# Introduction to GNU Parallel - Parallelizing Massive Individual Tasks

Feng Chen
HPC User Services
LSU HPC & LONI
sys-help@loni.org

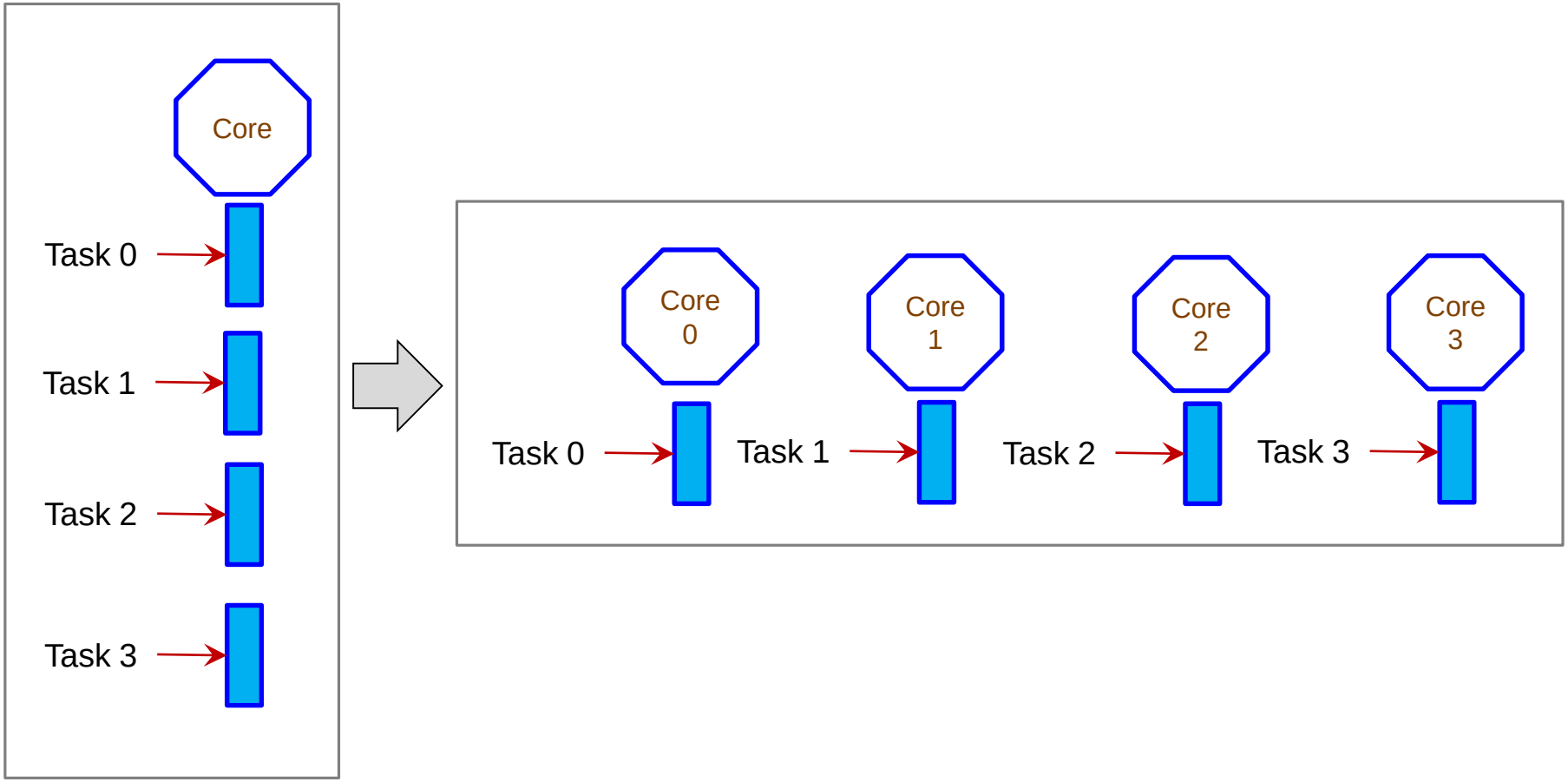
Louisiana State University
Baton Rouge
March 11, 2020

# Outline

- **Why GNU Parallel?**
- **Basic Usage of GNU Parallel**
  - GNU Parallel Syntax and Options
  - Introducing Running 3 Types of jobs:
    - Serial Tasks
      - ✓ Run each blast task in serial
    - Multi-Threaded Tasks
      - ✓ Run each blast task using 2 threads
    - Small MPI Jobs
      - ✓ Run each MPI Laplacian solver using 4 processes
- **Proper usage of GNU Parallel**
  - Memory consideration
  - Task granularity

# What do we want to accomplish?

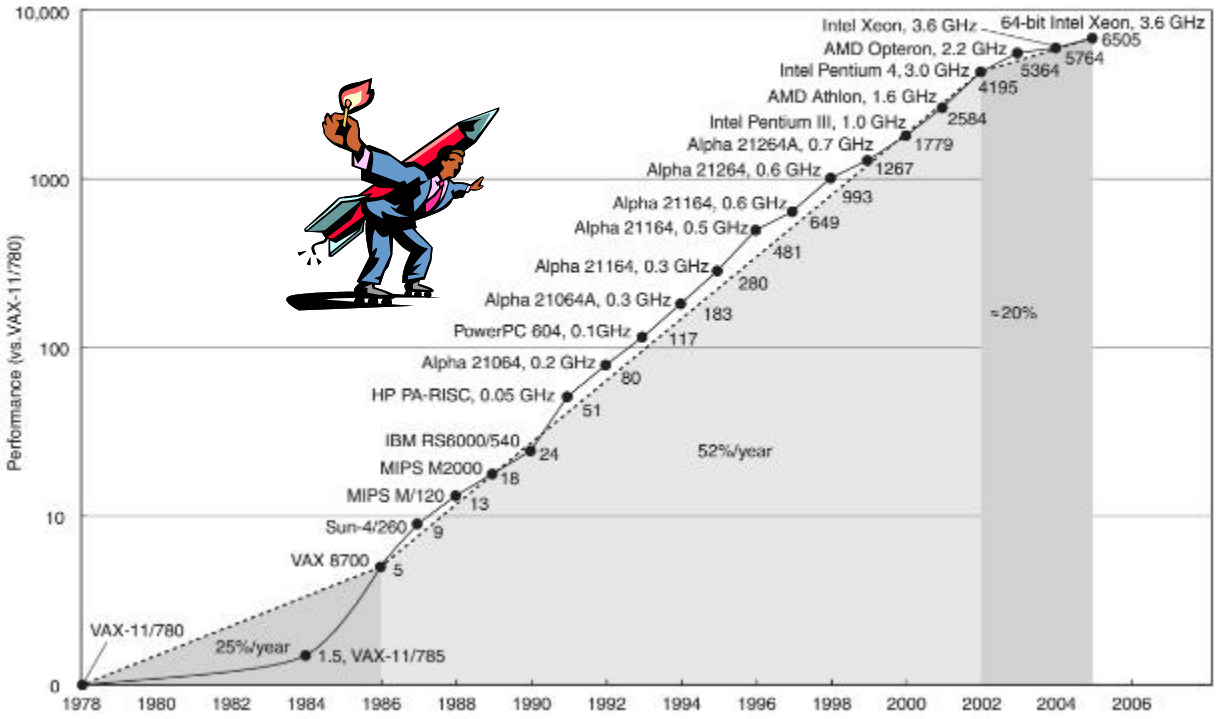
- **Parallize lots of serial independent tasks on a multi-core platform (compute nodes)**



