Principles of Software Construction: Objects, Design and Concurrency

Introduction to Distributed Systems, continued

15-214
toad

Spring 2013

Christian Kästner **Charlie Garrod**

School of Computer Science
institute for SOFTWARE RESEARCH

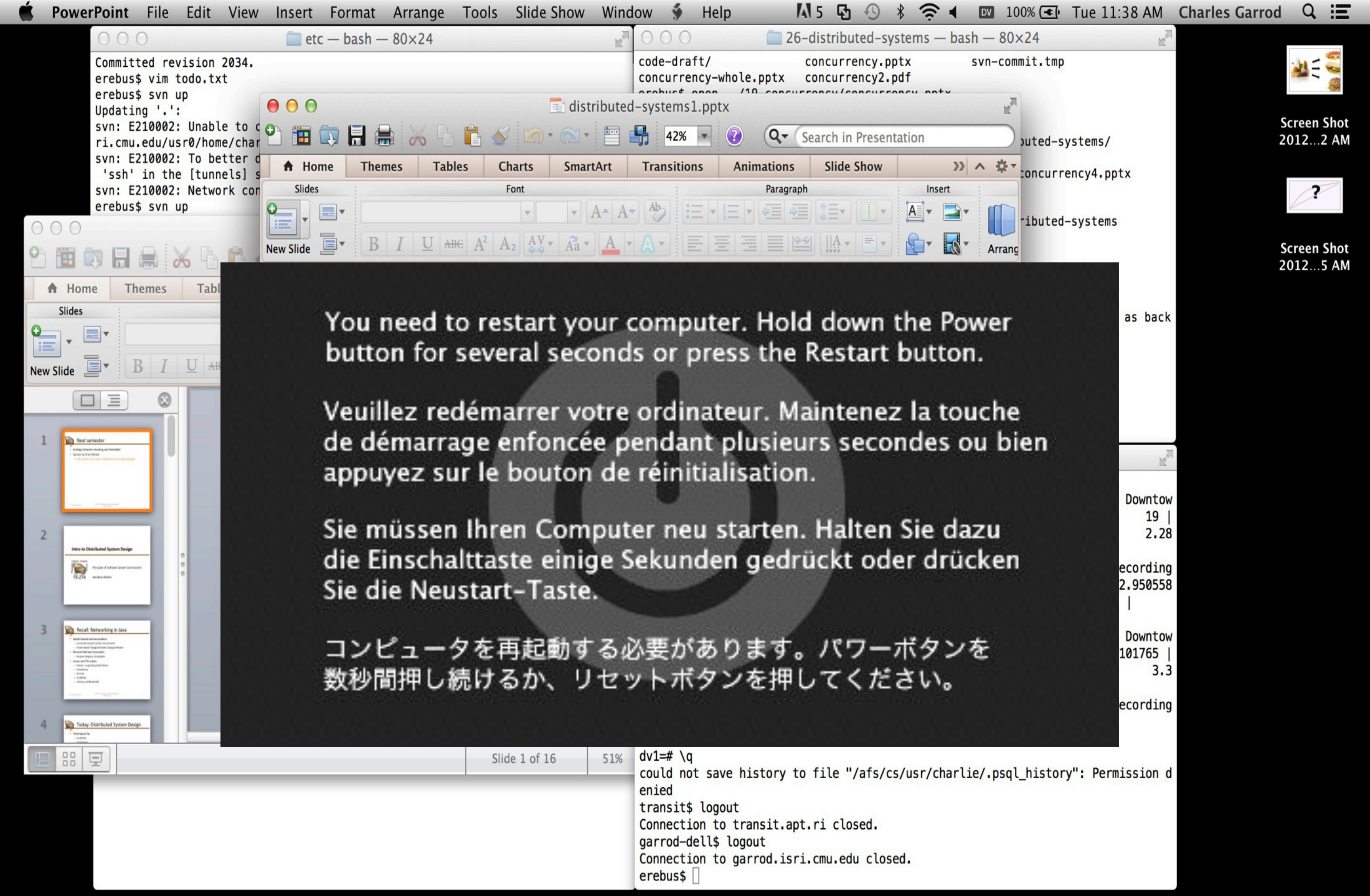
Administrivia

- Homework 5c framework info coming soon

Key topics from Tuesday

Higher levels of abstraction

- Application-level communication protocols
- Frameworks for simple distributed computation
  - Remote Procedure Call (RPC)
  - Java Remote Method Invocation (RMI)
- Common patterns of distributed system design
- Complex computational frameworks
  - e.g., distributed map-reduce


PowerPoint File Edit View Insert Format Arrange Tools Slide Show Window Help
Tue 11:38 AM Charles Garrod
etc — bash — 80×24
Committed revision 2034.
erebus$ vim todo.txt
erebus$ svn up
Updating '.':
erebus$ svn up
26-distributed-systems — bash — 80×24
code-draft/ concurrency.pptx svn-commit.tmp
concurrency-whole.pptx concurrency2.pdf
distributed-systems1.pptx
Search in Presentation
Home Themes Tables Charts SmartArt Transitions Animations Slide Show
Slides Font Paragraph Insert
New Slide
Slide 1 of 16
51%
You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.
Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.
Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.
コンピュータを再起動する必要があります。パワーボタンを数秒間押し続けるか、リセットボタンを押してください。
dv1=# \q
could not save history to file "/afs/cs/usr/charlie/.psql_history": Permission denied
transit$ logout
Connection to transit.apt.ri closed.
garrod-dell$ logout
Connection to garrod.isri.cmu.edu closed.
erebus$
Screen Shot 2012...2 AM
Screen Shot 2012...5 AM

