HBase Coprocessors and Security

Mingjie Lai, Apache Incubator Flume Committer,
Software Engineer at Trend Micro
mlai@apache.org

Overview

- A quick look at the latest HBase developments
- Focus on coprocessors, moving computation to the data
  - Observers: triggers on steroids
  - Endpoints: custom RPC servers
- HBase security, a coprocessor case study
  - Authentication: Kerberose + DIGEST-MD5 authentication using signed tokens
  - Authorization via AccessController coprocessor
    - Endpoint for maintaining ACLs
    - Observer for access control checks
    - Endpoint for obtaining authentication tokens
  - Securing ZooKeeper

HBase is Rapidly Evolving

- 0.92 release status
  - RC coming very soon
  - Support latest Hadoop versions: 0.20.205
  - Distributed WAL splitting
  - Region Server graceful decommission
  - Coprocessors
  - Security
  - Many more internal improvements, bug fixes and doc updates

Coprocessors

- Inspired by Bigtable coprocessors, described in Jeff Dean's LADIS '09 keynote talk
  - Arbitrary code that runs at each tablet in tablet servers
  - High-level call interface for clients
  - Calls addressed to rows or ranges of rows, mapped to locations and parallelized by client library
  - Automatic scaling and request routing for application logic along with data
- For HBase, a whole new interface to your data and how it's stored!
  - Embed custom processing with the data
  - Enhance or override behavior of core code
  - Define and export custom RPC protocols

Use Cases

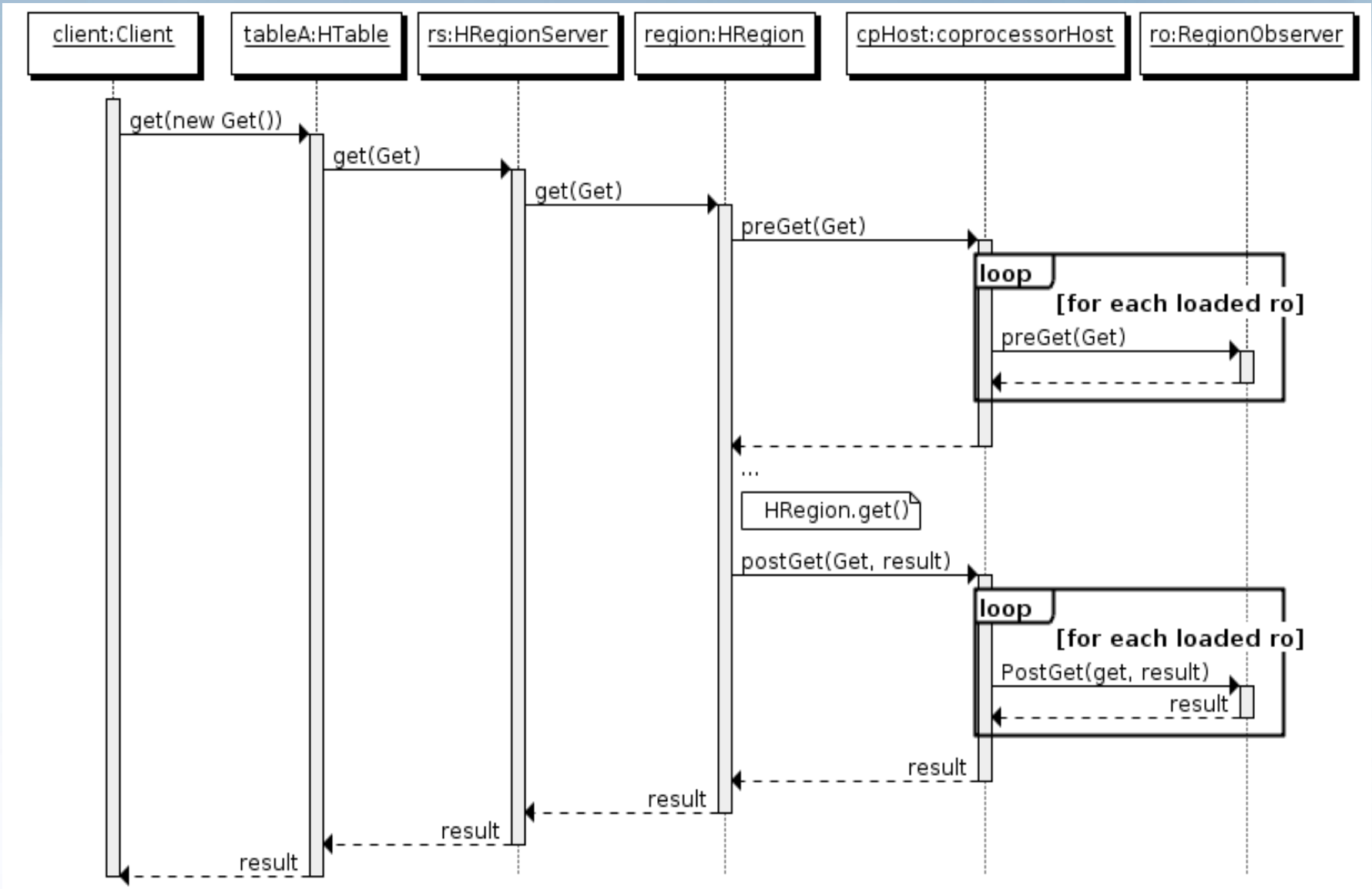
- Use case 1: column aggregation:
  - SQL-like: select count(*) from usertable
  - Option 1: Client side scan
  - Option 2: a map/reduce job
  - Option 3: use a counter
  - Option 4: a coprocessor
- Use case 2: customize WAL behavior
  - Ignore certain columns to WAL
- Use case 3: security
  - Check user privilege before a request, at region server

