Task Parallelism

Prof. Guido Araujo

www.ic.unicamp.br/~guido

Agenda

- Task parallelism: programming model
- Where are the limitations?

Agenda

- Task Parallelism: programming model
- Where are the limitations?

Task Parallelism

The OpenMP 4.0 task programming model

```
for (int i=1, j=1;  i<N;  i++)  {

      #pragma omp task in(v[i-1]) out(v[i])
       fun1(&v[i-1],  &v[i]);

       for (int k=0;  k<i;  k++,  j++)  {

    #pragma omp task in(v[i]) out(u[j])
            fun2(&v[i],   &u[j]);
       }

       fun3( 3 * i );
}
```

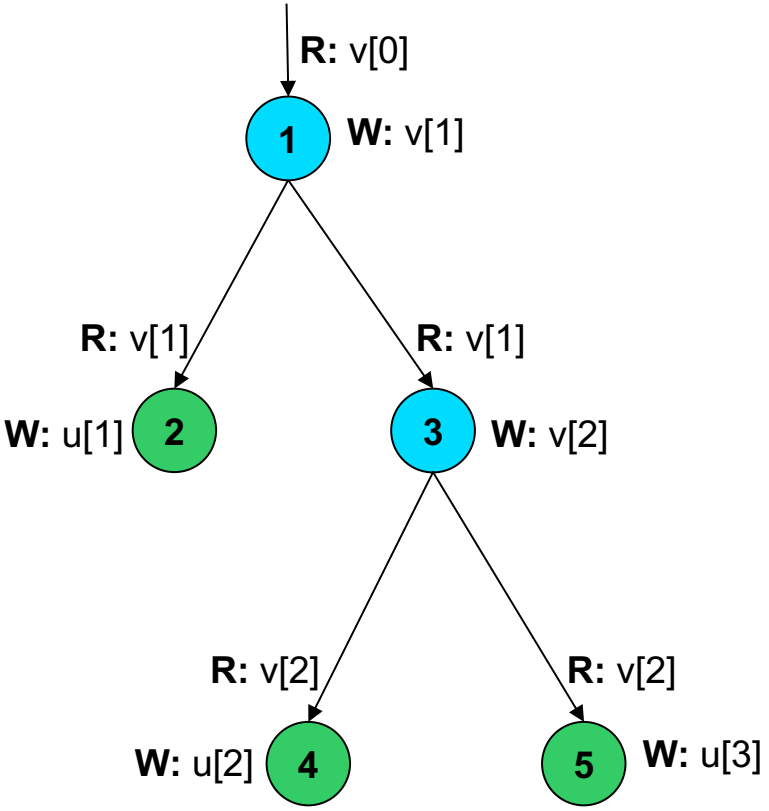
Create a task!

Create a task!

Create NO tasks!

Task Parallelism

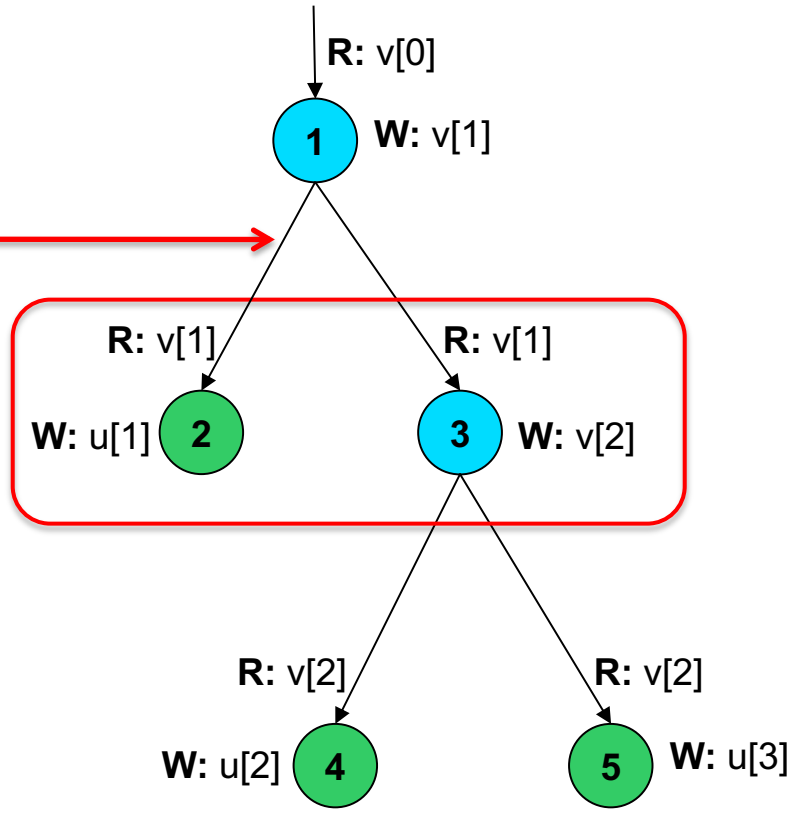
```
for (int i=1, j=1;  i<N;  i++)  {

     #pragma omp task in(v) out(v)
      fun1(&v[i-1],  &v[i]);

      for (int k=0;  k<i;  k++,  j++)  {

   #pragma omp task in(v) out(u)
        fun2(&v[i],   &u[j]);
      }

      fun3( 3 * i );
}
```



Task Parallelism Advantages

No need to communicate data. Just detect de dependence

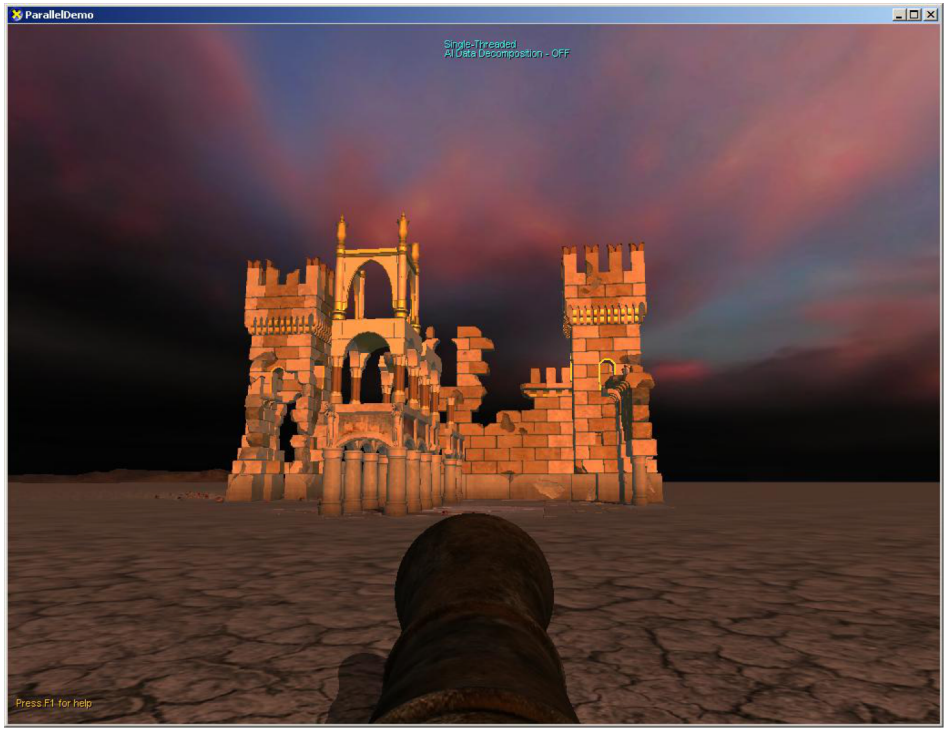
Parallelism between interations of different loops



But graphs are not always that simple!

Task parallelism in action

- Destroy the Castle
  - Developed by Intel using OpenMP 3.X
  - Game based on DirectX 8.1

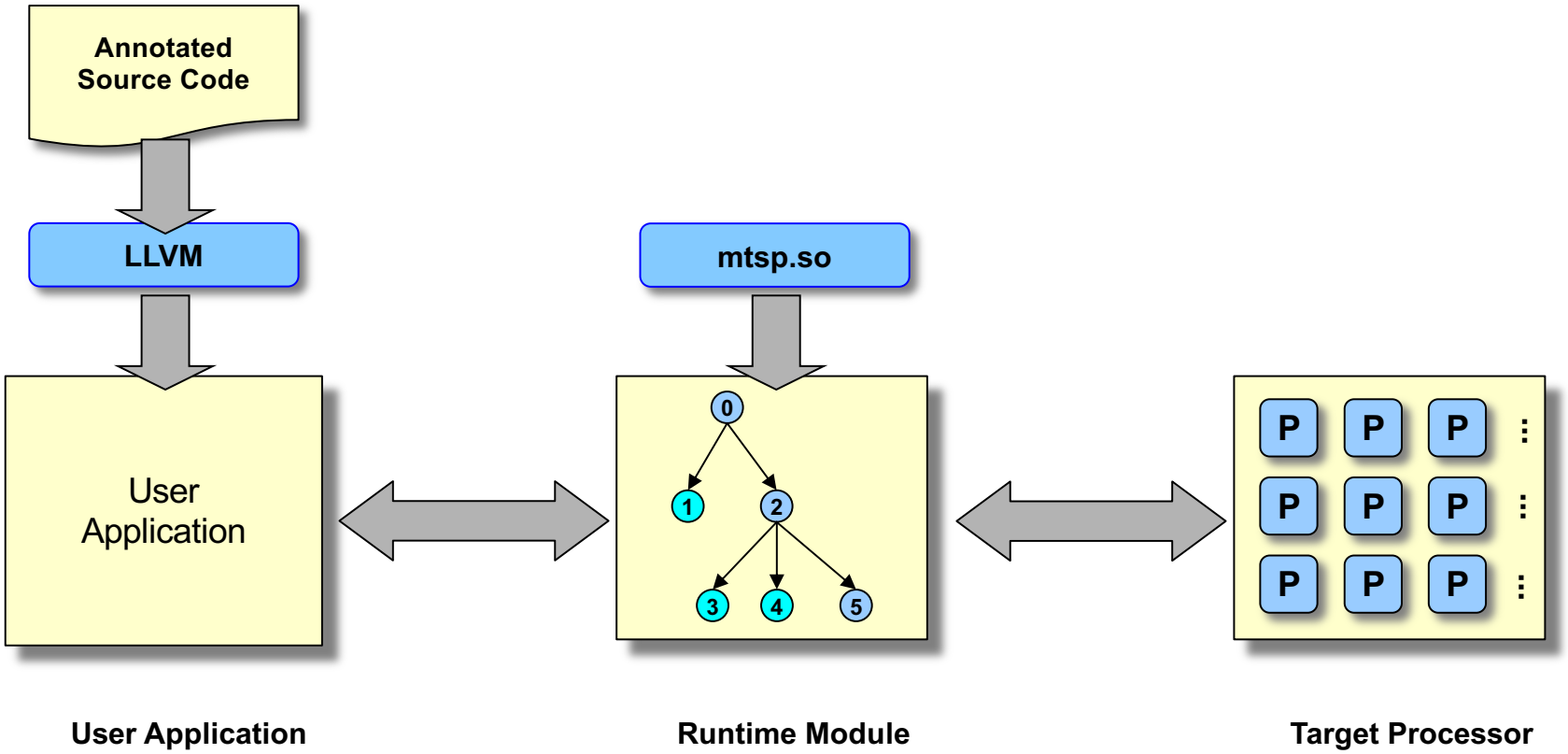


Task parallelism in action

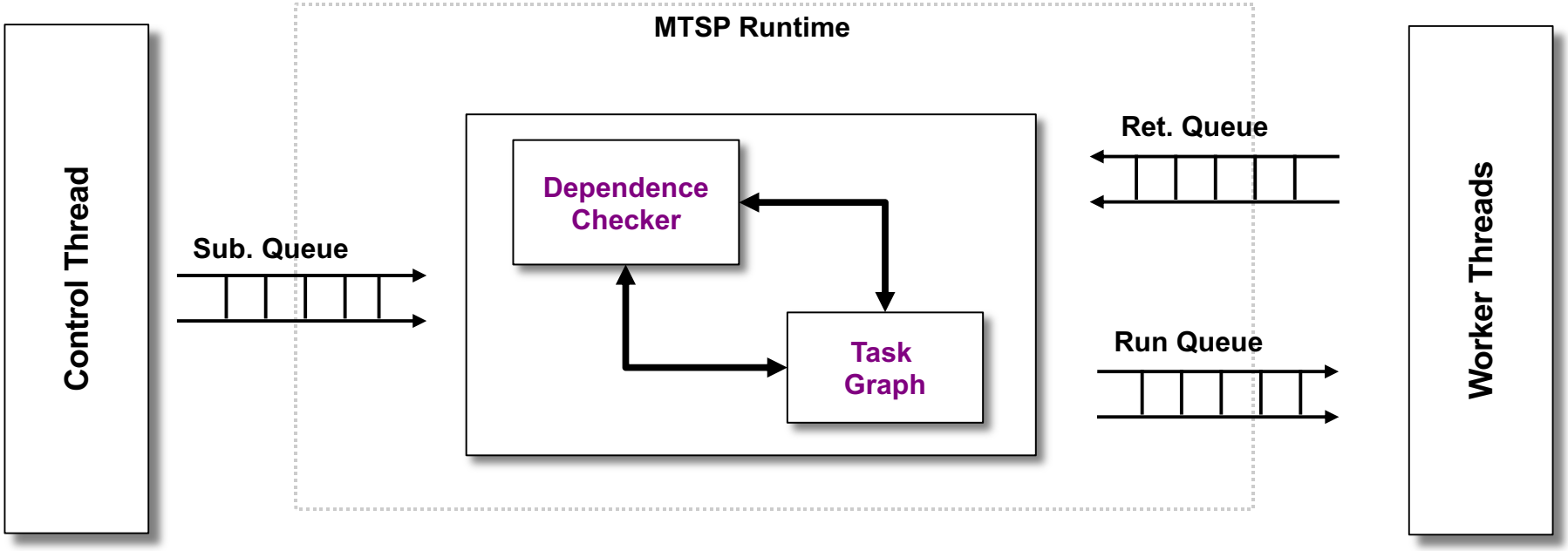
Agenda

- Task parallelism: programming model
- Where are the limitations?

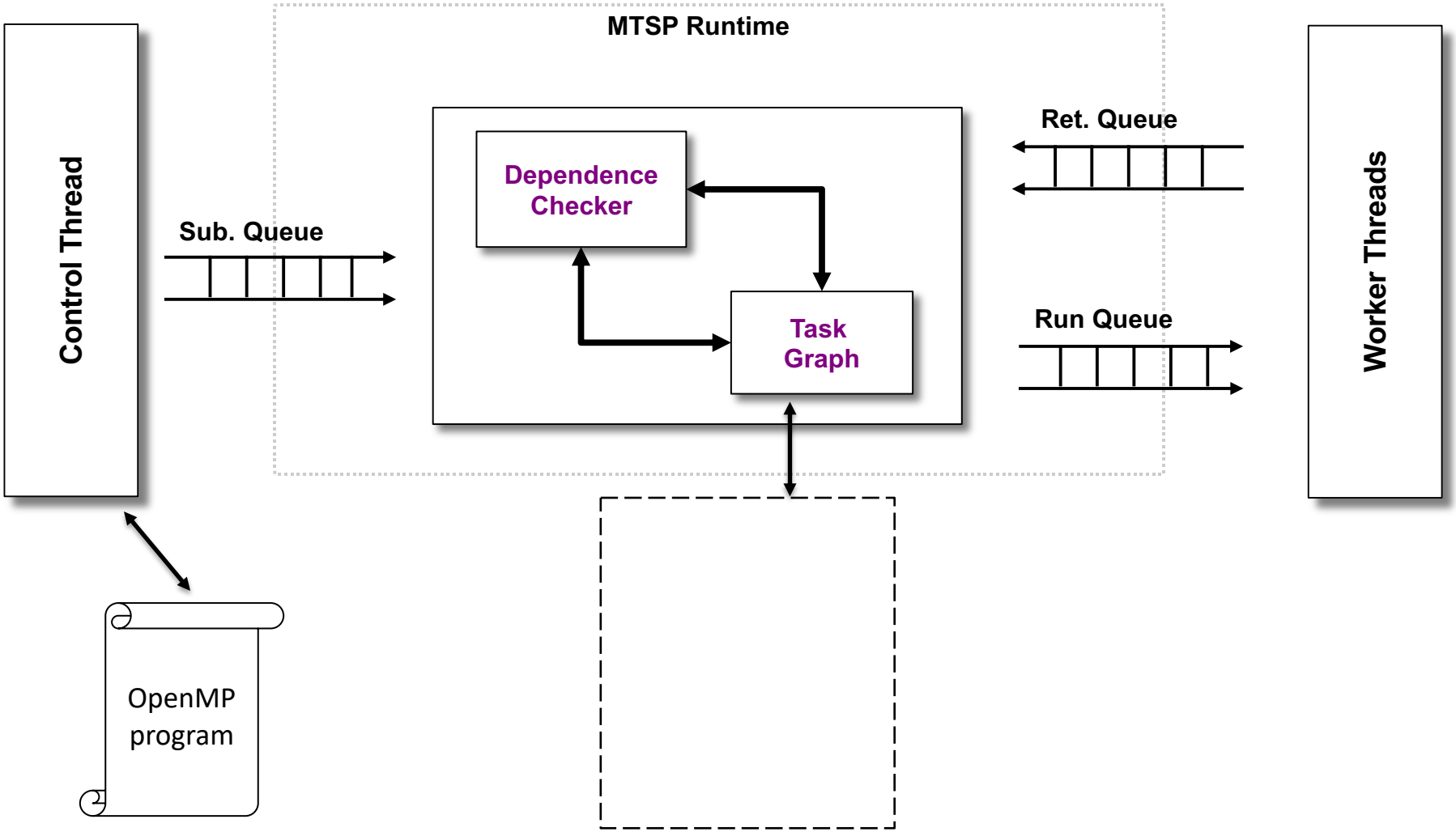
Basic elements of a task scheduling system



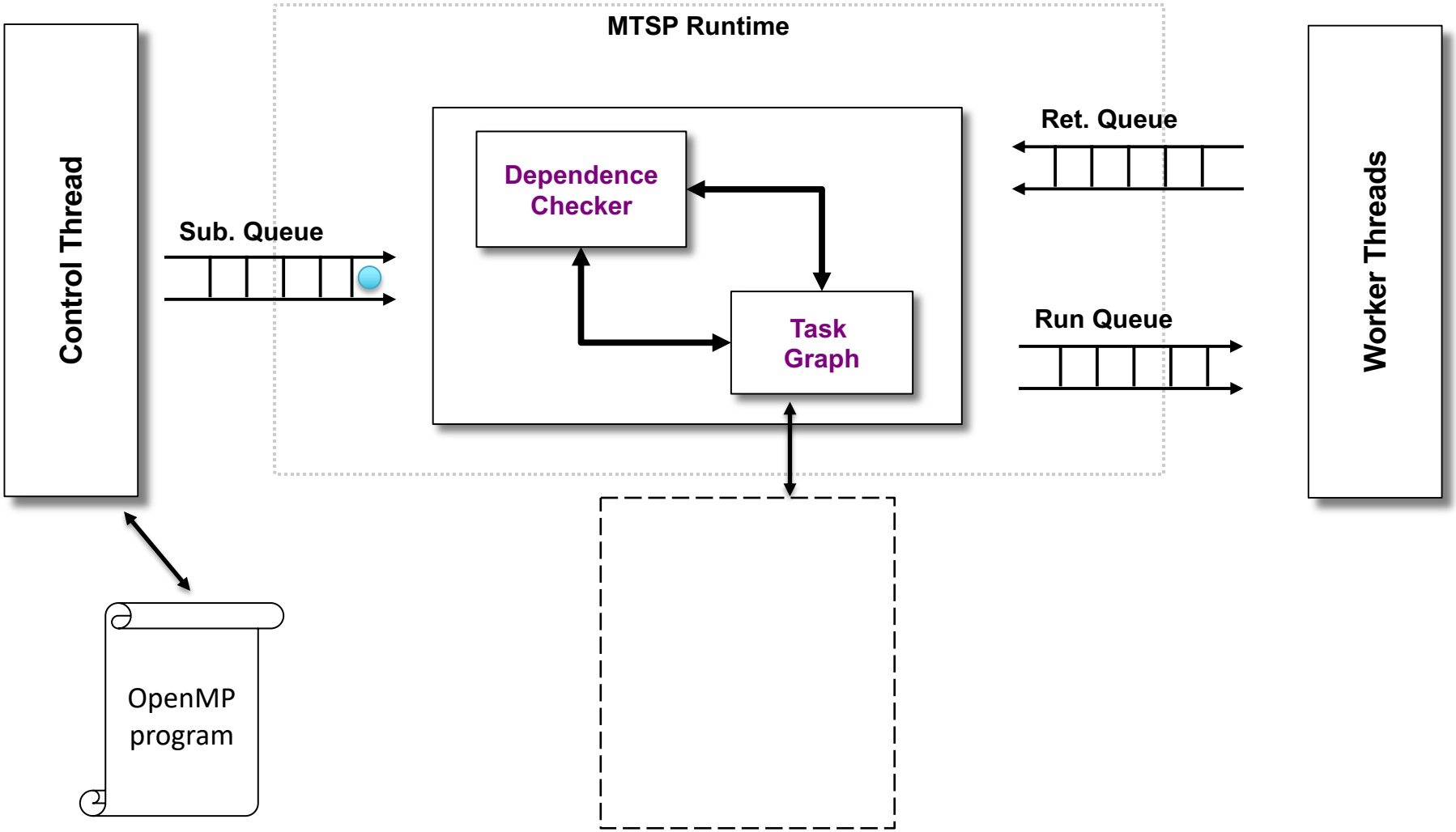
Basic elements of a task scheduling system



