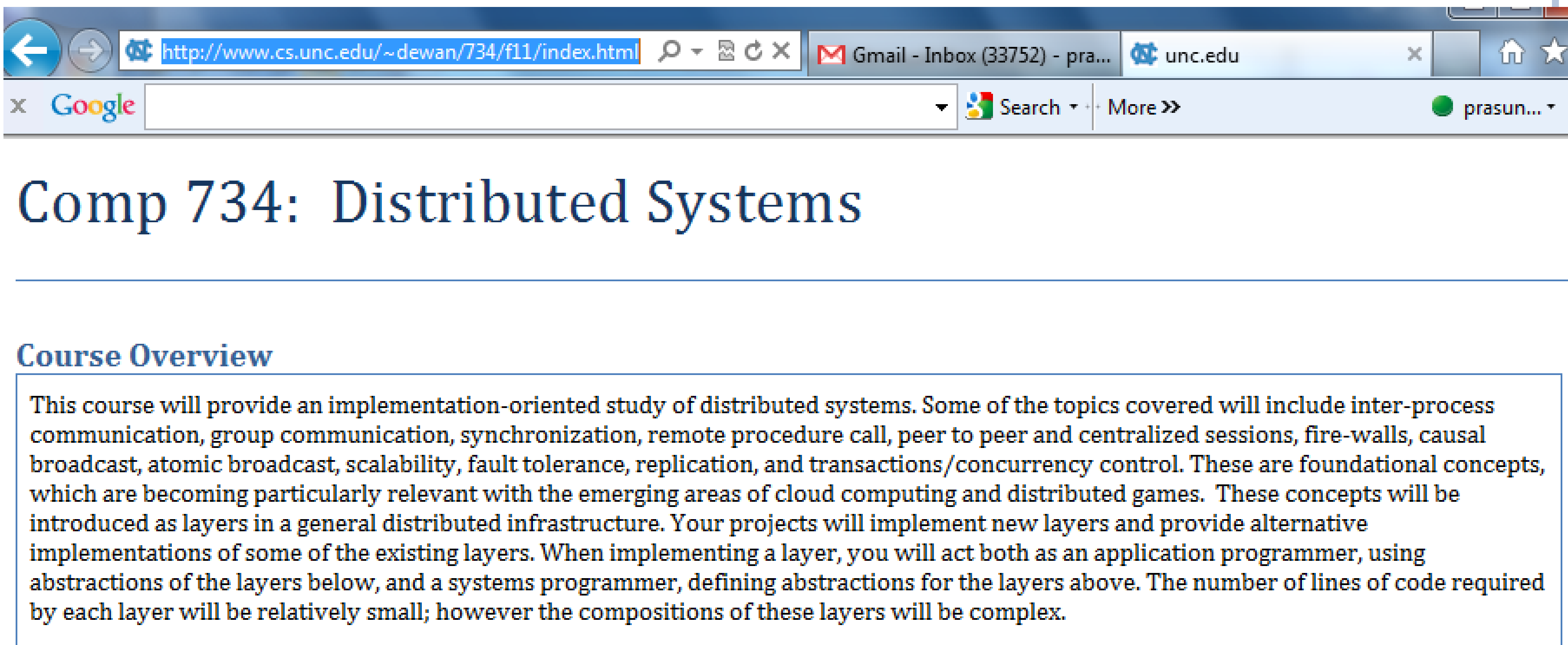
Distributed Systems

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)

Course Home Page



Comp 734: Distributed Systems

Course Overview

This course will provide an implementation-oriented study of distributed systems. Some of the topics covered will include inter-process communication, group communication, synchronization, remote procedure call, peer to peer and centralized sessions, fire-walls, causal broadcast, atomic broadcast, scalability, fault tolerance, replication, and transactions/concurrency control. These are foundational concepts, which are becoming particularly relevant with the emerging areas of cloud computing and distributed games. These concepts will be introduced as layers in a general distributed infrastructure. Your projects will implement new layers and provide alternative implementations of some of the existing layers. When implementing a layer, you will act both as an application programmer, using abstractions of the layers below, and a systems programmer, defining abstractions for the layers above. The number of lines of code required by each layer will be relatively small; however the compositions of these layers will be complex.

http://www.cs.unc.edu/~dewan/734/current/index.html

Lectures and Assignments

Schedule (Tentative)

	Unit (Start Date)	Slides	Chapters	Assignment
	Introduction	PowerPoint 2007		
	Threads and Thread Coordination (Read on your own)	PowerPoint 2007		
	Java Non-Blocking Socket Channel I/O			Distributed Non-Blocking Halloween Simulation
	Java Remote Method Invocation			Distributed RMI-based Halloween Simulation
	Sync replicated Objects			Replicated Sync-based Halloween Simulation
	GIPC Byte Communication (Oct 5, 10)	PowerPoint 2007		Socket-based GIPC
	NIO Driver (Oct 24, 26, 31, Nov 2, 4)	PowerPoint 2007		
				Extendible Multi-Platform Serialization Synchronous Receive, Procedure and Function Call
	GIPC P2P	PowerPoint 2007		

No book

PPT slides and sometimes word doc

Current assignment is on the web - start working ASAP on it

Outline of other assignments given

Software

Downloads

Beau Halloween Simulation (Library, keep it compressed)	beau_project.zip
Coupled Halloween Simulations (Eclipse project, uncompress and link to libraries)	CoupledTrickOrTreat.zip
ObjectEditor (Library)	oeall17.jar
GIPC	

Software to be continuously updated

Grade Distribution

Exams (Two midterms, no final)	40%
Assignments (Home work)	60%
Fudge Factor (Class participation, other factors)	10%

Getting Help

Can discuss solutions with each other at a high level

Not at the code level

Sharing of code is honor code violation

Can help each other with debugging as long as it does not lead to code sharing

Assignments may contain solution in English (read only if stuck)

Piazza

Getting Help and Class Discussion

We will be using Piazza for class discussion and getting help. The system is highly catered to getting you help fast and efficiently from classmates, the TA, and myself. Rather than emailing questions to the teaching staff, I encourage you to post your questions on Piazza. If you do not get a response within a day or two on Piazza, please send mail to help401@cs.unc.edu. But try Piazza first. Do not send mail to an individual instructor, as that can overwhelm him - such mail will be ignored.

Before posing a question, please check if this question has been asked before. This will reduce post clutter and reduce our burden. Repeat questions will be ignored by the instructors.

Piazza allows anyone to respond. So if you see a question that you think you can respond to, please do so, as that will reduce our burden and help you "teach" your fellow students.

This will be a form of class participation that will be noted when I allocate my fudge points!

Hope it works well

If you have any problems or feedback for the developers, email team@piazza.com.

Find our class page at: https://piazza.com/unc/fall2013/comp734

Distributed Program?

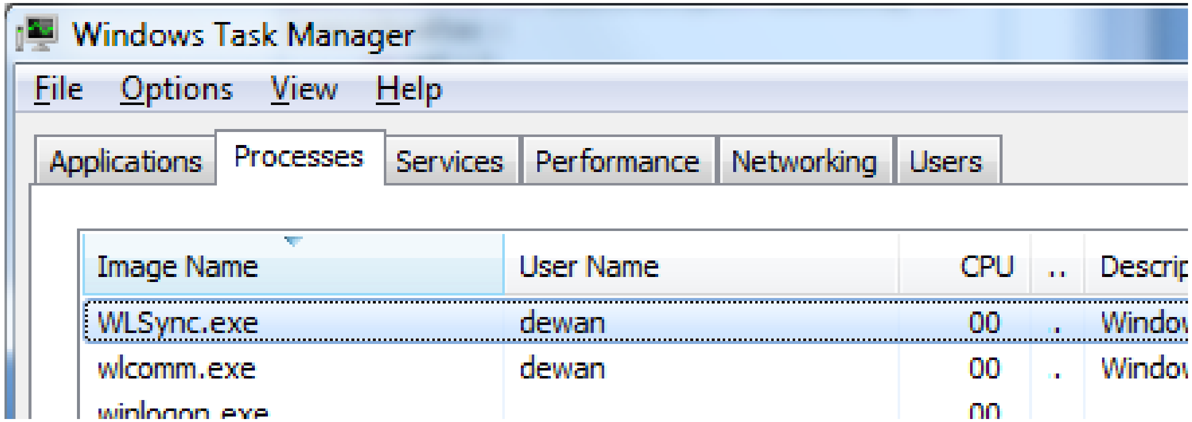
A program "involving" multiple computers

