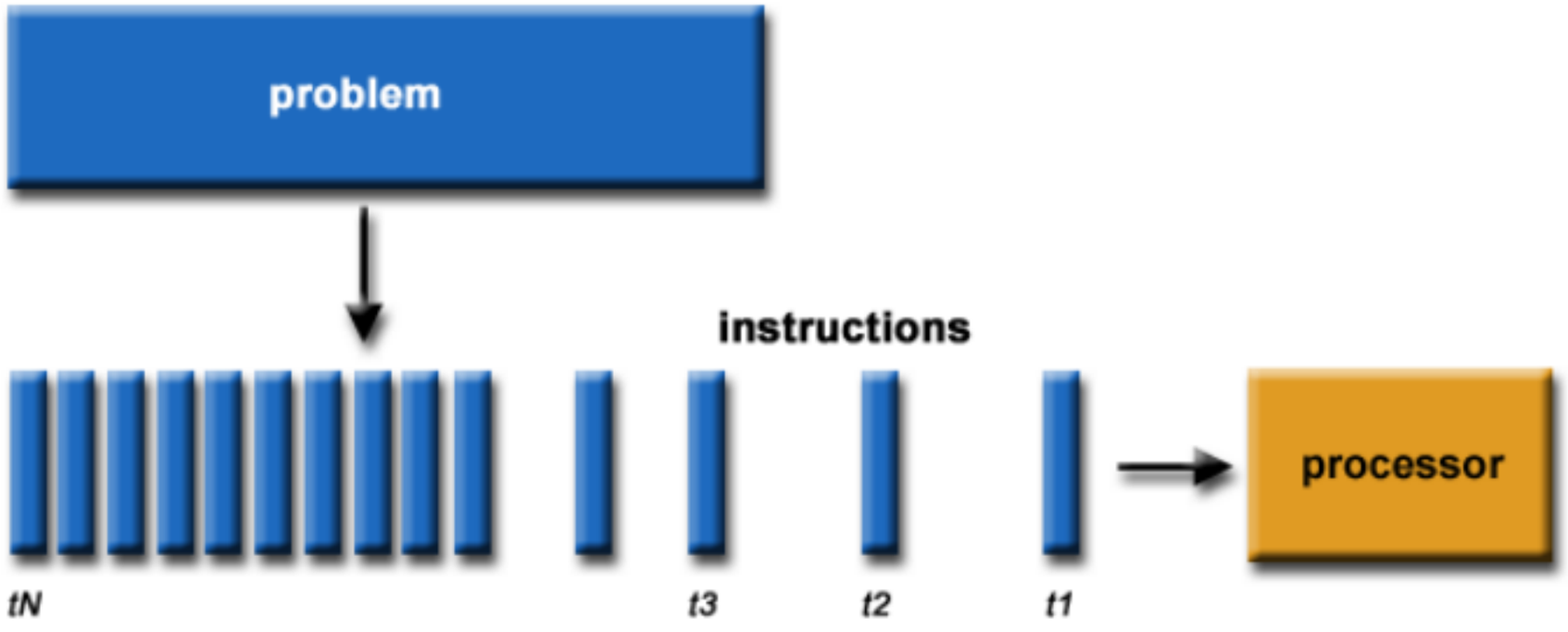
Εφαρμογές Η/Υ

9ο Μάθημα

- Εισαγωγή στους παράλληλους υπολογισμούς

Ν. Λαγαρός, Θ. Στάμος, Χ. Φραγκουδάκης

Σειριακοί Υπολογισμοί

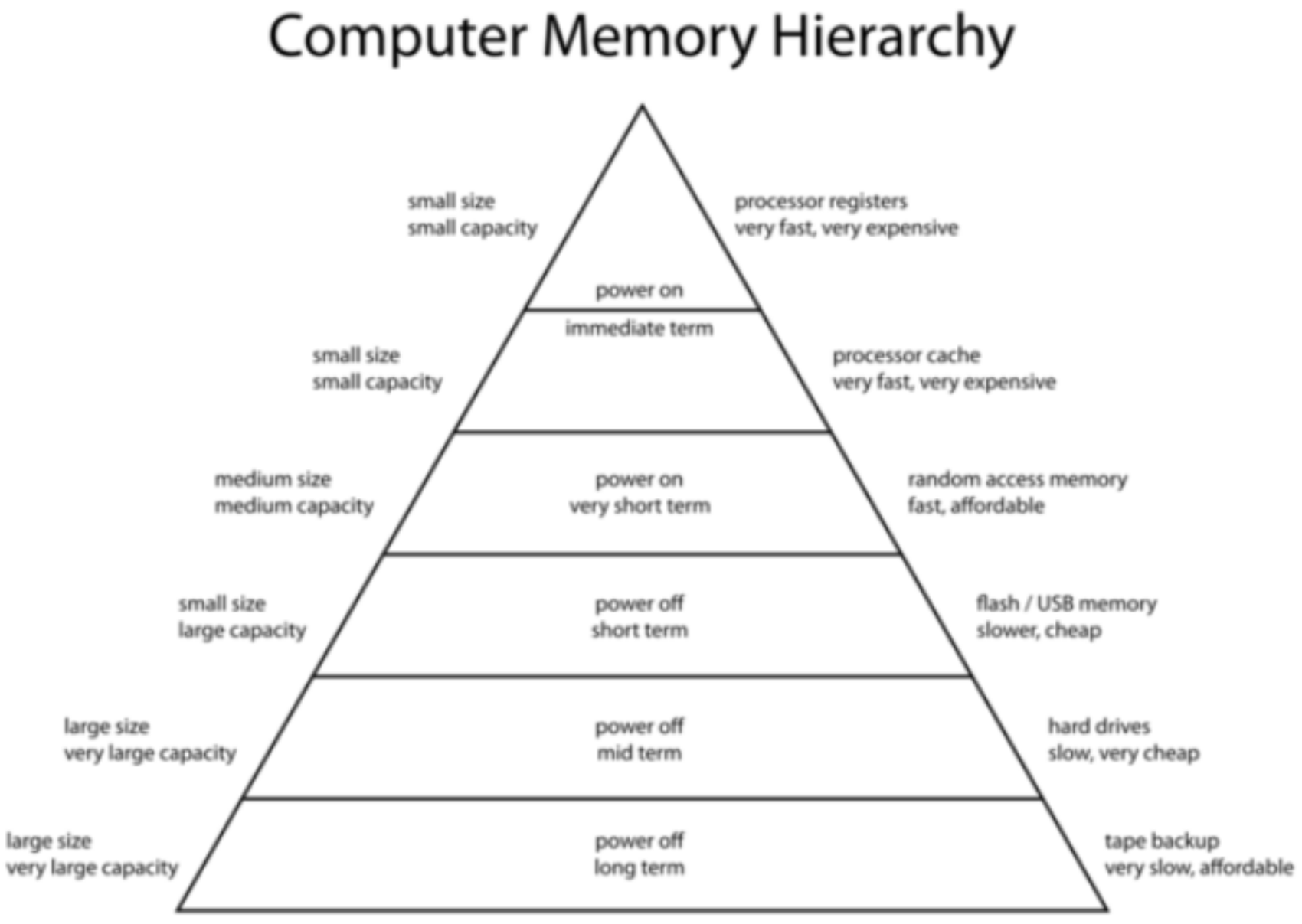


- Ο (μοναδικός) υπολογιστής εκτελεί σειριακά (μια-μια) τις εντολές
- Τόσο τα δεδομένα όσο και οι εντολές φορτώνονται στη μνήμη

