Section 1:
Intro to DSLabs
Labs 0 + 1

CS 5414 Spring 2024

Part of the information is adapted from UW CSE 452 Winter 2023 Section 1 slides
https://docs.google.com/presentation/d/1VrxF_qj5i0hQzV1mL02_533FhbDZ3BLNi2gOwUTuI4A/edit#slide=id.p

Plan for today

Overview of CS 5414 and DSLabs projects

Design docs

DSLabs intro

Lab 0 demo

Lab 1 intro

CS 5414

=

Exams (2) + HWs (3) + Projects

50% 15% 34%

CS 5414

=

Exams (2) + HWs (3) + Projects

50% 15% 34%

- Three HWs
- Theory-oriented
- Design distributed algorithms
- Proving their correctness
- Similar in style to HWs in CS 4820 (except that everything becomes distributed)

CS 5414

=

Exams (2) + HWs (3) + Projects

50%

- Midterm and Final
- Questions similar in style to HW problems
- But less difficult due to time constraints

15%

- Three HWs
- Theory-oriented
- Design distributed algorithms
- Proving their correctness
- Similar in style to HWs in CS 4820 (except that everything becomes distributed)

34%

CS 5414

=

Exams (2) + HWs (3) + Projects

50%

- Midterm and Final
- Questions similar in style to HW problems
- But less difficult due to time constraints

15%

- Three HWs
- Theory-oriented
- Design distributed algorithms
- Proving their correctness
- Similar in style to HWs in CS 4820 (except that everything becomes distributed)

34%

- Four labs using DSLabs
- Implement distributed protocols in Java
- Regular tests and model checker tests (search tests) - hard to pass all
- Work in groups of 2
- Substantial coding
- Debugging can become painful
- Design docs

CS 5414

=

Sections focus on this part

Exams (2) + HWs (3) + Projects

50%

- Midterm and Final
- Questions similar in style to HW problems
- But less difficult due to time constraints

15%

- Three HWs
- Theory-oriented
- Design distributed algorithms
- Proving their correctness
- Similar in style to HWs in CS 4820 (except that everything becomes distributed)

34%

- Four labs using DSLabs
- Implement distributed protocols in Java
- Regular tests and model checker tests (search tests) - hard to pass all
- Work in groups of 2
- Substantial coding
- Debugging can become painful
- Design docs


WARNING
CONTAINS SCENES THAT
SOME VIEWERS MAY
FIND DISTURBING

Time for labs [in **hours**]
(fairly rough) statistics from UW CSE 452 Spring 2021

Lab	Mean	Std. dev.	Median
Lab 1	7.438	3.336	7.0
Lab 2	32.94	15.98	30
Lab 3	53.57	30.48	49
Lab 4*	37.24	21.78	30

*= some people only did up to part 4.2

Time for labs [in **hours**]
(fairly rough) statistics from UW CSE 452 Spring 2021

Lab	Mean	Std. dev.	Median
Lab 1	7.438	3.336	7.0
Lab 2	32.94	15.98	30
Lab 3	53.57	30.48	49
Lab 4*	37.24	21.78	30

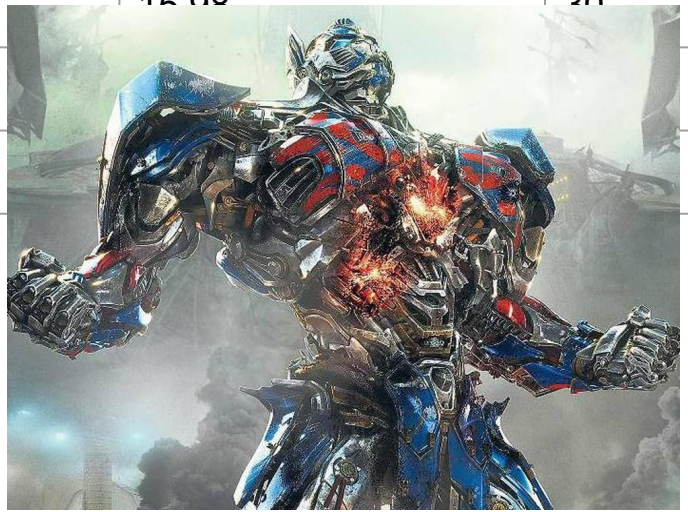
*= some people only did up to part 4.2

Time for labs [in **hours**]
(fairly rough) statistics from UW CSE 452 Spring 2021

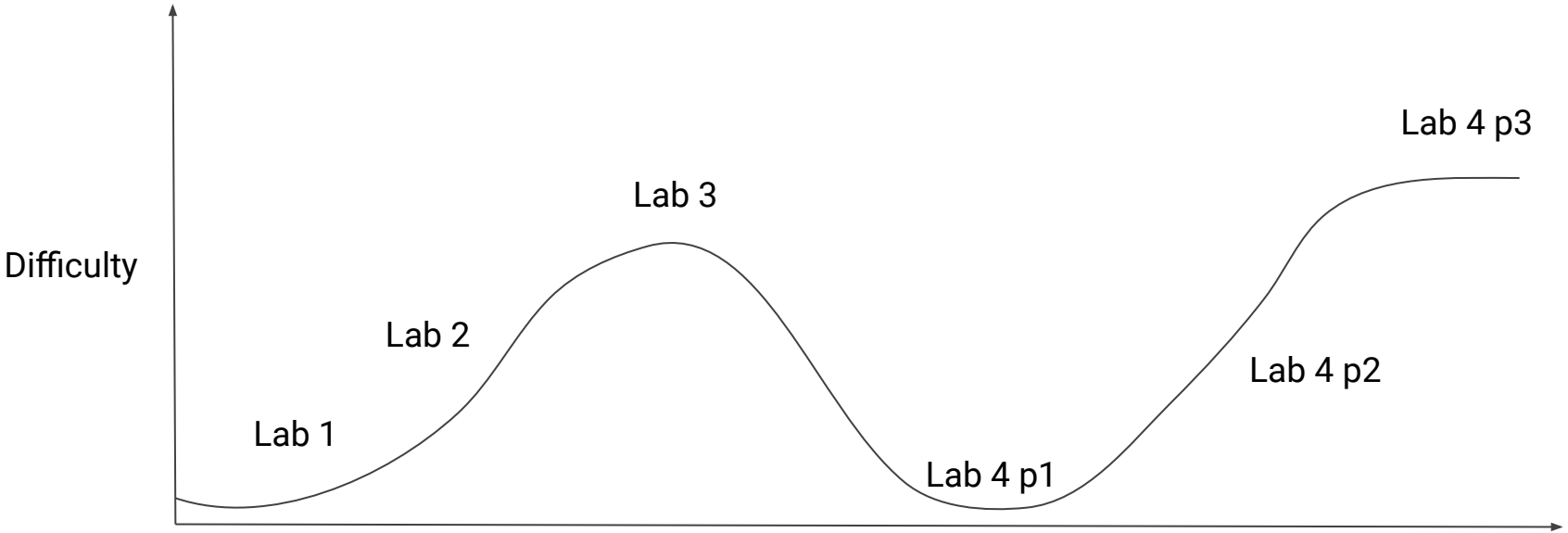
Lab	Mean	Std. dev.	Median
Lab 1	7.438	3.336	7.0
Lab 2	32.94	15.98	30
Lab 3	53.57		
Lab 4*	37.24		

!

*= some people only did up to part 4.2



Difficulty of each lab



Lines of code (including comments) of each assignment solution

https://ellismichael.com/papers/dslabs-eurosys19.pdf

Assignment	LOC
Lab 1: Exactly once RPC	164
Lab 2: Primary-backup	355
Lab 3: Multi-Paxos	647
Lab 4.1 + 4.2: Dynamic sharding	878
Lab 4.3: Transactions	597

Tips

- ***Start early!!!!***
- Ask questions in discussion session/lecture/on Ed/OH
- Labs get **much much much** harder
- Read the spec and reread the spec
- Take advantage of UW section slides.
  https://courses.cs.washington.edu/courses/cse452/23wi/section/
  They contain more hints, diagrams, and suggestions for implementation in case you get stuck and want to alleviate suffering.
- You may find yourself spending time to rewrite your code (to make it cleaner or to trim unnecessary implementation details) -> that's totally ok and expected
- Carefully think through your design ***before writing any code*** (that's why we ask for a **design doc**)
- Think through the design carefully, before starting to code
  - Agree with partner on design, then divide work up; Use pair programming;
  - DON'T divide work up, design independently, then integrate

Project Tools

- Automated Tests
  - All tests are given to you. Your grade is determined by whether you pass those tests.
- Visual debugger
  - You can control and redo all message delivery and node failures - more about it later
- Implemented in Java
  - For all labs, we recommend using **IntelliJ** as IDE.
  - Do not forget to add **Lombok plugin** to your Intellij: https://projectlombok.org/setup/intellij

Plan for today

Overview of CS 5414 and DSLabs projects

Design doc

DSLabs intro

Lab 0 demo

Lab 1 intro

Design doc

- describes a distributed protocol at a **high-level** but with enough detail that a competent programmer could implement it without having to think about any distributed systems aspects of the problem (a big ask!).
- It is ***not*** a goal to describe how your implementation works at a low level. Instead, you should describe conceptually how your system solves the problem it is trying to solve, in such a way that someone else could implement it without looking at your implementation.
- We will follow a highly structured template in this class, given by UW
    - https://courses.cs.washington.edu/courses/cse452/23sp/design-docs.html
- Previously, you may spend 50+ hours on lab3 trying to debug and still don't know the issue
- But, if you have a design that met all the requirements, you may only spend 30 hours.

