High performance computing and Madagascar

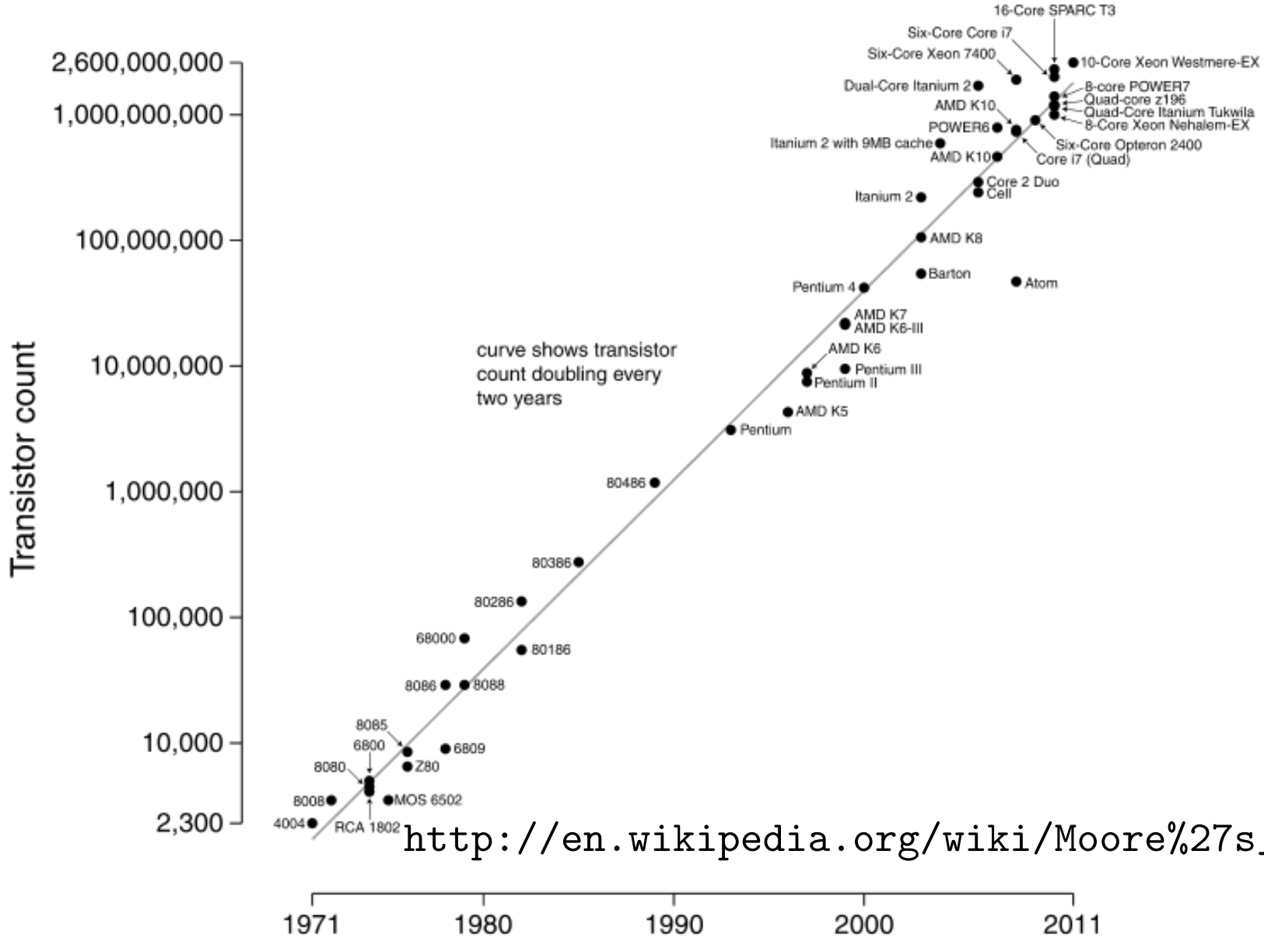
Esteban Díaz

Center for Wave Phenomena