# Changing Times

- **From 1986 - 2002, microprocessors were speeding like a rocket, increasing in performance an average of 50% per year.**
- **Since then, it's dropped to about 20% increase per year.**

## History of Processor Performance



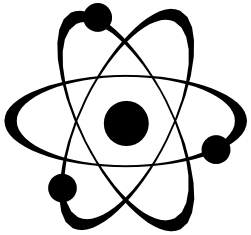

Limitation:

2 GHz Consumer
4 GHz Server

Source:
http://www.cs.columbia.edu/~sedwards/classes/2012/3827-spring/

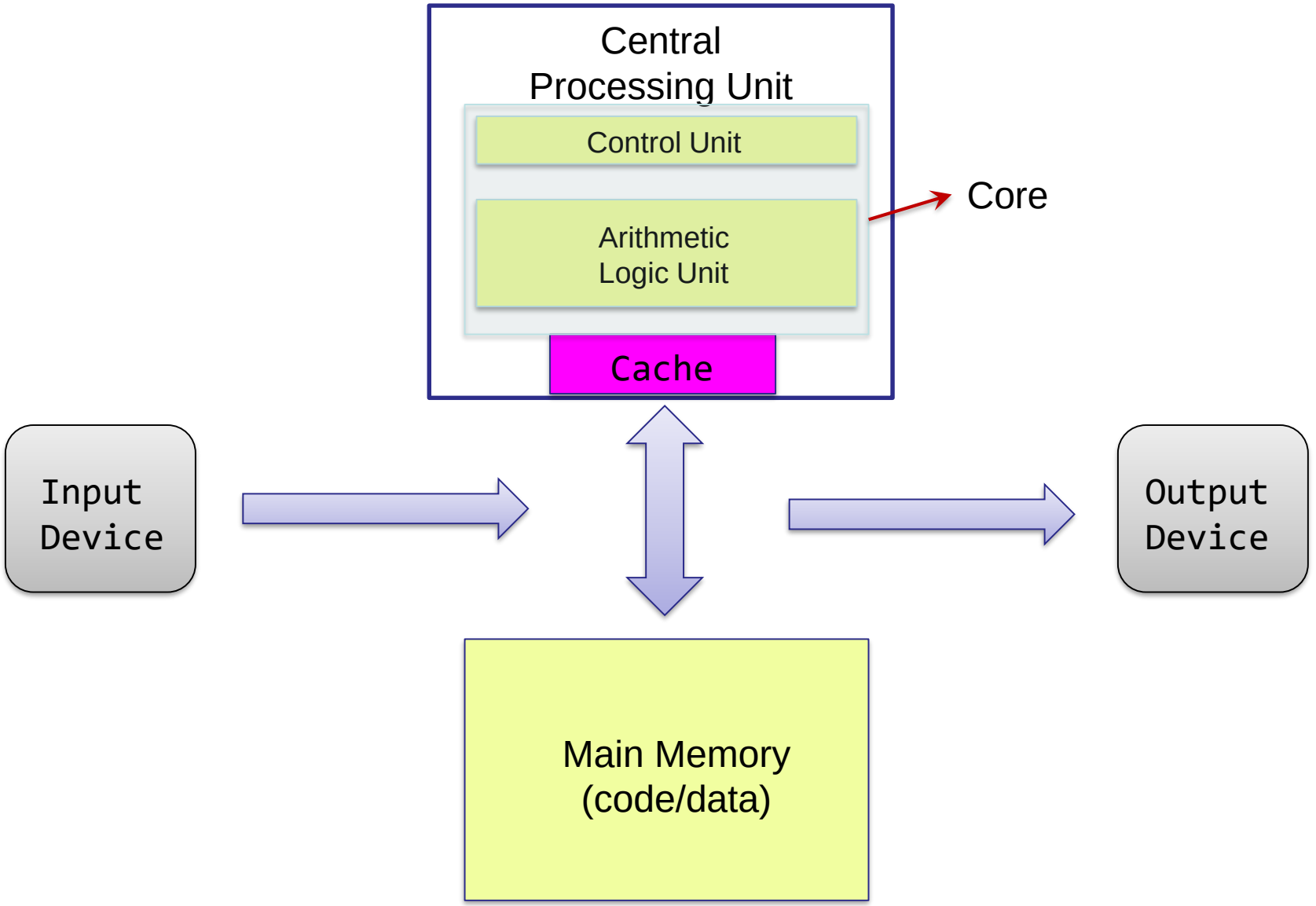
# A Little Physics Problem

- **Smaller transistors = faster processors.**
- **Faster processors = increased power consumption.**
- **Increased power consumption = increased heat.**
- **Increased heat = unreliable processors.**



