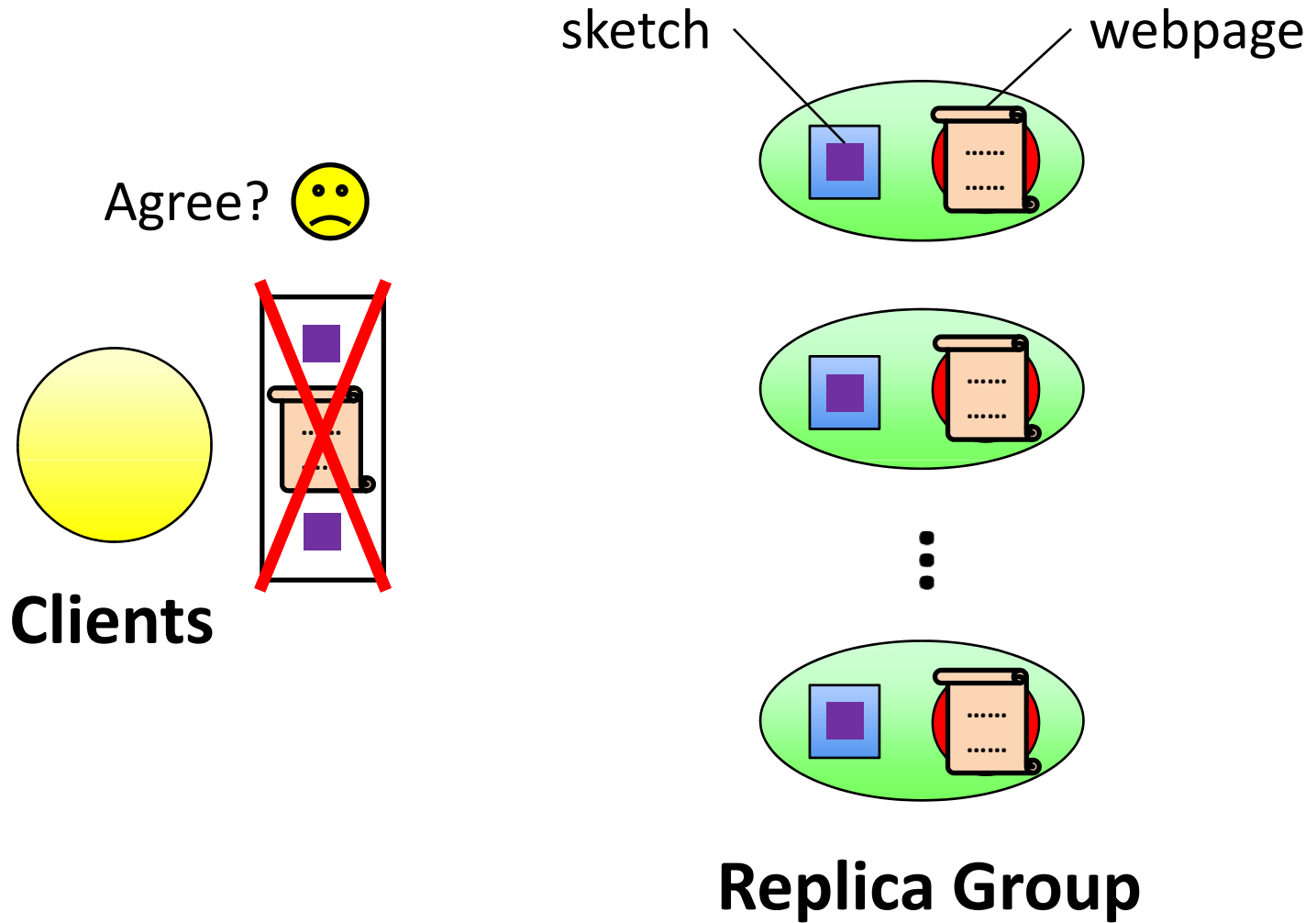
Executing a read



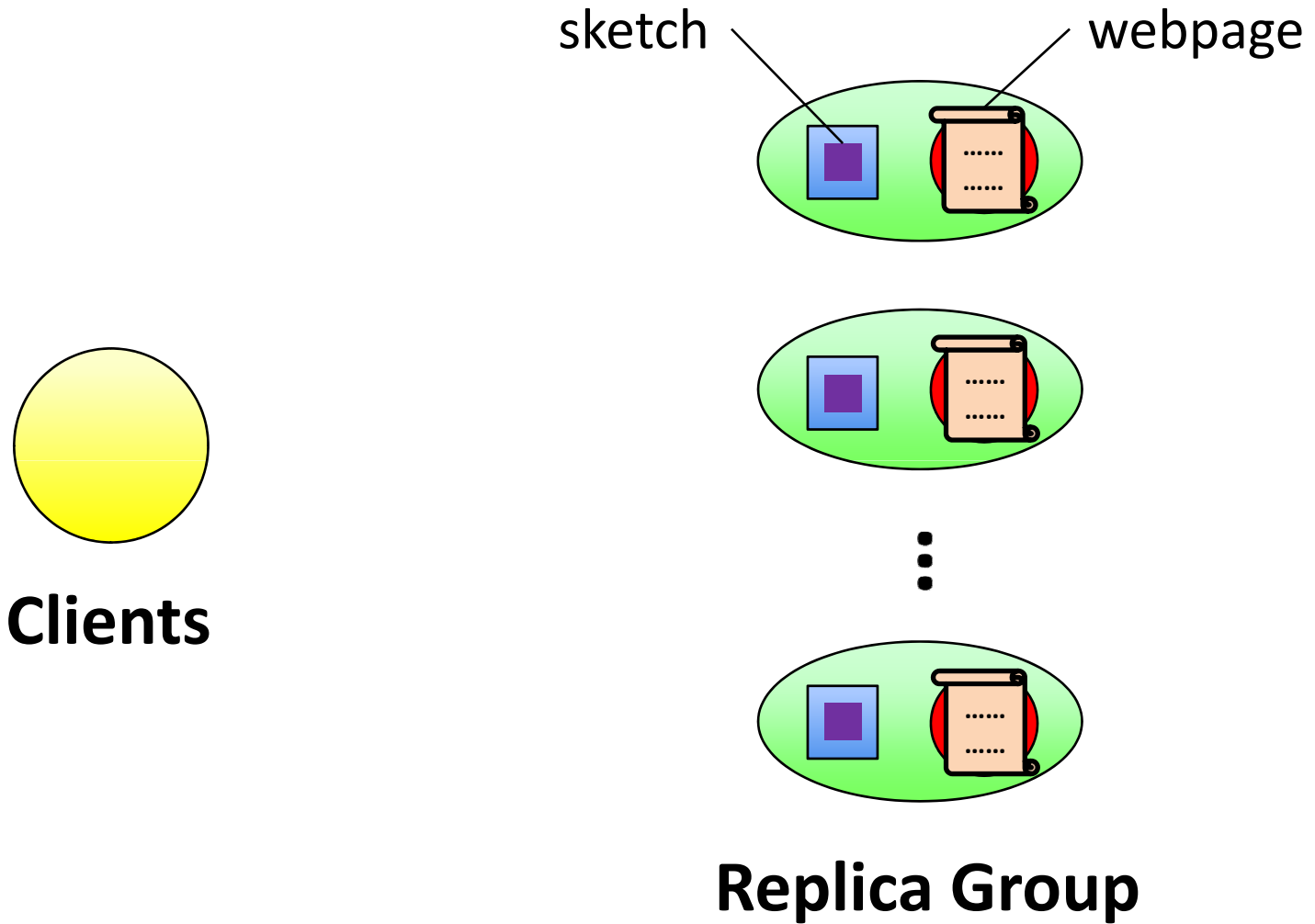
Executing a read



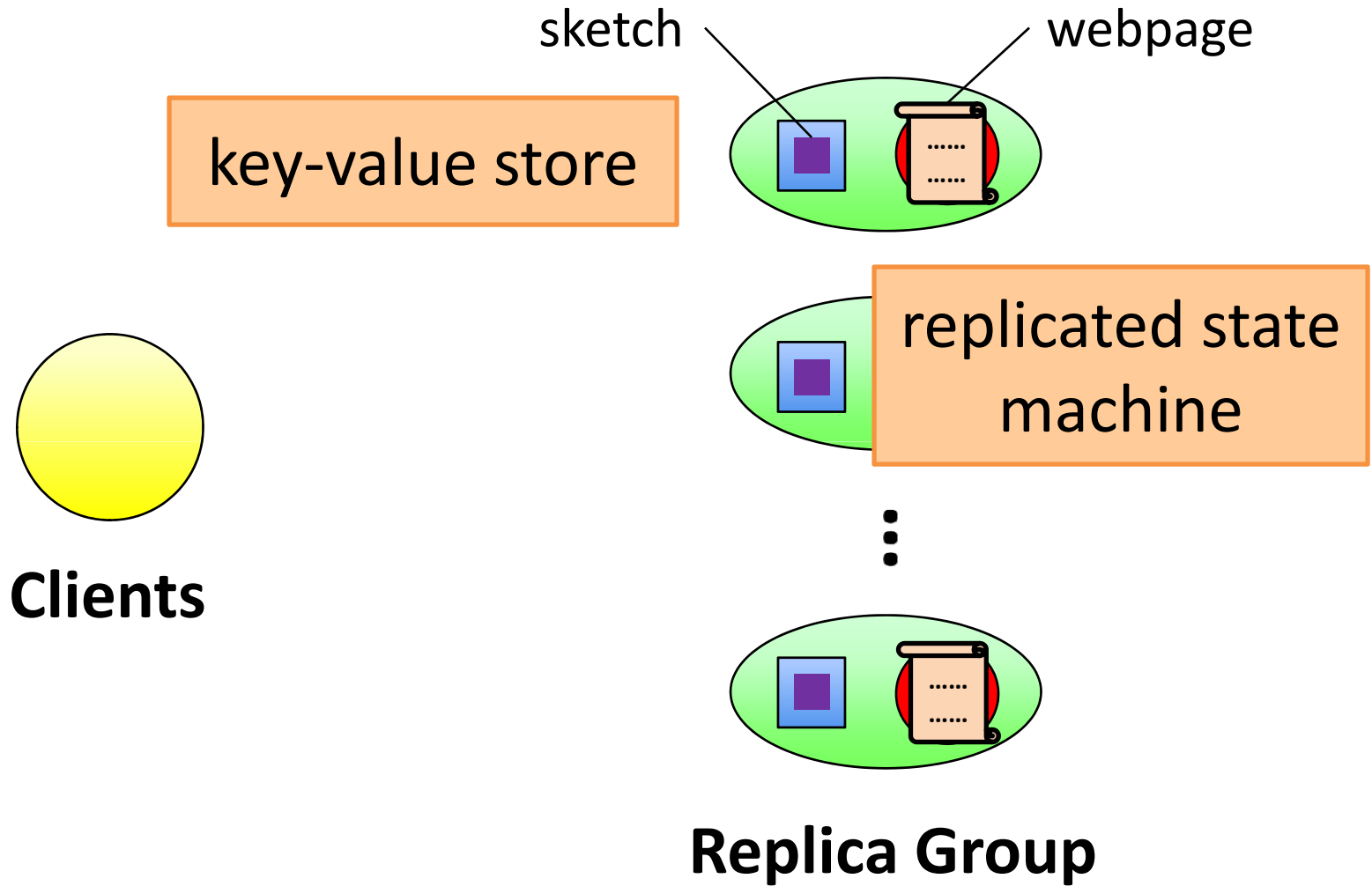
Executing a read



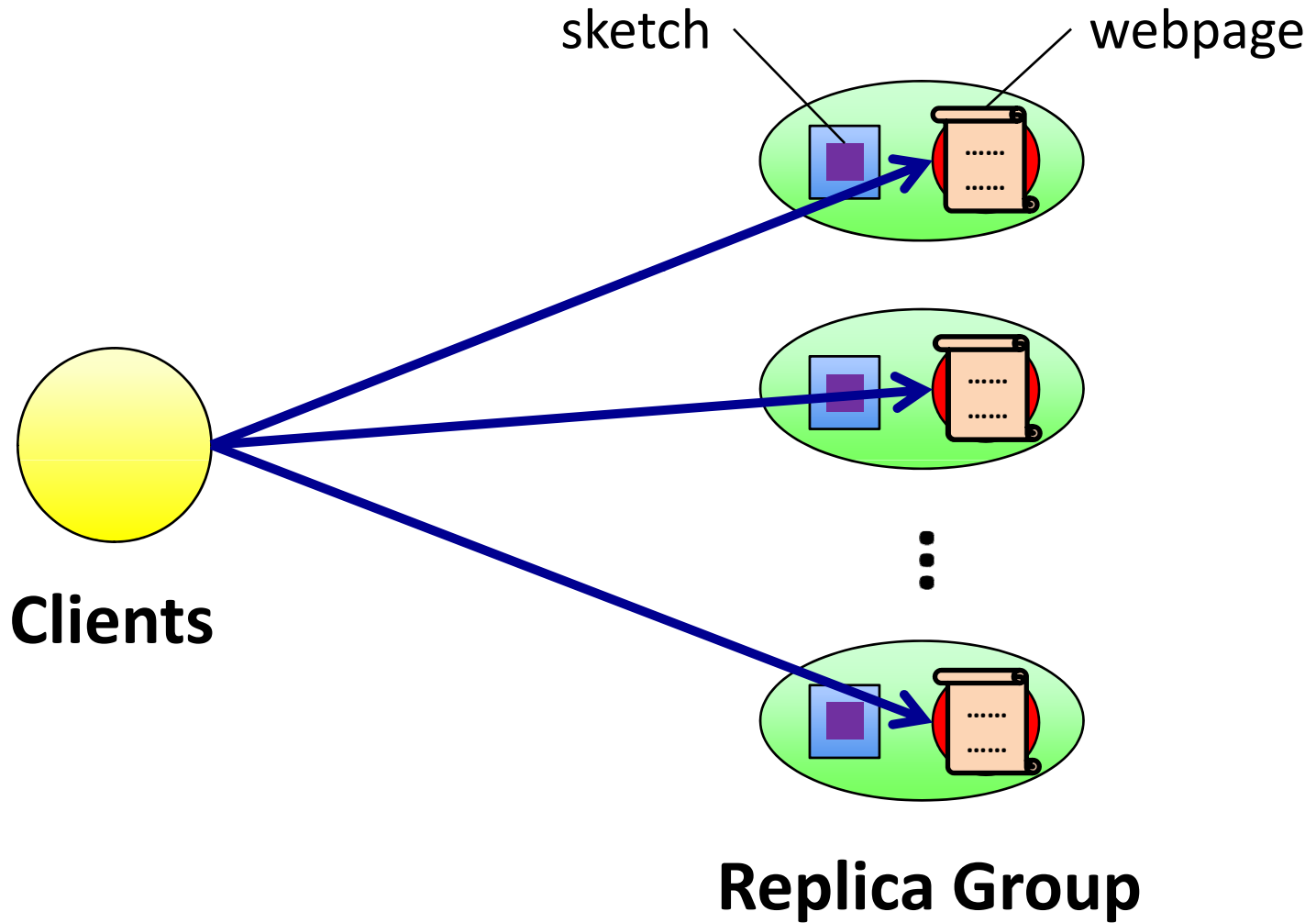
Executing a read



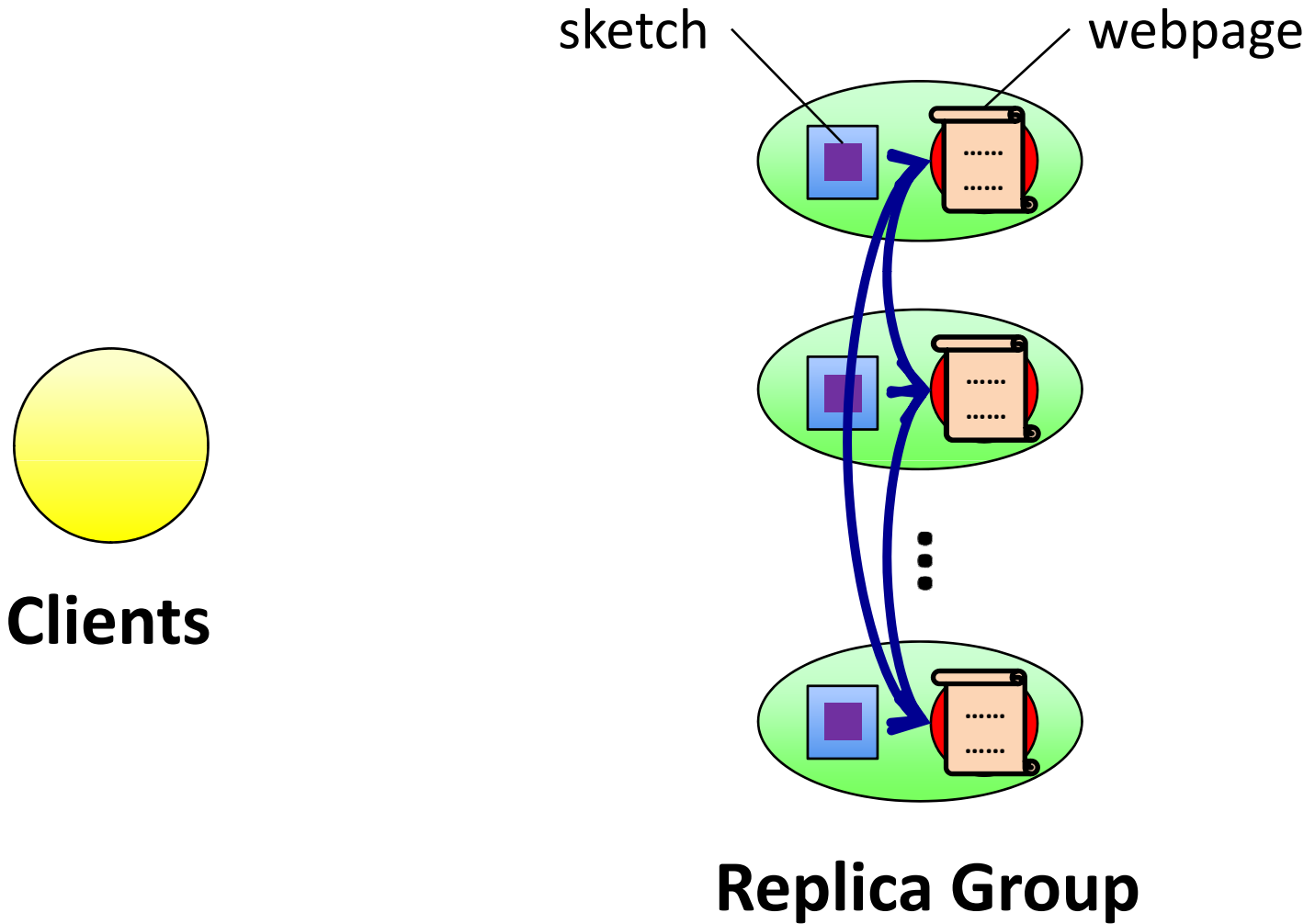
Executing a read



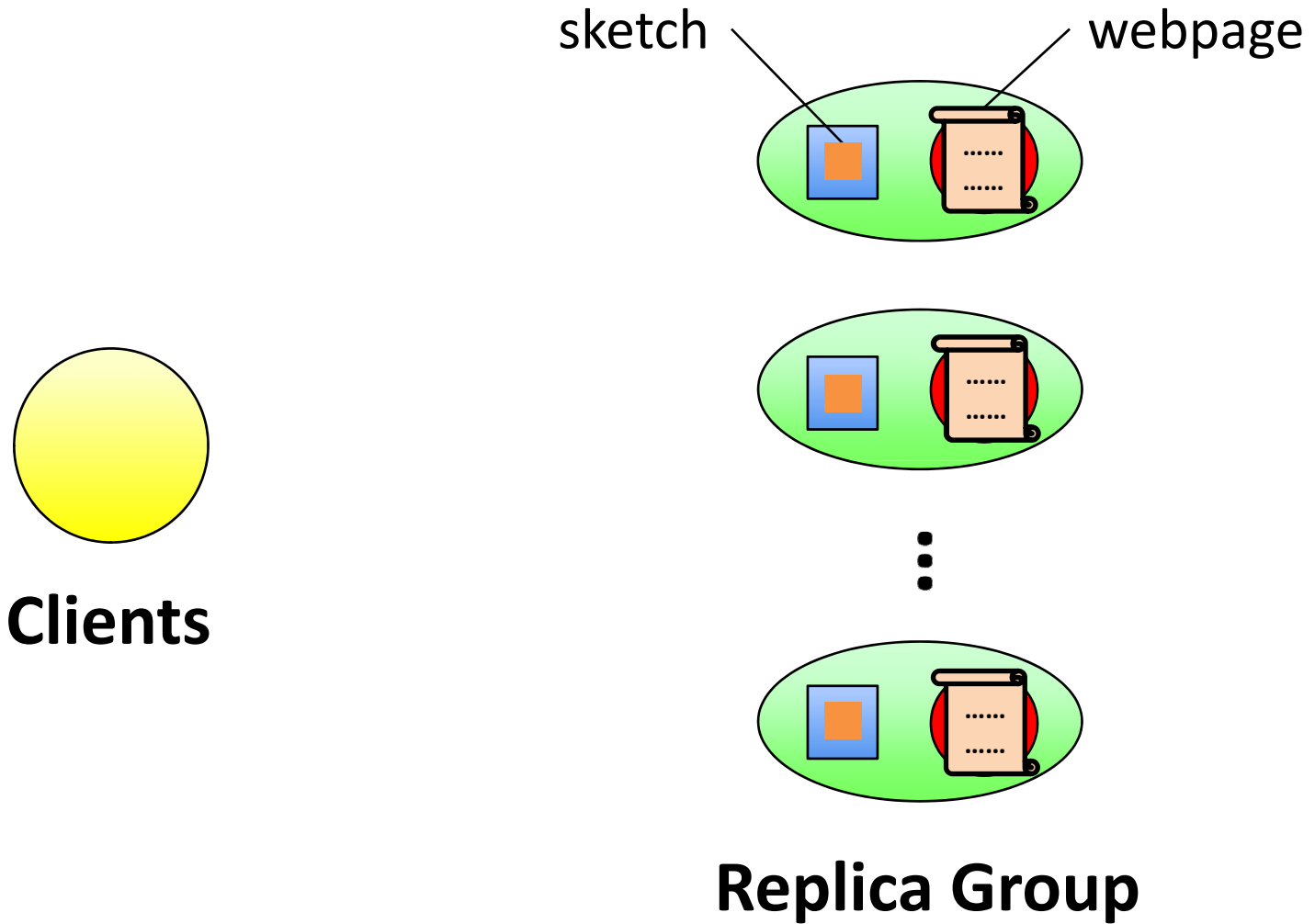
Executing a read



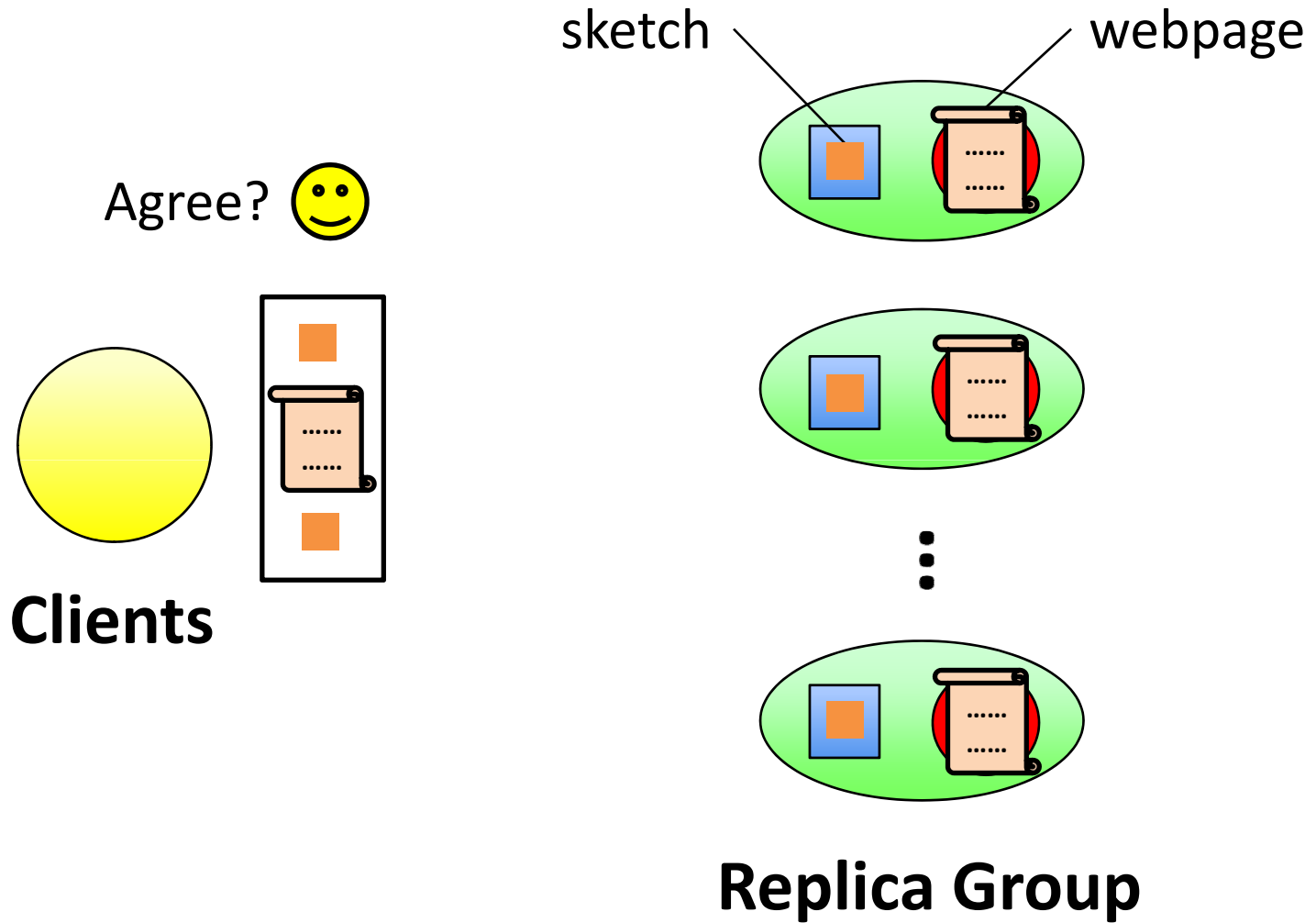
Executing a read



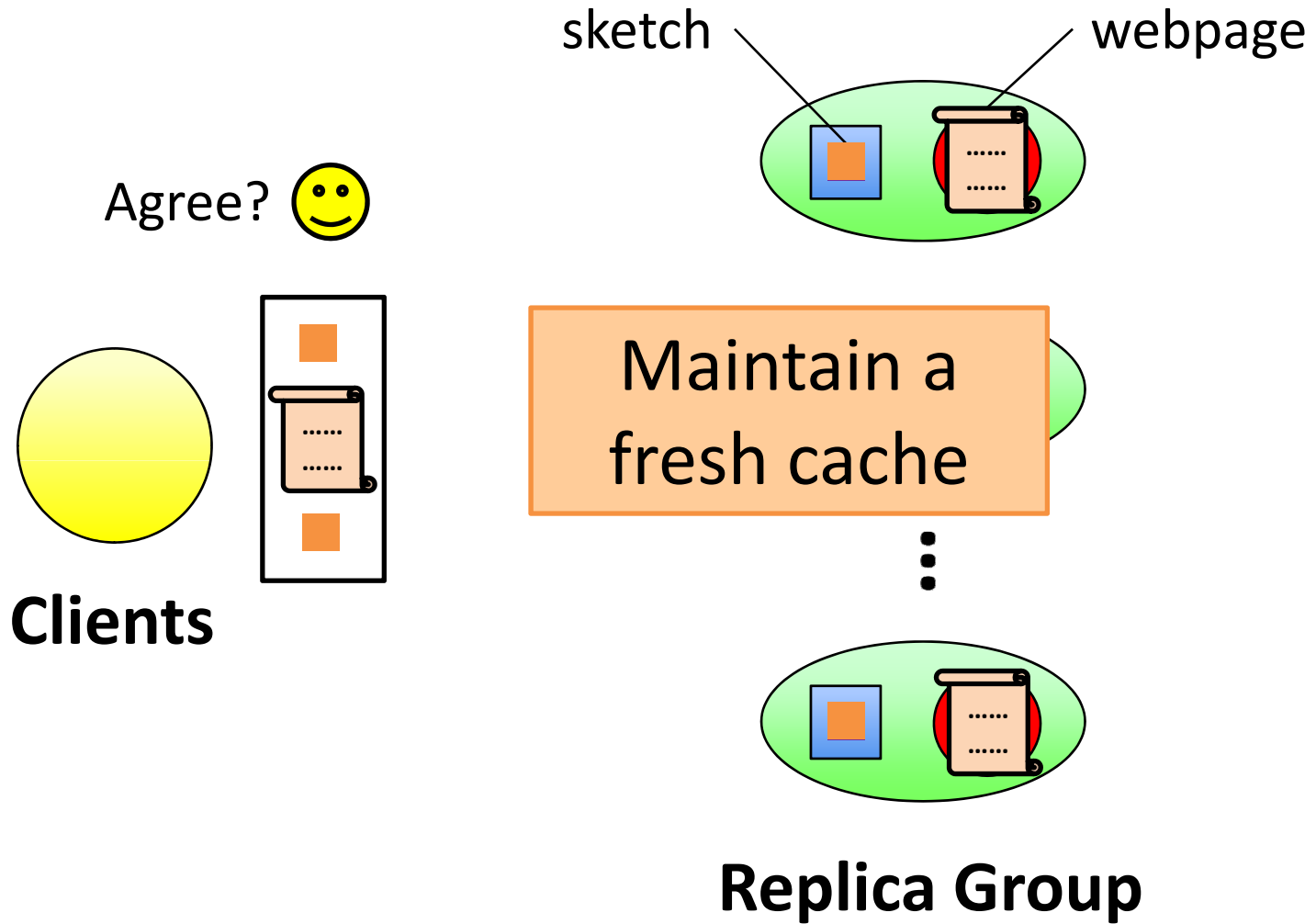
Executing a read



Executing a read



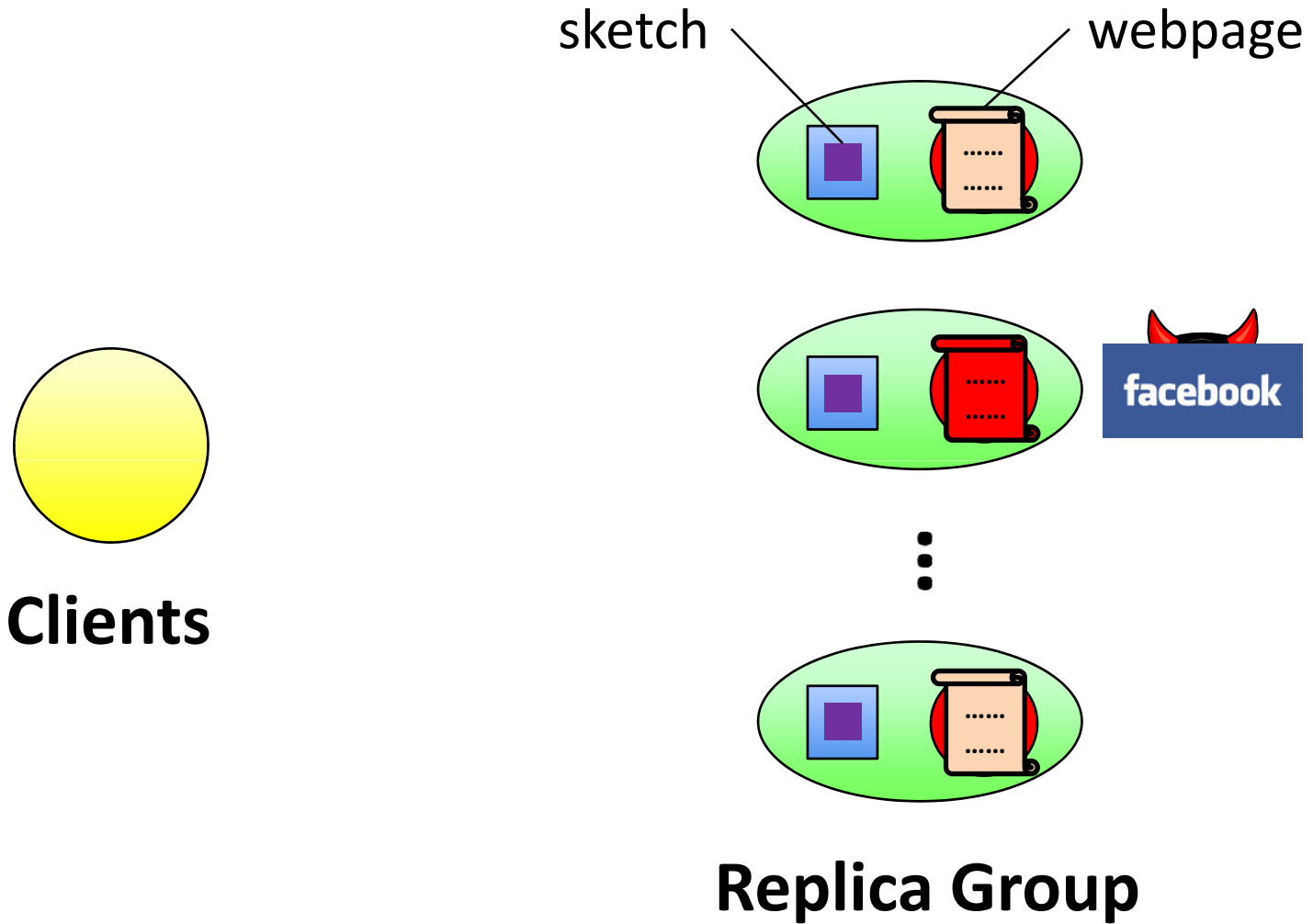
Executing a read



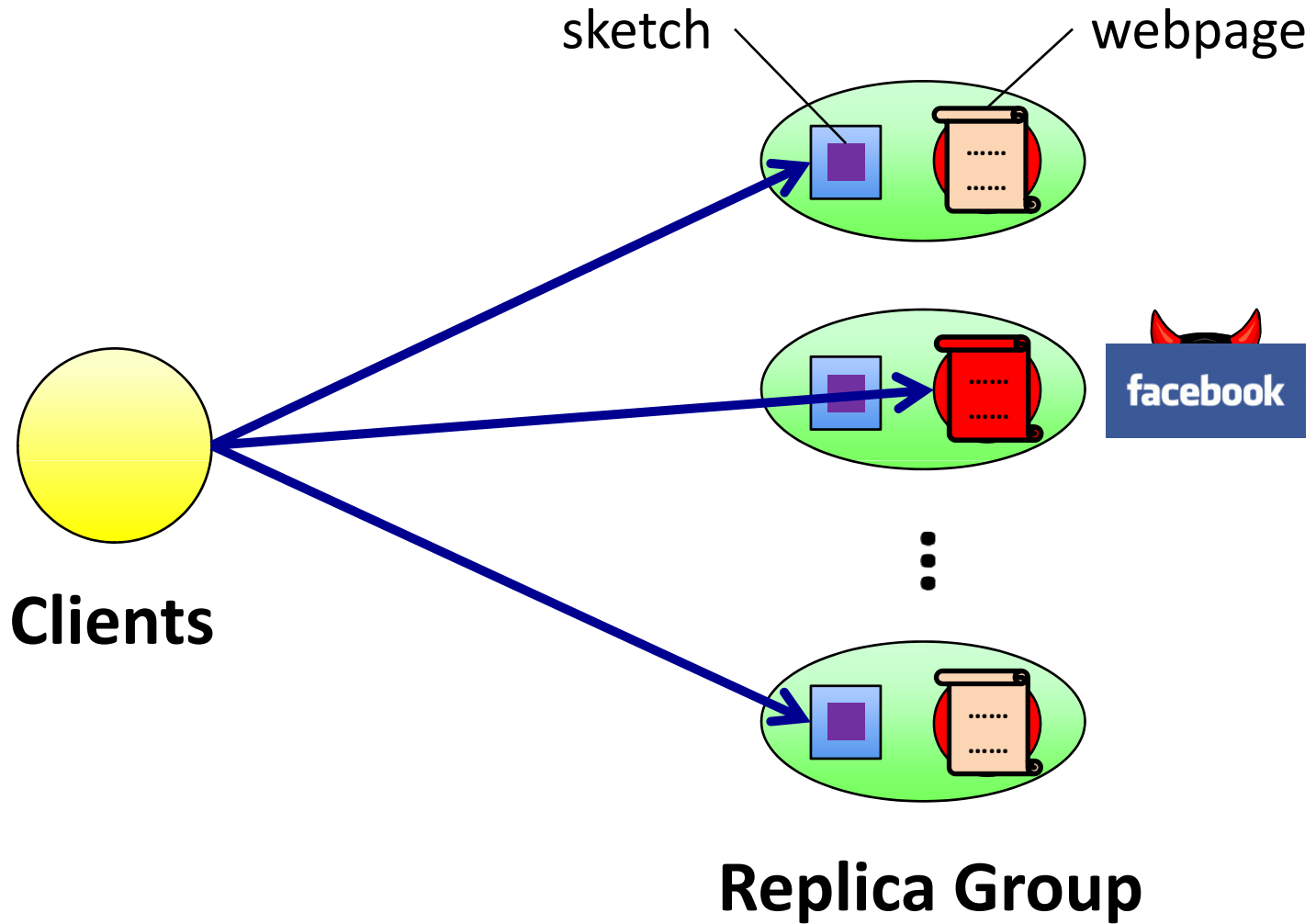
Did we achieve linearizability?

NO!

