ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

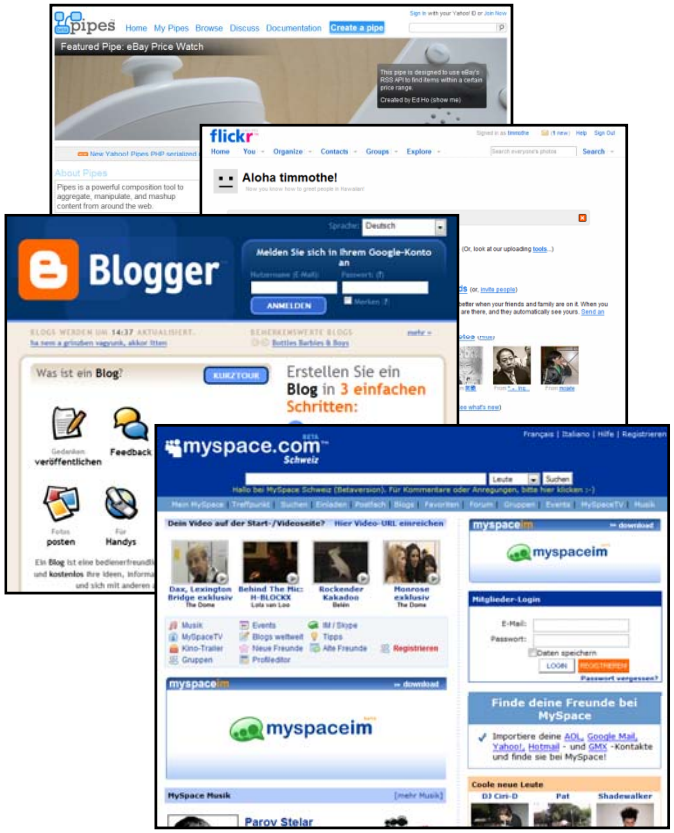
Building a Database on S3

Matthias Brantner ✳, Daniela Florescu+, David Graf ✳,
Donald Kossmann ♦✳, Tim Kraska♦

Systems Group, ETH Zurich♦
28msec Inc. ✳
Oracle +

Motivation

- Building a web page, starting a blog, and making both searchable for the public have become a commodity
- But providing your own service (and to get rich) still comes at high cost:
  - Have the right (business) idea
  - Run your own web-server and database
  - Maintain the infrastructure
  - Keep the service up 24 x 7
  - Backup the data
  - Tune the system if the service is used more often



And then comes the Digg-Effect

Requirements for DM on the Web

- Scalability
  - response time independent of number of clients
- No administration
  - „outsource“ patches, backups, fault tolerance
- 100 percent read + write availability
  - no client is ever blocked under any circumstances
- Cost ($$$)
  - get cheaper every year, leverage new technology
  - pay as you go along, no investment upfront

Utility Computing as a solution

- Scalability
  - response time independent of number of clients
- No administration
  - „outsource“ patches, backups, fault tolerance
- 100 percent read + write availability
  - no client is ever blocked under any circumstances
- Cost ($$$)
  - get cheaper every year, leverage new technology
  - pay as you go along, no investment upfront

Consistency: optimization goal, not constraint

Utility Computing as a solution

- Scalability
  - response time independent of number of clients
- No administratio
- back
- get cheaper eve ear, lev e new technology
- pay as you go a ng, no inves ent upfront

Consistency
How much consistency is required by my application?

Cost
How much does it cost?

Consistency: optimization goal, not constraint

Amazon Web Services (AWS)

- Most popular utility provider
  - Gives us all necessary building blocks (Storage, CPU-cycles, etc.)
  - Other providers also appear on the market
- Amazon infrastructure services:

Simple Storage Service (S3)

- (Virtually) infinite store
- Costs: $0.15 per GB-month + transfer costs ($0.1-$0.17 In/Out per GB)

Simple Queuing Service (SQS)

- Message service
- Allows to exclusively receive a message
- Costs: $0.0001 per message sent + transfer costs

Elastic Cloud Computing (EC2)

- Virtual instance: 1-8 virtual cores (=1.0-2.5 GHz Opterons), 1.7-15 GB of memory, 160GB-1690GB of instance storage
- Costs: $0.1-$0.8 per hour + transfer costs

SimpleDB

- Basically a text-index
- Costs: $0.14 per Amazon SimpleDB machine hour consumed

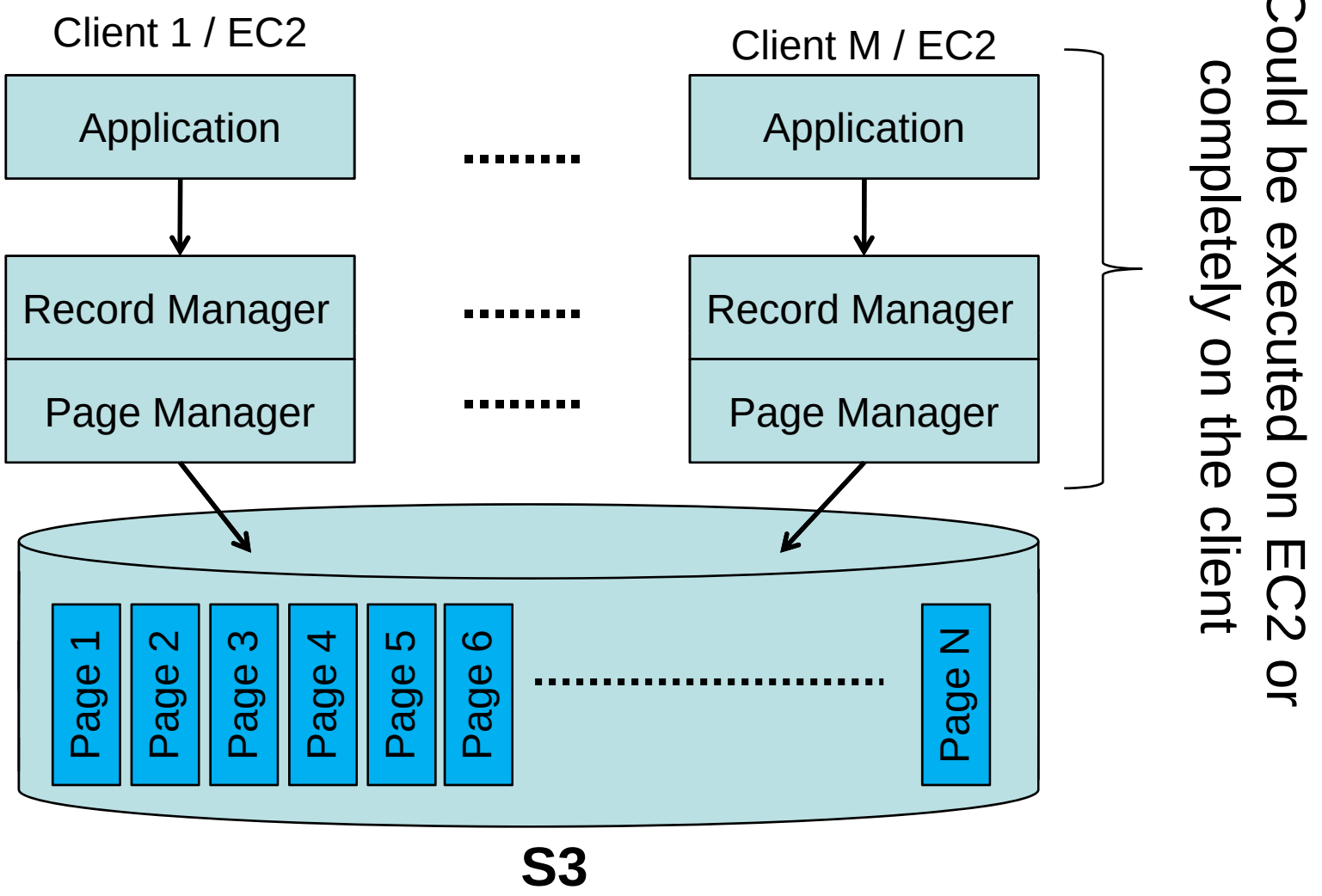
Plan of Attack

- Step 1: Use S3 as a huge shared disk
  - leverage scalability, no admin features
- Step 2: Allow concurrent access to shared disk in a distributed system
  - keep properties of a distributed system, maximize consistency
- Step 3: Do application-specific trade-offs
  - consistency vs. cost
  - consistency vs. availability
  - *consistency à la carte* (levels of consistency)