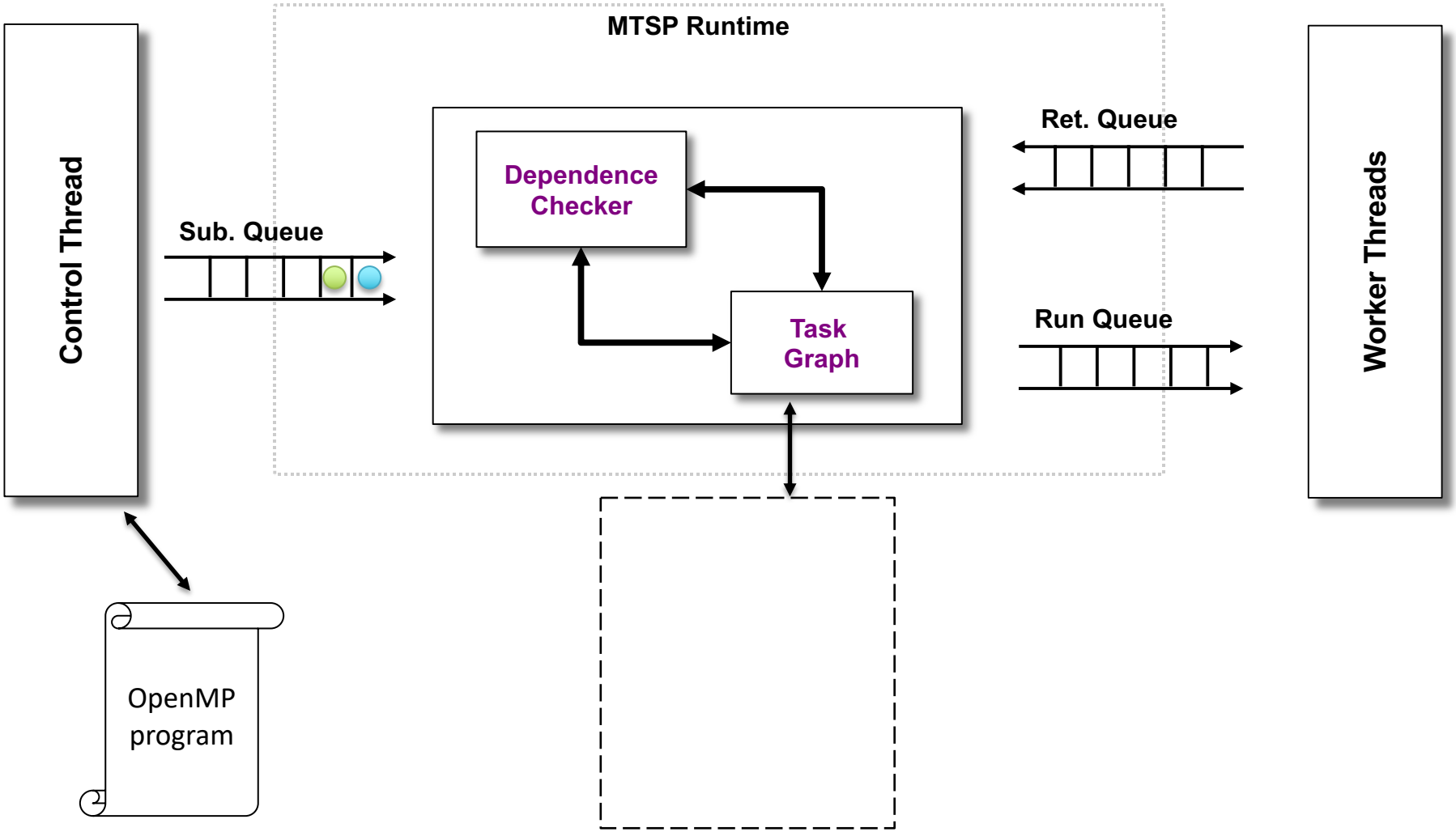
Basic elements of a task scheduling system



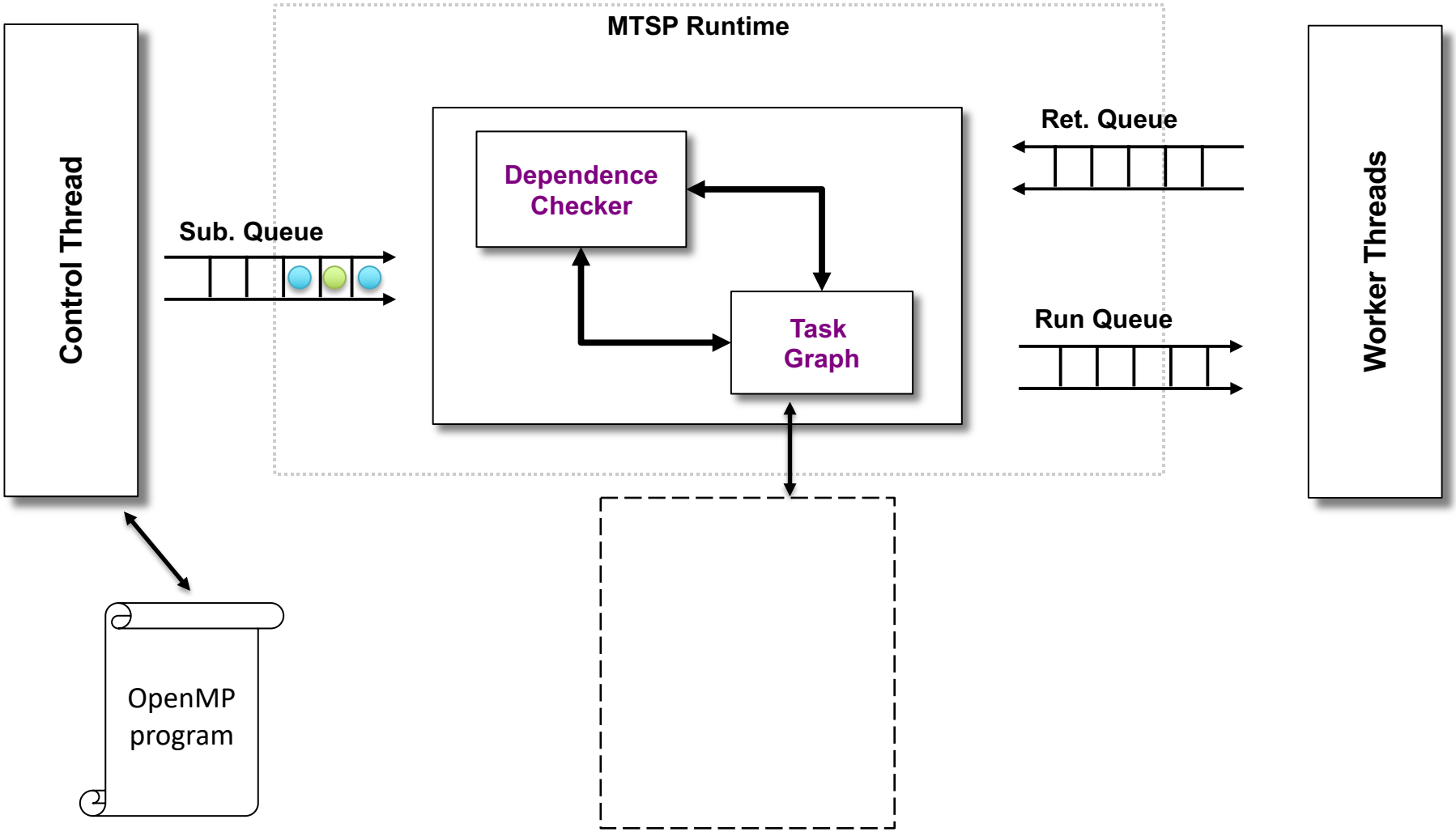
Basic elements of a task scheduling system



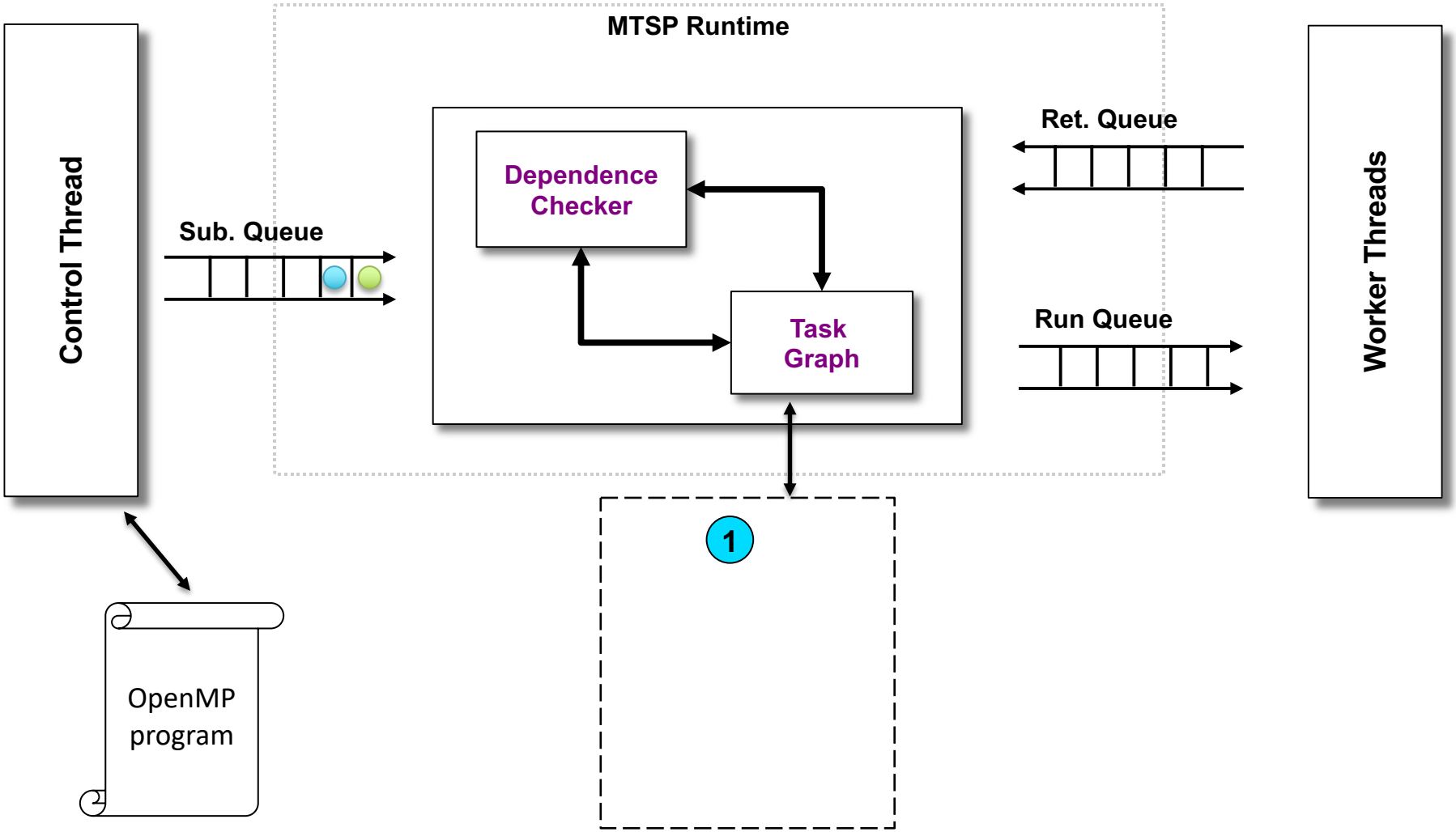
Basic elements of a task scheduling system



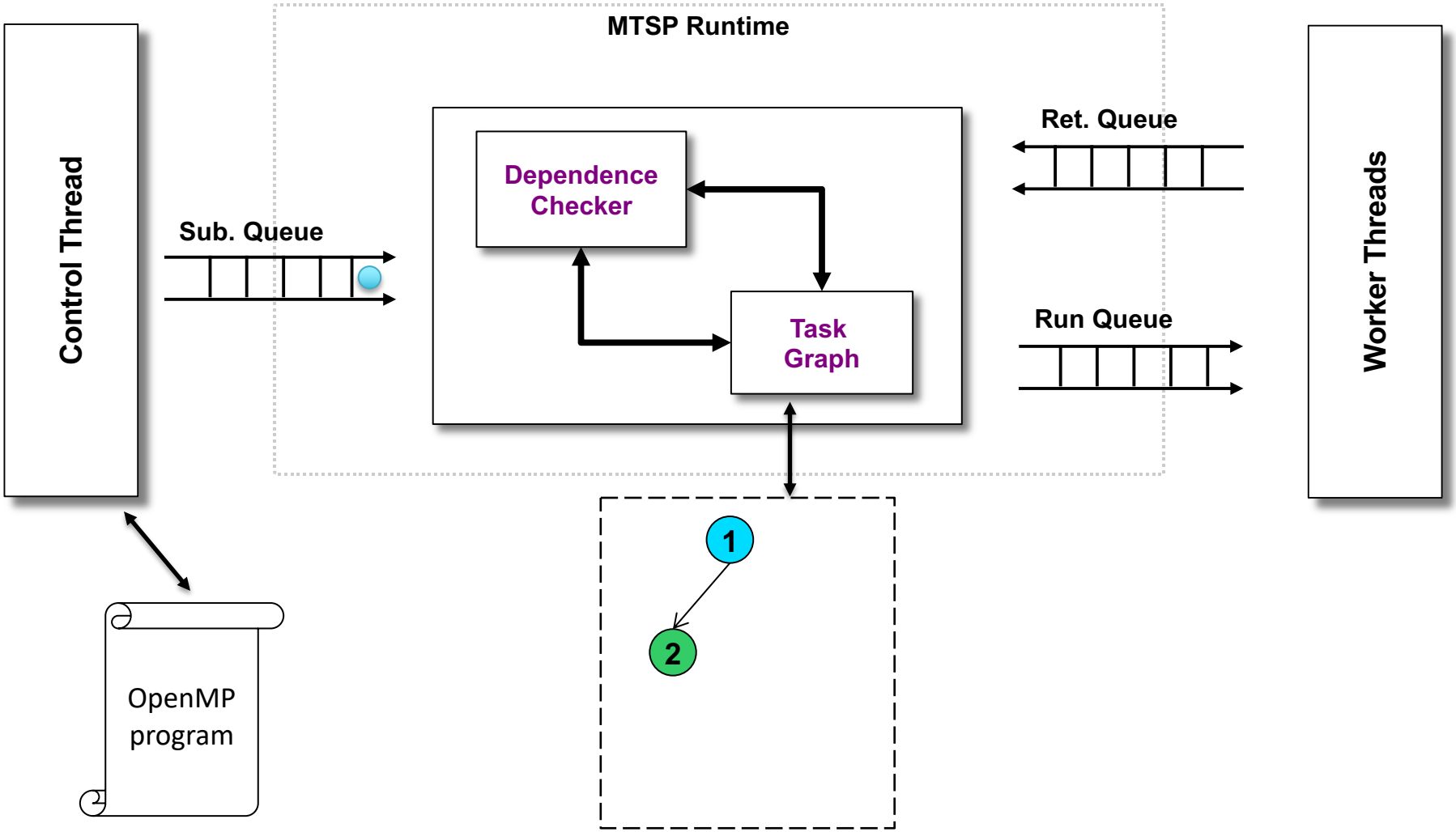
Basic elements of a task scheduling system



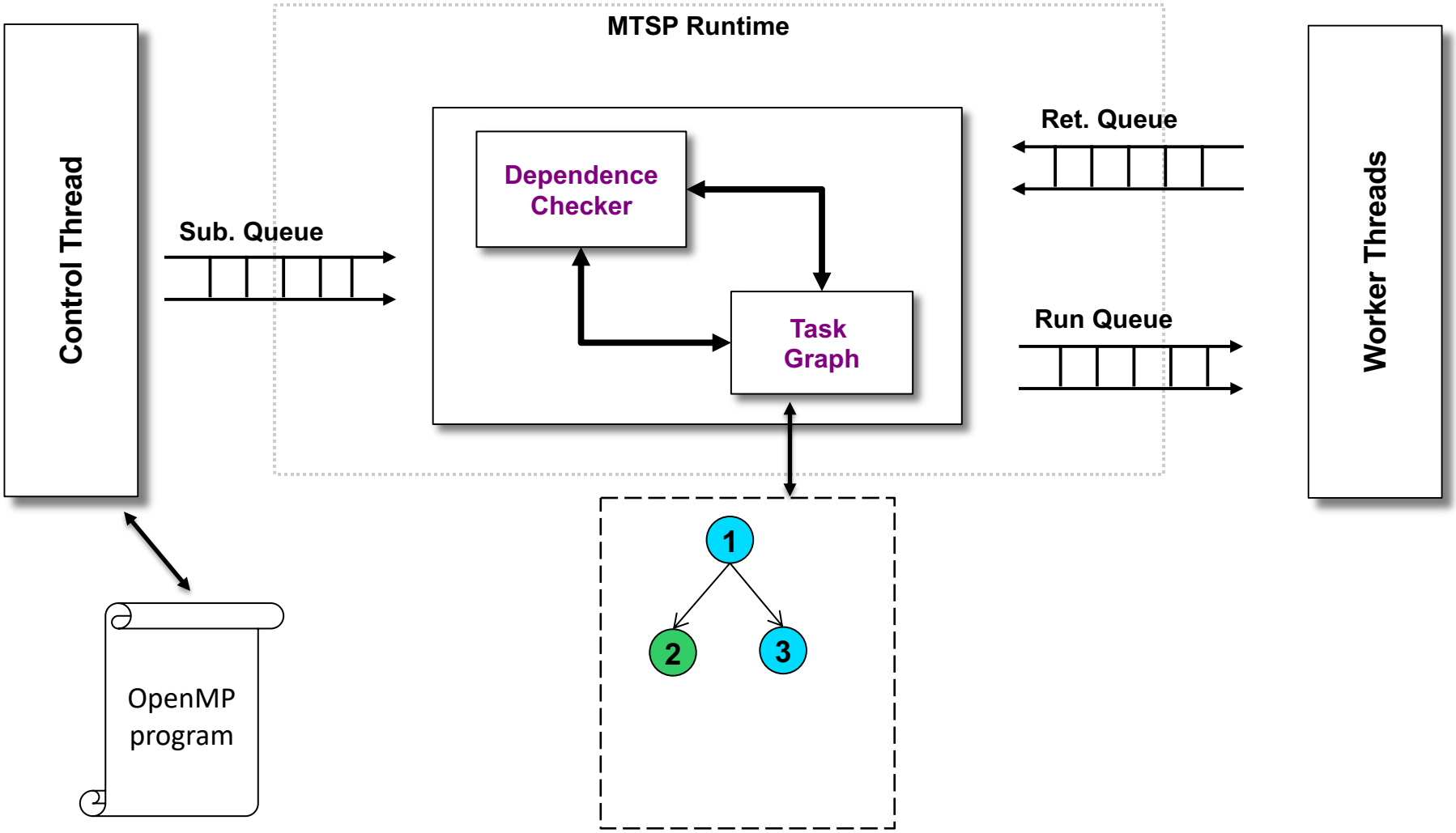
Basic elements of a task scheduling system