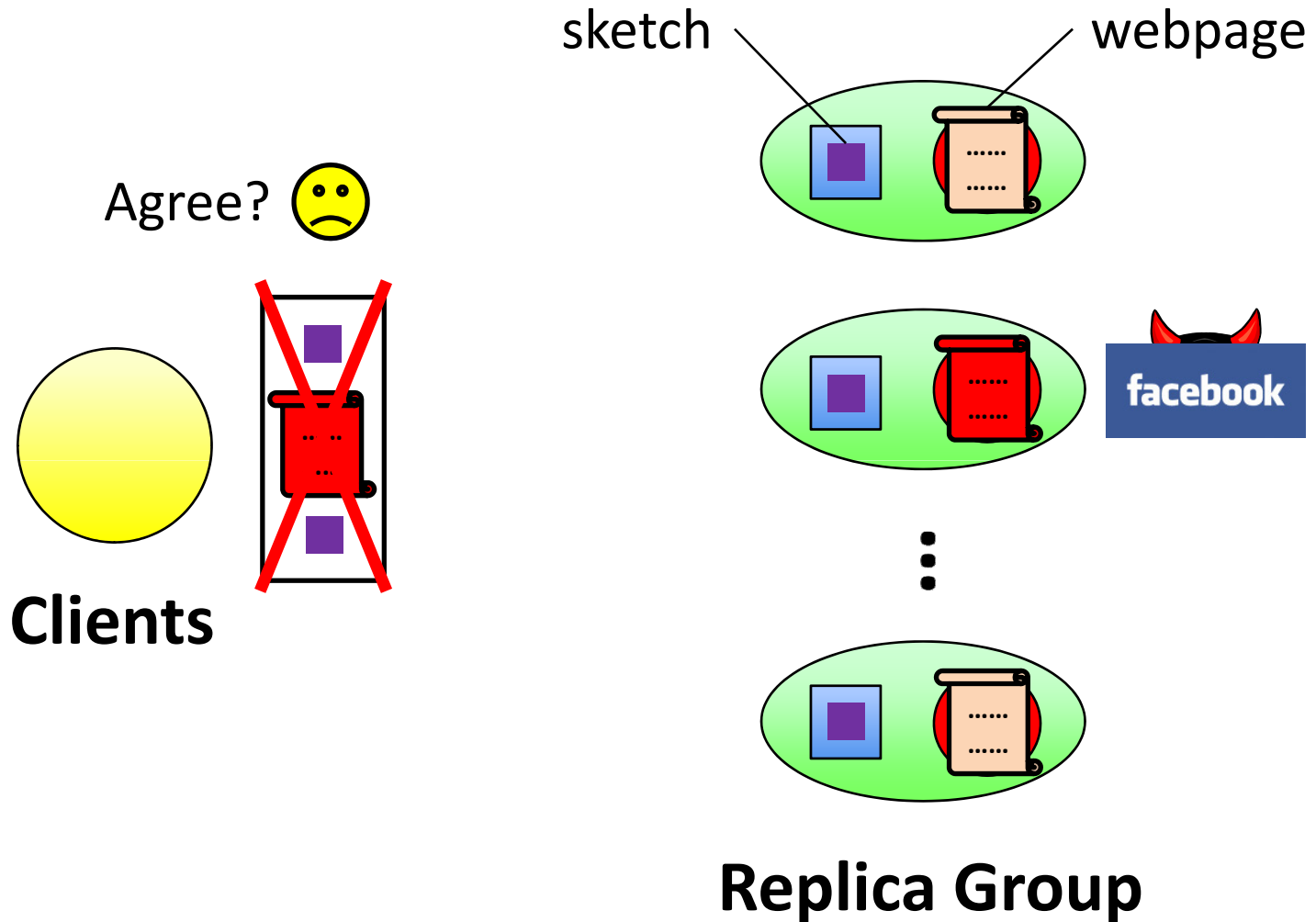
Executing a read



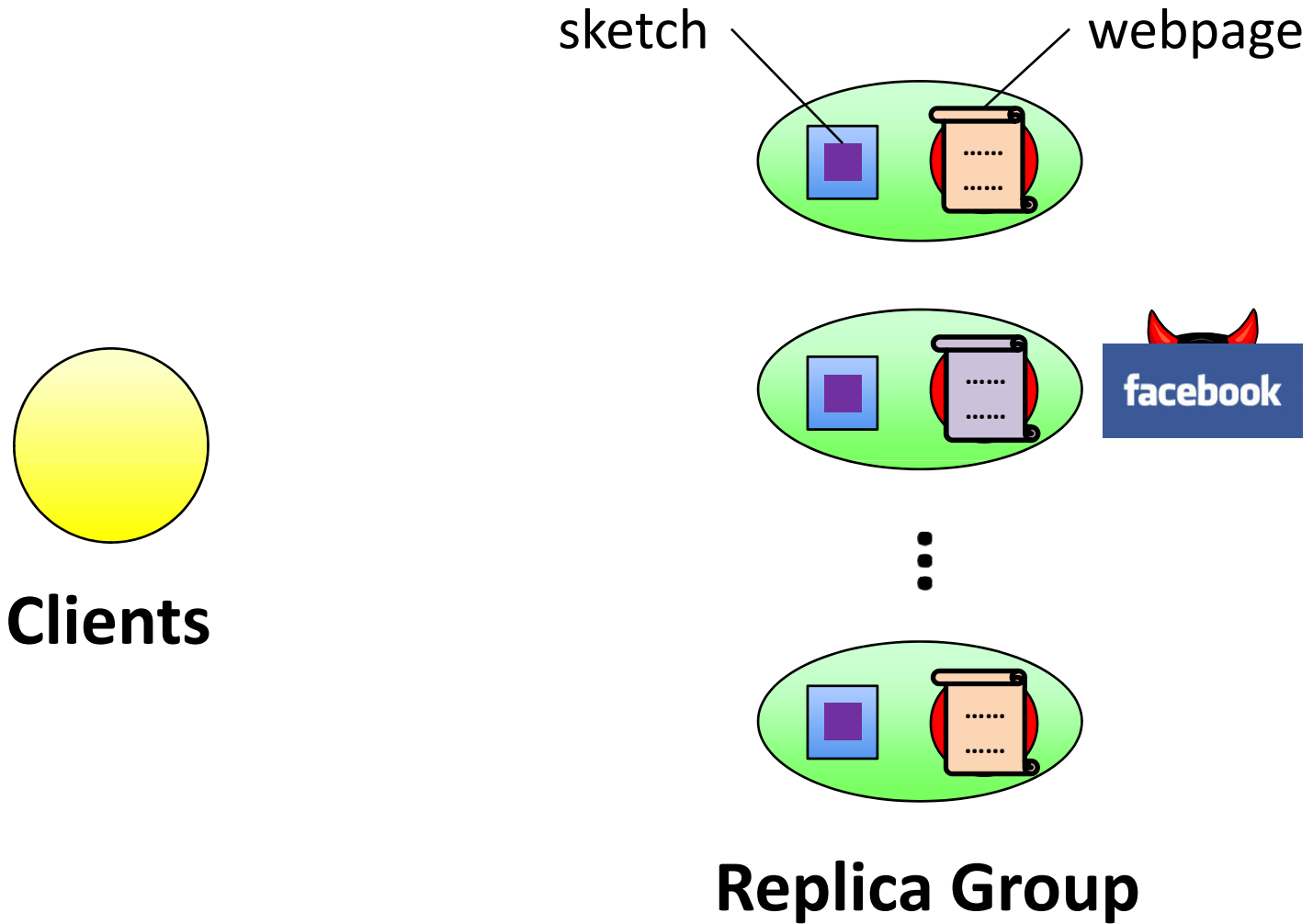
Executing a read



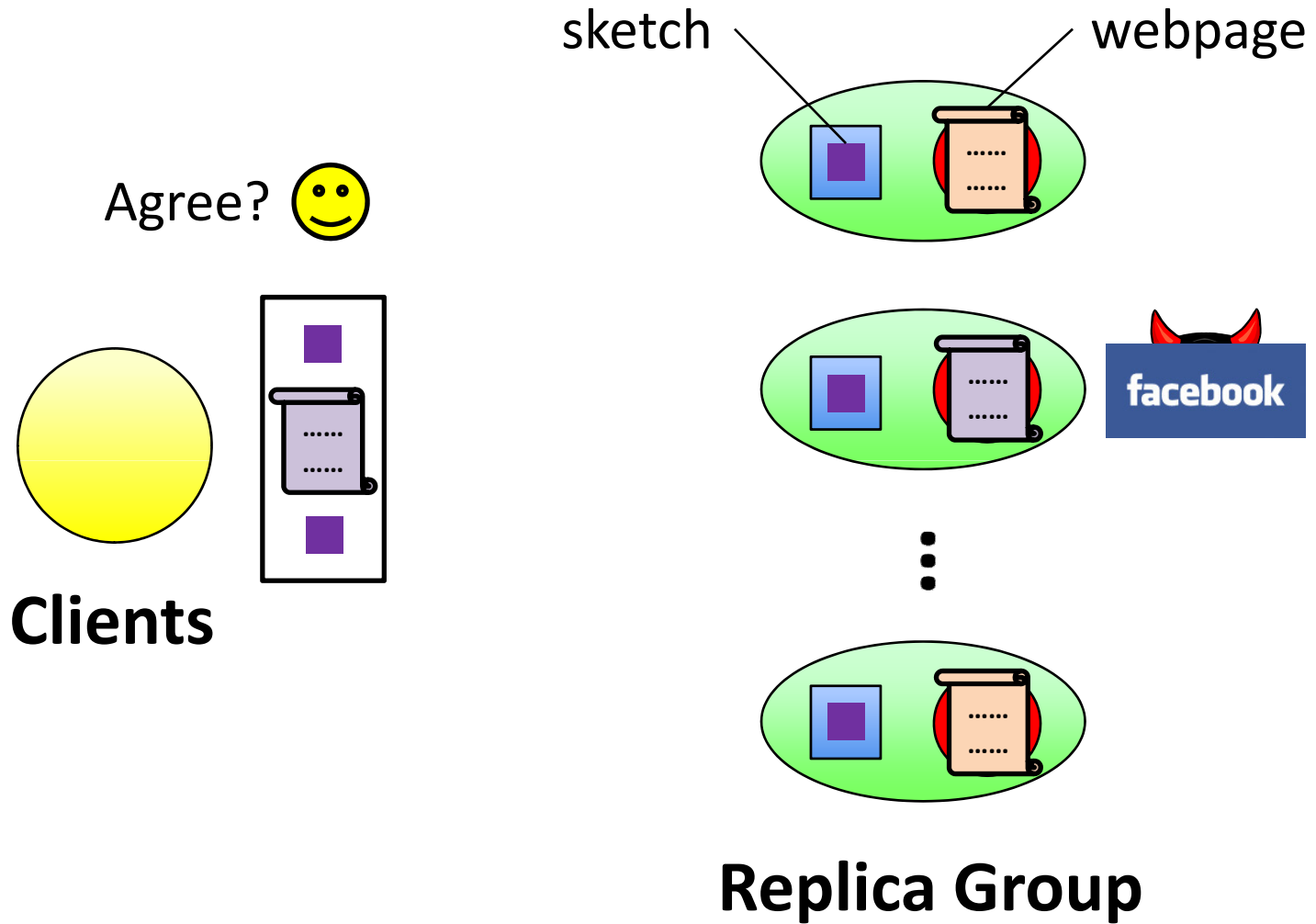
Executing a read



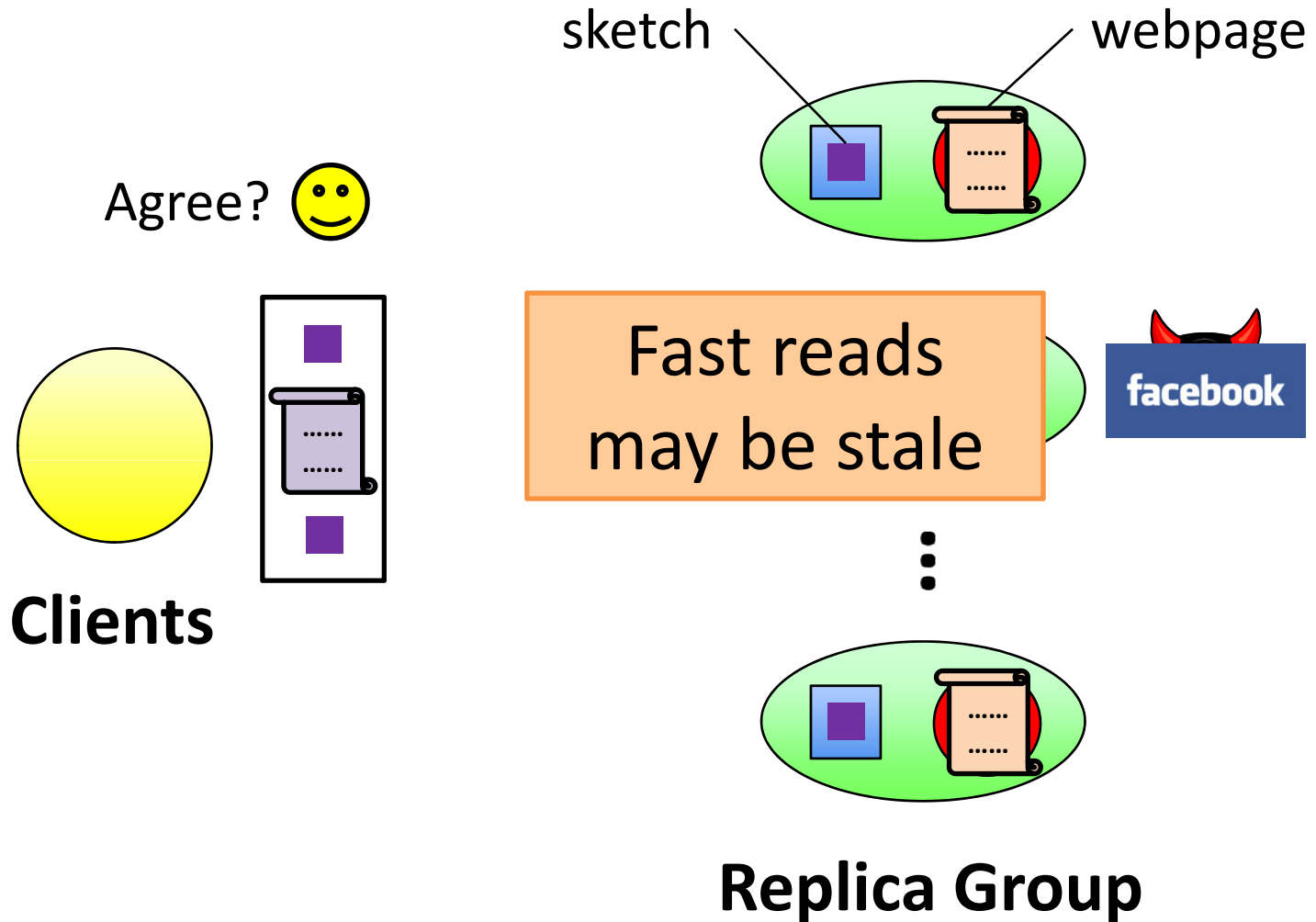
Executing a read



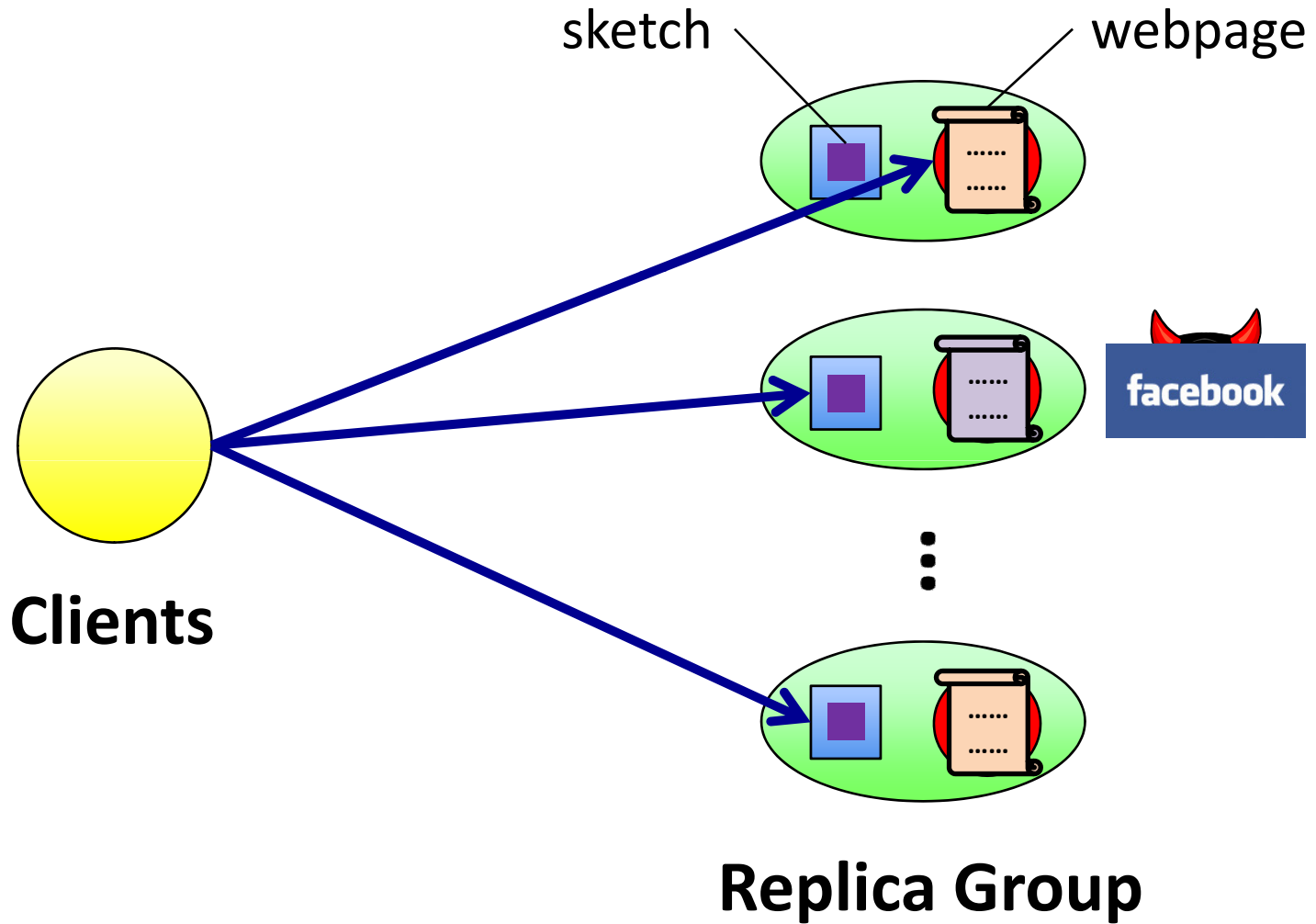
Executing a read



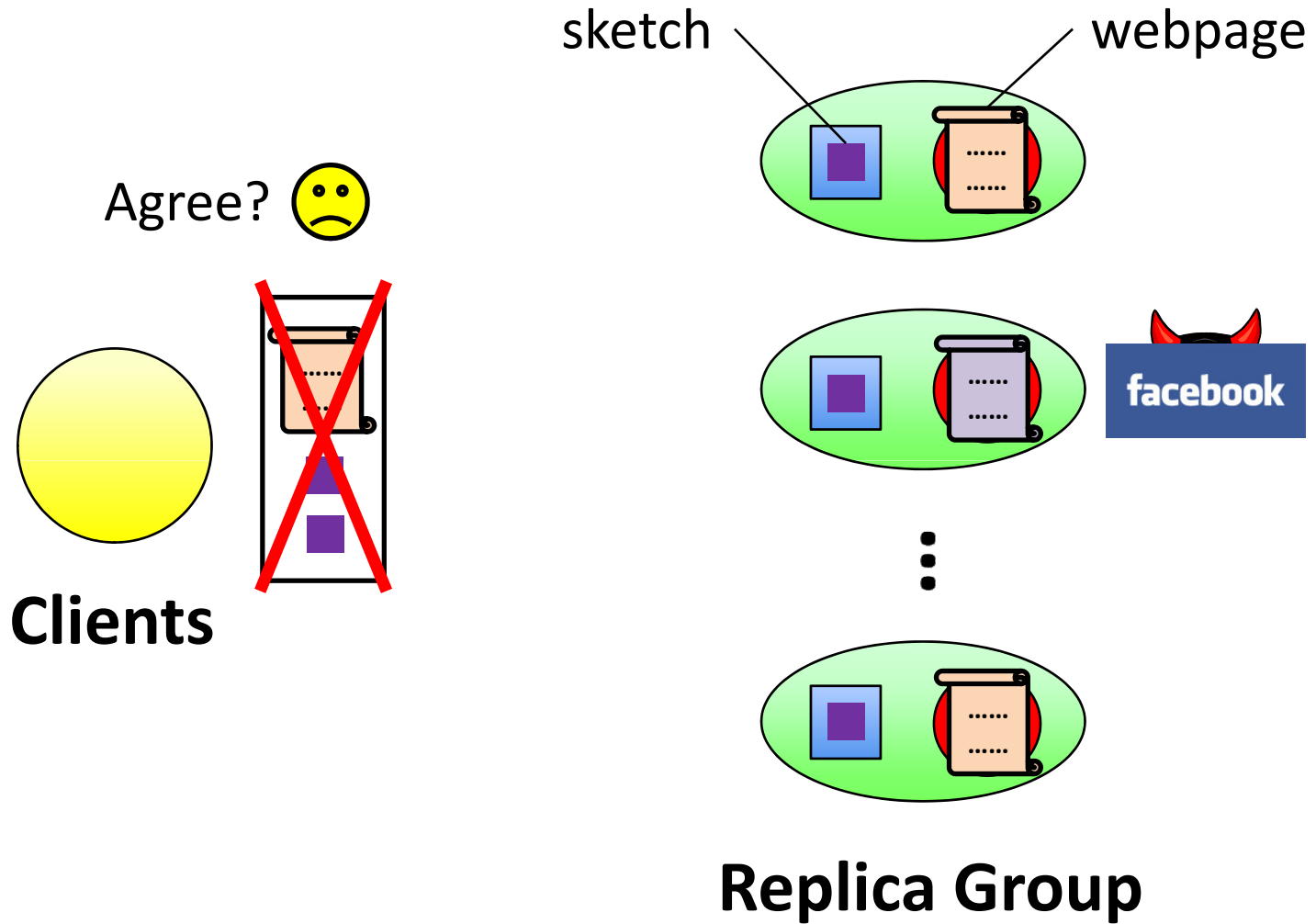
Executing a read



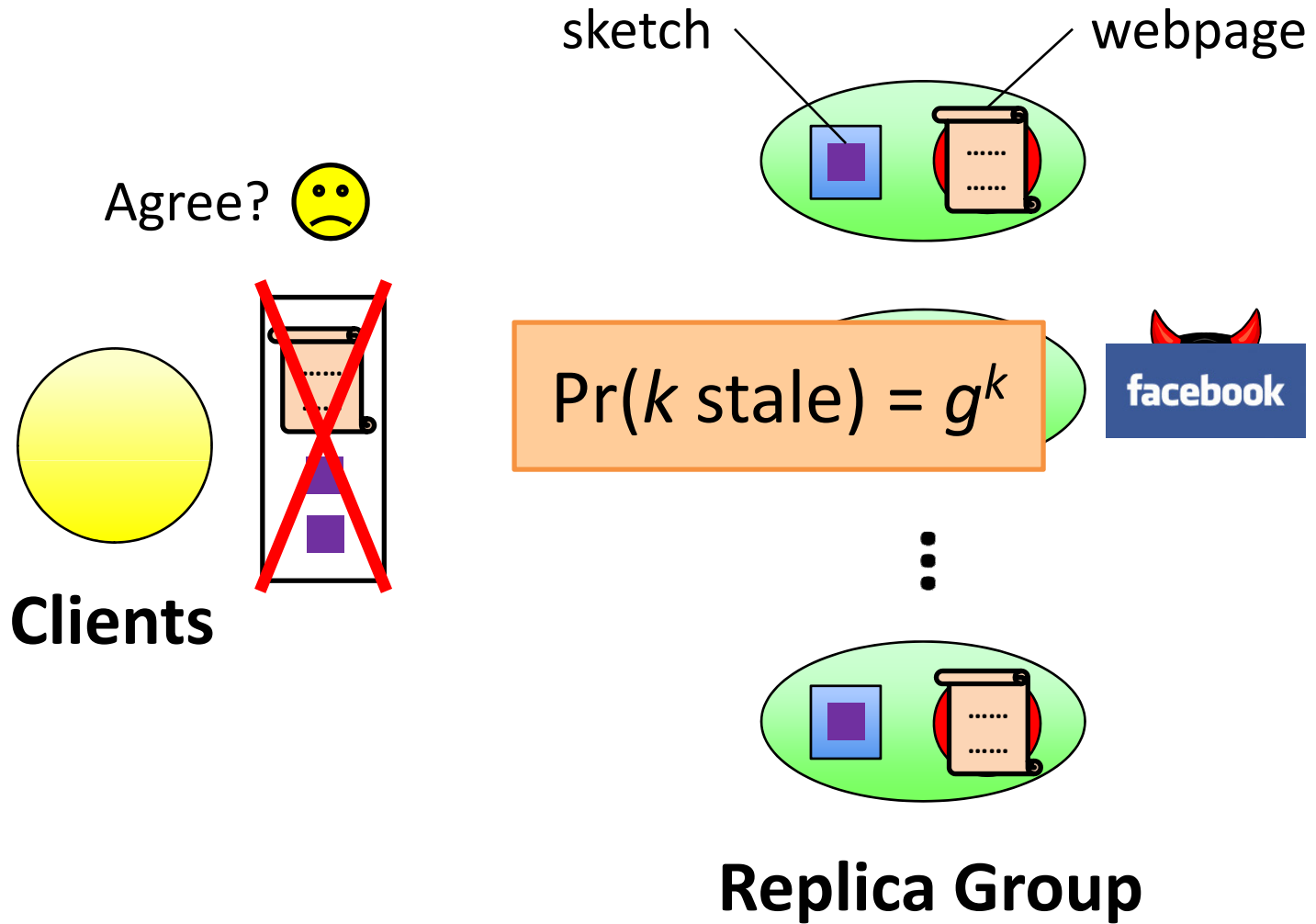
Load balancing



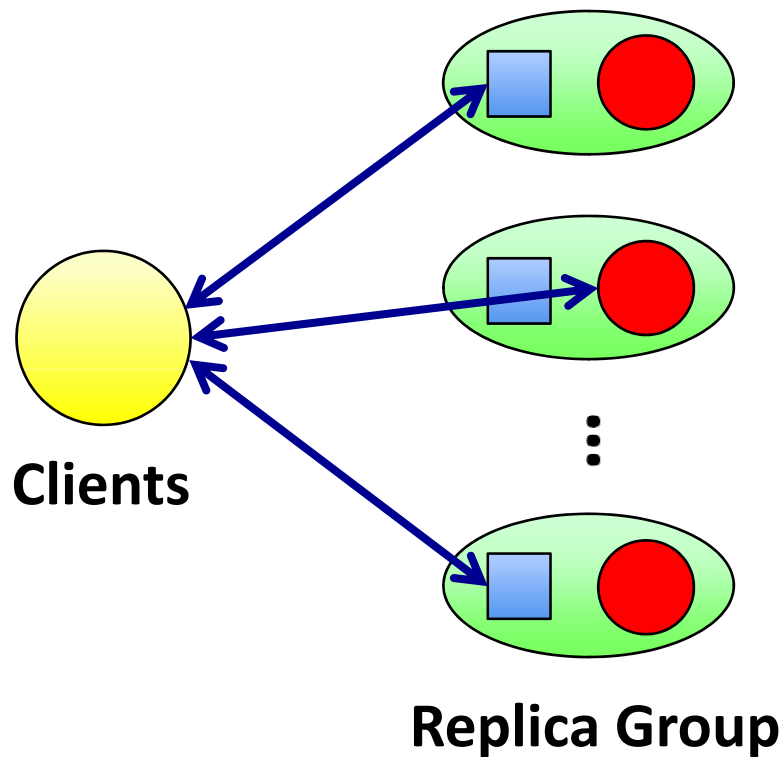
Load balancing



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~~

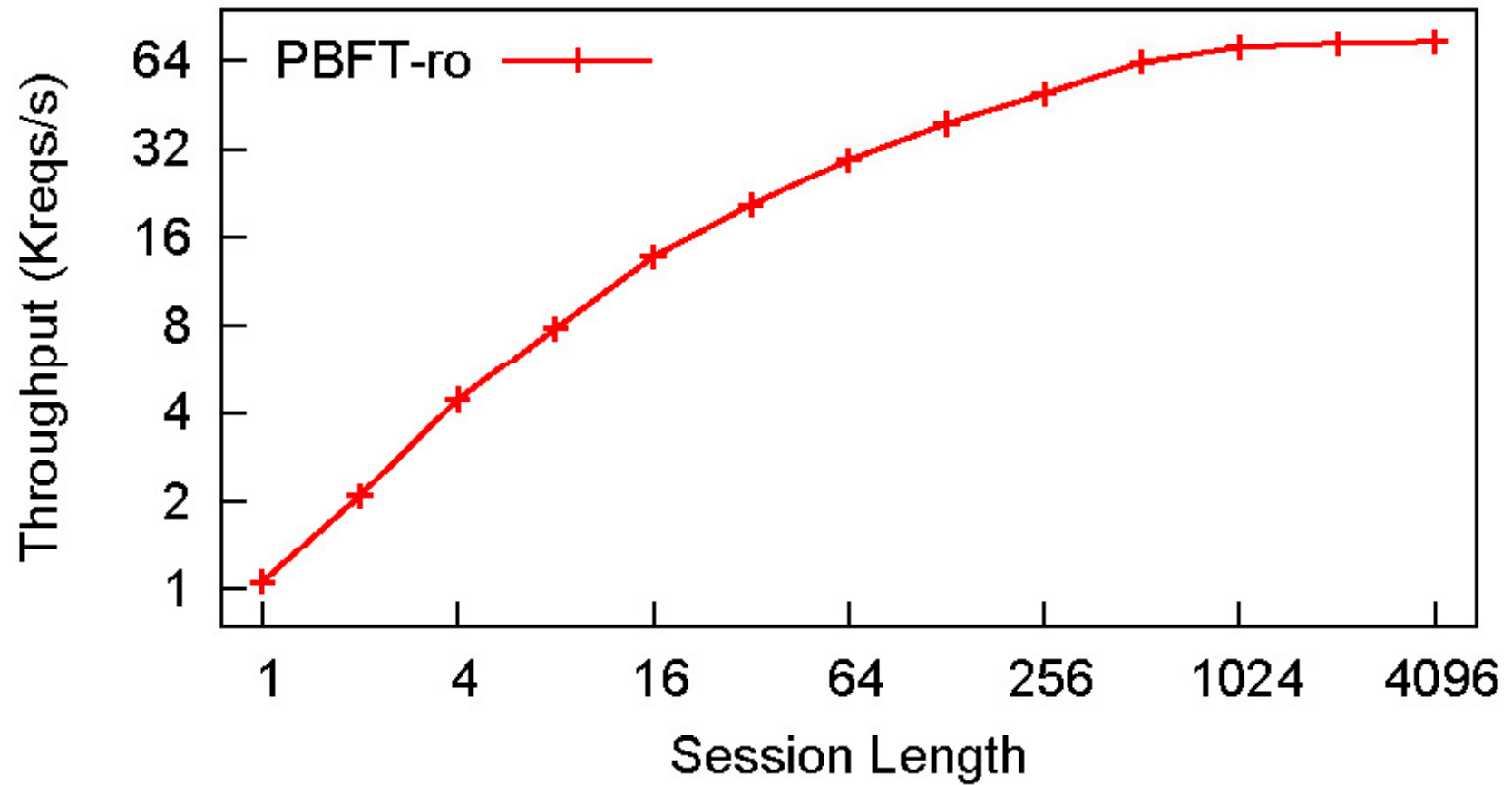
D-Prophecy

- Modifies clients

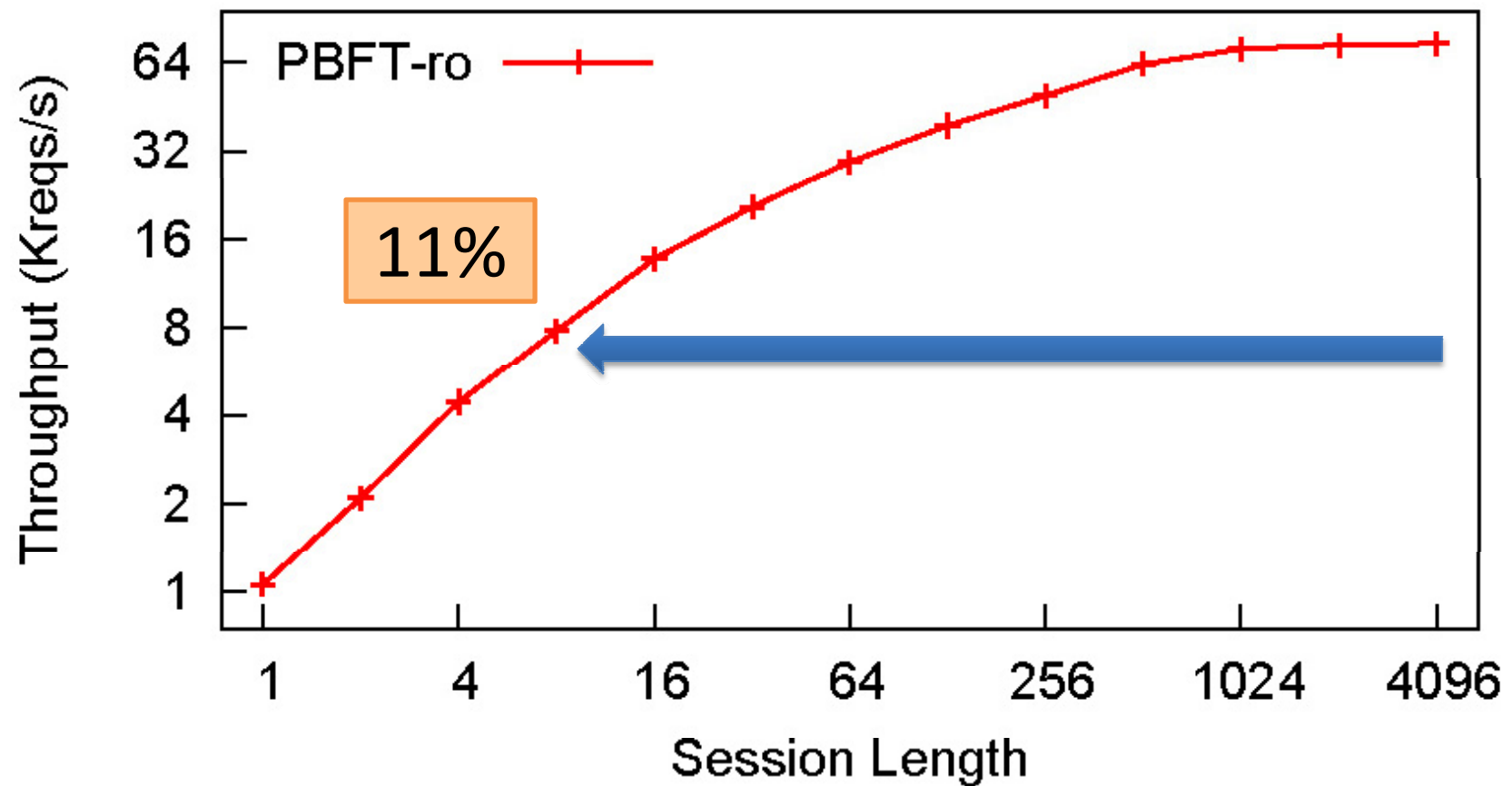
Prophecy

- Long-lived sessions

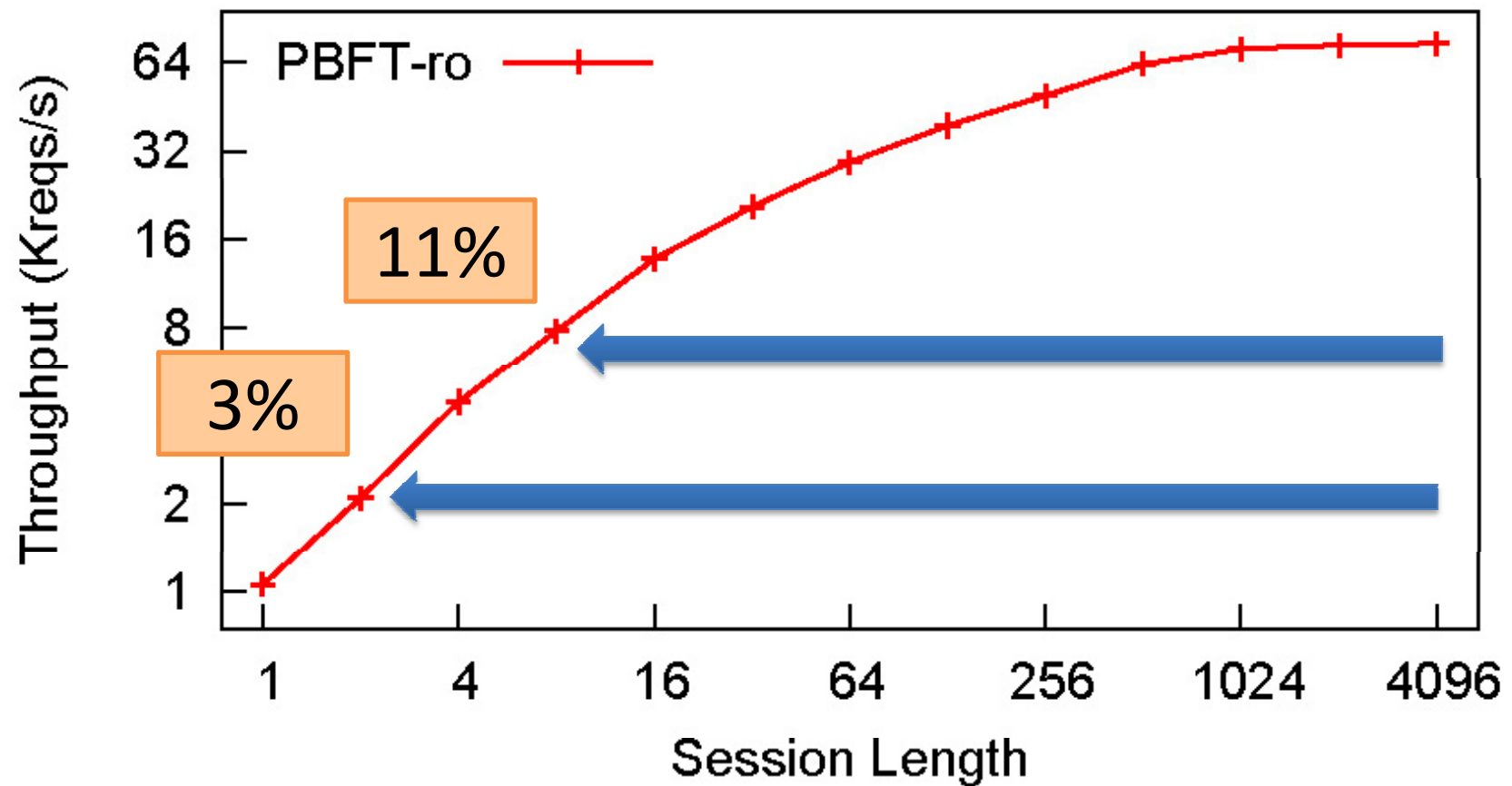
Key-exchange overhead



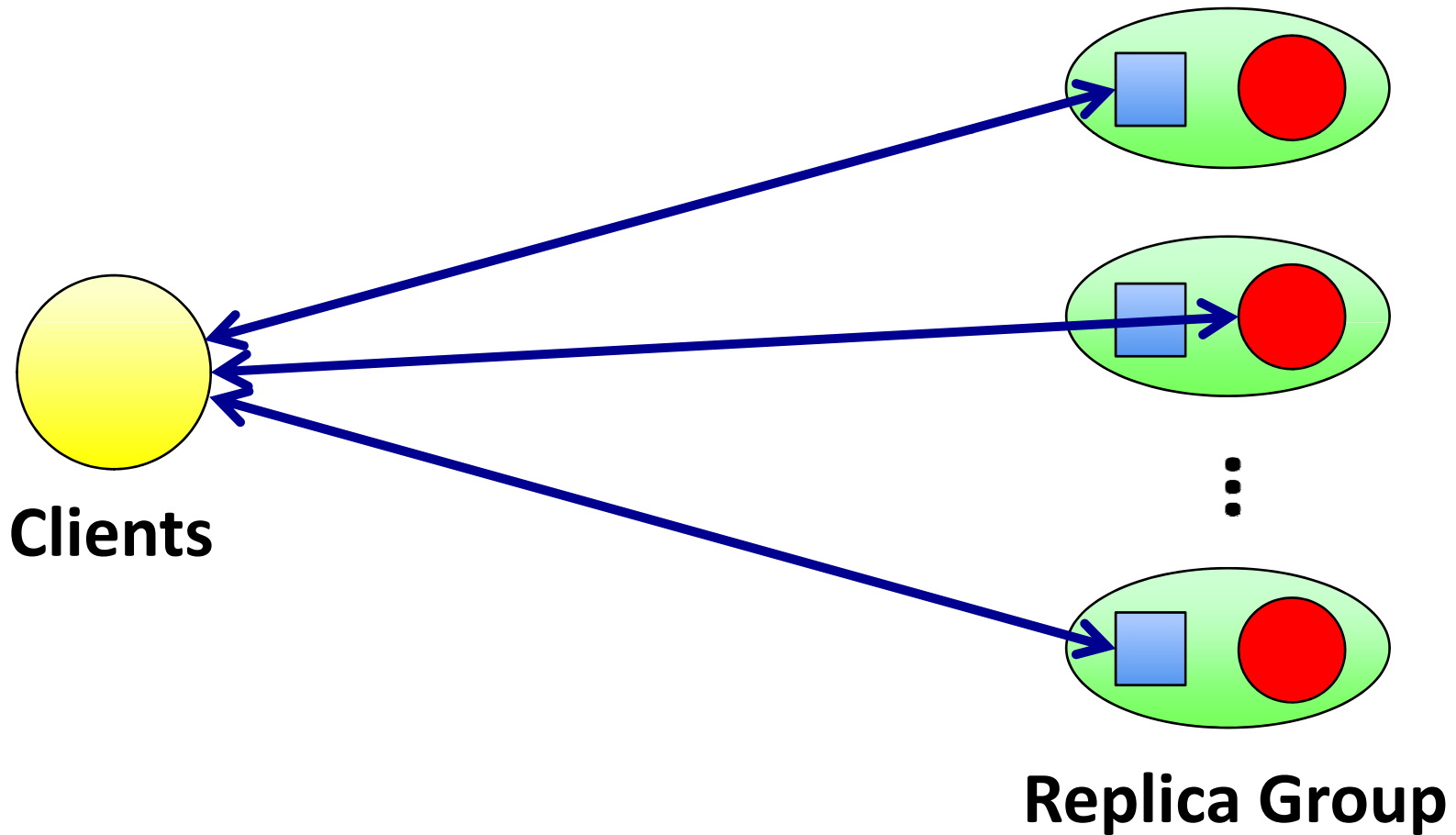
Key-exchange overhead



