Aprameyah
Technologies Pvt. Ltd.

CMAD Adv II

Closures

- Closure is a way of implementing private class state in javascript by playing on variable scope

```
function add() {
    var counter=0;
    counter += 1;
    console.log(counter);
}

add();
add();
------------------------------
var add = (function () {
    var counter = 0;
    return function () {
        counter += 1;
        console.log(counter);
        return counter;
    };
})();

add();
add();
```

Closures As Function Factories

```
function notifyUser(email) {
    return function(text) {
        console.log("Sending '"+text+"' to "+email);
    };
}

var notifyWill = notifyUser('wberger@lo.com');
var notifyKelly = notifyUser('kperry@lo.com');

notifyWill("Hello, you have a new join request");
notifyKelly("Hello you are late to work!");
```

Closures As Modules

```
var counter = (function() {
    var privateCounter = 0;
    function changeBy(val) {
        privateCounter += val;
    }
    return {
        increment : function() {
            changeBy(1);
        },
        decrement : function() {
            changeBy(-1);
        },
        value : function() {
            return privateCounter;
        }
    };
})();

console.log(counter.value());
counter.increment();
counter.increment();
console.log(counter.value());
counter.decrement();
console.log(counter.value());
```

Singletons With Closures

```
var mySingleton = (function() {
    var instance;
    function init() {
        function privateMethod() {
            console.log("I am private");
        }
        var privateVariable = "Im also private";
        var privateRandomNumber = Math.random();
        return {
            publicMethod : function() {
                console.log("The public can see me!");
            },
            publicProperty : "I am also public",
            getRandomNumber : function() {
                return privateRandomNumber;
            }
        };
    };

    return {
        getInstance : function() {
            if (!instance) {
                instance = init();
            }
            return instance;
        }
    };
})();
```

Closures & Objects

```
var cars = [{brand: 'Fiat', model:'Punto'},{brand: 'Hyundai',
model:'Santro'},{brand: 'Maruti', model:'Swift'}];
for (var i = 0; i < cars.length; i++) {
    cars[i].printLocation= function() {
        console.log(this.model+" is in location: "+i);
    };
}

cars[0].printLocation();
```

Fix the looping problem

```
for (var i = 0; i < cars.length; i++) {
   cars[i].printLocation = function(x) {
      return function() {
         console.log(this.model + " is in location: " + x);
      };
   }(i);
}

cars[0].printLocation();
```

Lets Try It

- Create a collection class in Javascript with closure to hold the private state. Provide only the add and size methods

Aprameyah Technologies Pvt. Ltd.

Closures

- Closures are a basic part of many languages today: Javascript, Python, Ruby (even java in a way)

- Closures pave the way to creating new language constructs without changing language specs. Consider this:

```
interface OneArg<A>{
    void invoke(A arg);
}

<T> void forEach(Collection<T> c, OneArg<T> block){
    Iterator<T> it = c.iterator();
    while(it.hasNext()){
        block.invoke(it.next());
    }
}

forEach(c, data->{
    System.out.print(data);
    System.out.println("Total size: "+c.size());
});
forEach(c, new OneArg() {
    public void invoke(Object data) {
        System.out.print(data);
        System.out.println("Total size: "+c.size());
    }
});
```

Build A Contextual Try

```
with(con,new OneArg<Connection>() {
    public void invoke(Connection con) {
        //Do something with connection
    }
});
with(con,conection->{
    connection.doSomething()
});

void with(Connection con, OneArg<Connection> block) {
    try {
        block.invoke(con);
    } finally {
        if(con!=null)
            try {
                con.close();
            } catch (SQLException e) {}
    }
}
```

Aprameyah
Technologies Pvt. Ltd.

Lambda Functions

- Typical patterns with anonymous classes and functional interfaces

```
JButton testButton = new JButton("Test Button");
testButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        System.out.println("Click Detected by Anon Class");
    }
});

public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent e);

}
```

Aprameyah
Technologies Pvt. Ltd.

Lambdas

- Try these in the IDE:

```
m = (int x, int y) -> x + y;

n = () -> 42;

o = (String s) -> { System.out.println(s); };
```

- Launch a thread using a runnable lambda
- Sort a list using the comparable lambda

Aprameyah
Technologies Pvt. Ltd.

Functional Interfaces

- java.util.function package has a lot of functional interfaces
  - Predicate: A property of the object passed as argument
  - Consumer: An action to be performed with the object passed as argument
  - Function: Transform a T to a U
  - Supplier: Provide an instance of a T (such as a factory)
  - UnaryOperator: A unary operator from T -> T
  - BinaryOperator: A binary operator from (T, T) -> T

Aprameyah
Technologies Pvt. Ltd.

