Principles of Software Construction: Objects, Design, and Concurrency

Distributed System Design, Part 2. MapReduce

Spring 2014

Charlie Garrod Christian Kästner

School of Computer Science
institute for SOFTWARE RESEARCH

Administrivia

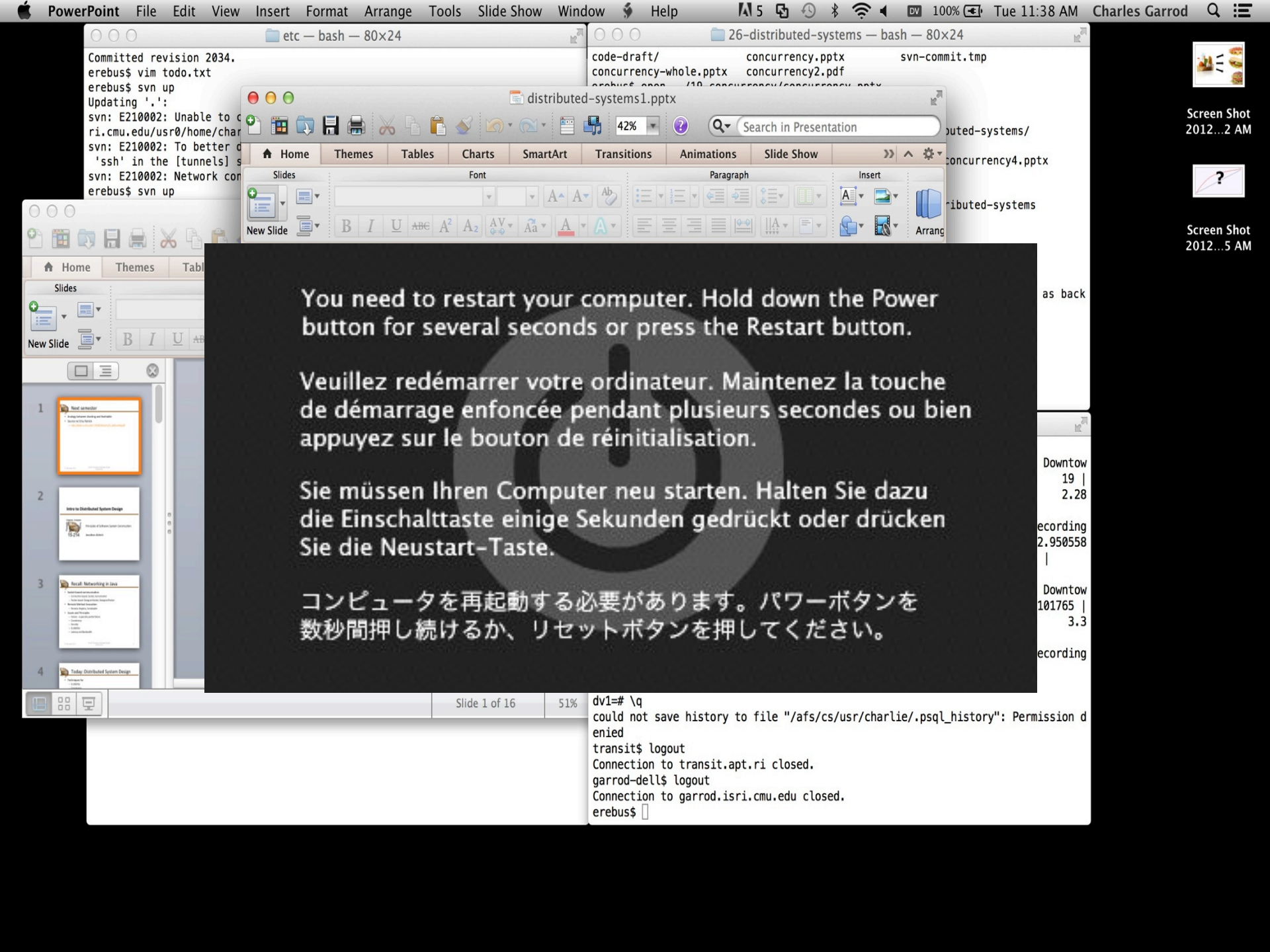
- Homework 5c due tonight
- Homework 6 coming tomorrow

