C.E.C.I

Consortium des Equipements
de Calcul Intensif
en Fédération Wallonie-Bruxelles

Introduction to Parallel Computing

damien.francois@uclouvain.be
November 2025

UCL Université catholique de Louvain

INSTITUT DE CALCUL INTENSIF ET DE STOCKAGE DE MASSE

Agenda

1. General concepts, definitions, challenges
2. Hardware for parallel computing
3. Programming models
4. User tools

1.

General concepts

Why parallel? (simplified)

Speed up – Solve a problem faster
→ more processing power
(a.k.a. strong scaling)

Scale up – Solve a larger problem
→ more memory and network capacity
(a.k.a. weak scaling)

Scale out – Solve many problems
→ more storage capacity

Also: energy consumption is a cubic function of clock frequency so using 2 compute units is 8 times cheaper than using one unit with double the frequency

Parallelization involves:

- *decomposition* of the work
  - **distributing instructions** to processors
  - **distributing data** to memories

- *collaboration* of the workers
  - **synchronization** of the distributed work
  - **communication** of data

Parallelization involves:

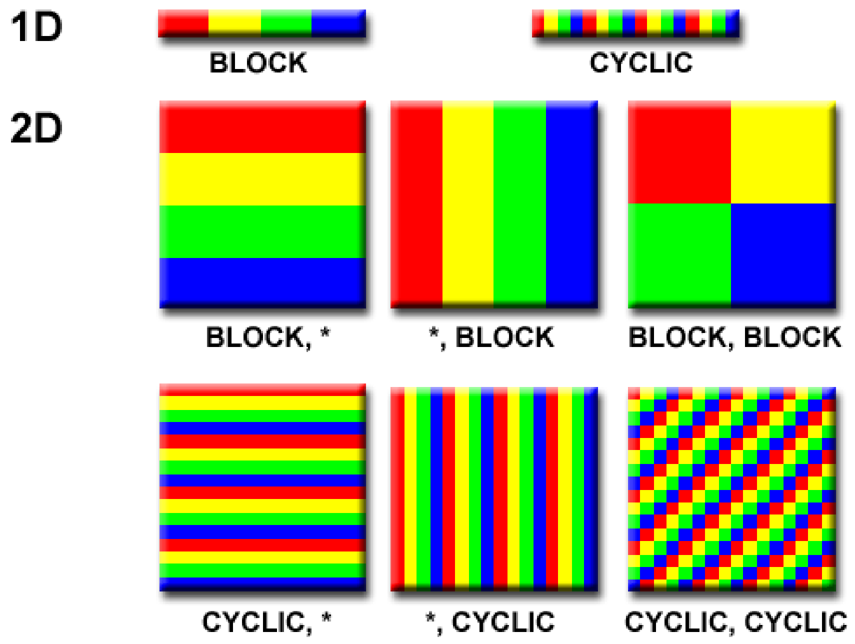
Decomposition of the work

- **Operation decomposition** : task-level parallelism
  - Multiple programs (functional decomposition)
  - Multiple instances of the same program (SPMD)
- **Data decomposition** : data-level parallelism

Parallelization involves:

Decomposition of the work

- **Operation decomposition** : task-level parallelism
- **Data decomposition** : data-level parallelism
  - **Block, cyclic**

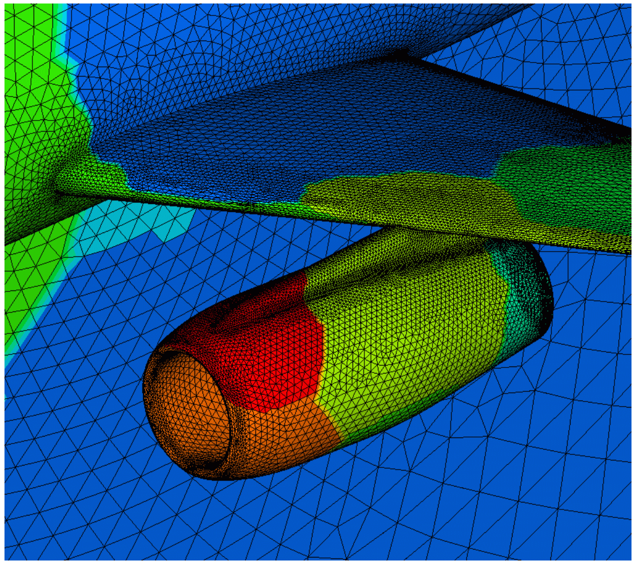


https://nyu-cds.github.io/python-mpi/04-decomposition/

Parallelization involves:

Decomposition of the work

- **Operation decomposition** : task-level parallelism
- **Data decomposition** : data-level parallelism
  - **Domain decomposition** : decomposition of work and data is done in a higher model, e.g. in the reality

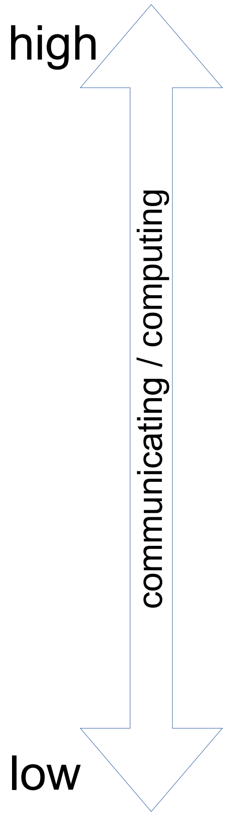


https://fun3d.larc.nasa.gov/example-54.html

Parallelization involves:

Collaboration of the workers

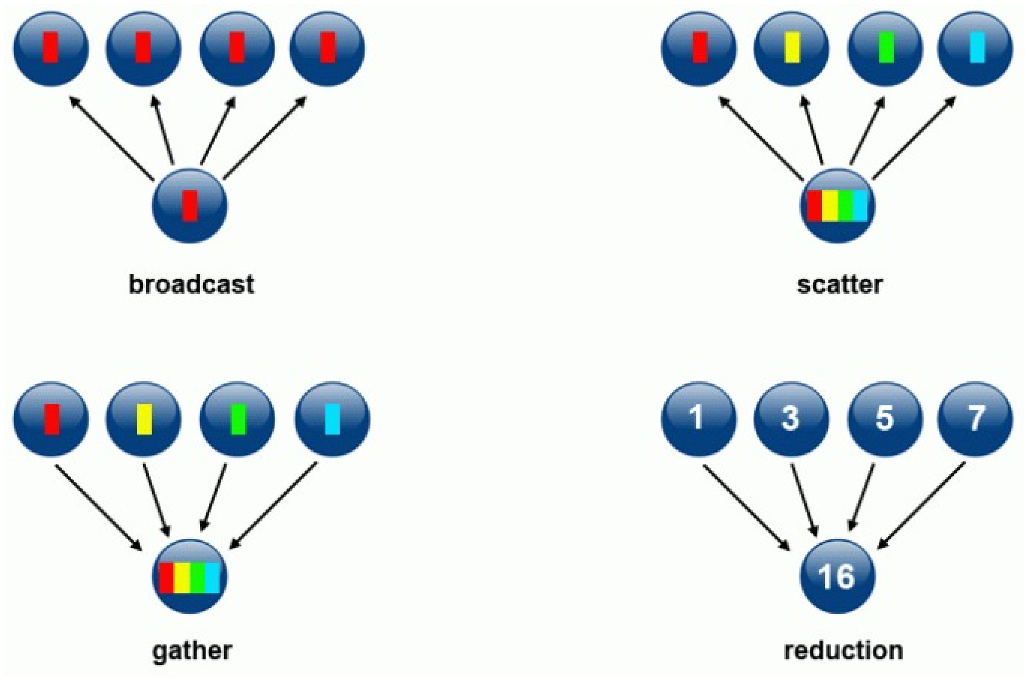
- **Synchronization of the workers**



  - **Synchronous** (SIMD) at the processor level ; the same processor instruction for each worker at any time ; (instruction level)
  - **Fine-grained** parallelism : subtasks communicate many times per second (typically at the loop level)
  - **Coarse-grained** parallelism : they do not communicate many times per second (typically function-call level)
  - **Embarrassingly parallel** : they rarely or never have to communicate (asynchronous)

Collaboration of the workers

- **Communication between workers**
  - Point to point
  - Broadcast
  - Scatter
  - Gather
  - Reduction



https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial

Does it work?
Speedup, Efficiency, Scalability

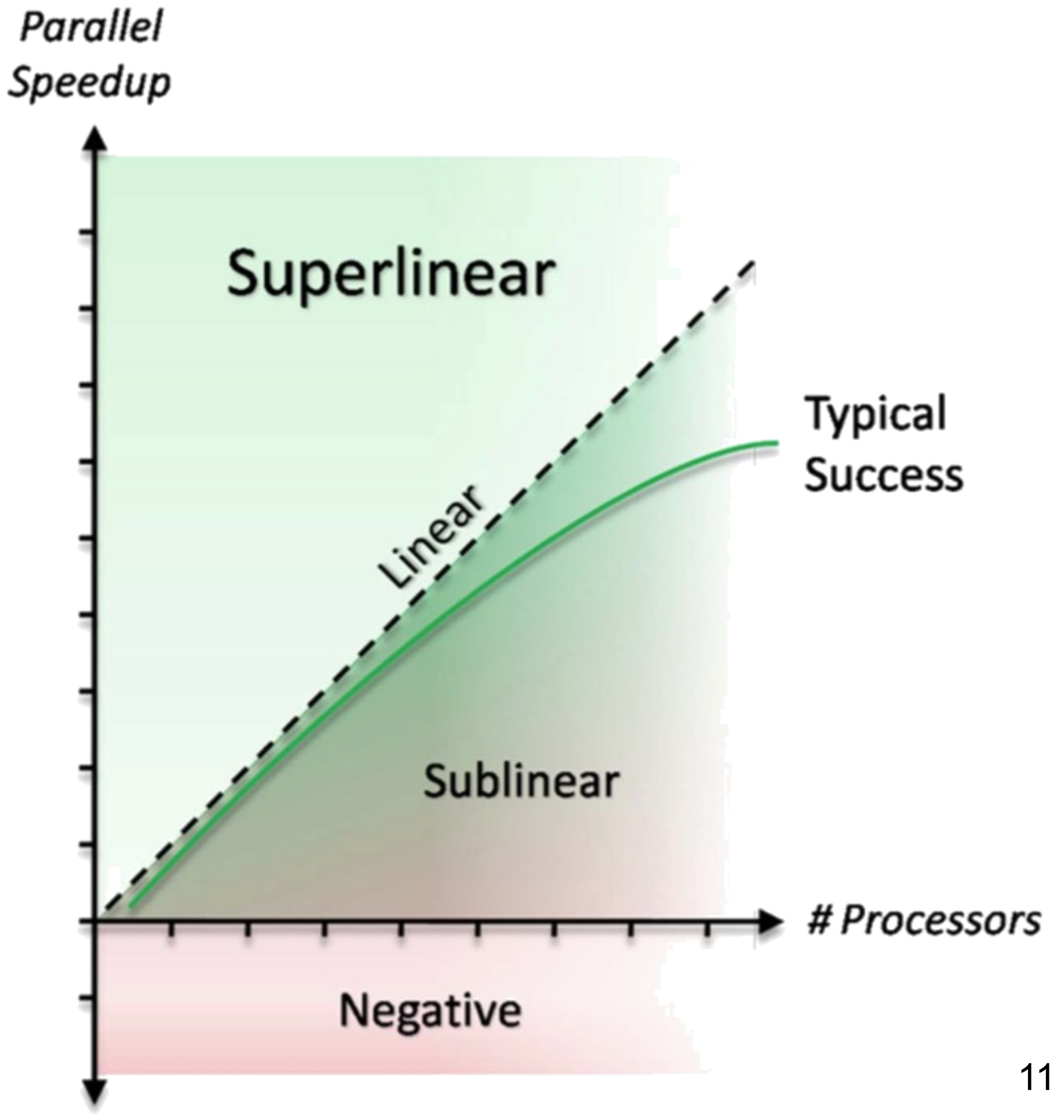
Time for serial operations

Speedup $S = \frac{T_S}{T_P}$

Time for parallel operations

Efficiency $E = \frac{S}{p} = \frac{T_S}{pT_p}$

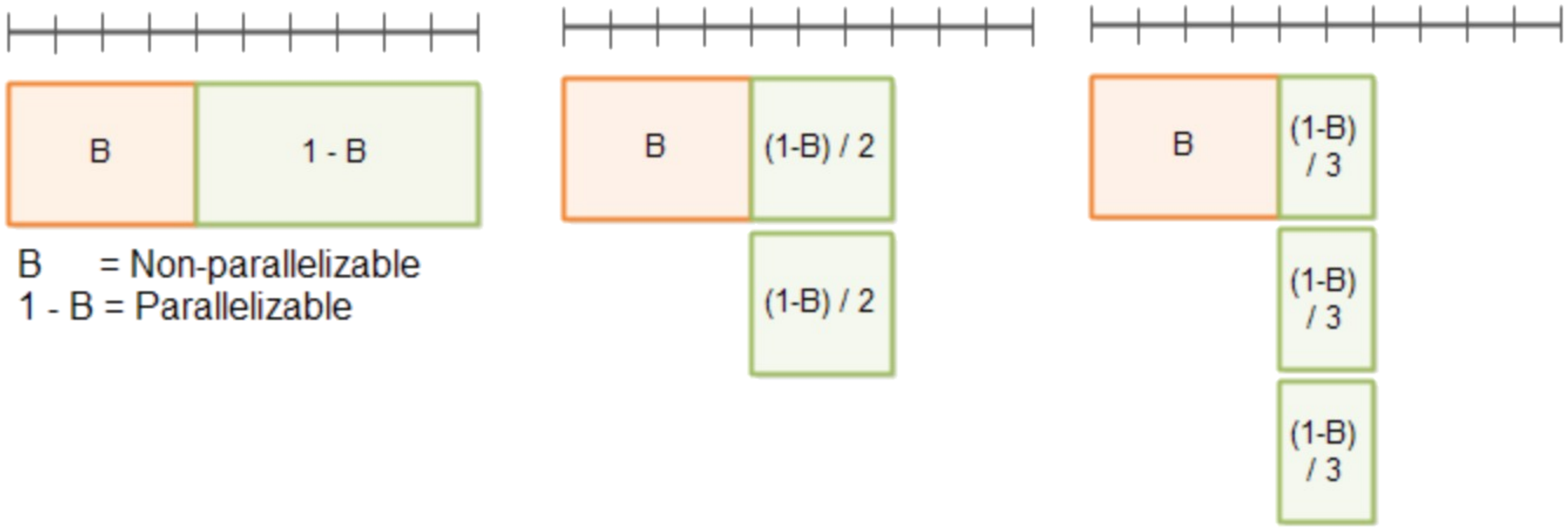
Number of processors



https://www.cs.uky.edu/~jzhang/CS621/chapter7.pdf

Challenge 1: Amdahl's Law

Not all the work can be decomposed



In parallel computing, Amdahl's law is mainly used to predict the theoretical maximum speedup for programs using multiple processors.

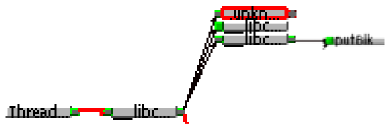
http://tutorials.jenkov.com/java-concurrency/amdahls-law.html

Why wouldn’t it work?

Challenge 2: Parallel overhead

Collaboration means communication and extra work

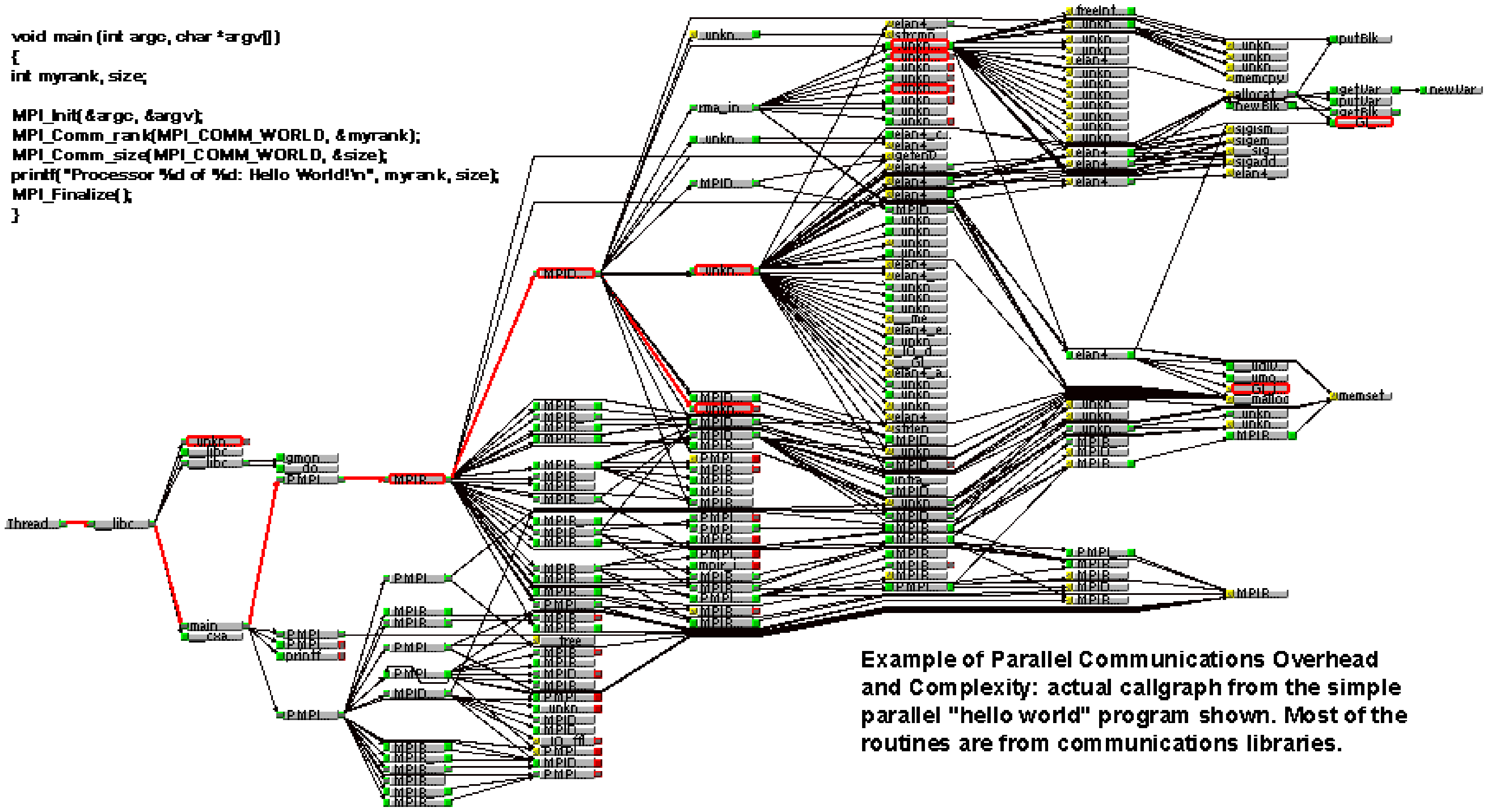
```
void main (int argc, char *argv[])
{




printf("Processor %d of %d: Hello World!\n",

}
```



https://computing.llnl.gov/tutorials/parallel_comp/images/helloWorldParallelCallgraph.gif

Challenge 2: Parallel overhead

Collaboration means communication and extra work



```
void main (int argc, char *argv[])
{
int myrank, size;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
printf("Processor %d of %d: Hello World!\n", myrank, size);
MPI_Finalize();
}
```

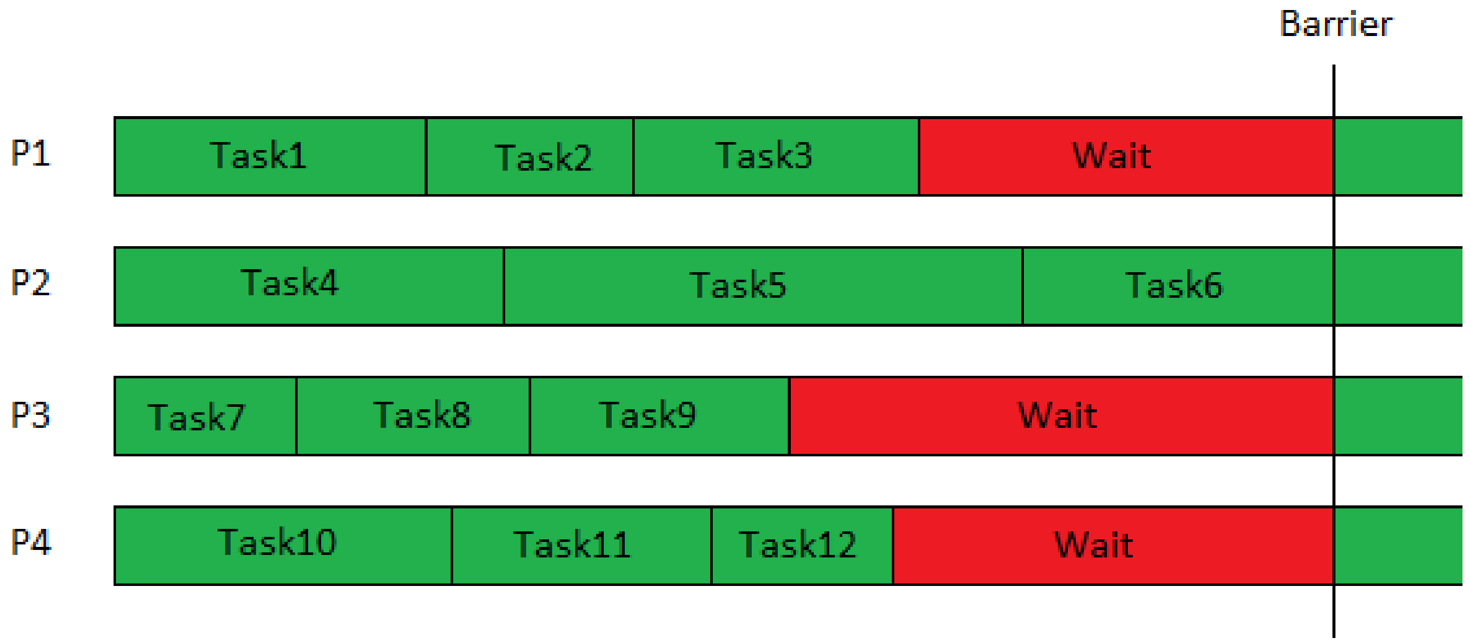
Example of Parallel Communications Overhead and Complexity: actual callgraph from the simple parallel "hello world" program shown. Most of the routines are from communications libraries.

https://computing.llnl.gov/tutorials/parallel_comp/images/helloWorldParallelCallgraph.gif

Why wouldn’t it work?

Challenge 3: Load imbalance

Parallelization is efficient only if every worker roughly has the same amount of work



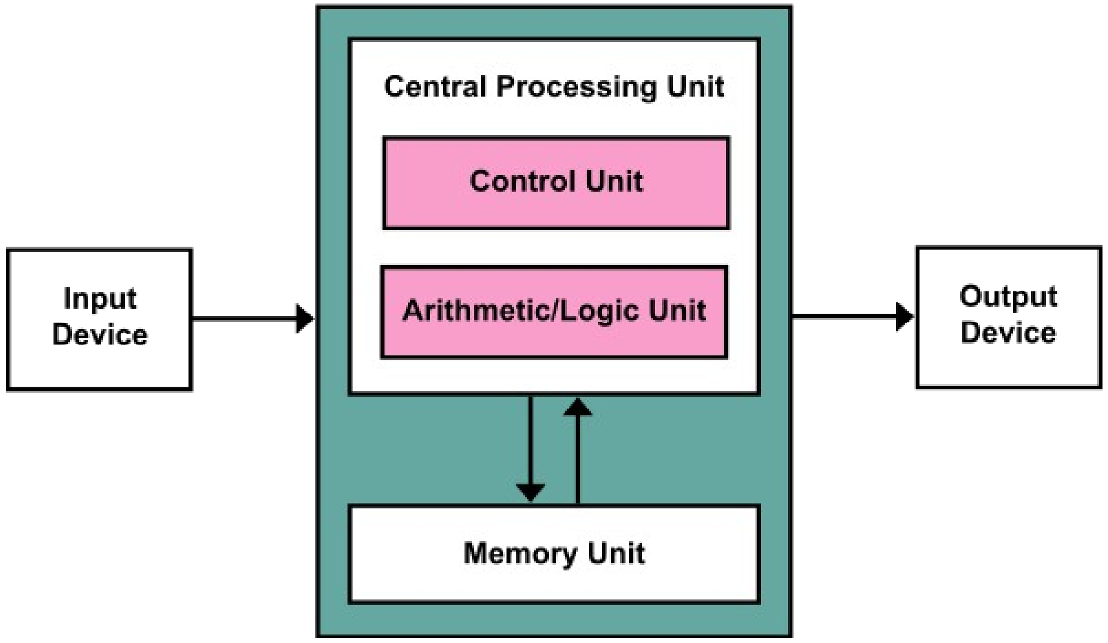
https://hpc-wiki.info/hpc/Load_Balancing

2.