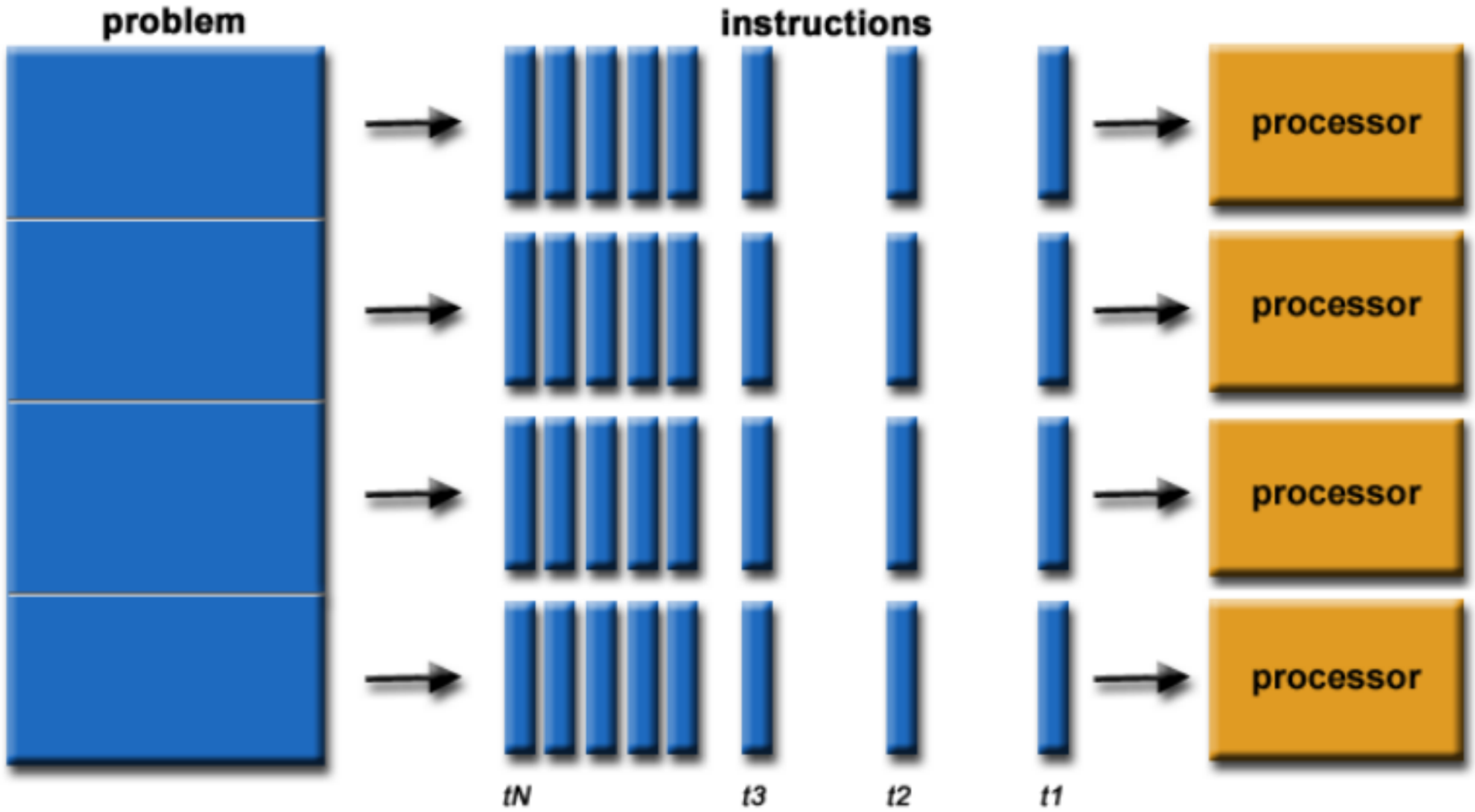
Βελτίωση της Υπολογιστικής Ισχύος

- Αύξηση της ταχύτητας του υπολογιστή
- Βελτίωση του χρόνου πρόσβασης στη μνήμη

Ιεραρχική Αρχιτεκτονική Μνήμης

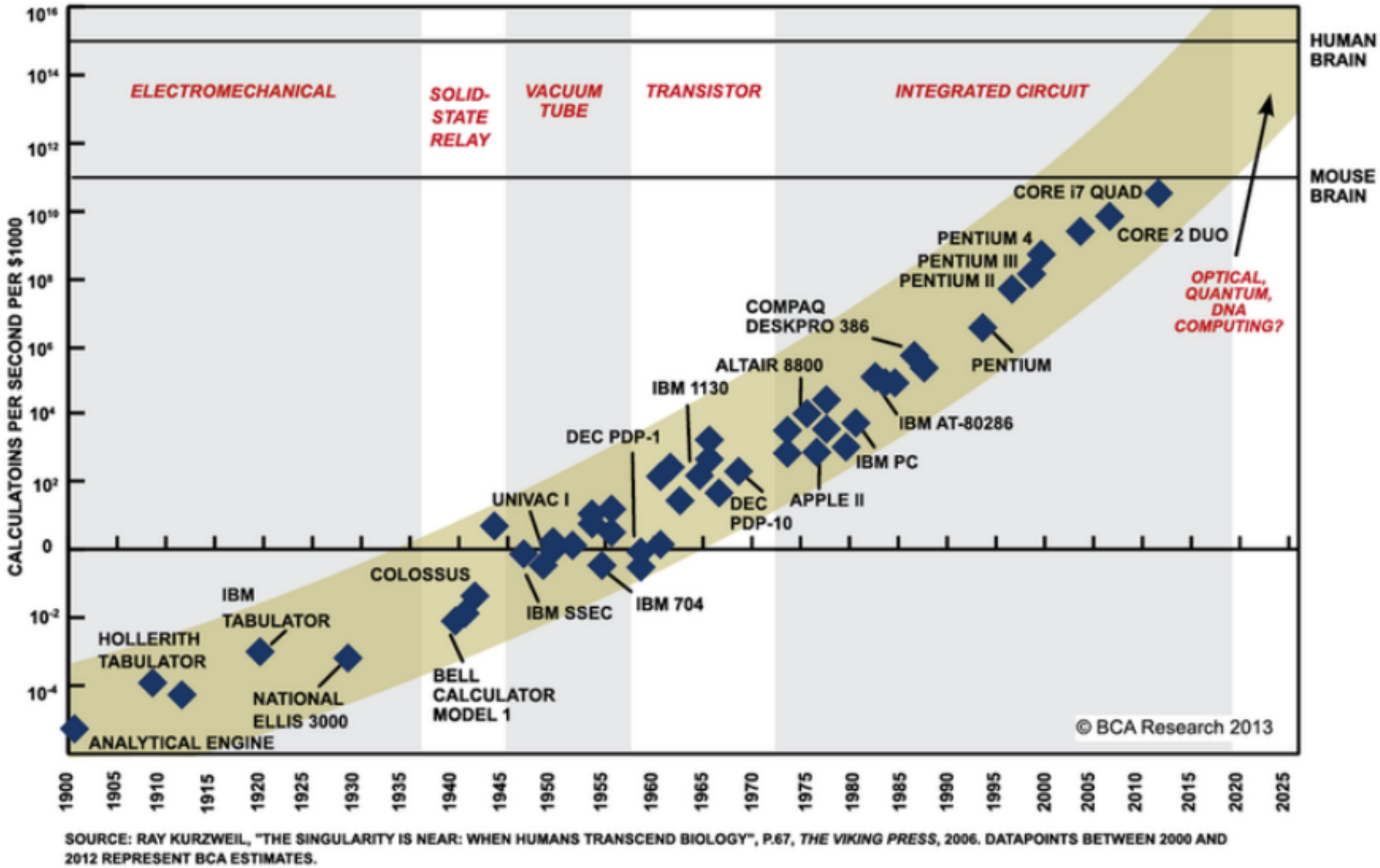


Παράλληλοι Υπολογισμοί



- Ένας μεγάλος υπολογισμός καταμερίζεται σε μικρότερους υπολογισμούς
- Οι καταμερισμένοι υπολογισμοί διανέμονται σε ανάλογους εργάτες (επεξεργαστές) που δουλεύουν ταυτόχρονα σειριακά
- Πρέπει να αναπτυχθούν κατάλληλοι παράλληλοι αλγόριθμοι

"Νόμος" του Moore: Ο αριθμός των τρανζίστορς σε ανά cm^2 σε ένα ολοκληρωμένο κύκλωμα διαπλασιάζεται κάθε 18 μήνες.



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

Οι Παράλληλοι Υπολογισμοί είναι αναγκαίοι

- Υπάρχει μεγάλη ανάγκη προσέγγισης δύσκολων και περίπλοκων προβλημάτων
- Περίπλοκα προβλήματα μεγάλης κλίμακας - Προσομοιώσεις (simulations):
  - Κβαντική χημεία, στατιστική φυσική
  - Σχεδιασμός υλικών
  - Βιολογικά συστήματα
    - Χαρτογράφηση DNA
    - Σχεδιασμός φαρμάκων
