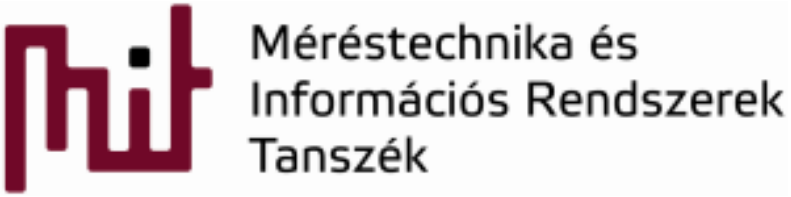
Konfigurációmenedzsment

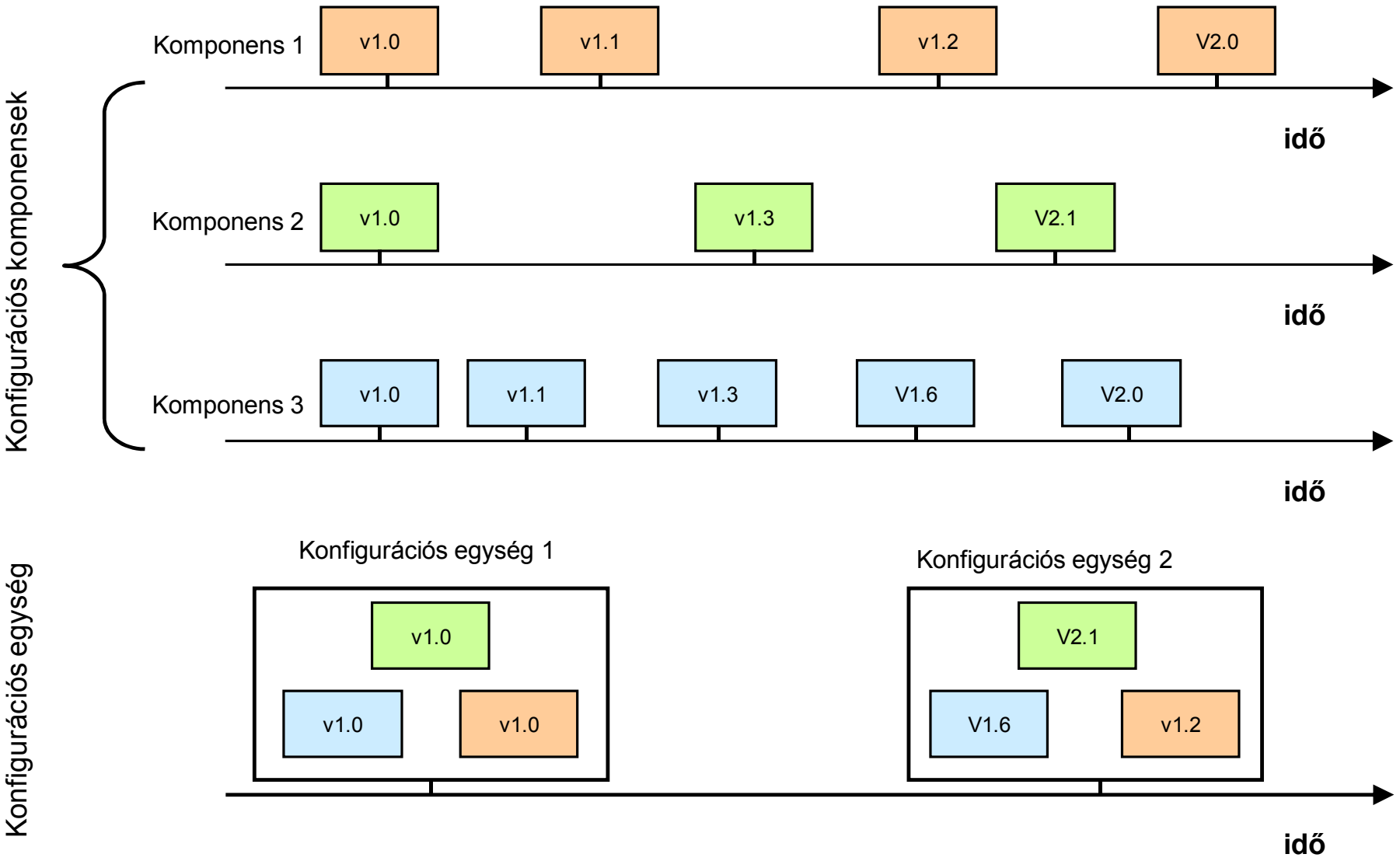
VIMIMA11 Rendszertervezés és –integráció

Scherer Balázs



© BME-MIT 2015

Az alapprobléma



© BME-MIT 2015

Konfigurációmenedzsment
CMMI folyamat

- **SG 1: A termékhez és elkészítéséhez szüksége alapkonfiguráció létrehozása (Establish Baselines)**
  - **SP 1.1:** A konfigurációs komponensek azonosítása (Identify Configuration Items)
  - **SP 1.2:** A konfiguráció menedzsment rendszer létrehozása (Establish a Configuration Management System)
  - **SP 1.3** Az alapkonfigurációk létrehozása, kibocsátása. Release kibocsájtása (Create or Release Baselines)
- **SG 2: Változások követése és kezelése (Track and Control Changes)**
  - **SP 2.1:** Változtatási kérések követése (Track Change Requests)
  - **SP 2.2:** A konfigurációs komponensek változásainak felügyelése (Control Configuration Items)
- **SG 3: Integritás létrehozása és megtartása (Establish Integrity)**
  - **SP 3.1: A konfigurációmenedzsmenthez tartozó loggok készítése** (Establish Configuration Management Records)
  - **SP 3.2: Konfigurációs auditok megrendezése** (Perform Configuration Audits)

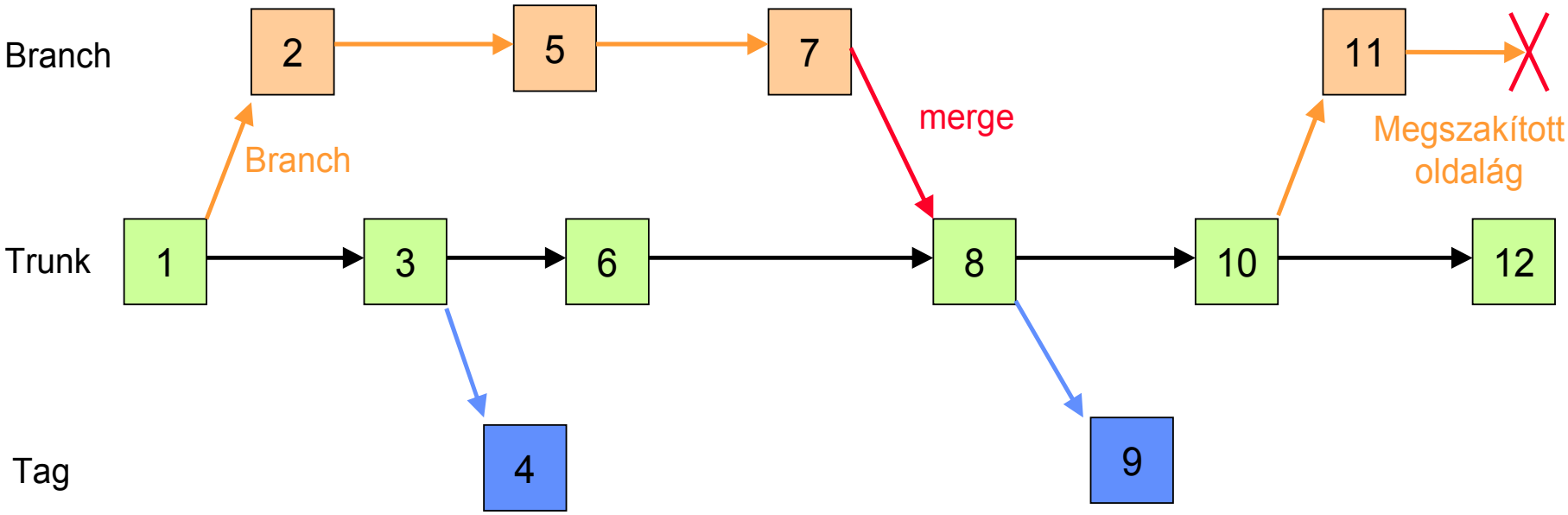
Konfigurációs komponensek azonosítása

- A végtermék, és azzal együtt kiszállításra kerülő komponensek
- Termékek, amelyek fontosak a belső munkafolyamatokhoz és változhatnak az időben.
- Eszközök, amelyek az egyes munkafolyamatokhoz kellenek
- Megvásárolt termékek
- Minden más, ami a fenti termékek előállításához, leírásához, kezeléséhez szükséges.

Konfigurációmenedzsment rendszer létrehozása

- Tárolási pontok meghatározása
  - Dinamikus: Helyben a fejlesztőnél tárolt verziók
  - Kontrolált, vagy központi tárolási pont: Központi tároló pontja a jelenlegi fejlesztésnek
  - Statikus, archívum: A már kibocsájtott release-ek archívuma
- Tipikus konfigurációs életciklusok meghatározása
- Hozzáférési jogok kiadása
  - Az egyes hozzáférési szintek meghatározása
  - A résztvevőkhöz felelősségek és jogok társítása

Példa tipikus konfigurációs életciklusra



Konfiguráció menedzsment eszközei

- Egyszerűbb esetekben tipikusan valamilyen verziókövető rendszert használnak

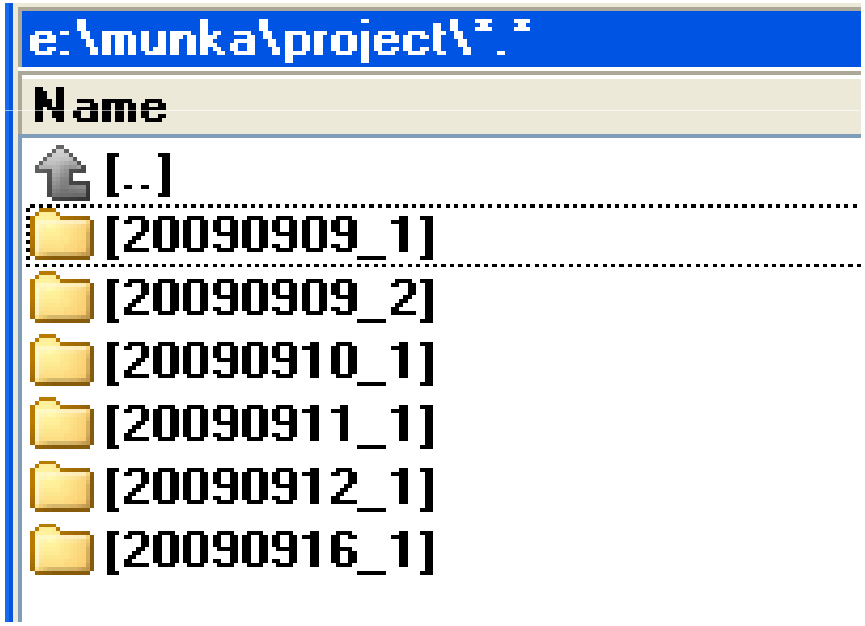
Verziókövető rendszerek
Az igény kialakulása