Observers

- ***Like database triggers***: provide event-based hooks for interacting with normal operations
- `RegionObserver`
  - CRUD or DML type operations
  - Pre/post-hooks for Get, Put, Scan, etc operations on table data
  - Can append, substitute it's own results for response to client
  - Can override normal processing of requests
- `MasterObserver`
  - DDL or metadata operations and cluster administration
- `WALObserver`
  - Write-ahead-log appending and restoration

Observers: Execution

- Multiple observers can be chained together

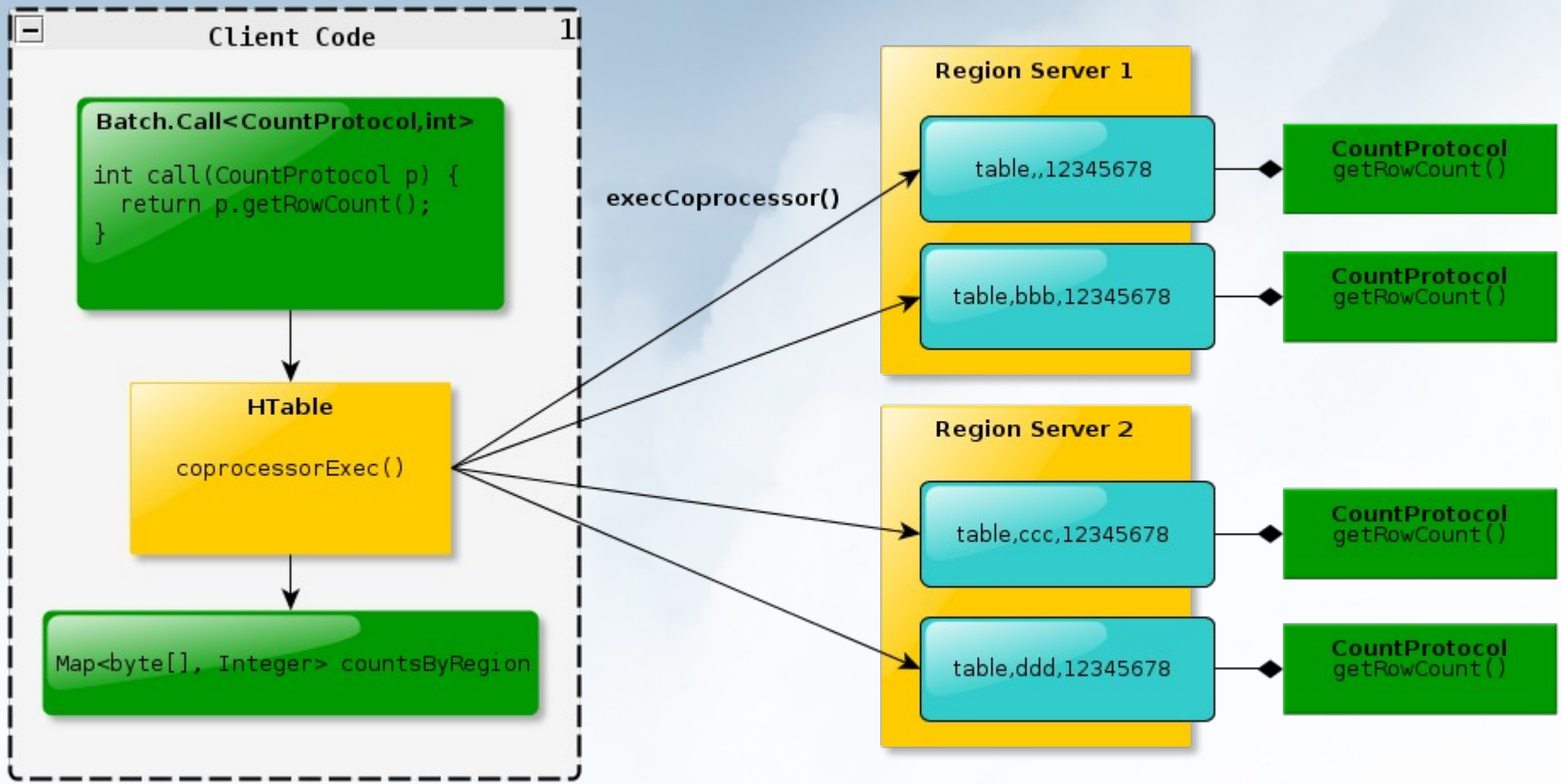


Endpoints: Custom RPC APIs

- ***Like stored procedures***: custom RPC methods called explicitly with parameters
- Loaded per table-region
  - Custom code executes in context with region data
- Only supported on Region Servers
- Clients call APIs over a single row or a row range
  - Framework translates row keys to region locations
  - Parallel execution

Endpoints: Call Routing

- Clients call RPC methods defined in a custom interface
  - `HTable.coprocessorExec()` call defines a row range
  - Ranges spanning regions are split and parallelized



Coprocessor Management

- Class loading:
  - Load from configuration:
    - class names as HBase configurations
    - will be picked up when region/master is opened, as default coprocessors
  - Load from shell: only for non-global, region level coprocessors
- Show loaded coprocessors from Web UI and shell

Committed Projects

- Security
  - `AccessController` Observer intercepts calls and performs authorization checks on calling client
  - `TokenProvider` exposes custom API to obtain authentication tokens when secure RPC is loaded
- Aggregate operations
  - Simple aggregate functions can be executed over table data using custom APIs
  - Functions exposed using a custom RPC protocol

Ongoing and Future Possibilities

- HBASE-3529: Embedded search
  - `RegionObserver` updates embedded Lucene indexes on data changes
  - Custom protocol provides search methods
- Parallel computation framework
  - Hadoop MapReduce API (mappers, reducers, partitioners, intermediates) but parallel region MapReduce ?
- Server-side dynamic scripting execution
  - Embedded JRuby scriptlets for queries or custom processing
- Eventually consistent secondary indexing

Need to Improve

- Code weaving:
  - Allow arbitrary code execution right now
  - Use a rewriting framework like ASM to weave in policies at load time
  - Improve fault isolation and system integrity protections
  - Wrap heap allocations to enforce limits
  - Monitor CPU time
  - Reject APIs considered unsafe
- Documentation!

