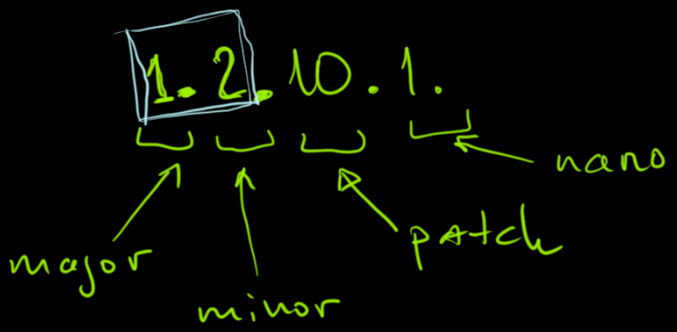
```java
/** version 1.0 */
public int add(int a, int b) {
  return a + b;
}
```

1.1 / 2.0 ?

```java
/** version 1.1 */
public int sub(int a, int b) {
  return a - b;
}
```



crucial details, new functionality

1.2.

```java
public double sqrt(double a) {
  return Math.sqrt(a);
}
```

```java
/** version 1.2 */
public double sqrt(double a) {
  return Math.sqrt(a);
}
```

? can it break ? → NO

```java
public double sqrt(double a) {
  if (a < 0) throw new IllegalArgumentException("not allowed");
  return Math.sqrt(a);
}
```

1) – we didn't change

2) double ⇒ double

double ⇒ double | Exception

```java
/** version 1.2.1 */
```

```java
/** version 2.0 */
public double sqrt(double a) throws IllegalArgumentException {
  if (a < 0) throw new IllegalArgumentException("not allowed");
  return Math.sqrt(a);
}
```

? can it break ? → YES!

```
x = sqrt(...)
```
it never brakes

```
try
  x = sqrt(...)
catch
```
it can brake

2.1 ~~3.0~~ – because it compatible w/ previous code base

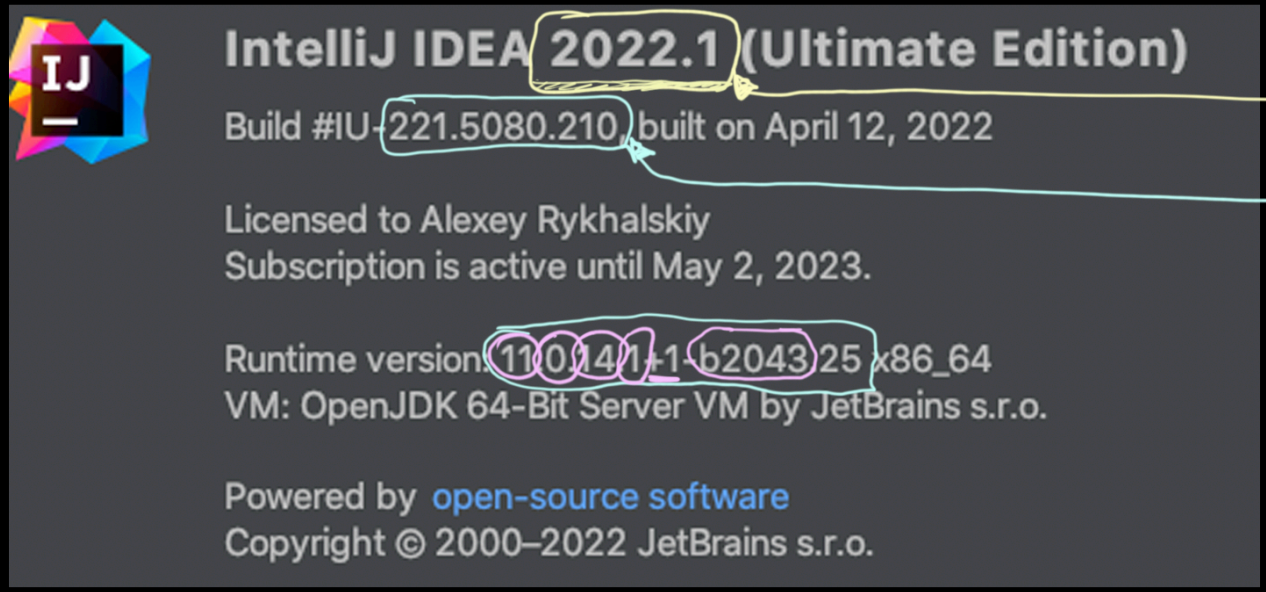
```
public double[] squareEq(double a, double b, double c) {
  throw new IllegalArgumentException("not implemented yet");
}
```

