



# Composer

## 入門與應用

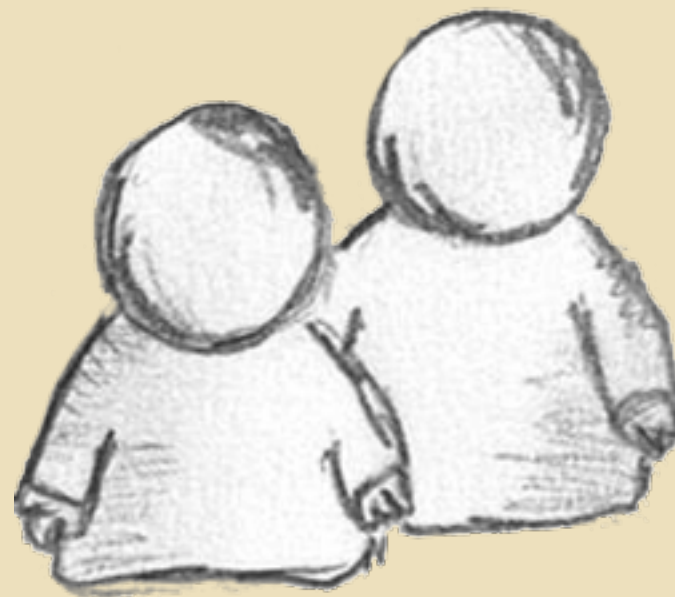
The background of the slide is a close-up, front-facing view of the Iron Man helmet. The helmet is primarily gold with red and blue accents. The eyes are glowing blue. The text is centered over the helmet.