- Egy tipikus fejlesztési probléma:
  - Az eddig futó alkalmazáshoz új kódrészletet rakunk
  - Az alkalmazás lefagy
  - Visszaállítjuk a módosításokat
  - Az alkalmazás még mindig fagy…

- A helyzet csak bonyolódik, ha nem egyedül dolgozunk:
  - Változtatunk a működő alkalmazáson
  - De valaki más is beleír egy picit
  - Az alkalmazás lefagy

Triviális verziókövetés

- Naponta egy külön könyvtárba bemásoljuk az aznapi munkánk eredményét
- Project mellé mellékelünk egy *changelog* file-t is



Triviális verziókövetés

- Naponta egy külön könyvtárba bemásoljuk az aznapi munkánk eredményét
- Project mellé mellékelünk egy *changelog* file-t is

Problémák

- A másolatok komplett másolatok, tehát sok helyet igényelnek.
- Milyen gyakorisággal készítsünk másolatot?
- Csak működő verziót másoljunk fel, vagy köztes verziót is?
- Hogyan tudjuk követni, hogy min változtattunk?
- A change file-t nagyon pontosan kell nyilvántartani, különben inkább kártékony, mint hasznos, de nincs semmilyen automatizmus, ami erre ösztönözne.

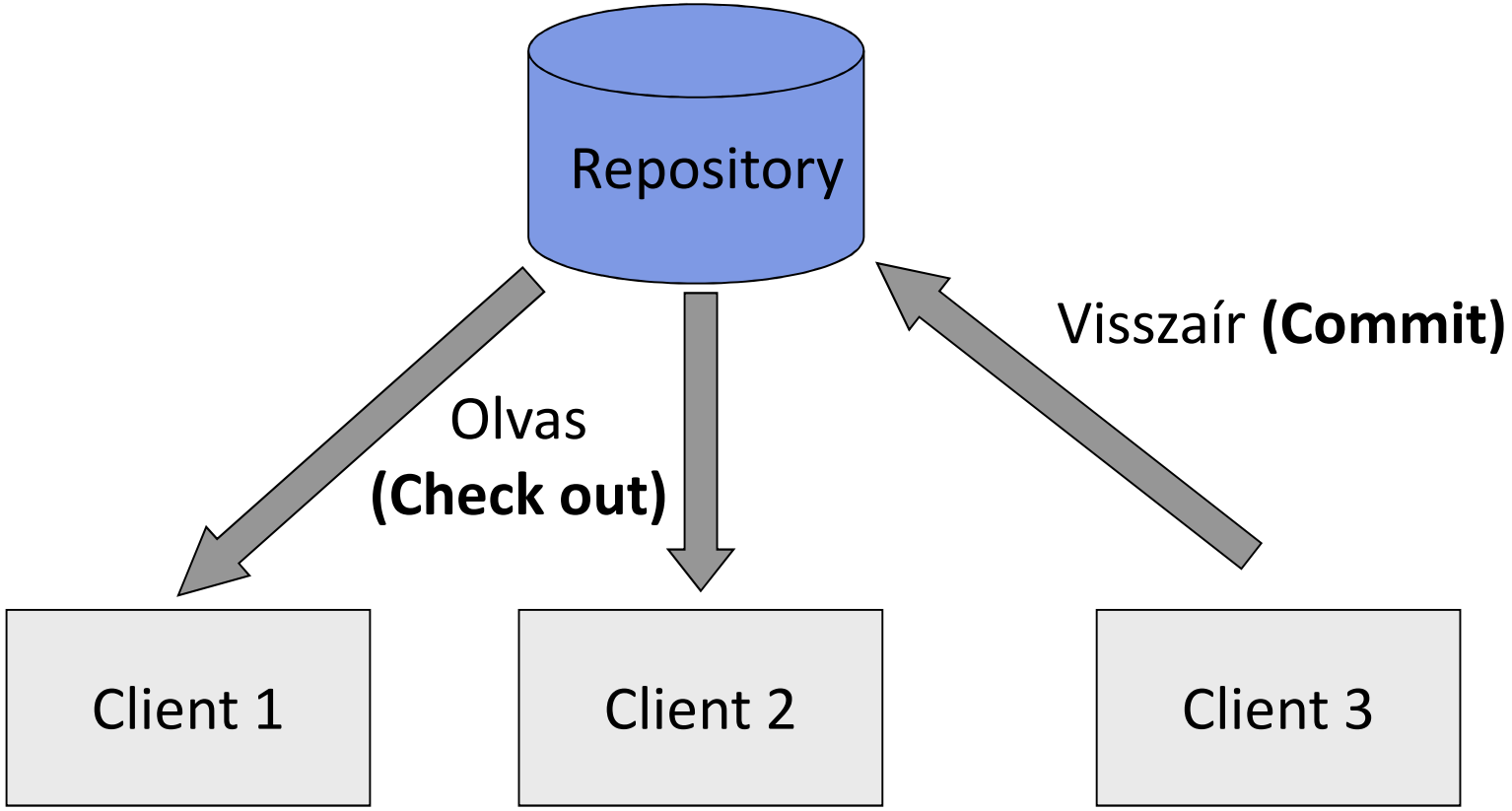
Centralizált verziókövető rendszerek
Alaptulajdonságok

- A verziómenedzsment alapja nem más, mint egy adott project összes változásának nyilvántartása.
- Egy verziómenedzsment rendszer nyilvántart
  - minden egyes file-on létrehozott változást,
  - a könyvtárstruktúrát érintő minden változást.
- A felhasználónak lehetősége van
  - megtekinteni a project vagy egy file állapotát egy adott pillanatban,
  - megtudni, hogy ki, mit és mikor változtatott az adott projecten
  - kommentet tenni minden változtatás mellé

Centralizált verziókövető rendszerek
Alapfogalmak

- Repository (raktár): *Központi nyilvántartása az adatoknak vagy projectnek (a master copy).*
- Client: *Felhasználó, aki dolgozni kíván a projecten.*
- Working copy: *Egy Client által a projectből létrehozott munkaváltozat, amit szabadon változtathat.*

Centralizált verziókövető rendszerek
Működési alapok



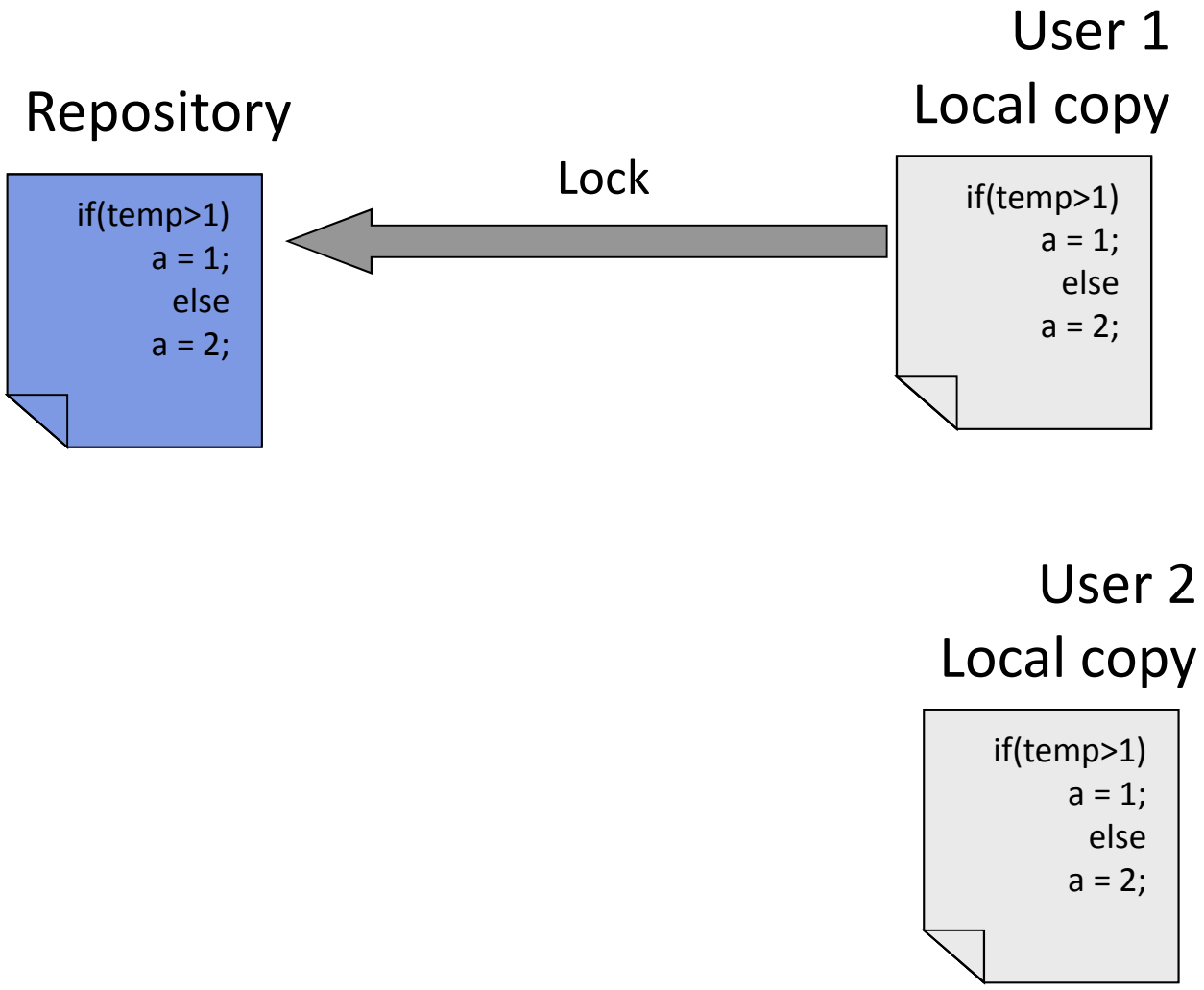
Verziómenedzsment stratégiák: *A probléma*

- Hogyan támogatja a verziókövető rendszer, hogy a felhasználók együtt dolgozzanak, de mégse lépjenek egymás lábára?
- Ilyen stratégia nélkül könnyen előfordulhat, hogy egy file-t vagy projectet egyszerre többen módosítanak, majd felülírják egymás módosításait (a módosítások nem tűnnek el, de nem is kerülnek bele az új verzióba).

