Australian National University

Android - Hello World!

Eric McCreath

Click-Click-Click

- With Android studio getting the "hello world" program running is trivial. All you need to do is "click" okay/finish on the wizard for a new application and Android studio will start you off at a "Hello world!" application.
- It is an important starting point for any Android development as if you can set it up and run "Hello World" it means, at least for the most part, you have all the required tools running.

Challenge

- Get "hello world!" running either in the labs or on your own computer.

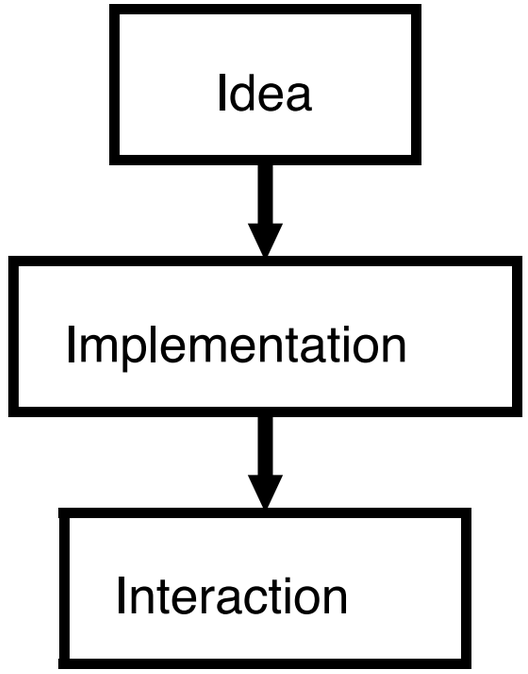
Australian National University

Design

Eric McCreath

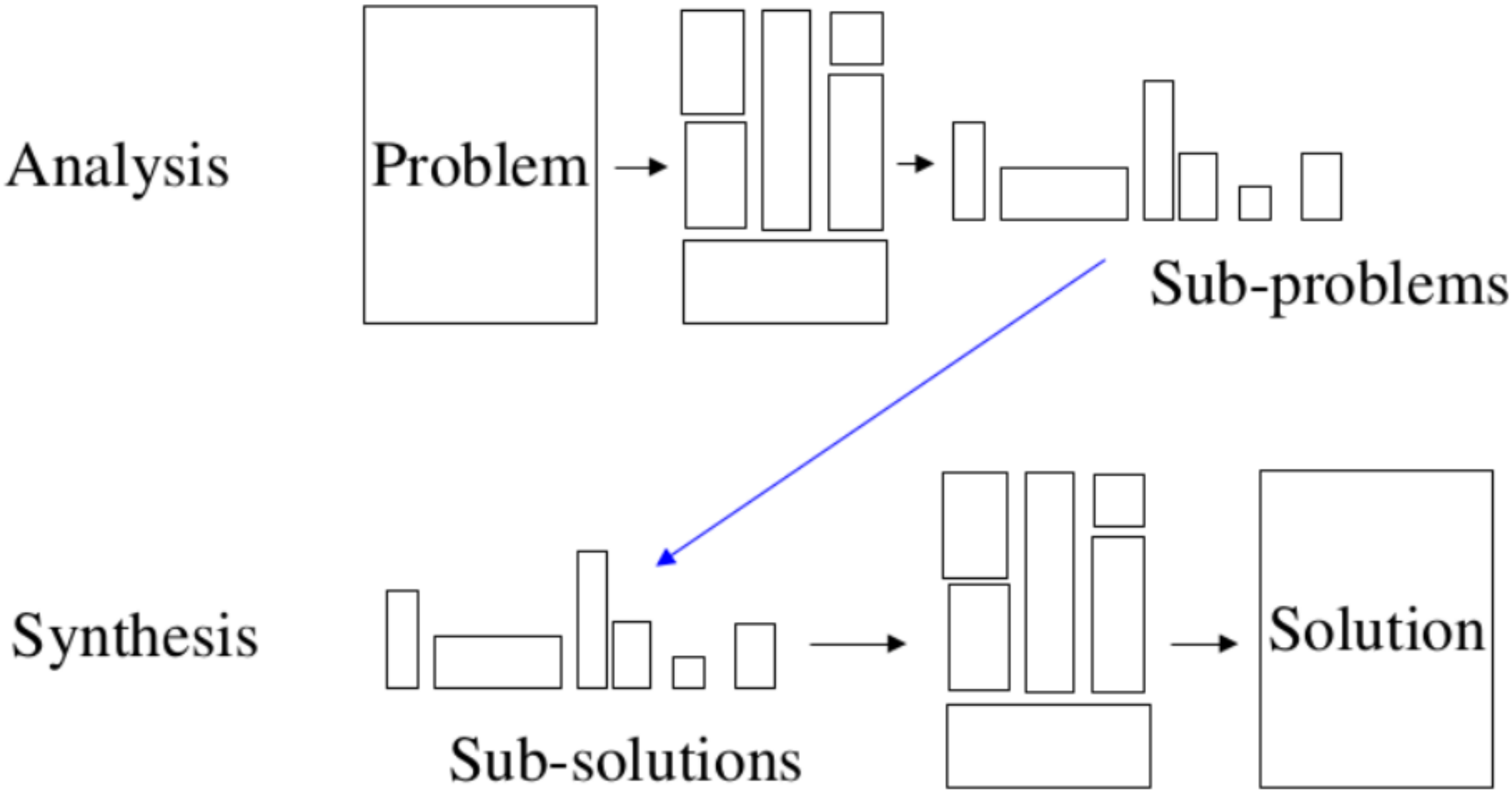
Good Design

- As you move from *Idea* to *Implementation* good design plays a key part in making software robust, maintainable, and flexible.
- Good design is difficult
- It is easy to overcomplicate design such that adding extra functionality becomes difficult.



Analysis and Synthesis

- The design/implementation process may be thought of as analysis and synthesis. This maps a problem to a solution.



Plan to Design

Once you have a clear idea of what you want the software to do (requirements) then spend some time designing your code.

The main and most important part of your design is how you break your code up into modules. From an OO persective this will basically be your class diagram (generally expressed using UML).

Spend some time designing your code. However, keep it in perspective of the length of your project.

Bad design will make for a difficult and fragile implementation.

Reasonably good design will produce an implementation that can always be re-designed (with considerable code reuse).

Top Down

- A common approach to design is known as "top-down"
  - Recursively divide the problem into smaller sub-problems until the sub-problems can be solved.
  - Once the sub-problems are solved, their solutions are accumulated to produce a final solution.
- Often when this approach is adopted, the problem can be envisioned in terms of a process of steps or functions to achieve the final goal.
- This is a useful step to go through. However, **the design it produces is generally not good OO design.**
- When sub-dividing a problem you need to think in terms of DATA more than the process.

Example

- Suppose we wish to develop a simple vector drawing program (like dia/xfig).
- We could think through a list of requirements and constraints. Such as:
  - we need to be able to use a GUI to draw a simple diagram that is made up of lines, boxes, circles, images, and text
  - we must be able to load and store images
  - render and export them as png
  - it must be written in Java

What would a flow diagram look like?

If this was used to construct a UML class diagram would it be any good?