Scope

- Lambda functions dont introduce another scoped class.

- this in lambda function refers to the class in which it is used

- Lambda function has access to local variables and methods of the calling class

Method References

- You use lambda expressions to create anonymous methods.

- Sometimes, however, a lambda expression does nothing but call an existing method.

- In those cases, refer to the existing method by name.

```
Arrays.sort(rosterAsArray, myComparisonProvider::compareByName);
Arrays.sort(stringArray, String::compareToIgnoreCase);
Set<Person> rosterSet = transferElements(roster, HashSet::new);
```

Aprameyah
Technologies Pvt. Ltd.

Generics

- Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods
- Stronger type checks at compile time
  - Fixing compile-time errors is easier than fixing runtime errors

Aprameyah
Technologies Pvt. Ltd.

Elimination Of Casts

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);

List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   // no cast
```

Aprameyah
Technologies Pvt. Ltd.

Generic Class

```
public class Box {
    private Object object;
    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}
```

```
public class Box<T> {
    private T t;
    public void set(T t) { this.t = t; }
    public T get() { return t; }
}

Box<Integer> integerBox = new Box<Integer>();
```

Aprameyah
Technologies Pvt. Ltd.

Generic Methods

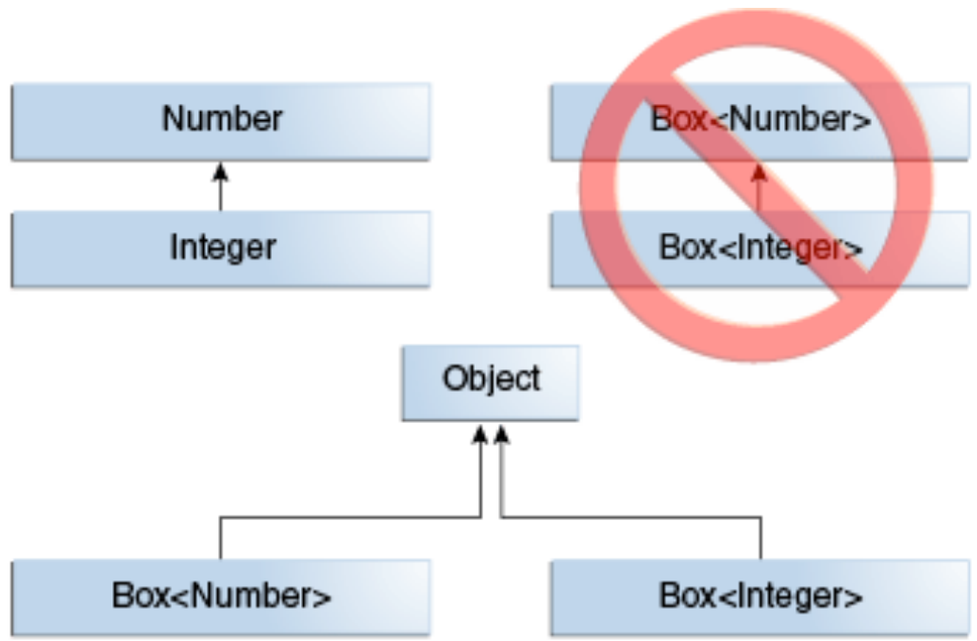
```
public static <K, V> boolean join(K p1, V p2) {
    return p1.toString() +" "+ p2.toString();
}

Util.<Integer, String>join(p1, p2);
```

Aprameyah
Technologies Pvt. Ltd.

Inheritance With Generics

- Generics are implemented at the compiler level using erasure
  - Runtime system does not know about generics at all
  - Hence…
- Instanceof wont work with generics



Aprameyah
Technologies Pvt. Ltd.

Effect Of Erasure

```
List<Integer> mn = new ArrayList<>();
mn.add(5);
Collection n = mn;// A raw type - compiler throws an unchecked warning
n.add("Hello");
Integer x = mn.get(1); //Problem! Causes a ClassCastException to be thrown.
```

Aprameyah Technologies Pvt. Ltd.

Restrictions

- Cant use primitive type params
  - List<int> k;
- Cannot Create Instances of Type Parameters
  - public static <E> void append(List<E> list) {
  - E elem = new E(); // compile-time error
- Cannot Declare Static Fields Whose Types are Type Parameters
  - public class MobileDevice<T> {
  - private static T os;
- Cannot Use Casts or instanceof with Parameterized Types
  - if (list instanceof ArrayList<Integer>) { // compile-time error
- Cannot Create Arrays of Parameterized Types
  - List<Integer>[] arrayOfLists = new List<Integer>[2];
- Cannot Create, Catch, or Throw Objects of Parameterized Types
  - catch (T e) {
- Cannot overload
  - public void print(Set<String> strSet) { }
  - public void print(Set<Integer> intSet) { }

Promises

- Promises are objects that have a “then” method
- Unlike node functions, which take a single callback, the then method of a promise can take two callbacks: a success callback and an error callback.
- When one of these two callbacks returns a value or throws an exception, then must behave in a way that enables stream-like chaining and simplified error handling.

Without Promises

