Software Engineering Project

Morgan Ericsson

morgan@cse.gu.se

@morganericsson

morganericsson

UNIVERSITY OF GOTHENBURG

CHALMERS

DON'T PANIC

What is a Version Control System (VCS)?

- Version control (source/revision) is about managing changes to documents (source code files)
- Logical way to organise and control versions
- VCS is an application (or part of) to manage the version control
  - Git, SVN, ...
  - Google Drive, Dropbox, Wikis

Why Use Version Control?

- Software exist in multiple versions
  - also deployed
- To maintain, important to access the right version
- Also, parallel development
  - features and/or fixes

Evolution of VCS

- Stage one, manual VCS
  - basically keep multiple copies of the source code (version)
  - distribute source files between developers, e.g., mail, floppies, etc.
  - track everything manually

Evolution of VCS

- Stage two, local VCS
  - SCCS and RCS
  - local file systems, and local locking

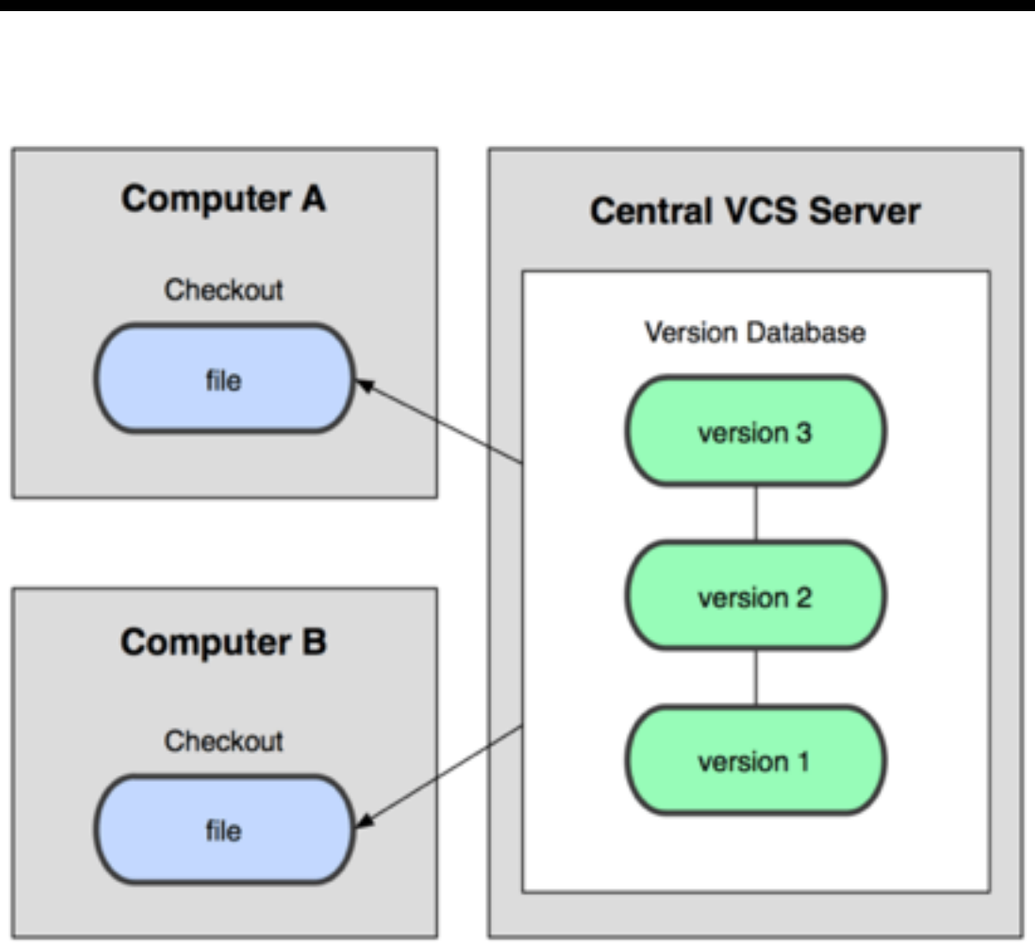
Evolution of VCS