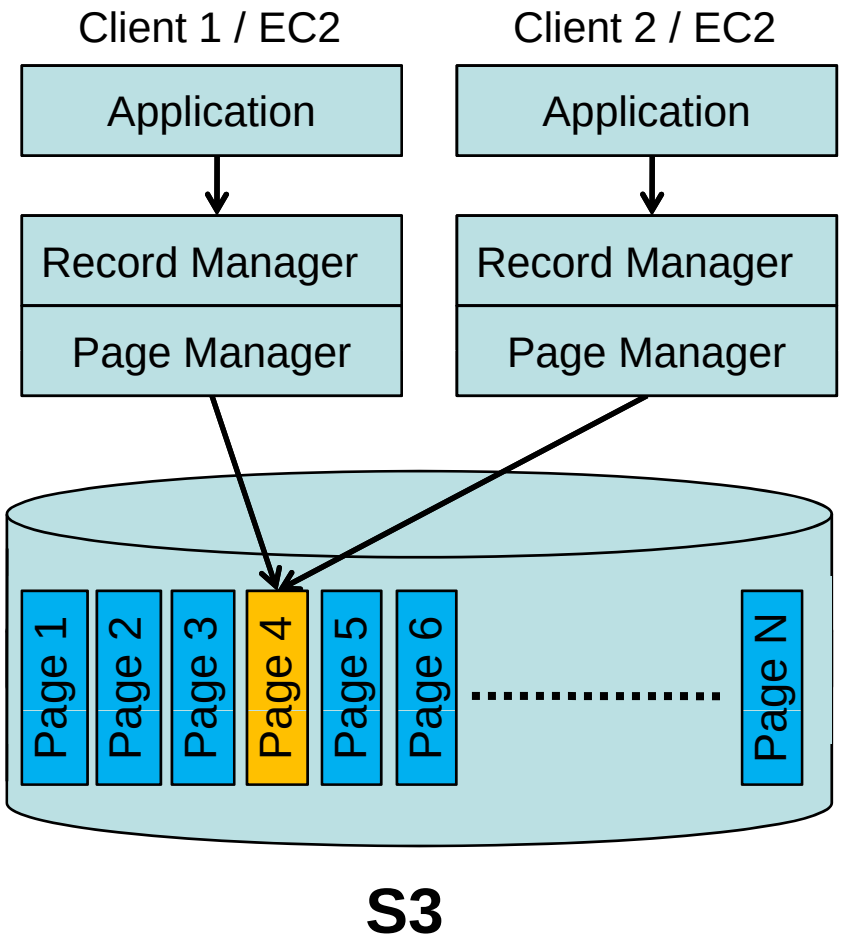
Plan of Attack

- Step 1: Use S3 as a huge shared disk
  - leverage scalability, no admin features
- Step 2: Allow concurrent access to shared disk in a distributed system
  - keep properties of a distributed system, maximize consistency
- Step 3: Do application-specific trade-offs
  - consistency vs. cost
  - consistency vs. availability
  - *consistency à la carte* (levels of consistency)

Shared-Disk Architecture



Problem: Eventual Consistency



- Two clients update the same page
- Last update wins
- Consistency problem
  - Inconsistency between indexes and page
  - Lost records
  - Lost updates

Plan of Attack

- Step 1: Use S3 as a huge shared disk
  - leverage scalability, no admin features
- Step 2: Allow concurrent access to shared disk in a distributed system
  - keep properties of a distributed system, maximize consistency
- Step 3: Do application-specific trade-offs
  - consistency vs. cost
  - consistency vs. availability
  - *consistency à la carte* (levels of consistency)

Levels of Consistency [Tanenbaum]

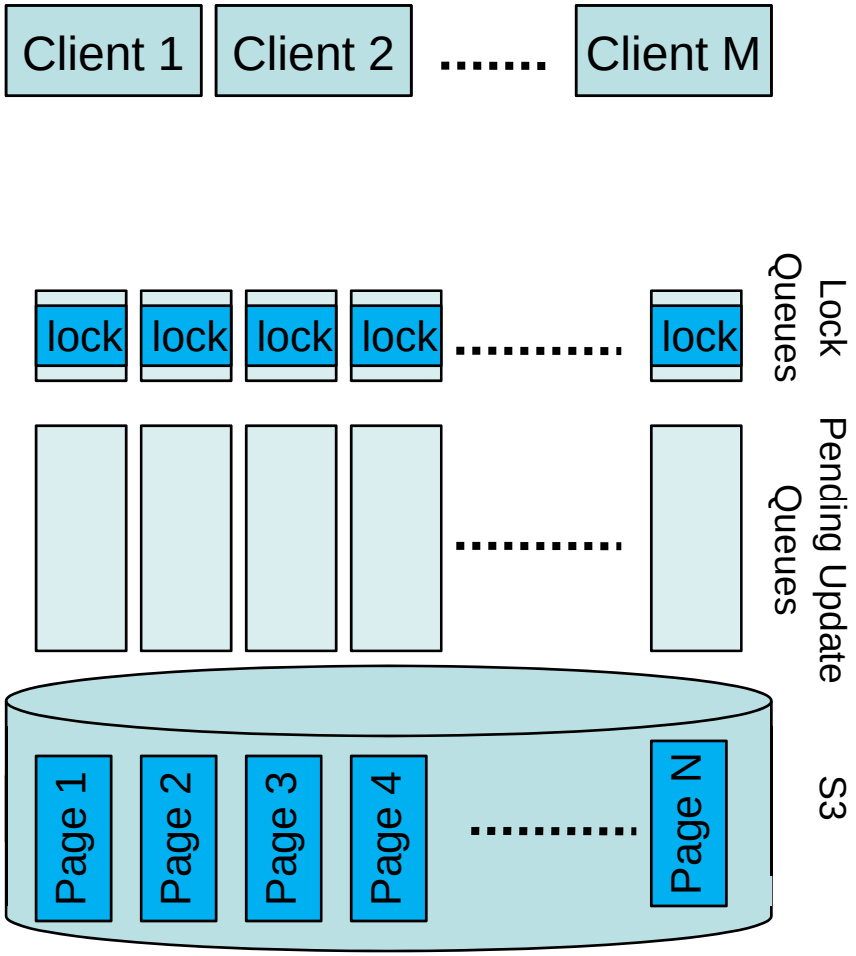
- Shared-Disk (Naïve approach)
  - No concurrency control at all
- Eventual Consistency (Basic Protocol)
  - Updates become visible any time and will persist
  - No lost update on page level
- Atomicity
  - All or no updates of a transaction become visible
- Monotonic reads, Read your writes, Monotonic writes, ...
- Strong Consistency
  - database-style consistency (ACID) via OCC

Levels of Consistency [Tanenbaum]

- Shared-Disk (Naïve approach)
  - No concurrency control at all
- Eventual Consistency (Basic Protocol)
  - Updates become visible any time and will persist
  - No lost update on page level
- Atomicity
  - All or no updates of a transaction become visible
- Monotonic reads, Read your writes, Monotonic writes, ...
- Strong Consistency
  - database-style consistency (ACID) via OCC