Hardware for parallel computing

Von Neumann (serial) architecture

An abstract view of early computers



https://en.wikipedia.org/wiki/Von_Neumann_architecture

Parallelism at the CPU (core) level

An abstract view of modern CPUs

- Instruction-level parallelism (**ILP**)
  - Instruction pipelining
  - Out-of-order execution
  - Speculative execution
  - ...
- Single Instruction Multiple Data (**SIMD**)
- Simultaneous multithreading (**SMT**)

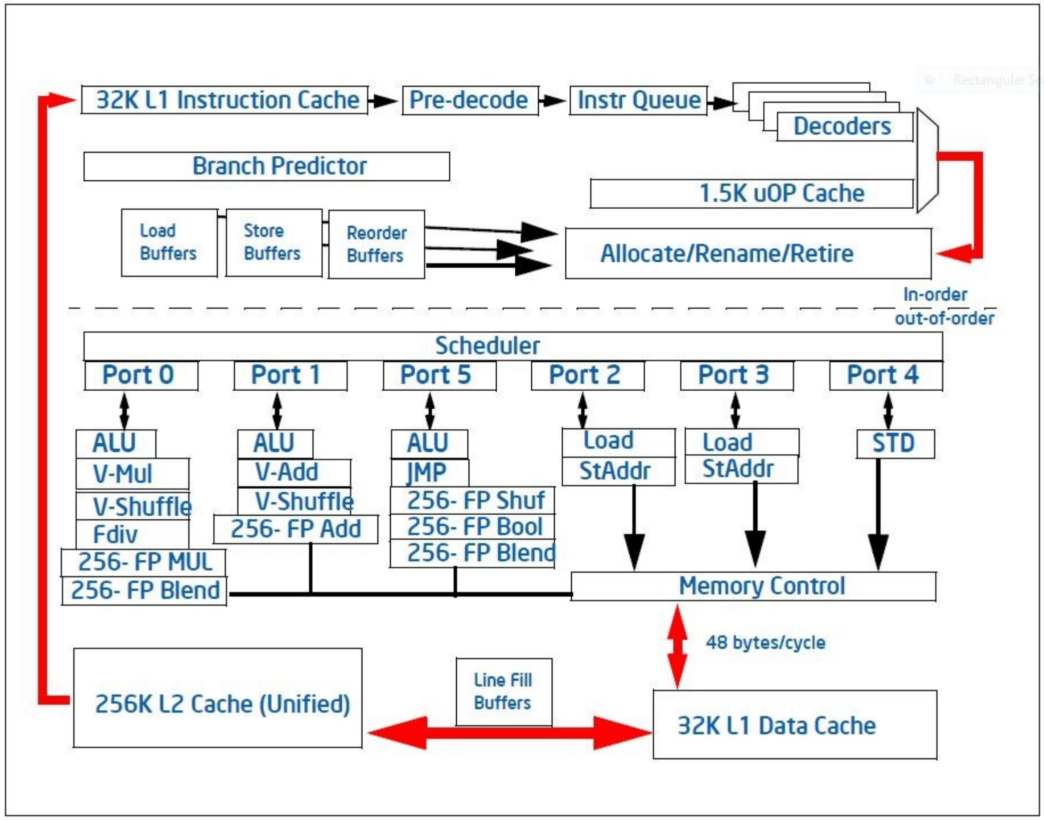
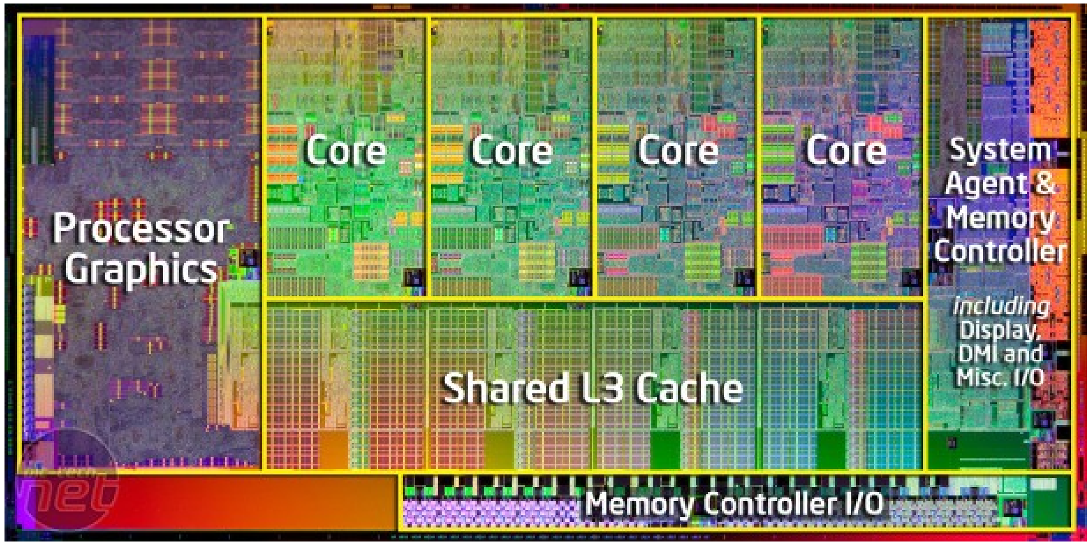


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

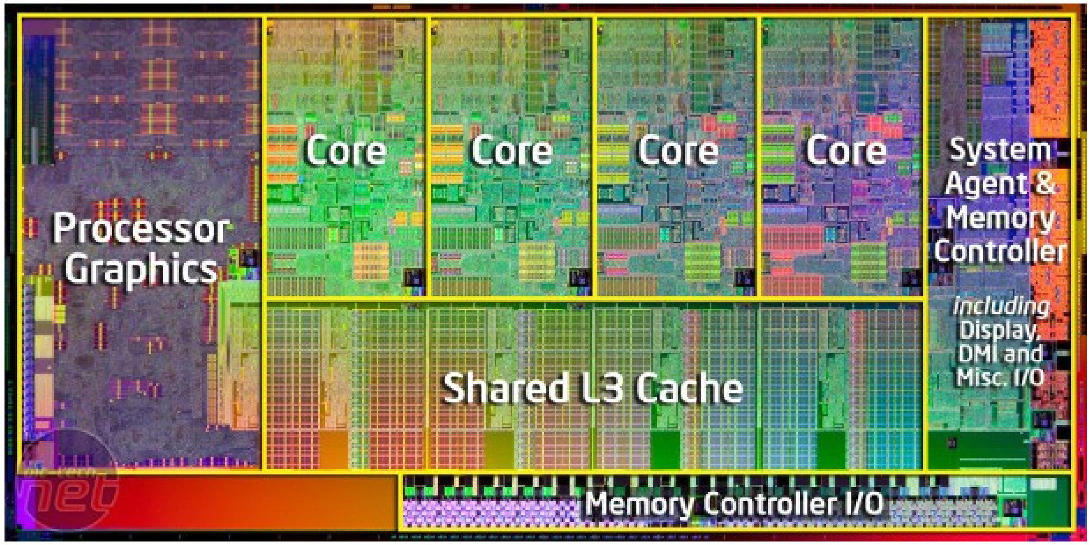
Parallelism at the chip (socket) level

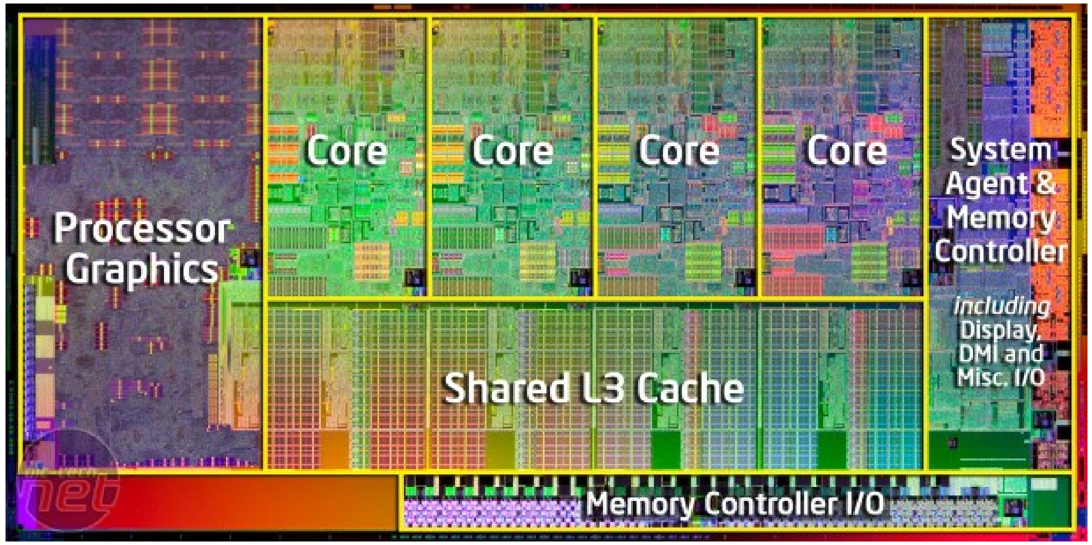
- **Multicore** parallelism



Parallelism at the computer level

- Multi-socket parallelism
  - SMP
  - NUMA





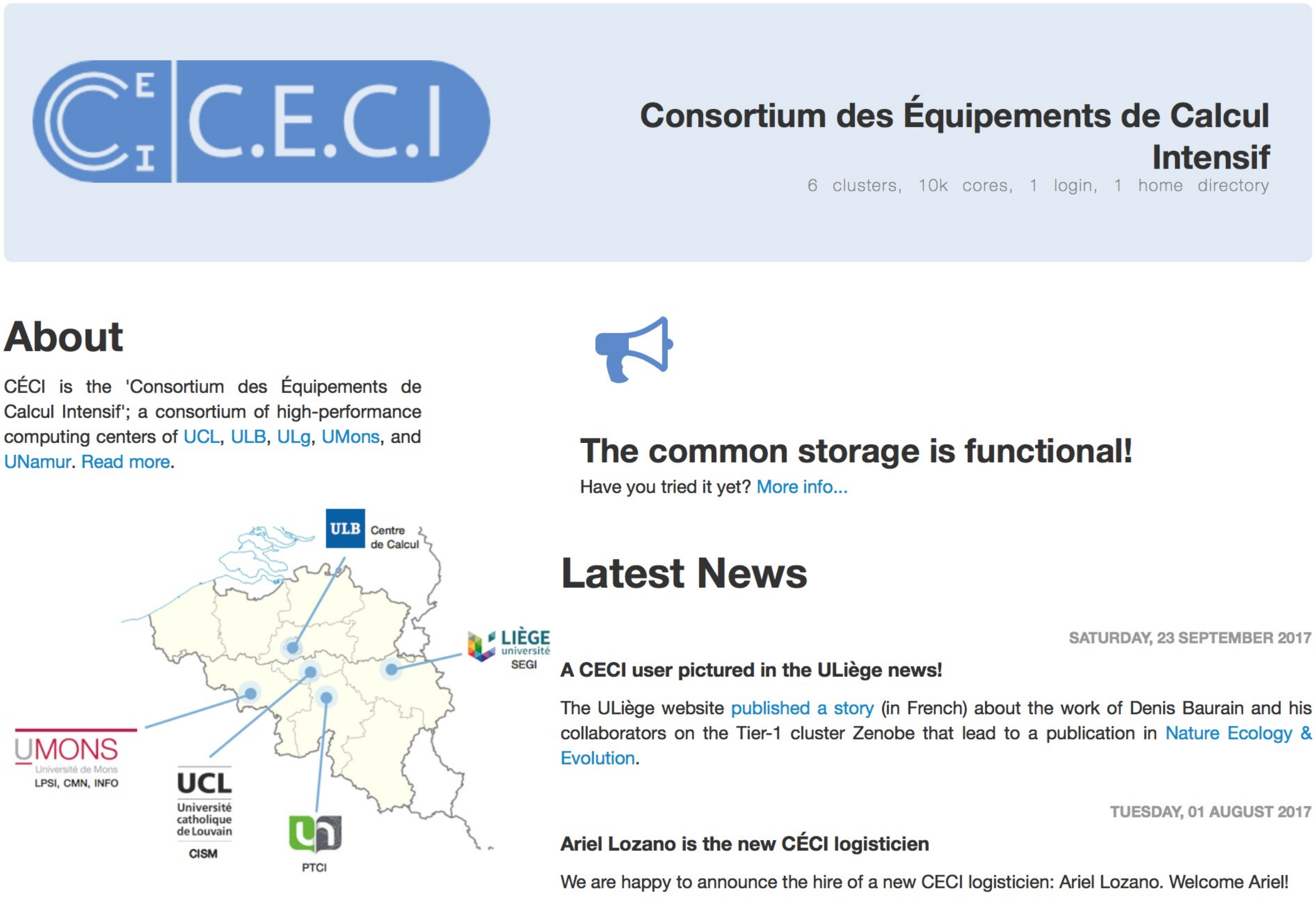
- Accelerators

Parallelism at the data center level

Multi-node parallelism

Parallelism at the data center level

Cluster computing



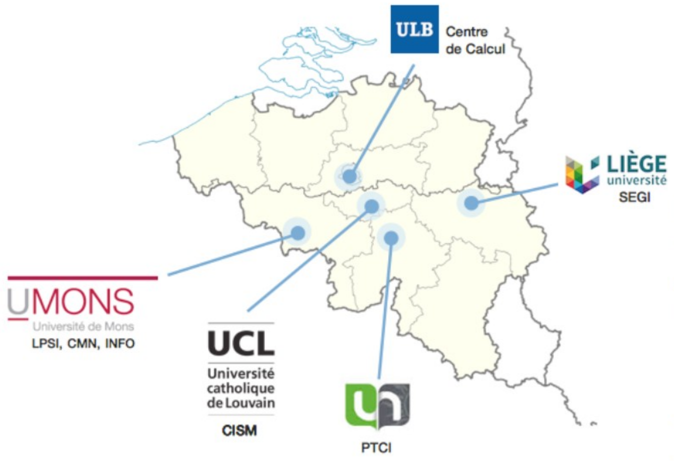
C.E.C.I

Consortium des Équipements de Calcul Intensif

6 clusters, 10k cores, 1 login, 1 home directory

About

CÉCI is the 'Consortium des Équipements de Calcul Intensif'; a consortium of high-performance computing centers of UCL, ULB, ULg, UMons, and UNamur. Read more.


ULB Centre de Calcul
LIÈGE université SEGI
UMONS Université de Mons LPSI, CMN, INFO
UCL Université catholique de Louvain CISM
PTCI

The common storage is functional!

Have you tried it yet? More info...

Latest News

SATURDAY, 23 SEPTEMBER 2017

A CECI user pictured in the ULiège news!

The ULiège website published a story (in French) about the work of Denis Baurain and his collaborators on the Tier-1 cluster Zenobe that lead to a publication in Nature Ecology & Evolution.

TUESDAY, 01 AUGUST 2017

Ariel Lozano is the new CÉCI logisticien

We are happy to announce the hire of a new CECI logisticien: Ariel Lozano. Welcome Ariel!

Parallelism at the data center level

Cloud computing "someone else's cluster"

amazon web services

Google Cloud Platform

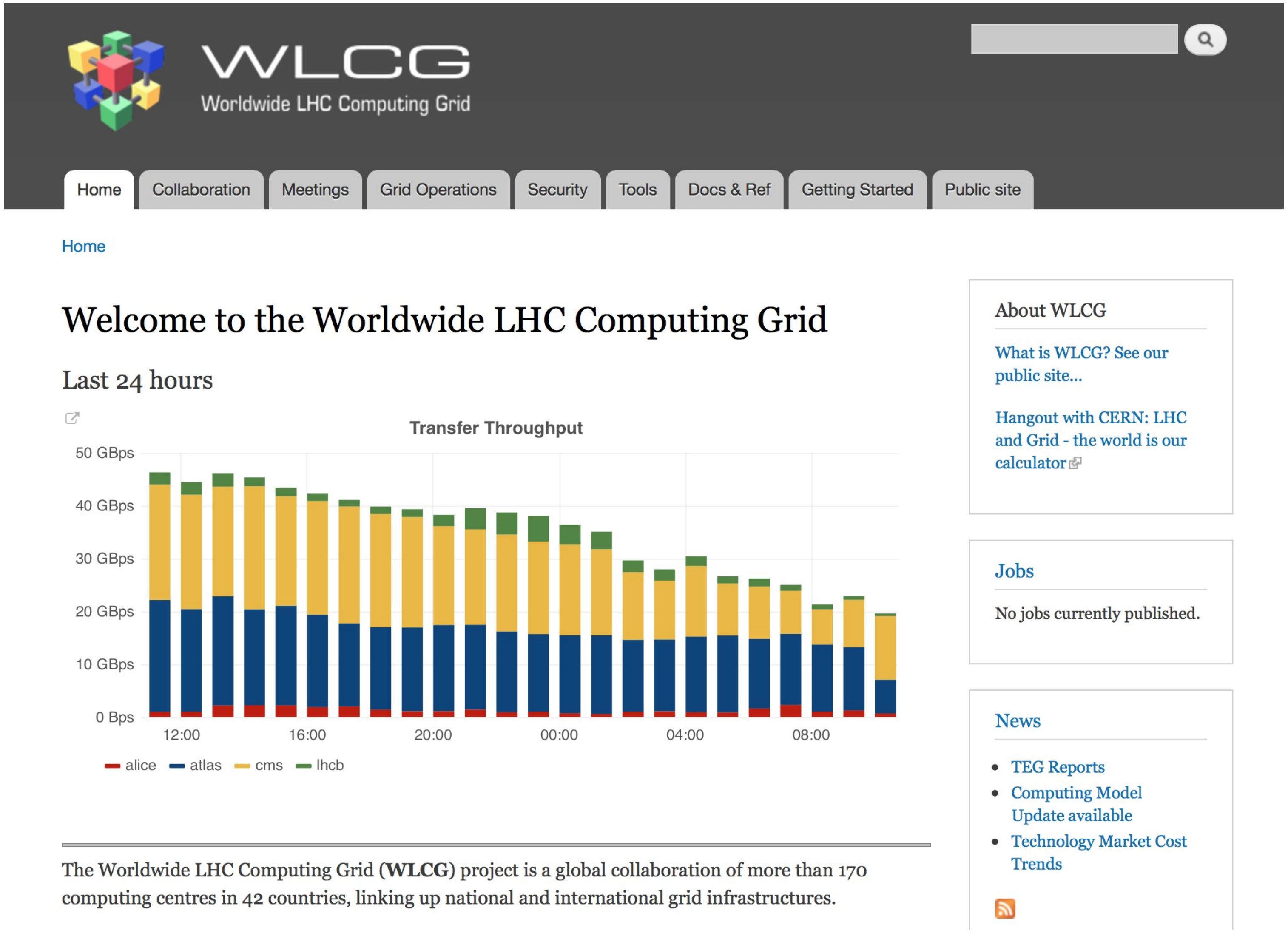
Microsoft Azure

OpenNebula

openstack™

Parallelism at the world level

Grid computing – “cluster of clusters”


WLCG
Worldwide LHC Computing Grid

Home | Collaboration | Meetings | Grid Operations | Security | Tools | Docs & Ref | Getting Started | Public site

Home

Welcome to the Worldwide LHC Computing Grid

Last 24 hours

The Worldwide LHC Computing Grid (**WLCG**) project is a global collaboration of more than 170 computing centres in 42 countries, linking up national and international grid infrastructures.

About WLCG

What is WLCG? See our public site...

Hangout with CERN: LHC and Grid - the world is our calculator

Jobs

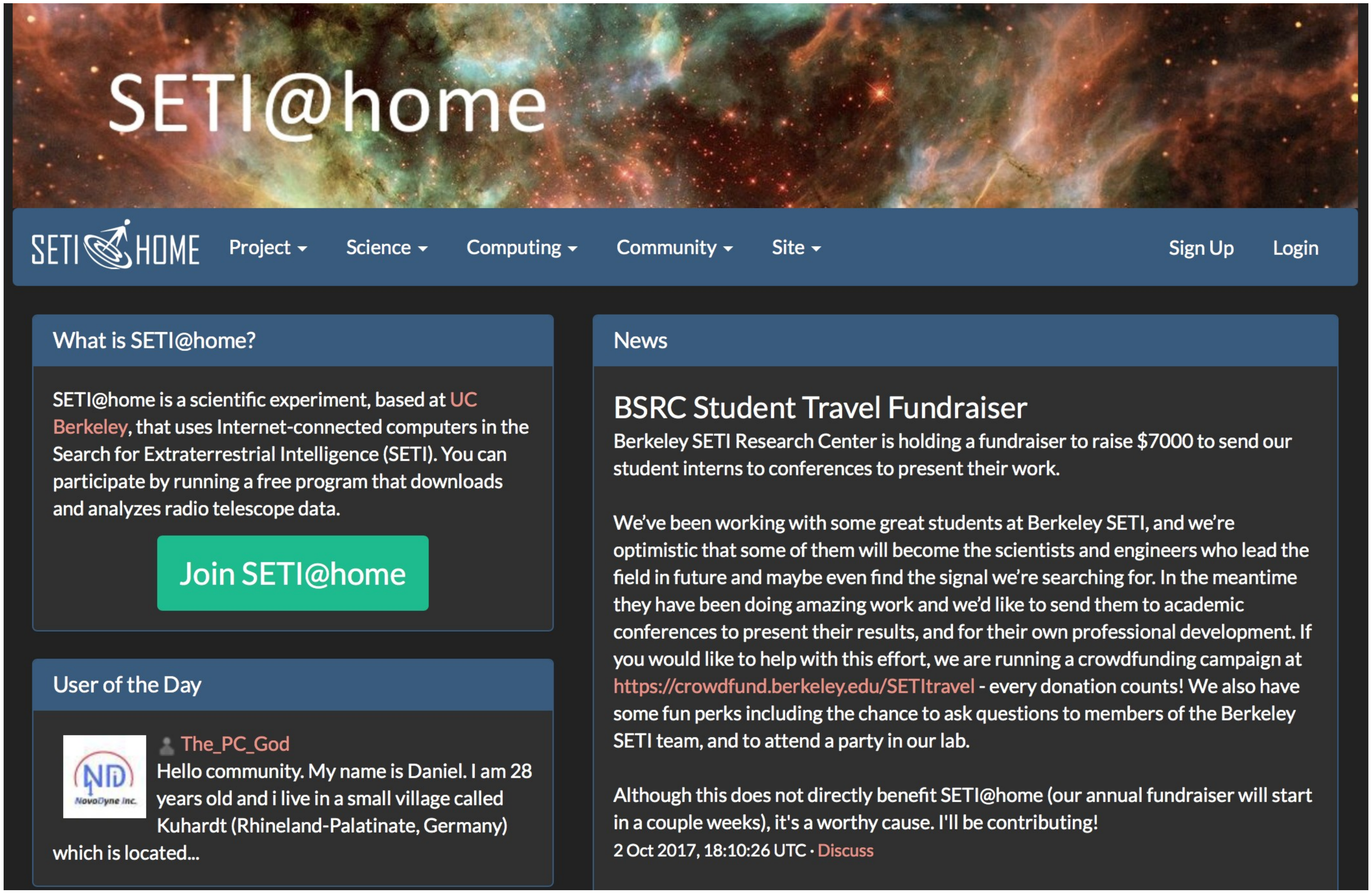
No jobs currently published.