Today

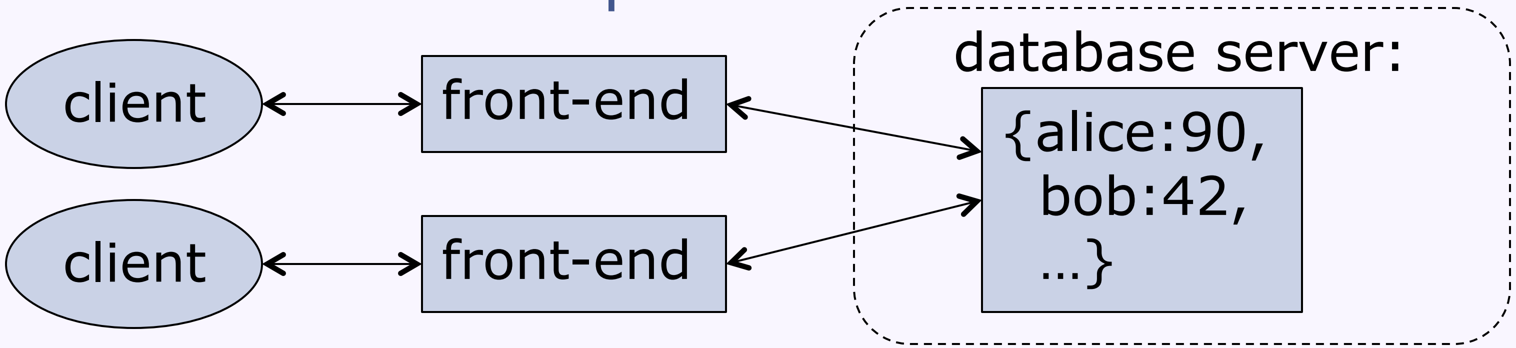
- Introduction to distributed systems
  - Motivation: reliability and scalability
  - Failure models
- Techniques for reliability and scalability
  - Replication
  - Partitioning

Metrics of success

- Reliability
  - Often in terms of availability: fraction of time system is working
    - 99.999% available is "5 nines of availability"
- Scalability
  - Ability to handle workload growth

A case study: Passive primary-backup replication

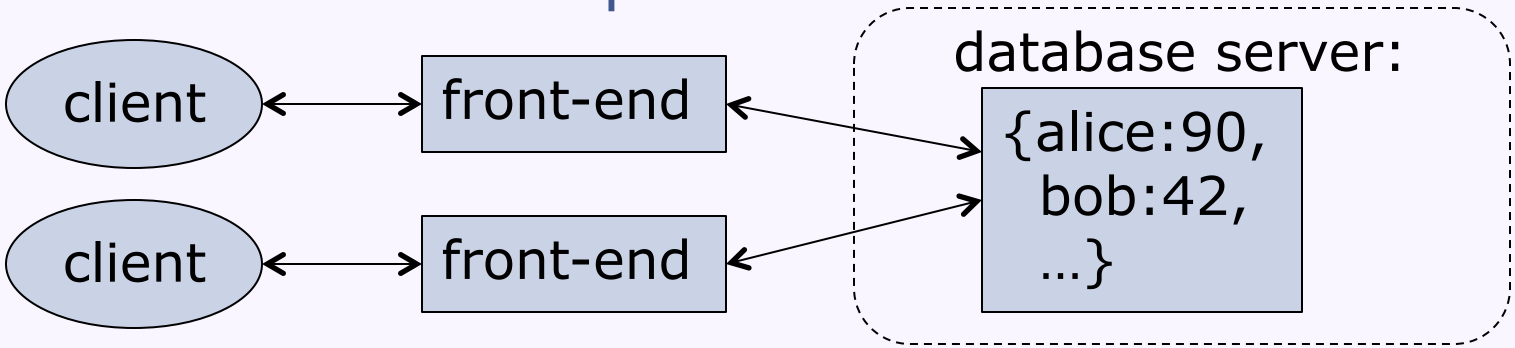
- Architecture before replication:



  - Problem: Database server might fail

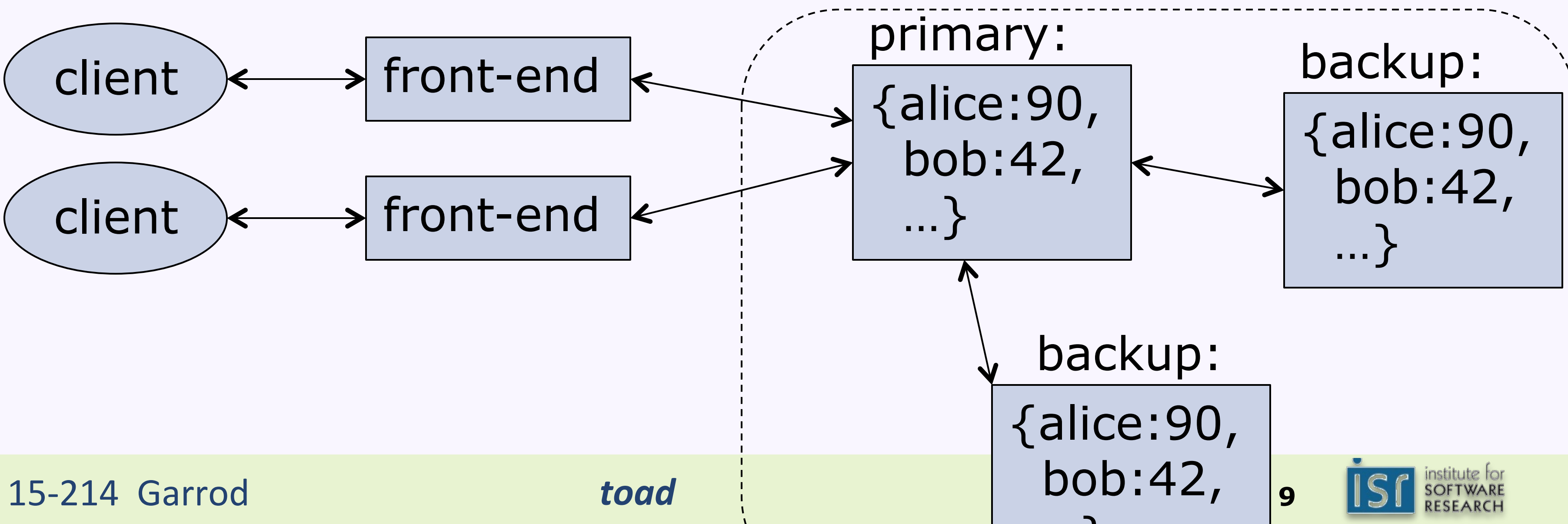
A case study: Passive primary-backup replication

- Architecture before replication:



  - Problem: Database server might fail

- Solution: Replicate data onto multiple servers



Passive primary-backup replication protocol

1. Front-end issues request with unique ID to primary DB
2. Primary checks request ID
   - If already executed request, re-send response and exit protocol
3. Primary executes request and stores response
4. If request is an update, primary DB sends updated state, ID, and response to all backups
   - Each backup sends an acknowledgement
5. After receiving all acknowledgements, primary DB sends response to front-end

Issues with passive primary-backup replication

- If primary DB crashes, front-ends need to agree upon which unique backup is new primary DB
  - Primary failure vs. network failure?
- If backup DB becomes new primary, surviving replicas must agree on current DB state
- If backup DB crashes, primary must detect failure to remove the backup from the cluster
  - Backup failure vs. network failure?
- If replica fails* and recovers, it must detect that it previously failed
- Many subtle issues with partial failures
- …

More issues…

- Concurrency problems?
  - Out of order message delivery?
    - Time…
- Performance problems?
  - 2n messages for n replicas
  - Failure of any replica can delay response
  - Routine network problems can delay response
- Scalability problems?
  - All replicas are written for each update, but primary DB responds to every request

Types of failure behaviors

- Fail-stop
- Other halting failures
- Communication failures
  - Send/receive omissions
  - Network partitions
  - Message corruption
- Performance failures
  - High packet loss rate
  - Low throughput
  - High latency
- Data corruption
- Byzantine failures

Common assumptions about failures

- Behavior of others is fail-stop (ugh)
- Network is reliable (ugh)
- Network is semi-reliable but asynchronous
- Network is lossy but messages are not corrupt
- Network failures are transitive
- Failures are independent
- Local data is not corrupt
- Failures are reliably detectable
- Failures are unreliably detectable

Some distributed system design goals

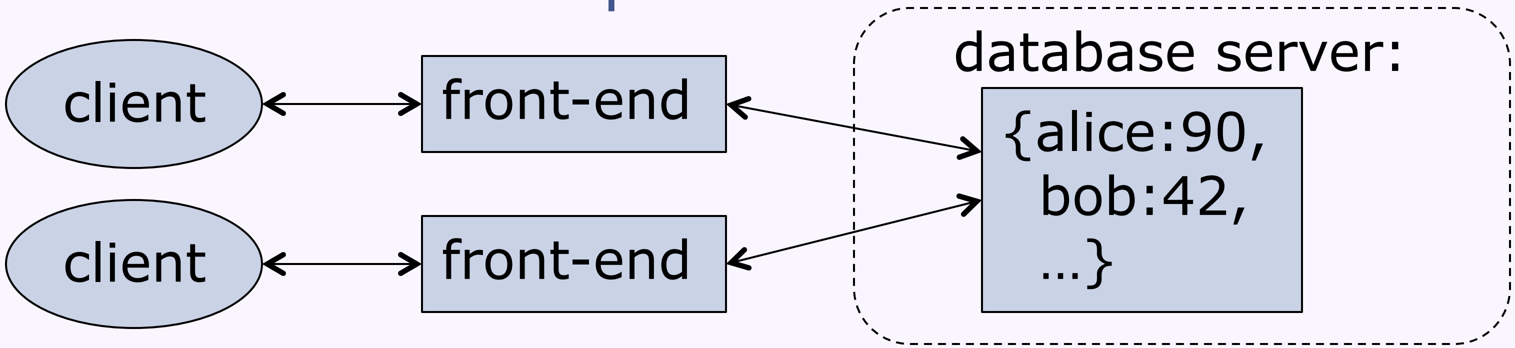
- The end-to-end principle
  - When possible, implement functionality at the end nodes (rather than the middle nodes) of a distributed system
- The robustness principle
  - Be strict in what you send, but be liberal in what you accept from others
    - Protocols
    - Failure behaviors
- Benefit from incremental changes
- Be redundant
  - Data replication
  - Checks for correctness