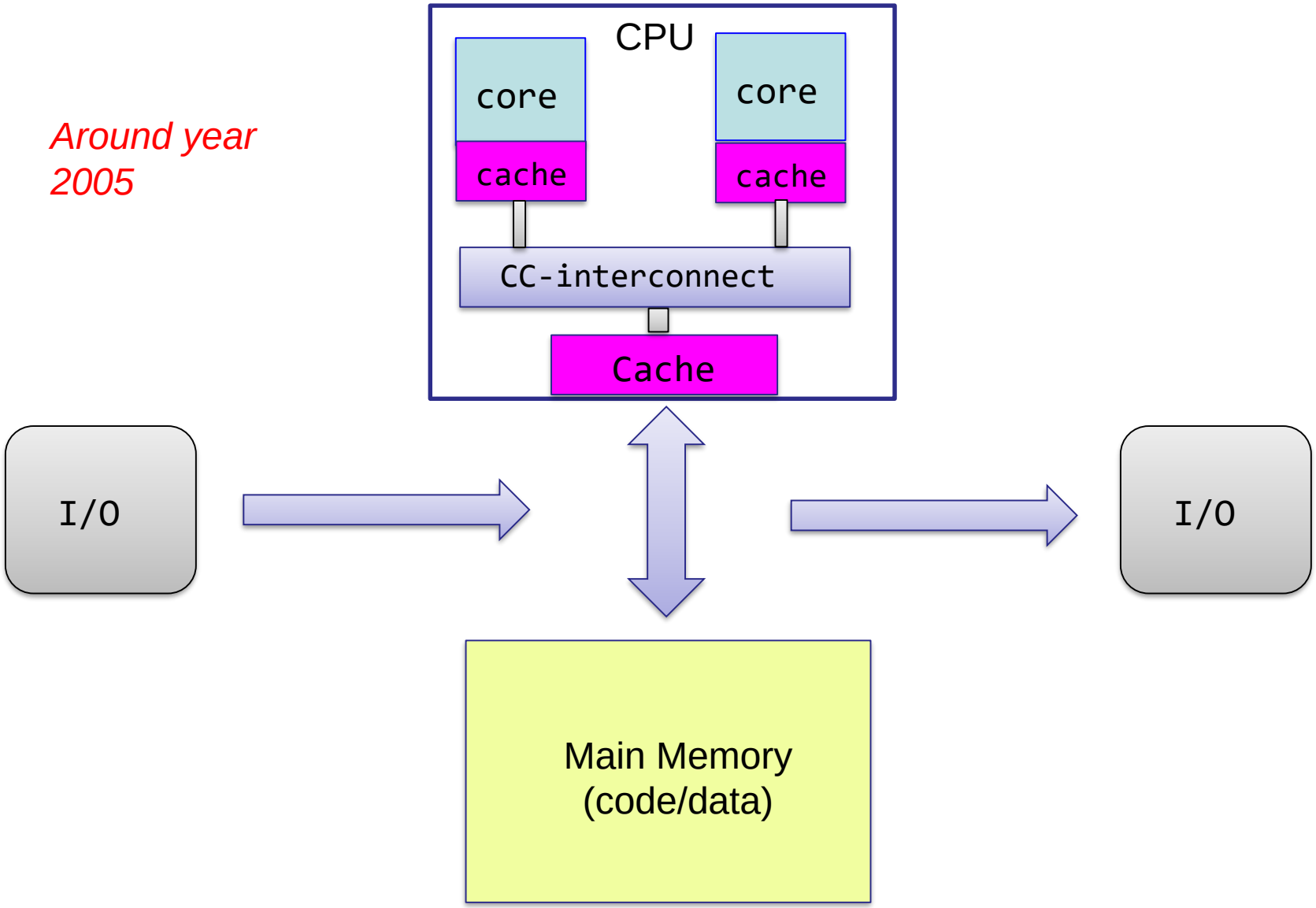
- **Solution:**
  - Move away from single-core systems to multicore processors.
  - “core” = central processing unit (CPU)
  - Introducing parallelism
    - *What if your problem is also not CPU dominant?*

# The von Neumann Architecture



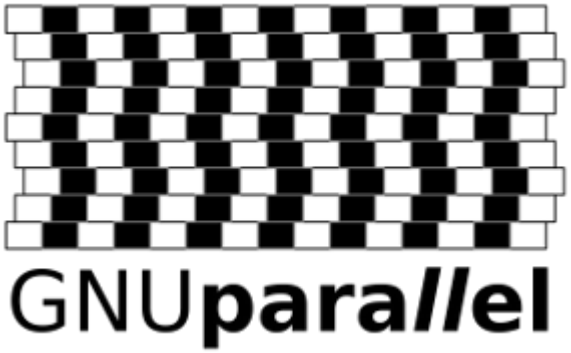
# The von Neumann Architecture

*Around year 2005*



# GNU Parallel

- **GNU parallel is a shell tool for executing jobs in parallel using one or more computers (*compute nodes*).**
- **A job can be a *single command or a small script* that has to be run for each of the lines in the input.**
- **The typical input is a *list of files*, a list of hosts, a list of users, a list of URLs, or a list of tables.**
- **See more at: https://www.gnu.org/software/parallel/**



# Adding GNU Parallel to Environment

- **On SuperMike2:**

```
[fchen14@mike421 ~]$ module av gnuparallel # check available version
----------- /usr/local/packages/Modules/3.2.10/modulefiles/apps
gnuparallel/20180222/INTEL-18.0.0
[fchen14@mike1 ~]$ module load gnuparallel/20180222/INTEL-18.0.0 # load
gnuparallel to your environment
[fchen14@mike1 ~]$ which parallel
/usr/local/packages/gnuparallel/20180222/INTEL-18.0.0/bin/parallel
[fchen14@mike421 ~]$ parallel --version
GNU parallel 20180222
...
```

- **Or add the below line to `~/.modules`:**

```
module load gnuparallel/20180222/INTEL-18.0.0
```

*Introduction to GNU Parallel*

# GNU Parallel Syntax

# GNU Parallel Syntax

- **Reading commands to be run in parallel from an input file:**

```
parallel [OPTIONS] < CMDFILE
```

- **Reading command arguments on the command line:**

```
parallel [OPTIONS] COMMAND [ARGUMENTS] ::: ARGLIST
```

- **Reading command arguments from an input file:**

```
parallel [OPTIONS] COMMAND [ARGUMENTS] :::: ARGFILE
```

# ARGLIST from command line

- **parallel [OPTIONS] COMMAND [ARGUMENTS] ::: ARGLIST**
- **Examples:**

```
[fchen14@mike421 ~]$ parallel echo ::: A B C
A
B
C
[fchen14@mike421 ~]$ parallel echo ::: `seq 1 3`
1
2
3
[fchen14@mike421 ~]$ parallel echo ::: {A..Z}
A
B
...
Z
[fchen14@mike421 test]$ ls -1 | parallel echo
2013-06-18.tgz
backups.sh
bigmem_test.pbs
...
```

# ARGLIST from file

- **parallel [OPTIONS] COMMAND [ARGUMENTS] ::: ARGFILE**

```
[fchen14@mike421 blast]$ cd /project/fchen14/gpar/blast
[fchen14@mike421 blast]$ cat input.lst | head
data/test10.faa
data/test11.faa
...
[fchen14@mike421 blast]$ head input.lst -n 5 | parallel echo
data/test10.faa
data/test11.faa
data/test12.faa
data/test13.faa
data/test14.faa
[fchen14@mike421 blast]$ parallel echo :::: input.lst
data/test10.faa
data/test11.faa
data/test12.faa
...
```

