RISC-V & HPC

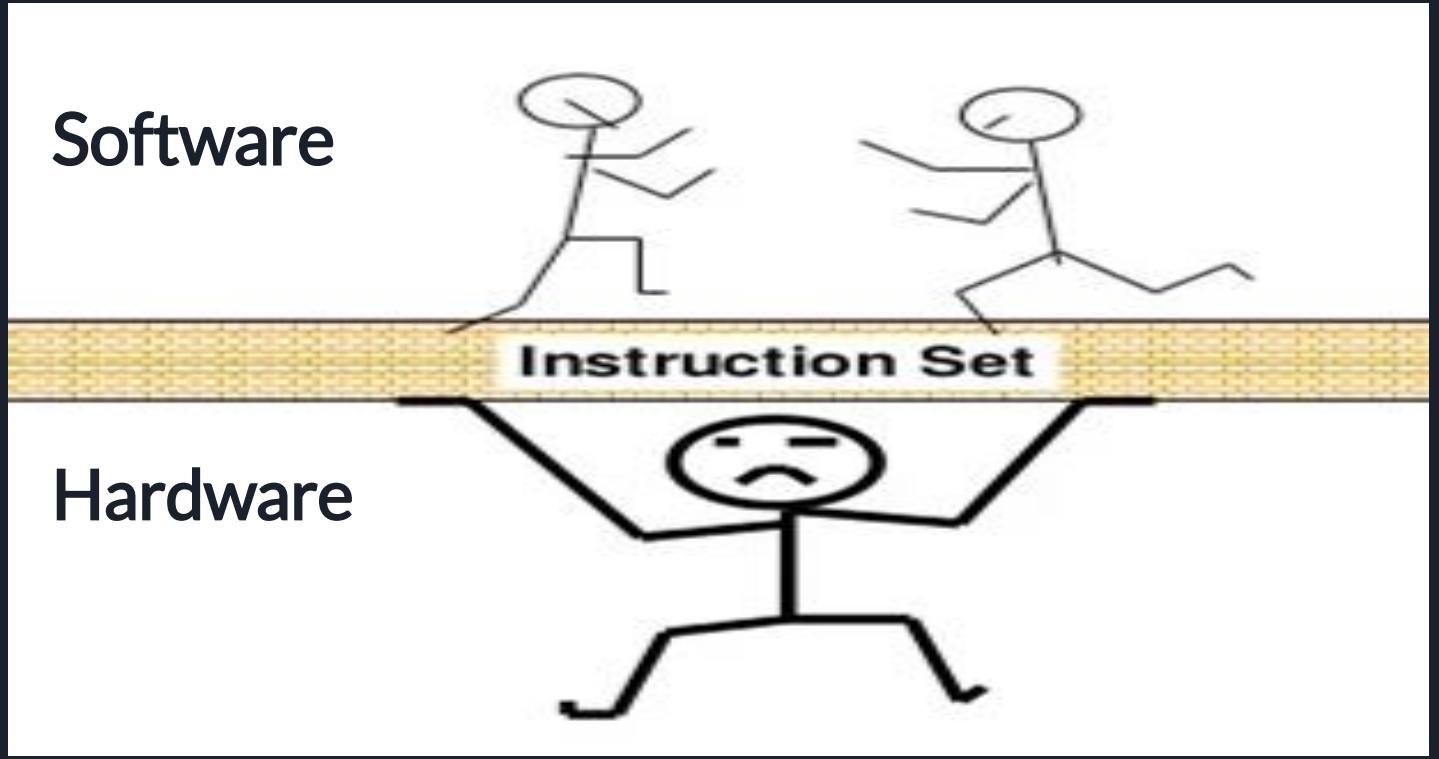
Guided by :

Mr. Surendra Billa Sir
Mr. Rushikesh Jadhav Sir

Project P5:

Mr. Tejas Bhanudas Katkar	PRN-230940130035
Mr. Bhanu Prakash Agrawal	PRN-230940130015
Ms. Muskan Sharma	PRN-230940130037
Ms. Akanksha Abhay Ghorpade	PRN-230940130025

Instruction Set Architecture




प्रगत संगणन विकास केंद्र
सी-डैक
CDAC
॥ज्ञानादेव तु कैवल्यम्॥
CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

Why RISC-V ?

ARM vs RISC V : ISA Comparison

ISAs can be broadly categorized into two types:

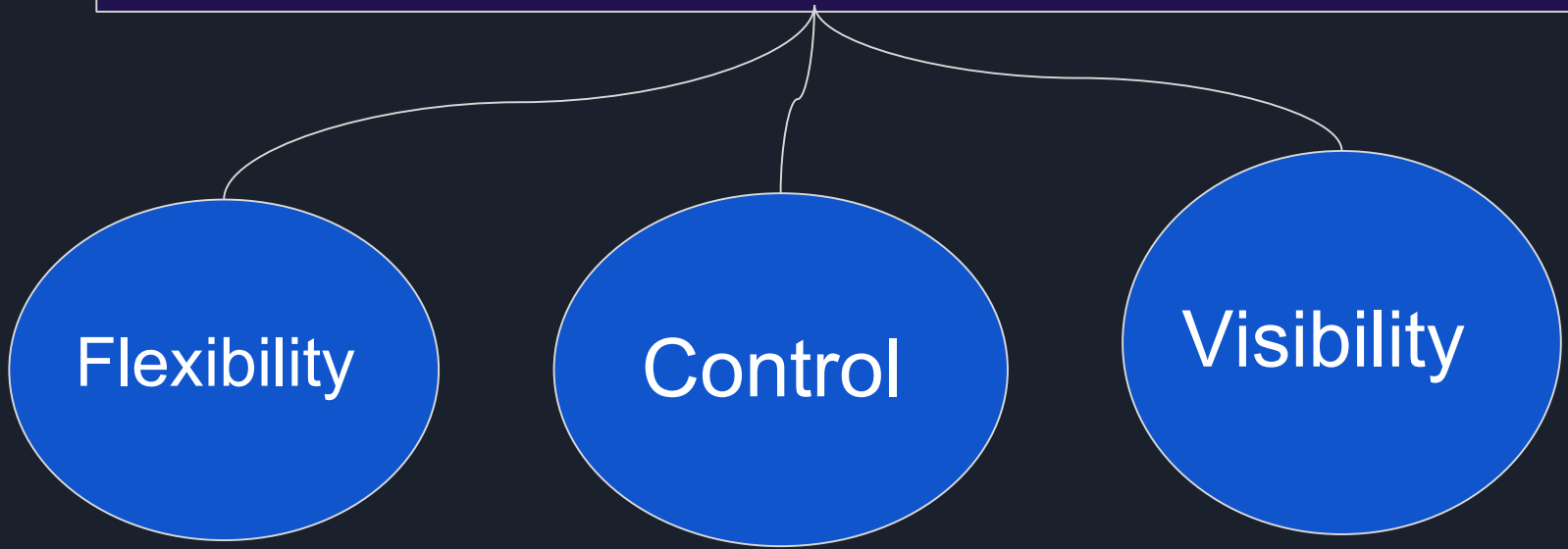
1→Open ISA

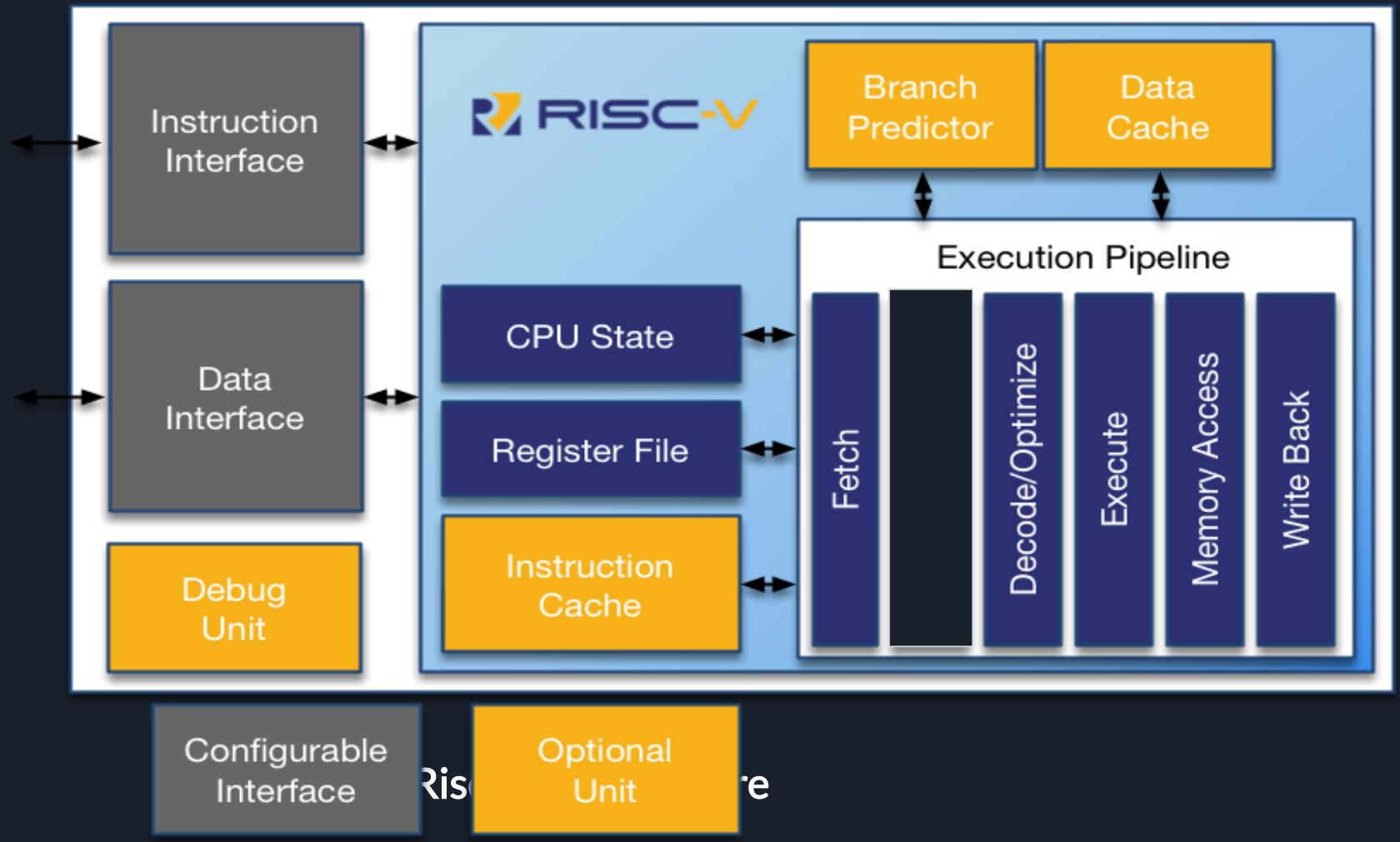
2→Closed ISA

- Closed ISAs, like ARM, are proprietary and tightly controlled by specific companies (Arm Holdings here), offering established reliability and compatibility but limiting customization.