Bottom Up or Data-driven design

- A bottom up design approach considers the modules one already has (or could build) and how they can be composed together to form a solution.
- When we think of OO design, the important things to think about are:
  - Data that needs representation
  - The relationships between different data elements
  - Libraries that need to be used and how they are normally interfaced with

Software Quality Metrics

There are may different ways the performance, quality of design and implementation of a particular piece of software can be measured. These include: *correctness, performance (time), performance (space), portability, library dependence, code readability, ...*

However, in terms of design and dividing the design up into modules there are three important metrics to consider:

- **Coupling** - a measure of how connected different modules of code are. Aim to minimize coupling
- **Cohesion** - a measure of how strongly different parts or functions within a single module of code are related. Aim to maximize cohesion
- **Open/Closed** - the principle that code should be *open* for extension, yet *closed* for modification

Classes and Interfaces

- **Classes** hold data, define relationships between data and provide methods to interact with data
  - Each class should focus on one main aspect of the design
  - Use nouns to name classes. Do not use verbs.
- **Interfaces** provide a flexible way for different parts of code to interact with each other
  - They help reduce coupling
  - They provide the ability to easily change or replace modules of code

Tips for design

- Generally us an iterative design-development approach. Be willing to (and expect to) redesign and re-factor your code.
- Look for code patterns that keep repeating in your implementation. Consider how they can be reduced and if there is better design approach.
- Look for "library" type code and separate and generalize it beyond this program giving it a clear interface.

Exam Question - Example

- Explain the difference between coupling and cohesion in terms of software design. Why is it advantageous to have low coupling and high cohesion in a design?

Australian National University

Design Patterns

Eric McCreath

Design Patterns

A design pattern describes a good solution to a recurring problem in a particular context of software design. A single design may be composed of a number of different design patterns. Advantages of design patterns include:

- Capture expert design knowledge and make it accessible in a standardized expression format
- Promote communication and streamline documentation by providing a shorthand for designers
- Support software reuse and increase software quality and designer/programmer productivity
- Provides a basis or model to improve upon

Design Patterns

As we consider different design approaches and patterns, we need to be mindful of:

- where information will be stored within the design
- how different parts of the design relate to each other
- how the requirements are likely to change over time

Designing to an interface provides enormous flexibility as your code changes over time

Design Patterns

- As a software engineer or programmer you become familiar with multiple design patterns, how to quickly implement them, and their strengths and weaknesses
- Most design patterns aim to produce good design in terms of coupling, cohesion, and the open/closed principle
- They provide software engineers with some common terminology enabling them to communicate with others about their design approach and increasing productivity

Design Patterns

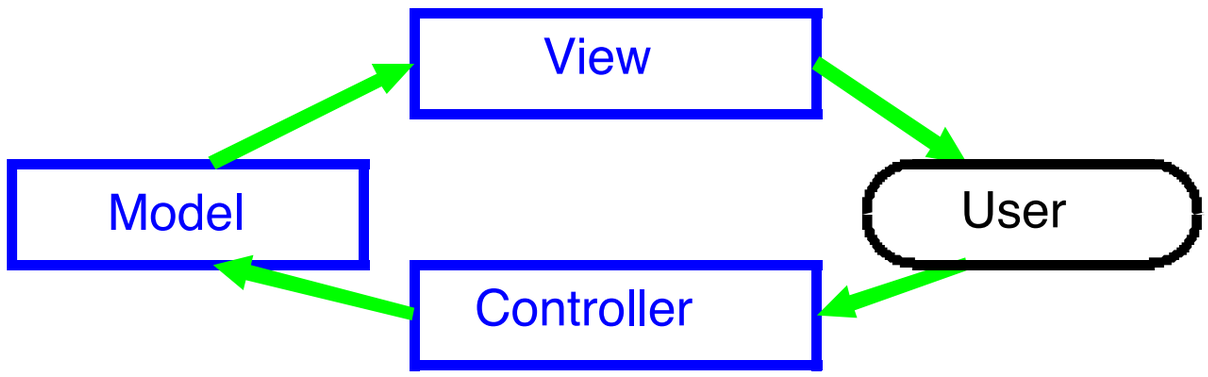
Disadvantages include:

- they can over complicate a design
- some design patterns are difficult to understand
- since they are language agnostic, they may be difficult to implement within a particular language
- they may adversely affect time and space performance

Model-View-Controler Design Pattern

The Model-View-Controller design pattern is often used for GUI development and the approach separates the design into three parts:

- **Model** - stores information. Does not include any functionality about how information is viewed or changed/updated.
- **View** - provides code that enables a user to view information stored in the model
- **Controller** - provides code that enables a user to control or change information stored in the model



Exam Question - Example

- Design a countdown timer GUI program. It could be used for timing how it takes to cook eggs, or to limit the time a student has for a talk. Show your design using a UML class diagram.

Australian National University

GIT - A Background to Revision Control

Eric McCreath

- Without a source control management (SCM) system it is very difficult to develop software in a team.

Revision Control Systems

There are a large number of revision control systems that help manage code. Some common and free ones include:

- **svn** - subversion
- **git**
- **hg** - mercurial

These are critical tools for software development projects.

They enable the sharing of source code such that multiple people can work on different aspects of a software program concurrently.

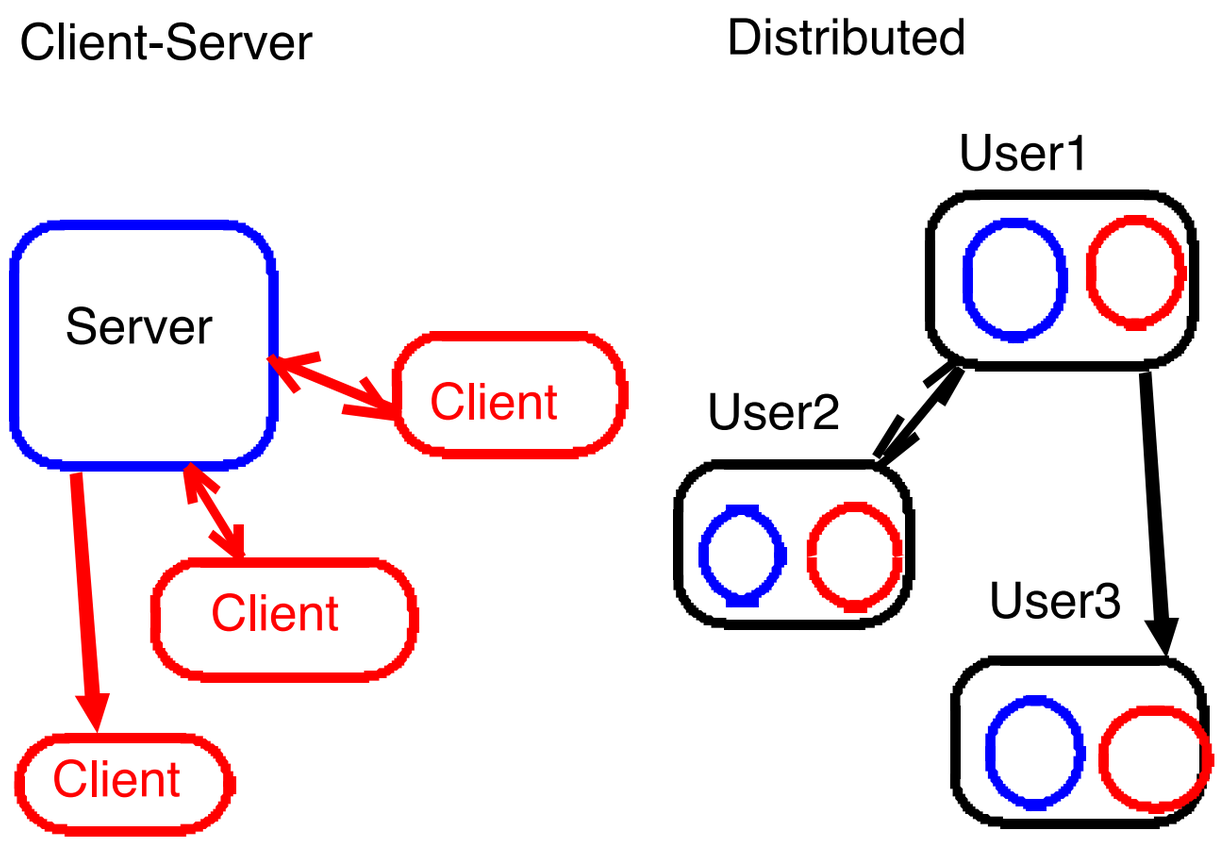
Revision Control Systems

They provide a useful *history* or *log* of how the software has changed and who is responsible for creating and modifying every line of code.