Today

- Introduction to distributed systems
  - Motivation: reliability and scalability
  - Failure models
- Techniques for reliability and scalability
  - Replication
  - Partitioning

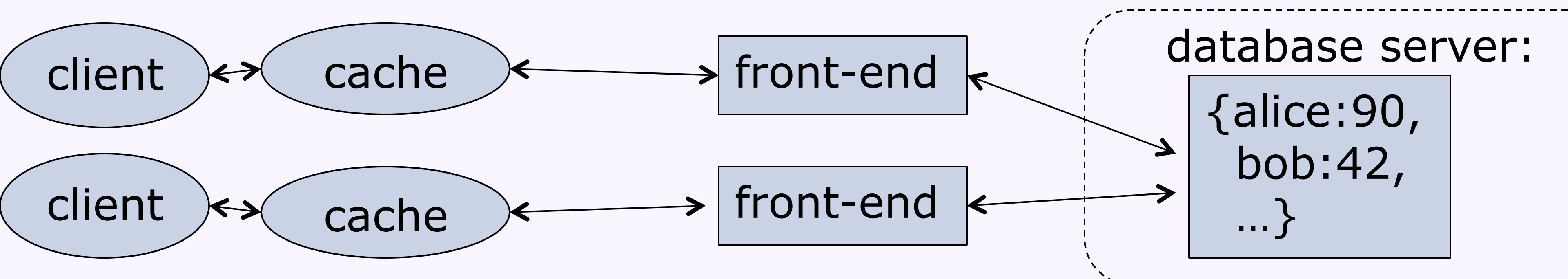
Replication for scalability: Client-side caching

- Architecture before replication:



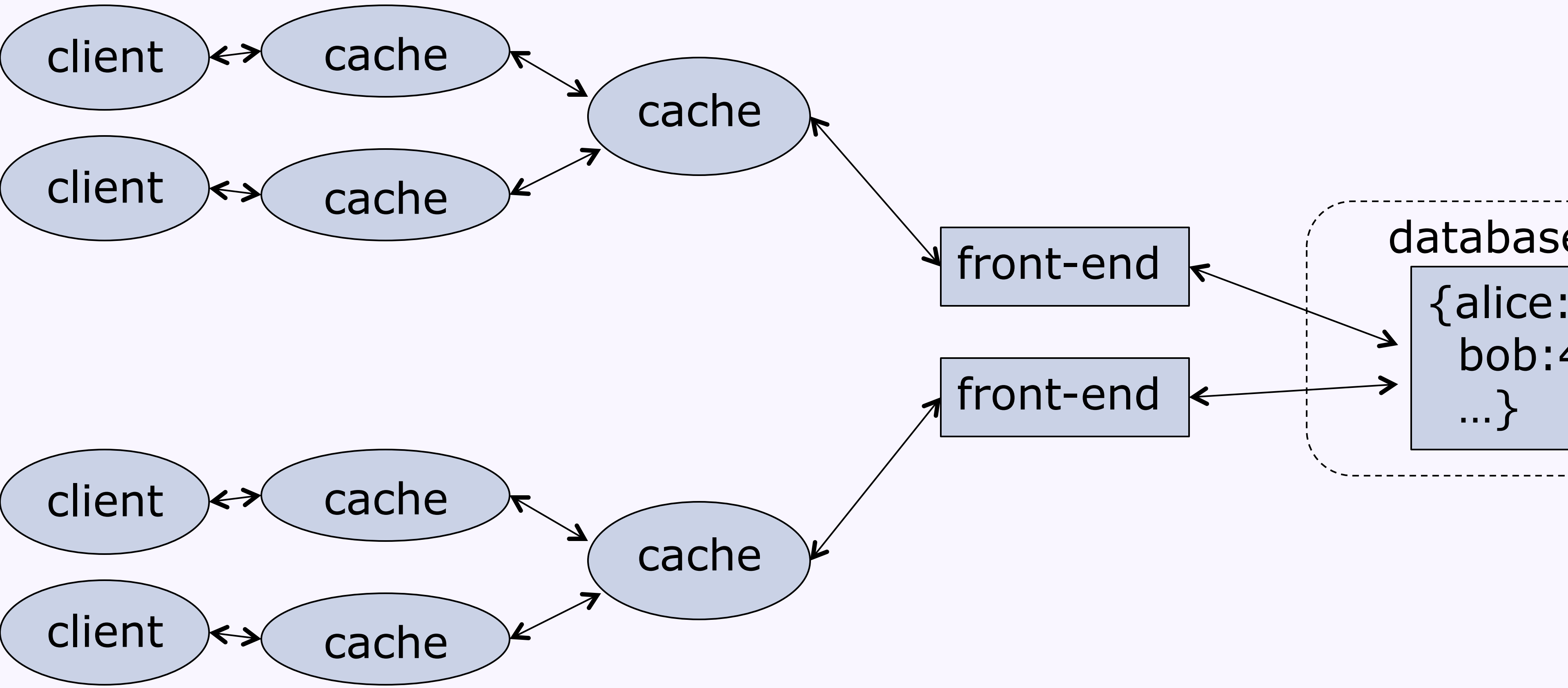
  - Problem: Server throughput is too low

- Solution: Cache responses at (or near) the client
  - Cache can respond to repeated read requests



