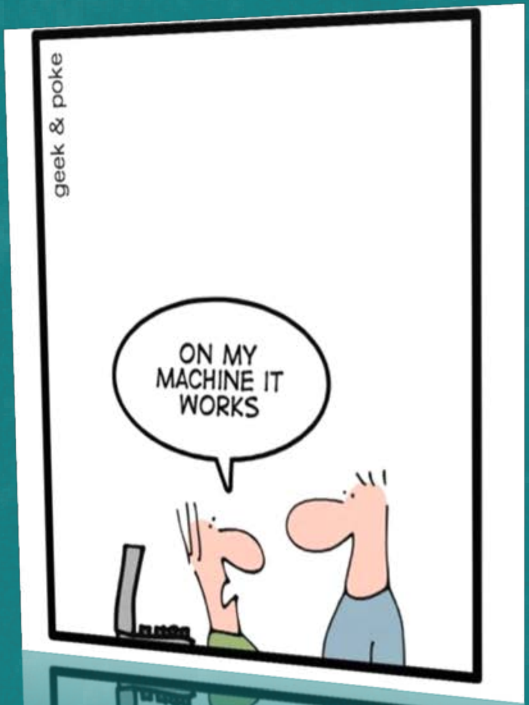


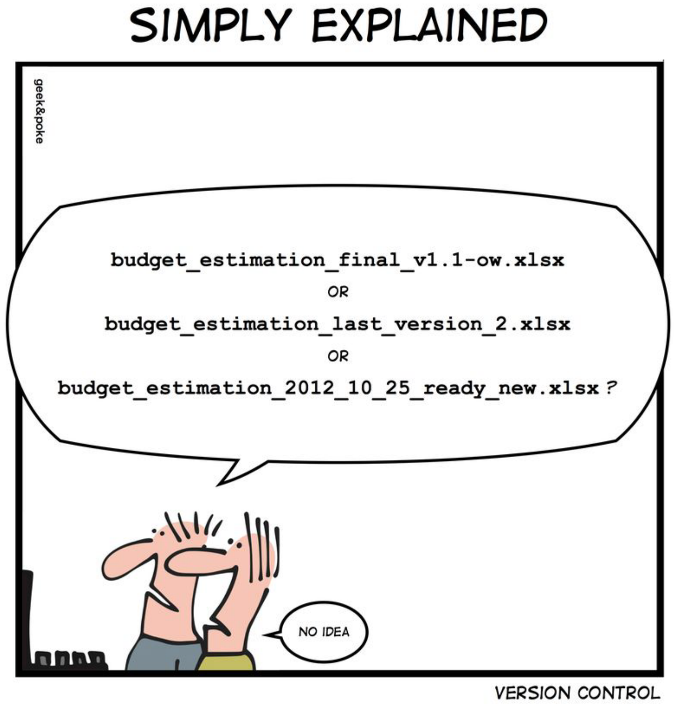
Collaborative Development Tool

Version Control

Sharing Code and Documents

- Passing copies from person to person using
  - e.g. MSN, e-mail or USB memory sticks?
- Who's got the latest version?
- Who's got the right to edit?

- Solution?
  - Save it to the server!
  - Problem solved!

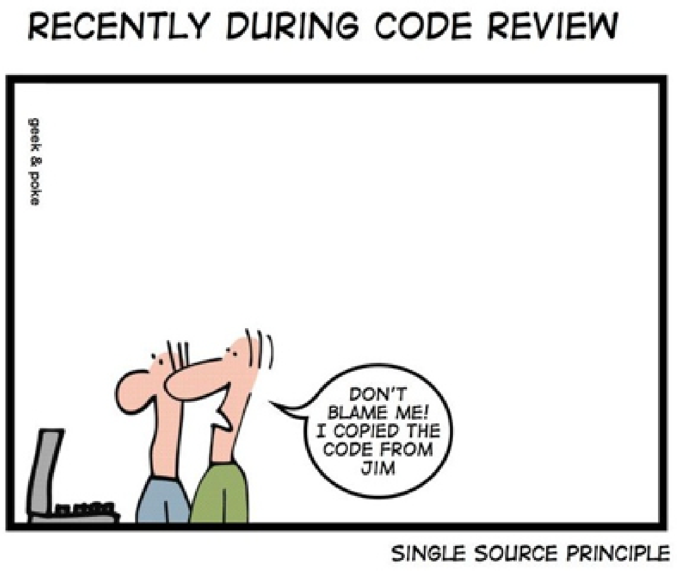

SIMPLY EXPLAINED
geek&poke
budget_estimation_final_v1.1-ow.xlsx
OR
budget_estimation_last_version_2.xlsx
OR
budget_estimation_2012_10_25_ready_new.xlsx ?
NO IDEA
VERSION CONTROL

