VC++ 기말요약

VR게임앱학과 이 은 석

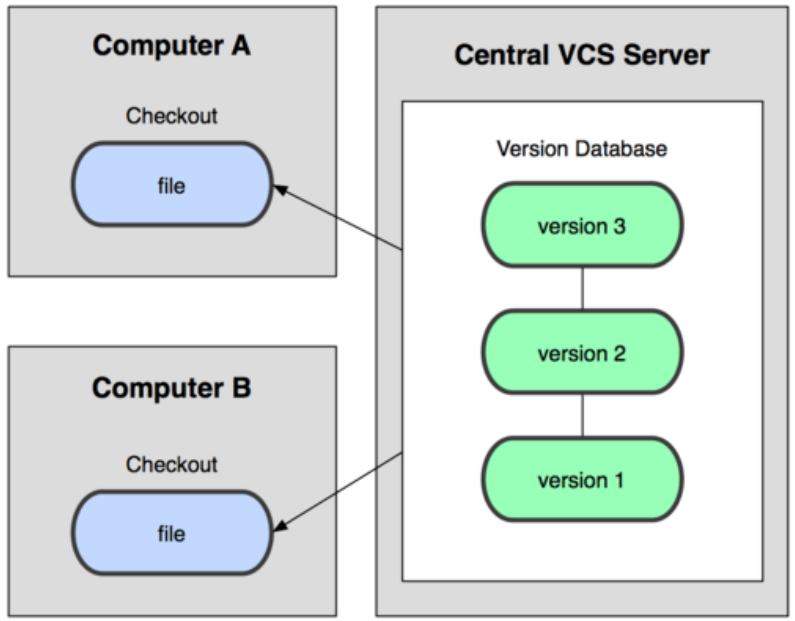
VCS 란 무엇인가?

