

# Token Machine Language (TML): An Intermediate Language for Sensor Networks

Ryan Newton, Arvind  
MIT CSAIL  
[{newton, arvind}@mit.edu](mailto:{newton, arvind}@mit.edu)

Matt Welsh  
Harvard University  
[mdw@eecs.harvard.edu](mailto:mdw@eecs.harvard.edu)

# High Level Languages

## Middleware

## Node-level runtime

# High Level Languages

Query Langs (Cougar, TinyDb)

Data parallel langs  
Regiment

Spatial Views, EIP

Rule-based langs

## Middleware

## Node-level runtime

# High Level Languages

Query Langs (Cougar, TinyDb)

Spatial Views, EIP

Data parallel langs

Regiment

Rule-based langs

# Middleware

Routing

Neighborhood mgmt

Neighborhood mgmt

Gradients

Localization

Heartbeats

Naming/Discovery

Caching

# Node-level runtime

# High Level Languages

Query Langs (Cougar, TinyDb)

Data parallel langs

Regiment

Spatial Views, EIP

Rule-based langs

# Middleware

Routing

Gradients

Neighborhood mgmt

Localization

Heartbeats

Neighborhood mgmt

Naming/Discovery

Caching

# Node-level runtime

TinyOS/NesC

Sensing

Event handling

Local messaging

Concurrency

Resource mgmt

# High Level Languages

Query Langs (Cougar, TinyDb)

Data parallel langs

Regiment

Spatial Views, EIP

Rule-based langs

# Compile

## Middleware

Routing

Neighborhood mgmt

Neighborhood mgmt

Naming/Discovery

Gradients

Localization

Heartbeats

Caching

# Down

## Node-level runtime

TinyOS/NesC

Sensing

Event handling

Local messaging

Concurrency

Resource mgmt



# High Level Languages

Query Langs (Cougar, TinyDb)

Spatial Views, EIP

Data parallel langs

Regiment

Compile

Rule-based langs

## Middleware

Routing

Neighborhood mgmt

Neighborhood mgmt

Gradients

Heartbeats

Caching

Localization

Naming/recovery

Down



## Node-level runtime

TinyOS/NesC

Sensing

Event handling

Local messaging

Concurrency

Resource mgmt

# Question

# Question

— [ Can an intermediate language for sensor networks

# Question

- [ Can an intermediate language for sensor networks
  - fit Tiny architectures

# Question

- Can an intermediate language for sensor networks
  - fit Tiny architectures
  - be conducive to building higher abstractions  
(expressive, extensible)

# Question

- Can an intermediate language for sensor networks
  - fit Tiny architectures
  - be conducive to building higher abstractions  
(expressive, extensible)
  - be semantically *simple* and easy to reason about

# Potential models for computation

	Tiny?	Rich/ Extensible?	Clean, Simple Model?
TinyOS/NesC	T	T	?
TinyDB	T	?	T
JVM	?	T	T

# Our approach: token machines

- Atomic actions
- Unified control, communication, and storage model (tokens)
- Simple and lightweight model

# Token Machine Model

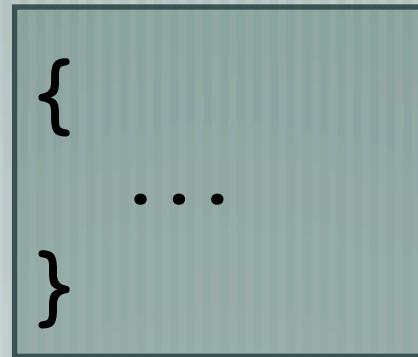
# TM Model

# TM Model : concurrency

```
{  
...  
}
```

# TM Model : concurrency

Atomic  
Blocks



# TM Model : concurrency

## Atomic Blocks

```
{  
    if ( . . . )  
    { . . . }  
}
```

# TM Model : concurrency

Atomic  
Blocks

```
{  
    if ( . . . )  
    { . . . }  
}
```

One execution context

# TM Model : concurrency

Atomic  
Blocks

```
token T
{
    if (...)
    ...
}
```

One execution context

# TM Model : concurrency

Atomic  
Blocks

```
token T
{
    if (... )
    { ... }
}
```

```
token
{
    if (... )
    { ... }
}
```

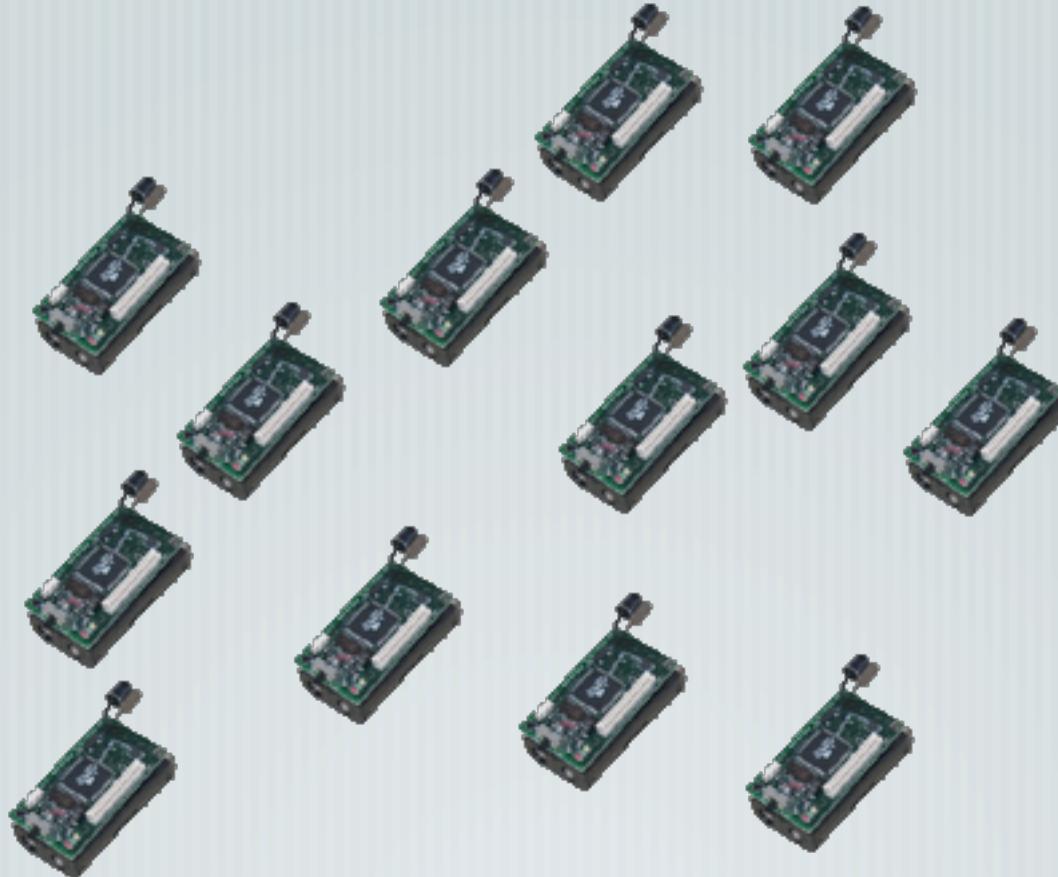
```
token
{
    if (... )
    { ... }
}
```

```
token
{
    if (... )
    { ... }
}
```

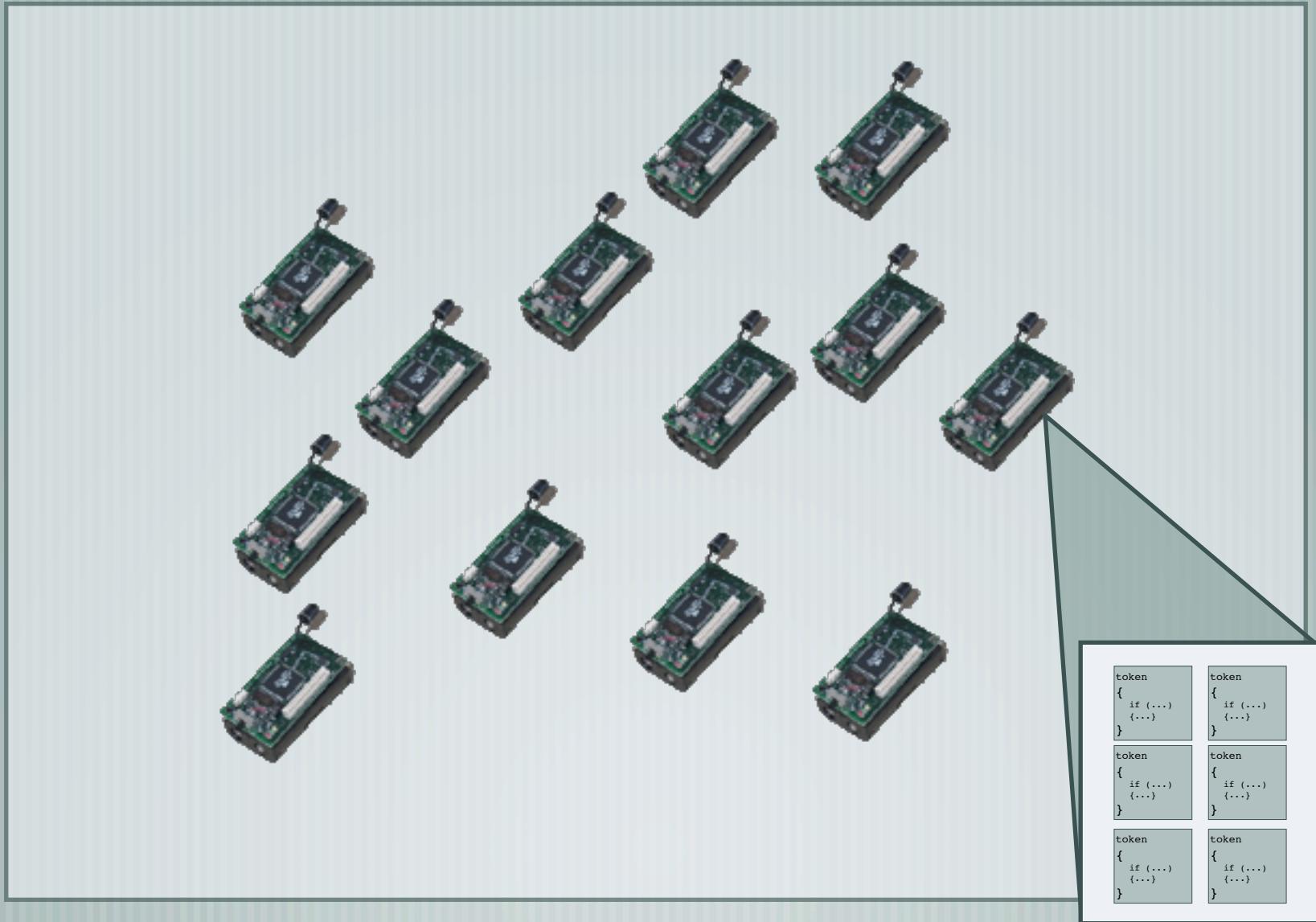
One execution context

```
token
{
    if (... )
    { ... }
}
```