**Jace Ju**  
**大澤木小鐵**

**KKBOX**  
**Senior Developer**  
**@jaceju**

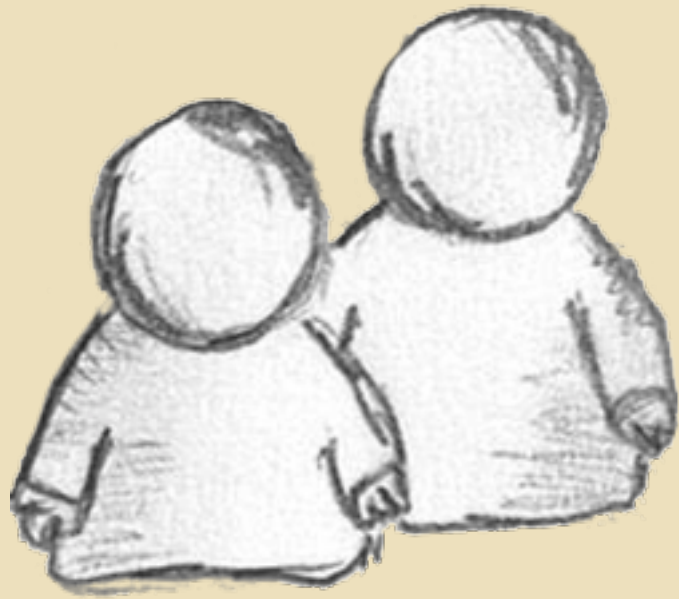
使用情境

# 曾經有一個開發團隊



A-Team

# 他們用 PHP 開發了一個新專案

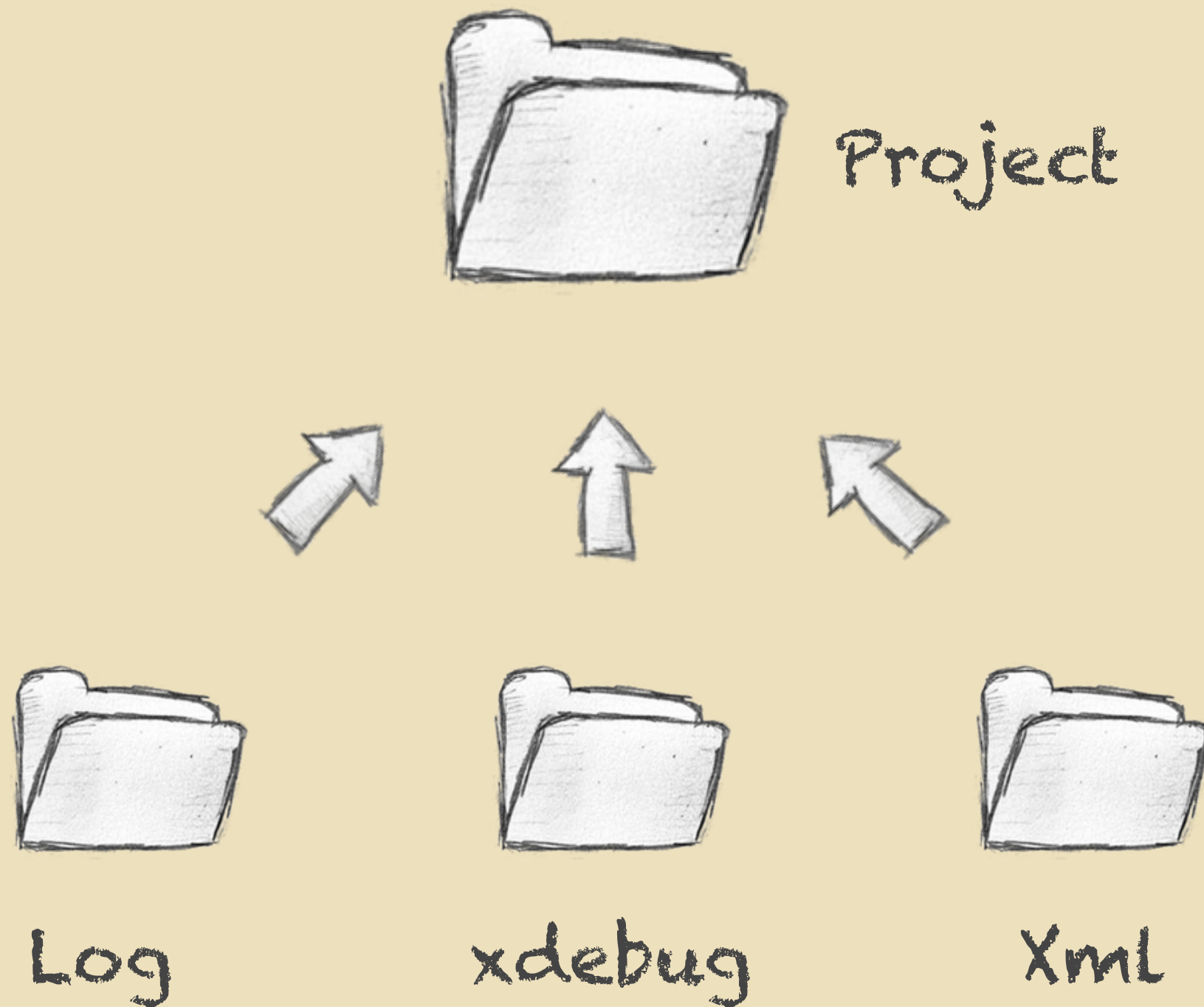


A-Team



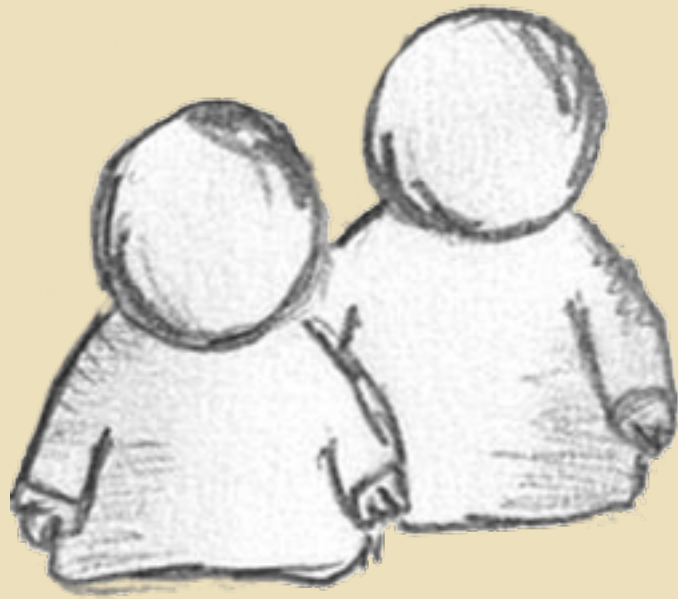
Project

# 這個專案用到了一些 PHP 套件





# 專案開發中，加了一位新人



A-Team



Newbie

# 他拿到了專案的原始碼



Newbie



Source Code



# 花了很多時間還是沒辦法讓專案執行

localhost



Source Code

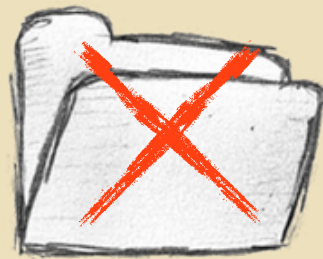
因為他不知道自己還有套件沒安裝



Project



Log



xdebug



Xml

這浪費掉很多工作時數



# 所以我們需要知道專案用了哪些套件



Project



Log

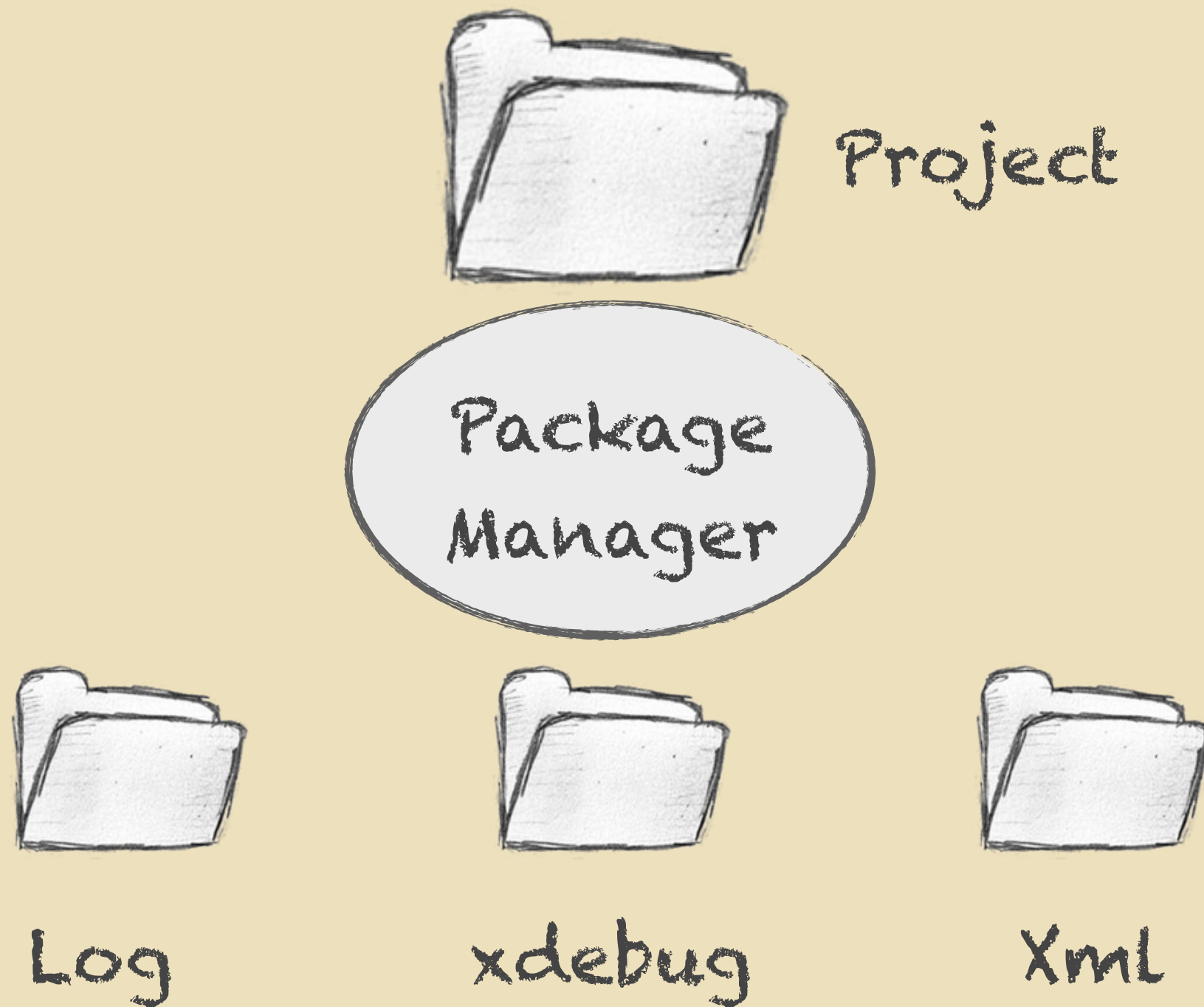


xdebug



XML

# 套件管理就是幫我們做好這件事



有哪些套件管理系統？



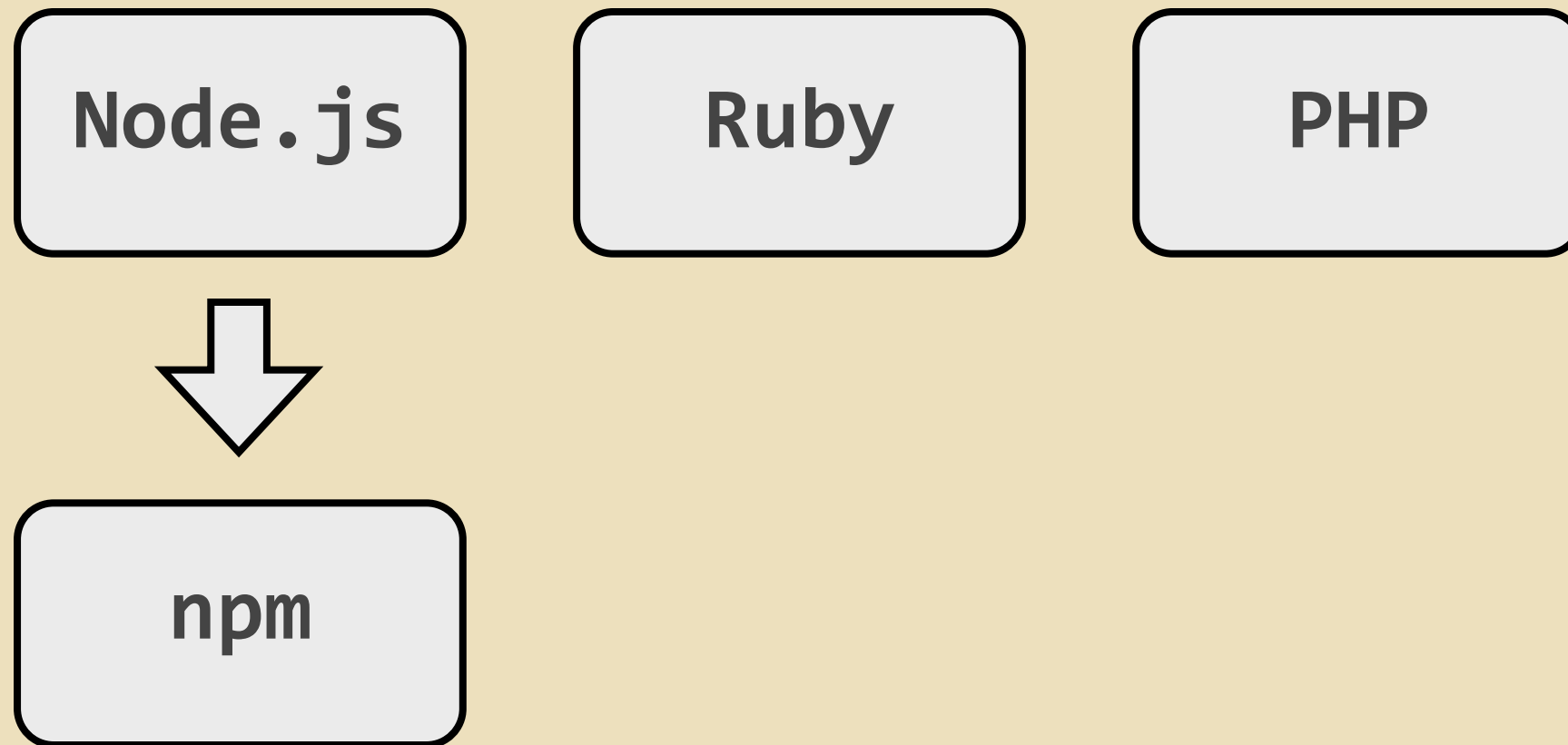
