UNIVERSITY OF PLYMOUTH
School of Engineering, Computing & Mathematics

Lecture 9: Beyond Client-Server & DevOps

Full-Stack Development

Mark Dixon

School of Engineering, Computing and Mathematics

Last Session

- What did we do last week (write down topics, what you can remember)?
  - Did you learn anything (was anything particularly useful or good)?
  - How did the lab session go (were you able to get something running)?
  - What could be better?

Introduction

Today's topics

1. What's beyond client-server?
2. WebSockets
3. Peer-to-peer communications
4. DevOps
5. Analytics

Session learning outcomes – **by the end of today's lecture you will be able to:**

- Use WebSockets to support server to client communications
- Describe the principles of peer-to-peer communications
- Describe the usage and benefits of continuous integration and continuous deployment
- Analyse the best way to release new application features to your user-base

What’s beyond client-server?

Ajax

- Beyond request-response-reload

Server-client

- **WebSockets**
- Push notifications

Client-client

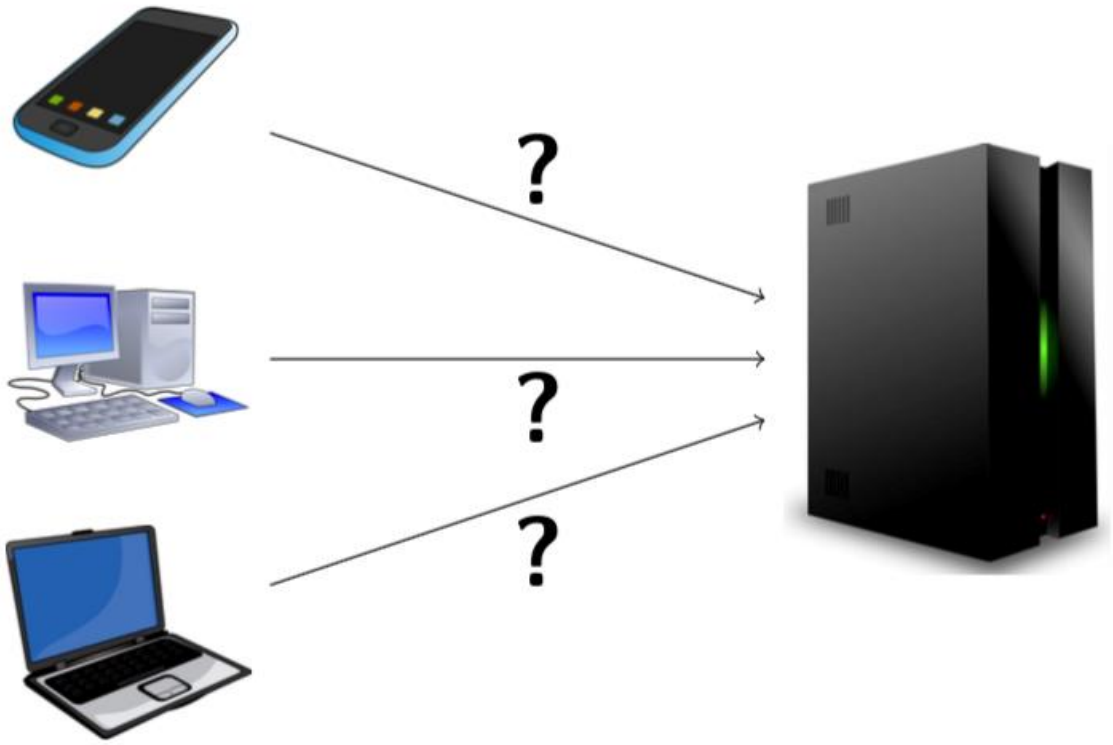
- Peer-to-peer communication

Exercise

Think of some use cases for using something other than client-server

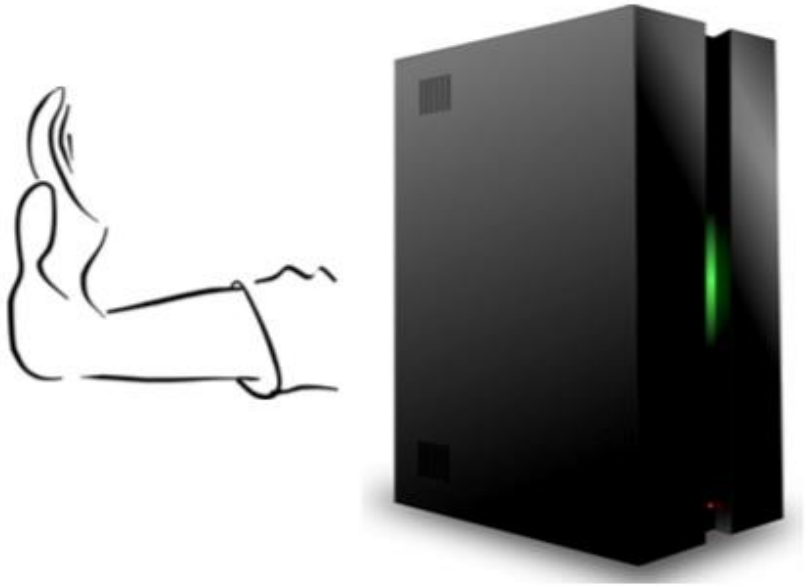
Polling

- Clients periodically query server for update (using Ajax)
- High number - wasted communications
- many clients may overwhelm server



Push notifications

- Server provides updates to the clients without a request
- When something has changed server-side
  - New post by another client
  - Change to the database
- Typically supported by one of two technologies
  - **WebSockets**
  - Google Cloud Messaging



WebSockets

Modelled on traditional TCP/IP sockets

- Two-way send and receive

New URL schemes ws and wss

- URL with port number

Point-to-point communication

- Connection must be established before data can be transferred
- Possess cross-origin privileges

Browsers can only initiate connection, not receive

- No peer-to-peer
- Keep live socket

Socket.io

Set up the socket server

1. Set up an **Express** and **Node** app
2. Use event handlers that **emit messages** and **process responses**

```
let express = require("express");
let path = require("path");
let http = require("http");
let socketIo = require("socket.io");

// Set up the app and server.
let app = express();
let server = http.createServer(app);
```

Is this server-side code or client-side code?

