BME TMIT
2022

14/13

Németh Gábor
Janky Ferenc Nándor előadásvázlatai alapján

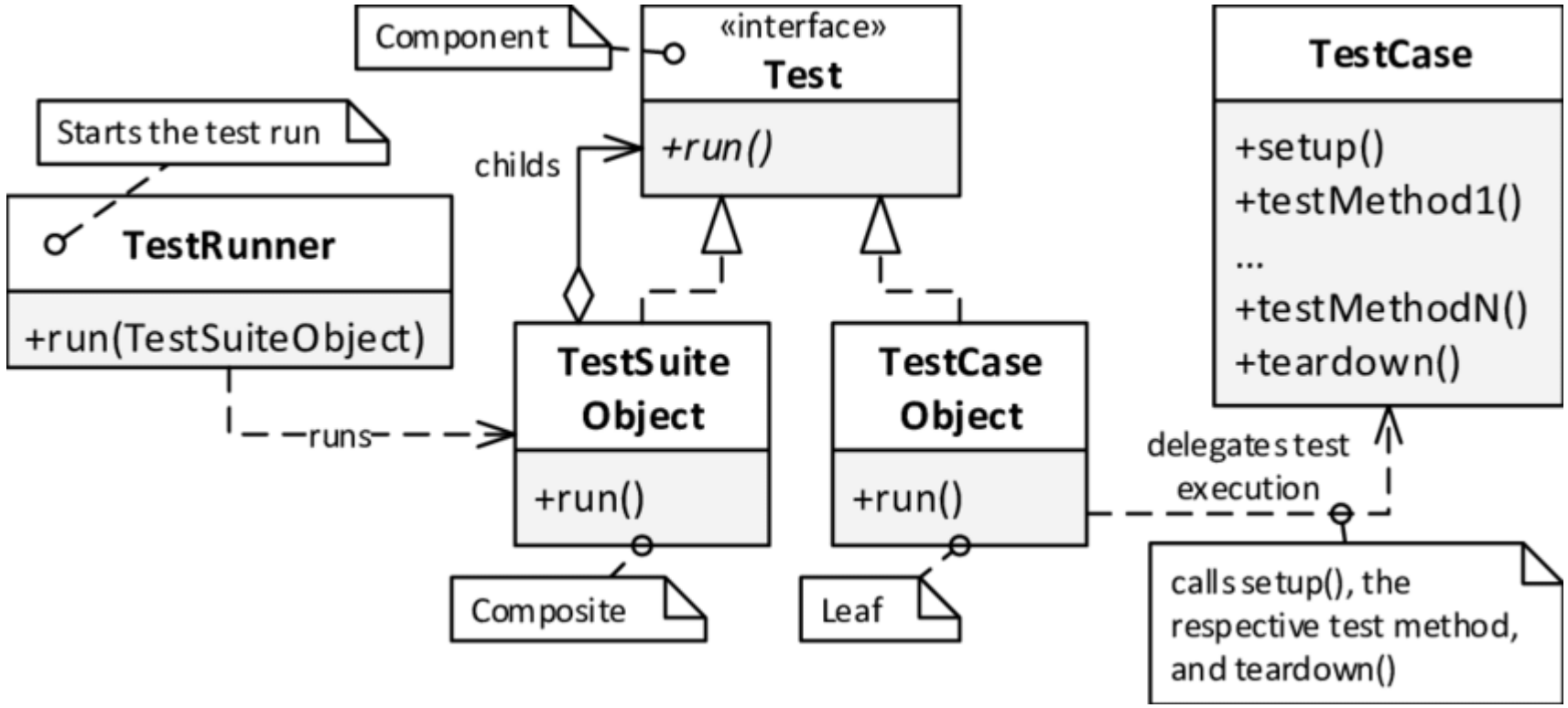
Funkcionális programozás C++-ban

Tesztelési módszerek

xUnit keretrendszerek I.

- Gyűjtőnév, olyan unit-test (egység teszt) keretrendszerek, amelyek a funkcióik és szerkezetük alapján a SmallTalk SUnit keretrendszerére hasonlítanak, például
  - Goole Test
  - CPPUnit
  - BoostTest
  - Catch2
  - stb.

xUnit keretrendszerek II.



Forrás: Plewnia, Christian. (2015). A Framework for Regression Test Prioritization and Selection. 10.13140/RG.2.2.11503.12968.