Road map from last time…

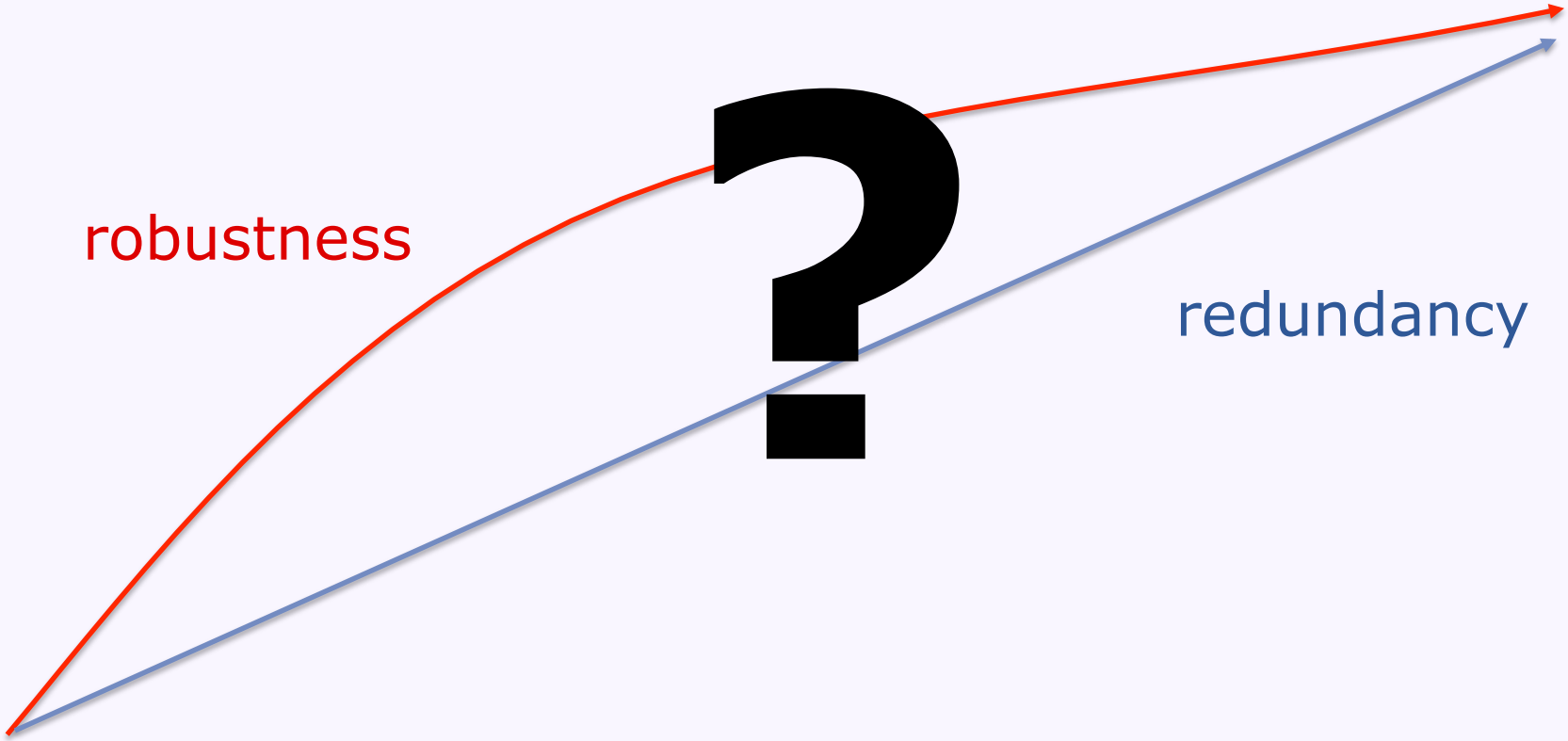
- Application-level communication protocols
- Frameworks for simple distributed computation
  - Remote Procedure Call (RPC)
  - Java Remote Method Invocation (RMI)
- Common patterns of distributed system design
- Complex computational frameworks
  - e.g., distributed map-reduce

Today: Distributed system design, part 2

- Introduction to distributed systems
  - Motivation: reliability and scalability
  - Replication for reliability
  - Partitioning for scalability
- MapReduce: A robust, scalable framework for distributed computation…
  - …on replicated, partitioned data


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 '.':
svn: E210002: Unable to c
ri.cmu.edu/usr0/home/char
svn: E210002: To better d
'ssh' in the [tunnels] s
svn: E210002: Network con
erebus$ svn up
26-distributed-systems — bash — 80×24
code-draft/ concurrency.pptx svn-commit.tmp
concurrency-whole.pptx concurrency2.pdf
distributed-systems1.pptx
42%
Search in Presentation
Home Themes Tables Charts SmartArt Transitions Animations Slide Show
Slides Font Paragraph Insert
New Slide
Arrang
Home Themes
Slides
New Slide
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。
Slide 1 of 16
51%
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

