Consortium des Equipements
de Calcul Intensif
en Fédération Wallonie-Bruxelles

Introduction to Parallel Computing

damien.francois@uclouvain.be
October 2018

Agenda

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

1.

General concepts

Why parallel?

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)

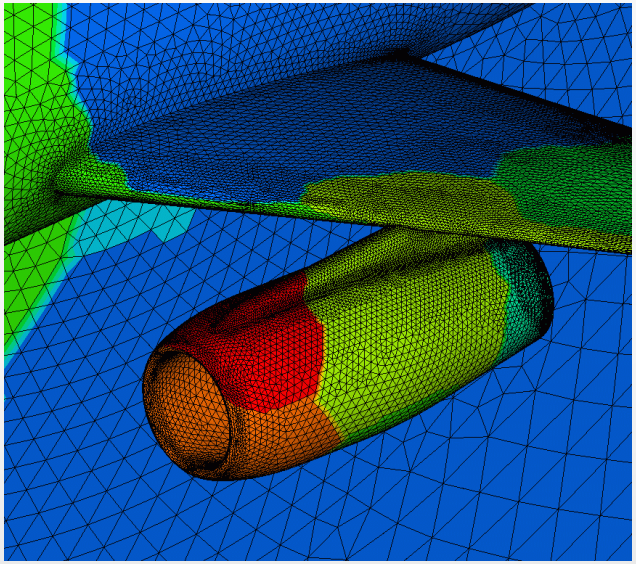
Scale out – Solve many problems
→ more storage capacity

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

Decomposition

- **Work 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



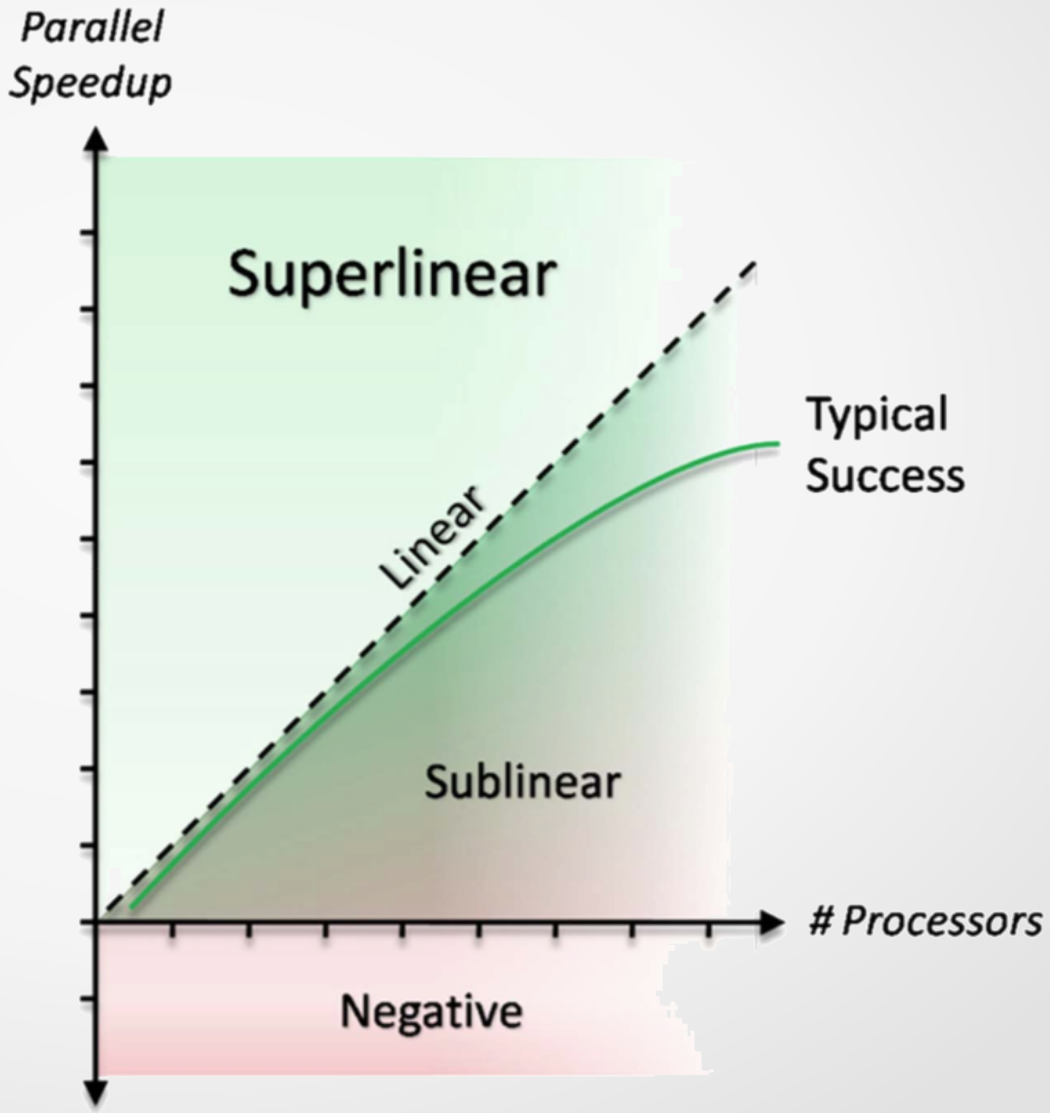
Collaboration

- **Synchronous** (SIMD) at the processor level
- **Fine-grained** parallelism if subtasks must communicate many times per second (instruction level); loosely synchronous
- **Coarse-grained** parallelism if they do not communicate many times per second (function-call level)
- **Embarrassingly parallel** if they rarely or never have to communicate (asynchronous)

