

# **Lygiagretusis programavimas naudojant MPI**

Anyone can build a fast CPU. The trick is to build a fast system.<sup>1</sup>

---

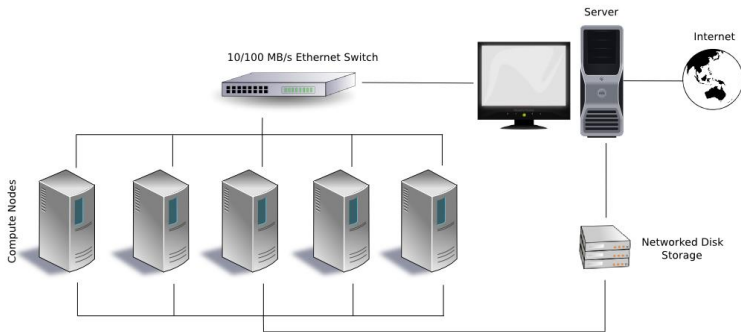
<sup>1</sup>Seymour Cray (1925 – 1996) – superkompiuterių architektas, Cray Inc. įkūrėjas

# Klasteriai ir superkompiuteriai

# Pagrindiniai terminai

- **Node** (*node – mazgas*) – A computer in the traditional sense: a desktop or laptop PC, or a server in any incarnation, including a self-standing pedestal, a rack module, or a blade, containing one or more central processing units.
- **Cluster** (*cluster – blokinys, grupelė*) – A collection of related nodes.
- **Grid** (*grid – tinklėlis*) – A collection of clusters.
- **Supercomputer** – A computer that leads the world in terms of processing capacity, particularly speed of calculation, at the time of its introduction.

# Klasteris<sup>2</sup> (1)



## Klasteris (2)

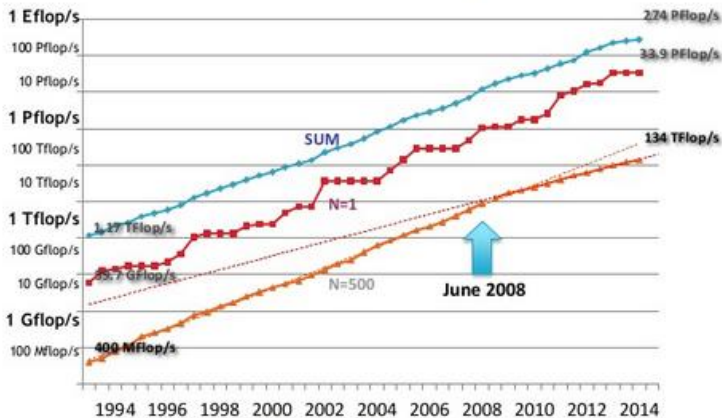


## Klasteris (3)



# Superkompiuterių pajėgumai<sup>3</sup>

## Performance Development

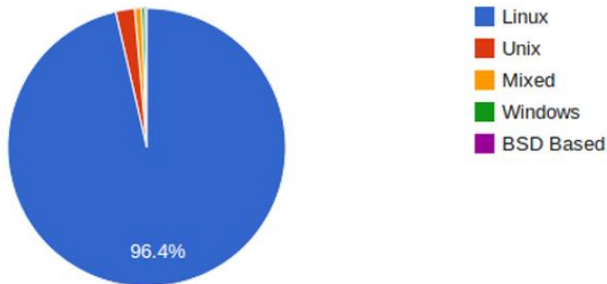


<sup>3</sup><https://www.nchc.org.tw/en/news>



# Superkompiuterių OS<sup>4</sup>

Operating system Family System Share



---

<sup>4</sup>2013 m. lapkritis

# Superkompiuteris Sunway TaihuLight - Sunway MPP<sup>5</sup> (1)

Site:	National Super Computer Center in Wuxi
Manufacturer:	NRCP
Cores:	10,649,600
Power:	15,371.00 kW
Installation Year:	2016
Memory:	1,310,720 GB
Operating System:	Sunway RaiseOS 2.0.5

---

<sup>5</sup>2016-06 duomenys iš [www.top500.org](http://www.top500.org)