Version **C**ontrol **S**ystem

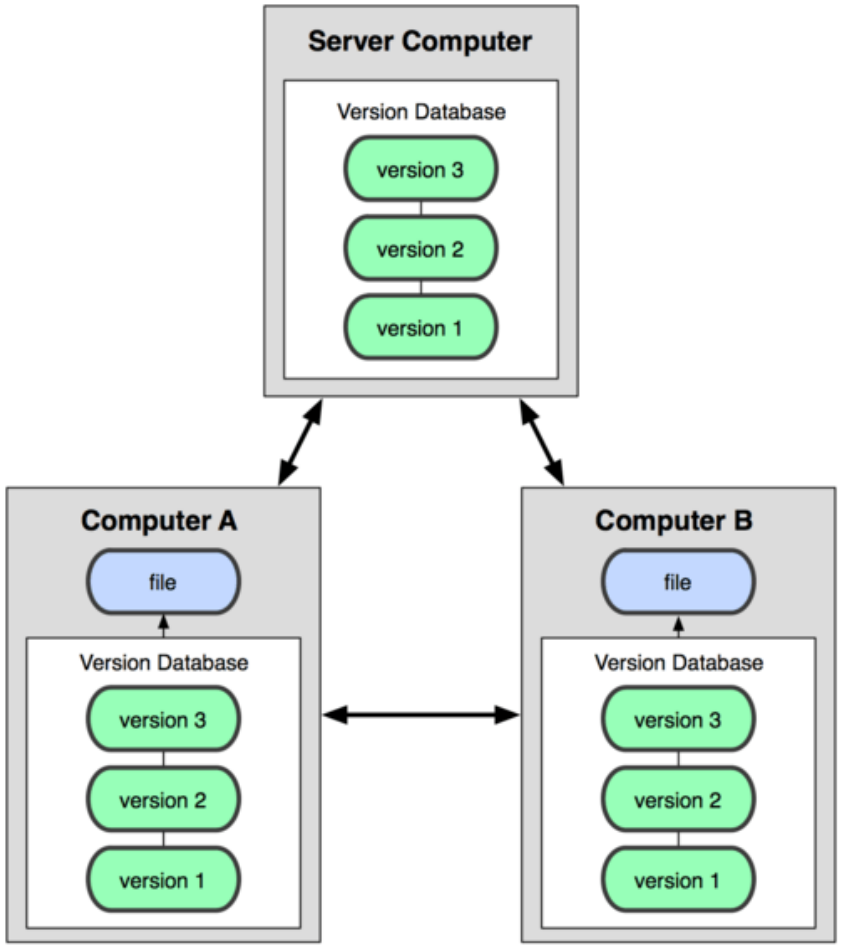
- 어떤 것이 가장 최신 파일인지(버저닝) 전문적으로 관리하는 시스템

VCS

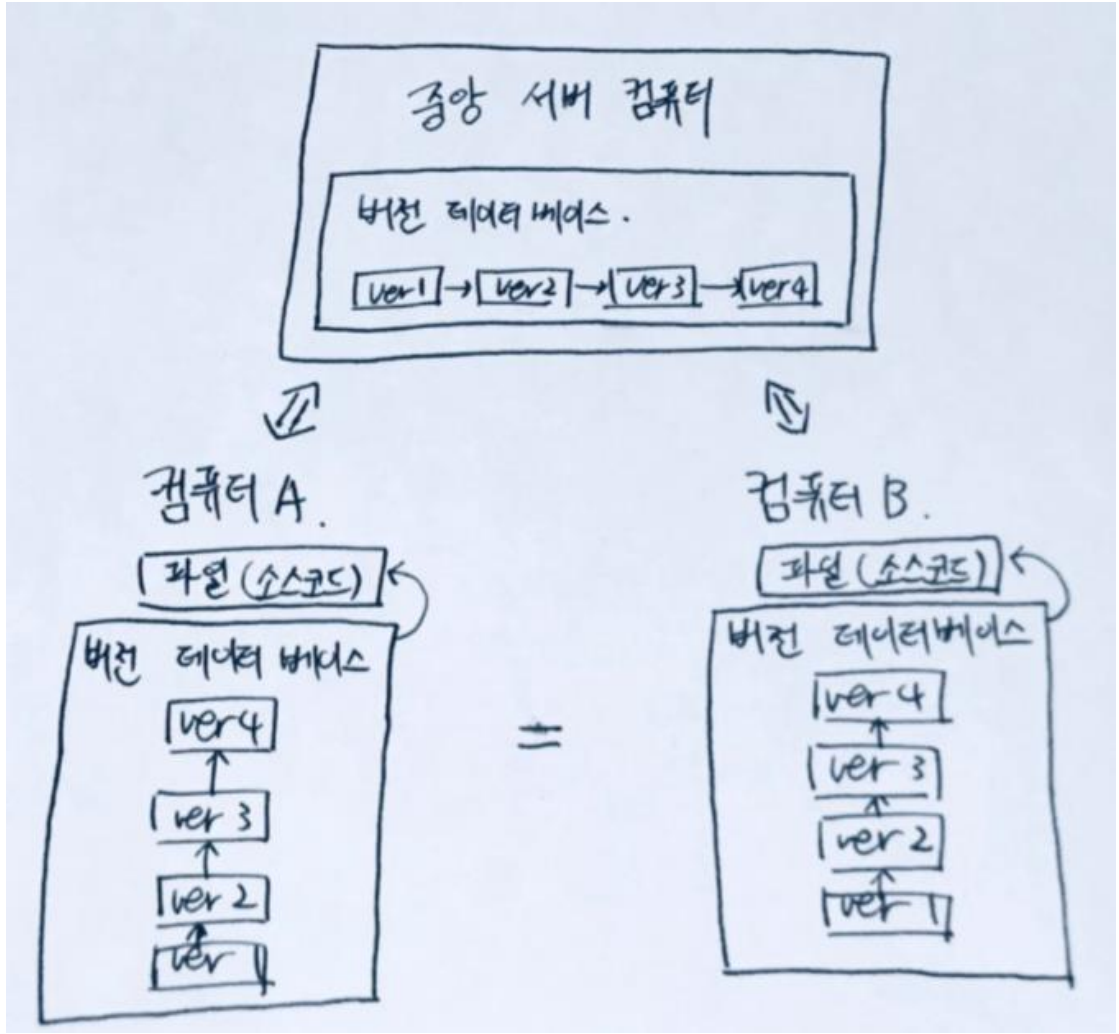
- Server-Client Model



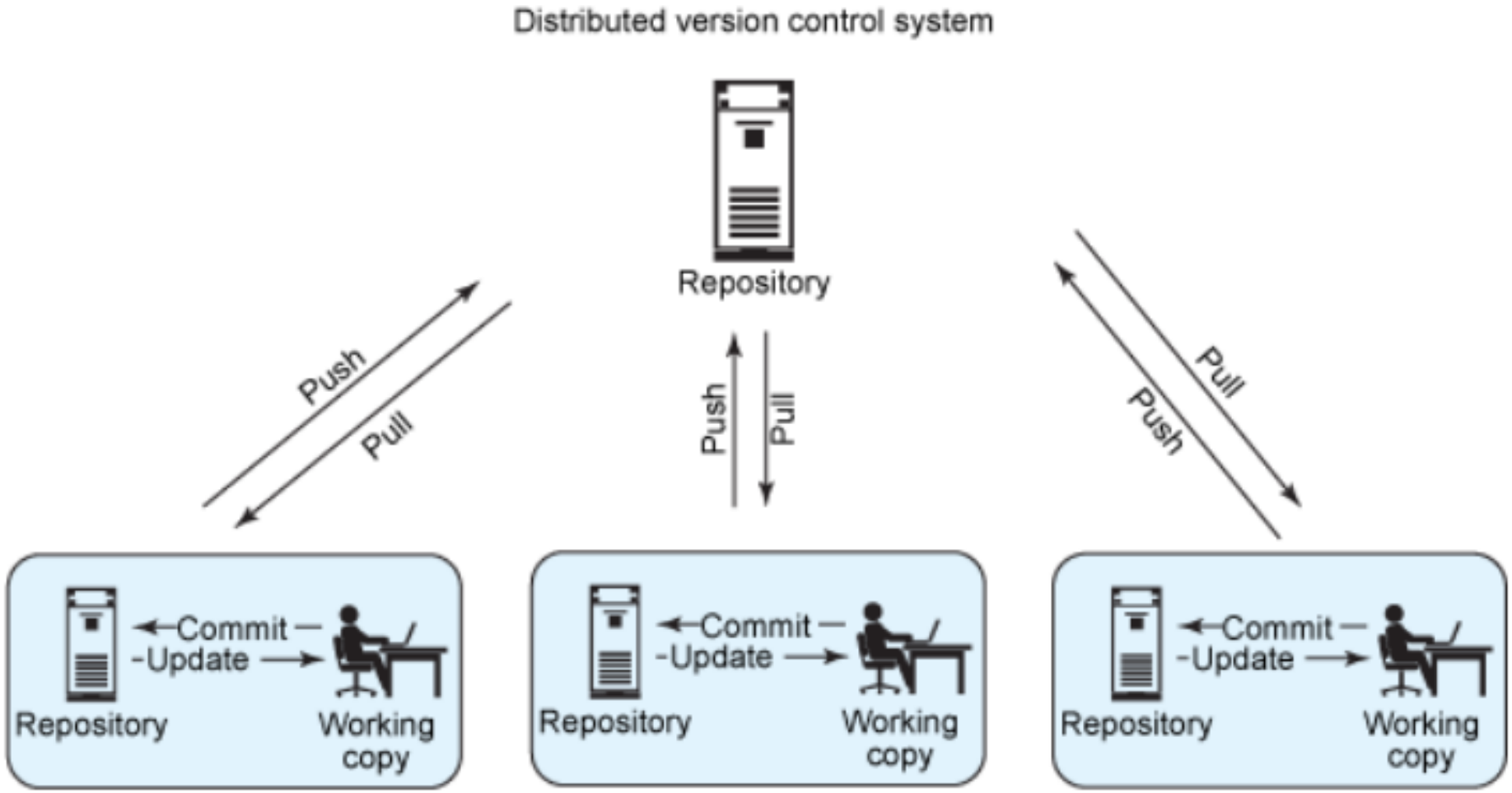
- Distributed Model



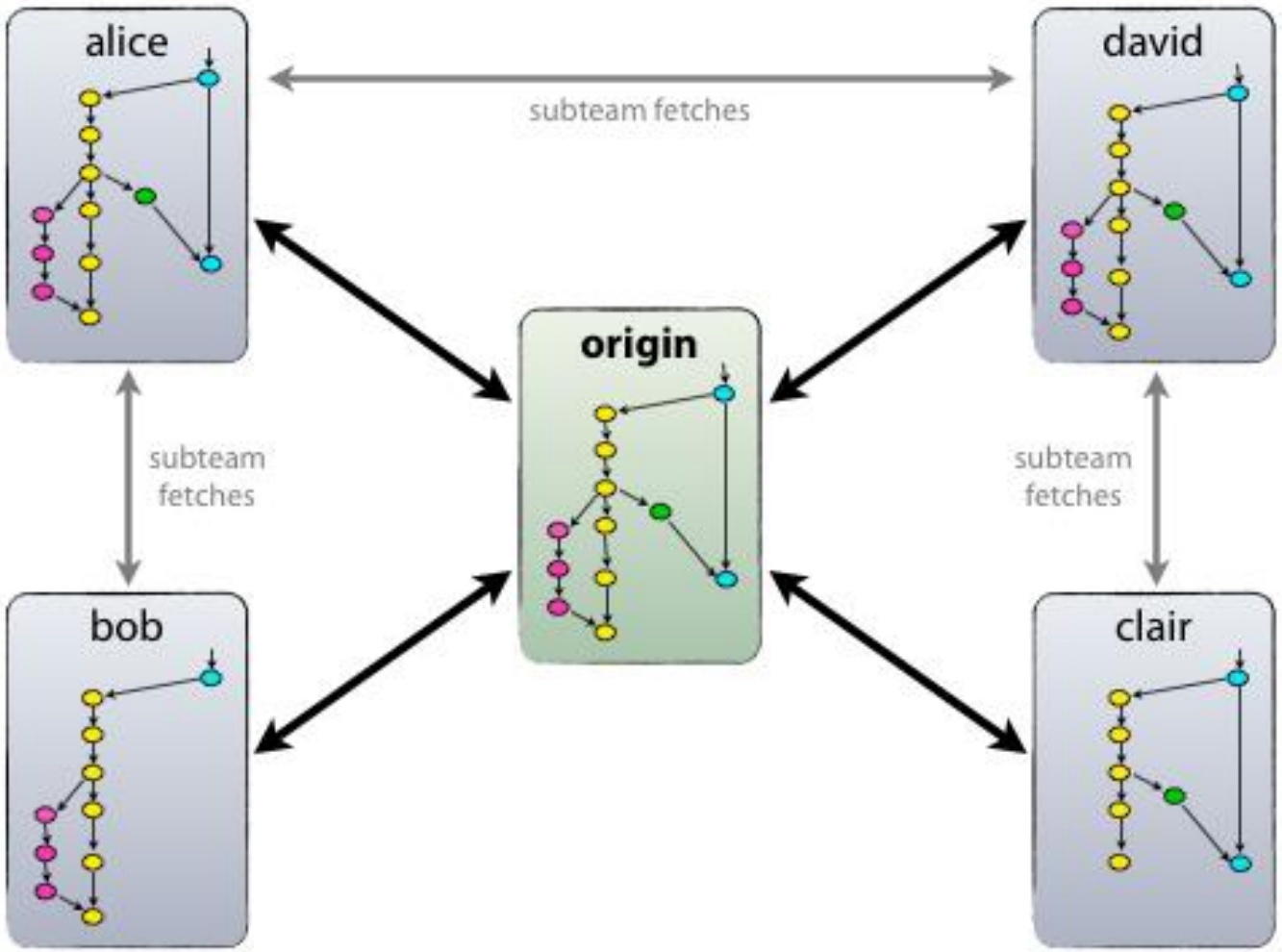
GIT의 구조

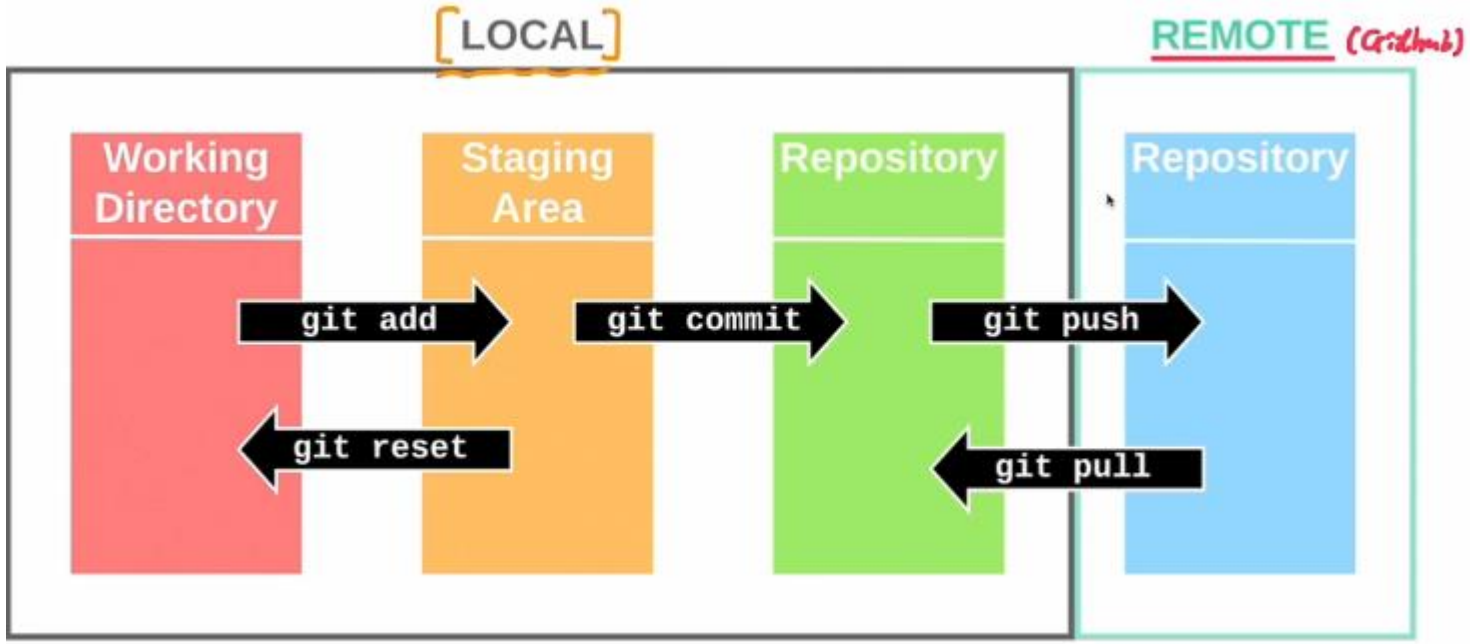

중앙 서버 컴퓨터
버전 데이터 베이스.