In the past…

- Small Team
  - Usually Single person, single project –works ok.
- Periodic backups
  - Daily, weekly, monthly, etc…
- Project Coordinator!!!!
  - Assist PM in coordinate works
  - file naming, versioning etc

Now…

- Console teams can be upwards of 100 people
  - Mobile games can be built with 10-15 people.
- Core group of people is divided up into engineering, art, animation, game design, and production.
- ….
- Who is keeping scores????

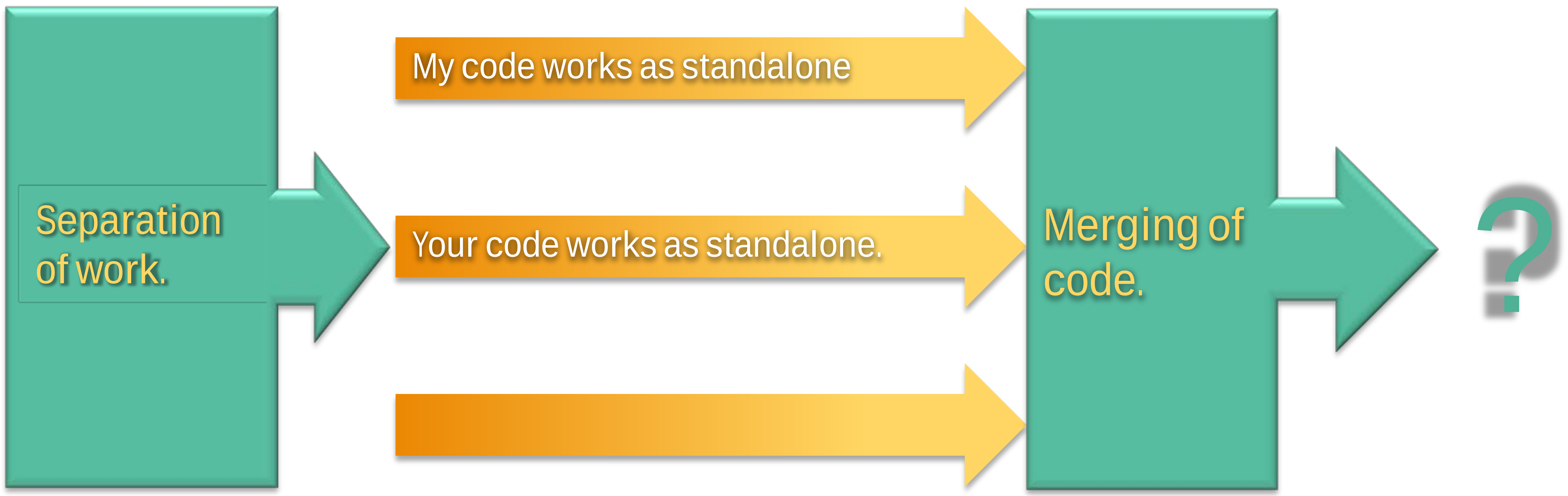

RECENTLY DURING CODE REVIEW
geek & poke
DON'T BLAME ME! I COPIED THE CODE FROM JIM
SINGLE SOURCE PRINCIPLE

Problems Just Get Bigger

- The bigger the team, the greater the potential for disaster
- Manual merging: a nightmare that grows exponentially
- So do the risks for errors
- Bugs creep in
- Doomsday!

Assassin's Creed II had 450 members 3 time more than origin
Assassin's Creed IV is made by nearly 1,000 members across 7 studios

Can our code work together?