News

- TEG Reports
- Computing Model Update available
- Technology Market Cost Trends

Parallelism at the world level

Distributed computing – “no unused cycle”



What is SETI@home?

SETI@home is a scientific experiment, based at UC Berkeley, that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You can participate by running a free program that downloads and analyzes radio telescope data.

User of the Day

The_PC_God
Hello community. My name is Daniel. I am 28 years old and i live in a small village called Kuhardt (Rhineland-Palatinate, Germany) which is located...

News

BSRC Student Travel Fundraiser

Berkeley SETI Research Center is holding a fundraiser to raise $7000 to send our student interns to conferences to present their work.

We’ve been working with some great students at Berkeley SETI, and we’re optimistic that some of them will become the scientists and engineers who lead the field in future and maybe even find the signal we’re searching for. In the meantime they have been doing amazing work and we’d like to send them to academic conferences to present their results, and for their own professional development. If you would like to help with this effort, we are running a crowdfunding campaign at https://crowdfund.berkeley.edu/SETItravel - every donation counts! We also have some fun perks including the chance to ask questions to members of the Berkeley SETI team, and to attend a party in our lab.

Although this does not directly benefit SETI@home (our annual fundraiser will start in a couple weeks), it's a worthy cause. I'll be contributing!
2 Oct 2017, 18:10:26 UTC · Discuss

3.

Programming paradigms

(logic of work sharing and organizing)

Programming models

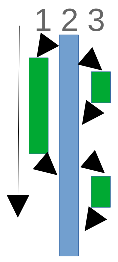
(software libraries and APIs)

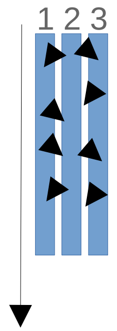
Is parallization automagic?

- **ILP**: yes, by the CPU and/or the compiler
- **SIMD**: mostly, by the compiler
- **Intra-node**: can be if the library/software you use is designed for it
- **GPUs**: can be if the library/software you use is designed for it
- **Inter-node**: never automagic.

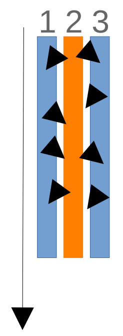
Main parallel programming paradigms

- **Task-farming:**
  - Master program distributes work to worker programs (*leader/follower*); or
  - Worker programs pick up tasks from pool (*work stealing*).
- **SPMD** (Single program multiple data)
  - A single program that contains both the logic for distributing work and computing
  - Multiple instances are started and “linked” together
  - Instances are identified with a distinct index

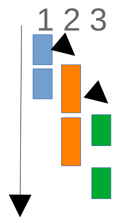




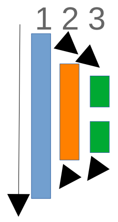
Other parallel programming paradigms



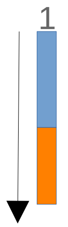
- **MPMD (Multiple program multiple data)**
- **Pipelining** : workers take care of a subtask in the processing chain and pass the intermediate result to the next worker



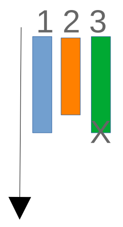
- **Divide and Conquer** :
  - workers are spawned at need and report their result to the parent
  - Speculative parallelism : workers are spawned and result possibly discarded



```
CPU1: If (very_long_computation())
CPU1: then
CPU1:     do A
CPU1: else
CPU1:     do B
```



```
CPU1: res=very_long_computation()
CPU2: do A
CPU3: do B
CPU1: if (res) discard B else discard A
```



Main programming models

- Single computer:
  - **CPUs**: PThreads, *OpenMP*, TBB, OpenCL
  - **Accelerators**: *CUDA*, OpenCL, *OpenAcc/OpenMP*, SYCL, Hipp, ROCm
- Multi-computer:
  - **Clusters**:
    - Message passing: *MPI*, PVM
    - PGAS: CoArray Fortran, UPC, Global Arrays
  - **Clouds:** MapReduce, Spark RDD
  - **Distributed computing**: BOINC

Linux program starting process

Code (program.c)

Text file

Compiler

↓

Binary (program.exe)

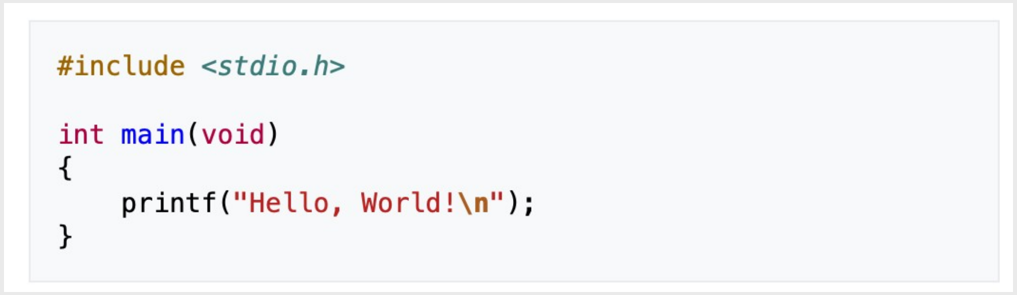
Executable file

Loader

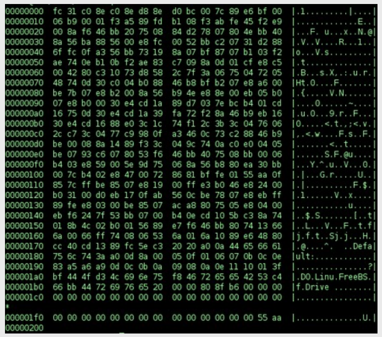
↓

Process (PID 1235)

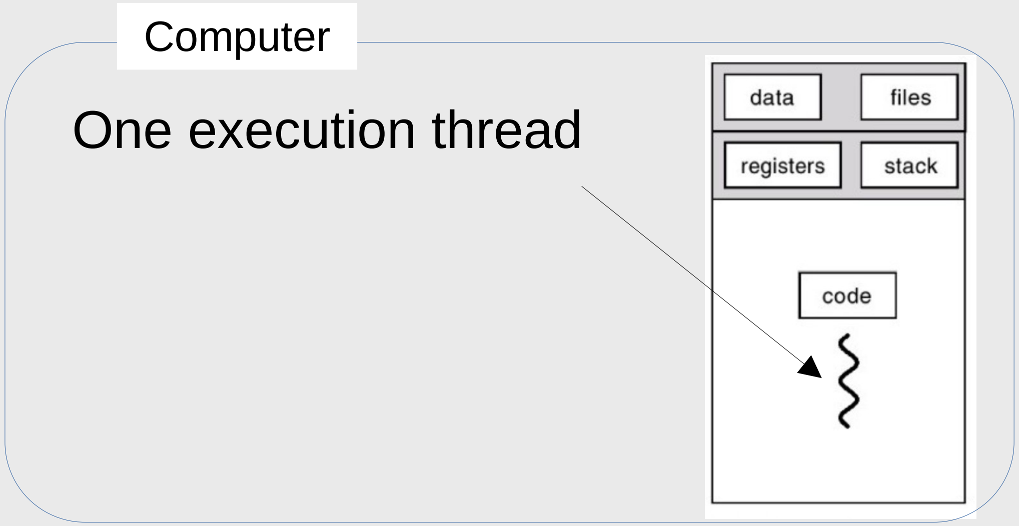
Running instance



```c
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
}
```



```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```



Code (program.c)

Text file

Compiler

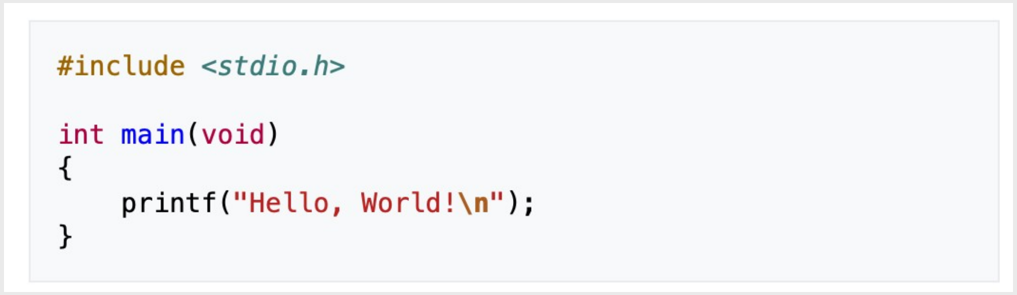
Binary (program.exe)

Executable file

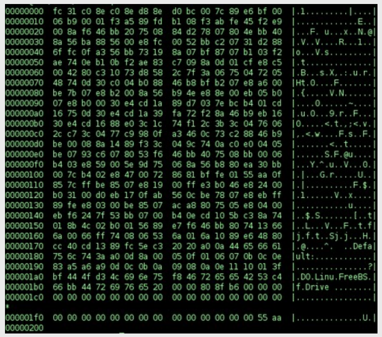
Loader

Process (PID 1235)

Running instance



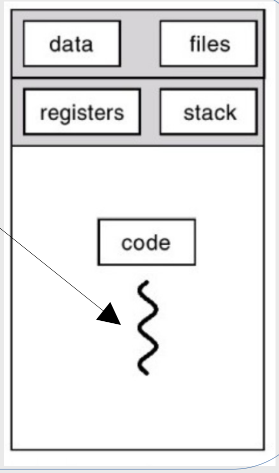
```c
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
}
```



```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```

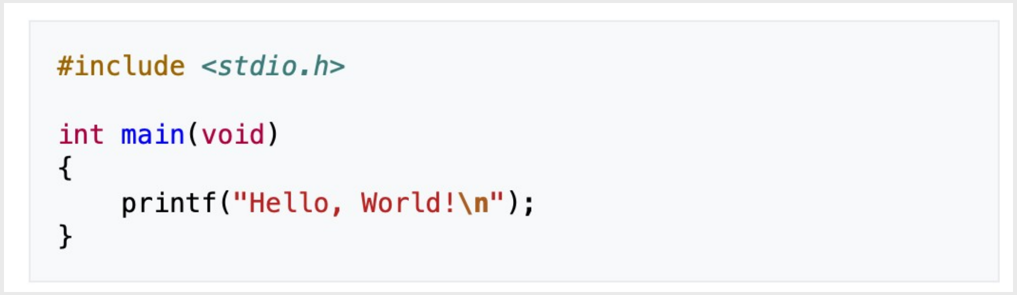
Computer

One execution thread is assigned by the OS one CPU core



Code (program.c)

Text file

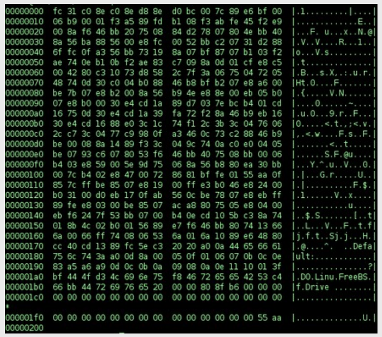


```c
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
}
```

Compiler

Binary (program.exe)

Executable file



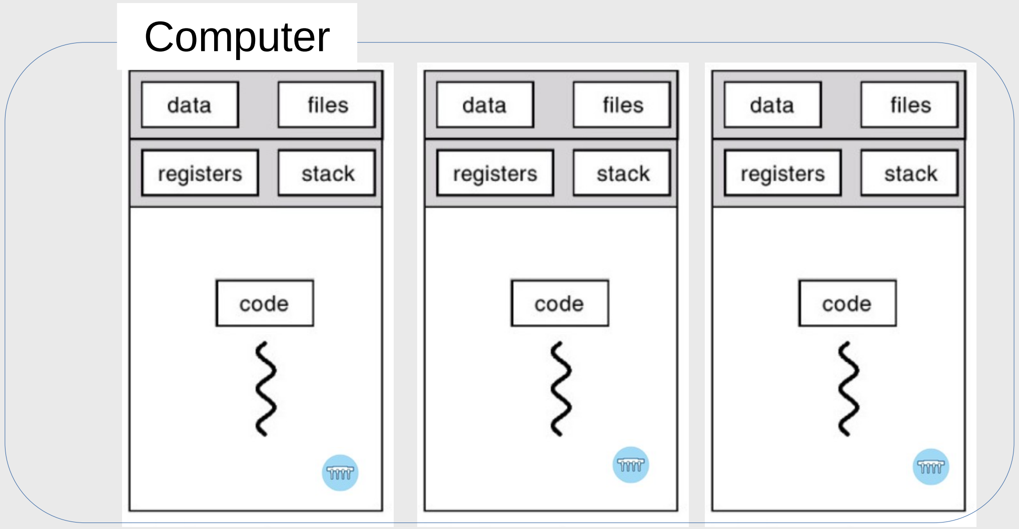
```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```

Loader,

called multiple times

Multiple **independent** processes

Running instances



Code (program.c)

Text file

Compiler

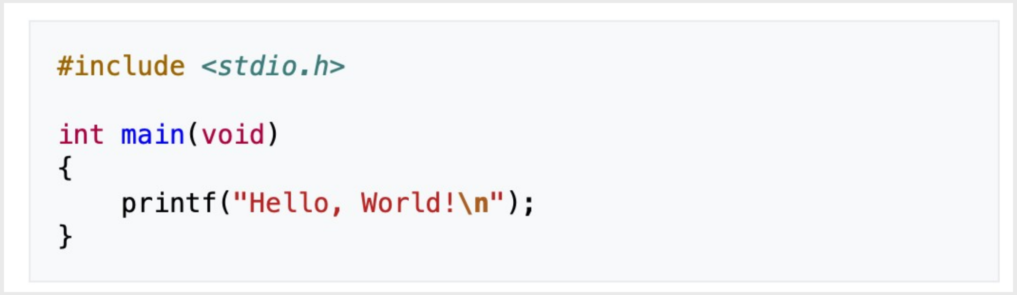
Binary (program.exe)

Executable file

Loader,
called multiple times with SSH on multiple servers

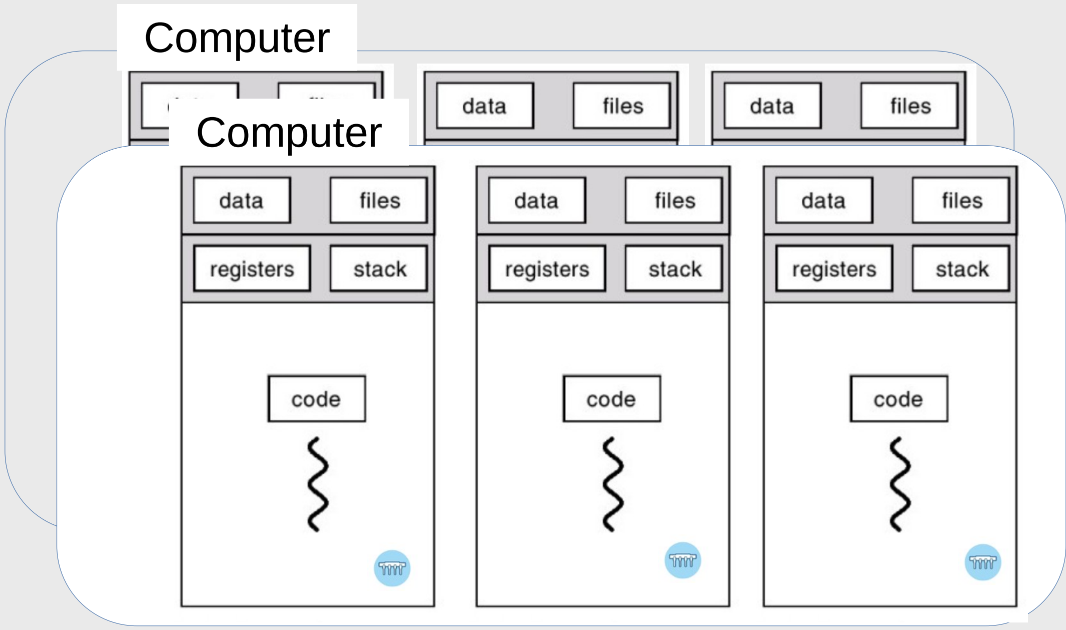
Multiple **independent** processes on multiple servers

Running instances



```c
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
}
```

```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```



Code (program.c)

Text file

Compiler

Binary (program.exe)

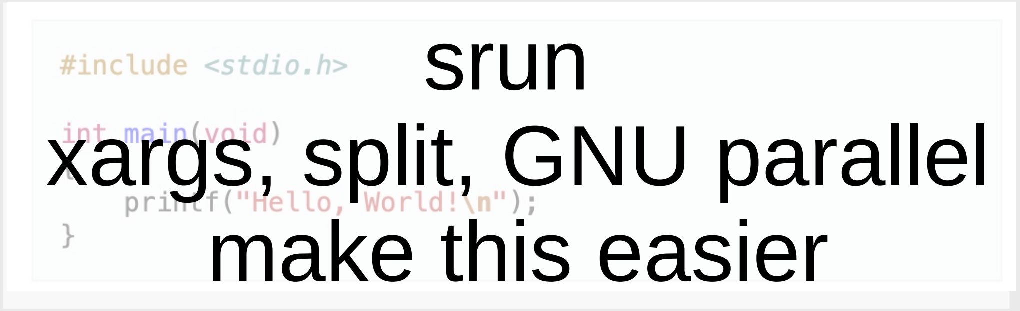
Executable file

Loader,

called multiple times with SSH on multiple servers

Multiple **independent** processes on multiple servers

Running instances

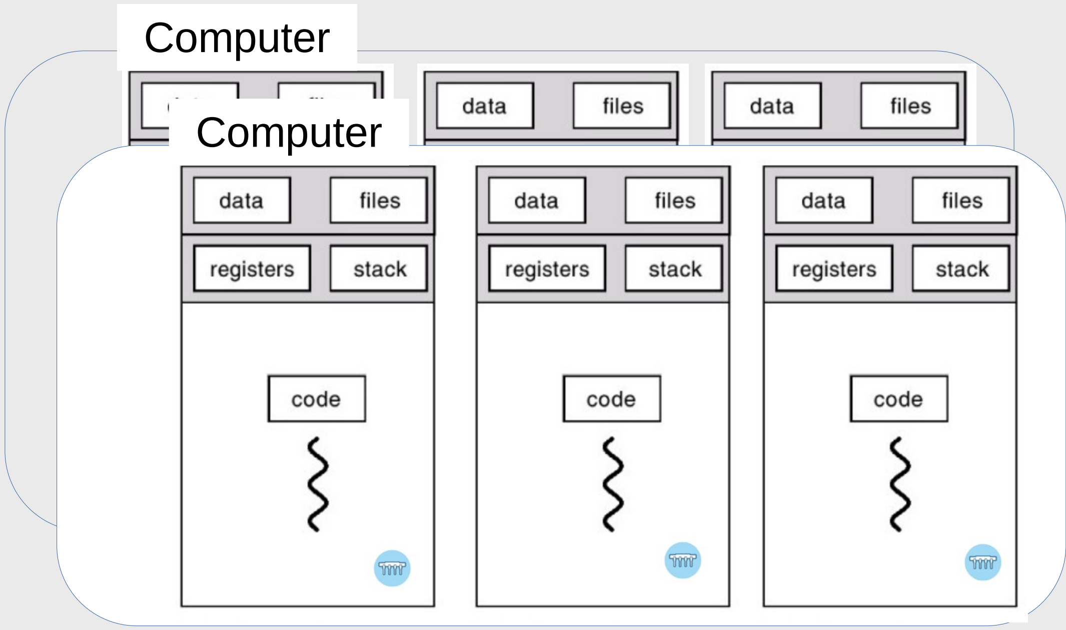


```
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
}
```

srun

xargs, split, GNU parallel

make this easier



Forking Code

Text file

Compiler

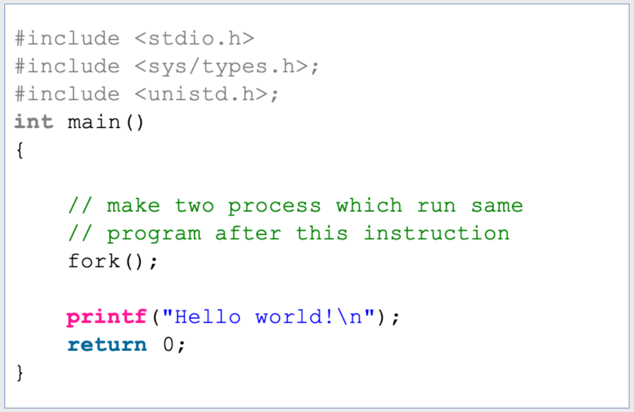
Single binary

Executable file

Loader, called once

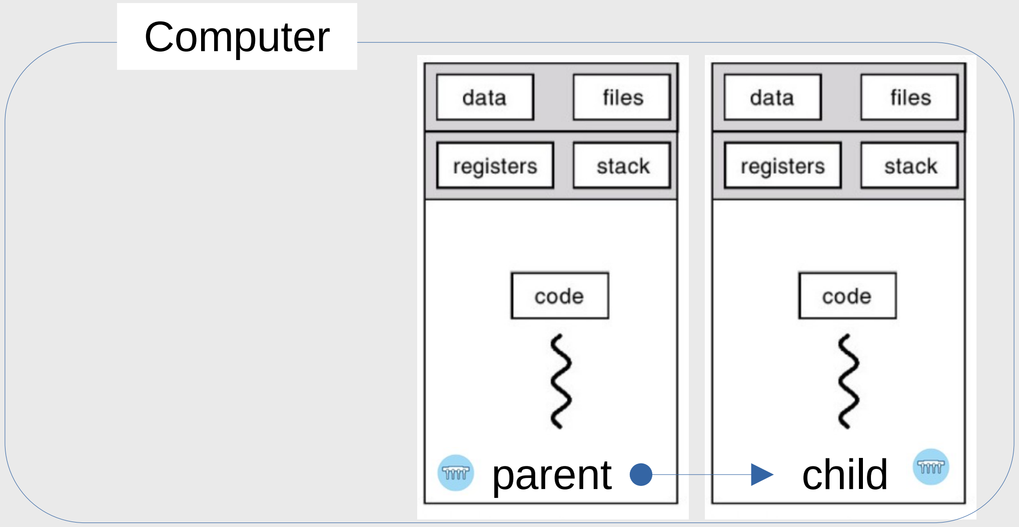
Multiple Processes with parent/child relationship