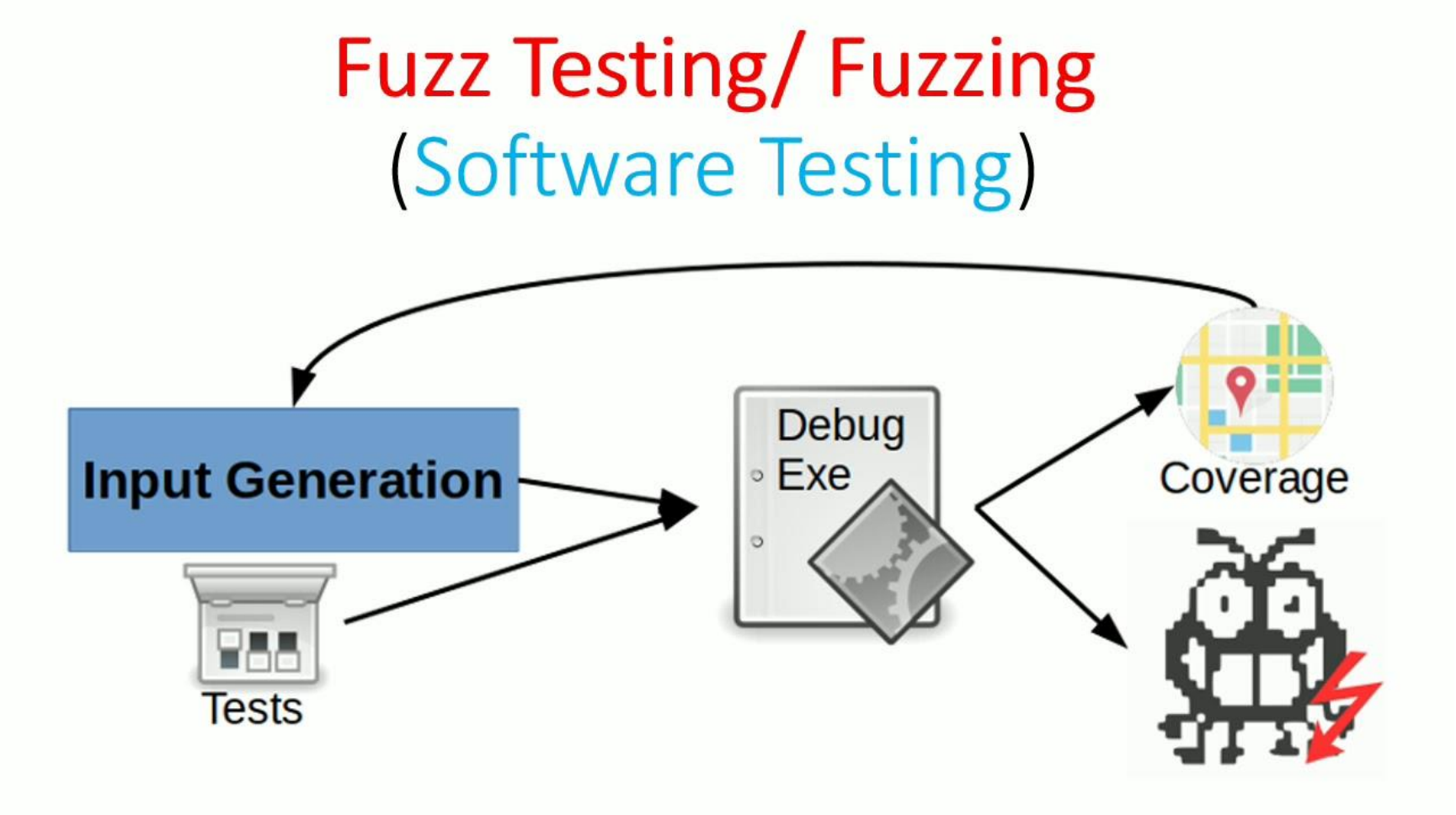
xUnit keretrendszerek elemi

- **Test runner:** futtatható program, amely automatikusan végrehajtja a teszteket, és jelentést generál az eredményről
  - általában gazdag filterezési lehetőség
  - többféle riport formátum, időzítés, véletlen sorrend támogatása, stb.
- **Test case:** alapvető egység, teszt eset
- **Test suite:** logikailag összetartozó tesztesetek gyűjteménye
- **Test fixture:** teszt kontextus, pre- és poszkondíciók gyűjteménye amelyek egy teszt futtatásához szükségesek
  - általában a teszt gyűjtemény *setup()/teardown()* metódusai realizálják
- **Assertion:** állítások, melyek a SUT (sytem-under-test) –tel kapcsolatosak