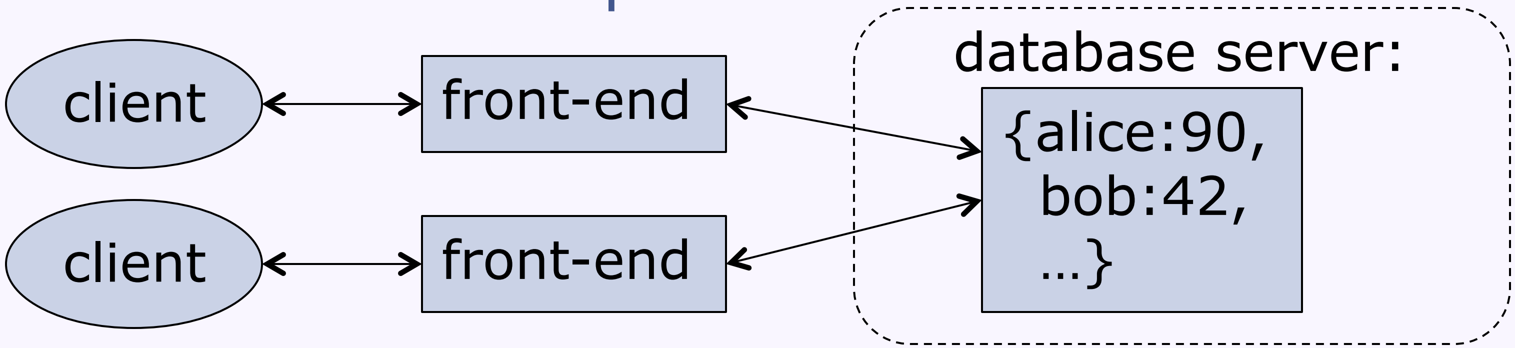
Replication for scalability: Client-side caching

- Hierarchical client-side caches:



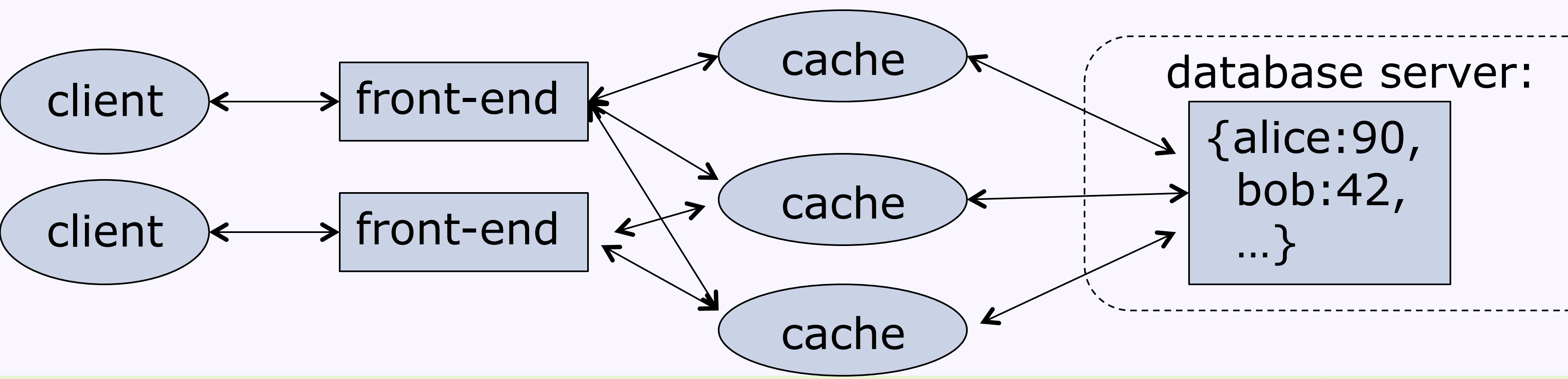
Replication for scalability: Server-side caching

- Architecture before replication:



  - Problem: Database server throughput is too low

- Solution: Cache responses on multiple servers
  - Cache can respond to repeated read requests



