Ministerstwo Nauki
i Szkolnictwa Wyższego

Regionalna Inicjatywa Doskonałości w Dyscyplinach
Informatyki, Elektrotechniki, Elektroniki, Automatyki i Robotyki
na Politechnice Częstochowskiej

Projekt w ramach programu Regionalna Inicjatywa Doskonałości, decyzja nr 020/RID/2018/19

Programowanie równoległe z wykorzystaniem współczesnych architektur komputerowych z pamięcią współdzieloną

Dr hab. inż. Łukasz Szustak
lszustak@icis.pcz.pl

Dr inż. Kamil Halbiniak
khalbniak@icis.pcz.pl

Politechnika Częstochowska

Programowanie równoległe z wykorzystaniem współczesnych architektur komputerowych z pamięcią współdzieloną

1. Wprowadzenie do współczesnych architektur komputerowych oraz programowania równoległego
2. Przegląd modeli i standardów programowania równoległego z pamięcią współdzieloną
3. Środowisko OpenMP programowania maszyn z pamięcią wspólna, w tym procesorów wielordzeniowych
4. **Zrównoleglanie pętli oraz równoważenie obciążenia w systemach równoległych z pamięcią współdzieloną**
5. Zrównoleglanie programów z wykorzystaniem modelu typu data parallelism
6. Zrównoleglanie programów z wykorzystaniem modelu typu task parallelism
7. Mapowanie obliczeń w hybrydowych systemach komputerowych z akceleratorami
8. Przykłady zrównoleglania obliczeń numerycznych

Ministerstwo Nauki
i Szkolnictwa Wyższego

Regionalna Inicjatywa Doskonałości w Dyscyplinach
Informatyki, Elektrotechniki, Elektroniki, Automatyki i Robotyki
na Politechnice Częstochowskiej

Projekt w ramach programu Regionalna Inicjatywa Doskonałości, decyzja nr 020/RID/2018/19

Temat nr 4

Zrównoleglanie pętli oraz równoważenie obciążenia w systemach równoległych z pamięcią współdzieloną

Dr hab. inż. Łukasz Szustak
lszustak@icis.pcz.pl

Politechnika Częstochowska

Plan prezentacji

- Manualne metody dystrybucji obliczeń realizowanych w ramach pętli
- Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli
- Mechanizmy równoważenia obciążenia
- Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

Plan prezentacji

- **Manualne metody dystrybucji obliczeń realizowanych w ramach pętli**
- Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli
- Mechanizmy równoważenia obciążenia
- Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

- Równoległe wykonanie pętli może zostać zrealizowane w różny sposób
- Na przykład, możemy utworzyć równoległy region wokół pętli i dostosować zakres iteracji dobierając jego rozmiar indywidualnie każdemu wątku
- Realizacja tego celu wiążę się z określeniem sposobu redystrybucji obliczeń pomiędzy dostępne wątki w regionie równoległym
- Minusem manualnej dystrybucji obliczeń jest jawne wyznaczenie granic czy też zakresu wewnątrz pętli

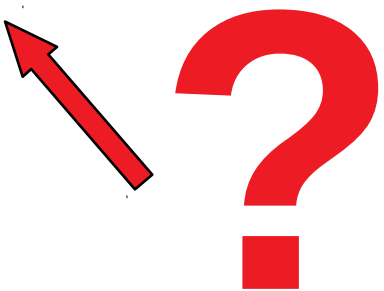
```
for(int i=0; i<n; ++i) {
    cout<<i<<" ";
}
cout<<endl;
```

Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        for(int i=0; i<n; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

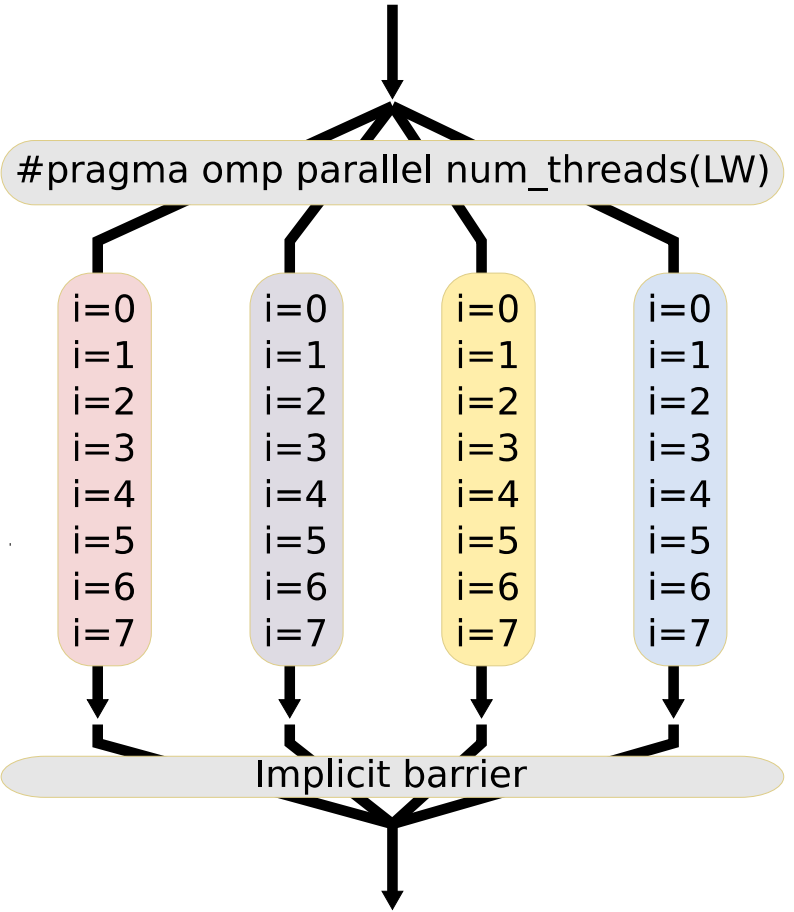
Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        for(int i=0; i<n; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```



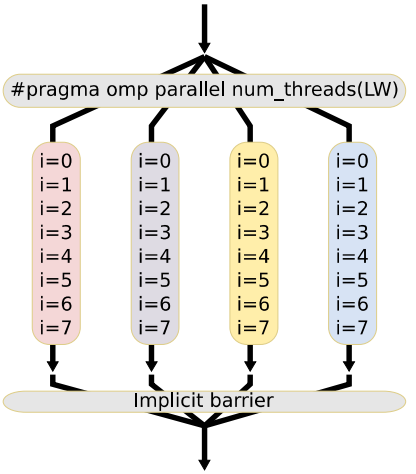
Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        for(int i=0; i<n; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```



Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        for(int i=0; i<n; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```



Należy jawnie i indywidualnie wyznaczyć zakres iteracji pętli dla każdego wątku

Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Krok 1

Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Krok 2

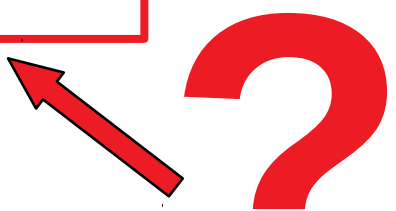
Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Krok 3

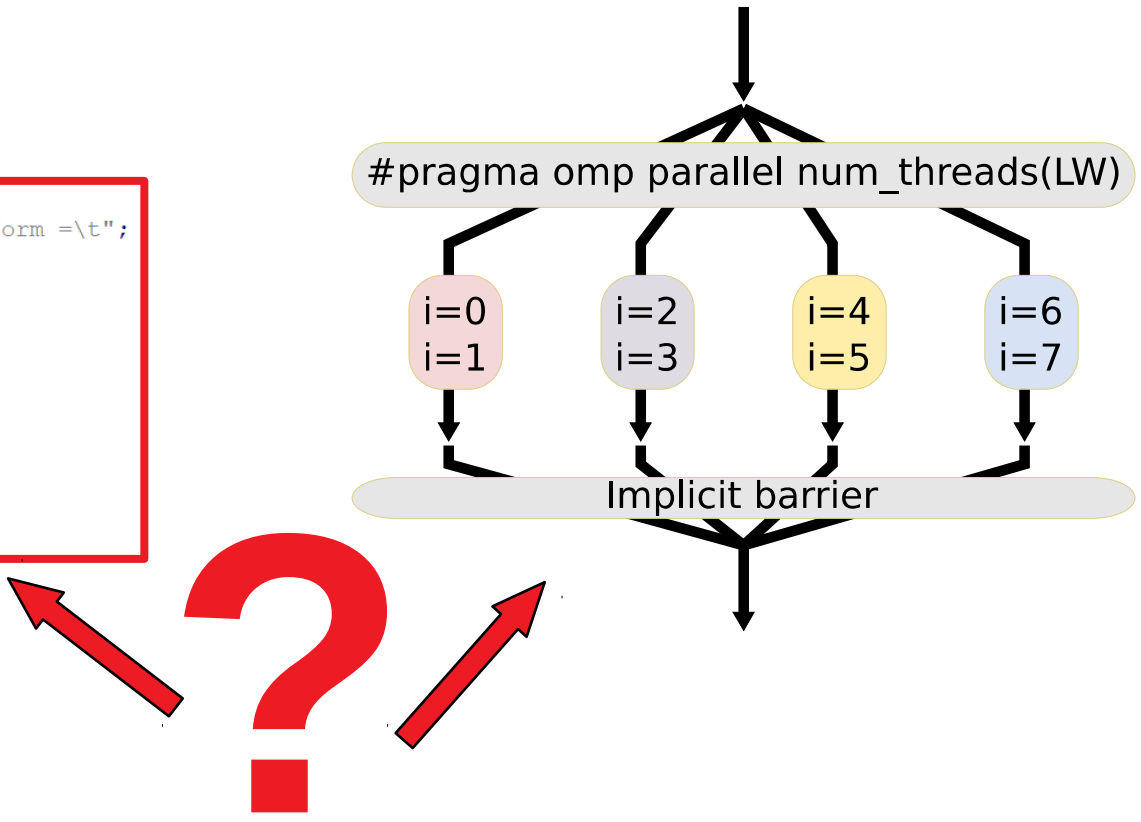
Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```



Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 8;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

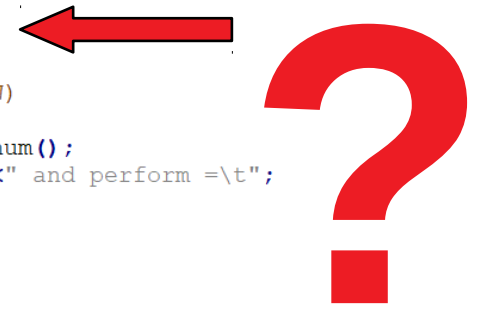


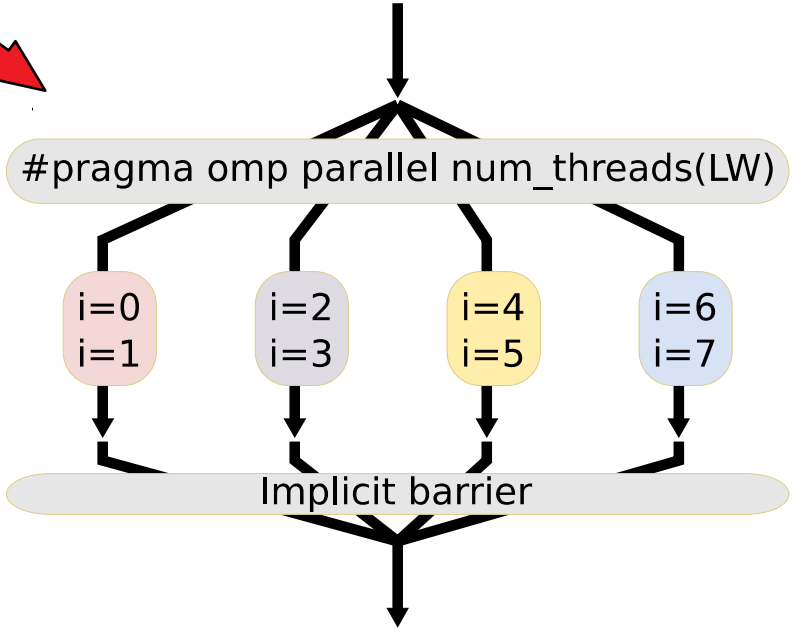
Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```



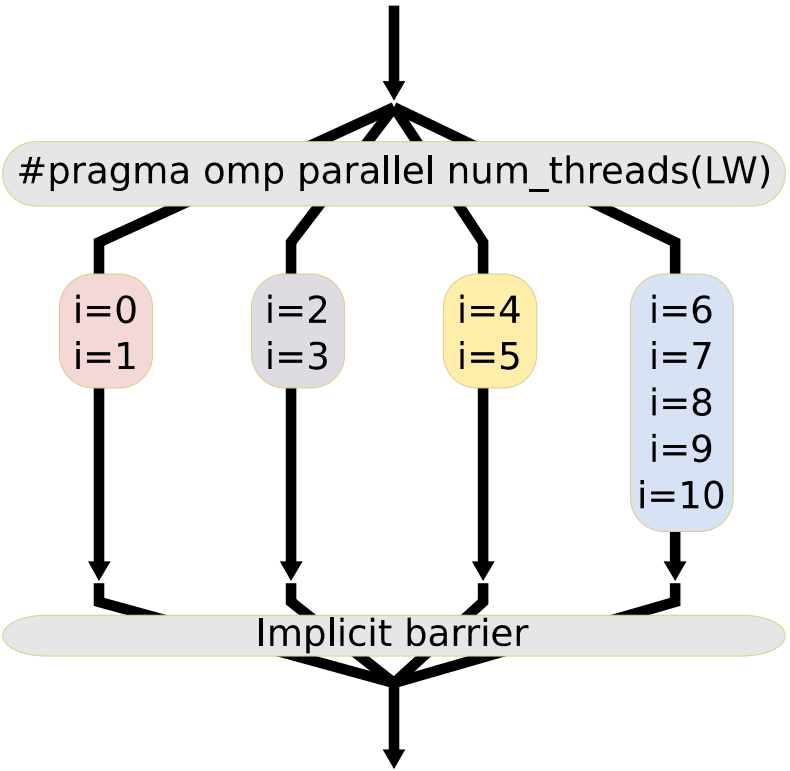


Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        if(id==LW-1) {
            end = n;
            range = end-start;
        }

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```cpp
#define LW 4

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;

        int range = quotient;
        int start = id*range;
        int end = start+range;

        if(id==LW-1) {
            end = n;
            range = end-start;
        }

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```



Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

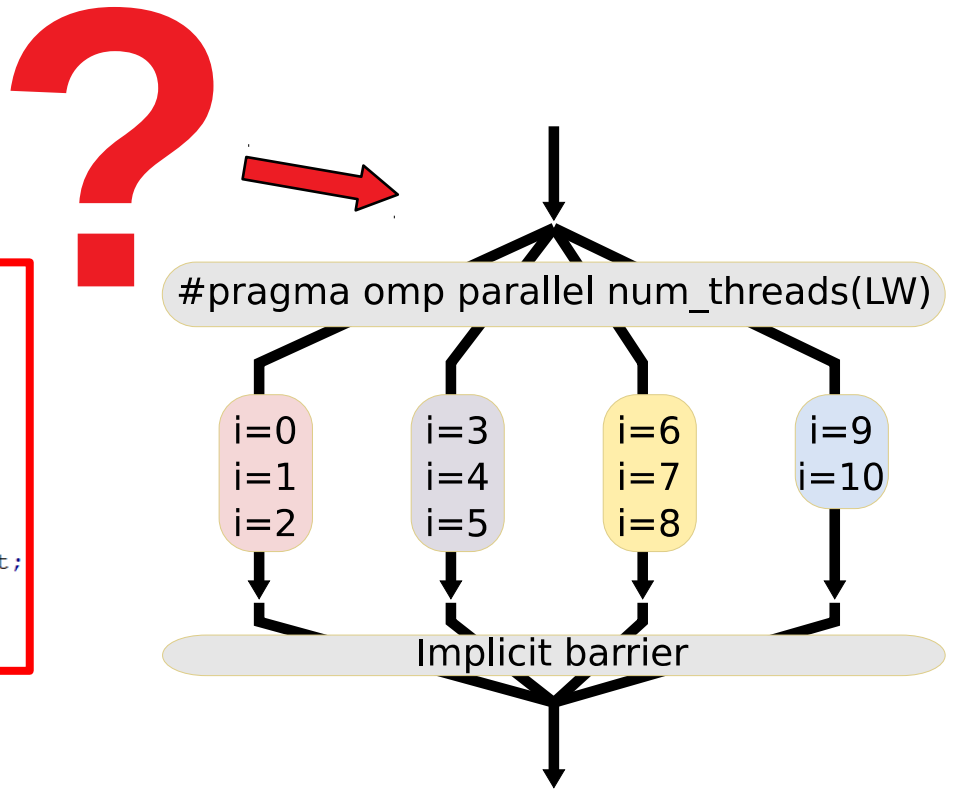
```
#define LW 4

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;
        int remainder = n%LW;

        int start;
        int range;

        if(id<remainder) {
            start = id*(quotient+1);
            range = quotient+1;
        } else {
            start = remainder * (quotient+1) + (id-remainder)*quotient;
            range = quotient;
        }
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;
        int remainder = n%LW;

        int start;
        int range;

        if(id<remainder) {
            start = id*(quotient+1);
            range = quotient+1;
        } else {
            start = remainder * (quotient+1) + (id-remainder)*quotient;
            range = quotient;
        }
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

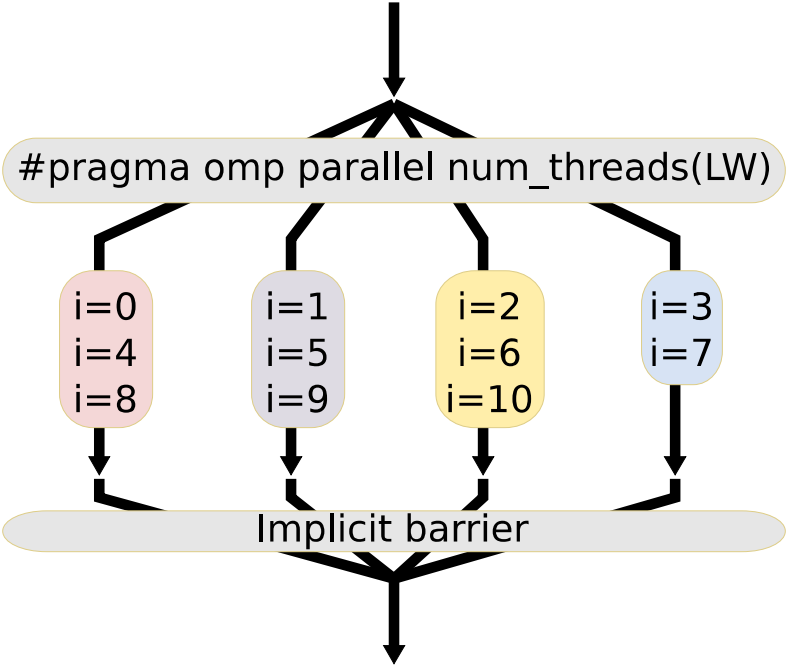
```cpp
#define LW 4

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        cout<<"I am thread no = "<<id<<" and perform =\t";

        int quotient = n/LW;
        int remainder = n%LW;

        int start;
        int range;

        if(id<remainder) {
            start = id*(quotient+1);
            range = quotient+1;
        } else {
            start = remainder * (quotient+1) + (id-remainder)*quotient;
            range = quotient;
        }
        int end = start+range;

        for(int i=start; i<end; ++i) {
            cout<<i<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```



Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

- Zadanie 1

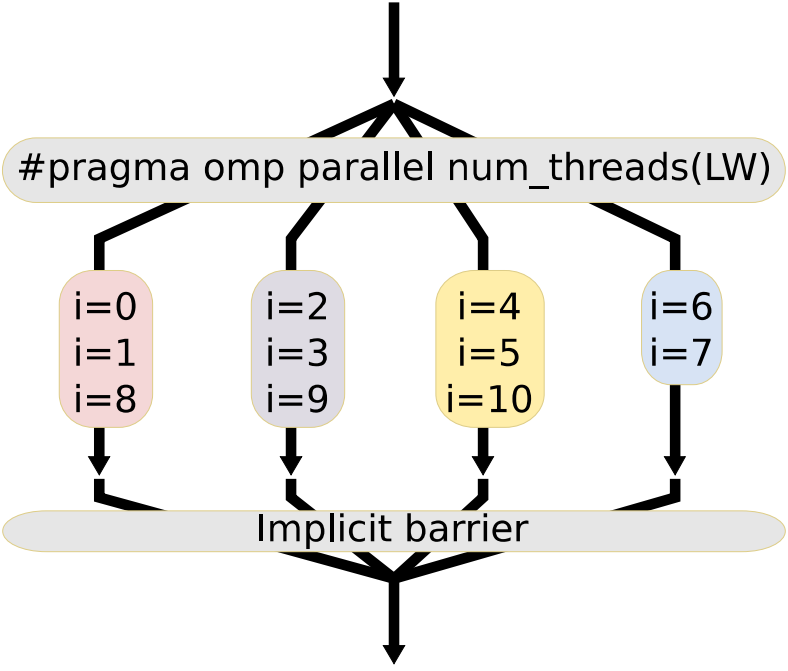
  Napisz program aby umożliwił wykonanie dystrybucji pętli pomiędzy dowolną liczbę wątków zgodnie z rysunkiem dla dowolnej liczby iteracji pętli