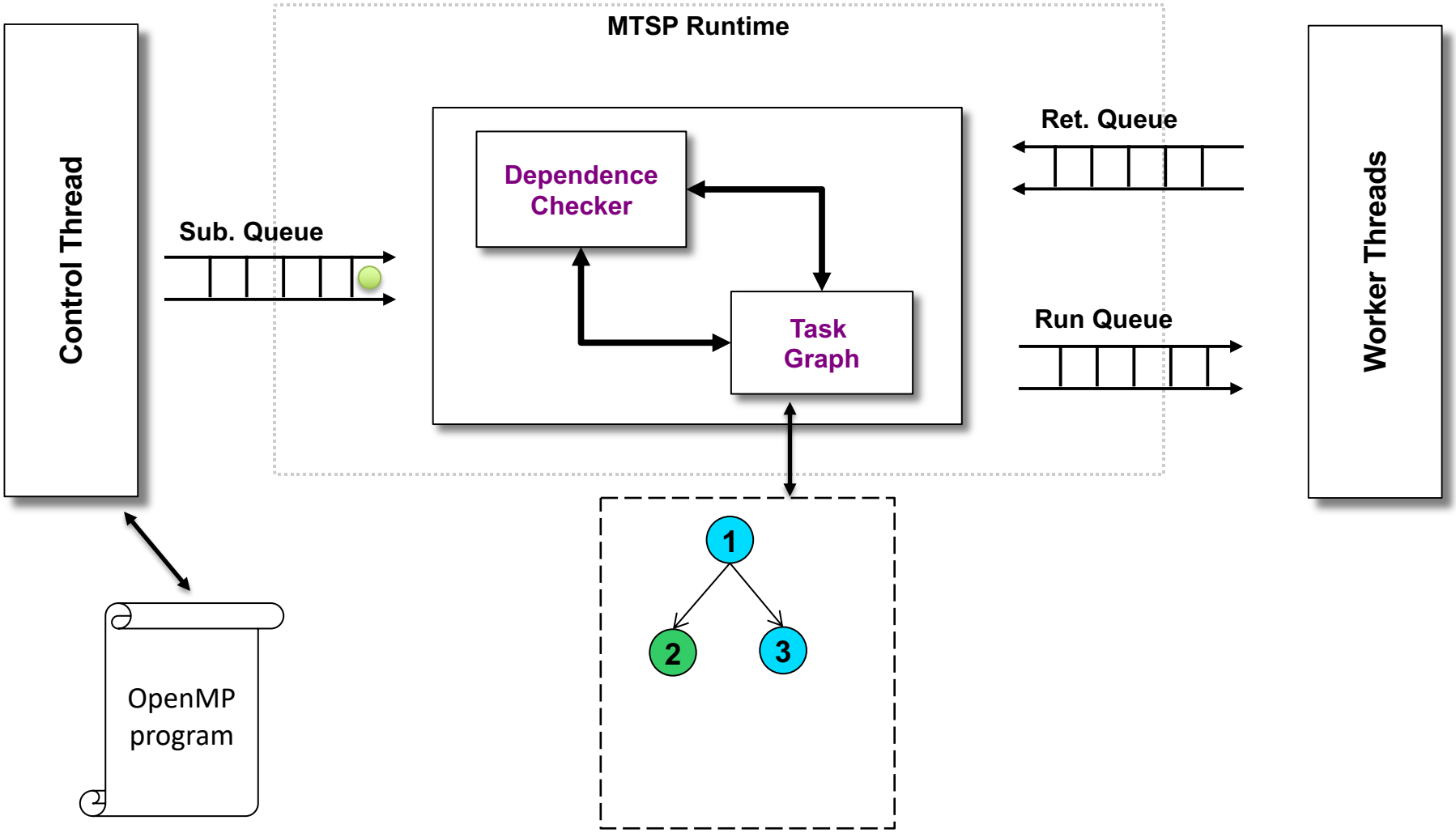
Basic elements of a task scheduling system



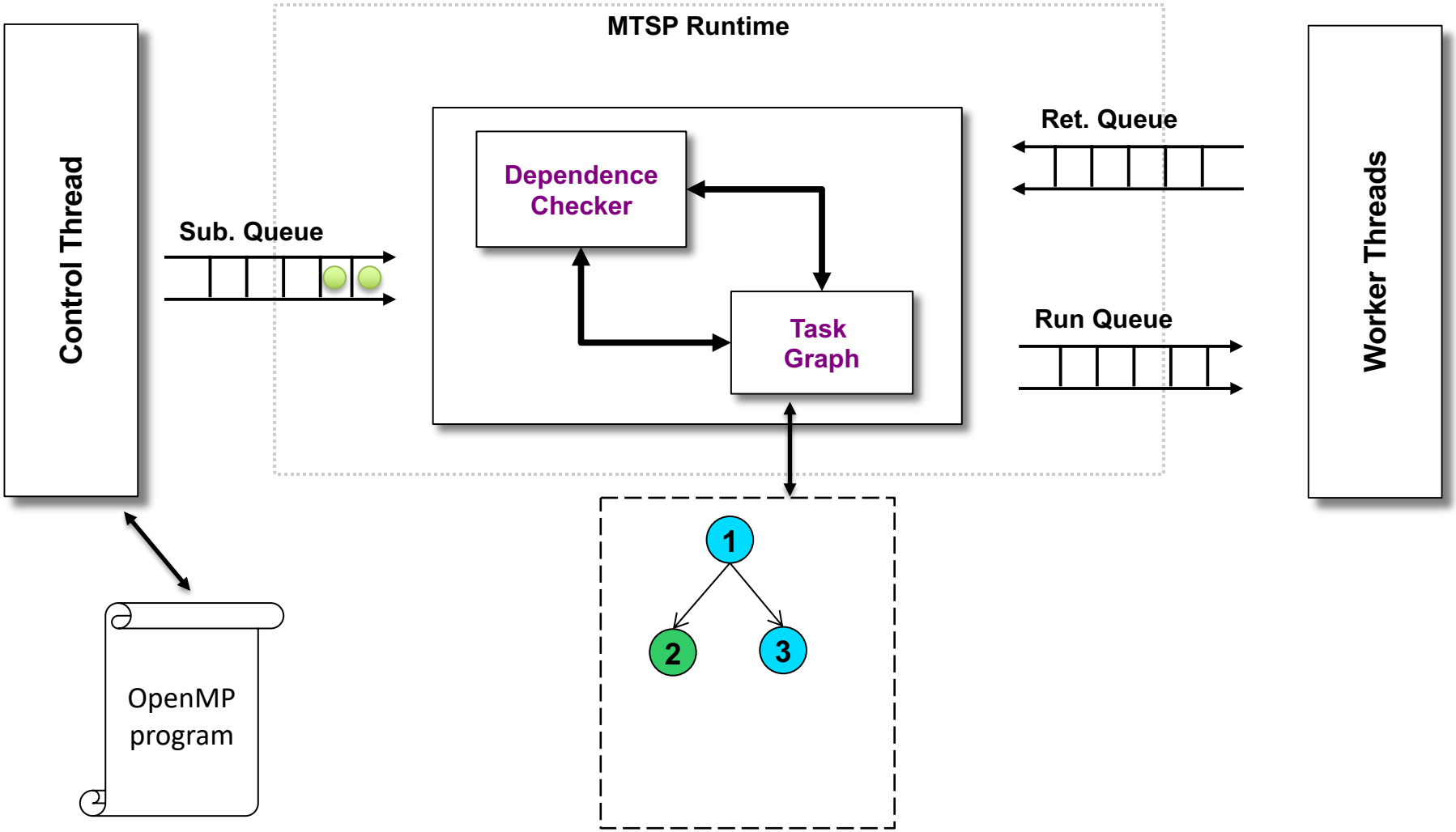
Basic elements of a task scheduling system



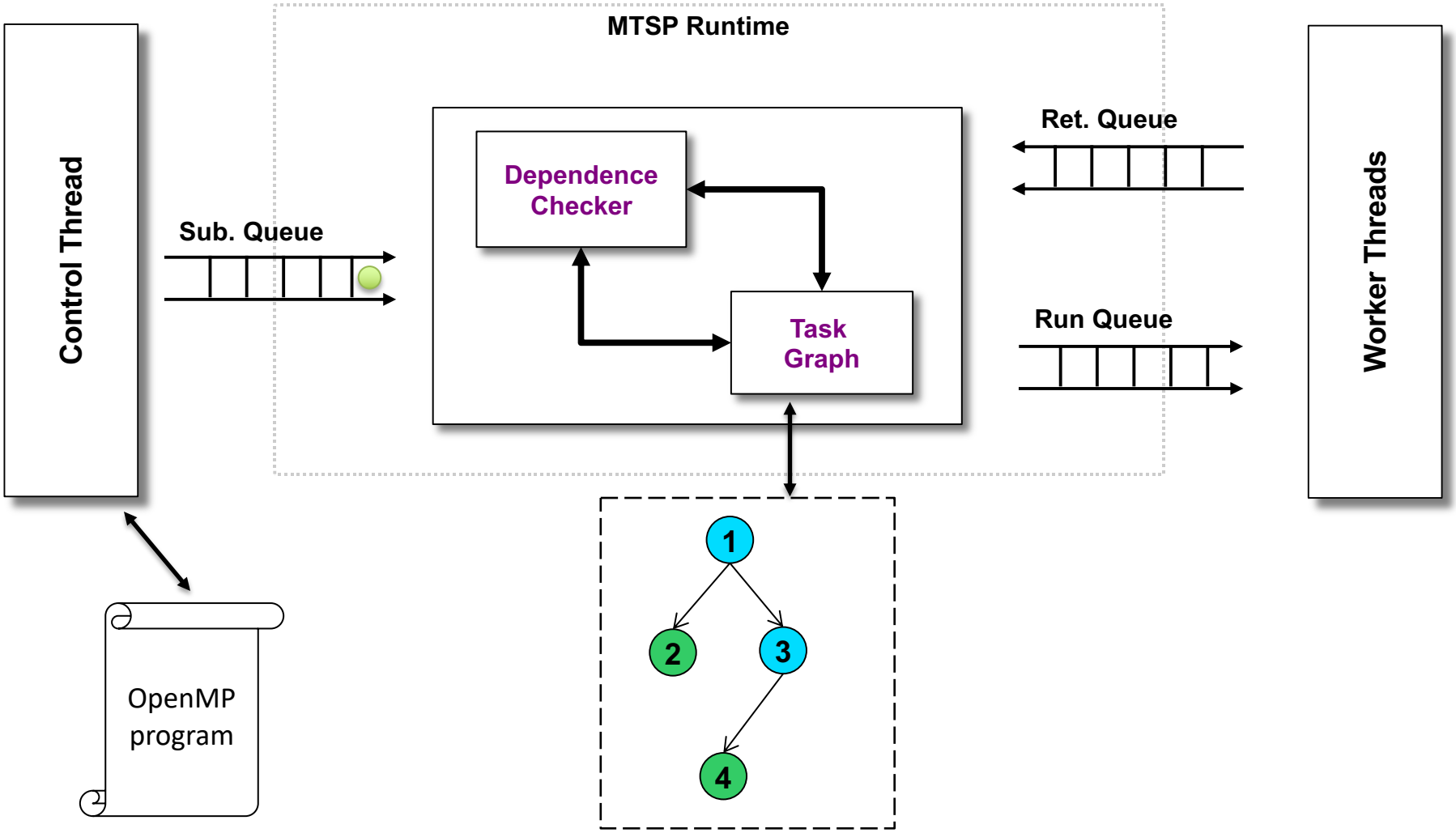
Basic elements of a task scheduling system



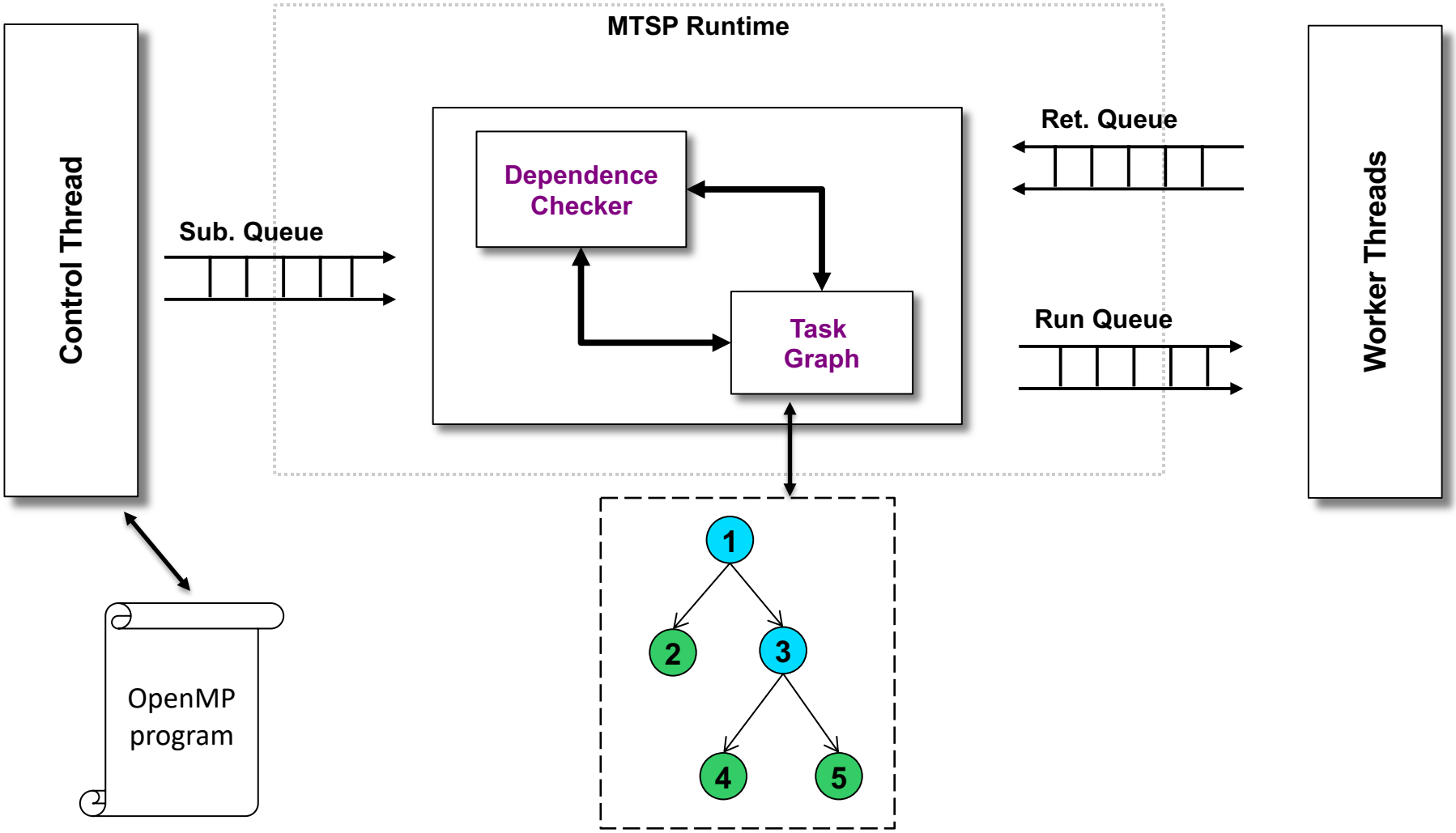
Basic elements of a task scheduling system



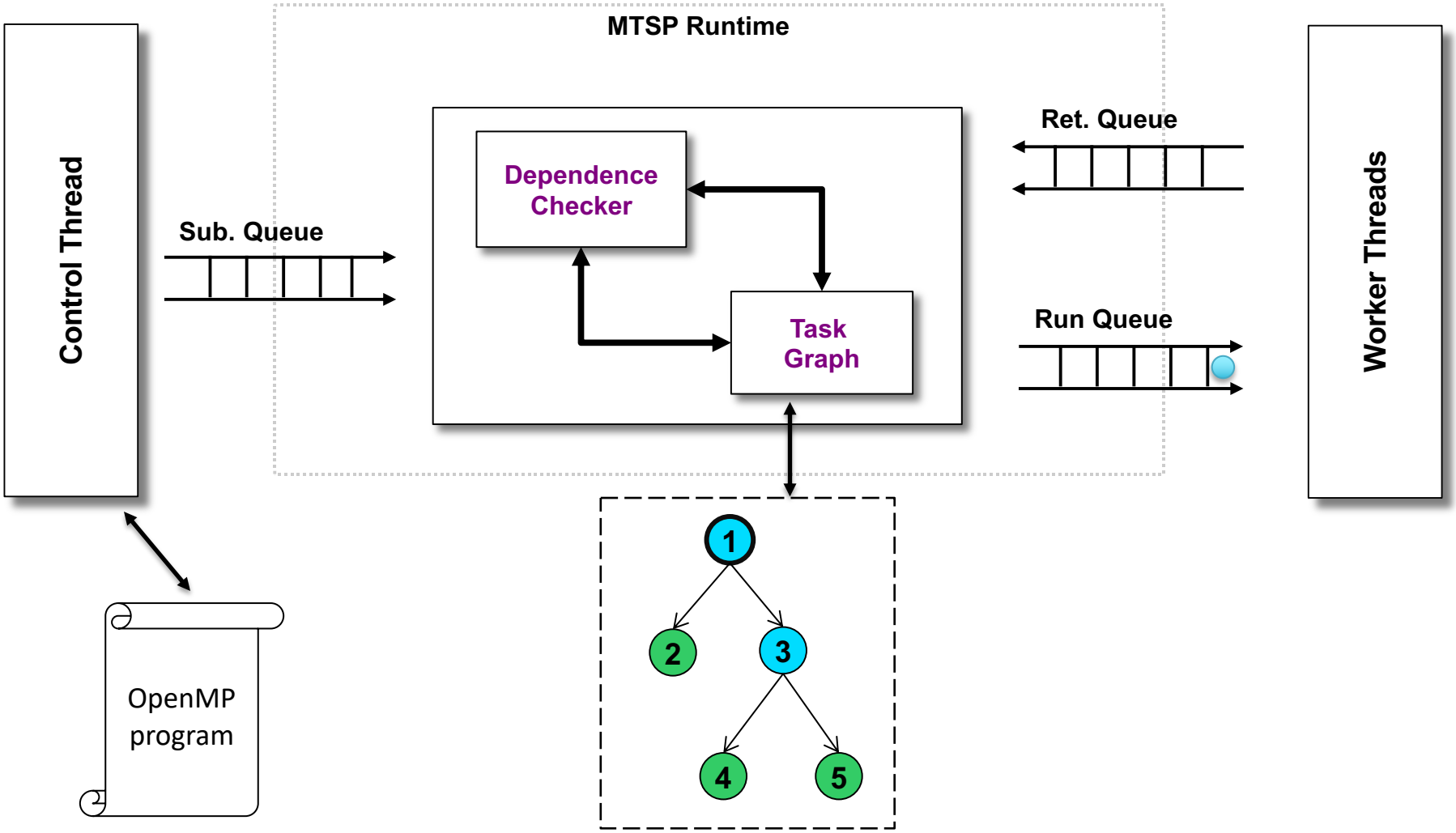
Basic elements of a task scheduling system



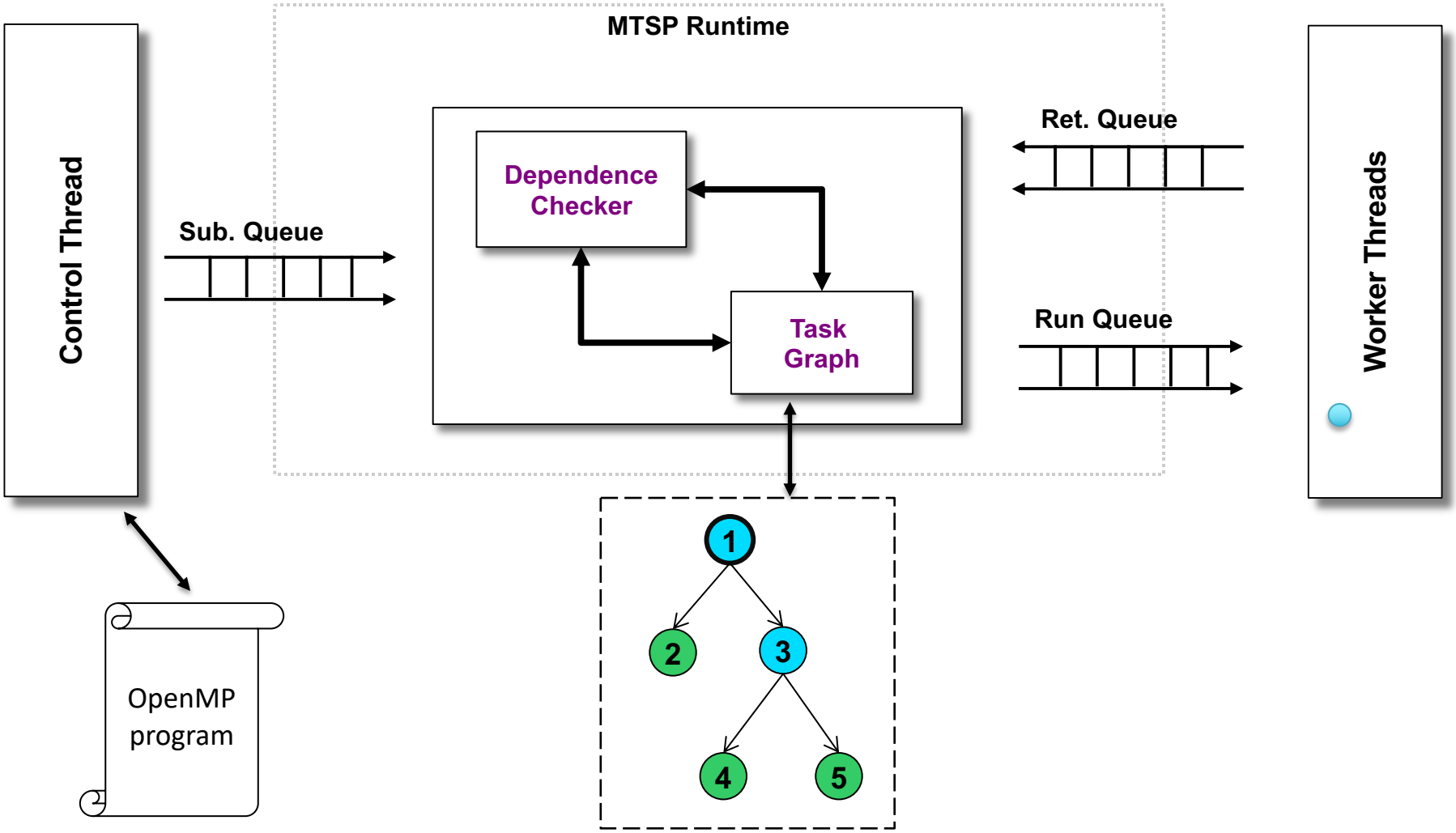
Basic elements of a task scheduling system