Specific computers must be bound at run time

→ Program can run on a single computer

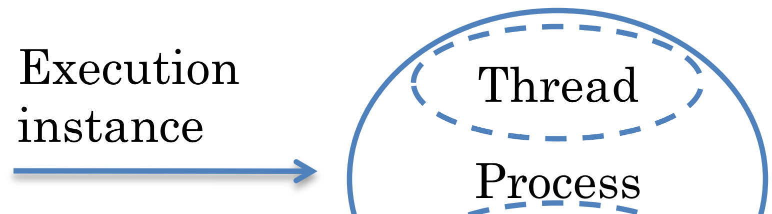
Definition involves processes

Program vs. Process vs. Thread



Program

```
public class AP2PAliceTOTSimulation {
    public static String ID = "9100";
    public static String NAME = "Alice";
    public static int USER_NUMBER = 0;
    public static void main (String[] args) {
        Tracer.showInfo(true);
```



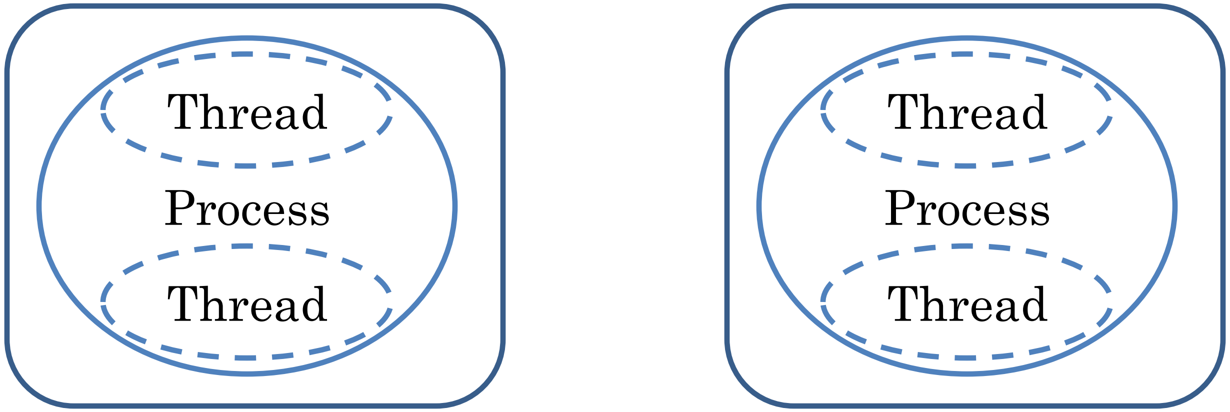
Process is execution instance of program, associated with program and memory

Processes are independent activities that can interleave or execute concurrently

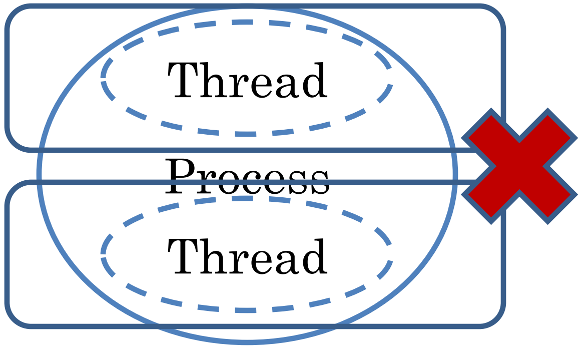
Same program can result in multiple processes

Thread is also an independent activity, but within a process, associated with a process and a stack

Distribution of Processes/Threads

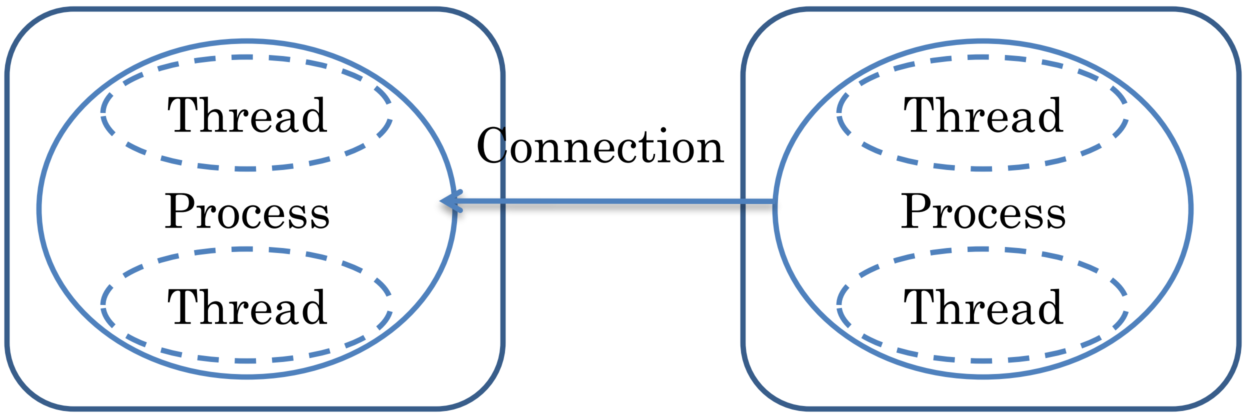


Different processes can execute on different (distributed) computers



A single process executes on one machine

Distributed Program



Connected process pair : Some computation of a process can be influenced by or influence computation of the other process

Connected process group: each process is coupled to at least one other process in the group

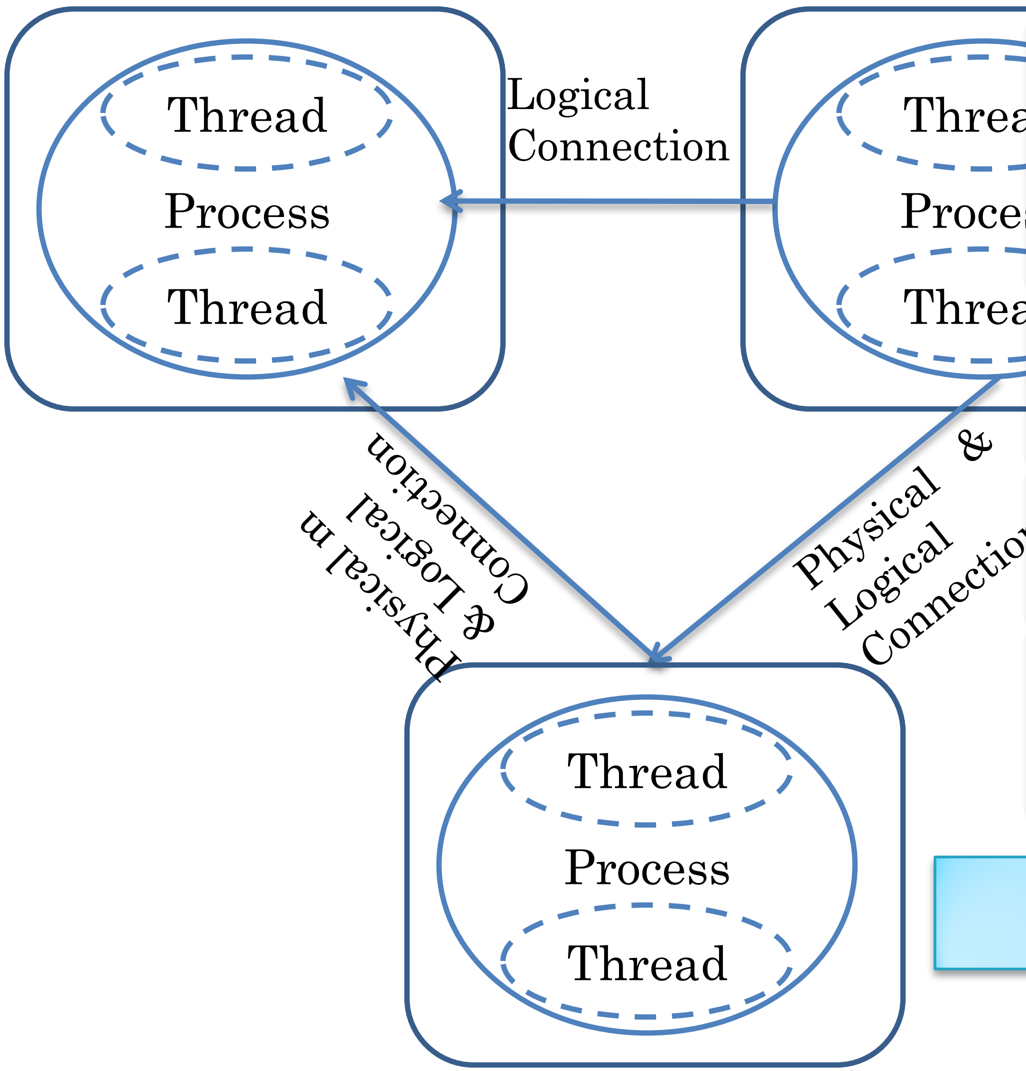
Exec

```
public
    public static String ID = "9100";
    public static String NAME = "Alice";
    public static int USER_NUMBER
    public static void main (Strin
        Tracer.showInfo(true);
        AP2PTOTSessionsClientCreat
    }
}
```

```
    public static String ID = "9100";
    public static String NAME = "Alice";
                          USER_NUMBER = 0;
                          main (String[] args) {
                          o(true);
                          sClientCreator.createP2P
```

Graph crated by creating pair-wise dependency links is not partitioned– every node reachable from every other node

Logical vs. Physical Inter Process Connection Links



Physical coupling links are physical inter process communication links along which information flows in the network

Logical links indicate computational dependencies

Can have logical links without physical links

Physical links usually imply logical links

Relayer

Distributed Applications

Distributed applications?