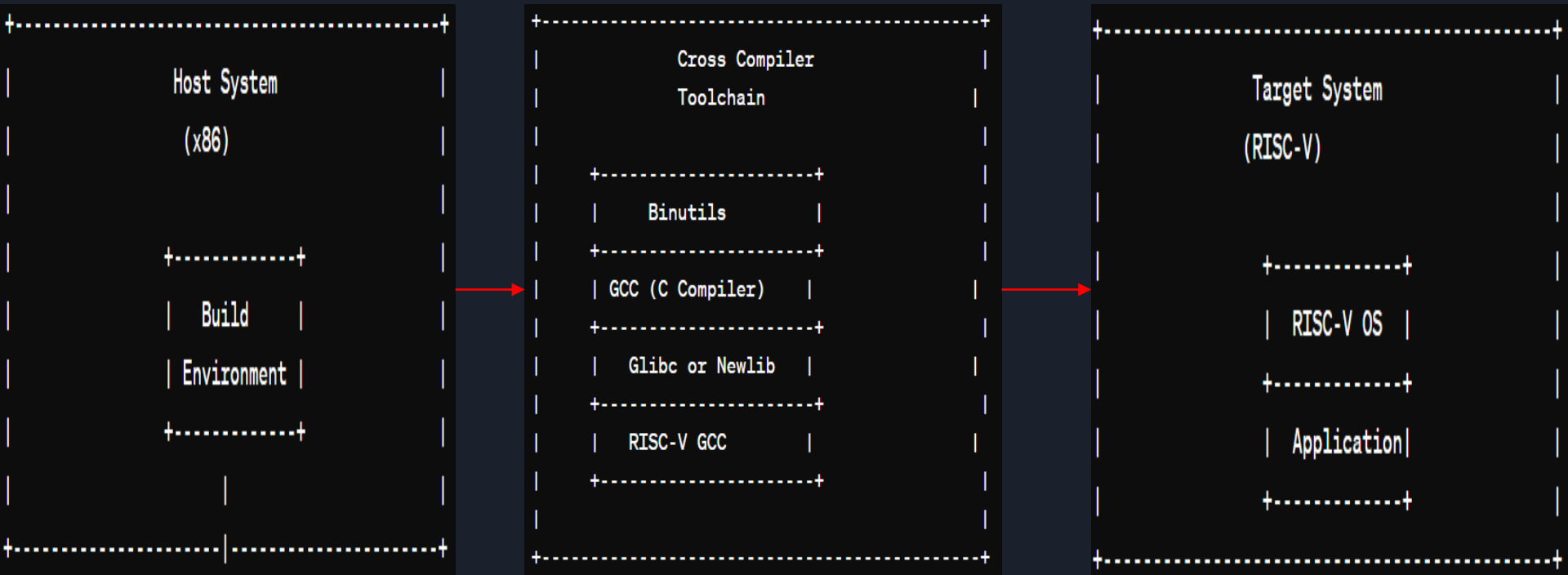
- Open ISAs, exemplified by RISC-V, are community-driven and provide greater flexibility for customization, fostering innovation and adaptation to specific needs.

3 main advantages of RISC-V

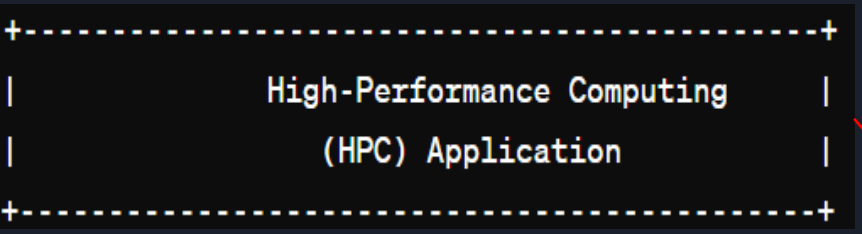


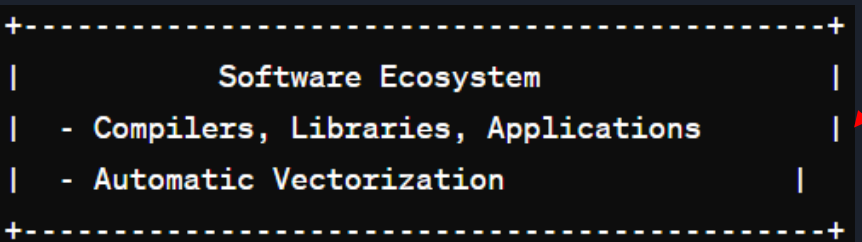

Instruction Interface
Data Interface
Debug Unit
RISC-V
Branch Predictor
Data Cache
CPU State
Register File
Instruction Cache
Execution Pipeline
Fetch
Decode/Optimize
Execute
Memory Access
Write Back
Configurable Interface
Optional Unit

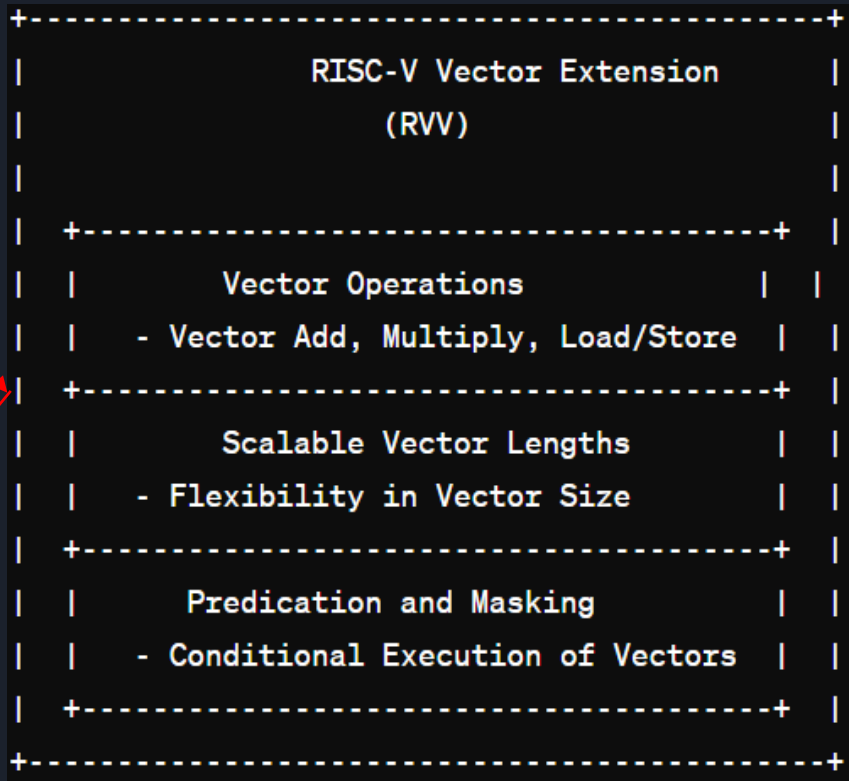
Cross Compiler Toolchain (Flow)



Vector Processing in HPC








CAM
CA/MEME FORUM


CALIFORNIA
SCHOOL


CAMBRIDGE
GROUP OF INSTITUTIONS


CAMEL


CATIE


cea


सी डैक
CDAC


CHINA UNIVERSITY OF PETROLEUM

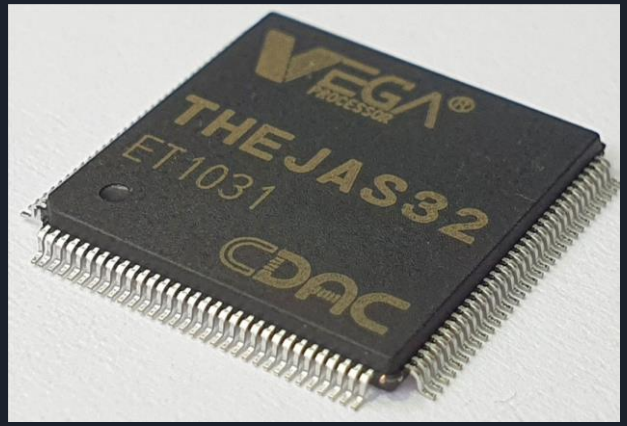

CHIPS
ALLIANCE


2017


CINECA


CLOUD NATIVE
COMPUTING FOUNDATION

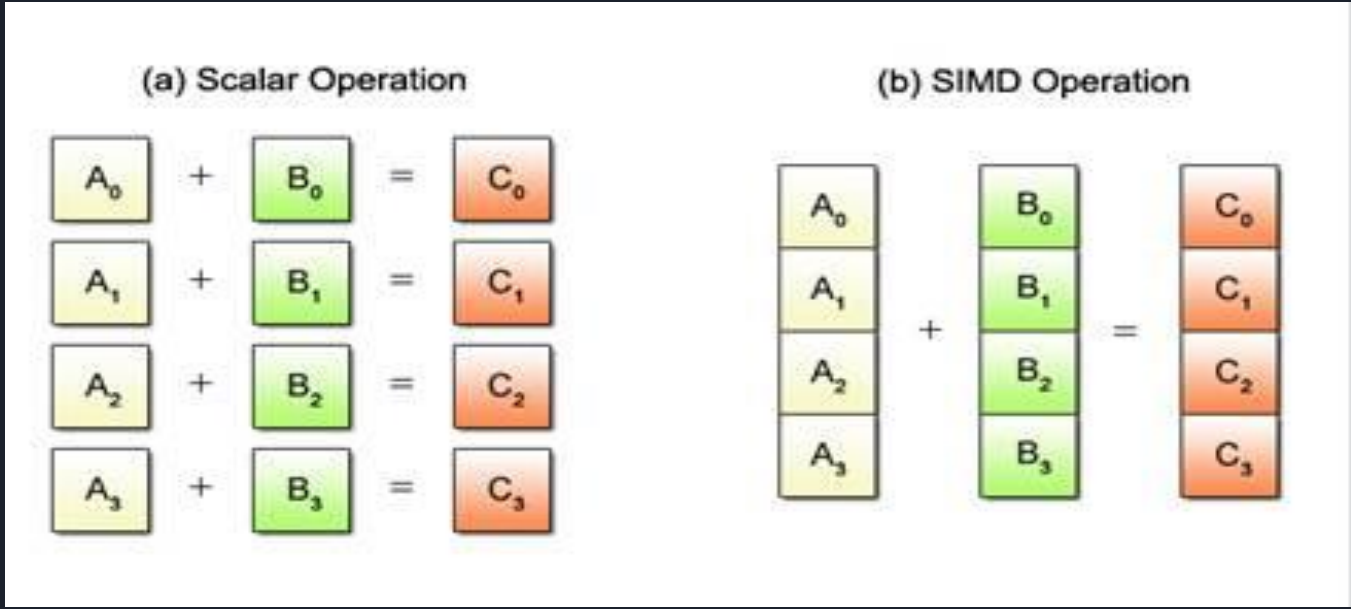
THEJAS32 SoC



THEJAS32 SoC is based on the VEGA ET1031 processor, which is a 32 bit single core in-order, 3-stage pipeline processor. This processor is based on the open source RISC-V Instruction Set Architecture and operates at a frequency of 100MHz. The SoC also includes peripherals like 256KB internal SRAM, Three UARTs, Four SPIs, Three TIMERs, Eight PWMs, Three I2C interfaces, 32 GPIOs etc. This SoC is designed and developed by Centre for Development of Advanced Computing (C-DAC) as part of the Digital India RISC-V (DIR-V) Program, by the Ministry of Electronics and Information Technology, Government of India.

SIMD : Single instruction Multiple Data

How SIMD Operates ?

