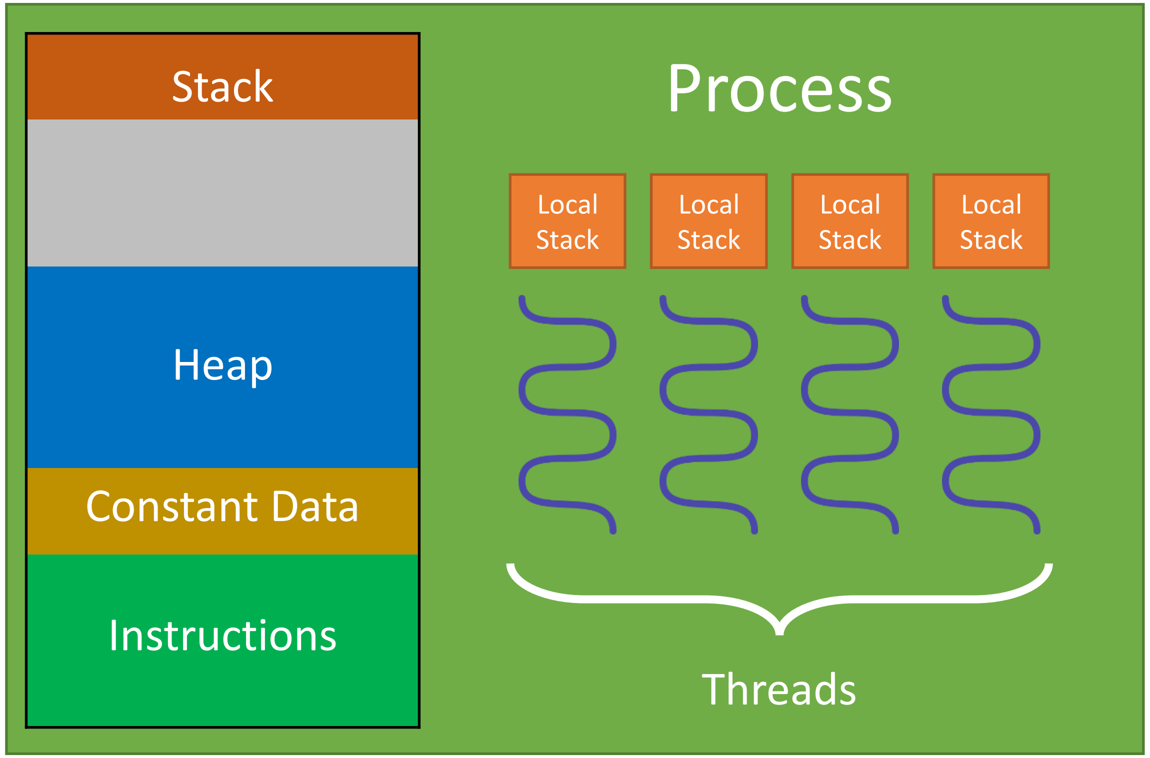
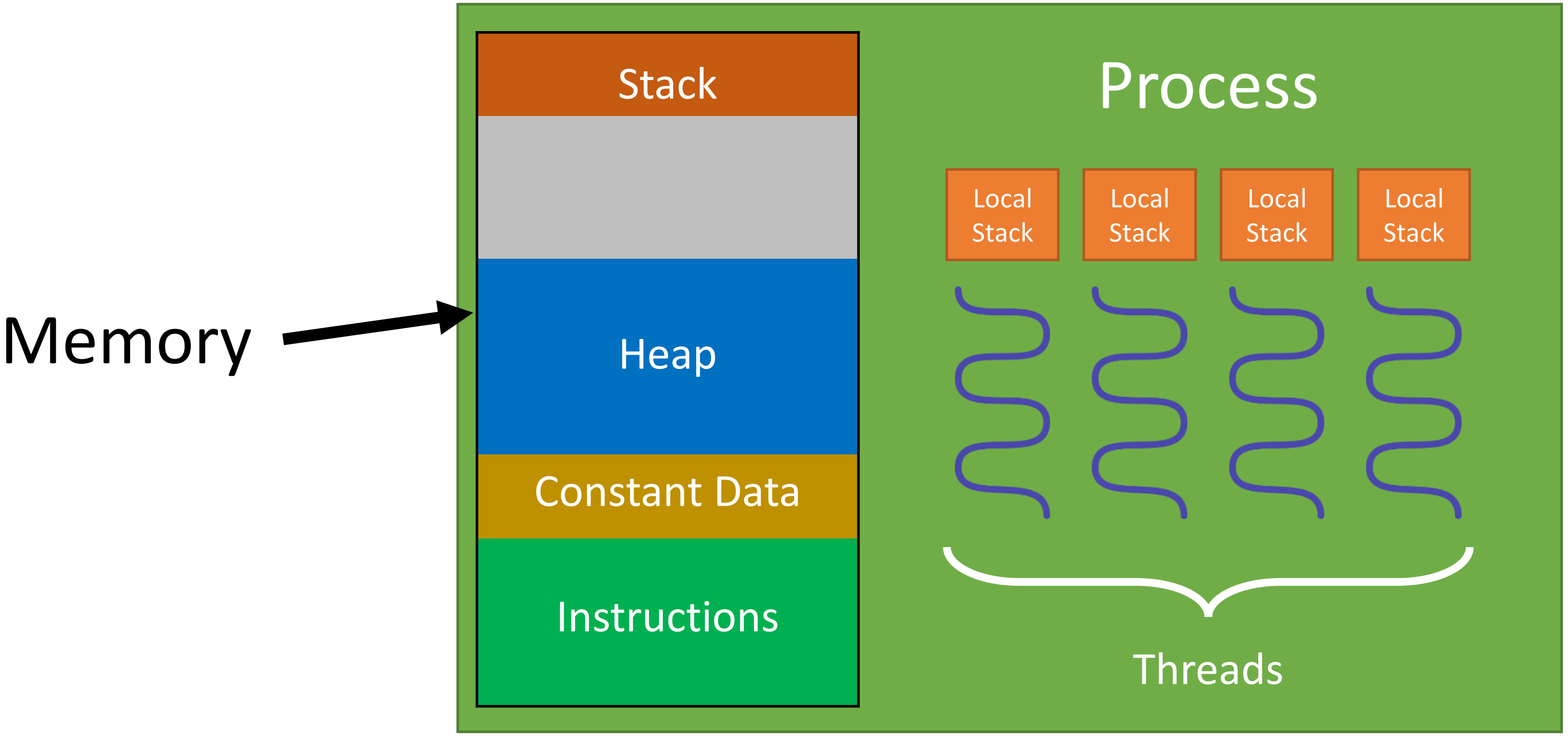
High Performance Computing in Julia