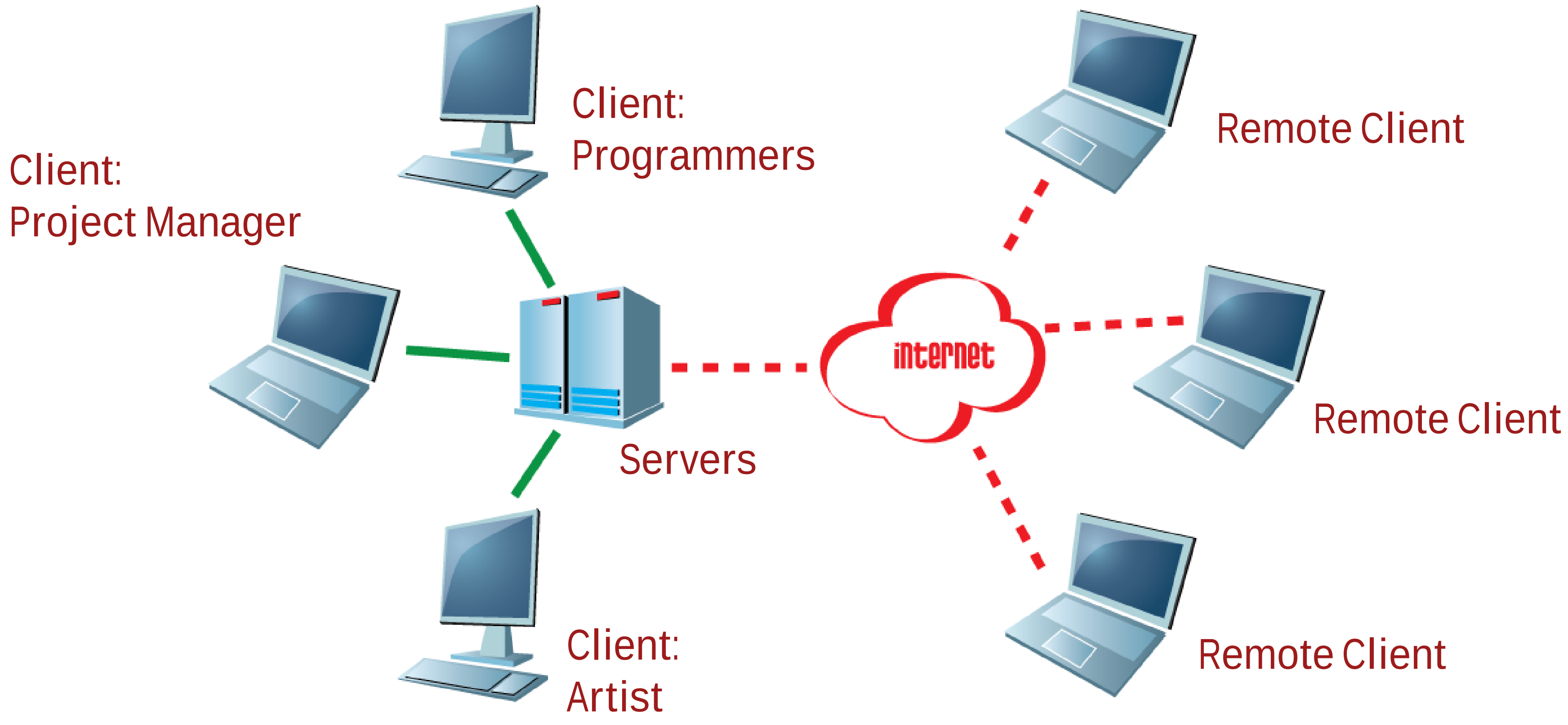
Collaborative Development

- Developers need tools to
  - Work in a team
  - Work efficiently
  - Manage project's progress reliably and accurately
  - *Across geographical borders?*
- How to ensure that everyone sees up-to-date versions of everything?

Version Control

- or Revision Control or Source Control
- Enable you to track multiple versions of your files over time (*across boundaries?*).
  - Every team members have “same” version
  - When you mess up, you can roll back to a previous working version.
- It provides a more powerful alternative to keeping backup files.
- Benefit small team too!

Version Control: Architecture



Version Control: Servers

- High volume asset repository
- Keep up-to-date information about
  - Who and when created
  - File size
  - Version histories, etc
- Control and coordinate access
  - Security & access collisions
  - Download & upload
  - Lock files
- Serve different platform

Version Control: Clients [PC/Mac]

- Manage Asset
- Access project databases
- View, import, lock and modify assets
- Usually optimized for different type of user.
  - Art
  - Design
  - Technical
  - Management
  - …

Control and Coordinate Access

- Authentication – Who has log on?
- Authorisation – Who can do what?
  - No unauthorized access to proprietary information
  - No unauthorized modification of file structures
  - Delete and renames files etc
- Security – Restricting specific groups from accessing assets
  - E.g Art & Design can see but cannot modify source codes.