Plan for today

Overview of CS 5414 and DSLabs projects

Design docs

DSLabs intro

Lab 0 demo

Lab 1 intro

DSLabs intro

High level picture of what you are going to build

Distributed programming in DSLabs framework

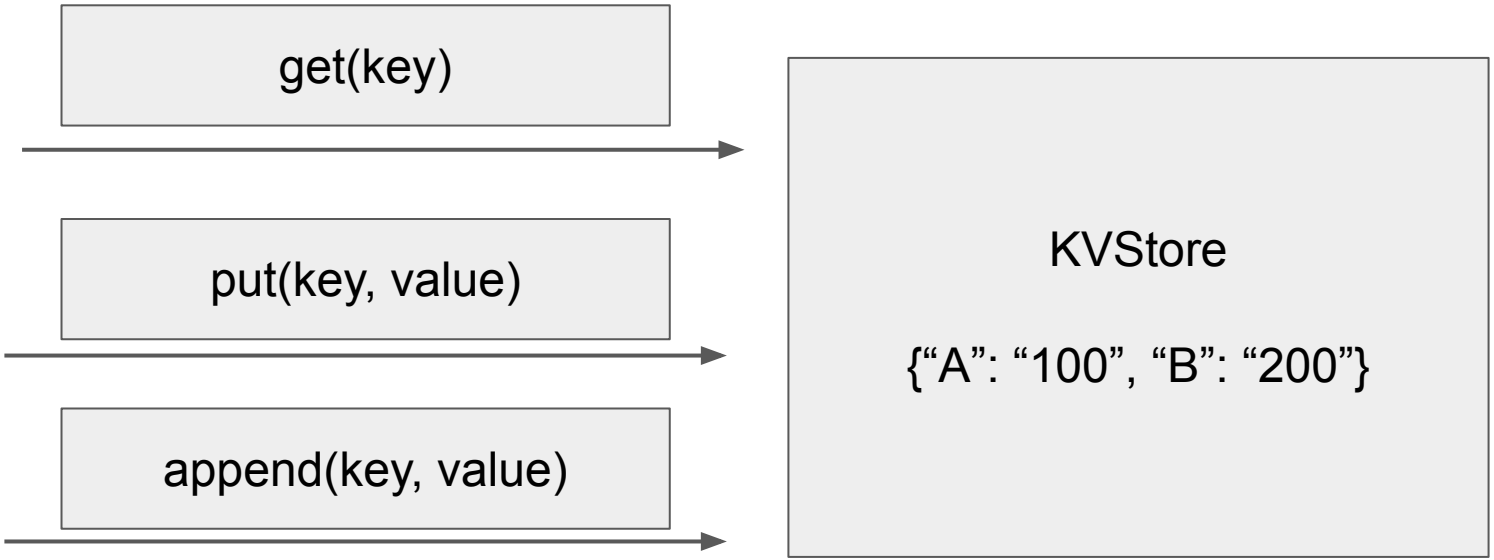
High level picture of what you are going to build

- A replicated, sharded, transactional key-value store

2 **3** **4** **1**

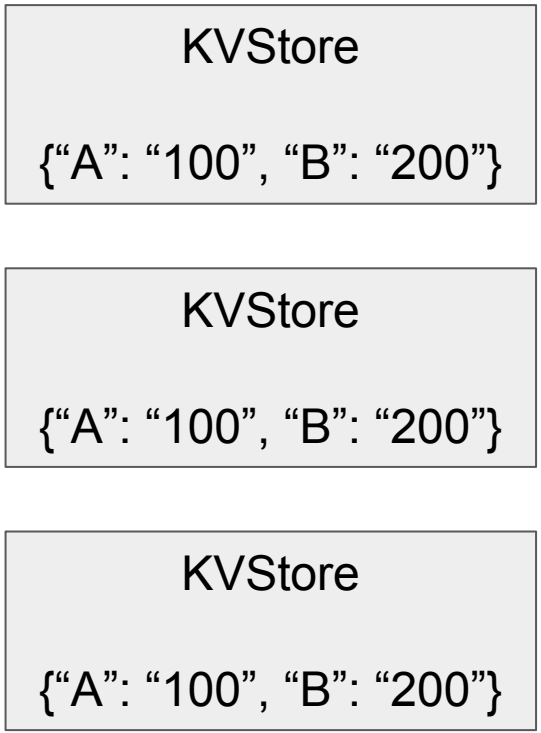
1. Key-value store

A set of (key, value) pairs that supports three operations: get, put, and append



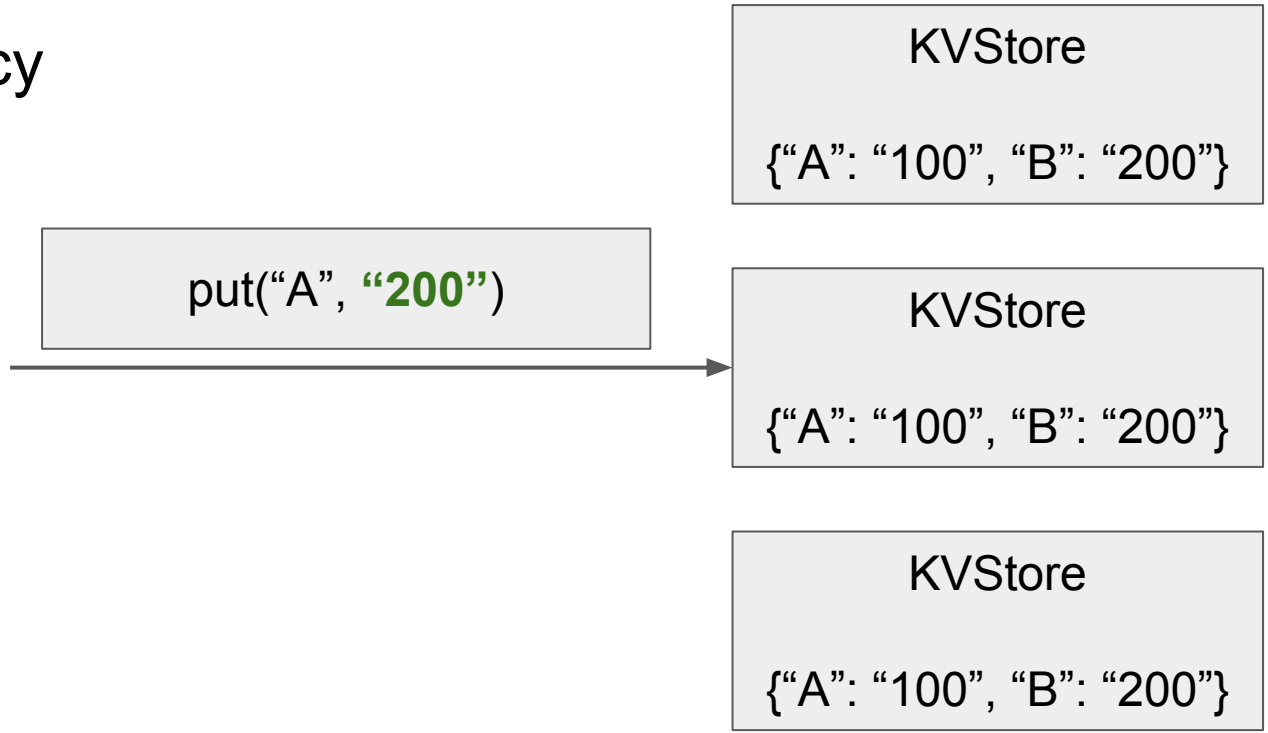
2. **Replicated** Key-value store

- Since machines can **fail**, if a key-value store has only one copy, then it will be unavailable on failures.
- For fault tolerance, we **replicate** multiple copies of a key-value store, so that it will remain **available** even when some machine fails.
- Replicated copies need to remain **consistent** - if we update one copy, other copies also need to be updated accordingly.



