

Verifying Concurrent Memory Reclamation Algorithms with Grace

+

From Hoare Logic to Separation Logic

Noam Rinetzky

Tel Aviv University, Israel

1/June/2013

Verifying Concurrent Memory Reclamation Algorithms with Grace

European Symposium on Programming (ESOP) 2013

Alexey Gotsman
Noam Rinetzky
Hongseok Yang

IMDEA, Spain
Tel Aviv University, Israel
University of Oxford, UK

research problem

- verify non-blocking data structures with explicit memory management

non-blocking concurrent data structures

- stack, queue, set
- highly concurrent
 - no global locks protecting the entire DS
 - fine grained concurrency

research problem

- verify non-blocking data structures with explicit memory management
 - memory safety
 - no memory leaks
 - no anomalies due to reallocation (ABA)
 - correctness (atomicity)

concurrency is challenging

- non-blocking concurrent data structures
 - stack, queue, set
- challenges
 - interference
 - pointers
 - dynamic allocation (malloc)

concurrency ^{very} is challenging

- non-blocking concurrent data structures
 - stack, queue, set
- challenges
 - interference
 - pointers
 - dynamic allocation (malloc)
 - deallocation (free)
 - reallocation (free ; malloc)

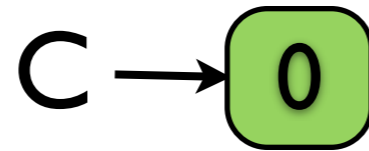
concurrent counter (with GC)

```
int *C;
```

```
int inc() {  
    int v, *s, *n;  
    n = new int;  
    do{  
        s = C;  
        v = *s;  
        *n = v + 1;  
    }while(!CAS(&C,s,n));  
  
    return v;  
}
```


concurrent counter (with GC)

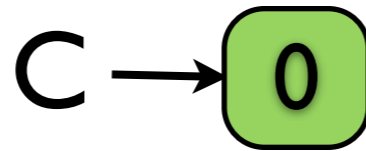
```
int *C;
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  
  return v;  
}
```

concurrent counter (with GC)

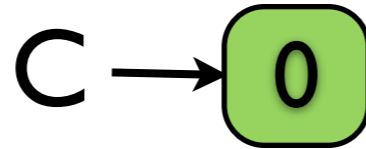
```
int *C;
```



```
int inc() {  
    int v, *s, *n;  
    n = new int;  
    do{  
        s = C;  
        v = *s;  
        *n = v + 1;  
    }while(!CAS(&C,s,n));  
    return v;  
}
```

concurrent counter (with GC)

```
int *C;
```



v

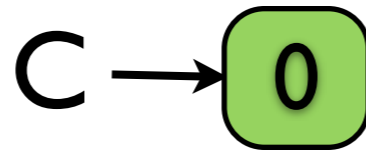
s

n

```
int inc() {  
    int v, *s, *n;  
    n = new int;  
    do{  
        s = C;  
        v = *s;  
        *n = v + 1;  
    }while(!CAS(&C,s,n));  
    return v;  
}
```

concurrent counter (with GC)

```
int *C;
```



v

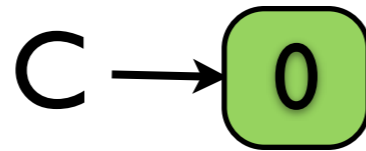
s

n

```
int inc() {  
    int v, *s, *n;  
    n = new int;  
    do{  
        s = C;  
        v = *s;  
        *n = v + 1;  
    }while(!CAS(&C,s,n));  
    return v;  
}
```

concurrent counter (with GC)

```
int *C;
```



v

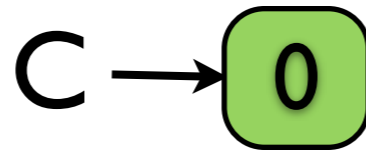
s

n → 

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```

concurrent counter (with GC)

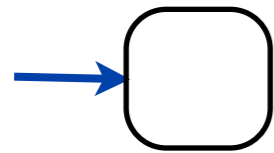
```
int *C;
```



`v`

`s`

`n`

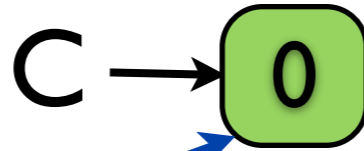


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```



concurrent counter (with GC)

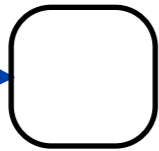
```
int *C;
```



`v`

`s`

`n`

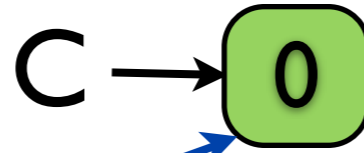


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```



concurrent counter (with GC)

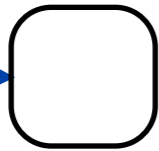
```
int *C;
```



`v`

`s`

`n`

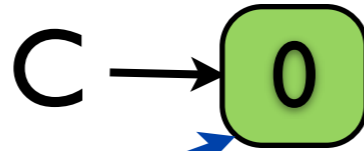


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```



concurrent counter (with GC)

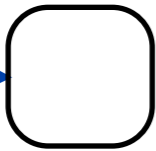
```
int *C;
```



`v = 0`

`s`

`n`

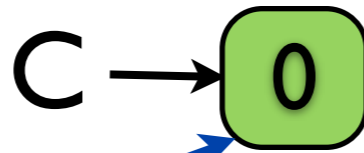


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```



concurrent counter (with GC)

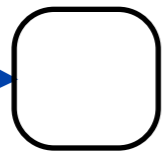
```
int *C;
```



`v = 0`

`s`

`n`

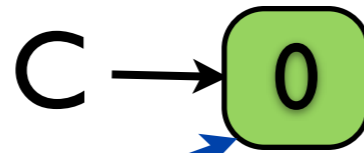


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```



concurrent counter (with GC)

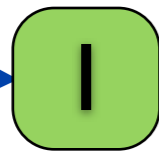
```
int *C;
```



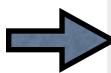
`v = 0`

`s`

`n`

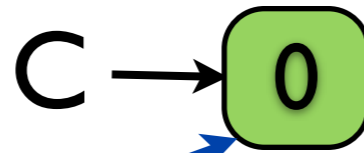


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```



concurrent counter (with GC)

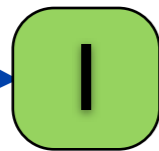
```
int *C;
```



v = 0

s

n

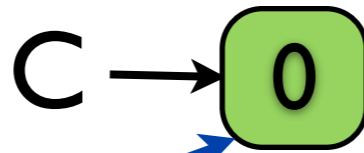


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```



concurrent counter (with GC)

```
int *C;
```



v = 0

s

n

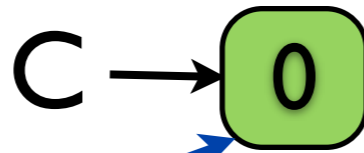


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
  <if (C==s)  
    C=n;  
    return true  
  else  
    return false>
```

concurrent counter (with GC)

```
int *C;
```



v = 0

s

n



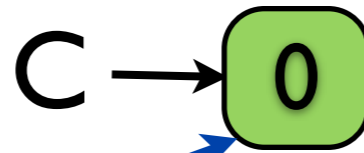
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```

```
CAS(&C,s,n) :=  
  <if(C==s)  
    C=n;  
    return true  
  else  
    return false>
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```

concurrent counter (with GC)

```
int *C;
```



v = 0

s

n



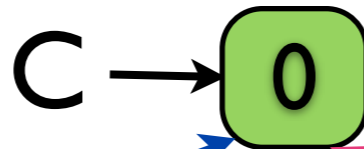
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
< if (C == s)  
    C = n;  
    return true  
else  
    return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

concurrent counter (with GC)

```
int *C;
```



v = 0

s

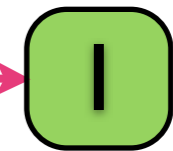
n



v = 0

s

n



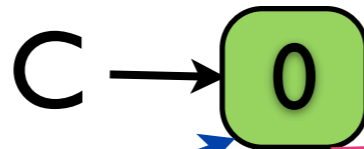
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
  <if (C==s)  
    C=n;  
    return true  
  else  
    return false>
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```


concurrent counter (with GC)

```
int *C;
```



v = 0

s

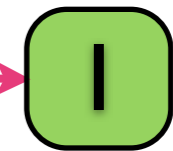
n



v = 0

s

n



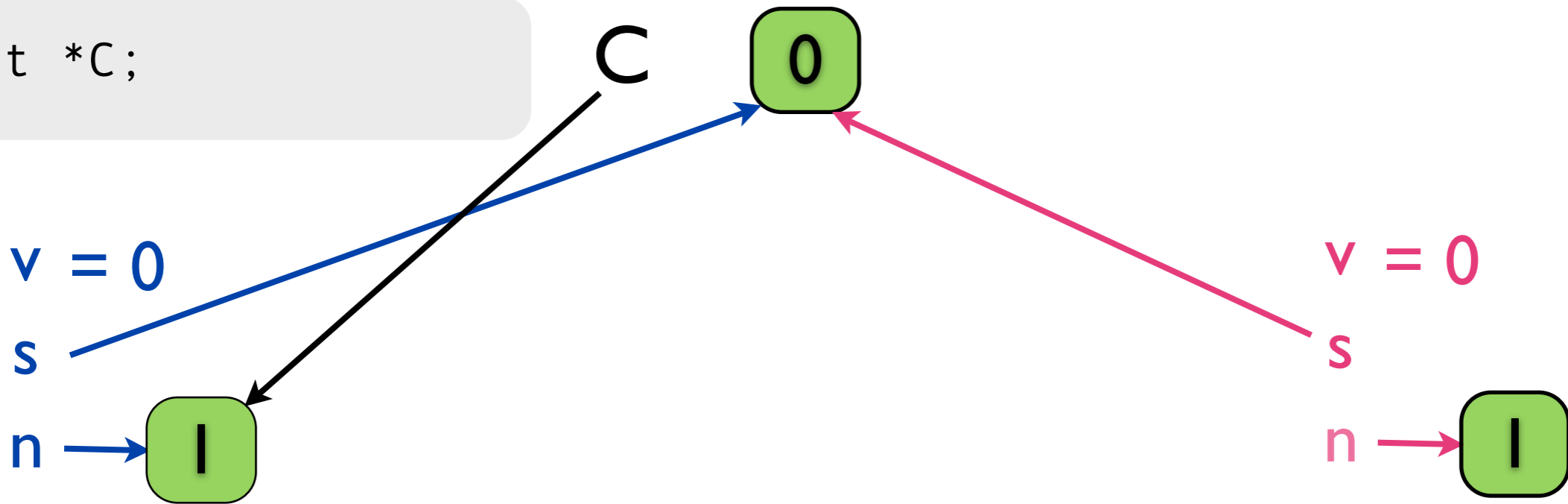
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
< if (C == s)  
    C = n;  
    return true  
else  
    return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

concurrent counter (with GC)

```
int *C;
```



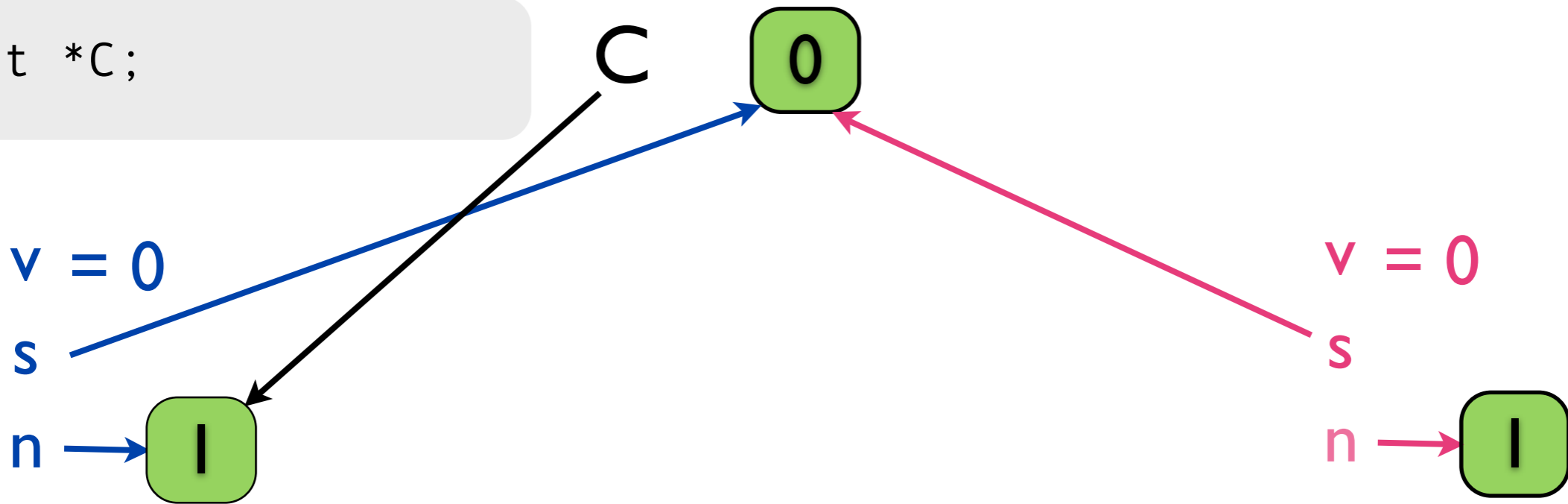
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
<if (C==s)  
  C=n;  
  return true  
else  
  return false>
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

concurrent counter (with GC)

```
int *C;
```



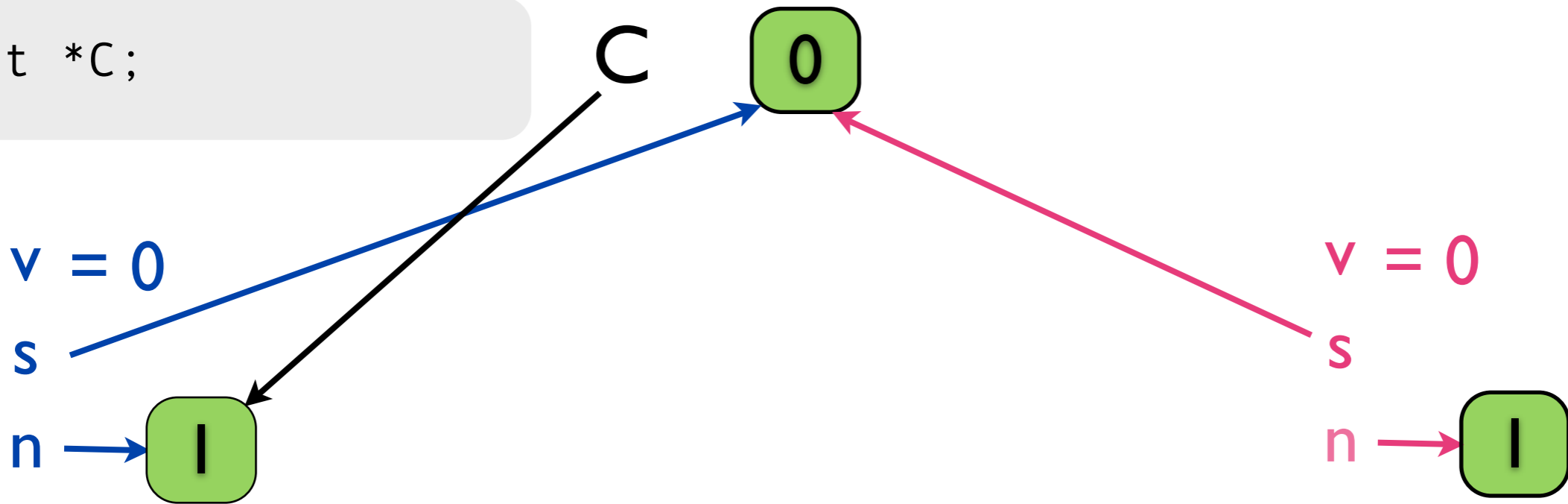
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
< if (C == s)  
    C = n;  
    return true  
else  
    return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

concurrent counter (with GC)

```
int *C;
```



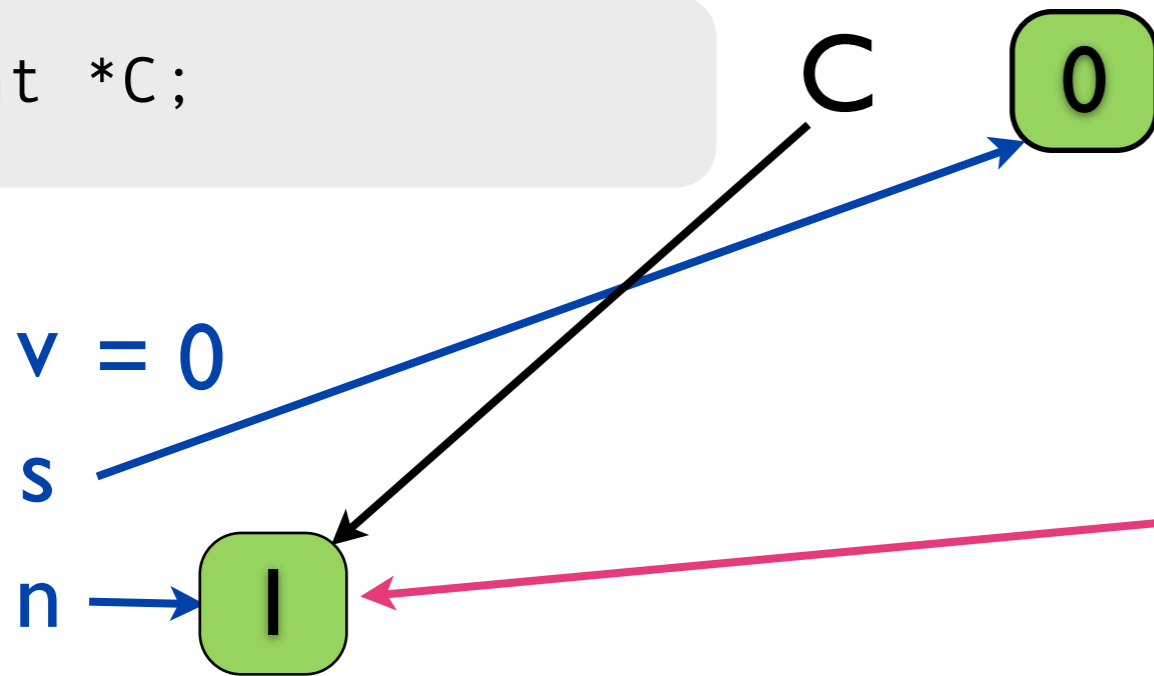
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
  <if (C==s)  
    C=n;  
    return true  
  else  
    return false>
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

concurrent counter (with GC)

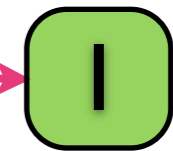
```
int *C;
```



`v = 0`

`s`

`n`



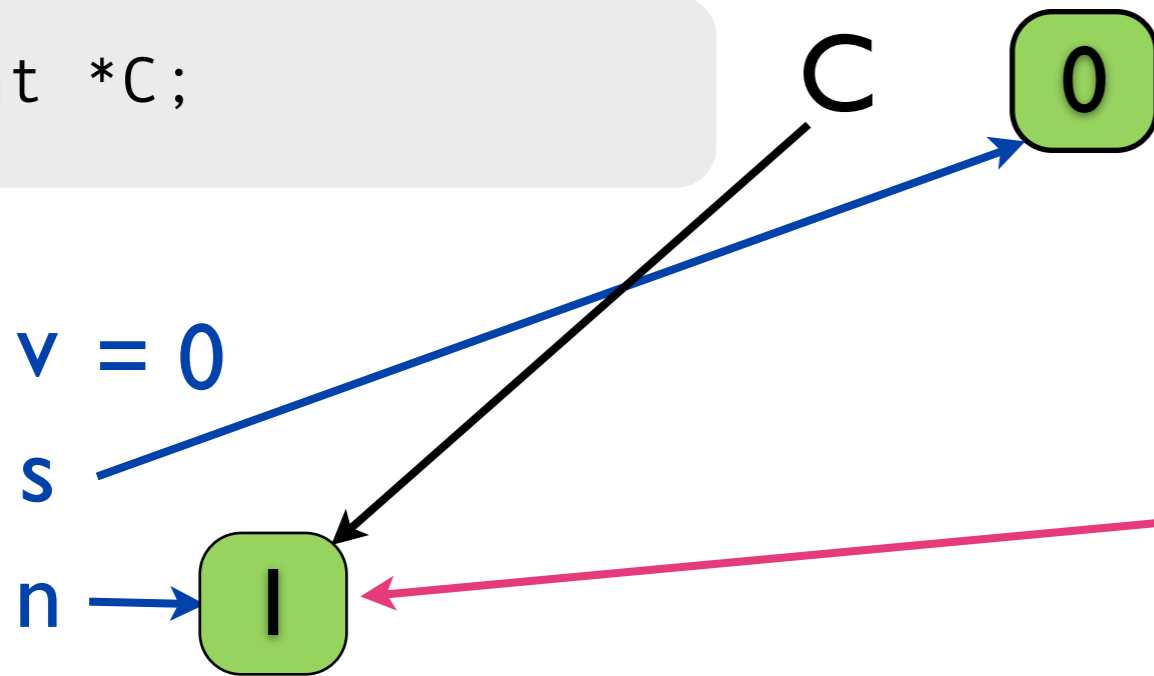
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
<if (C==s)  
  C=n;  
  return true  
else  
  return false>
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

concurrent counter (with GC)

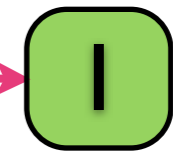
```
int *C;
```



`v = 0`

`s`

`n`



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
  <if (C==s)  
    C=n;  
    return true  
  else  
    return false>
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

concurrent counter (with GC)

```
int *C;
```

C

0

1

v = 0

s

n

1

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

```
CAS(&C, s, n) :=  
< if (C == s)  
    C = n;  
    return true  
else  
    return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

concurrent counter (with GC)

```
int *C;
```

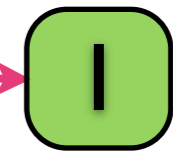
C



v = 0

s

n



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```

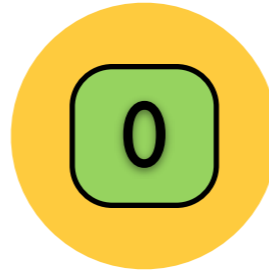
```
CAS(&C,s,n) :=  
< if (C==s)  
    C=n;  
    return true  
else  
    return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  return v;  
}
```


no GC \Rightarrow memory leaks

```
int *C;
```

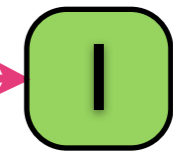
C



v = 0

s

n



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

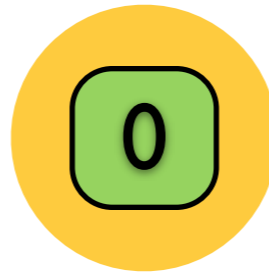
```
CAS(&C, s, n) :=  
< if (C == s)  
    C = n;  
    return true  
else  
    return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  return v;  
}
```

naive fix

```
int *C;
```

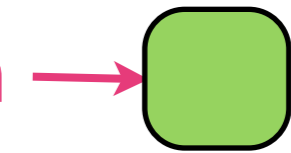
C



v = 0

s

n



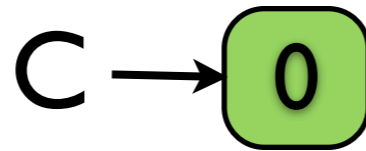
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
CAS(&C, s, n) :=  
< if (C == s)  
    C = n;  
    return true  
else  
    return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

```
int *C;
```

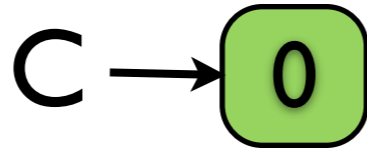


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

```
int *C;
```



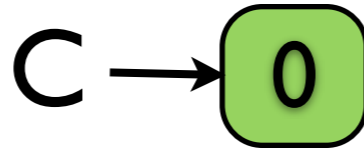
→

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

```
int *C;
```



v

s

n

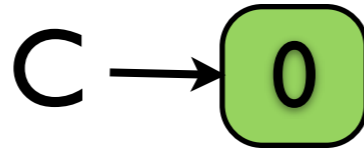
→

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