Aside: The robustness vs. redundancy curve

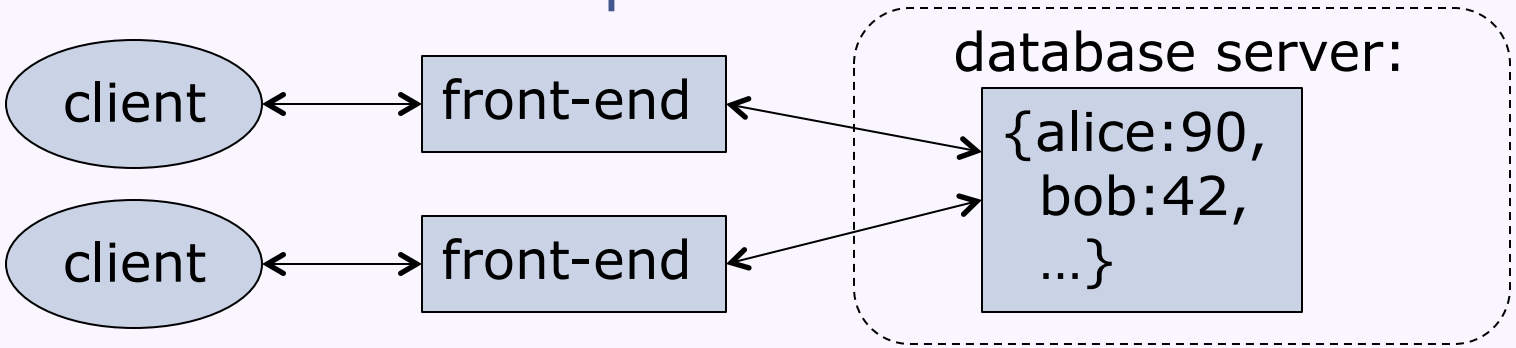


Metrics of success

- 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

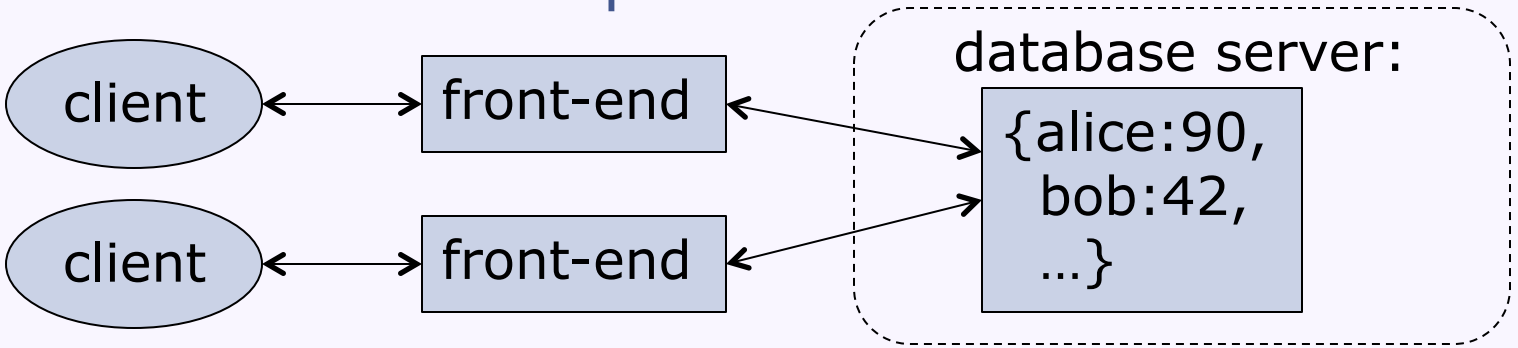
- Architecture before replication:



  - Problem: Database server might fail

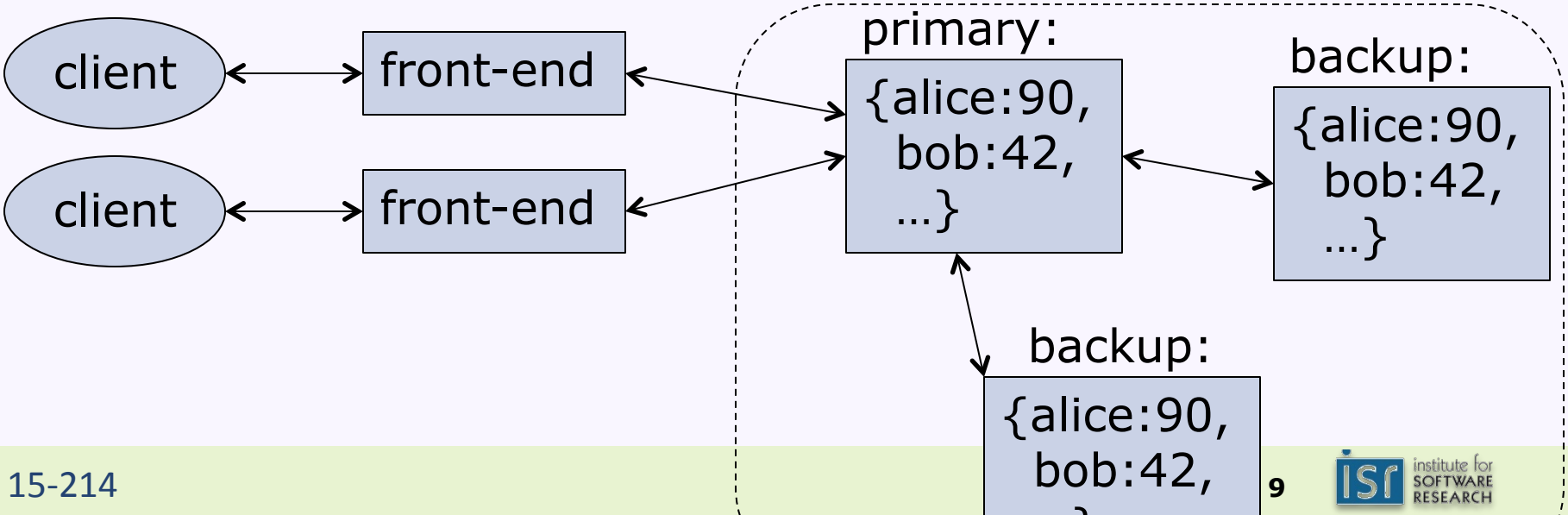
A case study: Passive primary-backup replication

