



Introduction to Scientific Computation

Lecture 0

Fall 2023

The rules of the game
Intro

General course info

Lectures can be downloaded and viewed on Github:

<https://github.com/Aelphy/ISC>

There you can find information on how to prepare the environment, launch the code, submit assignments, communicate with staff etc.

Team & TAs:

Mikhail Usvyatsov

Communication system

Communication with staff (questions, suggestions, etc) could be done via emails

m.usvyatsov+ethz@gmail.com. Please make sure that the subject in your email is [ISC2023]

Telegram chatroom for students is here: <https://t.me/iscfall23>

Plagiarism

Students are encouraged to discuss the material, but all the assignments should be done individually. Assignments will be checked for plagiarism. If noticed, can be punished in various ways from zeroing out some parts of a specific assignment to unsatisfactory grade for the course.

Assignments

This course uses Github for submitting assignments. You should create a dedicated **private!** repository with name **ISC2023-solutions**. E.g. github.com/YOUR_NAME/ISC2023-solutions. And add me (@Aelphy) to collaborators.

Assignment becomes submitted after sending email to m.usvyatsov+ethz@gmail.com with the subject [ISC2023: assignment_N], where N is the number of assignment. In the body of this email there should be a link to a dedicated folder for this particular assignment. For the first assignment it would be:

github.com/YOUR_NAME/ISC2023-solutions/assignment_0

LLMs

Various AI systems became way too advanced as of the Fall 2023.
When extensive similarity with AI generated content is suspected, the student
pretending for the authorships will face a defence procedure: detailed questions to
test how well did the student understand the underlying matters.

Deadlines & Attendance

Unless stated otherwise, submission deadline is 00:01 of 10's day after handing out. You lose 10% points per day until submission (from the full amount of points on the previous day).

By the time of grading the assignment will be checked in the state of the last commit before the deadline. All later commits are subject to late submission policy.

Assignment can't yield negative amount of points, even if submitted 2 years past deadline.

Attendance is not strict, but do not disappoint us.

Grading

The course grade depends entirely on points obtained mostly by completing homework assignments. There will be no course exam at the end.

Some assignment examples are "Implement and optimise factorial computation algorithm", "Implement gradient descent algorithm for logistic regression", "Compare Ridge and Lasso regression for house prices problem", "Implement Perceptron Algorithm"

Materials

The Hitchhiker's Guide to Python

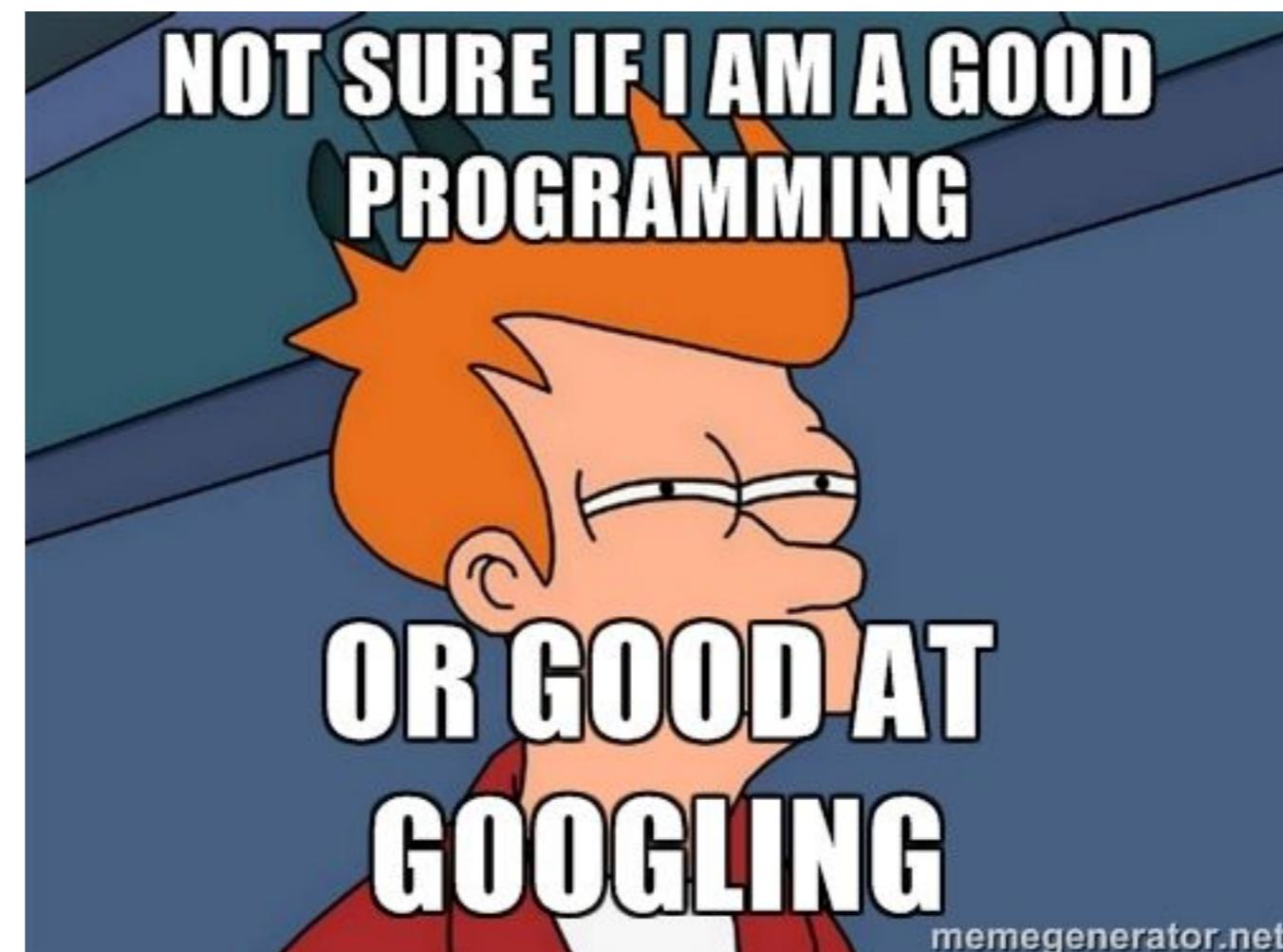
The Primer on Scientific Programming With Python. H. Langtangen