Colorado School of Mines

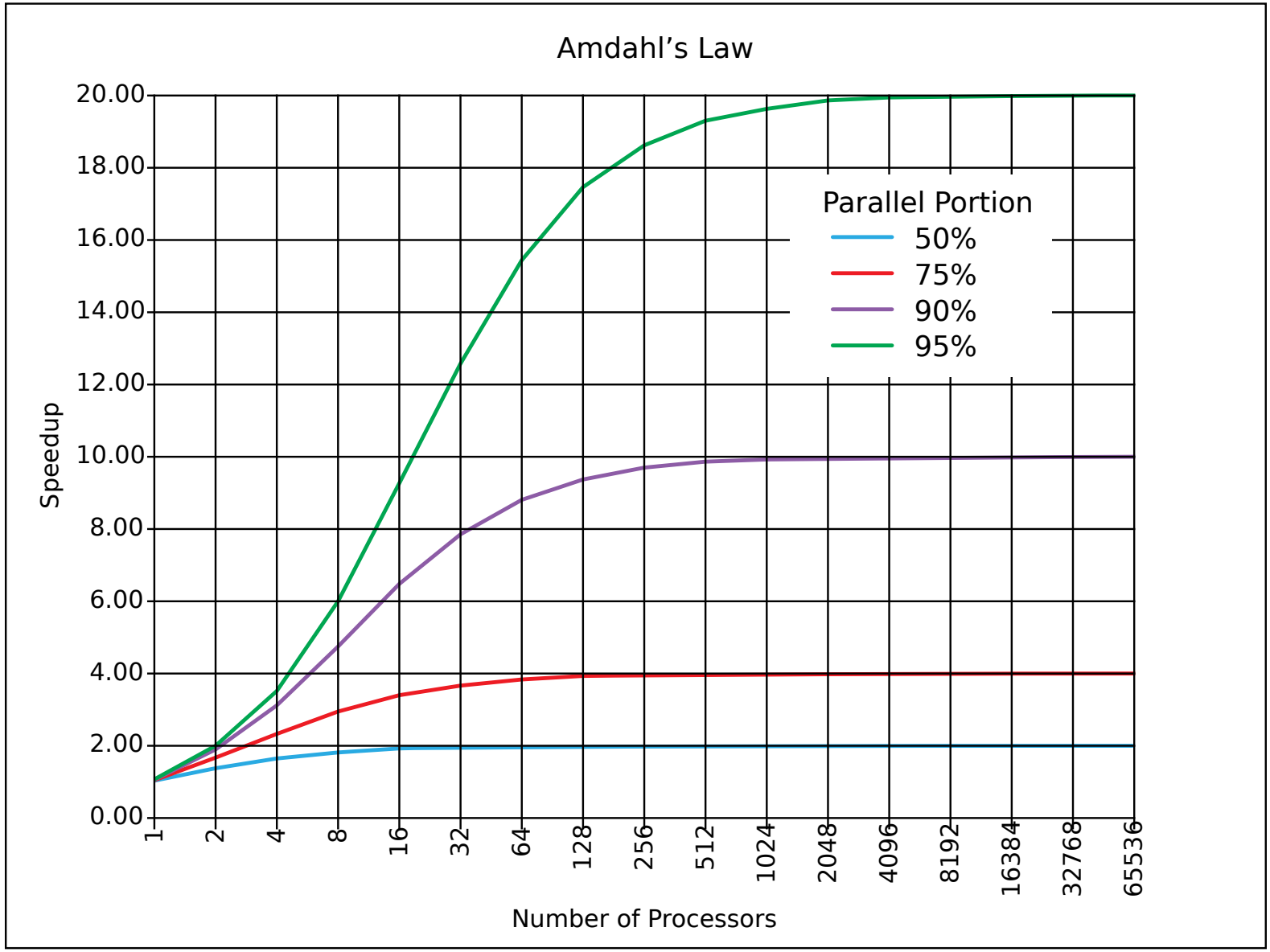
April 15, 2014

Why should we bother to go parallel?