# Replacement Strings

- '{}' returns a full line read from the input source.

```
[fchen14@mike421 blast]$ parallel echo {} ::: data/test1.faa
data/test1.faa
```

- '{/}' removes everything up to and including the last forward slash:

```
[fchen14@mike421 blast]$ parallel echo {/} ::: data/test1.faa
test1.faa
```

- '{//}' returns the directory name of input line.

```
[fchen14@mike421 blast]$ parallel echo {//} ::: data/test1.faa
data
```

- '{.}' removes any filename extension:

```
[fchen14@mike421 blast]$ parallel echo {.} ::: data/test1.faa
data/test1
```

- '{/.}' returns the basename of the input line without extension. It is a combination of {/} and {.}:

```
[fchen14@mike421 blast]$ parallel echo {/.} ::: data/test1.faa
test1
```

- See "man parallel" for more detailed explanation.

# Replacement String Example

- **Print the full path of the input file, and then print the desired output file name, e.g.:**
  - Input file: `data/test1.faa`
  - Output file name: `output/test1.out`

```
# Process data/test1.faa and send result to output/test1.out
$ parallel echo {} output/{/.}.out ::: data/test1.faa
data/test1.faa output/test1.out
```

# Parallelize Job Script

- **GNU parallel is often called as this:**

```
cat input_file | parallel command
parallel command ::: foo bar
```

- **If command is a script, parallel can be combined into a single file so this will run the script in parallel:**

```
parallel [OPTIONS] script [ARGUMENTS] ::: ARGLIST
```

  - or

```
parallel [OPTIONS] script [ARGUMENTS] :::: ARGFILE
```

- **See next slide for example...**

# Parallize Script Example

- **This is the script we want to parallize "cmd_ex.sh":**

```
#!/bin/bash
# print the input, on which host, which working directory
echo "This script uses input: $1 on $HOSTNAME:$PWD"
```

- **Parallize the script using ARGLIST from command line:**

```
[fchen14@mike421 misc]$ parallel --wd /project/fchen14/gpar/misc ./cmd_ex.sh ::: A B C
This script uses input: A on mike421:/project/fchen14/gpar/misc
This script uses input: B on mike421:/project/fchen14/gpar/misc
This script uses input: C on mike421:/project/fchen14/gpar/misc
```

- **Parallize the script using ARGFILE:**

```
[fchen14@mike421 misc]$ cat argfile
A
B
C
[fchen14@mike421 misc]$ parallel --wd /project/fchen14/gpar/misc ./cmd_ex.sh :::: argfile
This script uses input: A on mike421:/project/fchen14/gpar/misc
This script uses input: B on mike421:/project/fchen14/gpar/misc
This script uses input: C on mike421:/project/fchen14/gpar/misc
```

- **Can parallize Python/Perl scripts, see "man parallel" for details**

# Common OPTIONS --jobs (-j)

- **--jobs N (-j N)**
  - Number of jobslots on each machine (node). Run up to N jobs in parallel. 0 means as many as possible. Default is 100% which will run one job per CPU core on each machine.
  - On HPC/LONI clusters, **N** is number of jobslots per node.
  - Make sure you use GNU Parallel version **>=20161022** to avoid a "Max jobs to run" bug

```
[fchen14@mike421 test]$ parallel --version
GNU parallel 20161022
...
```

- **-j +N**
  - Add N to the number of CPU cores. Run this many jobs in parallel.
- **-j -N**
  - Subtract N from the number of CPU cores. Run this many jobs in parallel. If the evaluated number is less than 1 then 1 will be used.

# Common OPTIONS `--slf`

- **`--slf filename (--sshloginfile filename)`**
  - File with sshlogins. The file consists of sshlogins on separate lines. Empty lines and lines starting with '#' are ignored.
  - Look at "`man parallel`" for more detailed explanation.
  - A typical example on HPC/LONI clusters while running batch jobs:

    `--slf $PBS_NODEFILE`
  - Recall what is inside `$PBS_NODEFILE`?

```
[fchen14@mike421 laplace]$ cat $PBS_NODEFILE
mike421
mike421
...
mike421
mike429
mike429
...
mike429
```

**16 repeats (cores) on mike421**

**16 repeats (cores) on mike429**

# Common OPTIONS `--wd`

- **`--wd mydir (--workdir mydir)`**
  - Designate the working directory of your commands.
  - A typical value can be `$PBS_O_WORDIR`

# Common OPTIONS `--progress`

- **`--progress`**
  - Show progress of computations.
  - List the computers involved in the task with number of CPU cores detected and the max number of jobs to run.
  - After that show progress for each node: number of running jobs, number of completed jobs, and percentage of all jobs done by this computer.
  - Example:

```
[fchen14@mike421 ~]$ parallel --progress echo ::: A B C

Computers / CPU cores / Max jobs to run
1:local / 16 / 3

Computer:jobs running/jobs completed/%of started jobs/Average seconds to complete
local:3/0/100%/0.0s A
local:2/1/100%/1.0s B
local:1/2/100%/0.5s C
local:0/3/100%/0.3s
```

- **See also `--bar`**

# Common OPTIONS --joblog

- **--joblog logfile**
  - Creates a record for each completed subjob to be written to LOGFILE, with info on how long they took, their exit status, etc.
  - Can be used to identify failed jobs, e.g.:

```
[fchen14@mike421 misc]$ parallel --joblog logfile exit ::: 1 2 0 0
[fchen14@mike421 misc]$ cat logfile
Seq     Host    Starttime       JobRuntime      Send    Receive Exitval Signal  Command
1       :       1477514132.358        0.019     0       0       1       0       exit 1
2       :       1477514132.375        0.003     0       0       2       0       exit 2
3       :       1477514132.376        0.002     0       0       0       0       exit 0
4       :       1477514132.377        0.003     0       0       0       0       exit 0
```

# Common OPTIONS `--timeout`

- **`--timeout secs`**
  - Time out for command. If the command runs for longer than secs seconds it will get killed.
  - If secs is followed by a % then the timeout will dynamically be computed as a percentage of the median average runtime. Only values > 100% will make sense.

  - Useful if you know the command has failed if it runs longer than a threshold.