Always use

Google

StackOverflow

As of Fall 2023,
chatgpt, Bard, etc.,
might or might not be a good
resource for consulting. Mind
The references.



Introduction

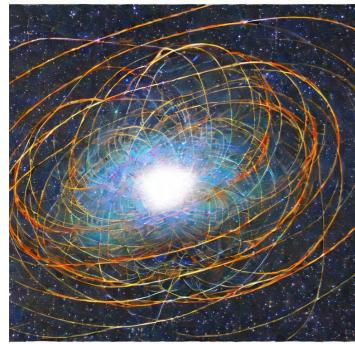
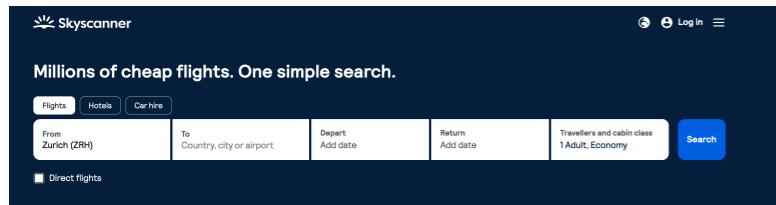
How is your code running ?



Semantic gaps

Domain applications

- Tickets search platform
 - Airlines
 - Routes
 - Prices
- Modelling celestial motion
 - Speed
 - Space
 - Mass
 - Coordinates
 - Time