Running instances



```
#include <stdio.h>
#include <sys/types.h>;
#include <unistd.h>;
int main()
{

    // make two process which run same
    // program after this instruction
    fork();

    printf("Hello world!\n");
    return 0;
}
```

```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```



IPC – Inter-process communication

Multithreaded Code

Text file

Compiler

Single binary

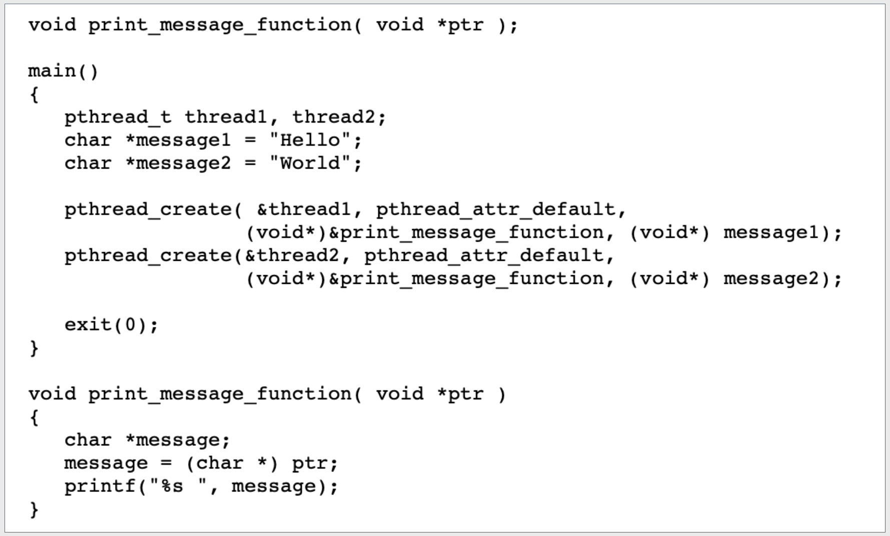
Executable file

Loader, called once

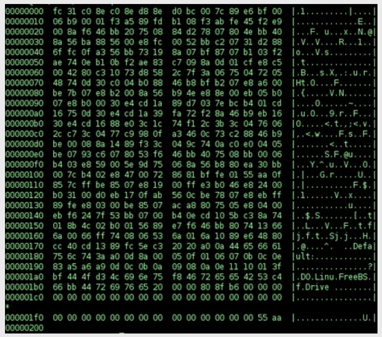
Single process with multiple threads

(multithreaded – shared memory)

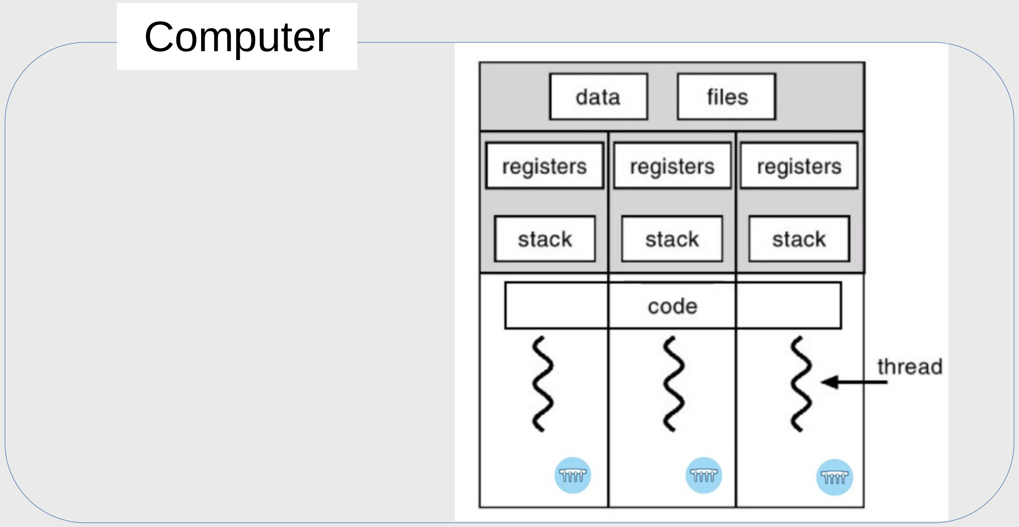
Running instance



```
void print_message_function( void *ptr );

main()
{
   pthread_t thread1, thread2;
   char *message1 = "Hello";
   char *message2 = "World";

   pthread_create( &thread1, pthread_attr_default,
                   (void*)&print_message_function, (void*) message1);
   pthread_create(&thread2, pthread_attr_default,
                   (void*)&print_message_function, (void*) message2);

   exit(0);
}

void print_message_function( void *ptr )
{
   char *message;
   message = (char *) ptr;
   printf("%s ", message);
}
```



```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```



Multithreaded Code

Text file

↓ Compiler

Single binary

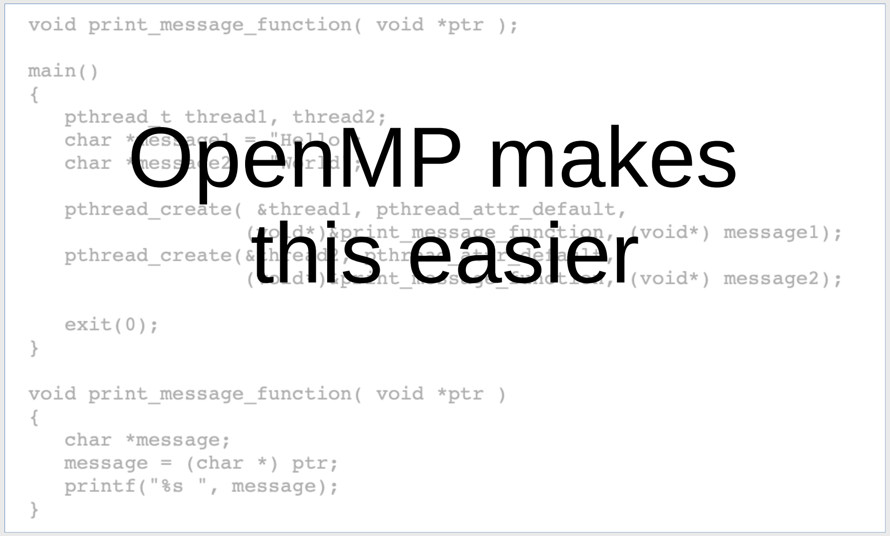
Executable file

↓ Loader, called once

Single process with multiple threads

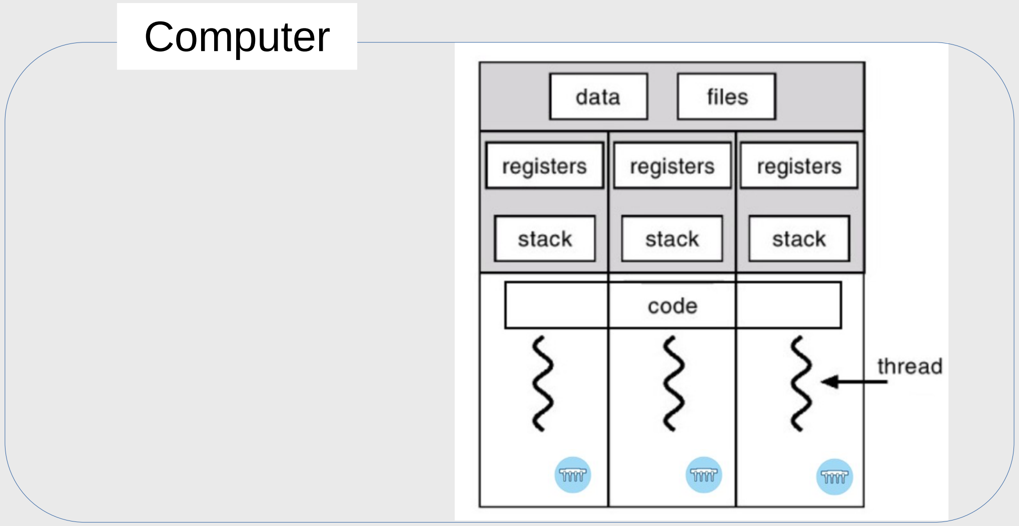
(multithreaded – shared memory)

Running instance



```
void print_message_function( void *ptr );

main()
{
   pthread_t thread1, thread2;
   char *message1 = "Hello";
   char *message2 = "World";

   pthread_create( &thread1, pthread_attr_default,
                   (void*)&print_message_function, (void*) message1);
   pthread_create(&thread2, pthread_attr_default,
                   (void*)&print_message_function, (void*) message2);

   exit(0);
}

void print_message_function( void *ptr )
{
   char *message;
   message = (char *) ptr;
   printf("%s ", message);
}
```

OpenMP makes this easier



Code (program.c)

Text file

Compiler

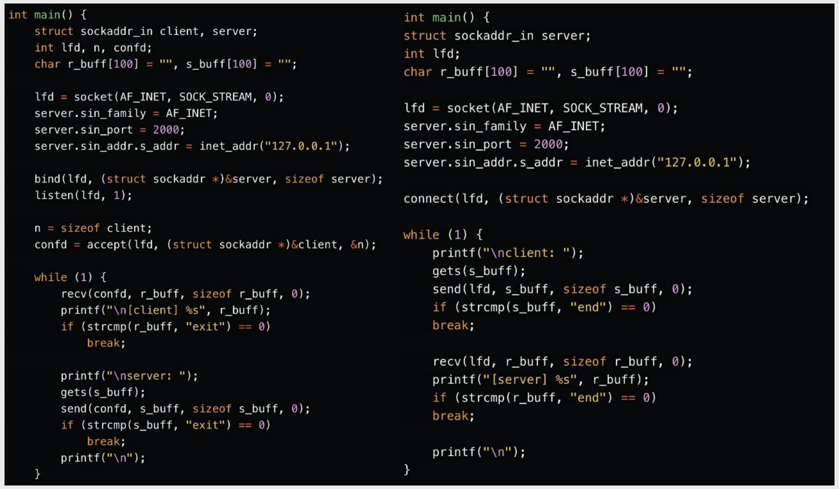
Binary (program.exe)

Executable file

Loader,
called multiple times with SSH on multiple servers

Multiple **connected** processes on multiple servers

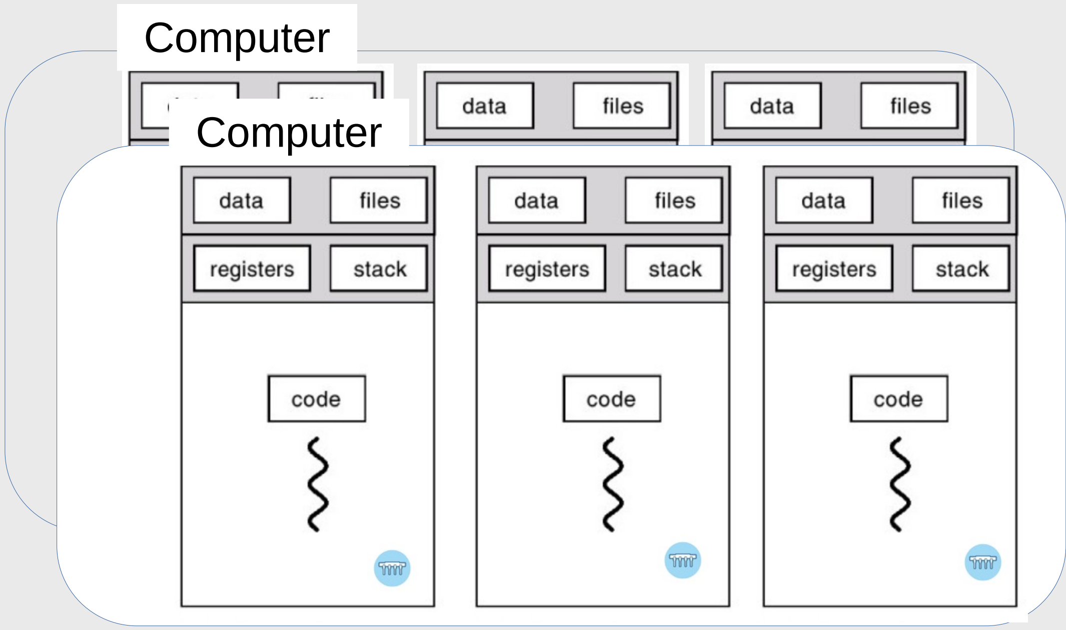
Running instances



```c
int main() {
    struct sockaddr_in client, server;
    int lfd, n, confd;
    char r_buff[100] = "", s_buff[100] = "";

    lfd = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = 2000;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");

    bind(lfd, (struct sockaddr *)&server, sizeof server);
    listen(lfd, 1);

    n = sizeof client;
    confd = accept(lfd, (struct sockaddr *)&client, &n);

    while (1) {
        recv(confd, r_buff, sizeof r_buff, 0);
        printf("\n[client] %s", r_buff);
        if (strcmp(r_buff, "exit") == 0)
            break;

        printf("\nserver: ");
        gets(s_buff);
        send(confd, s_buff, sizeof s_buff, 0);
        if (strcmp(s_buff, "exit") == 0)
            break;
        printf("\n");
    }
```

```c
int main() {
struct sockaddr_in server;
int lfd;
char r_buff[100] = "", s_buff[100] = "";

lfd = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = 2000;
server.sin_addr.s_addr = inet_addr("127.0.0.1");

connect(lfd, (struct sockaddr *)&server, sizeof server);

while (1) {
    printf("\nclient: ");
    gets(s_buff);
    send(lfd, s_buff, sizeof s_buff, 0);
    if (strcmp(s_buff, "end") == 0)
    break;

    recv(lfd, r_buff, sizeof r_buff, 0);
    printf("[server] %s", r_buff);
    if (strcmp(r_buff, "end") == 0)
    break;

    printf("\n");
}
```

```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```



Code (program.c)

Text file

Compiler

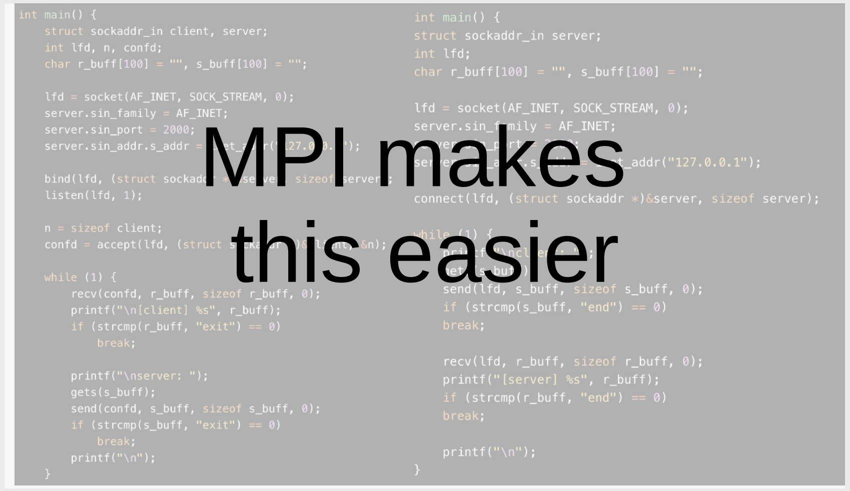
Binary (program.exe)

Executable file

Loader,
called multiple times with SSH on multiple servers

Multiple **connected** processes on multiple servers

Running instances

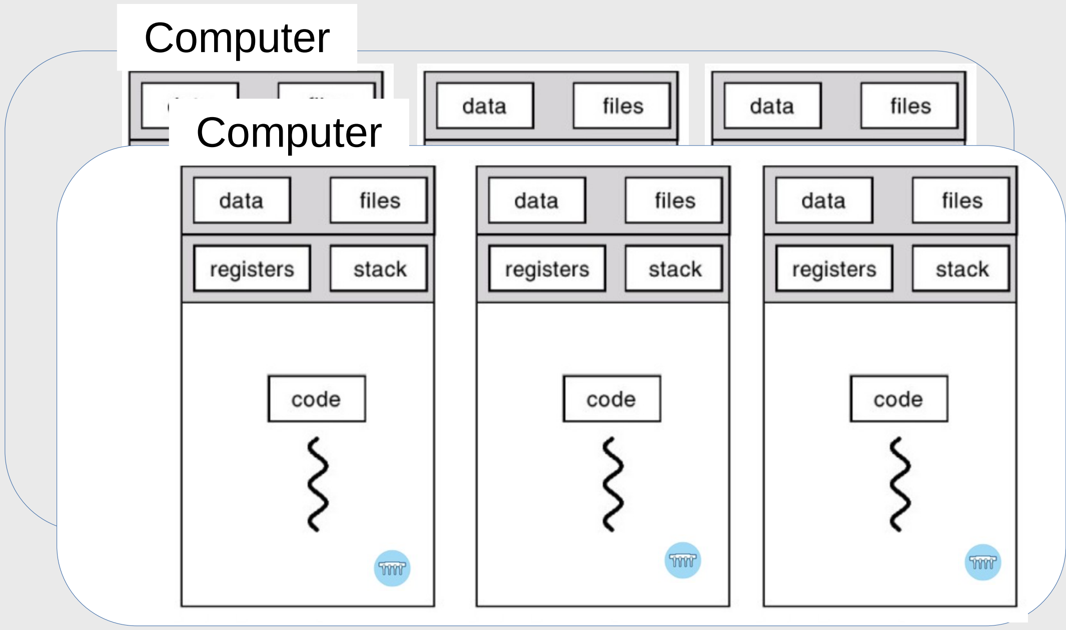


MPI makes this easier

```
int main() {
    struct sockaddr_in client, server;
    int lfd, n, confd;
    char r_buff[100] = "", s_buff[100] = "";

    lfd = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = 2000;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");

    bind(lfd, (struct sockaddr *)&server, sizeof server);
    listen(lfd, 1);

    n = sizeof client;
    confd = accept(lfd, (struct sockaddr *)&client, &n);

    while (1) {
        recv(confd, r_buff, sizeof r_buff, 0);
        printf("\n[client] %s", r_buff);
        if (strcmp(r_buff, "exit") == 0)
            break;

        printf("\nserver: ");
        gets(s_buff);
        send(confd, s_buff, sizeof s_buff, 0);
        if (strcmp(s_buff, "exit") == 0)
            break;
        printf("\n");
    }
```

```
int main() {
struct sockaddr_in server;
int lfd;
char r_buff[100] = "", s_buff[100] = "";

lfd = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = 2000;
server.sin_addr.s_addr = inet_addr("127.0.0.1");

connect(lfd, (struct sockaddr *)&server, sizeof server);

while (1) {
    printf("\nclient: ");
    gets(s_buff);
    send(lfd, s_buff, sizeof s_buff, 0);
    if (strcmp(s_buff, "end") == 0)
    break;

    recv(lfd, r_buff, sizeof r_buff, 0);
    printf("[server] %s", r_buff);
    if (strcmp(r_buff, "end") == 0)
    break;

    printf("\n");
}
```

```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```



xargs, split, GNU parallel → start multiple independent processes

OpenMP → write multithreaded programs

MPI → write multiprocess connected programs

4.

User tools
that GNU/Linux offers

4.1. Parallelized utilities

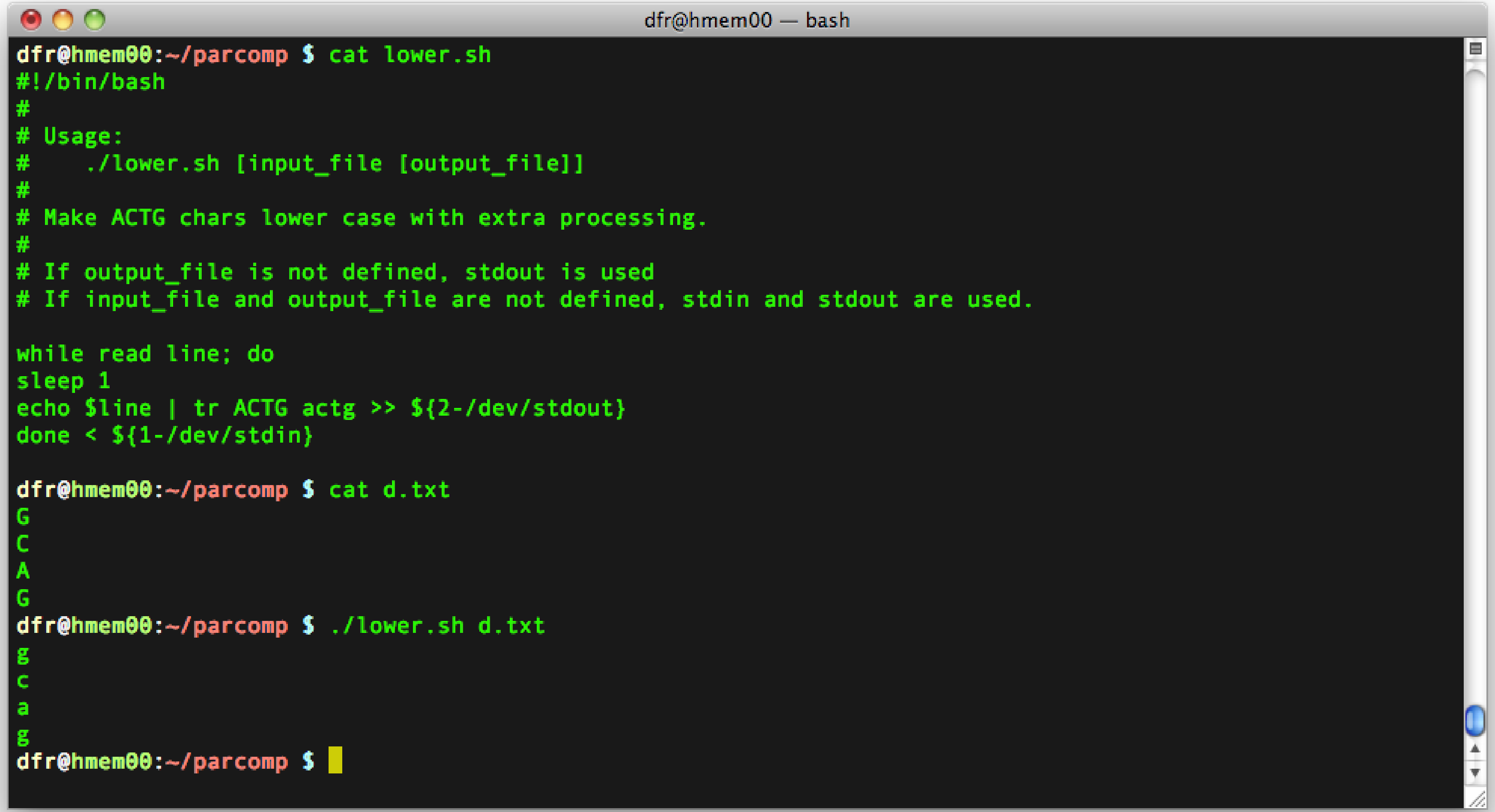
Some tools have a parallelized counterpart, or parallel options. Examples:

serial		parallel
gzip	⟷	pigz
grep	⟷	ripgrep, singrep
ssh	⟷	clustershell
sort	⟷	sort --parallel
scp	⟷	bbcp
bc	⟷	bcx
...		...