Non distributed applications?

In today's world, what is or should not be distributed?

Some Distributed Domains

- Distributed Repositories (Files, Databases)
- Remotely Accessible Services (Printers, Desktops)
- Collaborative Applications (Games, Shared Desktops)
- Distributed Sensing (Disaster Prediction)
- Computation Distribution (e.g. Simulations)

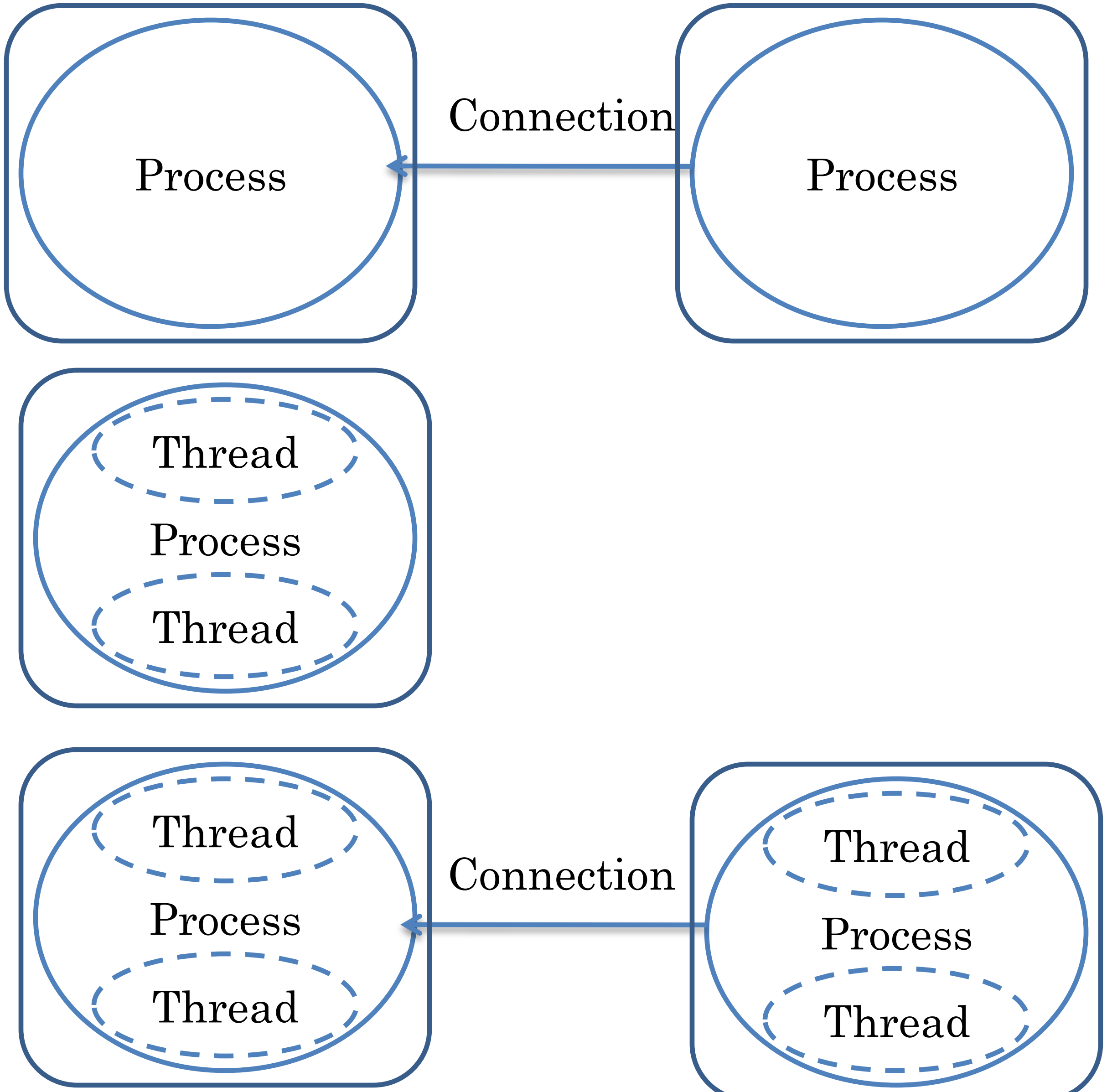
Full courses on some of these areas, with concepts specific to them (Distributed Databases, Collaborative Applications)

Will look at domain-independent concepts at the intersection of them

Will not take an application-centric view

Fundamental Issues?

Distribution vs. Concurrency Program



Distribution, no fine-grained concurrency

Concurrency, not distribution

Distribution and fine-grained concurrency (typical)

Non-Distributed vs. Distributed Program

Non-Distributed	Distributed
Creates a single process logically and physically unconnected to any other process	Creates a pair or larger group of connected processes
Must deal with sequential and possibly concurrency issues	Must also deal with distribution and usually concurrency issues

Systems ViewPoint

System	Computer abstractions to implement some class of programs
Operating System	Processes, Files, Memory Management , Threads…,
Database Management System	Query Language, Transactions, …
Programming Languages	Arrays, Loops, Classes, …
Distributed Systems	Data Communication, Remote Procedure Call (RPC), …

Byte/object communication consists of byte/object of exchange

RPC assumes communication consists of procedure requests and return value responses

Distributed Systems

Study of design and/or implementation of computer abstractions for developing distributed programs

Why distributed systems?

Why systems?

Alternatives to understand how to program some domain of applications?

Non distributed programs?

Alternatives to Understanding

Programming: Abstraction use

Programming: Use of a specific set of non distributed abstractions (e.g. , functional, MATLAB programming)

Distributed Programming : Use of a set of distributed abstractions (e.g. Socket/RPC Programming)

Systems: Abstraction design and/or implementation

Design and implementation of non distribution abstractions (Object-Oriented vs. Functional Languages, Compilers/Interpreters)

Design and implementation of distributed system abstractions (e.g. Data Communication /RPC Design and/or Implementation)

Theory: Models and algorithms

Non distributed model and algorithms (Turing Machines, HeapSort,)

Distributed Models and Algorithms(e.g. 2-Phase commit, Group Comm. Model)

Rationale

Abstraction Design vs. Implementation

Abstraction design linked to implementation: Designs are done of only efficiently implementable abstractions

Abstractions vs. Theory (Models, Algorithms)

Abstractions are implemented operational models and have (the more) practical algorithms in them

Abstraction Design & Implementation vs. Use

Maturity with design and implementation issues allows you to better understand the semantics of a specific abstraction.

Abstract implementations require advanced programming/ software engineering techniques– "you cant really program if you have not written a compiler"

Teaching Abstraction Design & Implementation?