Catch2 – unit test framework

- Modern C++ unit test keretrendszer
- Aktív fejlesztés alatt
- több stílusú tesztelést is támogat
- BDD stílusú leírás
- Számtalan riport formátum, jól integrálható CI rendszerekkel
- Csak fejállományból áll, nincs szükség bináris könyvtárakra

Fuzz testing

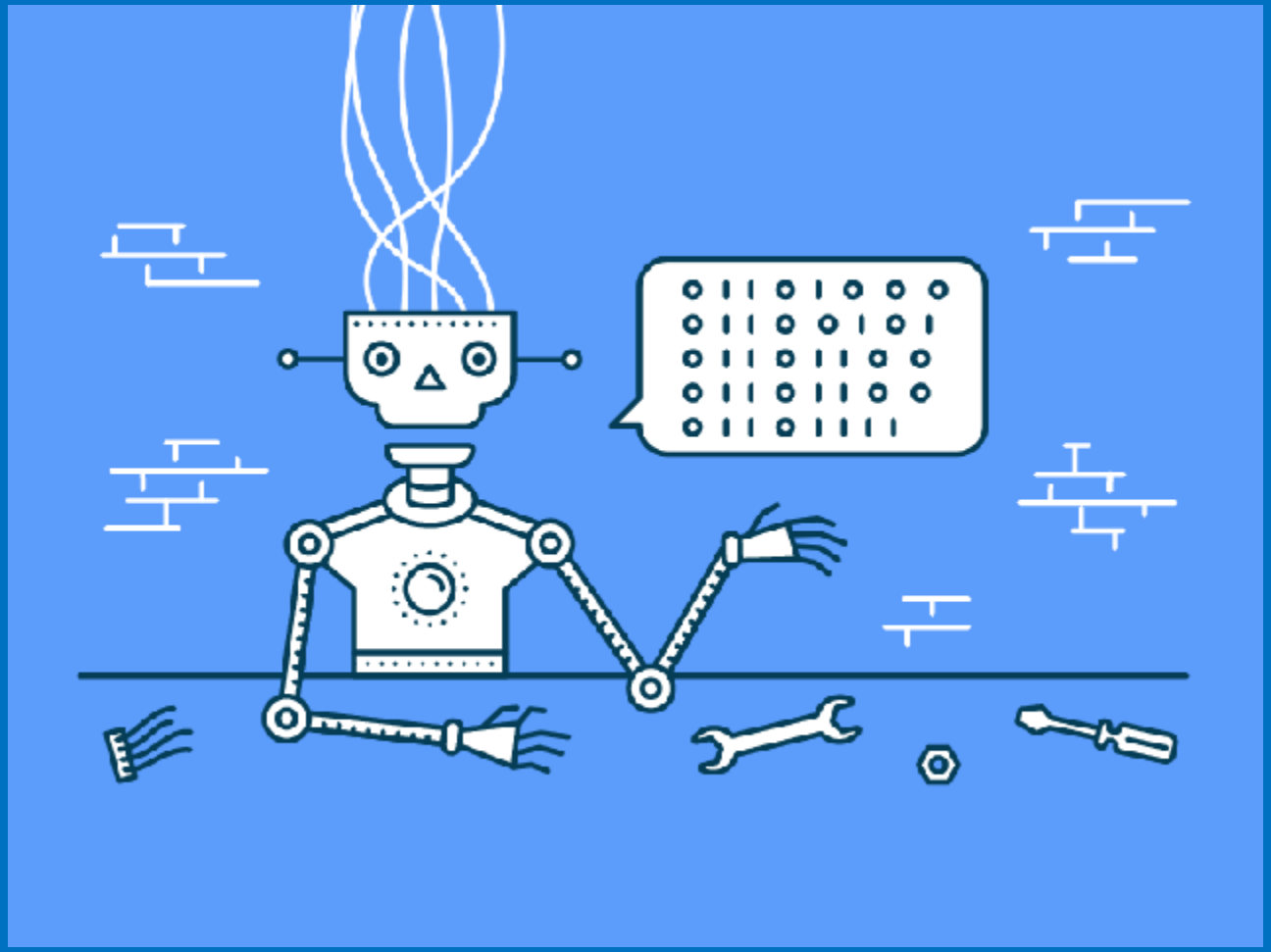


Forrás: https://i.ytimg.com/vi/pcEy-4eZF6g/maxresdefault.jpg

Fuzz testing