versions – is not about functionality
versions – is about compatibility

from development perspective

versions – is about functionality

from business / marketing perspective



version from business perspective

version from developer perspective

1.1. having that versioning can we exactly pick the code with a mistake? NO

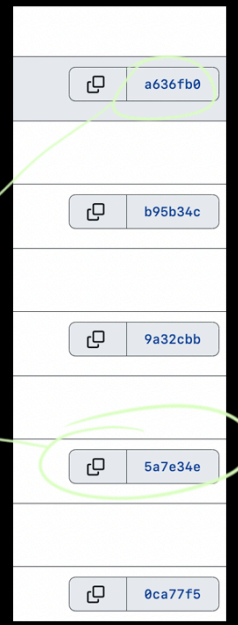
1.1 - a636fb0

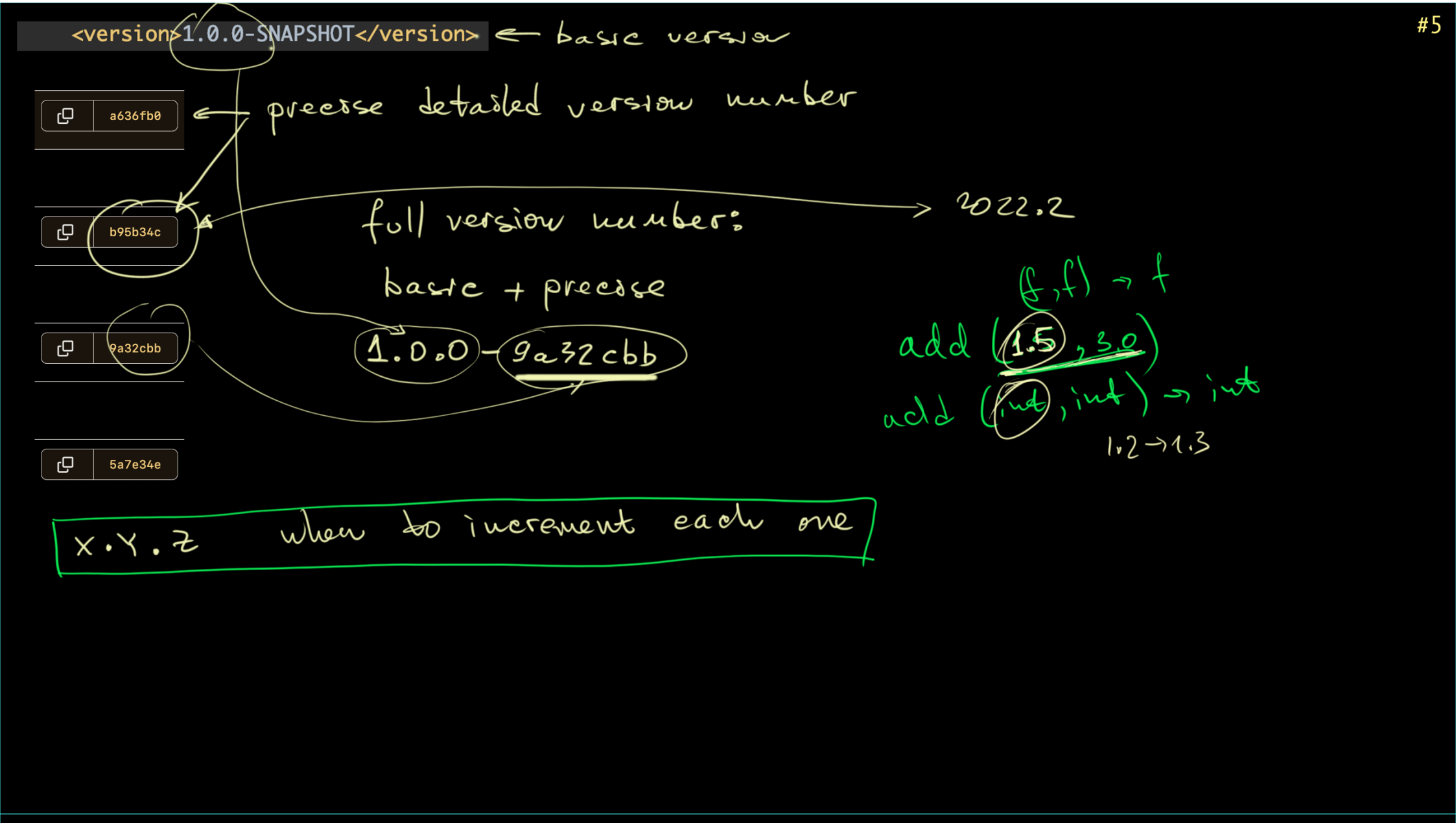
1.1 - 5a7e34e

that's better.

1.1 - abc123

1.2




<version>1.0.0-SNAPSHOT</version>
basic version
precise detailed version number
a636fb0
b95b34c
9a32cbb
5a7e34e
full version number:
basic + precise
1.0.0 - 9a32cbb
2022.2
(f,f) → f
add (1.5, 3.0)
add (int, int) → int
1.2 → 1.3
X.Y.Z when to increment each one

X.Y.Z
major | minor | patch

1) you added functionality `add(int x, int y) -> int` → X.Y+1.0
2) you added a class to represent `User(name:String)` → X.Y+1.0
3) we modified functionality `add(double x, double x) -> double` → X+1.0.0
4) we modified `User(name:String, age:int)` → X+1.0.0
5) we added `add(x,y,z) -> int` → X.Y+1.0
6) we deleted `add(_,_)`
   deleted `add(_,_,_)`
   added `add(int, int...)`
   → X+1.0.0

new User("Jim") ?

