# Parallel Programming With OpenMP

Edmund Sumbar
University of Alberta

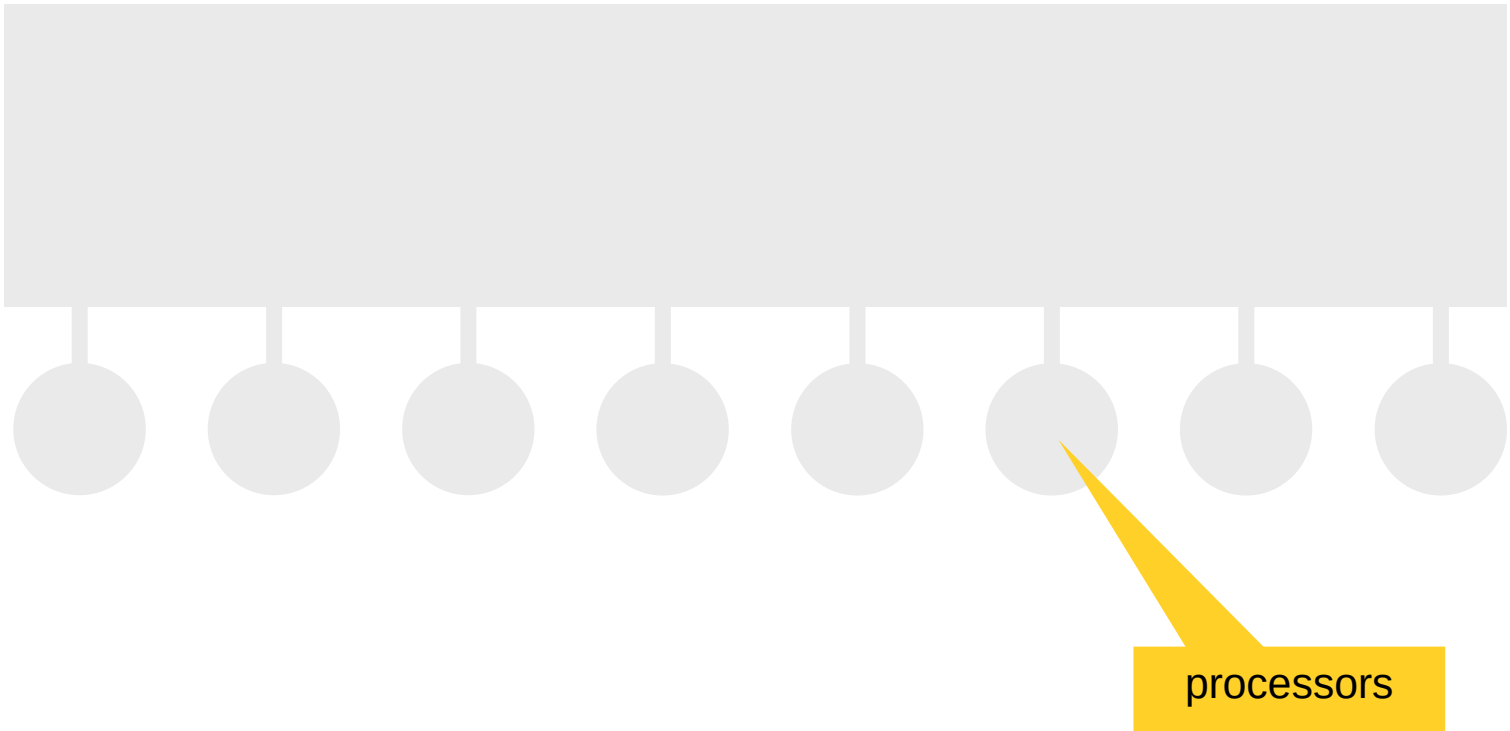
- Standardization
  - OpenMP standardizes previously existing vendor-specific directive-based parallelization environments
  - First implementations available in 1998
  - Current version is 2.0

- Key points
  - Shared memory multiprocessor nodes
  - Parallel programming using compiler directives
  - Fortran 77/90/95 and C/C++

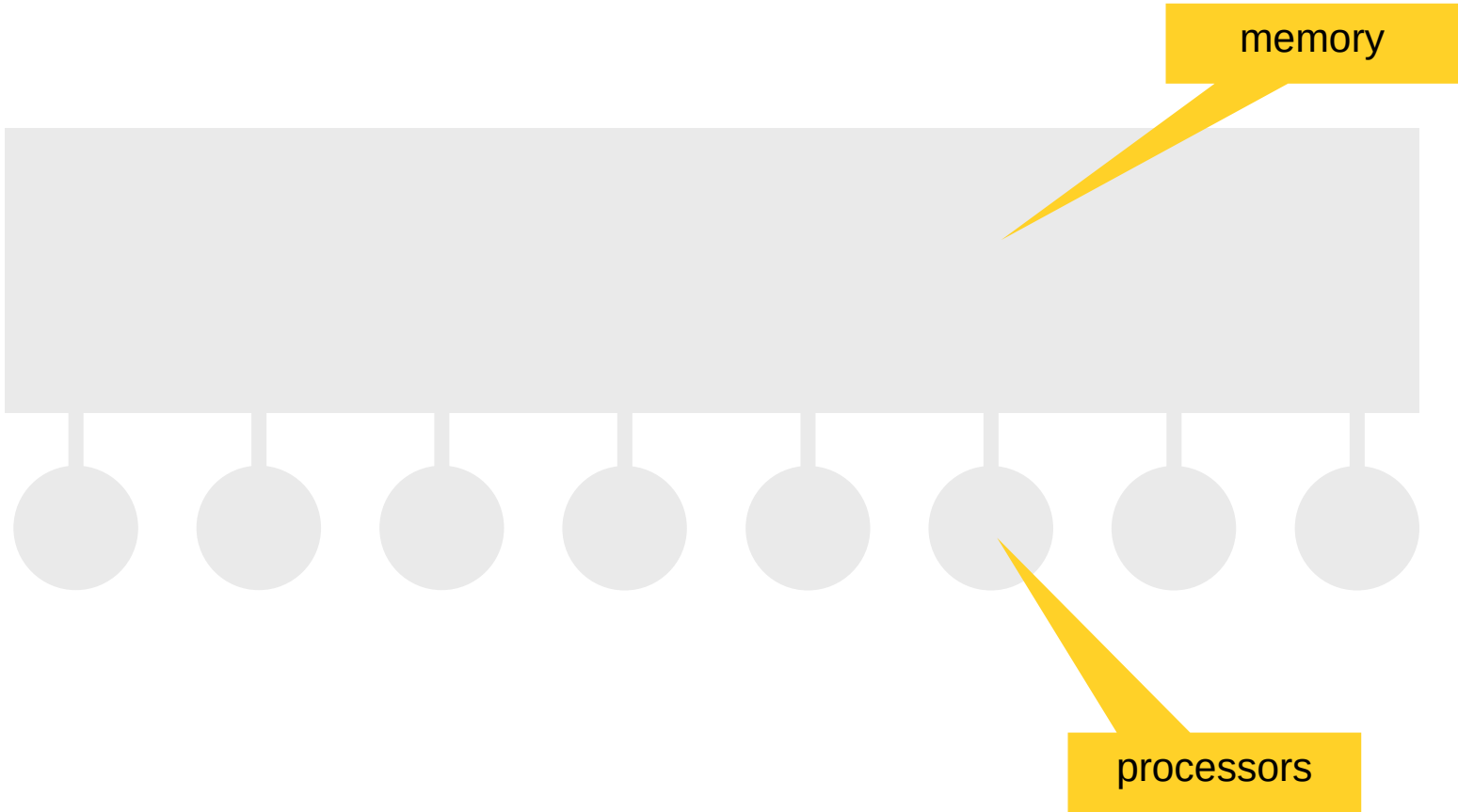
- Shared memory multiprocessor node…

- Shared memory multiprocessor node…

- Shared memory multiprocessor node…



- Shared memory multiprocessor node…

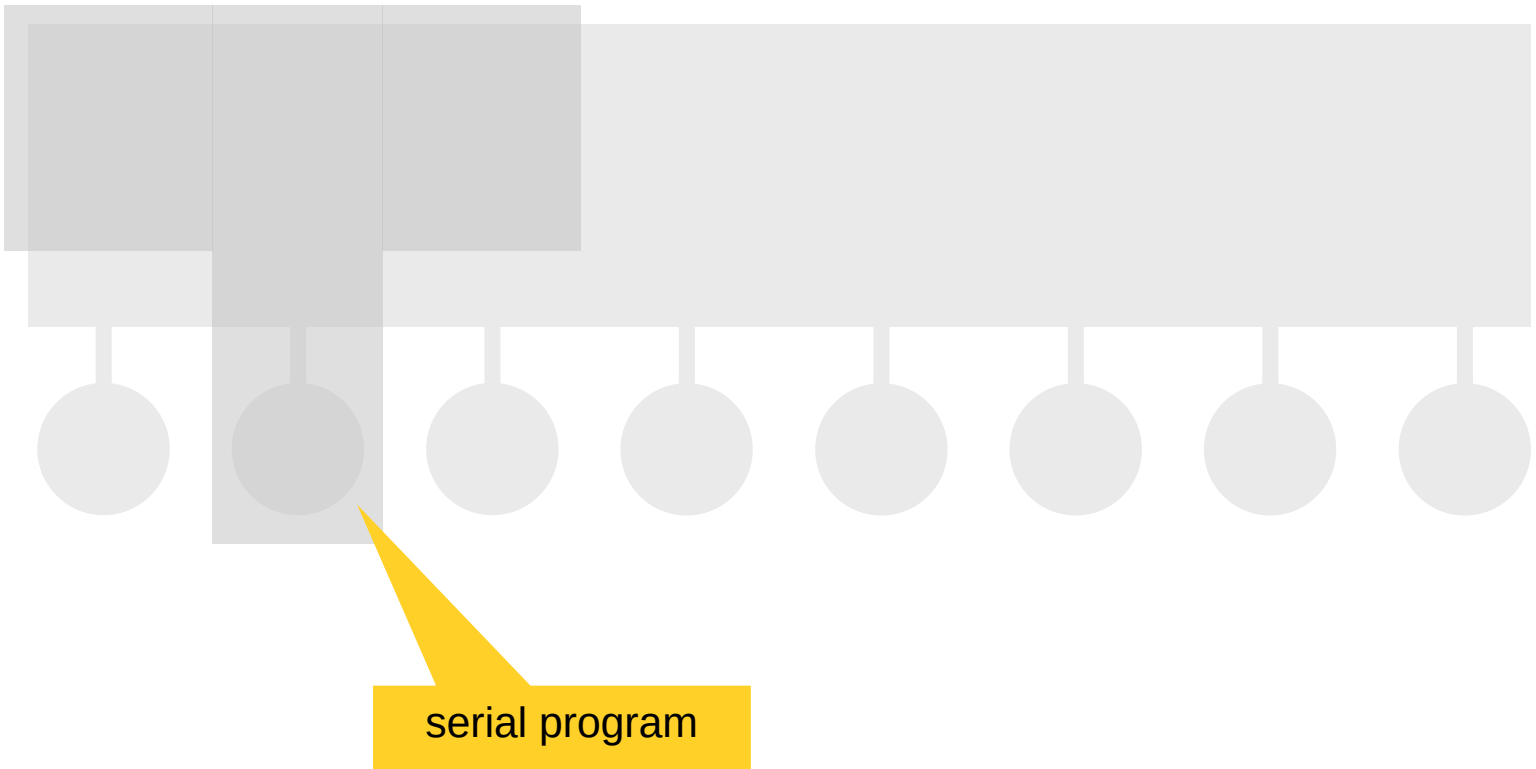




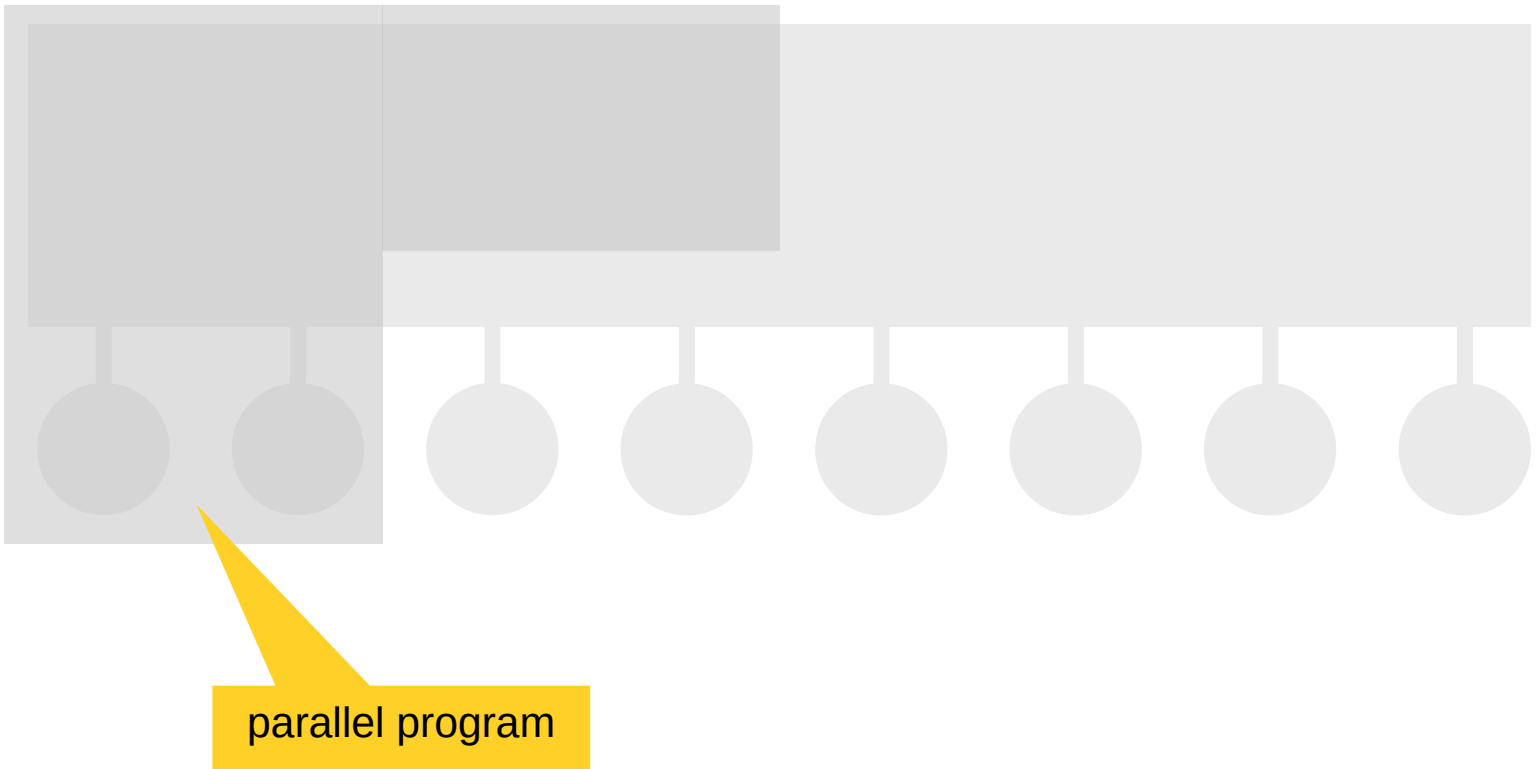


create delete

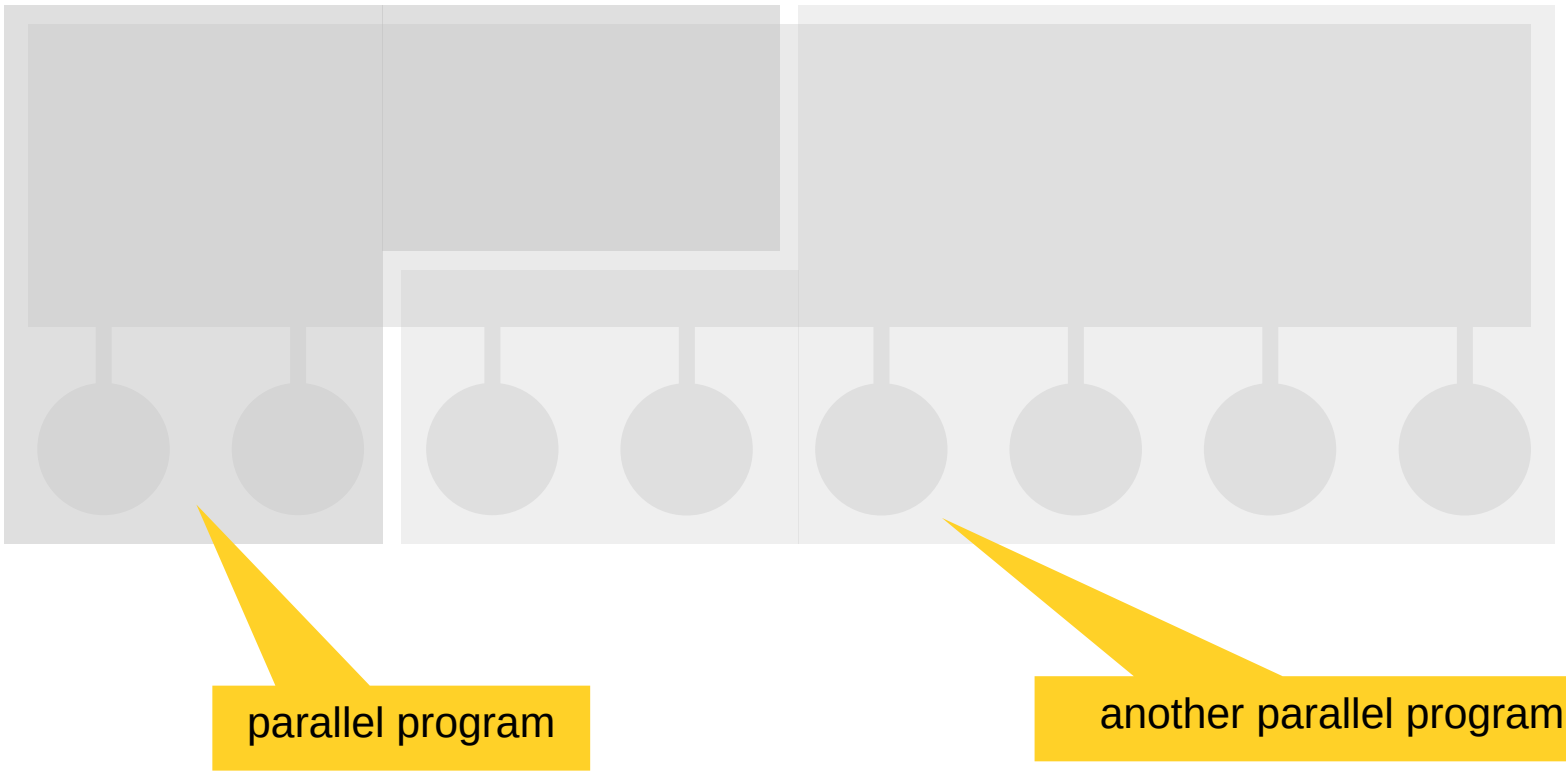
- Shared memory multiprocessor node…



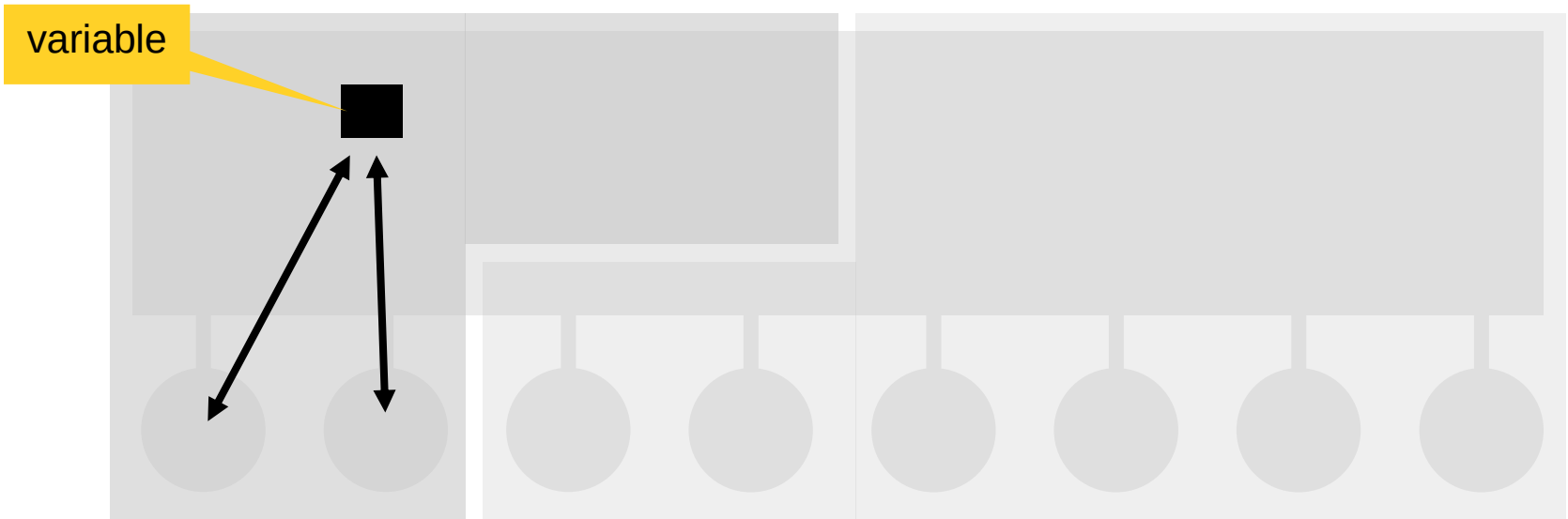
- Shared memory multiprocessor node…



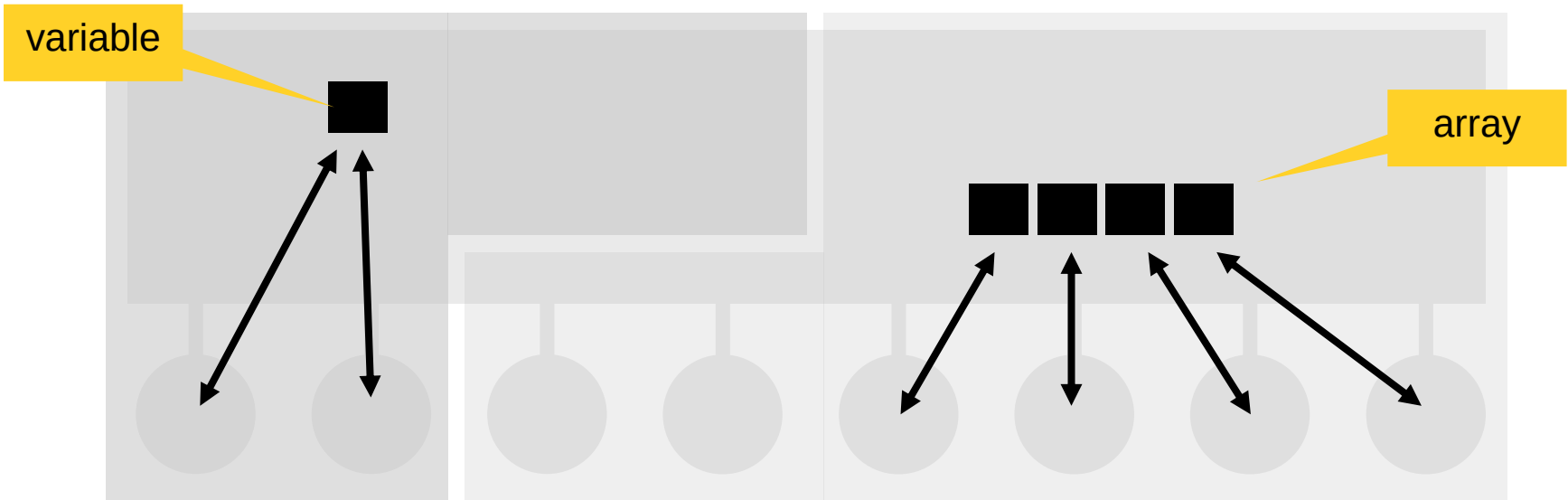
- Shared memory multiprocessor node…



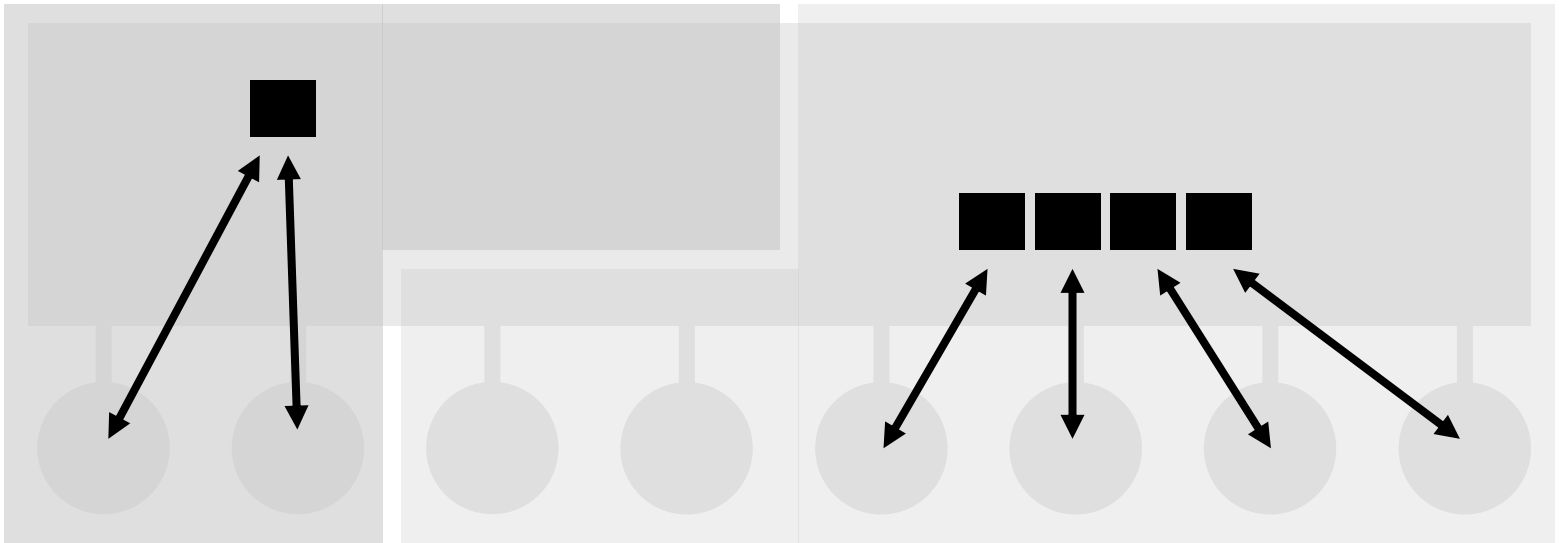
- Shared memory multiprocessor node…



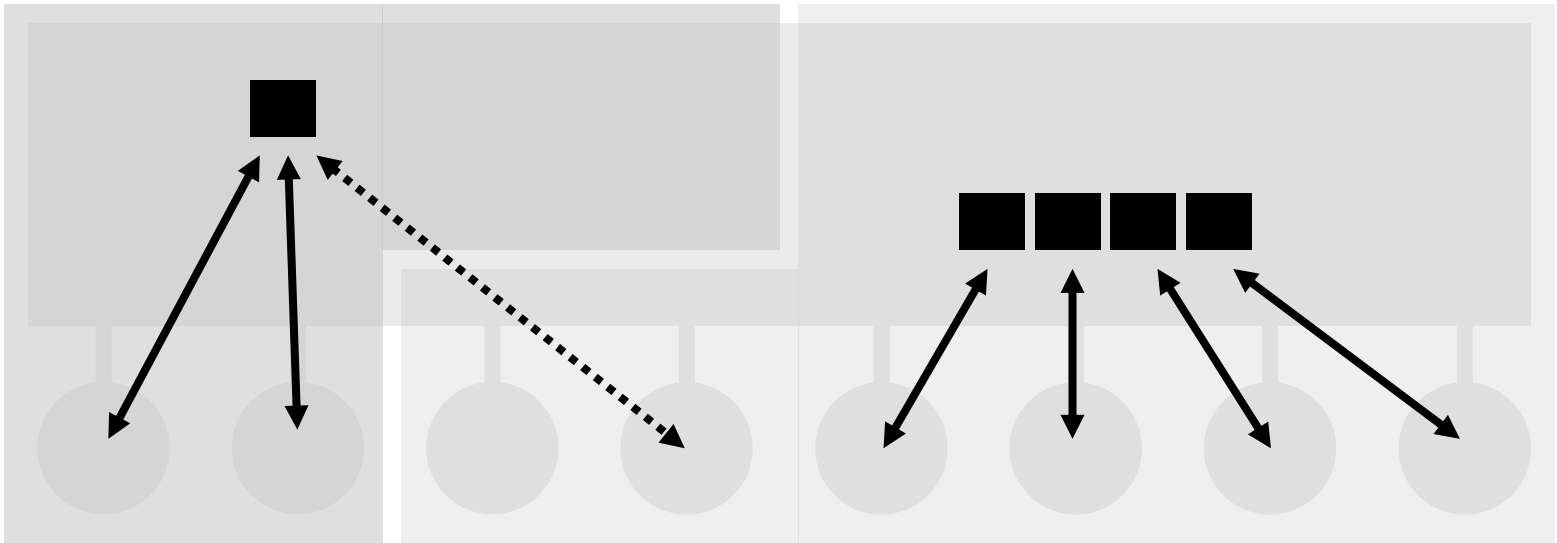
- Shared memory multiprocessor node…



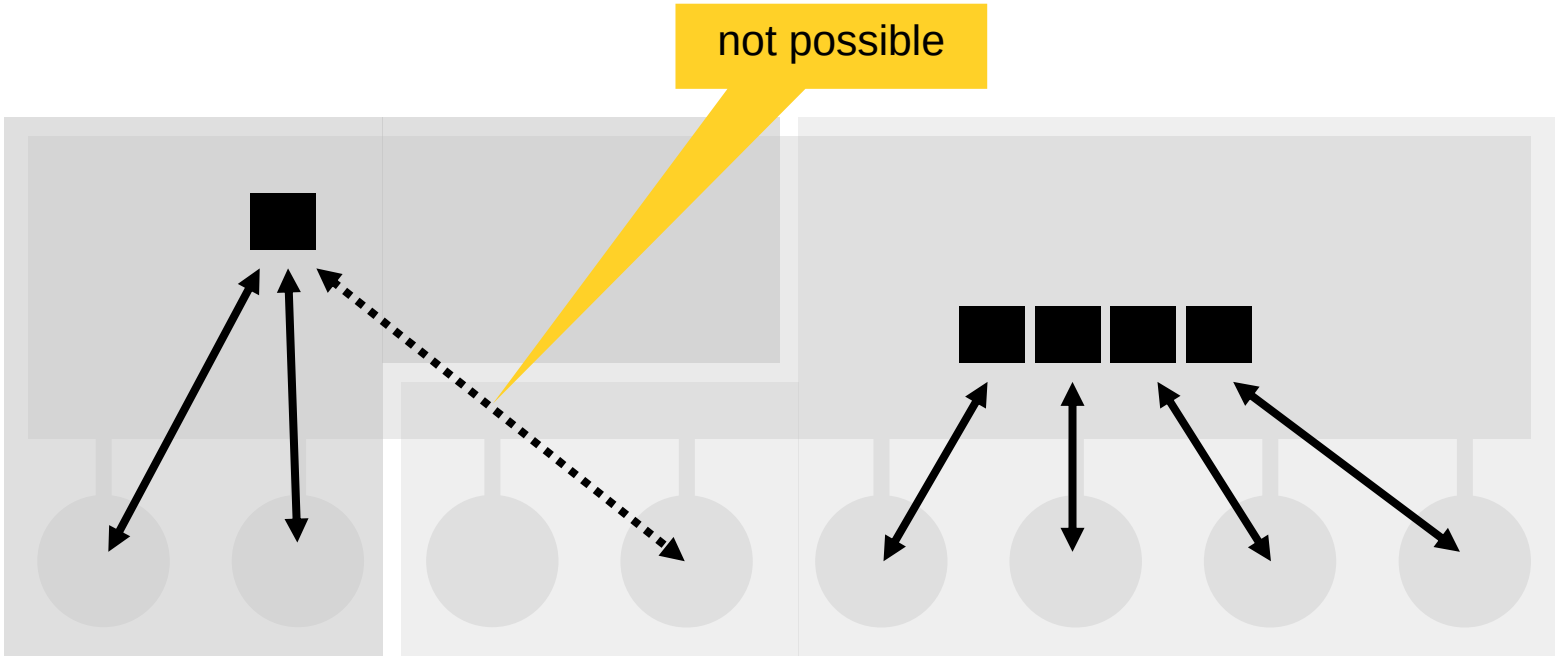
- Shared memory multiprocessor node…



- Shared memory multiprocessor node…

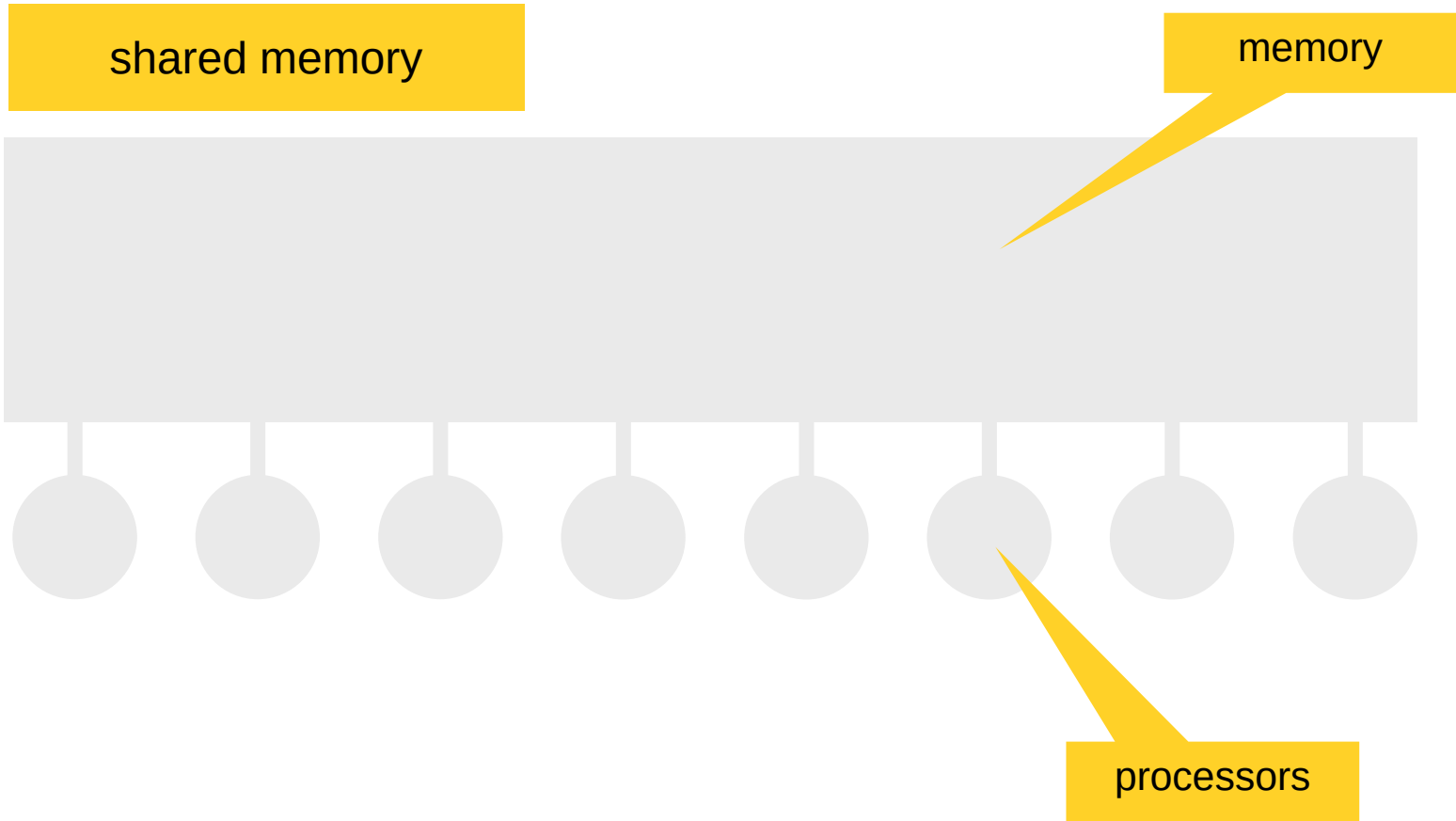


- Shared memory multiprocessor node…

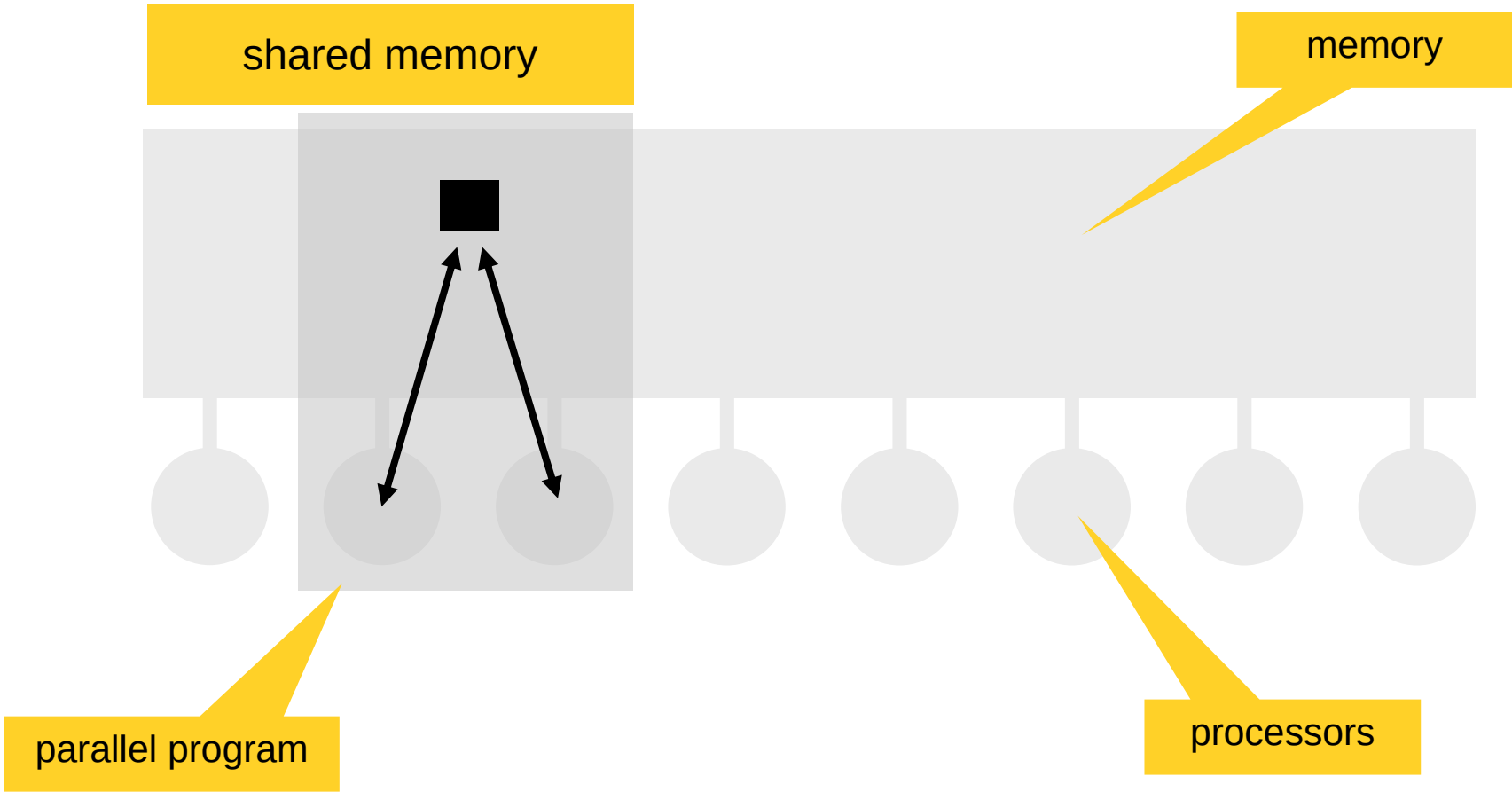


- Distributed memory comparison…

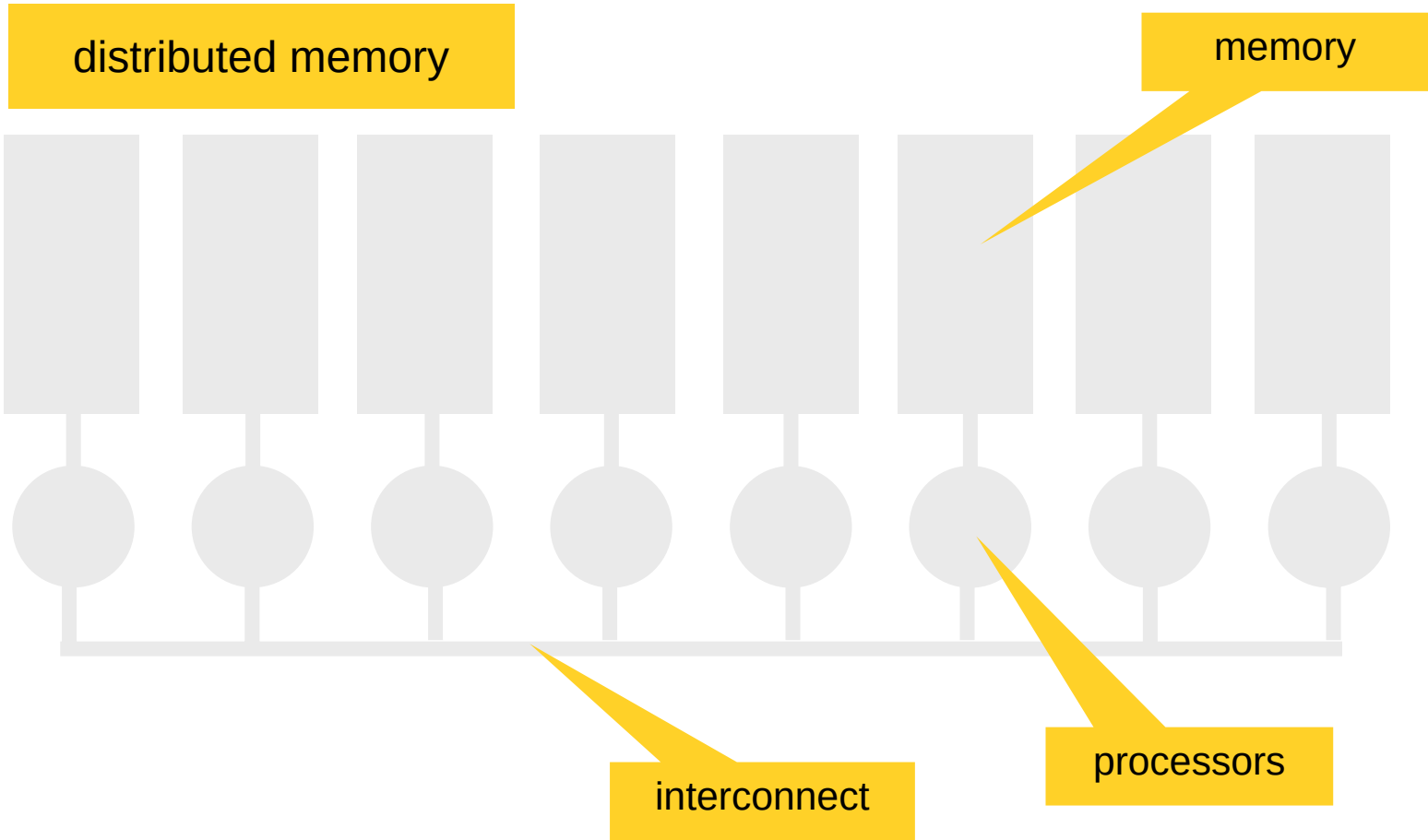
- Distributed memory comparison…



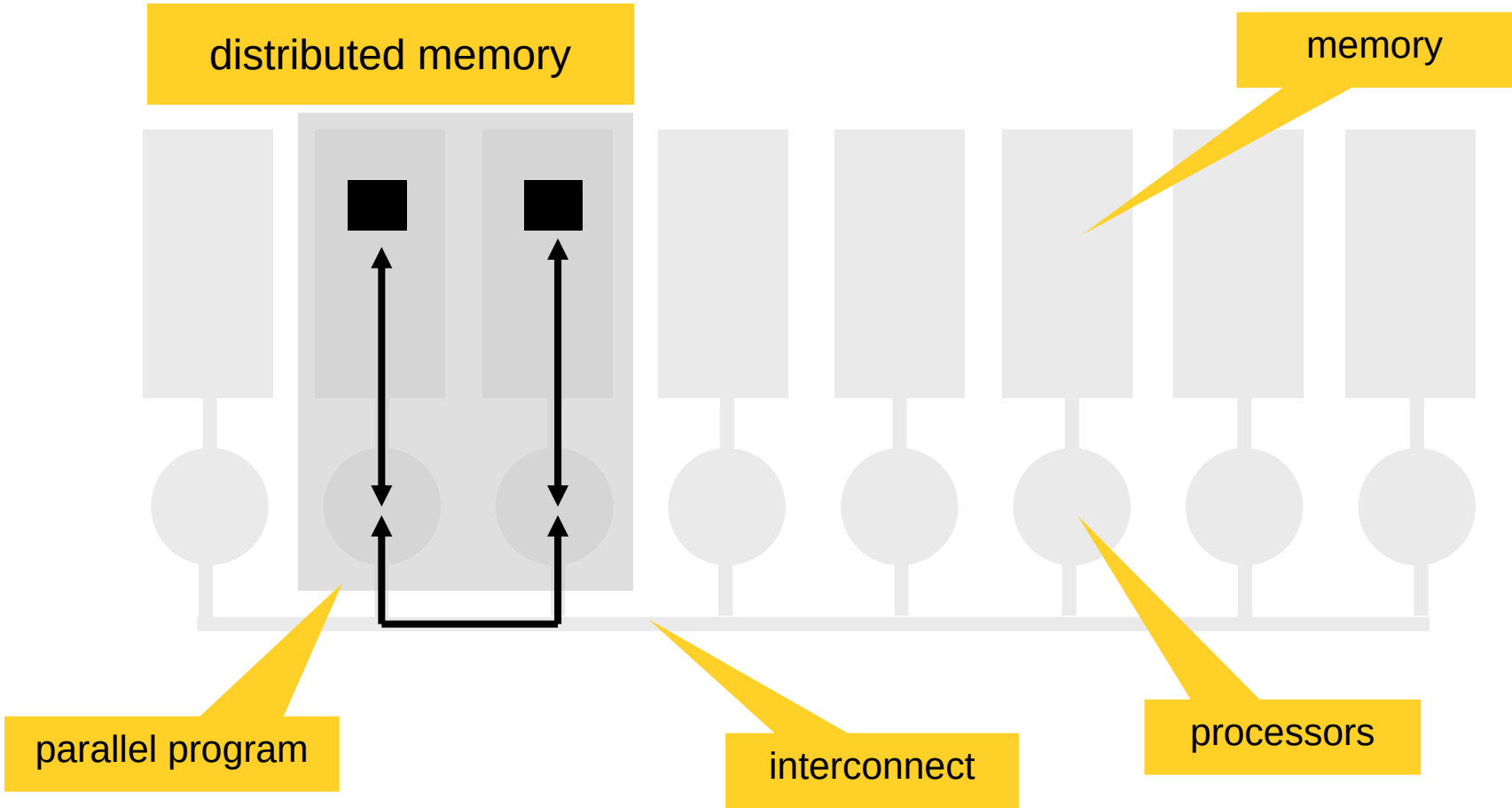
- Distributed memory comparison…



- Distributed memory comparison…



- Distributed memory comparison…



- Fundamental OpenMP concepts...