Cache invalidation

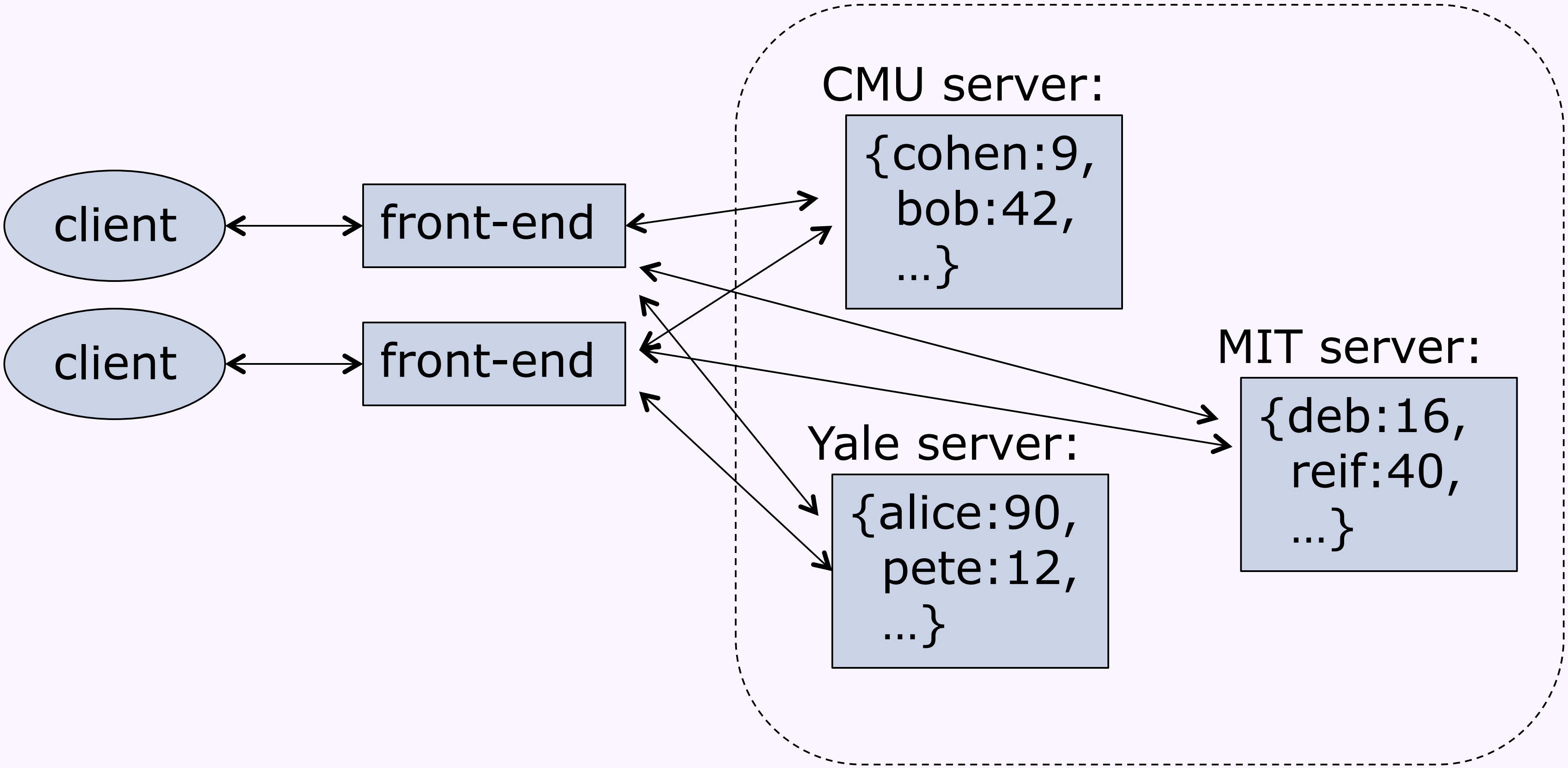
- Time-based invalidation (a.k.a. expiration)
  - Read-any, write-one
  - Old cache entries automatically discarded
  - No expiration date needed for read-only data
- Update-based invalidation
  - Read-any, write-all
  - DB server broadcasts invalidation message to all caches when the DB is updated
- What are the advantages and disadvantages of each approach?

Cache replacement policies

- Problem: caches have finite size
- Common* replacement policies
  - Optimal (Belady's) policy
    - Discard item not needed for longest time in future
  - Least Recently Used (LRU)
    - Track time of previous access, discard item accessed least recently
  - Least Frequently Used (LFU)
    - Count # times item is accessed, discard item accessed least frequently
  - Random
    - Discard a random item from the cache

Partitioning for scalability

- Partition data based on some property, put each partition on a different server