ver1
ver2
ver3
ver4
컴퓨터 A.
파일(소스코드)
버전 데이터 베이스
ver4
ver3
ver2
ver1
=
컴퓨터 B.
파일(소스코드)
버전 데이터베이스
ver4
ver3
ver2
ver1

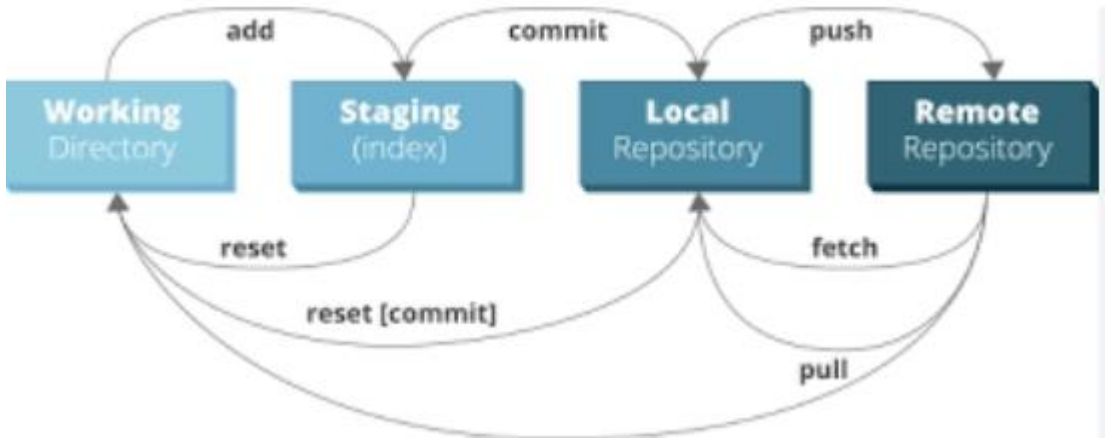
분산 모델


Distributed version control system
Repository
Push
Pull
Push
Pull
Pull
Push
Commit
Update
Repository
Working copy
Commit
Update
Repository
Working copy
Commit
Update
Repository
Working copy