Microprocessor Transistor Counts 1971-2011 & Moore's Law



http://en.wikipedia.org/wiki/Moore%27s_law



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

outline

concepts of parallel computing

approaches: OpenMP and MPI

do you need to code parallel? Let SCons do it for you!

demo example

outline

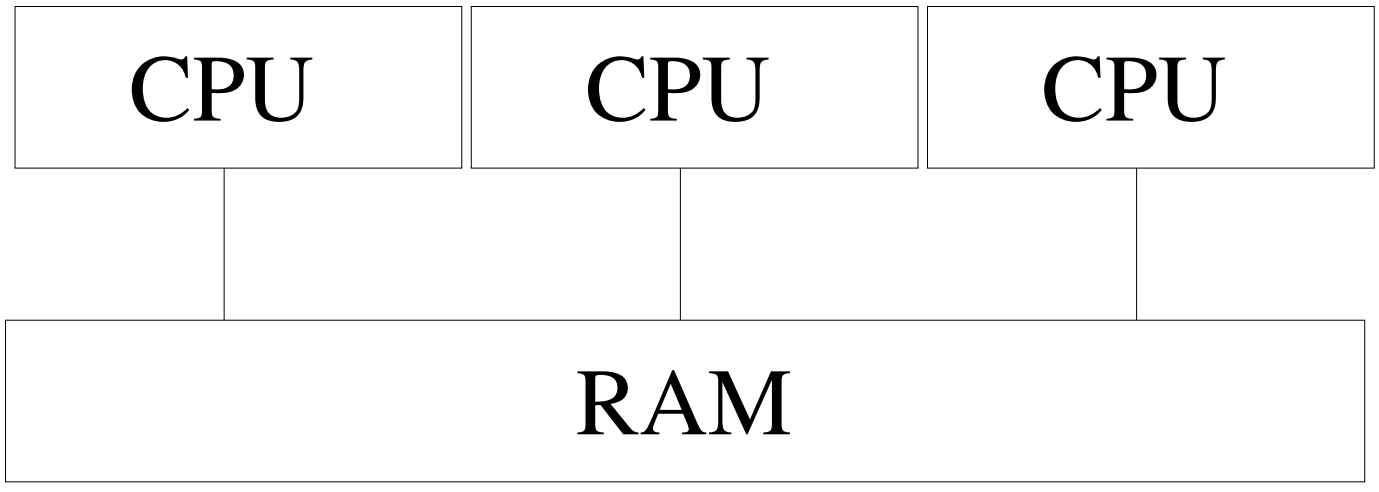
concepts of parallel computing

approaches: OpenMP and MPI

do you need to code parallel? Let SCons do it for you!