## Superkompiuteris Sunway TaihuLight (2)



# Superkompiuteris Tianhe-2 (MilkyWay-2)<sup>6</sup> (1)

Site:	National Super Computer Center in Guangzhou
Manufacturer:	NUDT
Cores:	3120000
Power:	17808.00 kW
Installation Year:	2013
Memory:	1024000 GB
Operating System:	Kylin Linux

---

<sup>6</sup>2015-11 duomenys iš [www.top500.org](http://www.top500.org)

## Superkompiuteris Tianhe-2 (2)



# Superkompiuteris Titan - Cray XK7<sup>7</sup> (1)

System Name:	Titan
Site:	Oak Ridge National Laboratory
Manufacturer:	Cray Inc.
Cores:	560640
Power:	8209.00 kW
Installation Year:	2013
Memory:	710144 GB
Operating System:	Cray Linux

---

<sup>7</sup>duomenys iš [www.top500.org](http://www.top500.org)

# Superkompiuteris Titan Cray (2)



# Apie MPI



# MPI paskirtis

- MPI – **M**essage **P**assing **I**nterface.
- MPI – tai pranešimų perdavimo funkcijų, skirtų realizuoti lygiagrečiuosius algoritmus, standartas.
- MPI realizacija – tai MPI priemonių biblioteka klasteriui, paskirstytos atminties superkompiuteriui ar heterogeniniam tinklui.
- Programavimo kalbos: C, C++, FORTRAN ir kt.

# MPI darbo principai

- SPMD – **S**ingle **P**rogram **M**ultiple **D**ata.
- Procesai (programos kopijos) sukuriama prieš programos vykdymą.
- Procesų sk.  $\geq$  superkompiuterio (klasterio) procesorių sk.
- Procesai **neturi** bendros atminties.
- Ryšys tarp procesų – siunčiant ir priimant pranešimus (*message*).

# MPI istorija

- **MPI 1.0** – 1994 m. birželis: apibrėžtas ir standartizuotas MPI funkcijų rinkinys,
- **MPI 1.1** – 1995 m. birželis: ištaisytos klaidos ir atlikti papildomi išaiškinimai,
- **MPI 1.2** – 1997 m. liepa: įtrauktos naujos funkcijos,
- **MPI 2.0** – 1997 m. liepa: papildomas funkcionalumas (dinaminis procesų valdymas ir kt.),
- **MPI 2.2** – 2009 m. rugsėjis: ištaisytos klaidos ir atlikti papildomi išaiškinimai,
- **MPI 3.0** – 2012 m. rugsėjis: išplėsta 2.2 versija, tačiau atsisakyta C++,
- **MPI 3.1** – 2015 m. birželis: naujos funkcijos, ištaisyti netikslumai.

# MPI realizacijos

- Komerčinės
- Gamintojų
- Nekomerčinės

# Nekomercinės MPI realizacijos

- Open MPI (2.0.1, 2016) - Los Alamos National Lab, Sun Microsystems ir kt.
- LAM/MPI (7.1.4, 2006) – Indiana University.
- MPICH (3.2, 2015) – Ohio State University ir kt.
- MP-MPICH (1.5.0, 2007) – RWTH: Scalable Computing, Aachen.

# Interneto šaltiniai

- Message Passing Interface Forum,  
<http://www.mpi-forum.org/>.
- Open MPI: Open Source High Performance Computing,  
<http://www.open-mpi.org/>.
- MPICH - High-Performance Portable MPI,  
<http://www.mpich.org/>.
- MP-MPICH - MPI for heterogeneous Clusters,  
<http://www.lfbs.rwth-aachen.de/content/172.html>.

# MPI struktūra ir galimybės

- MPI 1 – virš **120** funkcijų.
- MPI 2 – virš **150** funkcijų.
- MPI 3 – apie **400** funkcijų.
- Daugumai lygiagrečiųjų programų pakanka **6** pagrindinių funkcijų.

# Pagrindinės MPI funkcijos

- **Init()** – nurodo MPI darbo pradžią.
- **Finalize()** – nurodo MPI darbo pabaigą.
- **Comm\_size()**, **Get\_size()** – grąžina procesų, vykdančių skaičiavimus, kiekį.
- **Comm\_rank()**, **Get\_rank()** – grąžina proceso numerį.
- **Send()** – siunčia pranešimą.
- **Recv()** – priima pranešimą.