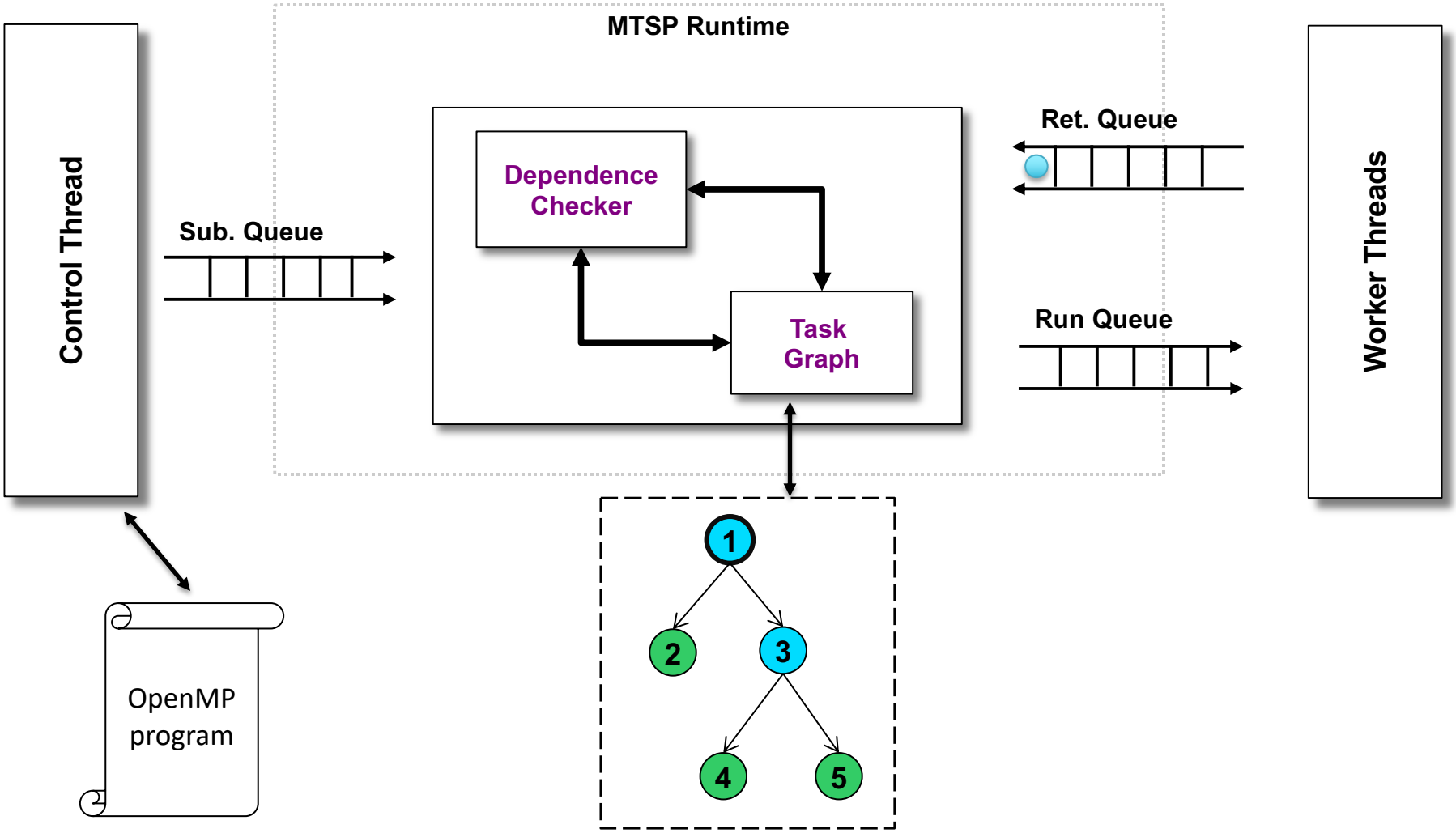
Basic elements of a task scheduling system



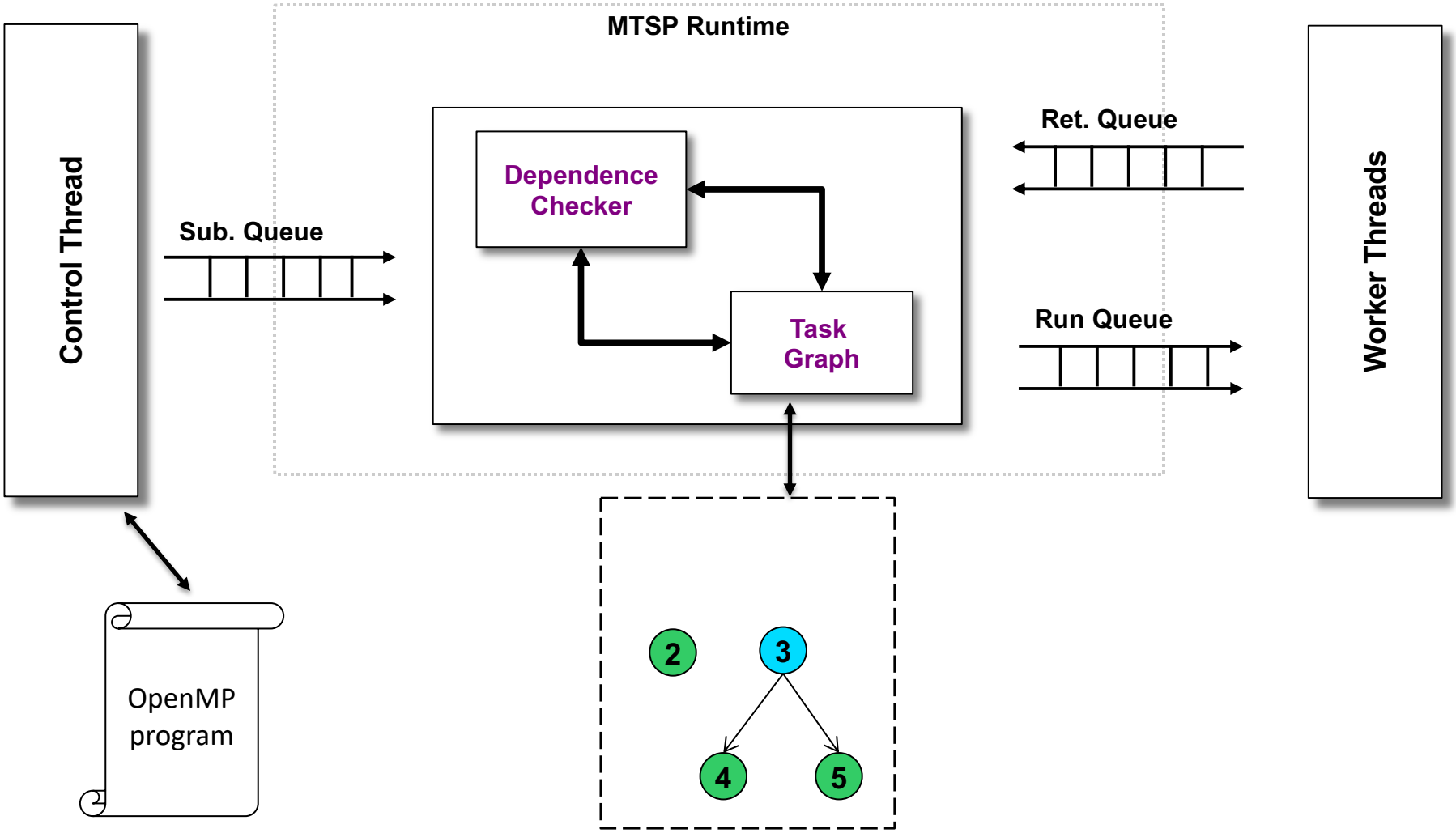
Basic elements of a task scheduling system



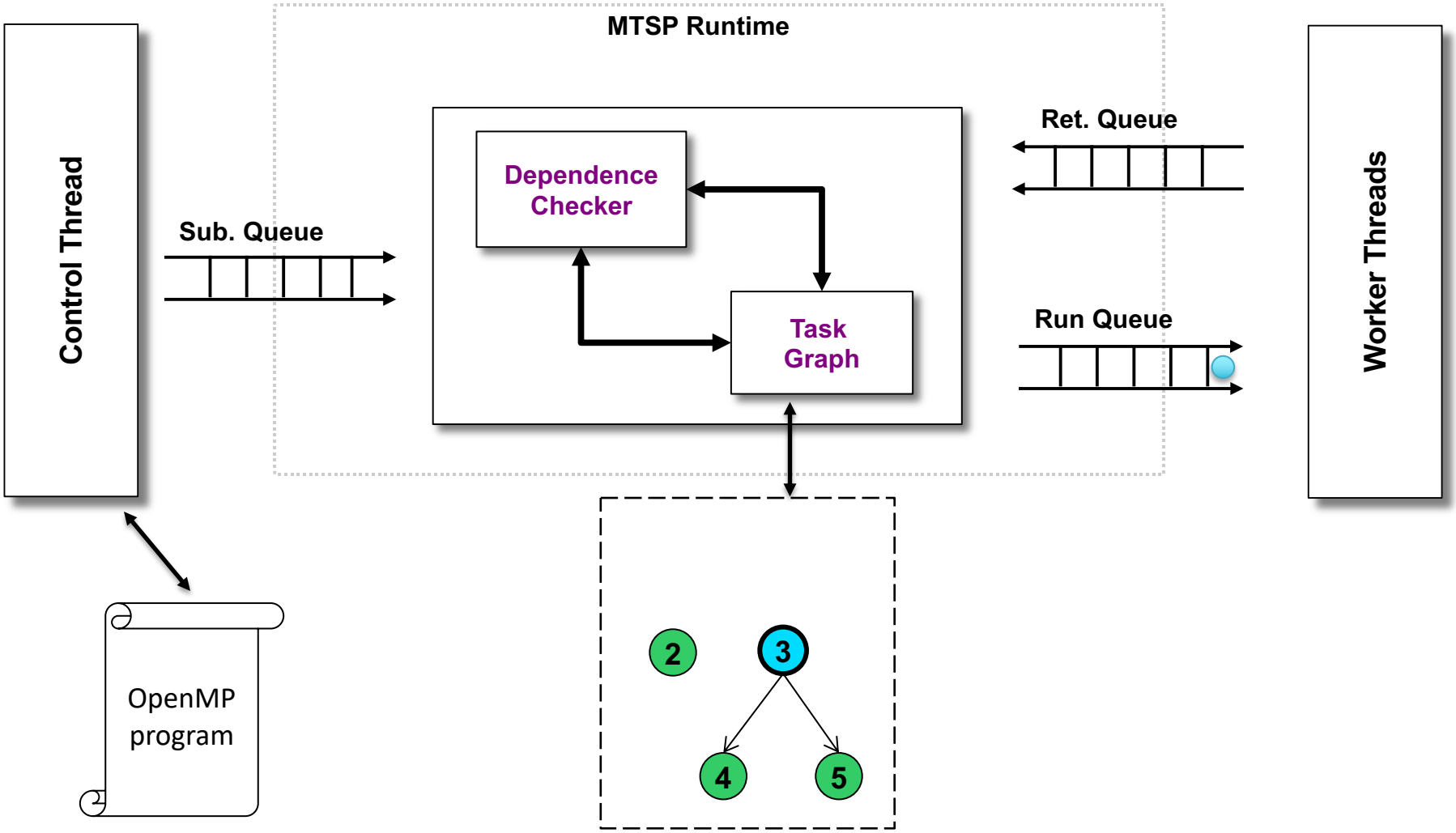
Basic elements of a task scheduling system



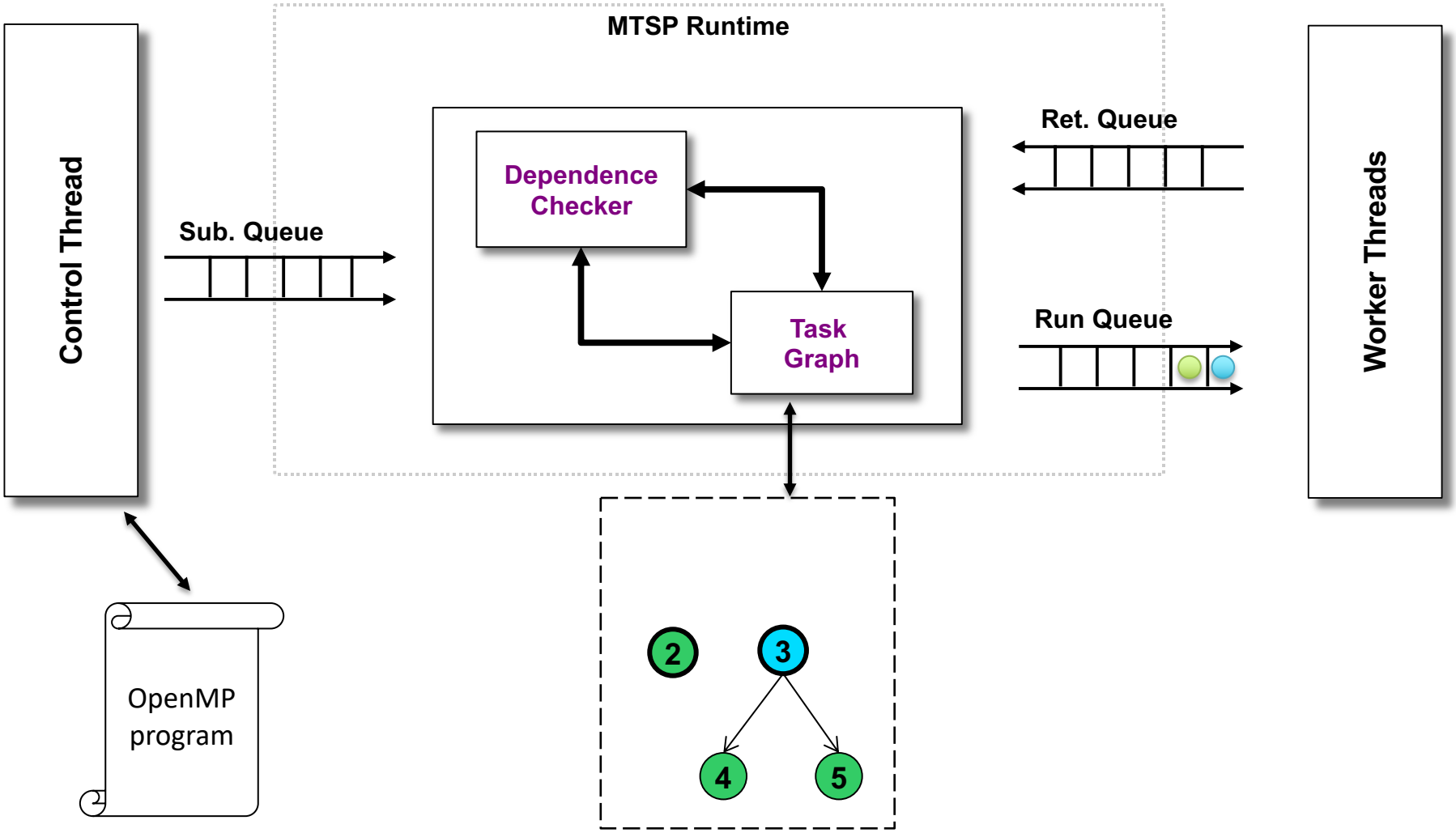
Basic elements of a task scheduling system



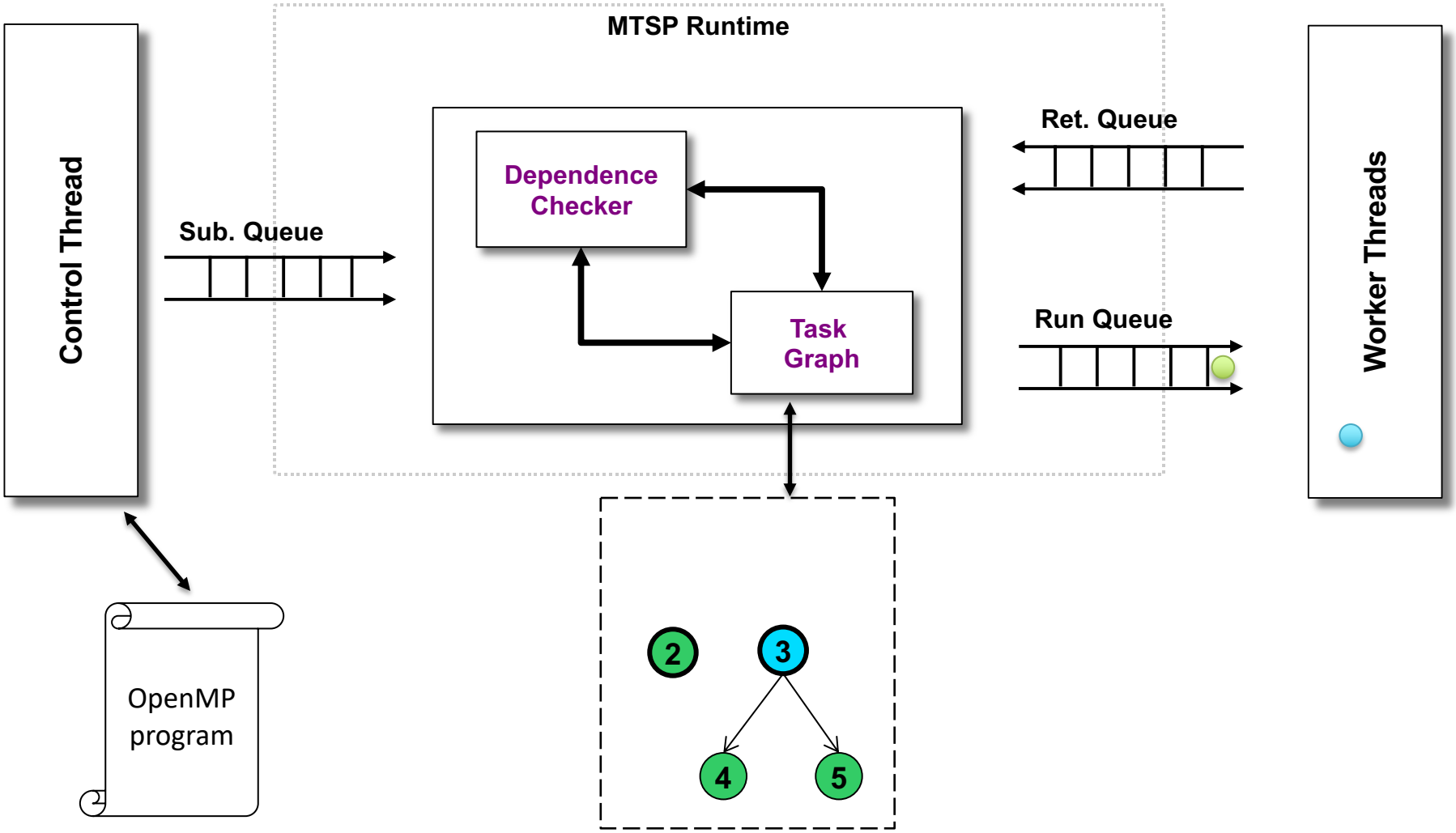
Basic elements of a task scheduling system