  - Πρόβλεψη κλίματος
  - Μελέτη γεωλογικών φαινομένων
  - Αστροφυσική και κοσμολογία
  - Αεροναυπηγική
  - ...
- Απαιτείται συνεχώς περισσότερη υπολογιστική ισχύς
- Ο νόμος του Moore θα αγγίξει τα φυσικά όρια: ο αριθμός των τρανζίστορς ανά cm^2 δεν μπορεί να αυξάνεται επ' άπειρο
- Λύση είναι η ταυτόχρονη εκτέλεση προγραμμάτων σε πολλούς επεξεργαστές

Βασικός στόχος των Παράλληλων Υπολογισμών

Ανάπτυξη υπολογιστικής ισχύος μέσω της χρήσης πολλών επεξεργαστών

- Μελέτη περίπλοκων προβλημάτων μεγάλης κλίμακας
- Επίλυση μεγαλύτερων προβλημάτων πολύ γρηγορότερα
- Επίλυση προβλημάτω που δεν μπορούν να λυθούν με ένα μόνο επεξεργαστή
- Εκμετάλευση των μεγάλων μεγεθών μνήμης που υπάρχουν

Προκλήσεις των Παράλληλων Υπολογισμών

- Συντονισμός, έλεγχος και παρακολούθηση των επεξεργαστών
- Ανάπτυξη κατάλληλων παράλληλων αλγορίθμων
- Βέλτιστη κατανομή στους επεξεργαστές
- Βέλτιστη (όσο το δυνατόν λιγότερη) επικοινωνία μεταξύ επεξεργαστών
- Εύκολη μεταφορά κωδίκων ανάμεσα σε διαφορετικά υπολογιστικά συστήματα