Lectures address design; assignments, implementation (e.g. Implement a PL interpreter in another PL)

Implementations can be complex and need instruction

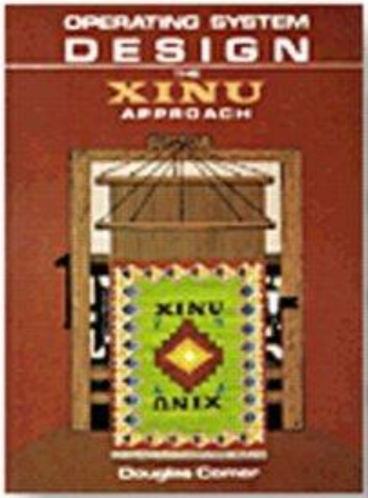
Lectures give high-level pseudo code for complex algorithms; assignments full implementation (e.g. compilers)

Pain/gain ratio high, semester barely enough time for compiler

Lectures discuss code for a system of abstractions : assignments extend/modify this code

Code must be understandable and ideally also elegant

THE XINU APPROACH TO TEACHING OS



Layering

Interrupt Management

Thread Communication

Thread Synchronization

Thread Management

Approach not used in distributed computing

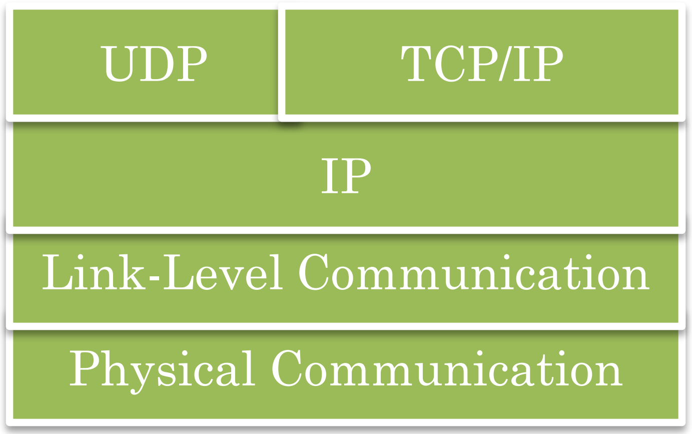
Need distributed system layers

Reuse of previous layers keeps code short (and hence presentable in class)

Can unravel a system in stages to a class

Layering good for software engineering as well as pedagogical reasons

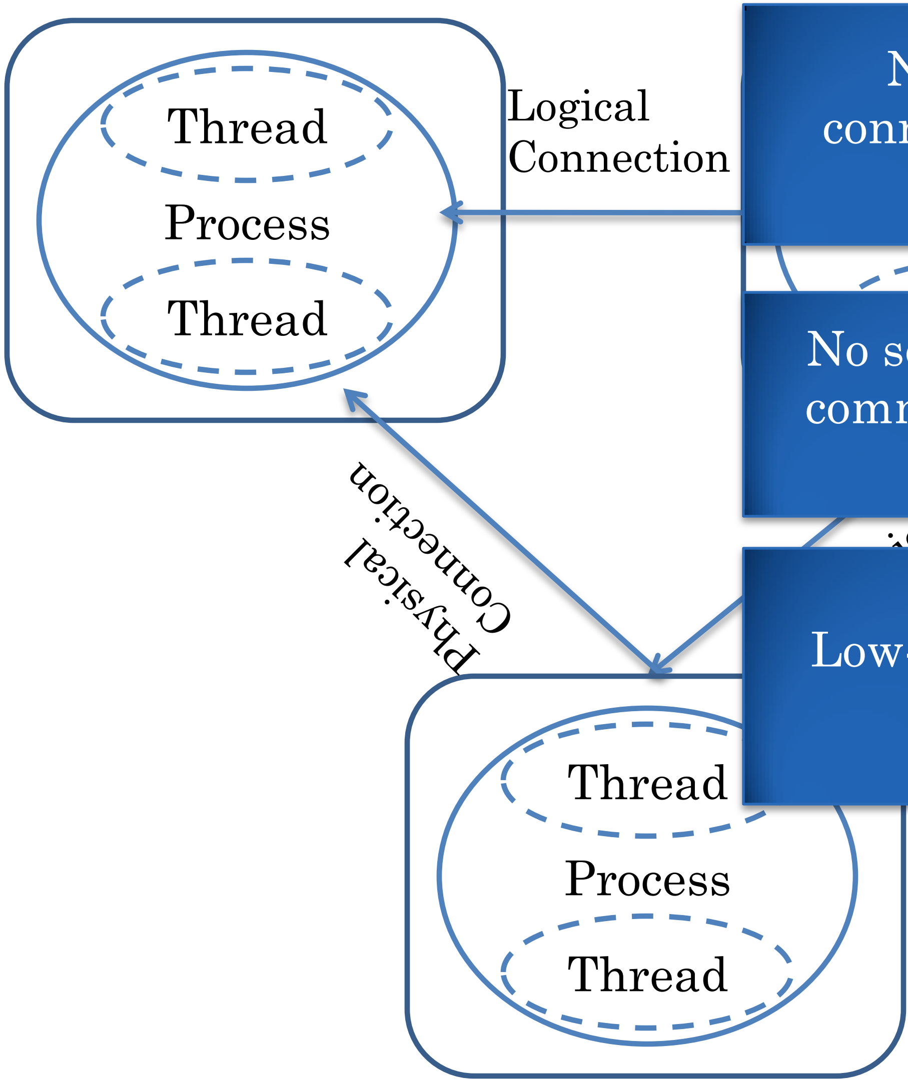
Layers Exist in Networking



Physical communication in networking involves machines and used hardware machine addresses

Physical communication in distributed systems is between processes and indicates routing of information among processes

DISTRIBUTED VS. NETWORK LAYERS



Networking addresses physical connections and byte communication among processes

No separate logical connections, object communication, synchronization, fault tolerance

Low-level (hidden from programmers) abstractions

Distributed System vs. Networking Abstractions

Programming Language Abstractions
Assembly Language Abstractions

Just as programming language abstractions are built on top of assembly language abstractions

Distributed Abstractions
Networked Abstractions

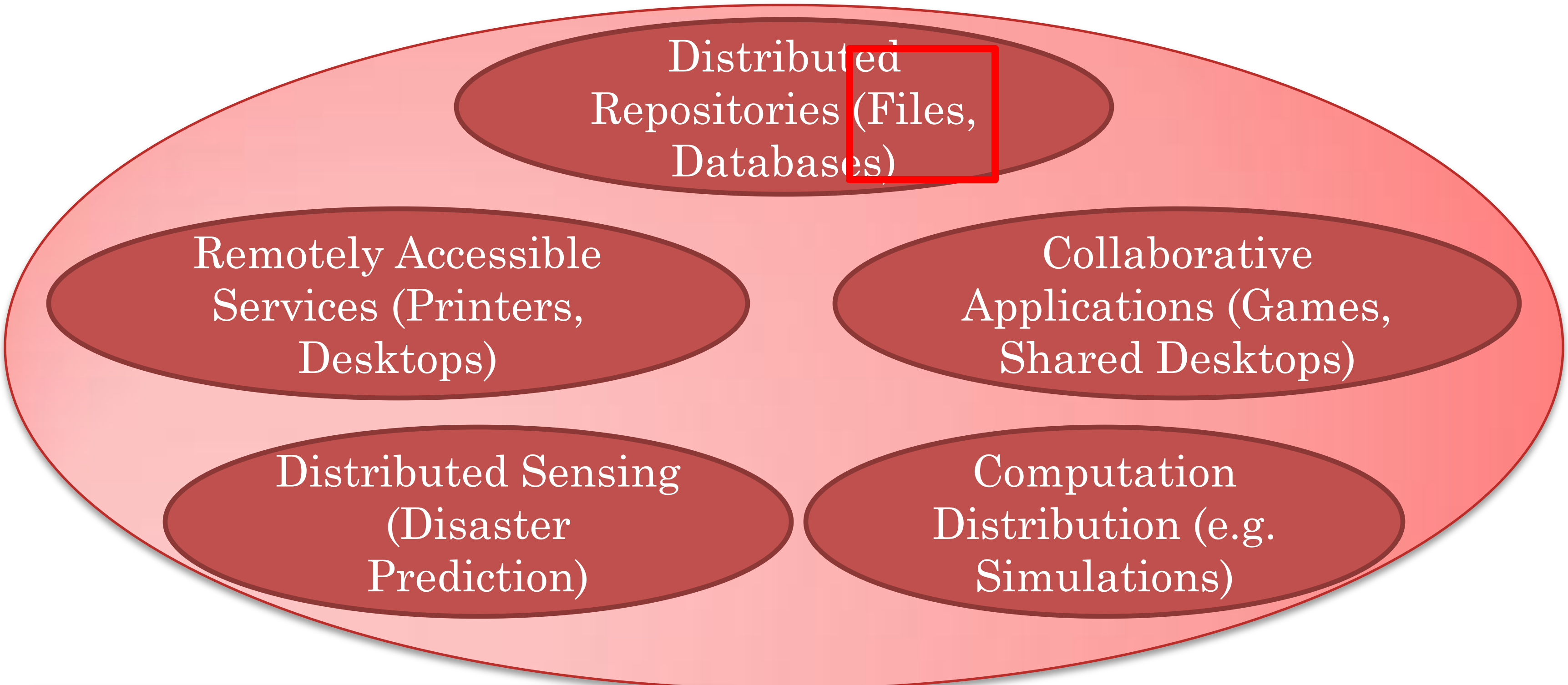
Distributed system abstractions are built on top of networked abstractions

Knowledge of assembly/networked abstractions important to implement PL/distributed abstractions

Distributed Abstractions
OS Byte Communication API

Byte communication APIs, close to networked abstractions, is provided by operating systems (e.g. sockets), which hide networking abstractions

Domain Independent?