Socket.io

```
// Initialise the socket server.
let io = socketIo(server);

// "On connection" handler.
io.on("connection", function(socket) {
    console.log("A user connected");

    // Send a message to the client.
    socket.emit("server message", "Hello World");

    // Handler for messages from the client.
    socket.on("client message", function(msg) {
        console.log("Rec'd from client: '" + msg + "'");
    });
});

server.listen(9000, () => { console.log("Listening on 9000"); });
```

Socket.io

Having created a server, create a client to link it to

- Within the JavaScript used in the client connect to the server
- Event handlers for received messages
- Emit messages back to the server

```
$(function() {
    let socket = io("http://localhost:9000");
    console.log("socket: " + socket);

    socket.on("server message", function(msg) {
        console.log("Rec'd from server: '" + msg + "'");
        socket.emit("client message", "Acknowledging your message");
        console.log("Emitted message");
    });
});
```

Is this server-side code or client-side code?

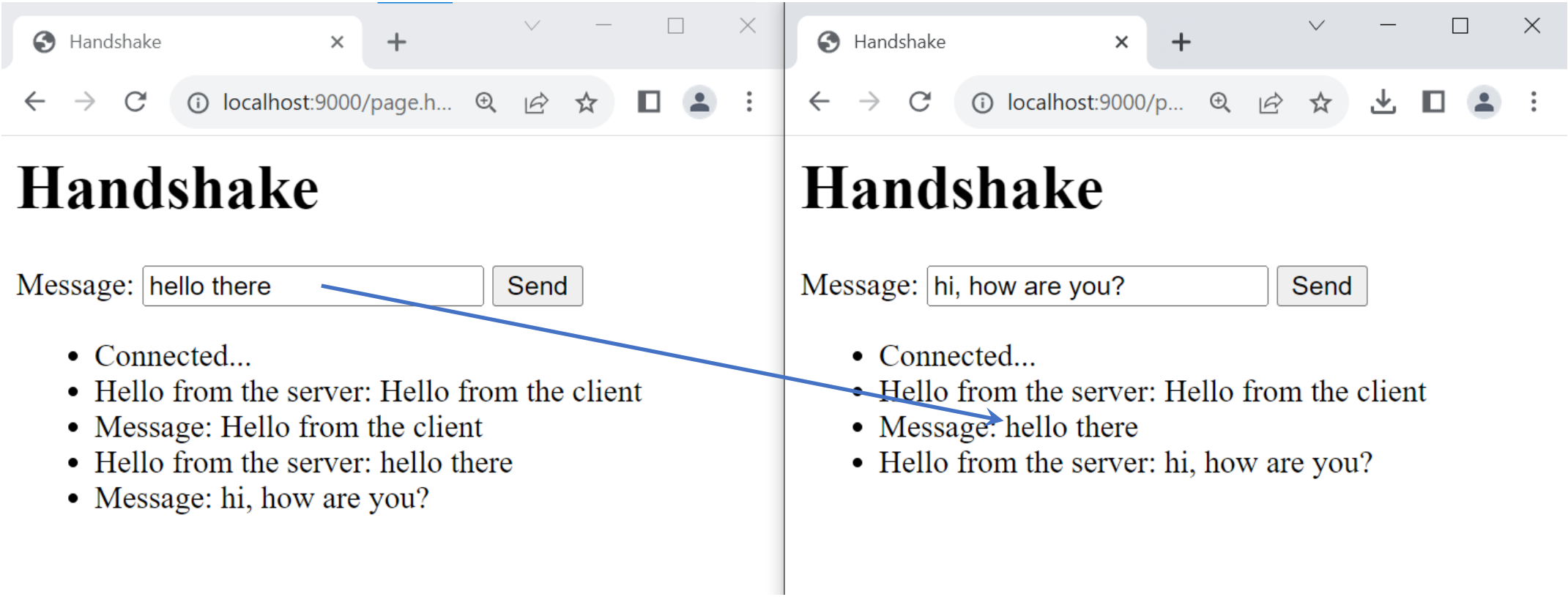
Socket.io - broadcast

- Real power (server-side code):

```
socket.on("request", function(msg) {
    socket.emit("response", "Reply to client who sent request.");
    socket.broadcast.emit("response", "Message to other clients.");
});
```

- emits to all connected clients
  (except the one who sent the request)

Socket.io - demo



Peer-to-peer

There are dedicated web peer-to-peer frameworks

What does WebRTC do?