Speedup, Efficiency, Scalability

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

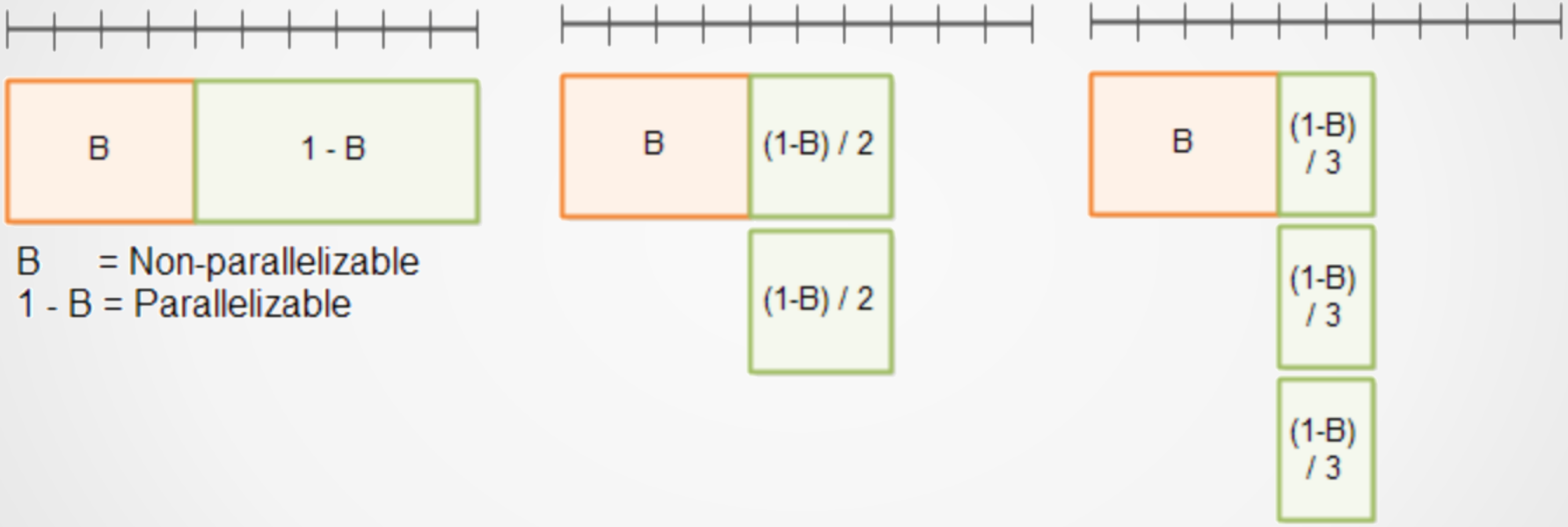
$$E = \frac{S}{p} = \frac{T_S}{pT_p}$$



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

Blocker 1: Amdahl's Law

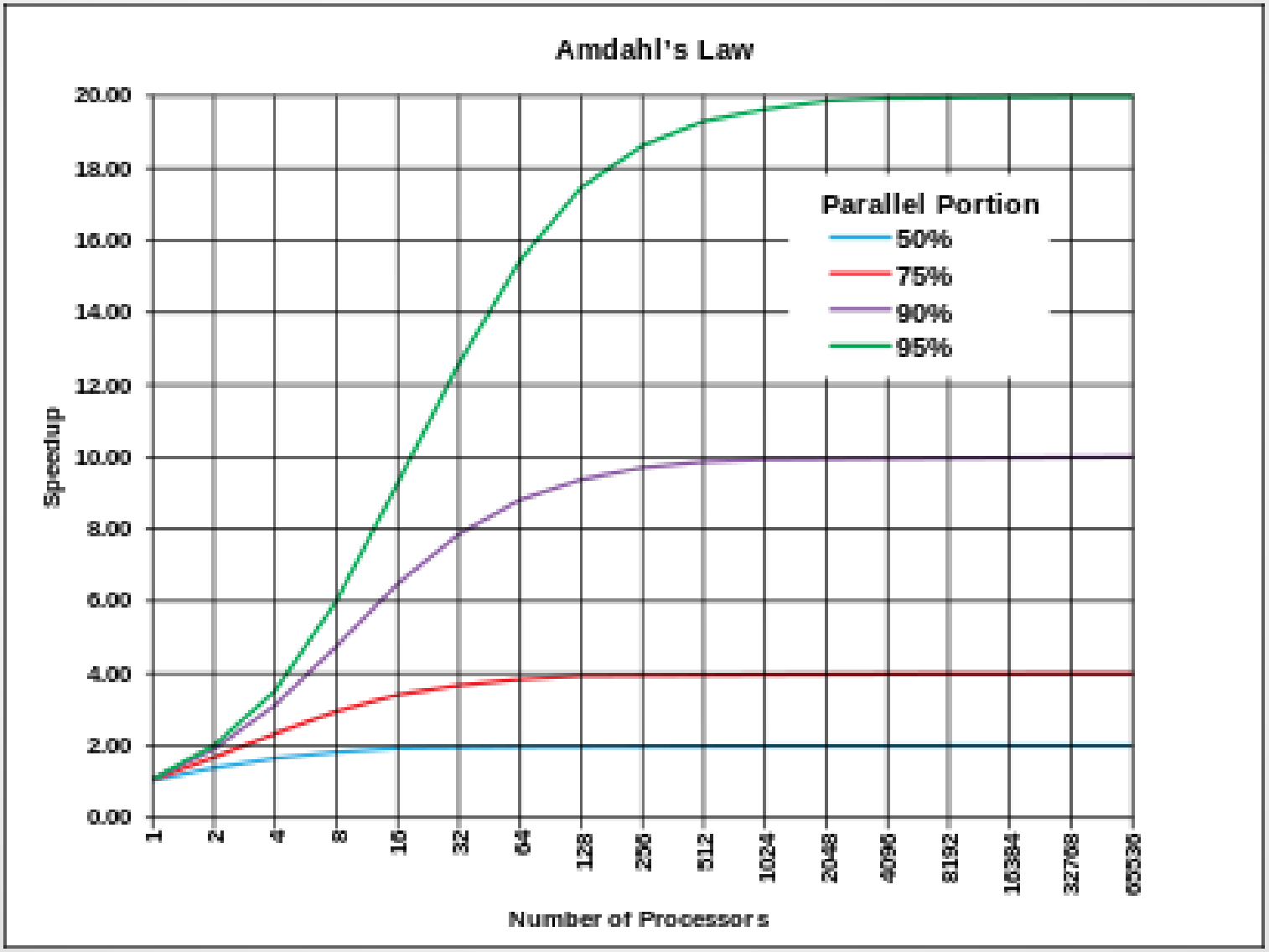
Often, not all the work can be decomposed