You might have to install them by yourself if they are not present on the clusters

4.2. Job control & Parallel processes in Bash

Consider the following example program

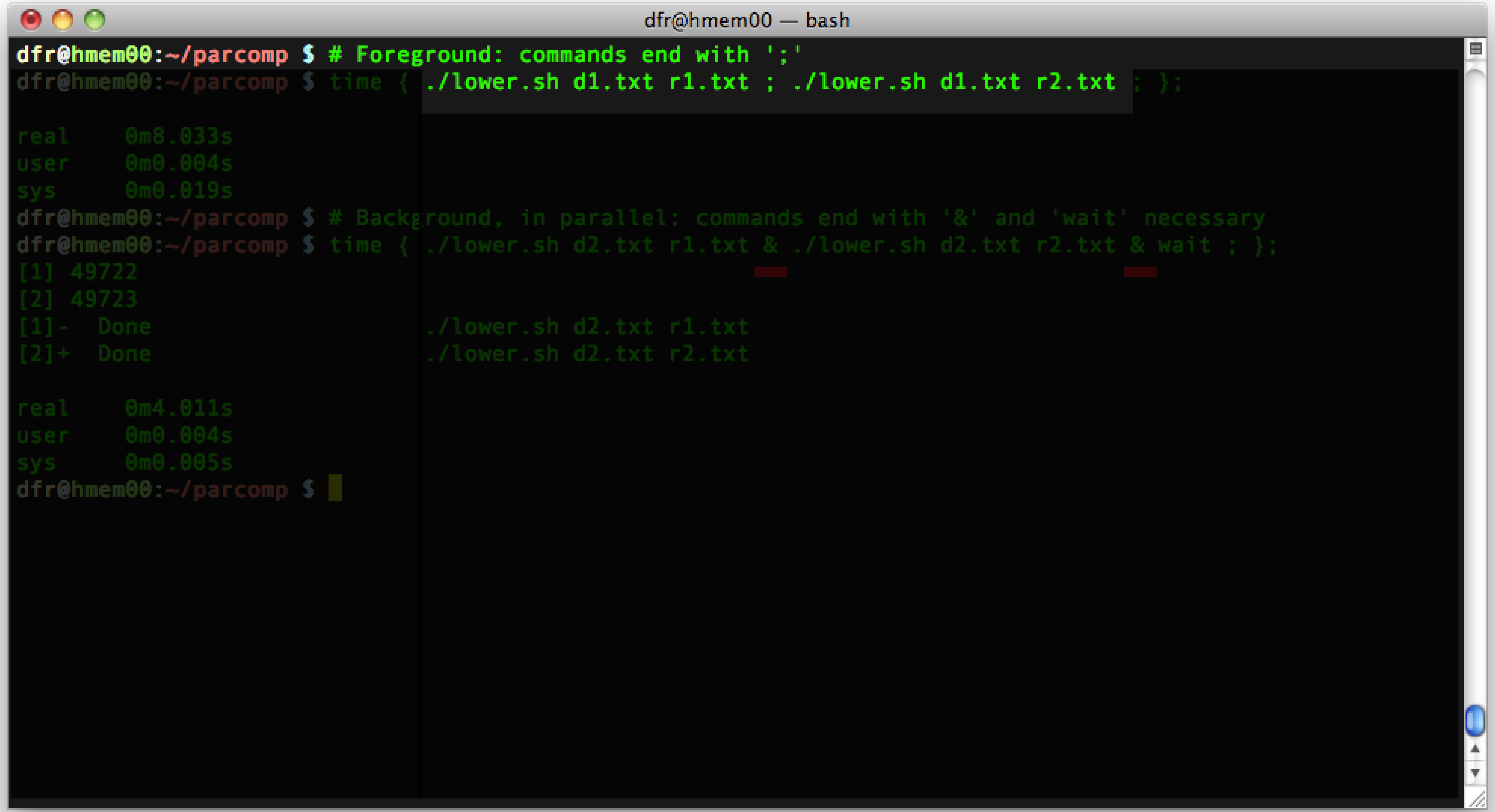

dfr@hmem00 — bash

```
dfr@hmem00:~/parcomp $ cat lower.sh
#!/bin/bash
#
# Usage:
#    ./lower.sh [input_file [output_file]]
#
# Make ACTG chars lower case with extra processing.
#
# If output_file is not defined, stdout is used
# If input_file and output_file are not defined, stdin and stdout are used.

while read line; do
sleep 1
echo $line | tr ACTG actg >> ${2-/dev/stdout}
done < ${1-/dev/stdin}

dfr@hmem00:~/parcomp $ cat d.txt
G
C
A
G
dfr@hmem00:~/parcomp $ ./lower.sh d.txt
g
c
a
g
dfr@hmem00:~/parcomp $ 
```

It is written in Bash and just transforms some upper case letters to lower case

4.2. Job control & Parallel processes in Bash

Run the program twice

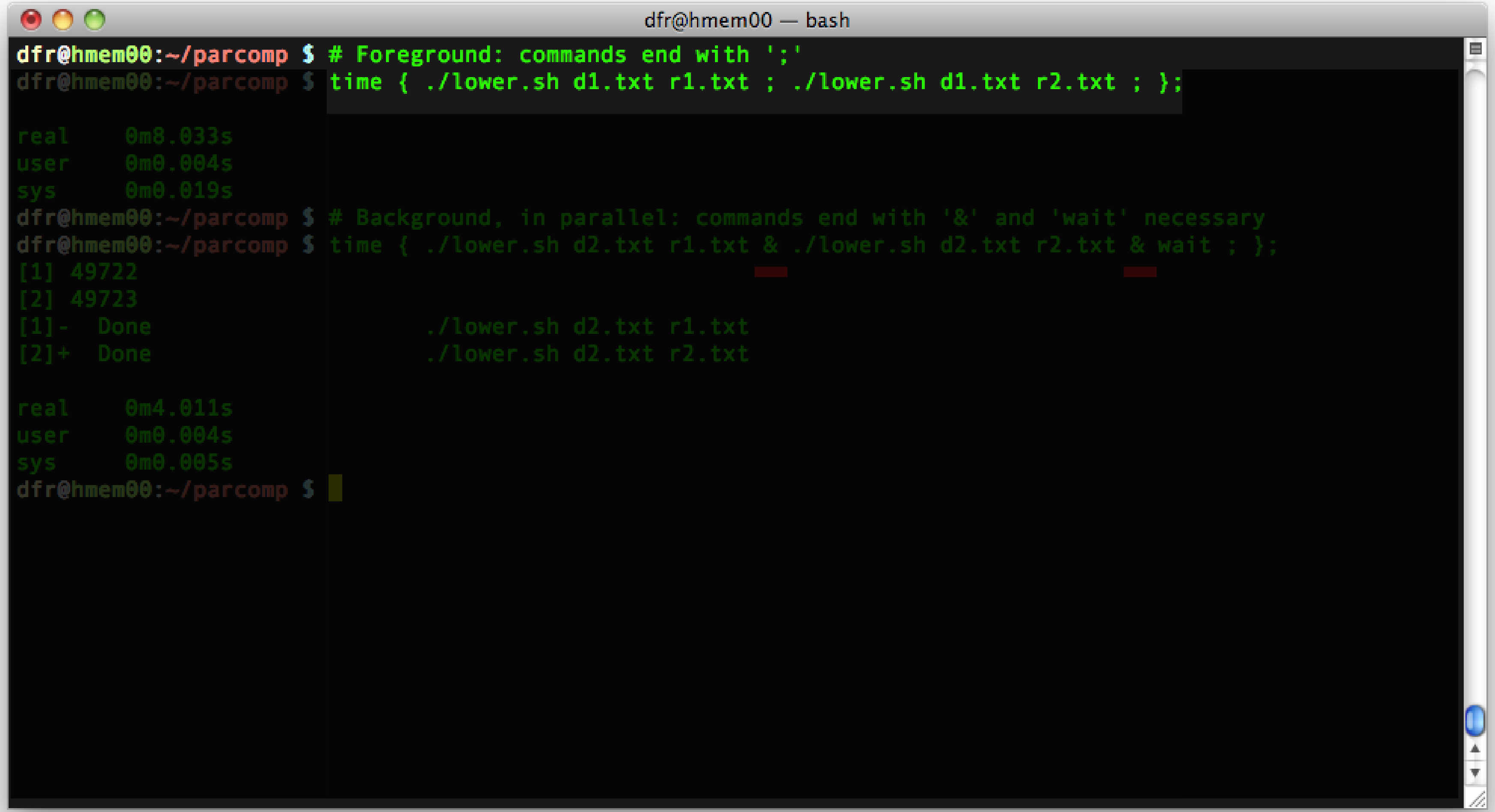


```
dfr@hmem00:~/parcomp $ # Foreground: commands end with ';'
dfr@hmem00:~/parcomp $ time { ./lower.sh d1.txt r1.txt ; ./lower.sh d1.txt r2.txt ; };

real	0m8.033s
user	0m0.004s
sys	0m0.019s
dfr@hmem00:~/parcomp $ # Background, in parallel: commands end with '&' and 'wait' necessary
dfr@hmem00:~/parcomp $ time { ./lower.sh d2.txt r1.txt & ./lower.sh d2.txt r2.txt & wait ; };
[1] 49722
[2] 49723
[1]-  Done                    ./lower.sh d2.txt r1.txt
[2]+  Done                    ./lower.sh d2.txt r2.txt

real	0m4.011s
user	0m0.004s
sys	0m0.005s
dfr@hmem00:~/parcomp $
```

https://www.gnu.org/software/bash/manual/html_node/Job-Control-Basics.html

4.2. Job control & Parallel processes in Bash

Run the program twice and measure the time it takes

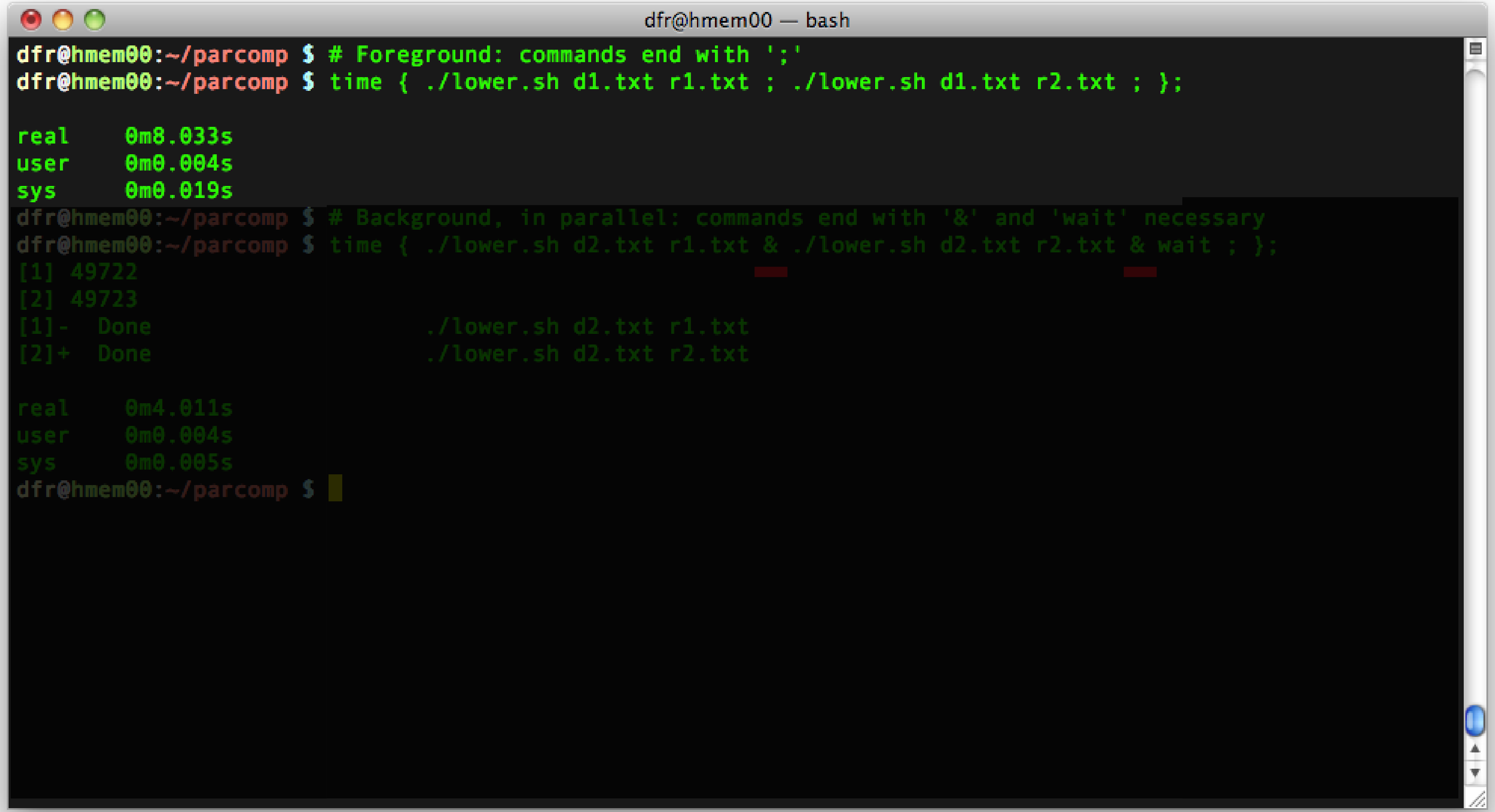


```
dfr@hmem00:~/parcomp $ # Foreground: commands end with ';'
dfr@hmem00:~/parcomp $ time { ./lower.sh d1.txt r1.txt ; ./lower.sh d1.txt r2.txt ; };

real    0m8.033s
user    0m0.004s
sys     0m0.019s
dfr@hmem00:~/parcomp $ # Background, in parallel: commands end with '&' and 'wait' necessary
dfr@hmem00:~/parcomp $ time { ./lower.sh d2.txt r1.txt & ./lower.sh d2.txt r2.txt & wait ; };
[1] 49722
[2] 49723
[1]-  Done                    ./lower.sh d2.txt r1.txt
[2]+  Done                    ./lower.sh d2.txt r2.txt

real    0m4.011s
user    0m0.004s
sys     0m0.005s
dfr@hmem00:~/parcomp $
```

https://www.gnu.org/software/bash/manual/html_node/Job-Control-Basics.html

4.2. Job control & Parallel processes in Bash

Run the program twice and measure the time it takes



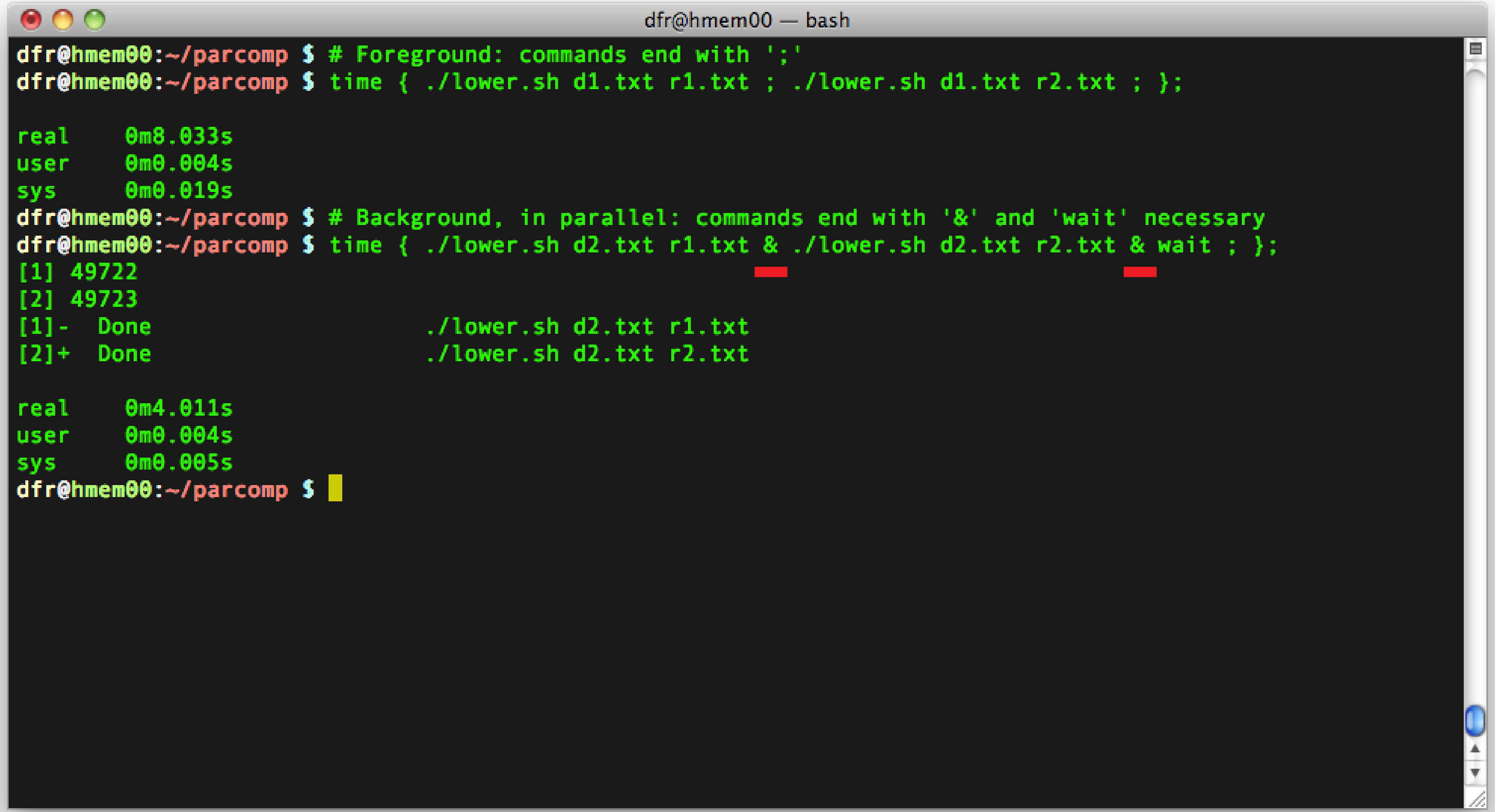
dfr@hmem00 — bash

```
dfr@hmem00:~/parcomp $ # Foreground: commands end with ';'
dfr@hmem00:~/parcomp $ time { ./lower.sh d1.txt r1.txt ; ./lower.sh d1.txt r2.txt ; };

real	0m8.033s
user	0m0.004s
sys	0m0.019s
dfr@hmem00:~/parcomp $ # Background, in parallel: commands end with '&' and 'wait' necessary
dfr@hmem00:~/parcomp $ time { ./lower.sh d2.txt r1.txt & ./lower.sh d2.txt r2.txt & wait ; };
[1] 49722
[2] 49723
[1]-  Done                    ./lower.sh d2.txt r1.txt
[2]+  Done                    ./lower.sh d2.txt r2.txt

real	0m4.011s
user	0m0.004s
sys	0m0.005s
dfr@hmem00:~/parcomp $
```

https://www.gnu.org/software/bash/manual/html_node/Job-Control-Basics.html

4.2. Job control & Parallel processes in Bash

Run the program twice “in the background” and measure the time



dfr@hmem00 — bash

```
dfr@hmem00:~/parcomp $ # Foreground: commands end with ';'
dfr@hmem00:~/parcomp $ time { ./lower.sh d1.txt r1.txt ; ./lower.sh d1.txt r2.txt ; };

real	0m8.033s
user	0m0.004s
sys	0m0.019s
dfr@hmem00:~/parcomp $ # Background, in parallel: commands end with '&' and 'wait' necessary
dfr@hmem00:~/parcomp $ time { ./lower.sh d2.txt r1.txt & ./lower.sh d2.txt r2.txt & wait ; };
[1] 49722
[2] 49723
[1]-  Done                    ./lower.sh d2.txt r1.txt
[2]+  Done                    ./lower.sh d2.txt r2.txt

real	0m4.011s
user	0m0.004s
sys	0m0.005s
dfr@hmem00:~/parcomp $ 
```

https://www.gnu.org/software/bash/manual/html_node/Job-Control-Basics.html

4.2. Job control & Parallel processes in Bash

Parallel for loop in Bash:

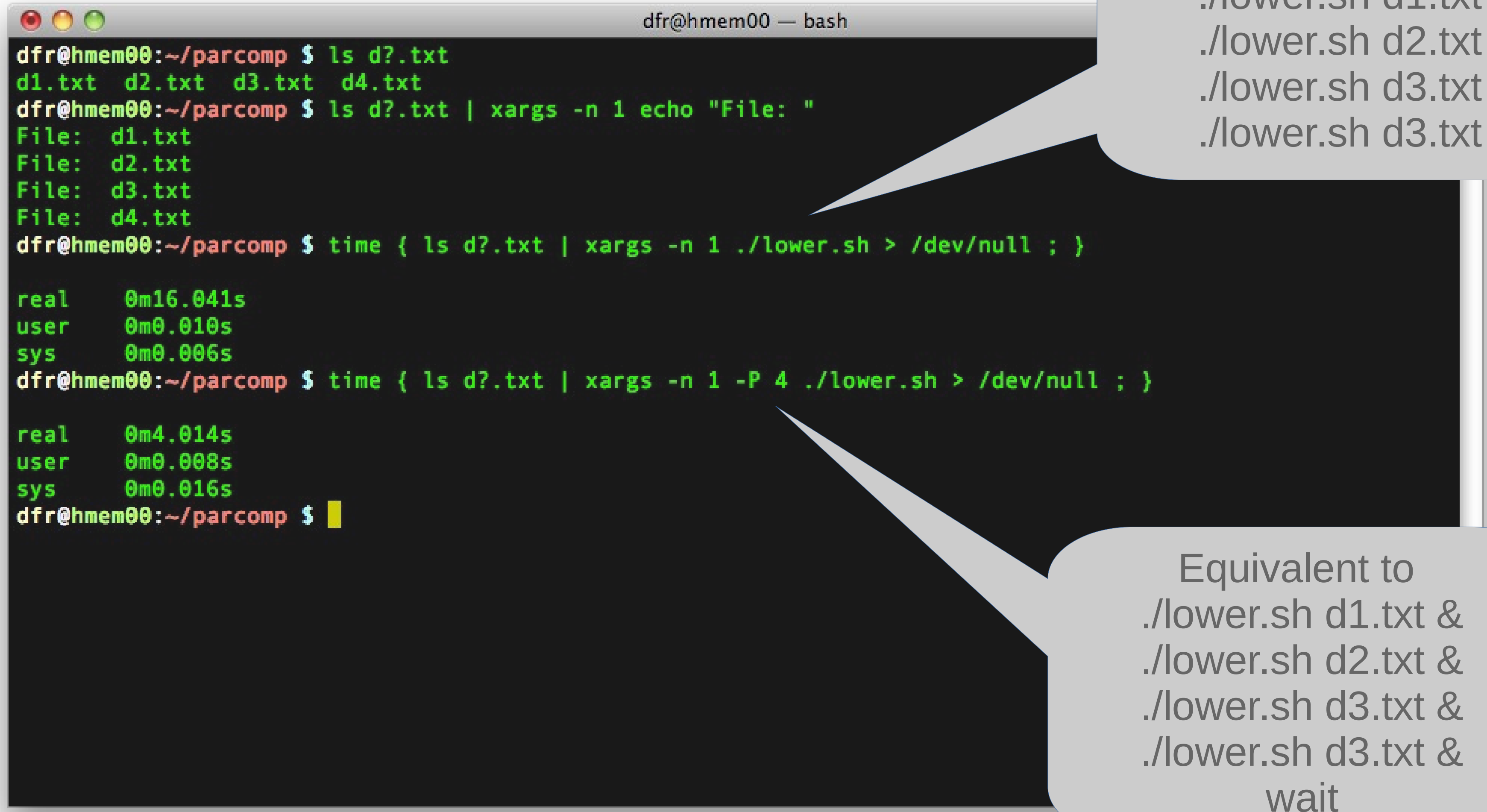
```
for i in {1..10}; do
  command1
  command2
done
```

```
for i in {1..10}; do
(
  command1
  command2
) &
done; wait
```

`(...) &` : creates a sub-shell with all commands in the bloc and start it in the background
`wait` : *barrier* to synchronize all sub-shells

4.3.1. One program and many files

The xargs command distributes data from stdin to program



```
dfr@hmem00:~/parcomp $ ls d?.txt
d1.txt  d2.txt  d3.txt  d4.txt
dfr@hmem00:~/parcomp $ ls d?.txt | xargs -n 1 echo "File: "
File:  d1.txt
File:  d2.txt
File:  d3.txt
File:  d4.txt
dfr@hmem00:~/parcomp $ time { ls d?.txt | xargs -n 1 ./lower.sh > /dev/null ; }

real    0m16.041s
user    0m0.010s
sys     0m0.006s
dfr@hmem00:~/parcomp $ time { ls d?.txt | xargs -n 1 -P 4 ./lower.sh > /dev/null ; }

real    0m4.014s
user    0m0.008s
sys     0m0.016s
dfr@hmem00:~/parcomp $
```