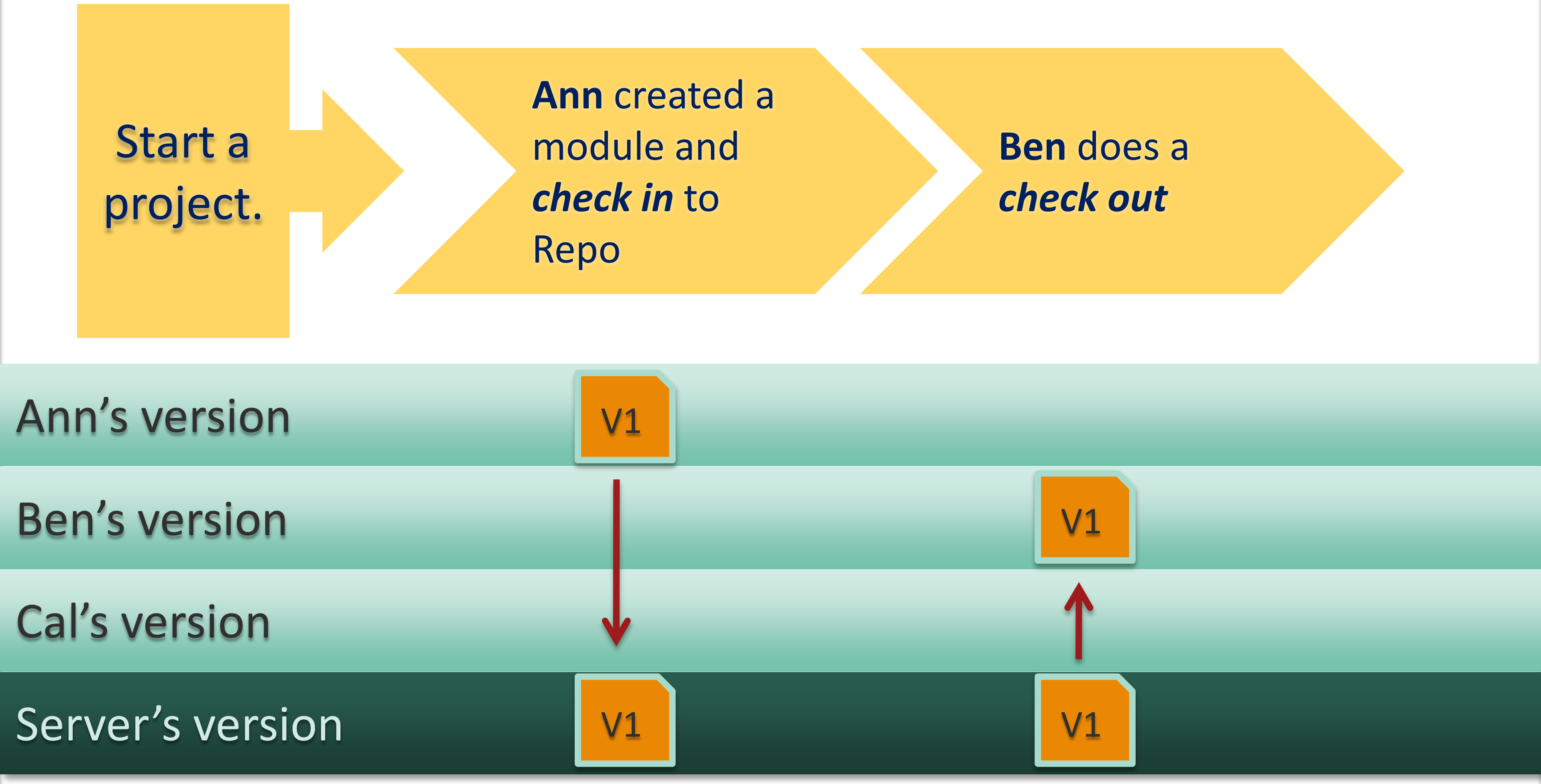
Version Control: Collaboration

- Work in parallel on projects/modules
- Work between groups
  - Art & Design, Technical…
- Simultaneous changes
- Work on a part of the project

Benefits of Version Control

- Synchronization
  - Members share latest version – Up to date
  - Concurrent access to resources
- Backup and Restore
  - To any moment of time
- Roll back
  - Build fail? No problem, we roll back!
- Change tracking
  - who made what changes.
  - when was the change made–Timestamps
- Auto-merge
  - no more worries about merging code.
- Independent development
  - Tested before “checking in” changes

How does Version Control Work?

