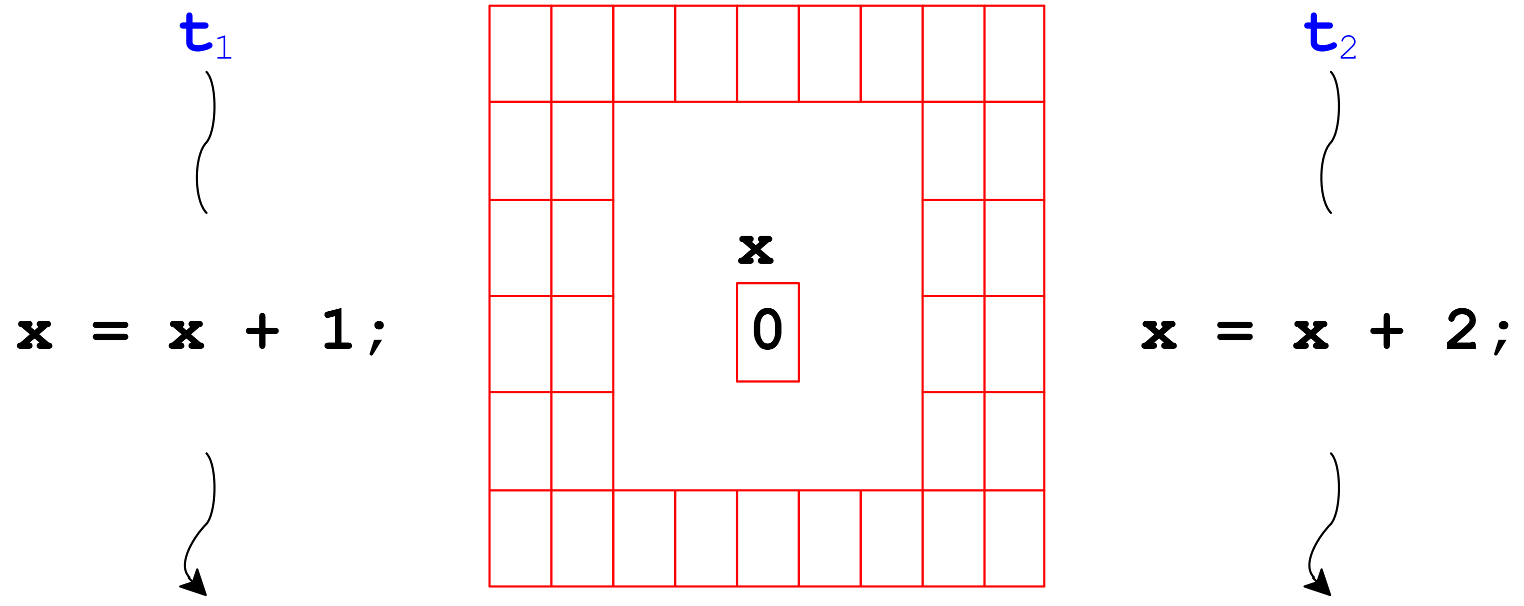
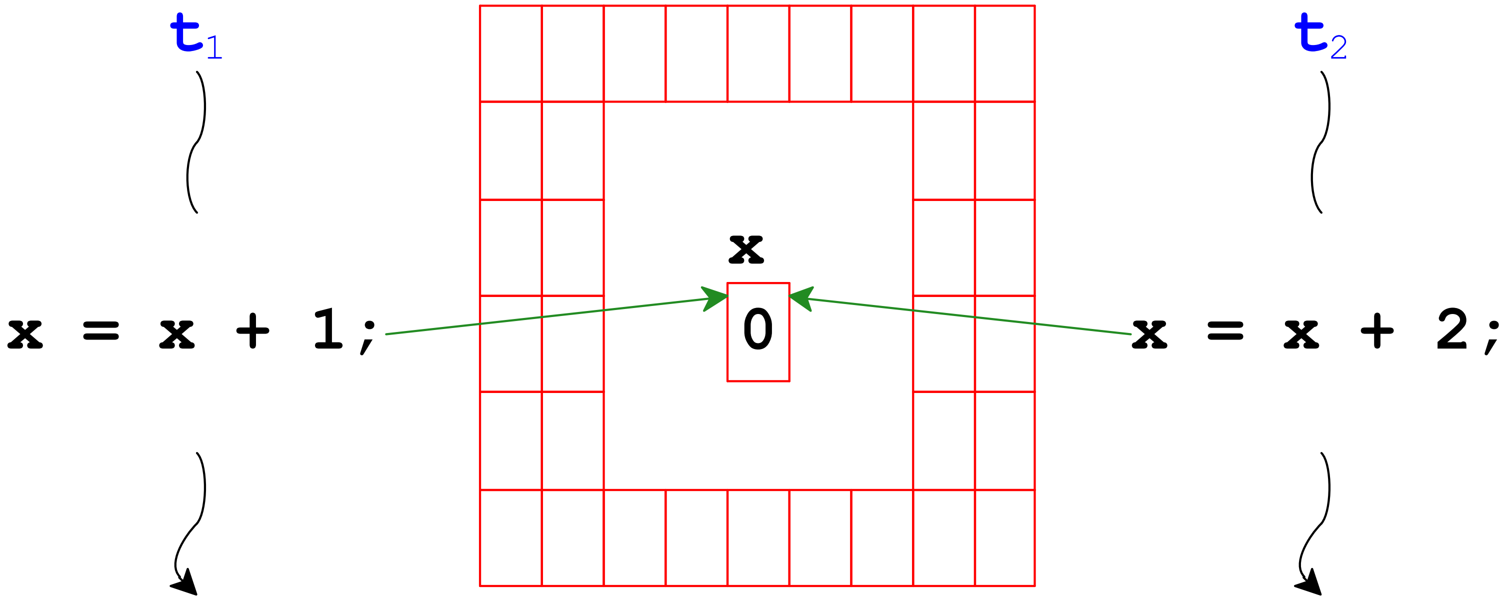