Programming primitives

- Class
- Function
- Variable
- Data structures
 - Array
 - List

 A screenshot of a GitHub pull request titled "[xtensa] Fixed a bug in interleave RGB for Q8 #7526". The code changes are shown in a diff view, specifically for file `src/CodeGen_Xtensa_vectors.template.cpp`. The changes involve fixing sequences of instructions and handling memory interleaving for Q8 format.

```

diff --git a/src/CodeGen_Xtensa_vectors.template.cpp b/src/CodeGen_Xtensa_vectors.template.cpp
--- a/src/CodeGen_Xtensa_vectors.template.cpp
+++ b/src/CodeGen_Xtensa_vectors.template.cpp
@@ -1313,6 +1313,7 @@ HALIDE_ALWAYS_INLINE native_vector<u16>_x3 halide_xtensa_interle
1314
1315 // This sequence of instructions is taken from the user guide.
1316 + // For Q8 the guide provides wrong q5 sequences.
1317 HALIDE_ALWAYS_INLINE native_vector<u16>_x3 halide_xtensa_interleave<const native_v
1318 1318 // 16-bit interleave patterns
1319 #if XHAL_VISION_TYPE == 3
1320
1321 // 16, 38, 37, 39, 194, 83, 33, 49, 19, 43, 195, 84, 29, 42, 23;
1322 // 74, 8, 22, 1, 23, 96, 76, 2, 24, 3, 25, 97, 76, 4, 26, 5,
1323 // 27, 98, 77, 6, 28, 7, 29, 99, 78, 8, 30, 9, 31, 198, 79, 19,
1324 // 32, 15, 33, 181, 48, 12, 34, 33, 35, 182, 9, 31, 14, 36, 35, 37, 183,
1325 // 82, 16, 38, 37, 39, 194, 83, 33, 49, 19, 43, 195, 84, 29, 42, 23;
1326 // 74, 8, 22, 68, 23, 96, 75, 2, 24, 67, 25, 97, 76, 4, 26, 69,
1327 // 32, 15, 33, 181, 48, 12, 34, 33, 35, 182, 9, 31, 14, 36, 35, 37, 183,
1328 // 82, 16, 38, 37, 39, 194, 83, 33, 49, 19, 43, 195, 84, 29, 42, 23;
1329 // _attribute_(aligned(XHAL_VISION SIMD0)) unsigned char int_16b_c3_step[1:28]
1330
1331 // 106, 43, 21, 85, 22, 44, 187, 45, 22, 86, 23, 46, 188, 47, 23, 87,
1332 // 24, 48, 189, 49, 24, 88, 28, 90, 118, 91, 25, 89, 26, 92, 111, 93,
1333 // 26, 98, 27, 94, 112, 95, 27, 91, 28, 96, 113, 97, 28, 92, 29, 98,
1334 // 117, 99, 29, 95, 108, 110, 98, 29, 93, 111, 99, 28, 94, 110, 99,
1335 // 33, 9, 111, 2, 93, 96, 2, 118, 97, 33, 97, 98, 117, 99,
1336 // 34, 98, 35, 9, 128, 7, 99, 99, 36, 8, 111, 9, 98, 100, 37, 18
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
16010
16011
16012
16013
16014
16015
16016
16017
16018
16019
16020
16021
16022
16023
16024
16025
16026
16027
16028
16029
16030
16031
16032
16033
16034
16035
16036
16037
16038
16039
16040
16041
16042
16043
16044
16045
16046
16047
16048
16049
16050
16051
16052
16053
16054
16055
16056
16057
16058
16059
16060
16061
16062
16063
16064
16065
16066
16067
16068
16069
160610
160611
160612
160613
160614
160615
160616
160617
160618
160619
160620
160621
160622
160623
160624
160625
160626
160627
160628
160629
160630
160631
160632
160633
160634
160635
160636
160637
160638
160639
160640
160641
160642
160643
160644
160645
160646
160647
160648
160649
160650
160651
160652
160653
160654
160655
160656
160657
160658
160659
160660
160661
160662
160663
160664
160665
160666
160667
160668
160669
160670
160671
160672
160673
160674
160675
160676
160677
160678
160679
160680
160681
160682
160683
160684
160685
160686
160687
160688
160689
160690
160691
160692
160693
160694
160695
160696
160697
160698
160699
1606100
1606101
1606102
1606103
1606104
1606105
1606106
1606107
1606108
1606109
1606110
1606111
1606112
1606113
1606114
1606115
1606116
1606117
1606118
1606119
16061100
16061101
16061102
16061103
16061104
16061105
16061106
16061107
16061108
16061109
16061110
16061111
16061112
16061113
16061114
16061115
16061116
16061117
16061118
16061119
160611100
160611101
160611102
160611103
160611104
160611105
160611106
160611107
160611108
160611109
160611110
160611111
160611112
160611113
160611114
160611115
160611116
160611117
160611118
160611119
1606111100
1606111101
1606111102
1606111103
1606111104
1606111105
1606111106
1606111107
1606111108
1606111109
1606111110
1606111111
1606111112
1606111113
1606111114
1606111115
1606111116
1606111117
1606111118
1606111119
16061111100
16061111101
16061111102
16061111103
16061111104
16061111105
16061111106
16061111107
16061111108
16061111109
16061111110
16061111111
16061111112
16061111113
16061111114
16061111115
16061111116
16061111117
16061111118
16061111119
160611111100
160611111101
160611111102
160611111103
160611111104
160611111105
160611111106
160611111107
160611111108
160611111109
160611111110
160611111111
160611111112
160611111113
160611111114
160611111115
160611111116
160611111117
160611111118
160611111119
1606111111100
1606111111101
1606111111102
1606111111103
1606111111104
1606111111105
1606111111106
1606111111107
1606111111108
1606111111109
1606111111110
1606111111111
1606111111112
1606111111113
1606111111114
1606111111115
1606111111116
1606111111117
1606111111118
1606111111119
16061111111100
16061111111101
16061111111102
16061111111103
16061111111104
16061111111105
16061111111106
16061111111107
16061111111108
16061111111109
16061111111110
16061111111111
16061111111112
16061111111113
16061111111114
16061111111115
16061111111116
16061111111117
16061111111118
16061111111119
160611111111100
160611111111101
160611111111102
160611111111103
160611111111104
160611111111105
160611111111106
160611111111107
160611111111108
160611111111109
160611111111110
160611111111111
160611111111112
160611111111113
160611111111114
160611111111115
160611111111116
160611111111117
160611111111118
160611111111119
1606111111111100
1606111111111101
1606111111111102
1606111111111103
1606111111111104
1606111111111105
1606111111111106
1606111111111107
1606111111111108
1606111111111109
1606111111111110
1606111111111111
1606111111111112
1606111111111113
1606111111111114
1606111111111115
1606111111111116
1606111111111117
1606111111111118
1606111111111119
16061111111111100
16061111111111101
16061111111111102
16061111111111103
16061111111111104
16061111111111105
16061111111111106
16061111111111107
16061111111111108
16061111111111109
16061111111111110
16061111111111111
16061111111111112
16061111111111113
16061111111111114
16061111111111115
16061111111111116
16061111111111117
16061111111111118
16061111111111119
160611111111111100
160611111111111101
160611111111111102
160611111111111103
160611111111111104
160611111111111105
160611111111111106
160611111111111107
160611111111111108
160611111111111109
160611111111111110
160611111111111111
160611111111111112
160611111111111113
160611111111111114
160611111111111115
160611111111111116
160611111111111117
160611111111111118
160611111111111119
1606111111111111100
1606111111111111101
1606111111111111102
1606111111111111103
1606111111111111104
1606111111111111105
1606111111111111106
1606111111111111107
1606111111111111108
1606111111111111109
1606111111111111110
1606111111111111111
1606111111111111112
1606111111111111113
1606111111111111114
1606111111111111115
1606111111111111116
1606111111111111117
1606111111111111118
1606111111111111119
16061111111111111100
16061111111111111101
16061111111111111102
16061111111111111103
16061111111111111104
16061111111111111105
16061111111111111106
16061111111111111107
16061111111111111108
16061111111111111109
16061111111111111110
16061111111111111111
16061111111111111112
16061111111111111113
16061111111111111114
16061111111111111115
16061111111111111116
16061111111111111117
16061111111111111118
16061111111111111119
160611111111111111100
160611111111111111101
160611111111111111102
160611111111111111103
160611111111111111104
160611111111111111105
160611111111111111106
160611111111111111107
160611111111111111108
160611111111111111109
1606111111111111
```

Semantic gaps

Programming primitives

- Class
- Function
- Variable
- Data structures
 - Array
 - List



Hardware

- CPU
 - ARM
 - X86
 - RISC-V
 - Instruction set
- GPU
- Bytes
- Memory
- Registers
- Cache
- RAM
- Bus

So, the main compiler task is to map human-readable code to hardware to solve the task efficiently