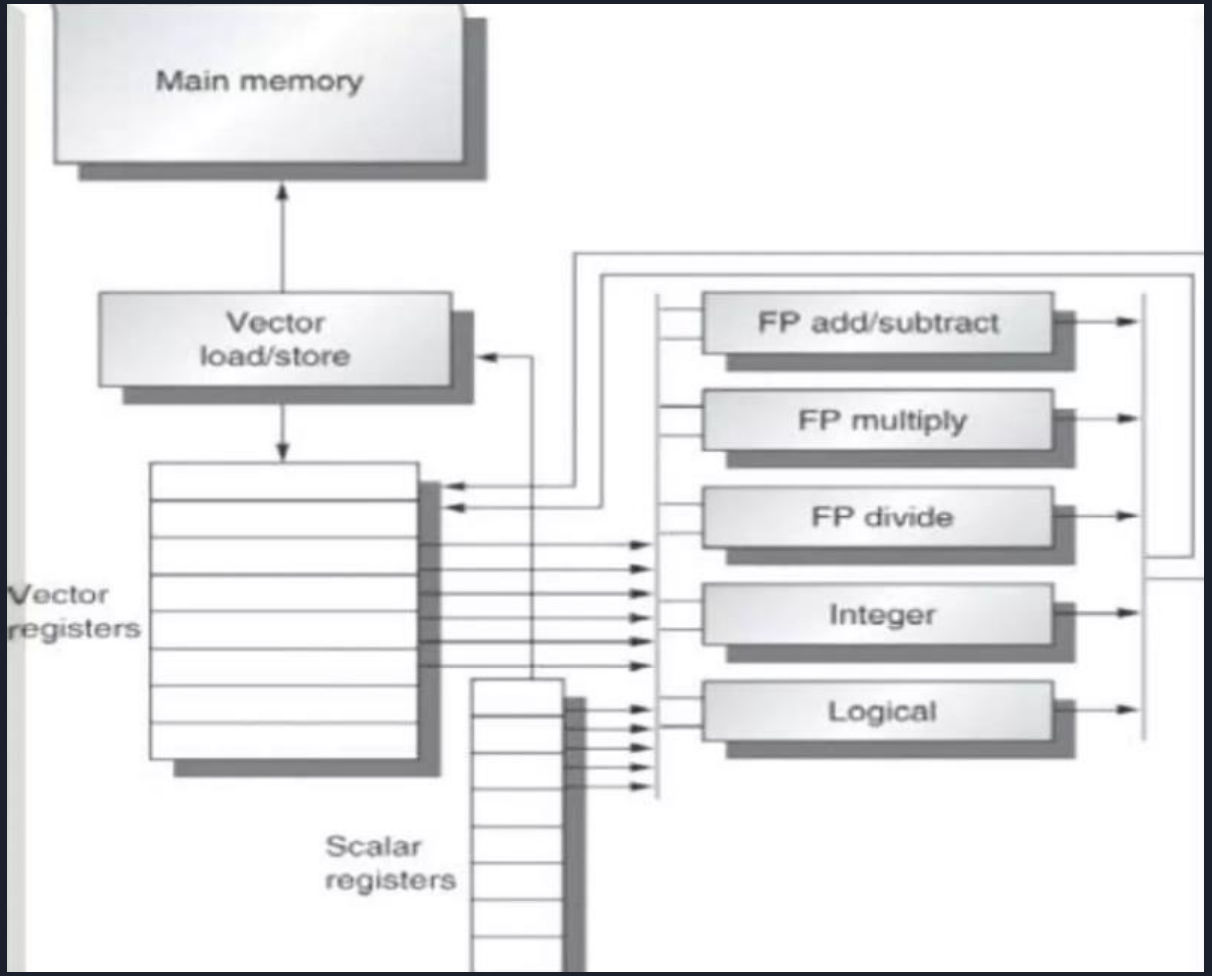


Vector Processing

- Vector Processing Architecture
  - Memory to Memory Architecture (Traditional)
  - Register to Register Architecture (Modern)

- Components of vector processors
  - Vector Registers
  - Vector functional Units
  - Vector load-store Units
  - Scalar Registers

- Advantages of Vector processing
  - Quick fetch and decode of single instruction for multiple operations
  - Easier Addressing of main memory
  - Elimination of memory Wastage
  - Simplification of control hazards
  - Reduced code size


Main memory
Vector load/store
FP add/subtract
FP multiply
FP divide
Integer
Logical
Vector registers
Scalar registers

RISC-V V/S RISC-V V

Risc-v

–Base instruction Set Architecture

–Focuses in Scalar Operations

–Handles individually data elements

–Performance is less as compared to RVV

–Easy to understand

–Widely Adopted

Risc-v v

–Optional extension adds capabilities of SIMD

–Focuses on Vector Operations

–Handles multiple data elements

–Designed to work on large data sets

–High Performance

–RVV support is growing but not universally implemented on all all Risc-v

RISC-V VECTOR EXTENSION

Unleashing Parallelism in RISC-V Architecture

RISC-V EXTENSIONS

- RISC-V is augmented via the concept of extensions
  - Extensions can add new instructions and CPU state
- Base ISA is called I (for Integer)
  - RV32I
  - RV64I (XLEN=64, adds a few arithmetic instructions to improve 32-bit integer arithmetic)
- Common Standard Extensions in a RISC-V 64-bit Linux capable core

IMAFD = G

  - M. Integer multiplication and division (mul, div, rem, …)
  - A. Atomic instructions (load reserve + store conditional, atomic read-modify-write)
  - F. Single-Precision Floating-Point (IEEE 754 Binary32)
  - D. Double-Precision Floating-Point (IEEE 754 Binary64)
  - C. Compressed Instructions (16-bit encodings for common I/F/D instructions)

INTRODUCTION

- **What it is:** A set of instructions for RISC-V processors that enable them to perform vector operations.
- **What it does:** Allows processing multiple data elements at once, leading to faster performance for specific tasks.
- **When it's useful:** **For tasks involving large amounts of data and repetitive calculations**, such as:
  - Signal processing
  - Image and video processing
  - Machine learning
  - Scientific computing

OVERVIEW OF ITS FEATURES AND CAPABILITIES

1. **SIMD (Single Instruction, Multiple Data)**: Allows a single instruction to perform the same operation on multiple data elements simultaneously, exploiting data-level parallelism.
2. **Enhanced Performance:** By processing multiple data elements in parallel, the Vector Extension significantly improves throughput and efficiency for tasks such as multimedia processing, scientific computing, and machine learning.
3. **Customizable Vector Length:** Supports vectors of variable lengths, enabling optimization for different applications and target platforms.
4. **Vector Registers:** Dedicated vector registers hold vector data, facilitating efficient manipulation and computation.
5. **Vector Memory Accesses:** Supports efficient vector memory accesses, reducing overhead in loading and storing vector data.
6. **Predication Support:** Enables conditional execution based on vector elements, enhancing flexibility in control flow.

RISC-V RV64GV TOOLCHAIN

```
muskan@cdackoji:~$ ls
lj-os  muskan  packages  packages1  qemu  riscv  self
muskan@cdackoji:~$ cd muskan/opt/bin/
muskan@cdackoji:~/muskan/opt/bin$ ls
riscv64-unknown-elf-addr2line      riscv64-unknown-elf-lto-dump          riscv64-unknown-linux-gnu-gcc-nm
riscv64-unknown-elf-ar             riscv64-unknown-elf-nm                riscv64-unknown-linux-gnu-gcc-ranlib
riscv64-unknown-elf-as             riscv64-unknown-elf-objcopy           riscv64-unknown-linux-gnu-gcov
riscv64-unknown-elf-c++            riscv64-unknown-elf-objdump           riscv64-unknown-linux-gnu-gcov-dump
riscv64-unknown-elf-c++filt        riscv64-unknown-elf-ranlib            riscv64-unknown-linux-gnu-gcov-tool
riscv64-unknown-elf-cpp            riscv64-unknown-elf-readelf           riscv64-unknown-linux-gnu-gdb
riscv64-unknown-elf-elfedit        riscv64-unknown-elf-run               riscv64-unknown-linux-gnu-gdb-add-index
riscv64-unknown-elf-g++            riscv64-unknown-elf-size              riscv64-unknown-linux-gnu-gfortran
riscv64-unknown-elf-gcc            riscv64-unknown-elf-strings           riscv64-unknown-linux-gnu-gprof
riscv64-unknown-elf-gcc-13.2.0     riscv64-unknown-elf-strip             riscv64-unknown-linux-gnu-ld
riscv64-unknown-elf-gcc-ar         riscv64-unknown-linux-gnu-addr2line   riscv64-unknown-linux-gnu-ld.bfd
riscv64-unknown-elf-gcc-nm         riscv64-unknown-linux-gnu-ar          riscv64-unknown-linux-gnu-lto-dump
riscv64-unknown-elf-gcc-ranlib     riscv64-unknown-linux-gnu-as          riscv64-unknown-linux-gnu-nm
riscv64-unknown-elf-gcov           riscv64-unknown-linux-gnu-c++         riscv64-unknown-linux-gnu-objcopy
riscv64-unknown-elf-gcov-dump      riscv64-unknown-linux-gnu-c++filt     riscv64-unknown-linux-gnu-objdump
riscv64-unknown-elf-gcov-tool      riscv64-unknown-linux-gnu-cpp         riscv64-unknown-linux-gnu-ranlib
riscv64-unknown-elf-gdb            riscv64-unknown-linux-gnu-elfedit     riscv64-unknown-linux-gnu-readelf
riscv64-unknown-elf-gdb-add-index  riscv64-unknown-linux-gnu-g++         riscv64-unknown-linux-gnu-run
riscv64-unknown-elf-gprof          riscv64-unknown-linux-gnu-gcc         riscv64-unknown-linux-gnu-size
riscv64-unknown-elf-ld             riscv64-unknown-linux-gnu-gcc-13.2.0  riscv64-unknown-linux-gnu-strings
riscv64-unknown-elf-ld.bfd         riscv64-unknown-linux-gnu-gcc-ar      riscv64-unknown-linux-gnu-strip
muskan@cdackoji:~/muskan/opt/bin$
```

RISC-V VECTOR INSTRUCTION

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>

#define MAXELEM         1000

uint64_t get_time_us() {
  struct timeval tv;
  gettimeofday(&tv, NULL);
  return (tv.tv_sec * 1000000) + tv.tv_usec;
}


extern void vvaddint32(size_t n, const int *x, const int *y, int *z);

int main()
{
        int x[MAXELEM] = {0};
        int y[MAXELEM] = {0};
        int z[MAXELEM] = {0};

        srand(time(NULL));

        for(int i=0; i < MAXELEM; i++)
        {
                x[i] = rand() % 1000;
                y[i] = rand() % 1000;
        }

        uint64_t begin = get_time_us();
        vvaddint32(MAXELEM, x, y, z);
        uint64_t elapsed = get_time_us() - begin;

        printf("\nTime taken for vector addition : %lu us\n", elapsed);



}
```

```asm
    .text
    .balign 4
    .global vvaddint32
    # vector-vector add routine of 32-bit integers
    # void vvaddint32(size_t n, const int*x, const int*y, int*z)
    # { for (size_t i=0; i<n; i++) { z[i]=x[i]+y[i]; } }
    #
    # a0 = n, a1 = x, a2 = y, a3 = z
    # Non-vector instructions are indented
vvaddint32:
    vsetvli t0, a0, e32, ta, ma  # Set vector length based on 32-bit vectors
    vle32.v v0, (a1)         # Get first vector
      sub a0, a0, t0         # Decrement number done
      slli t0, t0, 2         # Multiply number done by 4 bytes
      add a1, a1, t0         # Bump pointer
    vle32.v v1, (a2)         # Get second vector
      add a2, a2, t0         # Bump pointer
    vadd.vv v2, v0, v1       # Sum vectors
    vse32.v v2, (a3)         # Store result
      add a3, a3, t0         # Bump pointer
      bnez a0, vvaddint32    # Loop back
      ret                    # Finished
```

RISC-V SCALAR INSTRUCTION

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>

#define MAXELEM         1000

uint64_t get_time_us() {
  struct timeval tv;
  gettimeofday(&tv, NULL);
  return (tv.tv_sec * 1000000) + tv.tv_usec;
}


extern void vvaddint32(size_t n, const int *x, const int *y, int *z);

int main()
{
        int x[MAXELEM] = {0};
        int y[MAXELEM] = {0};
        int z[MAXELEM] = {0};

        srand(time(NULL));

        for(int i=0; i < MAXELEM; i++)
        {
                x[i] = rand() % 1000;
                y[i] = rand() % 1000;
        }

        uint64_t begin = get_time_us();
        for(int i=0; i< MAXELEM; i++)
    {
        z[i] = x[i] + y[i];
    }
        uint64_t elapsed = get_time_us() - begin;

        printf("\nTime taken for non-vector addition : %lu us\n", elapsed);
}
```

RISC-V MAKEFILE INSTRUCTION

```
CC = riscv64-unknown-elf-gcc

bin:
        $(CC) -c vvadd.s
        $(CC) -c main.c
        $(CC) main.o vvadd.o -o main
        $(CC) novect.c -o novect

run:
        @-qemu-riscv64-static -cpu rv64,v=true main
        @-qemu-riscv64-static -cpu rv64,v=true novect
```

EXPLORING RISC-V VECTOR EXTENSION WITH QEMU

```
muskan@cdackoji:~/self$ ls
main  main.c  main.o  Makefile  novect  novect.c  vvadd.o  vvadd.s
muskan@cdackoji:~/self$ make
riscv64-unknown-elf-gcc -c vvadd.s
riscv64-unknown-elf-gcc -c main.c
riscv64-unknown-elf-gcc main.o vvadd.o -o main
riscv64-unknown-elf-gcc novect.c -o novect
muskan@cdackoji:~/self$ make run

Time taken for vector addition : 339 us

Time taken for non-vector addition : 107 us
muskan@cdackoji:~/self$
```