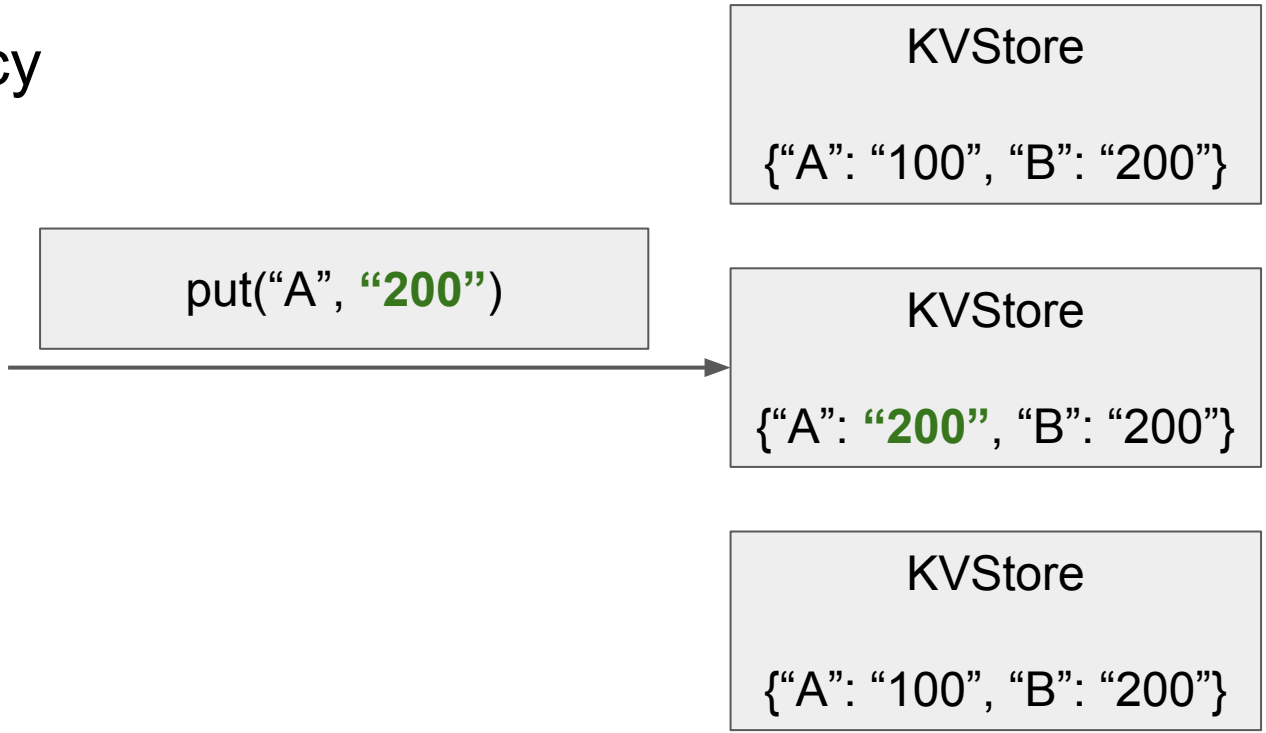
2. **Replicated** Key-value store

Consistency



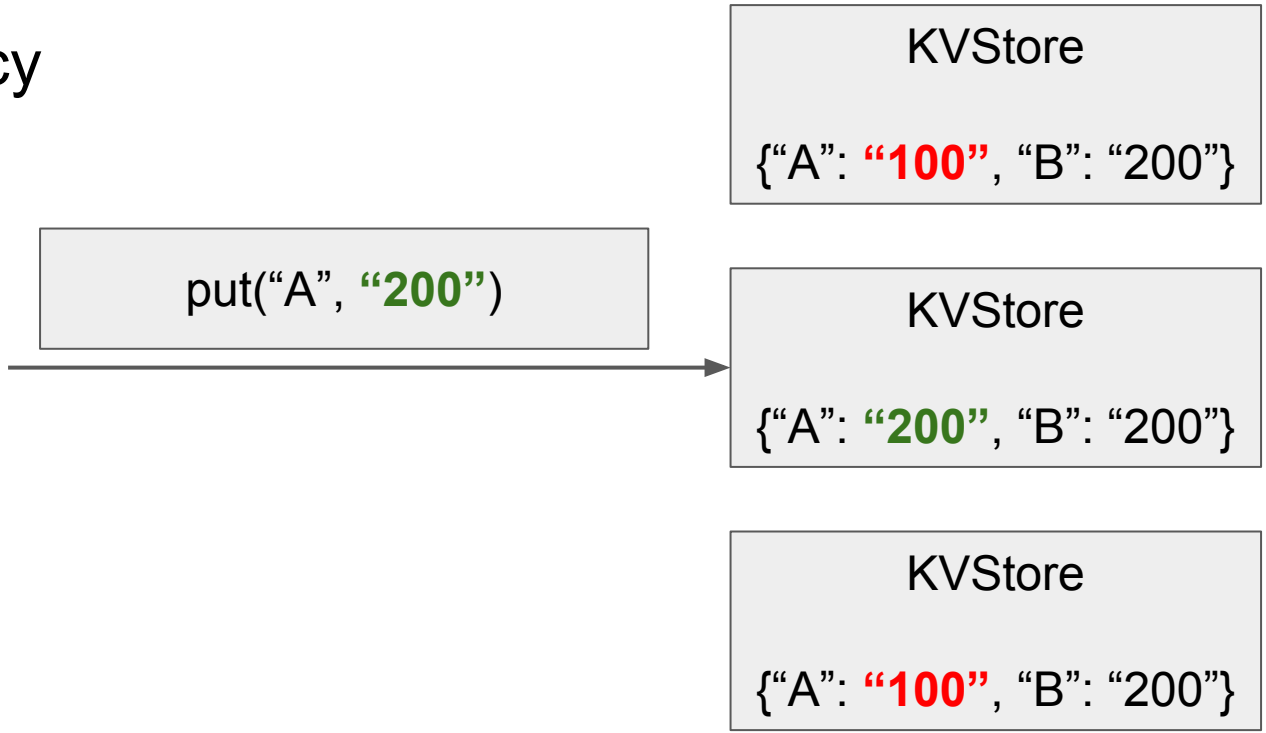
2. **Replicated** Key-value store

Consistency



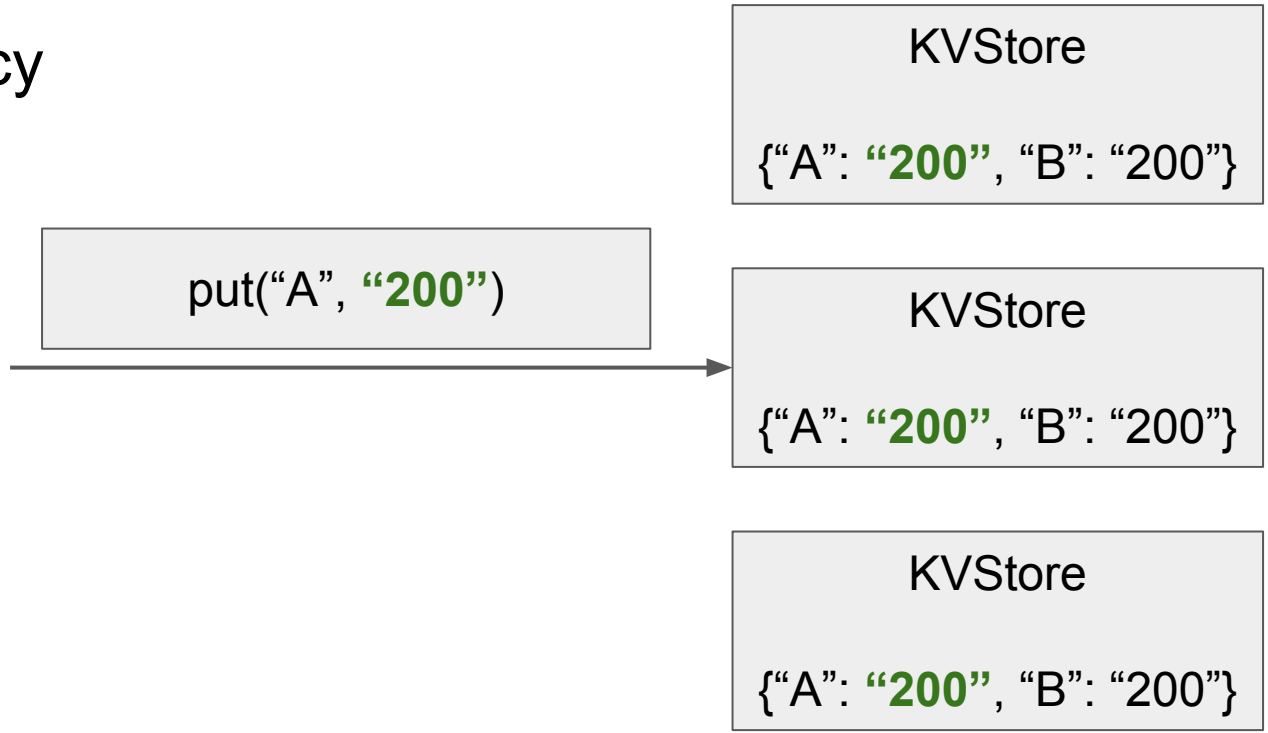
2. **Replicated** Key-value store

Consistency



2. **Replicated** Key-value store

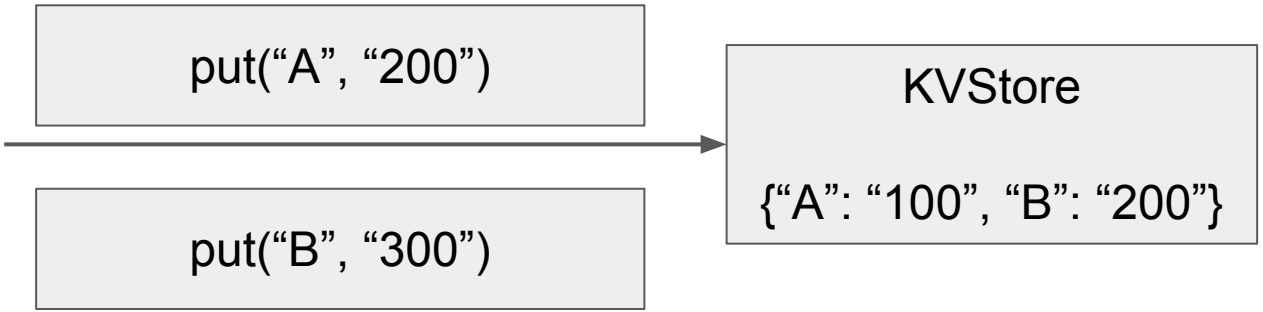
Consistency



3. Sharded, replicated Key-value store

Sharding:

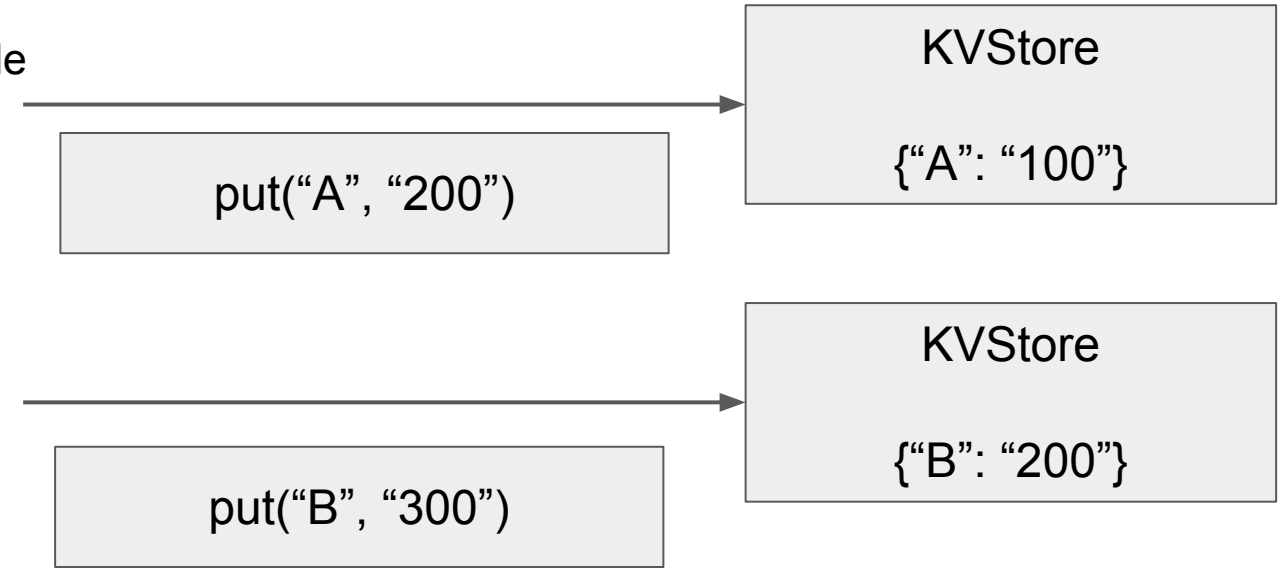
- Divide key space into multiple partitions
- Different machines handle different partitions
- Improves performance