Manualne metody dystrybucji obliczeń realizowanych w ramach pętli

- Zadanie 2

  Napisz program aby umożliwił wykonanie dystrybucji pętli pomiędzy dowolną liczbę wątków zgodnie z rysunkiem dla dowolnej liczby iteracji pętli



Plan prezentacji

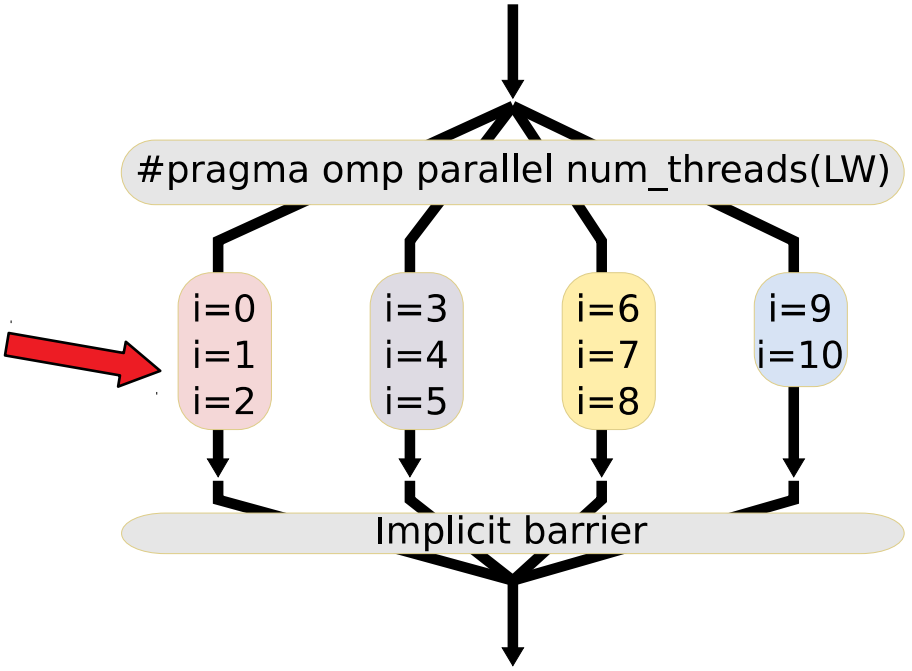
- Manualne metody dystrybucji obliczeń realizowanych w ramach pętli
- **Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli**
- Mechanizmy równoważenia obciążenia
- Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        int quotient = n/LW;
        int remainder = n%LW;

        int start;
        int range;

        if(id<remainder) {
            start = id*(quotient+1);
            range = quotient+1;
        } else {
            start = remainder * (quotient+1) + (id-remainder)*quotient;
            range = quotient;
        }
        int end = start+range;

        for(int i=start; i<end; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

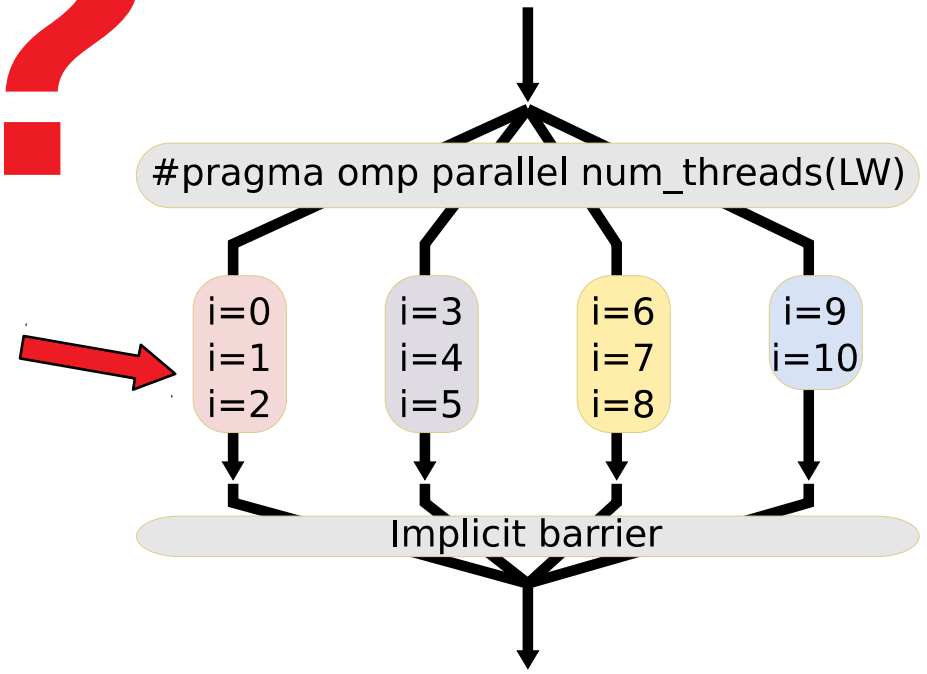
```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        int quotient = n/LW;
        int remainder = n%LW;

        int start;
        int range;

        if(id<remainder) {
            start = id*(quotient+1);
            range = quotient+1;
        } else {
            start = remainder * (quotient+1) + (id-remainder)*quotient;
            range = quotient;
        }
        int end = start+range;

        for(int i=start; i<end; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```



Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        int quotient = n/LW;
        int remainder = n%LW;

        int start;
        int range;

        if(id<remainder) {
            start = id*(quotient+1);
            range = quotient+1;
        } else {
            start = remainder * (quotient+1) + (id-remainder)*quotient;
            range = quotient;
        }
        int end = start+range;

        for(int i=start; i<end; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```

?



Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        int quotient = n/LW;
        int remainder = n%LW;

        int start;
        int range;

        if(id<remainder) {
            start = id*(quotient+1);
            range = quotient+1;
        } else {
            start = remainder * (quotient+1) + (id-remainder)*quotient;
            range = quotient;
        }
        int end = start+range;

        for(int i=start; i<end; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```

```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        #pragma omp for
        for(int i=0; i<n; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```

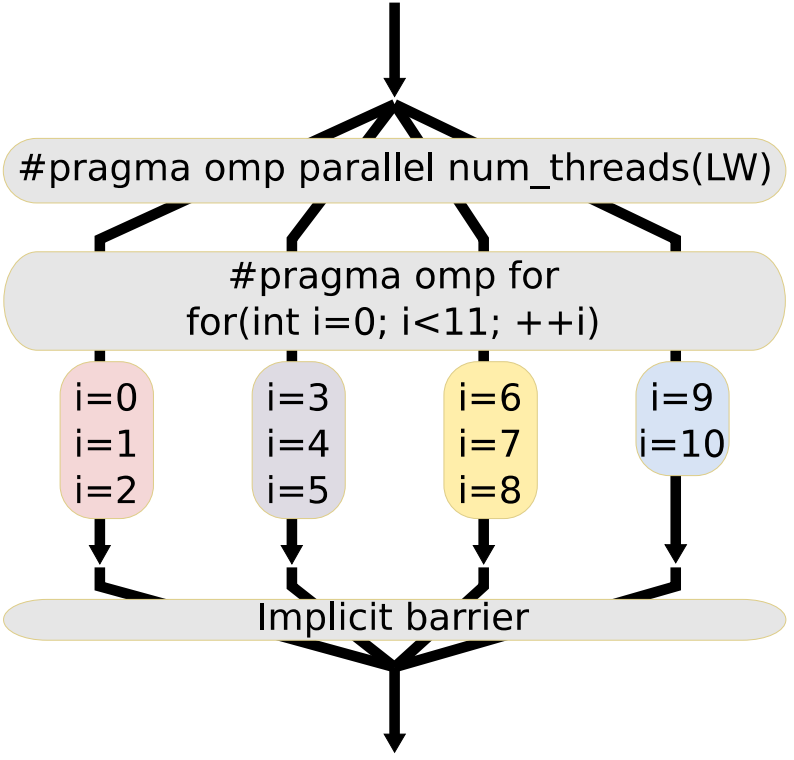
Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        int quotient = n/LW;
        int remainder = n%LW;

        int start;
        int range;

        if(id<remainder) {
            start = id*(quotient+1);
            range = quotient+1;
        } else {
            start = remainder * (quotient+1) + (id-remainder)*quotient;
            range = quotient;
        }
        int end = start+range;

        for(int i=start; i<end; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```

```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        #pragma omp for
        for(int i=0; i<n; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```

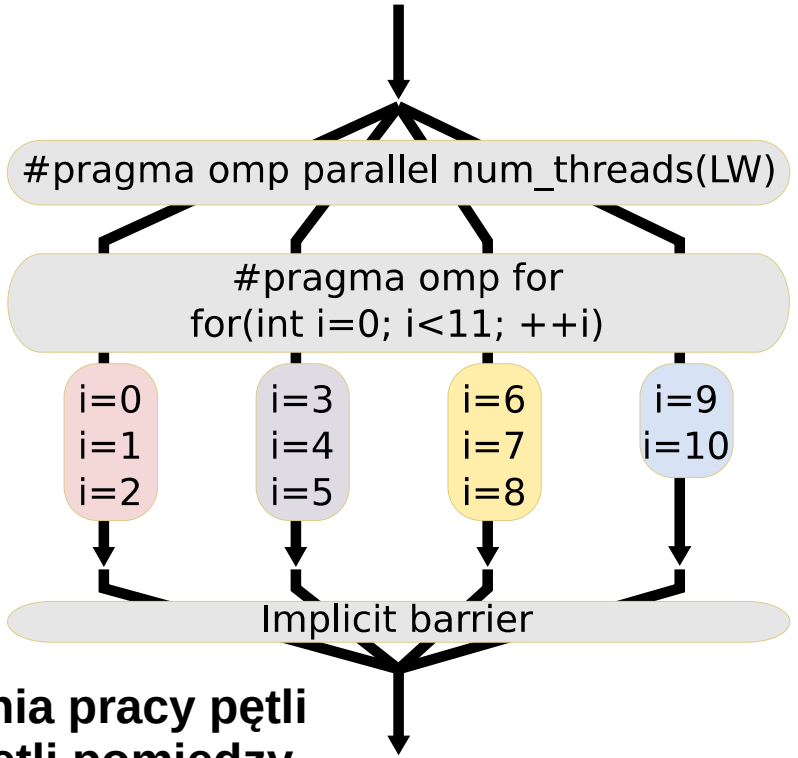
Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        #pragma omp for
        for(int i=0; i<n; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```



Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        #pragma omp for
        for(int i=0; i<n; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```



Powyższa konstrukcja dzielenia pracy pętli pozwala na podział iteracji pętli pomiędzy wątki w danym zespole

Dyrektywy podziału pracy #pragma omp for opcje i parametry

- OpenMP posiada mechanizmy umożliwiające ”automatyczną” dystrybucję obliczeń pomiędzy wątkami
- *#pragma omp for* informuje, że iteracje w danym bloku muszą być wykonane równolegle przez grupę wątków
- Konstrukcja podziału pracy (ang. work sharing construction) dzielą napotkane fragmenty kodu pomiędzy wątki grupy które napotkały tę konstrukcję
- W rezultacie wątki dzielą się pracą, a każdy wątek realizuje przydzielone sobie operacje na przydzielonej sobie części danych

https://www.openmp.org/spec-html/5.0/openmpsu40.html#x63-1260002.9.1

Dyrektywy podziału pracy #pragma omp for opcje i parametry

- Konstrukcji OpenMP dla automatycznego zrównoleglania pętli ma następującą postać:

  #pragma omp for [klauzula, [klauzula],...]

- Lista dostępnych klauzul:

  - schedule
  - private
  - firstprivate
  - lastprivate
  - ordered
  - collapse
  - nowait
  - reduction

https://www.openmp.org/spec-html/5.0/openmpsu41.html

Dyrektywy podziału pracy #pragma omp for opcje i parametry

- Konstrukcji OpenMP dla automatycznego zrównoleglania pętli ma następującą postać:

  #pragma omp for [klauzula, [klauzula],...]

- Lista dostępnych klauzul:

  - **schedule**
  - private
  - firstprivate
  - lastprivate
  - ordered
  - collapse
  - nowait
  - reduction

https://www.openmp.org/spec-html/5.0/openmpsu41.html

Dyrektywy podziału pracy #pragma omp for opcje i parametry

- Równoległe wykonanie pętli może zostać zrealizowane na wiele różny sposób
- Klauzula **schedule** daje wiele możliwość sterowania sposobem podziału poszczególnych iteracji pętli pomiędzy dostępne wątki:

  #pragma omp for schedule (kind[, chunksize])

- Wyróżniamy następujące sposoby podziału pętli:
  STATIC, **DYNAMIC**, **GUIDED**, **AUTO** or **RUNTIME**
- natomiast **chunksize** definiuje rozmiar porcji dla danego podziału

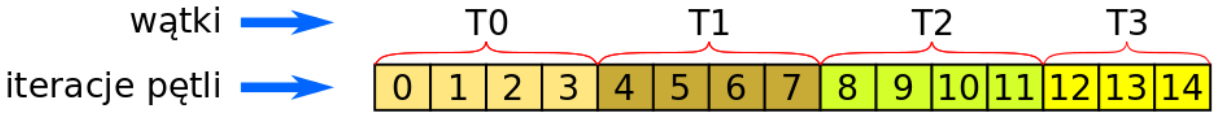
https://www.openmp.org/spec-html/5.0/openmpsu41.html

Dyrektywy podziału pracy
#pragma omp for opcje i parametry

a)

#pragma omp for schedule (static)

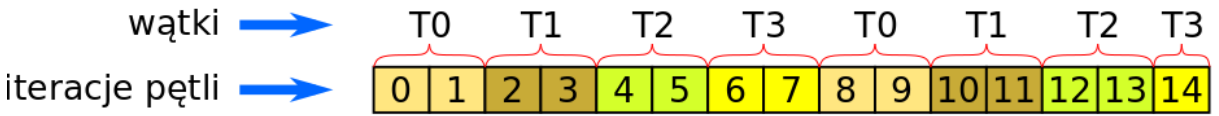
```
for(int i=0;i<15; ++i)
```



b)

#pragma omp for schedule (static, 2)

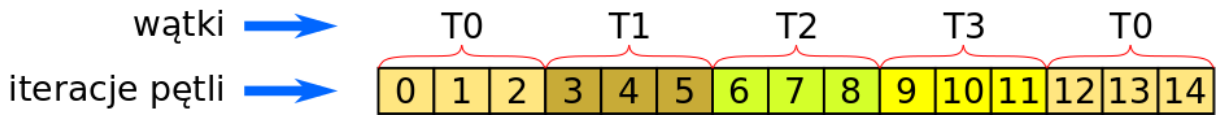
```
for(int i=0;i<15; ++i)
```



c)

#pragma omp for schedule (static, 3)

```
for(int i=0;i<15; ++i)
```



- Opcja **static**
- Iteracje są dzielone na fragmenty długości o rozmiarze chunk, a następnie statycznie przypisywane do wątków
- Gdy fragmentów jest więcej niż wątków przydział następuje według algorytmu karuzelowego (ang. round robin)

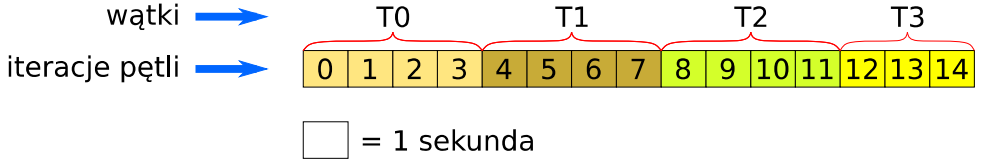
Dyrektywy podziału pracy
#pragma omp for opcje i parametry

- Czas wykonania pojedynczej iteracji jest stały i zajmuje np. 1 sekundę

a)

#pragma omp for schedule (static)

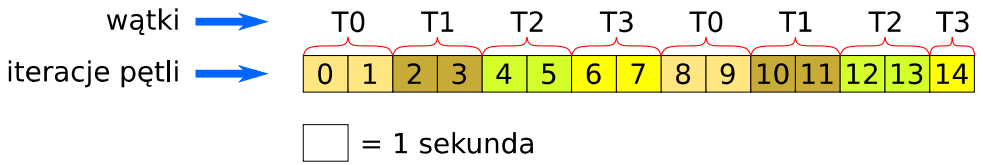
```
for(int i=0;i<15; ++i)
```



b)

#pragma omp for schedule (static, 2)

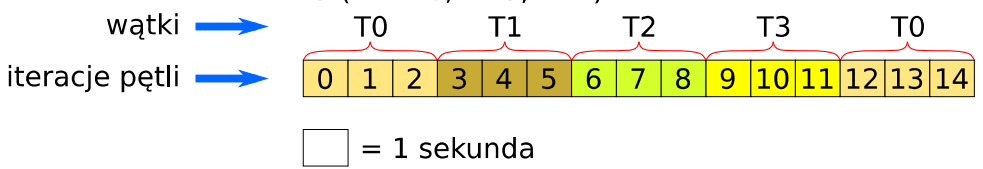
```
for(int i=0;i<15; ++i)
```



c)

#pragma omp for schedule (static, 3)

```
for(int i=0;i<15; ++i)
```



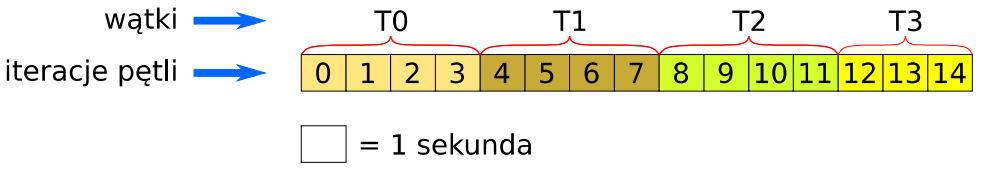
Dyrektywy podziału pracy
#pragma omp for opcje i parametry

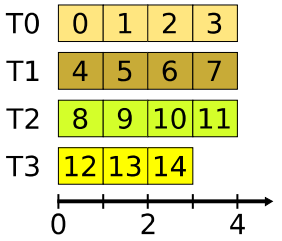
- Czas wykonania pojedynczej iteracji jest stały i zajmuje np. 1 sekundę

a)

#pragma omp for schedule (static)

```
for(int i=0;i<15; ++i)
```

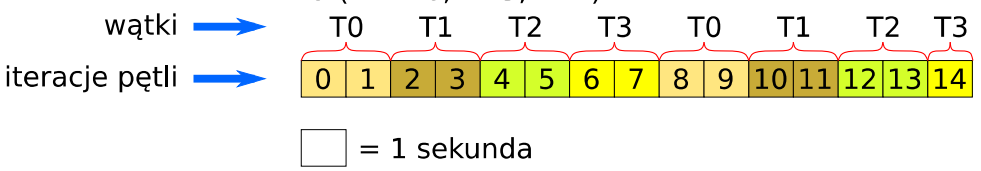


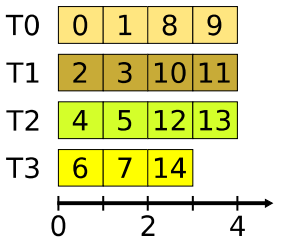


b)

#pragma omp for schedule (static, 2)

```
for(int i=0;i<15; ++i)
```

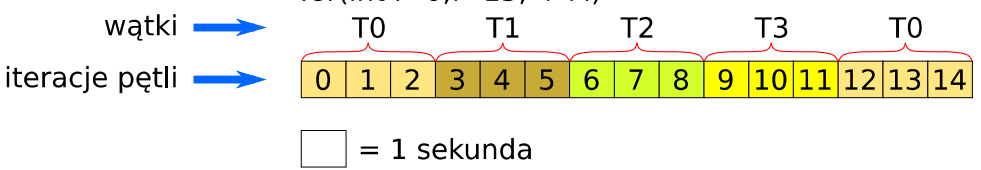


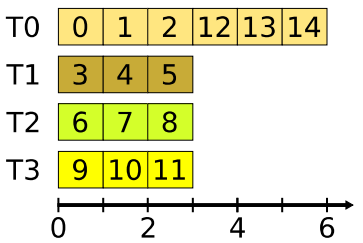


c)

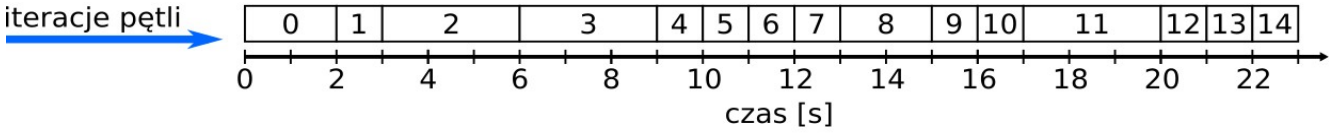
#pragma omp for schedule (static, 3)