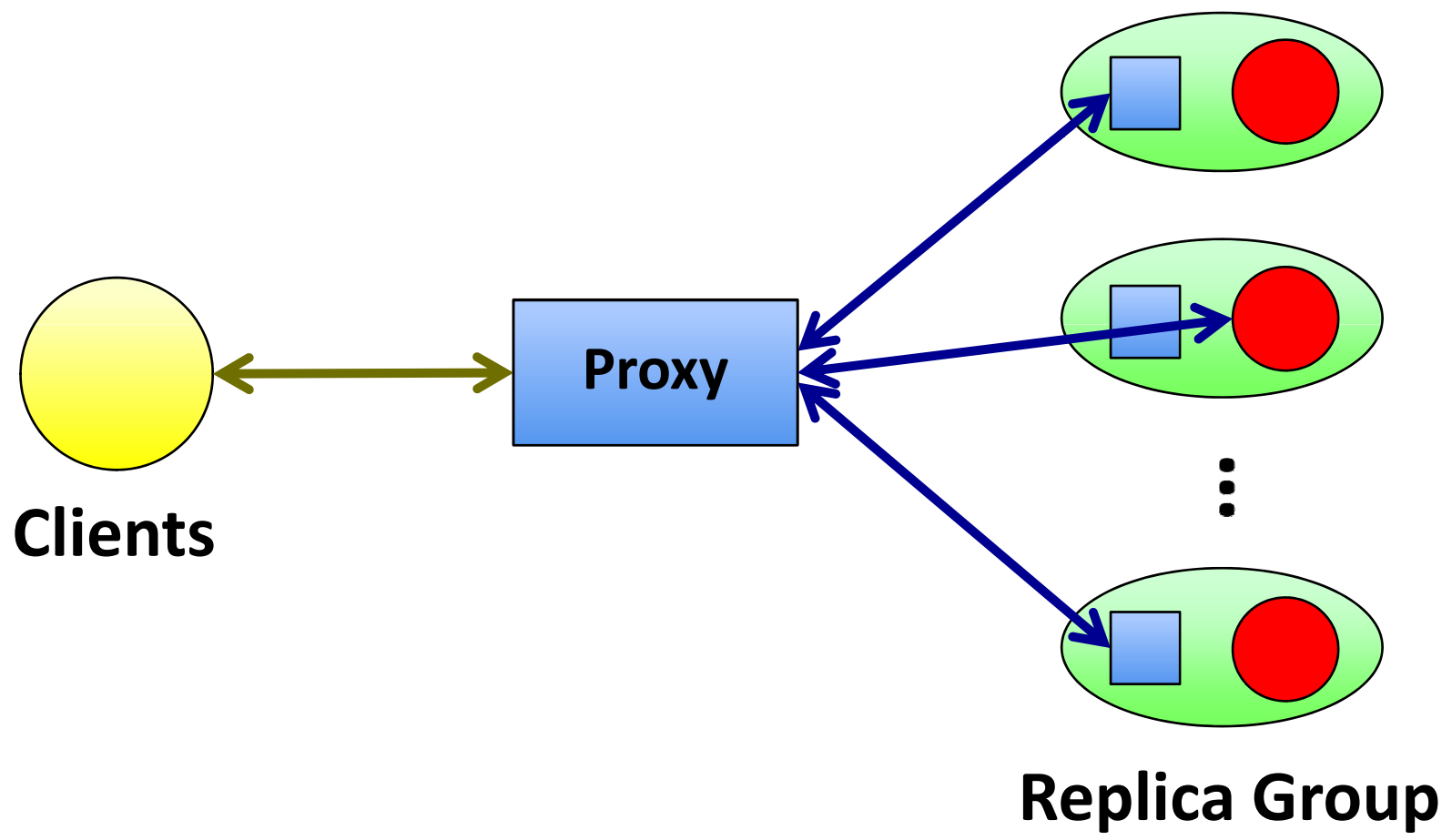
Key-exchange overhead



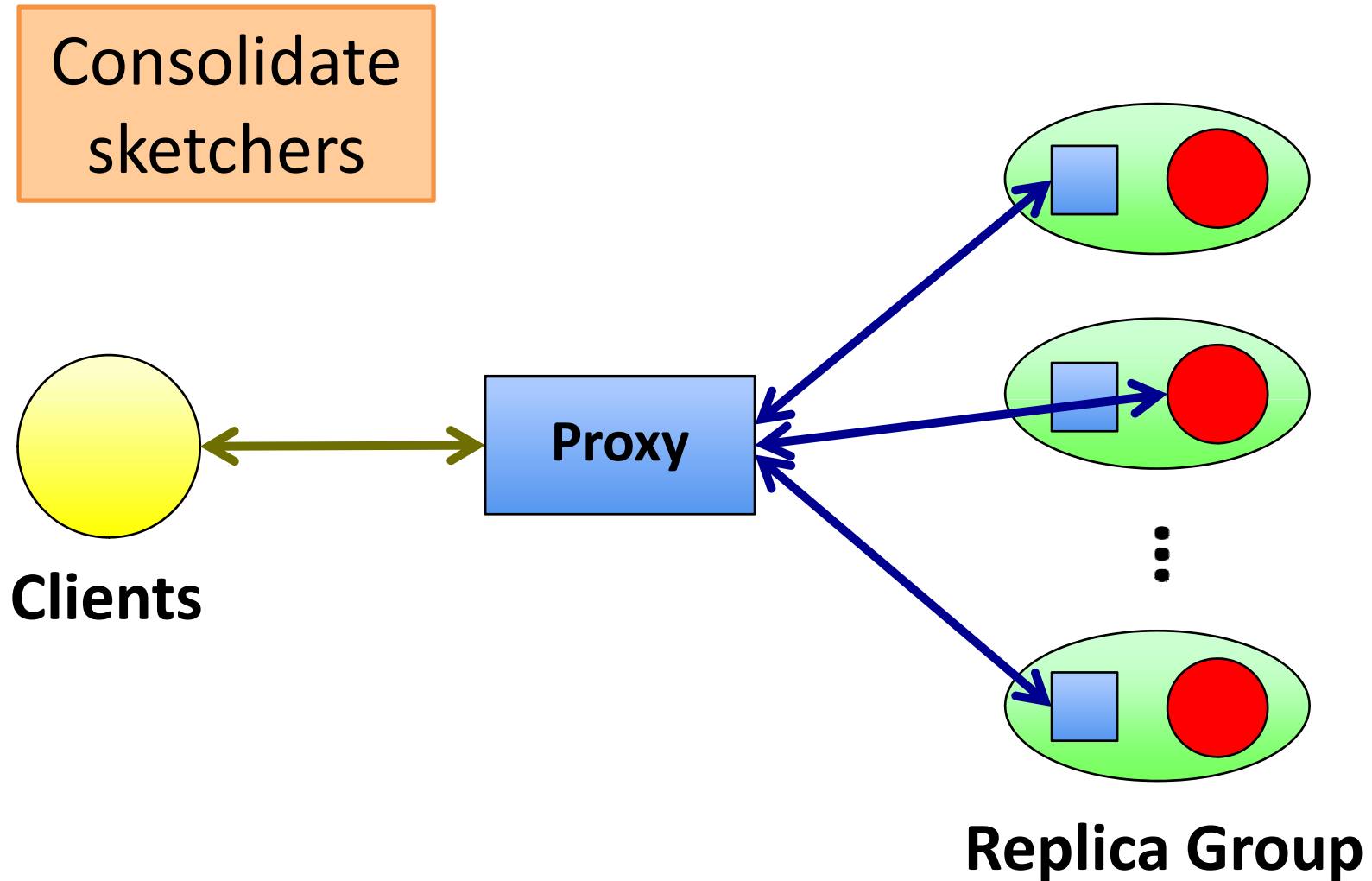
Internet services



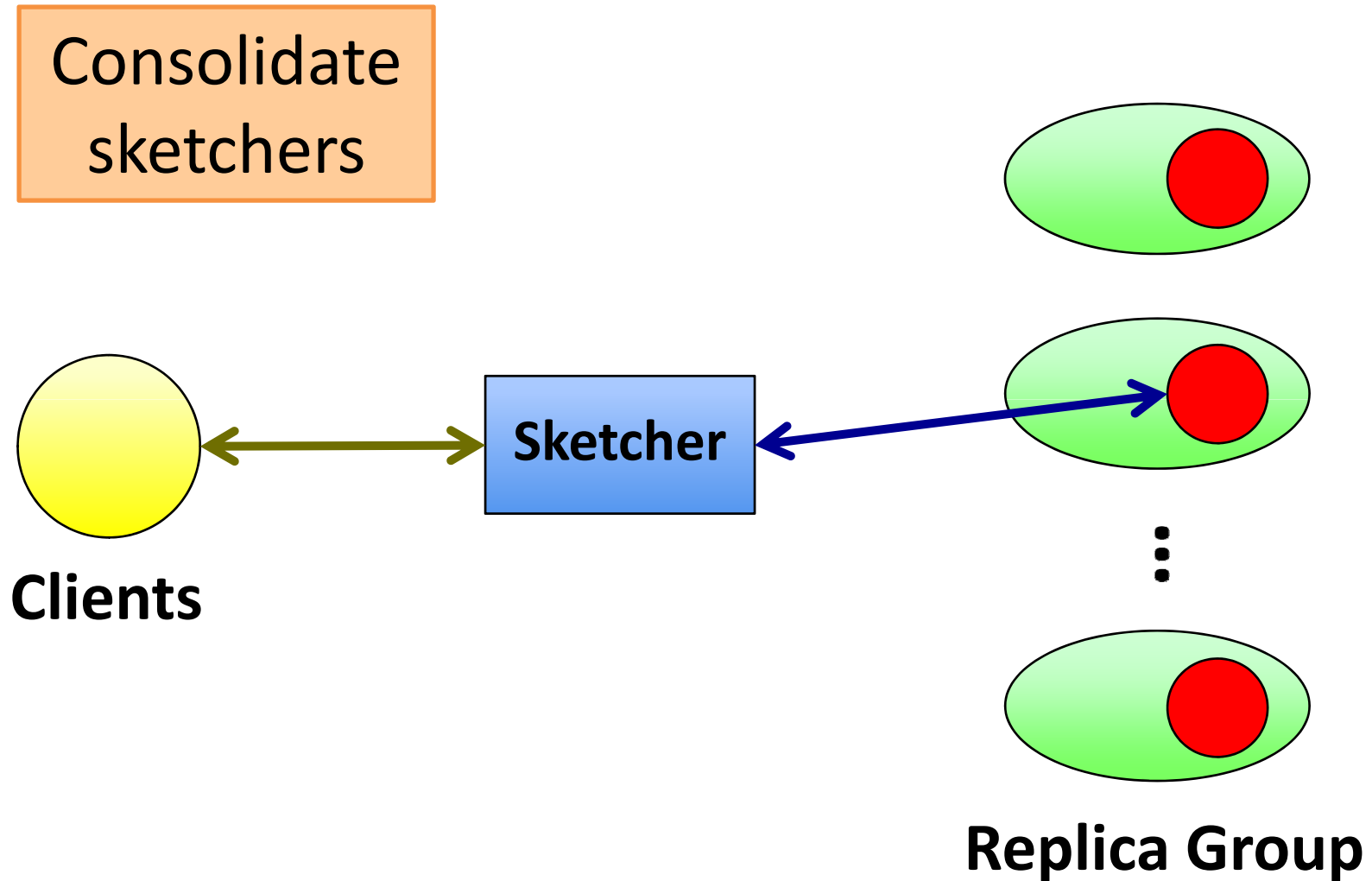
A proxy solution



A proxy solution

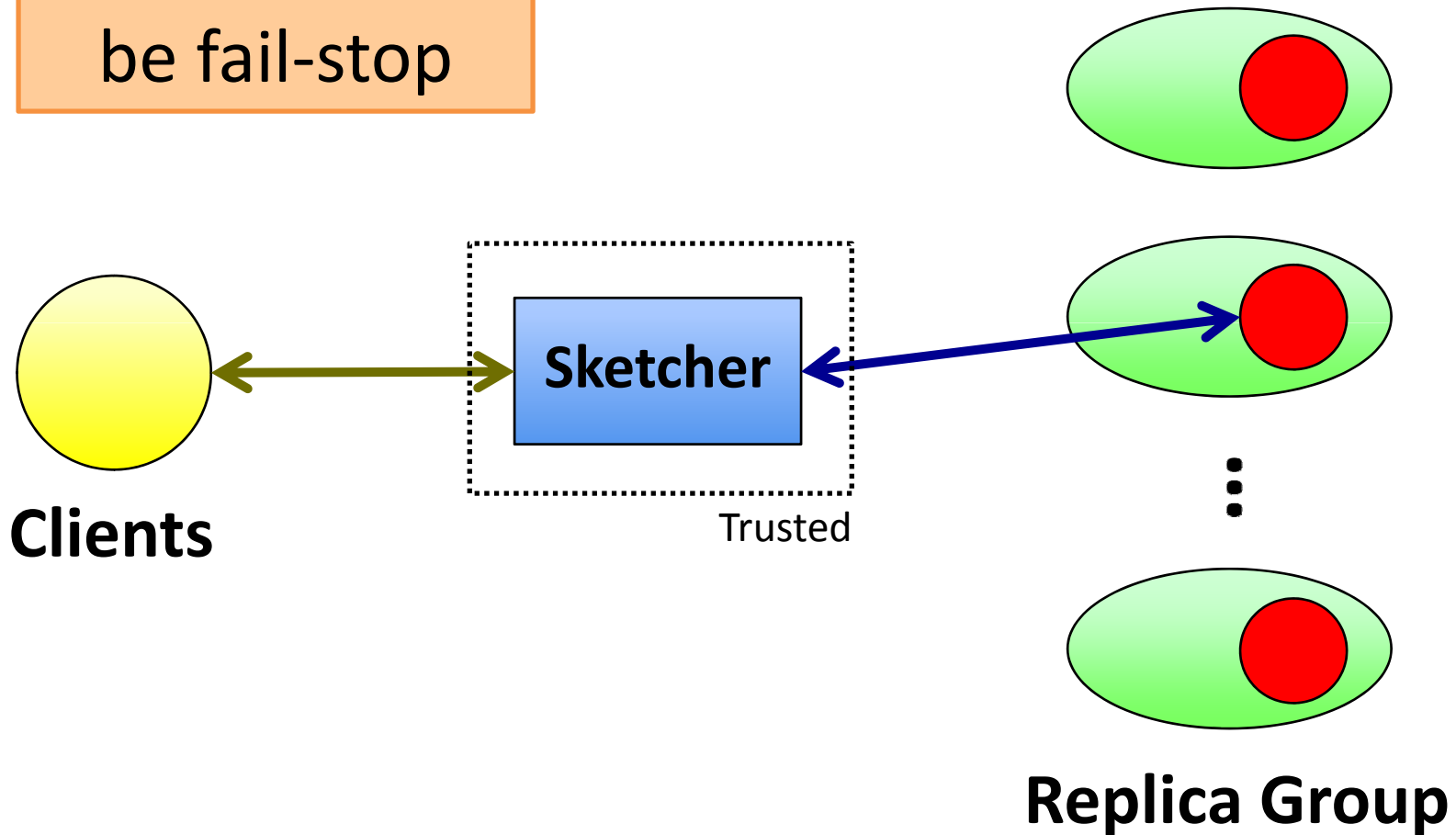


A proxy solution



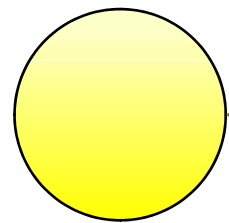
A proxy solution

Sketcher must be fail-stop



A proxy solution

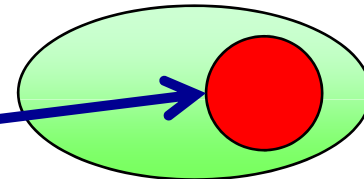
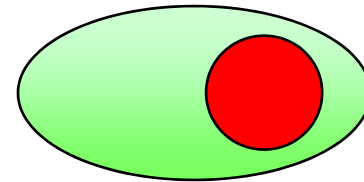
- Trust middlebox already
- Small and simple



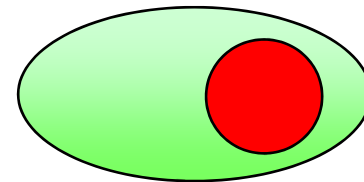
Clients



Trusted

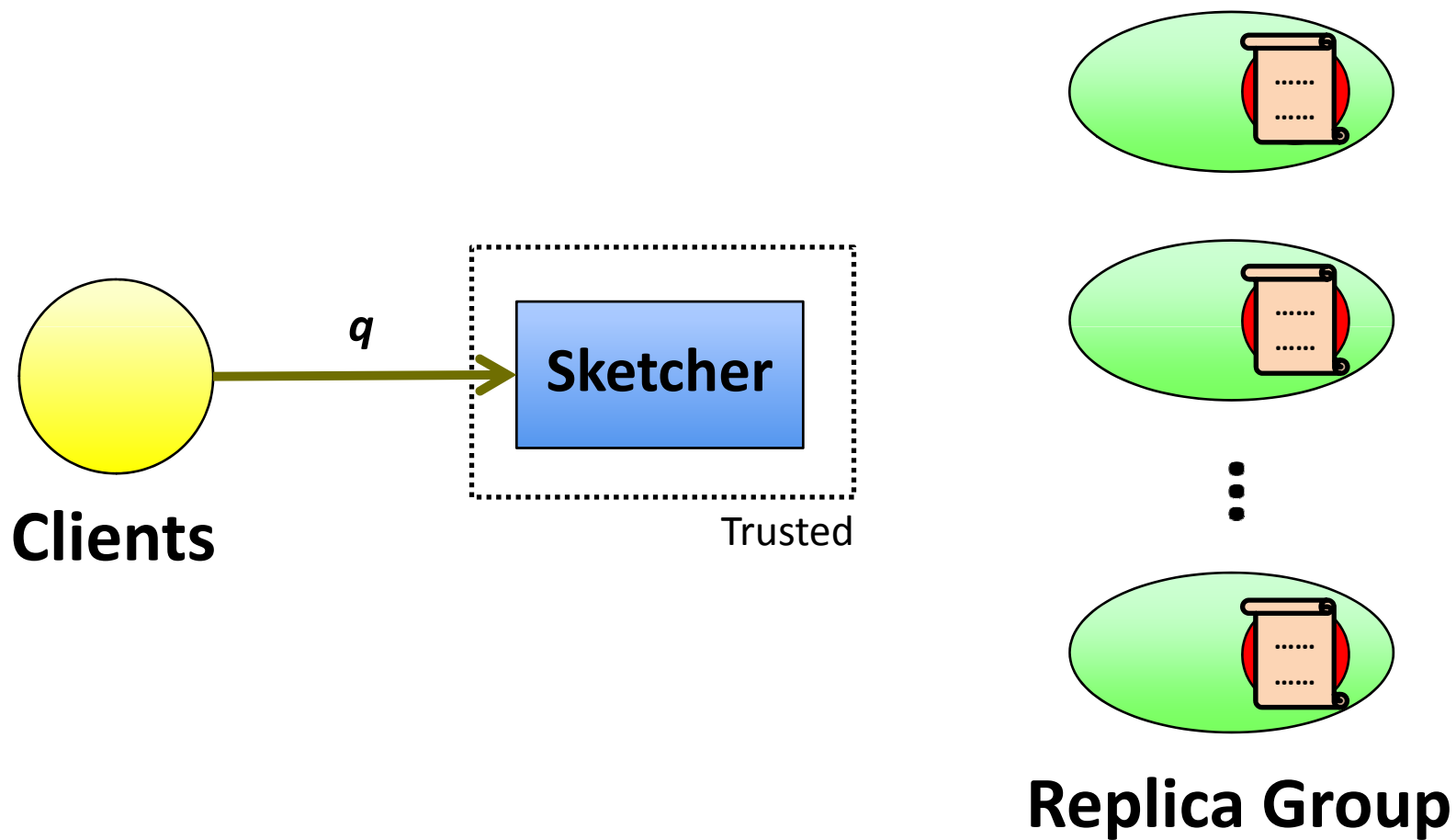


⋮

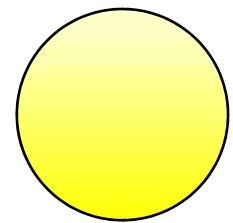


Replica Group

Executing a read



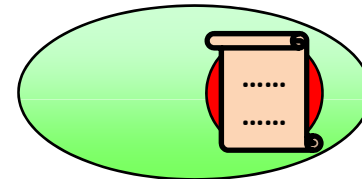
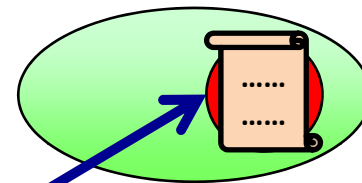
Executing a read



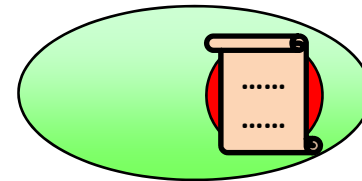