3. Sharded, replicated Key-value store

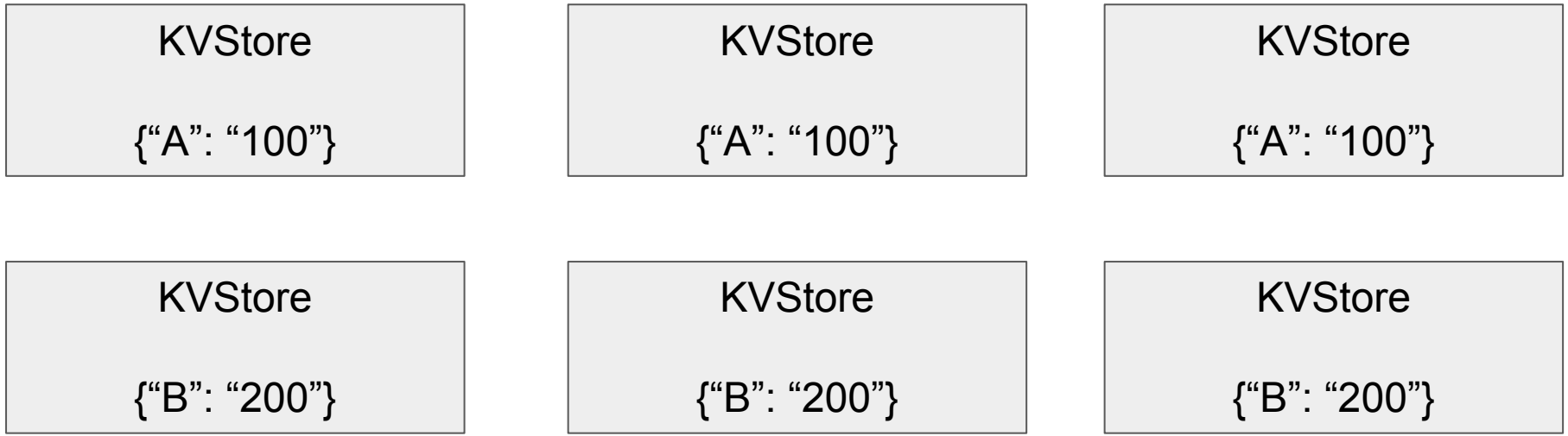
Sharding:

- Divide key space into multiple partitions
- Different machines handle different partitions
- Improves performance



3. Sharded, replicated Key-value store

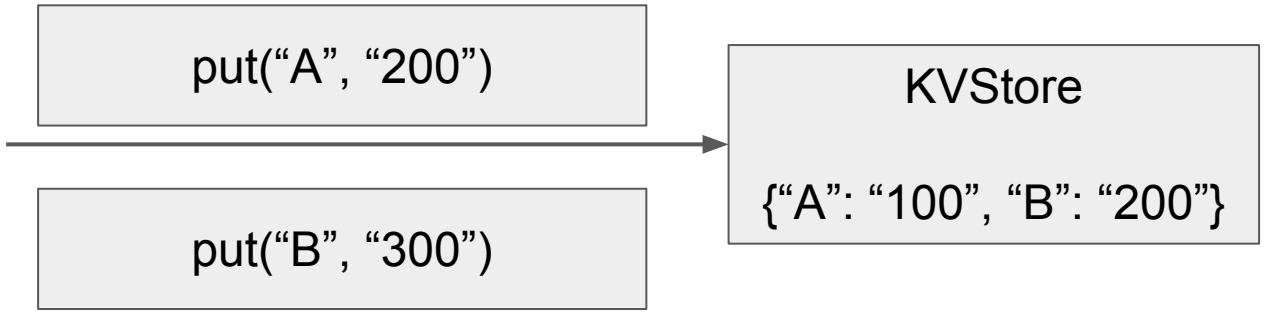
Sharding combined with replication



4. Sharded, replicated, **transactional** Key-value store

Support **transactions**:

- Allow **atomic** multi-key updates

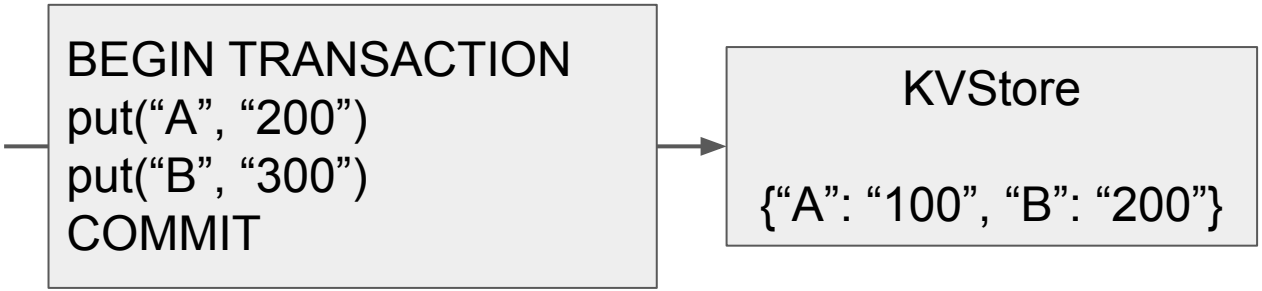


Without transactions, put(“A”, “200”) and put(“B”, “300”) are two **separate** operations. Possible visible states of the KVStore are {“A”: “100”, “B”: “200”}, {“A”: “200”, “B”: “200”}, {“A”: “100”, “B”: “300”}, {“A”: “200”, “B”: “300”}

4. Sharded, replicated, **transactional** Key-value store

Support **transactions**:

- Allow **atomic** multi-key updates



With transactions, put(“A”, “200”) and put(“B”, “300”) are one **atomic** operation. Possible visible states of the KVStore are {“A”: “100”, “B”: “200”}, {“A”: “200”, “B”: “300”}

High level picture of what you are going to build

- A replicated, sharded, transactional key-value store

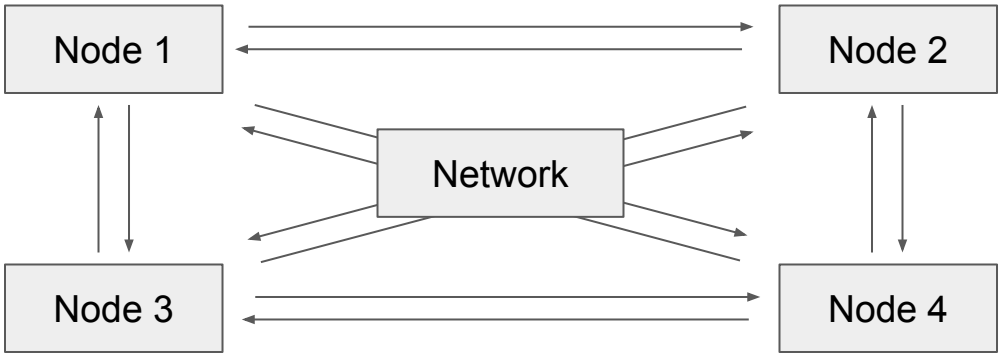
DSLabs intro

High level picture of what you are going to build

Distributed programming in DSLabs framework

Conceptual picture

- Nodes
  - Distributed processes running on different machines.
  - Nodes communicate by **sending messages** over the **network**
  - In the labs, you will program the nodes so that multiple nodes will cooperate to implement some distributed protocol
  - Nodes may **fail** at any point



Conceptual picture

- Network properties
  - **Asynchronous**
    - messages sent may take **unbounded** time to arrive
  - **Unreliable**
    - messages can be **dropped**, **duplicated**, **out of order**
- The protocols you implement must be robust enough to work **despite** these bad properties

Conceptual picture

- Timer
  - You can set a timer on a node to make it do something after a certain amount of time (say, 100ms).
  - Use timers to retry something / resend some message given the unreliability of the network

Conceptual picture

In DSLabs, each node runs a single-threaded event loop that looks like follows:

init() initialization function for node N

while True:

if some **timer T** goes off:

onT() is handler function for timer of type T (timer type T and handler function onT() are defined by you)

run **onT()**

else if some **message M** arrives:

handleM() is handler function for message of type M (message type M and handler function handleM() are defined by you)

run **handleM()**

You can have different kinds of messages/timers by defining different message/timer types

Conceptual picture

- Your job for each lab:
  - Define for each node its
    - init function
    - message types
    - message handlers
    - timer types
    - timer handlers
  - So that nodes cooperate to implement some distributed protocols

The code you write will **only** involve these 5 things, all the rest is handled for you by the DSLabs framework.

Conceptual picture

- Concurrency can cause **non-determinism** in the relative order in which messages are delivered and timers are triggered
- Consider a message M and a timer T
  - If message M is delivered **before** timer T goes off, then function handleM() will run **before** function onT()
  - If message M is delivered **after** timer T goes off, then function handleM() will run **after** function onT()
  - The two possibilities can produce different resulting state!
- Each run of your distributed program is only one of many possible runs!
- Thus, your distributed program may be bug-free on the first 999 runs but buggy on the 1,000th run!
- How can we test your distributed program **thoroughly**?

Conceptual picture

- **Model checking** to the rescue!
- Try every possible run, and if one of them is buggy, report an error!
- This is what DSLabs does
  - DSLabs contains two types of tests - **regular tests (run test)** and **model checker tests (search test)**
  - Regular tests (run test) try **only one** possible run and see if it's correct
  - Model checker tests (search test) tries to explore **all** possible runs and see whether all are correct
- Why need regular tests if we already have model checker?
  - Because model checking is too computationally expensive. In many cases, we cannot exhaustively search all possible states (**state explosion**). This is also true in DSLabs (alas!).