# Init() funkcija

Inicializuoja MPI:

```
int MPI_Init(int *argc, char ***argv);
```

```
void MPI::Init(int &argc, char **&argv);
```

# Finalize() funkcija

Nutraukia visus MPI veiksmus:

```
int MPI_Finalize();
```

```
void MPI::Finalize();
```

# Get\_rank() funkcija

Grąžina proceso numerį nurodytame komunikatoriuje:

```
int MPI_Comm_rank(MPI_Comm comm, int *rank);
```

```
int MPI::Comm::Get_rank() const;
```

## Get\_size() funkcija

Grąžina komunikatoriaus procesų skaičių:

```
int MPI_Comm_size(MPI_Comm comm, int *size);
```

```
int MPI::Comm::Get_size() const;
```

## Get\_processor\_name() funkcija

Grąžina procesoriaus, kuriame vyksta procesas, vardą:

```
int MPI_Get_processor_name(char *name, int *len);
```

```
void MPI::Get_processor_name(char *name, int &len);
```

# MPI C, C++ programos struktūra

```
#include <mpi.h>
...
int main(int argc, char **argv) {
    ...
    // Negalima kviesti jokių MPI funkcijų
    // prieš šią eilutę
    MPI_Init(&argc, &argv);
    ...
    MPI_Finalize();
    // Negalima kviesti jokių MPI funkcijų
    // už šios eilutės
    ...
}
```

# MPI realizacijos

# Open MPI

- Projektas, apjungiantis kituose projektuose (FT-MPI, LA-MPI, LAM/MPI ir PACX-MPI) taikytas technologijas.
- Naudojama daugelyje TOP500 superkompiuterių.
- Atviro kodo programinė įranga, realizuojanti MPI-3 standartą.
- Palaikoma įvairiose lygiagrečiųjų skaičiavimų aplinkose.



# mpiCC komanda

Kompiluoja MPI programą (C++):

```
mpiCC -o vardas vardas.cc
```

# mpirun komanda

Sukuria ir pradeda vykdyti **x** procesų-programos kopijų:

```
mpirun [options] -np x my_mpi_program
```

arba:

```
mpirun [options] -np x -npty my_mpi_program
```

# Open MPI programų vykdymas

- Procesai (programos kopijos) gali būti vykdomi tiek viename kompiuteryje, tiek keliuose kompiuteriuose.
- Jei vykdoma keliuose kompiuteriuose, prieš vykdymą reikia sukurti šių kompiuterių sąrašo failą.

# Open MPI programos kompiliavimas ir vykdymas

```
...> mpiCC -o vardas vardas.cc  
...> mpirun -np x vardas // viename kompiuteryje  
...> mpirun --hostfile hosts -np x vardas  
      // kompiuteriuose, kurių vardai nurodyti faile hosts
```

# LAM/MPI

- Programinė įranga, skirta darbui heterogeniame kompiuterių tinkle.
- Atviro kodo programinė įranga, realizuojanti MPI-1 standartą (MPI-2 – ne pilnai).
- Palaikoma įvairiose lygiagrečiųjų skaičiavimų aplinkose.
- Toliau nebevystoma, kadangi kūrėjai prisidėjo prie Open MPI kūrimo ir vystymo.