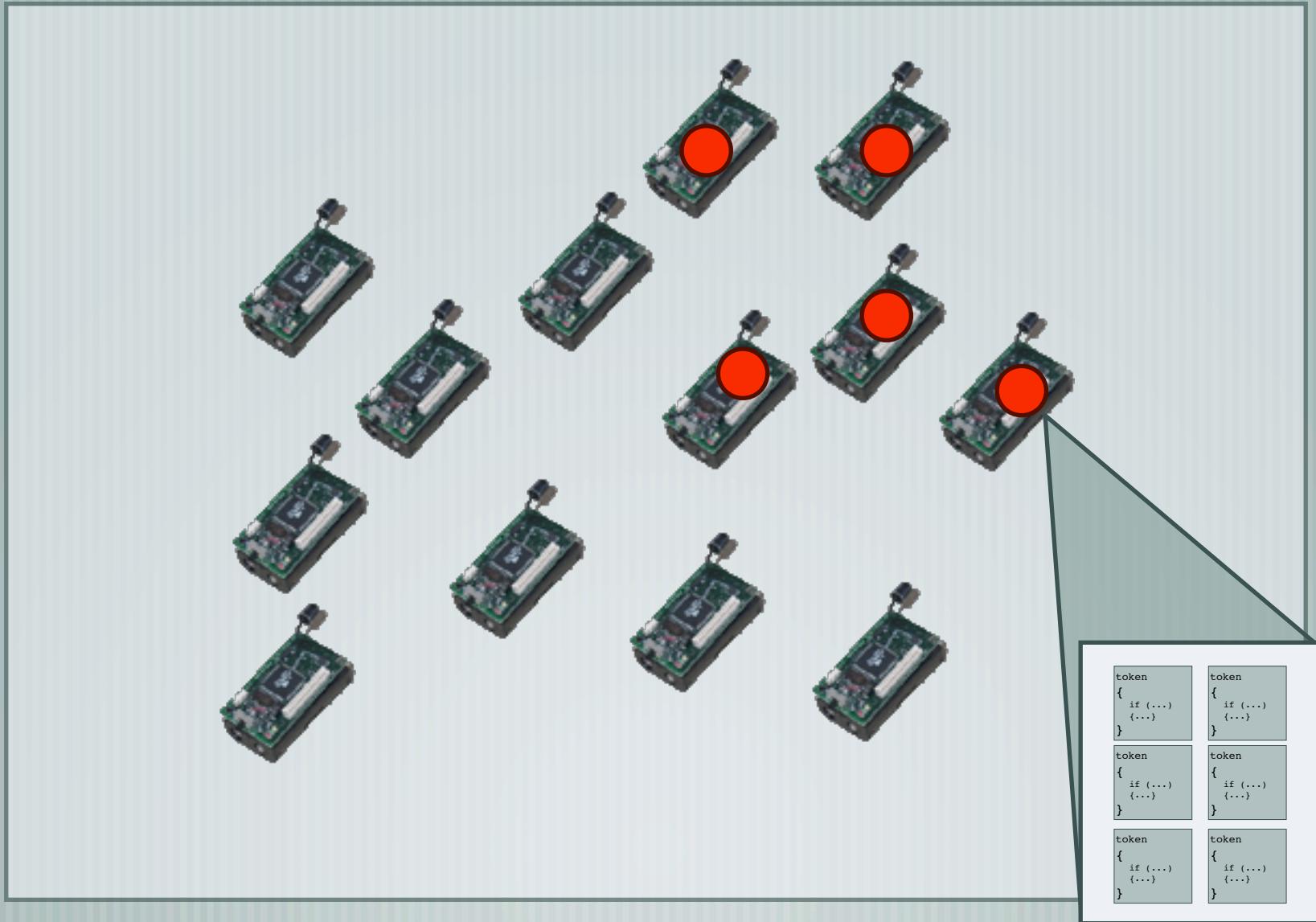
# TM Model : big picture



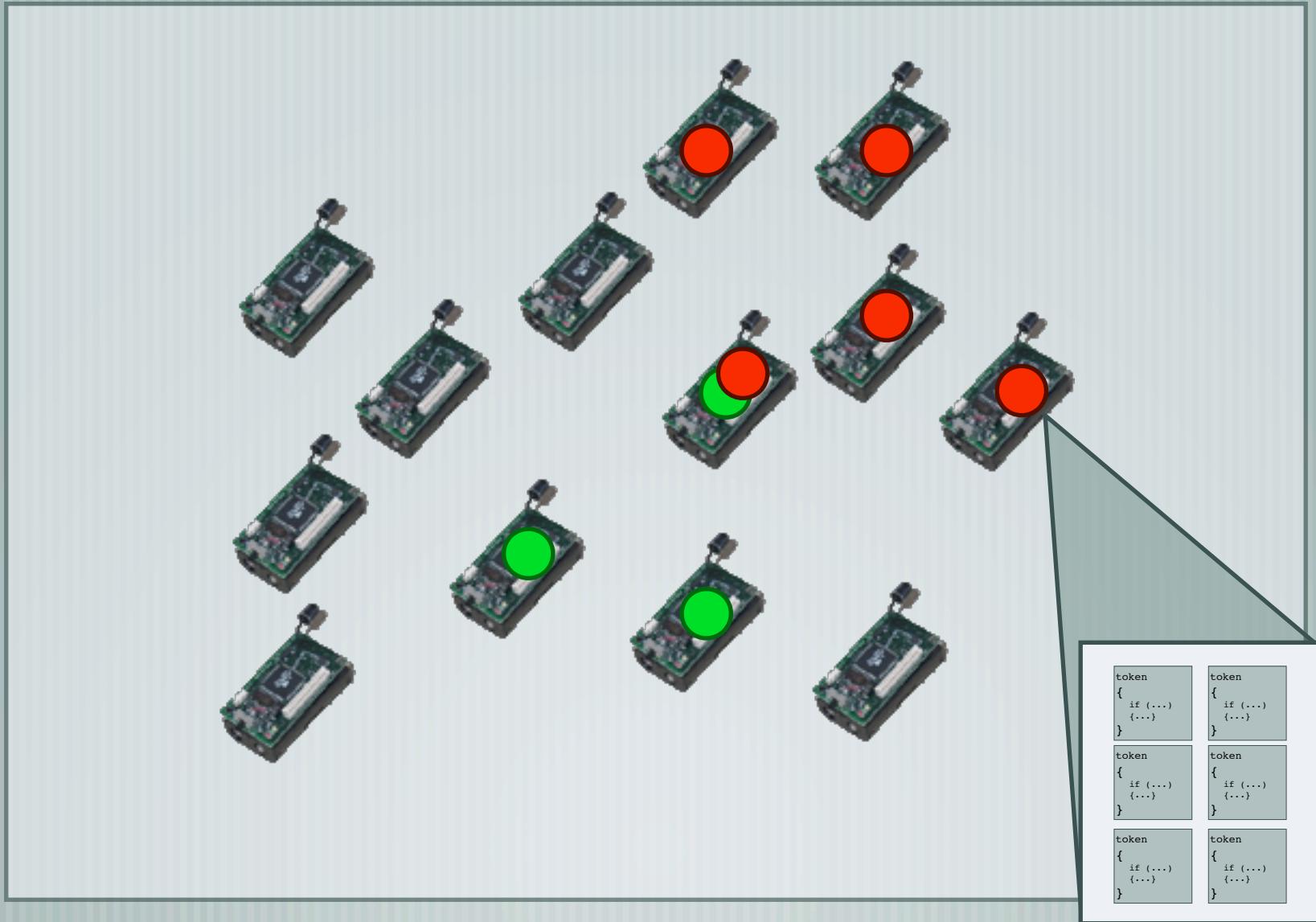
# TM Model : big picture



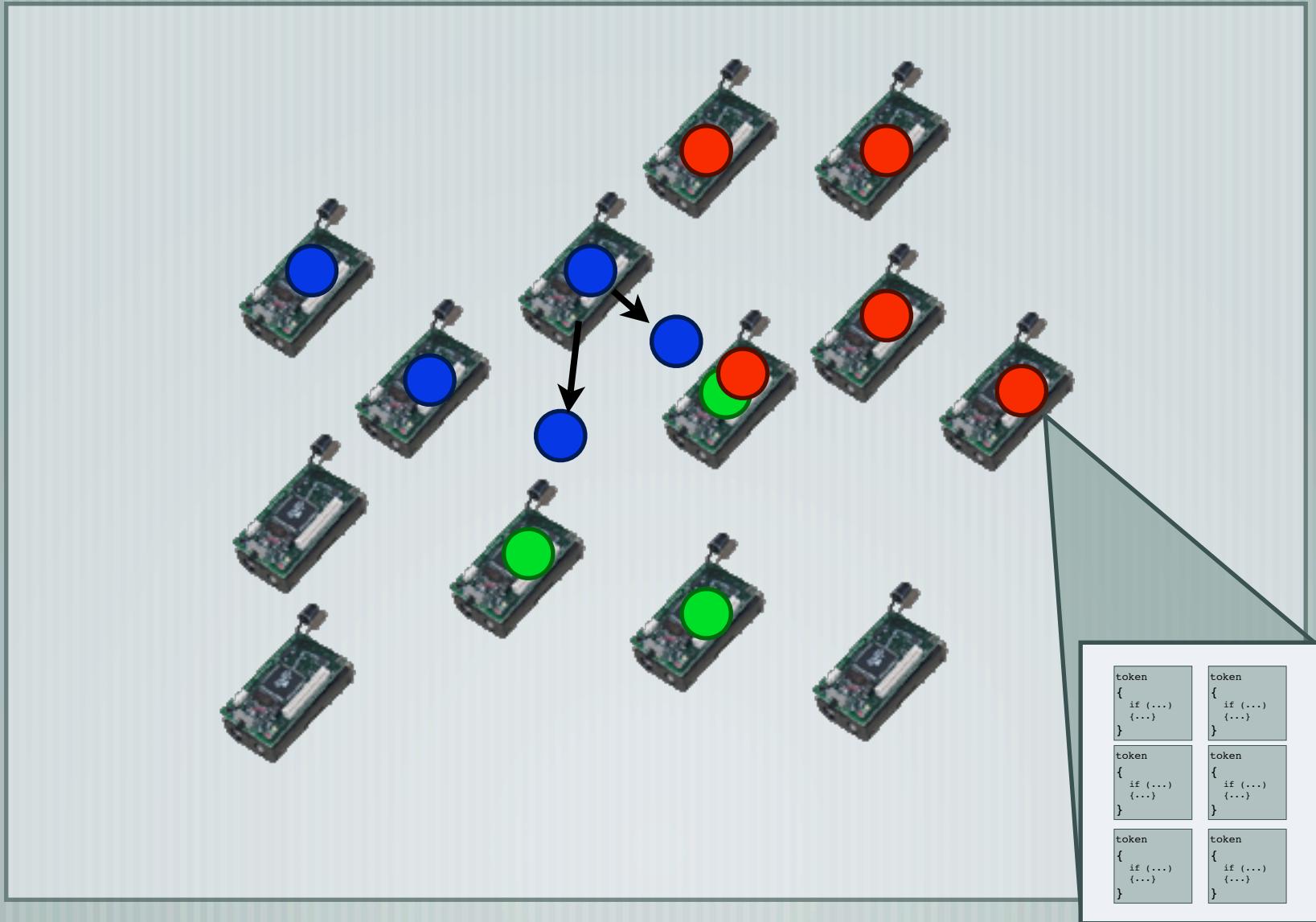
# TM Model : big picture



# TM Model : big picture



# TM Model : big picture

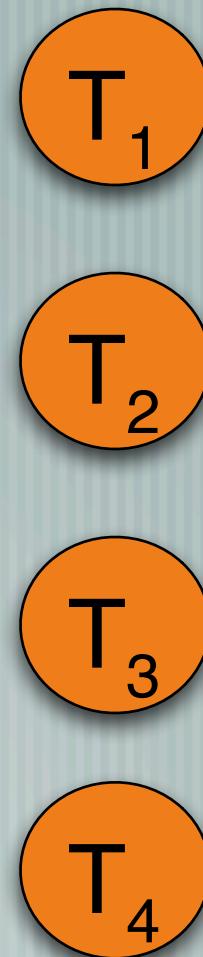


# TM Model : tokens

```
token T  
{  
    if (...)  
    {...}  
}
```

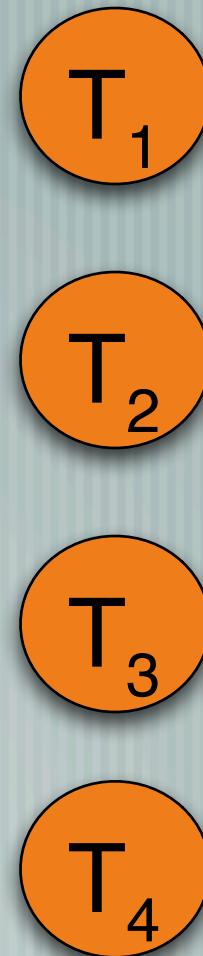
# TM Model : tokens

```
token T1  
{  
    if (...)  
    {...}  
}
```