With *branching* they give developers the ability to explore different programming approaches and eventually merge them into the main branch if and when desired.

svn is a *client-server* based revision control system.

Whereas, **git** and **hg** are distributed.



The Rise of Git/HG

As with many free and open source software projects they often begin with someone's need or desire for some software - along with a willingness to share their efforts. This was the case for both git and hg, however, git and hg also began with some controversy. At the heart of this controversy was our own Andrew Tridgell who worked out the BitKeeper protocol (BitKeeper was used to hold the Linux Kernel source code 2002-2005) enabling him and others to access the metadata without signing up to the BitKeeper license.

Dr Andrew Tridgell

Image from wikipedia CCA2.0

The Rise of Git

Git was started and initially developed by Linus Torvalds. Junio Hamano was also a significant contributor and now maintains the project.

Git is "expressly designed to make you feel less intelligent than you thought you were." - Andrew Morton

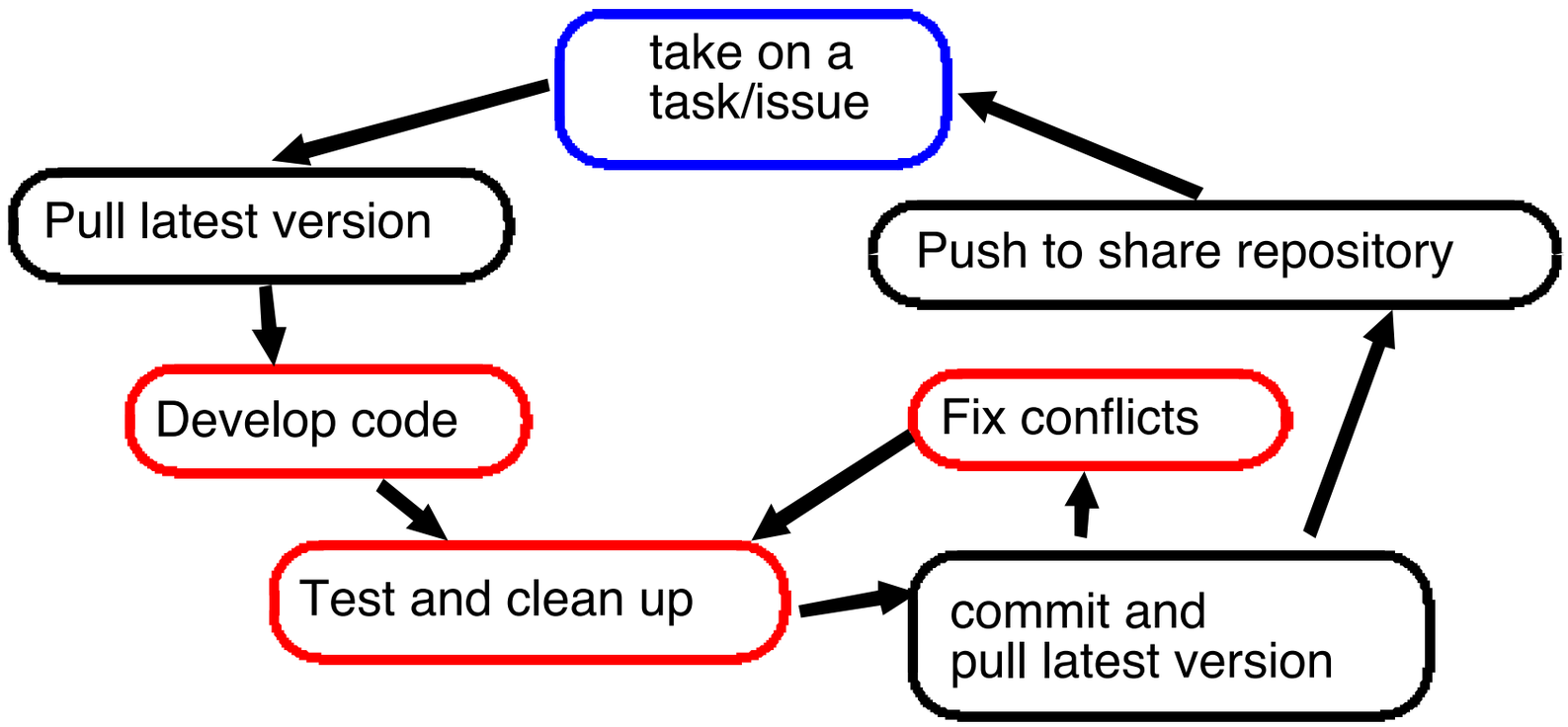
GIT Tools

Along with the basic set of command line tools there is also a large number of GUI and non-GUI tools to help manage and serve git repositories. Some notable ones include:

- **gitk** - GUI repo browser (developed by Paul Mackerras)
- **git-gui** - GUI for git, focuses on making changes to repos (run via git gui)
- **egit** - eclipse plugin for Git
- **gitlab** - a web client for managing git repositories (includes management of ssh keys, wiki, issues, etc). Used by github. We will also be using this.

Software Development Process

- Git is mainly for sharing code. Avoid adding binaries, dev toolkits, iso disk images, etc... under revision control
- If you are working with a team of people you only want code that compiles, is formatted nicely, works, is tested, and reviewed on your main branch. In git, branches are cheap, so you can use them to share experimental or draft code with others. As a team you need to work out your team's process. An example process is:



Chanllenge

- Read the Wikipedia article on version control.

https://en.wikipedia.org/wiki/Version_control

Australian National University

GIT - Internals

Eric McCreath

Creating a new git repo

- To create a new repository, *cd* into the directory you wish to place under revision control (this may or may not yet contain the files you wish to keep track of) and run:

```
% git init
```

This will create the ".git" directory which will store the repository and the change history. This ".git" directory can generally be ignored.