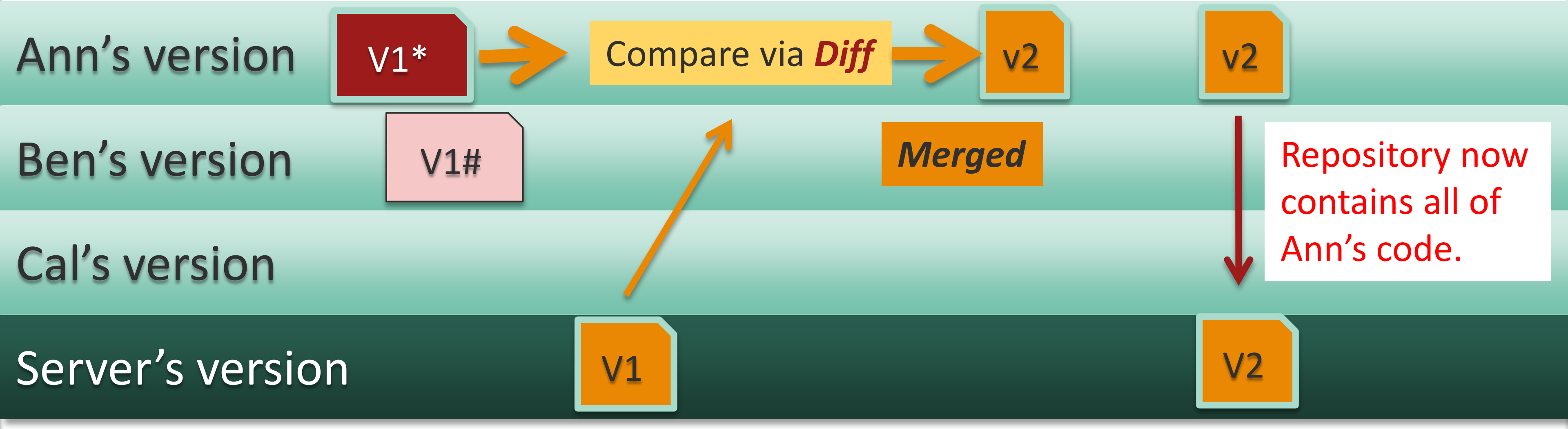

Start a project.
Ann created a module and ***check in*** to Repo
Ben does a ***check out***
Ann's version
Ben's version
Cal's version
Server's version
V1
V1
V1
V1

How does Version Control Work?

Ann and ***Ben*** work on their local **working set**

Ann do an ***Update***

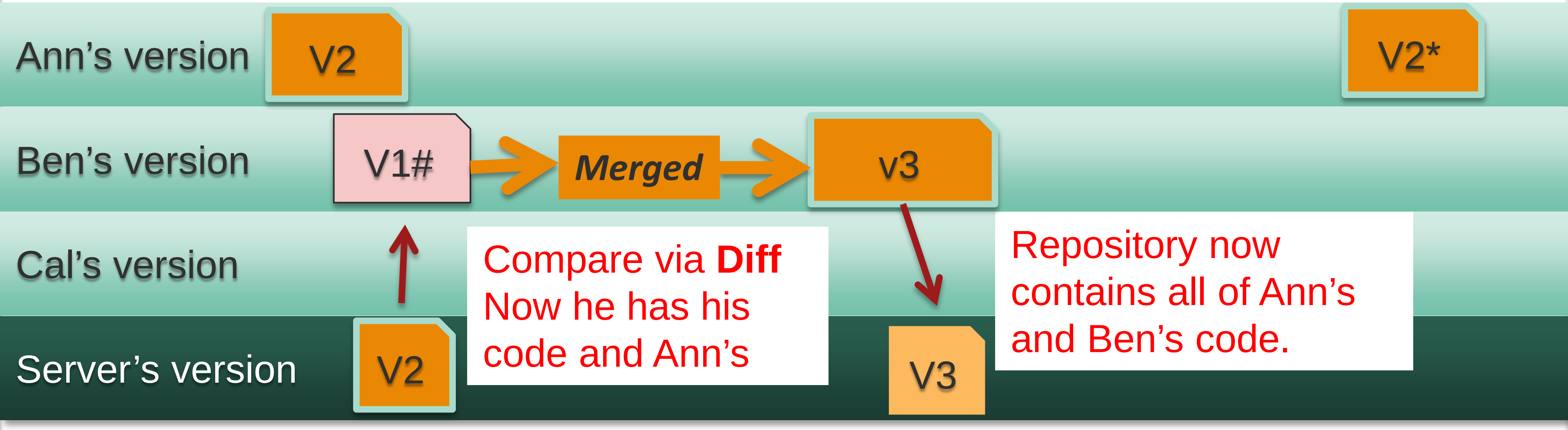
Ann commit and ***Check in*** the updated file



How does Version Control Work?