Framework Javadoc

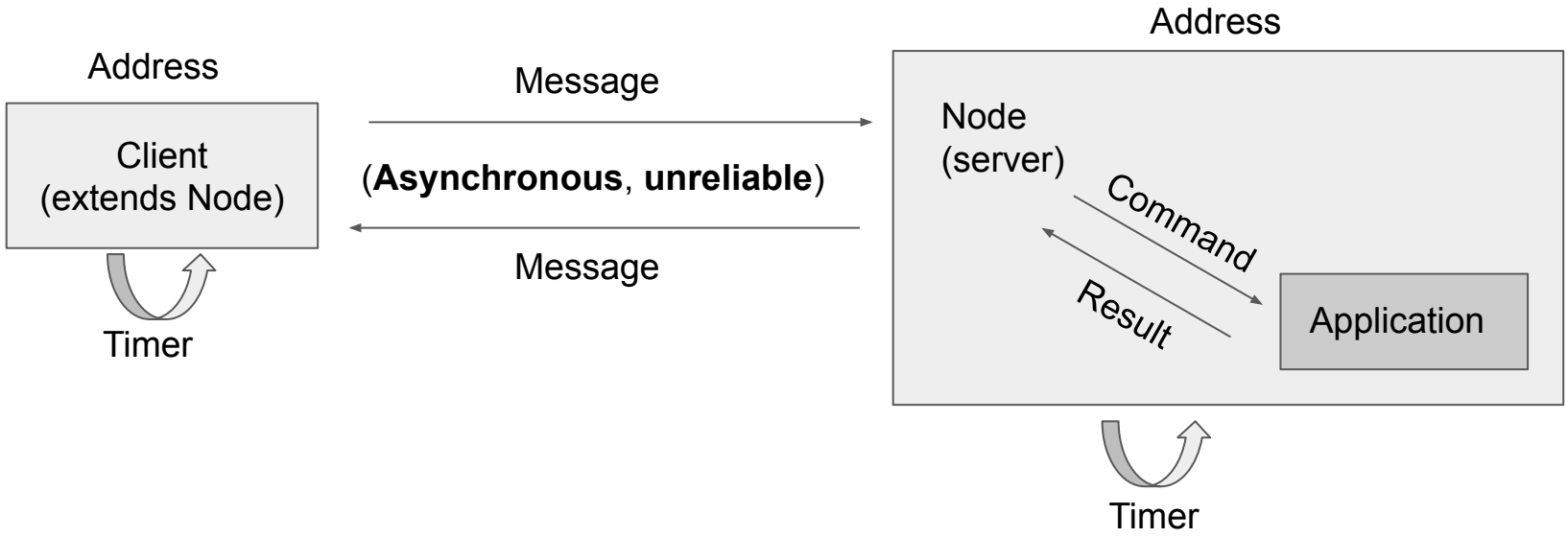
https://ellismichael.com/dslabs/javadoc/dslabs/framework/package-summary.html

All Classes and Interfaces | **Interfaces** | **Classes**

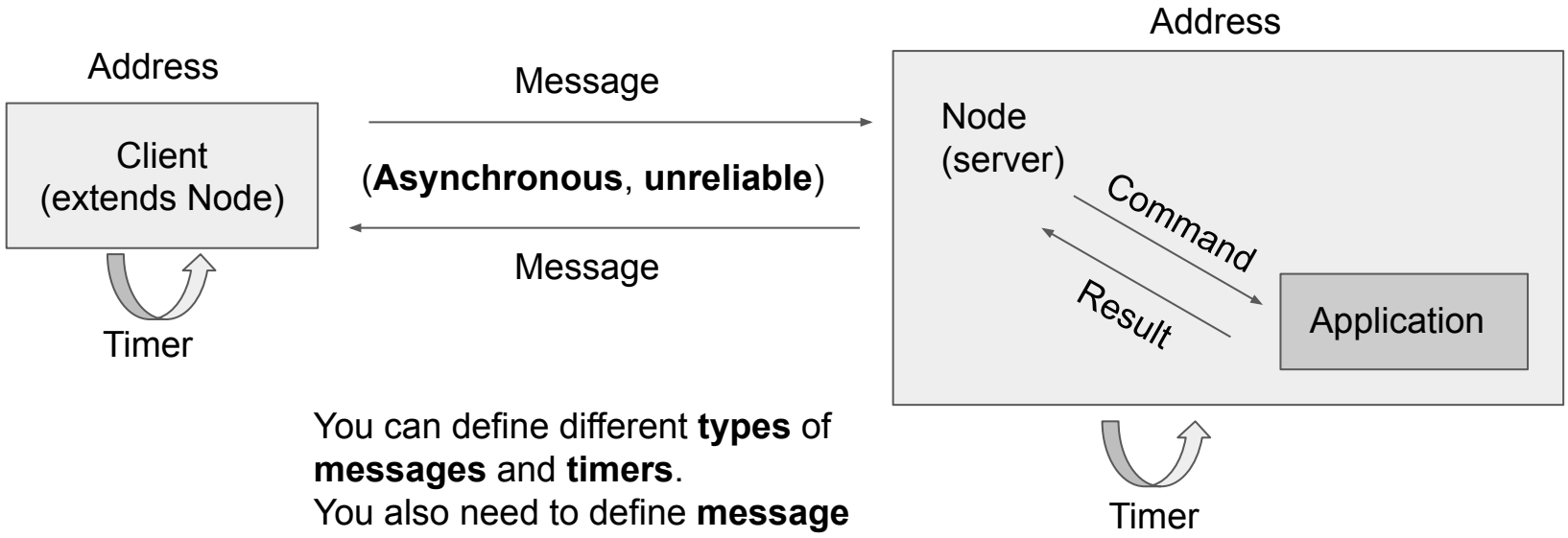
Class	Description
Address	Addresses are opaque objects that uniquely identify `Node`s.
Application	Applications are simple data structures used by your `Node`s.
Client	Clients are a special case of `Node`.
Command	An operation on an `Application`.
Message	Base interface for all messages in the system.
Node	Nodes are the basic unit of computation.
Result	The value returned by an operation on an `Application`.
Timer	Base interface for all timers in the system.

Let's see this visually…

Single client, single server

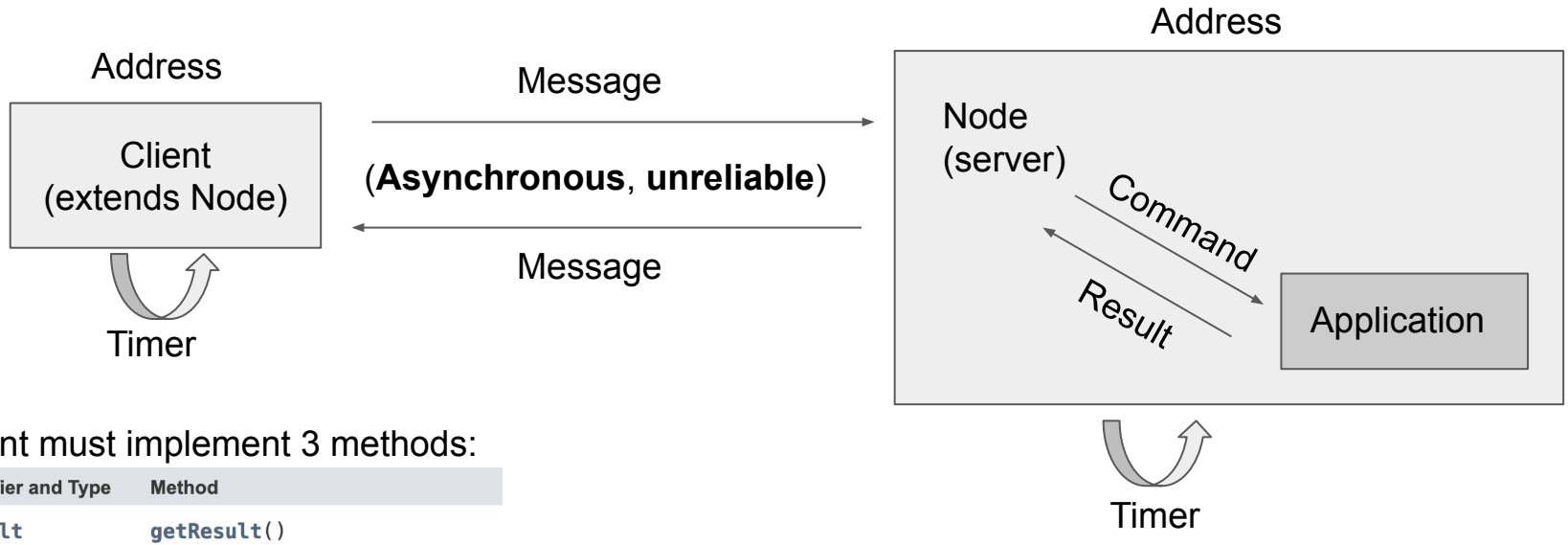


Single client, single server



You can define different **types** of **messages** and **timers**.
You also need to define **message handler** functions and **timer handler** functions accordingly.