# 常見語言的套件管理系統

**Node.js**

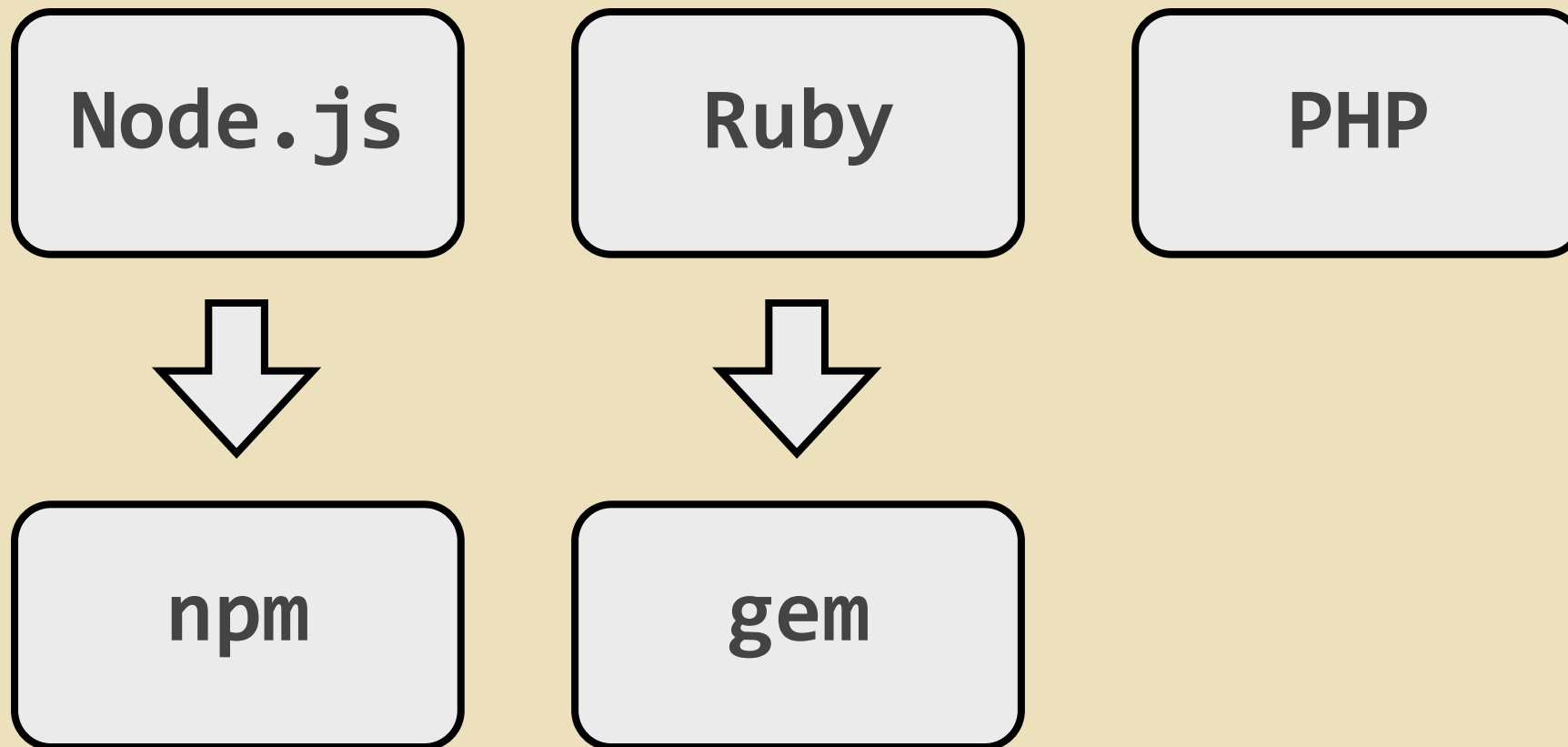
**Ruby**

**PHP**

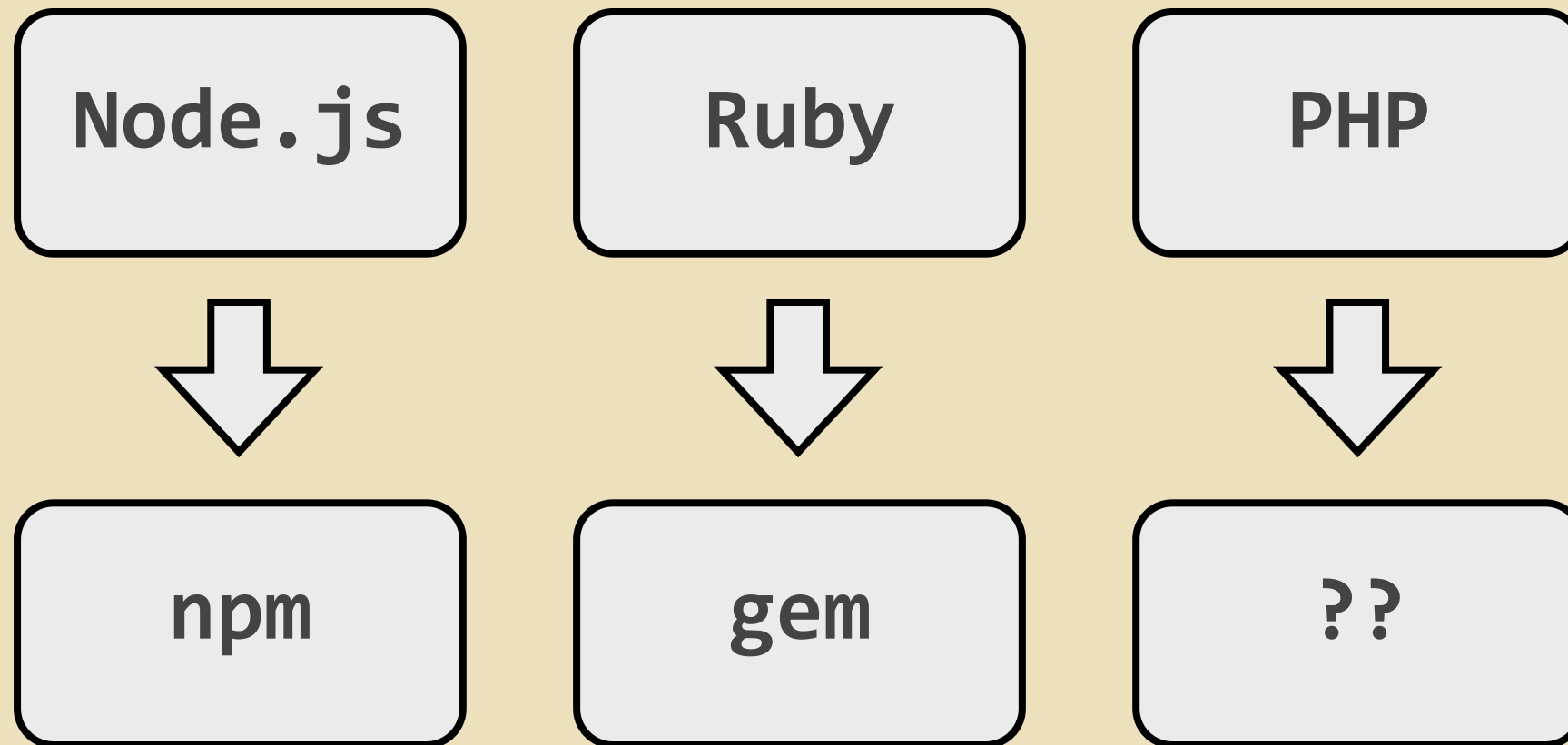
# 常見語言的套件管理系統



# 常見語言的套件管理系統



# 常見語言的套件管理系統



# PHP 的套件管理系統

# PHP 的套件管理系統

The PEAR logo is a light blue rounded square with a black border. The word "PEAR" is centered inside in a bold, dark gray, sans-serif font.

**PEAR**



# PHP 的套件管理系統

**PEAR**

**Pyrus  
(PEAR2)**

# PHP 的套件管理系統

**PEAR**

**Pyrus  
(PEAR2)**

**Composer**

# 為什麼不用 PEAR / Pyrus

- 套件管理是針對整個系統而非專案
- 沒辦法檢查執行環境
- 套件審核機制複雜
- 設定檔編寫複雜（XML）
- 必須使用官方的套件庫或自行架設
- 不潮了

# 為什麼要用 Composer


- 仿 npm / gem 的套件相依管理
- 針對專案的套件做管理
- 可以檢查執行環境
- 設定檔編寫簡單 (JSON)
- 支援用常見的版本控制系統做為  
套件下載來源 (SVN / GIT)