```
function doSomething(callBack) {
    setTimeout(function() {
        console.log("2 seconds up");
        callBack(null, "2 seconds up");
    }, 2000);
}

function doSomethingElse(callback) {
    setTimeout(function() {
        console.log("2 more seconds up");
        callback(null, "2 more seconds up");
    }, 2000);
}

doSomething(function() {
    doSomethingElse(function() {
        console.log("Should be 4 seconds now!");
    });
});
```

With Promises

```
var Promise = require("bluebird");
function doSomethingWithPromise() {
     return new Promise(function(resolve, reject) {
          setTimeout(function() {
               console.log("2 seconds up");
               resolve("2 seconds up”);
          },2000);
     });
}

function doSomethingElseWithPromise() {
     return new Promise(function(resolve, reject) {
          setTimeout(function() {
               console.log("2 more seconds up");
               resolve(null, "2 more seconds up");
          }, 2000);
     });
}

doSomethingWithPromise()
.then(function() {
     return doSomethingElseWithPromise();
})
.then(function() {
     console.log("Should be 4 seconds now!");
});
```

Promisification

```
var doSomethingWithPromise = Promise.promisify(doSomething);
var doSomethingElseWithPromise = Promise.promisify(doSomethingElse);
```

```
doSomethingWithPromise()
.then(function() {
    return doSomethingElseWithPromise();
}).then(function() {
    console.log("Should be 4 seconds now!");
});
```

Changes With Promises

```
//Regular Node code
fs.readFile(file, function(err, res) {
    if (err) handleError();
    doStuffWith(res);
});

//With promises
fs.readFile(file).then(function(res) {
    doStuffWith(res);
}, function(err) {
    handleError();
});
```

Promises…

- Promises are kept in future too

```
var filePromise = fs.readFile(file);
//do more stuff... and then
filePromise.then(function(res) { //process data in res
});
```

- Promises are good for multiple things

```
filePromise.then(function(res) { uploadData(url, res); });
filePromise.then(function(res) { saveLocal(url, res); });
```

- Returns from promises become promises

```
var firstLinePromise = filePromise.then(function(data) {
    return data.toString().split('\n')[0];
});

var beginsWithHelloPromise = firstLinePromise.then(function(line) {
    return /^hello/.test(line);
});
```

Error Handling With Promises

```
function readProcessAndSave(inPath, outPath) {
    var filePromise = fs.readFile(inPath);
    var transformedPromise = filePromise.then(function(content) {
        return service.transform(content);
    });
    var writeFilePromise =
            transformedPromise.then(function(transformed) {
        return fs.writeFile(otherPath, transformed)
    });
    return writeFilePromise;
}
readProcessAndSave(file, url, otherPath).then(function() {
    console.log("Success!");
}, function(err) {
    console.log("Error: ", err);
});
```

Promisify

- Bluebird.js is a promise library that allows for converting regular node libraries to support promises

- For this to work, node libraries should follow the convention of callback(err,successVal)

```
Promise.promisifyAll(CommandProcessorAsyncE.prototype);

//Method will be called in context of cpasyncce
var cp = Promise.promisify(cpasynce.processCommand, cpasynce);

var fs = require("fs");
Promise.promisifyAll(fs);
```

Chaining And Error Handling

```
return fs.readFileAsync(inPath)
      .then(service.transform)
      .then(function(transformed) {
         return fs.writeFileAsync(otherPath, transformed)
}).caught(function(e){
   console.log("Error in operations");
   console.log(e.stack);
}).lastly(function(){
   console.log("Executed finally");
});
```

Joins

- We can co-ordinate the result of multiple promises using join. here is an example:

```
function doSomething(callBack) {
    setTimeout(function() {
        console.log("2 seconds up");
        callBack(null, "2 seconds up");
    }, 2000);
}
function doSomethingElse(callback) {
    setTimeout(function() {
        console.log("5 seconds up");
        callback(null, "5 seconds up");
    }, 5000);
}

var doSomethingWithPromise = Promise.promisify(doSomething);
var doSomethingElseWithPromise = Promise.promisify(doSomethingElse);