```
int *C;
```



v

s

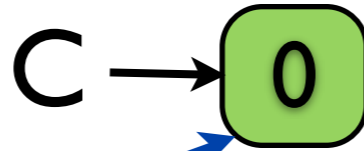
n

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

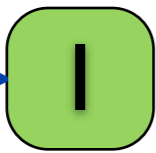
```
int *C;
```



v = 0

s

n

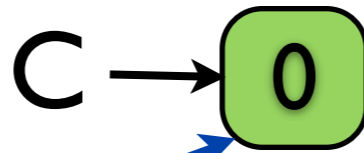


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

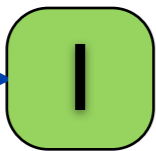
```
int *C;
```



v = 0

s

n



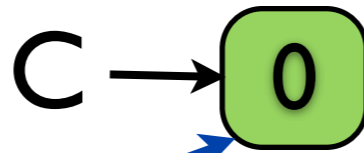
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```


problem: memory safety

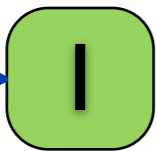
```
int *C;
```



v = 0

s

n



v

s

n

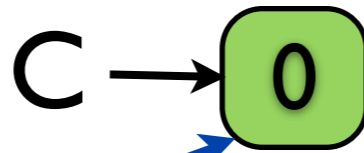
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

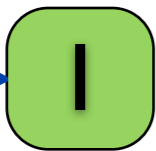
```
int *C;
```



v = 0

s

n



v

s

n

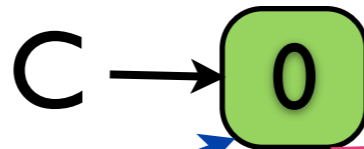
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

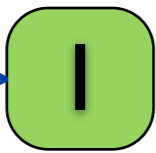
```
int *C;
```



v = 0

s

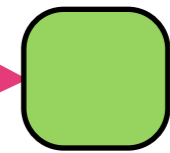
n



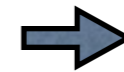
v

s

n



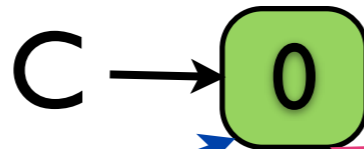
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

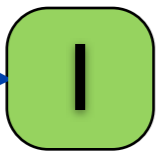
```
int *C;
```



v = 0

s

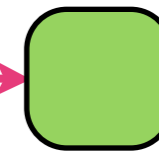
n



v

s

n



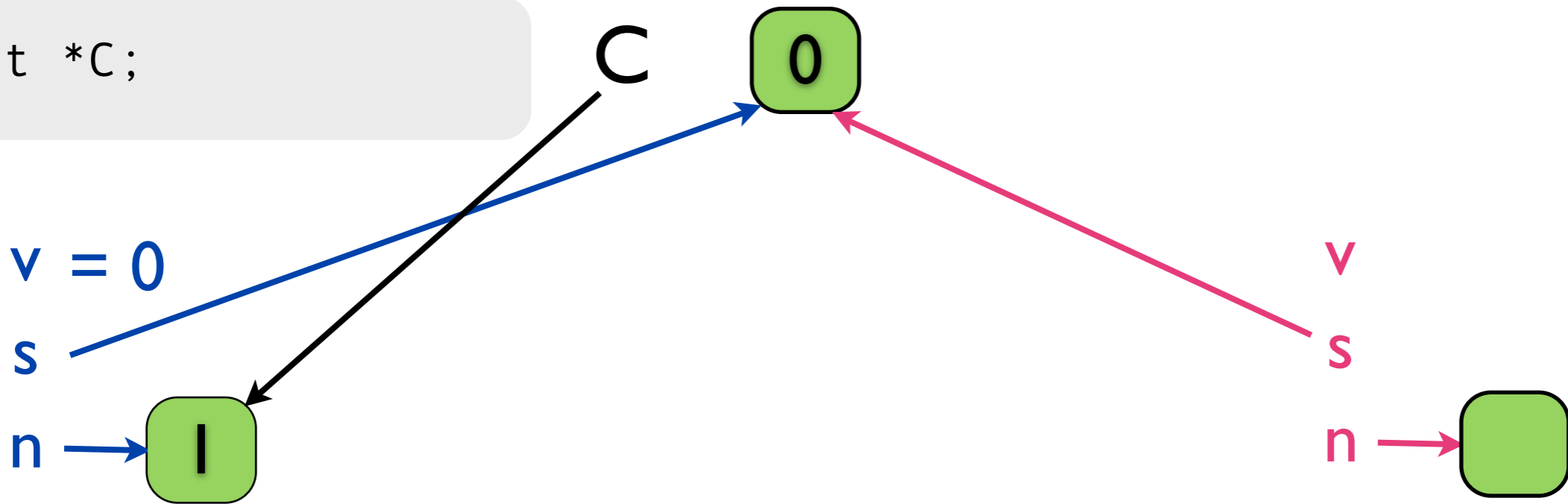
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

```
int *C;
```

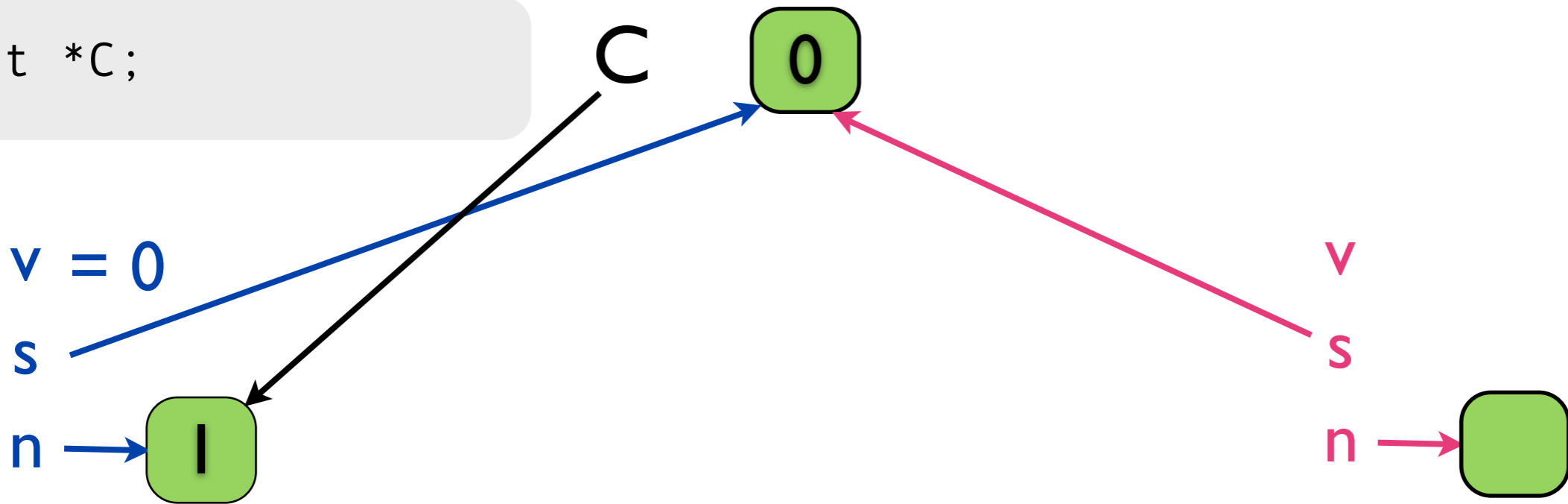


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

```
int *C;
```

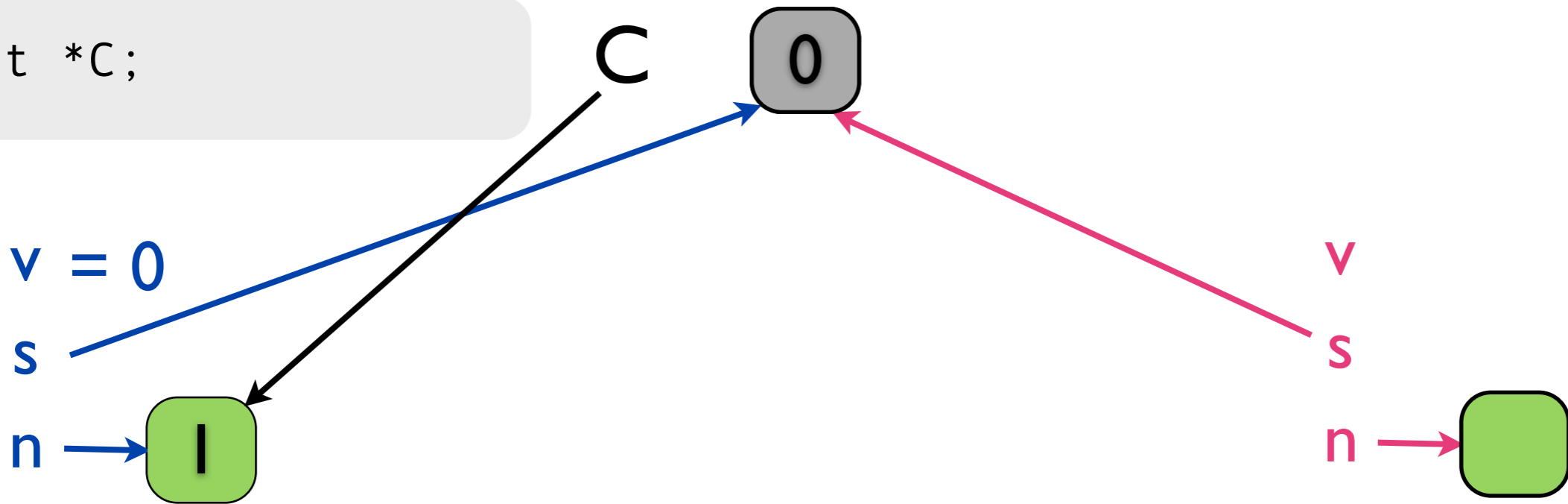


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

```
int *C;
```

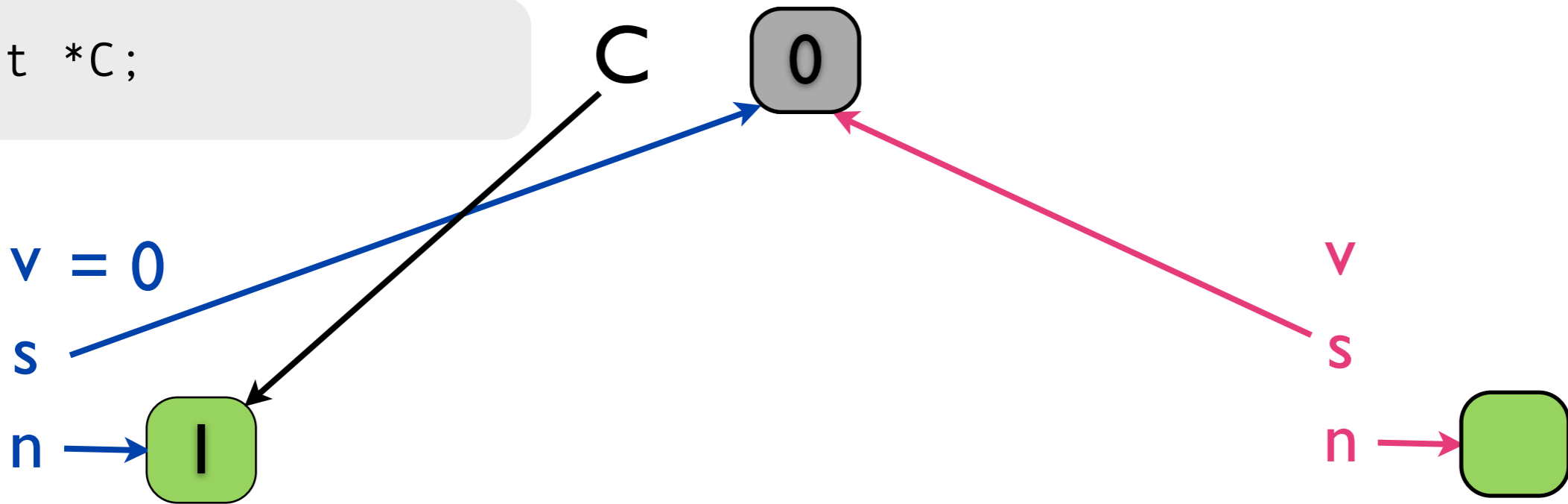


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: memory safety

```
int *C;
```

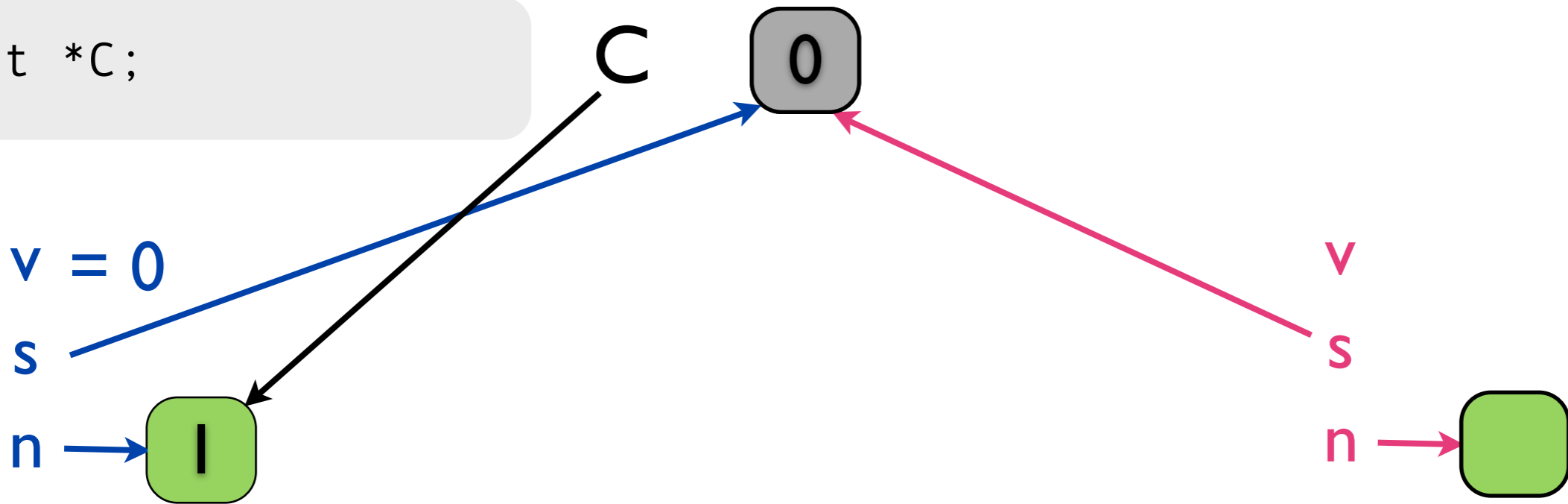


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

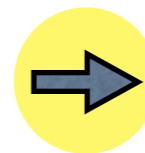

problem: memory safety

```
int *C;
```



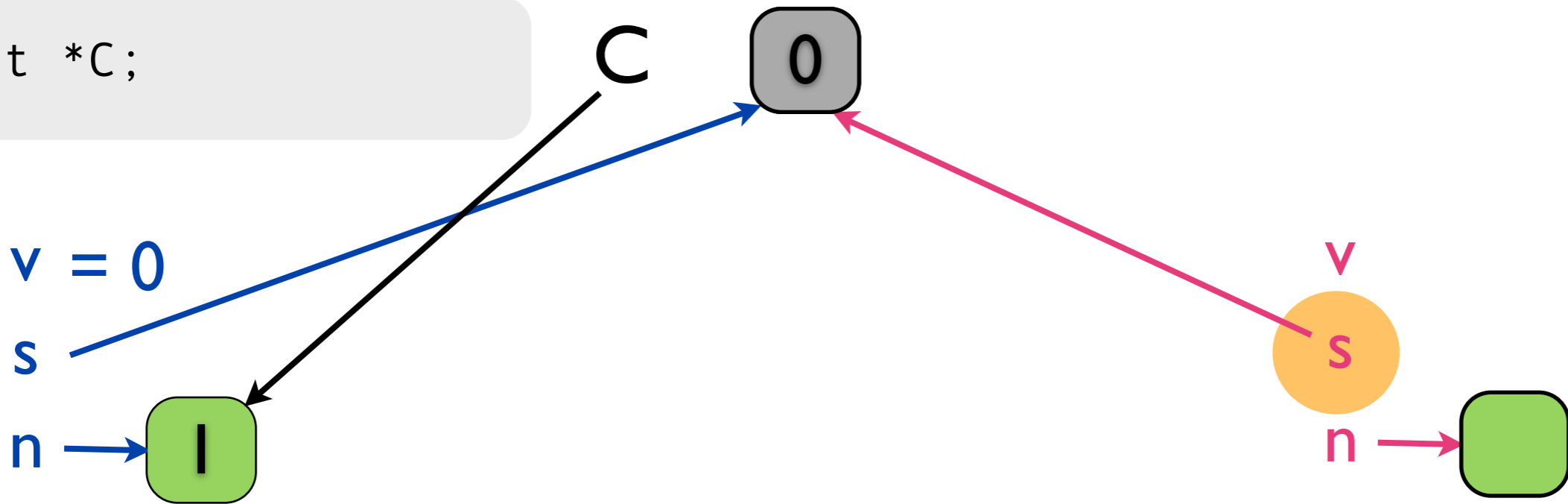
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



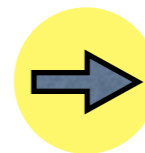
problem: memory safety

```
int *C;
```



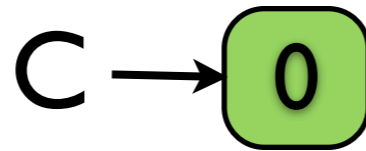
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



problem: correctness

```
int *C;
```

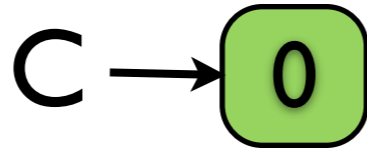


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```



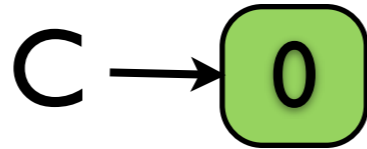
→

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness


```
int *C;
```



v

s

n

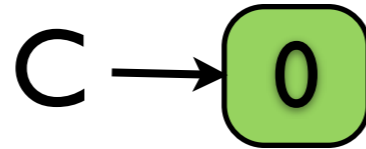


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```



v

s

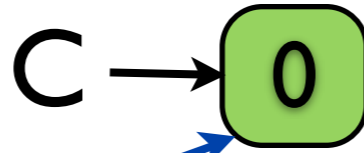
n

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness


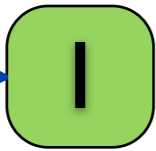
```
int *C;
```



v = 0

s

n

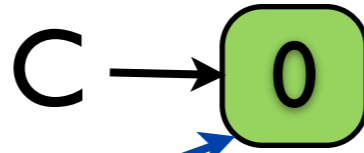


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

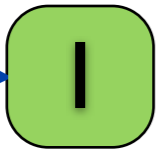
```
int *C;
```



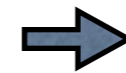
`v = 0`

`s`

`n`



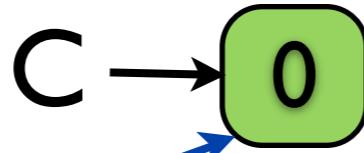
```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```