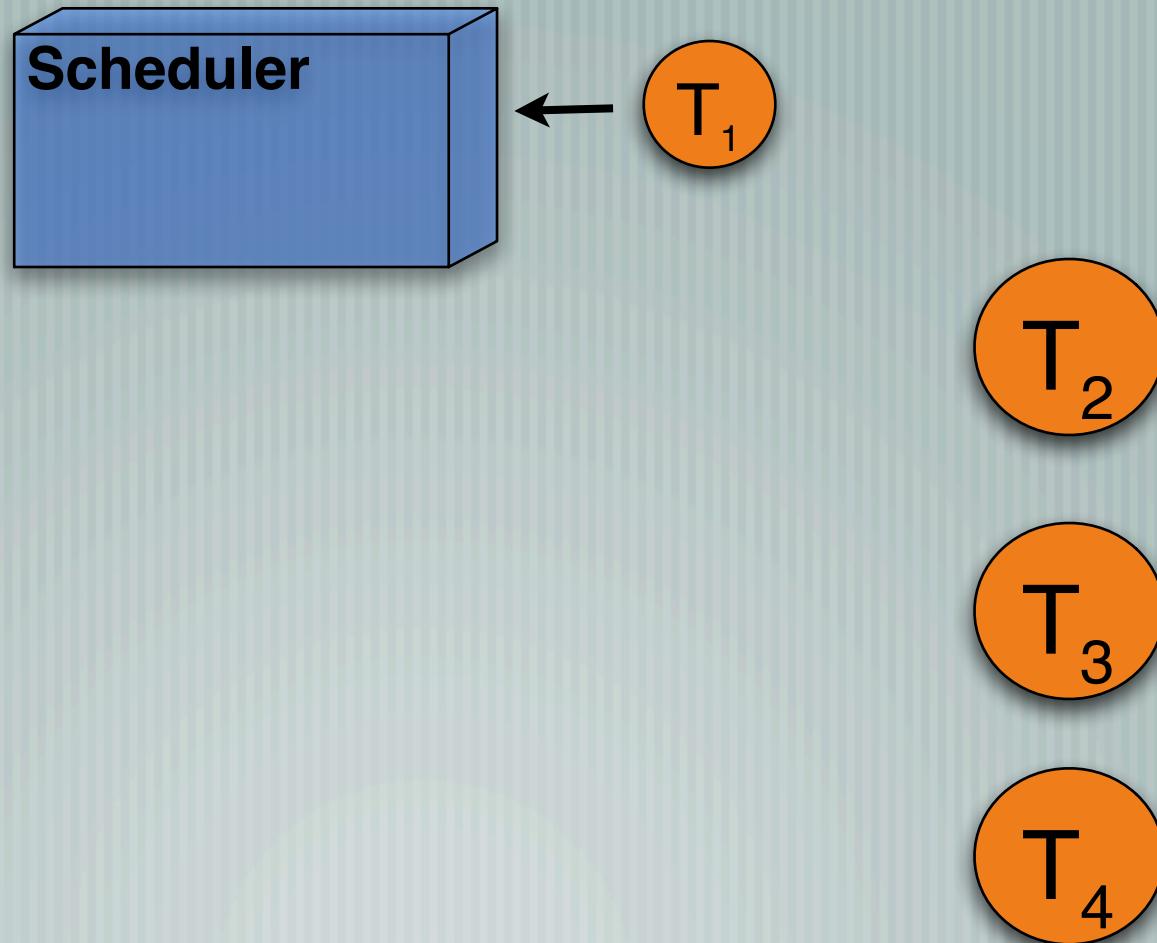
# TM Model : tokens

```
token T1  
{  
    if (...)  
    {...}  
}
```



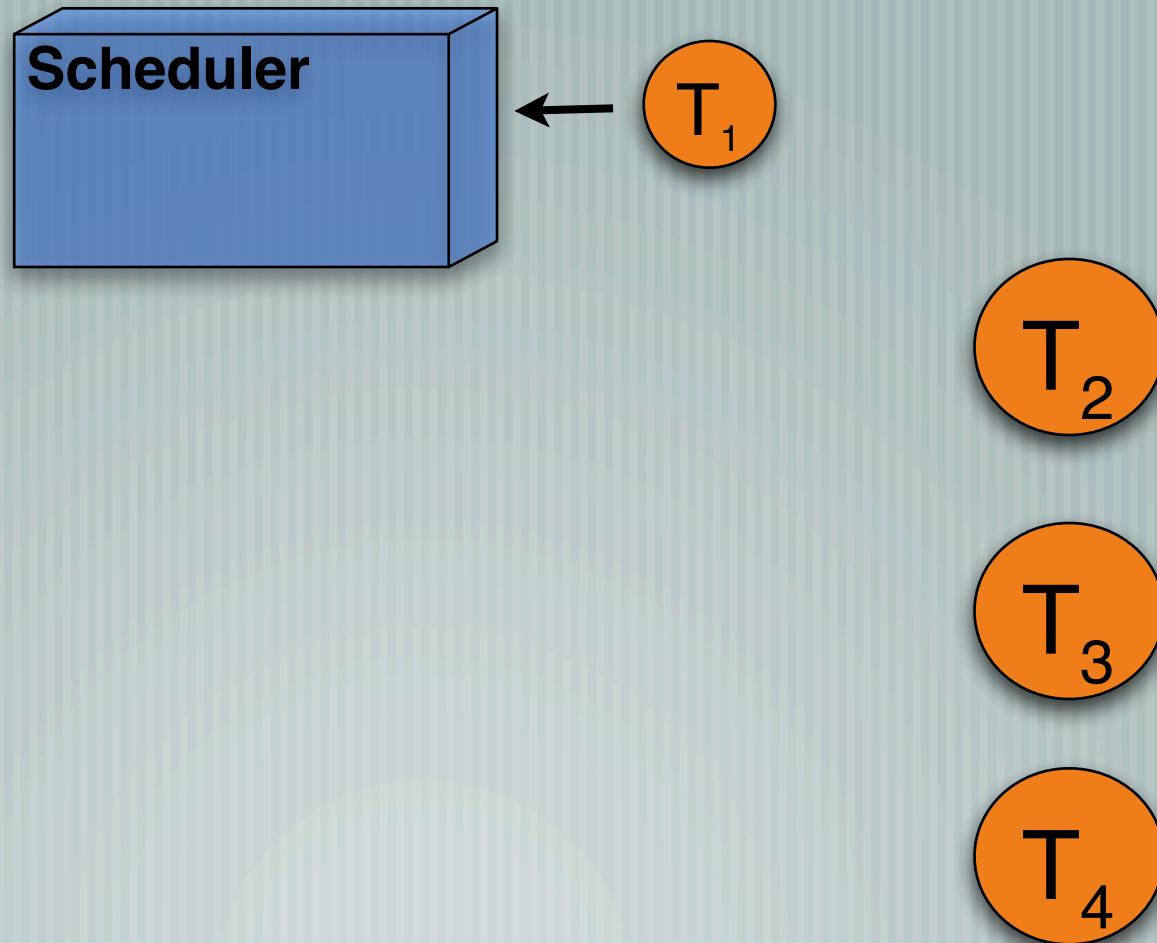
# TM Model : tokens

```
token T1  
{  
    if (...)  
    {...}  
}
```



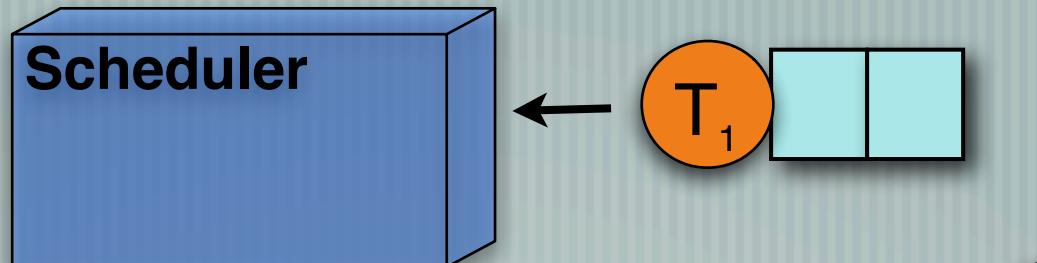
# TM Model : tokens

```
token T1(x,y)
{
    if (...)
    {...}
}
```