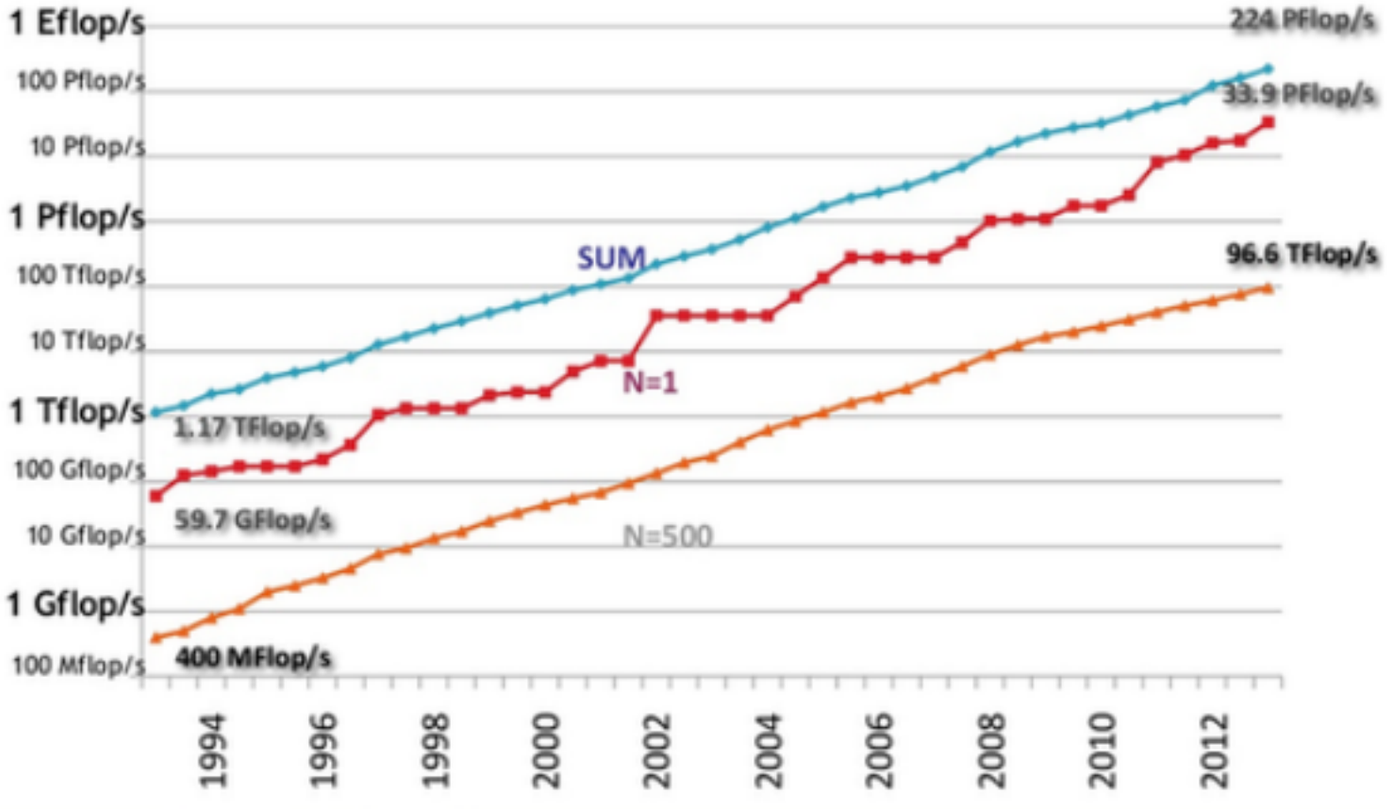
Μειονεκτήματα

- Δύσκολη κατασκευή παράλληλων υπολογιστών
- Δύσκολη ανάπτυξη παράλληλων αλγορίθμων
- Η βέλτιστη επικοινωνία μεταξύ των επεξεργαστών απαιτεί πανάκριβα δίκτυα

Υπολογιστική Ισχύς

FLOPS: πράξεις με αριθμούς κινητής υποδιαστολής (πραγματικούς αριθμούς) ανά δευτερόλεπτο (FLoating point Operations Per Second)

Kilo-FLOPS	(KFLOPS)	1.000 FLOPS
Mega-FLOPS	(MFLOPS)	1.000.000 FLOPS
Giga-FLOPS	(GFLOPS)	1.000.000.000 FLOPS
Tera-FLOPS	(TFLOPS)	1.000.000.000.000 FLOPS
Peta-FLOPS	(PFLOPS)	1.000.000.000.000.000 FLOPS



Ο No.1 supercomputer του κόσμου (Ιούνιος 2016)

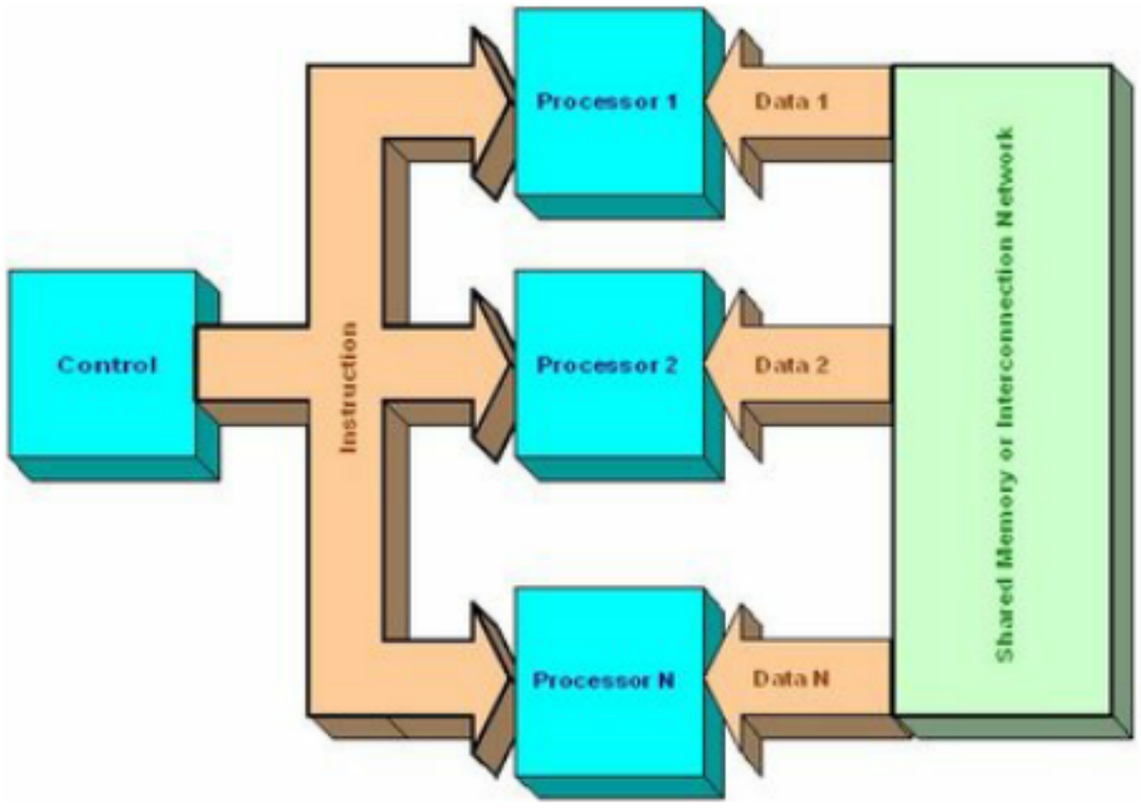
SUNWAY TAIHULIGHT (National Supercomputing Center, Wuxi China)

- 40.960 υπολογιστικοί κόμβοι
- 260-core επεξεργαστής σε κάθε κόμβο (3 TFLOPS)
- 10.649.600 cores
- 125 PFLOPS (peak performance), 93 PFLOPS (Linpack)
- 15 MW ηλεκτρική ισχύς

Αρχιτεκτονική παράλληλων συστημάτων

SIMD (Single Instruction Multiple Data)