```
for(int i=0;i<15; ++i)
```



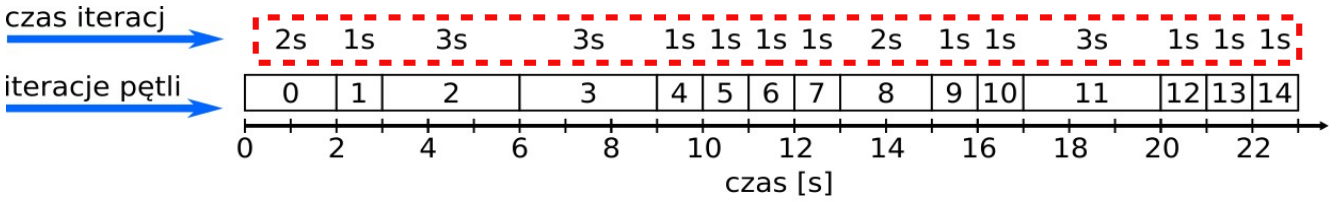


Dyrektywy podziału pracy
#pragma omp for opcje i parametry



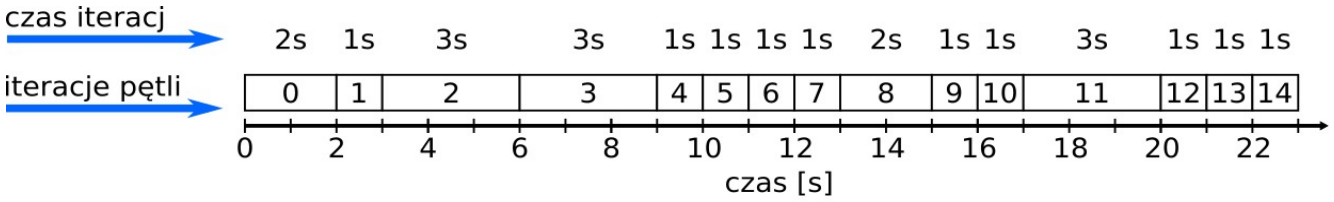
Czas wykonania poszczególnych iteracji jest różny

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



Czas wykonania poszczególnych iteracji jest różny

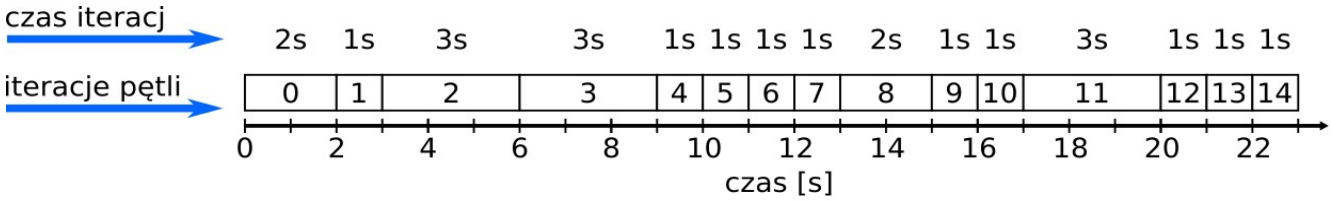
Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (static)

for(int i=0;i<15; ++i)

Dyrektywy podziału pracy
#pragma omp for opcje i parametry

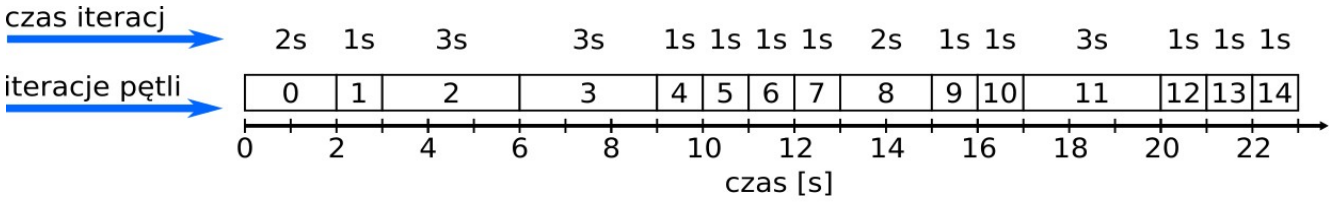


#pragma omp for schedule (static)

for(int i=0;i<15; ++i)

4 wątki OpenMP

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



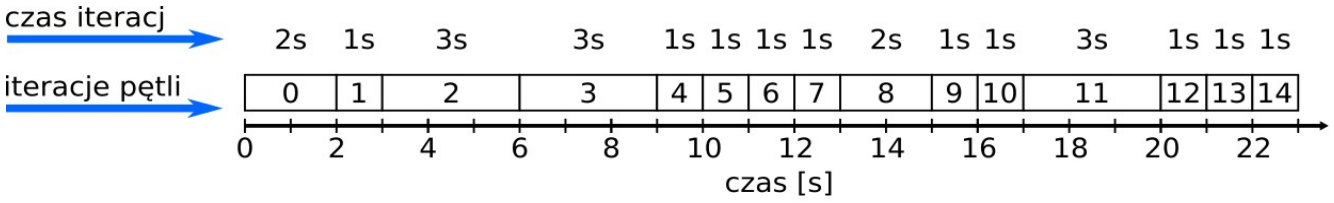
#pragma omp for schedule (static)

for(int i=0;i<15; ++i)

4 wątki OpenMP

?

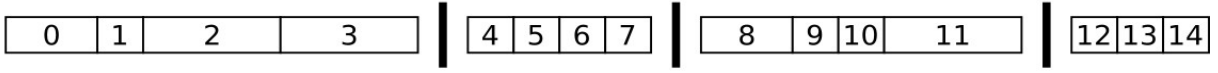
Dyrektywy podziału pracy
#pragma omp for opcje i parametry



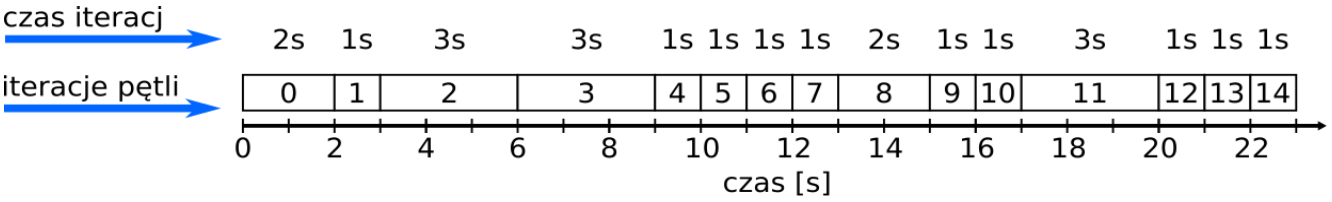
#pragma omp for schedule (static)

```
for(int i=0;i<15; ++i)
```

4 wątki OpenMP



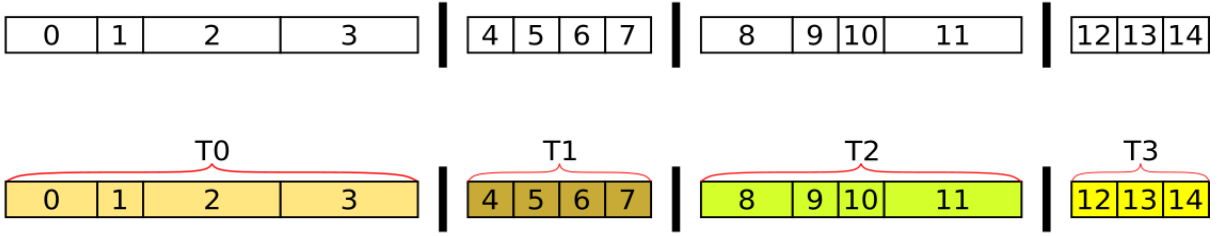
Dyrektywy podziału pracy
#pragma omp for opcje i parametry



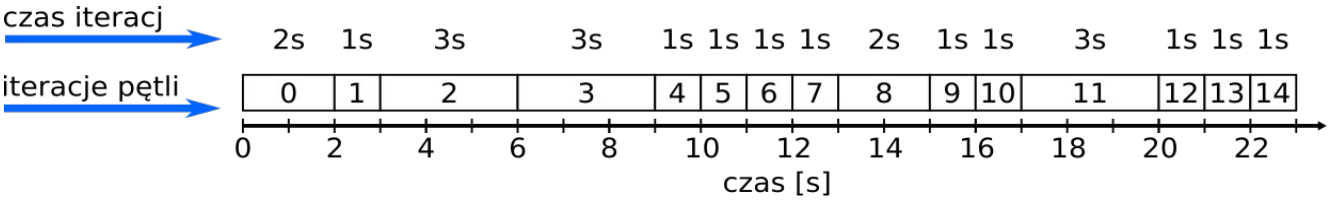
#pragma omp for schedule (static)

```
for(int i=0;i<15; ++i)
```

4 wątki OpenMP



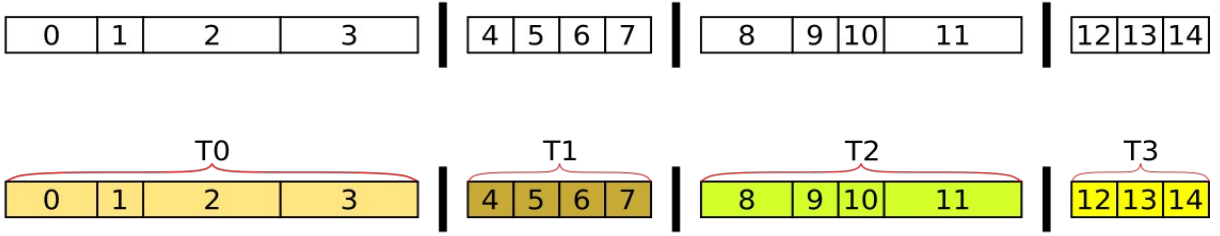
Dyrektywy podziału pracy
#pragma omp for opcje i parametry

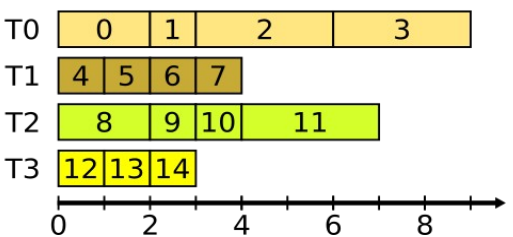


#pragma omp for schedule (static)

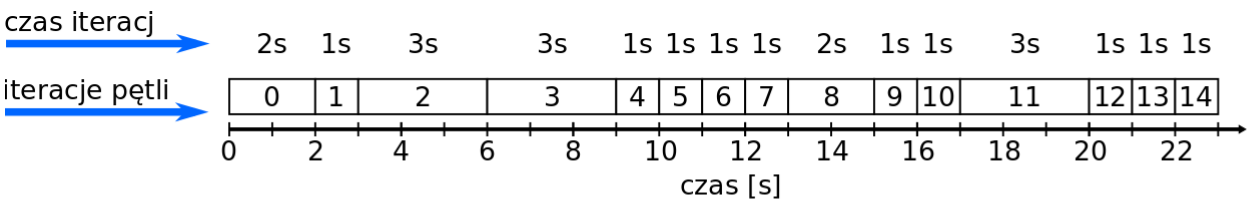
for(int i=0;i<15; ++i)

4 wątki OpenMP





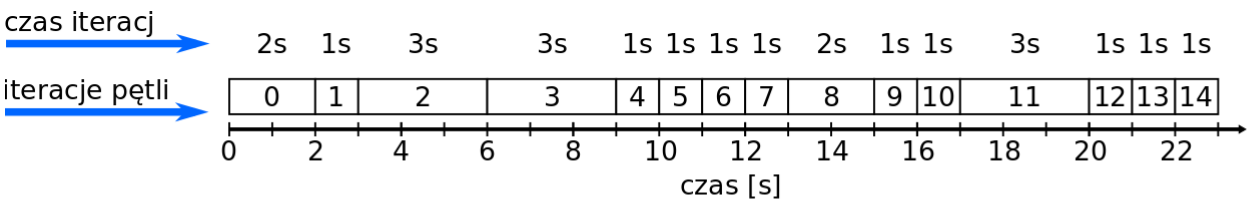
Dyrektywy podziału pracy
#pragma omp for opcje i parametry



Opcja **dynamic:**

- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry

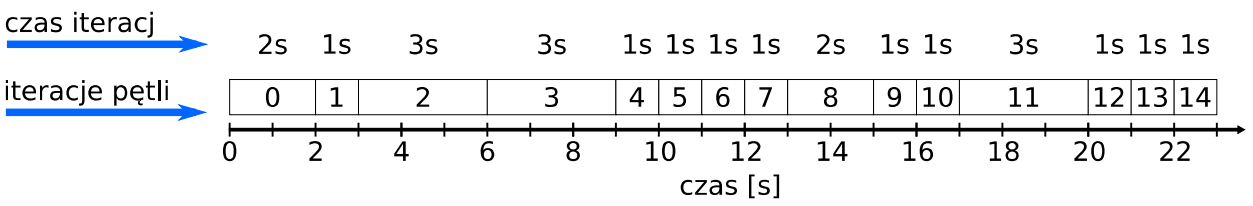



```
#pragma omp for schedule (dynamic,1)
  for(int i=0;i<15; ++i)
```

Opcja **dynamic:**

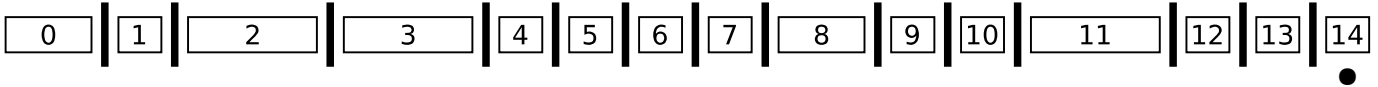
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

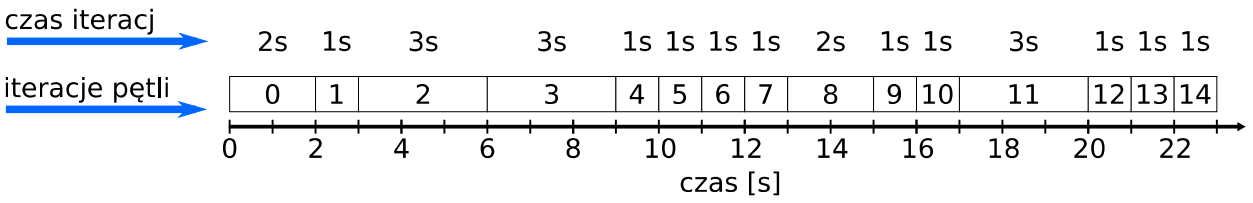
for(int i=0;i<15; ++i)



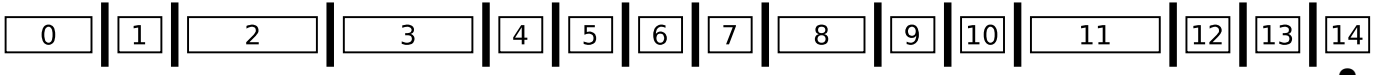
Opcja **dynamic:**

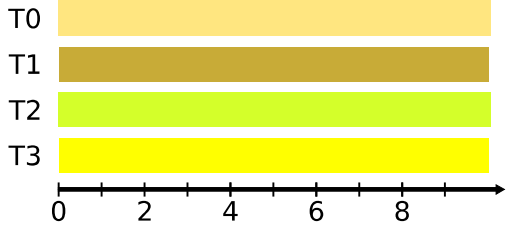
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry




```
#pragma omp for schedule (dynamic,1)
   for(int i=0;i<15; ++i)
```

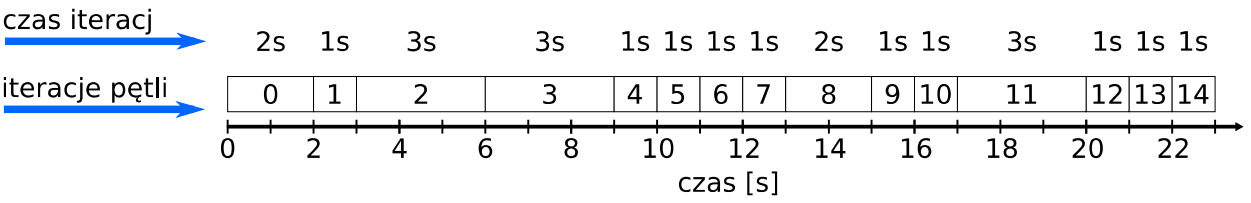




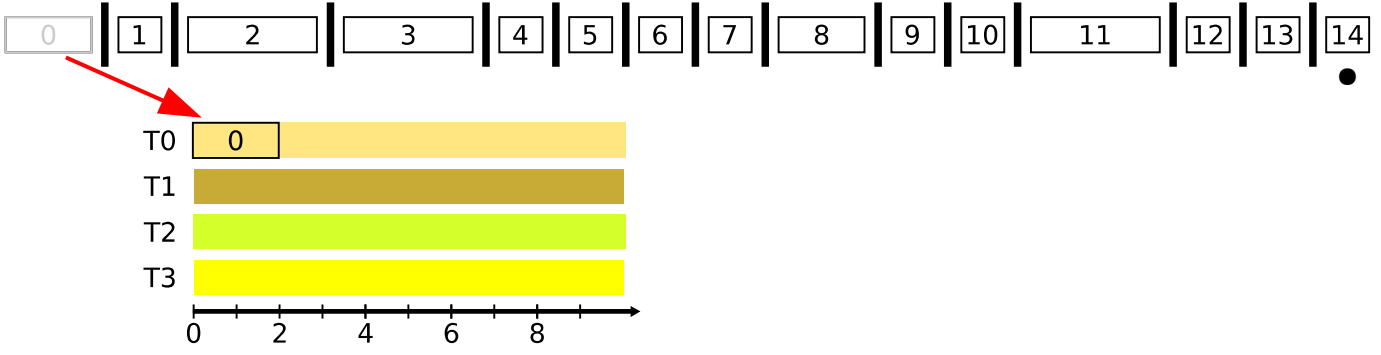
Opcja **dynamic:**

- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



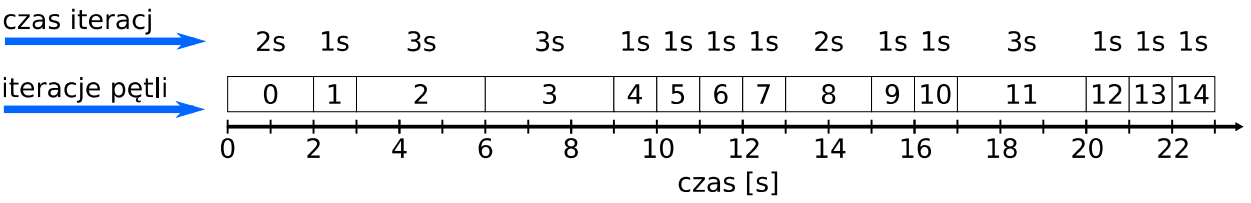

```
#pragma omp for schedule (dynamic,1)
for(int i=0;i<15; ++i)
```



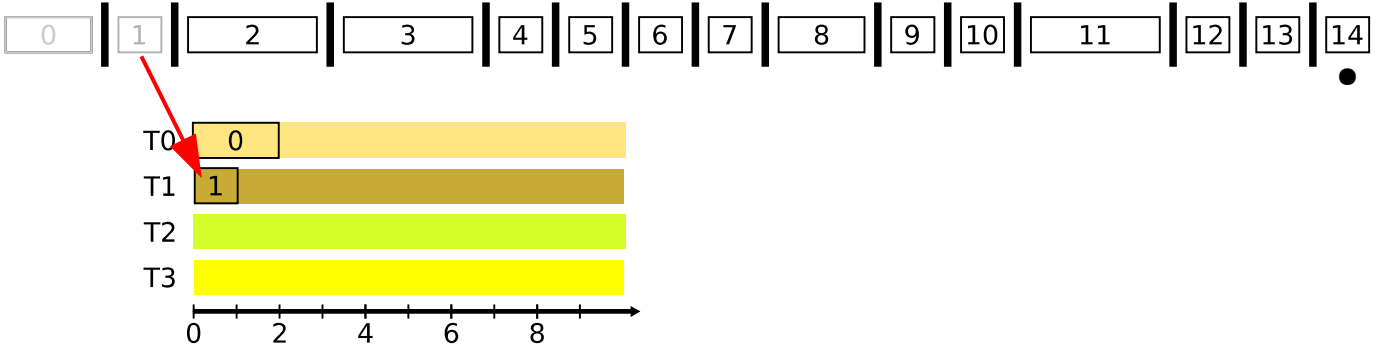
Opcja **dynamic:**

- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



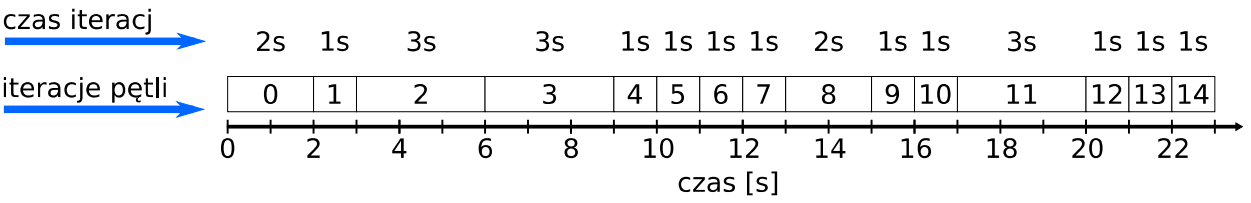