- Gets hold of local media
  - Streaming audio, video or other data
- Get network information
  - IP addresses and ports, media capabilities
- Exchange this with other WebRTC clients (peers)
  - Using third-party server (signalling)
  - To enable connection, even through NATs and firewalls
- Communicate streaming audio, video or data

Supported through three APIs

MediaStream

- Get access to data streams, such as from the user's camera or microphone
- Synchronised streams of media
  - camera, audio
  - Screen sharing
- Can be connected to video element or RTCPeerConnection

RTCPeerConnection

- Audio or video calling with facilities for encryption and bandwidth management (zoom bombing)

RTCDataChannel

- Peer-to-peer communication of generic data

P2P - Signalling

- RTCPeerConnections need to find endpoint (peer) details
  - Clients who support the ICE framework (published)
- Coordinating P2P communication
  - Exchanging P2P communication (finding candidates)
  - Exchange local and remote audio and video resolution and codec capabilities
- Not part of the RTCPeerConnection API
- Requires separate server initially (signaling channel)
  - e.g., Node with WebWorkers

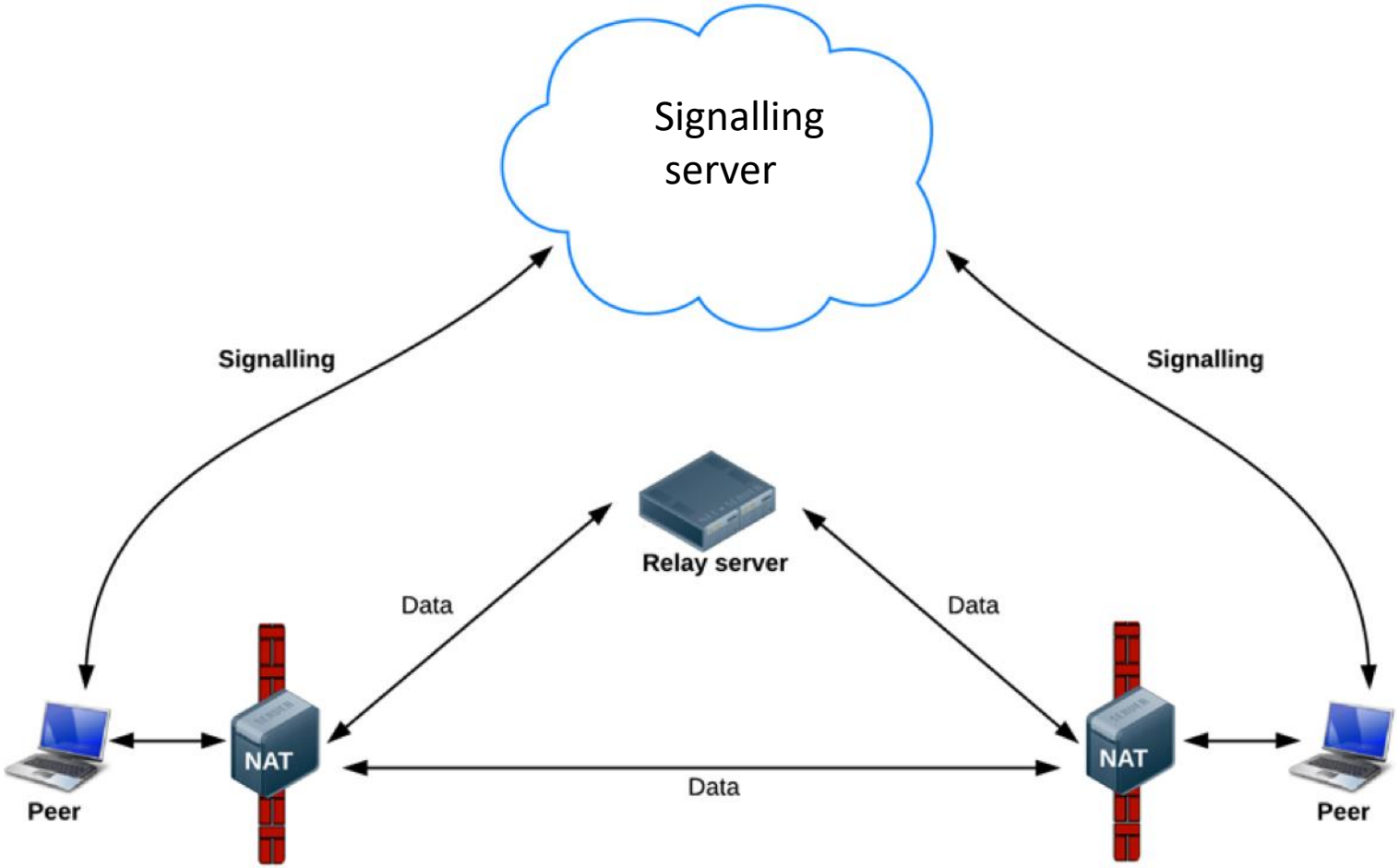
The ICE framework

STUN (signalling server)

- Find direct route

TURN

- Communicate via server



Procedurally – Alice and Bob

- Alice and Bob create **RTCPeerConnection** objects (signalling server)
- Alice calls **createOffer()**
- Alice sets her local descriptions and signals it to Bob
- Bob calls **createAnswer()**
  - Passing it Alice's remote description
  - Callback has **description** argument that matches remote description
- Bob sets his local description and sends it to Alice