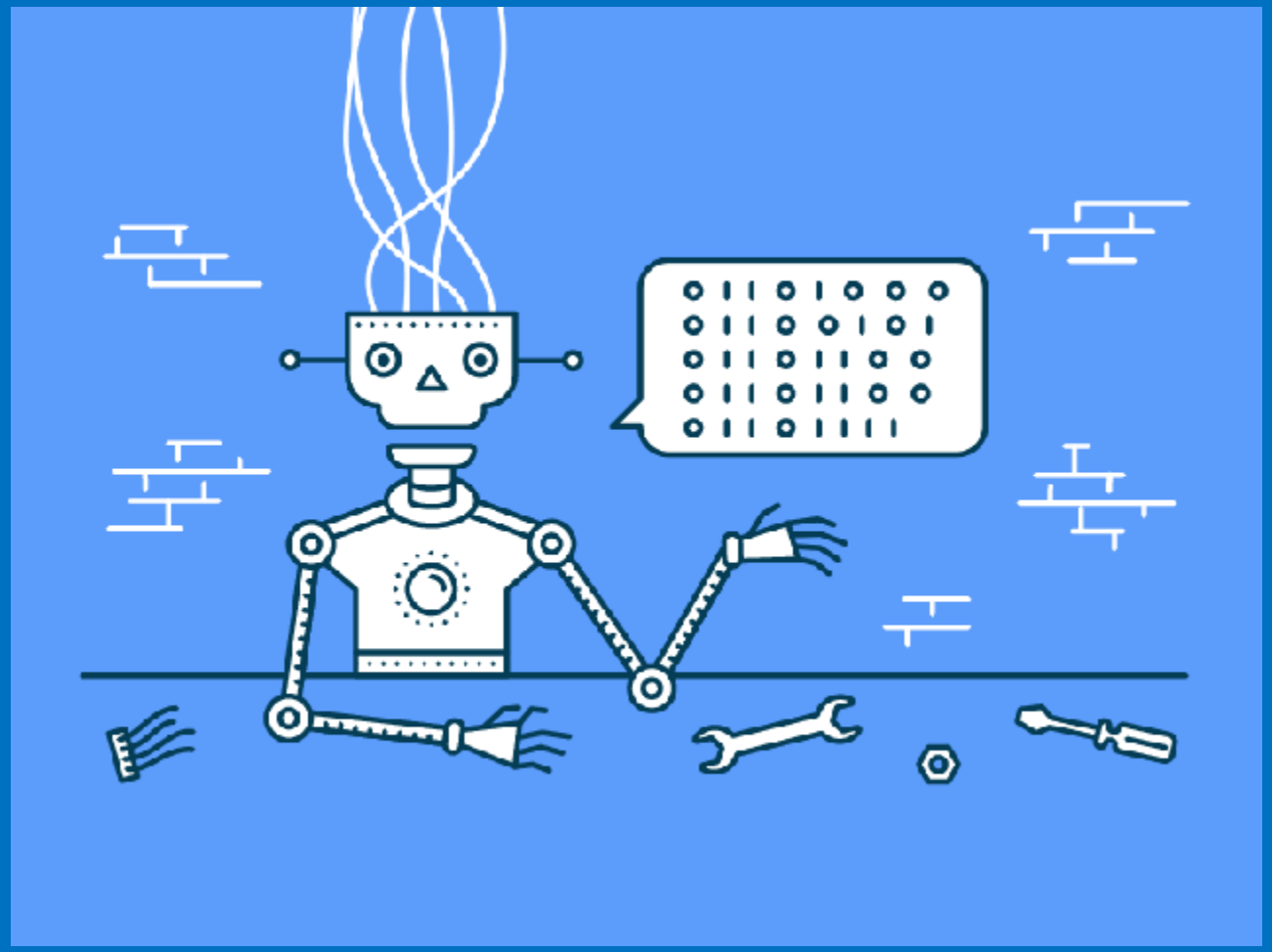
- Automatizált és „véletlenen” alapuló szoftvertesztelési eljárás
- Tipikusan a tesztelt program futása során meta-adatokat generál (pl. lefedettségi adatok)
- A fuzzer valamilyen stratégiát alkalmazva (pl. genetikus algoritmus) úgy változtatja a bemenet a meta-adatok alapján, hogy maximalizálja a lefedettséget
- Népszerű implementációk:
  - libAFL – Advance Fuzzing Library
  - libFuzzer