분산 모델

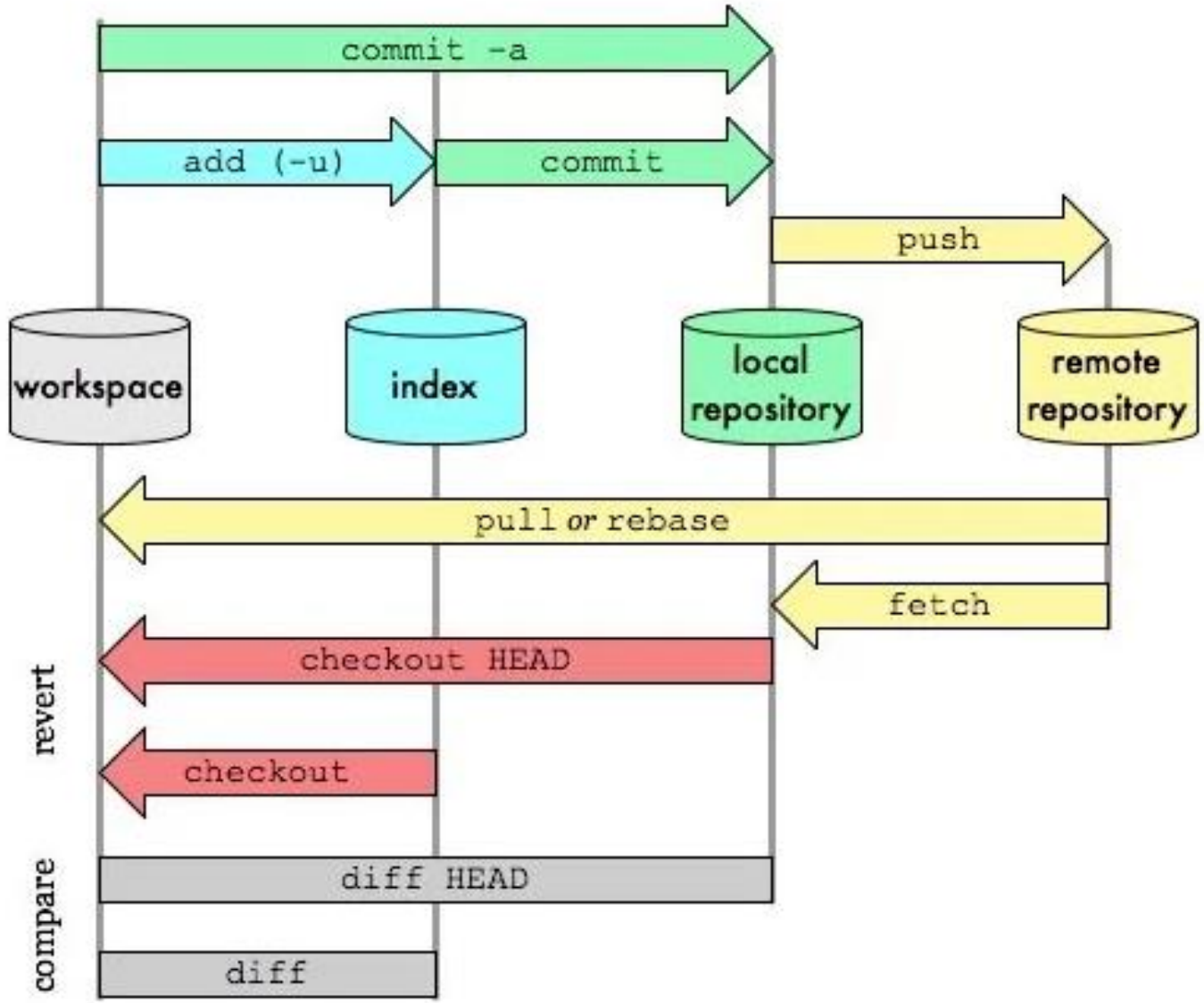

alice
david
subteam fetches
origin
subteam
fetches
subteam
fetches
bob
clair


LOCAL
REMOTE (GitHub)
Working Directory
Staging Area
Repository
Repository
git add
git commit
git push
git reset
git pull

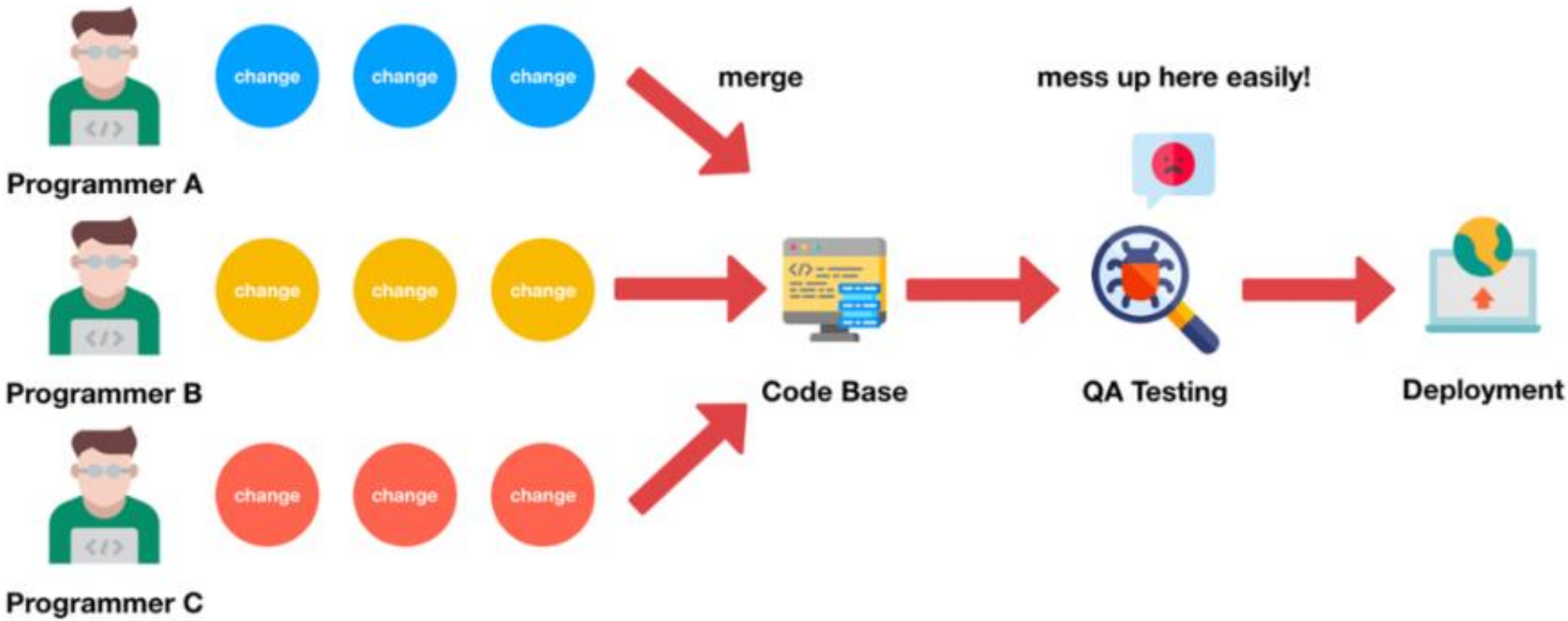

add
commit
push
Working Directory
Staging (index)
Local Repository
Remote Repository
reset
fetch
reset [commit]
pull

Git Data Transport Commands

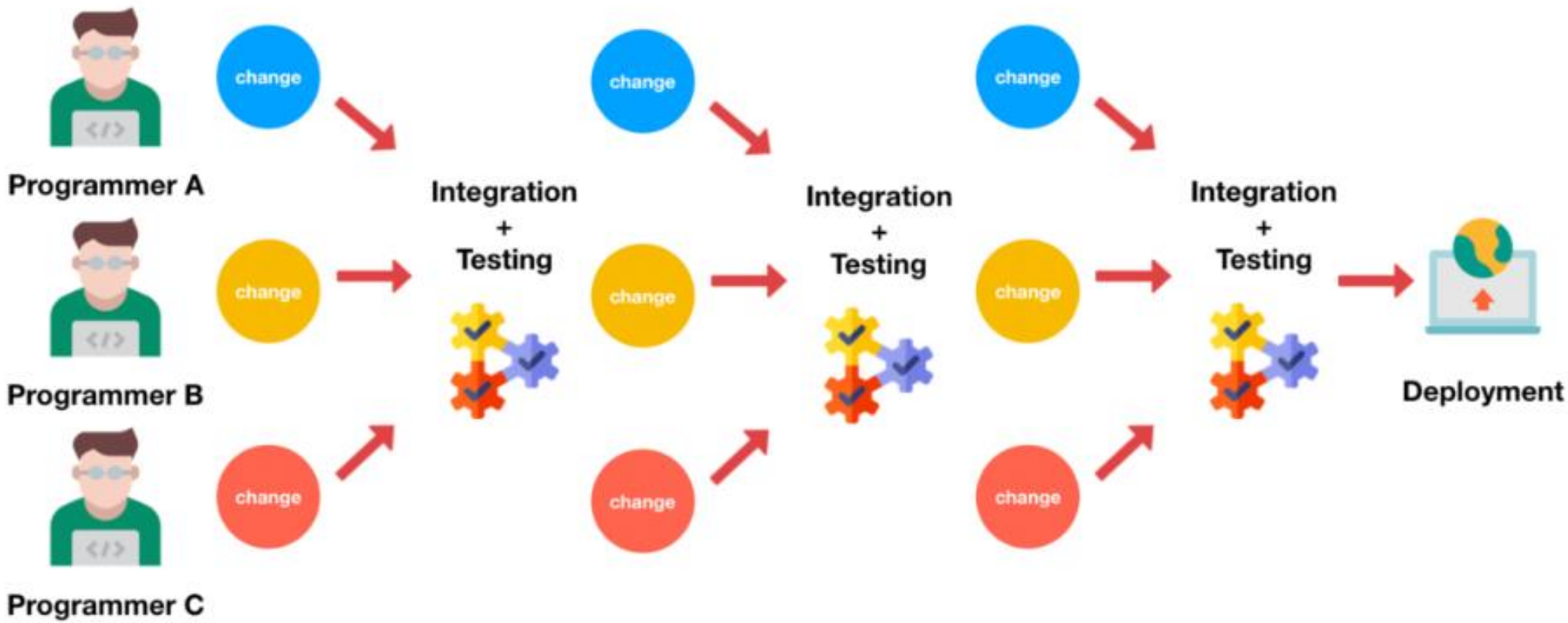
http://osteele.com



Traditional Way



With CI & CD


Programmer A
Programmer B
Programmer C
change
change
change
change
change
change
change
change
change
Integration
+
Testing
Integration
+
Testing
Integration
+
Testing
Deployment

3 Tier Architecture

프리젠테이션 계층

프레젠테이션 계층은 일반 사용자가 애플리케이션과 상호작용하는 애플리케이션의 사용자 인터페이스 및 커뮤니케이션 계층입니다. 주요 목적은 정보를 표시하고 사용자로부터 정보를 수집하는 것입니다. 이 최상위 레벨 계층은 예를 들어 웹 브라우저, 데스크탑 애플리케이션 또는 그래픽 사용자 인터페이스(GUI)에서 실행될 수 있습니다. 웹 프리젠테이션 계층은 일반적으로 HTML, CSS 및 JavaScript를 사용하여 개발됩니다. 데스크탑 애플리케이션은 플랫폼에 따라 다양한 언어로 작성될 수 있습니다.