Clients



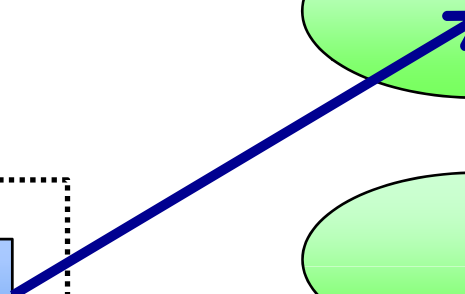
Trusted



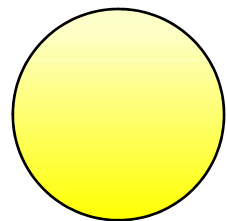
⋮



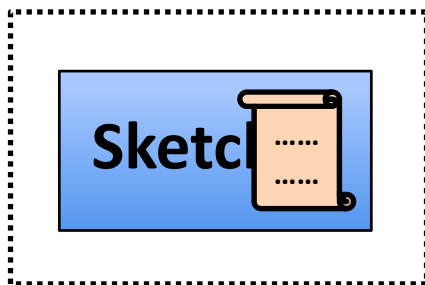
Replica Group



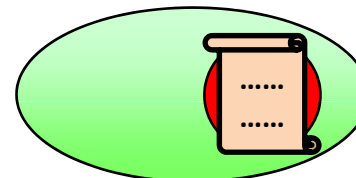
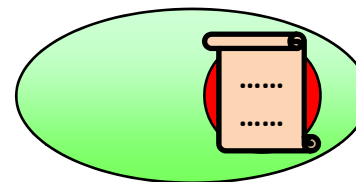
Executing a read



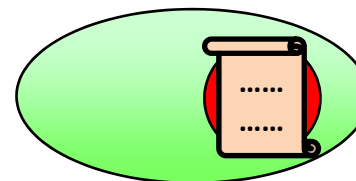
Clients



Trusted

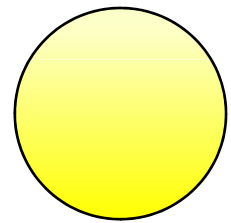


⋮

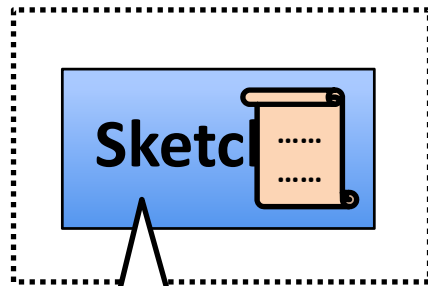


Replica Group

Executing a read

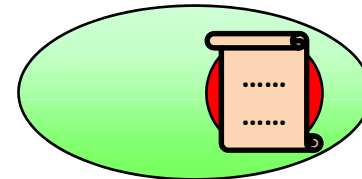
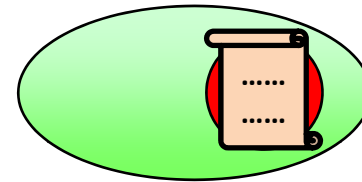


Clients

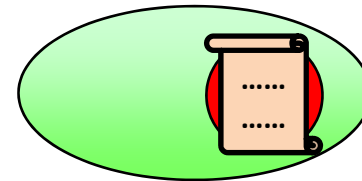


Trusted

Req	Resp
$s(q)$	
\vdots	\vdots

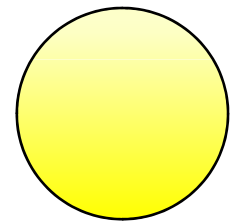


\vdots



Replica Group

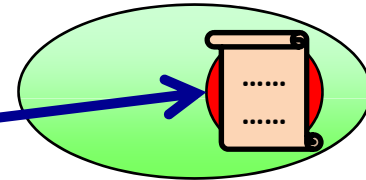
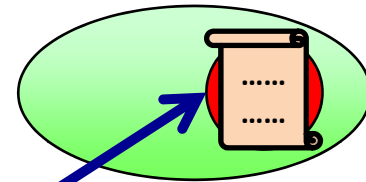
Executing a read