*Introduction to GNU Parallel*

# Serial Jobs Example

# Distribute Serial Jobs - Run blast

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -l walltime=1:00:00
#PBS -A hpc_hpcadmin3
#PBS -q workq
#PBS -N gpl_ser_blast
#PBS -o gpl_ser_blast.out
#PBS -e gpl_ser_blast.err
JOBS_PER_NODE=16
export WDIR=/project/$USER/distribution_workload/blast
cd $WDIR
# create a folder output
mkdir -p output
# the parallel command
parallel --progress \              # shows the progress
         --joblog logfile \        # specify a job logfile
         -j $JOBS_PER_NODE \       # specify the jobs per node
         --slf $PBS_NODEFILE \     # distribute workload among allocated nodes
         --workdir $WDIR \         # specify the working directory of the script
         $PBS_O_WORKDIR/cmd_ser_blast.sh {} {/.} :::: input.lst
         # script_to_parallize input output :::: ARGFILE
```

```
[fchen14@mike421 blast]$ head input.lst
data/test10.faa
data/test11.faa
data/test12.faa
data/test13.faa
data/test14.faa
data/test15.faa
...
```

# The script to Run blast

```
#!/bin/bash
FILE=$(eval echo $1) # process the input $1 when there is bash variable
echo "using input: $FILE"
TIC=`date +%s.%N`
blastp -query $FILE -db db/img_v400_PROT.00 -out output/$2.out \
       -outfmt 7 -max_target_seqs 100 -num_threads 1
TOC=`date +%s.%N`
J1_TIME=`echo "$TOC - $TIC" | bc -l`
echo "This serrun took=$J1_TIME sec using $FILE on $HOSTNAME"
```

$1 from {}

$2 from {/.}

# Multi-Threaded Example

# Distribute Multi-Threaded Jobs - blast

- **Distribute Multi-Threaded jobs is very similar to the pure serial job example, the only difference is** `JOBS_PER_NODE`**…**
  - `JOBS_PER_NODE=CPU_CORES_PER_NODE / NUM_THREADS_PER_JOB`
- **If each job uses 2 threads, each node on SuperMike2 has 16 cores, then**
  - `JOBS_PER_NODE=16/2=8`
- **PBS script (`#PBS comments` omitted):**

```
JOBS_PER_NODE=8
export WDIR=/project/$USER/distribution_workload/blast
cd $WDIR
mkdir -p output
NTHREADS=2
parallel --progress \
         -j $JOBS_PER_NODE \
         --slf $PBS_NODEFILE \
         --workdir $WDIR \
         $PBS_O_WORKDIR/cmd_mt_blast.sh {} {/.} $NTHREADS \
         :::: input.lst
```

changes needed compared to serial version

# The script to Run Multi-Threaded blast

```
#!/bin/bash
# This is the script we want to parallize and distribute
echo "using input: $1, output $2.out"
FILE=$(eval echo $1)
echo "using input: $FILE"
TIC=`date +%s.%N`
blastp -query $FILE \
       -db db/img_v400_PROT.00 \
       -out output/$2.out  \
       -outfmt 7 -max_target_seqs 100 \
       -num_threads $3
TOC=`date +%s.%N`
J1_TIME=`echo "$TOC - $TIC" | bc -l`
echo "This serrun took=$J1_TIME sec using $FILE with $3 threads on $HOSTNAME"
```

$1 from {}

$2 from {/.}

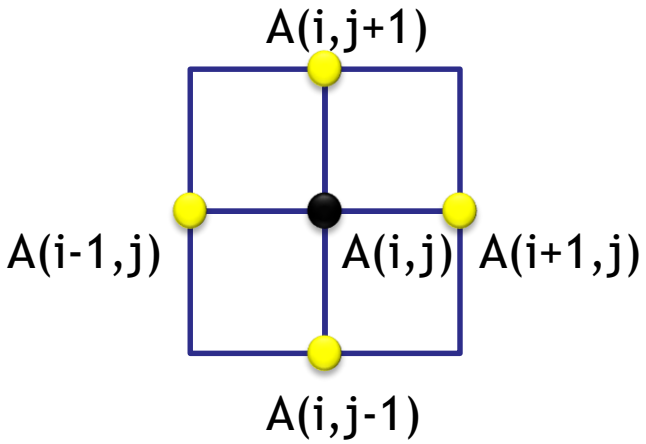
$3 from $NTHREADS, the only change compared to serial

# Multi-Process (MPI) Example

# Distribute MPI Jobs - Laplace Solver

- **This section describes how to distribute small MPI jobs.**
- **Example problem - Laplacian Solver**
  - Solves a 2D Laplacian equation on a 2D grid
  - Use 4096x4096 2D grid, run 2000 iterations

$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$



# Graphical representation for Jacobi iteration

Array: told

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

$t_{1,1} = 0.25 \times (1.0 + 1.0 + 0.0 + 0.0) = 0.5$

$dt = \max(t_{i,j} - told_{i,j}) = 0.5 - 0 = 0.5$

Array: t (tnew)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | |
| 1.0 | 0.5 | 0.375 | 0.344 | 0.336 | 0.334 | 0.333 | 0.0 |
| 1.0 | 0.375 | 0.188 | 0.133 | 0.117 | 0.113 | 0.112 | 0.0 |
| 1.0 | 0.344 | 0.133 | 0.066 | 0.046 | 0.04 | 0.038 | 0.0 |
| 1.0 | 0.336 | 0.117 | 0.046 | 0.023 | 0.016 | 0.013 | 0.0 |
| | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

# PBS Script for Distributing MPI Jobs

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -l walltime=1:00:00
#PBS -A hpc_hpcadmin3
#PBS -q workq
#PBS -N gpl_mpi
```

```
[fchen14@mike421 laplace]$ cat input.lst
/project/$USER/distribution_workload/laplace/input/input1
/project/$USER/distribution_workload/laplace/input/input2
/project/$USER/distribution_workload/laplace/input/input3
...
[fchen14@mike421 laplace]$ cat \
/project/$USER/distribution_workload/laplace/input/input1
4096 4096 2 2 0.08 20000 0 0
```

```
JOBS_PER_NODE=4 # uses 4 jobs per node, on SuperMike2: 16/4=4
echo "JOBS_PER_NODE="$JOBS_PER_NODE
NPROCS=4        # uses 4 mpi processes per MPI job
WDIR=/project/$USER/distribution_workload/laplace
cd $WDIR
parallel --progress \
         -j $JOBS_PER_NODE \
         --slf $PBS_NODEFILE \
         --workdir $WDIR \
         $PBS_O_WORKDIR/cmd_mpi.sh {} $NPROCS :::: input.lst
         # script_name input num_mpi_process
```