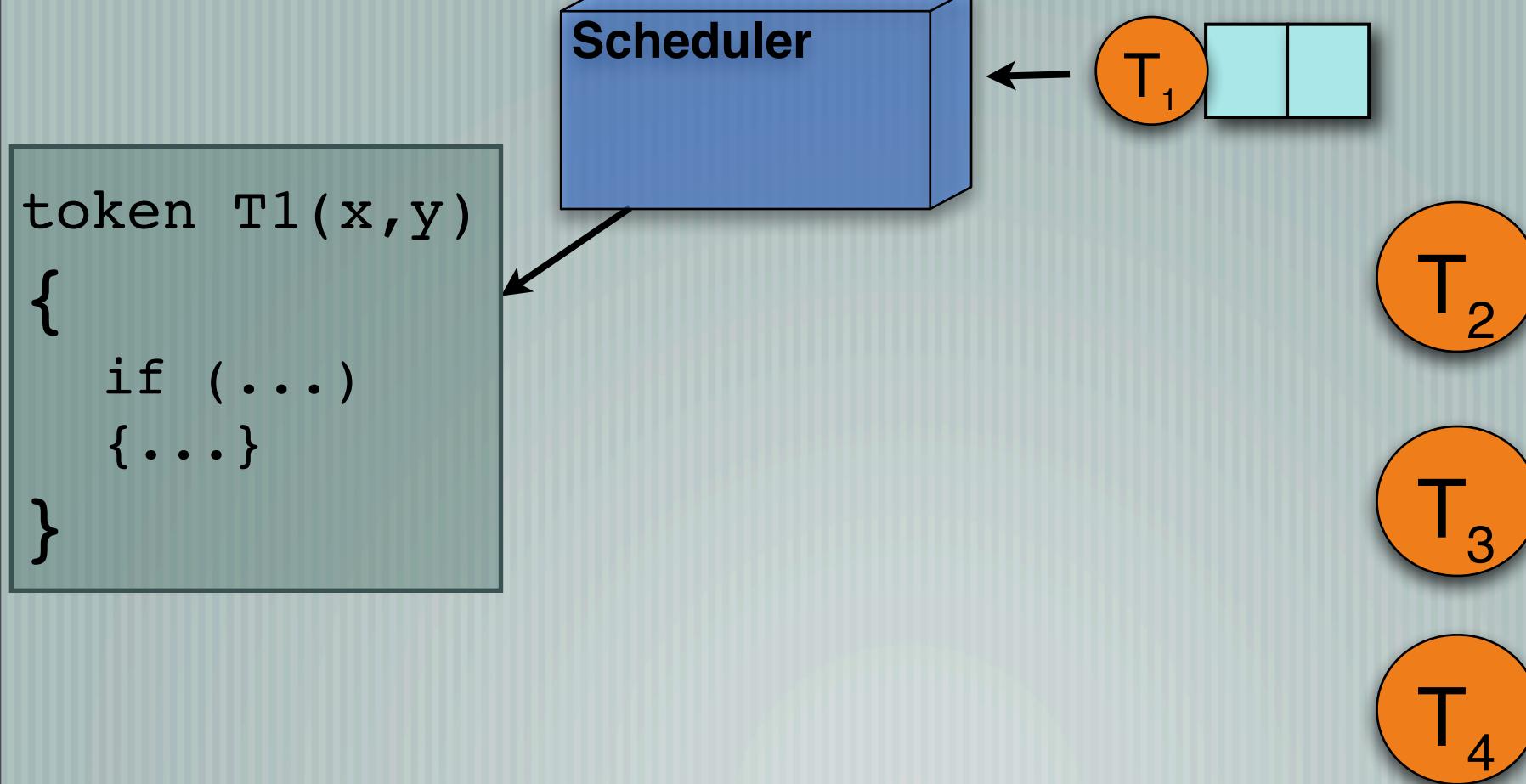


# TM Model : tokens

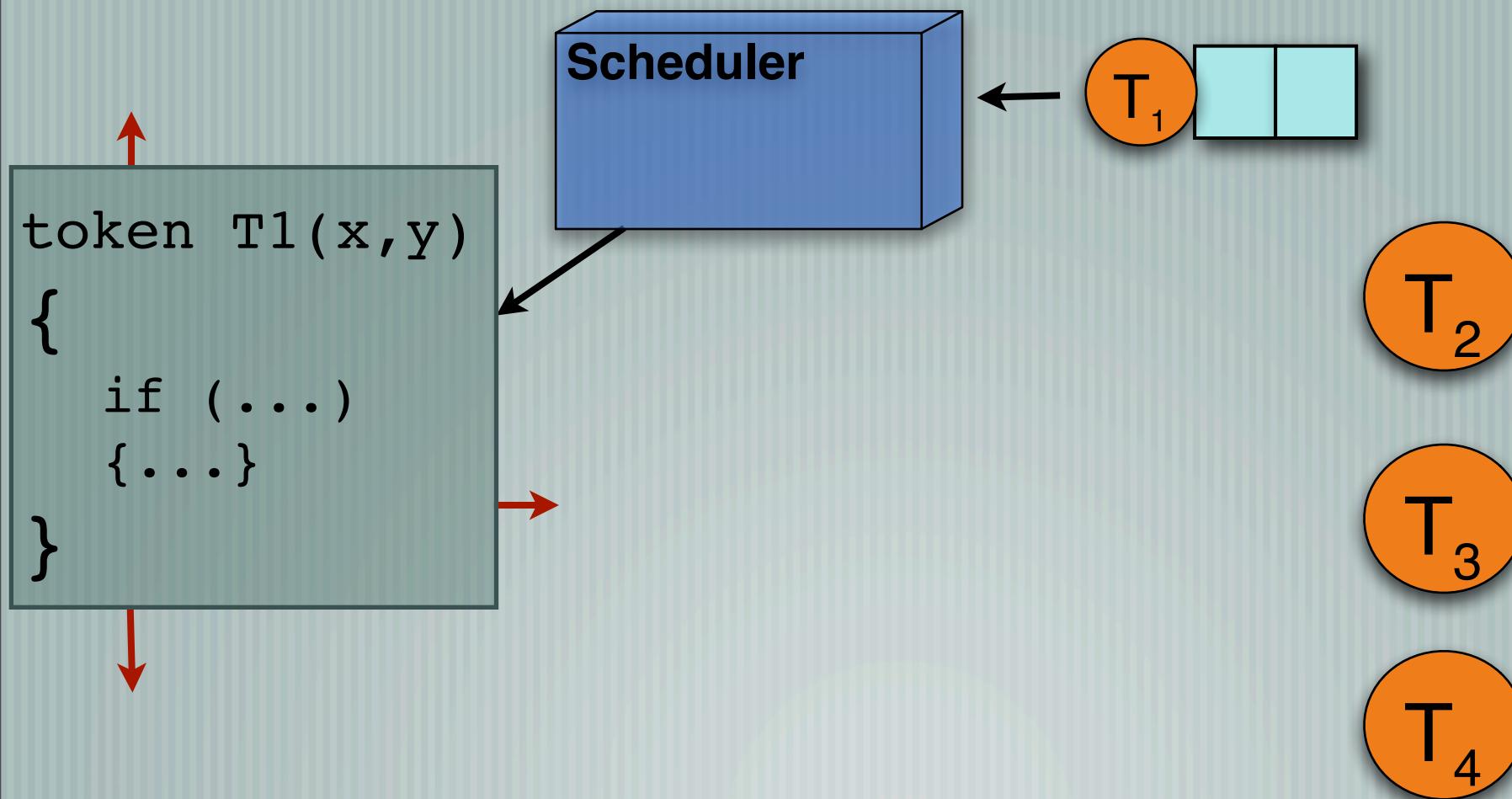
```
token T1(x,y)
{
    if (...)
    {...}
}
```