Distributed Abstractions

OS Byte Communication API

Will look at domain-independent concepts at the intersection of them

Even though OS abstractions developed to build distributed OS (file systems), they are by definition domain-independent

Language vs. OS Abstractions

Both operating systems and programming languages provide domain-independent abstractions

Operating systems support processes and language-independent abstractions for accessing protected info and sharing information among processes (files, IPC)

Programming languages must provide fine-grained abstractions needed within a process

They also provide an interface to OS abstractions through libraries or language constructs

They can also extend the OS abstractions (e.g. typed files)

Language vs. OS, Distributed Abstractions

Byte communication is all that operating systems provide

Non distributed programming languages such as C provide only OS abstractions

Distributed programming languages such as Java provide a richer variety of abstractions

Java provides threads and reflection, making it easy to implement our own replacements and extensions of Java abstractions

Will use Java as implementation language

To extend and replace Java abstractions/layers, knowledge of them useful

Java Abstractions

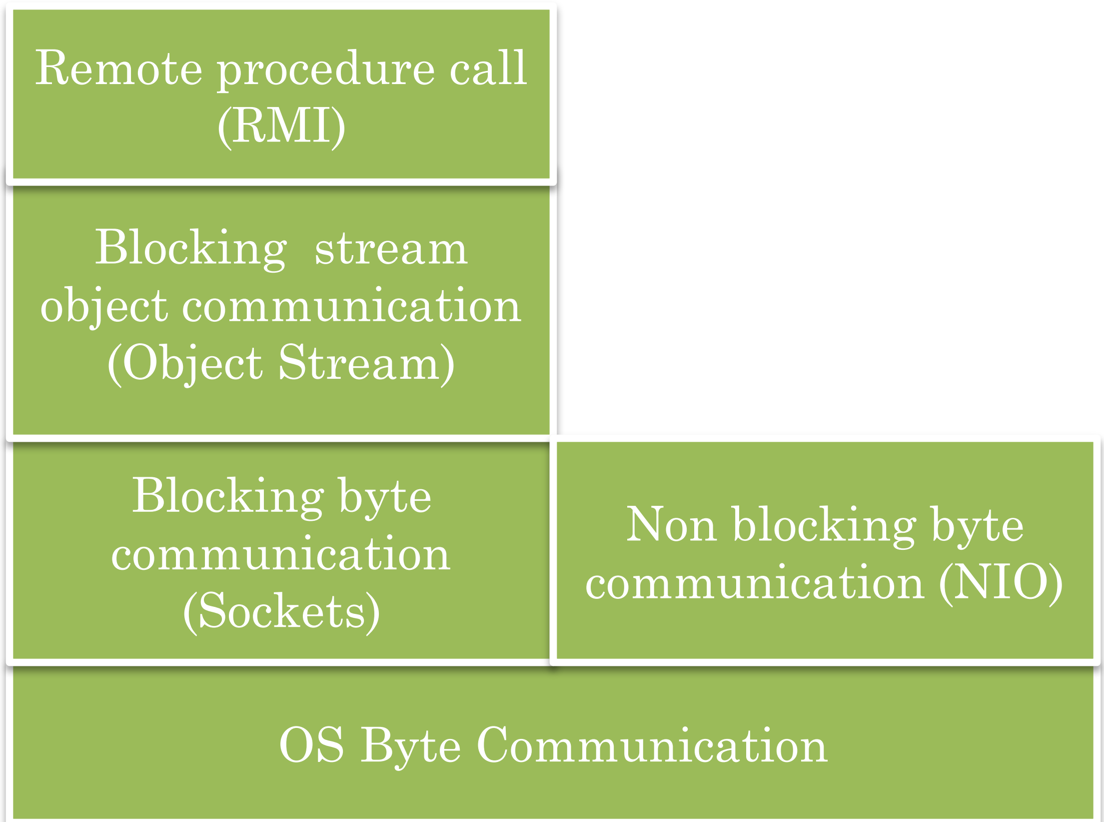
Blocking stream object communication (Object Stream)

Blocking byte communication (Sockets)

Non blocking byte communication (NIO)

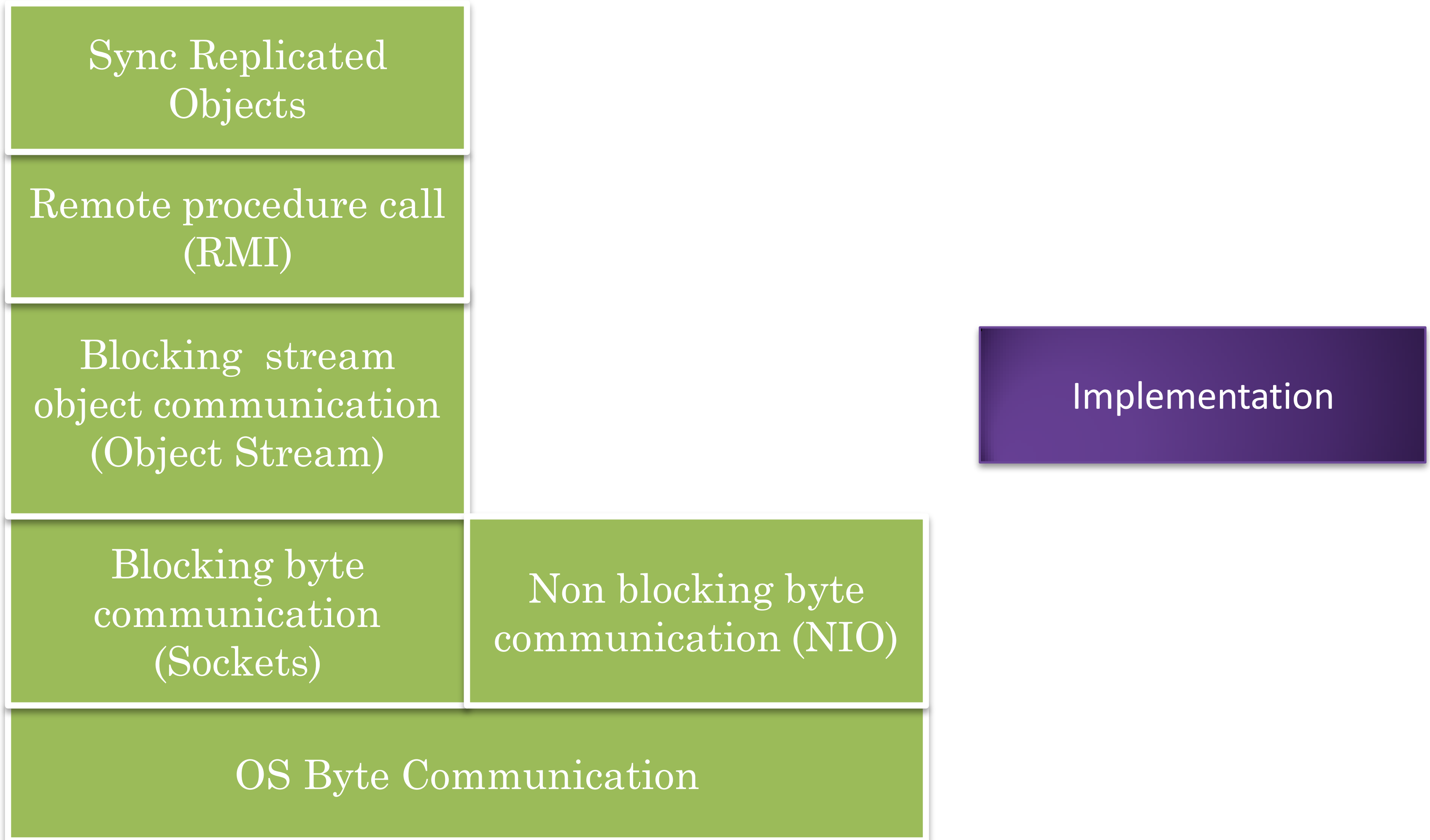
Remote procedure call (RMI)