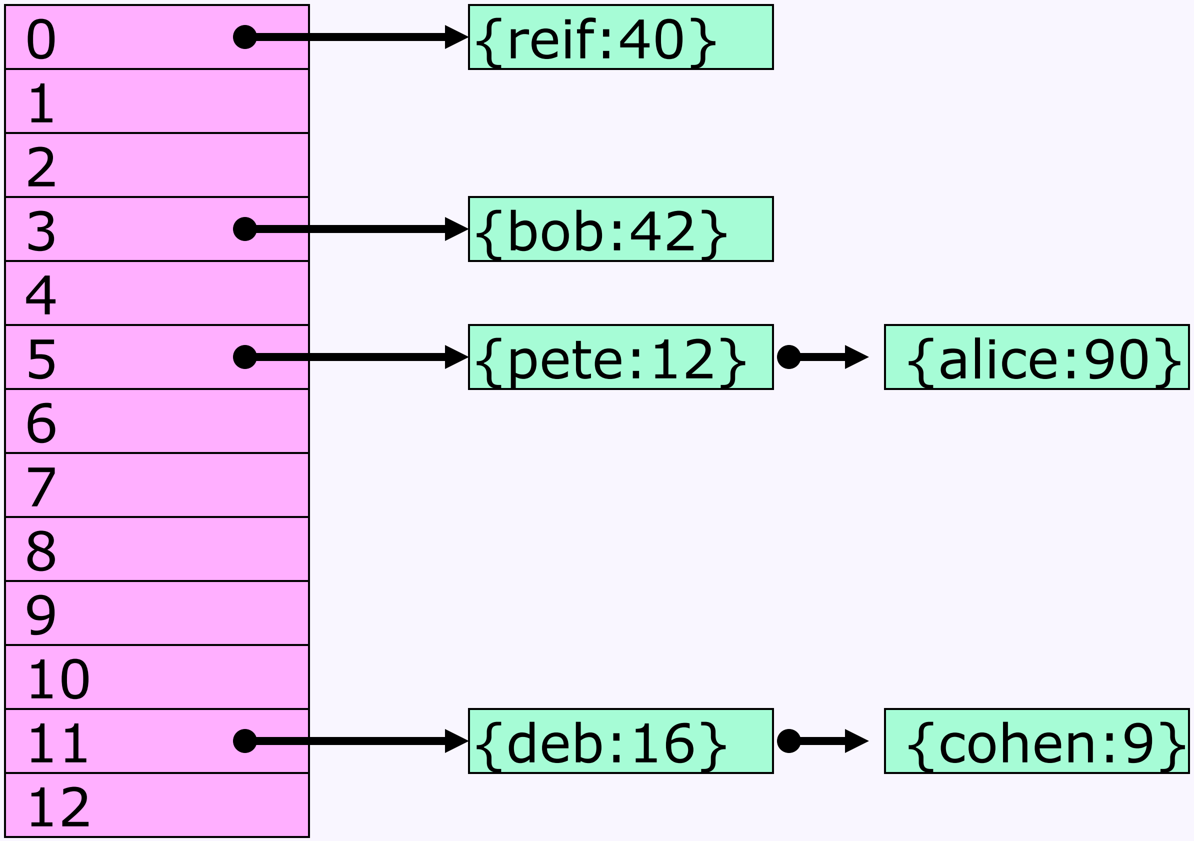
Horizontal partitioning

- a.k.a. "sharding"
- A table of data:

username	school	value
cohen	CMU	9
bob	CMU	42
alice	Yale	90
pete	Yale	12
deb	MIT	16
reif	MIT	40

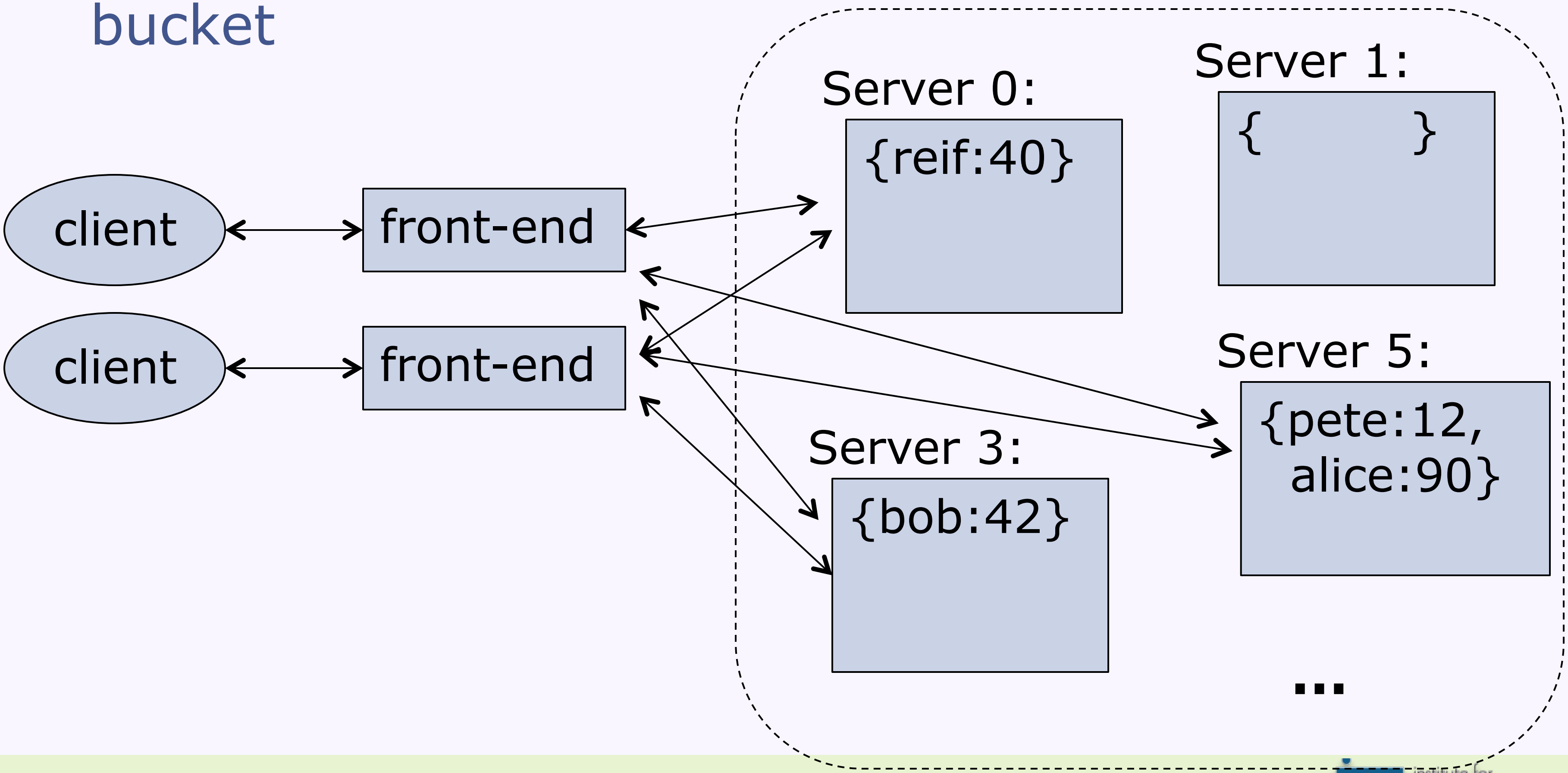
Recall: Basic hash tables

- For `n`-size hash table, put each item `x` in the bucket: `X.hashCode() % n`



Partitioning with a distributed hash table

- Each server stores data for one bucket
- To store or retrieve an item, front-end server hashes the key, contacts the server storing that bucket