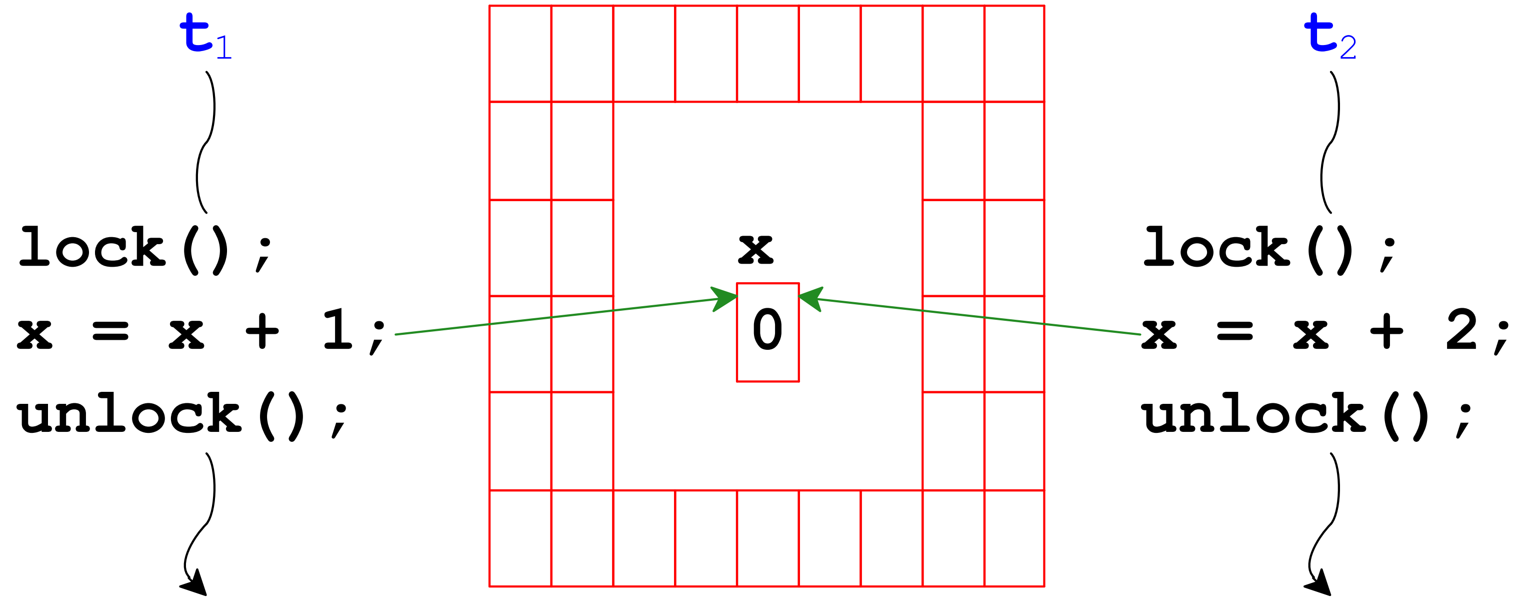
Threads and Shared Memory