- Fundamental OpenMP concept
  **parallel regions**

- Parallel regions in C...

```
#include <stdio.h>

int
main (void)
{
  printf ("Hello, world\n");
  return 0;
}
```

- Parallel regions in C...

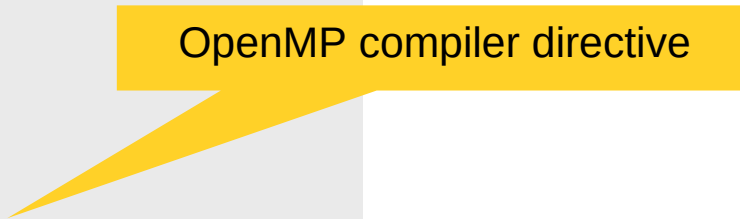
```
#include <stdio.h>

int
main (void)
{
  printf ("Hello, world\n");
  return 0;
}
```

- Parallel regions in C...

```
#include <stdio.h>

int
main (void)
{
  {
    printf ("Hello, world\n");
  }
  return 0;
}
```

- Parallel regions in C...

```
#include <stdio.h>

int
main (void)
{
#pragma omp parallel
  {
    printf ("Hello, world\n");
  }
  return 0;
}
```



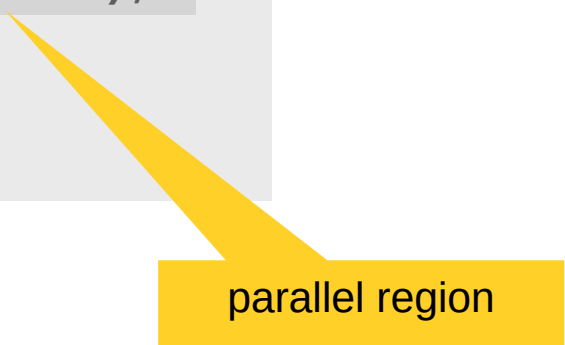




create delete

- Parallel regions in C...

```
#include <stdio.h>

int
main (void)
{
#pragma omp parallel
  {
    printf ("Hello, world\n");
  }
  return 0;
}
```



- Parallel regions in Fortran…

```
program hello
print*, ‘Hello, world’
end
```

- Parallel regions in Fortran…

```
program hello
print*, 'Hello, world'
end
```

- Parallel regions in Fortran…

```
      program hello
c$omp parallel
      print*, 'Hello, world'
      end
```

opening OpenMP compiler directive

- Parallel regions in Fortran…

```
      program hello
c$omp parallel
      print*, 'Hello, world'
c$omp end parallel
      end
```

closing OpenMP compiler directive

- Parallel regions in Fortran…

```
      program hello
c$omp parallel
      print*, 'Hello, world'
c$omp end parallel
      end
```

parallel region

- Parallel regions and the thread model…

- Parallel regions and the thread model…

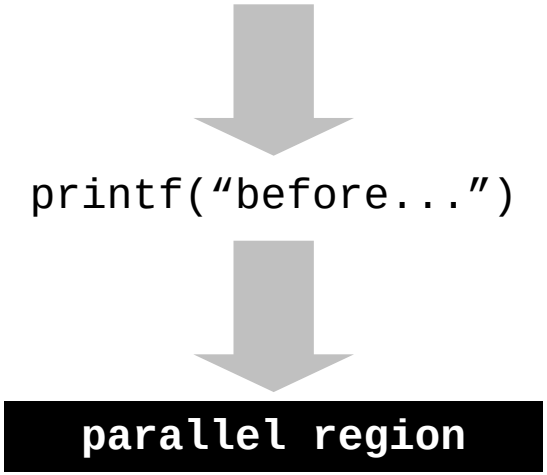
```
#include <stdio.h>

int main (void)
{
  printf (“before parallel region\n”);

#pragma omp parallel
  {
    printf (“Hello, world\n”);
  }

  printf (“after parallel region\n”);
  return 0;
}
```

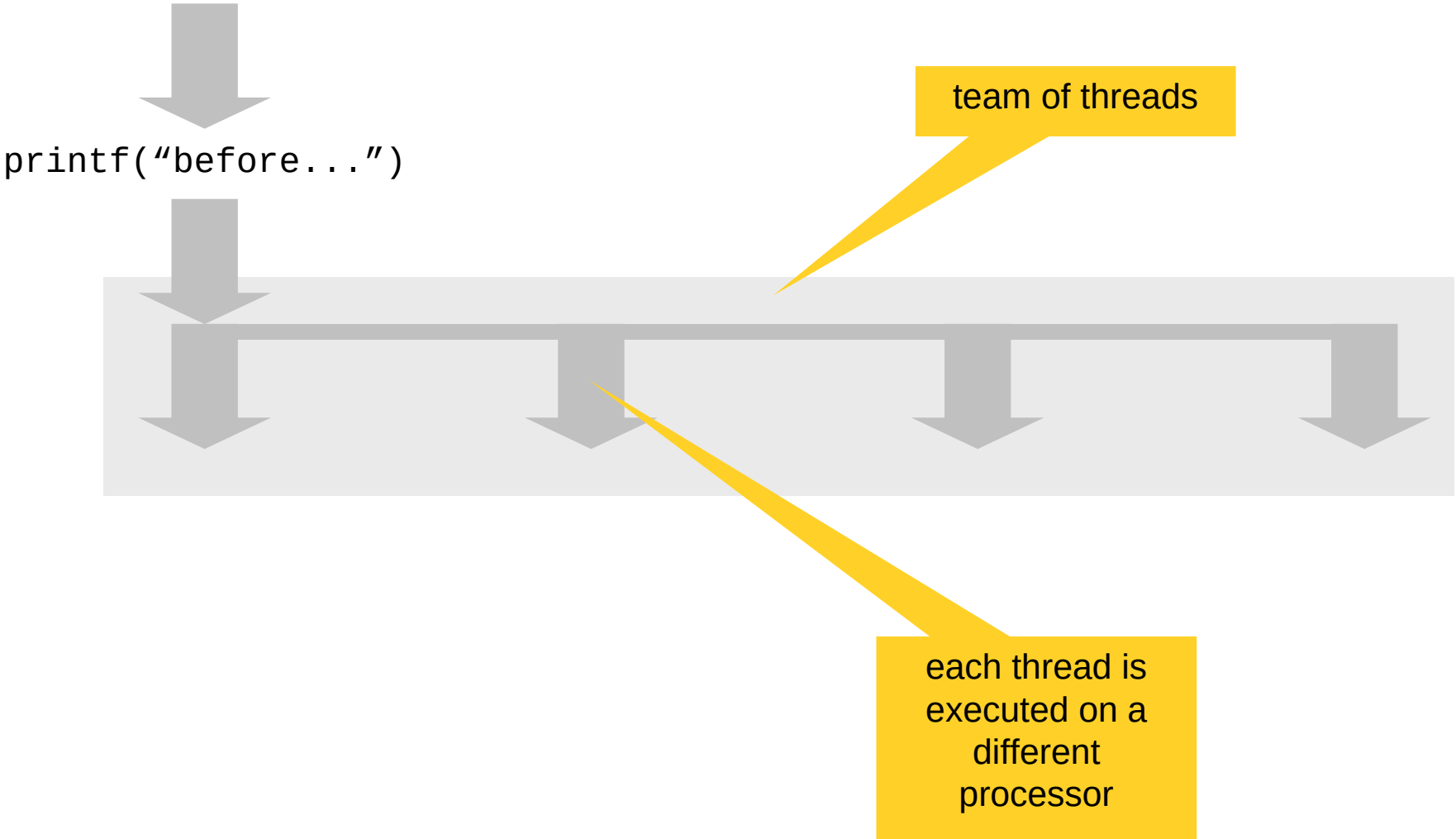


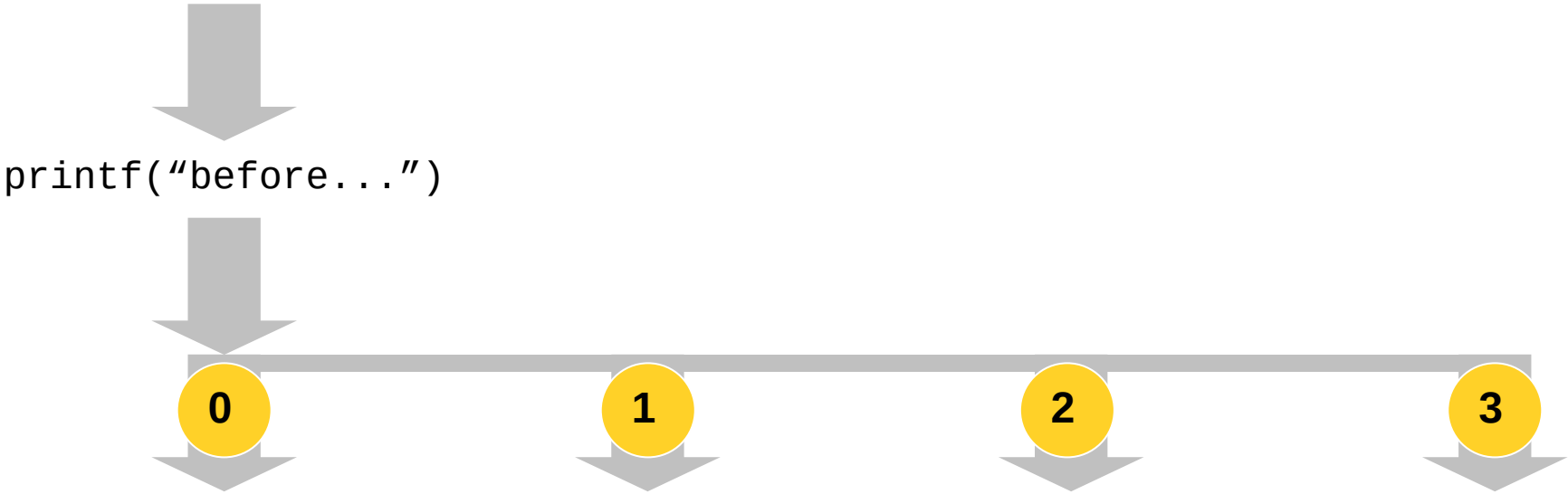


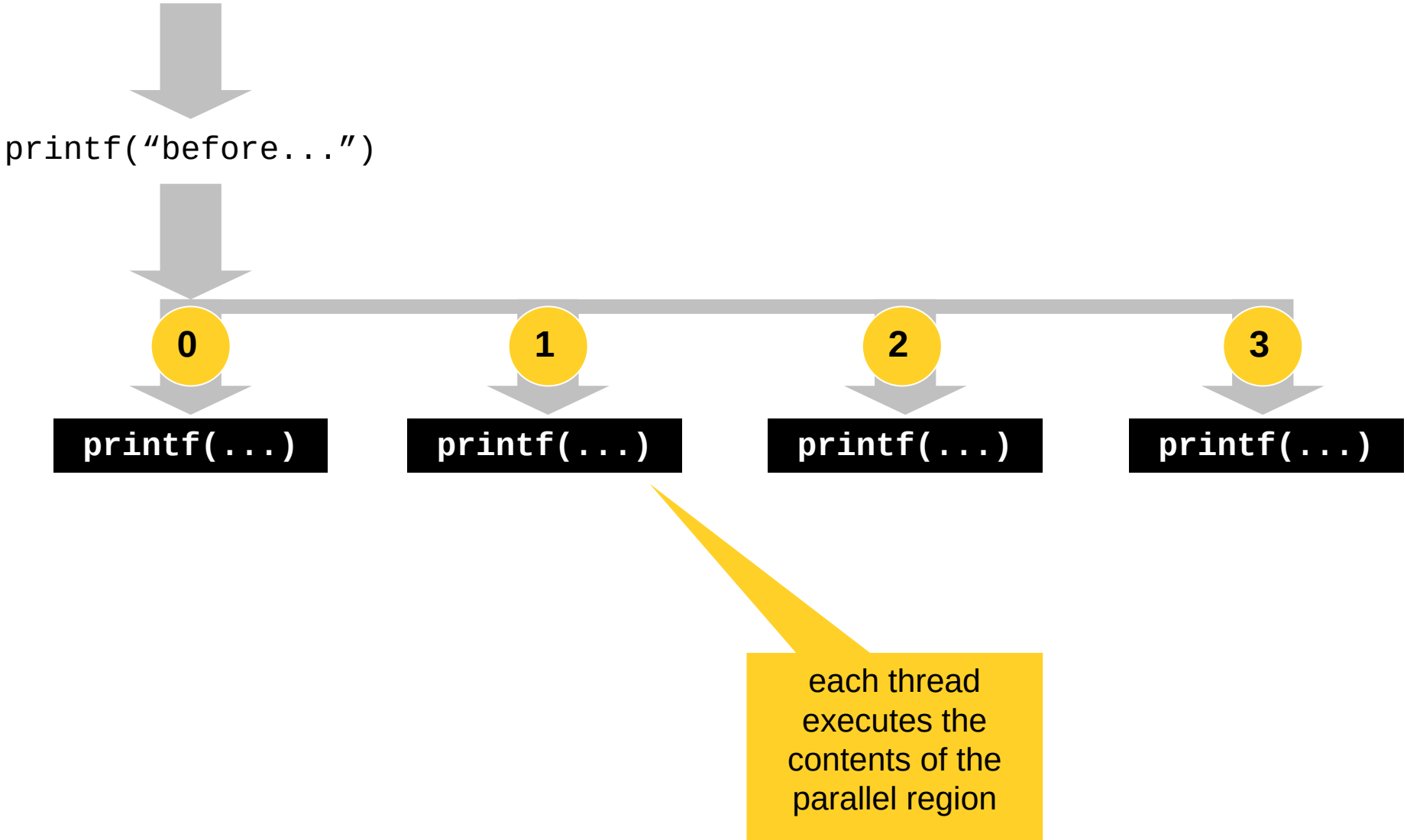
create delete

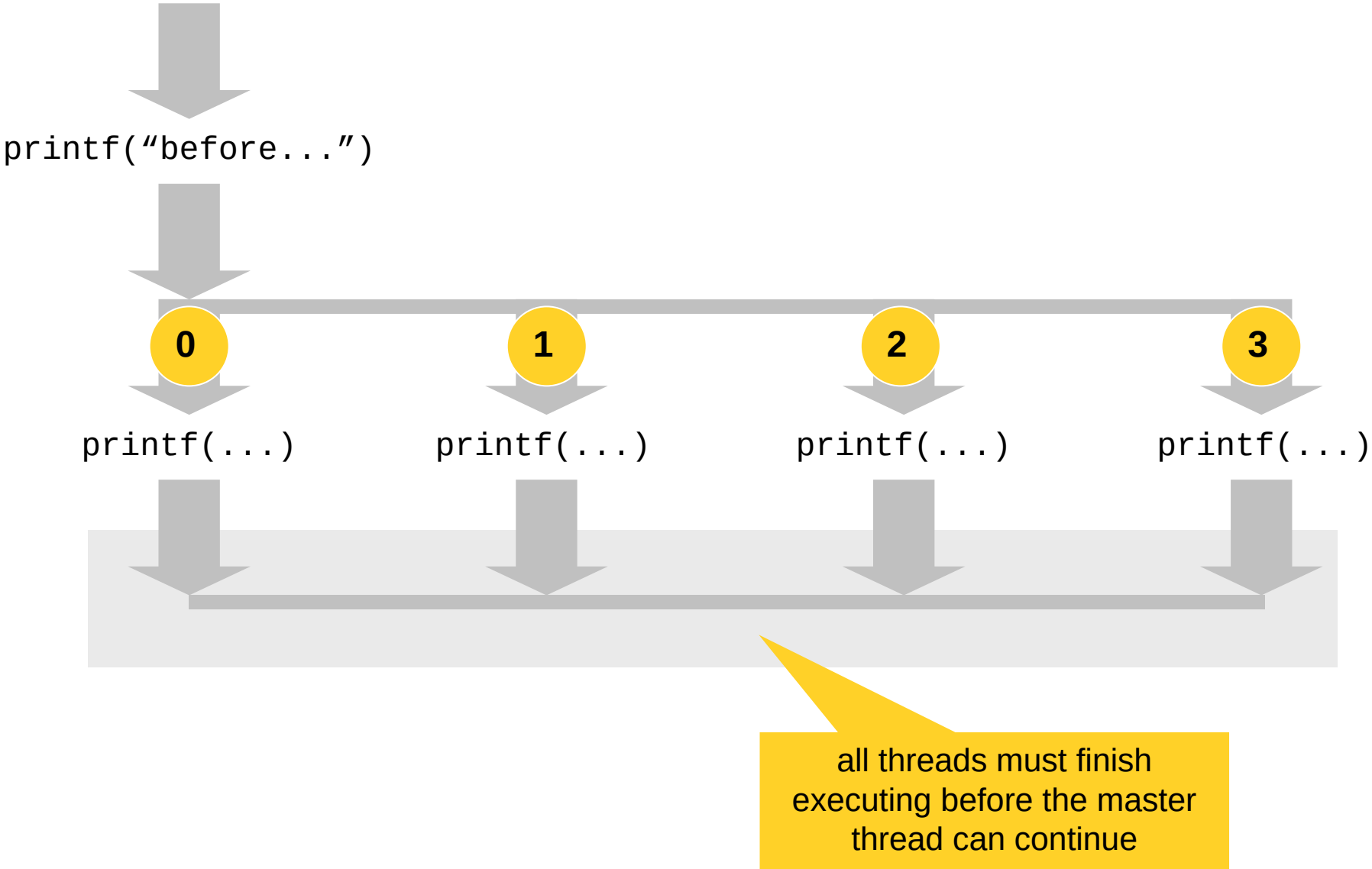
```
printf("before...")
```

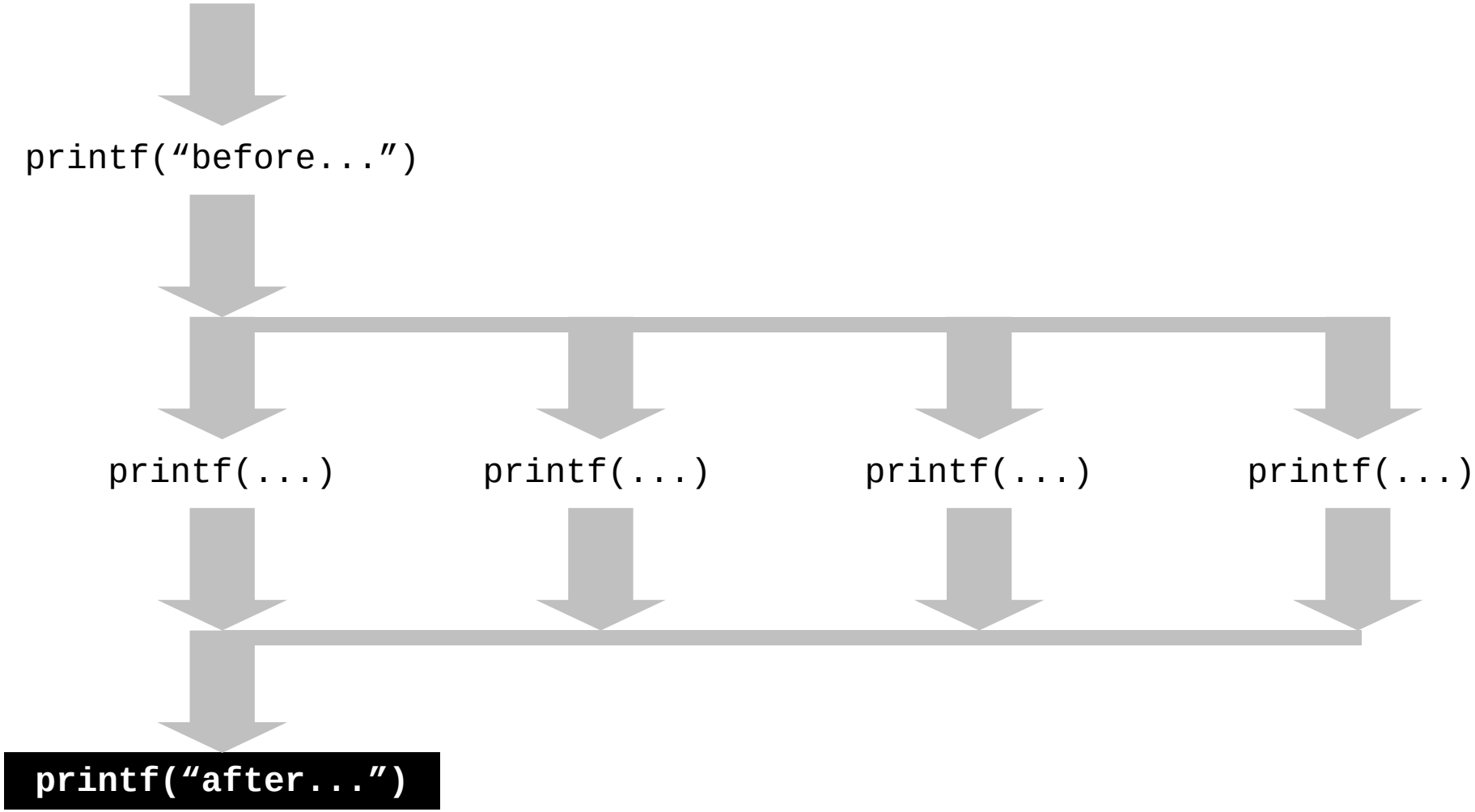


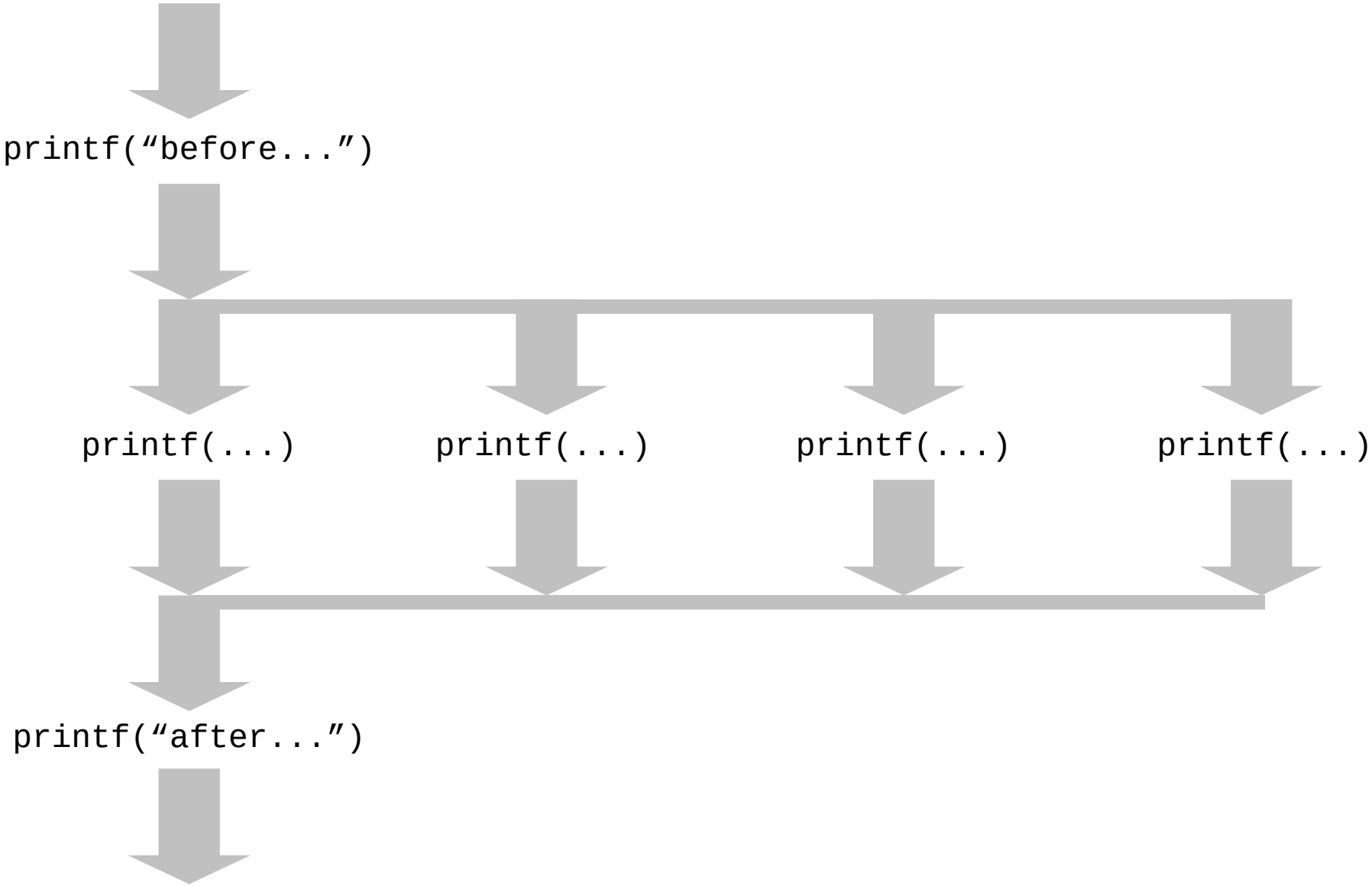












- Compiling…
  - SGI
    - Use the **`-mp`** switch
  - IBM
    - Use the **`_r`** compiler variant
    - Use the **`-qsmp=omp`** switch
    - For Fortran 77, also use the **`-qnosave`** switch
- Running…
  - Set environment variable **`OMP_NUM_THREADS`**
  - Execute the binary

- Compiling and running…

```
#include <stdio.h>

int
main (void)
{
#pragma omp parallel
  {
    printf ("Hello, world\n");
  }
  return 0;
}
```

- Compiling and running…

```
esumbar@aurora: cc hello.c
esumbar@aurora: setenv OMP_NUM_THREADS 4
esumbar@aurora: ./a.out
Hello, world
```

enable OpenMP support

- Compiling and running…

```
esumbar@aurora: cc -mp hello.c
esumbar@aurora: setenv OMP_NUM_THREADS 4
esumbar@aurora: ./a.out
Hello, world
Hello, world
Hello, world
Hello, world
```

- Compiling and running…

```
esumbar@aurora: cc -mp hello.c
esumbar@aurora: setenv OMP_NUM_THREADS 4
esumbar@aurora: ./a.out
Hello, world
Hello, world
Hello, world
Hello, world
esumbar@aurora: setenv OMP_NUM_THREADS 2
esumbar@aurora: ./a.out
Hello, world
Hello, world
```

no need to recompile

- Compiling and running…

```
#include <stdio.h>

int main (void)
{
  printf (“before parallel region\n”);

#pragma omp parallel
  {
    printf (“Hello, world\n”);
  }

  printf (“after parallel region\n”);
  return 0;
}
```

- Compiling and running…

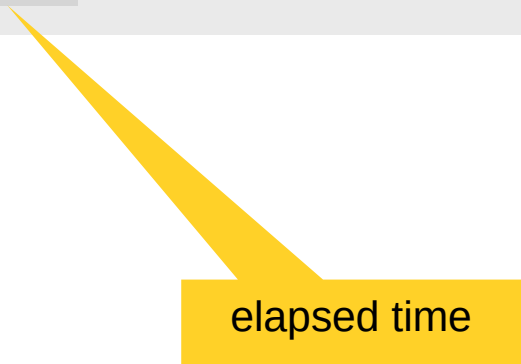
```
esumbar@aurora: cc -mp hello2.c
esumbar@aurora: setenv OMP_NUM_THREADS 4
esumbar@aurora: ./a.out
before parallel region
Hello, world
Hello, world
Hello, world
Hello, world
after parallel region
```

- Fundamental OpenMP concept
  **work sharing**

```
#include <stdio.h>
#include <math.h>

#define NX 10000000
#define NY 10000000
#define NZ 10000000

float x[NX], y[NY], z[NZ];

void work (float *a, int n)
{
  int i;
  for (i=0; i<n; i++)
    a[i] = sin((float)i);
  return;
}

int main (void)
{
  printf("start\n");

  work(x, NX);
  work(y, NY);
  work(z, NZ);

  printf("end\n");
  return 0;
}
```

- Work sharing…

```
esumbar@aurora: cc work.c
esumbar@aurora: time ./a.out
start
end
10.475u 0.907s 0:11.73 96.9% 0+0k 0+0io 0pf+0w
```



- Work sharing…

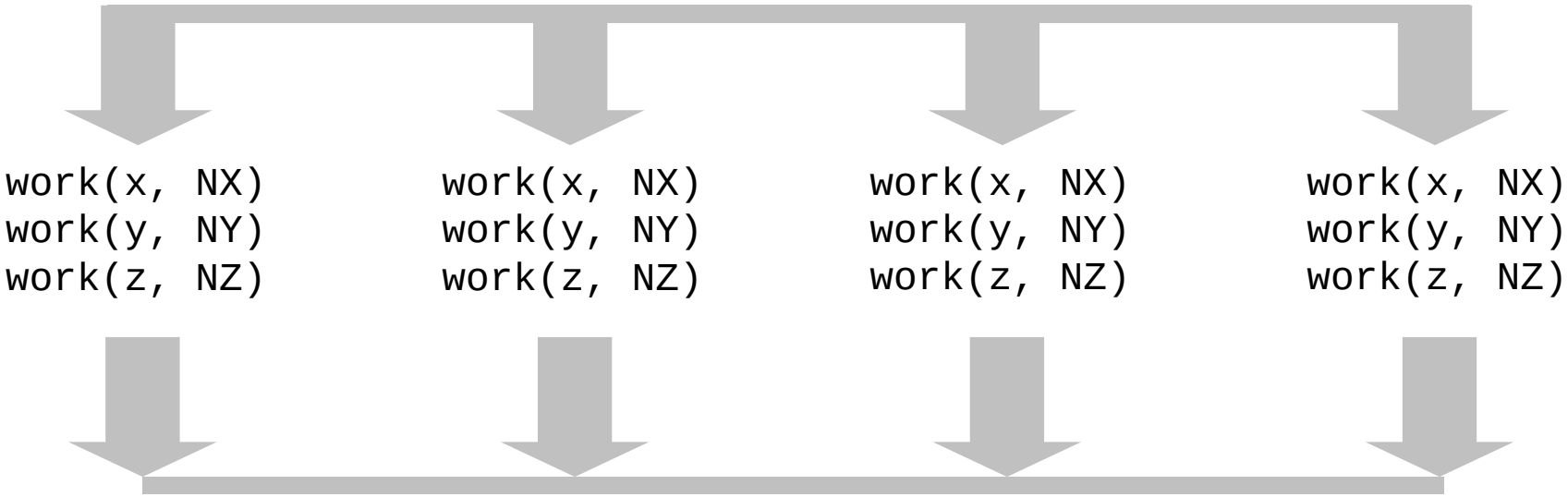
```
int main (void)
{
  printf("start\n");

  work(x, NX);
  work(y, NY);
  work(z, NZ);

  printf("end\n");
  return 0;
}
```

- Work sharing…

```
int main (void)
{
  printf("start\n");

  work(x, NX);
  work(y, NY);
  work(z, NZ);

  printf("end\n");
  return 0;
}
```

```
int main (void)
{
  printf("start\n");

#pragma omp parallel
  {
    work(x, NX);
    work(y, NY);
    work(z, NZ);
  }

  printf("end\n");
  return 0;
}
```

- Work sharing…



- Work sharing…

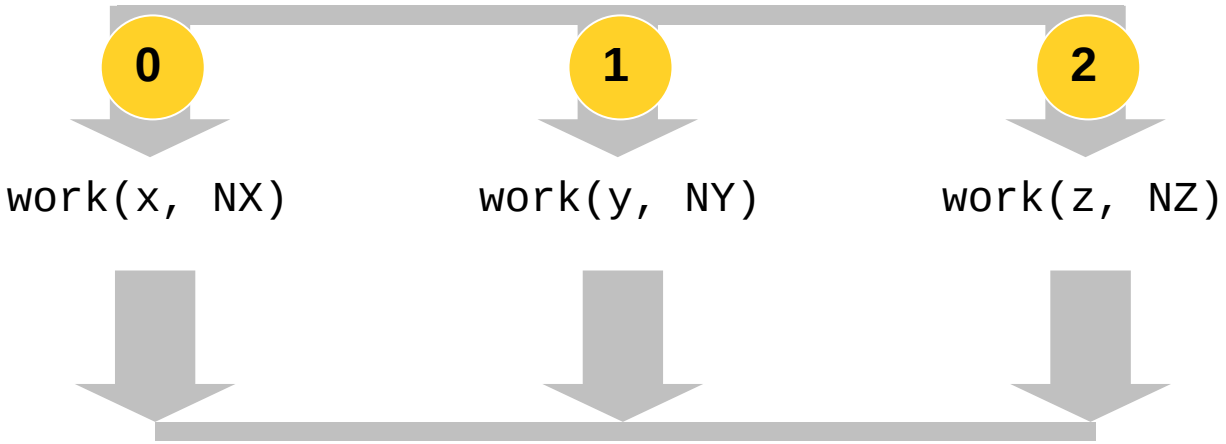
```
int main (void)
{
  printf("start\n");

#pragma omp parallel
  {
    work(x, NX);
    work(y, NY);
    work(z, NZ);
  }

  printf("end\n");
  return 0;
}
```

returns the id of the calling thread

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

- Work sharing…



- Work sharing…

```
esumbar@aurora: cc –mp work.c -lm
esumbar@aurora: setenv OMP_NUM_THREADS 3
esumbar@aurora: time ./a.out
start
end
10.733u 1.184s 0:04.20 283.5% 0+0k 1+4io 1pf+0w
```

- Work sharing…

$$\text{speedup} \equiv \frac{\text{elapsed time of serial program}}{\text{elapsed time of parallel program}}$$

$$= \frac{T_{serial}}{T_{parallel}} = \frac{11.73}{4.20} = 2.8$$

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

```
#pragma omp parallel
  {
    work(x, NX);
    work(y, NY);
    work(z, NZ);
  }
```

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

```
#pragma omp parallel
  {
    {
      work(x, NX);
      work(y, NY);
      work(z, NZ);
    }
  }
```

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

```
#pragma omp parallel
  {
#pragma omp sections
    {
      work(x, NX);
      work(y, NY);
      work(z, NZ);
    }
  }
```

work sharing directive

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

```
#pragma omp parallel
  {
#pragma omp sections
    {
      work(x, NX);
      work(y, NY);
      work(z, NZ);
    }
  }
```




create delete

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

```
#pragma omp parallel
  {
#pragma omp sections
    {
#pragma omp section
      work(x, NX);
      work(y, NY);
      work(z, NZ);
    }
  }
```