ex_0: Catch2 bemutató



Fordító, optimalizáló

ex_1: maybe (pointer) osztály



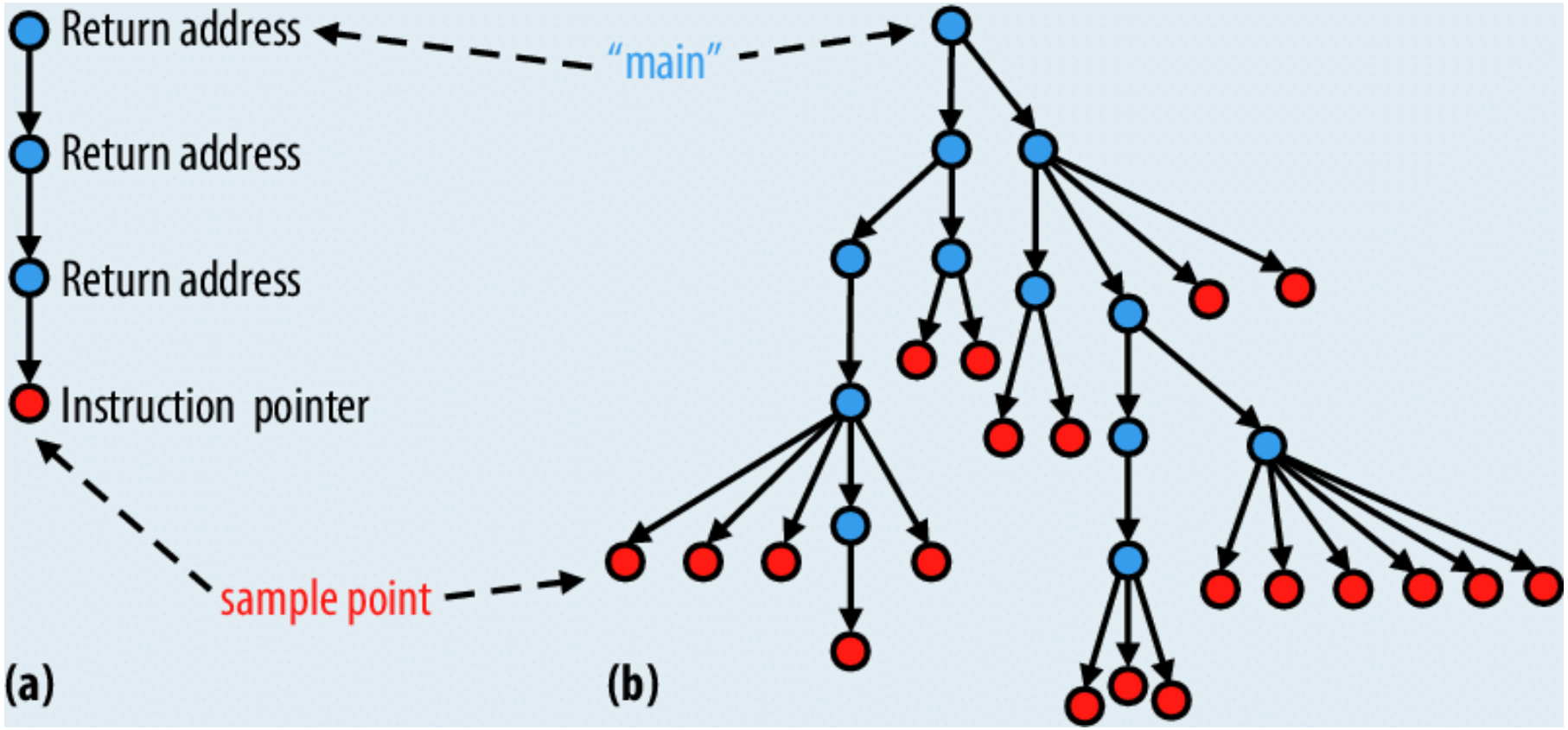
Teljesítmény mérés és optimalizáció

Teljesítmény mérés

- **Intruzív mérés**
  - speciálisan fordított bináris vagy emulátor használatával
  - általában nagyban befolyásolja a futási karakterisztikát
  - valós környezetben nem lehet mérni
  - pl.: *valgrind* vagy gcov,llcov stb.
  - pontosabb mérési eredmények