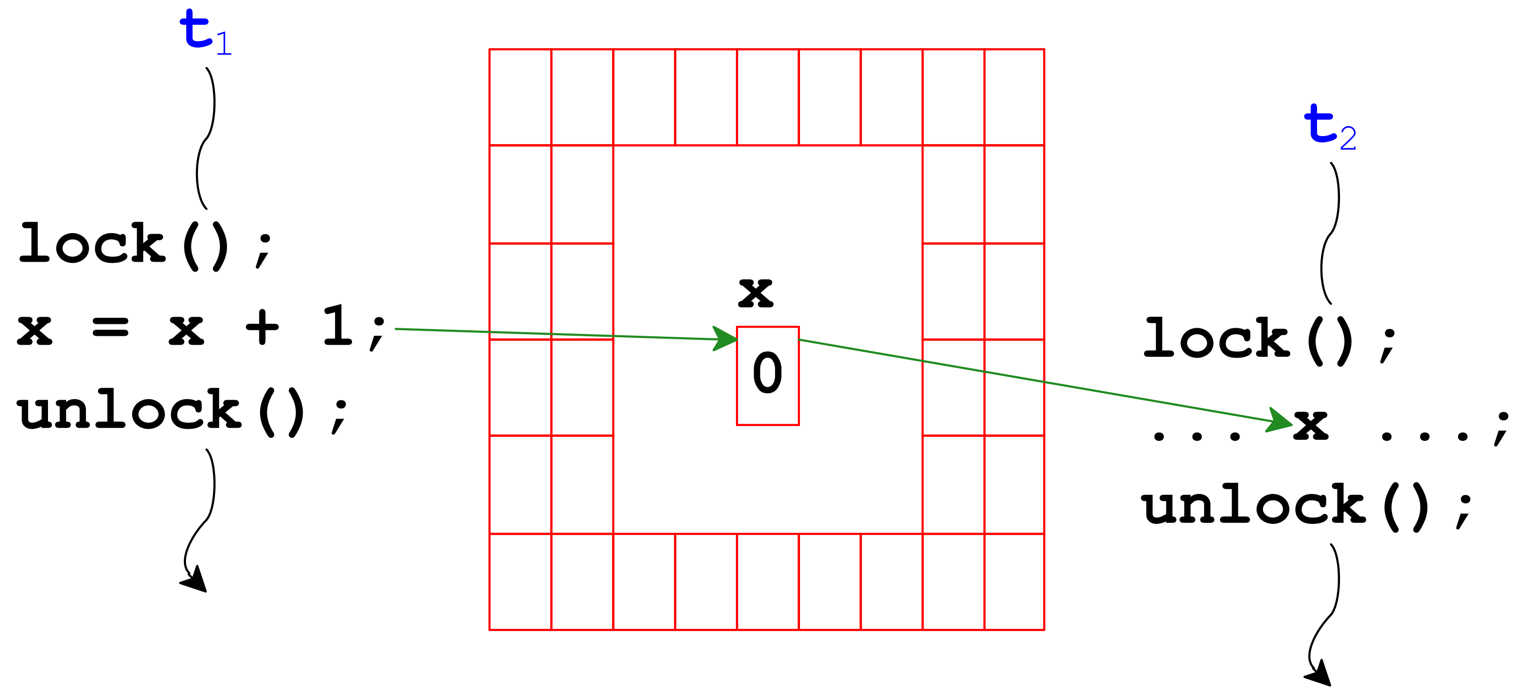
Threads and Shared Memory



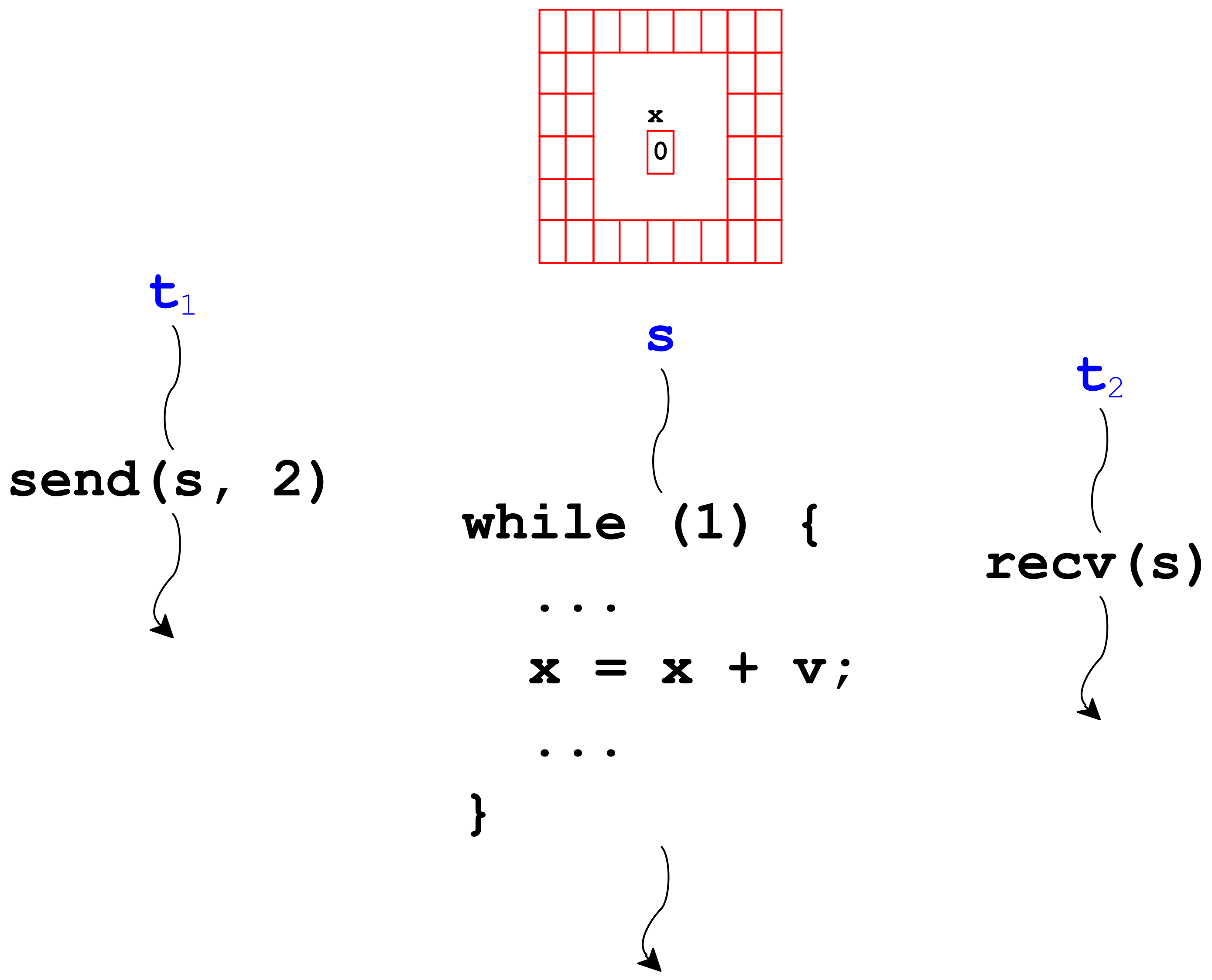