Basic Protocol: Queues

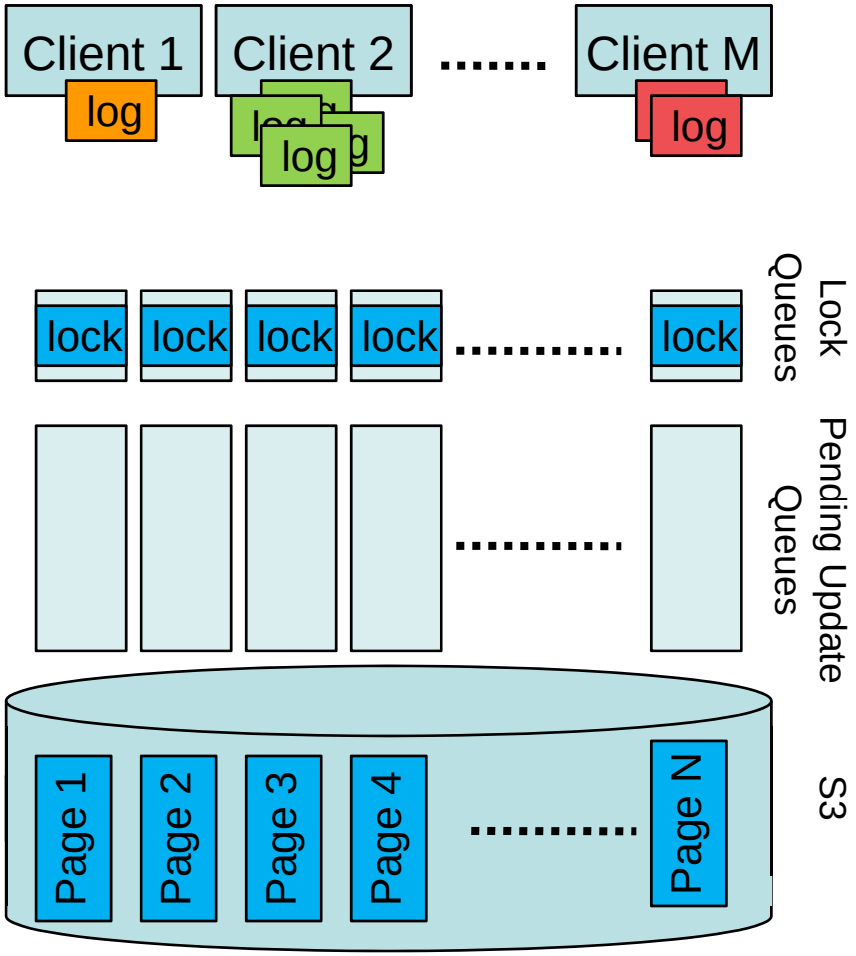
- One PU and Lock queue is associated to each page
- Lock queues contain exactly one message (inserted directly after creating the queue)
- Commit to pages in two phases



Basic Protocol

Step 1: Commit

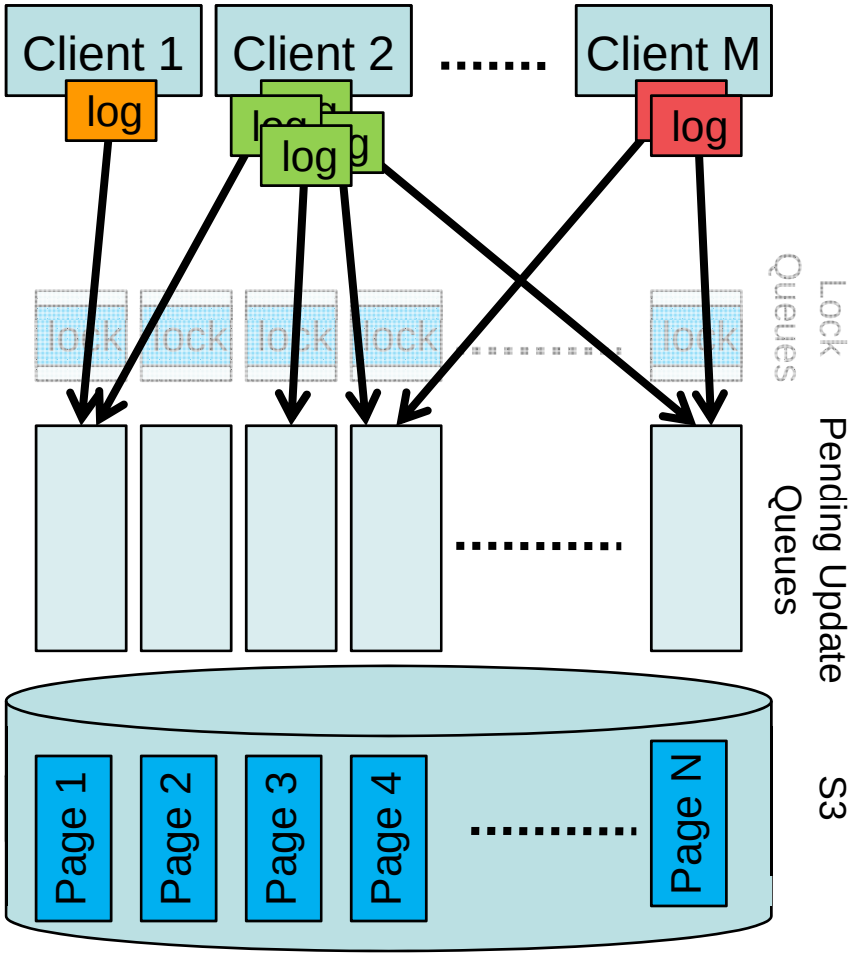
Clients commit update records to PU-Queues