Single client, single server

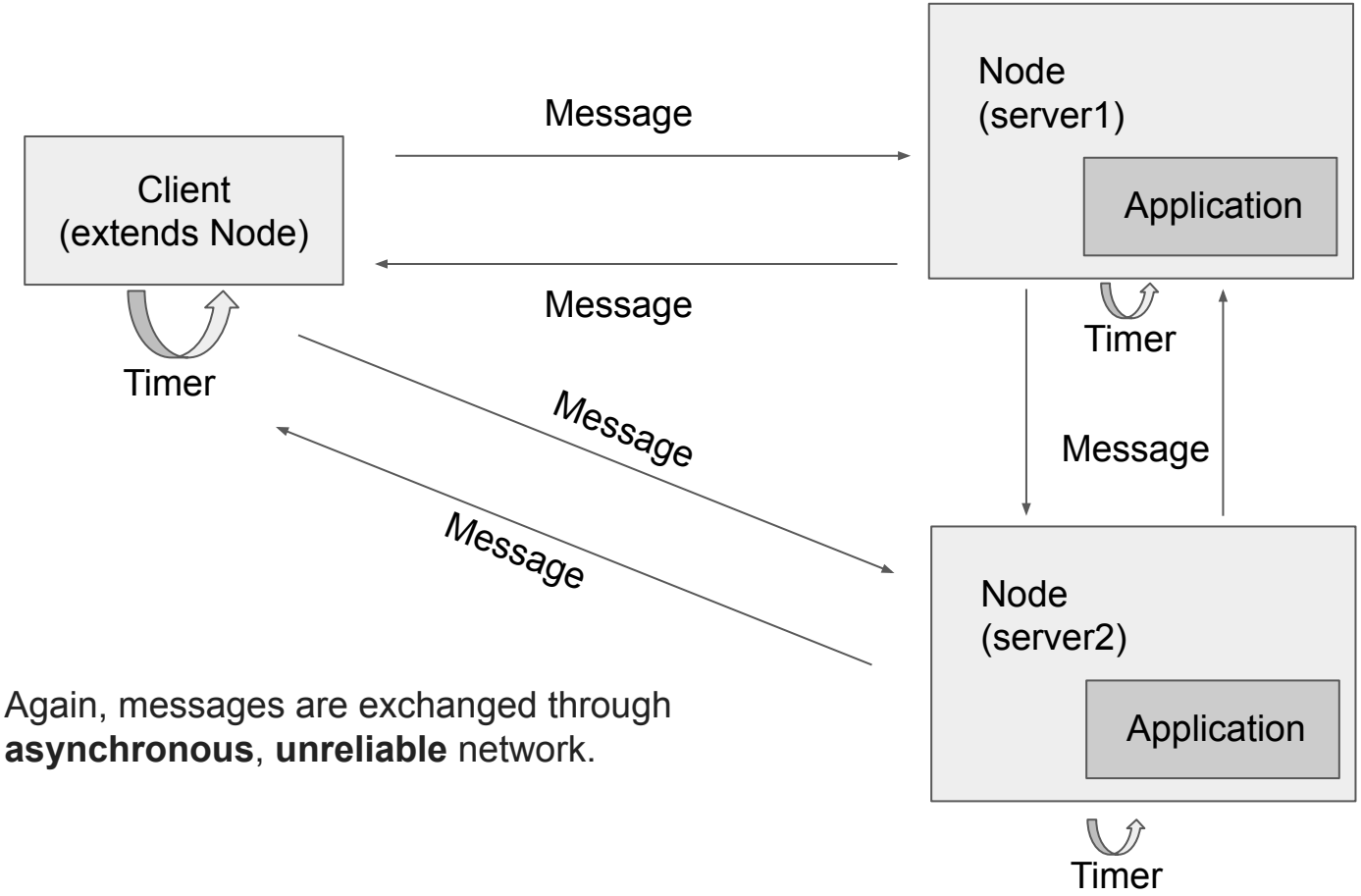


A client must implement 3 methods:

Modifier and Type	Method
`Result`	`getResult()`
`boolean`	`hasResult()`
`void`	`sendCommand(Command command)`

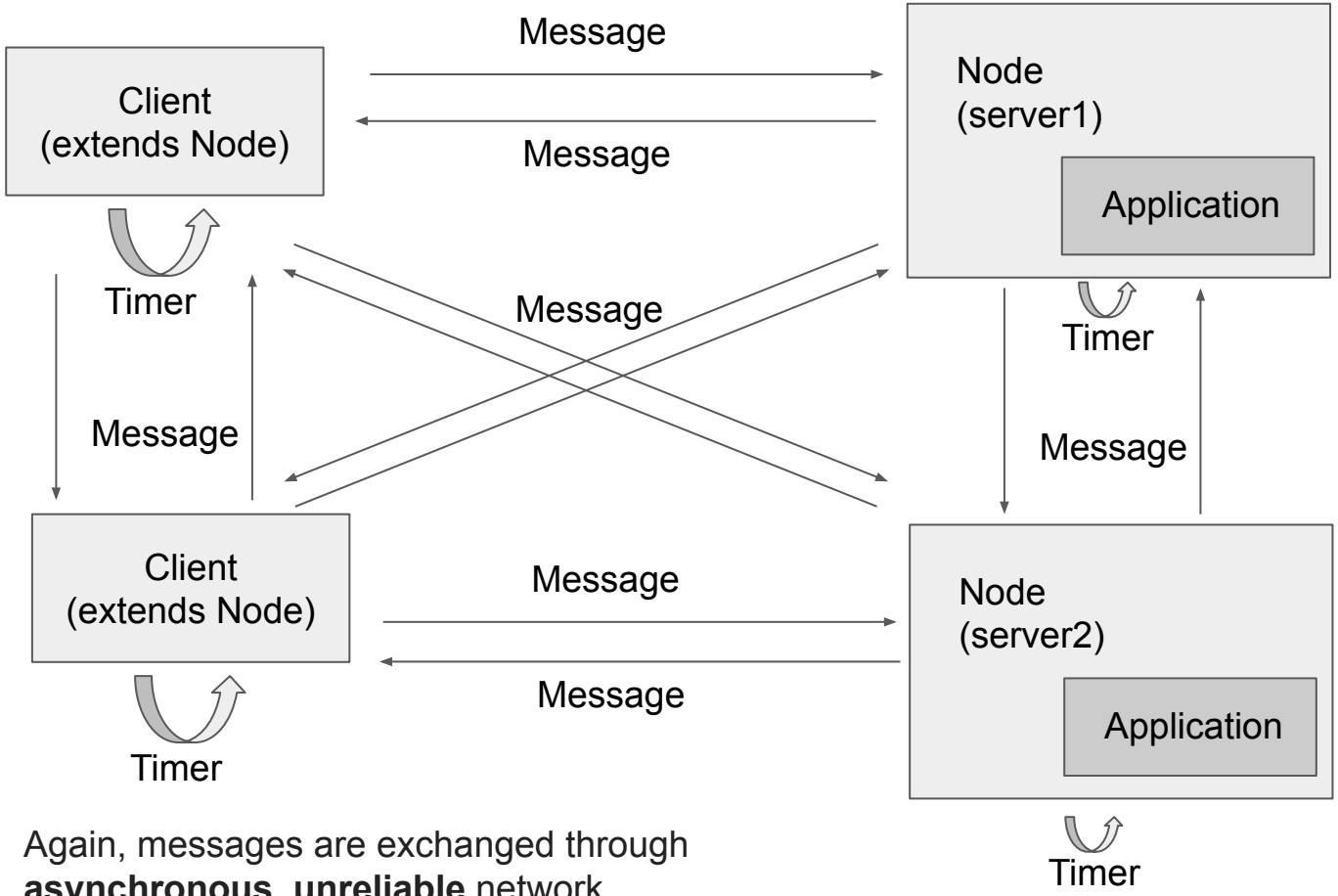
The automated tests will call these three methods to verify the correctness of your protocol implementation.

Single client, **multiple** server



Again, messages are exchanged through **asynchronous**, **unreliable** network.

Multiple client, multiple server



Again, messages are exchanged through **asynchronous**, **unreliable** network.

Lombok

- Instead of repeatedly adding boilerplate code, use Lombok annotations instead
- Read more about it here: https://projectlombok.org/features/Data
- `@EqualsAndHashCode`
  - Generates `equals` and `hashCode` methods for you
- `@Data`
  - "A shortcut for `@ToString`, `@EqualsAndHashCode`, `@Getter` on all fields, `@Setter` on all non-final fields, and `@RequiredArgsConstructor`!"
- **All messages and timers should have @Data**, will lead to an explosion in state space if you don't and you may fail some tests

If you use Intellij, install the Lombok Plugin:
https://projectlombok.org/setup/intellij

Plan for today

Overview of CS 5414 and DSLabs projects

Design docs

DSLabs intro

Lab 0 demo

Lab 1 intro

Lab 0 Demo

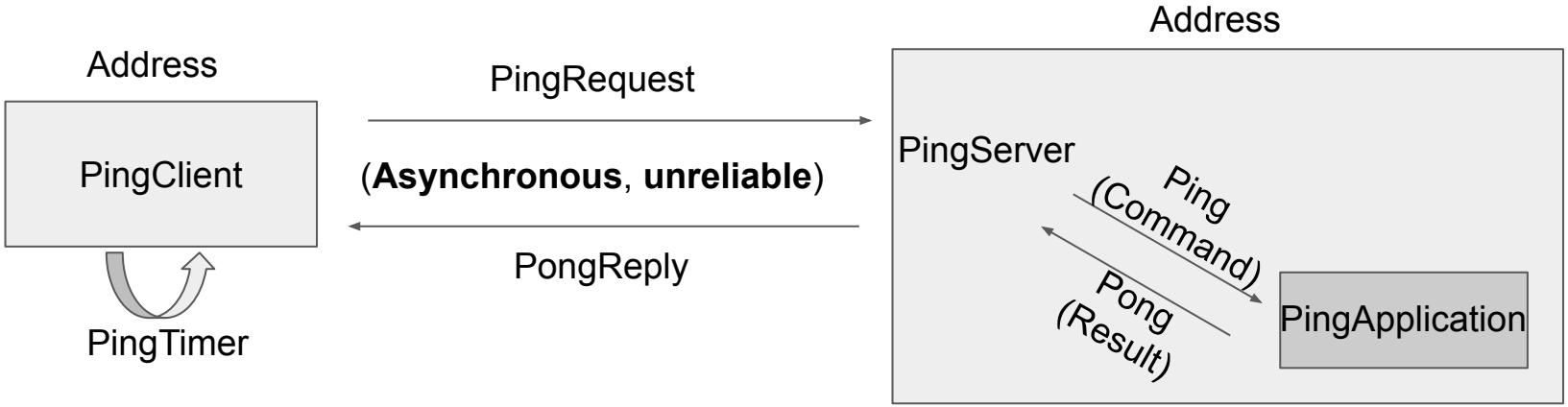
You **don’t** have to submit anything for Lab 0.