Threads and Shared Memory



Threads and Shared Memory



Message Passing

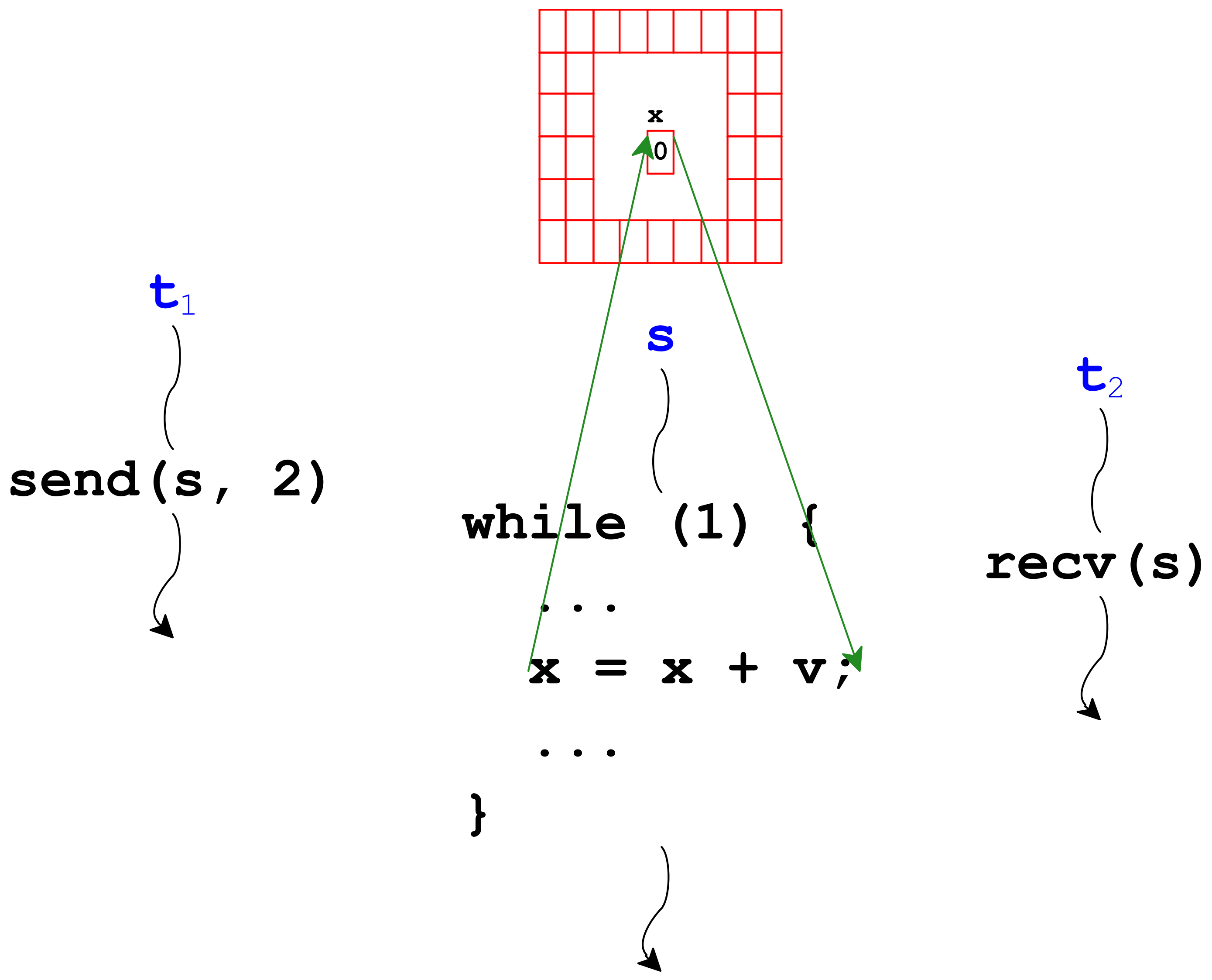


```
send(s, 2)