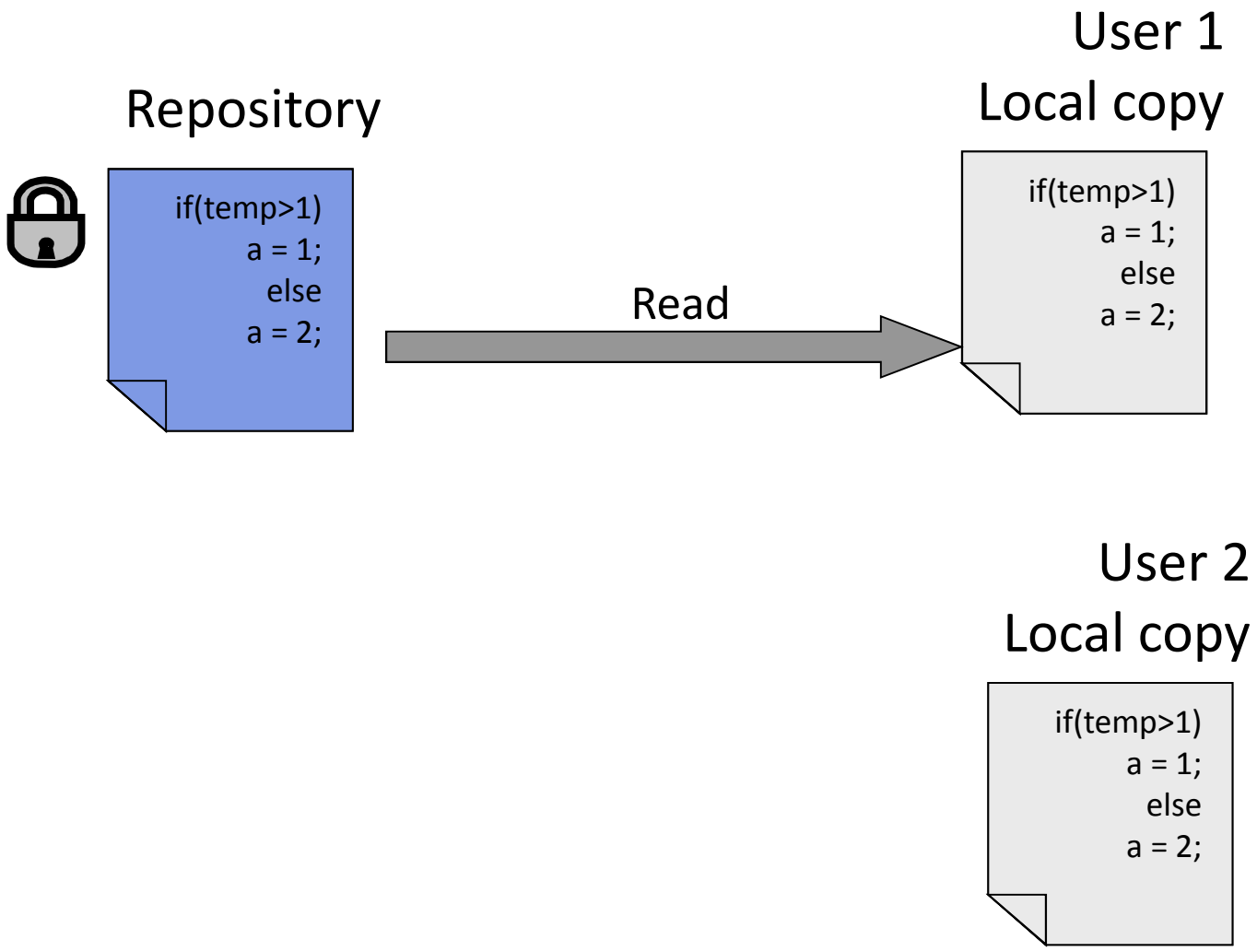
A Lock–Modify–Unlock megközelítés

- Módosítás előtt le kell lockolni egy file-t.
- Tehát egyszerre csak egy ember tudja módosítani a file-t. Olvasni tudja más is.