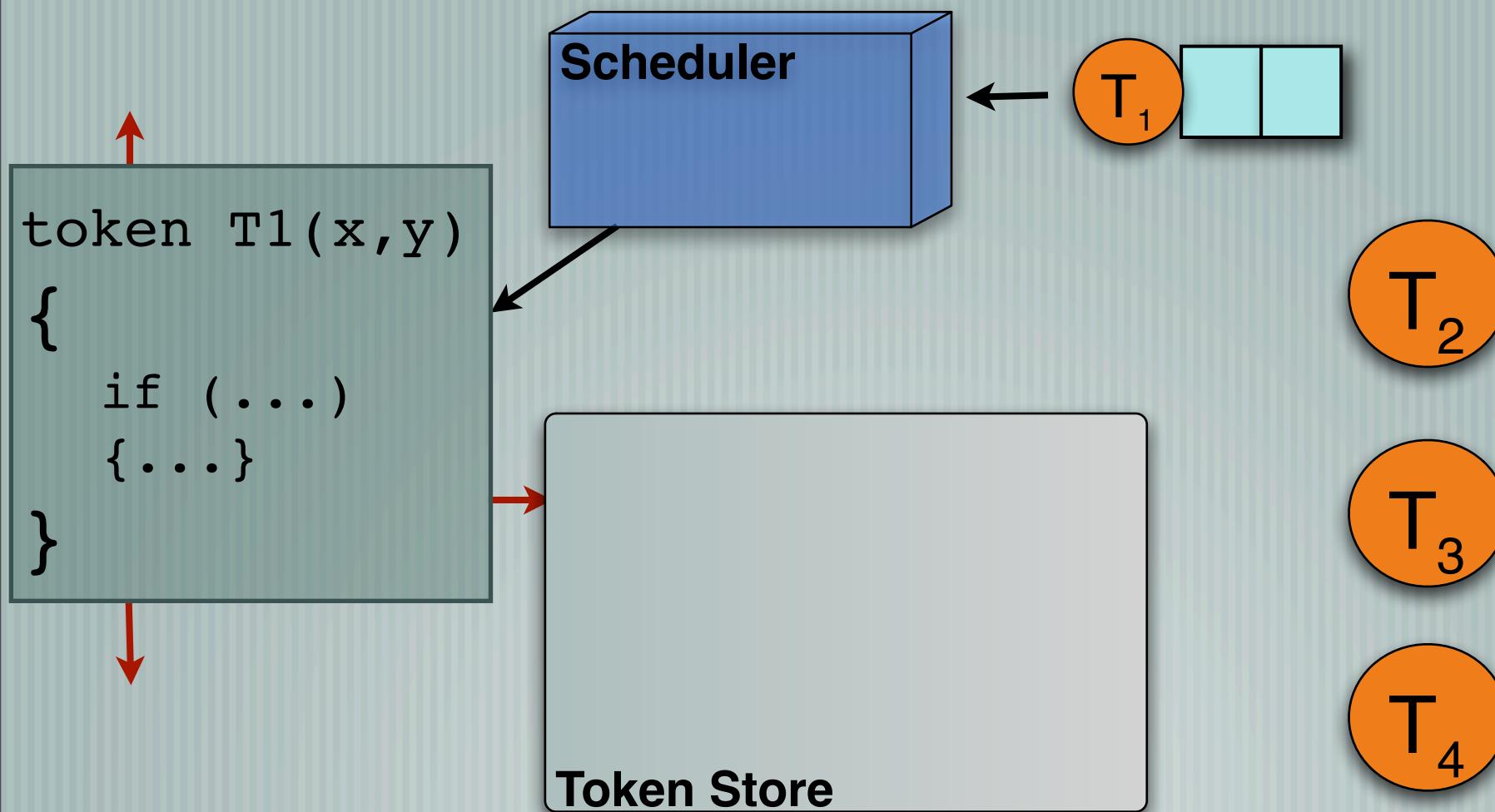
# TM Model : tokens



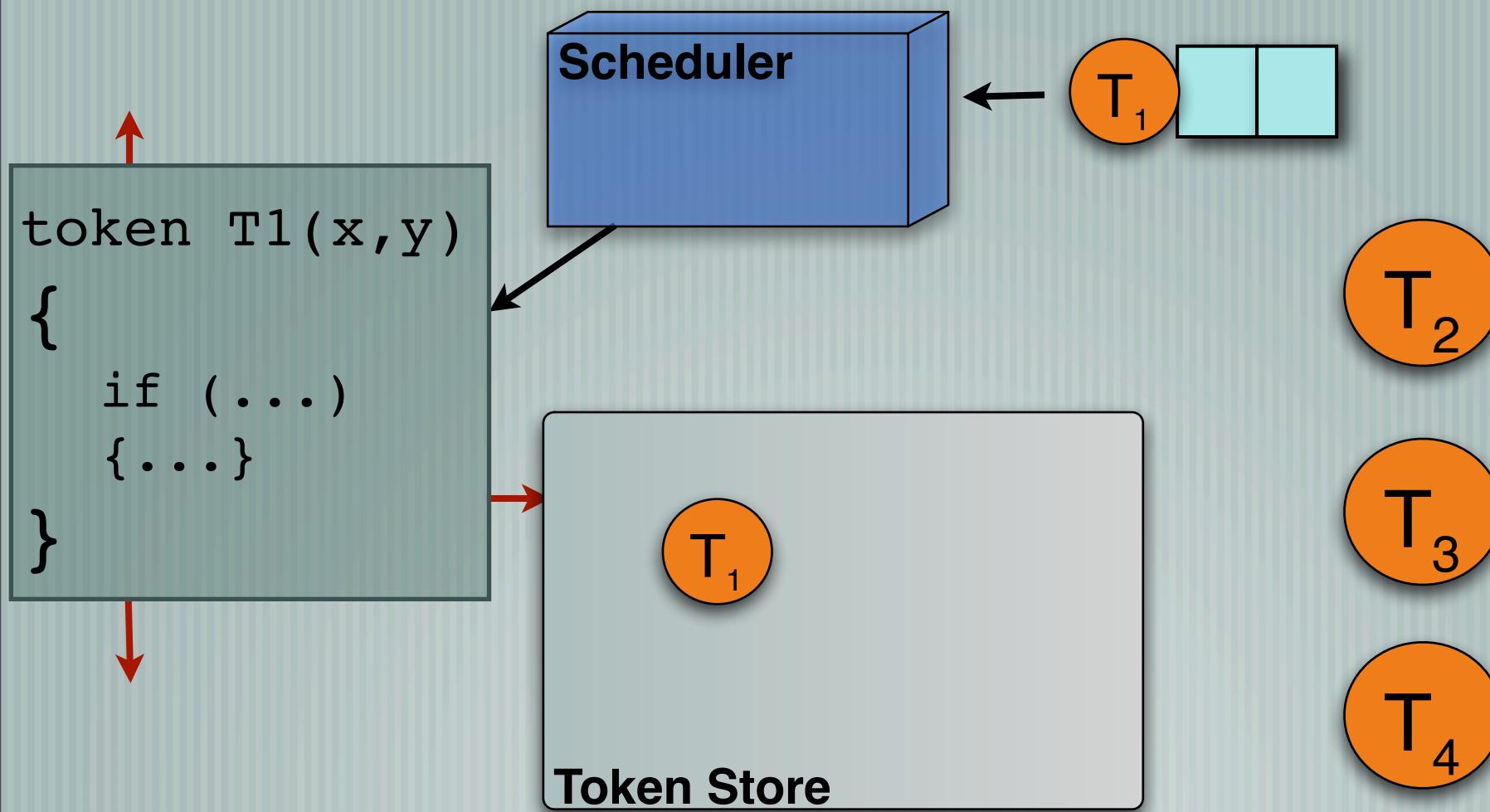
# TM Model : tokens



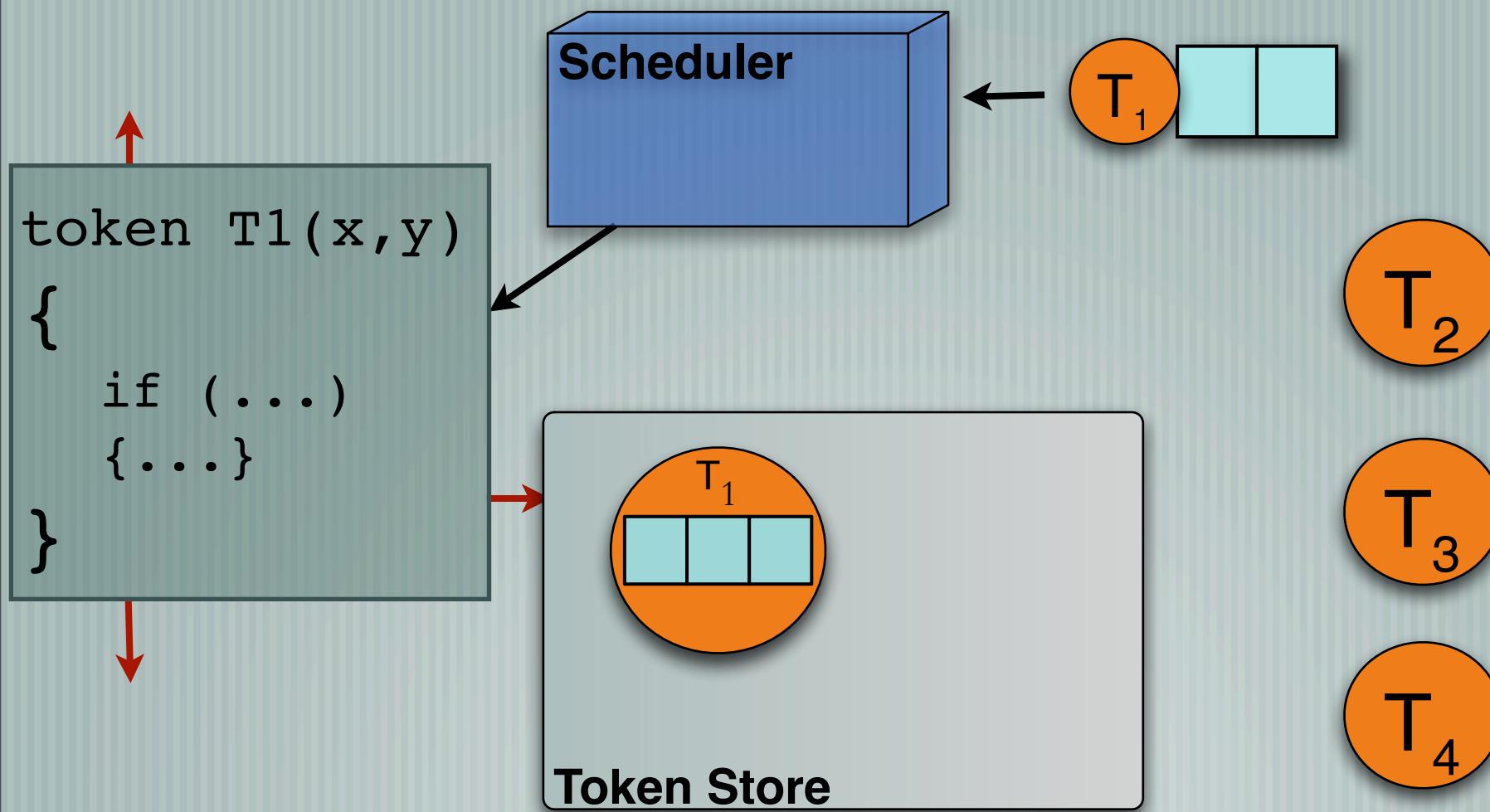
# TM Model : tokens



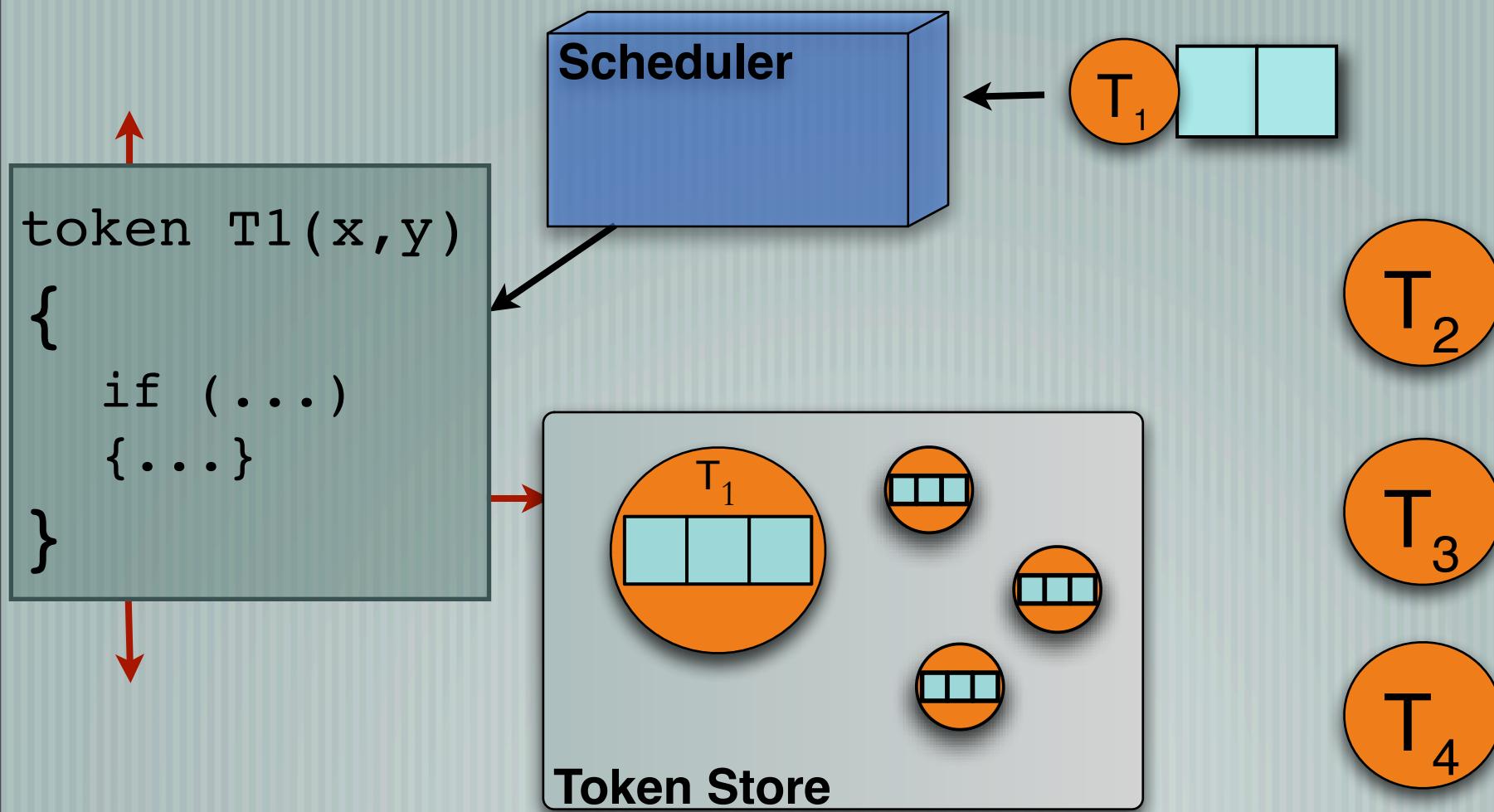
# TM Model : tokens



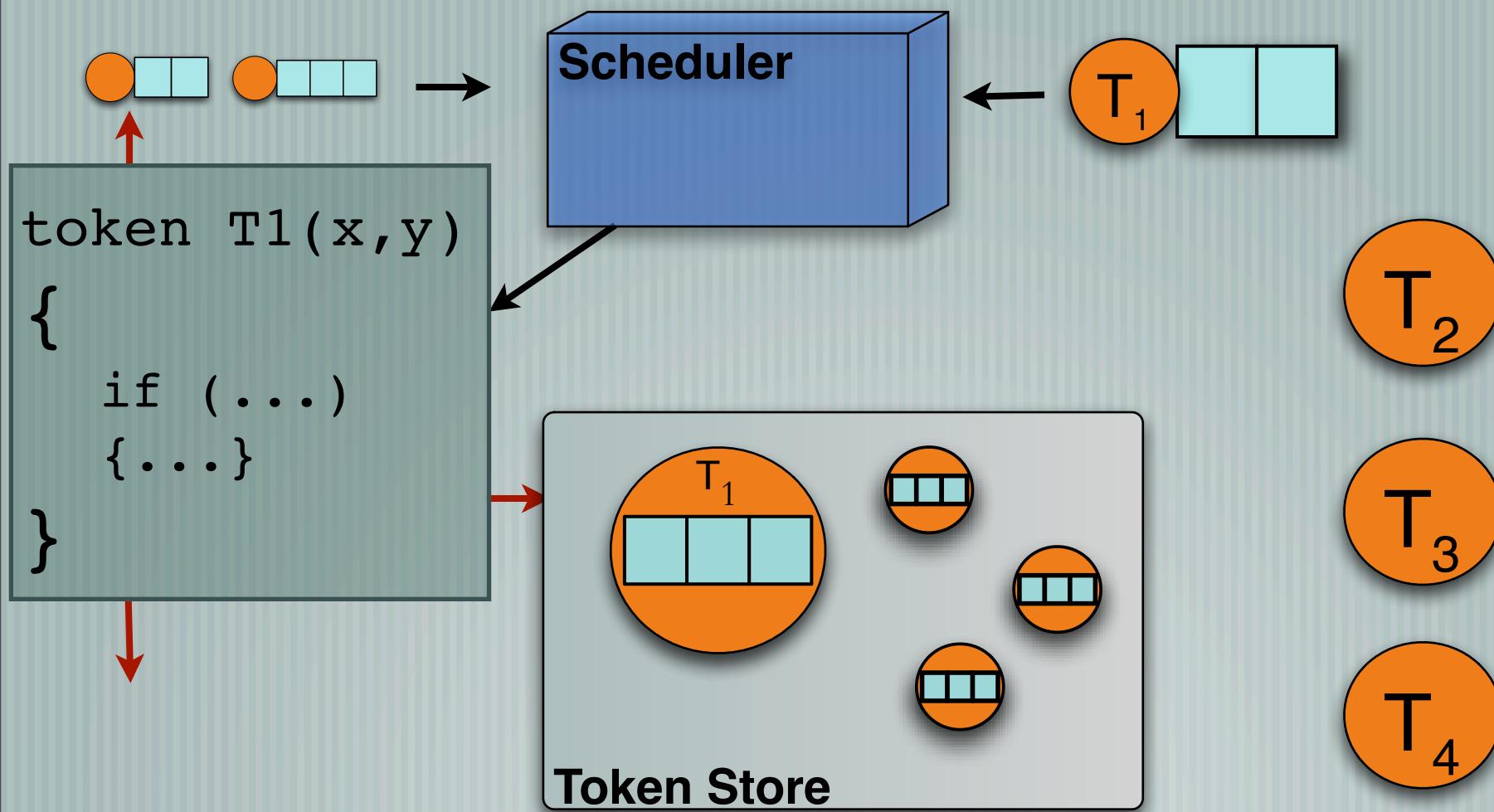
# TM Model : tokens



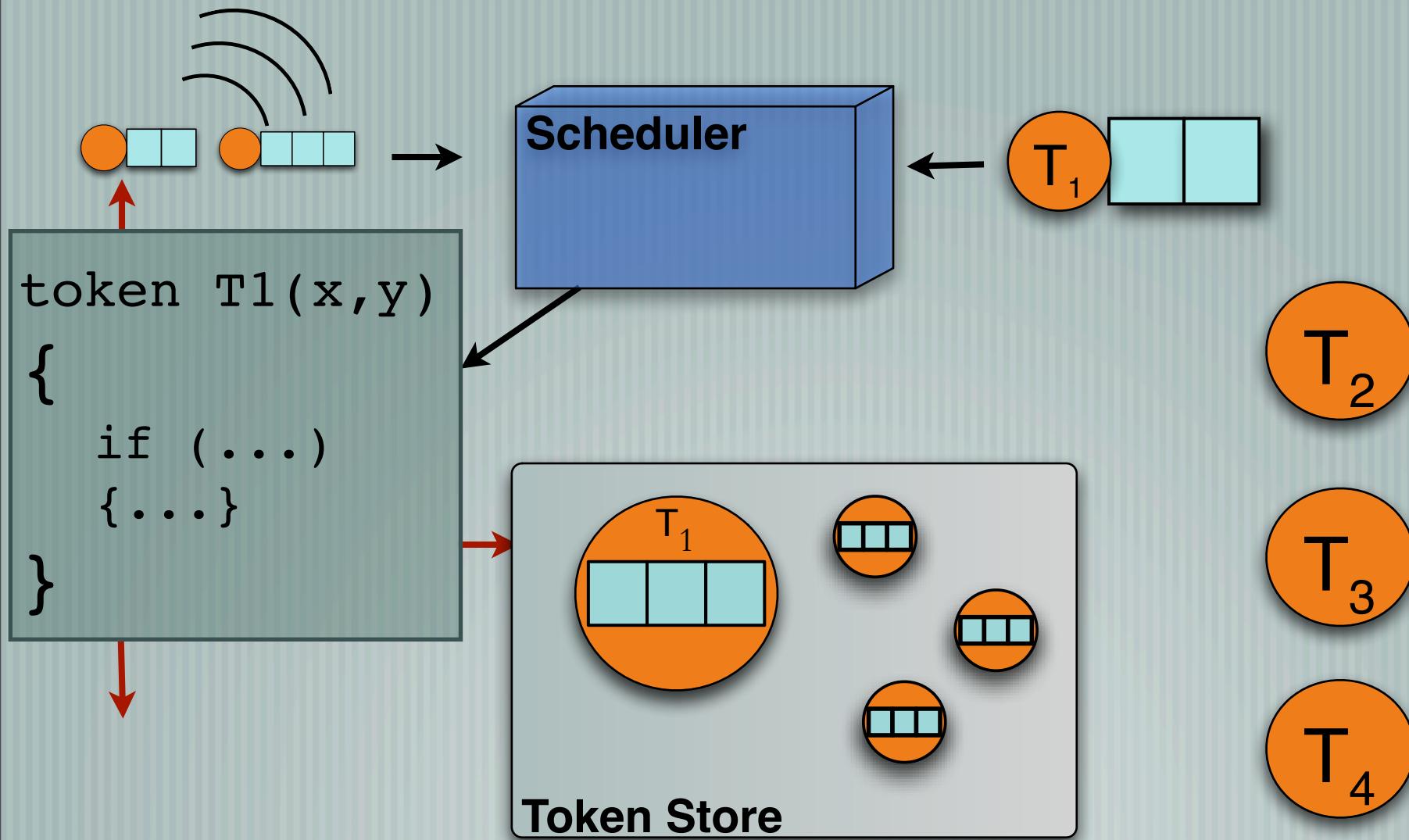
# TM Model : tokens



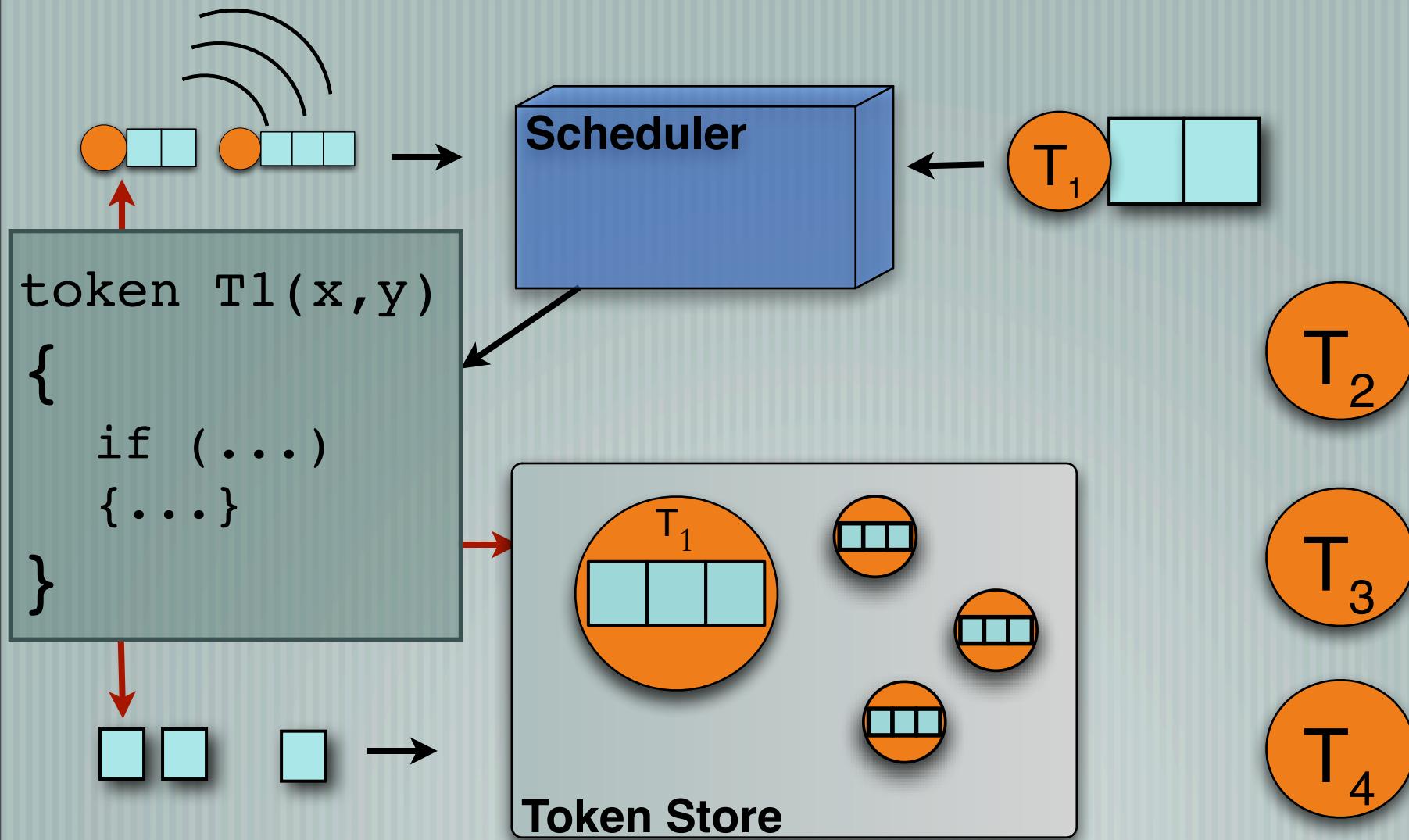
