Scalog: Seamless Reconfiguration and Total Order in a Scalable Shared Logs

NSDI'20

Cong Ding, David Chu, and Evan Zhao, Cornell University; Xiang Li, Alibaba Group;
Lorenzo Alvisi and **Robbert van Renesse**, Cornell University

Shared by Zevin at USTC-SYS Reading Group

USTC, CHINA ADSLAB

Lorenzo Alvisi

Tisch University Professor, Cornell University
在 cs.cornell.edu 的电子邮件经过验证 - 首页
Distributed Computing

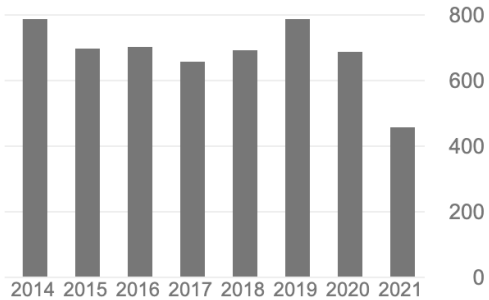
关注

创建我的个人资料

标题	引用次数	年份
A survey of rollback-recovery protocols in message-passing systems EN Elnozahy, L Alvisi, YM Wang, DB Johnson ACM Computing Surveys (CSUR) 34 (3), 375-408	2445	2002
Modeling the effect of technology trends on the soft error rate of combinational logic P Shivakumar, M Kistler, SW Keckler, D Burger, L Alvisi Proceedings International Conference on Dependable Systems and Networks, 389-398	1903	2002
Zyzzyva: speculative byzantine fault tolerance R Kotla, L Alvisi, M Dahlin, A Clement, E Wong Proceedings of twenty-first ACM SIGOPS symposium on Operating systems …	752	2007
Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. A Clement, EL Wong, L Alvisi, M Dahlin, M Marchetti NSDI 9, 153-168	460	2009
Separating agreement from execution for Byzantine fault tolerant services J Yin, JP Martin, A Venkataramani, L Alvisi, M Dahlin Proceedings of the nineteenth ACM symposium on Operating systems principles …	443	2003
BAR fault tolerance for cooperative services AS Aiyer, L Alvisi, A Clement, M Dahlin, JP Martin, C Porth Proceedings of the twentieth ACM symposium on Operating systems principles …	377	2005
Depot: Cloud storage with minimal trust P Mahajan, S Setty, S Lee, A Clement, L Alvisi, M Dahlin, M Walfish ACM Transactions on Computer Systems (TOCS) 29 (4), 1-38	373	2011
Fast byzantine consensus JP Martin, L Alvisi IEEE Transactions on Dependable and Secure Computing 3 (3), 202-215	369	2006
Message logging: Pessimistic, optimistic, causal, and optimal L Alvisi, K Marzullo IEEE Transactions on Software Engineering 24 (2), 149-159	335	1998

引用次数 查看全部

	总计	2016 年至今
引用	13539	3984
h 指数	50	29
i10 指数	93	51



可公开访问的出版物数量 查看全部

0 篇文章 9 篇文章

无法查看的文章 可查看的文章

根据资金授权书

合著作者 查看全部


M Dahlin
Professor of Computer Science …


Keith Marzullo
Professor and Dean, College of I…

Robbert van Renesse

关注

Professor of Computer Science, Cornell University
在 cs.cornell.edu 的电子邮件经过验证 - 首页

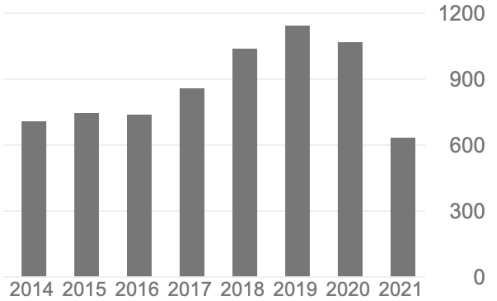
distributed systems fault tolerance

创建我的个人资料

标题	引用次数	年份
Bitcoin-ng: A scalable blockchain protocol I Eyal, AE Gencer, EG Sirer, R Van Renesse 13th {USENIX} symposium on networked systems design and implementation …	1242	2016
Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining R Van Renesse, KP Birman, W Vogels ACM transactions on computer systems (TOCS) 21 (2), 164-206	1205	2003
Horus: A flexible group communication system R Van Renesse, KP Birman, S Maffeis Communications of the ACM 39 (4), 76-83	1140	1996
Reliable distributed computing with the Isis toolkit KP Birman, RV Renesse IEEE Computer Society Press	987	1993
A gossip-style failure detection service R Van Renesse, Y Minsky, M Hayden Middleware'98, 55-70	804	1998
Amoeba: A distributed operating system for the 1990s SJ Mullender, G Van Rossum, AS Tananbaum, R Van Renesse, ... Computer 23 (5), 44-53	734	1990
Distributed operating systems AS Tanenbaum, R Van Renesse ACM Computing Surveys (CSUR) 17 (4), 419-470	627	1985
Experiences with the Amoeba distributed operating system AS Tanenbaum, R Van Renesse, H Van Staveren, GJ Sharp, ... Communications of the ACM 33 (12), 46-63	590	1990
COCA: A secure distributed online certification authority L Zhou, FB Schneider, R Van Renesse ACM Transactions on Computer Systems (TOCS) 20 (4), 329-368	552	2002

引用次数 查看全部

	总计	2016 年至今
引用	21254	5496
h 指数	66	34
i10 指数	179	87



可公开访问的出版物数量 查看全部

1 篇文章 28 篇文章

无法查看的文章 可查看的文章

根据资金授权书

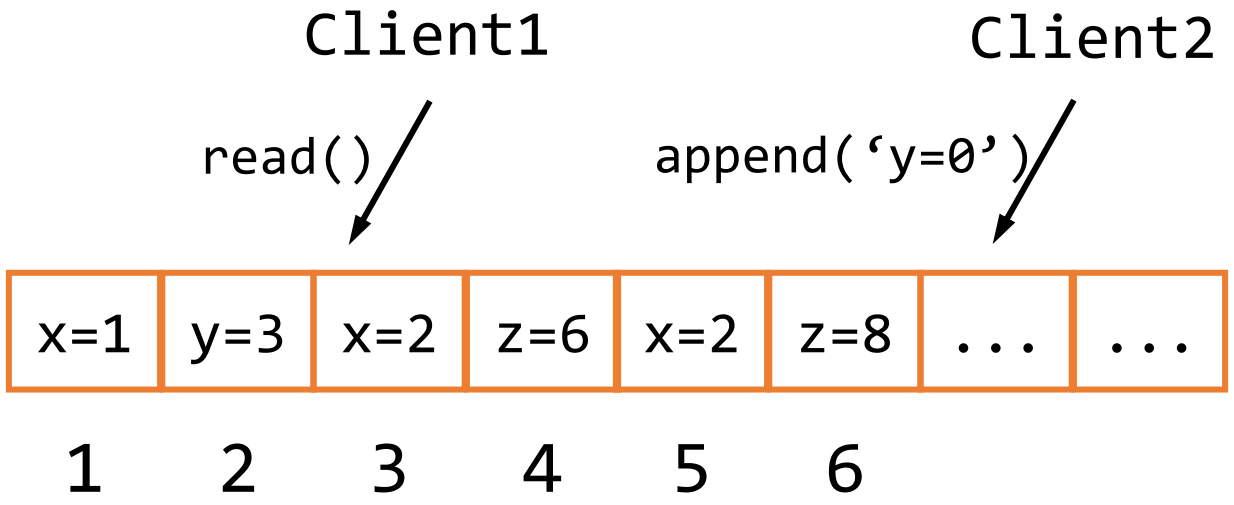
合著作者 查看全部


Kenneth P. Birman
Cornell University


Andrew Tanenbaum
Professor of Computer Science, …

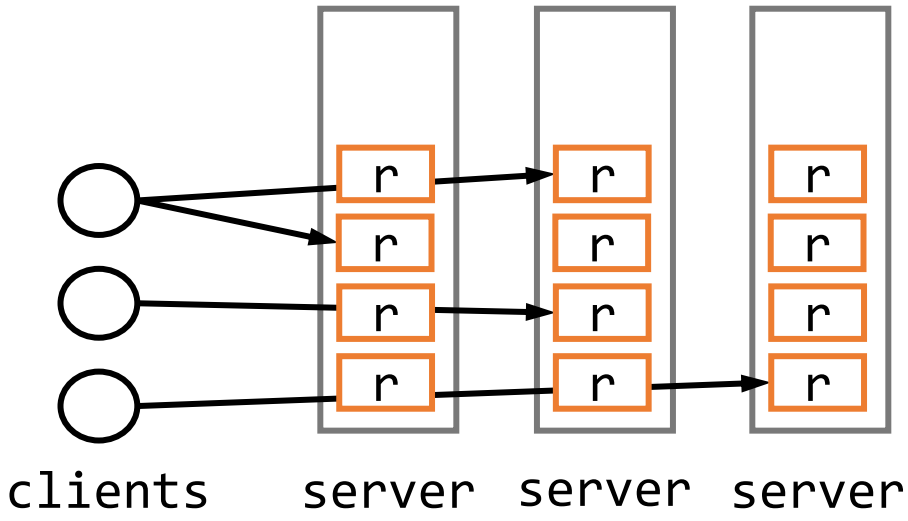
Background – Shared Log

- A set of ordered records
- Concurrently appended to and read by multiple clients



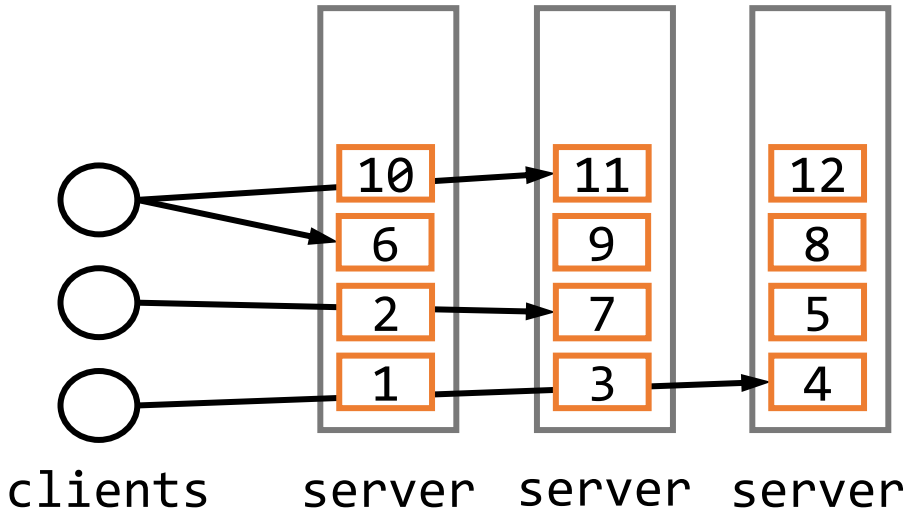
Background – Shared Log

- A set of ordered records
- Concurrently appended to and read by multiple clients
- Additional features:
  - Scalability