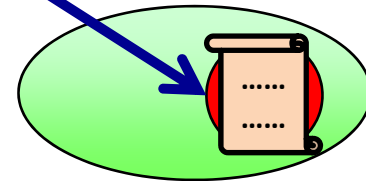
Clients



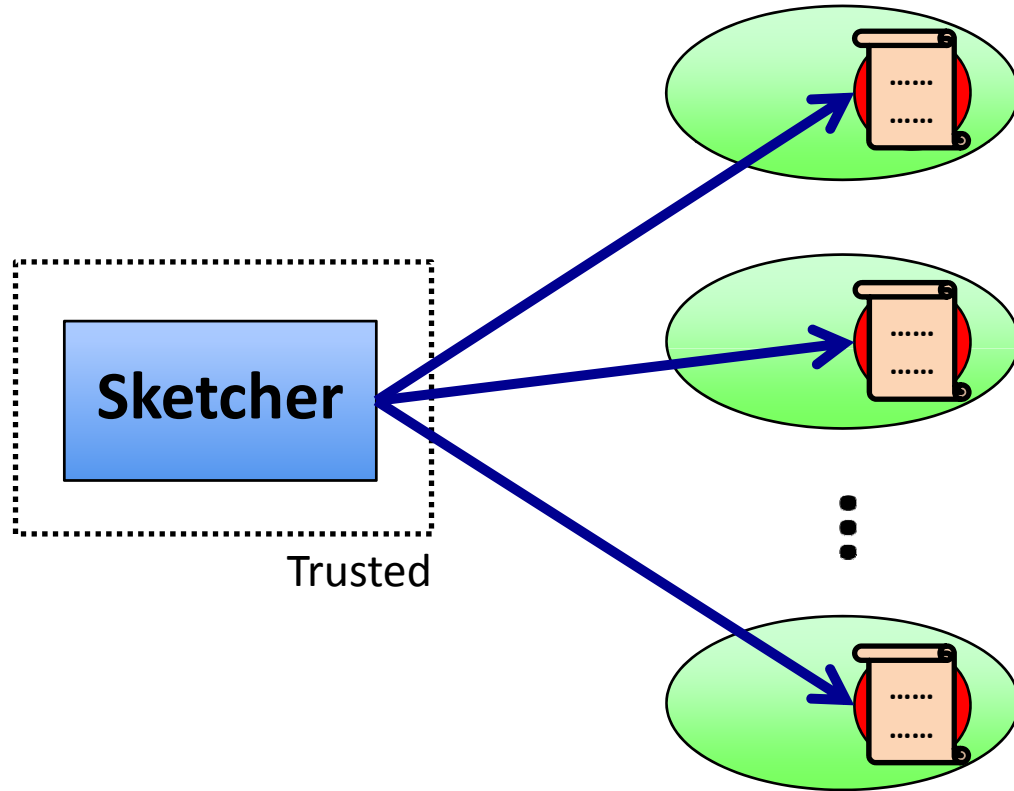
Trusted



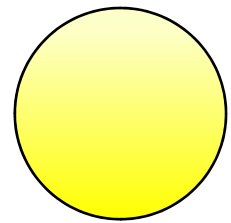
⋮



Replica Group



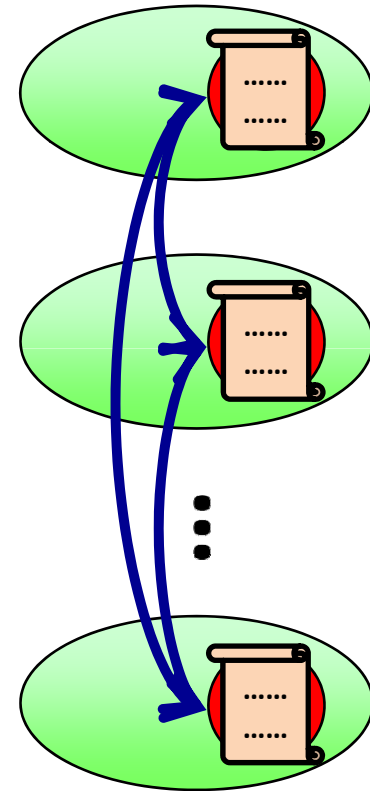
Executing a read



Clients

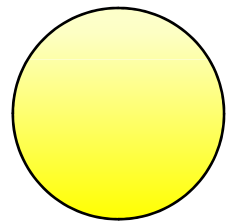


Trusted

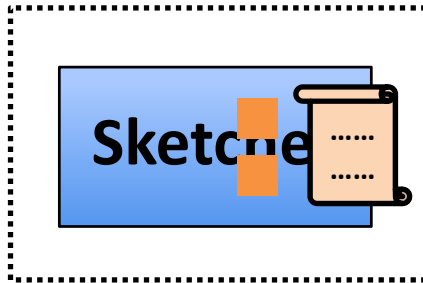


Replica Group

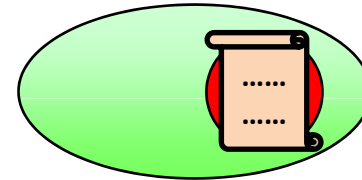
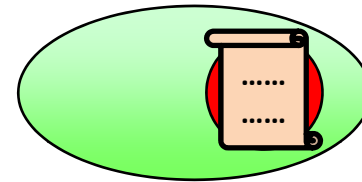
Executing a read



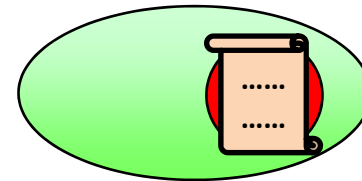
Clients



Trusted

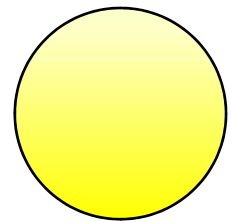


⋮

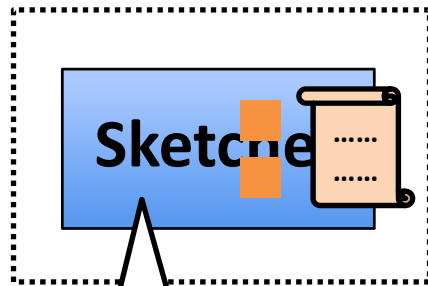


Replica Group

Executing a read

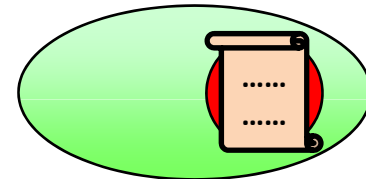
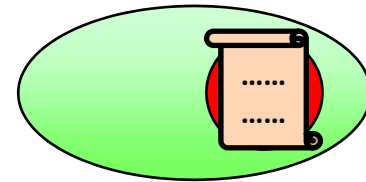


Clients

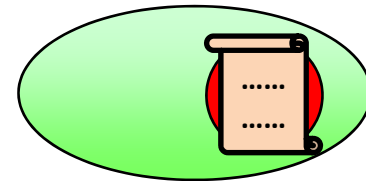


Trusted

Req	Resp
$s(q)$	■
⋮	⋮

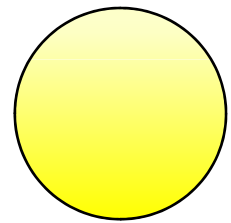


⋮

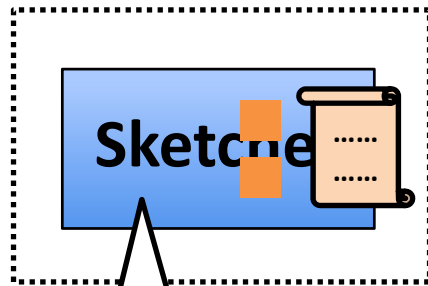


Replica Group

Executing a read

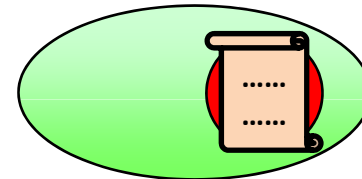
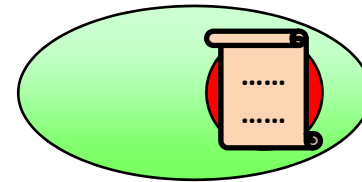


Clients

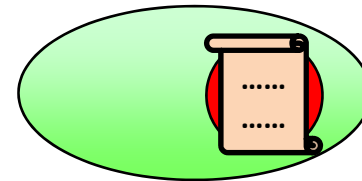


Trusted

Req	Resp
$s(q)$	■
⋮	⋮

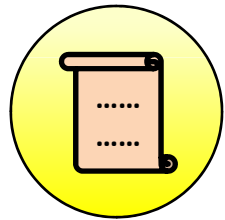


⋮



Replica Group

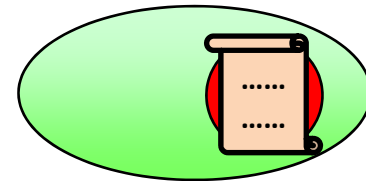
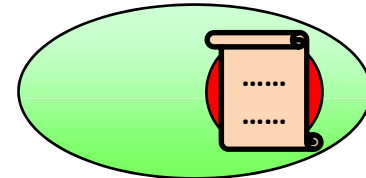
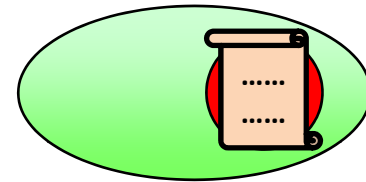
Prophecy



Clients



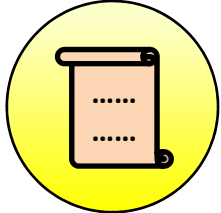
Trusted



Replica Group

Prophecy

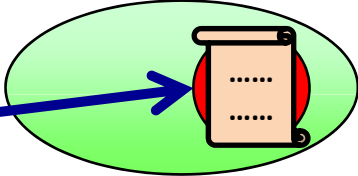
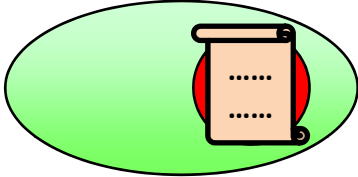
Fast, load-balanced reads



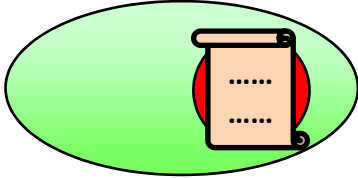
Clients



Trusted



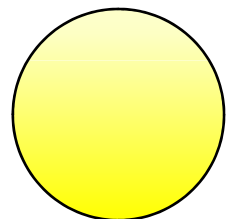
⋮



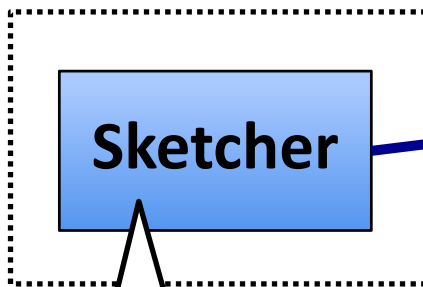
Replica Group

Prophecy

Fast reads may be stale

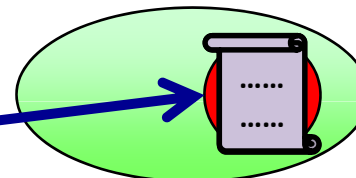
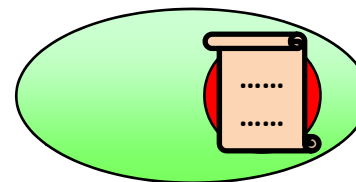


Clients

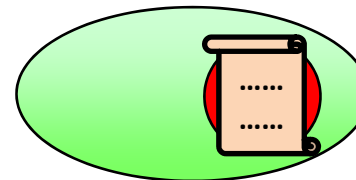


Trusted

Req	Resp
$s(q)$	■
⋮	⋮



⋮



Replica Group

Delay-once linearizability

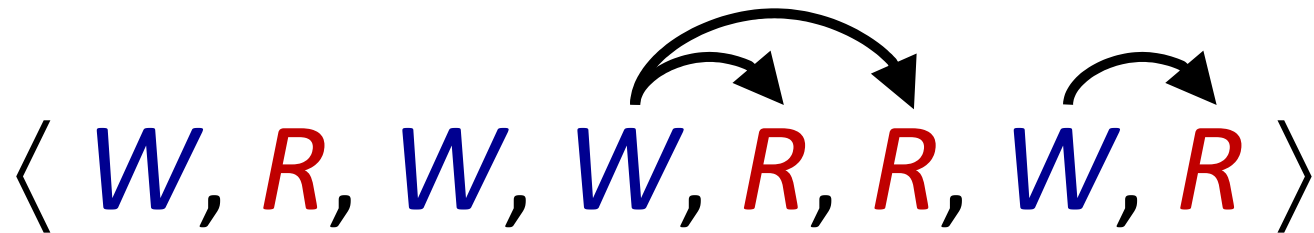
~~Delay once~~ linearizability

~~Delay once~~ linearizability

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

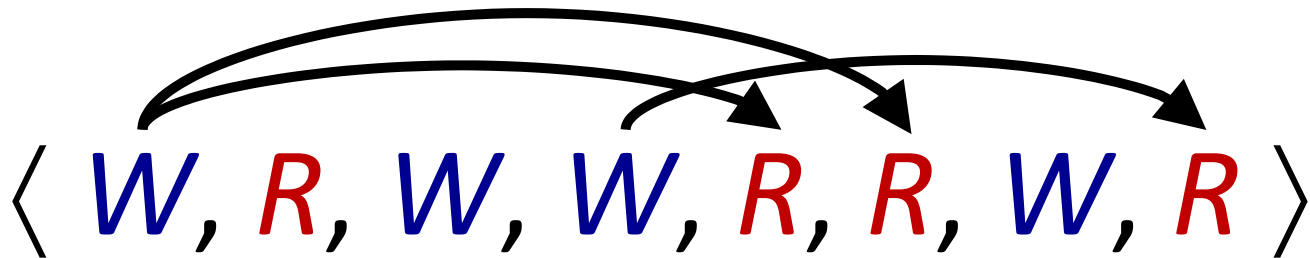
~~Delay once~~ linearizability

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~~

D-Prophecy

- ~~Modifies clients~~

Prophecy

- ~~Long lived sessions~~

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

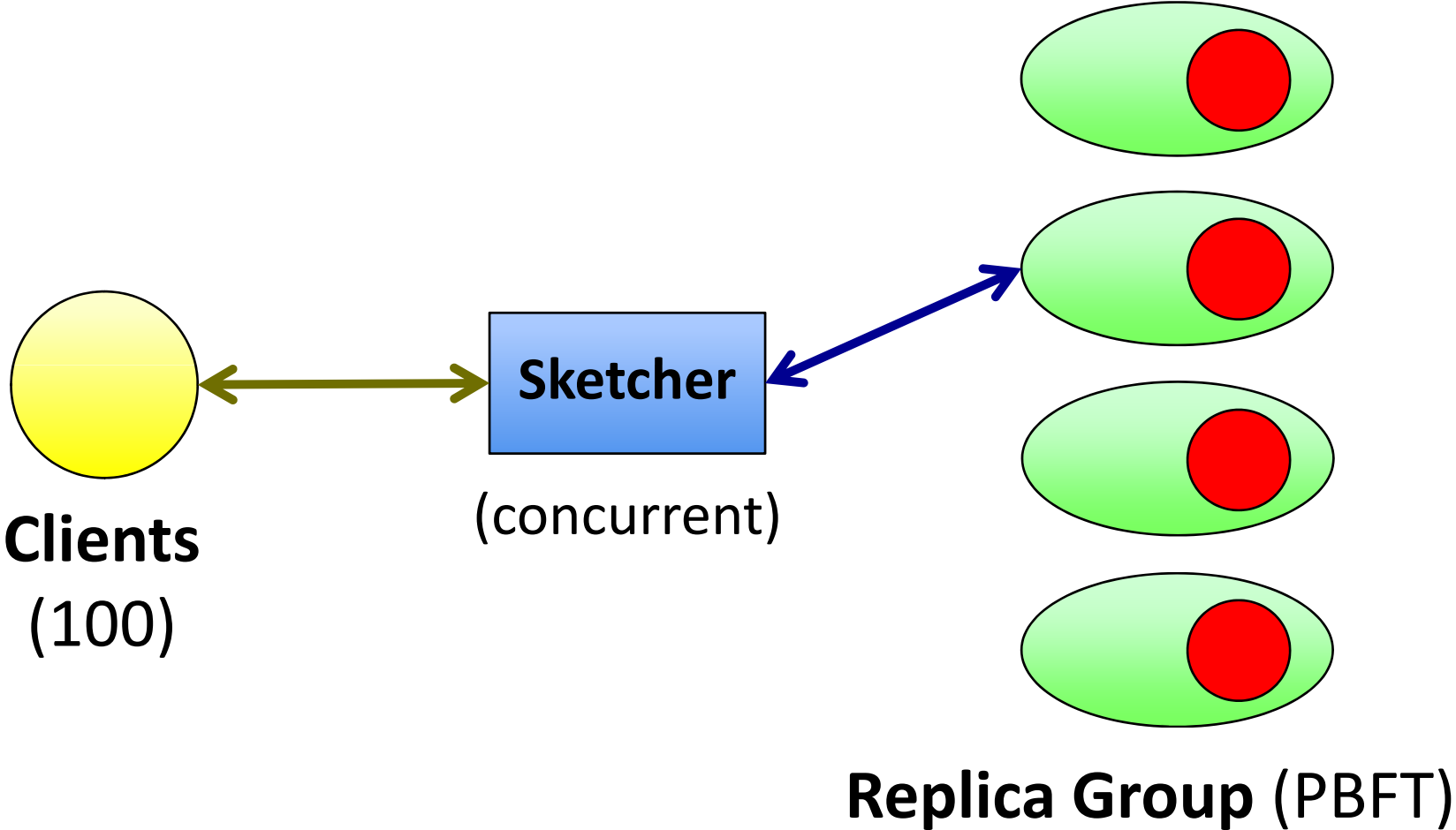
Evaluation

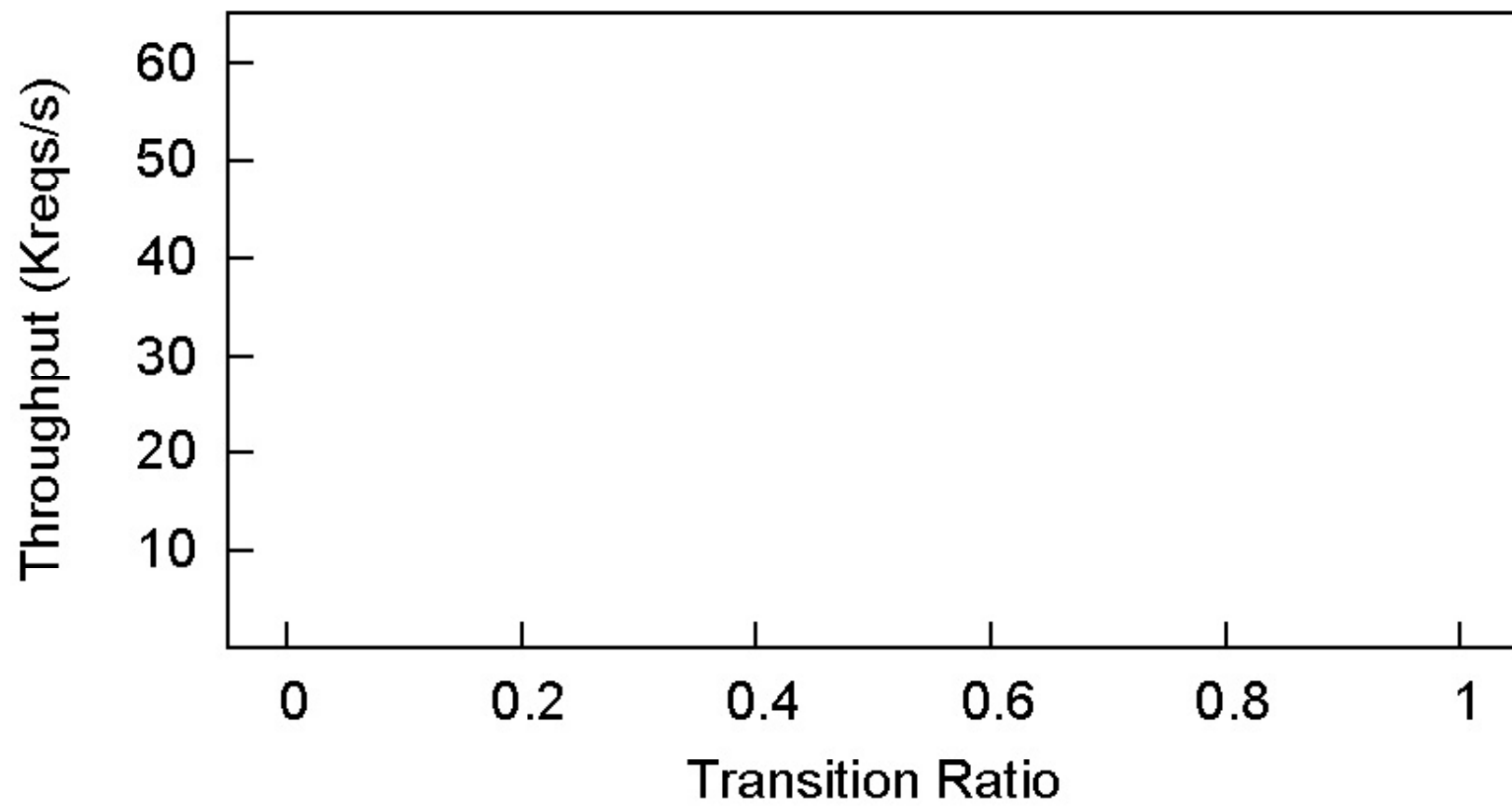
- Prophecy vs. proxied-PBFT
 - Proxied systems
- D-Prophecy vs. PBFT
 - Non-proxied systems

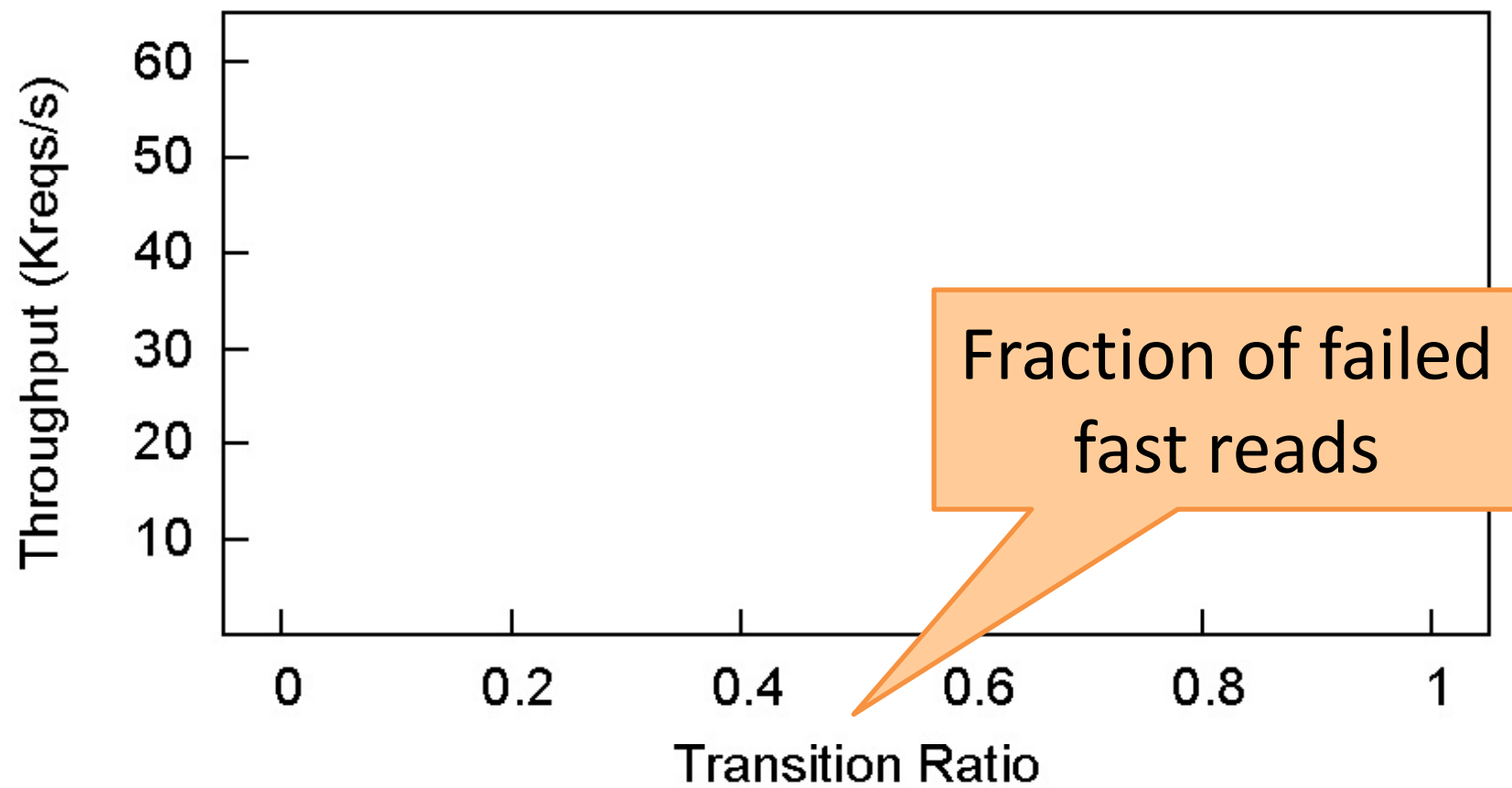
Evaluation

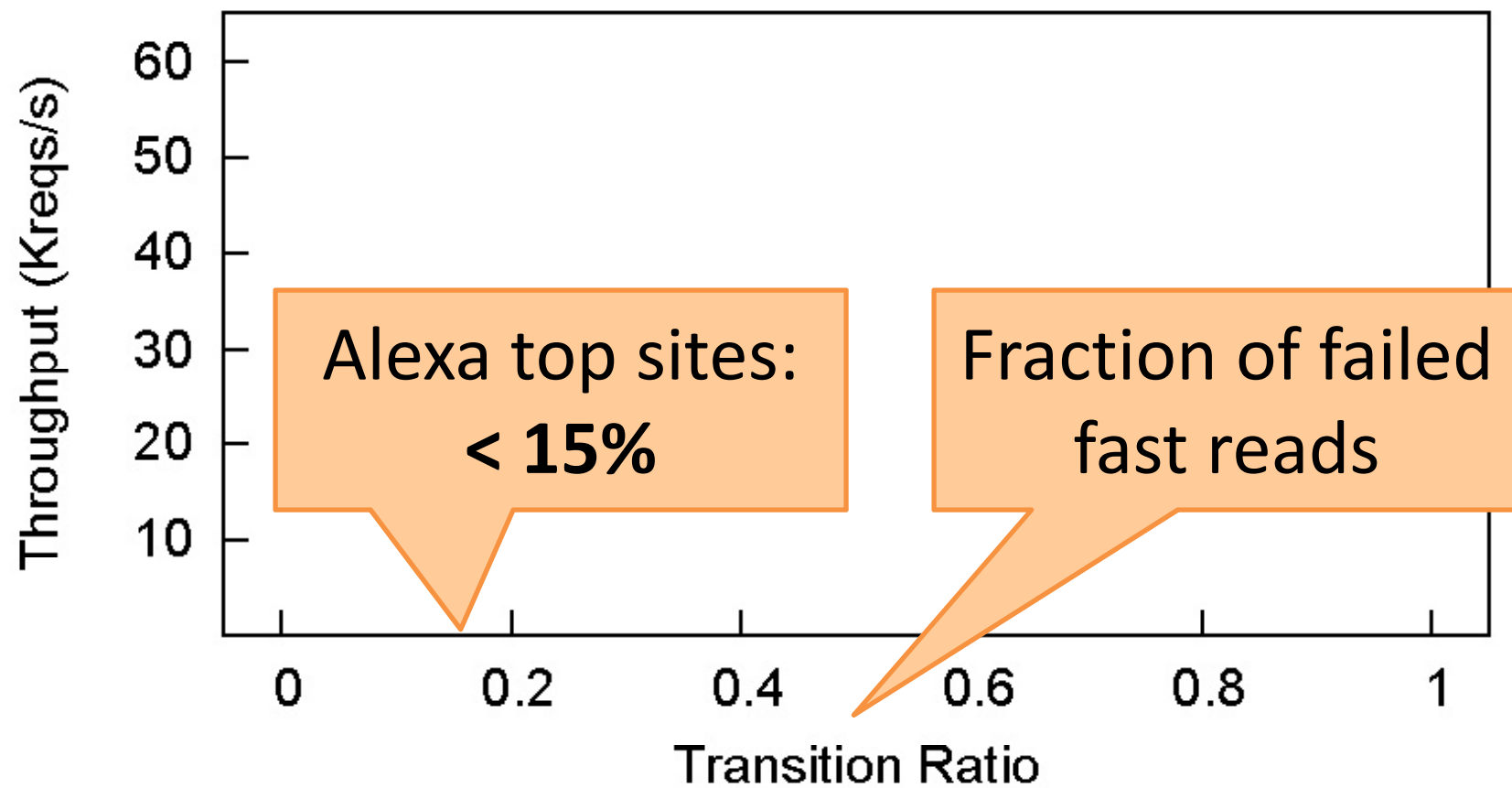
- Prophecy vs. proxied-PBFT
 - Proxied systems
- We will study:
 - Performance on “null” workloads
 - Performance with real replicated service
 - Where system bottlenecks, how to scale