X_86 ARCHITECTURE

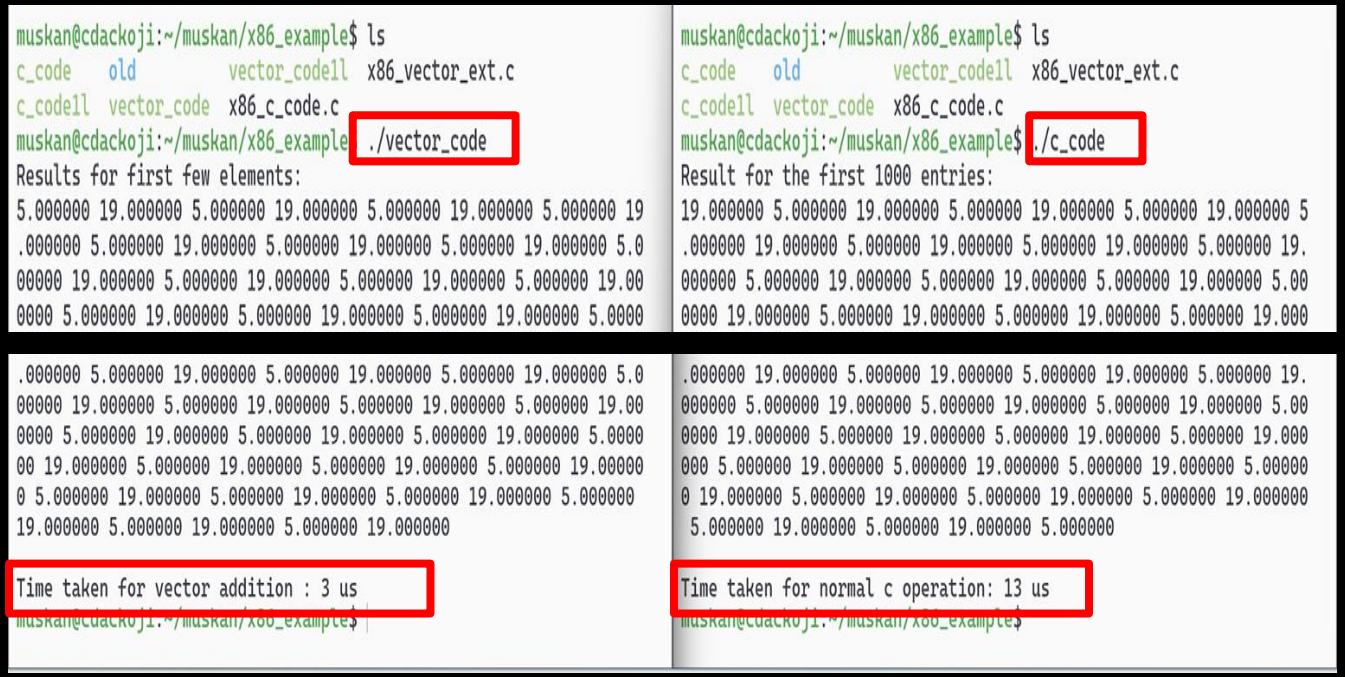
Vector Instruction

```c
#include <immintrin.h>
#include <stdio.h>
#include <sys/time.h>
#include <stdint.h>

#define n 100000
uint64_t get_time_us() {
  struct timeval tv;
  gettimeofday(&tv, NULL);
  return (tv.tv_sec * 1000000) + tv.tv_usec;
}
int main() {
    // Initialize vectors with constant values
    __m256d veca = _mm256_set1_pd(6.0);
    __m256d vecb = _mm256_set1_pd(2.0);
    __m256d vecc = _mm256_set1_pd(7.0);

    // Initialize result array to store 1000 result values
    double result[n];

        uint64_t begin = get_time_us();
    // Perform the operation for 1000 entries
    for (int i = 0; i < n; i += 4) {
        __m256d result_vec = _mm256_fmaddsub_pd(veca, vecb, vecc);
        _mm256_storeu_pd(&result[i], result_vec);
    }

     uint64_t elapsed = get_time_us() - begin;

    // Display a few elements of the result array
    printf("Results for first few elements:\n");
    for (int i = 0; i < n; i++) {
        printf("%lf ", result[i]);
    }
    printf("\n");
     printf("\nTime taken for vector addition : %lu us\n", elapsed);

    return 0;
}
```

Scalar Instruction

```c
#include <stdio.h>
#include <sys/time.h>
#include <stdint.h>

#define NUM_ENTRIES 100000

double get_time_us() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return (tv.tv_sec * 1000000) + tv.tv_usec;
}

void fmaddsub(double *a, double *b, double *c, double *result, int n) {
    for (int i = 0; i < n; i++) {
        result[i] = a[i] * b[i] + ((i % 2 == 0) ? c[i] : -c[i]);
    }
}

int main() {
    double veca[NUM_ENTRIES];
    double vecb[NUM_ENTRIES];
    double vecc[NUM_ENTRIES];
    double result[NUM_ENTRIES];

    // Initialize vectors
    for (int i = 0; i < NUM_ENTRIES; i++) {
        veca[i] = 6.0;
        vecb[i] = 2.0;
        vecc[i] = 7.0;
    }

    // Perform the operation
    uint64_t begin = get_time_us();
    fmaddsub(veca, vecb, vecc, result, NUM_ENTRIES);
    uint64_t elapsed = get_time_us() - begin;

    // Display the result
    printf("Result for the first %d entries:\n", NUM_ENTRIES);
    for (int i = 0; i < NUM_ENTRIES; i++) {
        printf("%lf ", result[i]);
    }
    printf("\n\nTime taken for normal c operation: %lu us\n", elapsed);

    return 0;
}
```

EXPLORING VECTOR EXTENSION WITH X_86 ARCHITECTURE



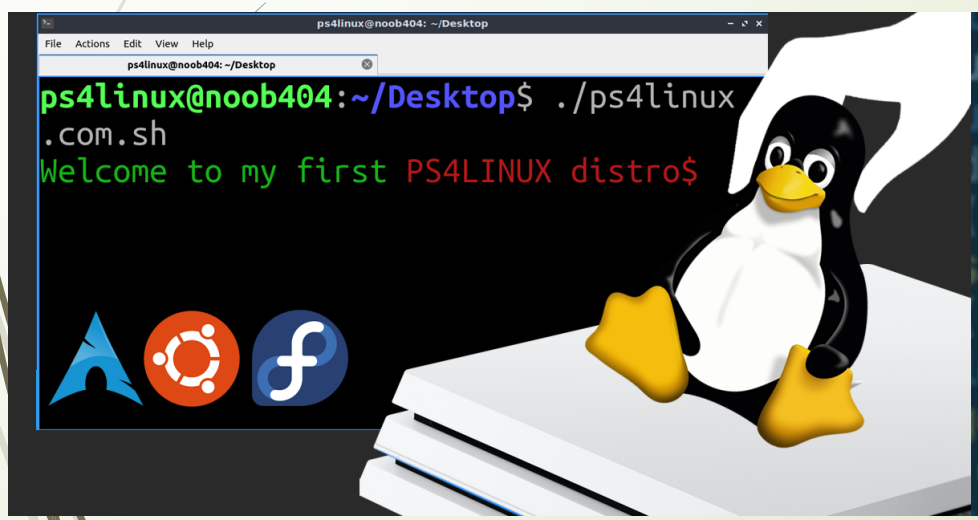
CHALLENGES AND LIMITATIONS

1. **Programming Complexity**: Utilizing vector instructions effectively requires specialized programming techniques such as loop unrolling, data alignment, and explicit vectorization, which can increase code complexity and development time.
2. **Data Dependency:** Dependencies among vector elements can limit parallelism and lead to inefficient utilization of vector resources, especially in algorithms with irregular data access patterns or conditional branching.
3. **Memory Bandwidth:** Vectorized operations can impose significant demands on memory bandwidth, especially for large vector lengths, leading to potential bottlenecks and inefficiencies in memory-bound applications.
4. **Hardware Complexity:** Implementing a high-performance vector unit entails additional hardware resources and complexity, including wider data paths, vector registers, and efficient memory access mechanisms, which may increase design complexity and power consumption.
5. **Software Ecosystem:** The availability of software tools, libraries, and compilers optimized for the Vector Extension may be limited compared to established architectures, posing challenges for developers in leveraging its full potential.

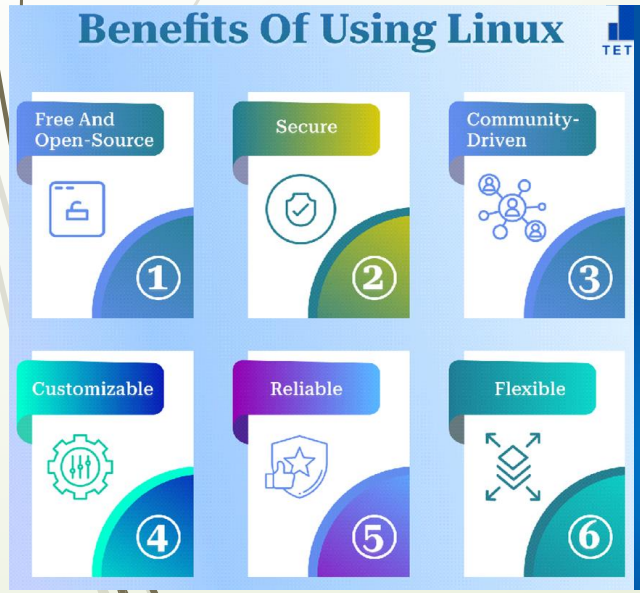
AREAS FOR IMPROVEMENT AND ONGOING RESEARCH

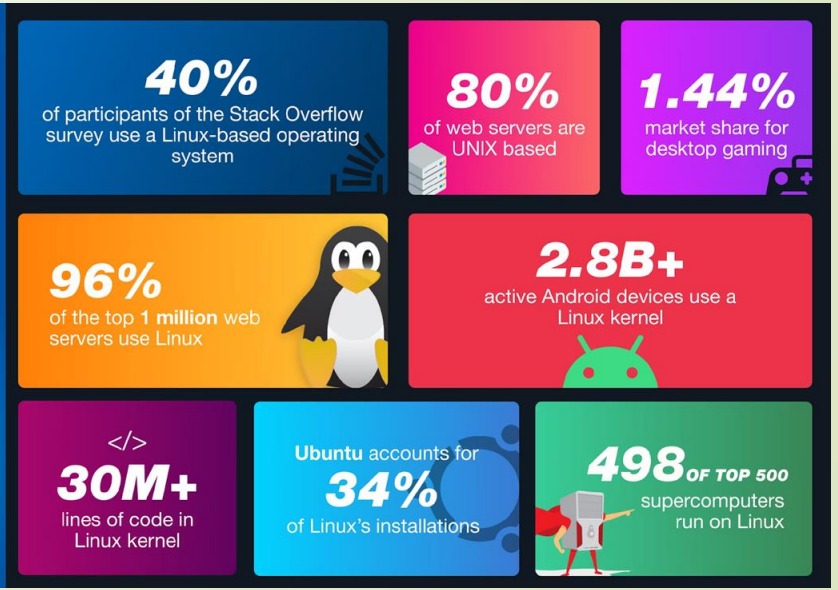
1. **Compiler Optimization**: Continued research is needed to develop advanced compiler techniques for automatic vectorization, loop restructuring, and optimization of memory access patterns to extract maximum parallelism from applications.
2. **Vectorization Libraries:** Further development of vectorization libraries and frameworks tailored for the Vector Extension can simplify programming and accelerate adoption by providing high-level abstractions and optimized routines for common tasks.
3. **Memory Hierarchy:** Research into efficient data prefetching, caching strategies, and memory hierarchy designs is crucial to mitigate the impact of memory bandwidth limitations and improve overall system performance.
4. **Scalability and Heterogeneity:** Investigating techniques for scalable vector processing across multiple cores and heterogeneous architectures, including integration with multi-core processors and accelerators, can enhance performance and flexibility in diverse computing environments.
5. **Energy Efficiency:** Optimizing vector architectures for energy efficiency through techniques such as dynamic voltage and frequency scaling, power gating, and instruction-level energy reduction is essential for enabling high-performance, energy-efficient computing systems.