Basic Protocol

Step 1: Commit

Clients commit update records to PU-Queues

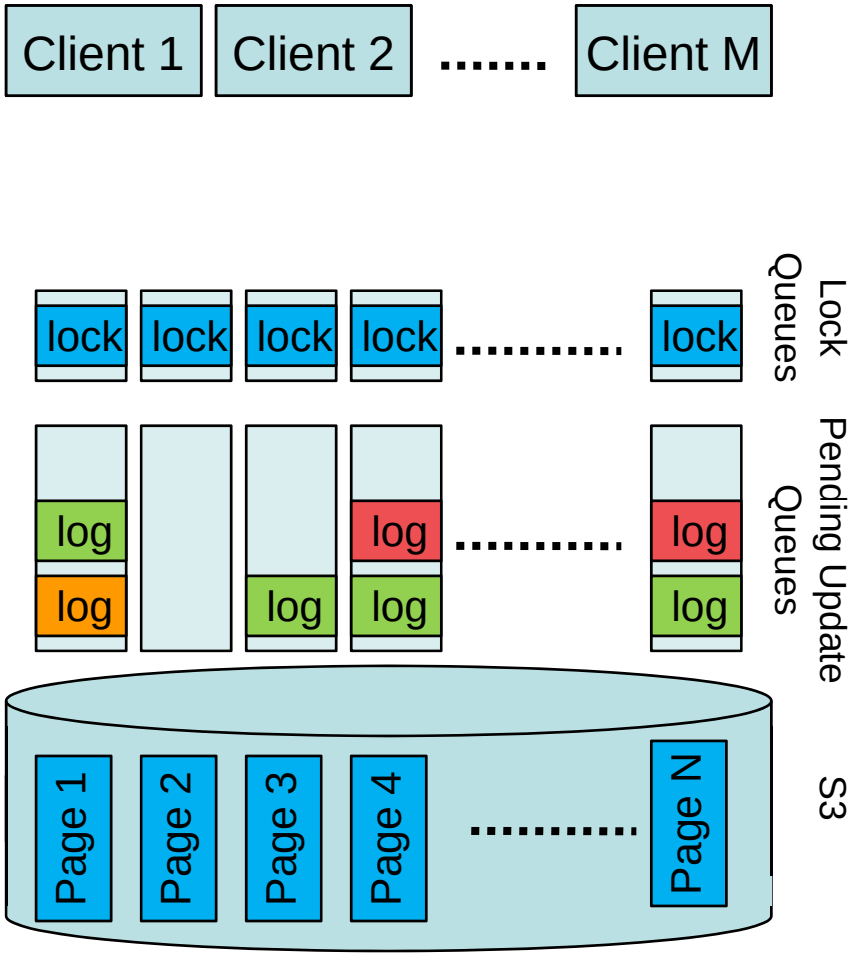


Basic Protocol

Step 1: Commit

Clients commit update records to PU-Queues

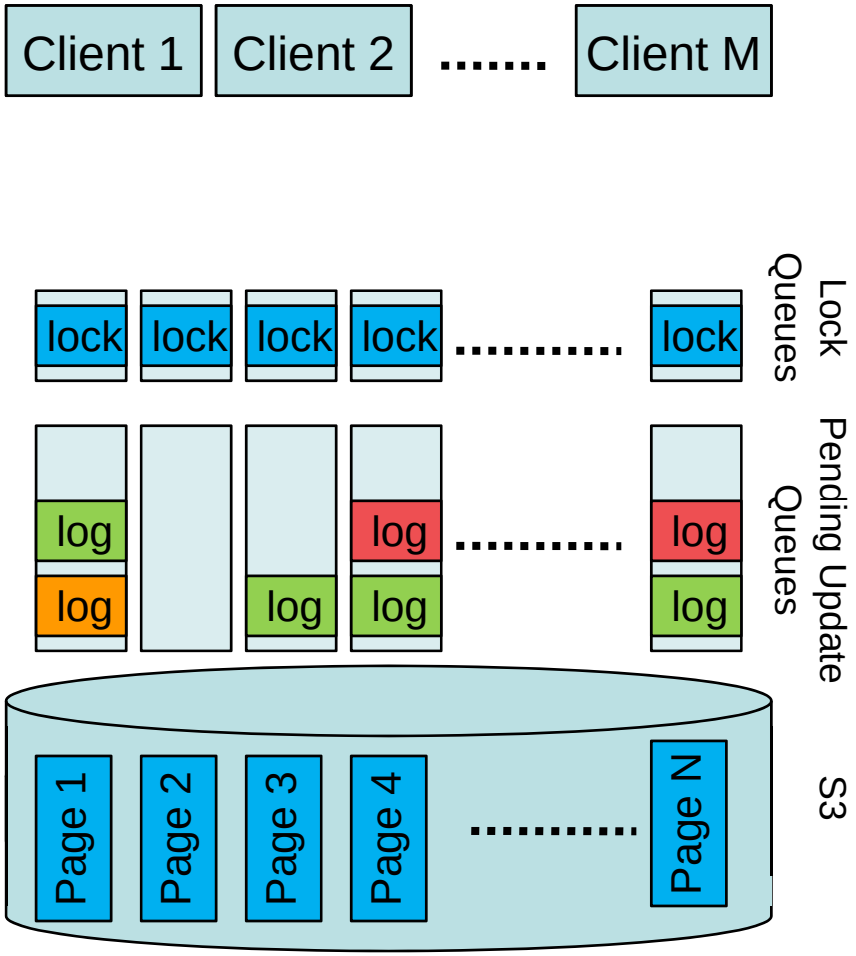
- Commit of the transaction
- Transaction is finished



Basic Protocol

Step 2: Checkpointing

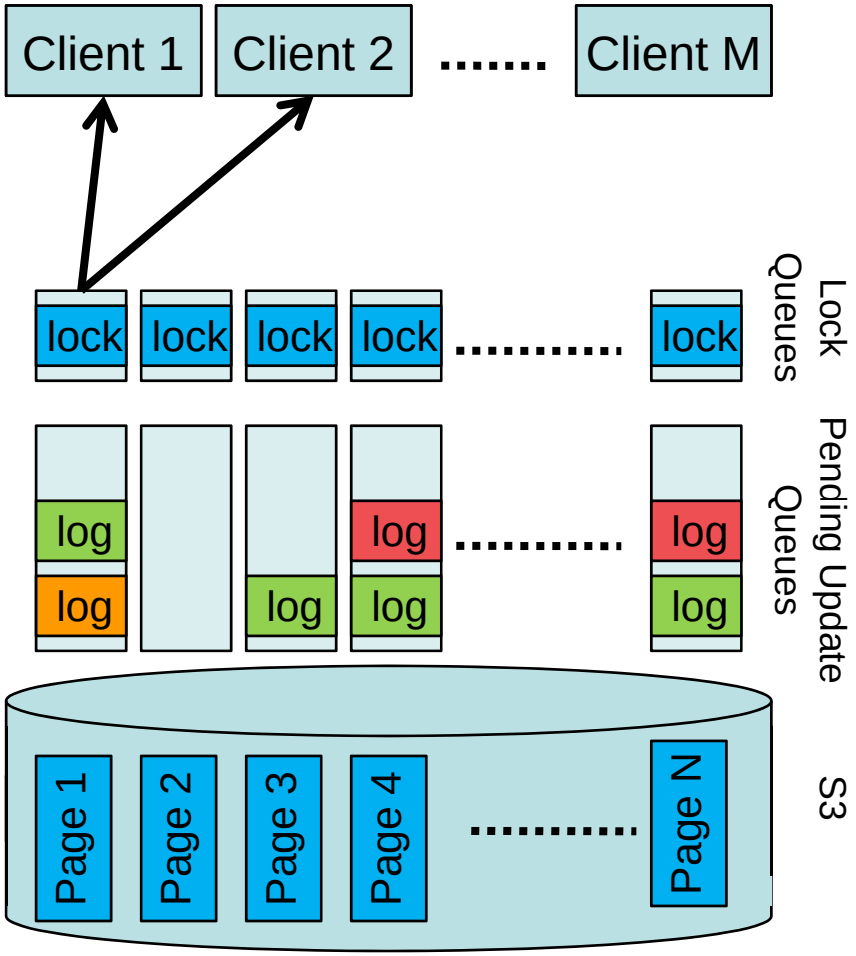
- Checkpointing propagates updates from SQS to S3
- Updates become visible on S3
- Checkpointing requires synchronization
- Achieved by using SQS as exclusive locks