```

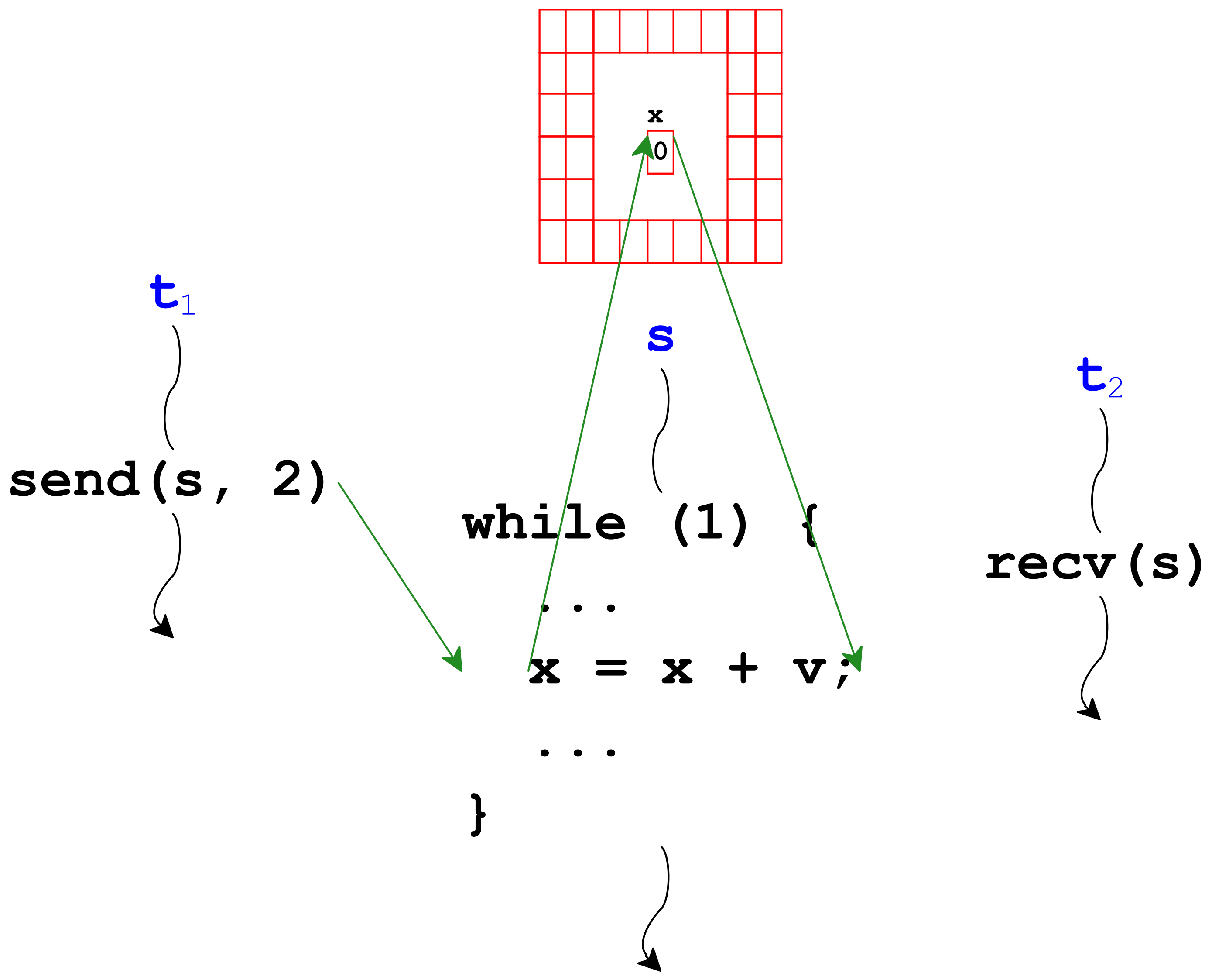
```
while (1) {
   ...
   x = x + v;
   ...
}
```

```
recv(s)
```