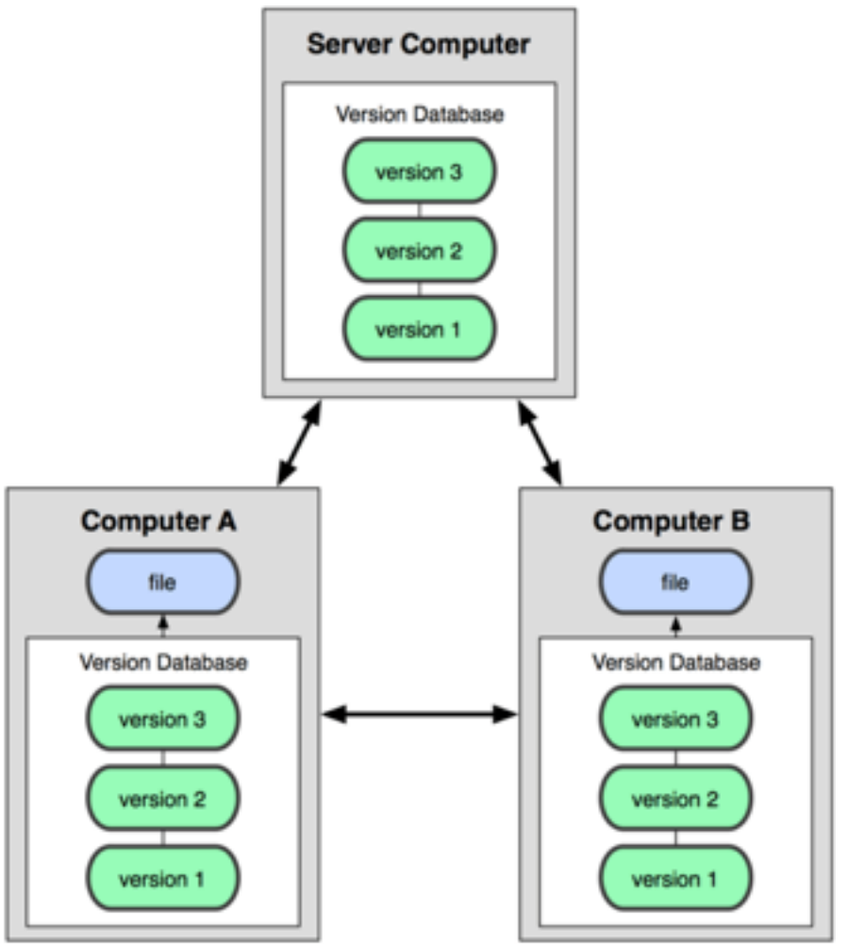
- Stage three, centralised VCS
  - CVS and SVN
    - all operations happen against a central server
    - local copy and remote repository

Evolution of VCS

- Stage four, decentralised VCS
  - Git, Mercurial, Bazaar
    - multiple "copies" of the repository that are kept in sync
    - "everything" is available locally

Evolution of VCS





Important Concepts

- Repository / Working Copy
- Change (List)
- Commit
- Trunk / HEAD
- Conflict / Merge
- History

Git

- Distributed version control
- Developed by Linus for Linux development

- Distributed
- Strong support for non-linear development
- Efficient for large projects

Getting Started

`git clone <url | path> [dir]`

- Clones a repository (more or less the entire project history)
- One of two ways to start using Git
- Locally (file system), or remotely (HTTPS, SSH, Git)

Git Basics

- Inspecting
  - status, log diff
- Modifying
  - add, rm, mv, commit
- Use `git help [<cmd>]` to get help!