# 套件相依觀念

套件如何組織？



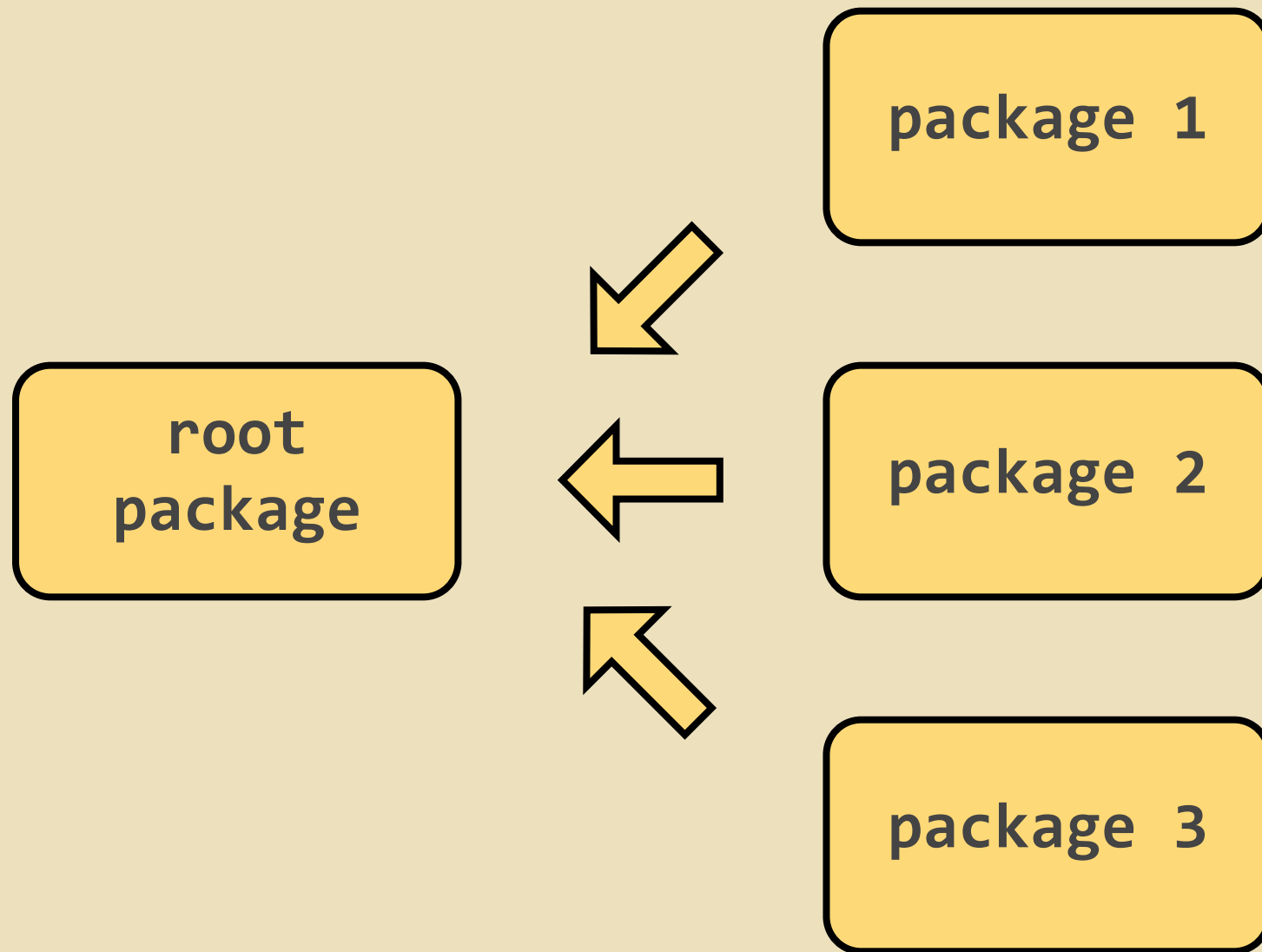
# 套件如何組織？



**root  
package**

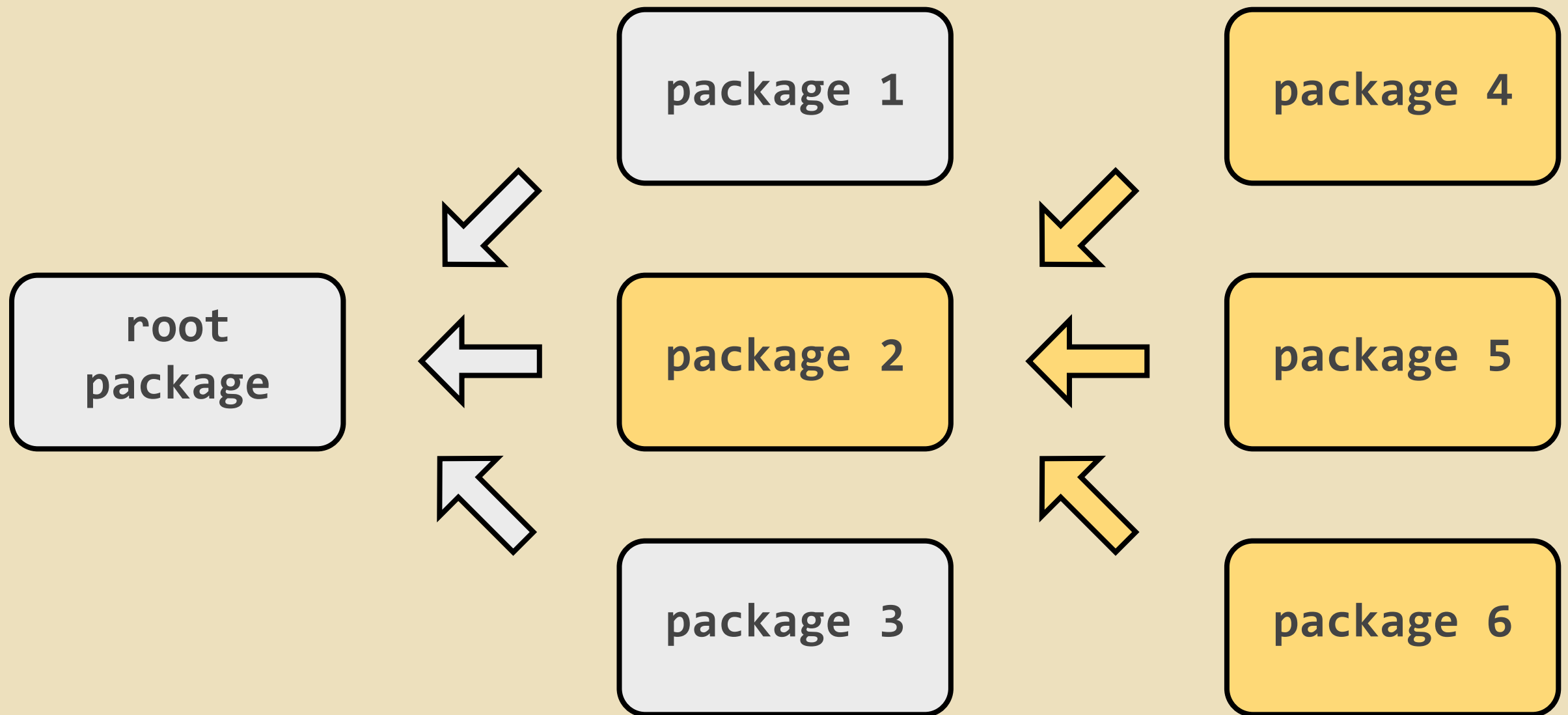
root-package 就是我們的專案

# 套件如何組織？



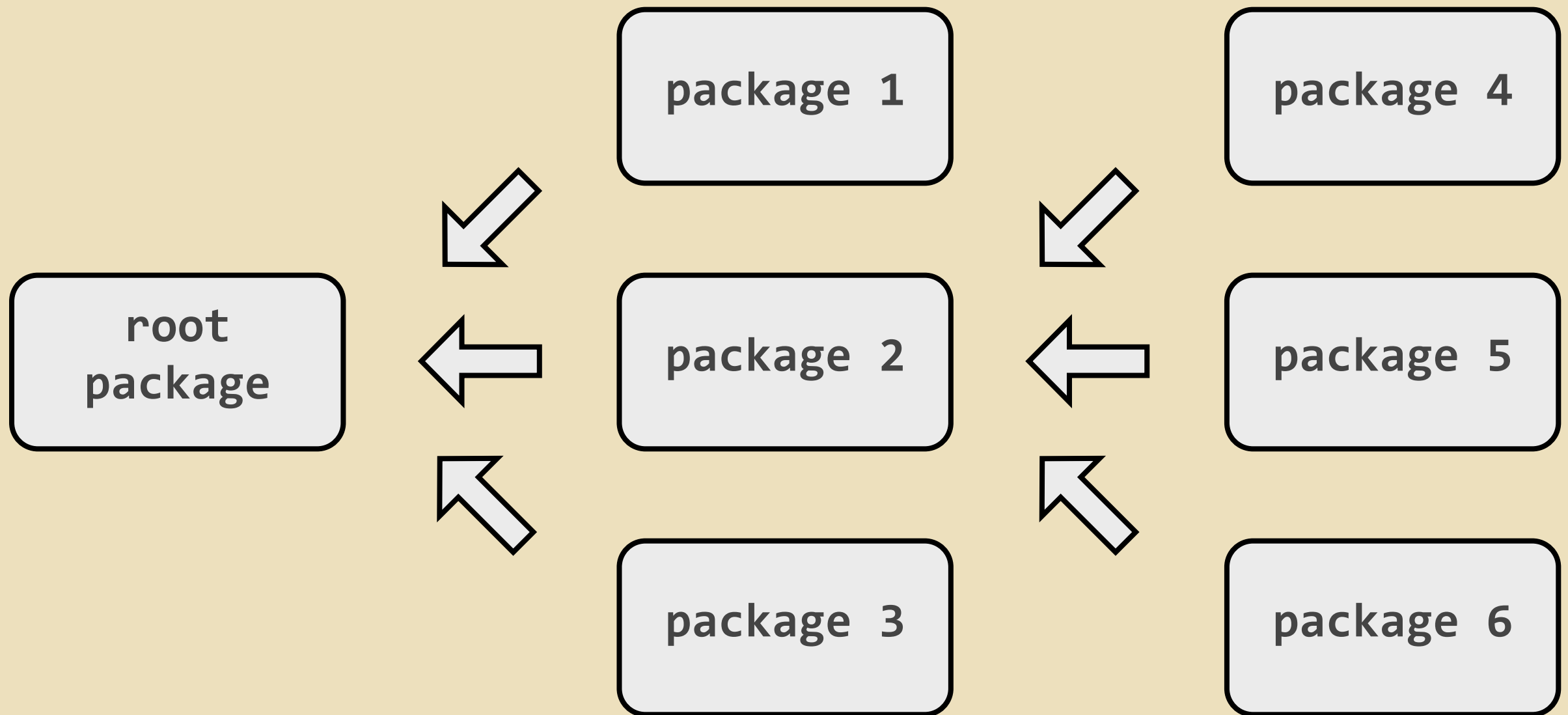
我們的專案相依了三個套件

# 套件如何組織？



package 2 則又相依其他套件

# 套件如何組織？



在 root package 操作就好  
Composer 會協助我們處理剩下的相依問題

# 小提醒

- composer 的 package 基本上可分為
  - application : 應用程式
  - library : 函式庫
- application 通常是 root-package
- application 通常相依多個 libraries
- library 也有可能相依多個 libraries

安裝 Composer



# \*nix 環境安裝

```
$ curl -sS https://getcomposer.org/installer | php
```

下載官方的安裝指令，並透過 PHP 安裝  
通常會安裝在專案根目錄下，  
安裝好的檔案為 composer.phar

# \*nix 環境安裝

```
$ curl -sS https://getcomposer.org/installer | php
```

```
$ php composer.phar <command>
```

在專案根目錄下，透過 php 來執行 composer.phar

# \*nix 環境安裝

```
$ curl -sS https://getcomposer.org/installer | php
```

```
$ php composer.phar <command>
```

```
$ sudo mv composer.phar /usr/local/bin/composer
```

如果希望讓所有人都可以使用 composer  
就將 composer.phar 搬到共用執行目錄，  
例如 /usr/local/bin

# \*nix 環境安裝

```
$ curl -sS https://getcomposer.org/installer | php
```

```
$ php composer.phar <command>
```

```
$ sudo mv composer.phar /user/local/bin/composer
```

```
$ composer <command>
```

這樣就可以直接執行 composer

若不能執行，則要以 chmod 加入可執行權限

# Windows 環境安裝

**Composer-Setup.exe**



**<https://getcomposer.org/Composer-Setup.exe>**

下載回來執行，一直點下一步就可以

# Composer 有什麼功能

```
$ composer list
```

顯示有哪些指令和選項可用

# Composer 有什麼功能

```
$ composer list
```

```
$ composer help <command>
```

顯示指令的詳細用法

# 將 Composer 套用到 PHP 專案上



# 初始化

```
$ cd /path/to/project
```

```
$ composer init
```

- name
- description
- authors
- minimum-stability
- license
- require
- require-dev

# name

- 格式： **"vendor/package-name"**
- **vendor** ：開發者（公司）名稱
- **package-name** ：套件名稱

# description

- 簡短的套件描述
- 儘量在一行內結束
- 一定要輸入

# authors

- 格式： **"Name <email>"**
- 可以有多个 **author**（需手动加入）
- 预设抓取 **~/.gitconfig** 的设定

# minimum-stability

- 用來過濾相依套件穩定度的設定
- 只能用在 **root-package**
- 可設定的值：  
（由最穩定到最不穩定的順序）
  - **stable**（預設）