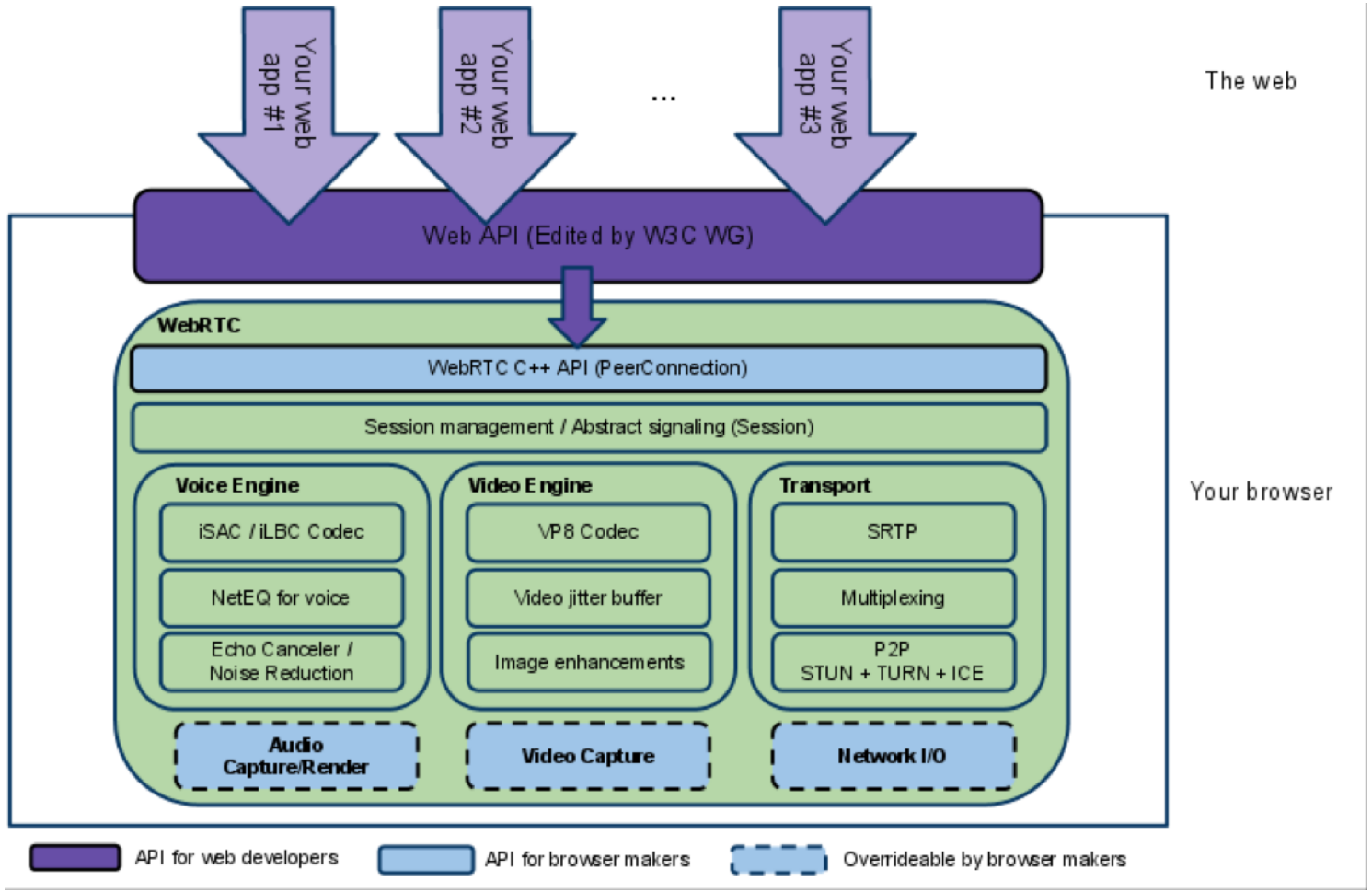
Once the signalling is done streaming can take place

- Between offerer and answerer
- Using **direct communication** or a **TURN server**

RTCPeerConnection

Hides large amounts of complexity

- Packet loss concealment
- Echo cancellation
- Bandwidth adaptivity
- Dynamic jitter buffering
- Noise reduction and suppression
- Image ‘cleaning’



Security

Security threats

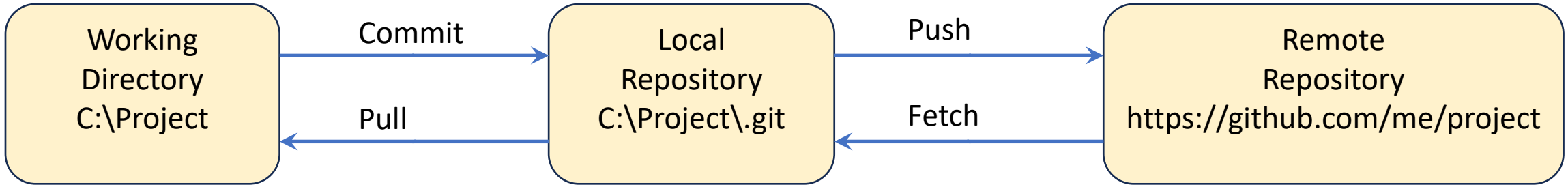
- Unencrypted media or data might be intercepted enroute between browsers
- An application might record or redistribute video/audio without users' knowledge
- Malware or viruses might be installed alongside apparently innocuous plugins or applications

Security features

- Encryption is mandatory for all WebRTC components, including signalling mechanisms
- WebRTC is not a plugin – its components run in the browser sandbox not in a separate process and do not require installation (updated when the browser is)
- Camera and microphone access must be granted explicitly and, when the camera or microphone are running, this is clearly shown by the UI

Break?