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

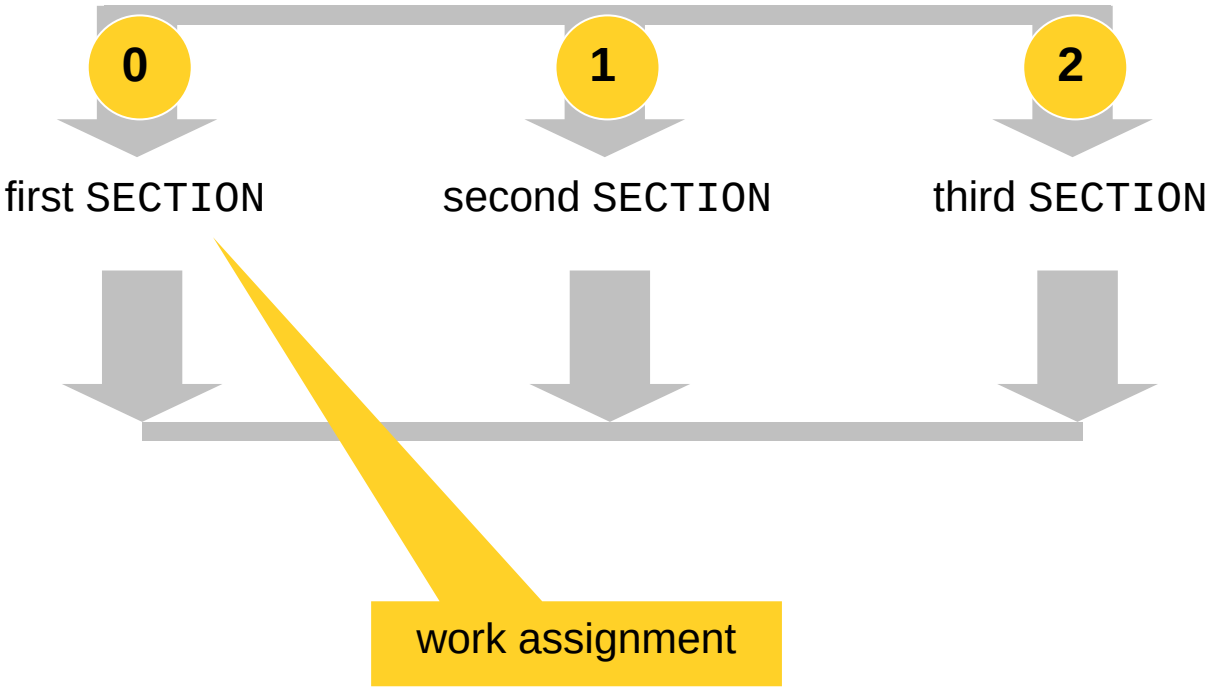
```
#pragma omp parallel
  {
#pragma omp sections
    {
#pragma omp section
      work(x, NX);
#pragma omp section
      work(y, NY);
      work(z, NZ);
    }
  }
```

- Work sharing…

```
#pragma omp parallel
  {
    switch ( omp_get_thread_num() )
      {
      case 0:
        work(x, NX);
        break;
      case 1:
        work(y, NY);
        break;
      case 2:
        work(z, NZ);
        break;
      }
  }
```

```
#pragma omp parallel
  {
#pragma omp sections
    {
#pragma omp section
      work(x, NX);
#pragma omp section
      work(y, NY);
#pragma omp section
      work(z, NZ);
    }
  }
```

- Work sharing…



- Work sharing…
  - **`OMP SECTIONS`** appropriate

    number of tasks ≤ number of threads

- More work sharing…

```
for (i = 0; i < N; i++)
  task(i);
```

- More work sharing…

```
for (i = 0; i < N; i++)
  task(i);
```

```
task(0);
task(1);
task(2);
.
.
.
task(N-1);
```

- More work sharing…

```
for (i = 0; i < N; i++)
  task(i);
```

N >> number of threads

```
task(0);
task(1);
task(2);
.
.
.
task(N-1);
```

- More work sharing…
  - **`OMP SECTIONS`** is inapproriate
  - **`OMP FOR`** (for C loops)

    **`OMP DO`** (for Fortran loops)

    where number of tasks (iterations) >> number of threads

- More work sharing…

```
for (i = 0; i < N; i++)
  task(i);
```

- More work sharing…

```
{
  for (i = 0; i < N; i++)
    task(i);
}
```

- More work sharing…

```
#pragma omp parallel
  {
    for (i = 0; i < N; i++)
      task(i);
  }
```

- More work sharing…

```
#pragma omp parallel
  {
#pragma omp for schedule(static)
    for (i = 0; i < N; i++)
      task(i);
  }
```

- More work sharing...

work sharing directive specifies how iterations are assigned

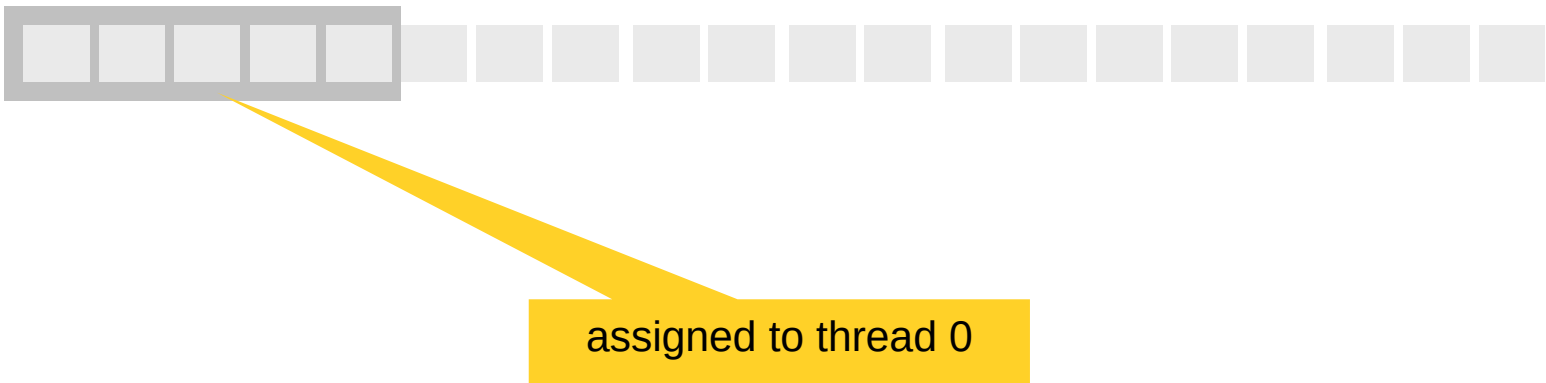
```
#pragma omp parallel
  {
#pragma omp for schedule(static)
    for (i = 0; i < N; i++)
      task(i);
  }
```

- More work sharing…
  - Many ways to schedule (assign) loop iterations to threads

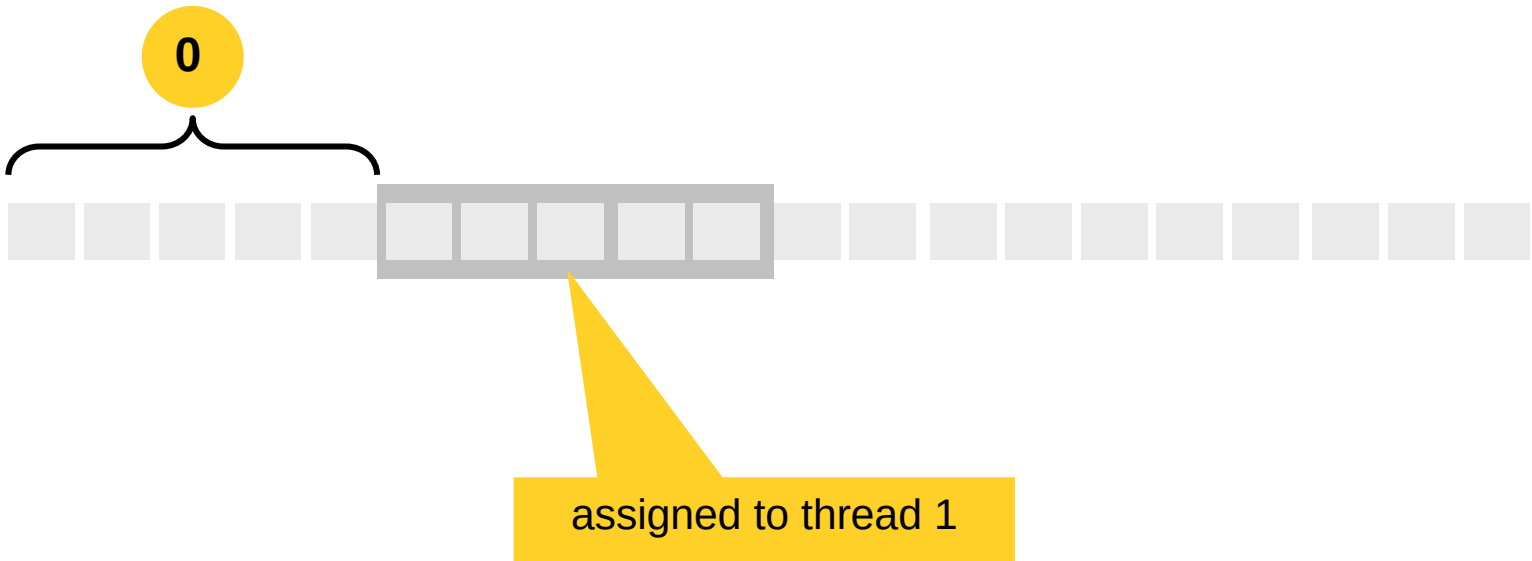
- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign contiguous chunks (blocks) of iterations of equal size

- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign contiguous chunks (blocks) of iterations of equal size

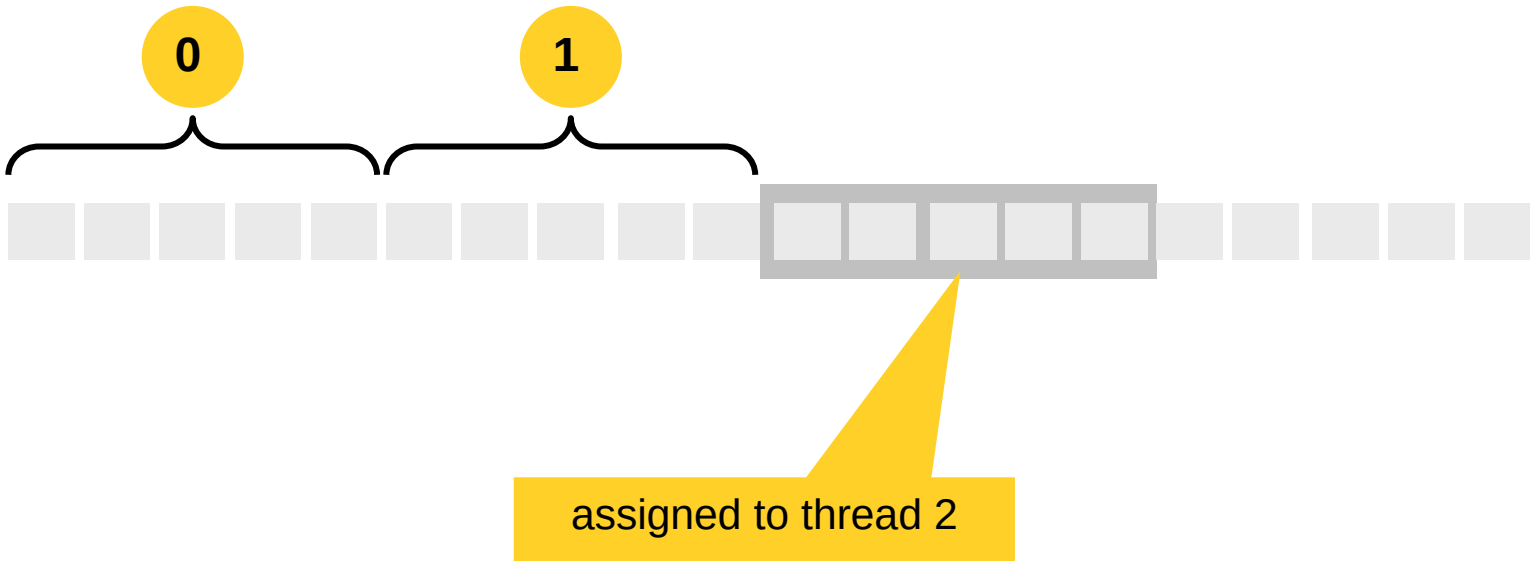
- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign contiguous chunks (blocks) of iterations of equal size



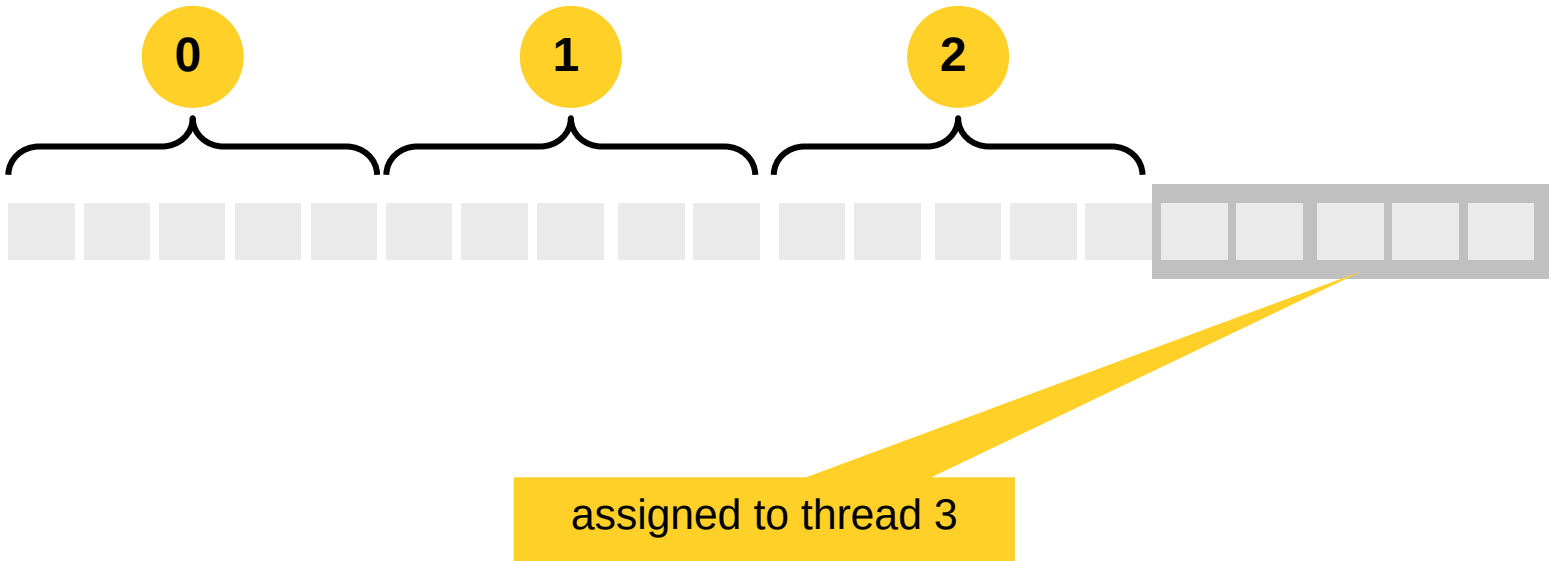
- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign contiguous chunks (blocks) of iterations of equal size



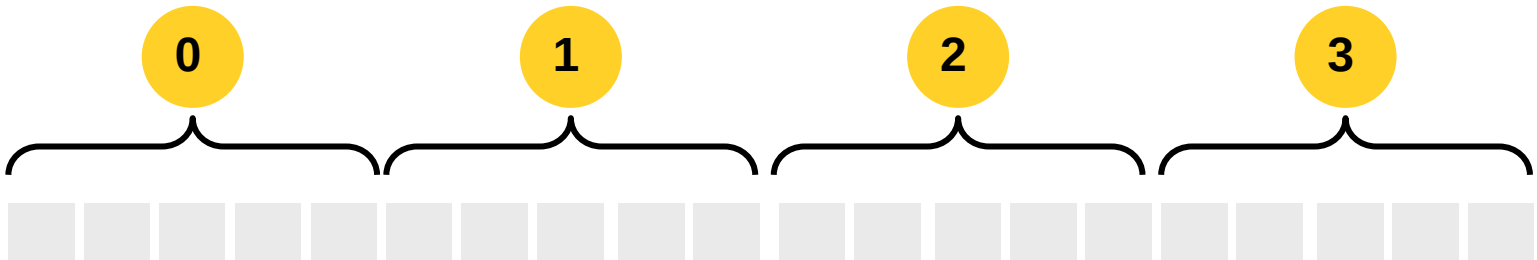
- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign contiguous chunks (blocks) of iterations of equal size



- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign contiguous chunks (blocks) of iterations of equal size



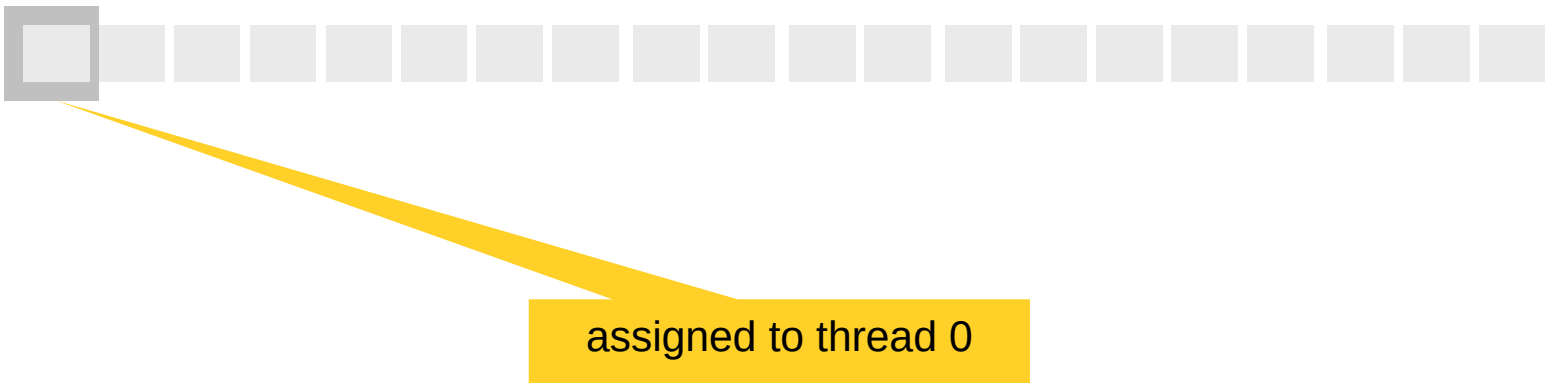
- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign contiguous chunks (blocks) of iterations of equal size



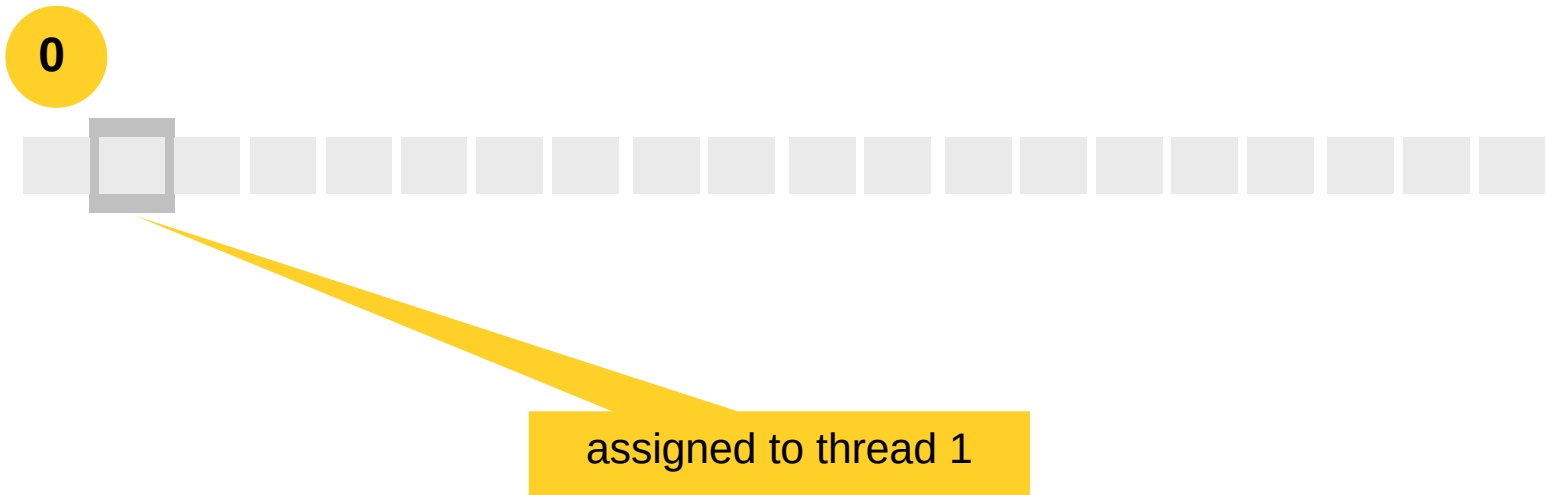
- More work sharing…
  - **`DYNAMIC`** loop scheduling
    - Assign consecutive iterations

- More work sharing…
  - **`DYNAMIC`** loop scheduling
    - Assign consecutive iterations

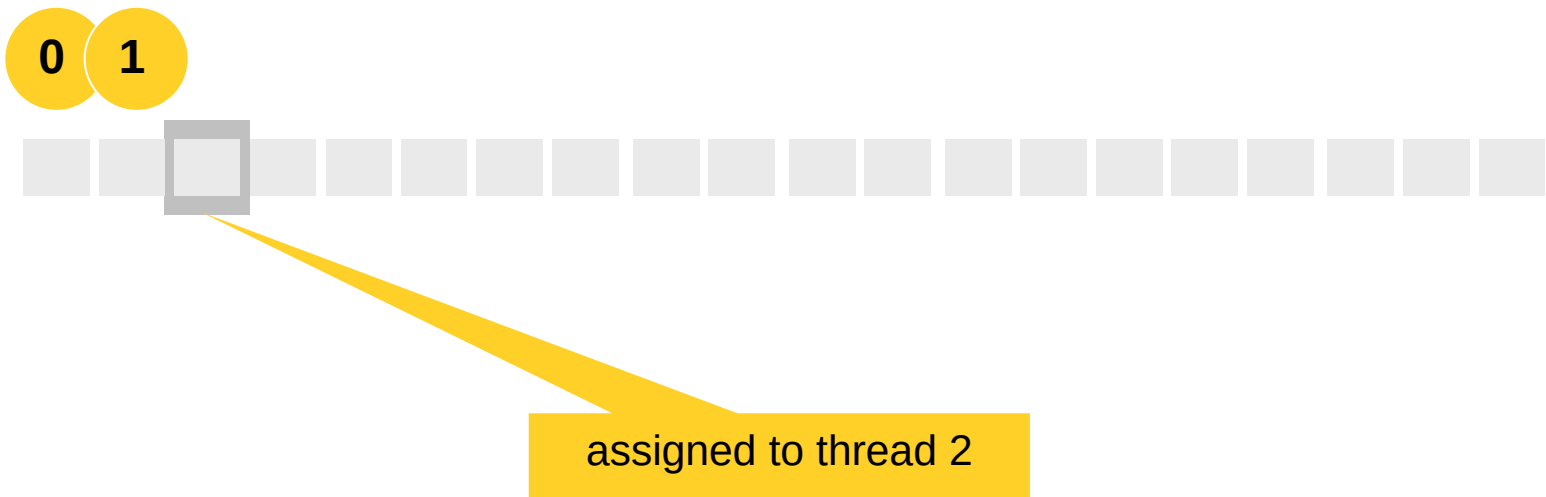
- More work sharing…
  - **`DYNAMIC`** loop scheduling
    - Assign consecutive iterations



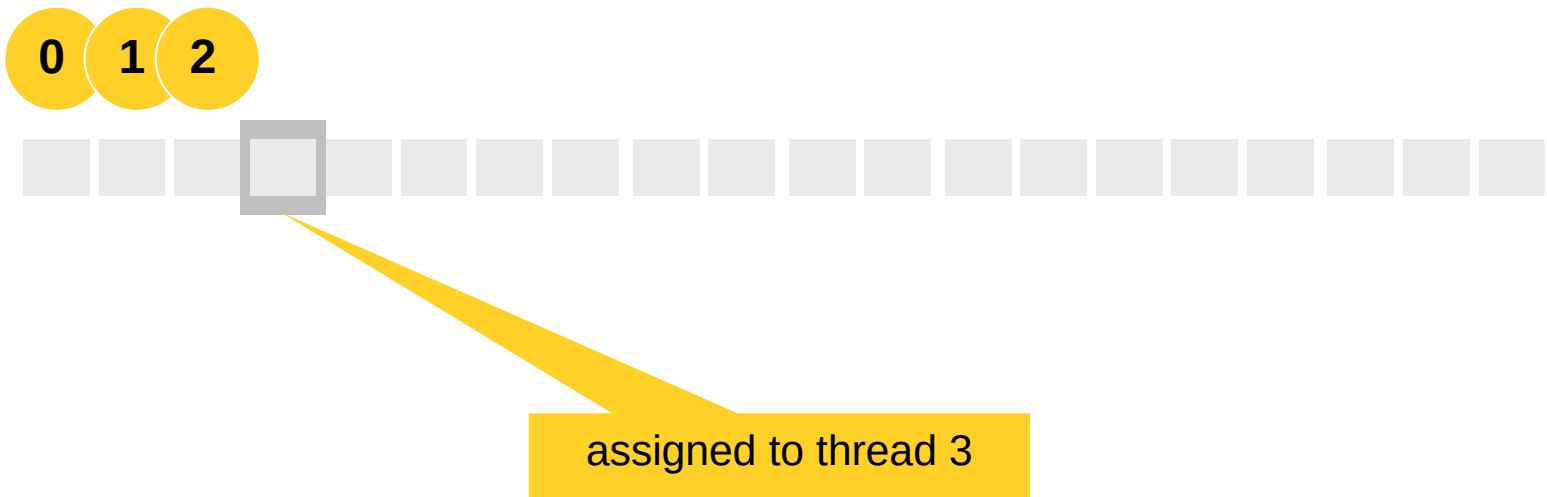
- More work sharing…
  - **`DYNAMIC`** loop scheduling
    - Assign consecutive iterations



- More work sharing…
  - **`DYNAMIC`** loop scheduling
    - Assign consecutive iterations



- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign consecutive iterations



- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign consecutive iterations

- More work sharing…
  - **`STATIC`** loop scheduling
    - Assign consecutive iterations

- Fundamental OpenMP concept

  **data scope**

- Data scope...

```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

  for (i=0; i<NX; i++)
    isum = isum + ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```

- Data scope...

```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

  for (i=0; i<NX; i++)
    isum = isum + ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```

```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

#pragma omp parallel for schedule(static)
  for (i=0; i<NX; i++)
    isum = isum + ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```

- Data scope…

```
esumbar@aurora: cc sum.c
esumbar@aurora: ./a.out
sum = 120000000
esumbar@aurora: cc -mp sum.c
"sum.c", line 15: Warning: Referenced scalar variable
     isum is SHARED by default
esumbar@aurora: setenv OMP_NUM_THREADS 2
esumbar@aurora: ./a.out
sum = 64134822
esumbar@aurora: setenv OMP_NUM_THREADS 4
esumbar@aurora: ./a.out
sum = 36490084
esumbar@aurora: ./a.out
sum = 36932008
```

- Data scope...

```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

#pragma omp parallel for schedule(static)
  for (i=0; i<NX; i++)
    isum = isum + ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```

- Data scope...

```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

#pragma omp parallel for schedule(static)
  for (i=0; i<NX; i++)
    isum = isum + ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```

global data

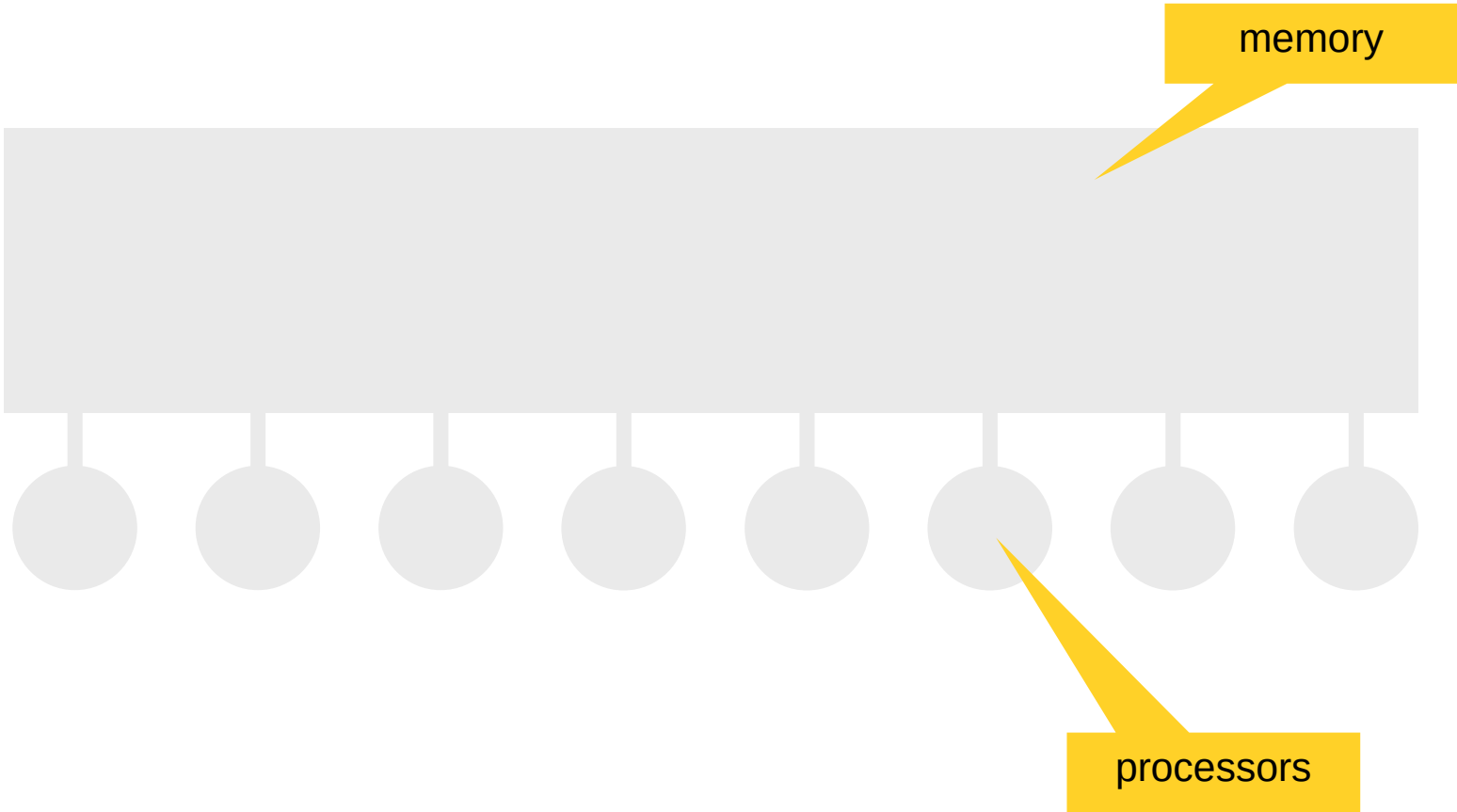
- Data scope...

```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

#pragma omp parallel for schedule(static)
  for (i=0; i<NX; i++)
    isum = isum + ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```

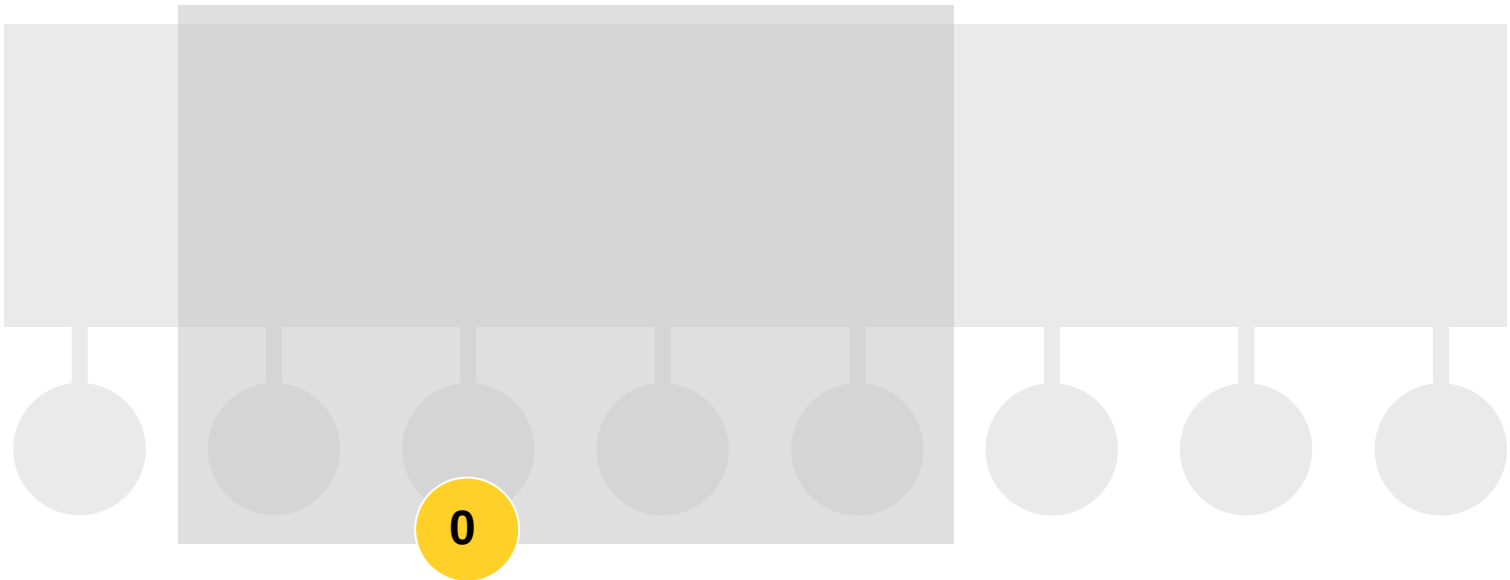
global data

local (automatic) data

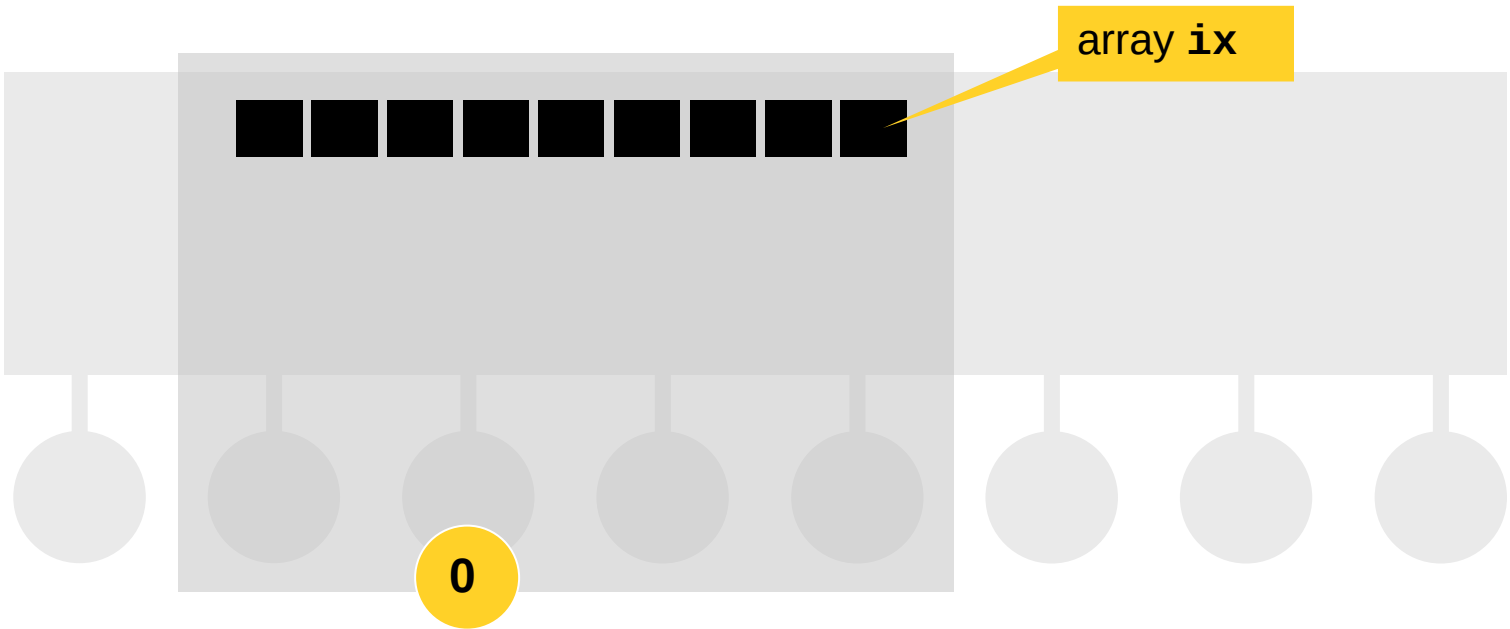
- Data scope...



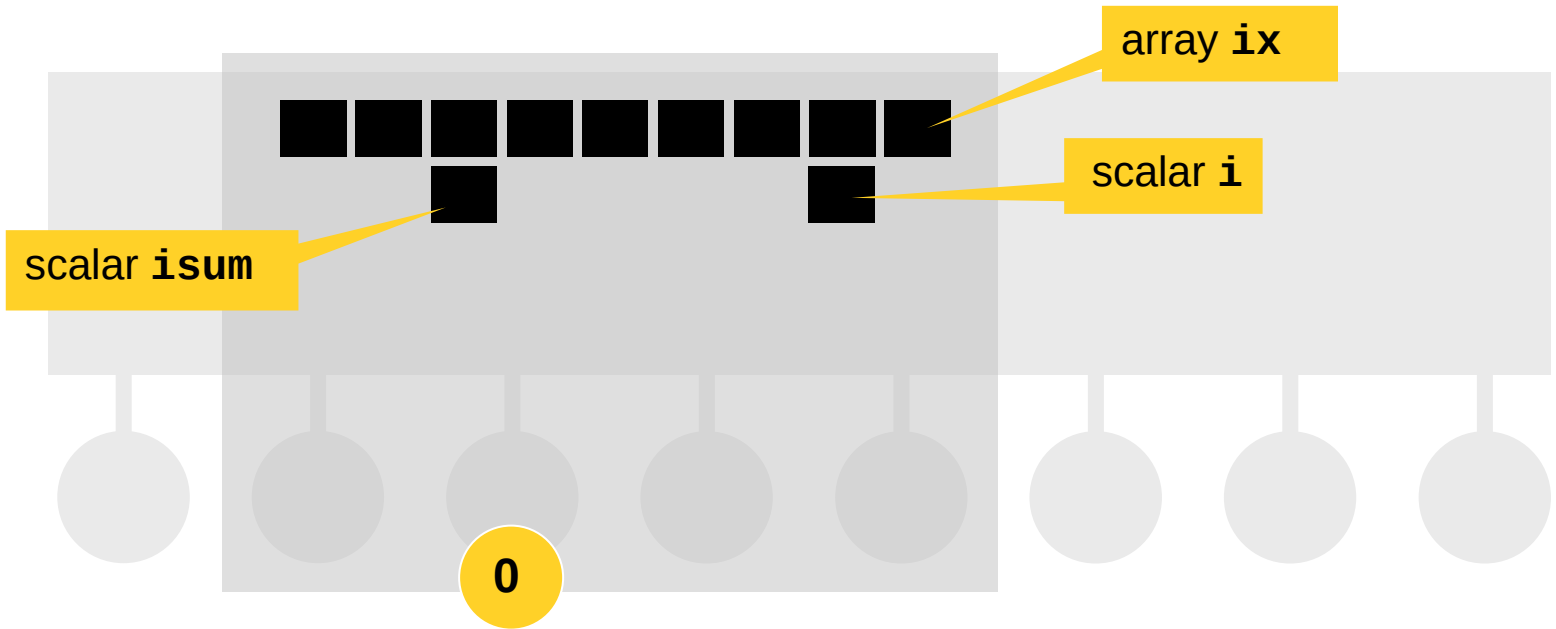
- Data scope…



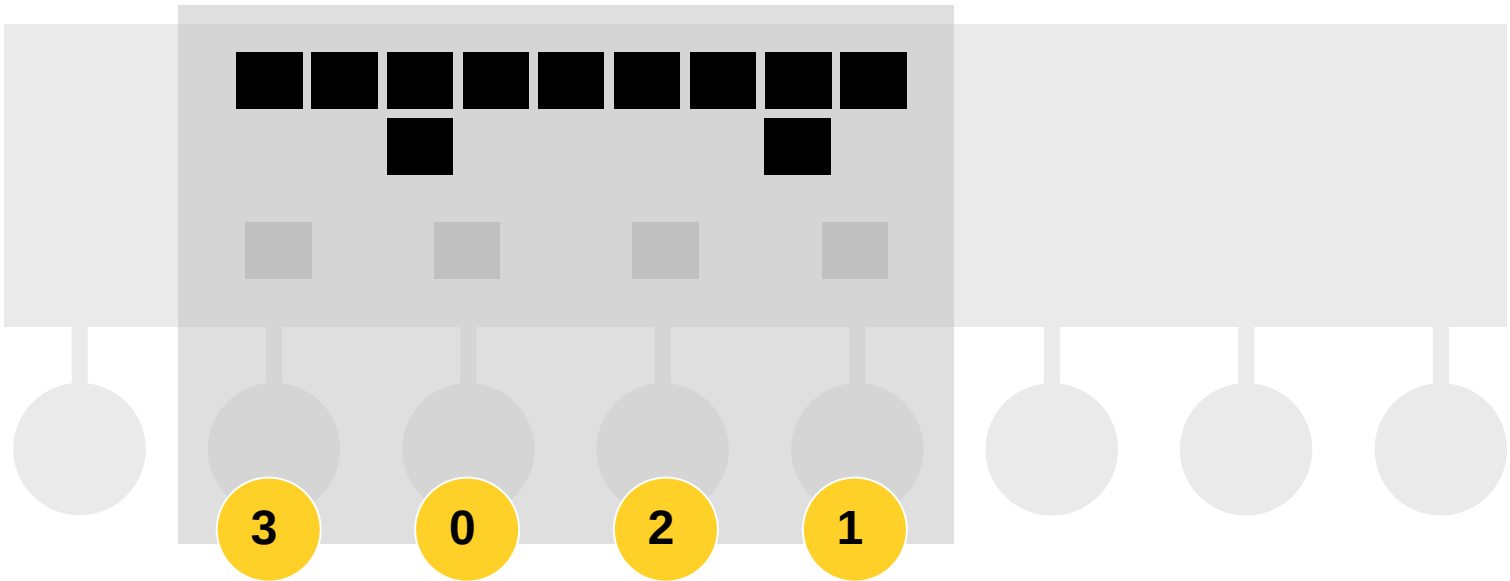
- Data scope...