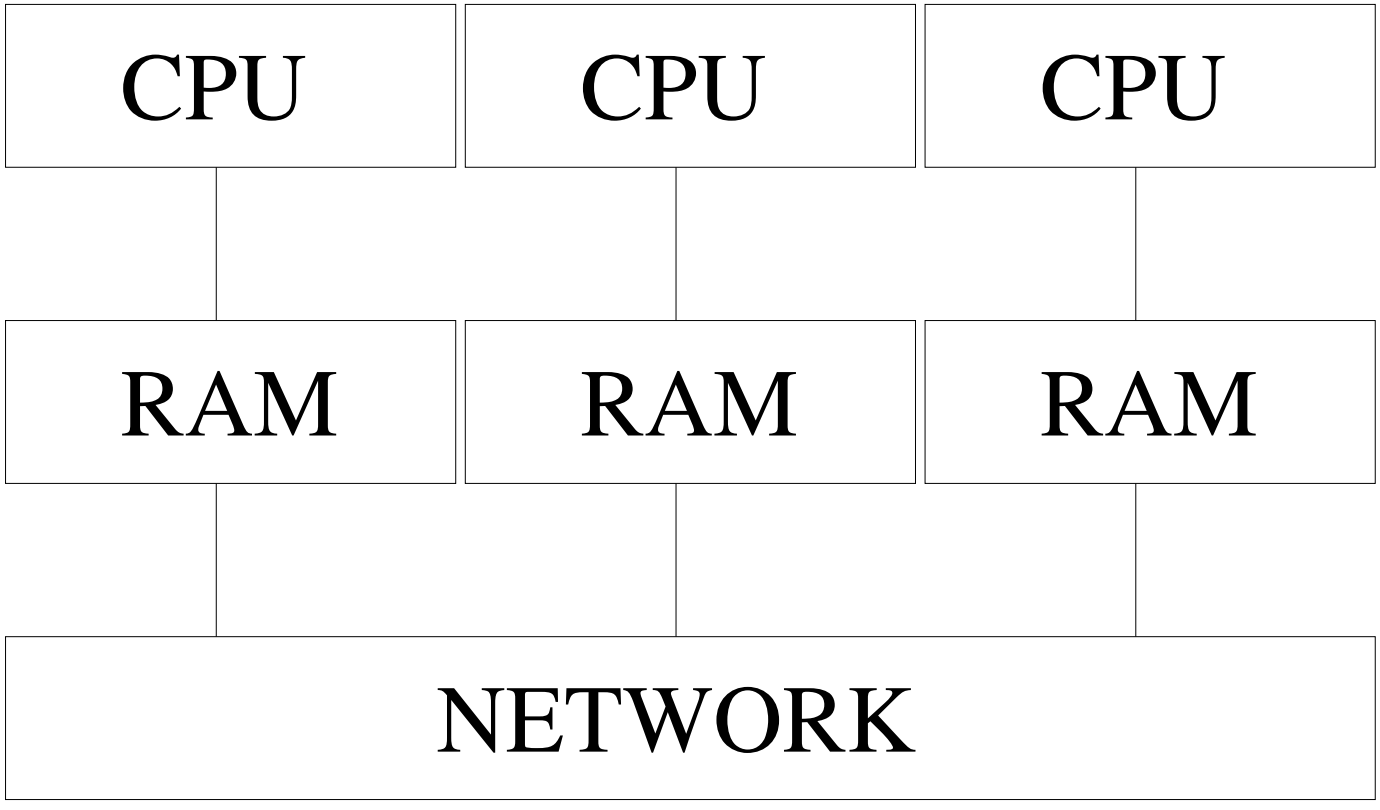
demo example

shared memory system

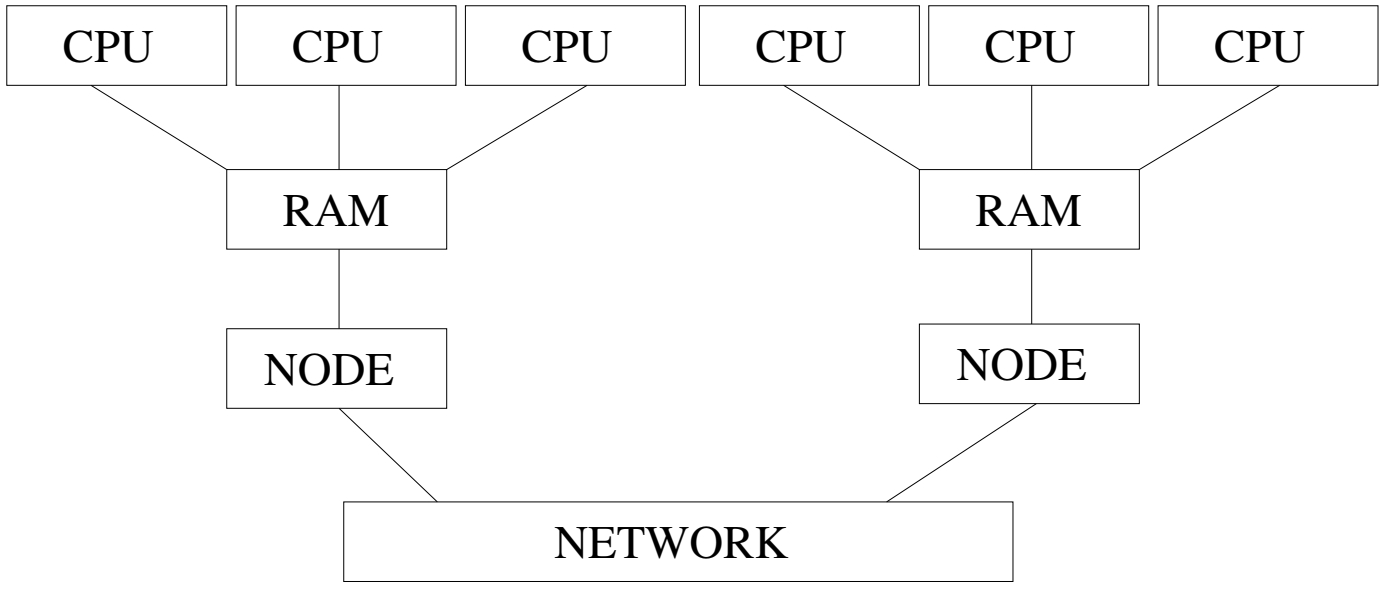


modified from Vladimir Bashkardin, 2010

distributed memory system



modified from Vladimir Bashkardin, 2010



hybrid memory system (i.e. clusters)

outline

concepts of parallel computing

approaches: OpenMP and MPI

do you need to code parallel? Let SCons do it for you!