Git



- git status – compares working directory with local repo (not remote!)
- git fetch – updates local repo with remote (do this before status)
- git commit – copies changes to local repo only (not remote!)
- git push – copies changes from local repo to remote repo

Different levels of “continuous”

Continuous integration

- Automatically building and testing your software on a regular basis
- Daily builds, build on every commit…

Continuous delivery

- Trust your software due to continuous integration
- Release new version after every commit
- May be more than customers want – always release-ready code

Continuous deployment

- Update your application silently in the background – cloud-based SaaS
- Provide automatic updates – mobile, desktop…

Wider benefits

Customers

- Get software updates faster
- Documented process informs users of changes and allows them to influence the development process

Management

- Progress is immediately visible

Developers

- Removes the “release window” concept, which minimises delays

System administrators

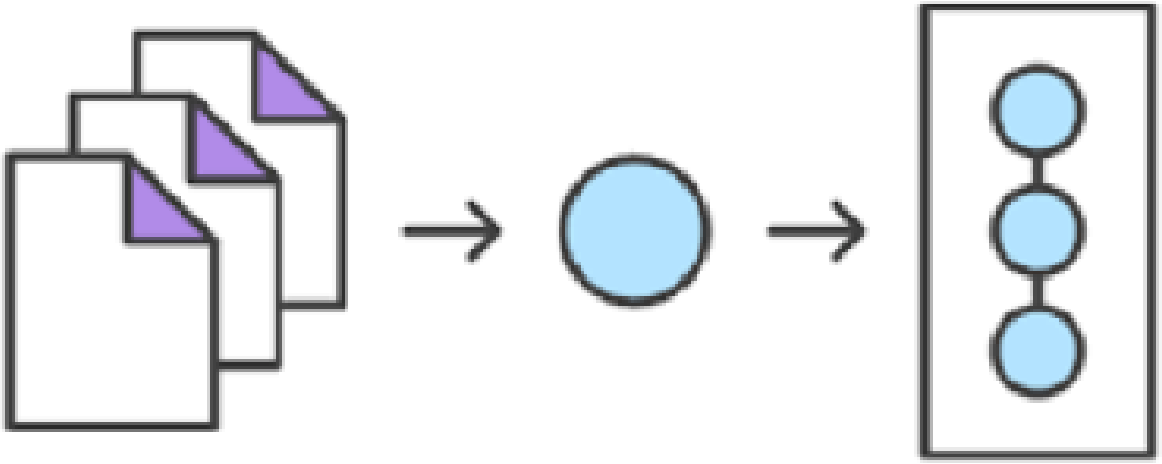
- Freed from deployment work and can focus on deployment analysis

System administrators

- Maintain live system performance
- Analyse performance after each deployment
- Simplify analysis and maintenance
  - Don’t release entangled features
  - Use atomic releases
  - Single-feature roll back

Atomic commits

- Only commit changes related to a single feature
- Collect related changes (files) on a staging area
  - **git add**
- Commit only the changes on the staging area
  - **git commit**



GitHub Actions

- https://docs.github.com/en/actions/quickstart
- .github/workflows folder
- workflow = pipeline
  - Sequence of steps & commands
  - Typically
    - Pull source code from git
    - Build
    - Run tests
    - Run static analysis
    - Deploy

demo-ci / .github / workflows / node.js.yml in main

Edit Preview Code 55% faster with GitHub Copilot

```yaml
# This workflow will do a clean installation of node dependencies, cache/restore them, build the
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/buildi

name: Node.js CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [14.x, 16.x, 18.x]
        # See supported Node.js release schedule at https://nodejs.org/en/about/releases/

    steps:
    - uses: actions/checkout@v3
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v3
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
    - run: npm ci
    - run: npm run build --if-present
    - run: npm test
```

Jenkins

- Difficult to install
- Powerful
- Excellent visual tools

Welcome to Jenkins!

Username

Password

Sign in

Keep me signed in

	build	test: integration-&-quality	test: functional	test: load-&-security	approval	deploy: prod
Average stage times: (Average full run time: ~5s)	836ms	20min 43s	9ms	7ms	89ms	5ms
#17 Sep 22 15:05 No Changes Retry Download	538ms	10s	10ms	8ms	72ms (paused for 7s)	4ms
#16 Sep 22 15:04 No Changes Retry Download	479ms	6s	9ms	9ms	74ms (paused for 6s)	5ms
#15 Sep 22 15:03 No Changes Retry Download	922ms	6s	10ms	9ms failed		
#14 Sep 22 15:03 No Changes Retry Download	1s	8s	12ms	9ms	80ms (paused for 5s)	5ms
#13 Sep 22 15:02 No Changes Download	942ms	9s	13ms failed			
#12 Sep 22 15:02 No Changes Retry Download	1s	6s	13ms	11ms	111ms (paused for 5s) aborted	

Travis

- An automation server
- Watches your Git
- Takes actions on commit
  - Analyse code (such as linting)
  - Compile
  - Run tests
  - Run scenarios
  - Deploy code to live server
- Other automation servers are available

Travis CI

Setup

- Connect your GitHub account to Travis – let Travis access your repos
- Add **travis.yml** to your repo
- Whenever you **git push** Travis will automatically build your project and test it

A Travis Example

djw213 / HelloCI build passing

Current Branches Build History Pull Requests More options

main Added secure key for Heroku.