we modified `User(name:String, age:int, skills...String)` → X+1.0.0

User("Jim", 33, "Java", "Python")
(Jim, 33, "Java")
(Jim, 33)

7) doSomething(a: A) → doSomething(a: B) X+1.0.0

we added functionality
- new function
- new class
- all previous code keeps compiling

X.Y+1.0

we added/modified functionality
- new parameter to function/method added
- new public/required field for class added
- type in function/method/pub. field changed
- at least one line from prev. code stops working

X+1.0.0

- when we modified/fixed some things inside
- didn't make a change to the public part
- all the code keeps compiling

X.Y.Z+1

it's better to have than

1.3 - abc123f useful

1.3.0
1.5.0 — useless

semantic versioning

73.5
27.10
} careful useful examples

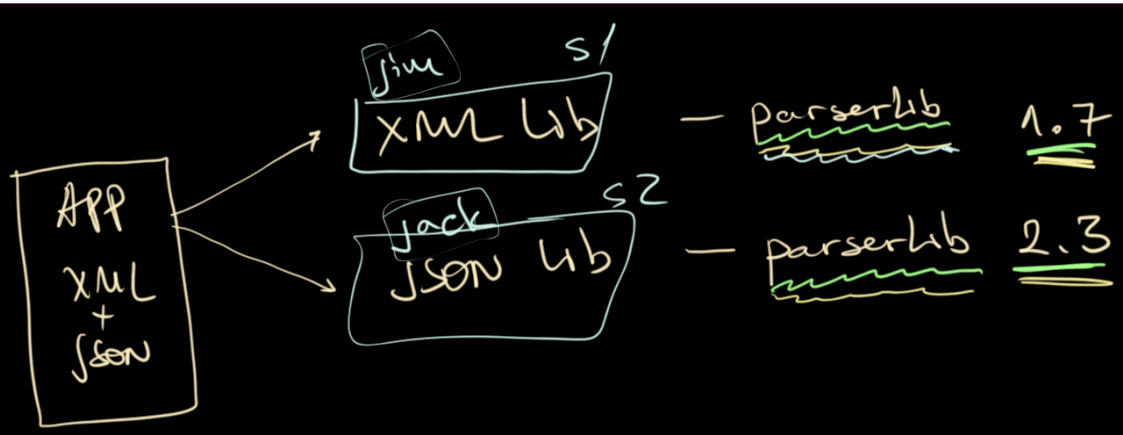
```
% "1.1.129-SNAPSHOT"
val ssmUserMgrV = "1.1.21-SNAPSHOT"
val decideUIServerV = "1.0.7-SNAPSHOT"
val decideAnalysisServerV = "1.0.6-SNAPSHOT"
val conversationMainV = "1.2"
val evidenceV = "0.1.12-SNAPSHOT"
val kdmDiscoveryV = "1.0.4-SNAPSHOT"
val userAccessMgrV = "1.0.2"
val userMonitoringMgrV = "1.0.2"
val kmmOntologyMgrV = "1.0.7"
val cmmRuleBaseV = "1.0.5"
val himLibV = "1.2.15-SNAPSHOT"
```