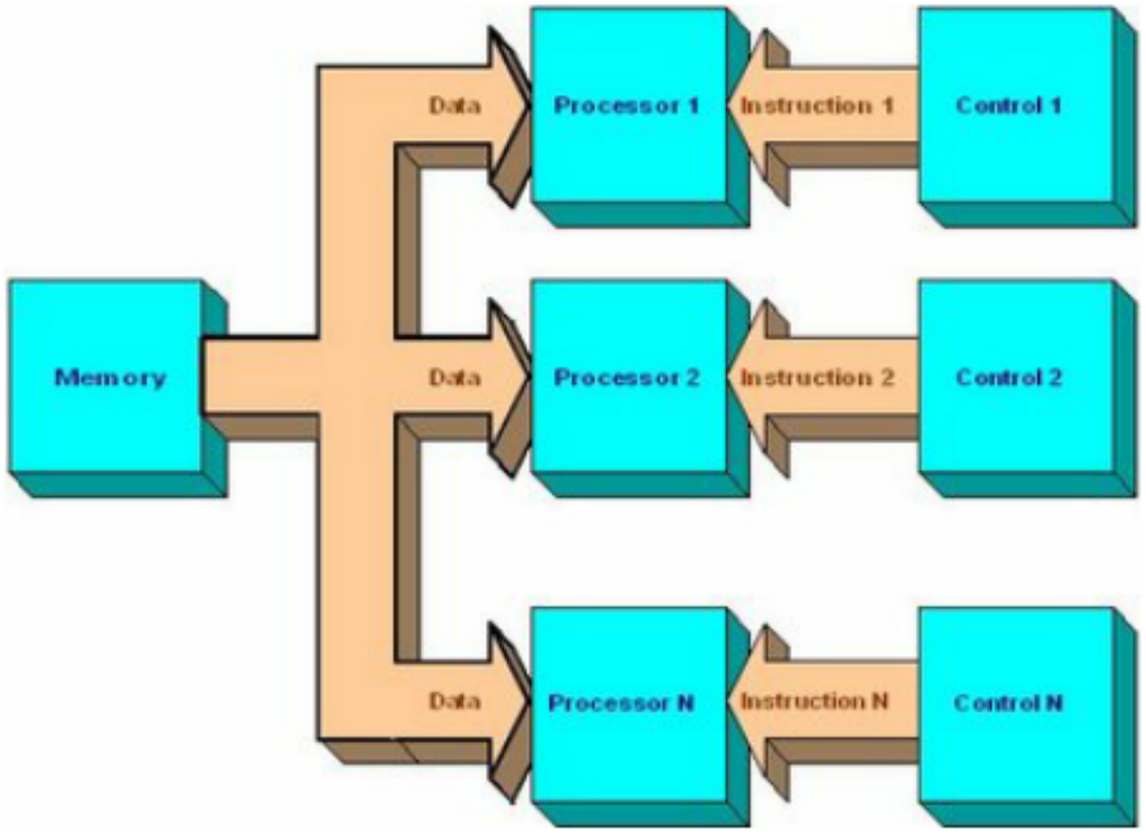
- Όλοι οι επεξεργαστές εκτελούν την ίδια εντολή με διαφορετικά δεδομένα
- Όλοι οι επεξεργαστές είναι συγχρονισμένοι στο ίδιο σημείο εκτέλεσης
- Ο master node ελέγχει τους υπόλοιπους επεξεργαστές
- Κατανεμημένης (distributed) ή κοινής (shared) μνήμης



Αρχιτεκτονική παράλληλων συστημάτων

MISD (Multiple Instruction Single Data)

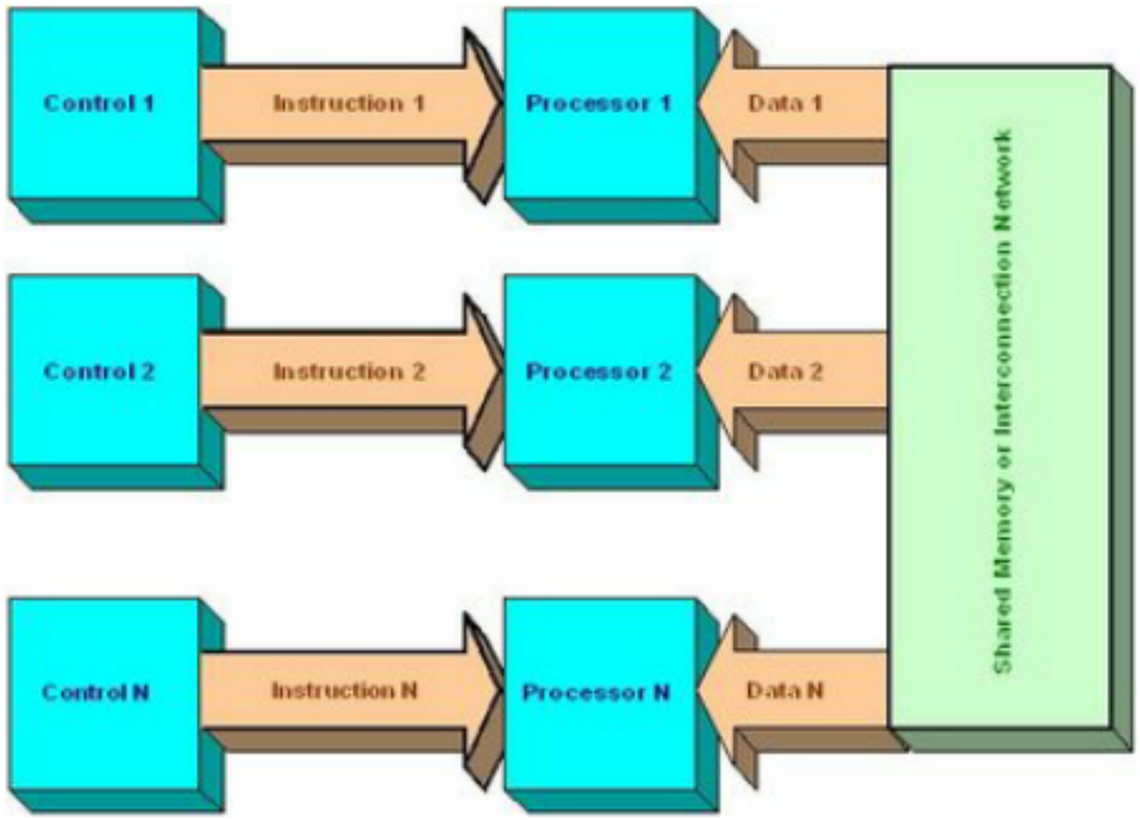
- Οι επεξεργαστές εκτελούν διαφορετικές εντολές με τα ίδια δεδομένα
- Δεν υπάρχουν πολλοί τέτοιοι υπολογιστές
  - Δύσκολη διαχείριση μνήμης
  - Μεγάλη επικοινωνία μεταξύ των επεξεργαστών



Αρχιτεκτονική παράλληλων συστημάτων

MIMD (Multiple Instruction Multiple Data)