Promise.join(doSomethingWithPromise(), doSomethingElseWithPromise(), function(val1,
val2){
    console.log("Joined: "+val1);
    console.log("Joined: "+val2);
});
```

Print all files in directory /Users/folder1 and /Users/folder2 together using our command processor

Map

- Map allows an array to be transformed to another array using a function that can transform an individual element using an async promise, using a specific concurrency

```
var fileNames = ["package.json", "package2.json"];
Promise.map(fileNames, function(fileName) {
    return fs.readFileAsync(fileName)
        .then(JSON.parse)
        .caught(SyntaxError, function(e) {
            e.fileName = fileName;
            throw e;
        });
},{concurrency: 1}).then(function(parsedJSONs) {
    console.log(parsedJSONs);
}).caught(SyntaxError, function(e) {
   console.log("Invalid JSON in file " + e.fileName + ": " + e.message);
});
```

Compress files in a directory using this map mechanism

Reduce

- Calls a function with each promise return value to arrive at a cumulative aggregate value.

- Reduction function is called as soon as any promise result is ready (unlike join/all)

```
Promise.reduce(
      [ promise1, promise2 ],
      function(reducedResult, result) {
         return reducedResult + result.value
      }, 0).then(function(reducedResult) {
            console.log("Final reduced result: " + reducedResult);
      });
```

Write a new folder size command in CommandProcessor using the reduction function. Use fs.readAsync method

Filter

```
Promise.filter([fs.readFileAsync("/Users/maruthir/
ir.sql"),fs.readFileAsync("/Users/maruthir/
maruthi.pem")],function(content){
   console.log("Filter func");
   if(content.length>5000)
      return false;
   else
      return true;
}).then(function(args){
   console.log(args);
});
```

Modify the list command to accept a file pattern and list files only with that pattern

Functional Reactive Programming

- Functional reactive programming (FRP) is a programming paradigm for reactive programming (asynchronous dataflow programming) using the building blocks of functional programming (e.g. map, reduce, filter)

Aprameyah
Technologies Pvt. Ltd.

Building Blocks

- Map
  - Transform operations that change an object
- Filter
  - Decision operation that returns boolean to determine if the object needs to be in the output collection
- Reduce
  - Aggregation operation that works on a stream of data and produces a single property

Where is this useful

- When there are too many events from diverse sources manipulating too many state fields

- We bundle events into different pipes. And attach functional programs to these pipes to perform actions.

- Events evolve with map, filter and reduce

Event Stream Concept

- First concept of FRP is assemble happenings in the system as a stream of events
    - Events can come from a promise (A single event and end of stream): Bacon.fromPromise(promise)
    - From Node.js event emitters: Bacon.fromEventTarget(eventEmitter,eventName)
    - Single event from a function that takes a callback: Bacon.fromNodeCallback(f)
    - Bacon.fromPoll(interval, f): f should return Bacon.next or Bacon.end. f is called in intervals

On the Browser

- Events can also come from jquery bound controls

```
$("input[name='loginname']").asEventStream("keyup")
```

- Or from an array of data

```
return Bacon.fromArray(data.data);
```

- Once we create a stream, we can listen to the stream using onValue function

```
myStream.onValue(function(streamData) {
    console.log(streamData);
});
```

Aprameyah
Technologies Pvt. Ltd.

Lets Try It

- Create an input box, listen to keyup events and log them to console

Aprameyah
Technologies Pvt. Ltd.

Transforming Events

- Map functions are used to transform events on streams

- stream.map(function(data){ return transformedData})

Aprameyah
Technologies Pvt. Ltd.

Lets Try It

- Transform the event objects coming out of the keyup stream to the text value contained in the event and see how the listening on the stream goes

```
var str = $(event.target).val()
```

- In the value listener, update a div next to the user input box to indicate if the username is valid or not (yes/not)
    - Transform the text value to Yes/No values depending on if the text is greater than 8 chars or not

Properties

- Properties are very similar to streams except that they have a current value & initial value.

- Properties are result of “reduce” operations on some stream

- property.sample(interval) - get current value at certain intervals

- property.sampledBy(stream) - get current value of property every time there is an event on the stream

Aprameyah
Technologies Pvt. Ltd.

Lets Try It

- Convert the user stream to a property by calling - toProperty(“No”)

- This should provide an initial value to the status display

Combining Streams With Properties

- A value from a stream can be combined with a value from a property to arrive at a new stream

- stream.combine(property, function(a,b){ return a+b;})

- Create another text field for email address input and create a yes/no property for it after validating it has the @ character in it.

    - Combine this property with the username yes/no property to arrive at a joint validity decision property

Aprameyah
Technologies Pvt. Ltd.

Streams from Ajax

- JQuery ajax calls return deferred objects