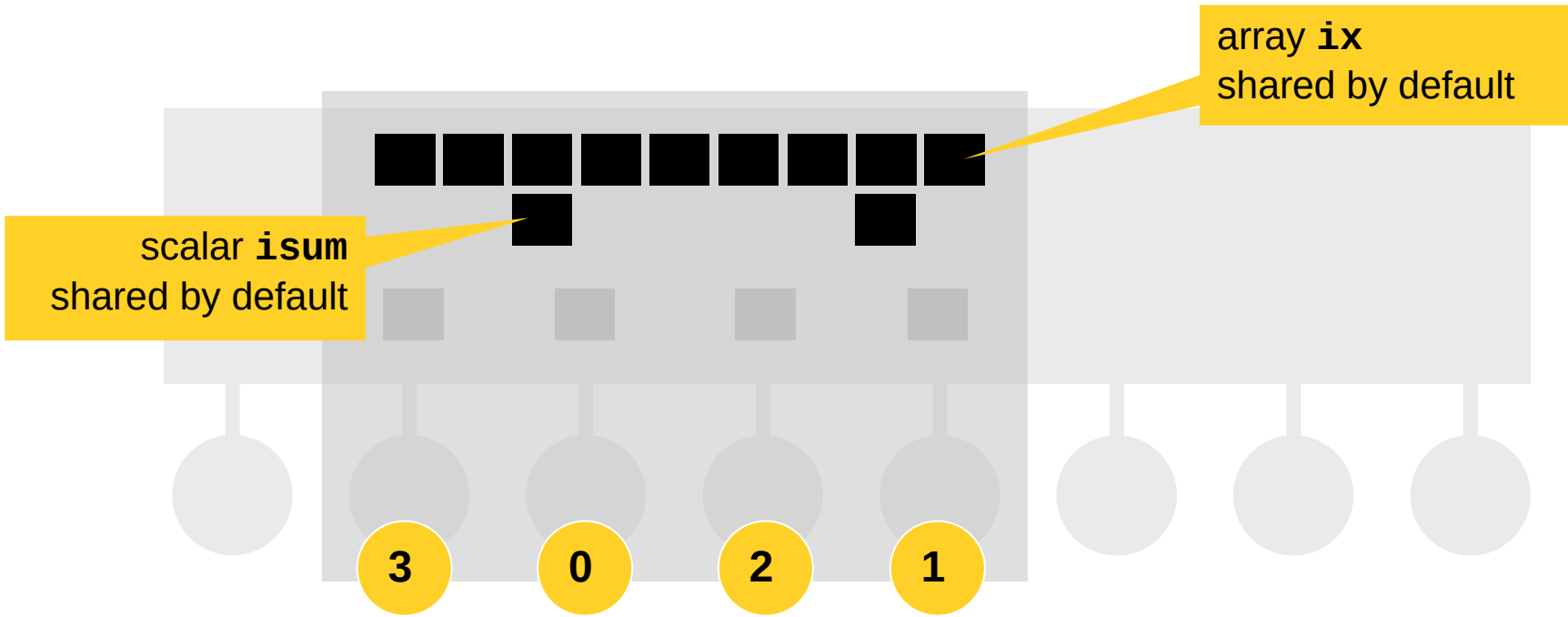
- Data scope...



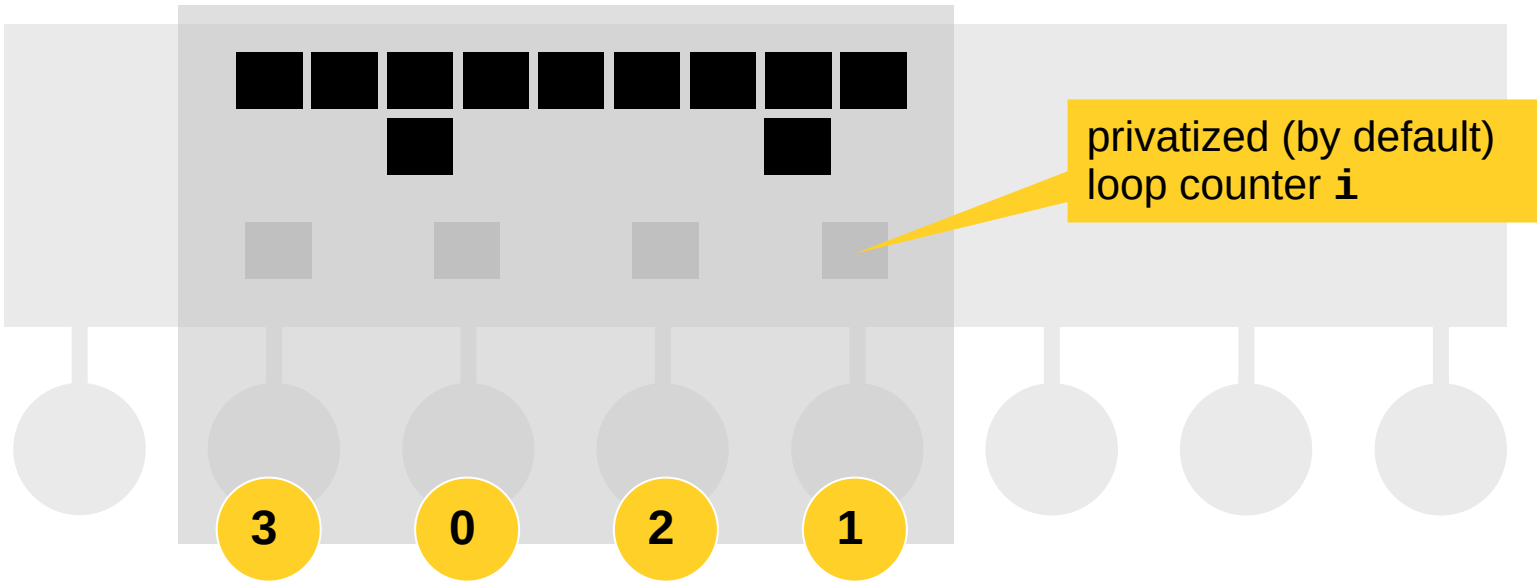
- Data scope...



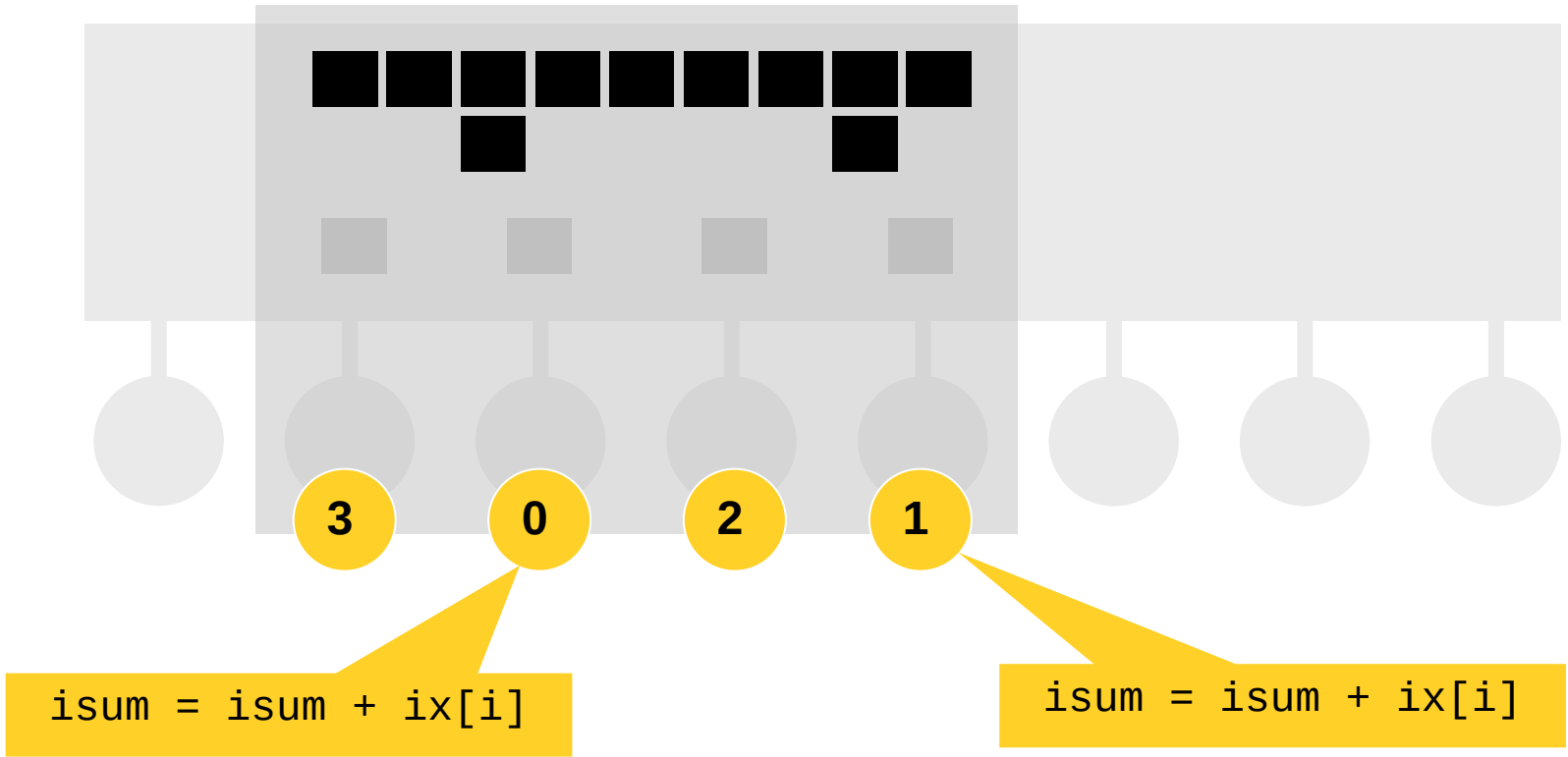
- Data scope...



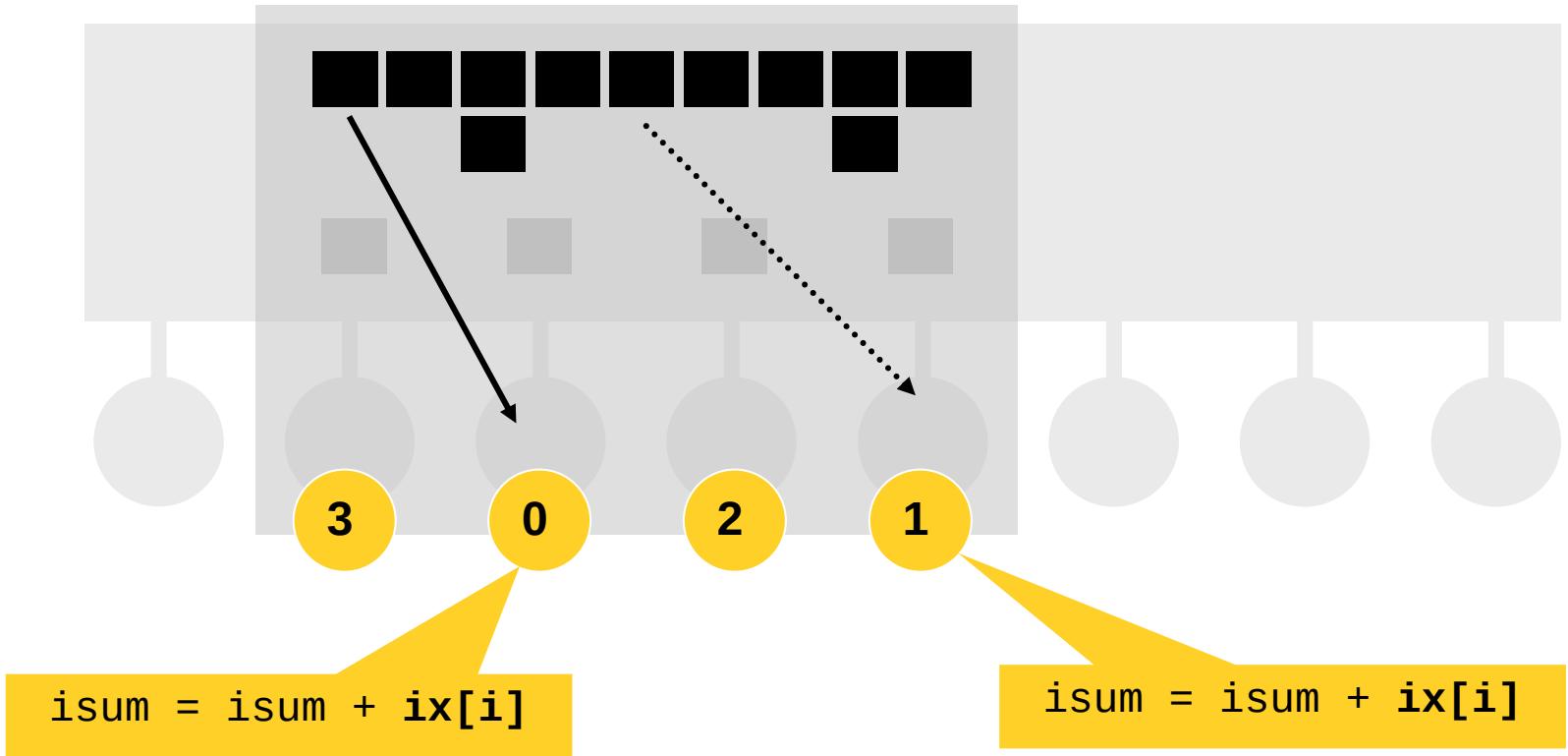
- Data scope...



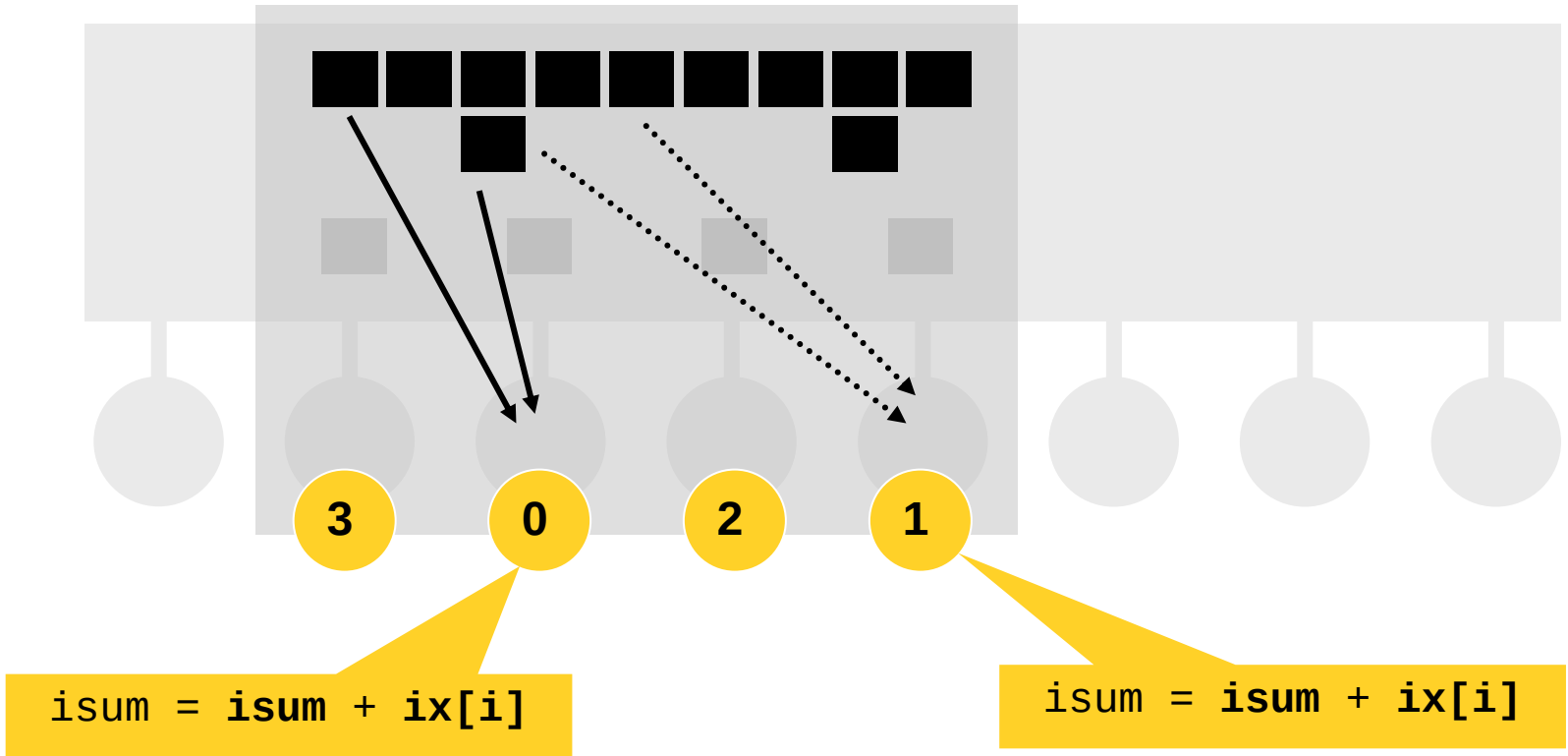
- Data scope...



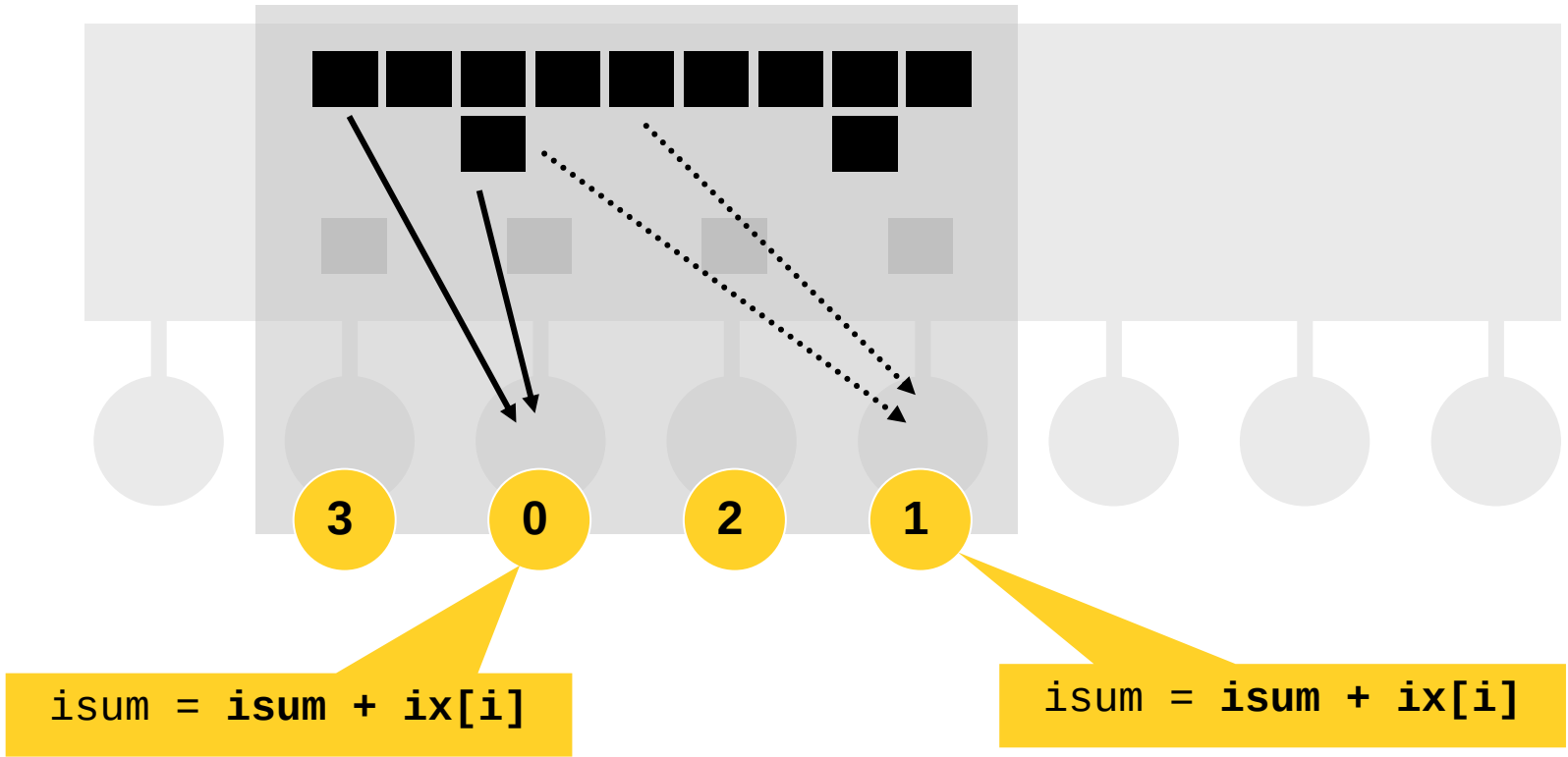
- Data scope...



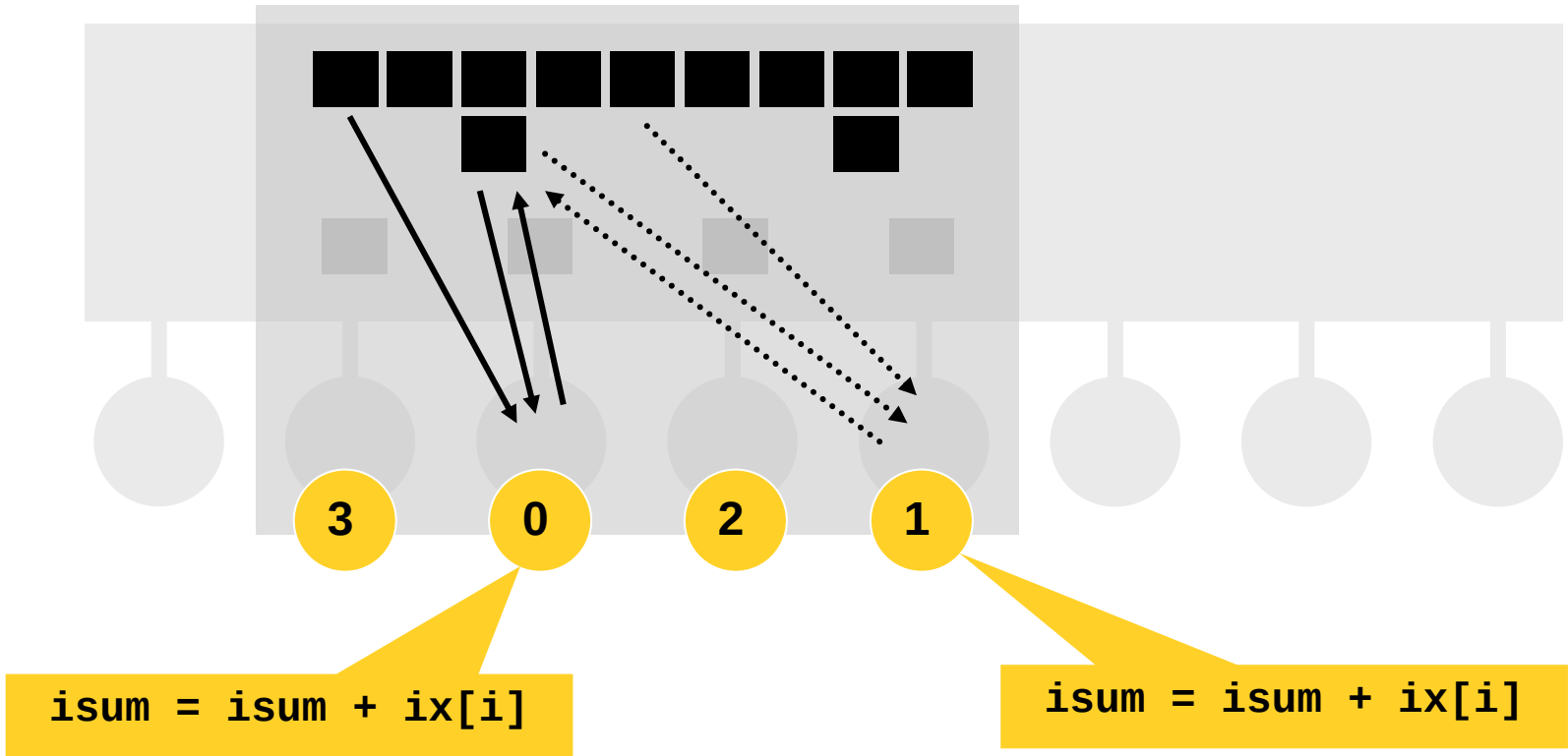
- Data scope...



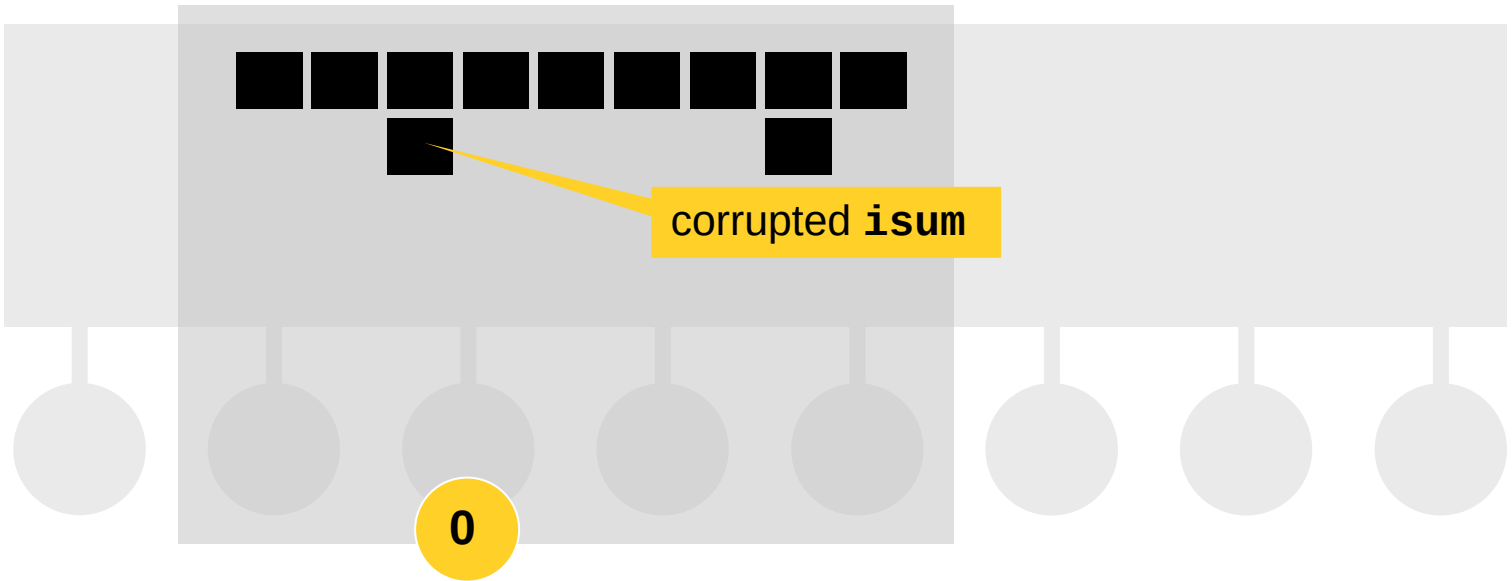
- Data scope...



- Data scope...

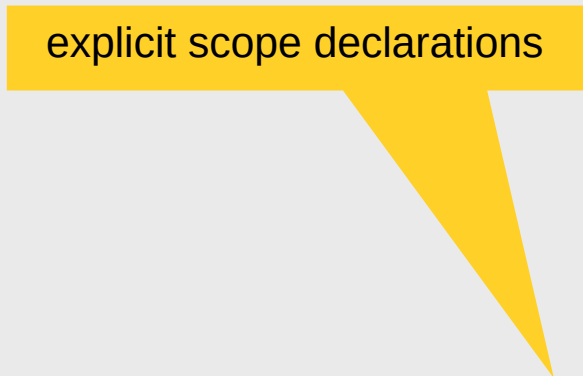


- Data scope...

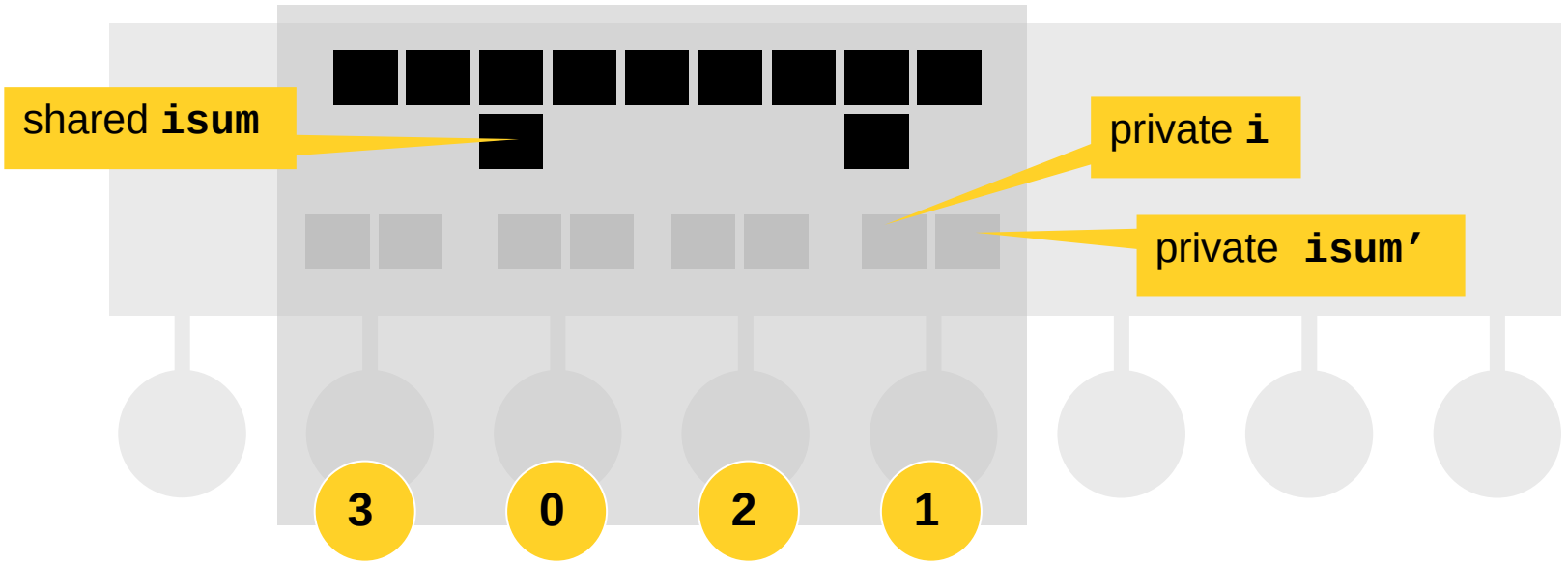


- Data scope...