Configuring Git

```
git config [--global] <key> <value>
```

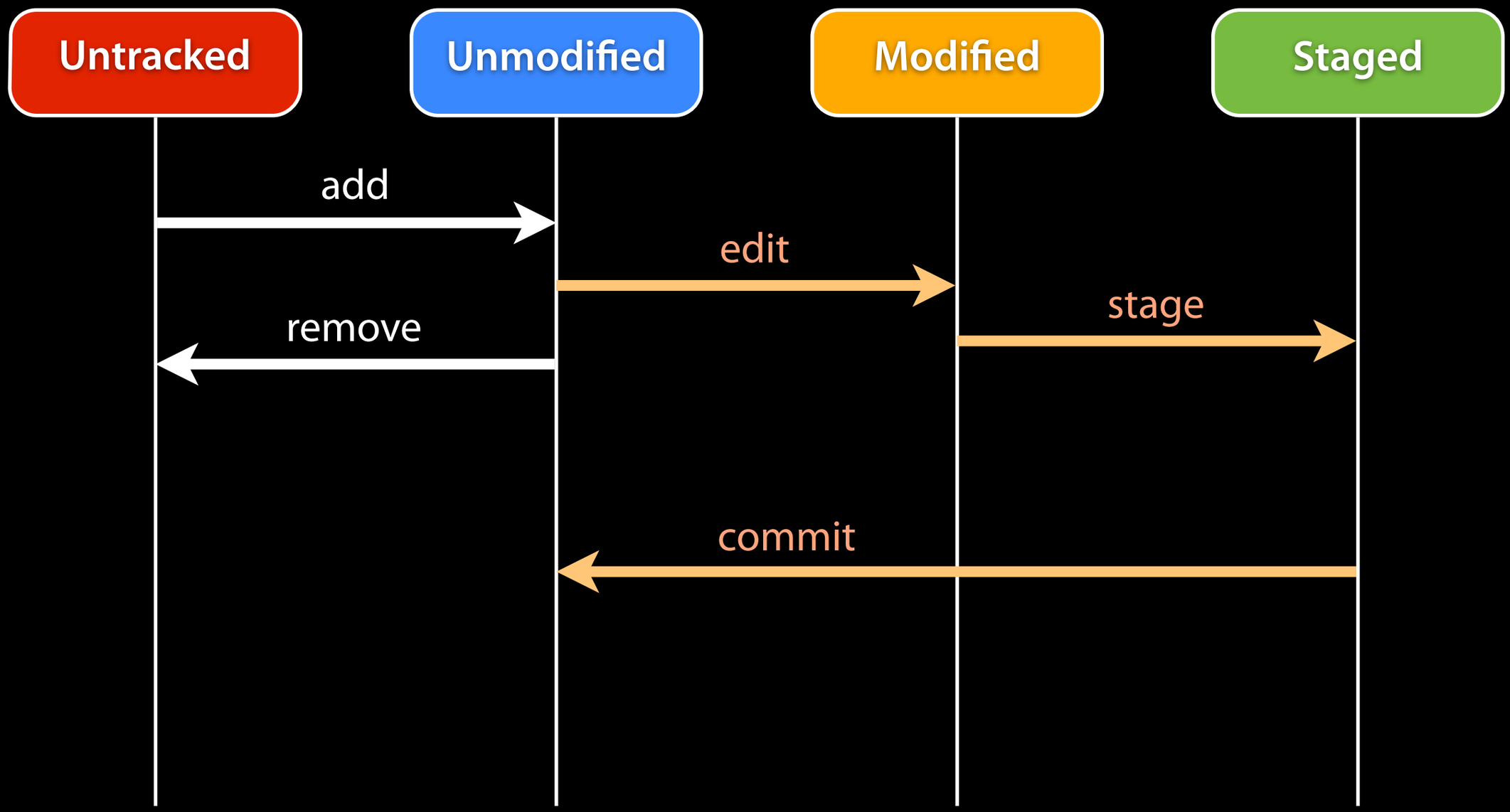
- Allows you to change settings, such as identity, editor, etc.
- Use `--global` to set information across all your repositories

Demo

File States

- **Untracked**: not managed by Git
- **Unmodified**: tracked, but not changed
- **Modified**: tracked and changed since last commit
- **Staged**: will be saved at next commit

Work Flow / Concepts



Add to the Repo

- Use `git add` to add files to the staging area (index)
- A staged file will be "saved" the next time you commit (basically create a snapshot of your repo)
- You can add files or directories, several at a time (and as many times as you like)

Committing Your Changes

- Saves a snapshot of your current repository
- Staged files become tracked by Git and are now considered "unmodified"
- When you commit, Git will ask you to comment on your changes

Removing and Moving Files

- You can also remove files ...
  - `git rm`
- ... and move files
  - `git mv`
- Note that when you remove a file, you will remove the local copy (as well)

Logging and Diffing

- You can use `git log` to show the commit history
  - so, good commit messages are important!
- And, you can use `git diff` to see changes between current and commit or local and remote

§

- The `.gitignore` file allows you to specify files that should never be tracked by Git
  - such as `.class` files or `.DS_Store`
  - Text file that lives in your repository
    - with a list of patterns of files and directories to ignore
  - Github can help you generate sane defaults!