```
$.ajax({ url : "isUserValid.txt"}).done(function(result)
{ console.log(result)});
```

- This deferred object can be part of a stream

- Lets check if the user is valid on every keyup event. To the earlier created username property, create a map function that returns ajax responses.

    - Base the “validity of field” decision on the Ajax response being true or false

Aprameyah
Technologies Pvt. Ltd.

FlatMaps

- flatMap is a function that creates a stream for every event in the incoming stream

- stream1.flatMap(function(event){ return subStream});

- Output is a single stream that has all elements of the sub-streams flattened

Aprameyah
Technologies Pvt. Ltd.

Lets Try It

- Load the players from players.json provided in the server.
    - Convert an ajax request into a stream
      Bacon.fromPromise($.ajax({ url : “players.json"}));
    - Call flatMap on resultant stream to make the data a stream Bacon.fromArray(data.data);
    - On the stream value handler, append the user to a div

Aprameyah
Technologies Pvt. Ltd.

Filter Operation

- Streams can be filtered to eliminate the events not needed
- stream.filter(function(eventData){ return boolean})
- Modify the previous user listing to eliminate retired players from the display.

Aprameyah
Technologies Pvt. Ltd.

Reduce Operation

- Scan function is used to perform reduce operation based on previous stream values:
  scan(initialValue,function(prev,current){return accumulate(prev and current);});

- Display a count of players by doing a scan operation on the filtered stream

Manipulate Streams - Map, Filter, Reduce

- Methods on streams: .onValue, .onError, .onEnd,
- .map(function(value){}) - Converts events in this stream using the provided function
- .map(property) - Puts property into stream for every event on stream
- .filter(property)

Aprameyah
Technologies Pvt. Ltd.

Micro Services

- Small (100? 1000? 5000? LOC)
- UI/ Without UI?
- Lightweight - run several per box
- Independent development/deployment
- Stateless - everything in DB. Can use chaos monkey
- Monitored

Aprameyah
Technologies Pvt. Ltd.

Conveys Law

- organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

- Conway's law is a valid sociological observation.

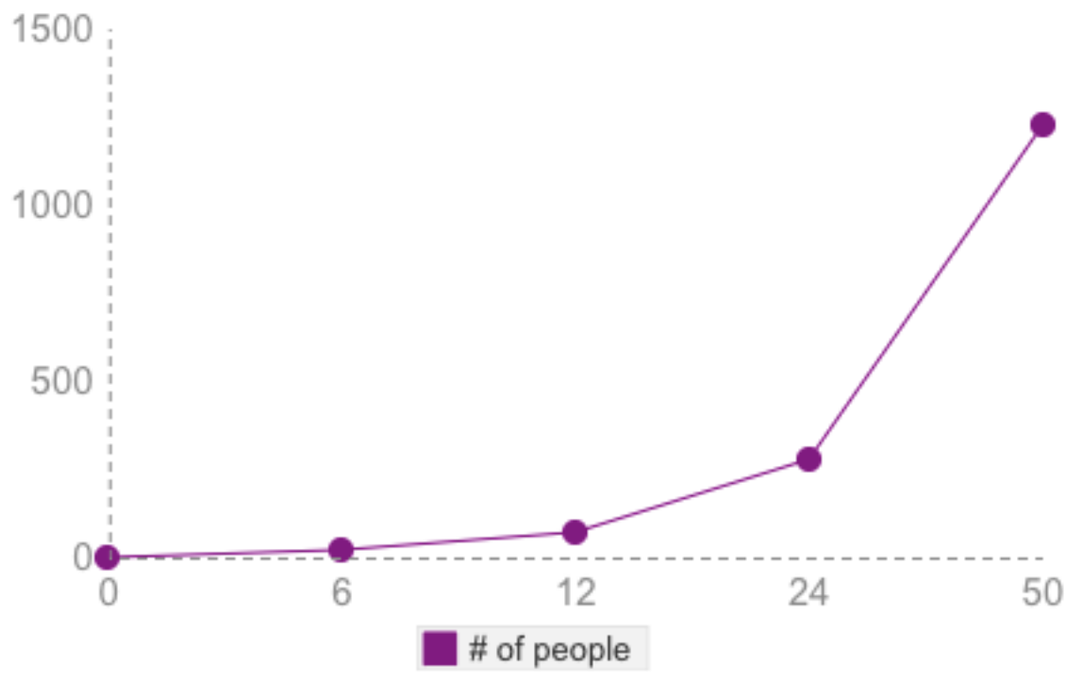
- For separate software modules to interface correctly, the designers and implementers of each module must communicate with each other as loosely as tightly as desired by the modules themselves.

Two Pizza Teams

- Amazon founder and CEO Jeff Bezos created the concept that teams shouldn't be larger than what two pizzas can feed

- Its like a dinner gathering. If u have to order more than two large pizzas, u r forming clusters of conversations

- n(n-1)/2 is the number of connections in team