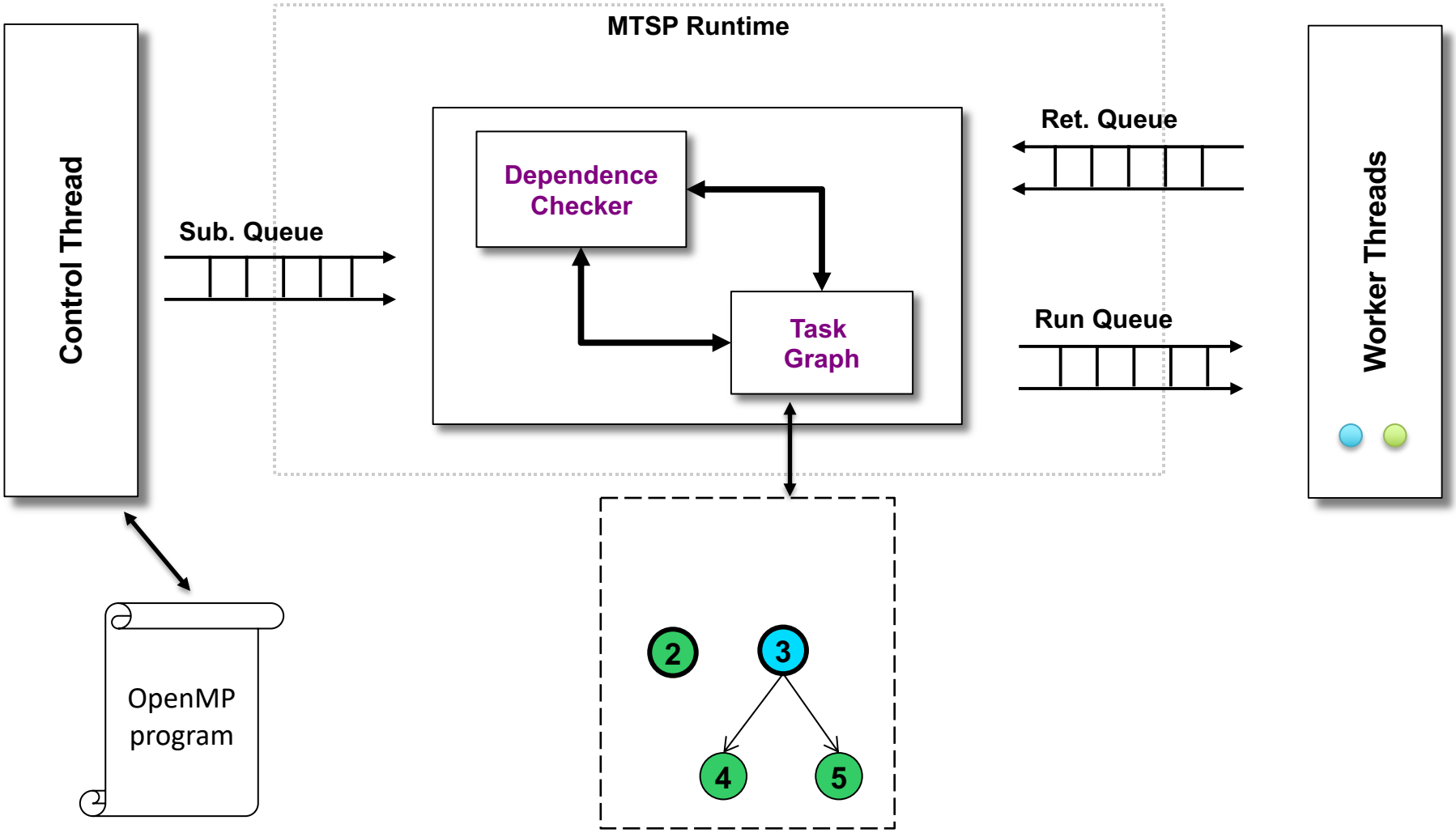
Basic elements of a task scheduling system



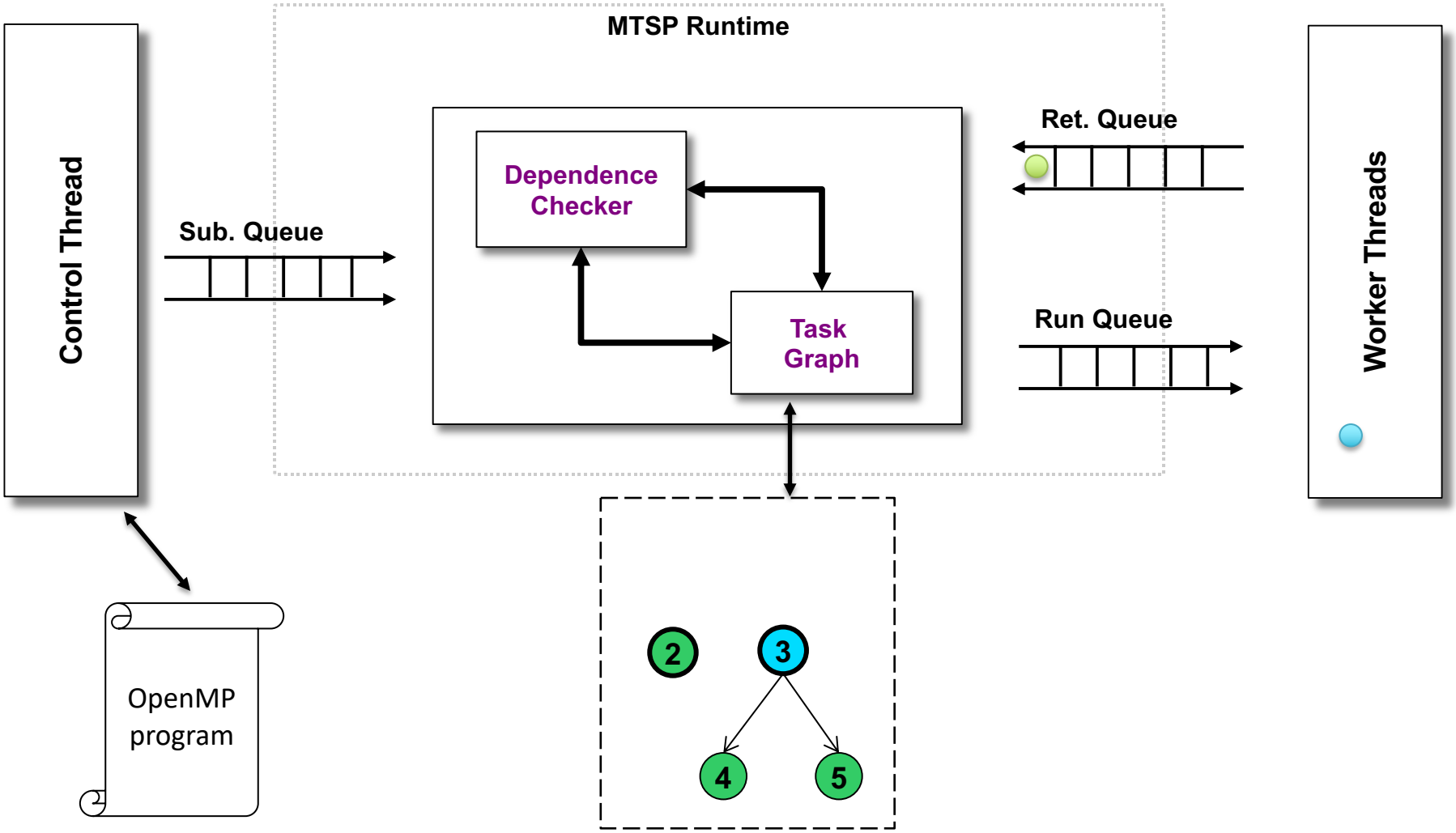
Basic elements of a task scheduling system



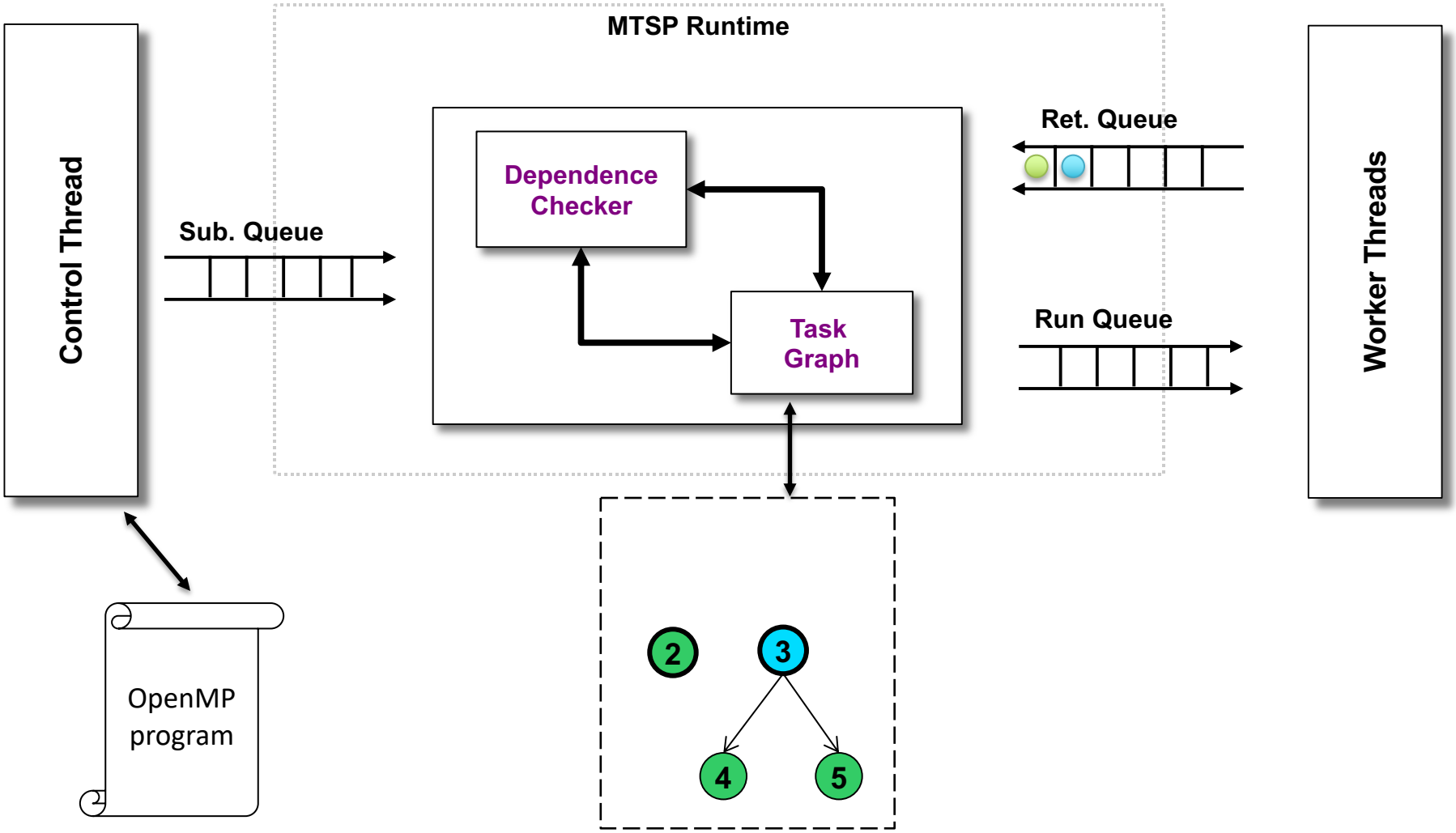
Basic elements of a task scheduling system



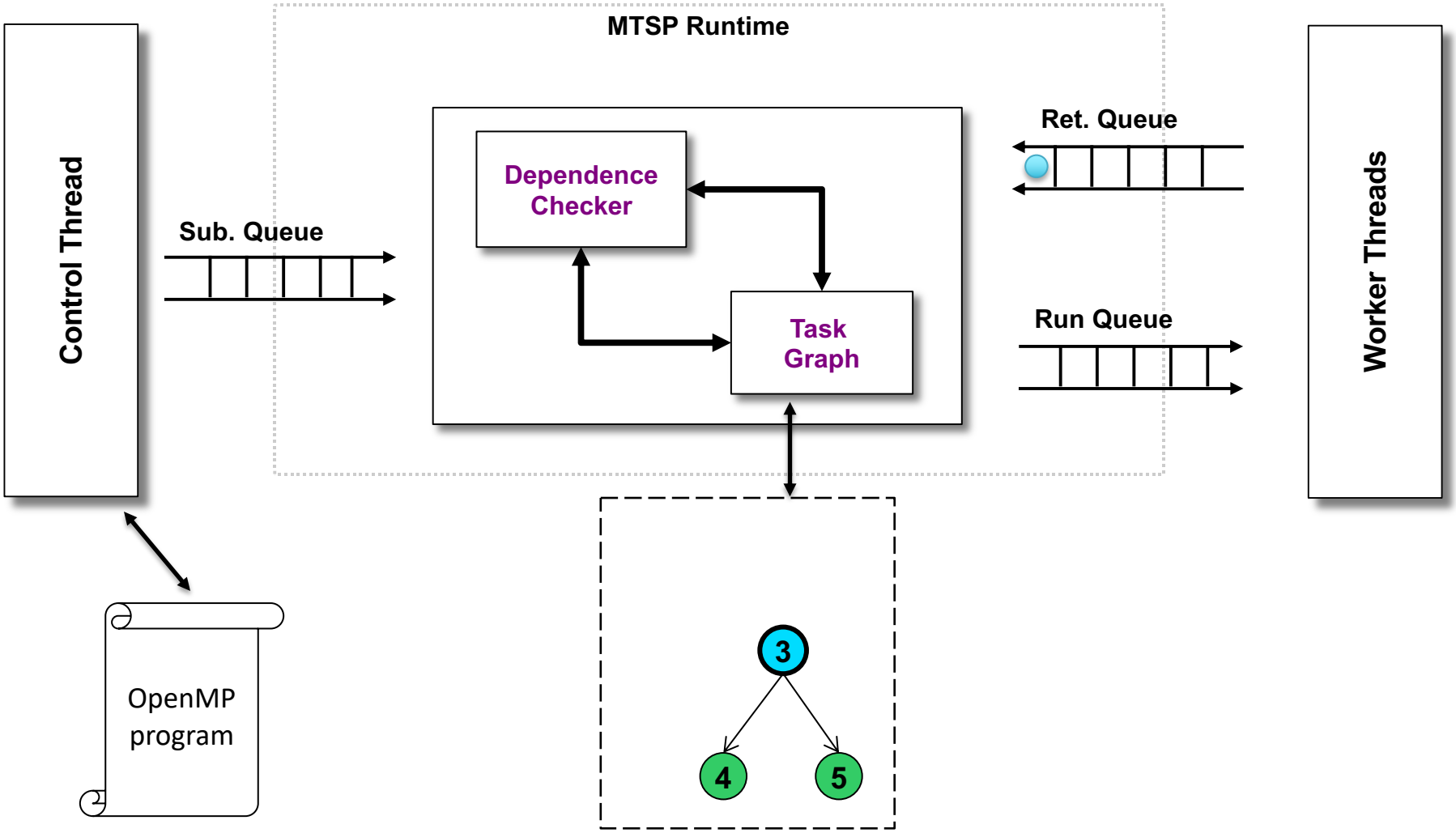
Basic elements of a task scheduling system



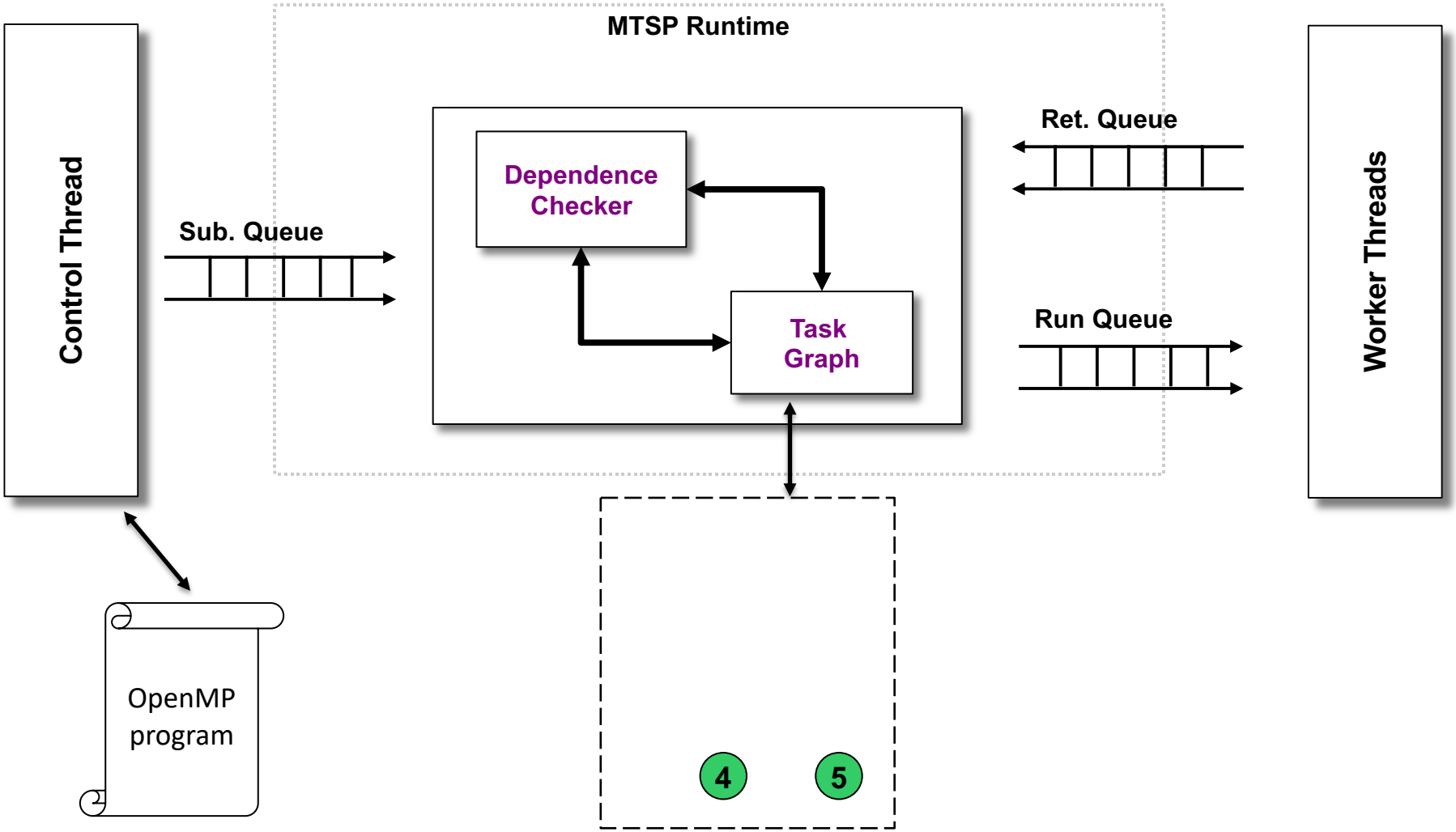
Basic elements of a task scheduling system