DIY LINUX – What?



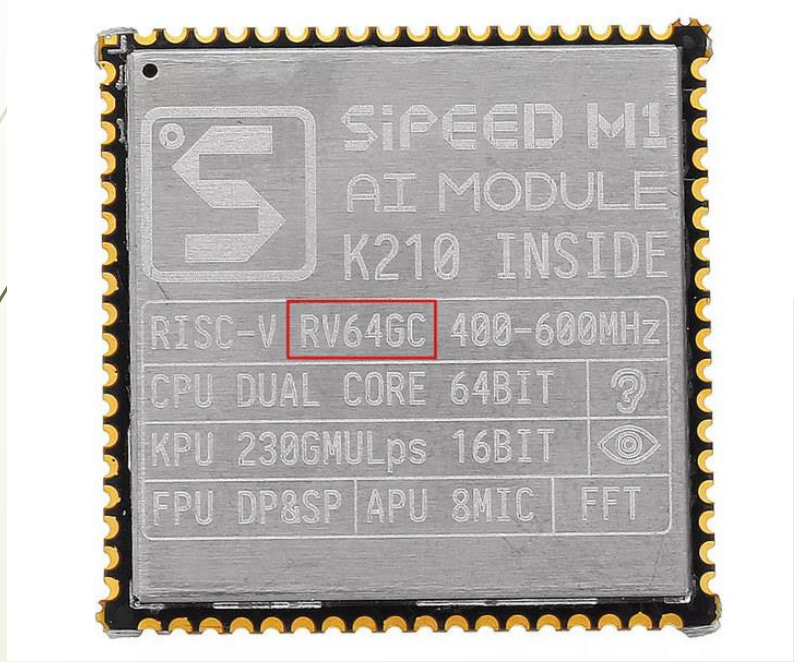
DIY LINUX – Why?

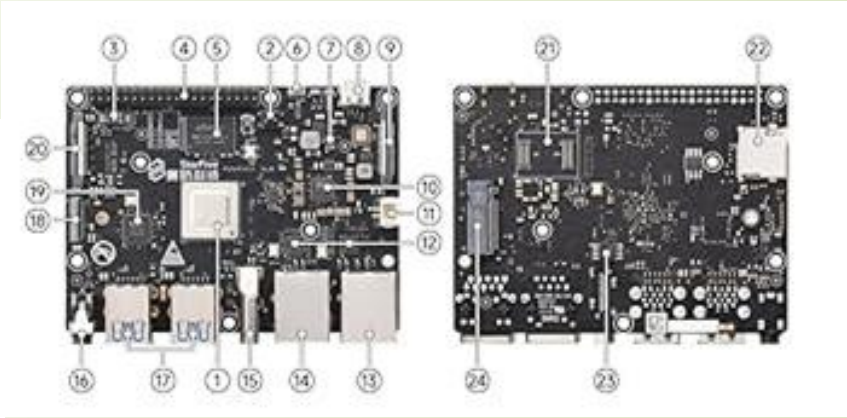




DIY LINUX – Where? – RVV (RISC-V Vector)

riscv64-gv (RV64GV)





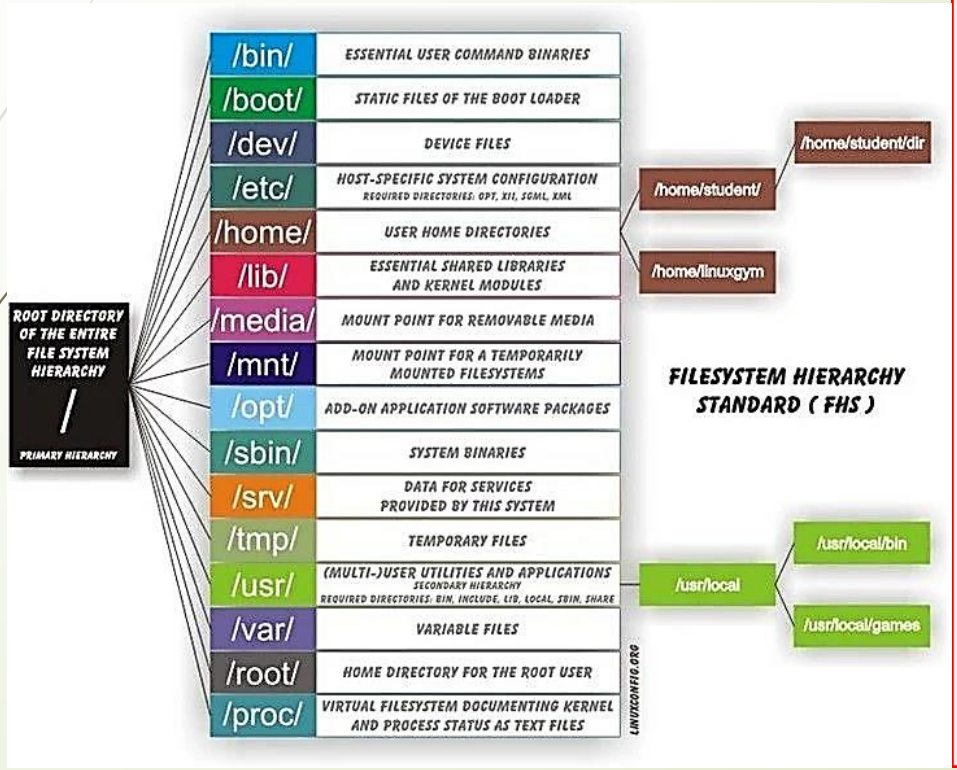
```
jaufranc@FX8350: /media/hdd/edev/sandbox/riscv-linux/release
ucbvax login: root
Password:

BusyBear Linux

root@ucbvax:~# uname -a
Linux ucbvax 4.14.0-00032-gd10799b22913-dirty #79 Sun Dec 17 10:41:31 NZDT 2017
riscv64 GNU/Linux
root@ucbvax:~# cat /proc/cpuinfo
hart    : 0
isa     : rv64imafdcsu
mmu     : sv48

root@ucbvax:~# df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/root                58.0M     44.9M      8.6M  84% /
devtmpfs                 59.8M         0     59.8M   0% /dev
none                     59.9M         0     59.9M   0% /tmp
none                     59.9M         0     59.9M   0% /var/tmp
root@ucbvax:~#
```

DIY LINUX – How?

1) Configuring the Environment
2) Making cross Toolchain

DIY LINUX – How?

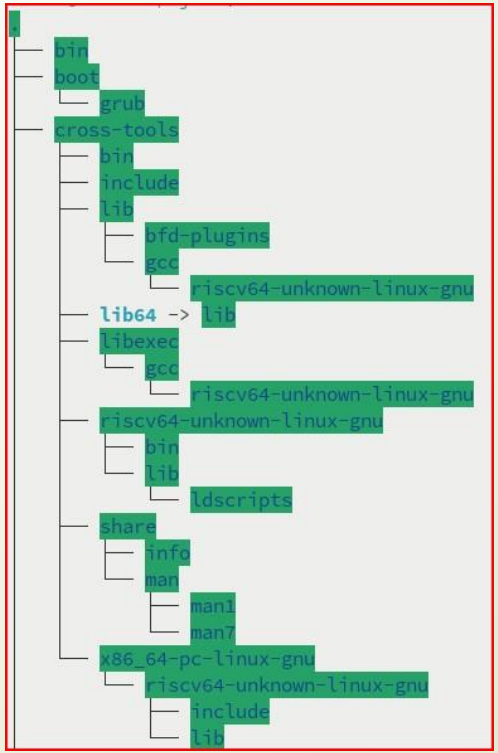
1) Configuring the Environment

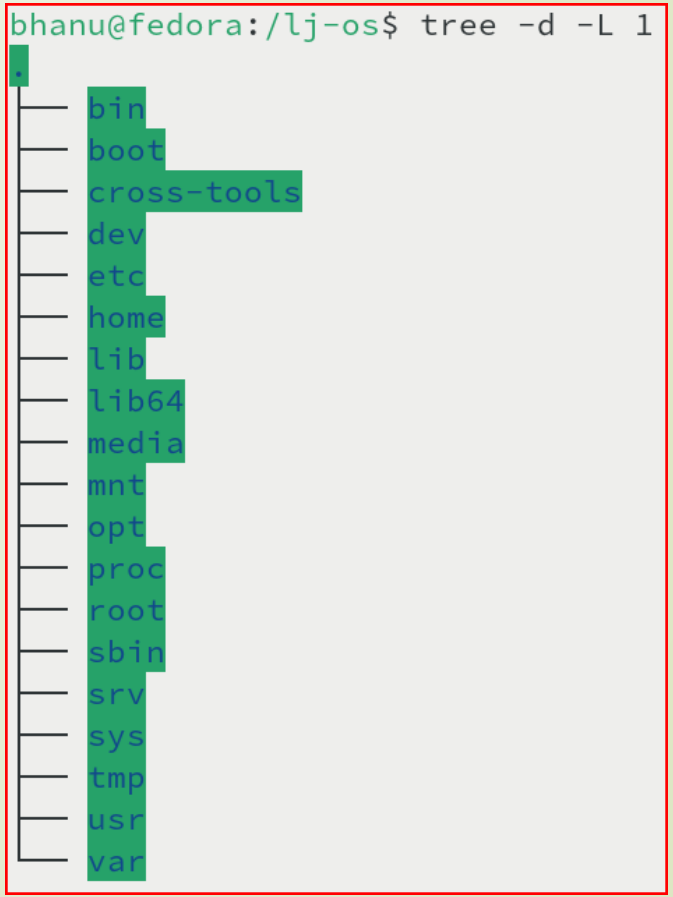



```
bhanu@fedora:/lj-os$ tree -d -L 1
.
├── bin
├── boot
├── cross-tools
├── dev
├── etc
├── home
├── lib
├── lib64
├── media
├── mnt
├── opt
├── proc
├── root
├── sbin
├── srv
├── sys
├── tmp
├── usr
└── var
```

DIY LINUX – How?

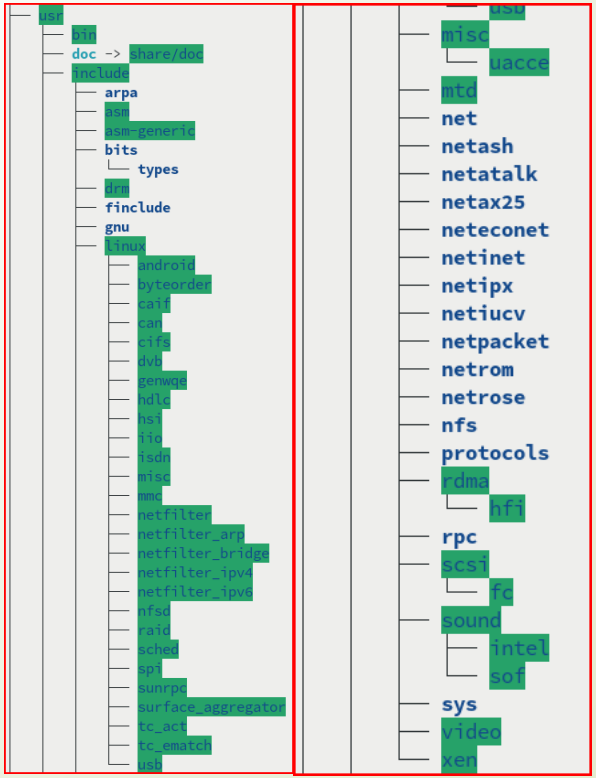
1) Configuring the Environment
    1) bin

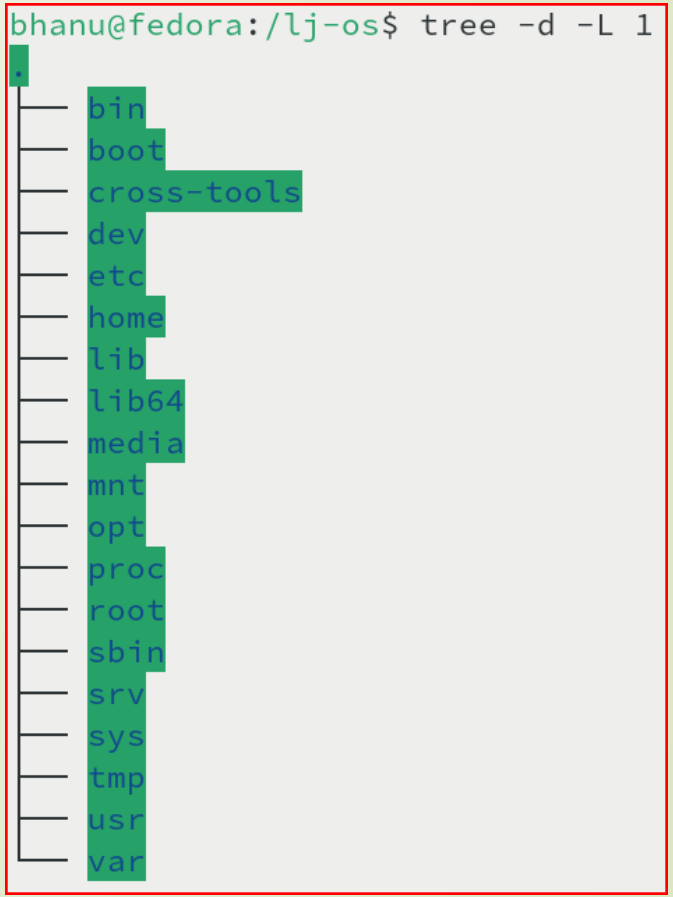




DIY LINUX – How?