Message Passing



Message Passing

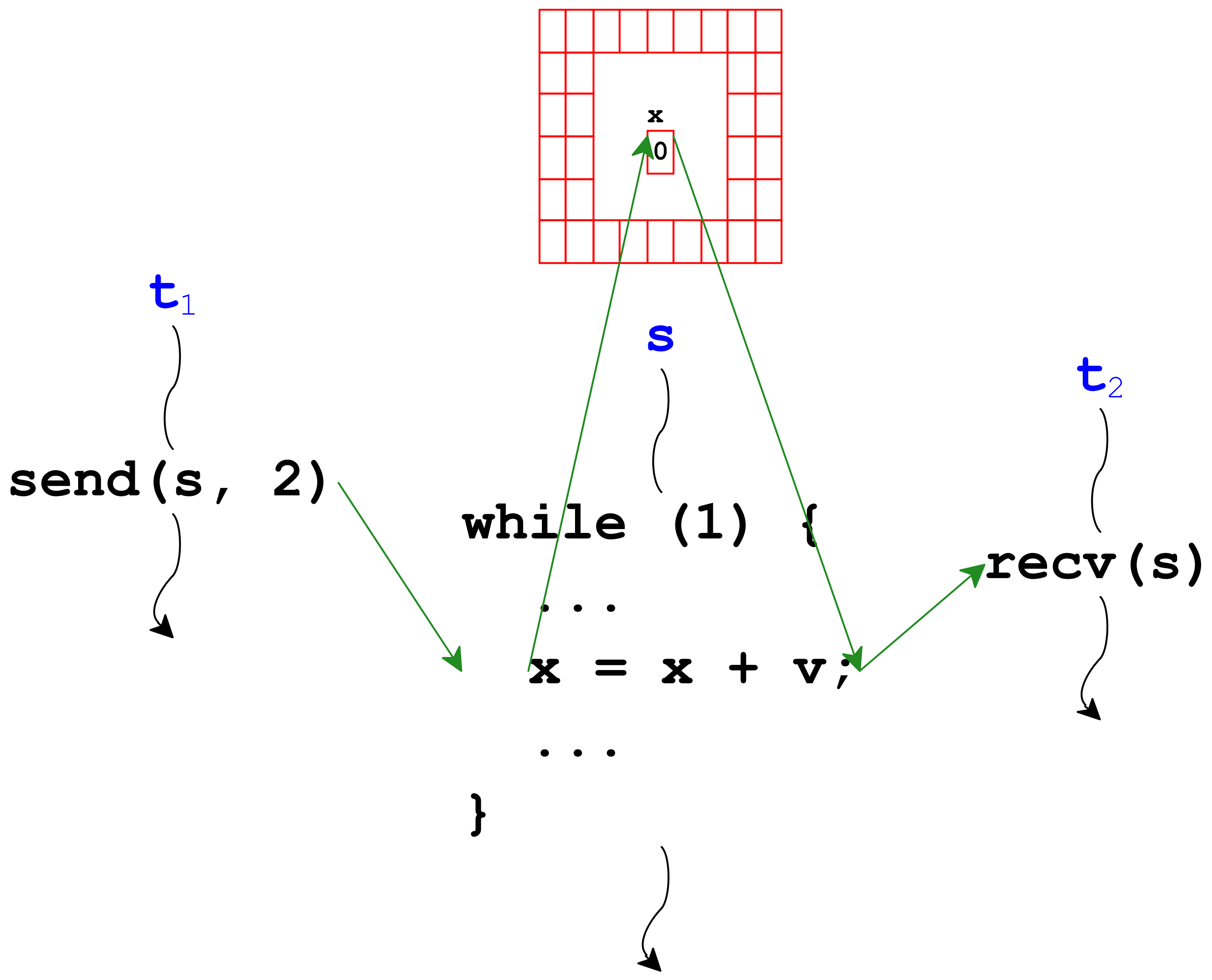


```
send(s, 2)
```

```
while (1) {
   ...
   x = x + v;
   ...
}
```

```
recv(s)
```