Ben completed his code and do an ***update***

Ben commit and ***Check in*** the updated file



How does Version Control Work?

		Ann	Ben	Charli	Server
1	**Ann** created a module and ***check in*** to Repo	V1			V1
2	**Ben** does a ***check out***	V1	V1		V1
3	**Ann** and ***Ben*** work on their local **working set**	V1*	V1+		V1
4	**Ann** do an ***update***	V1*	V1+		V1
5	**Ann** commit and ***Check in*** the updated file	V2	V1+		V2
6	***Ben*** completed his code and do an **update**	V2*	V1+		V2
7	**Ben** commit and ***Check in*** the updated file	V2*	v3		v3

How to get all updated to same version?

How does SVN work?

Version Control with SVN

How does SVN work?

- Can either work locally, or via the network.
  - Network usage is necessary for teamwork
  - Local usage is similar, except for setup, and environment variables used.
- Keeps versions of each file in a central repository on the SVN Server
- It handles requests from clients to make amendments, or retrieve (rollback) past versions of files.
- It also caters for conflict resolution if several changes are committed at once.

Best Practices

- Download a copy to your working directory and work on that copy.
- Always do an update (check out) before committing changes.
- Merge conflicts
  - Do not Panic! Solve conflicts in source files and check back in.
- As much as possible, do proper QA on code before a check in. Only thoroughly tested QA'ed builds make it to the stable release system, deemed to be clean and stable builds.
  - Some companies keep production and stable release VC systems.

Before we continue…

Version Control Terminologies (SVN Terminologies)

Terminologies

- Repository (or repo) –The root
  - Where file are tracked via database
  - Can contain multiple modules.
- Server
  - Where the repo is stored
- Client
- *Never access the files in the repository directly*.

Terminologies

- Module
  - maps to a project. Usually a folder on a computer drive. E.g. C++ project
  - Groups sources into modules
- Each Repository can contain multiple modules.

Terminologies

- Working Set/Working Copy
  - You local file directory
- Trunk/Main
  - Primary location for the code in the repo

Practical 1

Version Control with SVN

Terminologies - Starting a project

- Import *module*
  - adds a new module (selected folder) to the repository.
- Check out
  - transfer from *server (repository) to client*.