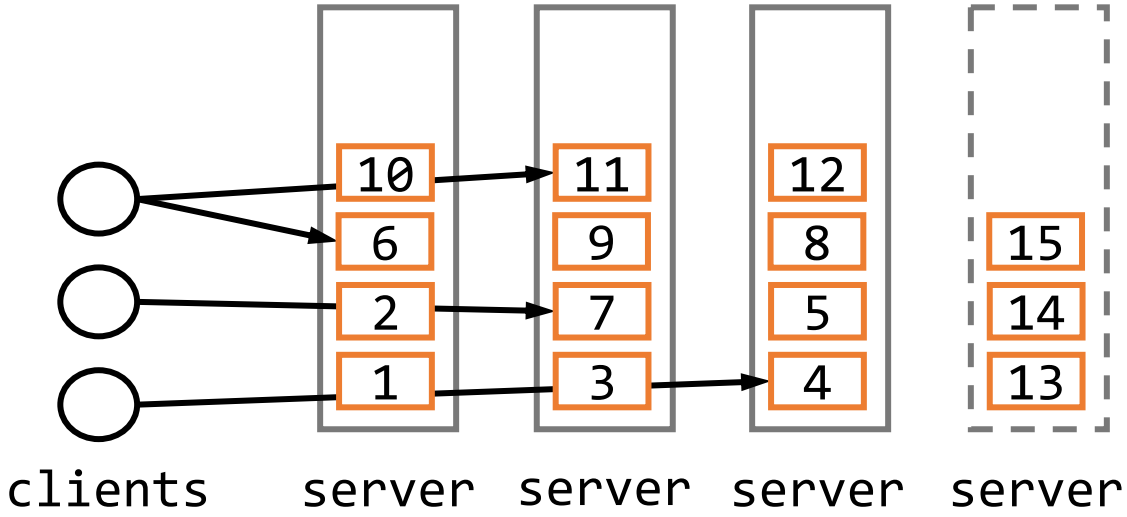
Background – Shared Log

- A set of ordered records
- Concurrently appended to and read by multiple clients
- Additional features:
  - Scalability
  - Total order



Background – Shared Log

- A set of ordered records
- Concurrently appended to and read by multiple clients
- Additional features:
  - Scalability
  - Total order
  - Seamless reconfiguration(add/remove instances)



Background – Existing Implementation

- 右边还有corfu、 fuzzylog、 kafka

Corfu[nsdi'12]

FuzzyLog[osdi'18]

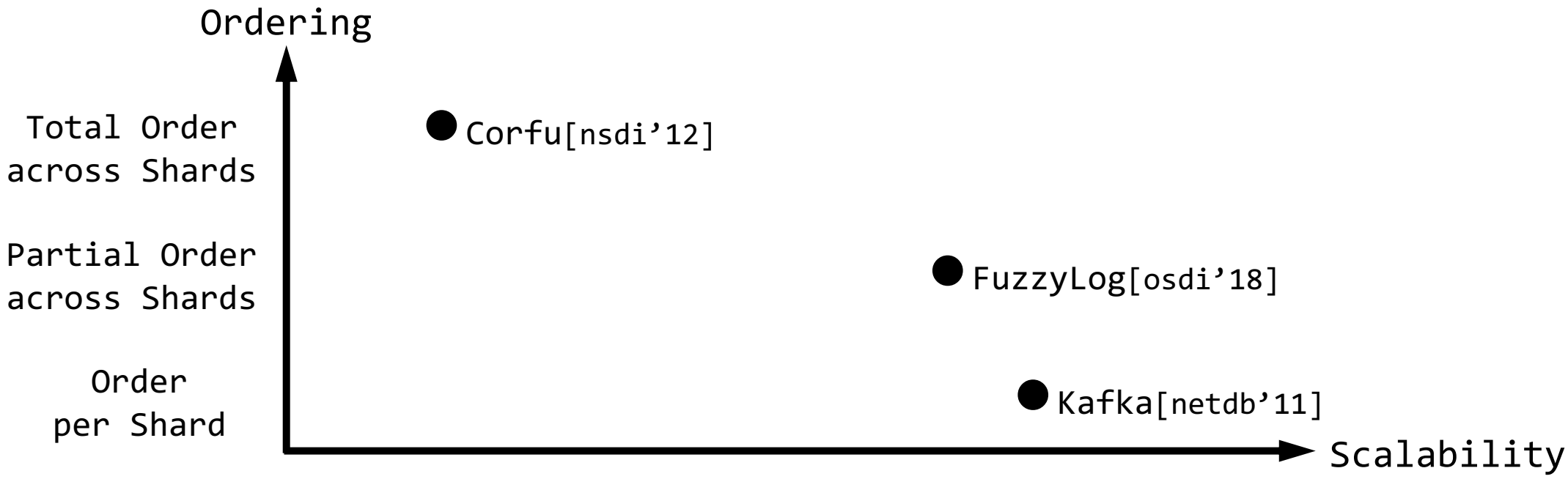
Kafka[netdb'11]

Background – Why Shared Log

- Analytics
  - Record and analyze web accesses for recommendations, ad placement, intrusion detection, performance debugging, etc
- Failure recovery
  - Committed steps to replay
- Consensus engine with an order

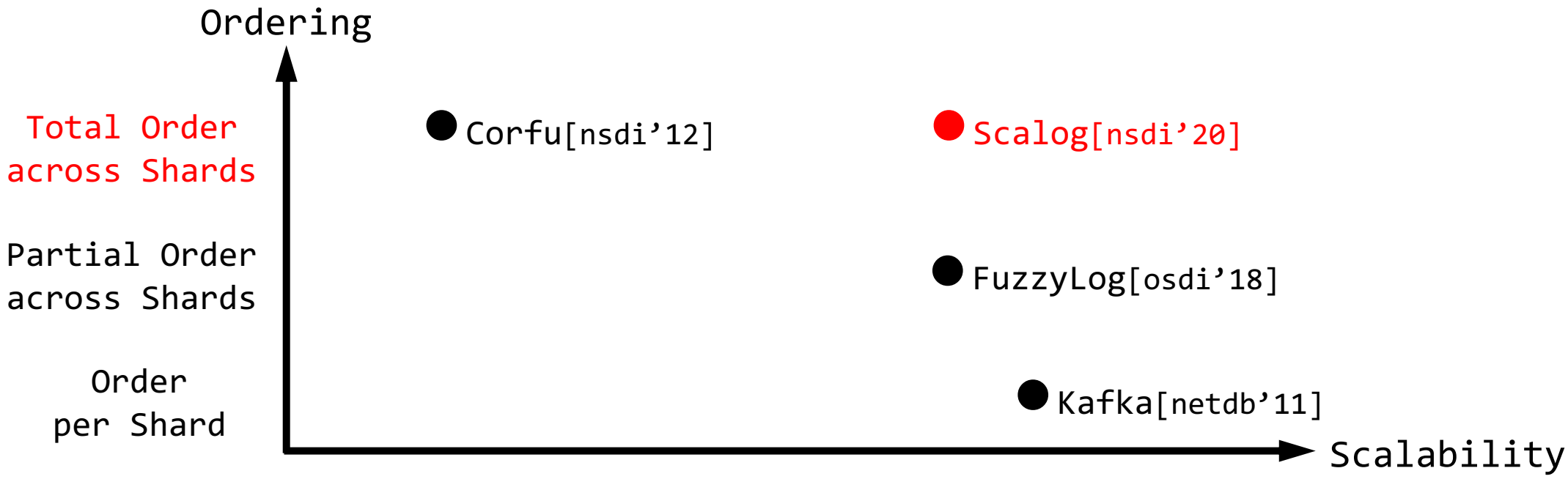
Motivation – Scalability vs Ordering

- Trade-off



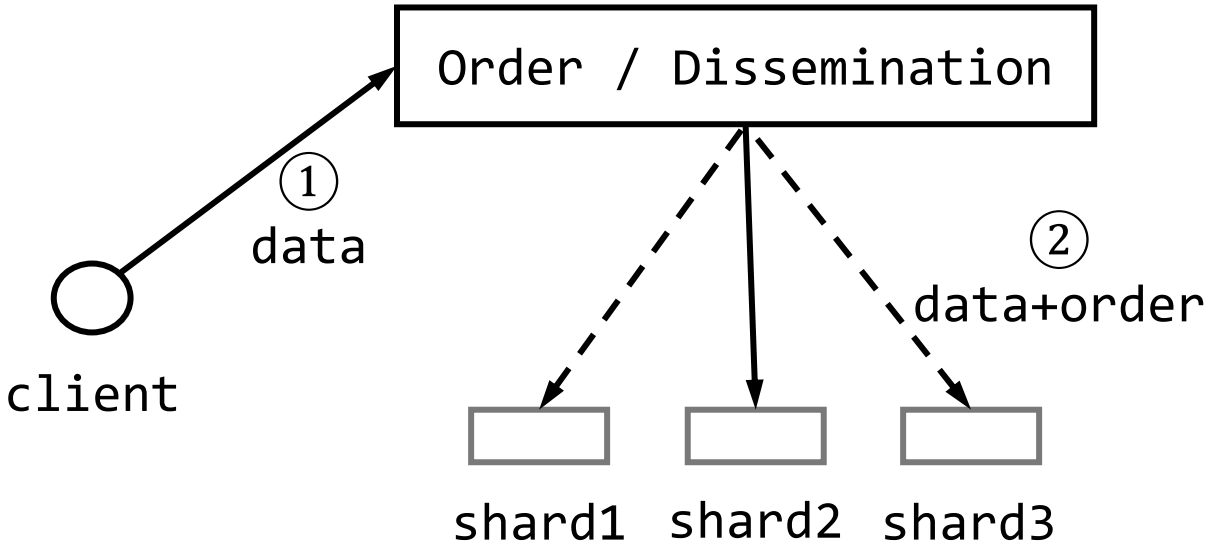
Motivation – Scalability vs Ordering

- Trade-off



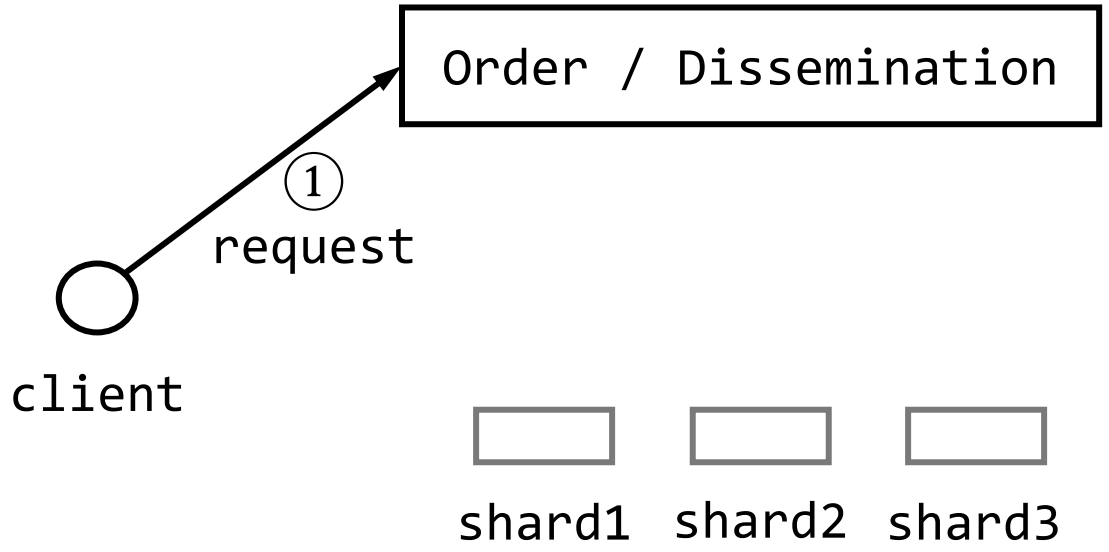
Motivation – Scalability vs Seamless Reconfiguration