# The Script to Run MPI Laplacian Solver

```
#!/bin/bash
echo "using input: $1"
TIC=`date +%s.%N`
## get an input from $1 (input.lst)
FILE=$(eval echo $1)
param=`cat ${FILE}`
echo param=$param
# get number of mpi process from $2 ($NPROCS)
mpirun -np $2 ./lap_mpi $param
TOC=`date +%s.%N`
J1_TIME=`echo "$TOC - $TIC" | bc -l`
echo "This mpirun took=$J1_TIME sec on $HOSTNAME"
```

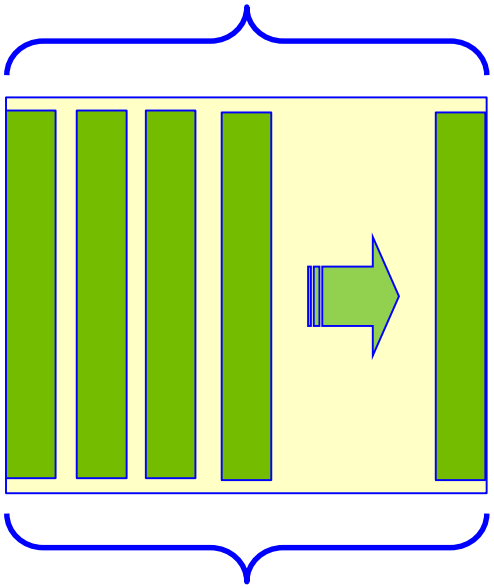
*Distributed Workload with GNU Parallel*

# Proper usage of GNU Parallel

# Memory Consideration

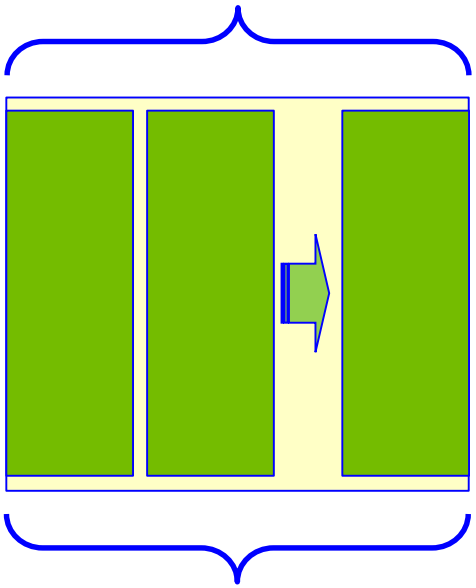
- **Relationship between node memory and cores**
  - Rule of Thumb: cannot exceed the available memory on a node

Available node memory 32 GB



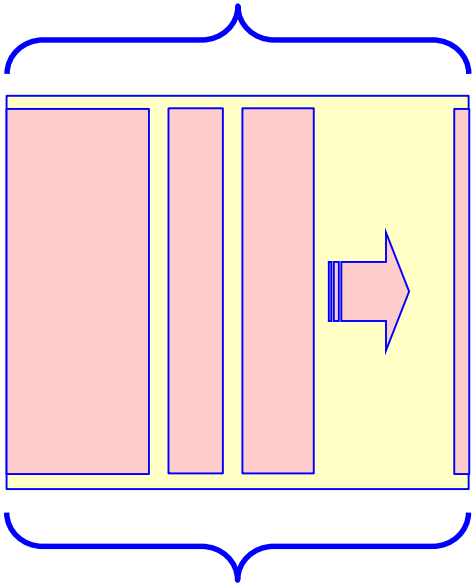
2 GB mem per task,
How many tasks per node?