Message Passing

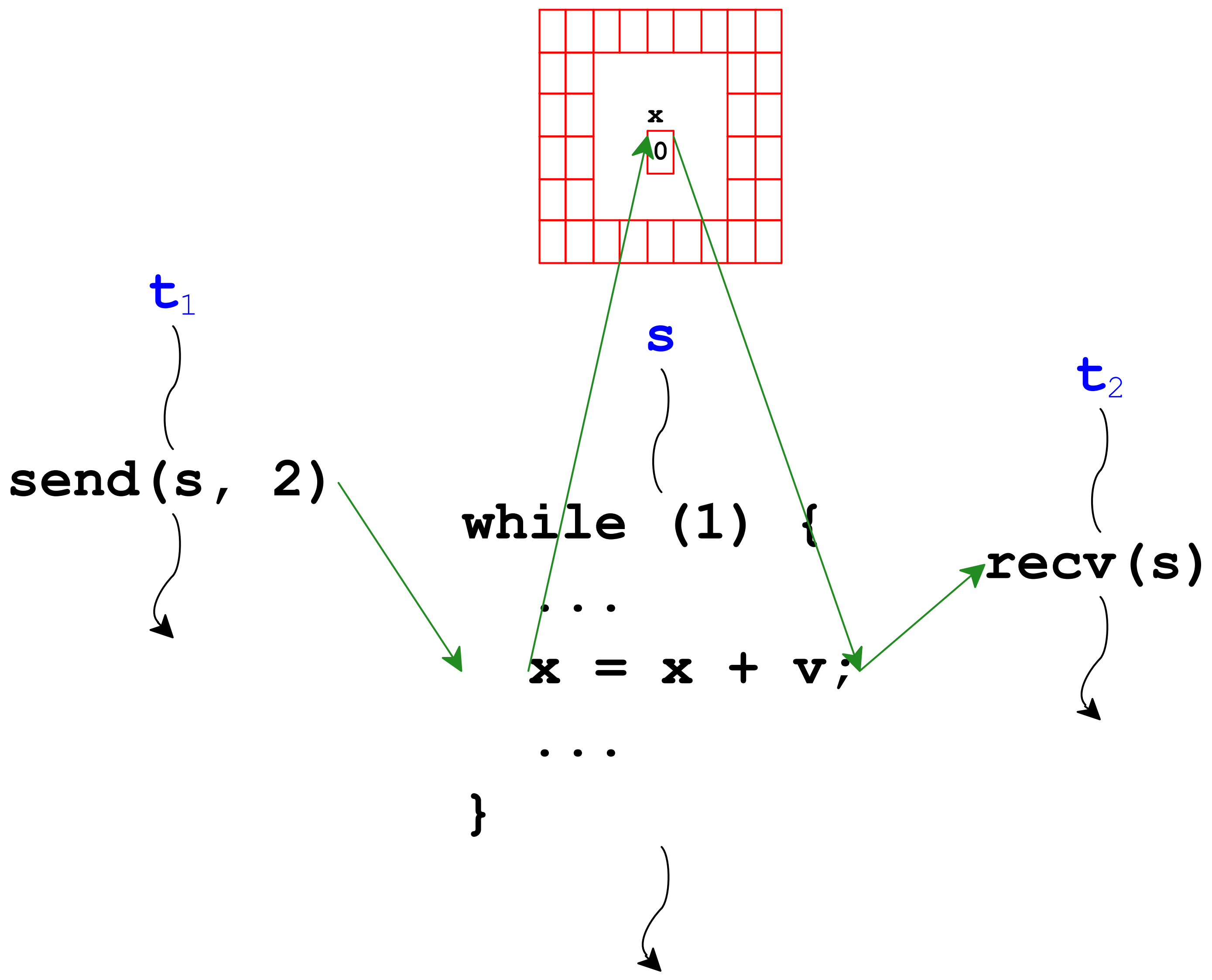


Message Passing in Racket

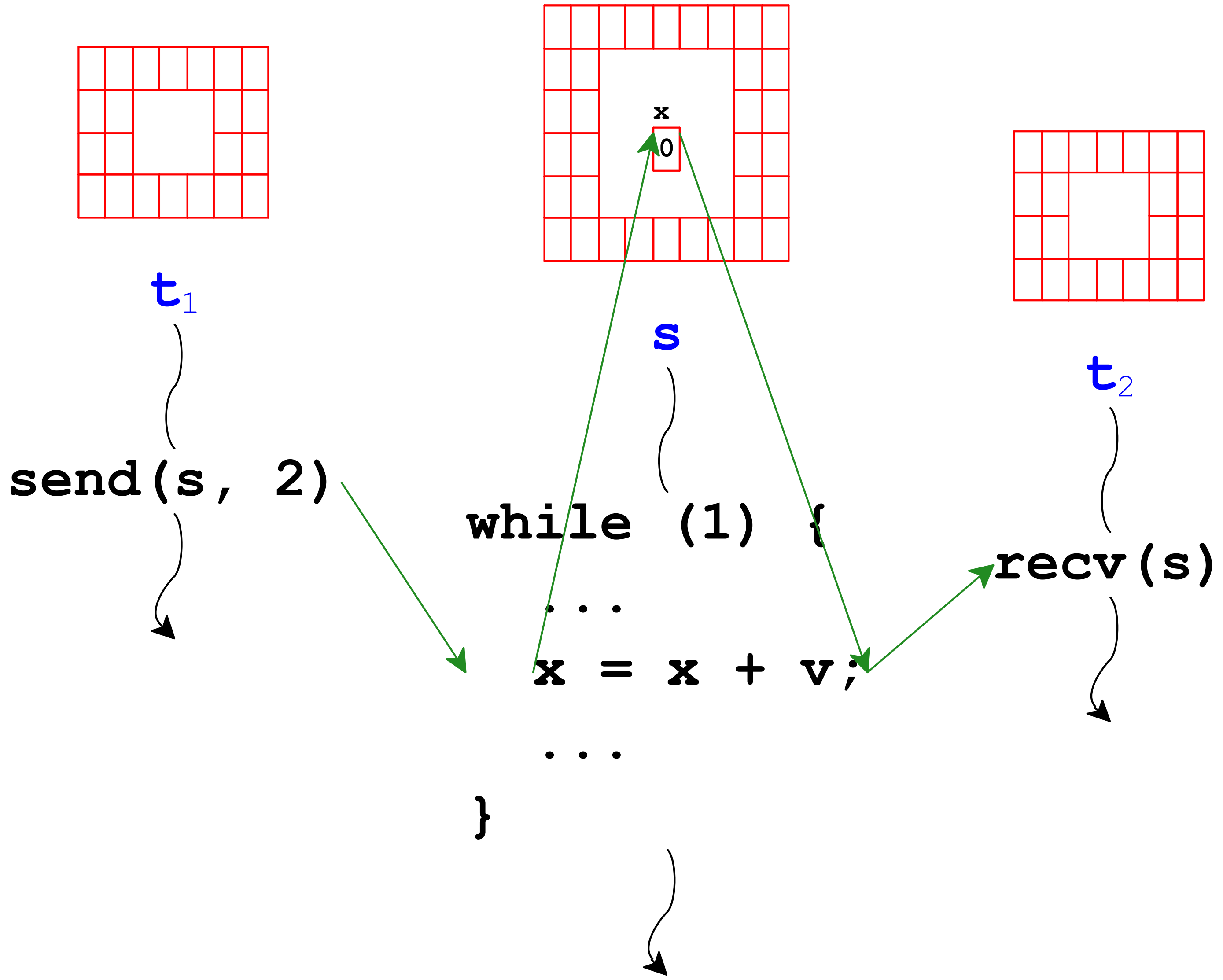
see **`cml`**

```
make-channel : -> channel-of-X

channel-put : channel-of-X X -> void

channel-get : channel-of-X -> X


sync : evt-of-X ... -> X

; An evt-of-X is either
;  - channel-of-X
;  ...


handle-evt : evt-of-X (X -> Y) -> evt-of-Y


never-evt : evt-of-X


channel-put-evt : channel-of-X X -> evt-of-void
```

Message Passing



Distributed Message Passing



Places in Racket

```
(define parent-channel-name
  (place child-channel-name
    expression))
```

Creates a new "copy" of Racket to evaluate `expression`

Messages sent to `parent-channel-name` can be received from `child-channel-name` and vice versa

```
place-channel-get : place-channel-of-X -> X

place-channel-put : place-channel-of-X X -> void
```

see `field/place-player.rkt`

MPI

Message Passing Interface (or ***MPI***) is a message-passing library for many languages

```
int MPI_Send(void *buf, int count,
            MPI_Datatype datatype,
            int dest, int tag,
            MPI_Comm comm);

int MPI_Recv(void *buf, int count,
             MPI_Datatype datatype,
             int source, int tag,
             MPI_Comm comm, MPI_Status *status);
```