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

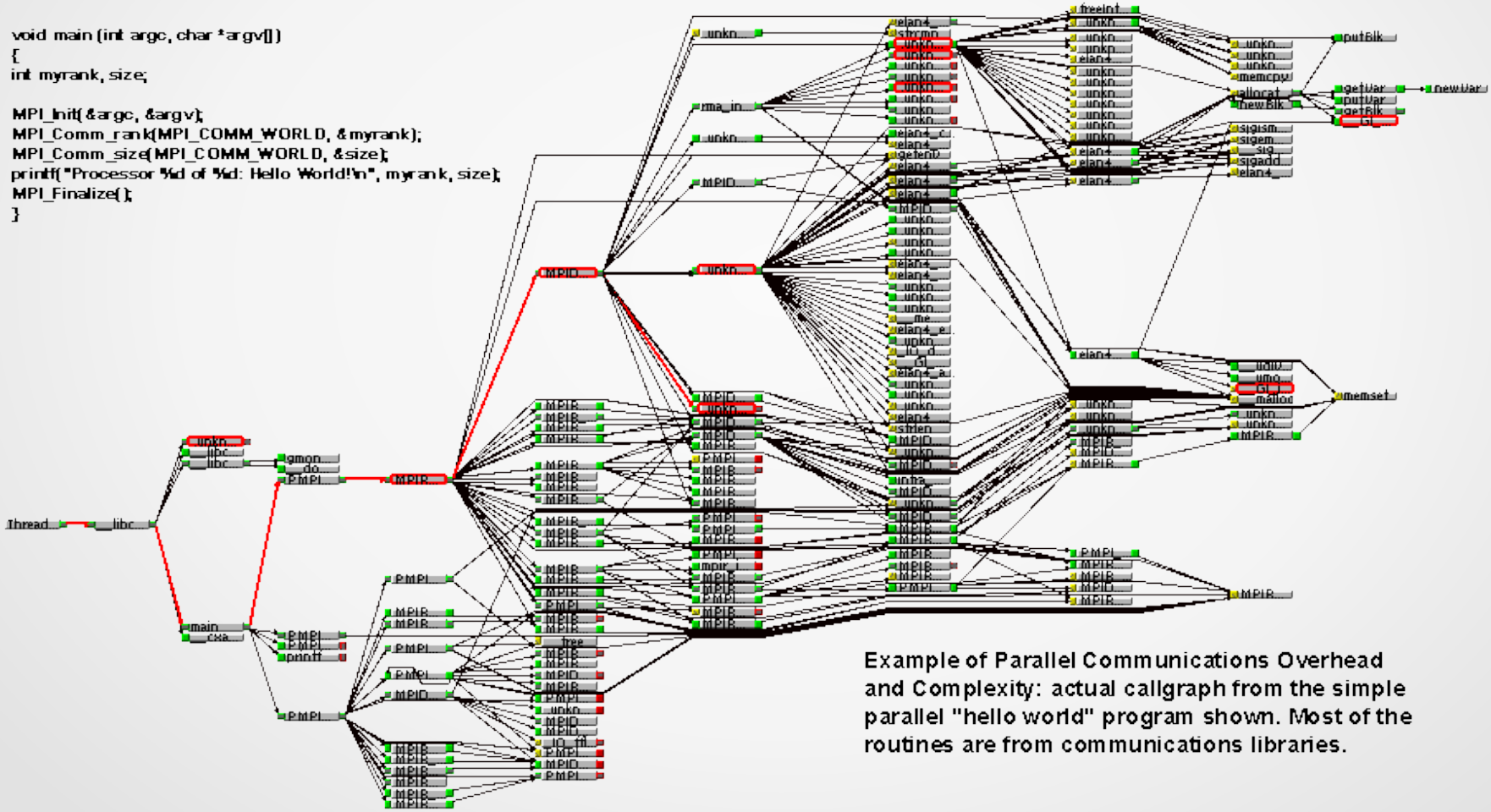
Blocker 1: Amdahl's Law



https://en.wikipedia.org/wiki/Amdahl%27s_law

Blocker 2: Parallel overhead

Collaboration means communication and a lot of extra work

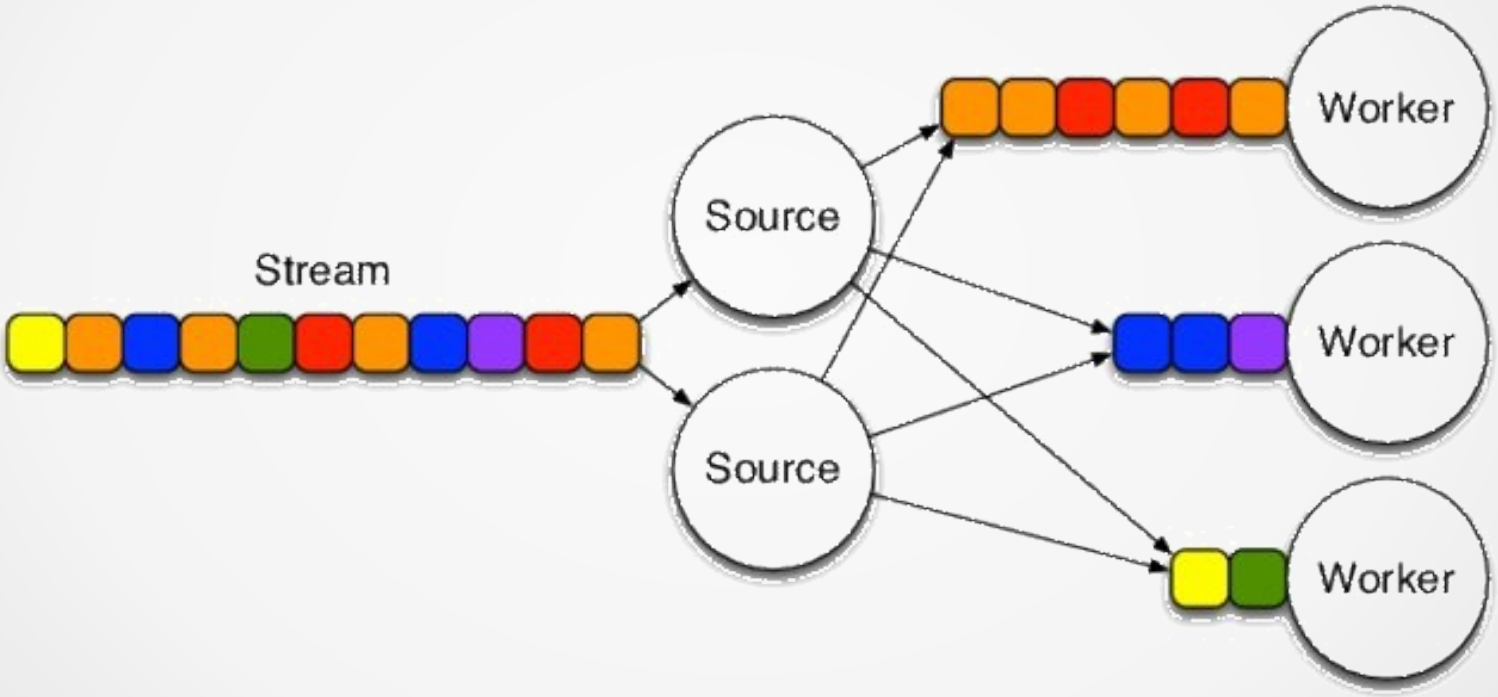


Example of Parallel Communications Overhead and Complexity: actual callgraph from the simple parallel "hello world" program shown. Most of the routines are from communications libraries.

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

Blocker 2: Parallel overhead

Load imbalance



2.

Hardware for parallel computing

At the core level

- Instruction-level parallelism (ILP)
  - Instruction pipelining
  - Superscalar execution
  - Out-of-order execution
  - Speculative execution
- Single Instruction Multiple Data (SIMD)