Git Internals

The heart of git is a **key-value** store. Elements within the store are know as *objects* . The *key* used for these objects are 40 digit SHA-1 hashes. These are hashes of the header information combined with the data of the objects. The data is stored in compressed files within ".git". Use:

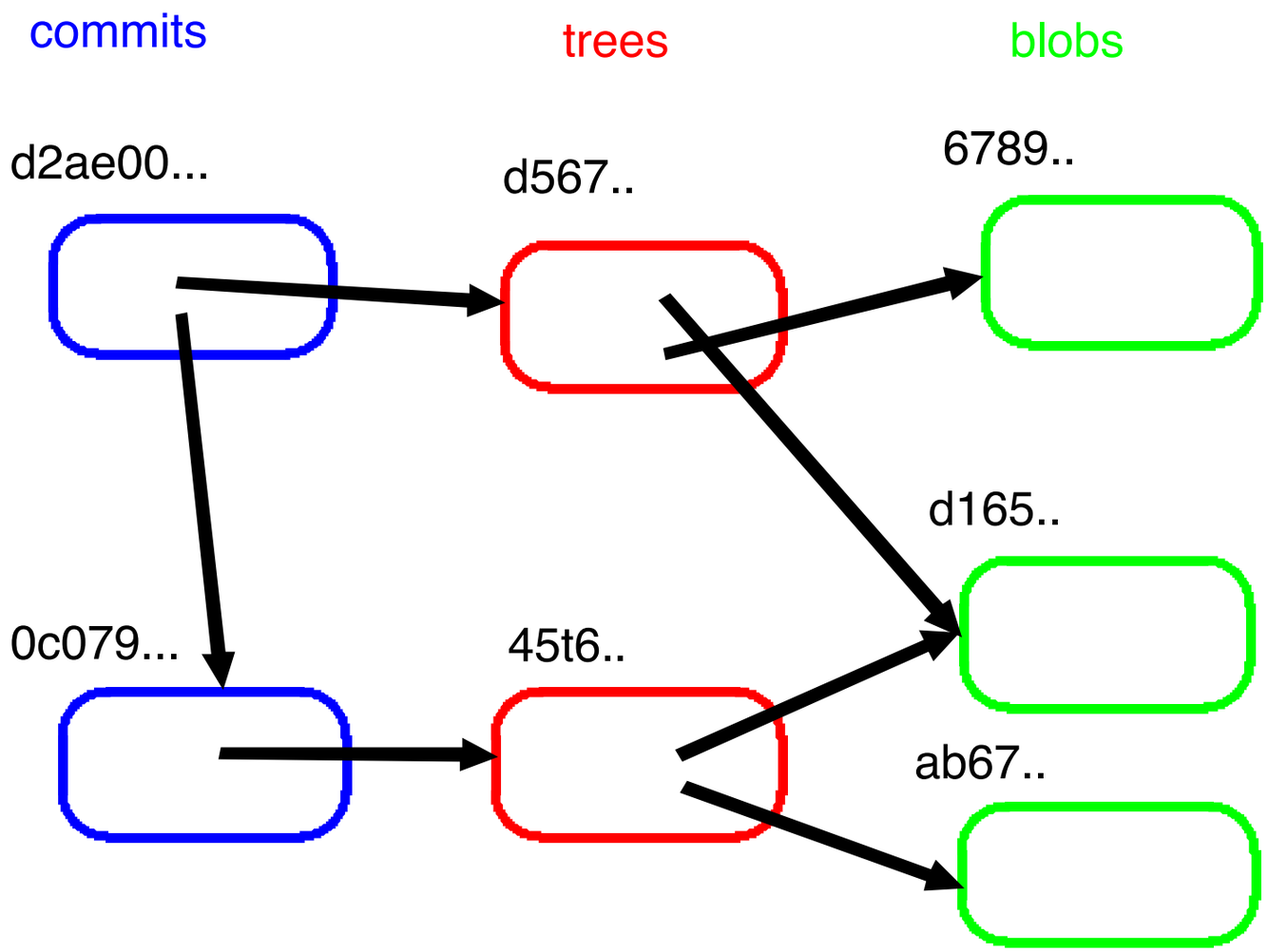
```
git cat-file -p sha-key
```

to see the contents of these objects. There are 3 types of objects:

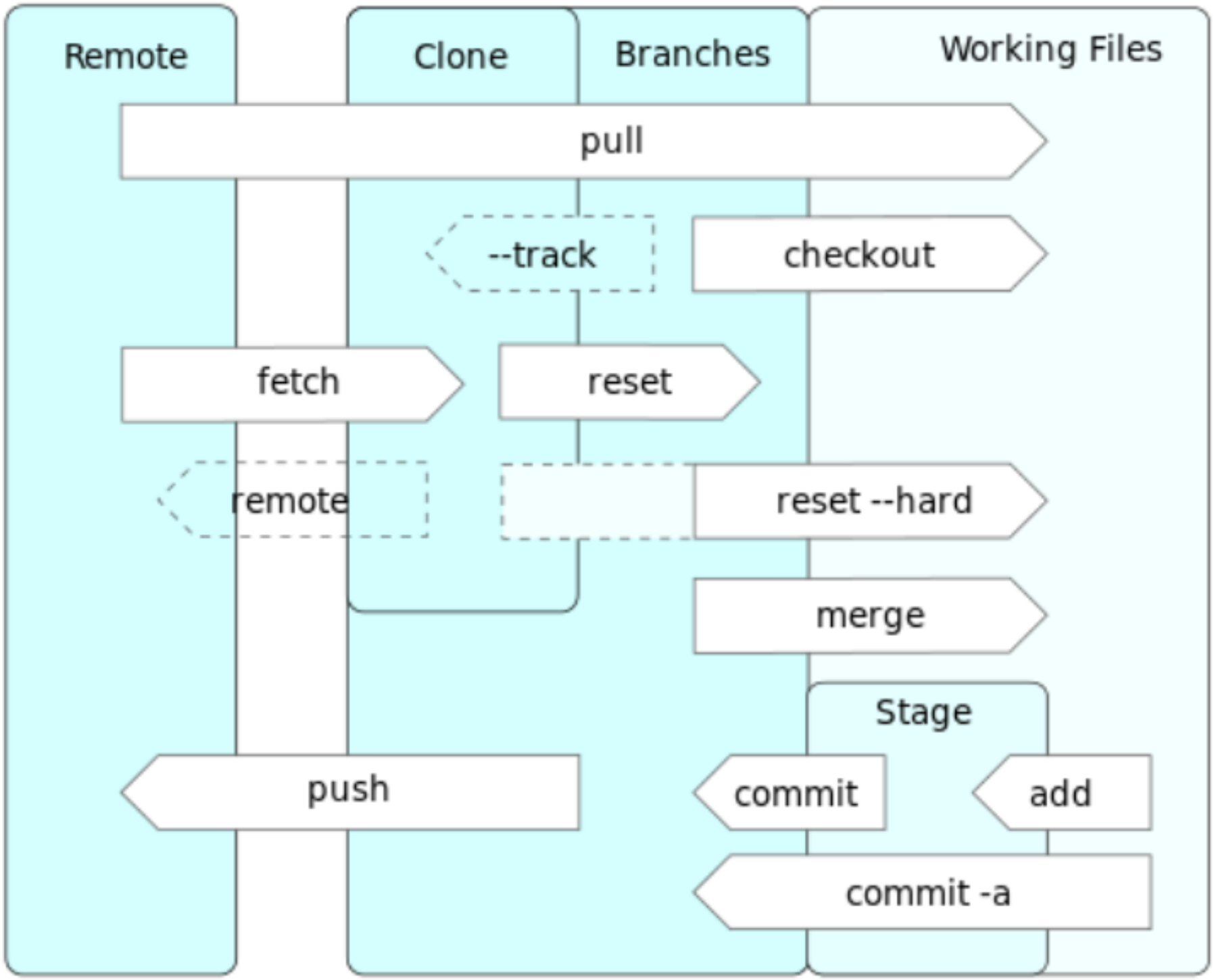
- **commits** - which have the info about a commit
- **trees** - which store info about file names and directories
- **blobs** - the data contained in files

Use of SHA-1 hashes makes changing the recorded history very difficult.

Git Internals



Git Operations



http://commons.wikimedia.org/wiki/File:Git_operations.svg

bare git repo

- A **bare** repository that does not have local *working files* can be created. It simply contains the .git directory with its compressed object database and info about tags and branch heads etc.