Branches

- A branch is a copy of the project state at a specific time
  - a fork
- There is a main branch, master (or trunk)
  - and as many branches as you create

Why Branch

- Feature development
  - maintain a stable version while you work on something experimental
- Bug fixes
- Release management
  - the DAT255 repo has a VT2013 branch

Branching

- Use `git branch` to manage branches
  - `git branch <name>`
- Use `git checkout` to switch branches
- Other commands work as expected with branches

Merging

- When we want to integrate or incorporate changes across branches, we merge
  - `git merge <branch_to_merge>`
- There can be conflicts
  - that you might have to resolve manually

Playing With Others

- Use `git clone` to get started
- To update to or from a remote repository
  - `git fetch`, `git push` and `git pull`
  - note, pull fetches and merges
- Use `git remote` to check / manage remote repositories

Branches

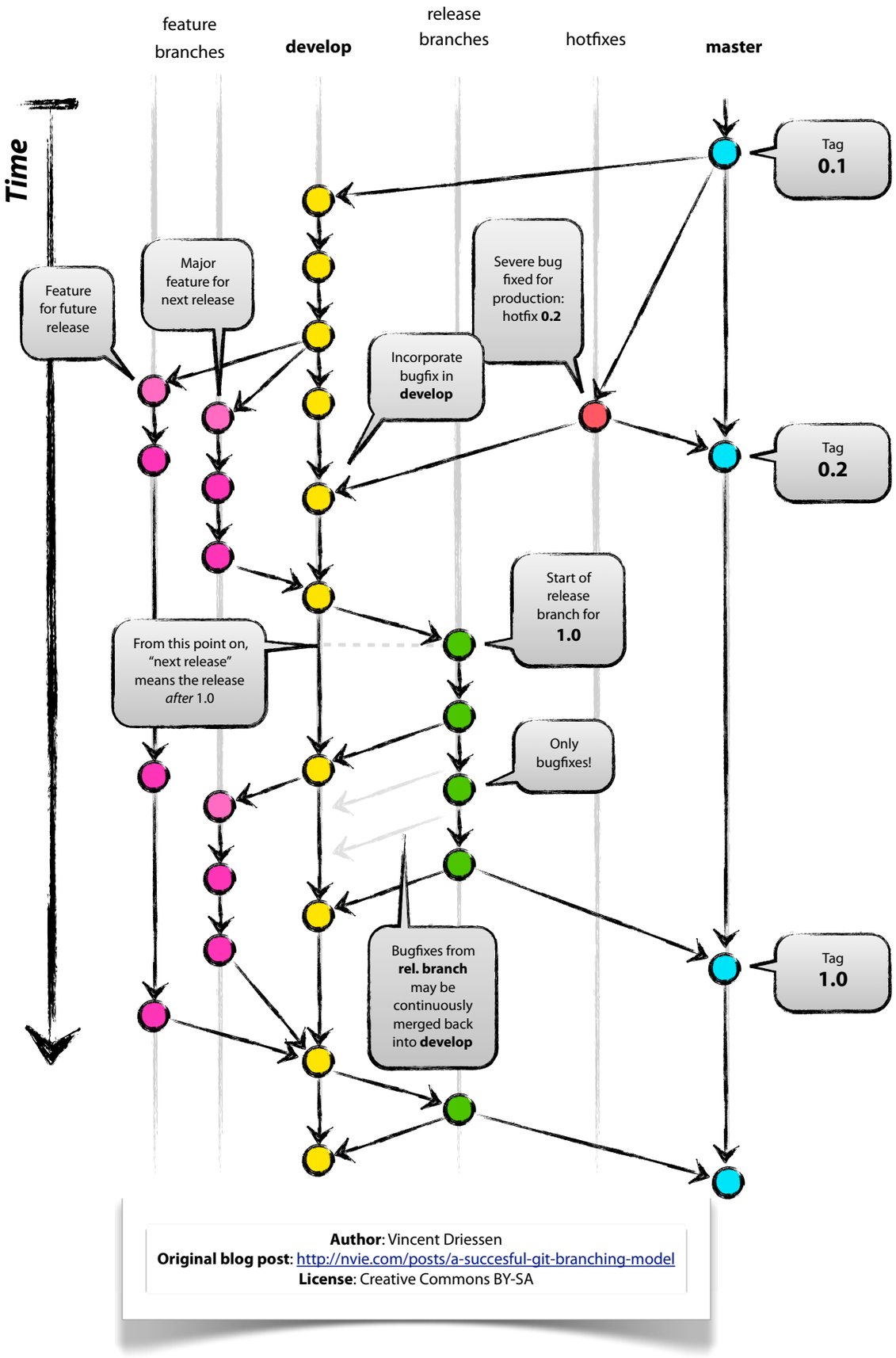
- Branches makes it more complicated, and working with other developers usually means more branches
- Remote branches are local but cannot be modified
  - you need to create a local branch to change / modify it