Java Layers

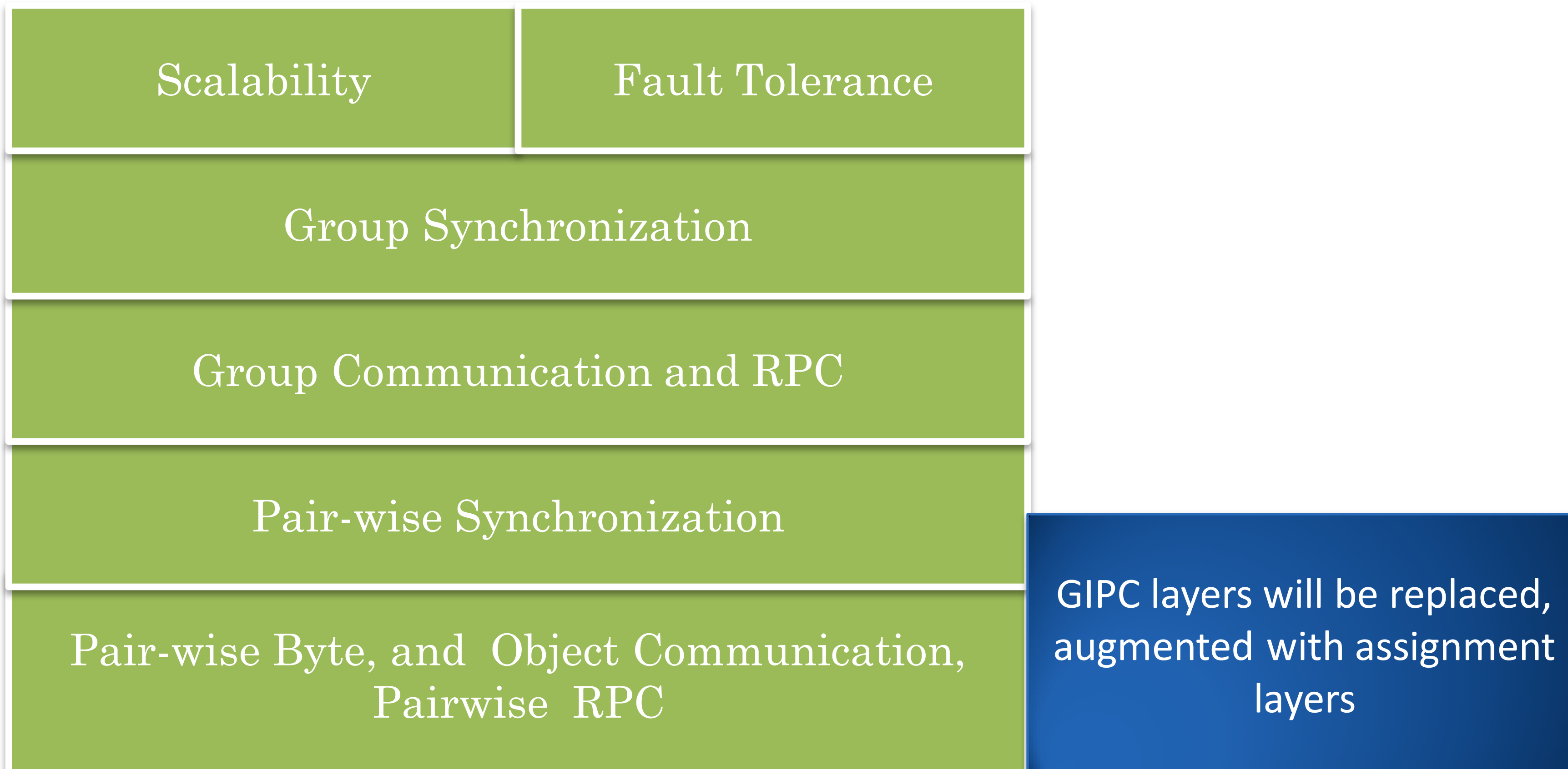


Go beyond Java layers?

Beyond Java Layers



GIPC: Improved Abstractions and Layers with Open Source



Course Plan Principle

Lectures

Assignments

Cover material for next assignment (and other relevant material)

Do next assignment

Boundary conditions?