Aprameyah
Technologies Pvt. Ltd.

Team Scaling Fallacy

- This is the tendency for people “to increasingly underestimate task completion time as team size grows,”

- One warton university research (http://www.opim.wharton.upenn.edu/~kmilkman/2012_OBHDPb.pdf) shows that when tasked to build the same Lego figure, two-person teams took 36 minutes while four-person teams took 52 minutes to finish — over 44% longer.

Aprameyah
Technologies Pvt. Ltd.

Back to Micro Sevices

- 1 micro service = 1 team

- 1 team = may be >1 micro service

- So a micro service cant be anything thats larger than what can be built with 5 to 6 people and deployable in a week or two week's time

Aprameyah
Technologies Pvt. Ltd.

YOU build, YOU run

- Micro services is an organizational change.
- The team that builds it also runs it on production with little infrastructure help.
- No passing the buck, no separate admin/dev support mess and since you feel the heat, you will also be more responsible!

Aprameyah
Technologies Pvt. Ltd.

Big Debate

- Should it have an UI or not?
- If it does not have UI, how is it different from SOA
- How do we handle sandboxing on a browser page?
  - Javascript sandboxing, CSS enclosures.
  - Angular JS concept of apps, controllers and services demo.

Aprameyah
Technologies Pvt. Ltd.

Netflix Case Study

- 2008 - large data center with a big .war file!
- 2012 - Amazon cloud with micro services
    - Amazon prime is a competitor, still they decided to use Amazon cloud. Adrian Cockcroft says, lets at the least have the same advantage that Amazon will get.

Adrian's recipe for Micro Services

- you can update the services independently; updating one service doesn't require changing any other services
    - If you have a bunch of small, specialized services but still have to update them together, they're not microservices because they're not **loosely coupled**.
    - Cant even talk to same db. You need to split the database up and denormalize it.
        - You need to add a tool that performs master data management (MDM) by operating in the background to find and fix inconsistencies.

Aprameyah
Technologies Pvt. Ltd.

Bounded Context

- bounded contexts comes from the book Domain Driven Design by Eric Evans. A microservice with correctly bounded context is self-contained for the purposes of software development.

- microservices and its peers interact strictly through APIs and so don’t share data structures, database schemata, or other internal representations of objects.

Aprameyah
Technologies Pvt. Ltd.

Immutable Infrastructure Principle

- If you need to add or rewrite some of the code in a deployed microservice that's working well, the best approach is usually to create a new microservice version for the new or changed code