This can be used as a central repository that is cloned, pulled from and pushed to. To create it, use:

```
% git --bare init
```

Challenge

- Create a new git repo, add and commit a file, and explore the **.git** directory.

Australian National University

GIT - Conflicts, Branches, Merging and Tags

Eric McCreath

conflicts

When managing a team of programmers working on the same code base it is generally a good idea to have them work on unrelated parts of the code. However, occasionally people will modify the same part of the code in different ways. This creates a conflict.

Once a conflict has occurred the part that conflicted will be modified by git giving both possible versions. To resolve this conflict, the file needs to be edited and changed in a way that keeps the correct version and discards the other. Following this, a *commit* will lock in the changes.

Australian National University

branches

Say you have released your project and there is a version of the code people are using. With this version you will need to make minor bug fixes and updates. However, at the same time you may be working on the next release which involves major changes. Clearly you need two working versions to be maintained. At some point they may be merged into a new version

- Or you may have a crazy idea you would like to try out on your code. However, you are uncertain if you want your entire code base moved in this direction

- In both these cases **branches** will help. Branches in git are very *cheap*

- The disadvantage is sometimes your team may get lost in a maze of branches

branches

- Branches have names. The default name for the initial main branch is "master". However, there is nothing particularly special about this branch

To make a new branch called "crazyidea":

```
% git branch crazyidea
```

- Normally once a new branch is made, you would like to work on the code. So you should check out this branch:

```
% git checkout crazyidea
```

- Once you have done some adds and commits on this new branch and finally wish to merge it back in with the master branch, you would first switch back to the master:

```
% git checkout master
```

- And then merge crazyidea into master:

```
% git merge crazyidea
```

tags

Using *git checkout* the working files can be changed to a different branch or point in the commit history. SHA-1 hashes can be referred to directly, or more meaningful labels to certain points in the commit tree can be used. Such labels are called "tags". For e.g.:

```
% git tag Version1.1
```

would tag the current point in the commit history as "Version1.1". In future, you can *checkout* "Version1.1"

- *tags* are not pushed by default to an external repository
- *tags* can also be GPG-signed points in the commit history

Challenge

- Make a branch, commit some changes to it such that when you merge the changes back into the master branch there is a conflict that needs resolving.

Australian National University

ssh - an introduction

Eric McCreath

ssh

"ssh (SSH client) is a program for logging into a remote machine and for executing commands on a remote machine. It is intended to replace rlogin and rsh, and provide secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP ports can also be forwarded over the secure channel.

ssh connects and logs into the specified hostname (with optional user name). The user must prove his/her identity to the remote machine using one of several methods depending on the protocol version used"

- from the ssh manual (man page)

Australian National University

Authentication

ssh uses a client server model. The server on the remote machine you wish to log into listens on port 22 for clients to connect.

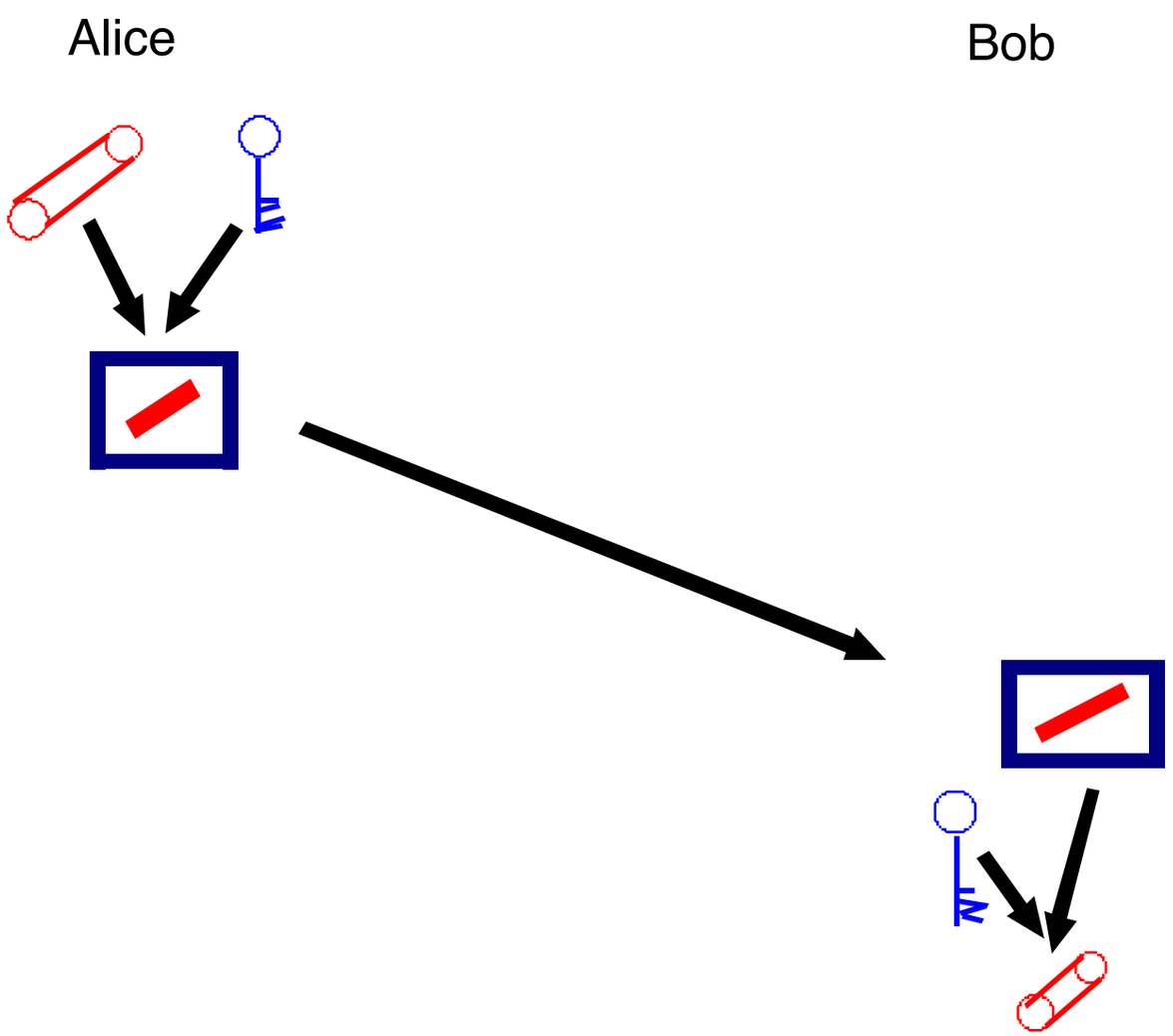
When the client connects with the server a secret session key is agreed by both parties. This session key is used for encrypting communication between the client and the server. This session key is never passed between the client and server. Rather, a Diffie-Hellman approach is used.

ssh can use a number of authentication approaches. The main ones are:

- password based authentication (assuming the remote machine permits this),
- public key authentication.