It’s just for demonstration purposes to help you get familiar with the DSLabs framework.

Read the README carefully to understand

Lab 1 is the first lab that you need to turn in (along with the **design doc**). It is **significantly easier** than the remaining labs, but the rest of the labs build upon it.

Ping-pong picture



Programming tips

- Think before you code
  - Changing your code in an ad hoc way without fully understanding them is almost always wrong
- Visualizer (search tests)
  - `./run-tests.py --lab 0 --debug 1 1 "Hello World,Goodbye World"`
  - Search tests are usually easier to debug. You might want to fix them first.
- Assert invariants liberally: `if (!invariant) throw …`
- Logging (this is a better alternative to printing)
  - Add `@Log` annotation before a class
  - To log messages in your code, write something like `LOG.info("log something");`
  - Use `run-tests.py -g LOG_LEVEL` to set log levels (e.g, INFO, WARNING, FINER, FINEST, …) when running tests. Default logging level is WARNING - logs with level lower than WARNING are not displayed.
  - Use to `&> log.txt` redirect logging output to text files

Plan for today

Overview of CS 5414 and DSLabs projects

Design docs

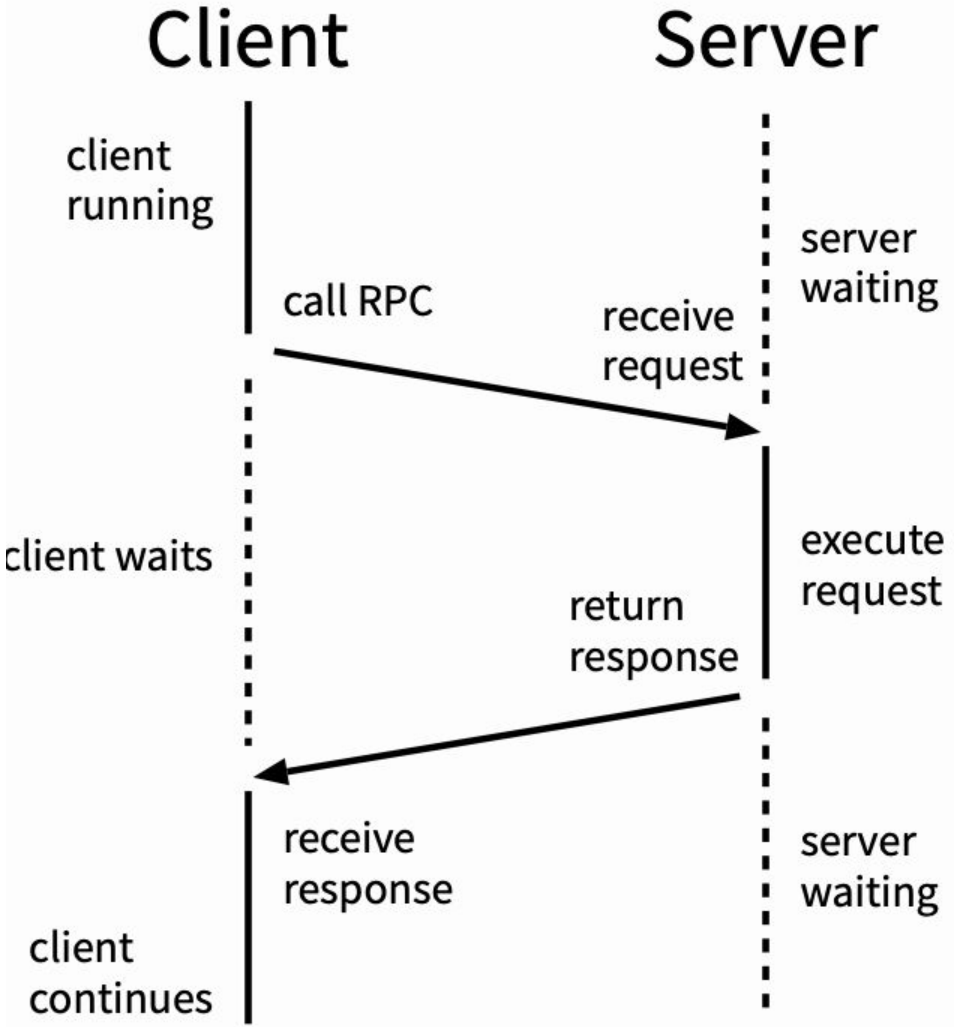
DSLabs intro

Lab 0 demo

Lab 1 intro

Lab 1 Concepts

Remote Procedure Call (RPC)

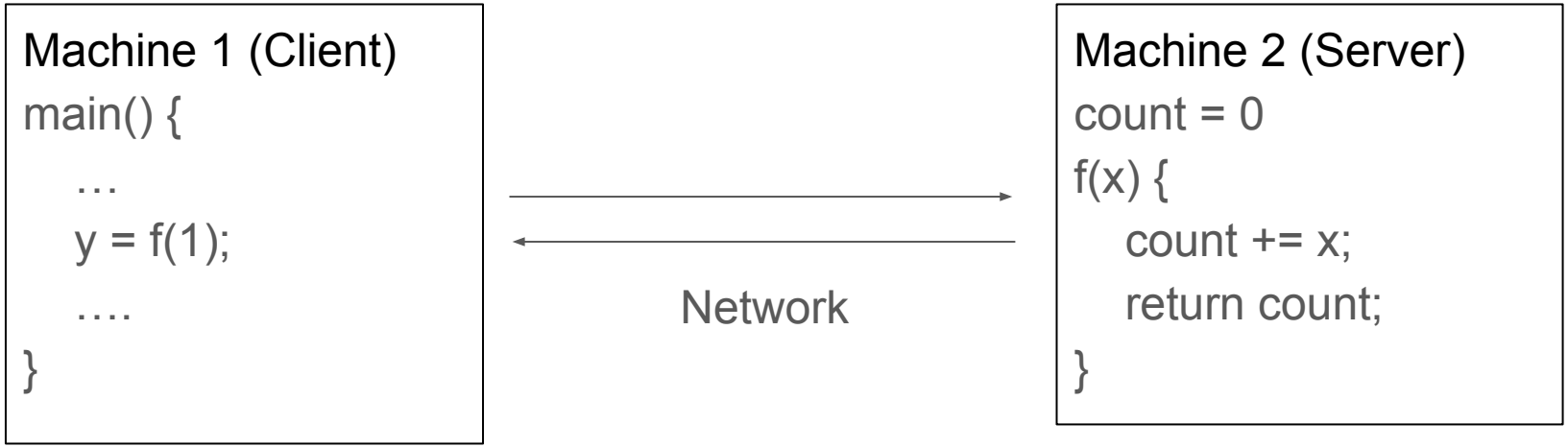


Local procedure call

```
count = 0
main() {
    …
    y = f(1);
    ….
}
…
f(x) {
    count += x;
    return count;
}
```

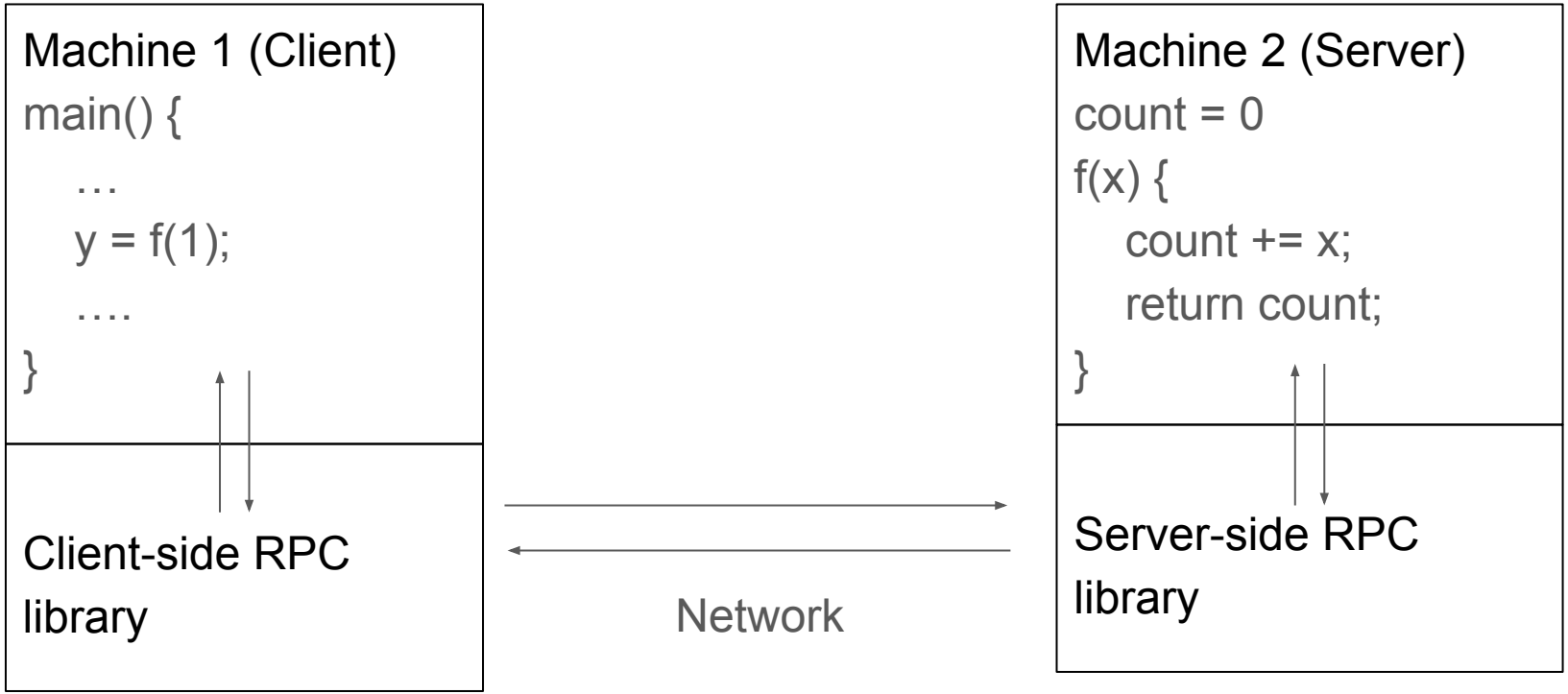
- main() and f(x) are on the **same machine**.
- When main() calls f(x), program counter **jumps** to f(x).
- When f(x) returns, program counter **jumps back** to main().