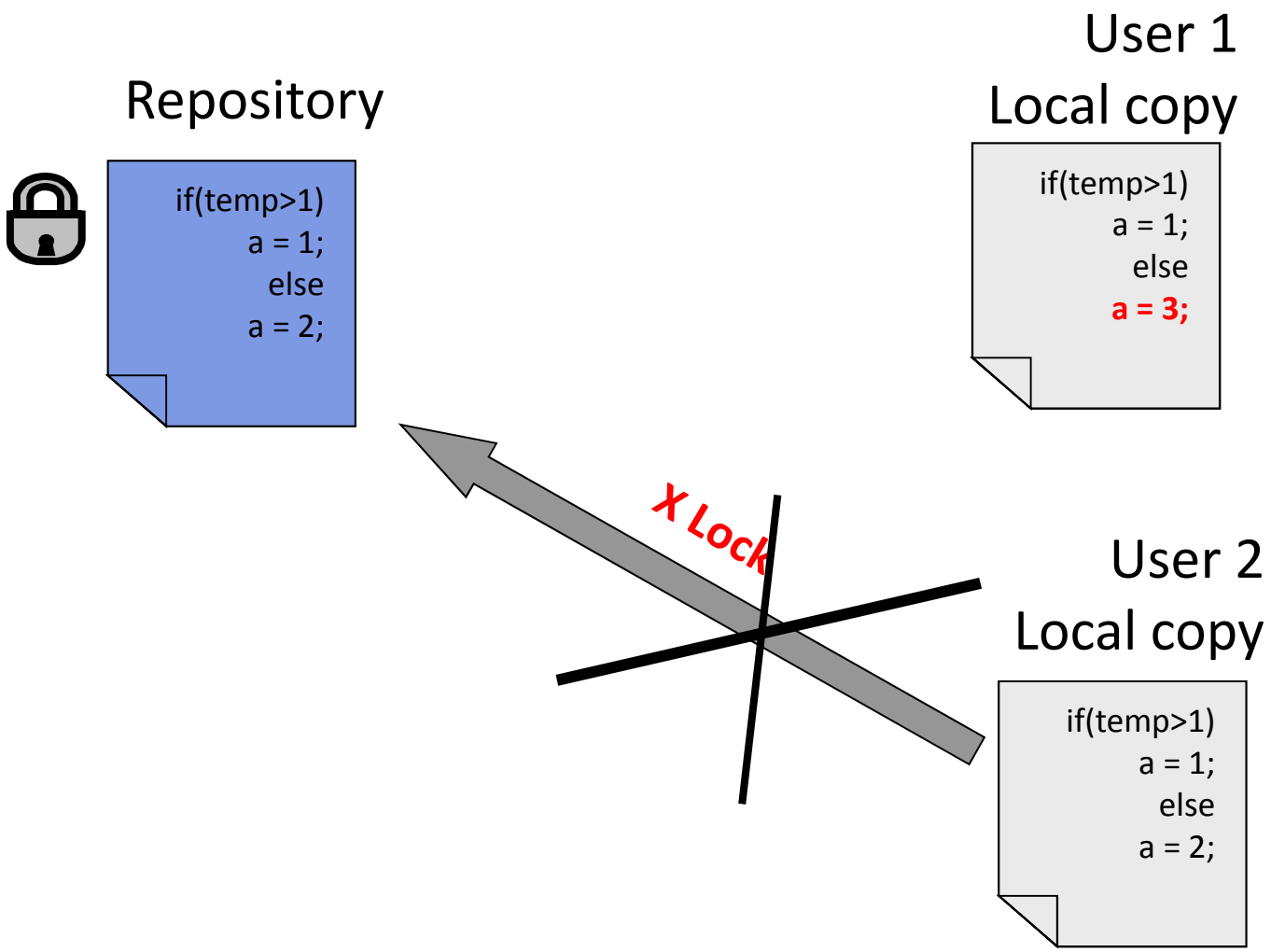
Lock–Modify–Unlock megközelítés



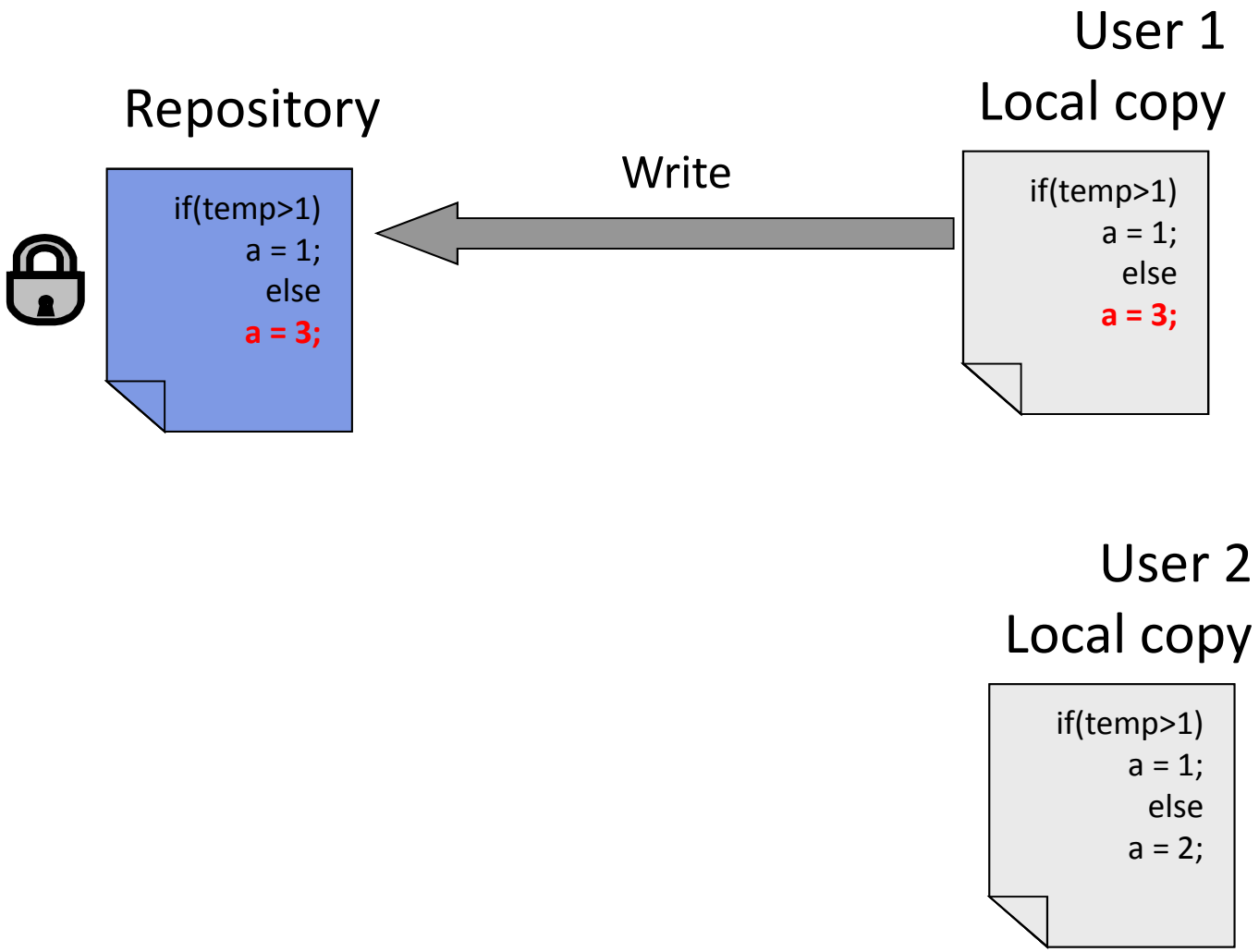
Lock–Modify–Unlock megközelítés



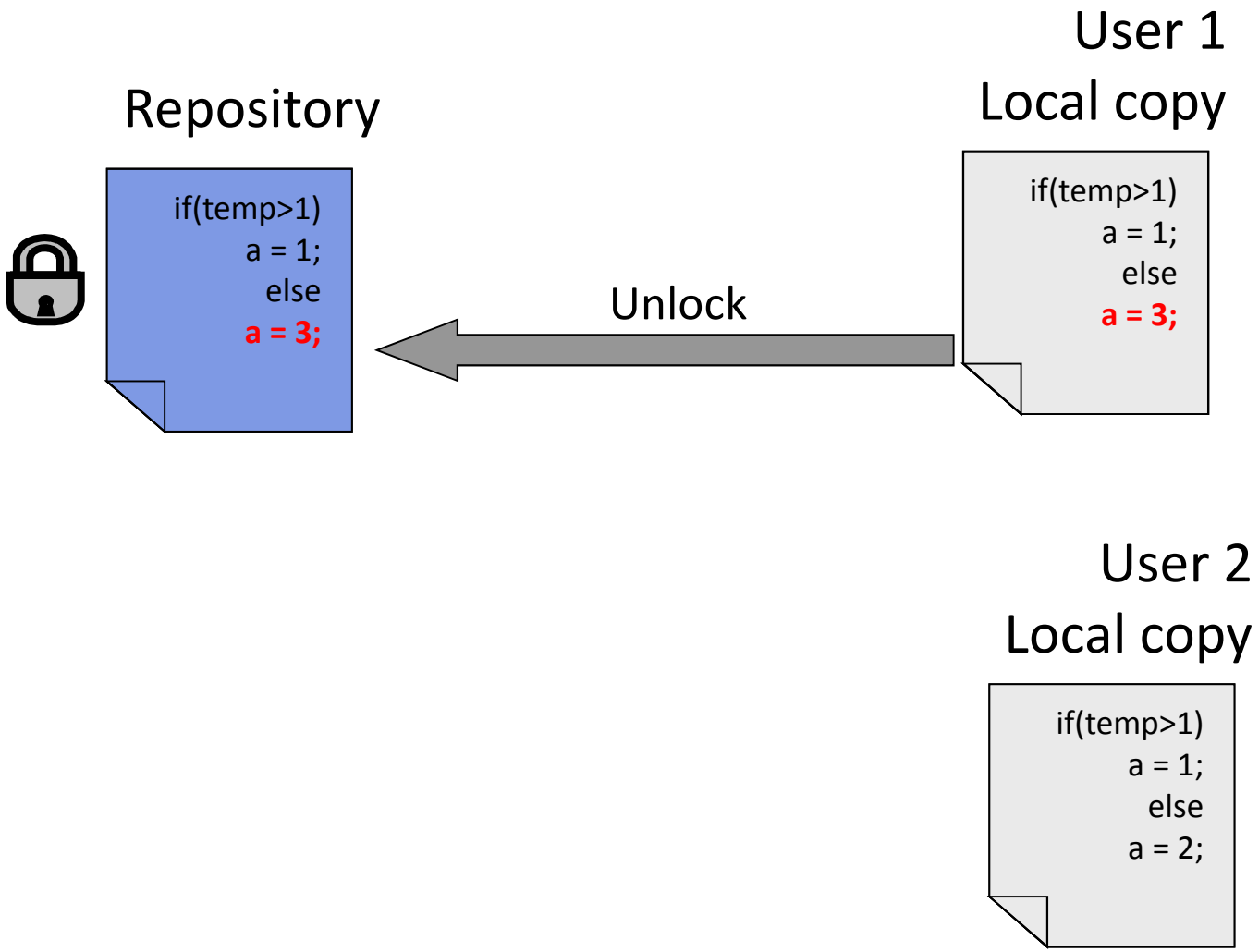
Lock–Modify–Unlock megközelítés



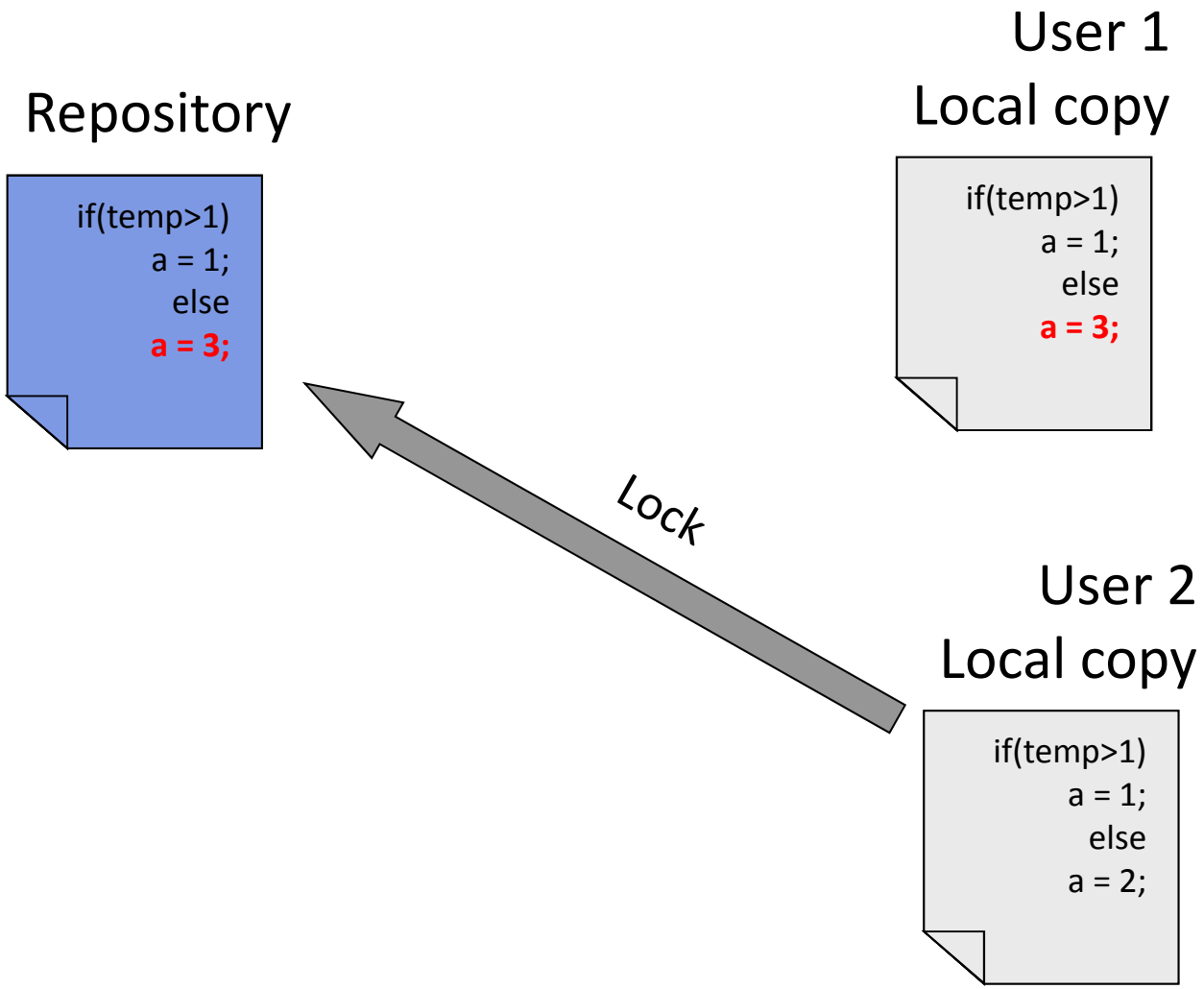
Lock–Modify–Unlock megközelítés



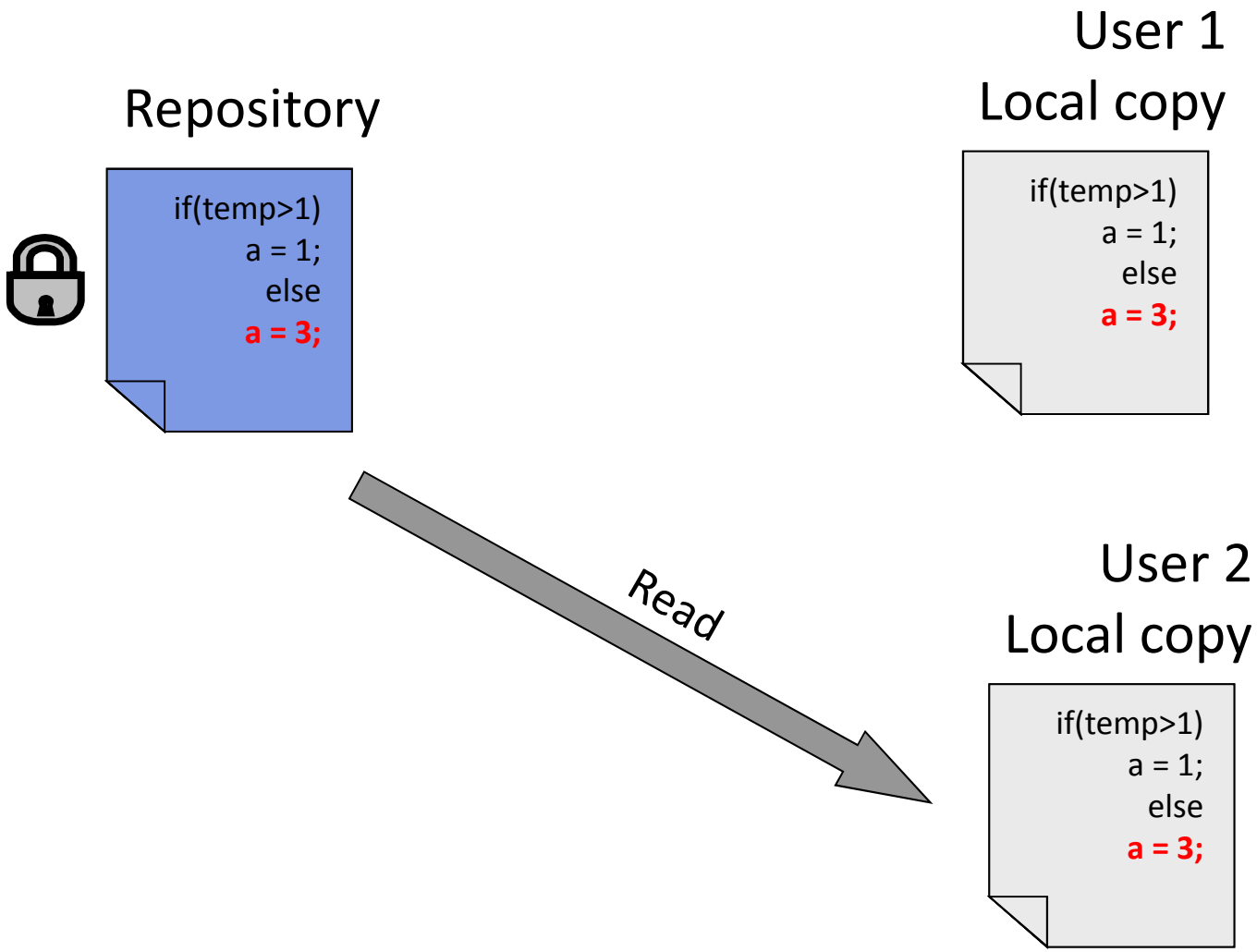
Lock–Modify–Unlock megközelítés



Lock–Modify–Unlock megközelítés



Lock–Modify–Unlock megközelítés



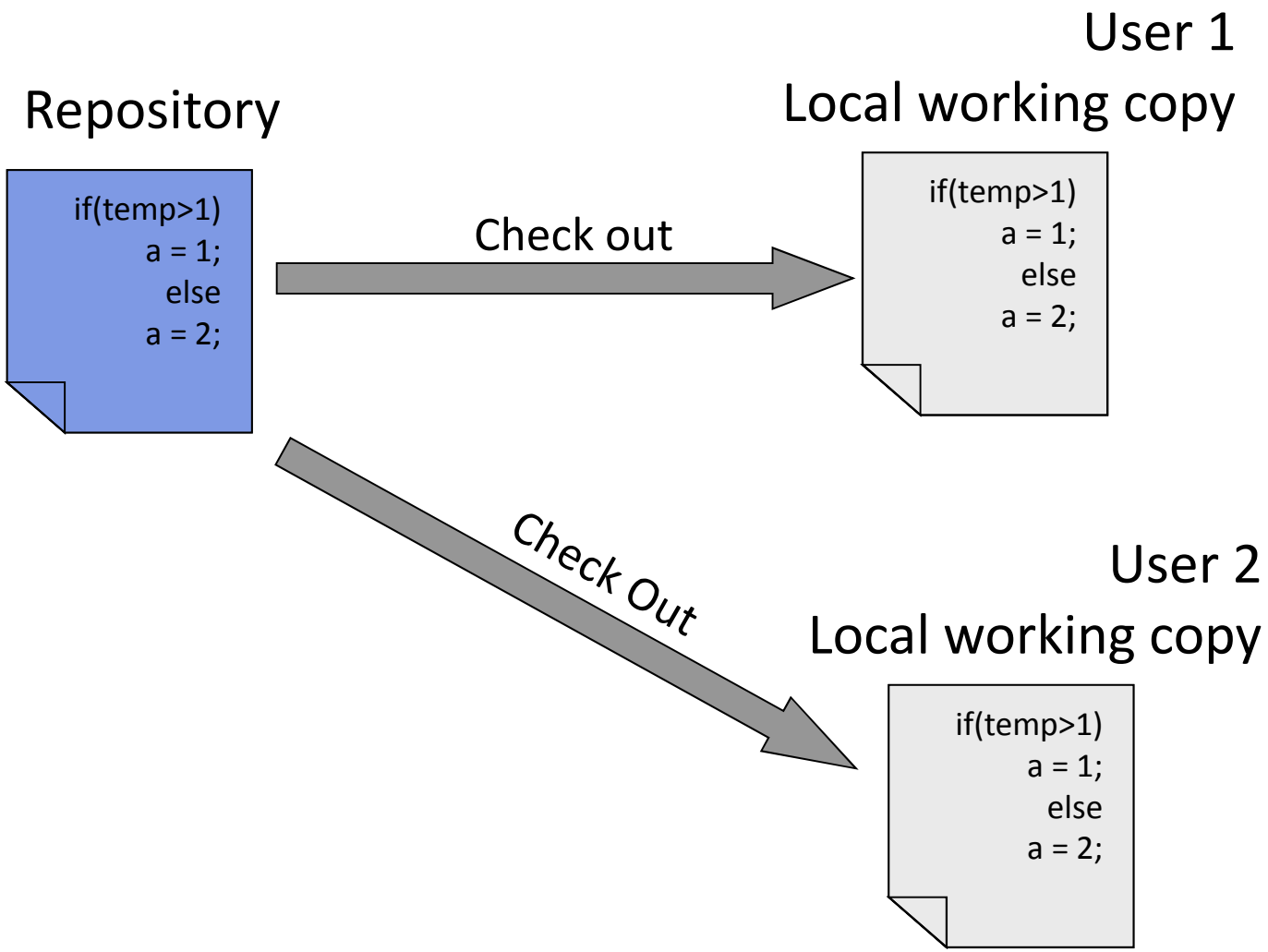
A Lock–Modify–Unlock megközelítés problémái

- Adminisztratív problémákhoz vezethet:
  - Ha egy fejlesztő elfelejt kilockolni egy file-t, akkor más nem férhet hozzá.
  - Ha szabadságra megy, akkor pl. rendszergazda kell.
- Felesleges egymásra várást okozhat.
  - Egy C file-on belül például valaki az F1 függvényt akarja módosítani, másvalaki pedig az F2-t. Semmi köze a kettőnek egymáshoz mégsem tudják egyszerre megcsinálni.
- A biztonság hamis illúzióját keltheti.