- Architecture before replication:



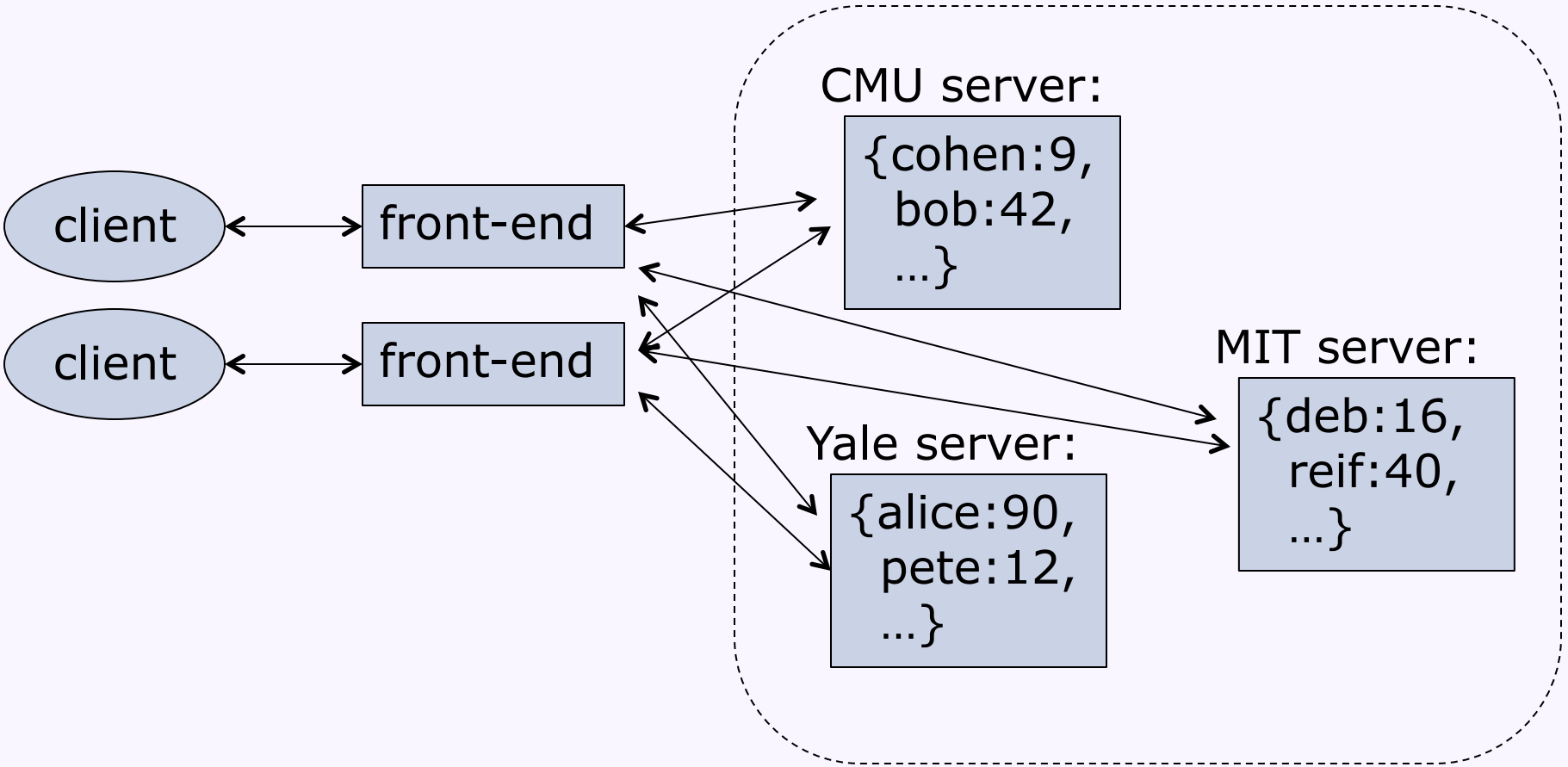
  - Problem: Database server might fail

- Solution: Replicate data onto multiple servers



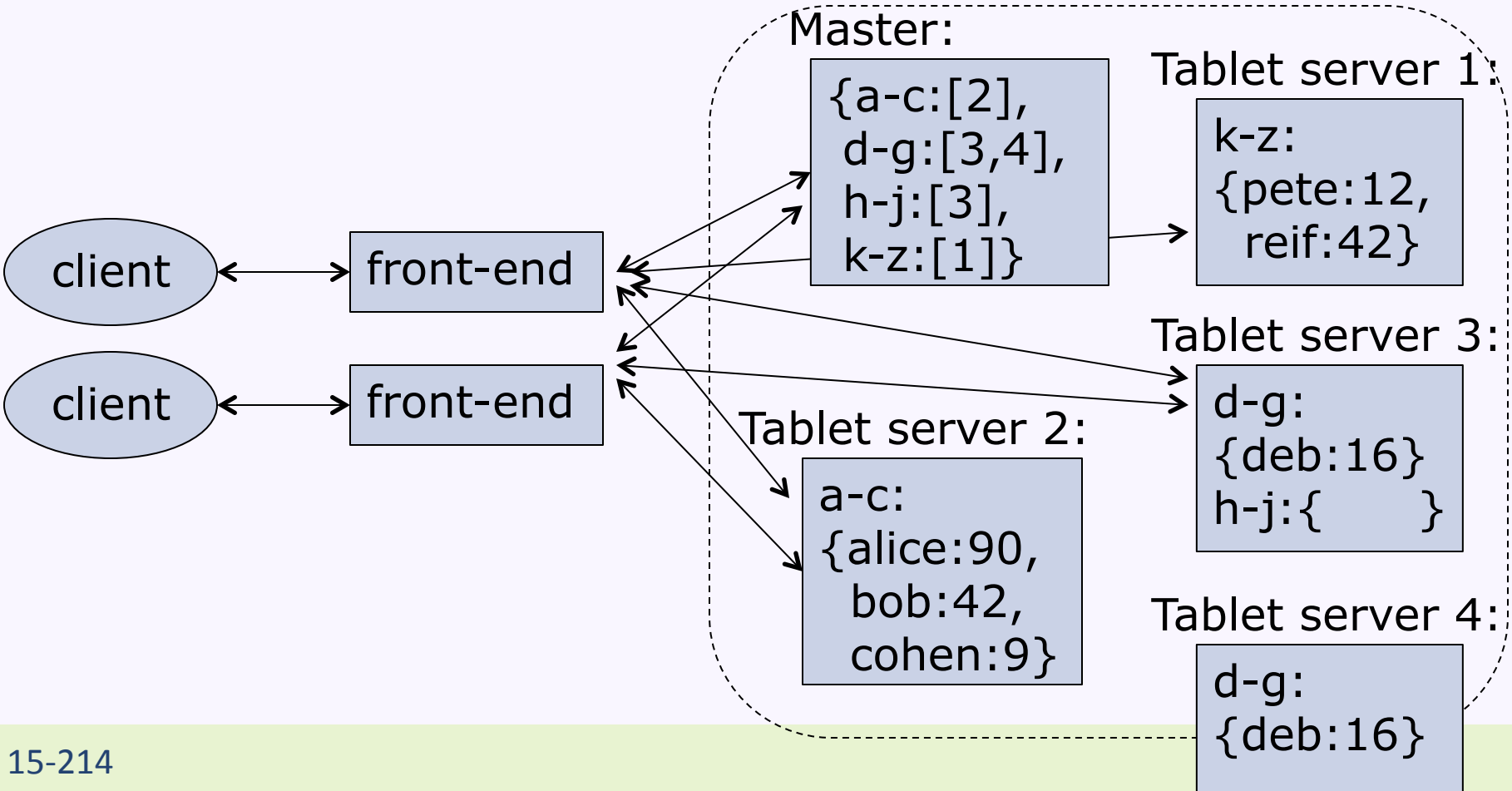
Partitioning for scalability

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



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