Remote procedure call (**RPC**)

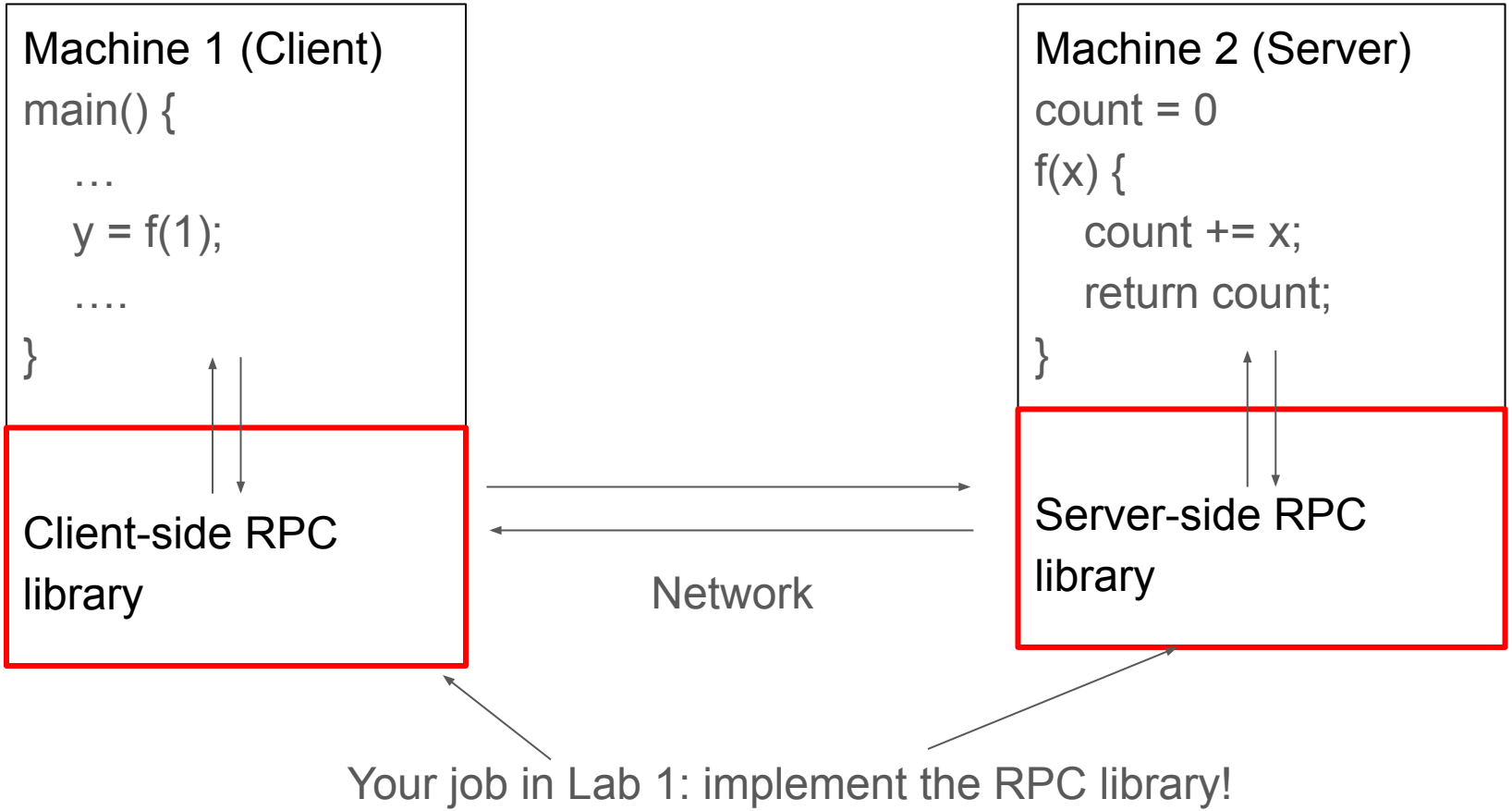


- main() and f(x) are on **separate** machines that communicate over the **network**
- When Machine 1 calls f(x), it calls its **client-side RPC library**
- Client-side RPC library communicates with **server-side RPC library** on Machine 2 over the network
- Server-side RPC library on Machine 2 runs f(x) and returns result back to client side RPC library
- Client-side RPC library returns the result back to main()

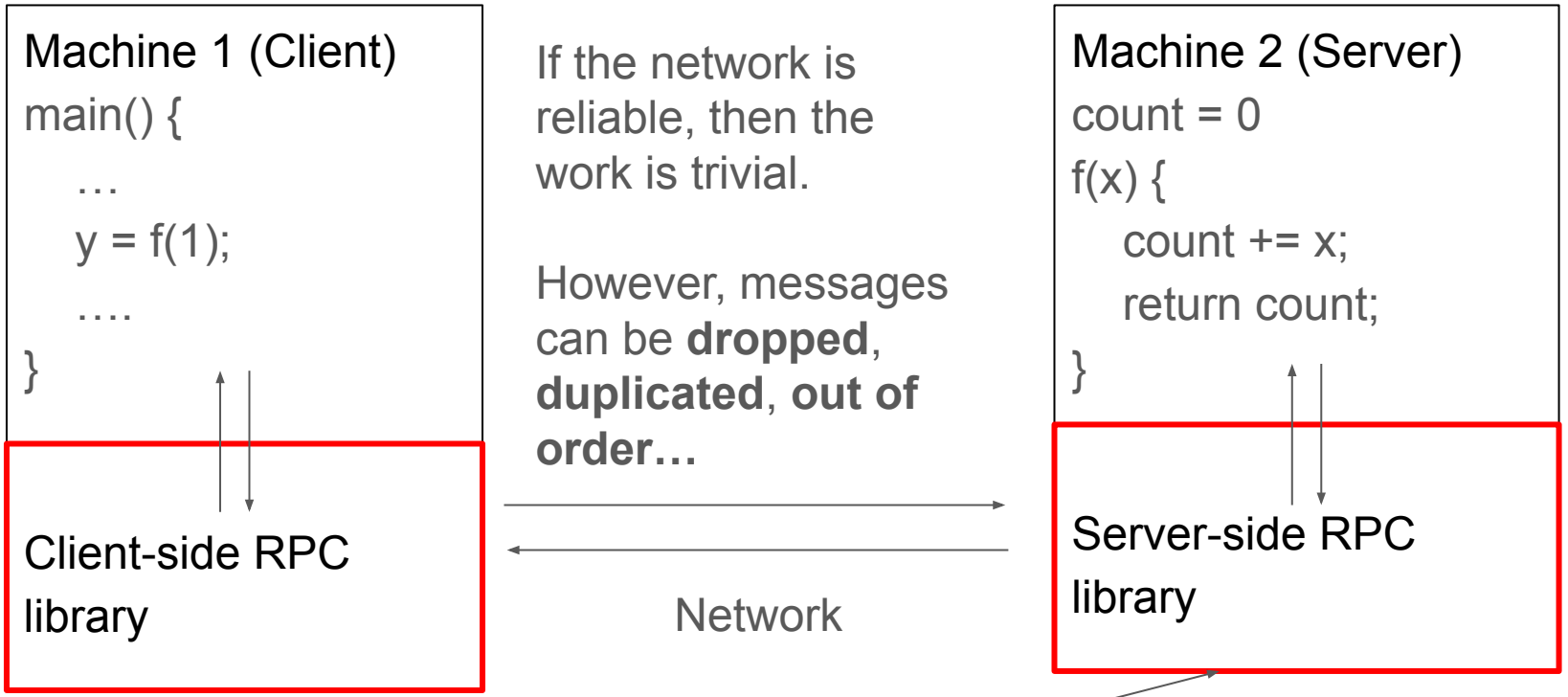
Remote procedure call (RPC)



Remote procedure call (RPC)



Remote procedure call (RPC)



Your job in Lab 1: implement the RPC library!

Hint: You will use **retry** and **sequence numbers** to build reliability on top of unreliable networks!
Remember, clients in this framework only have **one outstanding request at a time**.

Before you start, please read

DSLabs introduction handout: https://github.com/emichael/dslabs/tree/handout

Framework documentation (Javadoc):
https://ellismichael.com/dslabs/javadoc/dslabs/framework/package-summary.html

Lab 0 handout:
https://github.com/emichael/dslabs/tree/handout/labs/lab0-pingpong

Lab 1 handout:
https://github.com/emichael/dslabs/tree/handout/labs/lab1-clientserver

Design doc specification and template:
https://courses.cs.washington.edu/courses/cse452/23wi/resources/design-docs.html

UW slides on Lab 0 + 1 (more hints and suggestions):
https://docs.google.com/presentation/d/13Wn3LT-7Rgw4cMnVbzMPXyDWi8b_kl63cDcnIXGjciY/edit#slide=id.p

Other helpful resources

UW section slides (for lab hints and suggestions):
https://courses.cs.washington.edu/courses/cse452/23wi/section/

UW course website:
https://courses.cs.washington.edu/courses/cse452/23wi/schedule/

DSLabs website: https://ellismichael.com/dslabs/

Git repo: https://github.com/emichael/dslabs/tree/handout

EuroSys paper on DSLabs:
https://ellismichael.com/papers/dslabs-eurosys19.pdf

Final words

Lab 1 design doc and code are due **next Friday (Feb 2)**.

Lab 1 is **individual**. Lab 2-4 can be done in **groups of 2**.

For Lab 2-4, we strongly recommend you form groups. **Start to find your partner**!

Design docs are required and they **MUST** follow this **template**:
https://courses.cs.washington.edu/courses/cse452/23sp/design-docs.html

Start early!

Start early!

Start early!