- In Cockcroft's experience it is much more common to realize you should split up a microservice because it's gotten too big.

Aprameyah
Technologies Pvt. Ltd.

Building Micro Services

- Do a separate build for each microservice, so that it can pull in component files from the repository at the revision levels appropriate to it.

- Deploy in a standard type of container such as docker so the service deployment process is standard across the organization

- Servers are stateless “cattle” instead of being “pets”

Aprameyah
Technologies Pvt. Ltd.

Pros

- Isolation brings better availability
- Independent delivery cycle for each service
- Decentralised Governance

Aprameyah
Technologies Pvt. Ltd.

Cons

- Complexity of distributed systems
- Operational overhead in devops
- Service versioning and testing issues
- Needs load balanced service discovery for critical services

Aprameyah
Technologies Pvt. Ltd.

Failures

- If the probability of an app being up is 99.99%
  - Breaking it down into 30 services with an uptime of each 99.99% gives us a system thats up only 99.97%

- This is one of the reasons microservices have to be fully contained independently degradable end to end functionalities.
  - Not parts of the system that all need to work for the full system to be up.
  - This way individual failures break part of the system only.

Aprameyah
Technologies Pvt. Ltd.

Failures

- For critical services we need load balanced discovery
- Netflix created and open sourced ribbon
  - Ribbon is a client side IPC library that is battle-tested in cloud. It provides the following features
    - Load balancing
    - Fault tolerance
    - Multiple protocol (HTTP, TCP, UDP) support in an asynchronous and reactive model
    - Caching and batching

Aprameyah
Technologies Pvt. Ltd.

Distributed Systems

- What is distributed?
- Why it is distributed?
- What are the problems?
- How do we solve them?
- What are the tradeoffs?

Aprameyah
Technologies Pvt. Ltd.

What is distributed?

- Computation - least troublesome
- Storage
- Replication
- Shrading
- A combination

Aprameyah
Technologies Pvt. Ltd.

Why is it distributed

- Computation distribution is to support compute loads
- Also to add fault tolerance
- Stateless compute logic is easy to distribute and scale

Aprameyah
Technologies Pvt. Ltd.

Why is it distributed

- Storage distribution is to do one of the following:
  - Improve load capacity
  - Improve redundancy - backups are no longer possible on web scale
  - Improve concurrency

Aprameyah
Technologies Pvt. Ltd.

What are the problems

- Compute Systems
  - Ordering of events
  - Session failovers in stateful compute distribution
  - Distributed service discovery
  - Cluster wide rippling deployment

Aprameyah