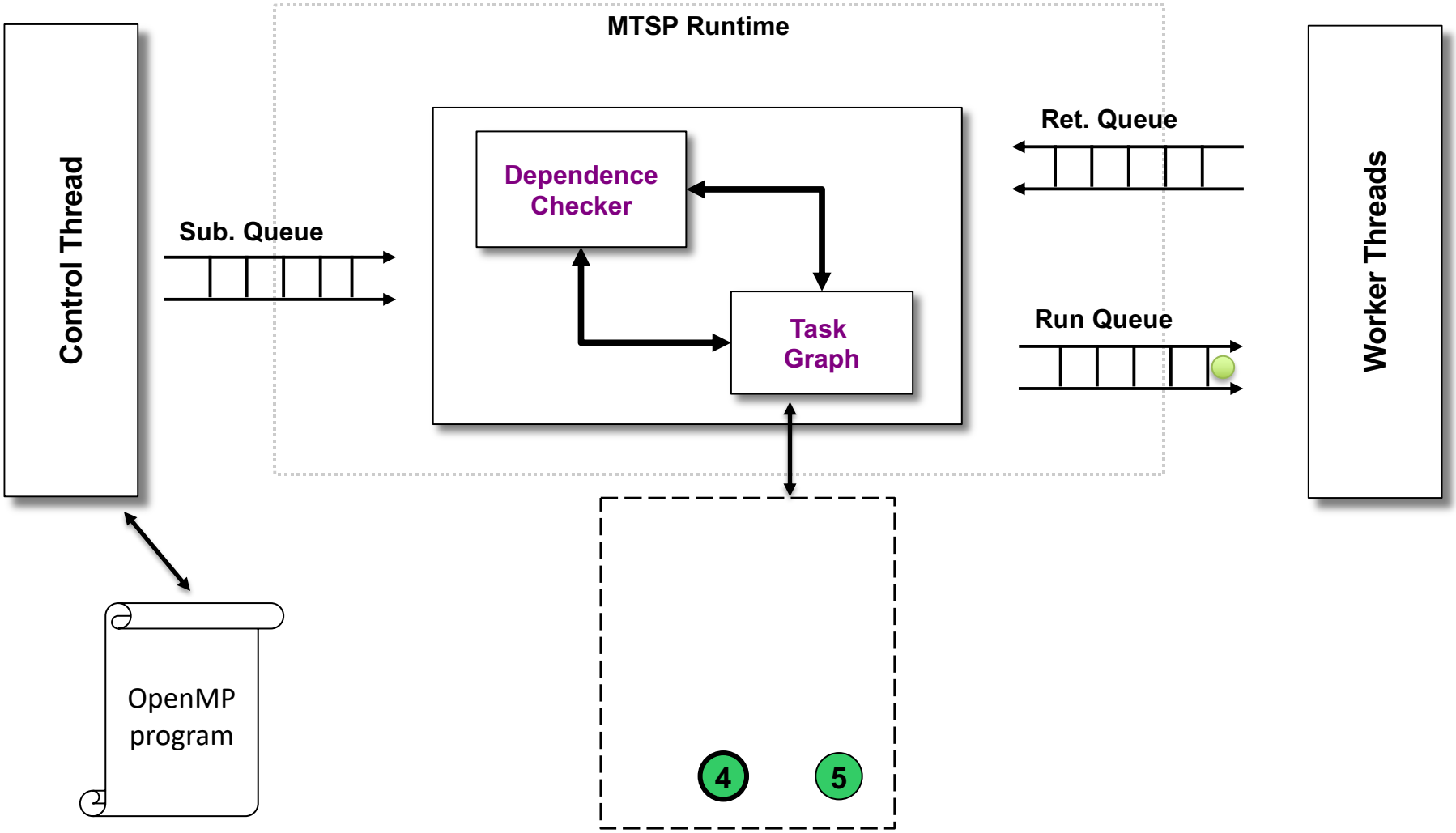
Basic elements of a task scheduling system



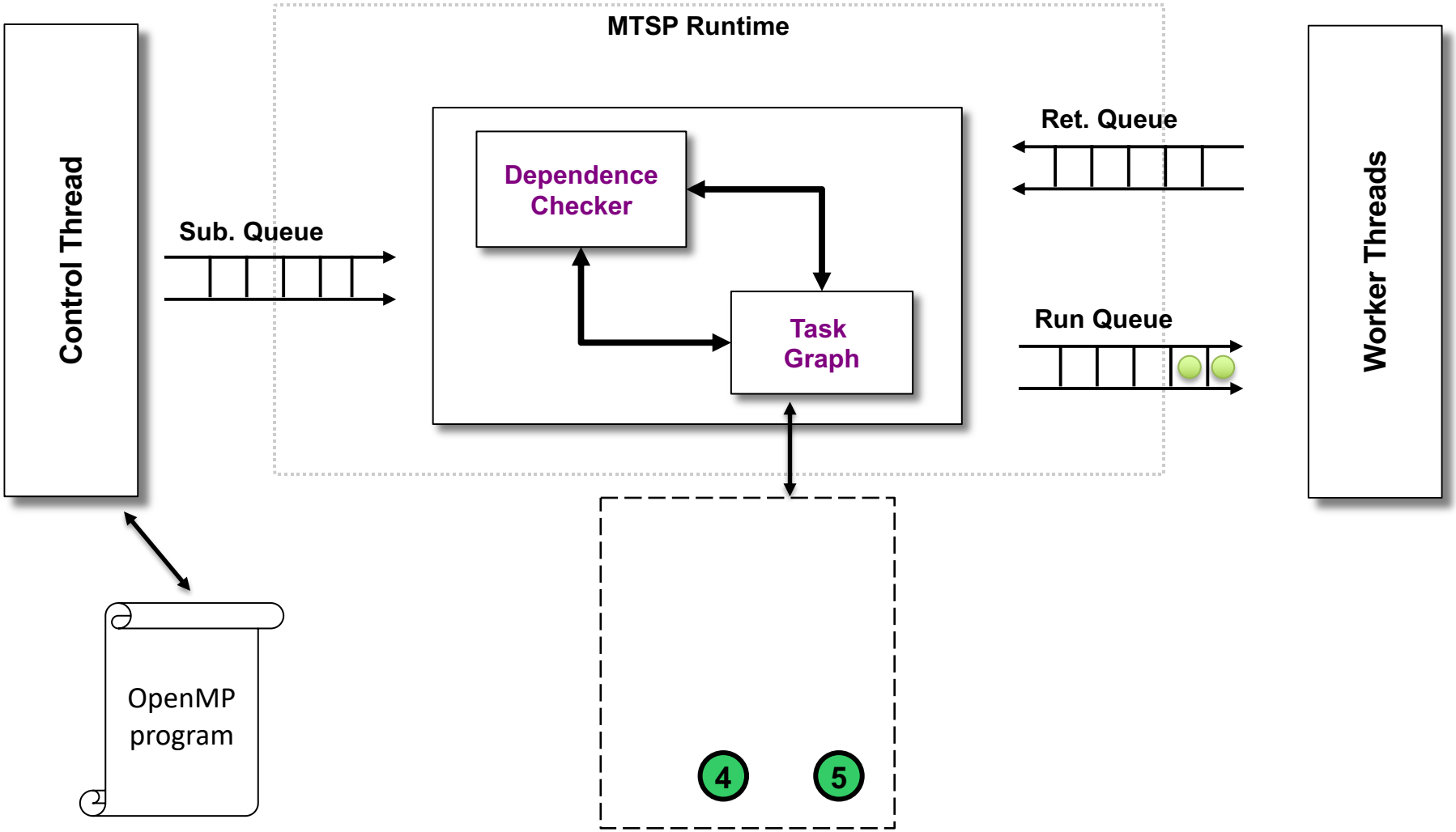
Basic elements of a task scheduling system



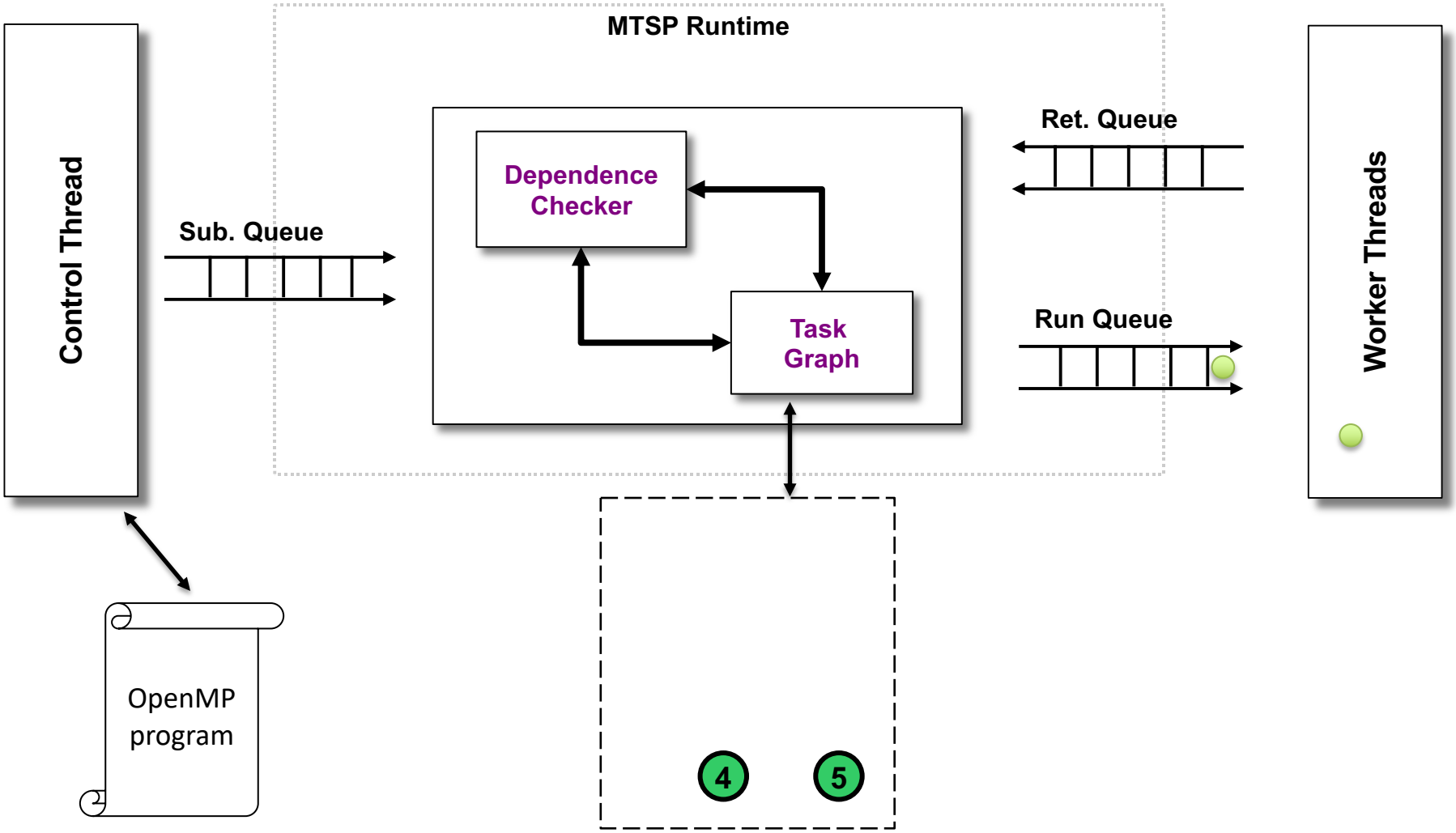
Basic elements of a task scheduling system



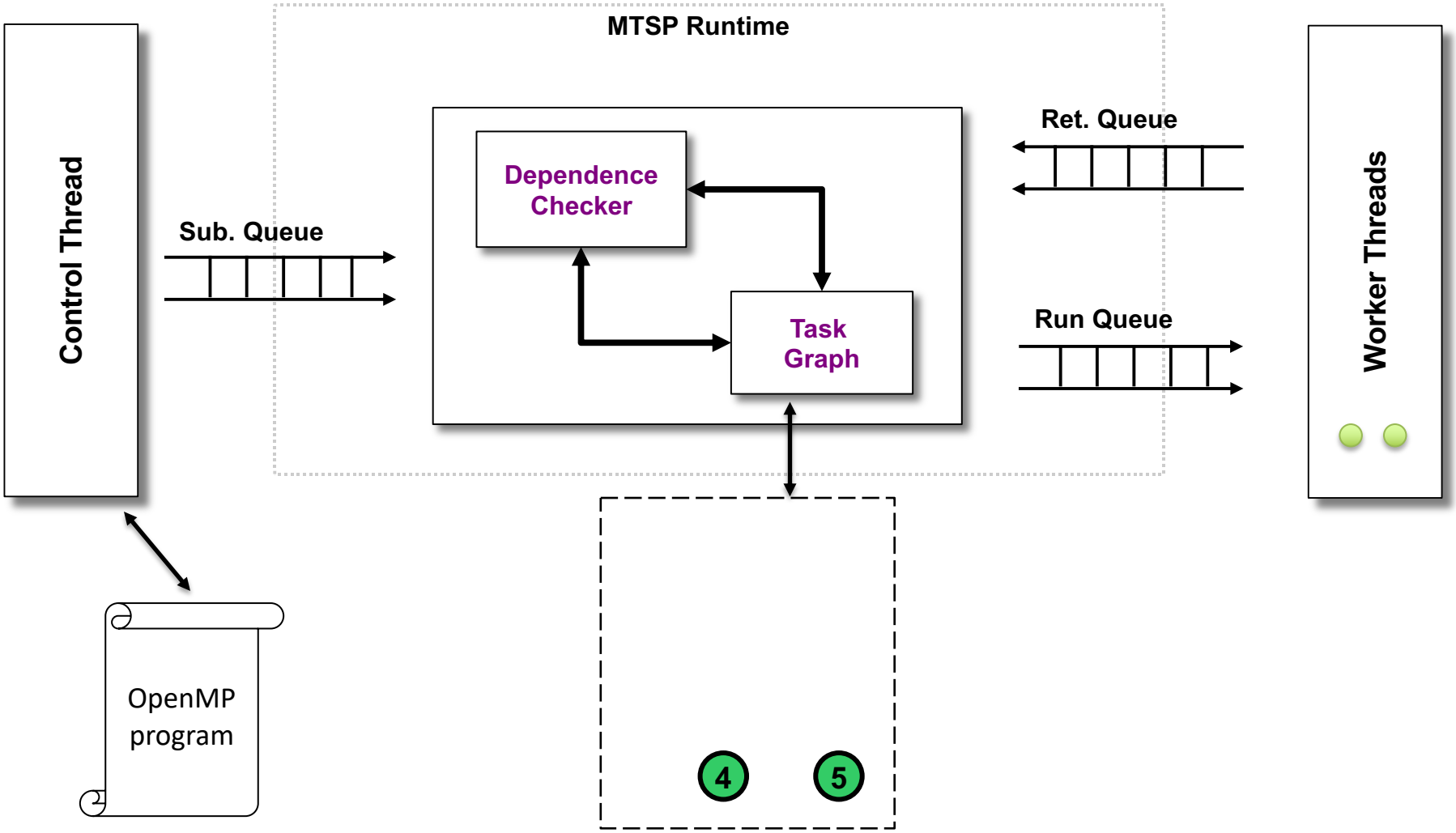
Basic elements of a task scheduling system



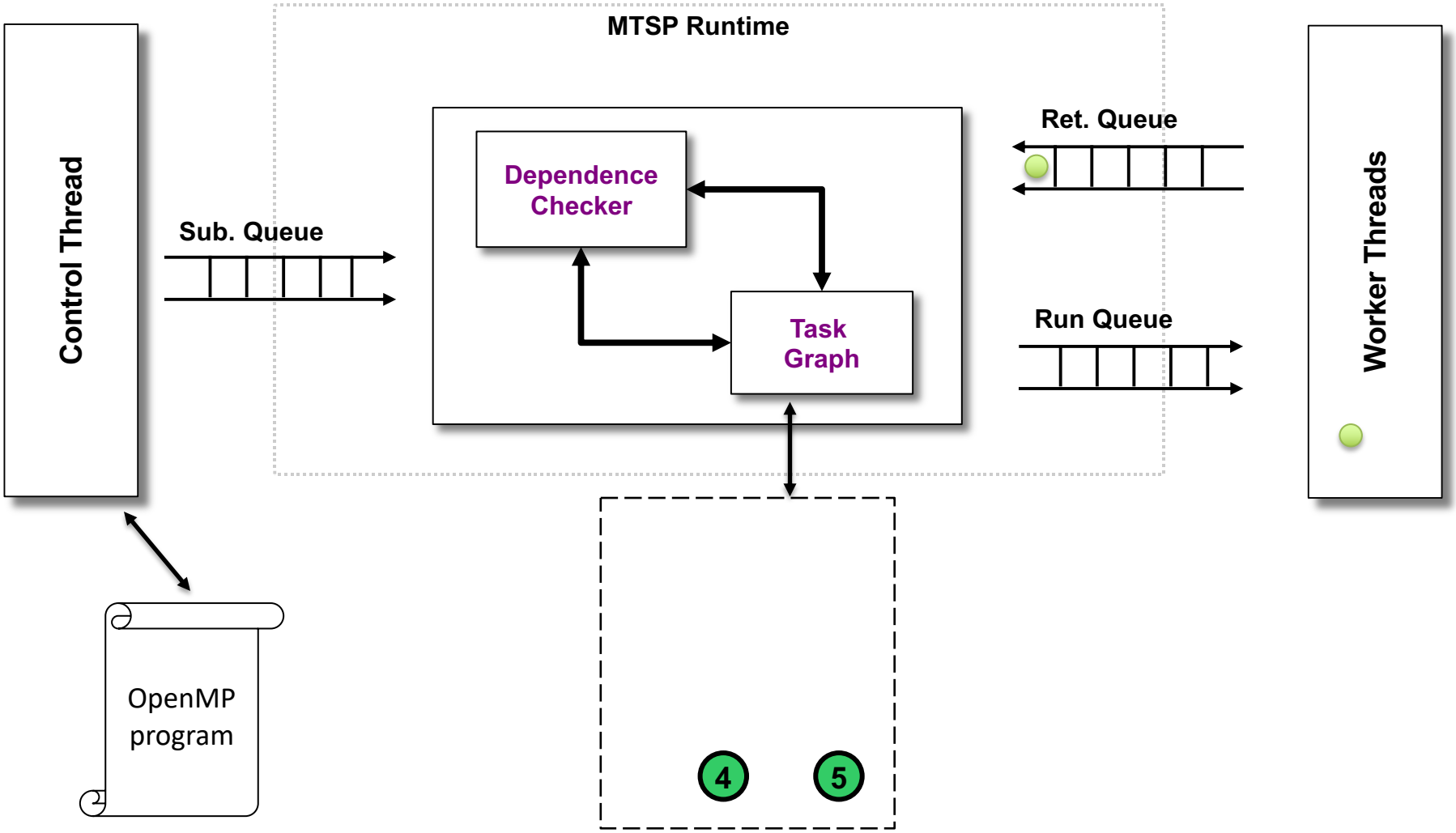
Basic elements of a task scheduling system



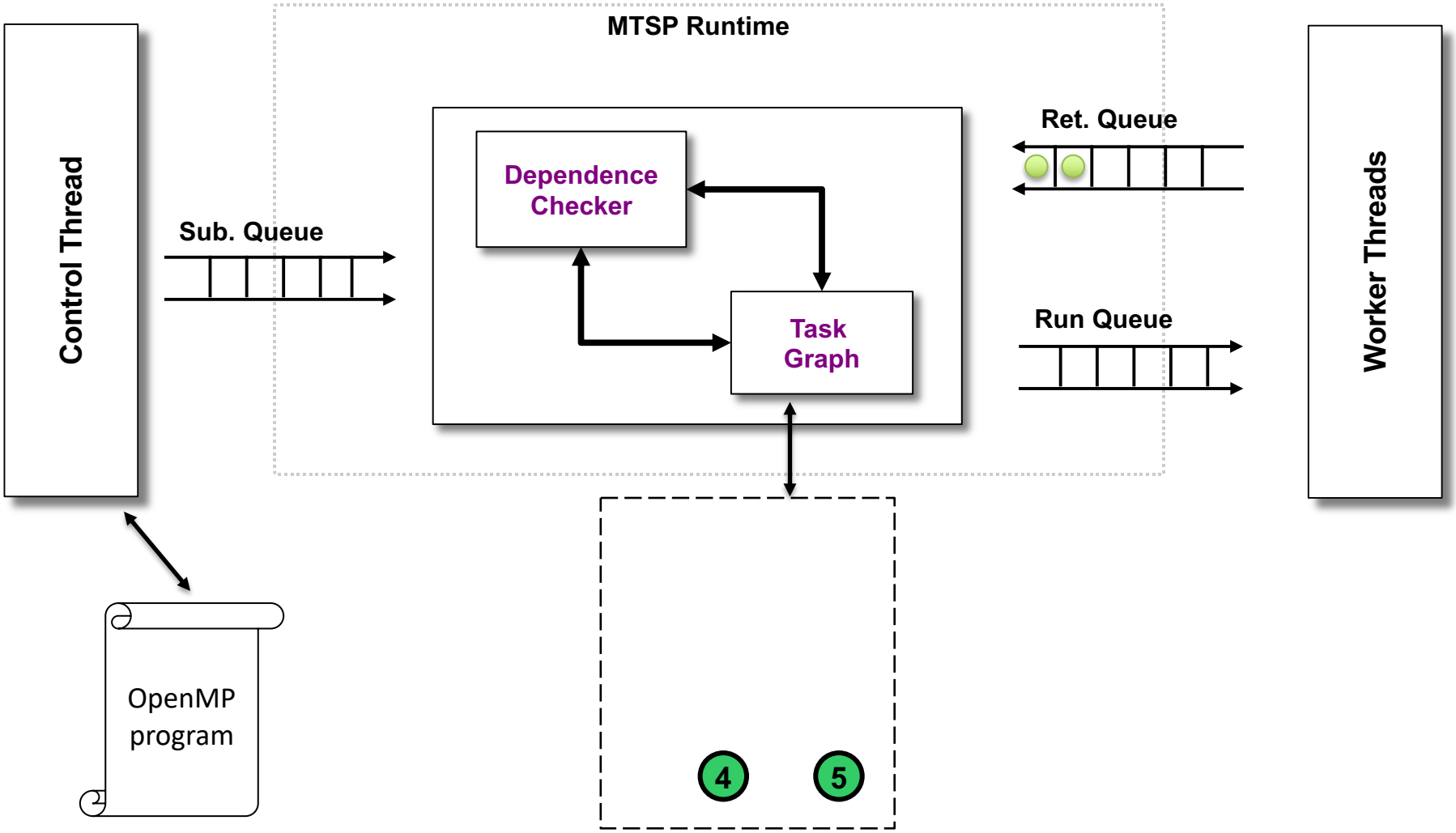
Basic elements of a task scheduling system