  - Például két fejlesztő dolgozik ugyanazon a projecten, az egyik lockolja az A file-t, a másik a B file-t. A két file között függőség áll fent. Mindketten azt hiszik biztonságban vannak, holott mégsem.

A Copy–Modify–Merge megközelítés

- Egyszerre több fejlesztő is ki „check out”-olhatja ugyanazt, mindenki a saját Working copy-ját használja.
- Working copy: A Repository (vagy annak egy részének) saját gépen található leképezése.
- A létrejövő konfliktusokat pedig Merge-gel, tehát fuzionálással oldják fel, és így hoznak létre egy új verziót.
- A Merge, bár támogatva van a verziókövető rendszer által, alapvetően mégis emberi döntéseket követel, tehát nem automatikusan történik.

A Copy–Modify–Merge megközelítés
a két fejlesző „check out”-ol



A Copy–Modify–Merge megközelítés
mindketten módosítanak

Repository

```
if(temp>1)
    a = 1;
  else
    a = 2;
```

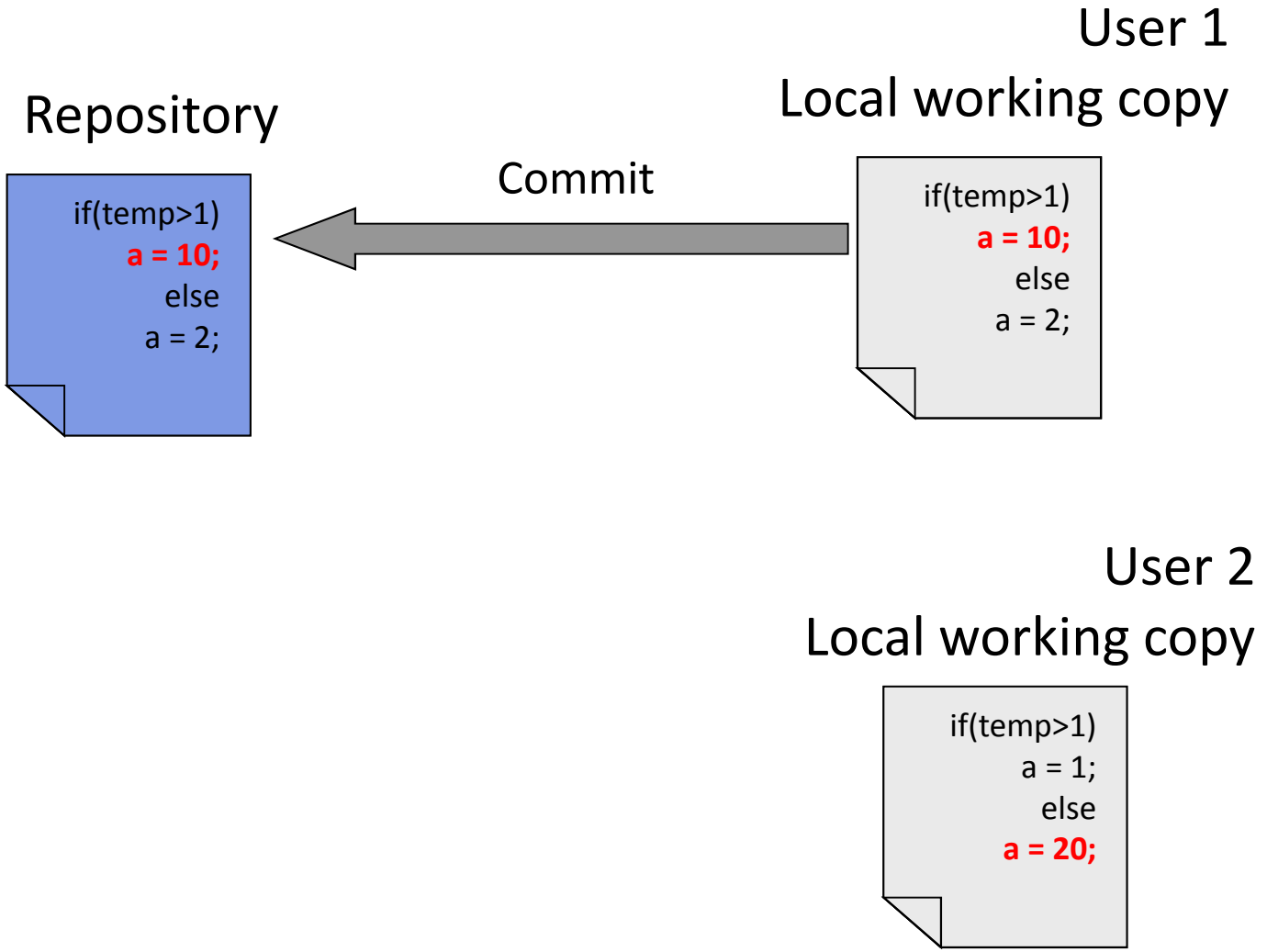
User 1
Local working copy

```
if(temp>1)
    a = 10;
  else
    a = 2;
```