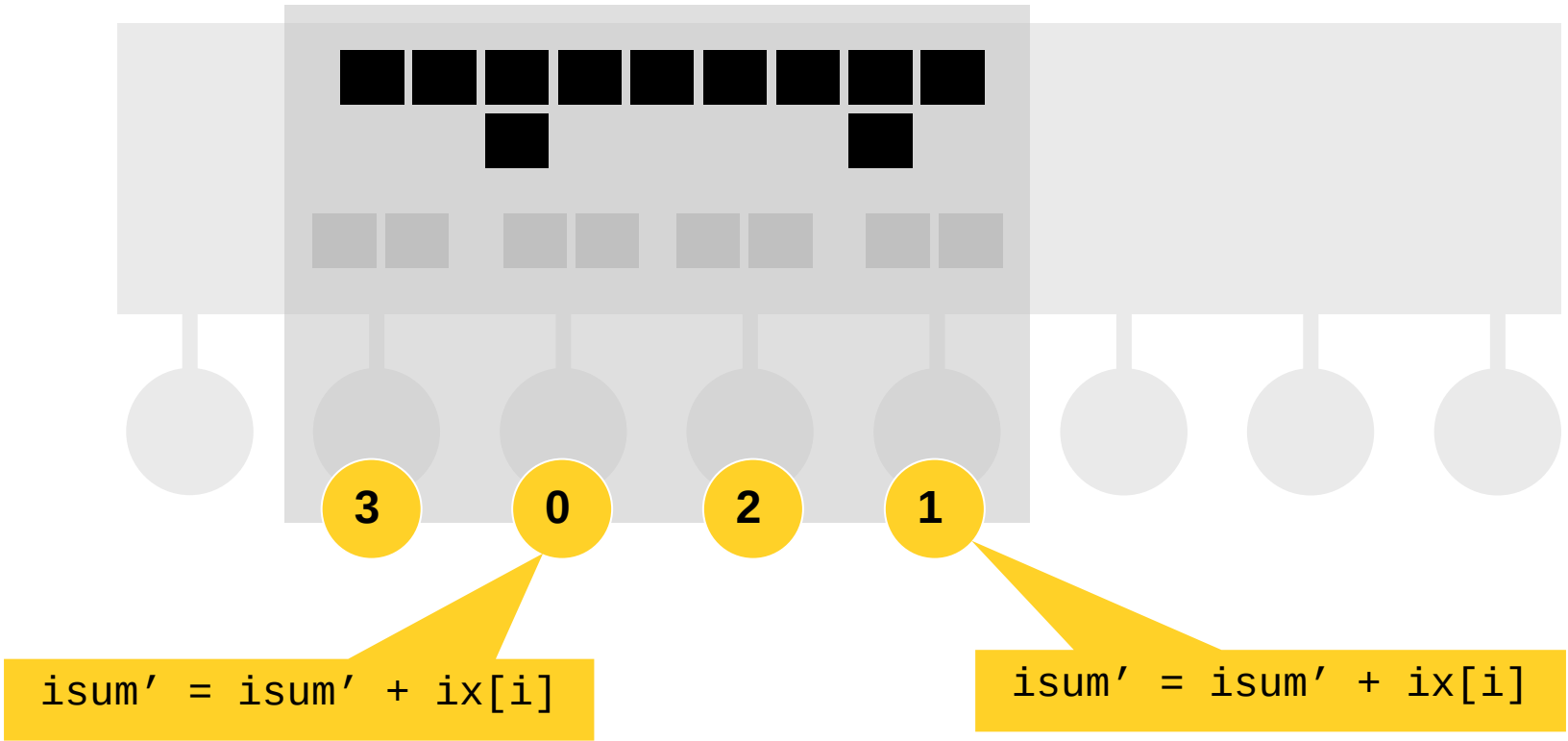
```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

#pragma omp parallel for schedule(static) private(i) shared(ix) reduction(+: isum)
  for (i=0; i<NX; i++)
    isum = isum + ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```



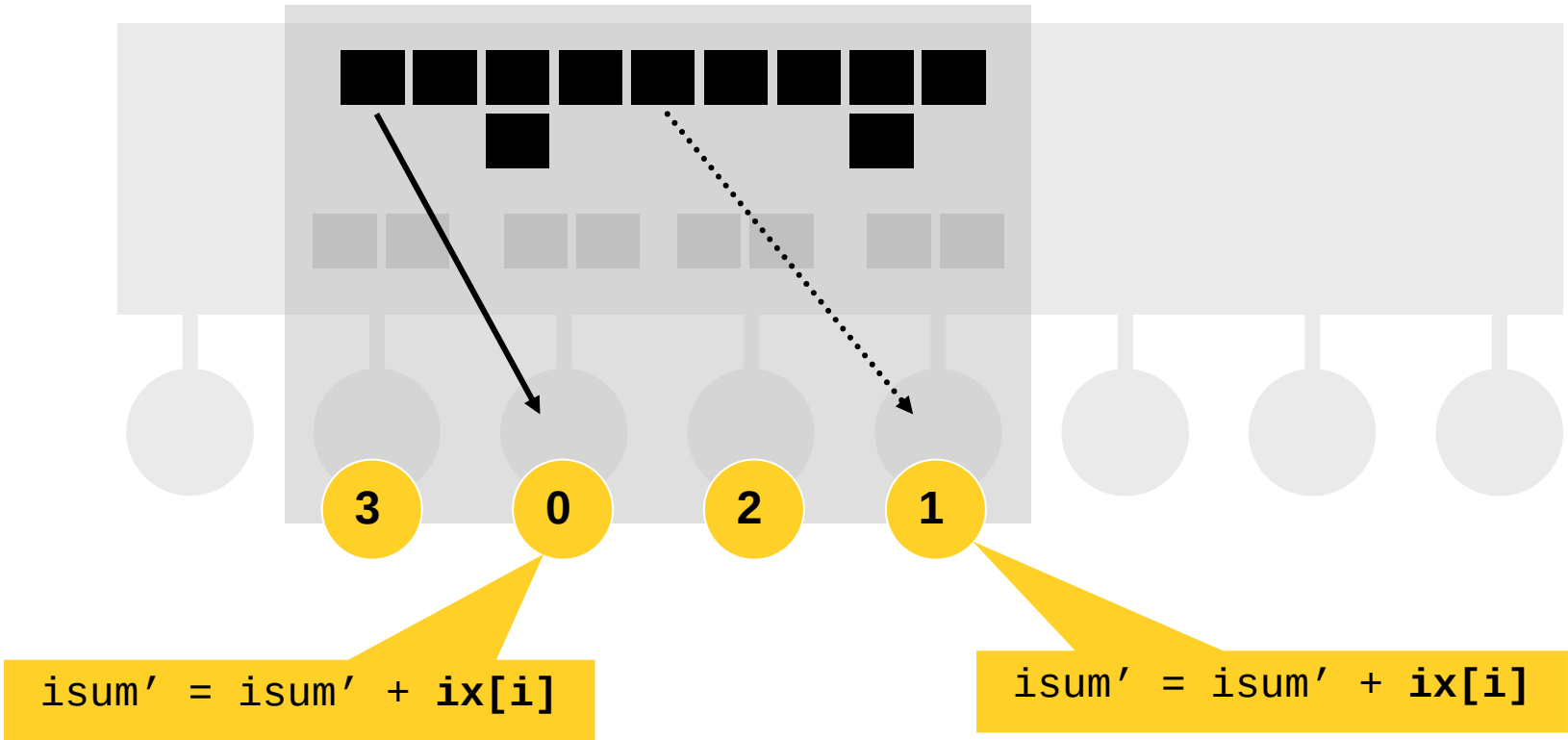
- Data scope...



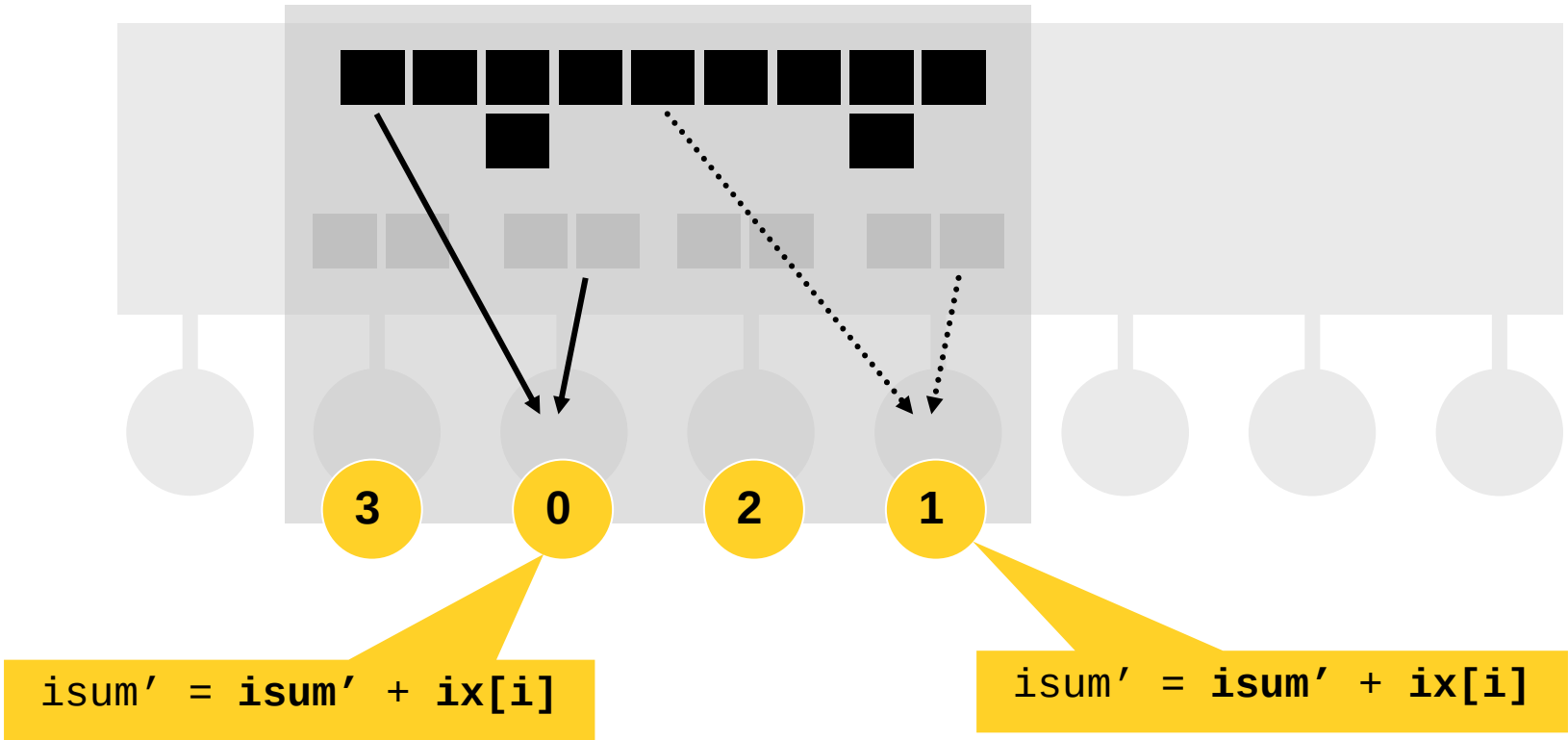
- Data scope...



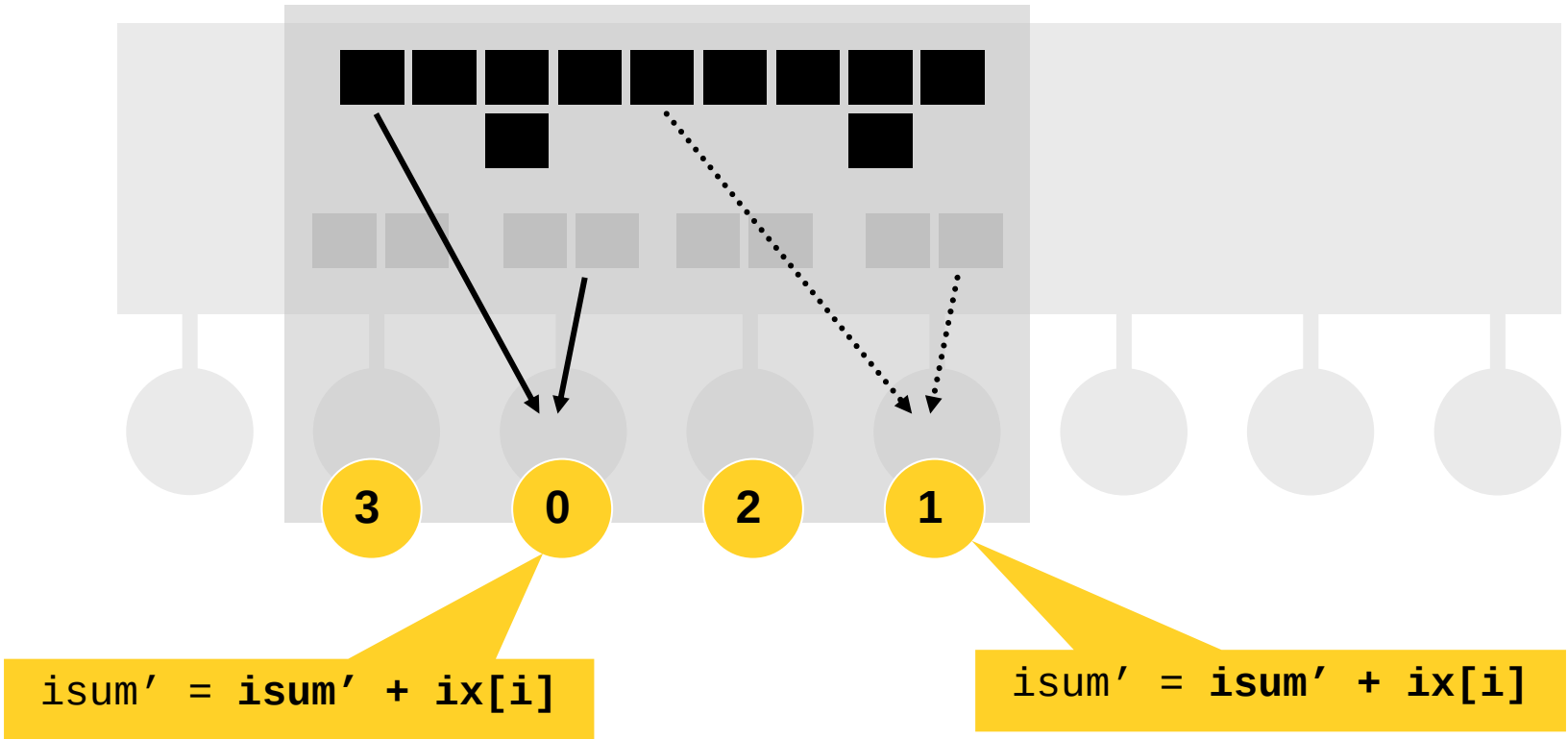
- Data scope...



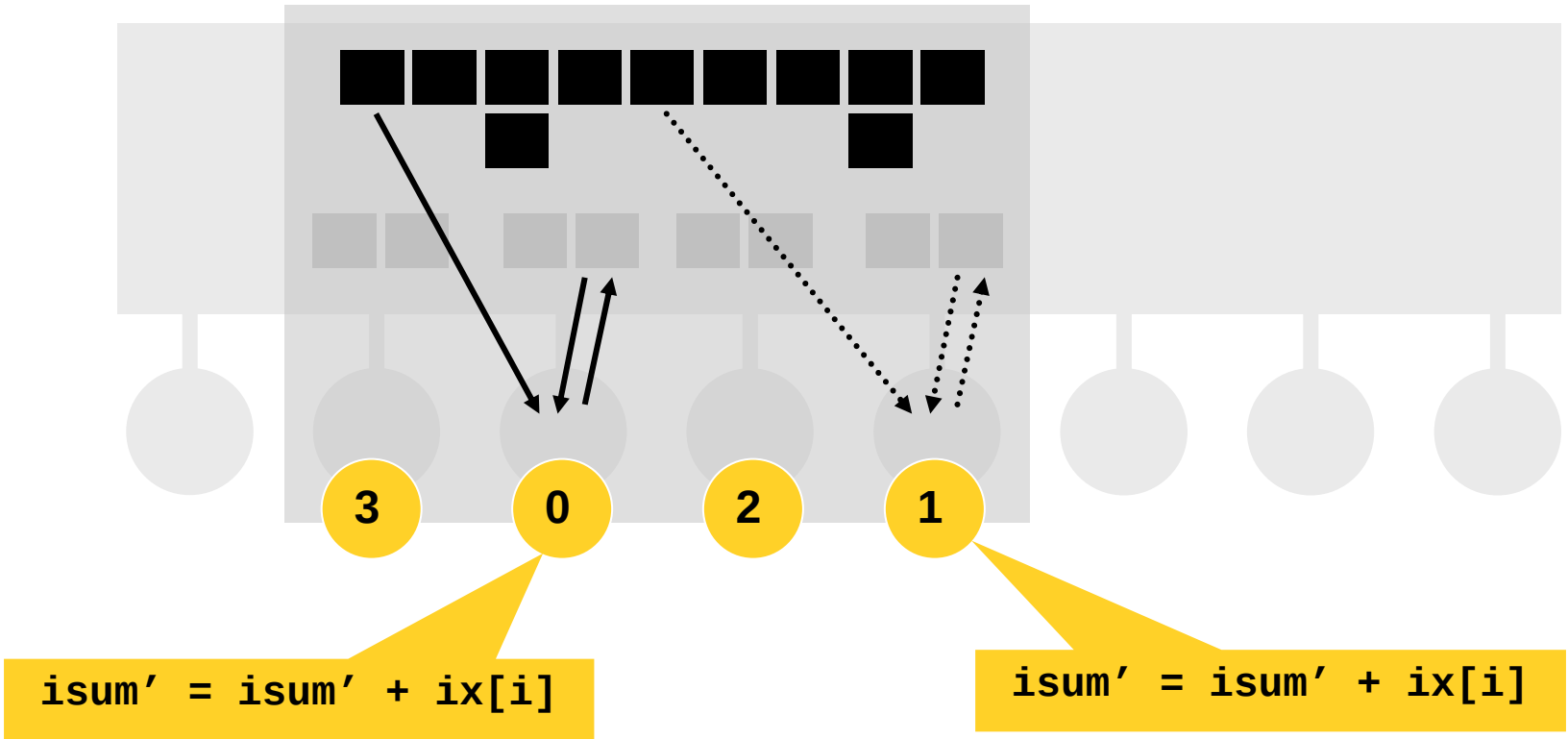
- Data scope...



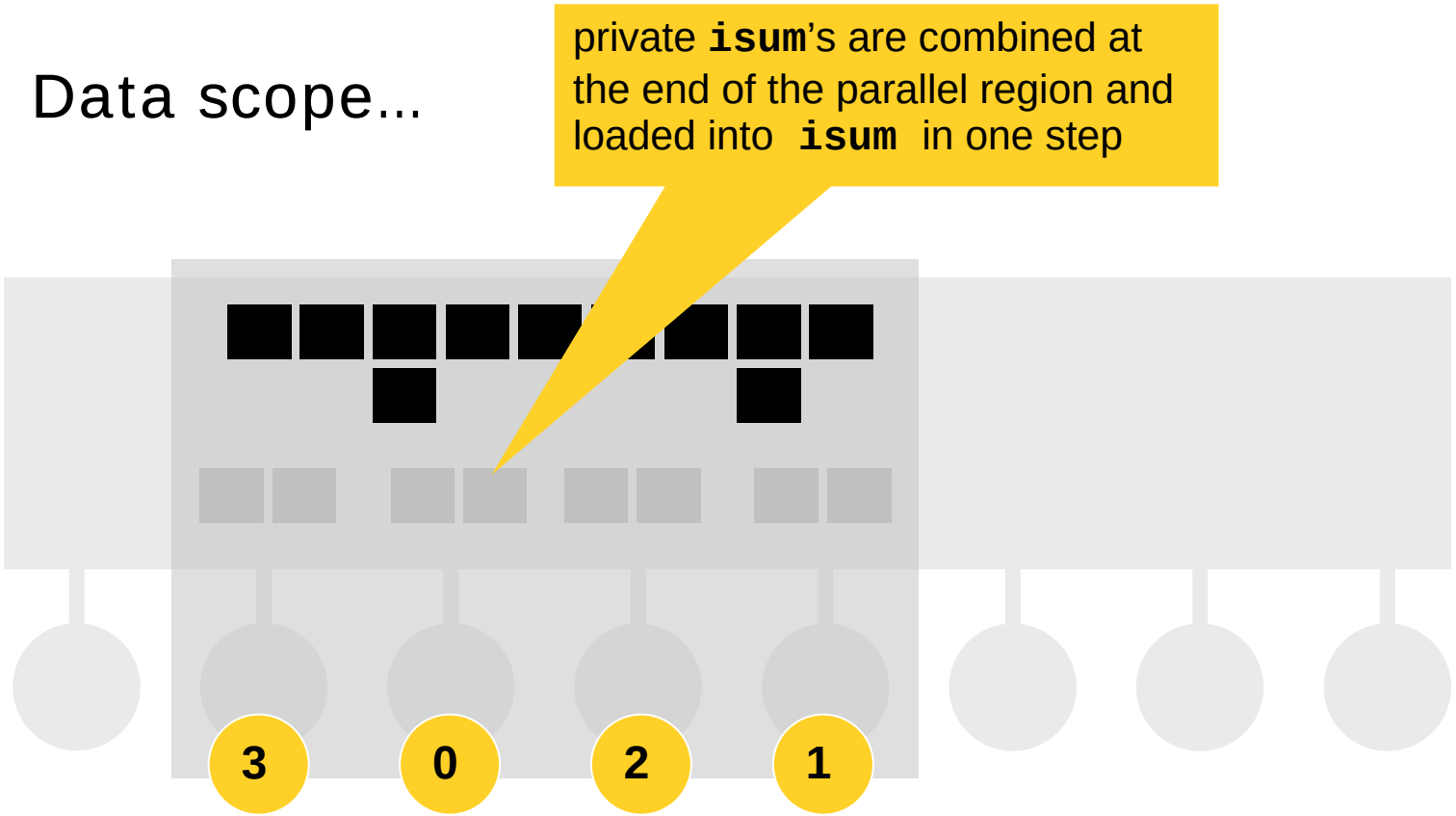
- Data scope...



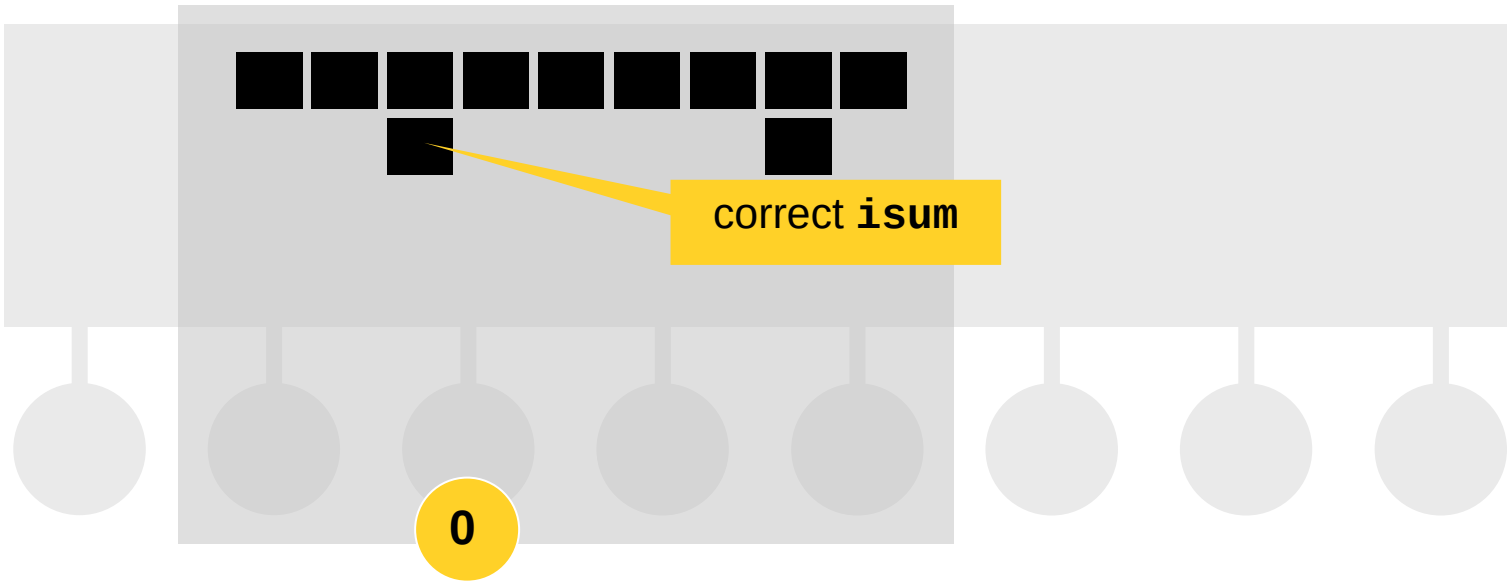
- Data scope…



- Data scope...



- Data scope...



- Data scope...

```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

#pragma omp parallel for schedule(static) private(i) shared(ix) reduction(+: isum)
  for (i=0; i<NX; i++)
    isum = isum + ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```

- Data scope…

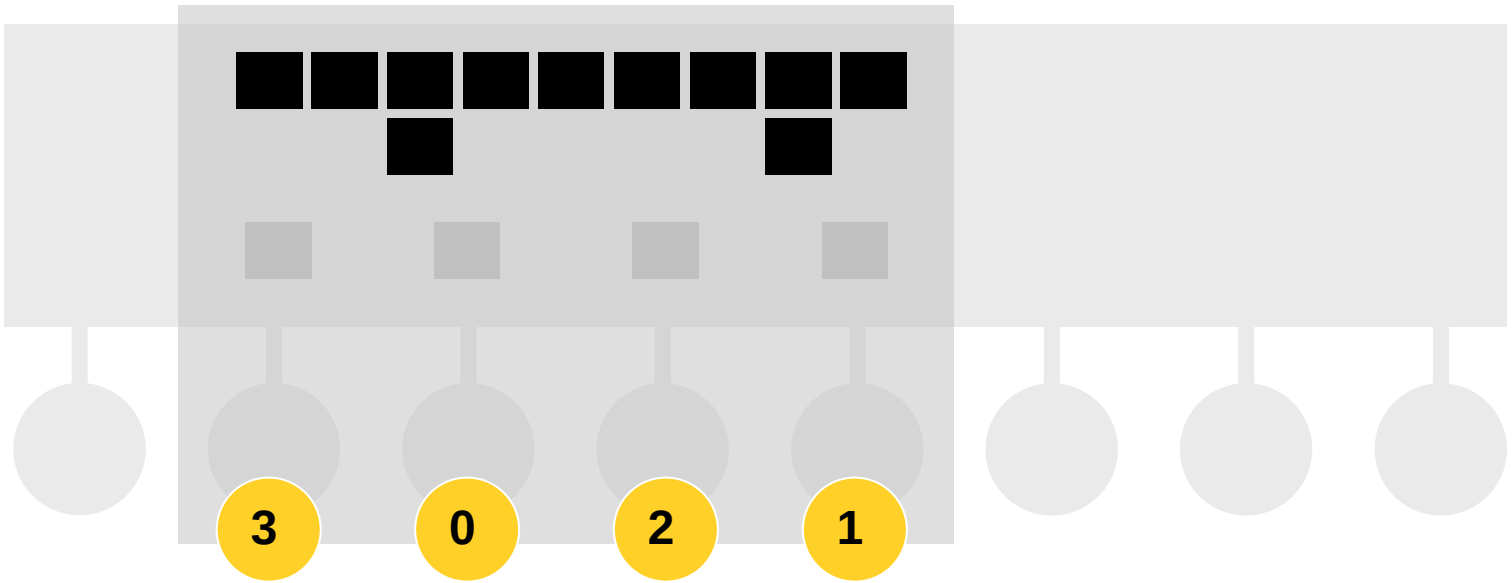
```
esumbar@aurora: cc -mp sum2.c
esumbar@aurora: setenv OMP_NUM_THREADS 2
esumbar@aurora: ./a.out
sum = 120000000
esumbar@aurora: setenv OMP_NUM_THREADS 4
esumbar@aurora: ./a.out
sum = 120000000
esumbar@aurora: ./a.out
sum = 120000000
```

- Fundamental OpenMP concept **synchronization**

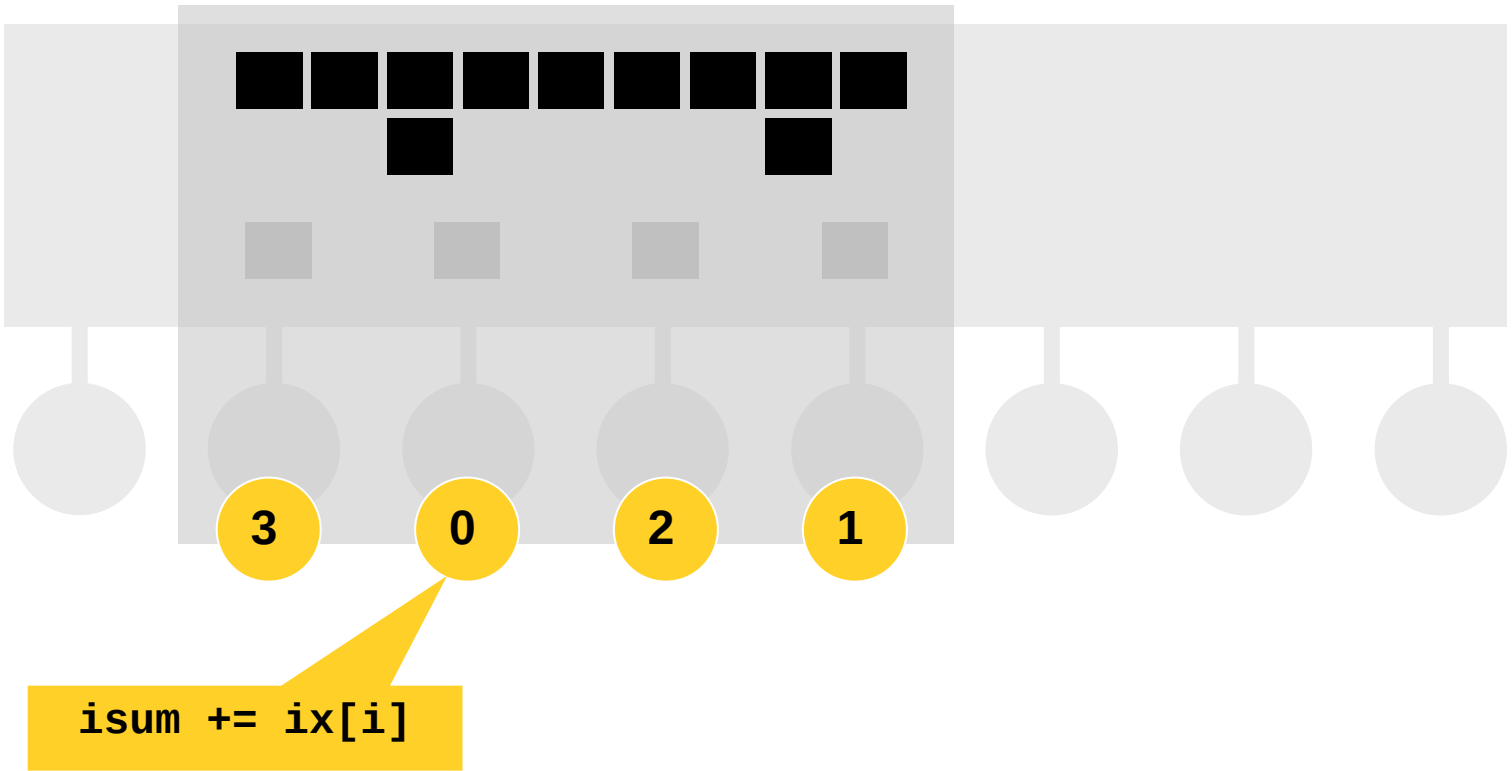
- Synchronization…

```
#include <stdio.h>
#define NX 60000000

int ix[NX];

int main (void)
{
  int i, isum;

  for (i=0; i<NX; i++)
    ix[i] = 2;

  isum = 0;

#pragma omp parallel for schedule(static) private(i) shared(isum,ix)
  for (i=0; i<NX; i++)
#pragma omp atomic
    isum += ix[i];

  printf("sum = %d\n", isum);
  return 0;
}
```

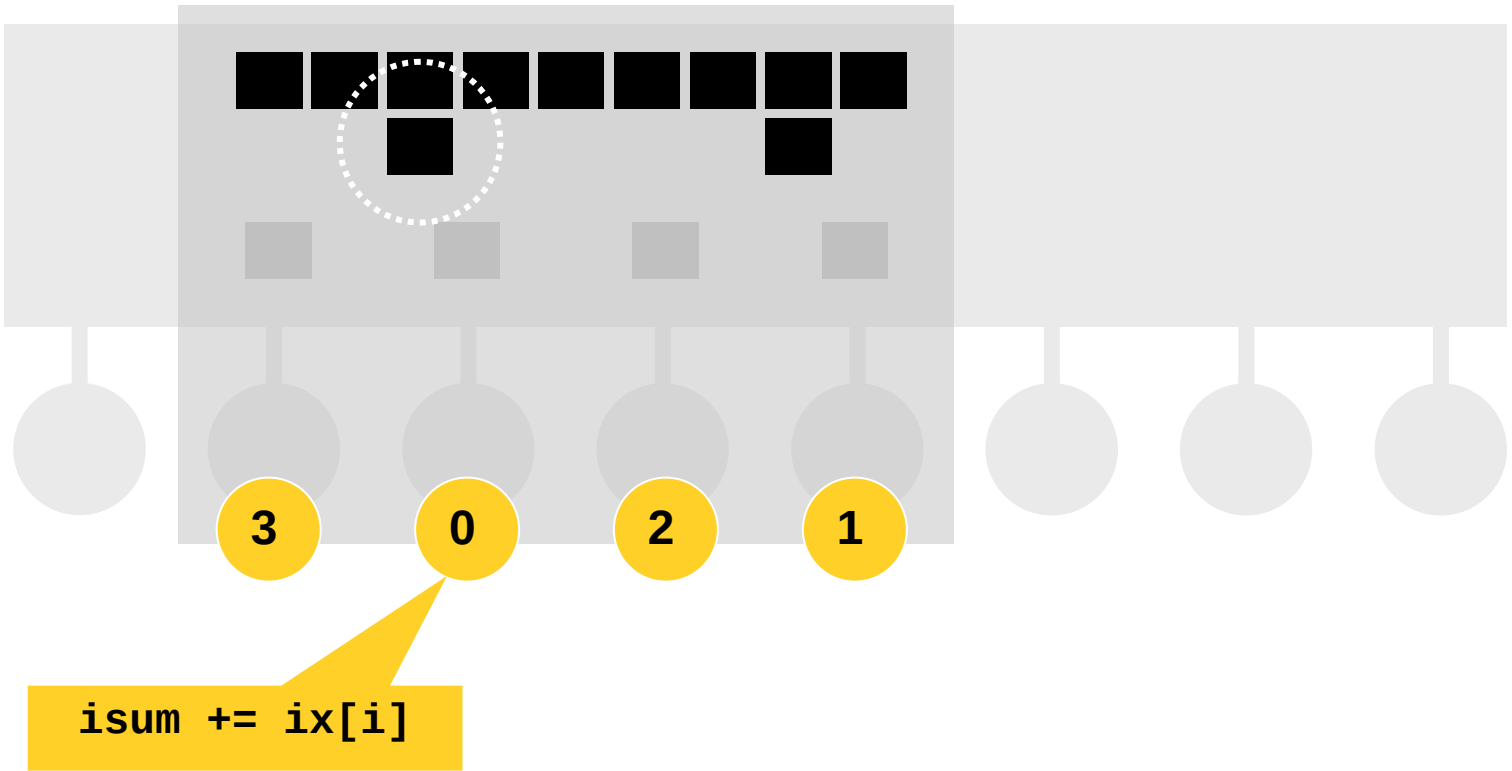
- Synchronization…



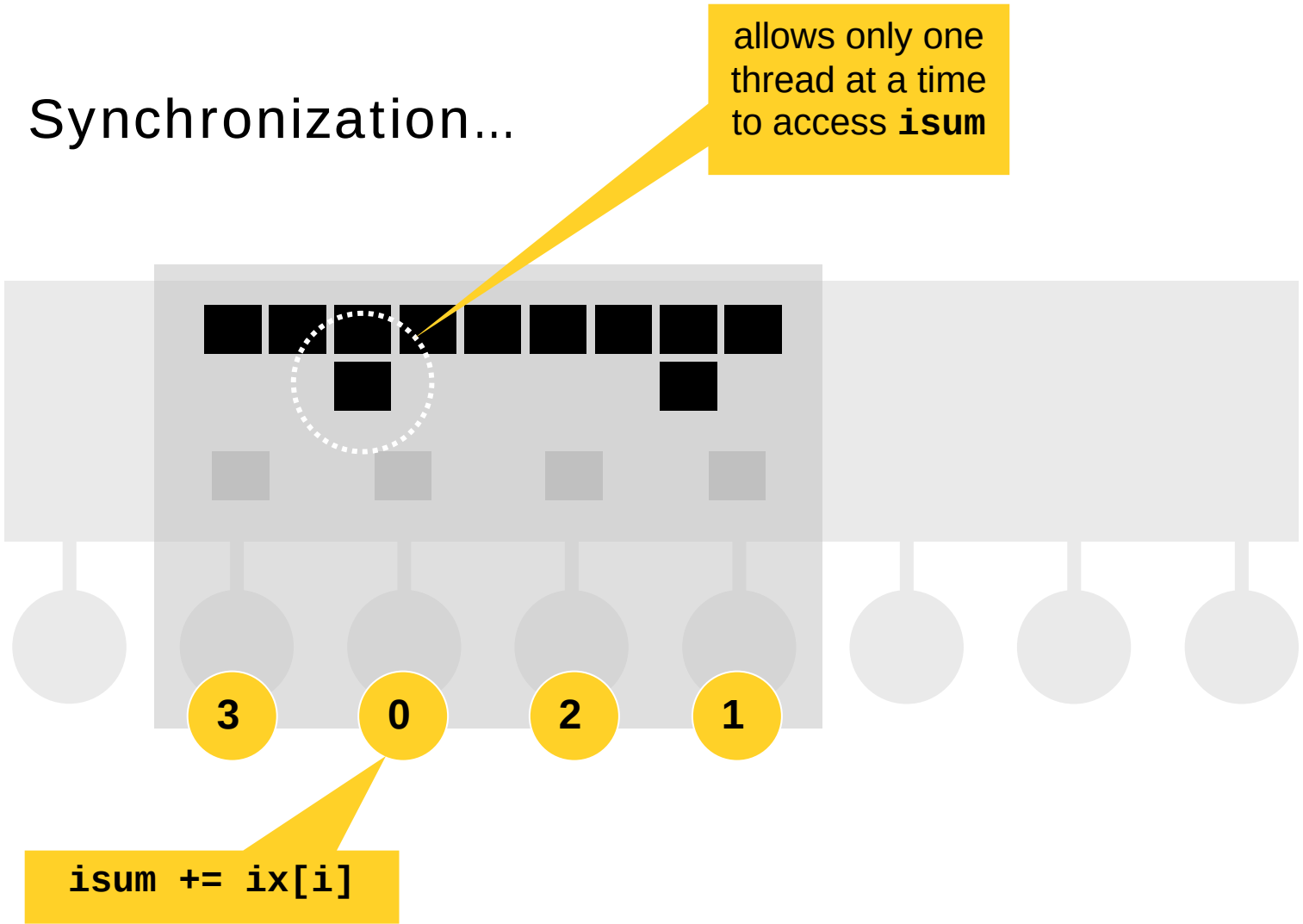
- Synchronization…



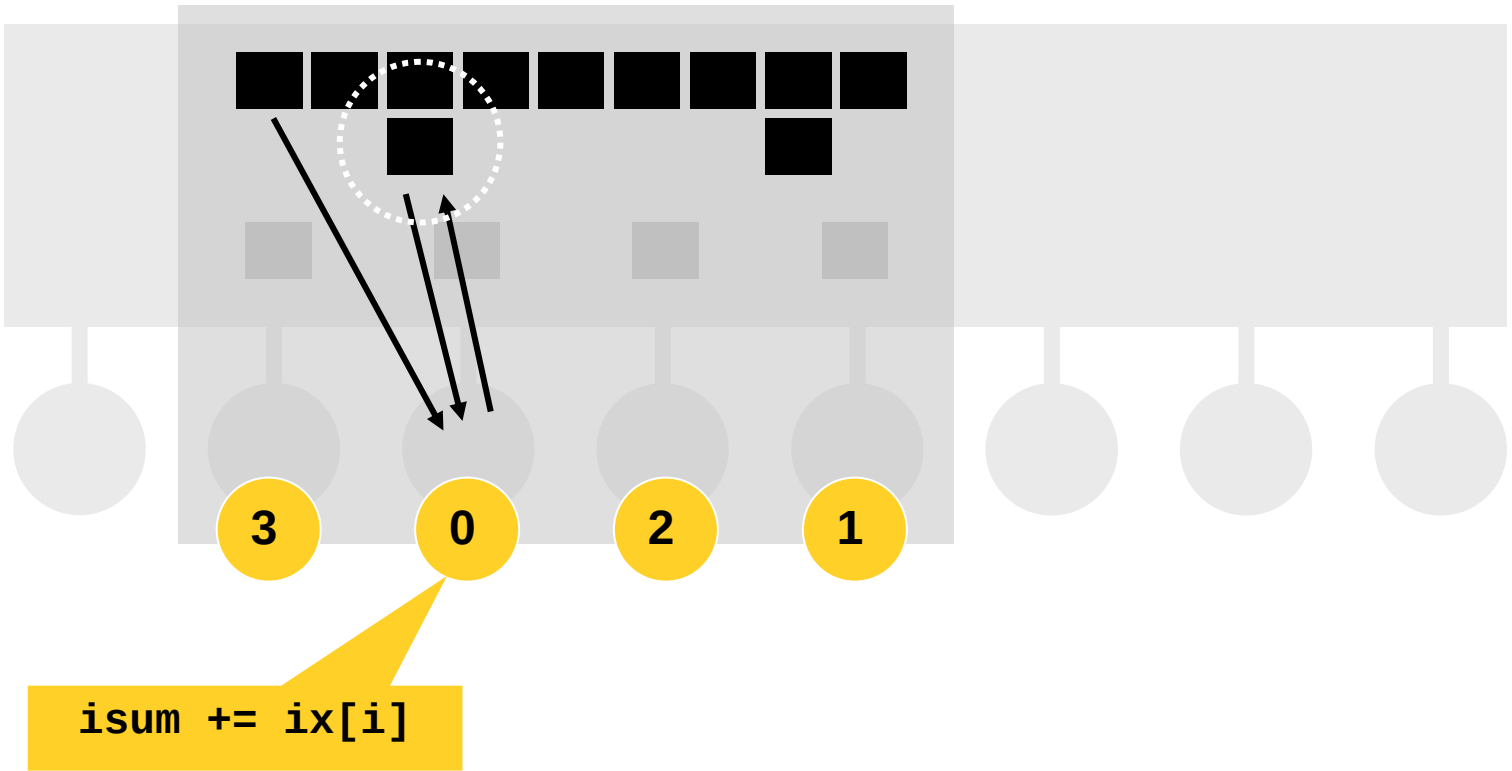
- Synchronization…



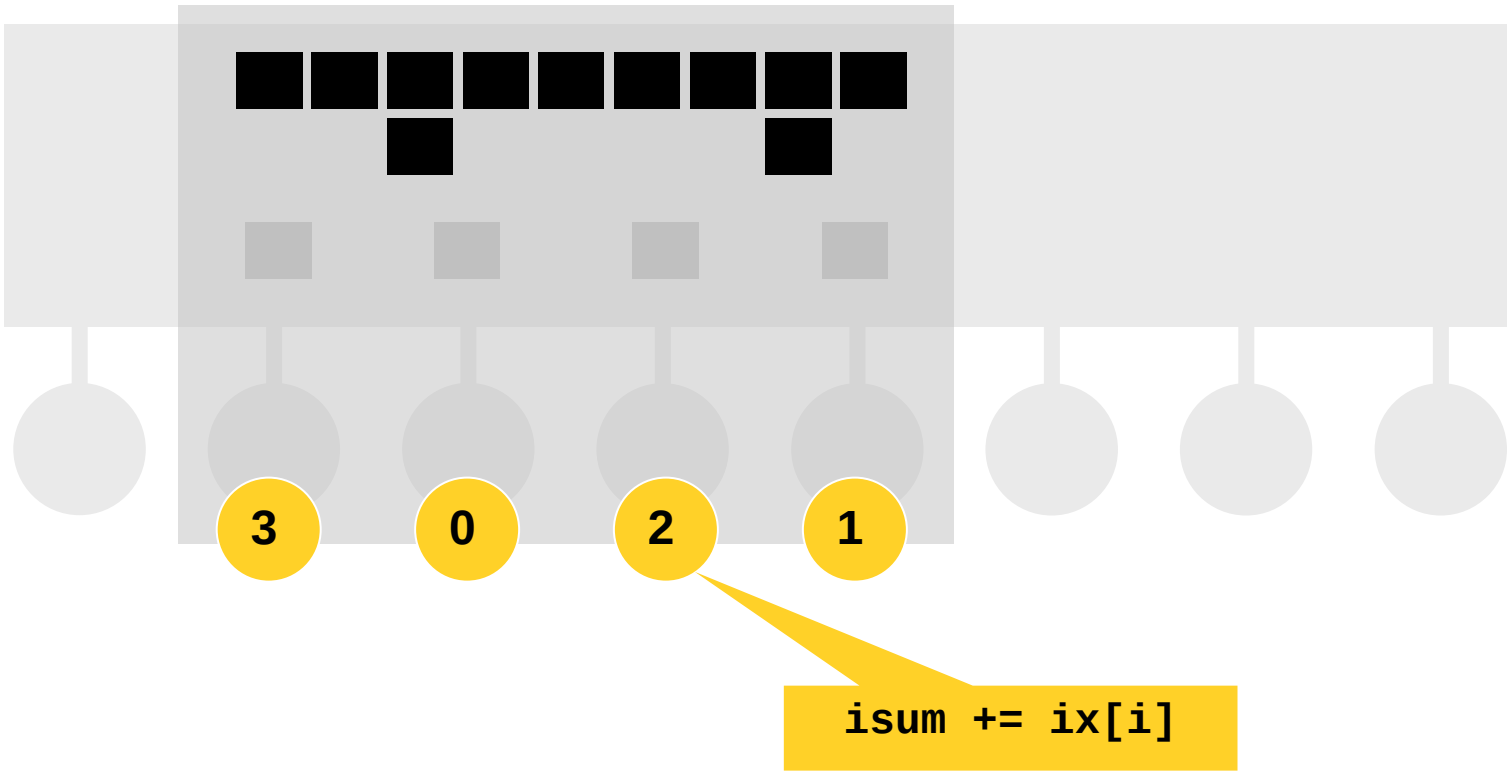
- Synchronization...