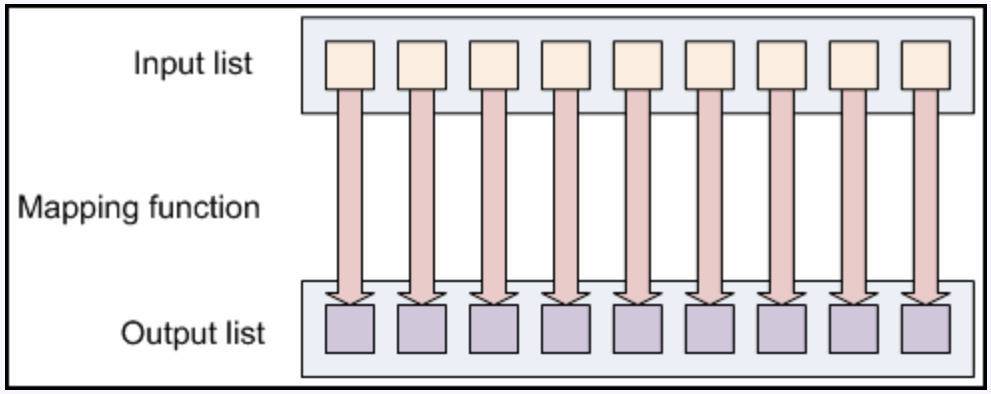


Today: Distributed system design, part 2

- Introduction to distributed systems
  - Motivation: reliability and scalability
  - Replication for reliability
  - Partitioning for scalability
- MapReduce: A robust, scalable framework for distributed computation…
  - …on replicated, partitioned data