demo example

OpenMP:

addresses shared memory parallelization

supported by most of compilers

OpenMP:

addresses shared memory parallelization

supported by most of compilers

resources:

`http://openmp.org/wp/`

`http://www.cOMPunity.org/`

MPI (Message Passing Interface):

addresses distributed memory parallelization

You need MPI libraries

MPI (Message Passing Interface):

addresses distributed memory parallelization

You need MPI libraries

resources:

`http://www.open-mpi.org/software/`

outline

concepts of parallel computing

approaches: OpenMP and MPI

do you need to code parallel? Let SCons do it for you!

demo example

parallelization with SCons

instead of scons use **pscons**

it will automatically run in parallel independent Flows

parallelization with SCons

instead of scons use **pscons**

it will automatically run in parallel independent Flows

---

useful environmental variables to have:

```
$ export RSF_THREADS='8'
$ export RSF_CLUSTER='localhost 8'
$ export OMP_NUM_THREADS=$RSF_THREADS
```

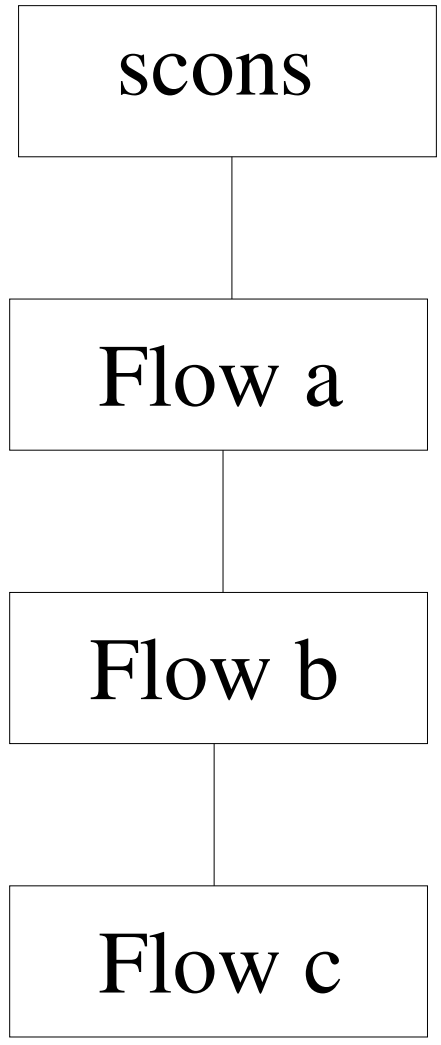
scons vs pscons

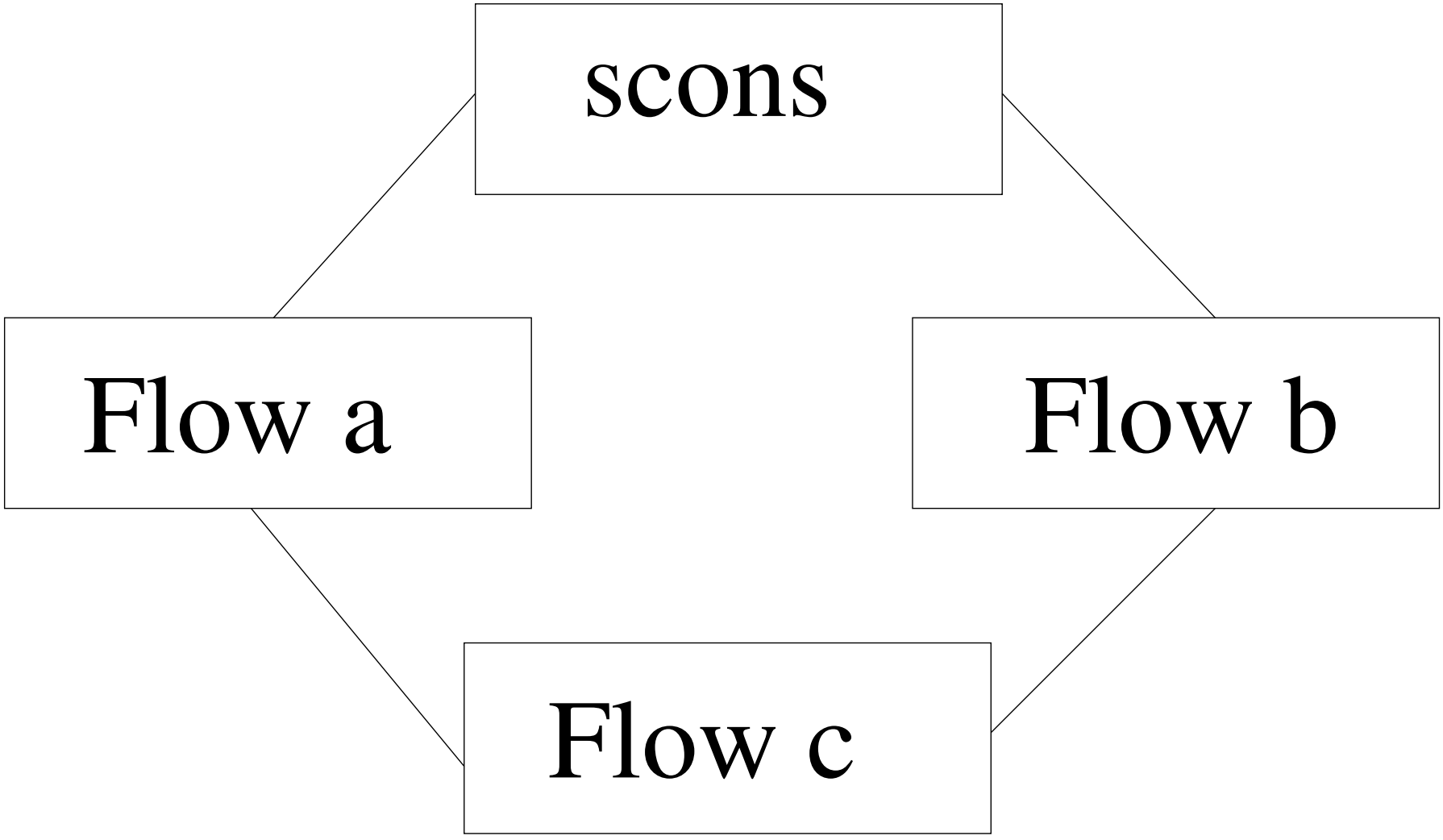
```
from rsf.proj import *


Flow('a','input','flow a')

Flow('b','input','flow b')

Flow('c','a b','flow c')
```


scons
Flow a
Flow b
Flow c


scons
Flow a
Flow b
Flow c

we can do better than the sequential flow:

```
Flow('Targets','Sources','flow')
```

... by setting up some parallel variables in Flow:

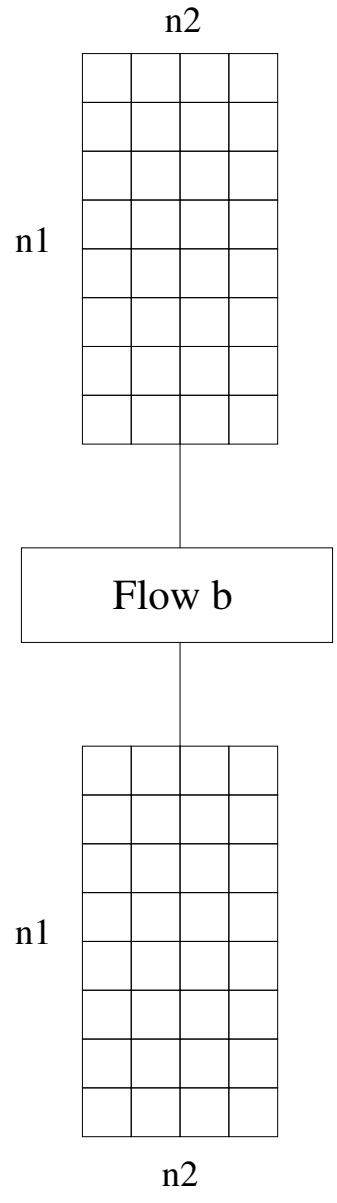
```
Flow('Targets','Sources','flow',
       split=[n,m],reduce='cat')
```

n = axis to split.