# TM Model : tokens



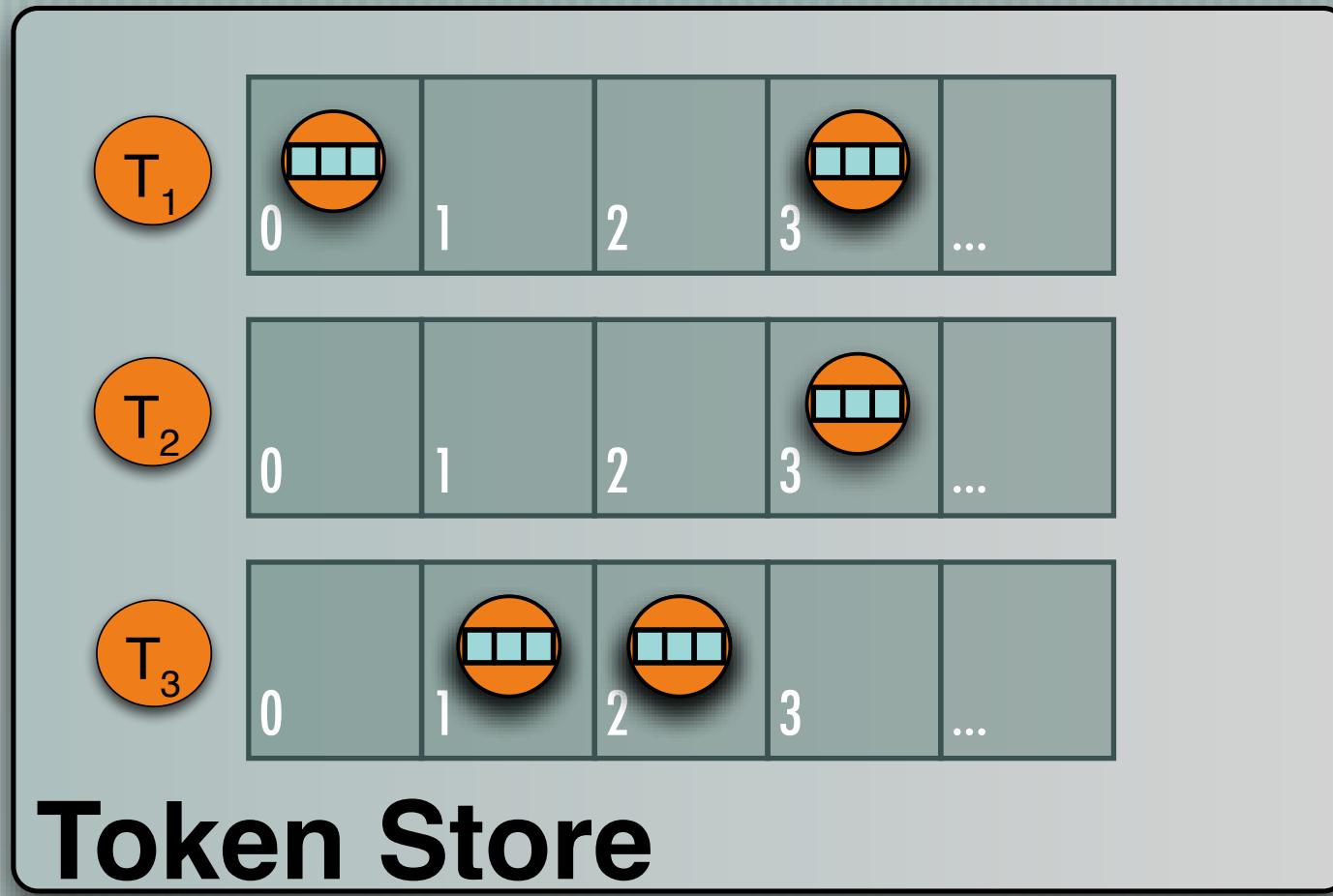
# TM Model : tokens



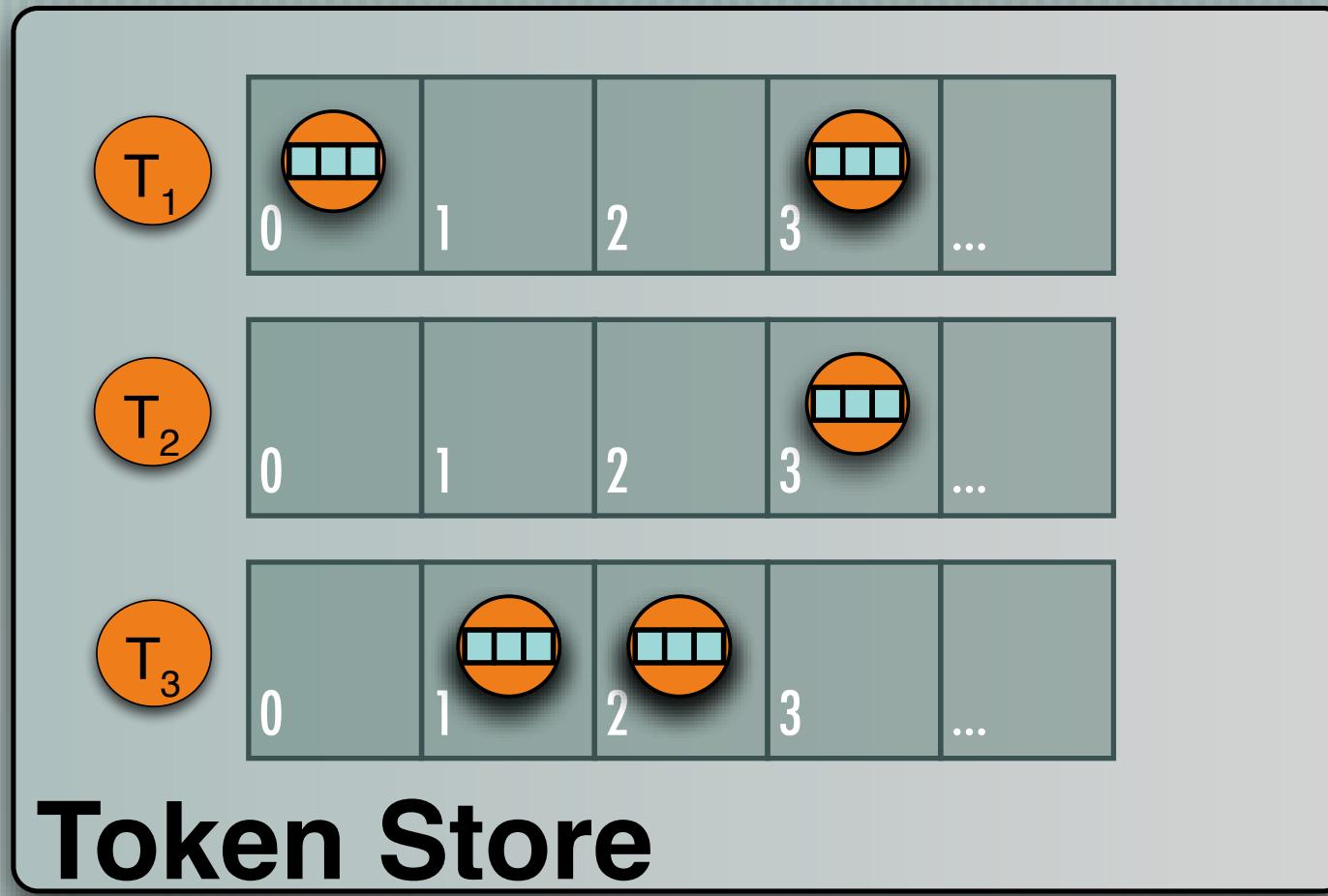
# TM Model : tokens



# TM Model : memory allocation



# TM Model : memory allocation



Consistent Inter-node virtual addresses:

$T_2[4]$

# Token Store replaces Stack

- `Red, Red[2], Red[7]`
- `call Red[1](x);`
- `(subcall Red[1](x)) + 8;`
- Uses implicit continuation objects.