(c) The Demon Seated. Mikhail Vrubel

Machine Code & Assembly Language

Off-topic

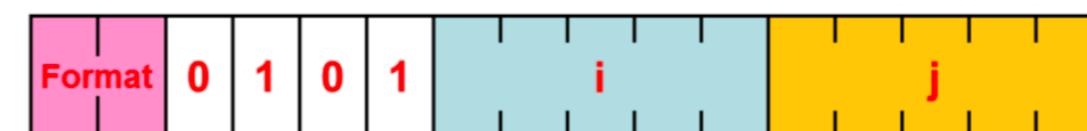
Mnemonic notation:

ADD 5 7



ADD i j

- The ADD instruction denotes the two's complement arithmetic addition. The contents of registers Ri and Rj are arithmetically added, and the result is put into the register Rj.
- The instruction format is as follows:



Binary code of the instruction:

1101010010100111



Hexadecimal code of the instruction:

D4A7

Assembler code:

.format 32



R5 += R7

Suggested assembly statement for the ADD instruction:

Rj += Ri;

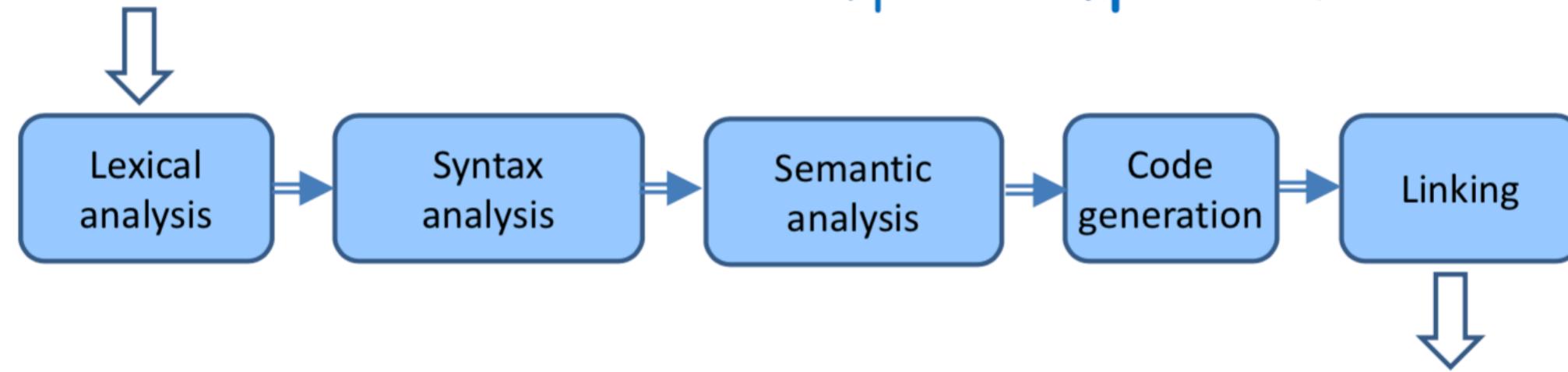
Additional assembly directives specifying the current instruction format:

.format 8; or .format 16; or .format 32;

Compilation: An Ideal Picture

*A program written by a human
(or by another program)*

Source program
text



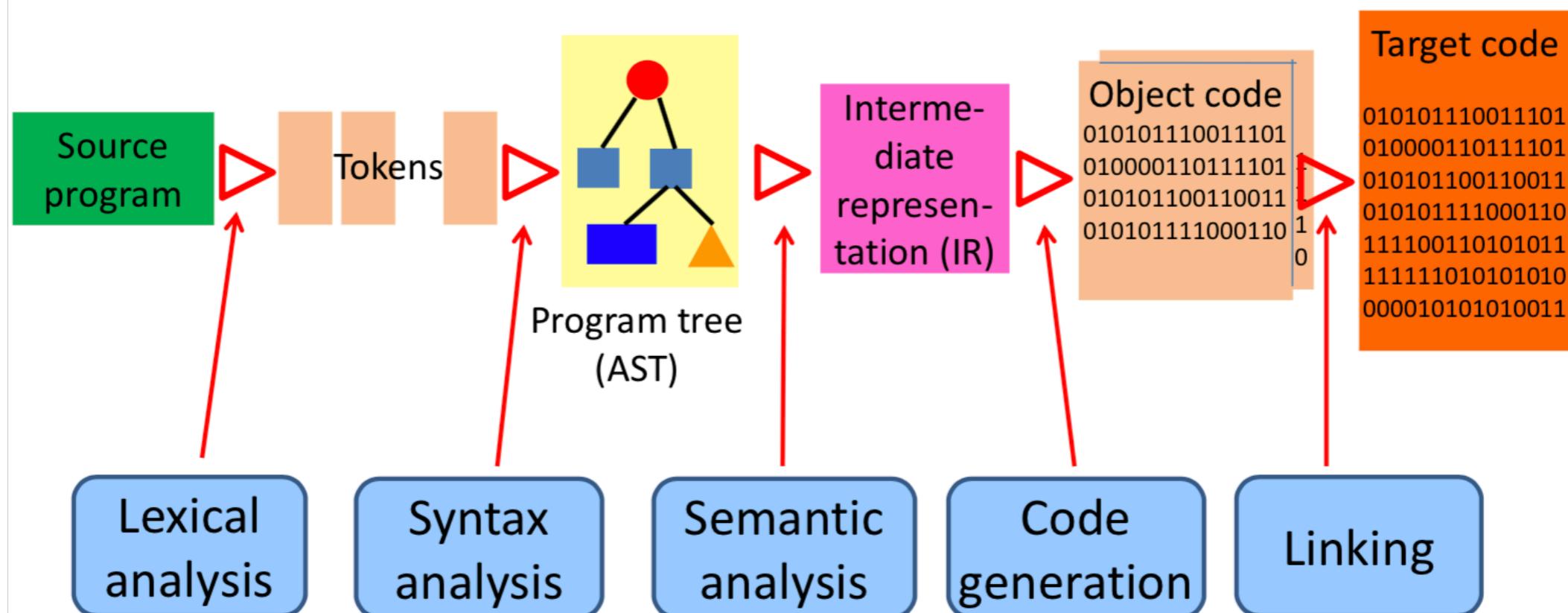
Blue squares just denote some actions typical to any compiler;
they are not necessarily actual compiler components.