Basic Protocol

Step 2: Checkpointing

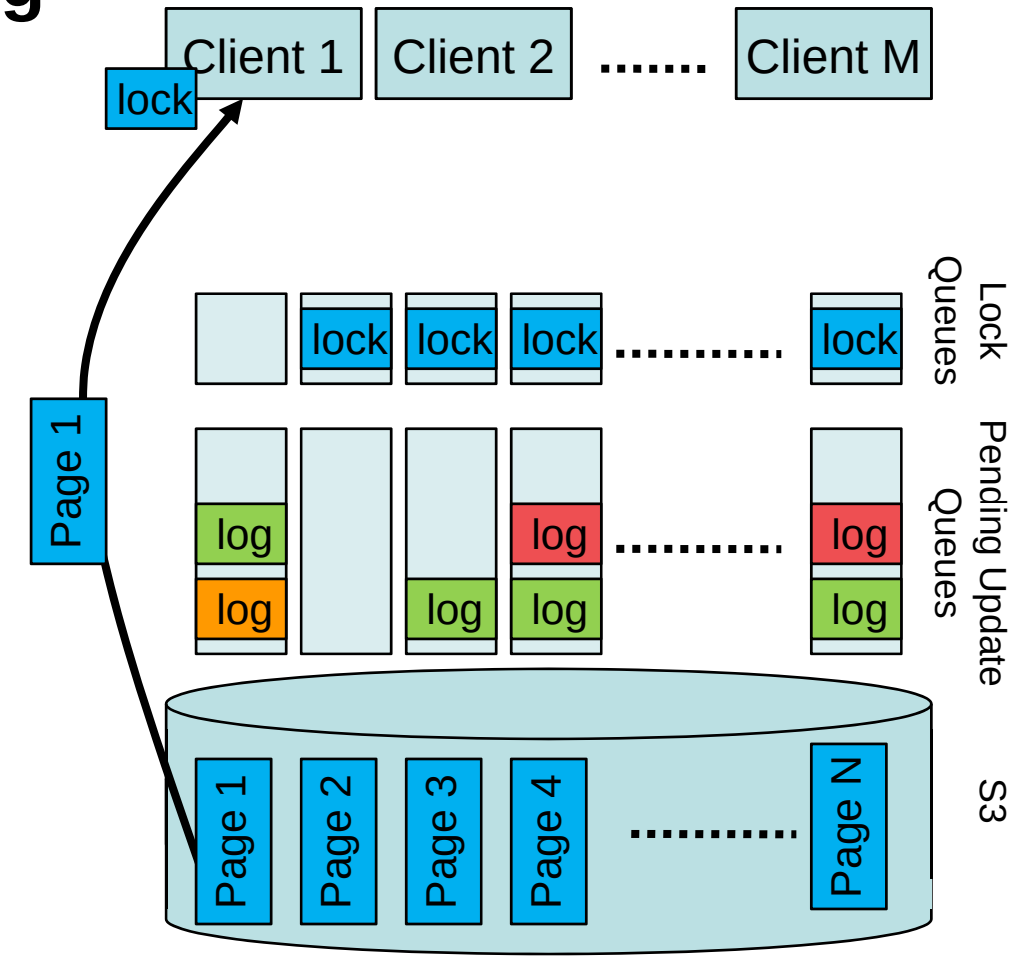
1. Receive Lock



Basic Protocol

Step 2: Checkpointing

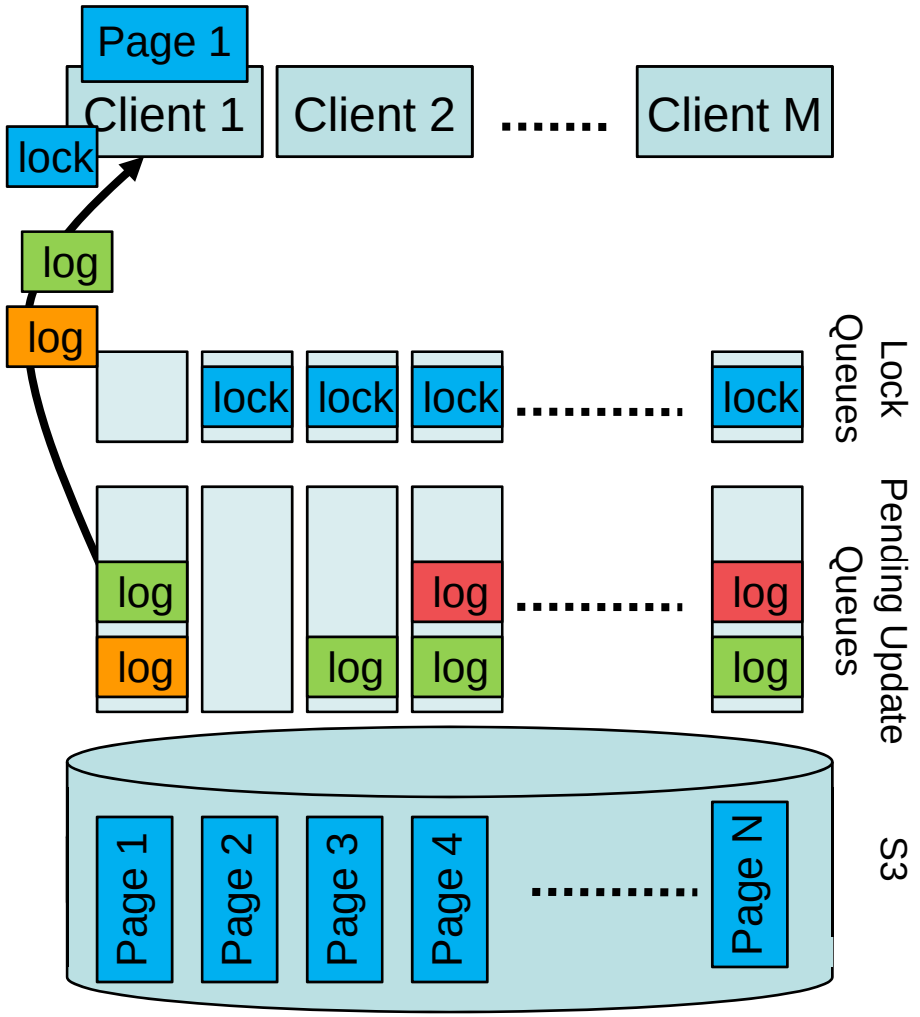
1. Receive Lock
2. Refresh Page



Basic Protocol

Step 2: Checkpointing

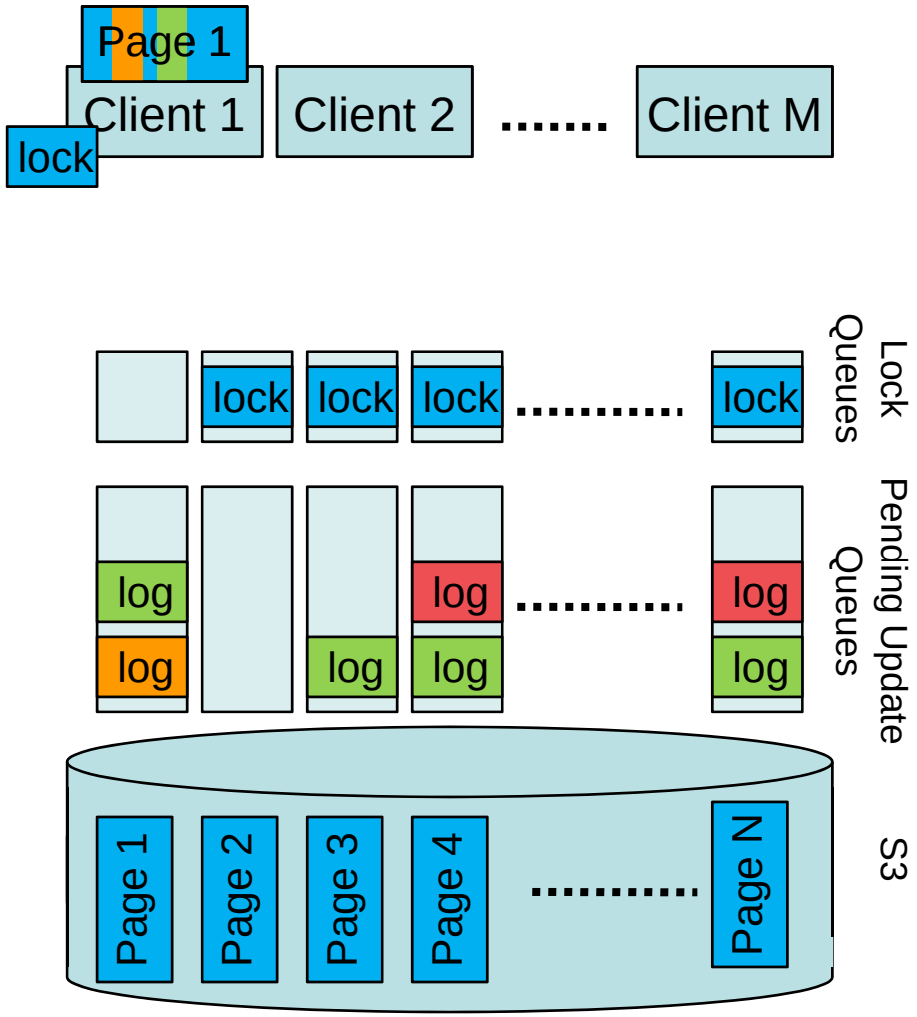
1. Receive Lock
2. Refresh Page
3. Receive Messages: as many as possible



Basic Protocol

Step 2: Checkpointing

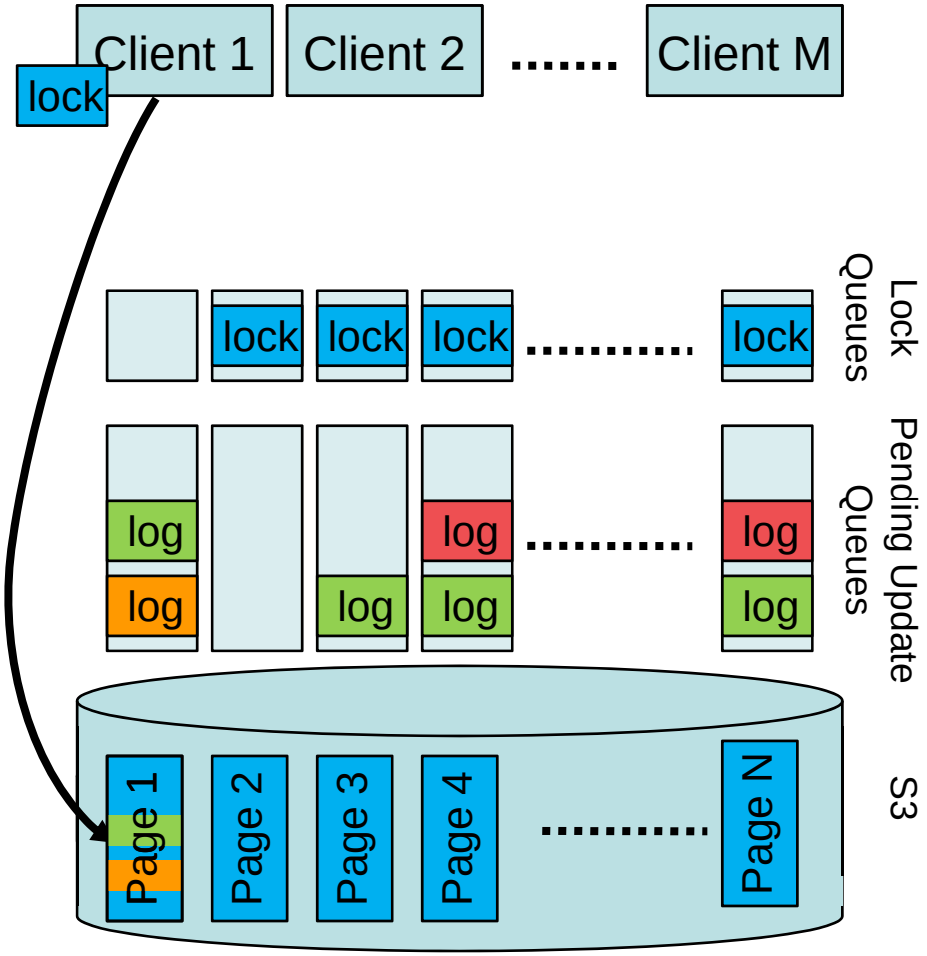
1. Receive Lock
2. Refresh Page
3. Receive Messages: as many as possible
4. Apply the log records to the cached page