```
#pragma omp for schedule (dynamic,1)
for(int i=0;i<15; ++i)
```



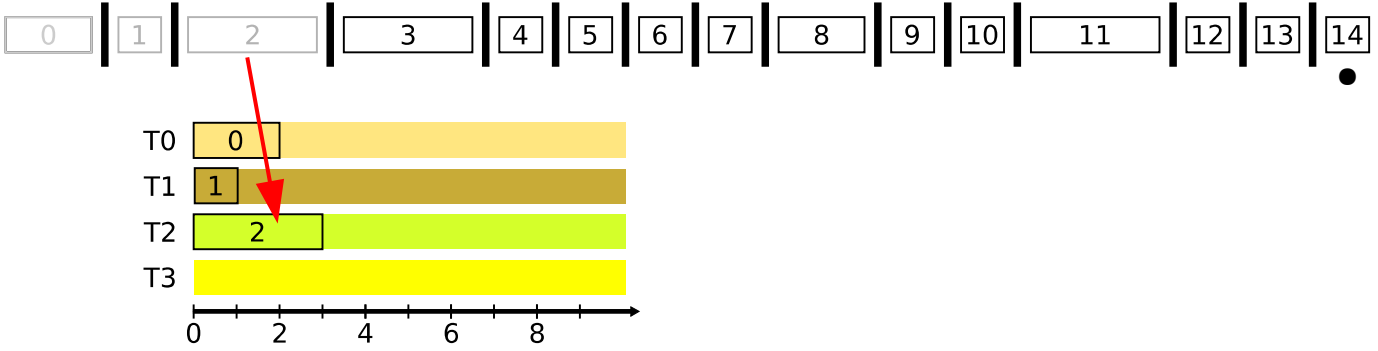
Opcja **dynamic:**

- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



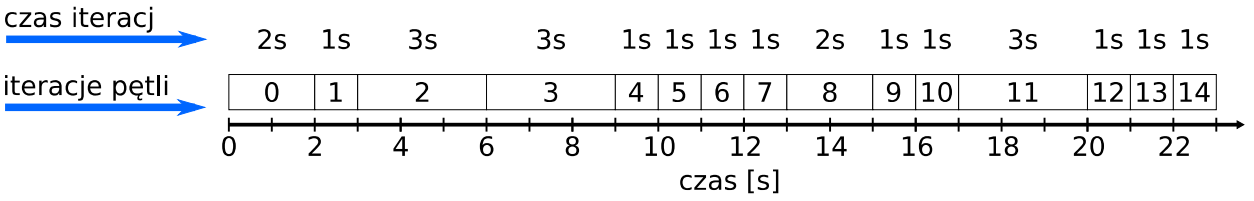

```
#pragma omp for schedule (dynamic,1)
for(int i=0;i<15; ++i)
```



Opcja **dynamic:**

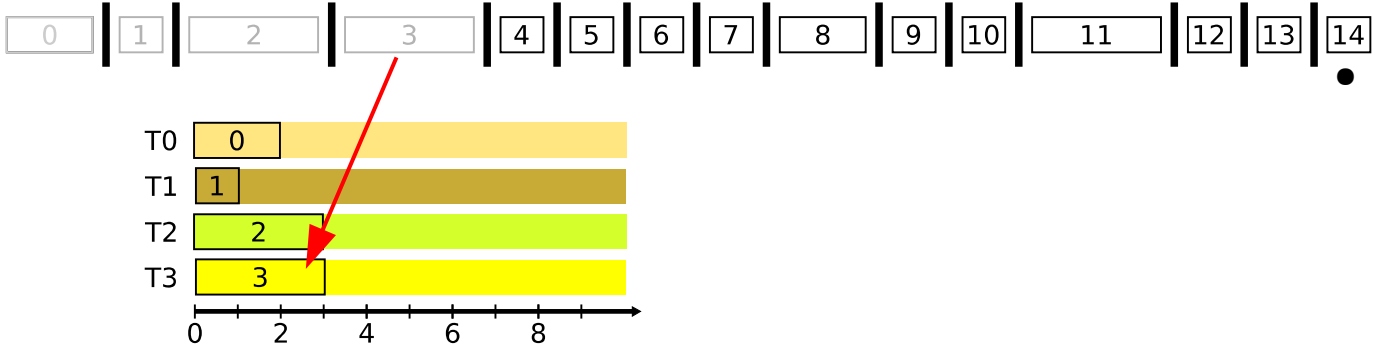
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

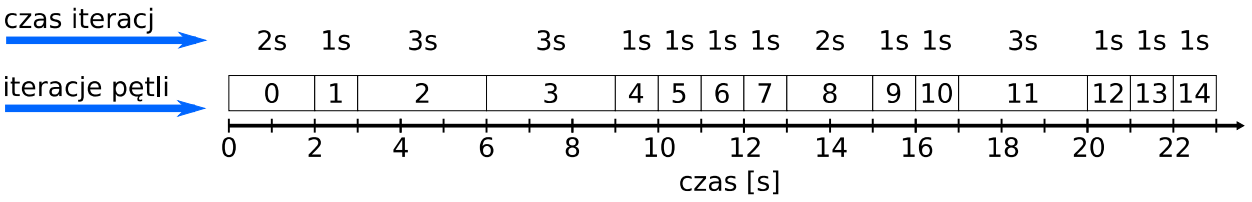
for(int i=0;i<15; ++i)



Opcja **dynamic:**

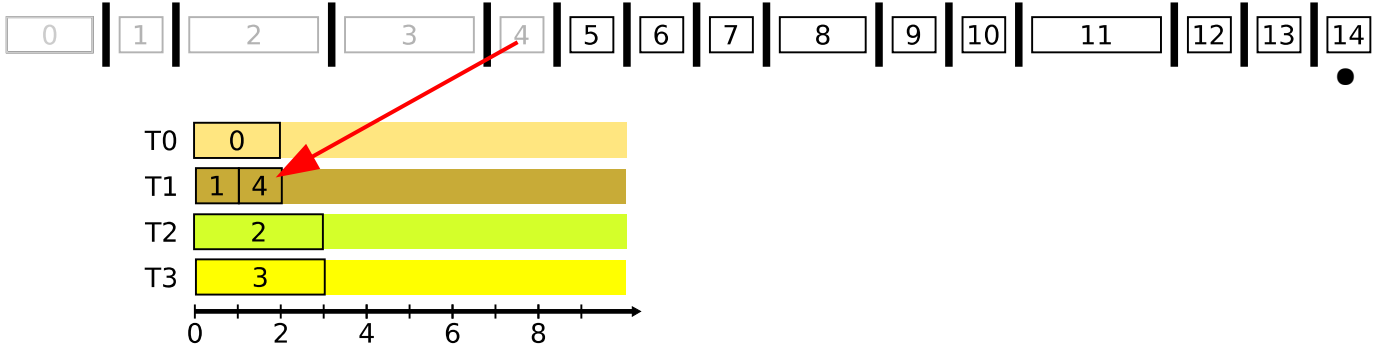
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

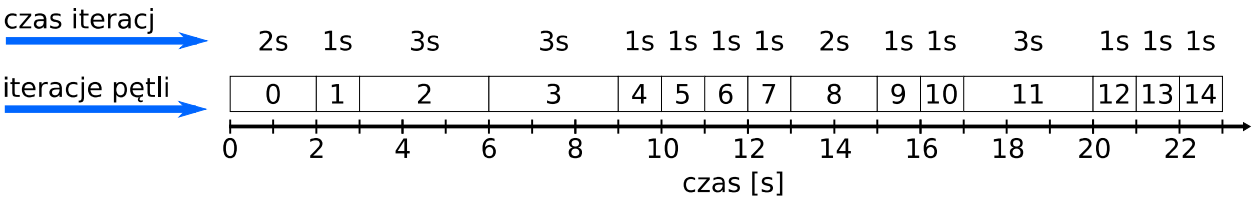
for(int i=0;i<15; ++i)



Opcja **dynamic:**

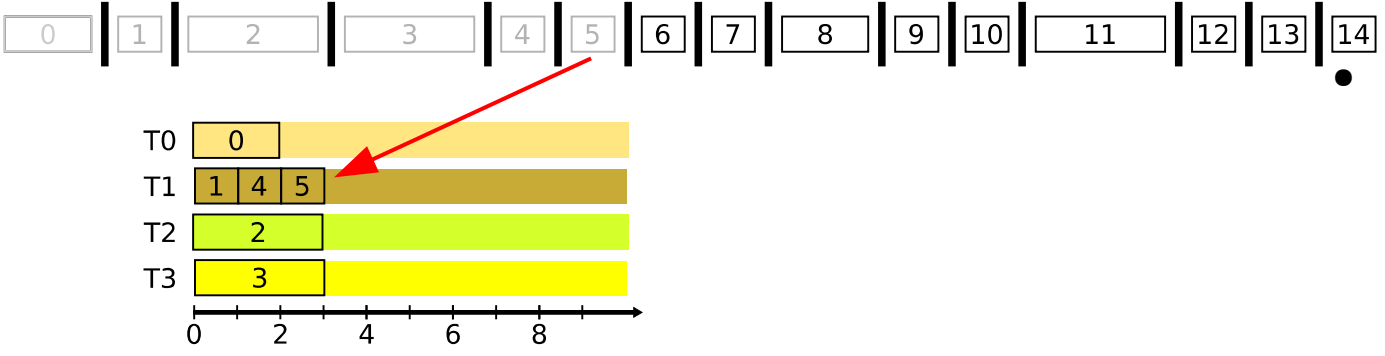
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

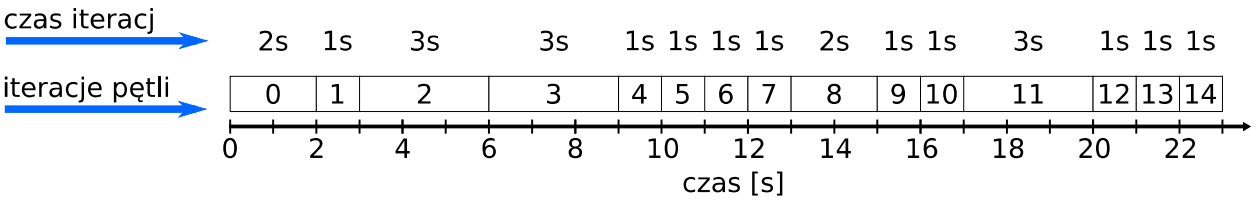
for(int i=0;i<15; ++i)



Opcja **dynamic:**

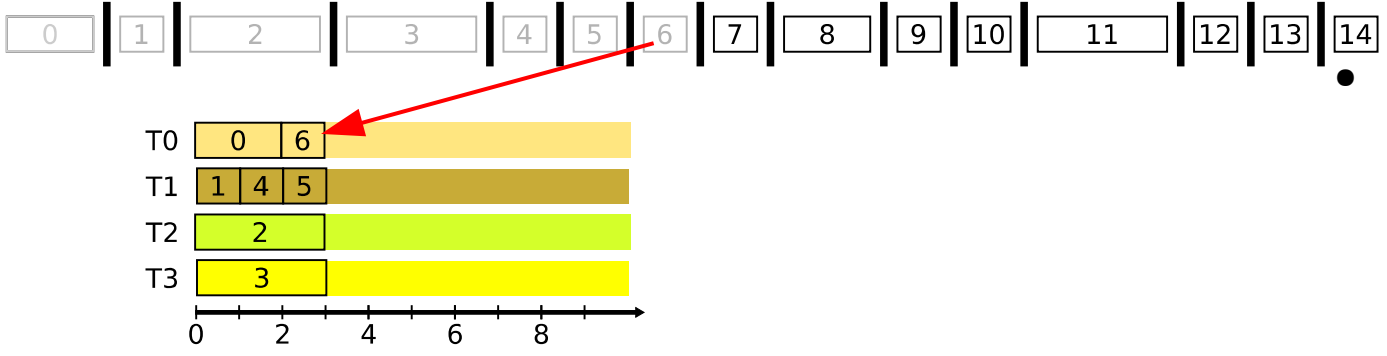
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

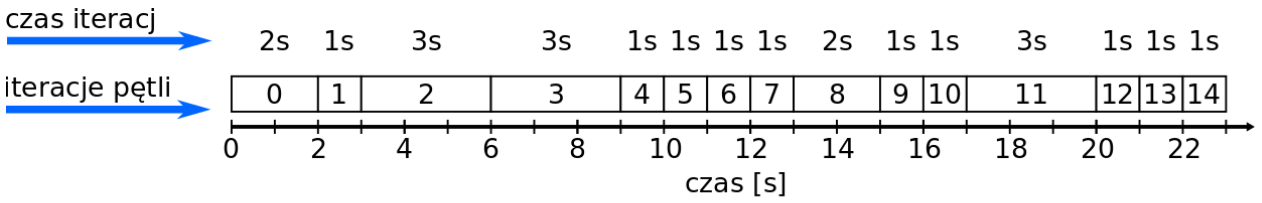
for(int i=0;i<15; ++i)



Opcja **dynamic:**

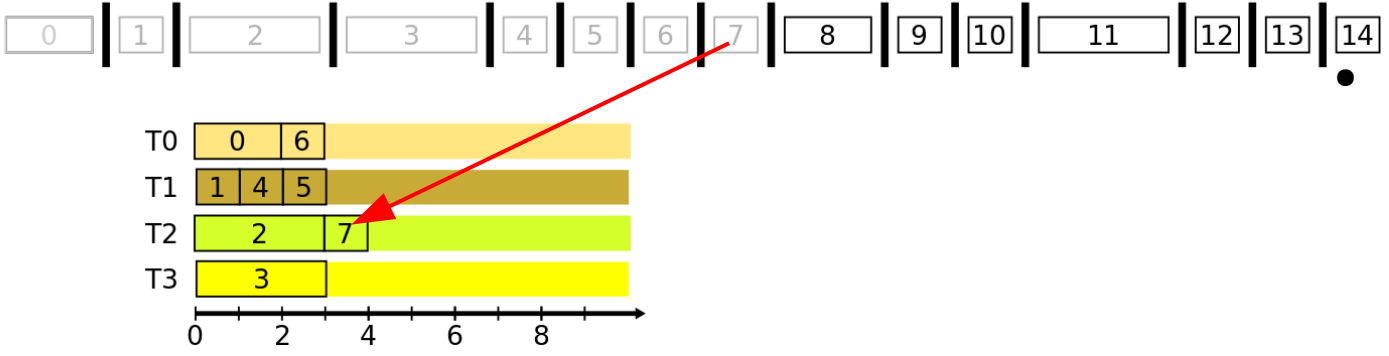
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

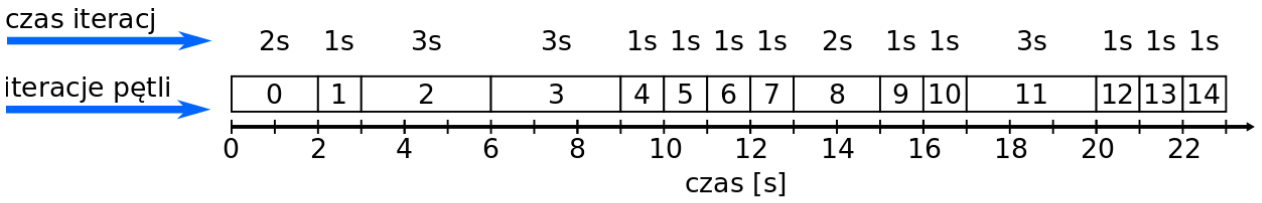
for(int i=0;i<15; ++i)



Opcja **dynamic:**

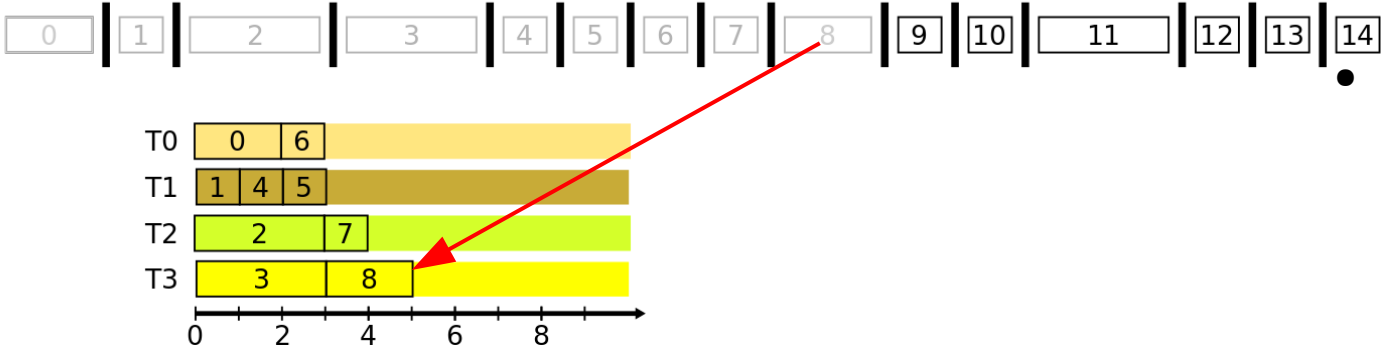
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

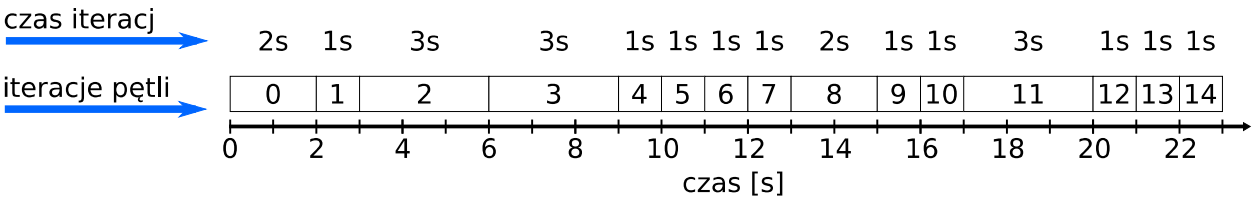
for(int i=0;i<15; ++i)



Opcja **dynamic:**

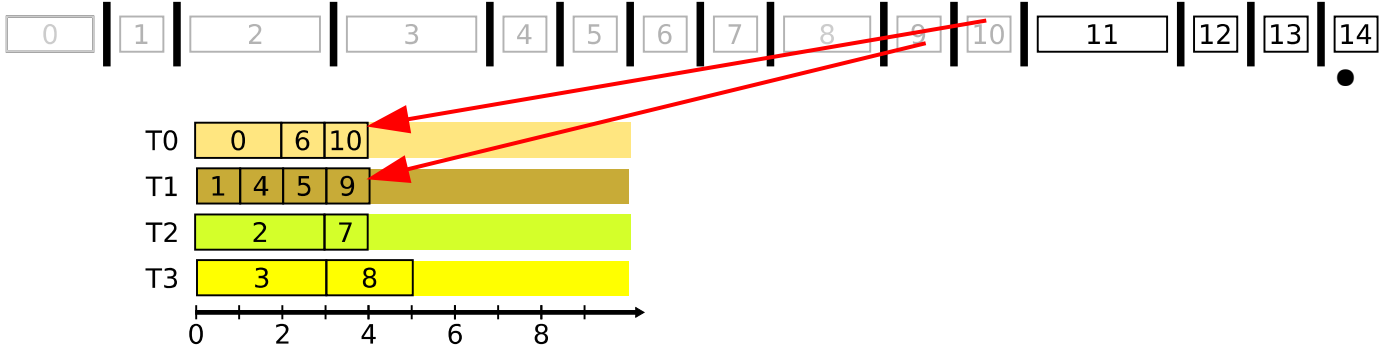
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

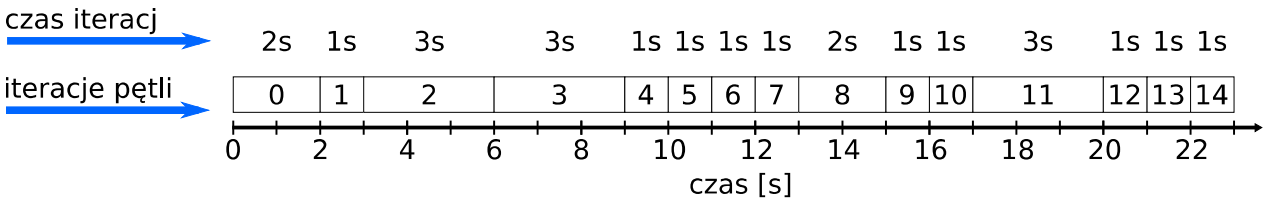
for(int i=0;i<15; ++i)



Opcja **dynamic:**

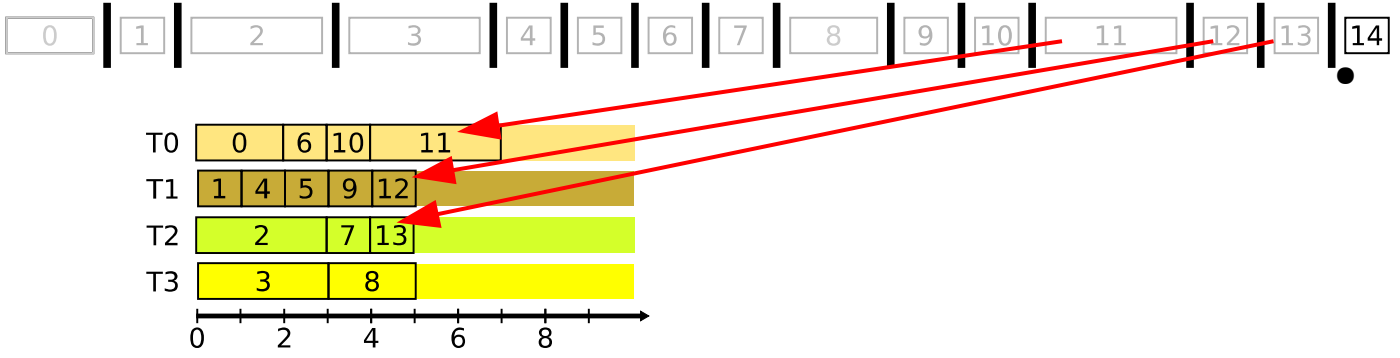
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

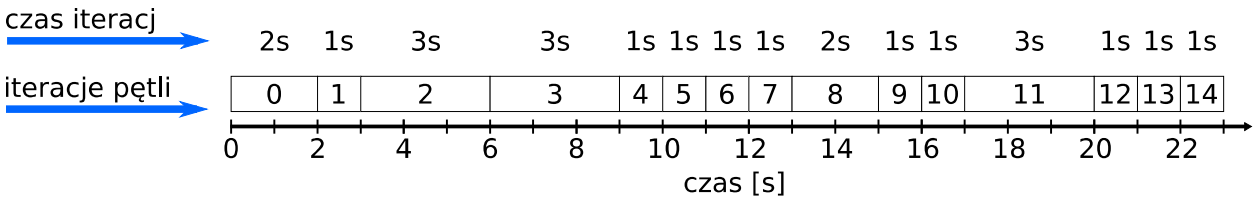
for(int i=0;i<15; ++i)



Opcja **dynamic:**

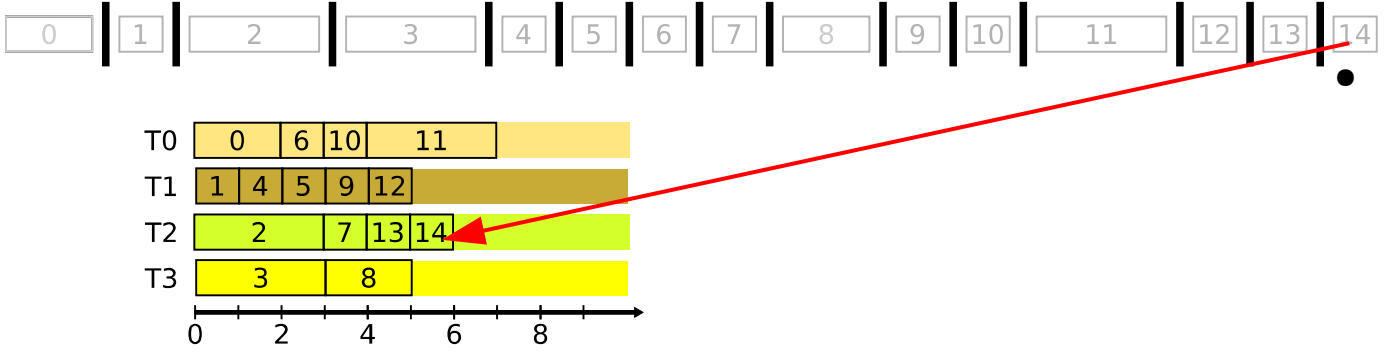
- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry



#pragma omp for schedule (dynamic,1)

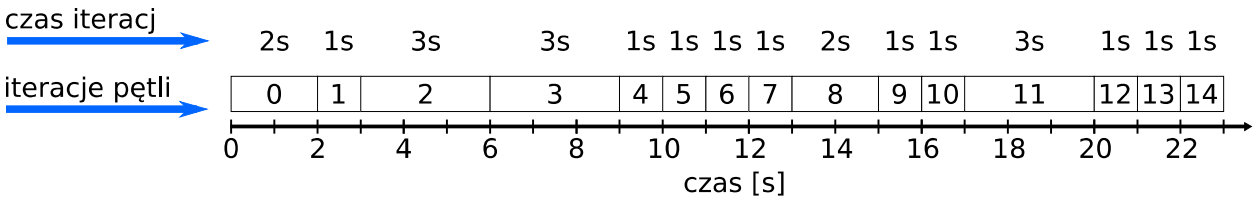
for(int i=0;i<15; ++i)



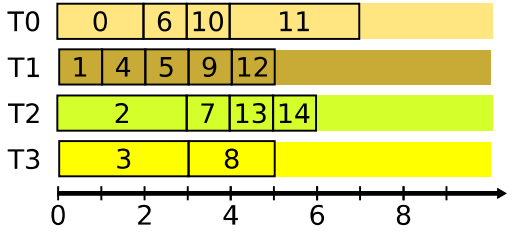
Opcja **dynamic:**

- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

Dyrektywy podziału pracy
#pragma omp for opcje i parametry