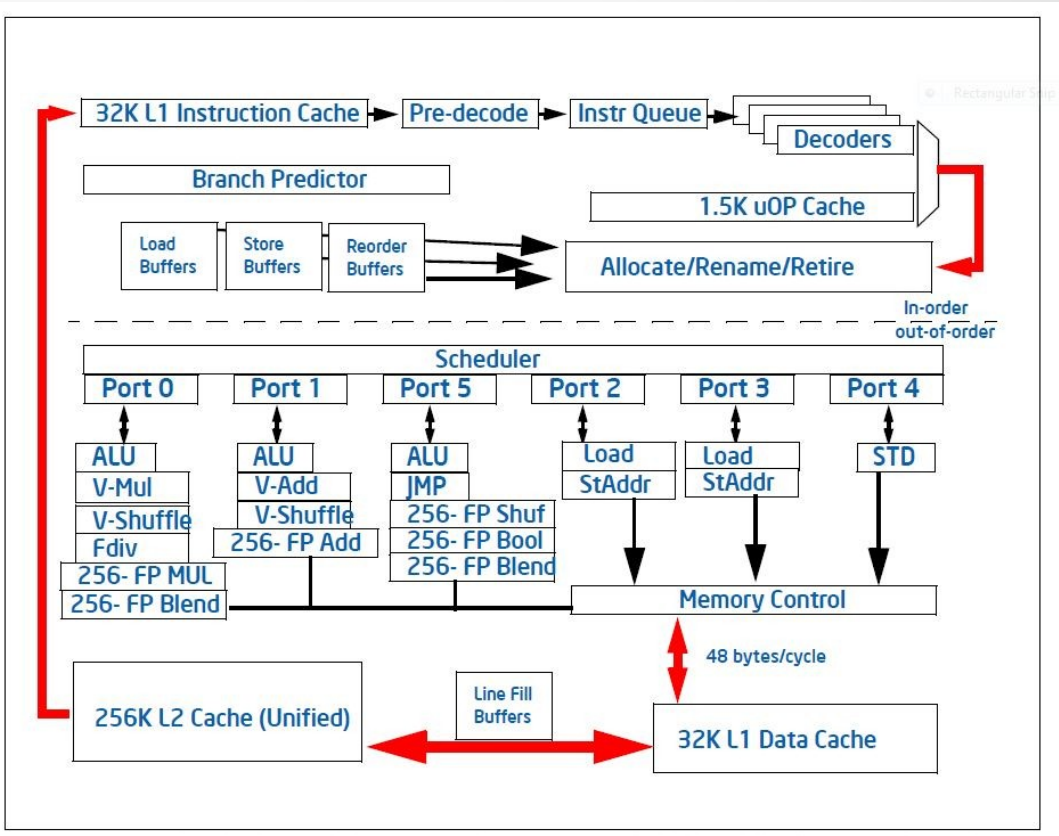
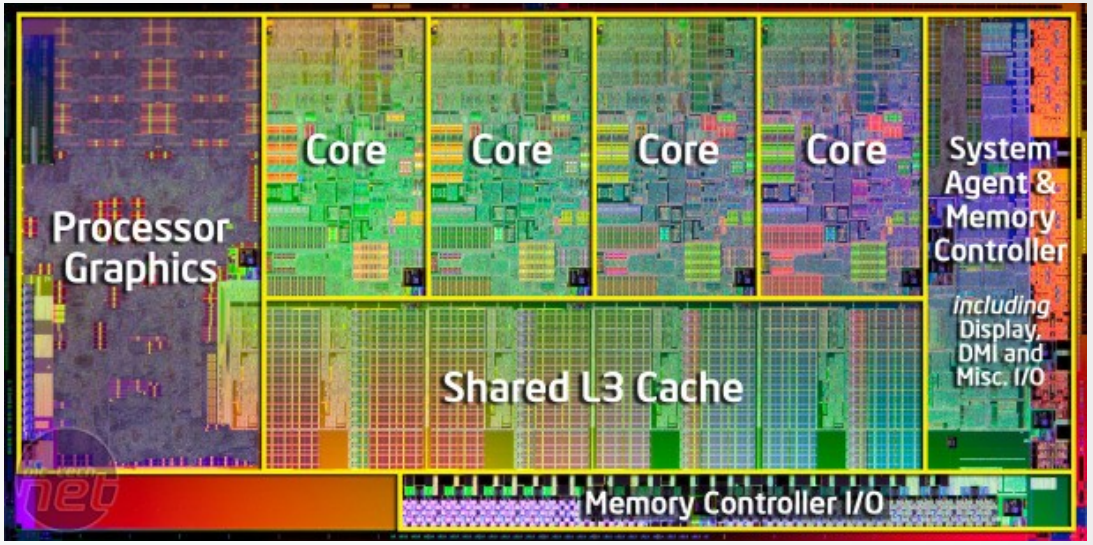


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

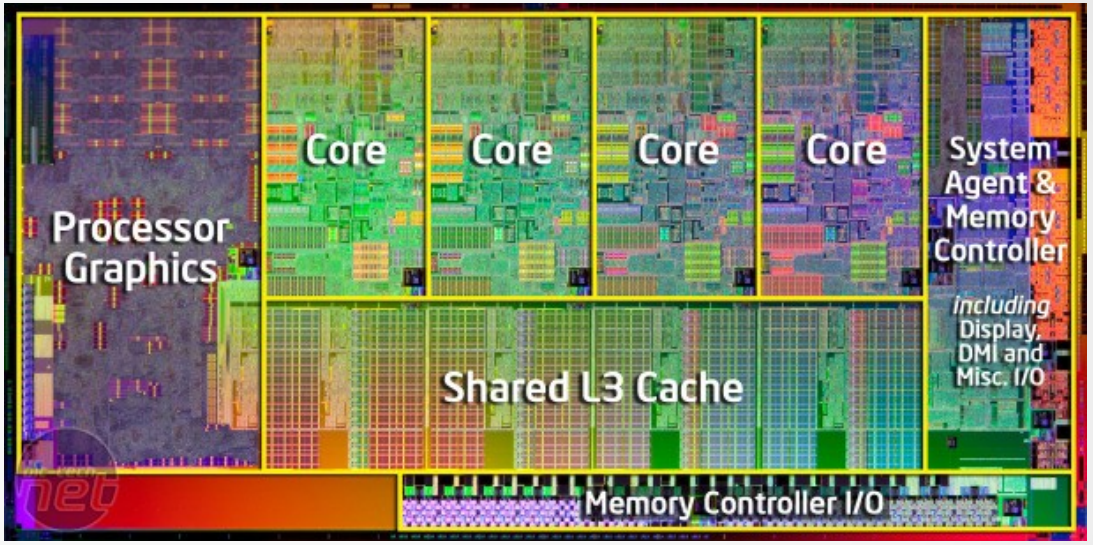
At the CPU (socket) level

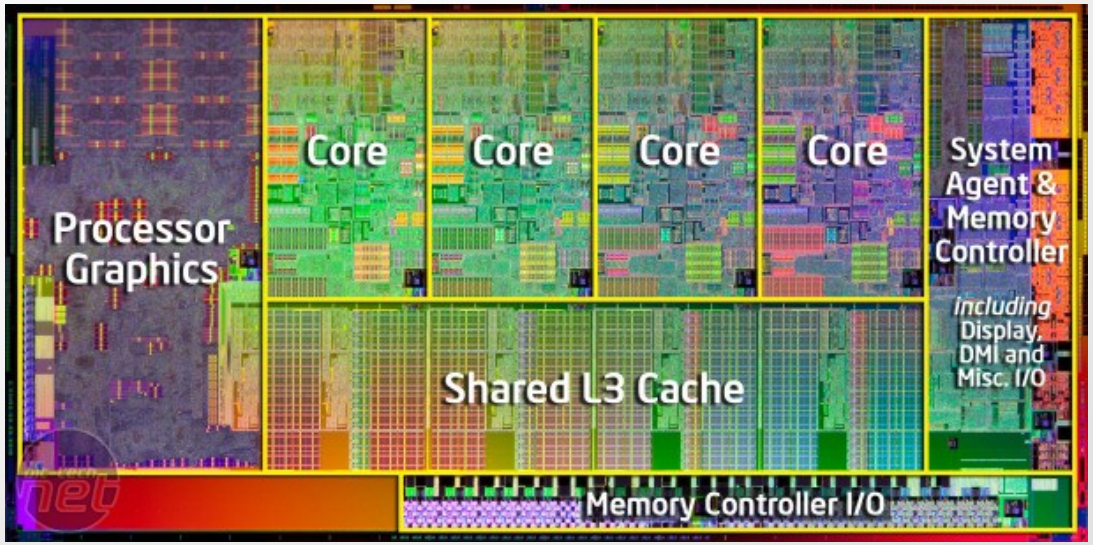
- Multicore parallelism



At the computer level

- Multi-socket parallelism
  - SMP
  - NUMA
- Accelerators





At the data center level

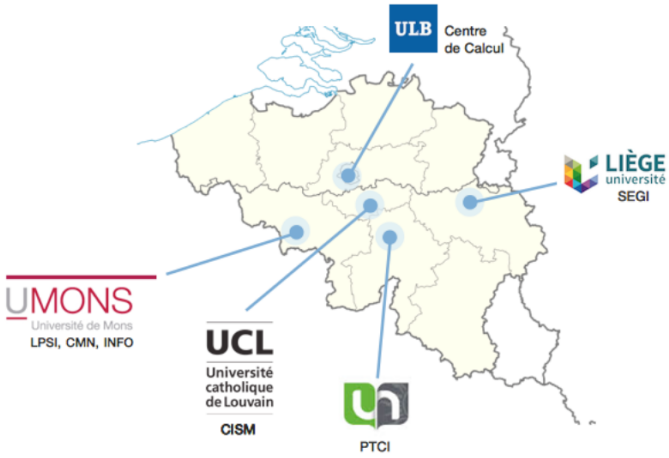
Cluster computing

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.