problem: correctness

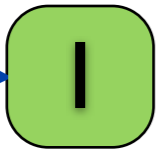
```
int *C;
```



`v = 0`

`s`

`n`



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

`v`

`s`

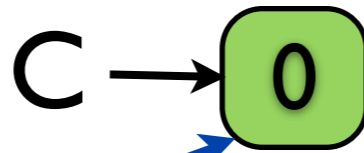
`n`



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

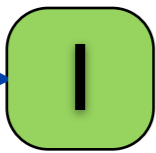
```
int *C;
```



v = 0

s

n



v

s

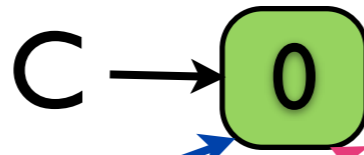
n

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

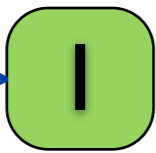
```
int *C;
```



v = 0

s

n

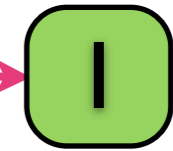


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

v = 0

s

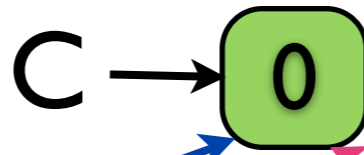
n



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

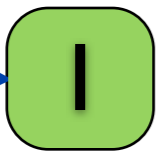
```
int *C;
```



v = 0

s

n

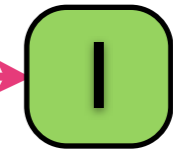


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

v = 0

s

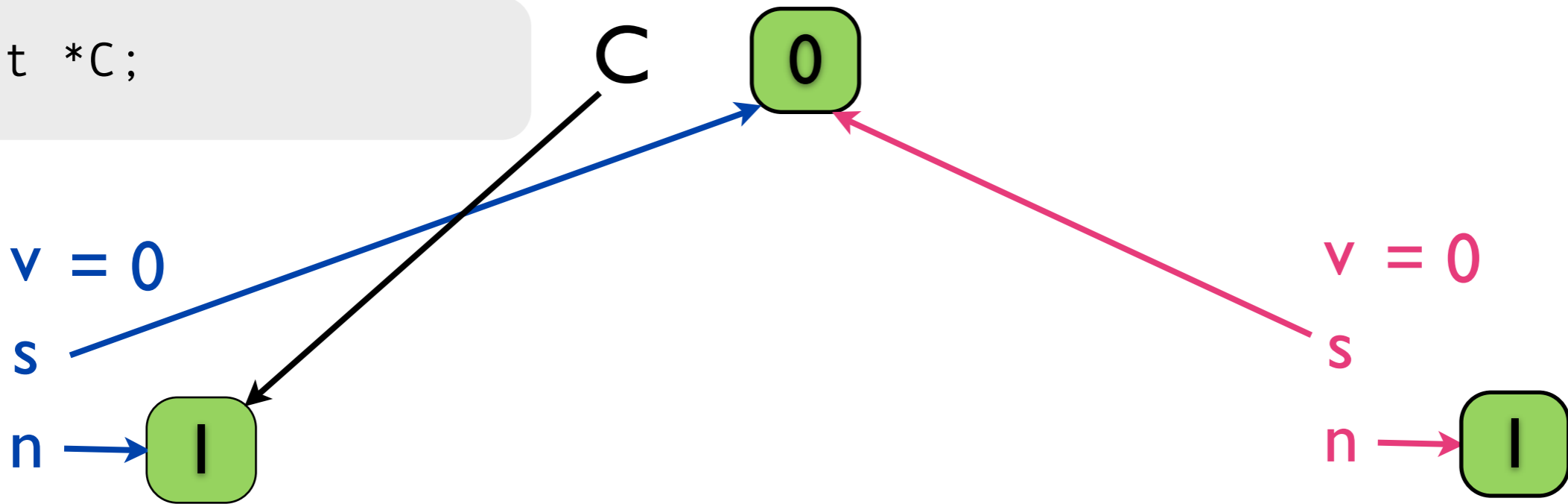
n



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

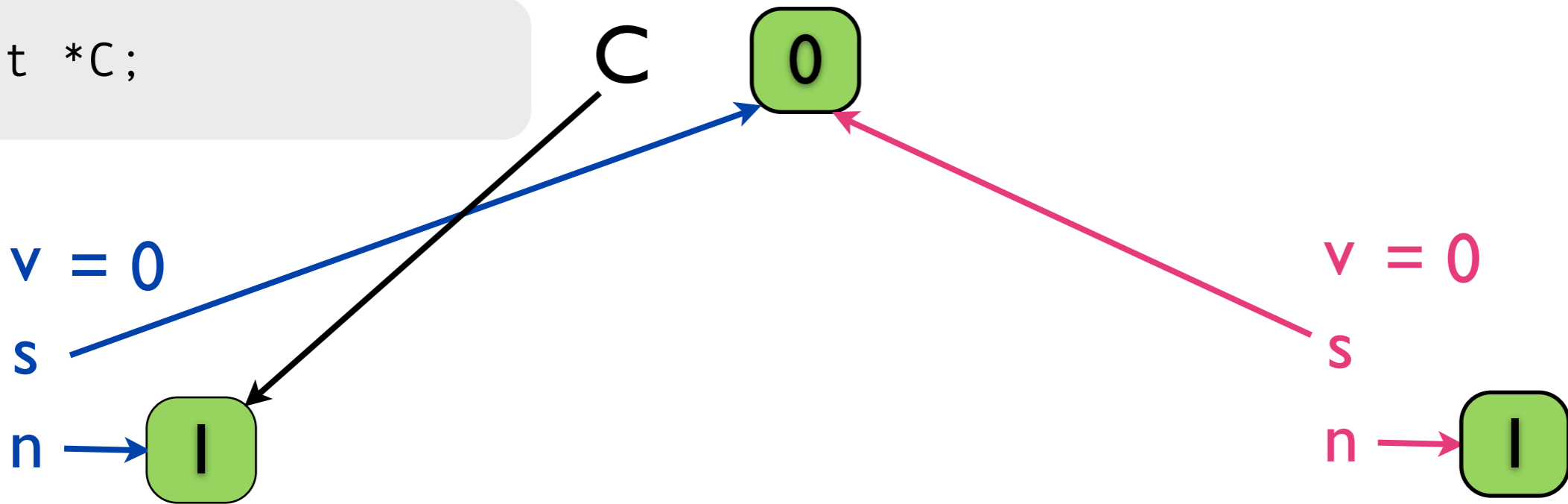


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

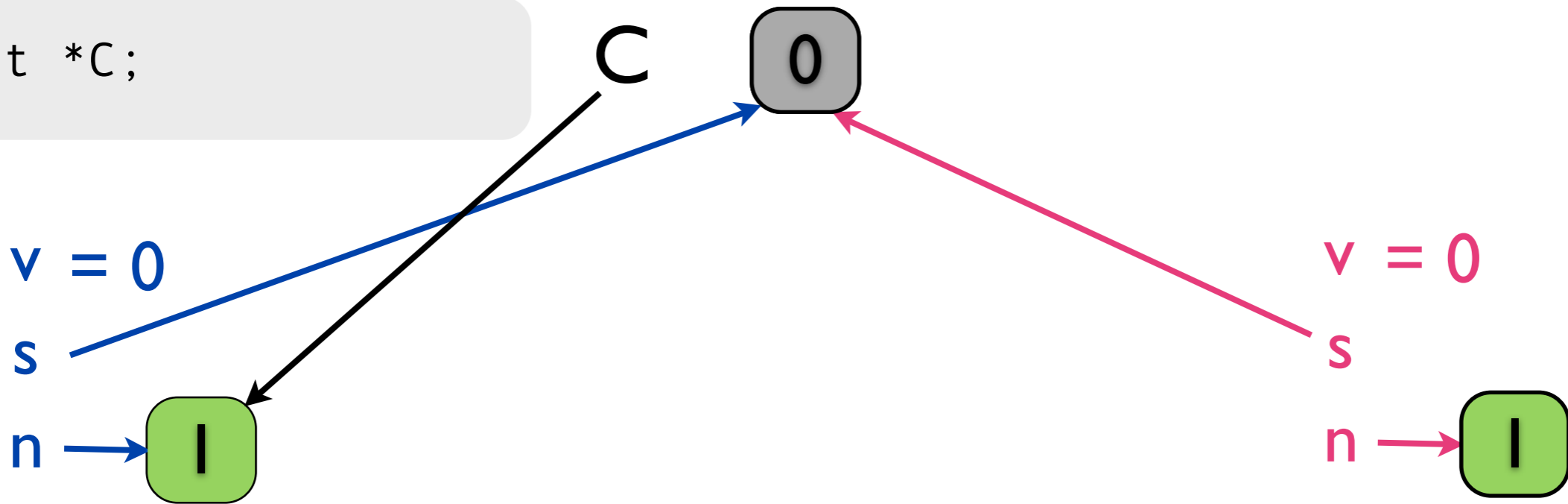


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

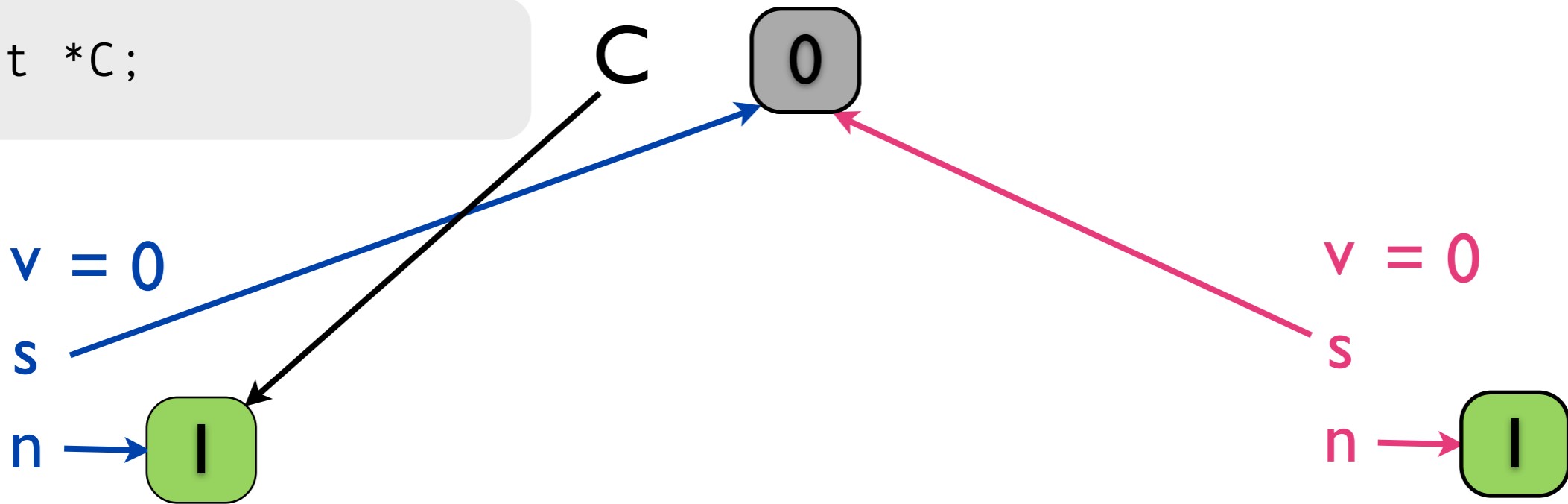


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



problem: correctness

```
int *C;
```

C

0

1

v = 0

s

n

1

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

C

0

1

v = 0

s

n

1

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

C

0

v

s

n

1

v = 0

s

n

1

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

C

0

v

s

n

1

v = 0

s

n

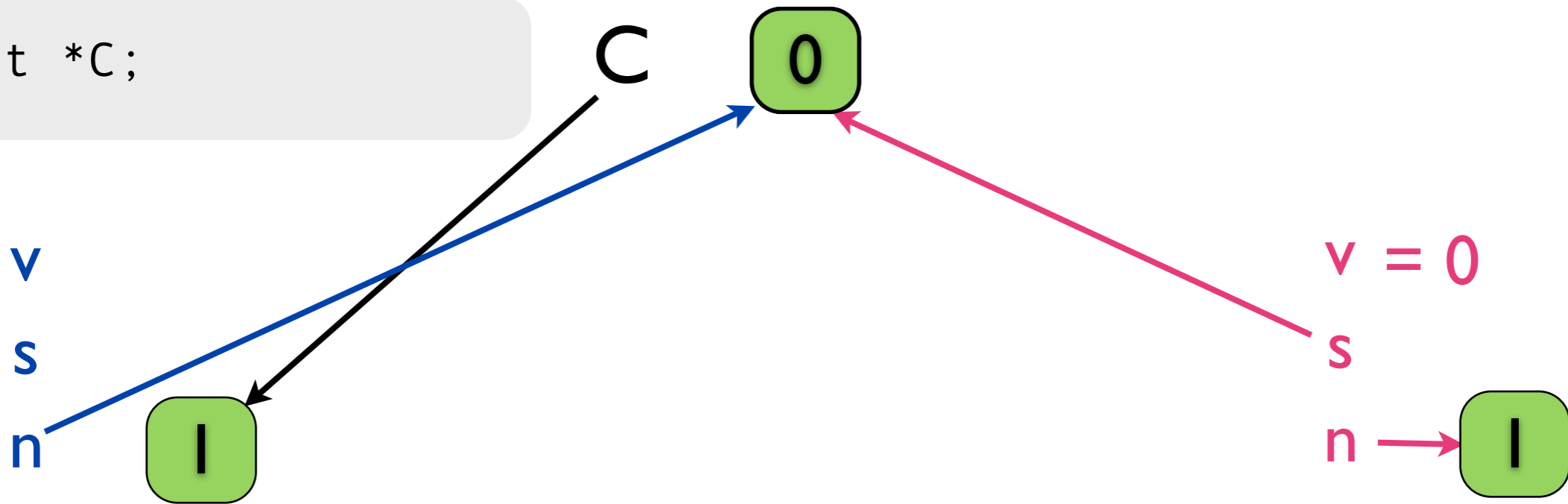
1

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

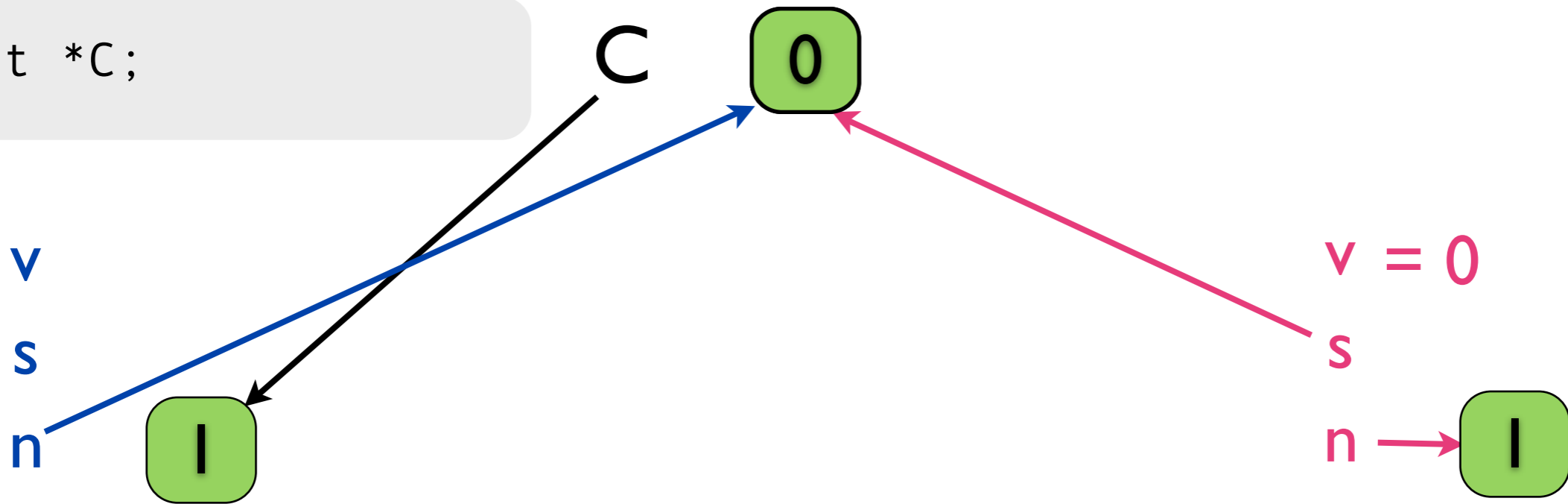


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

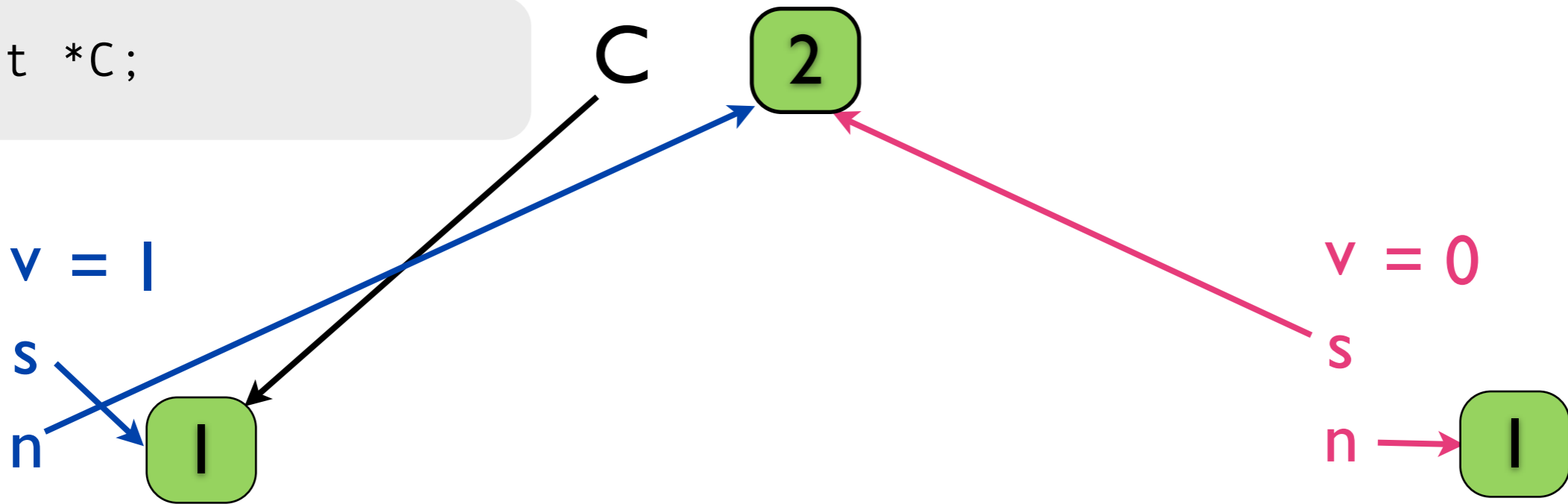


```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```



`v = 1`

`s`

`n`

`v = 0`

`s`

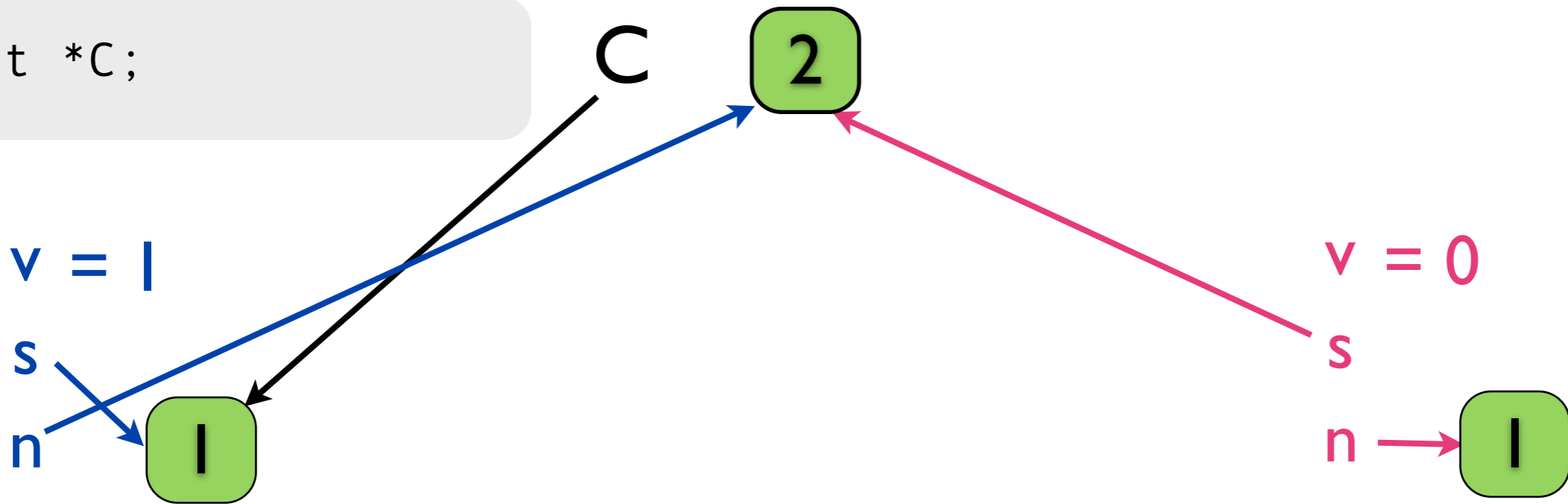
`n`

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```


problem: correctness

```
int *C;
```

C → 2

v = 1

s

n

1

v = 0

s

n

1

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

C → 2

v = 1

s

n

1

v = 0

s

n

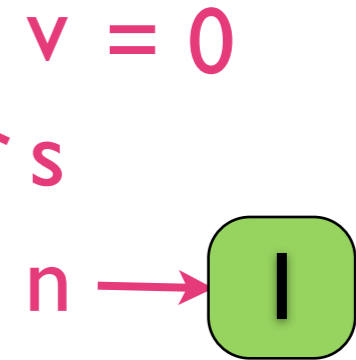
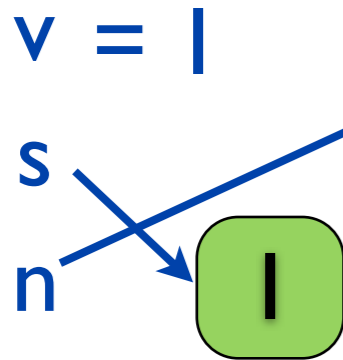
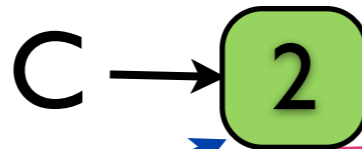
1

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  free(s);  
  return v;  
}
```

```
CAS(&C,s,n) :=  
<if(C==s)  
  C=n;  
  return true  
else  
  return false>
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C,s,n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

v = 1

s

n



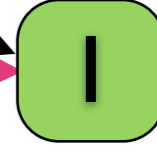
C

2

v = 0

s

n



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
CAS(&C, s, n) :=  
< if (C == s)  
    C = n;  
    return true  
else  
    return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

v = 1

s

n



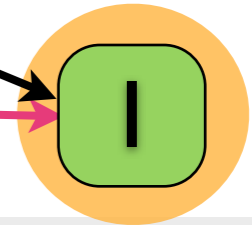
C

2

v = 0

s

n



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

```
CAS(&C, s, n) :=  
< if (C == s)  
  C = n;  
  return true  
else  
  return false >
```

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

problem: correctness

```
int *C;
```

v = 1

s

n



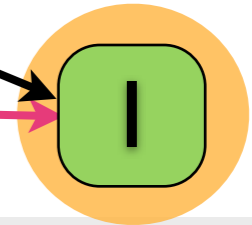
C

2

v = 0

s

n



```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

ABA
anomaly

```
int inc() {  
  int v, *s, *n;  
  n = new int;  
  do{  
    s = C;  
    v = *s;  
    *n = v + 1;  
  }while(!CAS(&C, s, n));  
  free(s);  
  return v;  
}
```

reclamation algorithms (paradigms)

- Hazard Pointers [Michael'02]
- RCU [McKenney+'98]
- Epoch [Fraser+'03]

main insight

- Hazard Pointers [Michael'02]
- RCU [McKenney+'98]
- Epoch [Fraser+'03]
- pattern of temporal synchronization invariant
 - grace periods
- same logical idiom for invariant preservation

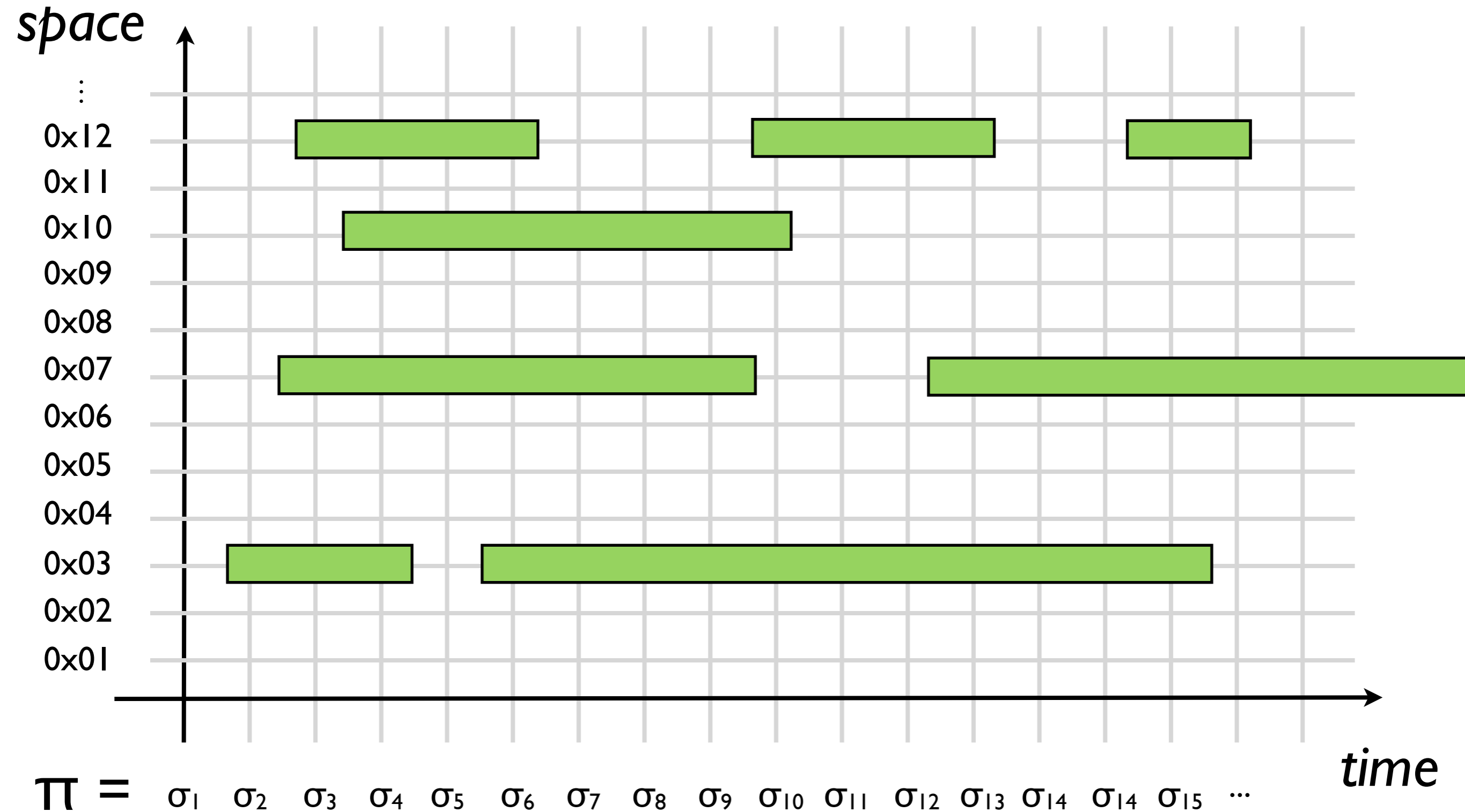
main contributions

- exposing algorithmic core
 - pattern for spatial-temporal synchronization
 - unified view based on **grace periods**
- pattern-based modular verification
 - past **temporal** separation logic (*since*)
 - uniform proofs of realistic algorithms

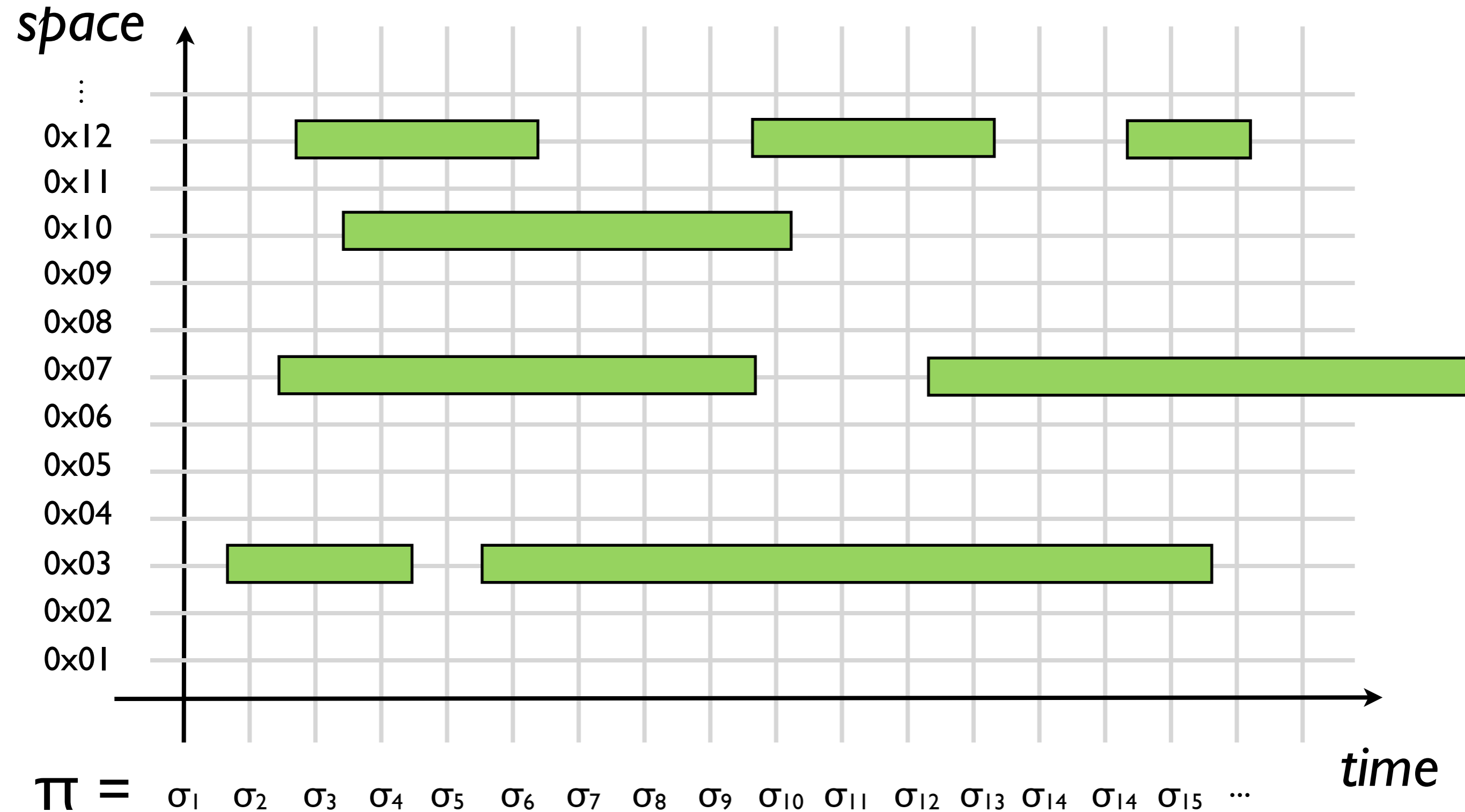
main contributions

- exposing algorithmic core
- pattern for spatial-temporal synchronization
- unified view based on **grace periods**
- pattern-based modular verification
 - past **temporal** separation logic (*since*)
 - uniform proofs of realistic algorithms