#13 passed

Restart build

Commit 194127b

Compare 78563d0..194127b

Branch main

David Walker

Ran for 49 sec

2 hours ago

Node.js: node

AMD64

Job log View config

Remove log Raw log

```
1                                                                                  0.07s
                                                                                   1.5s
                                                   docker_mtu_and_registry_mirrors
                                                                   resolvconf
2  $ git clone --depth=50 --branch=main https://github.com/djw213/HelloCI.git djw213/HelloCI   git.checkout   0.47s
6
7                                                                                  0.01s
8  $ nvm install node                                                              nvm.install    4.24s
14
15 Setting up build cache                                                          cache.1
21
22                                                                                 cache.npm
25 $ node --version
26 v15.3.0
```

Top

.travis.yml

Add information about your project and its dependencies

```
language: node_js
node_js:
- node
install:
- npm install -g mocha
- npm install chai
```

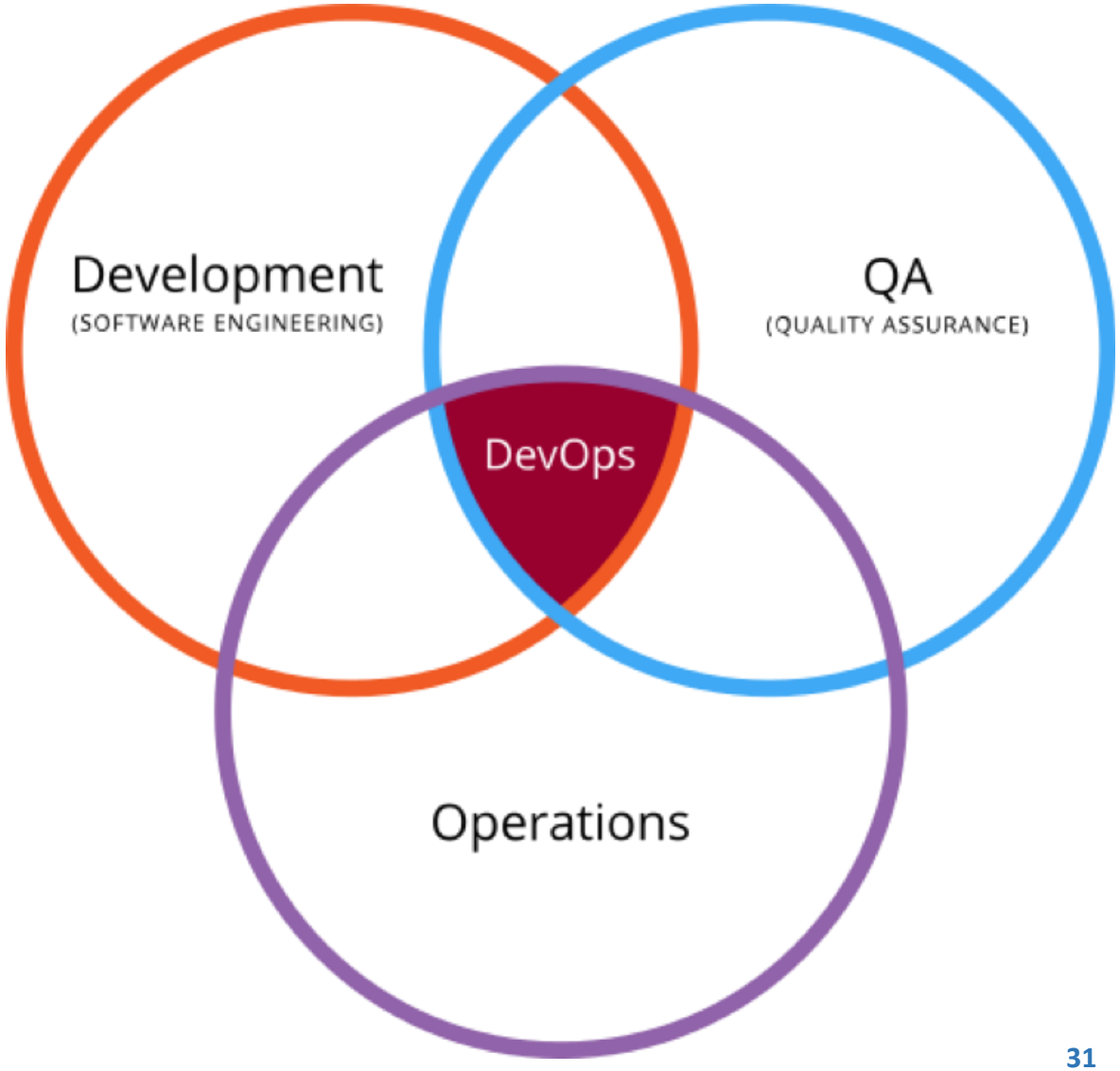
- What language have you used? Specify the version (**node** specifies the latest version of Node)
- Define any prerequisites – include a command to install them

Define test script in package.json

```
"scripts": {
    "test": "mocha -ui tdd test/",
    "start": "node server.js"
},
```

DevOps

- Development and operations
- Collaboration and communication between software developers and other IT professionals



Feature flagging

- Manipulating the availability of features to different users post-deployment
- e.g., a new navigation structure
  - Can be released to just 10% of users (to limit potentially poor user experience)
  - Can be used to collect usage data
  - Well-performing features are rolled out to 100% of users
  - Poorly-performing features can be killed.
- Allow comparative A/B tests
- Give users an opt-in choice