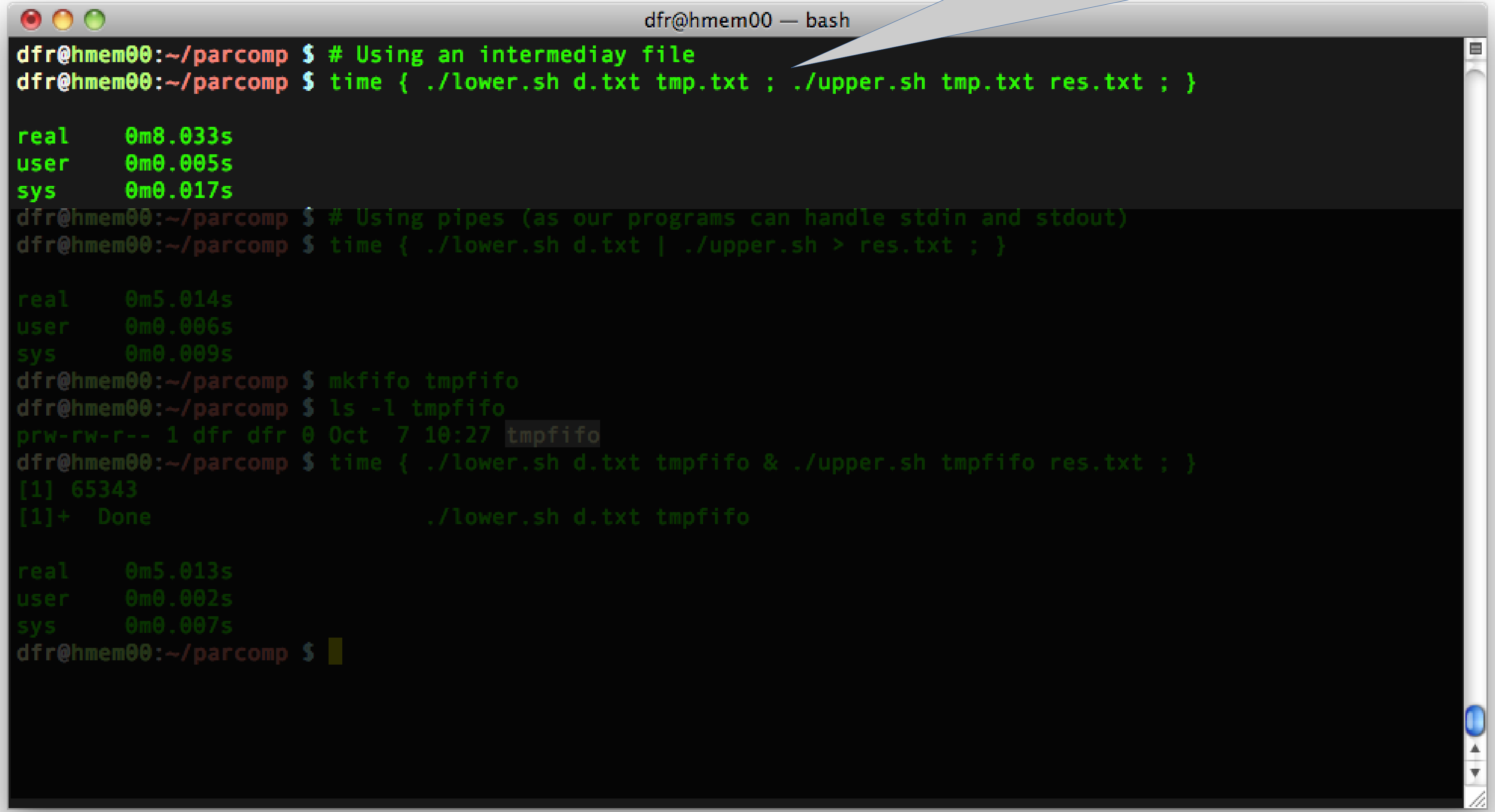
Equivalent to
./lower.sh d1.txt ;
./lower.sh d2.txt ;
./lower.sh d3.txt ;
./lower.sh d3.txt ;

Equivalent to
./lower.sh d1.txt &
./lower.sh d2.txt &
./lower.sh d3.txt &
./lower.sh d3.txt &
wait

4.3.2. Several programs and one file

Using UNIX pipes for pipelining operations

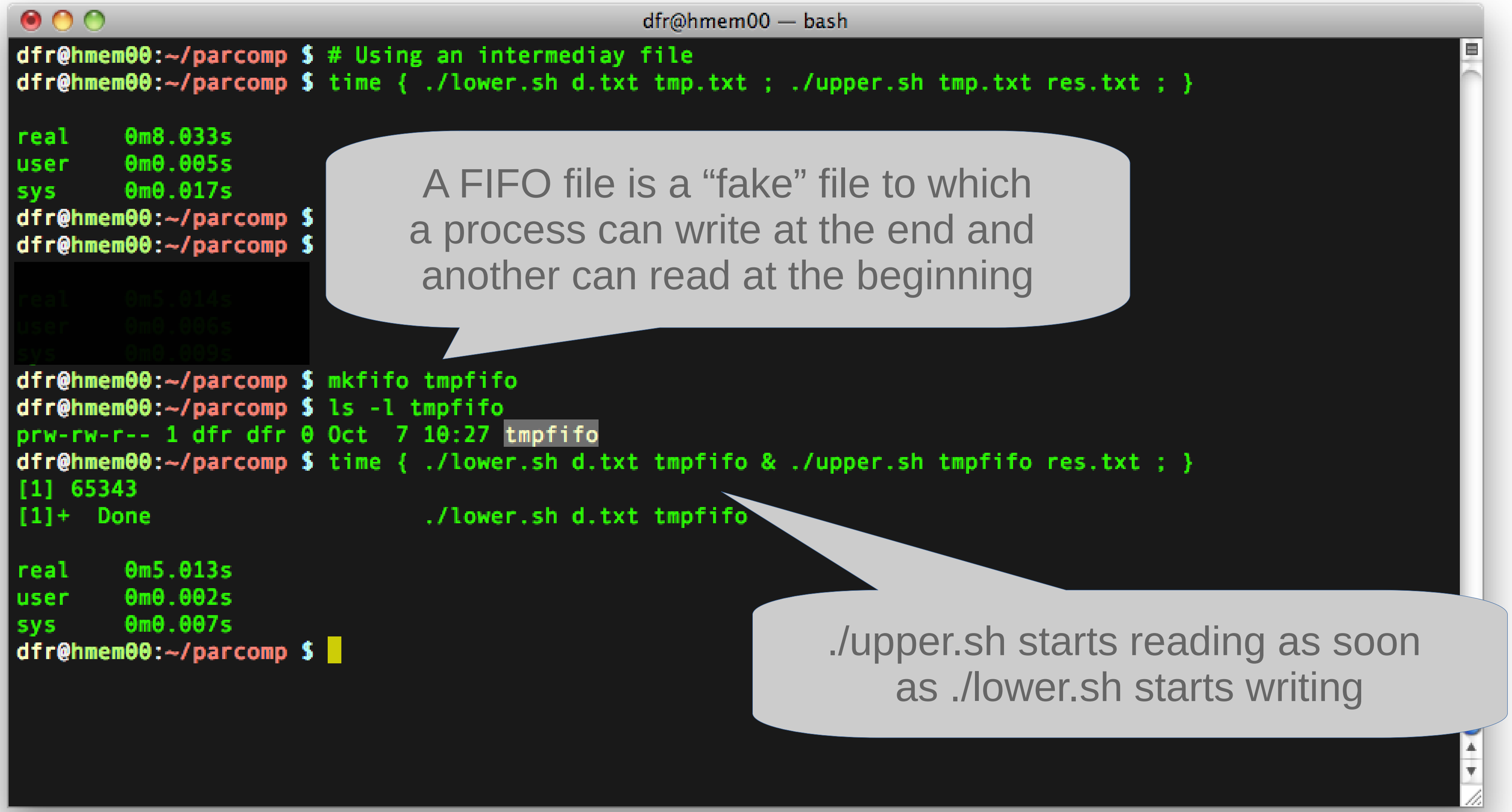
./upper.sh waits for ./lower.sh to finish
Note the intermediate file



```
dfr@hmem00:~/parcomp $ # Using an intermediay file
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmp.txt ; ./upper.sh tmp.txt res.txt ; }

real    0m8.033s
user    0m0.005s
sys     0m0.017s
dfr@hmem00:~/parcomp $ # Using pipes (as our programs can handle stdin and stdout)
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt | ./upper.sh > res.txt ; }

real    0m5.014s
user    0m0.006s
sys     0m0.009s
dfr@hmem00:~/parcomp $ mkfifo tmpfifo
dfr@hmem00:~/parcomp $ ls -l tmpfifo
prw-rw-r-- 1 dfr dfr 0 Oct  7 10:27 tmpfifo
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmpfifo & ./upper.sh tmpfifo res.txt ; }
[1] 65343
[1]+  Done                    ./lower.sh d.txt tmpfifo

real    0m5.013s
user    0m0.002s
sys     0m0.007s
dfr@hmem00:~/parcomp $
```

4.3.2. Several programs and one file

Using UNIX fifos for pipelining operations

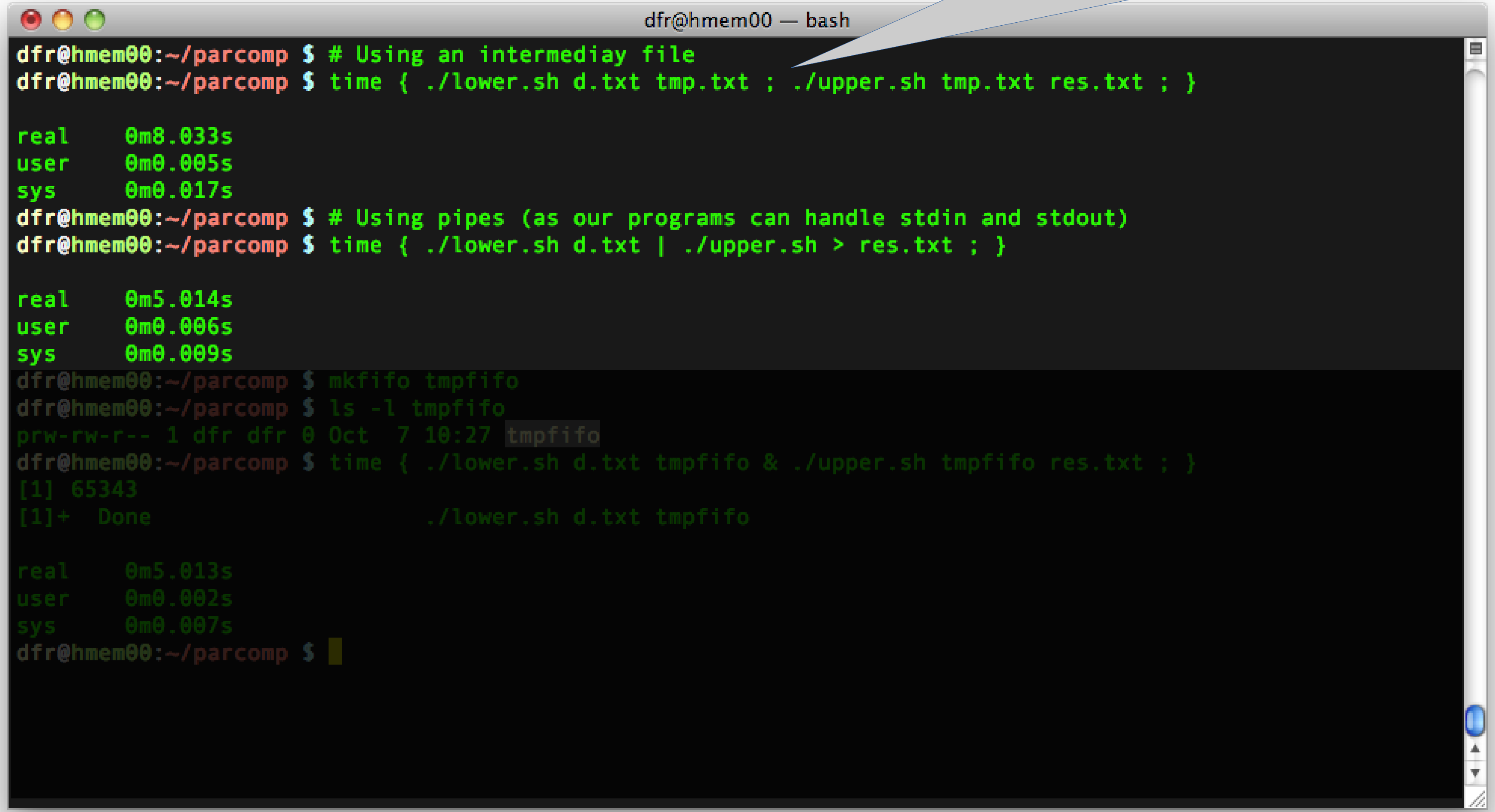

dfr@hmem00 — bash
A FIFO file is a “fake” file to which a process can write at the end and another can read at the beginning
./upper.sh starts reading as soon as ./lower.sh starts writing

```
dfr@hmem00:~/parcomp $ # Using an intermediay file
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmp.txt ; ./upper.sh tmp.txt res.txt ; }

real    0m8.033s
user    0m0.005s
sys     0m0.017s
dfr@hmem00:~/parcomp $
dfr@hmem00:~/parcomp $
[redacted]
dfr@hmem00:~/parcomp $ mkfifo tmpfifo
dfr@hmem00:~/parcomp $ ls -l tmpfifo
prw-rw-r-- 1 dfr dfr 0 Oct  7 10:27 tmpfifo
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmpfifo & ./upper.sh tmpfifo res.txt ; }
[1] 65343
[1]+  Done                    ./lower.sh d.txt tmpfifo

real    0m5.013s
user    0m0.002s
sys     0m0.007s
dfr@hmem00:~/parcomp $
```

4.3.2. Several programs and one file

Using UNIX pipes for pipelining operations

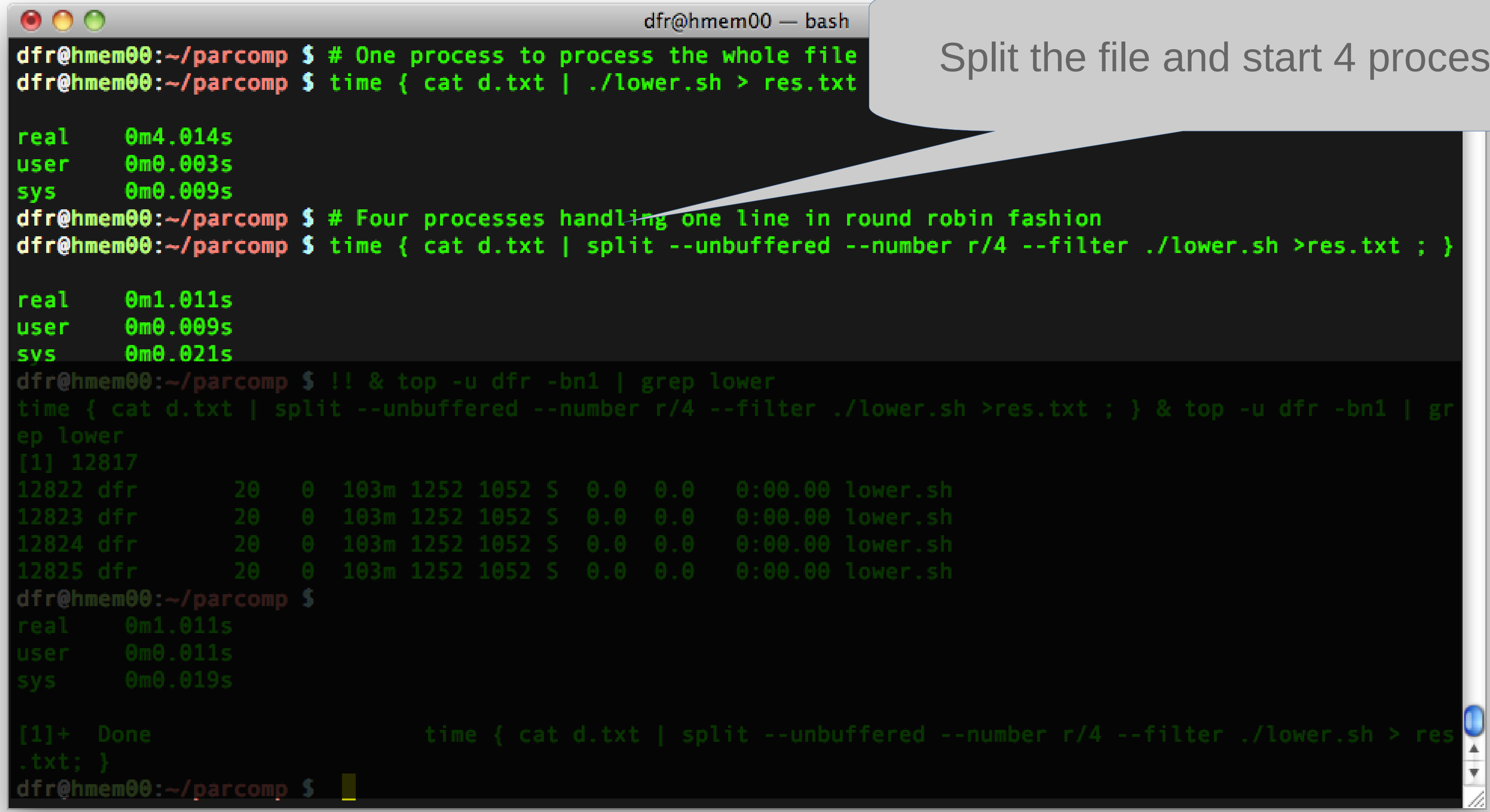
./upper.sh waits for ./lower.sh to finish
Note the intermediate file



```
dfr@hmem00:~/parcomp $ # Using an intermediay file
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmp.txt ; ./upper.sh tmp.txt res.txt ; }

real    0m8.033s
user    0m0.005s
sys     0m0.017s
dfr@hmem00:~/parcomp $ # Using pipes (as our programs can handle stdin and stdout)
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt | ./upper.sh > res.txt ; }

real    0m5.014s
user    0m0.006s
sys     0m0.009s
dfr@hmem00:~/parcomp $ mkfifo tmpfifo
dfr@hmem00:~/parcomp $ ls -l tmpfifo
prw-rw-r-- 1 dfr dfr 0 Oct  7 10:27 tmpfifo
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmpfifo & ./upper.sh tmpfifo res.txt ; }
[1] 65343
[1]+  Done                    ./lower.sh d.txt tmpfifo

real    0m5.013s
user    0m0.002s
sys     0m0.007s
dfr@hmem00:~/parcomp $
```

4.3.3. One program and one large file

The split command distributes data from stdin to program



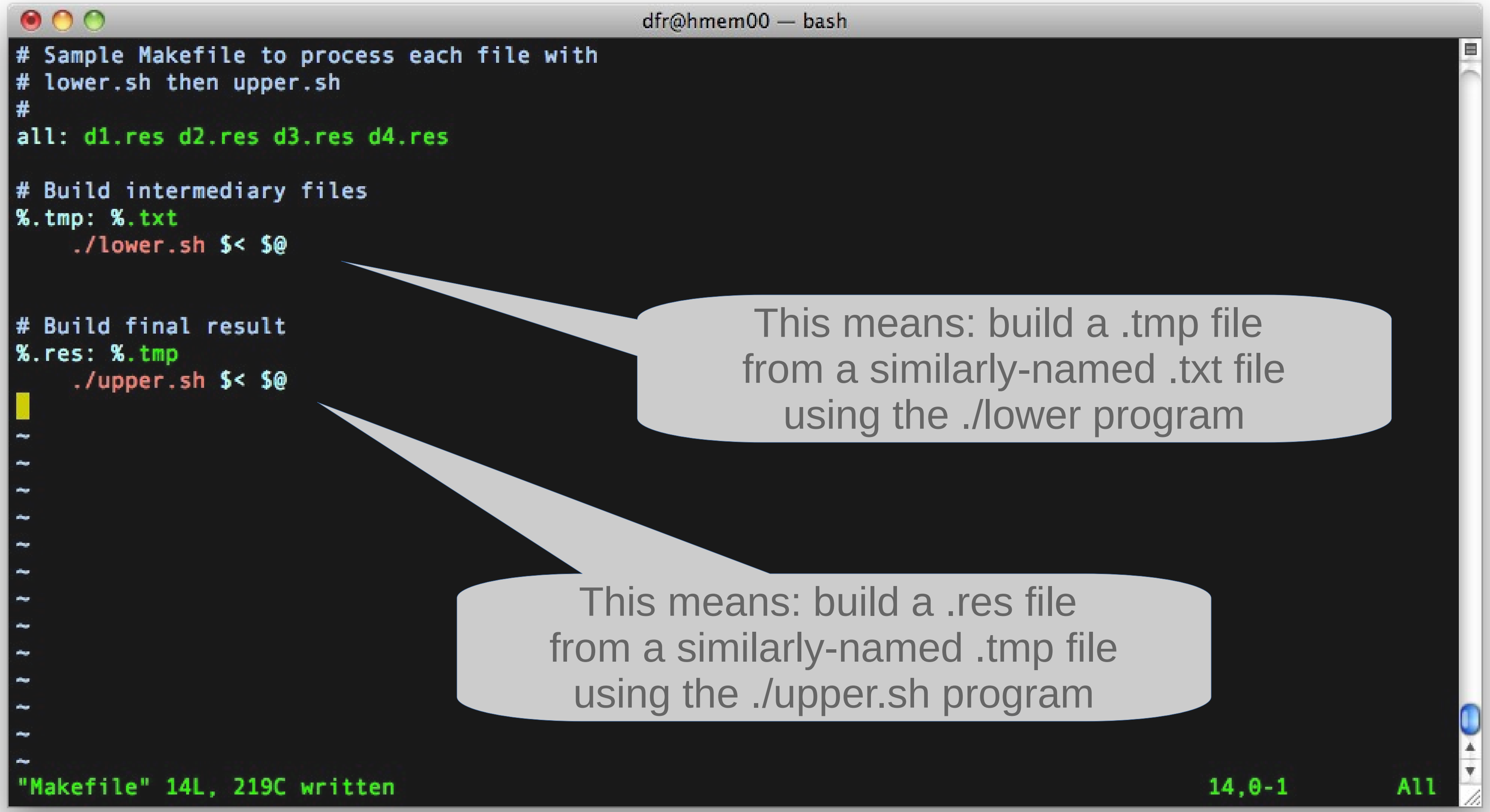
```
dfr@hmem00:~/parcomp $ # One process to process the whole file
dfr@hmem00:~/parcomp $ time { cat d.txt | ./lower.sh > res.txt

real    0m4.014s
user    0m0.003s
sys     0m0.009s
dfr@hmem00:~/parcomp $ # Four processes handling one line in round robin fashion
dfr@hmem00:~/parcomp $ time { cat d.txt | split --unbuffered --number r/4 --filter ./lower.sh >res.txt ; }

real    0m1.011s
user    0m0.009s
sys     0m0.021s
dfr@hmem00:~/parcomp $ !! & top -u dfr -bn1 | grep lower
time { cat d.txt | split --unbuffered --number r/4 --filter ./lower.sh >res.txt ; } & top -u dfr -bn1 | gr
ep lower
[1] 12817
12822 dfr       20   0  103m 1252 1052 S  0.0  0.0   0:00.00 lower.sh
12823 dfr       20   0  103m 1252 1052 S  0.0  0.0   0:00.00 lower.sh
12824 dfr       20   0  103m 1252 1052 S  0.0  0.0   0:00.00 lower.sh
12825 dfr       20   0  103m 1252 1052 S  0.0  0.0   0:00.00 lower.sh
dfr@hmem00:~/parcomp $
real    0m1.011s
user    0m0.011s
sys     0m0.019s

[1]+  Done                    time { cat d.txt | split --unbuffered --number r/4 --filter ./lower.sh > res
.txt; }
dfr@hmem00:~/parcomp $ 
```

Split the file and start 4 processes

Need recent version of Coreutils/8.22-goolf-1.4.10

4.3.4. Several programs and many files

A Makefile describes dependencies and is executed with ‘make’