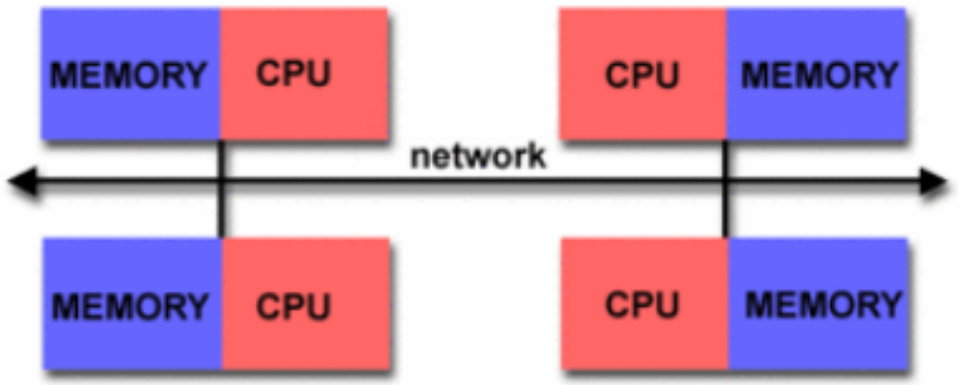
- Οι επεξεργαστές εκτελούν διαφορετικές εντολές με διαφορετικά δεδομένα
- Κάθε επεξεργαστής έχει ανεξάρτητο έλεγχο της εκτέλεσης
- Κάθε επεξεργαστής μπορεί να εκτελεί διαφορετικές εργασίες
- Κατανεμημένης (distributed) ή κοινής (shared) μνήμης

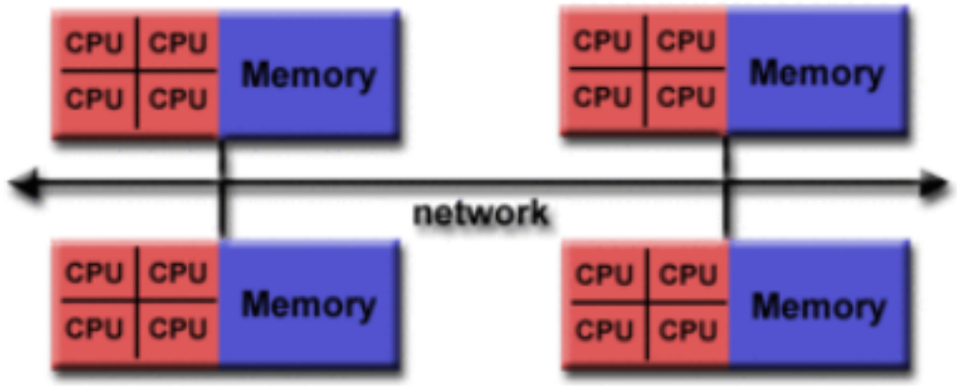


Αρχιτεκτονική παράλληλων συστημάτων

Κατανεμημένη μνήμη

- Κάθε επεξεργαστής έχει τη δικιά του μνήμη (private memory)
- Μπορούμε να χρησιμοποιήσουμε πολλές χιλιάδες επεξεργαστών
- Υπέρ: Ο μόνος τρόπος να φτάσουμε σε ισχύ PFLOPS
- Κατά: Δύσκολος προγραμματισμός

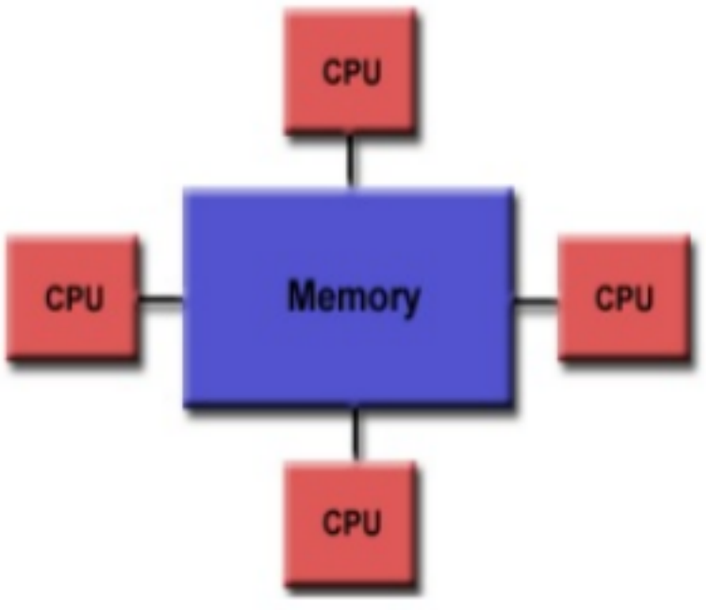




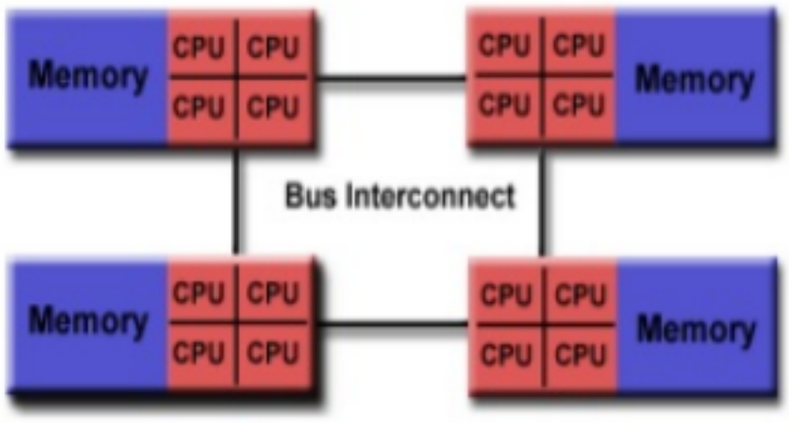
Αρχιτεκτονική παράλληλων συστημάτων

Κοινή μνήμη

- Όλοι οι επεξεργαστές μοιράζονται την ίδια μήμη
- Πολυεπεξεργαστές (multiprocessors)
- Υπέρ: Ο προγραμματισμός είναι συνήθως ευκολότερος
- Κατά: Πολύ δύσκολο να βάλουμε μαζί πολλούς επεξεργαστές (<64)



Uniform Memory Access (UMA)



Non-Uniform Memory Access(NUMA)

Εισαγωγή στο Message Passing Interface (MPI)

- Το MPI είναι μια βιβλιοθήκη που επιτρέπει την ανταλλαγή μηνυμάτων και τη μεταφορά δεδομένων μεταξύ των επεξεργαστών
- Είναι ανεπτυγμένο για συστήματα κοινής και κατανεμημένης μνήμης
- Είναι ανεπτυγμένο σε σχεδόν όλες τις αρχιτεκτονικές υπολογιστικών συστημάτων
- Κάθε υλοποίηση εκμεταλεύεται τα χαρακτηριστικά της αρχιτεκτονικής του hardware με αποτέλεσμα την καλύτερη απόδοση
- Διεθνές πρότυπο: Το MPI είναι η standard βιβλιοθήκη ανταλλαγής μηνυμάτων σε όλα τα υπολογιστικά συστήματα High Performace Computing
- Είναι Ελεύθερο Λογισμικό.
- Ιστορία και εξέλιξη:
  - 1980 - αρχές 1990: Ανομοιογενές λογισμικό. Προβλήματα μεταφοράς προγραμμάτων. Ανάγκη δημιουργίας διεθνούς προτύπου.
  - 1992 - 1994: Καθιέρωση του MPI σαν διεθνές πρότυπο.