- Shared log: LogDevice from Facebook
  - High throughput: ❌
    - bottleneck
  - Seamless reconfiguration: ✅
    - still available



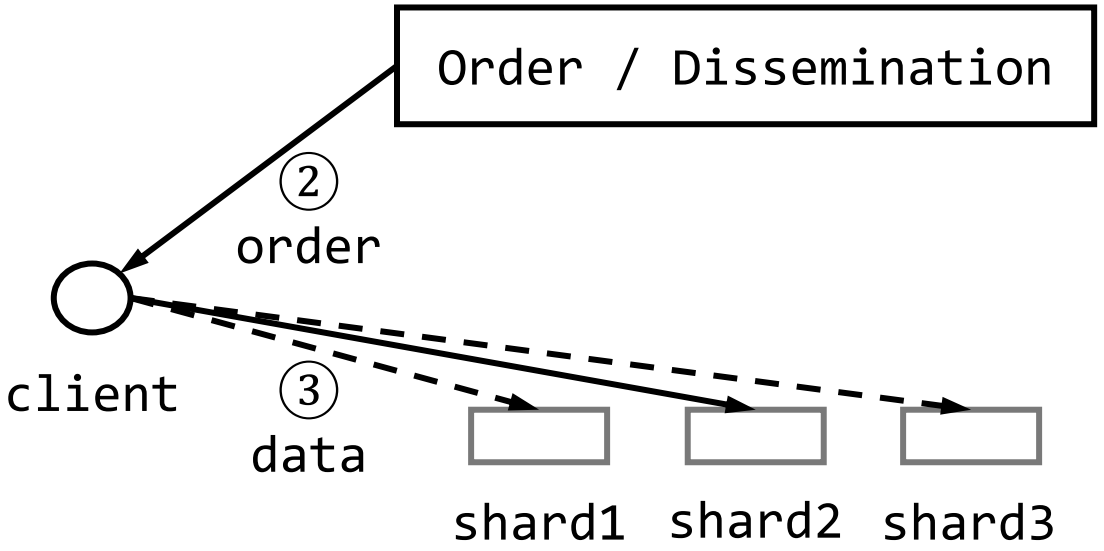
Motivation – Scalability vs Seamless Reconfiguration

- Shared log: Corfu[nsdi'12]



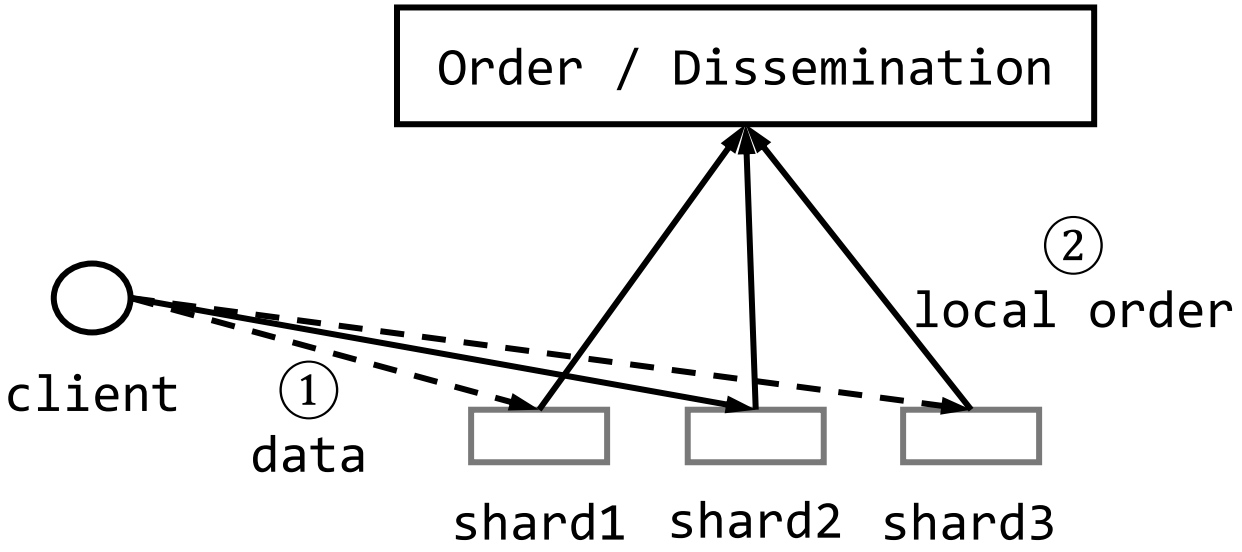
Motivation – Scalability vs Seamless Reconfiguration

- Shared log: Corfu[nsdi'12]
  - High throughput: ✅
    - data separated from ordering
  - Seamless reconfiguration: ❌
    - not available



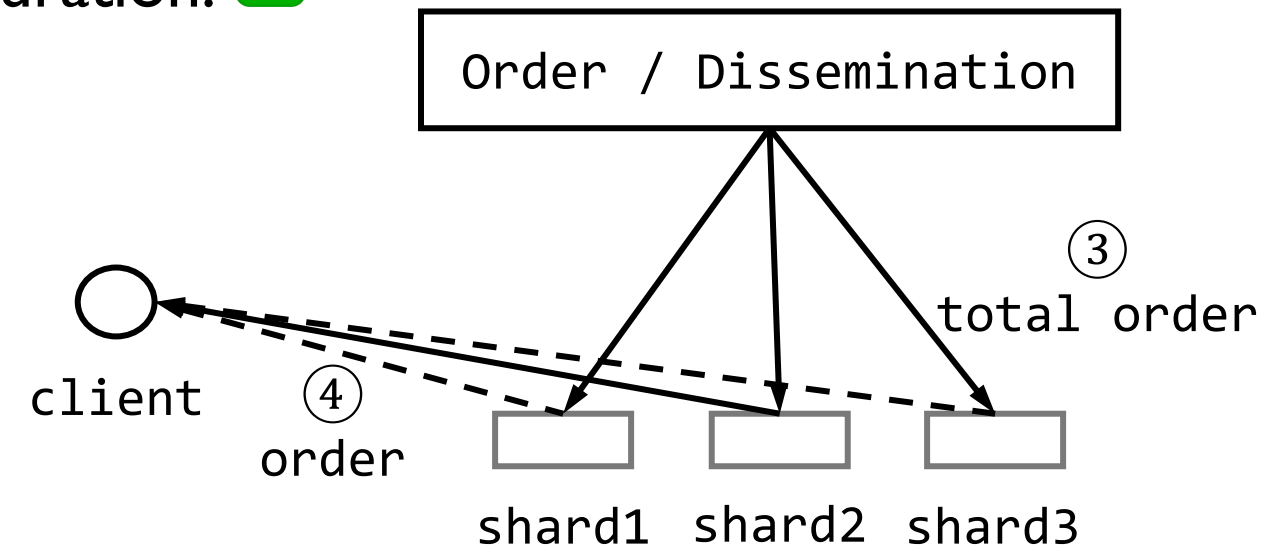
Motivation – Scalability vs Seamless Reconfiguration

- Shared log: Scalog[nsdi'20]