애플리케이션 계층

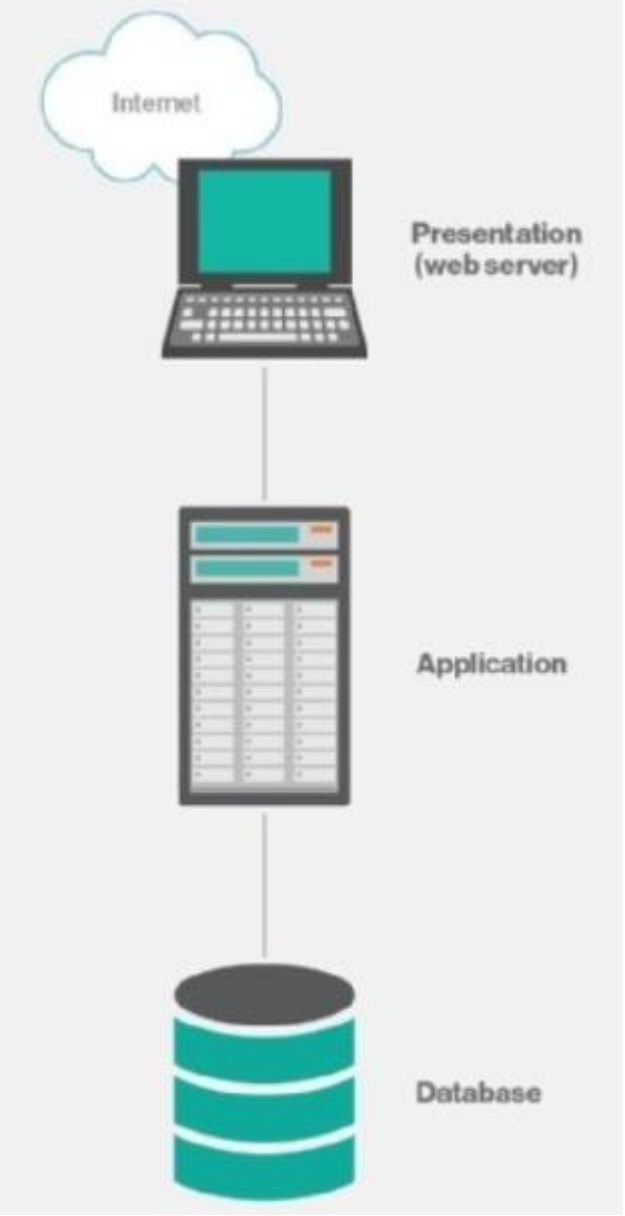
논리 계층 또는 중간 계층이라고도 하는 애플리케이션 계층은 애플리케이션의 핵심입니다. 이 계층에서는 특정 비즈니스 규칙 세트인 비즈니스 논리를 사용하여 프레젠테이션 계층에서 수집된 정보가 처리됩니다(경우에 따라 데이터 계층의 다른 정보와 관련하여 처리됨). 또한 애플리케이션 계층은 데이터 계층의 데이터를 추가, 삭제 또는 수정할 수도 있습니다.

애플리케이션 계층은 일반적으로 Python, Java, Perl, PHP 또는 Ruby를 사용하여 개발되며, API 호출을 사용하여 데이터 계층과 통신합니다.

데이터 계층

종종 데이터베이스 계층, 데이터 액세스 계층 또는 백엔드라고도 불리는 데이터 계층은 애플리케이션이 처리하는 정보가 저장 및 관리되는 곳입니다. 이는 관계형 데이터베이스 관리 시스템 (예: PostgreSQL, MySQL, MariaDB, Oracle, DB2, Informix 또는 Microsoft SQL Server) 또는 NoSQL 데이터베이스 서버(예: Cassandra, CouchDB 또는 MongoDB)일 수 있습니다.

https://www.ibm.com/kr-ko/topics/three-tier-architecture



3 Tier Architecture

3계층 아키텍처의 주요 장점은 기능의 논리적 및 물리적 분리입니다. 각 계층은 기능적 요구 사항에 가장 적합한 별도의 운영 체제와 서버 플랫폼(예: 웹 서버, 애플리케이션 서버, 데이터베이스 서버)에서 실행될 수 있습니다. 그리고 각 계층이 하나 이상의 전용 서버 하드웨어 또는 가상 서버에서 실행되므로, 다른 계층에 영향을 주지 않고도 각 계층의 서비스를 사용자 정의하고 최적화할 수 있습니다.

•**빠른 개발**: 각 계층이 서로 다른 팀에서 동시에 개발될 수 있으므로, 기업은 애플리케이션을 보다 빠르게 시장에 출시할 수 있으며 프로그래머는 각 계층에 최신 및 최상의 언어와 툴을 사용할 수 있습니다.

•**개선된 확장성**: 필요에 따라 어느 계층이든 다른 계층과 독립적으로 확장할 수 있습니다.

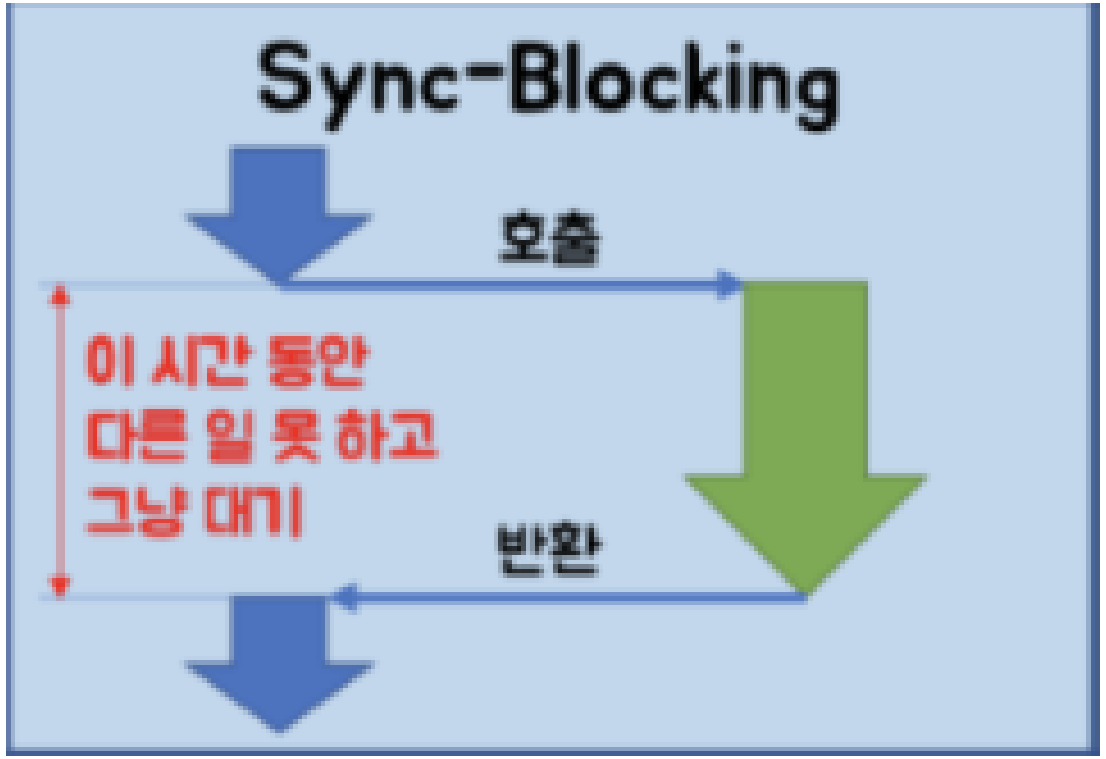
•**개선된 신뢰성**: 한 계층의 가동 중단은 다른 계층의 가용성 또는 성능에 별로 영향을 미치지 않습니다.