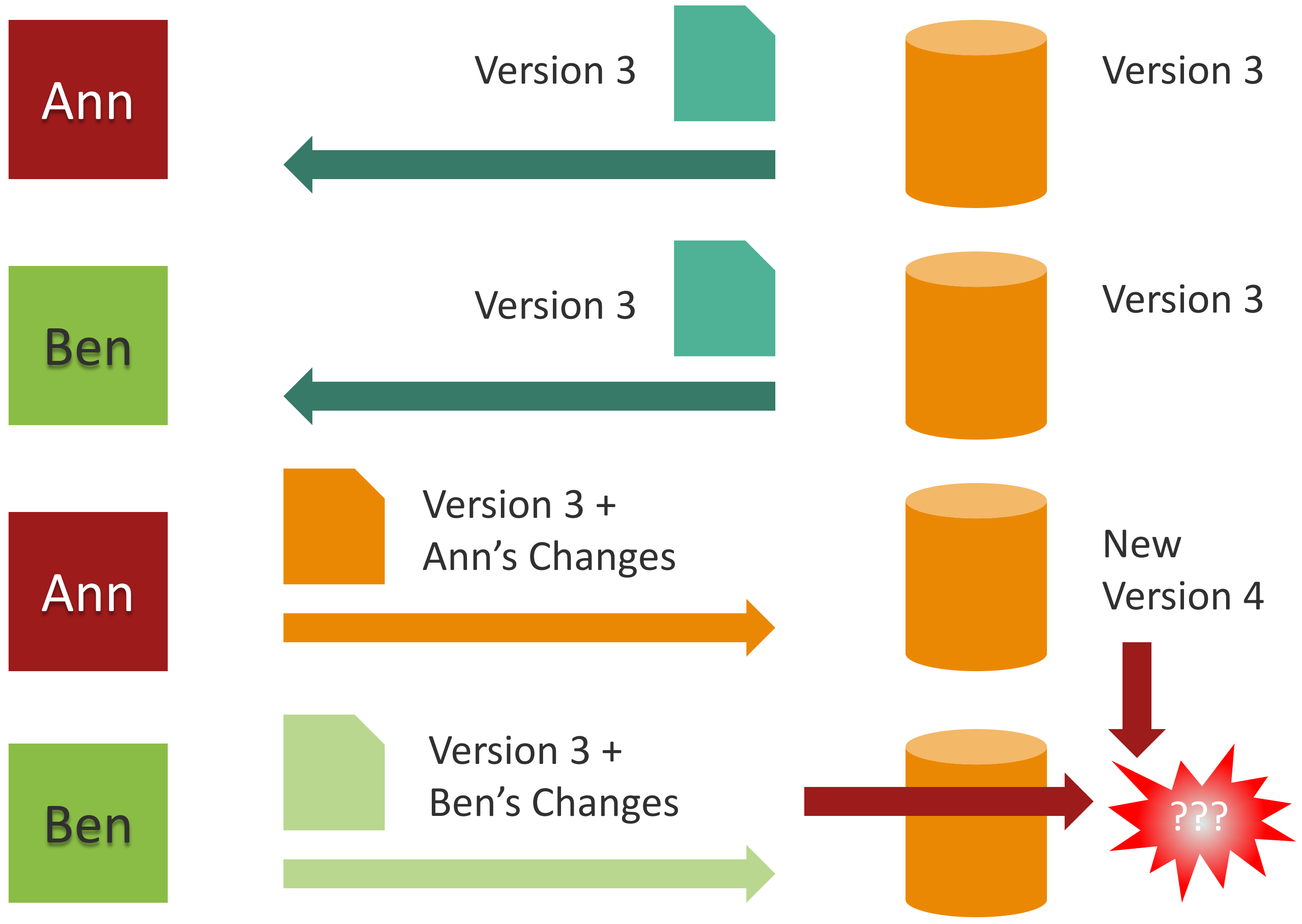
Terminologies

- Add *a new file*
  - Put a file into the *repository*
- Commit *(or check in)*
  - Similar to *Import Module* except that we upload the changes.
  - Always *Update* your local copy BEFORE Commit for existing files
- Update/Sync
  - Synchronize your files with the latest from the repo
  - Allow you to see changes made to the *repository*
  - Resolve any editing conflicts

Part 2

Conflict Resolution with SVN

When Conflict!



Conflict Resolution

- Most tools does not require you to acquire a lock to file before making changes.
  - Strict locking prevent others from updating the file before you “unlock” it.
- During update, the tools will try to merge the changes.
- Conflict are rare and usually occurs when changes are made to the same place.
  - These need to be solved manually.
- Cannot do conflict resolution for binary files.

Terminologies

- Diff/Change/Delta
  - The differences between two files.
- Conflict
  - When pending changes to a file contradict each other (both changes cannot be applied).
- Merge (or patch)
  - Apply the changes from one file to another, to bring it up-to-date.
- Resolve
  - Fixing the changes that contradict each other and checking in the correct version.

Example 1: Changes – no overlap

```
int main()
{
    int choice=0;
    int n=0;

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
}

// print all even numbers from 1 to n
int even(int n)
{
}
```

Server

```
int main()
{
    int choice=0;
    int n=0;

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
  //   Ann added here!!!
}

// print all even numbers from 1 to n
int even(int n)
{
}
```

First Person: Ann

```
int main()
{
    int choice=0;
    int n=0;

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
}

// print all even numbers from 1 to n
int even(int n)
{
  //   Ben added here!!!
}
```

Second Person: Ben

Example 1: Diff and Merge Server Version

Ann *update* and *commit*

```
int main()
{
    int choice=0;
    int n=0;

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
}

// print all even numbers from 1 to n
int even(int n)
{
}
```

```
int main()
{
    int choice=0;
    int n=0;

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
  //  Ann added here!!!
}

// print all even numbers from 1 to n
int even(int n)
{
}
```