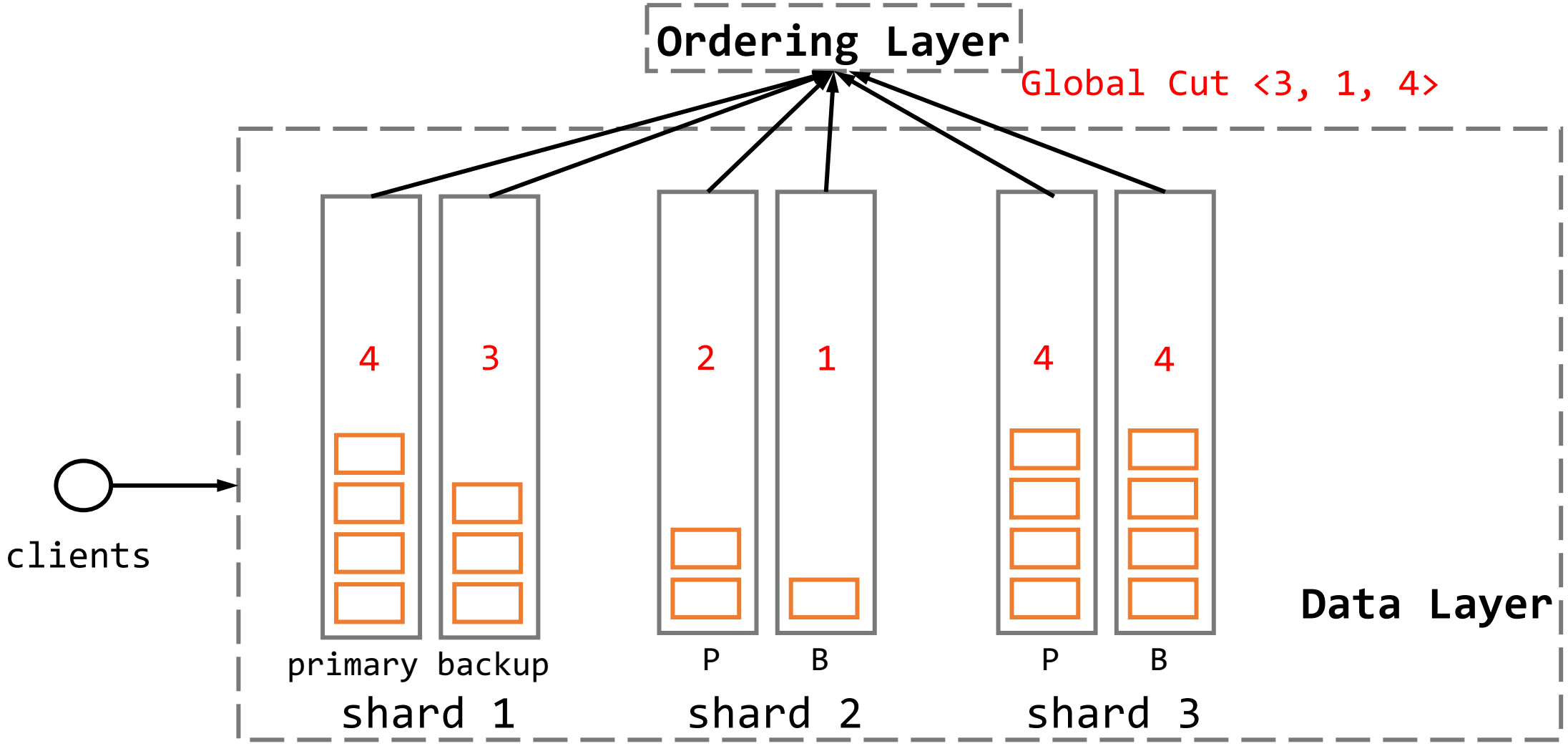


Motivation – Scalability vs Seamless Reconfiguration

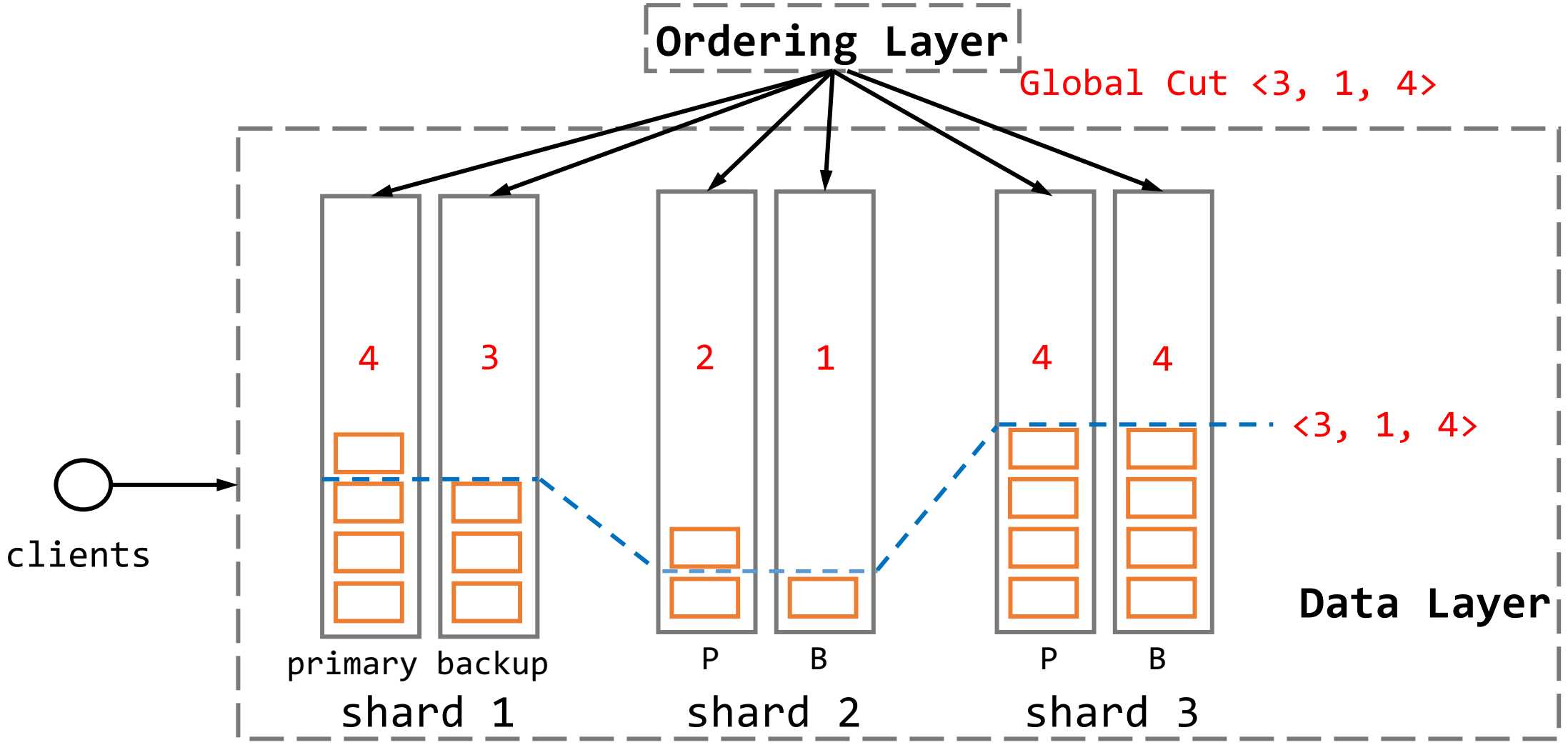
- Shared log: Scalog[nsdi'20]
  - High throughput: ✅ ✅
    - data separated from ordering
    - nearest server
  - Seamless reconfiguration: ✅
    - available!