Basic Protocol

Step 2: Checkpointing

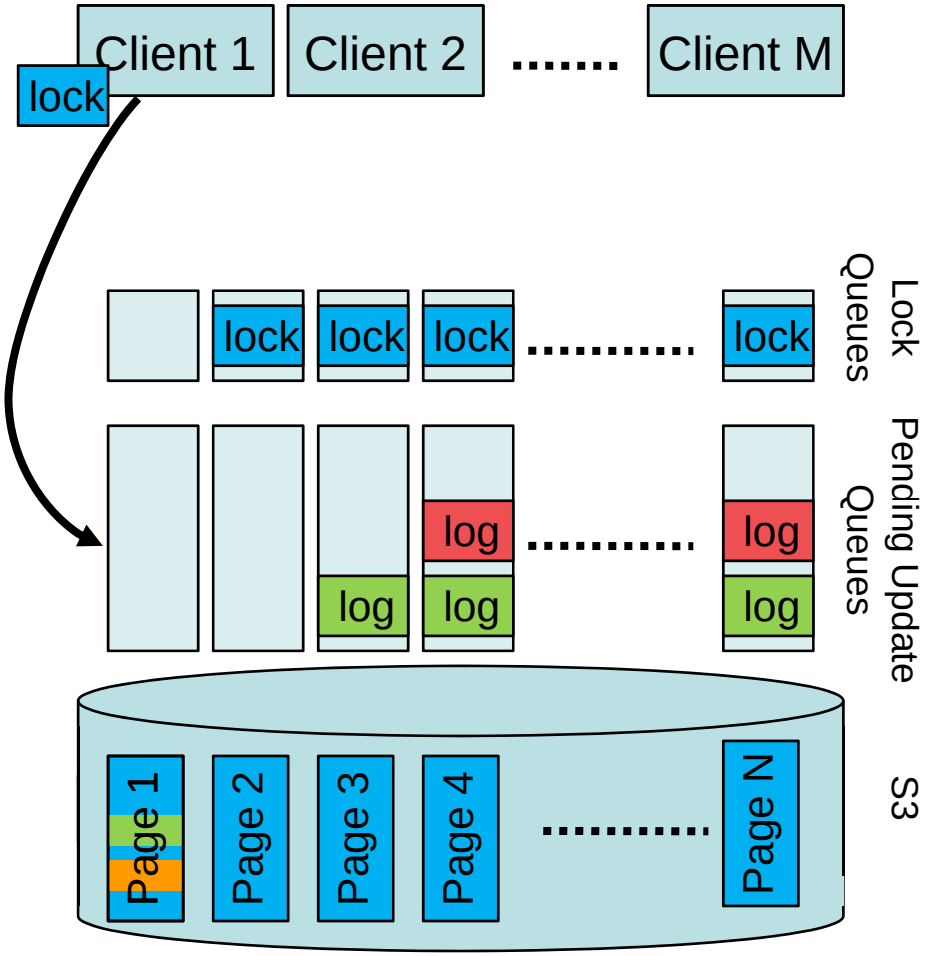
1. Receive Lock
2. Refresh Page
3. Receive Messages: as many as possible
4. Apply the log records to the cached page
5. Put the new version of the page to S3



Basic Protocol

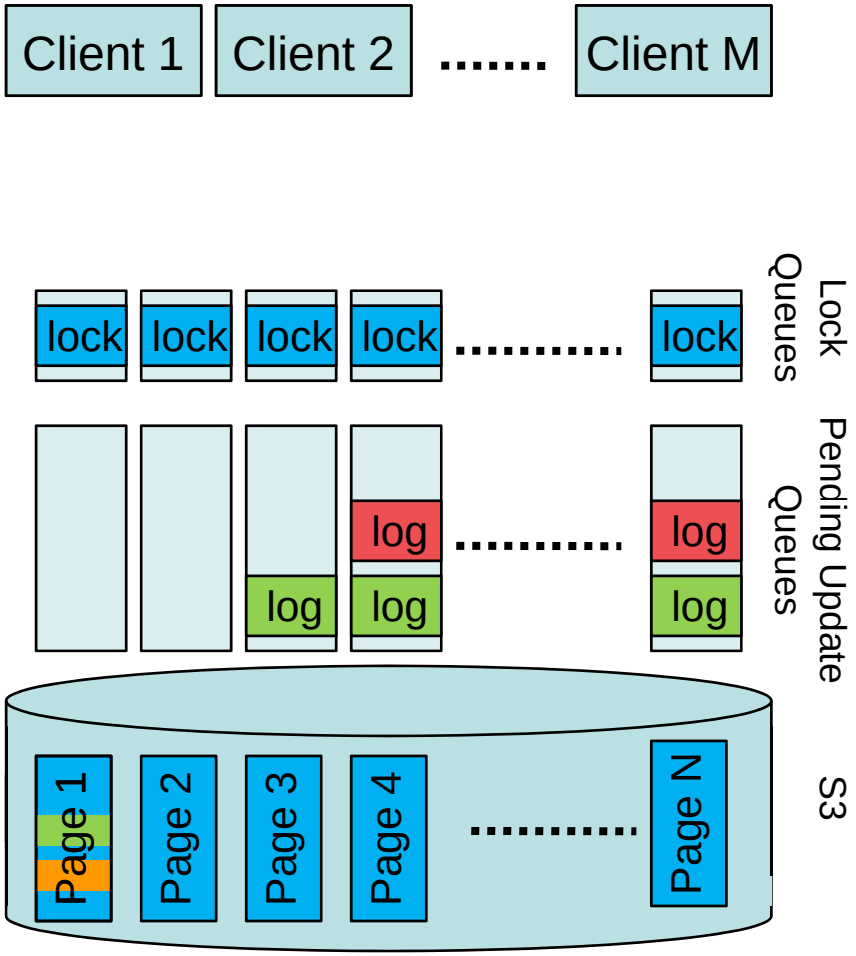
Step 2: Checkpointing

1. Receive Lock
2. Refresh Page
3. Receive Messages: as many as possible
4. Apply the log records to the cached page
5. Put the new version of the page to S3
6. Delete all the log records which were received in Step 3