Το πρόγραμμα "Hello World" που χρησιμοποιεί το MPI

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
print("Hello World!")
print("My name is", name, "and my rank is", comm.rank))
```

```
$ mpirun -np 4 -machinefile machinefile python example0.py
Hello World!
My name is blade02 and my rank is 3
Hello World!
My name is blade01 and my rank is 0
Hello World!
My name is blade02 and my rank is 1
Hello World!
My name is blade01 and my rank is 2
```

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
print("Hello World!")
print("My name is", name, "and my rank is", comm.rank))
```

- Διαχειριστής επικοινωνίας (Communicator) είναι ένα σύνολο διεργασιών που μπορούν να ανταλλάξουν μηνύματα
- `MPI.COMM_WORLD` είναι ο προκαθορισμένος διαχειριστής επικοινωνίας
- Συνήθως ο αριθμός των διεργασιών συμπίπτει με τον αριθμό των επεξεργαστών που θα χρησιμοποιήσουμε. Οι επεξεργαστές μπορεί να είναι οπουδήποτε.
- Ταξινόμηση των διεργασιών: μέσα στον Communicator οι διεργασίες αριθμούνται με ένα ακέραιο (rank) ξεκινώντας από το 0.
- Η ταξινόμιση χρησιμοποιείται από τον χρήστη για την επικοινωνία μεταξύ των διεργασιών (π.χ. αν το rank είναι 0 κάνε κάτι ενώ αν το rank είναι 1 κάνε κάτι άλλο)
- Οι εντολές I/O (π.χ. `print`) στις διεργασίες είναι κάκιστη πρακτική. Εμείς τις χρησιμοποιούμε για εκπαιδευτικούς σκοπούς.

```
$ mpirun -np 4 -machinefile machinefile python example0.py
Hello World!
My name is blade02 and my rank is 3
Hello World!
My name is blade01 and my rank is 0
Hello World!
My name is blade02 and my rank is 1
Hello World!
My name is blade01 and my rank is 2
```

- Η εντολή `mpirun` διανέμει το πρόγραμμά μας στους επεξεργαστές των διάφορων υπολογιστών που είναι διαθέσιμες
- Ο διακόπτης `-np` (number of processors) δηλώνει τον αριθμό των επεξεργαστών που θα χρησιμοποιήσουμε
- Ο διακόπτης `-machinefile` δηλώνει το αρχείο κειμένου που περιέχει τα ονόματα των υπολογιστών που θα χρησιμοποιήσουμε
- Τέλος είναι η εντολή που θα εκτελέσει η `mpirun` στους διάφορους επεξεργαστές
- Ο διακόπτης `-machinefile` μπορεί να παραλειφθεί οπότε θα χρησιμοποιηθεί μόνο ο τοπικός υπολογιστής

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
print("My name is", name, "and my rank is", rank)
if rank == 1:
    print("Doing the task of rank 1")
if rank == 0:
    print("Doing the task of rank 0")
```

```
$ mpirun -np 2 -machinefile machinefile python example1.py
My name is blade02 and my rank is 1
Doing the task of rank 1
My name is blade01 and my rank is 0
Doing the task of rank 0
```

- Χρήση του `comm.rank` για να καθοριστούν συγκεκριμένες εντολές ανάλογα με τον επεξεργαστή.

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
size = comm.size
print("My name is", name, "and my rank is", rank, "out of", size)
if rank % 2 == 0:
    print("Doing the task of an even rank")
else:
    print("Doing the task of an odd rank")
```

```
$ mpirun -np 4 -machinefile machinefile python example2.py
My name is blade02 and my rank is 1 out of 4
Doing the task of an odd rank
My name is blade01 and my rank is 2 out of 4
Doing the task of an even rank
My name is blade01 and my rank is 0 out of 4
Doing the task of an even rank
My name is blade02 and my rank is 3 out of 4
Doing the task of an odd rank
```

- Χρήση του `comm.size` για την εύρεση του συνολικού πλήθους των επεξεργαστών.

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
size = comm.size

data = (rank+1) * 5

if rank == 0:
    comm.send(data, dest=1)
    comm.send(data, dest=2)
    print("On node", name, "from rank", rank, "we sent", data)
elif rank == 1:
    data = comm.recv(source=0)
    print("On node", name, "from rank", rank, "we received", data)
elif rank == 2:
    data = comm.recv(source=0)
    print("On node", name, "from rank", rank, "we received", data)
```

```
$ mpirun -np 3 -machinefile machinefile python example3.py
On node blade01 from rank 2 we received 5
On node blade01 from rank 0 we sent 5
On node blade02 from rank 1 we received 5
```

- Χρήση των `comm.send` και `comm.recv` για την αποστολή δεδομένων μεταξύ των επεξεργαστών

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
size = comm.size
data = (rank+1) * 5
comm.send(data, dest=(rank+1)%size)
print("From node", name, "from rank", rank,
    "we sent", data, "to rank", (rank+1)%size)
data = comm.recv(source=(rank-1)%size)
print("On node", name, "with rank", rank,
    "we received", data, "from rank", (rank-1)%size)
```

```
$ mpirun -np 5 -machinefile machinefile python example4.py
From node blade01 from rank 4 we sent 25 to rank 0
On node blade01 with rank 4 we received 20 from rank 3
From node blade01 from rank 2 we sent 15 to rank 3
On node blade01 with rank 2 we received 10 from rank 1
From node blade02 from rank 3 we sent 20 to rank 4
On node blade02 with rank 3 we received 15 from rank 2
From node blade02 from rank 1 we sent 10 to rank 2
On node blade02 with rank 1 we received 5 from rank 0
From node blade01 from rank 0 we sent 5 to rank 1
On node blade01 with rank 0 we received 25 from rank 4
```

- Υλοποίηση "δακτυλιδιού": κάθε επεξεργαστής στέλνει δεδομένα στον επόμενο στην ταξινόμιση

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
size = comm.size

if rank == 0:
    data = {'a': 1, 'b': 2}
    comm.send(data, dest=1)
    print("From node", name, "and rank", rank, "we send", data)
if rank == 1:
    receive = comm.recv(source=0)
    print("On node", name, "and rank", rank, "we received")
    for key, value in receive.items():
        print(key, value)
```