```
#pragma omp for schedule (dynamic,1)
    for(int i=0;i<15; ++i)
```

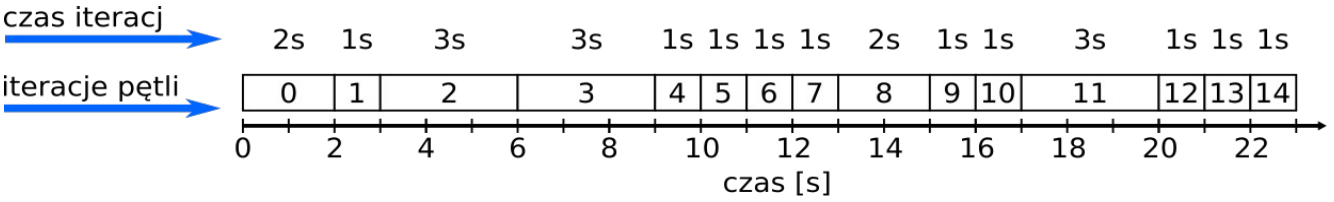


7 sekund

Opcja **dynamic:**

- Iteracje są dzielone na fragmenty długości chunk i przypisywane do wątków
- Gdy wątek wykona zadanie pobiera następny wolny fragment

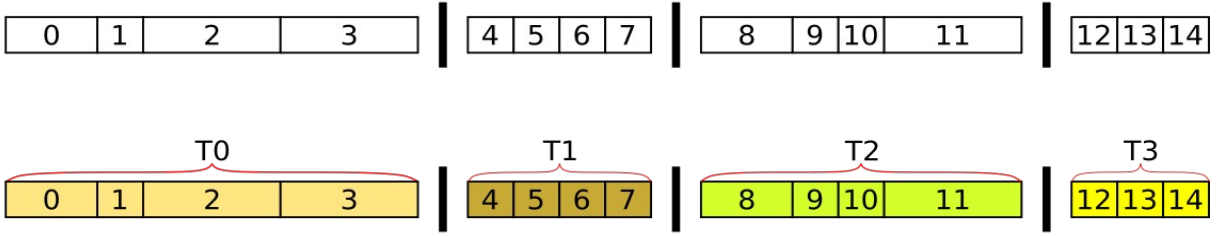
Dyrektywy podziału pracy
#pragma omp for opcje i parametry

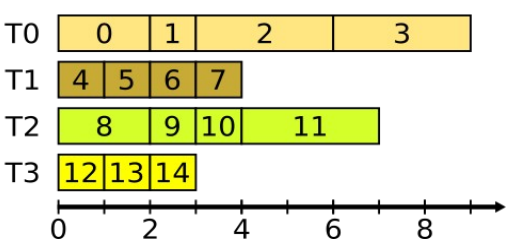


#pragma omp for schedule (static)

for(int i=0;i<15; ++i)

4 wątki OpenMP





9 sekund

Dyrektywy podziału pracy
#pragma omp for opcje i parametry

Opcja **guided:**

- Dla „chunk” równego 1, rozmiar zbioru jest opisany algorytmem:

  liczba_nieprzydzielonych_iteracji/liczba_wątków, wartość ta zmierza do 1
- Jeśli „chunk” ustalony zostaje na m gdzie m>1, to rozmiar porcji ustalany jest podobnie jak wyżej, lecz nie może on stać się mniejszy od m
- Duży fragment następujących po sobie iteracji jest przypisywany do każdego wątku dynamicznie
- Rozmiar fragmentu zmniejsza się wykładniczo, wraz z każdą pomyślną alokacją, do minimalnego rozmiaru zdefiniowanego w parametrze „chunk”.

Dyrektywy podziału pracy #pragma omp for opcje i parametry

Opcja **auto:**

- „auto” – System samodzielnie określa podział zrównoleglenia zadania

Opcja **runtime:**

- „runtime” – Ta opcja umożliwia zmianę podziału pętli w trakcie działania programu, np. poprzez zmienną środowiskową OMP_SCHEDULE

Dyrektywy podziału pracy #pragma omp for opcje i parametry

- Konstrukcji OpenMP dla automatycznego zrównoleglania pętli ma następującą postać:

  #pragma omp for [klauzula, [klauzula],...]

- Lista dostępnych klauzul:

  - schedule
  - **private**
  - **firstprivate**
  - **lastprivate**
  - ordered
  - collapse
  - nowait
  - reduction

https://www.openmp.org/spec-html/5.0/openmpsu41.html

Dyrektywy podziału pracy
#pragma omp for opcje i parametry

- **private –** dane wewnątrz równolegle przetwarzanego regionu są prywatne dla każdego wątku, co oznacza, że każdy wątek posiada lokalną kopię i używa jej jako zmienną tymczasową
- **firstprivate** – dane prywatne dla każdego wątku, inicjalizowane przy użyciu wartości zmiennych o nazwach takich jak w głównym wątku
- **lastprivate** – dane prywatne dla każdego wątku. Wartości są kopiowane na zewnątrz zrównoleglonego rejonu, do zmiennych globalnych o takiej samej nazwie, jeśli bieżąca iteracja jest ostatnią iteracją zrównoleglonej pętli.

  Zmienna może być zarówno firstprivate, jak i lastprivate.

Dyrektywy podziału pracy #pragma omp for opcje i parametry

- Konstrukcji OpenMP dla automatycznego zrównoleglania pętli ma następującą postać:

  #pragma omp for [klauzula, [klauzula],...]

- Lista dostępnych klauzul:

  - schedule
  - private
  - firstprivate
  - lastprivate
  - **ordered**
  - **collapse**
  - **nowait**
  - **reduction**

https://www.openmp.org/spec-html/5.0/openmpsu41.html

Dyrektywy podziału pracy
#pragma omp for opcje i parametry

- **ordered** – zawarty blok jest przetwarzany w porządku w którym iteracje są uruchamiane w sekwencyjnej pętli
- **collapse** – Określa, ile pętli w zagnieżdżonej pętli należy zwinąć w jedną dużą przestrzeń iteracyjną i podzielić zgodnie z klauzulą harmonogramu
- **nowait** – ustala, że wątek kompletujący przyporządkowaną mu pracę, może pracować bez czekania na inne wątki przed zakończeniem. W przypadku braku tej klauzuli, wątki przeprowadzają synchronizację bariery na końcu bloku pracy współdzielonej.

Dyrektywy podziału pracy #pragma omp for opcje i parametry

- **reduction**(operator | intrinsic: list) – zmienna posiada lokalną kopię w każdym wątku, lecz wartości lokalnych kopii są podsumowywane (redukowane) do współdzielonej globalnie zmiennej

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

Zadanie 1

- Napisz program równoległy z dowolną liczbą wątków OpenMP umożliwiający znalezienie maksymalnej liczby w jednowymiarowej tablicy przechowującej elementy typu int.
- Dodatkowo określ indeksy (maksymalnie 100 pozycji), pod którymi znajduje się maksymalna liczba oraz określ wątki, w ramach których znaleziono maksymalny element

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

Zadanie 2

- Napisz program równoległy z dowolną liczbą wątków OpenMP umożliwiający wyznaczenie sumy wszystkich elementów dwuwymiarowej tablicy
- W ramach zadania użyj różnych sposobów podzielanie pętli
- Spróbuj zrealizować zadania na dwa sposoby, z użyciem opcji reduction oraz bez niej.

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

Zadanie 3

- Napisz program równolegle wypełniający tablicę wartościami identyfikatorów poszczególnych wątków OpenMP
- Porównaj różne sposoby automatycznej dystrybucji obliczeń poprzez uruchomienie różnych kombinacji automatycznej dystrybucji pętli np.:

  g++ t4_cz2_zadanie_nr3.cpp -fopenmp

  OMP_NUM_THREADS=4 OMP_SCHEDULE=static ./a.out
- Dodatkowo „uśpij” wykonanie każdej iteracji pętli na losowy czas i ponownie porównaj efekt zastosowania różnych metod dystrybucji

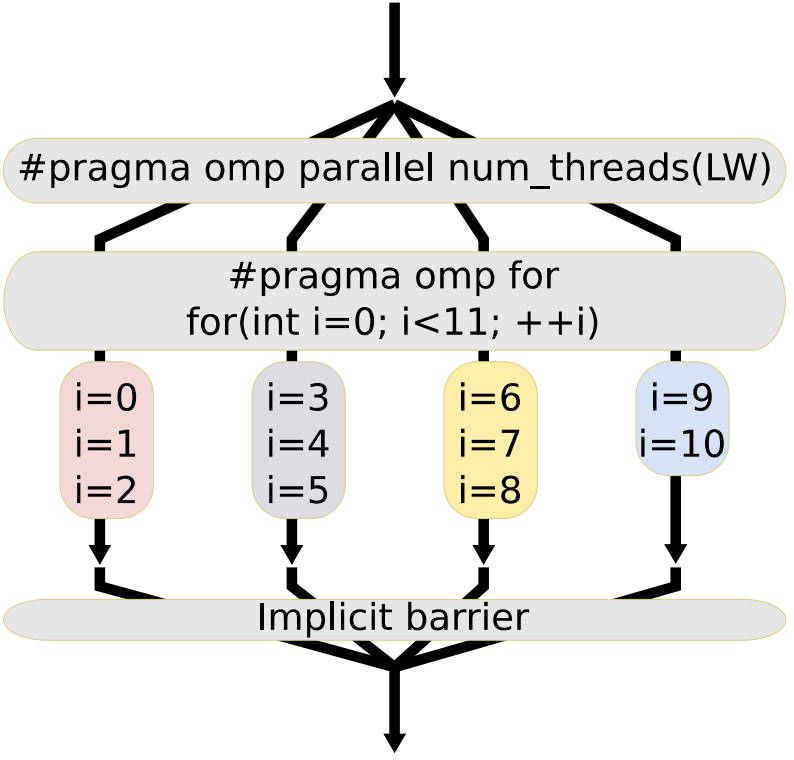
```
int main() {
    const unsigned int n = 20;
    int * tab = new int[n];

    #pragma omp parallel
    {
        srand(time(NULL));
        const int thid = omp_get_thread_num();
        ostringstream outC;
        outC<<"I am thread number "<<thid<<endl;
        cout<<outC.str();
        #pragma omp for schedule(runtime)
        for(int i=0; i<n; ++i) {
            //usleep( rand()%10000 );
            tab[i] = thid;
        }
    }

    for(int i=0; i<n; ++i) {
        cout<<tab[i]<<" ";
    }
    cout<<endl;

    delete [] tab;
    return 0;
}
```

Plan prezentacji

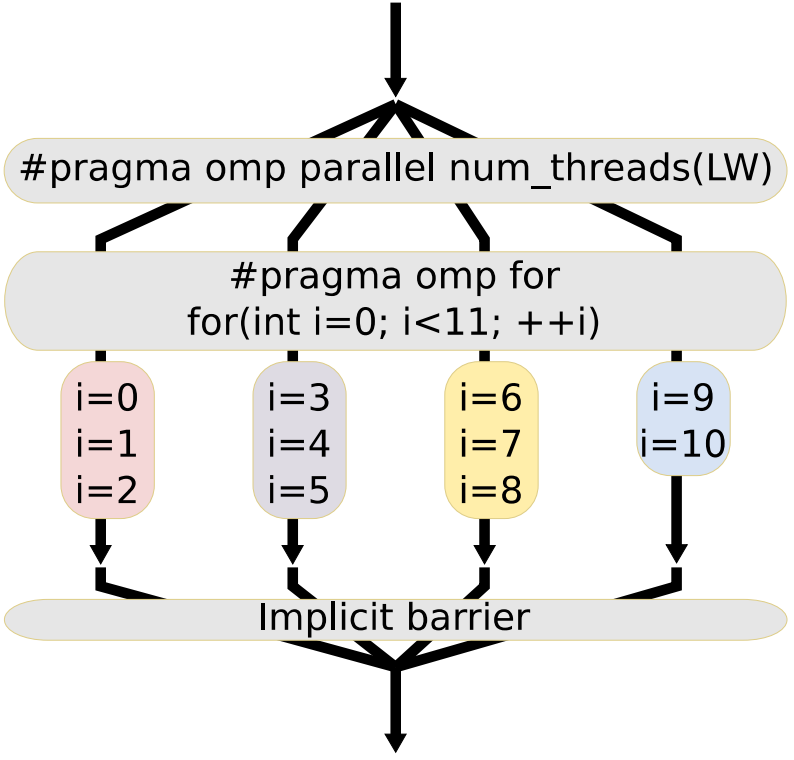
- Manualne metody dystrybucji obliczeń realizowanych w ramach pętli
- Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli
- Mechanizmy równoważenia obciążenia
- **Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe**

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

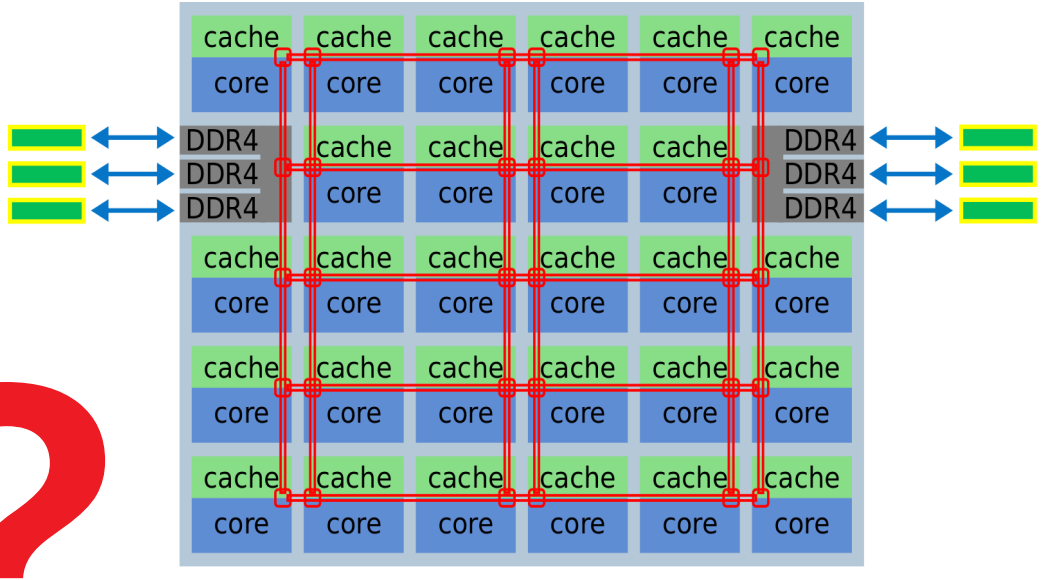
```
#define LW 4
ostringstream outC[LW];

int main() {

    const int n = 11;

    #pragma omp parallel num_threads(LW)
    {
        const int id = omp_get_thread_num();
        outC[id]<<"I am thread no = "<<id<<" and perform = ";

        #pragma omp for
        for(int i=0; i<n; ++i) {
            outC[id]<<i<<" ";
        }
        outC[id]<<endl;
    }

    for(int i=0; i<LW; ++i) {
        cout<<outC[i].str();
        outC[i].str("");
    }
```



Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe



?



Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