m = length of dimension **n**

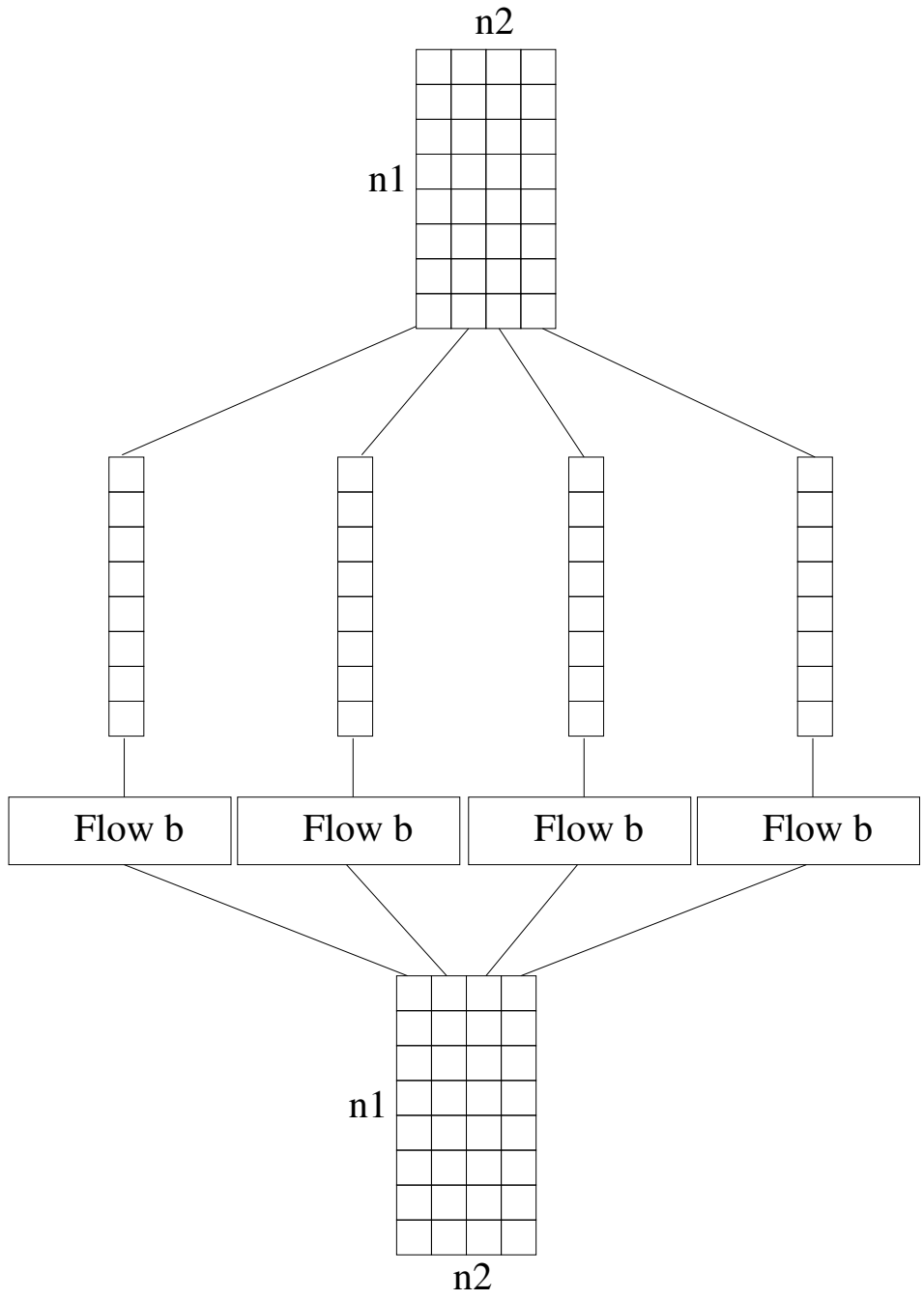
Let's consider a sequential example:

```
Flow('a',None,'spike n1=8 n2=4 ')
Flow('b','a','noise rep=y')
```


n2
n1
Flow b
n1
n2

now let's run it in parallel:

```
Flow('a',None,'spike n1=8 n2=4 ')
Flow('b','a','noise rep=y',split=[2,4],reduce='cat')
```


n2
n1
Flow b
Flow b
Flow b
Flow b
n1
n2

The variable ‘split‘ in Flow can take several options:

```
Flow('Targets','Sources','flow',
     split=[n,m],reduce='cat')
```