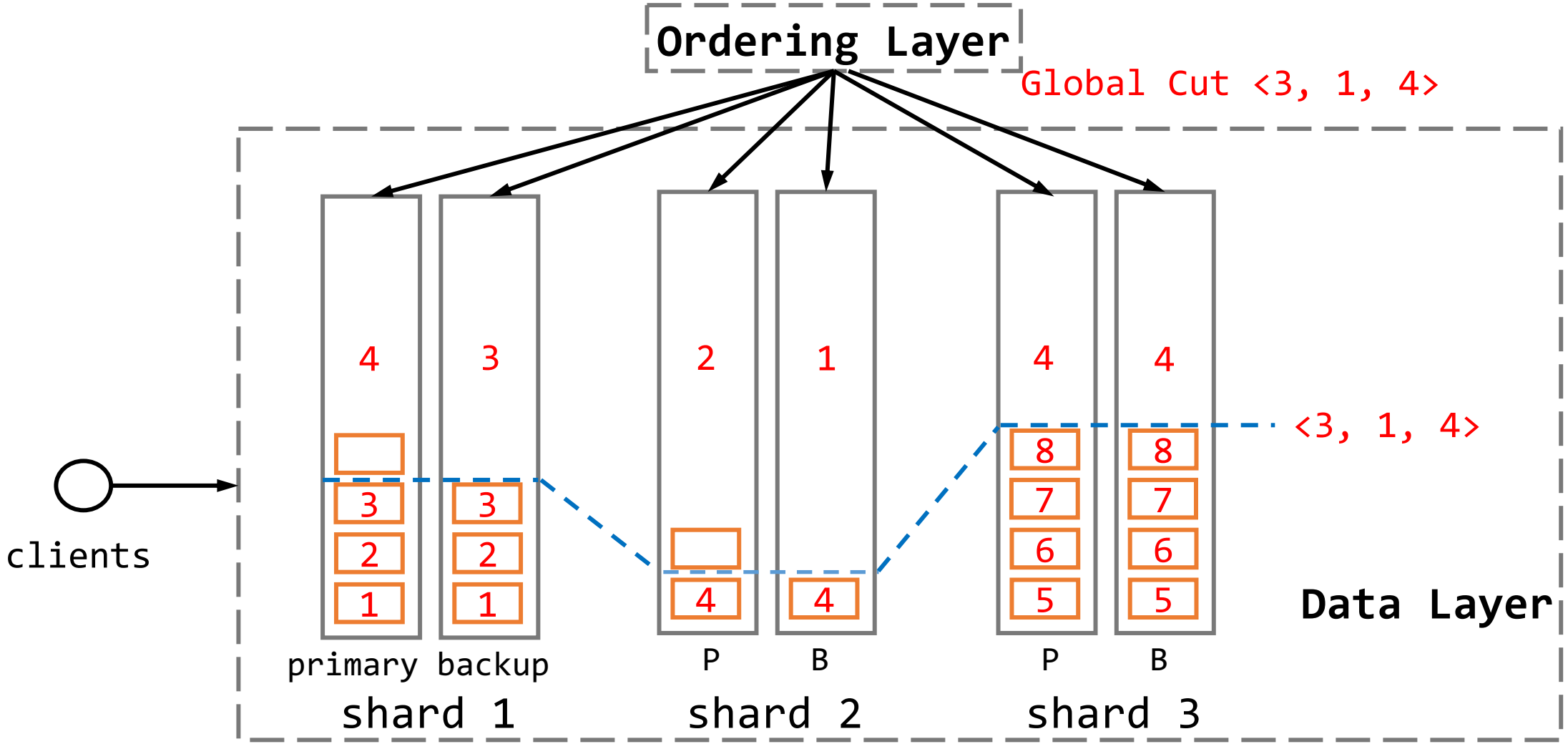
Scalog – Workflow: Append



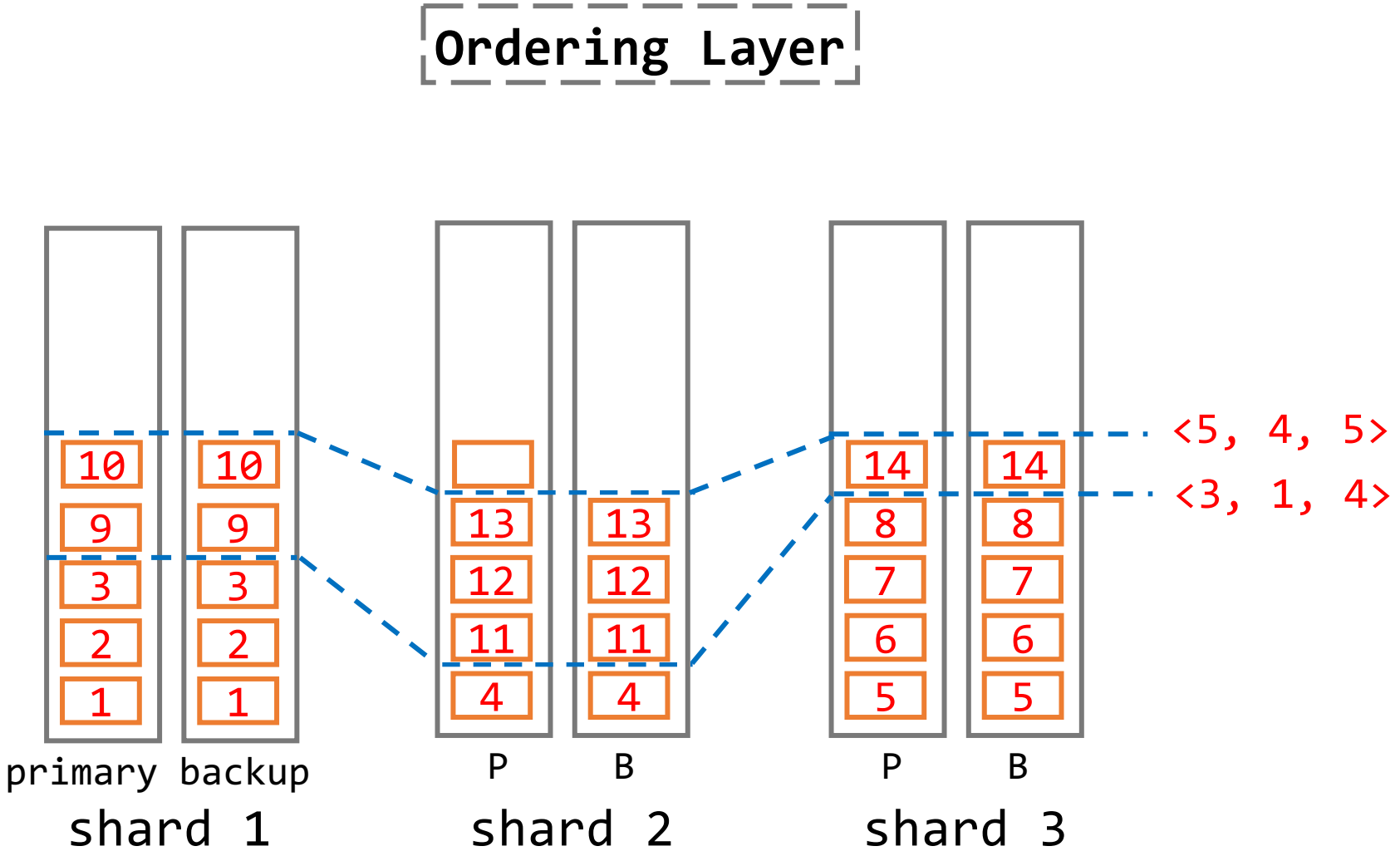
Scalog – Workflow: Append



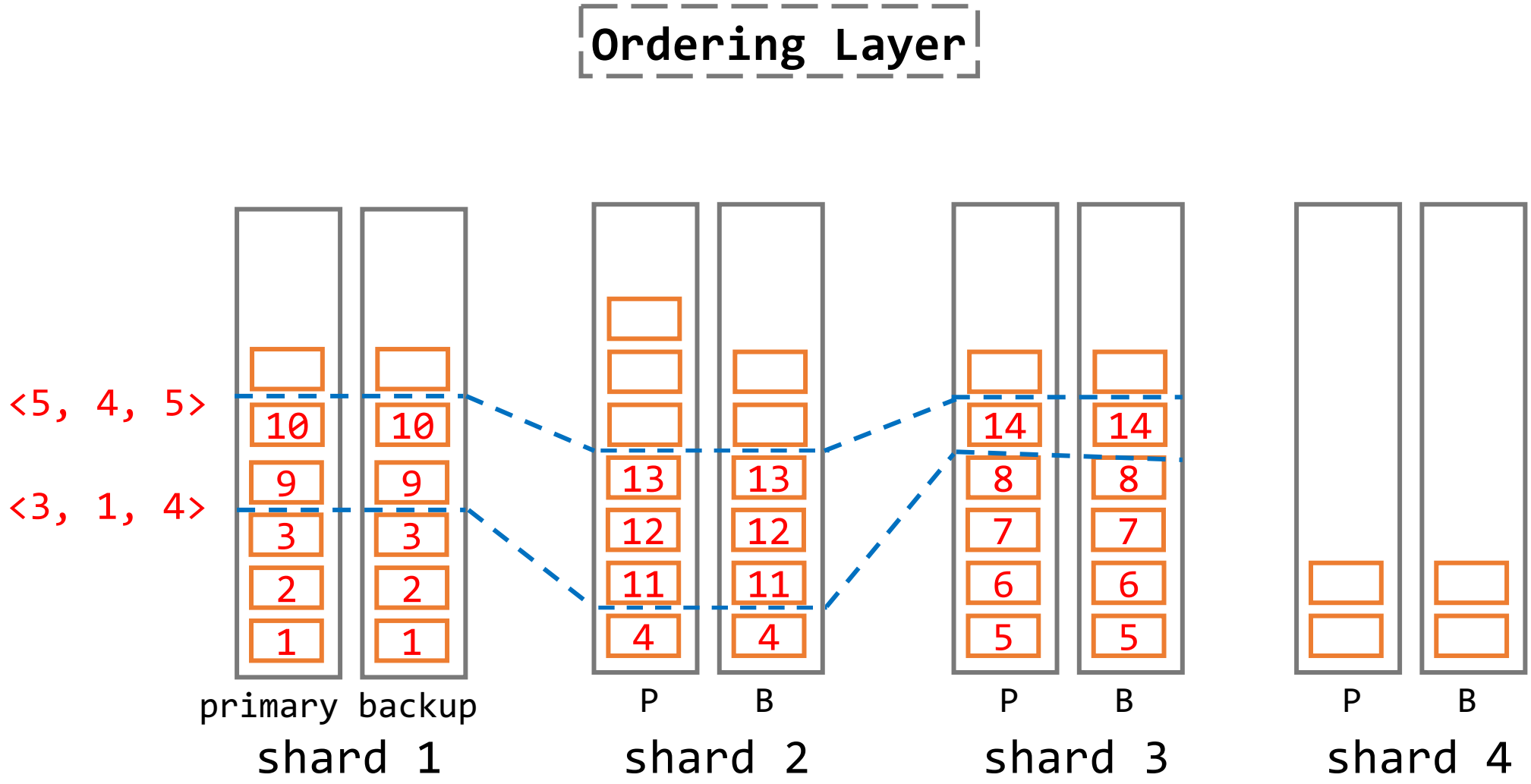
Scalog – Workflow: Append



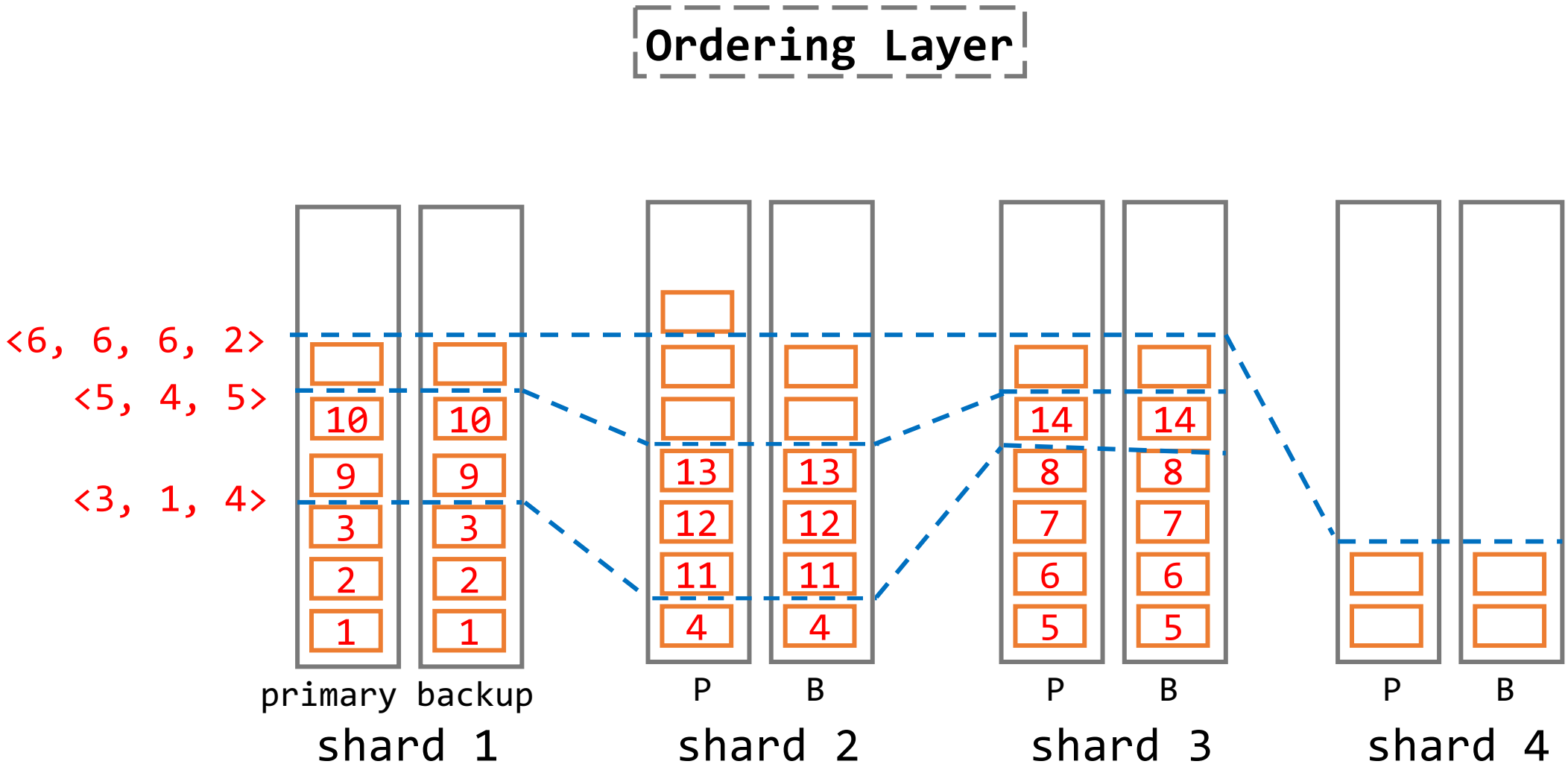
Scalog – Workflow: Append



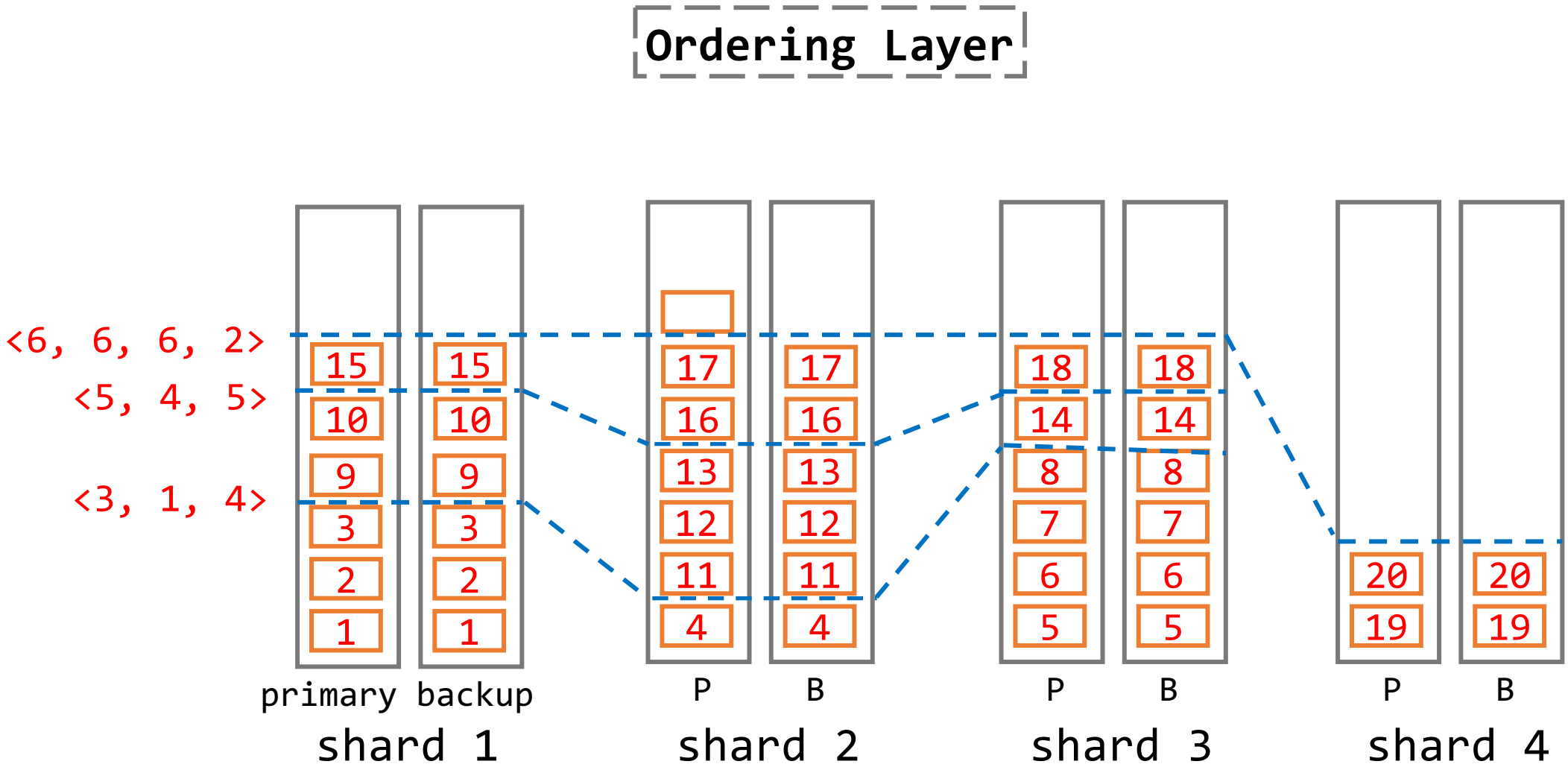
Scalog – Workflow: Adding a Shard



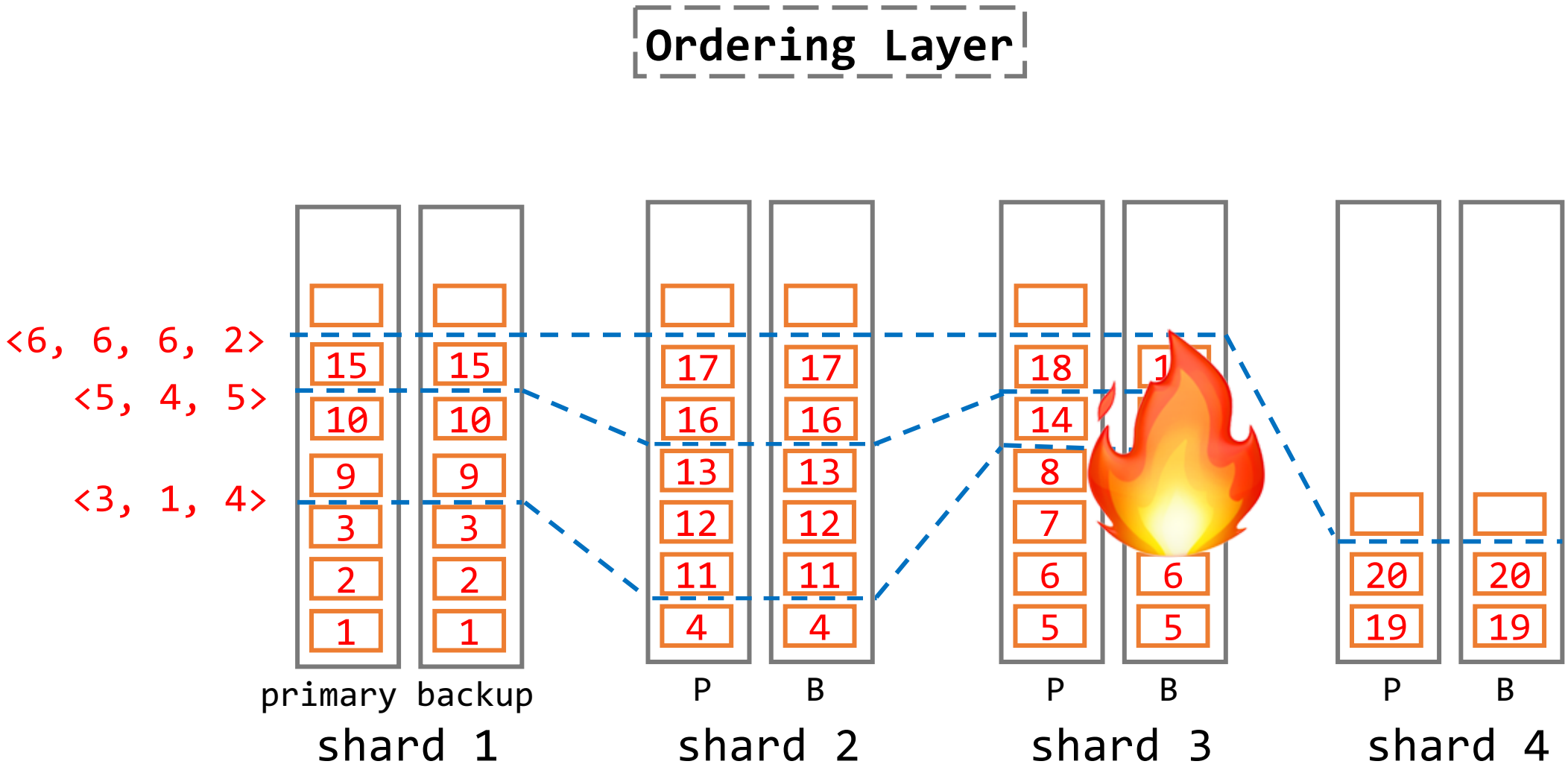
Scalog – Workflow: Adding a Shard



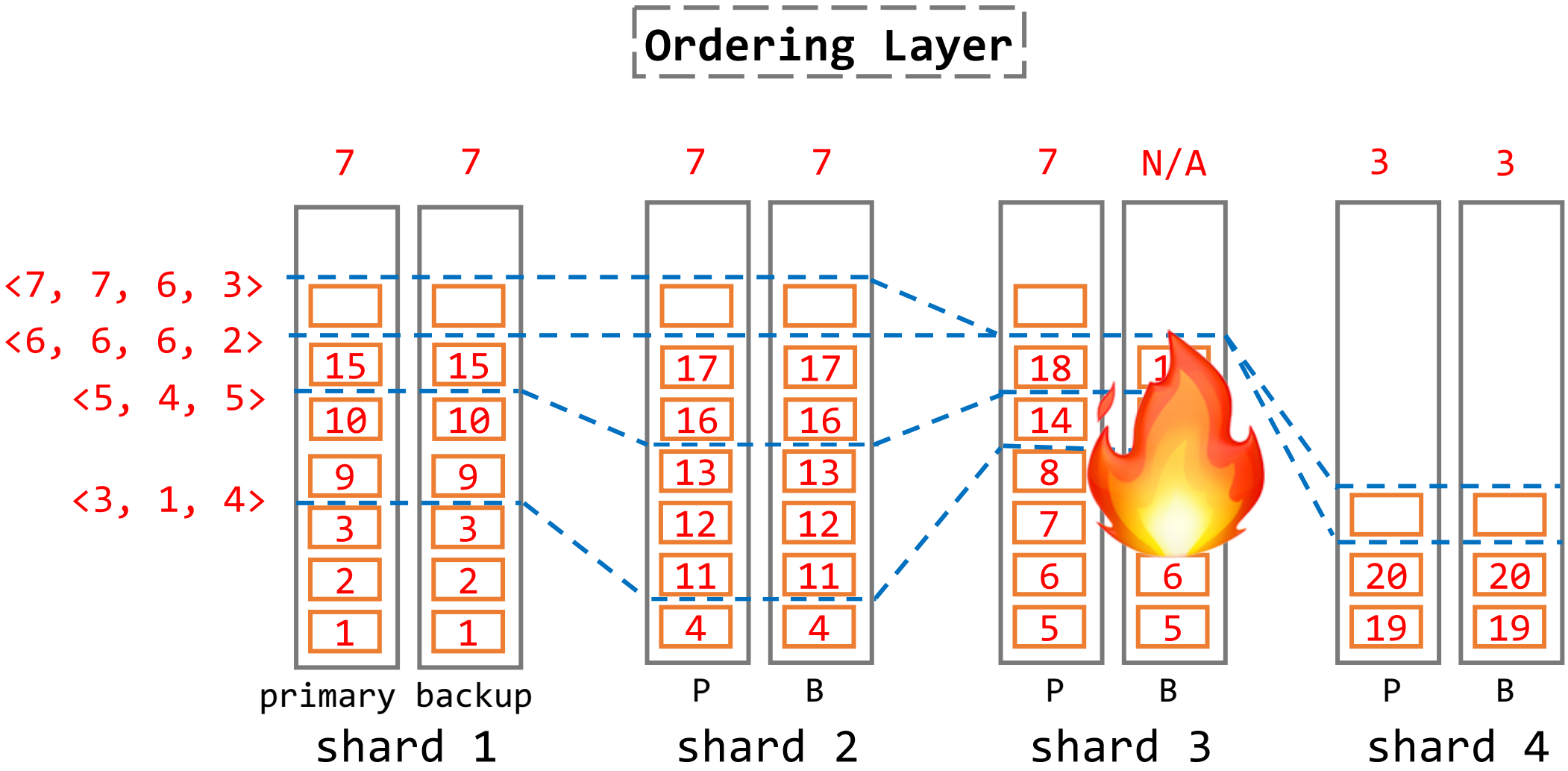
Scalog – Workflow: Adding a Shard



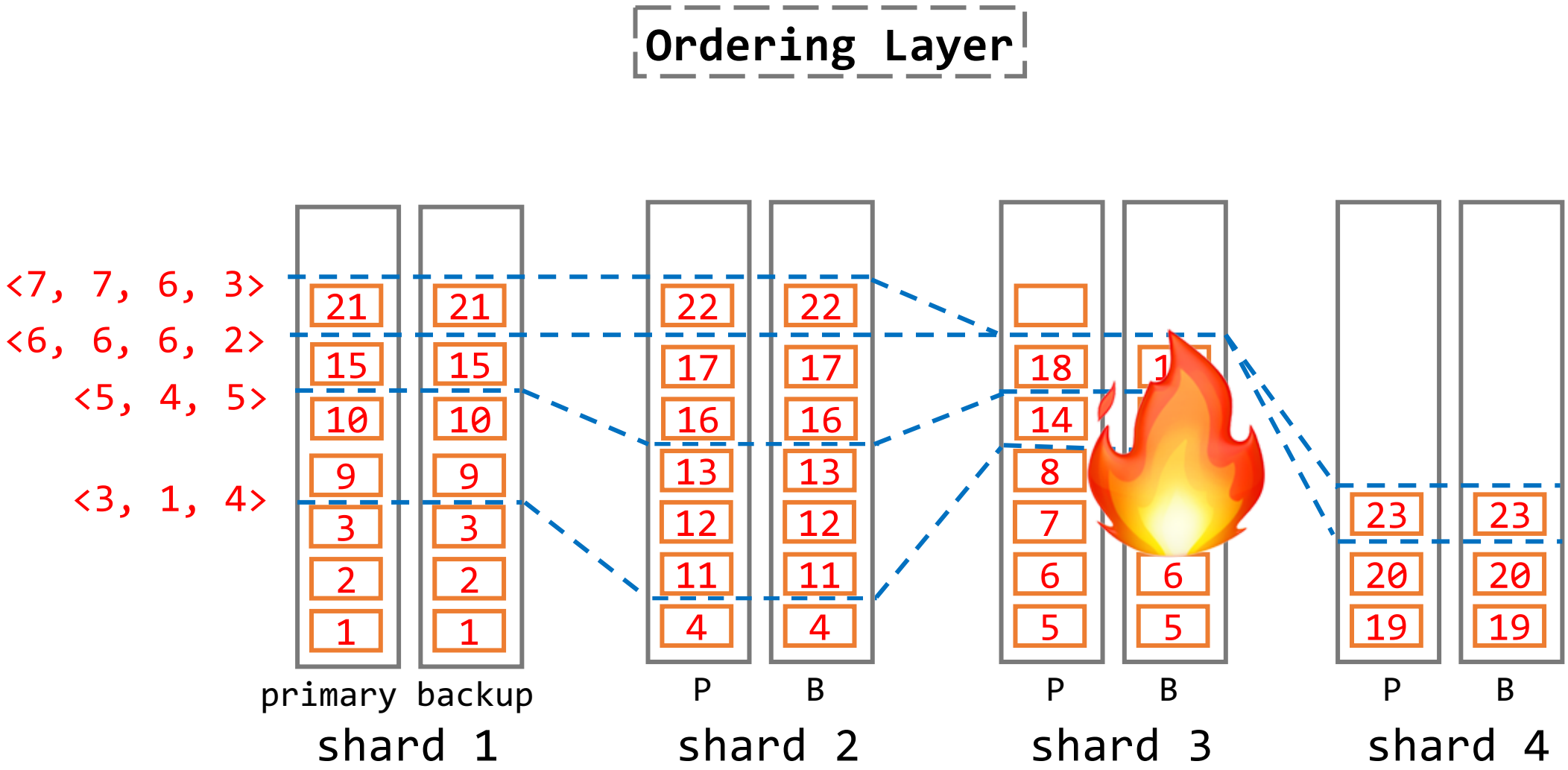
Scalog – Workflow: Handling Failures



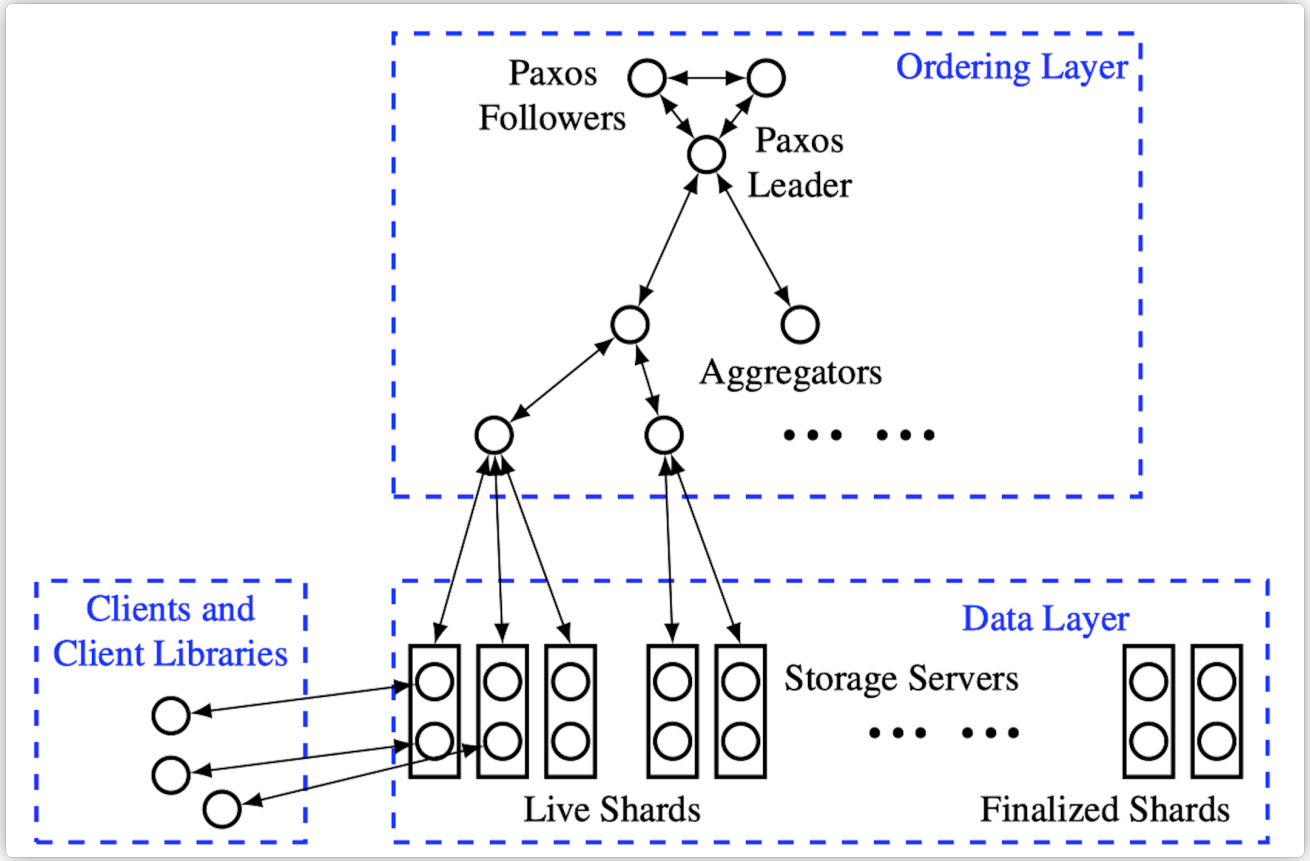
Scalog – Workflow: Handling Failures



Scalog – Workflow: Handling Failures



Scalog - Architecture



Scalog - API