  - **RC**
  - **beta**
  - **alpha**
  - **dev**
- 相依的套件之穩定度不可低於設定值

# license

- 指定套件的授權
- 可參考 **SPDX** 的 **identifier** 來設定
  - Apache-2.0
  - BSD-2-Clause
  - MIT
  - ...
- 私有專案可用 **proprietary**

# require

- 格式：`"vendor/package-name": "version"`
- 指定專案一定要安裝的套件或平台環境
- 如果現存的套件版本不符合 `version` 條件的話，  
就不會安裝任何套件

# require-dev

- 格式同 **require**
- 主要指定開發用的套件
- 只能用在 **root-package**



# composer.json

```
{
    "name": "jaceju/first-app",
    "description": "My First Application",
    "require": {
        "psr/log": "*"
    },
    "require-dev": {
        "phpunit/phpunit": ">= 3.7.0"
    },
    "license": "MIT",
    "authors": [
        {
            "name": "Jace Ju",
            "email": "jaceju@kkbox.com"
        }
    ],
    "minimum-stability": "dev"
}
```

# 小提醒

- 所有 package 都需要 **composer.json**
- **composer.json** 的部份屬性只有在 **root-package** 才有作用

套件版本

# 套件版本的規則

- 數字版本以 **semantic versioning** 方式指定
- 版本號寫法分為：
  - **Exact version** : 明確指定版本，例如 "1.0.1"
  - **Range** : 指定版本範圍，可用 ">", ">=", "<", "<=" 及 "!="，以 "," 分隔  
例如 ">= 1.0.1" 或 ">= 1.0, < 2.0"
  - **Wildcard** : 萬用字元，例如 "1.0.\*"，  
等同於 ">= 1.0, < 1.1"
  - **Next Significant Release** : 下一個重要版本；  
即版本號倒數第二位數字加 1，例如：  
"~1.2"，等同於 ">= 1.2, < 2.0"；  
"~1.3.1"，等同於 ">= 1.3.1, < 1.4.0"

# 其他規則

- 在版本號後面加上 **@<stability>** ，非必要；  
例如： **"1.0.\*@beta"** 或 **"@dev"**
- 分支表示法： **<stability>-<branch>** ；  
例如： **"dev-master"**
- 版號表示法： **#<ref>** ；  
例如： **"dev-master#2eb0c..."**  
或 **"1.0.x-dev#abc123"**

# 安裝與更新

# \$ composer install

Loading composer repositories with package information

Installing dependencies (including require-dev)

- Installing psr/log (dev-master fe0936e)

Cloning fe0936ee26643249e916849d48e3a51d5f5e278b

- Installing sebastian/diff (dev-master 1e09170)

Cloning 1e091702a5a38e6b4c1ba9ca816e3dd343df2e2d

(中略)

- Installing phpunit/phpunit (dev-master 900b501)

Cloning 900b501d8bd3e80da58dc6ff3cc9b01393149717

phpunit/phpunit-mock-objects suggests installing ext-soap (\*)

phpunit/phpunit suggests installing phpunit/php-invoker (>=1.1.0)