Technologies Pvt. Ltd.

What are the problems

- Storage Distribution
  - Consensus
  - CAP problems
  - Shrad location
  - Cross shrad queries
  - Write performance
  - Read performance

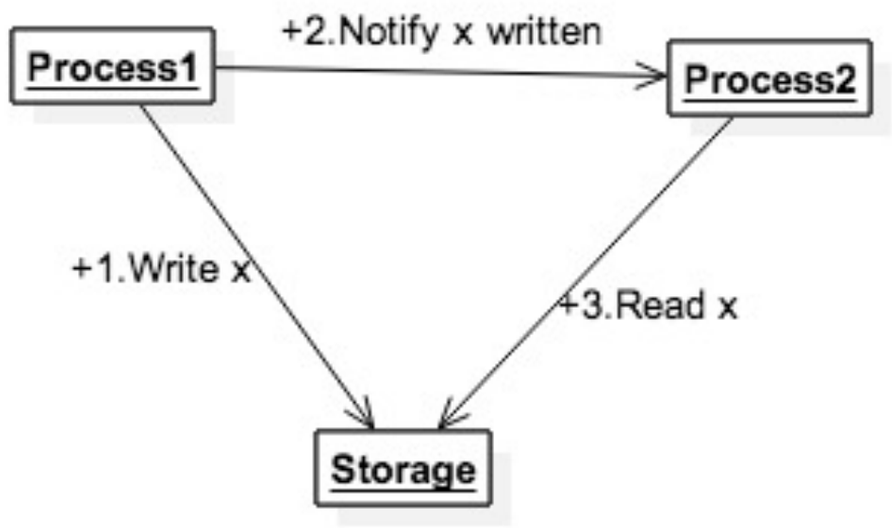
Aprameyah
Technologies Pvt. Ltd.

What are the solutions

- Consensus: Protocols like paxos and raft
  - http://thesecretlivesofdata.com/raft/

Aprameyah
Technologies Pvt. Ltd.

Event Ordering Problem



- Event 1 and 3 can arrive at storage node out or order or at the same time

Event ordering

- Lamport clocks
- The algorithm of Lamport timestamps is a simple algorithm used to determine the order of events in a distributed system.
- As different nodes or processes will typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead - Leslie Lamport.

Lamport Clocks

- A process increments its counter before each event in that process;

- When a process sends a message, it includes its counter value with the message;

- On receiving a message, the receiver process sets its counter to be the maximum of the message counter and its own counter incremented, before it considers the message received.

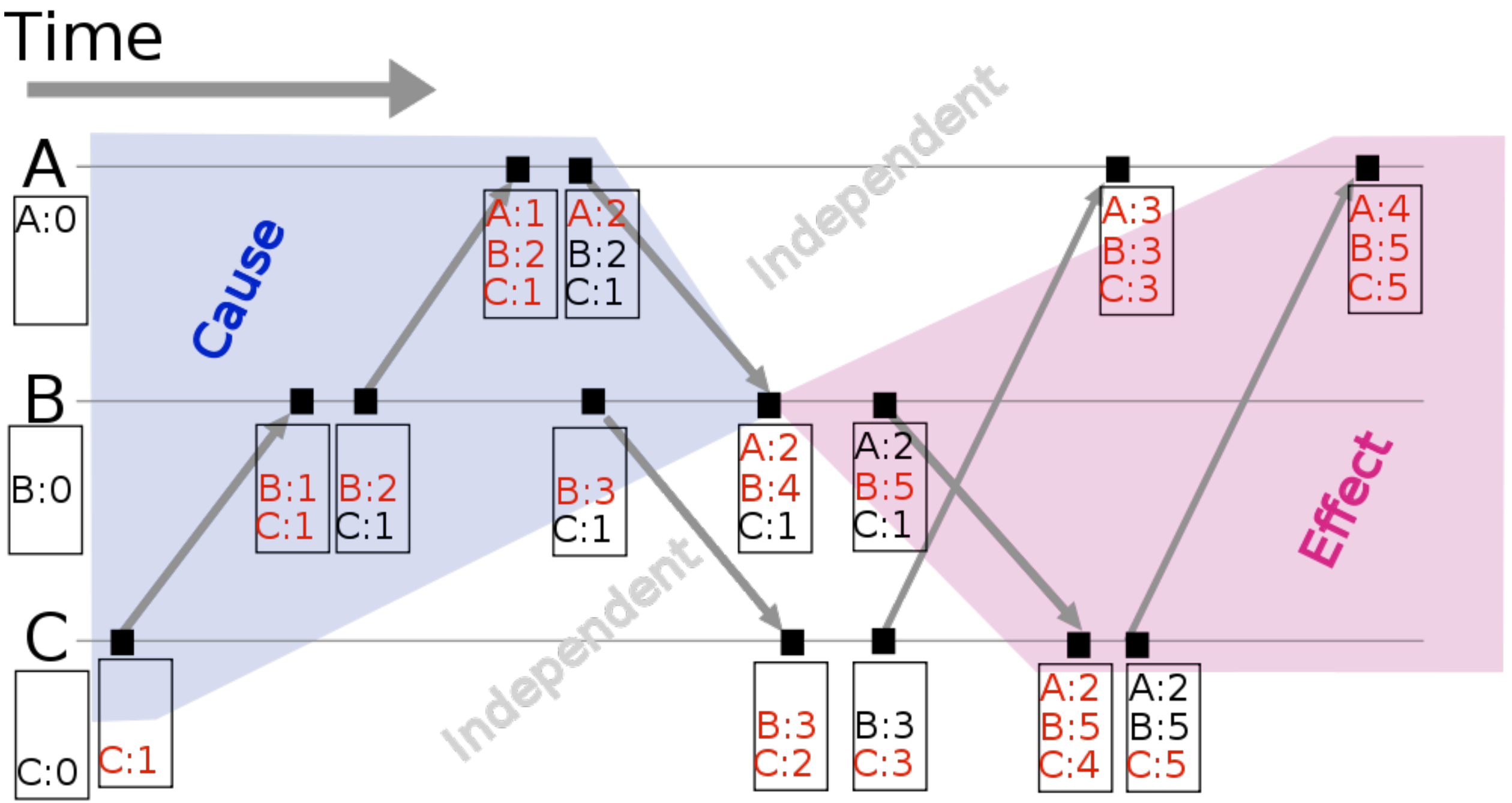
Aprameyah
Technologies Pvt. Ltd.

Lamport Clock Example

- Count on process1:
  - A1->A2 -> [write x A2]->A3 [notify x written A3]
- Count on process2:
  - B1->B2 -> [receive notify x written A3] ->B3 -> [read x B3]
- Count on process3:
  - C1 ->C2 -> [receive read x B3] -> C3 -> [write x **A2]** - **C4**

Aprameyah
Technologies Pvt. Ltd.

Vector Clocks



Aprameyah
Technologies Pvt. Ltd.

ZooKeeper

- ZooKeeper is a hierarchical data store thats
    - replicated for availability
        - replicated over a sets of hosts called an ensemble
        - The servers that make up the ZooKeeper service must all know about each other. They maintain an in-memory image of state, along with a transaction logs and snapshots in a persistent store. As long as a majority of the servers are available, the ZooKeeper service will be available.
        - Clients connect to a single ZooKeeper server. The client maintains a TCP connection through which it sends requests, gets responses, gets watch events, and sends heart beats. If the TCP connection to the server breaks, the client will connect to a different server.
- Guarantees order of updates
    - ZooKeeper stamps each update with a number that reflects the order of all ZooKeeper transactions. Subsequent operations can use the order to implement higher-level abstractions, such as synchronization primitives.
- Hi performance memory storage
    - It is especially fast in "read-dominant" workloads. ZooKeeper applications run on thousands of machines, and it performs best where reads are more common than writes, at ratios of around 10:1.