Toggle types

Release toggles

- Transitory toggles for careful deployment

Experiment toggles

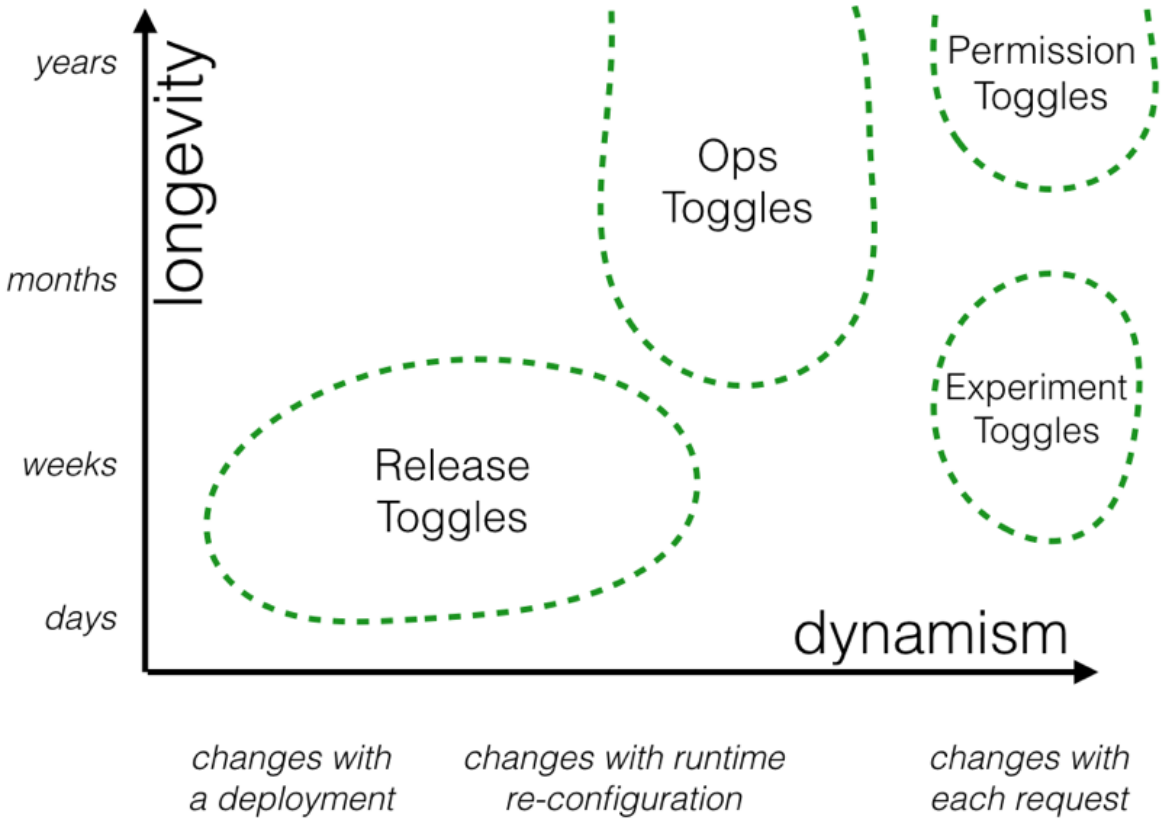
- Short-lived toggles for comparing performance of different features

Ops toggles

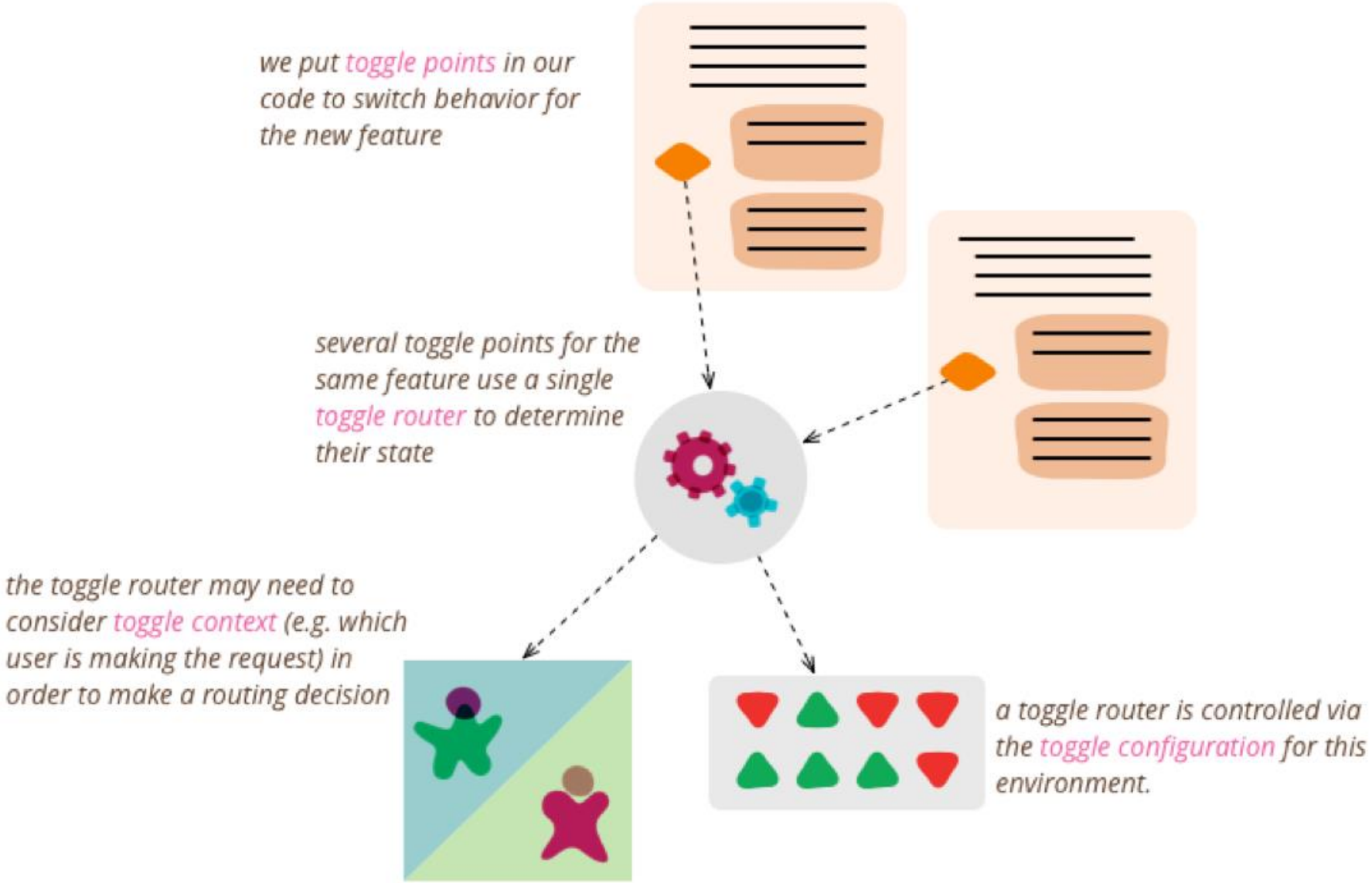
- Long-lived toggles for analysing system performance
- Disabling resource intensive features during high-demand periods