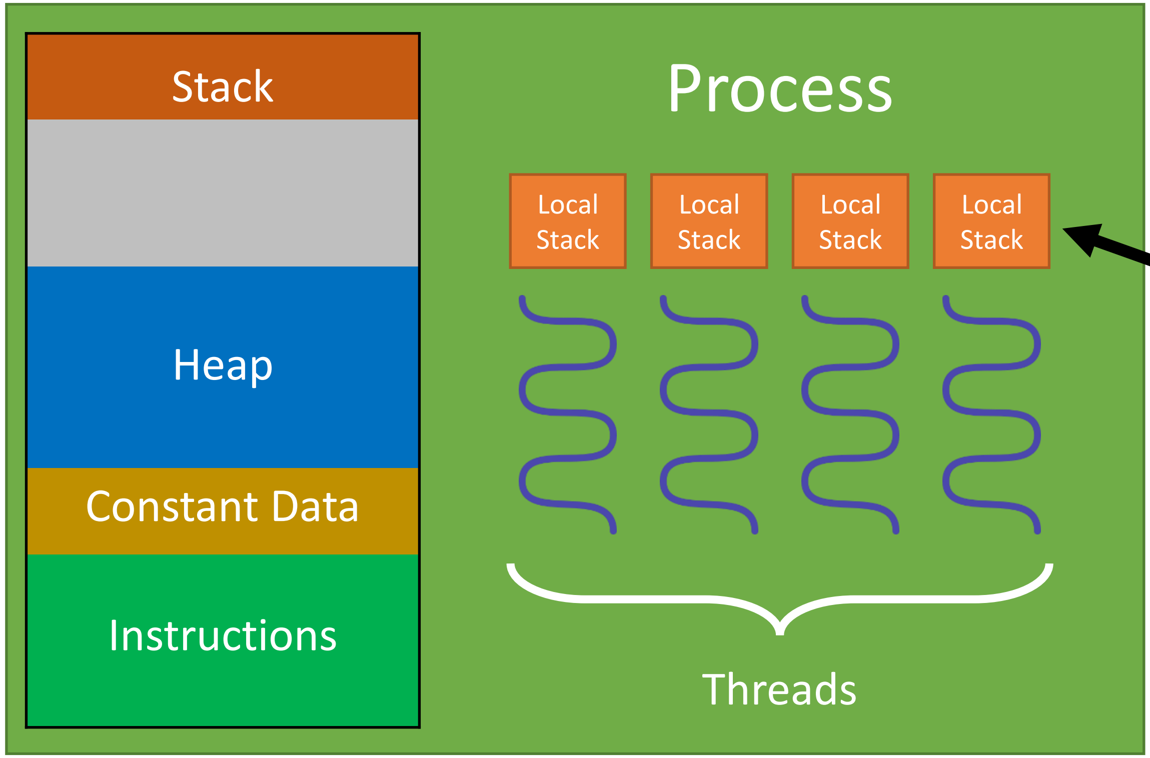
from the ground up.

Multiprocessing & Cluster Computing

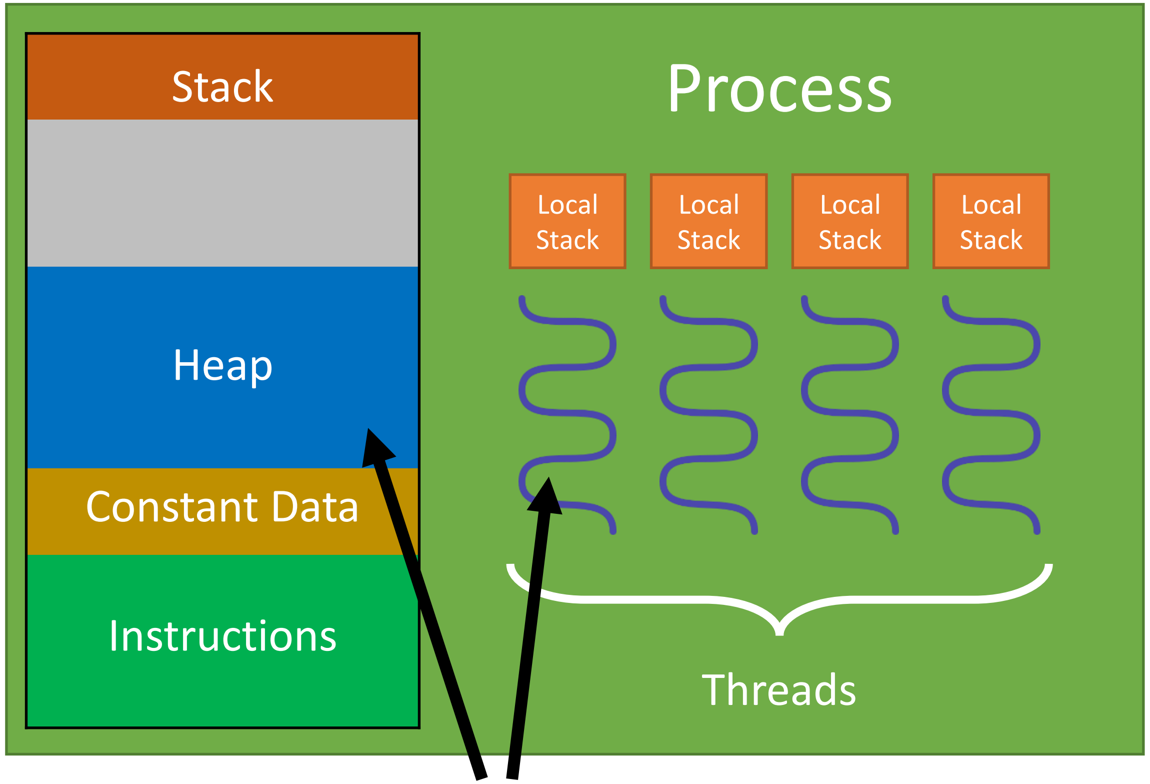
7/10


Stack
Heap
Constant Data
Instructions
Process
Local Stack
Local Stack
Local Stack
Local Stack
Threads


Memory
Stack
Heap
Constant Data
Instructions
Process
Local Stack
Local Stack
Local Stack
Local Stack
Threads