Basic setup

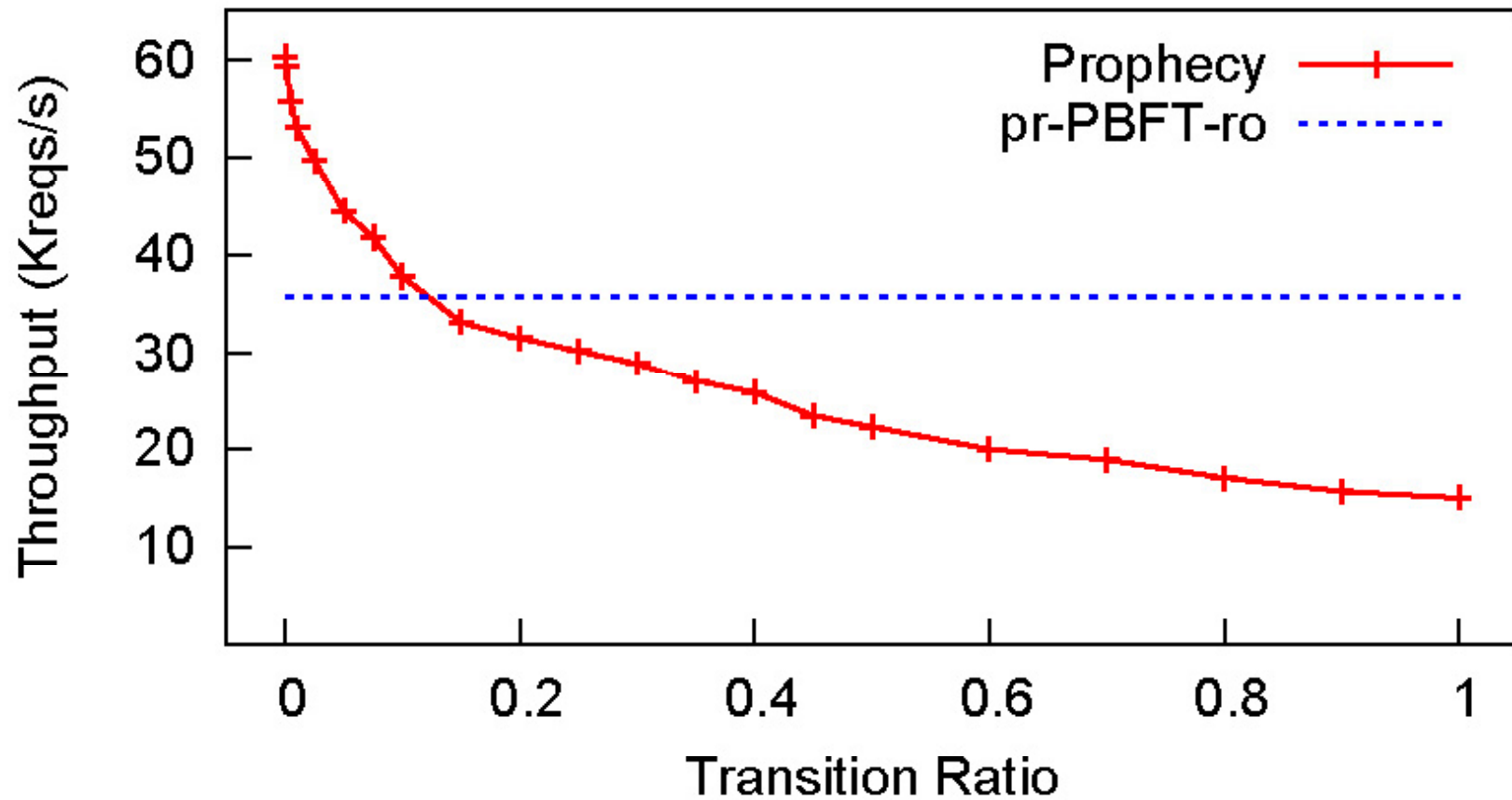




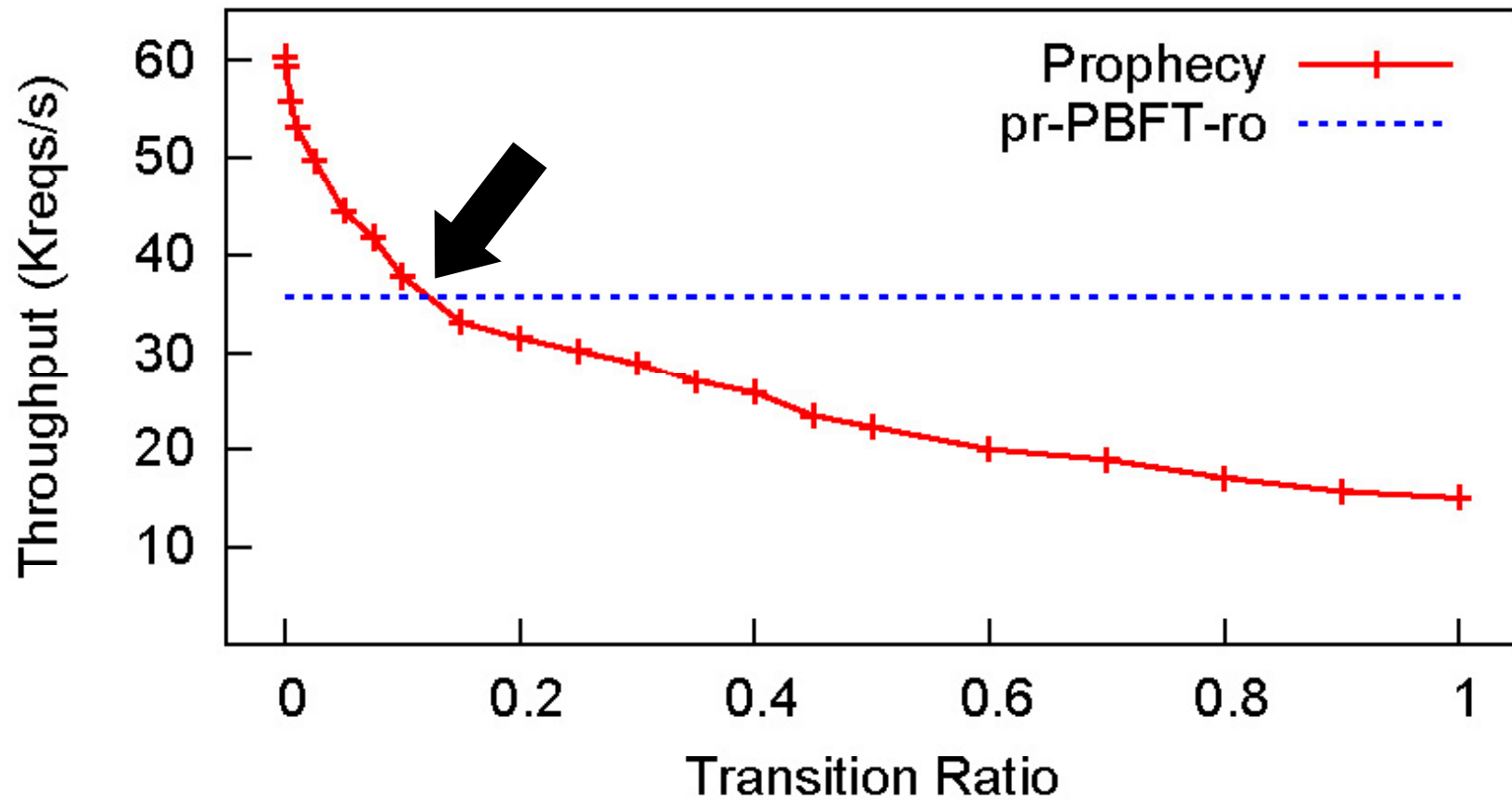




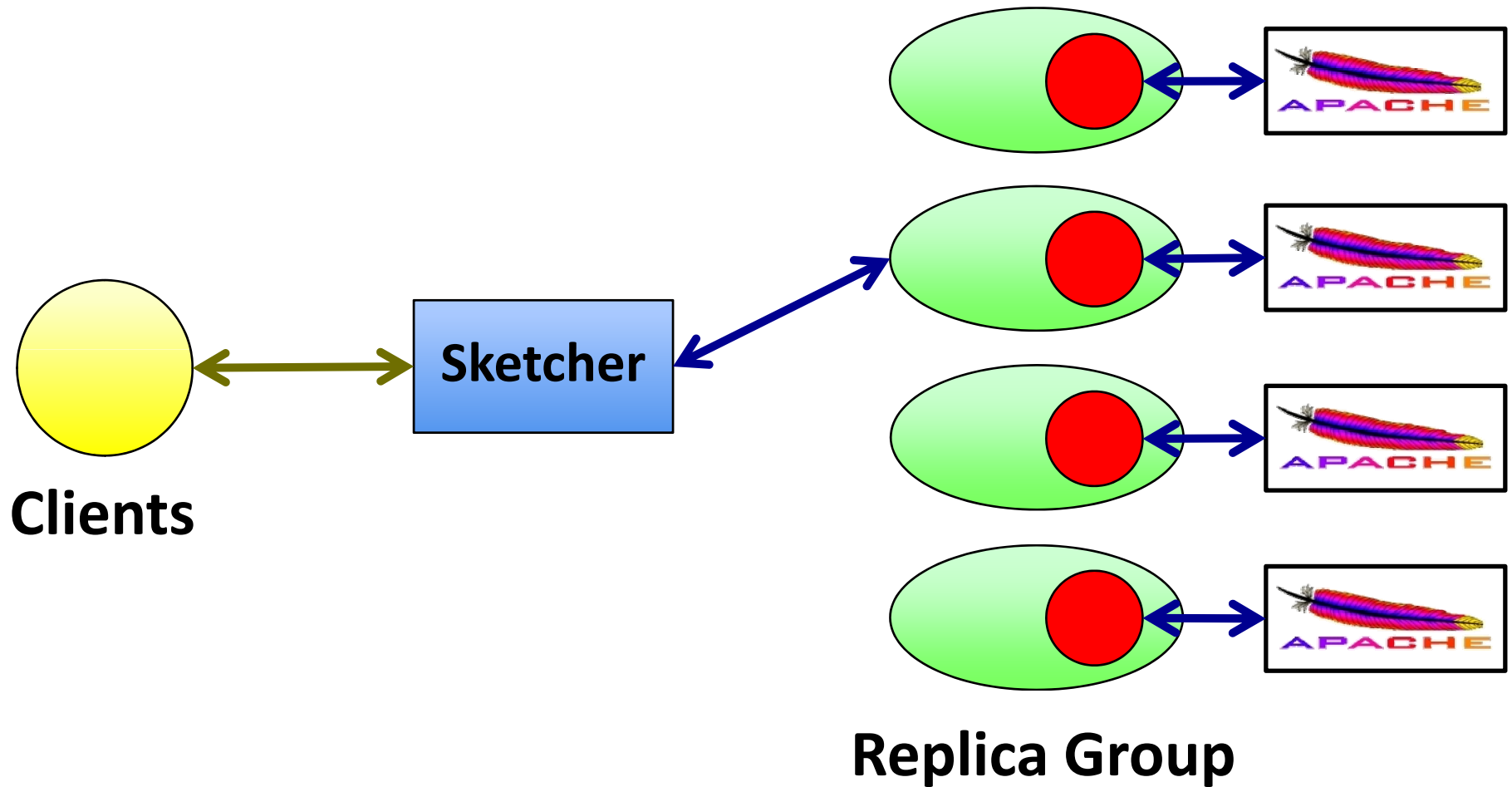
Small benefit on null reads



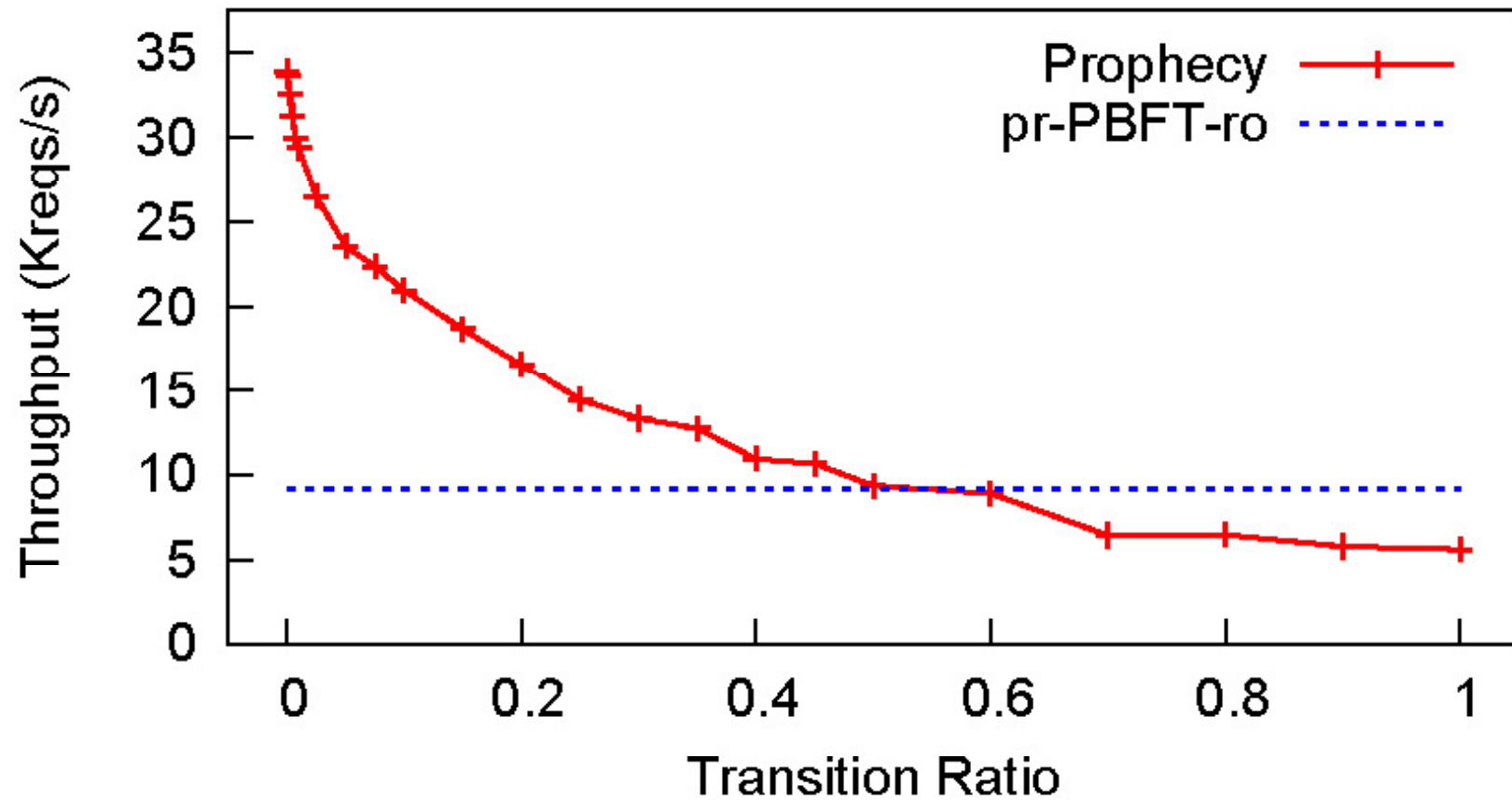
Small benefit on null reads



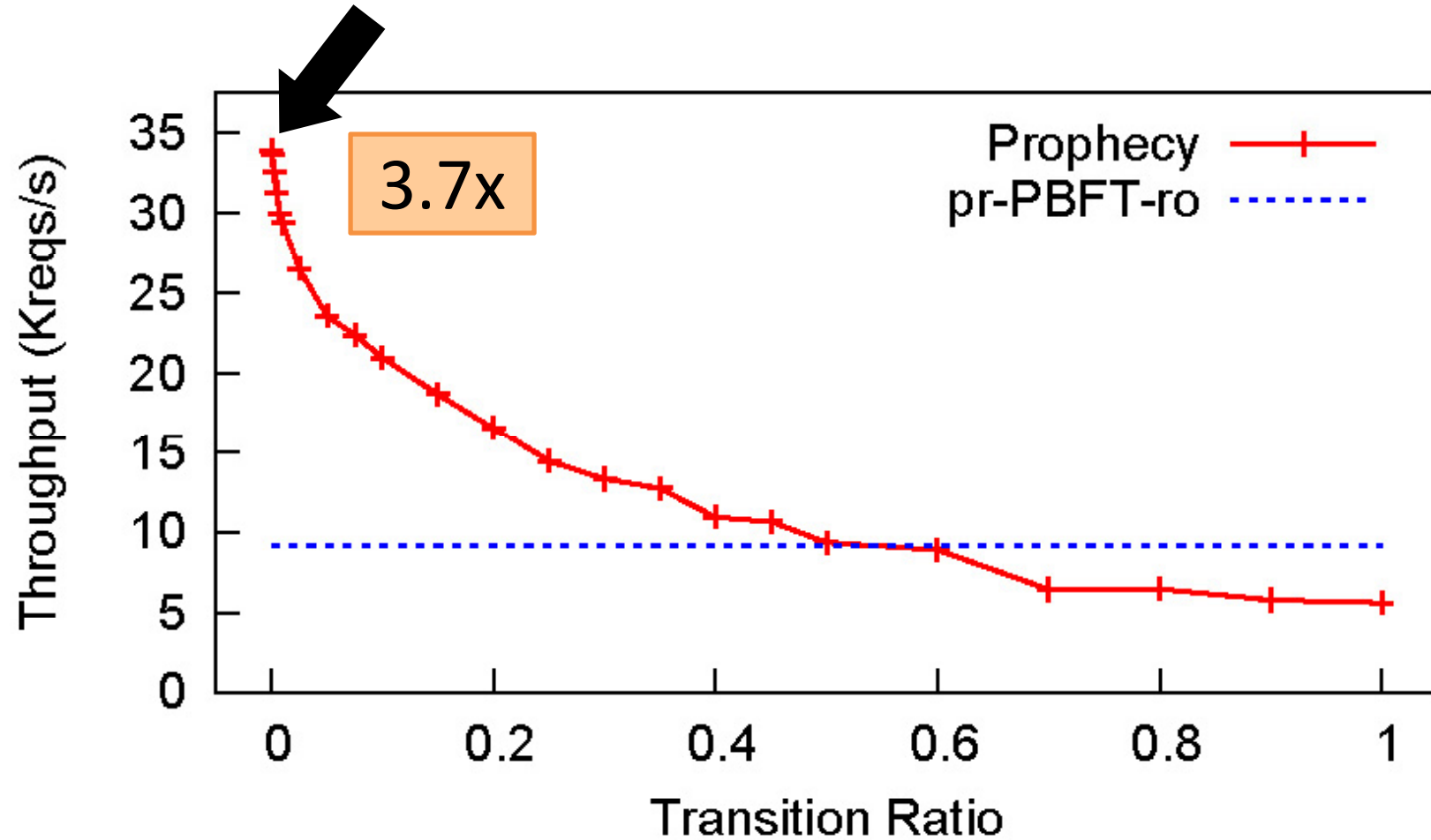
Apache webserver setup



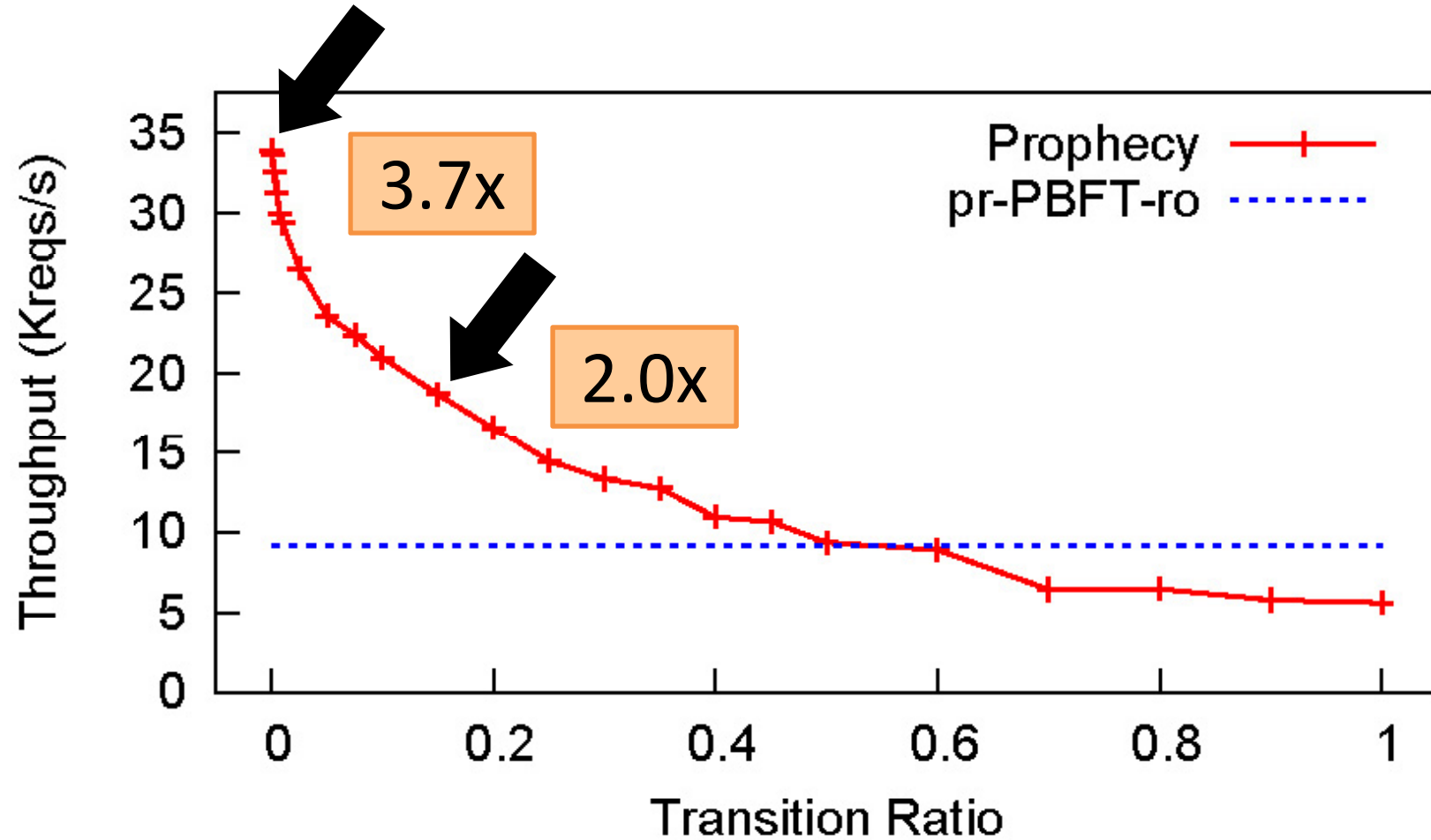
Large benefit on real workload



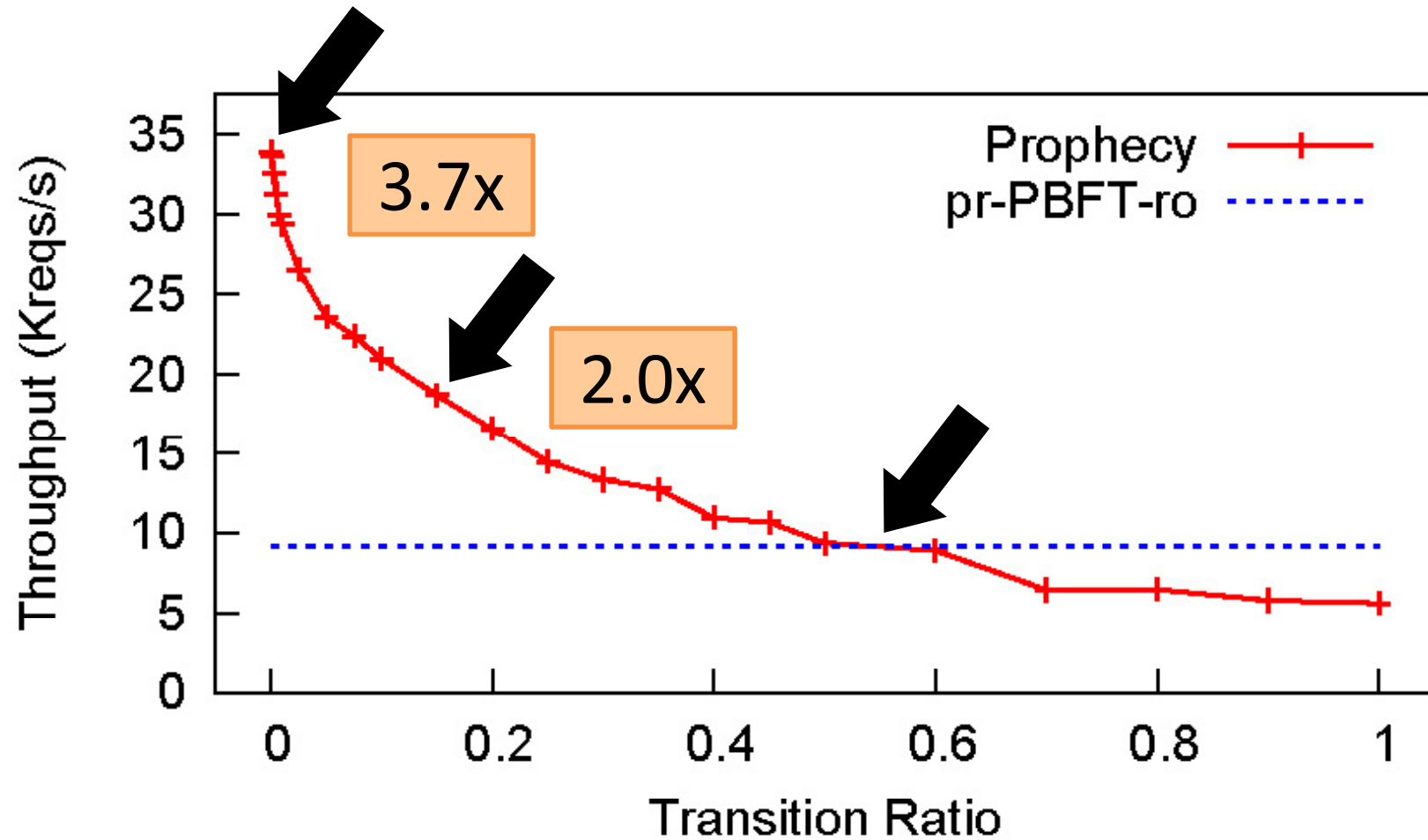
Large benefit on real workload