Security: A Case Study

- A bit of background
  - No real security in Hadoop 0.20.2 and prior
    - User impersonation trivial
    - No mutual client/server authentication
    - File permission enforcement assumes good actors
    - Instead clusters secured at the perimeter
  - Secure Hadoop (0.20.205) made secure HBase possible
    - Strong authentication using Kerberos
    - Mutual authentication of RPC connections
    - Data isolation at HDFS level
    - Multiple groups can share the same cluster

Security: Overview

- Target Application
  - User isolation (control over your data)
  - Multi-tenancy: private and public cloud
- Goals
  - Strong authentication of HBase clients
  - Mutual authentication of RPC connections
  - User data is private unless access has been granted
  - Access to data can be granted with table, column family, or column qualifier granularity
  - Simple administration (relatively!)
- Non-Goals
  - Row-level or per value (cell) ACLs
  - Push down of file ownership to HDFS
  - Full Role Based Access Control

Security: Implementation

- Secure RPC
  - Separate loadable SecureRPCEngine
  - Provides SASL authentication of clients
    - Kerberos/GSSAPI authentication
    - DIGEST-MD5 authentication using signed tokens for Map Reduce
- Authorization of all operations
  - Simple access control lists
    - READ, WRITE, EXEC, CREATE, ADMIN permissions
  - Permissions grantable at table, column family, and column qualifier granularity
  - Supports user based assignment
- Secure ZooKeeper
  - SASL based authentication using Kerberos
  - Uses existing ZooKeeper ACLs

Security: AccessController

- Heart of HBase security
- Access Control Lists (ACLs)
  - `AccessControllerProtocol` exposes RPC calls to update and query stored user permissions
  - ZooKeeper listeners synchronize ACL changes throughout the cluster
- Authorization
  - `RegionObserver` implementation mediates data operations (like DML)
  - `MasterObserver` implementation mediates metadata operations (like DDL)

AccessController as Endpoint

- Implements `AccessControllerProtocol`

```
public interface AccessControllerProtocol
    extends CoprocessorProtocol {

  /** Add a new permision for a user */
  boolean grant(byte[] user, TablePermission permission)
      throws IOException;

  /** Remove a currently granted permisson */
  boolean revoke(byte[] user, TablePermission permission)
      throws IOException;

  /** Return the currently granted permissions for a table */
  List<UserPermission> getUserPermissions(byte[] tableName)
      throws IOException;
}
```

AccessController as Endpoint

- Clients call `AccessControllerProtocol` methods for updates

```
// .META. table stores ACLs
HTable meta = new HTable(HConstants.META_TABLE_NAME);

// We use a dynamic proxy to invoke the RPC protocol
AccessControllerProtocol proxy = meta.coprocessorProxy(
    AccessControllerProtocol.class,
HConstants.EMPTY_START_ROW);

proxy.grant("sampleuser",
    new TablePermission("mytable", Action.READ, Action.WRITE));

List<UserPermission> perms =
proxy.getUserPermissions("mytable");
// now includes an entry for "sampleuser"
```