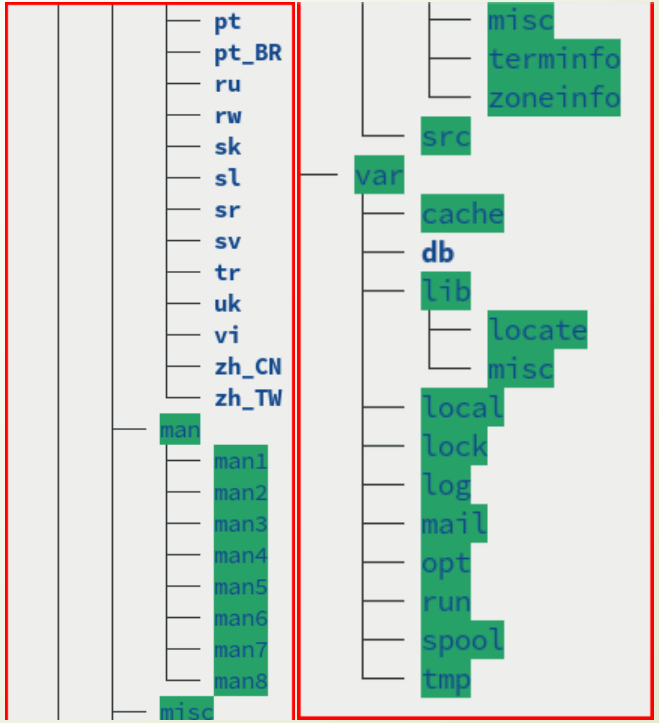
1) Configuring the Environment
   1) usr





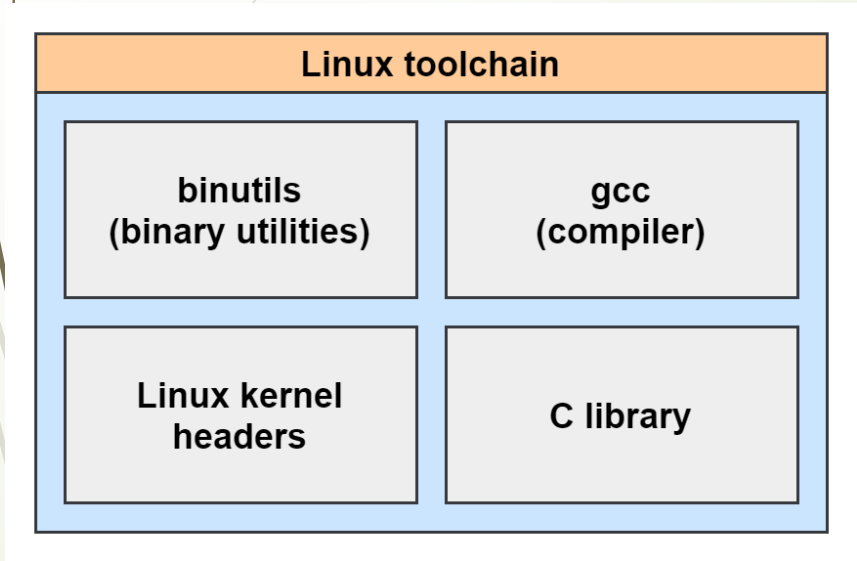
DIY LINUX – How?

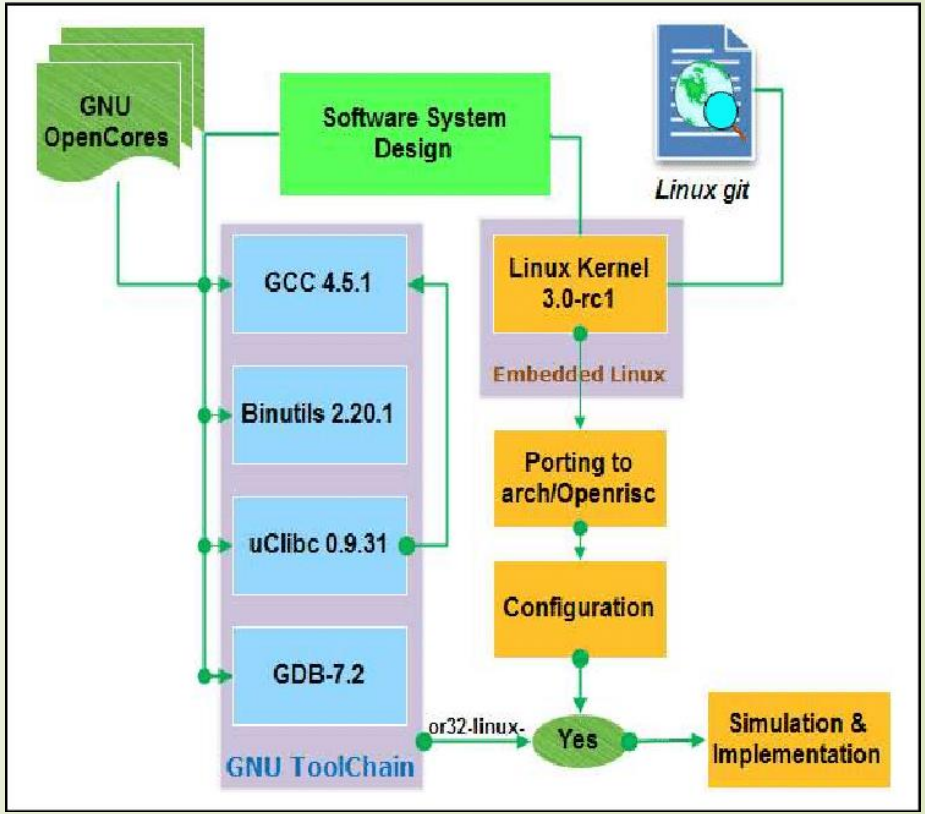
1) Configuring the Environment
    1) var




```
bhanu@fedora:/lj-os$ tree -d -L 1
.
├── bin
├── boot
├── cross-tools
├── dev
├── etc
├── home
├── lib
├── lib64
├── media
├── mnt
├── opt
├── proc
├── root
├── sbin
├── srv
├── sys
├── tmp
├── usr
└── var
```

DIY LINUX – How? – Toolchain

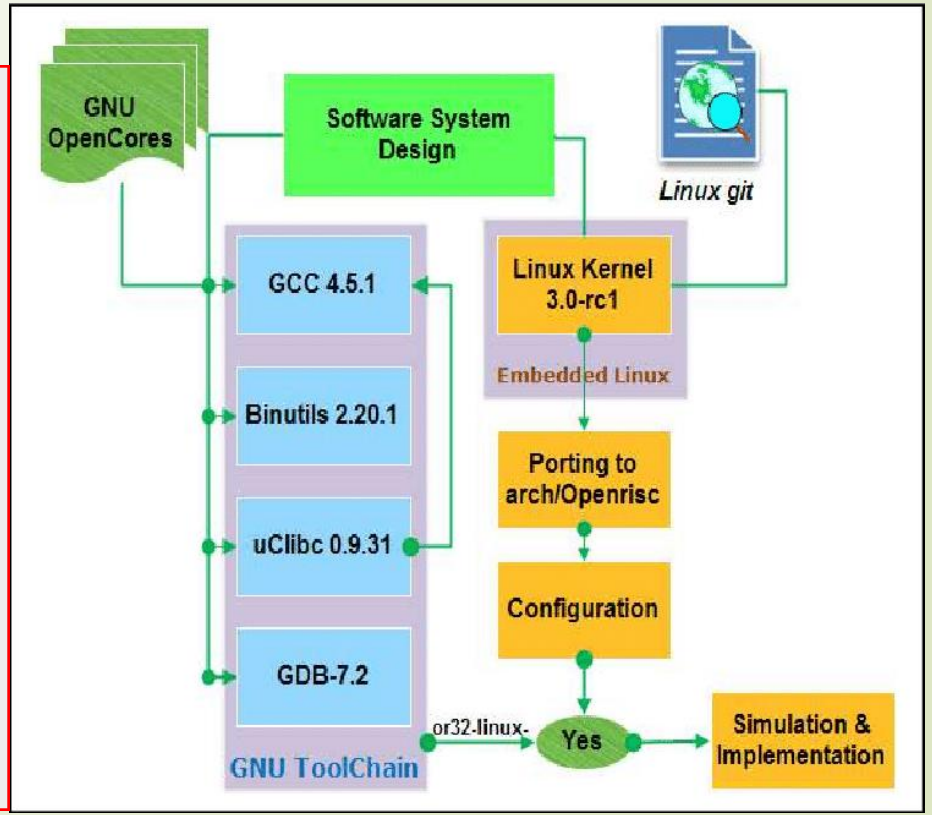




DIY LINUX – How? –

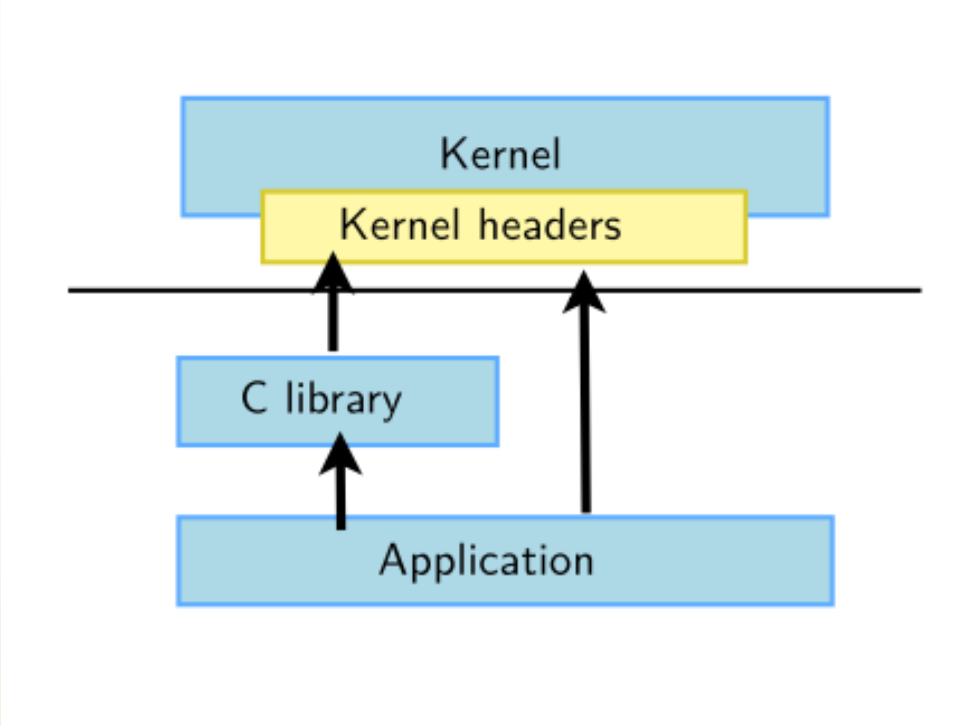
Toolchain

```
bhanu@fedora:~/Documents/tars_glibc/tars$ tree -d -L 2
.
├── binutils
│   ├── binutils-2.39
│   └── binutils-build
├── busybox
├── clfs
├── gcc
│   ├── gcc-13.0.1-20230401
│   ├── gcc-build
│   └── gcc-static
├── glibc
│   ├── glibc-2.37
│   └── glibc-build
├── gmp
│   └── gmp-6.2.1
├── isl
│   └── isl-0.24
├── linux
│   └── linux-6.7.3
├── mpc
│   └── mpc-0.34
├── mpfr
│   └── mpfr-4.1.1
└── zlib
```