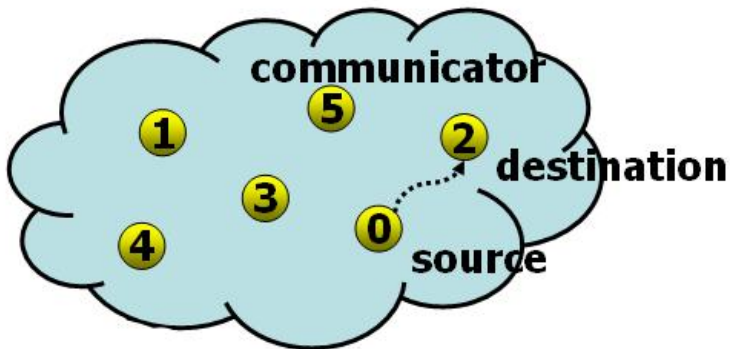
# MPI

## *Point-to-Point* komunikavimas

# Sąvokos (1)

- Komunikavimas vyksta tarp dviejų procesų (*Point-to-Point*).
- *Source* procesas **siunčia** žinutę (pranešimą) *destination* procesui.
- *Destination* procesas **priima** pranešimą.
- Komunikavimas vyksta *communicator* viduje.
- *Destination* procesą identifikuoja jo numeris (*rank*) komunikatoriuje.

## Sąvokos (2)





# Siuntimas

```
void Comm::Send(const void* buf, int count,  
                const Datatype& datatype, int dest, int tag) const
```

- **Comm** – komunikatorius – procesų rinkinys, kuriame siunčiama
- **buf** – siunčiamos informacijos pradžios adresas
- **count** – siunčiamos informacijos kiekis
- **datatype** – siunčiamos informacijos tipas (MPI tipas)
- **dest** – proceso, kuriam siunčiama, numeris (*rank*)
- **tag** – pranešimo skiriamoji žymė

# Priėmimas

```
void Comm::Recv(const void* buf, int count,  
                const Datatype& datatype, int source,  
                int tag, Status& status) const
```

- **Comm** – komunikatorius, kuriame siunčiama
- **buf** – atminties, kur išsaugomas pranešimas, adresas
- **count** – atminties dydis
- **datatype** – duomenų tipas (MPI tipas)
- **source** – proceso, kuris siunčia, numeris
- **tag** – pranešimo skiriamoji žymė
- **status** – informacija apie gautą pranešimą

# MPI komunikatoriai

- **Communicator** – tai abstrakti struktūra, nusakanti MPI procesų, kurie gali komunikuoti tarpusavyje, grupę.
- **MPI::COMM\_WORLD** – tai visų galimų procesų rinkinys.
- Programuotojas gali sukurti naują komunikatorių kaip MPI::COMM\_WORLD poaibį.
- **Rank** – proceso numeris – apibrėžiamas duoto komunikatoriaus ribose.
- **Pranešimai** – tik tarp to paties komunikatoriaus procesų.

# MPI duomenų tipai

MPI::CHAR

MPI::INT

MPI::SIGNED\_CHAR

MPI::UNSIGNED\_CHAR

MPI::UNSIGNED\_SHORT

MPI::UNSIGNED\_LONG

MPI::LONG\_DOUBLE

MPI::PACKED

MPI::COMPLEX

MPI::LONG\_DOUBLE\_COMPLEX

MPI::SHORT

MPI::LONG

MPI::UNSIGNED

MPI::DOUBLE

MPI::FLOAT

MPI::BOOL

MPI::BYTE

MPI::WCHAR

MPI::DOUBLE\_COMPLEX

# MPI proceso numeris

- Kiekvienas procesas komunikatoriuje turi unikalų numerį (`rank`).
- Procesai numeruojami nuo `0` iki (`size-1`), kur `size` - procesų skaičius komunikatoriuje.
- `int MPI::COMM_WORLD.Get_rank();`  
`int MPI::COMM_WORLD.Get_size();`
- Numeris naudojamas procesams identifikuoti `MPI::Send` ir `MPI::Recv` komandose.
- `MPI::Recv` gali priimti pranešimą iš bet kurio proceso (`MPI::ANY_SOURCE`).

## Pranešimo žymė (tag)

- Visi siunčiami pranešimai turi žymes – sveikuosius skaičius.
- `MPI::Recv` priima pranešimą tik su nurodyta žyme.
- `MPI::ANY_TAG` gali būti naudojama priimti pranešimą su bet kokia žyme.