# User code API

- call
- timed\_call
- bcast
- Plus! ○ subcall
- is\_scheduled
- deschedule
- is\_loaded
- evict

# Token-unified framework

- Concurrency: atomic token handlers
- Communication: token messages
  - both local and remote messaging
- Memory: token objects on heap
  - subtokens allow for dynamic allocation

```
token  
{  
    if (...)  
    {...}  
}
```

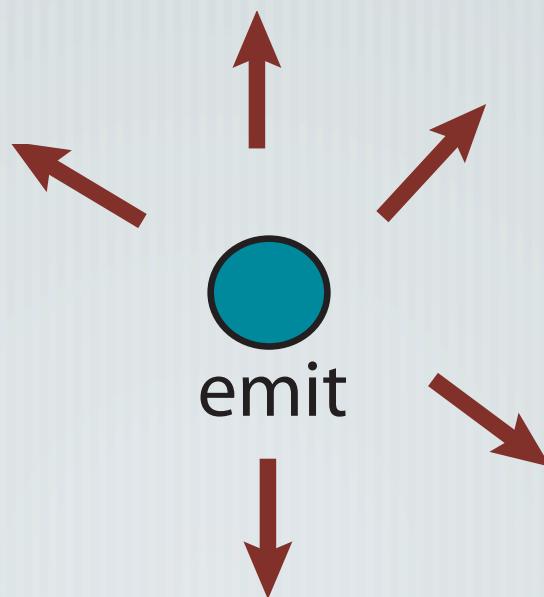


# Building on TML

# Gradients

# Gradients

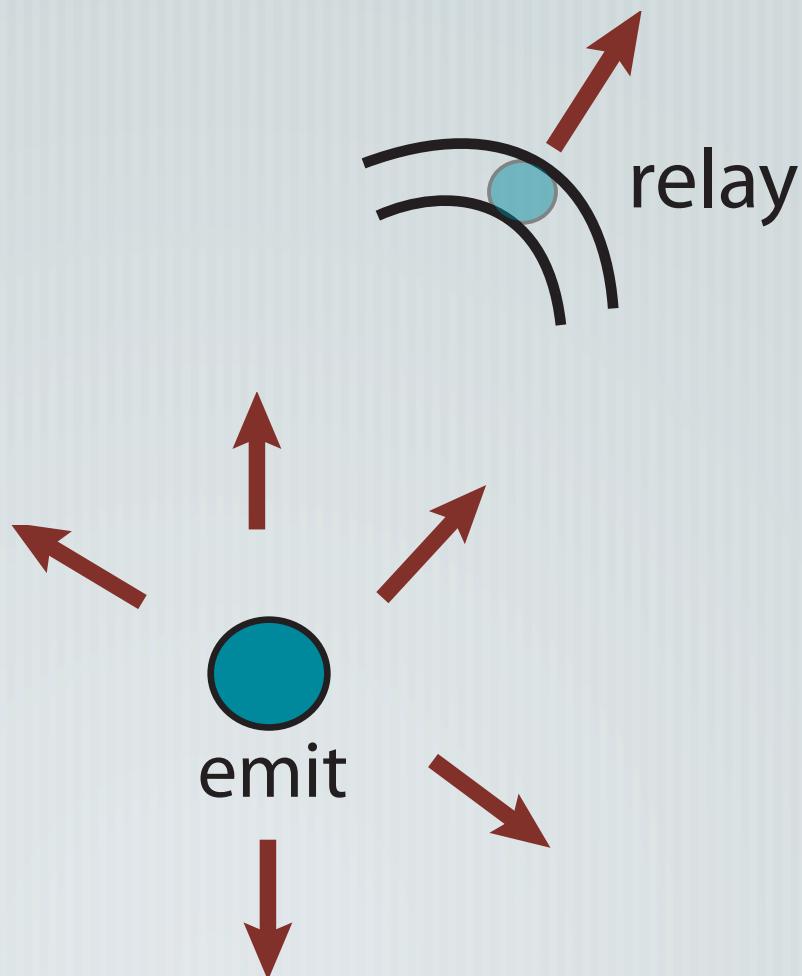
$\text{gemit}(T, v)$



# Gradients

$\text{gemit}(T, v)$

$\text{grelay}(T, v)$

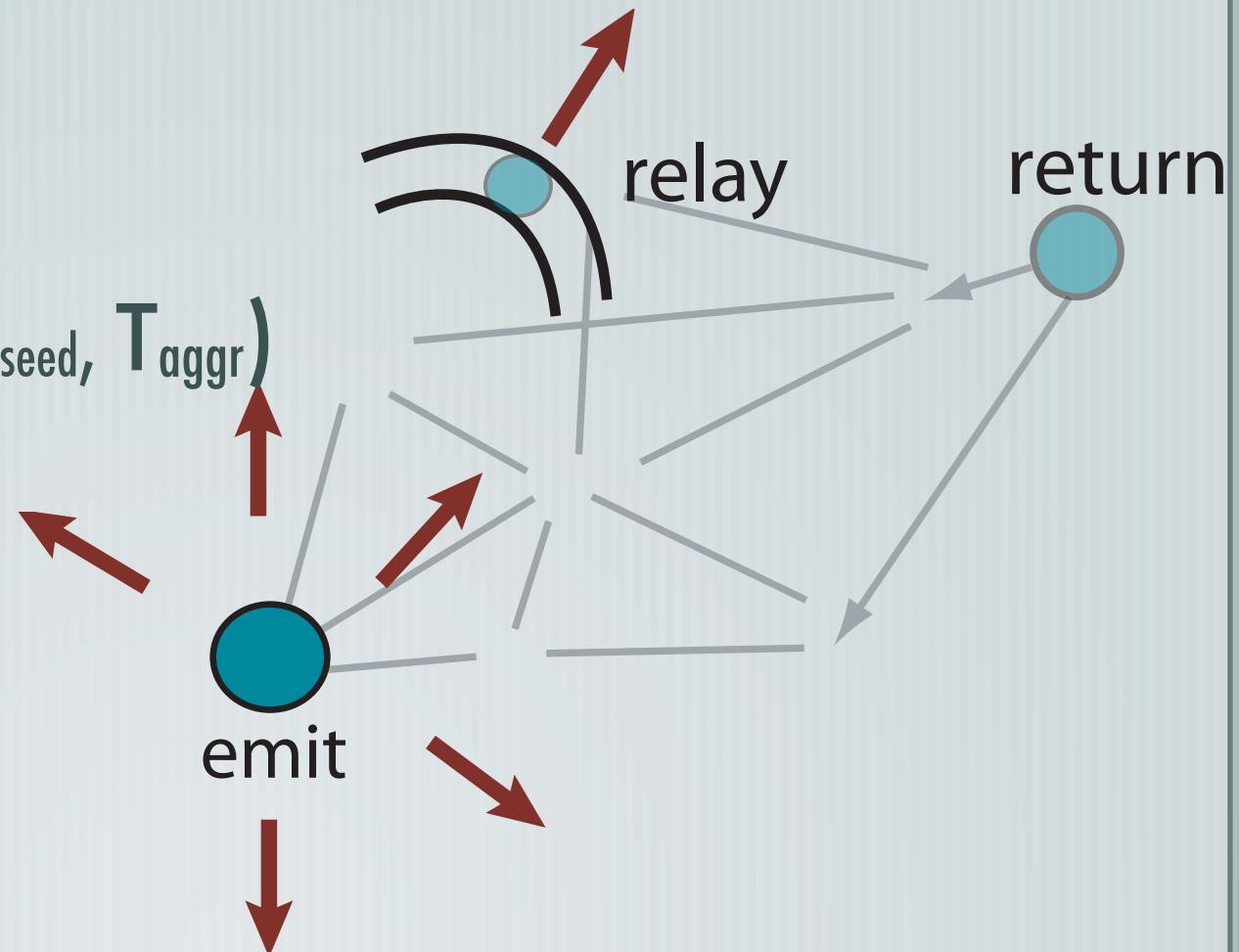


# Gradients

`gemit( $T, v$ )`

`grelay( $T, v$ )`

`greturn( $v, T_{to}, T_{via}, v_{seed}, T_{aggr}$ )`



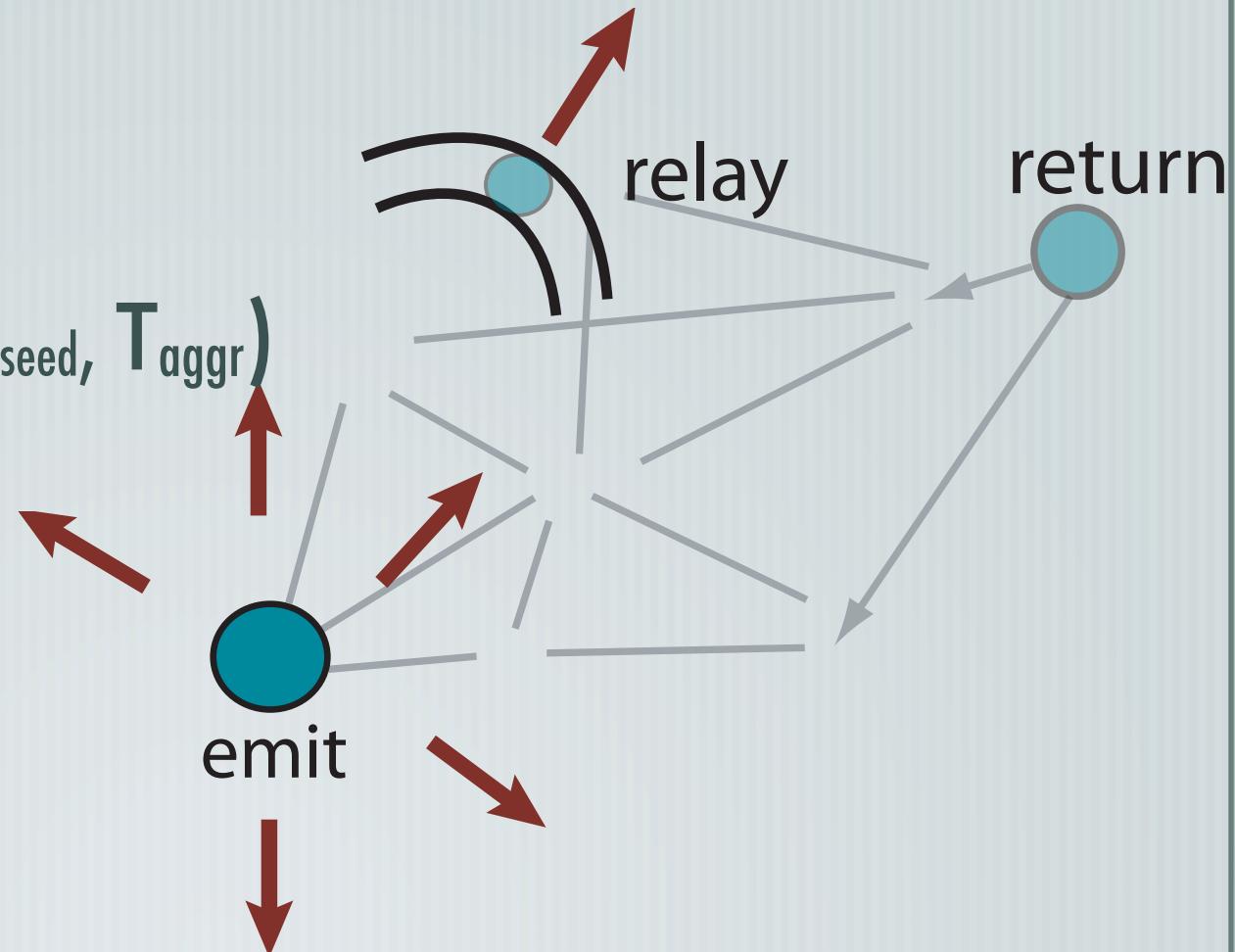
# Gradients

$\text{gemit}(T, v)$

$\text{grelay}(T, v)$

$\text{greturn}(v, T_{\text{to}}, T_{\text{via}}, v_{\text{seed}}, T_{\text{aggr}})$

Agnostic to routing  
and aggregation  
method



# A data gathering program

```
startup Gather;
base_startup SparkGlobal;

token SparkGlobal() {
    gemit GlobalTree();
    timed_schedule SparkGlobal(10000);
}

token GlobalTree() {
    grelay GlobalTree();
}

token Gather() {
    greturn(subcall sense_light(),
            BaseReceive,
            GlobalTree,
            NULL, NULL);
    -
    timed_schedule Gather(1000);
}
```

# Protocol Stacks and Language Towers

---

# Protocol Stacks and Language Towers

Bare Token Machines

---

# Protocol Stacks and Language Towers

Returning Subcalls

---

Bare Token Machines

---

# Protocol Stacks and Language Towers

subcall T(x) + 3

Returning Subcalls

---

Bare Token Machines

---

# Protocol Stacks and Language Towers

Gradients

---

Returning Subcalls

---

Bare Token Machines

---

# Protocol Stacks and Language Towers

Gradients

gemit(T,x...),  
grelay(...), greturn(...)