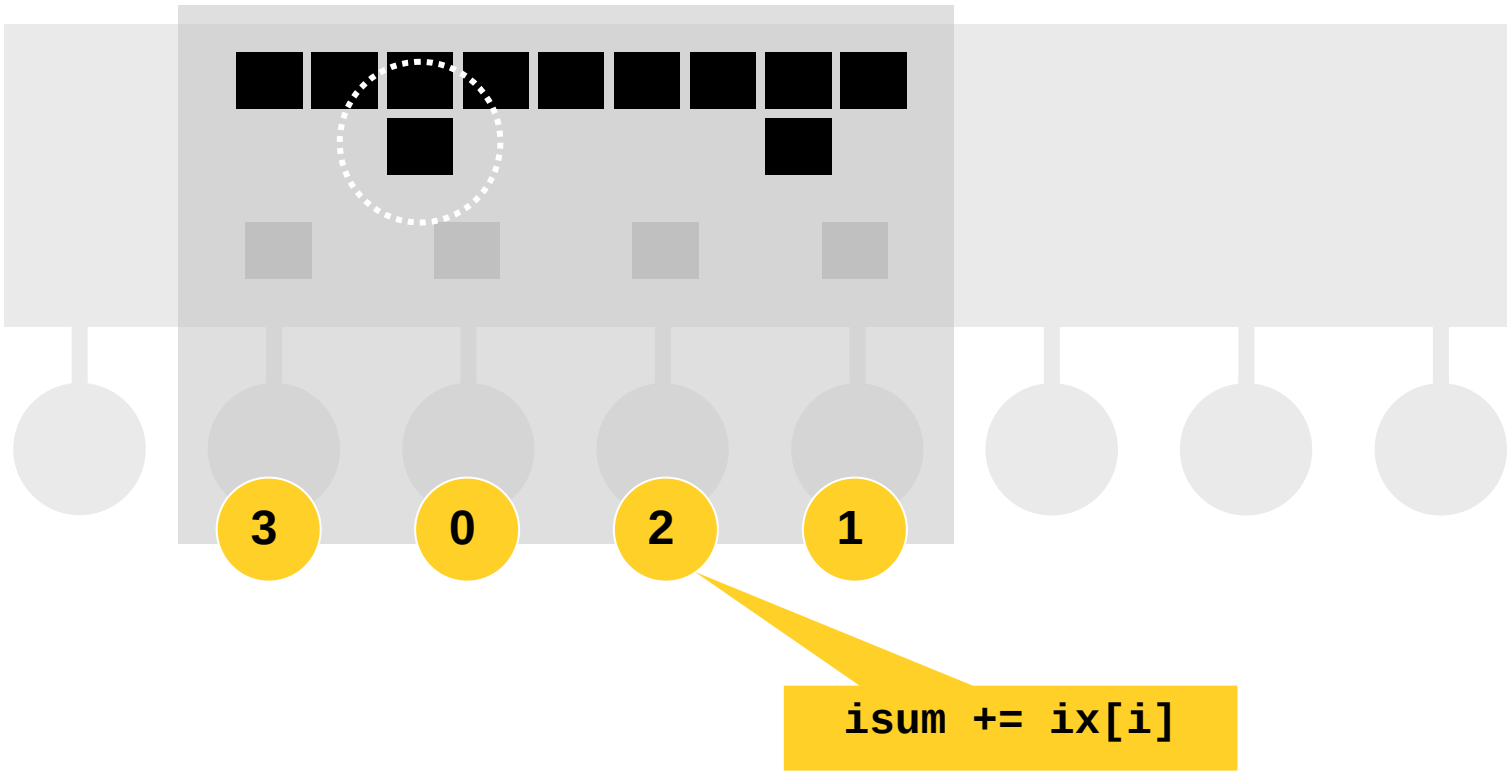
- Synchronization…



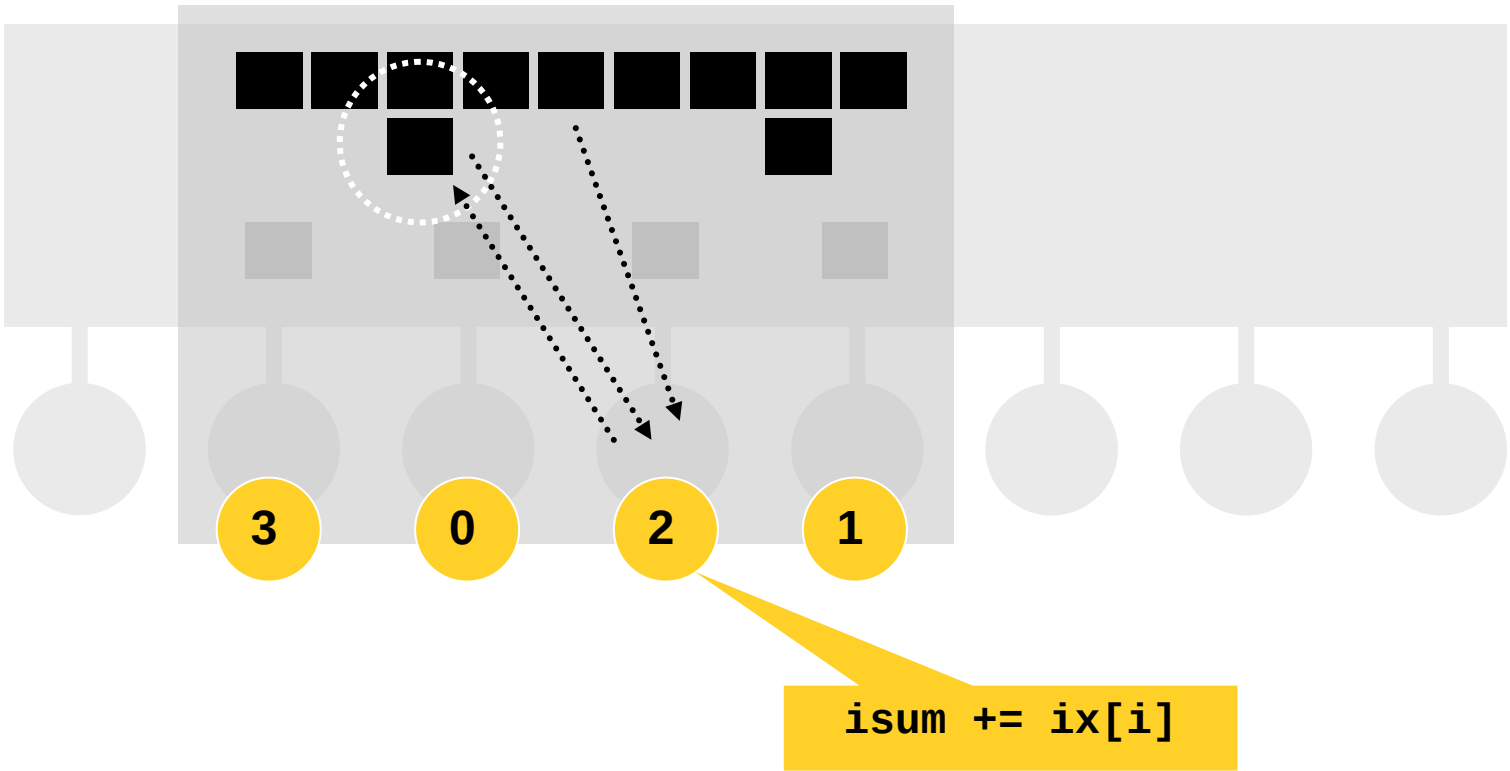
- Synchronization…



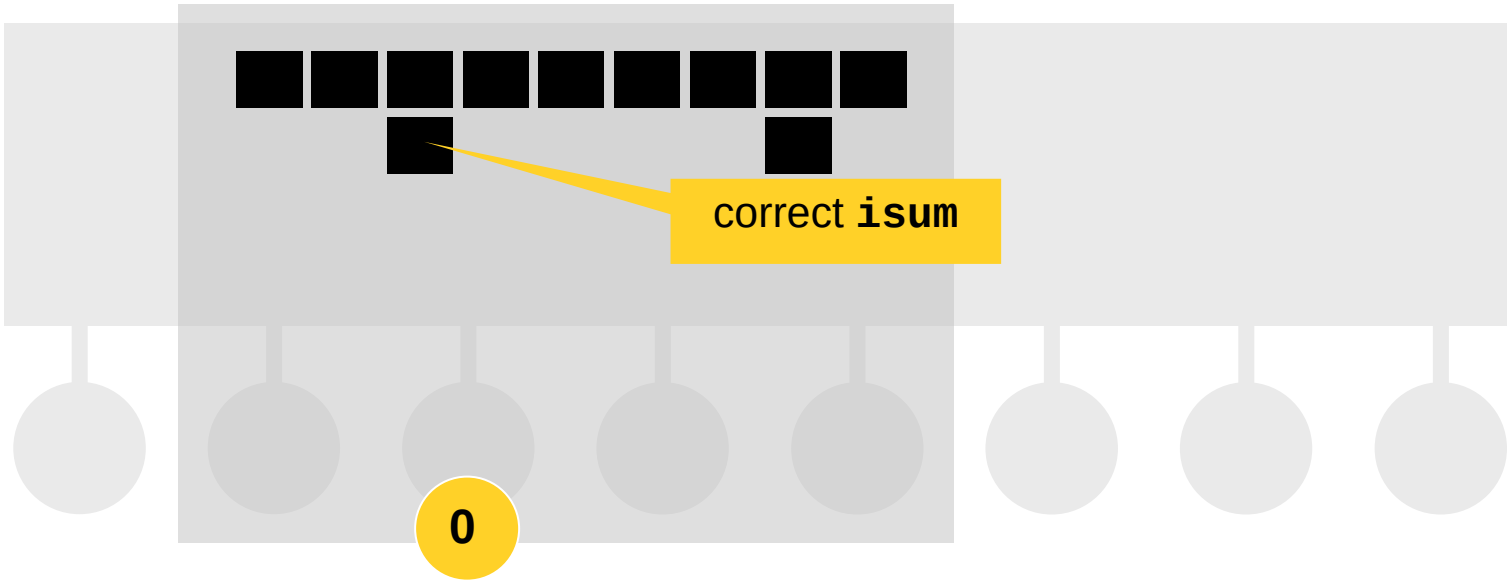
- Synchronization…



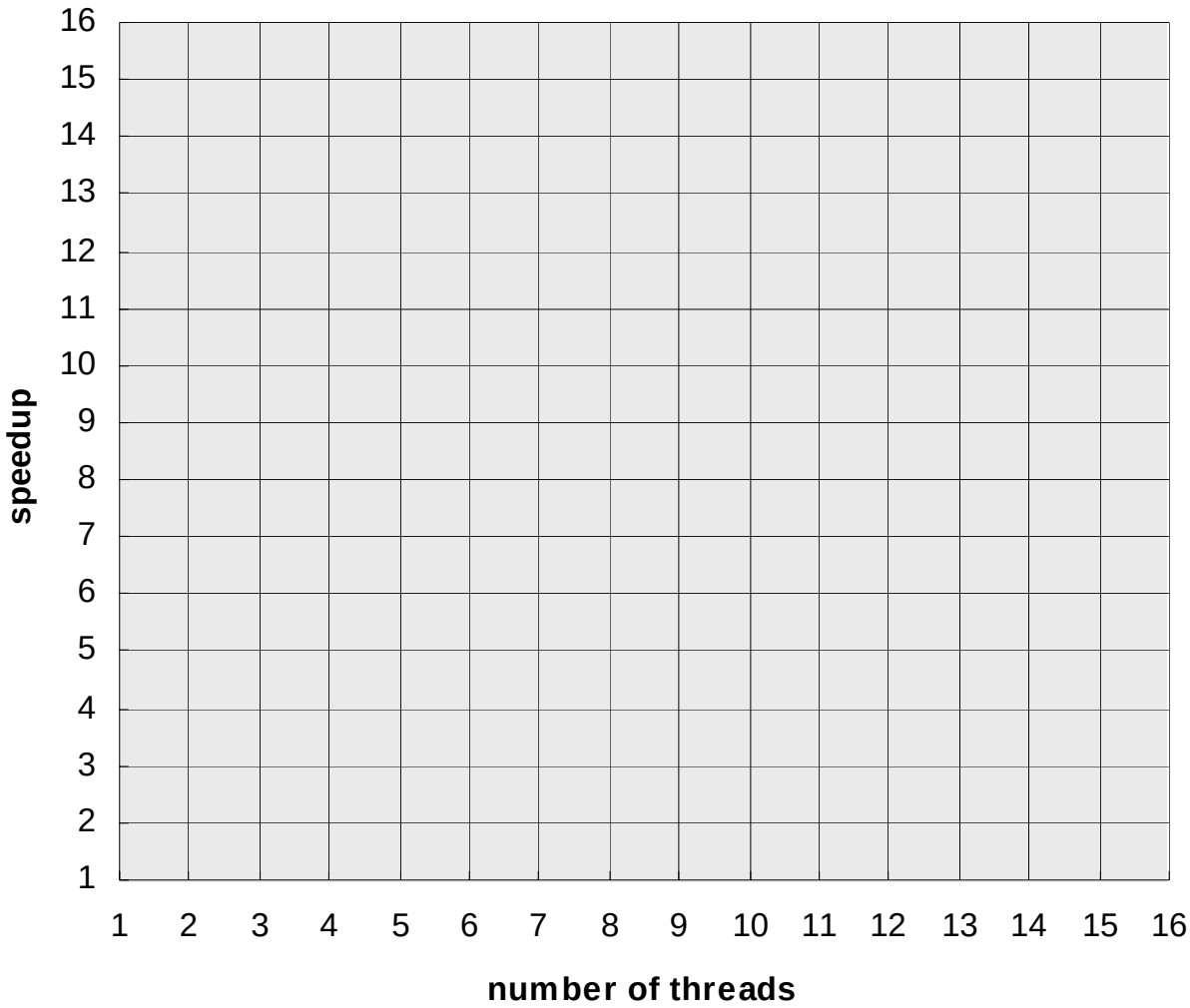
- Synchronization…



- Data scope...



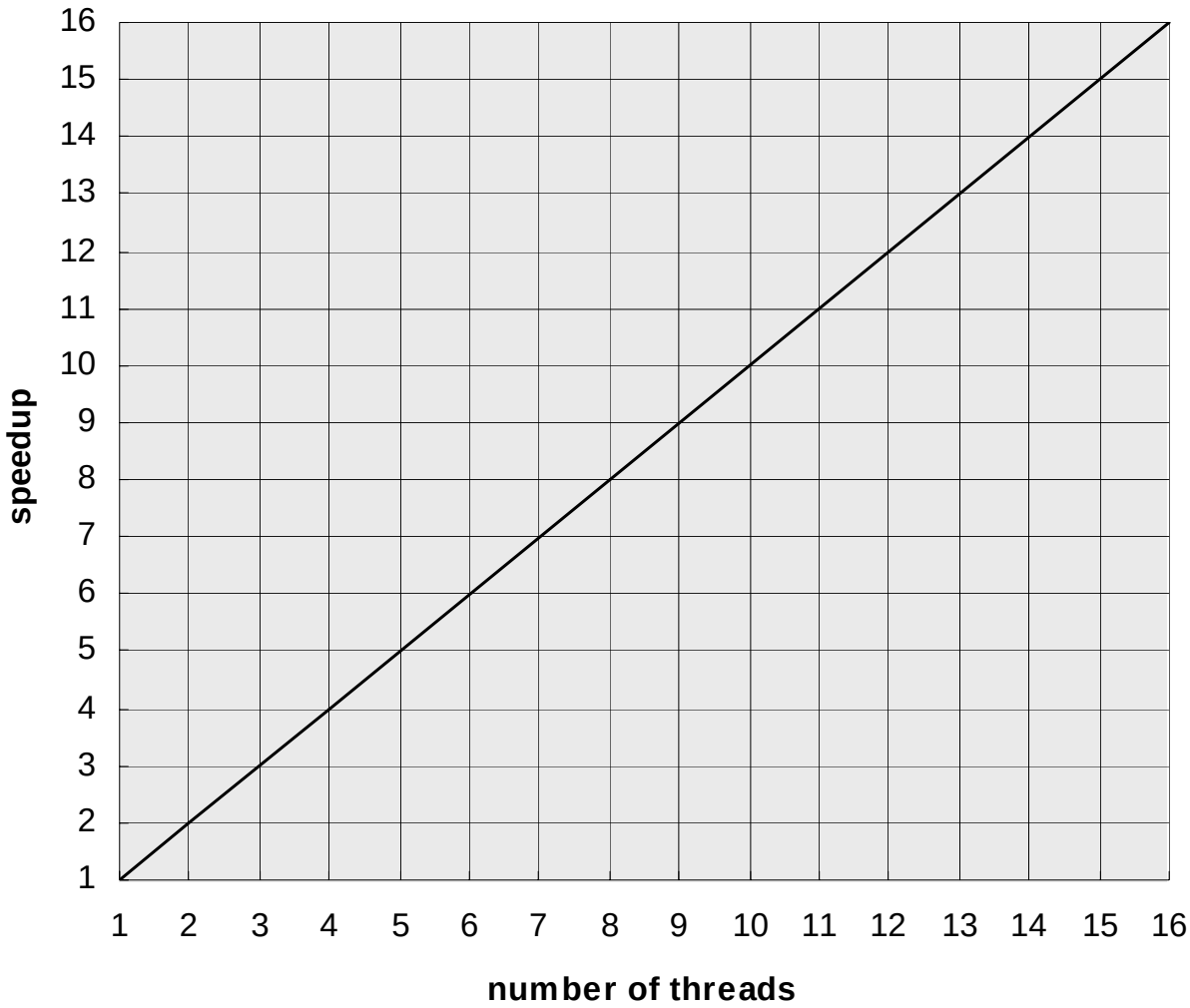
- Fundamental OpenMP concept **scalability**

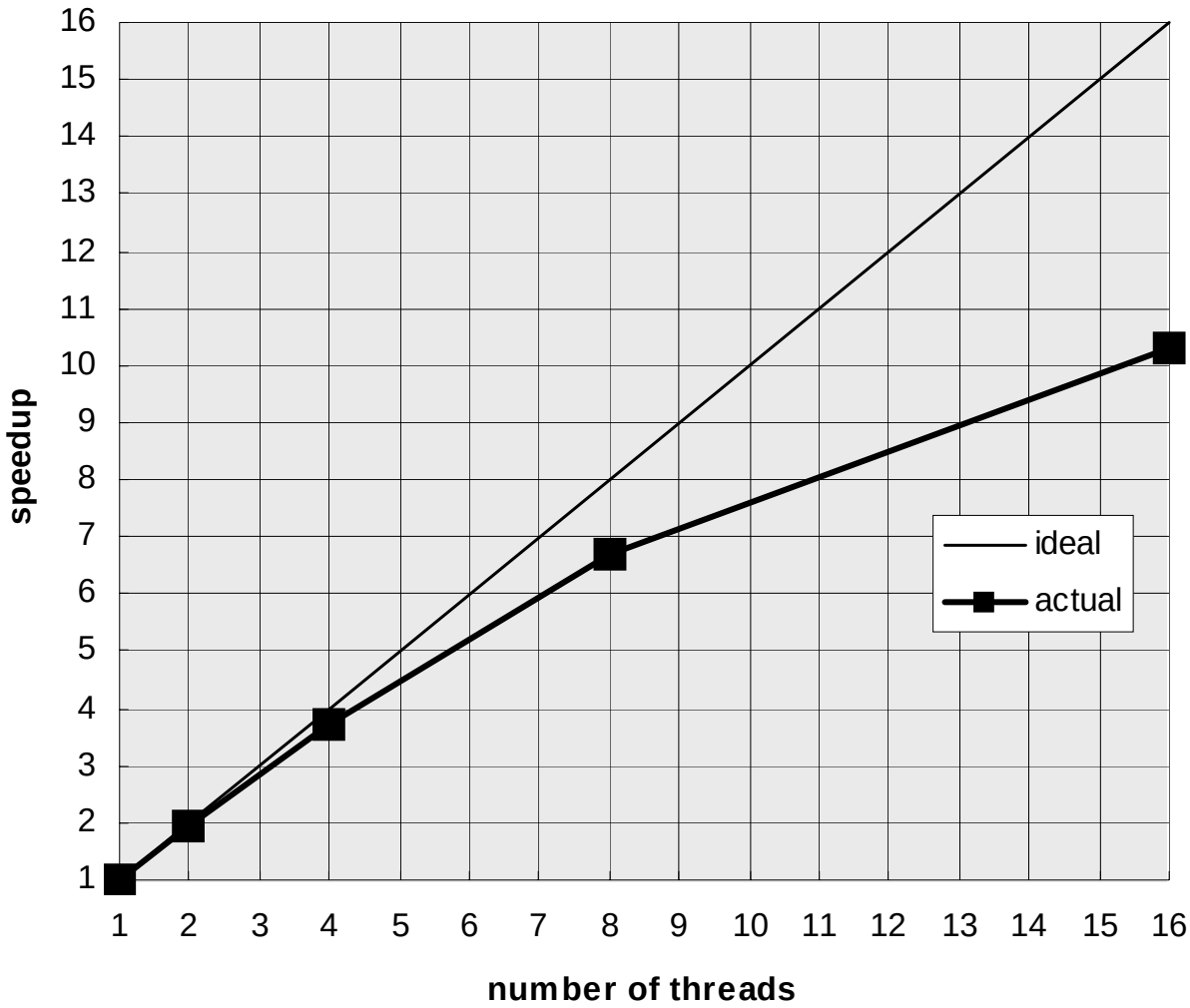






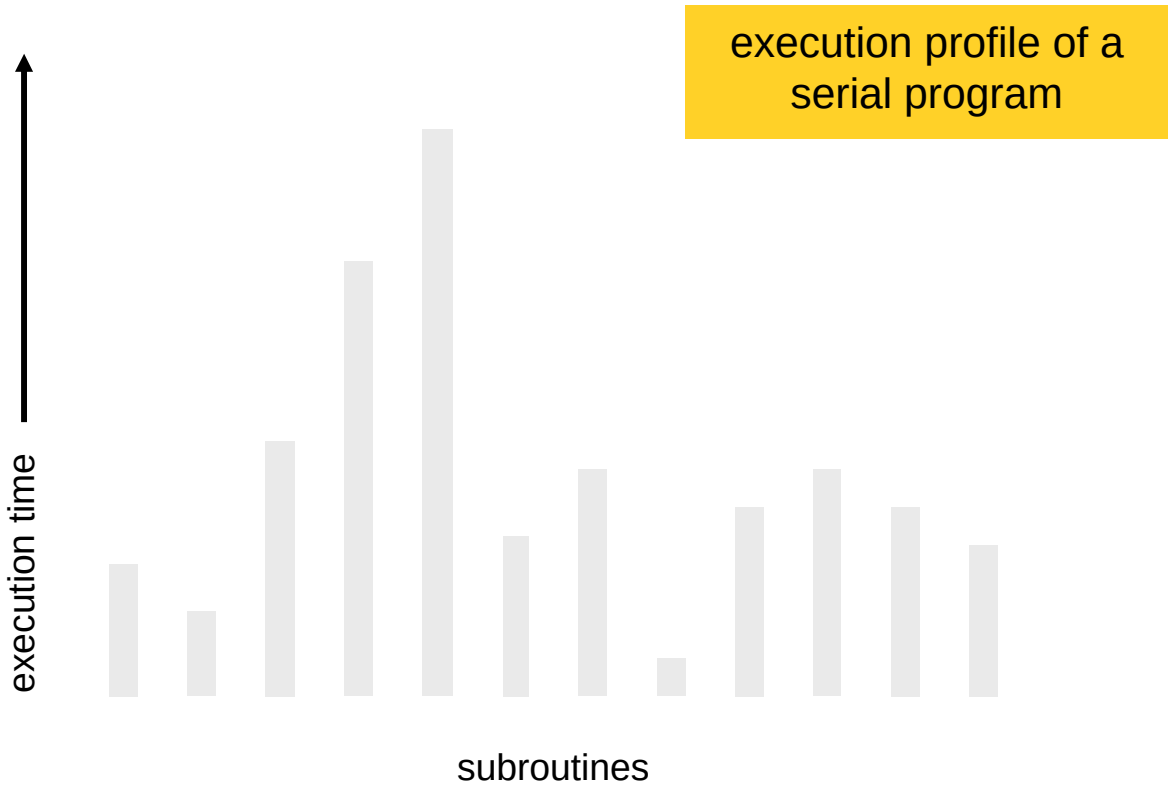
create delete



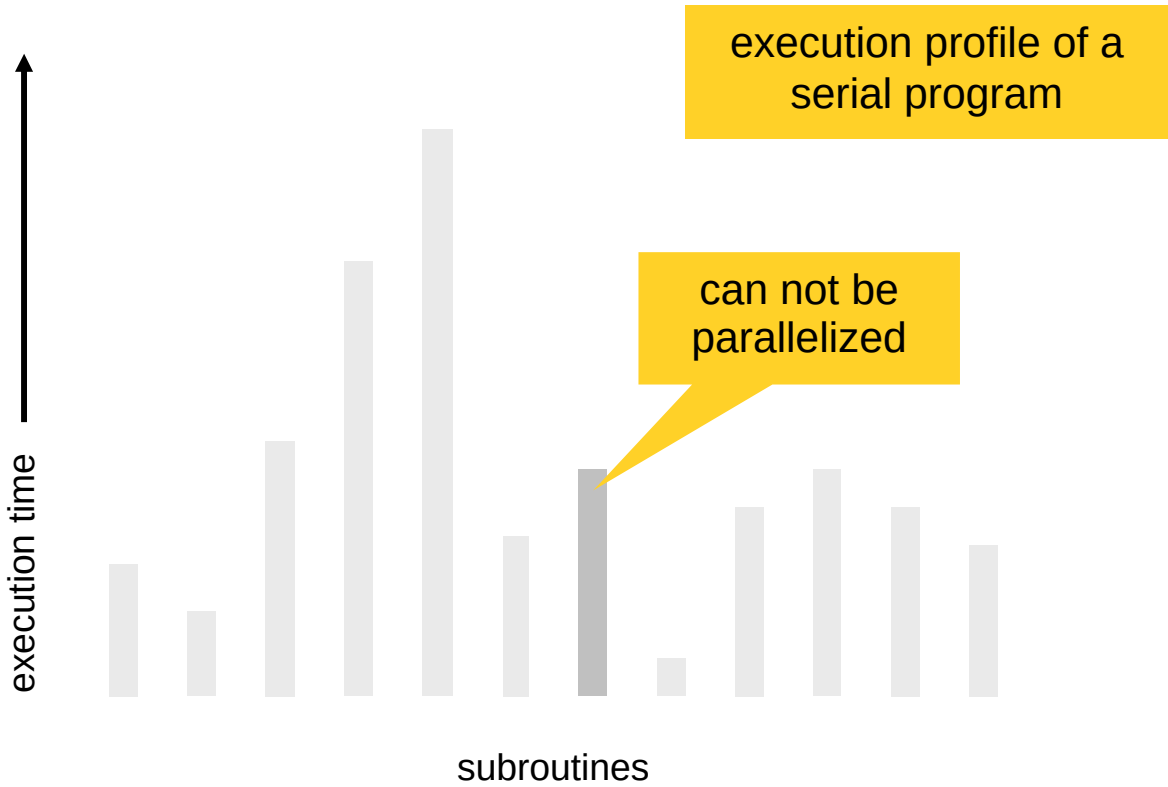


- Factors limiting scalability
  - Amdahl's law
  - Load imbalance
  - Barrier overhead
  - Synchronization
  - False sharing

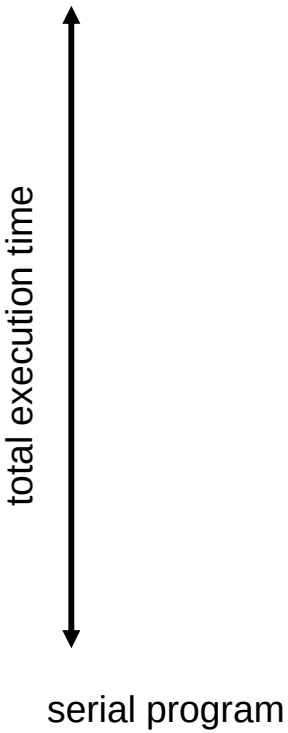
- Scalability and Amdahl's law...



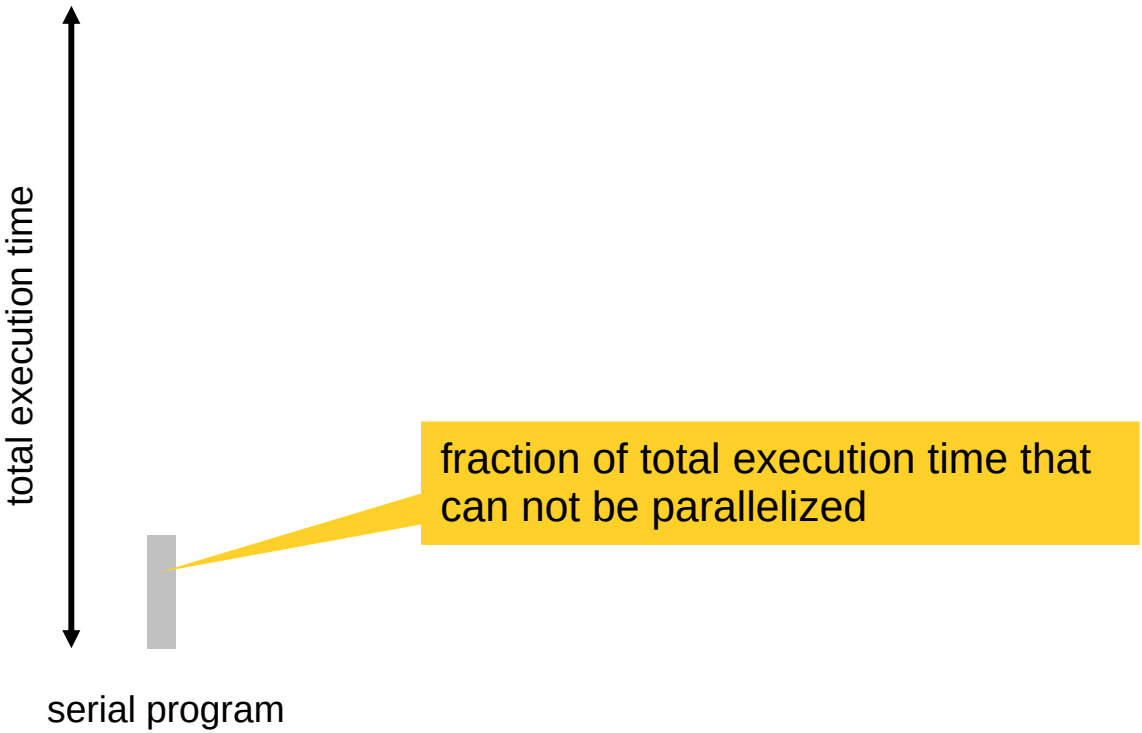
- Scalability and Amdahl's law...



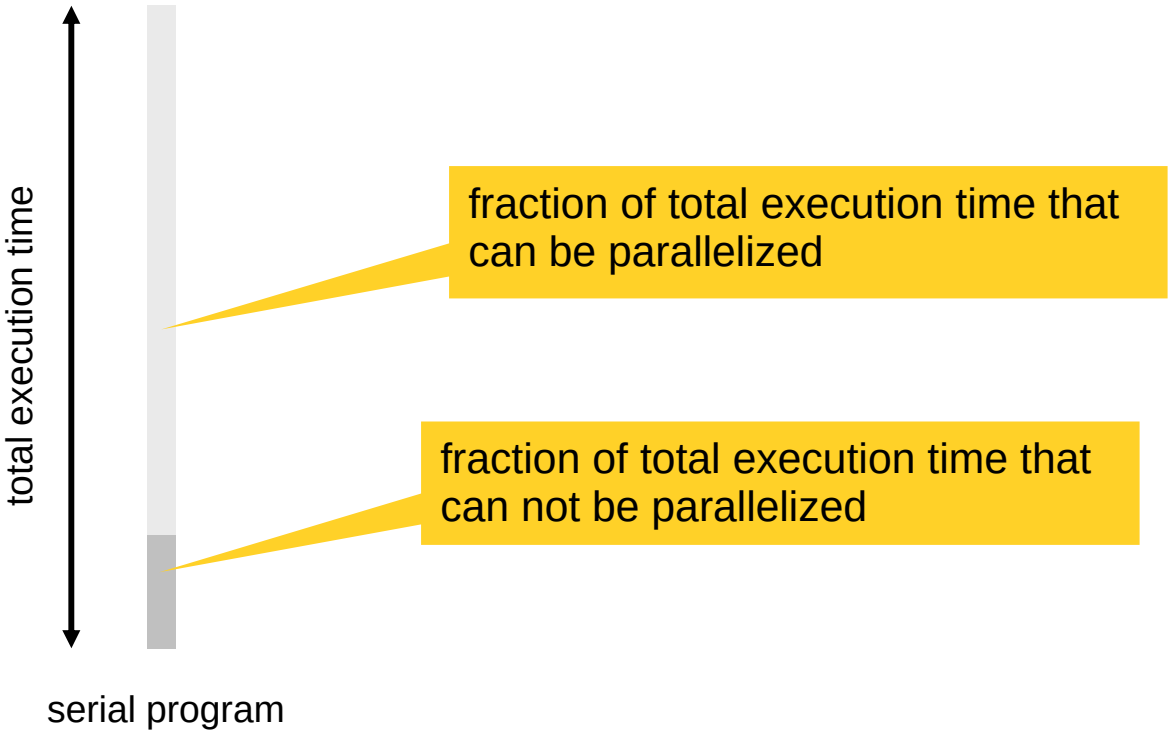
- Scalability and Amdahl's law...



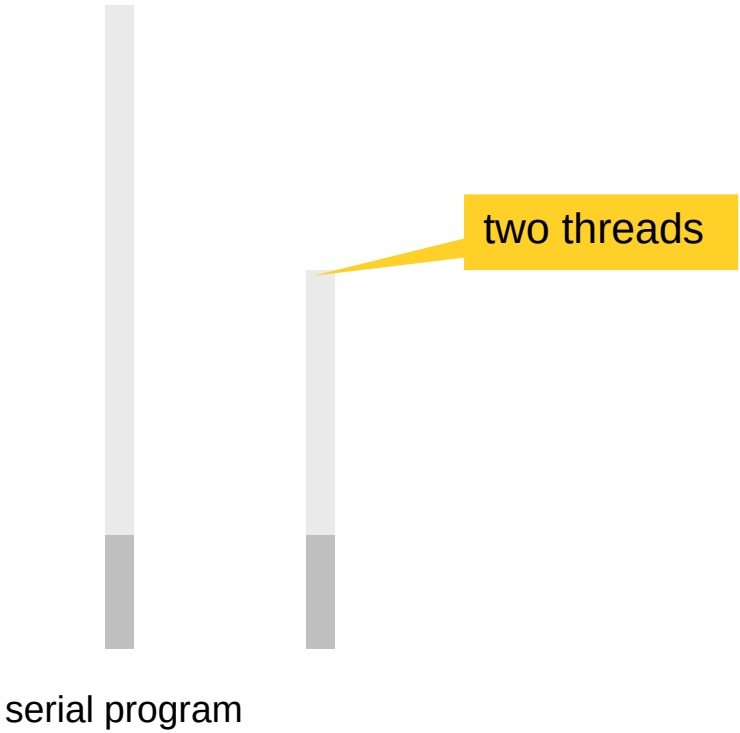
- Scalability and Amdahl's law...



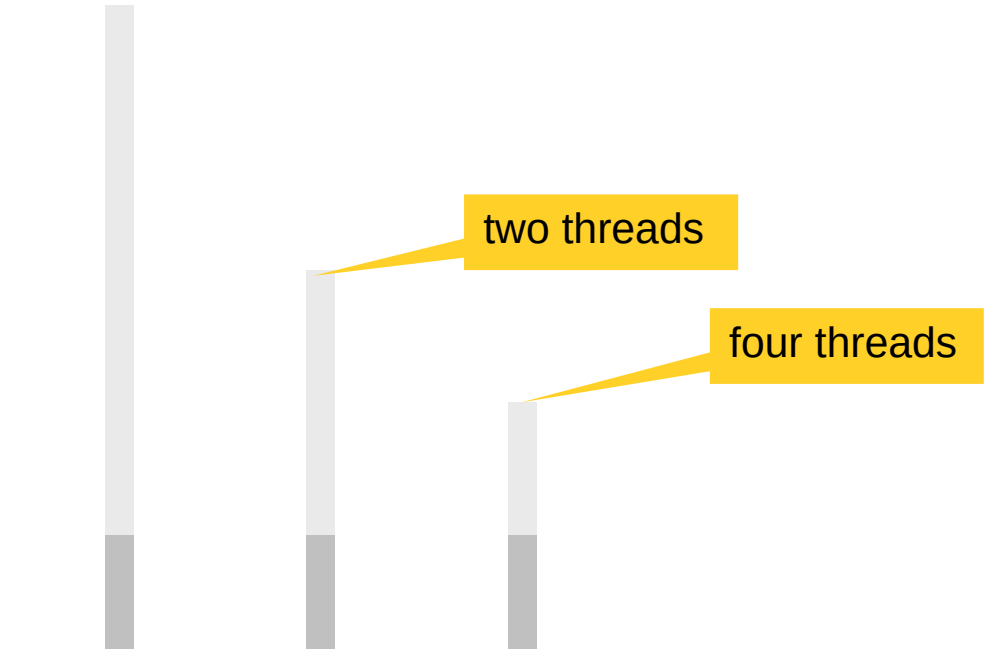
- Scalability and Amdahl's law...



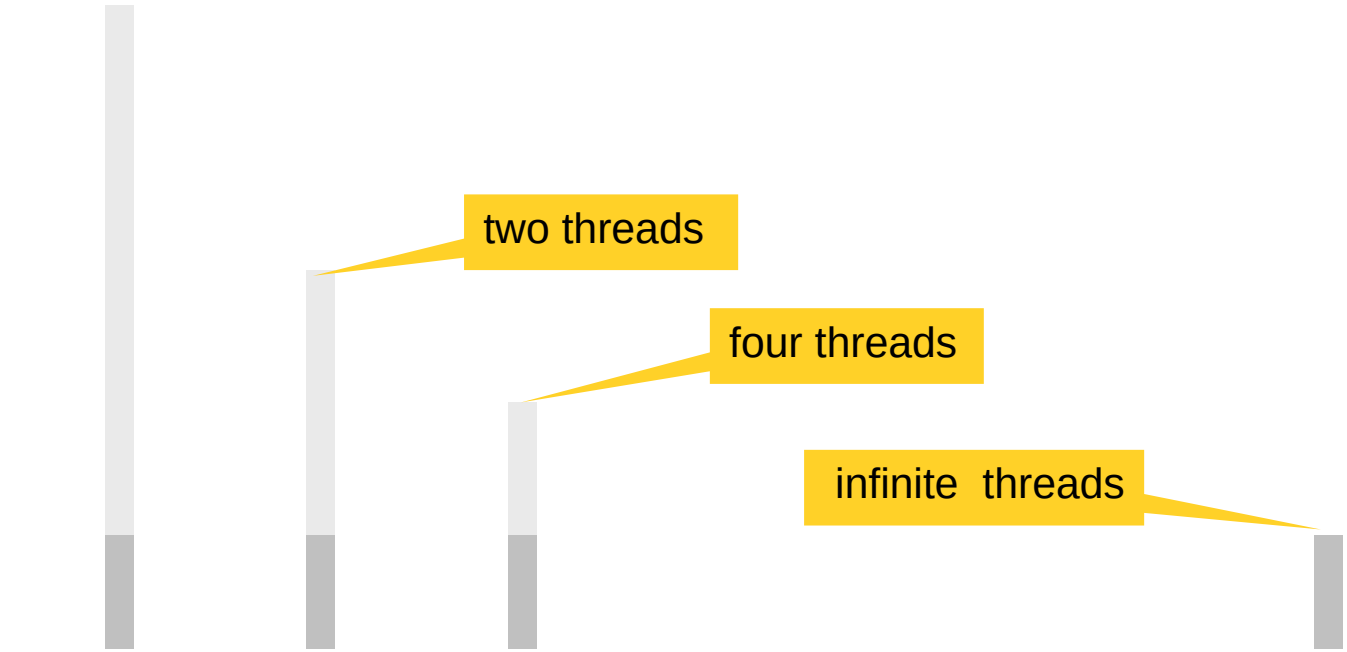
- Scalability and Amdahl's law...



- Scalability and Amdahl's law...



- Scalability and Amdahl's law...



- Scalability and Amdahl's law...

$$\text{max speedup} \equiv \frac{T_{serial}}{T_{parallel\ \infty}} = \frac{T_{serial}}{\alpha T_{serial}} = \frac{1}{\alpha}$$

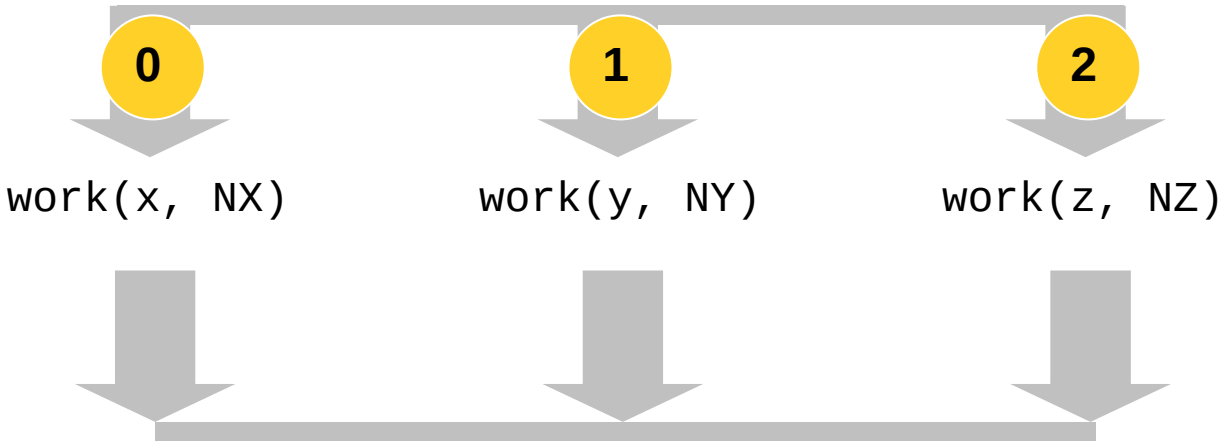
- Scalability and load imbalance…

```
#define NX 10000000
#define NY 10000000
#define NZ 10000000

float x[NX], y[NY], z[NZ];

...

int main (void)
{
  printf("start\n");

#pragma omp parallel
  {
#pragma omp sections
    {
#pragma omp section
      work(x, NX);
#pragma omp section
      work(y, NY);
#pragma omp section
      work(z, NZ);
    }
  }

  printf("end\n");
  return 0;
}
```

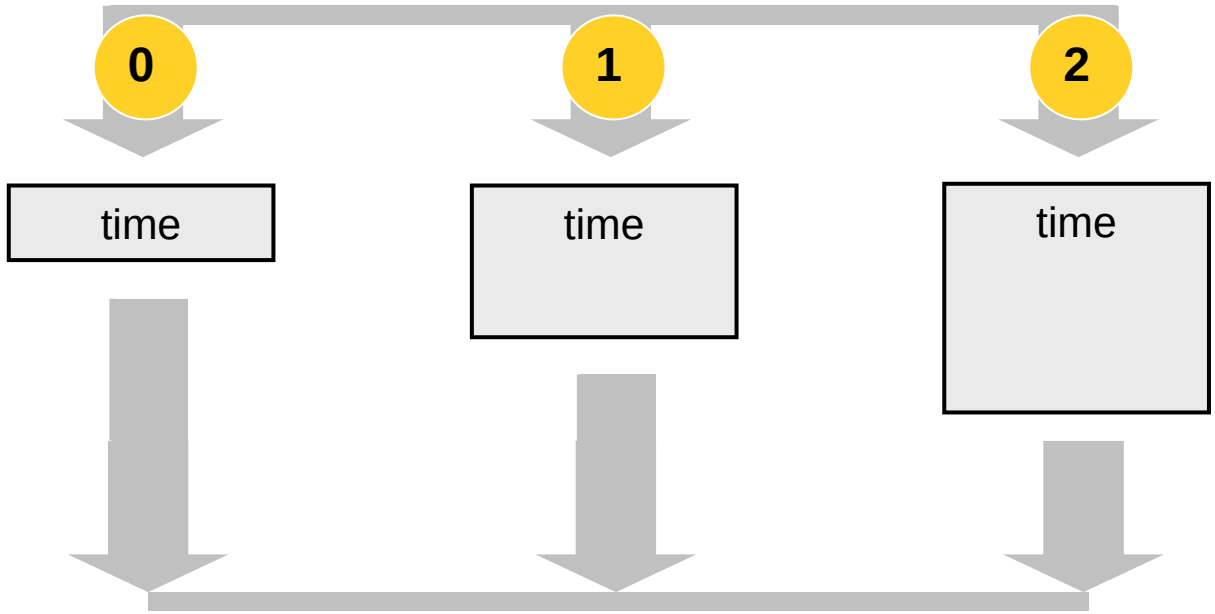
```
#define NX 10000000
#define NY 10000000
#define NZ 10000000

float x[NX], y[NY], z[NZ];

...

int main (void)
{
  printf("start\n");

#pragma omp parallel
  {
#pragma omp sections
    {
#pragma omp section
      work(x, NX);
#pragma omp section
      work(y, NY);
#pragma omp section
      work(z, NZ);
    }
  }

  printf("end\n");
  return 0;
}
```

```
#define NX 10000000
#define NY 20000000
#define NZ 30000000

float x[NX], y[NY], z[NZ];

...

int main (void)
{
  printf("start\n");

#pragma omp parallel
  {
#pragma omp sections
    {
#pragma omp section
      work(x, NX);
#pragma omp section
      work(y, NY);
#pragma omp section
      work(z, NZ);
    }
  }

  printf("end\n");
  return 0;
}
```

- Scalability and load imbalance…



- Scalability and load imbalance…



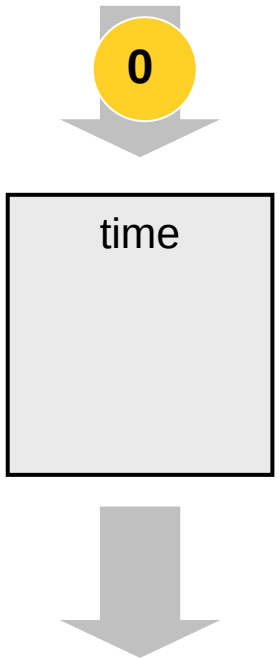
- Scalability and load imbalance…

```
esumbar@aurora: cc work2.c -lm
esumbar@aurora: time ./a.out
start
end
20.934u 1.680s 0:22.78 99.2% 0+0k 0+0io 0pf+0w
esumbar@aurora: cc -mp work2.c -lm
esumbar@aurora: setenv OMP_NUM_THREADS 3
esumbar@aurora: time ./a.out
start
end
31.191u 3.697s 0:12.15 287.0% 0+0k 1+5io 1pf+0w
```