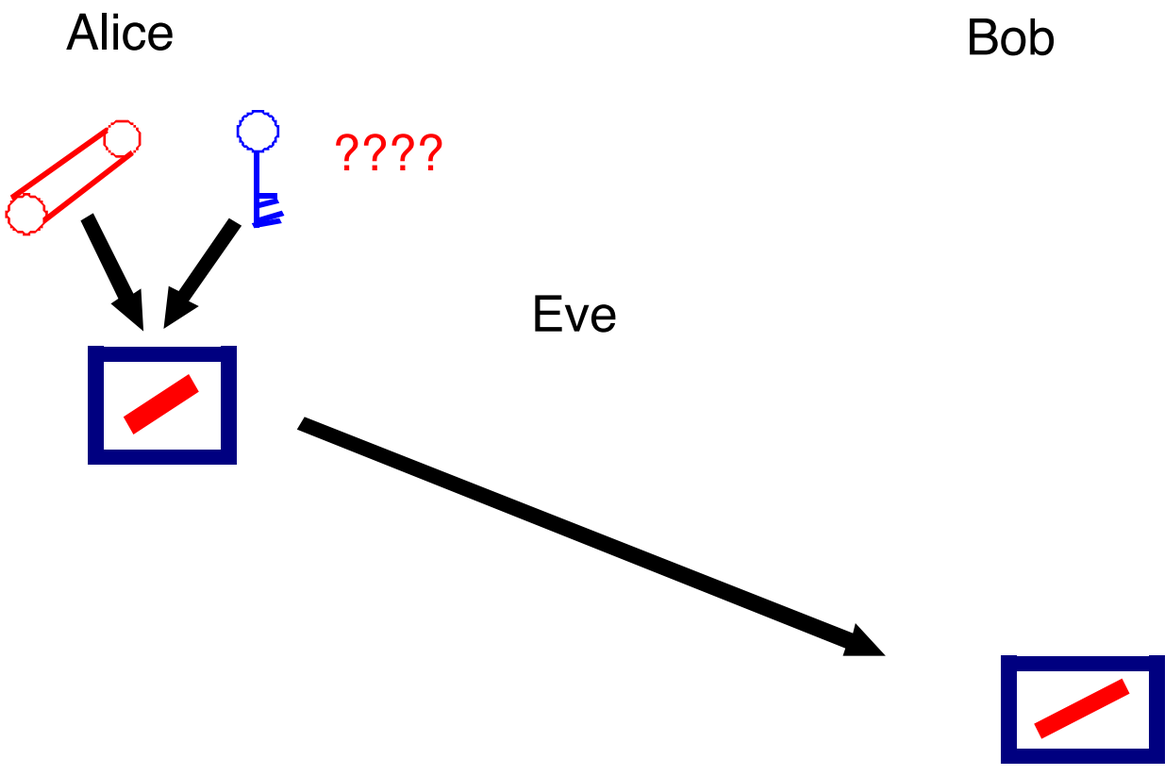
Symetric Key Cryptography

- In symmetric key cryptography the same key is used for encryption and decryption of a message. Common symmetric key approaches are: DES, 3DES, AES, and TwoFish.



Diffie-Hellman-Merkle key Exchange Scheme

- The problem with symmetric key cryptography is "How do you share keys over an unsecured channel?" In about 1976 Diffie, Hellman and Merkle came up with what is known as the Diffie-Hellman-Merkle key exchange scheme.



Diffie-Hellman-Merkle key Exchange Scheme

Alice

Both Alice and Bob know the number 5 and 7 (we can assume Eve may also know them)

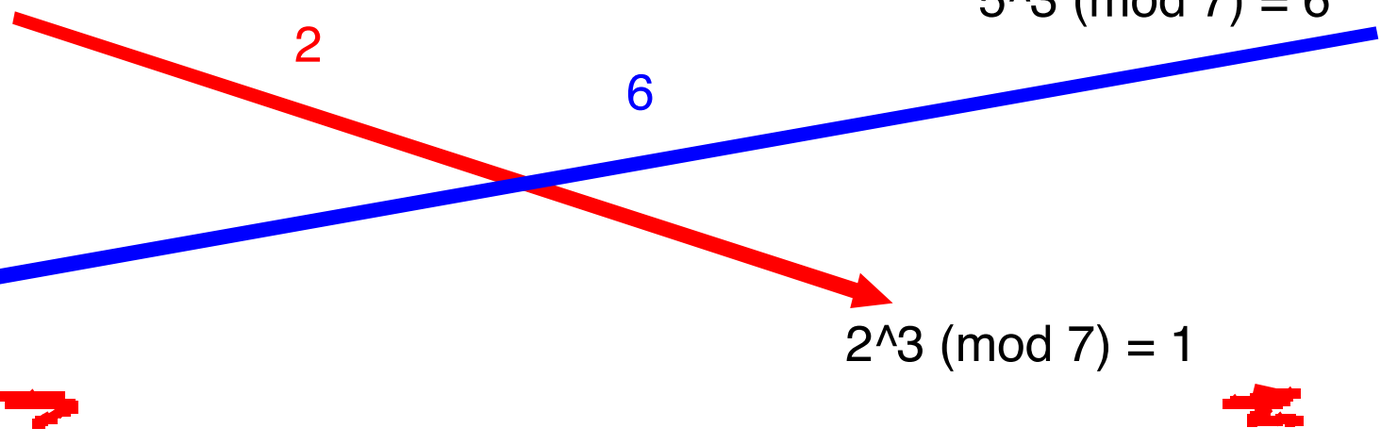
Bob

Alice thinks up a random number, say 4. This is not shared.

5^4 (mod 7) =2

Bob also thinks up a random number, say 3. This is not shared.