Permission toggles

- Enabling high-value content for premium users



Toggle infrastructure



Google analytics

Understand site users to evaluate content/product performance

- **analytics.js**
  - Sends a **pageview** for each page your users visit
  - Google Analytics processes this data and can infer a great deal of information

Information from Google Analytics

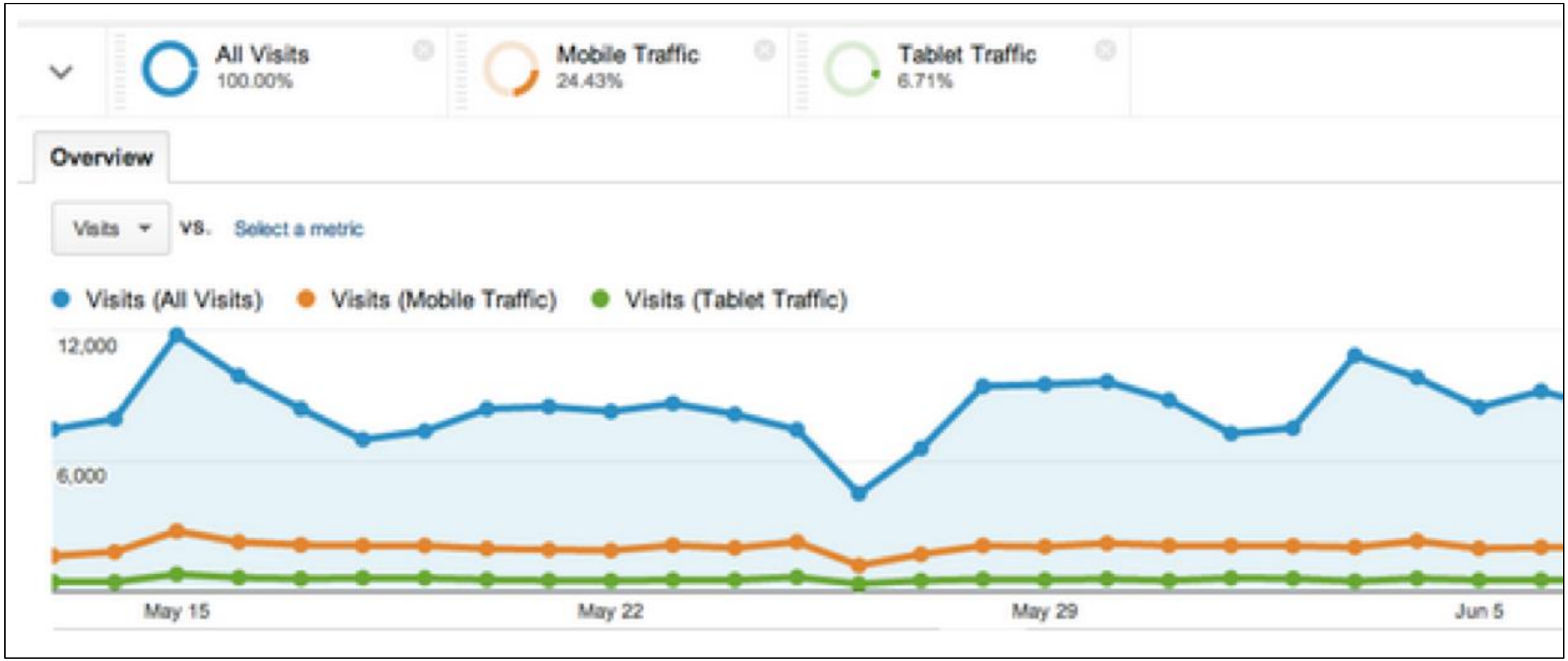
- Information from **pageview** data
  - Total time spent on site
  - Time spent on each page (and order each page visited)
  - Internal links clicked
- Information from IP address, user agent string, initial page inspection
  - Geographic location of a user
  - What browser/OS being used?
  - Screen size
  - Referring site

Exercise

What would you do with each piece of information?

User segmentation

Different user segments have different behaviours



appmetrics.js

A library for defining your own events and reporting to DevTools and Google Analytics

- Name events
- Record time and duration of events

```
// Each metric name should be unique.
let metric = new Metric("my_event");

// Mark name will be "mark_my_event_start".
metric.start();

// Mark name will be "mark_my_event_end".
metric.end();

metric.sendToAnalytics("my_event");
```

Summary

WebSockets

- Server-client communications
- Provide mechanism for (e.g.) notifications and avoids polling
- Use **socket.io**

Peer-to-peer communications

- WebRTC
- Signalling
- ICE framework

DevOps pipeline

- Pipeline automation
- Benefits for all!

Analytics

- How are users using your website or app?