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

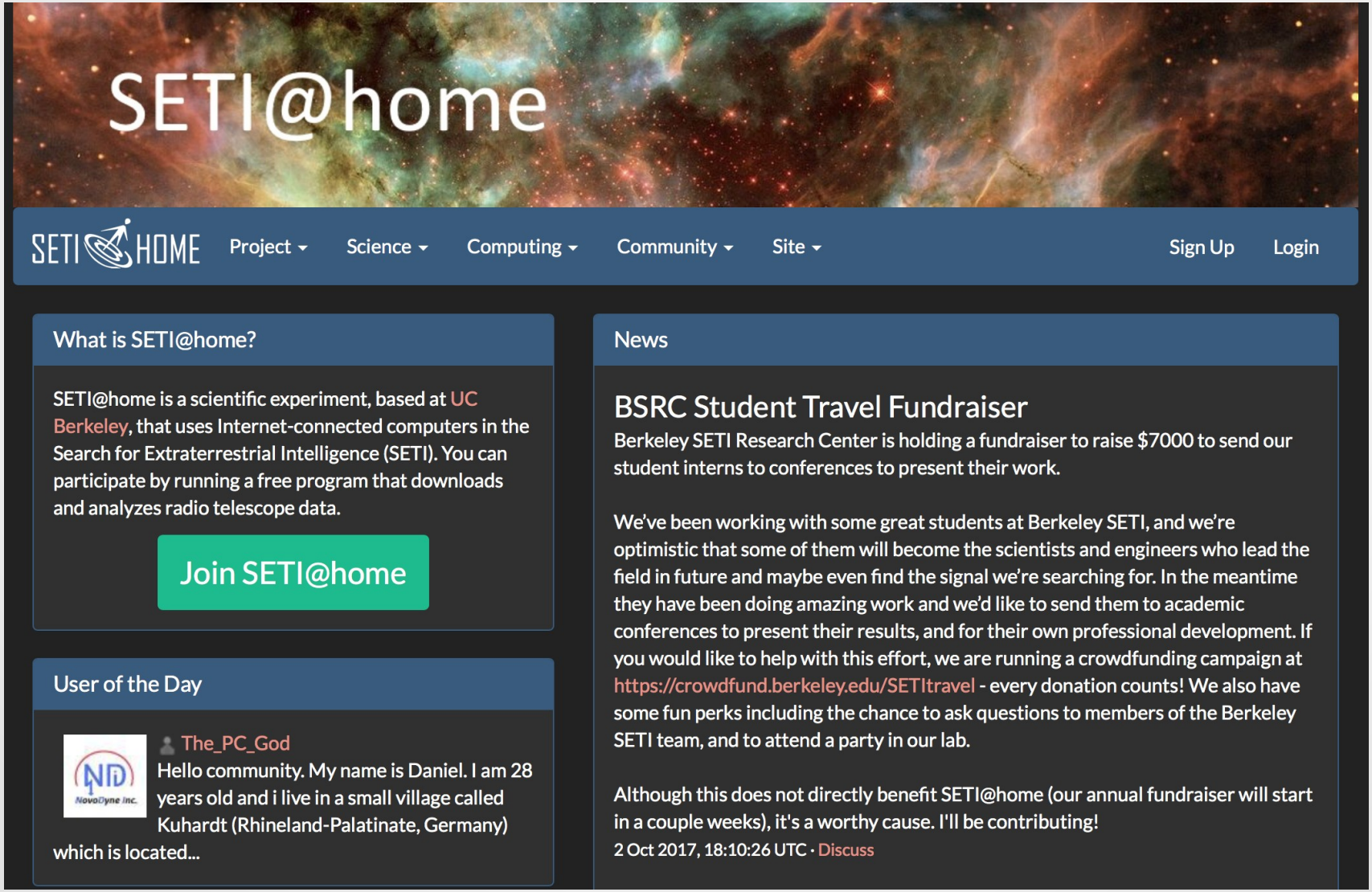
We are happy to announce the hire of a new CECI logisticien: Ariel Lozano. Welcome Ariel!

At the data center level

Cloud computing

At the world level

Distributed computing



At the world level

Grid computing

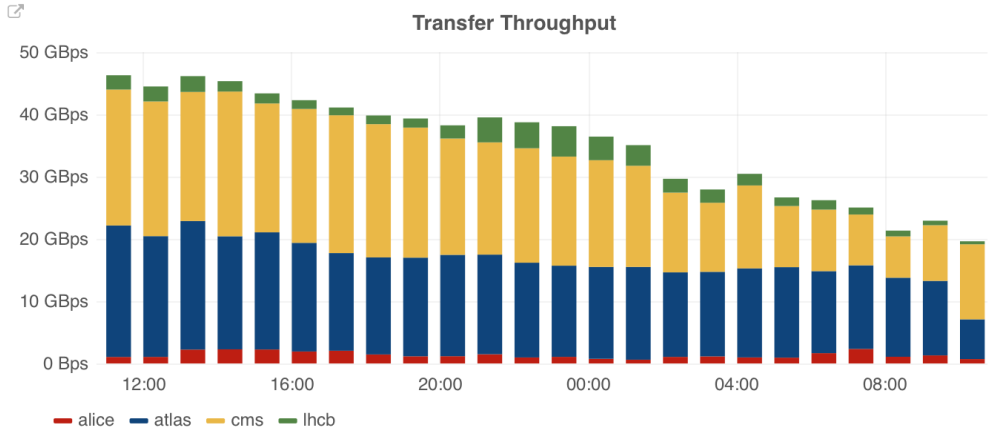
WLCG
Worldwide LHC Computing Grid

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

Programming models

Parallel programming paradigms

How is work organized?

- **Task-farming:** no communication among workers
  - Master distribute work to workers (master/slave); or
  - Workers pick up tasks from pool (work stealing).
- **SPMD** (Single program multiple data)

  A single program that contains both the logic for distributing work (master) and the computing part (workers) of which many instances are started and linked together at the same time
- **MPMD** (Multiple programs multiple data)

Parallel programming paradigms

How is work organized?

- **Pipelining** (A->B->C, one process per task concurrently)
- **Divide and Conquer** (processes spawned at need and report their result to the parent)
- **Speculative parallelism** (processes spawned and result possibly discarded)

Programming models

What programming libraries/syntax constructs, etc. exist?