5^3 (mod 7) = 6



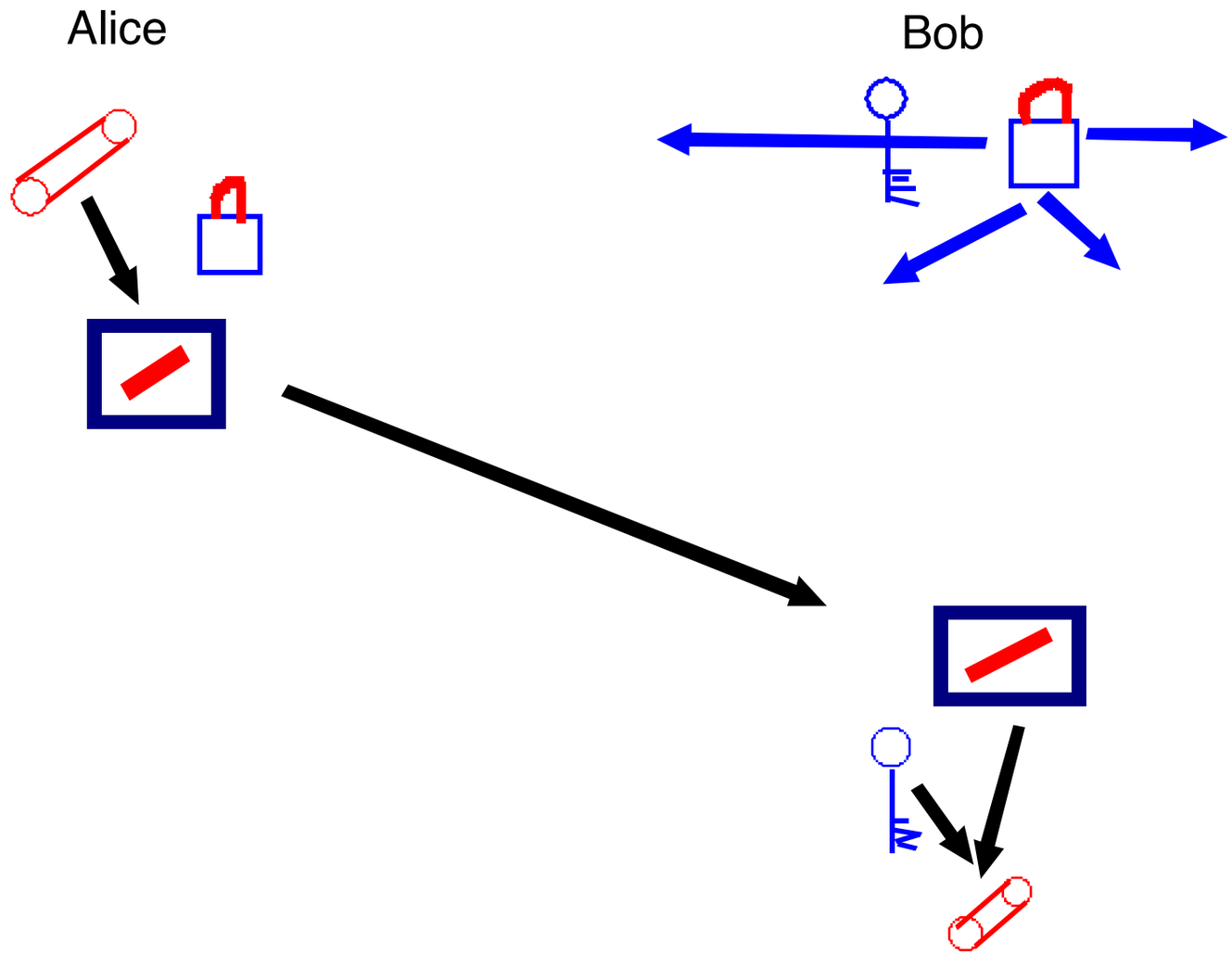
6^4 (mod 7) = 1

2^3 (mod 7) = 1

So both Alice and Bob share the same key, namely 1. Even if Eve sees 5,7,2, and 6, because "a^x (mod b)" is easy to calculate but hard to invert it is difficult for Eve to work out the shared key.

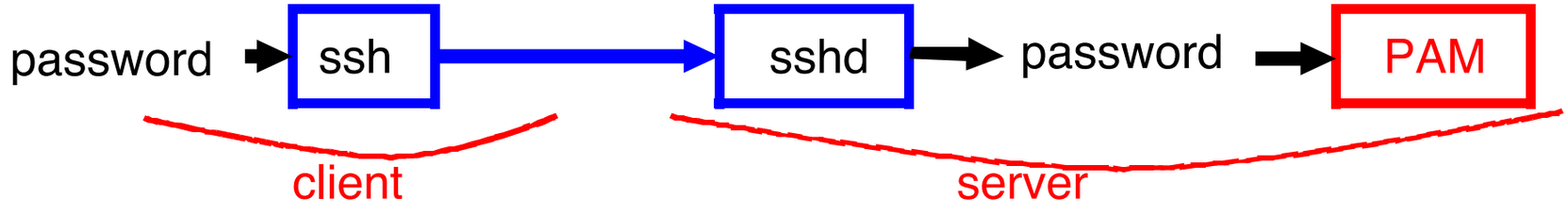
Asymetric Key Cryptography

- In asymmetric key cryptography two keys are used. One for encryption and a different one for decryption.



- In 1973 Diffie came up with this great idea of using two different keys. In 1977 Rivest, Shamir and Adleman worked out a way of implementing it (RSA).

password authentication



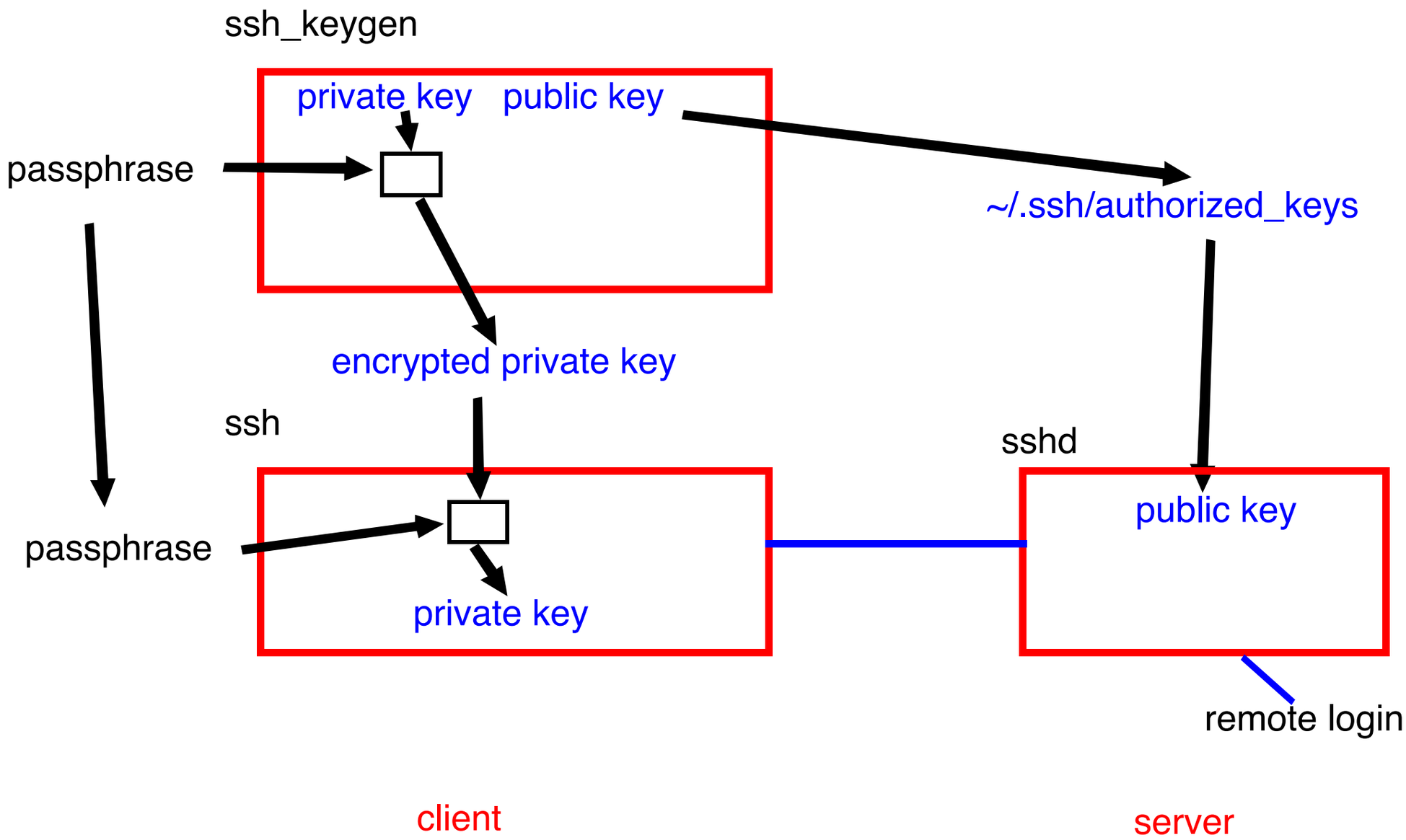
Strengths:

- simple - you don't need to store or maintain keys so you can log in from different clients,
- generally people will have a password on a unix based system when they obtain an account.

Weaknesses:

- people forget their passwords
- people use the same password for different systems
- passwords can be guessed

Public Key Authentication



Public Key Authentication

Public Key Authentication provides you with 2 factors. Namely, something you know, the passphrase, and something you have, the encrypted private key. This approach, combined with an authentication agent, such as ssh-agent, provides one with a convenient and secure way of accessing a remote machine.

Weaknesses of this approach include:

- the server can not check that strong passphrases are used (or even if a passphrases is used at all!)
- if someone is able to alter the authorized_keys file then they could include an extra key which may not be detected.