Writing lock file

Generating autoload files

# vendor 資料夾

- 套件預設會安裝在 **vendor** 資料夾
- 自動載入的載入器與快取檔放在 **vendor/composer** 下

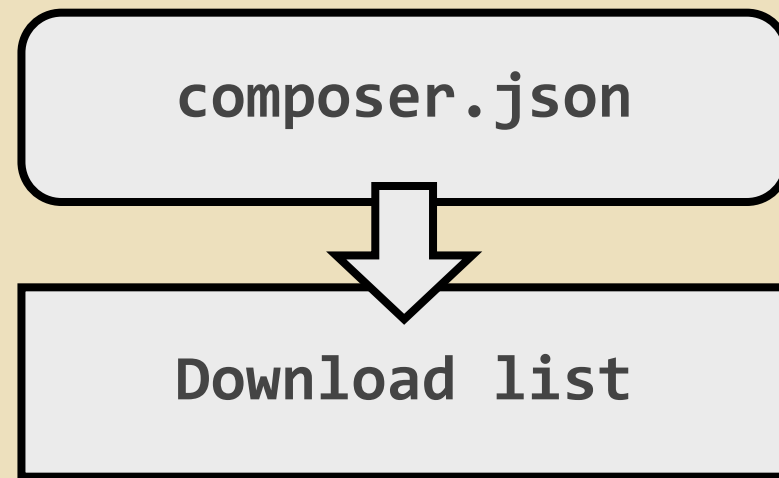


# 全新安裝（專案負責人）

`composer.json`

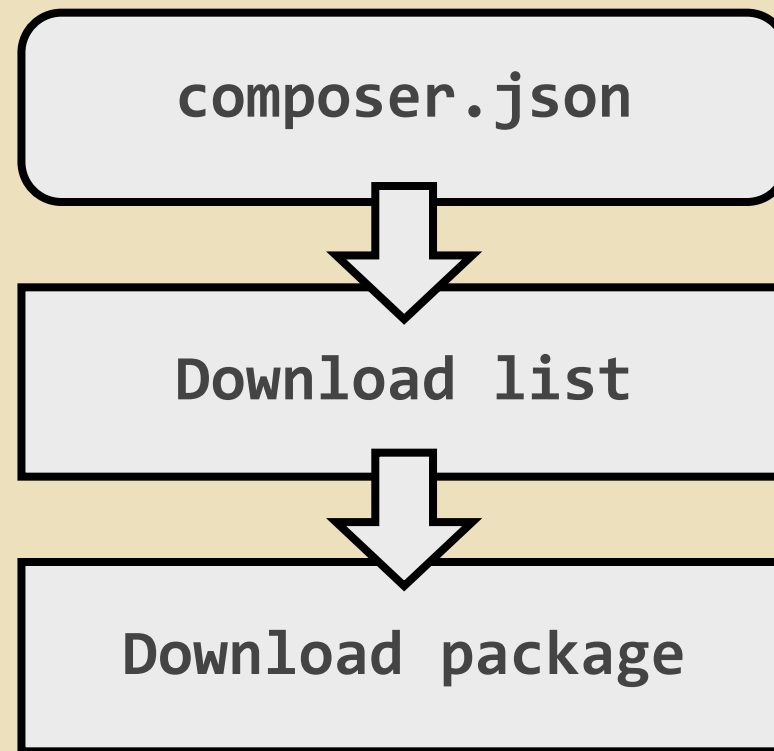
第一次全新安裝會參考  
`composer.json`

# 全新安裝（專案負責人）



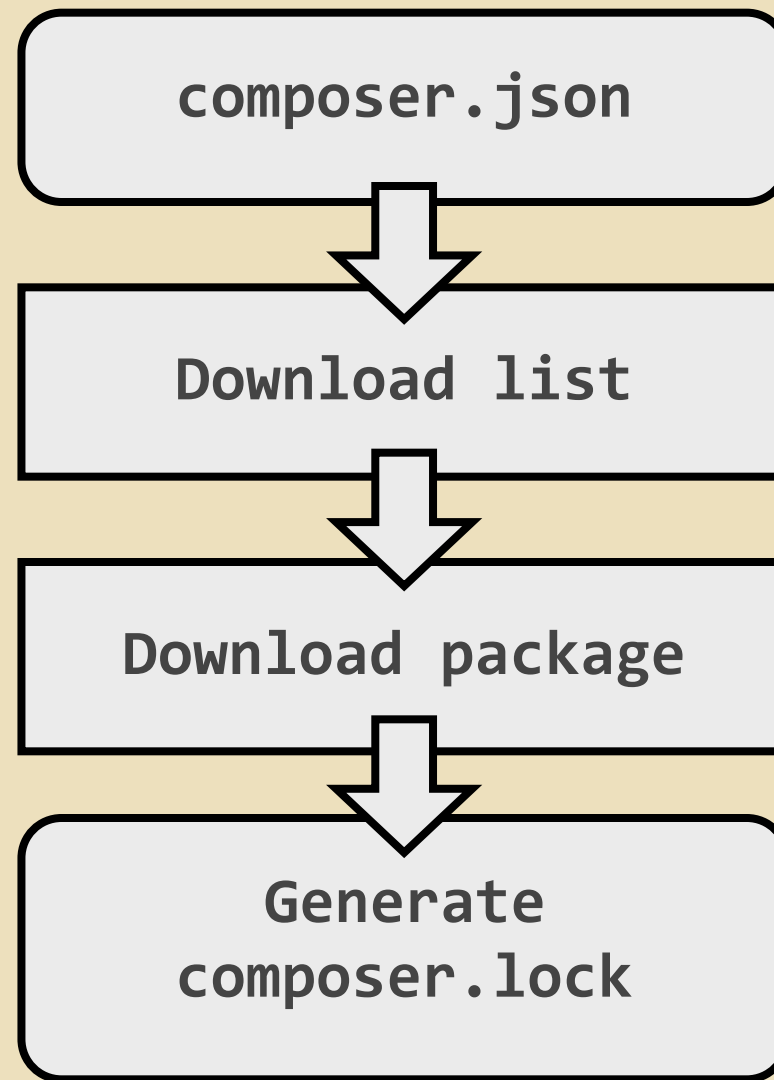
接下來就會去官方套件庫下載清單

# 全新安裝（專案負責人）



找到符合版本設定的套件並下載

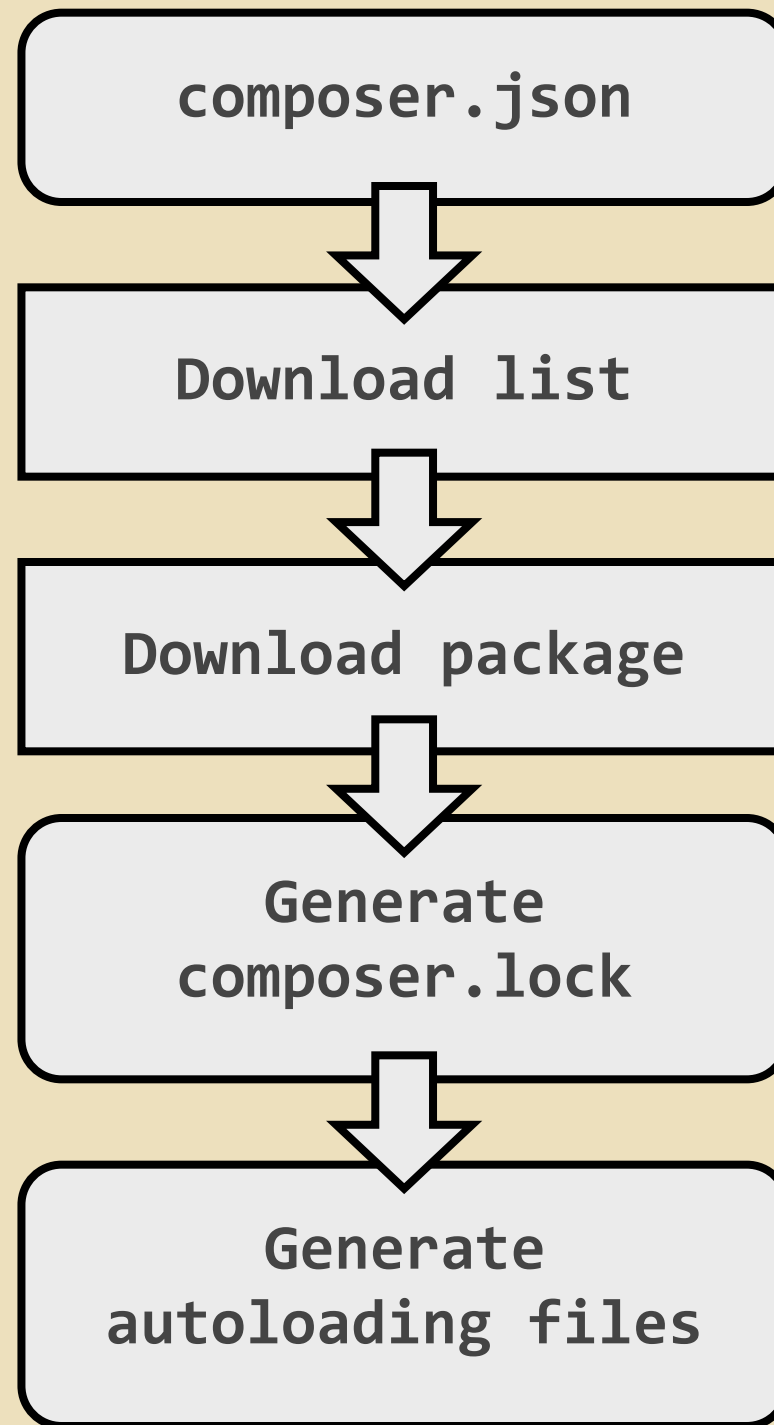
# 全新安裝（專案負責人）



產生 composer.lock

並記住每個套件在此次安裝時的版本，  
這樣一來就不用再從清單裡找符合的版本

# 全新安裝（專案負責人）



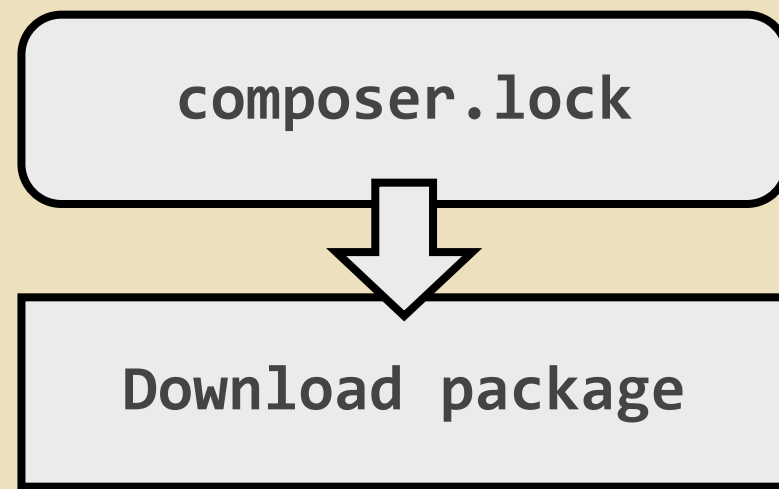
將所有相依套件的自動載入設定  
加到快取檔案中

# 再次安裝（其他成員安裝）

`composer.lock`

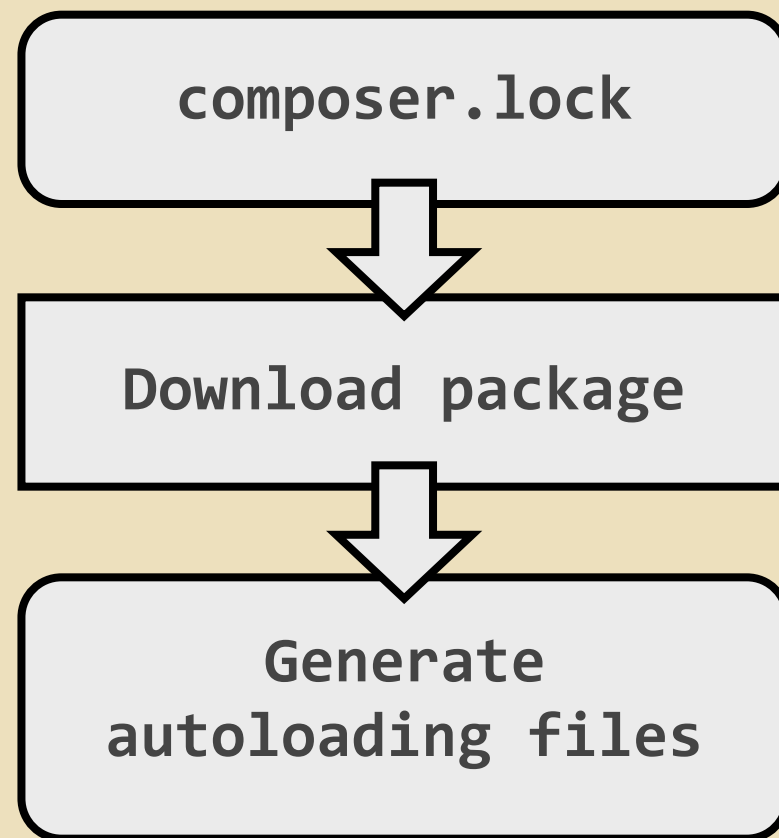
直接找 `composer.lock` 記錄的版本

# 再次安裝（其他成員安裝）



然後從快取或來源中取得套件檔案

# 再次安裝（其他成員安裝）



將所有相依套件的自動載入設定  
加到快取檔案中



# \$ composer update

```
Loading composer repositories with package information  
Updating dependencies (including require-dev)  
Generating autoload files
```

類似第一次 install 的動作，但會找符合條件且較新的版本

# 小提醒

- 只有專案負責人才做第一次 **install** 及之後的 **update** (會更動 **composer.lock** )
- 其他成員一律使用 **install**
- 第一次 **install** 及之後的 **update** 會將所有相依套件的資訊編寫到 **composer.lock** 上，並固定套件版本
- **--prefer-source** ：從版本控制下載
- **--prefer-dist** ：下載壓縮檔

# 版本控制

- **root-package** 的 **composer.json** 及 **composer.lock** 要放到版本控制系統裡，這麼一來其他成員可以直接使用
- **library** 的 **composer.lock** 則不要放到版本控制系統中
- **vendor** 資料夾不要放到版本控制系統，應該用 **composer install** 安裝

官方套件庫

# packagist.org

- 記錄常見的 Open Source library 資訊
- 預設 composer 搜尋套件的地方
- 不須審核，人人都可以註冊自己的套件
- 不是實際存放套件內容的地方