Available node memory 32 GB



6 GB mem per task,
How many tasks per node?

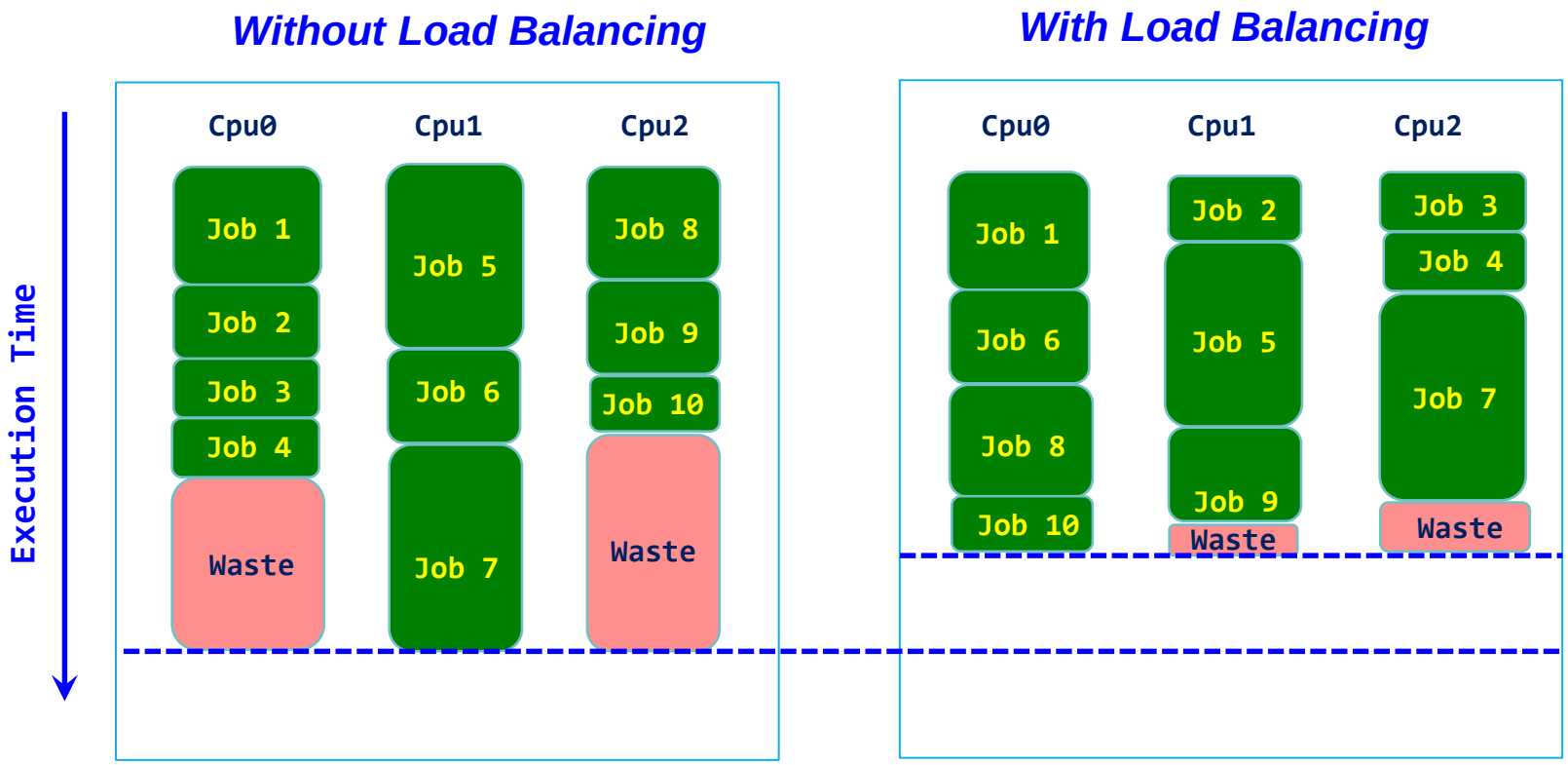
Available node memory 32 GB



Avoid this situation, hard to calculate/predict memory usage

# Load Balancing in GNU Parallel

- **GNU Parallel spawns the next job when one finishes - keeping the CPUs active and thus saving time.**

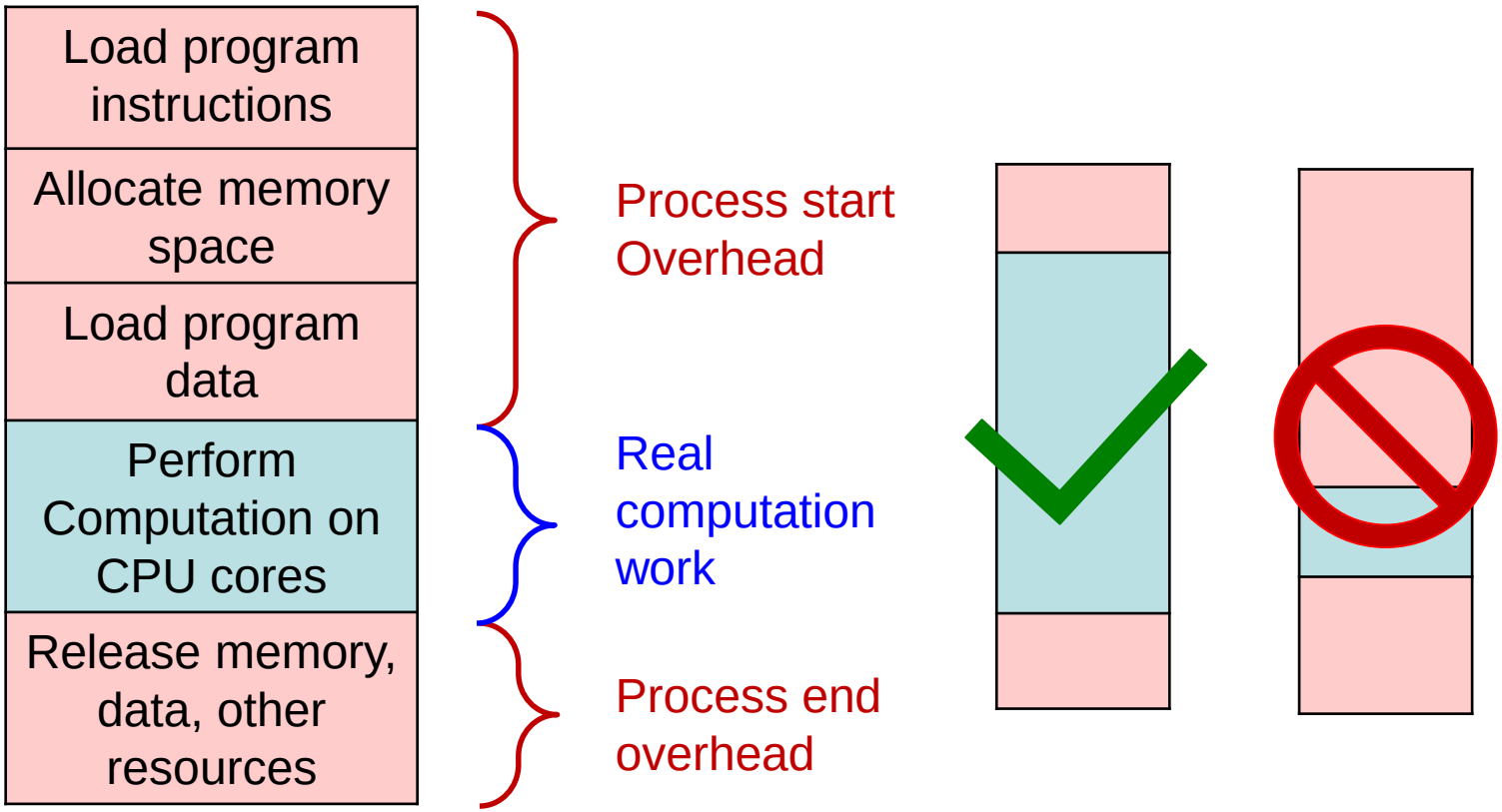


# Task Granularity

- **In parallel computing, granularity (or grain size) of a task is a measure of the amount of work (or computation) which is performed by that task.**
- **Impact of granularity on performance**
  - Using fine grains or small tasks results in more parallelism and hence increases the speedup. However, synchronization overhead, scheduling strategies etc. can negatively impact the performance of fine-grained tasks.
  - Simply increasing parallelism alone cannot give the best performance.
  - In order to reduce the communication overhead, granularity can be increased. Coarse grained tasks have less communication overhead but they often cause load imbalance. Hence optimal performance is achieved between the two extremes of fine-grained and coarse-grained parallelism.