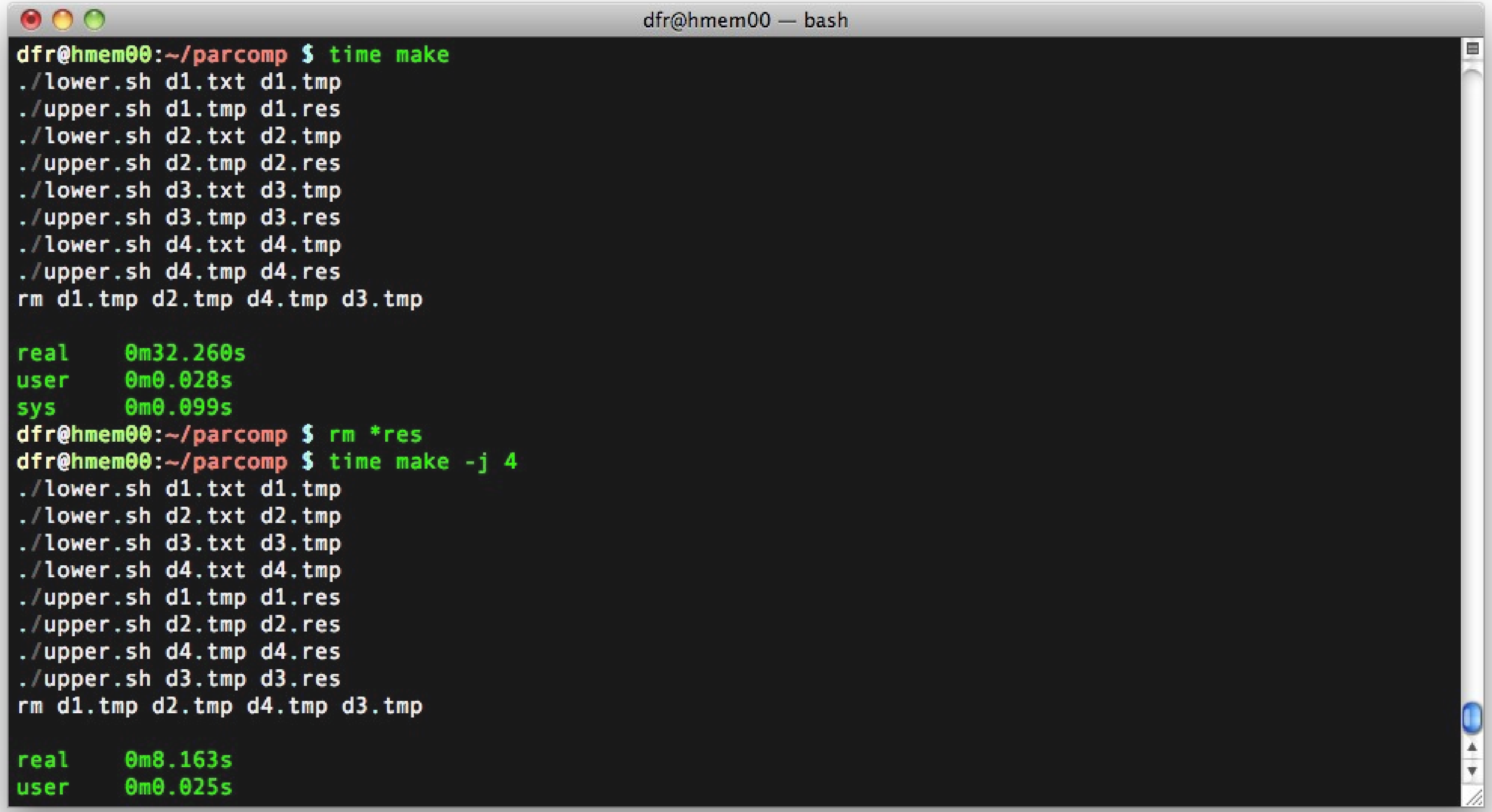
```
# Sample Makefile to process each file with
# lower.sh then upper.sh
#
all: d1.res d2.res d3.res d4.res

# Build intermediary files
%.tmp: %.txt
    ./lower.sh $< $@


# Build final result
%.res: %.tmp
    ./upper.sh $< $@

~
~
~
~
~
~
~
~
~
~
~
~
~
"Makefile" 14L, 219C written                                              14,0-1        All
```

This means: build a .tmp file from a similarly-named .txt file using the ./lower program
This means: build a .res file from a similarly-named .tmp file using the ./upper.sh program

https://www.gnu.org/software/make/manual/html_node/index.html

4.3.4. Several programs and many files

The ‘make’ command can operate in parallel



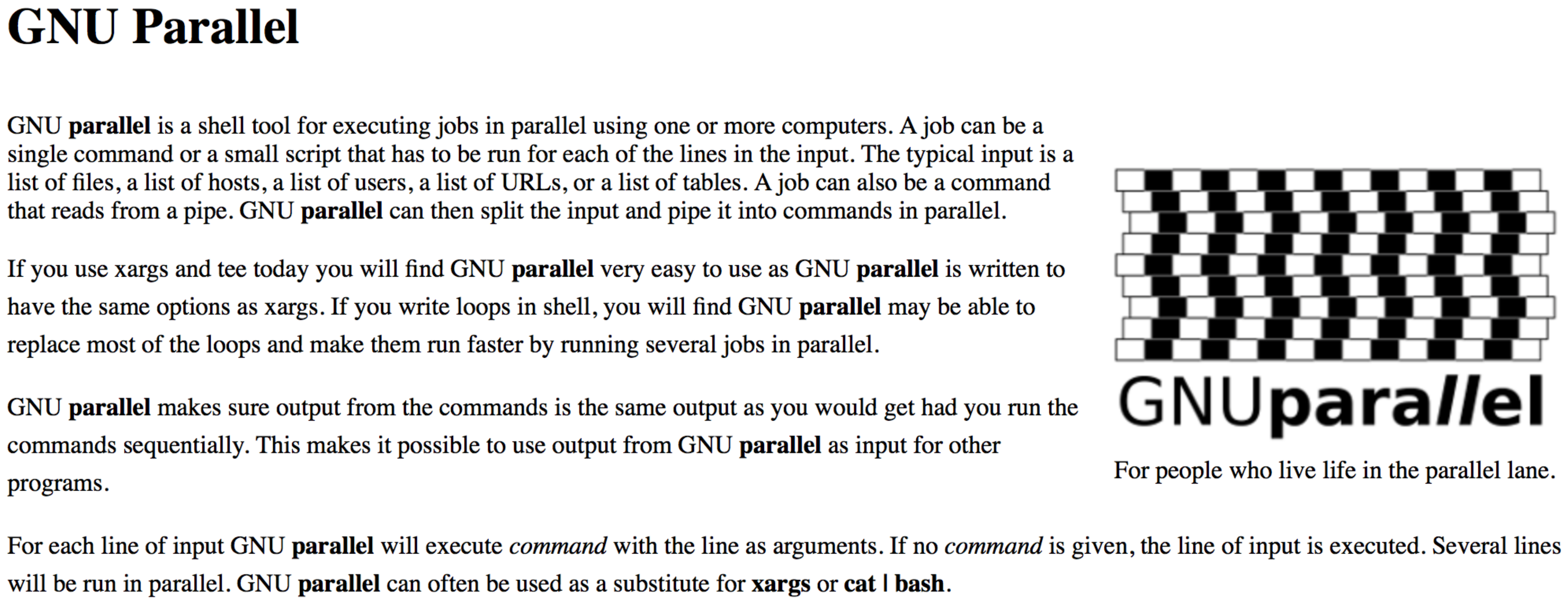
dfr@hmem00 — bash

```
dfr@hmem00:~/parcomp $ time make
./lower.sh d1.txt d1.tmp
./upper.sh d1.tmp d1.res
./lower.sh d2.txt d2.tmp
./upper.sh d2.tmp d2.res
./lower.sh d3.txt d3.tmp
./upper.sh d3.tmp d3.res
./lower.sh d4.txt d4.tmp
./upper.sh d4.tmp d4.res
rm d1.tmp d2.tmp d4.tmp d3.tmp

real    0m32.260s
user    0m0.028s
sys     0m0.099s
dfr@hmem00:~/parcomp $ rm *res
dfr@hmem00:~/parcomp $ time make -j 4
./lower.sh d1.txt d1.tmp
./lower.sh d2.txt d2.tmp
./lower.sh d3.txt d3.tmp
./lower.sh d4.txt d4.tmp
./upper.sh d1.tmp d1.res
./upper.sh d2.tmp d2.res
./upper.sh d4.tmp d4.res
./upper.sh d3.tmp d3.res
rm d1.tmp d2.tmp d4.tmp d3.tmp

real    0m8.163s
user    0m0.025s
```

Summary

- You have either
  - one very large file to process
    - with one program: split
    - with several programs: fifo (or pipes)
  - many files to process
    - with one program xargs
    - with many programs make

4.4. GNU Parallel



GNU Parallel

GNU **parallel** is a shell tool for executing jobs in parallel using one or more computers. 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. A job can also be a command that reads from a pipe. GNU **parallel** can then split the input and pipe it into commands in parallel.

If you use xargs and tee today you will find GNU **parallel** very easy to use as GNU **parallel** is written to have the same options as xargs. If you write loops in shell, you will find GNU **parallel** may be able to replace most of the loops and make them run faster by running several jobs in parallel.

GNU **parallel** makes sure output from the commands is the same output as you would get had you run the commands sequentially. This makes it possible to use output from GNU **parallel** as input for other programs.

GNUparallel

For people who live life in the parallel lane.

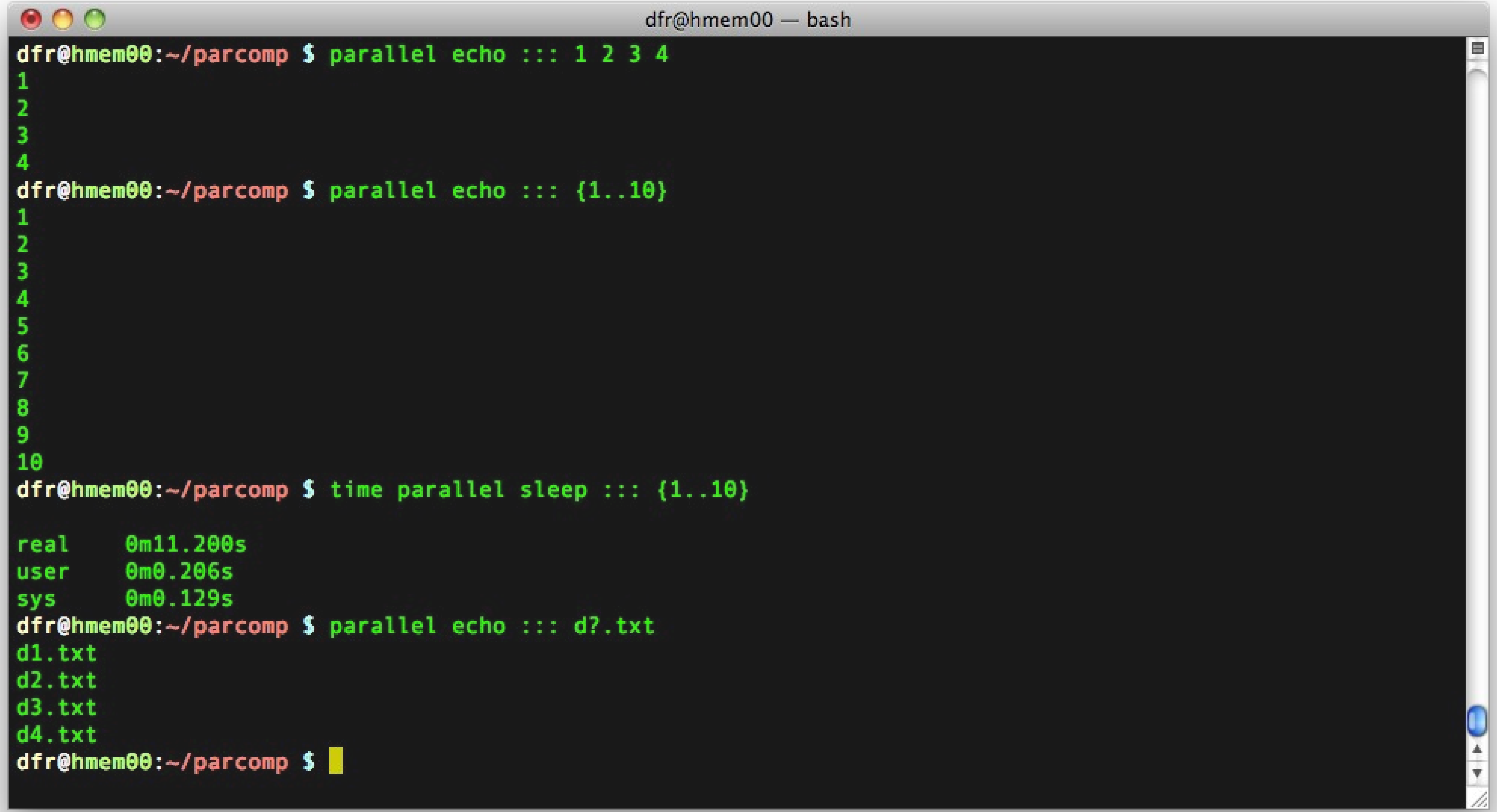
For each line of input GNU **parallel** will execute *command* with the line as arguments. If no *command* is given, the line of input is executed. Several lines will be run in parallel. GNU **parallel** can often be used as a substitute for **xargs** or **cat | bash**.

More complicated to use but very powerful

Might not be available everywhere but you can install it

4.4. GNU Parallel

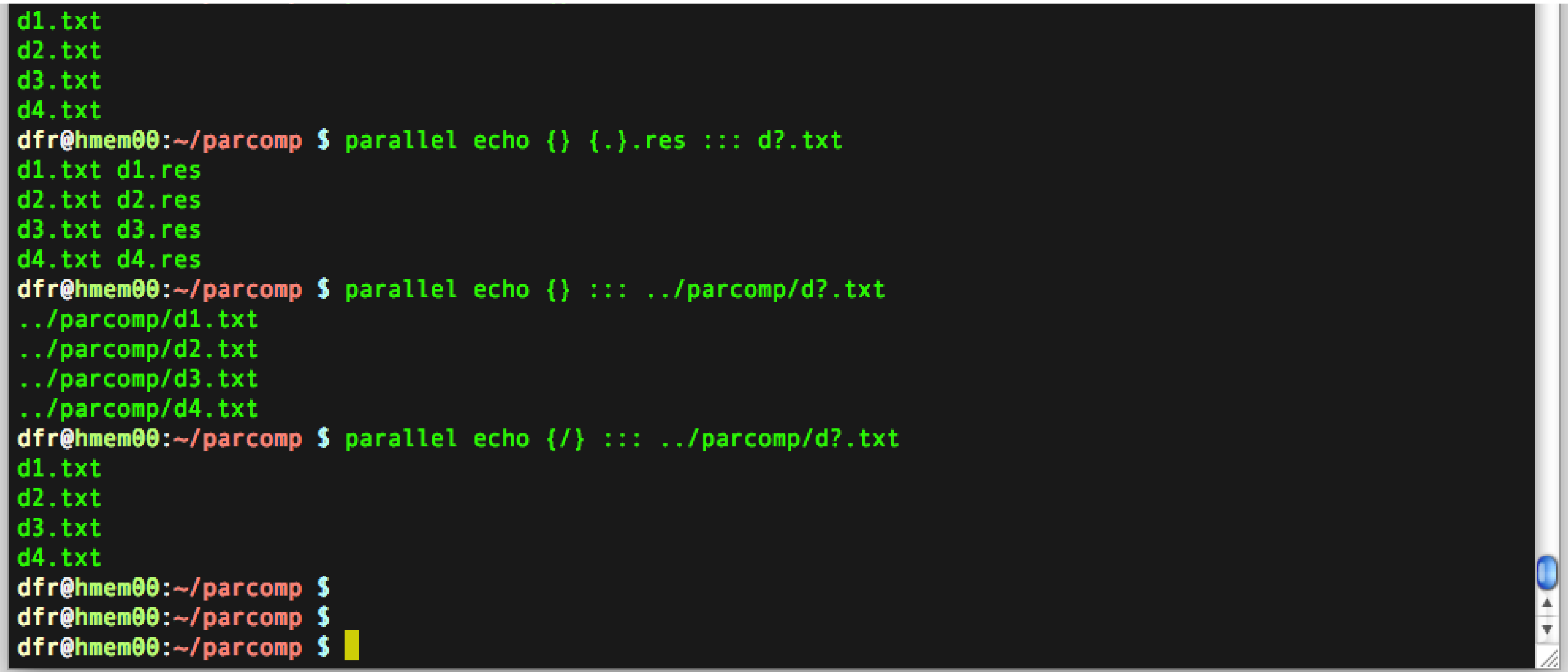
- Syntax: parallel *command* ::: *argument list*



```
dfr@hmem00:~/parcomp $ parallel echo ::: 1 2 3 4
1
2
3
4
dfr@hmem00:~/parcomp $ parallel echo ::: {1..10}
1
2
3
4
5
6
7
8
9
10
dfr@hmem00:~/parcomp $ time parallel sleep ::: {1..10}

real    0m11.200s
user    0m0.206s
sys     0m0.129s
dfr@hmem00:~/parcomp $ parallel echo ::: d?.txt
d1.txt
d2.txt
d3.txt
d4.txt
dfr@hmem00:~/parcomp $
```

4.4. GNU Parallel

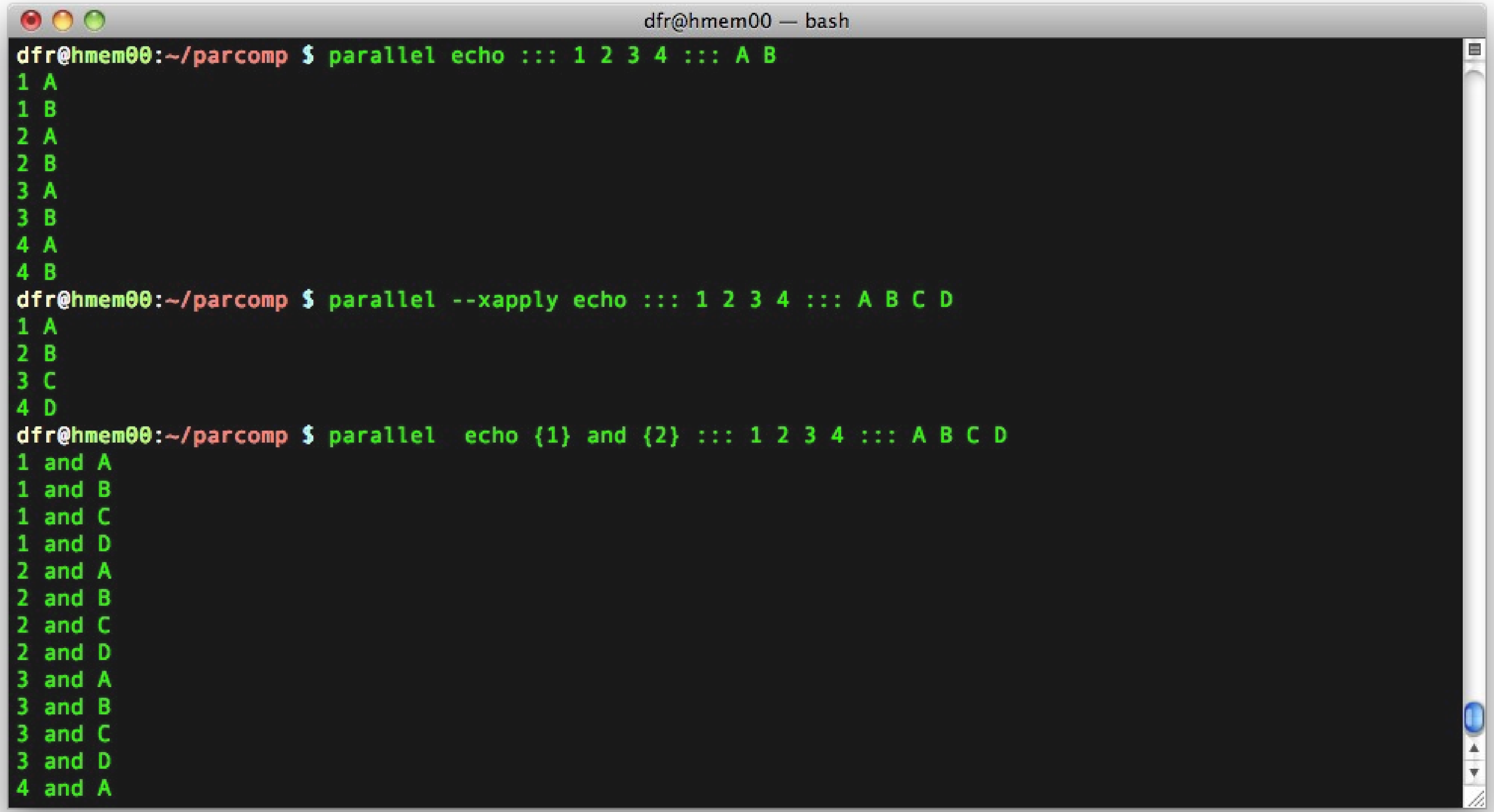
- Syntax: {} as argument placeholder.



```
d1.txt
d2.txt
d3.txt
d4.txt
dfr@hmem00:~/parcomp $ parallel echo {} {.}.res ::: d?.txt
d1.txt d1.res
d2.txt d2.res
d3.txt d3.res
d4.txt d4.res
dfr@hmem00:~/parcomp $ parallel echo {} ::: ../parcomp/d?.txt
../parcomp/d1.txt
../parcomp/d2.txt
../parcomp/d3.txt
../parcomp/d4.txt
dfr@hmem00:~/parcomp $ parallel echo {/} ::: ../parcomp/d?.txt
d1.txt
d2.txt
d3.txt
d4.txt
dfr@hmem00:~/parcomp $
dfr@hmem00:~/parcomp $
dfr@hmem00:~/parcomp $
```

4.4. GNU Parallel

- Multiple parameters and --xapply

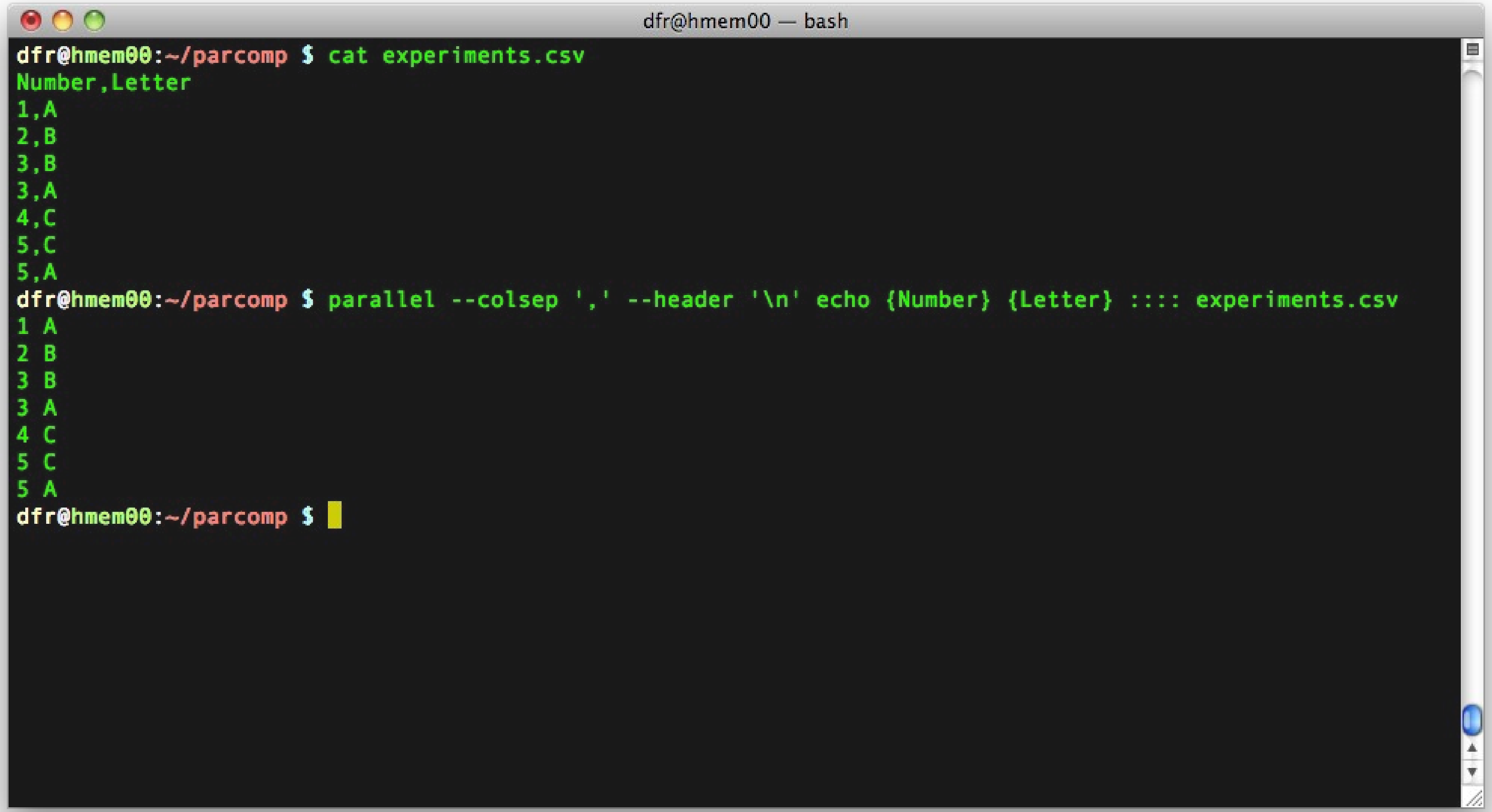


dfr@hmem00 — bash

```
dfr@hmem00:~/parcomp $ parallel echo ::: 1 2 3 4 ::: A B
1 A
1 B
2 A
2 B
3 A
3 B
4 A
4 B
dfr@hmem00:~/parcomp $ parallel --xapply echo ::: 1 2 3 4 ::: A B C D
1 A
2 B
3 C
4 D
dfr@hmem00:~/parcomp $ parallel  echo {1} and {2} ::: 1 2 3 4 ::: A B C D
1 and A
1 and B
1 and C
1 and D
2 and A
2 and B
2 and C
2 and D
3 and A
3 and B
3 and C
3 and D
4 and A
```