`append`(r)	Append record r, and return the global sequence number.
`trim`(l)	Delete records before global sequence number l.
`subscribe`(l)	Subscribe to records starting from global sequence number l.
`setShardPolicy`(p)	Set the policy for which records get placed at which storage servers in which shards.
`appendToShard`(r)	Append record r, and return the global sequence number and shard identifier.
`readRecord`(l, s)	Request the record with sequence number l from shard s.

Table 1: Scalog API

Evaluation – Setup

- Scalog:
  - implemented in Golang
- Baseline: Corfu
  - re-implemented and optimized in Golang
  - achieves **higher** throughput and latency than open-source implementation. **lower**
  - write throughput: 530k/s ≤ 570k/s
    - reported in Tango([nsdi'13]) with FPGA+SSD

Evaluation – Setup

- Hardware:
  - 40 c220g1 servers in **CloudLab** and each has
    - 2 Intel E52630 v3 8-core CPUs at 2.40GHz,
    - 128GB ECC memory
    - a 480GB SSD
    - a 10Gbps intra-datacenter network connection.
- For higher throughput, evaluate by simulation
- Configuration:
  - Record size: 4KB
  - Aggregating interval: 0.1ms

Evaluation – Reconfiguration

- 6 shards (2 servers per shard)
- clients are notified before finalizing
- clients write with throughput as 50% peak of Corfu

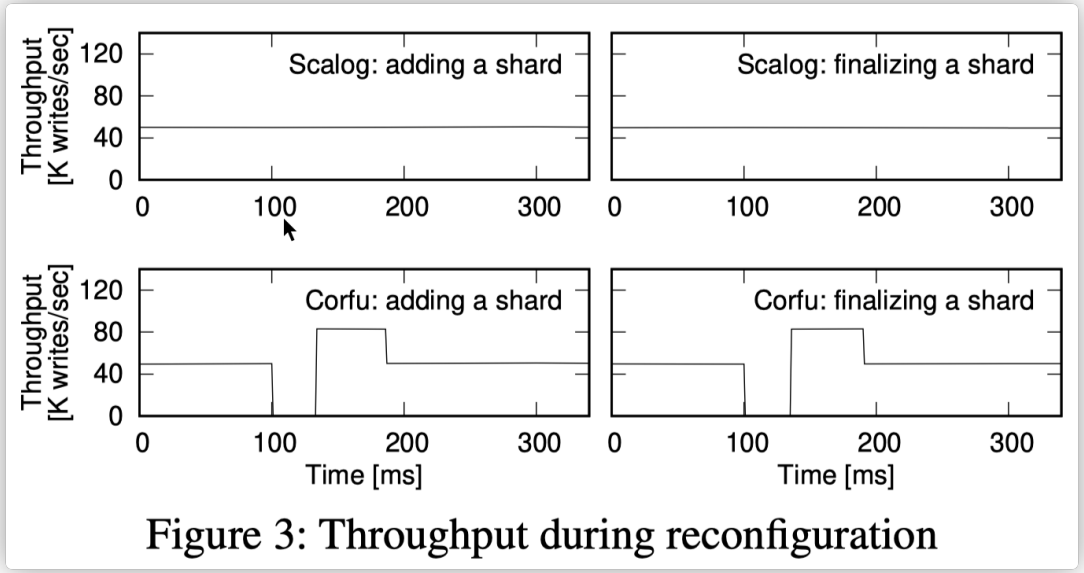


Figure 3: Throughput during reconfiguration

Evaluation – Reconfiguration

- 6 shards (2 servers per shard)
- clients are notified before finalizing
- clients write with throughput as 50% peak of Corfu