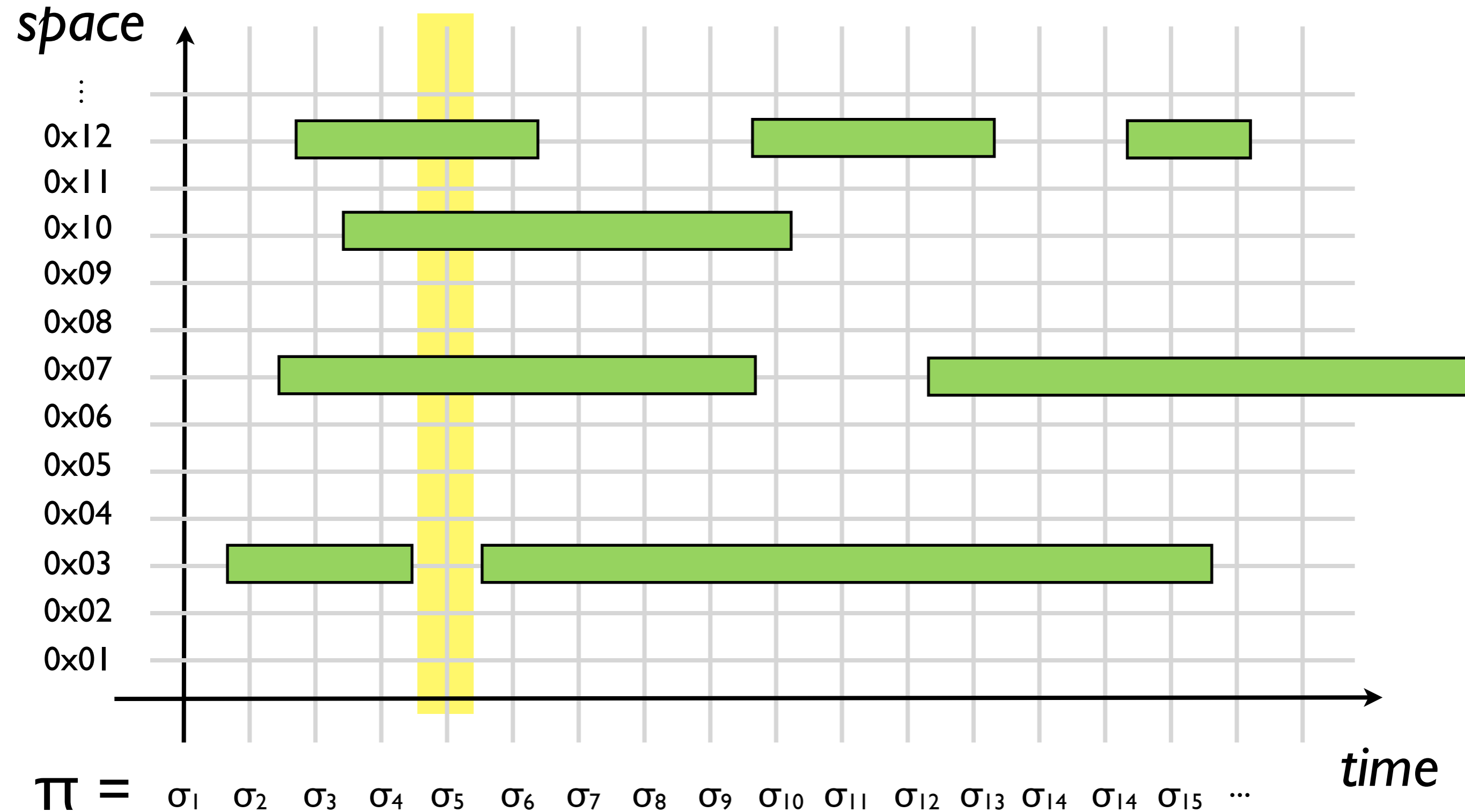
dual views: space / time



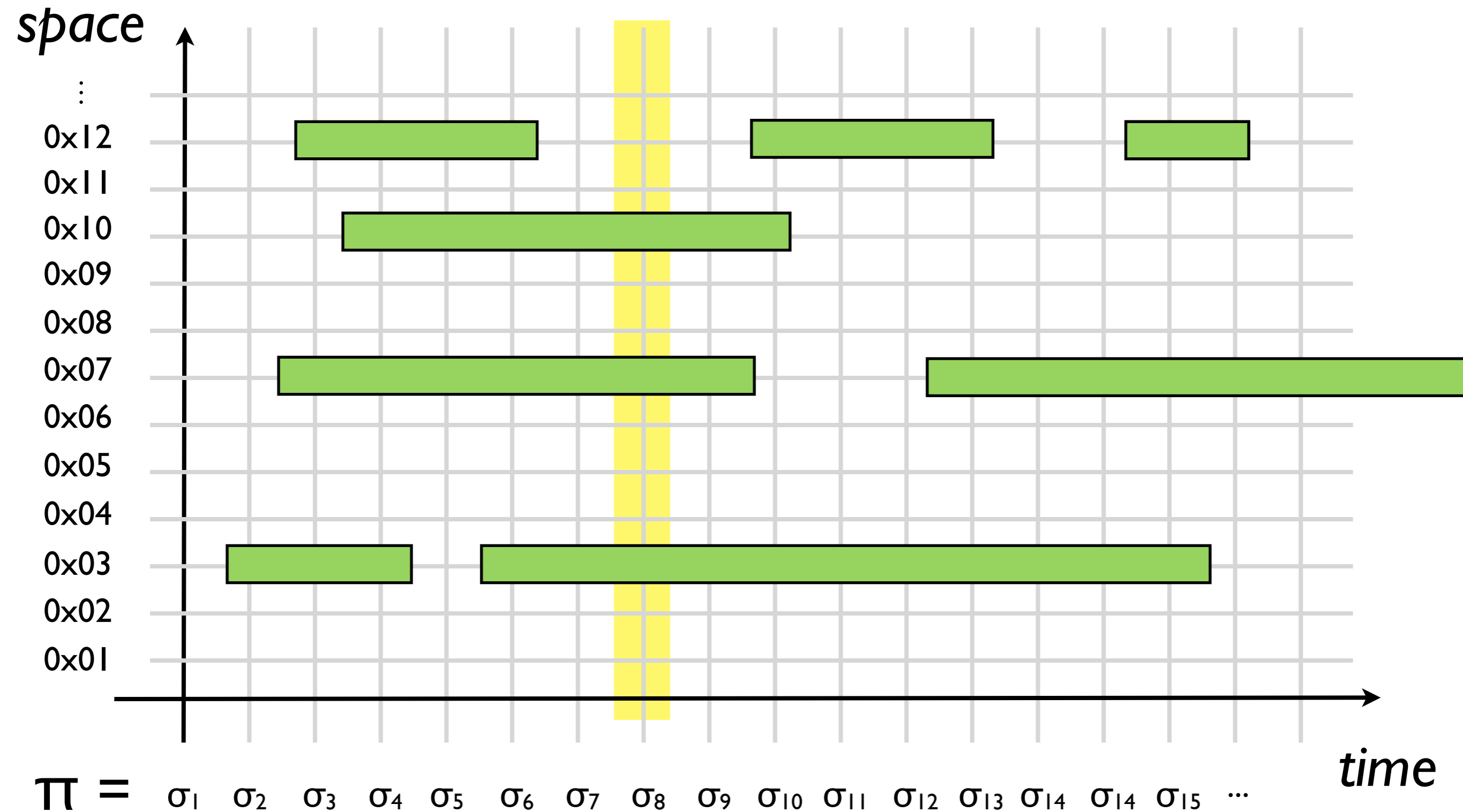
dual views: *space* / *time*



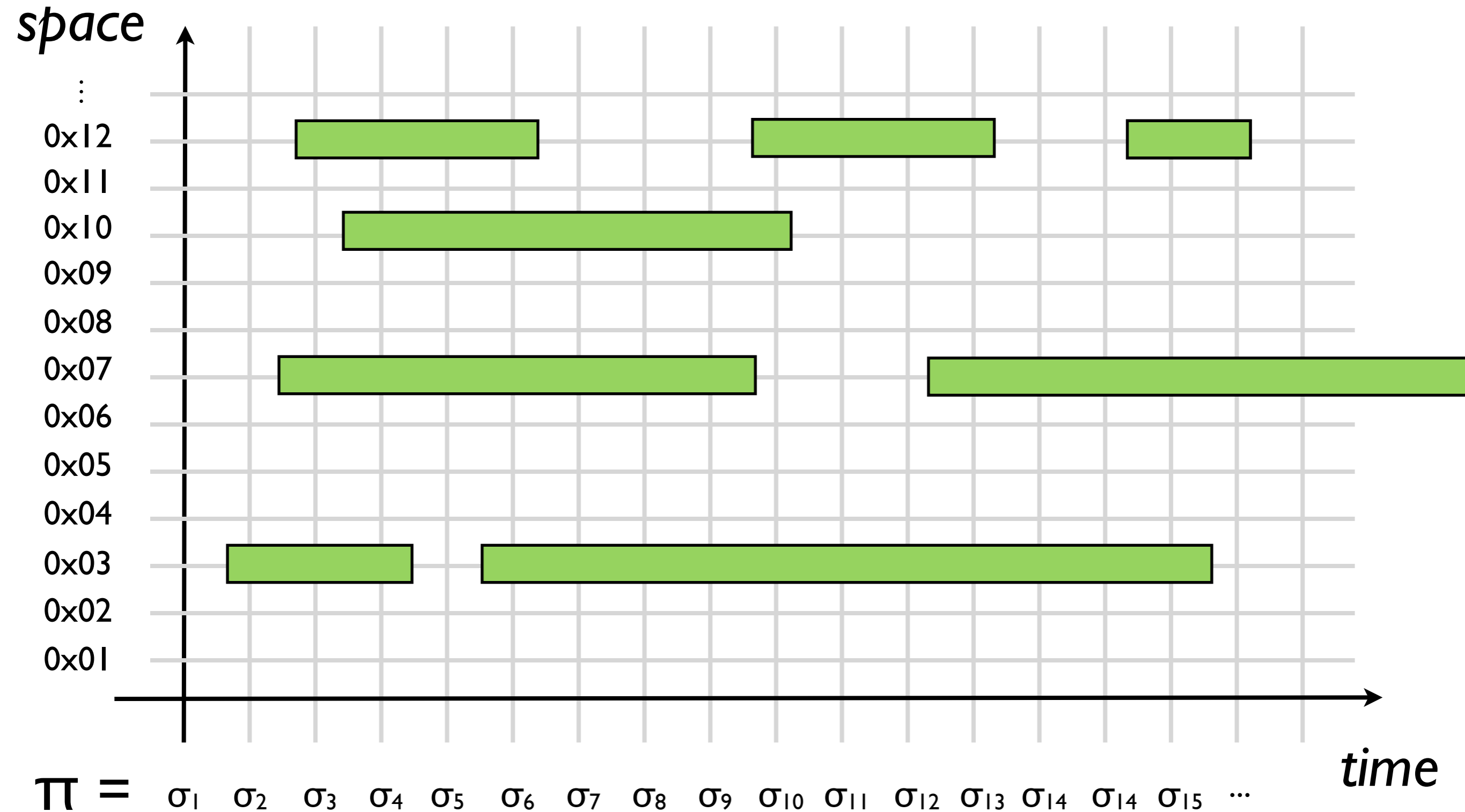
dual views: *space* / *time*



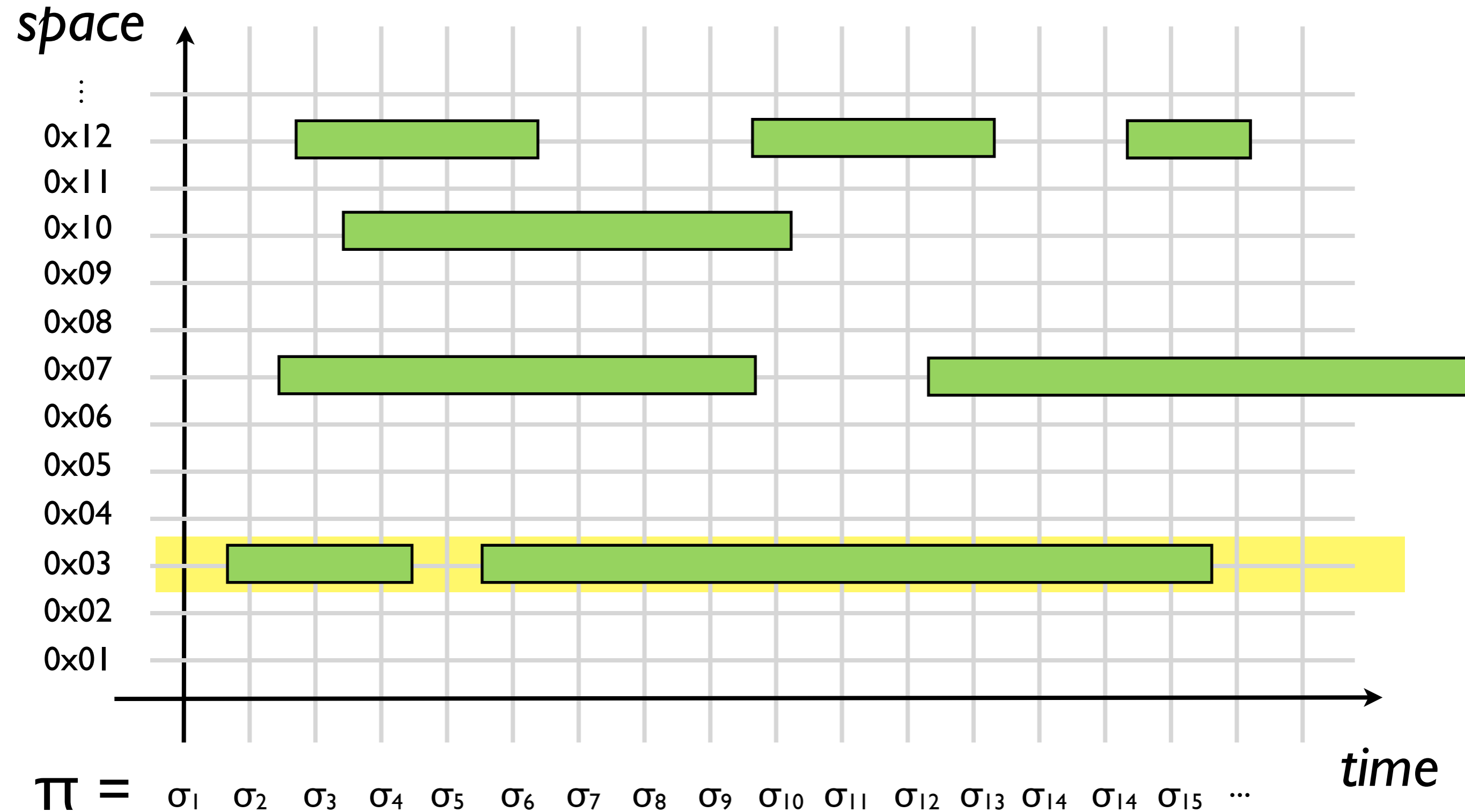
dual views: *space* / *time*



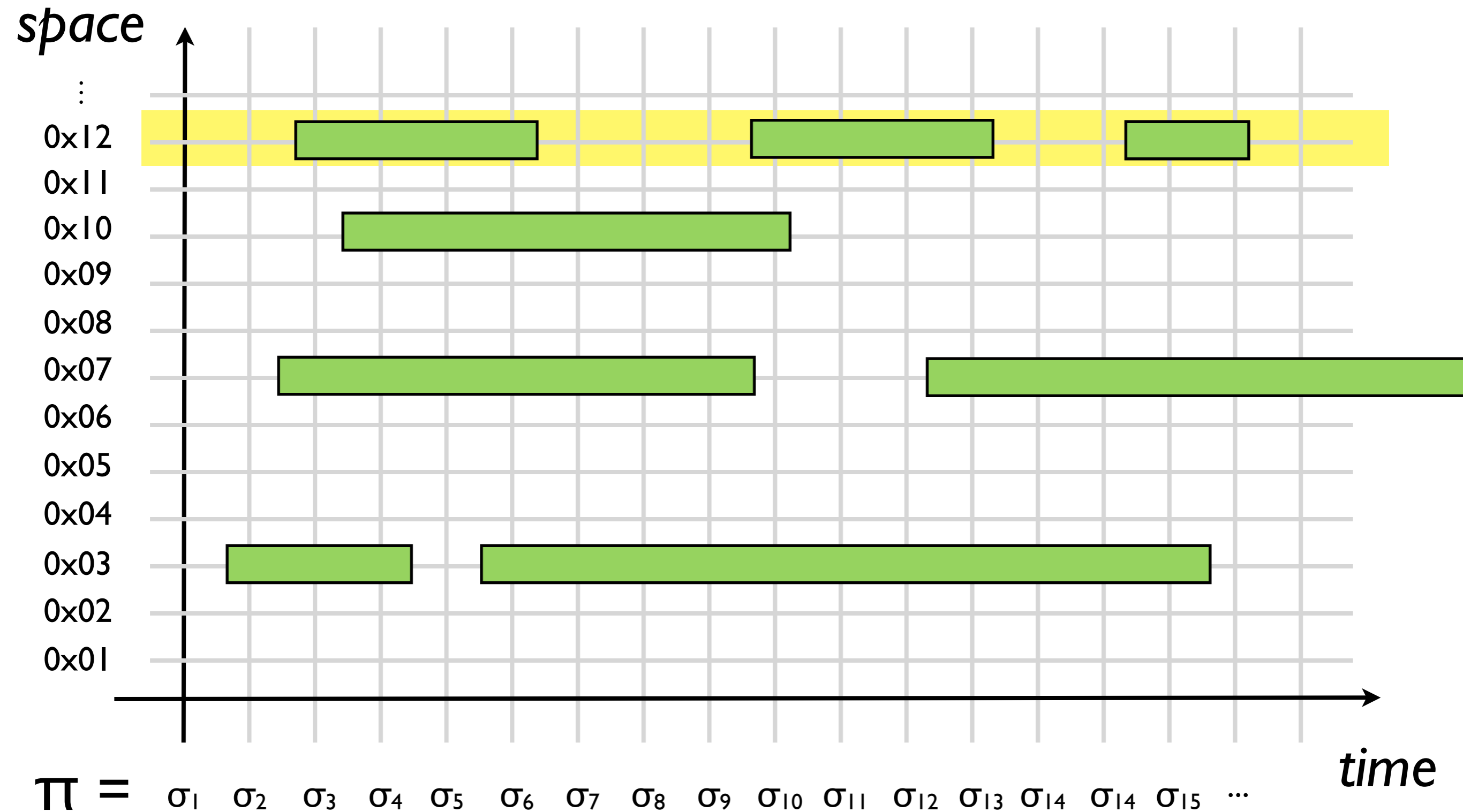
dual views: space / time



dual views: space / time

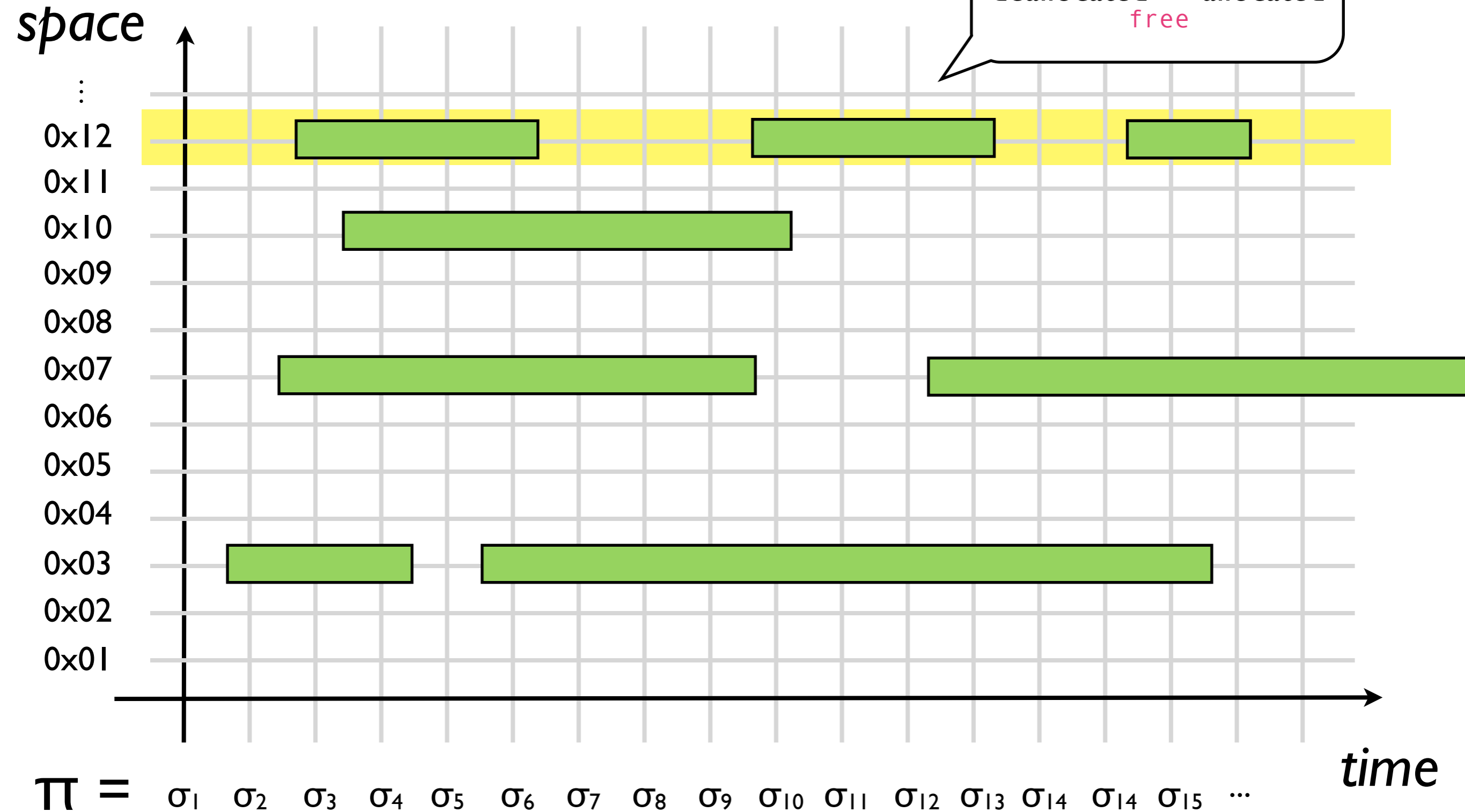


dual views: space / time

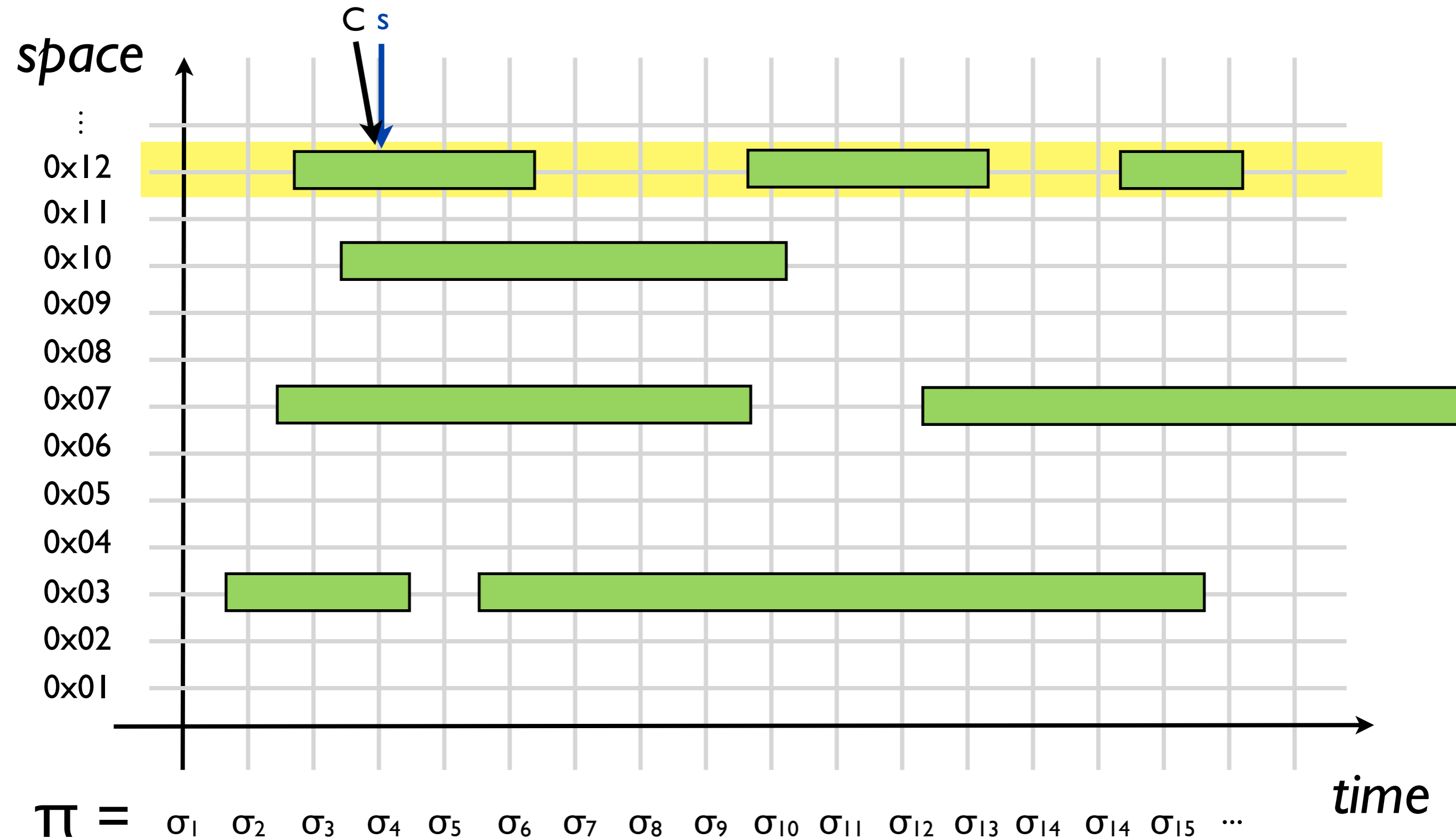


dual views: space / time

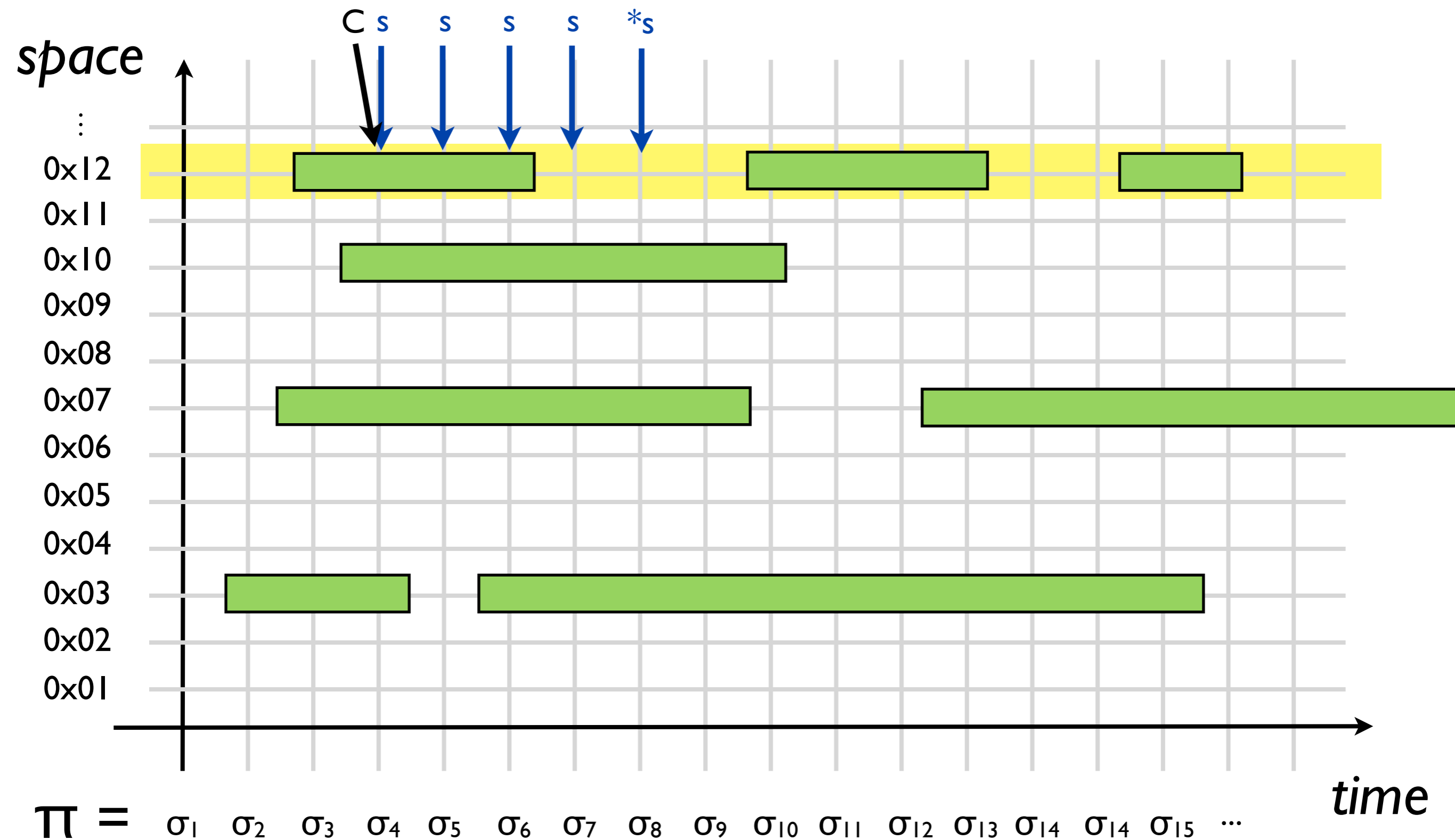
malloc
deallocated \Leftrightarrow allocated
free