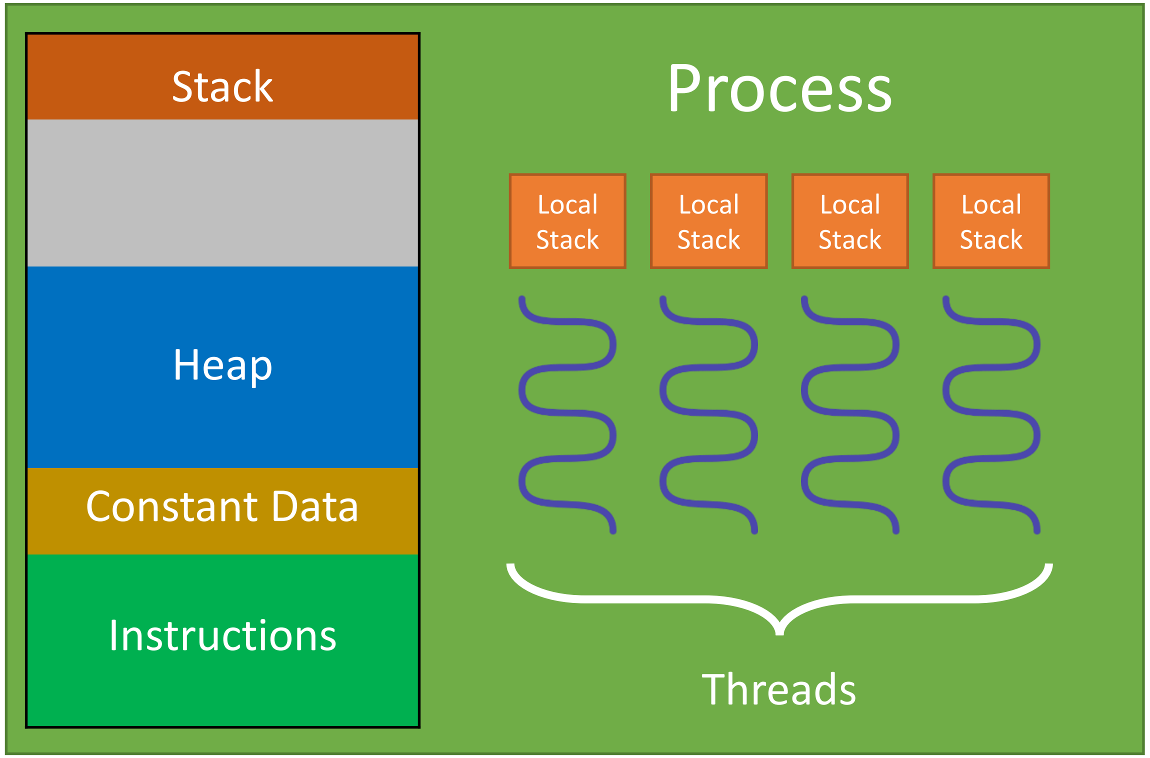


Local stack for each thread



Threads can access process memory

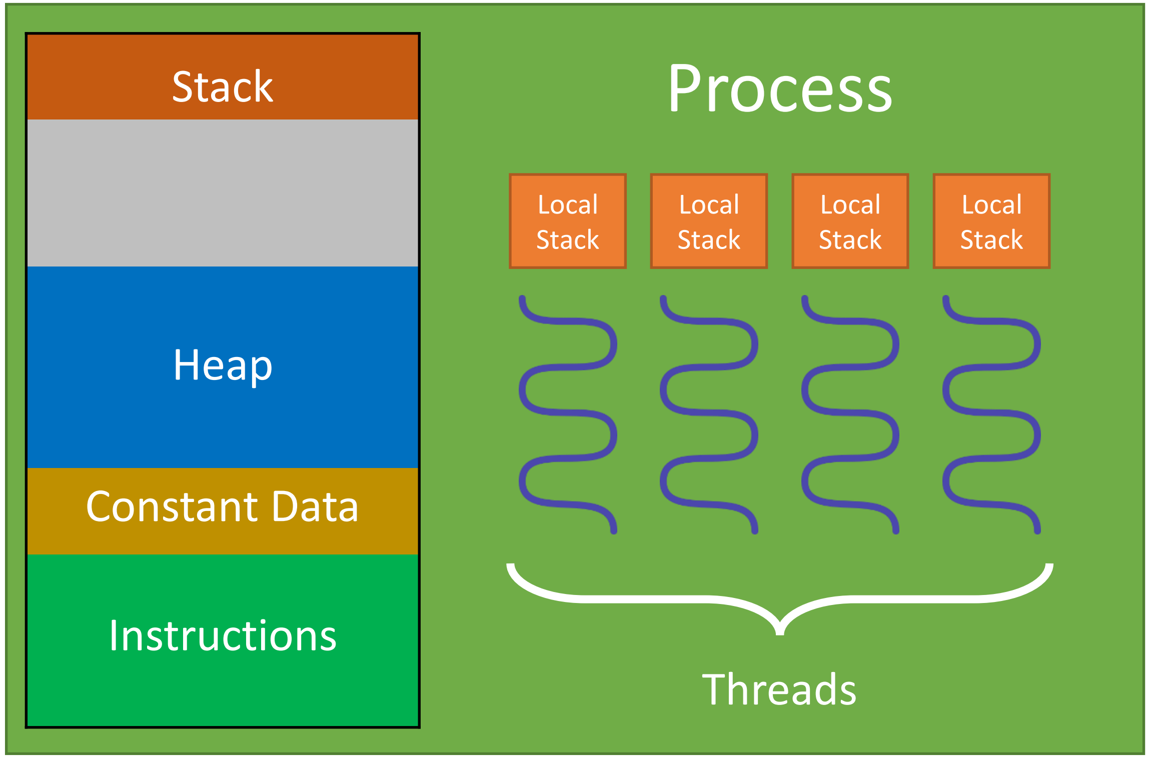
Multithreading Paradigm



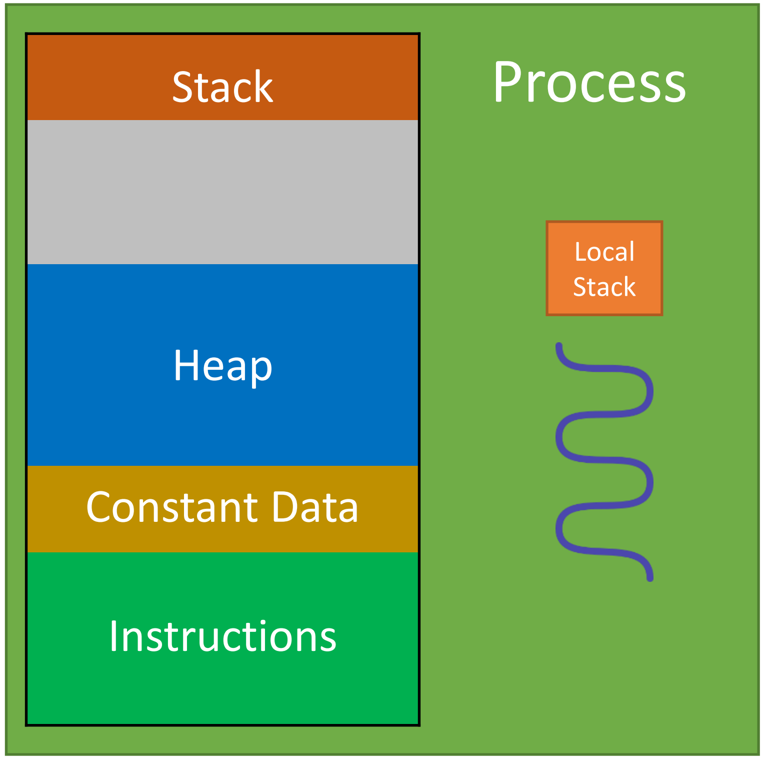
Limitations:

- Memory is limited to the memory of a single machine
- CPU cores are limited to the single machine
- Is not **scalable**

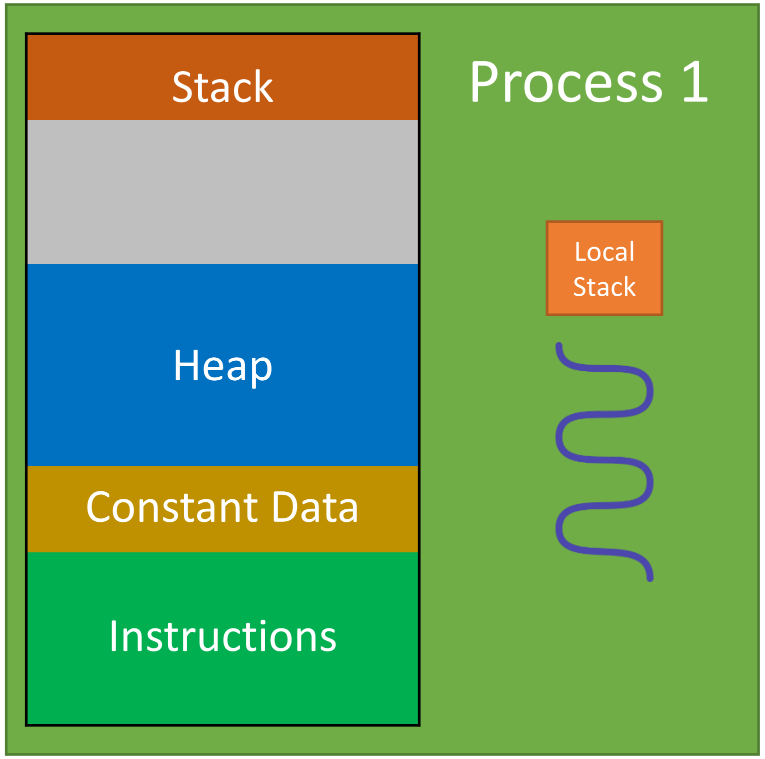
Multiprocessing Paradigm



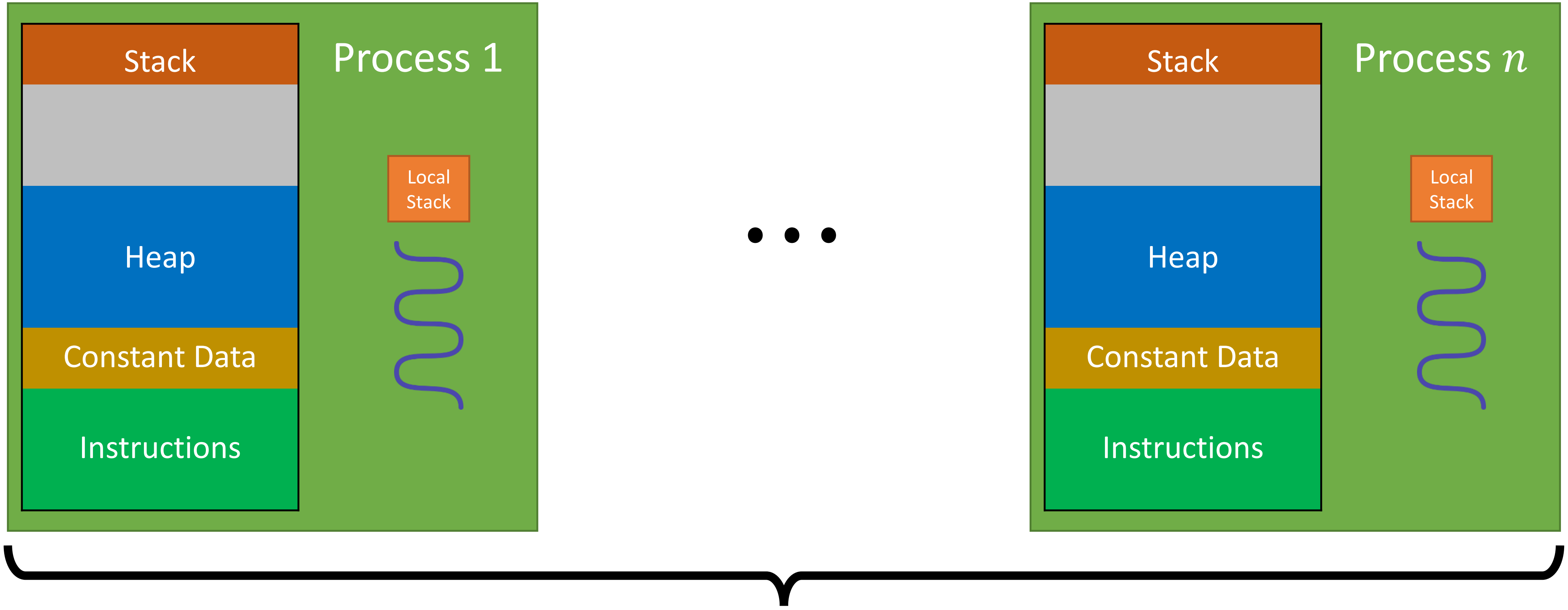
Multiprocessing Paradigm



Multiprocessing Paradigm



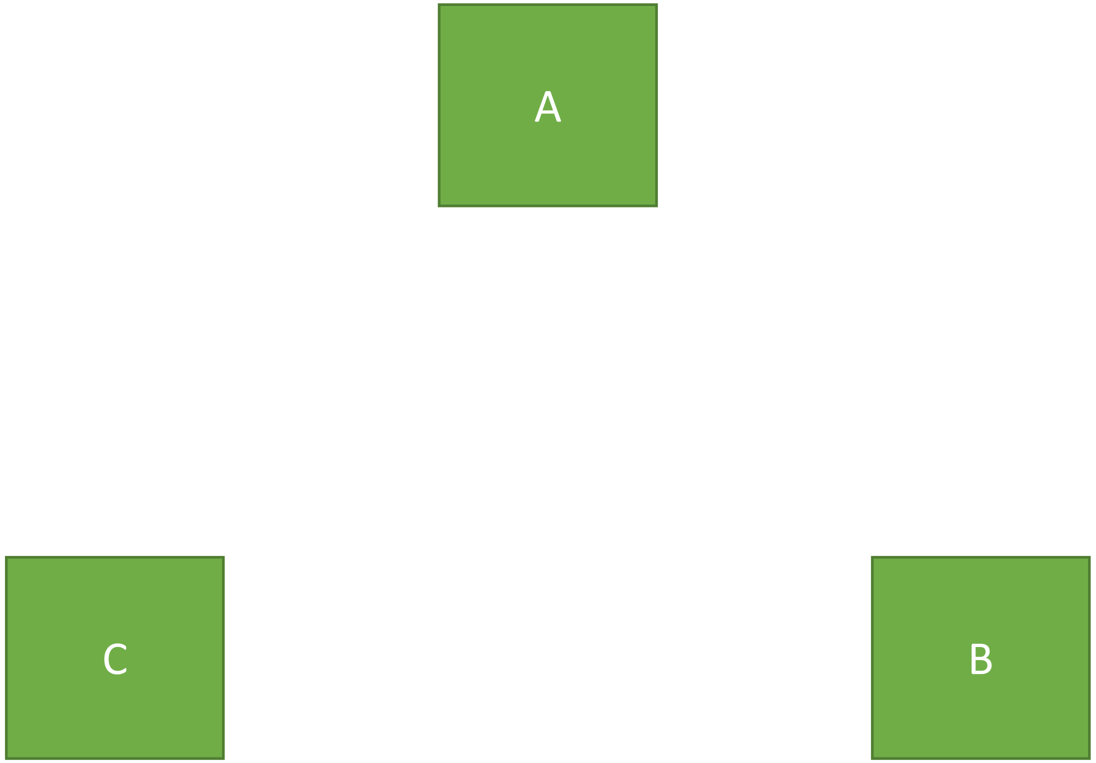
Multiprocessing Paradigm



Total processes: n

Multiprocessing Paradigm

Multiprocessing Paradigm

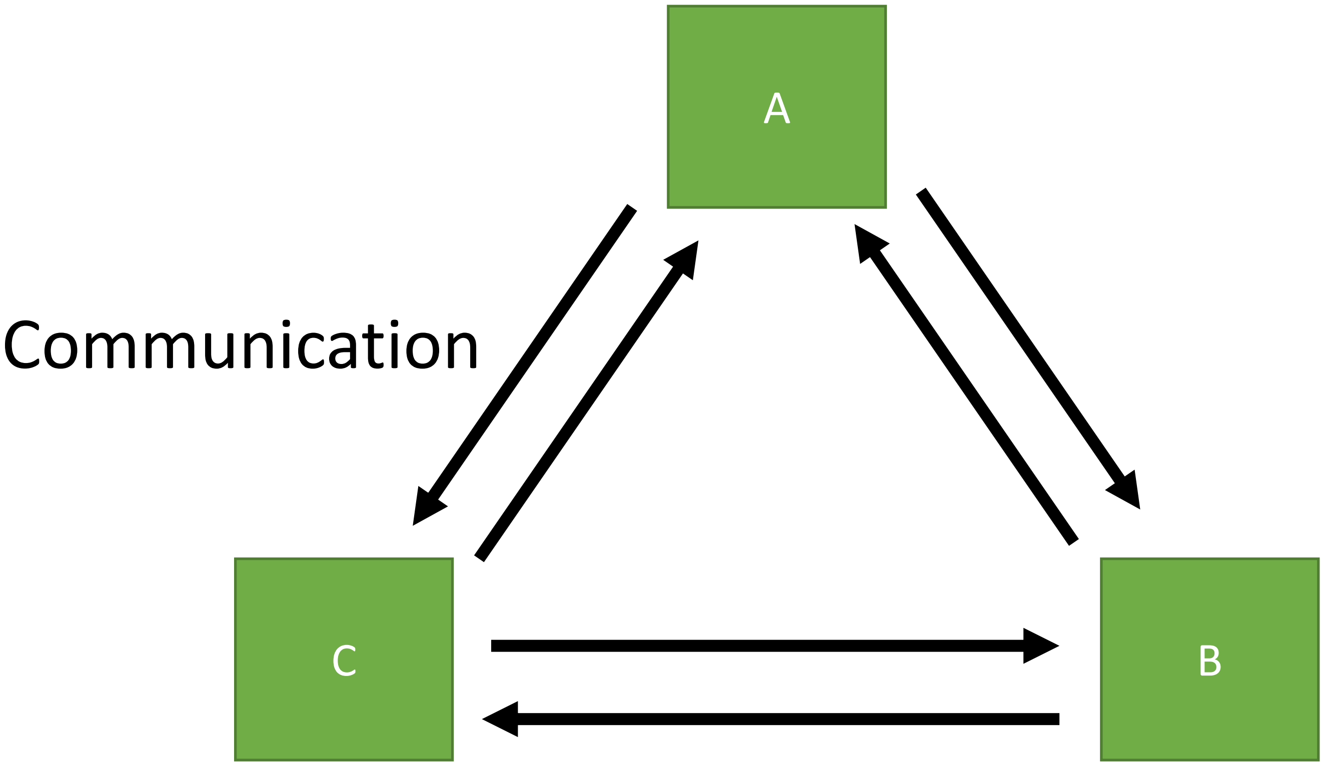


A

C

B

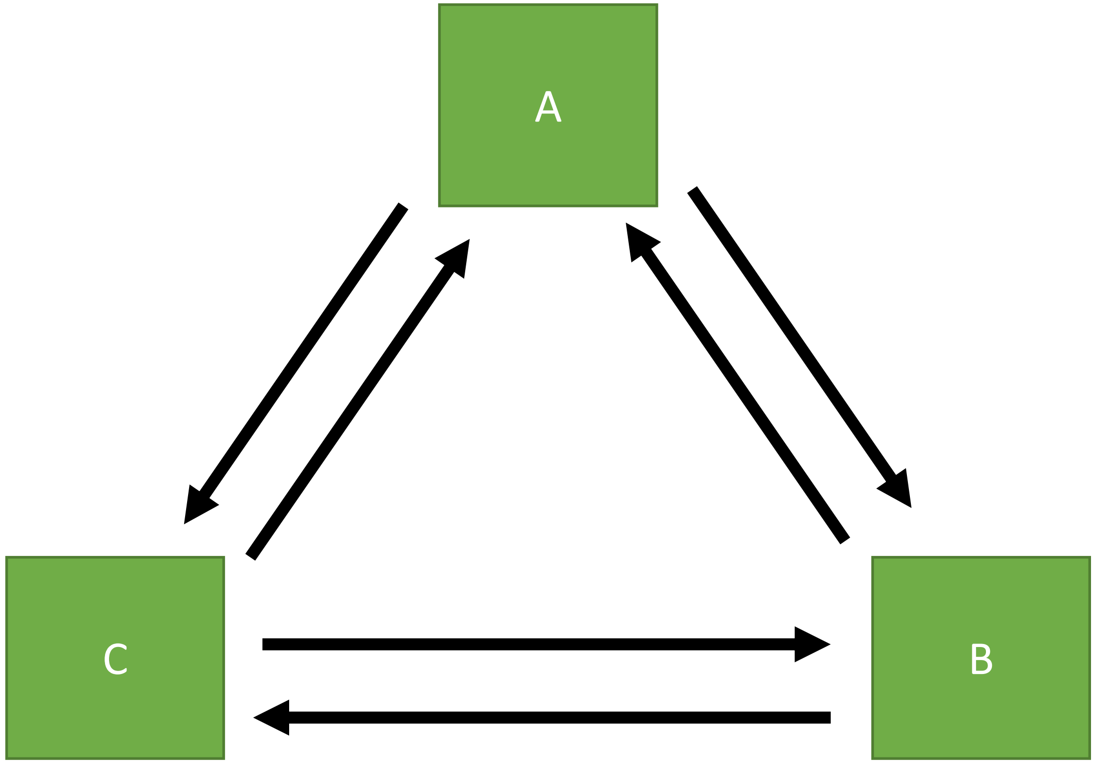
Multiprocessing Paradigm



Multiprocessing Paradigm

Communication:

- Abstract channels of communications (commands/data)
- Communication link can be over a socket (same machine) or over the network through TCP/IP
- Called **Message Passing**
- The **topology** of the network defines which processes can communicate
- Can also communicate via shared memory/shared storage



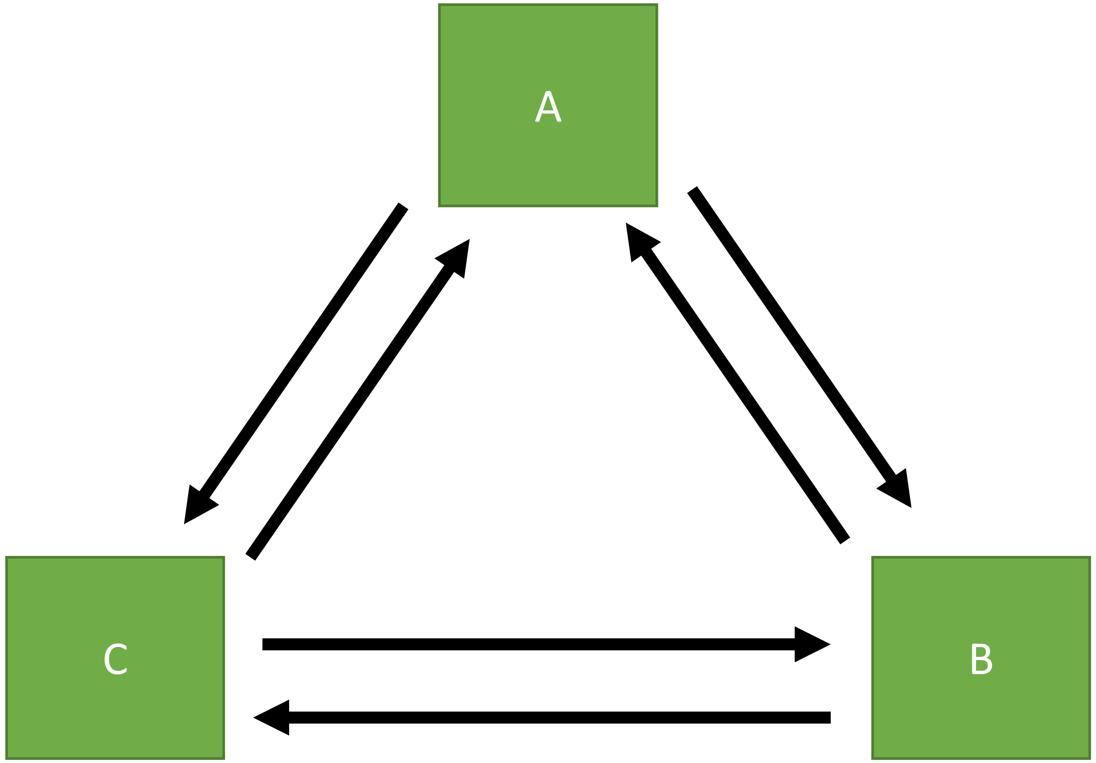
Multiprocessing Paradigm

Advantages:

- Using a message passing approach allows processes to run on **different machines** which are networked together
- A cluster of machines can have access to far more resources than any single machine – **scalability**

Disadvantages:

- Each process needs its own copy of functions and data to operate
- Communication between processes introduces a lot of **latency**
- Starting more processes has very high **latency**



Multiprocessing (MPI)

- The majority of languages use the **message passing interface (MPI)**
- **MPI** is a **standard** that has many implementations
- Some are specially developed for fast networking installed in supercomputing clusters (Infiniband etc)
- Julia wrapper using *MPI.jl*

Implementations:

- Open MPI
- MPICH (v3.1 or later)
- Intel MPI
- Microsoft MPI
- IBM Spectrum MPI
- MVAPICH
- Cray MPICH
- Fujitsu MPI
- HPE MPT/HMPT