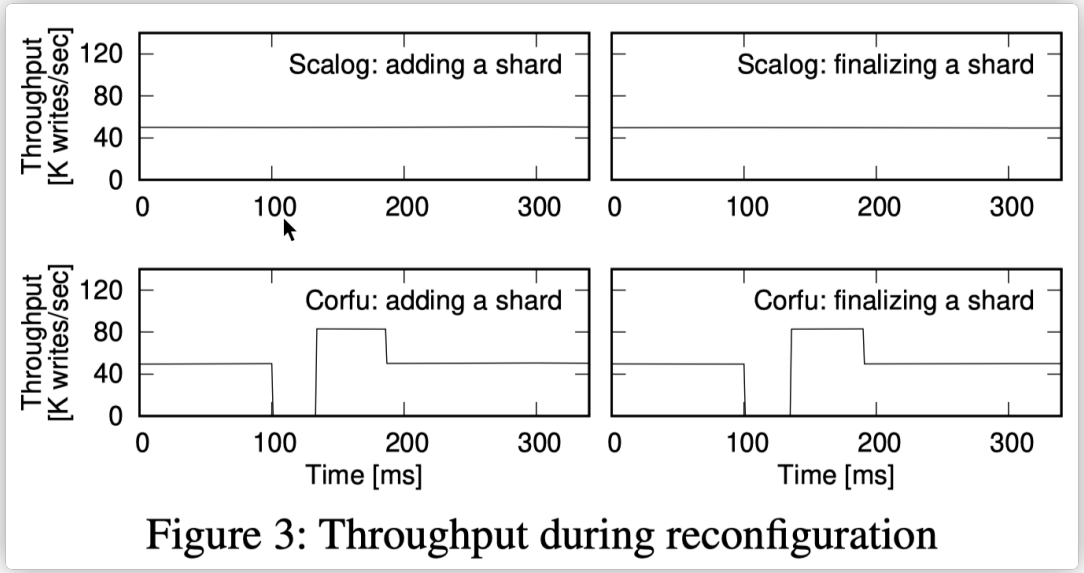
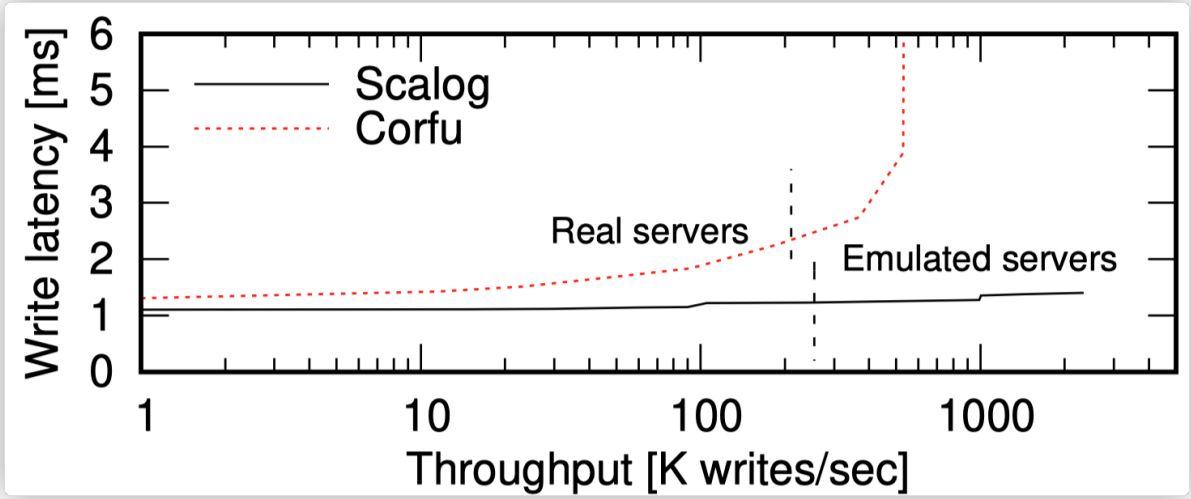


Figure 3: Throughput during reconfiguration

- Corfu is **unavailable** for ~30ms while Scalog isn't

Evaluation – Write Performance

- Less than 40 servers
  - Scalog: 2.34M writes/s vs Corfu: 530k writes/s
- Simulation for more than 40 servers
  - Paxos leader and aggregators are not simulated
  - Scalog's ordering layer can handle 3,500 shards with
    - 52M writes/p at 1.6ms latency



Evaluation – Read Performance

- Scalog and Corfu
  - Both reads from one server directly
  - Similar throughput and latency

Additional

- 3 applications are built on Scalog
  - Scalog-Store: vs Corfu-Store[from corfu paper]
    - key-value store based
    - mapping each key to a pair(***order_id***, ***shard_id***)
      - which specifies the latest value of that key
  - vScalog: vs vCorfu[nsdi'17]
    - object store based
    - higher robustness and read throughput
  - Online Marketplace
    - user activity analytics application

Additional

- Failure handling

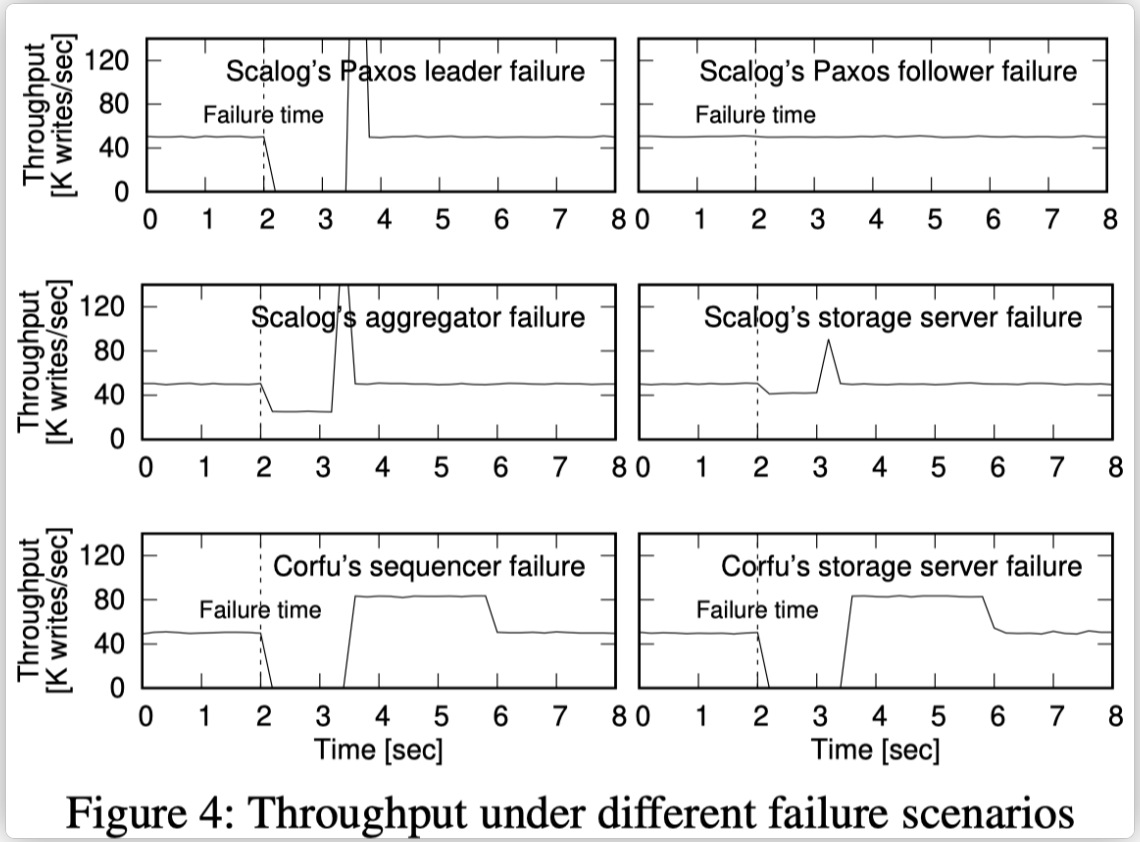
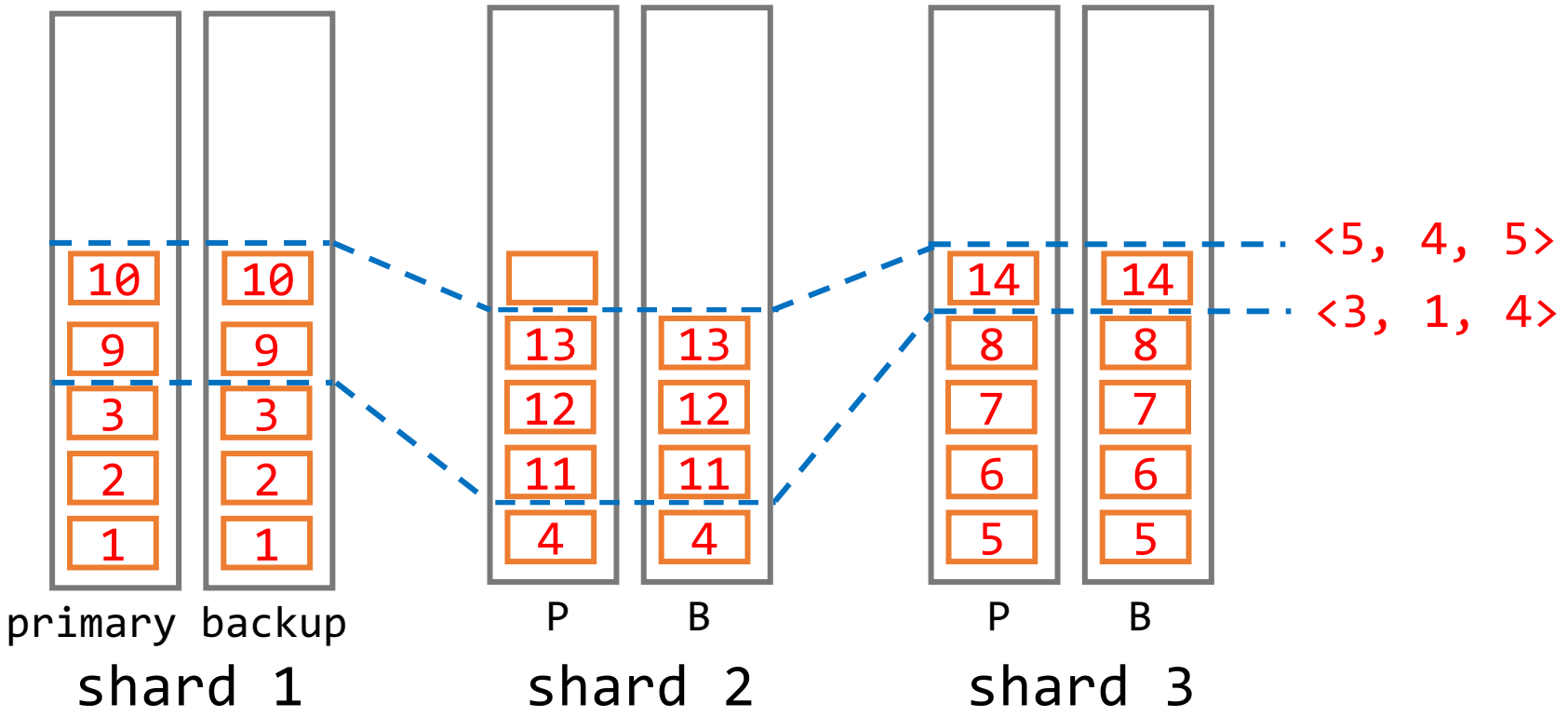


Figure 4: Throughput under different failure scenarios

Discussion

- Batching?
  - in Tango[nsdi'13]:
    - Corfu with 4 record batching attain 2M records/s



Scalog: Seamless Reconfiguration and Total Order in a Scalable Shared Logs

NSDI'20

Cong Ding, David Chu, and Evan Zhao, Cornell University; Xiang Li, Alibaba Group;
Lorenzo Alvisi and **Robbert van Renesse**, Cornell University

Shared by Zevin at USTC-SYS Reading Group

USTC, CHINA
ADSLAB