The variable ‘split‘ in Flow can take several options:

```
Flow('Targets','Sources','flow',
     split=[n,m],reduce='cat')
```

```
Flow('Targets','Sources','flow',
     split=[n,'omp'],reduce='cat')
```

The variable ‘split‘ in Flow can take several options:

```
Flow('Targets','Sources','flow',
     split=[n,m],reduce='cat')
```

```
Flow('Targets','Sources','flow',
     split=[n,'omp'],reduce='cat')
```

```
Flow('Targets','Sources','flow',
     split=[n,'mpi'],np=m,reduce='cat')
```

other reduce commands: sfrcat, sfadd, sfinterleave ...

outline

concepts of parallel computing

approaches: OpenMP and MPI

do you need to code parallel? Let SCons do it for you!

demo example

```
from rsf.proj import *

# Input data to use in example:
# produces random numbers with uniform
# distribution [-1000,1000]

Flow('random',None,
     '''spike n1=1024 n2=1024 n3=128|
        noise rep=y range=2000  type=n
       |math output="input-1000"''')

Flow('clip_seq','random','clip clip=500')
```

Let's see the timing:

```
time pscons clip_seq.rsf
```

```
# 1: In this case scons will split the input,
#    process separately, and cat all the
#    individual outputs into one target.
Flow('clip_scons','random',' clip clip=500',
      split=[3,128],reduce='cat')
```

```
time pscons clip_scons.rsf
```

```
# 2: Now, I am specifying the type of parallelization
#    using OpenMP, it is parallelizing over the
#    slower axis (the third one)
Flow('clip_omp_scons','random',' clip clip=500',
    split=[3,'omp'],reduce='cat')
```

```
time pscons clip_omp_scons.rsf
```

```
# 3: In this case I use explicit parallelization
#    with MPI [n,'mpi'] in this case m is the
#    number of CPU's to use, n is the axis to split

Flow('clip_mpi_scons','random',' clip clip=500',
    split=[3,'mpi'],np=8,reduce='cat')
```

```
time pscons clip_mpi_scons.rsf
```

```
# 4: OMP parallelization in the code. It uses the
#    number of maximum number of cores available if
#    $OMP_NUM_THREADS doesn't exist.
Program('clip_omp','Mclip.c')
Flow('clip_omp_incore','random clip_omp.exe',
     './${SOURCES[1]} clip=500')
```

```
time pscons clip_omp_incore.rsf
```

```
# 5: Hybrid case, OMP parallelization in the code, and
#    extra parallelization with MPI. In a single
#    computer doesn't make too much sense but with
#    a cluster it does.
Flow('clip_hybrid','random clip_omp.exe',
    './${SOURCES[1]} clip=500',split=[3,'mpi'],np=4,
                                    reduce='cat')
```

```
time pscons clip_hybrid.rsf
```

Issues that affect scalability:

- I/O (read and write data to disk)

Issues that affect scalability:

- I/O (read and write data to disk)
- network

Issues that affect scalability:

- I/O (read and write data to disk)
- network
- combination of both

conclusions

- parallelize as much as you can (Amdahl's law)
- use pscons instead of scons
- avoid I/O and network as much as possible
- do code parallelization if needed

resources

http://ahay.org/wiki/Parallel_Computing

Vladimir Bashkardin's, Houston 2010 school:

http://ahay.org/wikilocal/docs/Houston2010_HPC.pdf

Dave Hale's, Houston 2011 workshop:

http://www.beg.utexas.edu/pttc/2011/2011_pres_01_03_hale.pdf

Thank you!

check at the Yang Liu's 2010 school for a real data example with parallelization:
$RSFSRC/book/rsf/usp/data/