---

Returning Subcalls

---

Bare Token Machines

---

# Protocol Stacks and Language Towers

Gradients

---

Returning Subcalls

---

Bare Token Machines

---

# Protocol Stacks and Language Towers

Macros

---

Gradients

---

Returning Subcalls

---

Bare Token Machines

---

# Protocol Stacks and Language Towers

flood( $T$ )

Macros

---

Gradients

---

Returning Subcalls

---

Bare Token Machines

---

# Protocol Stacks and Language Towers

flood( $T$ )

Macros

elect-leader(...)

Gradients

Returning Subcalls

Bare Token Machines

# Protocol Stacks and Language Towers

Macros

---

Gradients

---

Returning Subcalls

---

Bare Token Machines

---

# Protocol Stacks and Language Towers

?

---

Macros

---

Gradients

---

Returning Subcalls

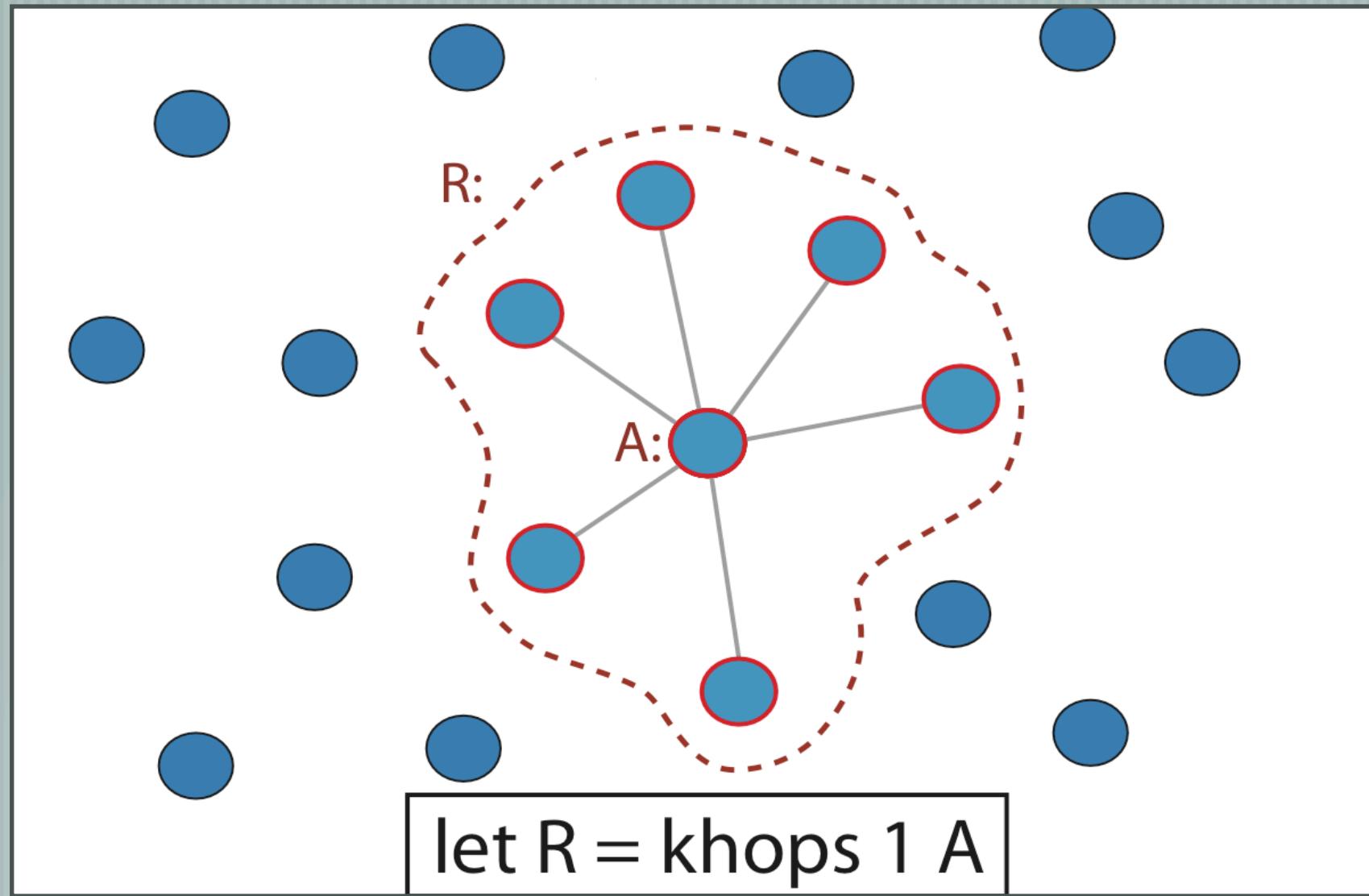
---

Bare Token Machines

# Compiling to TML

# High level language: Regiment example

# High level language: Regiment example



# Wins for Regiment + TML

- Token namespace serves for region coordination:
  - Region membership = holding a token
  - Gradients used for constructing and aggregating all continuous regions

# Future Work and Open Questions

- Dynamic loading
- Optimization: size
  - eliminating generated extra args
- Some dirty work.
- Implementation in real time OS?

# The End.

Ryan Newton, Arvind  
MIT CSAIL  
[{newton, arvind}@mit.edu](mailto:{newton, arvind}@mit.edu)

Matt Welsh  
Harvard University  
[mdw@eecs.harvard.edu](mailto:mdw@eecs.harvard.edu)

# TML Advantages

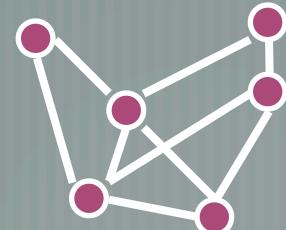
- Atomicity is a simple semantic model for code generators and transformers to target
- Lightweight - no GC, no threads, no blocking
- Action abortion => real-time potential

# Implementing Regiment

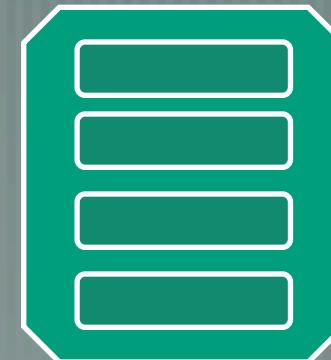
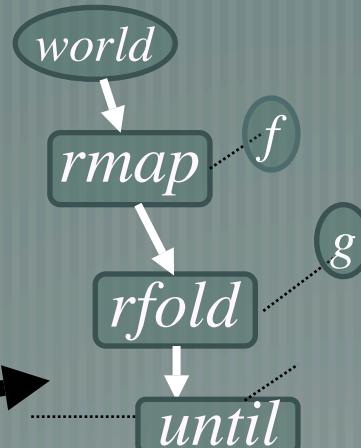
Stage 1:

Static  
Elaboration

2. Query  
'  
Circuit



5. Runtime



3. Token  
Machine

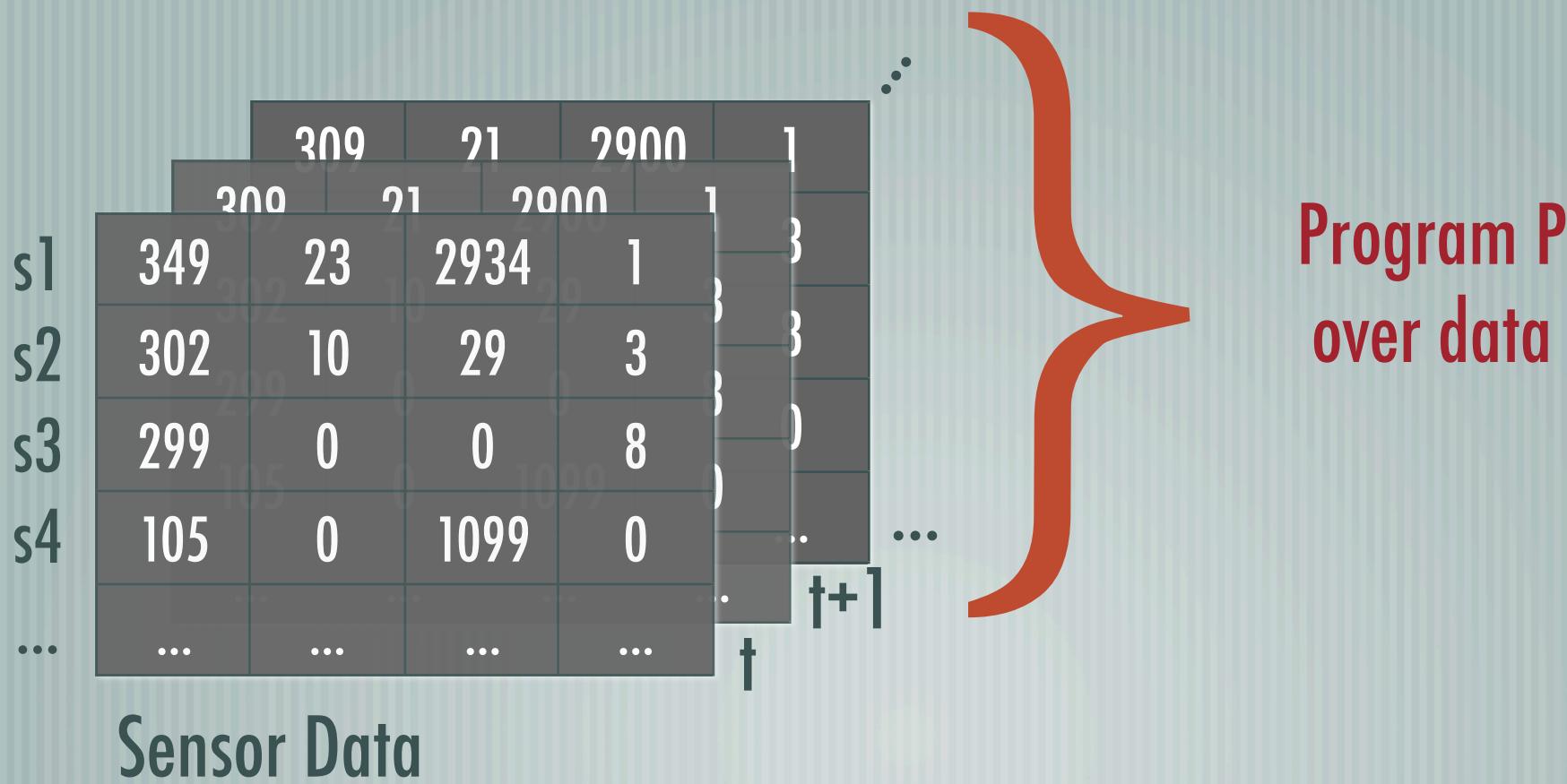
4. TinyOS  
program

# Motivation: macroprogramming

	309	21	2900	1	3	3	3	0	0	0	...	t+1
s1	349	23	2934	1	3	3	3	0	0	0	...	t
s2	302	10	29	3	3	3	3	0	0	0	...	t
s3	299	0	0	8	0	0	0	0	0	0	...	t
s4	105	0	1099	0	0	0	0	0	0	0	...	t
...	...	...	...	...	...	...	...	...	...	...	...	...

Sensor Data

# Motivation: macroprogramming



# Motivation: macroprogramming

“Full” access

A table representing sensor data. The columns are labeled with values: 309, 21, 2900, and 1. The rows are labeled s1, s2, s3, s4, and ... The table has a grid pattern with many overlapping values. A vertical line labeled t marks the current time step, and a vertical line labeled t+1 marks the next time step. A red curly brace on the right side of the table covers the column labeled 1, which is labeled 'Program P over data'.

	309	21	2900	1
	300	21	2000	1
s1	349	23	2934	1
s2	302	10	29	3
s3	299	0	0	8
s4	105	0	1099	0
...	...	...	...	...

Sensor Data

Program P  
over data

# Motivation: macroprogramming

## 1. RESTRICT P

“Full” access

	309	21	2900	1	3	3	3	3	...
s1	349	23	2934	1	3	3	3	3	...
s2	302	10	29	3	3	3	3	3	...
s3	299	0	0	8	0	0	0	0	...
s4	105	0	1099	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...

Sensor Data



Program P  
over data

# Motivation: macroprogramming

## 1. RESTRICT P

“Full” access

	309	21	2900	1	1	3	3
s1	349	23	2934	1	1	3	3
s2	302	10	29	3	3	0	0
s3	299	0	0	8	0	0	0
s4	105	0	1099	0	0	...	...
...	...	...	...	...	...	...	...

Sensor Data

Program P  
over data

## 2. PUSH P IN-NETWORK