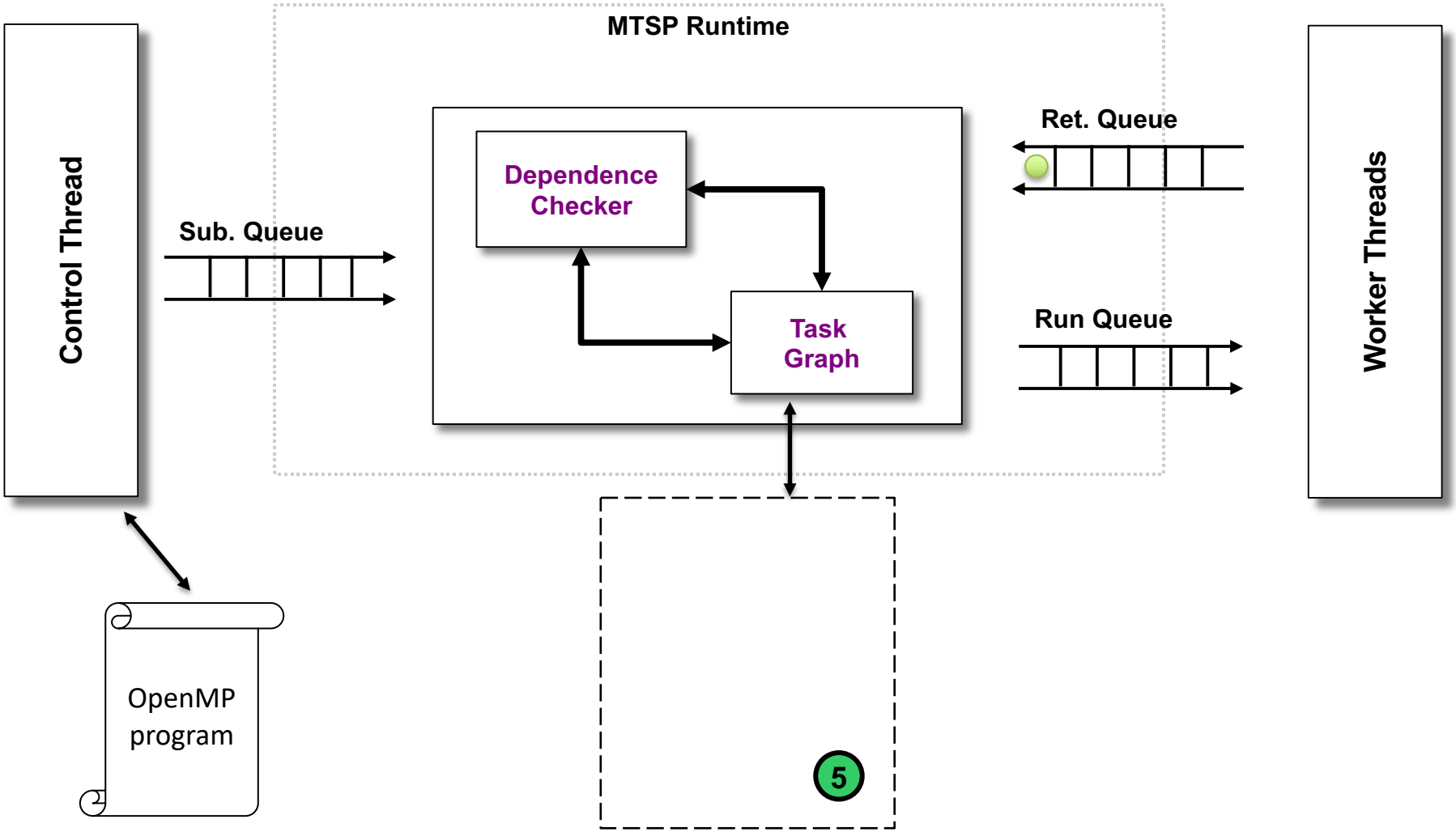
Basic elements of a task scheduling system



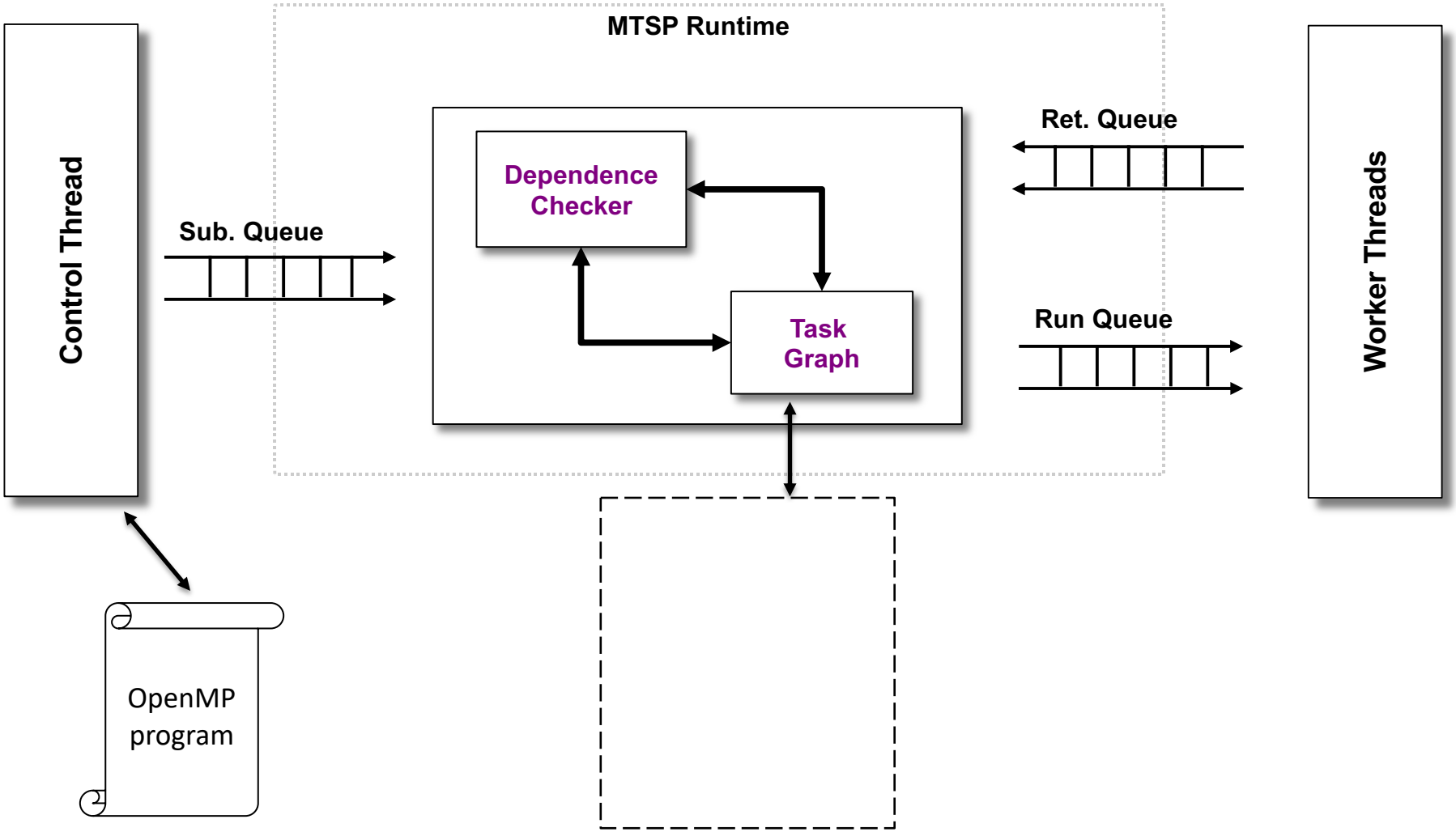
Basic elements of a task scheduling system



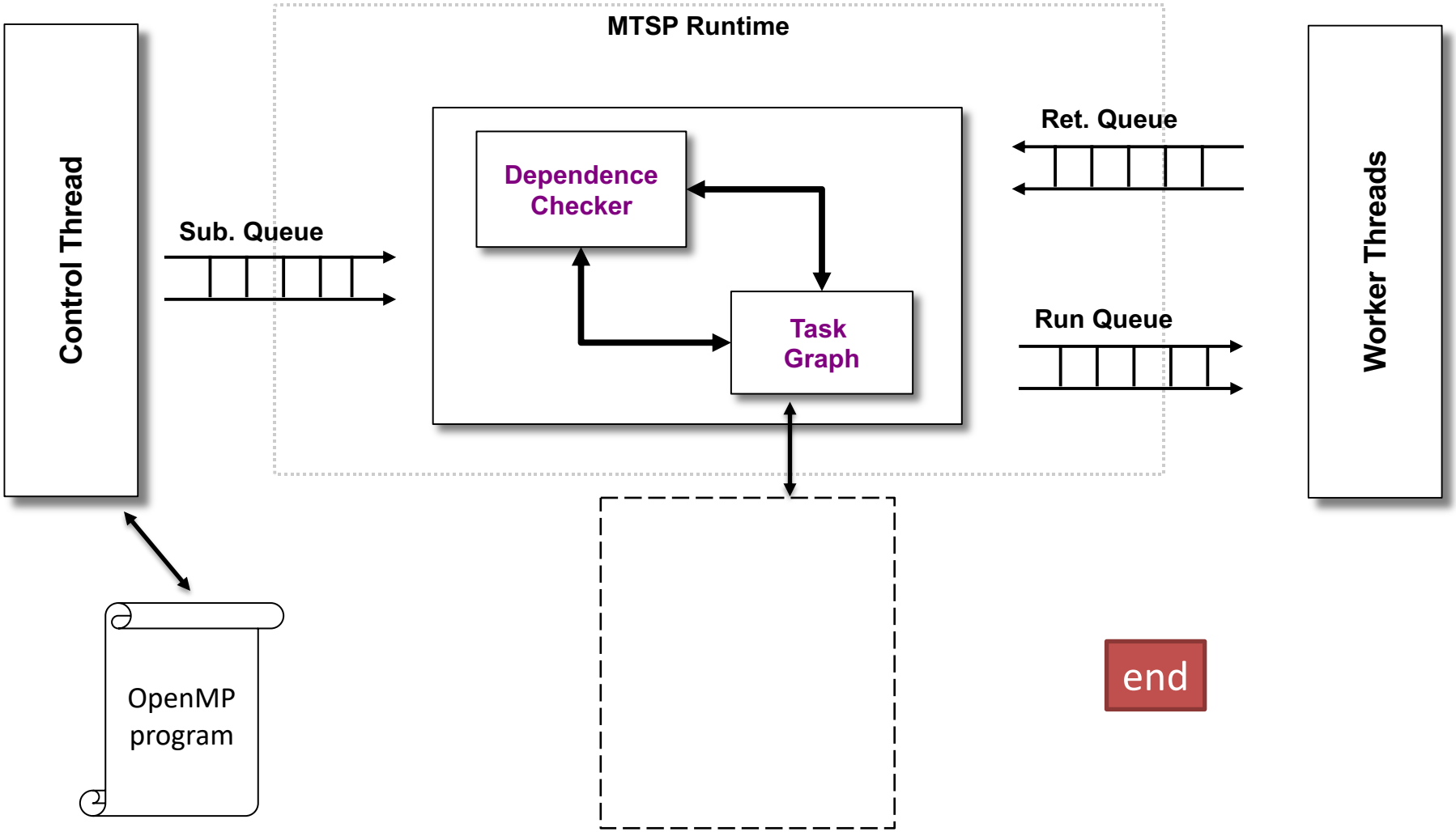
Basic elements of a task scheduling system



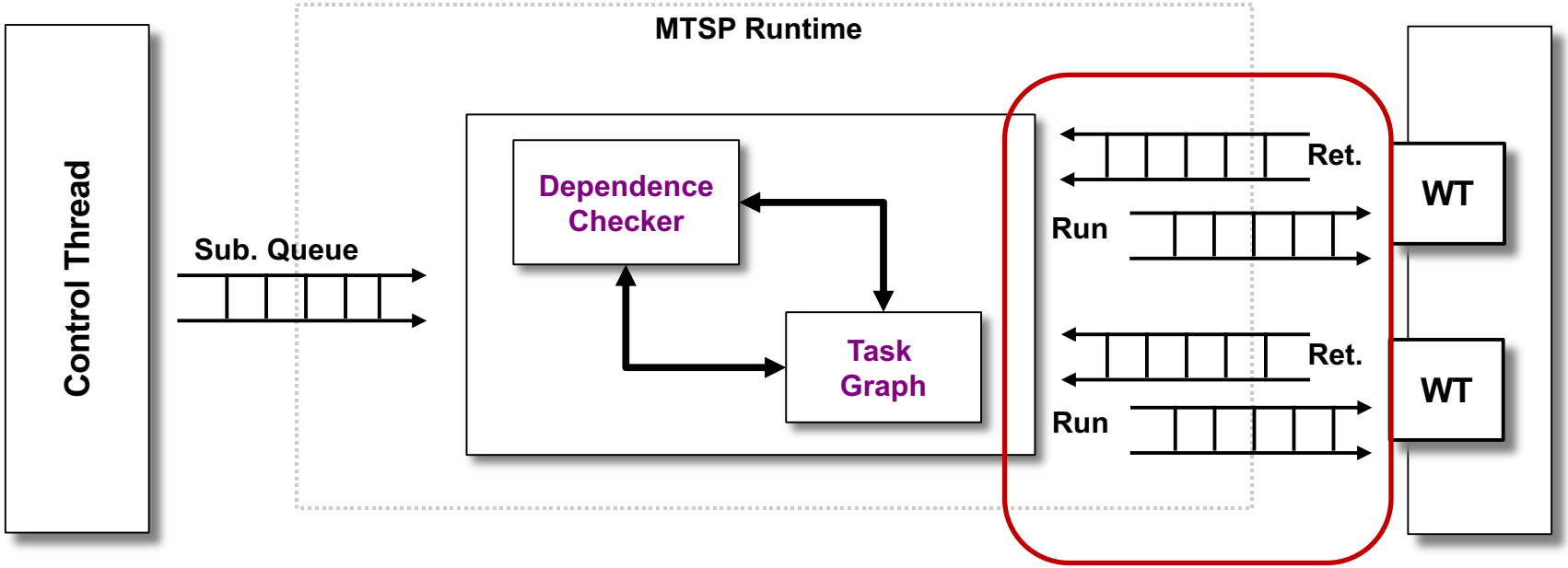
Basic elements of a task scheduling system



Basic elements of a task scheduling system

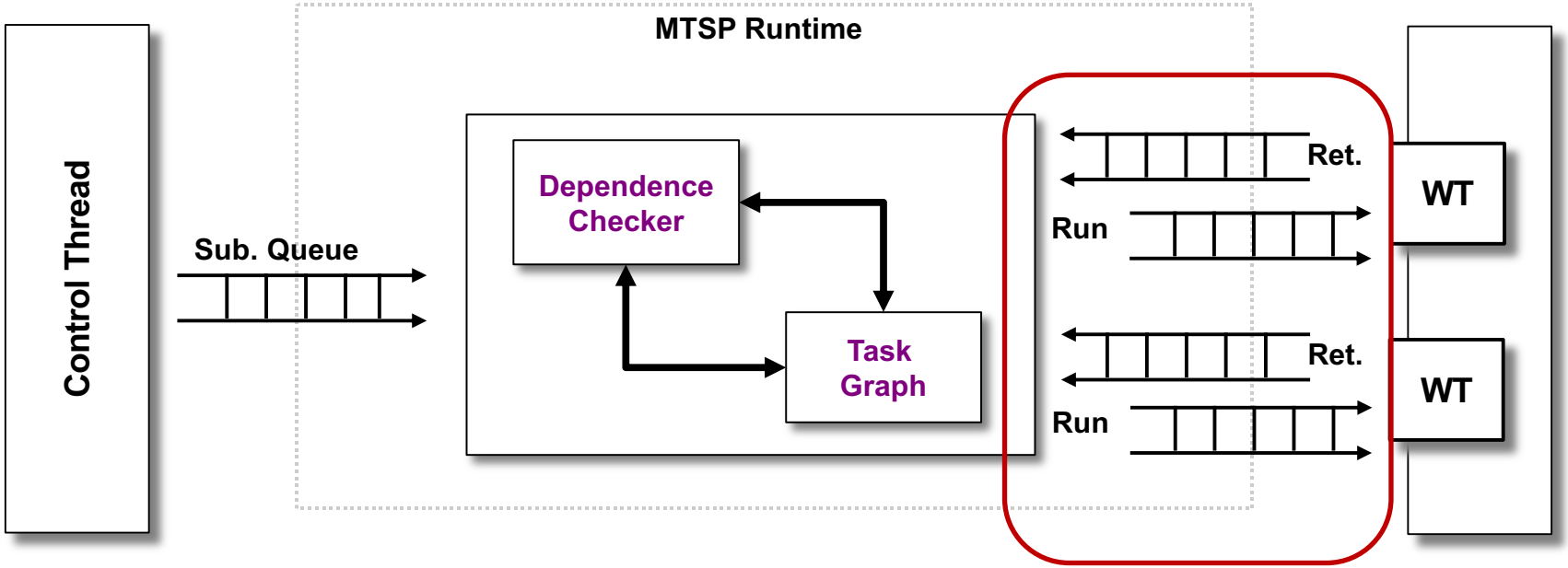


Basic elements of a task scheduling system



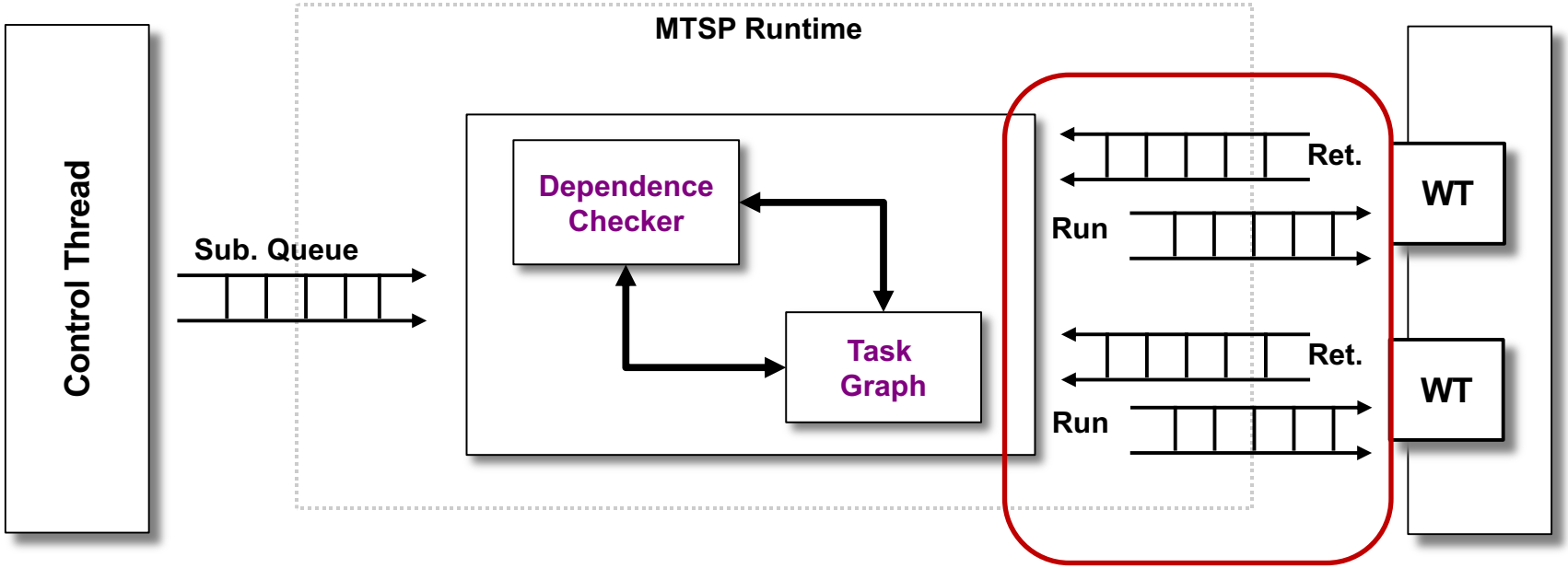
- MTSP actually has a runq/retq pair for each core