Target code
01010111001110101000
01101111010101011001
10011010101111000110

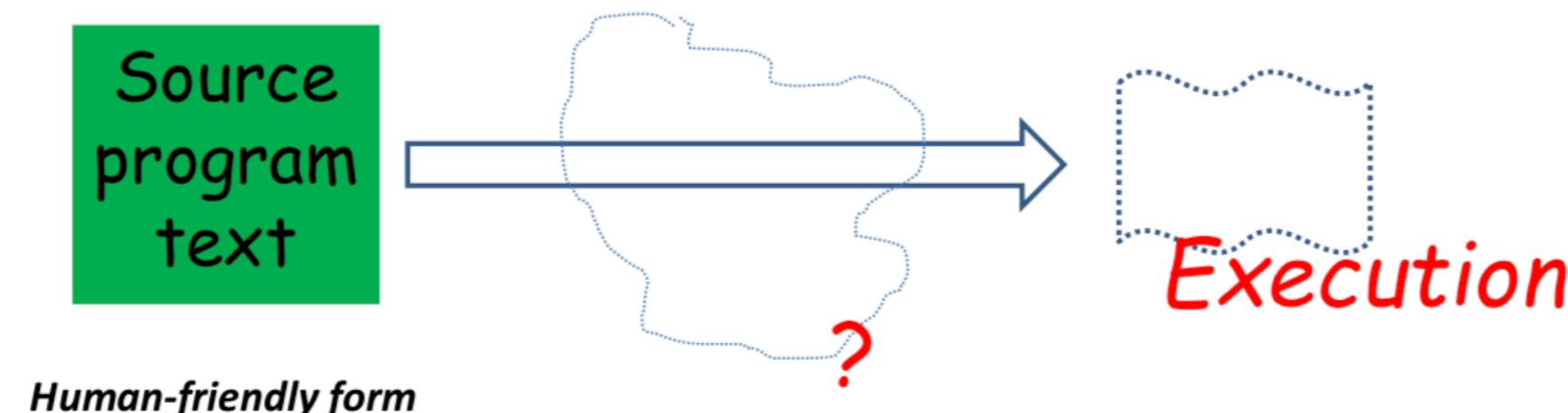
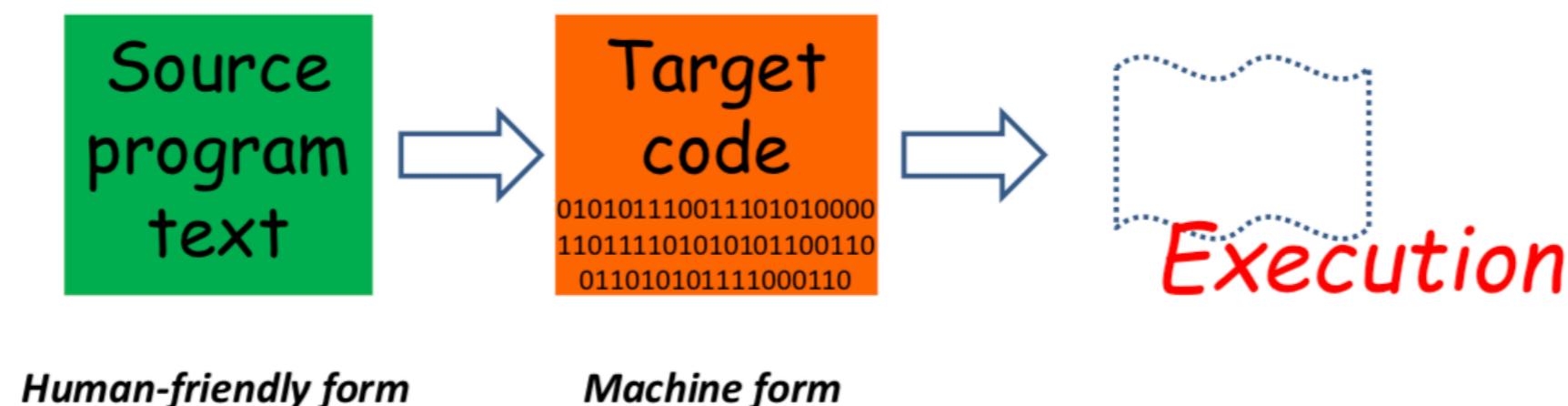
*A program binary image
suitable for immediate
execution by a machine*

Compilation Data Structures

This is a different, **data-centric** view at the compilation process. Compilation is a sequence of transformations of a source program.