- Wrapped in shell already

AccessController: Authorization

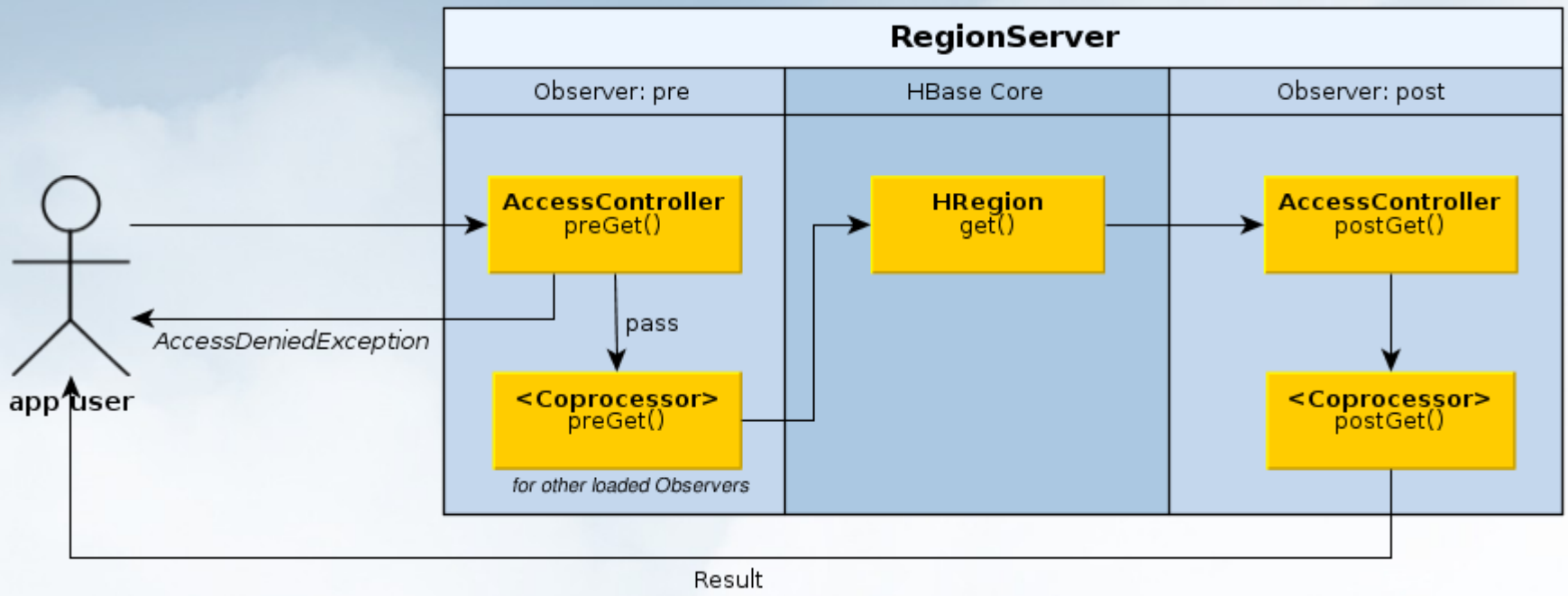
- Implements `RegionObserver` and `MasterObserver`
  - preXXX() methods perform access-control checks before allowing processing to continue

```
/** Called prior to normal execution of a Get operation */
public void preGet(final ObserverContext<~> ctx,
   final Get get, final List<KeyValue> result)
throws IOException {

  if (!permissionGranted(TablePermission.Action.READ, ctx,
      get.getFamilyMap())) {
    throw new AccessDeniedException(
        "Insufficient permissions (table=" + getTableName(ctx)
        + ", action=READ)");
  }
}
```

AccessController: Authorization

- `AccessController.preGet()` checks for authorization of client credentials
  - failure generates `AccessDeniedException`
  - success continues through normal processing