Consistent hashing

- Goal: Benefit from incremental changes
  - Resizing the hash table (i.e., adding or removing a server) should not require moving many objects
- E.g., Interpret the range of hash codes as a ring
  - Each bucket stores data for a range of the ring
    - Assign each bucket an ID in the range of hash codes
    - To store item `x` don't compute `x.hashCode() % n`. Instead, place `x` in bucket with the same ID as or next higher ID than `x.hashCode()`

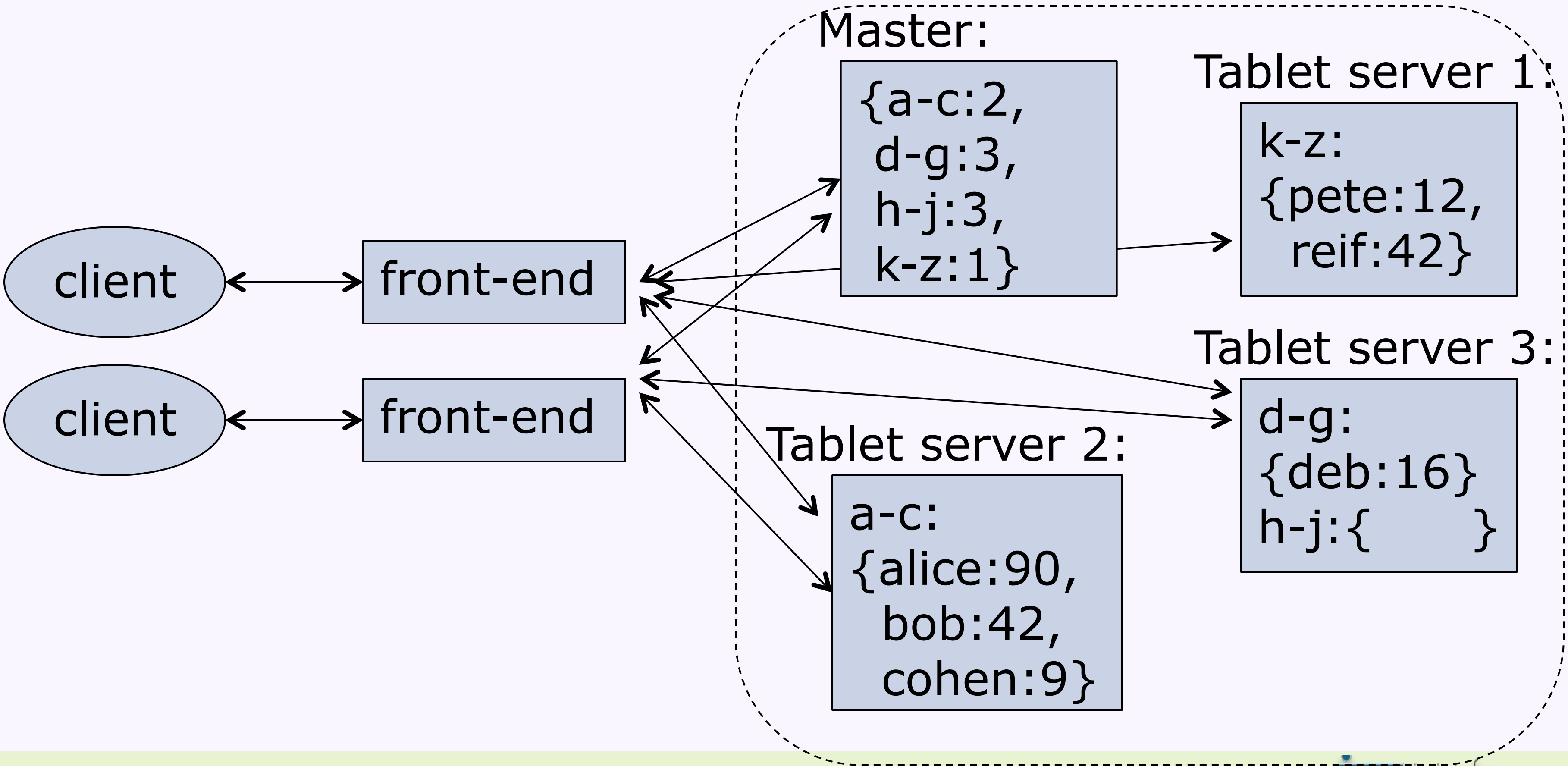
Problems with hash-based partitioning

- Front-ends need to determine server for each bucket
  - Each front-end stores look-up table?
  - Master server storing look-up table?
  - Routing-based approaches?
- Places related content on different servers
  - Consider *range* queries:

```
SELECT * FROM users WHERE lastname STARTSWITH 'G'
```

Master/tablet-based systems

- Dynamically allocate range-based partitions
  - Master server maintains tablet-to-server assignments
  - Tablet servers store actual data
  - Front-ends cache tablet-to-server assignments



Combining approaches

- Many of these approaches are *orthogonal*
- E.g., For master/tablet systems:
  - Masters are often partitioned and replicated
  - Tablets are replicated
  - Meta-data frequently cached
  - Whole master/tablet system can be replicated

Next time

- More distributed systems, map-reduce