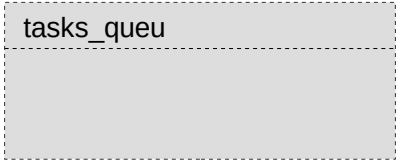
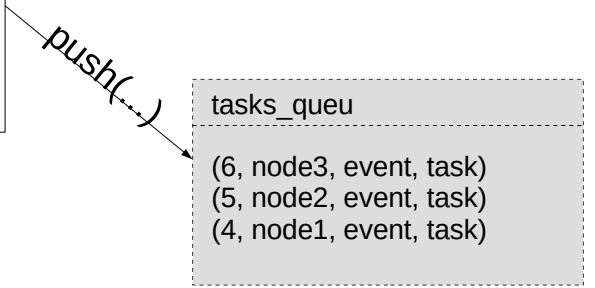
```
def run_udf_locals(udf_name,inputs):

    ids_events=[]
    for node in nodes:
        task=Task(ExecUDF, (udf_name,inputs))
        ( id, event)=tasks_queue.push(node, task)
        ids_events.append((id, event))

    result=[]
    for id, event in ids_events:
        event.wait(timeout=config.timeout)
        try:
            result.append(results[id])

    return result
```

```
class TasksQueue:

    def push(node,task):

        completion_event=threading.Event()
        task_id=task_counter.get_next_id()
        self.tasks_queue.push( (task_id, node, completion_event, task) )

    return ( task_id, completion_event)
```



```
def run_udf_locals(udf_name,inputs):

    ids_events=[]
    for node in nodes:
        task=Task(ExecUDF, (udf_name,inputs))
        ( id, event)=tasks_queue.push(node, task)
        ids_events.append((id, event))

    result=[]
    for id, event in ids_events:
        event.wait(timeout=config.timeout)
        try:
            result.append(results[id])

    return result
```

```
class TasksQueue:

    def push(node,task):

        completion_event=threading.Event()
        task_id=task_counter.get_next_id()
        self.tasks_queue.push( (task_id, node, completion_event, task) )

    return ( task_id, completion_event)
```



```
def run_udf_locals(udf_name,inputs):

    ids_events=[]
    for node in nodes:
        task=Task(ExecUDF, (udf_name,inputs))
        ( id, event)=tasks_queue.push(node, task)
        ids_events.append((id, event))

    result=[]
    for id, event in ids_events:
        event.wait(timeout=config.timeout)
        try:
            result.append(results[id])

    return result
```

```
class TasksQueue:

    def push(node,task):

        completion_event=threading.Event()
        task_id=task_counter.get_next_id()
        self.tasks_queue.push( (task_id, node, completion_event, task) )

    return ( task_id, completion_event)
```

tasks_queu

(6, node3, event, task)
(5, node2, event, task)
(4, node1, event, task)

pop()

```
...

while(!tasks_queue.empty()):
    (id,node, event, task)=tasks_queue.pop()
    threading.Thread( target=exec_task, args=(id, nodes, event, task) )

...
```

```
def exec_task(task_id, node, event, task) :
    response= ## http request to node, with parameters from task ##
    results[task_id]=response
    event.set()
```

AWAITED

```
def run_udf_locals(udf_name,inputs):

    ids_events=[]
    for node in nodes:
        task=Task(ExecUDF, (udf_name,inputs))
        ( id, event)=tasks_queue.push(node, task)
        ids_events.append((id, event))

    result=[]
    for id, event in ids_events:
        event.wait(timeout=config.timeout)
        try:
            result.append(results[id])

    return result
```

```
class TasksQueue:

    def push(node,task):

        completion_event=threading.Event()
        task_id=task_counter.get_next_id()
        self.tasks_queue.push( (task_id, node, completion_event, task) )

    return ( task_id, completion_event)
```

tasks_queu

(6, node3, event, task)
(5, node2, event, task)

```
...

while(!tasks_queue.empty()):
    (id,node, event, task)=tasks_queue.pop()
    threading.Thread( target=exec_task, args=(id, nodes, event, task) )

...
```

```
def exec_task(task_id, node, event, task) :
    response= ## http request to node, with parameters from task ##
    results[task_id]=response
    event.set()
```