useless

```
val commonVersion = "13.4"
val commonIdsVersion = "1.2"
val sourcesVersion = "21.1"
val accountsVersion = "22.0"
val shardCartographerVersion = "9.1"
val facilitiesVersion = "25.0"
val laborVersion = "37.0"
val inventoryVersion = "27.0"
val ordersVersion = "54.0"
val notificationsVersion = "59.0"
```

21.0... 21.111

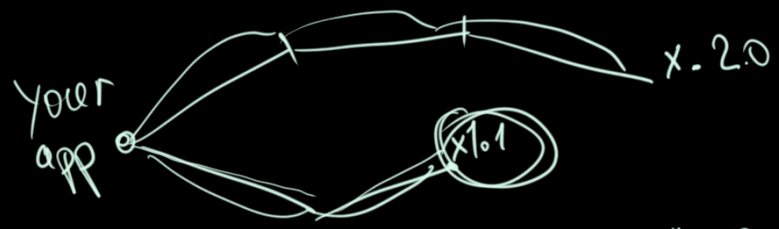
useful



app.jar
xml.jar
json.jar
parser.jar → 1.7
parser.jar → 2.3

can maven decide ? → NO

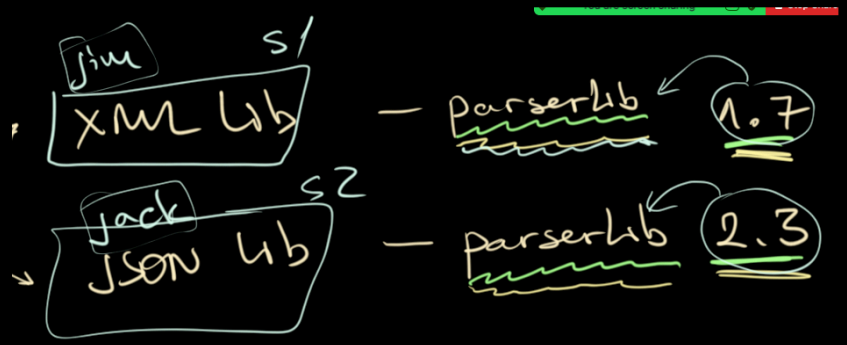
can you decide ? → NO



lenght to x1.1 = 2 ← maven takes the shortest path

length to x2.0 = 3

shading jar



1. detects incompatibilities / specify manually
2. renames / creates new jar

parser 1.7 — ParseText
parser17 1.0 — ParseText

parser 2.3
parser23 1.0

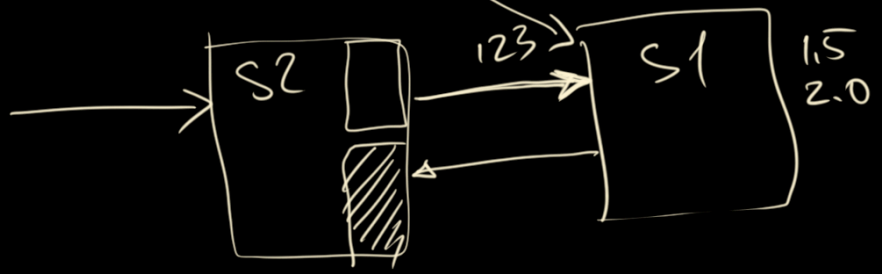
3. it rewires all usages : from parser 1.7 → parser17 in all dep. libs.