Ref: https://en.wikipedia.org/wiki/Granularity_(parallel_computing)

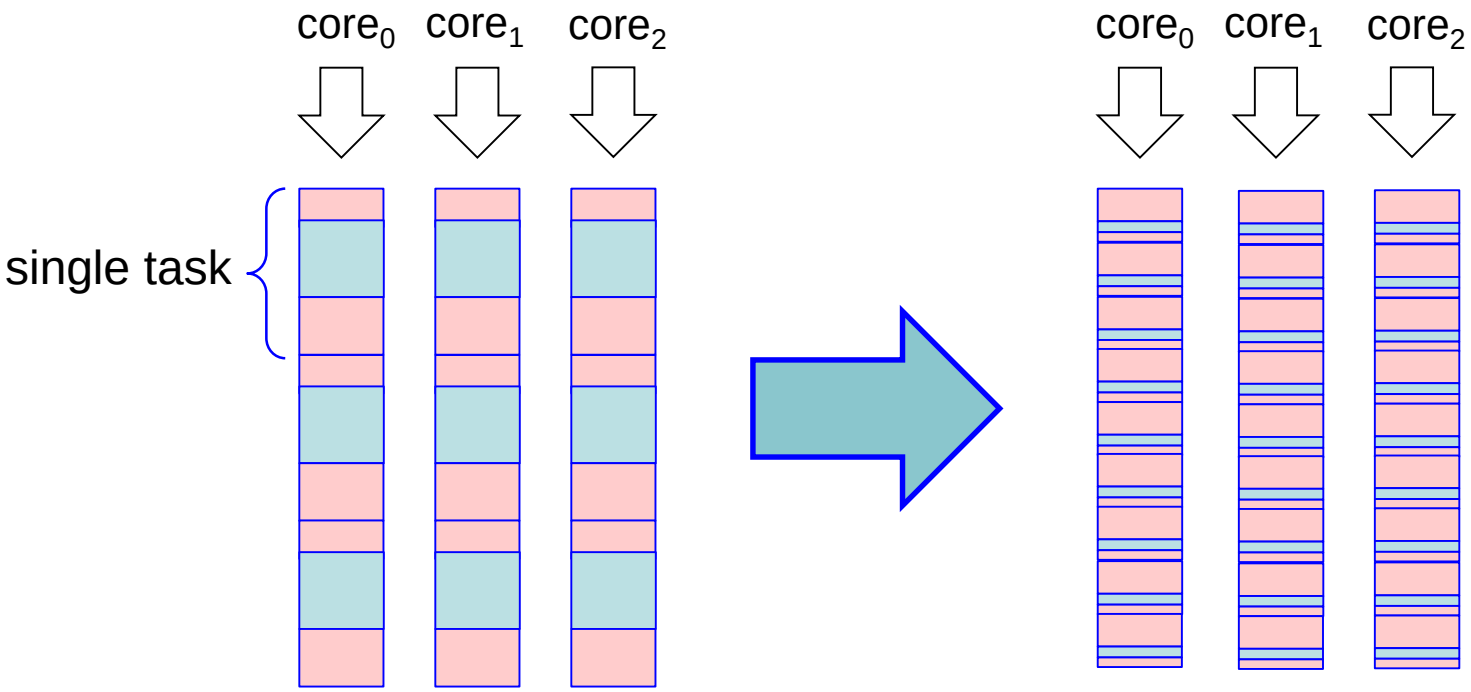
# Components of a Task (Process)



# Typical Misuse - Tiny grain size case

- **Tiny grain size**
  - E.g., each task takes little time (e.g. less than a second)
  - Most time will be spent on overhead

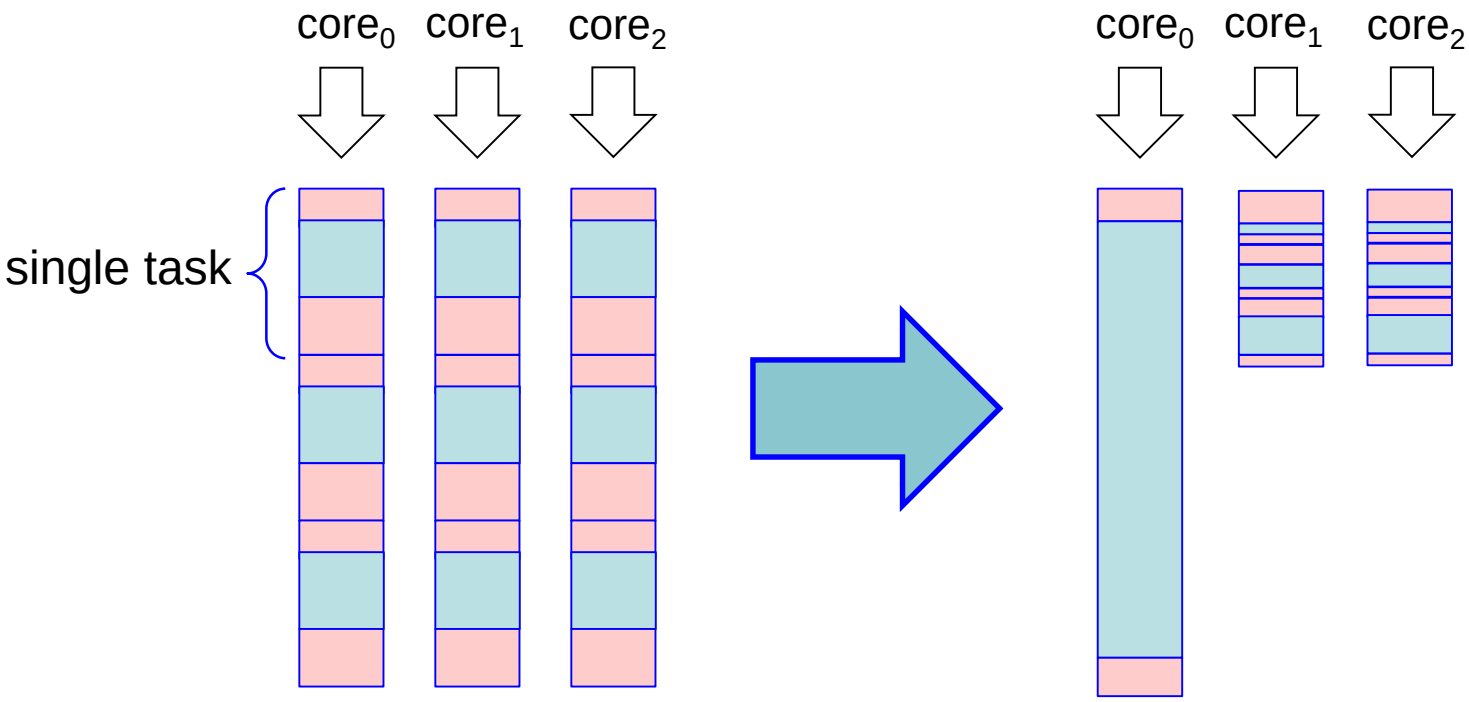
An extreme case - Cores are just idling



# Typical Misuse - Large grain size case

- **Large grain size**
  - Some tasks are much longer than the rest
  - Load balancing can never be achieved

An extreme case - Load imbalancing



# Summary

- **Why need GNU Parallel?**
- **Basic syntax of GNU Parallel and examples**
- **Use it wisely**