Challenge

- Read the man page for ssh. Focus on:
  - first 2 paragraphs of the **DESCRIPTION** ,
  - the **AUTHENTICATION** section,
  - the **TCP FORWARDING** section, and
  - the **VERIFYING HOST KEYS** section.

Australian National University

ssh - port forwarding

Eric McCreath

Port Forwarding

ssh enables you to forward ports between a client machine and a remote server. This enables you to obtain a secure connection for what may be an insecure service. It also enables you to access resources that sit behind a firewall.

For e.g. Say I run a web server at home for sharing files between my various devices. It sits behind a firewall so it is not exposed to the internet. However, I do permit ssh (port 22) access from external machines to my home server. If I wanted to access my home web server from work, I could create a tunnel using ssh with:

```
% ssh -L 5432:localhost:80 myname@myhomecomputeripaddress
```

X11 forwarding can be done (if the server permits) using ssh.

Australian National University

ssh - remote execution

Eric McCreath

Running a script remotely

- Sometime you wish to be able to execute commands/scripts on a remote machine. Running commands remotely creates potential security problems, so it needs to be done carefully.
- Running commands remotely is useful for a variety of purposes including:
  - backups,
  - checking a server is running properly,
  - giving controlled access to a server on the machine (like git or hg),
  - doing something that can only be done from a particular machine (compiling on a fast machine, testing, running software with lience ties it to a particular machine, accessing an I/O device).

Using ssh to run a command remotely

- This gives you the flexibility to enable other uses access resources on your account in a controlled way.
- Normally when you ssh into a machine the ssh daemon will execute your login shell (many people use bash). However if you specify a command then it will be run on the remote machine. e.g.

```
$ ssh u4033585@partch.anu.edu.au lpq
CSITFoyer is ready
no entries
```

- Within **authorized_keys** you can also specify the command that is run when access is granted via a particular key. This is enormously flexible. Giving limited/controlled access for yourself and others (or even other programs) to a remote machine you have access to.

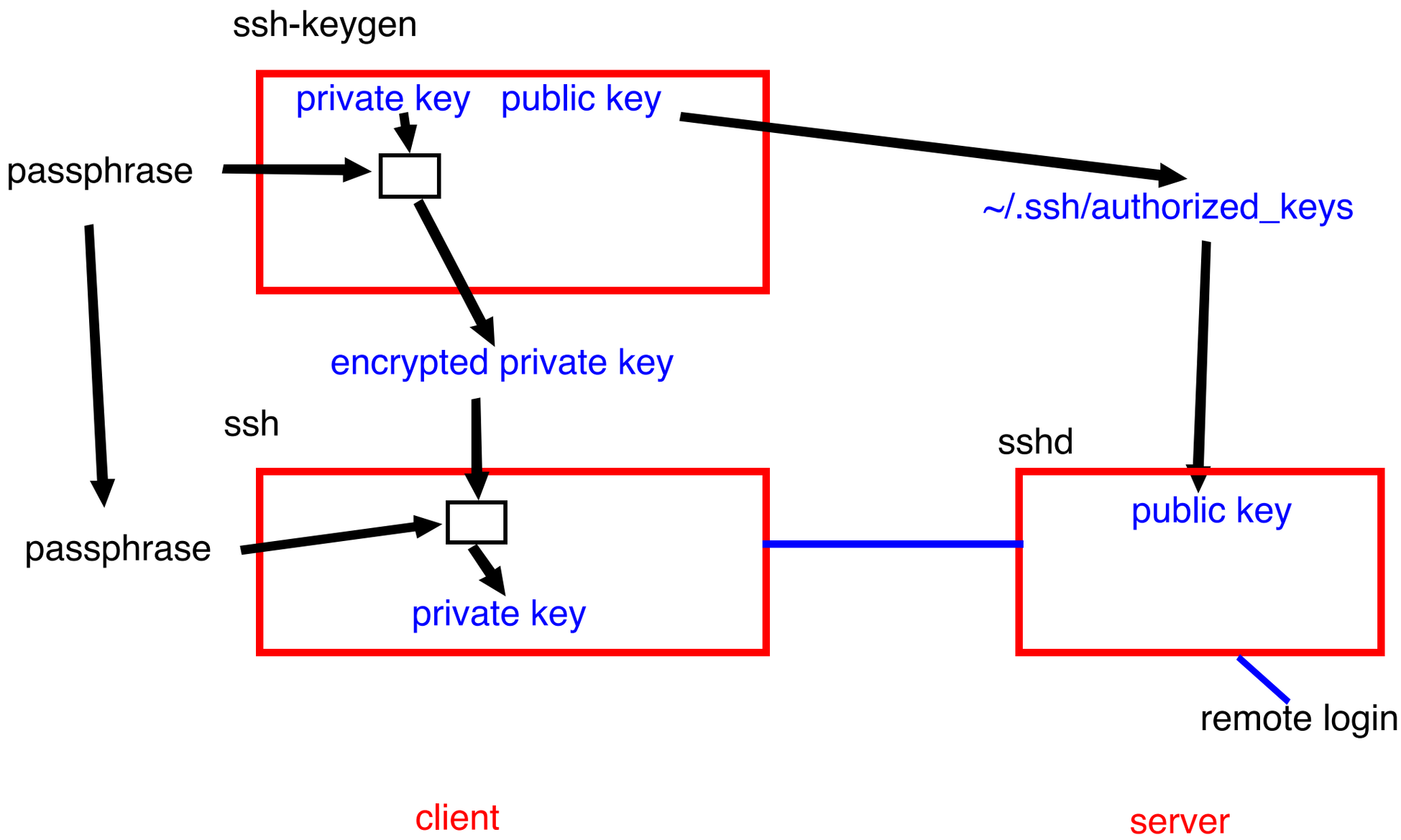
```
command="cd /home/web/ericm/public_html; git pull",no-port-forwarding,
no-X11-forwarding,no-agent-forwarding ssh-rsa AAAAB3NzaC1yc2EA......
```

Australian National University

ssh - setting up keys

Eric McCreath

Public Key Authentication



authorized_keys

The ~/.ssh/authorized_keys file holds a list of public keys that are authorized for logging in as the user.

Lines starting with "#" are ignored, and can be used for comments. Each line in the file has: options (which are optional), key type, encoded public key, and a comment. Options are comma separated **without** spaces and can be used to control and constrain what the user can assess with that particular key.

Two useful options include:

- command="command" which specifies the command to be used when this key is used,
- from="pattern-list" which only permits authentication from particular host name or IP address.

see: 'man sshd' for the details of the format and options.

ssh_config

The ssh "-i" option enables you to select a specific private key to use.

Note the default is ~/.ssh/id_rsa or ~/.ssh/id_dsa.

Also a configuration file may be used to control which private key to use for which server.

scp and rsync

The **scp** command uses **ssh** to provide a secure remote file copy program. The command line option works in a similar way to the normal **cp** command however you man also specify a remote user and host.

e.g. To copy a file to my RSCS account I could:

```
$ scp file.txt u1234567@partch.anu.edu.au:
```

rsync is a fast and versatile tool for copying files and directories. It is designed to reduce the amount of data sent over the Internet by only sending the files that are different. It may use **ssh** as the remote-shell program for the transport.

It was originally developed by our own Andrew Tridgell and Paul Mackerras.

Challenge

- Set up a public-private key pair on your computer. Then:
  - add the public key to authorized keys on partch.anu.edu.au ,
  - use this to login and scp a file to/from partch, and
  - add the public key to your gitlab account and use it for cloning a repo.