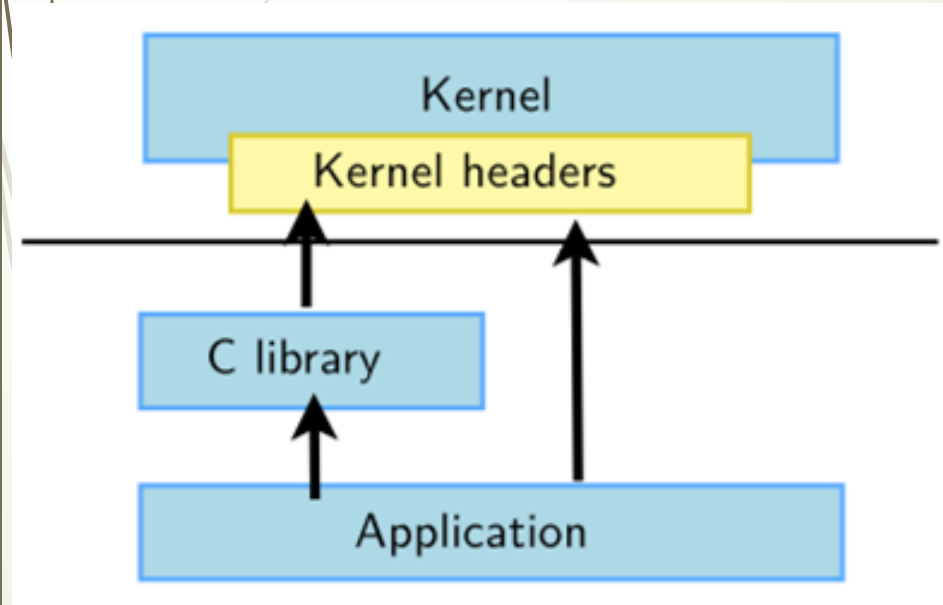
DIY LINUX – How? –

Kernel Headers - includes




```
include
    acpi
    asm-generic
    clocksource
    crypto
    drm
    dt-bindings
    generated
    keys
    kunit
    kvm
    linux
    math-emu
    media
    memory
    misc
    net
    pcmcia
    ras
    rdma
    rv
    scsi
    soc
    sound
    target
    trace
    uapi
    ufs
    vdso
    video
    xen
```

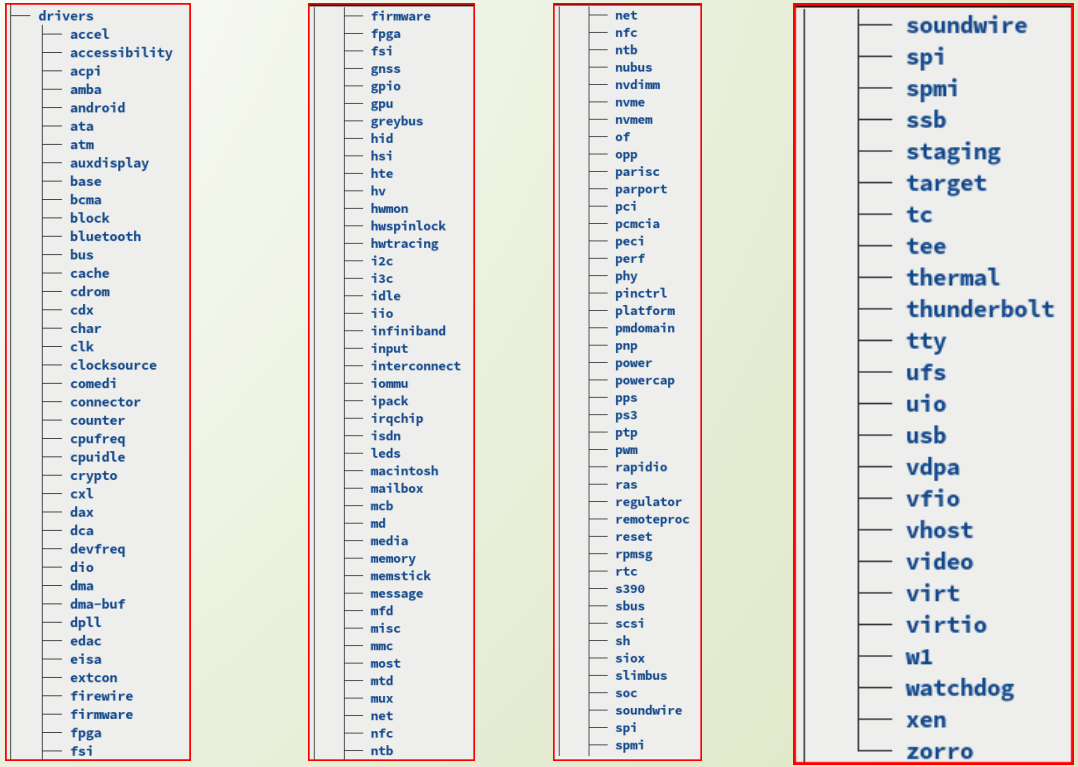
DIY LINUX – How? –
Kernel Headers – architecture dependent



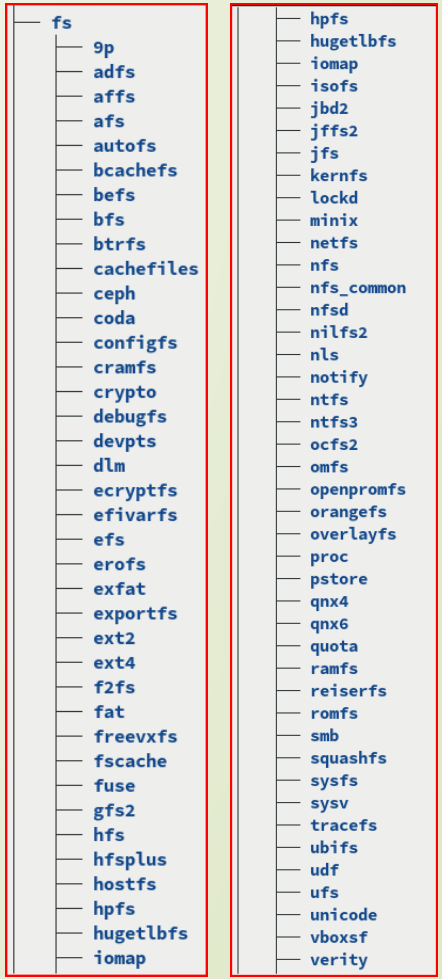

```
bhanu@fedora:~/Documents/tars_glibc/tars/linux/linux-6.7.3$ tree -d -L 2
.
├── arch
│   ├── alpha
│   ├── arc
│   ├── arm
│   ├── arm64
│   ├── csky
│   ├── hexagon
│   ├── loongarch
│   ├── m68k
│   ├── microblaze
│   ├── mips
│   ├── nios2
│   ├── openrisc
│   ├── parisc
│   ├── powerpc
│   ├── riscv
│   ├── s390
│   ├── sh
│   ├── sparc
│   ├── um
│   ├── x86
│   └── xtensa
```

DIY LINUX – How? –

Kernel Headers - drivers

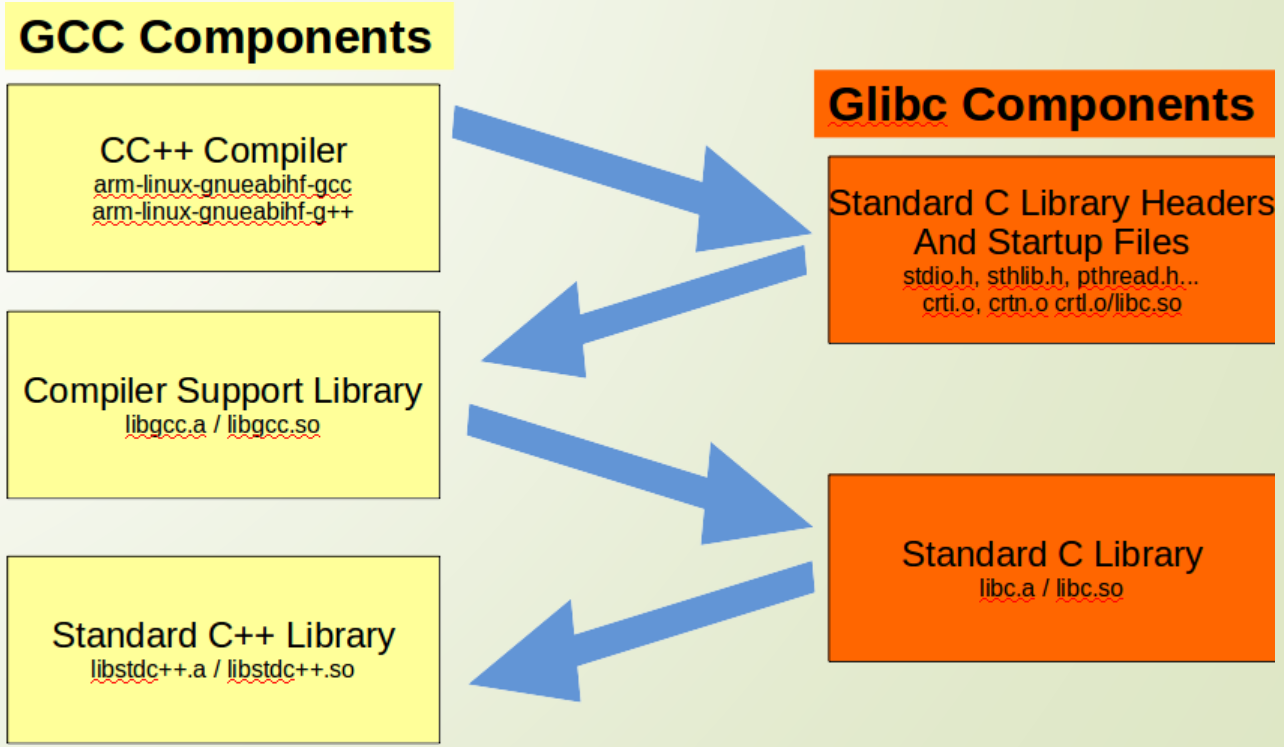


DIY LINUX – How? – Kernel Headers – file system



DIY LINUX – How?

Compiler



DIY LINUX – How?