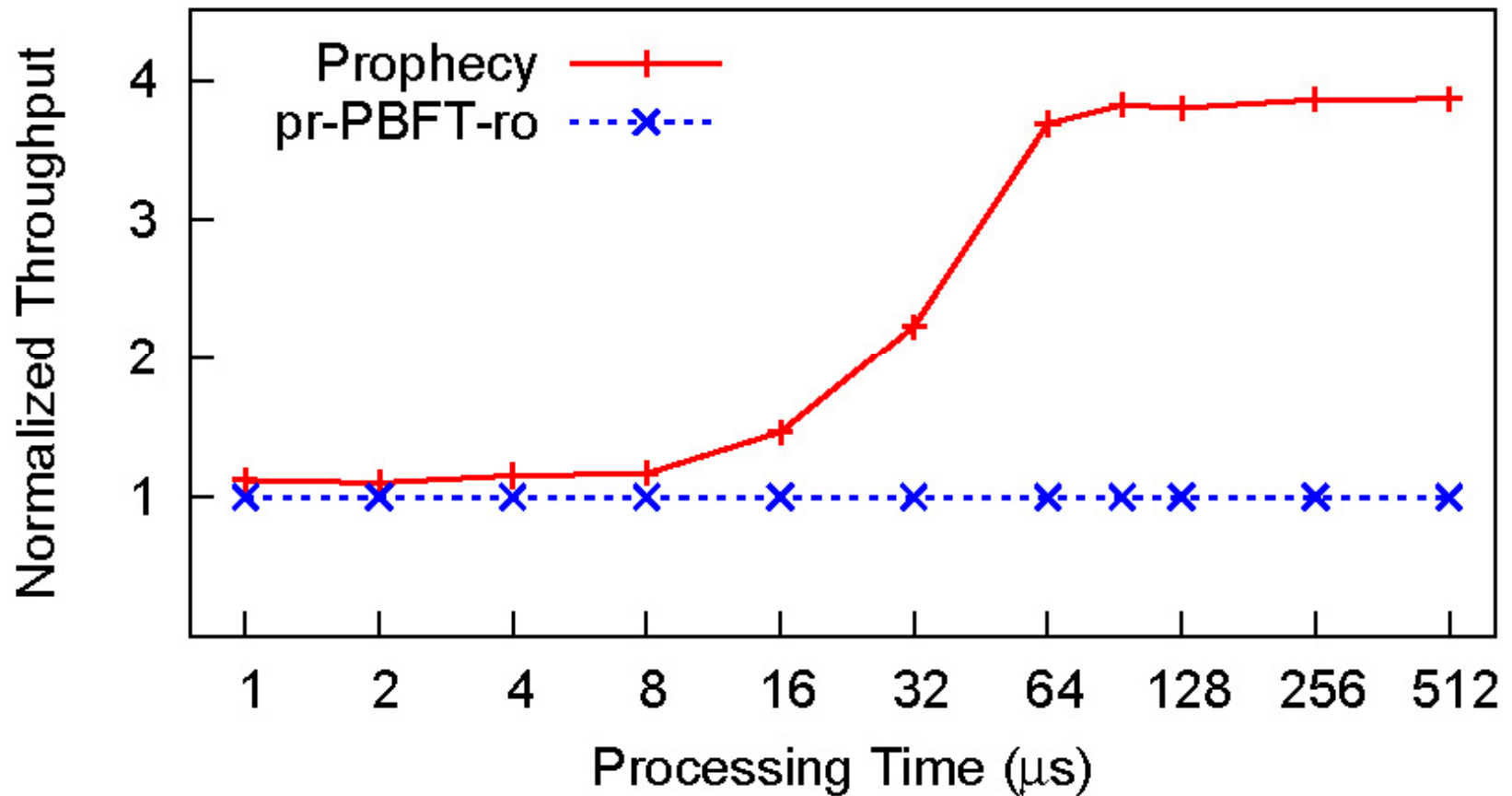
Large benefit on real workload



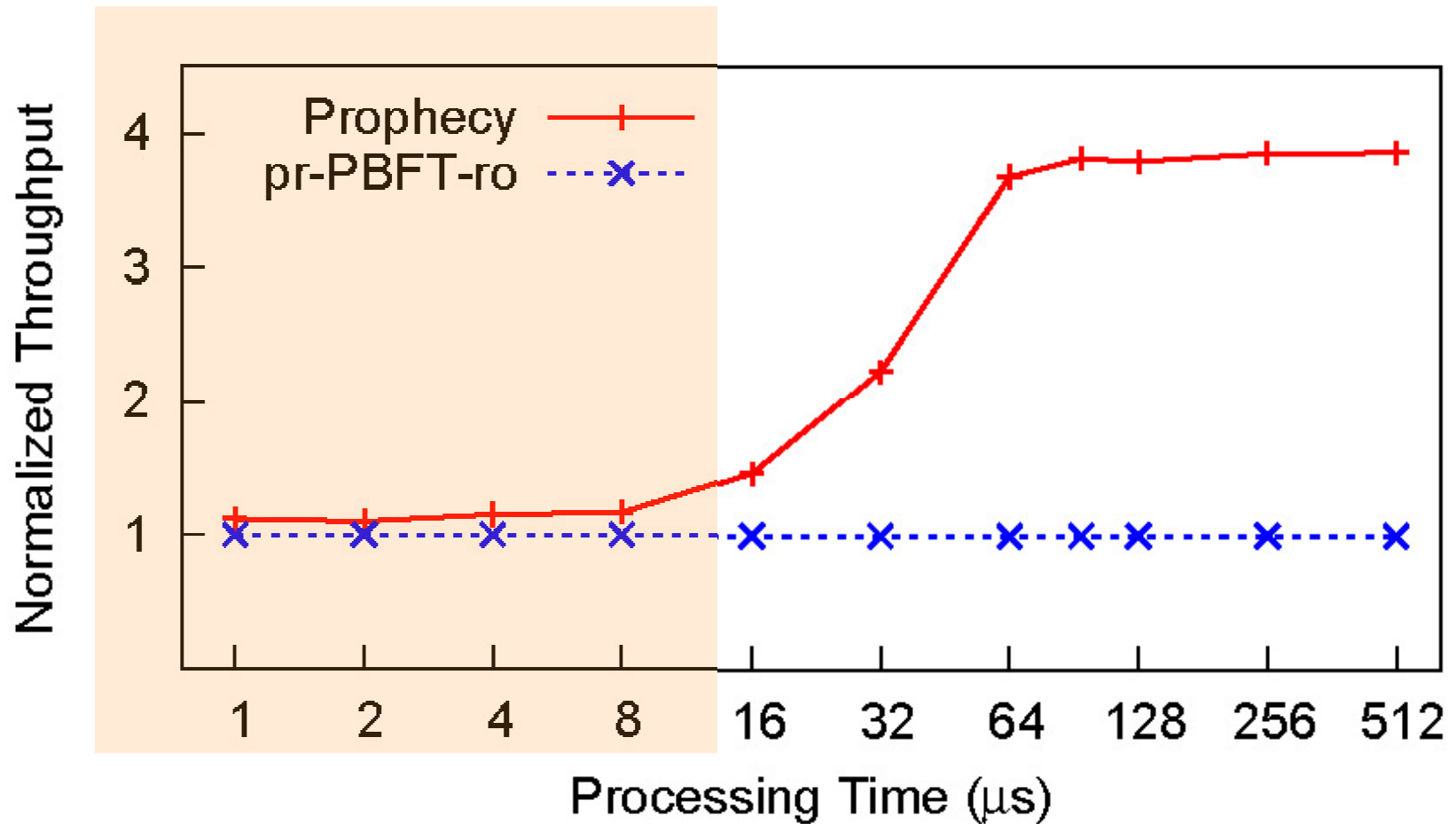
Large benefit on real workload



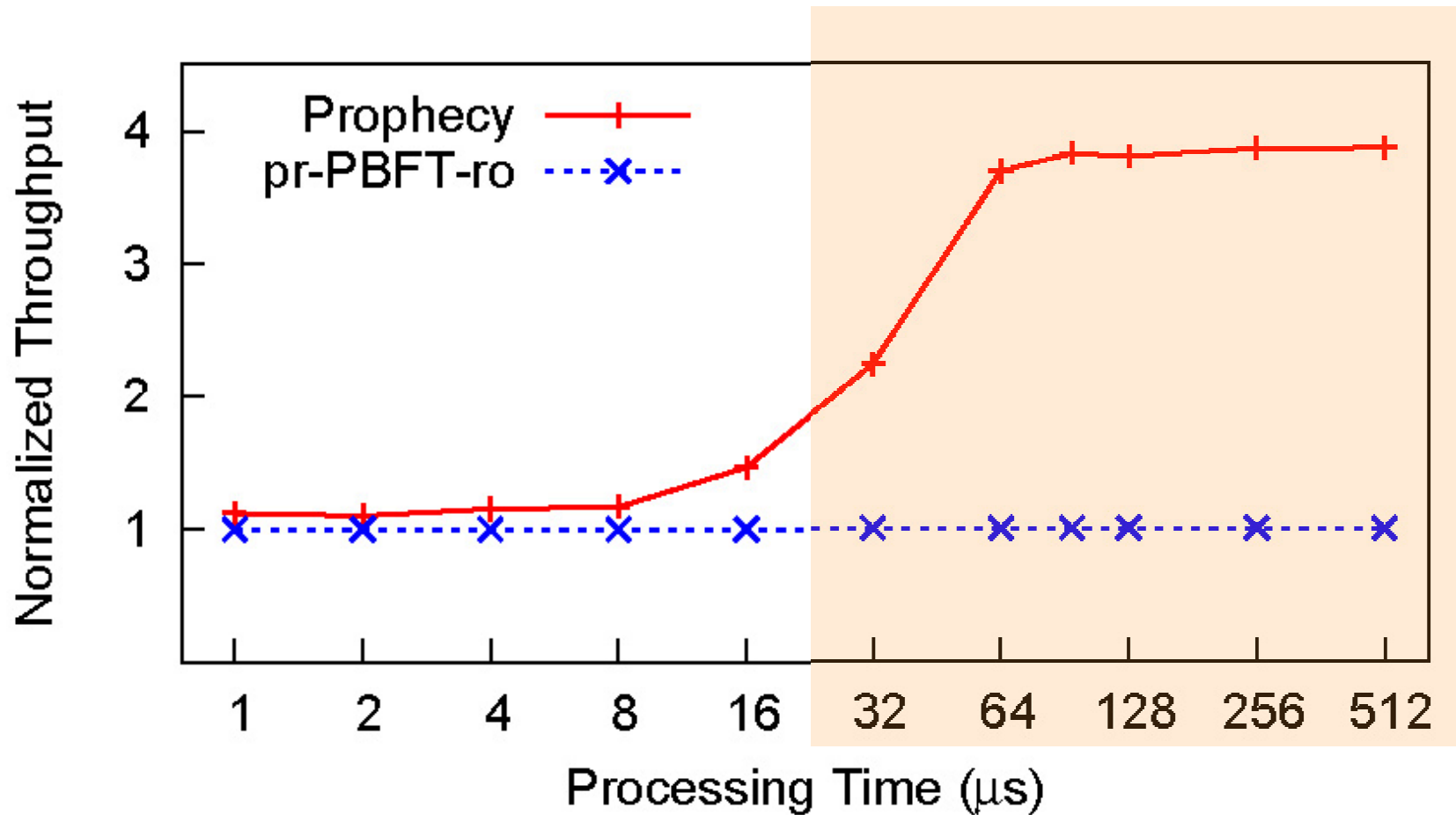
Benefit grows with work



Benefit grows with work

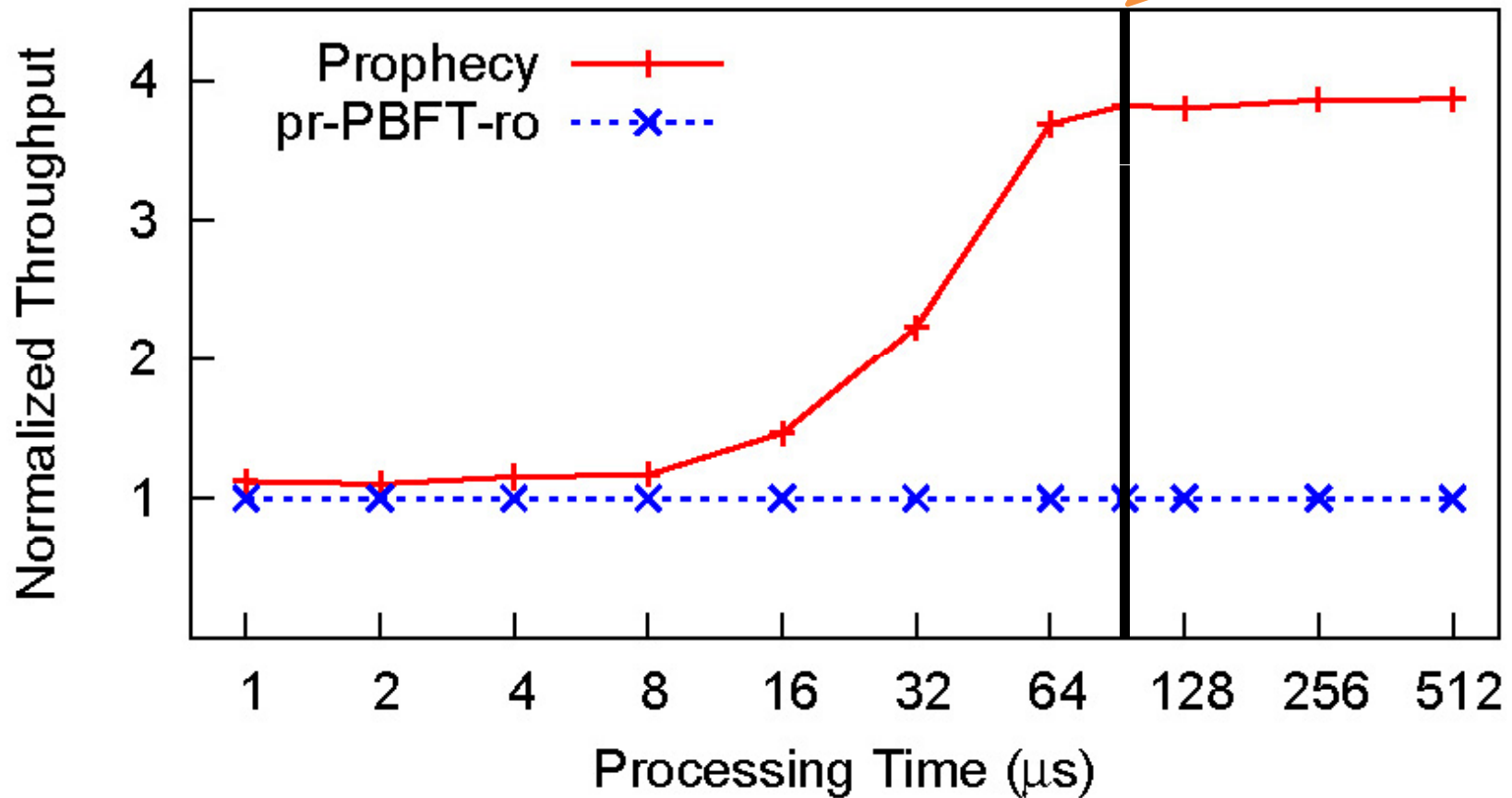


Benefit grows with work

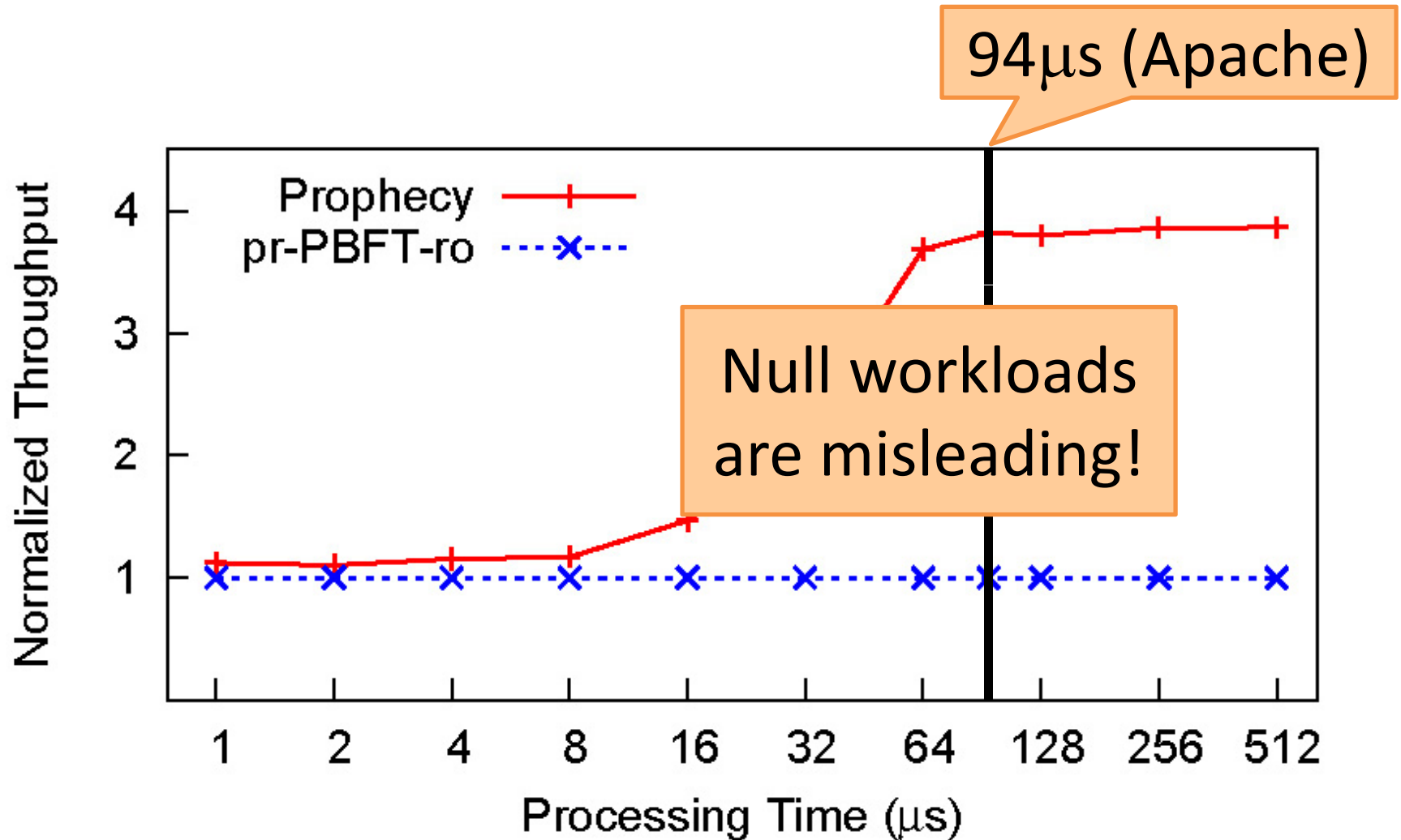


Benefit grows with work

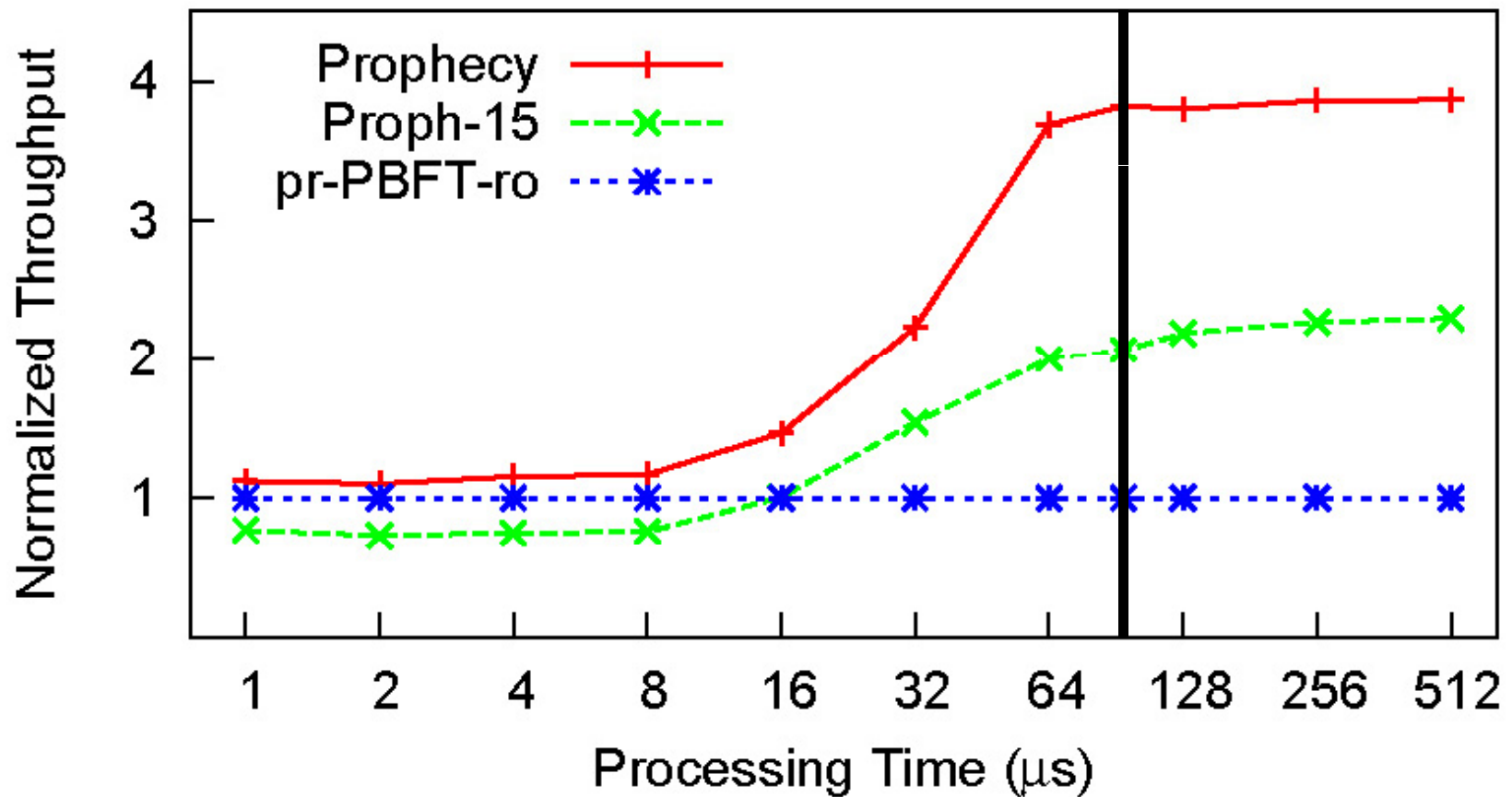
94 μ s (Apache)