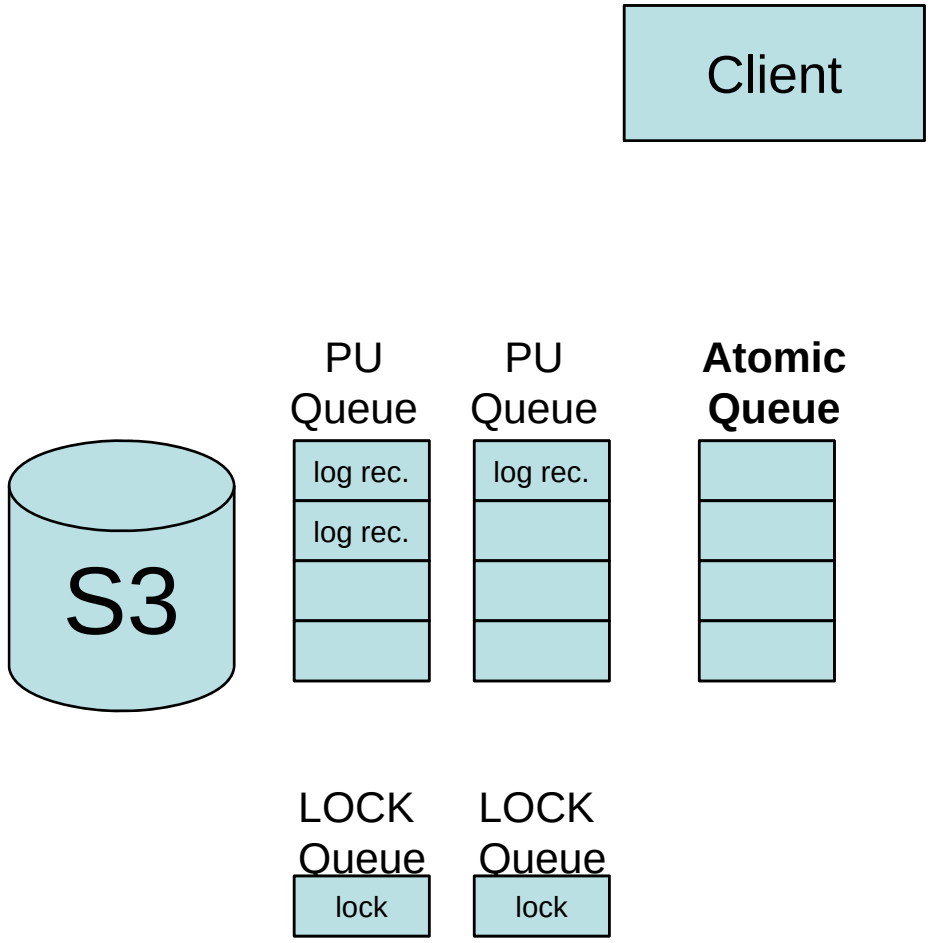
Basic Protocol

- Extremely simple
- No additional infrastructure (except SQS) is needed
- Protocol is also resilient to failures
  - Applying a log record twice does not harm as log records are idempotent
- Protocol has a price: dollar and freshness of the data
- **Still weak consistency/ transactional properties,**
  - No atomicity
  - No monotonic guaranties: Ordering of the log record messages is not important
  - No concurrency control on record level



Atomicity: All or none of the updates of a transaction become visible

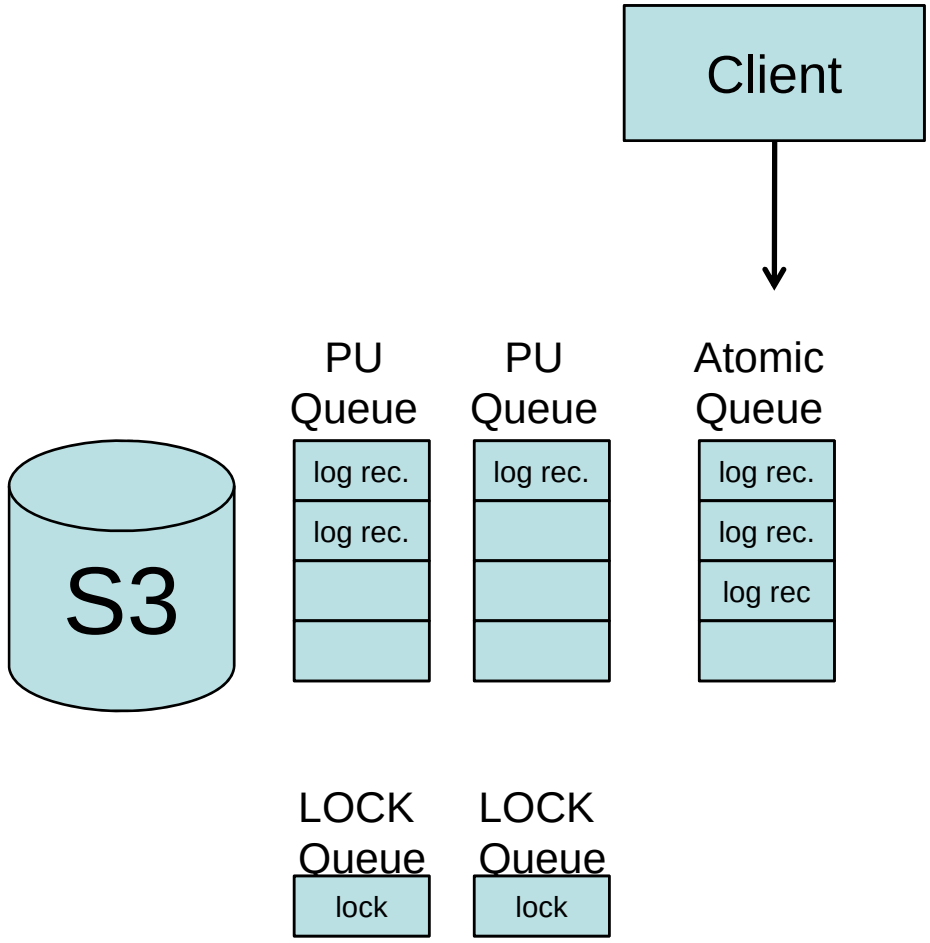
Each client manages an atomic queue.



Atomicity: All or none of the updates of a transaction become visible

Commit Protocol

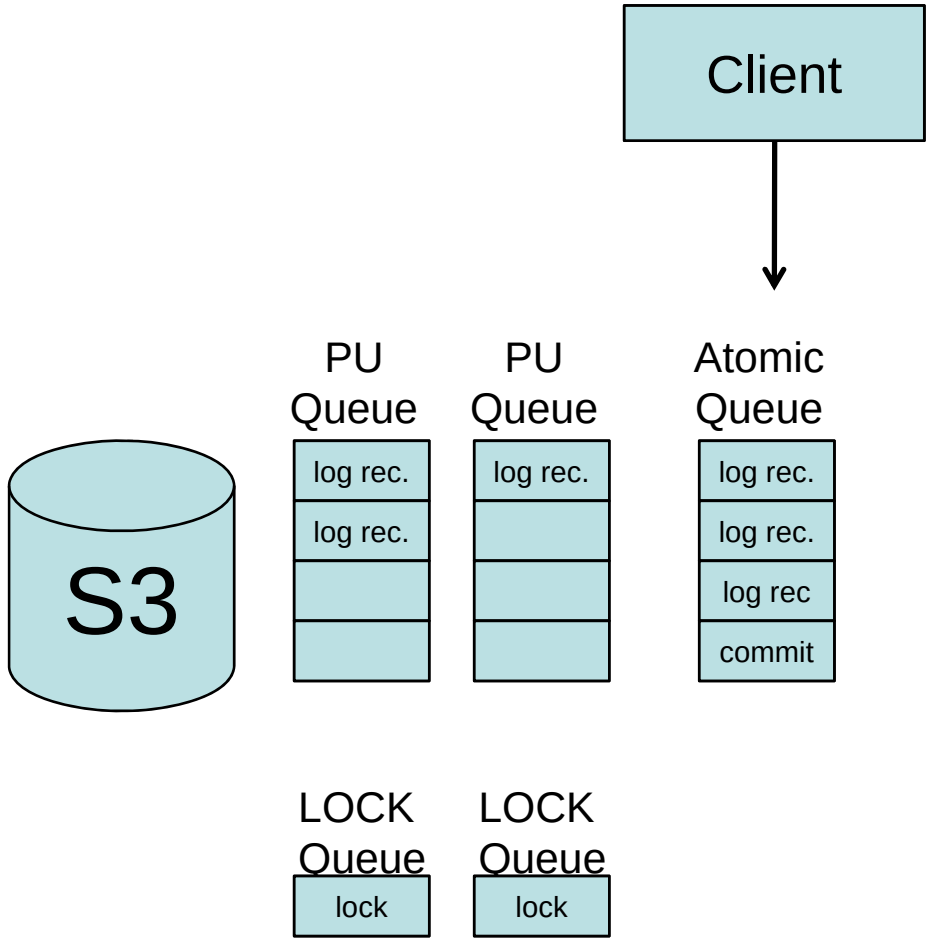
1. Send all log records to the ATOMIC queue.