## MPI :: Send – MPI :: Recv

- Vykdančiam `MPI :: Send`, "užregistruojamas" siuntimas.
- Vykdančiam `MPI :: Recv`, "užregistruojamas" priėmimas.
- "Registruotas" `MPI :: Recv` atitinka "registruotą" `MPI :: Send`, jei:
  - `MPI :: Send` kreipinio `destination` atitinka priimančią procesą;
  - `MPI :: Recv` kreipinio `source` atitinka siunčiantį procesą arba `source` yra `MPI :: ANY_SOURCE`;
  - `MPI :: Send` kreipinio `tag` atitinka `MPI :: Recv` kreipinio `tag` arba `MPI :: Recv` kreipinio `tag` yra `MPI :: ANY_TAG`;
  - `MPI :: Send` kreipinio `communicator` atitinka `MPI :: Recv` kreipinio `communicator`.

# Priėmimo buferio dydis

- Priėmimo buferis – paprastas kintamasis arba masyvas.
- Jei pranešimo ilgis mažesnis už buferio dydį, užpildoma tik dalis buferio.
- Jei pranešimo ilgis didesnis už buferio dydį, gaunama perpildymo klaida.
- `MPI::Probe` komanda galima nustatyti pranešimo ilgį prieš priėmimą:

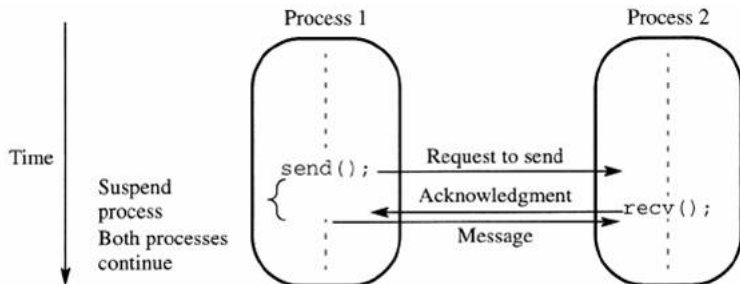
```
MPI::Probe(1, 99, MPI::COMM_WORLD, &status);  
// source, tag, communicator, status
```



# Send/Recv sinchronizavimas

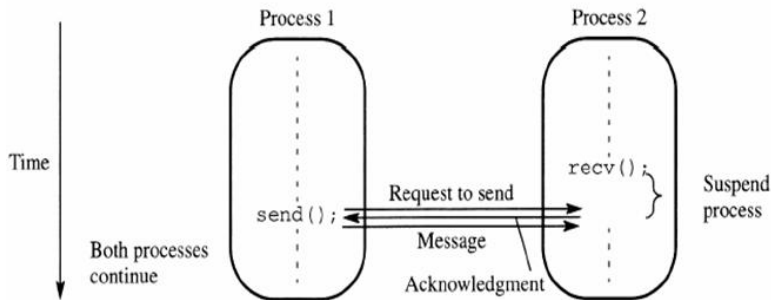
- standartos neapibrėžia sinchronizavimo pilnumo;
- galimi variantai:
  - *Fully Synchronized* (*Rendezvous*) – pilnas siuntėjo ir gavėjo sinchronizavimas,
  - *Buffered* – gavėjas laukia pranešimo, siuntėjas laukia, kol pranešimas bus patalpintas į buferį.

## Send/Recv Fully Synchronized (a)



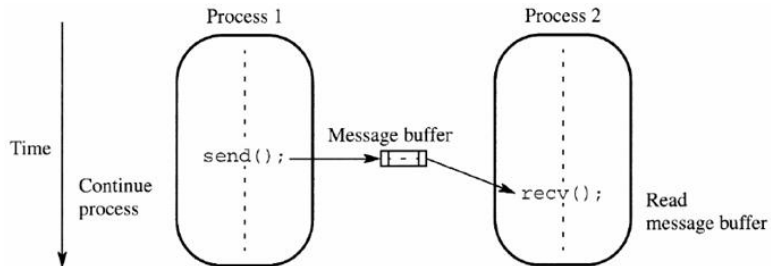
(a) When `send()` occurs before `recv()`