- Single computer:
  - **CPUs**: PThreads, OpenMP, TBB, OpenCL
  - **Accelerators**: CUDA, OpenCL, OpenAcc
- Multi-computer:
  - **Distributed memory**:
    - Shared storage: MPI (clusters)
    - Distributed storage: MapReduce (clouds)
    - No storage: BOINC (distributed computing)
  - **Shared memory**: CoArray, UPC

User tools

that GNU/Linux offers

Parallel processes in Bash

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 $
```

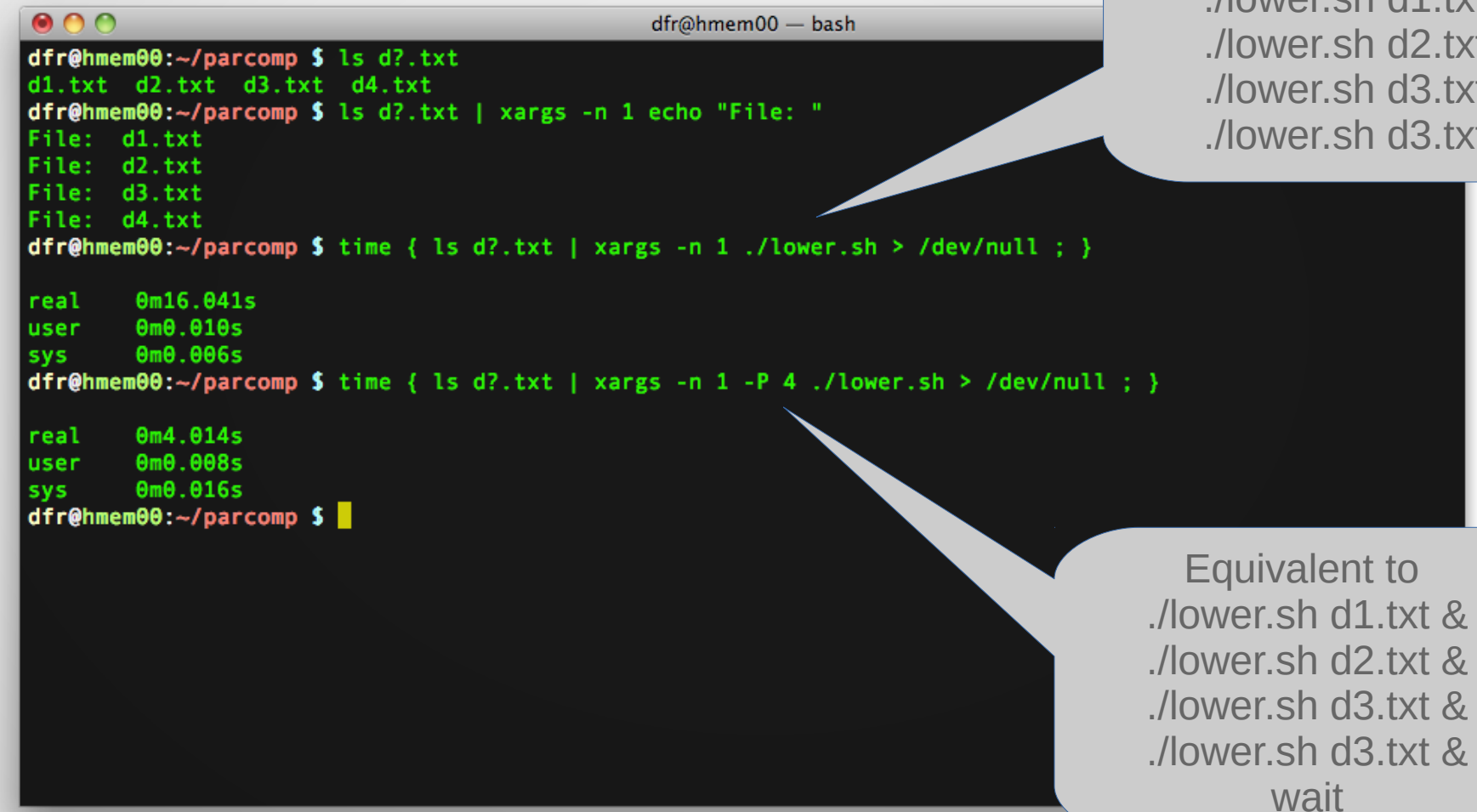
Parallel processes in Bash

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

One program and many files



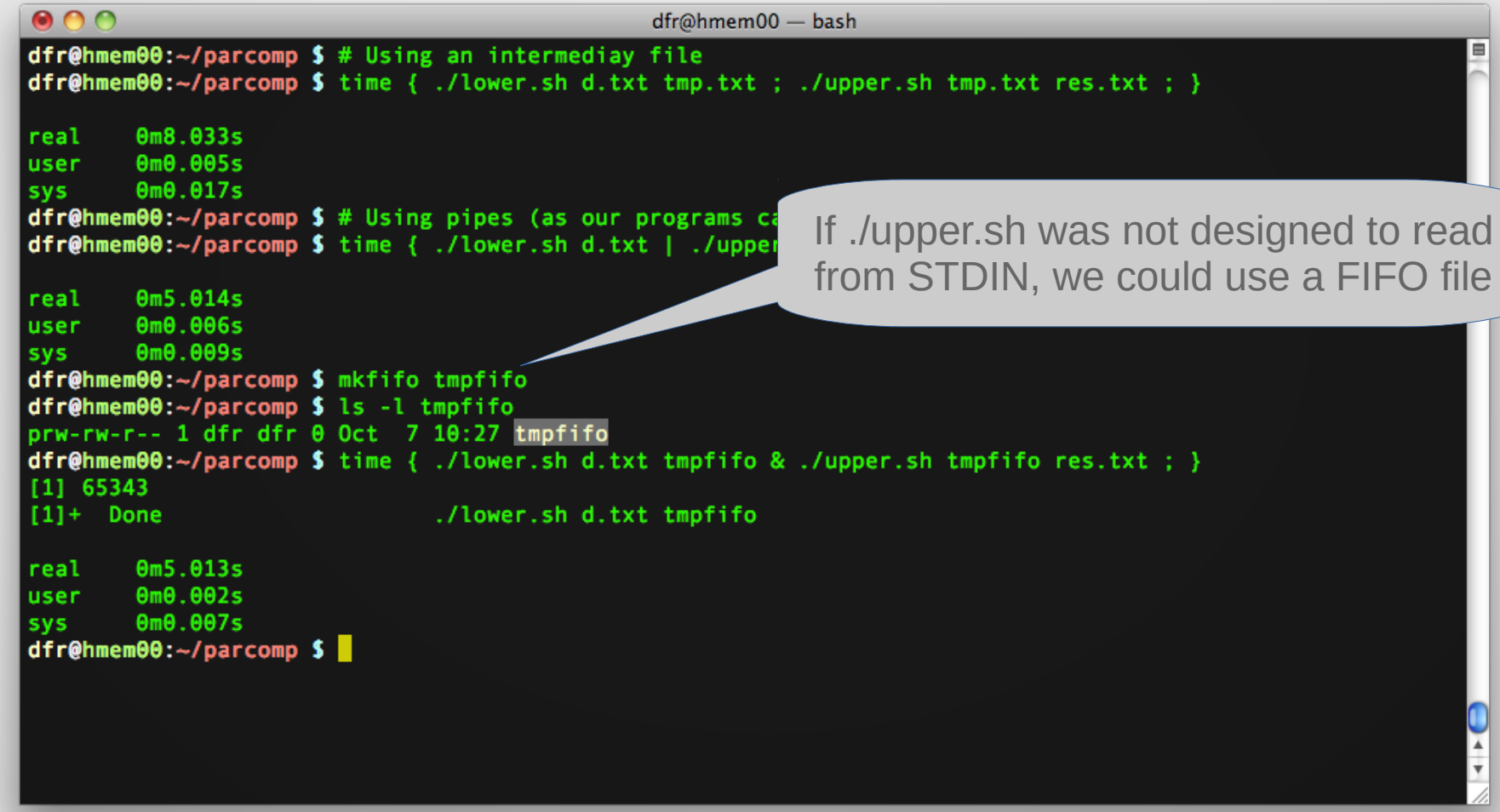
Several programs and one file

./upper.sh waits for ./lower.sh to finish

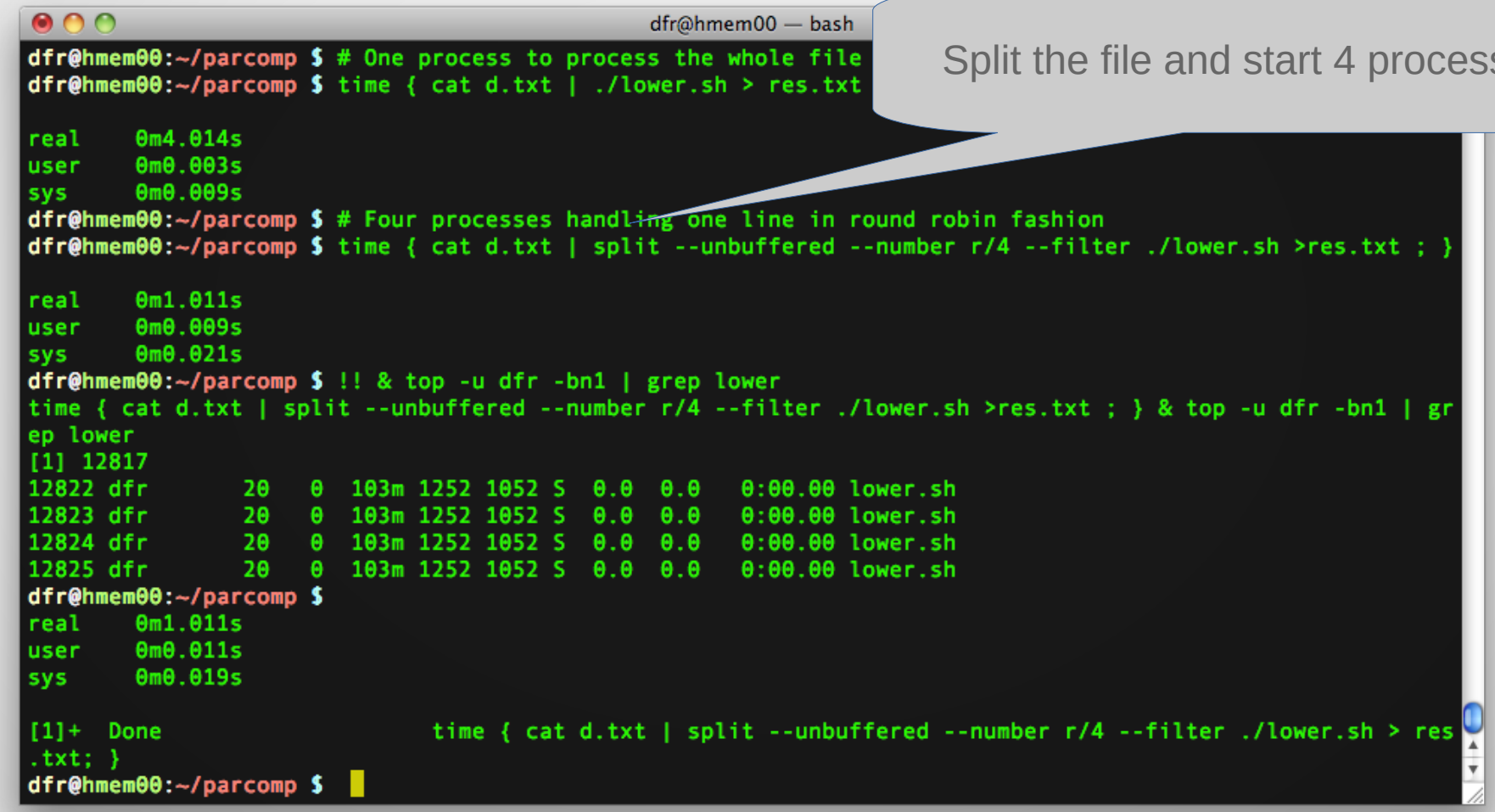
```
dfr@hmem00 — bash
dfr@hmem00:~/parcomp $ # Using an intermediay file
dfr@hmem00:~/parcomp $ time { ./lower.sh d.txt tmp.txt ; ./upper.sh tmp.txt res.txt ; }