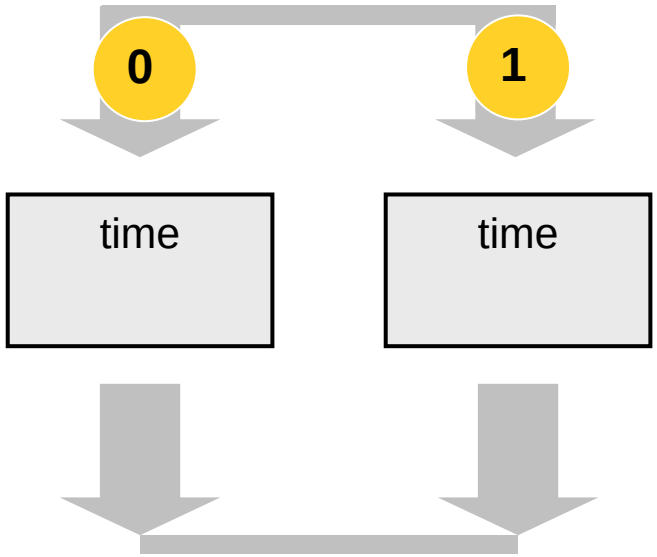
- Scalability and load imbalance…

$$\text{speedup} = \frac{T_{serial}}{T_{parallel}} = \frac{22.78}{12.15} = 1.9$$

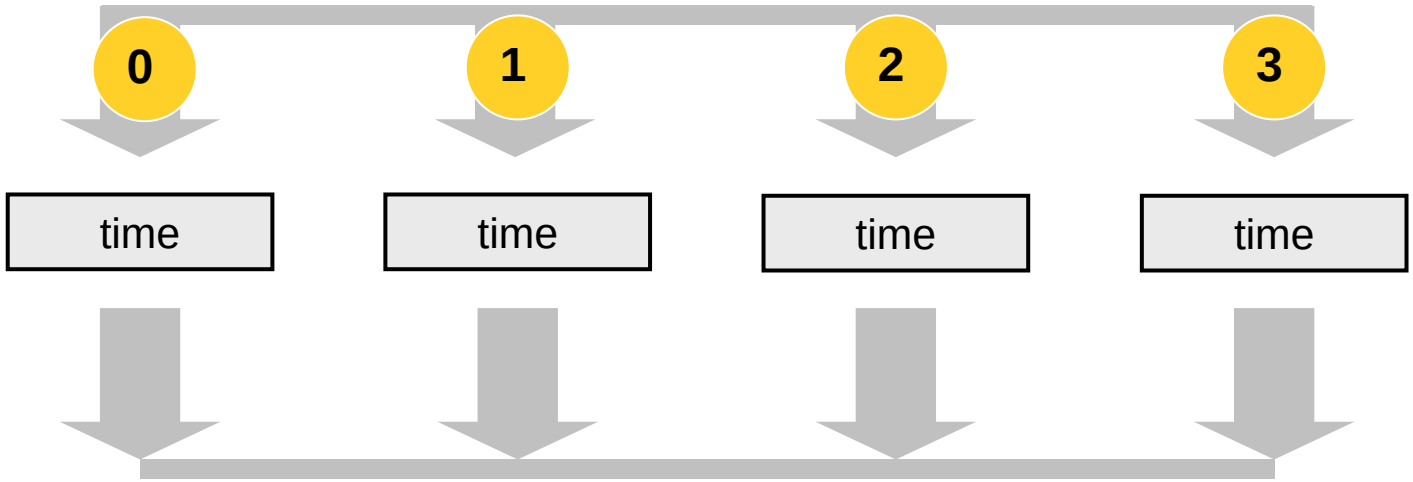
- Scalability and barrier overhead…



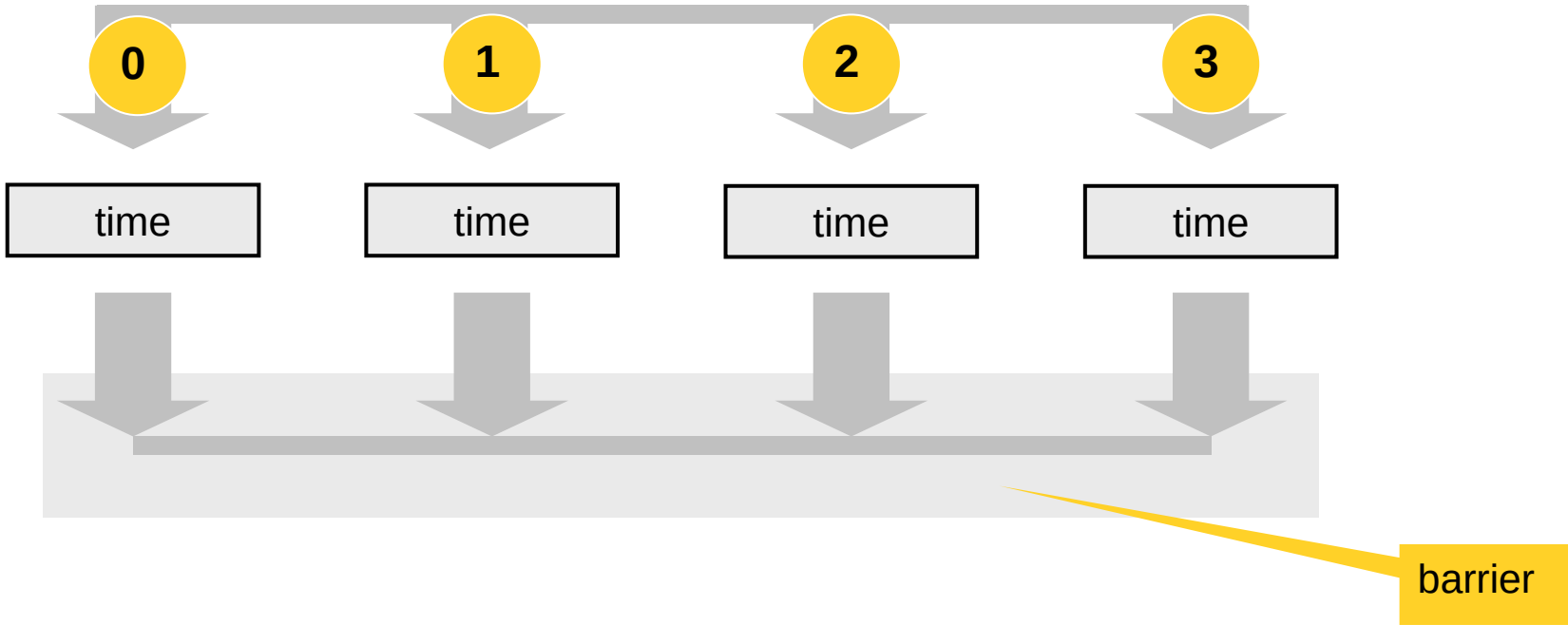
- Scalability and barrier overhead…



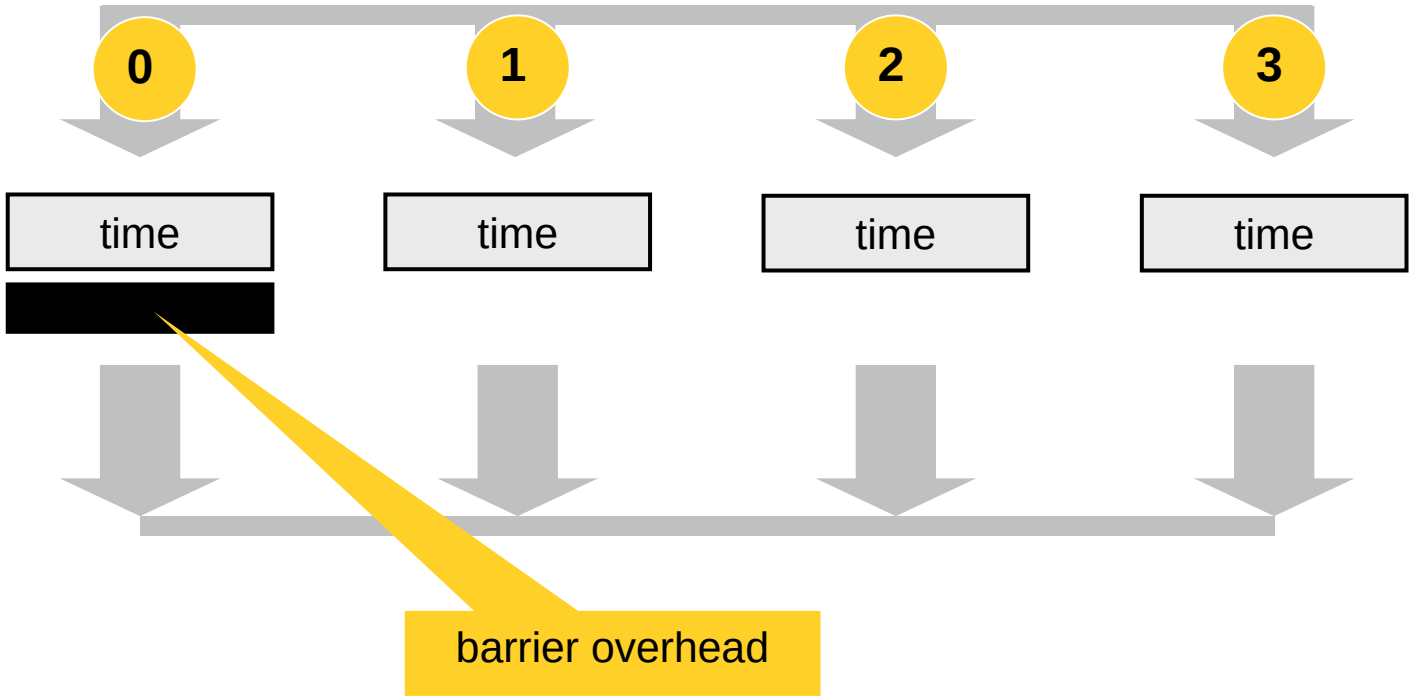
- Scalability and barrier overhead…



- Scalability and barrier overhead...

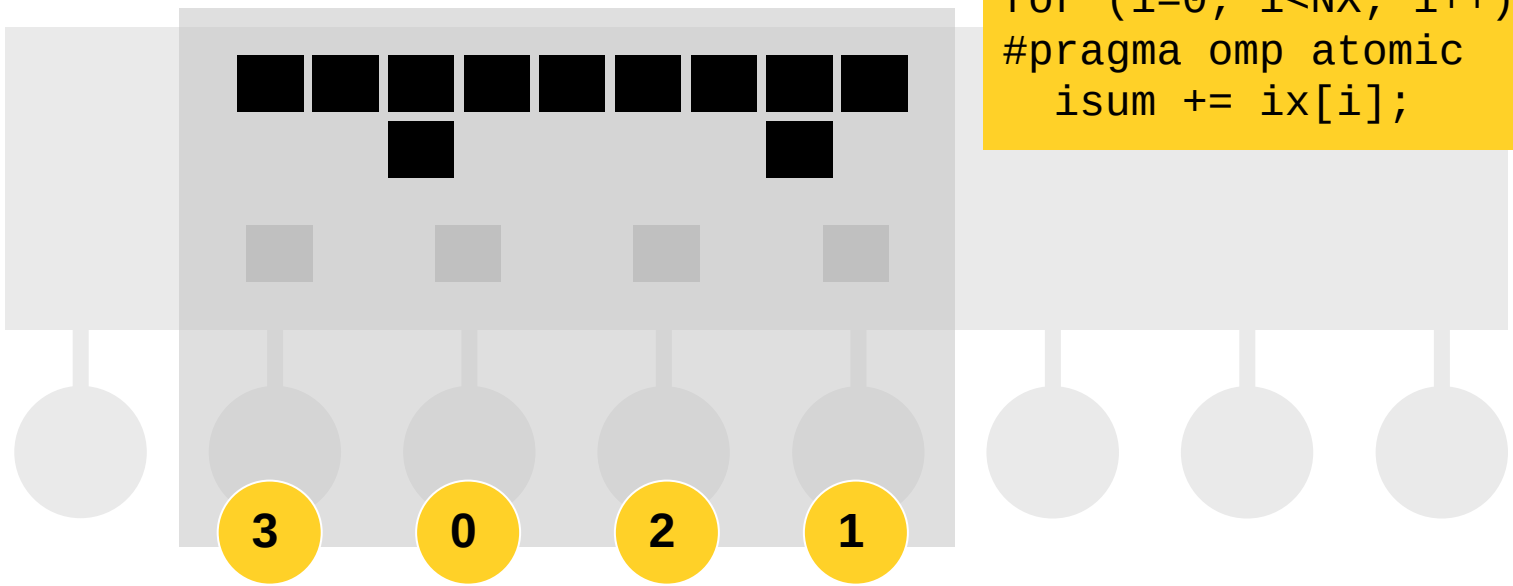


- Scalability and barrier overhead…



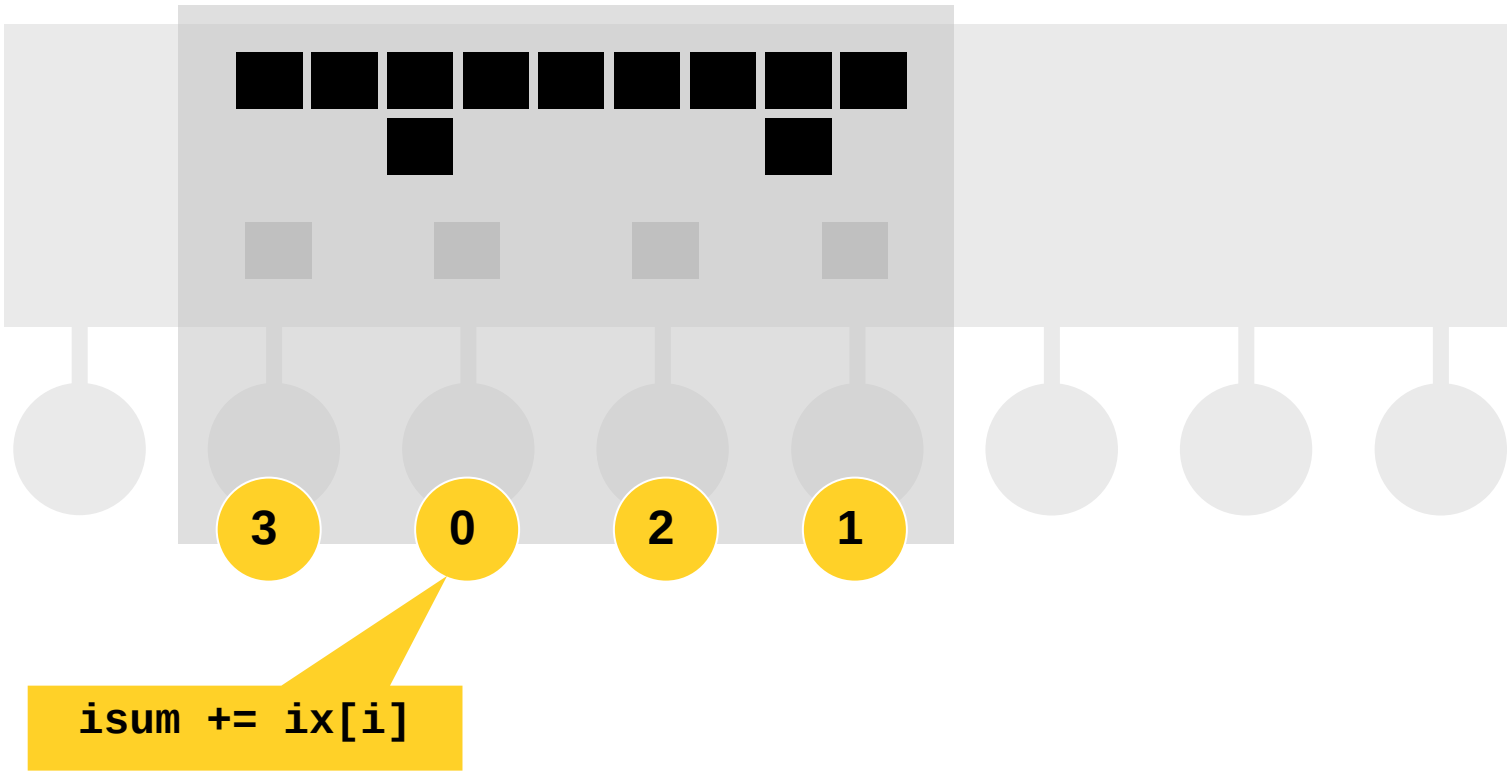
- Scalability and barrier overhead…
  - Non-deterministic
  - Increases with number of threads
  - On the order of a “fraction of a second” (SGI)
  - To make OpenMP worthwhile, the amount of work in the parallel region should be “much larger” than the barrier overhead… on the order of “tens of millions of floating point operations” (SGI)

- Scalability and synchronization…

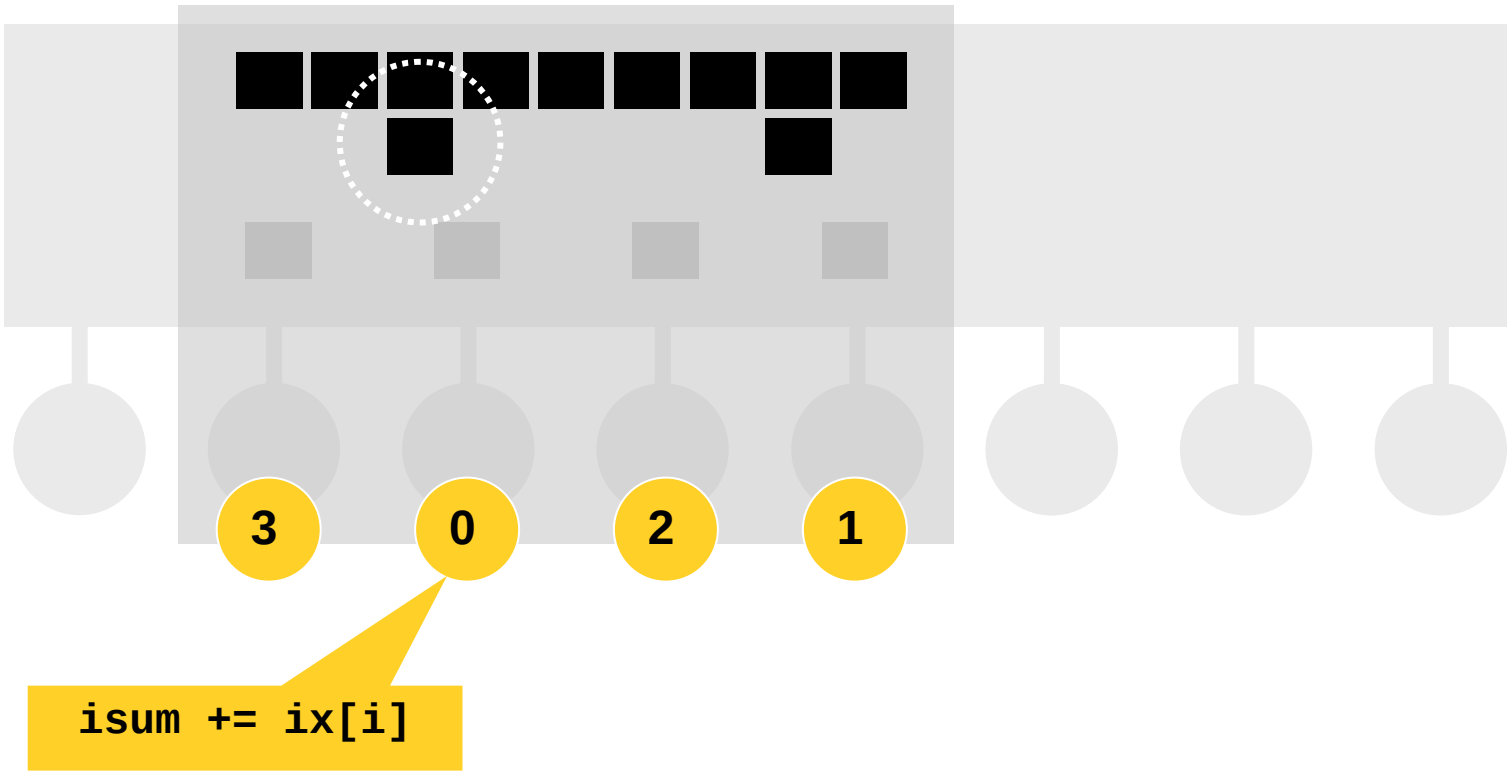



```
#pragma omp parallel for
for (i=0; i<NX; i++)
#pragma omp atomic
  isum += ix[i];
```

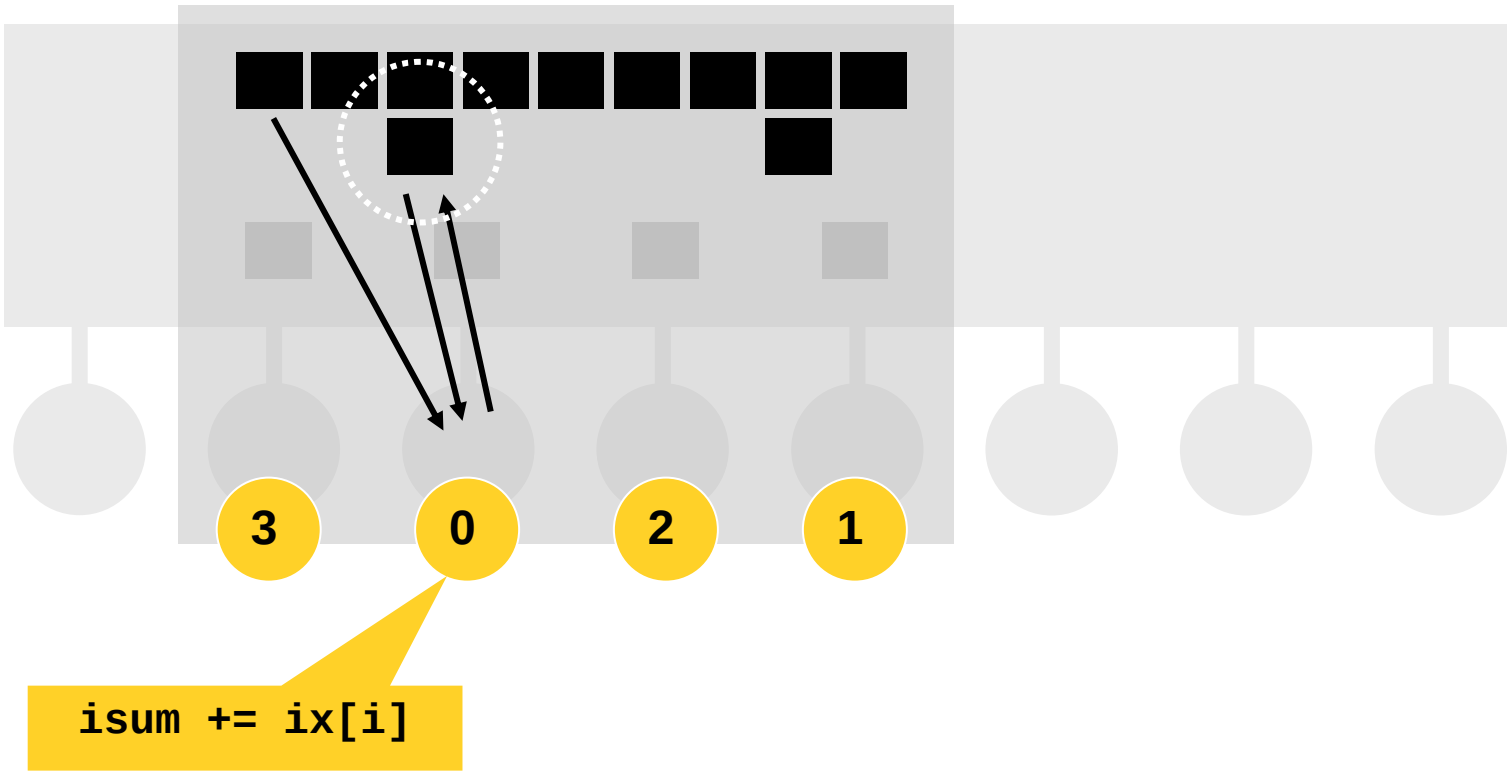
- Scalability and synchronization…



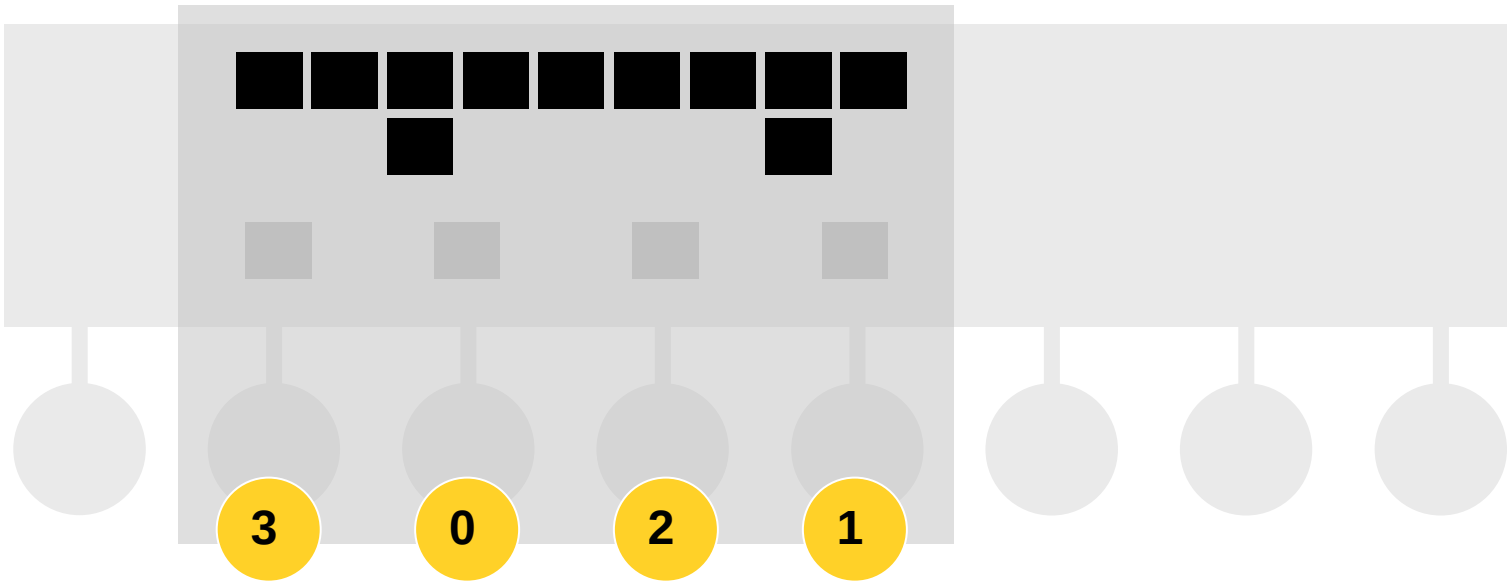
- Scalability and synchronization…



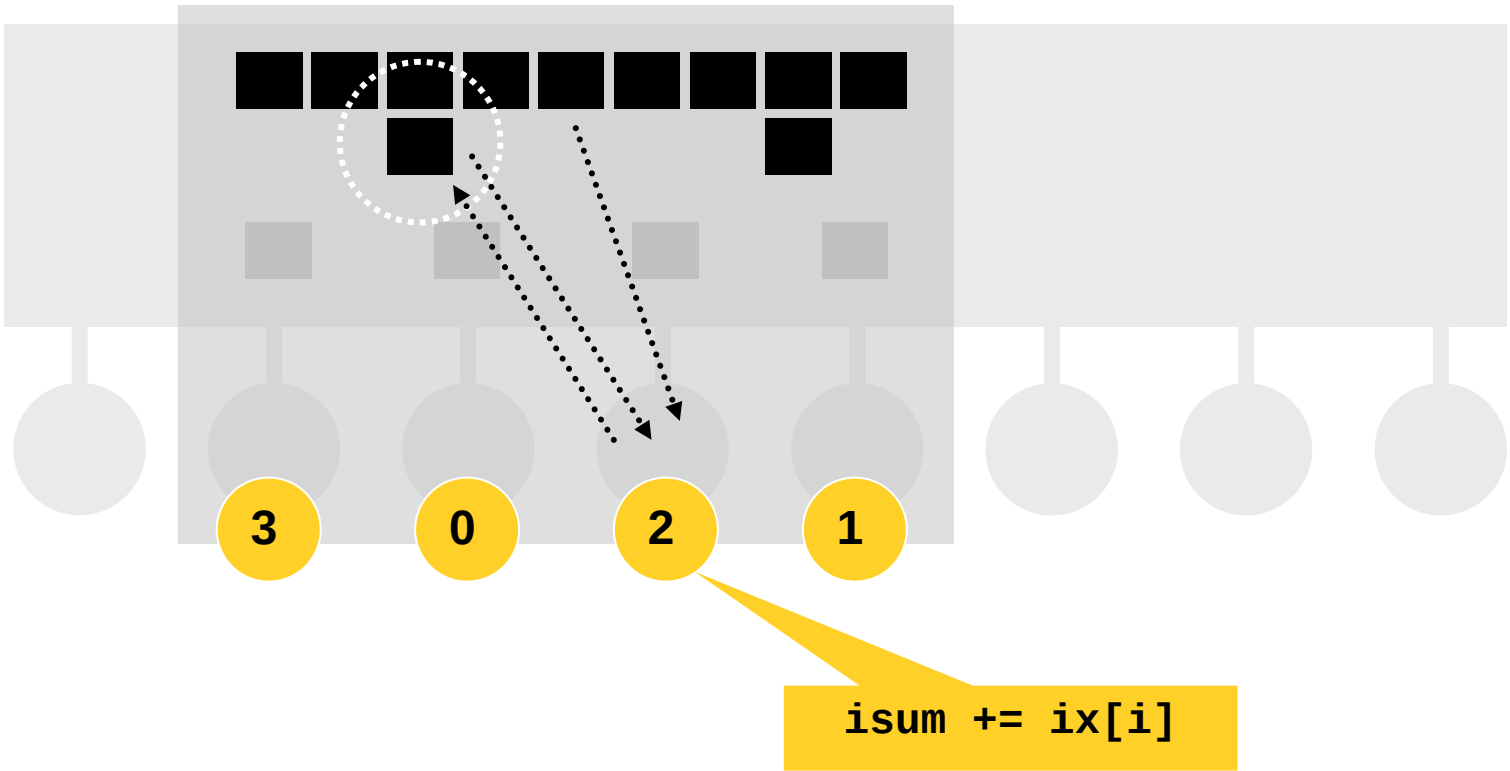
- Scalability and synchronization…



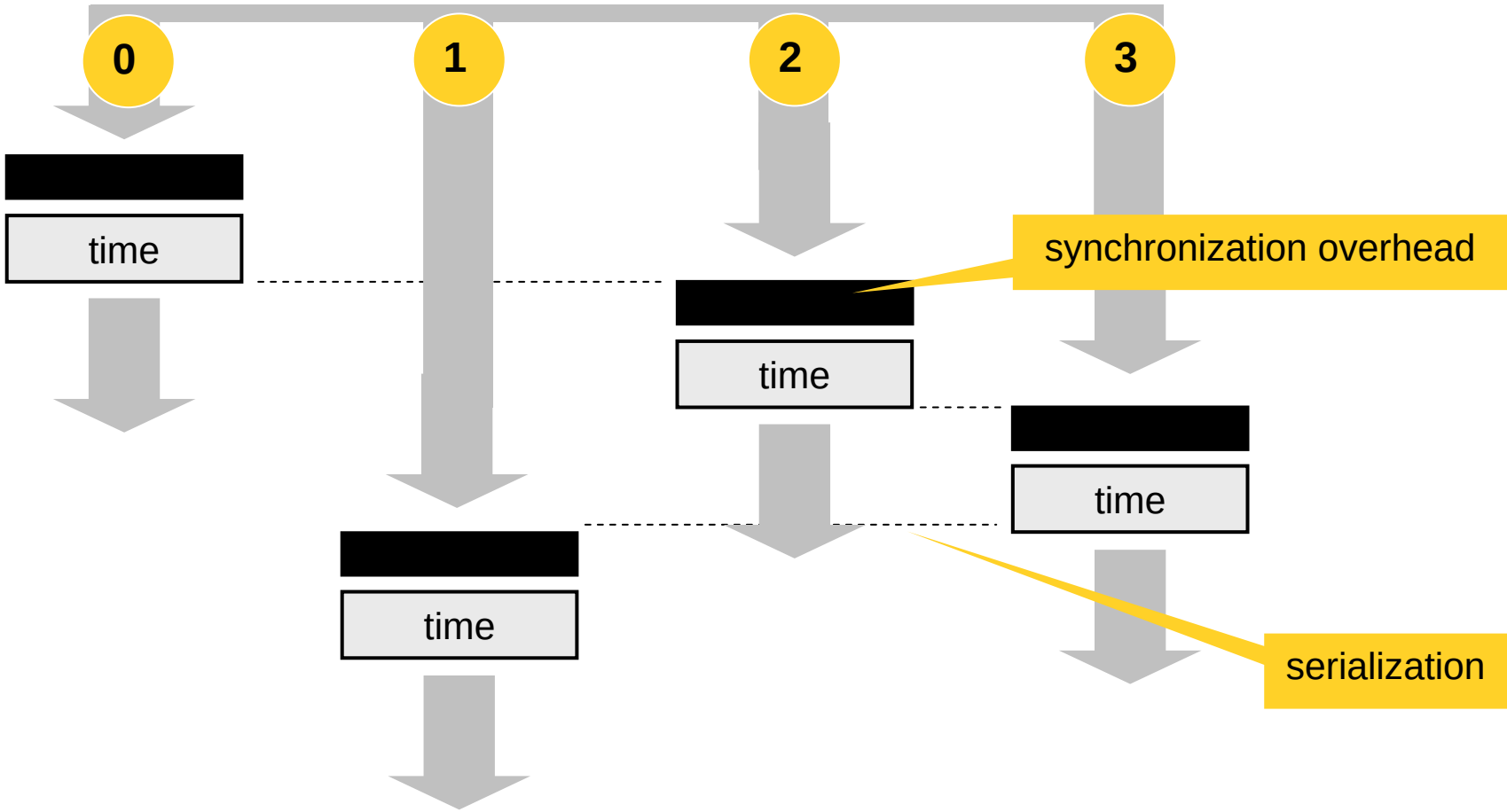
- Scalability and synchronization…



- Scalability and synchronization…

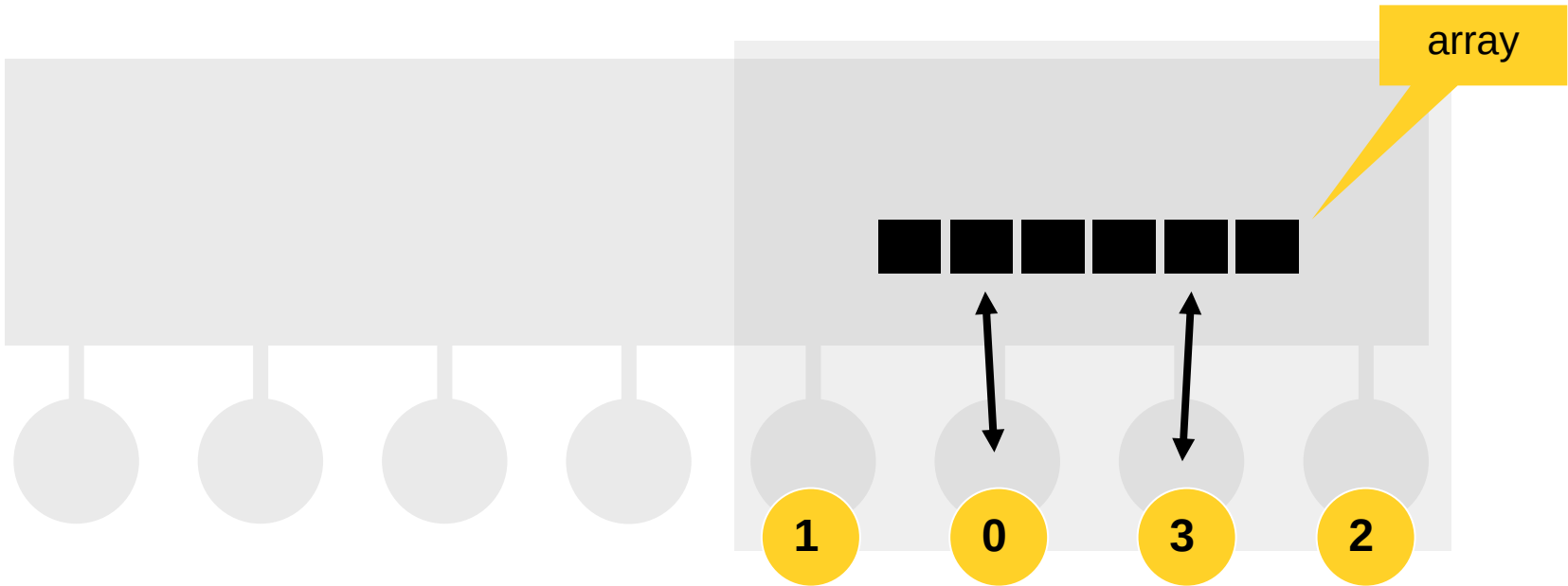


- Scalability and synchronization…

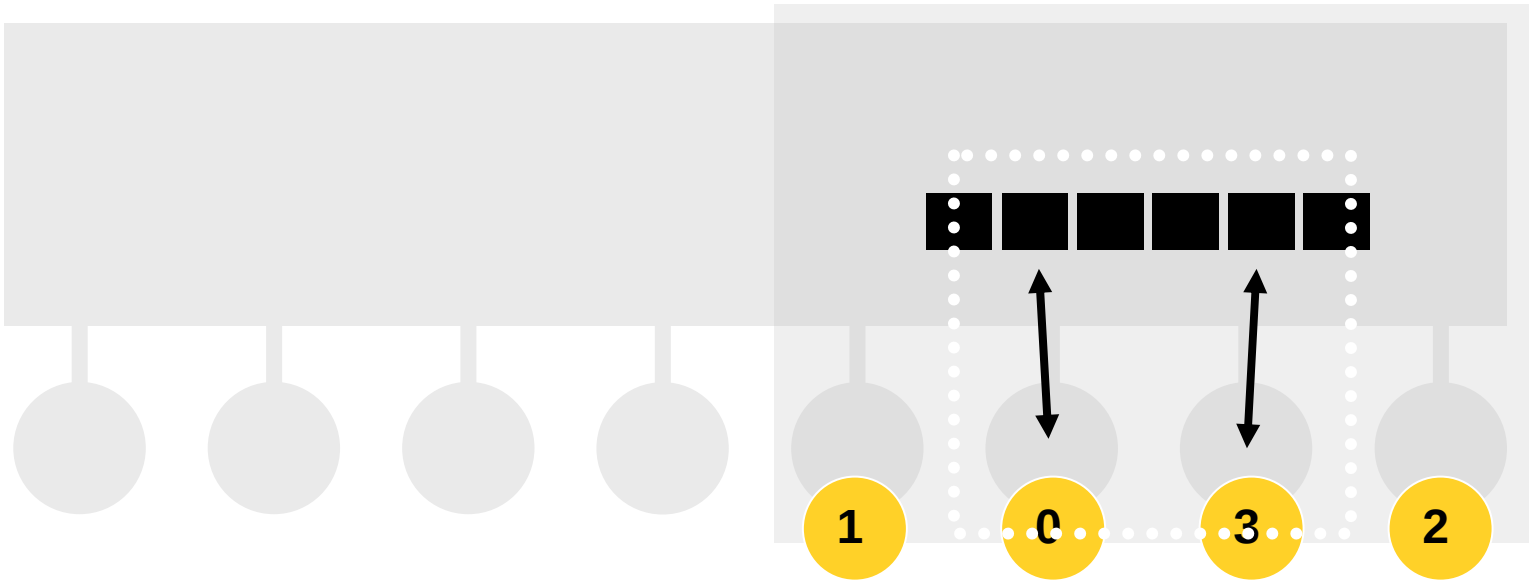


- Scalability and false sharing…
  - Symptomatic of modern hierarchical memory architectures
  - Impedes efficient memory handling

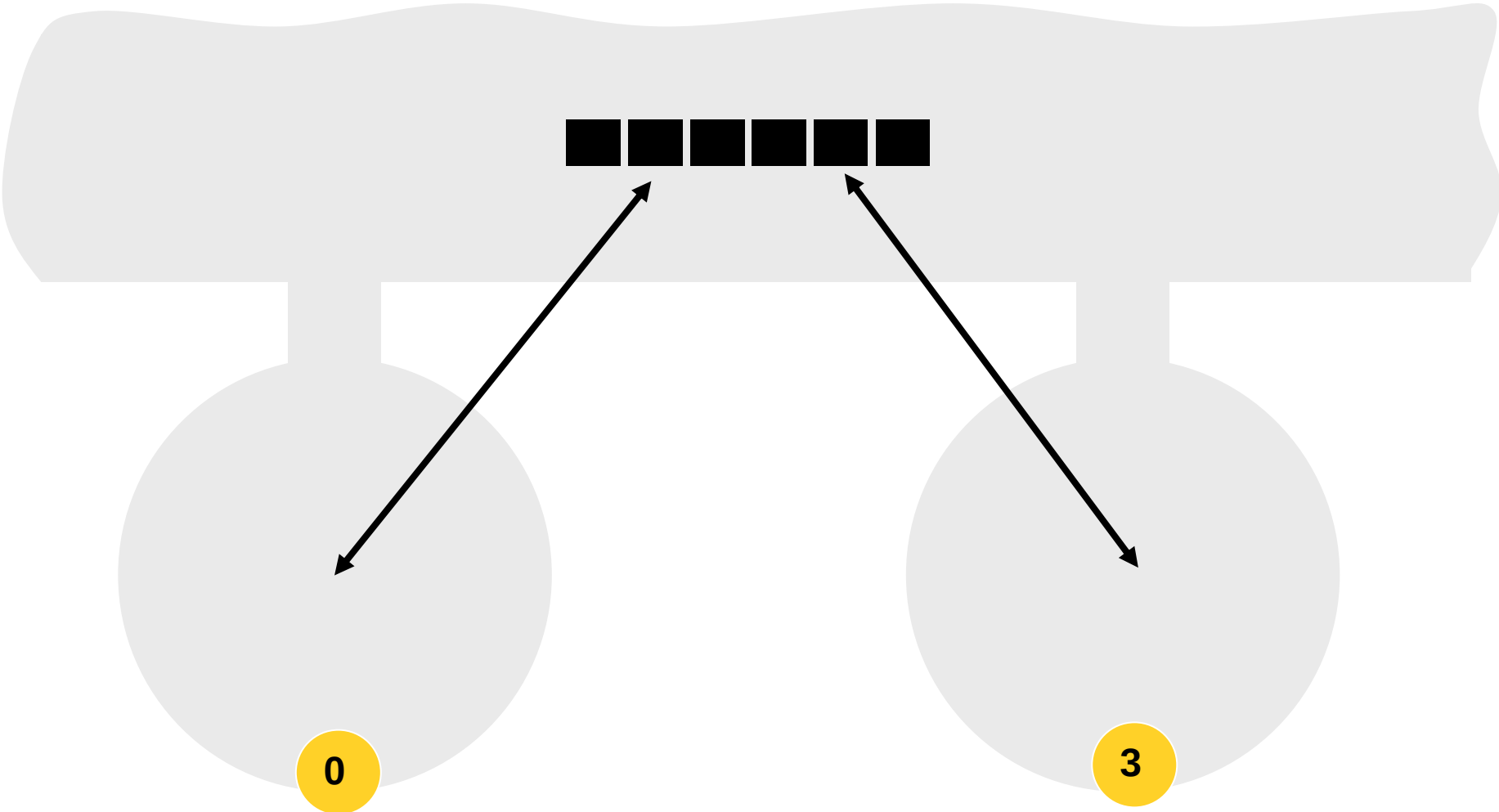
- Scalability and false sharing...