•**개선된 보안**: 프레젠테이션 계층과 데이터 계층이 직접 통신할 수 없으므로, 잘 설계된 애플리케이션 계층은 내부 방화벽의 일종으로 작동하여 SQL 인젝션 및 기타 악의적 공격을 방지할 수 있습니다.

https://www.ibm.com/kr-ko/topics/three-tier-architecture

Synchronous

이렇게 명령을 호출하고 반환할 때까지 기다리는게 Sync 이다.



Asyncronous

동기와 비동기

진동벨 : Callback

Win32

접두어	원래말	의미
cb	Count of Bytes	바이트 수
dw	double word	부호없는 long형 정수
h	handle	윈도우, 비트맵, 파일 등의 핸들
sz	Null Terminated	NULL 종료 문자열
w	Word	부호없는 정수형
i	Integer	정수형
b	Bool	논리형

데이터형	의미
BYTE	unsigned char형
WORD	unsigned short형
DWORD	unsigned long형
LONG	long과 동일하다.
LPSTR	char *와 동일하다.
BOOL	정수형이며 TRUE, FALSE 중 한 값을 가진다.

핸들(handle)이란 구체적인 어떤 대상에 붙여진 번호이며 문법적으로는 32비트의 정수값이다. 도스 프로그래밍에서는 거의 유일하게 파일 핸들만이 사용되었으며 그래서 도스에서 핸들은 곧 파일 핸들을 의미하는 경우가 많았다. 그러나 윈도우즈에서는 여러 가지 종류의 핸들이 사용되고 있다. 만들어진 윈도우에는 윈도우 핸들(hWnd)을 붙여 윈도우를 번호로 관리하며 아직은 잘 모르겠지만 DC에 대해서도 핸들을 사용하고 논리적 펜, 브러시에도 핸들을 붙여 관리한다. 심지어 메모리를 할당할 때도 할당한 메모리의 번지를 취급하기보다는 메모리에 번호를 붙인 메모리 핸들을 사용한다. 왜 이렇게 핸들을 자주 사용하는가 하면 대상끼리의 구분을 위해서는 문자열보다 정수를 사용하는 것이 훨씬 더 속도가 빠르기 때문이다.

윈도우즈에서 핸들을 이렇게 많이 사용하므로 우리는 핸들의 일반적인 특성에 관해서 미리 숙지하는 것이 좋다. 핸들은 일반적으로 다음과 같은 특징이 있다.

① 일단 핸들은 정수값이며 대부분의 경우 32비트값이다. 핸들을 사용하는 목적은 오로지 구분을 위한 것이므로 핸들끼리 중복되지 않아야하며 이런 목적으로는 정수형이 가장 적합하다.

② 핸들은 운영체제가 발급해 주며 사용자는 쓰기만 하면 된다. 예를 들어 윈도우를 만들거나 파일을 열면 운영체제는 만들어진 윈도우나 열려진 파일에 핸들을 붙여준다. 사용자는 이 핸들을 잘 보관해 두었다가 해당 윈도우나 파일을 다시 참조할 때 핸들을 사용하면 된다. 사용자가 직접 핸들을 만들 경우란 없다.

③ 같은 종류의 핸들끼리는 절대로 중복된 값을 가지지 않는다. 만약 이렇게 된다면 핸들은 구분을 위해 사용할 수 없을 것이다. 물론 다른 종류의 핸들끼리는 중복된 값을 가질 수도 있다.

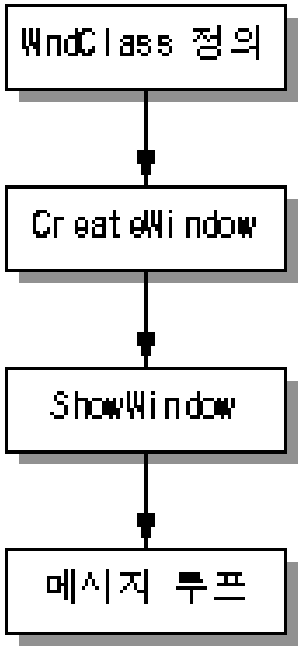
④ 핸들은 정수형이므로 값을 가지겠지만 그 실제값이 무엇인지는 몰라도 상관없다. 핸들은 크고 작음의 성질을 가지는 숫자가 아니라 단순한 표식일 뿐이다. 핸들형 변수를 만들어 핸들을 대입받아 쓰고 난 후에는 버리면 된다.

윈도우즈에서 핸들은 예외없이 접두어 h로 시작되며 핸들값을 저장하기 위해 별도의 데이터형까지 정의해 두고 있다. HWND, HPEN, HBRUSH, HDC 등이 핸들을 담기 위한 데이터형들이며 모두 부호없는 정수형이다.

Win32

인수	의미
hInstance	프로그램의 인스턴스 핸들
hPrevInstance	바로 앞에 실행된 현재 프로그램의 인스턴스 핸들, 없을 경우는 NULL이 되며 WIN32에서는 항상 NULL이다. 호환성을 위해서만 존재하는 인수이므로 신경쓰지 않아도 된다.
lpCmdLine	명령행으로 입력된 프로그램 인수이다. 도스의 argv인수에 해당한다.
nCmdShow	프로그램이 실행될 형태이며 최소화, 보통모양 등이 전달된다.

멤버	의미
hwnd	메시지를 받을 윈도우 핸들이다.
message	어떤 종류의 메시지인가를 나타낸다. 가장 중요한 값이다.
wParam	전달된 메시지에 대한 부가적인 정보를 가진다. 어떤 의미를 가지는가는 메시지별로 다르다. 32비트값이다.
lParam	전달된 메시지에 대한 부가적인 정보를 가진다. 어떤 의미를 가지는가는 메시지별로 다르다. 32비트값이다.
time	메시지가 발생한 시간이다.
pt	메시지가 발생했을 때의 마우스 위치이다.