```
$ mpirun -np 2 -machinefile machinefile python example5.py
From node blade01 and rank 0 we send {'b': 2, 'a': 1}
On node blade02 and rank 1 we received
a 1
b 2
```

- Η χρήση του `mpi4py` επιτρέπει την αποστολή και λήψη οποιουδήποτε αντικειμένου!

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
size = comm.size

if rank == 0:
    data1 = {'a': 1, 'b': 2}
    data2 = {'c': 3, 'd': 4}
    comm.send(data1, dest=1, tag=1)
    print("From node", name, "we send", data1)
    comm.send(data2, dest=1, tag=2)
    print("From node", name, "we send", data2)
if rank == 1:
    receive1 = comm.recv(source=0, tag=1)
    print("On node", name, "we received", receive1)
    receive2 = comm.recv(source=0, tag=2)
    print("On node", name, "we received", receive2)
```

```
$ mpirun -np 2 -machinefile machinefile python example6.py
From node blade01 we send {'a': 1, 'b': 2}
From node blade01 we send {'c': 3, 'd': 4}
On node blade02 we received {'b': 2, 'a': 1}
On node blade02 we received {'c': 3, 'd': 4}
```

- Χρήση ετικετών (tags) όταν ένας επεξεργαστής στέλνει πολλές φορές δεδομένα

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank

if rank == 0: data = {'a':1, 'b':2, 'c':3}
else: data = None

data = comm.bcast(data, root=0)

print("On node", name, "with rank", rank, "we received", data)
```

```
$ mpirun -np 5 -machinefile machinefile python example7.py
On node blade01 with rank 4 we received {'c': 3, 'a': 1, 'b': 2}
On node blade02 with rank 1 we received {'b': 2, 'c': 3, 'a': 1}
On node blade01 with rank 0 we received {'a': 1, 'b': 2, 'c': 3}
On node blade02 with rank 3 we received {'b': 2, 'a': 1, 'c': 3}
On node blade01 with rank 2 we received {'a': 1, 'c': 3, 'b': 2}
```

- Χρήση του `comm.bcast` για την ταυτόχρονη αποστολή σε όλους τους επεξεργαστές

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
size = comm.size

if rank == 0:
    data = [(x+1)**x for x in range(size)]
    print("We will be scattering", data)
else: data = None

data = comm.scatter(data, root=0)

print("On node", name, "rank", rank, "received", data)
```

```
$ mpirun -np 5 -machinefile machinefile python example8.py
On node blade01 rank 4 received 625
On node blade02 rank 1 received 2
We will be scattering [1, 2, 9, 64, 625]
On node blade01 rank 0 received 1
On node blade01 rank 2 received 9
On node blade02 rank 3 received 64
```

- Χρήση του `comm.scatter` για το μοίρασμα δεδομένων στους επεξεργαστές

Εισαγωγή στο Message Passing Interface (MPI)

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
size = comm.size

if rank == 0:
    data = [(x+1)**x for x in range(size)]
    print("We will be scattering", data)
else: data = None

data = comm.scatter(data, root=0)

data += 1   # Π.χ. μια απαιτητική υπολογιστικά επεξεργασία

new_data = comm.gather(data, root=0)

if rank == 0:
    print("Master received", new_data)
```

```
$ mpirun -np 5 -machinefile machinefile python example9.py
We will be scattering [1, 2, 9, 64, 625]
Master received [2, 3, 10, 65, 626]
```

- Χρήση του `comm.scatter` για το μοίρασμα δεδομένων στους επεξεργαστές
- `data += 1` αναπαριστά μια απαιτητική υπολογιστικά επεξεργασία
- Χρήση του `comm.gather` για τη συλλογή των επεξεργασμένων δεδομένων

"Embarrassigly parallel" προβλήματα

- Είναι τα προβλήματα που δεν χρειάζεται καμιά προσπάθεια για τον παράλληλο προγραμματισμό της λύσης τους
- Αυτό συνήθως σημαίνει ότι η σειριακή λογική μεταφέρεται σχεδόν αυτούσια στο παράλληλο πρόγραμμα
- Οι παράλληλες διεργασίες είναι ανεξάρτητες μεταξύ τους και δεν χρειάζεται μεταξύ τους επικοινωνία
- Εύκολο να υλοποιηθούν και να εκτελεστούν σε συμβατικά server farms που δεν διαθέτουν το πανάκριβο δίκτυο των πραγματικών υπερυπολογιστών
- Η ακριβώς αντίθετη έννοια είναι τα εγγενώς σειριακά προβλήματα που δεν μπορούν καθόλου να παραλληλιστούν

Embarrassigly parallel πρόβλημα: Να γραφτεί παράλληλο πρόγραμμα σε Python που να αθροίζει τους αριθμούς $0, 1, \ldots, n-1$

"Embarrassigly parallel" προβλήματα

- Είναι τα προβλήματα που δεν χρειάζεται καμιά προσπάθεια για τον παράλληλο προγραμματισμό της λύσης τους
- Αυτό συνήθως σημαίνει ότι η σειριακή λογική μεταφέρεται σχεδόν αυτούσια στο παράλληλο πρόγραμμα
- Οι παράλληλες διεργασίες είναι ανεξάρτητες μεταξύ τους και δεν χρειάζεται μεταξύ τους επικοινωνία
- Εύκολο να υλοποιηθούν και να εκτελεστούν σε συμβατικά server farms που δεν διαθέτουν το πανάκριβο δίκτυο των πραγματικών υπερυπολογιστών
- Η ακριβώς αντίθετη έννοια είναι τα εγγενώς σειριακά προβλήματα που δεν μπορούν καθόλου να παραλληλιστούν

Embarrassigly parallel πρόβλημα: Να γραφτεί παράλληλο πρόγραμμα σε Python που να αθροίζει τους αριθμούς $0, 1, \ldots, n-1$

- Αν διαθέτουμε k επεξεργαστές χωρίζουμε τους n αριθμούς σε k σύνολα των $\frac{n}{k}$ αριθμών
- Κάθε επεξεργαστής λαβάνει $\frac{n}{k}$ αριθμούς και τους αθροίζει
- Στο τέλος αθροίζονται τα μερικά αθροίσματα

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()
rank = comm.rank
size = comm.size
if rank == 0:
    data = [x+1 for x in range(100)]
    nums_per_rank = 100//size
    for r in range(1,size):
        comm.send(data[r*nums_per_rank:(r+1)*nums_per_rank], dest=r)
    psum = sum(data[:nums_per_rank])
    print("Master node", name, "rank", rank, "psum is", psum)
    for r in range(1,size): psum += comm.recv(source=MPI.ANY_SOURCE)
    print("Grand Total is", psum)
else:
    subdata = comm.recv(source=0)
    psum = sum(subdata)
    comm.send(psum, dest=0)
    print("On node", name, "rank", rank, "psum is", psum)
```

```
$ mpirun -np 5 -machinefile machinefile python example10.py
On node blade01 rank 2 psum is 1010
On node blade02 rank 1 psum is 610
On node blade01 rank 4 psum is 1810
Master node blade01 rank 0 psum is 210
Grand Total is 5050
On node blade02 rank 3 psum is 1410
```

Ερωτήσεις;