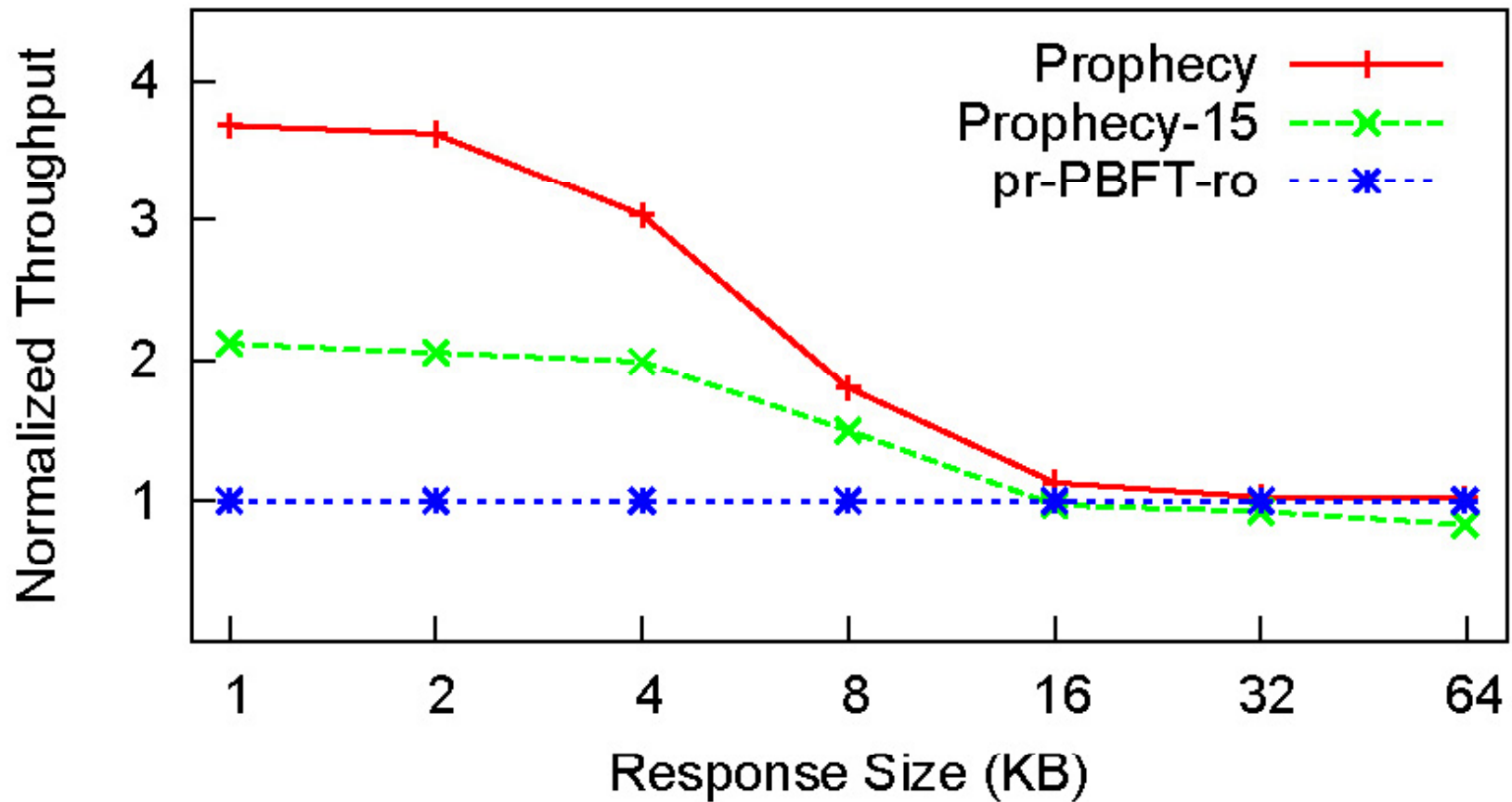
Benefit grows with work



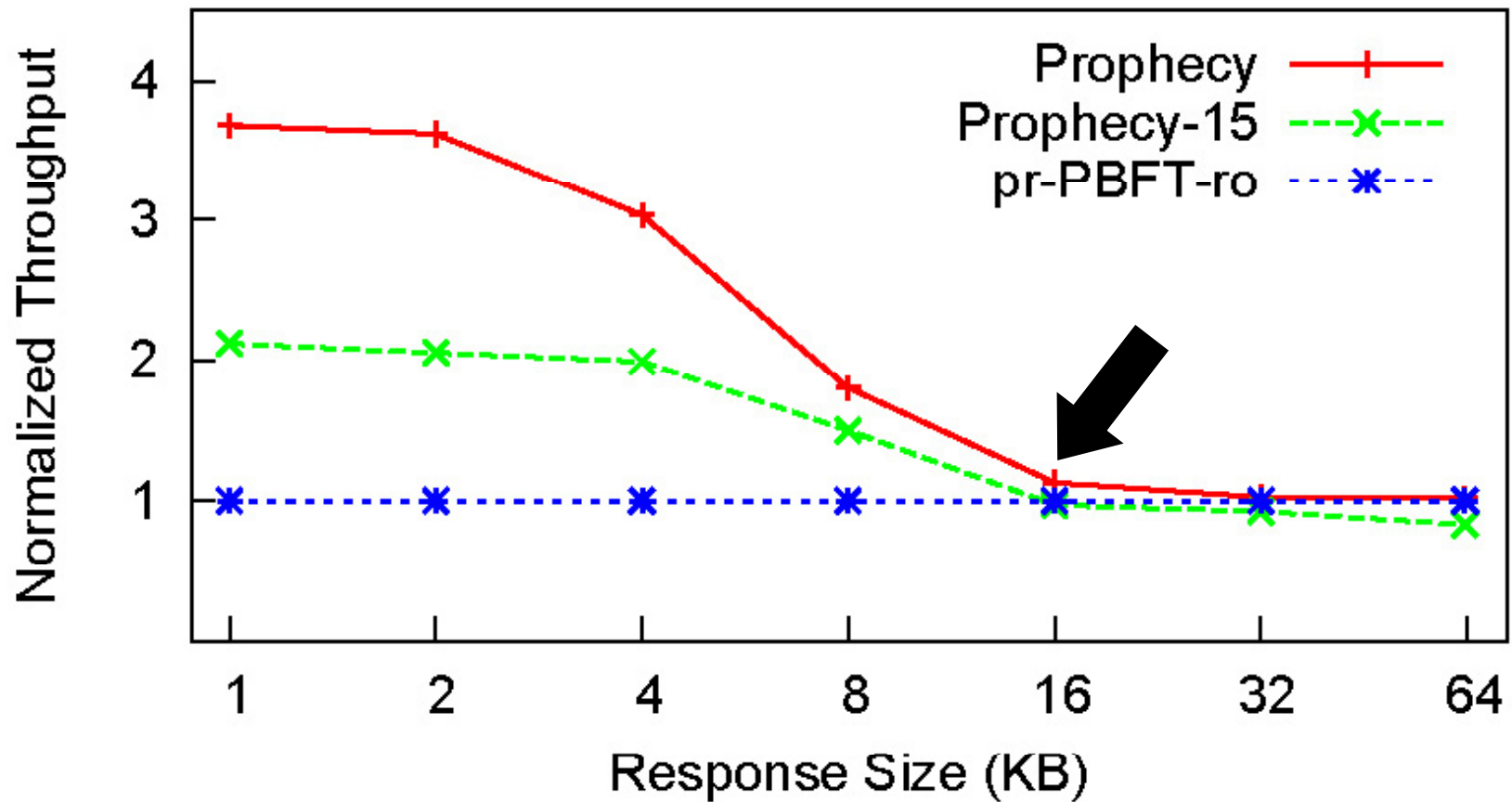
Benefit grows with work



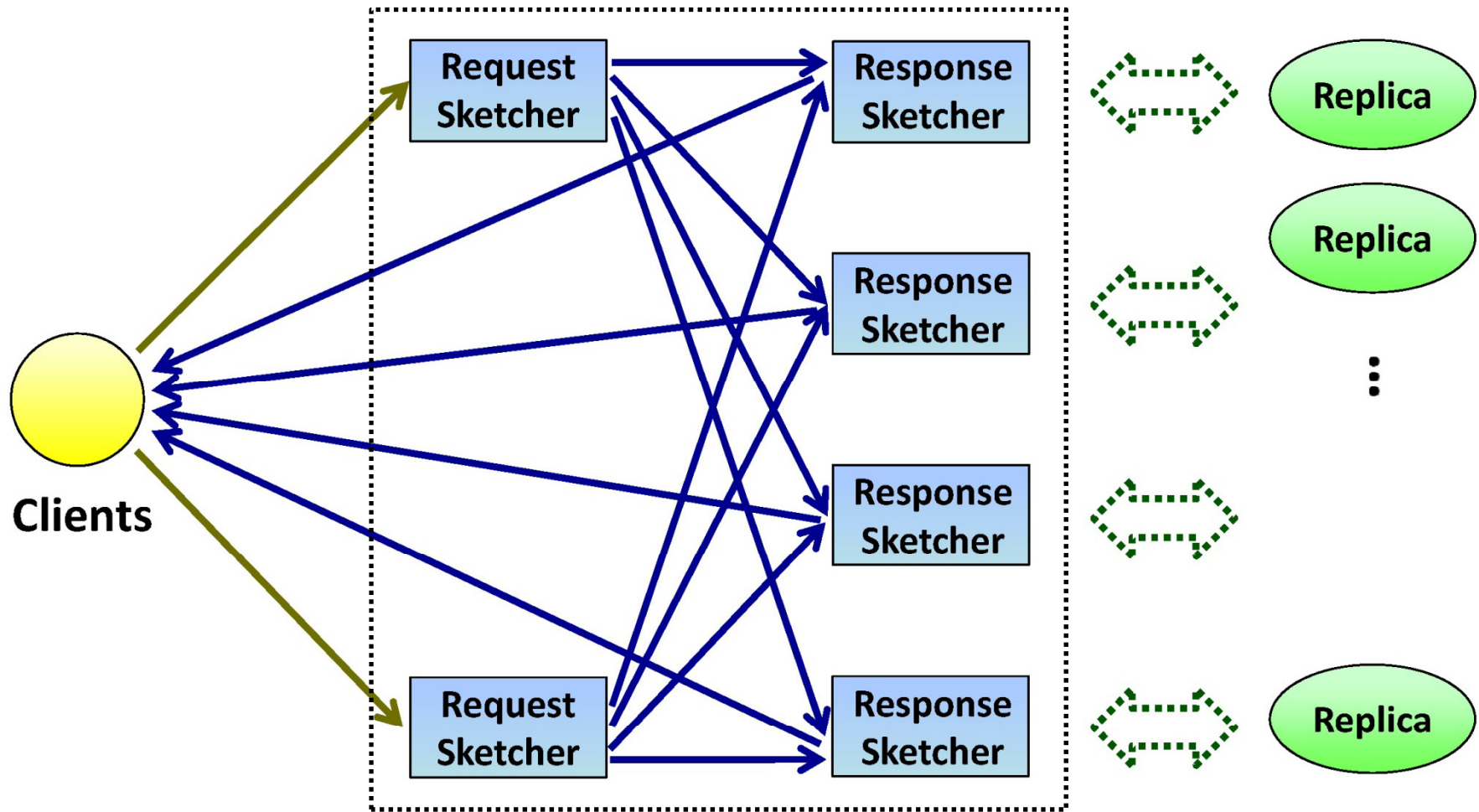
Single sketcher bottlenecks



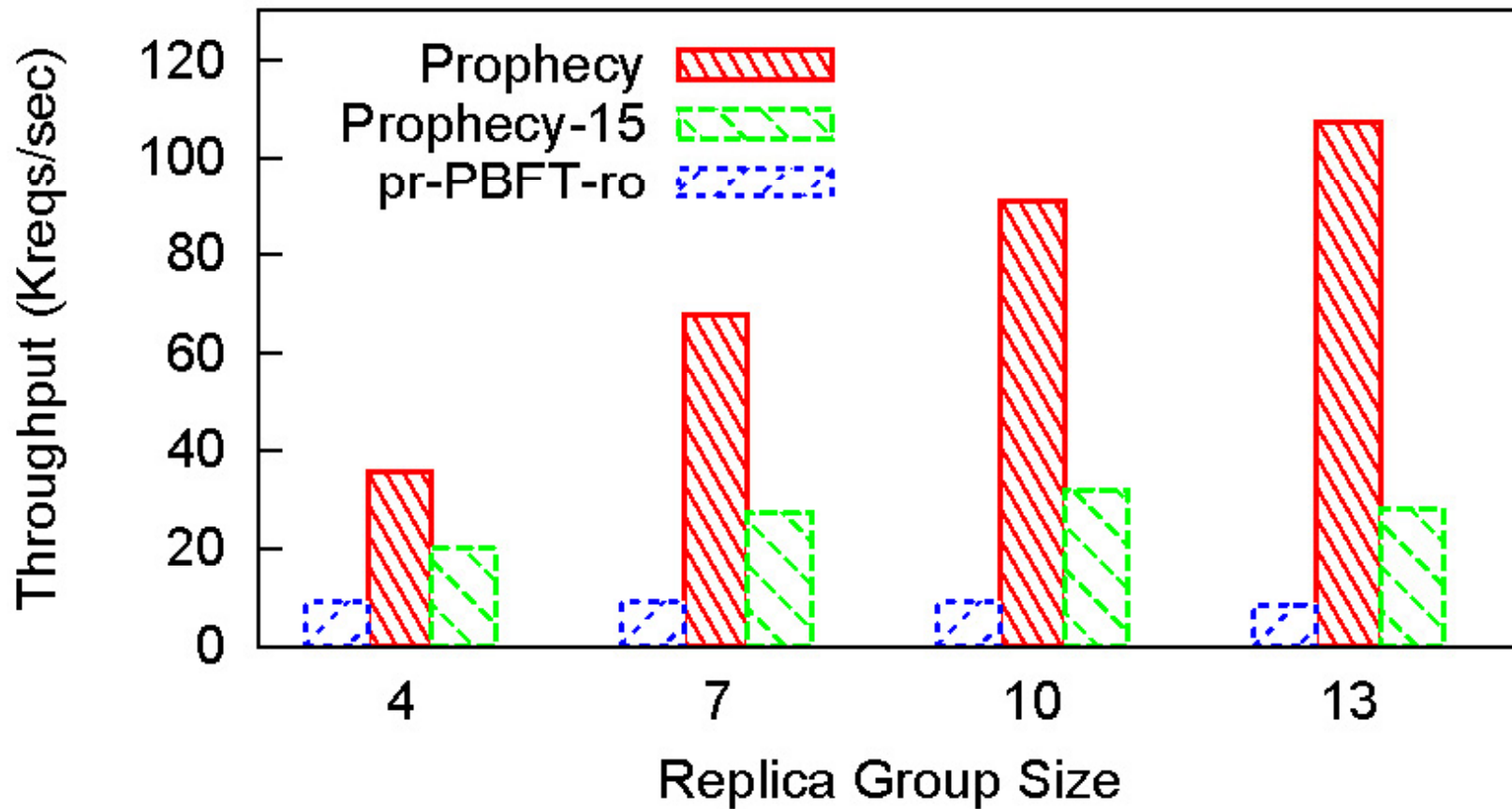
Single sketcher bottlenecks



Scaling out



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

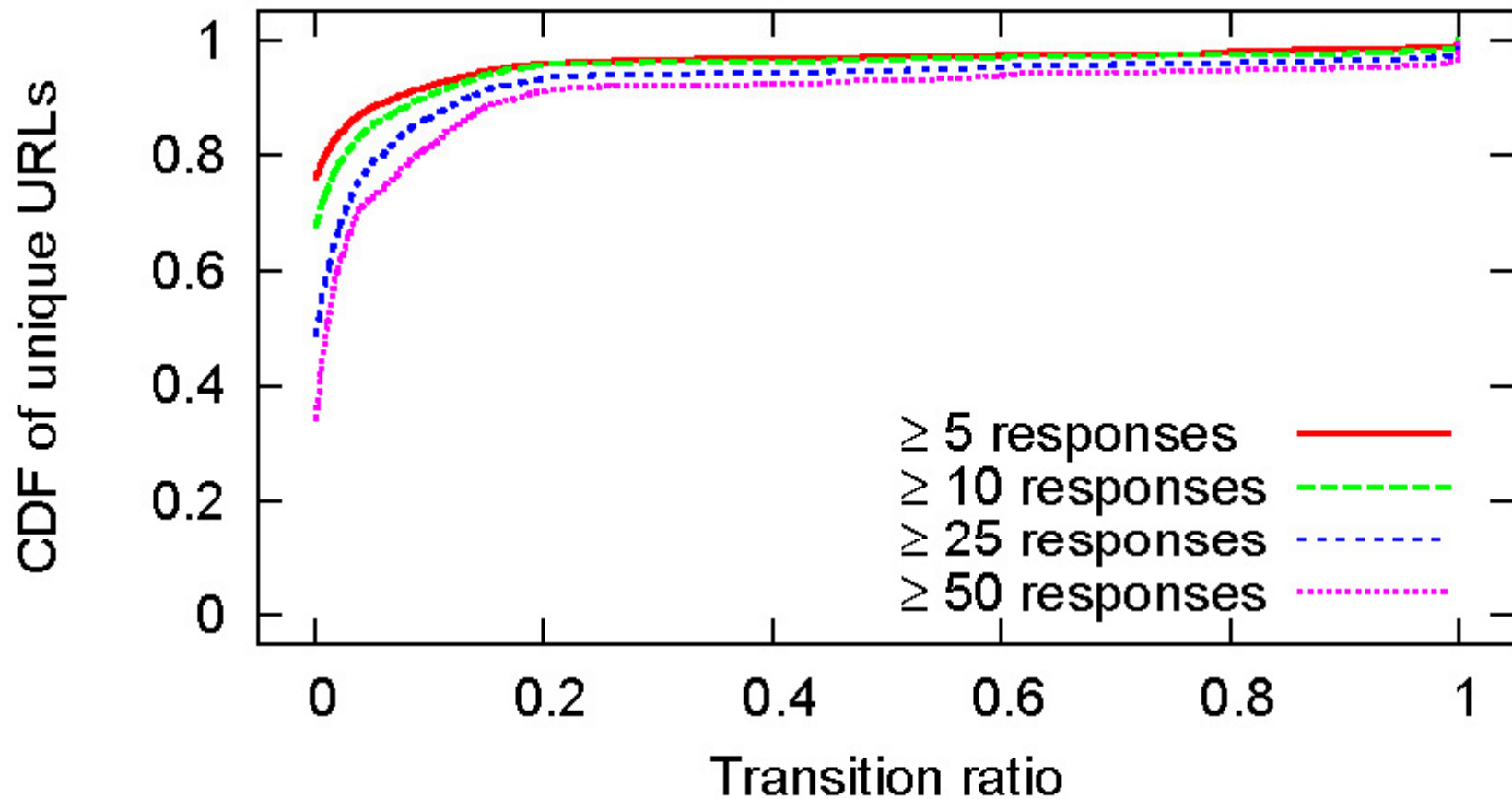
- Alexa top sites
- Access main page every 20 sec for 24 hrs



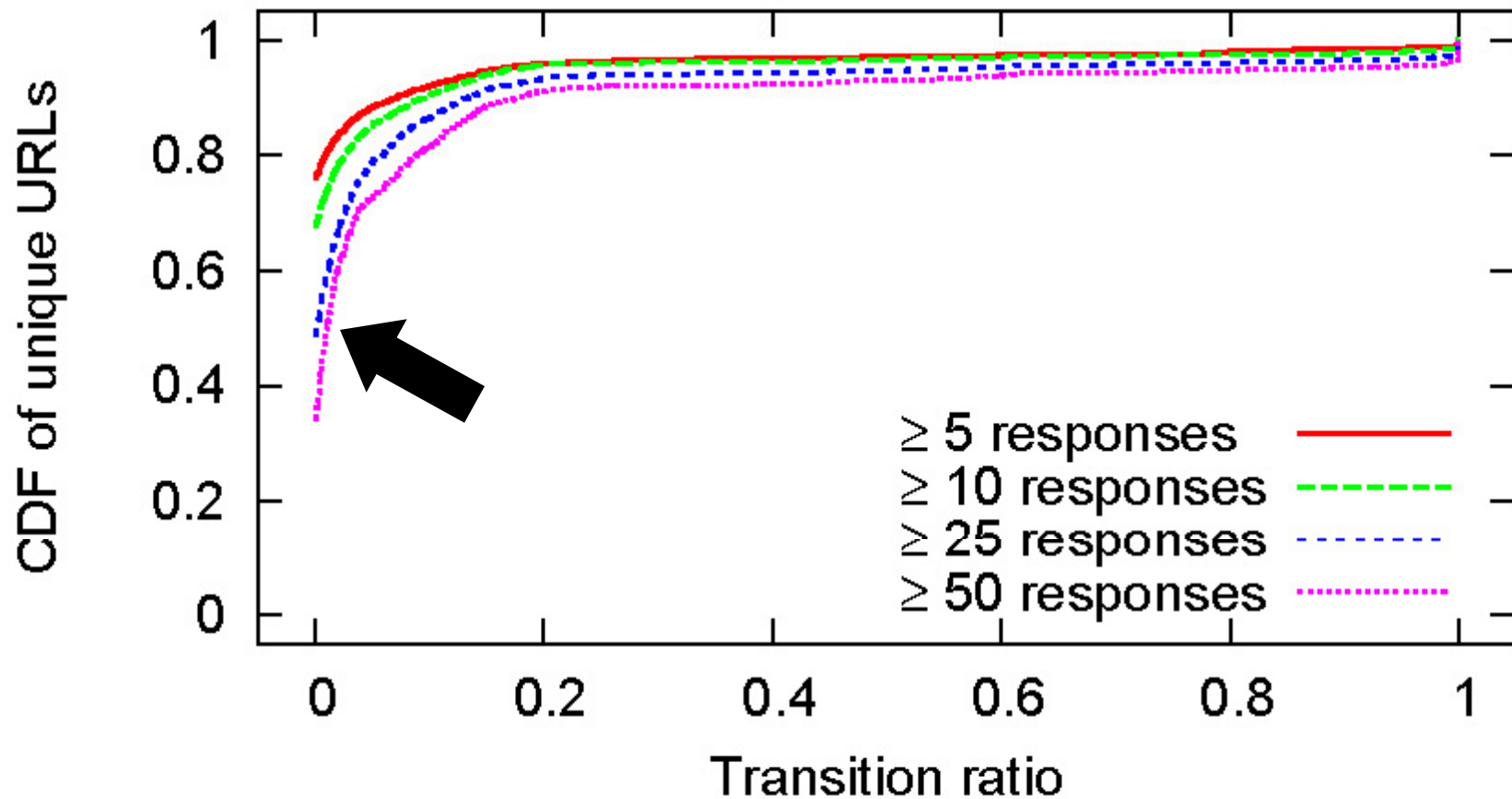
WIKIPEDIA
The Free Encyclopedia



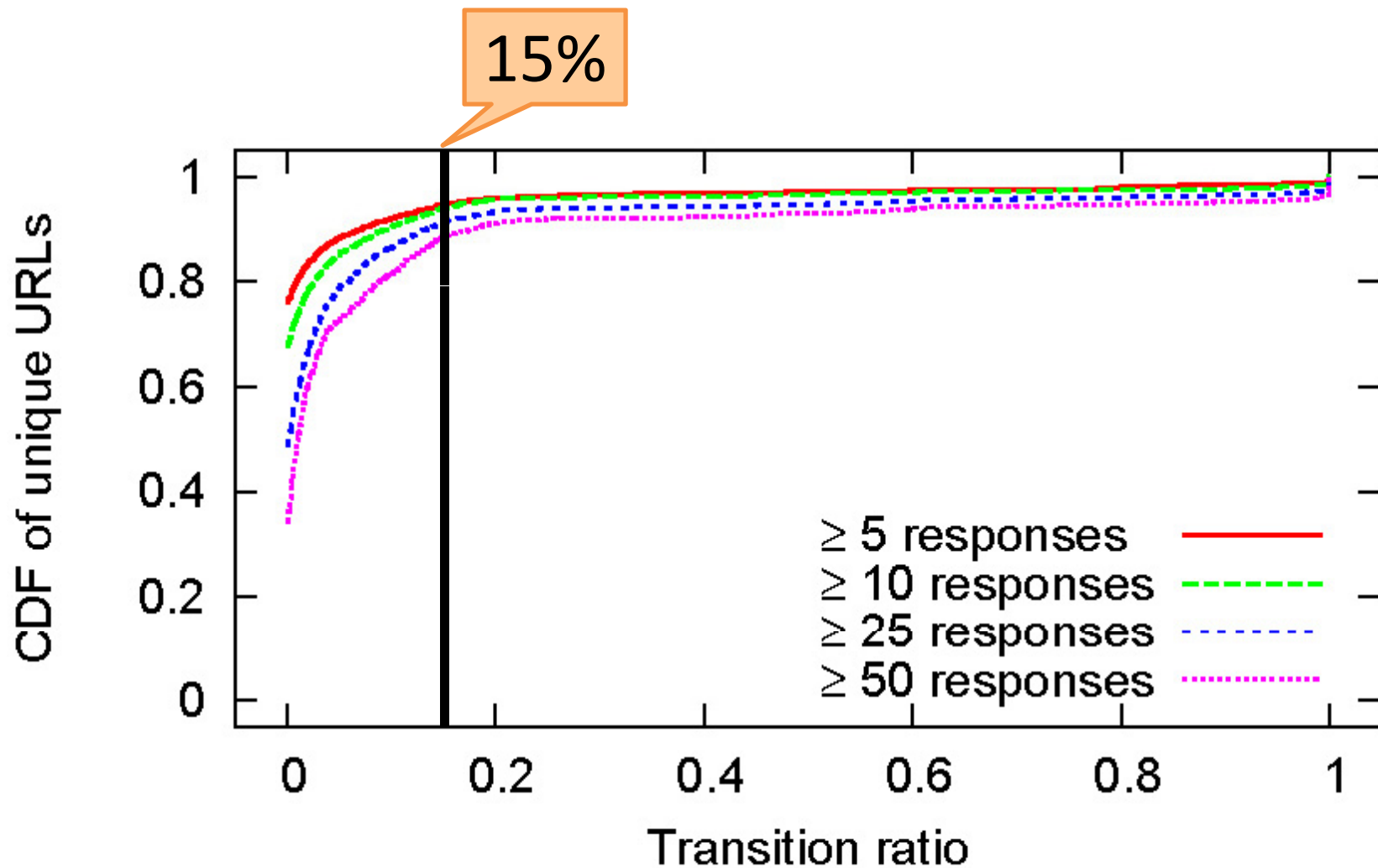
Mostly static content



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