Updated version on server

Ben *update* and *commit*

```
int main()
{
    int choice=0;
    int n=0;

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
  //  Ann added here!!!
}

// print all even numbers from 1 to n
int even(int n)
{
  //  Ben added here!!!
}
```

Updated version on server

Example 2: Changes –overlap

```
int main()
{
    int choice=0;
    int n=0;

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
}

// print all even numbers from 1 to n
int even(int n)
{
}
```

Server

```
int main()
{
    int choice=0;
    int n=0;
 //  Ann added here!!!

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
  //  AND added here!!!
}

// print all even numbers from 1 to n
int even(int n)
{
}
```

Ann

```
int main()
{
    int choice=0;
 //  Ben added here!!!
    int n=0;

return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
}

// print all even numbers from 1 to n
int even(int n)
{
   // AND added here!!!
}
```

Ben

Example 2: Diff and Merge Server Version

Ann *update* and *commit*

```
int main()
{
    int choice=0;
    int n=0;

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
}

// print all even numbers from 1 to n
int even(int n)
{
}
```

```
int main()
{
    int choice=0;
    int n=0;
 //   Ann added here!!!

    return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
  //   AND added here!!!
}

// print all even numbers from 1 to n
int even(int n)
{
}
```

No Problem

Updated version on server

But when Ben *update* and *commit*

Error!!!

```
int main()
{
    int choice=0;
 //   Ben added here!!!
// Ann added here!!!!
    int n=0;

return 0;
}

// print all odd numbers from 1 to n
int odd(int n)
{
 //   AND added here!!!
}

// print all even numbers from 1 to n
int even(int n)
{
   // AND added here!!!
}
```

Merge Failed!!!!

Make a serious mistake?

Revert and Rollback!

Terminologies

- Versioning/Revision
  - 0.1 to 1.0 and release candidates.
  - Based on milestones and feature completion/Increment
  - Usually by Producer
- Change Log
  - History of changes made to a file

Terminologies

- Revert
  - Throw away your local changes and reload the latest version from the repository.
- Roll-back
  - checking out an older version from the current one, eg in the case of buggy releases.

More Terminologies

- Stamping
  - a history of the source code tree in terms of a series of changes on the code.
  - Time it was made
  - Username of person who made it
  - Keep information about the change
- Tagging
  - adding a version number to the current build on the server, usually after a milestone or stable build.

More Terminologies

- Branch
  - Create a separate copy of a file/folder
- Locking
  - Taking control of a file so nobody else can edit it until you unlock it.
- Breaking the lock
  - Forcibly unlocking a file so you can edit it.
- Check out for edit
  - Checking out an “editable” version of a file.

Versions and Releases

Versioning

- Common practice:
  - Even version number(example -0.2):Stable.
  - Odd version number (example -0.3):Development build.
- Stable builds can be released to the team (level design, game play testing, etc.)
- Development builds stay in the programming team.

Releases

- Development/production release
  - usually contain bugs with occasional crash.
- Stable release
  - Fully tested build. Does not crash, might contain some bugs.
- Debug v/s Release mode.
- Why bother ?
  - Investors demo
  - Showcase demo
  - Publicity stunts

Topical Release Stages

- Version 0.1
  - very basic. Usually can load some models/sprites, with main character running around or navigation around map/world. Followed by intermediary milestones to first playable.
- First playable
  - usually one full level that shows most if not all gameplay elements –might not be fully stable.
- Alpha
  - All levels implemented –not fully feature complete. QA team partially setup.
- Beta
  - Feature complete. QA team fully in place by then.
- Release candidates
  - feature complete, almost fully tested. Platform certification done. Minor issues left, e.g. localization.
- Gold master
  - ready to be burnt on CDs/DVDs and shipped,

What is a Build?

- Executable snapshot of project at any one time showcasing current compiled code and game assets.
  - Increment in Scrum
- Build notes or Change logs
  - contain list or most recent features with a history of all changes made during the project. Also contain build number as per VC tags.

Choosing a

Version Control Software (VCS)

VCS Software

- Subversion (SVN), SmartSVN, TortoiseSVN
- CVS, CVSNT, WinCVS, TortoiseCVS
- Git
- Mercurial
- Perforce

Considerations

- Cost / maintainability / support
- Solutions cost money
  - For start-ups, open source Version Control software works great.
  - Companies with more resources can go for commercial software such as Perforce or Alien brain.
- Some incorporate solutions for assets management or specialize in media files management.
- Alien brain quite common in games industry.