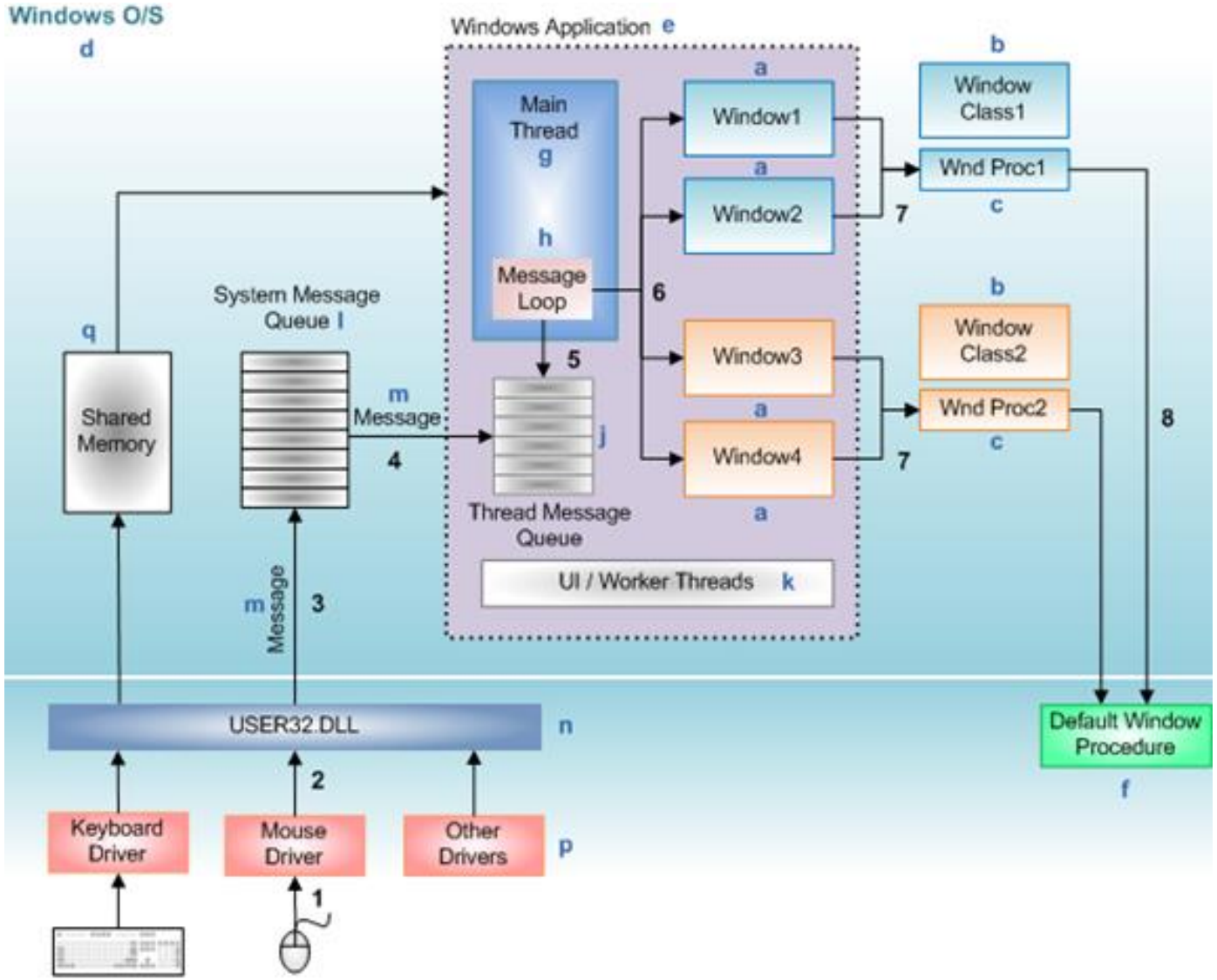
윈도우의 기반이 되는 클래스를 정의한다.

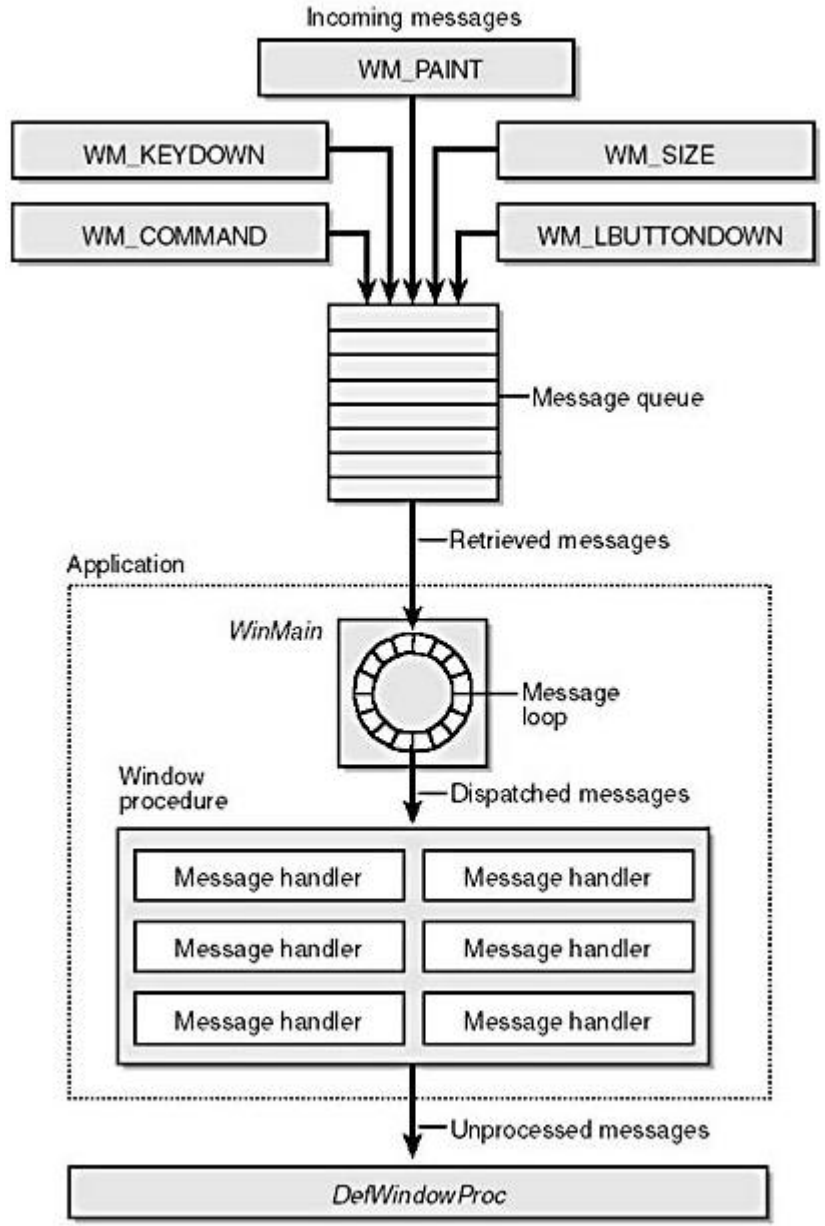
메모리상에 윈도우를 만든다.

윈도우를 화면에 표시한다.

사용자로부터의 메시지를 처리한다.

Win32


Windows O/S
d
Windows Application e
Main Thread
g
h
Message Loop
Window1
Window2
Window3
Window4
a
Window Class1
Window Class2
b
Wnd Proc1
Wnd Proc2
c
6
7
8
5
System Message Queue I
q
Shared Memory
m Message
4
3
j
Thread Message Queue
UI / Worker Threads k
USER32.DLL
n
2
1
Keyboard Driver
Mouse Driver
Other Drivers
p
Default Window Procedure
f


Incoming messages
WM_PAINT
WM_KEYDOWN
WM_SIZE
WM_COMMAND
WM_LBUTTONDOWN
Message queue
Retrieved messages
Application
WinMain
Message loop
Window procedure
Dispatched messages
Message handler
Message handler
Message handler
Message handler
Message handler
Message handler
Unprocessed messages
DefWindowProc