Releasing (and Tagging)

- A release often has a name or a version attached to it
- Use tags to mark specific commits
  - light-weight and annotated
  - `git tag [-a] <tagname> <commit>`
- Use annotated tags for releases
- Tags are not pushed by default (use `--tags`)

Best Practice

- The VCS can be a powerful tool or just another obligation (in this course)
- To make it powerful
  - set it up properly
  - find a good structure
  - embarace branching

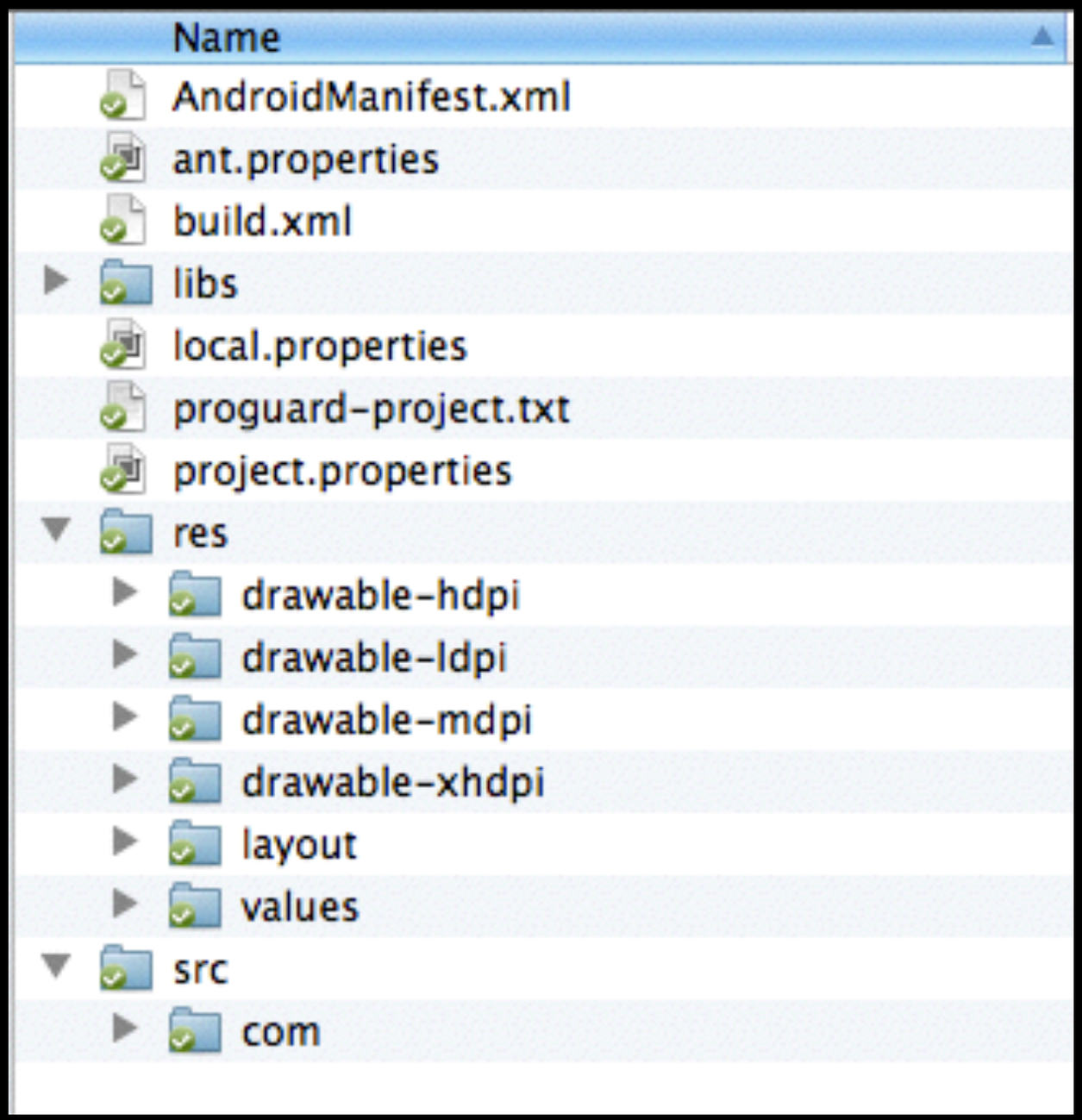


- Do not use *master* for everything!