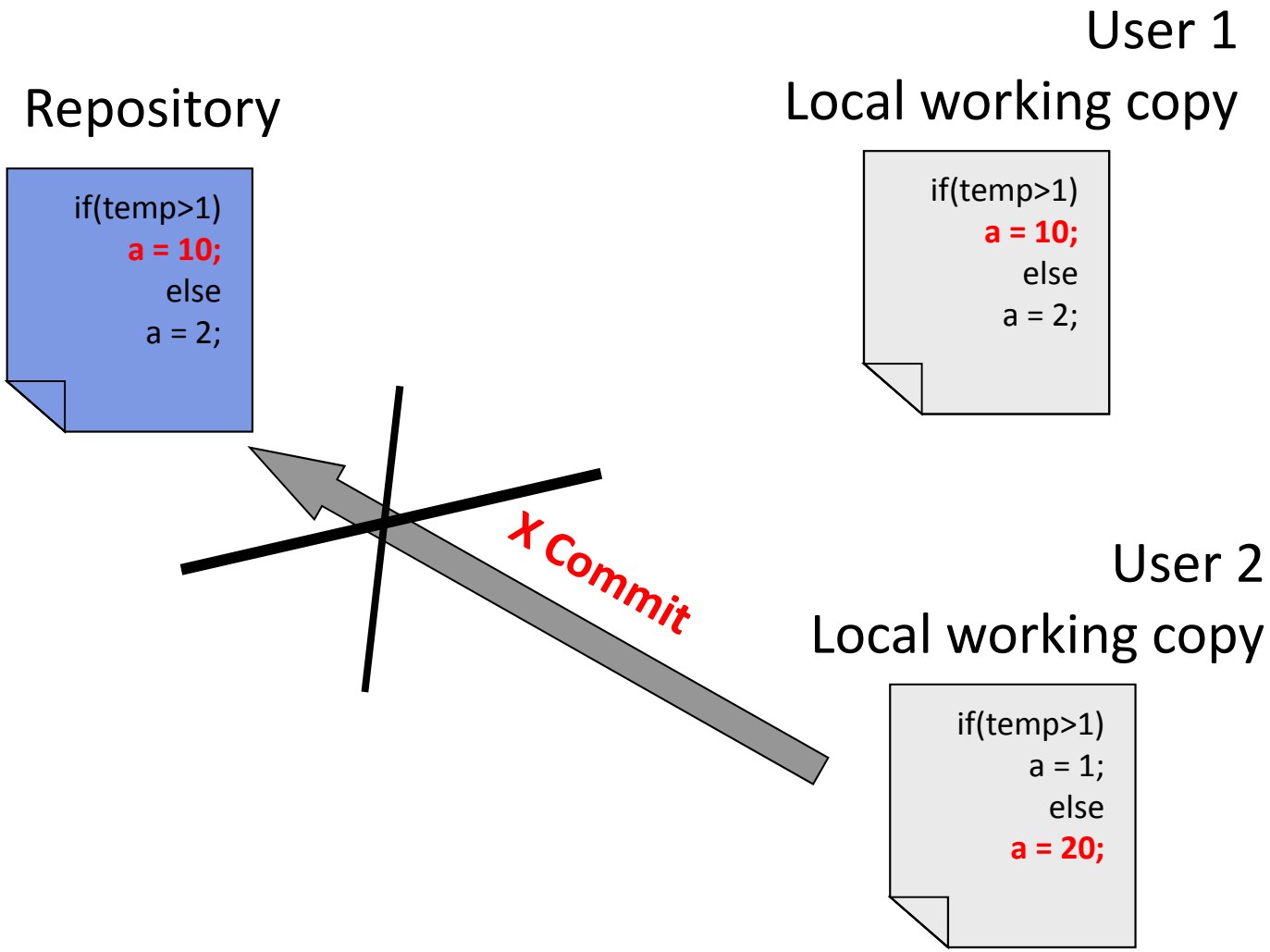
User 2
Local working copy

```
if(temp>1)
    a = 1;
  else
    a = 20;
```

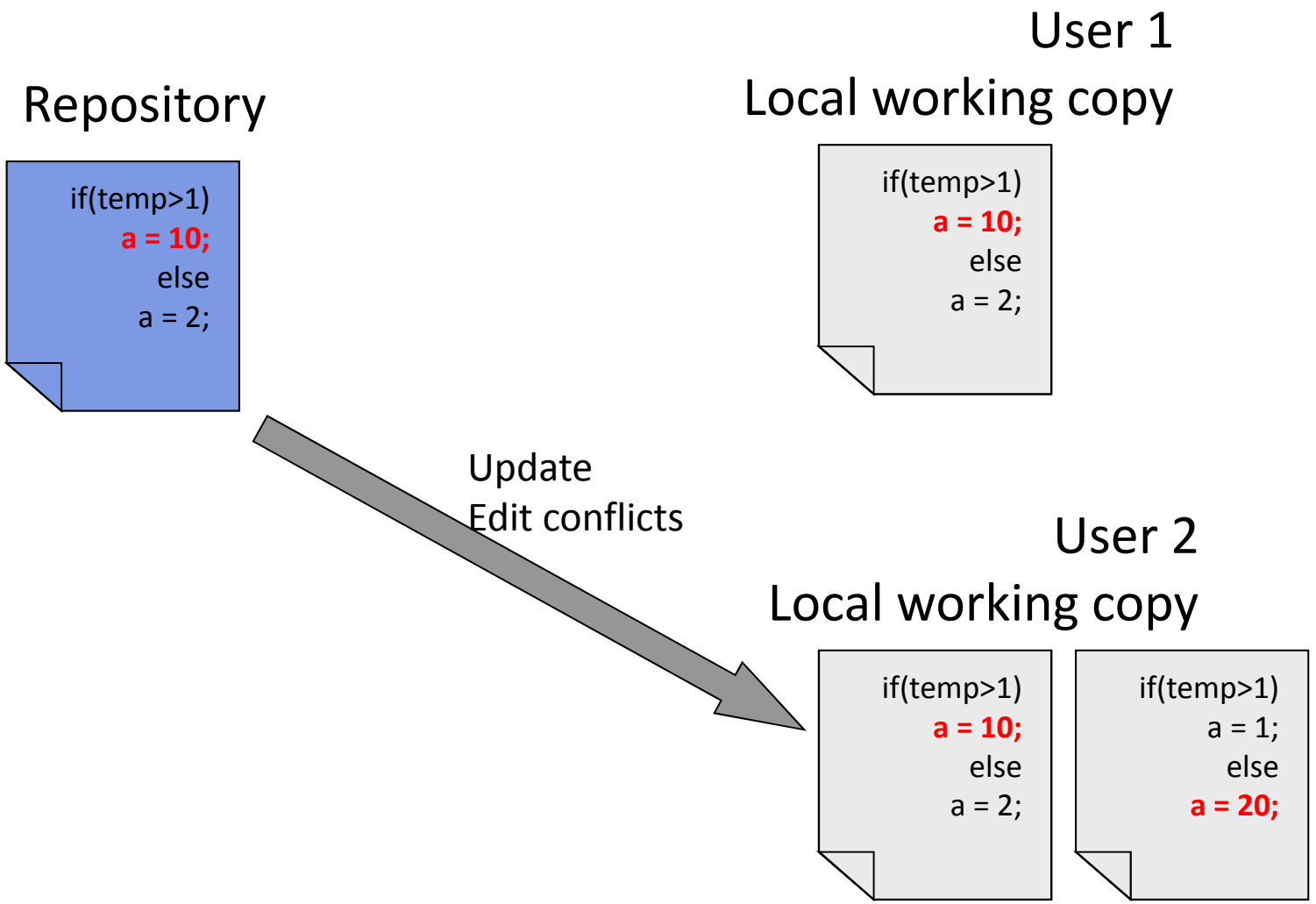
A Copy–Modify–Merge megközelítés
User 1 végzett, feltölti a módosításokat