API of microservices

/v1/getUser/123 → UserJson { name age }

/v2/getUser/123 → UserJson { firstName lastName age id skills }

=> we need to recompile S2 to consume new format



when all services migrated to use /v2/....

we remove /v1/....

from codebase perspective

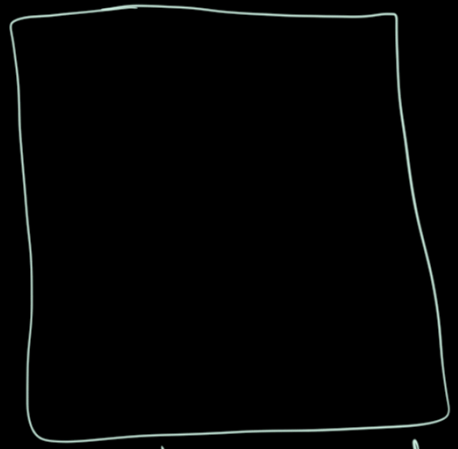
1.5 { name age }

2.0 { firstName lastName age }

1.3 { name age }

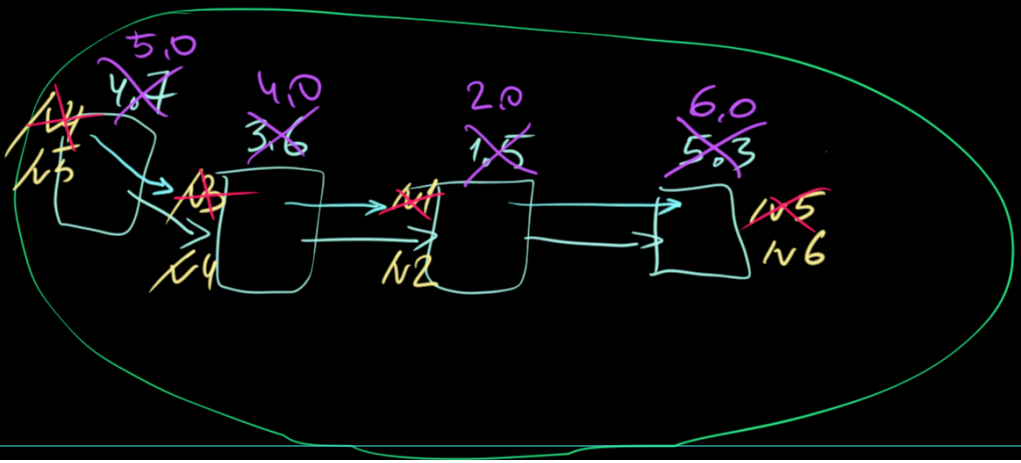
1.4 { name age skills }

Monolith

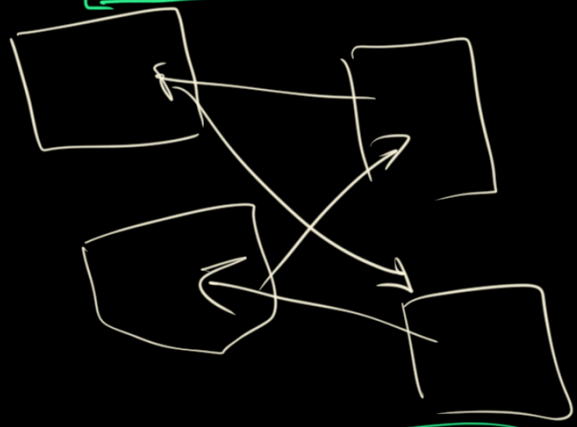


nobody cares about versioning

- you always recompile everything
- you always deploy latest



Microservices



- Code reusability (jar artifacts versioning)
- Data Flow (API HTTP versioning)

/v1/user/123

- you deploy 1 by 1
- you compile parts
- you always reuse
- you must care about version