- **Mintavételes mérés**
  - alig befolyásolja a futási karakterisztikát
  - akár valós környezetben futó alkalmazást is mérhetünk
  - statisztikai alapon működik, nehezebb értelmezés
  - különböző fajtájú hardveres v. szoftveres számlálók vezérlik a mintavételt (UNHALTED_CYCLES, L1_CACHE_MISS, stb.)
  - *pl.: perf (linux)* vagy *vsperf (windows)*

Teljesítmény mérés (mintavételezés)

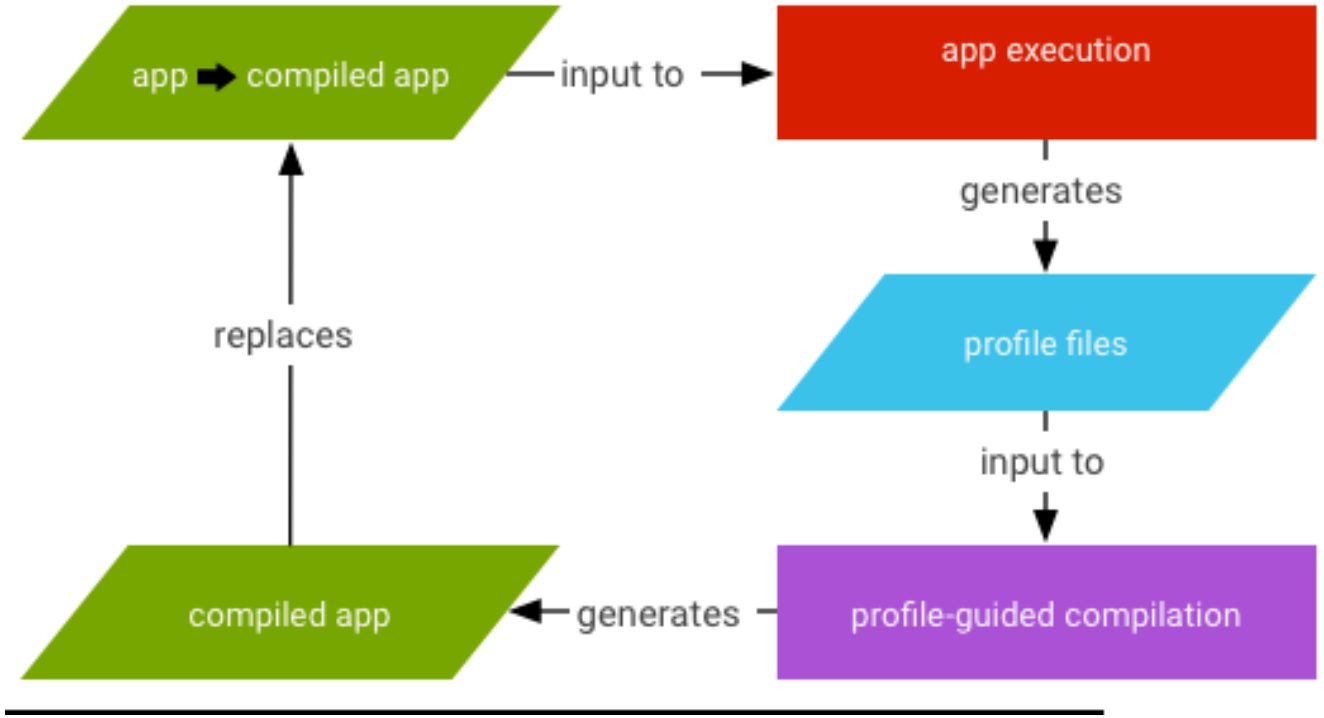


Forrás: Tallent, Nathan & Mellor-Crummey, John. (2009). Identifying Performance Bottlenecks in Work-Stealing Computations.. IEEE Computer. 42. 44-50.