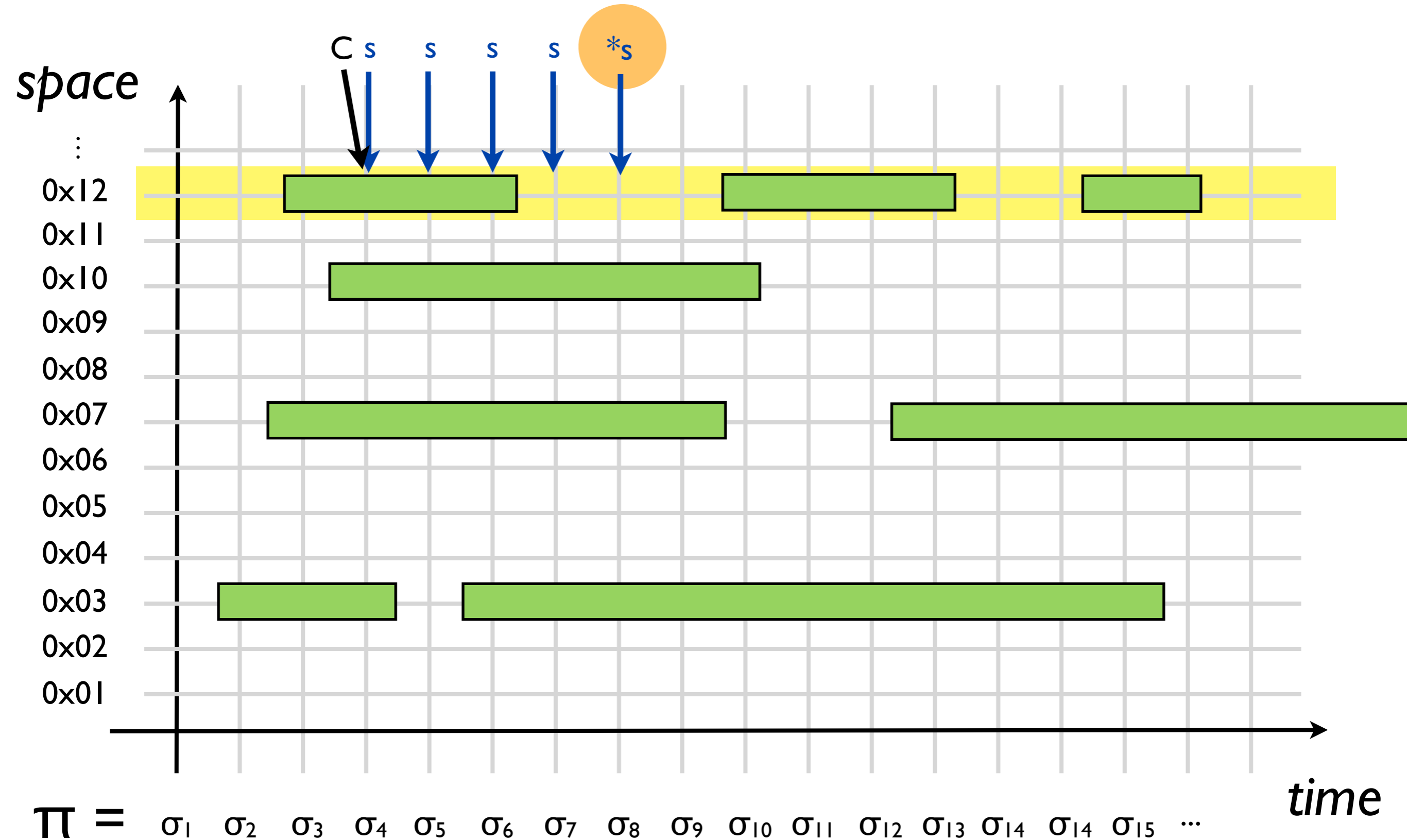
memory safety (temporal view)



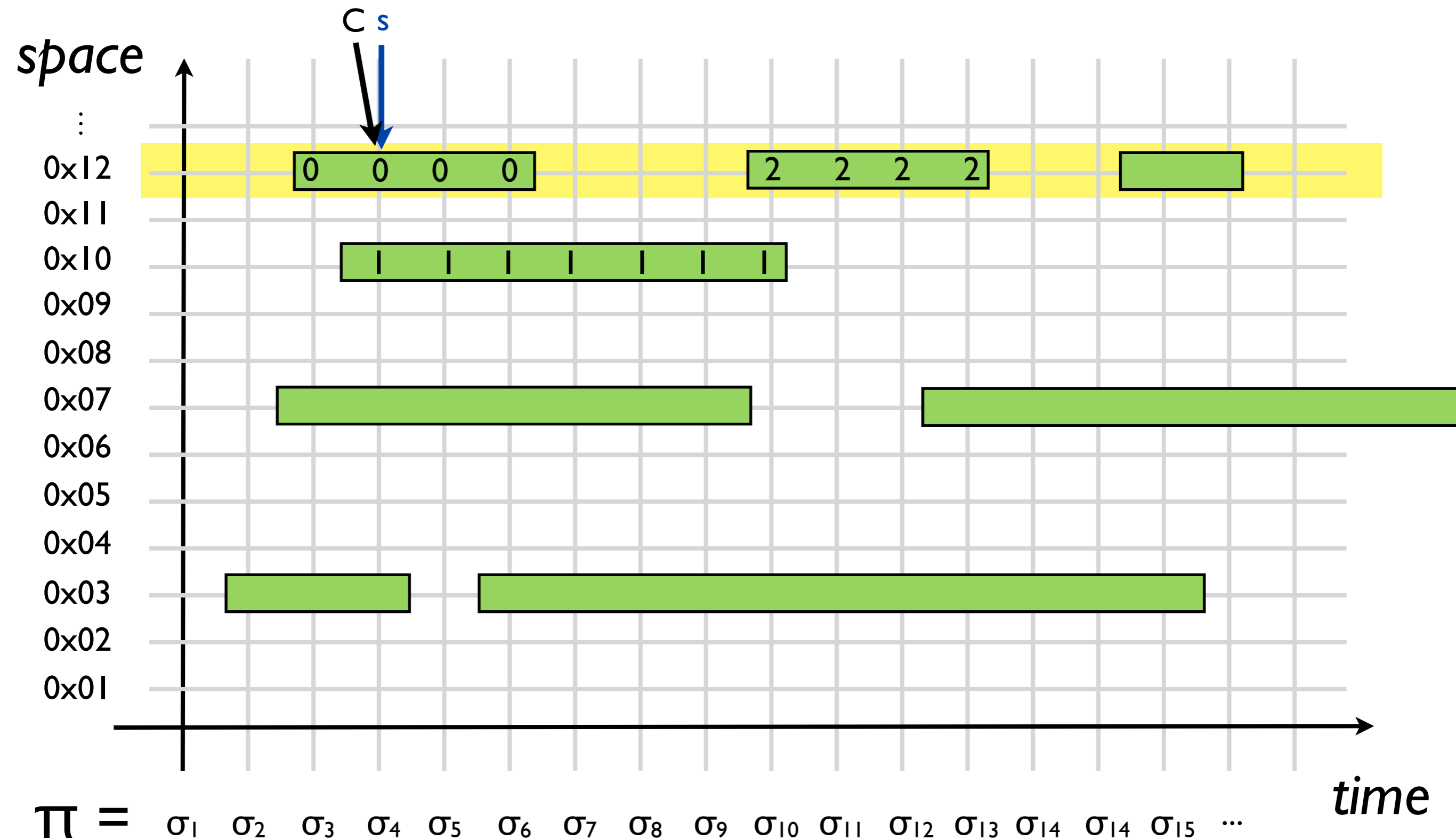
memory safety (temporal view)



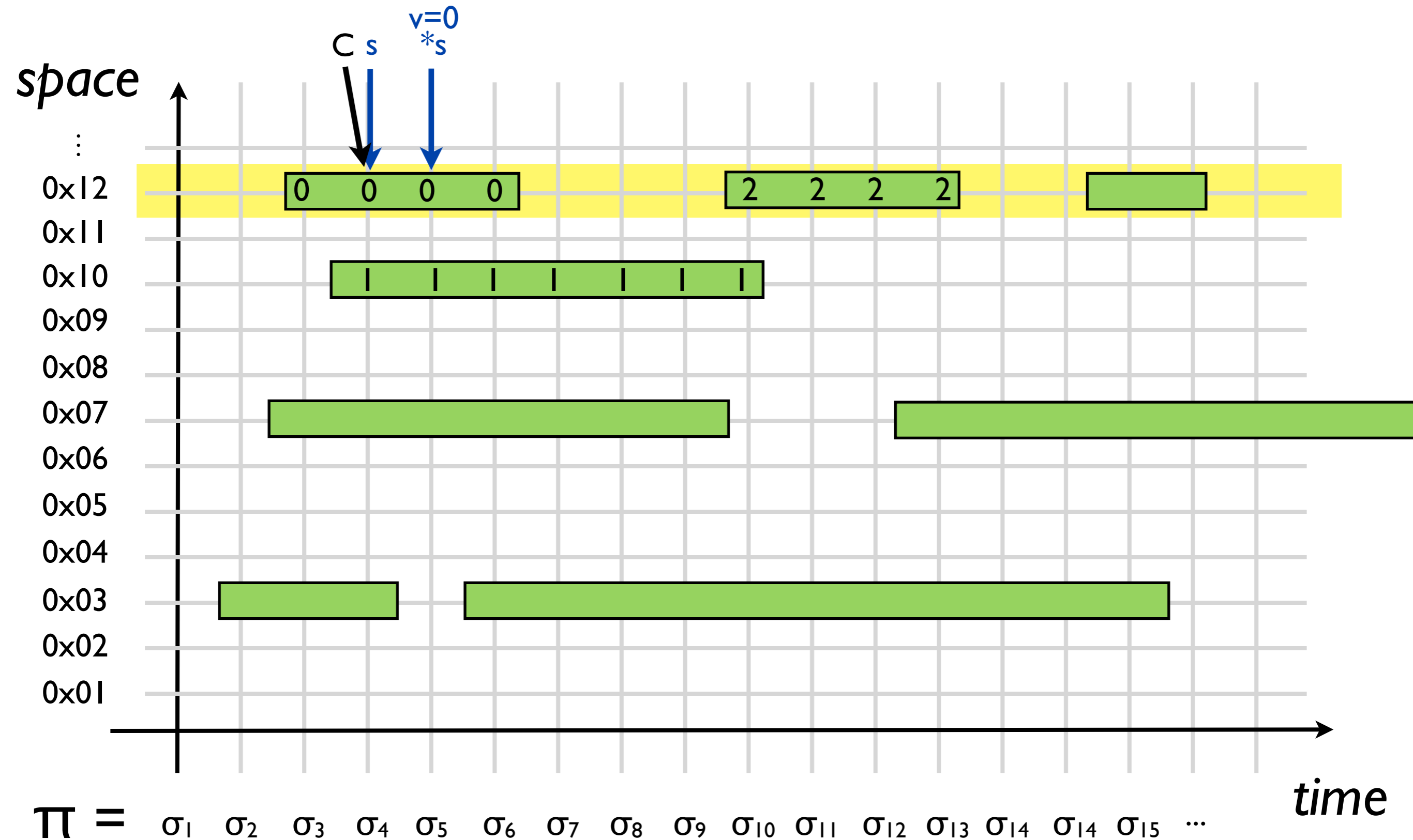
memory safety (temporal view)



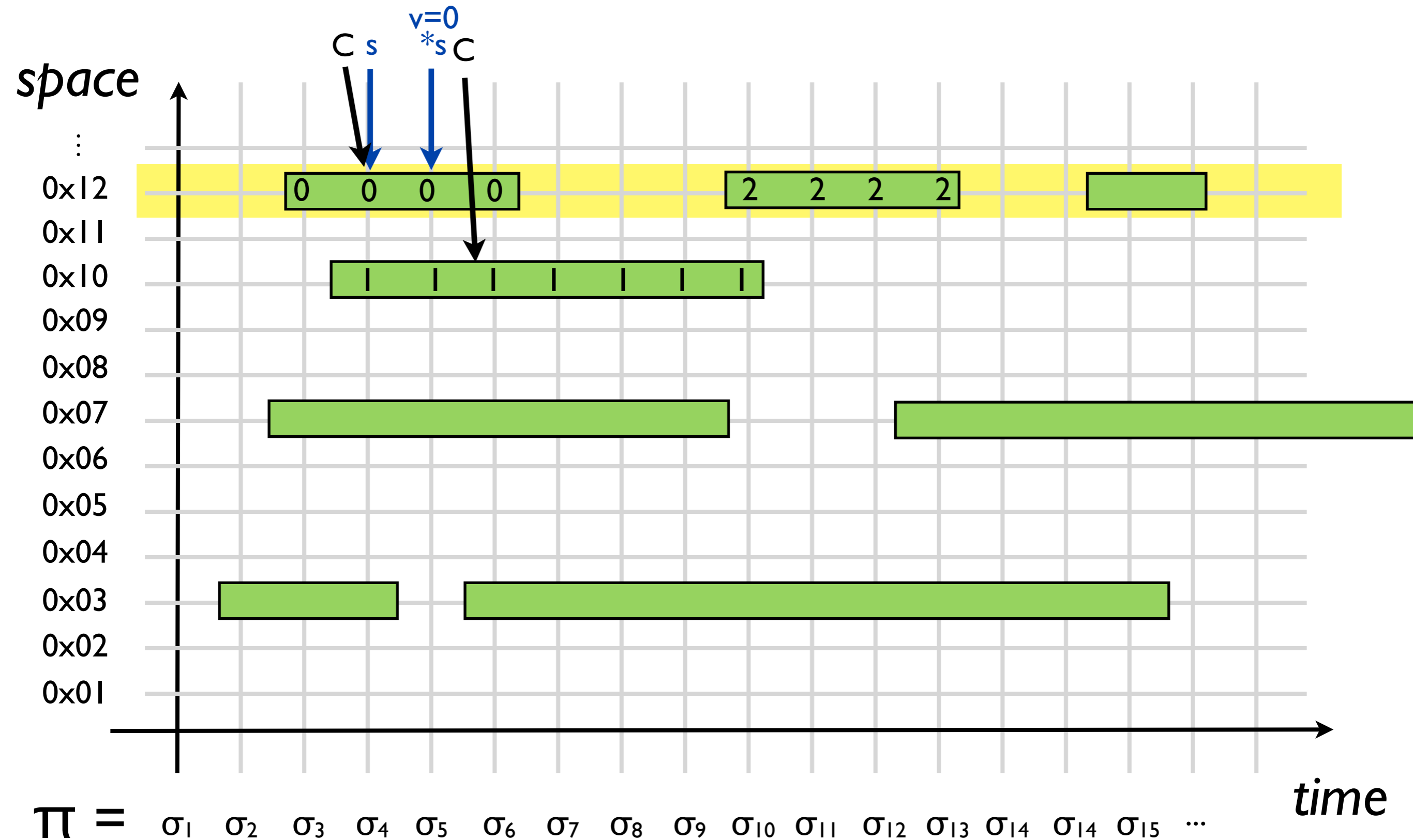
ABA (temporal view)



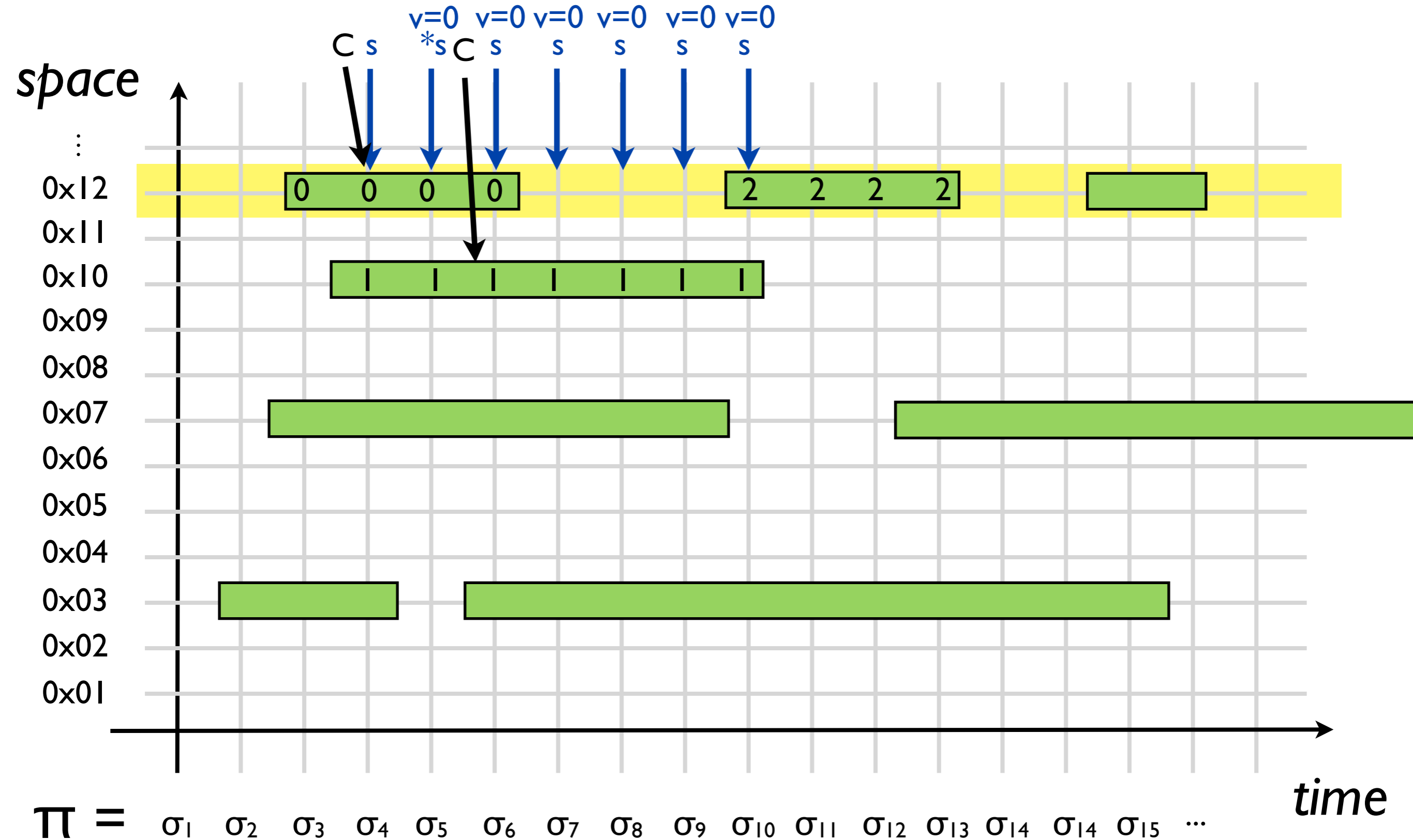
ABA (temporal view)



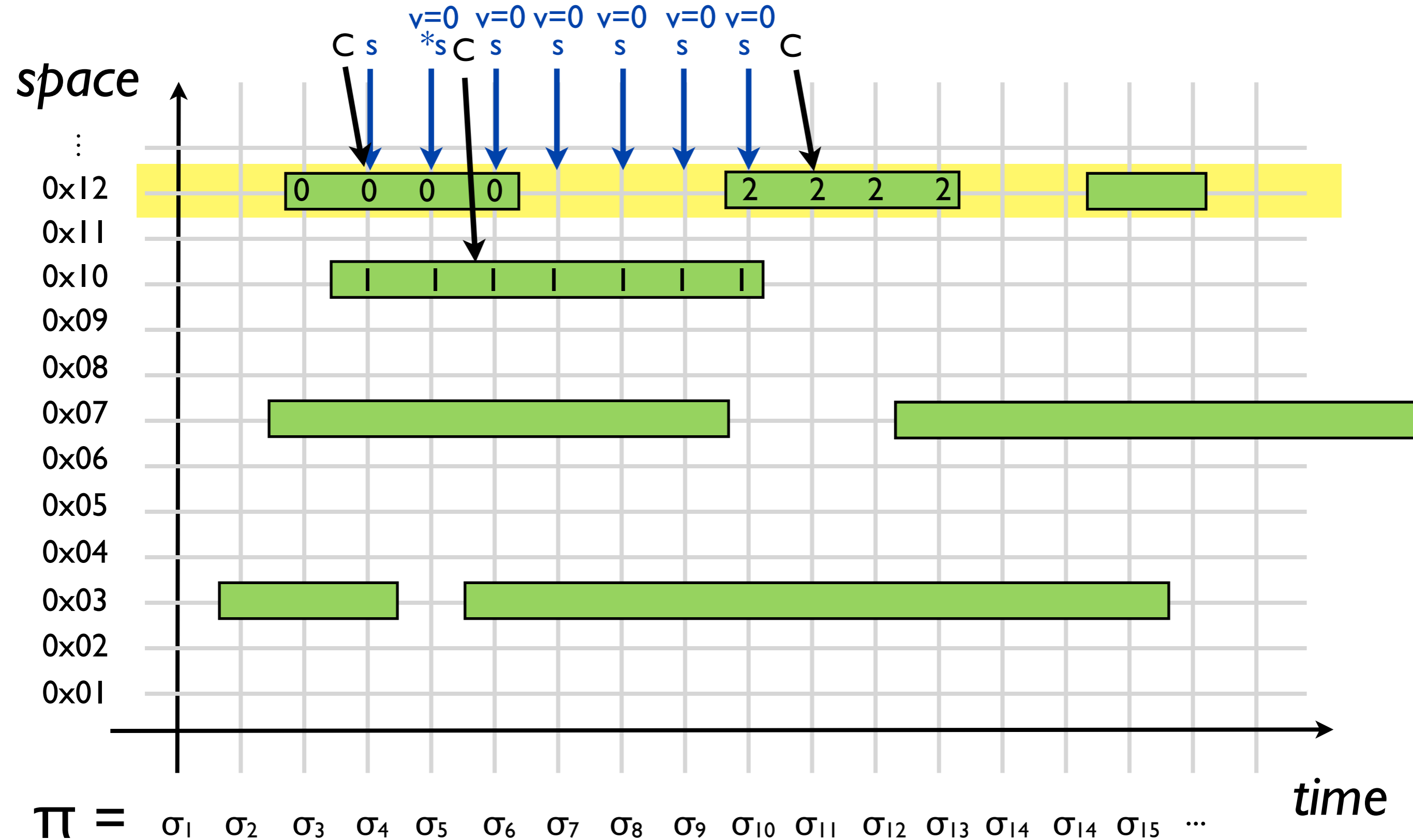
ABA (temporal view)



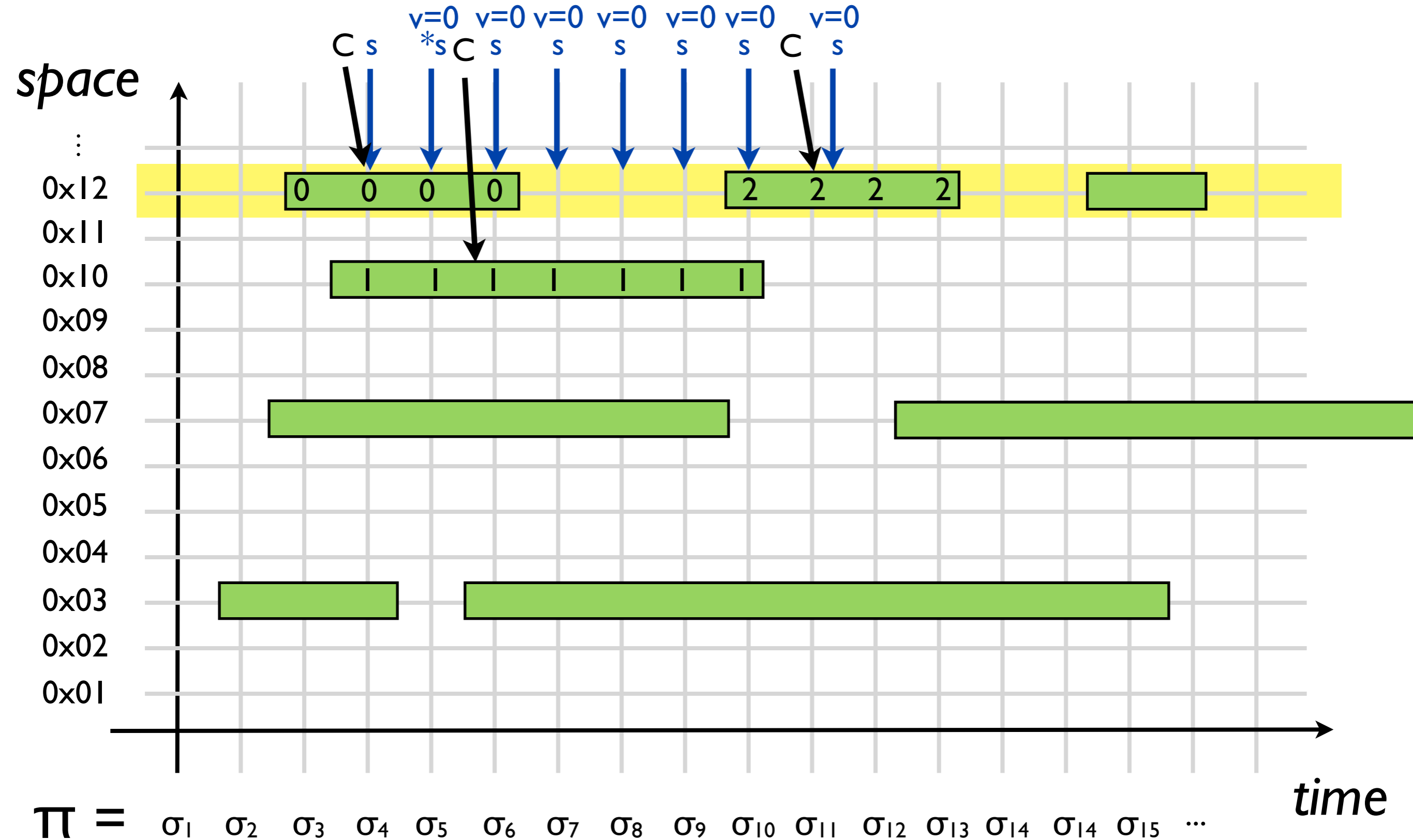
ABA (temporal view)