Use Non Blocking I/O

Lectures

Java NIO

Use NIO Tutorials and class lectures to Implement a Distributed Program

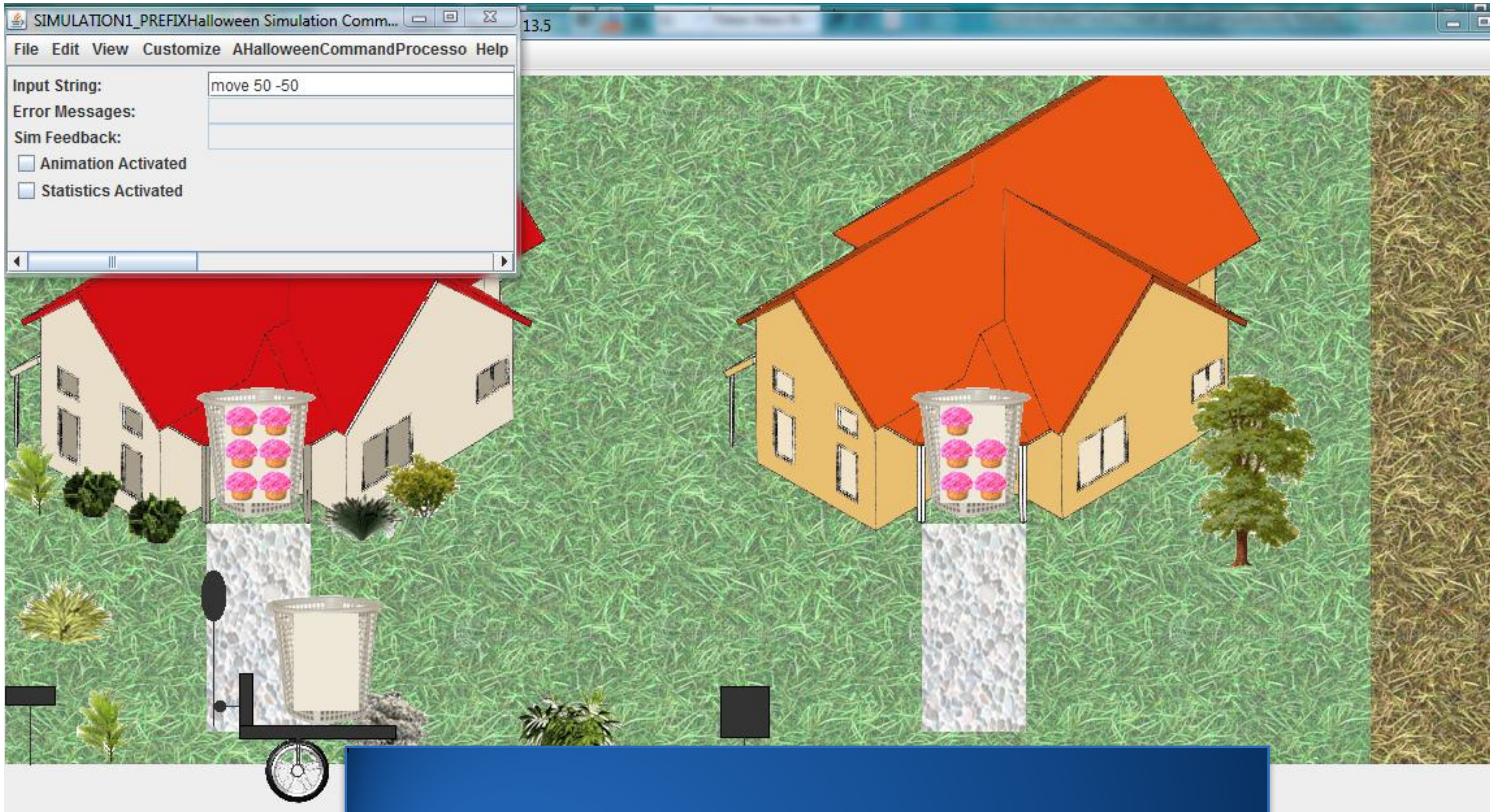
Assignments

Distributed Non Blocking Simulation

Existing non distributed simulation

Java NIO

Halloween Simulation



Make Beau Anderson's 401 Halloween implementation distributed

Use RMI

Lectures

Java RMI

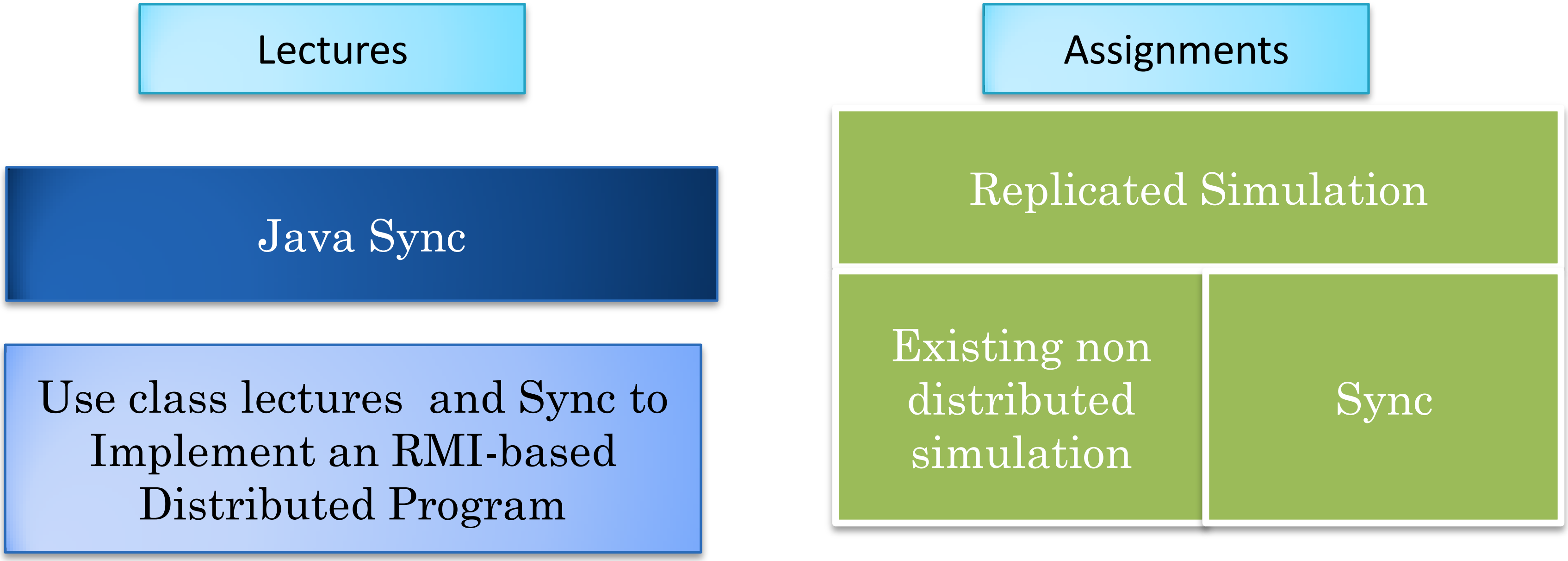
Use Java Tutorials and class lectures to Implement an RMI-based Distributed Program

Assignments

Distributed RMI-based Simulation

- Existing non distributed simulation
- RMI

Use Sync Replicated Objects



Blocking vs. Non Blocking Sockets

Lectures

Assignments

High-level buffer communication

Java (Non Blocking) Channels

Understand the differences between blocking and non blocking communication

High-level buffer communication

Java (Blocking) Sockets

Recursion and Serialization

Lectures

High-level object communication

High-level Buffer comm.

Java Object Streams

Understand serialization and really understand recursion

Assignments

High-level object communication

High-level Buffer comm.

Custom Object Serialization

Synchronization and RPC

Lectures

Assignments

Remote Procedure Call

High-level Object comm.

Java Thread Synchronization

Remote Procedure Call

High-level Object Communication with Synchronization

High-level Object comm.

Java Thread Synchronization

Use and implement pairwise synchronization

Group Communication and Fault Tolerance

Lectures

Fault Tolerance

Group Communication

High-level Object comm.

RPC

Use and implement group synchronization and fault tolerance and group communication

Assignments

More Efficient Fault Tolerance

More Functional Group Communication

High-level Object comm.

RPC

LAST PHASE

Lectures

More Efficient Fault Tolerance

More Functional Group Communication

High-level Object comm.

RPC

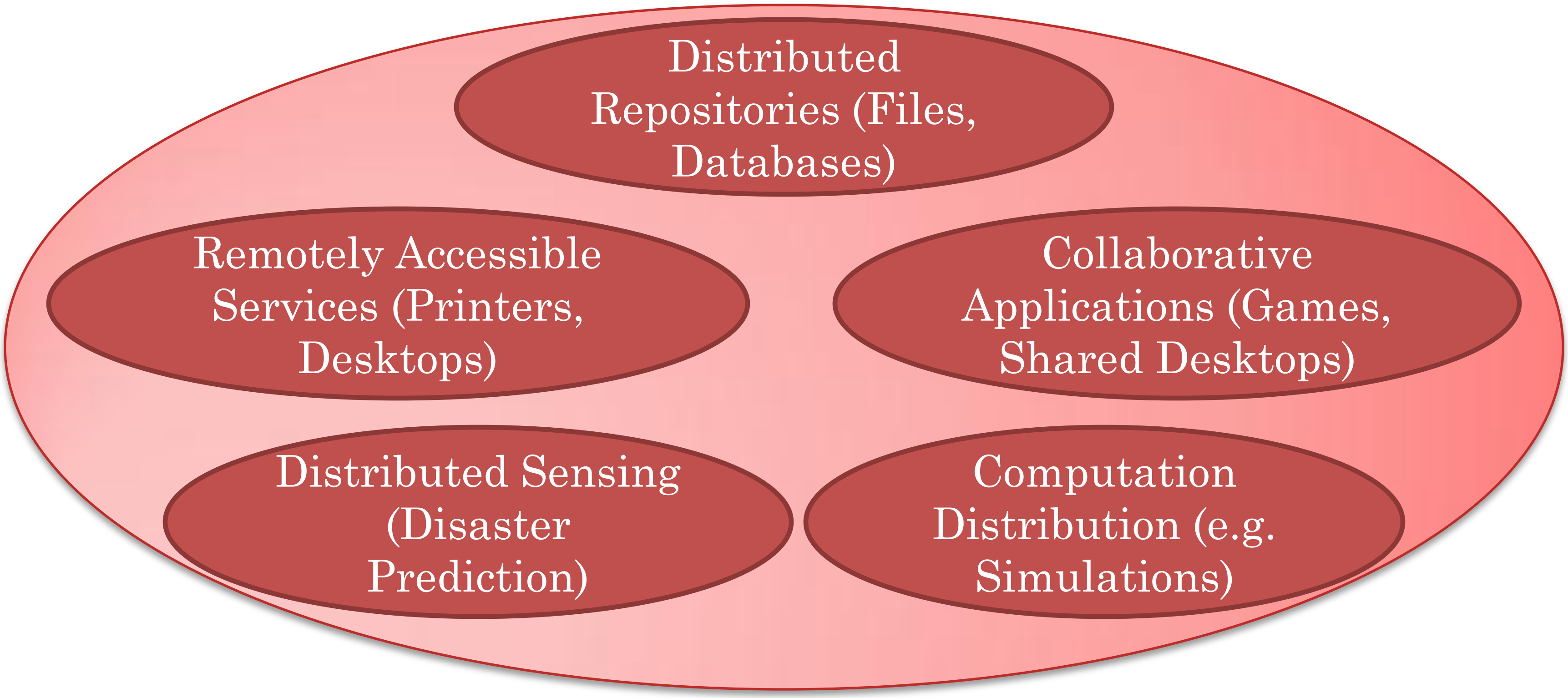
Assignments

Transactions?, Distributed Hashtables?, Multiprocessor systems?, ….

Objectives

At the end of the course you will …..

Distributed Computing



Internet/Cloud computing increasing relevance of the fundamental concepts

Practical Relevance

For distributed applications, likely to use the code you implemented than existing abstractions

Can send objects over NIO socket channels

Existing Java RPC does not work on Android devices, but the one you implement will

Use Sync, which apparently is the basis of some new Mobile platforms

Will implement many abstractions not part of standard Java

Software Engineering Principles

Classes	Existing classes will be used, inherited but not modified directly
Interfaces	Alternative implementations will create new classes implementing existing interfaces
Factories and Abstract factories	These will allow easy switching between different implementations
Generics	Implementation rather than use of generics to unite buffer and object communication

Will be both a distributed computing and software engineering course

Relevance to OS

Inter-process communication key to design of new OS's, even non distributed OS

Extensive use of bounded buffers

Will study and use thread synchronization in depth

Will study how distributed OS are implemented

Will gain understanding of fundamental OS concepts except memory management

Introduction to Systems

Systems: Abstraction design and implementation

Design and implementation of non distribution abstractions (Object-Oriented vs. Functional Languages, Compilers/Interpreters)

Design and implementation of distributed system abstractions (e.g. Data Communication /RPC Design and/or Implementation)

Distributed systems covers concepts from many fields

Extra Slides

ALTERNATIVE JAVA LAYERS

Remote procedure call (RMI)	Remote procedure call
Blocking stream object communication (Object Stream)	Non blocking object communication
Blocking byte communication (Sockets)	Non blocking byte communication (NIO)
OS Byte Communication	

Could have more efficient RPC and non blocking object communication

Two RPC's?