A Copy–Modify–Merge megközelítés
User 2 nem tudja feltölteni a módosításokat

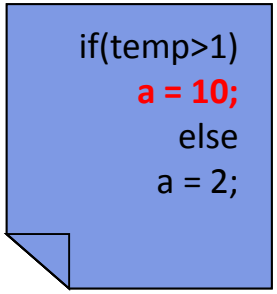


A Copy–Modify–Merge megközelítés
User 2 frissít az új verzióra

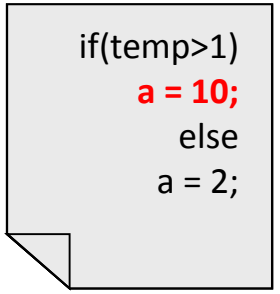


A Copy–Modify–Merge megközelítés
User 2 merge-el

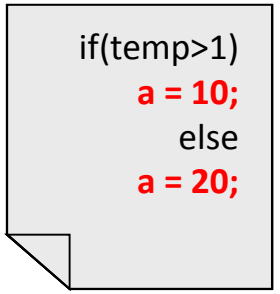
Repository



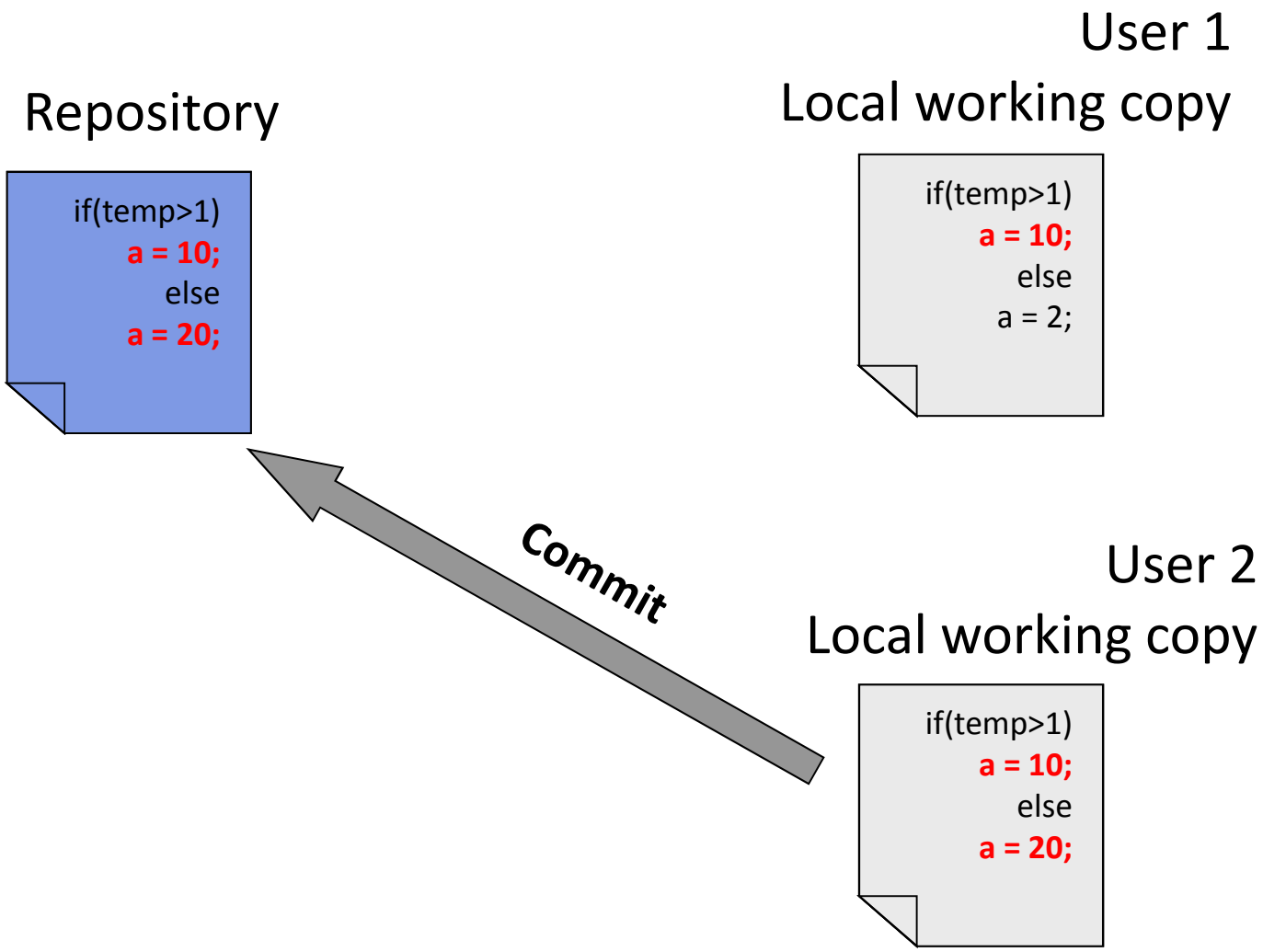
User 1
Local working copy



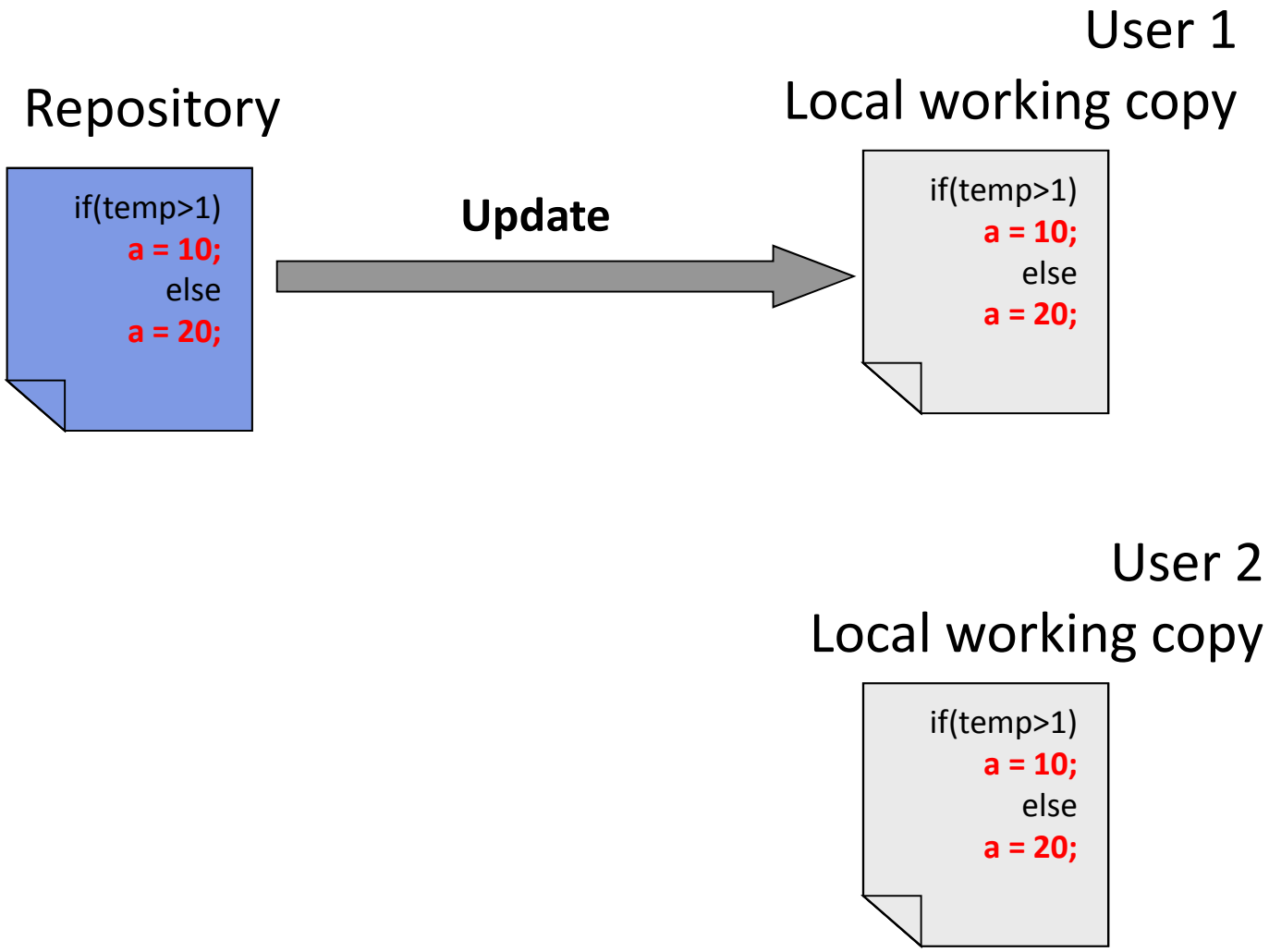
User 2
Local working copy



A Copy–Modify–Merge megközelítés
User 2 feltölti az új verziót



A Copy–Modify–Merge megközelítés
User 1 Update-el



A Copy–Modify–Merge megközelítés
előnyök és hátrányok

- Egyszerre több fejlesztő is dolgozhat ugyanazon a kódon.
- Commit-nél az esetleges konfliktusok kiderülnek.
- Embernek kell döntenie a konfliktus feloldásáról.
- A verziókövető rendszer nem helyettesíti az emberek közötti kommunikációt.

Melyek azok az esetek, amikor mégis lock-ot kell használni?

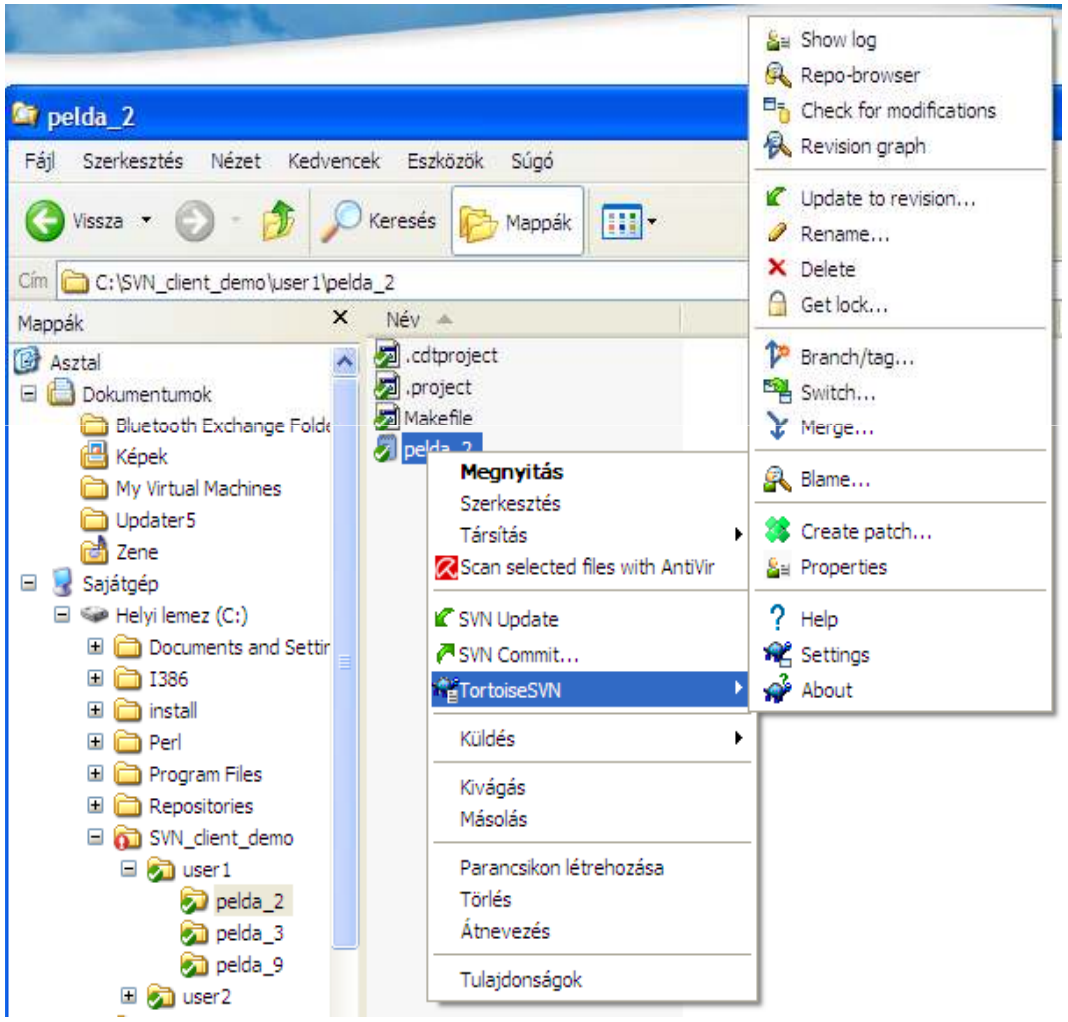
- Olyan bináris jellegű file-ok esetében, ahol a merge nem megoldható.
  - Például, hangfile-ok, bináris file-ok
  - NYÁK-tervek, kapcsolási rajzok

- Ezért a legtöbb verziókövető rendszerben megtartották a lock funkciót.