- Use development, feature, fixes and release branches
- Tag the various releases
- Keep the branches as part of the history/log!

Android SDK

- From our perspective
- Java-based API
- XML-based layout
- tools to build/install/simulate

Structure of a project



First example

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

First Example

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://
schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, MainActivity"
    />
</LinearLayout>
```

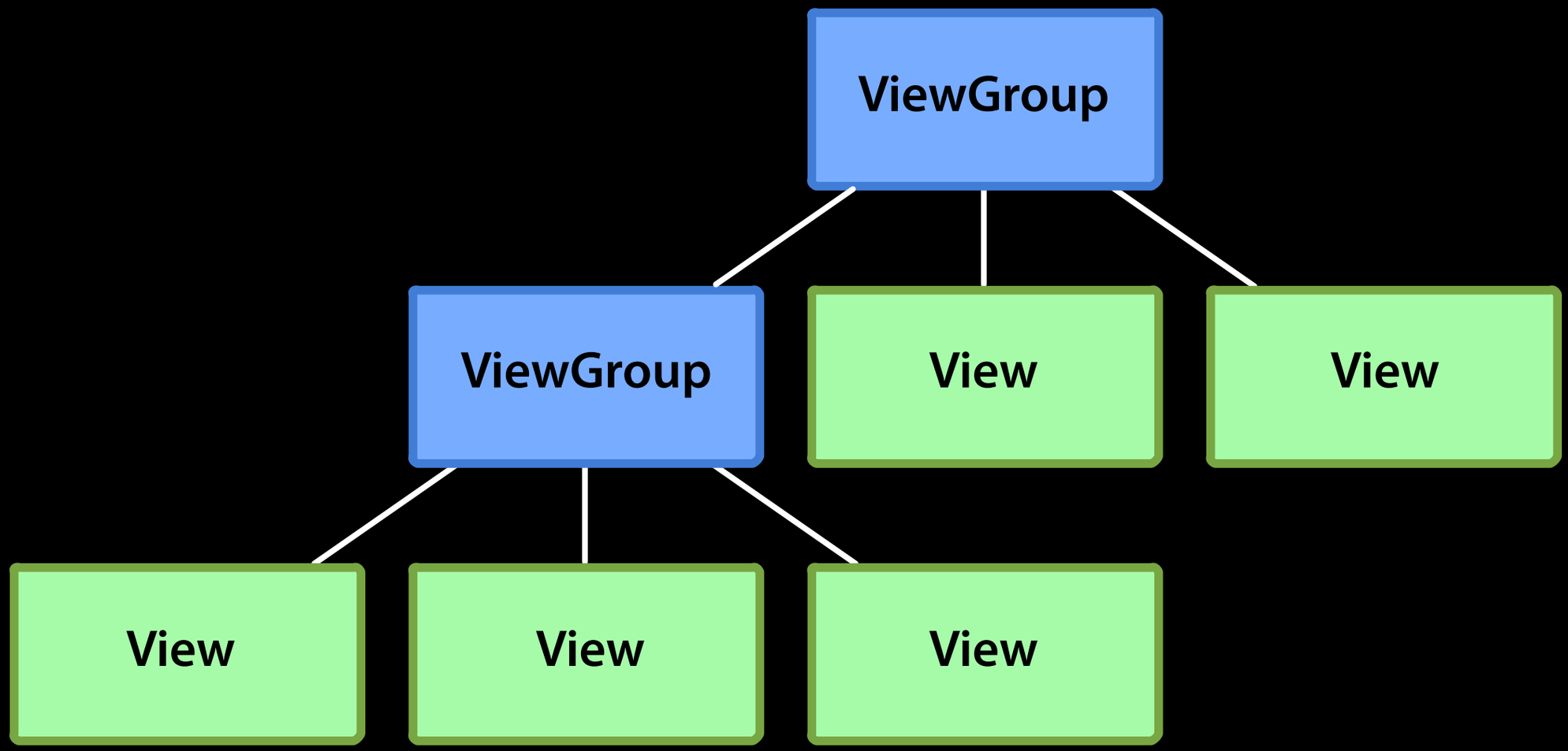
Add User Input