```
tester@gpu2:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
Address sizes:         46 bits physical, 48 bits virtual
CPU(s):                72
On-line CPU(s) list:   0-71
Thread(s) per core:    2
Core(s) per socket:    18
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz
Stepping:              2
CPU MHz:               1197.217
CPU max MHz:           3600,0000
CPU min MHz:           1200,0000
BogoMIPS:              4589.22
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              46080K
NUMA node0 CPU(s):     0-17,36-53
NUMA node1 CPU(s):     18-35,54-71
```

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

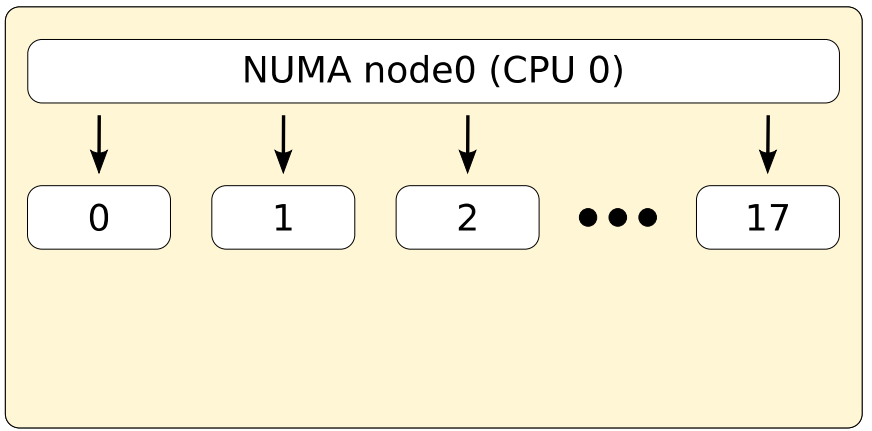
```
tester@gpu2:~$ lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
Address sizes:           46 bits physical, 48 bits virtual
CPU(s):                  72
On-line CPU(s) list:     0-71
Thread(s) per core:      2
Core(s) per socket:      18
Socket(s):               2
NUMA node(s):            2
Vendor ID:               GenuineIntel
CPU family:              6
Model:                   63
Model name:              Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz
Stepping:                2
CPU MHz:                 1197.217
CPU max MHz:             3600,0000
CPU min MHz:             1200,0000
BogoMIPS:                4589.22
Virtualization:          VT-x
L1d cache:               32K
L1i cache:               32K
L2 cache:                256K
L3 cache:                46080K
NUMA node0 CPU(s):       0-17,36-53
NUMA node1 CPU(s):       18-35,54-71
```

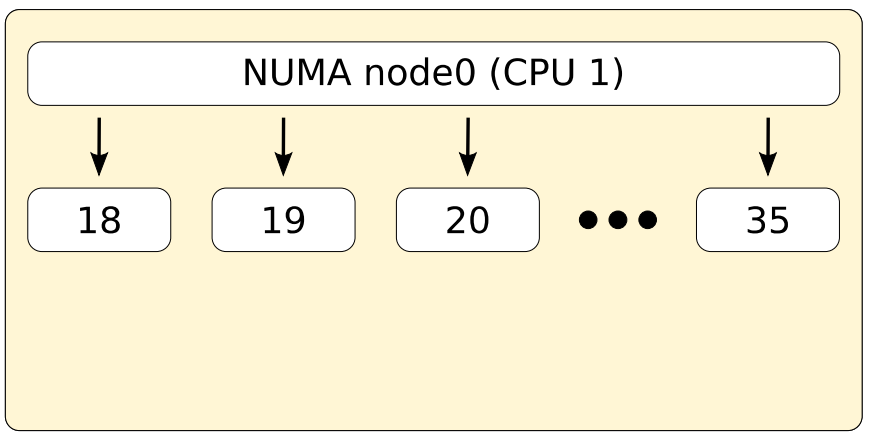
NUMA node0 (CPU 0)

NUMA node0 (CPU 1)

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

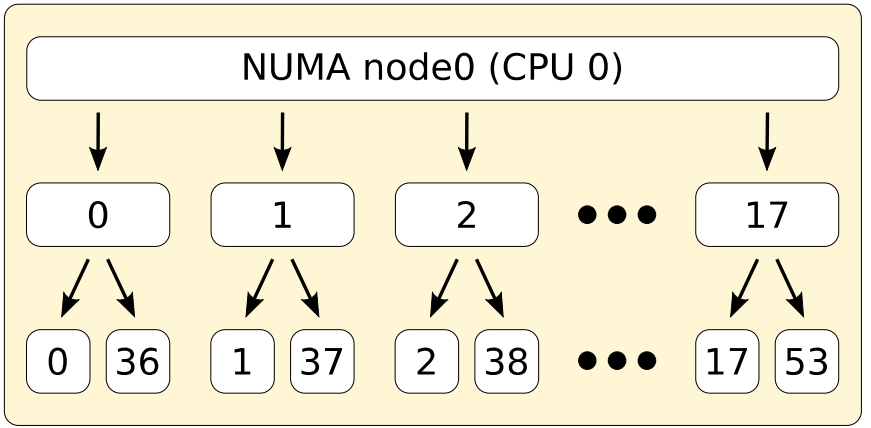
```
tester@gpu2:~$ lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
Address sizes:           46 bits physical, 48 bits virtual
CPU(s):                  72
On-line CPU(s) list:     0-71
Thread(s) per core:      2
Core(s) per socket:      18
Socket(s):               2
NUMA node(s):            2
Vendor ID:               GenuineIntel
CPU family:              6
Model:                   63
Model name:              Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz
Stepping:                2
CPU MHz:                 1197.217
CPU max MHz:             3600,0000
CPU min MHz:             1200,0000
BogoMIPS:                4589.22
Virtualization:          VT-x
L1d cache:               32K
L1i cache:               32K
L2 cache:                256K
L3 cache:                46080K
NUMA node0 CPU(s):       0-17,36-53
NUMA node1 CPU(s):       18-35,54-71
```

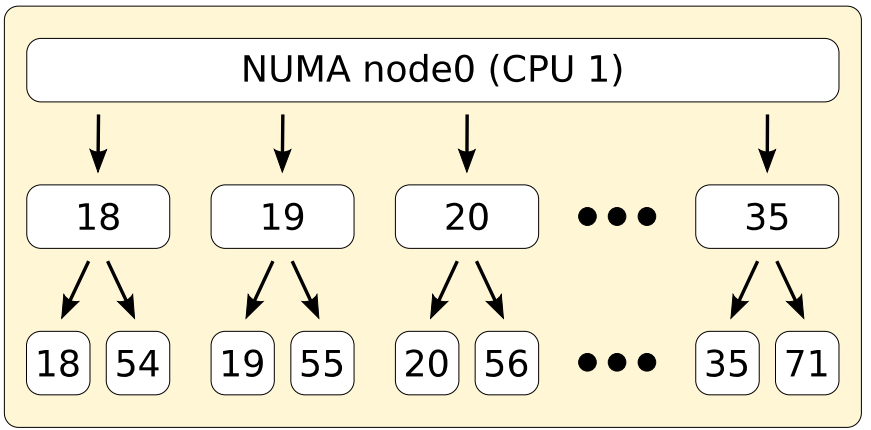




Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