Win32

1.그래픽 기반의 운영체제이다.
화면 처리를 문자 단위로 하는 것이 아니라 디지탈 표현의 최소단위인 픽셀(Pixel)로 처리함으로써 섬세한 화면 처리가 가능하다. 그래서 사용자는 무엇보다 친숙하고 예쁜 화면을 대하게 되었으며 훨씬 더 쉽게 프로그램을 사용할 수 있게 되었다. 뿐만 아니라 프로그래머에게도 무한한 표현의 자유를 준다.
2.멀티 태스킹이 가능하다.
한번에 여러가지 일을 수행할 수 있다. 그래서 한대의 컴퓨터로 여러대의 컴퓨터가 해야할 일을 수행할 수 있으며 하던 일을 잠시 접어두고 다른 프로그램을 언제든지 실행할 수 있다. 뿐만 아니라 다운로드를 받거나 랜더링을 하거나 컴파일을 하는 지루한 시간동안 가만히 앉아 기다릴 필요도 없어졌다.
3.장치에 영향을 받지 않는다.
윈도우즈는 디바이스 드라이버(Device Driver)에 의해 주변 장치들을 제어하고 관리한다. 따라서 장치가 바뀌면 디바이스 드라이버를 같이 교체해주면 될 뿐 소프트웨어는 이에 영향을 받지 않게 되었다. 그래서 프로그래머들은 사용자들의 시스템이 어떠한가에는 신경쓸 필요없이 자기가 만들고자 하는 프로그램의 개발에만 열중하면 된다. 과거에 프로그래머가 했었던 많은 잔일들을 운영체제가 대신해주므로 시간적, 금전적으로 엄청난 비용을 절감하게 되었다.
4.일관성
인터페이스 구성이 표준화됨에 따라 한번 배우기만 하면 어떤 프로그램이나 유사한 방법으로 사용할 수 있다. 이런 일관성은 사용자에게는 빨리 배워 빨리 쓸 수 있도록 해 주며 개발자에게는 화면 디자인을 훨씬 더 쉽고 빠르게 할 수 있도록 해 준다.
물론 이 외에도 윈도우즈는 아주 많은 장점들을 가지고 있다. 그러나 기능적인 이런 장점들외에 가장 중요한 장점은 현재까지 발표된 어떤 운영체제보다도 많은 사용자를 거느리고 있다는 대중성을 들 수 있다. 많은 사람들이 사용한다는 것은 그만큼 시장이 넓다는 뜻이며 따라서 무수히 많은 소프트웨어를 필요로 하고 고급 프로그래머에 대한 수요가 많다. 즉 경제성이 높다는 뜻이며 좀 속되게 표현하자면 돈이 되는 분야라는 점이다.
운영 체제의 기능만을 놓고 따진다면 사실 윈도우즈보다 훨씬 더 뛰어난 운영 체제도 많

Windows Message

WM_CHAR	A character is input from the keyboard.
WM_COMMAND	The user selects an item from a menu, or a control sends a notification to its parent.
WM_CREATE	A window is created.
WM_DESTROY	A window is destroyed.
WM_LBUTTONDOWN	The left mouse button is pressed.
WM_LBUTTONUP	The left mouse button is released.
WM_MOUSEMOVE	The mouse pointer is moved.
WM_PAINT	A window needs repainting.
WM_QUIT	The application is about to terminate.
WM_SIZE	A window is resized.

Windows Message

윈도우즈를 메시지 구동 시스템(Message Driven System)이라고 하며 이 점이 도스와 가장 뚜렷한 대비를 이루는 윈도우즈의 특징이다. 도스에서는 프로그래머에 의해 미리 입력된 일련의 명령들을 순서대로 실행하는 순차적 실행방법을 사용한다. 윈도우즈는 이와 다르게 프로그램의 실행 순서가 명확하게 정해져 있지 않으며 상황에 따라 실행 순서가 달라지는데 여기서 말하는 상황이란 바로 어떤 메시지가 주어졌는가를 말한다.
메시지란 사용자나 시스템 내부적인 동작에 의해 발생된 일체의 변화에 대한 정보를 말한다. 예를 들어 사용자가 마우스의 버튼을 눌렀다거나 키보드를 눌렀다거나 윈도우가 최소화 되었다거나 하는 변화에 대한 정보들이 메시지이다. 메시지가 발생하면 프로그램에서는 메시지가 어떤 정보를 담고 있는가를 분석하여 어떤 루틴을 호출할 것인가를 결정한다. 즉 순서를 따르지 않고 주어진 메시지에 대한 반응을 정의하는 방식으로 프로그램이 실행된다.
윈도우즈 프로그램에서 메시지를 처리하는 부분을 메시지 루프라고 하며 보통 WinMain 함수의 끝에 다음과 같은 형식으로 존재한다.
while(GetMessage(&Message,0,0,0)) { TranslateMessage(&Message); DispatchMessage(&Message); } 메시지 루프는 세 개의 함수 호출로 이루어져 있으며 전체 루프는 while문으로 싸여져 있다. 각 함수가 어떤 동작을 하는지 대충 알아보자.
BOOL GetMessage(LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax);
이 함수는 시스템이 유지하는 메시지 큐에서 메시지를 읽어들인다. 읽어들인 메시지는 첫번째 인수가 지정하는 MSG 구조체에 저장된다. 이 함수는 읽어들인 메시지가 프로그램을 종료하라는 WM_QUIT일 경우 False를 리턴하며 그 외의 메시지이면 True를 리턴한다. 따라서 WM_QUIT 메시지가 읽혀질 때까지, 즉 프로그램이 종료될 때까지 전체 while 루프가 계속 실행된다. 나머지 세 개의 인수는 읽어들일 메시지의 범위를 지정하는데 잘 사용되지 않으므로 일단 무시하기로 한다.
BOOL TranslateMessage(CONST MSG *lpMsg);
키보드 입력 메시지를 가공하여 프로그램에서 쉽게 쓸 수 있도록 해 준다. 윈도우즈는 키보드의 어떤 키가 눌러졌다거나 떨어졌을 때 키보드 메시지를 발생시키는데 이 함수는 키보드의 눌림(WM_KEYDOWN)과 떨어짐(WM_KEYUP)이 연속적으로 발생할 때 문자가 입력되었다는 메시지(WM_CHAR)를 만드는 역할을 한다. 예를 들어 A키를 누른 후 다시 A키를 떼면 A문자가 입력되었다는 메시지를 만들어 낸다.
LONG DispatchMessage(CONST MSG *lpmsg);
시스템 메시지 큐에서 꺼낸 메시지를 프로그램의 메시지 처리 함수(WndProc)로 전달한다. 이 함수에 의해 메시지가 프로그램으로 전달되며 프로그램에서는 전달된 메시지를 점검하여 다음 동작을 결정하게 된다.
메시지 루프에서 하는 일은 메시지를 꺼내고, 필요한 경우 약간 형태를 바꾼 후 응용 프로그램으로 전달하는 것 뿐이다. 이 과정은 WM_QUIT 메시지가 전달될 때까지, 즉 프로그램이 종료될때까지 반복된다. 결국 메시지 루프가 하는 일이란 메시지 큐에서 메시지를 꺼내 메시지 처리 함수로 보내주는 것 뿐이다.
실제 메시지 처리는 별도의 메시지 처리 함수(WndProc)에서 수행한다. 메시지는 시스템의 변화에 대한 정보이며 MSG라는 구조체에 보관된다. MSG 구조체는 다음과 같이 정의되어 있다.
typedef struct tagMSG { HWND hwnd; UINT message; WPARAM wParam; LPARAM lParam; DWORD time; POINT pt; } MSG;

message 멤버를 읽음으로써 메시지의 종류를 파악하며 message값에 따라 프로그램의 반응이 달라진다. wParam, lParam은 메시지에 대한 부가적인 정보를 가지되 메시지별로 의미가 다르다. 마치 인터럽터 루틴에서 각 레지스터의 의미가 인터럽터별로 다른 것과 마찬가지이다. GetMessage 함수는 읽은 메시지를 MSG형의 구조체에 대입해 주며 이 구조체는 DispatchMessage 함수에 의해 응용 프로그램의 메시지 처리 함수(WndProc)로 전달된다.
메시지는 실제로 하나의 정수값으로 표현되는데 메시지의 종류가 무척 많아 메시지의 번호를 일일이 암기하여 사용할 수가 없으므로 windows.h에 메시지별로 매크로 상수를 정의해 두었으며 접두어 WM_으로 시작된다. 가장 자주 사용되는 메시지의 종류를 몇 개만 보인다.
메시지의미WM_QUIT프로그램을 끝낼 때 발생하는 메시지이다.WM_LBUTTONDOWN마우스의 좌측 버튼을 누를 경우 발생한다.WM_CHAR키보드로부터 문자가 입력될 때 발생한다 WM_PAINT화면을 다시 그려야 할 필요가 있을 때 발생한다.WM_DESTROY윈도우가 메모리에서 파괴될 때 발생한다.WM_CREATE윈도우가 처음 만들어질 때 발생한다.
메시지 루프가 종료되면 프로그램은 마지막으로 Message.wParam을 리턴하고 종료한다. 이 값은 WM_QUIT 메시지로부터 전달된 탈출 코드(exit code)이다. 도스에서 사용하는 탈출 코드와 동일한 의미를 가지며 사용되는 경우가 거의 없다.