Map from a functional perspective

- `map(f, x[0…n-1])`
  - Apply the function `f` to each element of list `x`



map/reduce images src: Apache Hadoop tutorials

- E.g., in Python:

```
def square(x): return x*x
```

`map(square, [1, 2, 3, 4])` would return `[1, 4, 9, 16]`

- Parallel map implementation is trivial
  - What is the work? What is the depth?

Reduce from a functional perspective

- `reduce(f, x[0…n-1])`
  - Repeatedly apply binary function `f` to pairs of items in `x`, replacing the pair of items with the result until only one item remains
  - One sequential Python implementation:

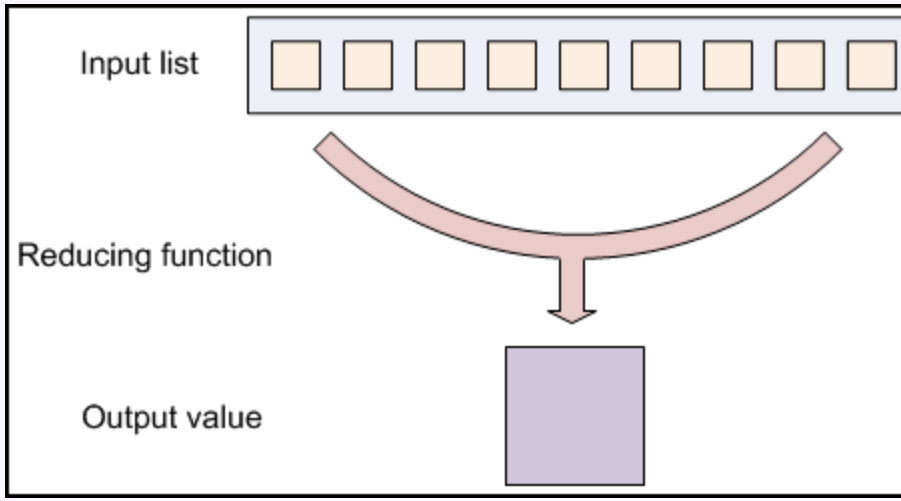
```
def reduce(f, x):
    if len(x) == 1: return x[0]
    return reduce(f, [f(x[0],x[1])] + x[2:])
```

  - e.g., in Python:

```
def add(x,y): return x+y
reduce(add, [1,2,3,4])
```

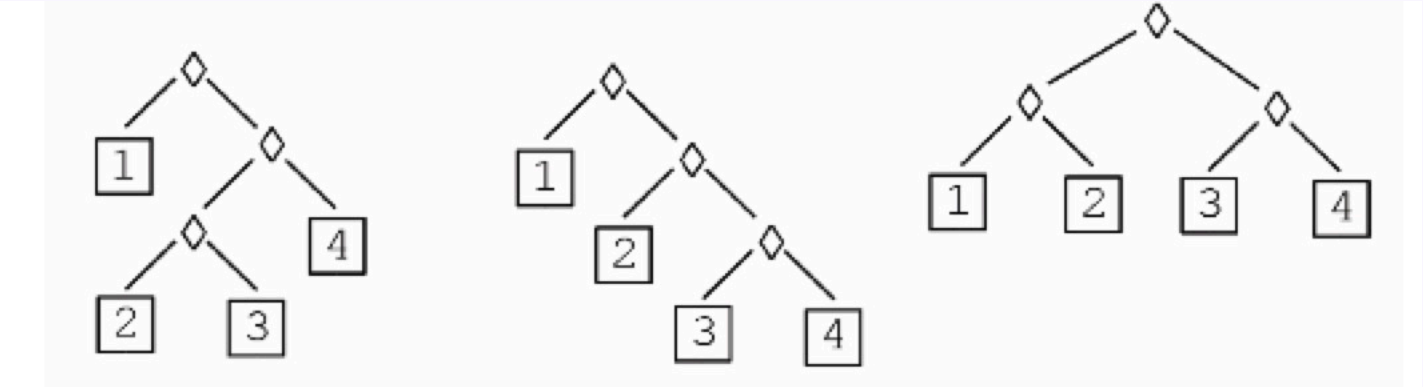
would return 10 as

```
reduce(add, [1,2,3,4])
reduce(add, [3,3,4])
reduce(add, [6,4])
reduce(add, [10]) -> 10
```



Reduce with an associative binary function

- If the function `f` is associative, the order `f` is applied does not affect the result



1 + ((2+3) + 4) 1 + (2 + (3+4)) (1+2) + (3+4)

- Parallel reduce implementation is also easy
  - What is the work? What is the depth?