Teljesítmény optimalizáció

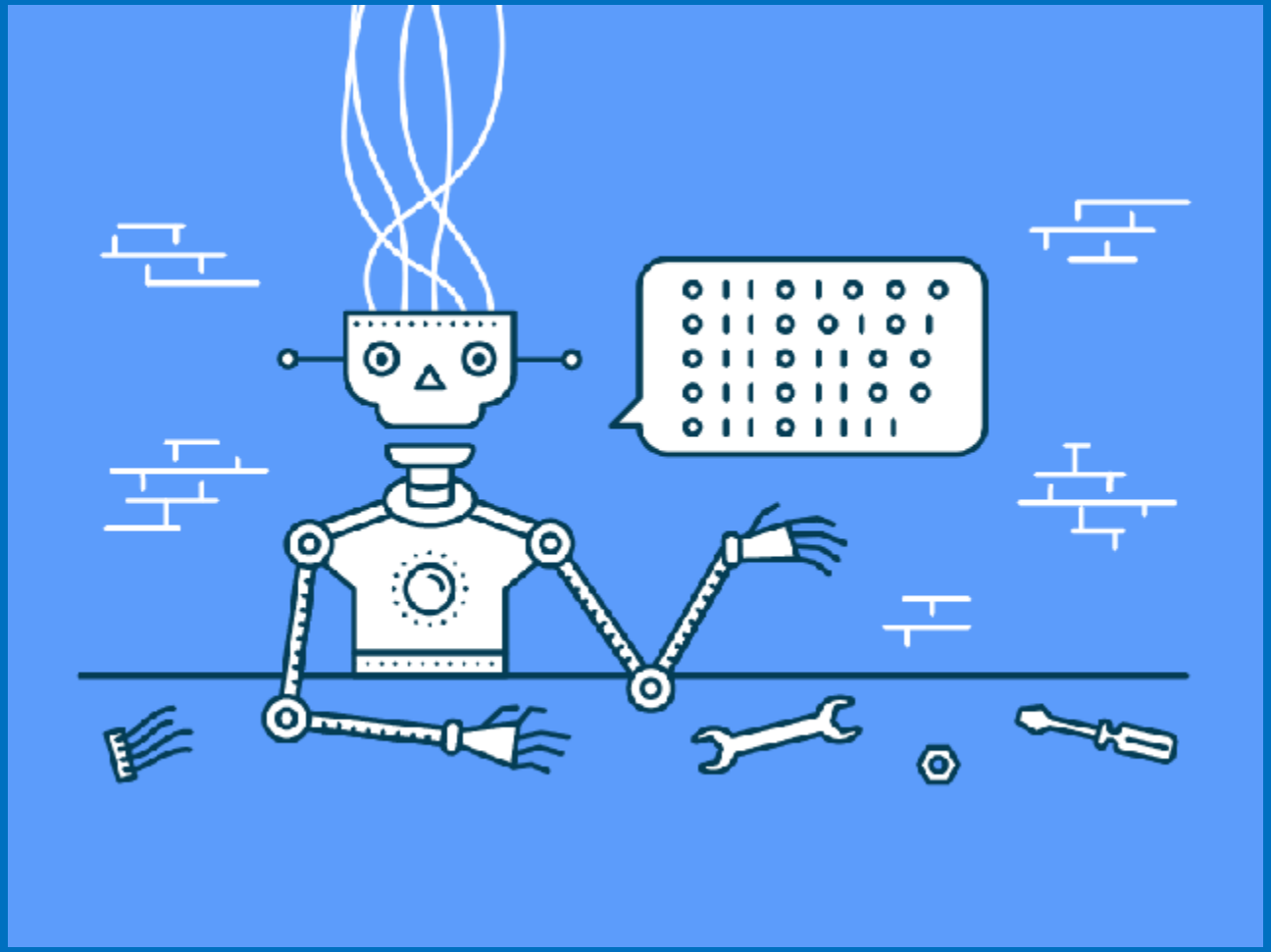
- optimalizáció előtt mindig törekedjünk az algoritmikus optimalizációra
- ha az algoritmusunk optimális, akkor mérjünk!
  - mérés legyen reprodukálható!
  - mérési adatok reprezentatívak!
- mindig a nem kielégítő teljesítmény 80 %-át adó részt optimalizáljuk (Pareto-elv)
- ellenőrizzük az optimalizáció eredményt kontroll mérésekkel
- PGO-t (profile-guided optimization) csak a legvégső esetben és csak nagyteljesítményű programoknál használjunk

Profile-Guided Optimization ciklus



Forrás: https://source.android.com/devices/tech/dalvik/jit-compiler

ex_2: HF1 optimalizálás



Köszönöm a figyelmet!

Folytatjuk…