Compilation vs Interpretation



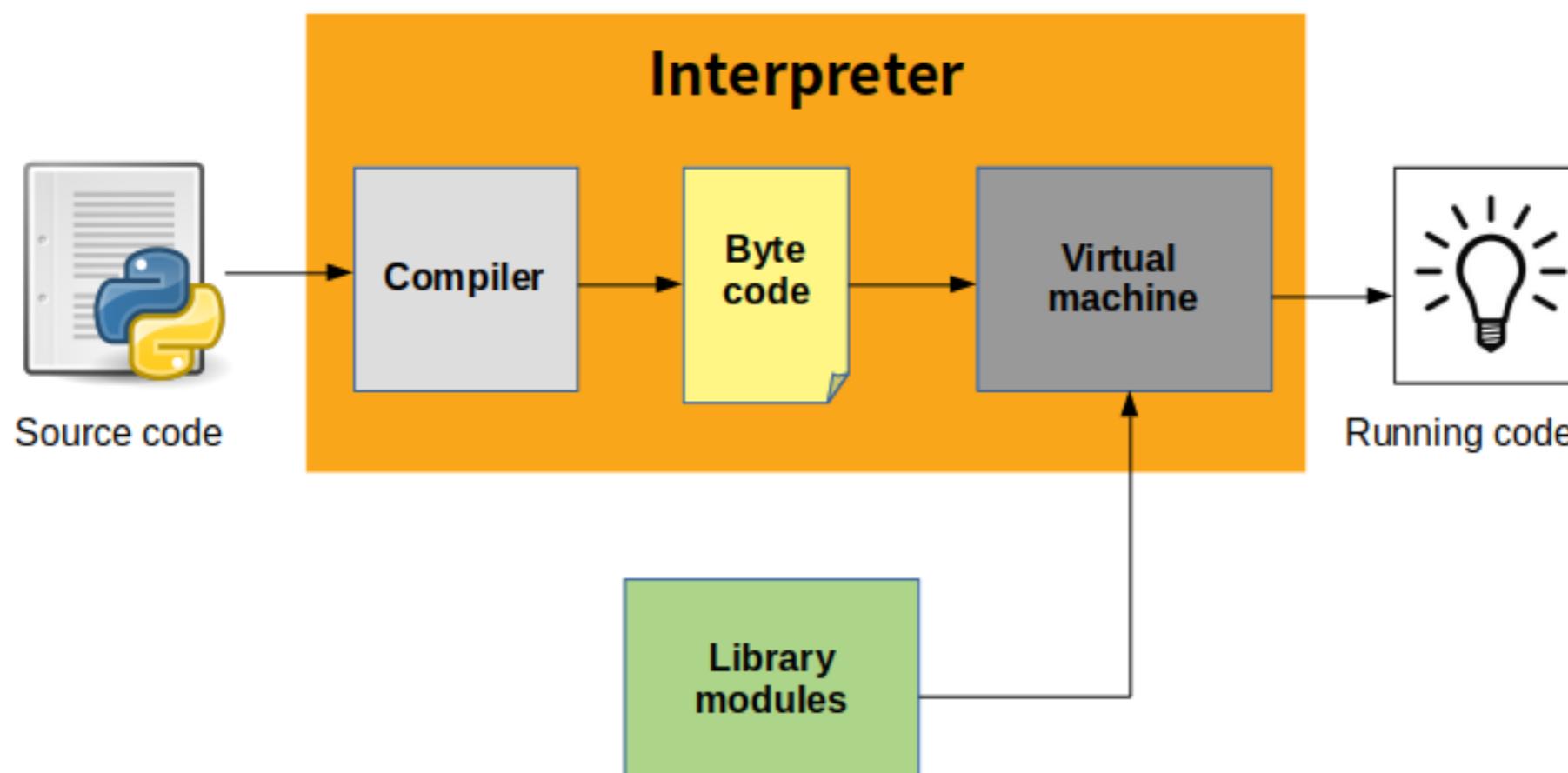
(c) Copyright 2017, E.Zouev

So, what about Python ?

From a programmer point of view



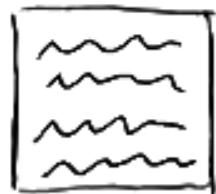
From Python point of view



So, what about Python ?

Source code:

hello.c



→ COMPILER →

Machine code:

11010
1011
10001

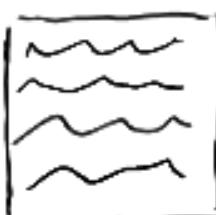
Program (also
called binary,
executable ...)

run the
program
result



Source code:

hello.py



→ INTERPRETER → result



How does Python interpret this:

$$X + Y * Z + U$$

How does Python interpret this:

$$X + Y * Z + U$$

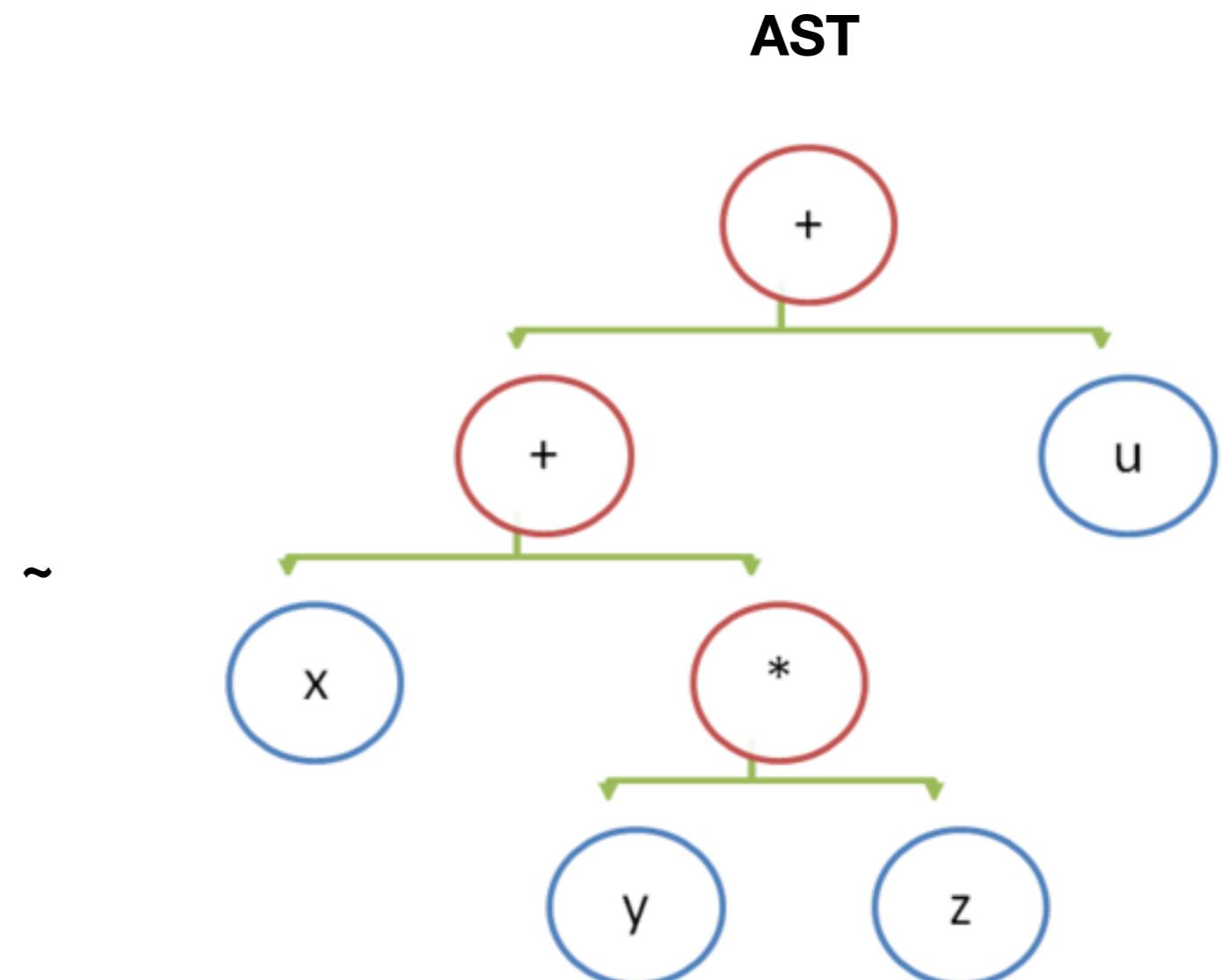

Stack

```
push x  
push y  
push z  
multiply  
add  
push u  
add
```

How does Python interpret this:

Stack

```
push x  
push y  
push z  
multiply  
add  
push u  
add
```



Python

Python

Pros:

- **Extensive Support Libraries**
- **Can use C/C++ libs via pybind11**
- **Cross platform**
- **Relatively low entry level**
- **Free & open-sourced**
- **Huge community**

Cons:

- **Slower than C/C++, assembly**
- **Runtime errors**
- **Parallel(GIL)**

Python environments

pip - Pip Installs Packages - package manager for python. Can be used together with conda but very carefully

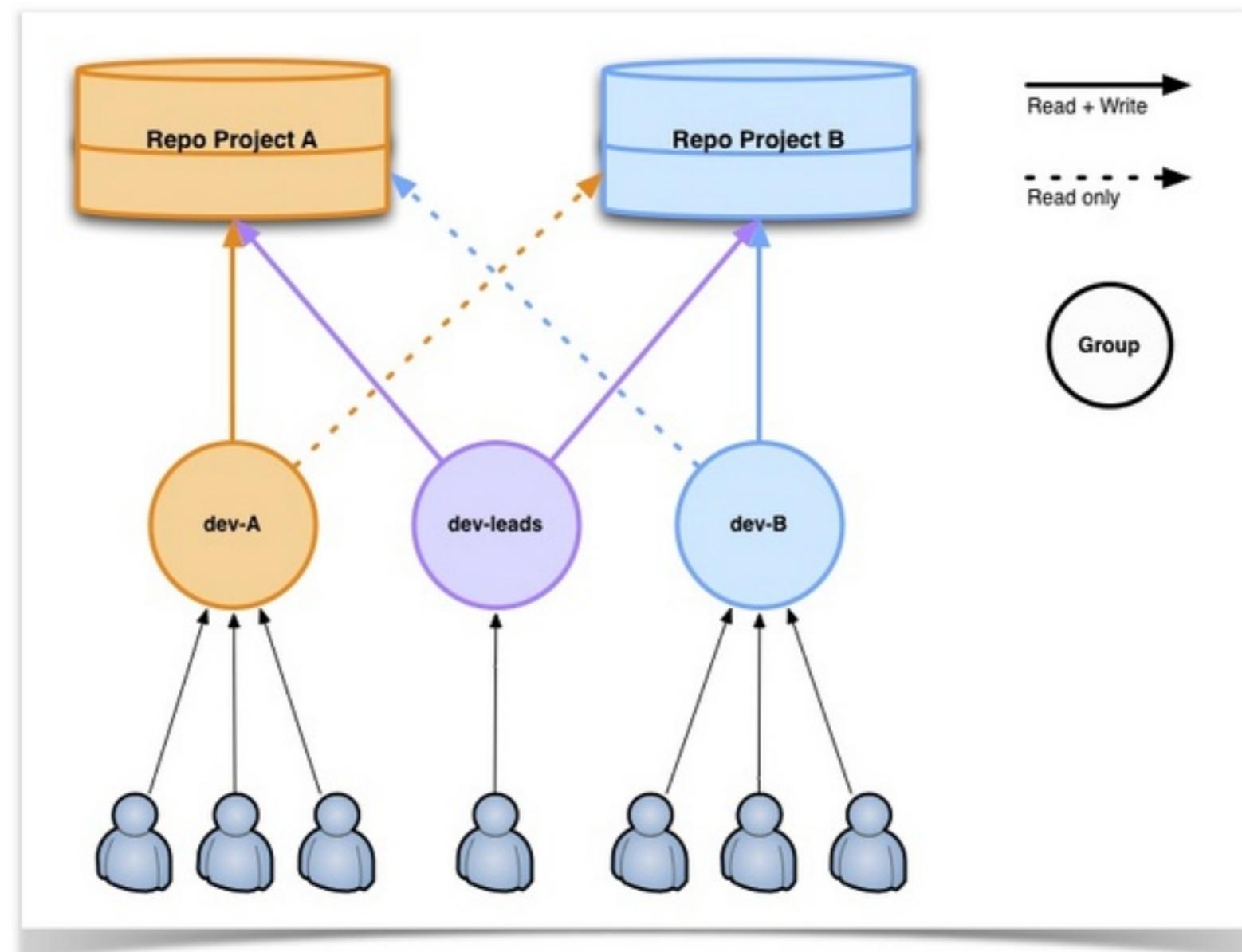
If you use python without anaconda:

- virtualenv, source activate, pip install

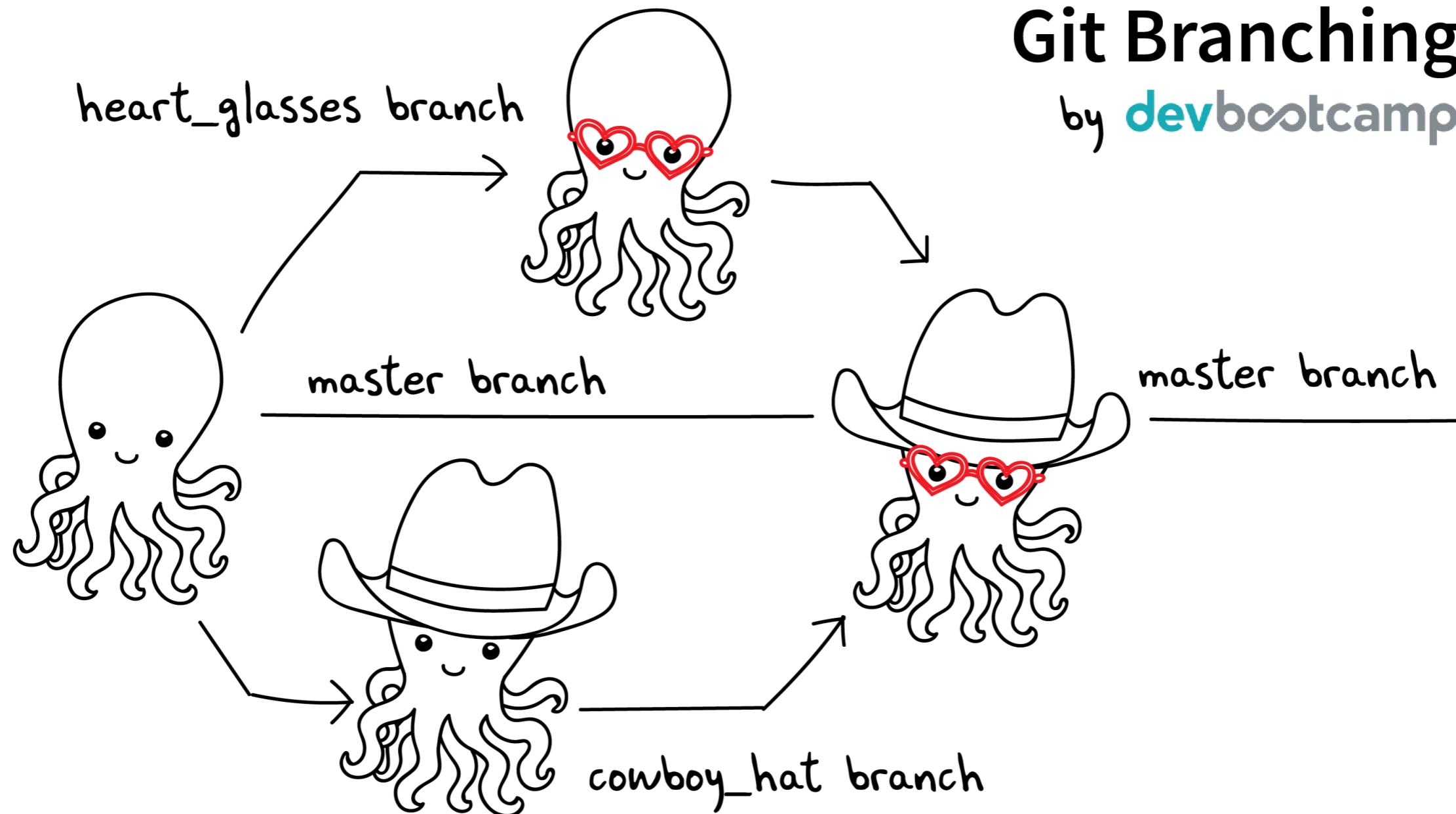
If you use anaconda:

- conda activate, conda install + pip install

Git: repositories



Git: branches



Let's code