Distributed MapReduce

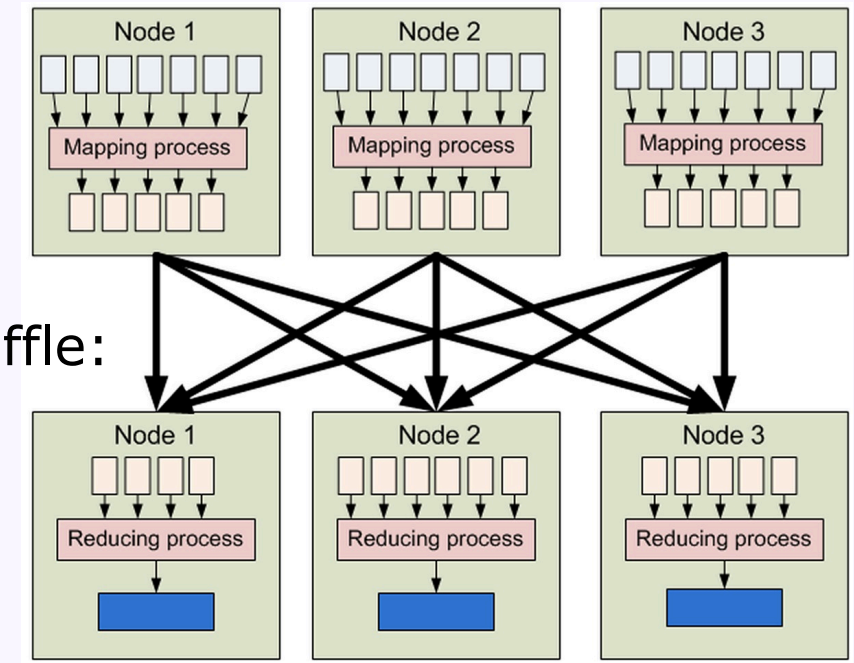
- The distributed MapReduce idea is similar to (but not the same as!):

```
reduce(f2, map(f1, x))
```

- Key idea: a "data-centric" architecture
  - Send function `f1` directly to the data
    - Execute it concurrently
  - Then merge results with reduce
    - Also concurrently
- Programmer can focus on the data processing rather than the challenges of distributed systems

MapReduce with key/value pairs (Google style)

- Master
  - Assign tasks to workers
  - Ping workers to test for failures
- Map workers
  - Map for each key/value pair
  - Emit intermediate key/value pairs
- Reduce workers
  - Sort data by intermediate key and aggregate by key
  - Reduce for each key



MapReduce with key/value pairs (Google style)

- E.g., for each word on the Web, count the number of times that word occurs
  - For Map: `key1` is a document name, `value` is the contents of that document
  - For Reduce: `key2` is a word, `values` is a list of the number of counts of that word

```
f1(String key1, String value):
  for each word w in value:
    EmitIntermediate(w, 1);
```

```
f2(String key2, Iterator values):
  int result = 0;
  for each v in values:
    result += v;
  Emit(key2, result);
```

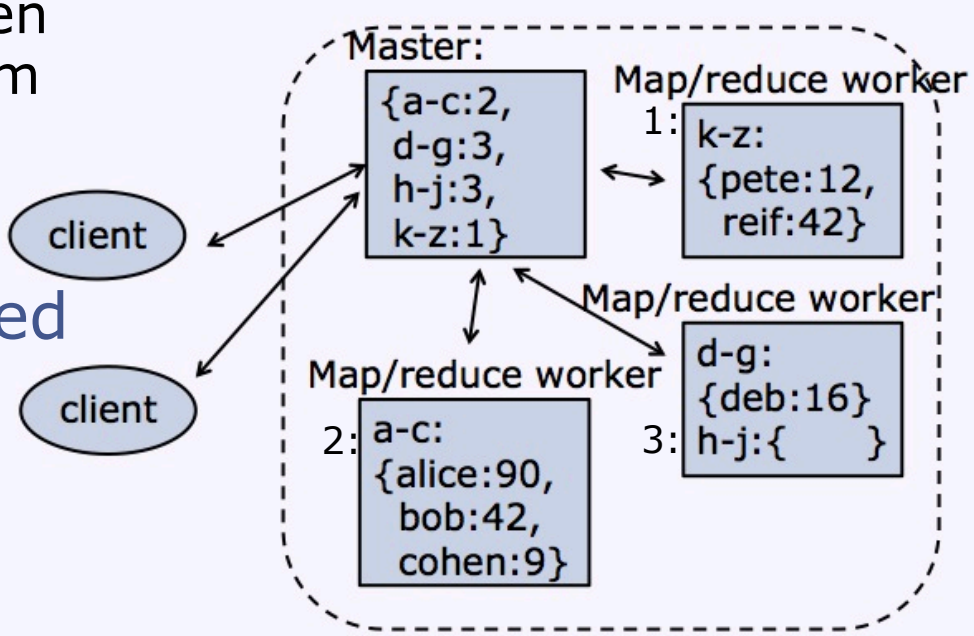
Map: (key1, v1) → (key2, v2)* Reduce: (key2, v2*) → (key3, v3)*