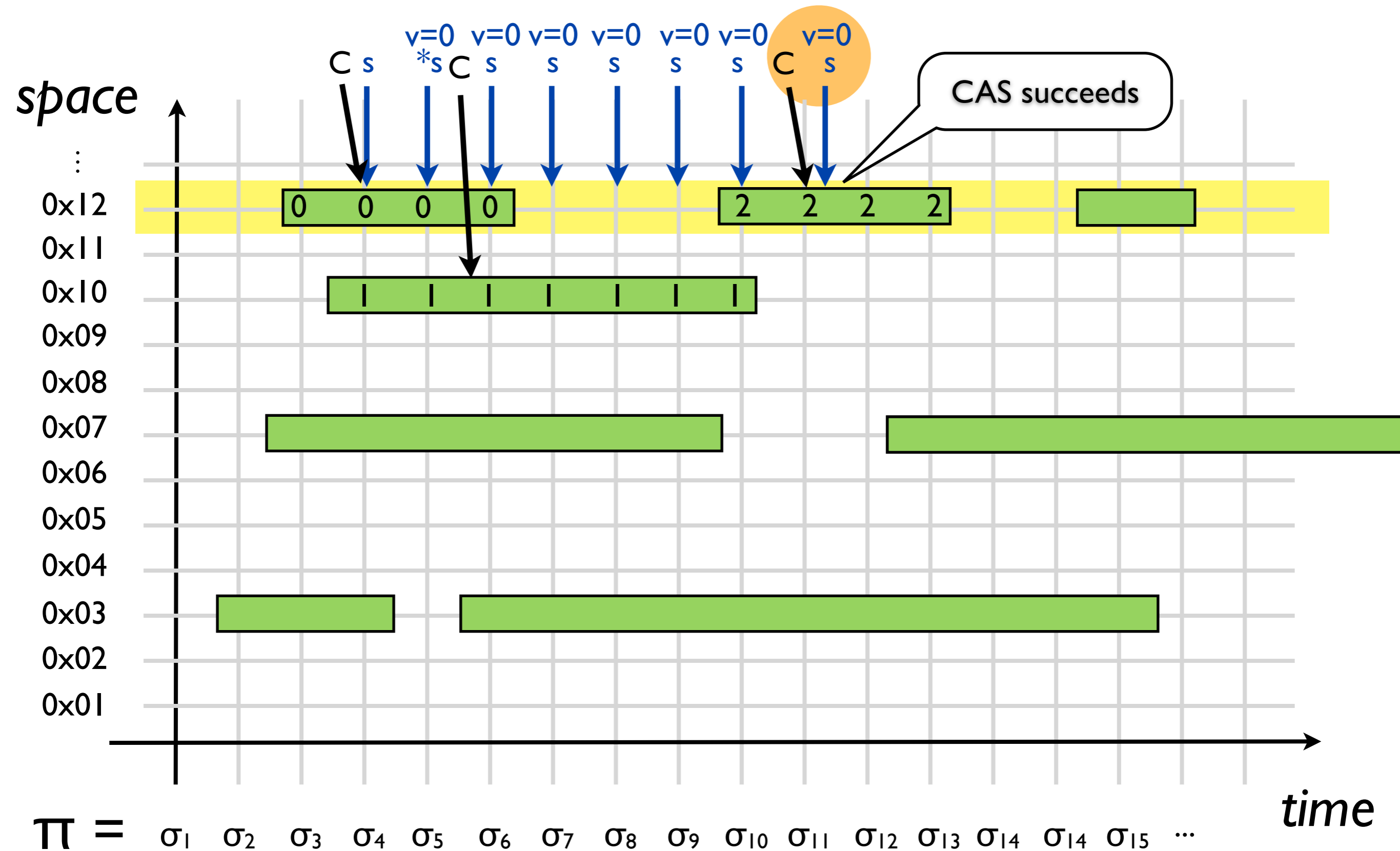
ABA (temporal view)



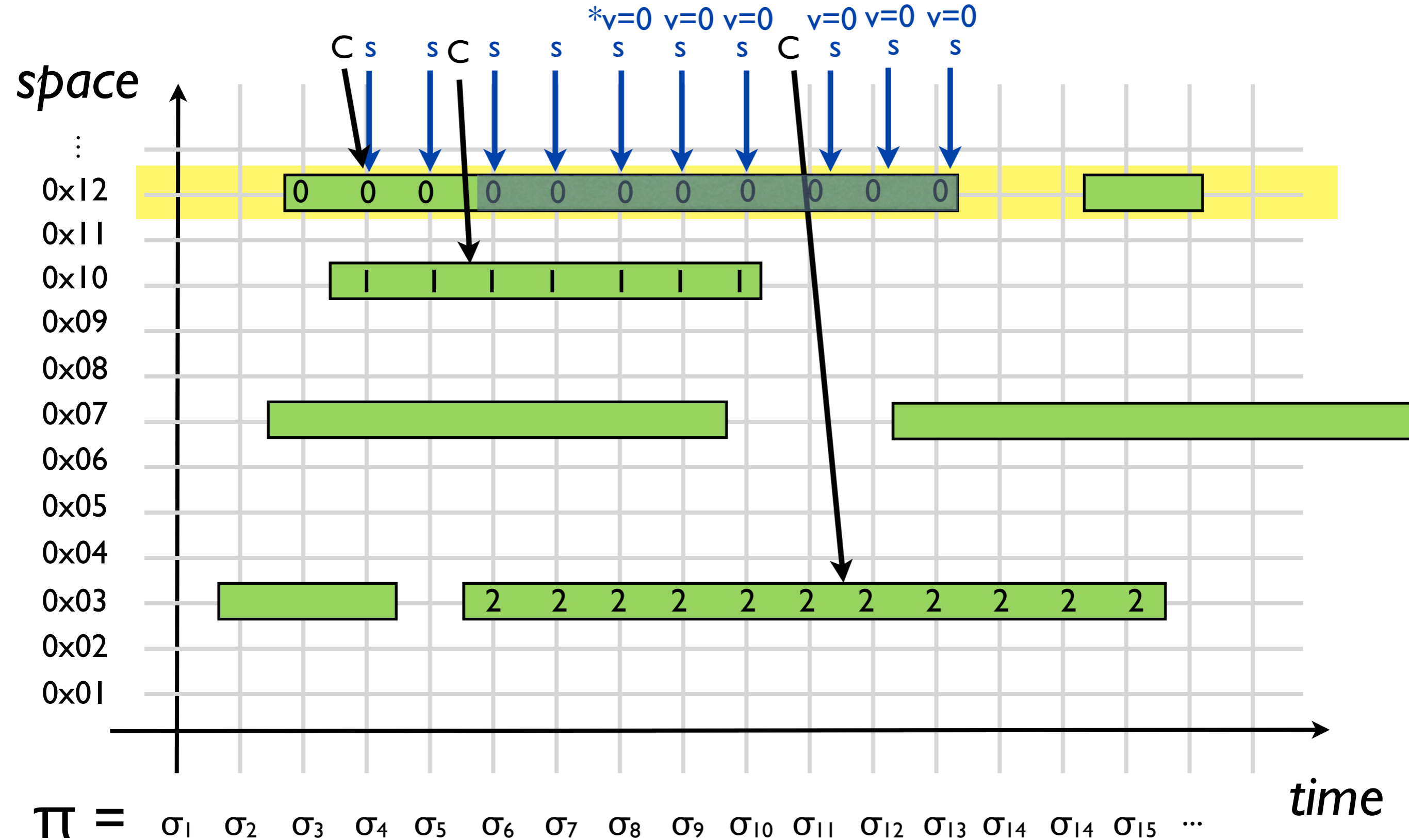
ABA (temporal view)



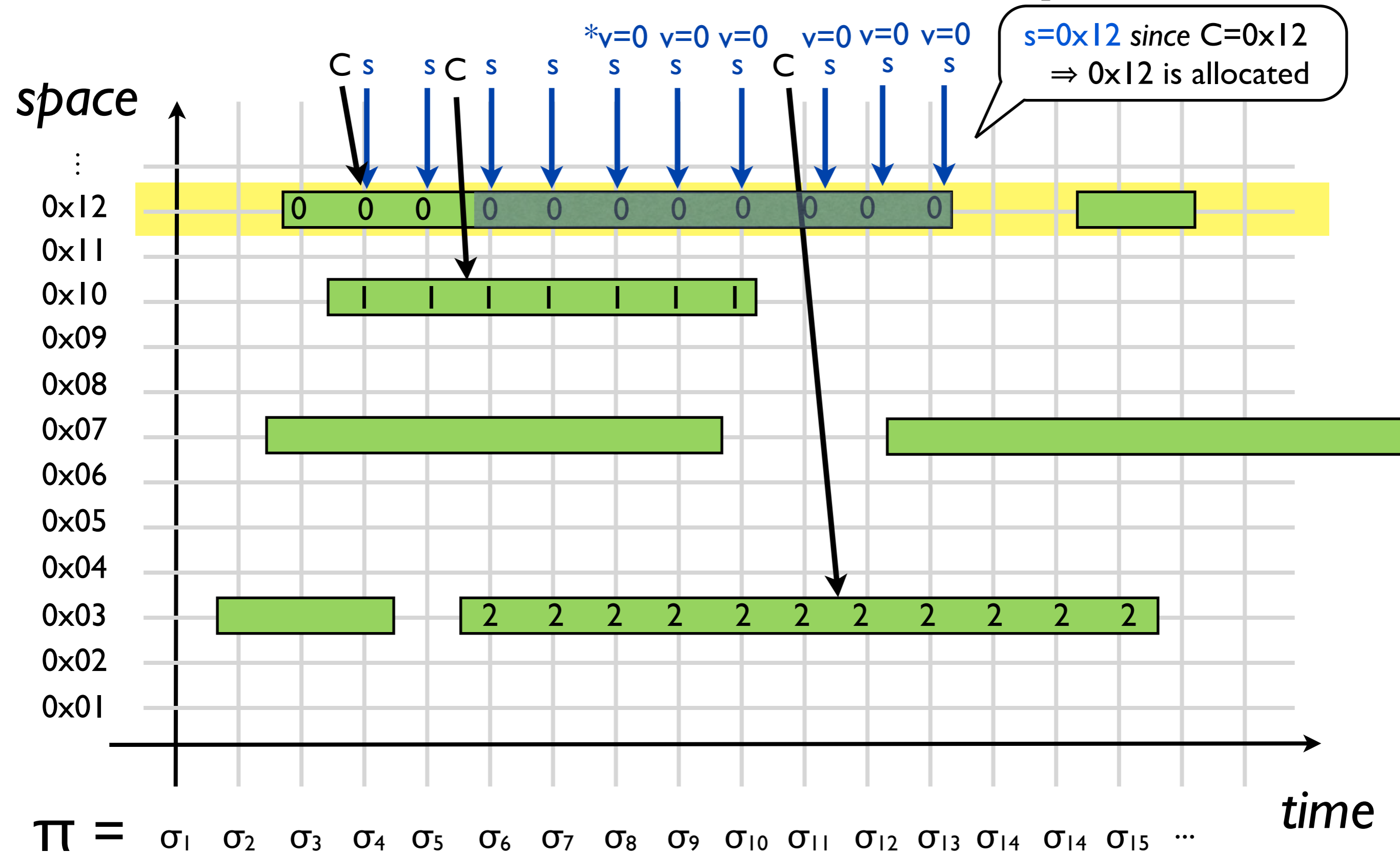
ABA (temporal view)