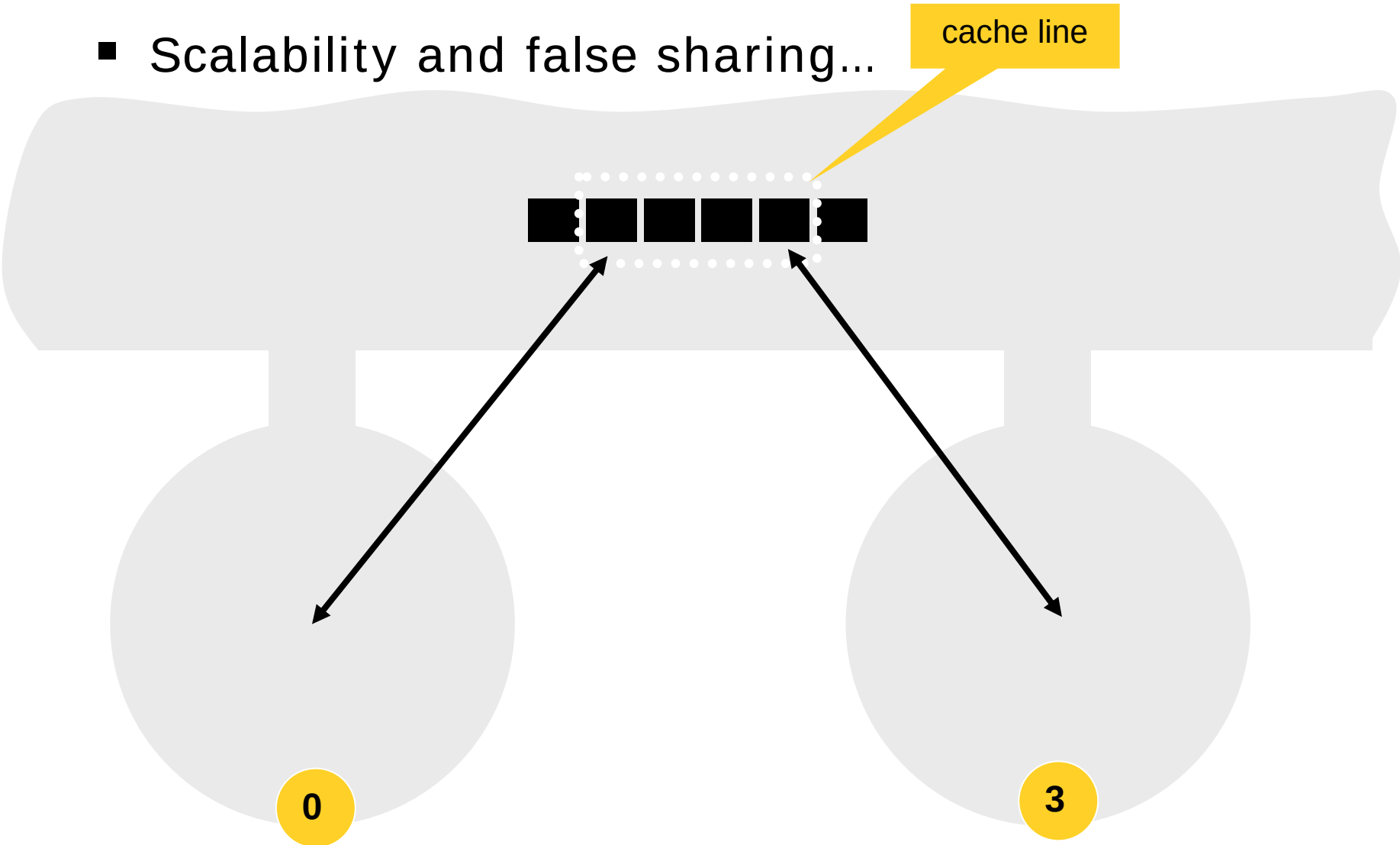
- Scalability and false sharing...



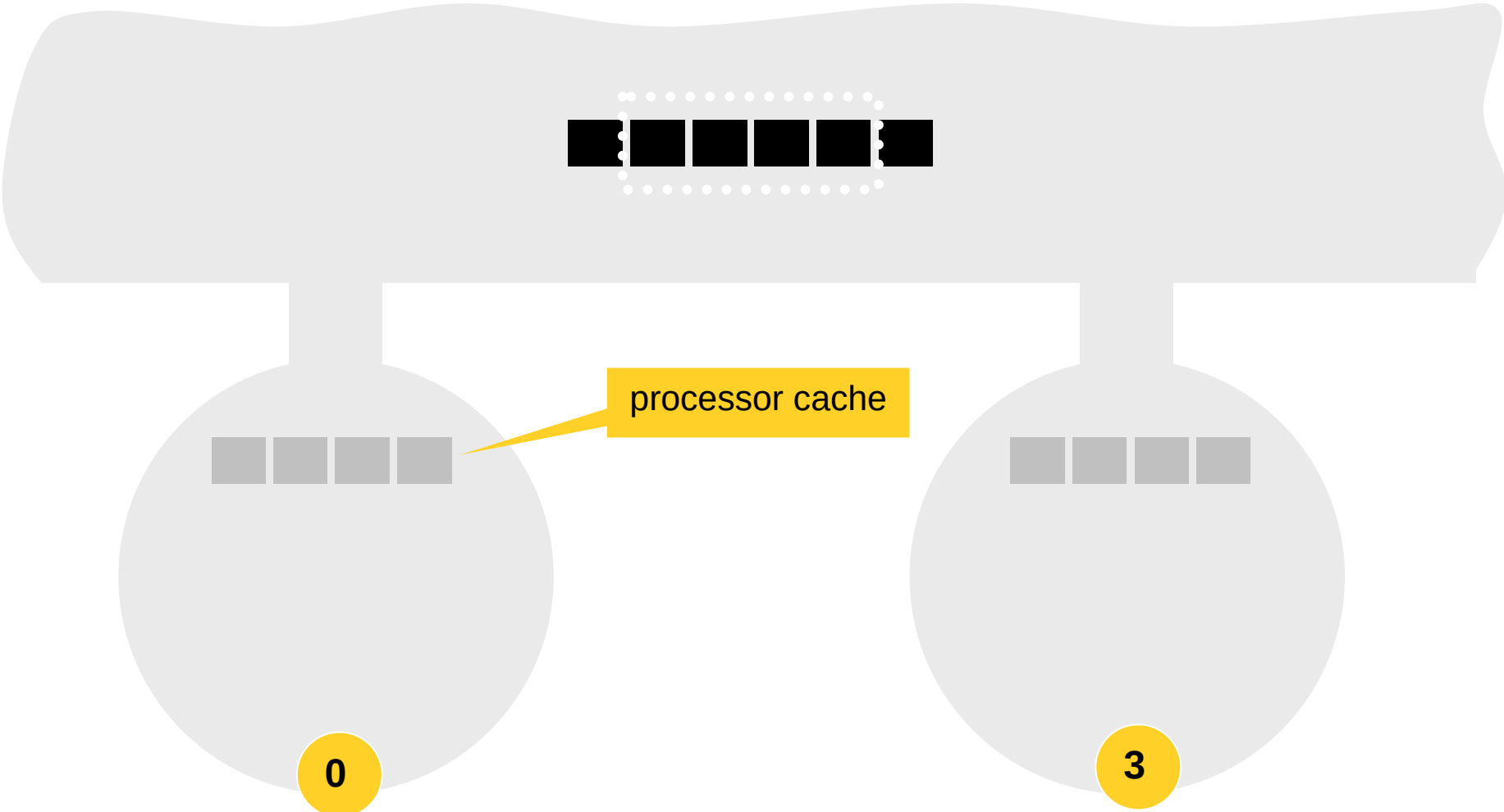
- Scalability and false sharing...



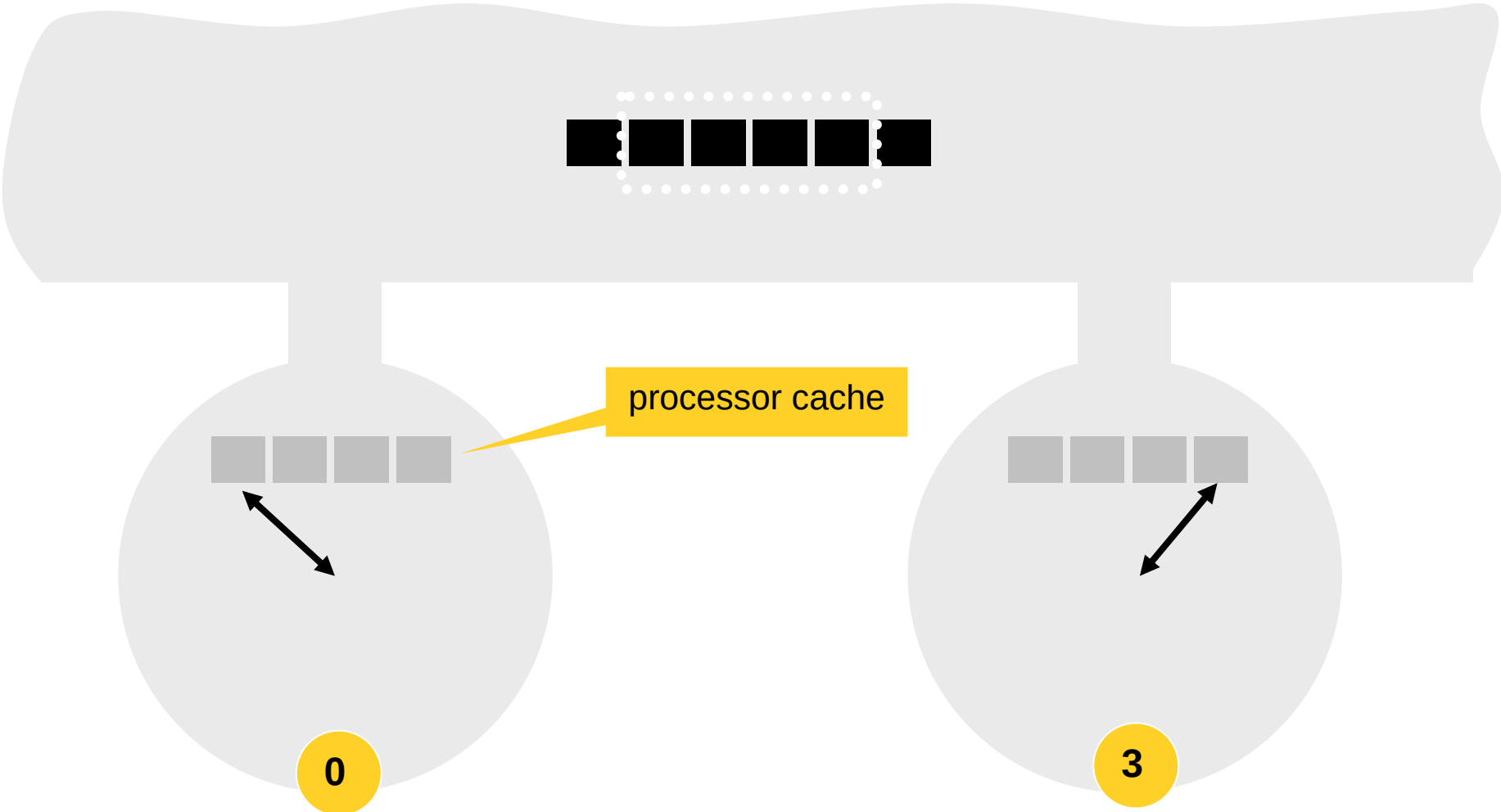
- Scalability and false sharing...



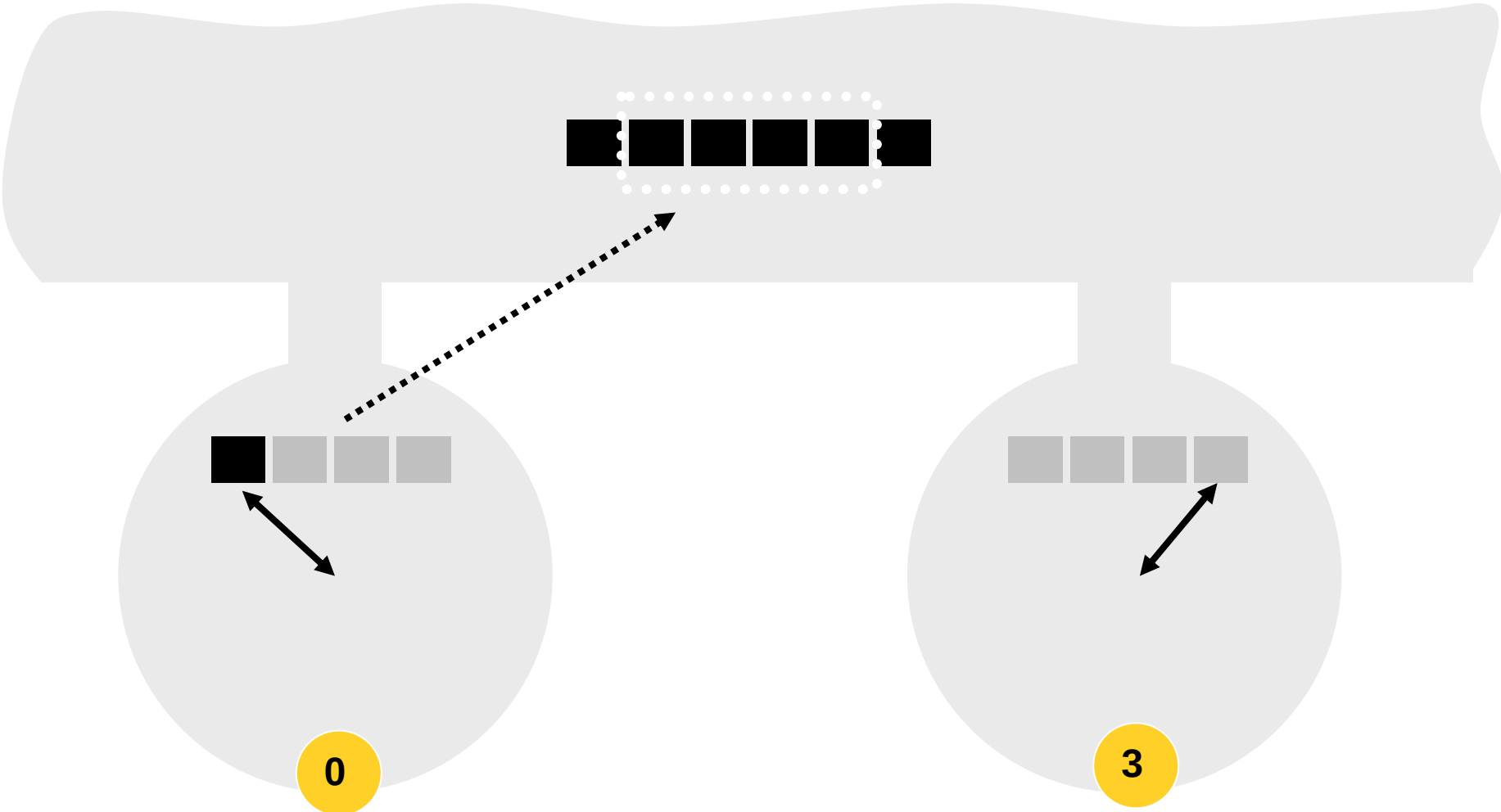
- Scalability and false sharing...



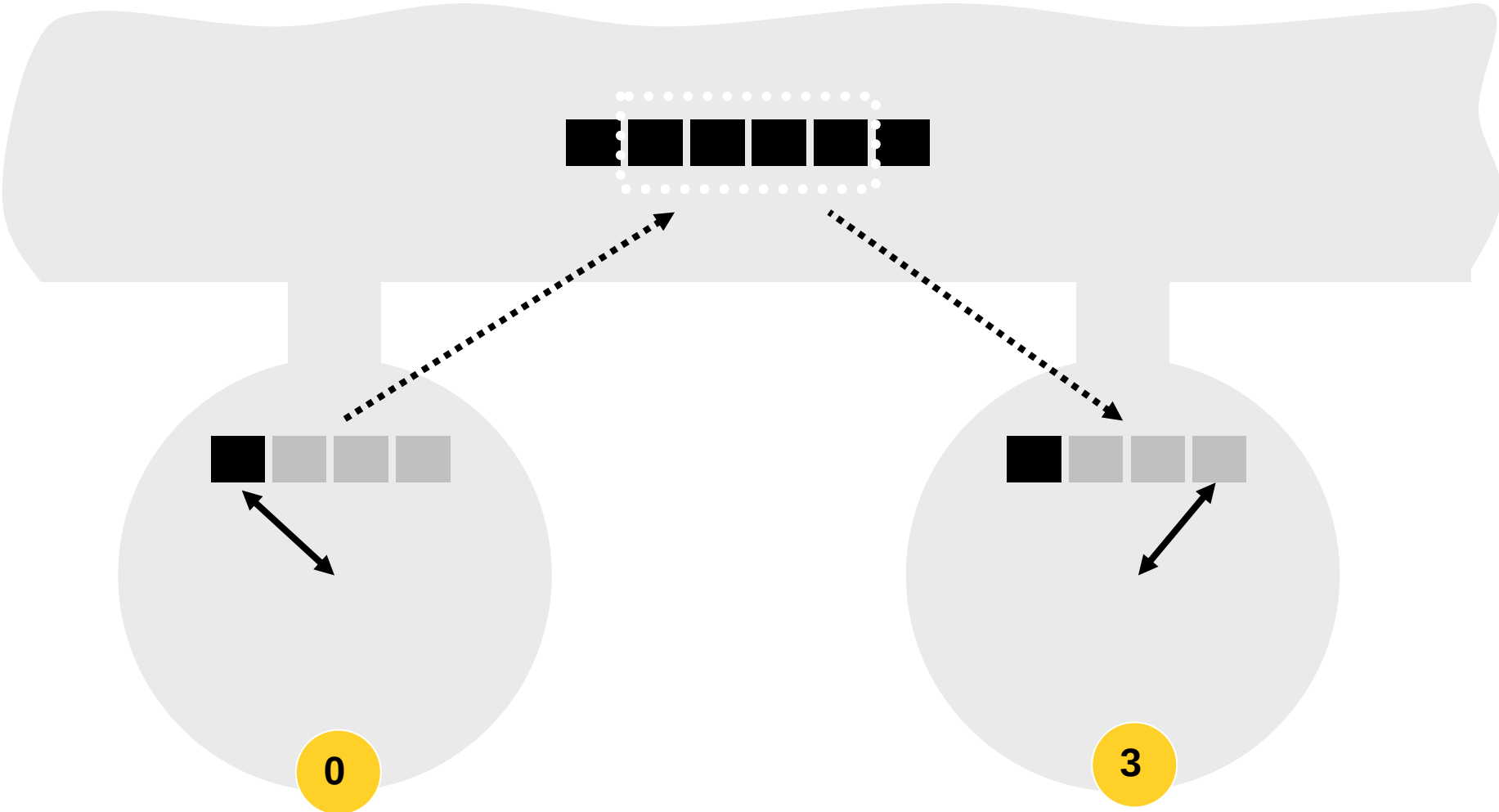
- Scalability and false sharing...



- Scalability and false sharing...



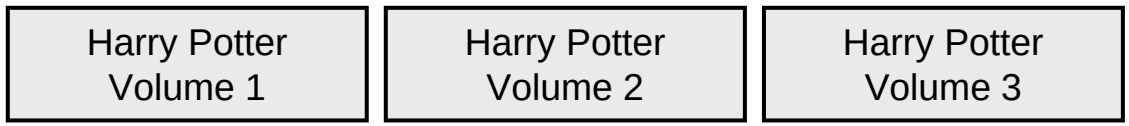
- Scalability and false sharing...



- OpenMP and MPI programming paradigms

- OpenMP and MPI programming paradigms
  - MPI… parallelizing data

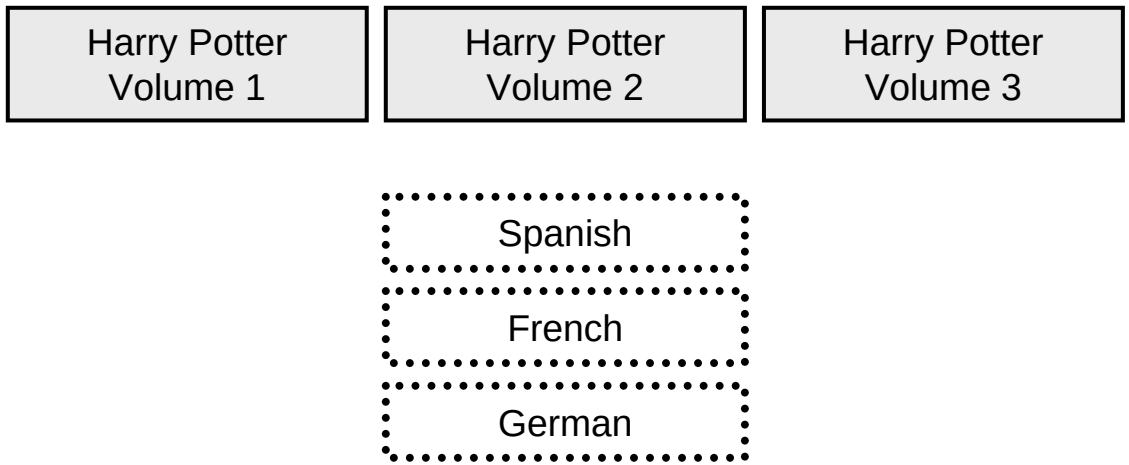
- OpenMP and MPI programming paradigms
  - MPI… parallelizing data
  - OpenMP… parallelizing tasks

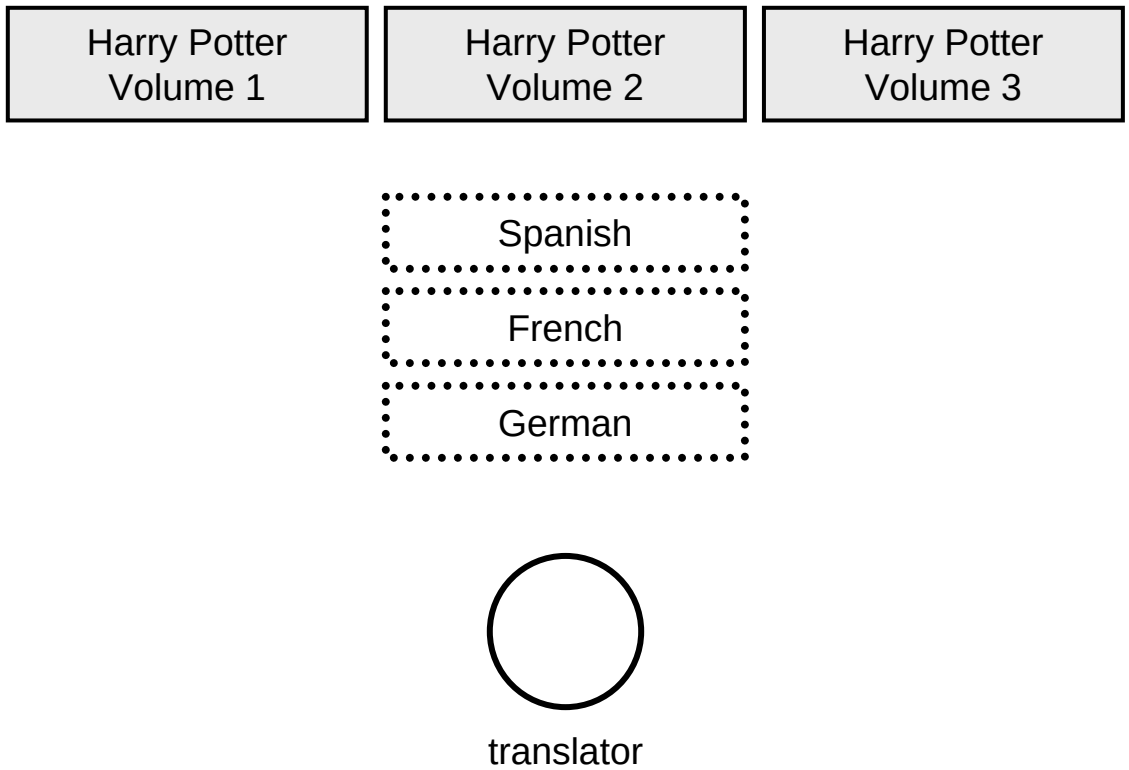


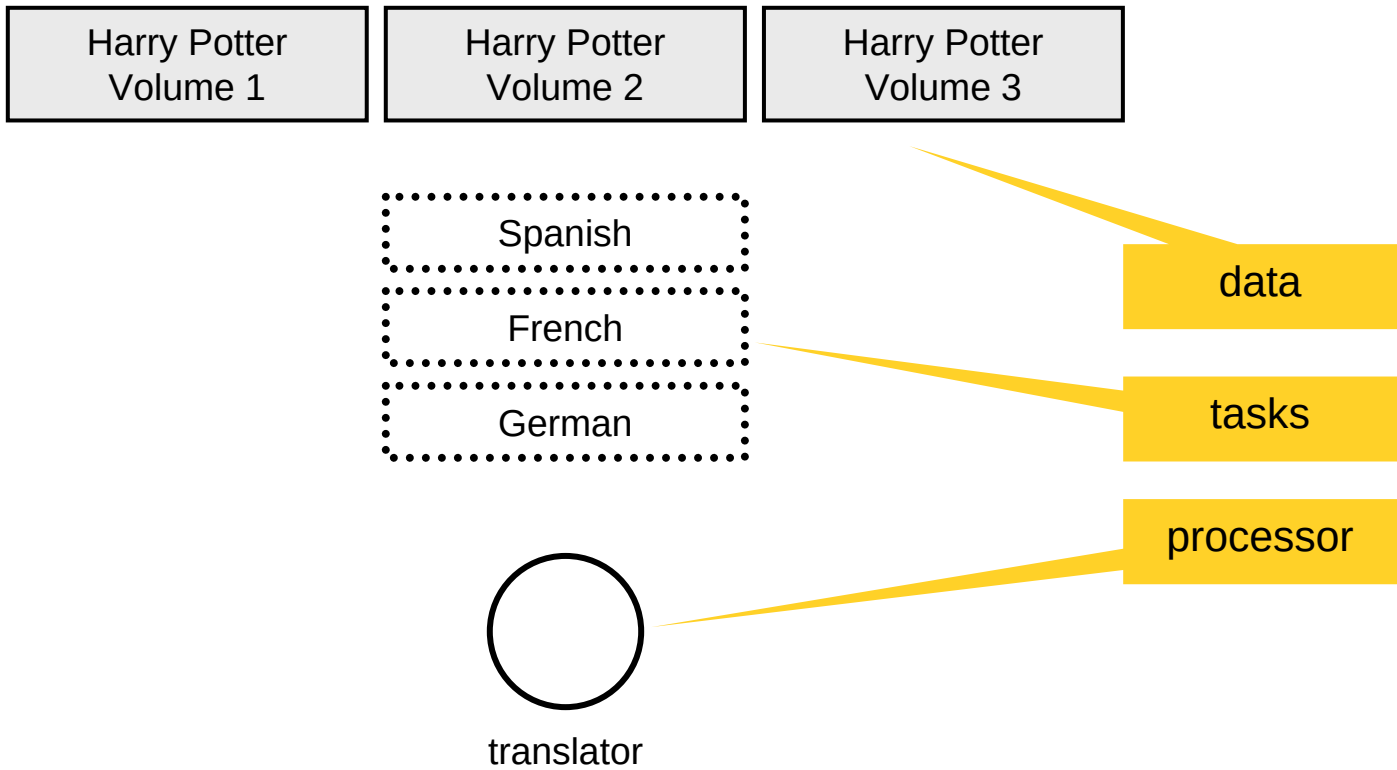


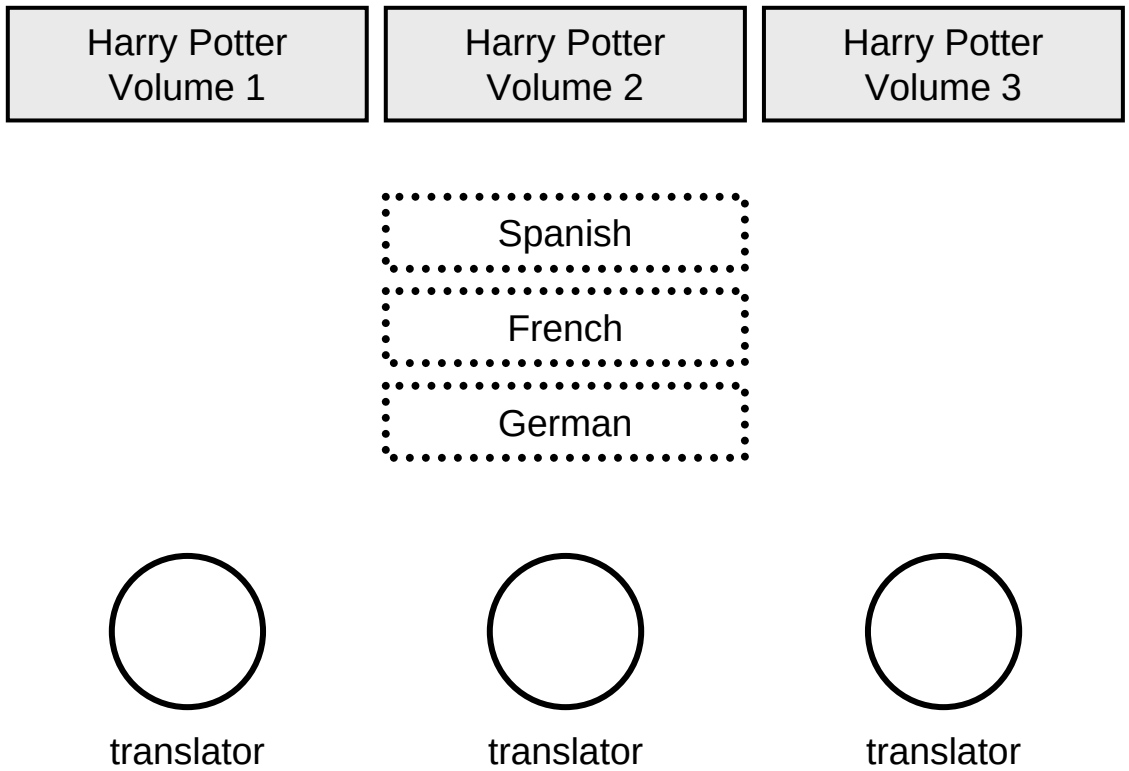


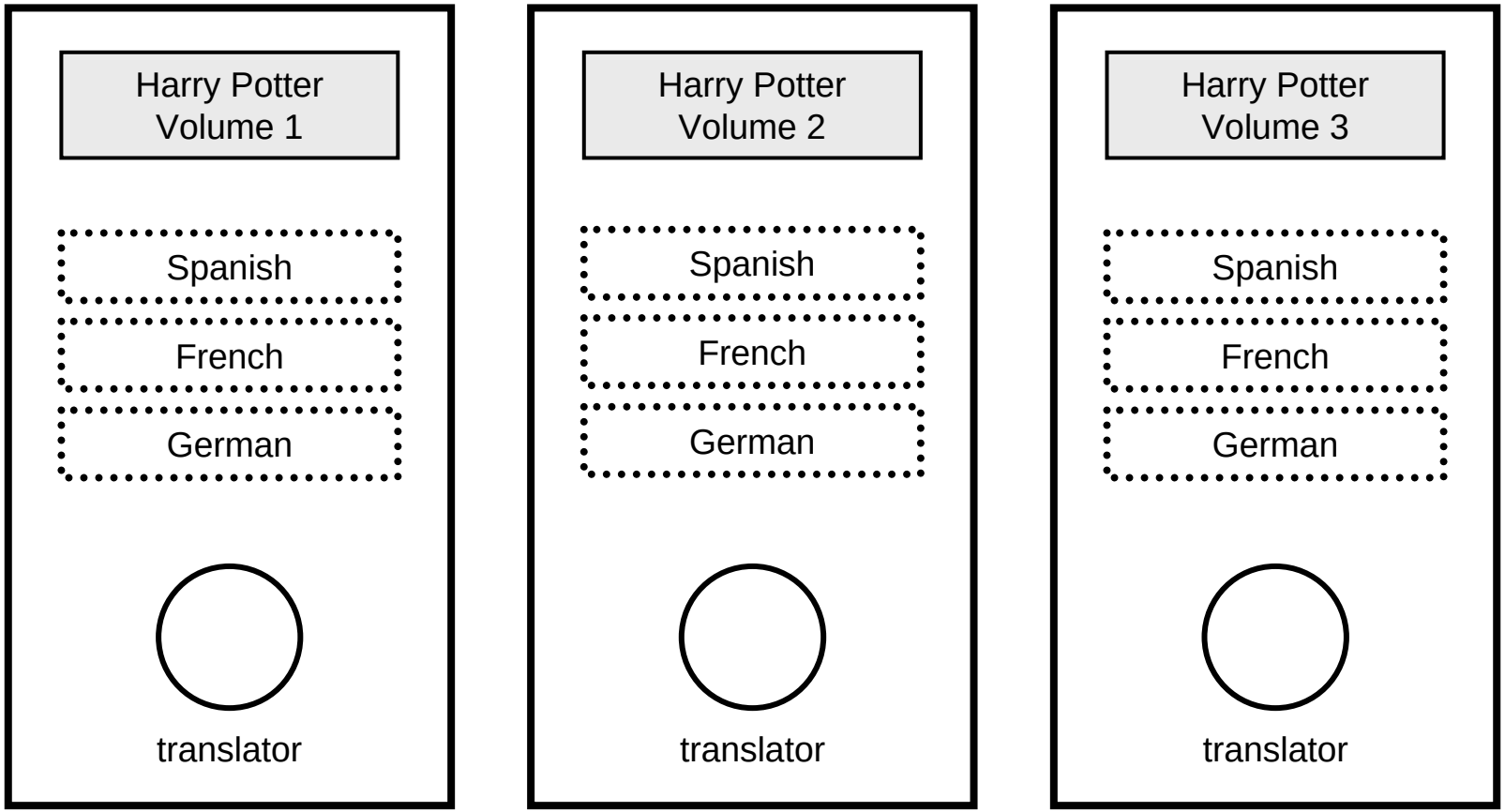
create delete

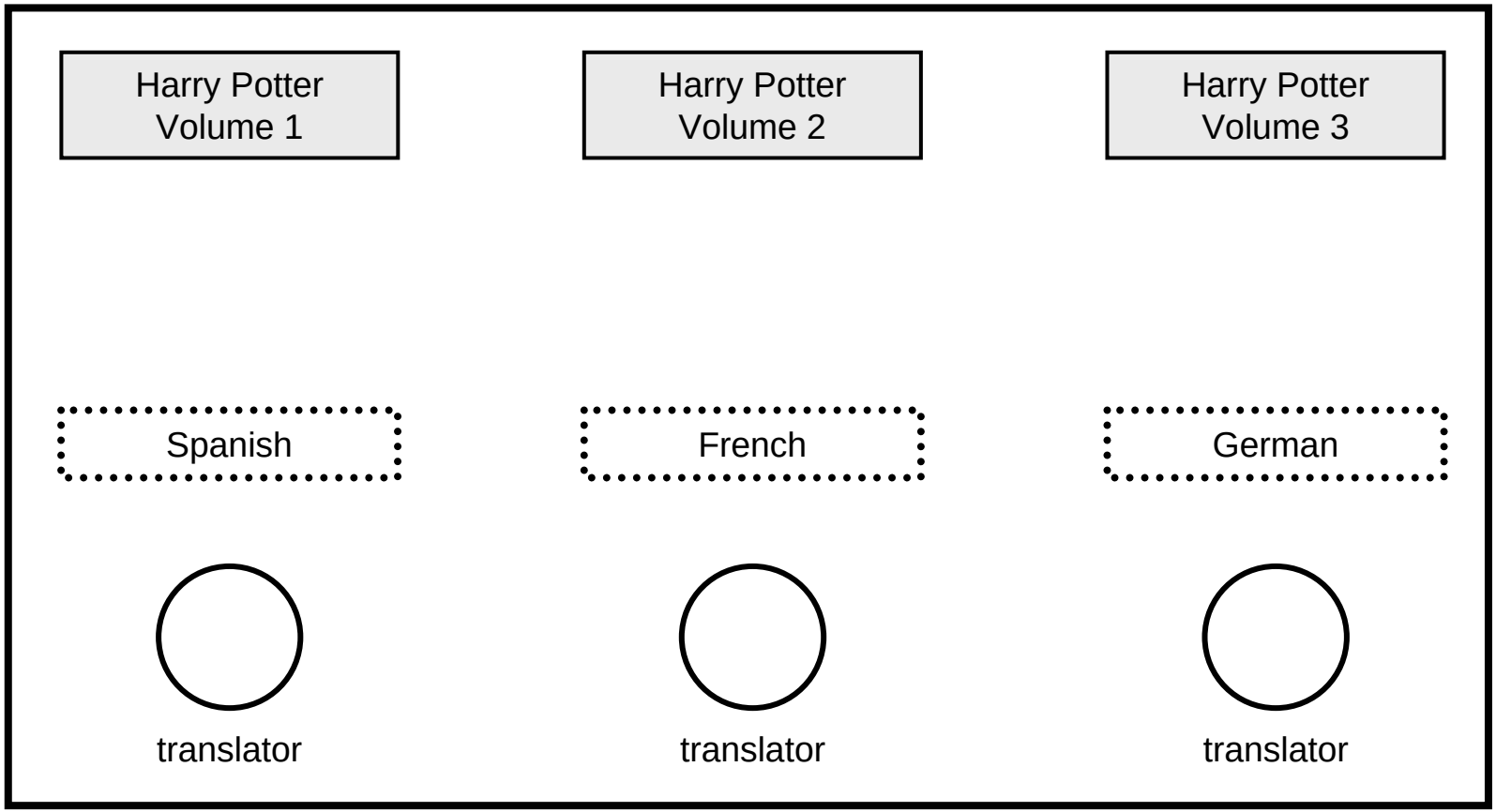













OpenMP

```
/* cc -o pi-mpi pi-mpi.c -lmpi */
/* setenv MPI_UNBUFFERED_STDIO */
/* mpirun -np <numprocs> ./pi-mpi */

#include <stdio.h>
#include <math.h>
#include <mpi.h>

int
main (int argc, char *argv[])
{
  int n, myid, numprocs, i;
  double PI25DT = 3.14159265358979323846264643;
  double mypi, pi, h, sum, x;

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);

  while (1) {
    if (myid == 0) {
      printf("Enter the number of intervals: (0 quits) ");
      scanf("%d", &n);
    }

    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (n == 0) {
      break;
    } else {
      h = 1.0/(double)n;
      sum = 0.0;

      for (i=myid+1; i<=n; i += numprocs) {
        x = h*((double)i - 0.5);
        sum += (4.0/(1.0 + x*x));
      }

      mypi = h*sum;

      MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

      if (myid == 0)
        printf("pi estimate %.16f, Error %.16f\n", pi, fabs(pi - PI25DT));
    }
  }

  MPI_Finalize();

  return 0;
}
```

```
/* cc -mp -o pi-omp pi-omp.c */
/* setenv OMP_NUM_THREADS numprocs */
/* ./pi-omp */

#include <stdio.h>
#include <math.h>
#include <omp.h>

int
main (int argc, char *argv[])
{
  int n;
  double PI25DT = 3.14159265358979323846264 3;
  double pi, sum;

#pragma omp parallel default(shared)
  {
    int i;
    double h, x;

    while (1) {
#pragma omp master
      {
        printf("Enter the number of intervals: (0 quits) ");
        scanf("%d", &n);
      }

#pragma omp barrier

      if (n == 0) {
        break;
      } else {

        h = 1.0/(double)n;
        sum = 0.0;

#pragma omp for reduction(+: sum) schedule(runtime)
        for (i=1; i<=n; i++) {
          x = h*((double)i - 0.5);
          sum += (4.0/(1.0 + x*x));
        }

        pi = h*sum;

#pragma omp master
        {
          printf("pi estimate %.16f, Error %.16f\n", pi, fabs(pi - PI25DT));
        }
      }
    }
  }/* end of parallel region */

  return 0;
}
```





OpenMP

- For more information…
  - http://www.openmp.org
  - “Parallel Programming in OpenMP” by Chandra et al
  - http://www.ualberta.ca/CNS/RESEARCH/Courses/archivecourses.html

- Questions...