```
package com.example.helloname;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.EditText;

public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void sendName(View view) {
      TextView dn = (TextView)findViewById(R.id.displayName);
      dn.setText("Hello "+
((EditText)findViewById(R.id.enteredName)).getText().toString());
    }
}
```

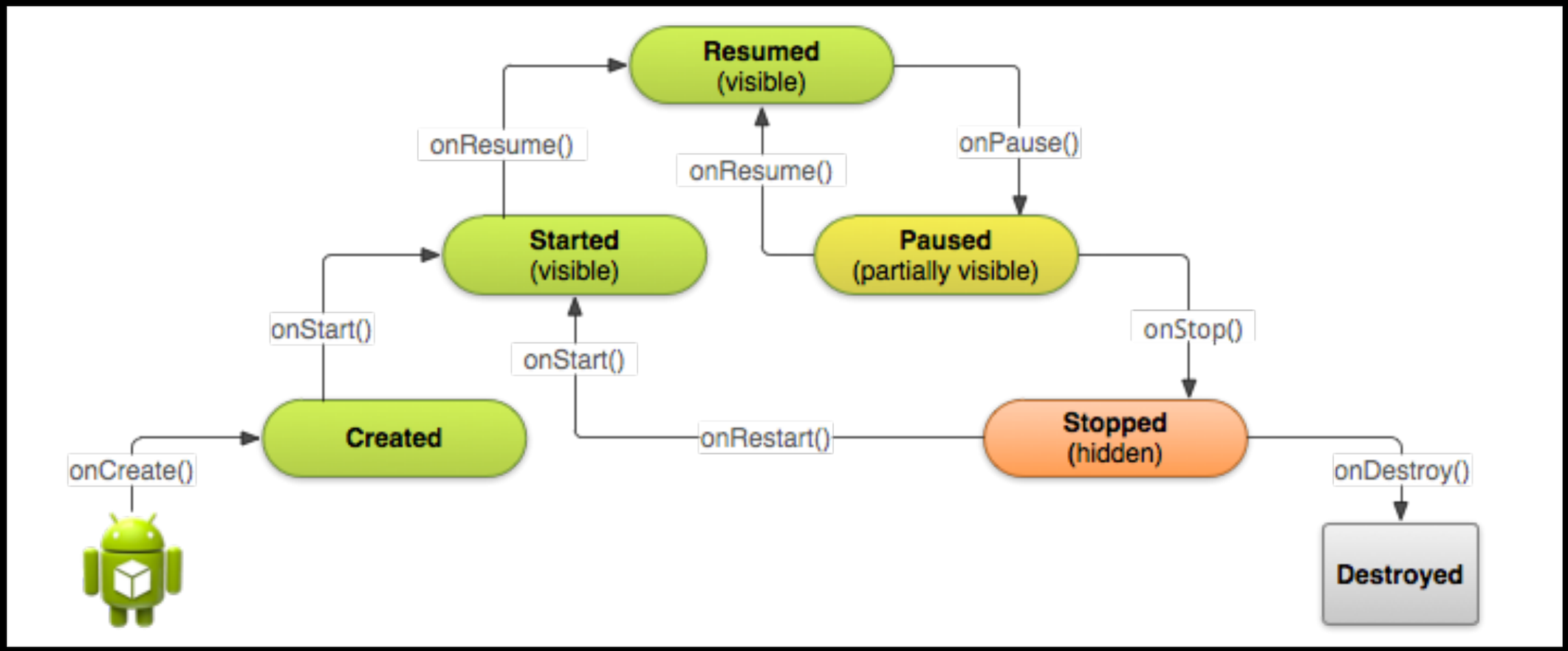
UI basics : Views



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://
schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
  <EditText
    android:id="@+id/enteredName"
    android:layout_width="0dp"
    android:layout_weight="1"
    android:layout_height="wrap_content"
    android:hint="Enter your name..."
    />
```

```
<Button
   android:text = "Submit..."
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:onClick="sendName"
   />
</LinearLayout>
<RelativeLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:id="@+id/displayName"
   android:layout_centerInParent="true"
   android:textSize="24sp"
   android:textColor="#FF0000"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Hello World, MainActivity"
     />
</RelativeLayout>
</LinearLayout>
```