AWAITED

results

(4 : [table_name])
(3: [<some response>])
(2: [<some response>])
(1: [<some response>])

```
def run_udf_locals(udf_name,inputs):

    ids_events=[]
    for node in nodes:
        task=Task(ExecUDF, (udf_name,inputs))
        ( id, event)=tasks_queue.push(node, task)
        ids_events.append((id, event))

    result=[]
    for id, event in ids_events:
        event.wait(timeout=config.timeout)
        try:
            result.append(results[id])

    return result
```

```
class TasksQueue:

    def push(node,task):

        completion_event=threading.Event()
        task_id=task_counter.get_next_id()
        self.tasks_queue.push( (task_id, node, completion_event, task) )

    return ( task_id, completion_event)
```

tasks_queu

(6, node3, event, task)
(5, node2, event, task)

```
...

while(!tasks_queue.empty()):
    (id,node, event, task)=tasks_queue.pop()
    threading.Thread( target=exec_task, args=(id, nodes, event, task) )

...
```

```
def exec_task(task_id, node, event, task) :
    response= ## http request to node, with parameters from task ##
    results[task_id]=response
    event.set()
```

AWAITED

results

(4 : [table_name])
(3: [<some response>])
(2: [<some response>])
(1: [<some response>])

```
def run_udf_locals(udf_name,inputs):

    ids_events=[]
    for node in nodes:
        task=Task(ExecUDF, (udf_name,inputs))
        ( id, event)=tasks_queue.push(node, task)
        ids_events.append((id, event))

    result=[]
    for id, event in ids_events:
        event.wait(timeout=config.timeout)
        try:
            result.append(results[id])

    return result
```

```
class TasksQueue:

    def push(node,task):

        completion_event=threading.Event()
        task_id=task_counter.get_next_id()
        self.tasks_queue.push( (task_id, node, completion_event, task) )

    return ( task_id, completion_event)
```

tasks_queu

```
...

while(!tasks_queue.empty()):
    (id,node, event, task)=tasks_queue.pop()
    threading.Thread( target=exec_task, args=(id, nodes, event, task) )

...
```

```
def exec_task(task_id, node, event, task) :
    response= ## http request to node, with parameters from task ##
    results[task_id]=response
    event.set()
```

AWAITED

results

(6 : [table_name])
(5 : [table_name])
(4 : [table_name])
(3: [<some response>])
(2: [<some response>])
(1: [<some response>])

```
def run_udf_locals(udf_name,inputs):

    ids_events=[]
    for node in nodes:
        task=Task(ExecUDF, (udf_name,inputs))
        ( id, event)=tasks_queue.push(node, task)
        ids_events.append((id, event))

    result=[]
    for id, event in ids_events:
        event.wait(timeout=config.timeout)
        try:
            result.append(results[id])

    return result
```

```
class TasksQueue:

    def push(node,task):

        completion_event=threading.Event()
        task_id=task_counter.get_next_id()
        self.tasks_queue.push( (task_id, node, completion_event, task) )

    return ( task_id, completion_event)
```

tasks_queu

[table_name,table_name,table_name]

```
...

while(!tasks_queue.empty()):
    (id,node, event, task)=tasks_queue.pop()
    threading.Thread( target=exec_task, args=(id, nodes, event, task) )

...
```

```
def exec_task(task_id, node, event, task) :
    response= ## http request to node, with parameters from task ##
    results[task_id]=response
    event.set()
```

AWAITED

results

(6 : [table_name])
(5 : [table_name])
(4 : [table_name])
(3: [<some response>])
(2: [<some response>])
(1: [<some response>])