Basic elements of a task scheduling system



- MTSP actually has a runq/retq pair for each core
- This avoids lock usage

Basic elements of a task scheduling system

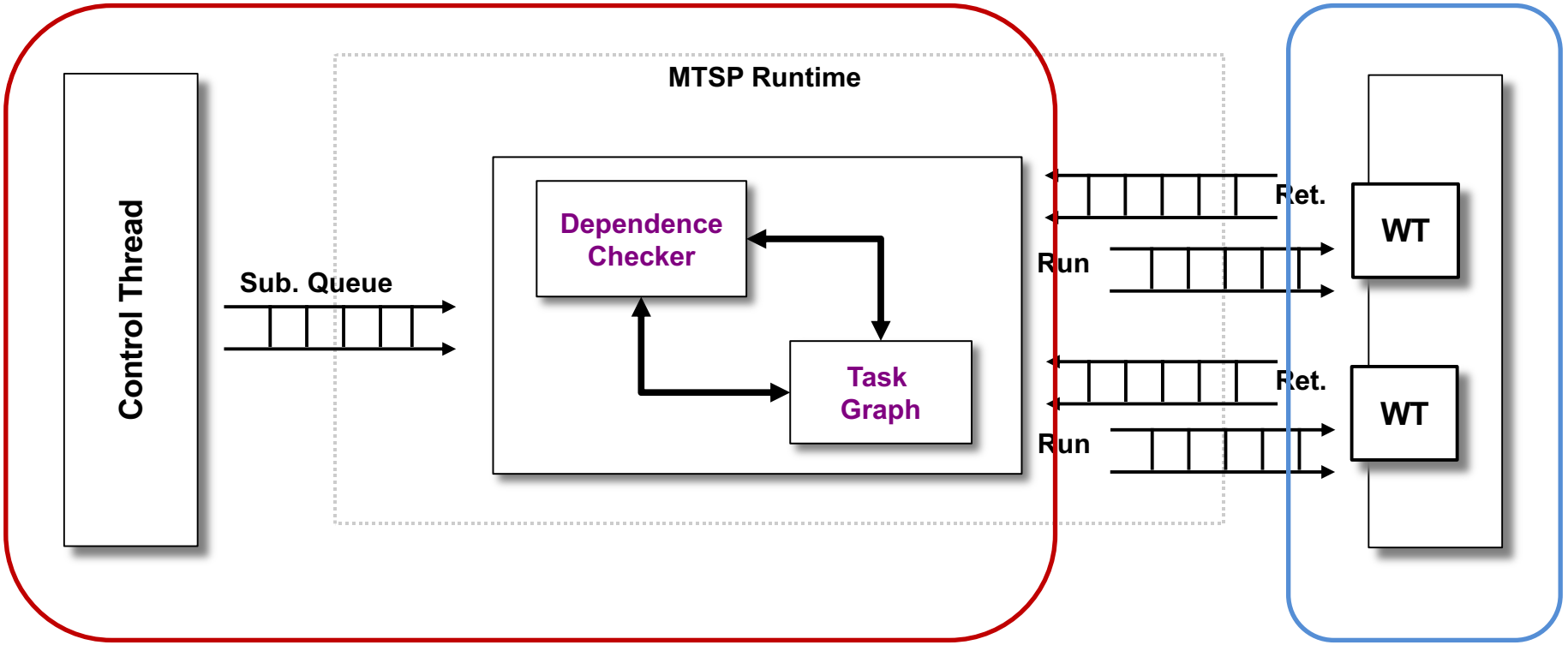


- MTSP actually has a runq/retq pair for each core
- This avoids lock usage
- But we must take care of load balancing

Agenda

- Loop Parallelism
- Task parallelism: programming model
- MTSP: A lightweight runtime
- Where are the limitations?
- Task Graph Accelerator (TGA)
- Collaboration

A small two stage pipe

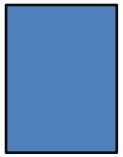


$D_{latency}$

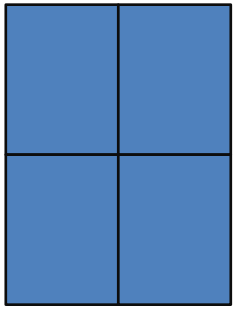
$$\frac{T_{et}}{N_{proc}}$$

When is the pipe balanced?

$$D_{latency} = \frac{T_{et}}{N_{proc}}$$



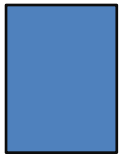
1 task / T_{et}



4 tasks / T_{et}

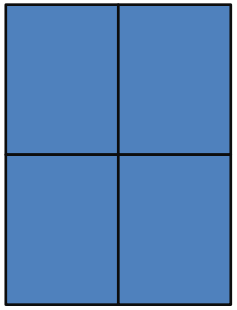
When is the pipe balanced?

$$D_{latency} = \frac{T_{et}}{N_{proc}}$$



1 task / T_{et}

$$N_{proc} < \frac{T_{et}}{D_{latency}}$$



4 tasks / T_{et}

Must keep pipeline balanced!!

When is the pipe balanced?

$$D_{latency} = \frac{T_{et}}{N_{proc}}$$

$$N_{proc} < \frac{T_{et}}{D_{latency}}$$

Upper bound for the number of serviced processors

Software task scheduling limitations

$$D_{latency} = \frac{T_{et}}{N_{proc}}$$

$$\uparrow N_{proc} < \frac{T_{et}}{D_{latency}}$$

- Higher parallelism means higher $N_{proc}\uparrow$

Software task scheduling limitations

$$D_{latency} = \frac{T_{et}}{N_{proc}}$$

$$\uparrow N_{proc} < \frac{\uparrow T_{et}}{D_{latency}}$$

- Higher parallelism means higher N_{proc} ↑
- Higher N_{proc} means either
  - Higher T_{et} ↑
    (coarse grained tasks)

Software task scheduling limitations

$$D_{latency} = \frac{T_{et}}{N_{proc}}$$

$$\uparrow N_{proc} < \frac{T_{et}}{D_{latency}\downarrow}$$

- Higher parallelism means higher N_{proc}
- Higher N_{proc} means either
  - Higher T_{et} (coarse grained tasks)
  - Lower $D_{latency}$ ↓ (faster scheduling)

Software task scheduling limitations

$$D_{latency} = \frac{T_{et}}{N_{proc}}$$

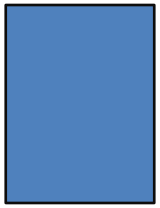
$$N_{proc} < \frac{T_{et}}{D_{latency}}$$

Hardware accelerated task scheduling could change what is possible to do with task parallelism

Software task scheduling limitations

Type A – 6,742 ns

$\uparrow N_{proc}$

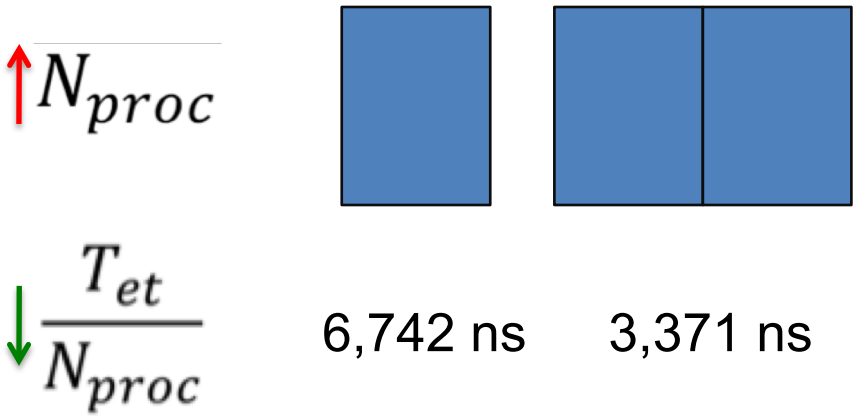


$\downarrow \frac{T_{et}}{N_{proc}}$

6,742 ns

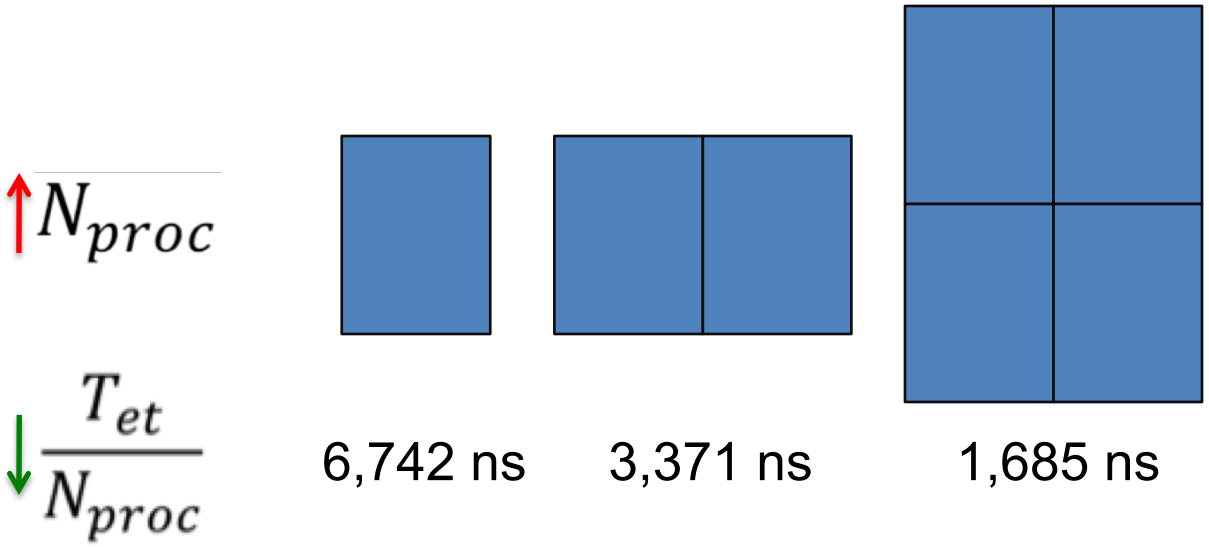
Software task scheduling limitations

Type A – 6,742 ns



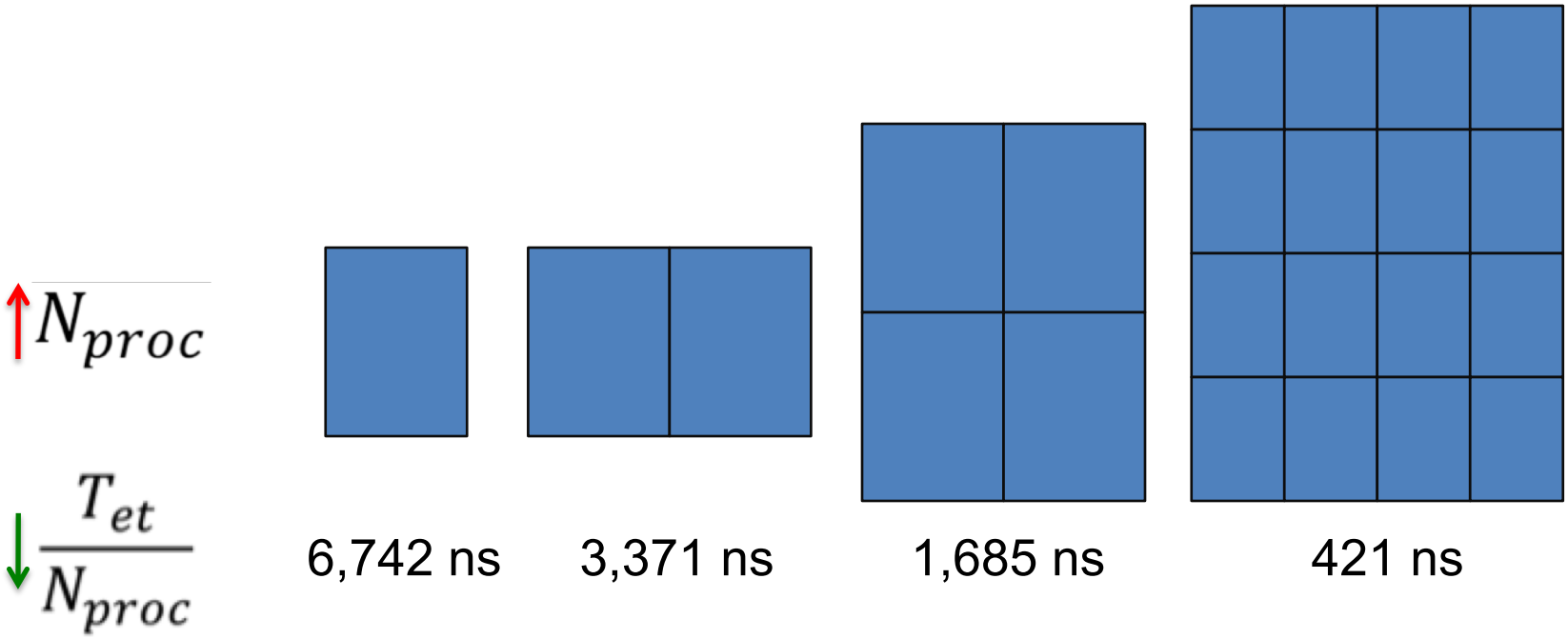
Software task scheduling limitations

Type A – 6,742 ns



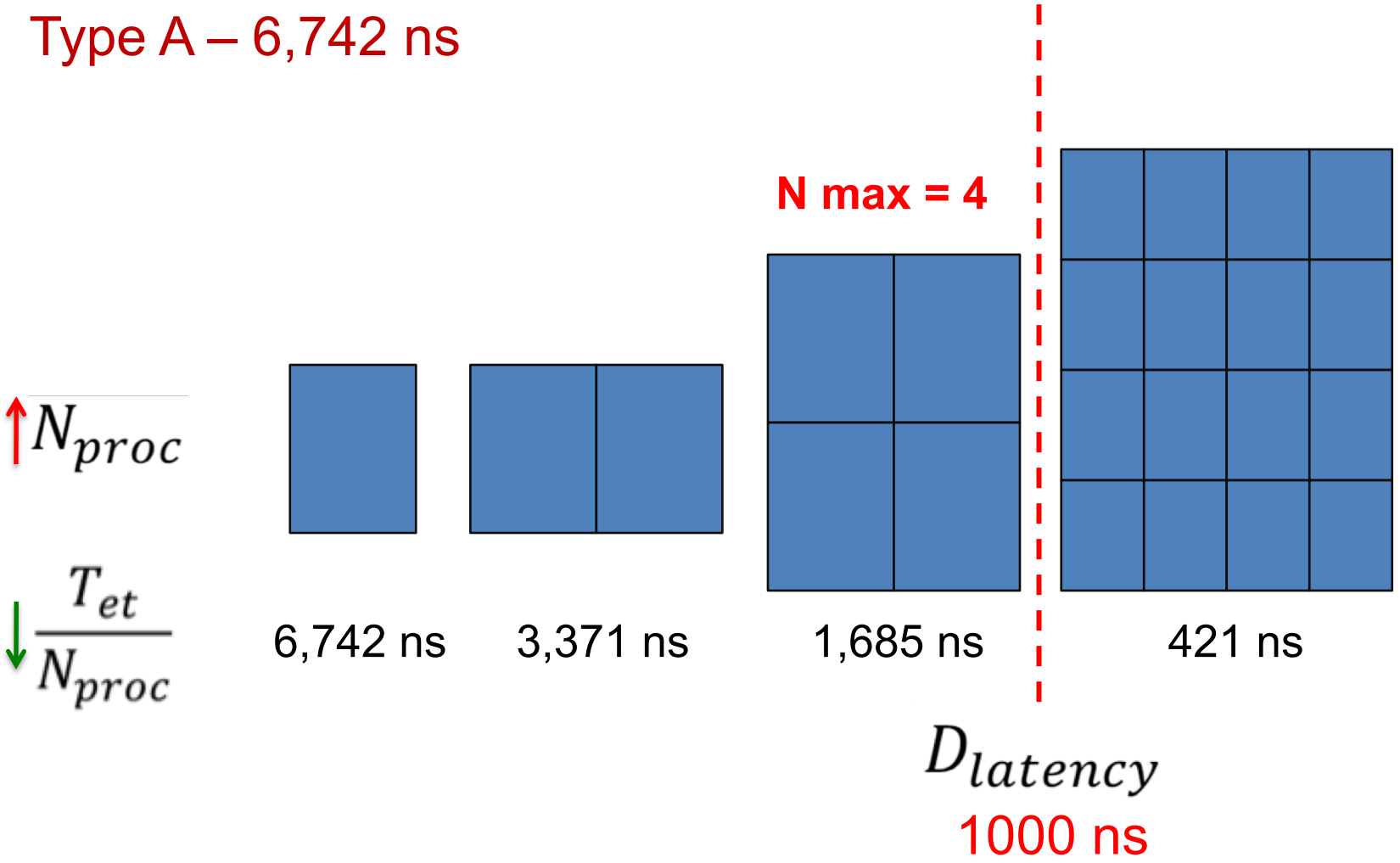
Software task scheduling limitations

Type A – 6,742 ns



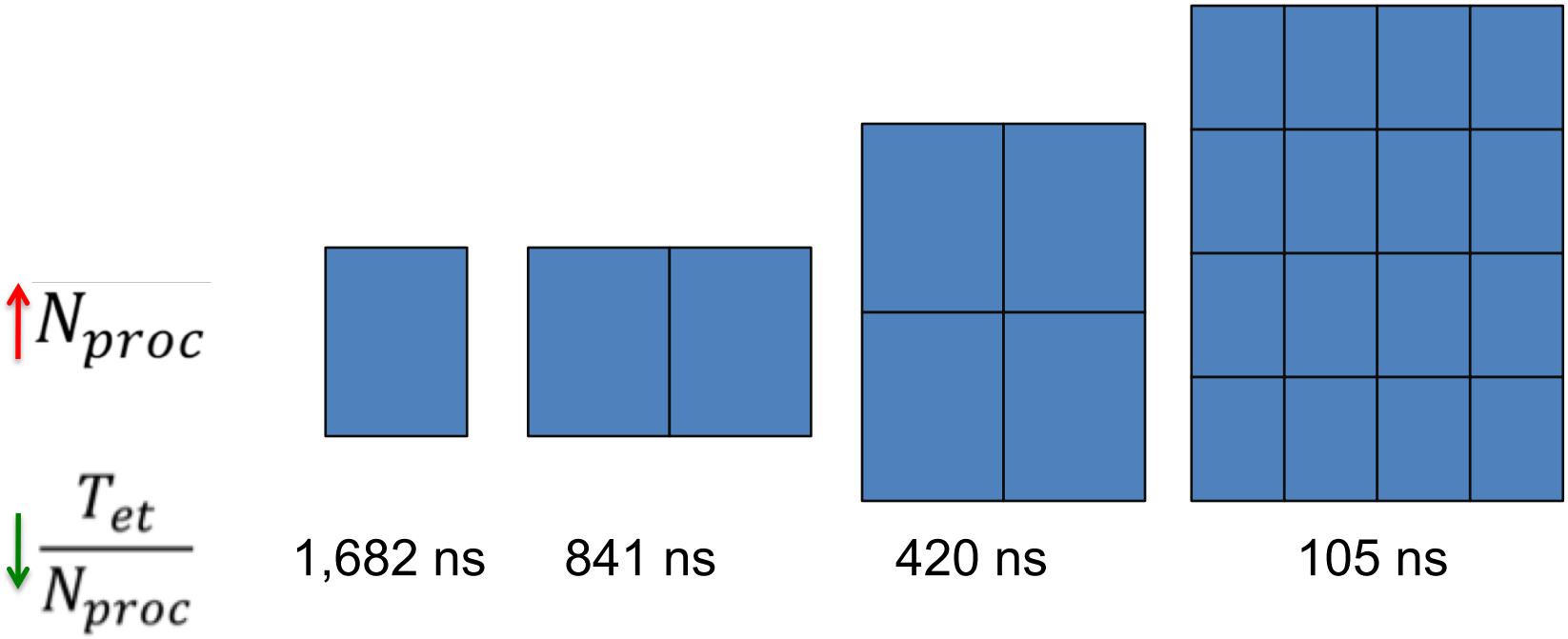
Software task scheduling limitations

Type A – 6,742 ns



Software task scheduling limitations

Type B – 1,682 ns



Software task scheduling limitations