Activity life cycle



Depending on the complexity of your activity, you probably don't need to implement all the lifecycle methods.

More information

- https://github.com/morganericsson/DAT255Demo
- https://github.com/morganericsson/AndroidExamples
- Video lecture (from VT2013)

AGA SDK

- Two versions, ROM and JAR (our names)
- Superset of Android, so anything Android should work
- From our perspective
  - Signals that we can read (and write)

Setup

- Two options for SDK
  1. Install and setup their SDK and ROMs
  2. Use 3 + 2 JAR-files in your “normal” Android projects
- Download and install the simulator (note: requires Java 8)

Demo

Can be difficult…

- Many different concepts, no clear way to test
- Simple error / missed thing can result in a complete “failure”
- No clear way to know if it is working or not
- Use the examples to test your installation

Hello AGA

1. Access the Automotive API
   - create an AutomotiveManager instance
   - with a certificate and listeners
2. Register for various signals that you care about
   - these will trigger your listener

Listen for signals

```
new AutomotiveListener() {
    @Override
    public void receive(final AutomotiveSignal asig) {
        ds.post(new Runnable() {
            public void run() {
                ds.setText(String.format("%.1f km/h",
                    ((SCSFloat) asig.getData()).getFloatValue()));
            }
        });
    }

    @Override
    public void timeout(int i) {}

    @Override
    public void notAllowed(int i) {}
}
```

And distraction level

```
new DriverDistractionListener() {
    @Override
    public void levelChanged(final DriverDistractionLevel dl) {
        ds.post(new Runnable() {
         public void run() {
                ds.setTextSize(dl.getLevel()*10.0F + 12.0F);
            }
        });
    }
}
```

Create the manager

```
AutomotiveFactory.createAutomotiveManagerInstance(
    new AutomotiveCertificate(new byte[0]),
    new AutomotiveListener() { ... },
    new DriverDistractionListener() { ... }
).register(AutomotiveSignalId.FMS_WHEEL_BASED_SPEED);
```

Note: you can specify several signals (comma separated)

What is missing from the example?

- You should do as little non-UI work as possible on the main thread
  - so we use an async task to listen / do networking on another thread
  - and post to communicate with the main thread

Demo

What about the ROM?

- A few major differences (for our example)
  - you get the manager from a service
  - the distraction level is a (sticky) broadcast

```
final AutomotiveManager manager = (AutomotiveManager)
   getApplicationContext().getSystemService(Context.AUTOMOTIVE_SERVICE);

manager.setListener( ... );
manager.register(AutomotiveSignalId.FMS_WHEEL_BASED_SPEED);
manager.requestValue(AutomotiveSignalId.FMS_WHEEL_BASED_SPEED);

final BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final int level =
intent.getIntExtra(AutomotiveBroadcast.EXTRA_DRIVER_DISTRACTION_LEVEL, 5);
        ds.post(new Runnable() {
            public void run() {
                ds.setTextSize(level*10.0F + 12.0F);
            }
        });
    }
};

final IntentFilter intentFilter = new
IntentFilter(AutomotiveBroadcast.ACTION_DRIVER_DISTRACTION_LEVEL_CHANGED);
final Intent currentValue =
getApplicationContext().registerReceiver(receiver, intentFilter);
```

Now what?

- A combination of various signals and distraction level should be enough for many projects
- If you need more advanced features
  - check the SDK documentation / use the forums

More information

- https://github.com/morganericsson/AGAExamples
- https://developer.lindholmen.se/redmine/projects/aga/wiki