Secure ZooKeeper

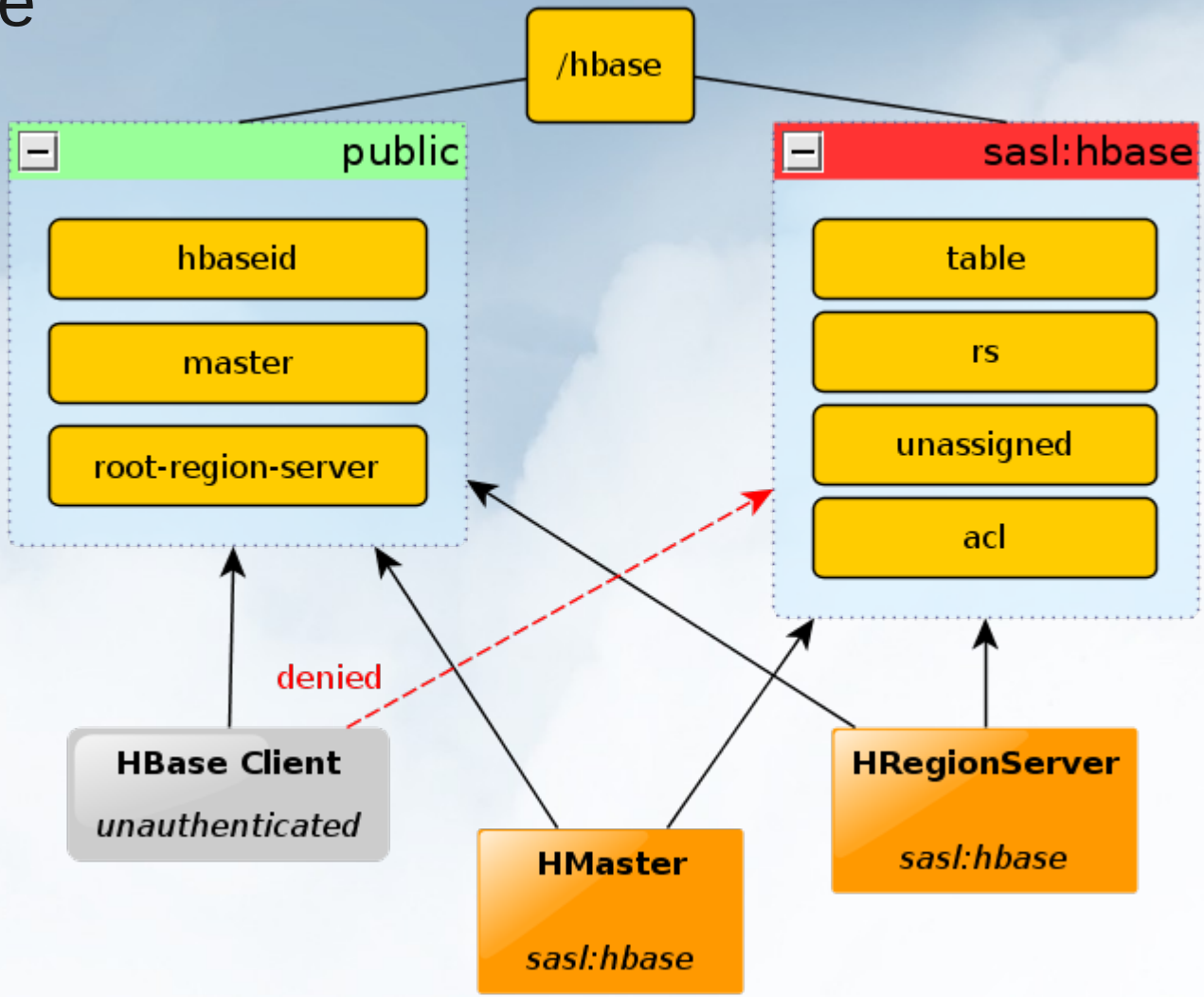
- ZooKeeper plays a critical role in HBase cluster operations and in the security implementation
  - Root catalog table location
  - Region assignment
  - Server "liveness"
  - Synchronizes ACLs throughout cluster
  - Synchronizes secret key rolling for token authentication
- Needs strong security or it becomes a weak point
  - Kerberos-based client authentication
  - Znode ACLs enforce SASL authenticated access for sensitive data

Secure ZooKeeper: RPC

- `SASLAuthenticationProvider` supports Kerberos authentication via GSSAPI
- `LoginThread` on client periodically refreshes credentials
- Clients can fallback to unauthenticated access (if allowed)
  - Supports mixed usage of a single ZooKeeper cluster by secure, and unsecured clients
  - ZooKeeper ACLs can still restrict sensitive znode access to authenticated users

Secure ZooKeeper: ACLs

- znodes needed for client operations are public
- znodes critical to cluster operation and security are private



Coprocessor and Security Feature Status

- All code has submitted to HBase, ZooKeeper
  - HBase 0.92
    - HBASE-2000: Coprocessor umbrella
    - HBASE-3025: Coprocessor based access control
    - HBASE-2742: Provide strong authentication with a secure RPC engine
    - etc.
  - Zookeeper 3.4:
    - ZOOKEEPER-938: Support Kerberos authentication of clients
- Need more documentation and blog posts on getting it running

For More Information

- *HBase Coprocessors*, by Mingjie Lai
  http://hbaseblog.com/2010/11/30/hbase-coprocessors/
- HBase Token Authentication
  http://wiki.apache.org/hadoop/Hbase/HBaseTokenAuthentication
- Hadoop Security Design
  http://github.com/trendmicro/hbase/tree/security

Thank You

Questions?