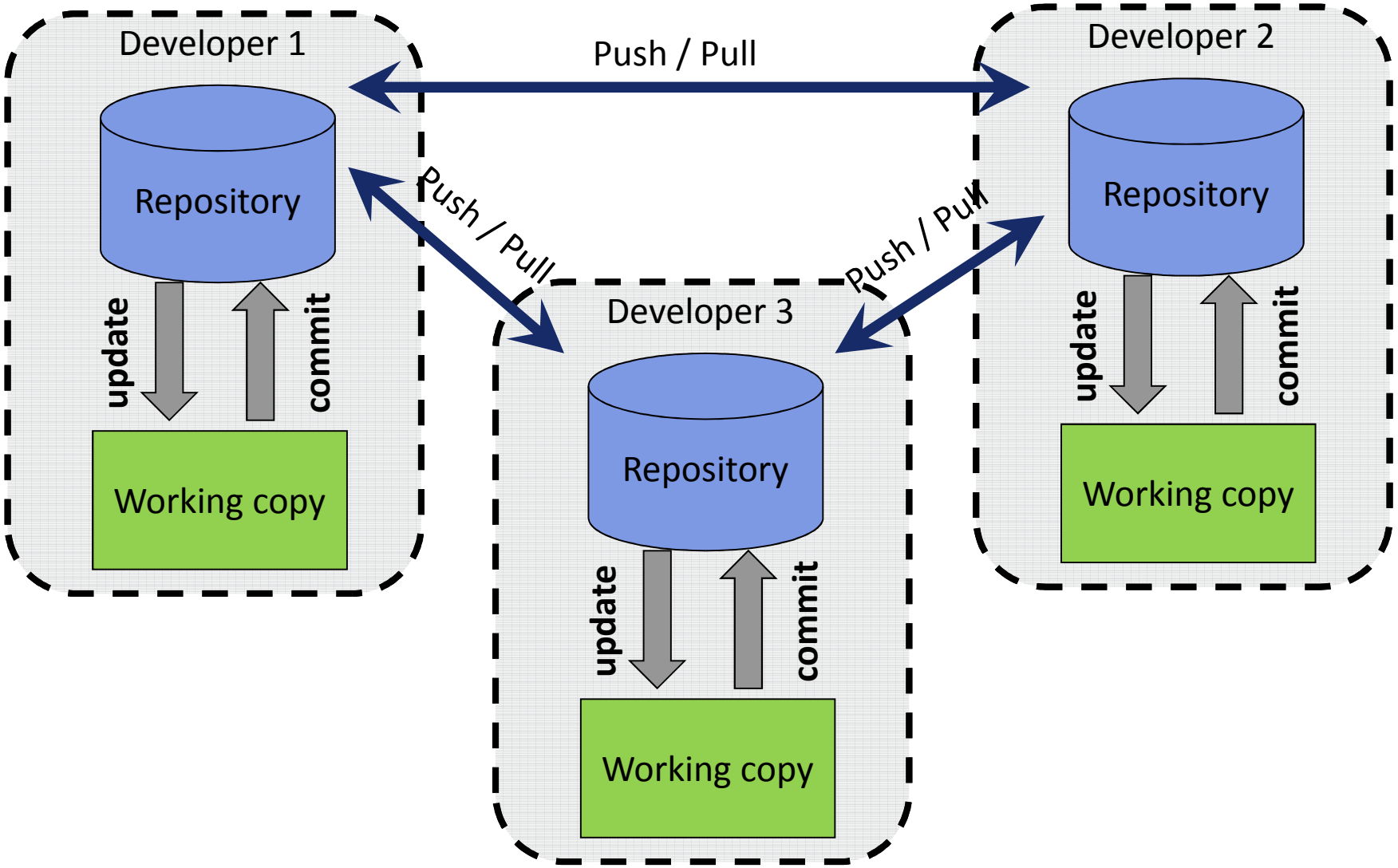
Centralizált verziókövető rendszerek SVN, szerver oldal

Centralizált verziókövető rendszerek SVN, kliens oldal, TortoiseSVN

- Ingyenes SVN kliens (létezik CVS változat is)
  - http://tortoisesvn.net/
- Windows-ba beépülő felhasználói felület



Elosztott verziókövető rendszerek
Distributed revision control



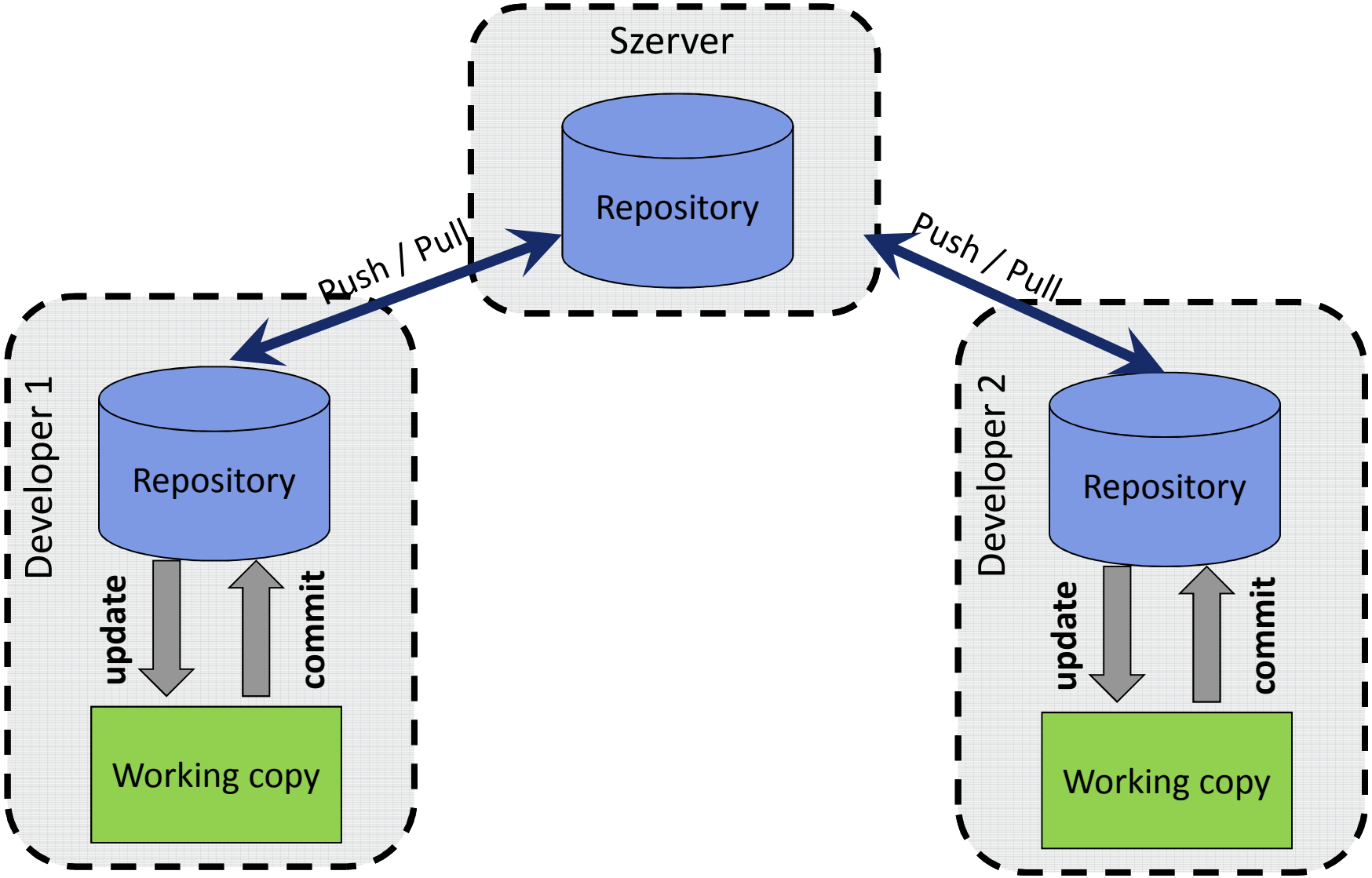
Elosztott verziókövető rendszerek előnyök

- Mindenkinek megvan a saját játszótere
  - Saját repo-ba tetszőleges update-ek
  - A saját repo előzményei logjai könnyen hozzáférhetőek
- Off-line is működik
  - A centralizált verzióknál állandóan on-line kell lenni és a server is állandóan elérhető kell, hogy legyen
- Gyors
  - Az alapváltoztatások lokálisan végrehajthatóak
- Kevés menedzsment igény
  - Nem kell szervert üzemeltetni
- Könnyű branch-elni
  - Gyakorlatilag minden külön fejlesztő külön branch-et visz

Elosztott verziókövető rendszerek hátrányok

- Backup-ra továbbra is szükség van
  - A többieknél lévő verzió egy branch-nek tekinthető, tehát nem egyenértékű a tieddel
- Gyakorlatilag nincs olyan, hogy jelenlegi verzió
  - Szintén abból ered, hogy mindenki gyakorlatilag független Branch-et visz
- Nem nagyon léteznek revizió számok
  - Minden új változásnak általában GUID-ja van, ami nem mutat folytatólagosságot

Gyakorlatban alkalmazott, elosztott verziókövető rendszerek

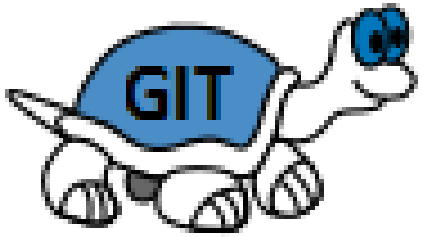


Elosztott verziókövető rendszerek GIT „szerver oldal”

- Gyakorlatilag nincs olyan, hogy szerveroldal, de ha központi tárhelyet akar valaki, akkor Pl: GitHub (**26 millió** repo)

Elosztott verziókövető rendszerek
GIT „kliens olda”

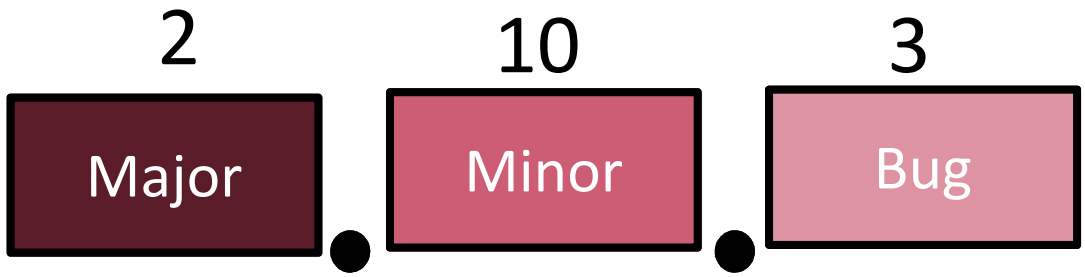
- Gyakorlatilag nincs ilyen sem …
- GitHub for windows
- TortoiseGIT



tortoisegit

Windows Shell Interface to Git

Verziószámok kezelése
Semantic Versioning



- **Major:** nagy változtatás. Általában ez olyan funkció / API változtatást jelent, ami inkompatibilis az előző verziókkal.
- **Minor:** Olyan kisebb funkcionalitás hozzáadása a rendszerhez, ami visszamenőlegesen kompatibilis (backwards-compatible)
- **Bug:** Visszamenőlegesen kompatibilis (backwards-compatible) hibajavítások.