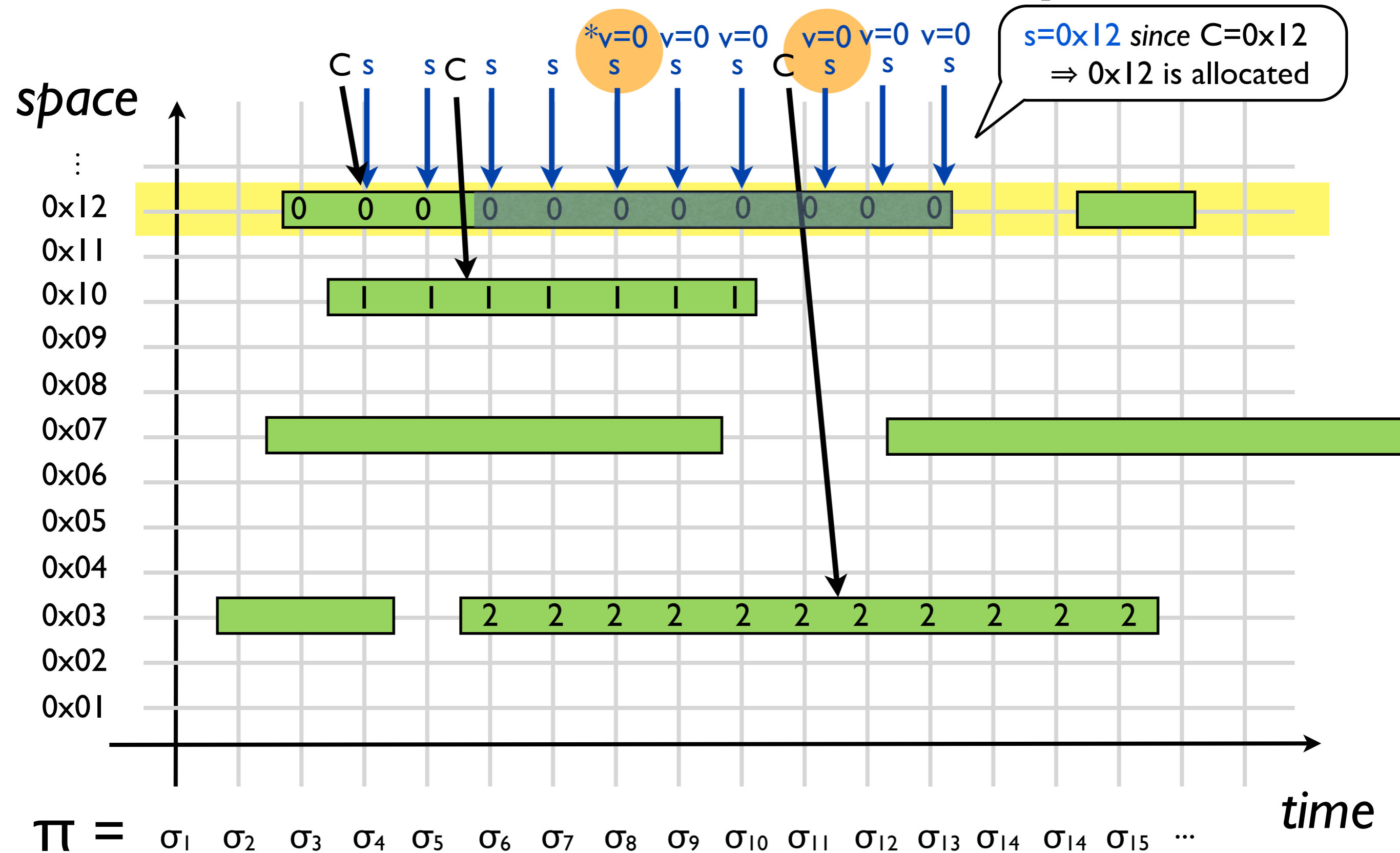
what we want, ideally



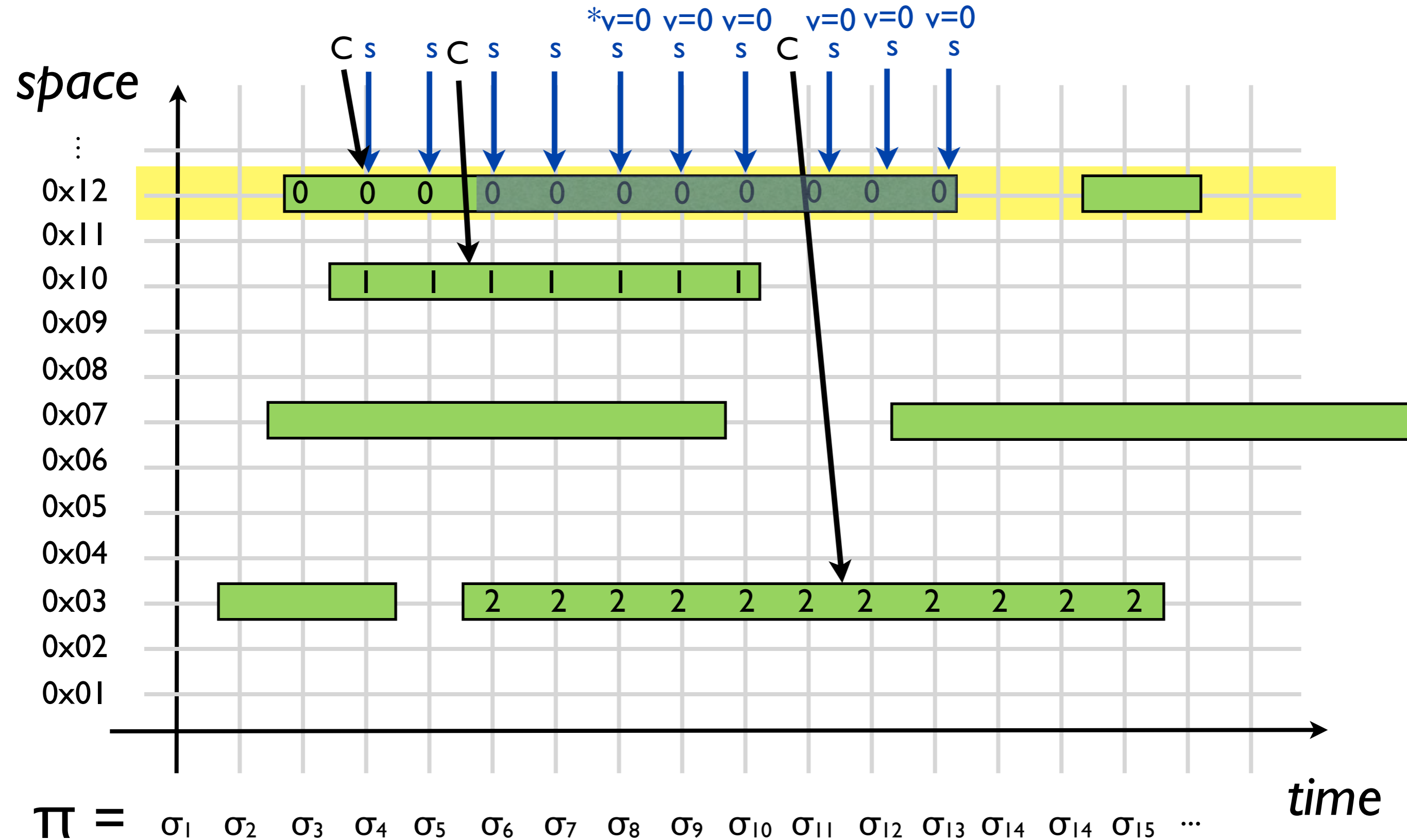
what we want, ideally



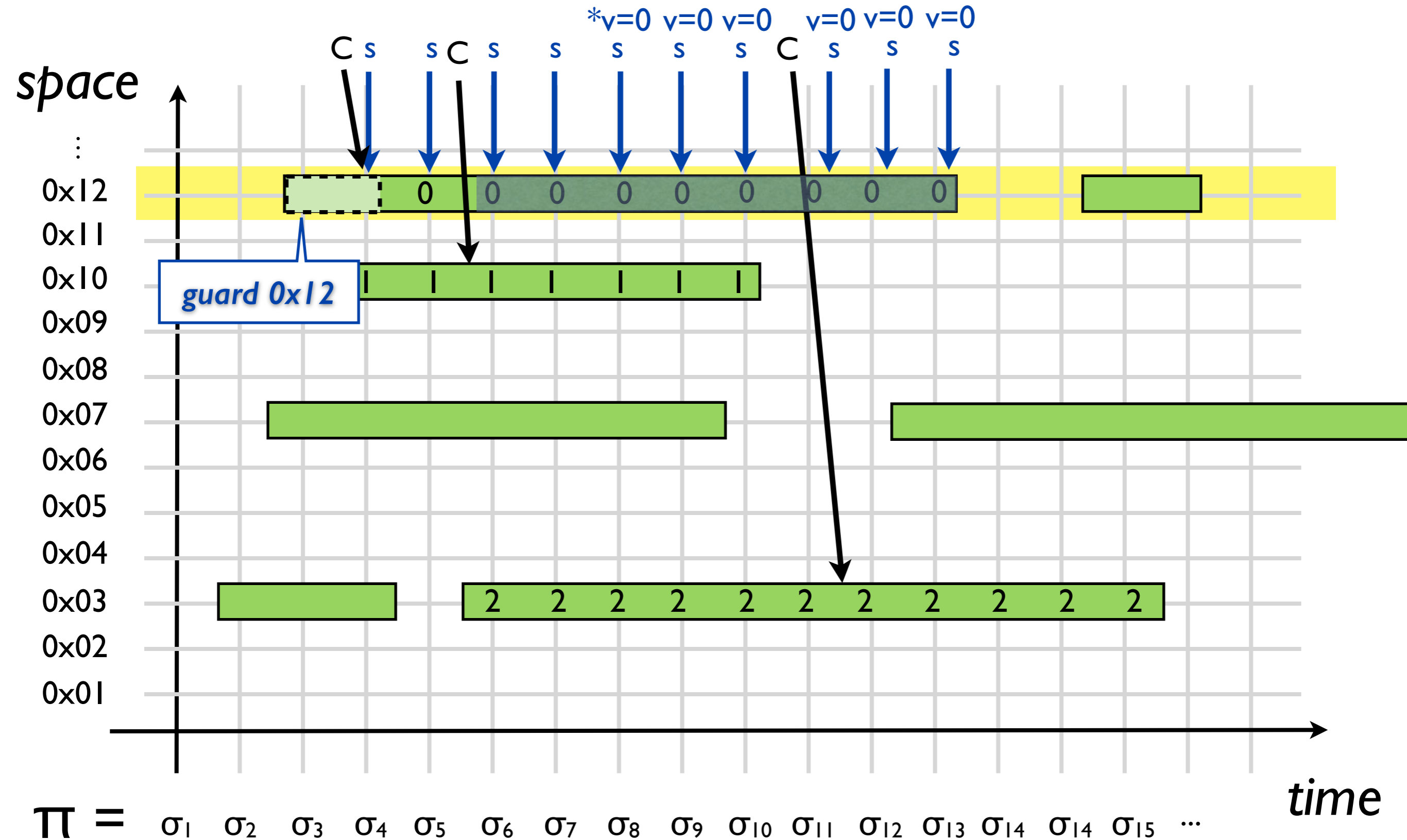
what we want, ideally



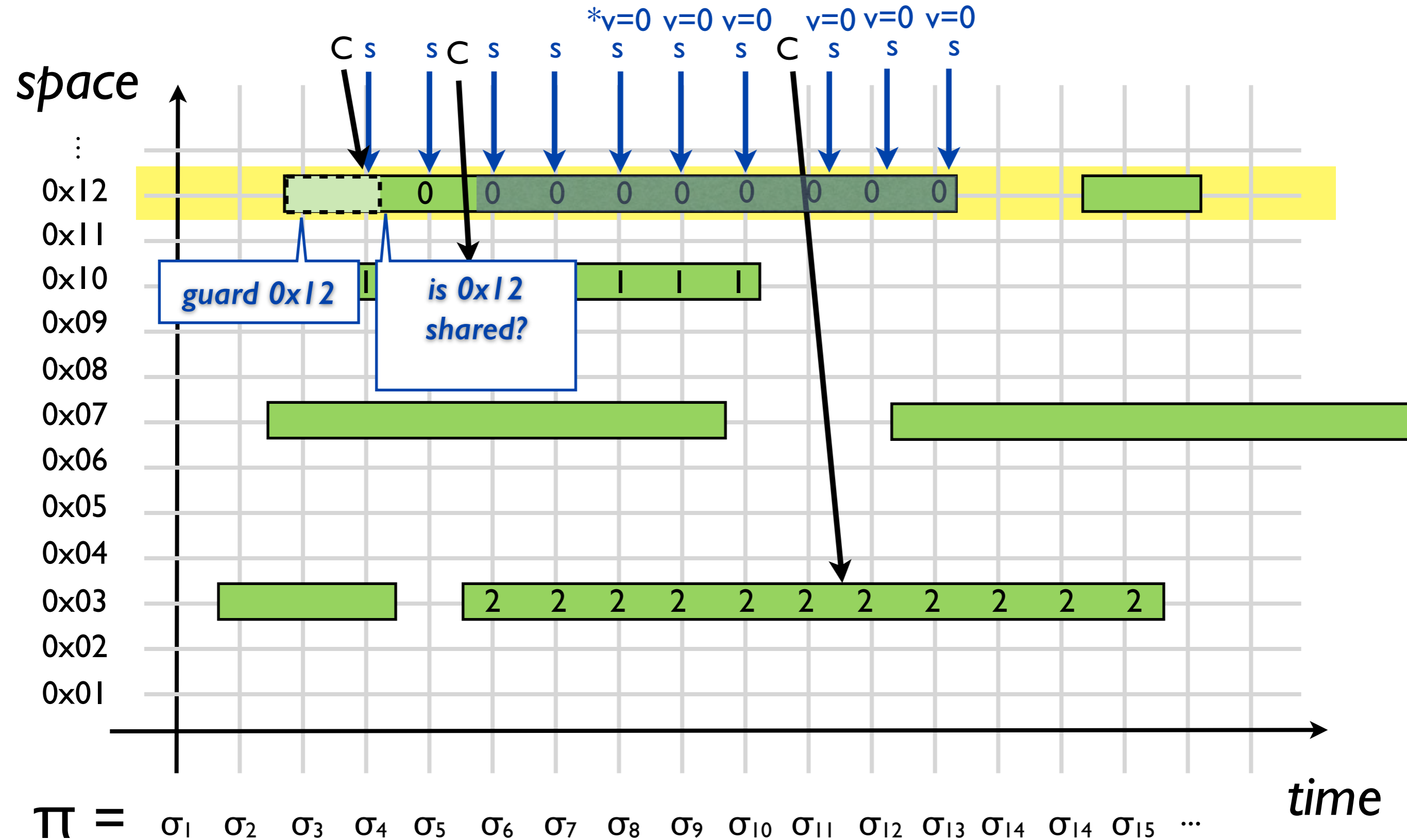
grace-based synchronization



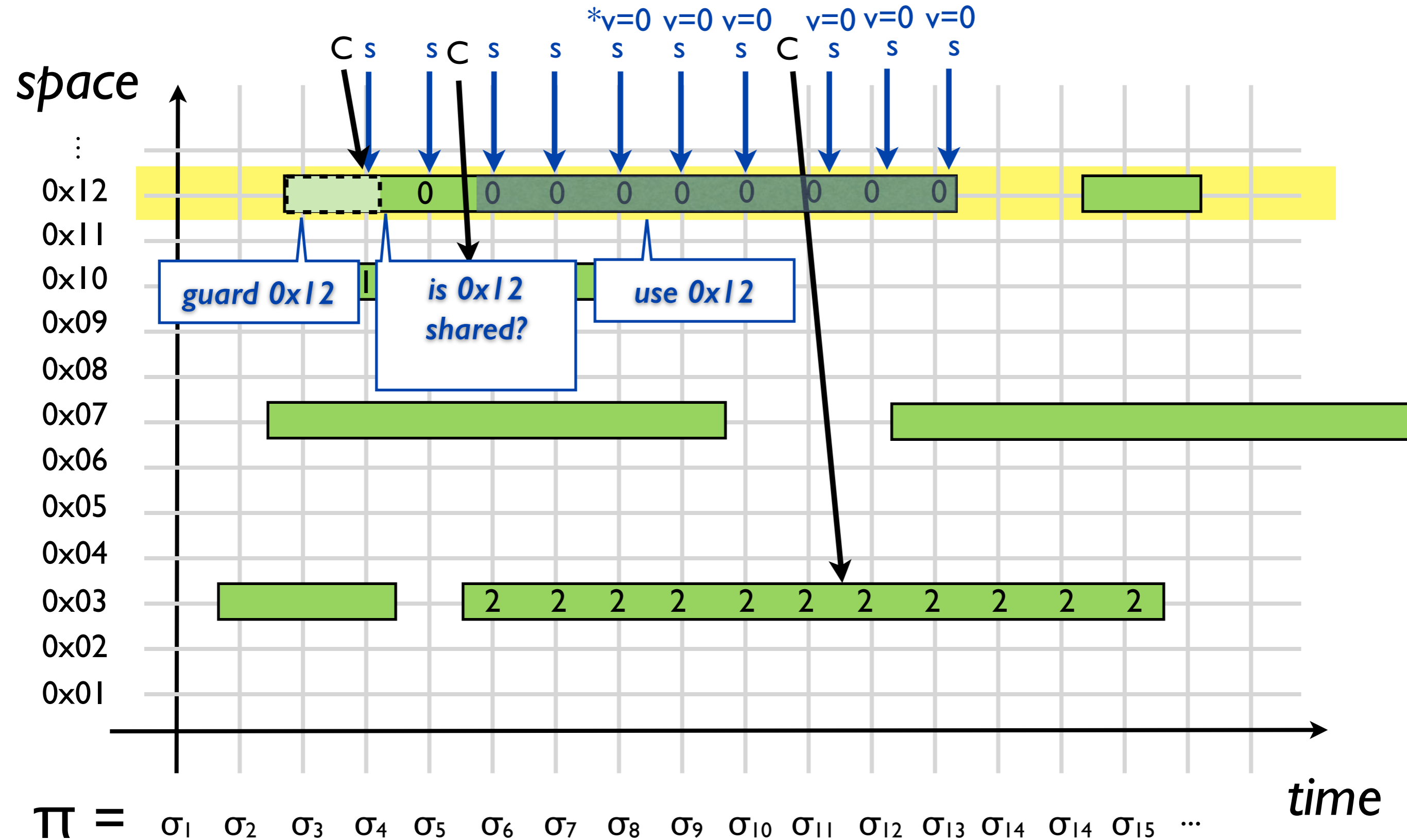
grace-based synchronization



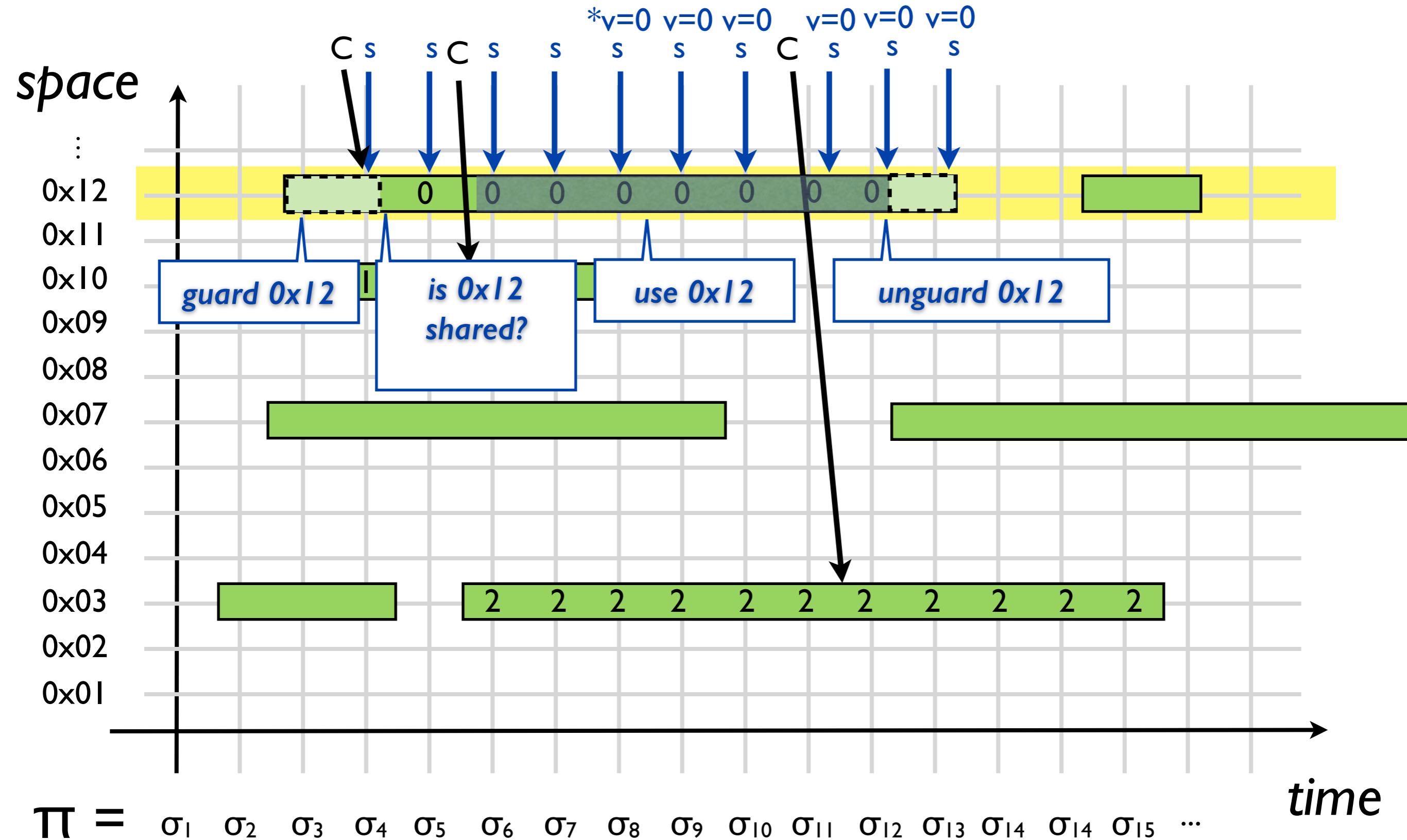
grace-based synchronization



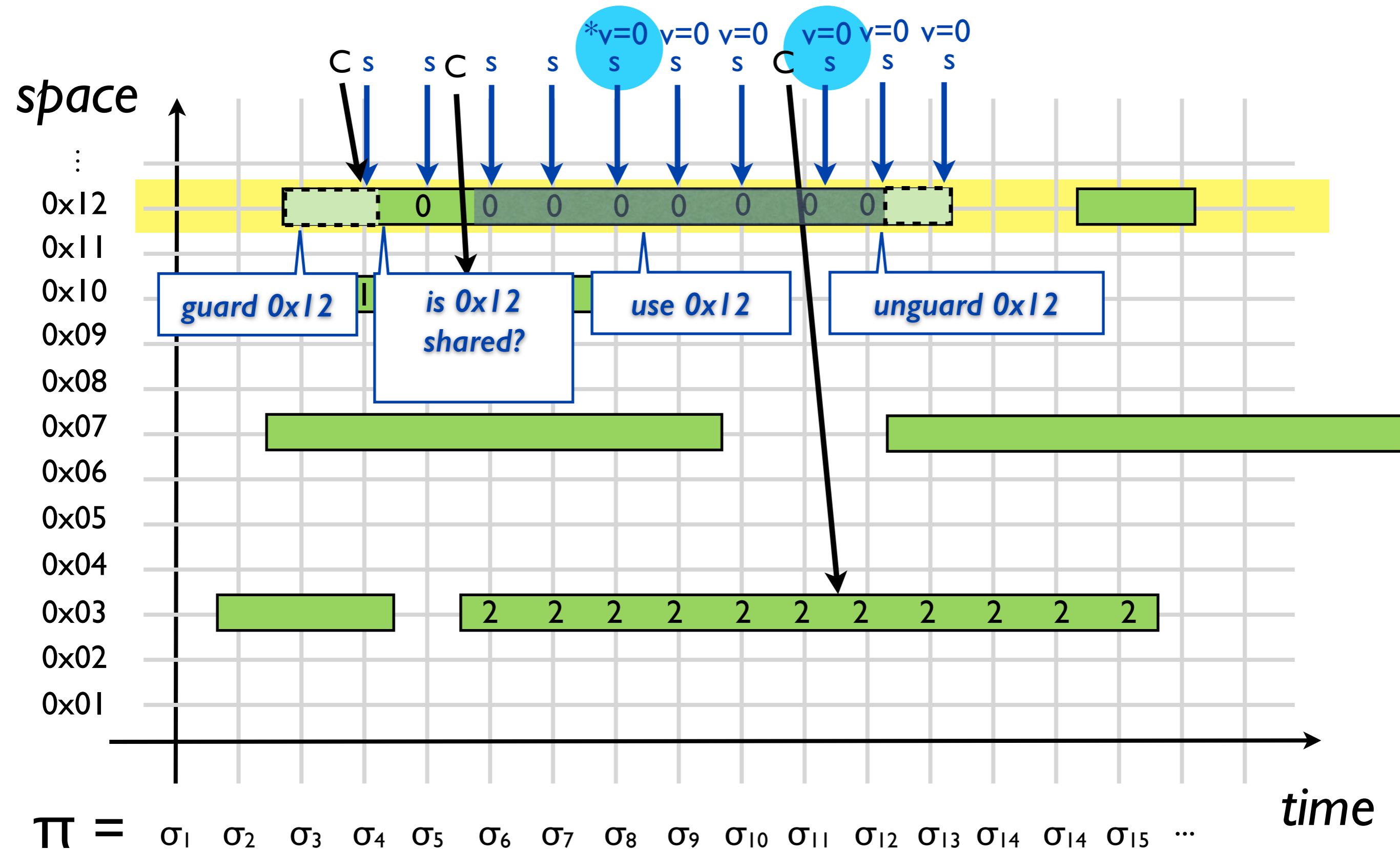
grace-based synchronization



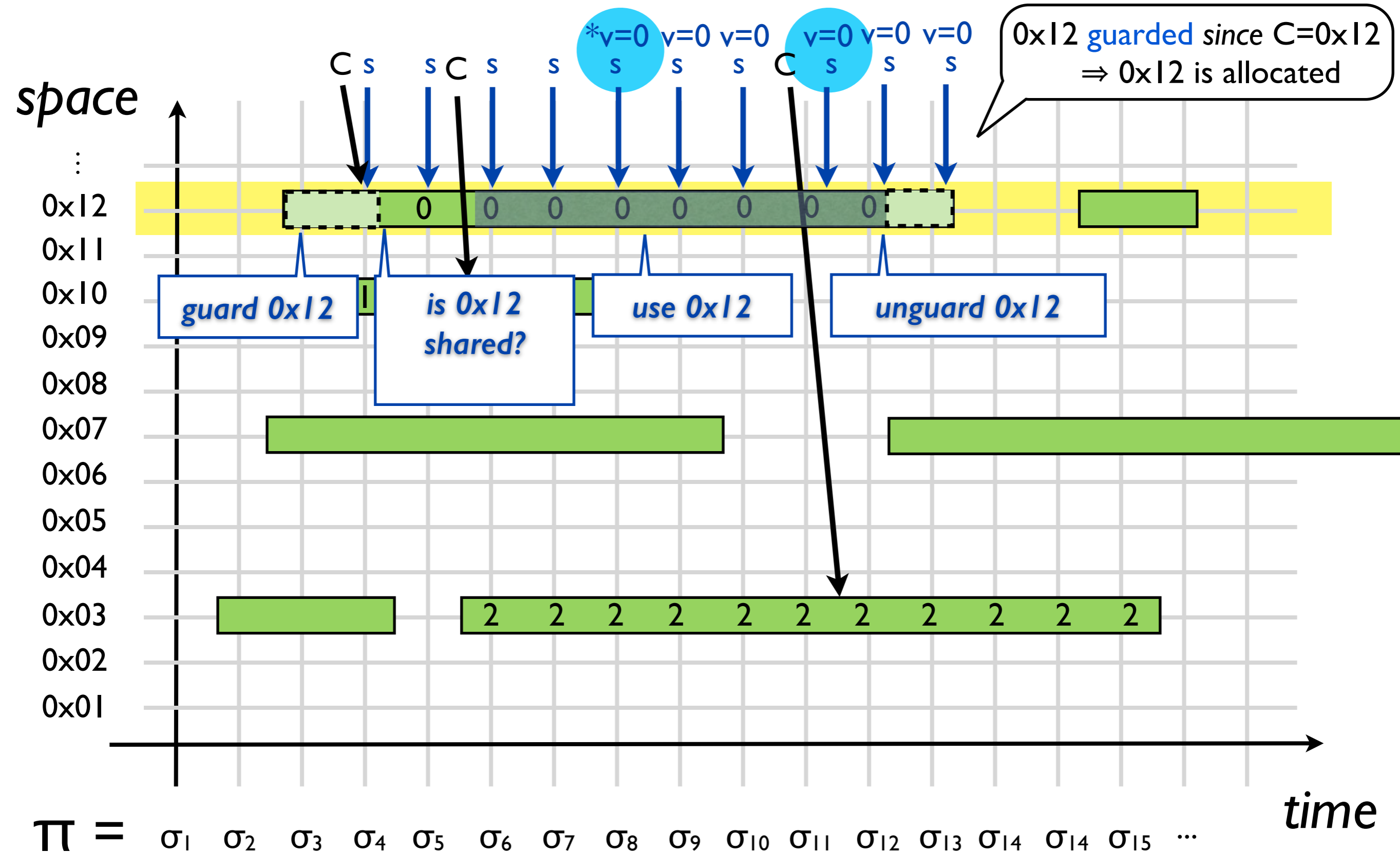
grace-based synchronization



grace-based synchronization

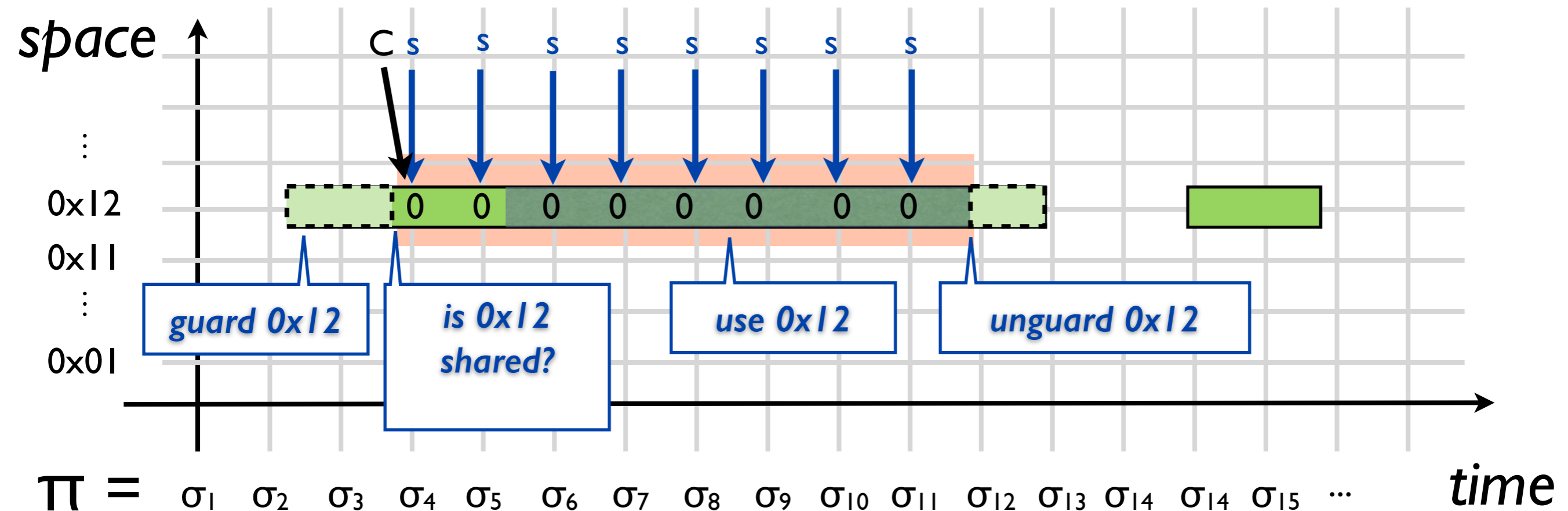


grace-based synchronization



grace-based synchronization

- **grace period** $_{t,s}$ the period of time during which a thread t can access a shared node s without a fear that s might get deallocated



grace-based synchronization

- **grace period** $_{t,s}$ the period of time during which a thread t can access a shared node s without a fear that s might get deallocated
 - ➔ t **access** node s **during** its grace period for s
 - ➔ t **reclaim** node s **after** the grace periods of all t' for s

temporal invariant

- **grace period**_{t,s}
 - if **t guards s** *since s was shared* then **s is allocated**

temporal invariant

- **grace invariant:** $\forall t,s.$ grace period_{t,s}

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ $G_{t,s}$ since R_s \Rightarrow A_s

temporal invariant

- **grace invariant:** $\forall t, s.$ grace period_{t,s}

- $\forall t, s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t, s.$ $G_{t,s}$ since $R_s \Rightarrow A_s$

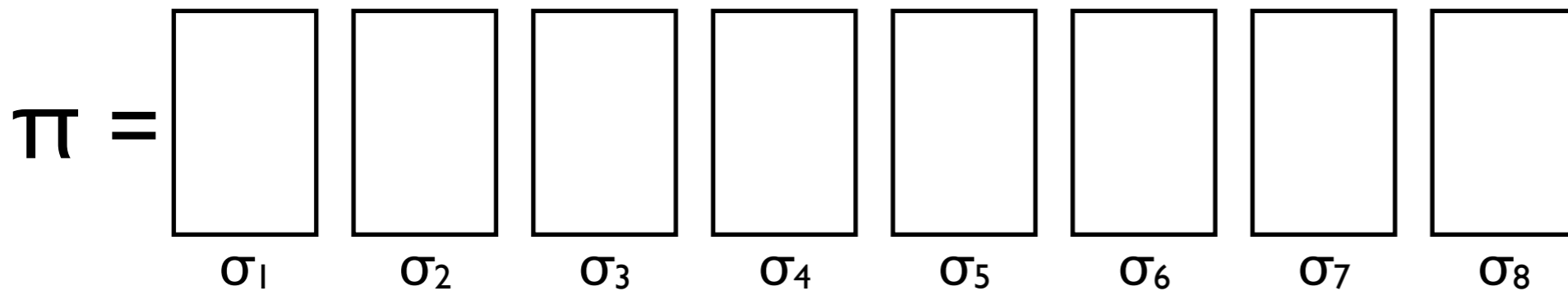
- **Hazard Pointers [Michael'02]**
- **RCU [McKenney+'98]**
- **Epoch [Fraser+'03]**

temporal invariant

- **grace invariant: $\forall t,s.$ grace period $_{t,s}$**

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ **$G_{t,s}$** since **R_s** \Rightarrow **A_s**

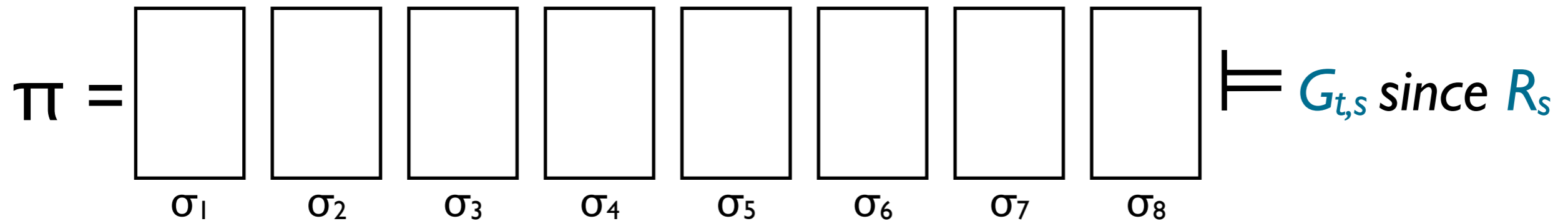


temporal invariant

- **grace invariant: $\forall t,s.$ grace period $_{t,s}$**

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ **$G_{t,s}$** since **R_s** \Rightarrow **A_s**

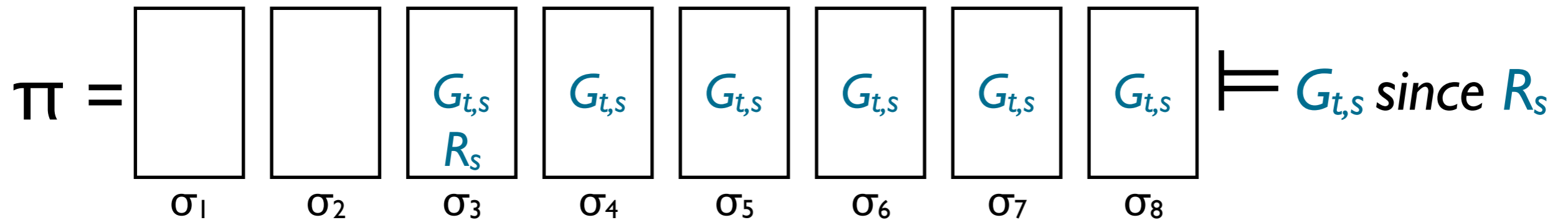


temporal invariant

- **grace invariant: $\forall t,s$. grace period $_{t,s}$**

- $\forall t,s$. **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s$. $G_{t,s}$ since $R_s \Rightarrow A_s$

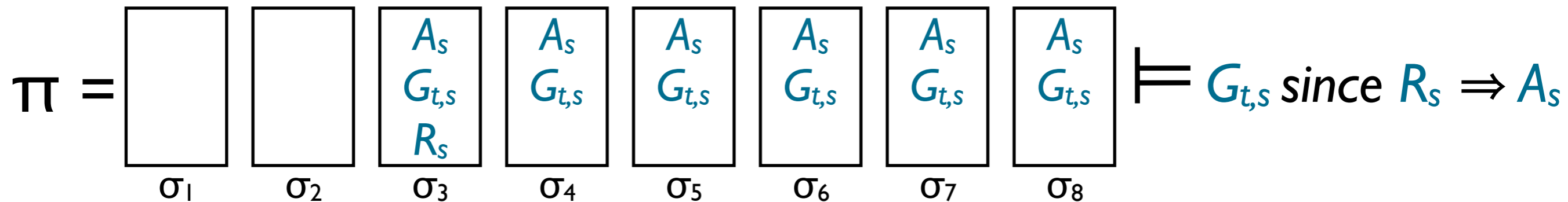
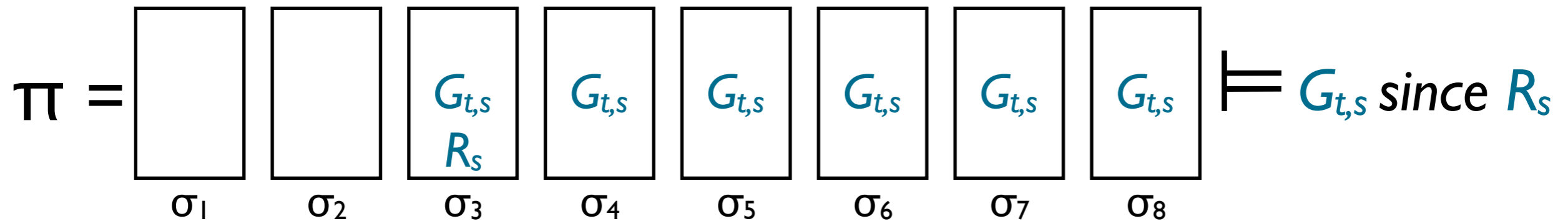


temporal invariant

- **grace invariant:** $\forall t,s.$ grace period_{t,s}

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ **$G_{t,s}$** since **R_s** \Rightarrow **A_s**

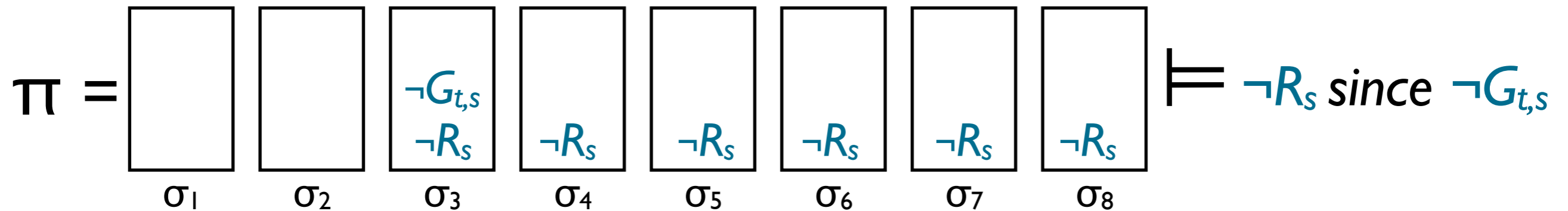


temporal invariant

- **grace invariant: $\forall t,s.$ grace period $_{t,s}$**

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ **$G_{t,s}$** since **R_s** \Rightarrow **A_s**