```
tester@gpu2:~$ lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
Address sizes:           46 bits physical, 48 bits virtual
CPU(s):                  72
On-line CPU(s) list:     0-71
Thread(s) per core:      2
Core(s) per socket:      18
Socket(s):               2
NUMA node(s):            2
Vendor ID:               GenuineIntel
CPU family:              6
Model:                   63
Model name:              Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz
Stepping:                2
CPU MHz:                 1197.217
CPU max MHz:             3600,0000
CPU min MHz:             1200,0000
BogoMIPS:                4589.22
Virtualization:          VT-x
L1d cache:               32K
L1i cache:               32K
L2 cache:                256K
L3 cache:                46080K
NUMA node0 CPU(s):       0-17,36-53
NUMA node1 CPU(s):       18-35,54-71
```





Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

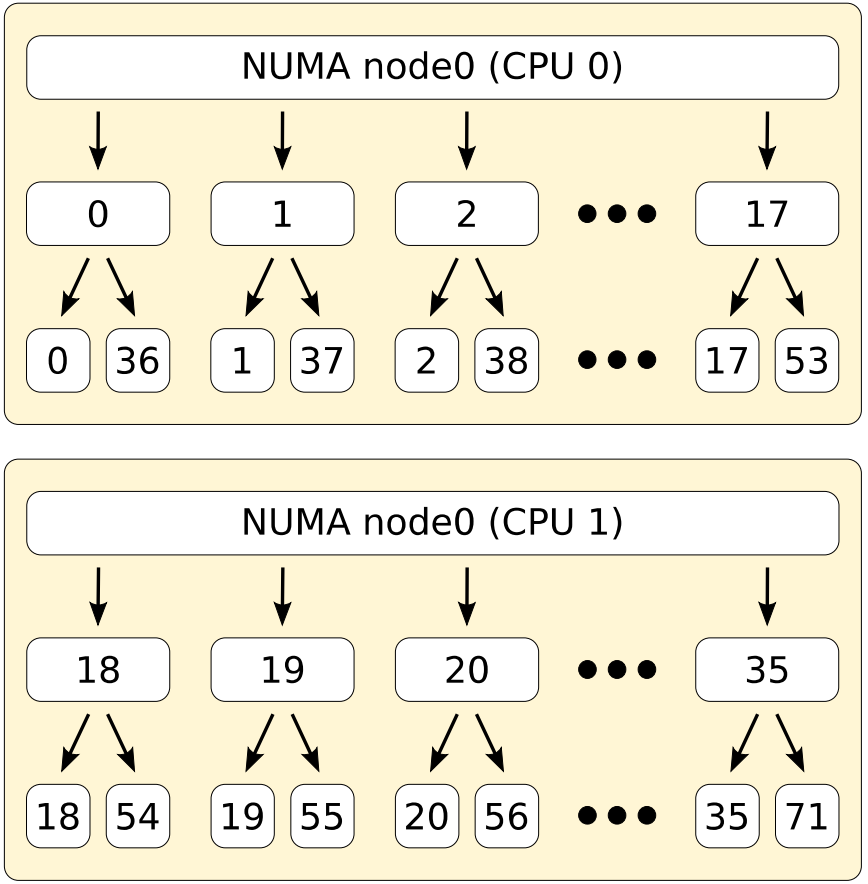
Procesor

↓

Rdzeń

↓

Logiczny rdzeń (nazywany również jako wątek)



Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

Procesor

Rdzeń

Logiczny rdzeń (nazywany również jako wątek)

Wątki OpenMP uruchamiane są z wykorzystaniem logicznych rdzeni

Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

Logiczny rdzeń
(wątek)

Reprezentuje sprzęt

Numer wątku

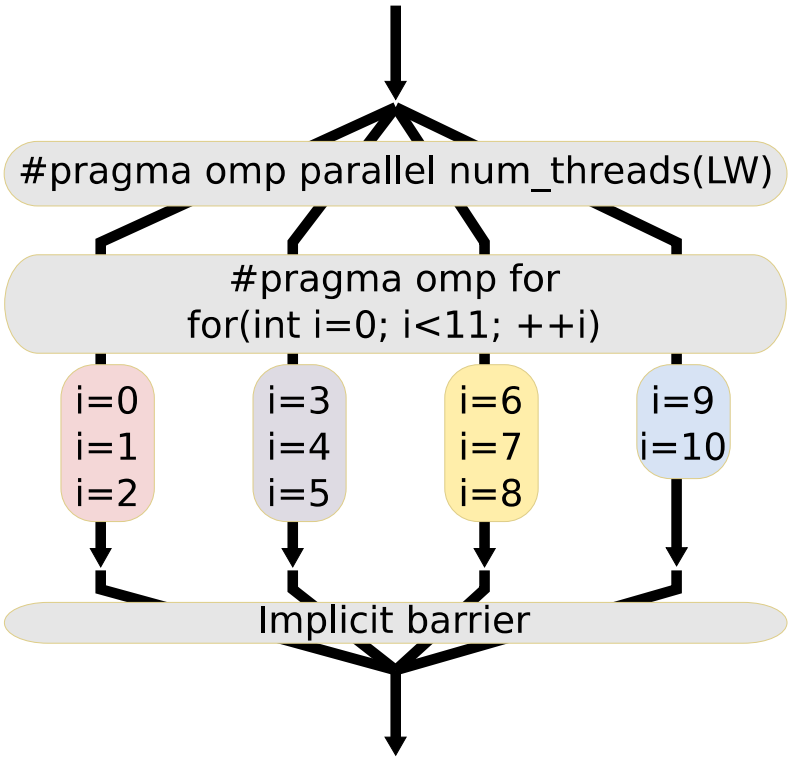
Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

Logiczny rdzeń (wątek)	Wątek OpenMP
Reprezentuje sprzęt	Reprezentuje program
Numer wątku	Numer wątku

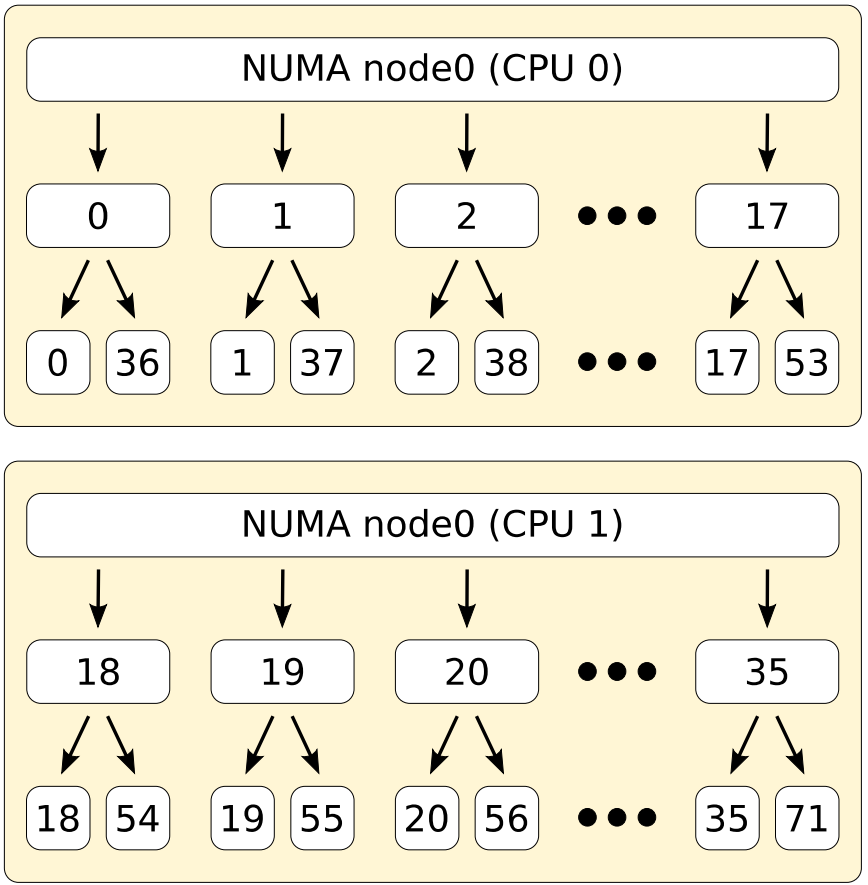
Automatyczne metody dystrybucji obliczeń realizowanych w ramach pętli

Logiczny rdzeń (wątek)		Wątek OpenMP
Reprezentuje sprzęt		Reprezentuje program
Numer wątku	$\neq$	Numer wątku

Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe



?



Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Mechanizmem „affinity” oferowany w ramach OpenMP wykorzystywany jest do zdefiniowania sposobu w jaki procesy i wątki OpenMP są przypisywane do jednostek obliczeniowych
- Mapowanie obliczeń równoległych na zasoby fizyczne realizowane jest z wykorzystaniem następujących zmiennych środowiskowych:
  - **OMP_PROC_BIND** – definiuje strategię wiązania wątków OpenMP z wątkami logicznych rdzeni
  - **OMP_PLACES** – określa miejsce lub możliwe miejsca ulokowania wątków OpenMP
  - **OMP_NUM_THREADS –** definiuje liczbę wątków OpenMP dla danego programu

Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

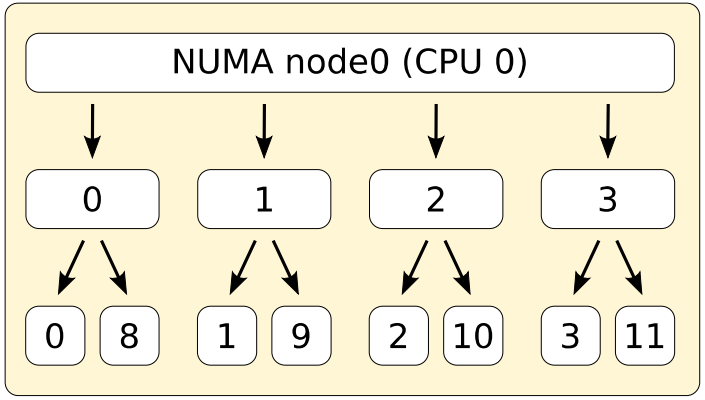
OMP_PROC_BIND	Znaczenie
false	Mapowanie wyłączone. Brak kontroli dla mapowania wątków OpenMP na wątki logicznych rdzeni. Wątki OpenMP mogą migrować pomiędzy rdzeniami.
True	Mapowanie włączone, zgodnie ze zdefiniowaną strategią lokowania wątków OpenMP. Wątki OpenMP nie mogą migrować pomiędzy rdzeniami.
close	Kolejne wątki OpenMP mapowane są na kolejne wątki logicznych rdzeni.
spread	Kolejne wątki OpenMP mapowane są na równomiernie rozdzielone wątki logicznych rdzeni.
master	Wszystkie wątki OpenMP mapowane są na logiczny rdzeń, z pomocą którego został uruchomiony proces główny.

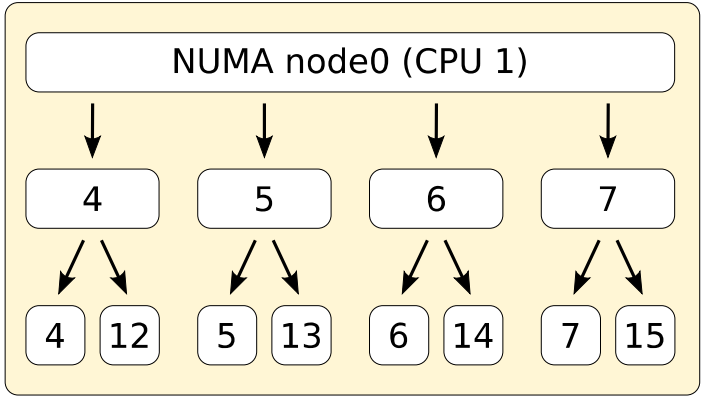
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

OMP_PLACES	Znaczenie
sockets	Dany wątek OpenMP może zostać uruchomionych na zasobach oferowanych w ramach dostępnego procesora.
core	Dany wątek OpenMP może zostać uruchomionych na zasobach oferowanych w ramach pojedynczego rdzenia fizycznego.
threads	Dany wątek OpenMP może zostać uruchomionych na zasobach oferowanych w ramach pojedynczego rdzenia logicznego nazywanego również wątkiem.

Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- 2x CPUs, 2x 4 cores, SMT=2
- **OMP_PROC_BIND = close**
- **OMP_PLACES:**
  - sockets
  - core
  - threads





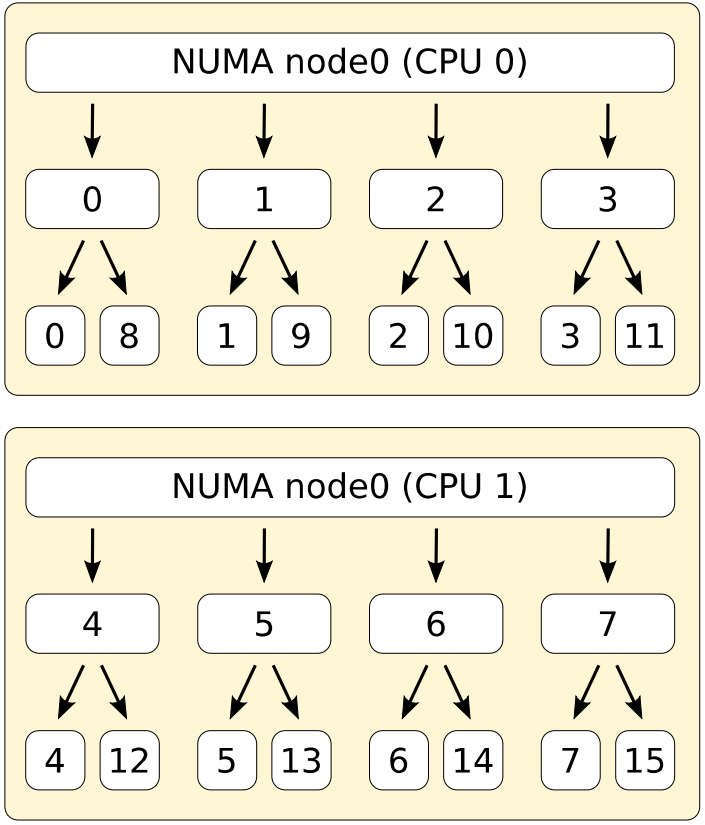
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **A:**

  OMP_NUM_THREADS=2

  OMP_PROC_BIND=close

  OMP_PLACES=sockets



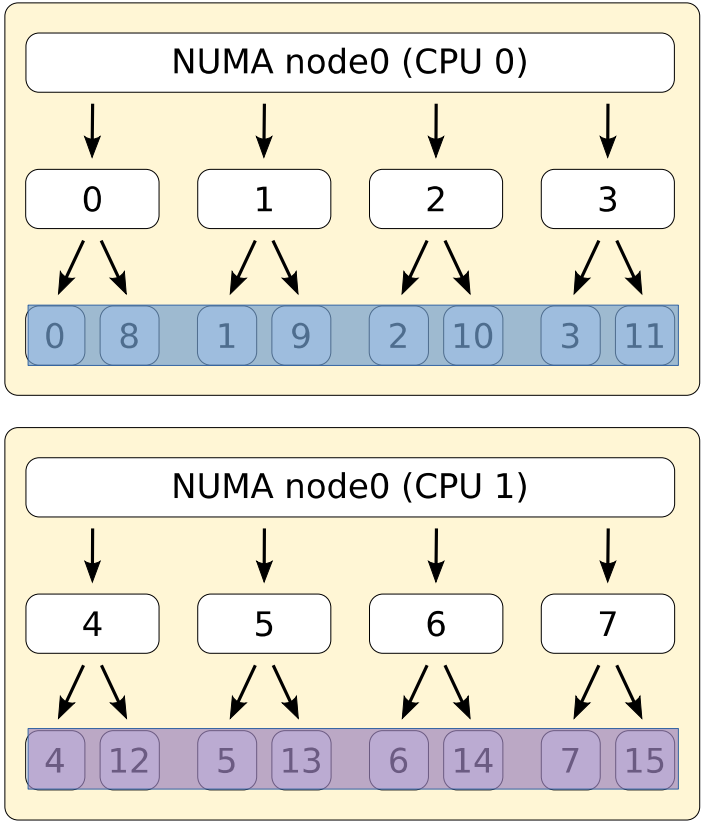
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **A**:

  OMP_NUM_THREADS=2

  OMP_PROC_BIND=close

  OMP_PLACES=sockets



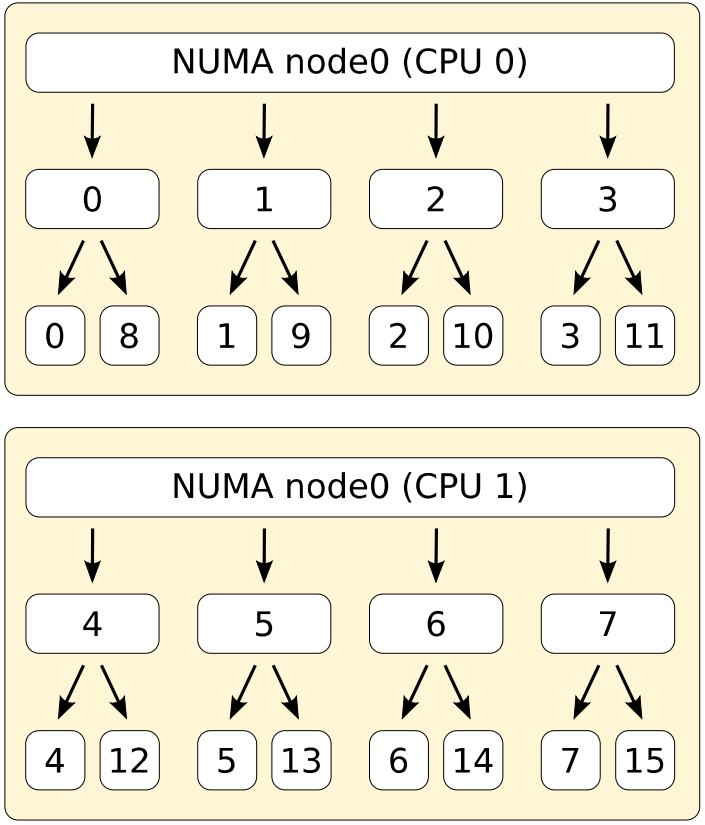
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **A1:**

  OMP_NUM_THREADS=**4**

  OMP_PROC_BIND=close

  OMP_PLACES=sockets



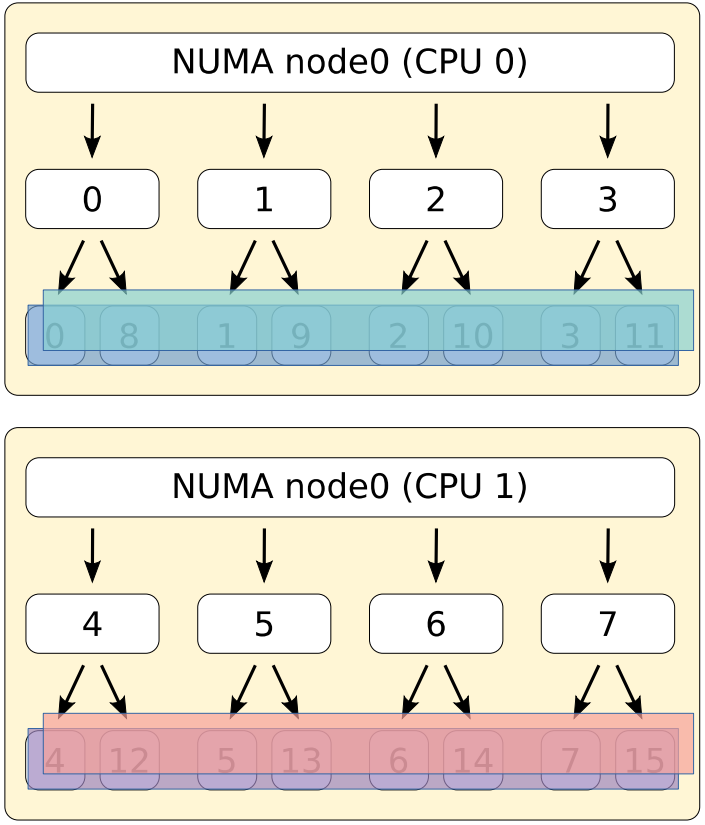
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **A1**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=close

  OMP_PLACES=sockets



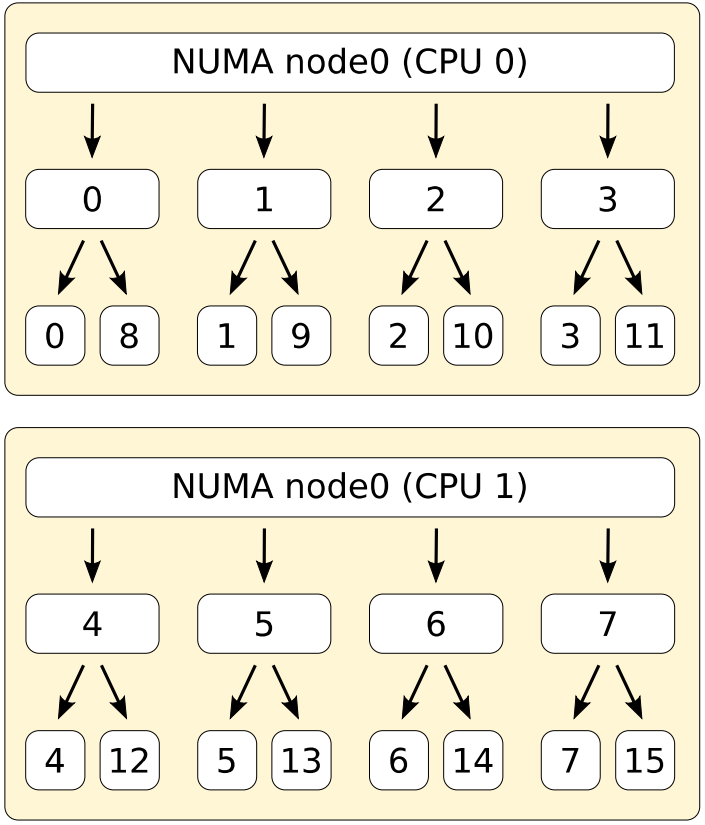
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **B**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=close

  OMP_PLACES=**cores**



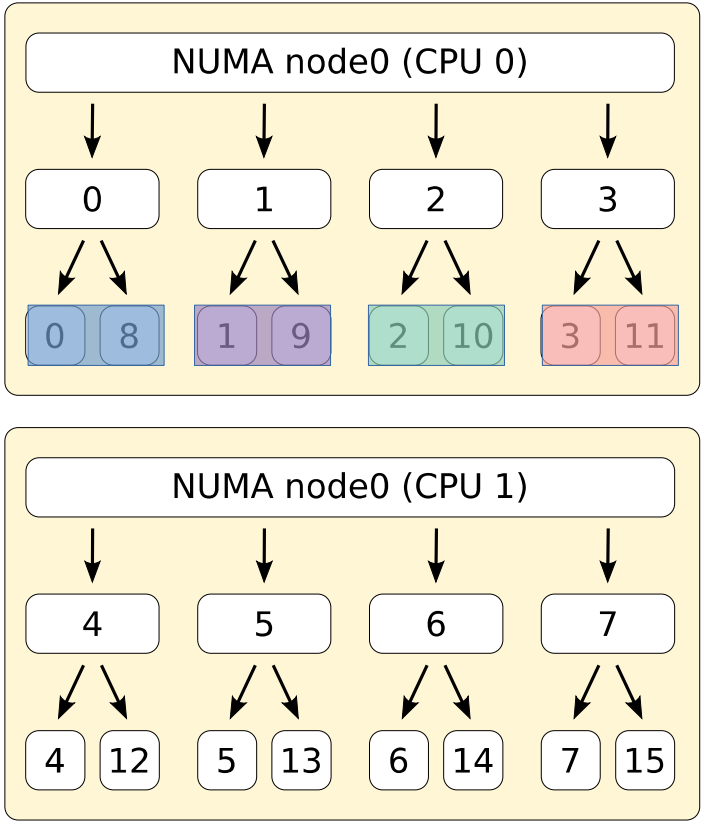
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **B**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=close

  OMP_PLACES=cores



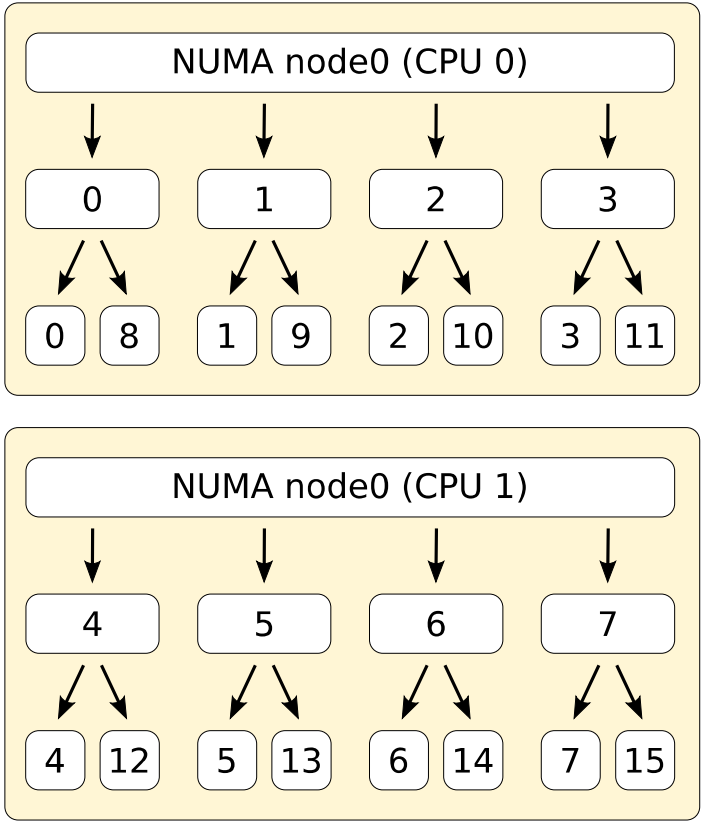
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **C**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=close

  OMP_PLACES=**threads**



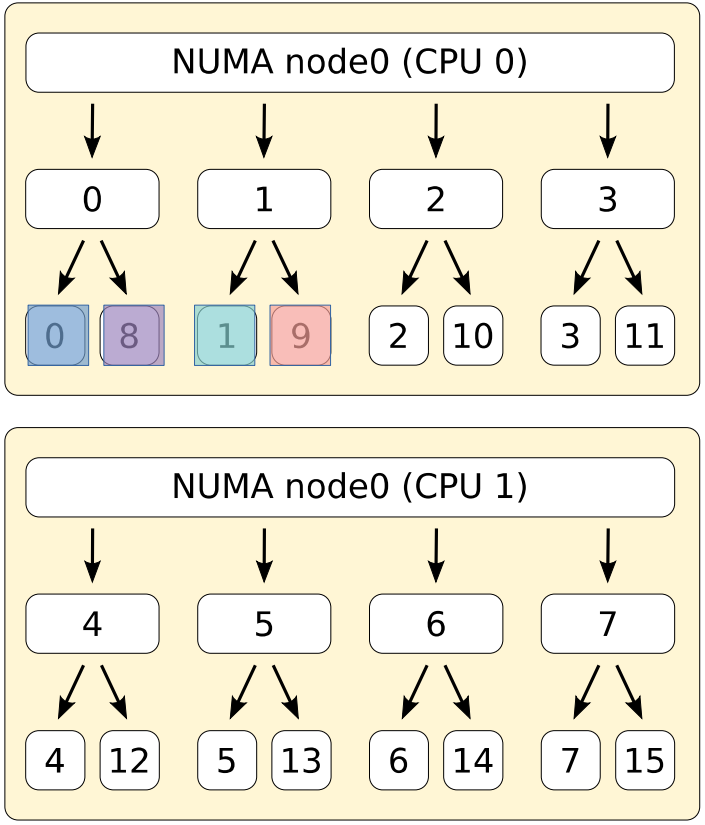
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **C**:

  OMP_NUM_THREADS=4

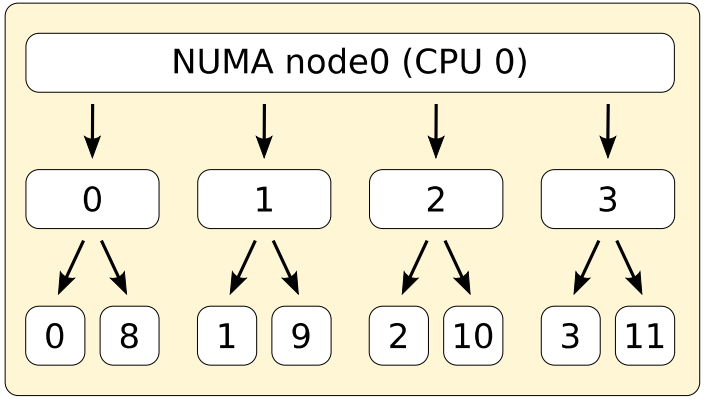
  OMP_PROC_BIND=close

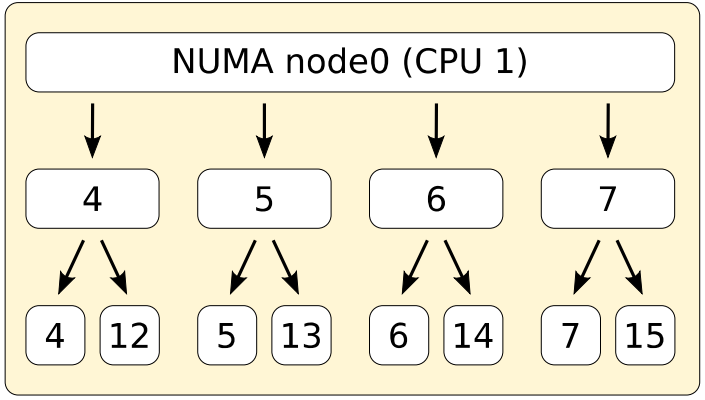
  OMP_PLACES=threads



Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- 2x CPUs, 2x 4 cores, SMT=2
- **OMP_PROC_BIND = spread**
- **OMP_PLACES:**
  - sockets
  - core
  - threads





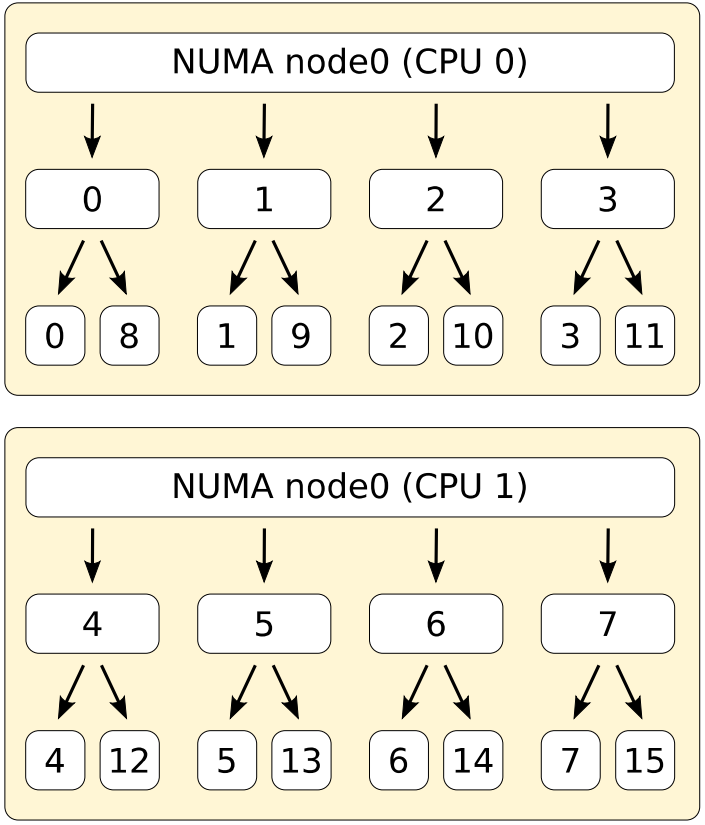
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **D:**

  OMP_NUM_THREADS=2

  OMP_PROC_BIND=**spread**

  OMP_PLACES=sockets



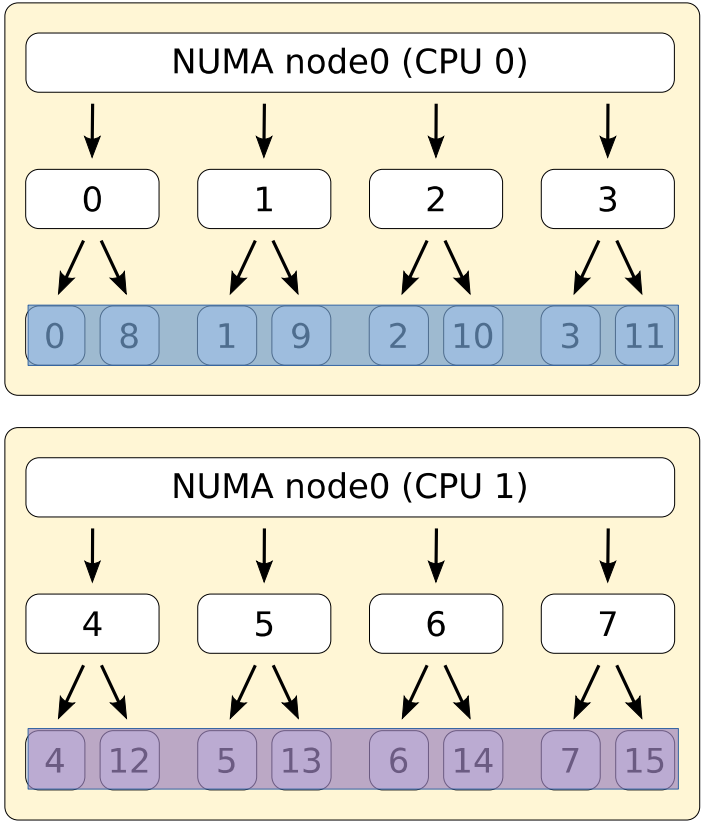
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **D**:

  OMP_NUM_THREADS=2

  OMP_PROC_BIND=spread

  OMP_PLACES=sockets



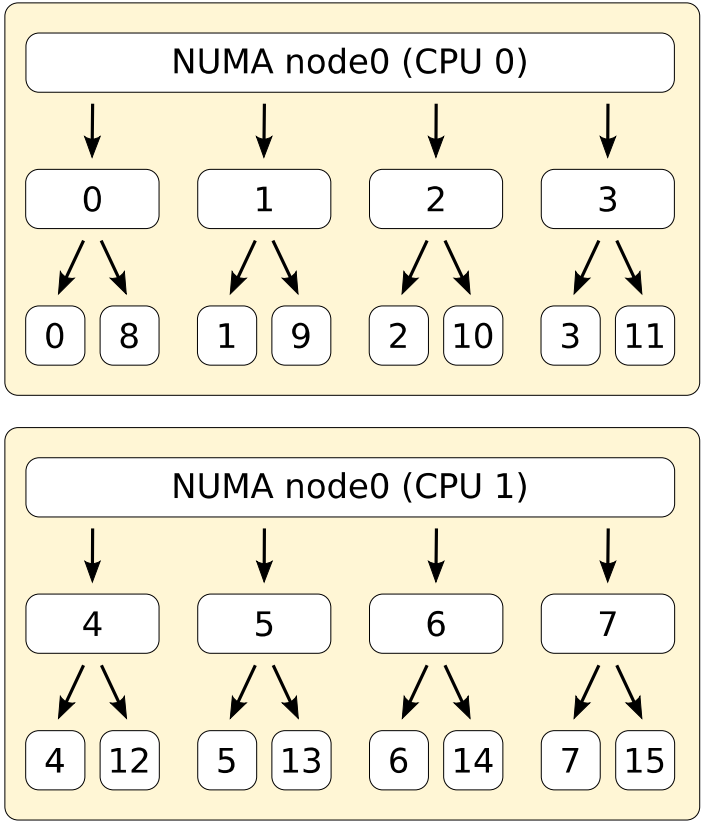
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **E**:

  OMP_NUM_THREADS=**4**

  OMP_PROC_BIND=spread

  OMP_PLACES=**cores**



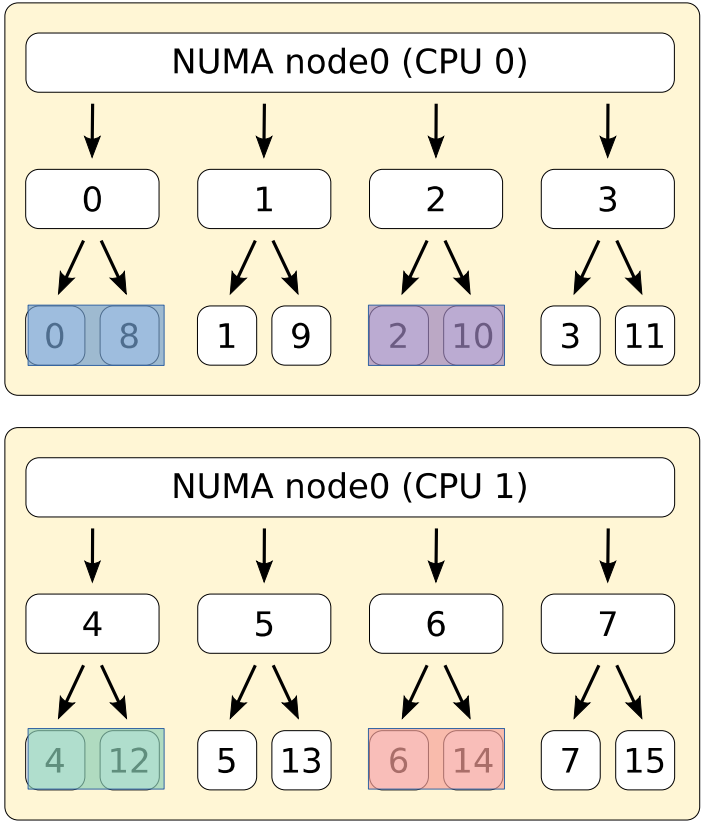
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **E**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=spread

  OMP_PLACES=cores



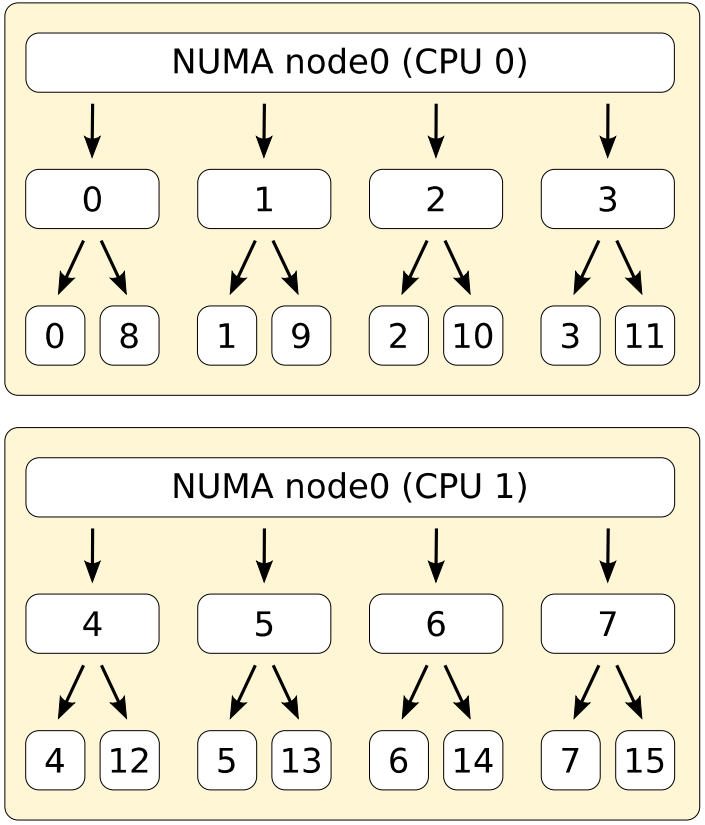
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **F**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=spread

  OMP_PLACES=**threads**



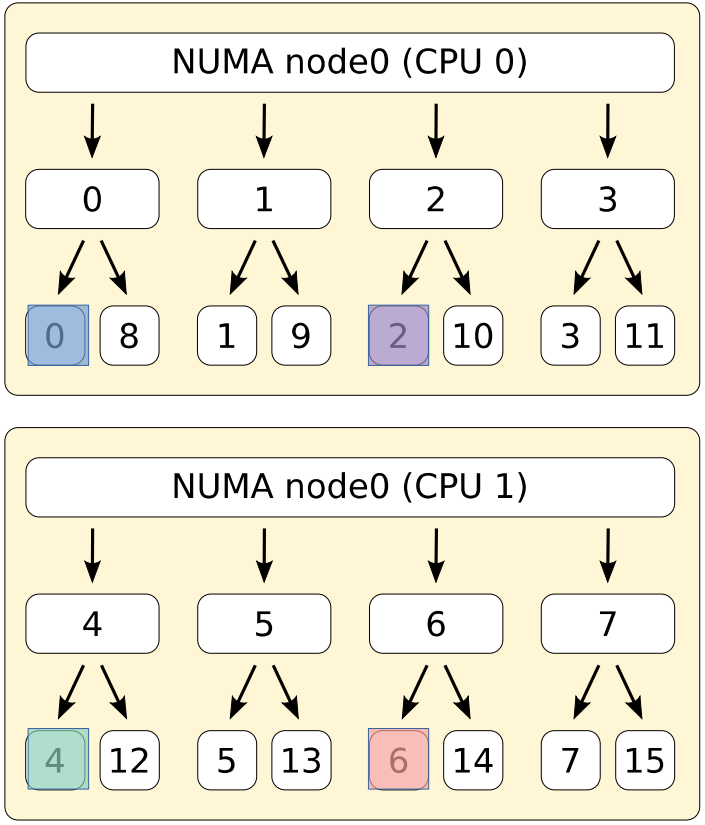
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **F**:

  OMP_NUM_THREADS=4

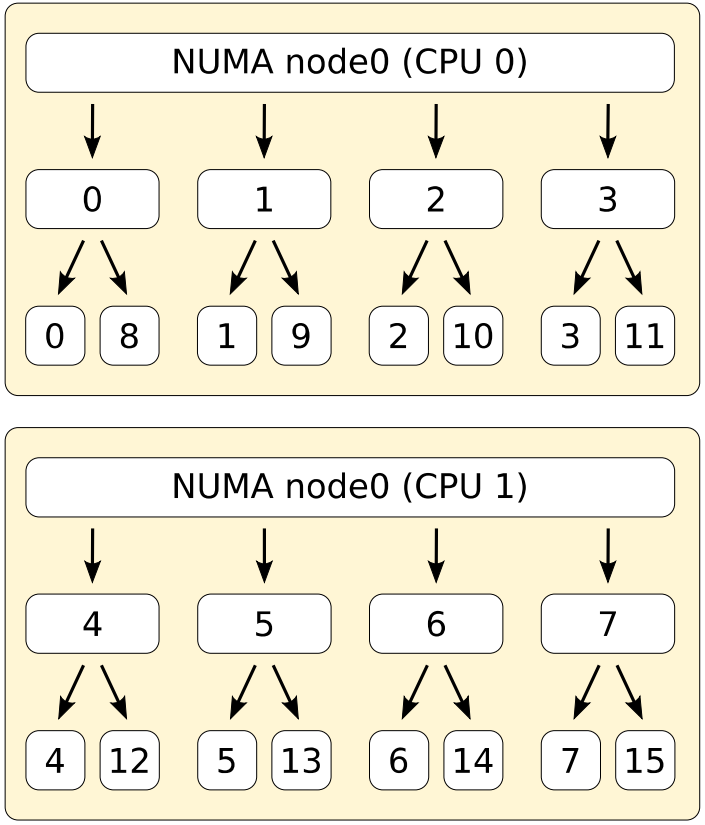
  OMP_PROC_BIND=spread

  OMP_PLACES=threads



Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- 2x CPUs, 2x 4 cores, SMT=2
- **OMP_PROC_BIND = master**
- **OMP_PLACES:**
  - sockets
  - core
  - threads



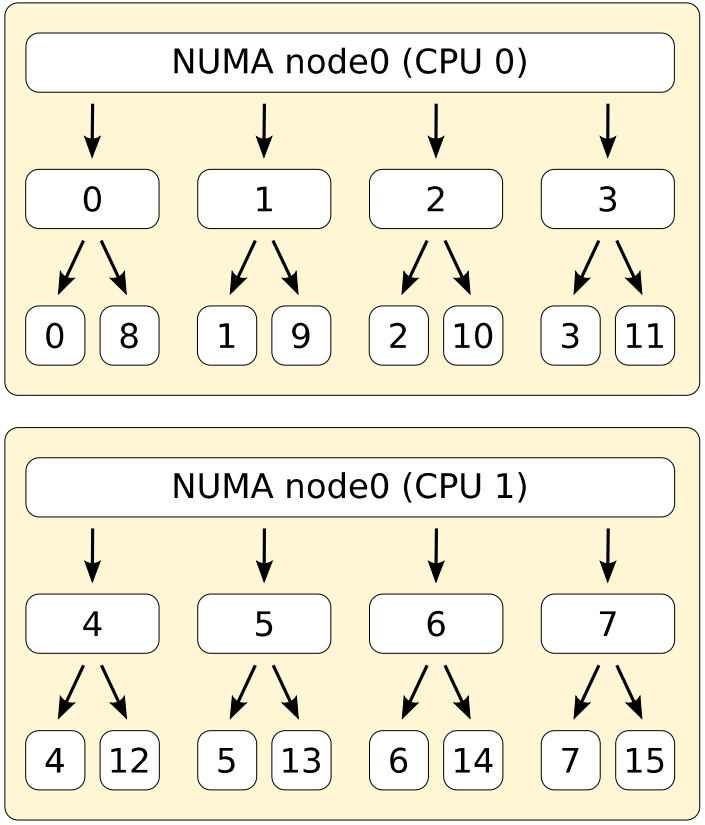
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **G:**

  OMP_NUM_THREADS=2

  OMP_PROC_BIND=master

  OMP_PLACES=sockets



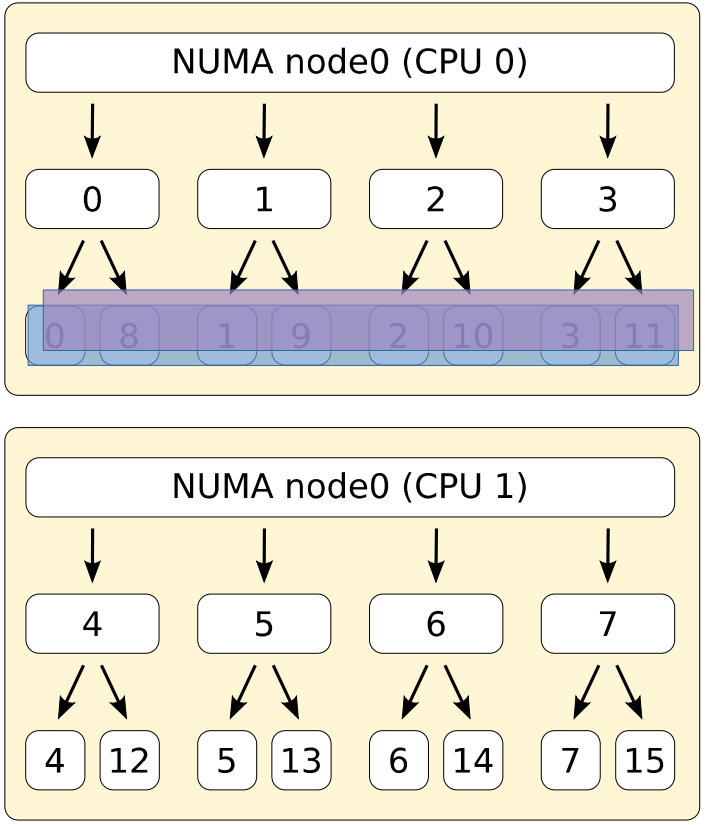
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **G**:

  OMP_NUM_THREADS=2

  OMP_PROC_BIND=master

  OMP_PLACES=sockets



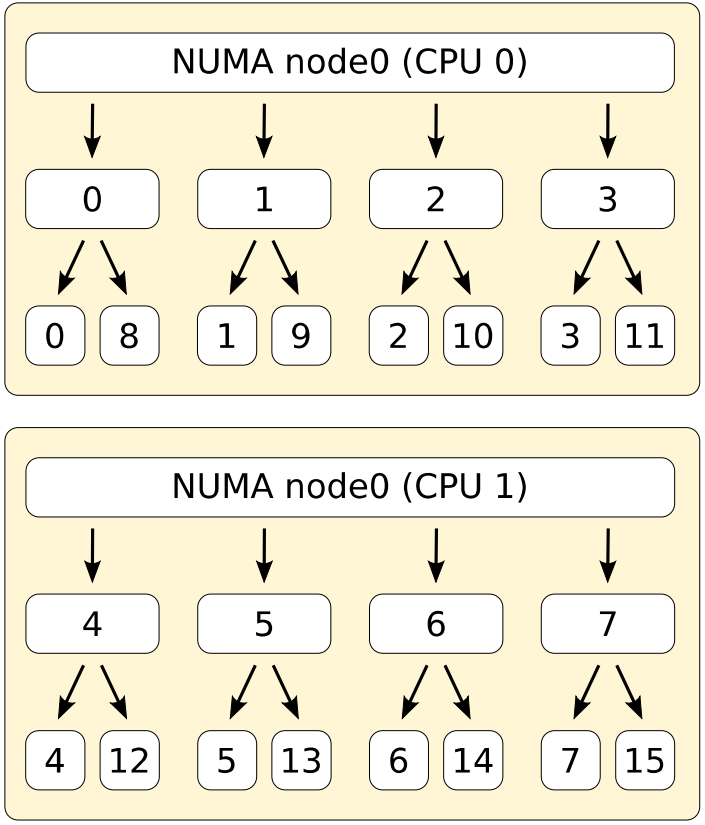
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **H**:

  OMP_NUM_THREADS=**4**

  OMP_PROC_BIND=master

  OMP_PLACES=**cores**



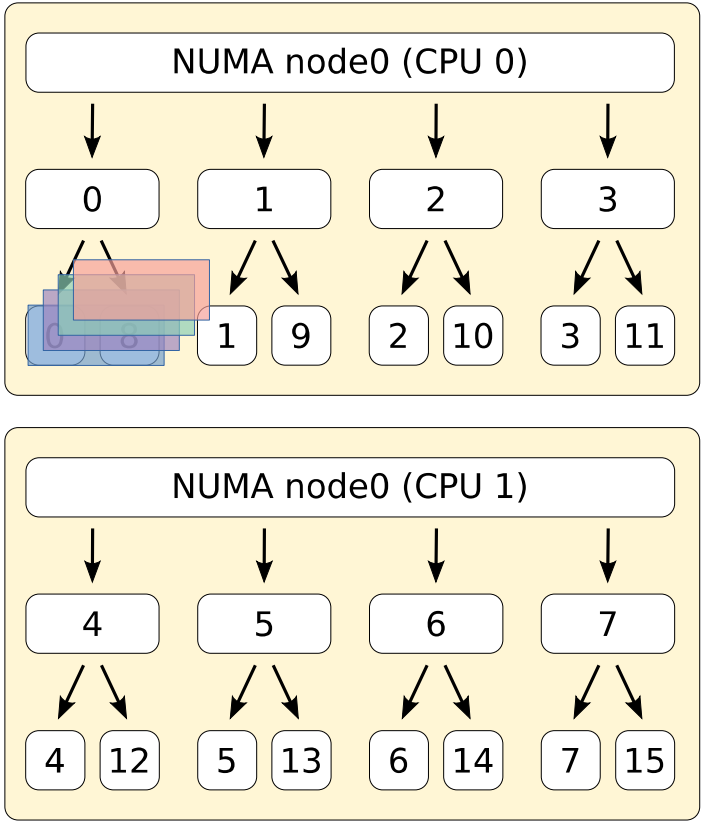
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **H**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=master

  OMP_PLACES=cores



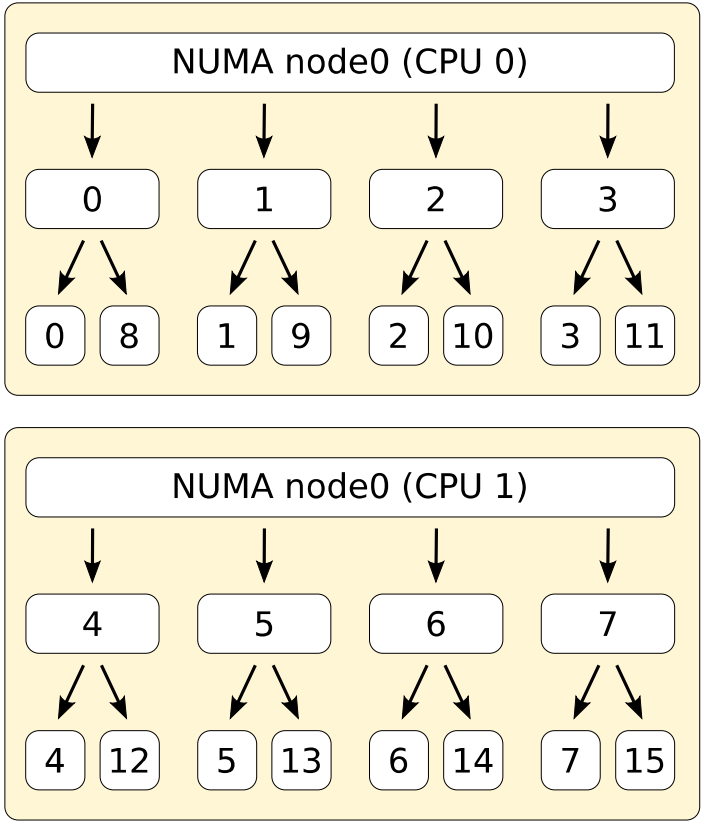
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **I**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=master

  OMP_PLACES=**threads**



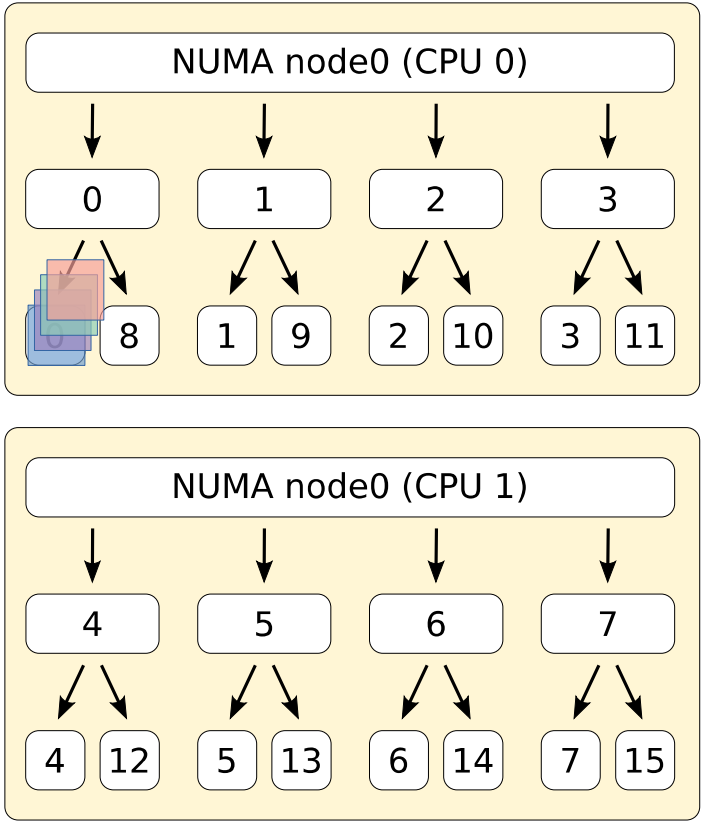
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Przykład **I**:

  OMP_NUM_THREADS=4

  OMP_PROC_BIND=master

  OMP_PLACES=threads



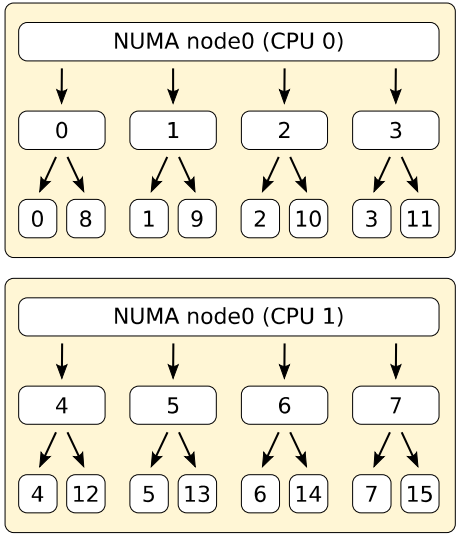
Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- OpenMP pozwala na identyfikację zasobów obliczeniowych, do których wątki OMP zostały przypisane