Atomicity: All or none of the updates of a transaction become visible

Commit Protocol

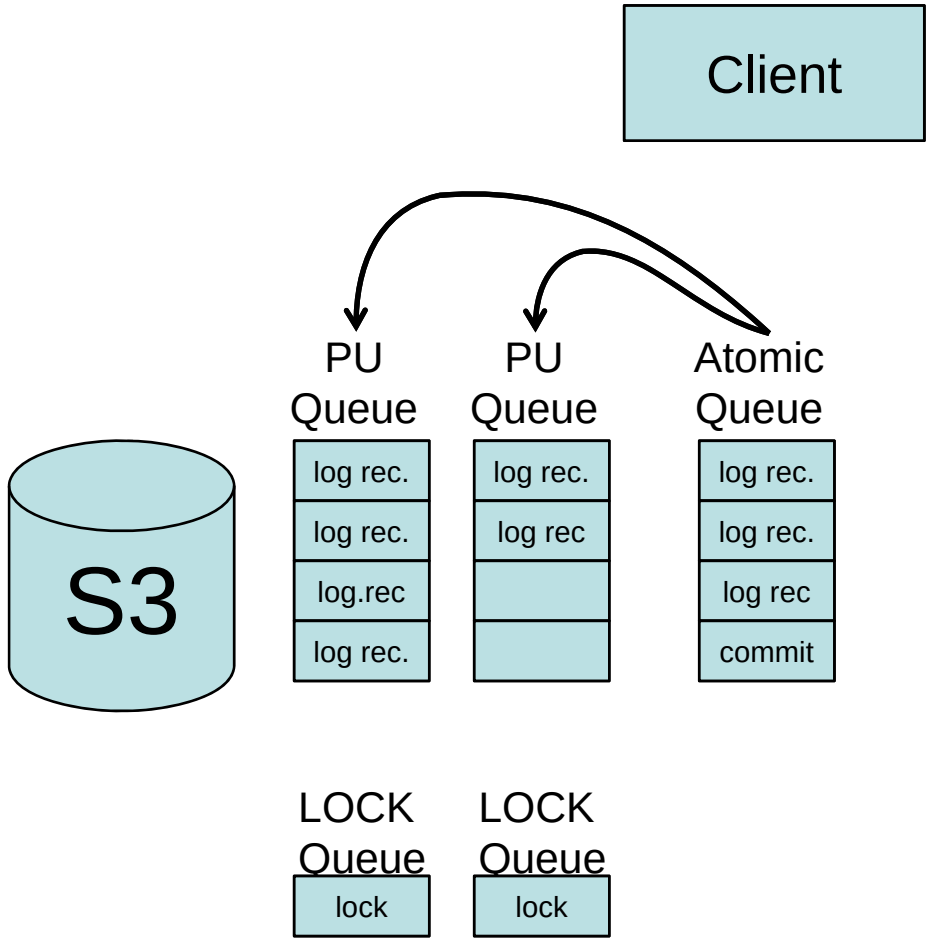
1. Send all log records to the ATOMIC queue.
2. Send commit log record.



Atomicity: All or none of the updates of a transaction become visible

Commit Protocol

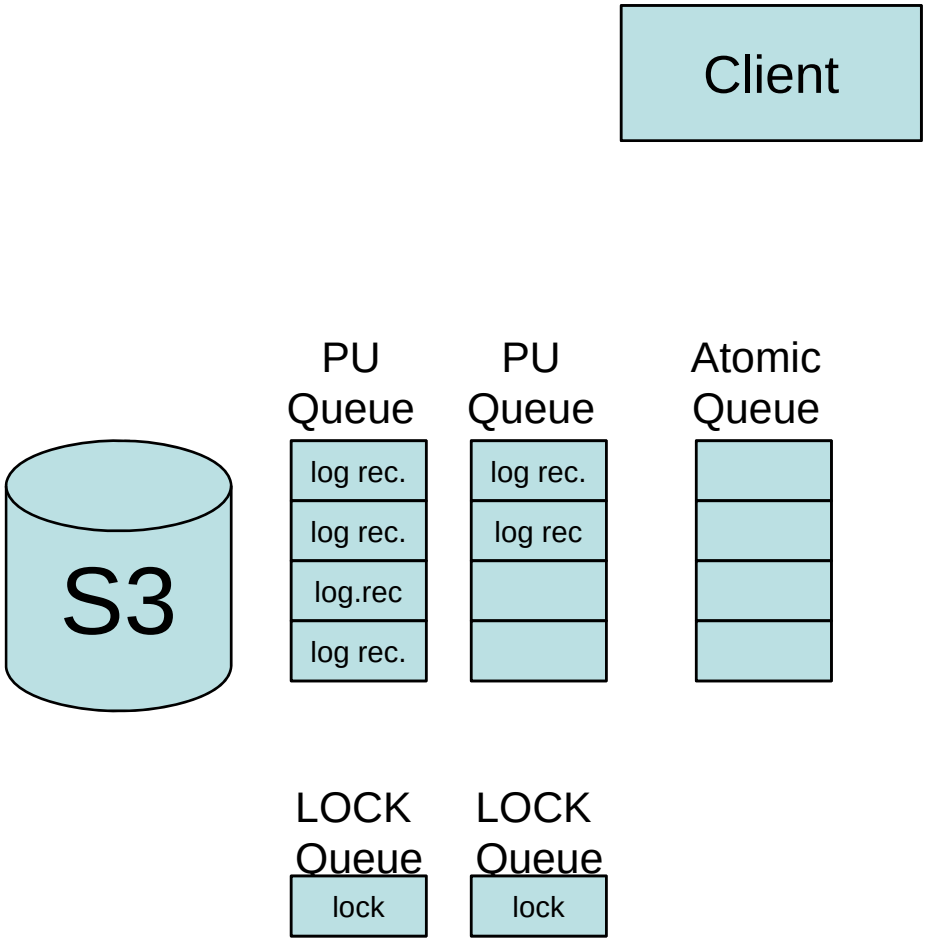
1. Send all log records to the ATOMIC queue.
2. Send commit log record.
3. Send all log records to the corresponding PU queues.



Atomicity: All or none of the updates of a transaction become visible

Commit Protocol

1. Send all log records to the ATOMIC queue.
2. Send commit log record.
3. Send all log records to the corresponding PU queues.
4. Delete all message after committing.



Atomicity cont'd.

- When a client fails, the client checks its ATOMIC queue at restart
- Winners are all log records which carry the same id as one of the commit records found in the ATOMIC queue; all other log records are losers
- Winners are propagated, others deleted

Plan of Attack

- Step 1: Use S3 as a huge shared disk
  - leverage scalability, no admin features
- Step 2: Allow concurrent access to shared disk in a distributed system
  - keep properties of a distributed system, maximize consistency
- Step 3: Do application-specific trade-offs
  - consistency vs. cost
  - consistency vs. availability
  - *consistency à la carte* (levels of consistency)

Experiments and Results

- Goal: Studying the trade-offs in terms of **consistency**, **latency** and **cost ($)**
- We used a sub-set of the TPC-W benchmark (models a bookstore scenario)
- All experiments were done with a complex customer transaction involving the following steps:
  a) retrieve the customer record from the database;
  b) search for six specific products;
  c) place orders for three of the six products.

Running Time per Transaction [secs]

	Avg.	Max.
Naïve (Shared-Disk)	11.3	12.1
Basic (Eventual Consistency)	4.0	5.9
Monotonicity	4.0	6.8
Atomicity + Monotonicity	2.8	4.6

- Doesn't include checkpointing (asynchronously)
- Every transaction simulates around 12 clicks
- Time is less than a sec. per click
- Time is independent of the number of clients

→It's fast. Not 15K-SCSI-RAID0-fast, but internet-latency-fast.

Cost per 1000 Transactions ($)

	Step1: Commit	Step2: Checkpoint + Atomic Queue	Total
Naïve (Shared-Disk)	0.15	0	**0.15**
Basic (Eventual Consistency)	0.7	1.1	**1.8**
Monotonicity	0.7	1.4	**2.1**
Atomicity + Monotonicity	0.3	2.6	**2.9**

- Interaction with SQS becomes expensive
- For a bookstore, a transactional cost of about 3 milli-dollars (i.e., 0.3 cents)
- Especially updates have a big influence on the cost

→Not cheap, but in many scenarios affordable

Summary and Future Work

- Architecture allows **transparent scaling**:
  No need to change code, hardware,...
- Consistency is a goal, not a constraint:
  Consistency à la carte for your applications
- Amazon's WebServices are a viable candidate for many Web 2.0 and interactive applications
- Future work:
  - SimpleDB as the main index
  - EC2 as application server
  - Further studies of stronger consistency protocols

Thank you for your interest

Questions?

Contact:
tim.kraska@inf.ethz.ch

Cost per 1000 Transacts., Various Checkpoint Intervals