# 運作方式

composer

Packagist

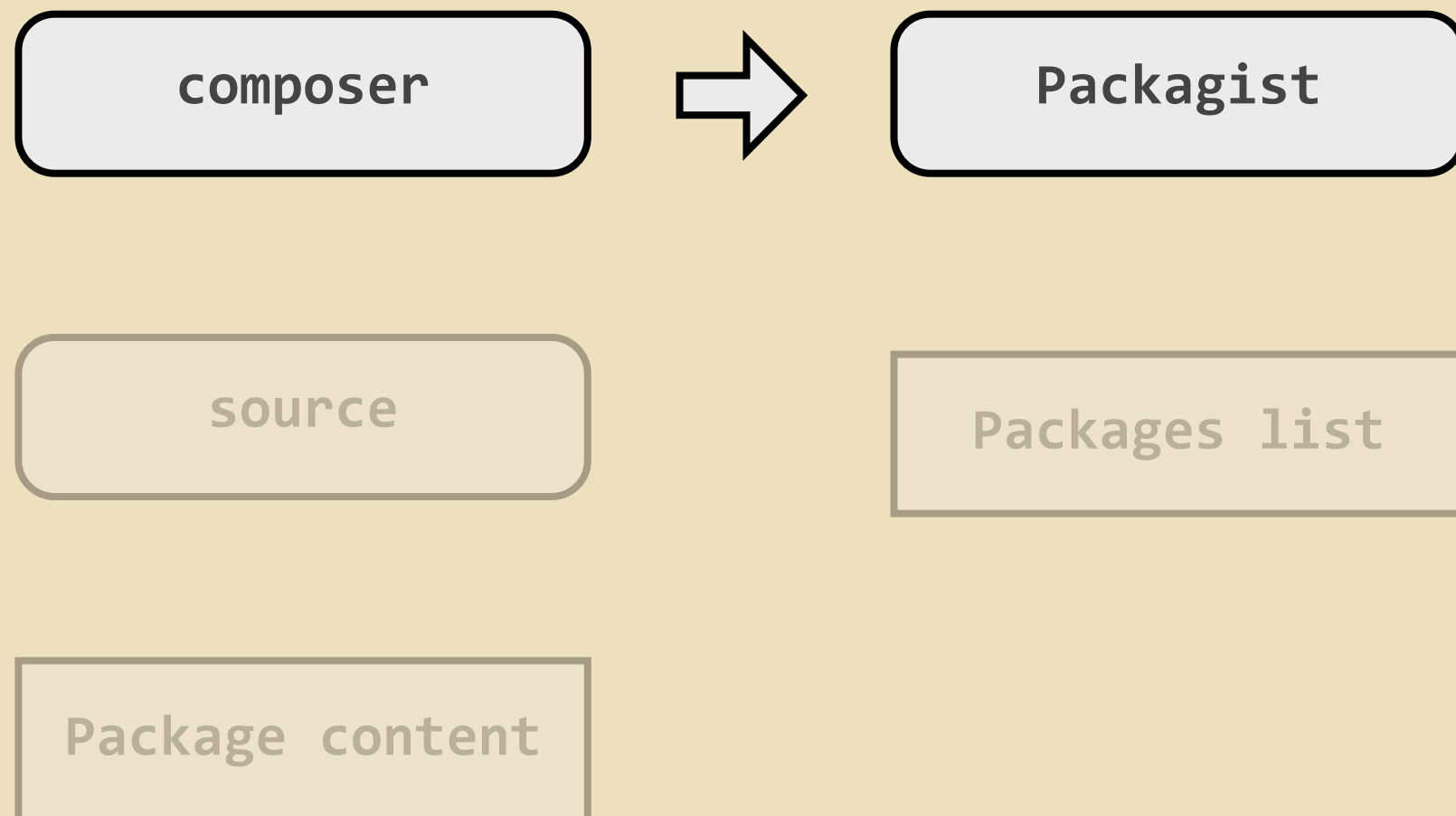
source

Packages list

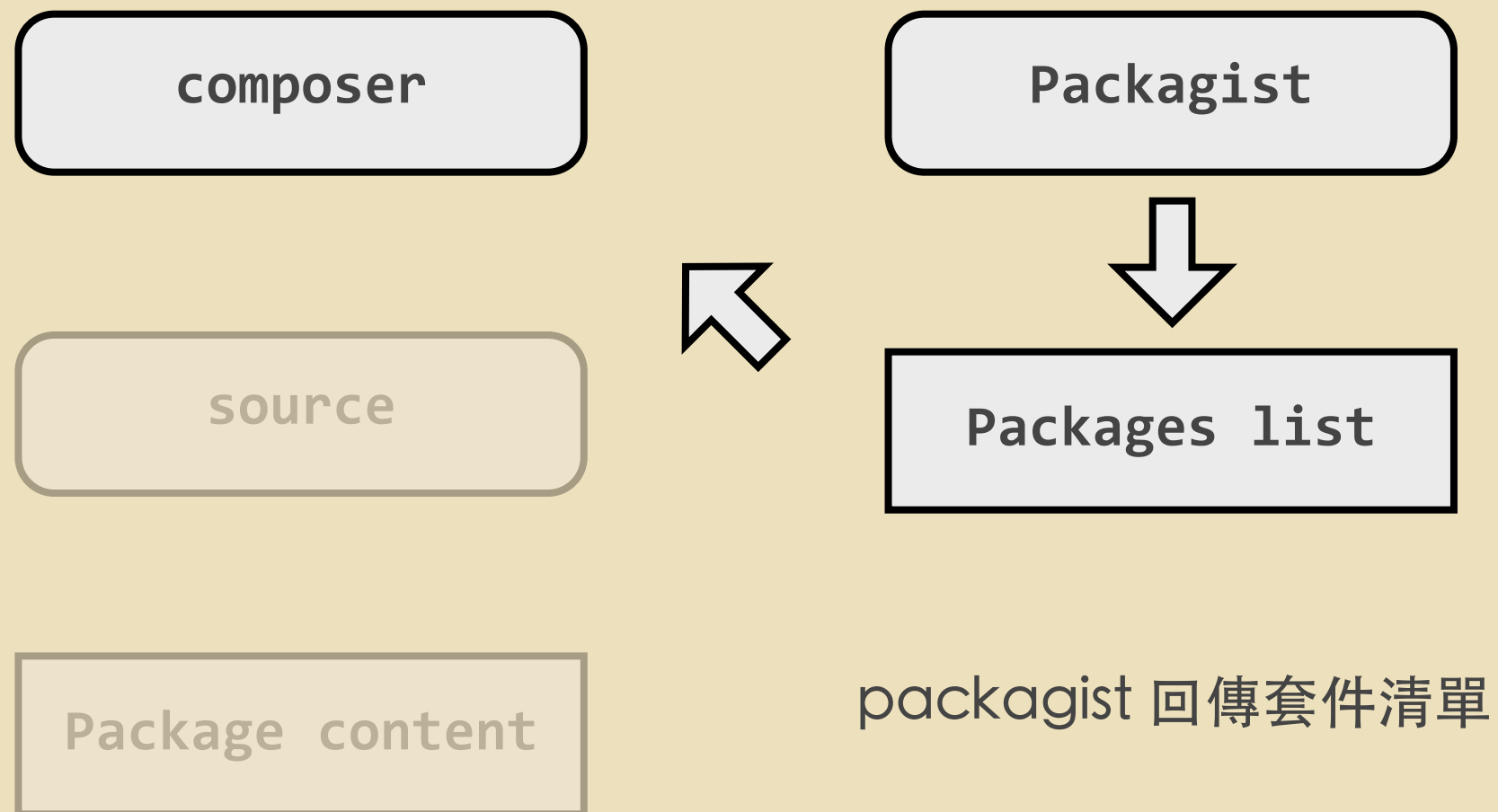
Package content

# 運作方式

composer 向 packagist 搜尋套件



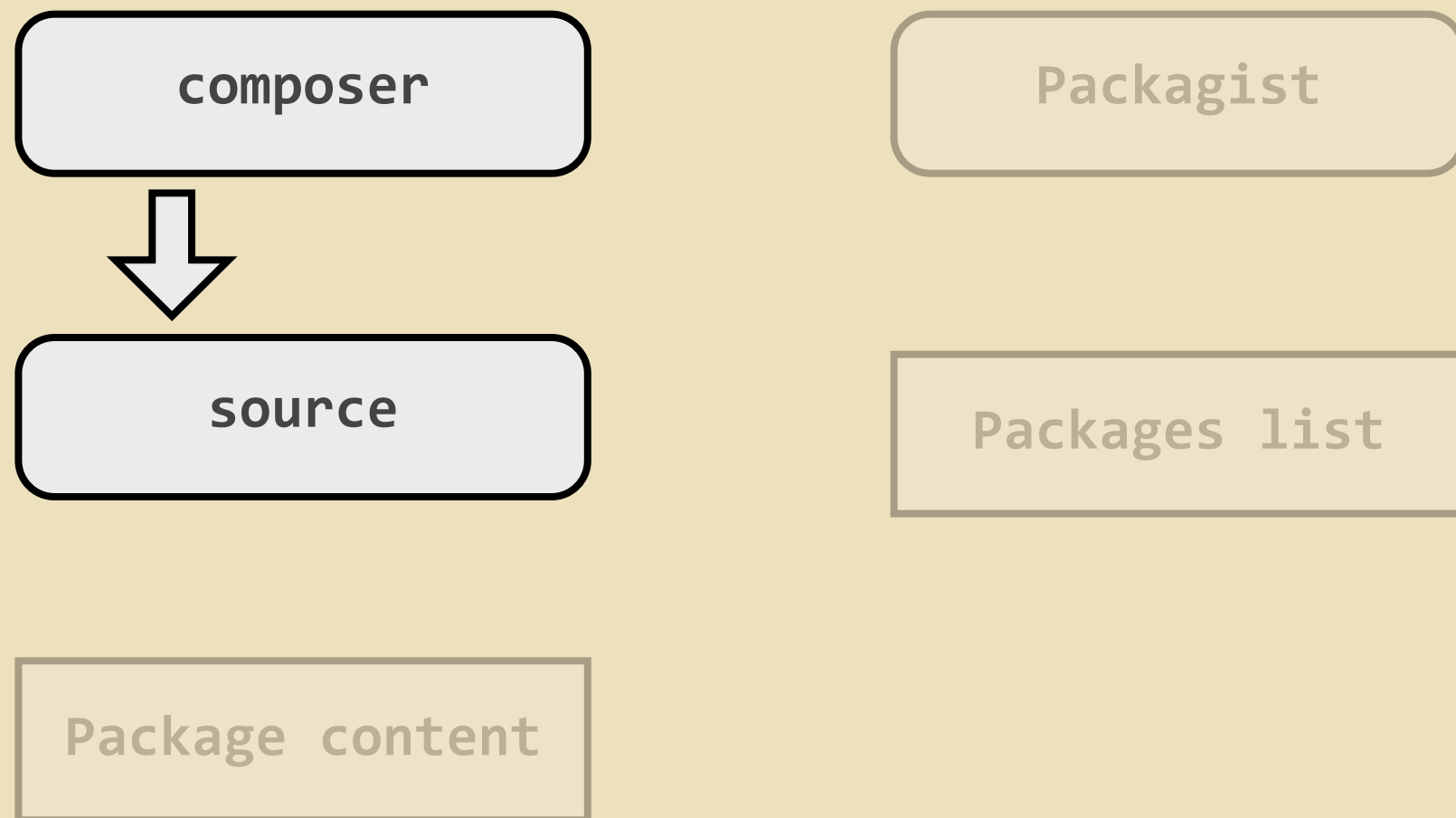
# 運作方式





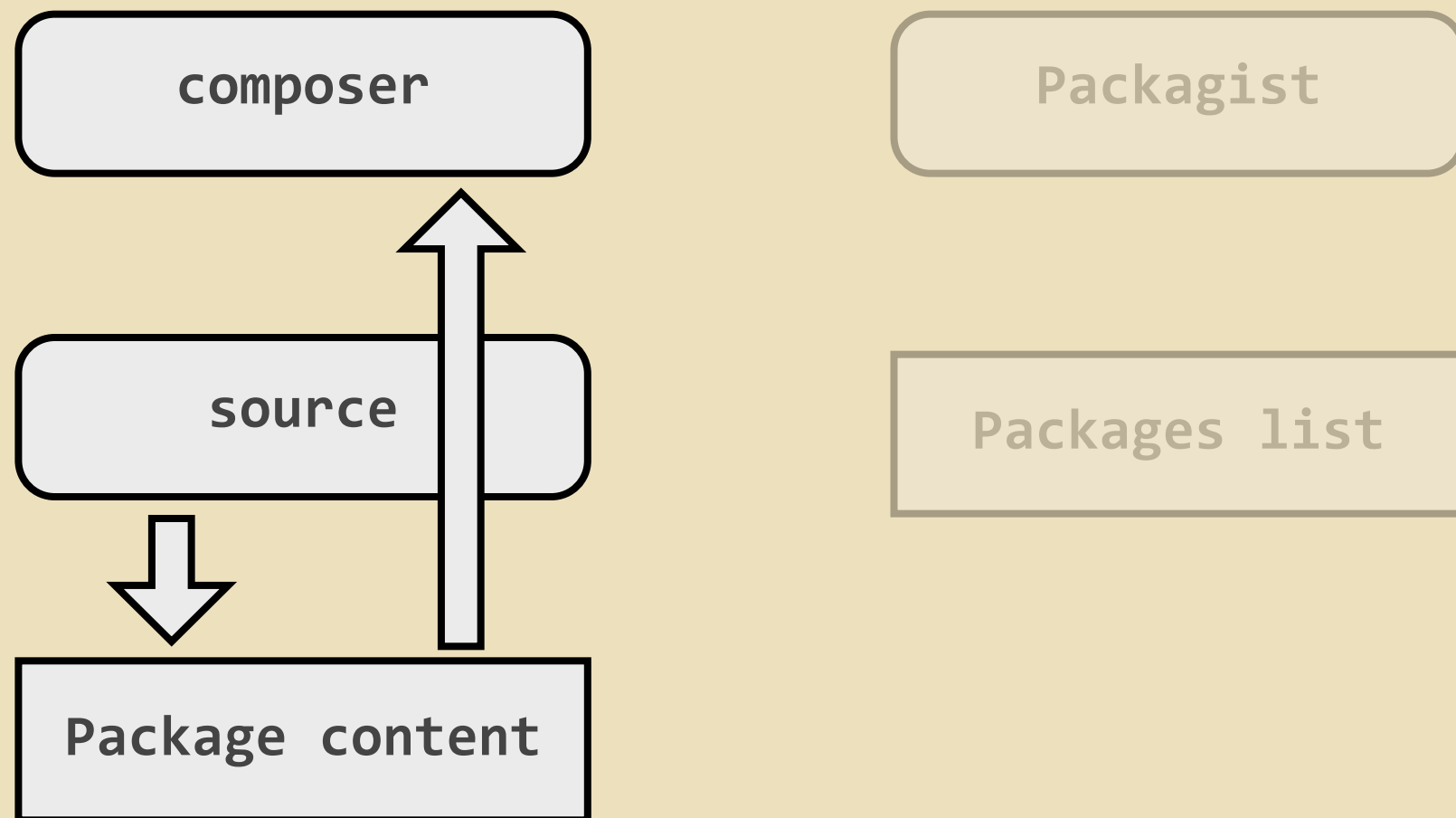
# 運作方式

composer 依照套件清單找到來源



# 運作方式

composer 從來源取回套件內容



增加相依套件

# \$ composer require

```
Search for a package []: phpunit
```

```
Found 15 packages matching phpunit
```

```
[0] phpunit/phpunit
```

```
[1] phpunit/phpunit-mock-objects
```

```
....
```

```
[14] ezzatron/phpunit-extensions
```

```
Enter package # to add, or the complete package name  
if it is not listed []: 0
```

```
Enter the version constraint to require []: >= 3.7.0
```