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 $
```

./upper.sh starts as soon as ./lower.sh writes the first output

Several programs and one file



One program and one large file



Split the file and start 4 processes

Need recent version of Coreutils/8.22-goolf-1.4.10

Several programs and many files

dfr@hmem00 — bash

```
# 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
```


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

Several programs and many files

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: pipes, fifo
  - many files to process
    - with one program xargs
    - with many programs make

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.

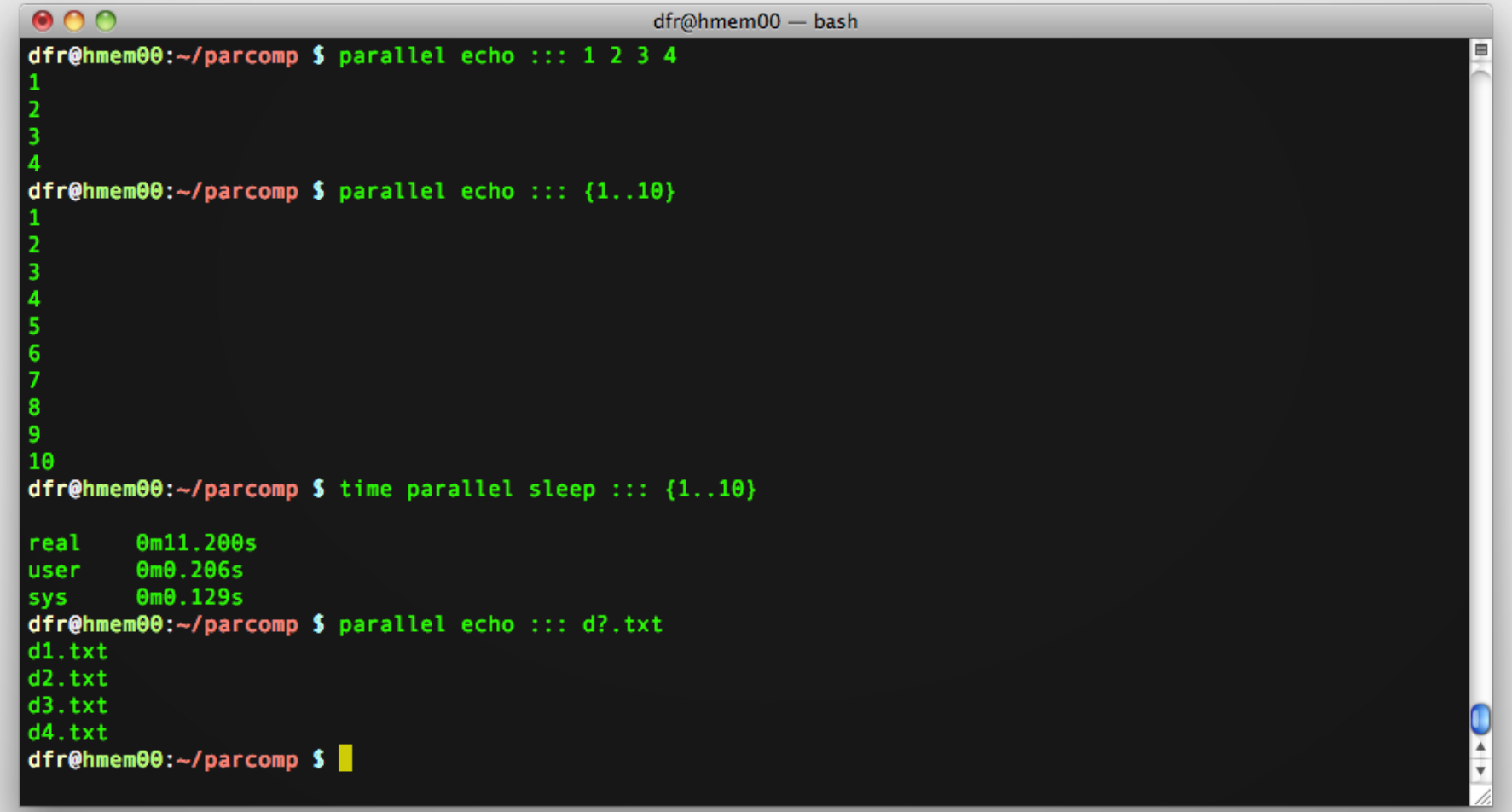
For people who live life in the parallel lane.

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

GNU Parallel

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



GNU Parallel

- Syntax: {} as argument placeholder. Can be modified

```
dfr@hmem00:~/parcomp $ parallel echo {} ::: d?.txt
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 $
```

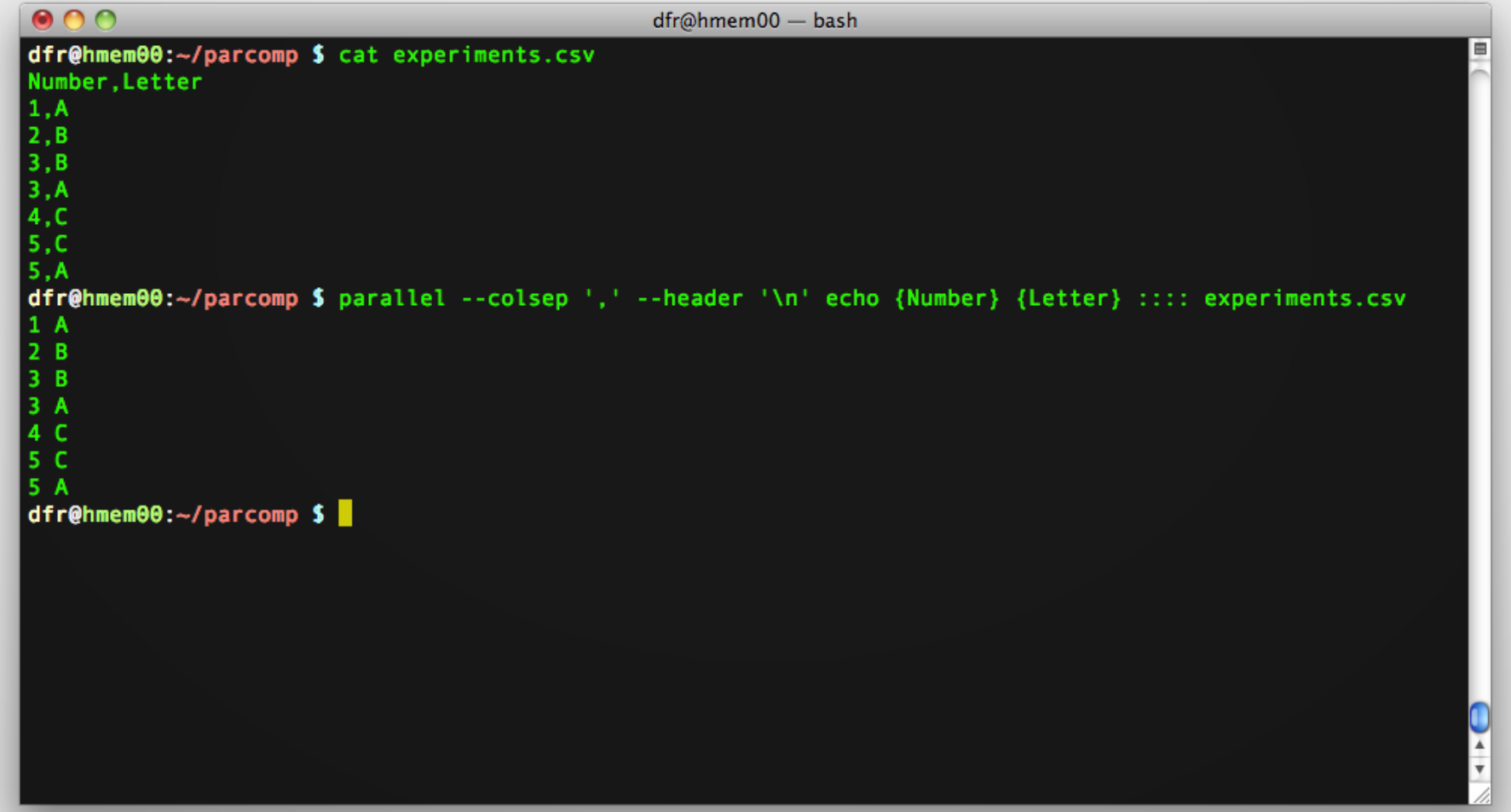
GNU Parallel

- 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
```

GNU Parallel

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



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 $
```

Other interesting options

`-S`	Use remote servers through SSH
`-j n`	Run n jobs in parallel
`-k`	Keep same order
`--delay n`	Ensure there are n seconds between each start
`--timeout n`	Kill task after n seconds if still running

Author asks to be cited: O. Tange (2011): *GNU Parallel - The Command-Line Power Tool*, The USENIX Magazine, February 2011:42-47.

Exercise

Reproduce the examples with ./lower and ./upper.sh using GNU Parallel

Solutions

- One program and many files

```
$ time parallel -k ./lower.sh {} > res.txt  ::: d?.txt
```

- One program and one large file

```
$ time cat d.txt | parallel -k -N1 --pipe ./lower.sh {} > res.txt
```

- Several programs and several files

```
$ time { parallel ./lower.sh {} {.}.tmp ::: d?.txt ; \
> parallel  ./upper.sh {} {.}.res ::: d?.tmp ; }
```