Improved Alternative Java Layers

Remote procedure call

Blocking stream object communication (Object Stream)

Non blocking object communication

Blocking byte communication (Sockets)

Non blocking byte communication (NIO)

OS Byte Communication

- Could do late binding between RPC and lower-level communication
- Go beyond Java abstractions?
- Cannot unite NIO and socket at byte or object level
- Socket communication is low level
- NIO is even lower level
- Programmers rely on usage patterns

Pattern vs. Abstraction (1-Computer Programming)

Pattern

```
public final static int RED = 0;
public final static int BLUE = 1;
public final static int GREEN = 2;
int color = RED;
```

```
public final static int LIKE= 0;
public final static int DISLIKE = 1;
public final static int NEUTRAL = 2;
int response = NEUTRAL;
```

Abstraction

```
public enum Color {RED, BLUE, GREEN};
Color color = Color. RED;
```

```
public enum Response {LIKE, DISLIKE, NEUTRAK};
Response response = Response.NEUTRAL;
```

Java NIOTutorial for Echo Server

```
public void run() {
   while (true) {
     try {
       // Process any pending changes
       synchronized(this.changeRequests) {
         Iterator changes = this.changeRequests.i
         while (changes.hasNext()) {
           ChangeRequest change = (ChangeReque
           switch(change.type) {
           case ChangeRequest.CHANGEOPS:
             SelectionKey key = change.socket.keyFor(this.selector);
             key.interestOps(change.ops);
           }
         }
         this.changeRequests.clear();
       }
       .....

 }
```

Vast majority of tutorial readers will copy and edit this pattern

Much better to identify a corresponding abstraction and implement it to understand channels

:

NioServer.java
EchoWorker.java
ServerDataEvent.java
ChangeRequest.java
NioClient.java
RspHandler.java

Problem with Java Abstraction Level

Remote procedure call

Blocking stream object communication (Object Stream)

Non blocking object communication

Blocking byte communication (Sockets)

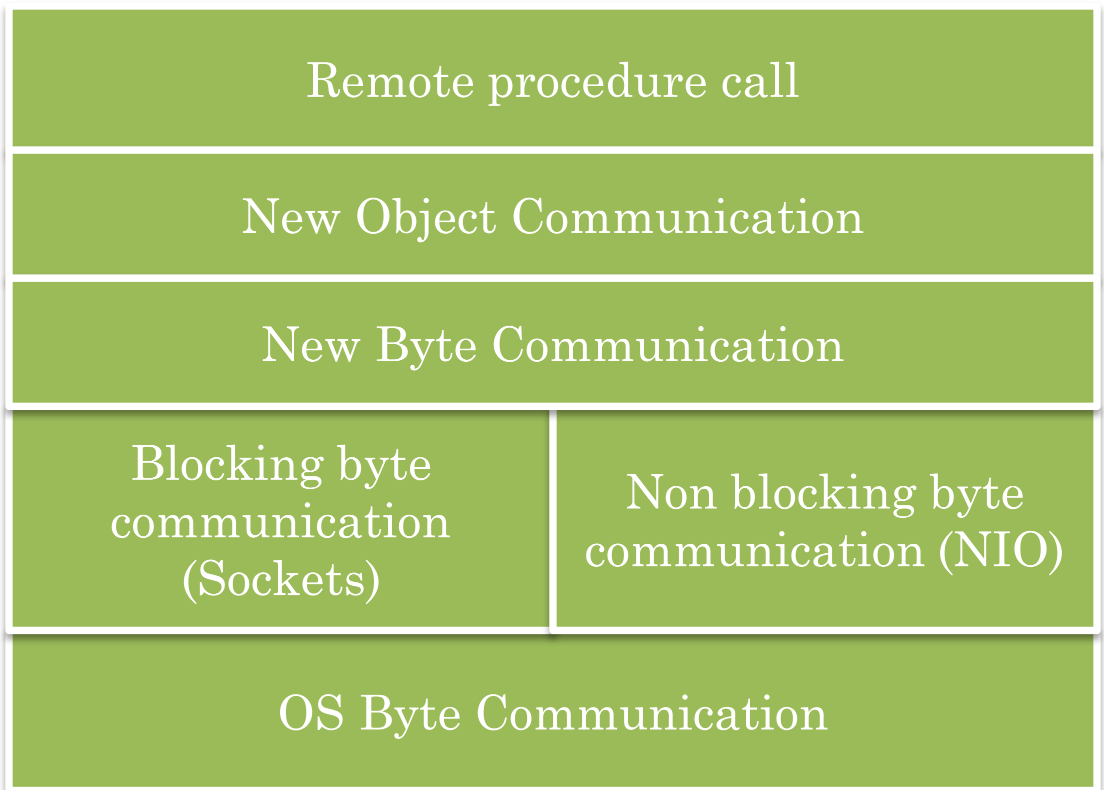
Non blocking byte communication (NIO)

OS Byte Communication

New picture?

- Cannot unite NIO and socket at byte or object level
- Socket communication is low level
- NIO is even lower level
- Programmers rely on usage patterns

Improved Alternative Java Abstractions and Layers



- Alternative high level layer to socket and NIO based byte communication
- Can be bound to either lower level layer
- Design and implementation challenge
- Picture complete? More abstractions and layers?
- Top down vs. bottom up view point

Improved Alternative Java Abstractions and Layers

Link setup and communication: How to create a group of physically/logically connected processes and communicate informing along these links?

Remote procedure

Scalability: How to allow group size to increase without degrading performance?

New Object Communi

New Byte Communi

Distributed fault tolerance: How to recover when one end of the link goes down but the other does not?

Non blocking byte communication (NIO)

Bl

co

Process synchronization: How to block a (thread in a) process until the information it needs to proceed is received from a (thread in a) remote process

OS Byte Communic

No direct support for group setup and communication, scalability, fault tolerance, and any process synchronization

New Abstractions: Design Challenge

Fault Tolerance

Pair-wise Synchronization

Group Synchronization

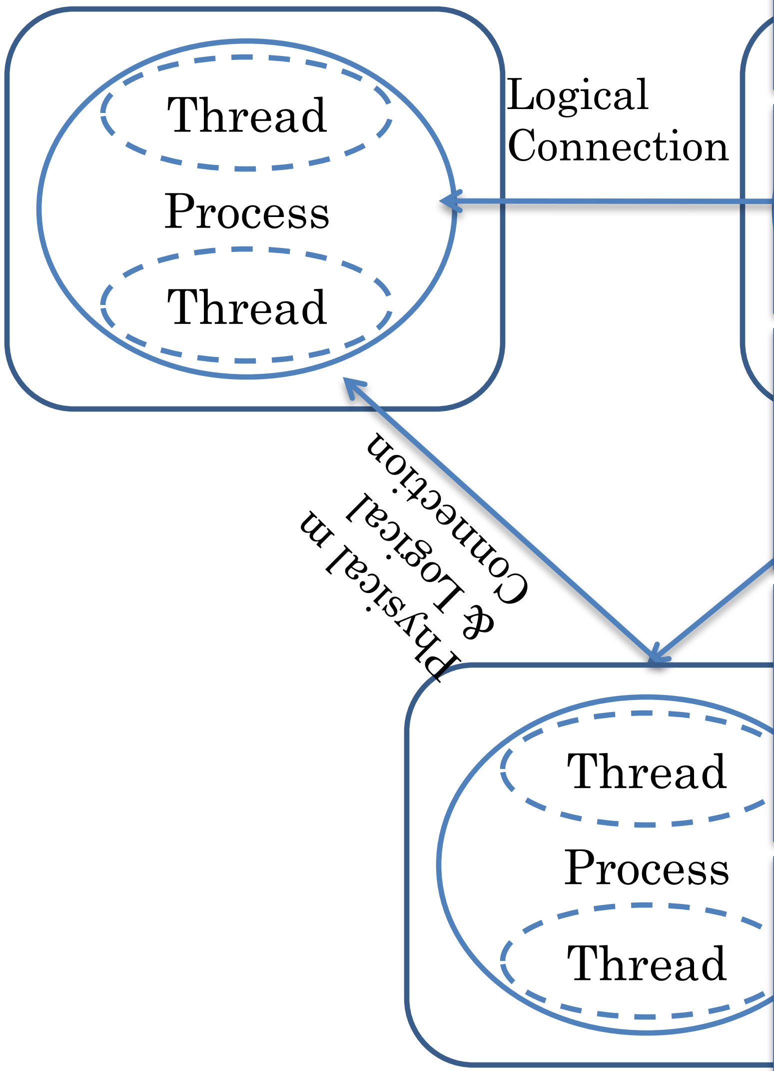
Group Communication and RPC

Scalability

Pair-wise Byte, and Object Communication, Pair-wise RPC

Layering?

Distribution Issues



Link setup and communication: How to create a group of physically/logically connected processes and communicate informing along these links?

Scalability: How to allow group size to increase without degrading performance?

Distributed fault tolerance: How to recover when one end of the link goes down but the other does not?

Process synchronization: How to block a (thread in a) process until the information it needs to proceed is received from a (thread in a) remote process

Thread synchronization: How to block a thread until a condition for proceeding is enabled by a local thread