Anatomy of an MPI program

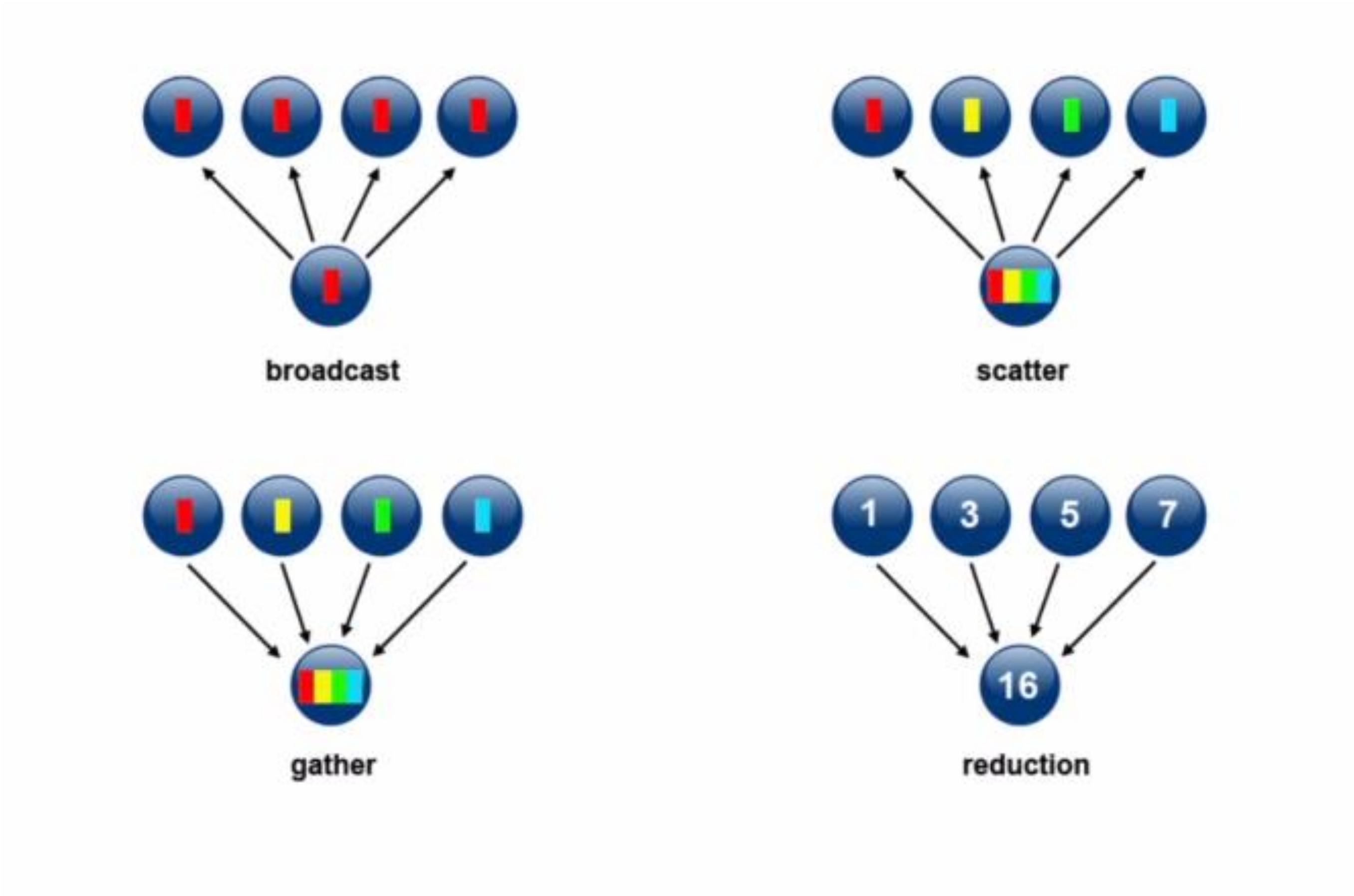
- Each process executes the **same** program
- Information about the topology is passed via environment variables
- **Barriers** are used to **synchronise** execution
- Can use MPI commands to send information to other processes

```
using MPI
MPI.Init()

comm = MPI.COMM_WORLD
comm_size = MPI.Comm_size(comm)
rank = MPI.Comm_rank(comm)

println("Hello world, I am $(rank) of $(comm_size)")
MPI.Barrier(comm)
```

Communication Modes (MPI)



Summary MPI

- MPI.jl is a **low-level** interface for multiprocessing
- On compute clusters with proprietary, fast, interconnect, MPI can be essential for achieving maximum performance
- A lot of developer investment to use MPI

- Julia has an inbuilt library for multiprocessing (or distributed) computing called *Distributed.jl* which provides a **high-level** interface for multiprocessing code

John T. Foster – Introduction to MPI and MPI.jl
https://www.youtube.com/watch?v=zPqX638rVG8

Distributed.jl

Live Example

Cluster Computing

Sulis – HPC Midlands+

Sulis has a total of **25,728** AMD EPYC compute cores

90 NVIDIA A100 GPUs

1.8 double-precision PFLOPS

Each compute node has 2 x AMD EPYC 7742 2.25 GHz **64**-core processors and **512GB** of RAM

Runs on Linux (**CentOS**)

Uses the **SLURM Scheduler**

Not Sulis

Connect to the Cluster

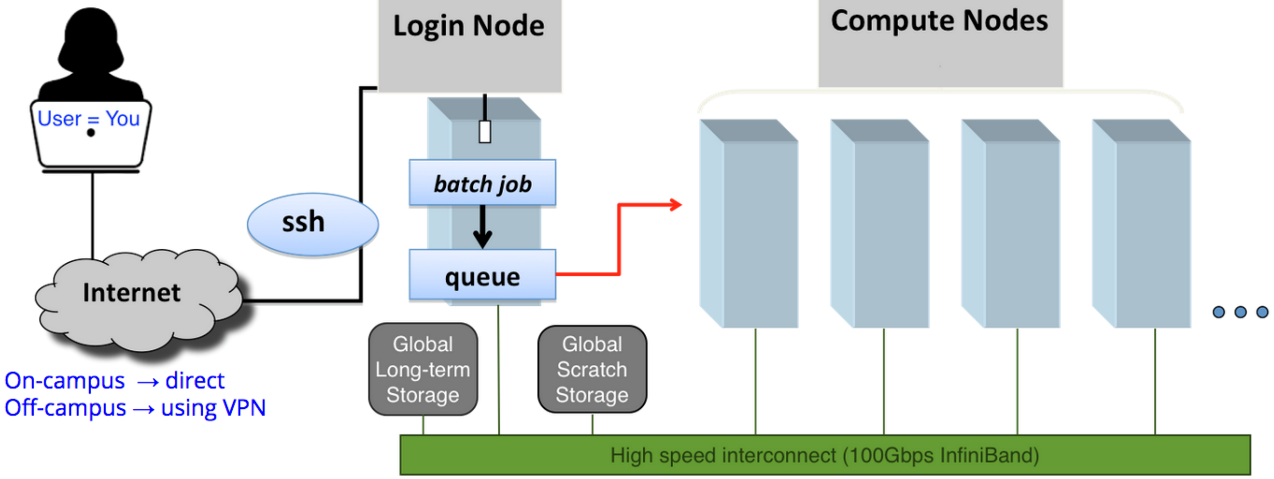
You can login to the cluster over the internet via an **SSH** tunnel

SSH is an encrypted way of communicating over a network, which provides the user a terminal (shell) on the remote machine

For security, most clusters will only allow access from a trusted range of IP addresses

You may need to connect to a VPN, and then SSH into the login node

Users **should not** run code on the login node, but instead only manage their files and submit jobs to run on the compute nodes