## Send/Recv Fully Synchronized (b)

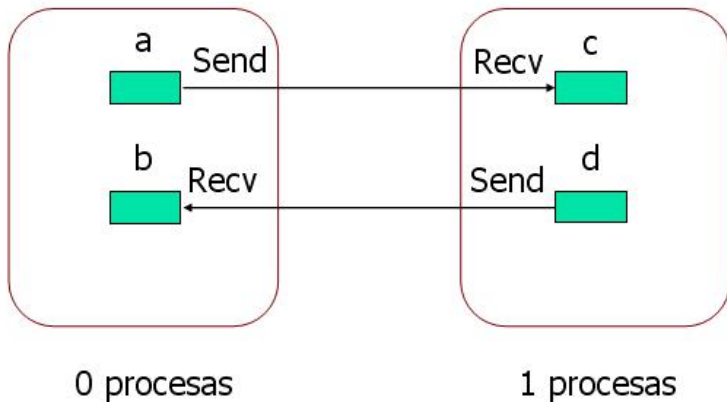


(b) When `recv()` occurs before `send()`

# Send/Recv Buffered



## Send/Recv 1 pavyzdys: apsikeitimas duomenimis, size=2



## Send/Recv 1 pavyzdys (a)

```
// Iliustruoja Send, Recv veikimą.  
// Dirba du procesai.  
#include <iostream>  
#include <mpi.h>  
using namespace std;  
int main(int argc, char *argv[])  
{  
    MPI::Init(argc, argv);  
    int rank = MPI::COMM_WORLD.Get_rank();  
    char name[MPI::MAX_PROCESSOR_NAME];  
    int namelength = 0;  
    MPI::Get_processor_name(name, namelength);
```

## Send/Recv 1 pavyzdys (b)

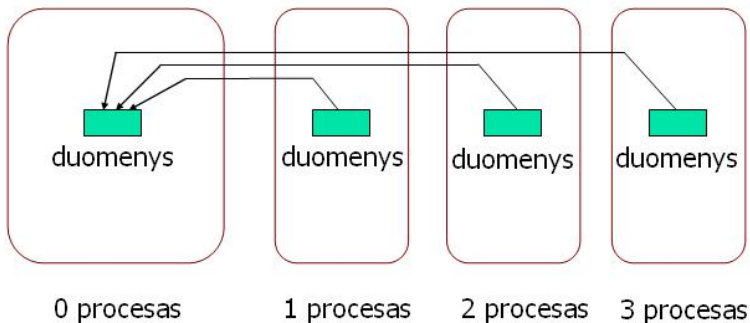
```
if (rank == 0) { // Dirba 0 procesas
    cout << "*** Procesu sk.: 2 ***" << endl;
    cout << "Dirba 0 proc.Komp.:" << name << "\n";
    int a = 0; int b = 0;
    MPI::COMM_WORLD.Send(&a, 1, MPI::INT, 1, 1);
    cout << name << ":issiunčiau a:" << a << "\n";
    MPI::COMM_WORLD.Recv(&b, 1, MPI::INT, 1, 1);
    cout << name << ":priėmiau b:" << b << "\n";
}
```

## Send/Recv 1 pavyzdys (c)

```
else { // Dirba 1 procesas
    cout << "Dirba 1 proc.Komp.:" << name << "\n";
    int c = 1; int d = 1;
    MPI::COMM_WORLD.Recv(&c, 1, MPI::INT, 0, 1);
    cout << name << ":priėmiau c:" << c << "\n";
    MPI::COMM_WORLD.Send(&d, 1, MPI::INT, 0, 1);
    cout << name << ":issiunėiau d:" << d << "\n";
}
MPI::Finalize();
return 0;
}
```



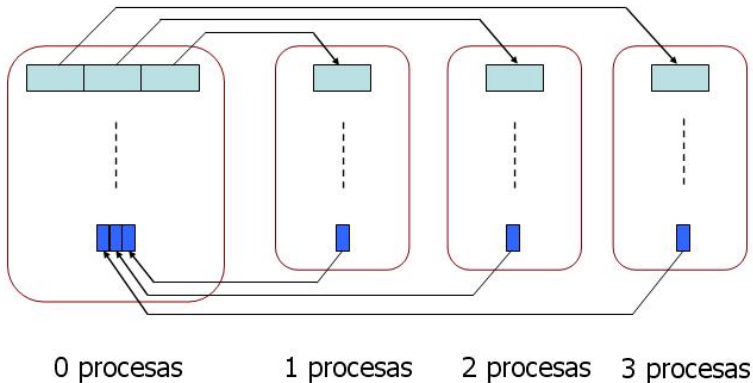
## Send/Recv 2 pavyzdys: duomenų surinkimas, size=4