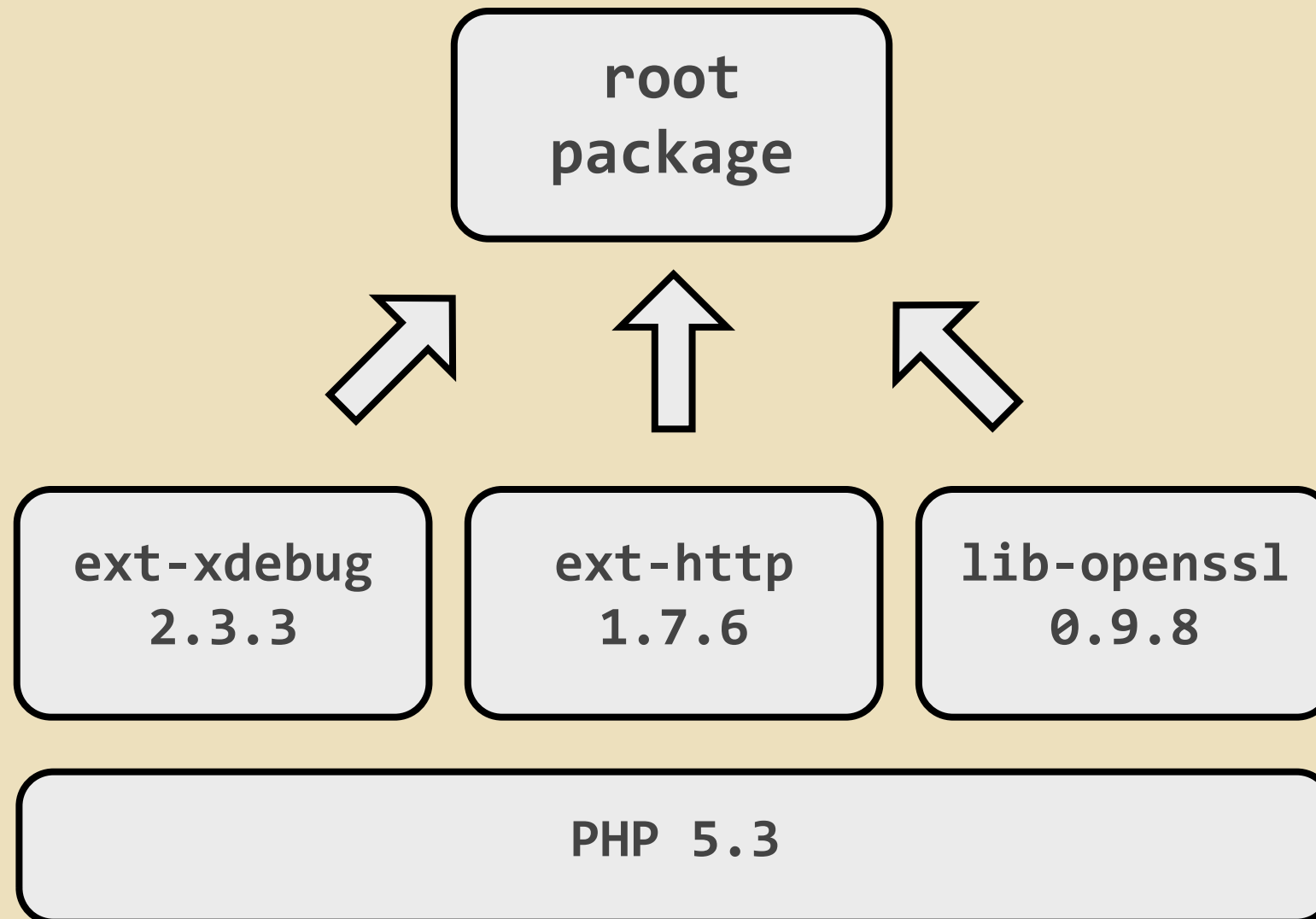
```
Search for a package []:
```

# 小提醒

- **--dev** : 安裝開發用的套件
- **--no-update** : 不直接安裝套件
- **composer show -i**  
可以看目前安裝的套件

# 平台環境相依性

# PHP 平台、Extension 與系統函式庫



# 類型

- **php** : PHP 平台
- **ext-<name>** : 以 PECL 安裝之套件
- **lib-<name>** : 系統函式庫
- **composer show --platform** :  
查看目前平台的套件



# composer.json 寫法

```
"require": {  
    "php": ">= 5.3",  
    "ext-xdebug": ">= 2.3.3",  
    "ext-http": ">= 1.7.6",  
    "lib-openssl": ">= 0.9.8"  
}
```

需要手動加入

# 套件來源

# 來源類型

**Composer**

在 **packagist.org** 管理的套件（預設）

**VCS**

以 **GIT / SVN / HG** 管理的套件

**PEAR**

以 **PEAR** 管理的套件

# VCS (GIT) 的 composer.json 寫法

```
{
  "require": {
    "vendor/private-repo": "dev-master"
  },
  "repositories": [
    {
      "type": "vcs",
      "url": "git@bitbucket.org:vendor/private-repo.git"
    }
  ]
}
```

針對私有的 git 專案採用的寫法

# 小提醒

- **repositories** 只作用在 **root-package** 上
- 所以就算在相依套件的 `composer.json` 有定義 **repositories**，也不會被 `composer` 取用
- 必須在 **root-package** 的 `composer.json` 中定義所有的套件來源

自動載入

# 為什麼 Composer 要提供自動載入？

- 統一自動載入的方法
- 每個 Package 的目錄結構不見得相同
- 讓 Package 自行提供載入方式

# 載入類型

**PSR-0**

符合 PSR-0 規範的類別檔案

**classmap**

對應類別名稱的檔案

**files**

非類別但需要一開始就自動載入的檔案



# composer.json 寫法

```
{
    "autoload": {
        "psr-0": {
            "Monolog\\": ["src/", "lib/"],
            "Zend_": "library/"
        },
        "classmap": [ "classes/", "Something.php" ],
        "files": ["src/MyLibrary/functions.php"]
    }
}
```

# loader 寫法

```
// Composer autoloading
include 'vendor/autoload.php';
/* @var $loader Composer\Autoload\ClassLoader */

var_dump(new Zend\Http\Client());
```

# 小提醒

- autoload 採相對於 `composer.json` 的路徑
- 每個套件只需負責自己的 autoload
- 在 root-package 執行：  
**composer dump-autoload**  
就會把所有相依套件的 autoload 屬性  
編譯到 vendor 目錄裡的快取檔案
- 載入 loader 之後就不用自行引用類別定義檔
- MVC Framework 只要在 `index.php`  
載入 loader 即可

PEAR-like Style 與  
Namespace 並存

# Namespace vs. PEAR-like

- **Namespace** 類別名稱格式：**Vendor\Package\Class**
- **PEAR-like** 類別名稱格式：**Vendor\_Package\_Class**
- **"Vendor\"**, **"Vendor\Package"**,  
**"Vendor\_"**, **"Vendor\_Package\_"** 統稱為 **Prefix**

# 以 ZF1 與 ZF2 為例

ZF1: `Zend_Http_Client`

ZF2: `Zend\Http\Client`

**Composer  
ClassLoader**

Composer 提供的 ClassLoader  
用了 SPL 的自動載入功能  
可以幫我們自動載入類別檔案

`/path/to/zf1/library/  
Zend/Http/Client.php`

`vendor/zendframework/  
zendframework/library/  
Zend/Http/Client.php`

# 以 ZF1 與 ZF2 為例

ZF1: Zend\_Http\_Client

ZF2: Zend\Http\Client

告訴 ClassLoader  
ZF1 的類別要去哪裡找  
新增 Prefix 的路徑對應

Composer  
ClassLoader

'Zend\_' => '/path/to/zf1/library'

/path/to/zf1/library/  
Zend/Http/Client.php

vendor/zendframework/  
zendframework/library/  
Zend/Http/Client.php

# 以 ZF1 與 ZF2 為例

ZF1: `Zend_Http_Client`

ZF2: `Zend\Http\Client`

Composer  
ClassLoader

`'Zend_' => '/path/to/zf1/library'`

`/path/to/zf1/library/  
Zend/Http/Client.php`

`vendor/zendframework/  
zendframework/library/  
Zend/Http/Client.php`



# 以 ZF1 與 ZF2 為例

ZF1: `Zend_Http_Client`

ZF2: `Zend\Http\Client`



Composer  
ClassLoader

`'Zend_' => '/path/to/zf1/library'`

`/path/to/zf1/library/  
Zend/Http/Client.php`

`vendor/zendframework/  
zendframework/library/  
Zend/Http/Client.php`

# 以 ZF1 與 ZF2 為例

ZF1: Zend\_Http\_Client

ZF2: Zend\Http\Client



PEAR-like  
使用 Prefix 定義

Composer  
ClassLoader

'Zend\_' => '/path/to/zf1/library'



**/path/to/zf1/library/**  
Zend/Http/Client.php

vendor/zendframework/  
zendframework/library/  
Zend/Http/Client.php

# 以 ZF1 與 ZF2 為例

ZF1: `Zend_Http_Client`

ZF2: `Zend\Http\Client`

**Composer  
ClassLoader**

`'Zend_' => '/path/to/zf1/library'`

`/path/to/zf1/library/  
Zend/Http/Client.php`

`vendor/zendframework/  
zendframework/library/  
Zend/Http/Client.php`

# 以 ZF1 與 ZF2 為例

ZF1: `Zend_Http_Client`

ZF2: `Zend\Http\Client`



**Composer  
ClassLoader**

`'Zend_' => '/path/to/zf1/library'`

`/path/to/zf1/library/  
Zend/Http/Client.php`

`vendor/zendframework/  
zendframework/library/  
Zend/Http/Client.php`

# 以 ZF1 與 ZF2 為例

ZF1: `Zend_Http_Client`

ZF2: `Zend\Http\Client`



Namespace  
則使用預設 vendor 路徑

**Composer  
ClassLoader**

`'Zend_' => '/path/to/zf1/library'`



`/path/to/zf1/library/  
/Zend/Http/Client.php`

**`vendor/zendframework/  
zendframework/library/  
Zend/Http/Client.php`**

# 並存的 loader 寫法

```
// 定義 ZF1 的 library 路徑
$zf1Path = getenv('ZF1_PATH');
if (!$zf1Path) {
    $zf1Path = dirname(__DIR__) . '/zf1/library';
}

// ZF1 的路徑需要加到 include_path 給 Zend/Loader.php 用
$includePath = implode(PATH_SEPARATOR, array(
    $zf1Path,
));
set_include_path($includePath);

// Composer autoloading
if (file_exists('vendor/autoload.php')) {
    $loader = include 'vendor/autoload.php';
    /* @var $loader Composer\Autoload\ClassLoader */
    $loader->add('Zend_', $zf1Path);
}

var_dump(new Zend_Http_Client());
var_dump(new Zend\Http\Client());
```

# 小提醒

- 預設會以 **vendor** 下定義的 **namespace** 來抓取類別定義檔案
- 先試 **Namespace** ，再試 **PEAR-like style**
- **Prefix** 有沒有底線不重要  
例如 '**Zend**' 或 '**Zend\_**' 是一樣的；加上底線  
主要是為了判斷它是不是 **PEAR-like**

# Framework Skeleton



# 為什麼需要 Skeleton ？

- **Skeleton** 是一個將目錄結構和檔案定義好的專案
- 部份 MVC Framework 直接提供下載包
- 部份 MVC Framework 用 generator 產生
- 較新的 MVC Framework 可透過 Composer 的 **create-project** 指令來下載

# Laravel Skeleton

```
$ composer create-project \  
    laravel/laravel \  
    --prefer-dist
```

Laravel 直接使用 packagist 上的套件定義

# ZF2 Skeleton

```
$ composer create-project \  
    --repository-url=\   
    "https://packages.zendframework.com" \  
    -s dev \  
    zendframework/skeleton-application \  
    path/to/install
```

ZF2 使用 Zend 官方自己的套件系統

# 重點說明

- **Skeleton** 通常是一個 **root-package**
- **composer create-project** 預設抓取 **packagist.org** 的套件
- 可以用 **satis** 架設私人的 **package-repository**

Q&A