```
string print_binding_info() {

  ostringstream out;

  int my_place = omp_get_place_num();
  int place_num_procs = omp_get_place_num_procs(my_place);

  //out<<"Place consists of "<<place_num_procs<<" processors located on: ";

  out<<"OMP places: ("<<place_num_procs<<") ";

  int *place_processors = new int[place_num_procs];
  omp_get_place_proc_ids(my_place, place_processors);

  for(int i = 0; i<place_num_procs; ++i) {
    out<<place_processors[i]<<" ";
  }

  delete [] place_processors;

  return out.str();
}
```

Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

- Dodatkowo, funkcja ***sched_getcpu()*** *pobiera* identyfikator logicznego rdzenia, z pomoc którego został uruchomiony wątek OpenMP




```
#include <iostream>
#include <omp.h>
#include <sstream>
#include <sched.h>
#include <iomanip>

using namespace std;

string print_binding_info();

int main() {

  ostringstream * outC = NULL;
  int LW = 0;

  #pragma omp parallel shared(LW,outC)
  {
    #pragma omp master
    {
      LW = omp_get_num_threads();
      outC = new ostringstream[LW];
    }
    #pragma omp barrier
    const int id = omp_get_thread_num();
    int lcId = sched_getcpu();
    outC[id]<<"omp="<<setw(2)<<id<<" "<<"lc="<<setw(2)<<lcId<<" "<<print_binding_info()<<endl;
  }

  for(int i=0; i<LW; ++i) {
    cout<<outC[i].str();
    outC[i].str("");
  }

  if(outC!=NULL)
    delete [] outC;

  return 0;
}
```

Mapowanie obliczeń równoległych na fizyczne zasoby sprzętowe

Zadanie 1

- Zbadaj czas wykonania programu zaimplementowanego w ramach zadania 2 z poprzedniej części, a następnie przetestuj różne konfiguracje dla mapowania obliczeń równoległych na zasoby fizyczne
- Znajdź „najlepszą” konfigurację
- Sprawdź wpływ Intel HT na czas wykonania zadania:
  - Uruchom program z pojedynczym wątkiem OpenMP na każdym rdzeniu
  - Uruchom program z dwoma wątkami OpenMP na każdym rdzeniu

Zadanie 2 z poprzedniej części

- Napisz program równoległy z dowolną liczbą wątków OpenMP umożliwiający wyznaczenie sumy wszystkich elementów dwuwymiarowej tablicy
- W ramach zadania użyj różnych sposobów podzielanie pętli oraz znajdź konfigurację pozwalającą na jak najszybsze wyznaczenie sumy
- Spróbuj zrealizować zadania na dwa sposoby, z użyciem opcji reduction oraz bez niej.

Projekt w ramach programu Regionalna Inicjatywa Doskonałości, decyzja nr 020/RID/2018/19

Dziękuję za uwagę!

Projekt w ramach programu Regionalna Inicjatywa Doskonałości, decyzja nr 020/RID/2018/19

Dziękuję za uwagę!