MapReduce: (key1, v1)* → (key3, v3)*

MapReduce: (docName, docText)* → (word, wordCount)*

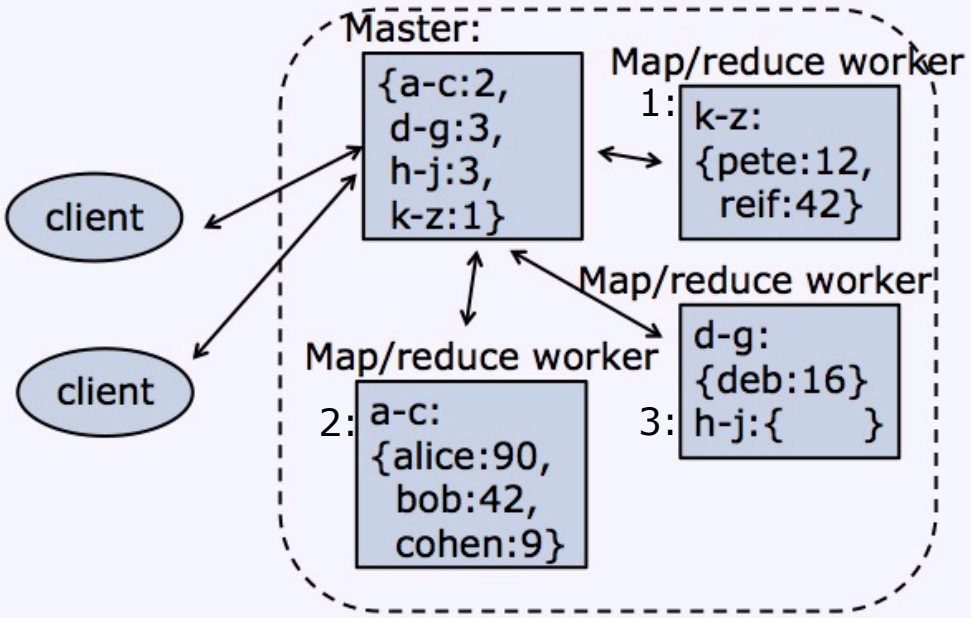
MapReduce architectural details

- Usually integrated with a distributed storage system
  - Map worker executes function on its share of the data
- Map output usually written to worker's local disk
  - Shuffle: reduce worker often pulls intermediate data from map worker's local disk
- Reduce output usually written back to distributed storage system



Handling server failures with MapReduce

- Map worker failure:
  - Re-map using replica of the storage system data
- Reduce worker failure:
  - New reduce worker can pull intermediate data from map worker's local disk, re-reduce
- Master failure:
  - Options:
    - Restart system using new master
    - Replicate master
    - …



The beauty of MapReduce

- Low communication costs (usually)
  - The shuffle (between map and reduce) is expensive
- MapReduce can be iterated
  - Input to MapReduce: key/value pairs in the distributed storage system
  - Output from MapReduce: key/value pairs in the distributed storage system

Another MapReduce example

- E.g., for person in a social network graph, output the number of mutual friends they have
  - For Map: `key1` is a person, `value` is the list of her friends
  - For Reduce: `key2` is ???, `values` is a list of ???

```
f1(String key1, String value):
```

```
f2(String key2, Iterator values):
```

MapReduce: (person, friends)* → (pair of people, count of mutual friends)*

Another MapReduce example

- E.g., for person in a social network graph, output the number of mutual friends they have
  - For Map: `key1` is a person, `value` is the list of her friends
  - For Reduce: `key2` is a pair of people, `values` is a list of 1s, for each mutual friend that pair has

```
f1(String key1, String value):
  for each pair of friends
       in value:
    EmitIntermediate(pair, 1);
```

```
f2(String key2, Iterator values):
  int result = 0;
  for each v in values:
    result += v;
  Emit(key2, result);
```

MapReduce: (person, friends)* → (pair of people, count of mutual friends)*

And another MapReduce example

- E.g., for each page on the Web, create a list of the pages that link to it
  - For Map: `key1` is a document name, `value` is the contents of that document
  - For Reduce: `key2` is ???, `values` is a list of ???

```
f1(String key1, String value):
```

```
f2(String key2, Iterator values):
```

MapReduce: (docName, docText)* → (docName, list of incoming links)*

Thursday

- More distributed systems..