Compiler –Resources to build compiler

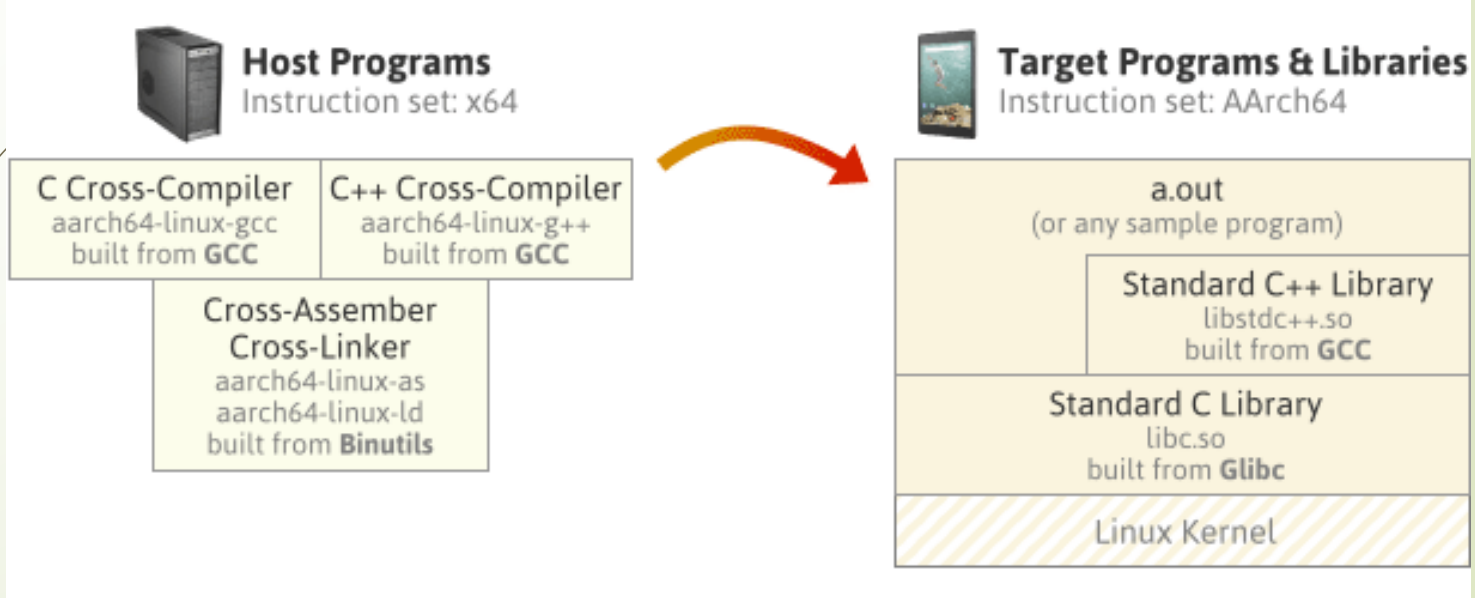
```
bhanu@fedora:~/Documents/tars_glibc/tars/gcc/gcc-13.0.1-20230401$ tree -d -L 1
.
├── config
├── contrib
├── c++tools
├── fixincludes
├── gcc
├── gnattools
├── gotools
├── include
├── INSTALL
├── intl
├── libada
├── libatomic
├── libbacktrace
├── libcc1
├── libcody
├── libcpp
├── libdecnumber
├── libffi
├── libgcc
├── libgfortran
├── libgm2
├── libgo
├── libgomp
├── libiberty
├── libitm
├── libobjc
├── libphobos
├── libquadmath
├── libsanitizer
├── libssp
├── libstdc++-v3
├── libvtv
├── lto-plugin
├── maintainer-scripts
```

```
bhanu@fedora:~/Documents/tars_glibc/tars/gcc/gcc-static$ tree -d -L 1
.
├── build-x86_64-cross-linux-gnu
├── fixincludes
├── gcc
├── intl
├── libbacktrace
├── libcody
├── libcpp
├── libdecnumber
├── libiberty
├── lto-plugin
├── riscv64-unknown-linux-gnu
└── zlib
```

```
bhanu@fedora:~/Documents/tars_glibc/tars/gcc/gcc-build$ tree -d -L 1
.
├── build-x86_64-cross-linux-gnu
├── build-x86_64-pc-linux-gnu
├── c++tools
├── fixincludes
├── gcc
├── intl
├── libbacktrace
├── libcody
├── libcpp
├── libdecnumber
├── libiberty
├── lto-plugin
├── x86_64-pc-linux-gnu
└── zlib
```

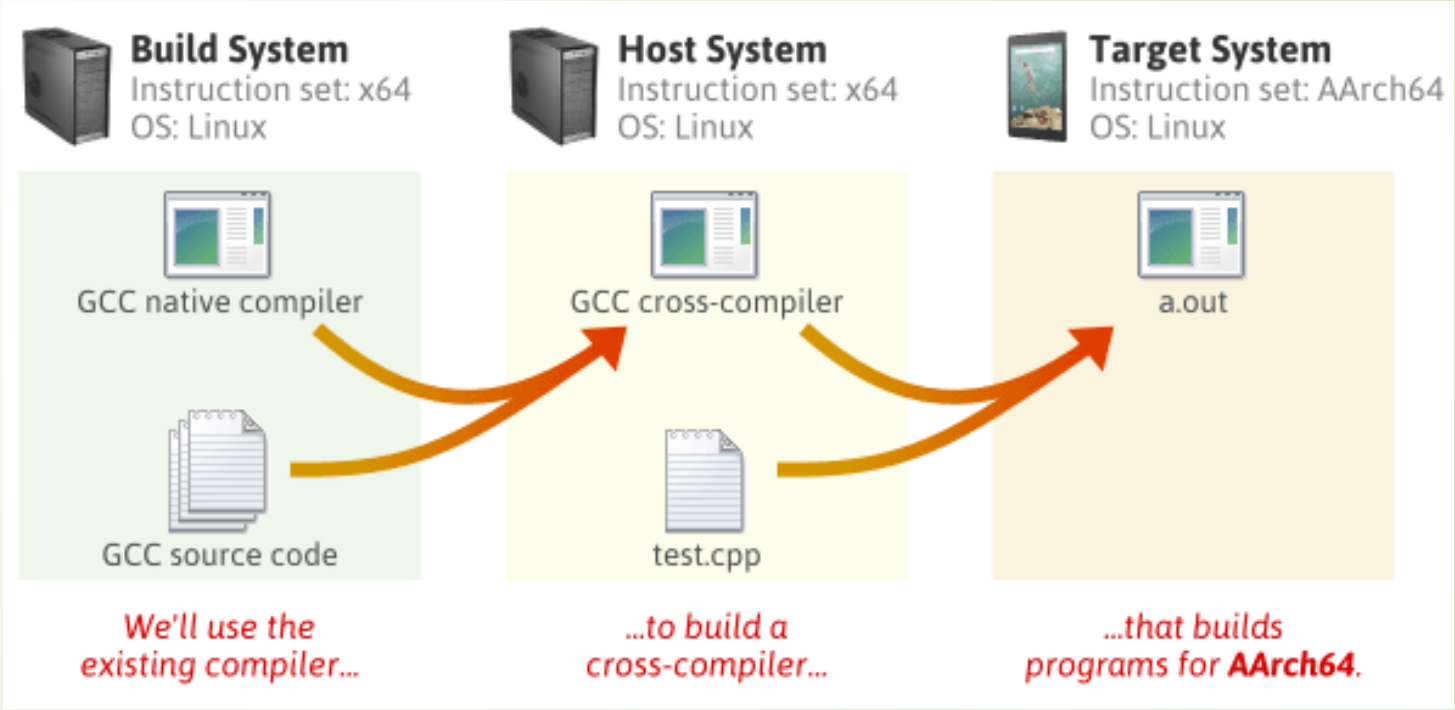
DIY LINUX – How?

Cross-Compiler



DIY LINUX – How?

Cross-Compiler - build



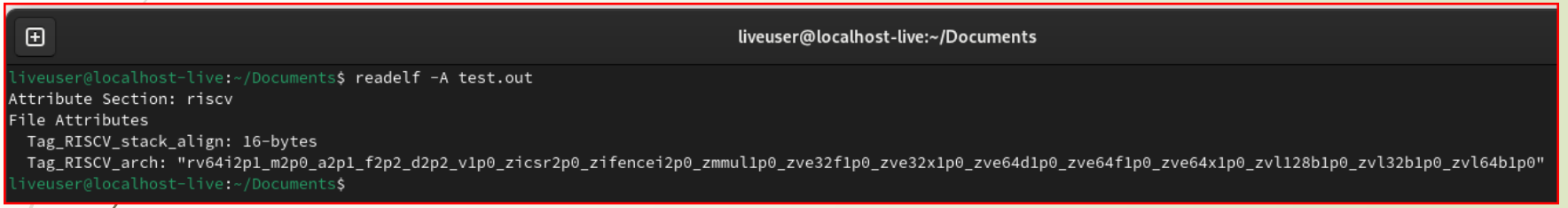
DIY LINUX – How?

Cross-Compiler - contains

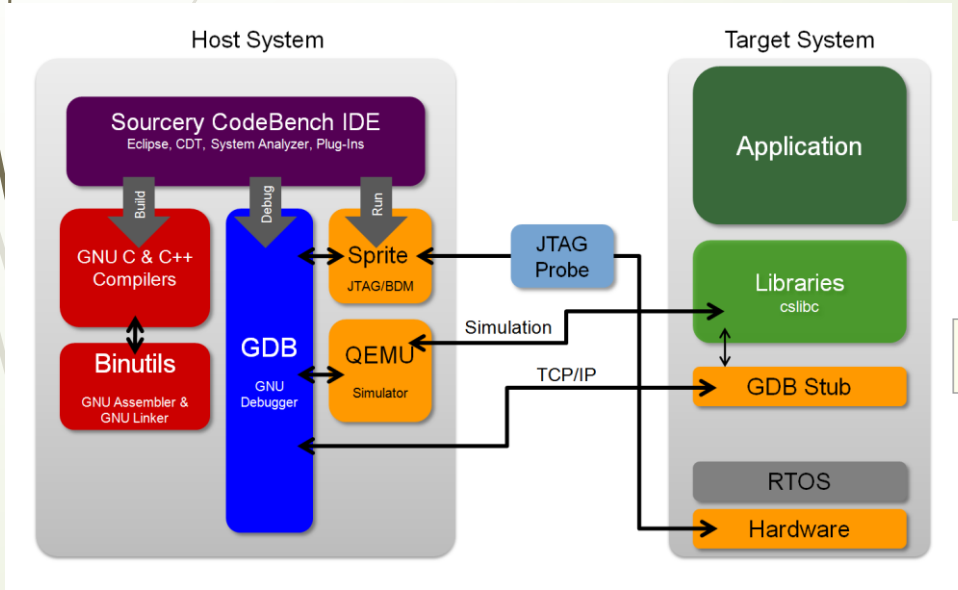
```
bhanu@fedora:/lj-os/cross-tools$ tree -d -L 2
.
├── bin
├── include
├── lib
│   ├── bfd-plugins
│   └── gcc
├── lib64 -> lib
├── libexec
│   └── gcc
├── riscv64-unknown-linux-gnu
│   ├── bin
│   └── lib
├── share
│   ├── info
│   └── man
└── x86_64-pc-linux-gnu
    └── riscv64-unknown-linux-gnu
```

```
bhanu@fedora:/lj-os/cross-tools$ tree -d -L 9
.
├── bin
├── include
├── lib
│   ├── bfd-plugins
│   └── gcc
│       └── riscv64-unknown-linux-gnu
│           └── 13.0.1
│               ├── include
│               ├── include-fixed
│               ├── install-tools
│               │   └── include
│               └── plugin
│                   └── include
│                       ├── ada
│                       │   └── gcc-interface
│                       ├── c-family
│                       ├── config
│                       │   └── riscv
│                       ├── cp
│                       ├── d
│                       ├── m2
│                       └── objc
├── lib64 -> lib
├── libexec
│   └── gcc
│       └── riscv64-unknown-linux-gnu
│           └── 13.0.1
│               ├── install-tools
│               └── plugin
├── riscv64-unknown-linux-gnu
│   ├── bin
│   └── lib
│       └── ldscripts
├── share
│   ├── info
│   └── man
│       ├── man1
│       └── man7
└── x86_64-pc-linux-gnu
    └── riscv64-unknown-linux-gnu
        ├── include
        └── lib
```

DIY LINUX – How?
Cross-Compiler executable test



DIY LINUX – How?
Cross-Compiler - Application




प्रगत संगणन विकास केंद्र
सी डैक
CDAC
॥ ज्ञानादेव तु कैवल्यम्॥
CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

THANK YOU !