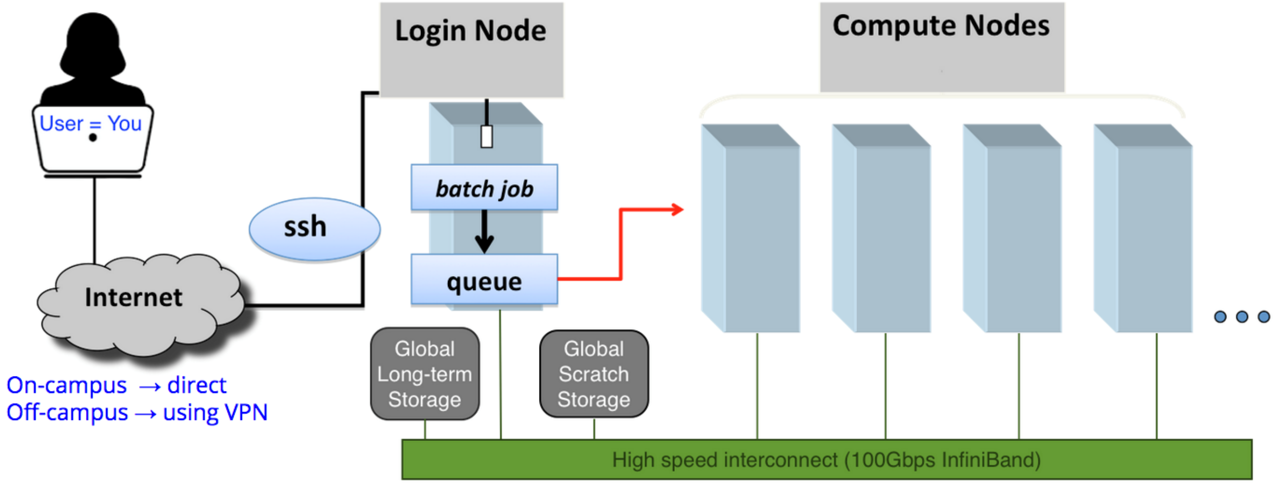
Submitting Jobs: SLURM

SLURM (Simple Linux for Utility Resource Management)

Standard for majority of HPCs

Acts a scheduling system to run many users jobs at the same time

Allocates isolated resources to a job so that no other users can access them



Environment Modules

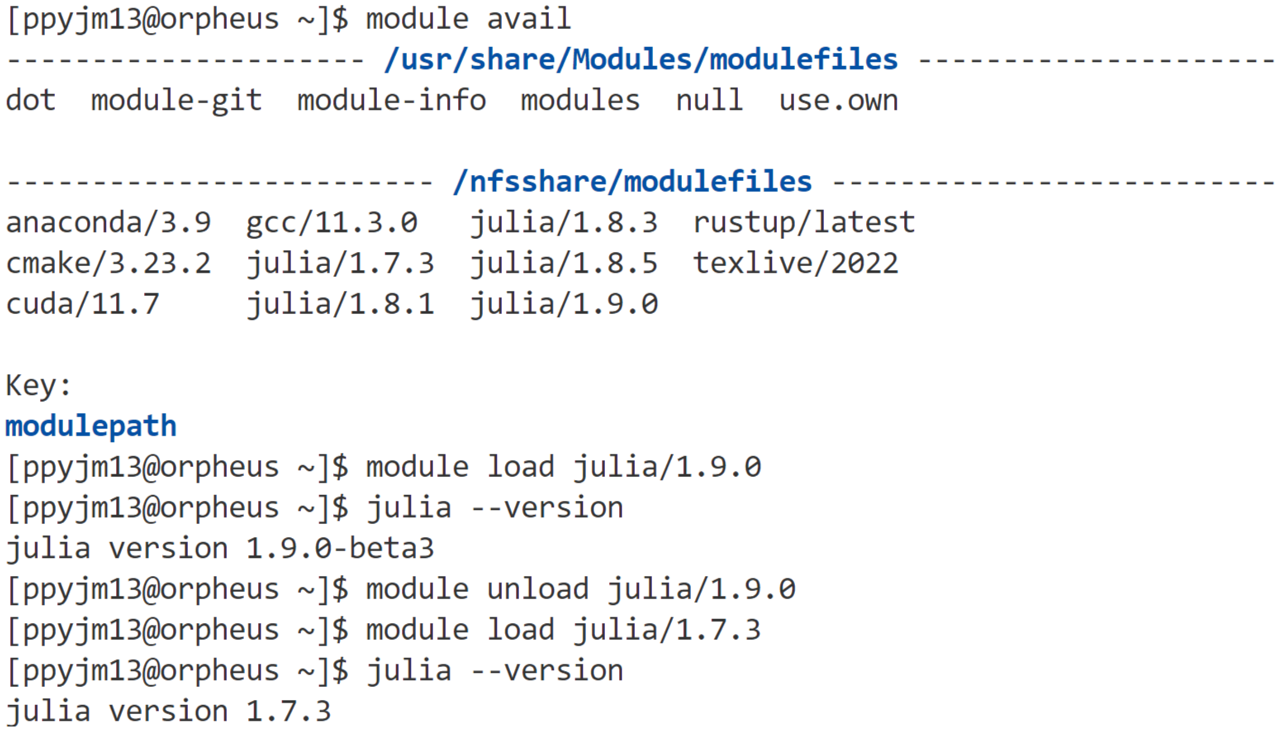
A HPC needs to manage a huge range of software with conflicting versions for many users

The admins install packages inside of environment modules

This allows the user to select which versions of software they want to use

Software is made available via **environment variables** which are cleared when opening a new terminal session

Modules can be loaded on each login by adding the module load command to the ".bashrc" file in your home directory



```
[ppyjm13@orpheus ~]$ module avail
--------------------- /usr/share/Modules/modulefiles ---------------------
dot  module-git  module-info  modules  null  use.own

------------------------- /nfsshare/modulefiles -------------------------
anaconda/3.9  gcc/11.3.0   julia/1.8.3  rustup/latest
cmake/3.23.2  julia/1.7.3  julia/1.8.5  texlive/2022
cuda/11.7     julia/1.8.1  julia/1.9.0

Key:
modulepath
[ppyjm13@orpheus ~]$ module load julia/1.9.0
[ppyjm13@orpheus ~]$ julia --version
julia version 1.9.0-beta3
[ppyjm13@orpheus ~]$ module unload julia/1.9.0
[ppyjm13@orpheus ~]$ module load julia/1.7.3
[ppyjm13@orpheus ~]$ julia --version
julia version 1.7.3
```

Working on a Cluster

Live Demonstration