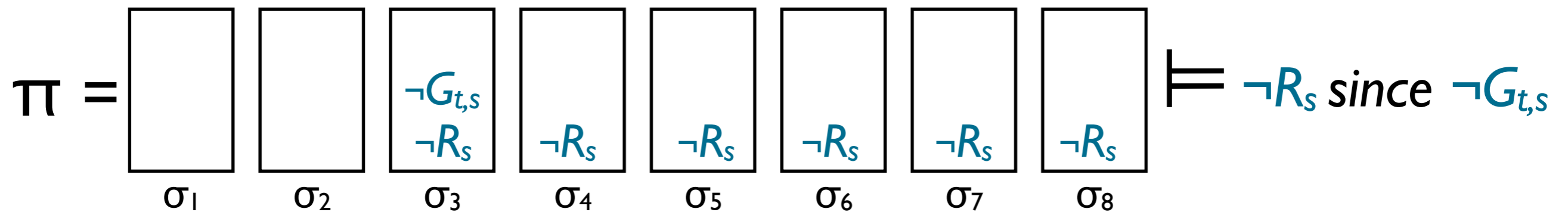


temporal invariant

- **grace invariant:** $\forall t,s.$ grace period_{t,s}

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ **$G_{t,s}$** since **R_s** \Rightarrow **A_s**



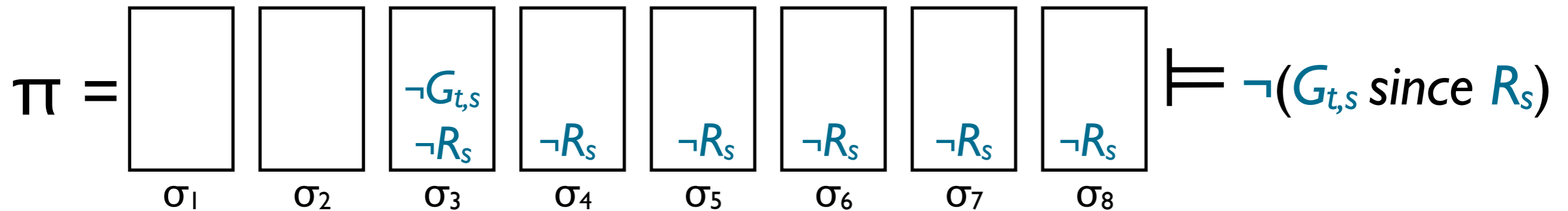
tautology: **$(\neg R_s \text{ since } \neg G_{t,s}) \Rightarrow \neg(G_{t,s} \text{ since } R_s)$**

temporal invariant

- **grace invariant:** $\forall t,s.$ grace period $_{t,s}$

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ $G_{t,s}$ since $R_s \Rightarrow A_s$



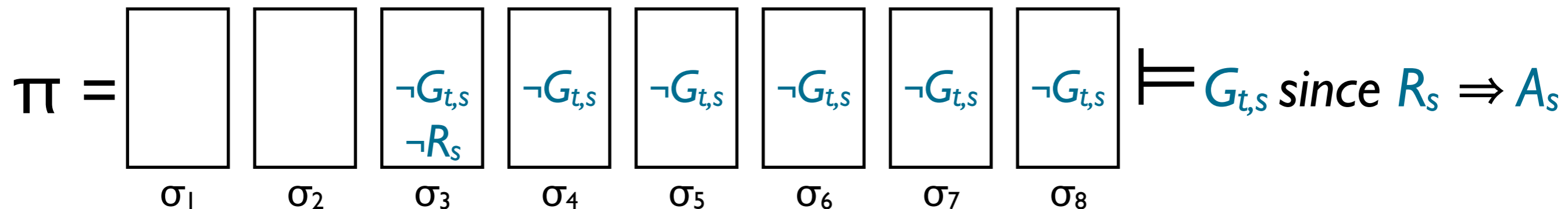
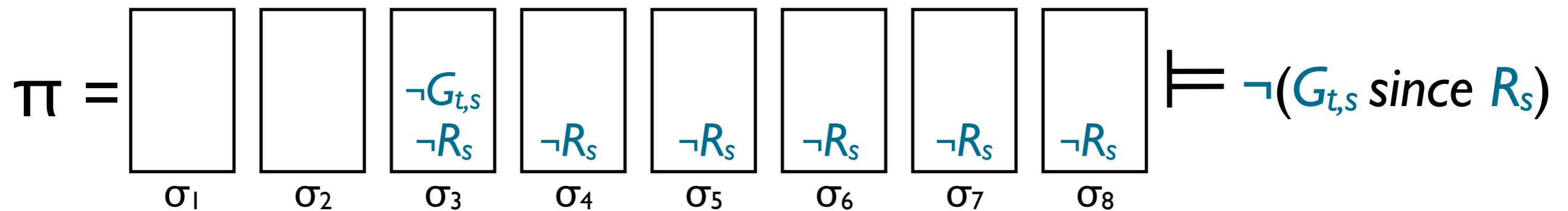
tautology: $(\neg R_s \text{ since } \neg G_{t,s}) \Rightarrow \neg(G_{t,s} \text{ since } R_s)$

temporal invariant

- **grace invariant:** $\forall t,s.$ grace period_{t,s}

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ **$G_{t,s}$** since **R_s** \Rightarrow **A_s**

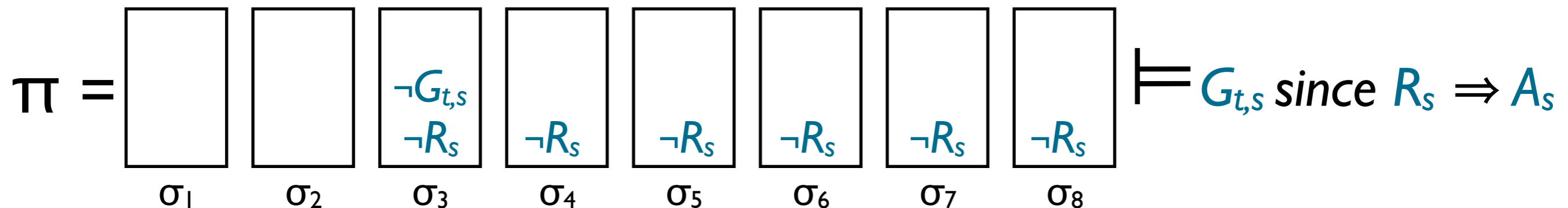
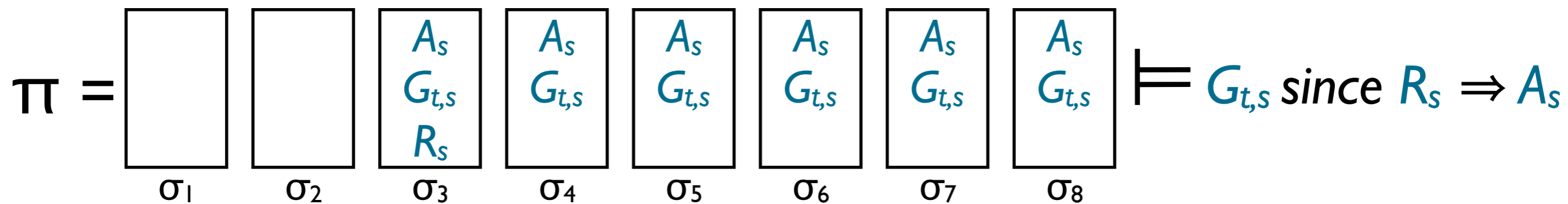


temporal invariant

- **grace invariant:** $\forall t,s.$ grace period $_{t,s}$

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ **$G_{t,s}$** since **R_s** \Rightarrow **A_s**

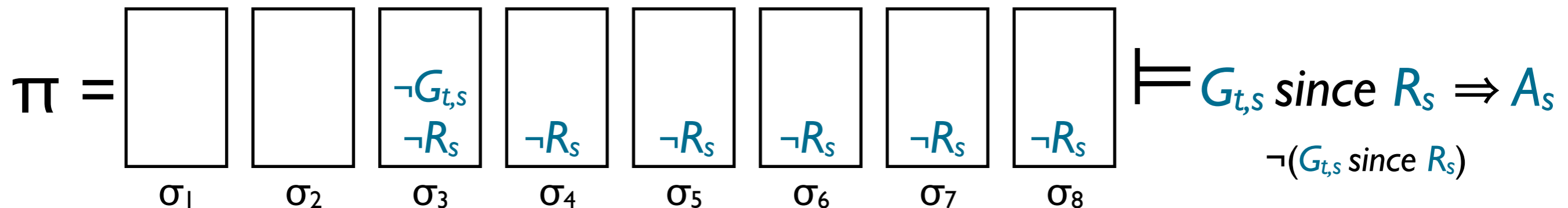
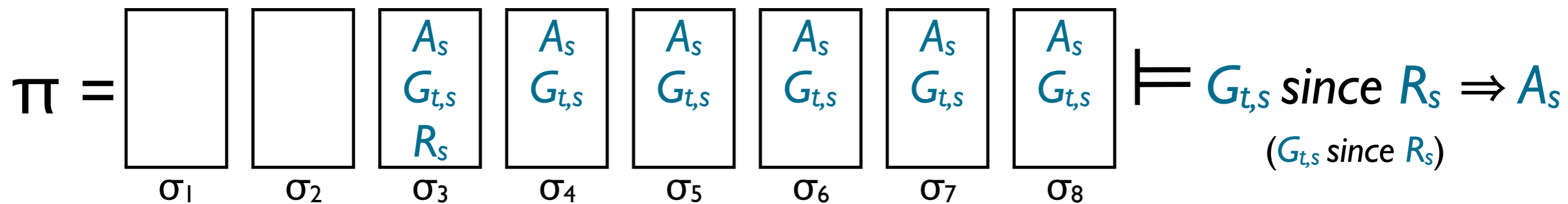


temporal invariant

- **grace invariant:** $\forall t,s.$ grace period_{t,s}

- $\forall t,s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t,s.$ **$G_{t,s}$** since **R_s** \Rightarrow **A_s**



grace-based synchronization

- **grace invariant:** $\forall t, s.$ grace period_{t,s}

- $\forall t, s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t, s.$ $G_{t,s}$ since $R_s \Rightarrow A_s$

➔ **t access s:** $G_{t,s}$ since R_s

➔ **t reclaim s:** $\forall t'. \neg R_s$ since $\neg G_{t',s}$

grace-based synchronization

- **grace invariant:** $\forall t,s.$ grace period_{t,s}

- $\forall t,s.$ t guards s since s was shared then s is allocated

- $\forall t,s.$ $G_{t,s}$ since $R_s \Rightarrow A_s$

➔ t access s : $G_{t,s}$ since $R_s \Rightarrow s$ is allocated)

➔ t reclaim s : $\forall t'. \neg R_s$ since $\neg G_{t,s} \Rightarrow s$ can be deallocated)

★ tautology: $(\neg R_s \text{ since } \neg G_{t,s}) \Rightarrow \neg(G_{t,s} \text{ since } R_s)$

reclamation algorithms (paradigms)

- Hazard Pointers [Michael'02]
- RCU [McKenney Slingwine'98]
- Epoch [Fraser Haris'03]
- pattern of temporal synchronization invariant
 - grace periods
- same idiom for invariant preservation
 - $(\neg R_s \text{ since } \neg G_{t,s}) \Rightarrow \neg(G_{t,s} \text{ since } R_s)$

reclamation algorithms (paradigms)

- Hazard Pointers [Michael'02]
- RCU [McKenney Slingwine'98]
- Epoch [Fraser Haris'03]
- pattern of temporal synchronization invariant
 - grace periods
- same idiom for invariant preservation
 - $(\neg R_s \text{ since } \neg G_{t,s}) \Rightarrow \neg(G_{t,s} \text{ since } R_s)$

(simplified) counter with hazard pointers

```
int *C;
```

```
int inc() {
    int v, *s, *n;
    n = new int;
    do{
        s = C;

        v = *s;
        *n = v + 1;
    }while(!CAS(&C, s, n));
    free(s);

    return v;
}
```

```
int *C;
int *HP[TNUM];
```

```
int inc() {
    int v, *s1, *s2, *n;
    n = new int;
    do{
        do{
            s1 = C;
            HP[tid] = s1;
            s2 = C;
        }while(s1 != s2);
        v = *s1;
        *n = v + 1;
    }while(!CAS(&C, s1, n));
    HP[tid] = null;
    reclaim(s1);
    return v;
}
```

(simplified) counter with hazard pointers

```
int *C;
```

```
int inc() {  
    int v, *s, *n;  
    n = new int;  
    do{  
        s = C;
```

```
        v = *s;  
        *n = v + 1;  
    }while(!CAS(&C, s, n));  
    free(s);
```

```
    return v;  
}
```

```
int *C;  
int *HP[TNUM];
```

```
int inc() {  
    int v, *s1, *s2, *n;  
    n = new int;  
    do{  
        do{  
            s1 = C;  
            HP[tid] = s1;  
            s2 = C;  
        }while(s1 != s2);  
        v = *s1;  
        *n = v + 1;  
    }while(!CAS(&C, s1, n));  
    HP[tid] = null;  
    reclaim(s1);  
    return v;  
}
```

(simplified) counter with hazard pointers

```
int *C;
```

```
reclaim(s) {  
    i = 0;  
    do{  
        do{  
            skip  
        }while(HP[i]== s);  
  
        i = i + 1;  
    }  
  
    free(s);  
}
```

```
int *C;  
int *HP[TNUM];
```

```
int inc() {  
    int v,*s1,*s2,*n;  
    n = new int;  
    do{  
        do{  
            s1 = C;  
            HP[tid] = s1;  
            s2 = C;  
        }while(s1 != s2);  
        v = *s1;  
        *n = v + 1;  
    }while(!CAS(&C,s1,n));  
    HP[tid] = null;  
    reclaim(s1);  
    return v;  
}
```

(simplified)

counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
    i = 0;  
    do{  
        do{  
            skip  
        }while(HP[i]== s);  
  
        i = i + 1;  
    }  
  
    free(s);  
}
```

guard s1

```
int inc() {  
    int v,*s1,*s2,*n;  
    n = new int;  
    do{  
        do{  
            s1 = C;  
            HP[tid] = s1;  
            s2 = C;  
        }while(s1 != s2);  
        v = *s1;  
        *n = v + 1;  
    }while(!CAS(&C,s1,n));  
    HP[tid] = null;  
    reclaim(s1);  
    return v;  
}
```

(simplified) counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
    i = 0;  
    do{  
        do{  
            skip  
        }while(HP[i]== s);  
  
        i = i + 1;  
    }  
  
    free(s);  
}
```

guard s1

is s1 shared?
(C=s1?)

```
int inc() {  
    int v,*s1,*s2,*n;  
    n = new int;  
    do{  
        do{  
            s1 = C;  
            HP[tid] = s1;  
            s2 = C;  
        }while(s1 != s2);  
        v = *s1;  
        *n = v + 1;  
    }while(!CAS(&C,s1,n));  
    HP[tid] = null;  
    reclaim(s1);  
    return v;  
}
```

(simplified) counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

guard s1

is s1 shared?
(C=s1?)

use s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified) counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
    i = 0;  
    do{  
        do{  
            skip  
        }while(HP[i]== s);  
  
        i = i + 1;  
    }  
  
    free(s);  
}
```

guard s1

is s1 shared?
(C=s1?)

use s1

unguard s1

```
int inc() {  
    int v,*s1,*s2,*n;  
    n = new int;  
    do{  
        do{  
            s1 = C;  
            HP[tid] = s1;  
            s2 = C;  
        }while(s1 != s2);  
        v = *s1;  
        *n = v + 1;  
    }while(!CAS(&C,s1,n));  
    HP[tid] = null;  
    reclaim(s1);  
    return v;  
}
```


(simplified) counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

assuming $C \neq s$

guard s1

is s1 shared?
($C=s1$?)

use s1

unguard s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified) counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

assuming $C \neq s$

t_i does not
guard s

guard $s1$

is $s1$ shared?
($C=s1$?)

use $s1$

unguard $s1$

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

partial ownership

- proof hinges on $C \neq s$ during `retire(s)`
- ensured by **ownership** annotations
 - the thread which removes `s` can
 - $C = s$
 - `free (s)`
 - other threads can only access `s`

(simplified)

counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

HP[tid]=s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)

counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

HP[tid]=s1

HP[tid]=s1
 $\wedge C=s2$

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

HP[tid]=s1

HP[tid]=s1
since C=s2

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

HP[tid]=s1

HP[tid]=s1
since C=s2

HP[tid]=s1
since C=s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```


(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

HP[tid]=s1

HP[tid]=s1
since C=s2

HP[tid]=s1
since C=s1

s1 is allocated
(memory safety)

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

HP[tid]=s1

HP[tid]=s1
since C=s2

HP[tid]=s1
since C=s1

s1 is allocated
(memory safety)

s1 is allocated
since C=s
(no ABA)

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)

counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

C≠s

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

$C \neq s$

$\forall t < i. C \neq s \text{ since } HP[t] \neq s$

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

$C \neq s$

$\forall t < i. C \neq s \text{ since } HP[t] \neq s$

$\forall t. C \neq s \text{ since } HP[t] \neq s$

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

(simplified)
counter with hazard pointers

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

$\forall t,s. (HP[t]=s \text{ since } C=s) \Rightarrow \text{allocated}(s)$

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

$C \neq s$

$\forall t < i. C \neq s \text{ since } HP[t] \neq s$

$\forall t. C \neq s \text{ since } HP[t] \neq s$

$\forall t. (C \neq s \text{ since } HP[t] \neq s) \Rightarrow \neg(HP[t]=s \text{ since } C=s)$

(Invariant is preserved)

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

grace-based synchronization

- **grace invariant:** $\forall t, s.$ grace period_{t,s}

- $\forall t, s.$ **t guards s** since **s was shared** then **s is allocated**

- $\forall t, s.$ $G_{t,s}$ since $R_s \Rightarrow A_s$

➔ **t access s:**

$G_{t,s}$ since R_s
 $HP[t]=s$ since $C=s$

➔ **t_l reclaim s:**

$\forall t. \neg R_s$ since $\neg G_{t,s}$
 $\forall t. C \neq s$ since $HP[t] \neq s$

★ **tautology:** $(\neg R_s \text{ since } \neg G_{t,s}) \Rightarrow \neg(G_{t,s} \text{ since } R_s)$

temporal separation logic

$$R, G, I \vdash \{P\} C \{Q\}$$

temporal separation logic

$R, G, I \vdash \{P\} C \{Q\}$

$$\sigma = (\boxed{p}, \boxed{s}) \in \Sigma_p \times \Sigma_s$$

$$\omega = (\boxed{p}, \boxed{s} \boxed{s} \boxed{s}) \in \Sigma_p \times \Sigma_s^*$$

temporal separation logic

$$R, G, I \vdash \{P\} C \{Q\}$$

$$\sigma = (\boxed{p}, \boxed{s}) \in \Sigma_p \times \Sigma_s$$

$$\omega = (\boxed{p}, \boxed{s} \boxed{s} \boxed{s}) \in \Sigma_p \times \Sigma_s^*$$

$$P, Q = \{(\boxed{p}, \boxed{s} \boxed{s} \boxed{s}), \dots\} \subseteq \Sigma_p \times \Sigma_s^*$$

$$I = \{\boxed{s} \boxed{s} \boxed{s}, \dots\} \subseteq \Sigma_s^*$$

$$R_t, G_t = \{(\boxed{s}, \boxed{s}), \dots\} \subseteq \Sigma_s \times \Sigma_s$$

temporal separation logic

$$R, G, I \vdash \{P\} C \{Q\}$$

$$\sigma = (\boxed{p}, \boxed{s}) \in \Sigma_p \times \Sigma_s$$

$$\omega = (\boxed{p}, \boxed{s} \boxed{s} \boxed{s}) \in \Sigma_p \times \Sigma_s^*$$

$$P, Q = \{(\boxed{p}, \boxed{s} \boxed{s} \boxed{s}), \dots\} \subseteq \Sigma_p \times \Sigma_s^*$$

$$I = \{\boxed{s} \boxed{s} \boxed{s}, \dots\} \subseteq \Sigma_s^*$$

$$R_t, G_t = \{(\boxed{s}, \boxed{s}), \dots\} \subseteq \Sigma_s \times \Sigma_s$$

temporal logic, separation logic, rely/guarantee, local actions, stability, temporal assertions, spatial resource invariants, local/shared state partitioning, permissions, temporal invariant

proof (retire)

```
1 {V ⊨ p ↦m - * Ftid ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}} \wedge \boxed{p \mapsto_e - * \text{true}}$ }
2 void retire(int* p) {
3   {V ⊨ p ↦m - * ∃A. detached ↦ A * D(A) ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}} \wedge \boxed{p \mapsto_e - * \text{true}}$ }
4   insert(detached, p);
5   {V ⊨ ∃A. detached ↦ A * D(A) ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}}$ }
6   if (nondet())
7     {V ⊨ Ftid ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}}$ }
8     return;
9   Set used;
10  {V ⊨ ∃A. detached ↦ A * D(A) ∧ used = ∅ ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}}$ }
11  while (!isEmpty(detached)) {
12    {V ⊨ ∃A. detached ↦ A * D(A) * D(used) ∧ A ≠ ∅ ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}}$ }
13    bool my = true;
14    Node *n = pop(detached);
15    {V ⊨ my ∧ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - *  $\boxed{n \mapsto_e - * \text{true}}$  *  $\boxed{H * \exists y. y \neq n \wedge C \mapsto y * y \mapsto - * \text{true}}$ }
16    for (int i = 0; i < N && my; i++) {
17      {V ⊨ my ∧ 0 ≤ i < N ∧ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - *  $\boxed{n \mapsto_e - * \text{true}}$  *
18         $\boxed{H * \exists y. y \neq n \wedge C \mapsto y * y \mapsto - * \text{true}} \wedge \forall 0 \leq j < i. \boxed{\exists y. y \neq n \wedge C \mapsto y * y \mapsto - * \text{true}}$  since  $\boxed{HP[i] \neq n * \text{true}}$ }
19      if (<HP[i] == n>id)
20        my = false;
21      {V ⊨ 0 ≤ i < N ∧ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - *  $\boxed{n \mapsto_e - * \text{true}}$  *
22         $\boxed{H * \exists y. y \neq n \wedge C \mapsto y * y \mapsto - * \text{true}} \wedge (my \Rightarrow \forall 0 \leq j \leq i. \boxed{\exists y. y \neq n \wedge C \mapsto y * y \mapsto - * \text{true}}$  since  $\boxed{HP[j] \neq n * \text{true}}$ )}
23    }
24    {V ⊨ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - *  $\boxed{n \mapsto_e - * \text{true}}$  ∧
25       $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}} \wedge (my \Rightarrow \forall 0 \leq j \leq N. \boxed{\exists y. y \neq n \wedge C \mapsto y * y \mapsto - * \text{true}}$  since  $\boxed{HP[j] \neq n * \text{true}}$ )}
26    if (my) {
27      < ; >Take;
28      {V ⊨ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - *  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}}$ }
29      free(n);
30    } else {
31      insert(used, n);
32    }
33    {V ⊨ ∃A. detached ↦ A * D(A) * D(used) ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}}$ }
34  }
35  {V ⊨ detached ↦ ∅ * D(used) ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}}$ }
36  moveAll(detached, used);
37  {V ⊨ Ftid ∧  $\boxed{H * \exists y. C \mapsto y * y \mapsto - * \text{true}}$ }
38 }
```

summary

- **Hazard Pointers [Michael'02]**

- non-blocking reclaim, single reading of hazard array, dynamic allocation of hazard pointers, non-blocking stack, reuse of next-pointer

- **RCU [McKenney Slingwine'98]**

- first formalization

- **Epoch [Fraser Haris'03]**

★ invariant: $\forall t,s. G_{t,s} \text{ since } R_s \Rightarrow A_s$

★ tautology: $(\neg R_s \text{ since } \neg G_{t,s}) \Rightarrow \neg(G_{t,s} \text{ since } R_s)$

related work

- separation logic [Reynolds, LICS'02] [O'Hearn⁺ POPL'01]
- CSL [O'Hearn, TCS'07]
- Stack + HP in CSL [Parkinson⁺, POPL'07]
- reductions [Elmas⁺, POPL'09]
- R/G Separation Logic [Vefiadis, PhD'08]
- temporal separation logic [Fu⁺, CONCUR'10]
- interval temporal logic [Tofan⁺, ICTAC'11]

future work

- **Pass the Buck [Herlihy⁺, DISC'02]**
- **weak memory**

conclusions

- *since* operator as abstraction of histories
 - natural specification
 - complicates logic
- programming patterns \Rightarrow proof patterns
 - simplify proofs using specialized rules

thank you!

“weirdnesses”

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

C ≠ s

t_i does not
guard s

guard s1

is s1 shared?
(C=s1?)

use s1

unguard s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

“weirdnesses”

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

C ≠ s

t_i does not guard s

guard s1

is s1 shared?
(C=s1?)

use s1

unguard s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

s1 might get deallocated

“weirdnesses”

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

C ≠ s

t_i does not guard s

guard s1

is s1 shared?
(C=s1?)

use s1

unguard s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

s1 might get deallocated

“weirdnesses”

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

C ≠ s

t_i does not guard s

t_i might guard s

guard s1

is s1 shared?
(C=s1?)

use s1

unguard s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CAS(&C,s1,n));  
  HP[tid] = null;  
  reclaim(s1);  
  return v;  
}
```

s1 might get deallocated

“weirdnesses”

```
int *C;
```

```
int *C;  
int *HP[TNUM];
```

```
reclaim(s) {  
  i = 0;  
  do{  
    do{  
      skip  
    }while(HP[i]== s);  
  
    i = i + 1;  
  }  
  
  free(s);  
}
```

C ≠ s

t_i does not guard s

t_i might guard s

guard s1

is s1 shared?
(C=s1?)

use s1

unguard s1

```
int inc() {  
  int v,*s1,*s2,*n;  
  n = new int;  
  do{  
    do{  
      s1 = C;  
      HP[tid] = s1;  
      s2 = C;  
    }while(s1 != s2);  
    v = *s1;  
    *n = v + 1;  
  }while(!CA);  
  HP[tid] =  
  reclaim(s1);  
  return v;  
}
```

s1 might get deallocated

s1 reallocated
⇒ “reguarded”