4.4. GNU Parallel

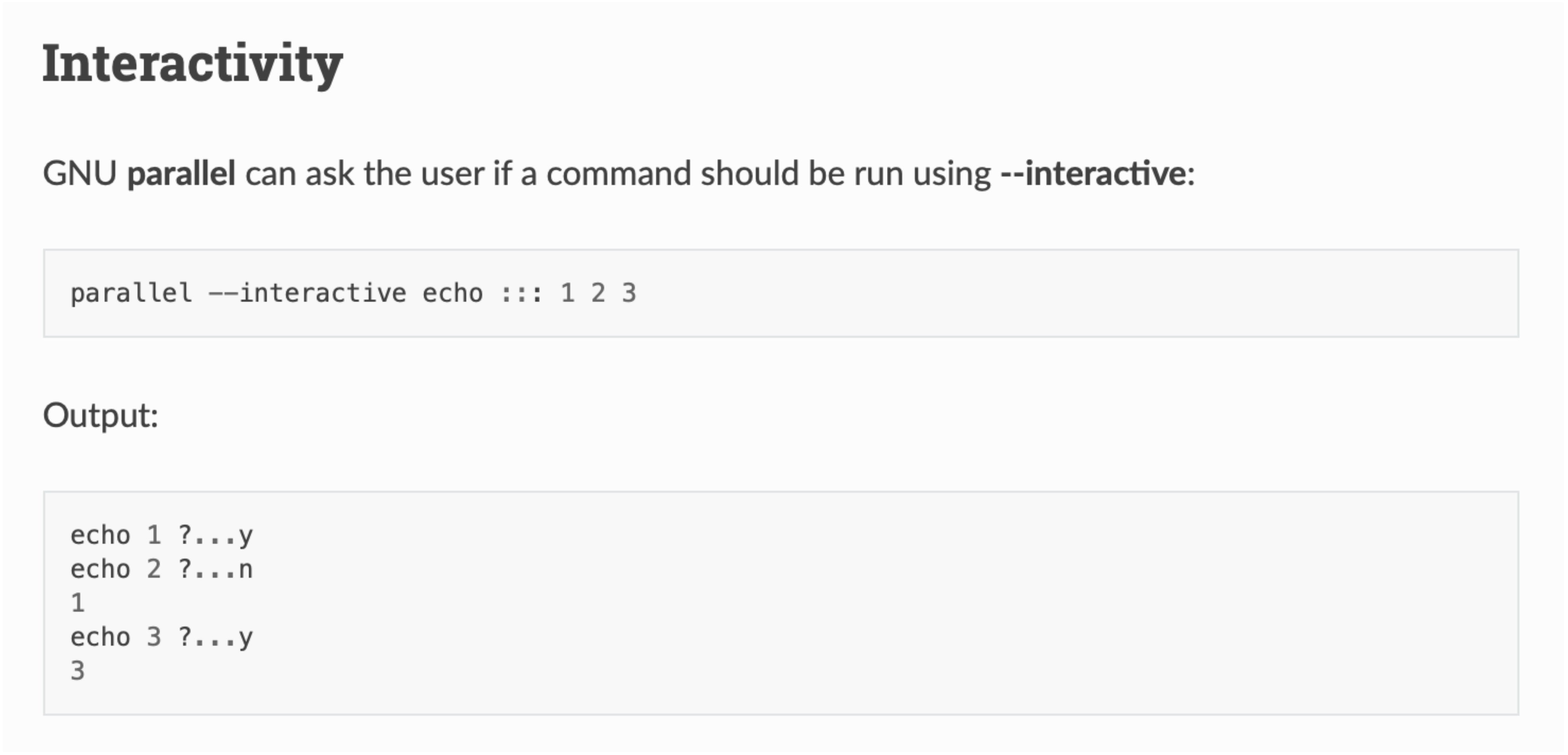
- When arguments are in a file : use :::: (4x ‘:’)



dfr@hmem00 — bash

```
dfr@hmem00:~/parcomp $ cat experiments.csv
Number,Letter
1,A
2,B
3,B
3,A
4,C
5,C
5,A
dfr@hmem00:~/parcomp $ parallel --colsep ',' --header '\n' echo {Number} {Letter} :::: experiments.csv
1 A
2 B
3 B
3 A
4 C
5 C
5 A
dfr@hmem00:~/parcomp $ 
```

4.4. GNU Parallel



Interactivity

GNU **parallel** can ask the user if a command should be run using **--interactive**:

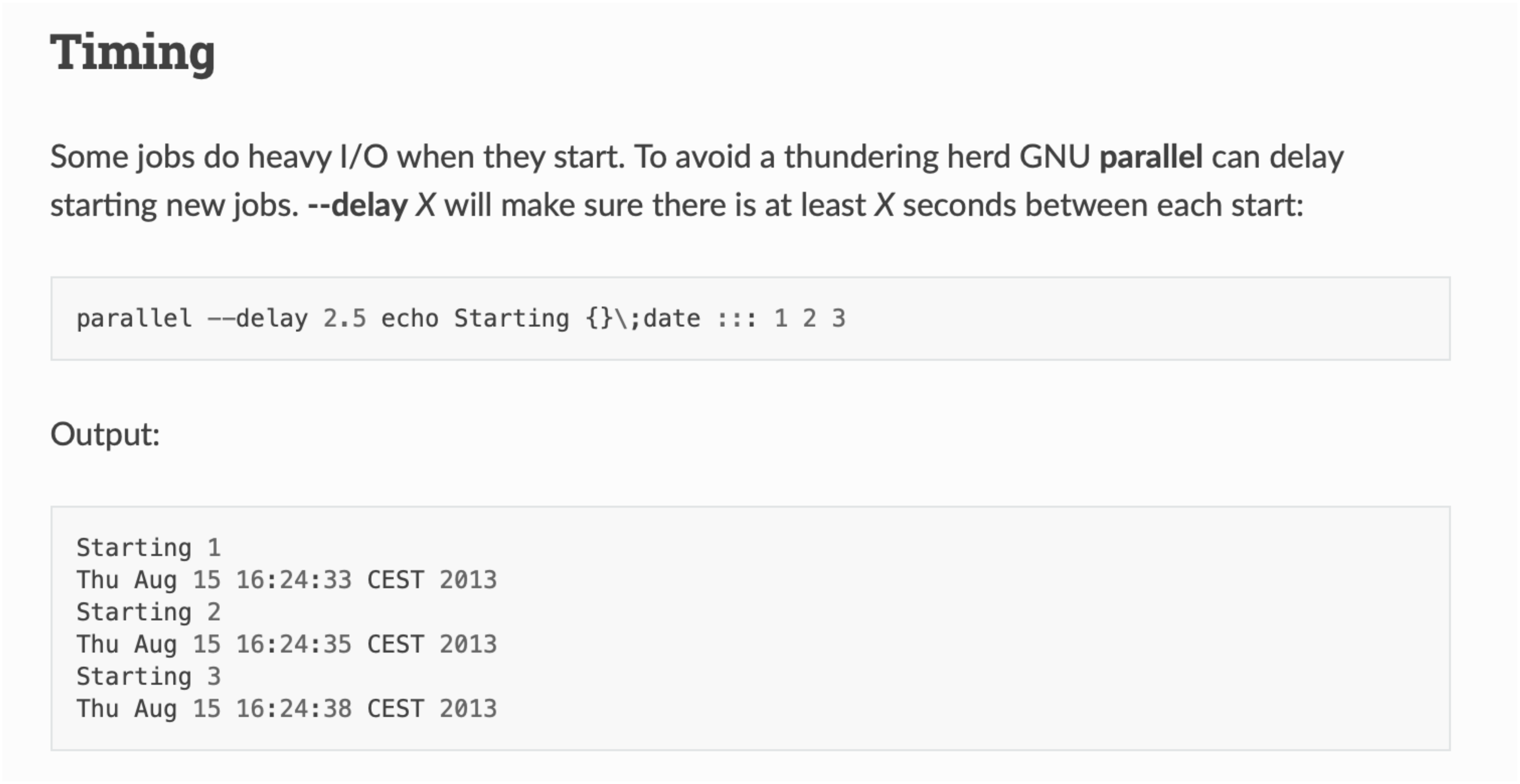
```
parallel --interactive echo ::: 1 2 3
```

Output:

```
echo 1 ?...y
echo 2 ?...n
1
echo 3 ?...y
3
```

https://www.gnu.org/software/parallel/parallel_tutorial.html

4.4. GNU Parallel


Timing

Some jobs do heavy I/O when they start. To avoid a thundering herd GNU **parallel** can delay starting new jobs. **--delay** *X* will make sure there is at least *X* seconds between each start:

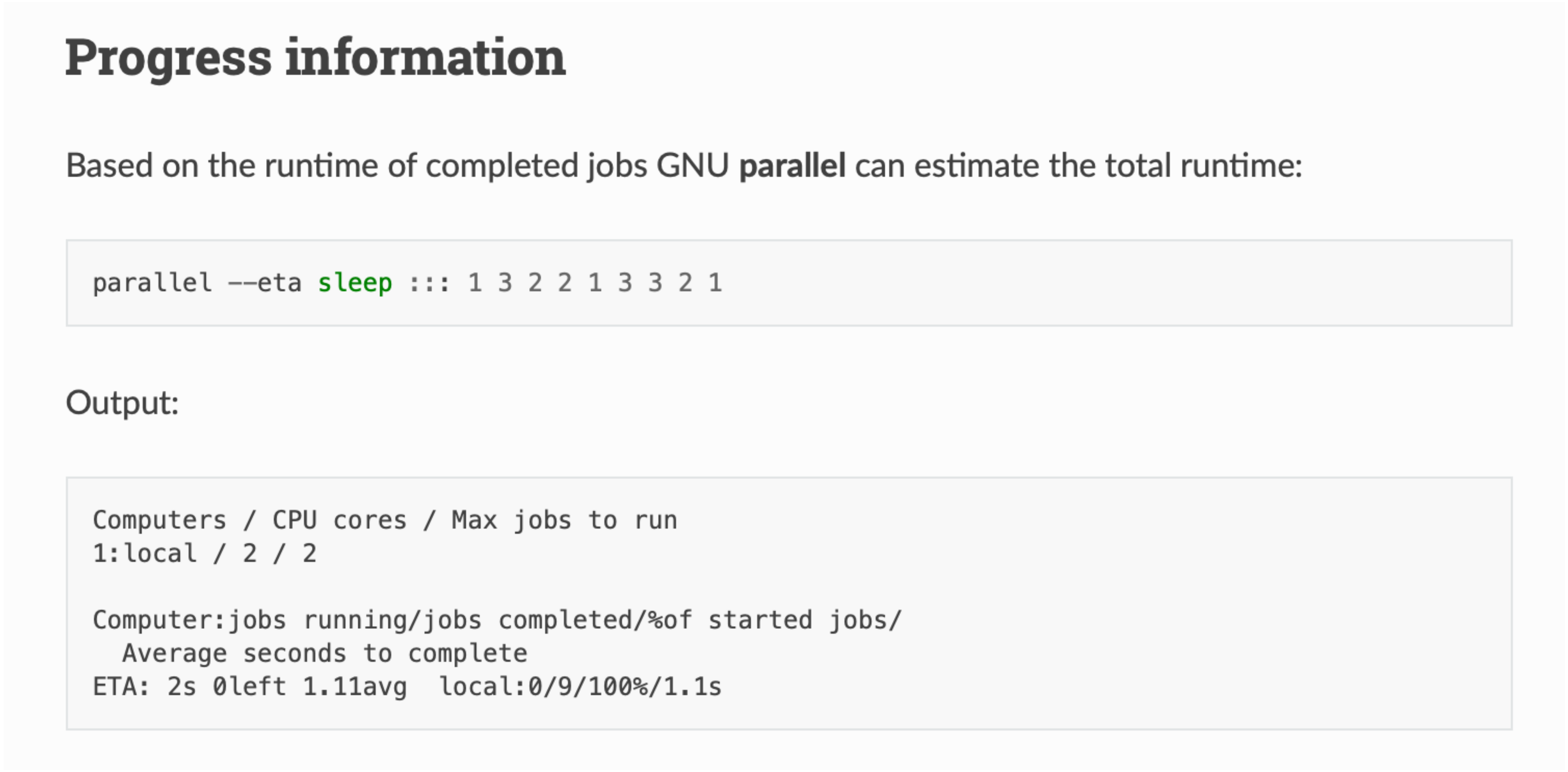
```
parallel --delay 2.5 echo Starting {}\;date ::: 1 2 3
```

Output:

```
Starting 1
Thu Aug 15 16:24:33 CEST 2013
Starting 2
Thu Aug 15 16:24:35 CEST 2013
Starting 3
Thu Aug 15 16:24:38 CEST 2013
```

https://www.gnu.org/software/parallel/parallel_tutorial.html

4.4. GNU Parallel



Progress information

Based on the runtime of completed jobs GNU **parallel** can estimate the total runtime:

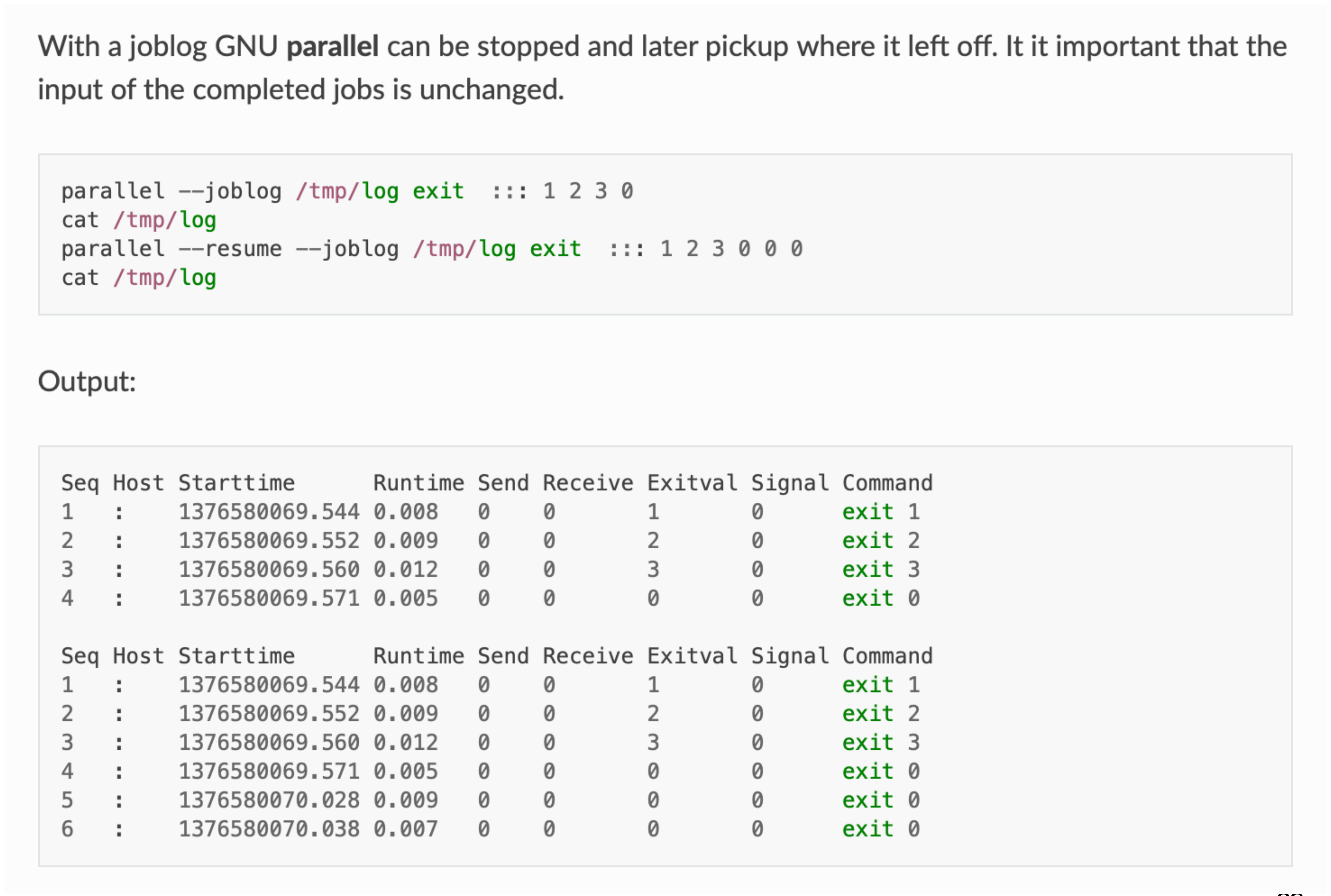
```
parallel --eta sleep ::: 1 3 2 2 1 3 3 2 1
```

Output:

```
Computers / CPU cores / Max jobs to run
1:local / 2 / 2

Computer:jobs running/jobs completed/%of started jobs/
  Average seconds to complete
ETA: 2s 0left 1.11avg  local:0/9/100%/1.1s
```

https://www.gnu.org/software/parallel/parallel_tutorial.html

4.4. GNU Parallel



With a joblog GNU **parallel** can be stopped and later pickup where it left off. It it important that the input of the completed jobs is unchanged.

```
parallel --joblog /tmp/log exit  ::: 1 2 3 0
cat /tmp/log
parallel --resume --joblog /tmp/log exit  ::: 1 2 3 0 0 0
cat /tmp/log
```

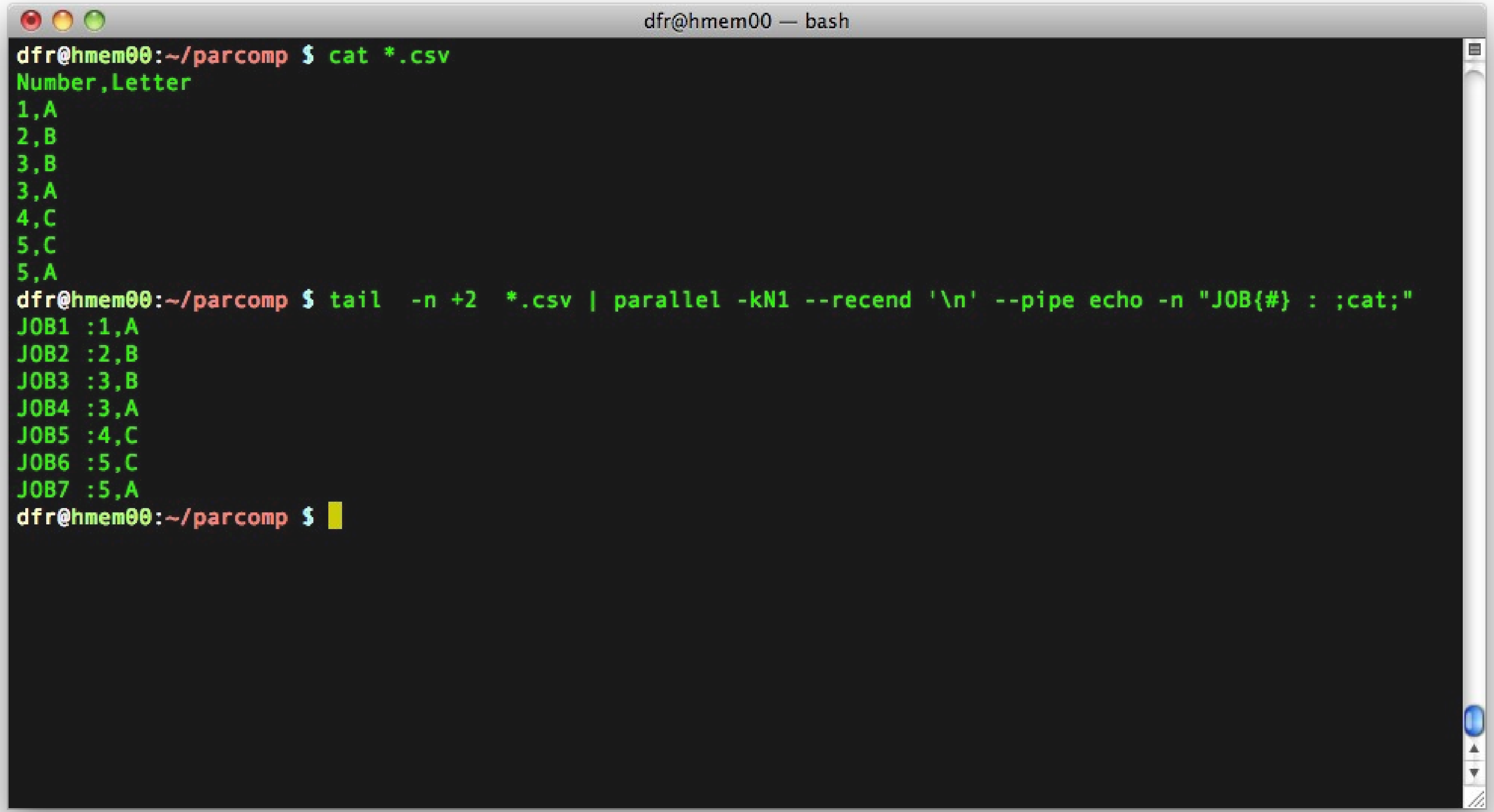
Output:

```
Seq Host Starttime      Runtime Send Receive Exitval Signal Command
1   :    1376580069.544 0.008   0    0       1       0      exit 1
2   :    1376580069.552 0.009   0    0       2       0      exit 2
3   :    1376580069.560 0.012   0    0       3       0      exit 3
4   :    1376580069.571 0.005   0    0       0       0      exit 0

Seq Host Starttime      Runtime Send Receive Exitval Signal Command
1   :    1376580069.544 0.008   0    0       1       0      exit 1
2   :    1376580069.552 0.009   0    0       2       0      exit 2
3   :    1376580069.560 0.012   0    0       3       0      exit 3
4   :    1376580069.571 0.005   0    0       0       0      exit 0
5   :    1376580070.028 0.009   0    0       0       0      exit 0
6   :    1376580070.038 0.007   0    0       0       0      exit 0
```

https://www.gnu.org/software/parallel/parallel_tutorial.html

4.4. GNU Parallel

- Split a file with --pipe


dfr@hmem00 — bash

```
dfr@hmem00:~/parcomp $ cat *.csv
Number,Letter
1,A
2,B
3,B
3,A
4,C
5,C
5,A
dfr@hmem00:~/parcomp $ tail  -n +2  *.csv | parallel -kN1 --recend '\n' --pipe echo -n "JOB{#} : ;cat;"
JOB1 :1,A
JOB2 :2,B
JOB3 :3,B
JOB4 :3,A
JOB5 :4,C
JOB6 :5,C
JOB7 :5,A
dfr@hmem00:~/parcomp $ 
```

Summary

1. General concepts, definitions, challenges
2. Hardware for parallel computing
3. Programming models
4. User tools