## Send/Recv 2 pavyzdys (a)

```
// Iliustruoja Send, Recv veikimą.  
// Dirba mpirun nurodytas skaičius procesų.  
...  
char d[256];  
sprintf(d, "Dirba %d proc.Komp.vardas:%10s", rank, name);  
if (rank == 0) { // Dirba 0 procesas  
    for(int i=1; i<size; i++) {  
        MPI::COMM_WORLD.Recv(d, sizeof(d), MPI::CHAR,  
                               MPI::ANY_SOURCE, 1);  
        // MPI::COMM_WORLD.Recv(d, sizeof(d), MPI::CHAR, i, 1);  
        cout << d << endl;  
    }  
}  
else // Dirba 1 - (size-1) procesai  
    MPI::COMM_WORLD.Send(d, strlen(d)+1, MPI::CHAR, 0, 1);  
...
```

## Send/Recv 3 pavyzdys: sumos skaičiavimas, size=4



## Send/Recv 3 pavyzdys (a)

```
// Skaičiuoja masyvo elementų sumą, kai
// elementų sk. kartotinis (size-1).
// Dalines sumas skaičiuoja 1-(size-1) procesai.
...
int main(int argc, char *argv[]) {
    MPI::Init(argc, argv);
    int rank = MPI::COMM_WORLD.Get_rank();
    if (rank == 0)
        PagrindinisProcesas();
    else
        DirbantisProcesas(rank);
    MPI::Finalize();
    return 0;
}
```









## Send/Recv 3 pavyzdys (b)

```
void PagrindinisProcesas() {  
    int size = MPI::COMM_WORLD.Get_size();  
    ...  
    int Masyvas[1000], m = 0;  
    IvestiDuomenis("Duomenys.txt", Masyvas, m);  
    int kS = m/(size-1); // Kiek siųsti  
    int S[100]; // Dalines sumos  
    for(int i=1; i<size; i++)  
        MPI::COMM_WORLD.Send(&kS, 1, MPI::INT, i, 1);  
    for(int i=1; i<size; i++)  
        MPI::COMM_WORLD.Send(Masyvas+kS*(i-1), kS, MPI::INT, i, 2);  
    for(int i=1; i<size; i++)  
        MPI::COMM_WORLD.Recv(&S[i-1], 1, MPI::INT, MPI::ANY_SOURCE, 1);  
}
```

## Send/Recv 3 pavyzdys (c)

```
void DirbantisProcesas(int nr) {  
    ...  
    int A[100], n = 0;  
    MPI::COMM_WORLD.Recv(&n, 1, MPI::INT, 0, 1);  
    MPI::COMM_WORLD.Recv(A, n, MPI::INT, 0, 2);  
    int sum = Suma(A, n);  
    MPI::COMM_WORLD.Send(&sum, 1, MPI::INT, 0, 1);  
}
```

## Send/Recv kombinacijos

0 procesas	1 procesas	Situacija
Send į 1 proc.  Recv iš 1 proc. 	Recv iš 0 proc. Send į 0 proc.	Aklavietės nėra
Send į 1 proc.  Recv iš 1 proc. 	Send į 0 proc. Recv iš 0 proc. 	Gali susidaryti aklavieta
Recv iš 1 proc.  Send į 1 proc. 	Recv iš 0 proc. Send į 0 proc. 	Aklavieta

# Klausimai pakartojimui (1)

- 1 Kompiuterių klasteris ir superkompiuteris – kas tai?
- 2 Kokia yra pagrindinė MPI paskirtis?
- 3 Kokios yra pagrindinės MPI funkcijos?
- 4 Kuo pasižymi MPI point-to-point komunikavimas?
- 5 Kas yra MPI komunikatorius?
- 6 Kuo skiriasi pilnai sinchronizuotas ir buferizuotas siuntimas?
- 7 Kokioje situacijoje Send/Recv siuntimo metu susidaro aklavietė?